

Behind the Scenes

Exploring React's Source Code

Katja Pericin

```

cloc --exclude-dir=node_modules,__tests__ packages/
      1132 text files.
      1102 unique files.
      108 files ignored.

github.com/AlDanial/cloc v 1.96  T=0.67 s (1634.7 files/s, 257882.2 lines/s)
-----
Language           files    blank   comment    code
-----
JavaScript        935     15423    24319    125719
CSS               68      358       22      2489
Markdown          35      991       1      2342
JSON              40       0       0      1359
HTML              19      40       8      708
YAML              1       11      10       37
SVG               4       0       0       4
-----
SUM:             1102    16823    24360    132658
-----

```

Number of lines	Average	Min	Max
Packages	211	5	5180 🤯
react	131	14	573
react-dom	171	26	744
react-reconciler	419	10	4645

```
React.__SECRET_INTERNALS_DO_NOT_USE_OR_YOU_WILL_BE_FIRED
```

```
error = typeSpecs[typeSpecName](  
  values,  
  typeSpecName,  
  componentName,  
  location,  
  null,  
  'SECRET_DO_NOT_PASS_THIS_OR_YOU_WILL_BE_FIRED',  
);
```

```
function updateState<S>(  
  initialState: (() => S) | S,  
): [S, Dispatch<BasicStateAction<S>>] {  
  return updateReducer(basicStateReducer, (initialState: any));  
}
```

```
export function trapClickOnNonInteractiveElement(node: HTMLElement) {  
  // Mobile Safari does not fire properly bubble click events on  
  // non-interactive elements, which means delegated click listeners do not  
  // fire. The workaround for this bug involves attaching an empty click  
  // listener on the target node.  
  // https://www.quirksmode.org/blog/archives/2010/09/click_event_del.html  
  // Just set it using the onclick property so that we don't have to manage any  
  // bookkeeping for it. Not sure if we need to clear it when the listener is  
  // removed.  
  // TODO: Only do this for the relevant Safaris maybe?  
  node.onclick = noop;  
}
```

React Fiber

- React reconciliation algorithm (introduced in v16)
- Splits render and commit phases for better performance
 - Render phase builds fiber tree and computes changes
 - Commit phase updates DOM

Fiber data structure

```
FiberNode {  
  tag,  
  flags,  
  elementType,  
  memoizedProps,  
  memoizedState,  
  pendingProps,  
  child,  
  sibling,  
  return,  
  stateNode,  
  updateQueue,  
  lanes,  
  ...  
}
```

```
export const FunctionComponent = 0;  
export const ClassComponent = 1;  
export const IndeterminateComponent = 2; // Before we know whether it is function or class  
export const HostRoot = 3; // Root of a host tree. Could be nested inside another node.  
export const HostPortal = 4; // A subtree. Could be an entry point to a different renderer.  
export const HostComponent = 5;  
export const HostText = 6;  
export const Fragment = 7;  
export const Mode = 8;  
export const ContextConsumer = 9;  
export const ContextProvider = 10;  
export const ForwardRef = 11;  
export const Profiler = 12;  
export const SuspenseComponent = 13;  
export const MemoComponent = 14;  
export const SimpleMemoComponent = 15;  
export const LazyComponent = 16;  
export const IncompleteClassComponent = 17;  
export const DehydratedFragment = 18;  
export const SuspenseListComponent = 19;  
export const ScopeComponent = 21;  
export const OffscreenComponent = 22;  
export const LegacyHiddenComponent = 23;  
export const CacheComponent = 24;  
export const TracingMarkerComponent = 25;  
export const HostHoistable = 26;  
export const HostSingleton = 27;
```

Tags

<https://5j42r6.csb.app/>

Base Counter

0

Increment

Reset All

Counter 1

0

Multiplier: 

Counter 2

0

Multiplier: 

Counter 3

0

Multiplier: 

HostRoot

Base Counter

0

Increment **Reset All**

Counter 1

0

Multiplier: ^

Counter 2

0

Multiplier: ^

Counter 3

0

Multiplier: ^

App

Counter 1

0

Multiplier: 

Counter 2

0

Multiplier: 

Counter 3

0

Multiplier: 

Base Counter

0

Increment

Reset All

Base Counter

Base Counter

0

Increment

Reset All

Counter 1

0

Multiplier: 

Counter 2

0

Multiplier: 

Counter 3

0

Multiplier: 

Base Counter

0

Increment

Reset All

Multiple Counters

Counter 1

0

Multiplier:

2



Counter 2

0

Multiplier:

3



Counter 3

0

Multiplier:

4



Base Counter

0

Increment

Reset All

Counter 1

Counter 1

0

Multiplier: 2



Counter 2

0

Multiplier: 3



Counter 3

0

Multiplier: 4



Base Counter

0

Increment

Reset All

Counter 1

0

Multiplier: 

Counter 2

Counter 2

0

Multiplier: 

Counter 3

0

Multiplier: 

Base Counter

0

Increment

Reset All

Counter 1

0

Multiplier: ^

Counter 2

0

Multiplier: ^

Counter 3

Counter 3

0

Multiplier: ^

Child →
Return →
Sibling →

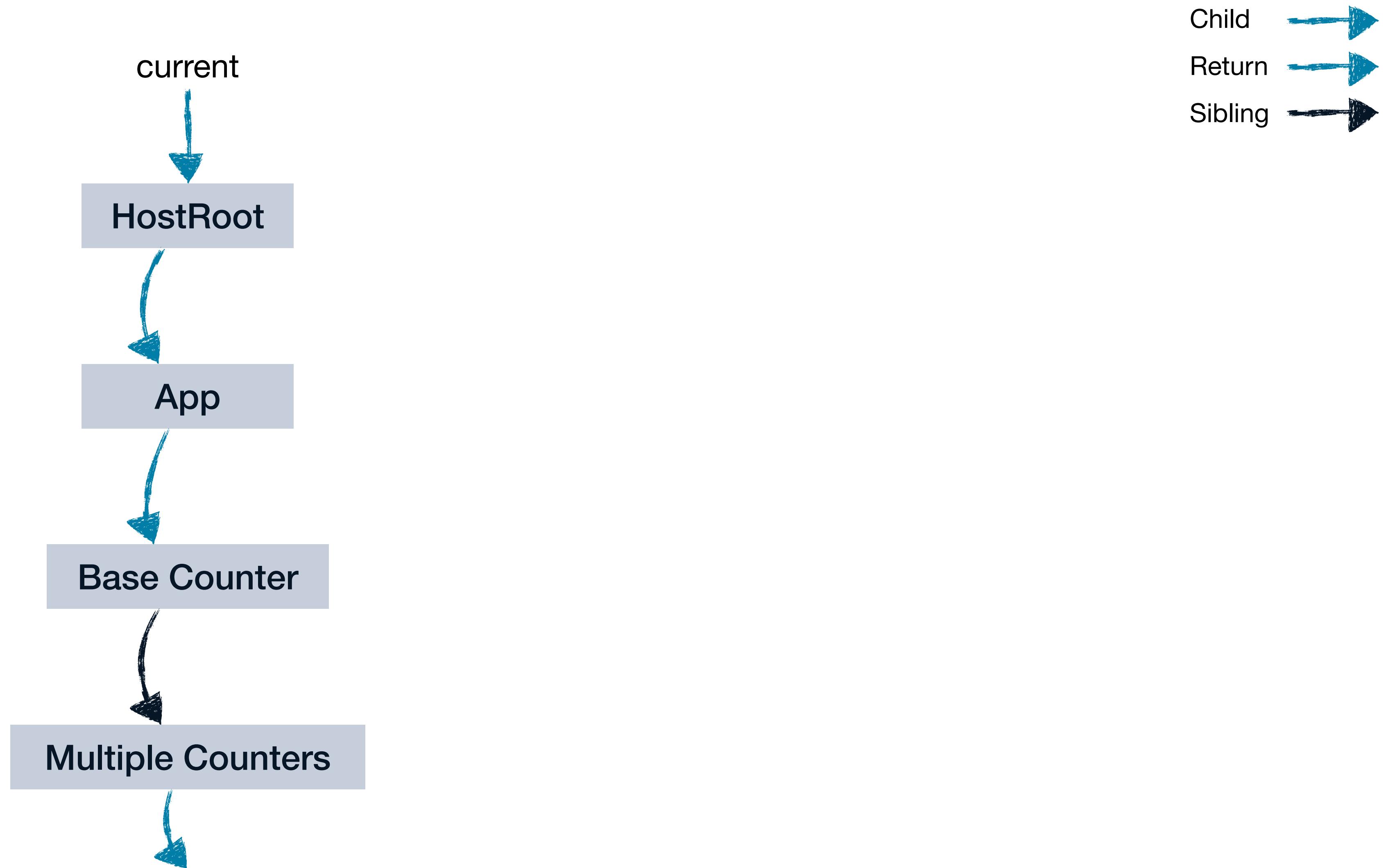
current

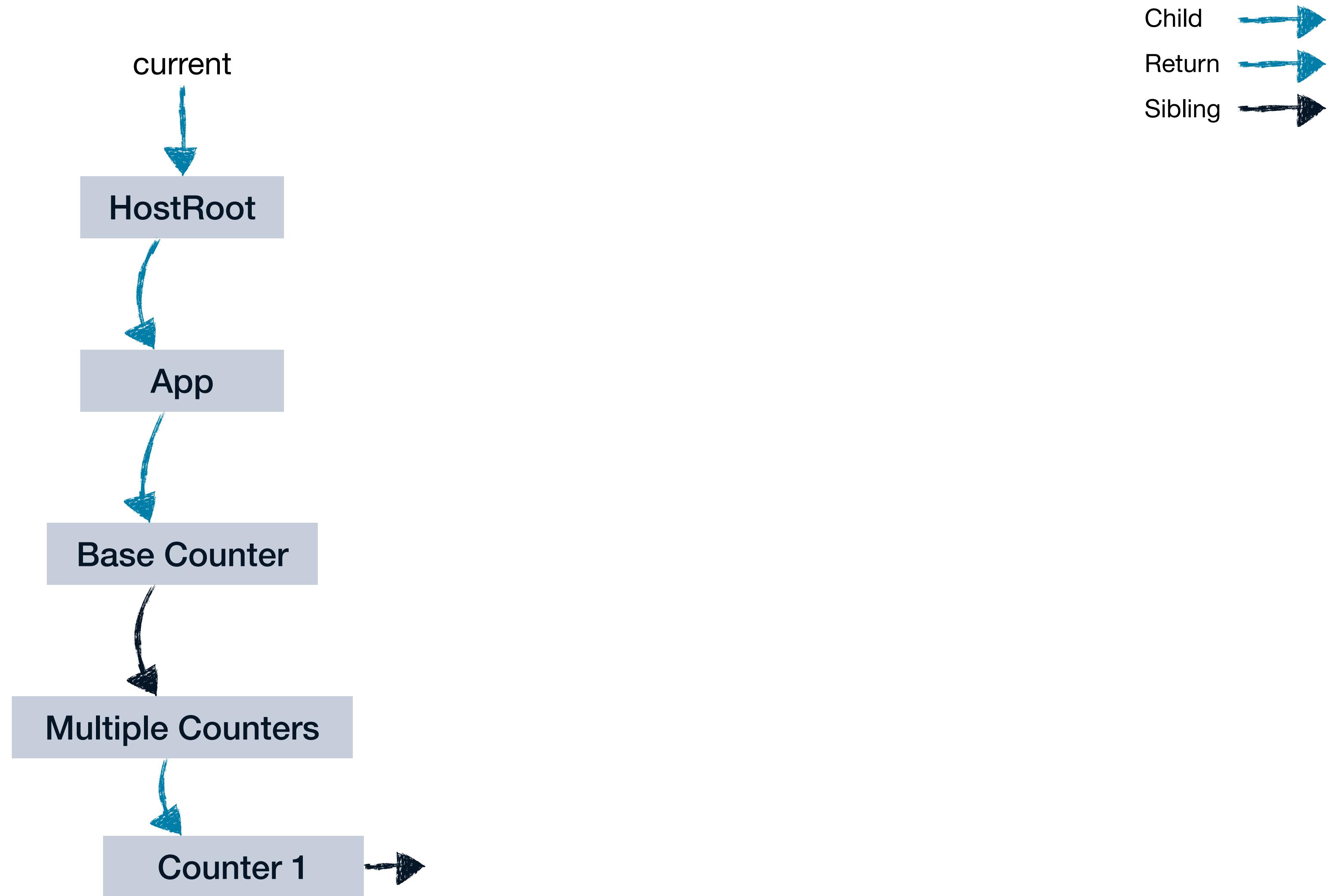


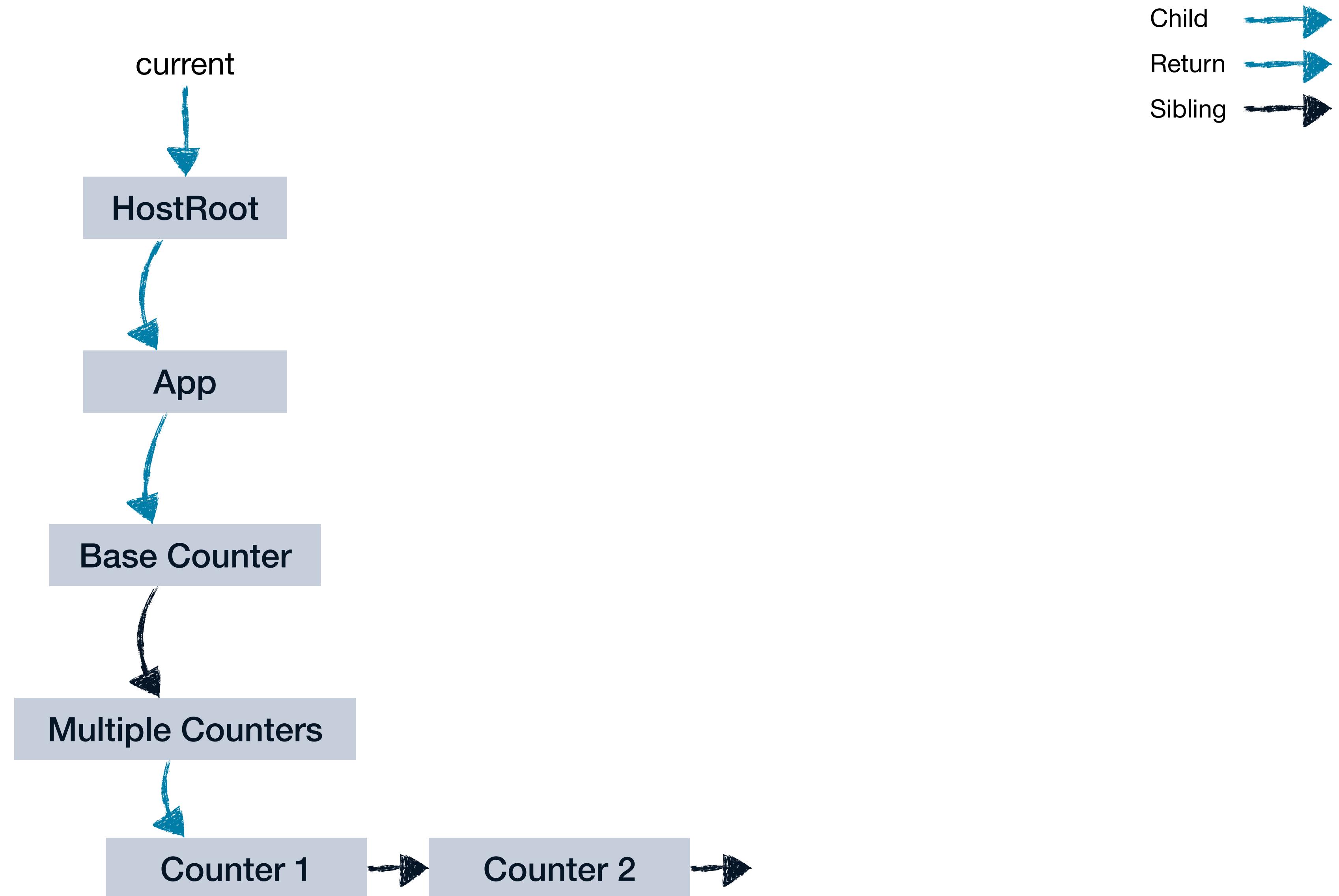


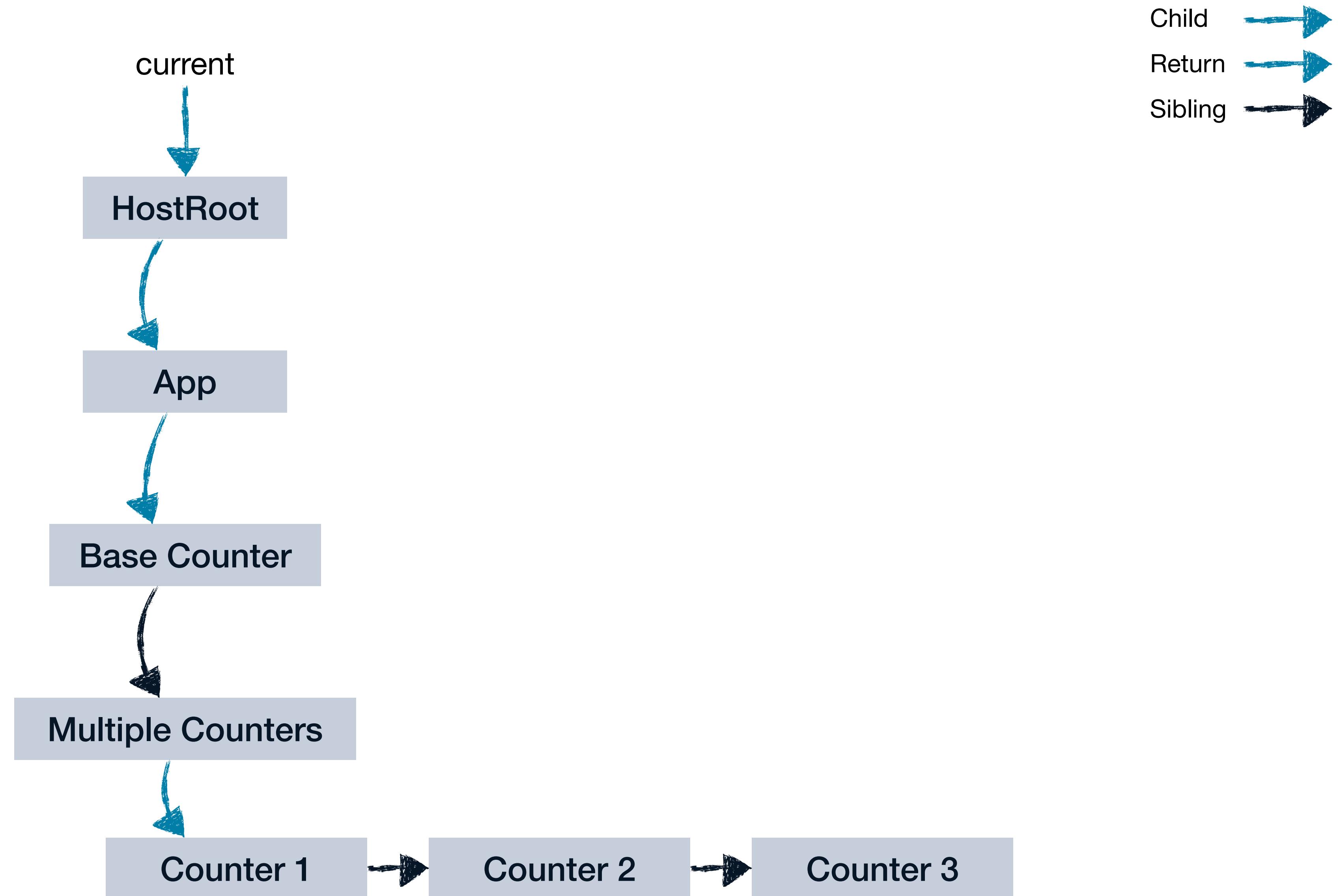


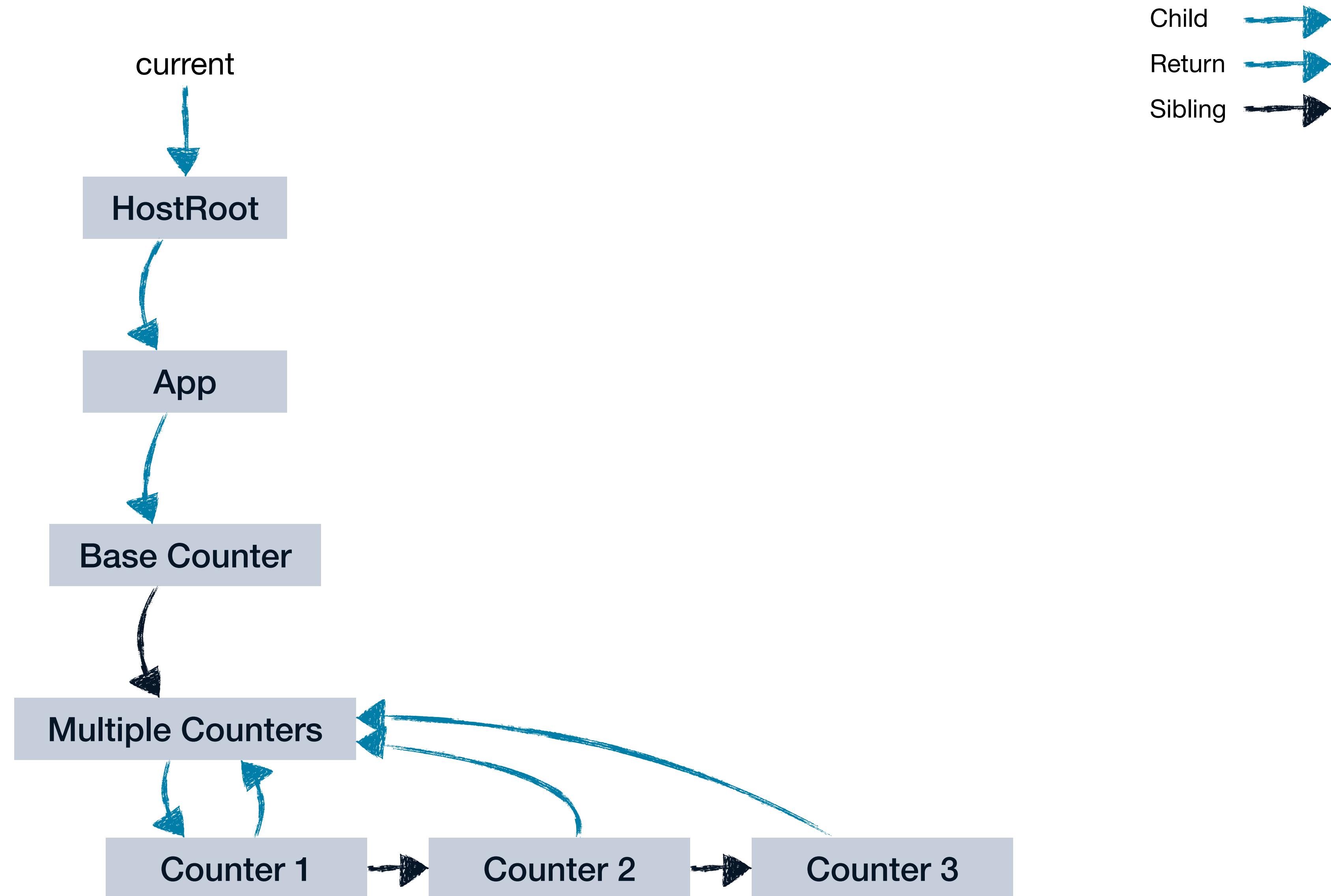


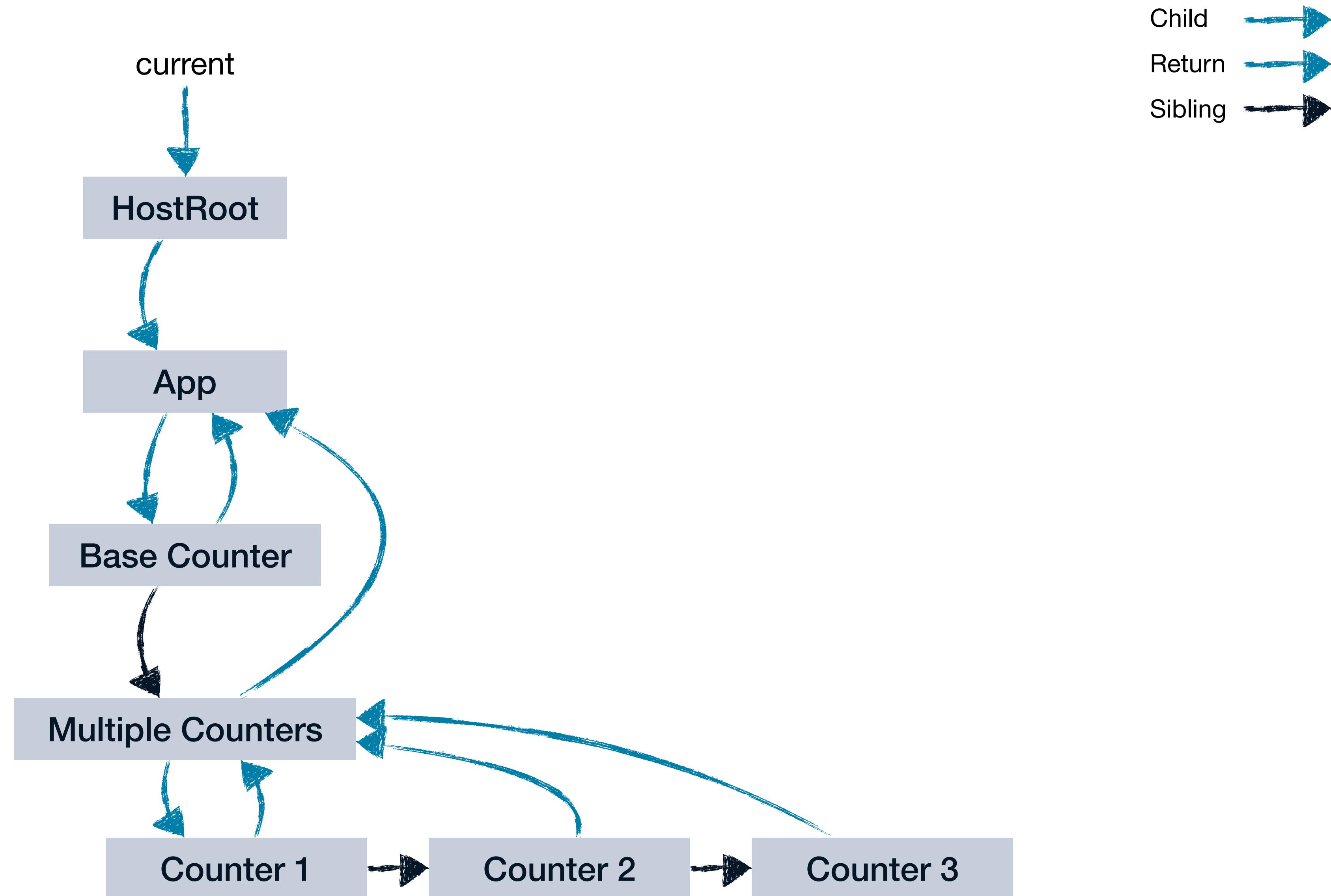


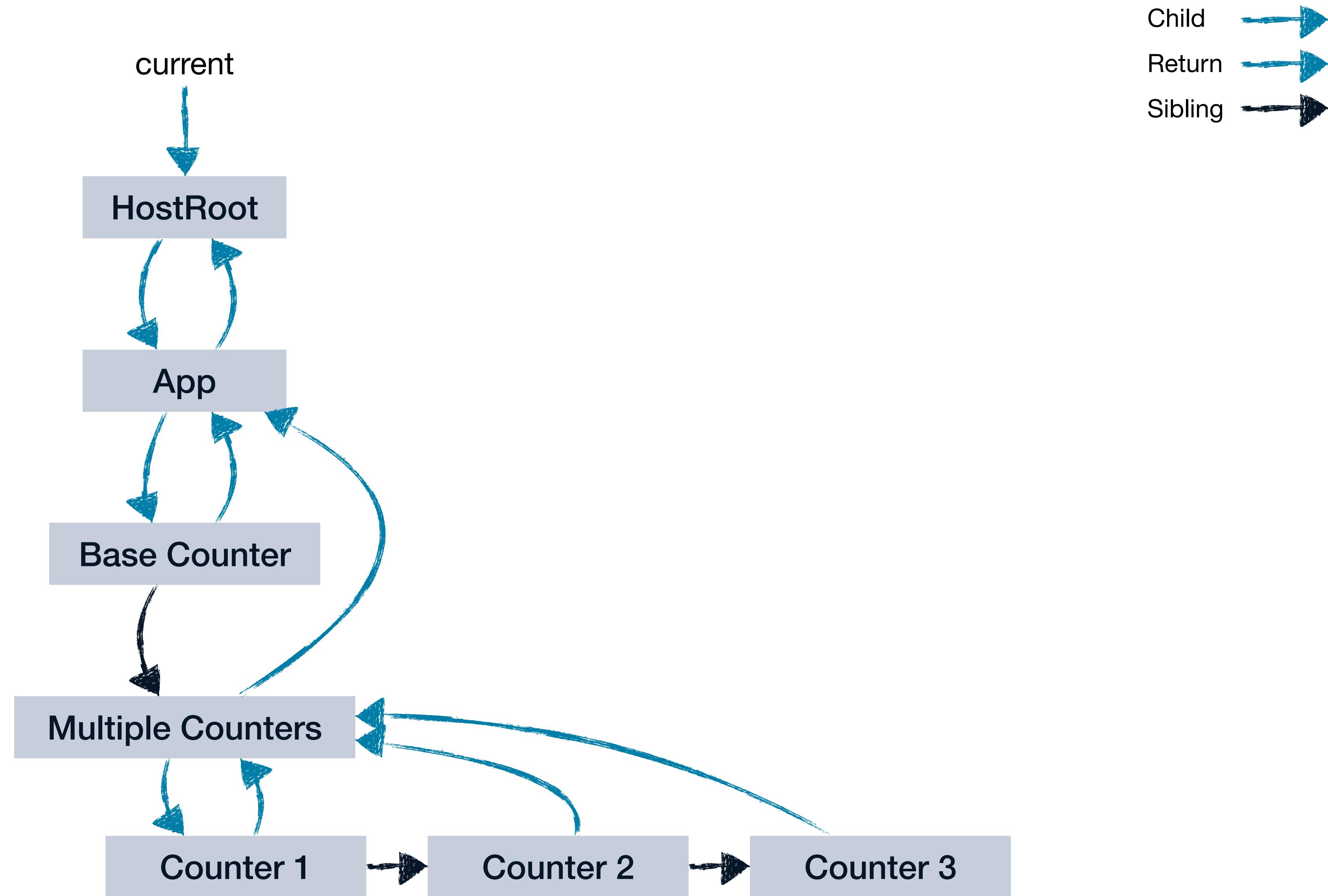












Base Counter

0

Increment

Reset All

Counter 1

0

Multiplier: 2

Counter 2

0

Multiplier: 3

Counter 3

0

Multiplier: 4

workInProgress



Render Phase

`workInProgress`



`HostRoot`



Render Phase

`workInProgress`



HostRoot



App

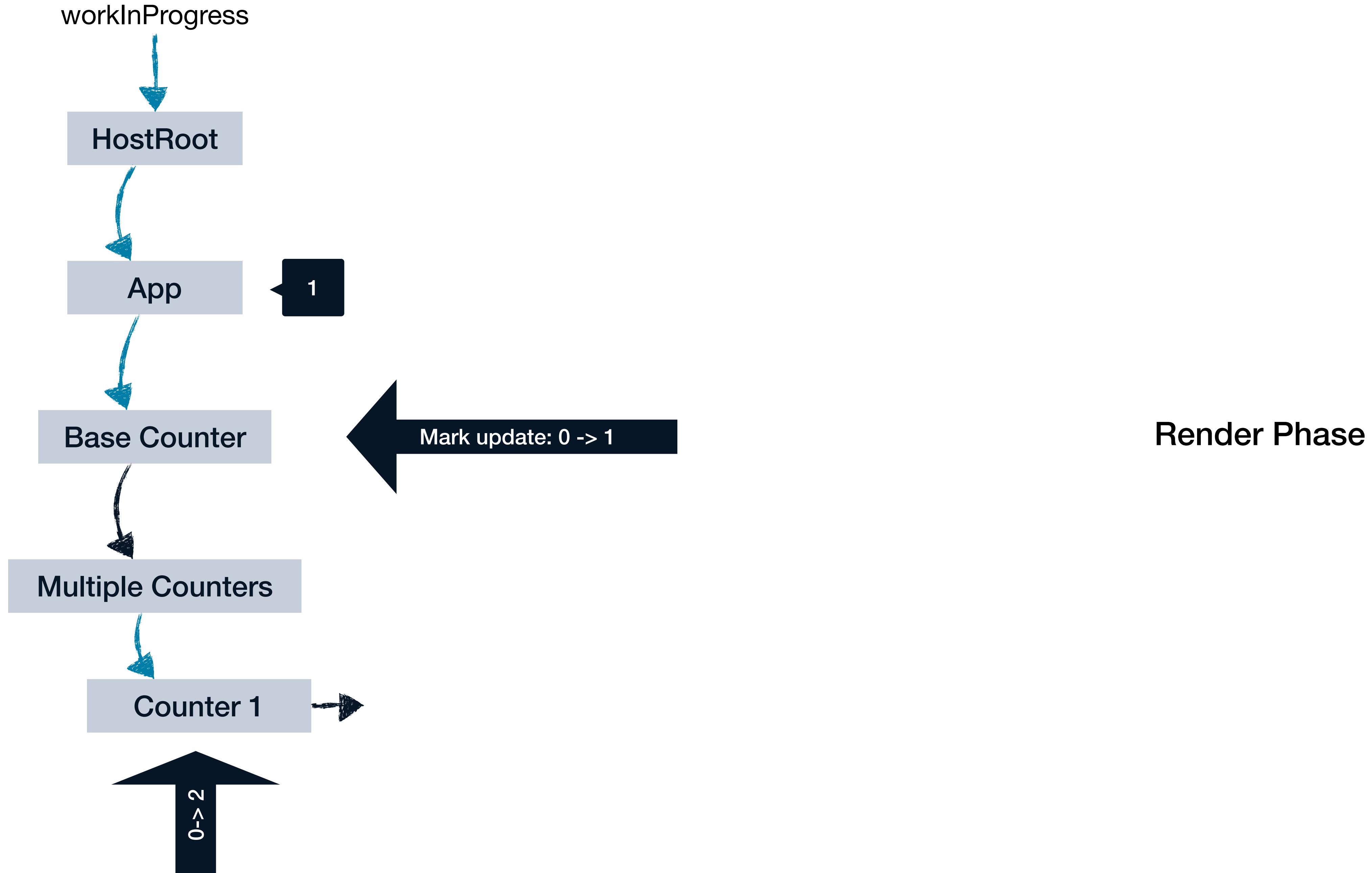
1

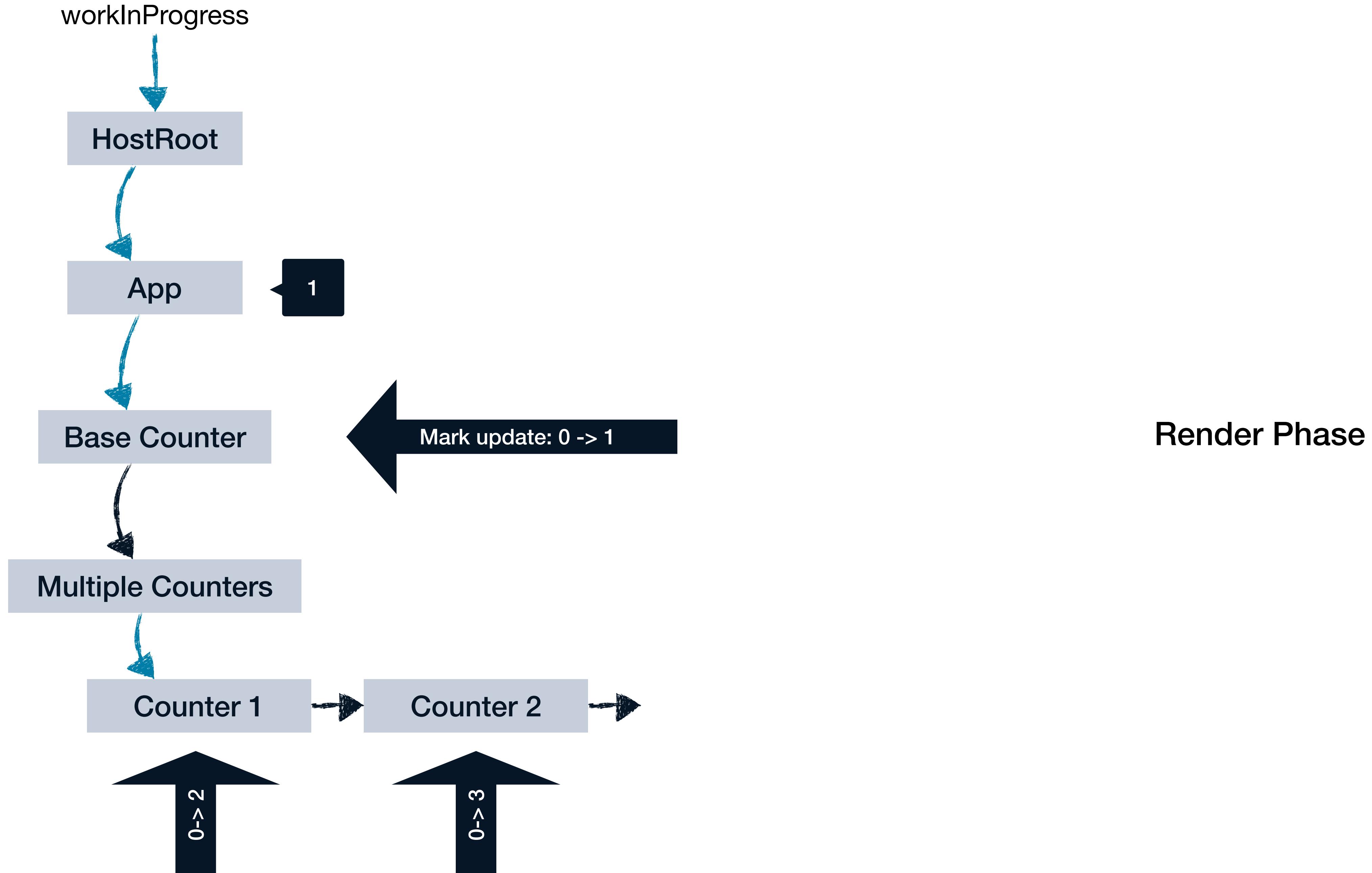


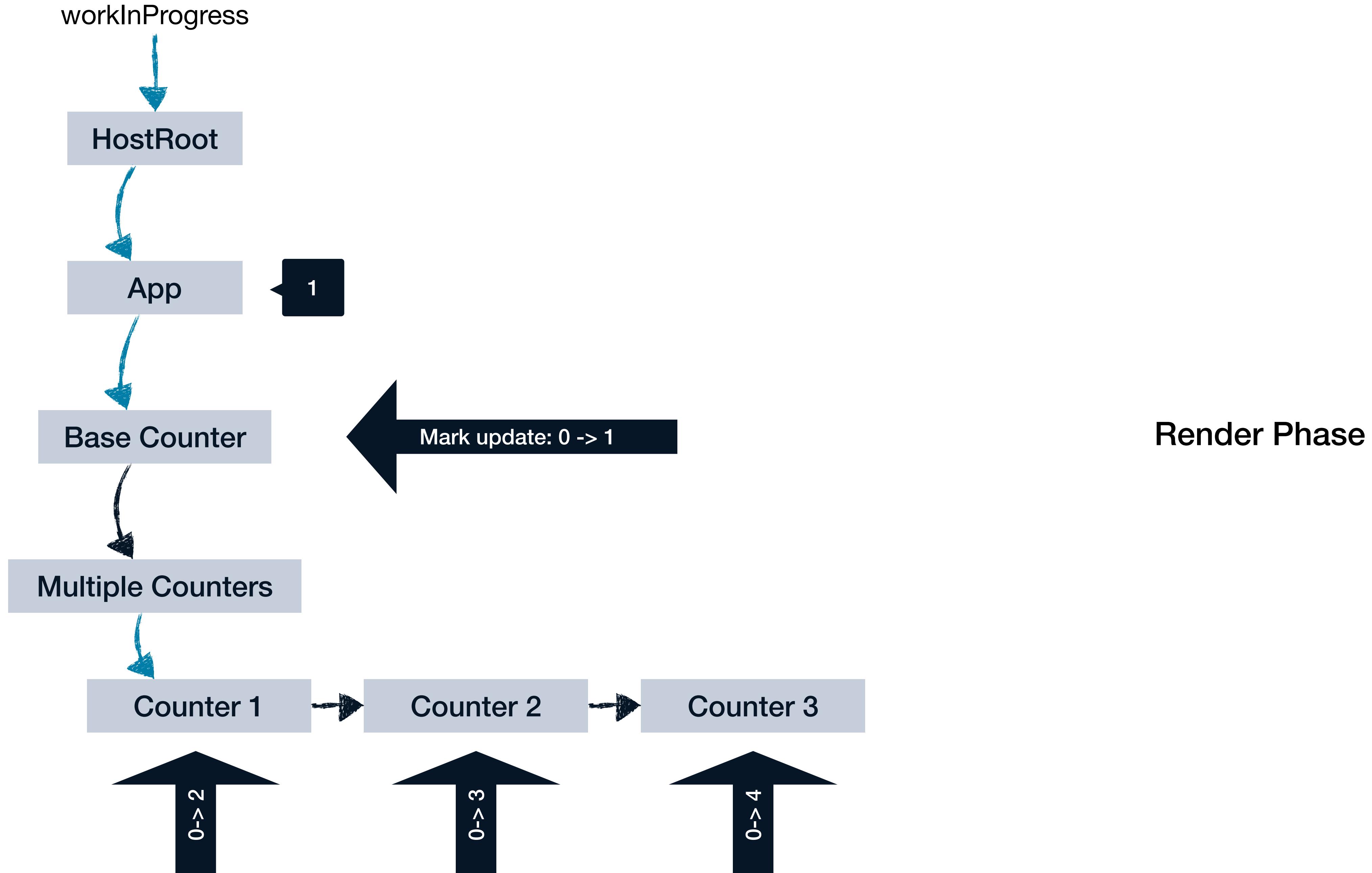
Render Phase

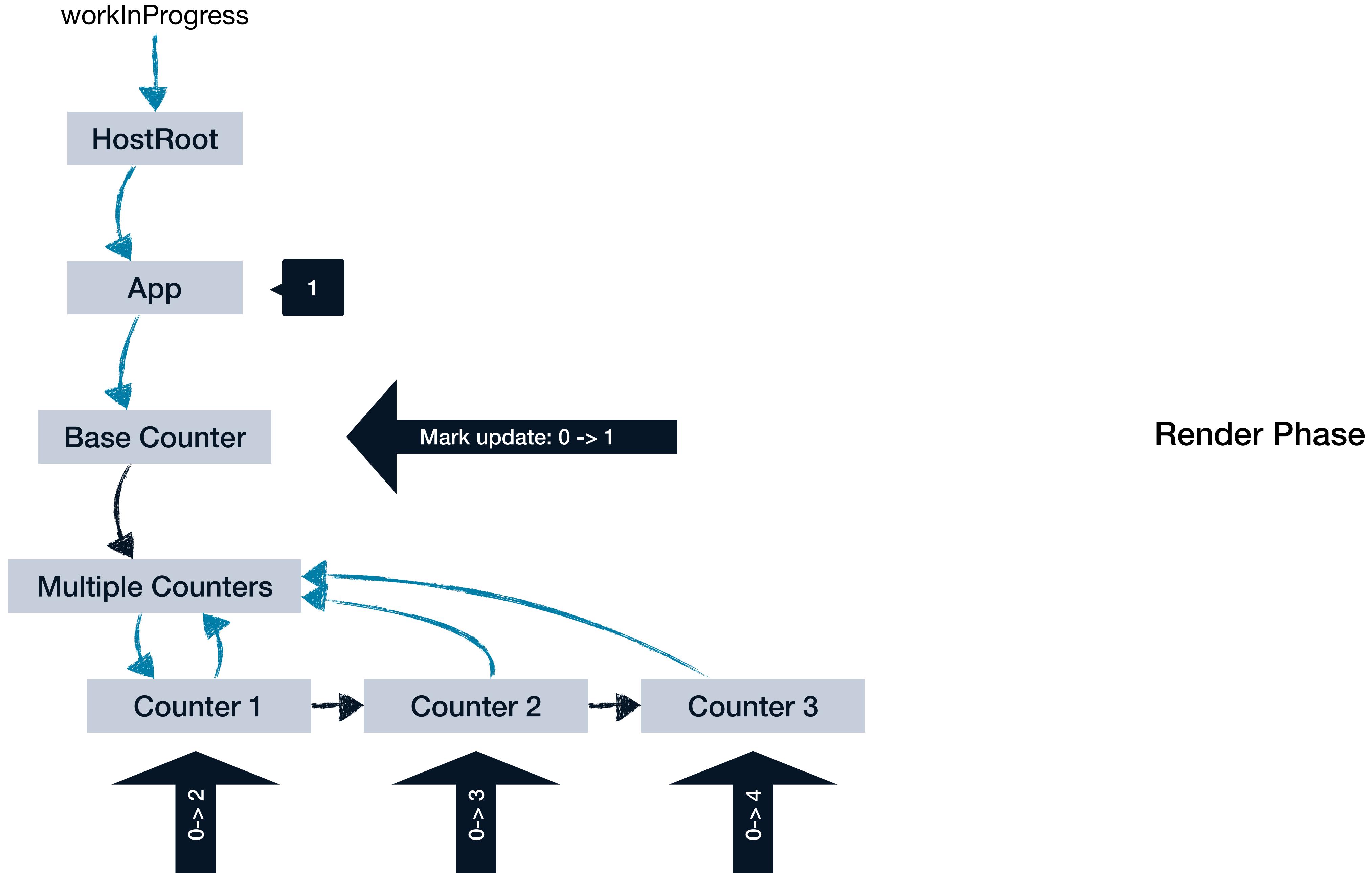


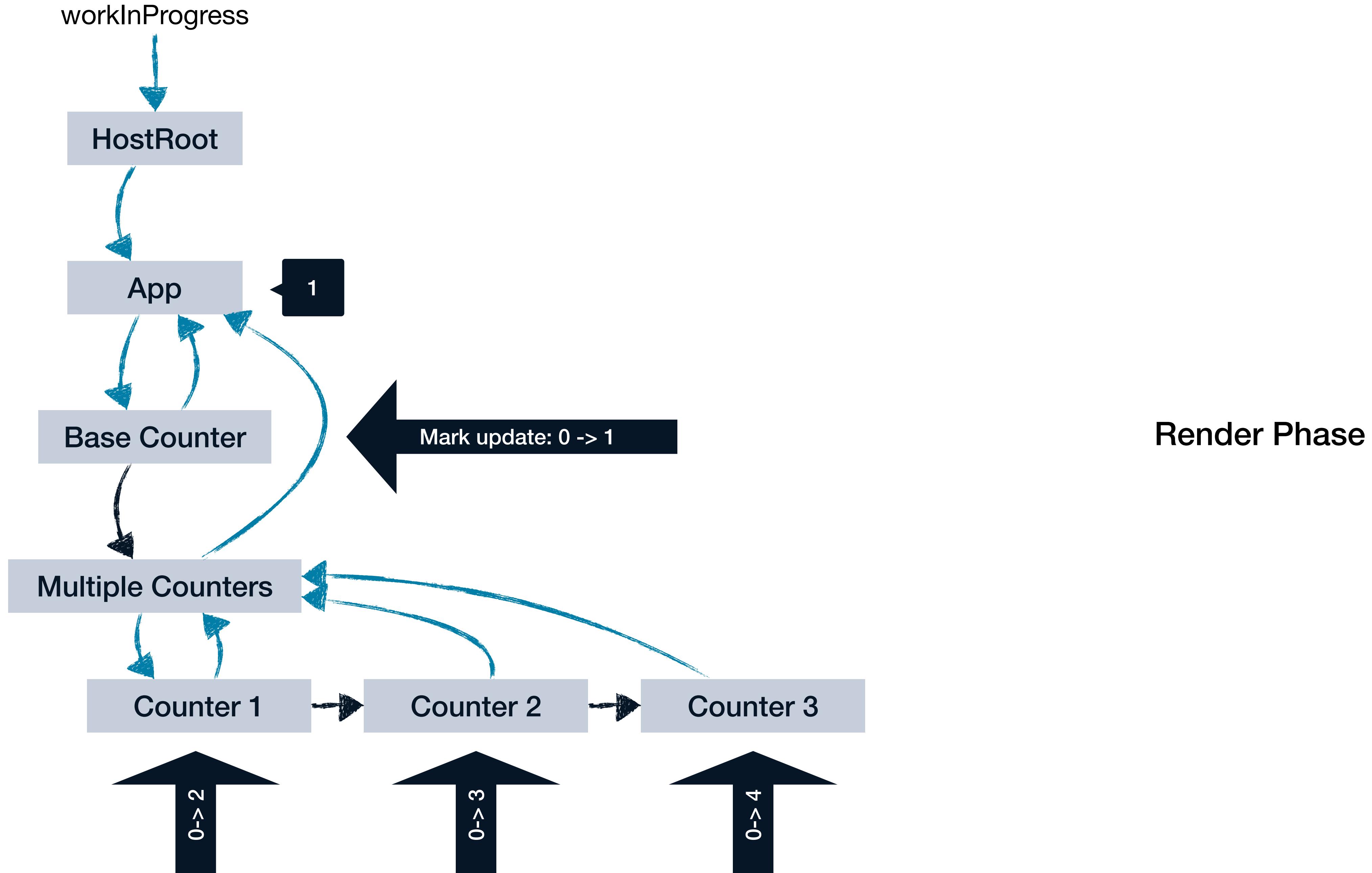


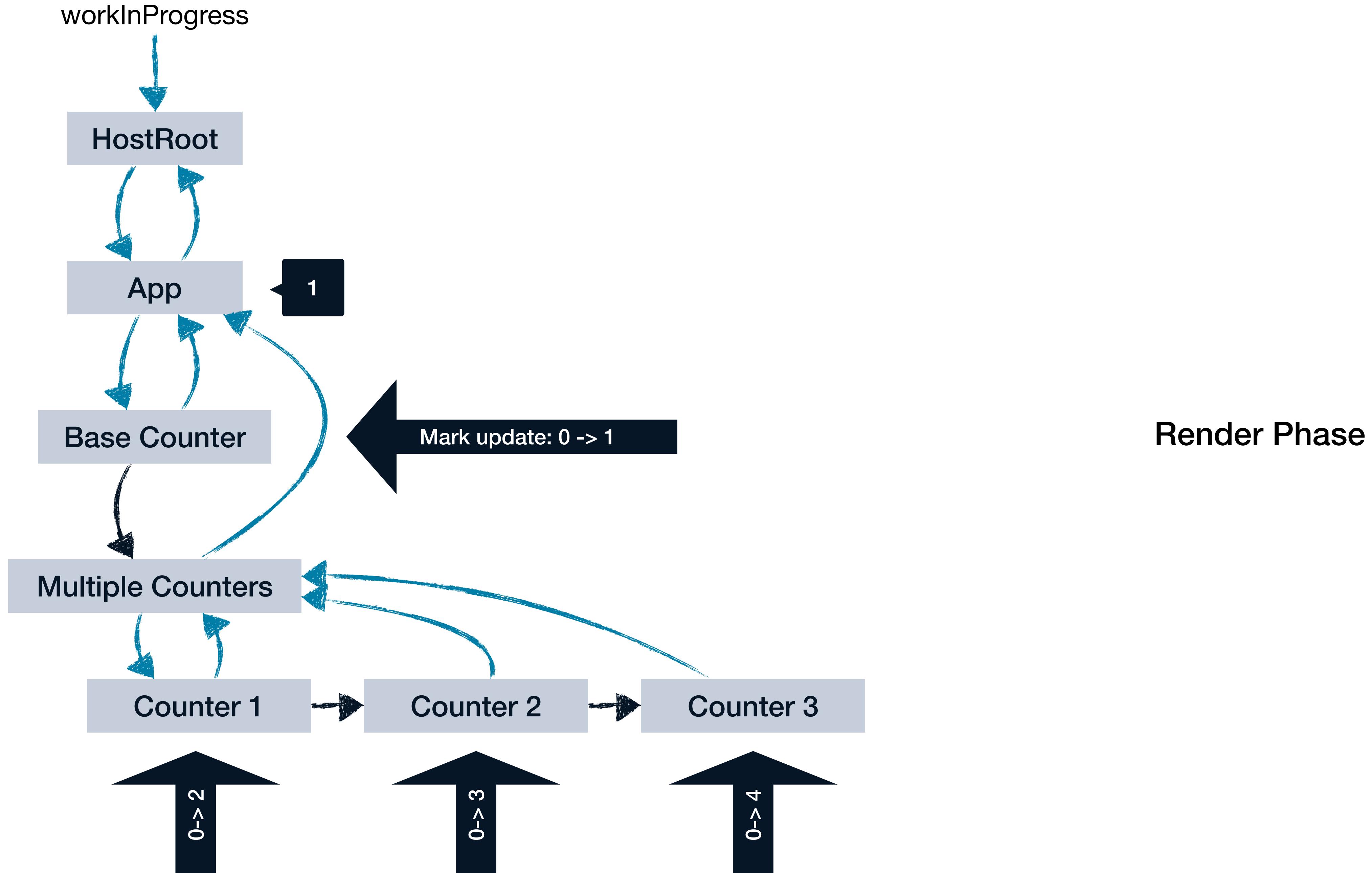


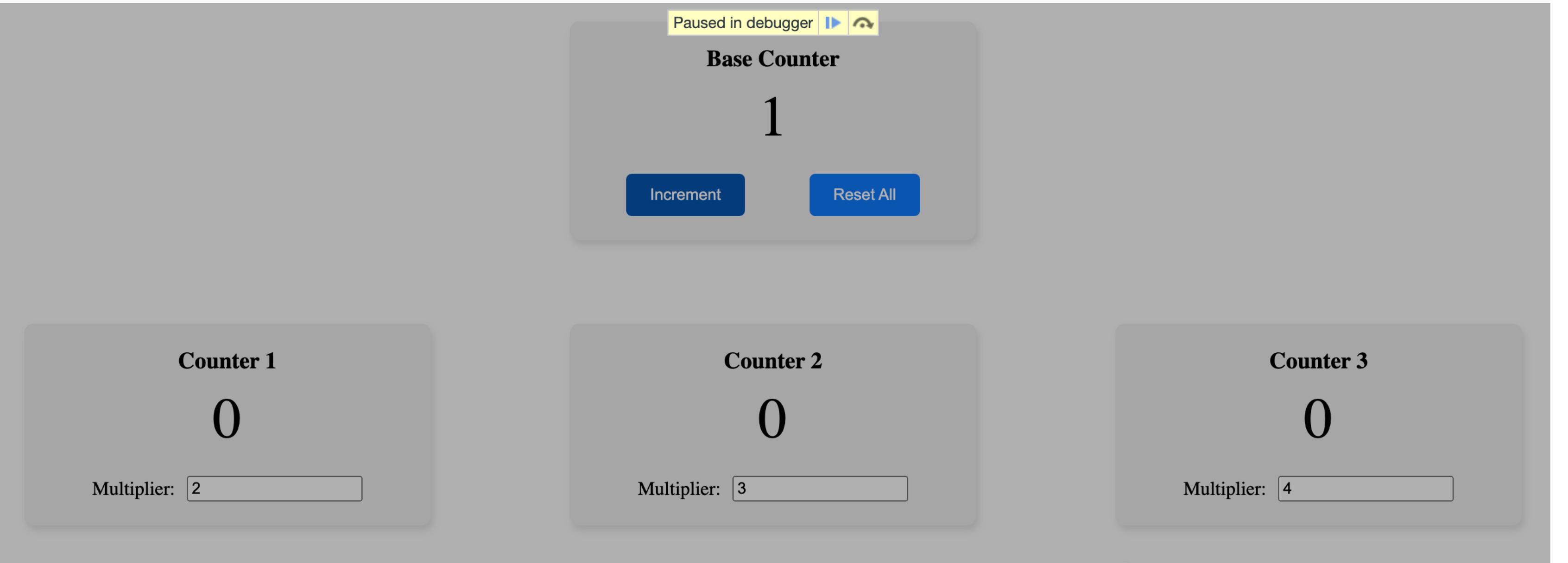












Commit phase - first DOM change

Paused in debugger | ⏪ | ⏴

Base Counter

1

Increment Reset All

Counter 1

0

Multiplier: 2

Counter 2

0

Multiplier: 3

Counter 3

0

Multiplier: 4

Commit phase - first DOM change

Paused in debugger | ⏪ | ⏴

Base Counter

1

Increment Reset All

Counter 1

2

Multiplier: 2

Counter 2

0

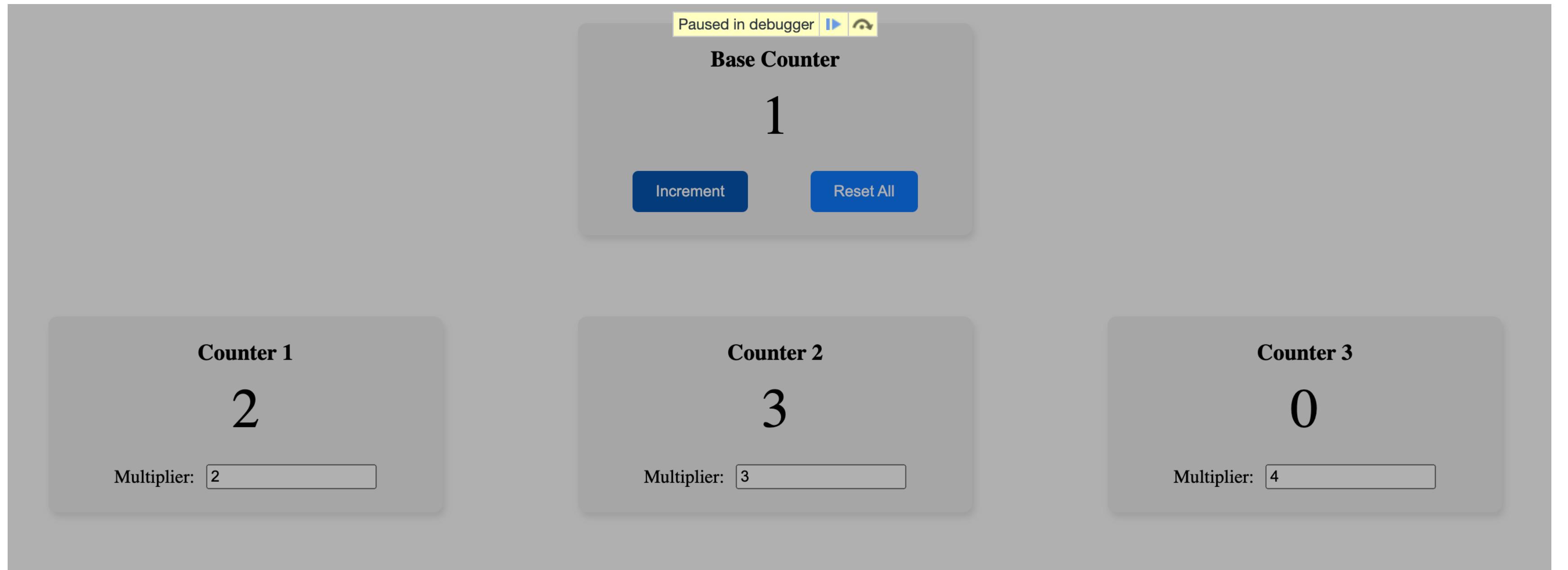
Multiplier: 3

Counter 3

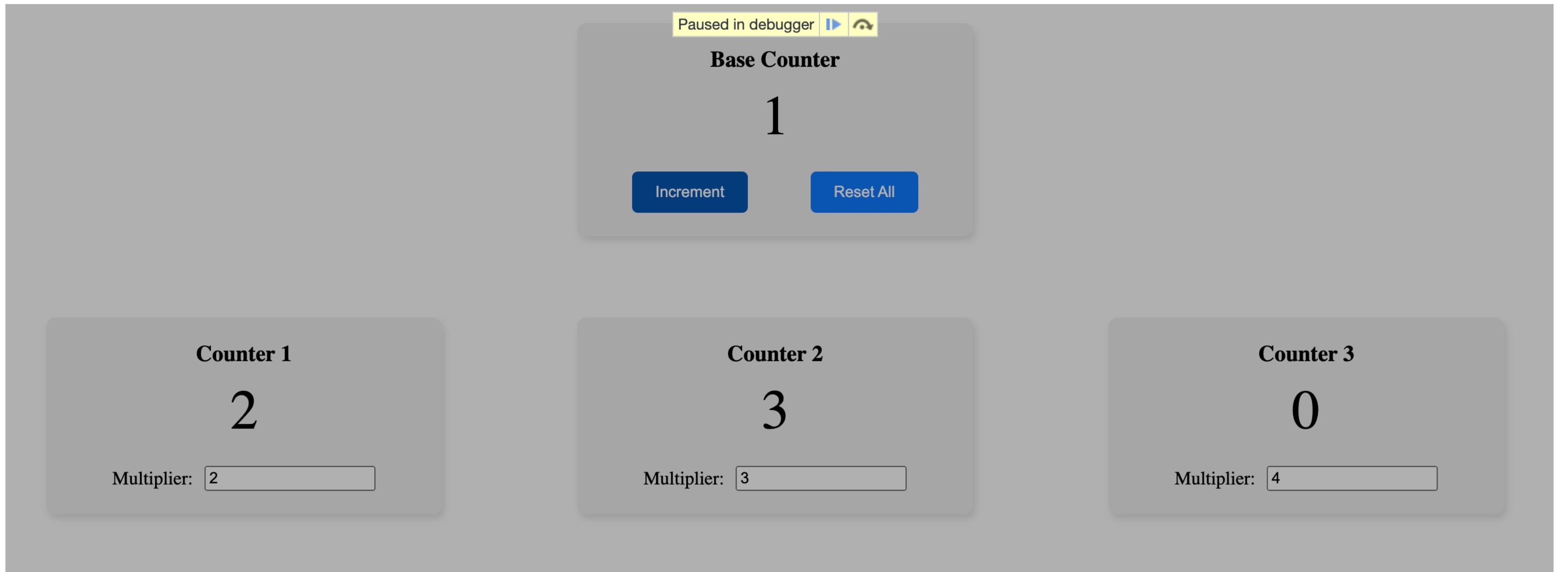
0

Multiplier: 4

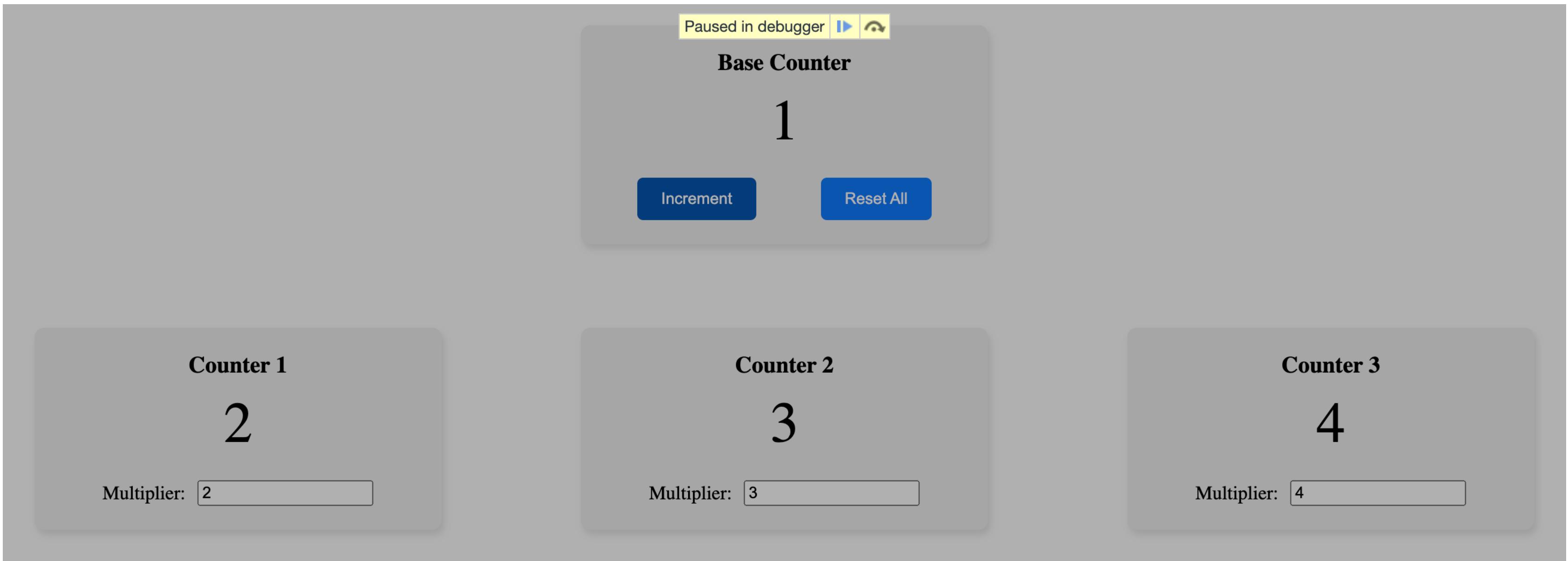
Commit phase - second DOM change



Commit phase - third DOM change



Commit phase - fourth DOM change



Summary

- When mounting for the first time, React builds a linked list of Fiber nodes called Fiber tree
- Each node has links to parent node (return), children node (child) and sibling nodes (sibling)
- Starting from hostRoot, tree is traversed (depth first)
- Each node is processed as a single unit of work
- If there is time to continue (single frame not yet done), React continues with the next unit of work until the tree is traversed or time runs out
- In commit phase, same tree is traversed. Nodes marked with Update flag are committed to the DOM

There is something I need to
admit

There is something I need to
admit

This presentation had a hidden helper, ChatGPT-4

Original Idea

ChatGPT-4 will write the talk, I'll check all facts for correctness

Reality

ChatGPT is worse than expected. Checking correctness ended up being most of the work thus it barely helped with this presentation

Things ChatGPT does well

- Brainstorming
 - It is pretty good with generating random ideas. It won't solve the problem, but it can generate enough to point in the new direction
- ChatGPT will provide answers much quicker than googling or browsing stack overflow
 - When problem is very specific and has specific answer
 - When library/tool is poorly documented
- Providing code for problems that have been solved over and over
 - Things like ToDo lists, hello worlds and other how to projects

Things ChatGPT does well

- Writing tests and boilerplate code
 - ChatGPT can save a lot of time with boilerplate code
 - It is pretty decent when writing tests, although it should not be blindly trusted
- Explaining code
 - Pasting code to ChatGPT can result with helpful explanations. It does do better if code is commented. However, explanations should be double checked
- Providing sources
 - Even if generated answer is wrong, sources are helpful and can point in the right direction

Things ChatGPT does not do well

- Generating more complex code
 - If prompt is complex, ChatGPT will generate code. However, sometimes explanation and the code will not match
- Generating “new” code
 - If it is asked to create something that’s not usually done, it tends to generate code which doesn’t work
 - However, if errors are pointed out, occasionally it can fix it

Examples



Give me 5 ideas for a frontend meetup talk which covers react in depth



1. "Demystifying React Performance: Optimizing for Speed and Efficiency"

Dive into the inner workings of React, exploring performance bottlenecks, common optimization techniques, and best practices for building fast, efficient applications. Cover topics such as virtual DOM, diffing algorithms, `React.memo`, and using the React Profiler to identify and resolve performance issues.

2. "React Hooks: Unlocking Their Full Potential"

Take a comprehensive look at React Hooks, discussing their introduction, benefits, and use cases. Share real-world examples to demonstrate the power of Hooks, and cover advanced use cases such as creating custom hooks, combining hooks, and managing complex state logic with the `useReducer` hook.

3. "React and TypeScript: A Match Made in Heaven"

Show how TypeScript can improve the React development experience by adding type safety and enhancing the developer experience. Discuss the benefits of using TypeScript with React, how to set up a project, and best practices for creating type-safe components, handling props, and managing state.

4. "Design Systems and Component Libraries: Streamlining React Development"

Explore the concept of design systems and their importance in maintaining consistency and efficiency in large-scale applications. Discuss popular component libraries like Material-UI, Ant Design, and Chakra UI, and guide attendees through the process of creating a custom component library with React and a CSS-in-JS solution like styled-components or Emotion.

5. "Advanced State Management with React: Context API, Redux, and Beyond"

Investigate various state management solutions for React applications, starting with React's built-in Context API and moving on to popular libraries like Redux and MobX.

Compare their pros and cons, and provide examples to help attendees choose the right solution for their specific use cases.

Send a message.



Focus on react itself, not on applications of react



1. "The Evolution of React: A Deep Dive into the History and Core Concepts"

Trace the history of React, from its creation at Facebook to its rise as a dominant frontend library. Explore the core concepts that make React unique, such as components, JSX, unidirectional data flow, and the virtual DOM. Discuss how these concepts have evolved over time and their impact on frontend development.

2. "Inside React: Understanding the Reconciliation Algorithm and the Virtual DOM"

Delve into the inner workings of React's reconciliation algorithm and the virtual DOM. Explain how React efficiently updates the actual DOM by creating a lightweight representation of the UI, comparing it to the current virtual DOM, and making the minimum number of updates necessary. Discuss the benefits of this approach and how it influences application performance.

3. "React Component Patterns: Function vs. Class Components, Higher-Order Components, and Render Props"

Examine various React component patterns and their implications for application design. Compare function and class components, discuss the introduction of hooks, and explore the use of higher-order components and render props for code reuse and composition. Provide practical examples and best practices for implementing these patterns in real-world applications.

4. "React's Concurrent Mode: Unlocking Smoother User Experiences"

Investigate React's experimental Concurrent Mode, which aims to improve the user experience by allowing the rendering engine to work on multiple tasks simultaneously. Explain how Concurrent Mode enables smoother animations, improved responsiveness, and better resource management. Discuss the implications of this feature and its potential impact on the React ecosystem.

5. "React Internals: Understanding the Library Functions Under the Hood"

Take a deep dive into React's source code, demystifying how the library functions under the hood.

Send a message.





Write me slides for idea 5



Slide 1: Title

"React Internals: Understanding React's Source Code for Better Development"

Slide 2: Introduction

- Why understanding React internals matters
- How it can improve development practices and problem-solving skills
- Overview of key topics to be covered

Slide 3: React's Architecture

- Overview of React's architecture
- The role of the Fiber architecture
- How React components are represented as fibers

Slide 4: The Fiber Architecture

- What is a fiber?
- Structure of a fiber node
- The Fiber tree and the work loop

Slide 5: The Scheduler

- Purpose of the scheduler

⟳ Regenerate response

Send a message.



Slide 5: The Scheduler

- Purpose of the scheduler
- Scheduling priorities and tasks
- React's cooperative multitasking approach

Slide 6: The Reconciliation Process

- Overview of the reconciliation process
- The role of the virtual DOM
- The diffing algorithm and its significance

Slide 7: The Event System

- React's synthetic event system
- Event delegation and event pooling
- Benefits of using a synthetic event system

Slide 8: Component Lifecycle

- Overview of React component lifecycle
- Mounting, updating, and unmounting phases
- Effects of understanding lifecycle methods on component optimization

Slide 9: Error Boundaries and Error Handling

- Understanding error boundaries in React
- How error boundaries catch an error and handle it
- Best practices for implementing error boundaries

Send a message.





Title: "Behind the Scenes: Exploring React's Source Code"



Abstract:

Whether you're a frontend developer with React experience or have yet to explore this popular library, this talk is for you! We'll dive into the inner workings of React's source code, unveiling the core mechanisms and design choices that make it a powerful tool for building user interfaces. This session is perfect for developers who are eager to explore React's internals and deepen their frontend knowledge.

Throughout the presentation, we will examine essential concepts and techniques used within the React source code, such as the virtual DOM, the reconciliation algorithm, and the event system. We'll also delve into the implementation details of React hooks, including useState, useEffect, and useReducer, providing valuable insights for both newcomers and experienced React developers.

Our goal is to equip attendees with a comprehensive understanding of React's inner workings, empowering them to make informed decisions when building applications. By taking a closer look at React's core, developers will acquire valuable insights that can enhance their development process and lead to more efficient, maintainable, and performant applications.

Join us on this enlightening journey as we explore React's source code, shedding light on its inner workings, and providing frontend developers of all levels with the knowledge to confidently approach and excel with this powerful library.



Can you shorten it a bit? Focus more on technical details it covers, less on selling it



Title: "Behind the Scenes: Exploring React's Source Code"



Abstract:

This talk is designed for frontend developers, both experienced with React and newcomers, who want to dive deep into React's source code to understand its core mechanisms and design choices. We'll cover essential concepts and techniques, such as the virtual DOM, the reconciliation algorithm, and the event system, while also examining the implementation details of React hooks, including useState, useEffect, and useReducer.

The goal of this session is to provide a solid understanding of React's internals, enabling attendees to build more efficient, maintainable, and performant applications. Join us as we explore React's source code and uncover the technical details that make this library a powerful tool for frontend development.

Optimism Level





To demonstrate all of the internals of react, I need an example. Write me a small app which has a counter. Counter can be incremented by 1 or reset back to 0. There is also an option to increment by user specified value after an user specified delay in seconds.



Here's a simple React app that fulfills these requirements. We'll use `useState` and `useEffect` hooks to manage state and side effects, respectively.

```
jsx Copy code
import React, { useState, useEffect } from 'react';

const CounterApp = () => {
  const [counter, setCounter] = useState(0);
  const [incrementBy, setIncrementBy] = useState(1);
  const [delay, setDelay] = useState(0);

  const handleIncrement = () => {
    setCounter(prevCounter => prevCounter + incrementBy);
  };

  const handleReset = () => {
    setCounter(0);
  };

  const handleIncrementByDelay = () => {
    setTimeout(() => {
      setCounter(prevCounter => prevCounter + incrementBy);
    }, delay * 1000);
  };
}

return (
  <div>
    <h2>Counter: {counter}</h2>
    <button onClick={handleIncrement}>Increment</button>
    <button onClick={handleReset}>Reset</button>
    <div>
      <label>
        Increment by:
        <input type="number" value={incrementBy} onChange={(e) => setIncrementBy(e.target.value)}>
      </label>
    </div>
    <div>
      <label>
        Delay:
        <input type="number" value={delay} onChange={(e) => setDelay(Number(e.target.value))}>
      </label>
    </div>
    <button onClick={handleIncrementByDelay}>Increment After Delay</button>
  </div>
);
```

Regenerate response

Send a message.



```
const handleReset = () => {
  setCounter(0);
};

const handleIncrementByDelay = () => {
  setTimeout(() => {
    setCounter(prevCounter => prevCounter + incrementBy);
  }, delay * 1000);
};

return (
  <div>
    <h2>Counter: {counter}</h2>
    <button onClick={handleIncrement}>Increment</button>
    <button onClick={handleReset}>Reset</button>
    <div>
      <label>
        Increment by:
        <input type="number" value={incrementBy} onChange={(e) => setIncrementBy(e.target.value)}>
      </label>
    </div>
    <div>
      <label>
        Delay:
        <input type="number" value={delay} onChange={(e) => setDelay(Number(e.target.value))}>
      </label>
    </div>
    <button onClick={handleIncrementByDelay}>Increment After Delay</button>
  </div>
);

export default CounterApp; Regenerate response
```



where did you use useEffect?



Apologies for the confusion earlier, I made a mistake in the introduction. In the provided code, we didn't use the `useEffect` hook as there was no need for handling side effects based on state or prop changes in this example.



Optimism Level



- ChatGPT generated counter app example used for demo
- It did well with styling and incremental updates - creating cards, positioning them, adding a global reset button...
- But... then I asked it to split screen into two and render fiber tree representation on the app on the right hand side

HTML + CSS

Base Counter

0

Increment **Reset All**

Counter 1

0

Multiplier:

Counter 2

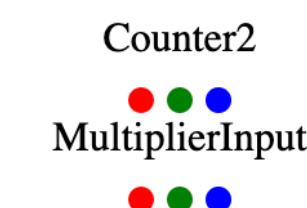
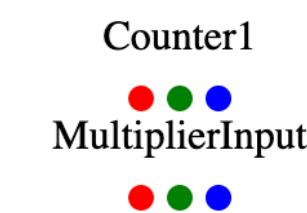
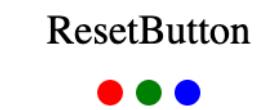
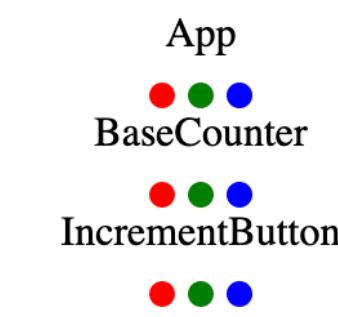
0

Multiplier:

Counter 3

0

Multiplier:



Graphviz - First try

Base Counter
0

Increment Reset All

Counter 1
0

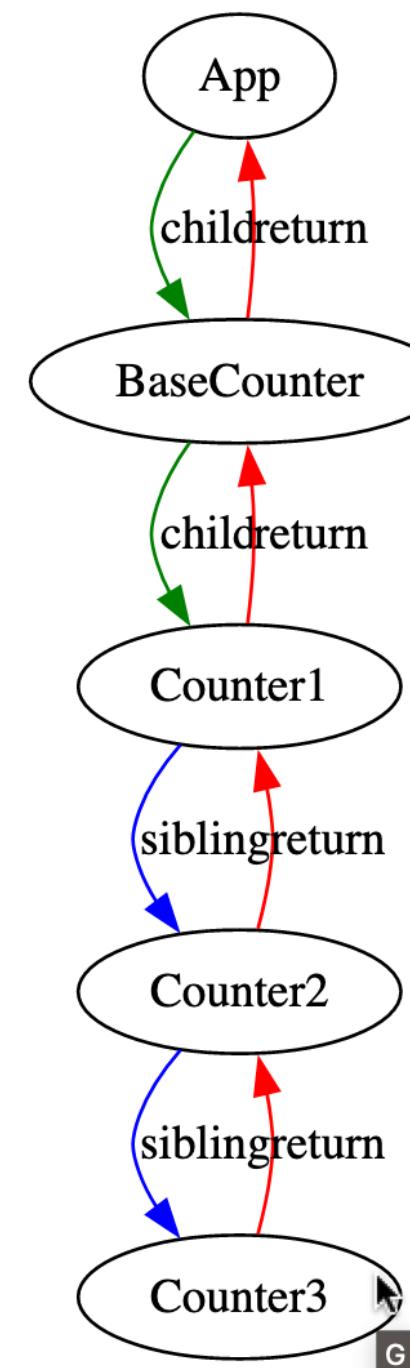
Multiplier:

Counter 2
0

Multiplier:

Counter 3
0

Multiplier:



Wrong connections - return connects to parent. In this representation, it looks like node can be a sibling and a child at the same time which is incorrect

Graphviz - after a few correcting prompts

Base Counter

0

Increment

Reset All

Counter 1

0

Multiplier: 2

Counter 2

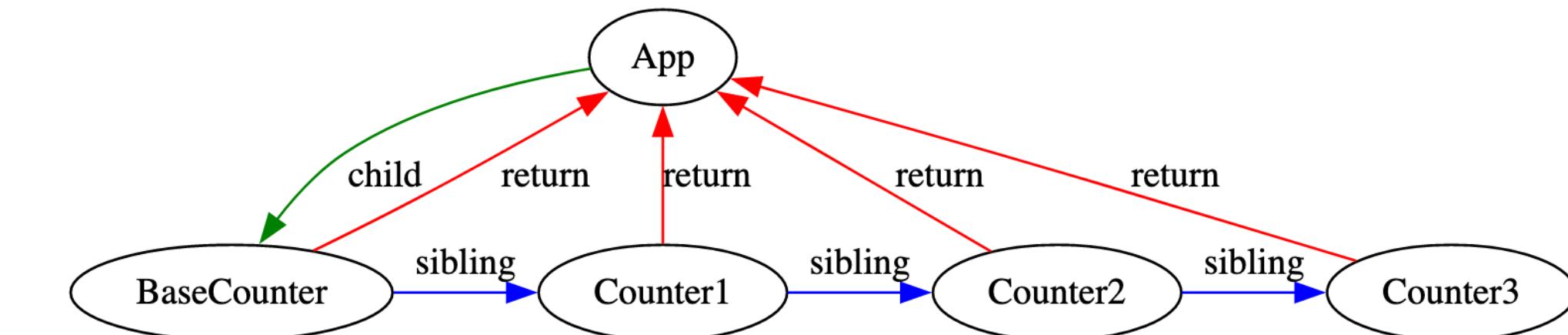
0

Multiplier: 3

Counter 3

0

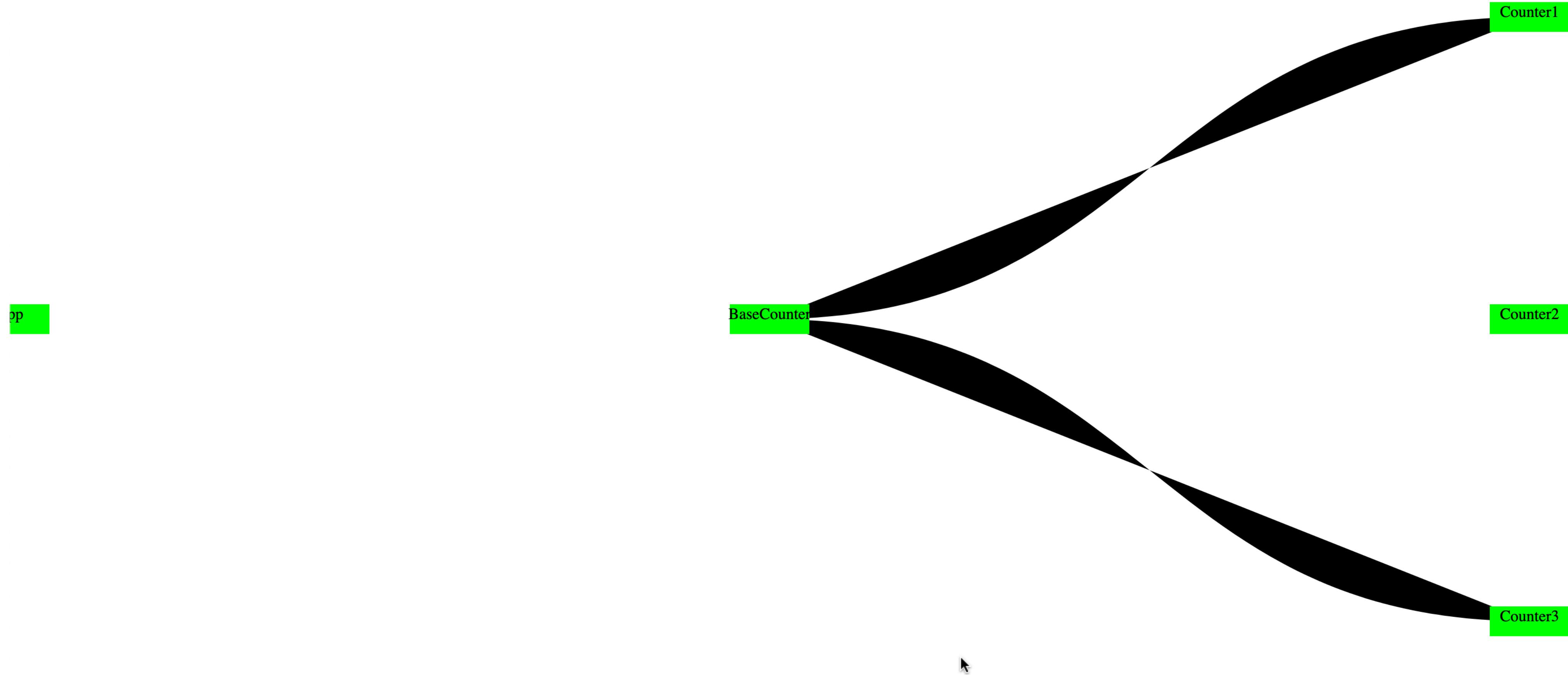
Multiplier: 4



Optimism Level

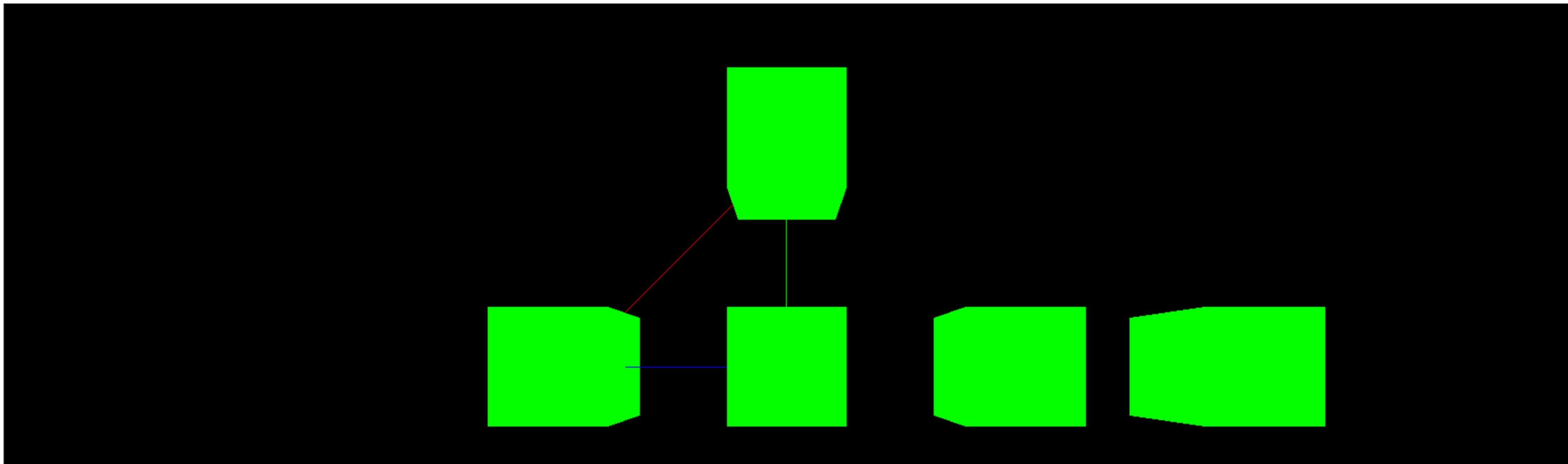


D3 ?!?!?!?!



Three.js

Not saying it was a good idea to even ask



Optimism Level



Warnings

- Do not use ChatGPT for work unless approved
 - Do not send company code to ChatGPT unless approved
 - Everything sent to ChatGPT may be used in training
 - Amazon warns employees against use of ChatGPT: Report
 - Do not paste code provided by ChatGPT into production
 - Licensing issues - what if generated code has copyleft license?
 - Patent issues - what if generated code is patented?
 - Do not blindly trust it
 - Use critical thinking and common sense, verify provided information and be very critical of generated code
- (c) Use of Content to Improve Services. We do not use Content that you provide to or receive from our API ("API Content") to develop or improve our Services. We may use Content from Services other than our API ("Non-API Content") to help develop and improve our Services. You can read more here about how Non-API Content may be used to improve model performance. If you do not want your Non-API Content used to improve Services, you can opt out by filling out this form. Please note that in some cases this may limit the ability of our Services to better address your specific use case.
- <https://openai.com/policies/terms-of-use>

Sources

- <https://github.com/facebook/react>
- <https://legacy.reactjs.org/docs/how-to-contribute.html>
- [React Fiber Architecture](#)
- [Lin Clark - A Cartoon Intro to Fiber - React Conf 2017](#)
- [SMOOSHCAST: React Fiber Deep Dive with Dan Abramov](#)
- [“Hacking the Mainframe”: Digging into React Fiber with Nothing but the Debugger](#)
- [Inside Fiber: in-depth overview of the new reconciliation algorithm in React](#)
- [Build your own React - Rodrigo Pombo](#)
- [React Source Code Walkthrough Playlist - JSer](#)

Questions?