< Frontend Interview />

Intensive Course

Evgenii Ray

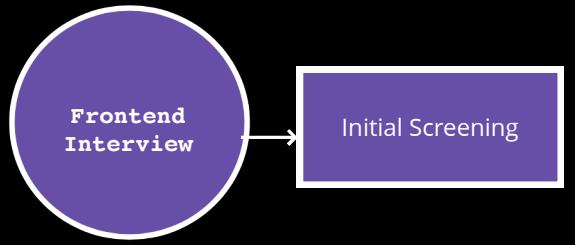
Introduction

- Who am I
- Goal of this course

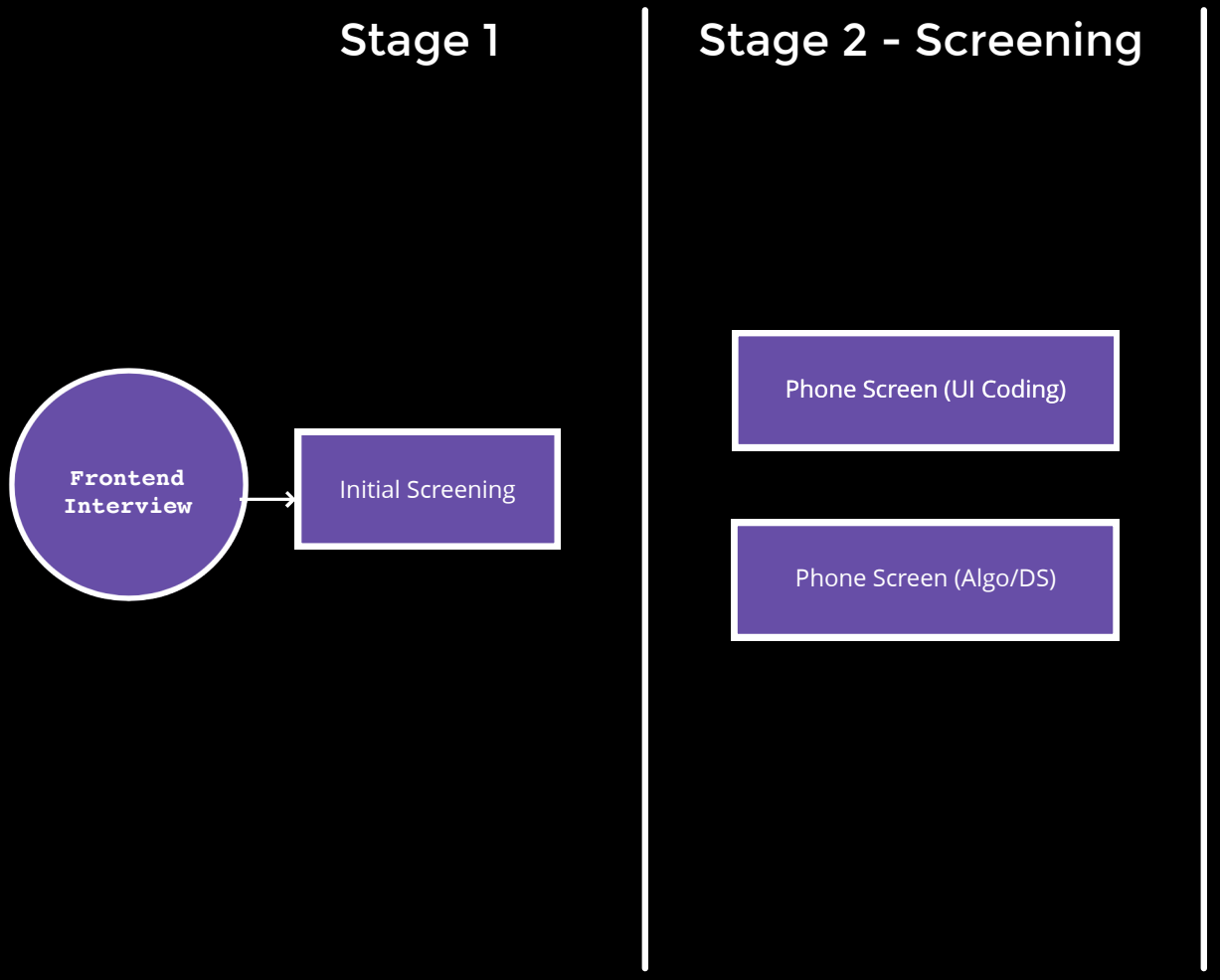
What this course is about

Typical UI Interview Process

Stage 1

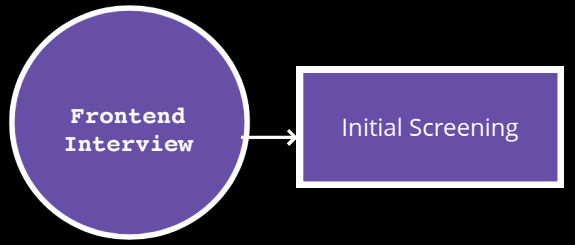


Typical UI Interview Process



Typical UI Interview Process

Stage 1



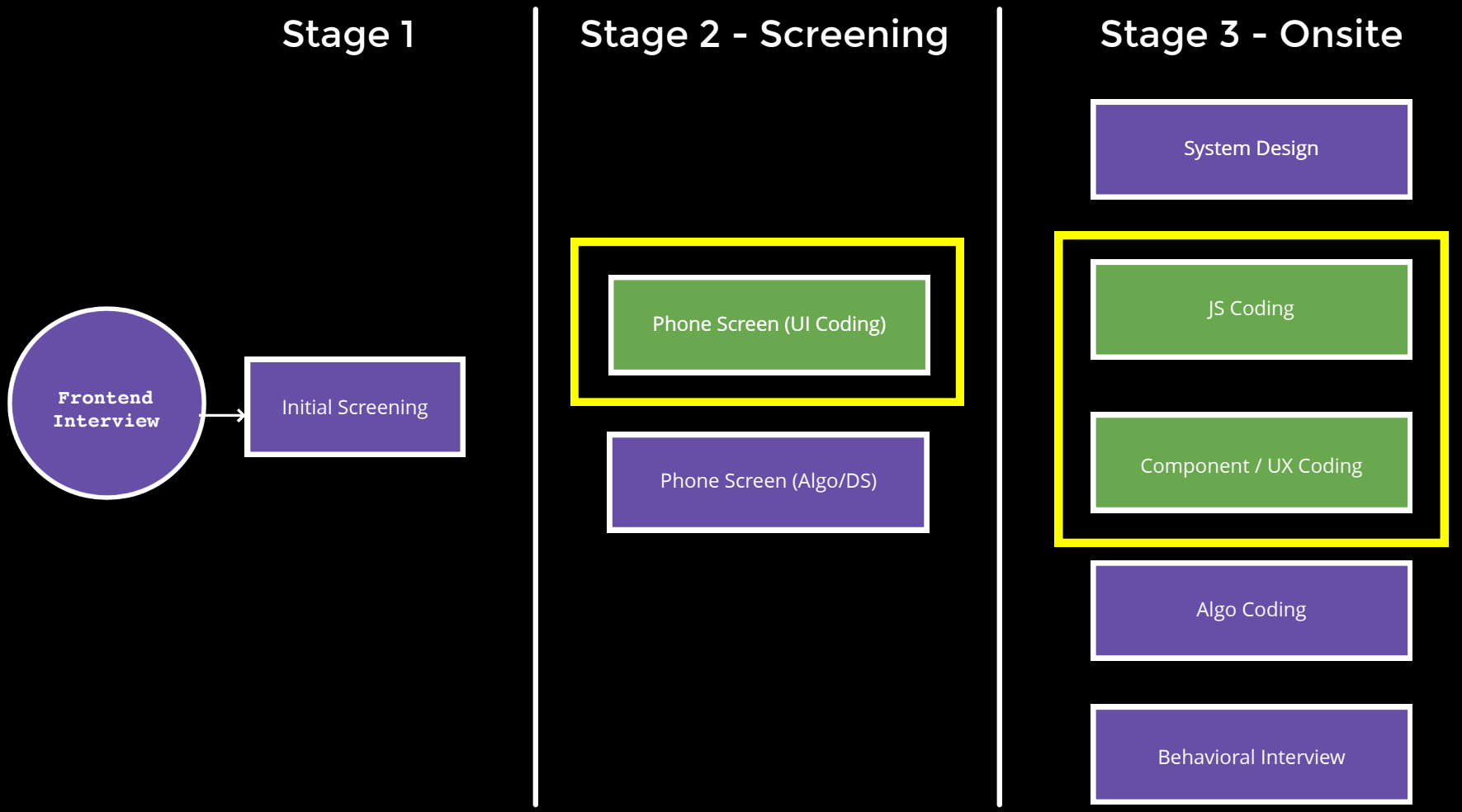
Stage 2 - Screening

- Phone Screen (UI Coding)
- Phone Screen (Algo/DS)

Stage 3 - Onsite

- System Design
- JS Coding
- Component / UX Coding
- Algo Coding
- Behavioral Interview

Typical UI Interview Process



Course Plan

1. Vanila / JS Coding

- Detect Type
- Debounce
- Throttle
- ES5 Inheritance
- Deep Equals
- Deep Clone
- Stringify
- Custom Promise
- Tree Select

Course Plan

1. Vanila / JS Coding

- Detect Type
- Debounce
- Throttle
- ES5 Inheritance
- Deep Equals
- Deep Clone
- Stringify
- Custom Promise
- Tree Select

2. Basic Components

- Accordion
- Star Rating
- Tabs
- Tooltip
- Dialog
- Table
- Reddit Thread
- Gallery
- Nested Checkboxes
- Toast

Course Plan

1. Vanila / JS Coding

- Detect Type
- Debounce
- Throttle
- ES5 Inheritance
- Deep Equals
- Deep Clone
- Stringify
- Custom Promise
- Tree Select

2. Basic Components

- Accordion
- Star Rating
- Tabs
- Tooltip
- Dialog
- Table
- Reddit Thread
- Gallery
- Nested Checkboxes
- Toast

3. Advanced components

- Calculator
- Square Game
- Typeahead
- Progress Bar
- File Upload
- Portfolio Visualizer
- Markdown Editor

Course Plan

1. Vanila / JS Coding

- Detect Type
- Debounce
- Throttle
- ES5 Inheritance
- Deep Equals
- Deep Clone
- Stringify
- Custom Promise
- Tree Select

2. Basic Components

- Accordion
- Star Rating
- Tabs
- Tooltip
- Dialog
- Table
- Reddit Thread
- Gallery
- Nested Checkboxes
- Toast

3. Advanced components

- Calculator
- Square Game
- Typeahead
- Progress Bar
- File Upload
- Portfolio Visualizer
- Markdown Editor

4. Staff+ Coding Problems

- GPT Chat Interface (Streaming)
- Google Sheets
- - Parser & Tokenizer
  - Topological Sorting
  - Table Engine
  - UX / UI

Course Plan

1. Vanila / JS Coding

- Detect Type
- Debounce
- Throttle
- ES5 Inheritance
- Deep Equals
- Deep Clone
- Stringify
- Custom Promise
- Tree Select

2. Basic Components

- Accordion
- Star Rating
- Tabs
- Tooltip
- Dialog
- Table
- Reddit Thread
- Gallery
- Nested Checkboxes
- Toast

3. Advanced components

- Calculator
- Square Game
- Typeahead
- Progress Bar
- File Upload
- Portfolio Visualizer
- Markdown Editor

4. Staff+ Coding Problems

- GPT Chat Interface (Streaming)
- Google Sheets
- - Parser & Tokenizer
  - Topological Sorting
  - Table Engine
  - UX / UI

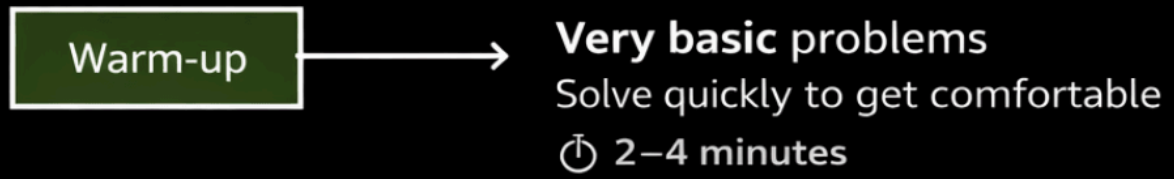
5. Typescript Challenges

- Basics
- Mapped Types
- Conditional Types
- Infer
- Template Literals
- Recursion
- Distributive
- Advanced Patterns
- Expert Techniques

Difficulty of the questions

NOTE: The problems in this workshop are intentionally slightly harder than real interview questions.

Difficulty of the questions

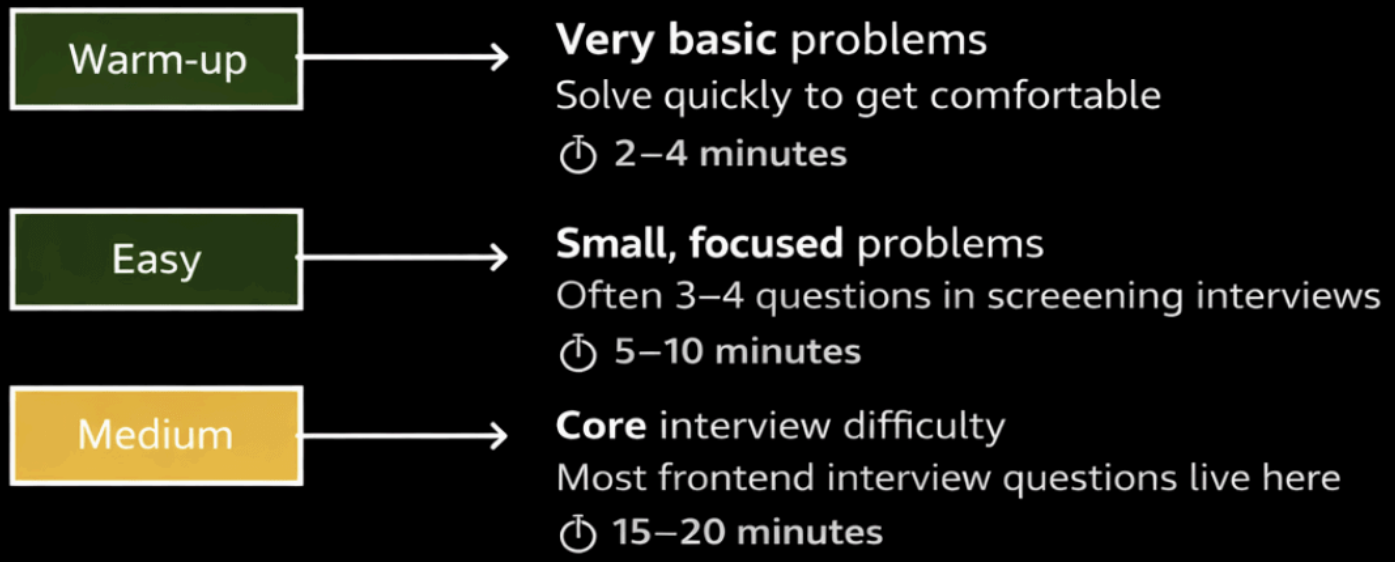


Difficulty of the questions

Warm-up → **Very basic** problems
Solve quickly to get comfortable
⏱ 2–4 minutes

Easy → **Small, focused** problems
Often 3–4 questions in screeening interviews
⏱ 5–10 minutes

Difficulty of the questions



Difficulty of the questions

Warm-up →
Very basic problems
Solve quickly to get comfortable
⏱ **2–4 minutes**

Easy →
Small, focused problems
Often 3–4 questions in screeening interviews
⏱ **5–10 minutes**

Medium →
Core interview difficulty
Most frontend interview questions live here
⏱ **15–20 minutes**

Hard →
Complex problems requiring practice
Browser APIs, edge cases, real-world constraints
⏱ **45–60 minutes** (Workshop target: 20–25 minutes)

Difficulty of the questions

Warm-up →
Very basic problems
Solve quickly to get comfortable
⏱ 2–4 minutes

Easy →
Small, focused problems
Often 3–4 questions in screeening interviews
⏱ 5–10 minutes

Medium →
Core interview difficulty
Most frontend interview questions live here
⏱ 15–20 minutes

Hard →
Complex problems requiring practice
Browser APIs, edge cases, real-world constraints
⏱ 45–60 minutes (Workshop target: 20–25 minutes)

EXTREME! →
End-to-end product-style problems
Build a feature from scratch
Architecture & structure matters most
⏱ 1–2 hours
🎯 Typical for **Staff-level interviews**

Finding solutions

https://github.com/EvgeniiRay/preparing-for-ui-interview-v2

How to follow the course

IDE Setup: → **VSCode** or **WebStorm**

Any other IDE you're comfortable with

How to follow the course

IDE Setup: → **VSCode** or **WebStorm**
Any other IDE you're comfortable with

Disable AI assistants: → **Disable AI assistants and autocompletions →**
You're here to **practice and learn**, not to get solutions from AI

How to follow the course

IDE Setup: → **VSCode** or **WebStorm**
Any other IDE you're comfortable with

Disable AI assistants: → **Disable AI assistants and autocompletions** →
You're here to **practice and learn**, not to get solutions from AI

Mistakes are inevitable: → During the **course**, we'll most likely need to **debug the code**

How to follow the course

IDE Setup: → **VSCode** or **WebStorm**
Any other IDE you're comfortable with

Disable AI assistants: → **Disable AI assistants and autocompletions →**
You're here to **practice and learn**, not to get solutions from AI

Mistakes are inevitable: → During the **course**, we'll most likely need to **debug the code**

Only Necessary CSS: → We'll not waste precious time on styling components
Focus is on **logic and structure**

Github Repo Structure

Each component (e.g. 01-accordion) acts as a **self-contained** unit located under

src/problems/

Student Starter Files:

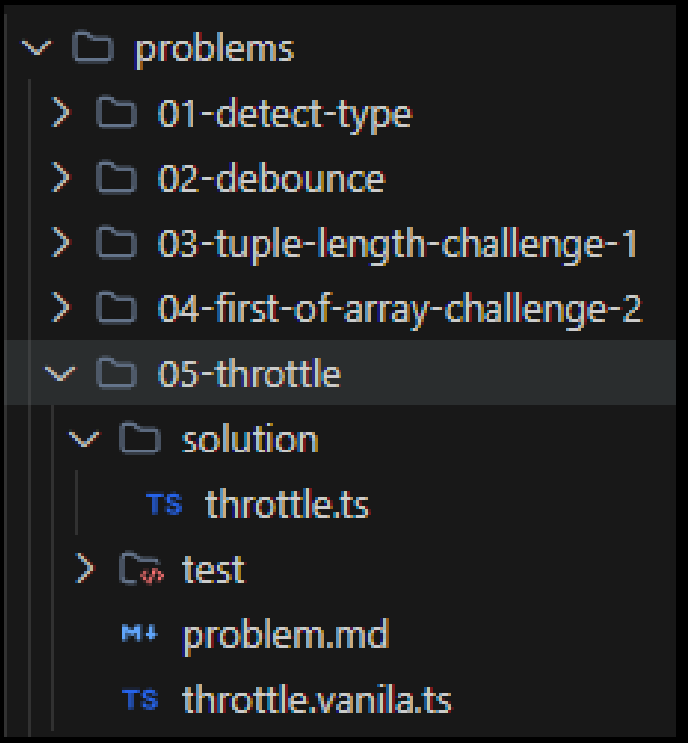
- **<name>**.react.tsx (React) or **<name>.vanila.ts** (Vanilla TS)
- **<name>.module.css** (Styles)

These are the empty files you will work on.

Reference Solution:

- **/solution/** subfolder contains the **full** working implementation.

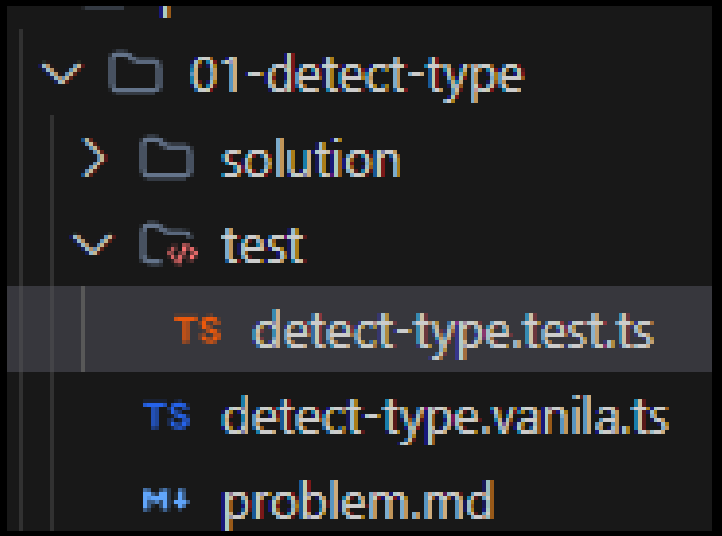
Check this only if you are stuck or for comparison after finishing.



Github Repo Structure: Running tests

```
bun test src/problems/01-detect-type/test/detect-type.test.ts
```

```
22 tests failed:
✗ Student Solution > detectType > null and undefined > should return "null" for null
✗ Student Solution > detectType > null and undefined > should return "undefined" for undefined
✗ Student Solution > detectType > primitives > should return "number" for numbers
✗ Student Solution > detectType > primitives > should return "string" for strings
✗ Student Solution > detectType > primitives > should return "boolean" for booleans
✗ Student Solution > detectType > primitives > should return "symbol" for symbols
✗ Student Solution > detectType > primitives > should return "bigint" for BigInt
✗ Student Solution > detectType > objects > should return "object" for plain objects
✗ Student Solution > detectType > objects > should return "array" for arrays
✗ Student Solution > detectType > objects > should return "function" for functions
✗ Student Solution > detectType > objects > should return "asyncfunction" for async functions
✗ Student Solution > detectType > built-in objects > should return "date" for Date
✗ Student Solution > detectType > built-in objects > should return "regexp" for RegExp
✗ Student Solution > detectType > built-in objects > should return "map" for Map
✗ Student Solution > detectType > built-in objects > should return "set" for Set
✗ Student Solution > detectType > built-in objects > should return "weakmap" for WeakMap
✗ Student Solution > detectType > built-in objects > should return "weakset" for WeakSet
✗ Student Solution > detectType > built-in objects > should return "promise" for Promise
✗ Student Solution > detectType > built-in objects > should return "error" for Error
✗ Student Solution > detectType > built-in objects > should return "arraybuffer" for ArrayBuffer
✗ Student Solution > detectType > edge cases > should return "object" for objects with null prototype
✗ Student Solution > detectType > edge cases > should handle boxed primitives
```



Github Repo Structure: Utility styles

```
import styles from '@course/styles'
import cx from '@course/cx'
```

Flexbox Basics	`flexRowCenter`, `flexColumnCenter`, `flexRowBetween`, `flexRowStart`, `flexRowEnd`, `itemsCenter`, `justifyStart`
Spacing (4-32px)	`padding[N]`, `paddingHor[N]`, `paddingVer[N]`, `margin[N]`, `marginHor[N]`, `marginVer[N]`
Sizing	`w100`, `h100`, `wh100` (100%), `w[N]px` to `h[N]px`
Visuals	`shadow[1-4]`, `br[0-128]`, `bgBlack[1-10]`, `bgWhite[1-10]`, `cBlack[1-10]`, `cWhite[1-10]`
Positioning	`pRel`, `pAbs`, `inset0`, `top0`, `left0`, `z1`, `z10`, `z100`
Typography	`fontXS`, `fontS`, `fontM`, `fontX`, `fontXL`

Github Repo Structure: Utility styles

```
import css from './card.module.css'
import flex from '@course/styles' // Shared utility classes
import cx from '@course/cx' // Classname composition helper

export const UserCard = ({ name, role }) => {
  return (
    <div className={cx(css.card, flex.flexRowBetween, flex.itemsCenter, flex.padding16)}>
      <div className={cx(flex.flexRowStart, flex.flexGap12, flex.itemsCenter)}>
        <Avatar />
        <div className={flex.flexColumnGap4}>
          <span className={css.name}>{name}</span>
          <span className={css.role}>{role}</span>
        </div>
      </div>

      <Button className={flex.marginLeft16}>Edit</Button>
    </div>
  )
}
```

Part 1

< Vanila Problems />

How to solve vanila problems

1 **Read & understand** Read the problem carefully and understand the requirements. 2–3 min

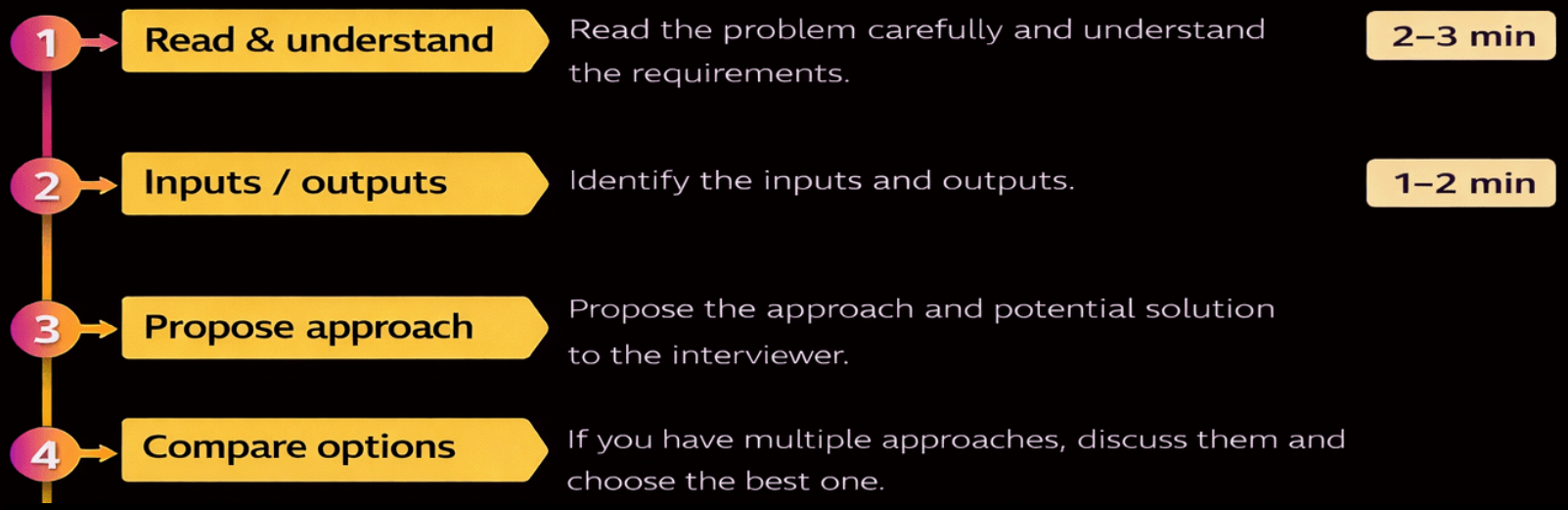
How to solve vanila problems

1	Read & understand	Read the problem carefully and understand the requirements.	2–3 min
2	Inputs / outputs	Identify the inputs and outputs.	1–2 min

How to solve vanila problems

1. **Read & understand** — Read the problem carefully and understand the requirements. — 2–3 min
2. **Inputs / outputs** — Identify the inputs and outputs. — 1–2 min
3. **Propose approach** — Propose the approach and potential solution to the interviewer.

How to solve vanila problems



How to solve vanila problems

1. **Read & understand** — Read the problem carefully and understand the requirements. — 2–3 min
2. **Inputs / outputs** — Identify the inputs and outputs. — 1–2 min
3. **Propose approach** — Propose the approach and potential solution to the interviewer.
4. **Compare options** — If you have multiple approaches, discuss them and choose the best one.
5. **Complexity** — Discuss the time and space complexity (if applicable).

How to solve vanila problems

1	Read & understand	Read the problem carefully and understand the requirements.	2–3 min
2	Inputs / outputs	Identify the inputs and outputs.	1–2 min
3	Propose approach	Propose the approach and potential solution to the interviewer.	
4	Compare options	If you have multiple approaches, discuss them and choose the best one.	
5	Complexity	Discuss the time and space complexity (if applicable).	
6	Implement	Implement your solution.	10–20 min

How to solve vanila problems

1	Read & understand	Read the problem carefully and understand the requirements.	2–3 min
2	Inputs / outputs	Identify the inputs and outputs.	1–2 min
3	Propose approach	Propose the approach and potential solution to the interviewer.	
4	Compare options	If you have multiple approaches, discuss them and choose the best one.	
5	Complexity	Discuss the time and space complexity (if applicable).	
6	Implement	Implement your solution.	10–20 min
7	Test & edge cases	Test your solution and handle edge cases.	5 min

How to solve vanila problems

1	Read & understand	Read the problem carefully and understand the requirements.	2–3 min
2	Inputs / outputs	Identify the inputs and outputs.	1–2 min
3	Propose approach	Propose the approach and potential solution to the interviewer.	
4	Compare options	If you have multiple approaches, discuss them and choose the best one.	
5	Complexity	Discuss the time and space complexity (if applicable).	
6	Implement	Implement your solution.	10–20 min
7	Test & edge cases	Test your solution and handle edge cases.	5 min
8	Optimize	If you have time, optimize your solution.	5 min

Problem 1: detectType

Goal: Implement a function `detectType(value)` that returns the type of any JavaScript value (returns `TType` union).

```
"array", "date", "number", "string,", ...
```

Level: warm up

Solving a problem

Inputs & Outputs:

Input: any JS value

Output: lowercase string representing the specific type

Approach:

This specific problem tests how well you know the JavaScript language and its built-in fields. The most straight forward approach is to use the typeof operator in combination with instanceof operator.

Naive approach

```
const detectType = (value: any) => {
  if (value === null) return 'null'

  if (typeof value === 'object') {
    if (value instanceof Date) return 'date'
    if (value instanceof Map) return 'map'
    if (Array.isArray(value)) return 'array'
    // ... check for other types
    return 'object'
  }

  return typeof value
}
```

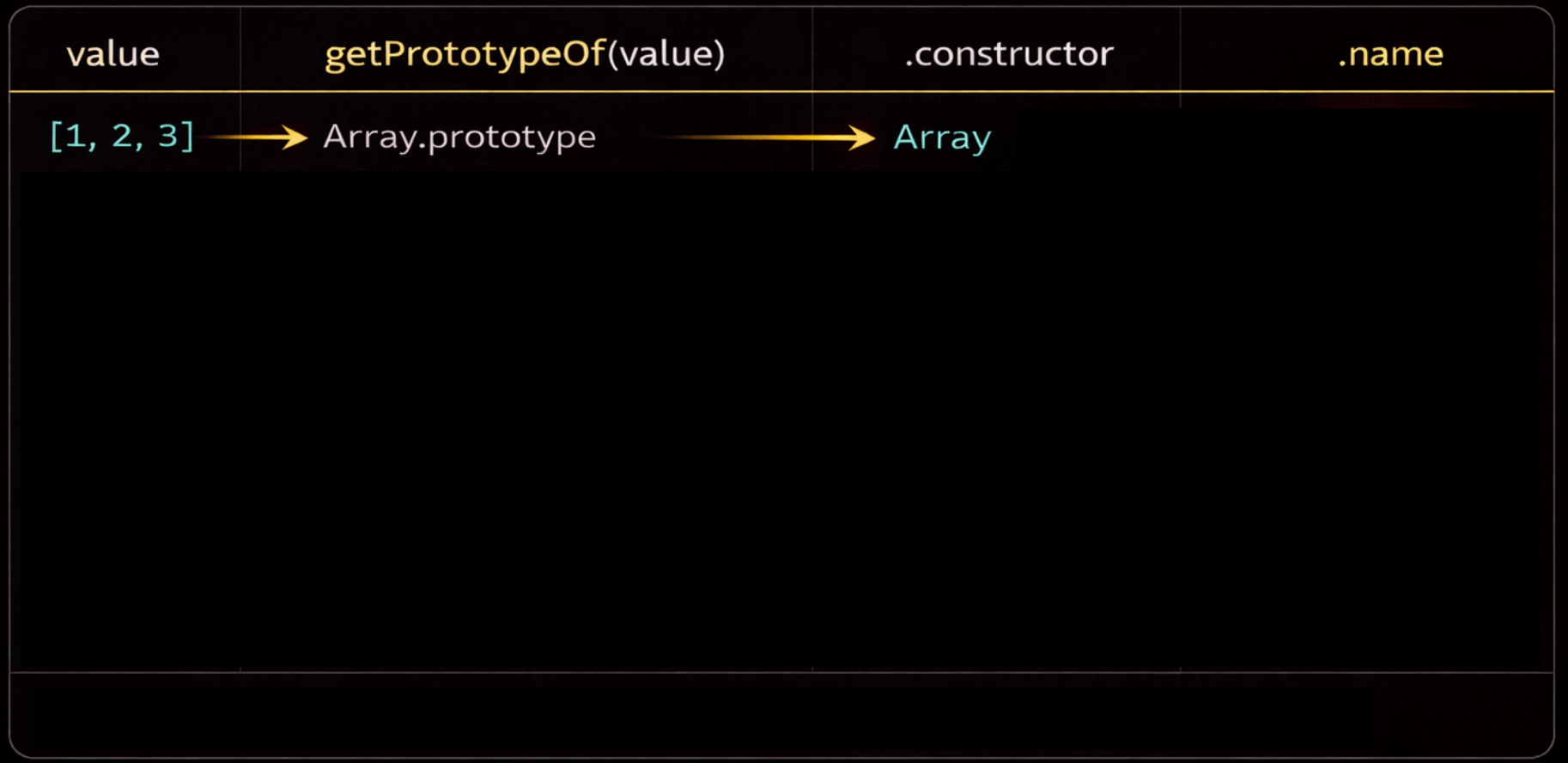
Optimization

value	getPrototypeOf(value)	.constructor	.name
[1, 2, 3]			

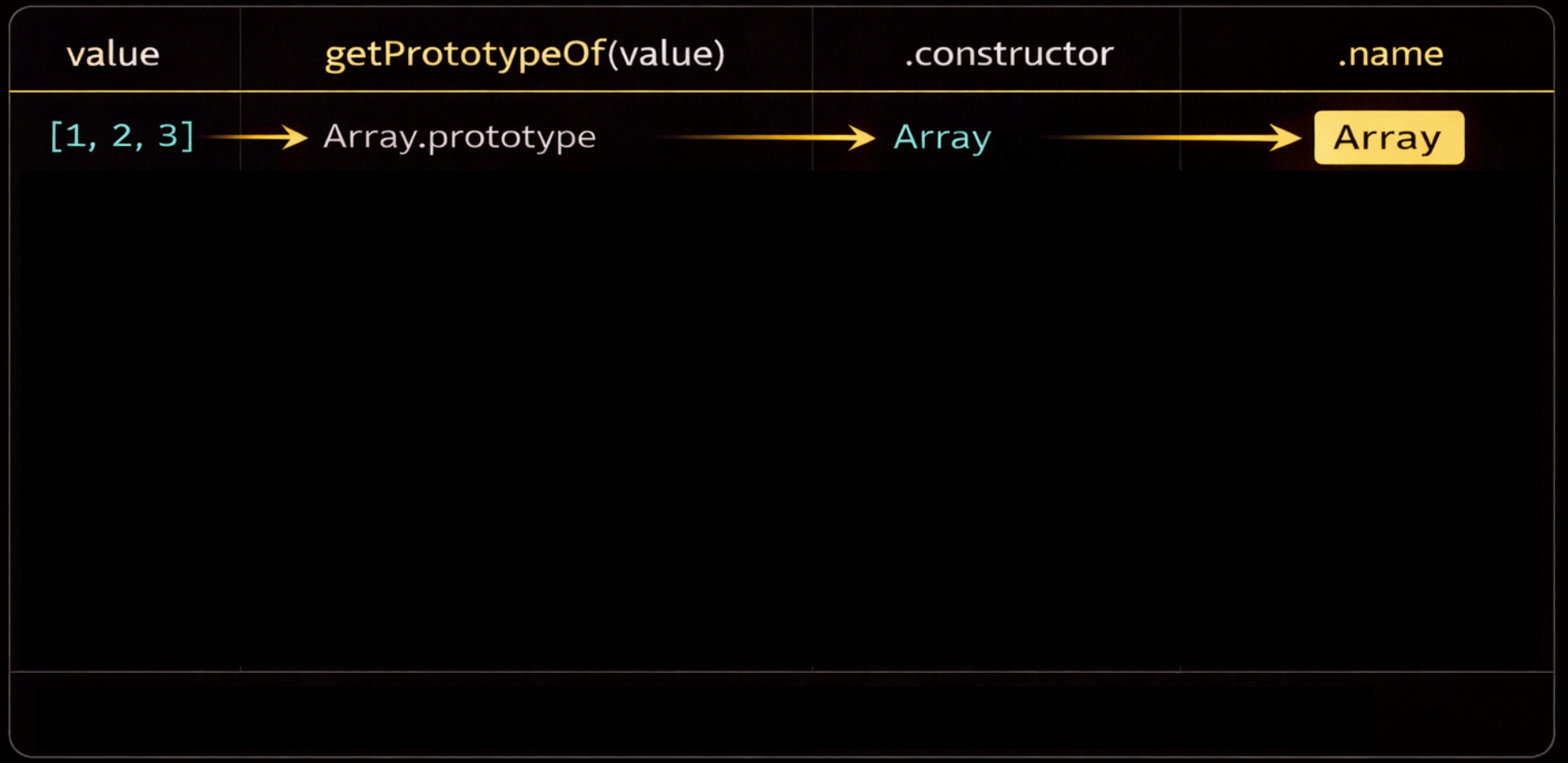
Optimization

value	getPrototypeOf(value)	.constructor	.name
[1, 2, 3] →	Array.prototype		

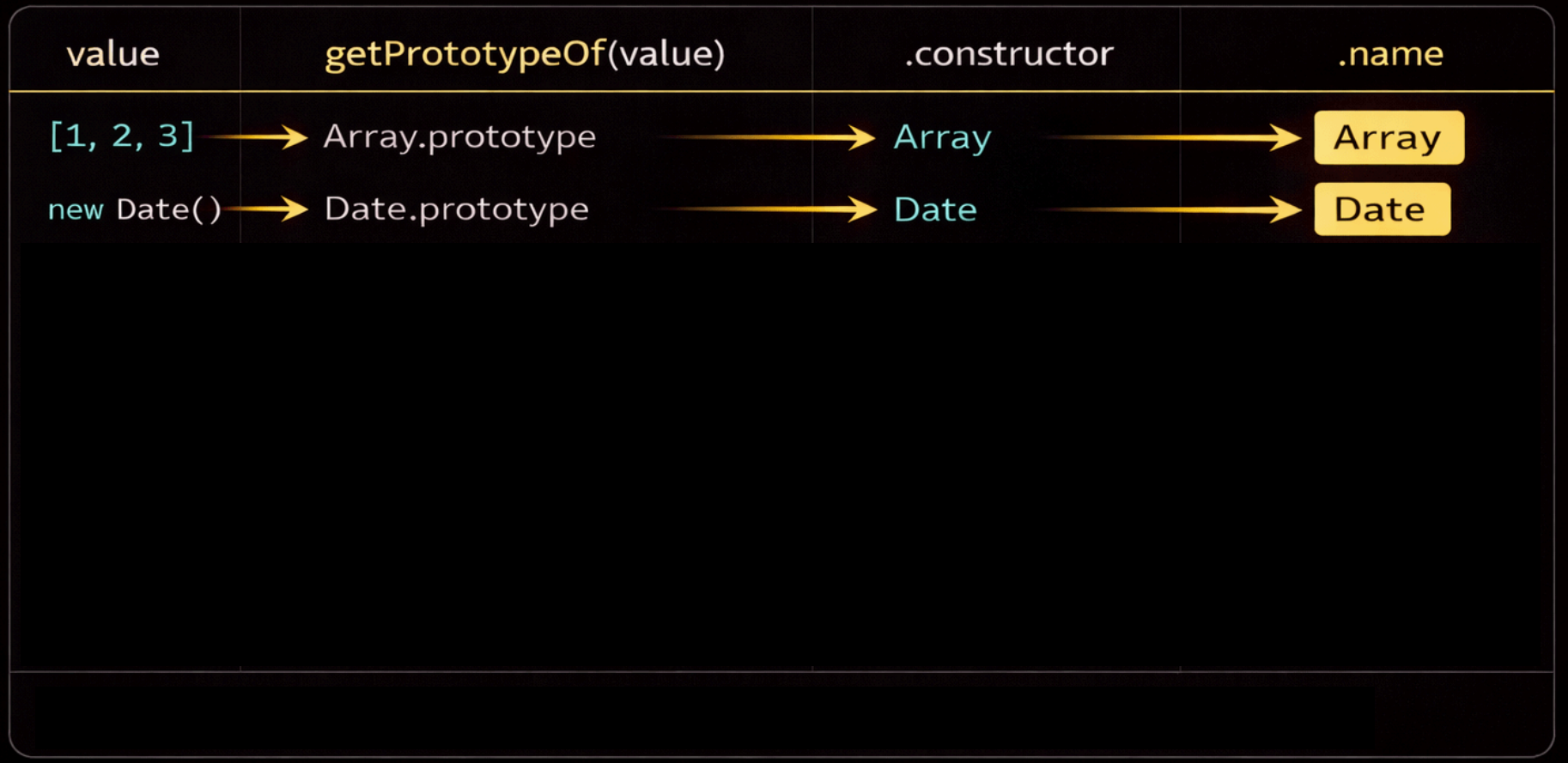
Optimization



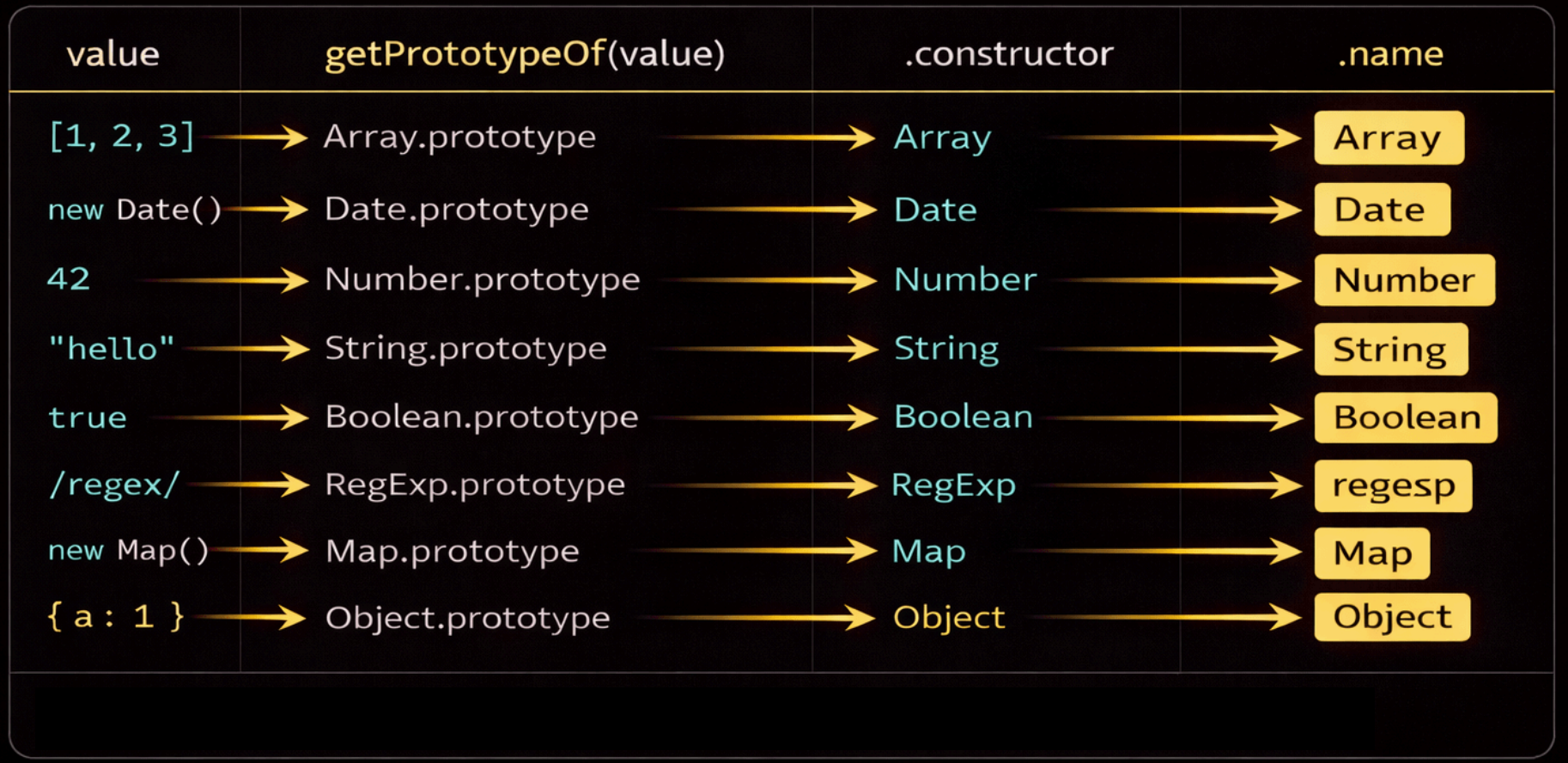
Optimization



Optimization



Optimization



Optimization

value	getPrototypeOf(value)	.constructor	.name
`[1, 2, 3]`	Array.prototype	Array	Array
`new Date()`	Date.prototype	Date	Date
`42`	Number.prototype	Number	Number
`"hello"`	String.prototype	String	String
`true`	Boolean.prototype	Boolean	Boolean
`/regex/`	RegExp.prototype	RegExp	regesp
`new Map()`	Map.prototype	Map	Map
`{ a : 1 }`	Object.prototype	Object	Object

Then just `.toLowerCase()` -> `"array"`, `"date"`, `"number"`, `"string,"`, ...

Solution

```
export const detectType = (value: any): TType => {
  if (value == null) {
    return `${value}`
  }
  return (
    Object.getPrototypeOf(value)?.constructor?.name ?? 'object'
  ).toLowerCase()
}
```

Problem 2: Debounce

Goal: Implement **debounce(fn, delay)** to ensure a function is only executed after **delay** milliseconds have passed since the last call.

Why?

- **Search bars:** Don't search on every keystroke, wait until user stops typing.
- **Save buttons:** Prevent accidental double submissions.

Requirements:

- Return a function that delays execution.
- Pass arguments **(...args)** and context **(this)** correctly.

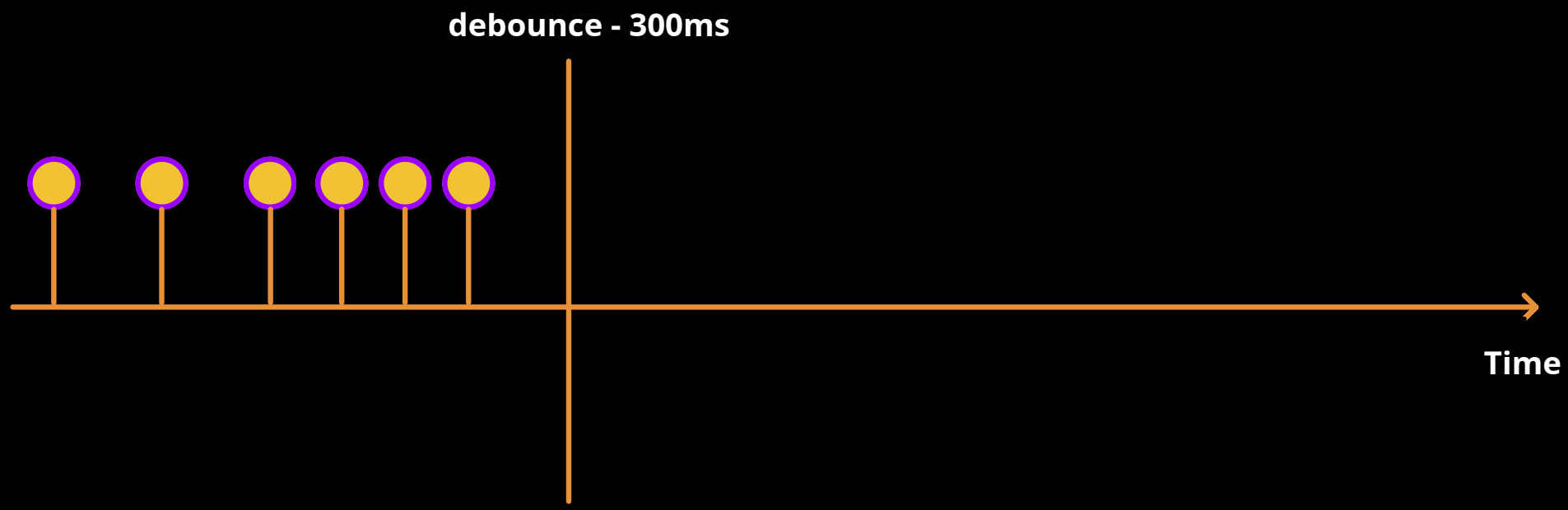
Level: easy

How debounce works

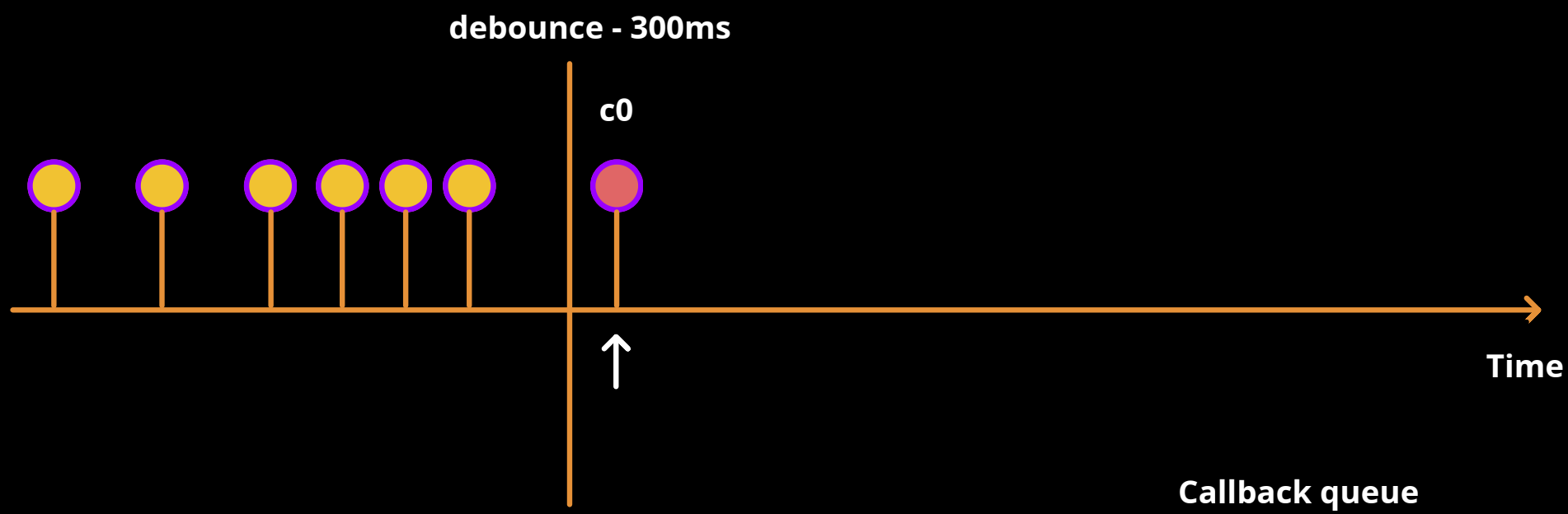
How debounce works

How debounce works

How debounce works



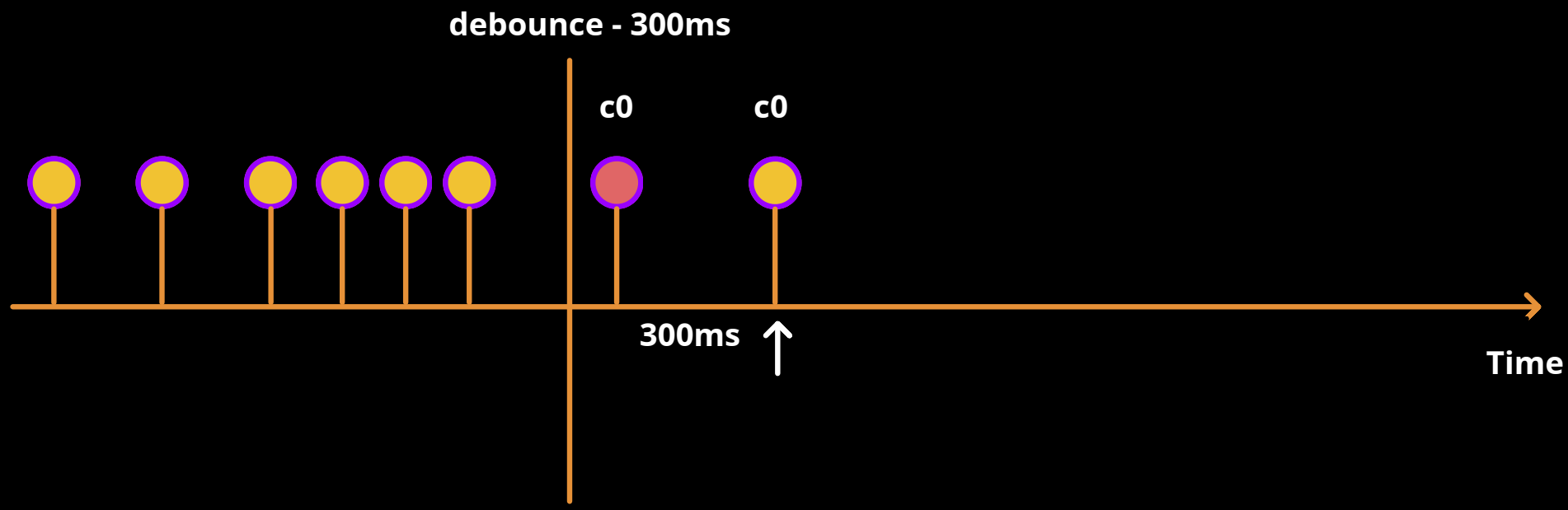
How debounce works



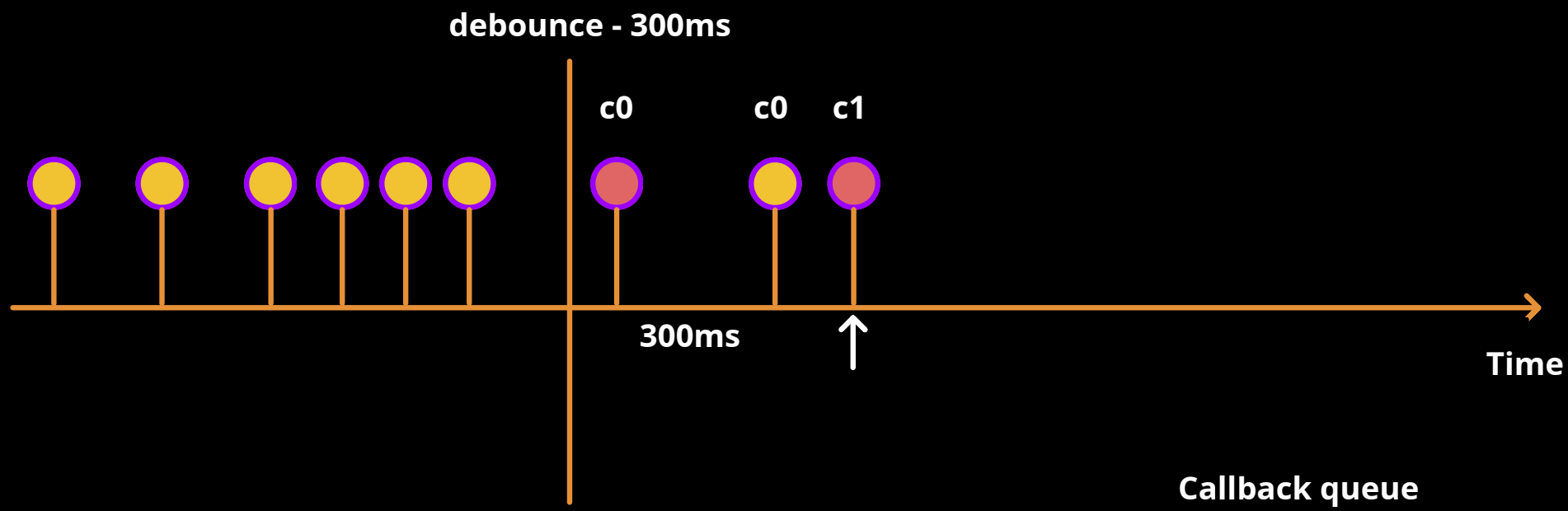
Callback queue

c0: {..args}	300ms

How debounce works



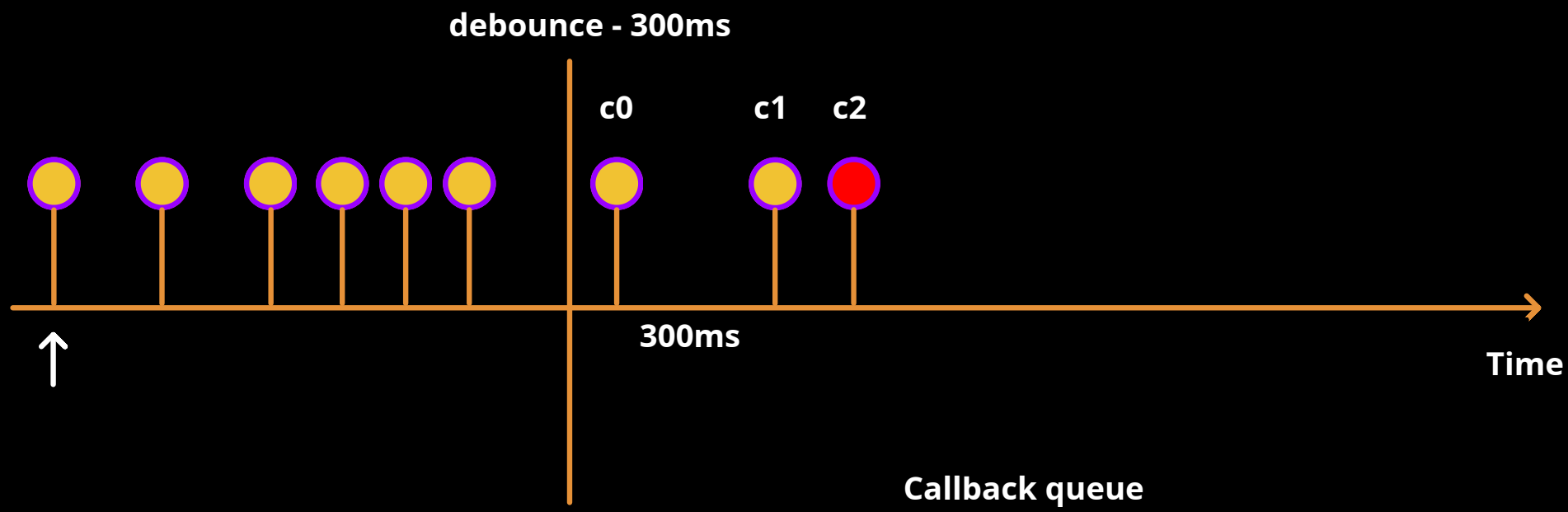
How debounce works



Callback queue

c1: {..args}	300ms

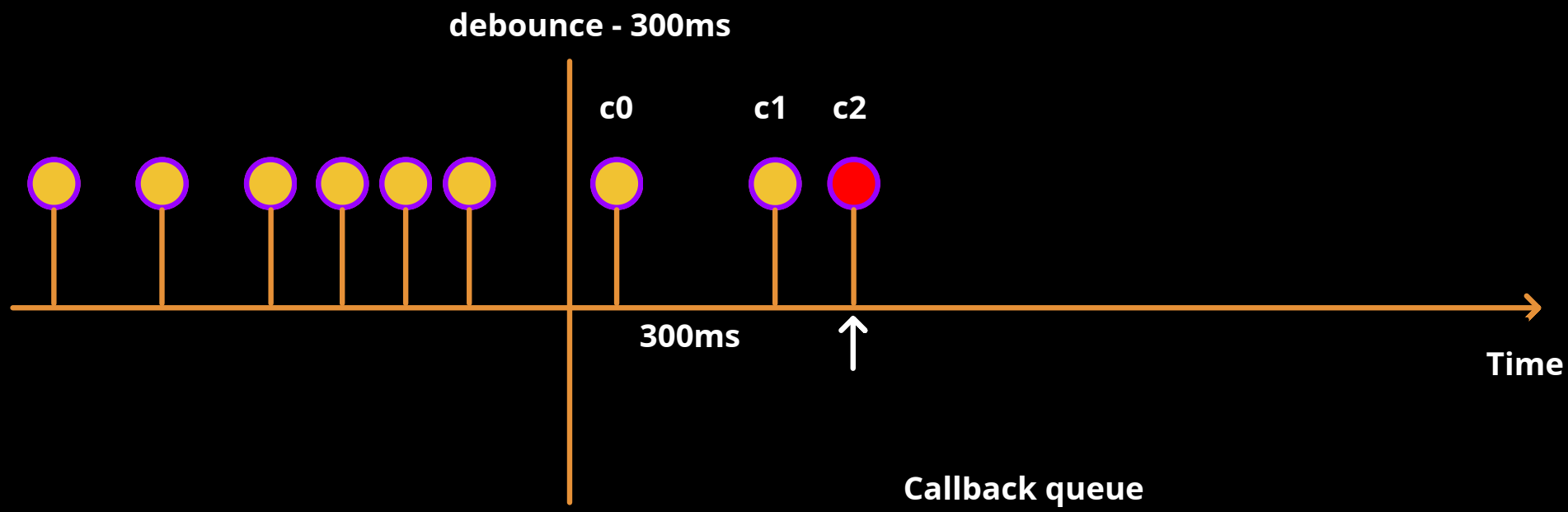
How debounce works



Callback queue

c2: {..args}	300ms

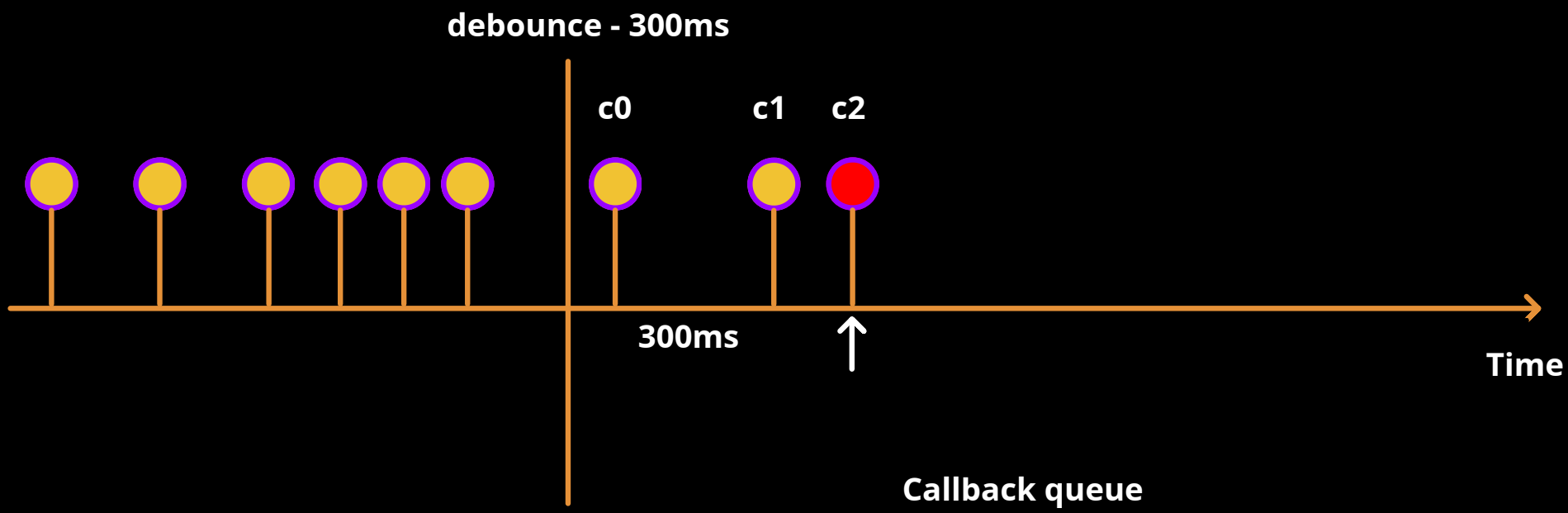
How debounce works



Callback queue

c2: {..args}	300ms

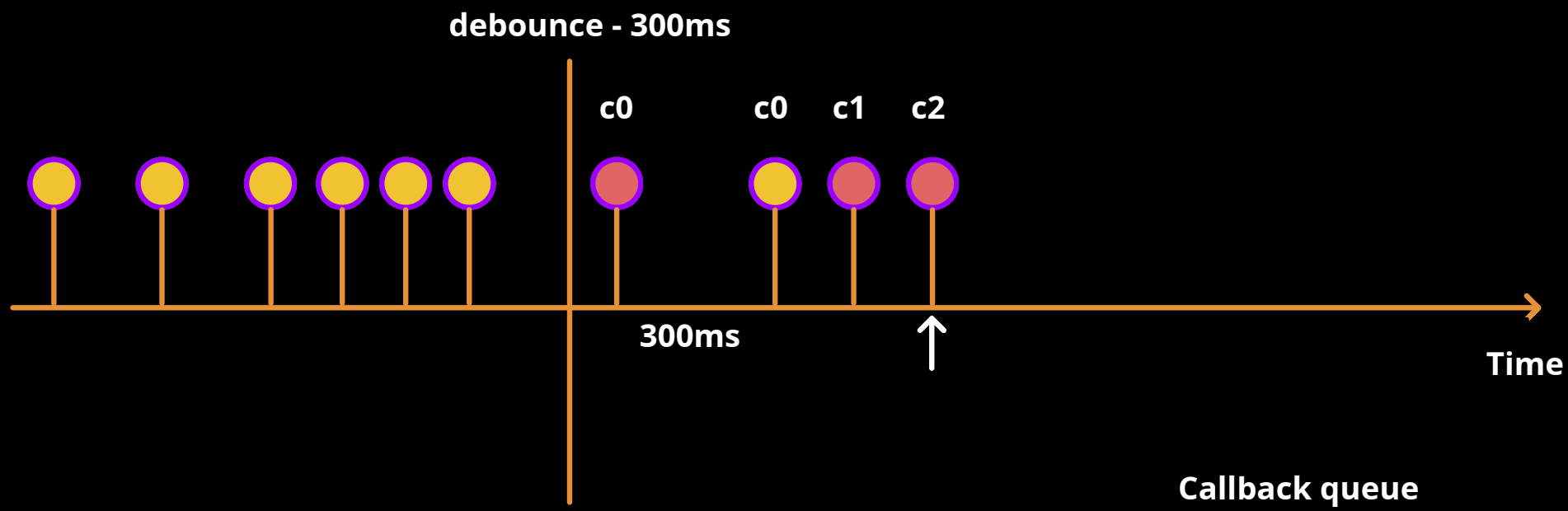
How debounce works



Callback queue

c2: {..args}	300ms

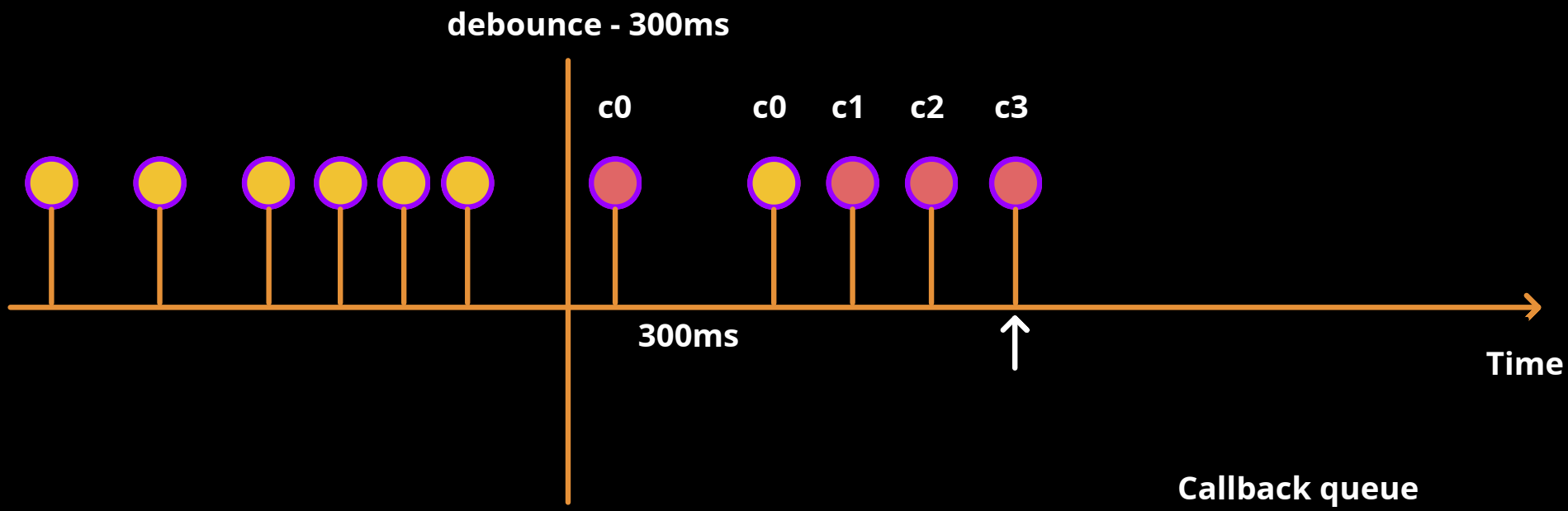
How debounce works



Callback queue

c2: {..args}	300ms

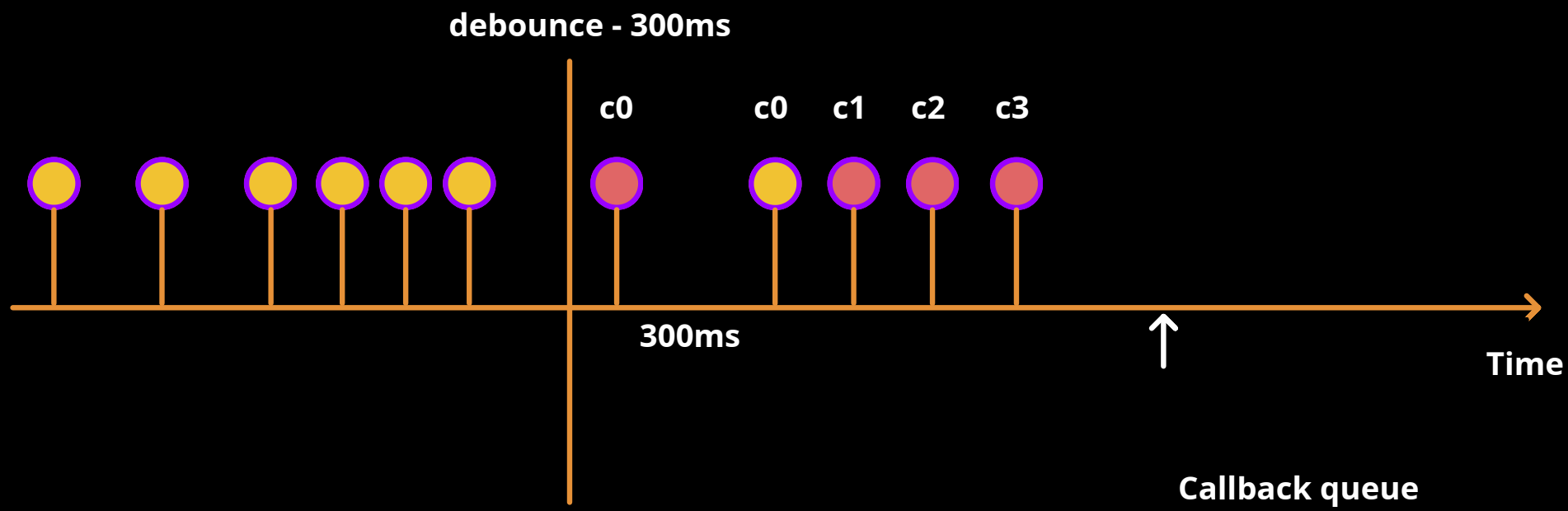
How debounce works



Callback queue

c3: {..args}	300ms

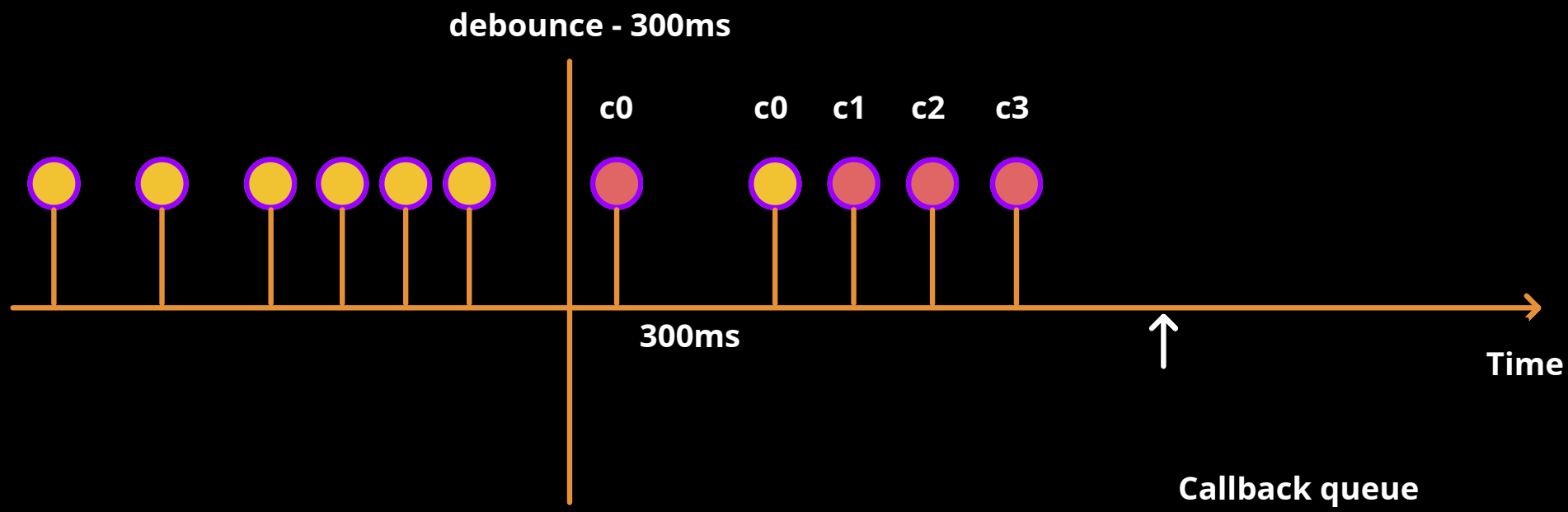
How debounce works



Callback queue

c3: {..args}	0ms

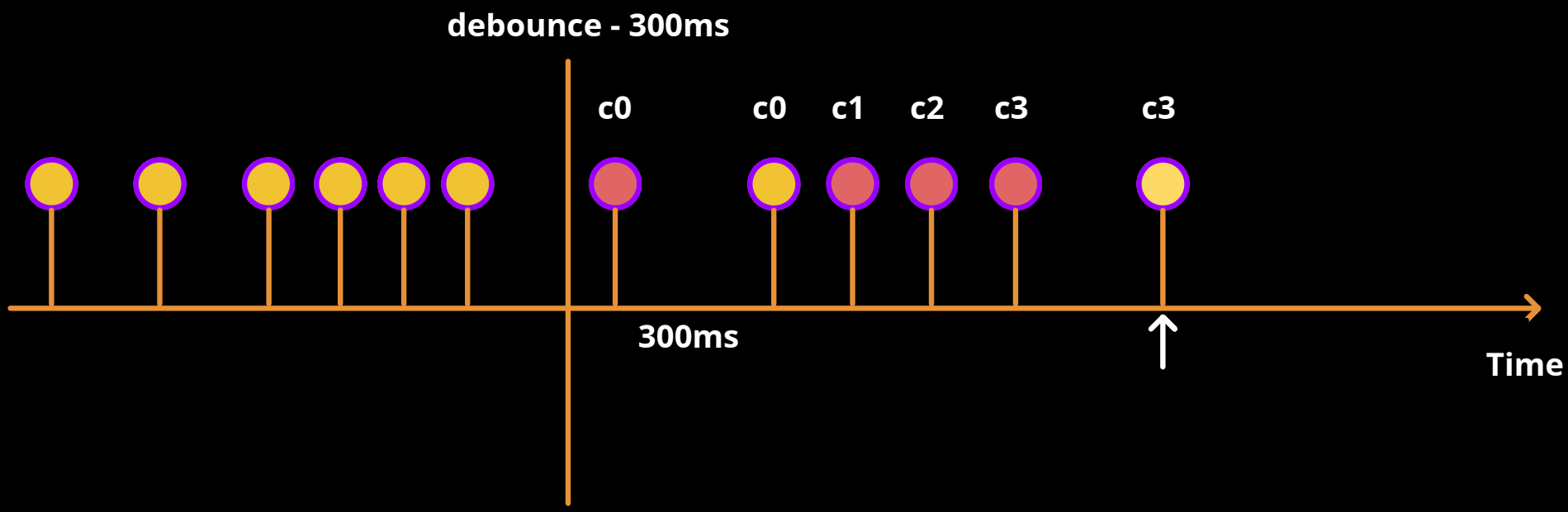
How debounce works



Callback queue

c3: {..args}	0ms

How debounce works



1. Inputs & Outputs

- Function `fn`
- Number `delay`
- Output
  - A **debounced** function

2. Approach

- Each call **cancels the previously scheduled execution**
- `setTimeout` returns a `timerId`
- `clearTimeout(timerId)` cancels the pending call
  - On every invocation:
    - clear previous timer
    - **schedule** a new one with the **latest arguments**

Solution

```ts
export function debounce<T extends (...args: any[]) => any>(
  fn: T,
  delay: number,
 ): (...args: Parameters<T>) => void {
  let timerId: Timer | null = null

  return function (this: any, ...args: Parameters<T>) {
    // 1. Clear existing timer
    if (timerId) clearTimeout(timerId)

    // 2. Schedule new execution
    timerId = setTimeout(() => {
      fn.apply(this, args)
    }, delay)
  }
}
```

Challenge 1: Tuple length

Given a tuple, create a generic `**Length**` that ouputs the length of tuple array

```
type tesla = ['tesla', 'model 3', 'model X', 'model Y']
type L = Length<tesla>  // 4
```

Challenge 1: Tuple length | Solution

Given a tuple, create a generic `**Length**` that ouputs the length of tuple array

```ts
type Length<T extends readonly any[]> = T['length']
```

Challenge 2: Extract first element type

Given a tuple, create a generic `**First**` that ouputs the type of first element

```
type A = First<[3, 2, 1]>  // 3
type B = First<[]>         // never
```

Challenge 2: Extract first element type | Solution

Given a tuple, create a generic `**First**` that ouputs the type of first element

```
1 type First<T extends any[]> = T extends [infer F, ...any[]] ? F : never
```

Problem 3: Throttle

Goal:

- Implement `throttle(fn, delay)` to ensure a function executes at most once every `delay` milliseconds.
- Why?
  - Scroll listeners: Check scroll position only every **100ms**, not every pixel.
  - Gaming: Limit fire rate regardless of how fast usage clicks.

2. Requirements:

- Execute immediately if enough time has passed.
- If called too frequently, ignore calls until cooldown expires.

- Requirements:
  - Execute immediately if enough time has passed.
  - If called too frequently, ignore calls until cooldown expires.

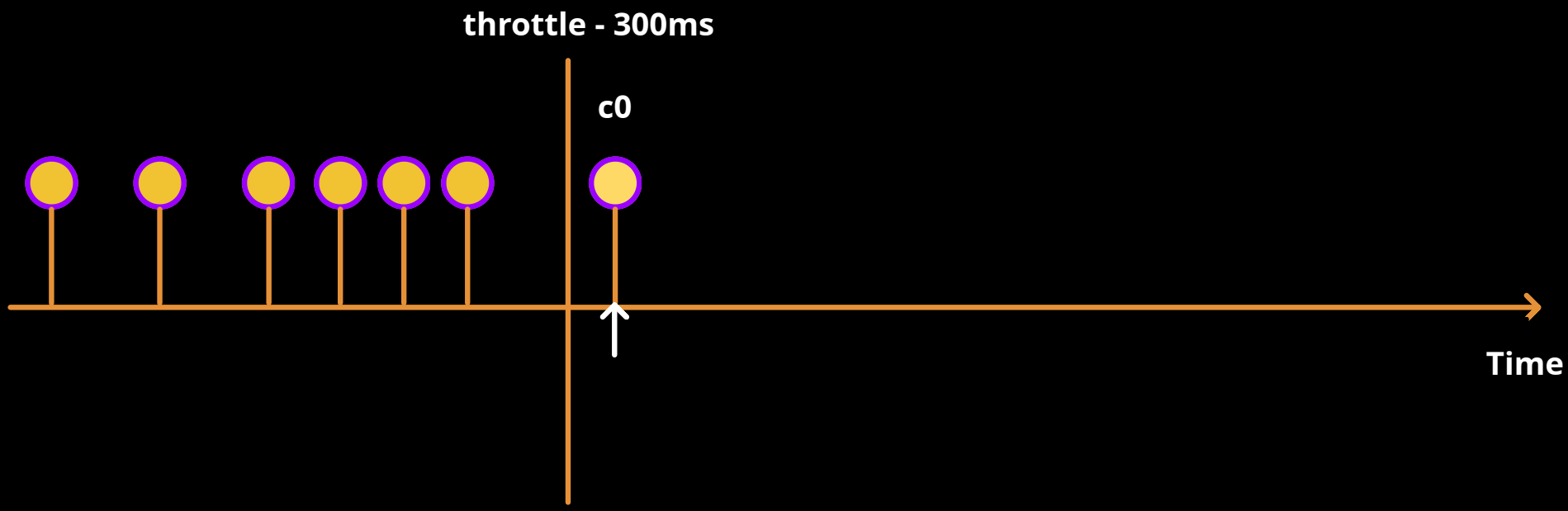
Level: Easy

How throttle works?

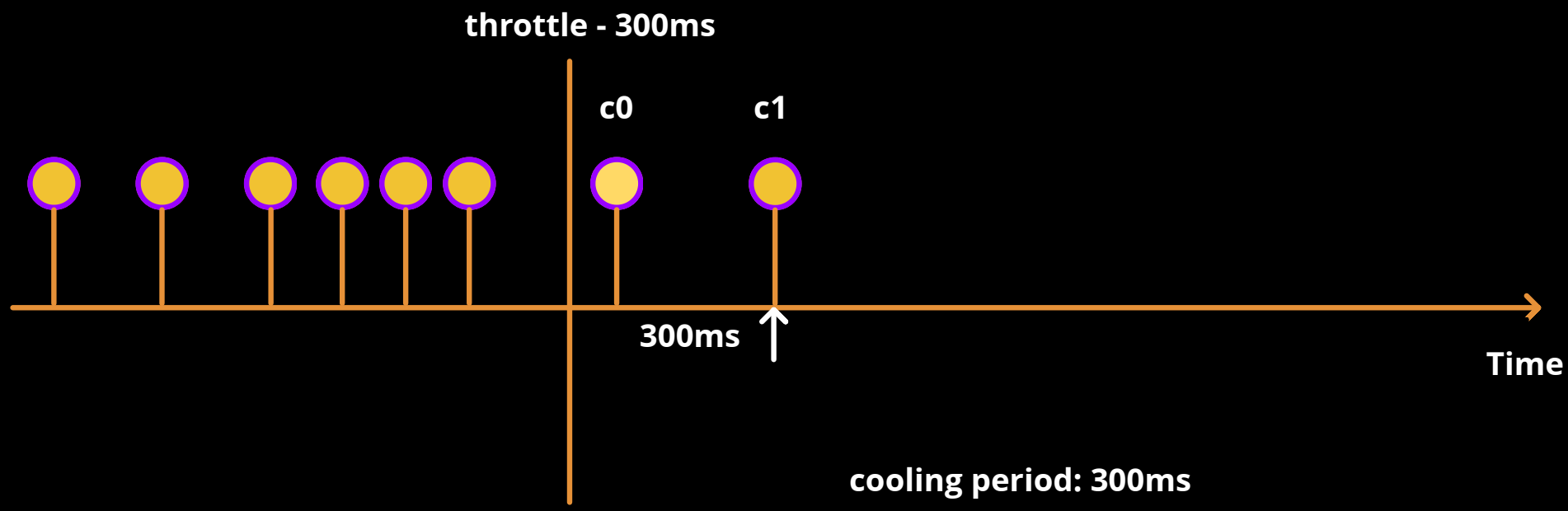
How throttle works?

How throttle works?

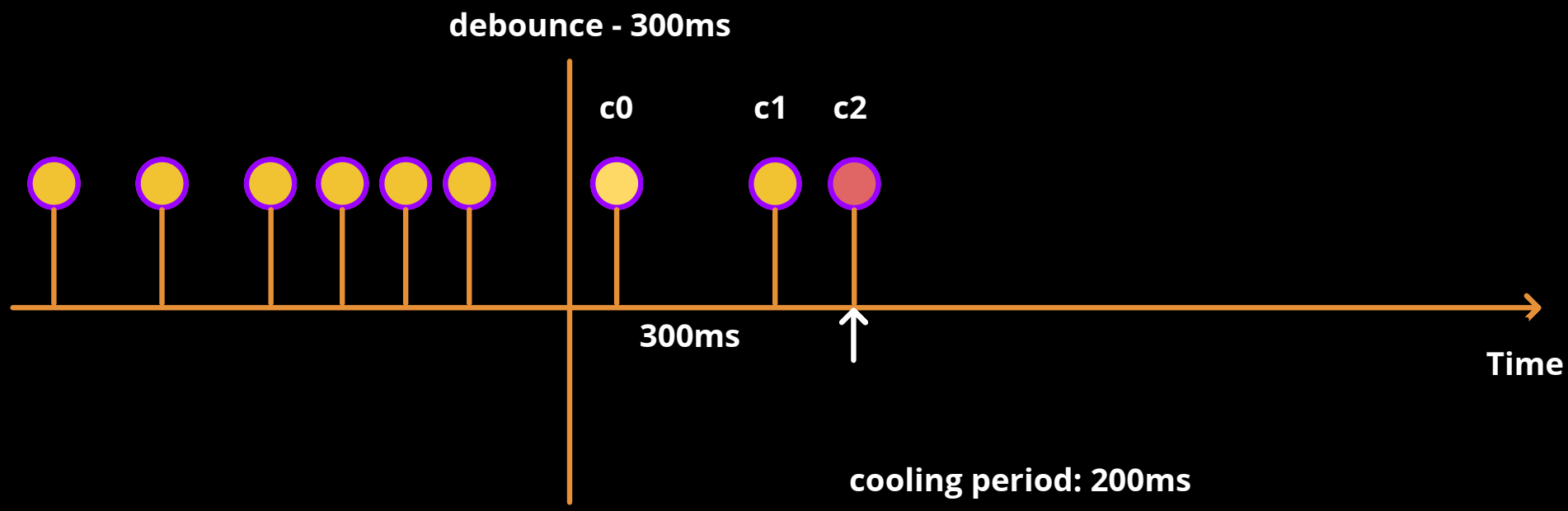
How throttle works?



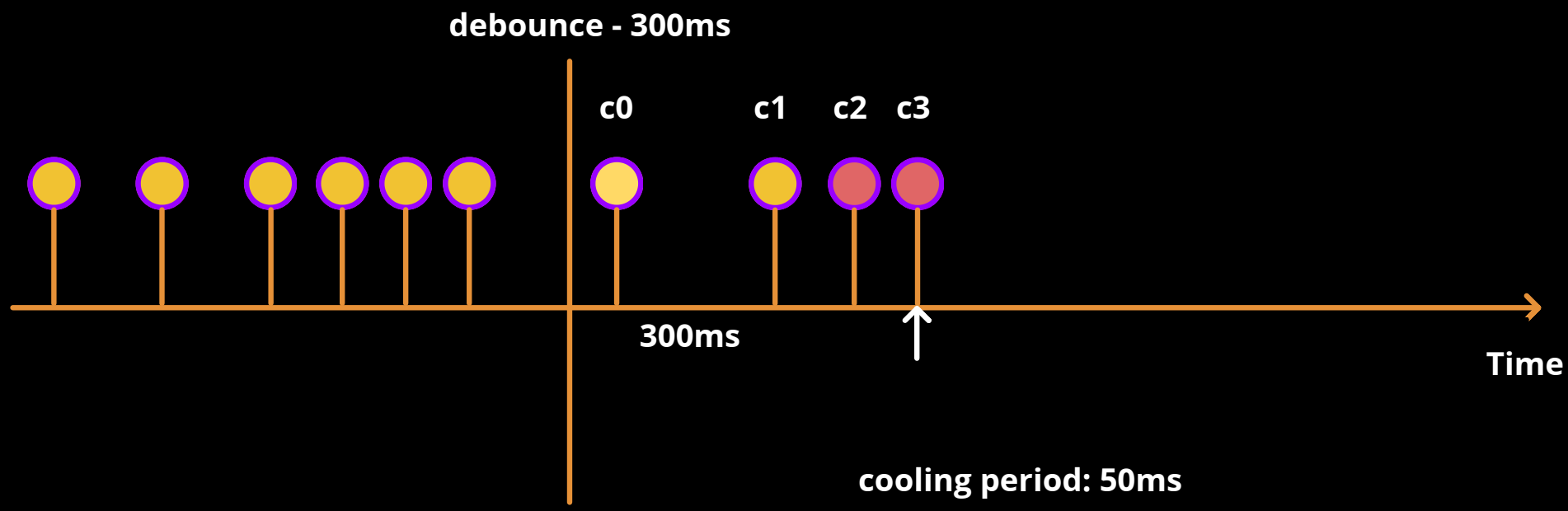
How throttle works?



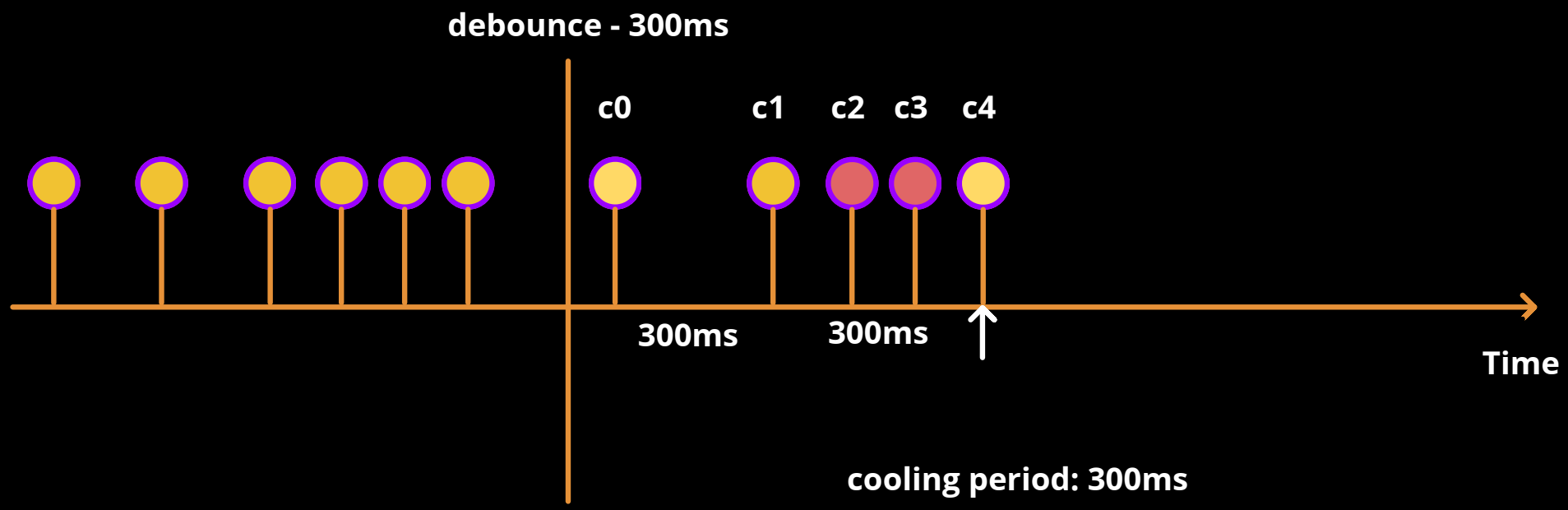
How throttle works?



How throttle works?



How throttle works?



1. Inputs & Outputs

- **Input** Function **fn**
- Number **delay**
- **Output**
  - A **throttled** function

2. Approach

- Track the `lastTime` the function executed
- On every call:
  - Compute: `now - lastTime >> delay ?`
- **IF YES**
  - Run the function
  - Update `lastTime = now`
- **IF NO**
  - Do nothing
  - Drop the call (cooldown still active)

Solution: throttle

```ts
export function throttle<T extends (...args: any[]) => any>(
  fn: T,
  delay: number,
 ): (...args: Parameters<T>) => void {
  let lastTime = 0

  return function (this: any, ...args: any[]) {
    const now = Date.now()

    // Check if cooldown has passed
    if (now - lastTime >= delay) {
      fn.apply(this, args)
      lastTime = now
    }
  }
}
```

Problem 4: ES5 Extends

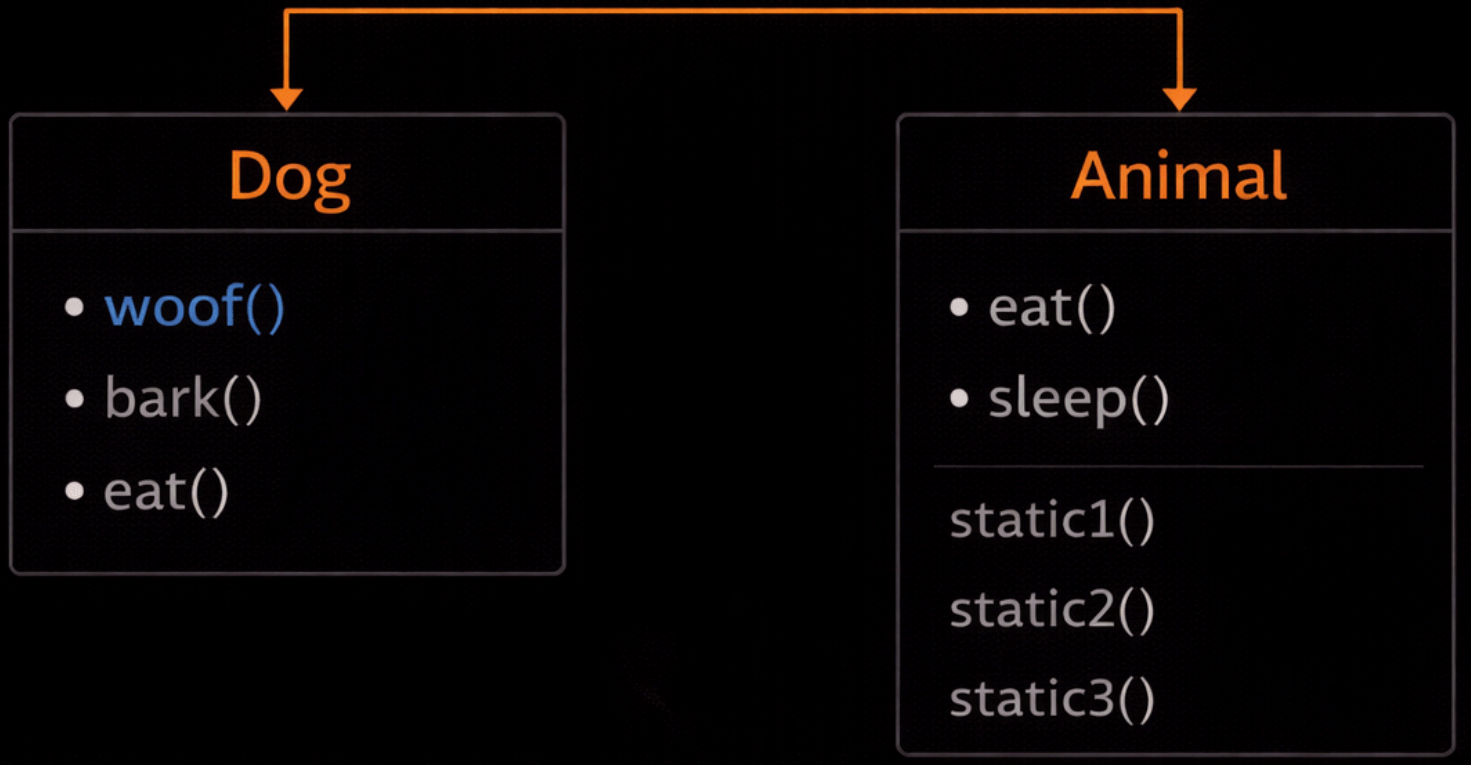
- **Goal:** Implement `myExtends(SuperType, Subtype)` function that mimics ES5 prototype-based inheritance — combining two constructor functions into one that inherits both instance properties and prototype methods.
- **Why?**
  - Understanding the prototype chain is fundamental to JS.
  - Modern `'class'` syntax is just syntactic sugar over this.

2. Requirements:

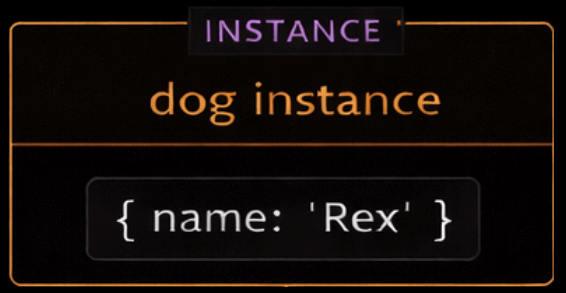
1. The returned constructor must call both `SuperType` and `SubType` constructors (Constructor Stealing).
2. Instances must have access to methods from both `SuperType.prototype` and `SubType prototype` (Prototype Chain).
3. Static methods from `SuperType` should be inherited by the returned constructor (Static Inheritance).

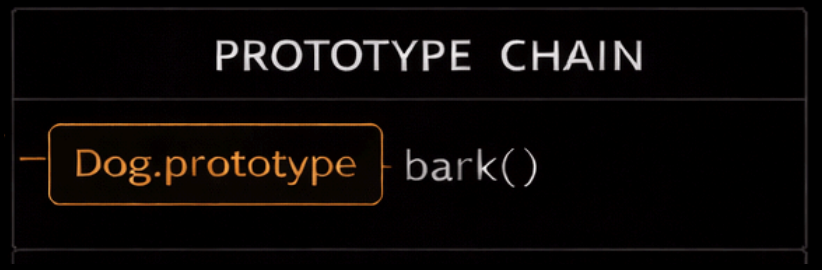
Level: Easy

Dog extends Animal

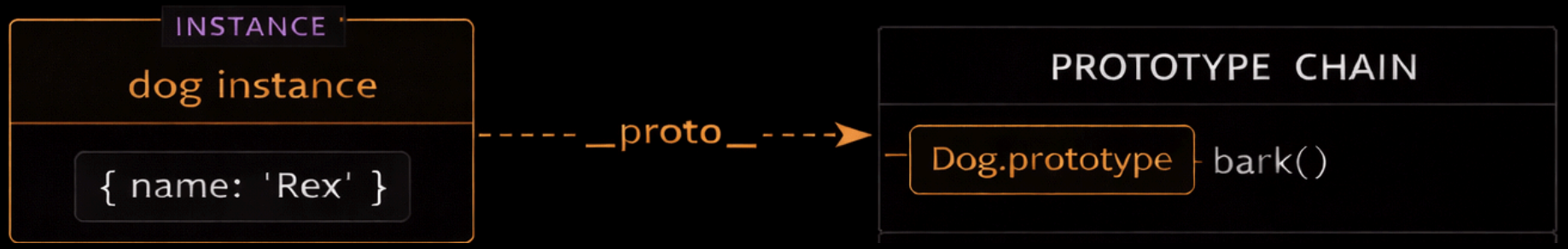


Understand ES5 Extends

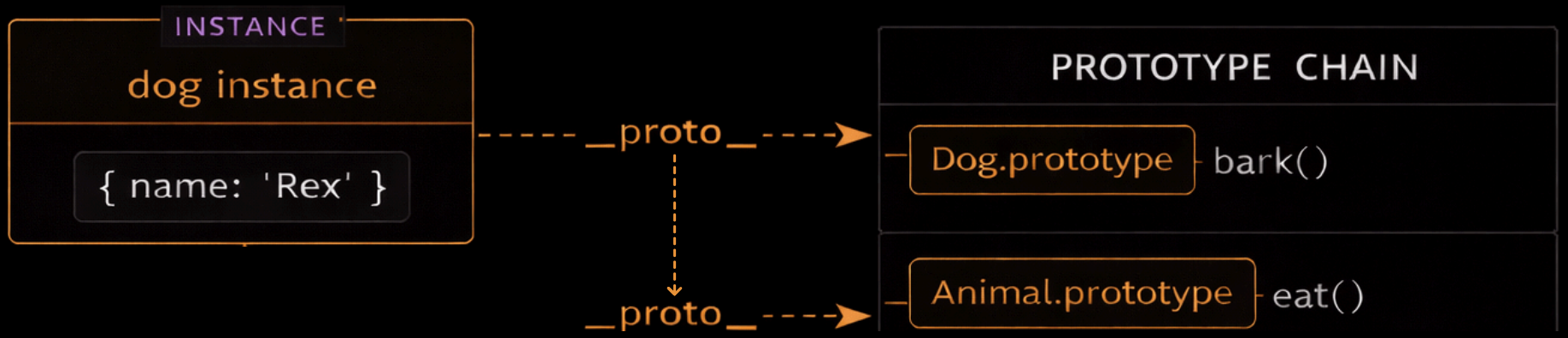




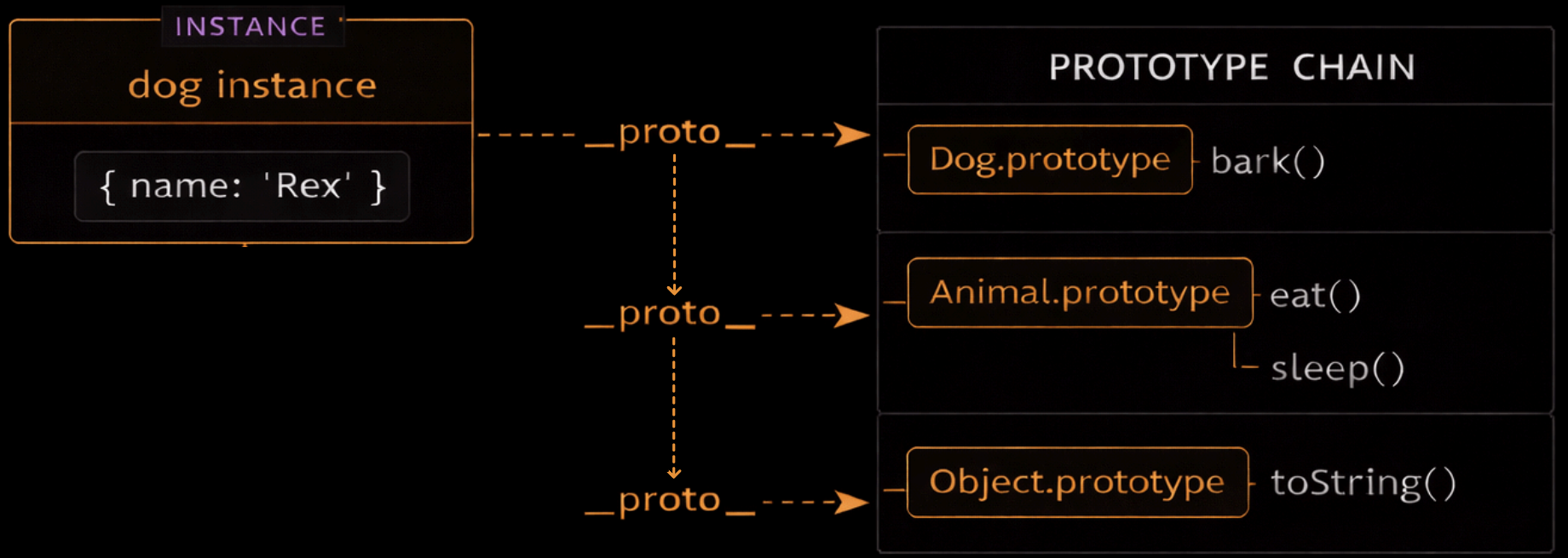
Understand ES5 Extends



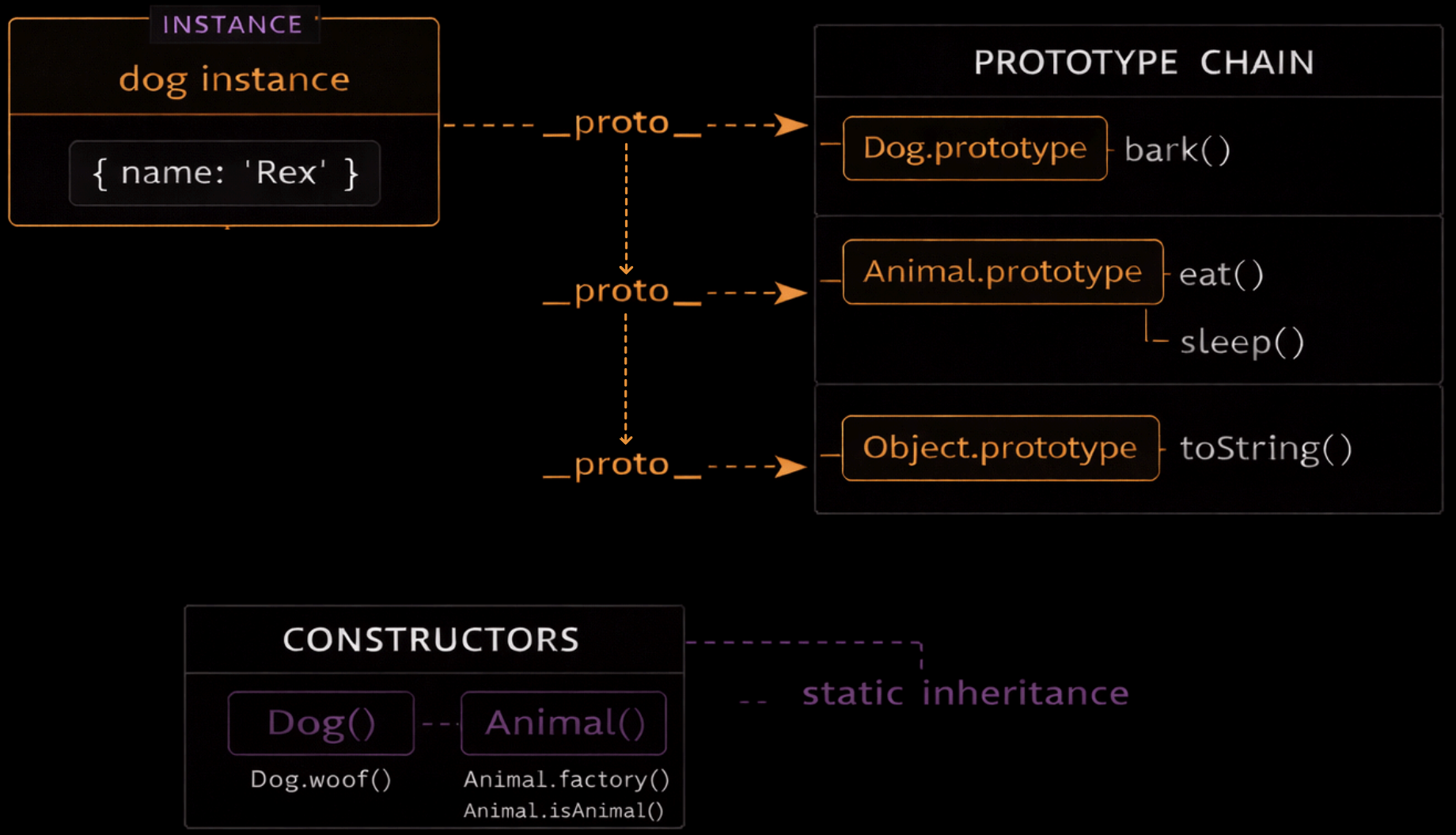
Understand ES5 Extends



Understand ES5 Extends



Understand ES5 Extends



Solution: myExtends

```ts
export const myExtends = (SuperType: Function, SubType: Function) => {
  function MyType(this: any, ...args: any[]) {
    // Create a new object with SubType.prototype as its prototype
    // This sets up: instance.__proto__ === SubType.prototype
    const obj = Object.create(SubType.prototype)

    // Call SuperType constructor with the new object as `this`
    // This adds SuperType's instance properties (e.g., this.name = name)
    SuperType.apply(obj, args)

    // Call SubType constructor with the new object as `this`
    // This adds SubType's instance properties (e.g., this.subProp = 'sub')
    SubType.apply(obj, args)

    // Return the fully initialized object
    return obj
  }

  // Set up prototype inheritance:
  // SubType.prototype.__proto__ = SuperType.prototype
  // This allows instances to access methods from SuperType.prototype
  // through the prototype chain: instance -> SubType.prototype -> SuperType.prototype
  Object.setPrototypeOf(SubType.prototype, SuperType.prototype)

  // Set up static/constructor inheritance:
  // MyType.__proto__ = SuperType
  // This allows MyType to inherit static methods from SuperType
  Object.setPrototypeOf(MyType, SuperType)

  return MyType
}
```

Challenge 3: Tuple to Union

Given a tuple, create a generic `**TupleToUnion**` that creates a union

```ts
type R = TupleToUnion<[123, '456', true]>  // 123 | '456' | true
```

Challenge 3: Tuple to Union | Solution

Given a tuple, create a generic `**TupleToUnion**` that creates a union

```ts
type TupleToUnion<T extends readonly any[]> = T[number]

type R = TupleToUnion<[123, '456', true]>  // 123 | '456' | true
```

Challenge 4: Pick

Implement the built-in Pick<T, K> generic without using it.
Constructs a type by picking the set of properties `K` from `T`.

```
interface Todo {
  title: string
  description: string
  completed: boolean
}

type TodoPreview = MyPick<Todo, 'title' | 'completed'>

// { title: string, completed: boolean }
```

Challenge 4: Pick | Solution

Implement the built-in Pick<T, K> generic without using it.
Constructs a type by picking the set of properties `K` from `T`.

```
type MyPick<T, K extends keyof T> = {
  [P in K]: T[P]
}
```

Problem 5: deep equals

Goal Implement `deepEquals(a, b)`

→ Check whether two values are structurally identical

Requirements:

1 Primitives
- Use strict equality `===`
- Special case: handle NaN

2 Objects / Arrays
- Compare keys length
- Recursively compare values

3 No Type Coercion
- `"1" !== 1`

4 Circular References
- Detect and handle cycles (optional, but strong solution)

Level: Medium

How Circular References Break Recursive Functions

A circular reference occurs when an object refers back to itself (directly or indirectly).

Without cycle detection, a recursive function will follow the reference forever and eventually **crash:**

Setup:

```
const a = { name: "Alice"}
a.self = a  // circular reference
```

```
const b = { name: "Alice"}
b.self = b  // same structure, also circular
```

What Happens Without Cycle Detection?

How Circular References Break Recursive Functions

A circular reference occurs when an object refers back to itself (directly or indirectly).

Without cycle detection, a recursive function will follow the reference forever and eventually **crash:**

Setup:

```
const a = { name: "Alice"}
a.self = a  // circular reference
```

```
const b = { name: "Alice"}
b.self = b  // same structure, also circular
```

What Happens Without Cycle Detection?

```
deepEquals(a, b)
```

How Circular References Break Recursive Functions

A circular reference occurs when an object refers back to itself (directly or indirectly).

Without cycle detection, a recursive function will follow the reference forever and eventually **crash:**

Setup:

```
const a = { name: "Alice"}
a.self = a   // circular reference
```

```
const b = { name: "Alice"}
b.self = b   // same structure, also circular
```

What Happens Without Cycle Detection?

```
deepEquals(a, b)
  ↪ key "name" → "Alice" === "Alice" ✓
```

How Circular References Break Recursive Functions

A circular reference occurs when an object refers back to itself (directly or indirectly).

Without cycle detection, a recursive function will follow the reference forever and eventually **crash:**

Setup:

```
const a = { name: "Alice"}
a.self = a   // circular reference
```

```
const b = { name: "Alice"}
b.self = b   // same structure, also circular
```

What Happens Without Cycle Detection?

```
deepEquals(a, b)
  → key "name" → "Alice" === "Alice" ✓
    → self → deepEquals(a.self, b.self)      // a.self === a
```

How Circular References Break Recursive Functions

A circular reference occurs when an object refers back to itself (directly or indirectly).

Without cycle detection, a recursive function will follow the reference forever and eventually **crash:**

Setup:

```
const a = { name: "Alice"}
a.self = a   // circular reference
```

```
const b = { name: "Alice"}
b.self = b   // same structure, also circular
```

What Happens Without Cycle Detection?

```
deepEquals(a, b)
  ↪ key "name" → "Alice" === "Alice" ✓
    ▸→ self → deepEquals(a.self, b.self)      // a.self === a
       ▸→→ key "name" ✓                       // b.self === b
```

How Circular References Break Recursive Functions

A circular reference occurs when an object refers back to itself (directly or indirectly).

Without cycle detection, a recursive function will follow the reference forever and eventually **crash:**

Setup:

```
const a = { name: "Alice"}
a.self = a   // circular reference
```

```
const b = { name: "Alice"}
b.self = b   // same structure, also circular
```

What Happens Without Cycle Detection?

```
deepEquals(a, b)
  → key "name" → "Alice" === "Alice" ✓
    → self → deepEquals(a.self, b.self)      // a.self === a
      →→ key "name" ✓                        // b.self === b
      →→ key "self → deepEquals(a.self.self, b.self.self
```

How Circular References Break Recursive Functions

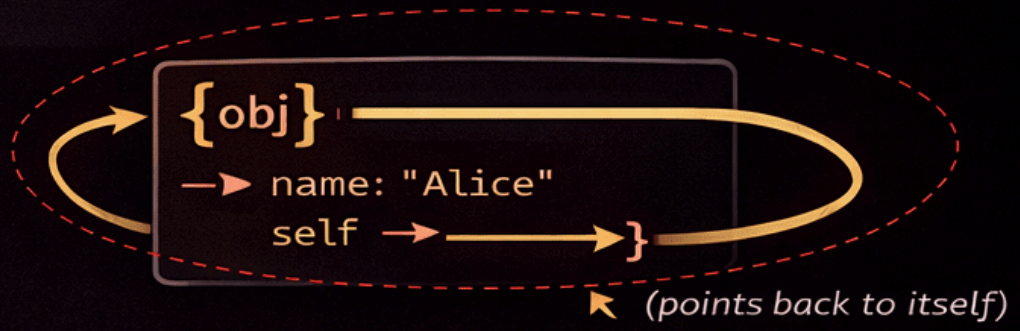
A circular reference occurs when an object refers back to itself (directly or indirectly).

Without cycle detection, a recursive function will follow the reference forever and eventually **crash:**

Setup:

```
const a = { name: "Alice"}
a.self = a   // circular reference
```

```
const b = { name: "Alice"}
b.self = b   // same structure, also circular
```



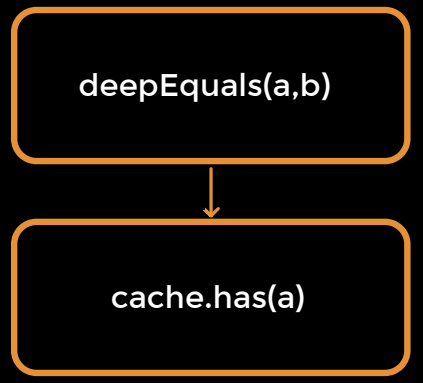
What Happens Without Cycle Detection?

```
deepEquals(a, b)
  → key "name" → "Alice" === "Alice" ✓
    → self → deepEquals(a.self, b.self)      // a.self === a
                                             // b.self === b
      → key "name" ✓
      → key "self → deepEquals(a.self.self, b.self.self
      RangeError:
      RangeError: Maximum call stack size exceeded
```

The algorihm keeps comparing the same objects repeatedly because it has no memory of visited references.

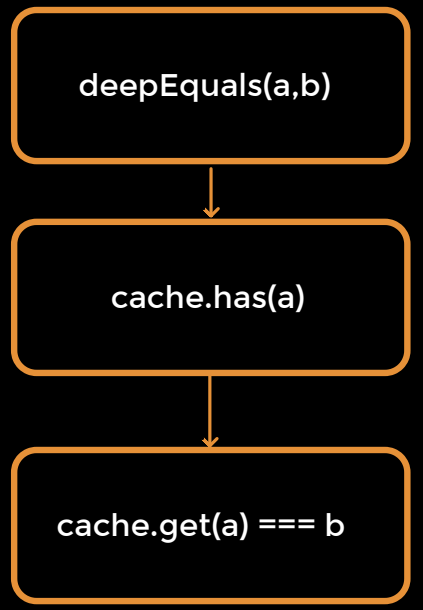
deepEquals(a,b)

```
cache = Map<object, object>
```

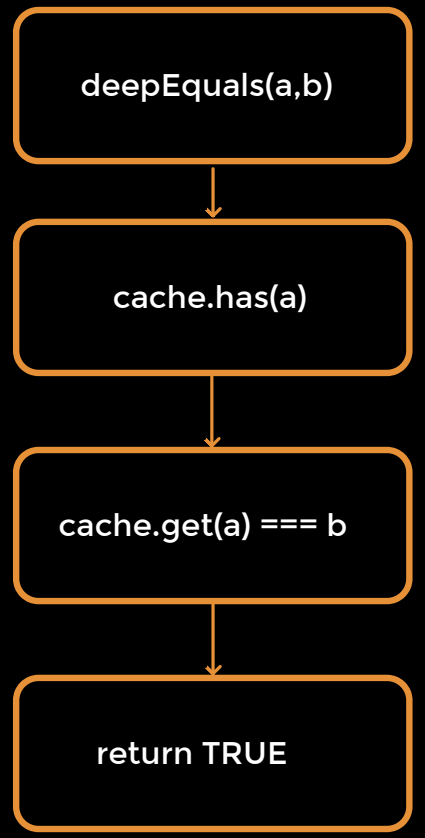



```
cache = Map<object, object>
```

```
cache = Map<object, object>
```



```
cache = Map<object, object>
```



Solving Deep Equals

1. **Inputs & Outputs:**
   - Input: Two values **a and b** (any type).
   - Output: **boolean.**
2. **Approach:**
   - **Base Case: If a === b return true.**
   - **Type Check:** If *typeof* differs or **one is null**, return **false.**
   - **Recursion:**
     - If **Array:** Compare length, then each item.
     - If **Object:** Compare keys length, keys existence, then each value.
   - **Cycle Detection:**
     - Use **Map** or **WeakMap** to track visited pairs.

Solution: deep equals

```
export function deepEquals(a: any, b: any, cache = new Map()): boolean {
    if (a === b || (cache.has(a) && cache.get(a) === b)) {
        return true;
    }
    const [typeA, typeB] = [
        detectType(a),
        detectType(b),
    ];
    if (typeA !== typeB) {
        return false;
    }
    if (typeof a !== 'object') {
        return a === b;
    }
    const [keysA, keysB] = [
        new Set(Object.keys(a)),
        new Set(Object.keys(b))
    ];
    if (keysA.symmetricDifference(keysB).size > 0) {
        return false;
    }
    cache.set(a, b);
    for (const key of keysA) {
        if (!deepEquals(a[key], b[key], cache)) {
            return false;
        }
    }
    return true;
}
```

Problem 7: deep clone

Goal: Implement deepClone(value) to create an independent copy of a value.

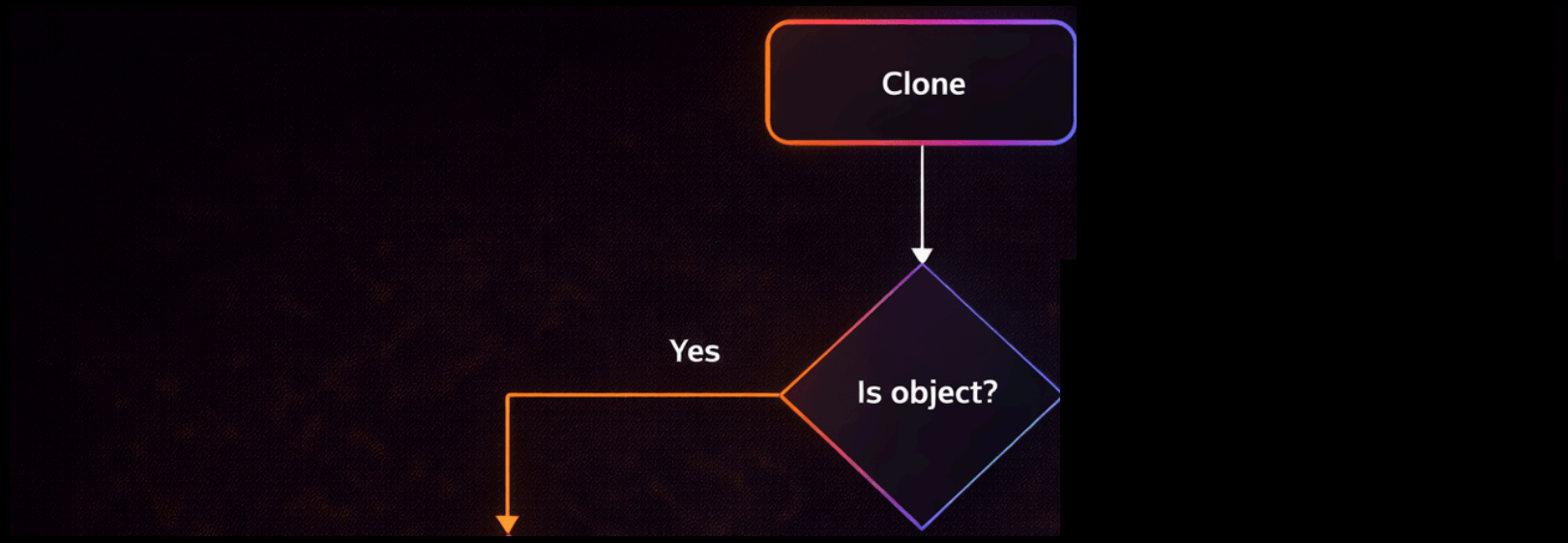
Requirements:

1. **Primitives:** Return as-is.
2. **Objects/Arrays:** New instances with cloned children.
3. **Special Types:** Map, Set, Date
4. **Circular References:** Critical here, or you get stack overflow.

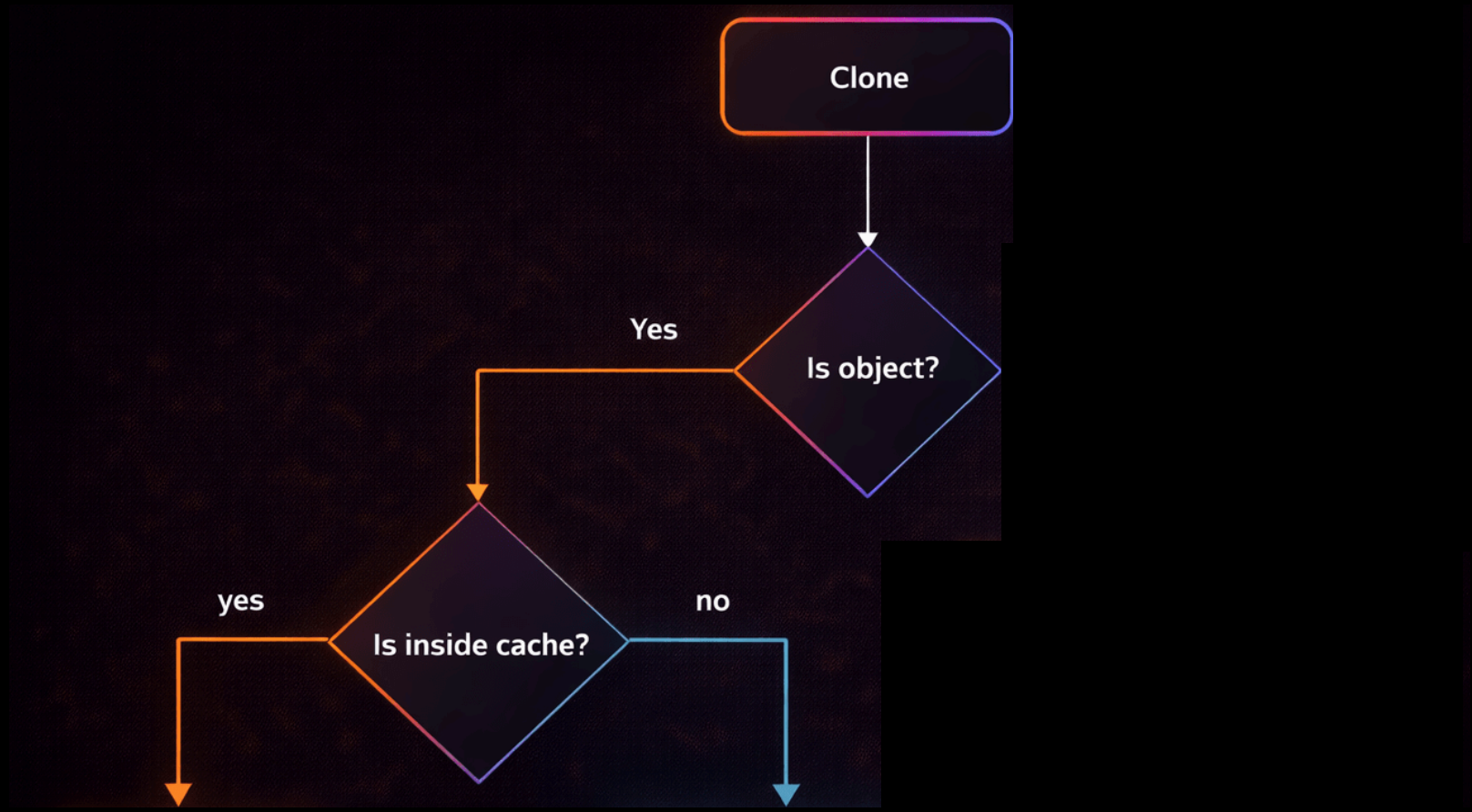
Level: Medium

Solving Deep clone

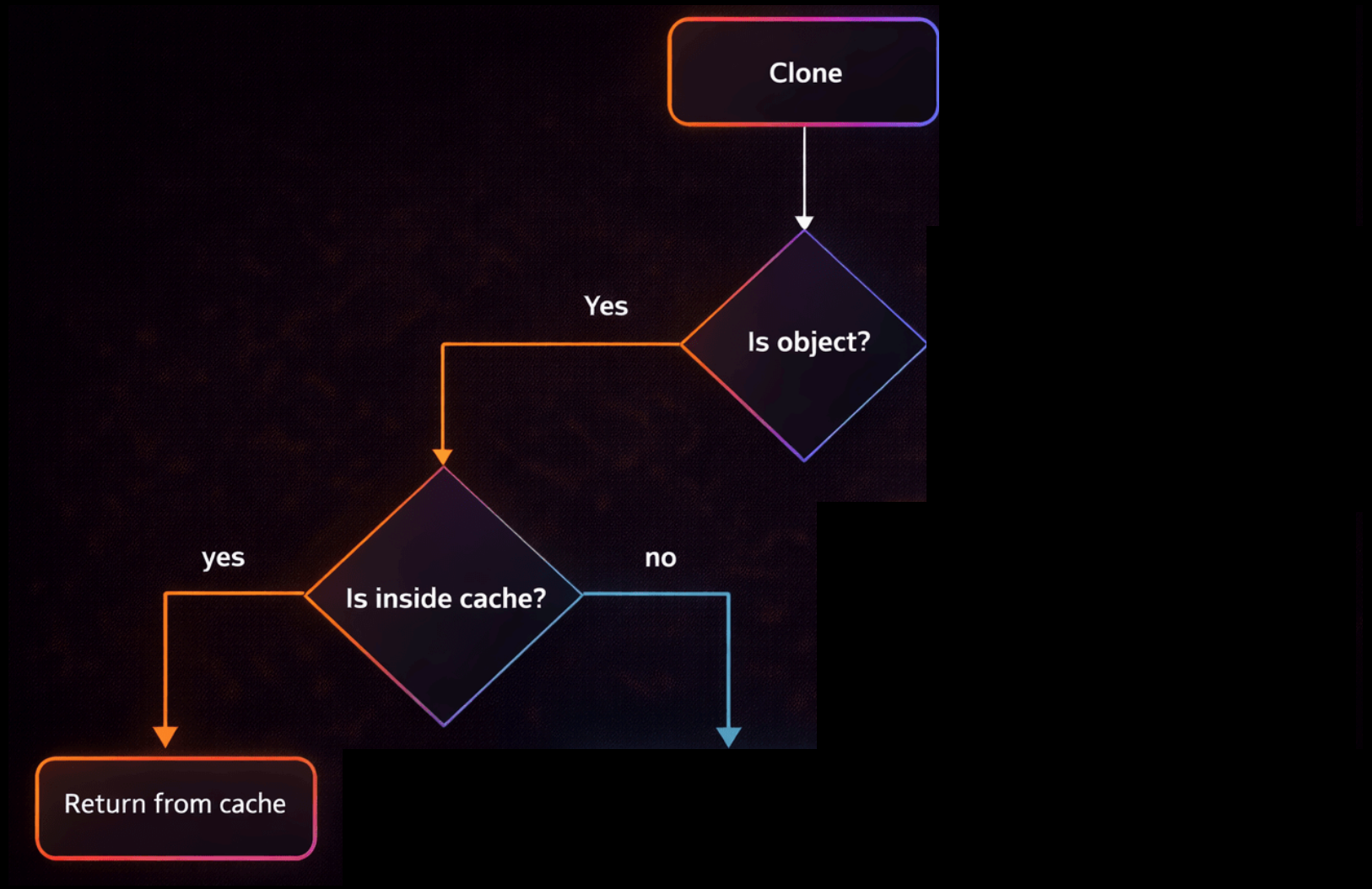
Solving Deep clone



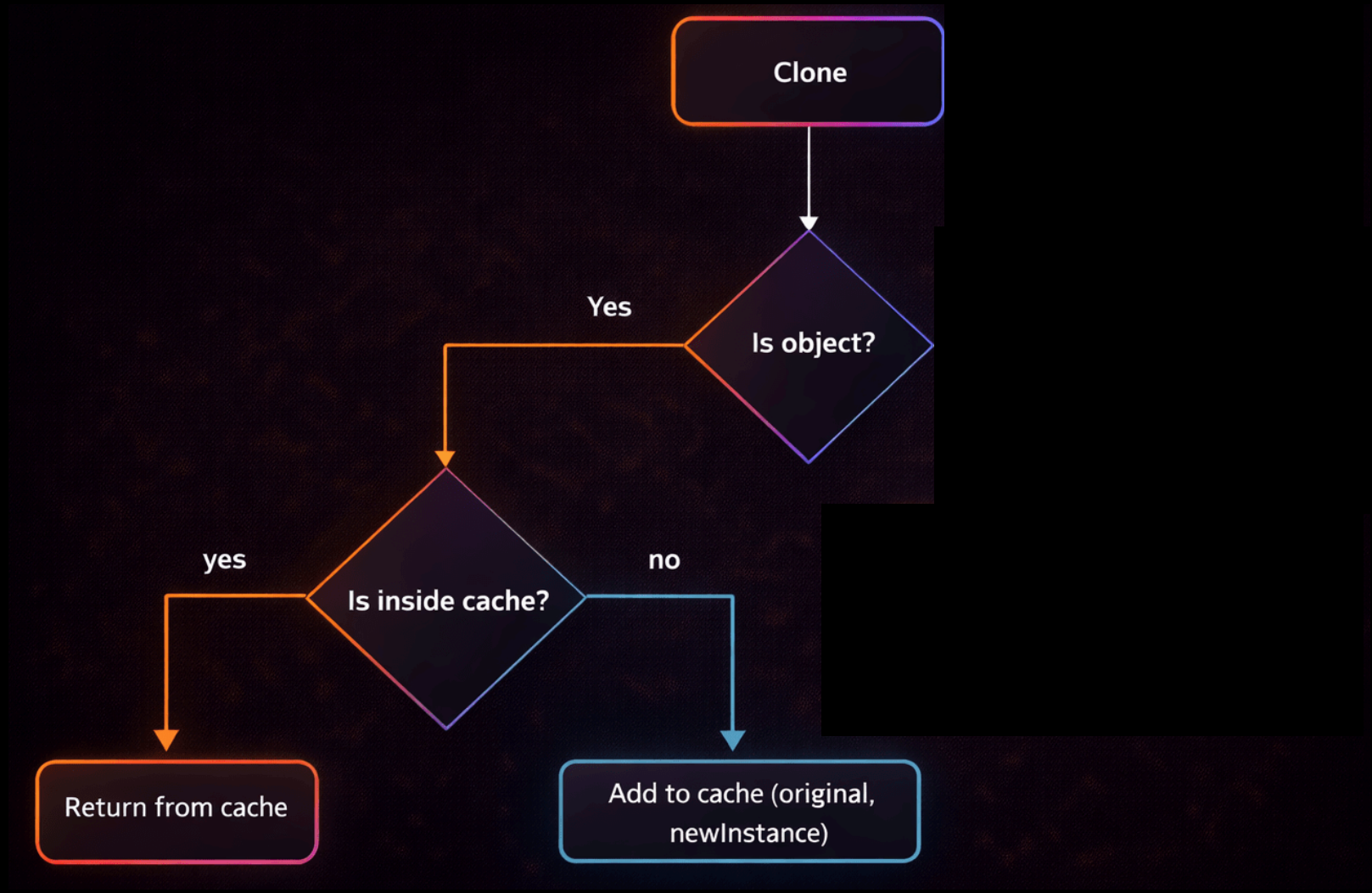
Solving Deep clone



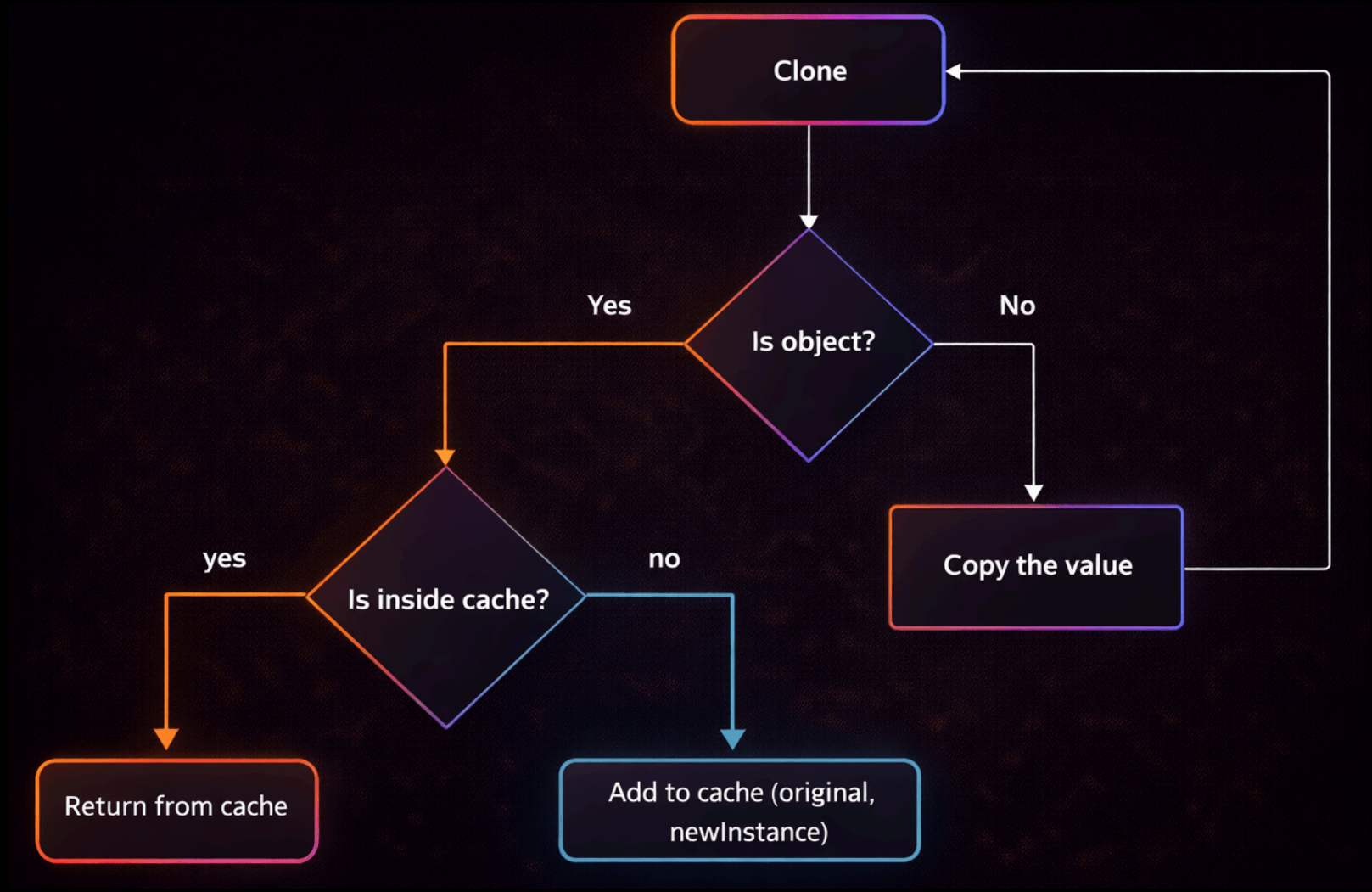
Solving Deep clone



Solving Deep clone



Solving Deep clone



Solution: Deep clone

```ts
type TCollection = Map<any, any> | Set<any> | Record<any, any> | Array<any>

export function deepClone<T>(value: T, cache = new Map()): T {
  const type = getType(value)
  if (cache.has(value)) {
    return cache.get(value)
  }
  switch (type) {
    case 'map':
    case 'set':
    case 'object':
    case 'array': {
      const target = createTarget(type)
      cache.set(value, target)
      each(value as TCollection, (val, key) => {
        addIntoTarget(target, deepClone(val, cache), key)
      })
      return target as T
    }
    case 'date':
      return new Date((value as Date).getTime()) as T
    default:
      return value as T
  }
}
```

Challenge 5: ReadOnly Challange

Implement the built-in Readonly<T> generic without using it. Constructs a type with all properties of T set to readonly, meaning the properties cannot be reassigned.

```
1 interface Todo {
2   title: string
3 }
4
5 const todo: MyReadonly<Todo> = { title: 'Hey' }
6 todo.title = 'Hello' // Error: cannot reassign a readonly property
```

Challenge 5: ReadOnly Challange | Solution

```
1 type MyReadonly<T> = {
2   readonly [P in keyof T]: T[P]
3 }
```

Challenge 5: Tuple to Object

Give an array, transform into an object type and the key/value must be in the given array.

```ts
const tuple = ['tesla', 'model 3', 'model X', 'model Y'] as const
type result = TupleToObject<typeof tuple>
  // {
  //   'tesla': 'tesla',
  //   'model 3': 'model 3',
  //   'model X': 'model X',
  //   'model Y':
  //   'model Y
  //   }
```

Challenge 5: Tuple to Object | Solution

Give an array, transform into an object type and the key/value must be in the given array.

```ts
type TupleToObject<T extends readonly (string | number | symbol)[]> = {
  [P in T[number]]: P
}
```

```ts
type TupleToObject<T extends readonly (string | number | symbol)[]> = {
  [P in (string | number | symbol)]: P
}
```

Problem 8: stringify

Goal:

Implement stringify(value) (like **JSON.stringify**), but with circular reference support.

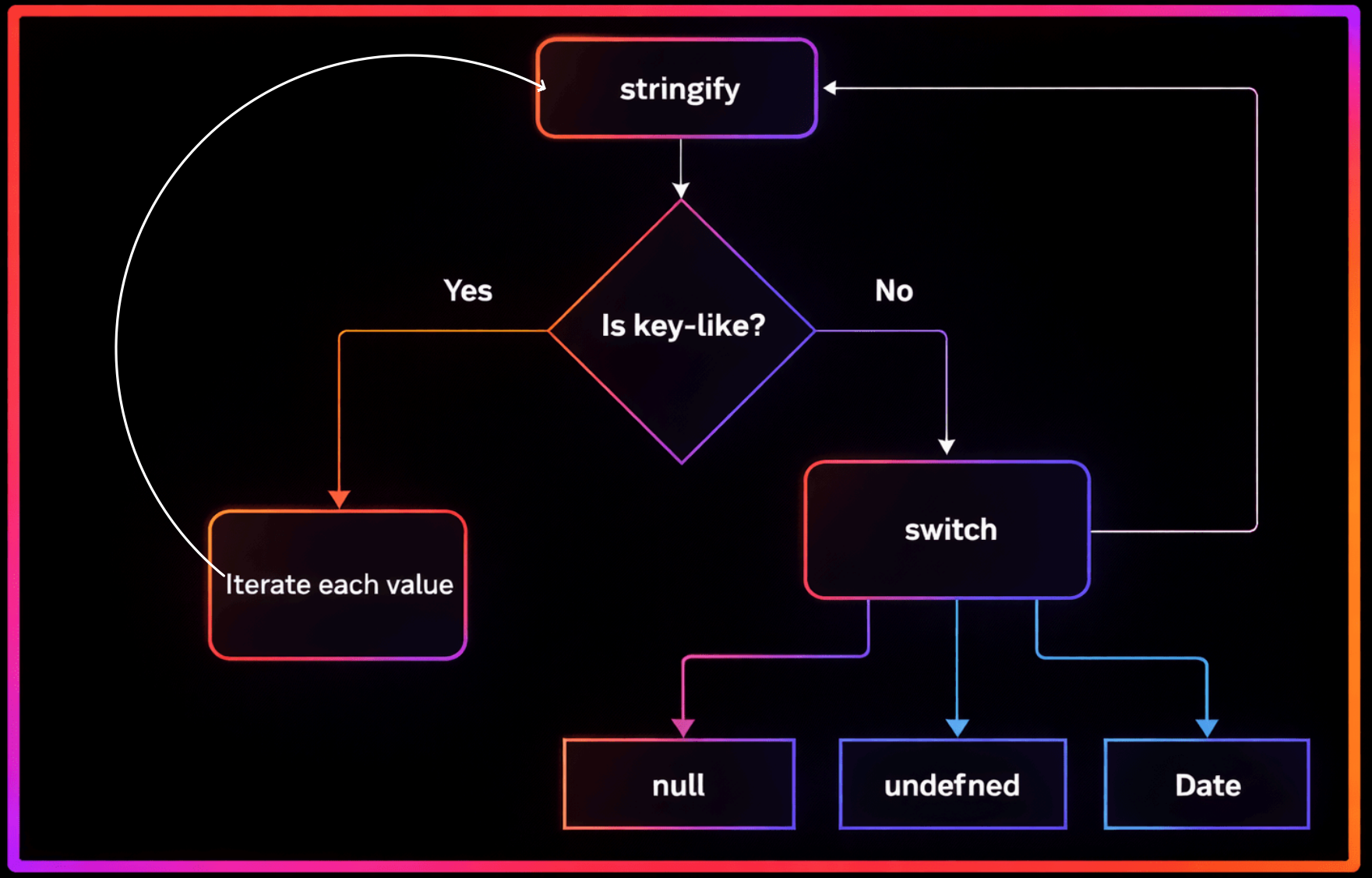
This is another classic **DFS-like** problem, similar to **deep-equals**. Once you're comfortable with such problems, it shouldn't be a problem for you to solve them.

Requirements:

1. null → "null"
2. Numbers → "42", "3.14"
3. Booleans → "true", "false"
4. Strings → '"hello"'
5. BigInt → "123"
6. Arrays/Sets: Format as [value1,value2,...]
7. Objects/Maps: Format as { key1: value1, key2: value2 }
8. Date: Use .toLocaleString()
9. RegExp: Use .toString()
10. Circular references: Return "[Circular]" marker

Level: Medium

Solving: stringify



Solution: stringify

```ts
export function stringify<T>(value: T, seen = new WeakSet()): string {
  const type = getType(value)
  switch (type) {
    case 'null':
    case 'number':
    case 'bigint':
    case 'boolean':
      return `${value}`
    case 'symbol':
      return `"${String(value)}"`
    case 'undefined':
    case 'string':
      return `"${value}"`
    case 'map':
    case 'object': {
      if (seen.has(value as object)) {
        return '[Circular]'
      }
      seen.add(value as object)
      const entries = Array.from(
        type === 'map' ? (value as Map<any, any>).entries() : Object.entries(value as {}),
      )
      const content = entries
        .map(([key, val]) => {
          return `${key}: ${stringify(val, seen)}`
        })
        .join(', ')
      return `{ ${content} }`
    }
    case 'set':
    case 'array': {
      if (seen.has(value as object)) {
        return '[Circular]'
      }
      seen.add(value as object)
      const content = Array.from(value as Array<any> | Set<any>)
        .map((val) => stringify(val, seen))
        .join(',')
      return `[${content}]`
    }
    case 'date':
      return (value as Date).toLocaleString()
    case 'regexp':
      return (value as RegExp).toString()
    default:
      return '"Unsupported Type"'
  }
}
```

5 min break

Problem 8: Promise

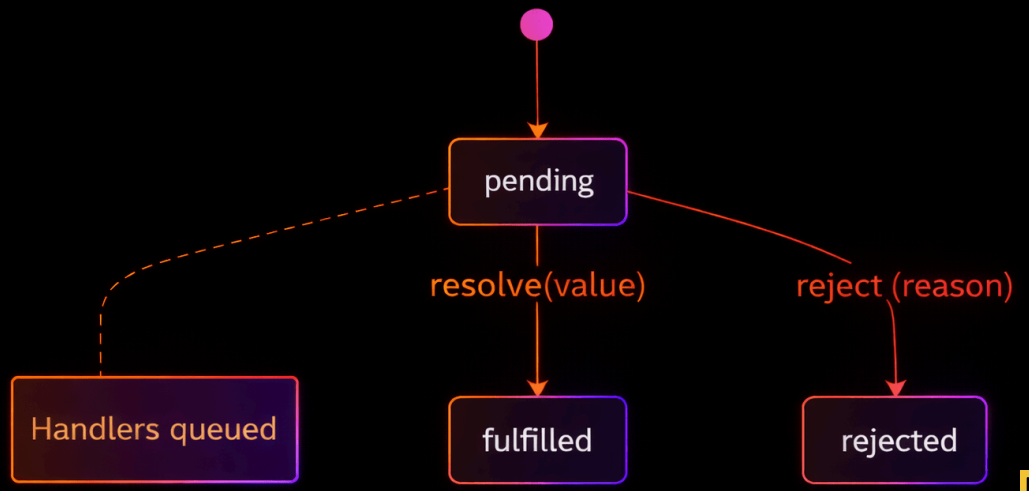
Goal: Implement a **Promise** class that covers base functionality:

1. then, catch, chaining

2. Static resolve / reject

Requirements:

1. **Three States:** pending, fulfilled, rejected.
2. **Async Execution:** Callbacks don't run immediately on resolve.



Level: Hard

Promise: How it works

new Promise(executor)

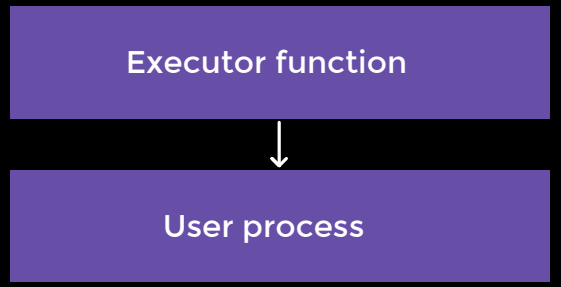
Promise: How it works

new Promise(executor)

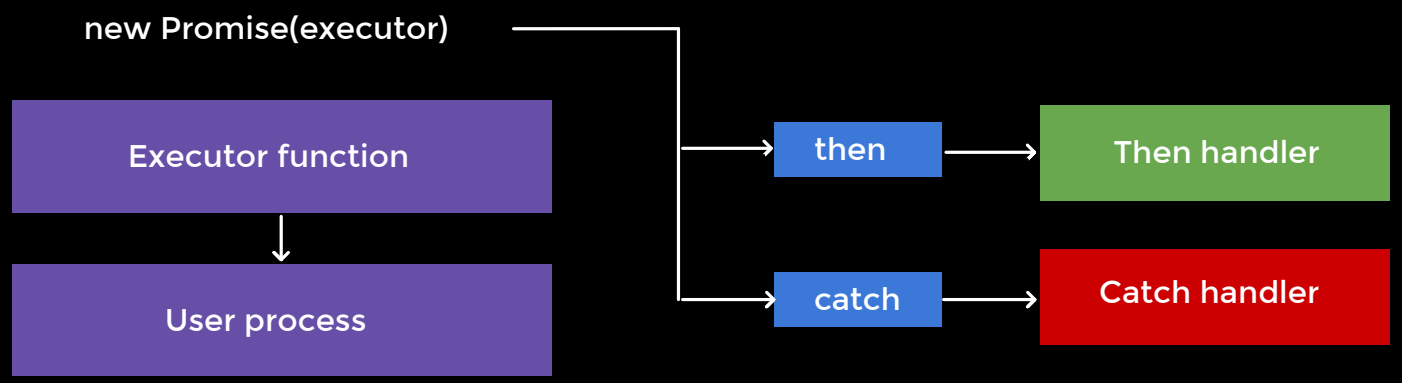
Executor function

Promise: How it works

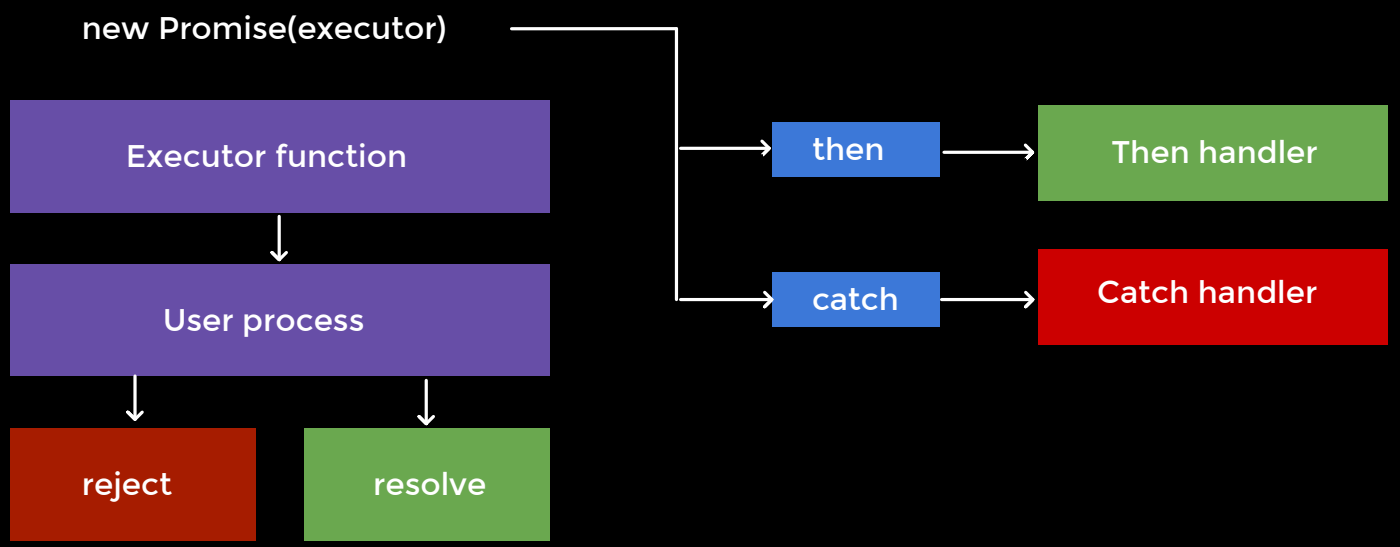
new Promise(executor)



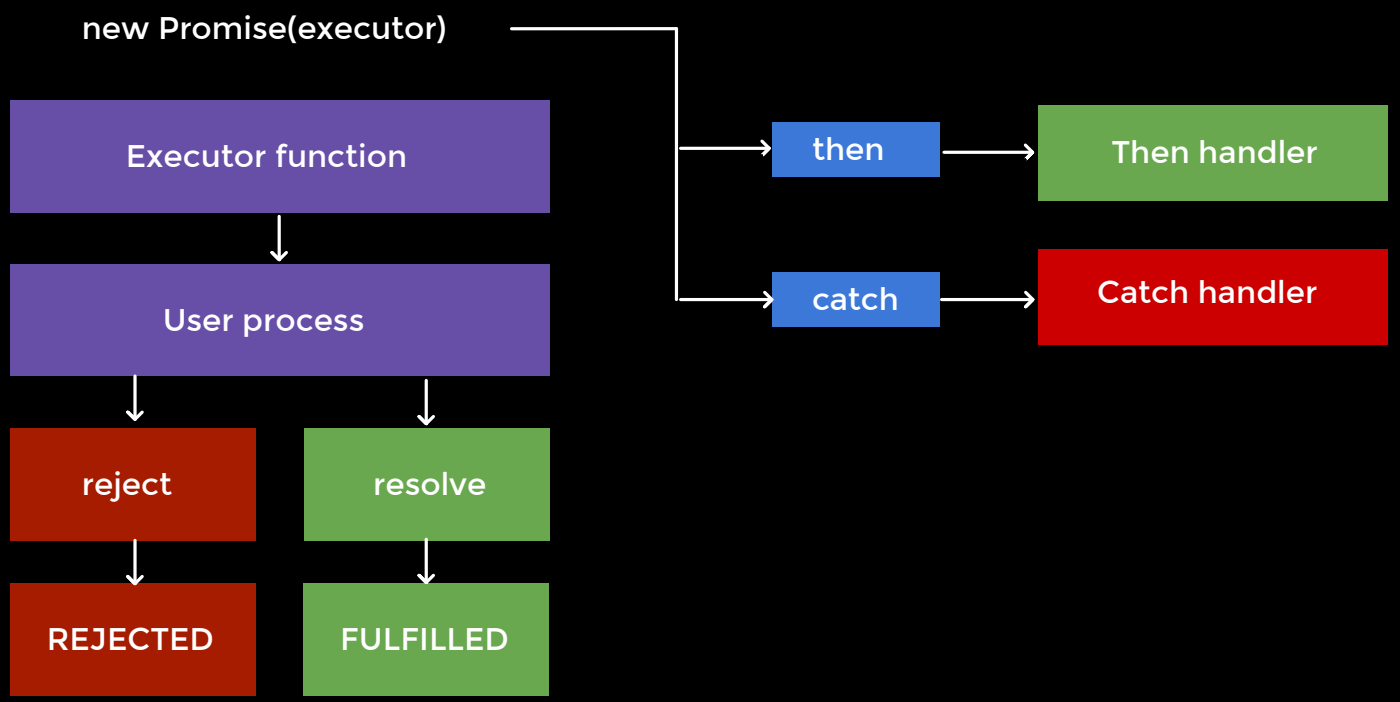
Promise: How it works



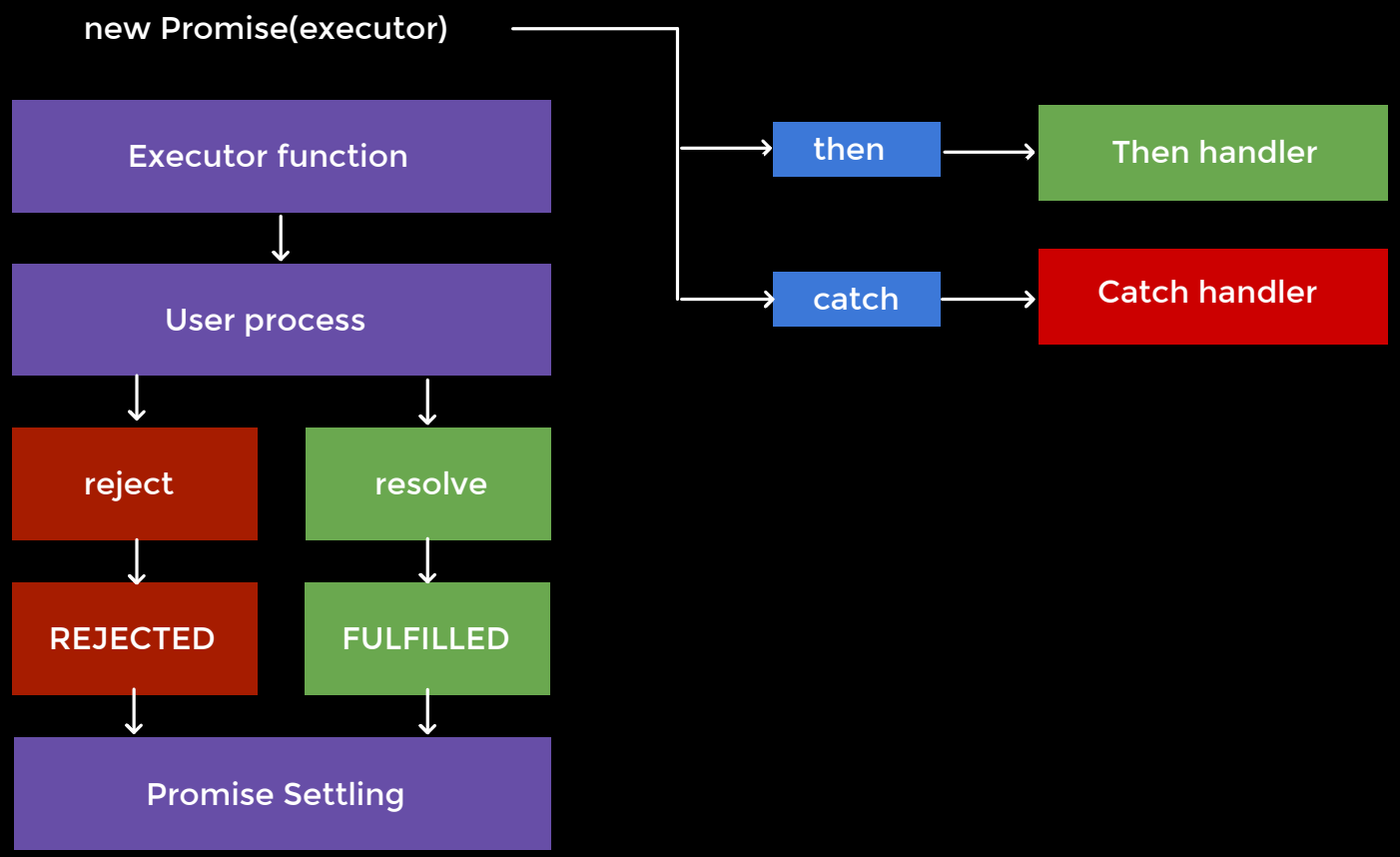
Promise: How it works



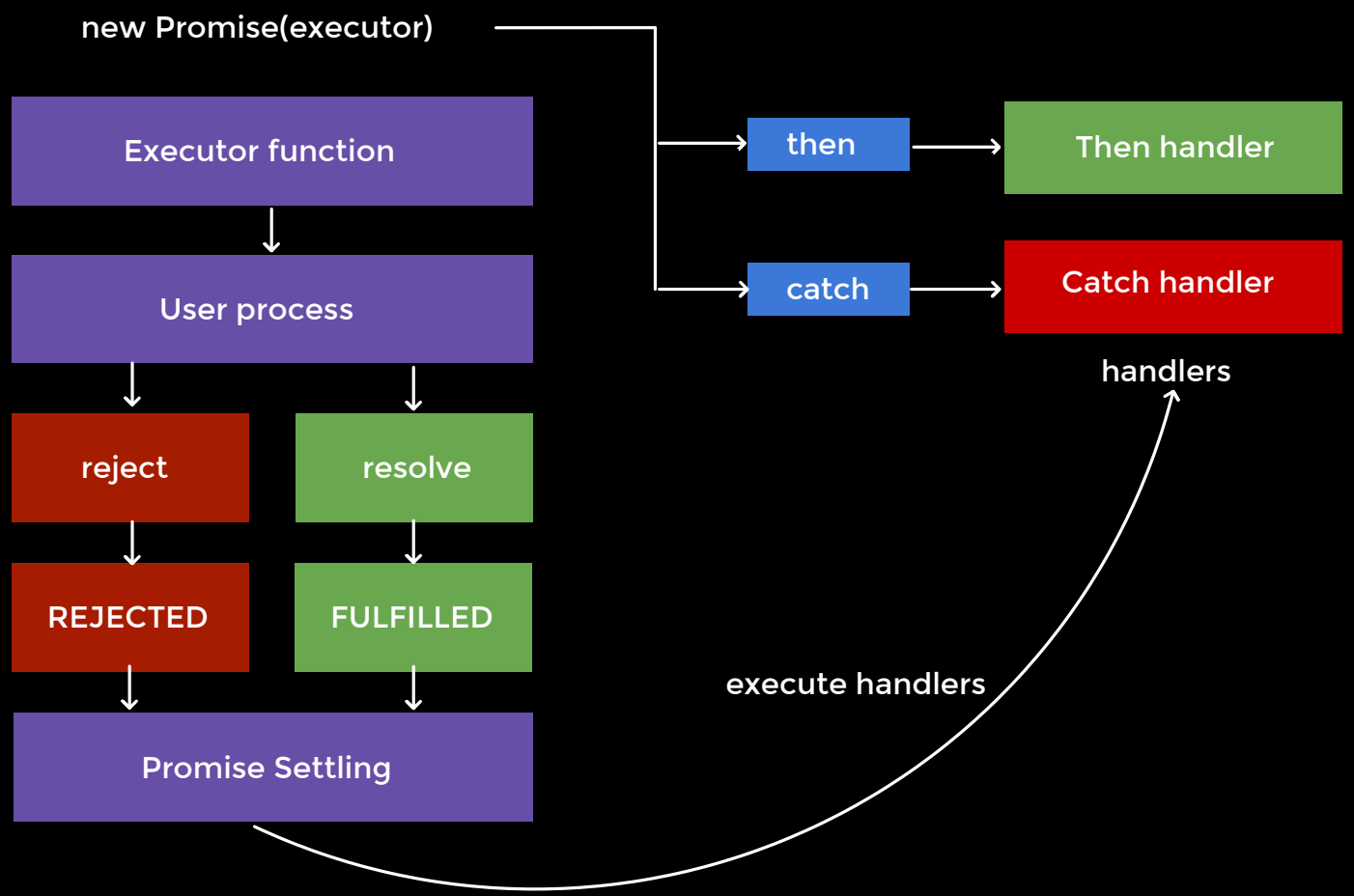
Promise: How it works



Promise: How it works



Promise: How it works



Ready to start?

Step 1: Basic types

```ts
type PromiseStatus = 'pending' | 'fulfilled' | 'rejected'

const PENDING: PromiseStatus = 'pending'
const FULFILLED: PromiseStatus = 'fulfilled'
const REJECTED: PromiseStatus = 'rejected'

type Executor<T> = (
  resolve: (value: T | PromiseLike<T>) => void,
  reject: (reason?: any) => void,
) => void

type OnFulfilled<T, R> = ((value: T) => R | PromiseLike<R>) | undefined | null
type OnRejected<R> = ((reason: any) => R | PromiseLike<R>) | undefined | null

type Handler<T> = {
  onFulfilled: (value: T) => any
  onRejected: (reason: any) => any
  resolve: (value: any) => void
  reject: (reason: any) => void
}
```

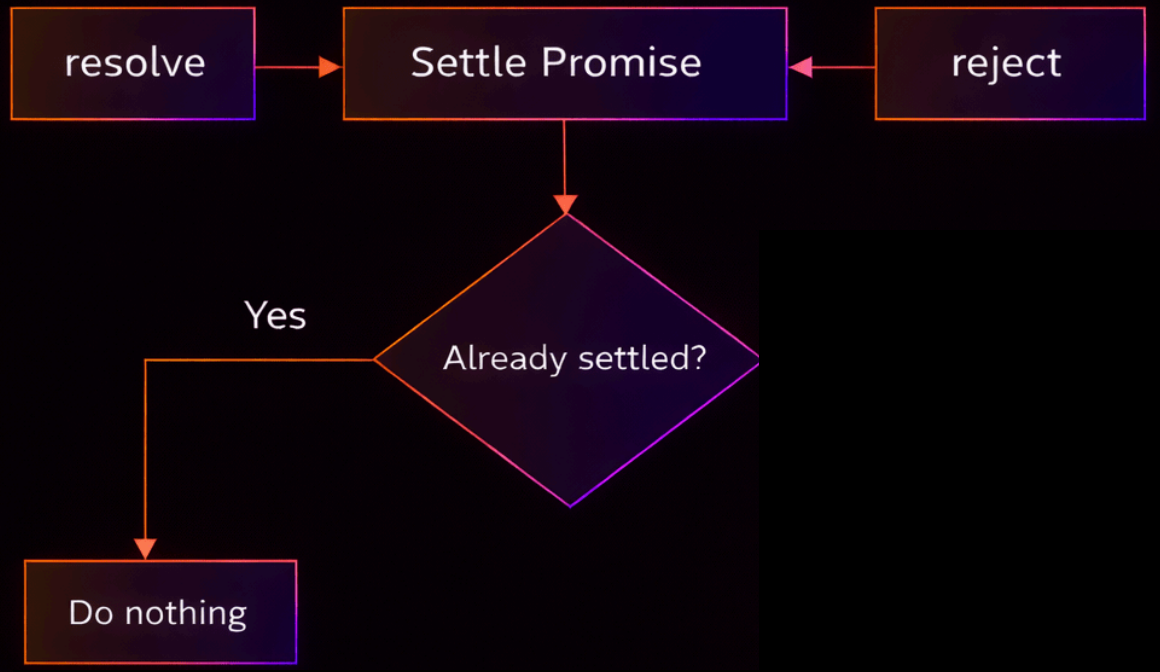
Text

Step 2: Class Fields

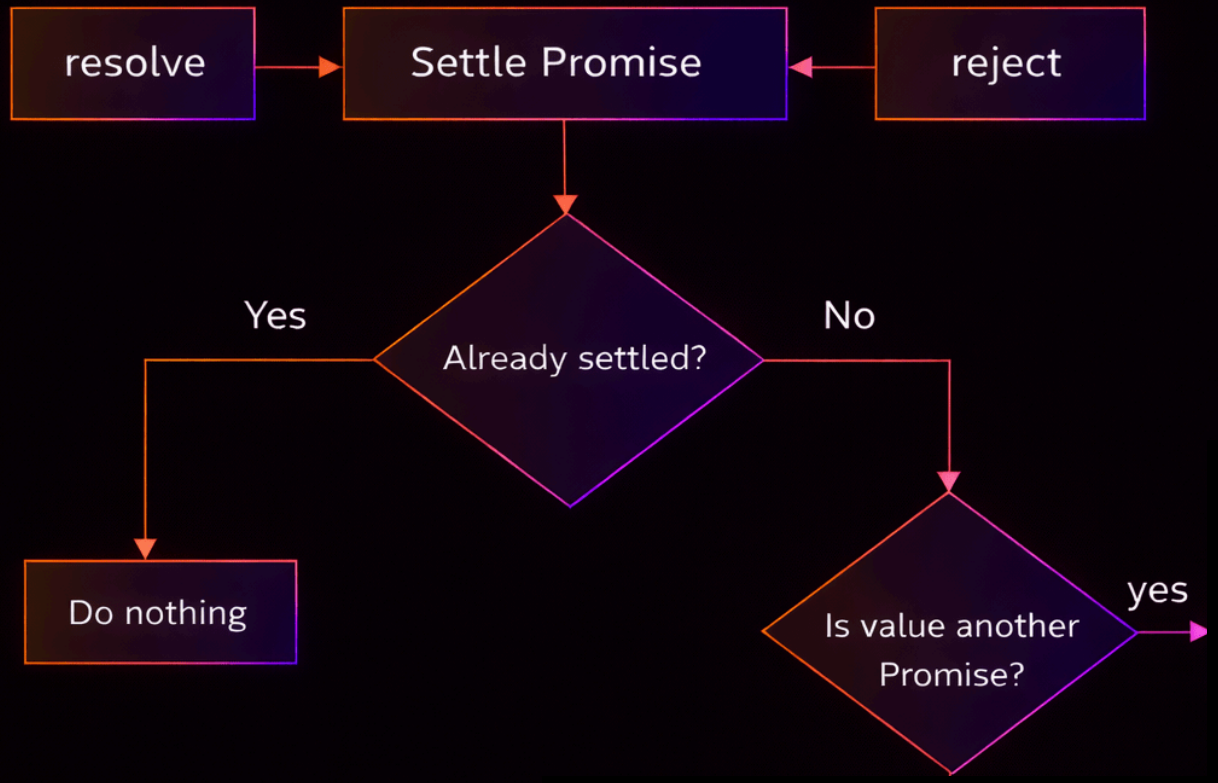
```
export class MyPromise<T = any> {
  #handlers: Handler<T>[] = []
  #status: PromiseStatus = PENDING
  #value: T | any
  #isResolved: boolean = false
}
```

Step 3: Settling logic

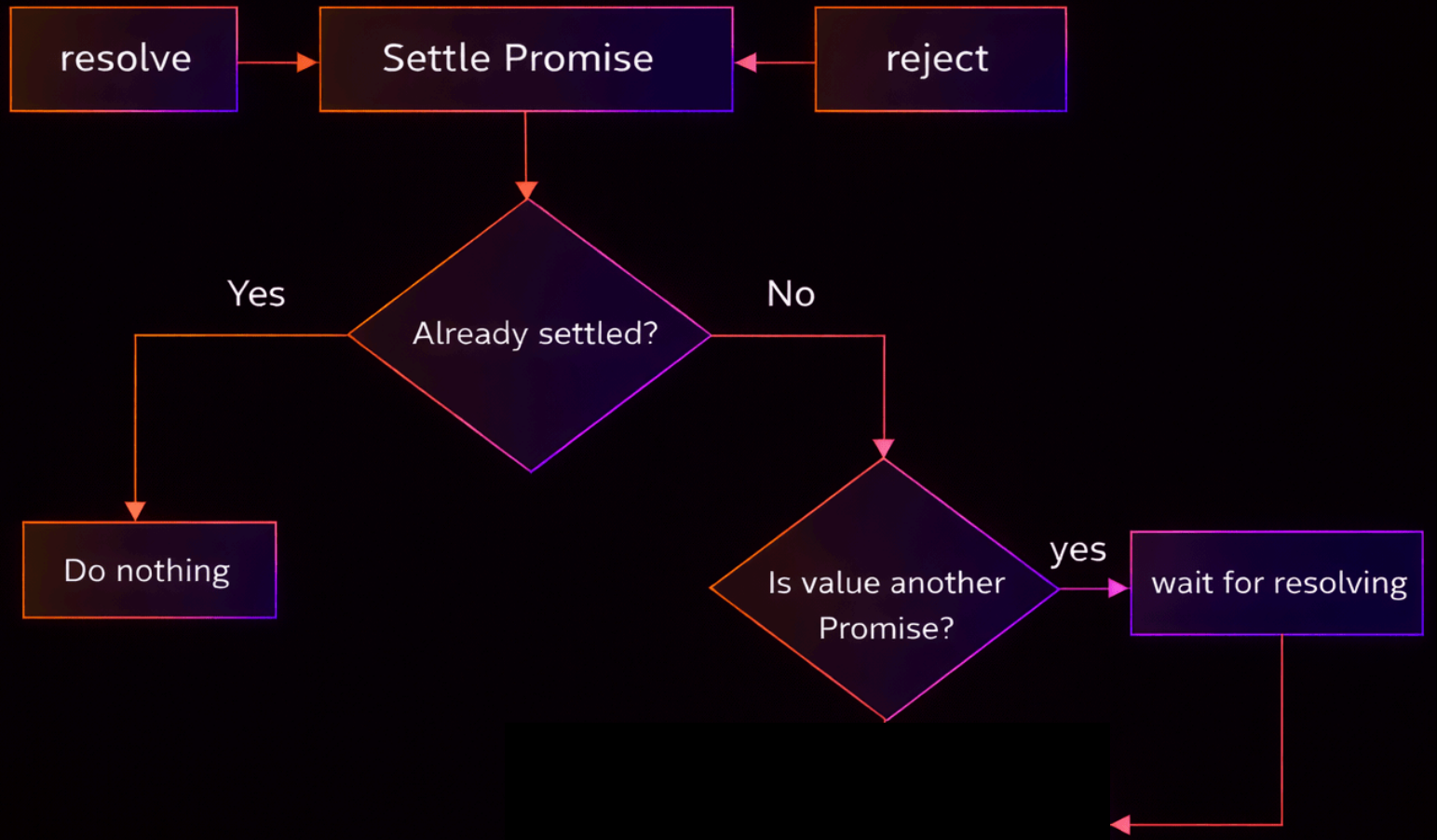
Step 3: Settling logic



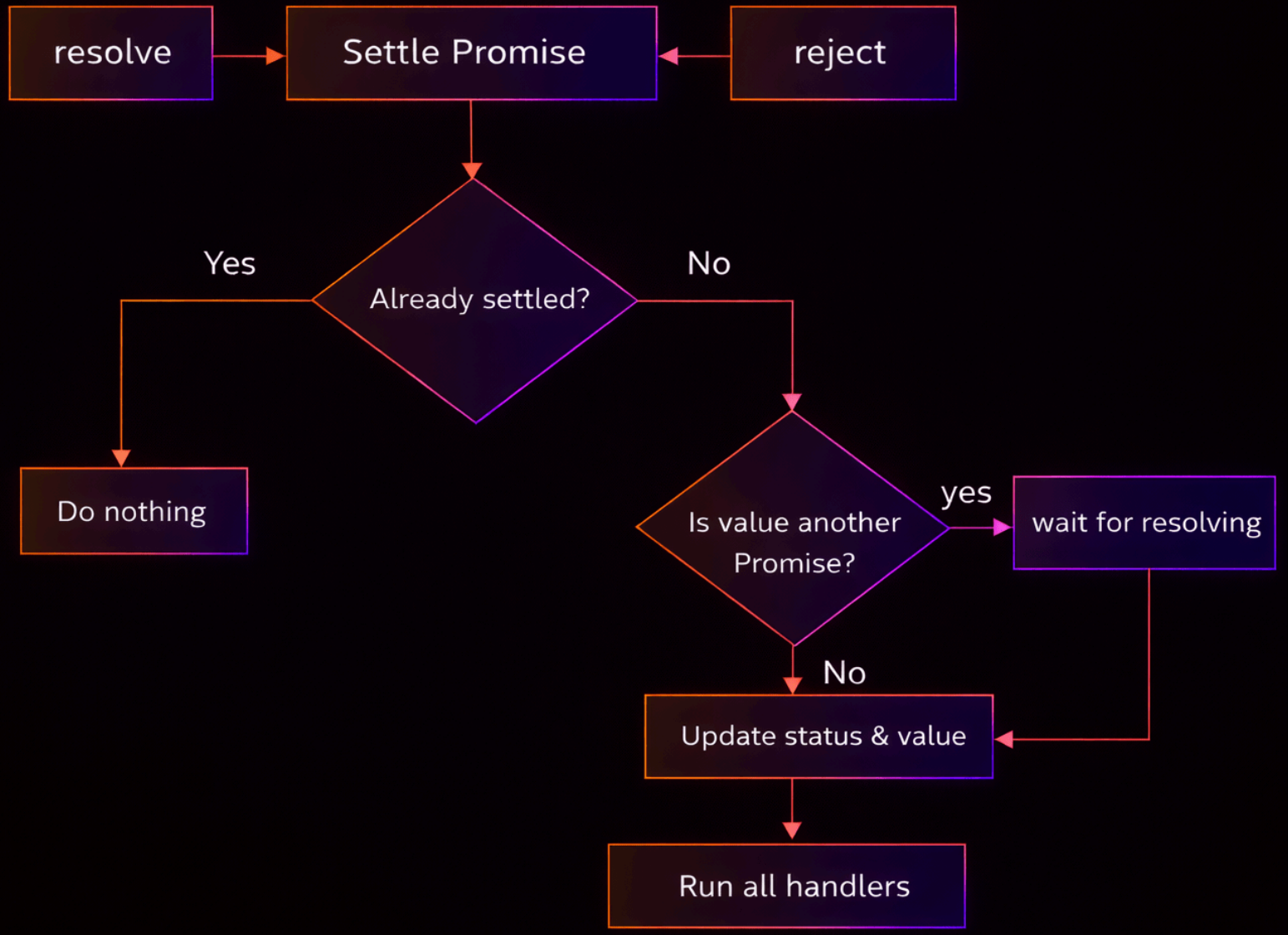
Step 3: Settling logic



Step 3: Settling logic



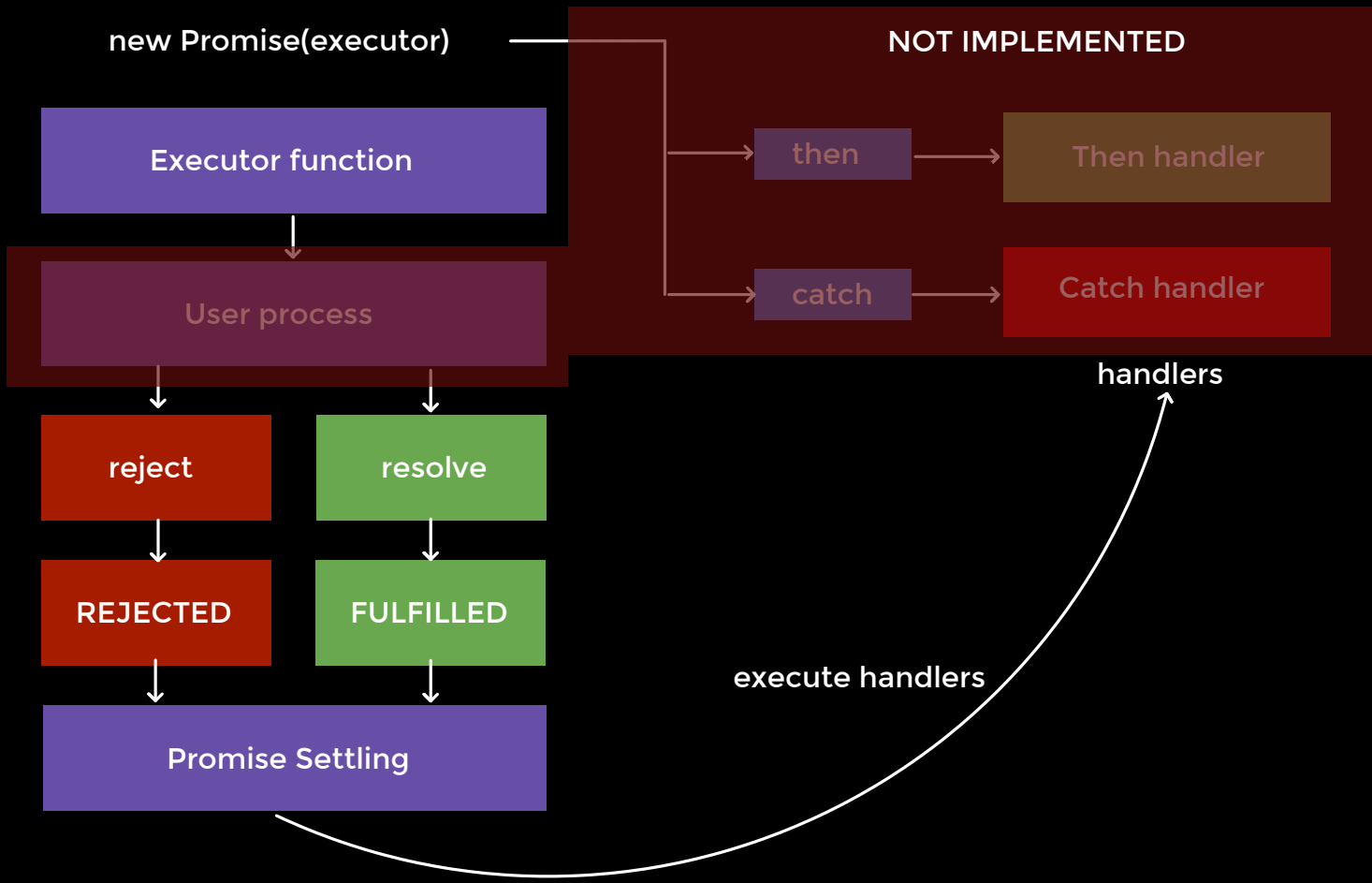
Step 3: Settling logic



Step 3: Settling logic

```
#settle = (v: T | any, status: PromiseStatus = FULFILLED): void => {
  if (this.#isResolved) return    // Can only settle ONCE
  this.#isResolved = true

  const update = (v: T | any): void => {
    this.#value = v
    this.#status = status
    this.#execute()               // Trigger queued handlers
  }

  // If resolving with another MyPromise, wait for it first
  if (v instanceof MyPromise && status === FULFILLED) {
    v.then(update)
  } else {
    update(v)
  }
}

#resolve = (v: T | PromiseLike<T>): void => void this.#settle(v, FULFILLED)
#reject = (e: any): void => void this.#settle(e, REJECTED)
```

Step 3: Settling logic



Step 4: Executor

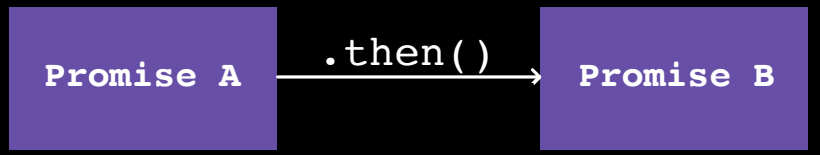
```
constructor(executor: Executor<T>) {
  try {
    executor(this.#resolve, this.#reject)
  } catch (e) {
    this.#reject(e)
  }
}
```

Step 5: then method

Promise A

```
1 const p = new MyPromise(
2     res => setTimeout(
3                       () => res(5), 50)
4                      ));
```

Step 5: then method




```
1 const p = new MyPromise(
2     res => setTimeout(
3                       () => res(5), 50)
4                      ));
5
6 p.then(v => v + 1)    // Promise A → creates Promise B
```

Step 5: then method


```
1 const p = new MyPromise(
2     res => setTimeout(
3                       () => res(5), 50)
4                      ));
5
6 p.then(v => v + 1)    // Promise A → creates Promise B
7  .then(v => v * 2)    // Promise B → creates Promise C
```

Step 5: then method


```
1 const p = new MyPromise(
2     res => setTimeout(
3                       () => res(5), 50)
4                      ));
5
6 p.then(v => v + 1)    // Promise A → creates Promise B
7  .then(v => v * 2)    // Promise B → creates Promise C
```

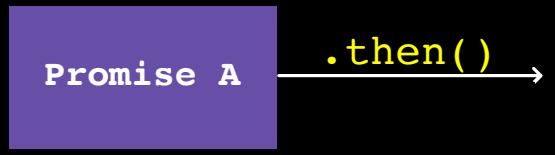
Step 5: then method


```
1 const p = new MyPromise(
2     res => setTimeout(
3                       () => res(5), 50)
4                      ));
5
6 p.then(v => v + 1)    // Promise A → creates Promise B
7  .then(v => v * 2)    // Promise B → creates Promise C
```

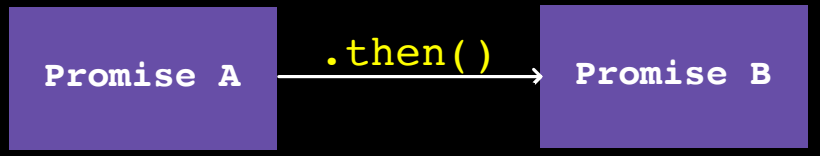
Step 5: then method


```
1 const p = new MyPromise(
2     res => setTimeout(
3                       () => res(5), 50)
4                      ));
5
6 p.then(v => v + 1)    // Promise A → creates Promise B
7  .then(v => v * 2)    // Promise B → creates Promise C
```

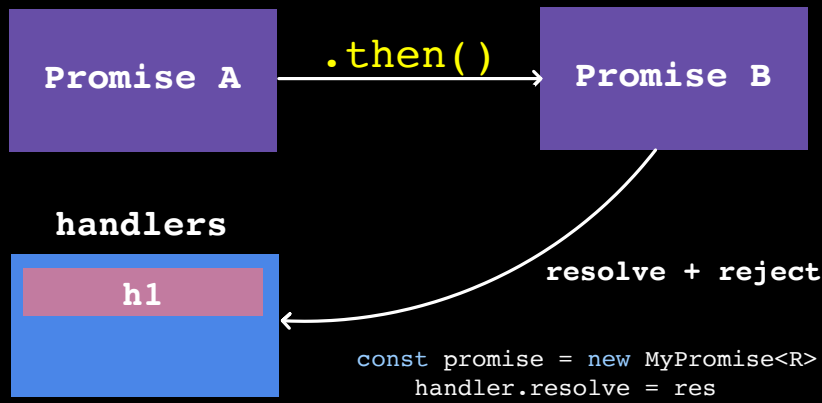
Step 5: then method



Step 5: then method

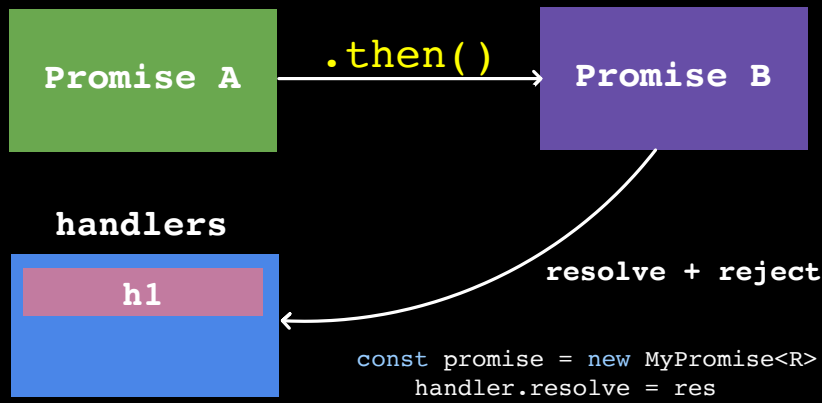


Step 5: then method



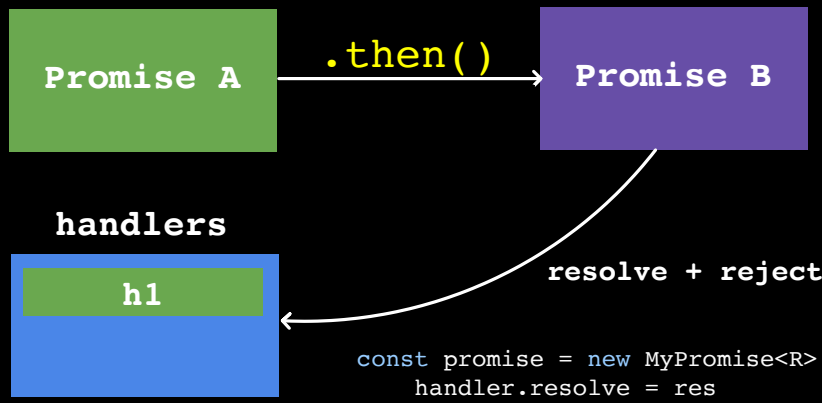

```
const promise = new MyPromise<R>((res, rej) => {
    handler.resolve = res
    handler.reject = rej
})
this.#handlers.push(handler)
```

Step 5: then method



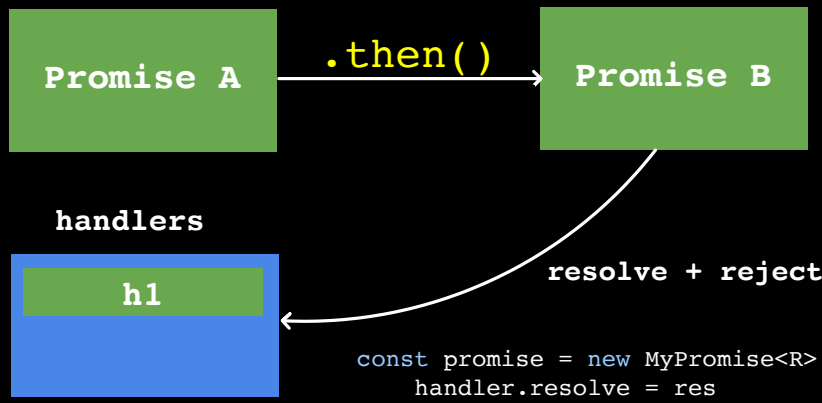

```
const promise = new MyPromise<R>((res, rej) => {
    handler.resolve = res
    handler.reject = rej
})
this.#handlers.push(handler)
```

Step 5: then method



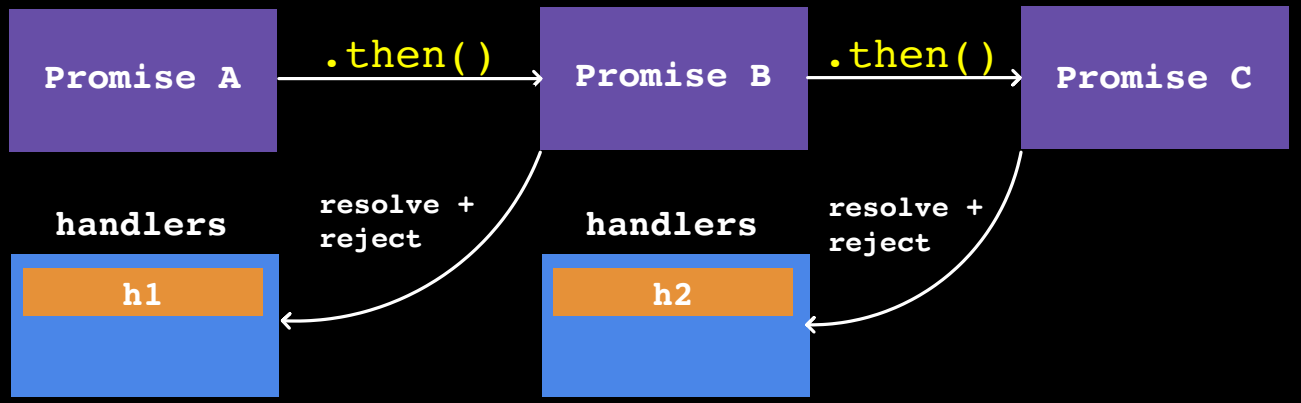

```
const promise = new MyPromise<R>((res, rej) => {
    handler.resolve = res
    handler.reject = rej
})
this.#handlers.push(handler)
```

Step 5: then method

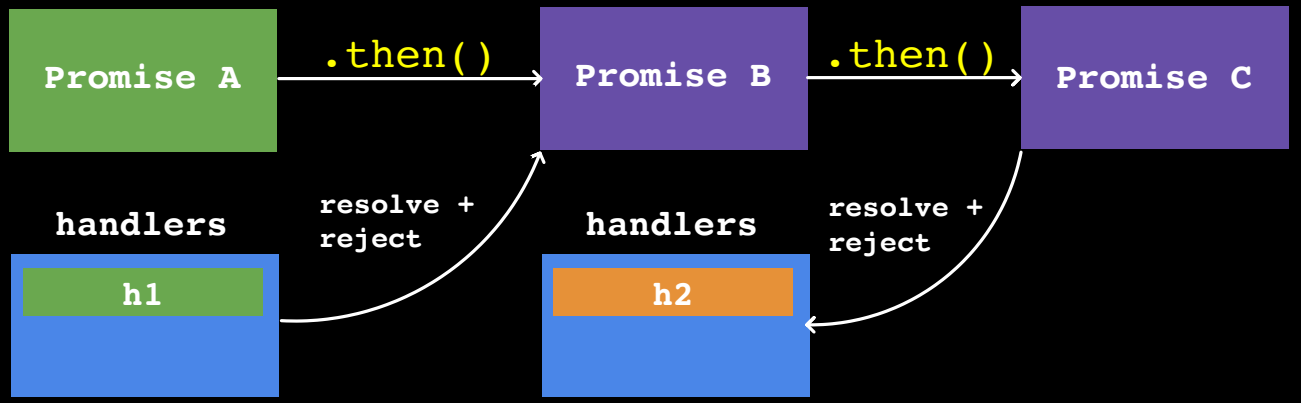



```
const promise = new MyPromise<R>((res, rej) => {
    handler.resolve = res
    handler.reject = rej
})
this.#handlers.push(handler)
```

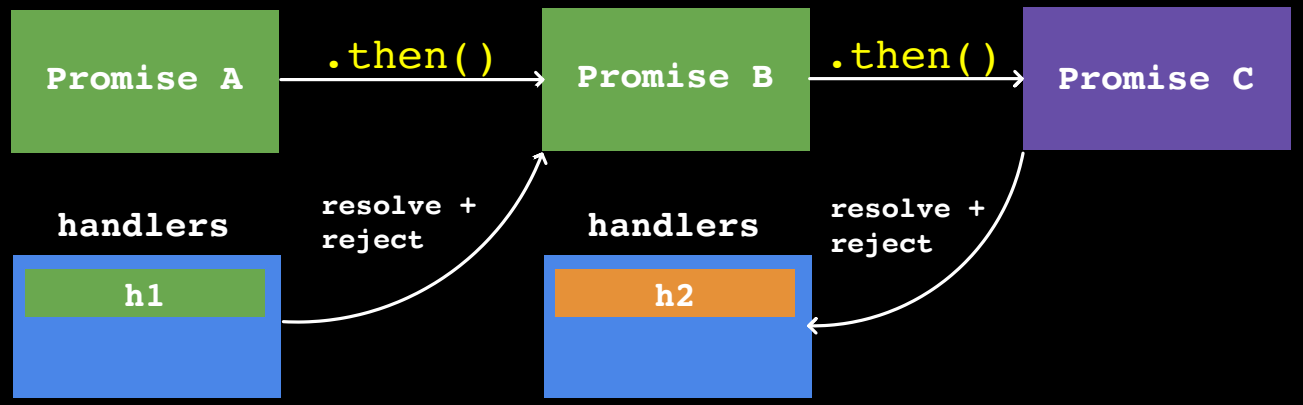
Step 5: then method



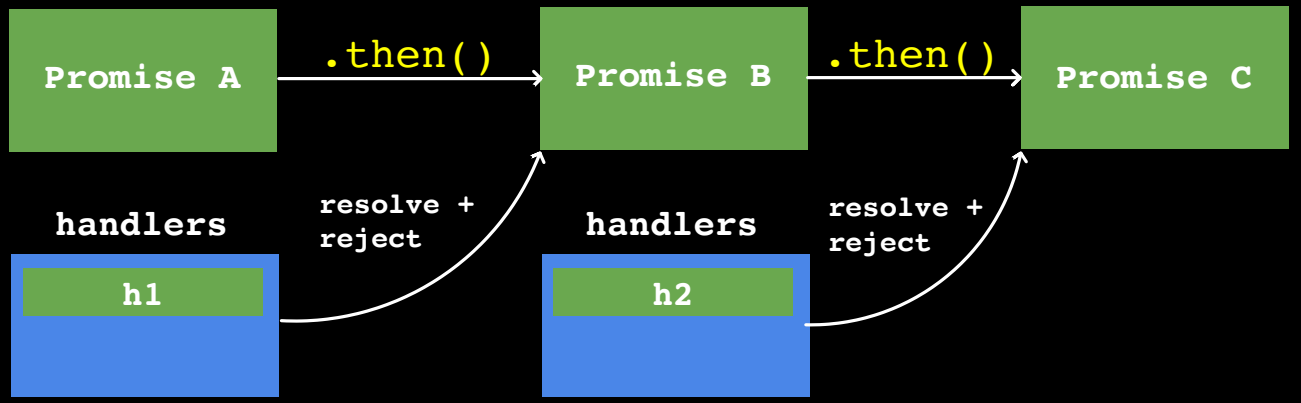
Step 5: then method



Step 5: then method



Step 5: then method



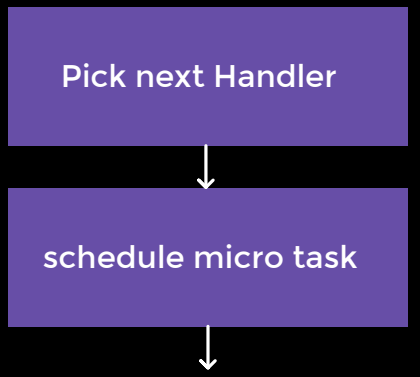
Step 5: then method

```
then<R = T>(onFulfilled?: OnFulfilled<T, R>, onRejected?: OnRejected<R>): MyPromise<R> {
  const handler: Handler<T> = {
    onFulfilled: typeof onFulfilled === 'function' ? onFulfilled : (v: T) => v as any,
    onRejected:
      typeof onRejected === 'function'
        ? onRejected
        : (err: any) => {
            throw err
          },
    resolve: () => {},
    reject: () => {},
  }

  const promise = new MyPromise<R>((res, rej) => {
    handler.resolve = res
    handler.reject = rej
  })

  this.#handlers.push(handler)

  if (this.#status !== PENDING) {
    this.#execute()
  }

  return promise
}
```

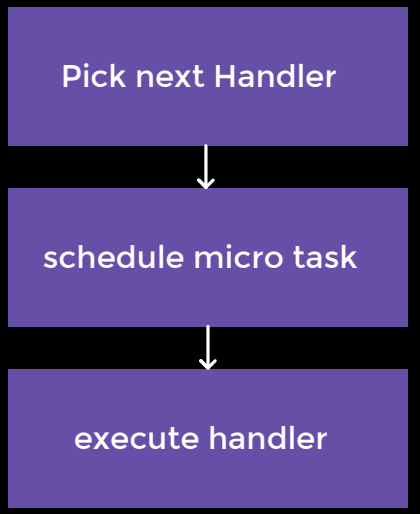
Step 6: handler execution

Pick next Handler

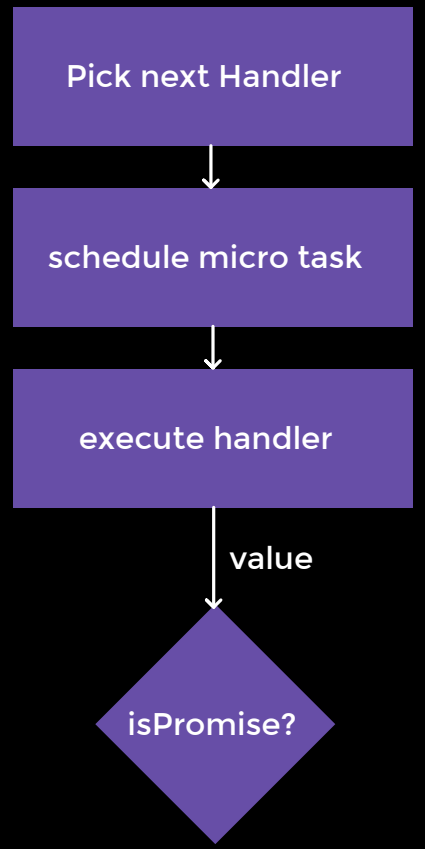
Step 6: handler execution



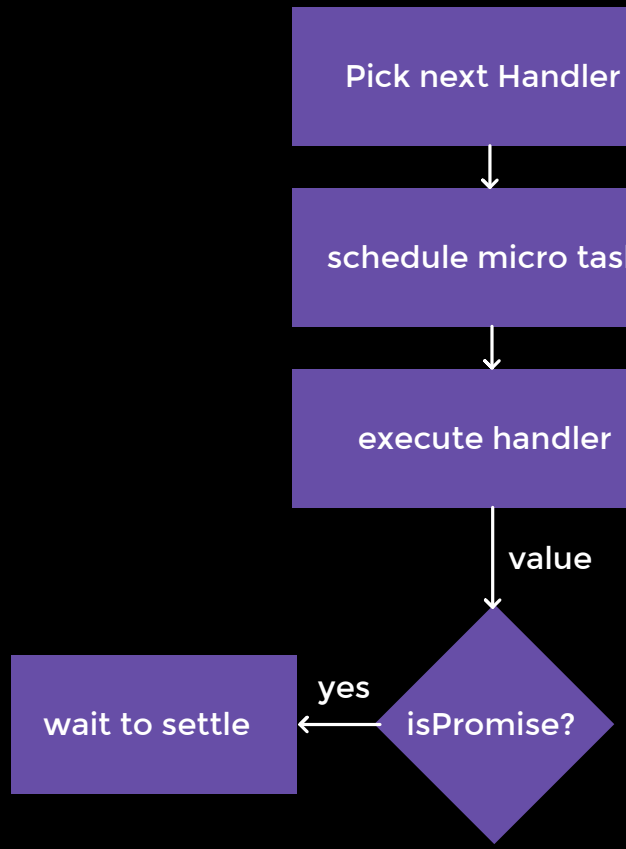
Step 6: handler execution



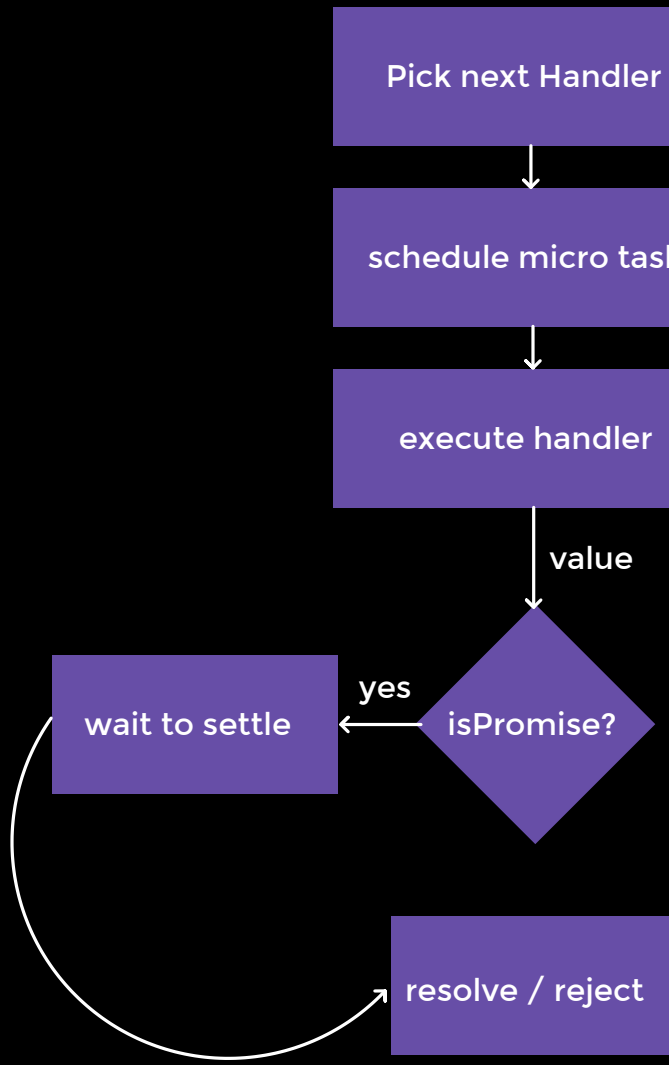
Step 6: handler execution



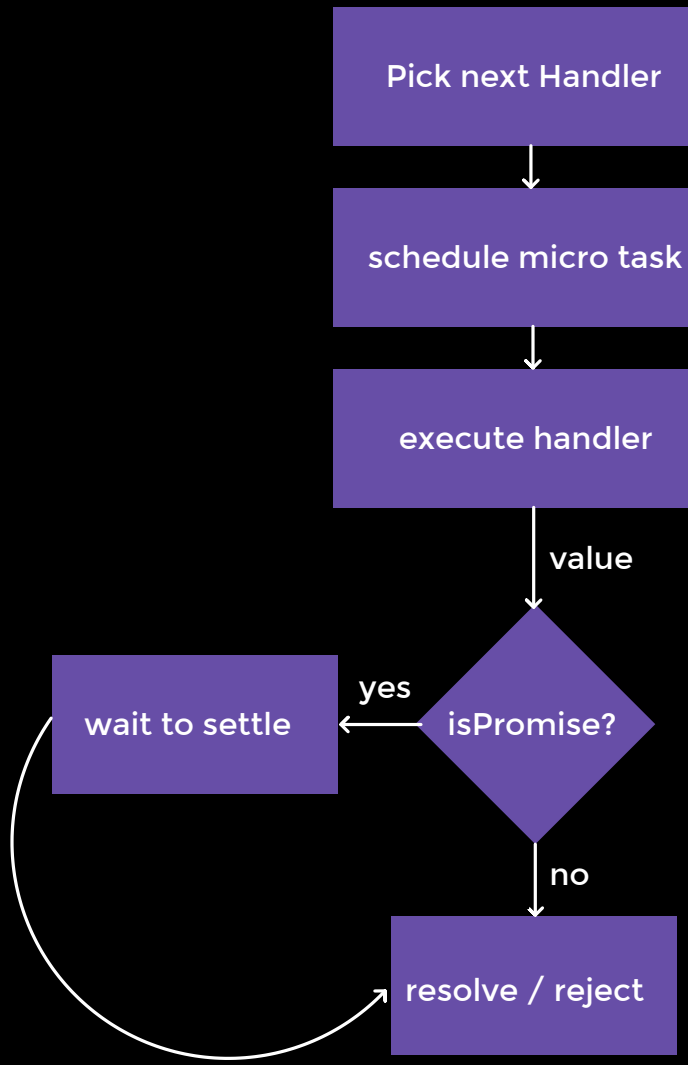
Step 6: handler execution



Step 6: handler execution



Step 6: handler execution



Step 6: handler execution

```
#execute = (): void => {
  const handlers = this.#handlers
  for (const { onFulfilled, onRejected, resolve, reject } of handlers) {
    const handler = this.#status === FULFILLED ? onFulfilled : onRejected
    queueMicrotask(() => {
      try {
        const result = handler(this.#value)
        if (result instanceof MyPromise) {
          result.then(resolve, reject)
        } else {
          resolve(result)
        }
      } catch (e) {
        reject(e)
      }
    })
  }
  this.#handlers = []
}
```

Step 7: Statics

```
catch<R = never>(onRejected?: OnRejected<R>): MyPromise<T | R> {
  return this.then<T | R>(undefined, onRejected)
}

static resolve<T>(value: T): MyPromise<T> {
  return new MyPromise<T>((res) => res(value))
}

static reject<T = never>(value: any): MyPromise<T> {
  return new MyPromise<T>((_, rej) => rej(value))
}
```

Let's run tests

```
✓ Reference Solution > MyPromise > structure > MyPromise.prototype.then should exist
✓ Reference Solution > MyPromise > structure > then() should return a new Promise
✓ Reference Solution > MyPromise > structure > catch() should exist
✓ Reference Solution > MyPromise > structure > catch() should return a new Promise
✓ Reference Solution > MyPromise > constructor > new MyPromise(() => {})
✓ Reference Solution > MyPromise > settlement > Promise could only be settled once
✓ Reference Solution > MyPromise > settlement > Promise could only be resolved once
✓ Reference Solution > MyPromise > settlement > Promise could only be rejected once
✓ Reference Solution > MyPromise > settlement > Promise could only be resolved or rejected once
✓ Reference Solution > MyPromise > then() > then(onFulfilled)
✓ Reference Solution > MyPromise > then() > then(onFulfilled, onRejected)
✓ Reference Solution > MyPromise > then() > then(onFulfilled, onRejected) rejection handler should swallow the rejection
✓ Reference Solution > MyPromise > catch() > MyPromise.prototype.catch should work
✓ Reference Solution > MyPromise > catch() > catch() should swallow error and works like then()
✓ Reference Solution > MyPromise > chaining > support chaining then().then().then()
✓ Reference Solution > MyPromise > chaining > multiple then()
✓ Reference Solution > MyPromise > chaining > then() returns a resolved promise if callback returns value
✓ Reference Solution > MyPromise > chaining > then() returns a resolved promise of undefined if callback returns nothing
✓ Reference Solution > MyPromise > chaining > then(callback) return a promise chained to the promise(fulfilled) from callback
✓ Reference Solution > MyPromise > chaining > then(callback) return a promise chained to the promise(rejected) from callback
✓ Reference Solution > MyPromise > error handling > error in constructor should reject
✓ Reference Solution > MyPromise > error handling > error in fulfill handler should reject
✓ Reference Solution > MyPromise > async behavior > constructor should be sync, then handlers should be async
✓ Reference Solution > MyPromise > static methods > MyPromise.resolve(1)
✓ Reference Solution > MyPromise > static methods > MyPromise.reject(1)
```

First hard problem solved!

Challenge 6: Append to Object

Implement a type that appends a new field to the interface.
The type takes three arguments. The output should be an object with the new field.

```
type Test = { id: '1' }
type Result = AppendToObject<Test, 'value', 4>
// { id: '1', value: 4 }
```

Challenge 6: Append to Object | Solution

Implement a type that appends a new field to the interface.
The type takes three arguments. The output should be an object with the new field.

```
type AppendToObject<T extends object, U extends string, V> = {
  [P in keyof T | U]: P extends keyof T ? T[P] : V
}
```

Challenge 7: Merge

Merge two types into a new type. Keys of the second type overrides the first type.

```ts
type Foo = { a: number; b: string }
type Bar = { b: number; c: boolean }

type Result = Merge<Foo, Bar>
// { a: number; b: number; c: boolean }
```

Challenge 7: Merge | Solution

Merge two types into a new type. Keys of the second type overrides the first type.

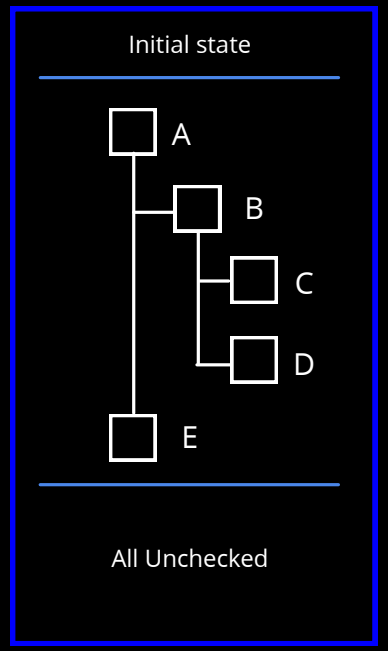
```
type Merge<F, S> = {
  [P in keyof F | keyof S]: P extends keyof S
                              ? S[P]
                              : P extends keyof F
                                  ? F[P]
                                  : never
}
```

Problem 9: Tree Select

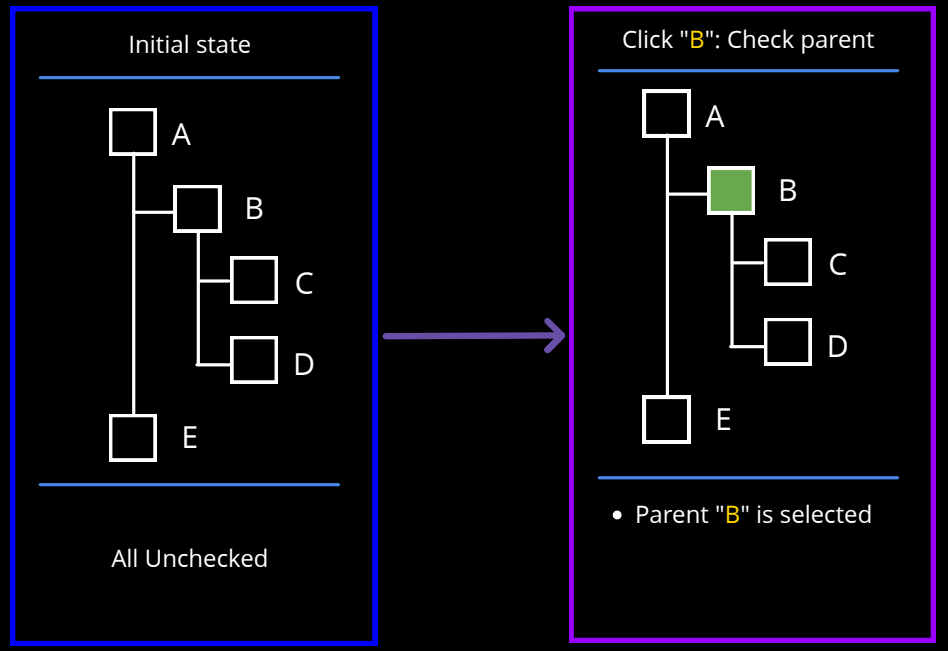
Implement a tree selection system where clicking a node selects/deselects it and all its children, while parent nodes show partial selection state when some (but not all) children are selected.

Level: Hard

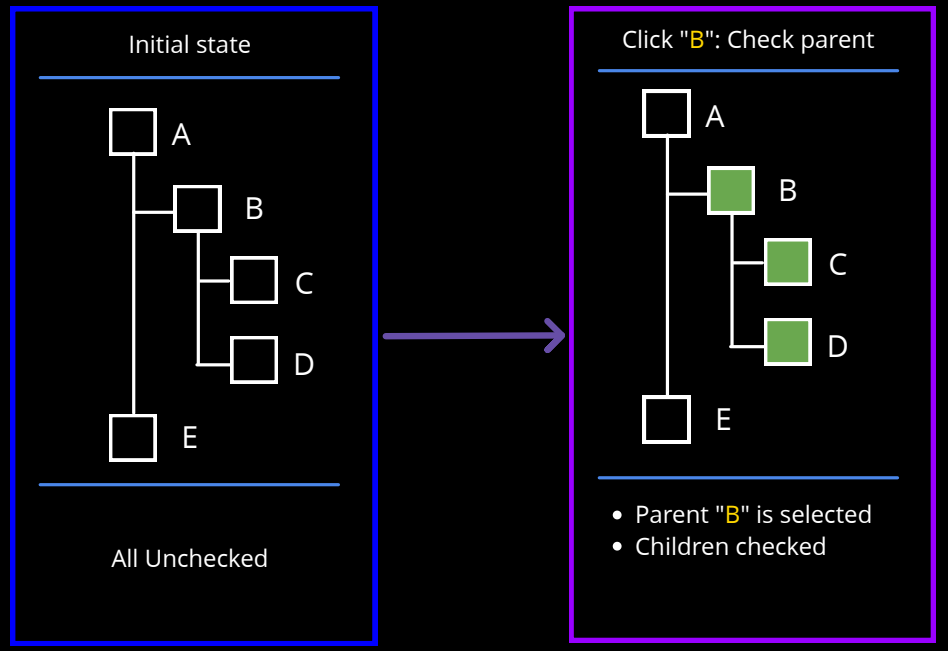
Problem 9: Tree Select



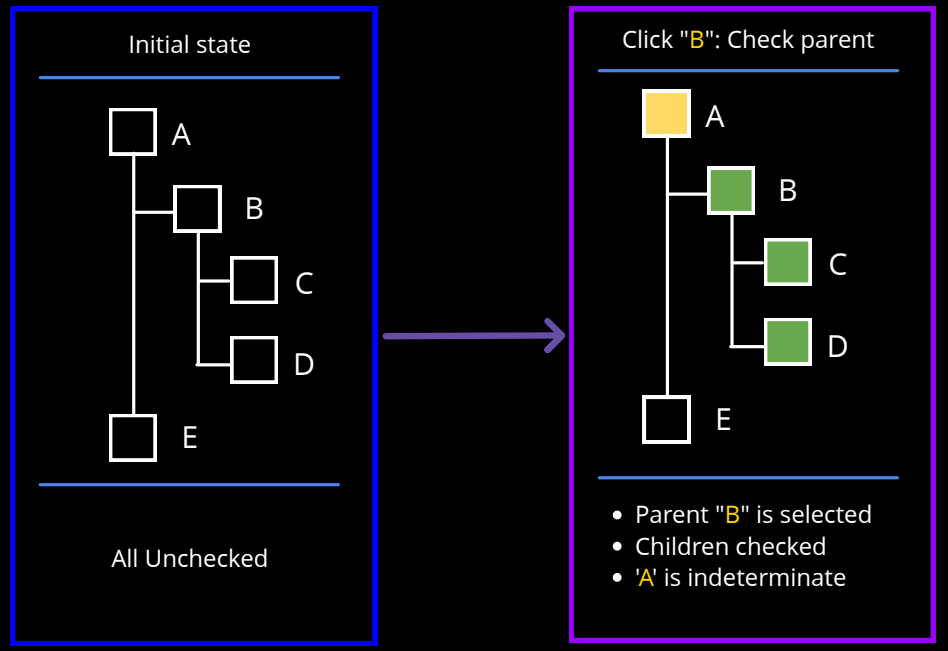
Problem 9: Tree Select



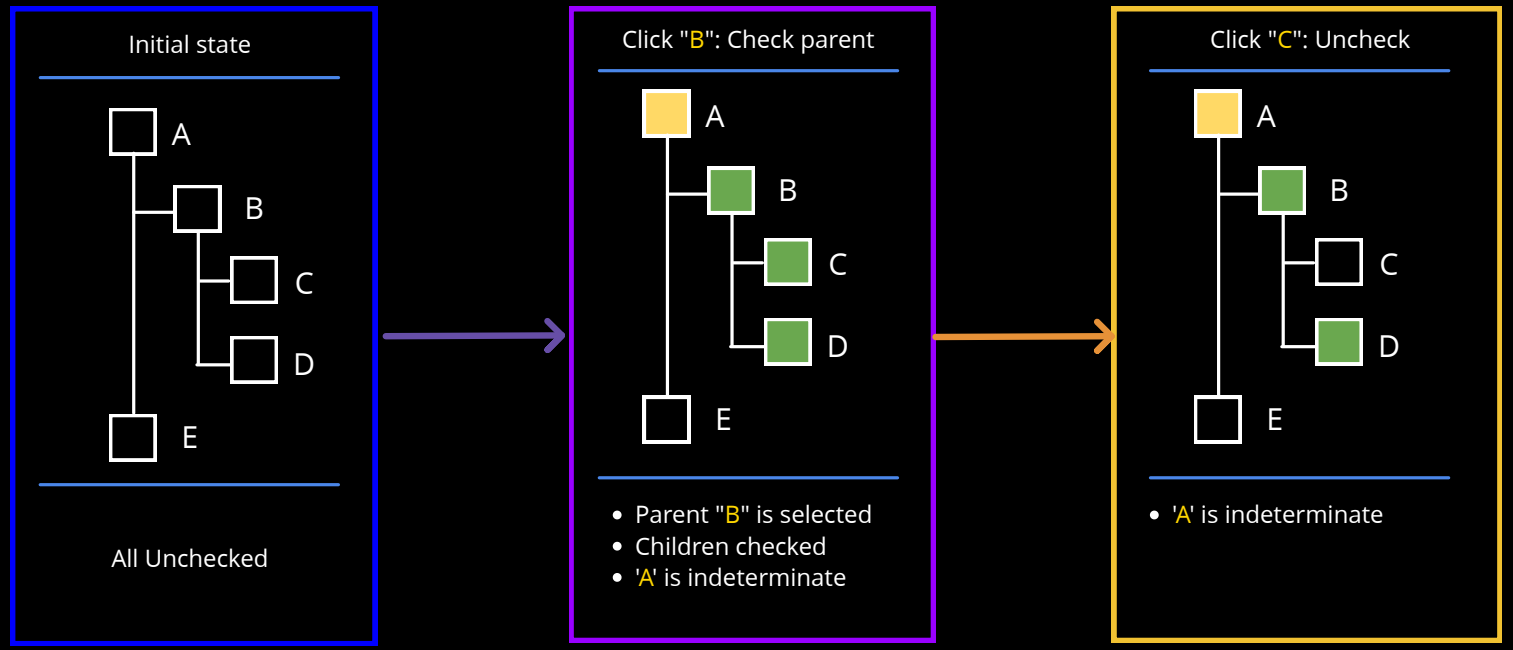
Problem 9: Tree Select



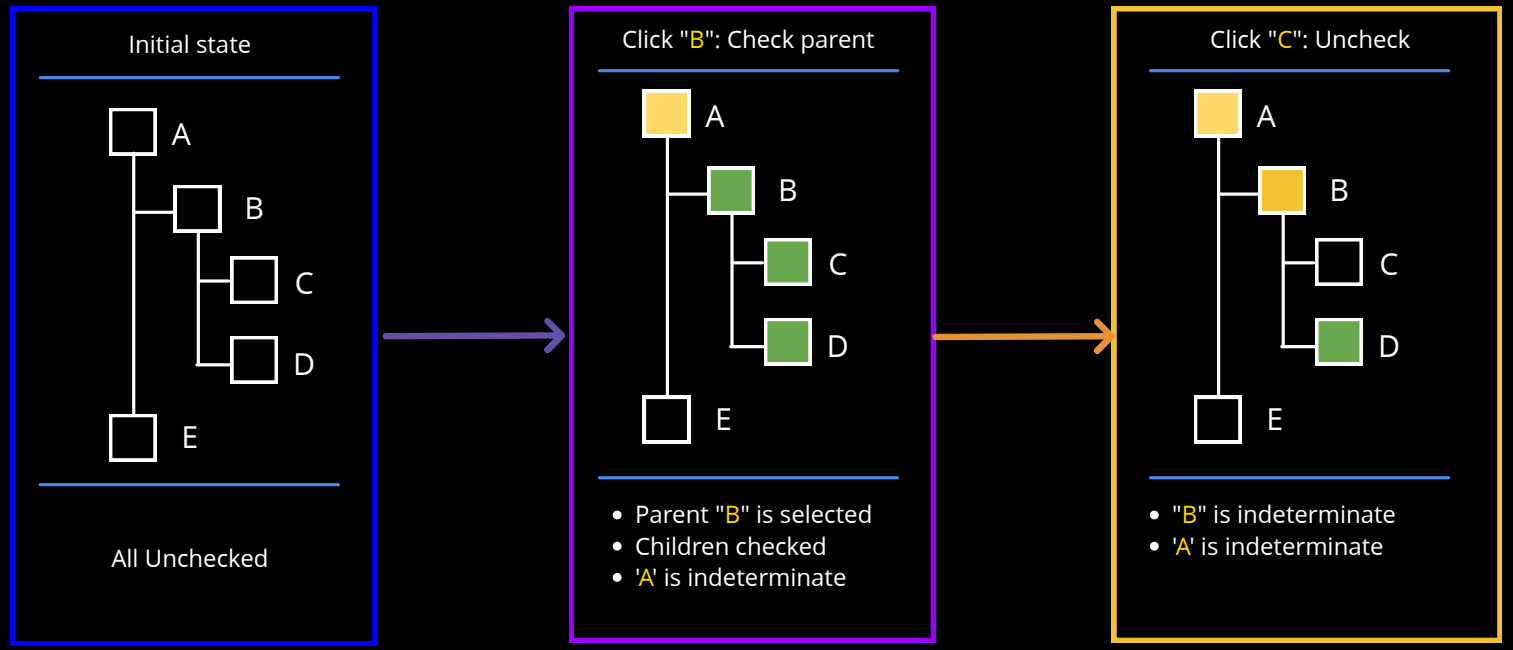
Problem 9: Tree Select



Problem 9: Tree Select



Problem 9: Tree Select



Problem 9: Tree Select

Requirements:

- **Check Parent:** Check all children.
- **Uncheck Parent:** Uncheck all children.
- **Check some children:** Parent becomes "Partial" (`[-]`).
- **Check all children:** Parent becomes "Checked" (`[v]`).

Input:

```
// paths: defines tree structure
const paths = ['a/b/c', 'a/b/d', 'a/e']
// clicks: sequence of node clicks
const clicks = ['b'] // Selects b and all its children (c, d)
```

Output

```
// Output string:
// [o]a
// .[v]b
// ..[v]c
// ..[v]d
// .[ ]e
```

Solving Tree Select

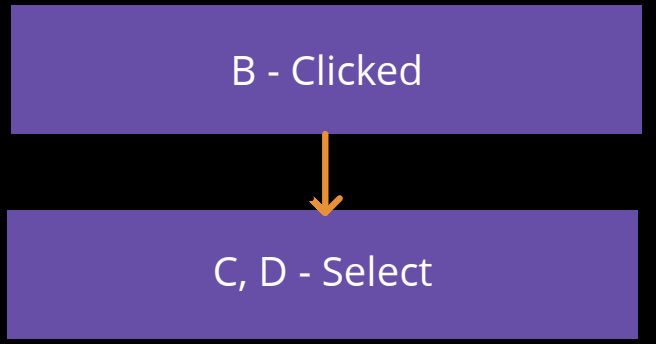
1. Set status

B - Clicked

Solving Tree Select

1. Set status

2. Propagate Down

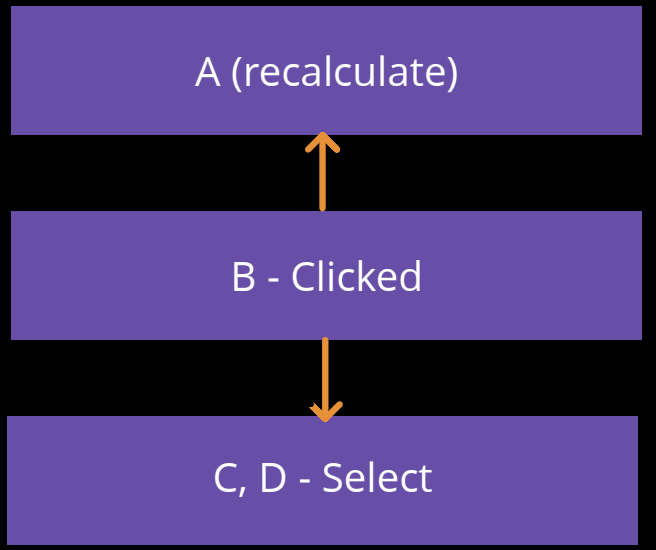


Solving Tree Select

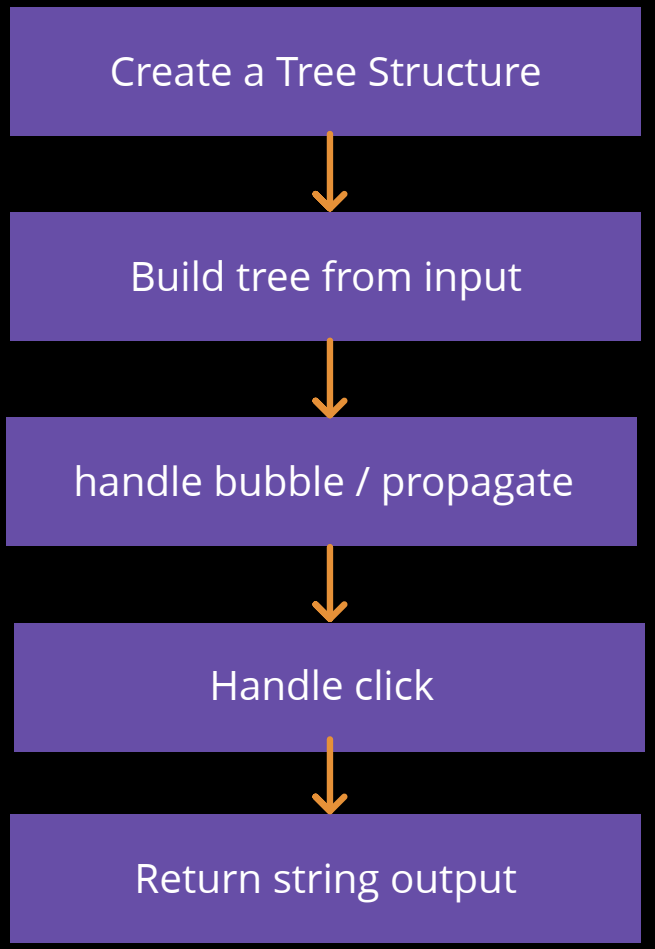
3. Bubble UP

1. Set status

2. Propagate Down



The plan


Create a Tree Structure
Build tree from input
handle bubble / propagate
Handle click
Return string output

Tree Select: DS

```ts
type SelectionStatus = 'v' | ' ' | 'o'
const SELECTED: SelectionStatus = 'v'
const NOT_SELECTED: SelectionStatus = ' '
const PARTIAL: SelectionStatus = 'o'

class TreeNode {
  children: TreeNode[] = []
  status: SelectionStatus = NOT_SELECTED

  constructor(public name: string, public parent: TreeNode | null) {}

  addChild(node: TreeNode): void {
    node.parent = this
    this.children.push(node)
  }

  updateStatus(): void {
    const selectedCount = this.children.reduce(
      (acc, node) => acc + (node.status === SELECTED ? 1 : 0), 0
    )
    const hasPartialChild = this.children.some((c) => c.status === PARTIAL)

    if (selectedCount === this.children.length && !hasPartialChild) {
      this.status = SELECTED
    } else if (selectedCount === 0 && !hasPartialChild) {
      this.status = NOT_SELECTED
    } else {
      this.status = PARTIAL
    }
  }
}
```

Tree Select: build tree

```
function createTree(paths: string[]): [TreeNode, Map<string, TreeNode>] {
  const root = new TreeNode('', null)
  const store = new Map<string, TreeNode>()

  for (const path of paths) {
    let parent: TreeNode = root
    const tokens = path.split('/')

    for (const token of tokens) {
      let node = store.get(token)
      if (!node) {
        node = new TreeNode(token, parent)
        parent.addChild(node)
        store.set(token, node)
      }
      parent = node
    }
  }

  return [root, store]
}
```

Tree Select: bubble / prop

```
function* bubble(target: TreeNode): Generator<TreeNode> {
  if (target.parent != null) {
    yield target.parent
    yield* bubble(target.parent)
  }
}

function* propagate(target: TreeNode | null): Generator<TreeNode> {
  if (target == null) return
  for (const ch of target.children) {
    yield ch
    yield* propagate(ch)
  }
}
```

Tree Select: Click

```
// Toggle + Propagate Down + Bubble Up
for (const click of clicks) {
  const node = store.get(click)
  if (!node) continue

  // Toggle status
  node.status = node.status !== ' ' ? ' ' : 'v'

  // Propagate DOWN to all descendants
  for (const child of propagate(node)) {
    child.status = node.status
  }

  // Bubble UP to recalculate ancestors
  for (const parent of bubble(node)) {
    parent.updateStatus()
  }
}
```

Tree Select: toString

```
toString(level: number = -1): string {
  const dots = Math.max(0, level)
  const root = level === -1
                    ? ''
                    : `${'.'.repeat(dots)}[${this.status}]${this.name}\n`
  return root.concat(
      this.children.map((n) => n.toString(level + 1)).join('')
  );
}
```

Part 1: Completed

5 minutes break

Part 2: Components / UX Problems

Step	Focus
1. Requirements	What does the component do? What are the user interactions?
2. Data Model	What's the input/state shape?
3. API Design	What props/methods does the component expose?
4. UI Patterns	Controlled vs Uncontrolled? Event delegation? State management?
5. Accessibility	Semantic HTML, ARIA attributes, keyboard navigation
6. Minimal Styles	Don't focus on the beauty of the component, focus on the functionality

Vanila problem solving / React problem solving

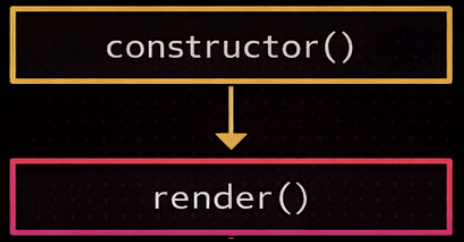
When you write a component using any library you already have some kind of a structure:

1. Render function (JSX)
2. State management (useState, useReducer)
3. Life-cycle methods / hooks (useEffect)

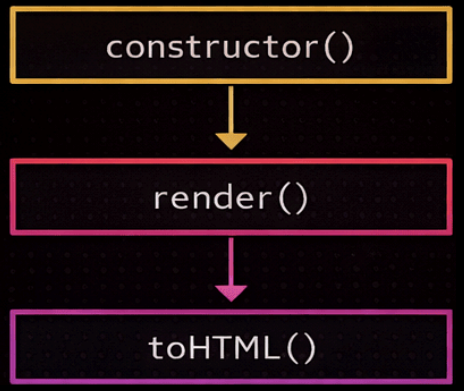
Creating component life-cycle

constructor()

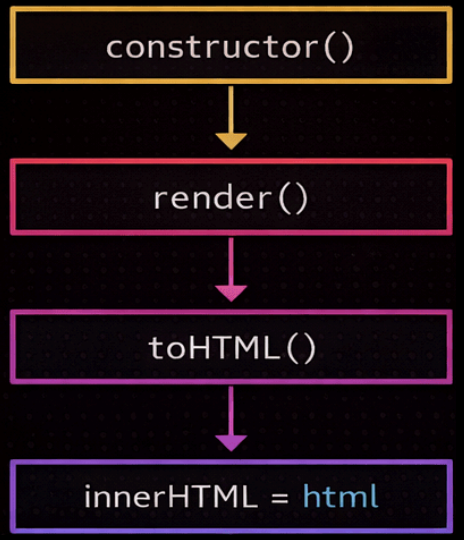
Creating component life-cycle



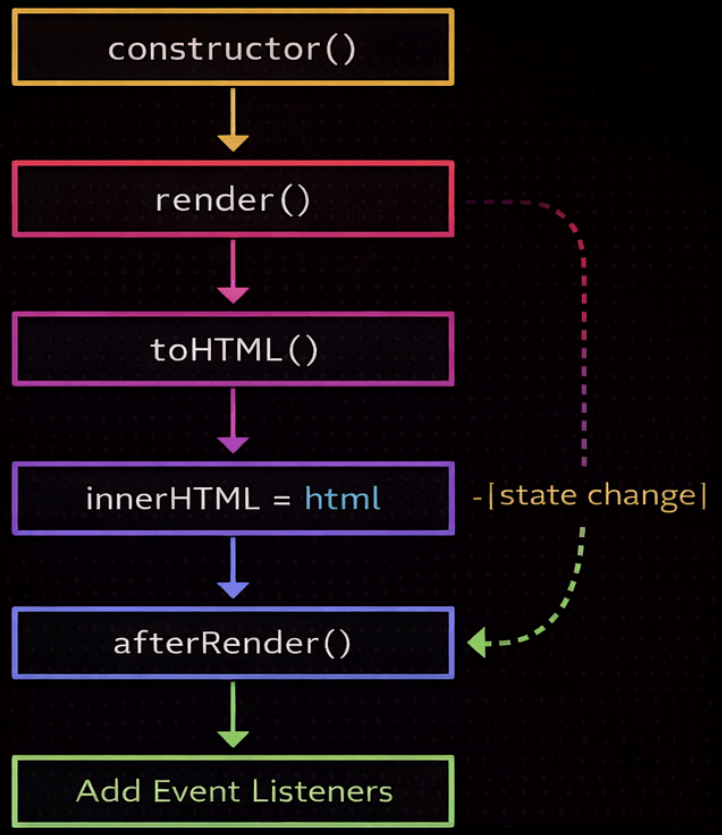
Creating component life-cycle



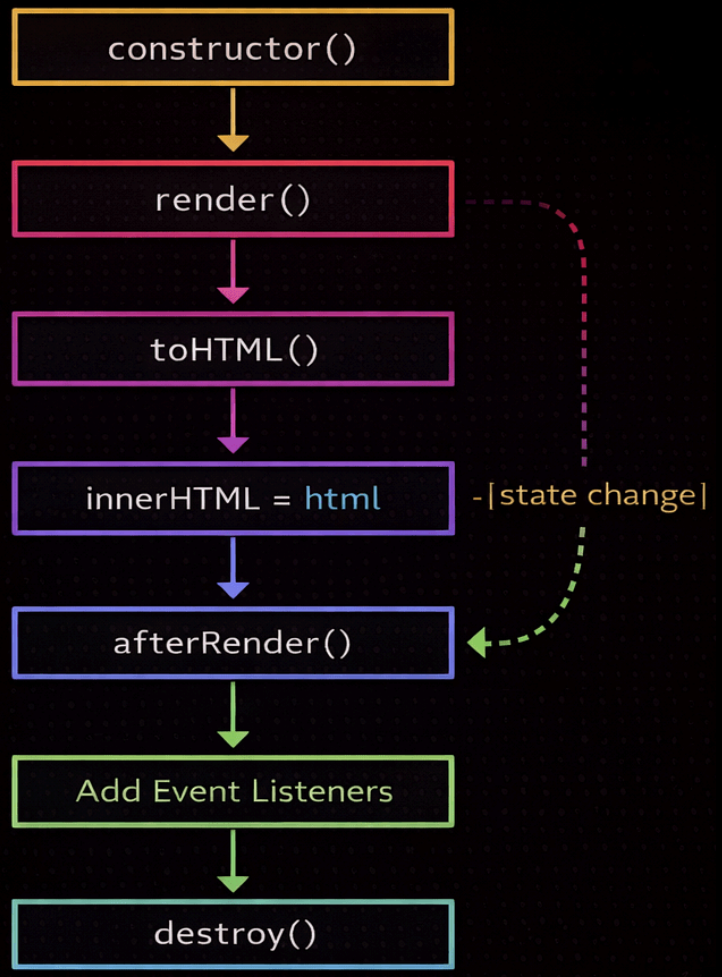
Creating component life-cycle



Creating component life-cycle



Creating component life-cycle



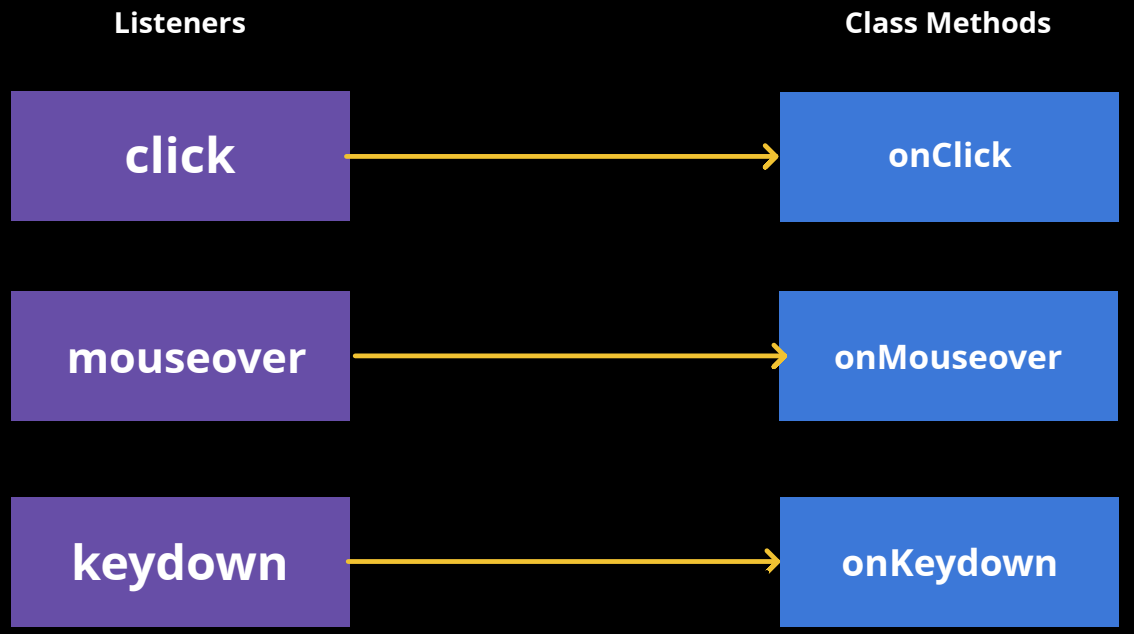
Creating AbstractComponent

```
type TComponentConfig<T extends object> = T & {
  root: HTMLElement        // Where to mount the component
  className?: string[]     // CSS classes to apply
  listeners?: string[]     // Event types to auto-bind (e.g., ['click', 'mouseover'])
  tag?: keyof HTMLElementTagNameMap  // Container element tag (default: 'div')
}
```

AbstractComponent: constructor

```
const DEFAULT_CONFIG: Partial<TComponentConfig<any>> = {
  className: [],
  listeners: [],
  tag: 'div',
}

export abstract class AbstractComponent<T extends object> {
  container: HTMLElement | null
  config: TComponentConfig<T>
  events: Array<{ type: string; callback: EventListenerOrEventListenerObject }>

  constructor(config: TComponentConfig<T>) {
    this.config = { ...DEFAULT_CONFIG, ...config }
    this.container = null
    this.events = []
  }
}
```

AbstractComponent: events



AbstractComponent: init

```
// Helper: 'click' → 'onClick', 'mouseover' → 'onMouseover'
const toEventName = (type: string): string => {
  return `on${type[0].toUpperCase()}${type.slice(1)}`
}

init() {
  this.container = document.createElement(this.config.tag)

  // Apply CSS classes
  for (const className of this.config.className) {
    this.container.classList.add(className)
  }

  // Auto-bind listeners: looks for this.onClick, this.onMouseover, etc.
  this.events = (this.config.listeners || []).map((type) => {
    const event = toEventName(type)            // 'click' → 'onClick'
    let callback = this[event]                 // Find handler on subclass
    if (!callback) {
      throw Error(`handler ${event} for ${type} is not implemented`)
    }
    callback = callback.bind(this)             // Bind to component instance
    this.container!.addEventListener(type, callback)
    return { type, callback }                  // Store for cleanup
  })
}
```

AbstractComponent: lifecycle methods

```
// Override in subclass — returns the component's HTML template
toHTML(): string {
  return ``
}

// Optional hook — runs after component is in the DOM
afterRender() {}

// The main method — orchestrates the full render cycle
render() {
  if (this.container) this.destroy()              // Clean up previous render
  this.init()                                     // Create element + bind listeners
  this.container!.innerHTML = this.toHTML()       // Set HTML content
  this.config.root.appendChild(this.container!)   // Attach to DOM
  this.afterRender()                              // Post-render hook
}

// Cleanup — remove listeners and element from DOM
destroy() {
  this.events.forEach(({ type, callback }) => {
    this.container!.removeEventListener(type, callback)
  })
  this.events = []
  this.container!.remove()
}
```

Usage example

```
type TButtonProps = { label: string }

class Button extends AbstractComponent<TButtonProps> {
  constructor(config: TComponentConfig<TButtonProps>) {
    super({ ...config, listeners: ['click'] })
  }

  onClick() {
    console.log('clicked!')
  }

  toHTML() {
    return `<button>${this.config.label}</button>`
  }
}

// Usage:
new Button({ root: document.body, label: 'Submit' }).render()
```

Challenge 8: Diff

Get an Object that is the difference between O & O1 – properties that exist in one but not both.

```
1 type Foo = { name: string; age: string }
2 type Bar = { name: string; age: string; gender: number }
3
4 type Result = Diff<Foo, Bar> // { gender: number }
```

Challenge 8: Diff | Solution

Get an Object that is the difference between O & O1 – properties that exist in one but not both.

```
1 type Diff<O extends object, O1 extends object> = {
2   [K in keyof (O & O1) as K extends keyof (O | O1) ? never : K]: (O & O1)[K]
3 }
```

(A | B | C) & (B | D) = A | B | C | D

(A | B | C)

Challenge 9: Exclude

Remove members from a union (re-implement Exclude).

```ts
type R = MyExclude<'a' | 'b' | 'c', 'a'>  // 'b' | 'c'
```

Challenge 9: Exclude | Solution

Remove members from a union (re-implement Exclude).

```
1 type MyExclude<T, U> = T extends U ? never : T
```

Problem 1: Accordion

Build an accordion component that displays a list of expandable/collapsible items. Each item consists of a title (summary) and content details. The component should allow users to toggle the visibility of the content for each item **independently**.

Requirements

- **Core Functionality**.
  - Render a list of accordion items based on the provided configuration.
  - Each item should display a **title** that can be clicked to toggle the visibility of its content.
  - Allow **independent control** of each section (multiple sections can be open at once).
- **Accessibility** (A11y)
  - Ensure **native accessibility** and keyboard support.
  - Ensure content is hidden/shown appropriatly for **screen readers**.

API Design

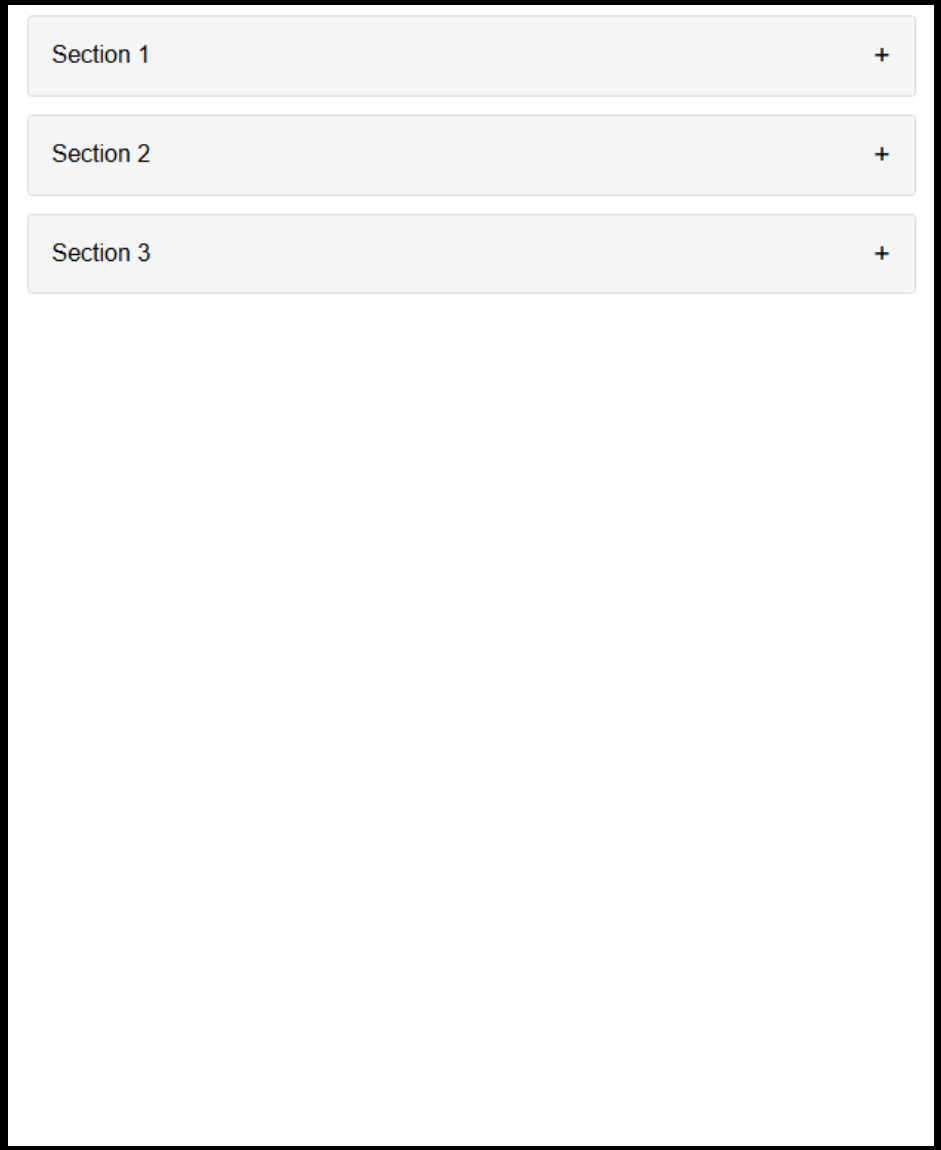
The component should accept the following props:

`items: TAccordionItem[]` – An array of items to render.

- `id: string` - Unique identifier for the item.
- `title: string` - The text to display in the **summary**/header.
- `content: string` – The text content to display when expanded.

Level: Easy

Problem 1: Accordion



Accordion - The Native HTML Approach

```
<details>
  <summary>Click me to expand</summary>
  <p>This content is hidden until you click!</p>
</details>
```

Accordion: Basic structure

```
type TAccordionItem = { id: string; title: string; content: string }
type TAccordionProps = { items: TAccordionItem[] }

export class Accordion extends AbstractComponent<TAccordionProps> {
  constructor(config: TComponentConfig<TAccordionProps>) {
    super({
      ...config,
      className: [styles.container], // No listeners needed!
    })
  }
}
```

Accordion: defining template

```
toHTML(): string {
  return this.config.items
    .map((item) => `
      <details class="${styles.details}">
        <summary class="${styles.summary}">${item.title}</summary>
        <p class="${styles.content}">${item.content}</p>
      </details>
    `)
    .join('')
}
```

Accordion: basic css

```css
summary::-webkit-details-marker {
  display: none;
}

details::details-content {
  transition:
    max-height 0.3s ease,
    content-visibility 0.3s allow-discrete;
  max-height: 0;
  overflow: hidden;
}

details[open]::details-content {
  max-height: 500px;
}
```

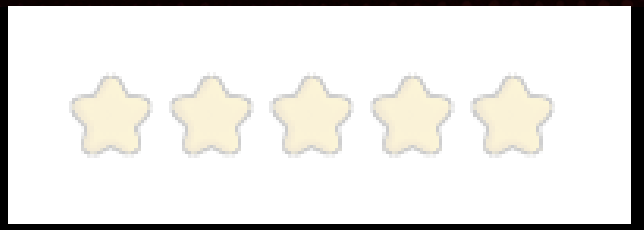
Accordion: Completed

Problem 2: Star rating

Build a **star rating** component that allows users to provide feedback by **selecting a rating** from 1 to 5 **stars**. It should support both read-only display and **interactive** selection modes.

Requirements

- **Core Functionality**
  1. **Render 5 stars by default** (configurable count is a plus).
  2. Support **controlled** and **uncontrolled** modes.
  3. **Read-only Mode:** Display the current rating without hover effects or interactivity.
  4. **Visuals:** Filled stars should be **visually distinct** (e.g, gold/yellow). Empty stars should be outlined or gray.



Level: Medium

Controlled vs Uncontrolled

Before we code, let's understand this pattern:

Controlled:

Parent manages state via **value + onChange**

```tsx
<StarRating value{rating} onChange={setRating} />
```

Uncontrolled:

Component manages its own state via **defaultValue**

```tsx
<StarRating defaultValue={3} />
```

The key question is: **who owns the state?**

In controlled mode, the parent does. In **unconollred** mode, the component does.

Our solution will need to support both.

Star Rating: Basic structure

```
type TStarRatingProps = {
  value: number
  onValueChange: (value: number) => void
  readOnly?: boolean
}

export class StarRating extends AbstractComponent<TStarRatingProps> {
  private value: number = 0

  constructor(config: TComponentConfig<TStarRatingProps>) {
    super({
      ...config,
      listeners: ['click'], // We need to handle clicks!
    })
    this.value = config.value
  }
}
```

Star Rating: Click handler

```
onClick(event: MouseEvent): void {
  if (this.config.readOnly) return  // Respect read-only mode

  const button = (event.target as HTMLElement).closest('button')
  if (!button) return

  const starValue = Number(button.dataset.starValue)
  if (!Number.isNaN(starValue)) {
    this.value = starValue
    this.config.onValueChange(starValue)  // Notify parent
    this.render()  // Re-render with new value
  }
}
```

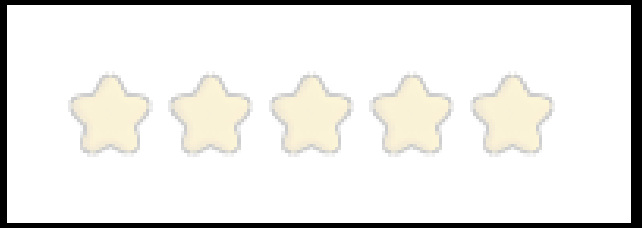
Star Rating: rendering stars

```
toHTML(): string {
  const stars = Array.from({ length: 5 }, (_, index) => {
    const starValue = index + 1
    return `
      <button
        data-star-value="${starValue}"
        data-active="${this.value >= starValue}"
        role="radio"
        aria-checked="${this.value === starValue}"
        aria-label="${starValue} Star${starValue === 1 ? '' : 's'}"
        ${this.config.readOnly ? 'disabled' : ''}
      >⭐</button>
    `
  }).join('')

  return `<div>${stars}</div>`
}
```

Star Rating: A11y

```
toHTML(): string {
  const stars = Array.from({ length: 5 }, (_, index) => {
    const starValue = index + 1
    return `
      <button
        data-star-value="${starValue}"
        data-active="${this.value >= starValue}"
        role="radio"
        aria-checked="${this.value === starValue}"
        aria-label="${starValue} Star${starValue === 1 ? '' : 's'}"
        ${this.config.readOnly ? 'disabled' : ''}
      >⭐</button>
    `
  }).join('')

  return `<div>${stars}</div>`
}
```

Star Rating: Result



Challenge 10: IsNever

Implement a type IsNever<T>, which takes input type T. If the type of T is never, return true, otherwise return false.

```
type A = IsNever<never>     // true
type B = IsNever<string>    // false
type C = IsNever<undefined> // false
```

Challenge 10: IsNever

Implement a type IsNever<T>, which takes input type T. If the type of T is never, return true, otherwise return false.

```
type Bad<T> = T extends never ? true : false // will not work


type A = string | never // this will be just string
```

Challenge 10: IsNever | Solution

Implement a type IsNever<T>, which takes input type T. If the type of T is never, return true, otherwise return false.

```
type Bad<T> = [T] extends [never] ? true : false
```

Challenge 11: AnyOf

Implement a type which takes an array type as argument. If any element in the array is "truthy", return true, otherwise return false.

```
1 type Sample1 = AnyOf<[1, "", false, []]>     // true (1 is truthy)
2 type Sample2 = AnyOf<[0, "", false, [], {}]> // false (all falsy)
```

Challenge 11: AnyOf | Solution

Implement a type which takes an array type as argument. If any element in the array is "truthy", return true, otherwise return false.

```
type Falsy = null | undefined | false | '' | [] | 0

type IsTruthy<T> = T extends Falsy ? false : keyof T extends never ? false : true

type AnyOf<T extends readonly any[]> = T extends [infer First, ...infer Tail]
  ? IsTruthy<First> extends true
    ? true
    : AnyOf<Tail>
  : false
```

Problem 3: Tabs

Build a **reusable Tabs** component that allows switching between different content views.

Requirements

- Core Functionality

1. **Render** a list of **tab buttons** (**tablist**) and corresponding panels (**tabpanel**).
2. Only one panel should be visible at a time.
3. Clicking a tab should **activate it** and show its associated panel.

Level: Medium

Tabs: API Design

What do we need?

- **Multiple tabs** with headers, one active at a time, content swaps when you click

Data shape?

- An array of **{ name, content }** objects plus tracking the active tab name

The pattern?

- **Partial DOM updates** - we won't re-render the whole component, just swap the content area's **innerHTML**.

Accessibility?

- `role="tablist"` on the nav
- `role="tab"` on buttons
- `role="tapanel"` on content

Tabs: Basic structure

```
type TTabsProps = {
  tabs: { name: string; content: string }[]
  defaultTab?: string
  target?: HTMLElement // Optional external content container
}

export class Tabs extends AbstractComponent<TTabsProps> {
  #activeTabName: string
  #contentContainer: HTMLElement | null = null

  constructor(config: TComponentConfig<TTabsProps>) {
    super({ ...config, listeners: ['click'] })
    this.#activeTabName = config.defaultTab || config.tabs[0].name
  }
}
```

Tabs: Rendering tabs

```
toHTML(): string {
  const tabButtons = this.config.tabs
    .map(tab => `
      <li>
        <button data-tab-name="${tab.name}">${tab.name}</button>
      </li>
    `)
    .join('')

  const contentArea = this.config.target
    ? ''
    : `<section class="${css.container}"></section>`

  return `
    <nav>
      <ul>${tabButtons}</ul>
    </nav>
    ${contentArea}
  `
}
```

Tabs: Active tab

```
#activateTab(tabName: string): void {
  const tab = this.config.tabs.find(t => t.name === tabName)
  if (!tab) return

  this.#activeTabName = tabName

  // Update the content container (external or internal)
  const container = this.config.target || this.#contentContainer
  if (container) {
    container.innerHTML = tab.content  // Just swap content, no full
  }
}
```

Tabs: Handle clicks

```
onClick(event: MouseEvent): void {
  const button = (event.target as HTMLElement).closest('button')
  const tabName = button?.dataset.tabName

  if (tabName && tabName !== this.#activeTabName) {
    this.#activateTab(tabName)
  }
}

afterRender(): void {
  if (!this.config.target) {
    this.#contentContainer = this.container!.querySelector(`.${css.c
  }
  this.#activateTab(this.#activeTabName)  // Show initial tab
}
```

Tabs: Result

Problem 4: Dialog

Requirements

Core Functionality

1. **Modal Behavior** – `showModal()` + backdrop
2. **Open / Close** – controlled via props/methods
3. **Actions** – `Confirm` & `Cancel` buttons
4. **Native Close** – handle `"close"` event (Escape key)

Accessibility (A11y)

1. **Focus Management** – focus first focusable element on open.
2. **Escape Key** – closes dialog (native)

Level: Medium

Dialog: native element

Before we even write code, let's appreciate what `<dialog>` gives us for free:

HTML

```
<dialog>
  <p>Are you sure?</p>
  <button> Cancel</button>
</dialog>
```

The API:

- `dialog.showModal()` – opens as modal with backdrop
- `dialog.close()` – closes the dialog
- `close` event – fired on Escape key or `close()` call

Free a11y:

- ✓ **Focus trapping** (can't tab outside)
- ✓ **Escape Key** closes it
- ✓ **Background is inert** (can't click behind)

Dialog: Basic structure

```
type TDialogProps = {
  content: string
  onConfirm: () => void
  onCancel: () => void
}

export class Dialog extends AbstractComponent<TDialogProps> {
  #dialogElement: HTMLDialogElement | null = null

  constructor(config: TComponentConfig<TDialogProps>) {
    super({
      ...config,
      listeners: ['click'],
    })
  }
}
```

Dialog: HTML

```
toHTML(): string {
  return `
    <dialog class="${css.container}">
      <section>${this.config.content}</section>
      <footer>
        <button data-action="confirm" autofocus>Confirm</button>
        <button data-action="cancel">Cancel</button>
      </footer>
    </dialog>
  `
}
```

Dialog: Click Handler & Life-cycle

```
onClick(event: MouseEvent): void {
  const action = (event.target as HTMLElement).dataset.action

  if (action === 'confirm') {
    this.config.onConfirm()
    this.close()
  } else if (action === 'cancel') {
    this.config.onCancel()
    this.close()
  }
}
```

Dialog: close event and API

```
onClose(): void {
  this.config.onCancel()
}

open(): void { this.#dialogElement?.showModal() }
close(): void { this.#dialogElement?.close() }
```

Dialog: CSS

```css
dialog::backdrop {
  background-color: rgba(0, 0, 0, 0.2);
  backdrop-filter: blur(6px); /* Frosted glass effect! */
}

dialog {
  border: none;
  border-radius: 8px;
  filter: drop-shadow(0 0 0.75rem rgba(0, 0, 0, 0.2));
}
```

Challenge 12: Look up

Get the corresponding type by searching for the common `type` field in a union.

```ts
interface Cat { type: 'cat'; breeds: 'Abyssinian' | 'Shorthair' | 'Curl' | 'Bengal' }
interface Dog { type: 'dog'; breeds: 'Hound' | 'Brittany' | 'Bulldog' | 'Boxer' }

type MyDog = LookUp<Cat | Dog, 'dog'> // Dog
```

Challenge 12: Look up | Solution

Get the corresponding type by searching for the common `type` field in a union.

```
type LookUp<U, T> = U extends { type: T } ? U : never

(Cat extends { type: 'cat' } ? true : never) |  (Dog extends { type: 'dog' } ? true | never)
never | true
true

```

Challenge 13: ReturnType

Implement the built-in `ReturnType<T>` generic without using it.

```ts
const fn = (v: boolean) => v ? 1 : 2
type Result = MyReturnType<typeof fn> // 1 | 2
```

Challenge 13: ReturnType | Solution

Implement the built-in `ReturnType<T>` generic without using it.

```
type MyReturnType<T extends (...args: any) => any> =
                T extends (...args: any) => infer R ? R : never
```

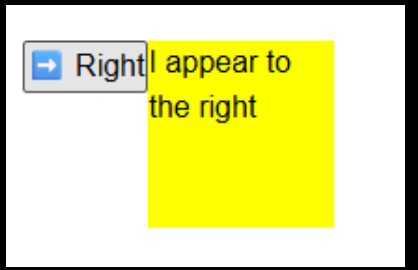
Problem 5 - Tooltip

Implement a `Tooltip` component that displays additional information when a user interacts with an element (hover or focus). It should support customizable positioning and behave accessibly.

Requirements

Core Functionality

1. **Triggering:** The tooltip should appear on `mouseenter` and `focusin` events,
2. **Positioning:**
   - Support explicit positions: `top`, `bottom`, `left`, `right`
   - Support `auto` positioning: Automatically choose the best position based on available viewport space
3. **Content:**
   - ✓ Accept **arbitrary HTML content** (string for Vanilla, ReactNode for React) for the tooltip body
4. **Resiliency**
   - ✓ Handle edge cases where the **tooltip** might overflow the viewport



Level: Medium

Tooltip: Solution approach

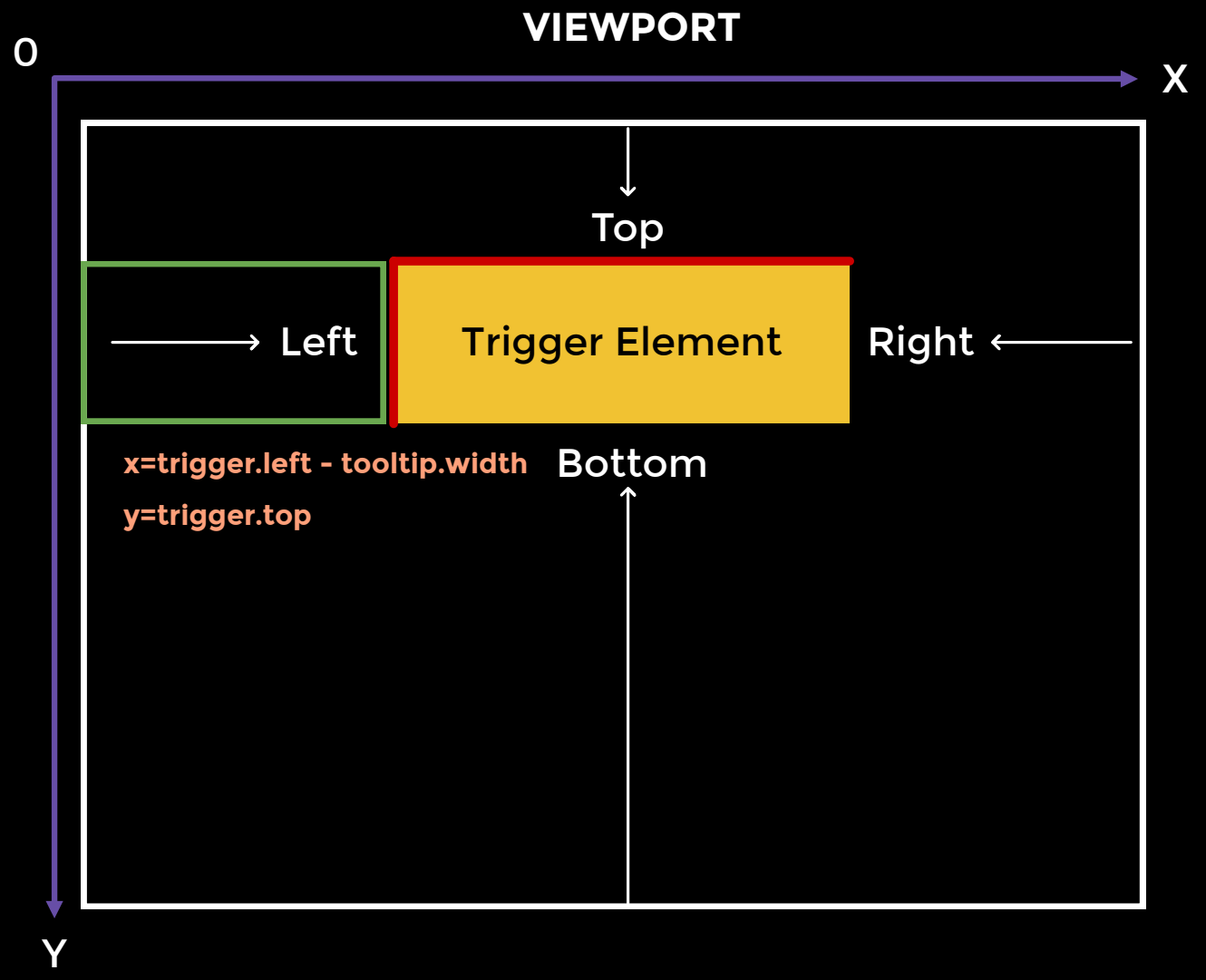
1 **Event Listeners:**

- Bind `mouseenter` / `mouseleave` to show / hide
- Bind `focusin` / `focusout` to show/hide (bubbling events are crucial).
- Bind `keydown` to listen for **Escape**

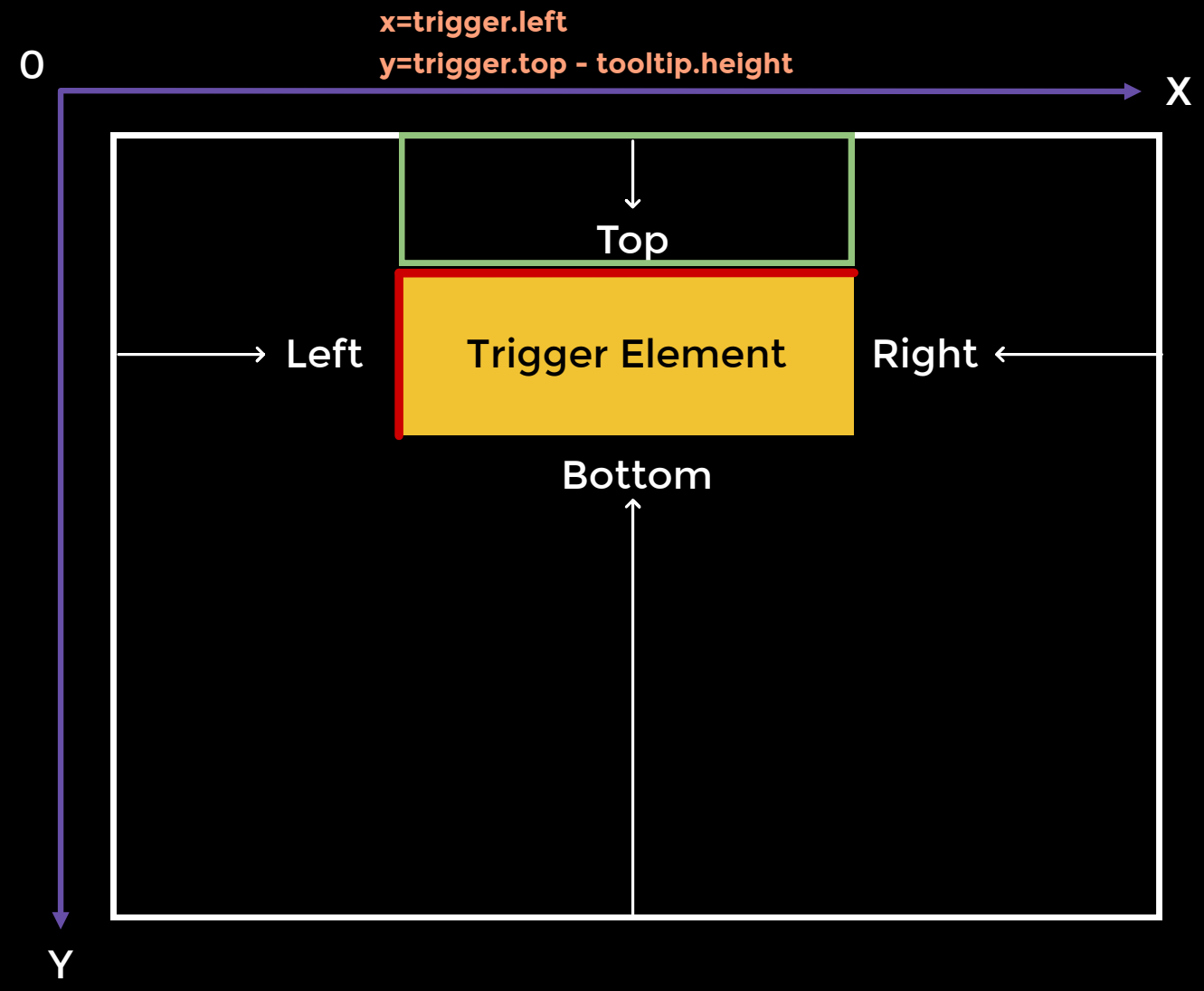
2 **Positioning logic:**

- Use absolute positioning relative to a container.
- For `auto`, calculate available space in all 4 directions using `getBoundingClientRect()` and the viewport dimensions.
- Switch class names or styles dynamically based on the chosen position.

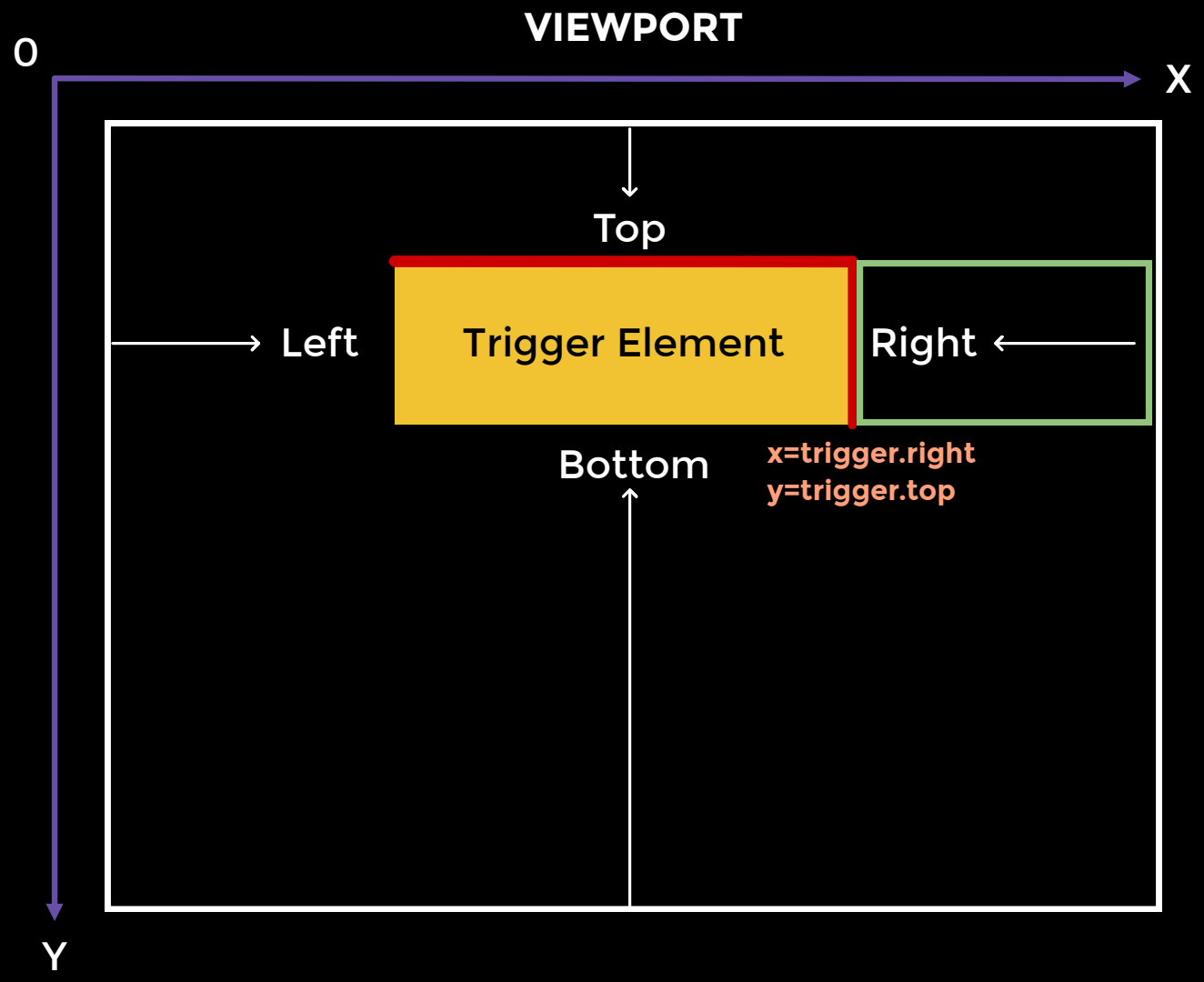
Tooltip: How to handle autoposioning



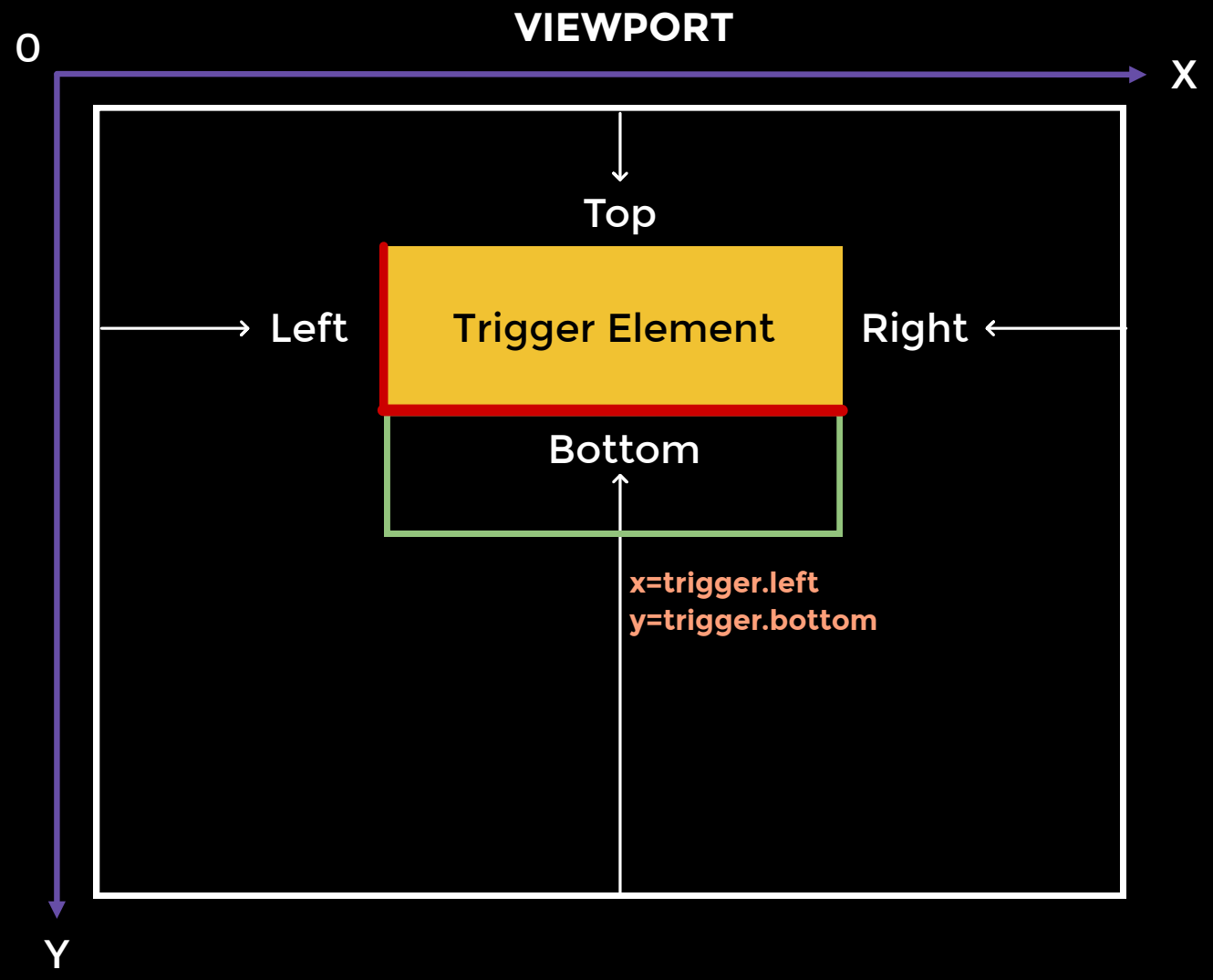
Tooltip: How to handle autoposioning



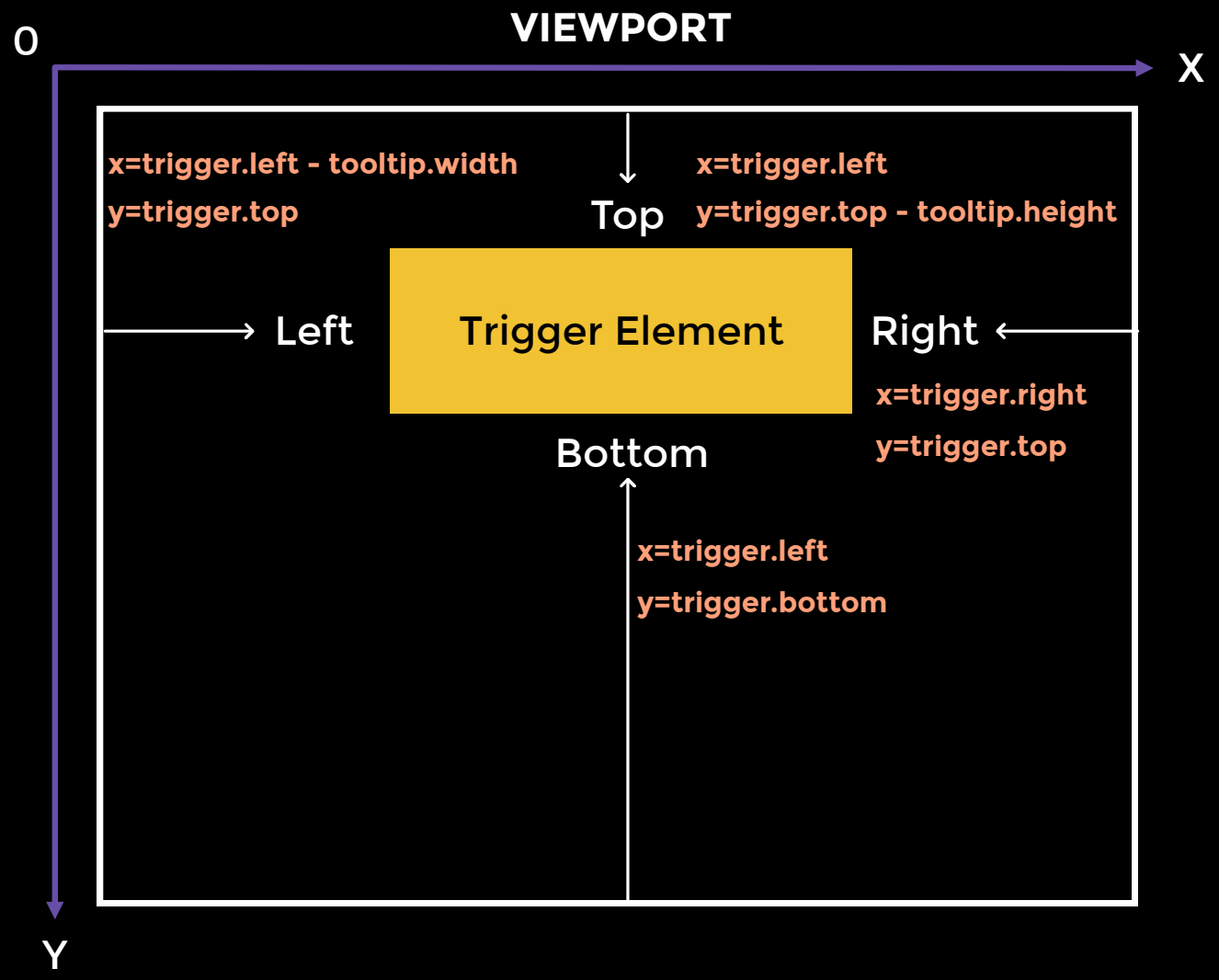
Tooltip: How to handle autoposioning



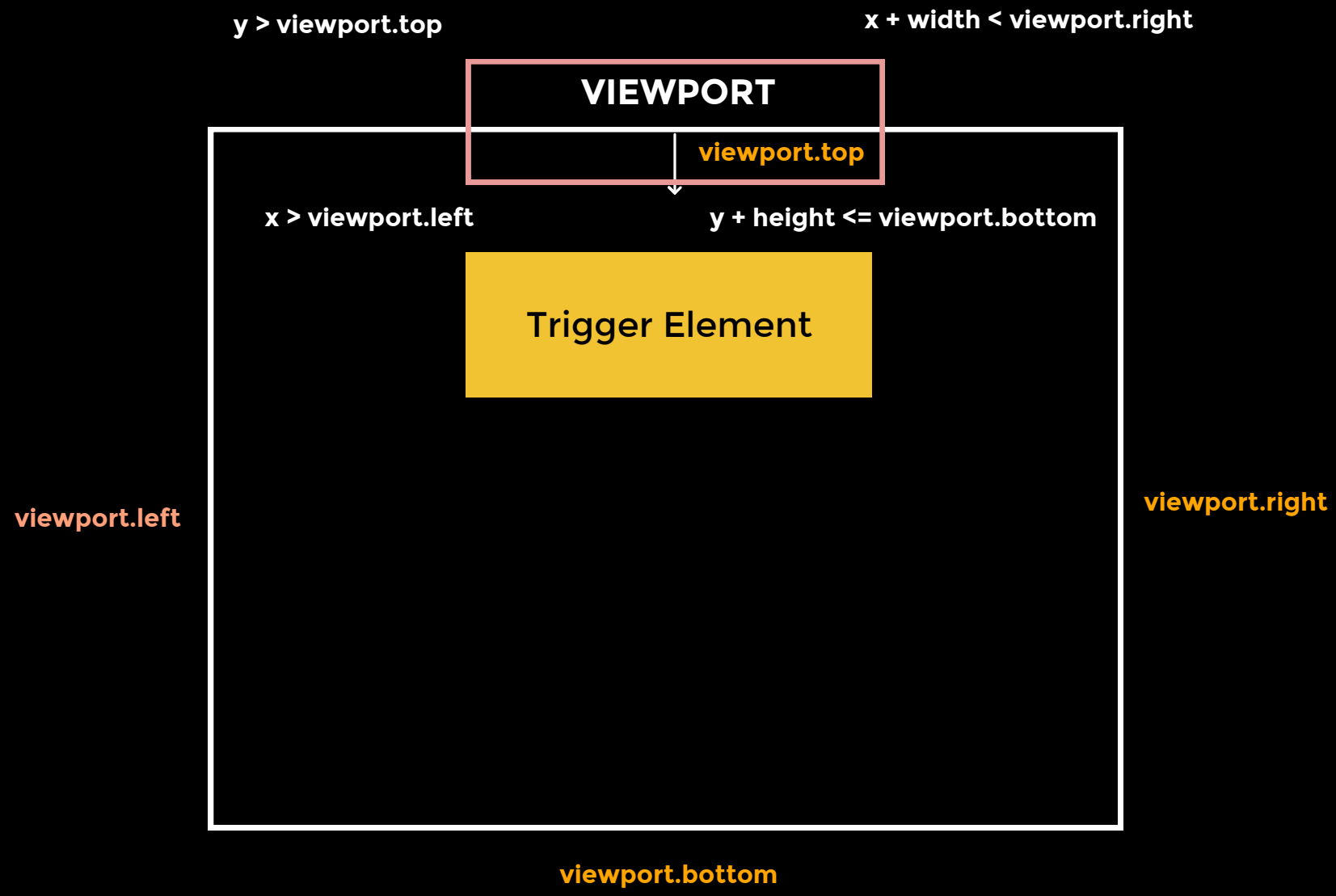
Tooltip: How to handle autoposioning



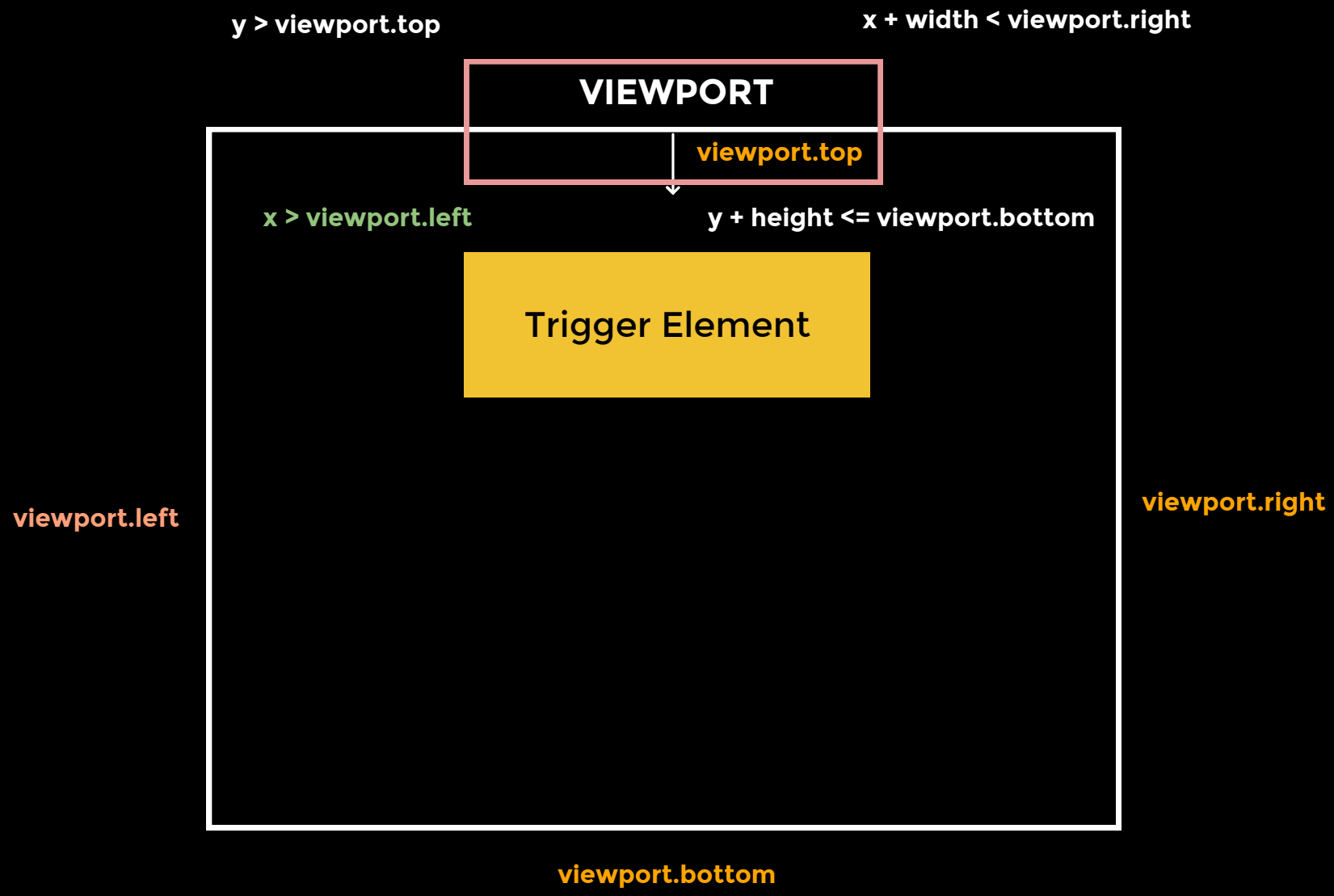
Tooltip: How to handle autoposioning



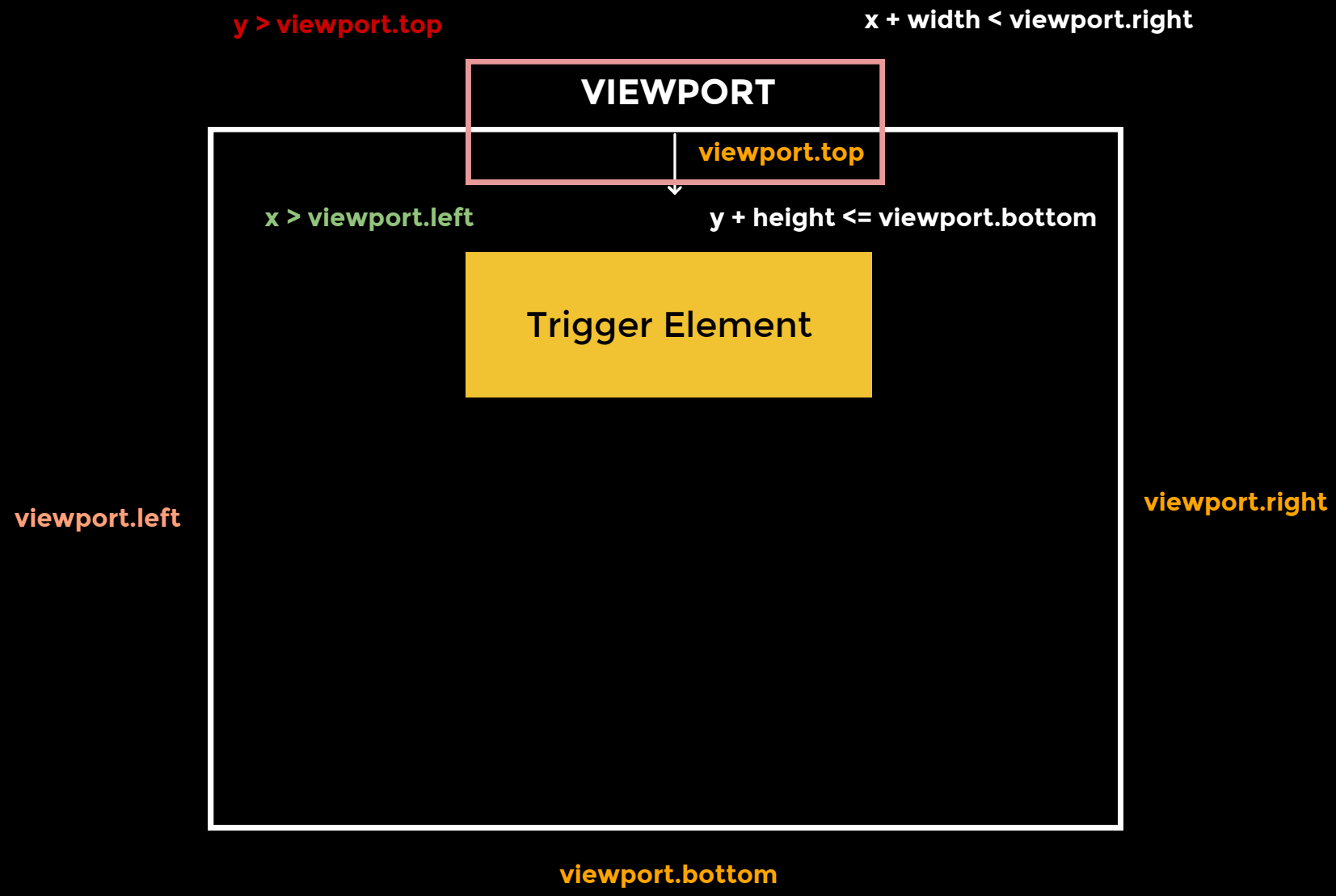
Tooltip: How to handle autoposioning



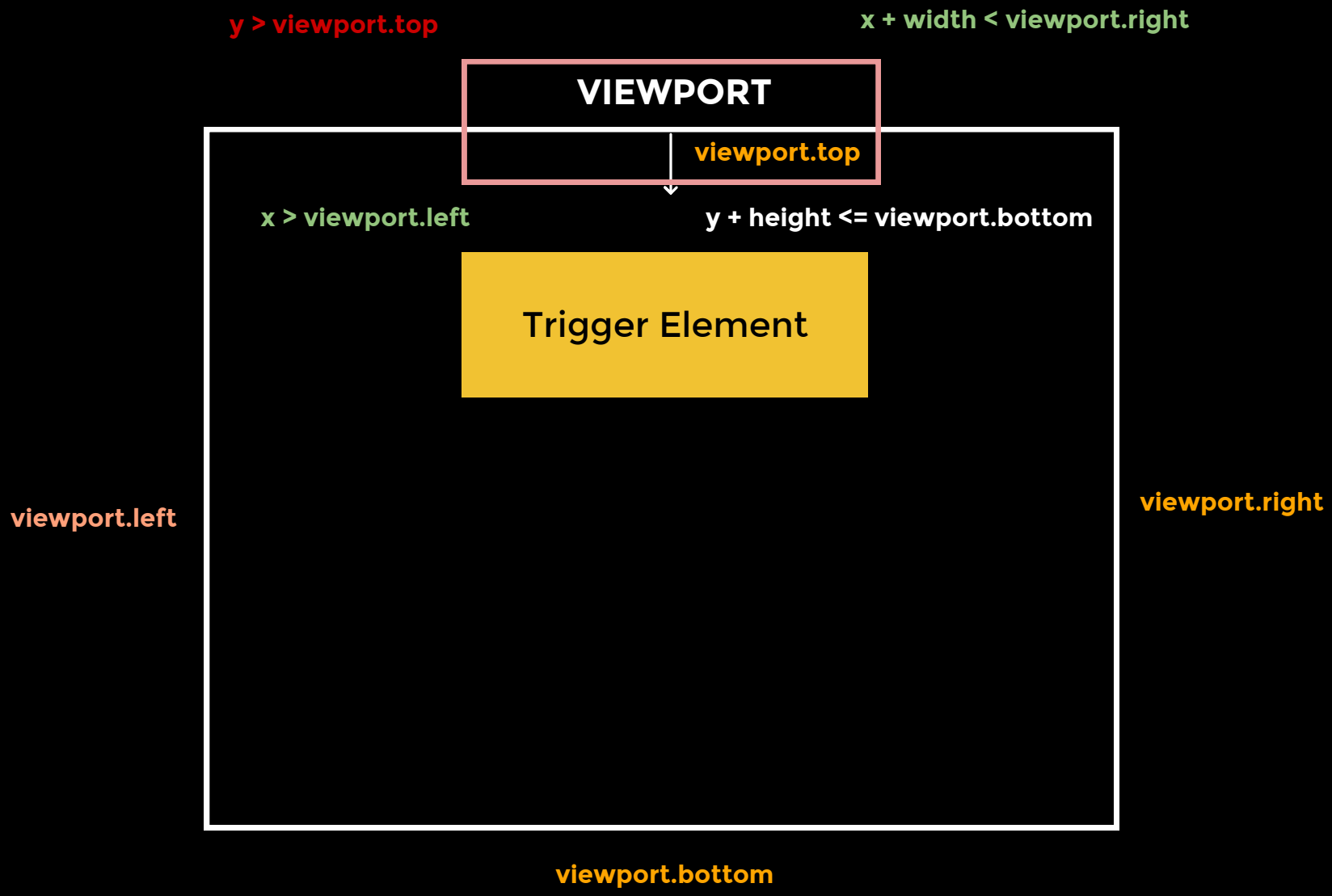
Tooltip: How to handle autoposioning



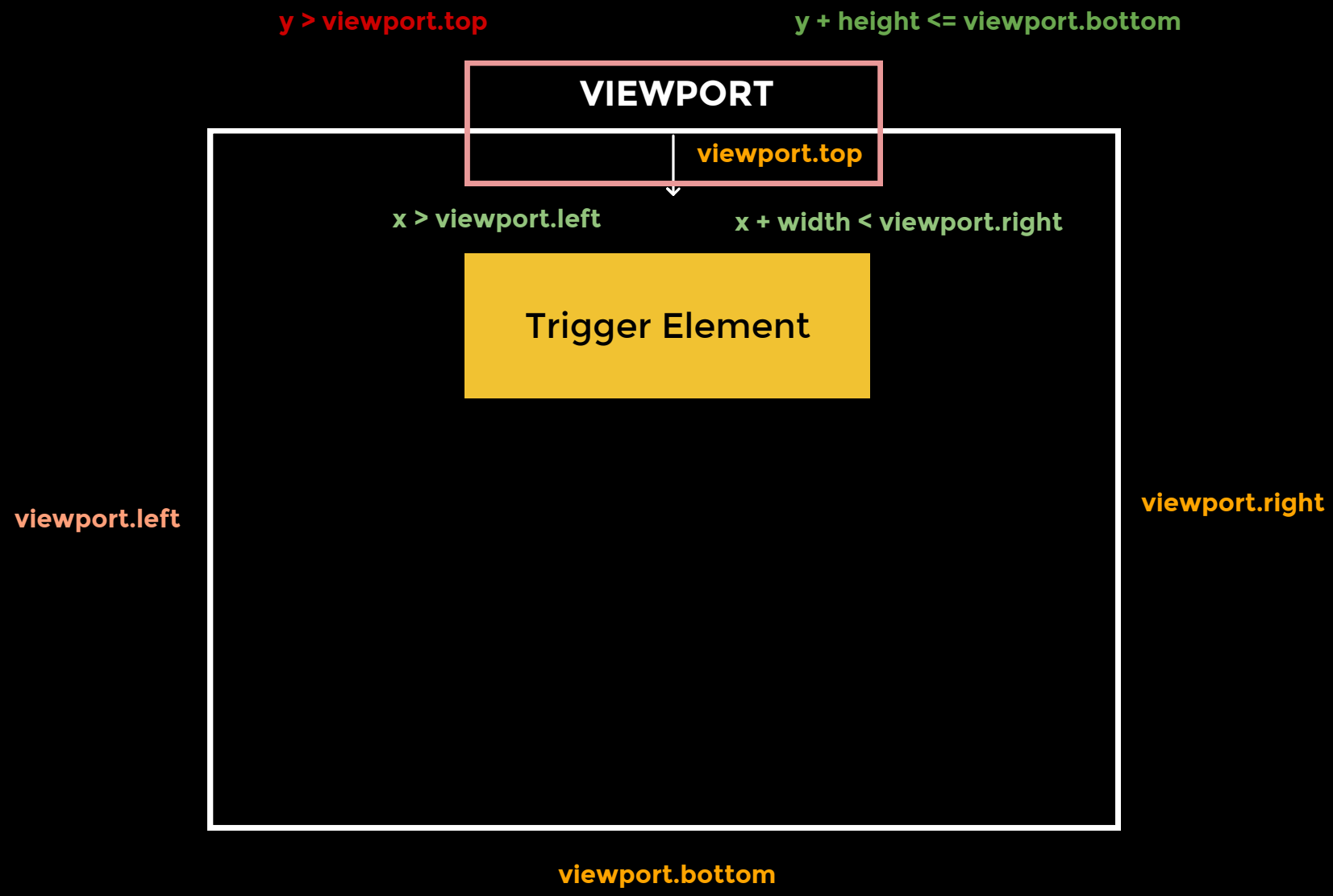
Tooltip: How to handle autoposioning



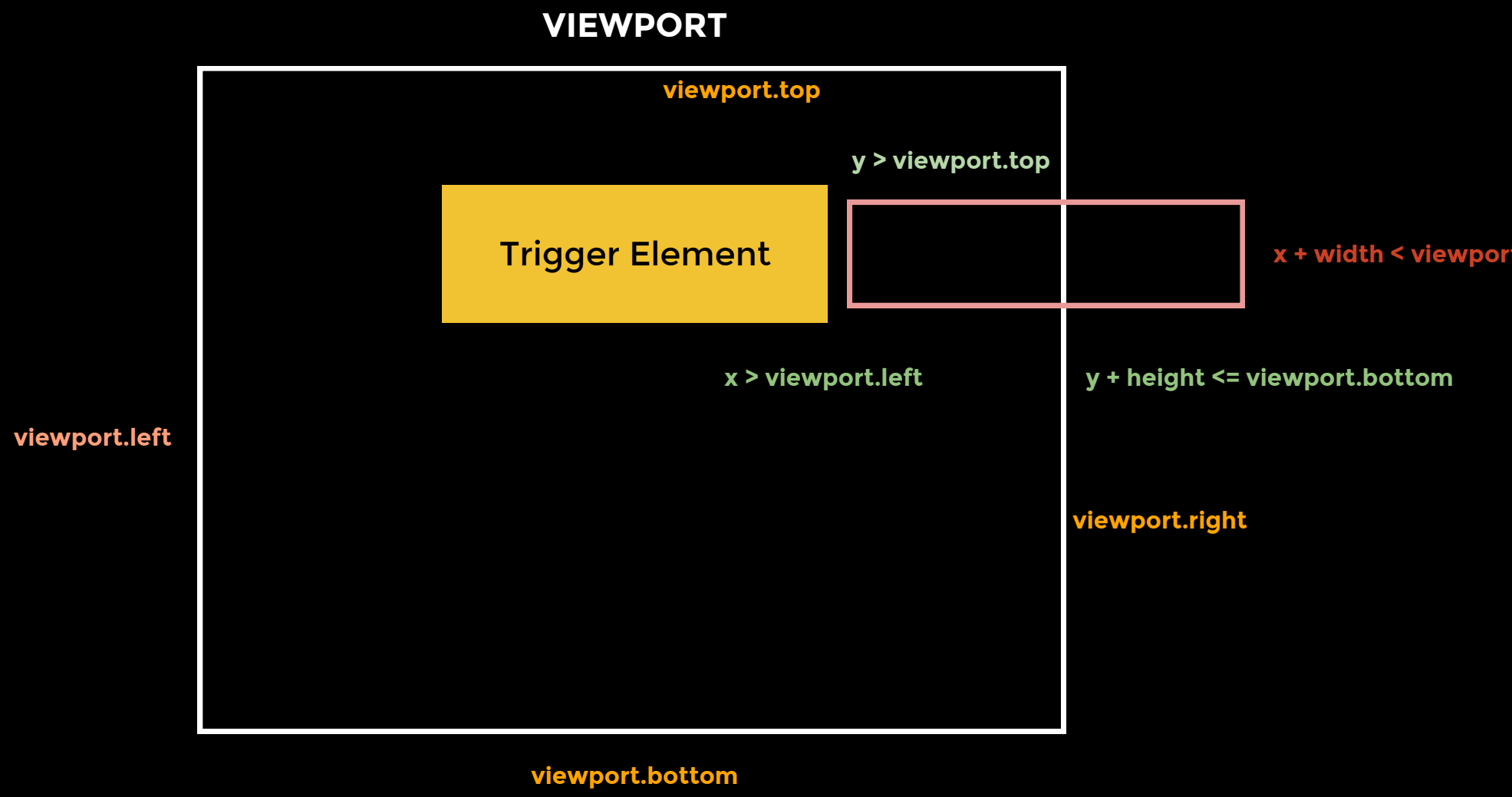
Tooltip: How to handle autoposioning



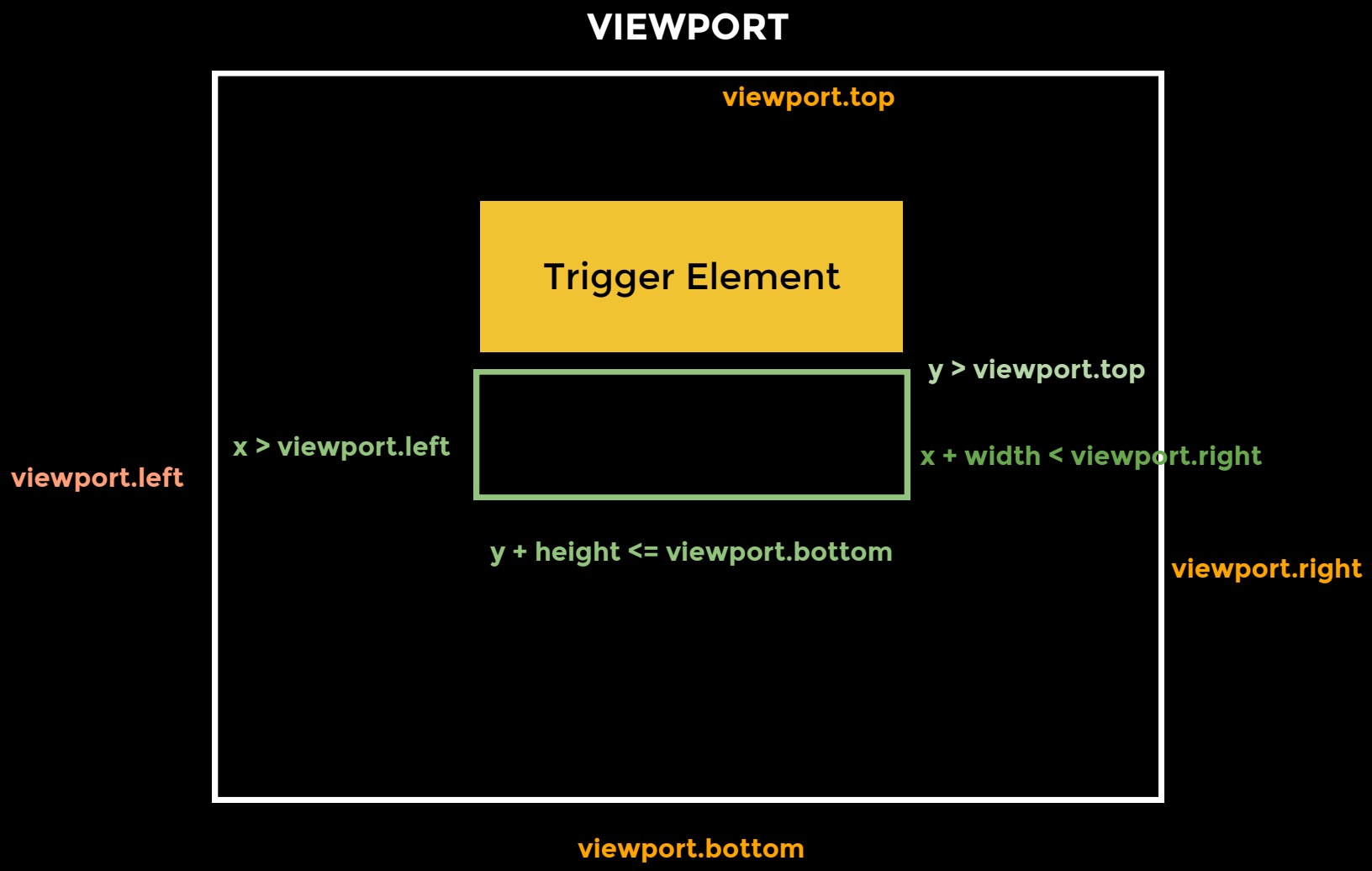
Tooltip: How to handle autoposioning



Tooltip: How to handle autoposioning



Tooltip: How to handle autoposioning



Tooltip: Setting up

```
type TTooltipProps = {
  position?: 'top' | 'bottom' | 'left' | 'right' | 'auto'
  children: HTMLElement
  content: string
}

export class Tooltip extends AbstractComponent<TTooltipProps> {
  tooltipElement: HTMLElement | null = null

  constructor(config: TComponentConfig<TTooltipProps>) {
    super({
      ...config,
      listeners: [
        'mouseenter',
        'mouseleave',
        'focusin',
        'focusout',
        'keydown'
      ],
    })
  }
}
```

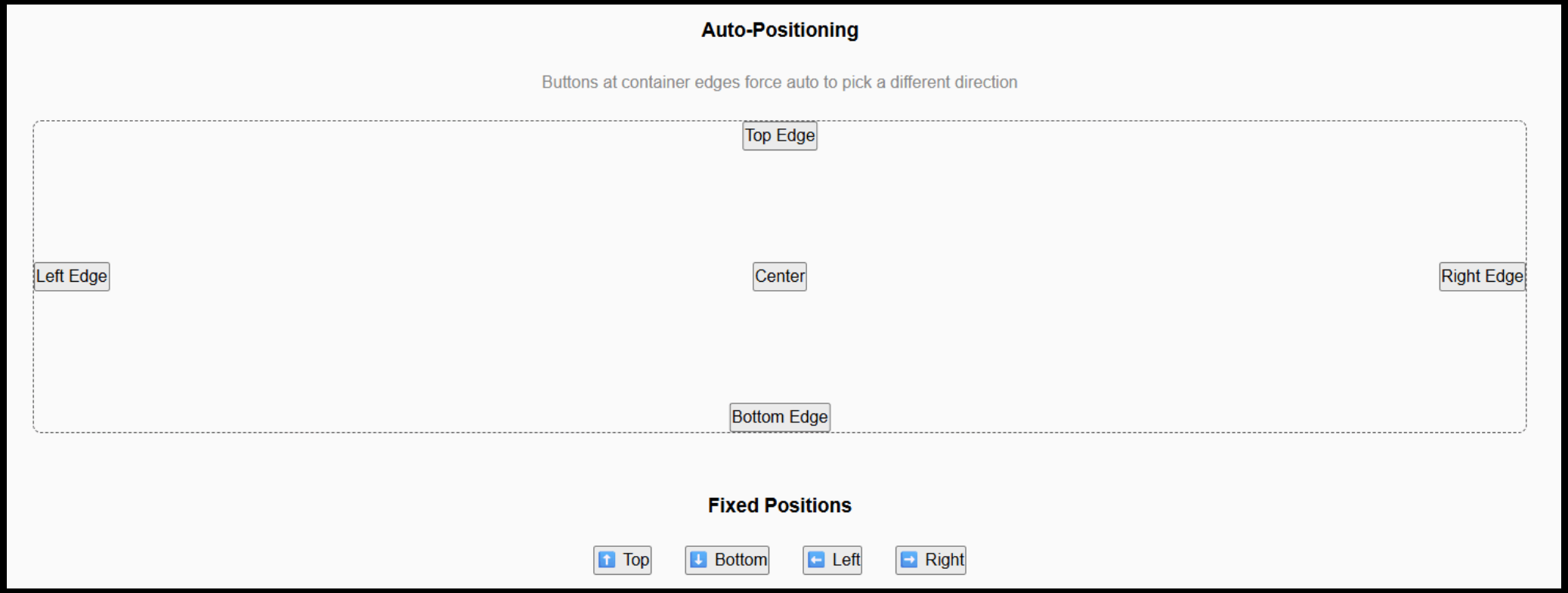
Tooltip: Event listeners

```
onMouseenter() { this.showTooltip() }
onMouseleave() { this.tooltipElement!.style.display = 'none' }

onFocusin() { this.showTooltip() }
onFocusout() { this.tooltipElement!.style.display = 'none' }

onKeydown(e: KeyboardEvent) {
  if (e.key === 'Escape') {
    this.tooltipElement!.style.display = 'none'
  }
}
```

Tooltip: Auto-positioning

```ts
function getAutoPosition(
  tooltip: HTMLElement,
  container: HTMLElement,
  boundaryRect: { left: number; top: number; right: number; bottom: number },
 ) {
  const t = tooltip.getBoundingClientRect()
  const c = container.getBoundingClientRect()

  const fits = (x: number, y: number) =>
    x >= boundaryRect.left &&
    y >= boundaryRect.top &&
    Math.ceil(x + t.width) <= boundaryRect.right &&
    Math.ceil(y + t.height) <= boundaryRect.bottom

  const candidates: TCandidate[] = [
    { position: 'top', x: c.left, y: c.top - t.height },
    { position: 'right', x: c.right, y: c.top },
    { position: 'bottom', x: c.left, y: c.bottom },
    { position: 'left', x: c.left - t.width, y: c.top },
  ]

  return candidates.find(({ x, y }) => fits(x, y))?.position ?? 'top'
}
```

Tooltip: Result



Problem 6: Data Table

Build a Data Table component that supports displaying **structured data** with support for **sorting**, **pagination**, and **filtering**.

Requirements

1. **State management:** Accept afiguration array for columns, defining header names, ids, and custom renderers for cell content.
2. **Pagination:** Display specific number of rows per page. **Provide** “Next” and “Previous” buttons and display current page info (e.g., “1 / 5”).
3. **Sorting:** Allow clicking on headers to sort the table by that column (*Ascending* → *Descending* → None). Visual indicators (↑ / ↓) should show sort direction.
4. **Filtering:** Provide a search input to filter rows based on content.

Level: Medium

Problem 6: Preview

Symbol	Name	Price	Change	% Change	Volume	Market Cap	P/E Ratio
AAPL	Apple Inc.	$144.65	-5.35	-3.56%	33M	2.5T	25.5
GOOGL	Alphabet Inc.	$2901.00	+101.00	3.61%	81M	1.9T	28.1
MSFT	Microsoft Corp.	$293.85	-6.15	-2.05%	22M	2.3T	35.2
AMZN	Amazon.com Inc.	$3463.51	+63.51	1.87%	48M	1.7T	60.5
TSLA	Tesla Inc.	$787.21	+37.21	4.96%	5M	750B	120

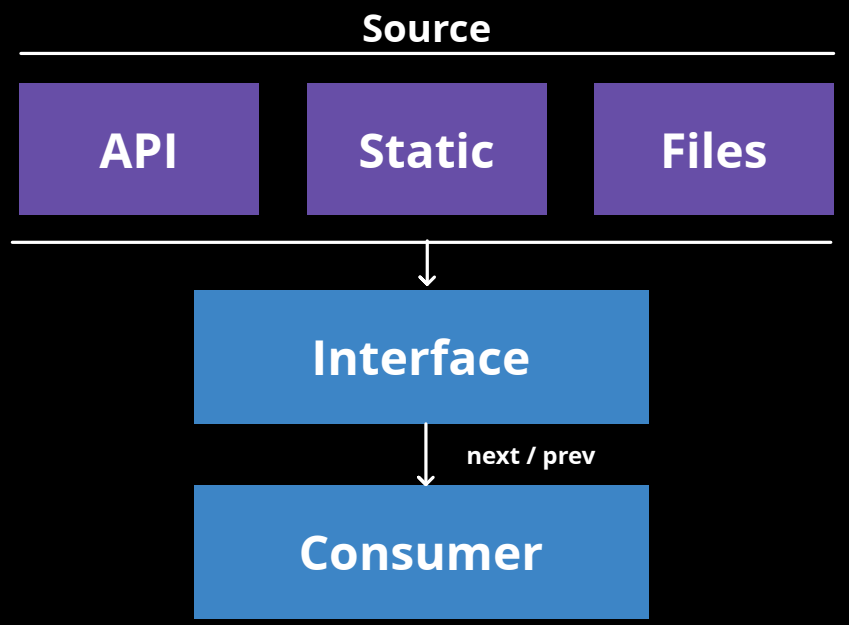
Prev 1 / 4 Next Filter

Data Table: API Design

1. `"columns": TTableColumn<T>[]` – Array of column definitions.
   - `"id": string` – Unique identifier.
   - `"name": string` – Header display text.
   - `"renderer": (item: T) => ReactNode` – Function to render cell content.
2. `"datasource": TTableDataSource<T>` – Data provider interface.
   - `"pageSize": number` – Number of items per page.
   - `"pages": number` – Total number of available pages.
   - `"next": (page: number, pageSize: number) => Promise<T[]>` – Async function to fetch data for a specific page.
3. `"search": (query: string, data: T[]) => T[]` – (Optional) Custom search filter function.
4. `"comparator": (columnId: keyof T, direction: SortDirection) => (a: T, b: T) => number` – (Optional) Custom sort comparator factory.

Data Table: Datasource Concept

```
1 interface TTableDataSource<T> {
2   pageSize: number
3   pages: number
4   next: (page: number, pageSize: number) => Promise<T[]>
5 }
```



Data Table: Basic types

```
1 export interface TTableDataSource<T> {
2   pageSize: number,
3   pages: number,
4   next: (page: number, pageSize: number) => Promise<T[]>
5 }
```

Data Table: Basic types

```
export type TTableColumn<T> = {
  id: string
  name: string
  renderer: (item: T) => React.ReactNode
  sort?: 'asc' | 'desc' | 'none'
}
```

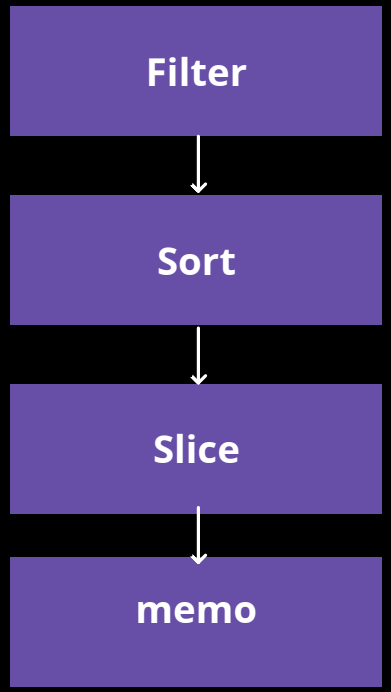
Data Table: Basic types

```
type TTableProps<T extends { id: string }> = {
  columns: TTableColumn<T>[]
  datasource: TTableDataSource<T>
  search?: (query: string, data: T[]) => T[],
  comparator?: (columnId: keyof T, direction: 'asc' | 'desc')  => (a: T, b: T) => number
}
```

Data Table: State

```
const [_query, setQuery] = useState('')
const query = useDeferredValue(_query);  // Defer search for smoother typing
const [data, setData] = useState<T[]>([])
const [currentPage, setCurrentPage] = useState(0);
const [sort, setSort]
          = useState<{ columnId: keyof T, direction: 'asc' | 'desc' | 'none' } | null>(null)
```

Data Table: Data Pipeline




```
1   const slice = useMemo(() => {
2     const filtered = query
3       ? search
4         ? search(query, data)
5         : data.filter((item) => item.id.includes(query))
6       : data
7     const sorted =
8       sort && comparator && sort.direction !== 'none'
9         ? [...filtered].sort(comparator(sort.columnId as keyof T, sort.direction))
10        : filtered
11    const start = currentPage * datasource.pageSize
12    return sorted.slice(start, start + datasource.pageSize)
13  }, [data, query, search, sort, comparator, currentPage, datasource])
```

Data Table: Data Fetching

```
useEffect(() => {
  if (data.length === 0) {
    datasource.next(0, datasource.pageSize).then(d => setData(d));
  }
}, [datasource]);

const next = useCallback(() => {
  if (currentPage >= datasource.pages - 1) return;
  const nextPage = currentPage + 1;
  setCurrentPage(nextPage);
  if (data.length <= nextPage * datasource.pageSize && data.length < datasource.pages * datasource.pageSize) {
    datasource.next(nextPage, datasource.pageSize).then(d => setData(prev => [...prev, ...d]))
  }
}, [datasource, currentPage, data.length])
```

Data Table: Sorting

```
const onSort: React.MouseEventHandler<HTMLTableSectionElement> = ({ target }) => {
  if (!(target instanceof HTMLElement) || !target.dataset.columnId) return
  const columnId = target.dataset.columnId as keyof T
  const column = columns.find((c) => c.id === columnId)
  if (!column) return
  setSort(prevSort => {
    const currentDirection = prevSort?.columnId === columnId ? prevSort.direction : (column.sort ?? 'none');
    const newDirection = currentDirection === 'desc' ? 'none' : currentDirection === 'asc' ? 'desc' : 'asc';
    return { columnId, direction: newDirection };
  });
}
```

Data Table: Search

```
1 const onSearch: ChangeEventHandler<HTMLInputElement> = ({ target }) => {
2   setQuery(target.value);
3   setCurrentPage(0);
4 };
```

Data Table: JSX

```jsx
return (
  <div className={styles.table}>
    <table>
      <thead onClickCapture={onSort}>
        <tr>
          {columns.map((c) => {
            const currentSort = sort?.columnId === c.id ? sort.direction : c.sort;
            return (
              <th data-column-id={c.id} key={c.id}>
                {c.name}
                {currentSort === 'asc' ? ' ↑' : currentSort === 'desc' ? ' ↓' : ''}
              </th>
            )
          })}
        </tr>
      </thead>
      <tbody>
        {slice.map((item) => (
          <tr key={item.id}>
            {columns.map((col) => (
              <td key={col.id}>{col.renderer(item)}</td>
            ))}
          </tr>
        ))}
      </tbody>
    </table>
    <div className={cx(flex.flexRowCenter, flex.flexGap8, styles.controls)}>
      <button disabled={currentPage === 0} onClick={prev}>Prev</button>
      <span>{currentPage + 1} / {datasource.pages}</span>
      <button disabled={currentPage === datasource.pages - 1} onClick={next}>Next</button>
      <input type="search" placeholder="Filter" onChange={onSearch} />
    </div>
  </div>
```

Challenge 14: Parameters

Implement the built-in `Parameters<T>` generic without using it.

```ts
const foo = (arg1: string, arg2: number): void => {}

type FunctionParamsType = MyParameters<typeof foo> // [string, number]
```

Challenge 14: Parameters | Solution

Implement the built-in `Parameters<T>` generic without using it.

```
type MyParameters<T extends (...args: any[]) => any> =
  T extends (...args: infer P) => any
                            ? P
                            : never
```

Challenge 15: Awaited

If we have a type which is wrapped like `Promise`, how can we get the type inside?

Unwrap nested promises recursively.

```
type ExampleType = Promise<string>
type Result = MyAwaited<ExampleType> // string

type Nested = Promise<Promise<string | number>>
type Result2 = MyAwaited<Nested> // string | number
```

Challenge 15: Awaited | Solution

If we have a type which is wrapped like `Promise`, how can we get the type inside?

Unwrap nested promises recursively.

```
type MyAwaited<T> = T extends { then: (onfulfilled: (arg: infer V) => any) => any }
  ? MyAwaited<V>
  : T
```

Problem 7: Reddit Thread

Build a recursive **comment thread component similar to Reddit**, supporting nested comments and collapse/expand functionality.

Requirements:

1. **Recursive Rendering:** Display comments and their nested replies in a hierarchical tree structure.
2. **Collapsible Threads:** Allow users to collapse and expand comment threads. Collapsing a parent comment should hide all its descendants.
3. **Indentation:** Visual indentation should clearly indicate the nesting level of comments.

Level: Medium

Reddit Thread: Preview

frontend_wizard 2 hours ago

This new React compiler is going to change everything! Finally we can stop worrying about useMemo everywhere.

▶ Replies

design_guru 3 hours ago

The UI looks clean but the contrast ratio on the buttons might be a bit low for accessibility.

▶ Replies

Reddit Thread: Data Shape

```
interface IRedditComment {
  id: string
  nickname: string
  text: string
  date: string
  replies: IRedditComment[]
}

const comments: IRedditComment[] = [
  {
    id: '1', nickname: 'user_one', date: '2h ago',
    text: 'This is a top-level comment',
    replies: [
      {
        id: '2', nickname: 'user_two', date: '1h ago',
        text: 'This is a reply',
        replies: [
          { id: '3', nickname: 'user_three', date: '30m ago',
            text: 'Nested reply!', replies: [] }
        ]
      }
    ]
  }
]
```

Reddit Thread: Comment JSX

```tsx
function RedditComment({ comment }: { comment: IRedditComment }) {
  return (
    <article className={cx(css.comment, flex.padding16)}>
      <header className={cx(flex.flexRowBetween)}>
        <strong>{comment.nickname}</strong>
        <time>{comment.date}</time>
      </header>
      <p>{comment.text}</p>
      {comment.replies.length > 0 && (
        <details>
          <summary>Replies</summary>
          <ul className={cx(flex.paddingLeft16, css.repliesList)}>
            {comment.replies.map((reply) => (
              <li key={reply.id}>
                <RedditComment comment={reply} />
              </li>
            ))}
          </ul>
        </details>
      )}
    </article>
  )
}
```

Reddit Thread: Thread JSX

```
export const RedditThreadComponent = ({ comments }: { comments: IRedditComment[] }) => {
  return (
    <div className={css.container}>
      {comments.map((comment) => (
        <RedditComment key={comment.id} comment={comment} />
      ))}
    </div>
  )
}
```

Problem 8: Gallery

Build an **image gallery** component that displays a list of images with support for navigation controls and thumbnail indicators.

Requirements:

1. **Image Display:** Display one image at a time from a provided list.
2. **Navigation:** Provide “Next” and “Previous” buttons to navigate through images. Buttons should be disabled when at the start or end of the list.
3. **Thumbnails/Indicators:** Display a list of indicators *(dots)* representing each image. Clicking an indicator should jump to that image.

Level: Medium

Gallery: Preview

Gallery: Approach

Key challenges:

- **Index clamping** (don't go below 0 or above length)
- **Keyboard navigation** (arrow keys)
- **Lazy loading** (don't load all images upfront)

Let's map it out:

1. **What do we need?** Show images one at a time from a provided list.
2. **Navigation:** Provide "Next" and "Previous" buttons to navigate through images. Buttons should be disabled when at the start or end of the list.
3. **Thumbnails/Indicators:** Display a list of indicators *(dots)* representing each image. Clicking an indicator should jump to that image.

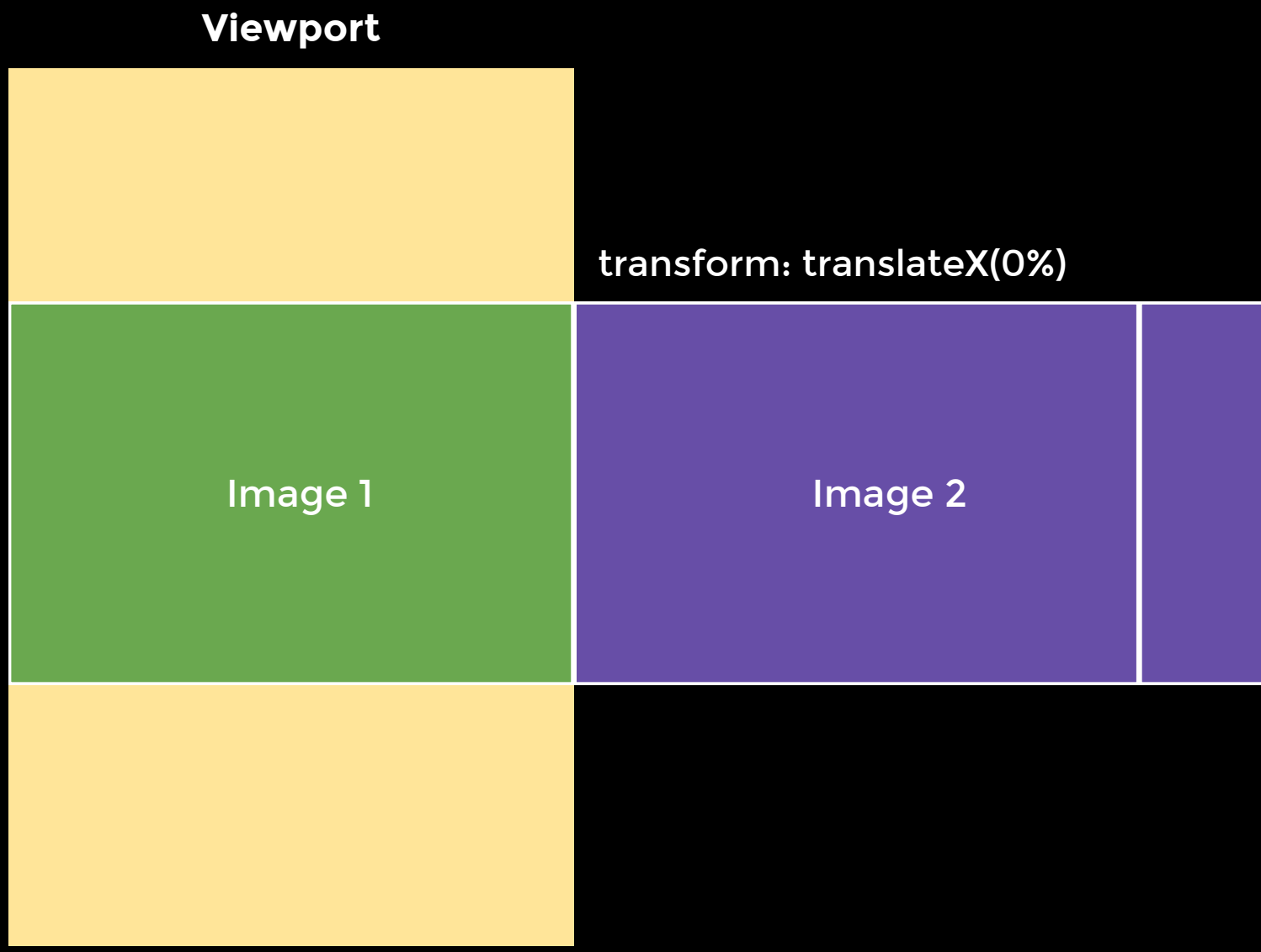
Gallery: State & Navigation

```tsx
export const Gallery = ({ images }: TGalleryProps) => {
  const [currentIndex, setCurrentIndex] = useState(0)

  const handlePrev = useCallback(() => {
    setCurrentIndex((prev) => Math.max(0, prev - 1))
  }, [])

  const handleNext = useCallback(() => {
    setCurrentIndex((prev) => Math.min(images.length - 1, prev + 1))
  }, [images.length])
```

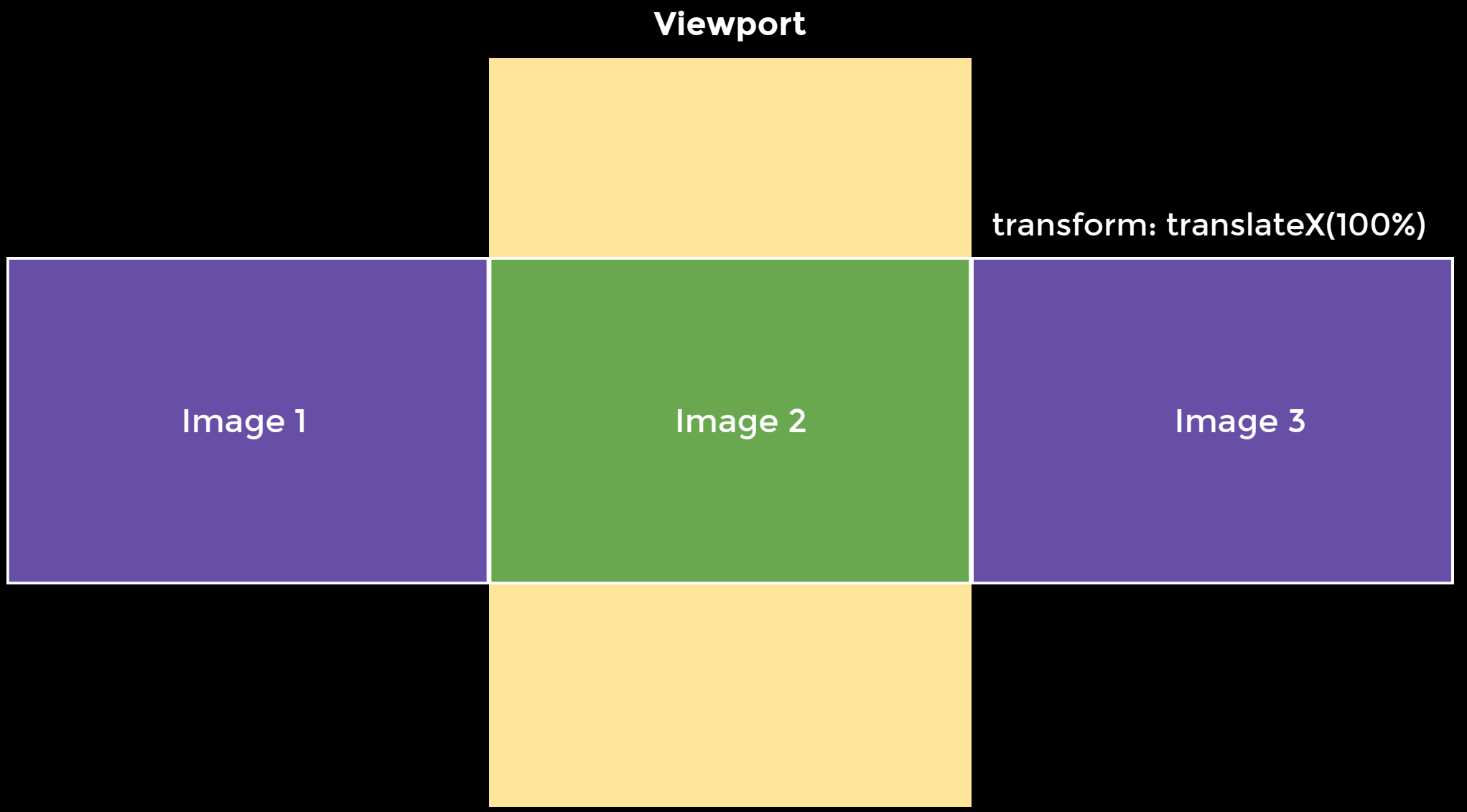
Gallery: Keyboard

```
useEffect(() => {
  const handleKeyDown = (e: KeyboardEvent) => {
    if (e.key === 'ArrowLeft') handlePrev()
    if (e.key === 'ArrowRight') handleNext()
  }

  window.addEventListener('keydown', handleKeyDown)
  return () => window.removeEventListener('keydown', handleKeyDown)
}, [handlePrev, handleNext])
```

Gallery: Rendering & Animation



Gallery: Rendering & Animation



Gallery: Rendering & Animation

```
return (
  <section className={cx(flex.w100, flex.pRel, css.container)}>
    <ul
      className={cx(flex.itemsStretch, flex.h100, css.list)}
      style={{ transform: `translateX(-${currentIndex * 100}%)` }}
    >
      {images.map((image, i) => (
        <li key={i} className={cx(flex.wh100, css.item)}>
          <img
            src={currentIndex + 2 >= i ? image : undefined}
            alt={`Gallery image ${i + 1}`}
          />
        </li>
      ))}
    </ul>
  </section>
)
```

Gallery: Result

Challenge 16: Last

Implement a generic `Last<T>` that takes an Array `T` and returns its last element.

```
type arr1 = ['a', 'b', 'c']
type arr2 = [3, 2, 1]

type tail1 = Last<arr1> // 'c'
type tail2 = Last<arr2> // 1
```

Challenge 16: Last | Solution

Implement a generic `Last<T>` that takes an Array `T` and returns its last element.

```
type Last<T extends any[]> = T extends [...any[], infer L] ? L : never
```

Challenge 16: Last | Solution

Implement a generic `Last<T>` that takes an Array `T` and returns its last element.

```
type Last<T extends any[]> = T extends [...any[], infer L] ? L : never
```

Challenge 17: Flatten

Write a type that takes an array and returns the flattened array type (one level deep).

```
1 type Last<T extends any[]> = T extends [...any[], infer L] ? L : never
```

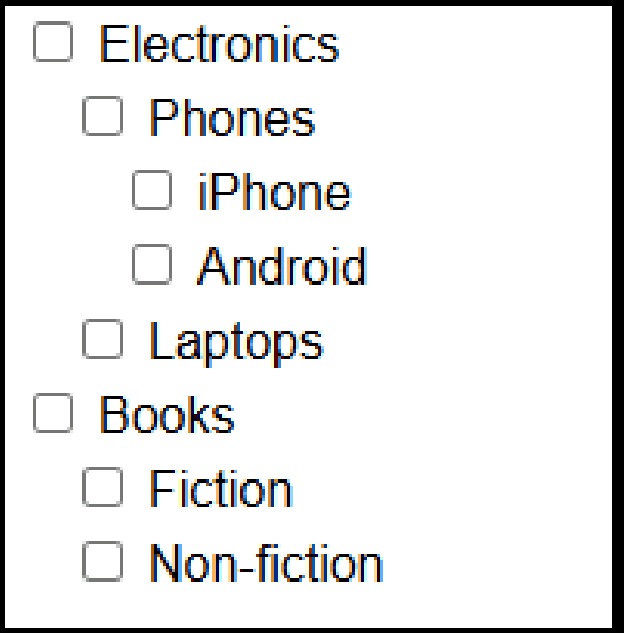
Challenge 17: Flatten | Solution

Write a type that takes an array and returns the flattened array type (one level deep).

```
type Flatten<T extends any[]> = T extends [infer First, ...infer Rest]
  ? First extends any[]
    ? [...Flatten<First>, ...Flatten<Rest>]
    : [First, ...Flatten<Rest>]
  : []
```

Problem 9: Nested Checkboxes

```ts
const MOCK_DATA: TCheckboxItem[] = [
  {
    id: '1',
    label: 'Electronics',
    children: [
      {
        id: '1-1',
        label: 'Phones',
        children: [
          { id: '1-1-1', label: 'iPhone' },
          { id: '1-1-2', label: 'Android' },
        ],
      },
      { id: '1-2', label: 'Laptops' },
    ],
  },
  {
    id: '2',
    label: 'Books',
    children: [
      { id: '2-1', label: 'Fiction' },
      { id: '2-2', label: 'Non-fiction' },
    ],
  },
]
```



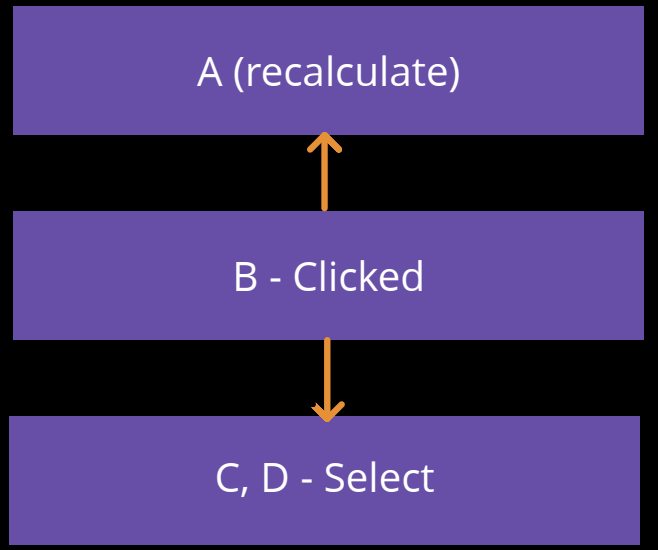
Level: Hard

Approach: Nested checkboxes

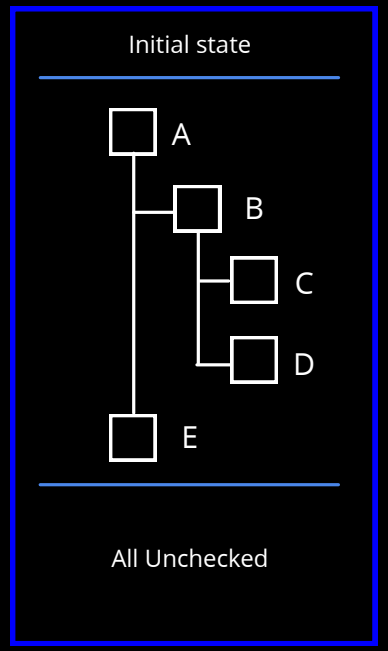
3. Bubble UP

1. Set status

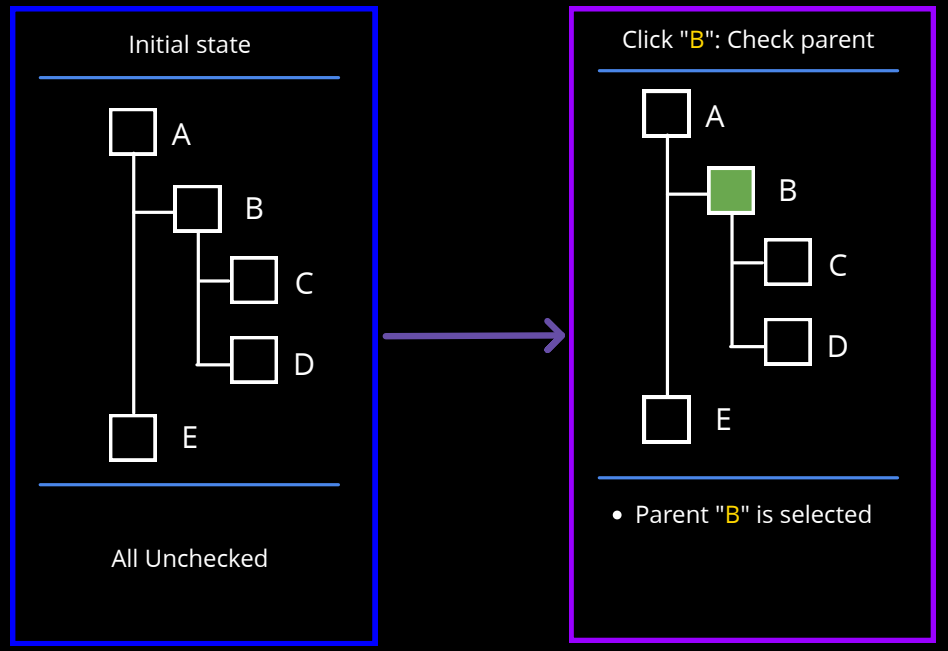
2. Propagate Down



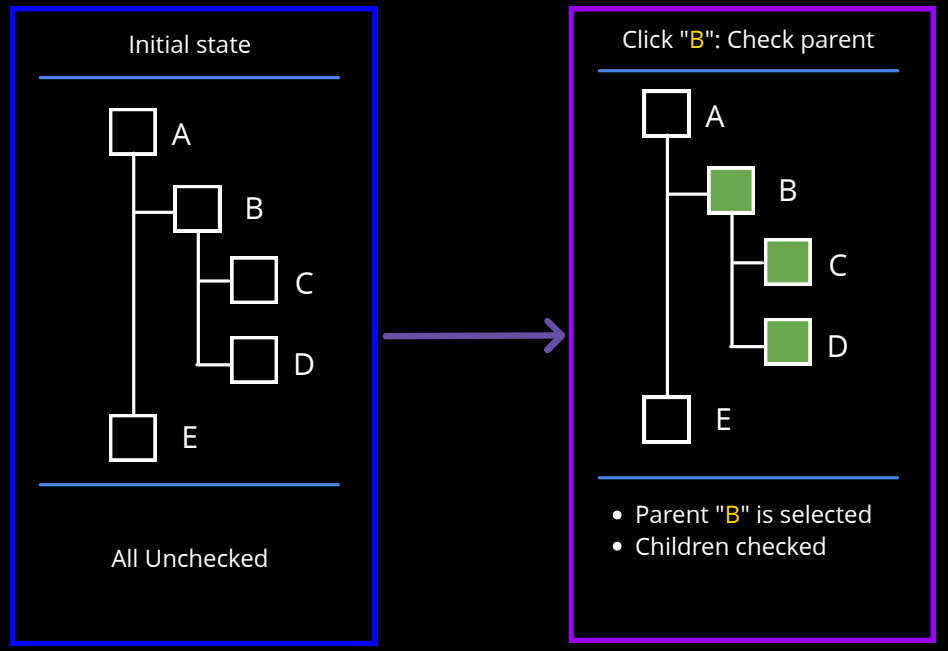
Approach: Nested checkboxes



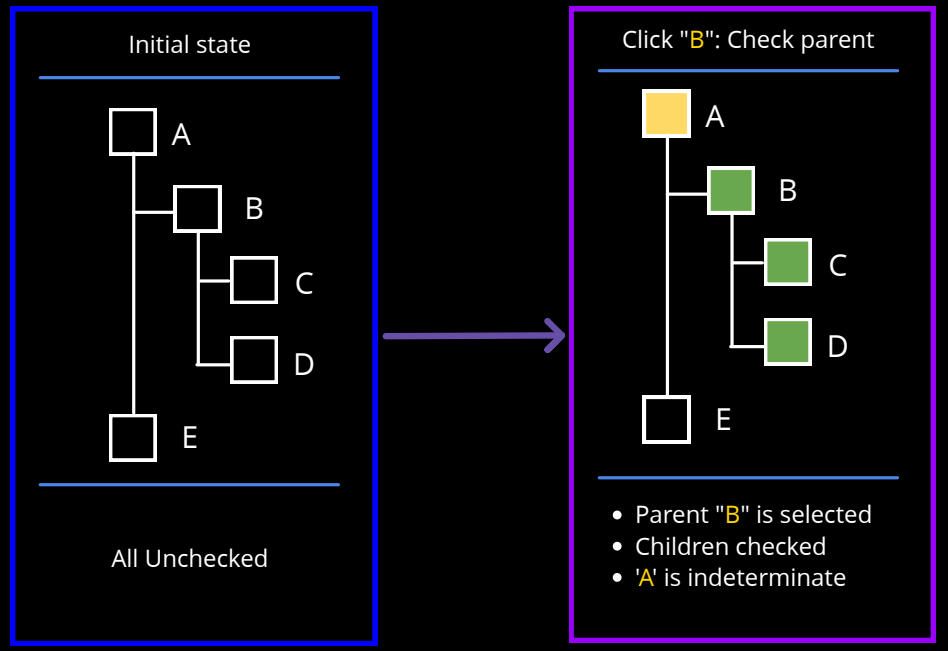
Approach: Nested checkboxes



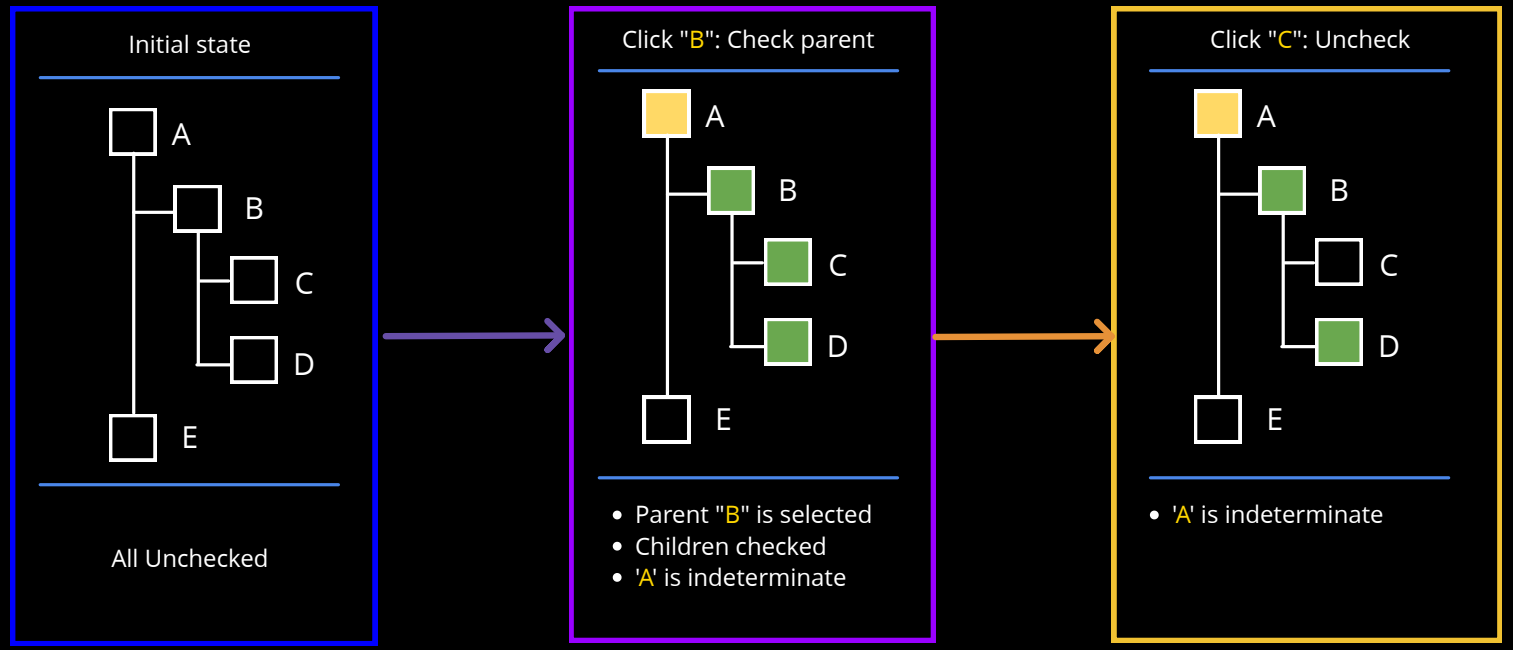
Approach: Nested checkboxes



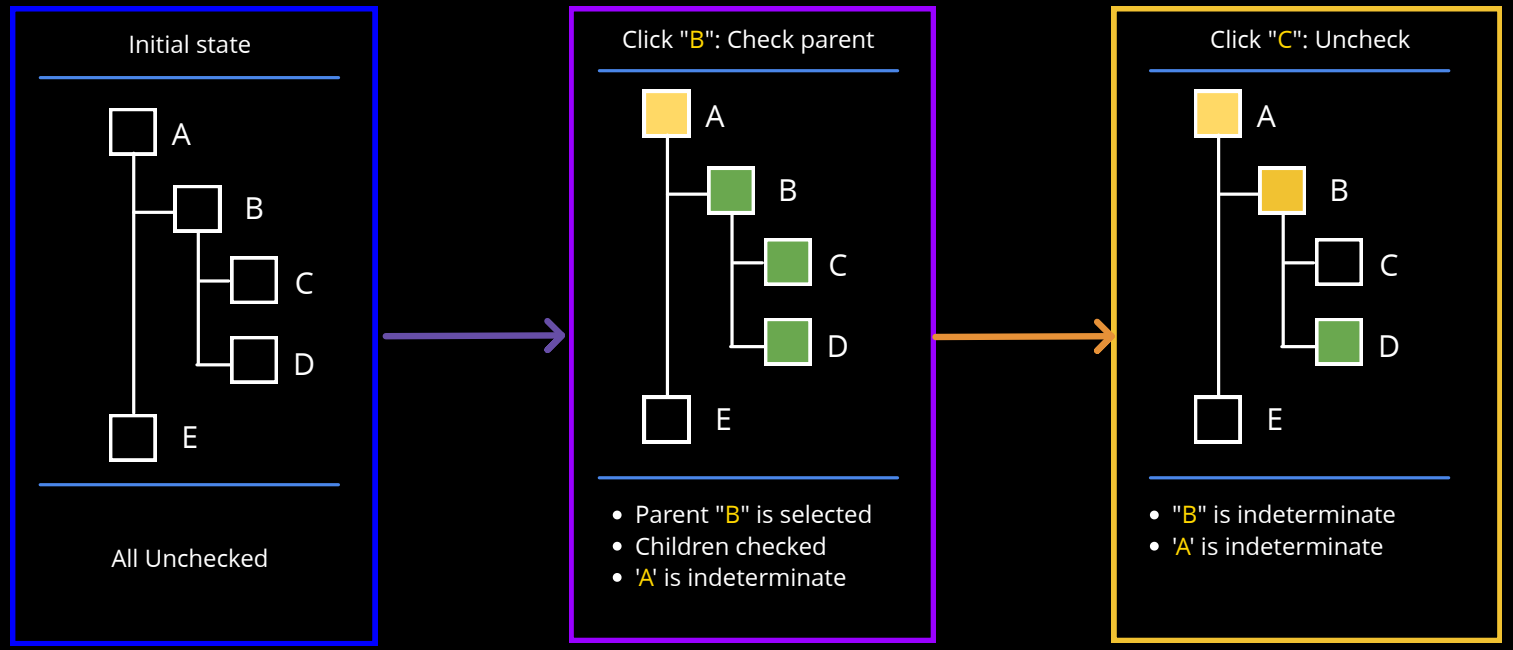
Approach: Nested checkboxes



Approach: Nested checkboxes



Approach: Nested checkboxes

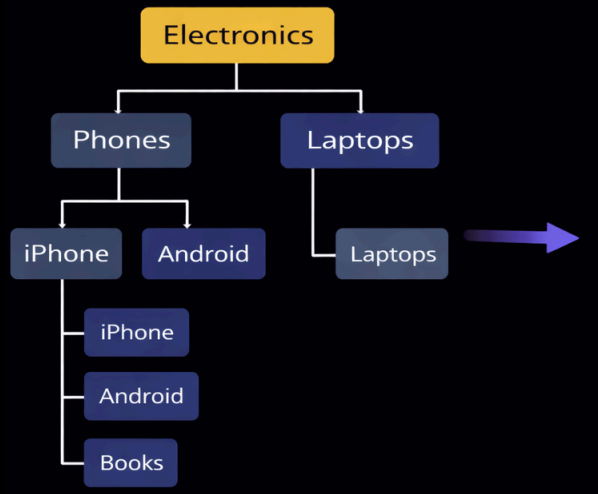


Solving: Nested checkboxes - Basic types

```ts
export type TCheckboxItem = {
  id: string
  label: string
  parent?: TCheckboxItem
  selected?: boolean
  indeterminate?: boolean
  children?: TCheckboxItem[]
}

type CheckboxTreeProps = {
  items: TCheckboxItem[]
}
```

Solving: Nested checkboxes - Data Flattening

```ts
const MOCK_DATA: TCheckboxItem[] = [
  {
    id: '1',
    label: 'Electronics',
    children: [
      {
        id: '1-1',
        label: 'Phones',
        children: [
          { id: '1-1-1', label: 'iPhone' },
          { id: '1-1-2', label: 'Android' },
        ],
      },
      { id: '1-2', label: 'Laptops' },
    ],
  },
  {
    id: '2',
    label: 'Books',
    children: [
      { id: '2-1', label: 'Fiction' },
      { id: '2-2', label: 'Non-fiction' },
    ],
  },
]
```



Flat Map

1 → Electronics
1-1 → Phones
1-1-1 → iPhone
1-1-2 → Android
1-2 → Laptops
2 → Books
2-1 → Fiction
2-2 → Non-fiction

Solving: Nested checkboxes - Data Flattening

```ts
const MOCK_DATA: TCheckboxItem[] = [
  {
    id: '1',
    label: 'Electronics',
    children: [
      {
        id: '1-1',
        label: 'Phones',
        children: [
          { id: '1-1-1', label: 'iPhone' },
          { id: '1-1-2', label: 'Android' },
        ],
      },
      { id: '1-2', label: 'Laptops' },
    ],
  },
  {
    id: '2',
    label: 'Books',
    children: [
      { id: '2-1', label: 'Fiction' },
      { id: '2-2', label: 'Non-fiction' },
    ],
  },
]
```

```ts
function process(acc: Record<string, TCheckboxItem>, item: TCheckboxItem, parent?: TChe
  acc[item.id] = item
  item.parent = parent
  item.selected = item.selected || !!parent?.selected

  item.children?.forEach((child) => process(acc, child, item))

  if (item.children?.length) {
    const allChecked = item.children.every((it) => it.selected)
    const someChecked = item.children.some((it) => it.selected || it.indeterminate)

    item.selected = allChecked
    item.indeterminate = !allChecked && someChecked
  }

  return acc
}
```

Solving: Nested checkboxes - Rendering

```tsx
function Checkbox({ label, children, id, selected, indeterminate }: TCheckboxItem) {
  const ref = useRef<HTMLInputElement>(null)

  useEffect(() => {
    if (ref.current) {
      ref.current.indeterminate = indeterminate ?? false
    }
  }, [indeterminate])

  return (
    <>
      <label>
        <input
          ref={ref}
          checked={selected ?? false}
          data-id={id}
          id={id}
          type="checkbox"
          readOnly
        />
        <span className={cx(flex.paddingLeft8)}>{label}</span>
      </label>
      {children && children.length > 0 && (
        <ul className={cx(flex.paddingLeft16)}>
          {children.map((item) => (
            <li key={item.id}>
              <Checkbox {...item} />
            </li>
          ))}
        </ul>
      )}
    </>
  )
}
```

Solving: Nested checkboxes - Intermediate state

```
<!-- This doesn't work: -->
<input type="checkbox" indeterminate />  <!-- ❌ Not a valid attribute! -->
```

```
  useEffect(() => {
    if (ref.current) {
      ref.current.indeterminate = indeterminate ?? false
    }
  }, [indeterminate])
```

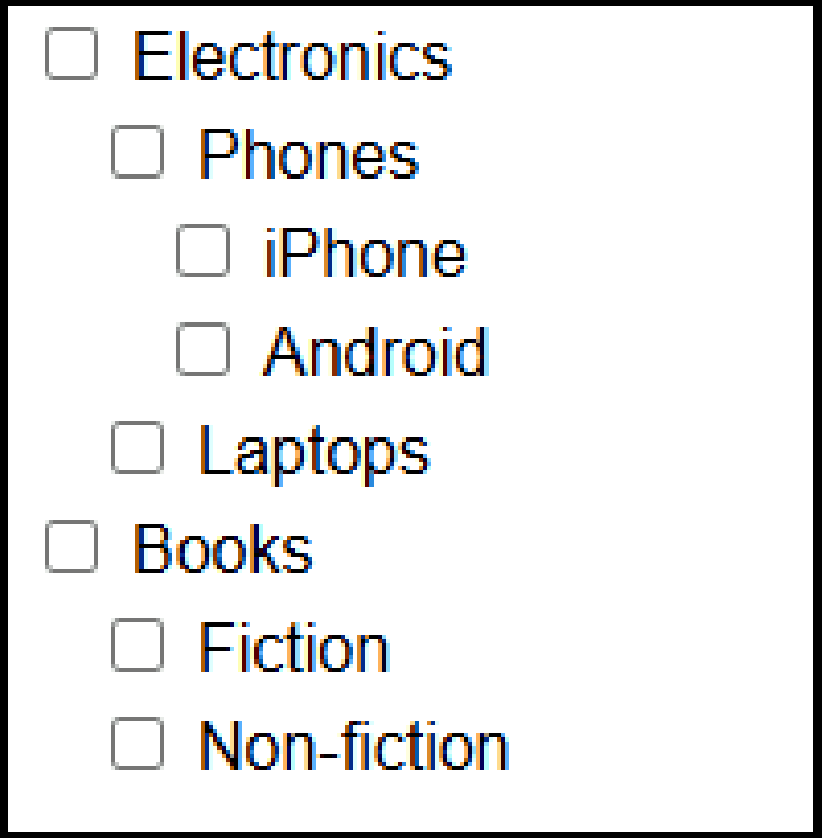
Solving: Nested checkboxes - Propagation

```
1 function propagate(children: TCheckboxItem[], value: boolean) {
2   for (const child of children) {
3     child.selected = value
4     child.indeterminate = false
5     propagate(child.children ?? [], value)
6   }
7 }
```

Solving: Nested checkboxes - Bubble Up

```
function bubble(state: TCheckboxTreeState, target: TCheckboxItem) {
  if (target == null || target?.parent == null) return
  const parent = target?.parent

  const children = parent.children ?? []
  const allChecked = children.every((it) => it.selected)
  const someChecked = children.some((it) => it.selected || it.indeterminate)

  parent.selected = allChecked
  parent.indeterminate = !allChecked && someChecked

  bubble(state, parent)
}
```

Nested checkboxes - Result



Problem 10 - Toast

Create a notification system that displays **brief**, **auto-expiring messages** to the user. Toasts should appear non-intrusively and automatically dismiss themselves after a set duration.

Requirements:

1. **Display:** Show a toast message with the provided text content.
2. **Positioning:** Toasts should stack (usually at the bottom or top of the screen).
3. **Auto-dismiss:** Messages should disappear automatically after a configured timeoutt (e.g.3000ms).
4. **Animation:** New toasts should fade in/Slide in, and expiring toasts should fade out/slide out.
4. **Queueing:** Multiple toasts should stack comfortably without overlapping.

API Design

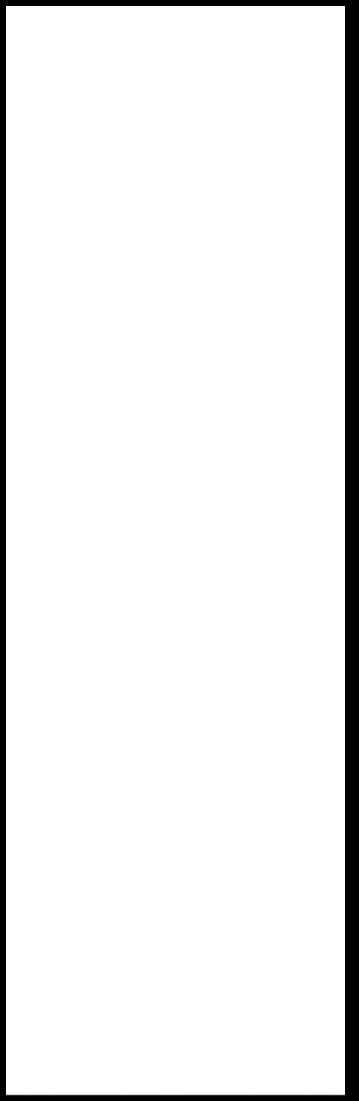
The component should support an imperative API for triggering toasts, for example:

```
toast({
  text: 'Operation successful',
  duration: 3000,
})
```

Level: Hard

Toast: Preview

```
toast({
  text: 'Operation successful',
  duration: 3000,
})
```



Toast: Solution approach

Let's plan:

1. **What do we need?** Show a message, auto-dismiss with animation, stack multiple toasts
2. **Data shape?** A state array of `{ id, text, removed }` objects
3. **The pattern?** **AnimationEnd** for animation lifecycle
4. **Accessibility?** `role="status"` with `aria-live="polite"` for screen reader announcements

Define Types

Toast: Solution approach

Let's plan:

1. **What do we need?** Show a message, auto-dismiss with animation, stack multiple toasts
2. **Data shape?** A state array of `{ id, text, removed }` objects
3. **The pattern?** **AnimationEnd** for animation lifecycle
4. **Accessibility?** `role="status"` with `aria-live="polite"` for screen reader announcements

Define Types

Template

Toast: Solution approach

Let's plan:

1. **What do we need?** Show a message, auto-dismiss with animation, stack multiple toasts
2. **Data shape?** A state array of `{ id, text, removed }` objects
3. **The pattern?** **AnimationEnd** for animation lifecycle
4. **Accessibility?** `role="status"` with `aria-live="polite"` for screen reader announcements

Define Types | **Template** | **API**

Toast: Solution approach

Let's plan:

1. **What do we need?** Show a message, auto-dismiss with animation, stack multiple toasts
2. **Data shape?** A state array of `{ id, text, removed }` objects
3. **The pattern?** **AnimationEnd** for animation lifecycle
4. **Accessibility?** `role="status"` with `aria-live="polite"` for screen reader announcements

Define Types | **Template** | **API** | **Event Handleing**

Toast: Solution approach

Let's plan:

1. **What do we need?** Show a message, auto-dismiss with animation, stack multiple toasts
2. **Data shape?** A state array of `{ id, text, removed }` objects
3. **The pattern?** **AnimationEnd** for animation lifecycle
4. **Accessibility?** `role="status"` with `aria-live="polite"` for screen reader announcements

Define Types | **Template** | **API** | **Event Handleing** | **DOM cleanup**

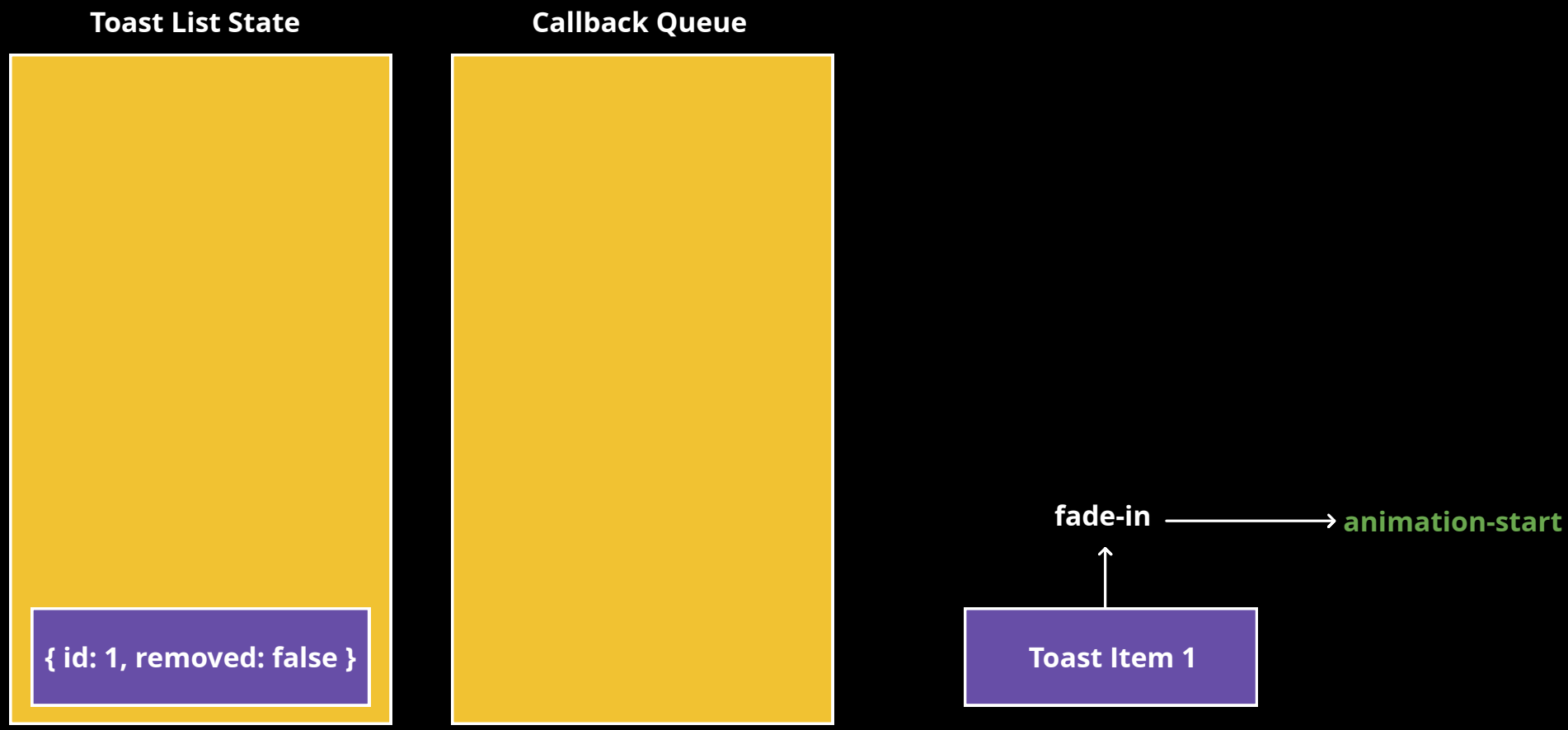
Toast: how to use component

```
// Inside any child component, use the hook
const component = new Toast({ root: document.body })

toast({ id: '1', text: 'Settings saved successfully!' })
toast({ id: '2', text: 'Error connecting to server.' })
```

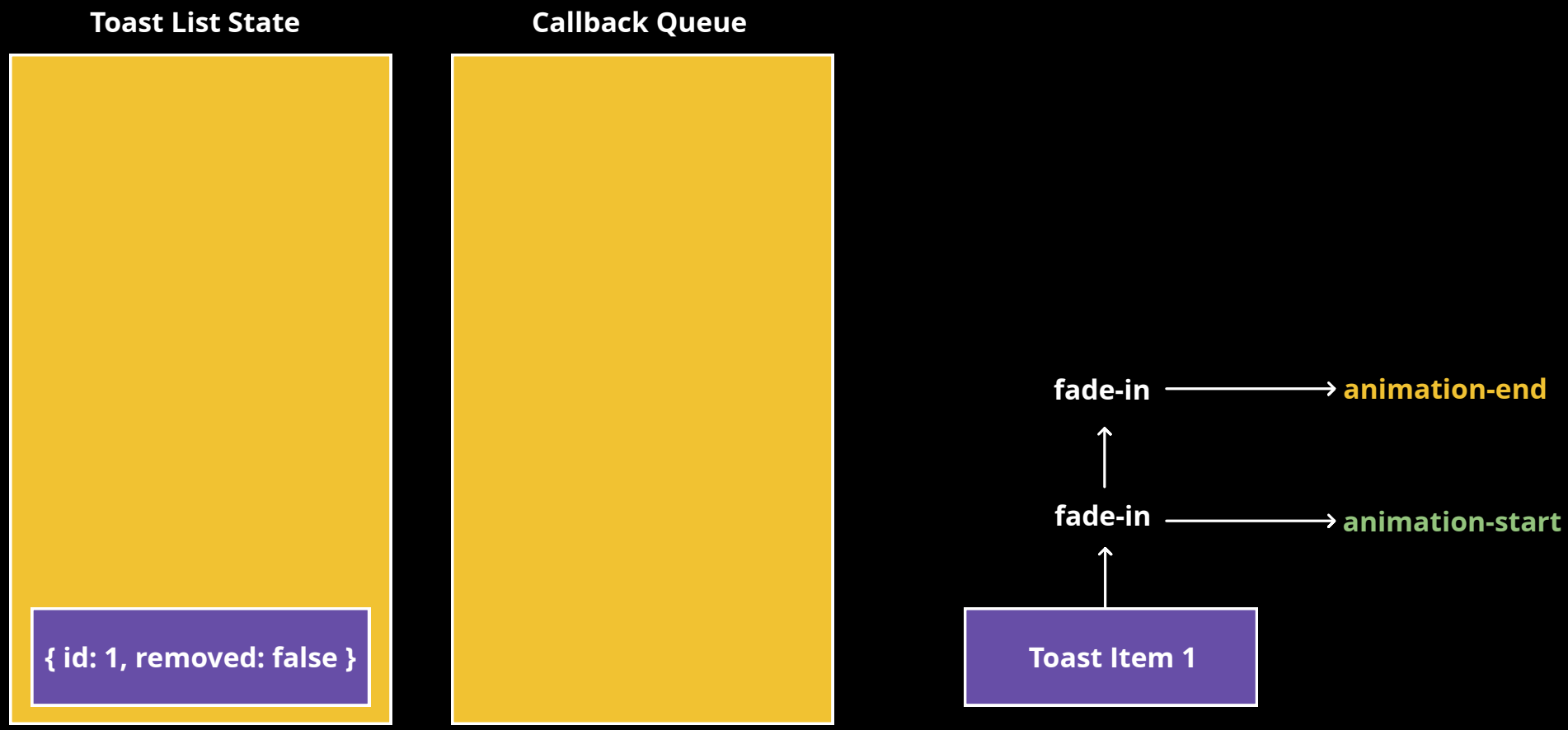
Toast: Define types

```
1 const TIMER = 3000
2
3 type TToastItem = {
4   id: string
5   text: string
6 }
```

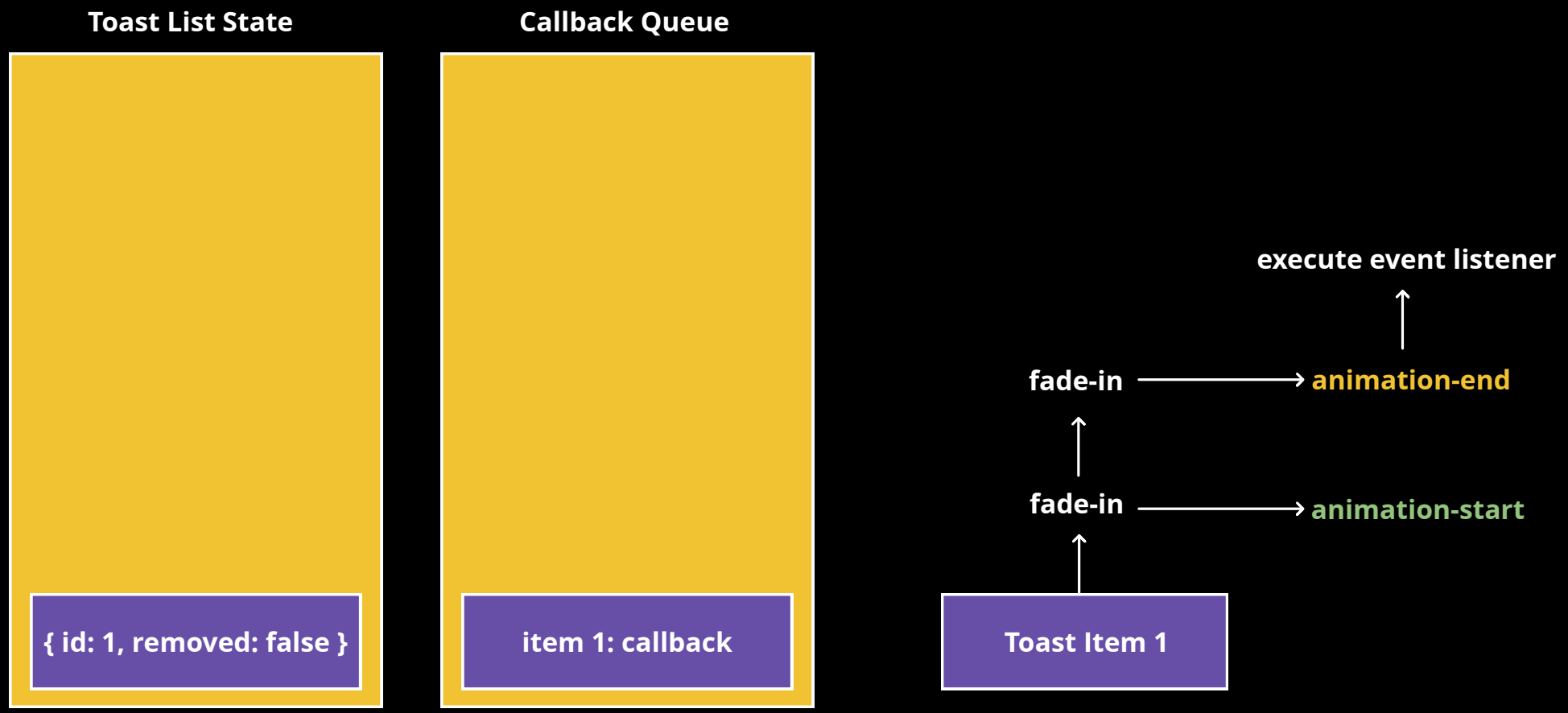
Toast: Animation Lifecycle: Fade In



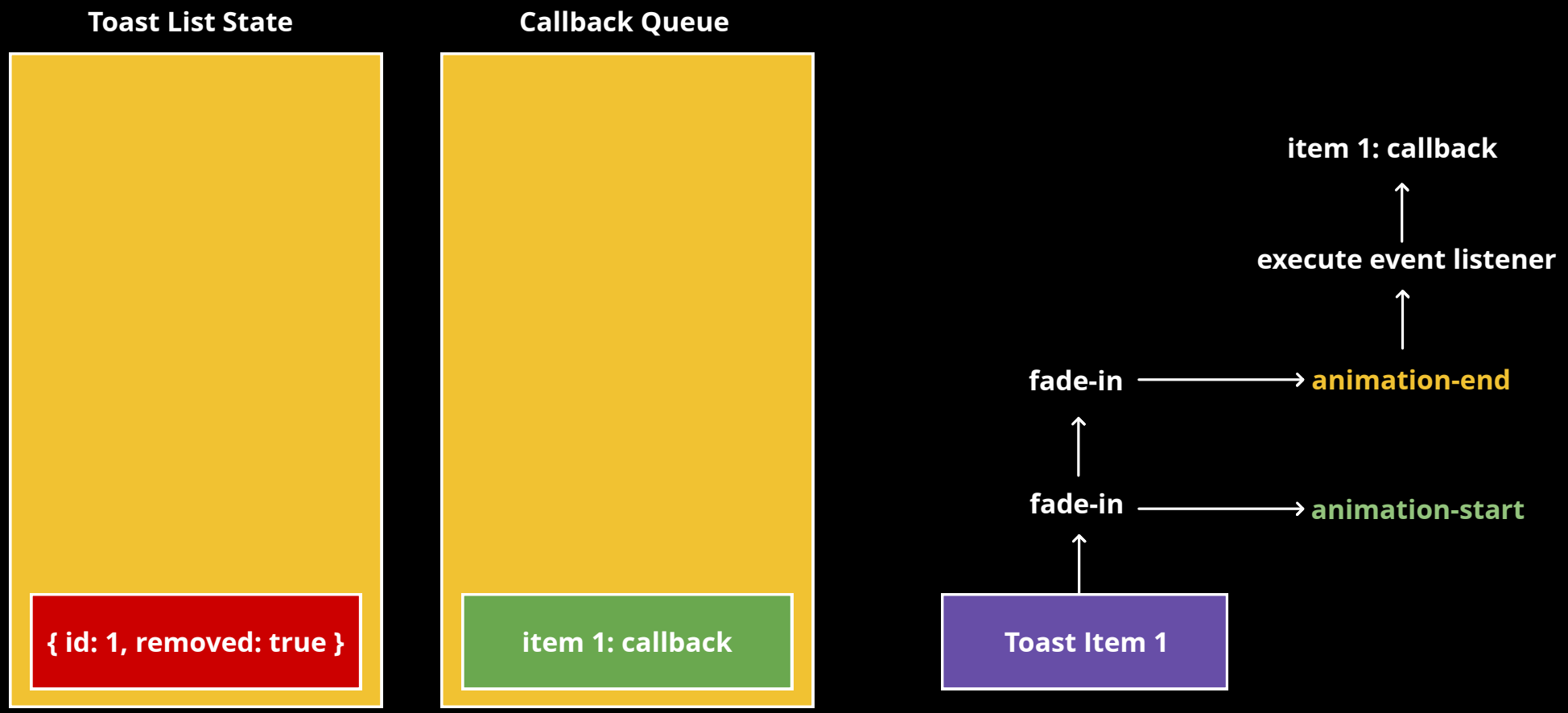
Toast: Animation Lifecycle: Fade In



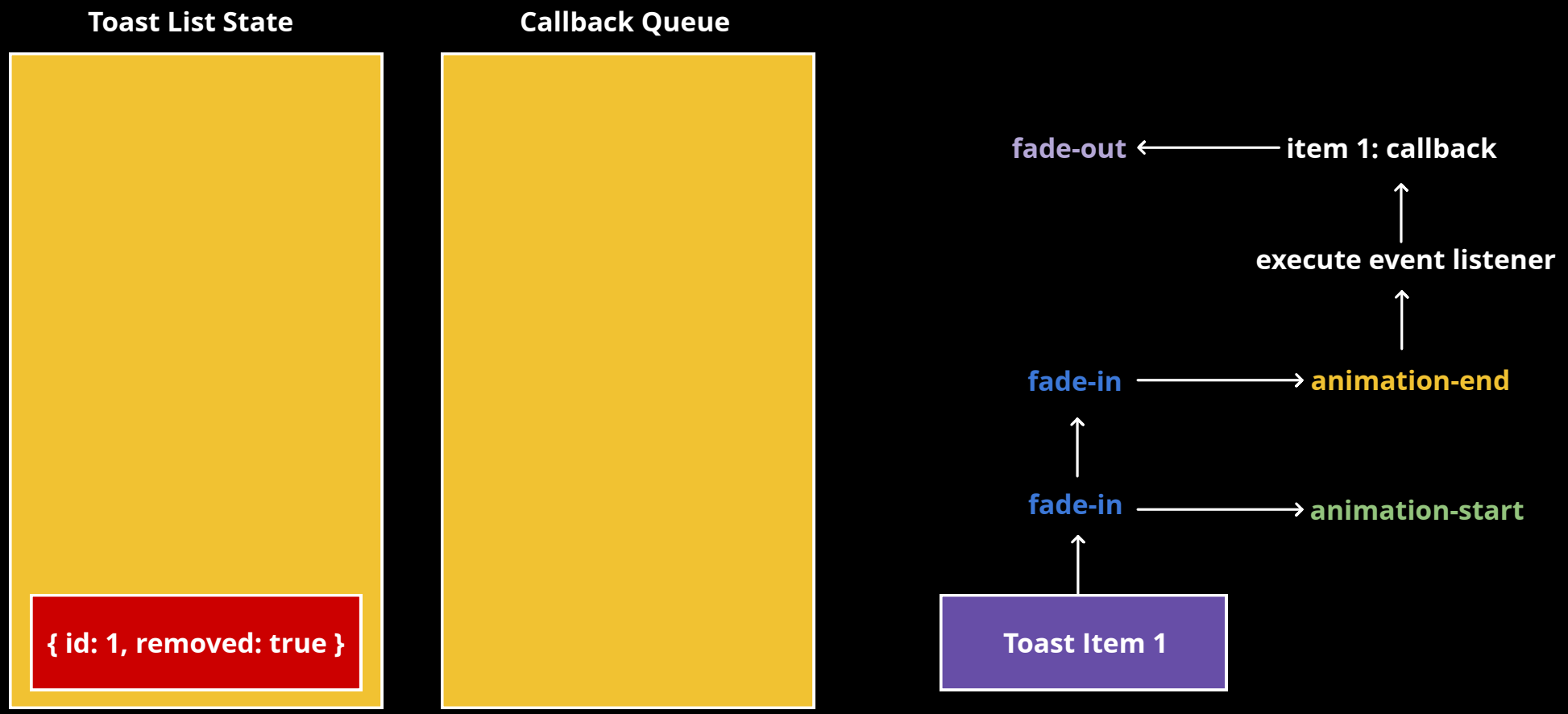
Toast: Animation Lifecycle: Fade In



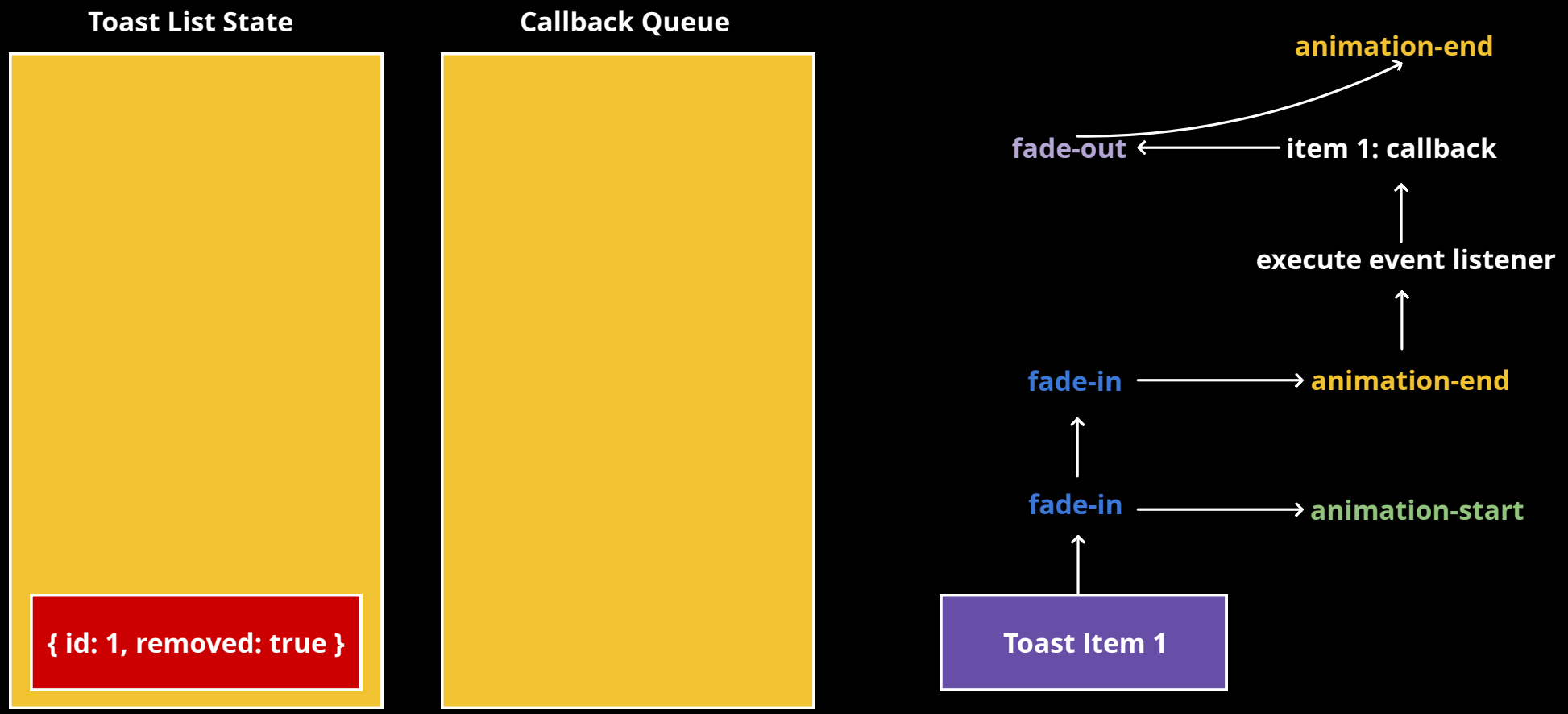
Toast: Animation Lifecycle: Fade In



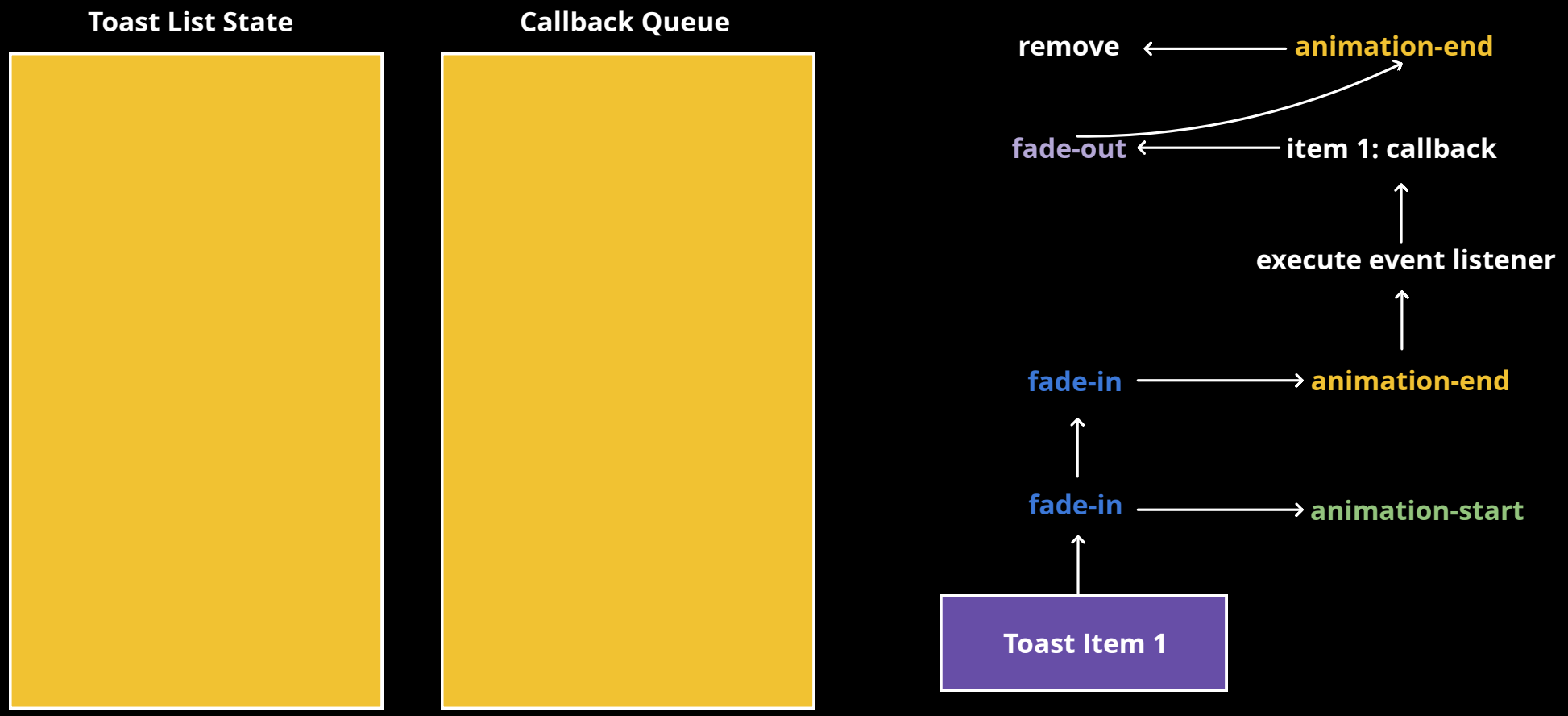
Toast: Animation Lifecycle: Fade Out



Toast: Animation Lifecycle: Fade Out



Toast: Animation Lifecycle: Fade Out



Toast: Template

```
1   toHTML() {
2     return `<ul id="toast-instance-${this.id}"
3                 aria-live="polite"
4                 aria-relevant="additions removals"
5                 class="${cx(flex.flexColumnStart, css['toast-list'])}"></ul>`
6   }
7
8   getToastTemplate(item: TToastItem) {
9     return `<li role="status" aria-atomic="true"
10                             aria-live="polite"
11                             key="${item.id}"
12                             data-removed="false"
13                             data-id="${item.id}"
14                             class="${css.fadeIn}">
15                    <div class="${cx(flex.flexColumnCenter, css.toast)}">
16                        <p>${item.text}</p>
17                    </div>
18                </li>`
19  }
20
21  afterRender() {
22    this.listElement = document.getElementById(`toast-instance-${this.id}`) as HTMLULListEleme
23  }
```

Toast: Animation Lifecycle

```
toast(item: TToastItem) {
  const element = document.createElement('div')
  element.innerHTML = this.getToastTemplate(item)
  const toastElement = element.firstElementChild as HTMLLIElement
  this.listElement?.appendChild(toastElement)

  setTimeout(() => {
    toastElement.classList.remove(css.fadeIn)
    toastElement.classList.add(css.fadeOut)
    toastElement.dataset.removed = 'true'
  }, 3000)
}

onAnimationend({ target }: AnimationEvent) {
  if (target instanceof HTMLElement && target.dataset.removed === 'true') {
    target.remove()
  }
}
```

Toast: Rendering

```
  return (
    <ul
      aria-live="polite"
      aria-relevant="additions removals"
      className={cx(css['toast-list'], flex.flexColumnStart)}
      onAnimationEndCapture={onAnimationEnd}
    >
      {items.map((it) => (
        <li
          role="status"
          aria-atomic="true"
          aria-live="polite"
          key={it.id}
          data-removed={it.removed}
          data-id={it.id}
          className={`${it.removed ? css.fadeOut : css.fadeIn}`}
        >
          <div className={cx(css.toast, flex.flexColumnCenter)}>
            <p>{it.text}</p>
          </div>
        </li>
      ))}
    </ul>
  )
}
```

Challenge 18: Capitilise

Implement `Capitalize<T>` which converts the first letter of a string to uppercase and leaves the rest as-is.

```
1 type Result = MyCapitalize<'hello'> // 'Hello'
```

Challenge 18: Capitilise | Solution

Implement `Capitalize<T>` which converts the first letter of a string to uppercase and leaves the rest as-is.

```
type MyCapitalize<S extends string> = S extends `${infer C}${infer Rest}`
  ? `${Uppercase<C>}${Rest}`
  : S
```

Challenge 19: TrimLeft

Implement `TrimLeft<T>` which takes an exact string type and returns a new string with the whitespace beginning removed.

```
1 type trimmed = TrimLeft<'  Hello World  '> // 'Hello World  '
```

Challenge 19: Trimleft | Solution

Implement `TrimLeft<T>` which takes an exact string type and returns a new string with the whitespace beginning removed.

```ts
type Space = ' ' | '\t' | '\n'

type TrimLeft<S extends string> = S extends `${Space}${infer T}` ? TrimLeft<T> : S
```

Part 2.1 - Completed

Part 2.2 - Advanced Components

Problem 11 - Calculator

Implement a functional **calculator** that can perform basic arithmetic operations (addition, subtraction, multiplication, division).

Core Functionality

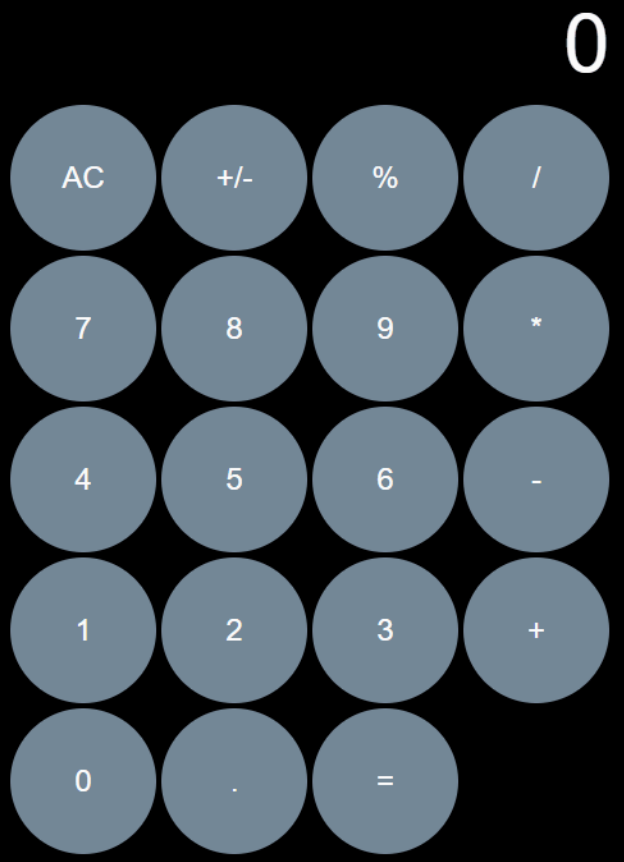
1. **Display:** Show current input and result.
2. **Input:** Buttons for digits (0–9) and operations (+, –, *, /).
3. **Advanced Operations:** Support for Modulo (%) and **Negation** (+/–).
4. **Decimals:** Support for decimal point inputs.
5. **Calculation:** Perform calculation on "=" press.
6. **Clear:** Button to clear current input ("AC").

Accessibility (A11y)

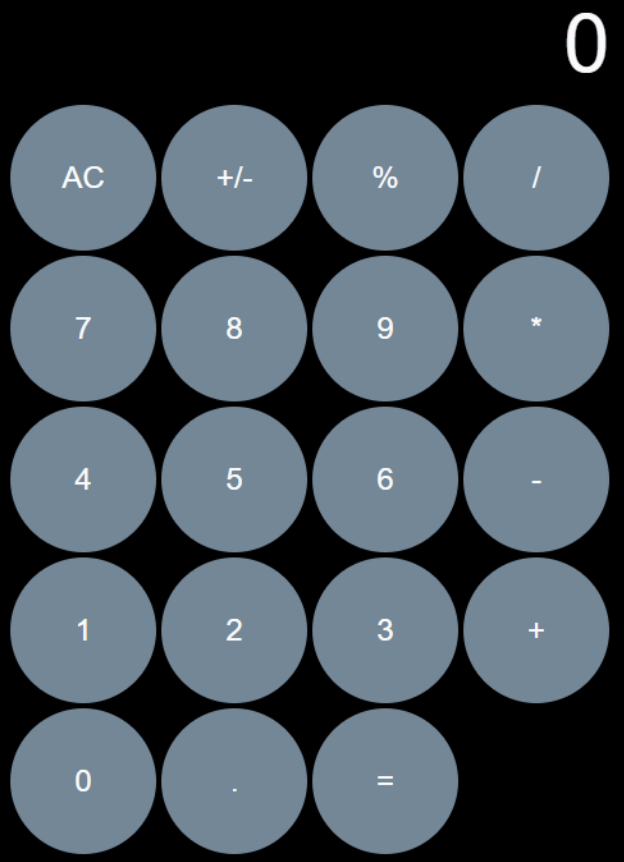
- Buttons should be **focusable** and **operable** via keyboard.
- Output should be announced correctly (using 'output' tag).

Level: Hard

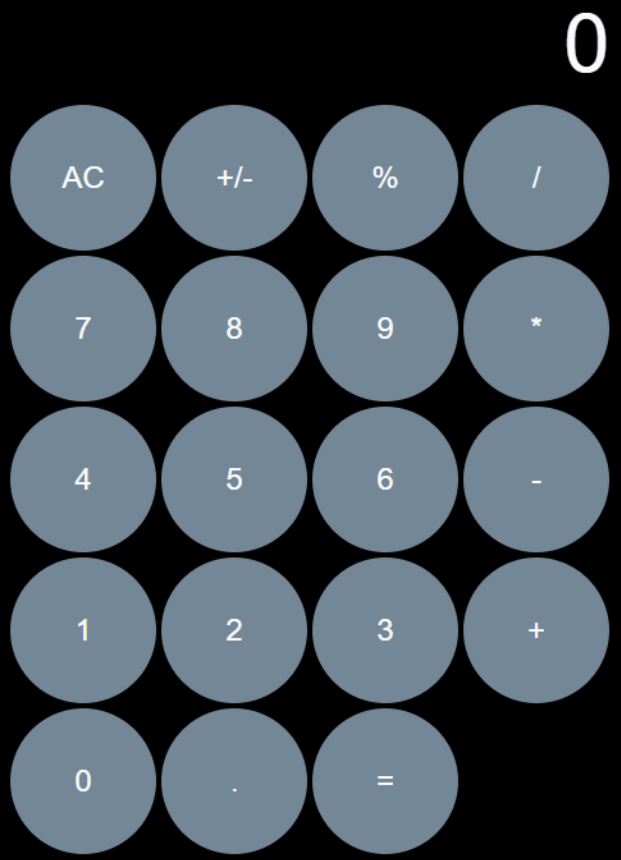
Problem 11: Preview



Problem 11: Preview

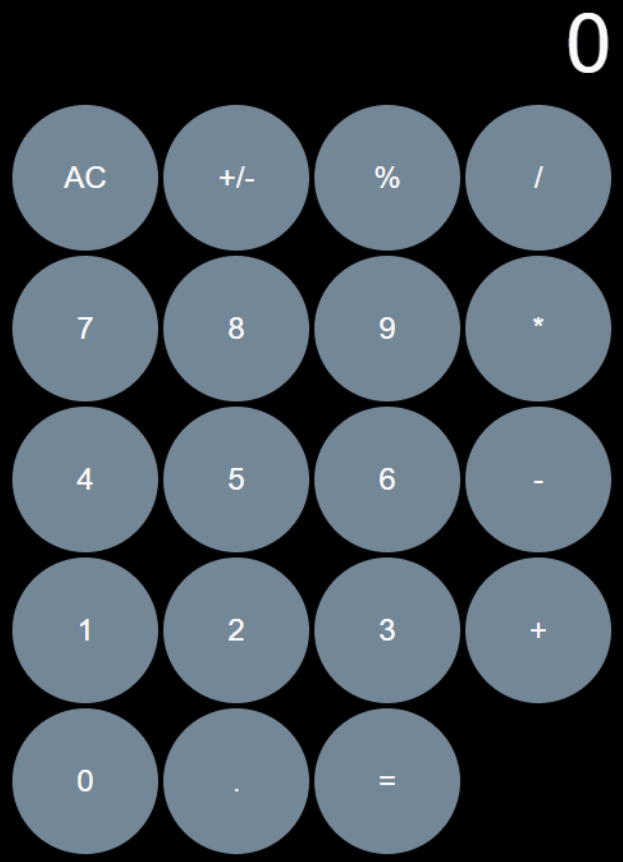


Problem 11: Solution Approach



output

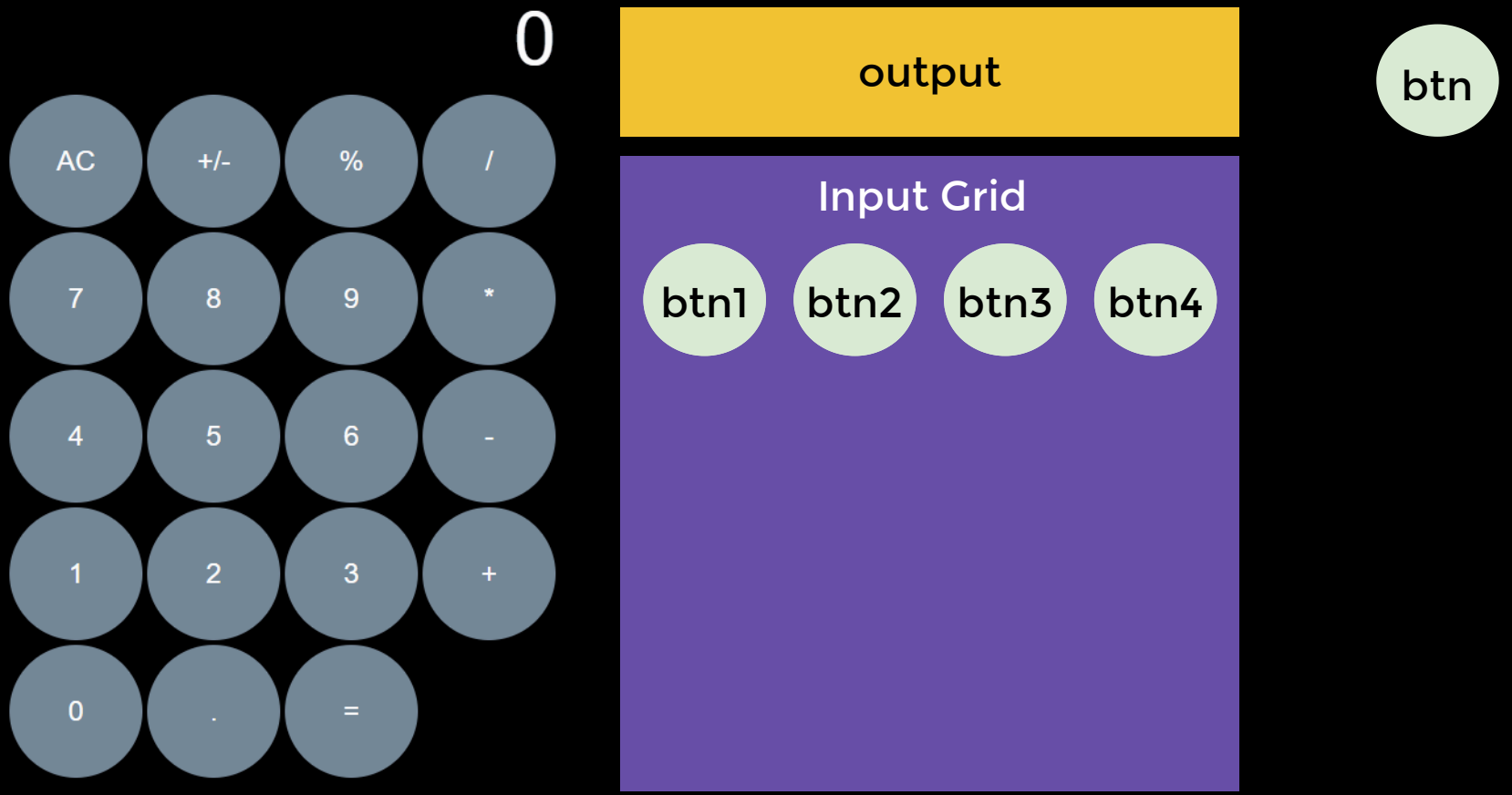
Problem 11: Solution Approach



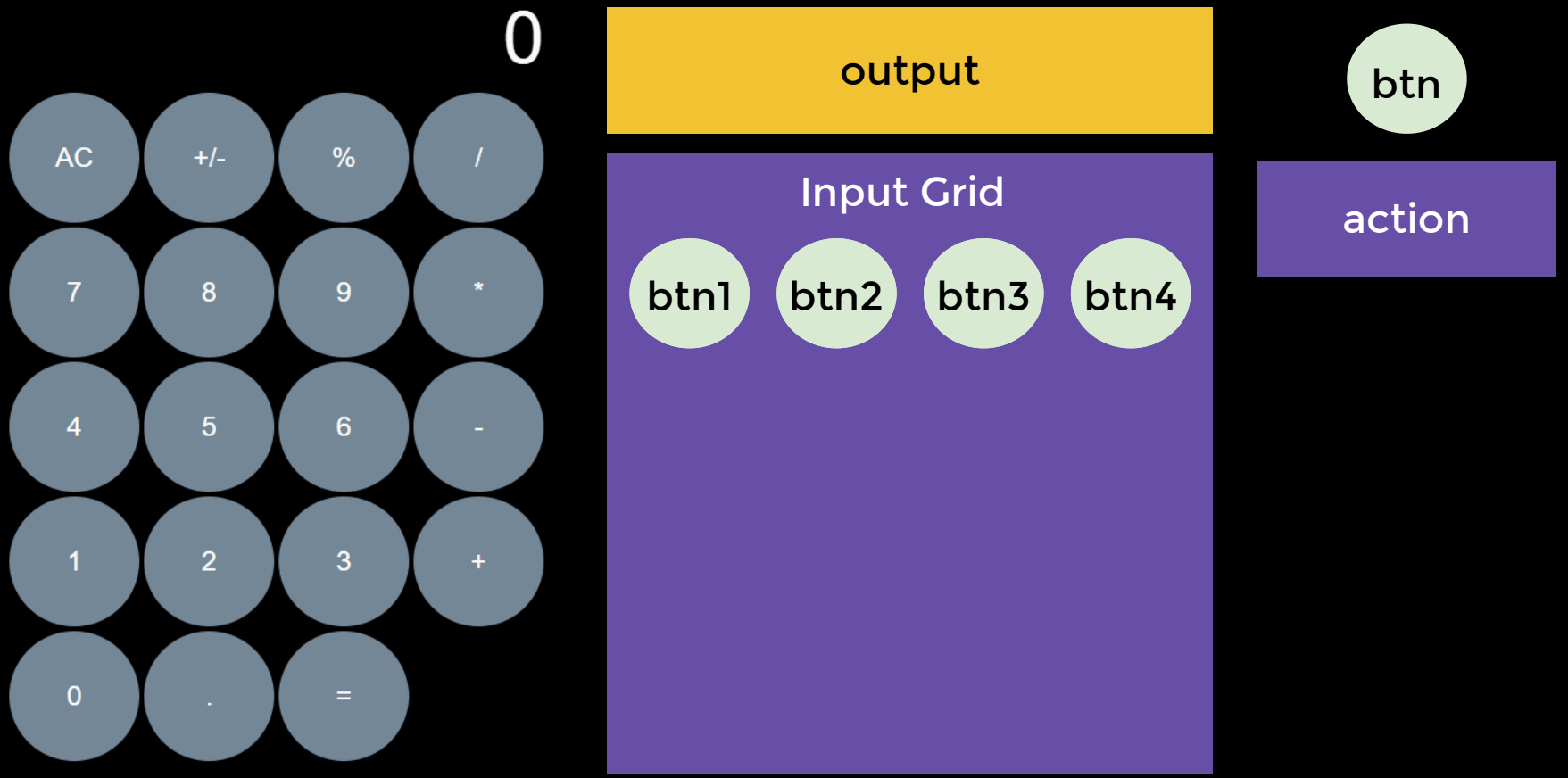
output

Input Grid

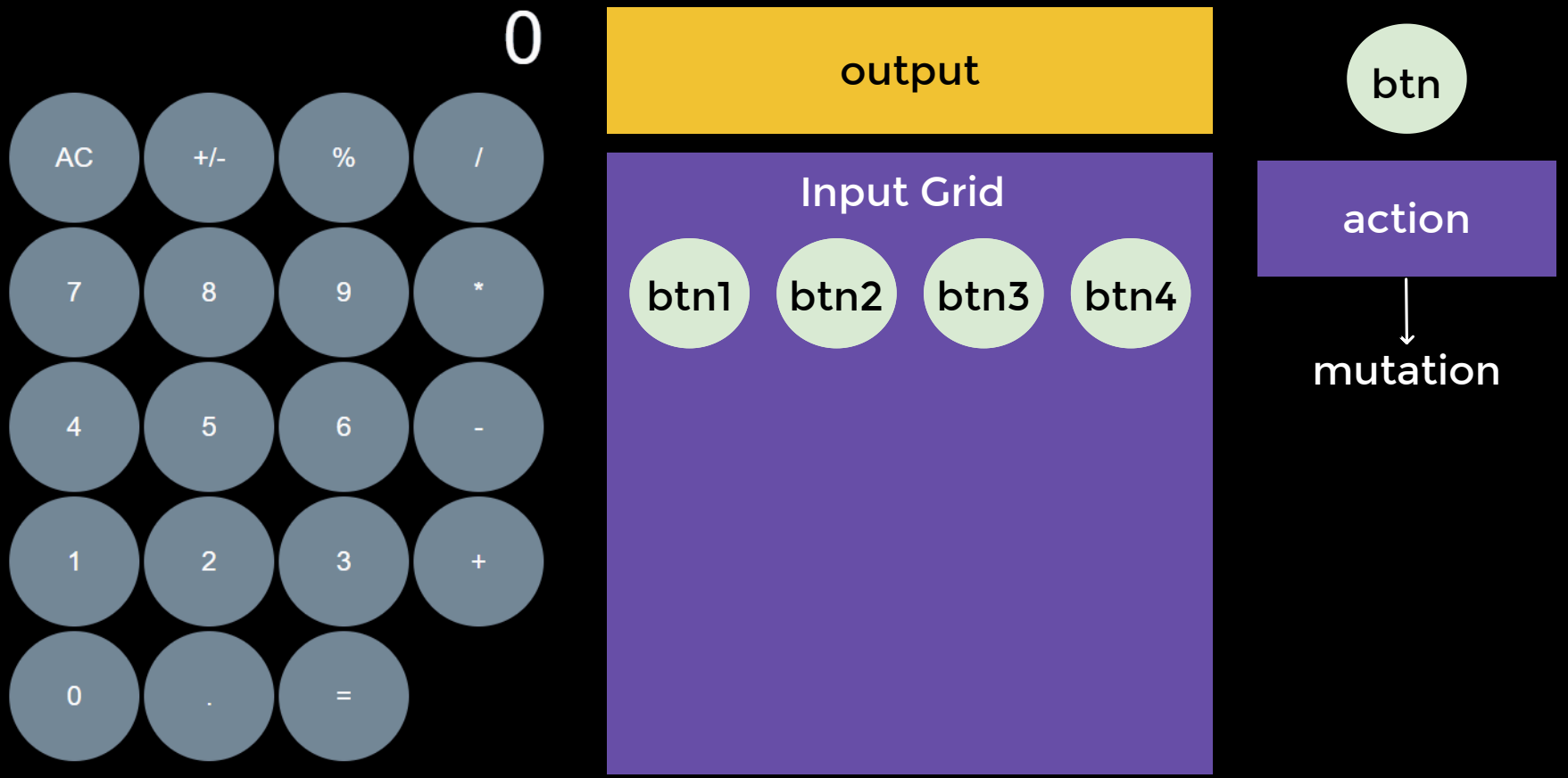
Problem 11: Solution Approach



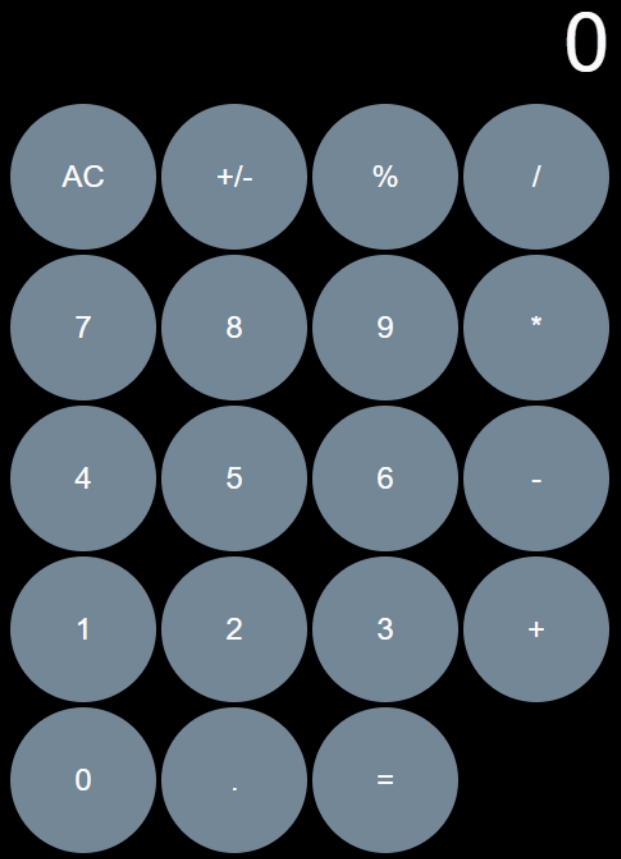
Problem 11: Solution Approach



Problem 11: Solution Approach



Problem 11: Solution Approach



output

Input Grid

btn1 btn2 btn3 btn4

btn

action

mutation

new state

Problem 11: Solution Approach

define actions

Problem 11: Solution Approach

define actions

component props

Problem 11: Solution Approach

define actions

component props

render UI

Problem 11: Solution Approach

define actions

component props

render UI

click handlers

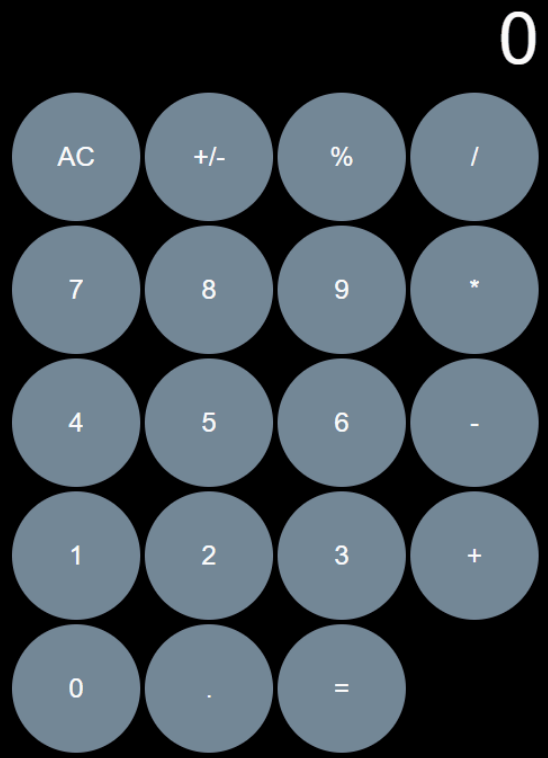
Calculator: applyNumber action

```
// Appends a digit to the expression string.
// If current state is '0', replaces it (avoids leading zeros like '05')
// '0' + '5' → '5',  '12' + '3' → '123'
const applyNumber: TButtonAction = (state, number) =>
  state === '0' ? number : state + number
```

0

Calculator: applyOperation action

```
// Appends an operator (+, -, *, /, %) to the expression.
// If the last character is already an operator, replaces it
// (prevents invalid expressions like '2+-')
// '5' + '+' → '5+',  '5+' + '*' → '5*'
const applyOperation: TButtonAction = (state, operator) =>
  OPERATIONS.has(state.at(-1) ?? '')
    ? state.slice(0, -1) + operator
    : state + operator
```

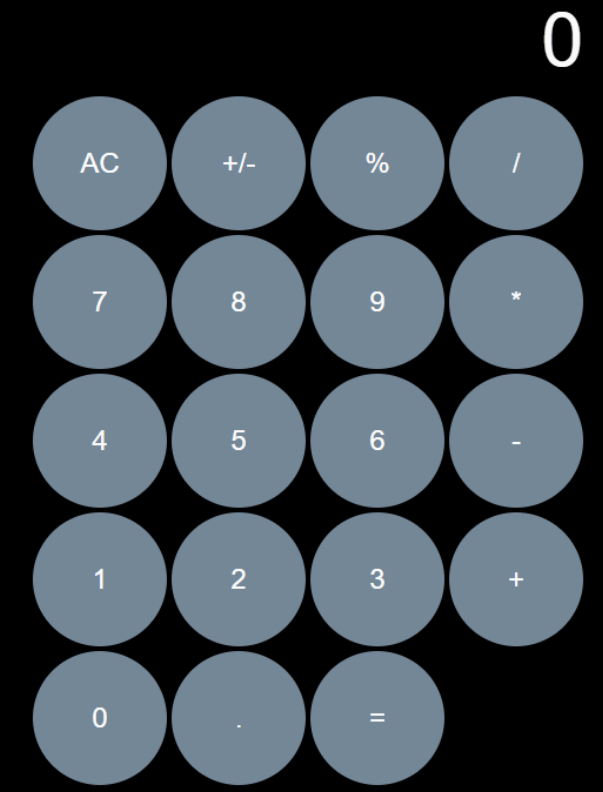


Calculator: clear action

```
const clear: TButtonAction = () => '0'
```

Calculator: negate action

```ts
// Wraps the expression in -(…) to negate, or unwraps if already negated
// '5+3' → '-(5+3)',  '-(5+3)' → '5+3'
const negate: TButtonAction = (state) =>
  state.match(/-\(.*\)/) ? state.slice(2, -1) : `-(${state})`
```



Calculator: calculate

```ts
/**
 * Formats a number with fixed precision, stripping trailing zeros.
 * @example toFixedWithoutZeros(3.10000, 5) // '3.1'
 * @example toFixedWithoutZeros(42, 5)      // '42'
 */
export const toFixedWithoutZeros = (num: number, precision: number) =>
    num.toFixed(precision).replace(/\.*0+$/, '')

const calculate: TButtonAction = (state) => {
  try {
    const result = Number(new Function('return ' + state)())
    return Number.isNaN(result) || !Number.isFinite(result)
      ? INVALID_VALUE : toFixedWithoutZeros(result, 5)
  } catch { return INVALID_VALUE }
}
```

Calculator: creating button map

```
/**
 * Complete map of calculator buttons.
 * Drives both rendering (iterate to create buttons) and logic (look up action by label on click).
 * Layout order matches a standard calculator: AC, +/-, %, ÷, then digits 7-9, ×, 4-6, -, 1-3, +, 0, ., =
 */
export const BUTTONS = new Map<string, TCalculatorButton>([
  [ 'AC', { label: 'AC', action: clear,} ],
  [ '+/-', { label: '+/-', action: negate }],
  ['%', { label: '%', action: applyOperation }],
  ['/', { label: '/', action: applyOperation }],
  ['7', { label: '7', action: applyNumber }],
  ['8', { label: '8', action: applyNumber }],
  ['9', { label: '9', action: applyNumber }],
  ['*', { label: '*', action: applyOperation }],
  ['4', { label: '4', action: applyNumber }],
  ['5', { label: '5', action: applyNumber }],
  ['6', { label: '6', action: applyNumber }],
  ['-', { label: '-', action: applyOperation }],
  ['1', { label: '1', action: applyNumber }],
  ['2', { label: '2', action: applyNumber }],
  ['3', { label: '3', action: applyNumber }],
  ['+', { label: '+', action: applyOperation }],
  ['0', { label: '0', action: applyNumber }],
  ['.', { label: '.', action: applyNumber }],
  ['=', { label: '=', action: calculate } ]
])
```

Calculator: creating button map

```
/**
 * Complete map of calculator buttons.
 * Drives both rendering (iterate to create buttons) and logic (look up action by label on click).
 * Layout order matches a standard calculator: AC, +/-, %, ÷, then digits 7-9, ×, 4-6, -, 1-3, +, 0, ., =
 */
export const BUTTONS = new Map<string, TCalculatorButton>([
  [ 'AC', { label: 'AC', action: clear,} ],
  [ '+/-', { label: '+/-', action: negate }],
  ['%', { label: '%', action: applyOperation }],
  ['/', { label: '/', action: applyOperation }],
  ['7', { label: '7', action: applyNumber }],
  ['8', { label: '8', action: applyNumber }],
  ['9', { label: '9', action: applyNumber }],
  ['*', { label: '*', action: applyOperation }],
  ['4', { label: '4', action: applyNumber }],
  ['5', { label: '5', action: applyNumber }],
  ['6', { label: '6', action: applyNumber }],
  ['-', { label: '-', action: applyOperation }],
  ['1', { label: '1', action: applyNumber }],
  ['2', { label: '2', action: applyNumber }],
  ['3', { label: '3', action: applyNumber }],
  ['+', { label: '+', action: applyOperation }],
  ['0', { label: '0', action: applyNumber }],
  ['.', { label: '.', action: applyNumber }],
  ['=', { label: '=', action: calculate } ]
])
```

Calculator: React component

```tsx
export const Calculator = () => {
  const [state, setState] = useState<string>('0')

  const handleButtonClick = ({ target }: React.MouseEvent<HTMLButtonElement>) => {
    if (target instanceof HTMLElement && target.dataset.label?.length) {
      const label = target.dataset.label
      const button = BUTTONS.get(label)
      if (button) {
        setState((state) => button.action(state, label))
      }
    }
  }

  return (
    <div className={cx(styles.flexColumnCenter, styles.bgBlack10, styles.w100, css.calculator)}>
      <output className={cx(css.output, styles.w100, styles.cWhite10)}>{state}</output>
      <section className={cx(css.keypad, styles.w100)} onClick={handleButtonClick}>
        {Array.from(BUTTONS.values()).map((button) => (
          <button
            disabled={state === INVALID_VALUE && button.label !== 'AC'}
            key={button.label}
            className={cx(css.button, styles.bNone, styles.cWhite10, styles.br128)}
            data-label={button.label}
          >
            {button.label}
          </button>
        ))}
      </section>
    </div>
  )
}
```

Problem 12: 8-puzzle game

Create a 3x3 grid game (also known as the 8-puzzle) containing tiles numbered 1 through 8 and one empty space. The objective is to rearrange the tiles from a random configuration into sequential order (1-8), leaving the last cell empty.

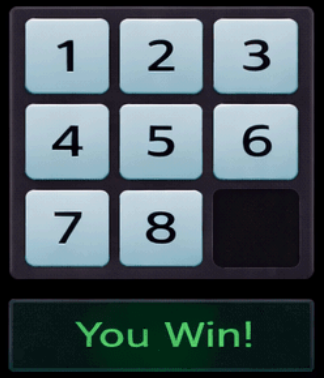
---

1 Rendering:

- Display a 3x3 grid.
- Render tiles numbered 1 to 8.
- Render one empty cell.
- Initial state should be randomized.

2 Interactivity:

- Users can click on a tile adjacent to the empty space (horizontally or vertically) to move it into the empty space.
- If a user clicks a tile not adjacent to the empty space, nothing should happen.



3 Win Condition:

- Detect when the tiles are ordered 1-8 in the first 8 positions, with the empty space at the end (index 8).

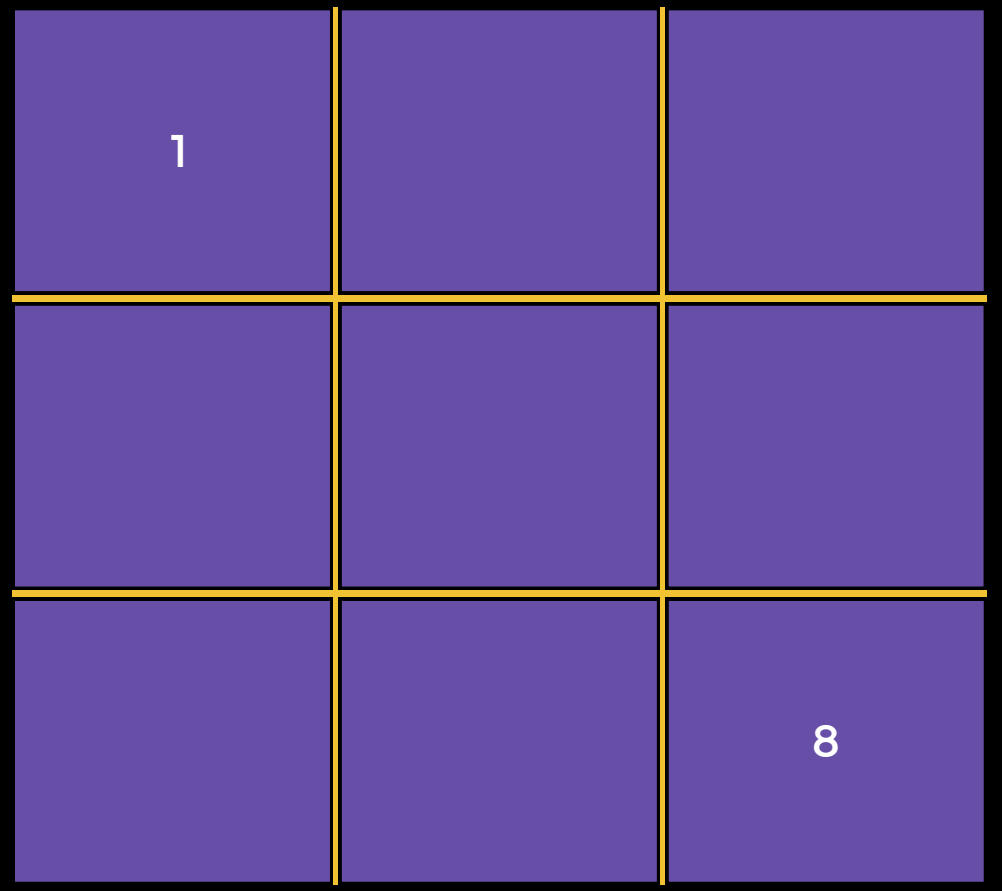
Level: Hard

8-puzzle game: Solution approach

Game field

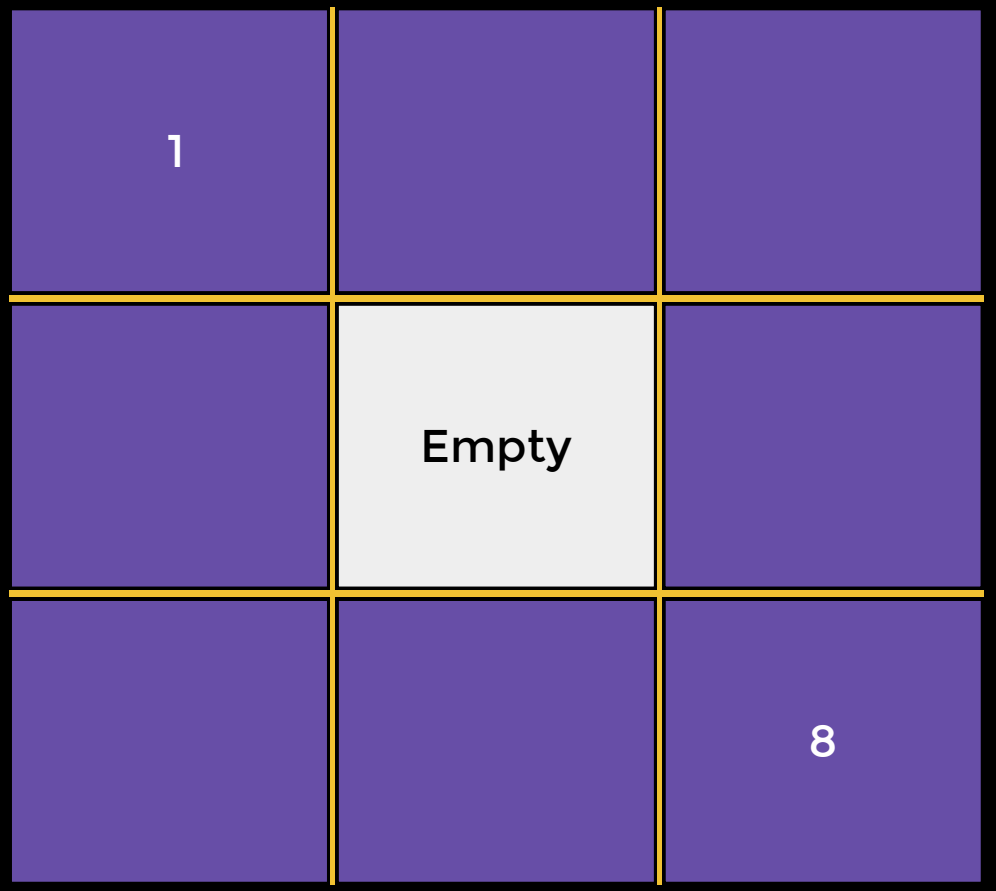
8-puzzle game: Solution approach

Game field



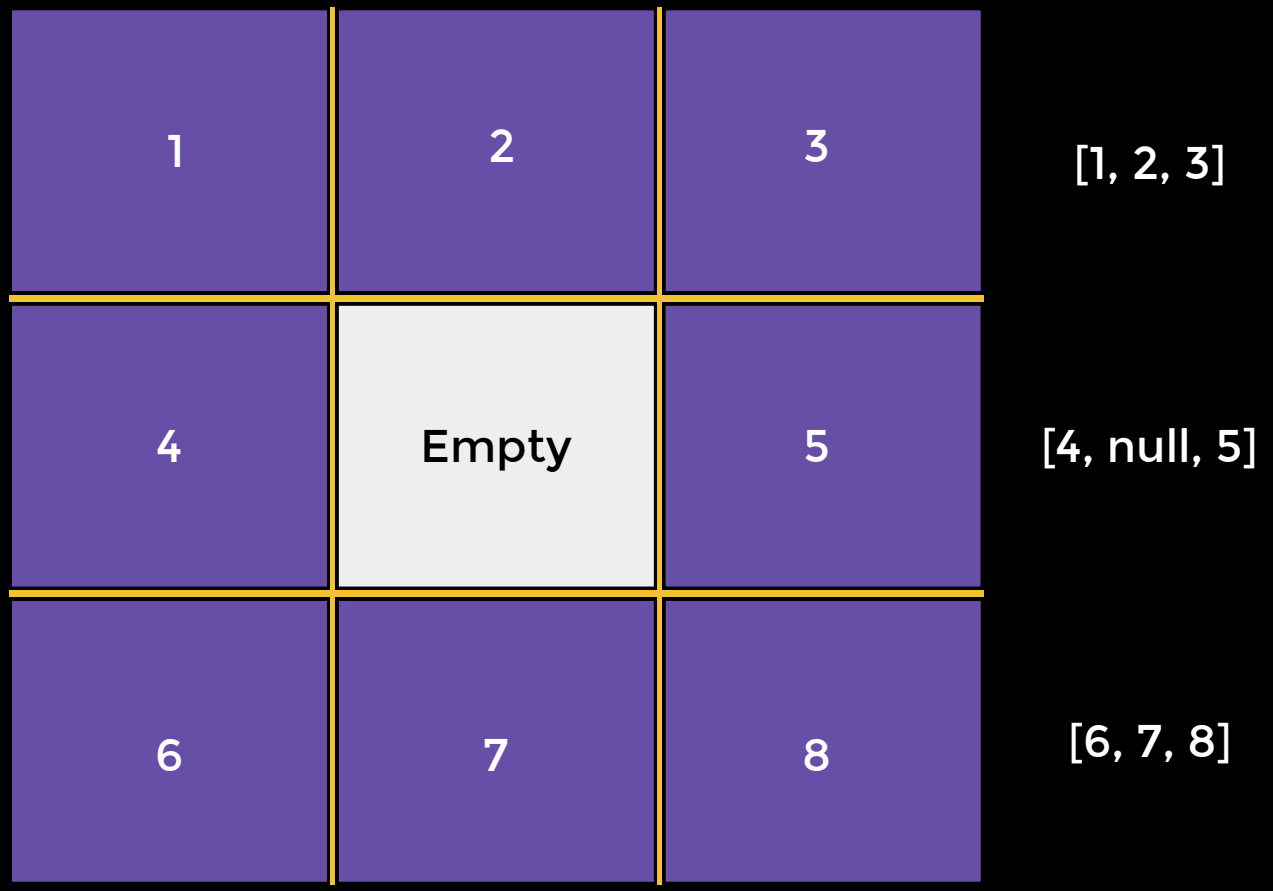
8-puzzle game: Solution approach

Game field



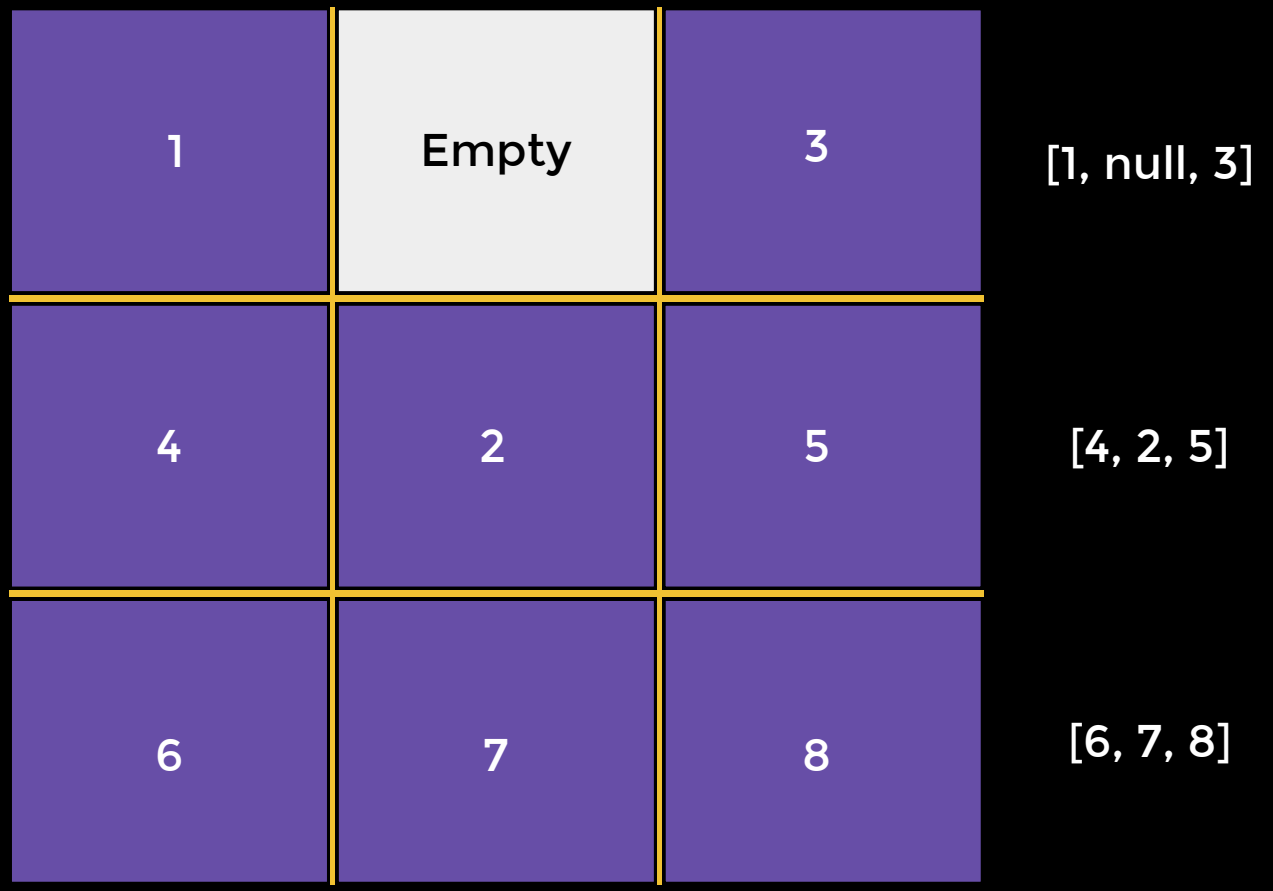
8-puzzle game: Solution approach

Game field



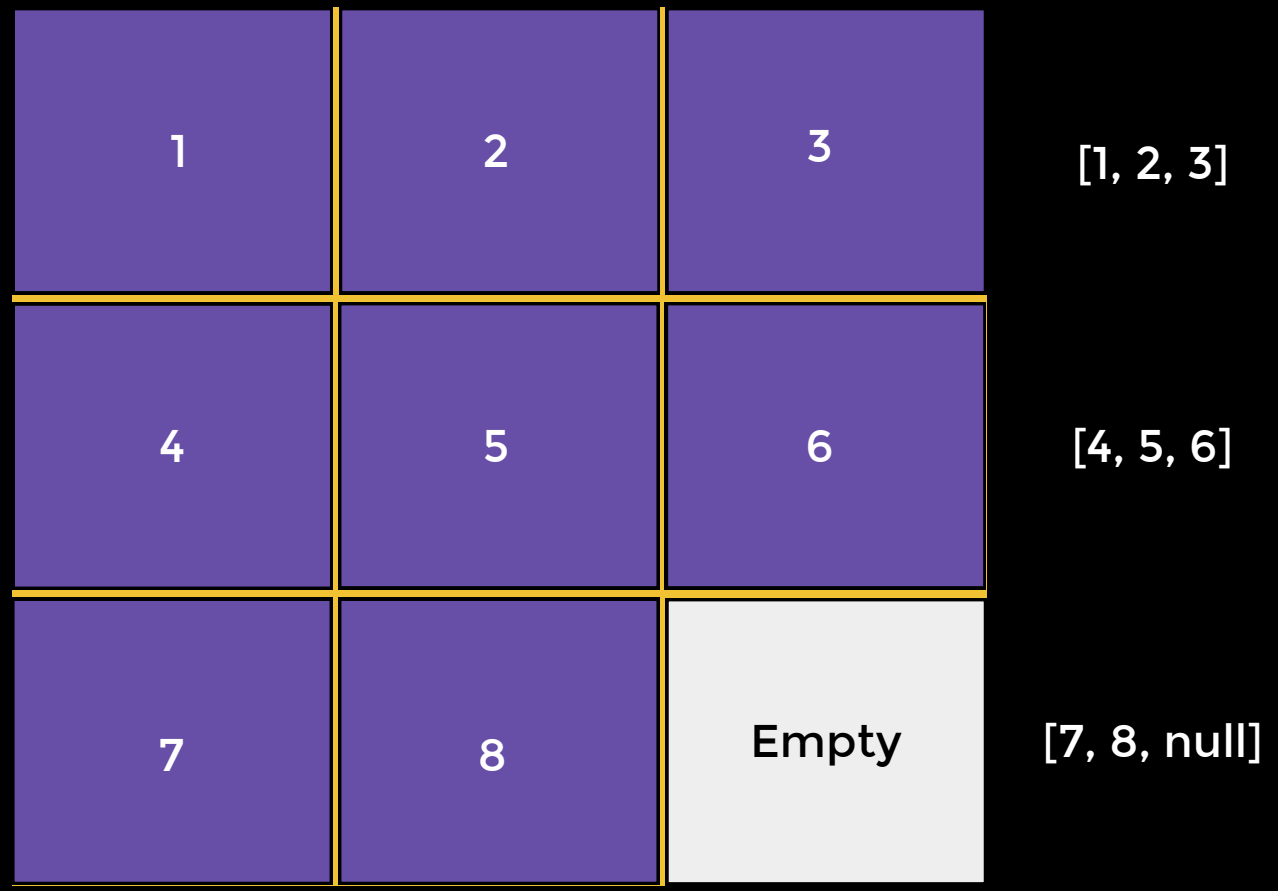
8-puzzle game: Solution approach

Game field



8-puzzle game: Solution approach

Game field



8-puzzle game: Solution approach

Generate Game State

Create array [1,2,3,4,5,6,7,8, null]

8-puzzle game: Solution approach

Generate Game State

Create array [1,2,3,4,5,6,7,8, null]

randomise [...]

8-puzzle game: Solution approach

Generate Game State

Create array	[1,2,3,4,5,6,7,8, null]
randomise	[...]
2Dify	[[1,2,3] ,[4,5,6] ,[7,8,null]]

8-puzzle game: Solution approach

Generate Game State

Create array — [1,2,3,4,5,6,7,8, null]

randomise — [...]

2Dify — [[1,2,3] ,[4,5,6] ,[7,8,null]]

State Validation

8-puzzle game: Solution approach

Generate Game State

Create array — [1,2,3,4,5,6,7,8, null]

randomise — [...]

2Dify — [[1,2,3] ,[4,5,6] ,[7,8,null]]

State Validation

is move valid

8-puzzle game: Solution approach

- **Generate Game State**
  - Create array — [1,2,3,4,5,6,7,8, null]
  - randomise — [...]
  - 2Dify — [[1,2,3] ,[4,5,6] ,[7,8,null]]
- **State Validation**
  - is move valid
  - is win condition

8-puzzle game: Solution approach

- **Generate Game State**
  - Create array — [1,2,3,4,5,6,7,8, null]
  - randomise — [...]
  - 2Dify — [[1,2,3] ,[4,5,6] ,[7,8,null]]
- **State Validation**
  - is move valid
  - is win condition
- **Rendering & Clicks Handling**

Solving 8-puzzle game: State creation

```
1 /**
2  * Shuffles an array using sort with a random comparator.
3  * @param arr The array to shuffle.
4  * @returns The shuffled array.
5  */
6 function randomizeArray(arr: Array<number | null>): Array<number | null> {
7   return arr.sort(() => Math.random() - 0.5)
8 }
```

Solving 8-puzzle game: State creation

```
1  /**
2   * Splits an array into chunks of a specified size (creating a 2D array).
3   * @param arr The 1D array to split.
4   * @param n The size of each chunk.
5   * @returns A 2D array.
6   */
7  function chunkify(arr: Array<number | null>, n: number): Array<Array<number | null>> {
8    return Array.from(
9        Array(n),
10       (_, i) => arr.slice(i * n, (i + 1) * n)
11   )
12 }
```

Solving 8-puzzle game: State creation

```ts
/**
 * Generates the initial game state for the square game.
 * Creates a sorted array, shuffles it, and converts it to a 2D grid.
 * @param size The size of the grid (e.g., 3 for a 3x3 grid).
 * @returns A 2D array representing the game board.
 */
export function getGameState(size: number): Array<Array<number | null>> {
  const arr = randomizeArray(
    Array(size * size)
      .fill(null)
      .map((_, i) => (i === size * size - 1 ? null : i + 1))
  )
  return chunkify(arr, size)
}
```

Solving 8-puzzle game: State validation

```ts
type TPosition = [row: number, col: number]

/**
 * Validates if a move is legal.
 * A move is legal if the target cell is adjacent (horizontally or vertically) to the empty cell.
 * @param param0 [row, col] coordinates of the clicked cell.
 * @param param1 [emptyRow, emptyCol] coordinates of the empty cell.
 * @returns True if the move is valid, false otherwise.
 */
export function validate([row, col]: TPosition, [emptyRow, emptyCol]: TPosition) {
  const validHorizontally = row === emptyRow && (col === emptyCol + 1 || col === emptyCol - 1)
  const validVertically = col === emptyCol && (row === emptyRow + 1 || row === emptyRow - 1)
  return validHorizontally || validVertically
}
```

Solving 8-puzzle game: Find empty cell

```ts
/**
 * Finds the coordinates of the empty cell (null value) in the grid.
 * @param arr The 2D game grid.
 * @returns [row, col] of the empty cell.
 */
export function getEmptyPosition(arr: Array<Array<number | null>>): TPosition {
  for (let row = 0; row < arr.length; row++) {
    for (let col = 0; col < arr[0].length; col++) {
      if (arr[row][col] === null) {
        return [row, col]
      }
    }
  }
  throw Error('Invalid array')
}
```

Solving 8-puzzle game: Win condition

```ts
/**
 * Checks if the game is won.
 * The game is won if tiles are in order 1..n with null at the end.
 * @param arr The 2D game grid.
 * @returns True if the game is won.
 */
export function isWin(arr: Array<Array<number | null>>): boolean {
  const flat = arr.flat()
  return flat.every((val, i) => i === flat.length - 1 ? val === null : val === i + 1)
}
```

Solving 8-puzzle game: Rendering

```
toHTML() {
  const cells = this.state
    .map((row, rowIndex) => {
      return row
        .map((col, colIndex) => {
          const cellClass = col === null ? css.cell__empty : css.cell__filled
          return `
            <div
              class="${css.cell} ${cellClass}"
              data-row="${rowIndex}"
              data-col="${colIndex}"
            >
              ${col == null ? '' : col}
            </div>`
        })
        .join('')
    })
    .join('')

  return `
    <section class="${css.container}">
      <div>Game status: ${isWin(this.state) ? 'win' : 'not yet'}</div>
      <div class="${css.board}">
        ${cells}
      </div>
    </section>
  `
}
```

Solving 8-puzzle game: Handle Click

```
onClick(e: Event) {
  const target = e.target as HTMLElement
  const row = Number(target.dataset.row)
  const col = Number(target.dataset.col)

  if (isNaN(row) || isNaN(col)) return

  const [emptyRow, emptyCol] = getEmptyPosition(this.state)
  if (validate([row, col], [emptyRow, emptyCol])) {
    const newState = structuredClone(this.state);
    [newState[row][col], newState[emptyRow][emptyCol]] = [
      newState[emptyRow][emptyCol],
      newState[row][col],
    ]
    this.state = newState
    this.render()
  }
}
```

Solving 8-puzzle game: Handle Click

```
1   onClick(e: Event) {
2     const target = e.target as HTMLElement
3     const row = Number(target.dataset.row)
4     const col = Number(target.dataset.col)
5
6     if (isNaN(row) || isNaN(col)) return
7
8     const [emptyRow, emptyCol] = getEmptyPosition(this.state)
9     if (validate([row, col], [emptyRow, emptyCol])) {
10      const newState = structuredClone(this.state);
11      [newState[row][col], newState[emptyRow][emptyCol]] = [
12        newState[emptyRow][emptyCol],
13        newState[row][col],
14      ]
15      this.state = newState
16      this.render()
17    }
18  }
```

Challenge 20: Replace

Implement Replace<S, From, To> which replaces the first occurrence of the string From with string To in the given string S

```
type replaced = Replace<'types are fun!', 'fun', 'awesome'> // 'types are awesome!'
```

Challenge 21: Replace | Solution

Implement Replace<S, From, To> which replaces the first occurrence of the string From with string To in the given string S

```
type Replace<S extends string, From extends string, To extends string> = From extends ''
  ? S
  : S extends `${infer P1}${From}${infer P2}`
    ? `${P1}${To}${P2}`
    : S
```

Challenge 22: Deep Readonly

Implement a generic DeepReadonly<T> which makes every parameter of an object – and its sub-objects recursively – readonly.

```
type X = {
  x: {
    a: 1
    b: 'hi'
  }
  y: 'hey'
}

type Expected = {
  readonly x: {
    readonly a: 1
    readonly b: 'hi'
  }
  readonly y: 'hey'
}

type Result = DeepReadonly<X> // should be same as Expected
```

Challenge 22: Deep Readonly | Solution

Implement a generic DeepReadonly<T> which makes every parameter of an object – and its sub-objects recursively – readonly.

```
1 type DeepReadonly<T> = {
2   readonly [K in keyof T]: keyof T[K] extends never ? T[K] : DeepReadonly<T[K]>
3 }
```

Problem 13: Typeahead

1 Rendering & Data Fetching

- Display **input field**.
- Fetch suggestions from **/api/typeahead**.
- Implement **debouncing** or **useDeferredValue**.
- Render dropdown list with results,
- Show **loading indicator** while fetching.
- Display a **loading** indicator while fetching.

2 Interactivity & UX

- Allow selection via **mouse click**.
- Support **keyboard navigation**:
  - Arrow Down → Move highlight down
  - Arrow Up → Move highlight up
  - **Enter** / **Space** → Select highlighted item
  - **Escape** → Close suggestions
- Close dropdown on **outside click**.

3 Network Robustness

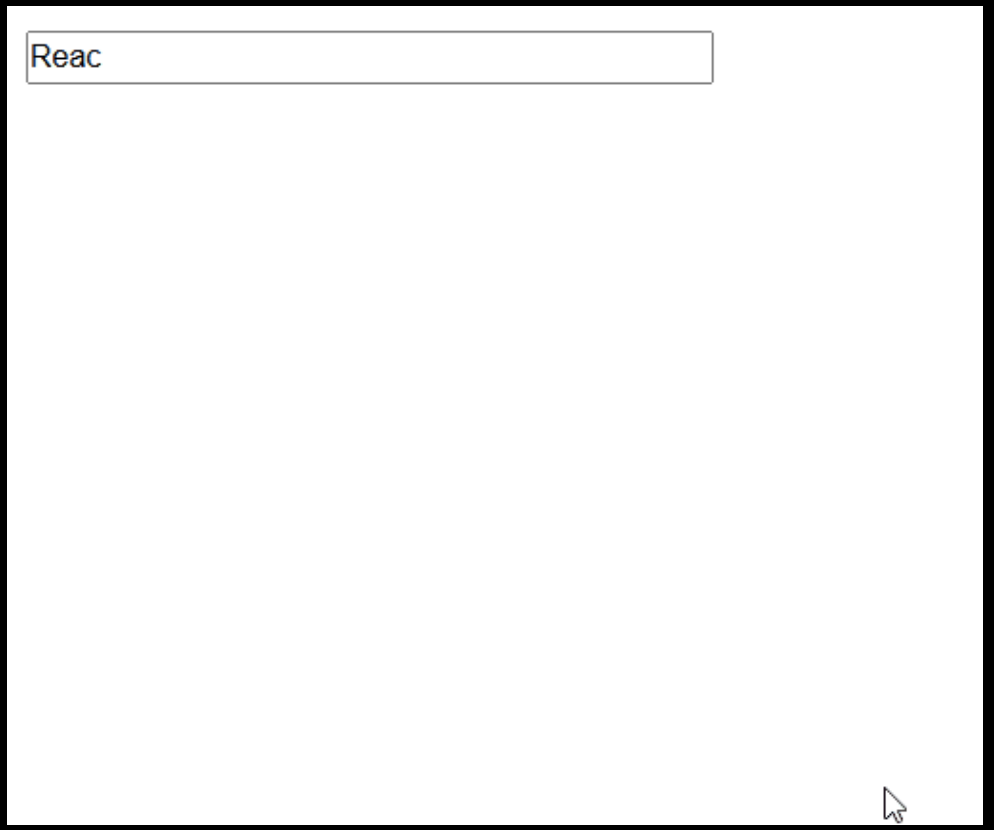
- Handle **race conditions**, prevent overwrites.
- Keep UI **responsive** during fast typing.
- Gracefully handle network errors.

4 API Integration

- GET **/api/typeahead?query=**{string}
- Returns JSON array of matching entries.
- Simulate network delay in **response**.

Level: Hard

Typeahead: Preview



Typeahead: API

The Typeahead component fetches suggestions from the backend API `/api/typeahead`. The server returns a JSON array of matched items.

```ts
const handleQuery = async (query: string, signal?: AbortSignal) => {
  if (!query) return []
  try {
    const response = await fetch(`/api/typeahead?query=${encodeURIComponent(query)}`, { signal })
    if (!response.ok) throw new Error('Network response was not ok')
    return await response.json()
  } catch (error: any) {
    if (error.name === 'AbortError') return []
    console.error('Fetch error:', error)
    return []
  }
}
```

```json
[
  {
    "id": "1",
    "query": "React",
    "value": "React — Item ID 1"
  },
  {
    "id": "31",
    "query": "TypeScript",
    "value": "TypeScript — Item ID 31"
  }
]
```

Solving Typeahead: Property Model

```
export type TTypeaheadEntry<T> = {
  query: string
  id: string
  value: T
}

type TTypeaheadProps<T> = {
  id?: string
  entries?: TTypeaheadEntry<T>[]
  onQuery: (query: string) => Promise<TTypeaheadEntry<T>[]>
  itemRender: (item: TTypeaheadEntry<T>) => React.ReactNode
}
```

Solving Typeahead: Basic State

```tsx
export function Typeahead<T>({ id = 'typeahead', entries = [], onQuery, itemRender }: TTypeaheadProps<T>) {
  const trieRef = useRef<Trie<TTypeaheadEntry<T>>>(new Trie())

  const [items, setItems] = useState<TTypeaheadEntry<T>[]>([])
  const [query, setQuery] = useState('')
  const [isLoading, setIsLoading] = useState(false)

  const deferredQuery = useDeferredValue(query)

  // Insert entries into the Trie cache
  const updateTrie = useCallback(
    (entries: TTypeaheadEntry<T>[]) =>
      entries.forEach((entry) => trieRef.current.insert(entry.query, entry)),
    [],
  )

  // Read cached results from the Trie
  const updateVisibleItems = useCallback(() => {
    setItems(trieRef.current.getWithPrefix(deferredQuery))
  }, [deferredQuery])
```

Solving Typeahead: Async Fetching & Race Conditions

```
// Seed trie with initial entries
 useEffect(() => {
   trieRef.current = new Trie()
   updateTrie(entries)
 }, [entries, updateTrie])

 // Fetch on query change with race condition handling
 useEffect(() => {
   updateVisibleItems()   // Show cached results immediately
   setIsLoading(true)

   const controller = new AbortController()

   onQuery(deferredQuery, controller.signal)
     .then((entries) => {
       updateTrie(entries)    // Cache new results in Trie
       updateVisibleItems()   // Update UI with merged results
     })
     .catch((error) => {
       if (error.name === 'AbortError') return // ← stale request aborted
       console.error(error)
     })
     .finally(() => {
       if (controller.signal.aborted) return
       setIsLoading(false)
     })

   return () => controller.abort() // Abort previous fetch on cleanup
 }, [deferredQuery, onQuery, updateTrie, updateVisibleItems])
```

Solving Typeahead: Cache Invalidation

```
useEffect(() => {
  const id = setInterval(() => {
    trieRef.current = new Trie()
    updateTrie(entries)
  }, 60 * 1000)
  return () => clearInterval(id)
}, [entries, updateTrie])
```

Solving Typeahead: Rendering

```
return (
  <section className={styles.container}>
    <input
      type="text"
      value={query}
      onChange={(e) => setQuery(e.target.value)}
    />
    {(items.length > 0 || isLoading) && (
      <ul>
        {items.map((item) => (
          <li key={item.id}>
            {itemRender(item)}
          </li>
        ))}
        {isLoading && <li>Loading...</li>}
      </ul>
    )}
  </section>
)
```

Solving Typeahead: a11y

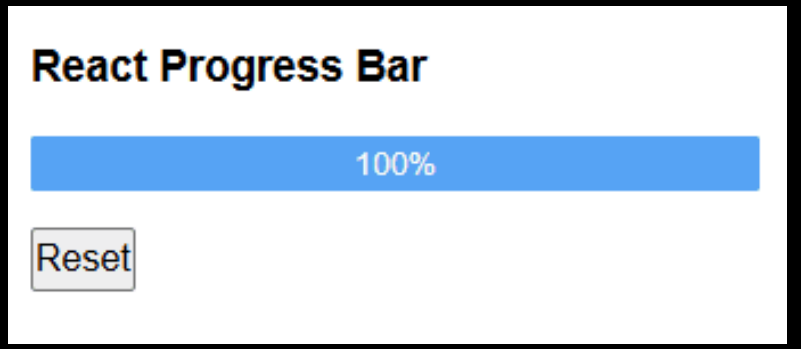
```
  return (
    <section className={styles.container}>
      {/* Live Region for Screen Readers */}
      <div role="status" aria-live="polite" className={styles.visuallyHidden}>
        {items.length} results available.
      </div>
      <input
        id={id}
        role="combobox"
        aria-autocomplete="list"
        aria-expanded={items.length > 0}
        aria-controls={`${id}-listbox`}
        type="text" // ...
      />
      {/* ... */}
        <ul id={`${id}-listbox`} role="listbox">
          {items.map((item) => (
            <li tabIndex={0} role="option" aria-selected={false} key={item.id}>
              {/* ... */}
            </li>
          ))}
          {isLoading && <li role="status">Loading...</li>}
        </ul>
    </section>
  )
}
```

Problem 14: Progress bar

Build a **progress bar** component that visually indicates completion **percentage** with **smooth animations** and an **optional text** label that changes color as the bar fills.

Requirements:

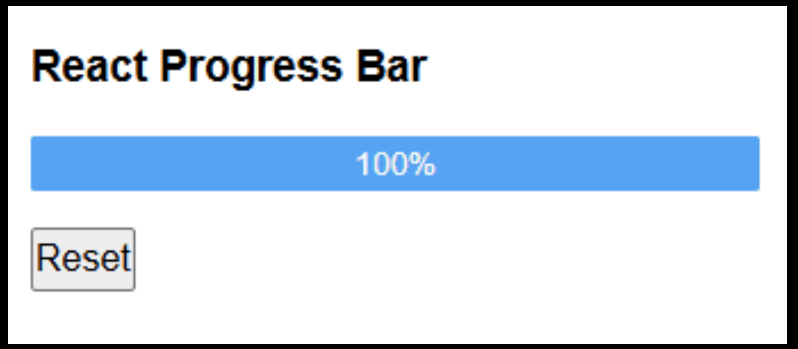
1. **Progress Visualization:** Display a filled bar representing the current value as a percentage of the max.
2. **Smooth Animation:** Animate the progress bar width using CSS **transform: translateX()** for GPU-accelerated rendering.



Level: Easy

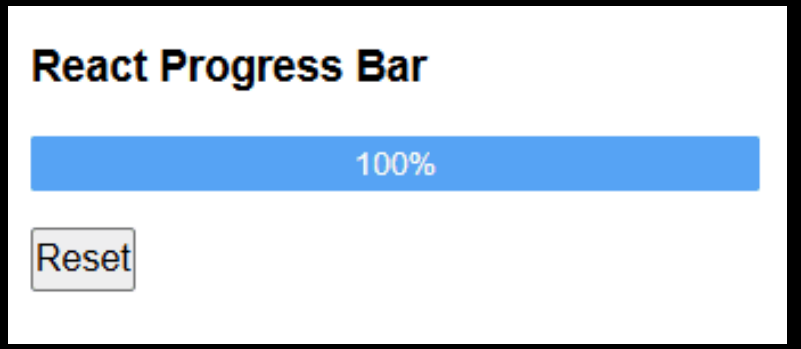
Progress bar: API Design

```tsx
/**
 * Props for the ProgressBar component.
 */
interface ProgressBarProps {
  /** Current progress value (0-100) */
  value: number
  /** Maximum value (default: 100) */
  max?: number
  /** Optional label to display */
  label?: string
}
```



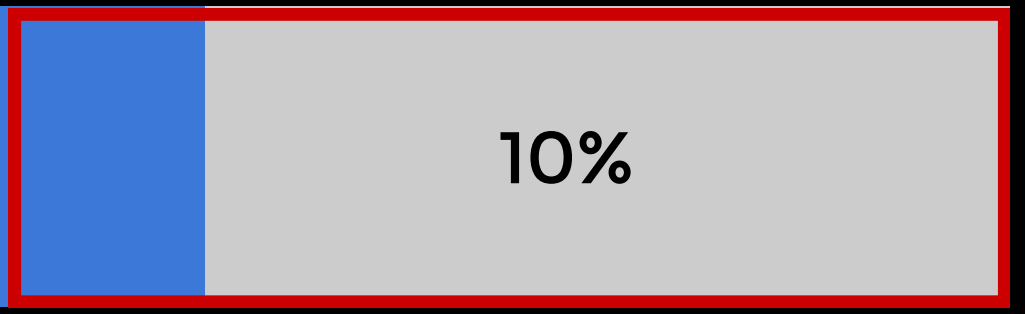
Progress bar: Fill logic

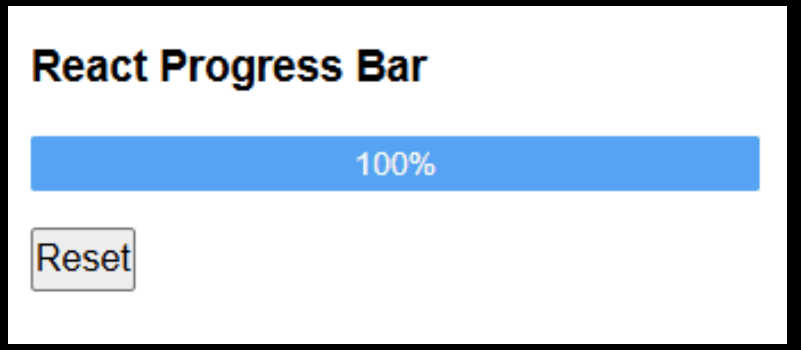
translateX(-100%)	0%



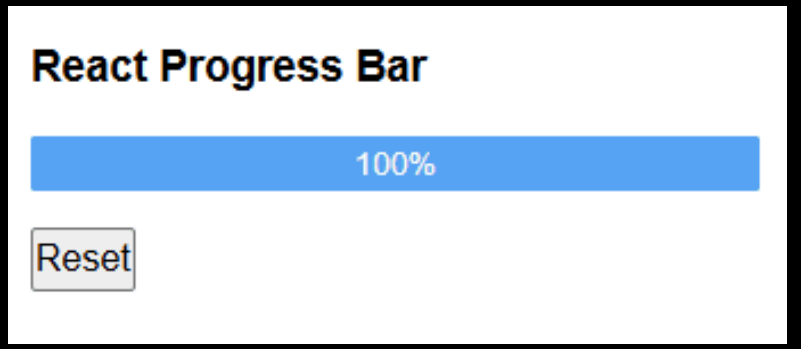
Progress bar: Fill logic

translateX(-90%)



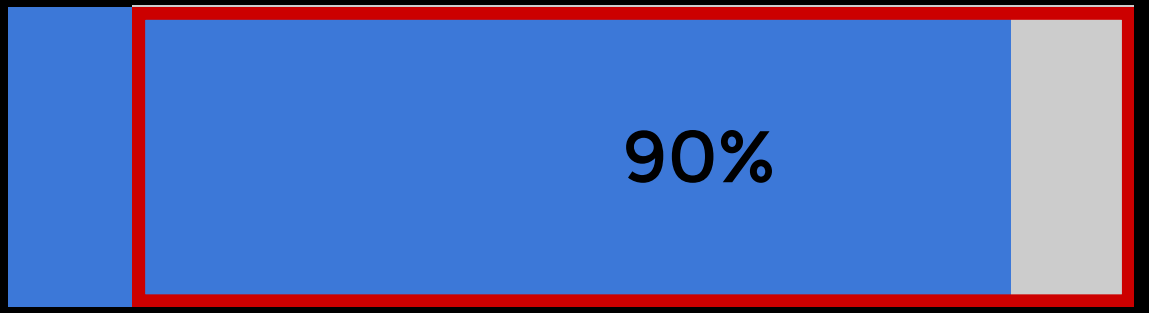


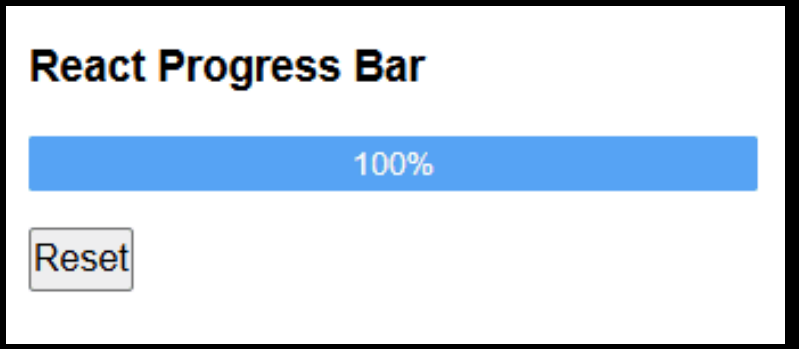
Progress bar: Fill logic



Progress bar: Fill logic

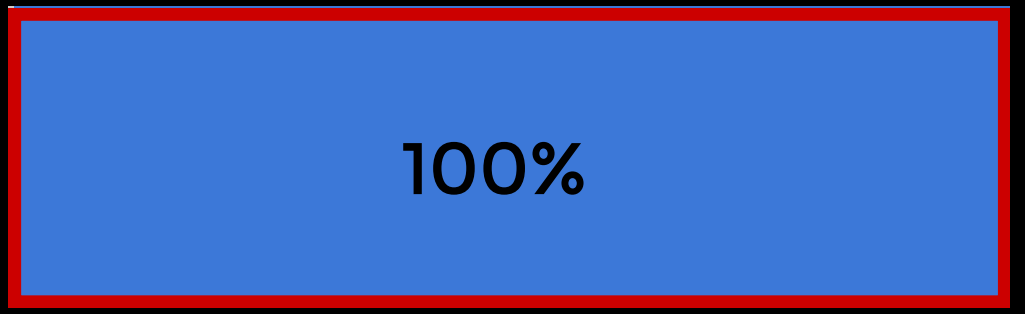
translateX(-90%)

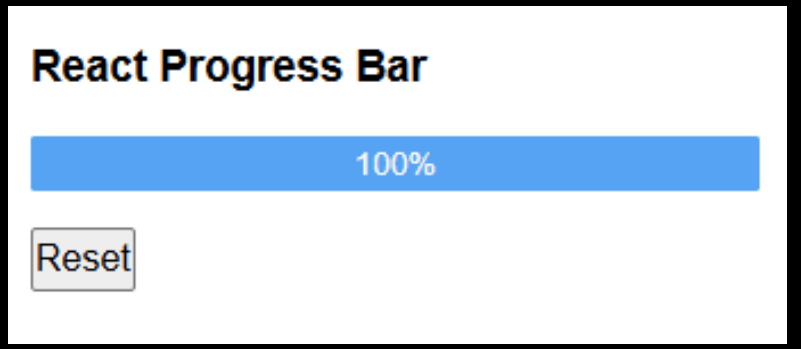




Progress bar: Fill logic

translateX(0%)





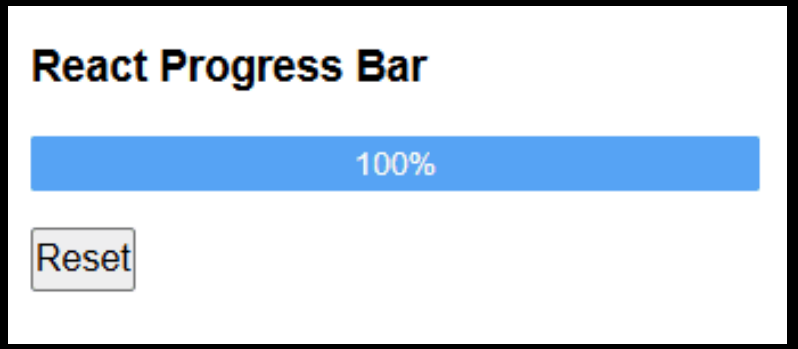
Progress bar: Color Reverse trick

0%

0%

clipPath: `inset(0 ${100% 0 0)`

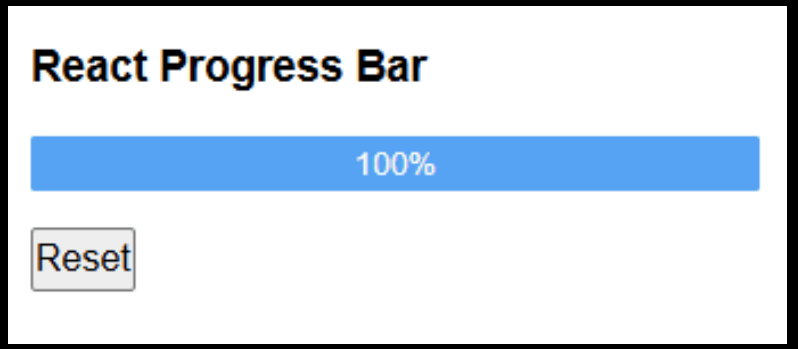
Clipped Area



Progress bar: Color Reverse trick

clipPath: `inset(0 ${50% 0 0)`

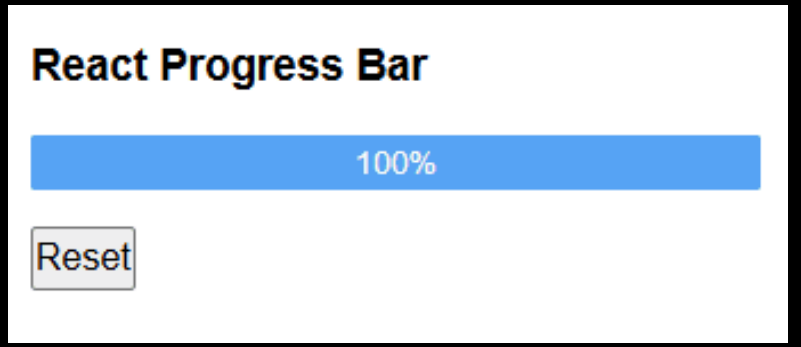
Clipped Area



Progress bar: Color Reverse trick

100%

clipPath: `inset(0 ${100% 0 0)`



Progress bar: Solution

```tsx
export const ProgressBar = ({ value, max = 100, label }: ProgressBarProps) => {
  const percentage = Math.min(100, Math.max(0, (value / max) * 100))

  return (
    <div
      role="progressbar"
      aria-valuenow={value}
      aria-valuemin={0}
      aria-valuemax={max}
      aria-label={label ?? `Progress: ${percentage}%`}
      className={cx(css['progress-bar'], flex.pRel, flex.w100)}
    >
      <div className={cx(css.progress, flex.pRel, flex.wh100)} style={{ transform: `translateX(-${100 - percentage}%)` }} />
      {label && (
        <div className={cx(css.label, flex.flexRowCenter, flex.itemsCenter, flex.justifyCenter, flex.pAbs, flex.inset0, flex.z1)}>
          {label}
        </div>
      )}
      {label && (
        <div
          className={cx(css.labelOverlay, flex.flexRowCenter, flex.itemsCenter, flex.justifyCenter, flex.pAbs, flex.inset0)}
          style={{ clipPath: `inset(0 ${100 - percentage}% 0 0)` }}
          aria-hidden="true"
        >
          {label}
        </div>
      )}
    </div>
  )
}
```

Challenge 23: Reverse | Solution

Implement the type version of `Array.reverse`.

```ts
type Reverse<T extends any[]> = T extends [infer First, ...infer Rest]
  ? [...Reverse<Rest>, First]
  : []
```

Challenge 23: Reverse

Implement the type version of `Array.reverse`.

```ts
type a = Reverse<['a', 'b']>      // ['b', 'a']
type b = Reverse<['a', 'b', 'c']> // ['c', 'b', 'a']
```

Challenge 24: isUnion

Implement a type IsUnion that takes an input type T and returns whether T resolves to a union type.

```
type A = IsUnion<'a' | 'b'> // true
type B = IsUnion<'a'>       // false
```

Challenge 24: isUnion | Solution

Implement a type IsUnion that takes an input type T and returns whether T resolves to a union type.

```
type IsUnion<T, Copy = T> =

(T extends Copy  ? (Copy extends T ? true : fals
                 : false) extends true
                                        ? false
                                        : true
```

Problem 15.1 - Upload Hook

Goal: Build a custom React hook **useFileUpload** to manage **chunked file uploads** with XHR logic, **pause/resume**, **upload speed & estimated time**.

Requirements:

1. **State Management:** Track upload status ('idle' | 'uploading' | 'paused' | 'completed' | 'cancelled' | 'error'), progress (0–100), speed (KB/s), uploadedBytes, remainingTimeMs, error string.

2. **Control Functions:** Provide start(file), pause(), resume(file), cancel() to imperatively control the upload process.

3. **Resumable Uploads:** Resume an upload from where it left off.

Level: Hard

Problem 15.1 - API Requirement

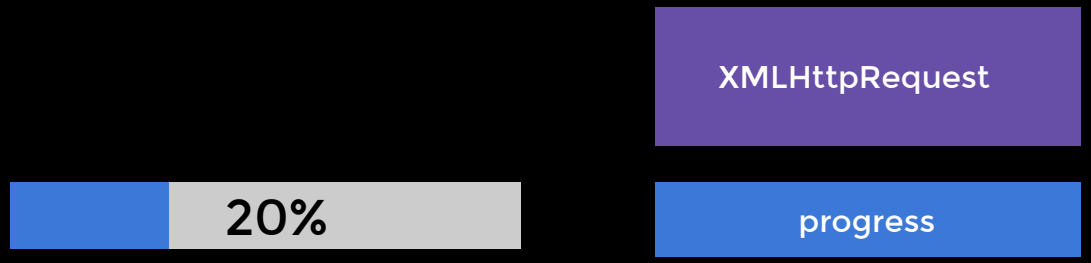
```ts
export type TUploadStatus =
  'idle'
  | 'uploading'
  | 'paused'
  | 'completed'
  | 'cancelled'
  | 'error'

/** The state of the file upload exposed to the consumer */
export type TUploadState = {
  /** The current status. 'idle' means no upload has started. */
  status: TUploadStatus
  /** The percentage of completion, ranging from 0 to 100 */
  progress: number
  /** The current upload speed in KB/s */
  speed: number // KB/s
  /** The absolute number of bytes successfully
   *  uploaded so far */
  bytes: number
  /** The estimated time remaining in milliseconds.
   * Null if unknown or not uploading. */
  remainingTimeMs: number | null
  /** If the status is 'error', contains
   *  the error message */
  error: string | null
}
```

```ts
export type TUploadControls = {
  /**
   * Starts an upload for a given File,
   * optionally starting from a specific byte offset.
   * @param file The file object from the browser to upload
   * @param from The byte offset to start slice from (default 0)
   */
  start: (file: File, from?: number) => void
  /**
   * Pauses the active upload, severing
   * the network connection but maintaining state.
   */
  pause: () => void
  /**
   * Resumes a paused upload by finding the last
   * known byte offset and starting again.
   * @param file The exact same file object previously paused.
   */
  resume: (file: File) => void
  /**
   * Hard aborts the upload and completely
   * resets the state back to 'idle'.
   */
  cancel: () => void
}
```

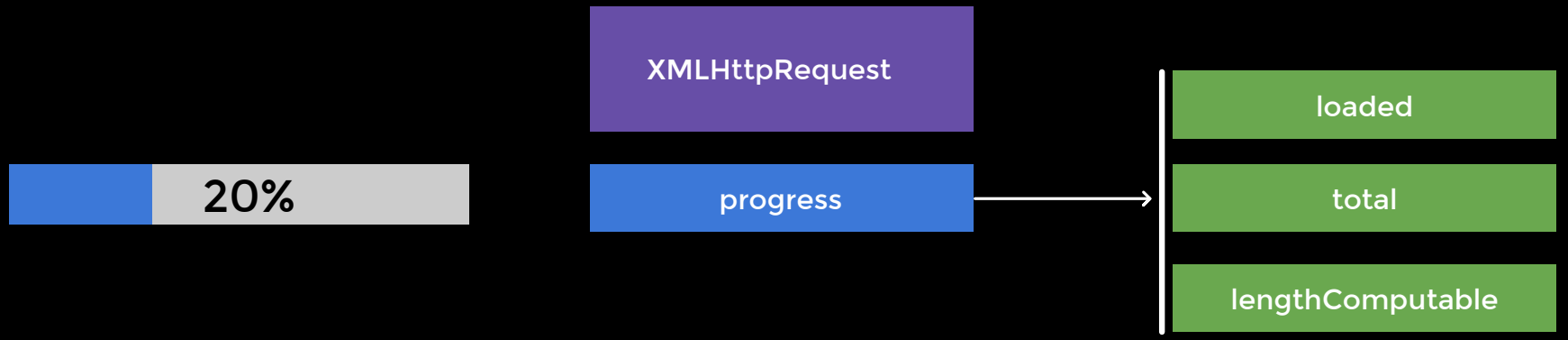
File Upload: Solution approach

XMLHttpRequest

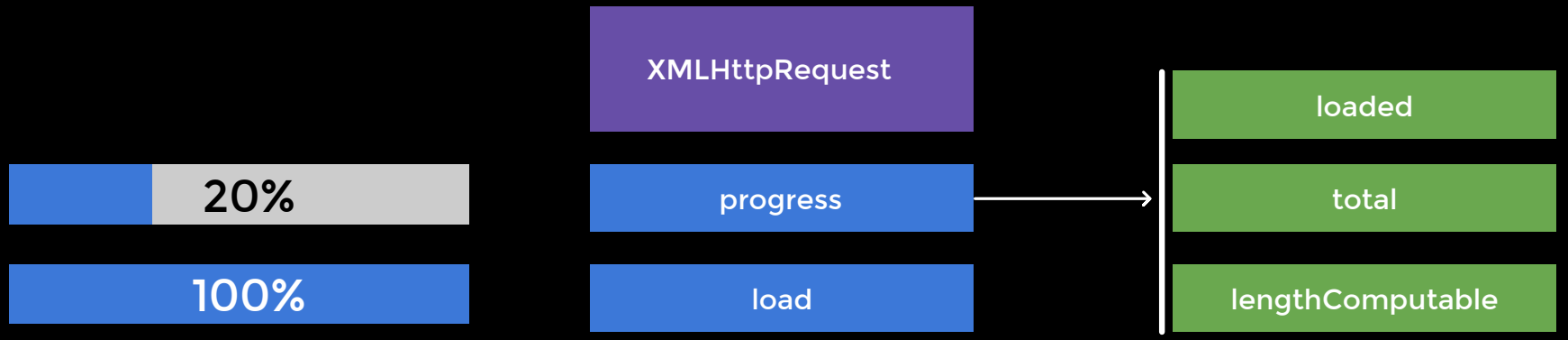
File Upload: Solution approach



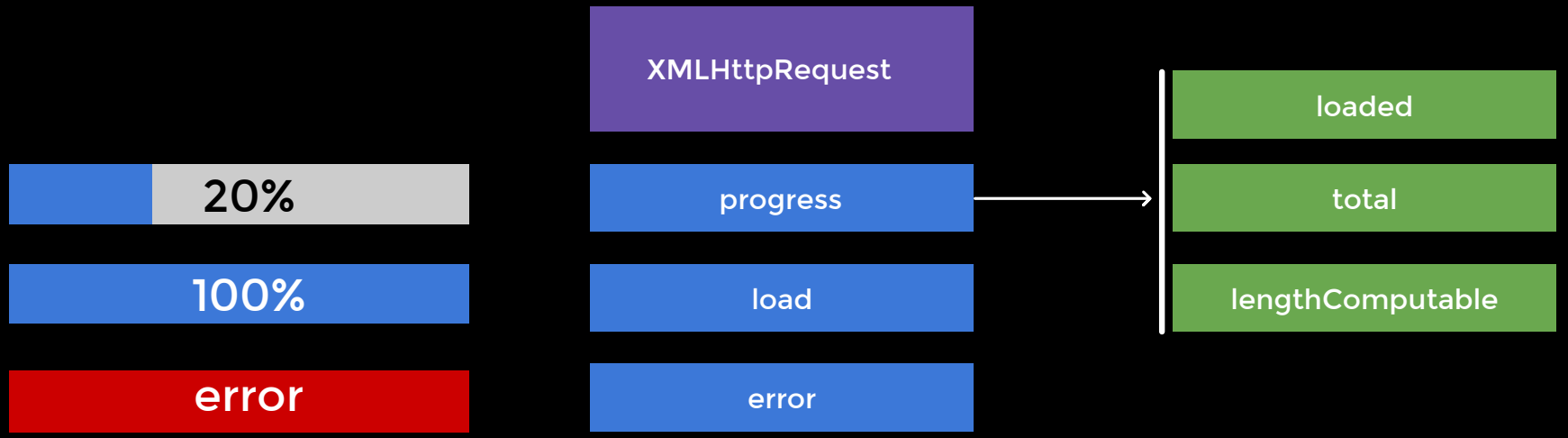
File Upload: Solution approach



File Upload: Solution approach



File Upload: Solution approach



Solving File Upload: minimal setup

```ts
export function useFileUpload(): [TUploadState, TUploadControls] {
  const [state, setState] = useState<TUploadState>(DEFAULT_STATE)

  // Track the active request to abort it when paused/cancelled
  const xhrRef = useRef<XMLHttpRequest | null>(null)

  // Track metrics for speed calculation (last bytes loaded, last timestamp)
  const metricsRef = useRef({ lastLoaded: 0, lastTime: 0 })

  // Track current successfully uploaded offset to use with slicing
  const offsetRef = useRef(0)

  // Helper to kill active requests silently
  const cleanup = useCallback(() => {
    if (xhrRef.current) {
      xhrRef.current.abort()
      xhrRef.current = null
    }
  }, [])
  // ...
```

Solving File Upload: minimal setup

```
export function useFileUpload(): [TUploadState, TUploadControls] {
  const [state, setState] = useState<TUploadState>(DEFAULT_STATE)

  // Track the active request to abort it when paused/cancelled
  const xhrRef = useRef<XMLHttpRequest | null>(null)

  // Track metrics for speed calculation (last bytes loaded, last timestamp)
  const metricsRef = useRef({ lastLoaded: 0, lastTime: 0 })

  // Track current successfully uploaded offset to use with slicing
  const offsetRef = useRef(0)

  // Helper to kill active requests silently
  const cleanup = useCallback(() => {
    if (xhrRef.current) {
      xhrRef.current.abort()
      xhrRef.current = null
    }
  }, [])
  // ...
```

Solving File Upload: start upload

```
const start = useCallback(
  (file: File, from: number = 0) => {
    cleanup()
    offsetRef.current = from
    metricsRef.current = { lastLoaded: from, lastTime: Date.now() }

    setState((prev) => ({
      ...prev,
      status: 'uploading',
      progress: file.size > 0 ? (from / file.size) * 100 : 0,
      bytes: from,
    }))

    const xhr = new XMLHttpRequest()
    xhrRef.current = xhr

    // Execute Request
    xhr.open('POST', 'http://localhost:3000/api/upload')
    xhr.setRequestHeader('X-File-Name', file.name)
    // Send only the part of the file we haven't uploaded yet!
    xhr.send(file.slice(from))

    // ... event listeners ...
  },
  [cleanup],
)
```

Solving File Upload: tracking progress

```
xhr.upload.onprogress = (e) => {
  if (!e.lengthComputable) return

  const totalLoaded = from + e.loaded
  offsetRef.current = totalLoaded

  const now = Date.now()
  const timeDiffMs = now - metricsRef.current.lastTime

  setState((prev) => {
    let speed = prev.speed
    // Throttle speed updates to every 500ms to avoid UI jitter
    if (timeDiffMs >= 500) {
      const loadedDiff = totalLoaded - metricsRef.current.lastLoaded
      speed = loadedDiff / 1024 / (timeDiffMs / 1000) // KB/s
      metricsRef.current = { lastLoaded: totalLoaded, lastTime: now }
    }

    const progress = file.size > 0 ? Math.min(100, (totalLoaded / file.size) * 100) : 0
    const remainingTimeMs = speed > 0 ? ((file.size - totalLoaded) / 1024 / speed) * 1000

    return { ...prev, status: 'uploading', progress, speed, bytes: totalLoaded, remainingT
  })
}
```

Solving File Upload: finilising upload

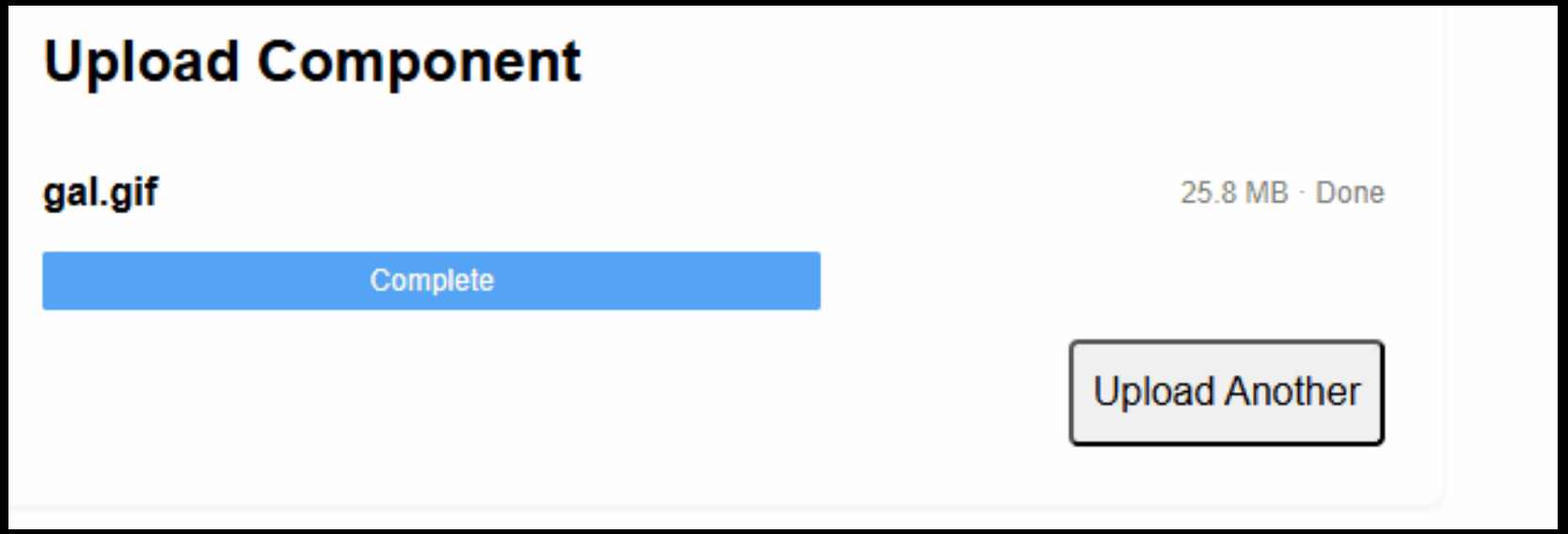
```
xhr.onload = () => {
  if (xhr.status >= 200 && xhr.status < 300) {
    offsetRef.current = file.size
    setState({
      status: 'completed',
      progress: 100,
      speed: 0,
      bytes: file.size,
      remainingTimeMs: 0,
      error: null,
    })
  } else {
    setState((prev) => ({
      ...prev,
      status: 'error',
      error: 'Upload failed',
      remainingTimeMs: null,
    }))
  }
}

xhr.onerror = () =>
  setState((prev) => ({ ...prev, status: 'error', error: 'Network error', remainingTimeMs:
```

Solving File Upload: pause / resume

```
const pause = useCallback(() => {
  cleanup()
  setState((prev) => ({ ...prev, status: 'paused', speed: 0, remainingTimeMs: null }))
}, [cleanup])

const resume = useCallback(
  (file: File) => {
    start(file, offsetRef.current)
  },
  [start],
)
```

Problem 15.2: File Upload component

Build a complete File Upload UI component utilizing a custom hook.



Upload Component: Adding file input

```tsx
export const UploadComponent = () => {
  const [
    { status, progress, speed, bytes, remainingTimeMs, error }, controls
  ] = useFileUpload()
  const [file, setFile] = useState<File | null>(null)

  const onFile = (e: React.ChangeEvent<HTMLInputElement>) => {
    const f = e.target.files?.[0]
    if (f) {
      setFile(f)
      controls.start(f)
    }
  }

  return (
    <div>
      {!file && (
        <input
          type="file"
          onChange={onFile}
          aria-label="Select a file to upload"
        />
      )}
```

Upload Component: Display progress

```
<div className={flex.flexRowBetween}>
  <b>{file.name}</b>
  <span className={flex.fontM} aria-live="polite">
    {formatBytes(bytes)} {status === 'uploading' && `· ${formatSpeed(speed)}`}
    {status === 'uploading' && remainingTimeMs !== null && ` · ${formatTime(remainingTimeMs)}`}
    {status === 'paused' && '· Paused'}
    {status === 'completed' && '· Done'}
  </span>
</div>

<ProgressBar
  value={progress}
  label={status === 'completed'
          ? 'Complete'
          : status === 'error'
              ? 'Failed'
              : `${Math.round(progress)}%`}
/>
```

Upload Component: Controls

```
<div className={cx(flex.flexRowGap8, flex.justifyEnd)}>
  {status === 'uploading' && (
    <button onClick={controls.pause} className={btnClass} aria-label="Pause upload">
      Pause
    </button>
  )}

  {status === 'paused' && file && (
    <button onClick={() => controls.resume(file)} className={btnClass} aria-label="Resume upload">
      Resume
    </button>
  )}

  {status !== 'completed' && (
    <button onClick={reset} className={btnClass} aria-label="Cancel upload">
      Cancel
    </button>
  )}

  {status === 'completed' && (
    <button onClick={reset} className={btnClass} aria-label="Upload another file">
      Upload Another
    </button>
  )}
</div>
```

Challenge 25: UnionToIntersection

Implement UnionToIntersection<U> which turns a union type U into an intersection type.

```
type Result = UnionToIntersection<{ a: 1 } | { b: 2 }>
// { a: 1 } & { b: 2 }
```

Challenge 25: UnionToIntersecction | Solution

Implement UnionToIntersection<U> which turns a union type U into an intersection type.

```
type UnionToIntersection<U> = (U extends any ? (arg: U) => any : never) extends (
  arg: infer I,
) => void
  ? I
  : never
```

Challenge 26: isAny

Implement IsAny<T> which takes an input type T and returns true if T is any, otherwise false.

```
type A = IsAny<any>     // true
type B = IsAny<unknown> // false
type C = IsAny<string>  // false
```

Challenge 26: isAny | Solution

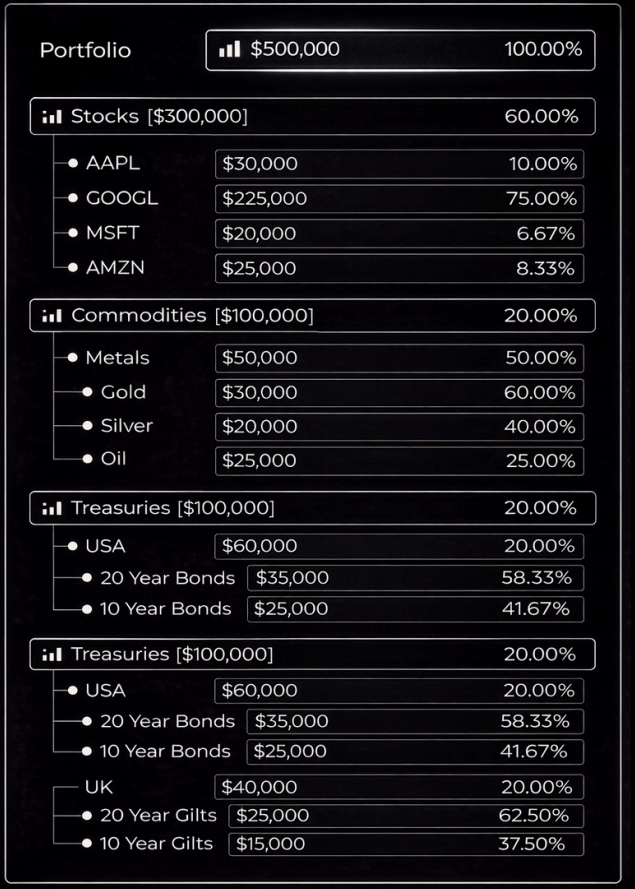
Implement IsAny<T> which takes an input type T and returns true if T is any, otherwise false.

```
1 type IsAny<T> = 0 extends 1 & T ? true : false
```

Problem 16: Portfolio Visualiser

Implement an interactive **Portfolio Visualizer** component that displays investment allocations as a hierarchical tree structure. Users can edit values at any node level.

1. **Tree Visualization** : Display portfolio as a collapsible tree with indentation.
2. **Value Display** : Show allocation amount and percentage of root for each node.
3. **Editable Nodes** : Allow inline editing of all nodes via text input fields. The update should only commit when the user presses "Enter".
4. **Budget Constraints : Any node's value is restricted by two rules:**
   - A parent cannot be reduced below the sum of its children's values.
   - A child cannot be increased beyond the "Unallocated cash" budget of its immediate parent.



Level: Hard

Portfolio Visualiser: Input Data

```
const portfolioData = {
  name: 'Total Portfolio',
  value: 10000,
  children: [
    {
      name: 'Stocks',
      value: 8000,
      children: [
        { name: 'AAPL', value: 5000 },
        { name: 'TSLA', value: 3000 },
      ],
    },
    { name: 'Bonds', value: 1500 },
  ],
}
```

Portfolio Visualiser: Preview

Portfolio	300000	100.00%
Stocks	100000	33.33%
AAPL	30000	30.00%
GOOGL	25000	25.00%
MSFT	20000	20.00%
AMZN	25000	25.00%
Commodities	100000	33.33%
Metals	50000	50.00%
Gold	30000	60.00%
Silver	20000	40.00%
Oil	25000	25.00%
Gas	25000	25.00%
Treasuries	100000	33.33%
USA	60000	60.00%
20 Year Bonds	35000	58.33%
10 Year Bonds	25000	41.67%
UK	40000	40.00%
20 Year Gilts	25000	62.50%
10 Year Gilts	15000	37.50%

Portfolio Visualiser: Defining Property model

```ts
/**
 * Represents the raw portfolio data structure.
 */
export type TPortfolioNode = {
  id: string
  name: string
  value: number
  children?: TPortfolioNode[]
}

type TPortfolioVisualizerProps = {
  data: TPortfolioNode
}
```

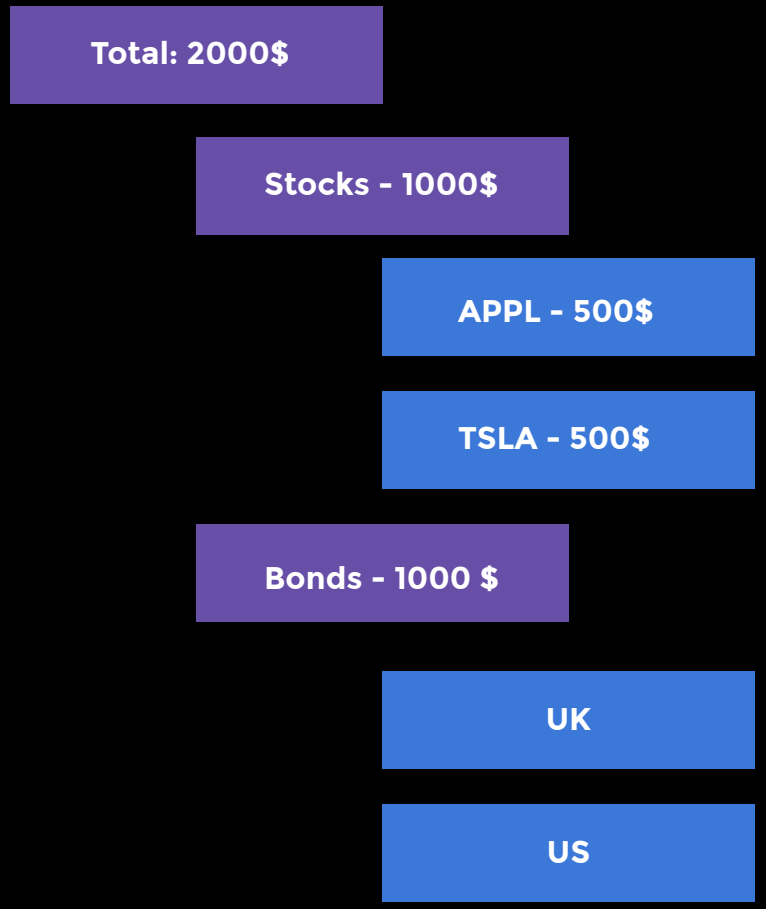
Portfolio Visualiser: Creating Node

```
/**
 * Renders a single node in the portfolio tree recursively.
 */
function PortfolioNode({
  id,
  name,
  value,
  children,
  total,
}: TPortfolioNode & {
  total: number
}) {
  const percentage = total > 0 ? ((value / total) * 100).toFixed(2) : '0.00'

  const hasChildren = children && children.length > 0
  return (
    <details className={cx(styles.paddingLeft16, styles.paddingVer8)} open>
      <summary className={css.listNone}>
        <div className={cx(styles.flexRowBetween, styles.flexRowGap16)}>
          <strong>{name}</strong>
          <div className={cx(styles.flexRowGap8)}>
            <input
              data-node-id={id}
              type="text"
              defaultValue={value}
              readOnly={hasChildren}
            />
            <output className={cx(styles.textLeft, styles.w100px)}>{percentage}%</output>
          </div>
        </div>
      </summary>
      {hasChildren && children!.map((ch) => <PortfolioNode total={value} key={ch.id} {...ch} />)}
    </details>
  )
}
```

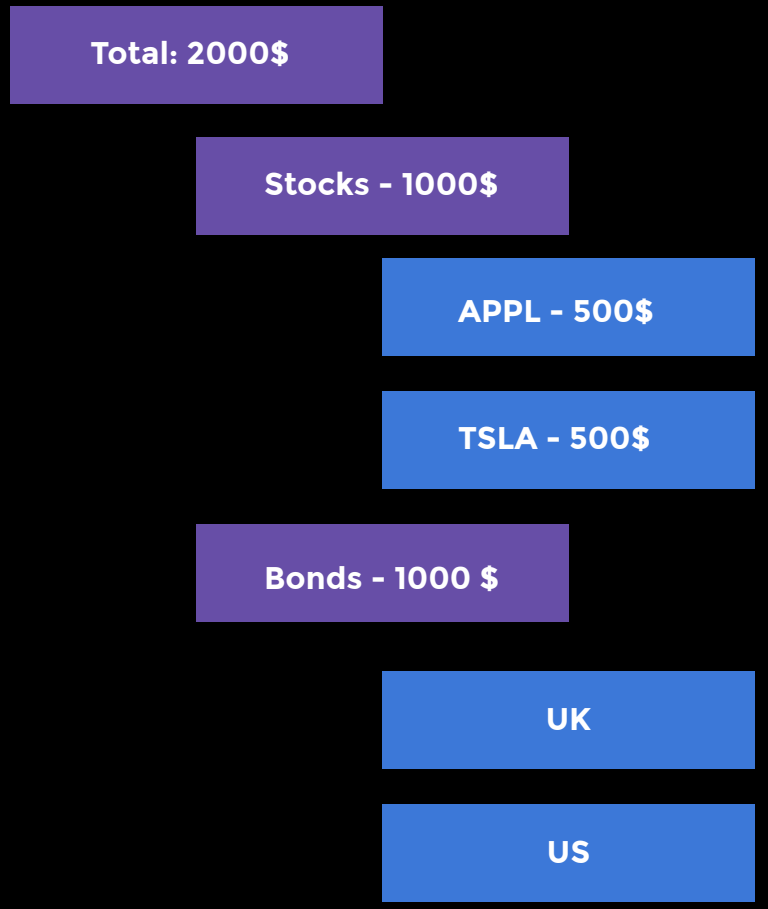
Portfolio Visualiser: Creating Root

```tsx
export function PortfolioVisualizer({ data }: TPortfolioVisualizerProps) {
  return (
    <div
      className={cx(
        styles.maxW600px,
        styles.padding16,
        styles.bgWhite5,
        styles.border4px,
        styles.br8,
      )}
    >
      <PortfolioNode total={data.value} {...data} />
    </div>
  )
}
```

Portfolio Visualiser: Follow up requirements

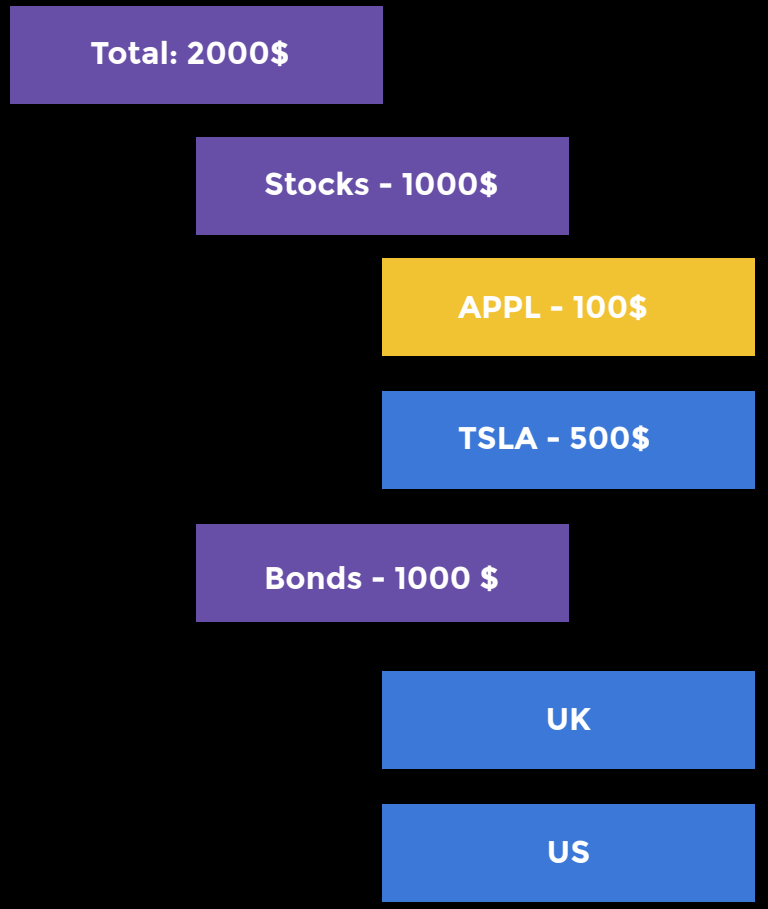


Portfolio Visualiser: Follow up requirements



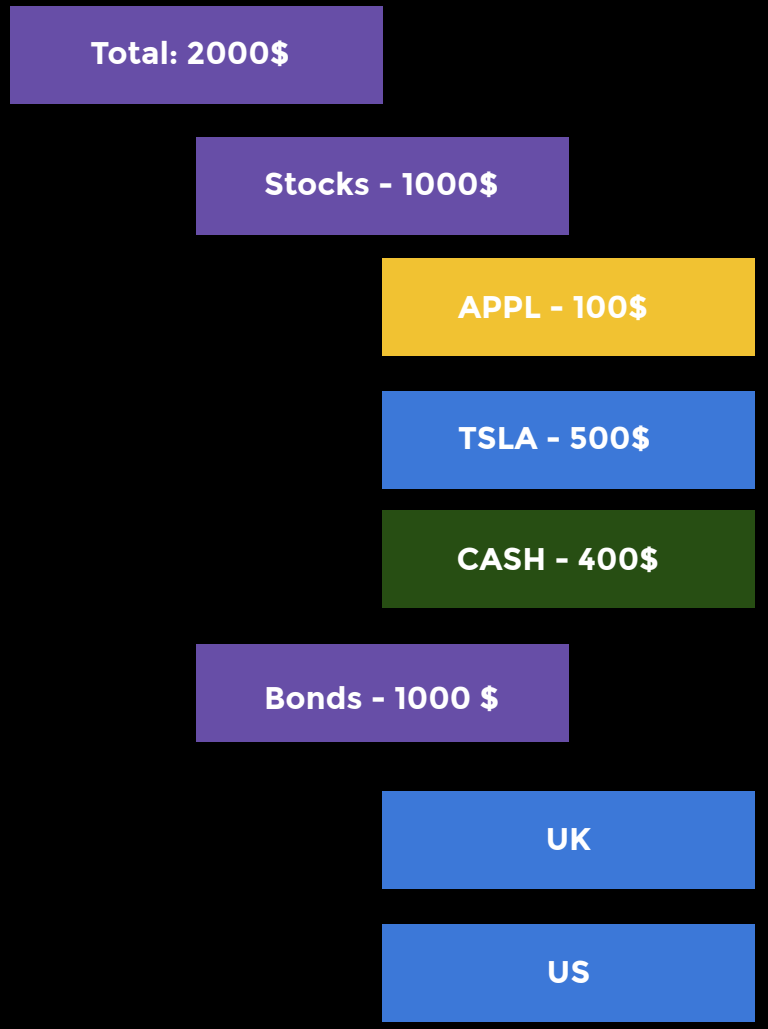
Editable Nodes: Allow node editing - we can change allocated budget to the node

Portfolio Visualiser: Follow up requirements



Editable Nodes: Allow node editing - we can change allocated budget to the node

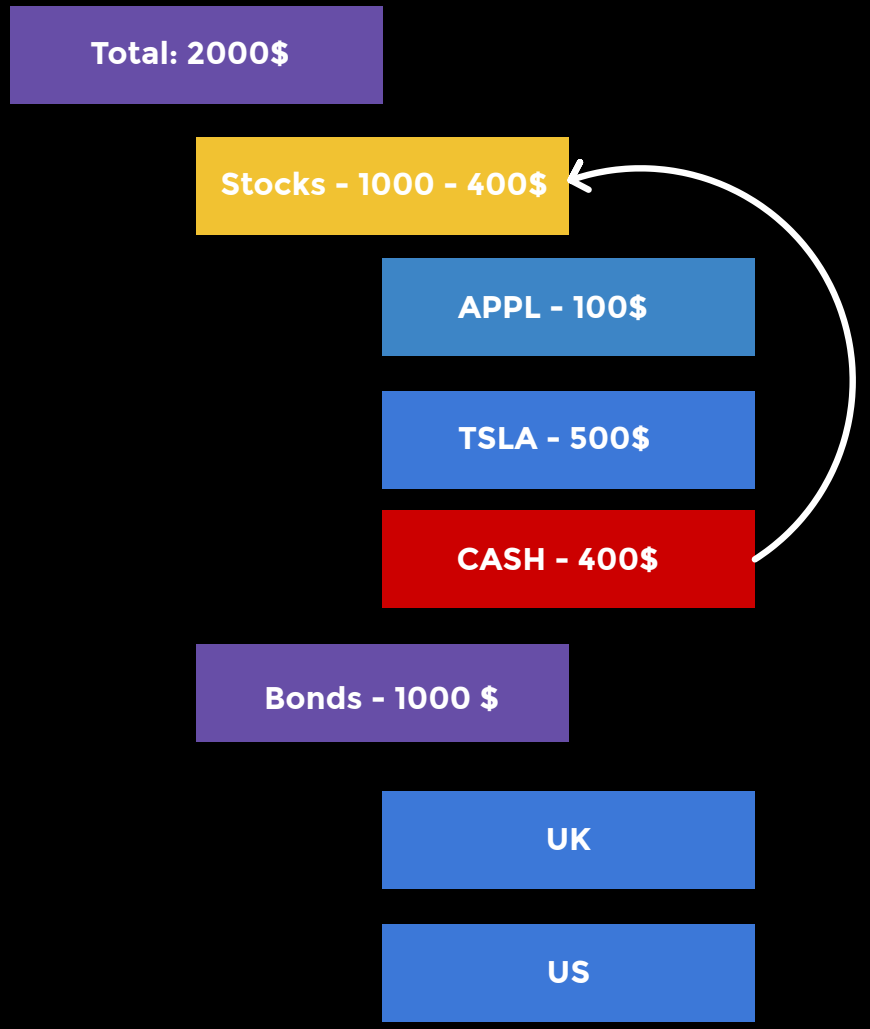
Portfolio Visualiser: Follow up requirements



Editable Nodes: Allow node editing - we can change allocated budget to the node

Unallocated Cash: If a parent's value sum of children is strictly less than its value, dynamically render a read-only leaf node titled `Unallocated cash` at the bottom of its child list showing the remainder.

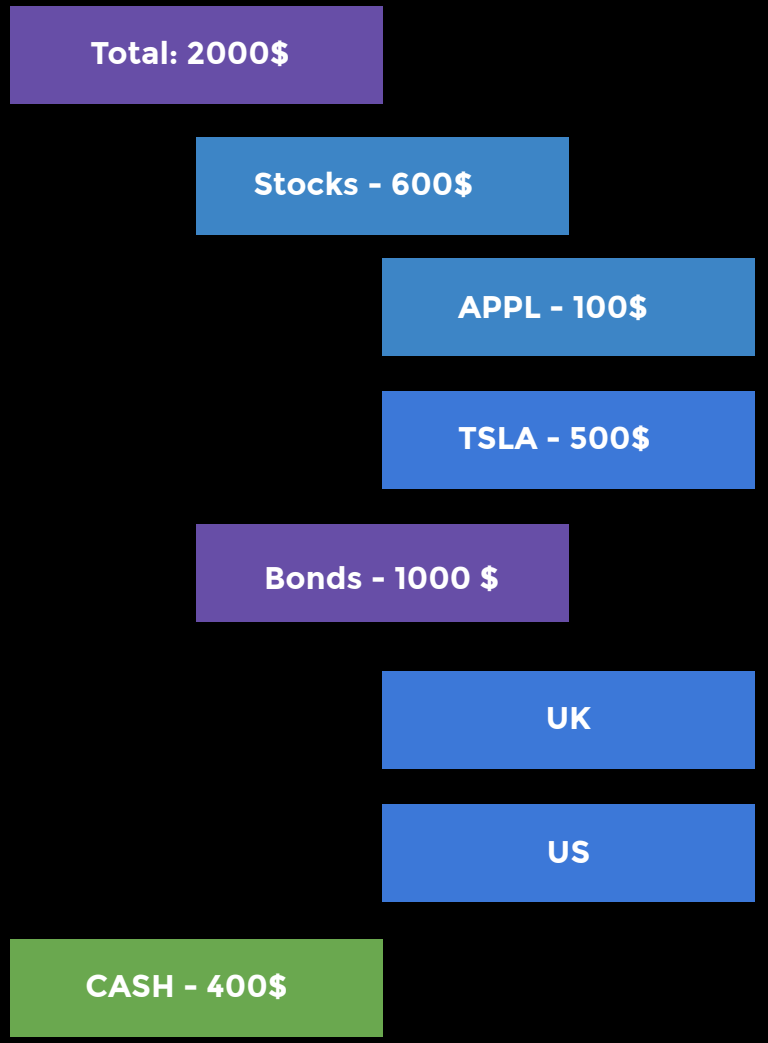
Portfolio Visualiser: Follow up requirements



Editable Nodes: Allow node editing - we can change allocated budget to the node

Unallocated Cash: If a parent's value sum of children is strictly less than its value, dynamically render a read-only leaf node titled `Unallocated cash` at the bottom of its child list showing the remainder.

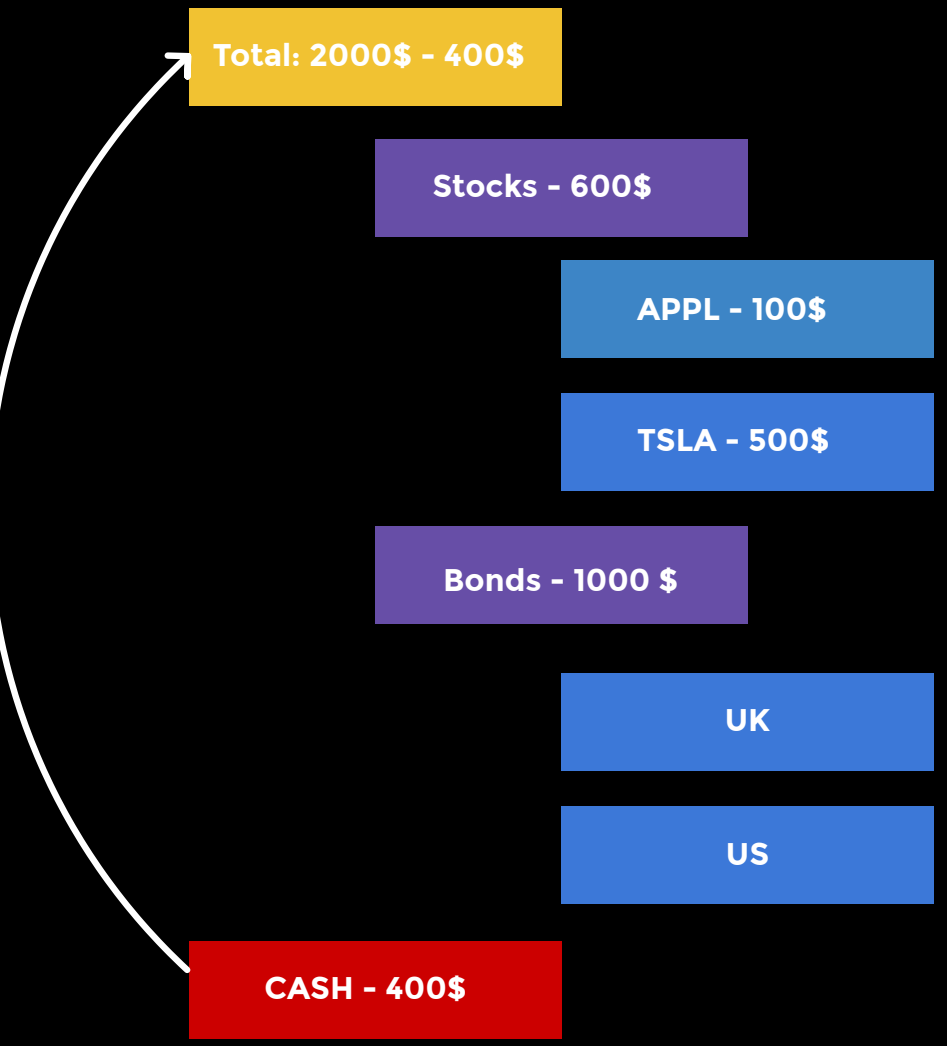
Portfolio Visualiser: Follow up requirements



Editable Nodes: Allow node editing - we can change allocated budget to the node

Unallocated Cash: If a parent's value sum of children is strictly less than its value, dynamically render a read-only leaf node titled `Unallocated cash` at the bottom of its child list showing the remainder.

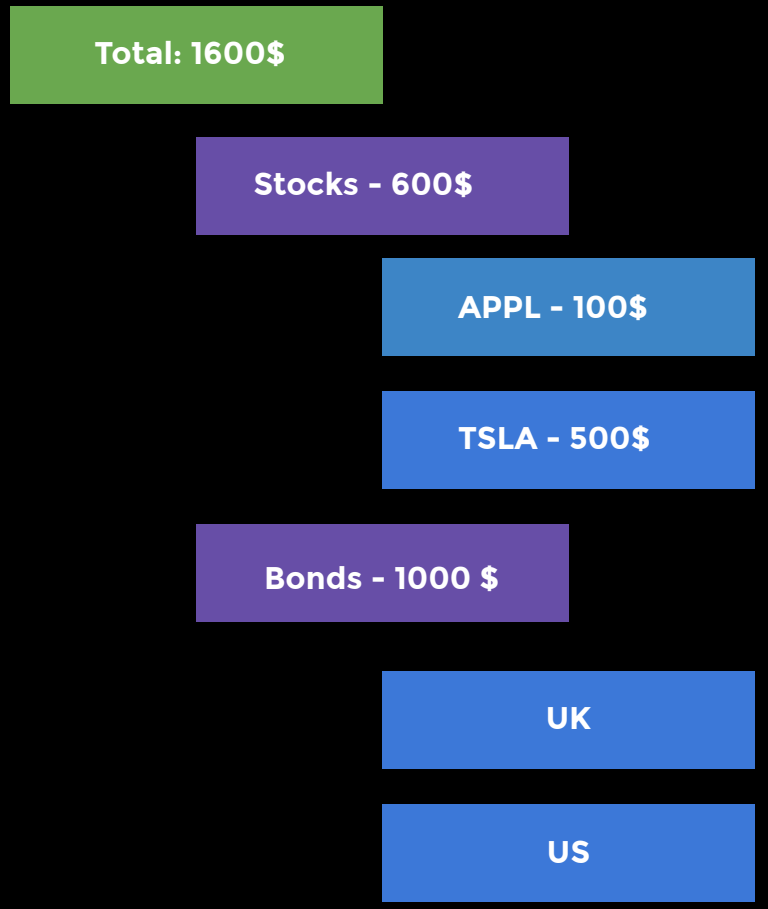
Portfolio Visualiser: Follow up requirements



Editable Nodes: Allow node editing - we can change allocated budget to the node

Unallocated Cash: If a parent's value sum of children is strictly less than its value, dynamically render a read-only leaf node titled `Unallocated cash` at the bottom of its child list showing the remainder.

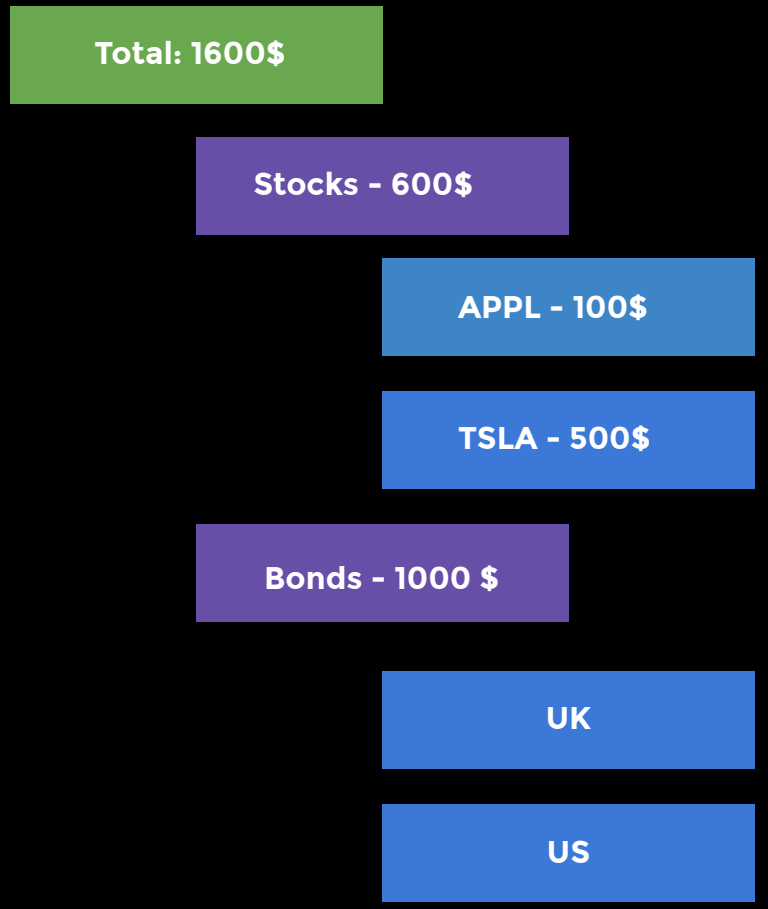
Portfolio Visualiser: Follow up requirements



Editable Nodes: Allow node editing - we can change allocated budget to the node

Unallocated Cash: If a parent's value sum of children is strictly less than its value, dynamically render a read-only leaf node titled `Unallocated cash` at the bottom of its child list showing the remainder.

Portfolio Visualiser: Follow up requirements



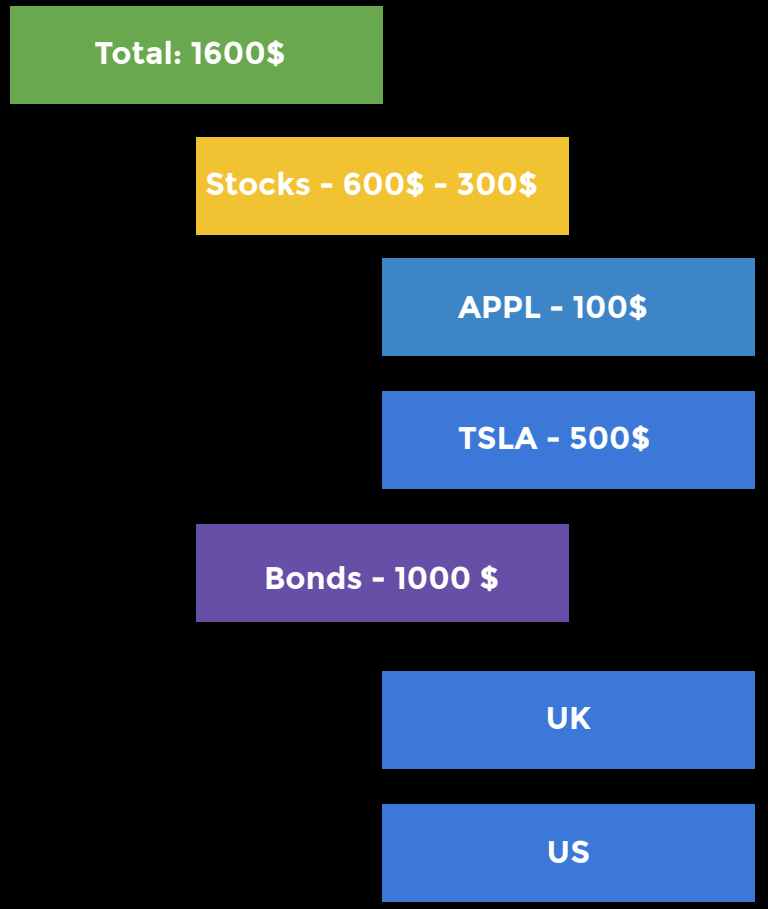
Editable Nodes: Allow node editing - we can change allocated budget to the node

Unallocated Cash: If a parent's value sum of children is strictly less than its value, dynamically render a read-only leaf node titled `Unallocated cash` at the bottom of its child list showing the remainder.

Budgest Constrains: any node's value is restricted by two rules:

- A parent cannot be reduced below the sum of its children's values.
- A child cannot be increased beyond the **`Unallocated`** cash budget of its immediate parent.

Portfolio Visualiser: Follow up requirements



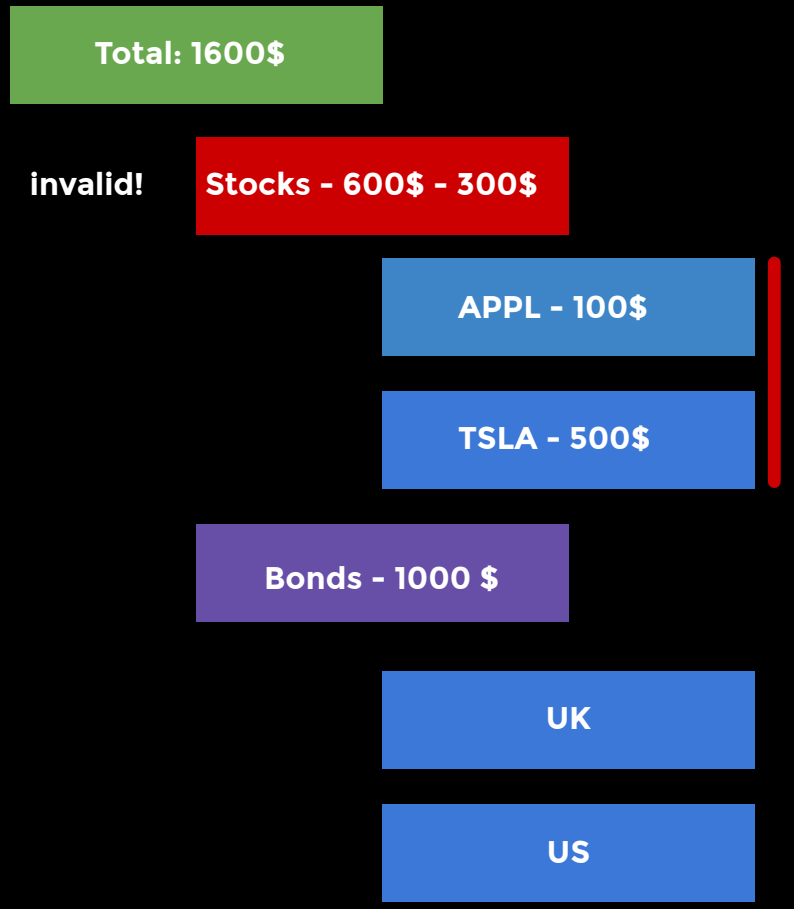
Editable Nodes: Allow node editing - we can change allocated budget to the node

Unallocated Cash: If a parent's value sum of children is strictly less than its value, dynamically render a read-only leaf node titled `Unallocated cash` at the bottom of its child list showing the remainder.

Budgest Constrains: any node's value is restricted by two rules:

- A parent cannot be reduced below the sum of its children's values.
- A child cannot be increased beyond the **`Unallocated`** cash budget of its immediate parent.

Portfolio Visualiser: Follow up requirements



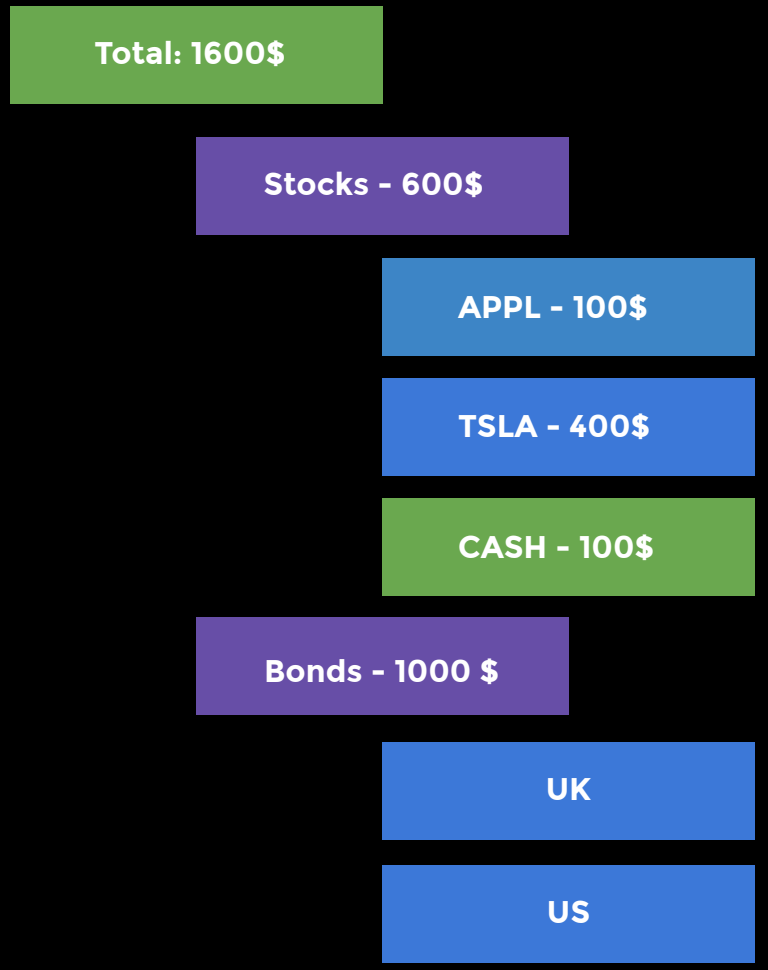
Editable Nodes: Allow node editing - we can change allocated budget to the node

Unallocated Cash: If a parent's value sum of children is strictly less than its value, dynamically render a read-only leaf node titled `Unallocated cash` at the bottom of its child list showing the remainder.

Budgest Constrains: any node's value is restricted by two rules:

- A parent cannot be reduced below the sum of its children's values.
- A child cannot be increased beyond the **`Unallocated`** cash budget of its immediate parent.

Portfolio Visualiser: Follow up requirements



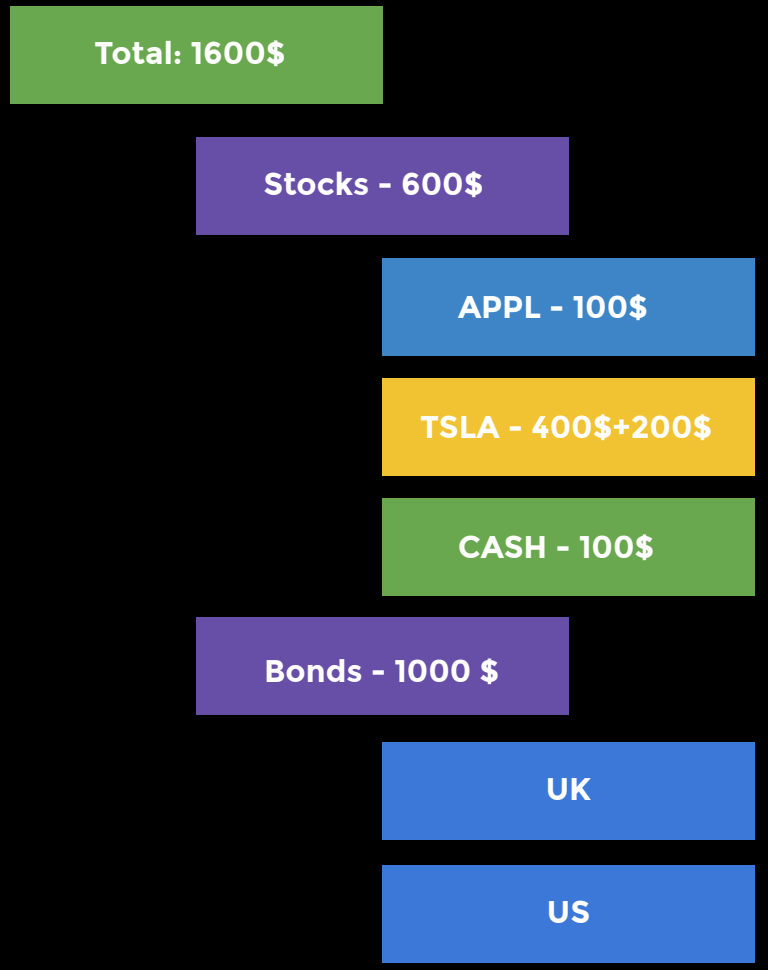
Editable Nodes: Allow node editing - we can change allocated budget to the node

Unallocated Cash: If a parent's value sum of children is strictly less than its value, dynamically render a read-only leaf node titled `Unallocated cash` at the bottom of its child list showing the remainder.

Budgest Constrains: any node's value is restricted by two rules:

- A parent cannot be reduced below the sum of its children's values.
- A child cannot be increased beyond the **`Unallocated`** cash budget of its immediate parent.

Portfolio Visualiser: Follow up requirements



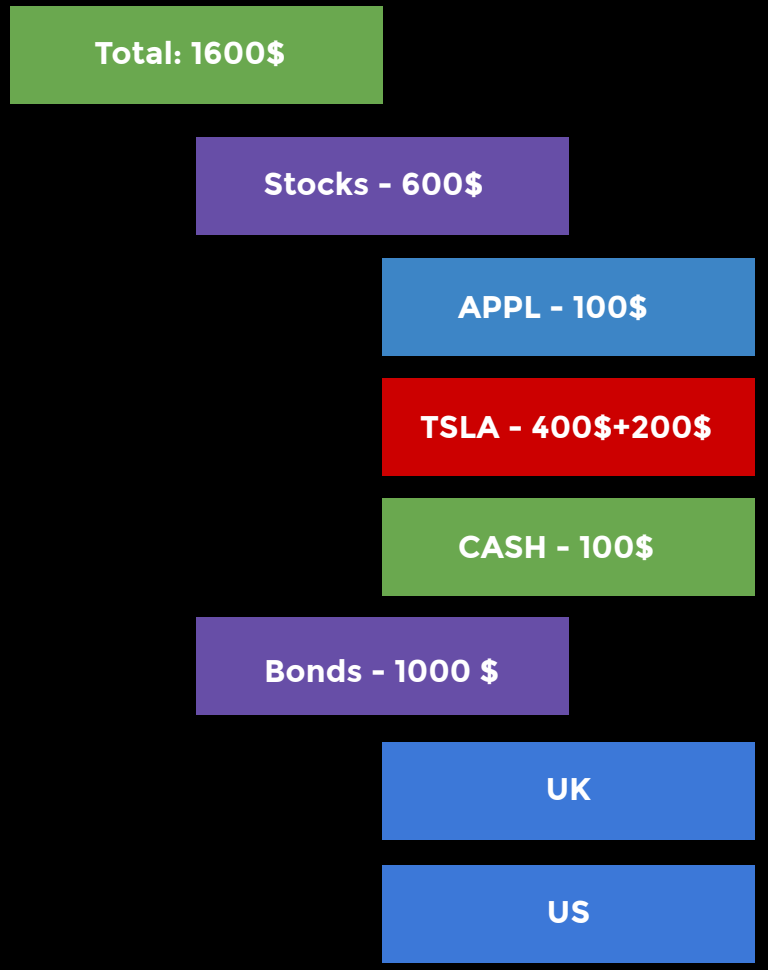
Editable Nodes: Allow node editing - we can change allocated budget to the node

Unallocated Cash: If a parent's value sum of children is strictly less than its value, dynamically render a read-only leaf node titled `Unallocated cash` at the bottom of its child list showing the remainder.

Budgest Constrains: any node's value is restricted by two rules:

- A parent cannot be reduced below the sum of its children's values.
- A child cannot be increased beyond the **`Unallocated`** cash budget of its immediate parent.

Portfolio Visualiser: Follow up requirements



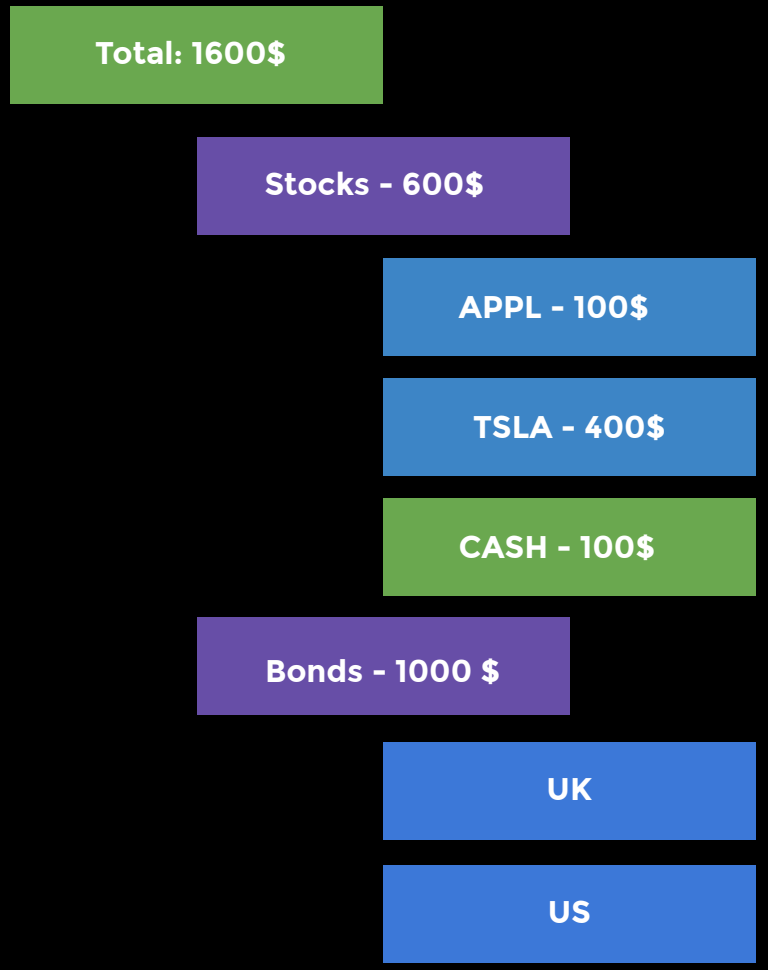
Editable Nodes: Allow node editing - we can change allocated budget to the node

Unallocated Cash: If a parent's value sum of children is strictly less than its value, dynamically render a read-only leaf node titled `Unallocated cash` at the bottom of its child list showing the remainder.

Budgest Constrains: any node's value is restricted by two rules:

- A parent cannot be reduced below the sum of its children's values.
- A child cannot be increased beyond the **`Unallocated`** cash budget of its immediate parent.

Portfolio Visualiser: Follow up requirements



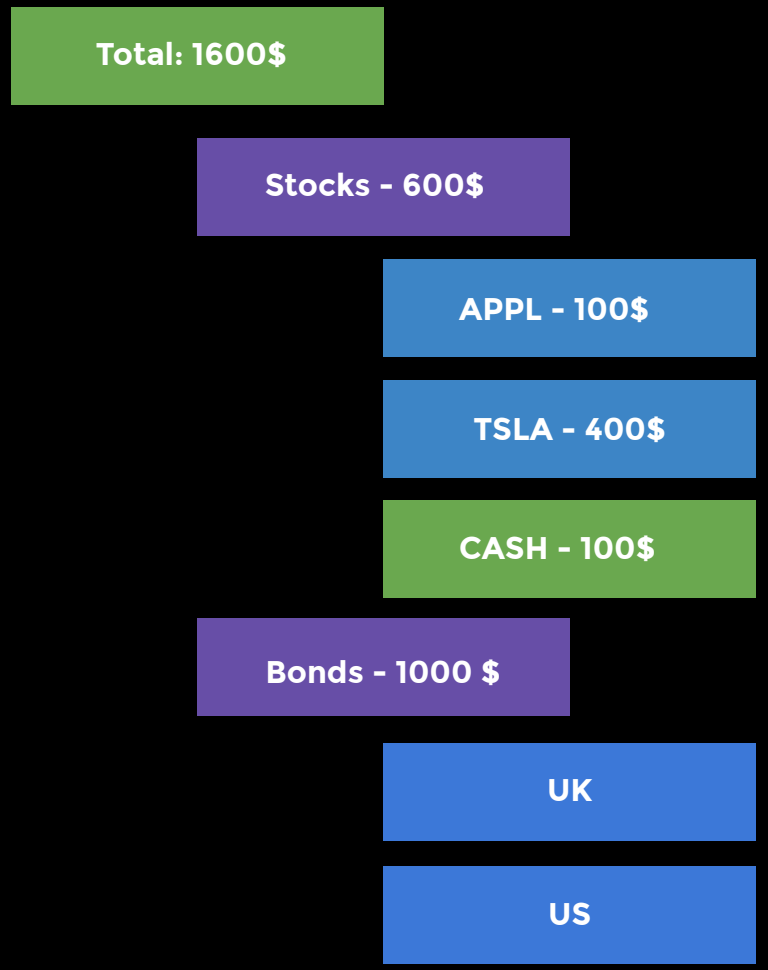
Editable Nodes: Allow node editing - we can change allocated budget to the node

Unallocated Cash: If a parent's value sum of children is strictly less than its value, dynamically render a read-only leaf node titled `Unallocated cash` at the bottom of its child list showing the remainder.

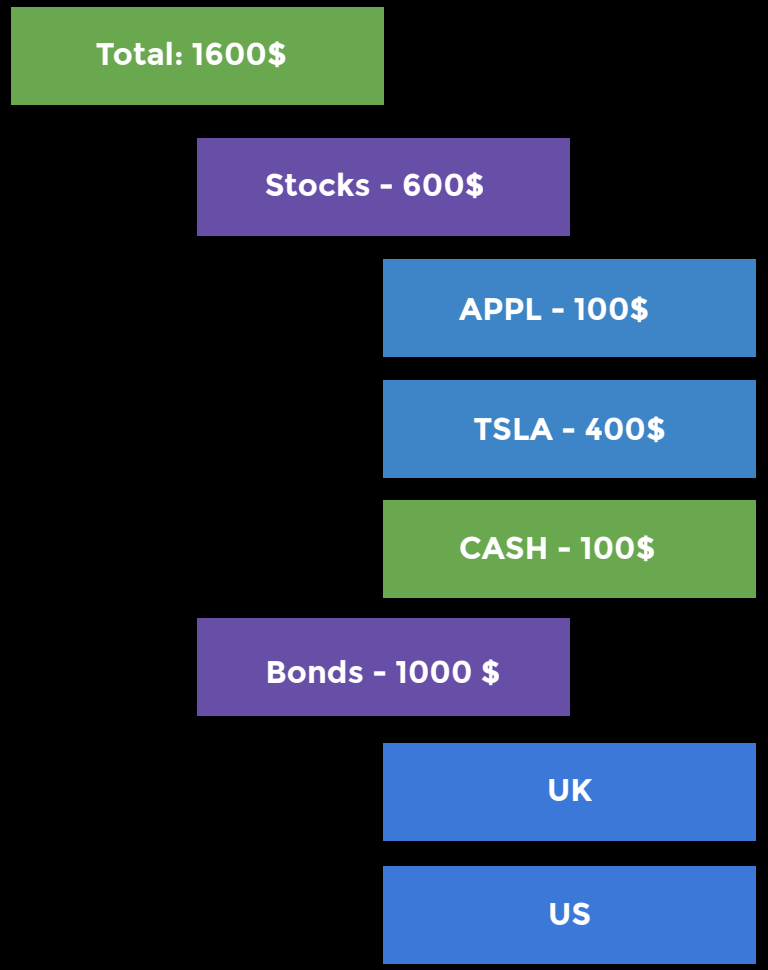
Budgest Constrains: any node's value is restricted by two rules:

- A parent cannot be reduced below the sum of its children's values.
- A child cannot be increased beyond the **`Unallocated`** cash budget of its immediate parent.

Portfolio Visualiser: Solution Approach



Portfolio Visualiser: Solution Approach



Portfolio Visualiser: State pre-processing

```
1
2  /**
3   * Recursively flattens the tree into a map instance with `parentID` references
4   * for bottom-up state propagation without deep nesting traversal logic.
5   */
6  function prepare(
7    data: TPortfolioNode,
8    parentID: string | null,
9    acc: Map<string, TPortfolioStateNode> = new Map<string, TPortfolioStateNode>(),
10  ): [TPortfolioStateNode, Map<string, TPortfolioStateNode>] {
11   const node: TPortfolioStateNode = {
12     ...data,
13     parentID,
14     children: data.children?.map((ch) => prepare(ch, data.id, acc)[0]) || [],
15   }
16   acc.set(data.id, node)
17   return [node, acc]
18 }
```

Portfolio Visualiser: Update Portfolio Node

```
function PortfolioNode({ id, name, value, children, total, readonly }) {
  const percentage = total > 0 ? ((value / total) * 100).toFixed(2) : '0.00'
  const [inputValue, setInputValue] = useState(String(value))

  // Sync when tree is updated from the top down via Enter
  useEffect(() => setInputValue(String(value)), [value])

  const hasChildren = !!children?.length
  const unallocated = hasChildren ? value - children!.reduce((sum, ch) => sum + ch.value, 0) : 0

  return (
    <details open>
      <summary>
        <input
          data-node-id={id}
          value={readonly ? value : inputValue}
          onChange={(e) => !readonly && setInputValue(e.target.value)}
          onBlur={() => setInputValue(String(value))} // Revert if blurred without Enter
        />
        <output>{percentage}%</output>
      </summary>

      {hasChildren && children!.map((ch) => <PortfolioNode total={value} key={ch.id} {...ch} />)}

      {hasChildren && unallocated > 0 && (
        <PortfolioNode id={`${id}-unallocated`} name="Unallocated cash" value={unallocated} total={value} readonly />
      )}
    </details>
  )
}
```

Portfolio Visualiser: Enter new data

```
/**
   * Event delegation on the root container, saving us from binding
   * deeply nested callback functions throughout the tree.
   */
  const handleKeyDown = (e: React.KeyboardEvent<HTMLDivElement>) => {
    if (e.key === 'Enter' && e.target instanceof HTMLInputElement) {
      const { dataset: { nodeId }, value } = e.target;
      if (!nodeId) {
        return;
      }
      const num = Number(value)
      if (!isNaN(num)) {
        handleUpdate(nodeId, num)
      }
    }
  }
```

Portfolio Visualiser: Update data

```
/**
  * Main state updater. Validates the fixed-budget tree rules:
  * 1. Parents cannot be reduced below the sum of their direct children.
  * 2. Children cannot be increased above their parent's immediate unallocated cash bucket.
  */
 const handleUpdate = (id: string, newValue: number) => {
   const node = store.get(id)
   if (!node) return

   // Rule 1: A parent node's explicit budget cannot be reduced below
   // what its children are already consuming.
   if (!!node?.children?.length && newValue < getSum(node)) {
     return
   }
   const delta = newValue - node.value
   // Rule 2: A child node cannot be increased if its parent does not have
   // enough "Unallocated cash" remaining to absorb the increase.
   if (delta > 0 && node.parentID) {
     const parent = store.get(node.parentID)
     if (!parent || delta > parent.value - getSum(parent)) {
       return;
     }
   }
   const newStore = structuredClone(store)
   newStore.get(id)!.value = newValue
   setStore(newStore)
 }
```

Portfolio Visualiser: Result

Problem 17-18: ChatGPT Client

Implement a basic version of ChatGPT experience that supports:

1. Dynamic text streaming from backend
2. Markdown transformation with a set of pre-defined rules

Markdown should support:

1. **Tables**
2. **Headings (h1-h3)**
3. **Unordered and Ordered lists**
4. **Links**
5. **Text Formatting (italic, bold, strikethrough)**

Important: You're provided with an API endpoint and a set of **RegExp** to build the basic language parsing, no need to come up with custom regular expressions.

Supported Markdown Syntax

Links	`/(^|[^!])\[([^[]+)\]\(([^)]+)\)/g`	Excludes image links `![...]`
Headers	`/^#{1}\s?([^\n]+)/gm`	(And similar for `h2`, `h3`, `h4`)
Bold	`/(?<!\*)\*\*(?!\*)(.+?)(?<!\*)\*\*(?!\*)/g`	Uses lookarounds to avoid overlapping with italic
Italic	`/(?<!\*)\*(?!\*)(.+?)(?<!\*)\*(?!\*)/g`	Extracts text between single asterisks
~~**Strikethrough**~~	`/~~\s?([^\n]+?)~~/g`	Wraps deleted text blocks
Tables	`/^(|.+|\r?\n)(|[-:| ]+|\r?\n)((?:|.*|\r?\n?)*)/gm`	Uses custom parsing function for rows
O. Lists	`/(?:^|\n)(?![^\n]*<[^>]*>)(\s*[0-9]+\.\s.*)+/g`	Handles ordered lists avoiding raw HTML
U. Lists	`/(?:^|\n)(?![^\n]*<[^>]*>)(\s*[-+]\s.*(?:\n\s*[-+]\s.*)*)/g`	Handles unordered lists avoiding raw HTML
Paragraph	`/^(?!#)(?!.*<\/?(ul|ol|h1|...|p)>)(.*\S.*)$/gm`	Wraps unformatted text blocks

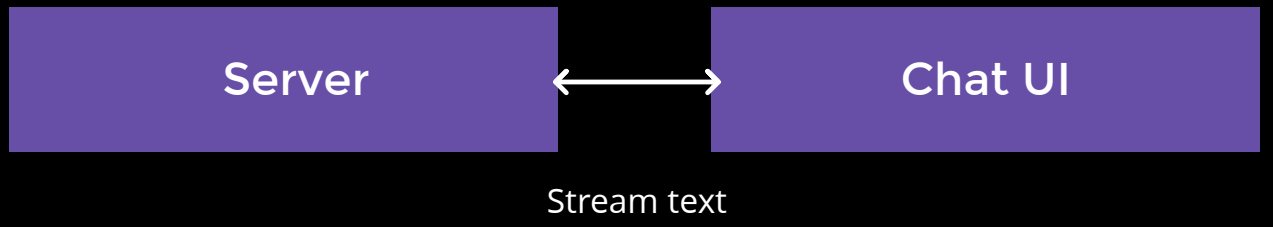
`GET /api/stream-markdown`

Level: EXTREME

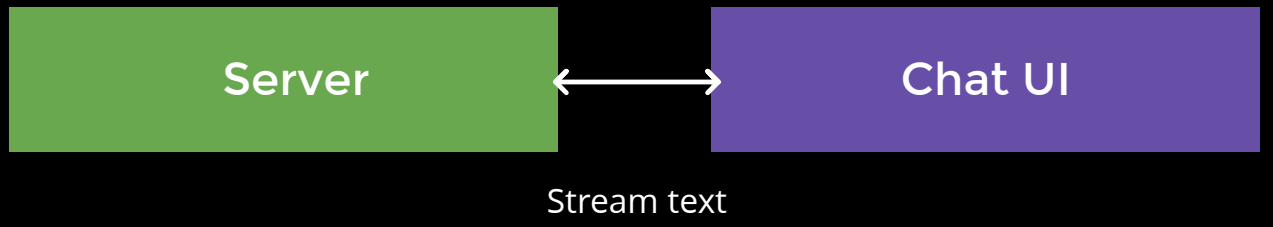
ChatGPT Client: Docomposing a problem

Chat UI

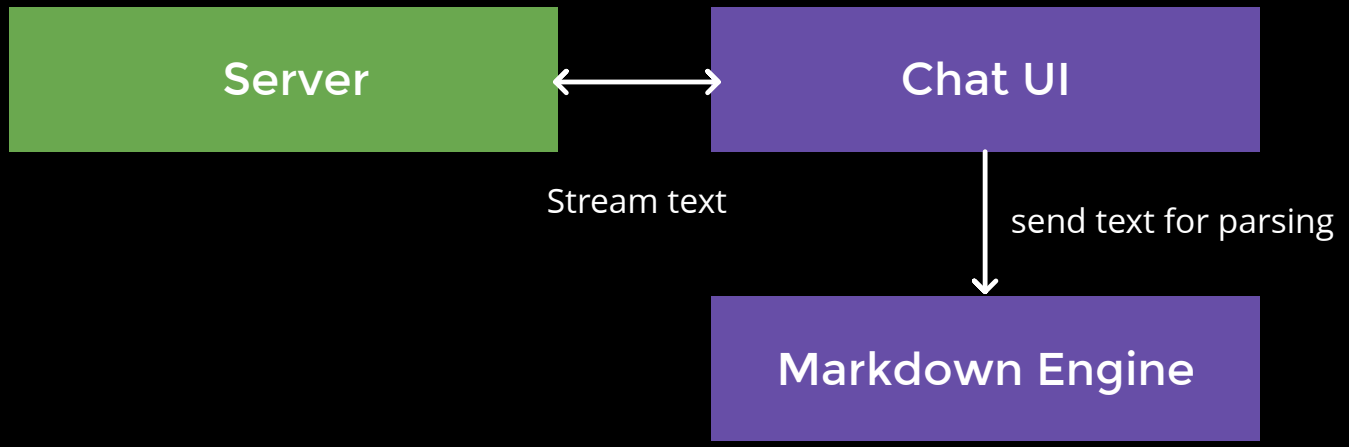
ChatGPT Client: Docomposing a problem



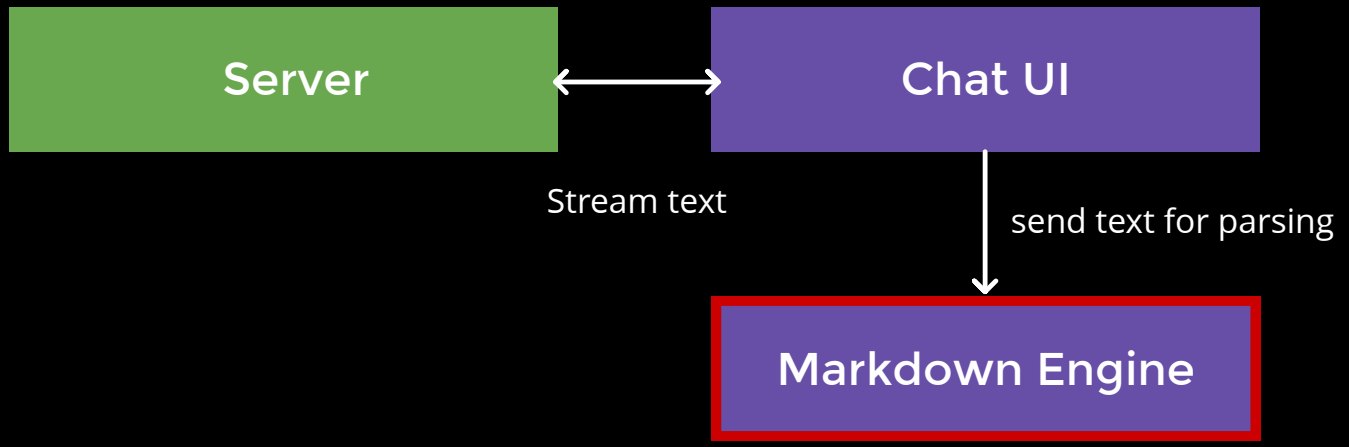
ChatGPT Client: Docomposing a problem



ChatGPT Client: Docomposing a problem

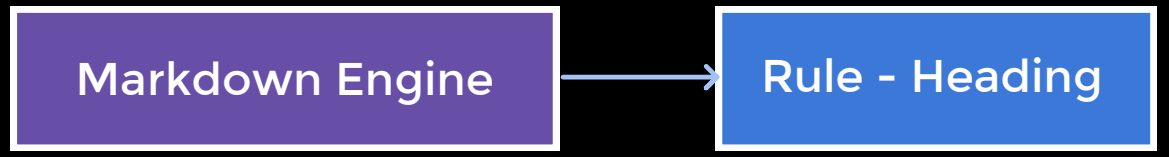


ChatGPT Client: Docomposing a problem

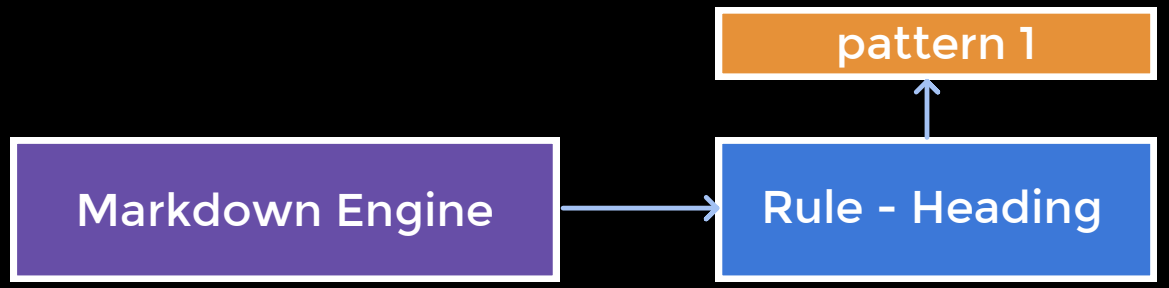


ChatGPT Client: Docomposing a problem

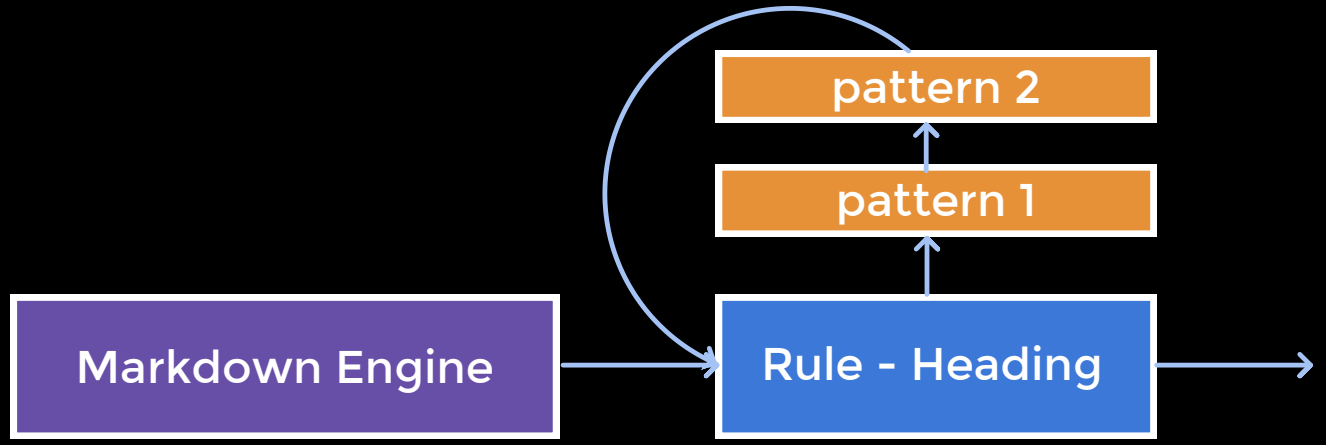
ChatGPT Client: Docomposing a problem



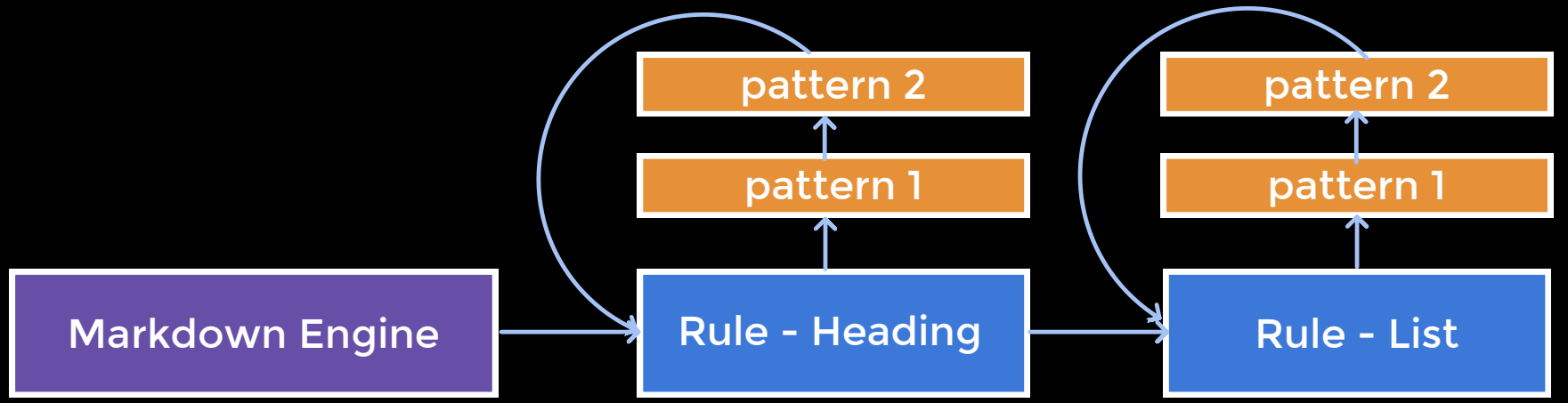
ChatGPT Client: Docomposing a problem



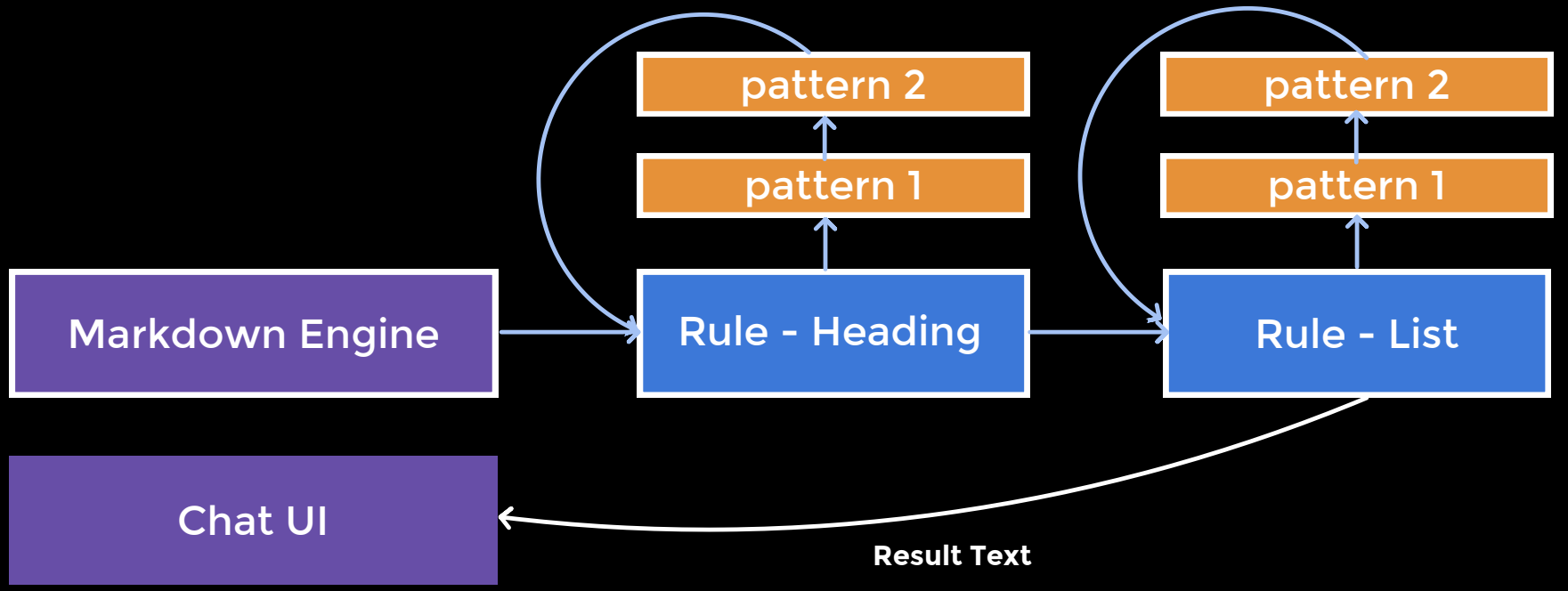
ChatGPT Client: Docomposing a problem



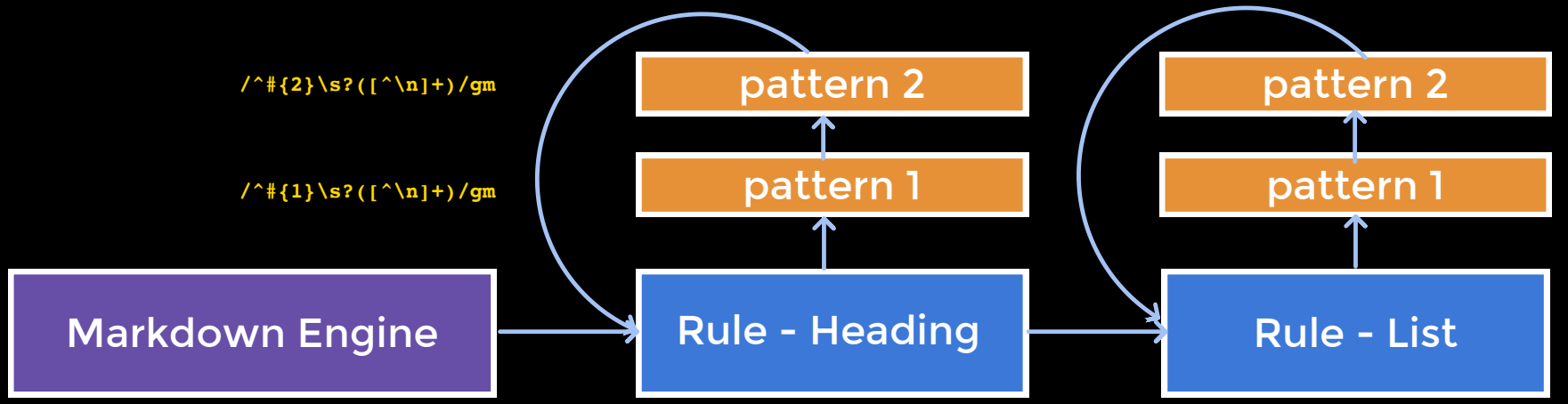
ChatGPT Client: Docomposing a problem



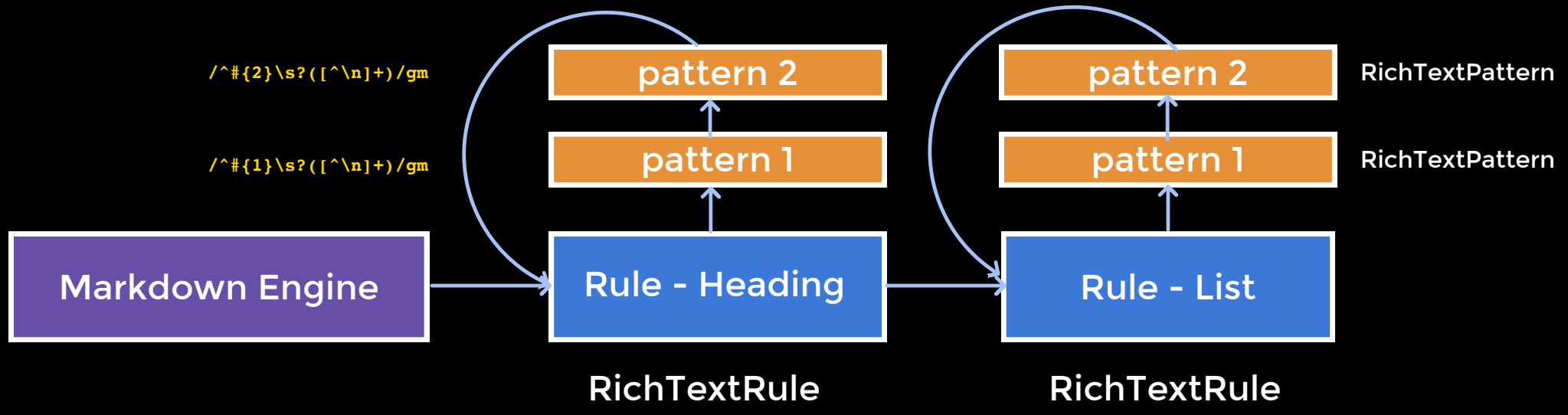
ChatGPT Client: Docomposing a problem



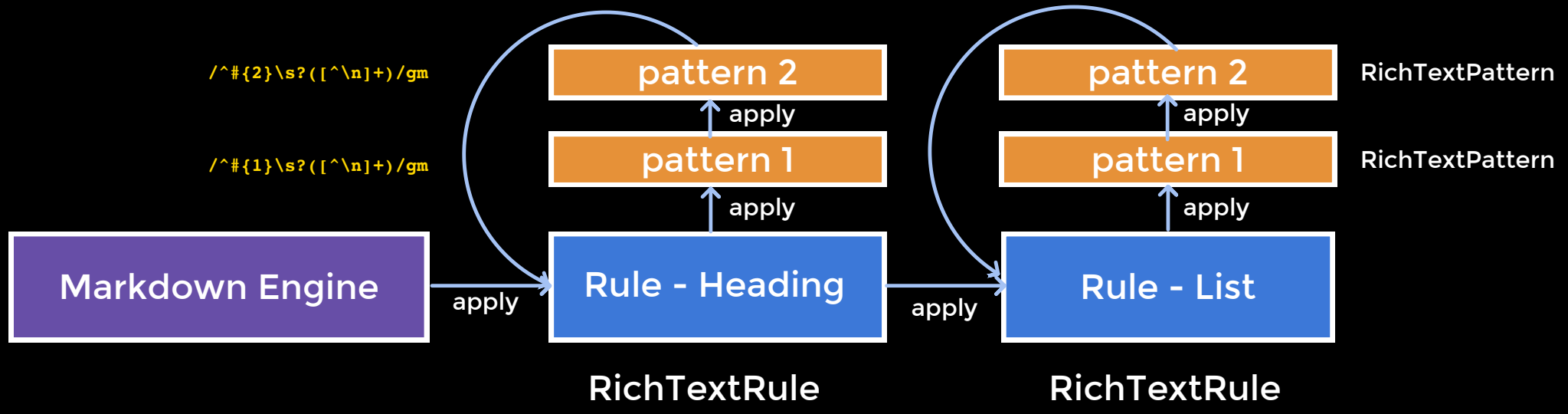
ChatGPT Client: Creating Markdown Engine



Markdown Engine: Converting to OOP structure



Markdown Engine: Converting to OOP structure



Markdown Engine: Understanding Groups

Matches: [text](url)

group 1

```
/(^|[^!])\[([^[]+)\]\(([^)]+)\)/g
```

Markdown Engine: Understanding Groups

Matches: [text](url)

group 1 group 2 -text group 3 - url

```
/(^|[^!])\[([^[]+)\]\(([^)]+)\)/g
```

Markdown Engine: Replacer logic

Matches: [text](url)

group 1 group 2 -text group 3 - url

```
/(^|[^!])\[([^[]+)\]\(([^)]+)\)/g
```

```
"[super link](https://example.com)".replace(regexp, '$1<a href="$3">$2</a>')
```

Markdown Engine: Replacer logic

Matches: [text](url)

group 1 **group 2 -text** **group 3 - url**

```
/(^|[^!])\[([^[]+)\]\(([^)]+)\)/g
```

```
"[super link](https://example.com)".replace(regexp, '$1<a href="$3">$2</a>')
```

```
"<a href='https://example.com'>super link</a>"
```

Markdown Engine: Converting to OOP structure

```
export class TRichTextPattern {
  constructor(
     public regexp: RegExp,
     public replacer: string | Function
  ) {}

  apply(text: string) {
    return text.replace(this.regexp, this.replacer as any)
  }
}

export class TRichTextRule {
  constructor(
     public name: string,
     public patterns: TRichTextPattern[]
  ) {}

  apply(text: string) {
    return this.patterns.reduce((acc, pattern) => pattern.apply(acc), text)
  }
}
```

Markdown Engine: Rules & Replacers

```
const BOLD_RULE = new TRichTextRule('bold', [
  new TRichTextPattern(REGEX_BOLD, '<b>$1</b>')
])

const HEADER_RULE = new TRichTextRule('header', [
  new TRichTextPattern(REGEX_HEADER_4, '<h4>$1</h4>'),
  new TRichTextPattern(REGEX_HEADER_3, '<h3>$1</h3>'),
  new TRichTextPattern(REGEX_HEADER_2, '<h2>$1</h2>'),
  new TRichTextPattern(REGEX_HEADER_1, '<h1>$1</h1>'),
])
```

Markdown Engine: Rules & Replacers

```
const UNORDERED_LIST_REPLACER = (fullMatch: string) => {
  // 1. Split the block by newlines
  // 2. Wrap each line substring in <li> tags
  const items = fullMatch
    .trim()
    .split('\n')
    .reduce((acc, next) => acc + '<li>' + next.substring(2) + '</li>', '')

  // 3. Wrap the compiled list items in the parent <ul> container
  return `<ul>${items}</ul>\n`
}

const LIST_RULE = new TRichTextRule('lists', [
  new TRichTextPattern(REGEX_UNORDERED_LIST, UNORDERED_LIST_REPLACER)
])
```

Markdown Engine: Rules & Replacers

```
export const RICH_TEXT_RULES: Array<TRichTextRule> = [
  LINK_RULE,
  HEADER_RULE,
  TABLE_RULE,
  LIST_RULE,
  PARAGRAPH_RULE,
  FORMATTING_RULE,
]
```

Markdown Engine: sanitizing source text

```
function cleanMarkdownFromHTML(text: string) {
  const xmlParser = new DOMParser()
  try {
    const xml = xmlParser.parseFromString(`<section>${text}</section>`, 'application/xml')
    return xml.firstElementChild?.textContent || text // Extracts only flat text, stripping tags
  } catch {
    return text
  }
}
```

Markdown Engine: Executing rules

```
export function parseRichText(text: string, rules: Array<TRichTextRule> = []) {
  return rules.reduce((acc, rule) => rule.apply(acc), text)
}
```

Markdown Engine: Converting to react

```
const components = useMemo(() => {
  const xmlParser = new DOMParser()
  const rawText = cleanMarkdownFromHTML(text) // Strip real HTML tags
  const rawXML = parseRichText(rawText, RICH_TEXT_RULES) // Convert Markdown to fake HTML tags

  try {
    // Parse the fake HTML tags strictly into an interactive XML DOM tree
    const xml = xmlParser.parseFromString(`<article>${rawXML}</article>`, 'application/xml')
    return traverseXMLTree(xml)
  } catch {
    return <p>Markdown parsing failed</p>
  }
}, [text])
```

Markdown Engine: Parsing tree

```tsx
/**
 * Recursively traverses an XML tree and converts nodes to React elements.
 * @param node - The current node to process
 * @returns React element(s) representing the node and its children
 */
function traverseXMLTree(node: Node | null): React.ReactNode {
  if (node == null) return null
  // 1. Handle base text nodes
  if (node.nodeName === TRichTextXMLNodes.Text) {
    return node.textContent
  }
  // 2. Extract children recursively
  const children = Array.from(node.childNodes).map((child, index) => (
    <React.Fragment key={index}>{traverseXMLTree(child)}</React.Fragment>
  ))
  const type = node.nodeName.toLowerCase()
  // 3. Extract dynamic attributes (like src, href, class)
  const props: Record<string, any> = {}
  if (node instanceof Element) {
    for (let i = 0; i < node.attributes.length; i++) {
      const attr = node.attributes[i]
      if (attr.name === 'class') {
        props.className = attr.value
      } else {
        props[attr.name] = attr.value
      }
    }
  }
  // Handle top-level markdown wrapper
  if (type === 'article') {
    props.className = css.markdown
  }
  // Handle document root
  if (type === '#document') {
    if (node.childNodes.length > 0) return <>{children}</>
    return node.textContent
  }
  // 4. Map DOM element types to React components dynamically!
  return React.createElement(type, Object.keys(props).length ? props : null, ...children)
}
```

Markdown Engine: Execute tests

```
bun test src\problems\components\17-markdown\markdown-parser.test.ts
```

```
✓ Markdown Parser > [Reference] Implementation > parseRichText > should return empty string for empty input
✓ Markdown Parser > [Reference] Implementation > parseRichText > should return unchanged text when no rules provided
✓ Markdown Parser > [Reference] Implementation > parseRichText > should apply multiple rules in sequence
✓ Markdown Parser > [Reference] Implementation > LINK_RULE > should convert markdown link to anchor tag
✓ Markdown Parser > [Reference] Implementation > LINK_RULE > should handle multiple links in same line
✓ Markdown Parser > [Reference] Implementation > LINK_RULE > should not convert image syntax to link
✓ Markdown Parser > [Reference] Implementation > LINK_RULE > should handle links with complex URLs
✓ Markdown Parser > [Reference] Implementation > HEADER_RULE > should convert # to h1
✓ Markdown Parser > [Reference] Implementation > HEADER_RULE > should convert ## to h2
✓ Markdown Parser > [Reference] Implementation > HEADER_RULE > should handle h3 and h4
✓ Markdown Parser > [Reference] Implementation > HEADER_RULE > should handle header without space after #
✓ Markdown Parser > [Reference] Implementation > HEADER_RULE > should handle multiple headers
✓ Markdown Parser > [Reference] Implementation > TABLE_RULE > should convert simple markdown table to HTML
✓ Markdown Parser > [Reference] Implementation > TABLE_RULE > should handle table with multiple rows
✓ Markdown Parser > [Reference] Implementation > TABLE_RULE > should handle table with three columns
✓ Markdown Parser > [Reference] Implementation > LIST_RULE - Unordered Lists > should convert single unordered list item
✓ Markdown Parser > [Reference] Implementation > LIST_RULE - Unordered Lists > should convert multiple unordered list items
✓ Markdown Parser > [Reference] Implementation > LIST_RULE - Unordered Lists > should support + as list marker
✓ Markdown Parser > [Reference] Implementation > LIST_RULE - Ordered Lists > should convert single ordered list item
✓ Markdown Parser > [Reference] Implementation > LIST_RULE - Ordered Lists > should convert multiple ordered list items
✓ Markdown Parser > [Reference] Implementation > LIST_RULE - Ordered Lists > should handle double-digit numbers
✓ Markdown Parser > [Reference] Implementation > BOLD_RULE > should convert **text** to bold
✓ Markdown Parser > [Reference] Implementation > BOLD_RULE > should handle multiple bold segments
✓ Markdown Parser > [Reference] Implementation > BOLD_RULE > should handle bold with special characters
✓ Markdown Parser > [Reference] Implementation > ITALIC_RULE > should convert *text* to italic
✓ Markdown Parser > [Reference] Implementation > ITALIC_RULE > should handle multiple italic segments
✓ Markdown Parser > [Reference] Implementation > STRIKETHROUGH_RULE > should convert ~~text~~ to strikethrough
✓ Markdown Parser > [Reference] Implementation > STRIKETHROUGH_RULE > should handle strikethrough with space after tildes
✓ Markdown Parser > [Reference] Implementation > STRIKETHROUGH_RULE > should handle multiple strikethrough segments
✓ Markdown Parser > [Reference] Implementation > PARAGRAPH_RULE > should wrap plain text in paragraph tags
✓ Markdown Parser > [Reference] Implementation > PARAGRAPH_RULE > should not double-wrap existing HTML elements
✓ Markdown Parser > [Reference] Implementation > PARAGRAPH_RULE > should handle multiple paragraphs
✓ Markdown Parser > [Reference] Implementation > RICH_TEXT_RULES Integration > should handle complex markdown document
✓ Markdown Parser > [Reference] Implementation > RICH_TEXT_RULES Integration > should handle document with table and formatting
```

Markdown Engine: Result

Markdown Preview:

Welcome to Markdown Parser

Text Formatting

This is a paragraph with **bold text**, *italic text*, and ~~strikethrough text~~.

You can also combine them: ***bold and italic*** or **~~bold strikethrough~~**.

Lists

Unordered List

- First item
- Second item
- Third item

Ordered List

1. Step one
2. Step two
3. Step three

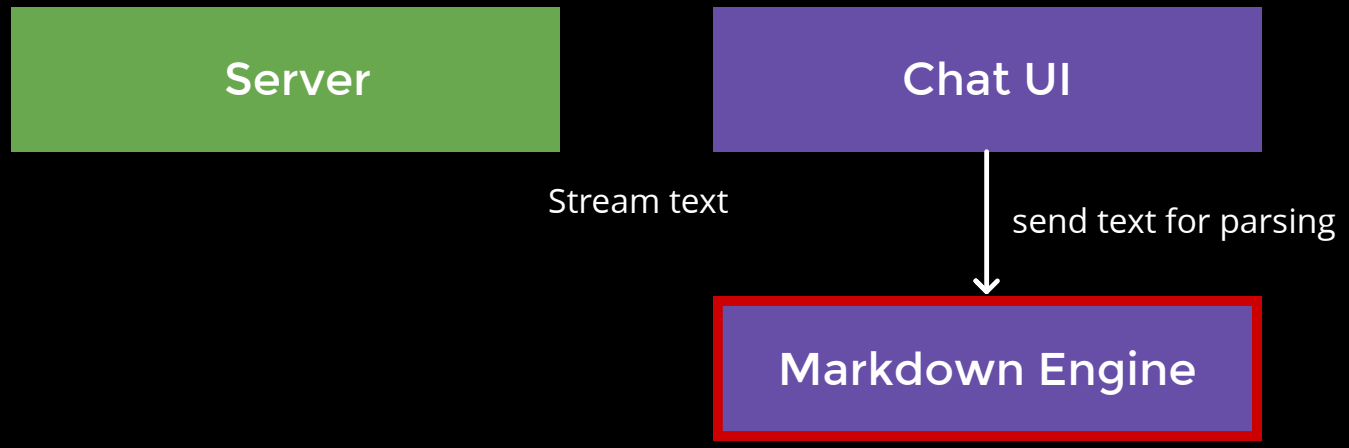
Links

Check out Google or GitHub.

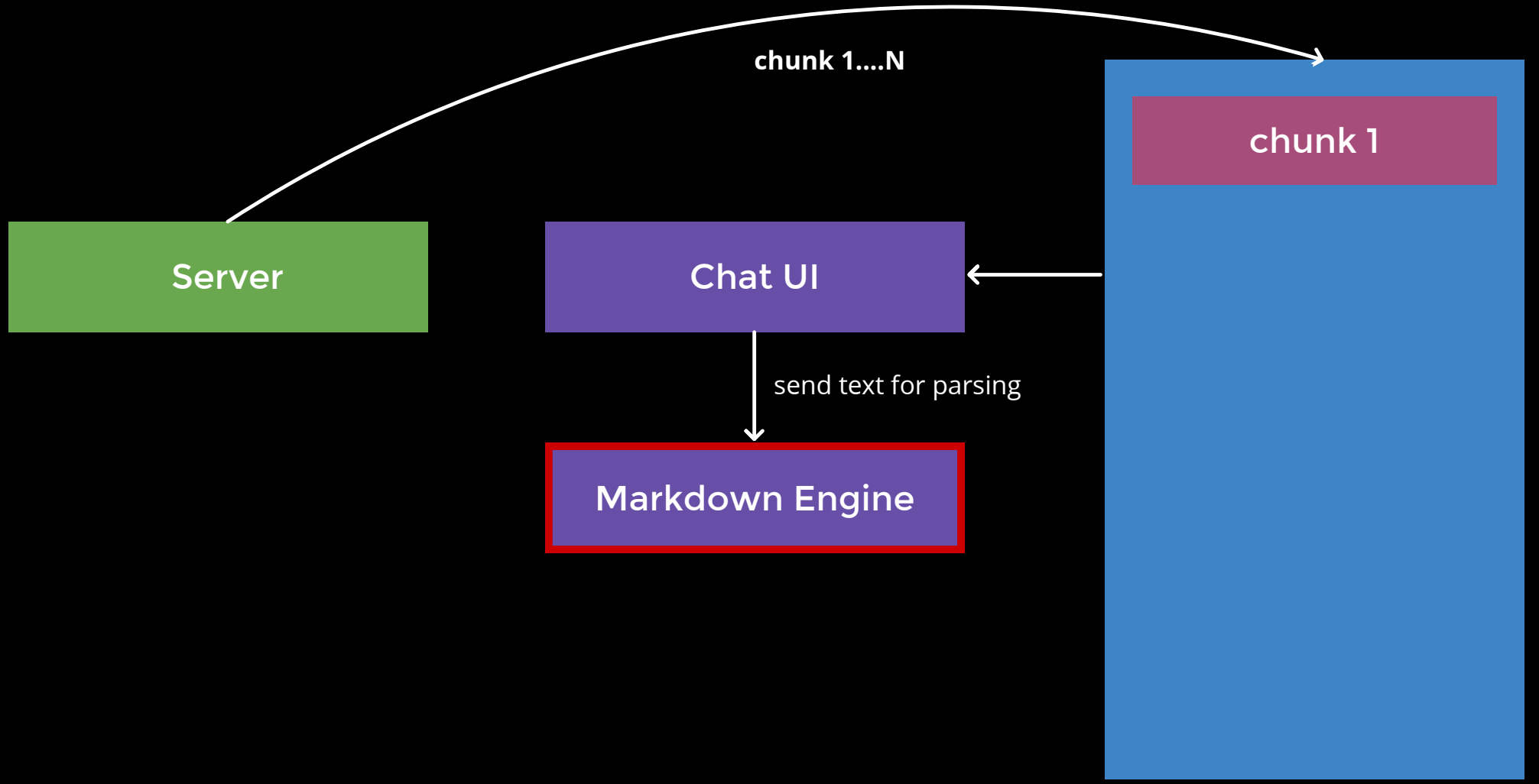
Tables

Name	Age	City
Alice	25	NYC

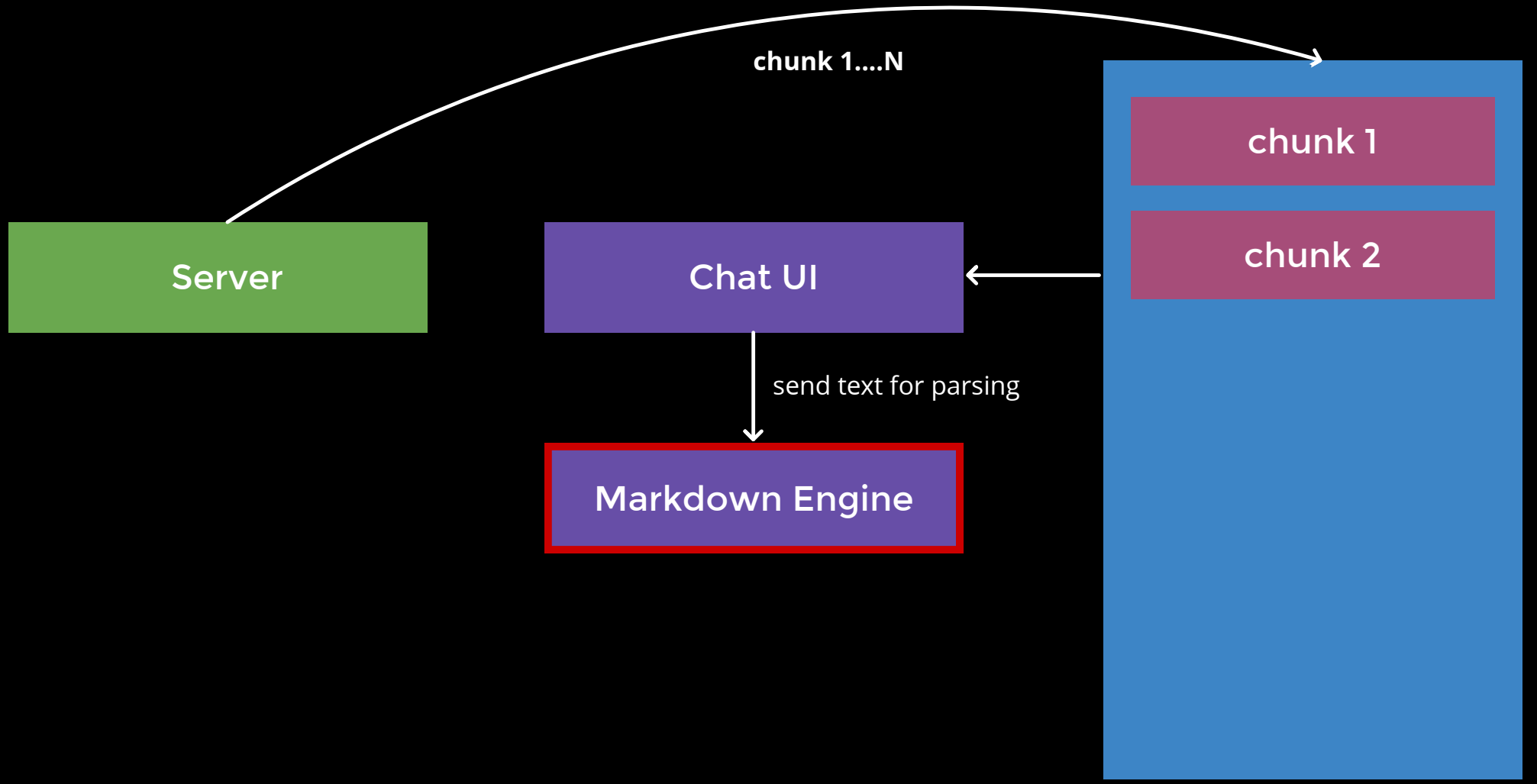
ChatGPT: Markdown Engine integration



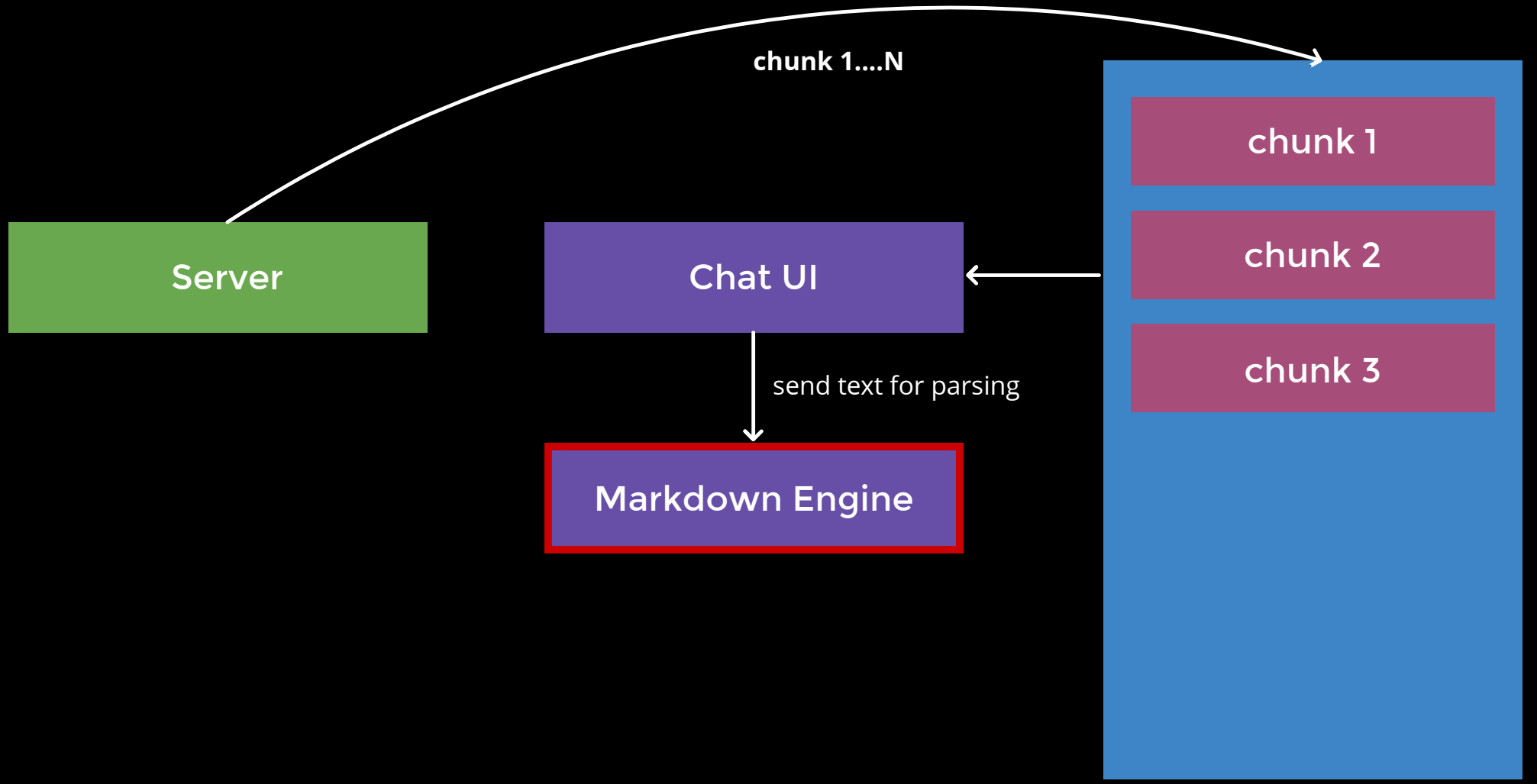
ChatGPT: Markdown Engine integration



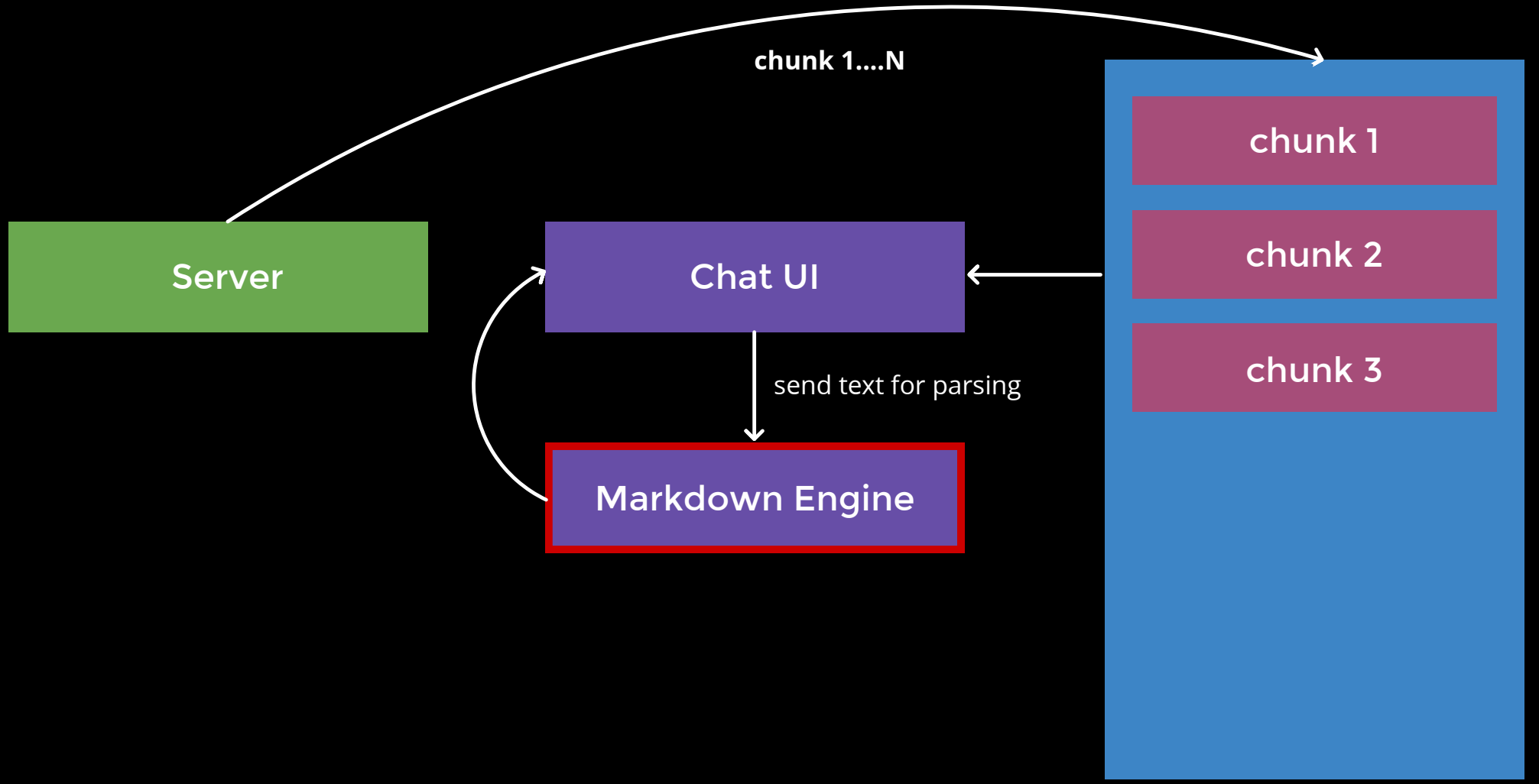
ChatGPT: Markdown Engine integration



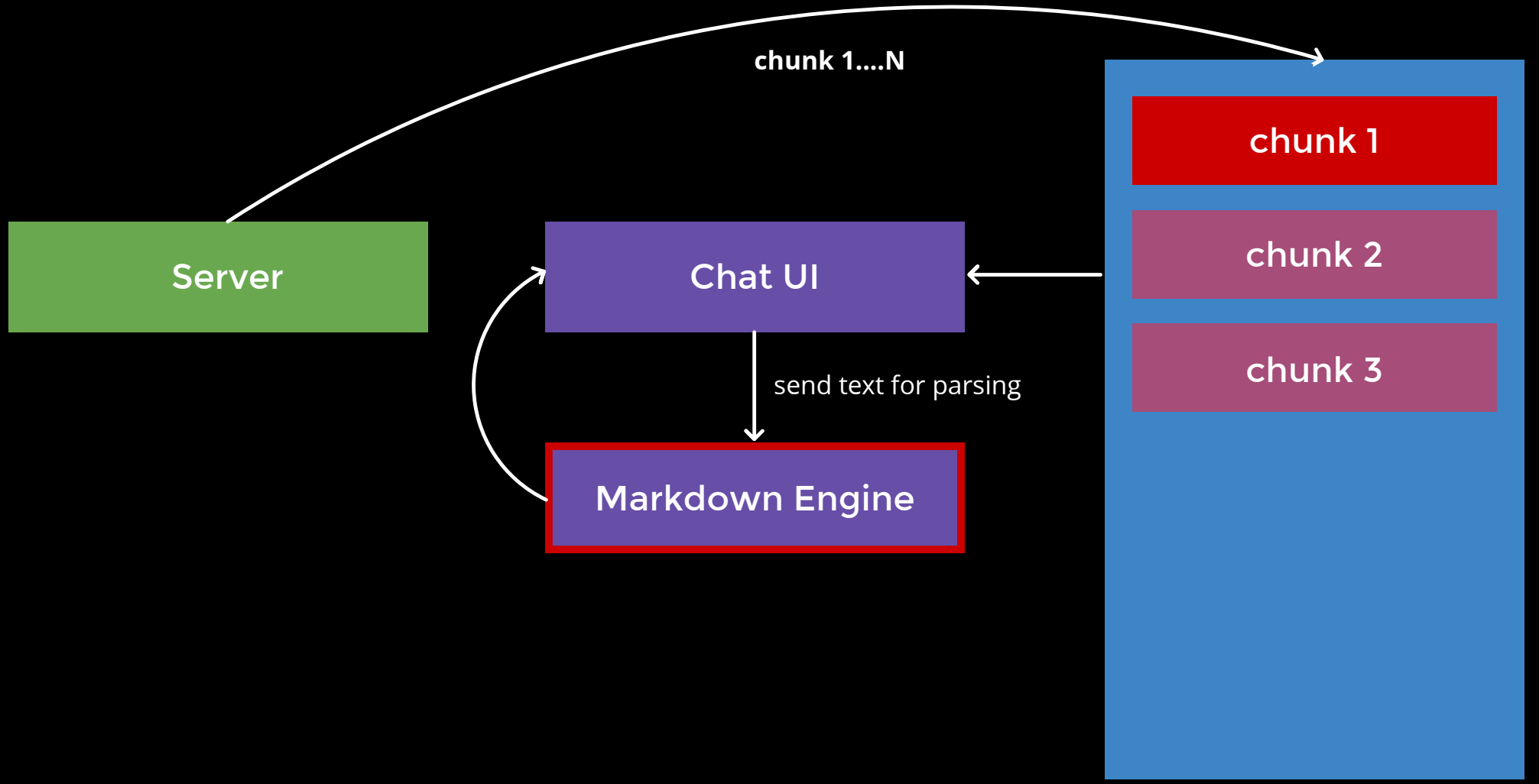
ChatGPT: Markdown Engine integration



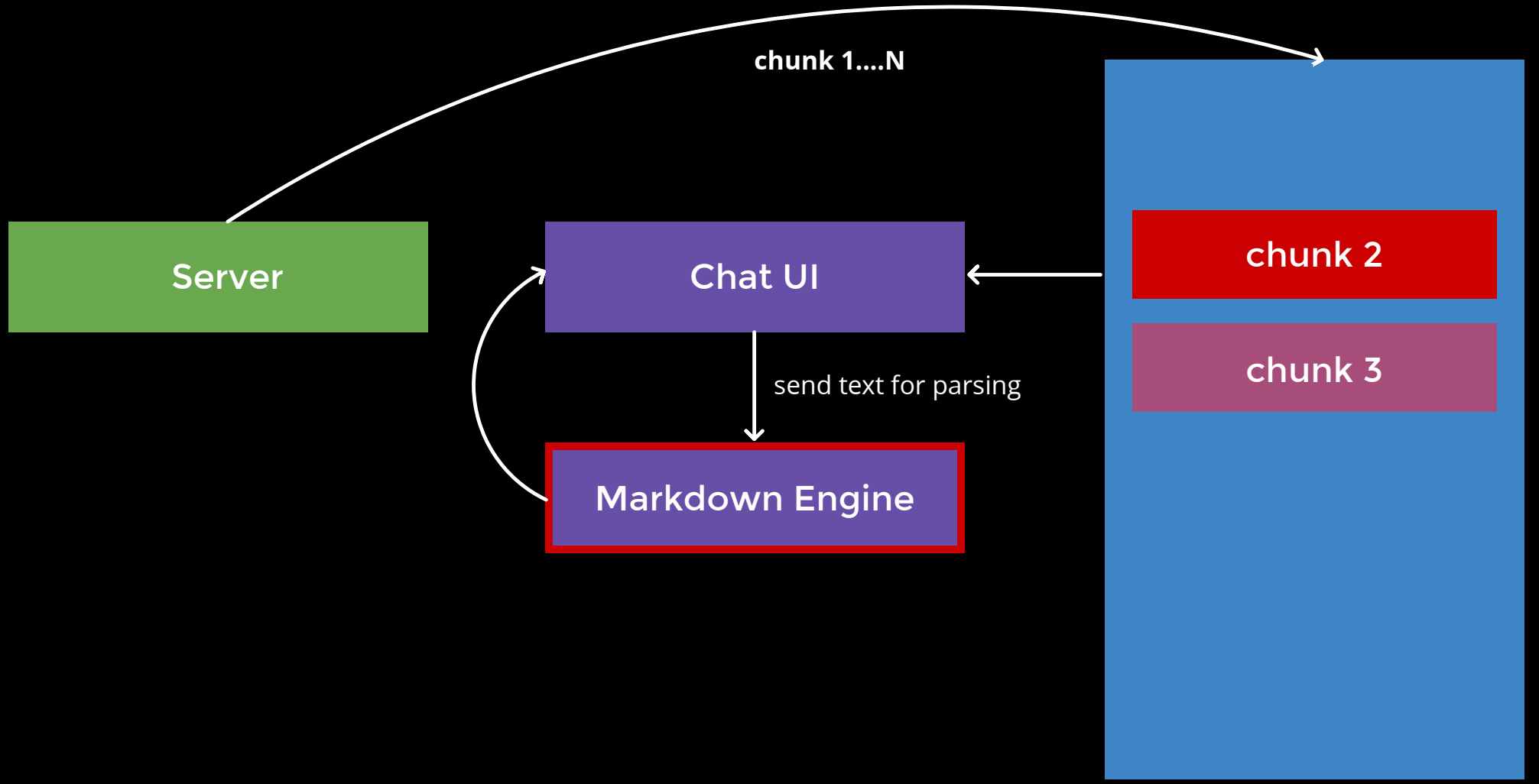
ChatGPT: Markdown Engine integration



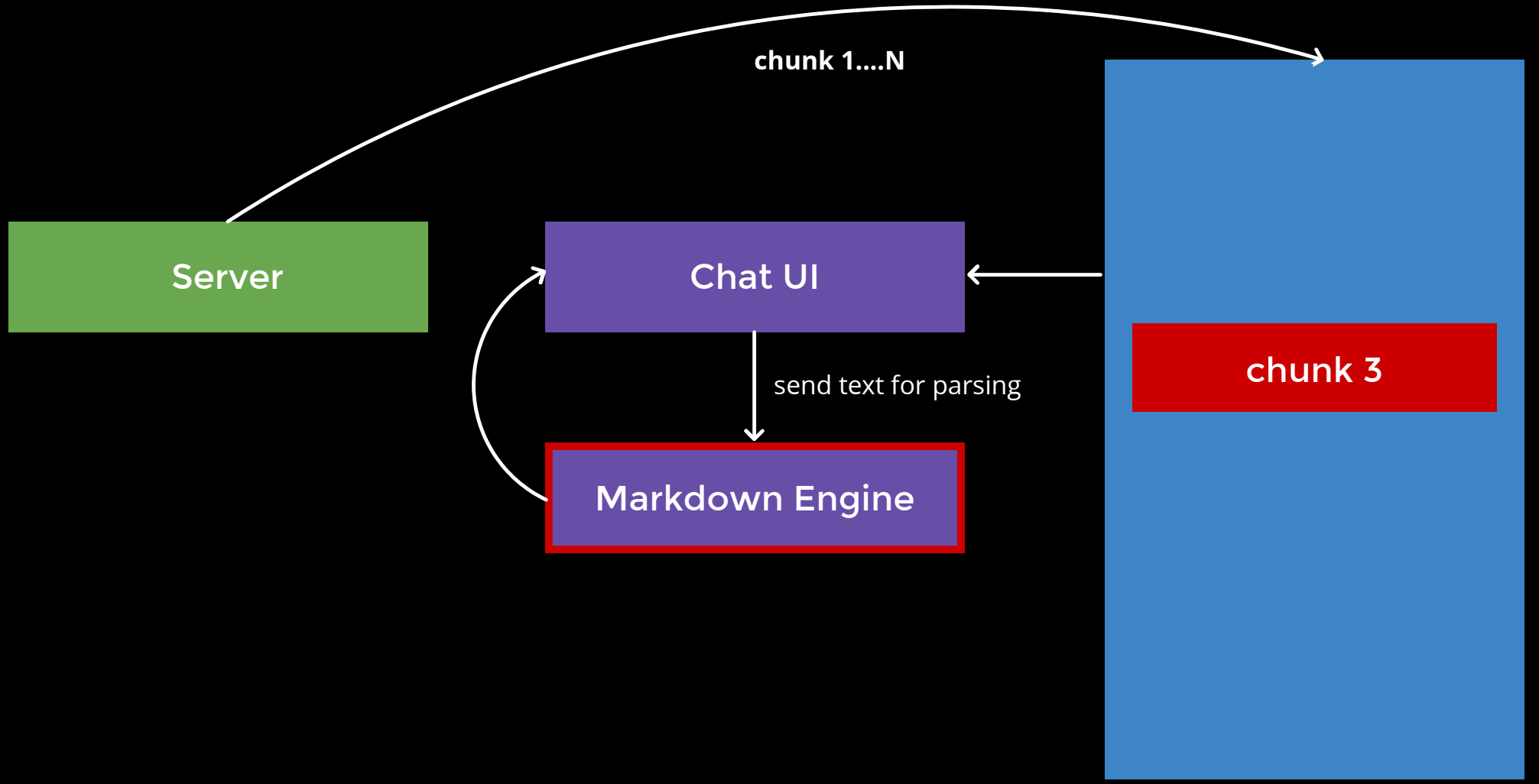
ChatGPT: Markdown Engine integration



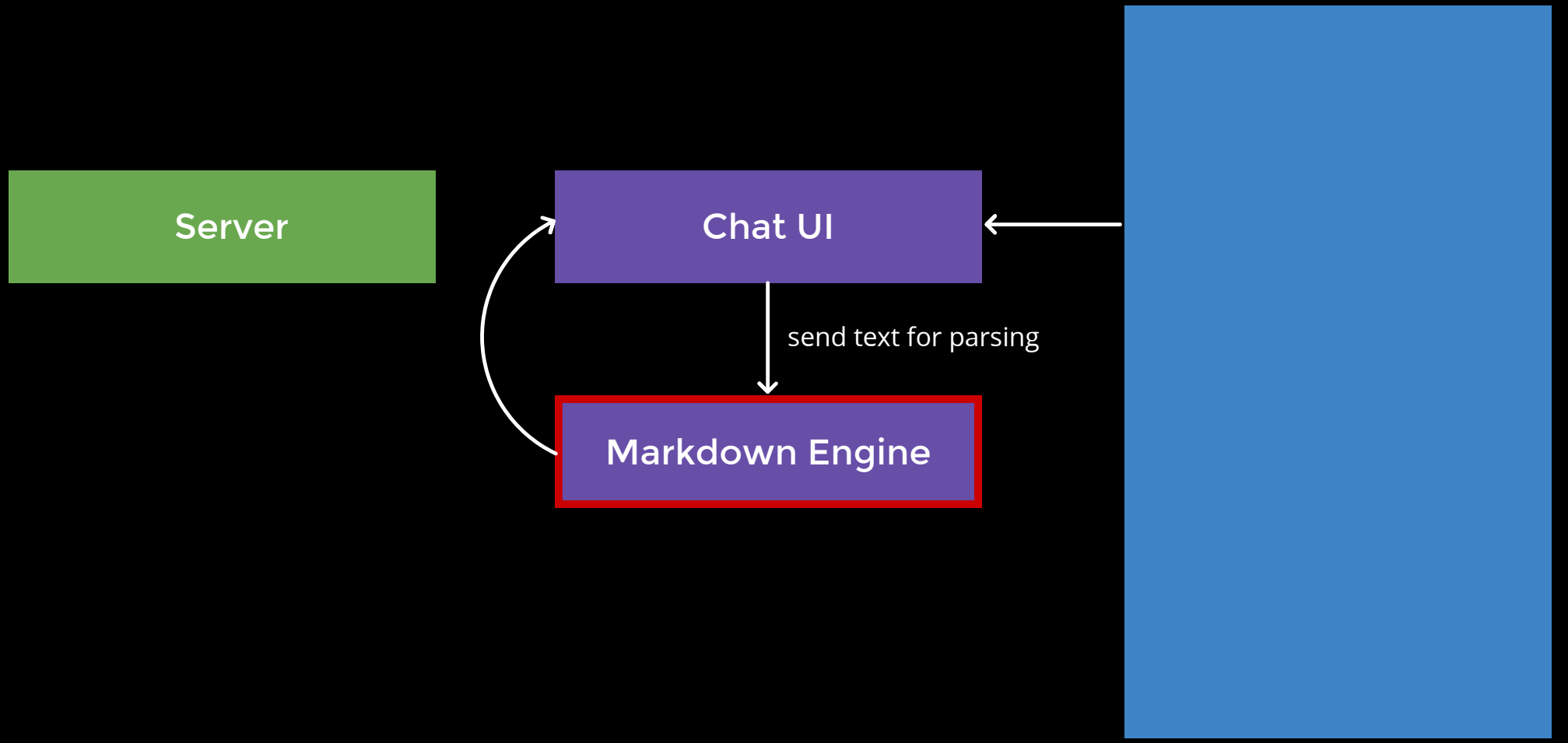
ChatGPT: Markdown Engine integration



ChatGPT: Markdown Engine integration



ChatGPT: Markdown Engine integration



ChatGPT: Provided API

```
type StreamCallback = (chunk: string) => void

type StreamOptions = {
    onComplete?: () => void
    onError?: (error: Error) => void
    delay?: number
}

/**
 * Hook for streaming markdown content from the server.
 * @param options - Optional configuration for completion/error callbacks and delay
 * @returns Object with stream, abort functions and inProgress state
 */
export function useMarkdownStream(options: StreamOptions = {}) {}
```

ChatGPT: Sending a request

```
const [chunks, setChunks] = useState<string[]>([])

const handleSend = () => {
  setChunks([]) // Clear previous conversation buffer
  stream((newChunk) => {
    setChunks((prev) => [...prev, newChunk])
  })
}
```

ChatGPT: Consume Chunk

```
1   const type = useCallback(function recursiveType(chunk: string) {
2     if (chunk.length === 0) {
3       setIsTyping(false)
4       return
5     }
6     const charsToType = 2
7     const chars = chunk.slice(0, charsToType)
8     const rest = chunk.slice(charsToType)
9     setContent((prev) => prev + chars)
10    if (rest.length > 0) {
11      requestAnimationFrame(() => recursiveType(rest))
12    } else {
13      setIsTyping(false)
14    }
15    contentRef.current?.scrollTo({
16      top: contentRef.current.scrollHeight,
17      behavior: 'smooth',
18    })
19  }, [])
20
21  useEffect(() => {
22    if (!isTyping && chunks.length > 0) {
23      setIsTyping(true)
24      const [nextChunk, ...rest] = chunks
25      setChunks(rest)
26      type(nextChunk)
27    }
28  }, [chunks, isTyping, type])
```

ChatGPT: JSX

```
return (
  <div className={cx(css.container, flex.w100)}>
    <section className={css.content} ref={contentRef}>
      <Markdown text={content} />
    </section>
    <section className={cx(flex.flexRowCenter, flex.flexGap32)}>
      <textarea className={cx(css.textarea, flex.w100, flex.maxW500px)}></textarea>
      {inProgress ? (
        <button className={css.button} onClick={abort}>
          Stop
        </button>
      ) : (
        <button className={css.button} onClick={handleSend}>
          Send
        </button>
      )}
    </section>
  </div>
)
```

ChatGPT: Result

Tips for Success

For optimal results:

1. Keep requests focused on single topics
2. Use the stop button if needed
3. Review the response as it appears
4. Experiment with different queries

Additional Information

The system supports various formatting options. You can use **bold text** in the middle of a sentence, or *italicize* important terms.

Lists can be nested and combined with other elements for clear organization of information.

Check out our documentation to learn more.

Thank you for using **GPT Chat!** This demonstrates our streaming markdown capabilities with a longer response.

Feel free to experiment with the *typing animation* and observe how the content appears in real-time. The streaming effect creates an

121212

Send

Final BOSS problem

Problem 19: Google Sheet

Your task is to implement a simplified spreadsheet application inspired by Google Sheets. The goal is to design a 2D grid system with basic formula support and cell references.

1 Grid Constraints

1. **Columns:** Limited to A–Z (26 columns total)
2. **Rows**: Limited to 1–500
3. **Each cell is uniquely identified using spreadsheet notation:** A1, B25, Z500

2 Supported Formula Features:

1. Addition: +
2. Subtraction: -
3. Multiplication: *
4. Division: /
5. Parentheses: (and) for precedence control
6. Cell References: =A1 + B1

3 Cell Content

1. Numeric values.
2. String values
3. Example: 42, 3.14
4. Formulas: =A1 + B1

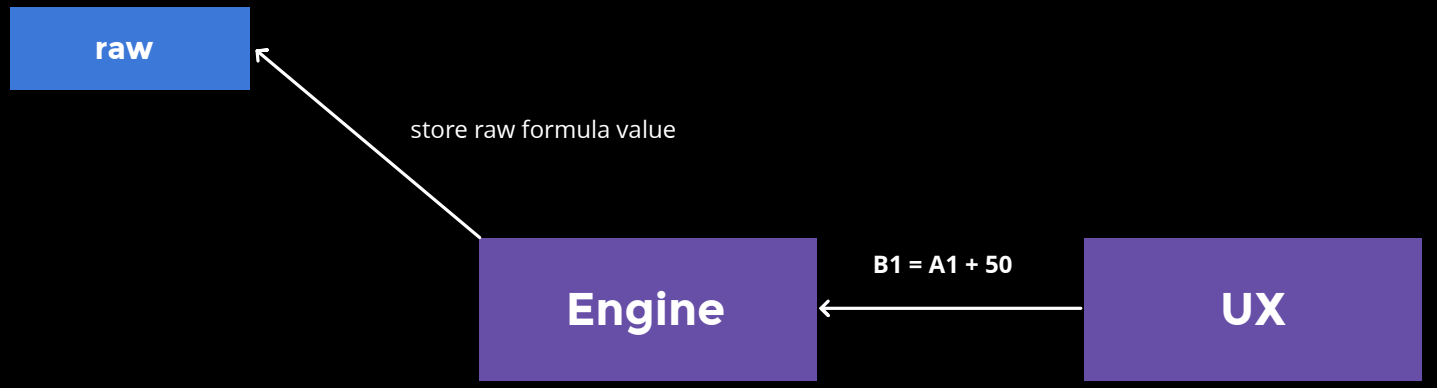
4 Tools provided

1. Formula AST Builder (tokenizer)
2. RPN Converter + Evaluator

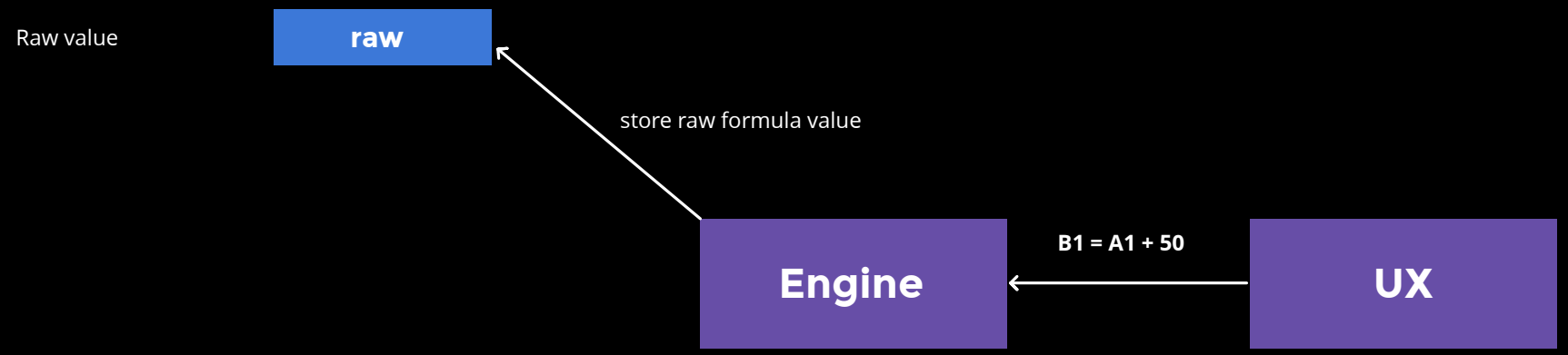
Level: EXTREME

Google Sheet - Decompousing a problem

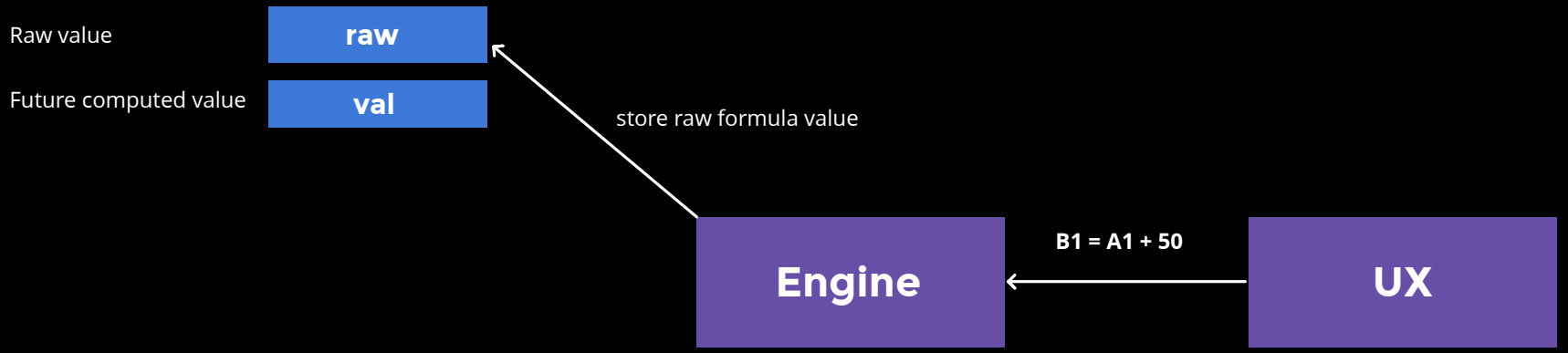
Google Sheet - Decompousing a problem



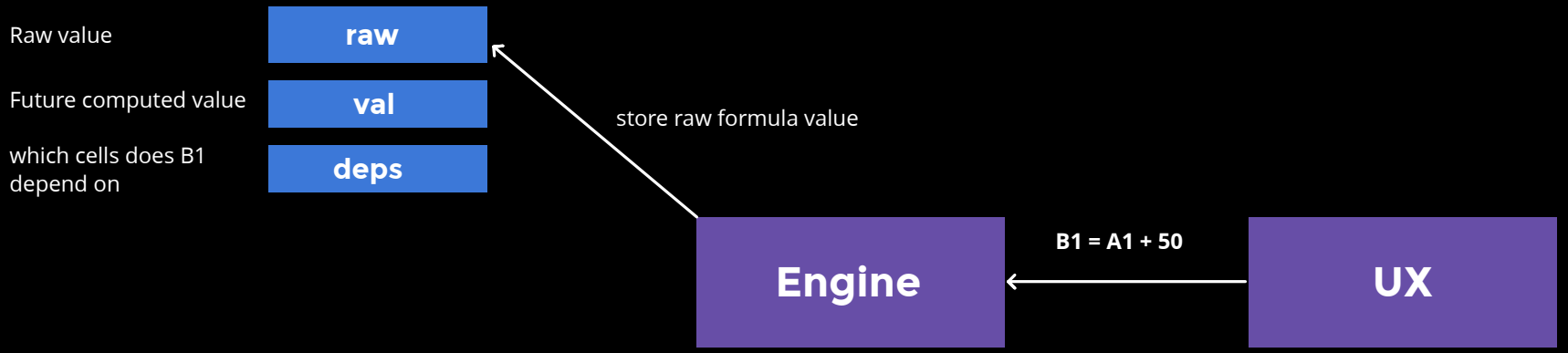
Google Sheet - Decompousing a problem



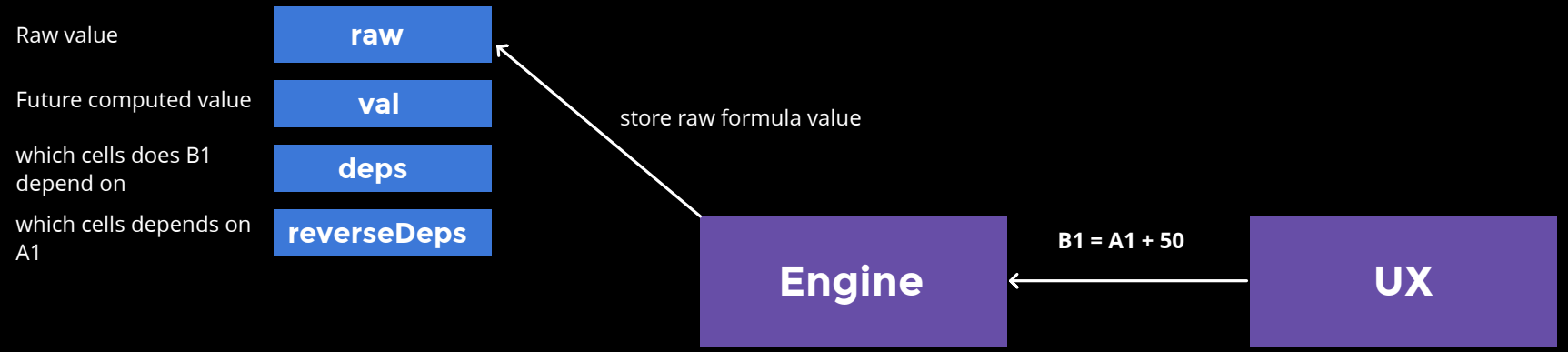
Google Sheet - Decompousing a problem



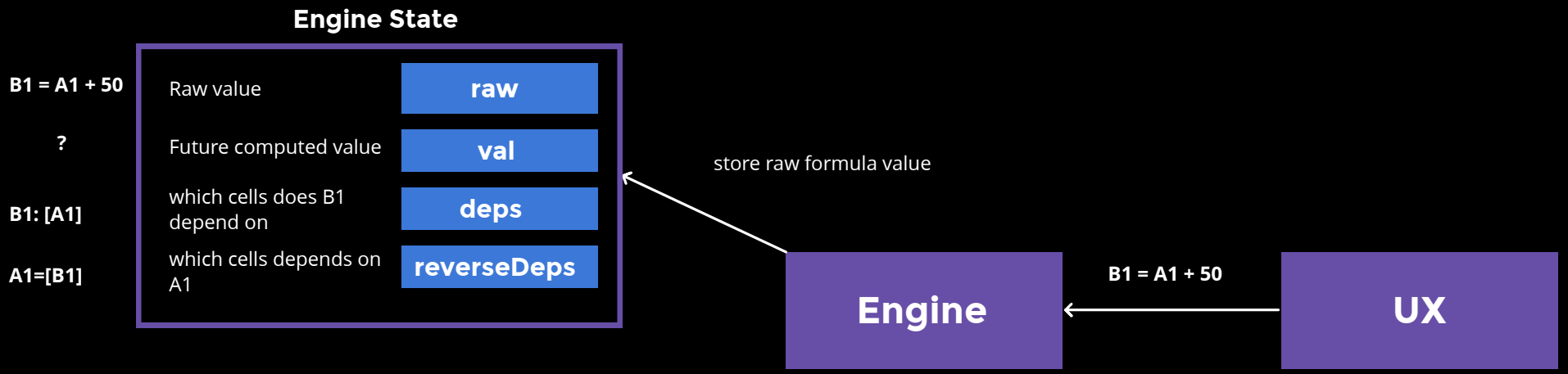
Google Sheet - Decompousing a problem



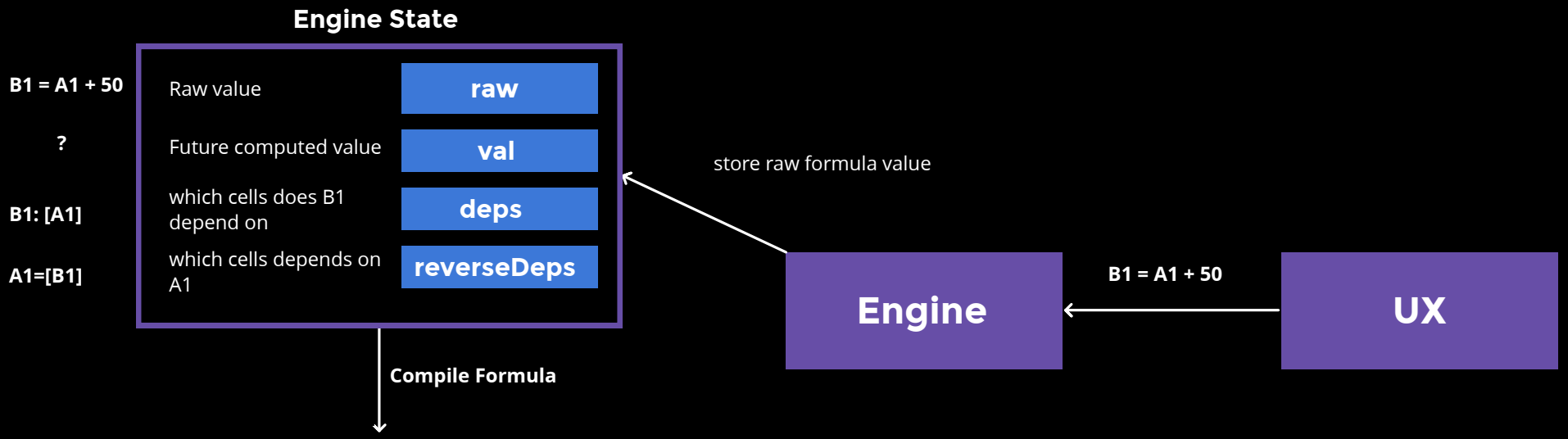
Google Sheet - Decompousing a problem



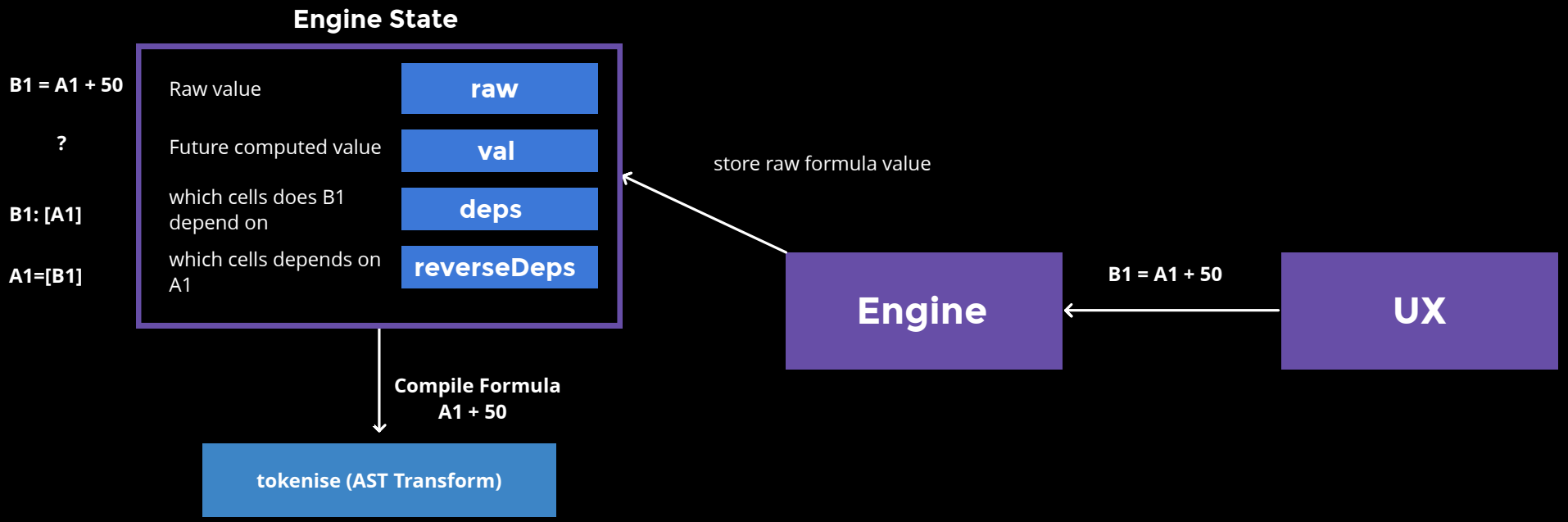
Google Sheet - Decompousing a problem



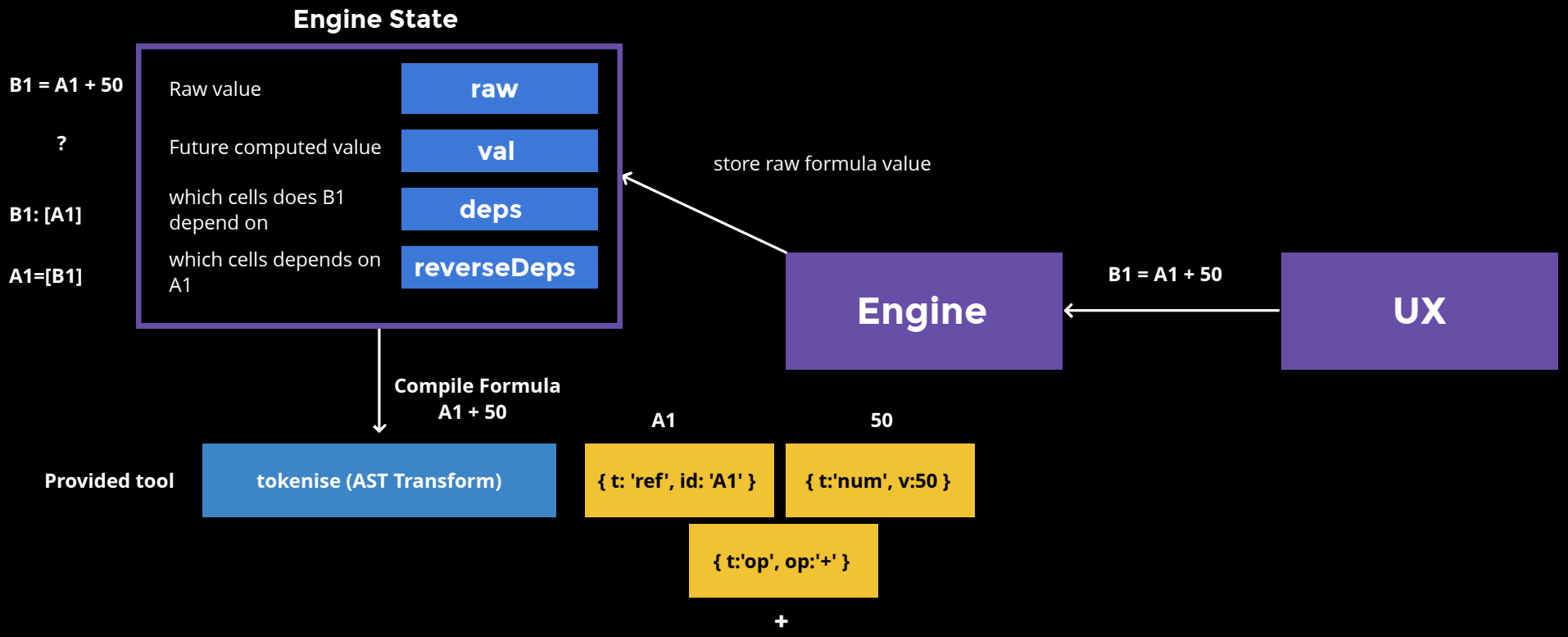
Google Sheet - Decompousing a problem



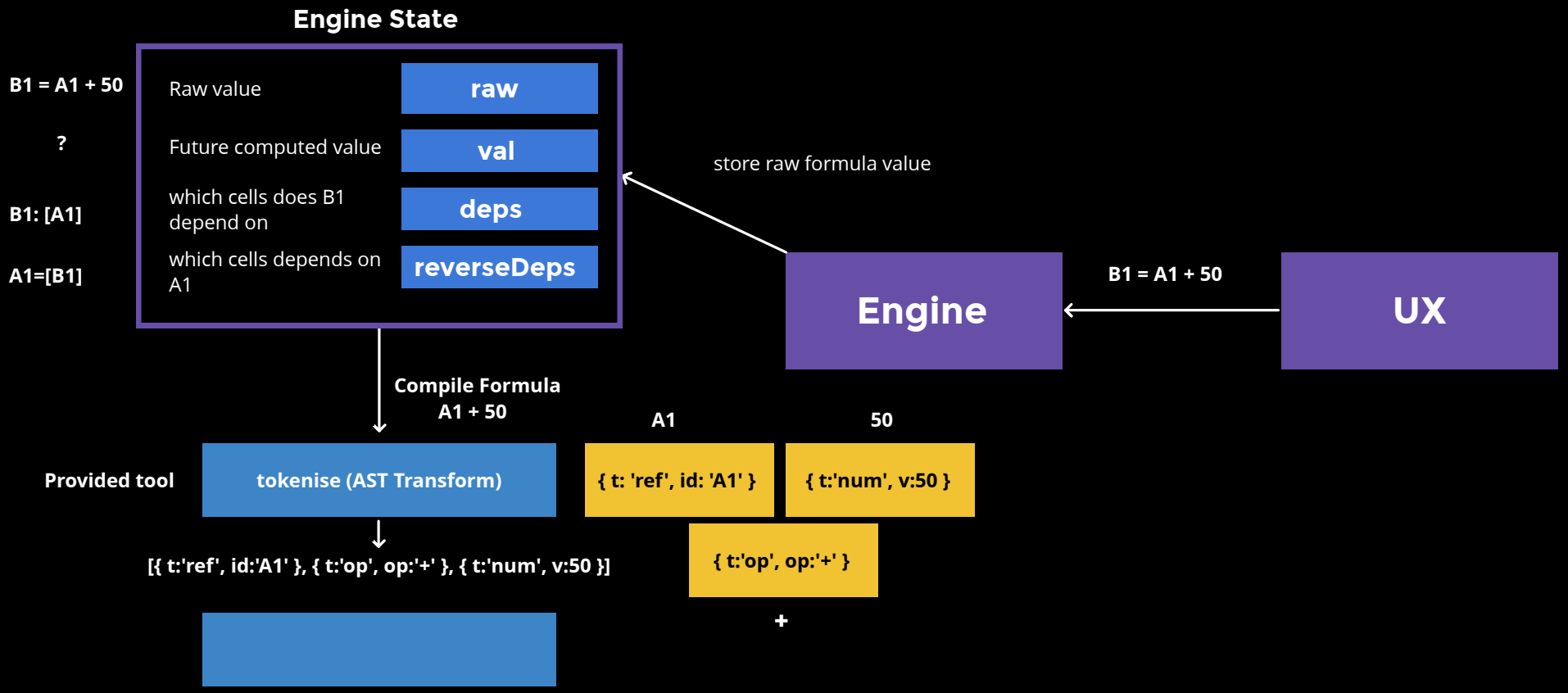
Google Sheet - Decompousing a problem



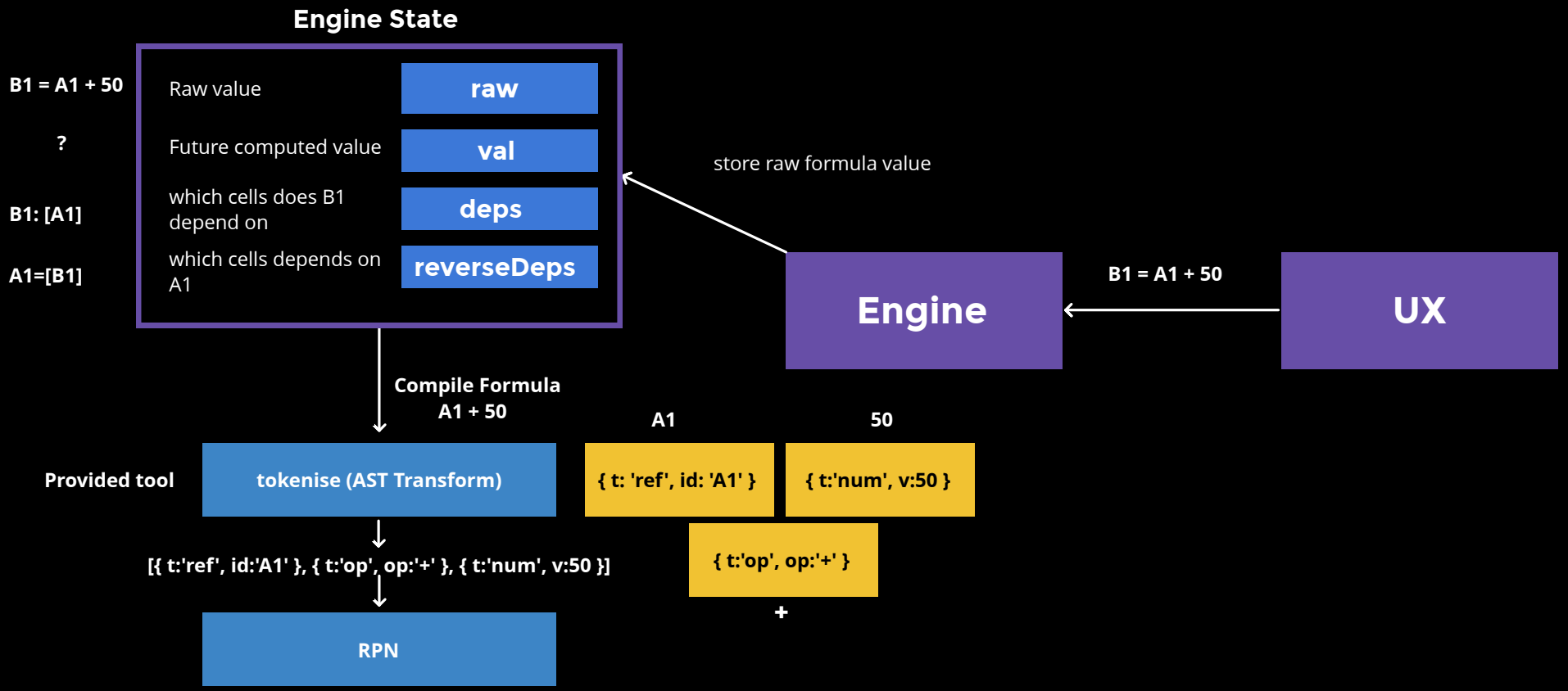
Google Sheet - Decompousing a problem



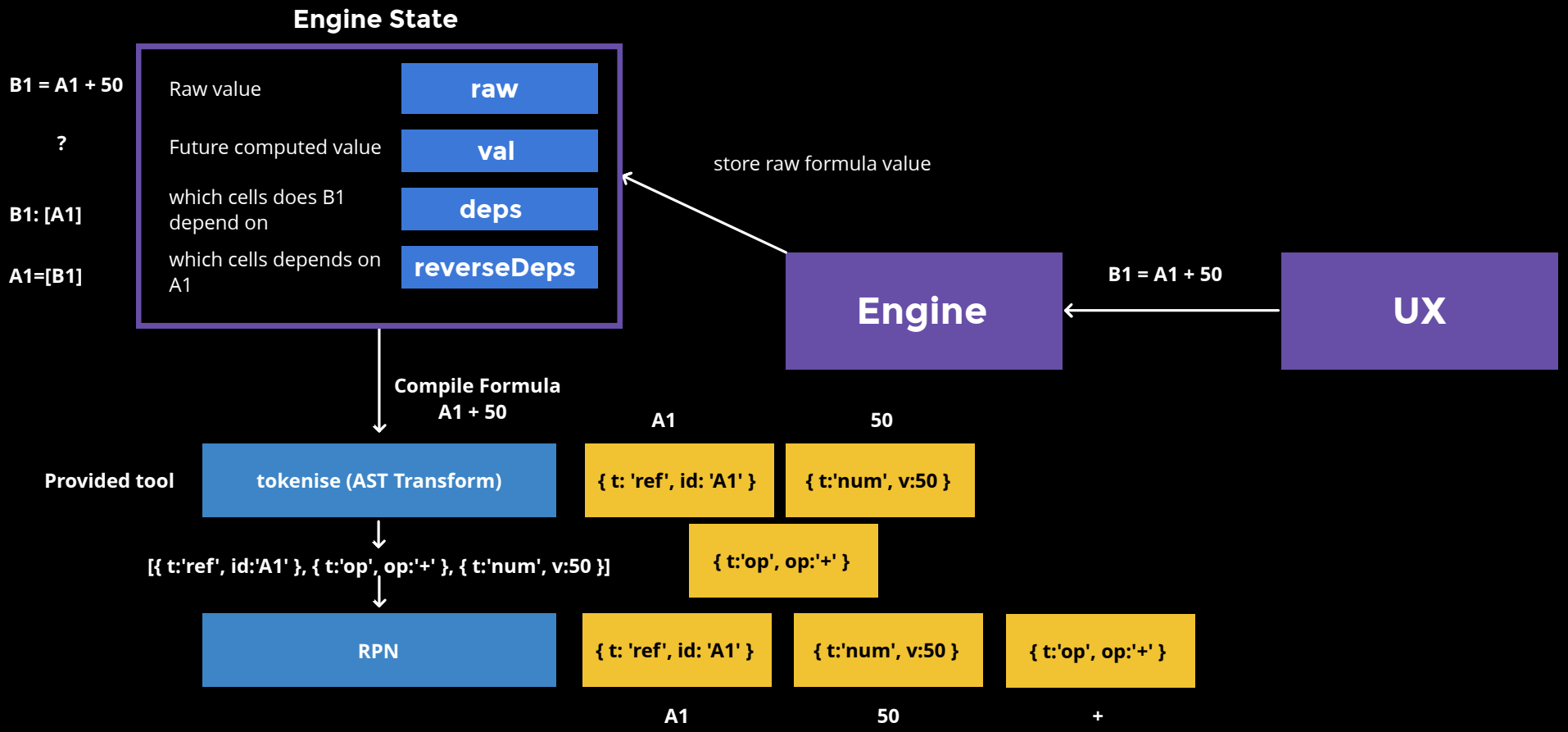
Google Sheet - Decompousing a problem



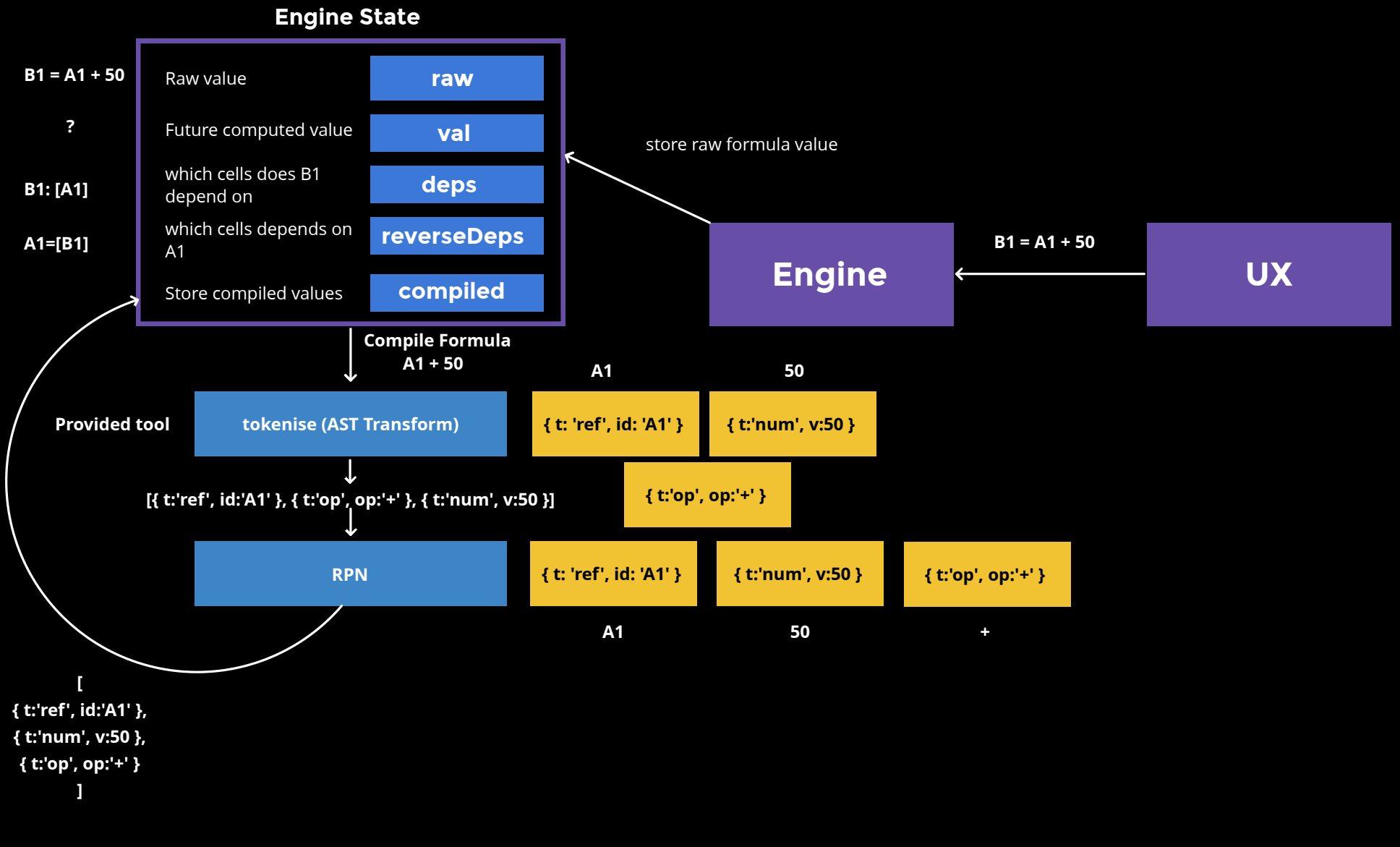
Google Sheet - Decompousing a problem



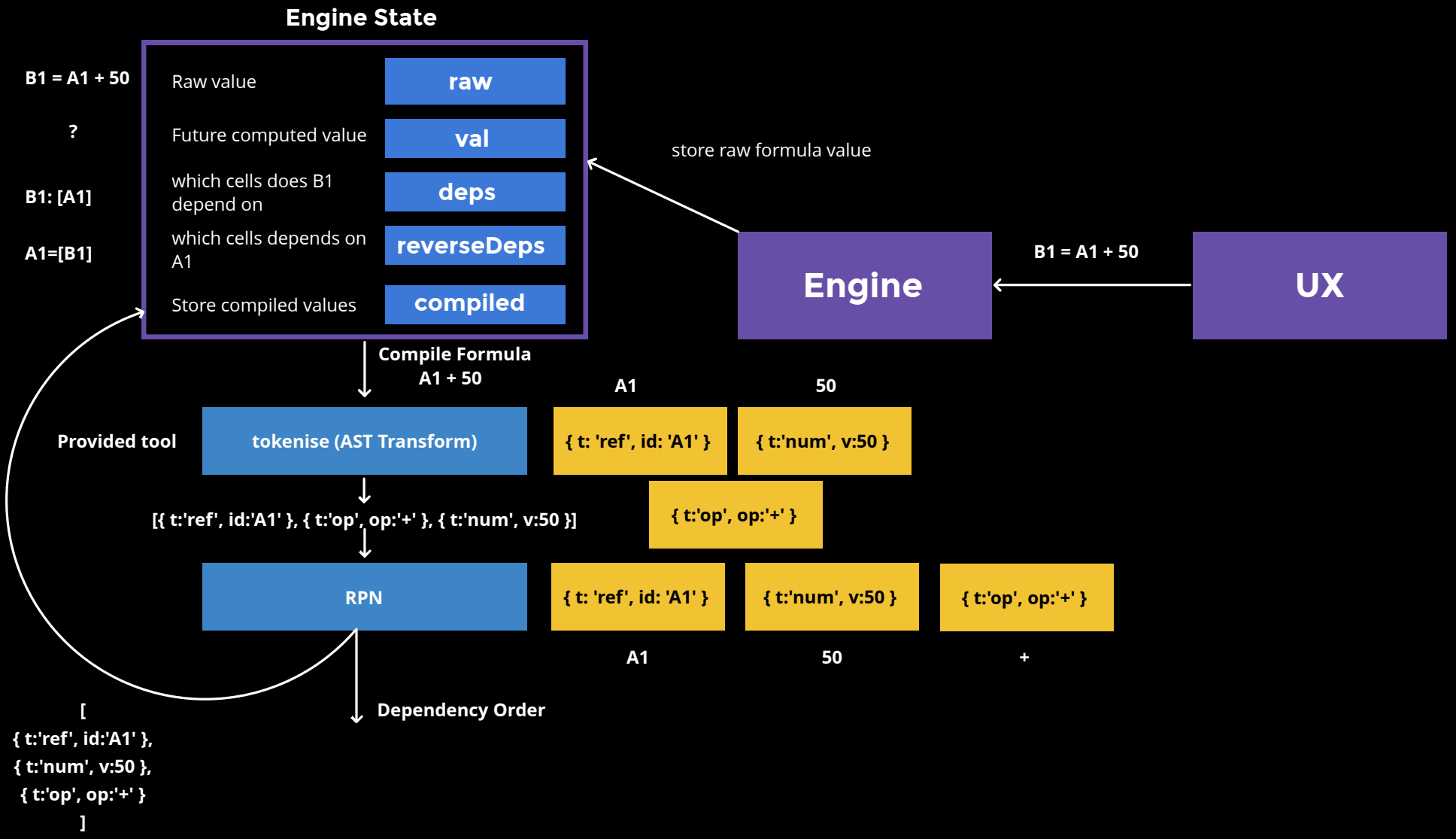
Google Sheet - Decompousing a problem



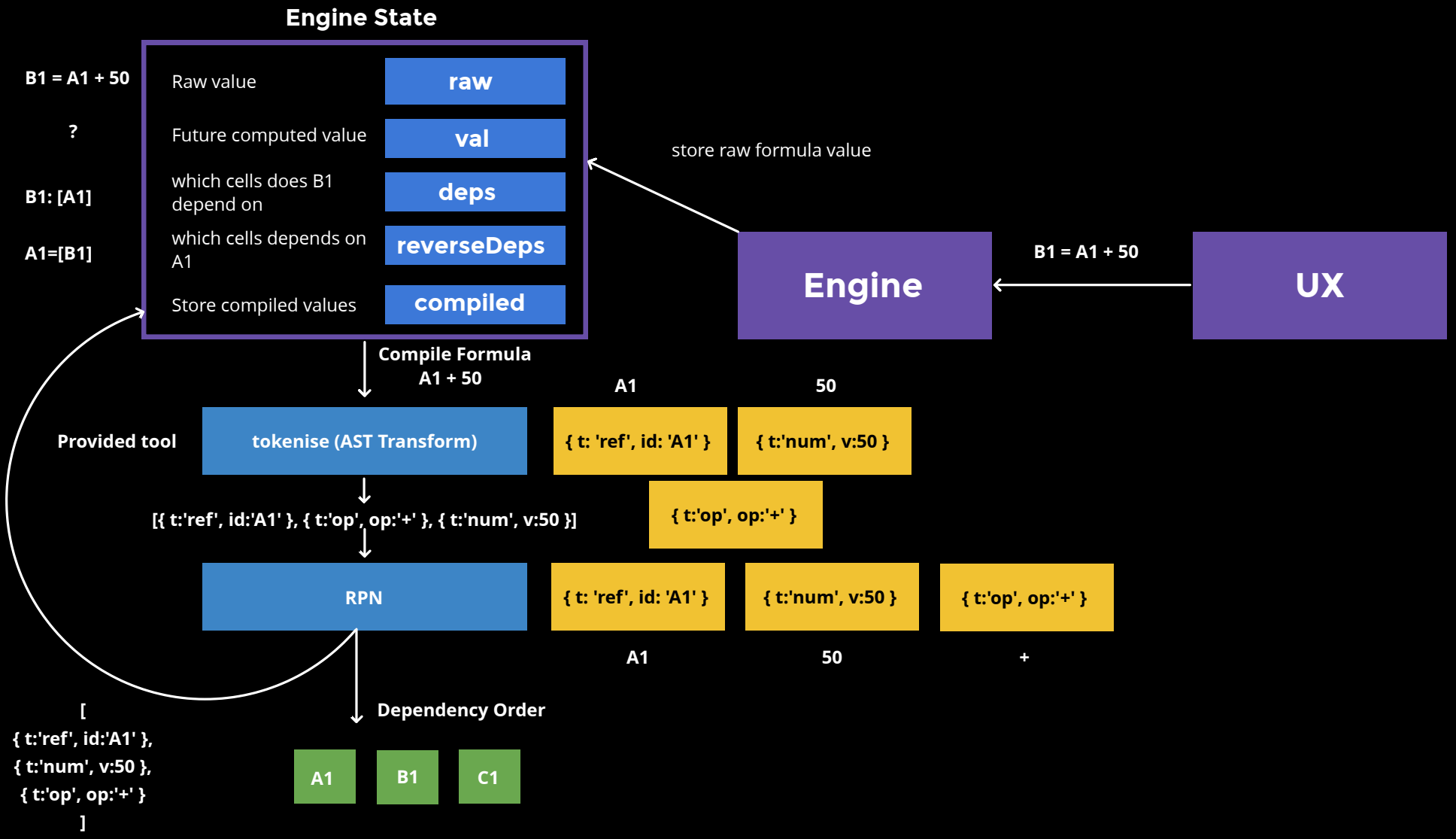
Google Sheet - Decompousing a problem



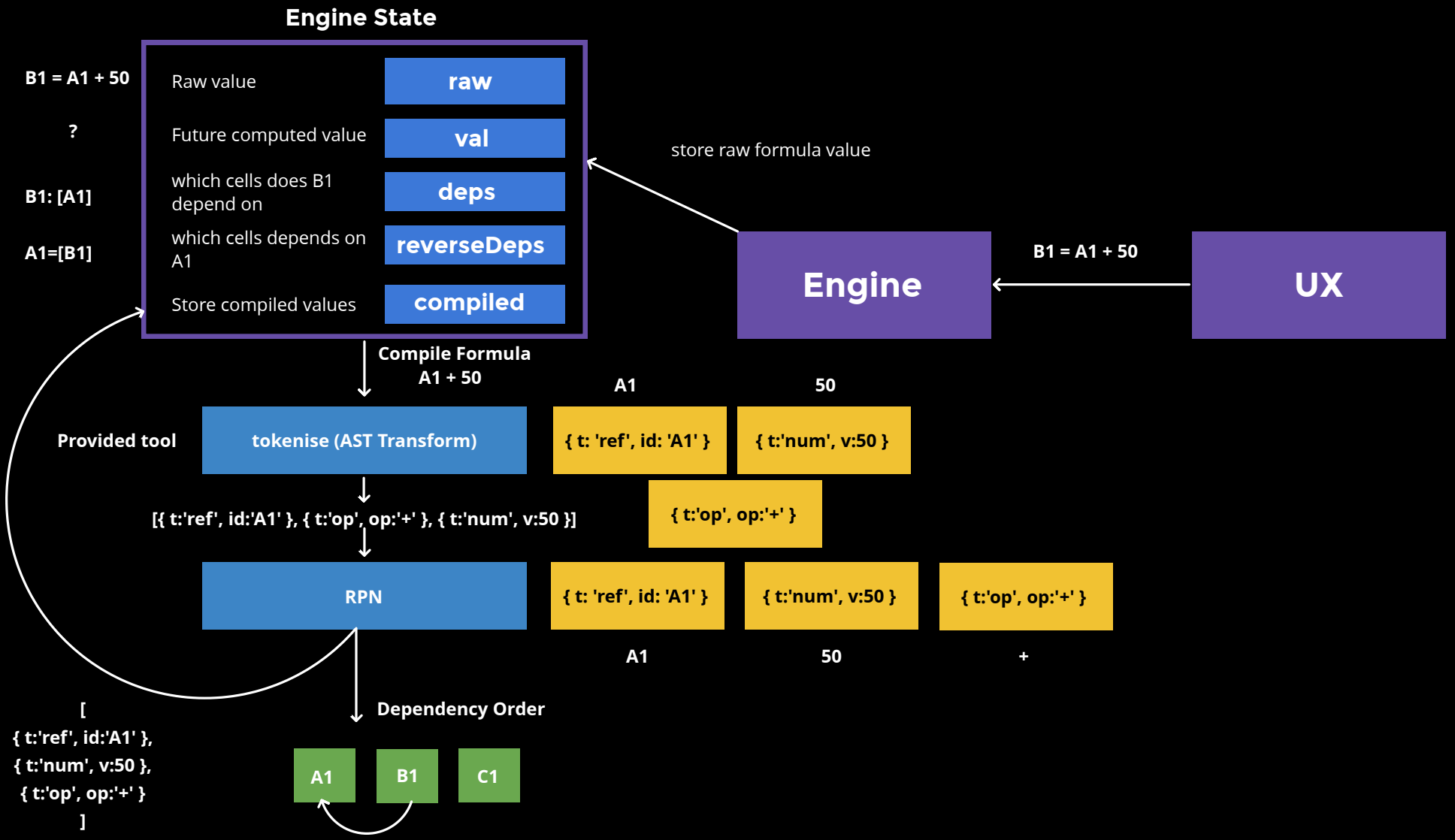
Google Sheet - Decompousing a problem



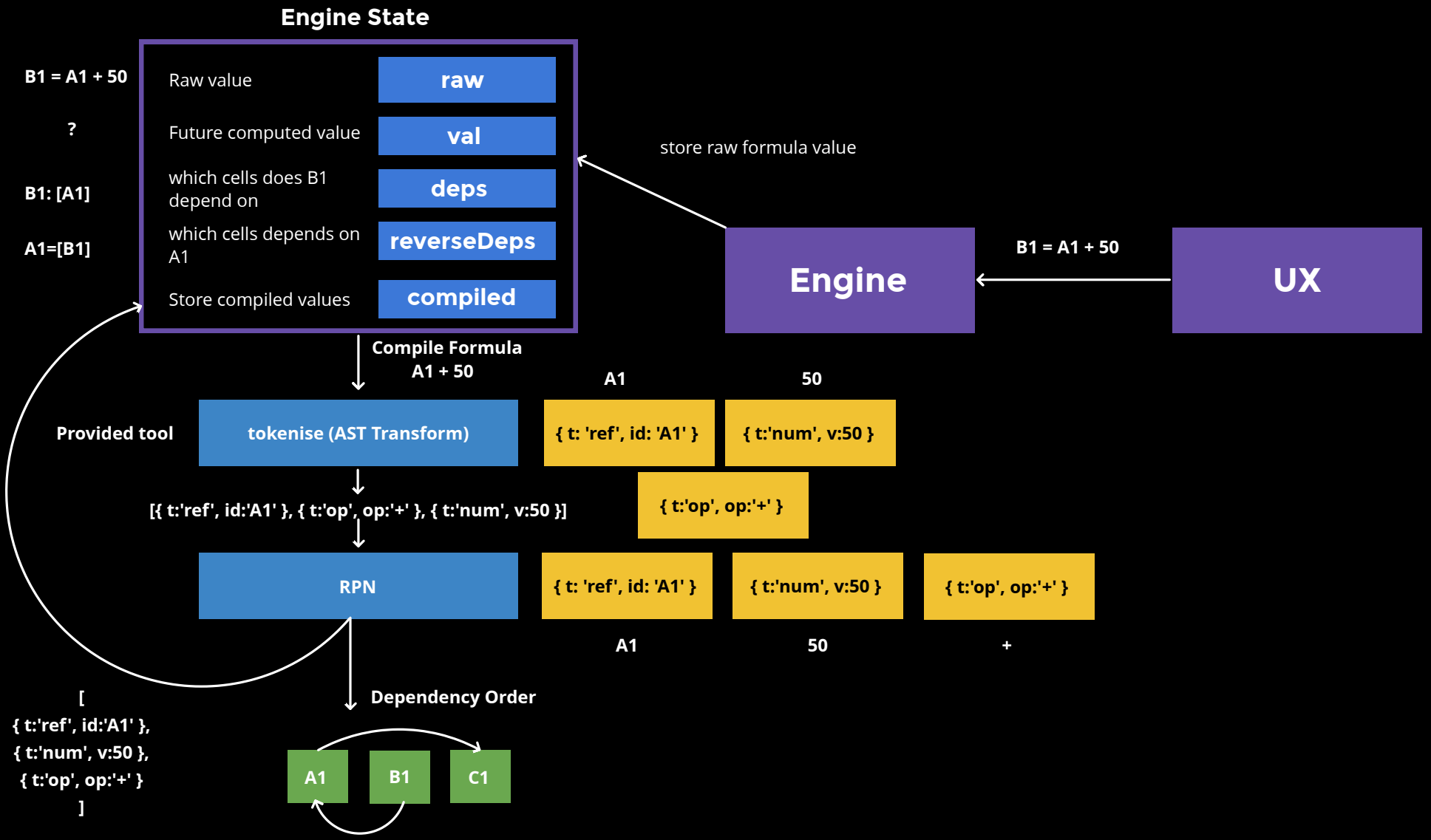
Google Sheet - Decompousing a problem



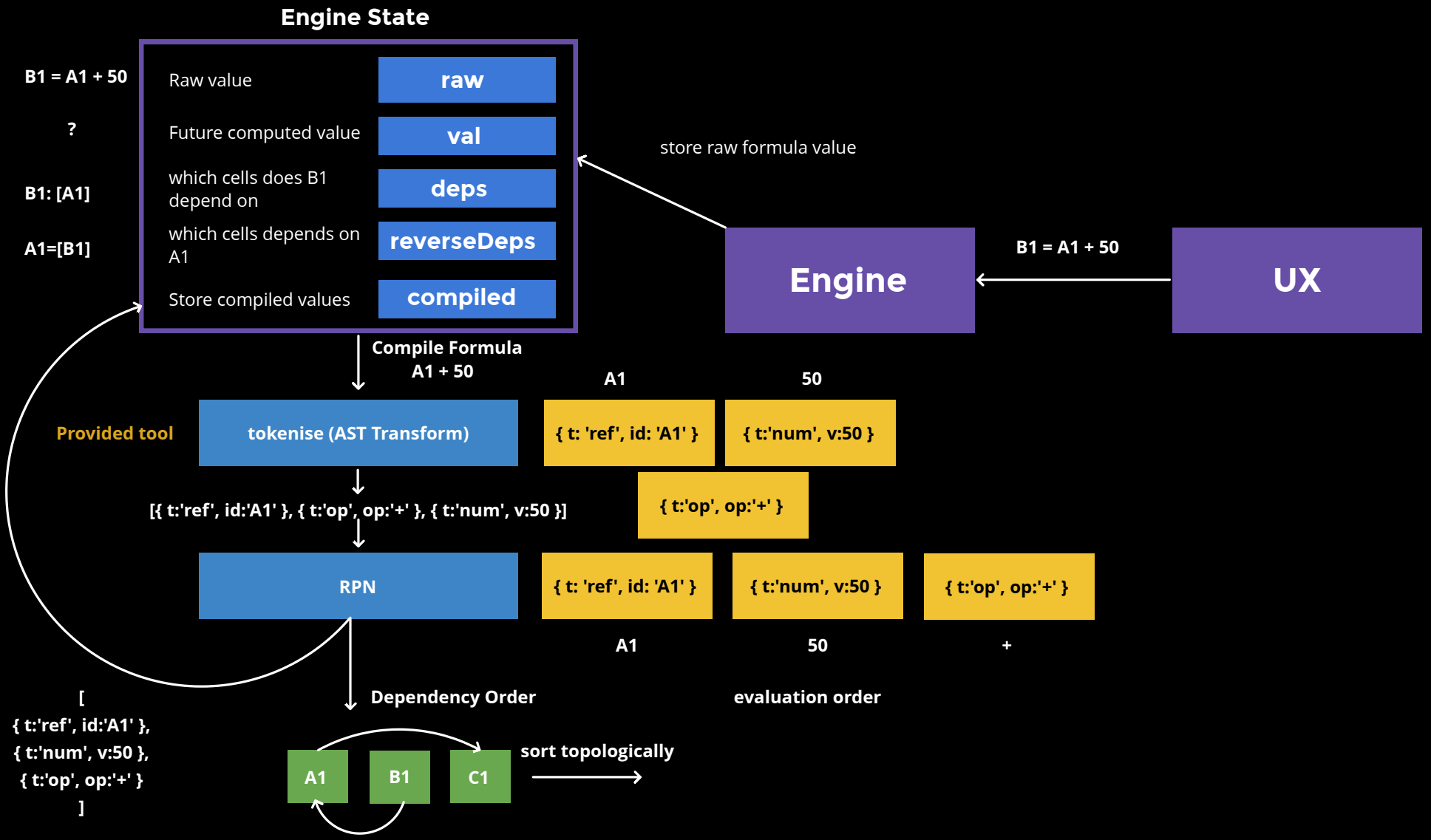
Google Sheet - Decompousing a problem



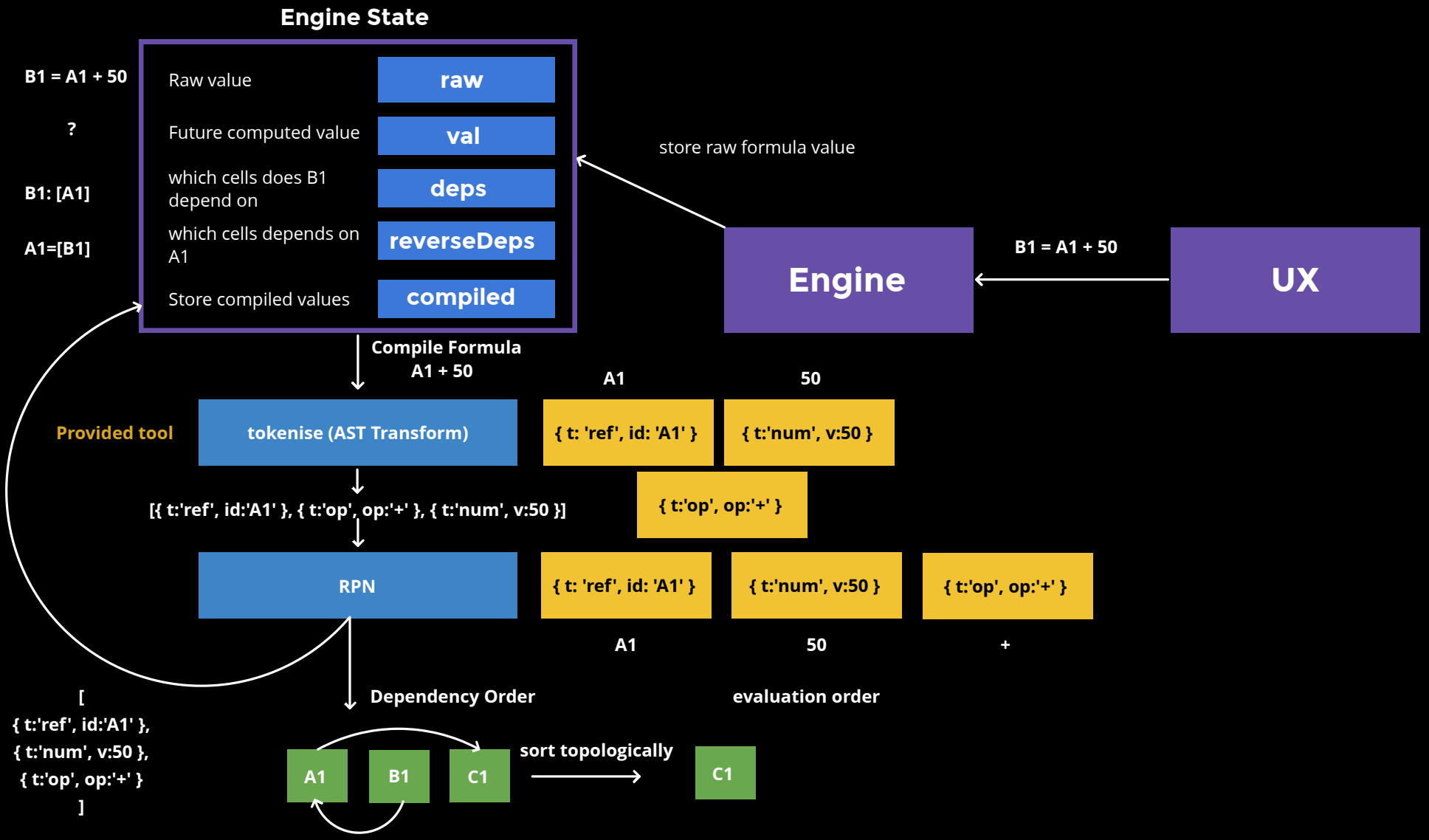
Google Sheet - Decompousing a problem



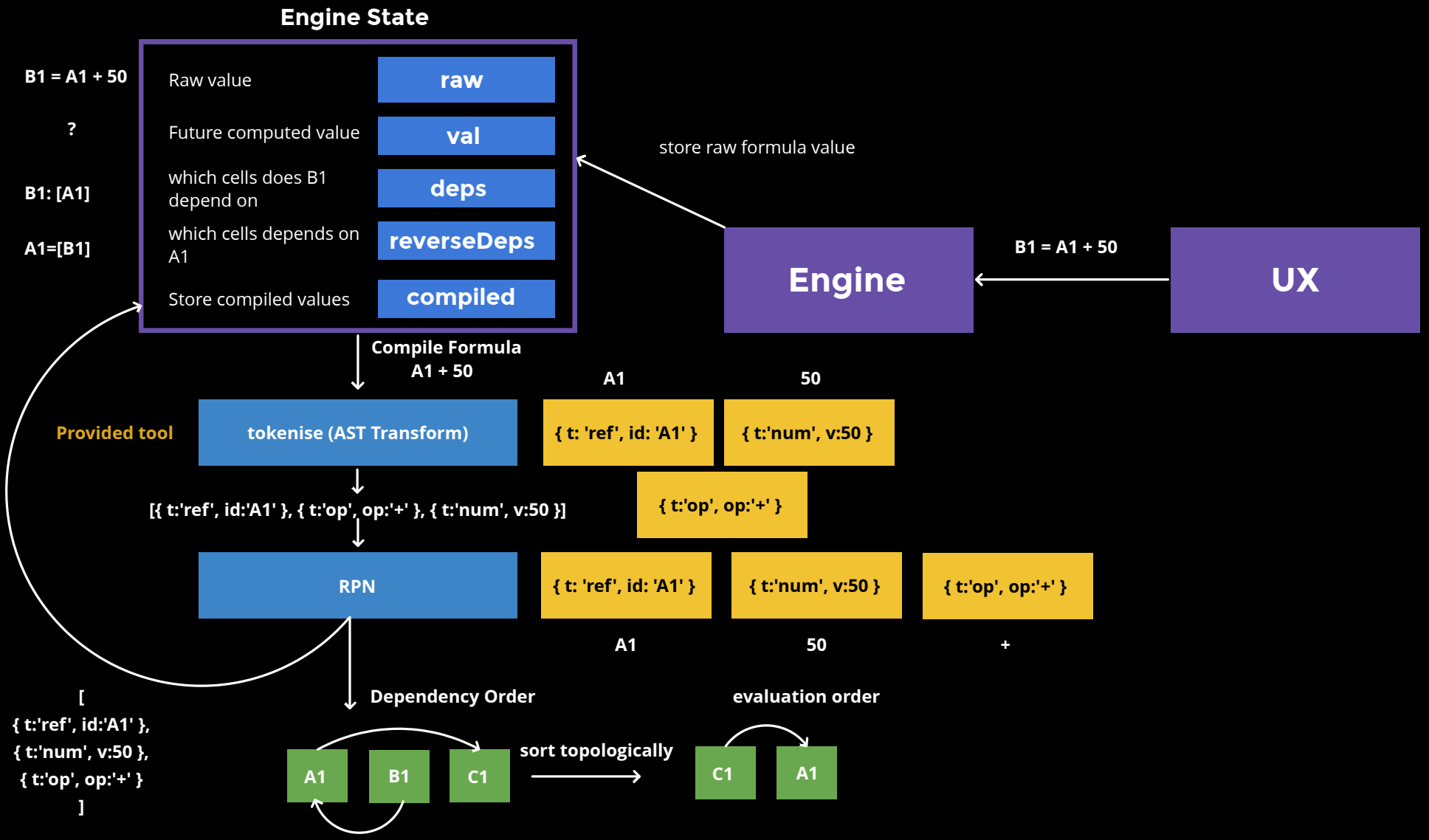
Google Sheet - Decompousing a problem



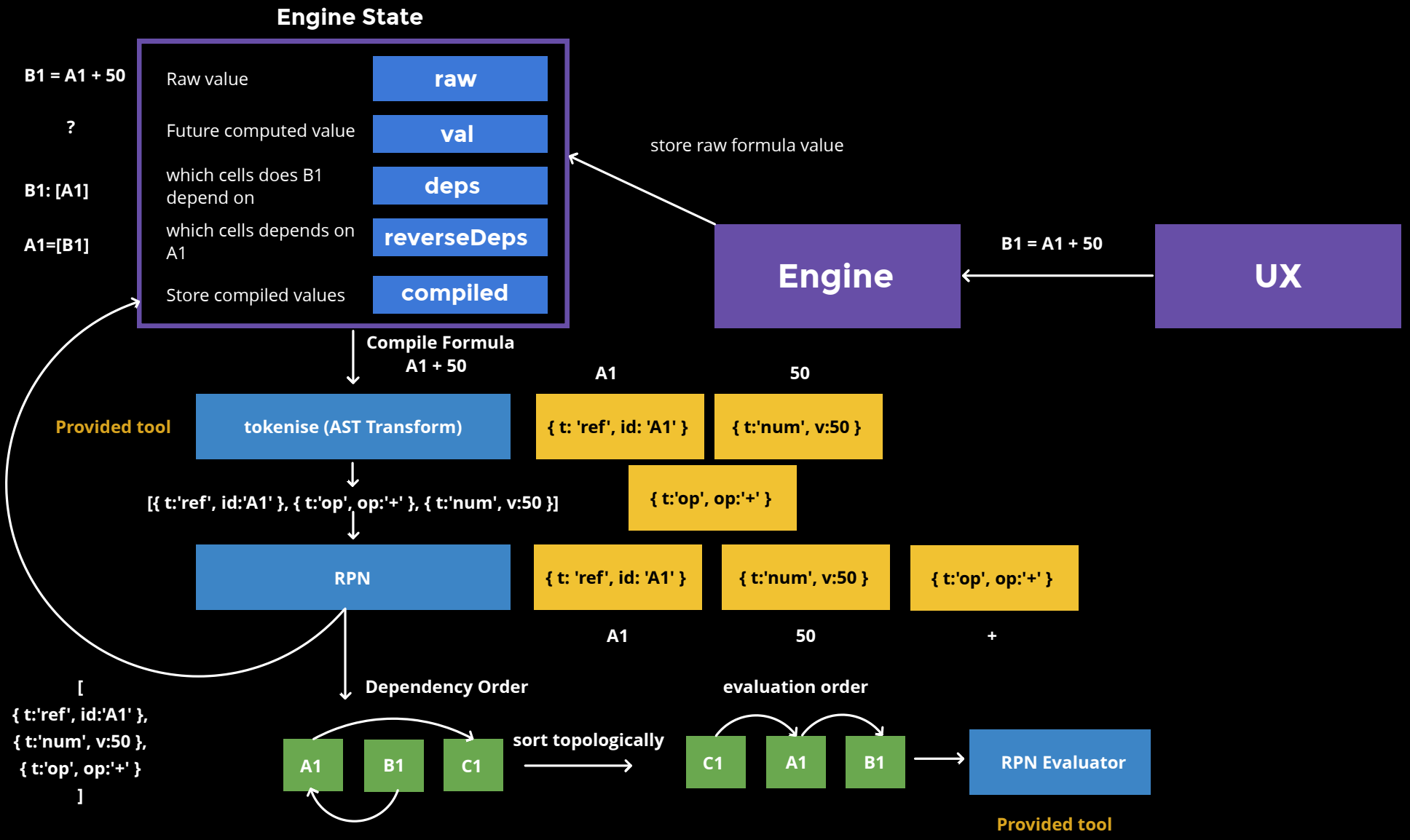
Google Sheet - Decompousing a problem



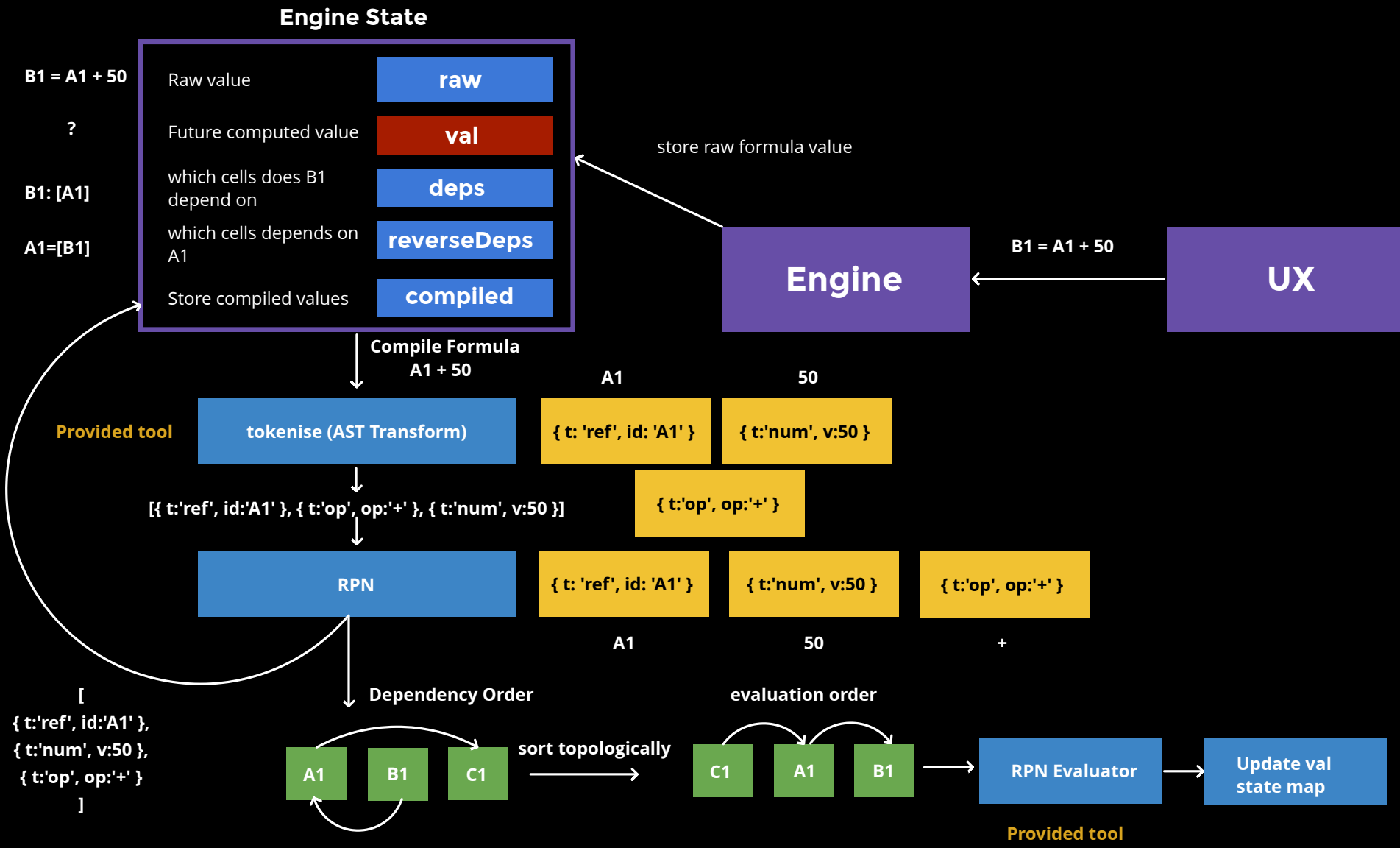
Google Sheet - Decompousing a problem



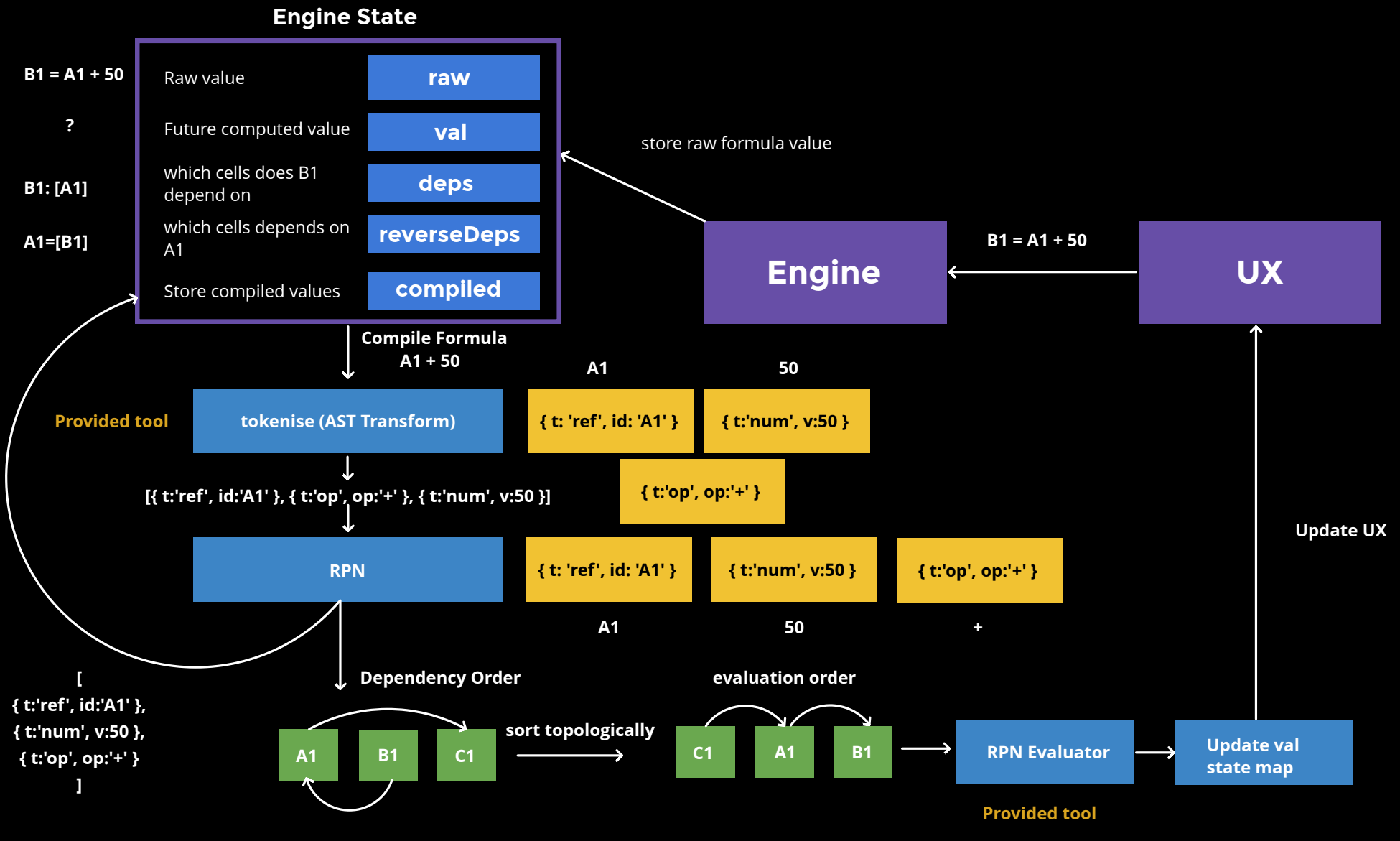
Google Sheet - Decompousing a problem



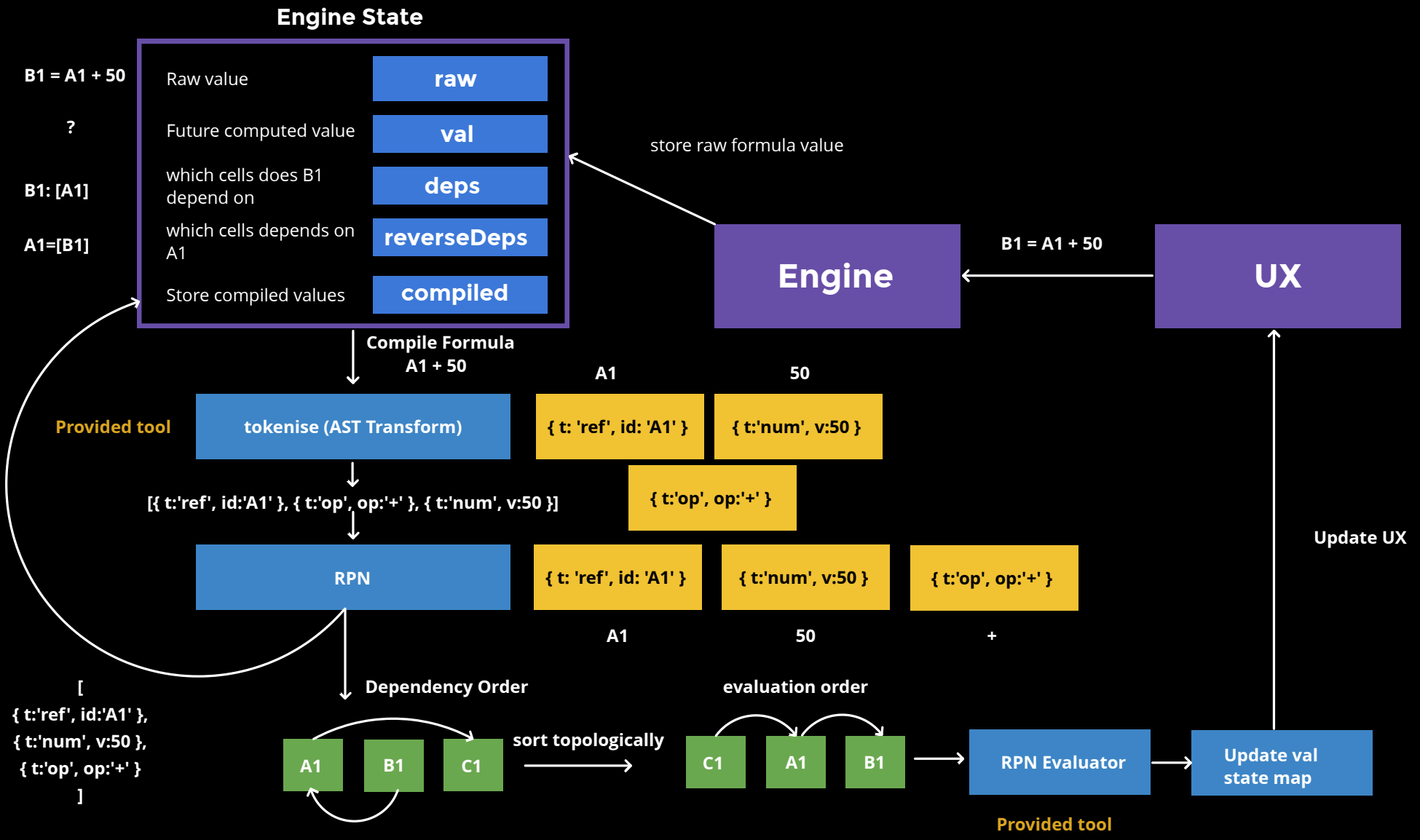
Google Sheet - Decompousing a problem



Google Sheet - Decompousing a problem

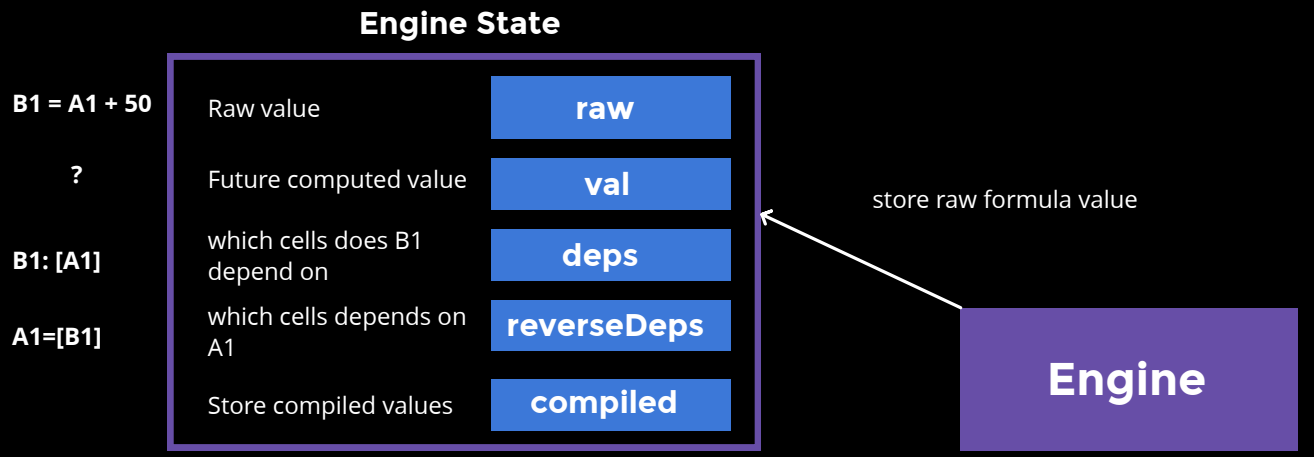


Google Sheet - Decompousing a problem



Ready to tame the beast?

Step 1: Build basic class structure



Step 1: Build basic class structure

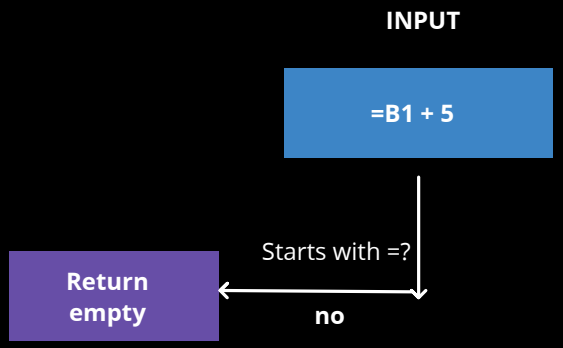
```ts
export class TableEngine {
  #raw: Map<CellId, string> = new Map();
  #val: Map<CellId, string> = new Map();
  #deps: Map<CellId, Set<CellId>> = new Map();
  #reverseDeps: Map<CellId, Set<CellId>> = new Map();
  #compiled: Map<CellId, Compiled> = new Map();

  setRaw(_id: CellId, _raw: string): { changed: CellId[] } {
    this.#raw.set(_id, _raw);
    this.#val.set(_id, _raw);
    return {changed: [_id]}
  }

  getRaw(_id: CellId): string {
    return this.#raw.get(_id) ?? '';
  }

  getValue(_id: CellId): string {
    return this.#val.get(_id) ?? '';
  }

  getDeps(_id: CellId): ReadonlySet<CellId> {
    const deps = this.#deps.get(_id) ?? new Set();
    this.#deps.set(_id, deps);
    return deps;
  }
  getRevDeps(_id: CellId): ReadonlySet<CellId> {
    const deps = this.#reverseDeps.get(_id) ?? new Set();
    this.#reverseDeps.set(_id, deps);
    return deps;
  }
}
```

Step 2: Understand compilation step

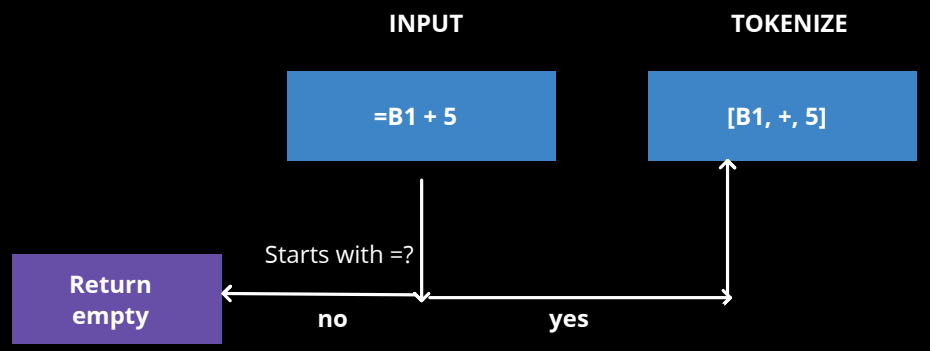
INPUT

=B1 + 5

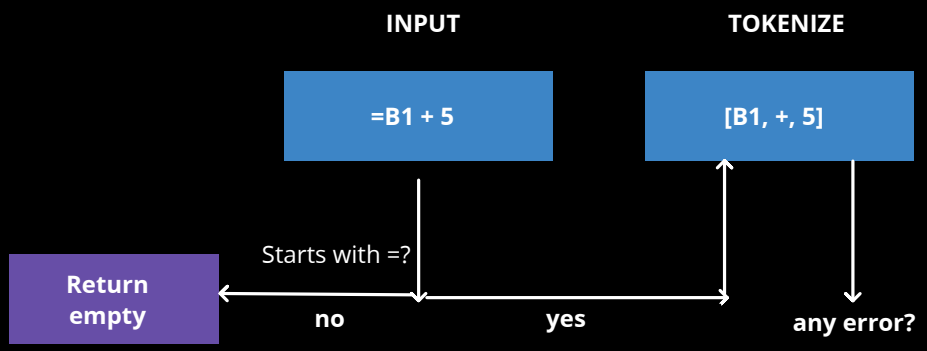
Step 2: Understand compilation step



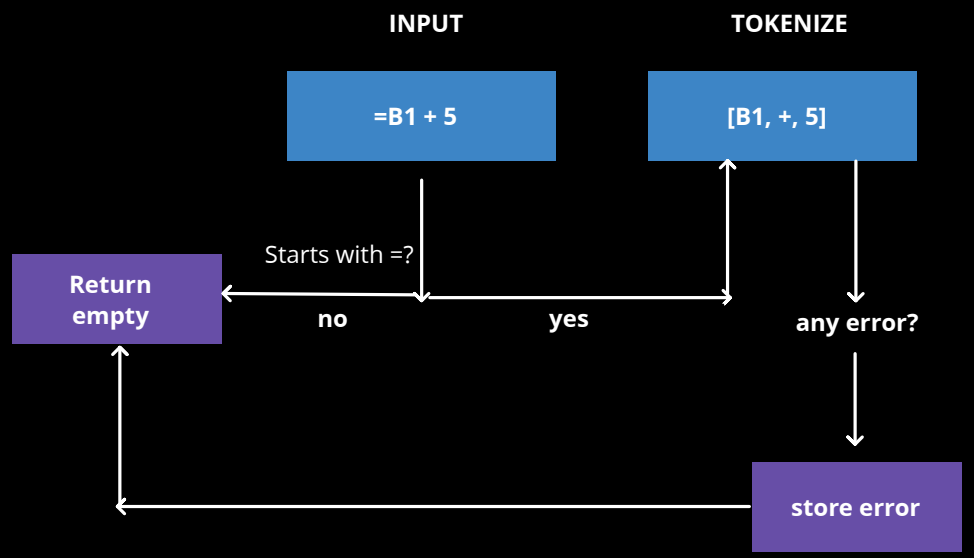
Step 2: Understand compilation step



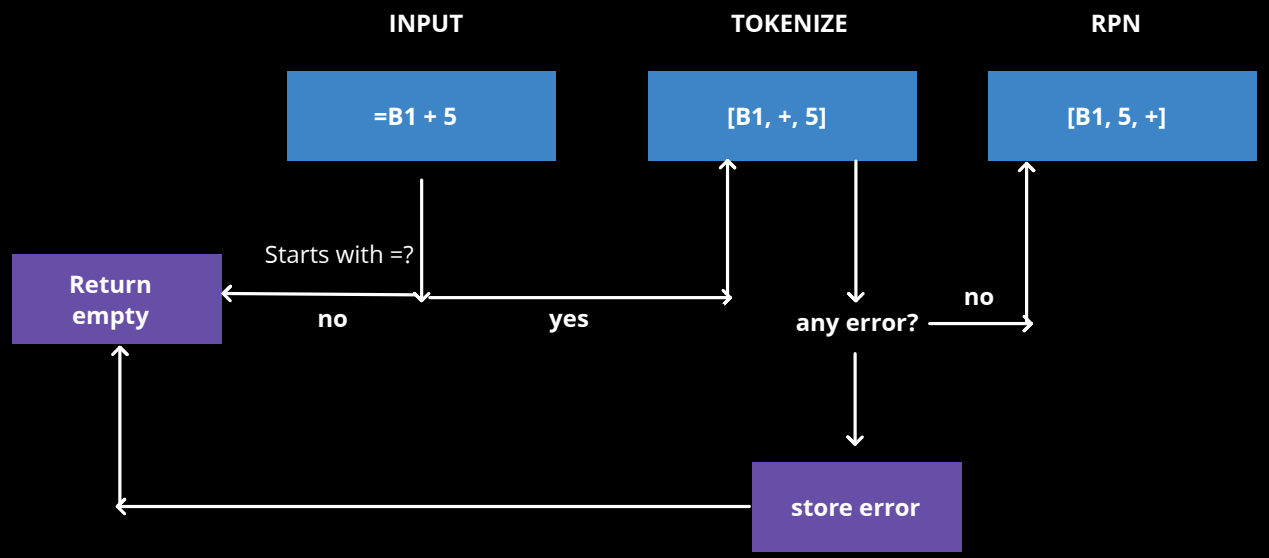
Step 2: Understand compilation step



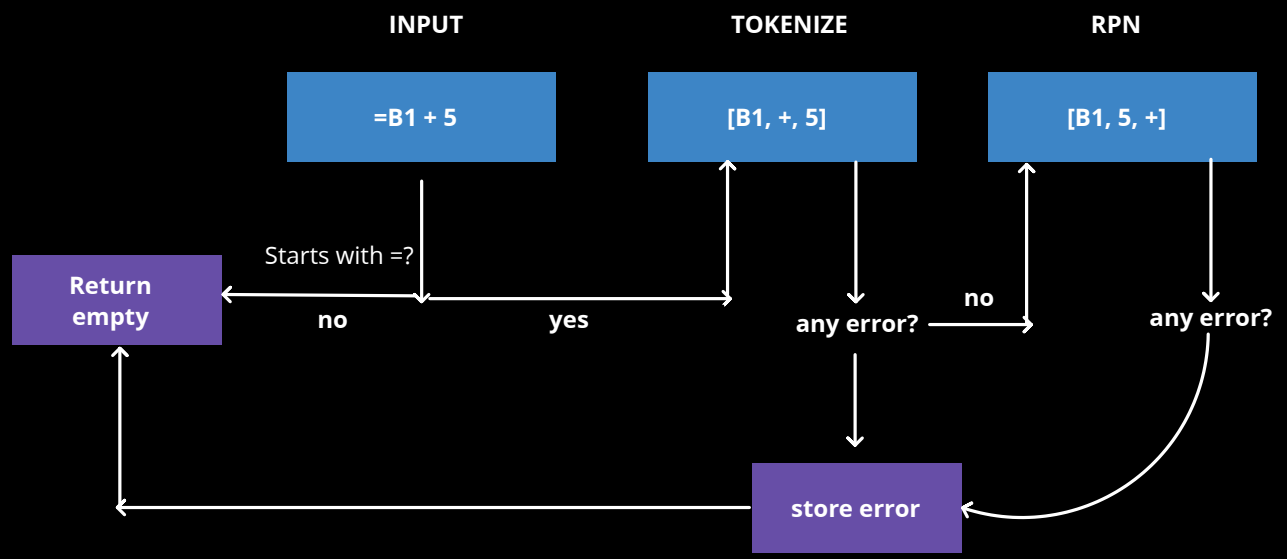
Step 2: Understand compilation step



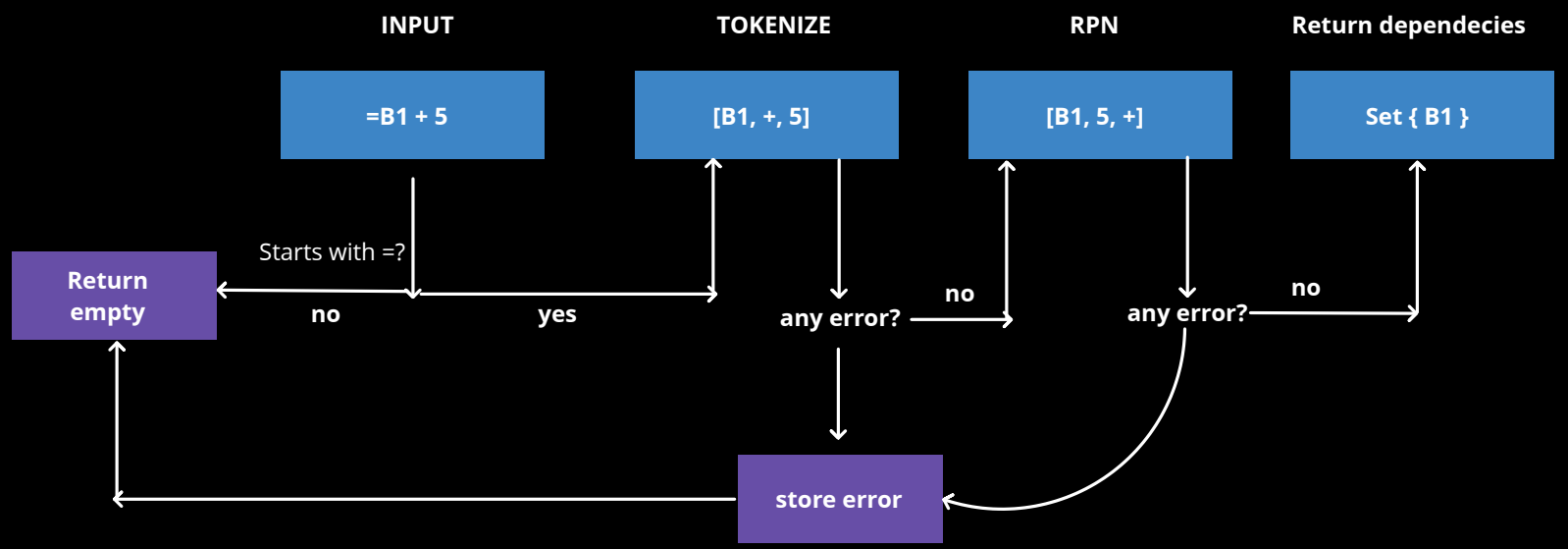
Step 2: Understand compilation step



Step 2: Understand compilation step



Step 2: Understand compilation step



Step 2: Understand compilation step

```
#compile(id: CellId, raw: string): Set<CellId> {
    const deps = new Set<CellId>()
    raw = raw.trim()

    if (!raw.startsWith('=')) {
      this.#compiled.set(id, null)
      return deps
    }

    const expr = raw.slice(1).trim()

    const tokens = tokenize(expr)
    if (!tokens.ok) {
      this.#compiled.set(id, { error: tokens.error })
      return deps
    }

    const rpn = toRpn(tokens.tokens)
    if (!rpn.ok) {
      this.#compiled.set(id, { error: rpn.error })
      return deps
    }

    for (const t of rpn.rpn) {
      if (t.t === 'ref') deps.add(t.id)
    }

    this.#compiled.set(id, { rpn: rpn.rpn })
    return deps
}
```

Step 3: Update direct and reverse dependecies

Previous

A1 = B1+C1

Step 3: Update direct and reverse dependecies

Previous

A1 = B1+C1

Next

A1 = B1+D1

Step 3: Update direct and reverse dependecies

Previous

A1 = B1+C1

A1 deps: B1, C1

Reverse:

B1: A1

C1: A1

Next

A1 = B1+D1

1. Remove C1 from A1 deps
2. Remove A1 from C1 reverse deps

Step 3: Update direct and reverse dependecies

Previous

A1 = B1+C1

A1 deps: B1, C1

Reverse:

B1: A1

C1: A1

Next

A1 = B1+D1

A1 Deps: B1, D1

Reverse:

B1: A1

C1: <none>

D1: A1

Step 3: Update direct and reverse dependecies

Previous

A1 = B1+C1

A1 deps: B1, C1

Reverse:

B1: A1

C1: A1

Next

A1 = B1+D1

A1 Deps: B1, D1

Reverse:

B1: A1

C1: A1

D1: A1

1. Remove C1 from A1 deps

Step 3: Update direct and reverse dependecies

Previous

A1 = B1+C1

A1 deps: B1, C1

Reverse:

B1: A1

C1: A1

Next

A1 = B1+D1

A1 Deps: B1, D1

Reverse:

B1: A1

C1: <none>

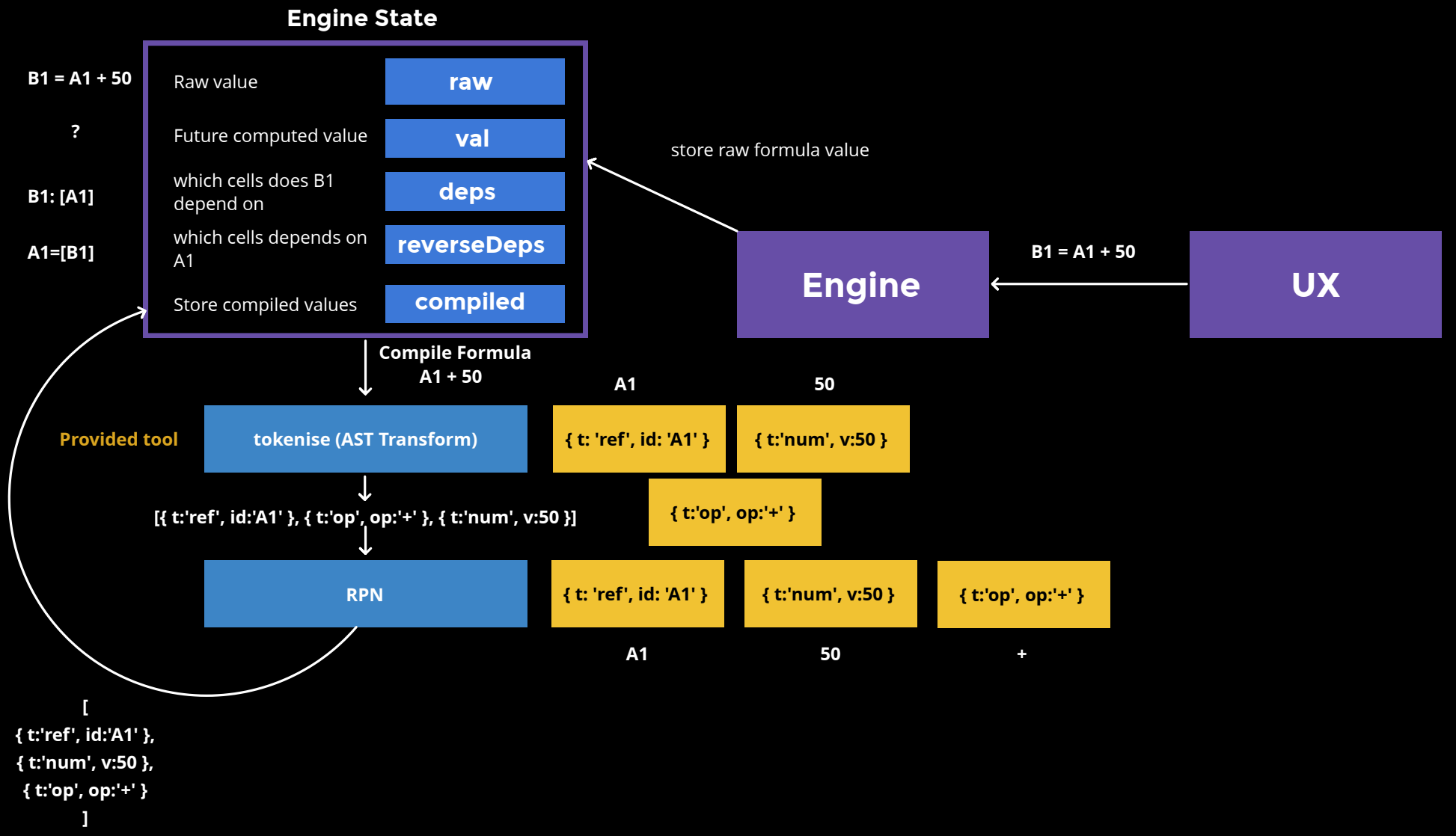
D1: A1

1. Remove C1 from A1 deps
2. Remove A1 from C1 reverse deps

Step 3: Solution

```
#setDeps(id: CellId, nextDeps: Set<CellId>) {
  const prevDeps = this.#getDeps(id)
  for (const dep of prevDeps) {
    if (!nextDeps.has(dep)) this.#getRevDeps(dep).delete(id)
  }
  for (const dep of nextDeps) {
    if (!prevDeps.has(dep)) this.#getRevDeps(dep).add(id)
  }
  this.#deps.set(id, nextDeps)
}

setRaw(id: CellId, raw: string): { changed: CellId[] } {
  this.#raw.set(id, raw)
  this.#value.set(id, raw)          // still no evaluation yet
  const deps = this.#compile(id, raw)
  this.#setDeps(id, deps)
  return { changed: [id] }
}
```

Where are we so far?



Step 4: Topological Ordering

	A	B	C
1			
2			

Step 4: Topological Ordering

	A	B	C
1	10	=A1 + 5	
2			

Step 4: Topological Ordering

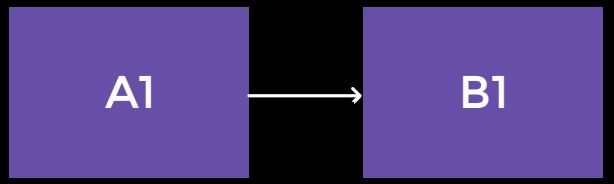
	A	B	C
1	10	=A1 + 5	=B1*2
2			

Step 4: Topological Ordering

	A	B	C
1	10	=A1 + 5	**=B1*2**
2			

Step 4: Topological Ordering

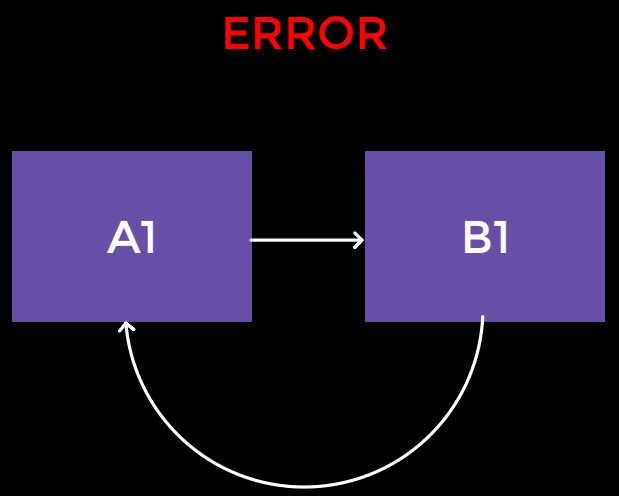
	A	B	C
1	10	=A1 + 5	**=B1*2**
2			



Step 4: Topological Ordering

	A	B	C
1	10	=A1 + 5	**=B1*2**
2			

Step 4: Topological Ordering - Cycle Detection

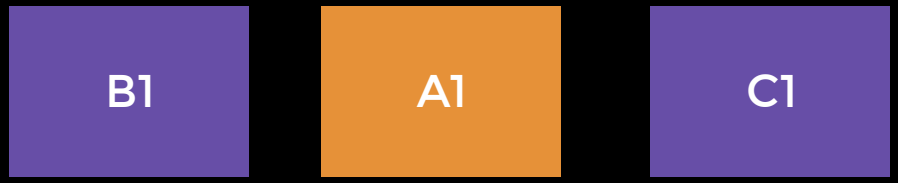


Step 4: Topological Ordering - Khan Algorithm

	A	B	C
1	10	=A1 + 5	**=B1*2**
2			

Step 4: Topological Ordering - Khan Algorithm

	A	B	C
1	10	=A1 + 5	**=B1*2**
2			



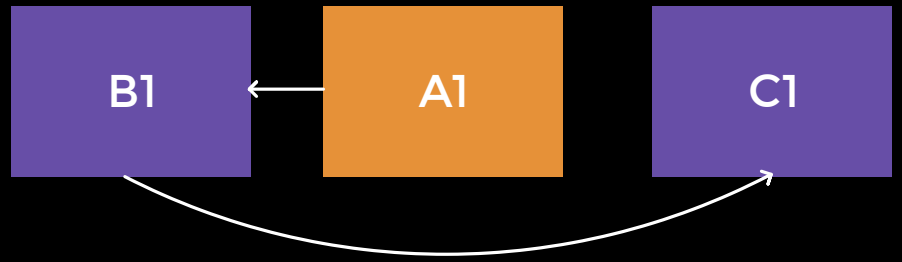
Step 1 - A1 is edited

rev(A1) = {B1} → add B1 to affected
rev(B1) = {C1} → add C1 to affected
rev(C1) = {} → done

Affected set = {A1, B1, C1}

Step 4: Topological Ordering - Khan Algorithm

	A	B	C
1	10	=A1 + 5	**=B1*2**
2			



Step 1 - A1 is edited

rev(A1) = {B1} → add B1 to affected
rev(B1) = {C1} → add C1 to affected
rev(C1) = {} → done

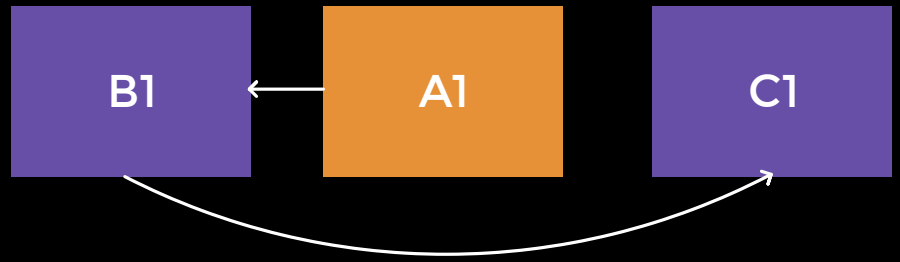
Affected set = {A1, B1, C1}

Step 2 - in-degrees

A1: deps = {} → in-degree = 0
B1: deps = {A1} → in-degree = 1
C1: deps = {B1} → in-degree = 1

Step 4: Topological Ordering - Khan Algorithm

	A	B	C
1	10	=A1 + 5	**=B1*2**
2			



Step 1 - A1 is edited

rev(A1) = {B1} → add B1 to affected
rev(B1) = {C1} → add C1 to affected
rev(C1) = {} → done

Affected set = {A1, B1, C1}

Step 2 - in-degrees

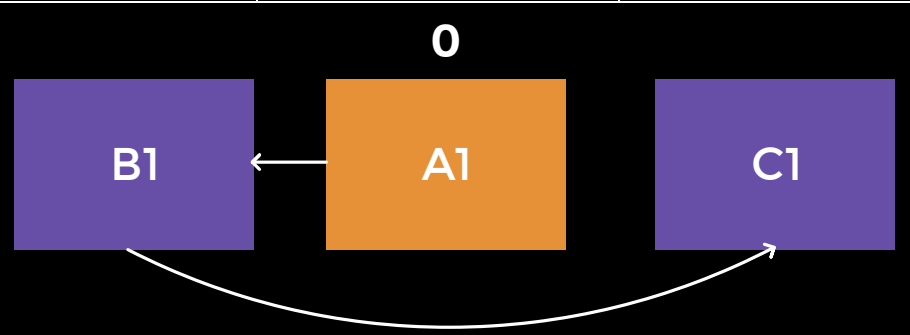
A1: deps = {} → in-degree = 0
B1: deps = {A1} → in-degree = 1
C1: deps = {B1} → in-degree = 1

Step 3 - Process

A1: deps = {} → in-degree = 0
B1: deps = {A1} → in-degree = 1
C1: deps = {B1} → in-degree = 1

Step 4: Topological Ordering - Khan Algorithm

	A	B	C
1	10	=A1 + 5	**=B1*2**
2			



Step 1 - A1 is edited

rev(A1) = {B1} → add B1 to affected
rev(B1) = {C1} → add C1 to affected
rev(C1) = {} → done

Affected set = {A1, B1, C1}

Step 2 - in-degrees

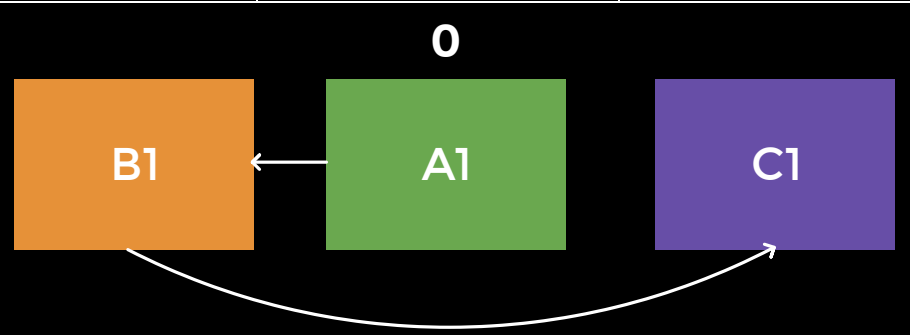
A1: deps = {} → in-degree = 0
B1: deps = {A1} → in-degree = 1
C1: deps = {B1} → in-degree = 1

Step 3 - Process

A1: deps = {} → in-degree = 0
B1: deps = {A1} → in-degree = 1
C1: deps = {B1} → in-degree = 1

Step 4: Topological Ordering - Khan Algorithm

	A	B	C
1	10	=A1 + 5	**=B1*2**
2			



Step 1 - A1 is edited

rev(A1) = {B1} **→ add B1 to affected**
rev(B1) = {C1} → add C1 to affected
rev(C1) = {} → done

Affected set = {A1, B1, C1}

Step 2 - in-degrees

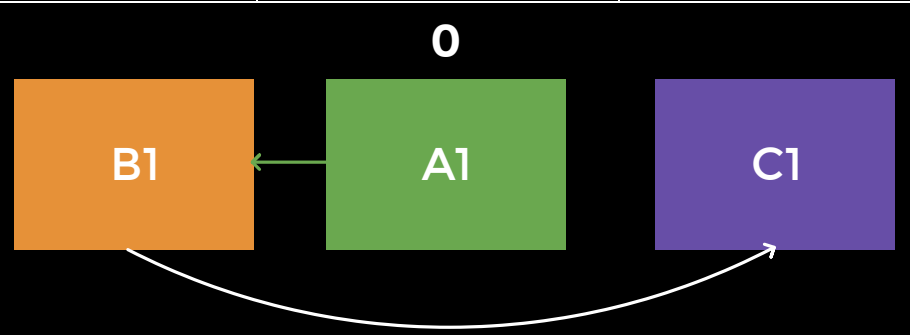
A1: deps = {} → in-degree = 0
B1: deps = {A1} → in-degree = 1
C1: deps = {B1} → in-degree = 1

Step 3 - Process

A1: deps = {} **→ in-degree = 0**
B1: deps = {A1} → in-degree = 1
C1: deps = {B1} → in-degree = 1

Step 4: Topological Ordering - Khan Algorithm

	A	B	C
1	10	=A1 + 5	**=B1*2**
2			



Step 1 - A1 is edited

rev(A1) = {B1} **→ add B1 to affected**
rev(B1) = {C1} → add C1 to affected
rev(C1) = {} → done

Affected set = {A1, B1, C1}

Step 2 - in-degrees

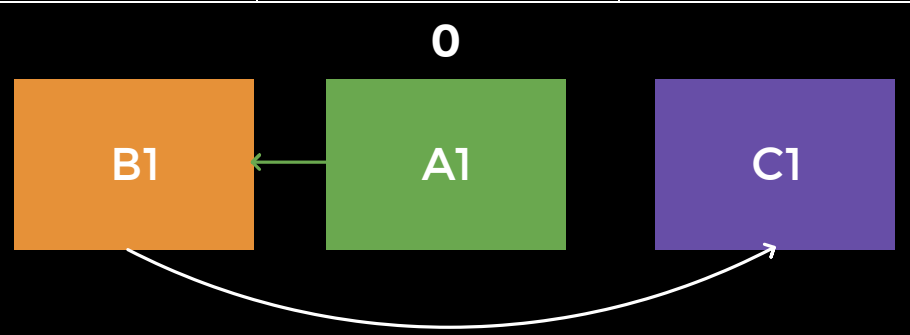
A1: deps = {} → in-degree = 0
B1: deps = {A1} → in-degree = 1
C1: deps = {B1} → in-degree = 1

Step 3 - Process

A1: deps = {} **→ in-degree = 0**
B1: deps = {} **→ in-degree = 0**
C1: deps = {B1} → in-degree = 1

Step 4: Topological Ordering - Khan Algorithm

	A	B	C
1	10	=A1 + 5	**=B1*2**
2			



Step 1 - A1 is edited

rev(A1) = {B1} → **add B1 to affected**
rev(B1) = {C1} → **add C1 to affected**
rev(C1) = {} → done

Affected set = {A1, B1, C1}

Step 2 - in-degrees

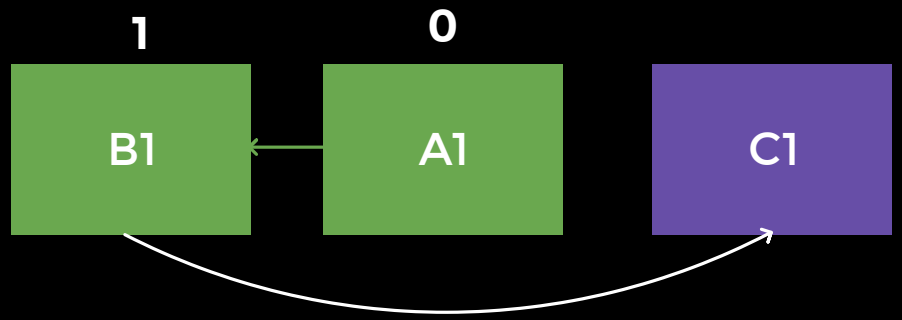
A1: deps = {} → in-degree = 0
B1: deps = {A1} → in-degree = 1
C1: deps = {B1} → in-degree = 1

Step 3 - Process

A1: deps = {} → **in-degree = 0**
B1: deps = {} → **in-degree = 0**
C1: deps = {B1} → in-degree = 1

Step 4: Topological Ordering - Khan Algorithm

	A	B	C
1	10	=A1 + 5	**=B1*2**
2			



Step 1 - A1 is edited

rev(A1) = {B1} → **add B1 to affected**
rev(B1) = {C1} → **add C1 to affected**
rev(C1) = {} → done

Affected set = {A1, B1, C1}

Step 2 - in-degrees

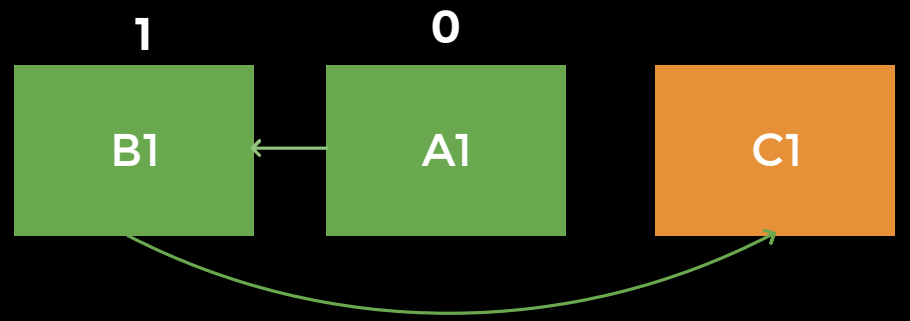
A1: deps = {} → in-degree = 0
B1: deps = {A1} → in-degree = 1
C1: deps = {B1} → in-degree = 1

Step 3 - Process

A1: deps = {} → **in-degree = 0**
B1: deps = {} → **in-degree = 0**
C1: deps = {B1} → in-degree = 1

Step 4: Topological Ordering - Khan Algorithm

	A	B	C
1	10	=A1 + 5	**=B1*2**
2			



Step 1 - A1 is edited

rev(A1) = {B1} → **add B1 to affected**
rev(B1) = {C1} → **add C1 to affected**
rev(C1) = {} → done

Affected set = {A1, B1, C1}

Step 2 - in-degrees

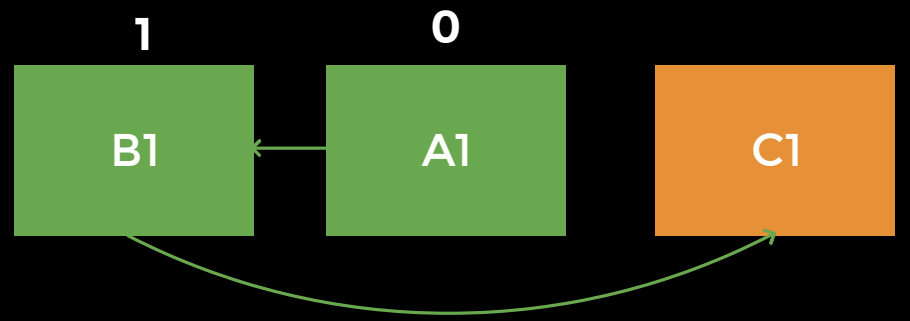
A1: deps = {} → in-degree = 0
B1: deps = {A1} → in-degree = 1
C1: deps = {B1} → in-degree = 1

Step 3 - Process

A1: deps = {} → **in-degree = 0**
B1: deps = {} → **in-degree = 0**
C1: deps = {B1} → in-degree = 1

Step 4: Topological Ordering - Khan Algorithm

	A	B	C
1	10	=A1 + 5	**=B1*2**
2			



Step 1 - A1 is edited

rev(A1) = {B1} → **add B1 to affected**
rev(B1) = {C1} → **add C1 to affected**
rev(C1) = {} → done

Affected set = {A1, B1, C1}

Step 2 - in-degrees

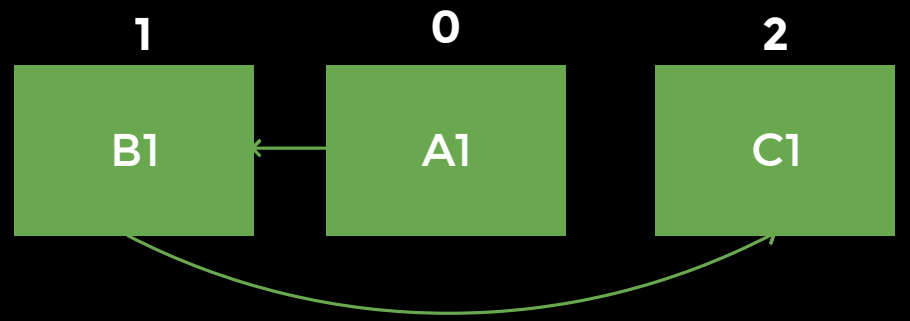
A1: deps = {} → in-degree = 0
B1: deps = {A1} → in-degree = 1
C1: deps = {B1} → in-degree = 1

Step 3 - Process

A1: deps = {} → **in-degree = 0**
B1: deps = {} → **in-degree = 0**
C1: deps = {} → **in-degree = 0**

Step 4: Topological Ordering - Khan Algorithm

	A	B	C
1	10	=A1 + 5	**=B1*2**
2			



Step 1 - A1 is edited

rev(A1) = {B1} → **add B1 to affected**
rev(B1) = {C1} → **add C1 to affected**
rev(C1) = {} → **done**

Affected set = {A1, B1, C1}

Step 2 - in-degrees

A1: deps = {} → in-degree = 0
B1: deps = {A1} → in-degree = 1
C1: deps = {B1} → in-degree = 1

Step 3 - Process

A1: deps = {} → **in-degree = 0**
B1: deps = {} → **in-degree = 0**
C1: deps = {} → **in-degree = 0**

Step 4: Topological Ordering - Khan Algorithm

	A	B	C
1	10	=A1 + 5	**=B1*2**
2			

Step 1 - A1 is edited

rev(A1) = {B1} → **add B1 to affected**
rev(B1) = {C1} → **add C1 to affected**
rev(C1) = {} → **done**

Affected set = {A1, B1, C1}

Step 2 - in-degrees

A1: deps = {} → in-degree = 0
B1: deps = {A1} → in-degree = 1
C1: deps = {B1} → in-degree = 1

Step 3 - Process

A1: deps = {} → **in-degree = 0**
B1: deps = {} → **in-degree = 0**
C1: deps = {} → **in-degree = 0**

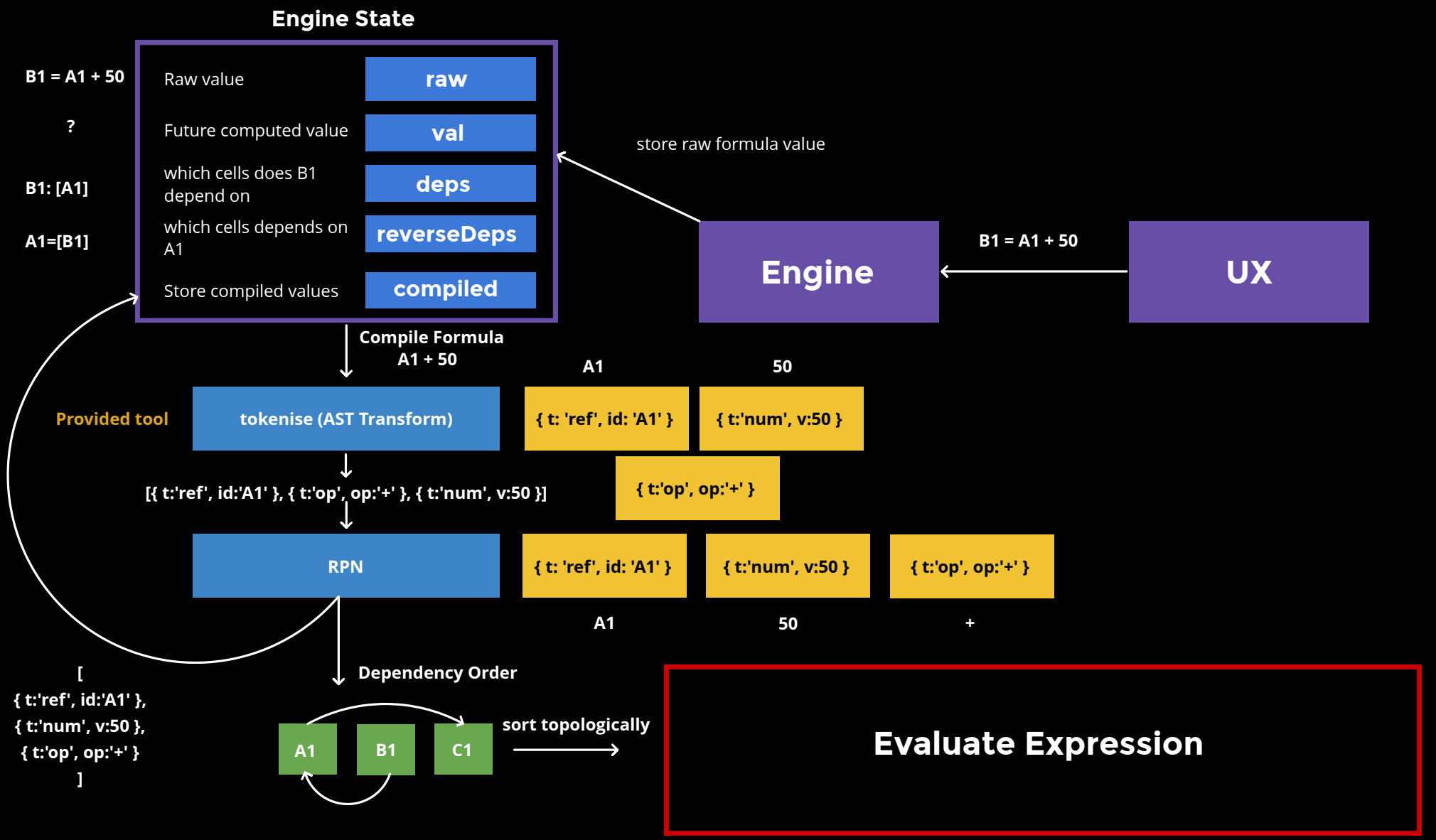
Step 4: get affected cells

```
#affectedFrom(start: CellId): Set<CellId> {
  const affected = new Set<CellId>()
  const queue: CellId[] = [start]
  for (let i = 0; i < queue.length; i++) {
    const id = queue[i]!
    if (affected.has(id)) continue
    affected.add(id)
    for (const dep of this.#getRevDeps(id)) {
       queue.push(dep)
    }
  }
  return affected
}
```

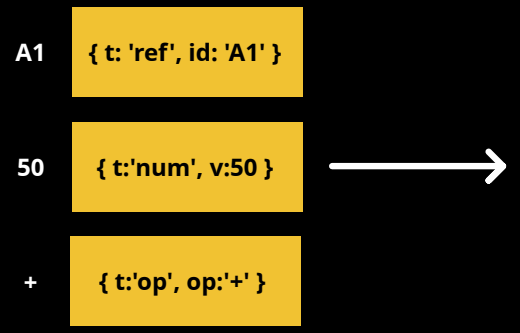
Step 4: Topological sort

```
#topoSort(affected: Set<CellId>): { order: CellId[]; cyclic: Set<CellId> } {
  // Phase 2: in-degrees
  const inDegree = new Map<CellId, number>()
  for (const id of affected) {
    let deg = 0
    for (const dep of this.#getDeps(id)) {
      if (affected.has(dep)) deg++
    }
    inDegree.set(id, deg)
  }
  // Phase 3: BFS from in-degree 0
  const queue: CellId[] = []
  for (const [id, deg] of inDegree) {
    if (deg === 0) queue.push(id)
  }
  const order: CellId[] = []
  for (let i = 0; i < queue.length; i++) {
    const id = queue[i]!
    order.push(id)
    for (const dependent of this.#getRevDeps(id)) {
      if (!affected.has(dependent)) continue
      const next = (inDegree.get(dependent) ?? 0) - 1
      inDegree.set(dependent, next)
      if (next === 0) queue.push(dependent)
    }
  }
  // Phase 4: cycle detection
  const cyclic = new Set<CellId>()
  if (order.length !== affected.size) {
    const inOrder = new Set(order)
    for (const id of affected) {
      if (!inOrder.has(id)) cyclic.add(id)
    }
  }
  return { order, cyclic }
}
```

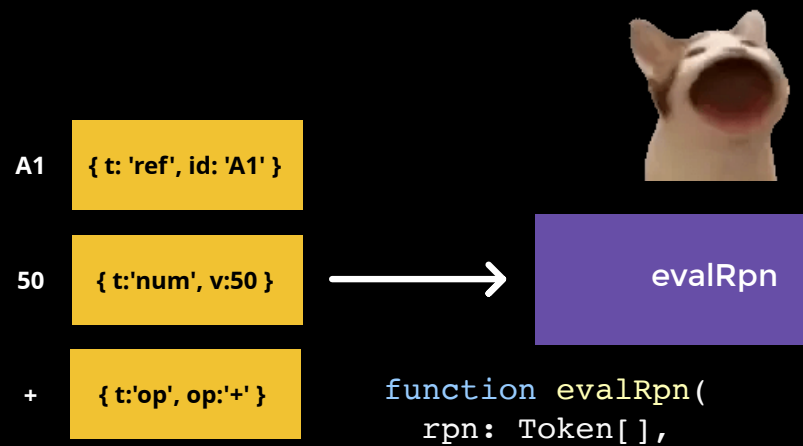
Where are we now?



Evaluating a single cell

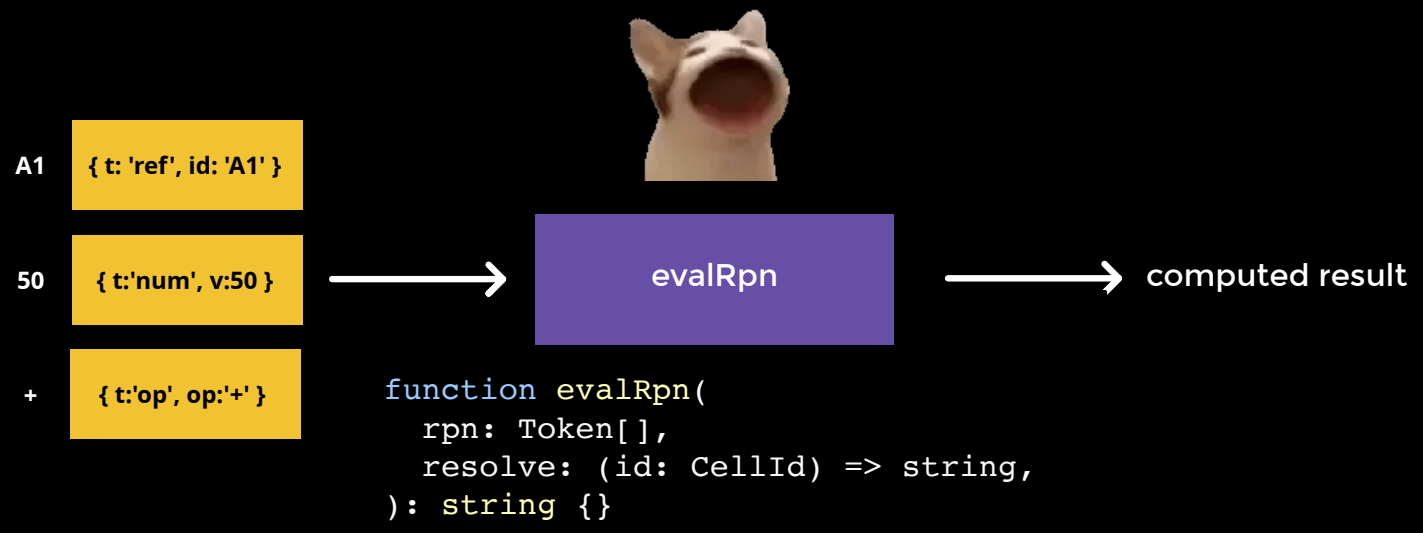


Evaluating a single cell




```
function evalRpn(
  rpn: Token[],
  resolve: (id: CellId) => string,
): string {}
```

Evaluating a single cell




```
function evalRpn(
  rpn: Token[],
  resolve: (id: CellId) => string,
): string {}
```

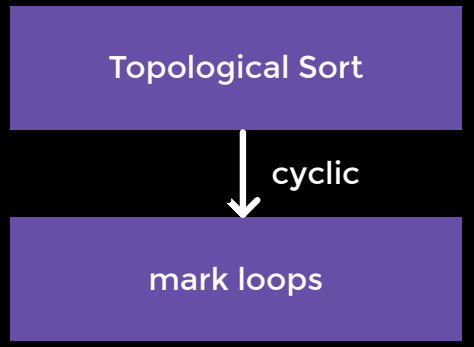
Solution: Evaluating a single cell

```
_evalCell(id: CellId): string {
  const raw = (this.#raw.get(id) ?? '').trim()
  if (!raw.startsWith('='))
     return raw
  const compiled = this.#compiled.get(id)
  if (!compiled)
     return ERROR
  if ('error' in compiled)
     return ERROR
  return evalRpn(compiled.rpn, (refId) => this.#value.get(refId) ?? '')
}

// Updated setRaw — now evaluates the cell
setRaw(id: CellId, raw: string): { changed: CellId[] } {
  this.#raw.set(id, raw)
  const deps = this.#compile(id, raw)
  this.#setDeps(id, deps)
  const next = this._evalCell(id)
  this.#value.set(id, next)
  return { changed: [id] }
}
```

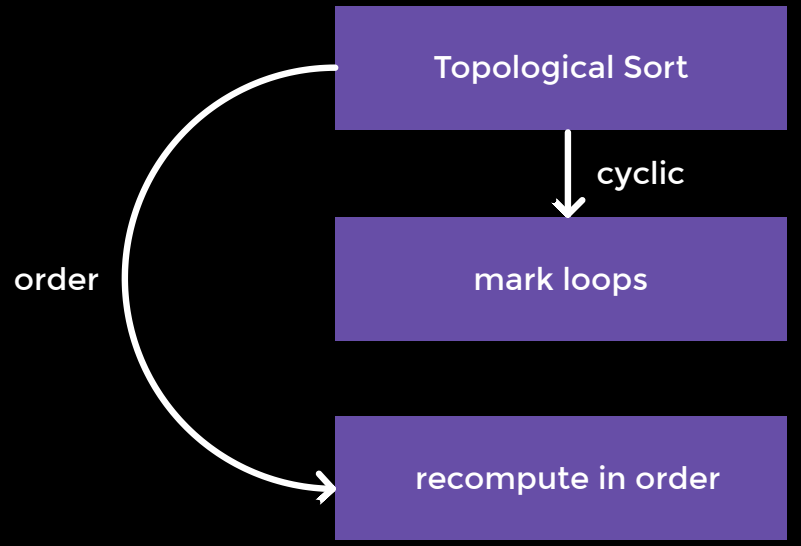
Recomputing the whole chain

Topological Sort

Recomputing the whole chain



Recomputing the whole chain



Solution: Chain recompute

```
#recomputeFrom(start: CellId): CellId[] {
  const affected = this.#affectedFrom(start)
  const { order, cyclic } = this.#topoSort(affected)
  const changed: CellId[] = []
  // Handle cycles first — so non-cyclic cells see #CYCLE! via resolve
  for (const id of cyclic) {
    const prev = this.#value.get(id) ?? ''
    if (prev !== CYCLE) {
      this.#value.set(id, CYCLE)
      changed.push(id)
    }
  }
  // Evaluate in topological order
  for (const id of order) {
    if (cyclic.has(id)) continue
    const prev = this.#value.get(id) ?? ''
    const next = this.#evalCell(id)
    if (prev !== next) {
      this.#value.set(id, next)
      changed.push(id)
    }
  }
  return changed
}

// Final setRaw — the complete pipeline
setRaw(id: CellId, raw: string): { changed: CellId[] } {
  this.#raw.set(id, raw)
  const deps = this.#compile(id, raw)
  this.#setDeps(id, deps)
  return { changed: this.#recomputeFrom(id) }
}
```

Last step: Rendering the table

- Create a Cell component
- Prepare Headers
- Prepare Rows
- Handle Events

Create a Cell component

```tsx
type TCellProps = {
  column: string | symbol
  row: number
  value: React.ReactNode
}

function Cell({ column, row, value }: TCellProps) {
  const isHeader = column === EMPTY          // row header (1, 2, 3…)
  const isColHeader = row === 0              // column header (A, B, C…)
  const role = isColHeader ? 'columnheader'
             : isHeader    ? 'rowheader'
             :               'gridcell'

  return (
    <div
      role={role}
      data-value={value}
      contentEditable={role === 'gridcell'}   // only data cells are editable
      data-column={String(column)}
      data-row={row}
      className={cx(css.cell, isColHeader || isHeader ? css.header : '')}
      suppressContentEditableWarning
    >
      {value}
    </div>
  )
}
```

Prepare Table Headers

```
const EMPTY = Symbol(' ')
const COLS = ['A','B','C', /* … */ 'Z'] as const
const TABLE_COLUMNS = [EMPTY, ...COLS]   // first column = row numbers

const HEADER_ROWS = (
  <div role="row" style={{ display: 'contents' }}>
    {TABLE_COLUMNS.map((column) => (
      <Cell
        key={String(column)}
        column={column}
        row={0}
        value={column === EMPTY ? '' : String(column)}
      />
    ))}
  </div>
```

Prepare Table rows

```
1  const MAX_ROWS = 500
2
3  const BODY_ROWS = Array.from({ length: MAX_ROWS }).map((_, idx) => {
4    const rowId = idx + 1
5    return (
6      <div role="row" key={idx} style={{ display: 'contents' }}>
7        {TABLE_COLUMNS.map((column) => (
8          <Cell
9            key={String(column) + idx}
10           column={column}
11           row={rowId}
12           value={column === EMPTY ? rowId : null}
13         />
14       ))}
15     </div>
16   )
17 })
```

Render table

```
const engine = new TableEngine()

export function GoogleSheet() {
  return (
    <section>
      <div
        className={css.container}
        role="grid"
        aria-label="Spreadsheet"
        onBlurCapture={handleCellChange}     // Step 6
        onFocusCapture={handleCellFocus}     // Step 5
      >
        {HEADER_ROWS}
        {BODY_ROWS}
      </div>
    </section>
  )
}
```

Handle focus event to show raw formula

```
const handleCellFocus: React.FocusEventHandler<HTMLDivElement> = ({ target }) => {
  if (target instanceof HTMLElement) {
    const column = target.dataset.column as TTableColumn
    const row = Number(target.dataset.row)
    const id = toCellReference(row, column)
    if (column && row) {
      target.textContent = engine.getRaw(id)   // show raw formula
    }
  }
}
```

Handle blur event to show raw formula

```
const updateCellView = (id: CellId) => {
  const { col, row } = fromCellReference(id)
  if (!col || !row) return
  const cell = document.querySelector(`[data-column="${col}"][data-row="${row}"]`)
  if (cell) {
    if (document.activeElement === cell) {
      cell.textContent = engine.getRaw(id)
    } else {
      cell.textContent = engine.getValue(id)
    }
  }
}

const handleCellChange: React.FocusEventHandler<HTMLDivElement> = ({ target }) => {
  if (target instanceof HTMLElement) {
    const column = target.dataset.column as TTableColumn
    const row = Number(target.dataset.row)
    const id = toCellReference(row, column)
    if (column && row) {
      const raw = target.textContent ?? ''
      const { changed } = engine.setRaw(id, raw)    // commit to engine

      target.textContent = engine.getValue(id)       // show computed value

      for (const changedId of changed) {             // update dependents
        updateCellView(changedId)
      }
    }
  }
}
```

Congratulations - Part 2.2 Completed

Workshop Summary