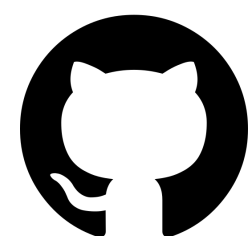


React Render Tracker v0.6

Overview & Instructions

Revision November 23, 2021



[react-render-tracker](https://github.com/react-render-tracker/react-render-tracker)

Why React Render Tracker?

Filling the gap in React's tooling

- [React Devtools](#) shows current state of components tree and components themselves, doesn't track unmounted components, lacks details on update reasons and changes in components, etc.
- [Why Did You Render](#) is focused on updates triggered by props changes only, limited in provided information (due to no access to internals), mostly a message log in a console (no UI), etc.
- [<Profile>](#) allows to collect timings for a subtree, no details for update reasons and lifecycle events, etc.

React Render Tracker (RRT) – a tool to discover performance issues related to unintentional updates in React apps.

It presents component's tree state over the time, with detailed information related to a selected component like props and state updates, as well as reasons for an update, changes in memo hooks, used contexts etc.

⚠ It's not a replacement for React Devtools, but a compliment to it with a focus on exploration changes in the app's component tree (like mounts, updates and unmounts) and their causes.

Inside of RRT

Navigation over selected fibers history

Search for a fiber by display name

Fiber's (component) tree

Details on selected fiber

Toggle hierarchy mode: owner-based (default) or parent-based

Toggle displaying of unmounted fibers

Toggle displaying timings

Reset loaded events

Pin fiber's subtree

Navigation over instances of selected fiber type

Event's log

Statusbar with overall statistics

← → | 🔍 Find by display name

Render root #1 Legacy Mode Demo for RRT...

App #2 4

- Settings.Provider #3 ! 4
- Header #4 Memo 4
 - Loader #6
 - AvatarPlaceholder #7
 - Avatar #35
- List #13 Memo 2 2
 - Loader #14 1
 - ListItem #20 1
 - Checkbox #22 1 2
 - ListItem #25 23 1
 - Checkbox #27 2 2
 - ListItem #30 566
 - Checkbox #32 1 2
- Overlay #16 4

MEMO

List #13
Child of Settings.Provider #3, created by App #2

PROPS UPDATES & MEMO (4)

Reaction	items	limit	selectedId
Update	SE	-	±
React.memo() bailout	-	-	-
Update	±	-	-
React.memo() bailout	-	-	-

EVENTS (5) include subtree

- Commit #0
- Render root #1 Demo for RRT...
- List #13
- Commit #1
- App #2 > ± state
- List #13 > ± props, state
- Commit #2
- App #2 > ± state
- List #13 React.memo()
- Commit #3
- App #2 > ± state

62 events (15.6Kb) for 16 component instances

mounts: 16 updates: 24 unmounts: 9

Fiber vs. Component

React Fiber is a reimplementation of React's core algorithm (reconciliation) introduced in React 16.

A **fiber** is an object (internal instance) to represent a "unit of work" including instances of components. A tree is built from fibers (**fiber's tree**), separate for each render root which is creating by [ReactDOM.render\(\)](#) or [ReactDOM.createRoot\(\)](#). Tools like [React Devtools](#) or [React Render Tracker](#) are displaying the state of fiber's tree.

A **component** usually stands for a class or a function component, the things that are carrying a unit of app's logic and are defined by developers. That's fine to use component instead of fiber in most cases.

[React Render Tracker](#) is distinguish the following **types of fiber**:

- Render root
- Host component (doesn't handle by RRT for now)
- Class component
- Function component
- Memo
- ForwardRef
- Context.Provider
- Context.Consumer
- Suspense
- Suspense list
- Profiler
- Unknown (anything else)

Tree hierarchy types

```
function Header({ avatar }) {  
  if (avatar.loading) {  
    return (  
      <Loader>  
        <AvatarPlaceholder />  
      </Loader>  
    );  
  }  
  return <Avatar avatar={avatar} />  
}
```

Owner-based hierarchy (by default)

- Header #4
 - Loader #6
 - AvatarPlaceholder #7
 - Avatar #35

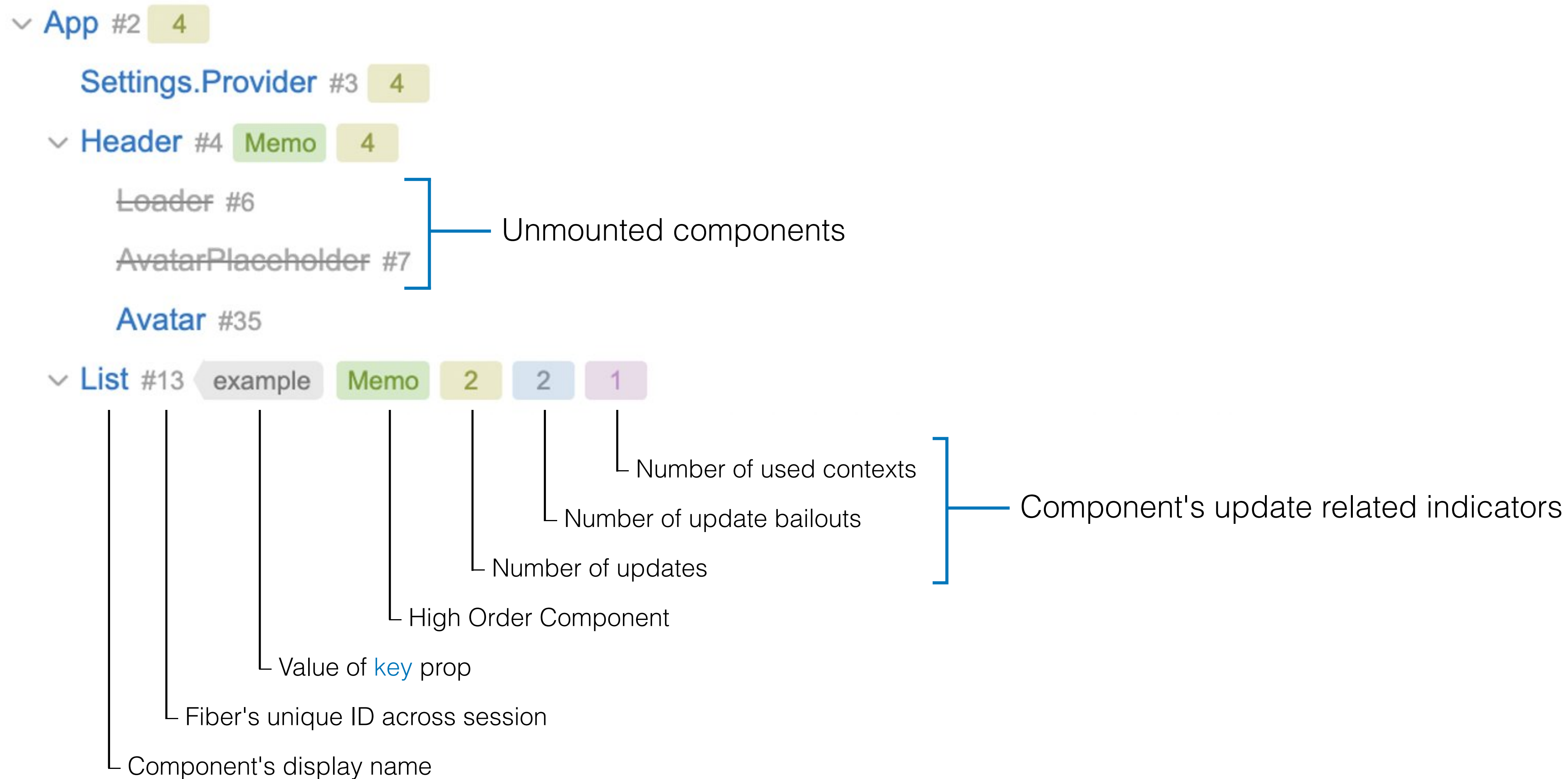
Parent-based hierarchy

- Header #4
 - Loader #6
 - AvatarPlaceholder #7
 - Avatar #35

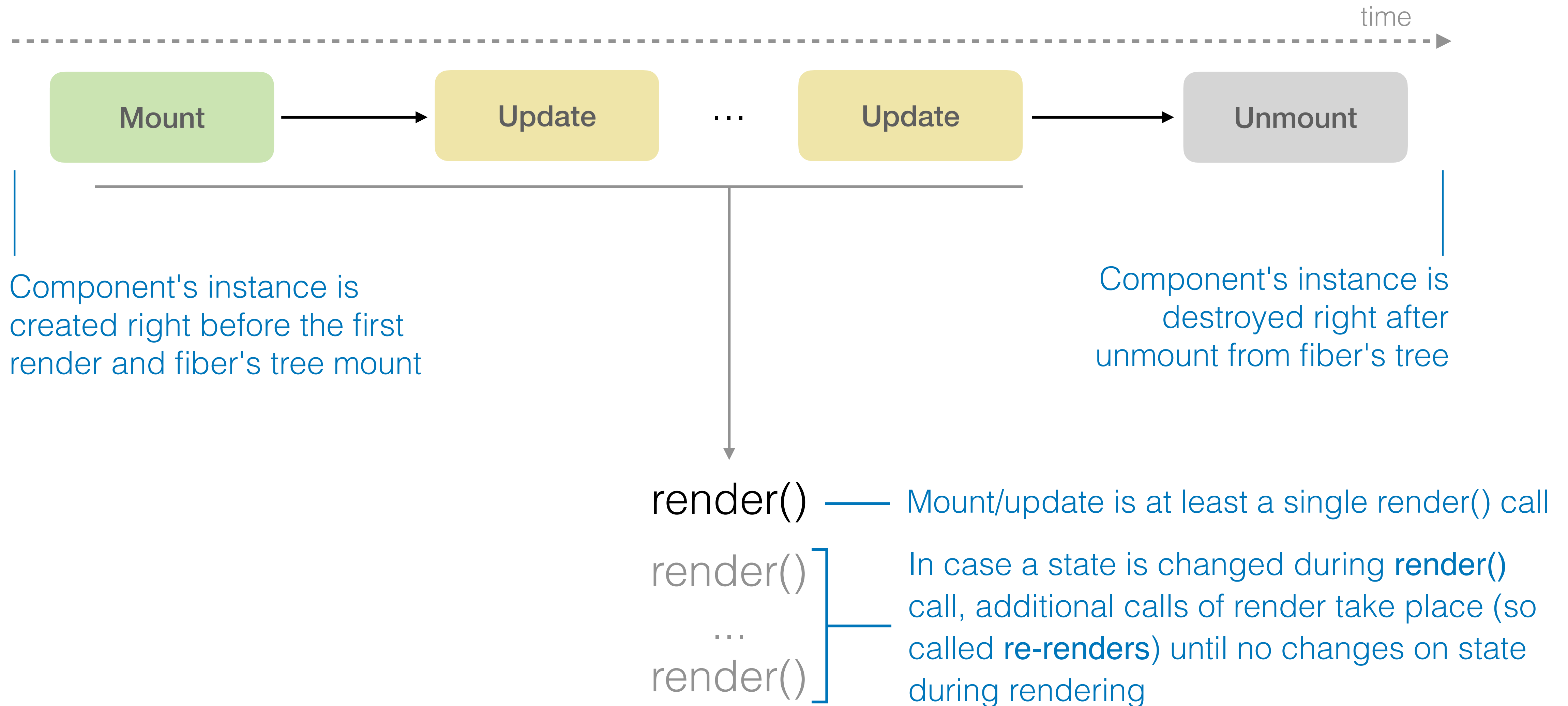
Owner-based hierarchy (or owner-ownee relationship) is more useful for understanding which components will be updated on owner's update (if no update bailout will take place). An owner is the component that sets the props of other components. In other words, an owner is a component that creates other components on its render which become its ownees.

Parent-based hierarchy (or parent-child relationship) is the way how components are composed and effects like context or suspense take place. A parent is responsible for which **child components** will be mounted into fiber's tree.

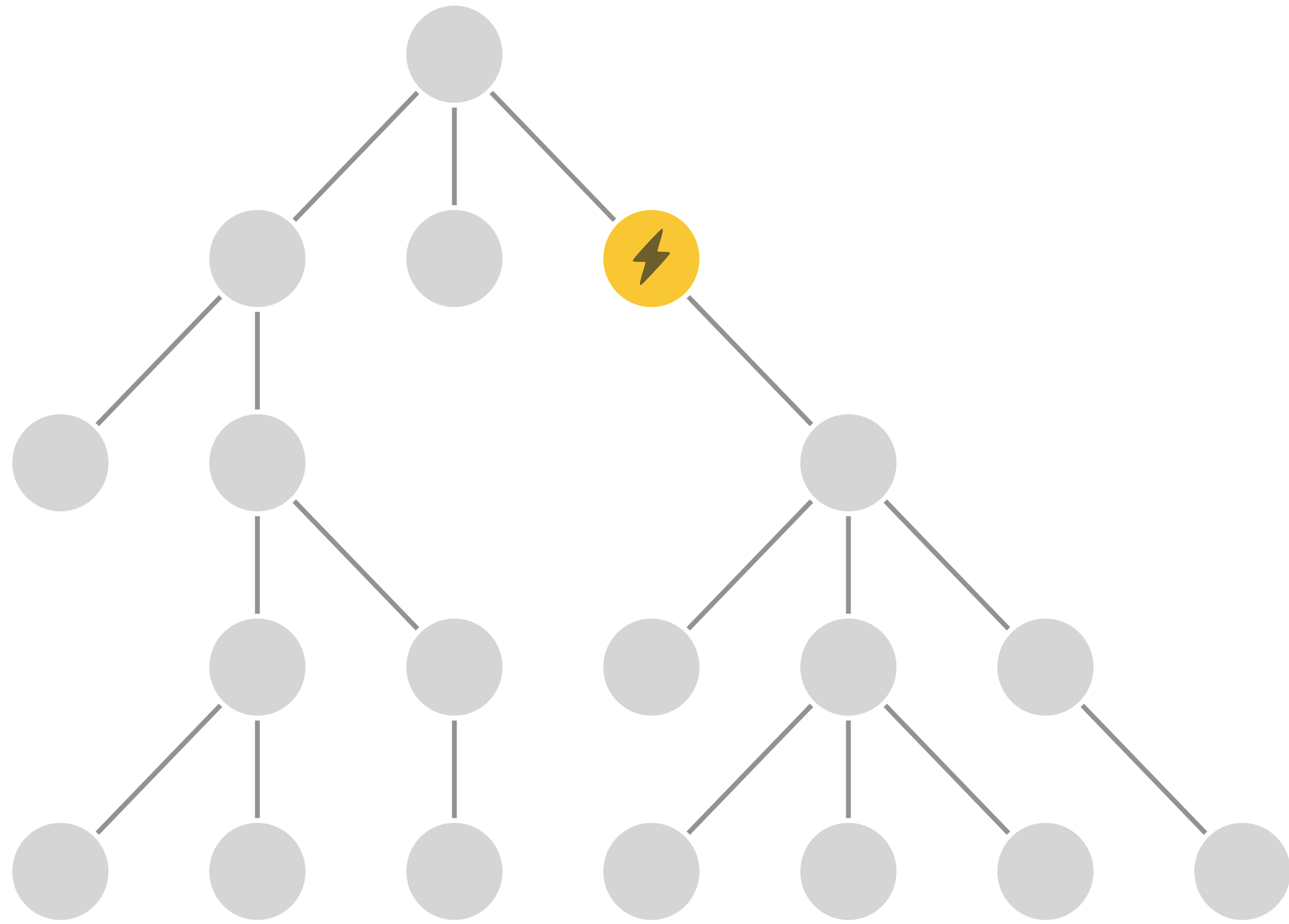
Fiber's tree



React component lifecycle

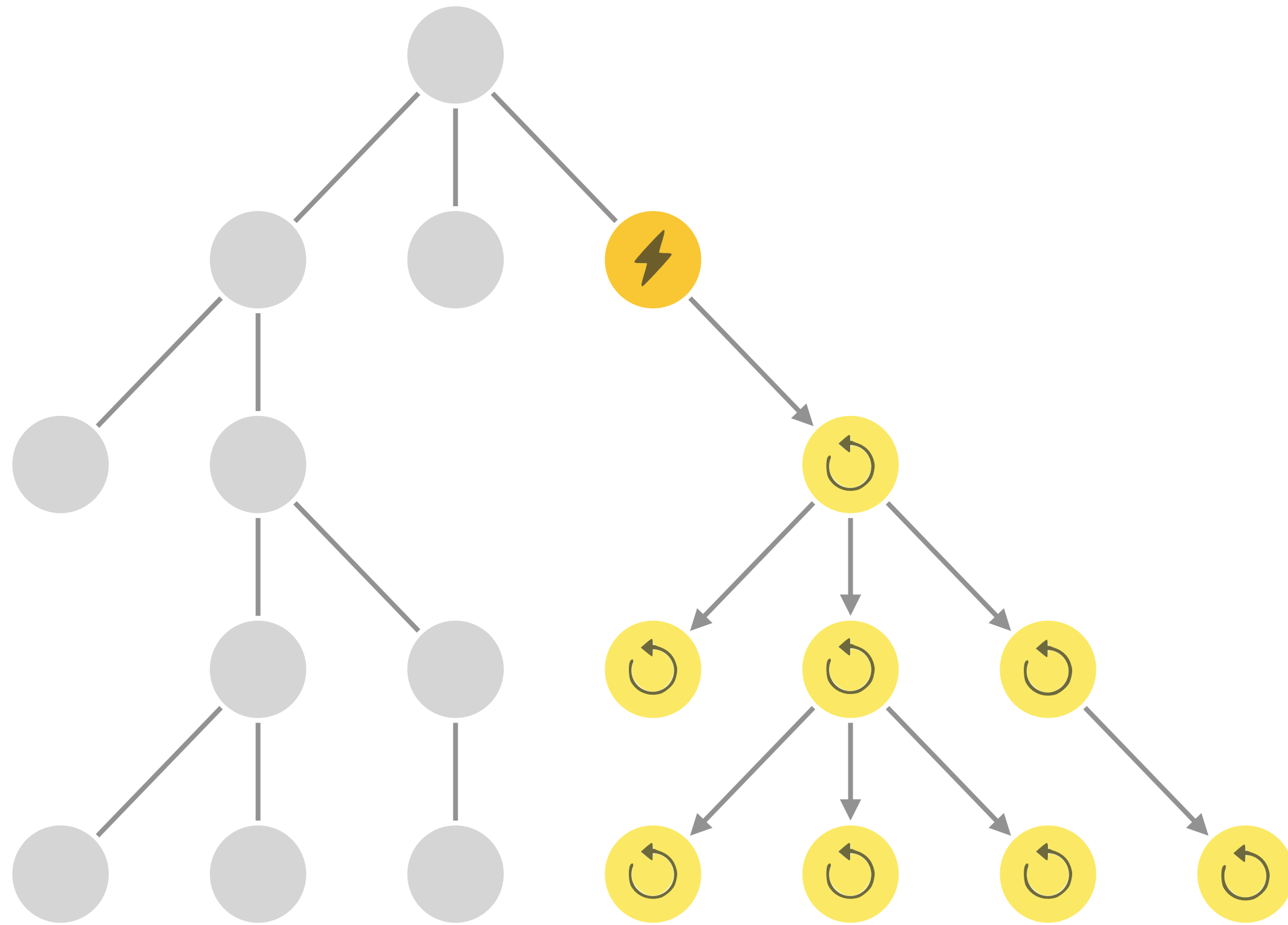


Updating fiber's tree



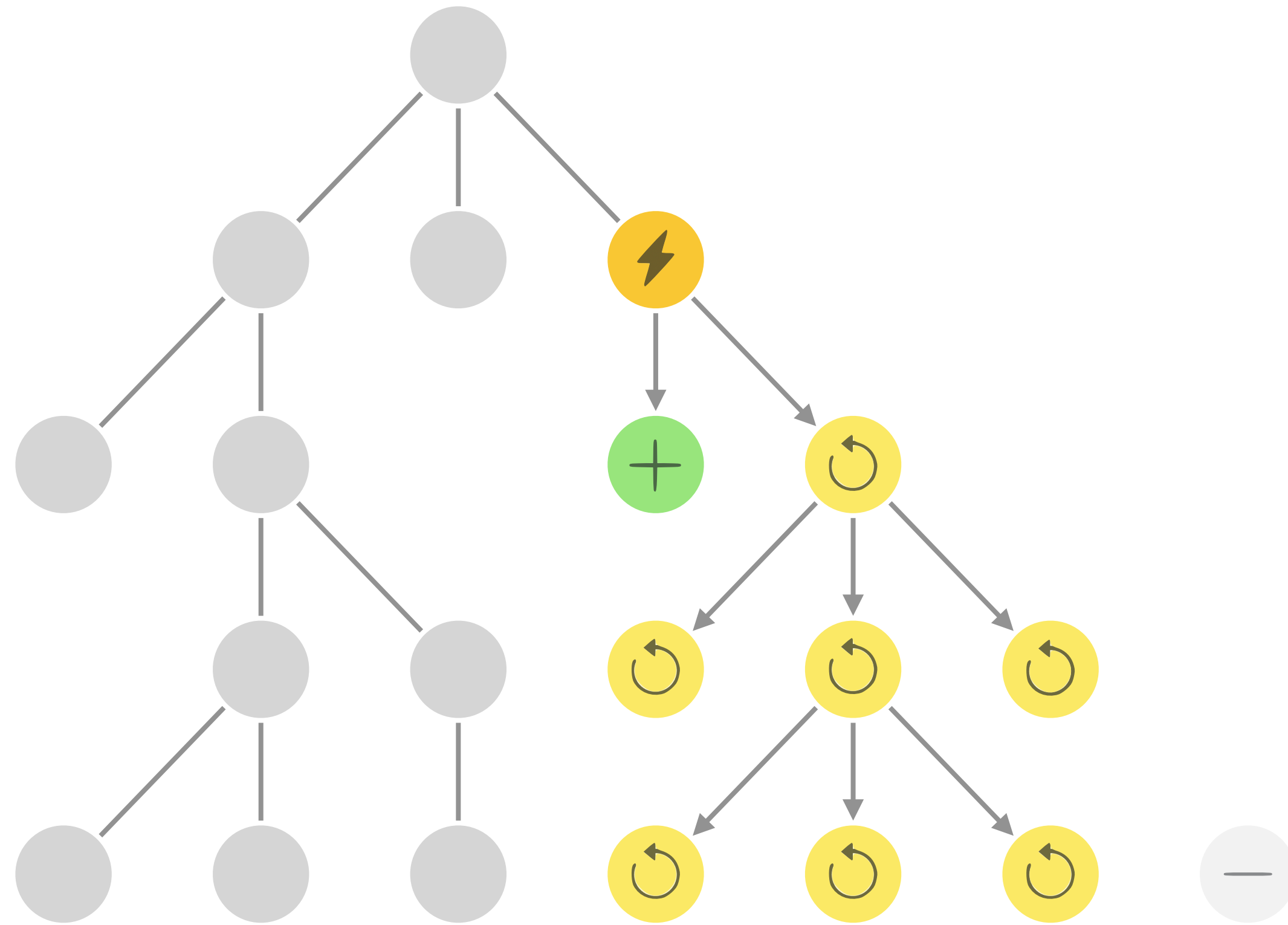
- ⚡ A component update can be initiated for one of the reasons, in most cases because of state change. Such reasons can be called **fire starters**, as they lead to an update of the fiber tree (a reconciliation).

Updating fiber's tree



- ⚡ A component update can be initiated for one of the reasons, in most cases because of state change. Such reasons can be called **fire starters**, as they lead to an update of the fiber tree (a reconciliation).
- 🔄 A component update usually results in nested components being updated because of props or context value change. Such effect reasons can be called **continutors**, since each updated component results in child components being updated until the end of the subtree is reached. It looks like a **wave of updates**.

Updating fiber's tree

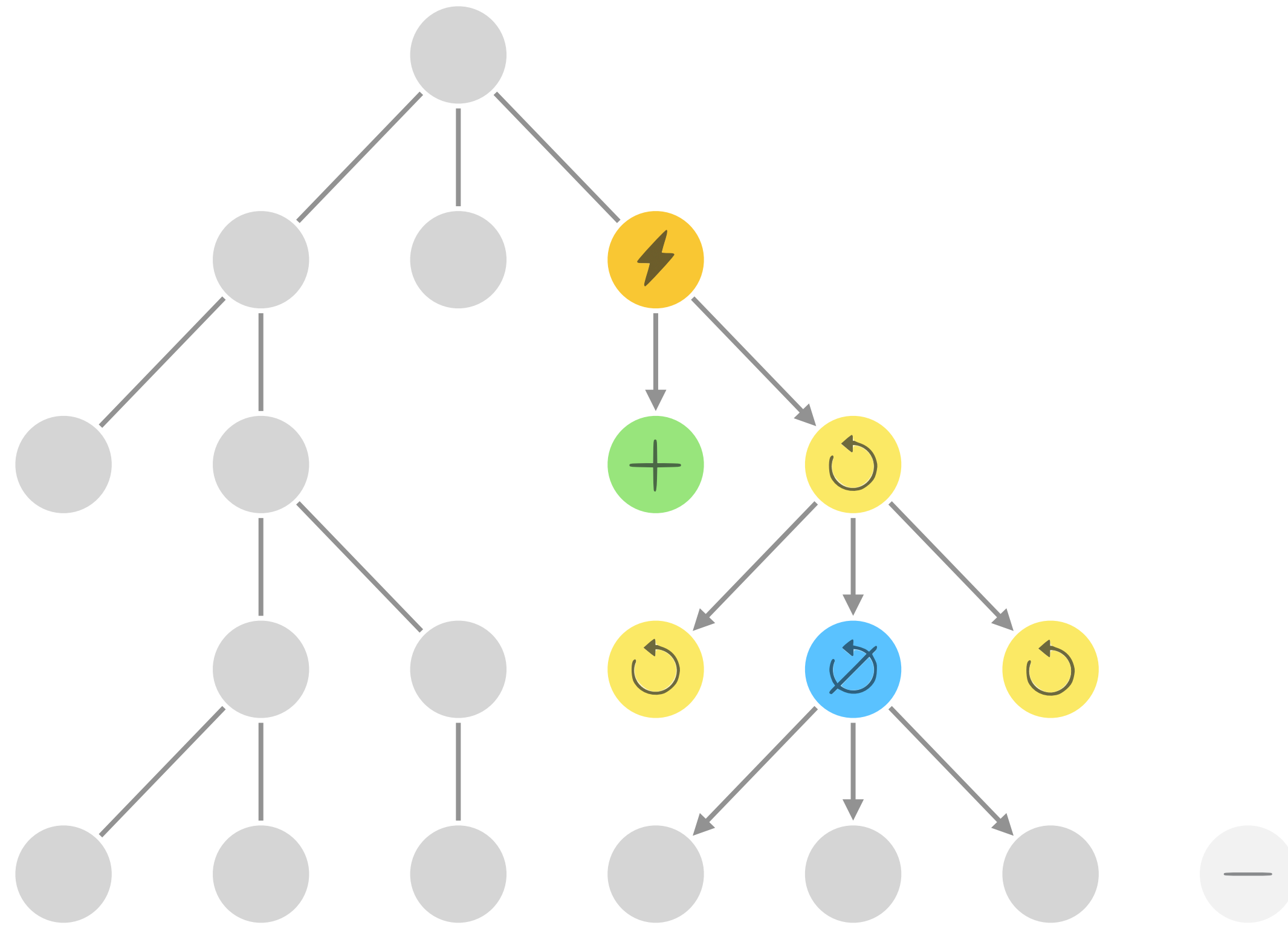


⚡ A component update can be initiated for one of the reasons, in most cases because of state change. Such reasons can be called **fire starters**, as they lead to an update of the fiber tree (a reconciliation).

🔄 A component update usually results in nested components being updated because of props or context value change. Such effect reasons can be called **continuator**s, since each updated component results in child components being updated until the end of the subtree is reached. It looks like a **wave of updates**.

— + On update wave some components might be mounted or unmounted

Updating fiber's tree



⚡ A component update can be initiated for one of the reasons, in most cases because of state change. Such reasons can be called **fire starters**, as they lead to an update of the fiber tree (a reconciliation).

🔄 A component update usually results in nested components being updated because of props or context value change. Such effect reasons can be called **continutors**, since each updated component results in child components being updated until the end of the subtree is reached. It looks like a **wave of updates**.

— + On update wave some components might be mounted or unmounted

🚫 A component can avoid its own update as well as its subtree (update bailout), e.g. by using **React.memo()**

Reasons for update

State changed

Component#setState() or useState() / useReducer() hooks callback

Component#forceUpdate()

Root render, e.g. ReactDOM.render() call

Suspense / React.lazy()

Props changed by component's owner

Used context provider's value prop changed

Fire starters

Triggers for fiber's tree reconciliation.
Can be batched.

Continuators

Reasons which are expanding update area of fiber's tree on reconciliation.

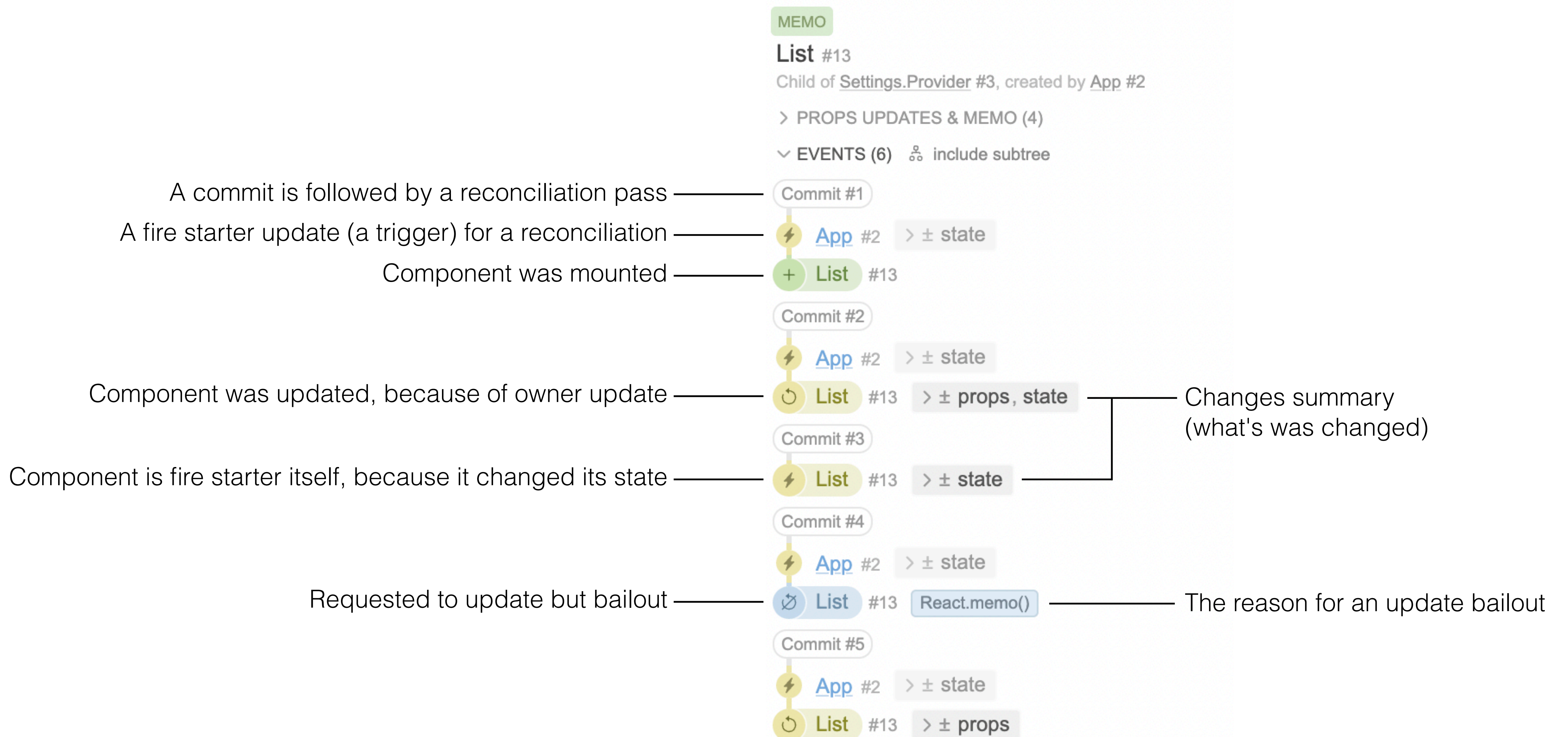
Supported by React Render Tracker v0.6

Reasons for update bailout

- `React.memo()`
- `Component#shouldComponentUpdate()`
 - `PureComponent` has predefined `shouldComponentUpdate()` method which shallow compares prev/next props and state
- No state changes
 - when queued/batched updates is applied on render
- Same type & props

Supported by React Render Tracker v0.6

Event log



Change details

A click on the change summary badge will show a block with details of a change. Only changed things are displaying in this block.

Commit #2

App #2 $\nabla \pm$ state

- STATE [useState](#) #1 [\[\[useState\]\]](#) `null` \rightarrow `[]`

List #13 $\nabla \pm$ props, state

- PROP items `[]` \rightarrow `[]` [⚠ Shallow equal](#)
- STATE [useState](#) #0 [\[\[useState\]\]](#) {
 - initial: `true`
 - dataset: `"none"` \rightarrow `"default"`
 - ... all the rest 2 entries have not changed}

Change details

The screenshot shows Redux DevTools with two components: App #2 and List #13. App #2's state is expanded to show a useState hook (#1) where the value changed from null to an empty array []. List #13's props and state are expanded. The props section shows the items prop changing from an empty array to another empty array, with a 'Shallow equal' warning. The state section shows a useConfig hook pointing to a useState hook (#0) with the following changes: initial: true (unchanged), dataset: 'none' to 'default', and a note that the other two entries have not changed.

```
Commit #2
App #2
  state
    STATE useState #1 [[setState]] null → []
List #13
  props, state
    PROP items [] → [] ⚠ Shallow equal
    STATE useConfig → useState #0 [[setState]] {
      - initial: true
      dataset: "none" → "default"
      ... all the rest 2 entries have not changed
    }
```

A click on the change summary badge will show a block with details of a change. Only changed things are displaying in this block.

"Shallow equal" means that all entries of objects or all elements of arrays are equal, but objects or arrays are different by references. In most cases it means that we can avoid the change (if it makes sense)

Change details

Commit #2

App #2

- STATE [useState #1](#) [\[\[useState\]\]](#) null → []

List #13

- PROP items [] → [] [⚠ Shallow equal](#)
- STATE [useState #0](#) [\[\[useState\]\]](#) {
 - initial: true
 - dataset: none → default
 - ... all the rest 2 entries have not changed}

A click on the change summary badge will show a block with details of a change. Only changed things are displaying in this block.

"Shallow equal" means that all entries of objects or all elements of arrays are equal, but objects or arrays are different by references. In most cases it means that we can avoid the change (if it makes sense)

[useState #0](#) [\[\[useState\]\]](#)

A stack trace to place where React.useState() hook is used and a location where its setState callback was invoked.

RRT could open known locations right in a editor (additional configuration is needed, see below)

Warnings

The screenshot shows the React Render Tracker interface. On the left, a component tree is visible with 'Settings.Provider #3' highlighted, showing a warning icon and the number '4'. On the right, the 'Settings.Provider #3' component is selected, showing its props: 'value' and 'children'. The 'value' prop is highlighted with a warning icon and the text 'Shallow equal', indicating a shallow equality check warning.

 React Render Tracker indicates potential problem places with an exclamation mark icon

For now (v0.6.0) only two types of warning are implemented:

- Shallow equal change on state
- Shallow equal change on Provider's value prop

[Suggest a new warnings](#)

Props updates section

MEMO

BaseAppLayout #130

Child of UNSAFE_DialogOffsetContext.Provider #129, created by LayoutWithResponsive #106

▼ PROPS UPDATES & MEMO (7)

Reaction	appLayoutAre...	correlationIdRef	experienceCha...	layoutPhaseSt...	matchedSlots...	matchedTempl...	slots
React.memo() bailout		-					
React.memo() bailout		-					
Update	-	-	-	-	-	-	±

```

slots {
  header: {...} → {...}
  main: {...} → {...}
  modal: {...} → {...}
  ... all the rest 4 entries have not changed
}

```

Update	±	-	-	-	SE	±	-
React.memo() bailout		-					
Update	±	-	-	-	SE	±	-
React.memo() bailout		-					

How often props are changing and which ones.

Answers the questions:

- Does it **make sense to add** React.memo() / shouldComponentUpdate()?
- **How effective** React.memo() / shouldComponentUpdate() is?
- Is it possible to avoid props change to **increase effectiveness** of React.memo() / shouldComponentUpdate()?

± Value of a prop has been changed

SE Prev and next value of a prop are **shallow equal**

Memo hooks section

MEMO

LayoutWithResponsive #106

Child of & created by LayoutWithConfiguredSlots #127

> PROPS UPDATES & MEMO (2)

> CONTEXTS (2)

MEMO HOOKS (15)

1. [useLogger](#) → [useMemo\(...\)](#)
Never recompute
2. [useMatchedTemplate](#) → [useCallback\(...\)](#)
Never recompute
3. [useMatchedTemplate](#) → [useLogger](#) → [useMemo\(...\)](#)
Never recompute
4. [useMatchedTemplate](#) → [useMemo\(...\)](#)
3 of 7 updates recompute

Update	[value]	dep 0	dep 1
1. ± props	±	-	±
2. ± context	-	-	-
3. ± props	-	-	-
4. ± state	±	±	-
5. ± context	-	-	-
6. ± state	±	±	-
7. ± context	-	-	-

5. [useSubNavResponsiveWidths](#) → [useCallback\(...\)](#)
Never recompute

Usage of useMemo() and useCallback() hooks

Answers the questions:

- How many memo hooks are used and where?
- How often memo hooks recompute and why?
- Is it possible to avoid deps change to increase effectiveness of a memo hook usage?

± Value of a prop has been changed

SE Prev and next value of a prop are shallow equal

Used context section

MEMO

SlotInner #241

Child of & created by [SlotTransitionAndInner #240](#)

PROPS UPDATES & MEMO – NO NEW PROPS SINCE MOUNT

CONTEXTS (3)

[AnonymousContext #31](#)

1. [useLeakDetector](#) → [useContext\(...\)](#)

[UserContext #21](#)

1. [useUserContext](#) → [useContext\(...\)](#)

2. [useCoreSettings](#) → [useUserContext](#) → [useContext\(...\)](#)

3. [useUpdateEntity](#) → [useLogger](#) → [useUserContext](#) → [useContext\(...\)](#)

[AnonymousContext`2 #6](#)

1. [useUpdateEntity](#) → [useApolloClient](#) → [useContext\(...\)](#)

FUNCTION COMPONENT

Checkbox #22

Child of & created by [ListItem #20](#)

PROPS UPDATES – NO NEW PROPS SINCE MOUNT

CONTEXTS (2)

[Settings #3](#)

1. [useDarkmode](#) → [useContext\(...\)](#)

ContextWithNoProvider – **No provider found**

1. [useContext\(...\)](#)

Usage of contexts on a component

Answers the questions:

- Which contexts are used by a component and where their value are read?
- Is it possible to get rid of using a context? (It might be used in several custom hooks)
- Are all context providers in place?

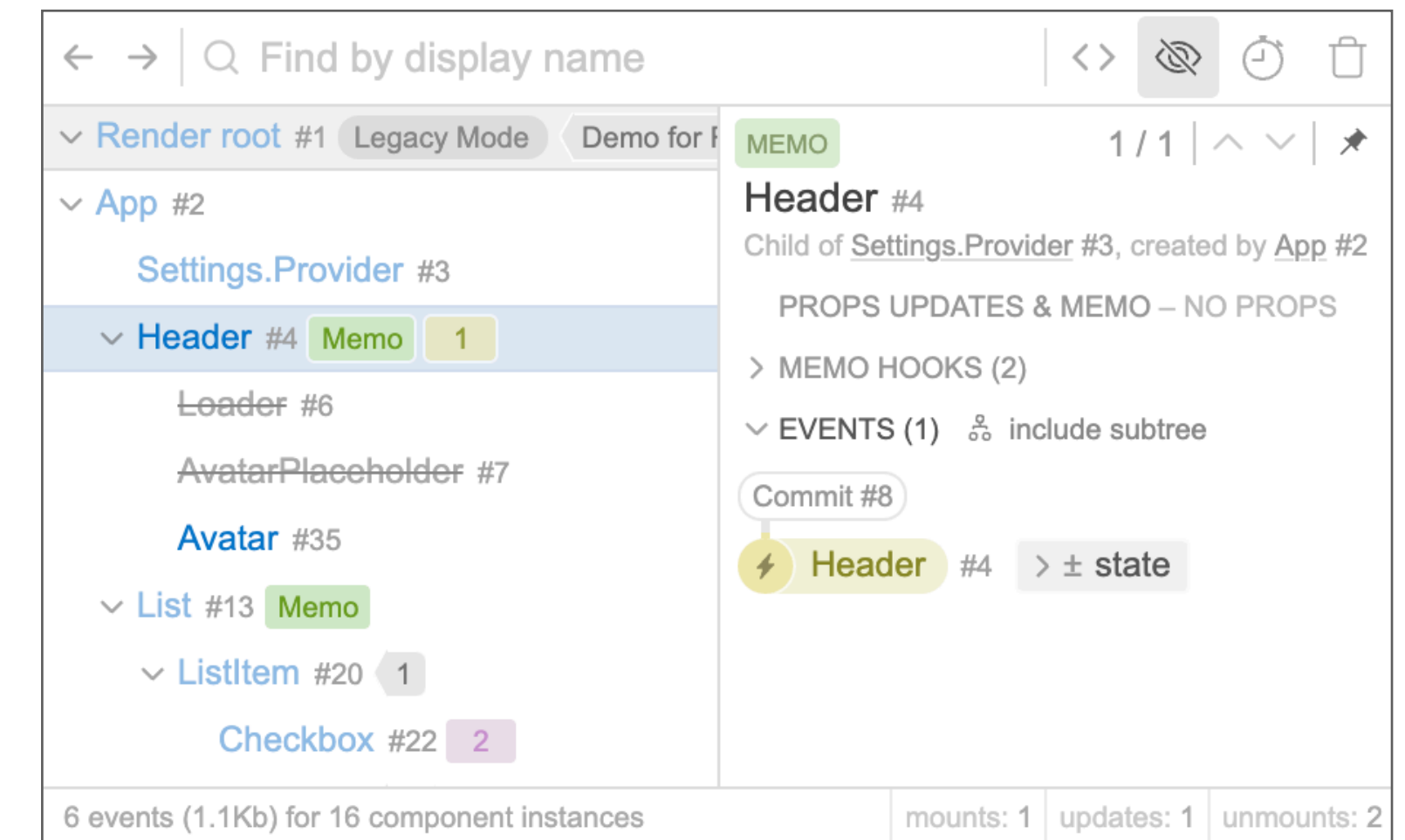
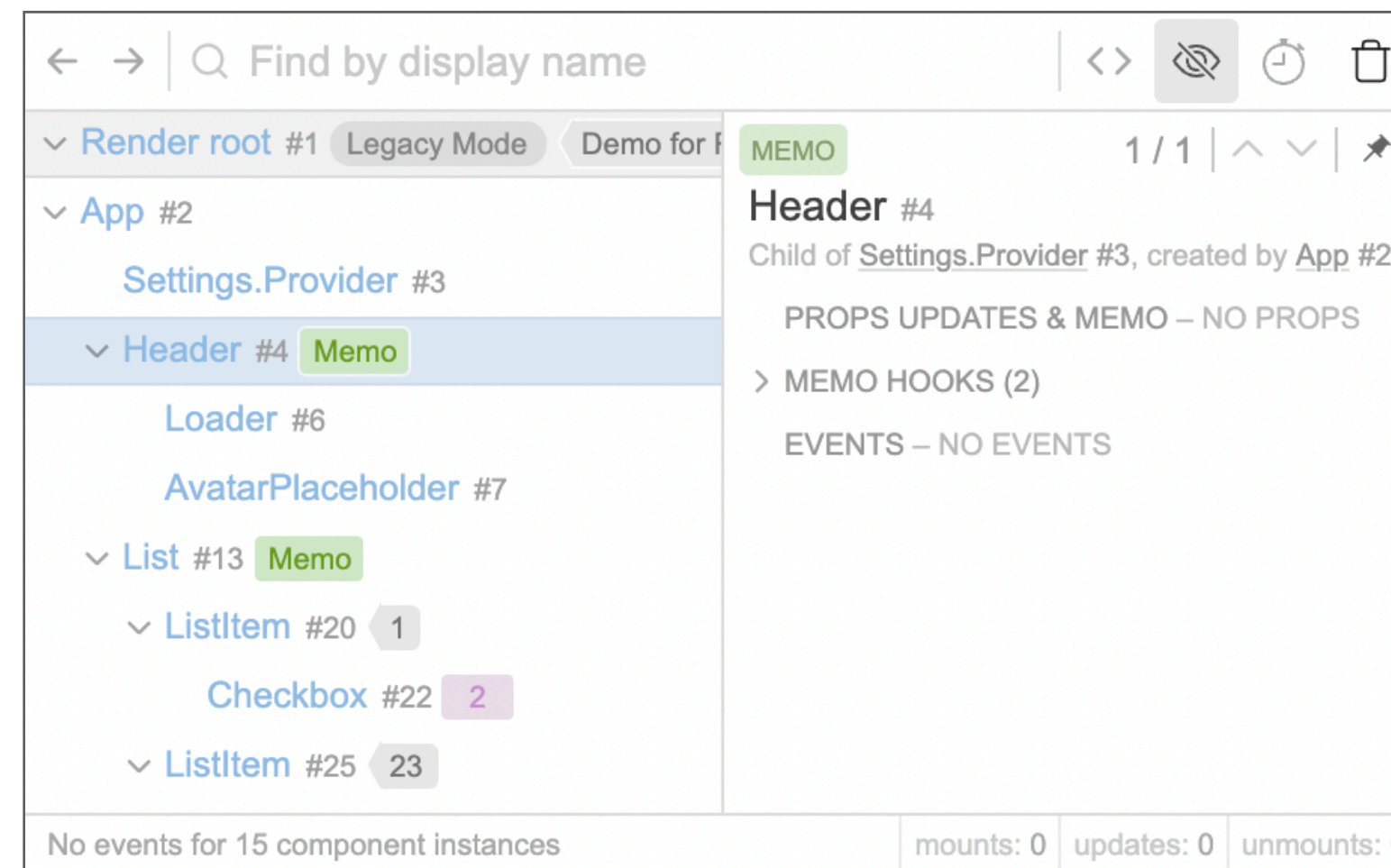
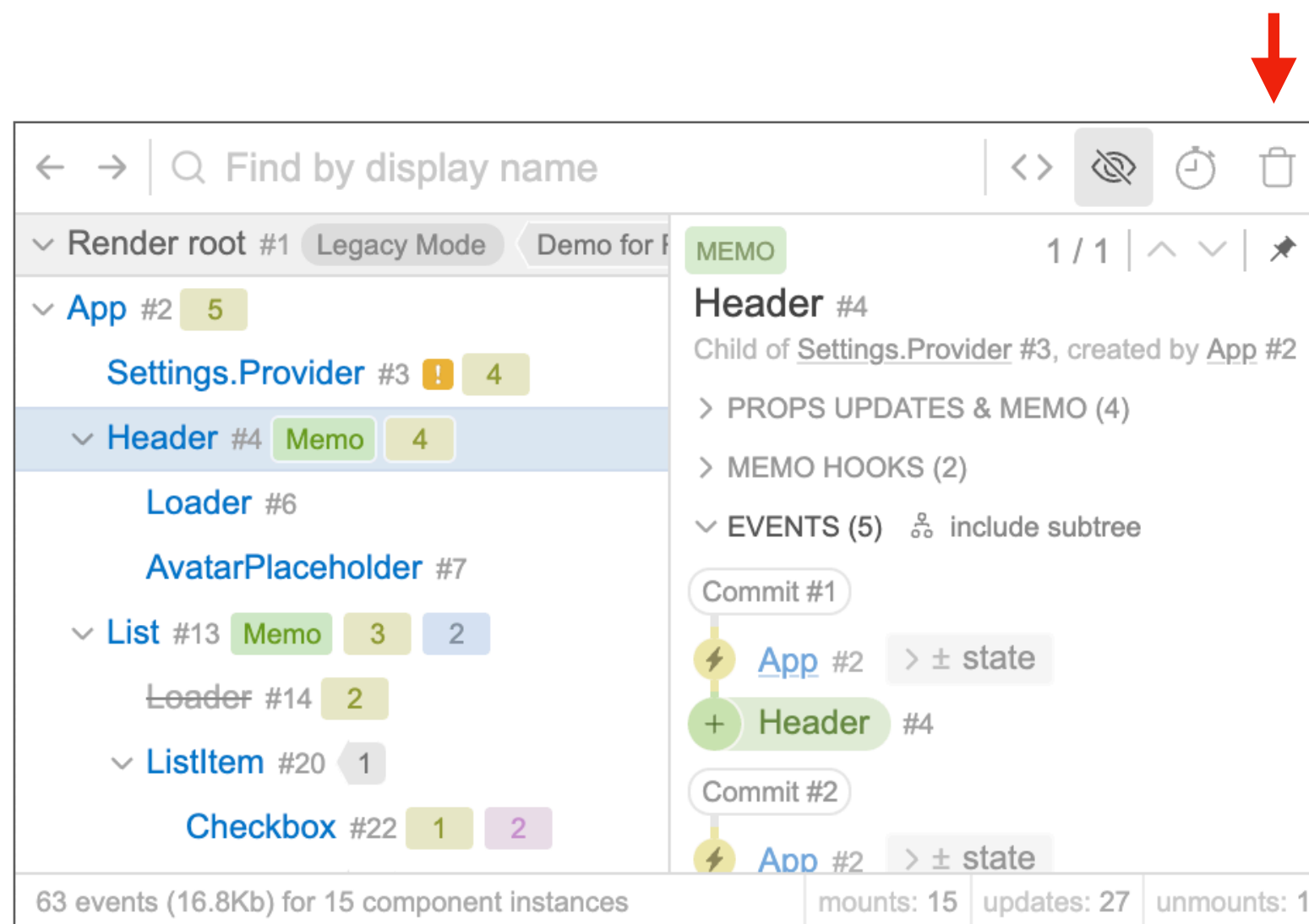
Event diffings

Problem

Need to know what's changing in the app on some actions like a click on a button

Solution

Reset loaded events, do actions, see new events (changes)

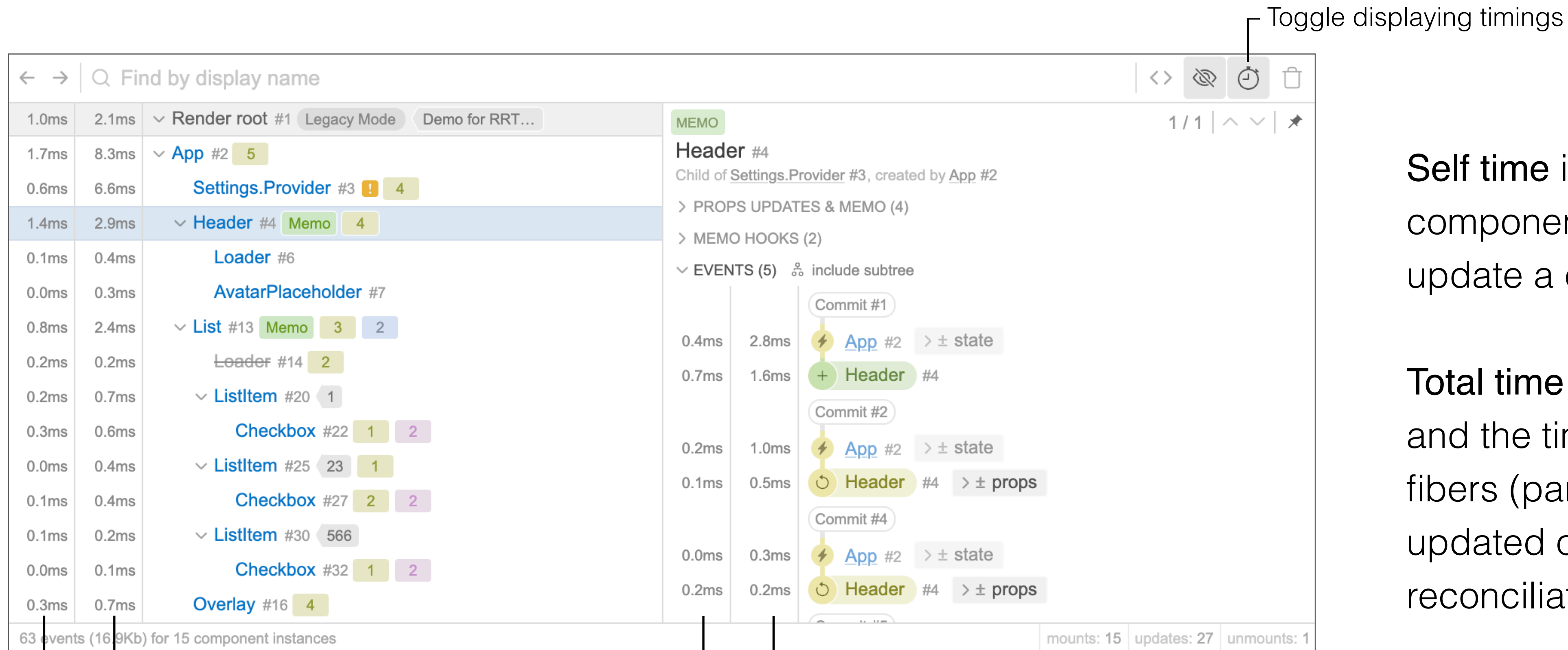


1 Click on button with trash bin icon to reset loaded events. This will also remove any unmounted components.

2 When events are reset on a component, it becomes dimmed in the tree. All event-based statistics vanish as well.

3 This allows to spotlight new events, i.e. changes in the app

Timings



Self time is a time to mount a component to the tree or to update a component

Total time is a sum of self time and the time of all descendant fibers (parent-based) being updated or mounted during a reconciliation

Timings on the tree is a sum of all the mount and update events on a component

How to use RRT?

Add a single `<script>` to the HTML page before React app script.

```
<html>
```

```
...
```

```
<script src="path/to/react-render-tracker.js"></script>
```

```
<script src="./react-app.js"></script>
```

```
...
```

React Render Tracker will attach to React and start collecting data.

The next step is to open UI in one of the ways that works best for your case.

You can use a CDN service to include script with no installation:

```
<!-- jsDelivr -->
```

```
<script src="https://cdn.jsdelivr.net/npm/react-render-tracker"></script>
```

```
<!-- unpkg -->
```

```
<script src="https://unpkg.com/react-render-tracker"></script>
```

Open UI right in the page

```
<script  
  src="https://cdn.jsdelivr.net/npm/react-render-tracker"  
  data-config="inpage:true"  
></script>
```

The screenshot shows a web browser window with the URL `localhost:3000/#`. The page title is "React Render Tracker playground" and it indicates "React version: 17.0". The page content includes a list of links on the left: "All", "Class components", "Basic nested render with setState() via useEffect()", "Props changes", "Basic render with changed parent element", and "Context". The main content area shows "Class components" with several "[OK]" status indicators and a "Trigger" button. Below that, it says "Basic nested render with sets" and "OK". At the bottom, there is a link for "Props changes".

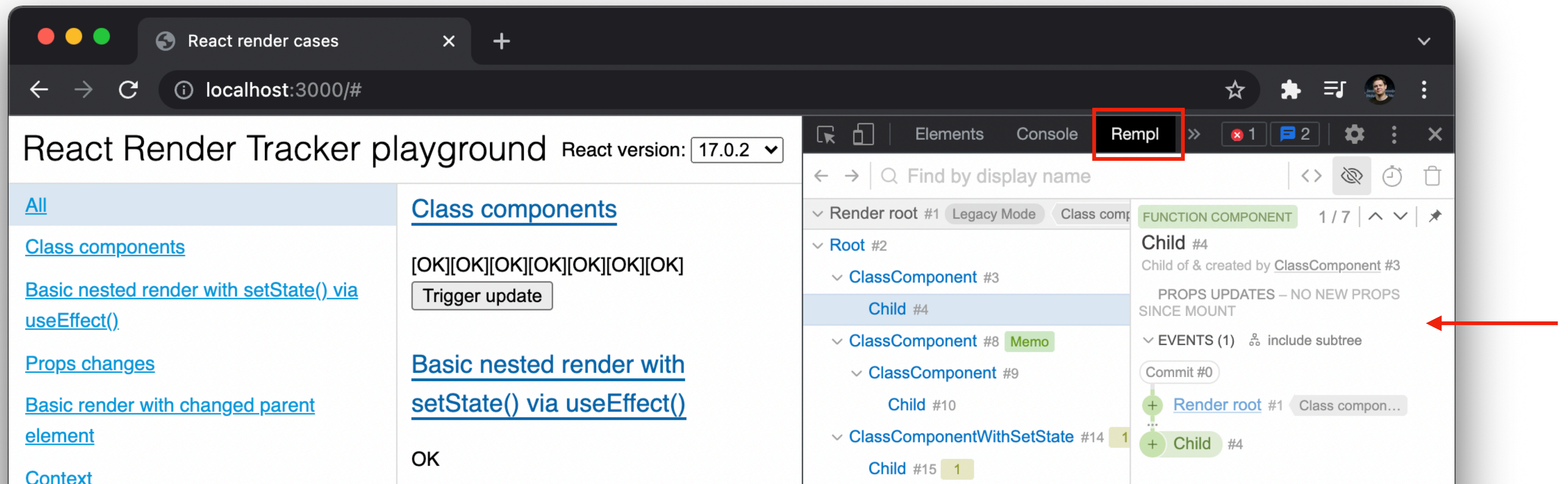
Overlaid on the right side of the browser is the React Render Tracker UI. It has a search bar "Find by display name" and a tree view of the component hierarchy. The tree shows:

- Render root #1 (Legacy Mode) - Class compon...
- Root #2
 - ClassComponent #3
 - Child #4 (highlighted) - Child of & created by ClassComponent #3. A red arrow points to this node.
 - ClassComponent #8 (Memo)
 - ClassComponent #9
 - Child #10
 - ClassComponentWithSetState #14 (1)
 - Child #15 (1)

The right panel shows details for the selected "Child #4" component, including "FUNCTION COMPONENT", "1 / 7", and "PROPS UPDATES - NO NEW PROPS SINCE MOUNT". It also shows "EVENTS (1)" and "Commit #0".

Open UI in browser's devtools

- 1 Install [Rempl extension](#) for a browser: [Chrome](#), [Edge](#) or [Firefox](#)
- 2 Open browser's devtools and choose [Rempl](#) tab to see React Render Tracker UI



Open UI in any webview

- 1 Install and launch Rempl server

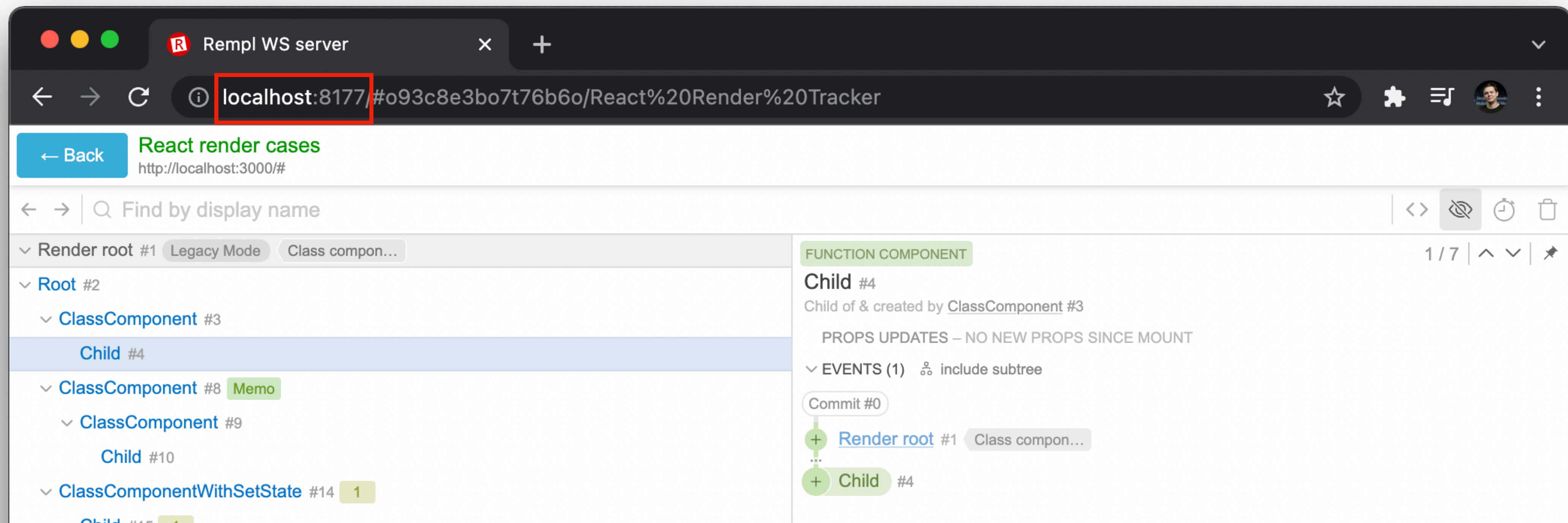
```
> npm install --global repl-cli  
> repl
```

Init standalone version of repl
Server run at <http://localhost:8177>

- 2 Add `<meta>` with a server host to the HTML page

```
<meta name="repl:server"  
      content="localhost:8177" />
```

- 3 Open server's host and select a connected instance of React Render Tracker



Setup "open in editor" feature

- 1 Add a configuration to React Render Tracker to specify dev server endpoint to open file in a editor

```
<script
  src="https://cdn.jsdelivr.net/npm/react-render-tracker"
  data-config="
    openSourceLoc: {
      pattern: '//localhost/open-file?file=[file]',
      projectRoot: '/abspath/to/git/project-name'
    }
  "
></script>
```

← See more details
in [README](#)

- 2 Configure your dev server to provide an endpoint to open file in a editor using one of solutions:

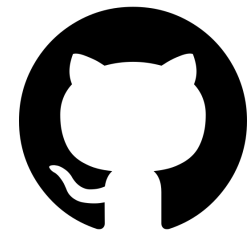
- [express-open-in-editor](#) (based on [open-in-editor](#), supports express/webpack-dev-server)
- [launch-editor-middleware](#) (based on [launch-editor](#), supports express/connect/webpack-dev-server)

Setup "open in editor" feature

For [VS Code](#) no server is needed just use pattern [vscode://file/\[file\]](#)

```
<script
  src="https://cdn.jsdelivr.net/npm/react-render-tracker"
  data-config="
    openSourceLoc: {
      pattern: 'vscode://file/[file]',
      projectRoot: '/abspath/to/git/project-name'
    }
  "
></script>
```

React Render Tracker



[react-render-tracker](#)



[Releases](#)



[Roadmap \(todo list\)](#)