React Render Tracker v0.6

Overview & Instructions

Revision November 23, 2021

react-render-tracker

Why React Render Tracker?

Filling the gap in React's tooling

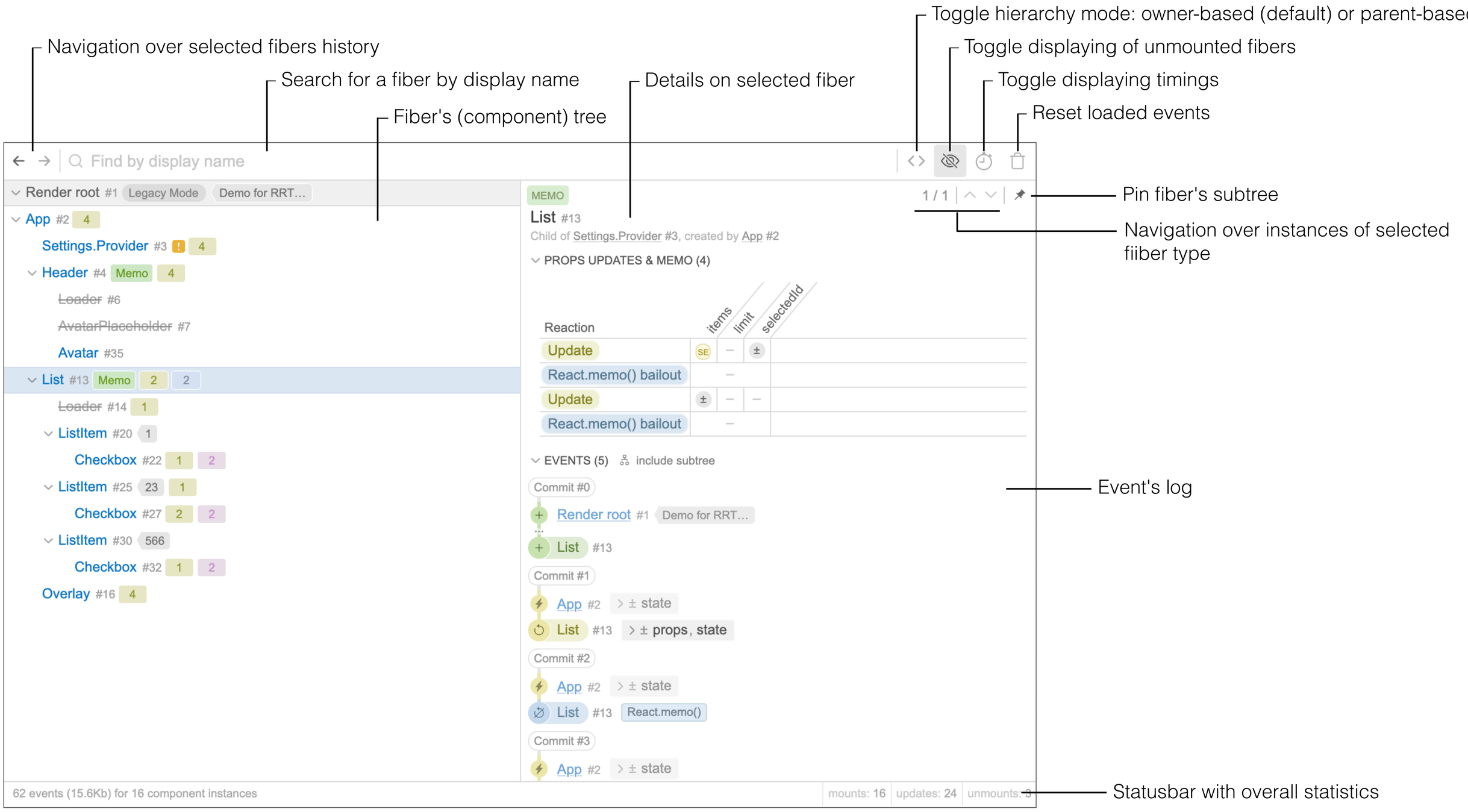
- React Devtools shows current state of components tree and components themselves, doesn't track unmounted components, lacks details on update reasons and changes in components, etc.
- Why Did You Render is focused on updates triggered by props changes only, limited in provided information (due to no 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
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
1 / 1
Pin fiber's subtree
Navigation over instances of selected fiiber type
PROPS UPDATES & MEMO (4)
Reaction
items
limit
selectedId
Update
React.memo() bailout
Update
React.memo() bailout
EVENTS (5)
include subtree
Event's log
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: 3
Statusbar with overall statistics

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.

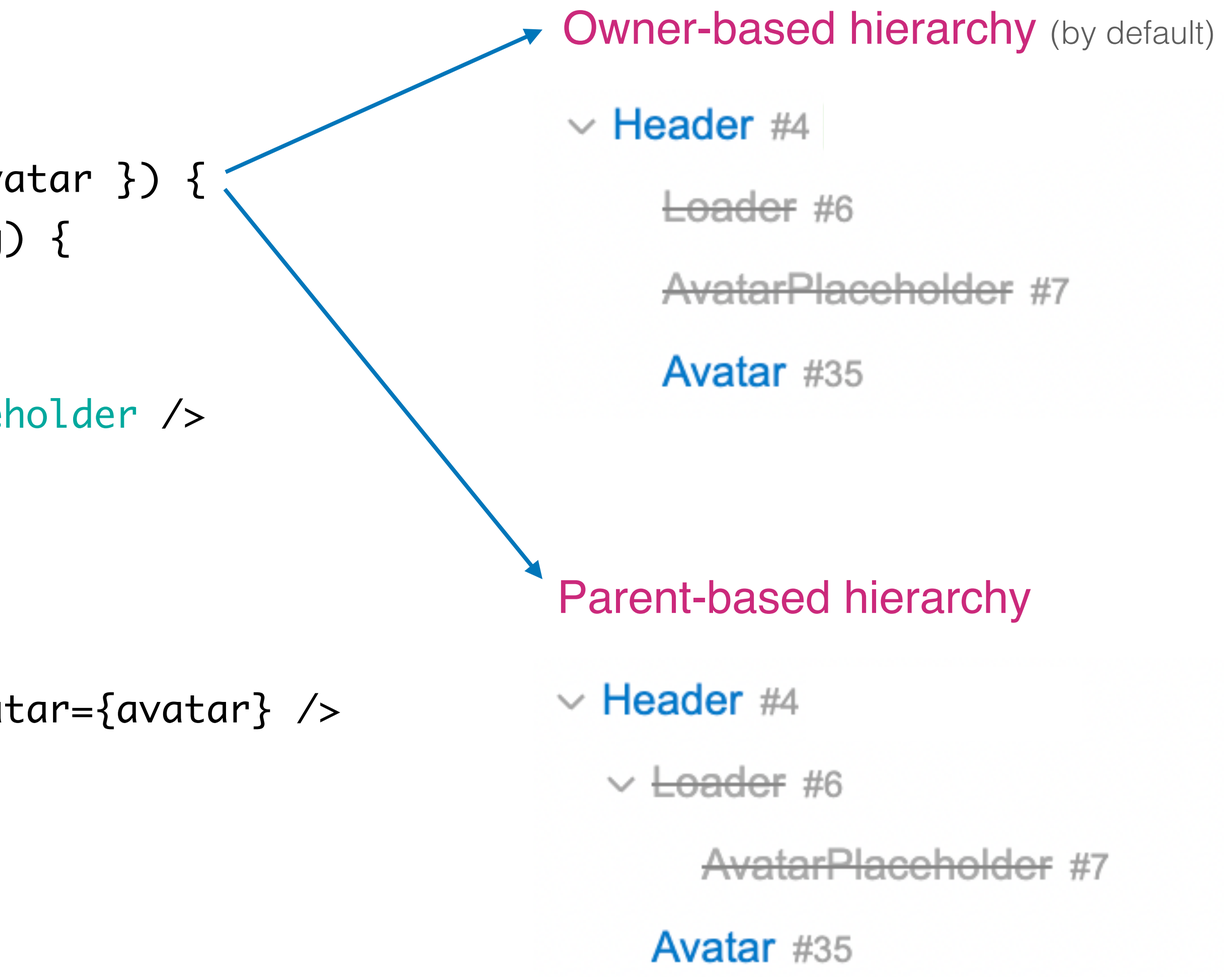
A **component** usually stands for a class or a function component, the things that are carrying a unit of app's logic and are defined by developers. That's fine to use component instead of fiber in most cases.

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

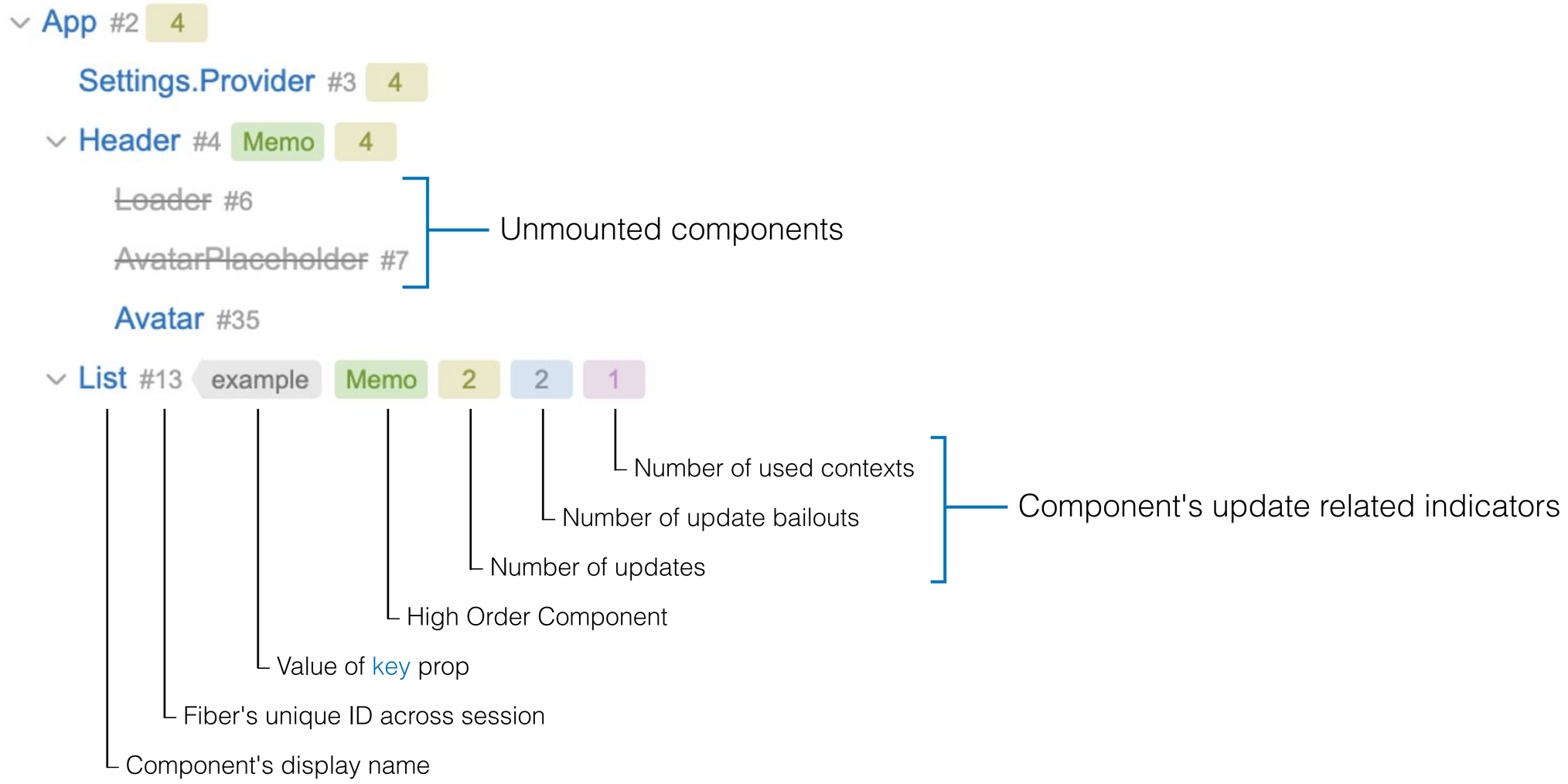
```
function Header({ avatar }) {
  if (avatar.loading) {
    return (
      <Loader>
        <AvatarPlaceholder />
      </Loader>
    );
  }

  return <Avatar avatar={avatar} />
}
```



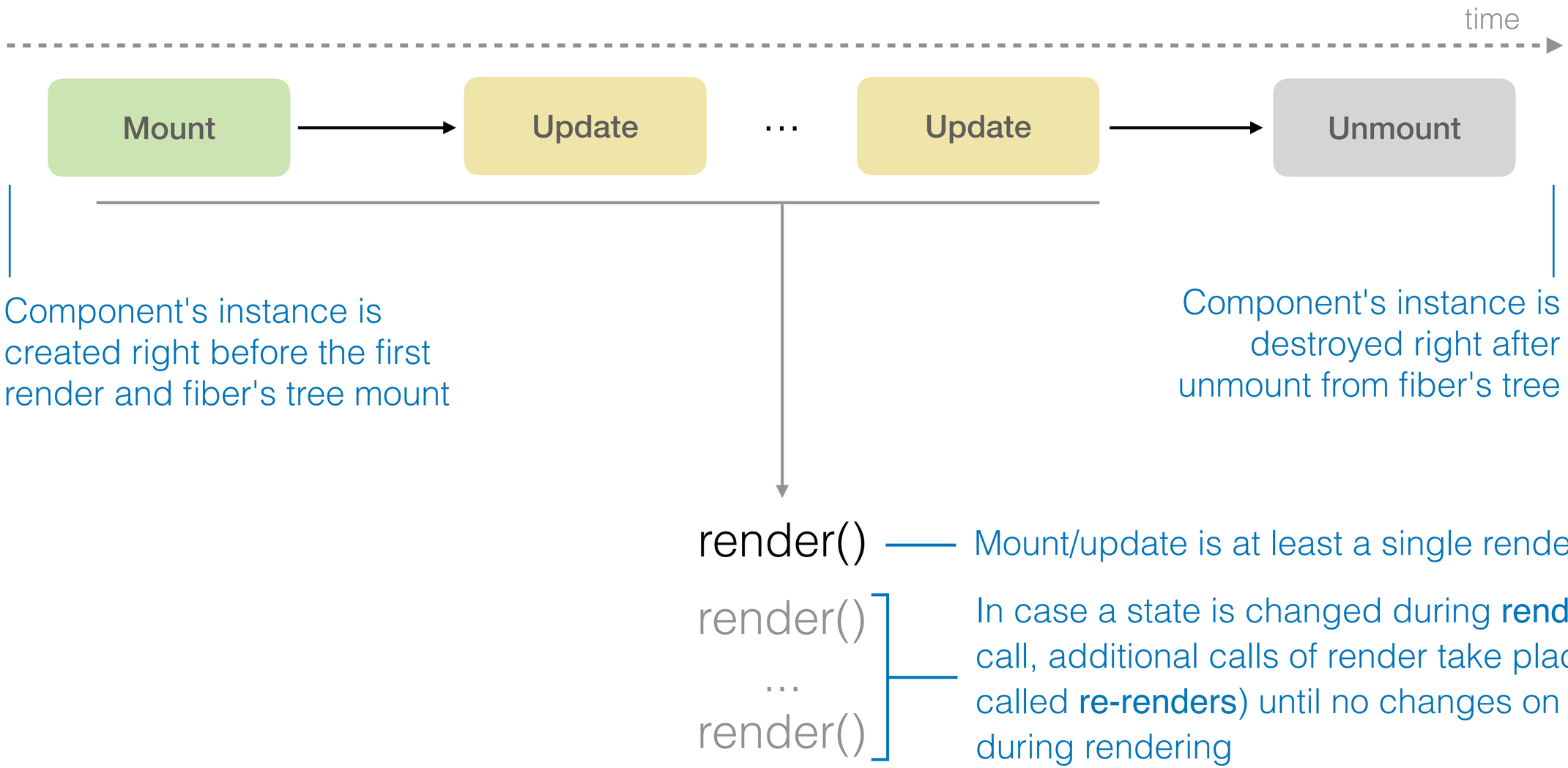
Owner-based hierarchy (or owner-ownee relationship) is more useful for understanding which components will be updated on owner's update (if no update bailout will take place).
An owner is the component that sets the props of other components. In other words, an owner is a component that creates other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

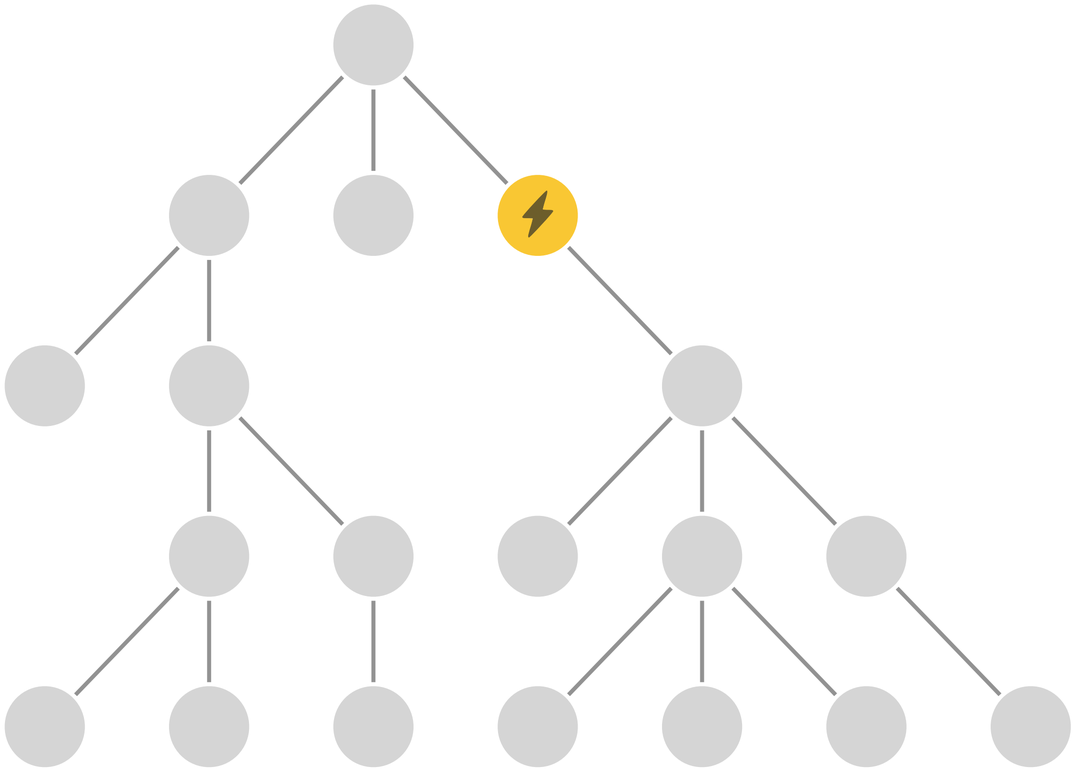
Fiber's tree


App #2 4
Settings.Provider #3 4
Header #4 Memo 4
Loader #6
AvatarPlaceholder #7
Avatar #35
List #13 example Memo 2 2 1
Unmounted components
Number of used contexts
Number of update bailouts
Number of updates
Component's update related indicators
High Order Component
Value of key prop
Fiber's unique ID across session
Component's display name

React component lifecycle

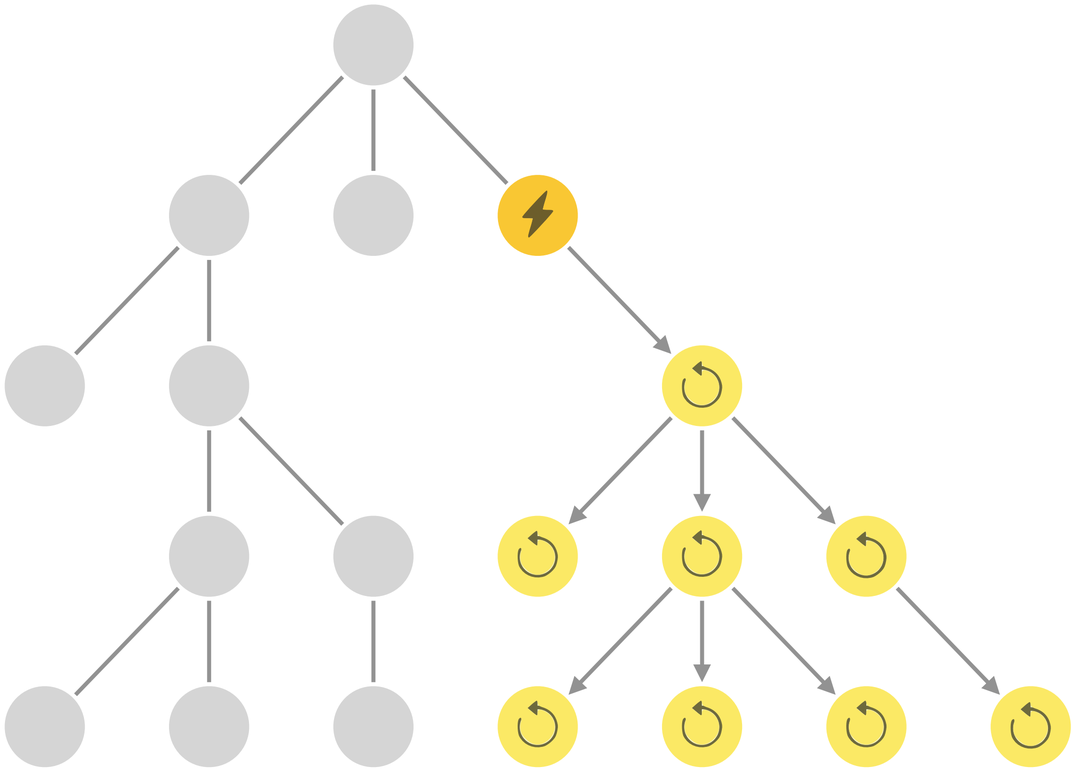


Updating fiber's tree



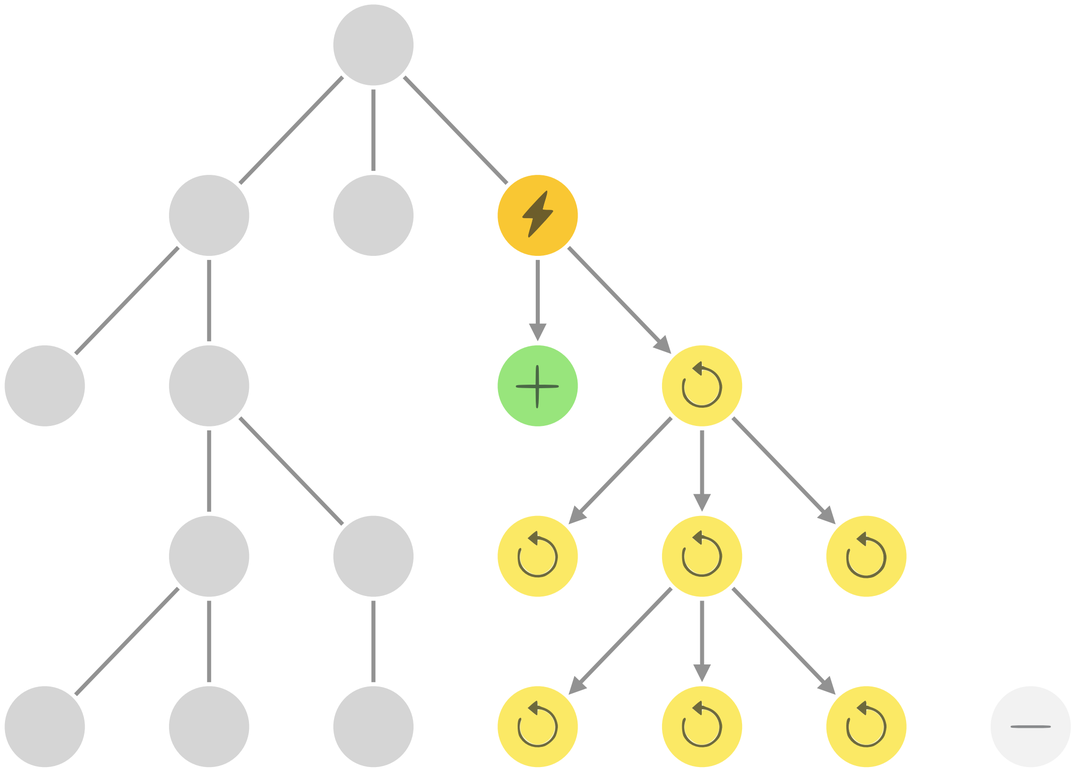
- A component update can be initiated for one of the reasons, in most cases because of state change. Such reasons can be called **fire starters**, as they lead to an update of the fiber tree (a reconciliation).

Updating fiber's tree



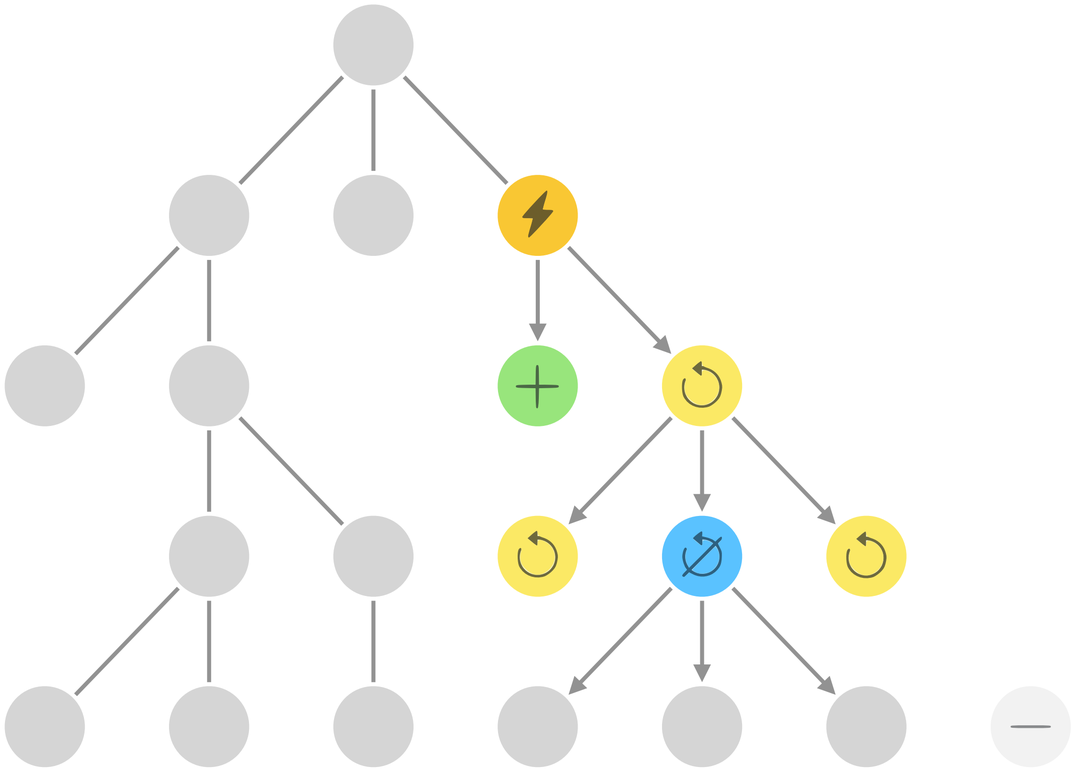
- A component update can be initiated for one of the reasons, in most cases because of state change. Such reasons can be called **fire starters**, as they lead to an update of the fiber tree (a reconciliation).
- A component update usually results in nested components being updated because of props or context value change. Such effect reasons can be called **continuators**, 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 **continuators**, 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 **continuators**, 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

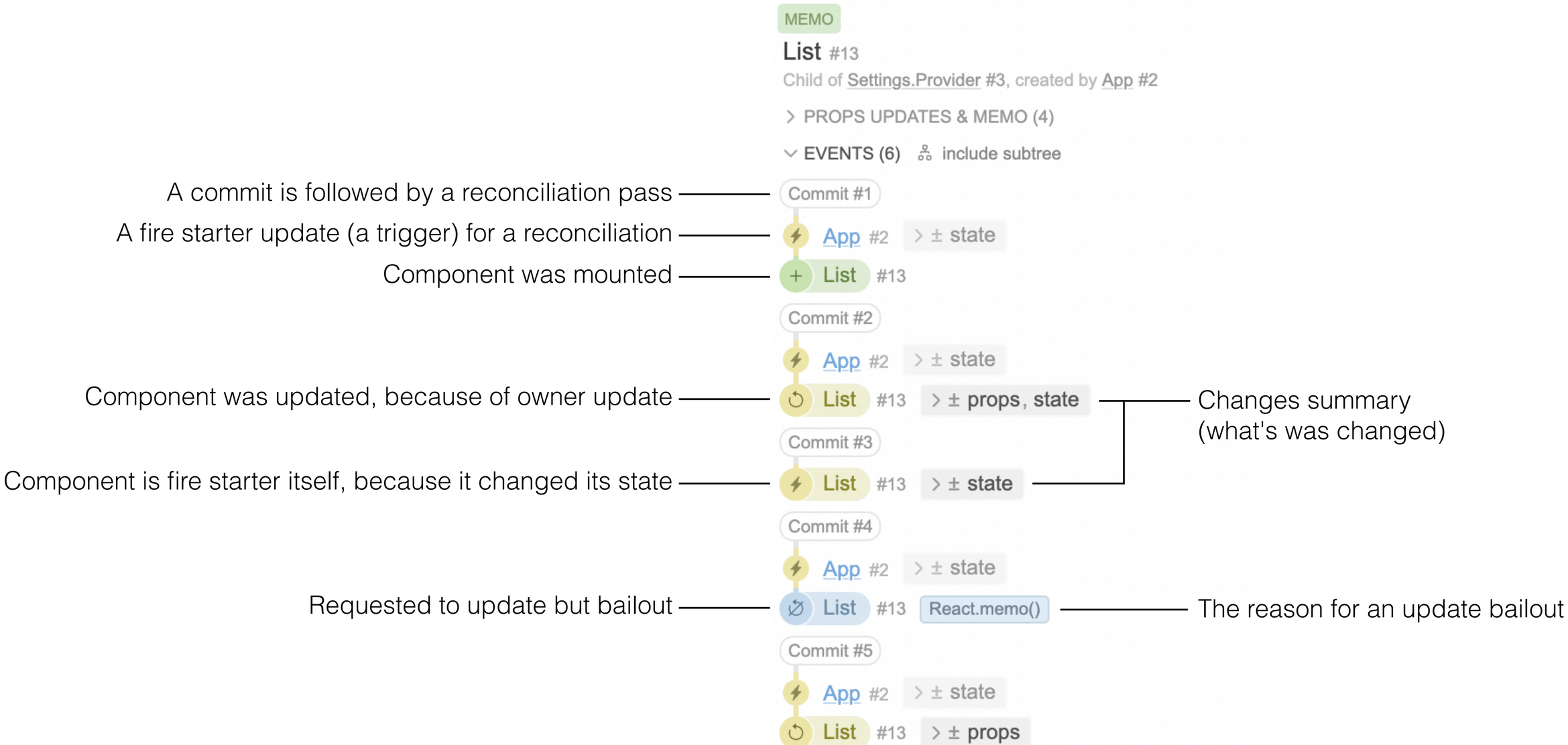
Reasons for update bailout

- ☑ 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

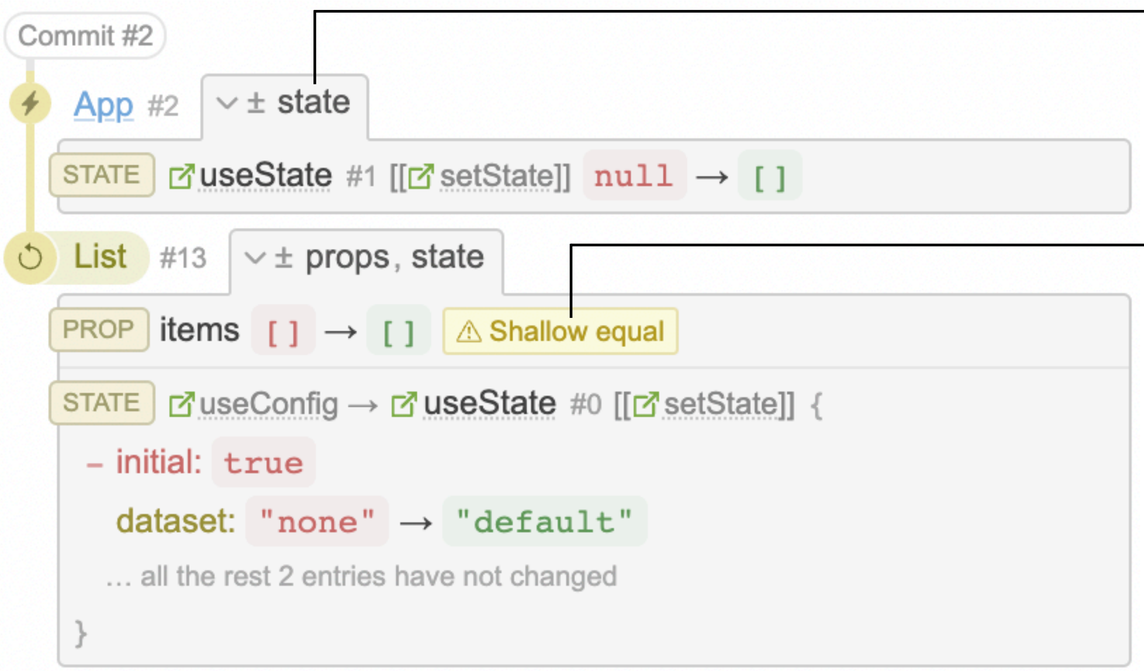
Event log


MEMO
List #13
Child of Settings.Provider #3, created by App #2
PROPS UPDATES & MEMO (4)
EVENTS (6)
include subtree
A commit is followed by a reconciliation pass
Commit #1
A fire starter update (a trigger) for a reconciliation
App #2
± state
Component was mounted
List #13
Commit #2
App #2
± state
Component was updated, because of owner update
List #13
± props, state
Changes summary
(what's was changed)
Commit #3
Component is fire starter itself, because it changed its state
List #13
± state
Commit #4
App #2
± state
Requested to update but bailout
List #13
React.memo()
The reason for an update bailout
Commit #5
App #2
± state
List #13
± props

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.

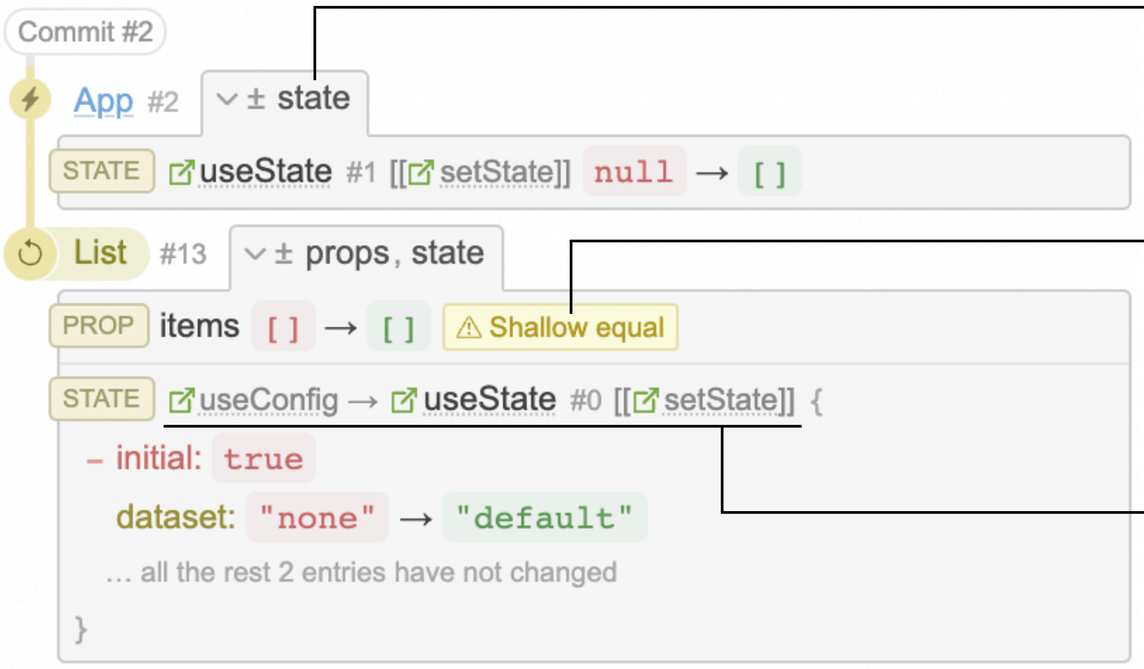
Change details



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

"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



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

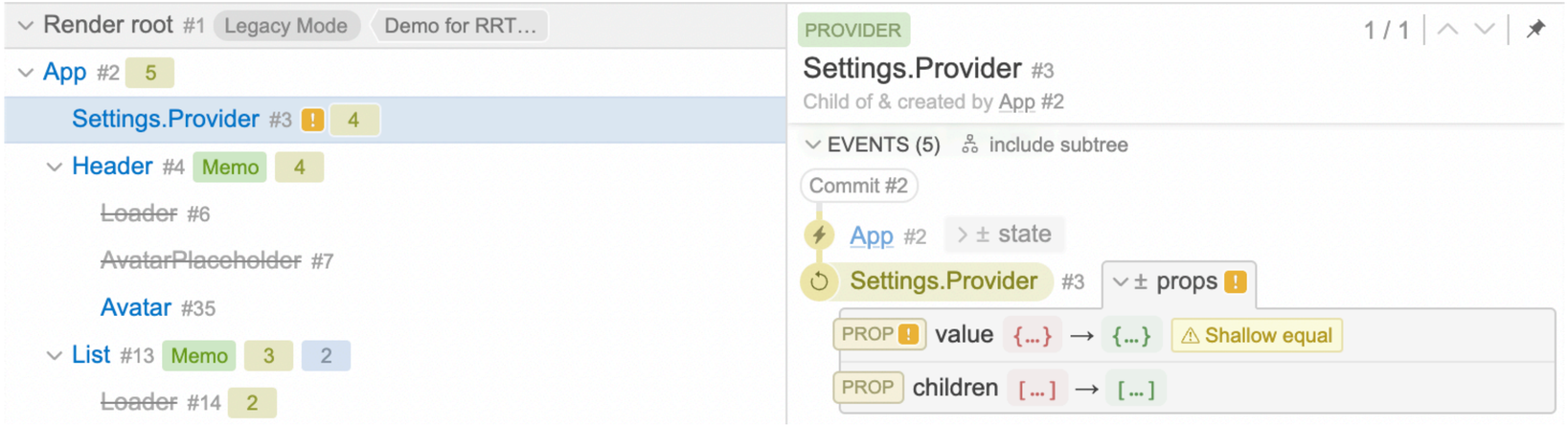
"Shallow equal" means that all entries of objects or all elements of arrays are equal, but objects or arrays are different by references. In most cases it means that we can avoid the change (if it makes sense)

useConfig → useState #0 [[setState]]

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



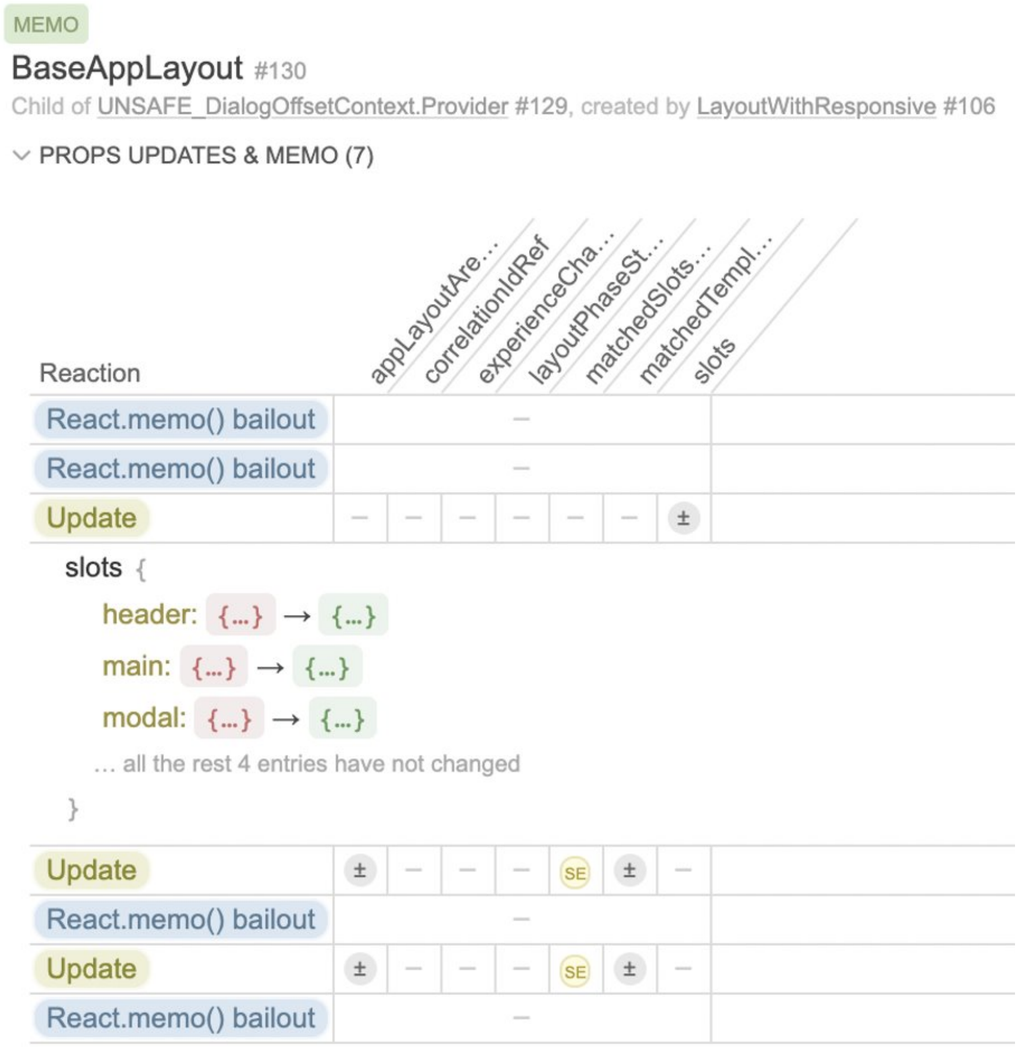
React Render Tracker indicates potential problem places with an exclamation mark icon

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



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

Usage of useMemo() and useCallback() hooks

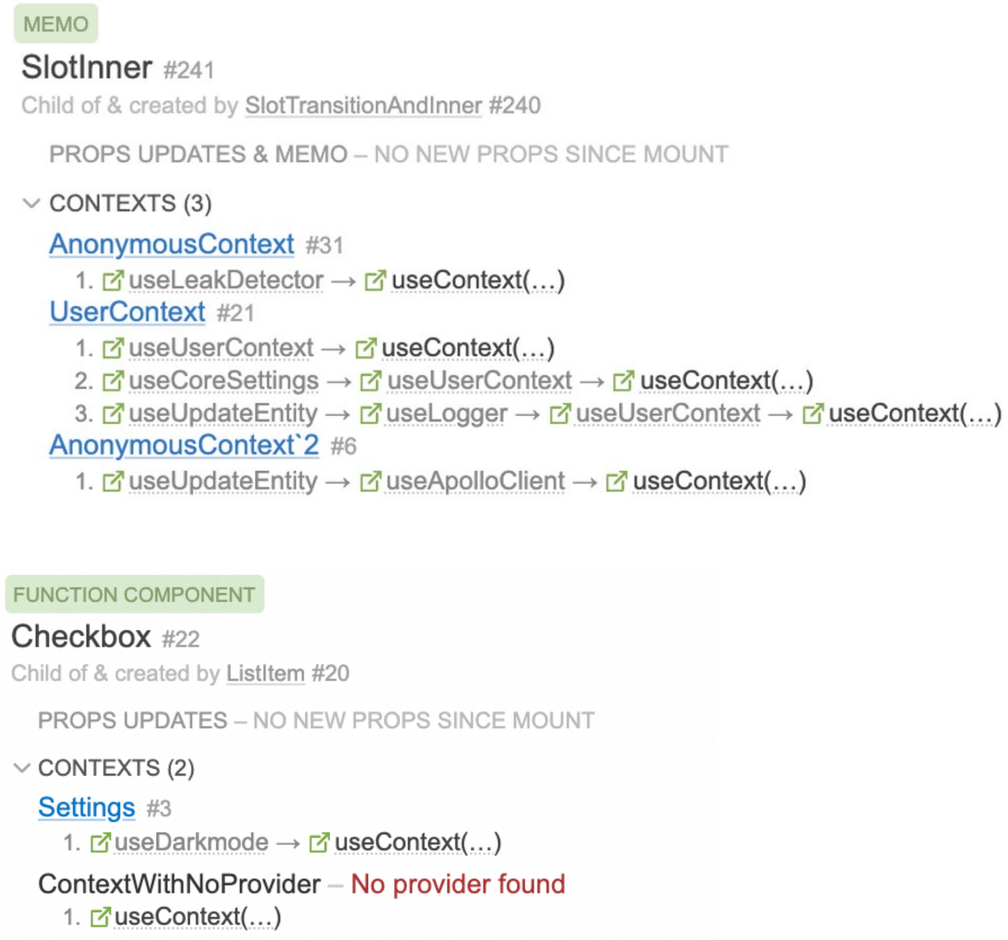
Answers the questions:

- 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



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?

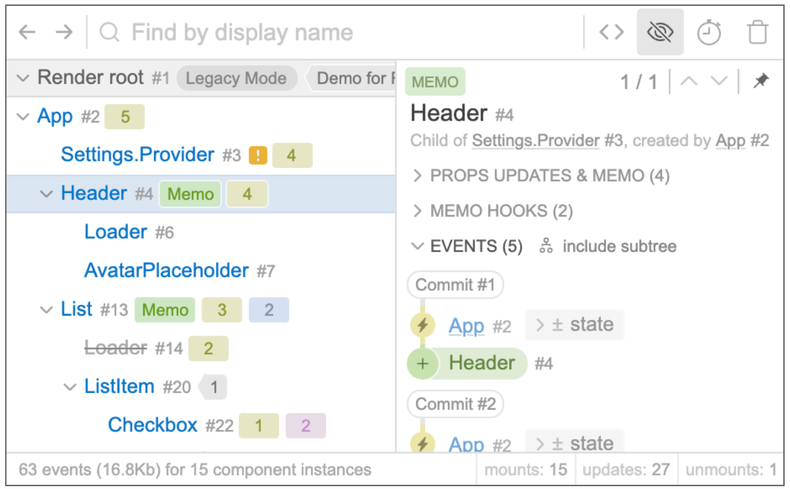
Event diffings

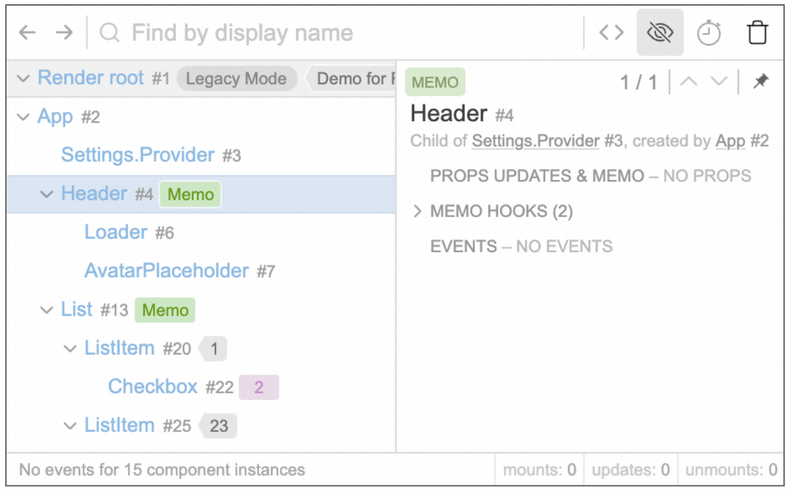
Problem

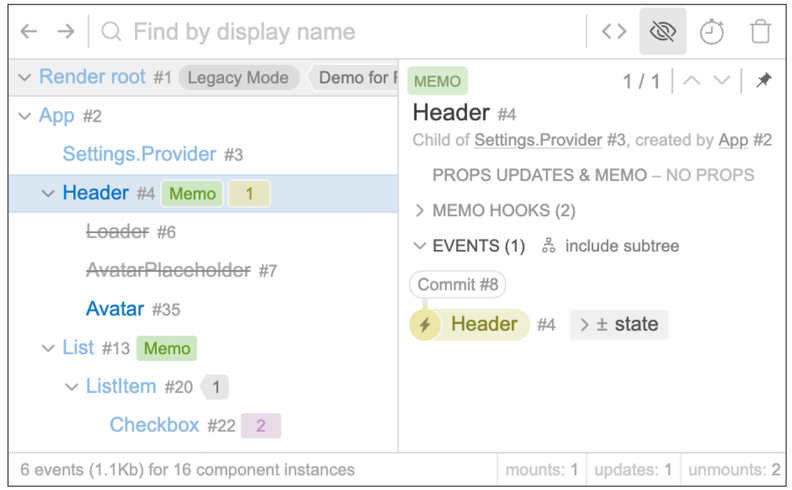
Need to know what's changing in the app on some actions like a click on a button

Solution

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





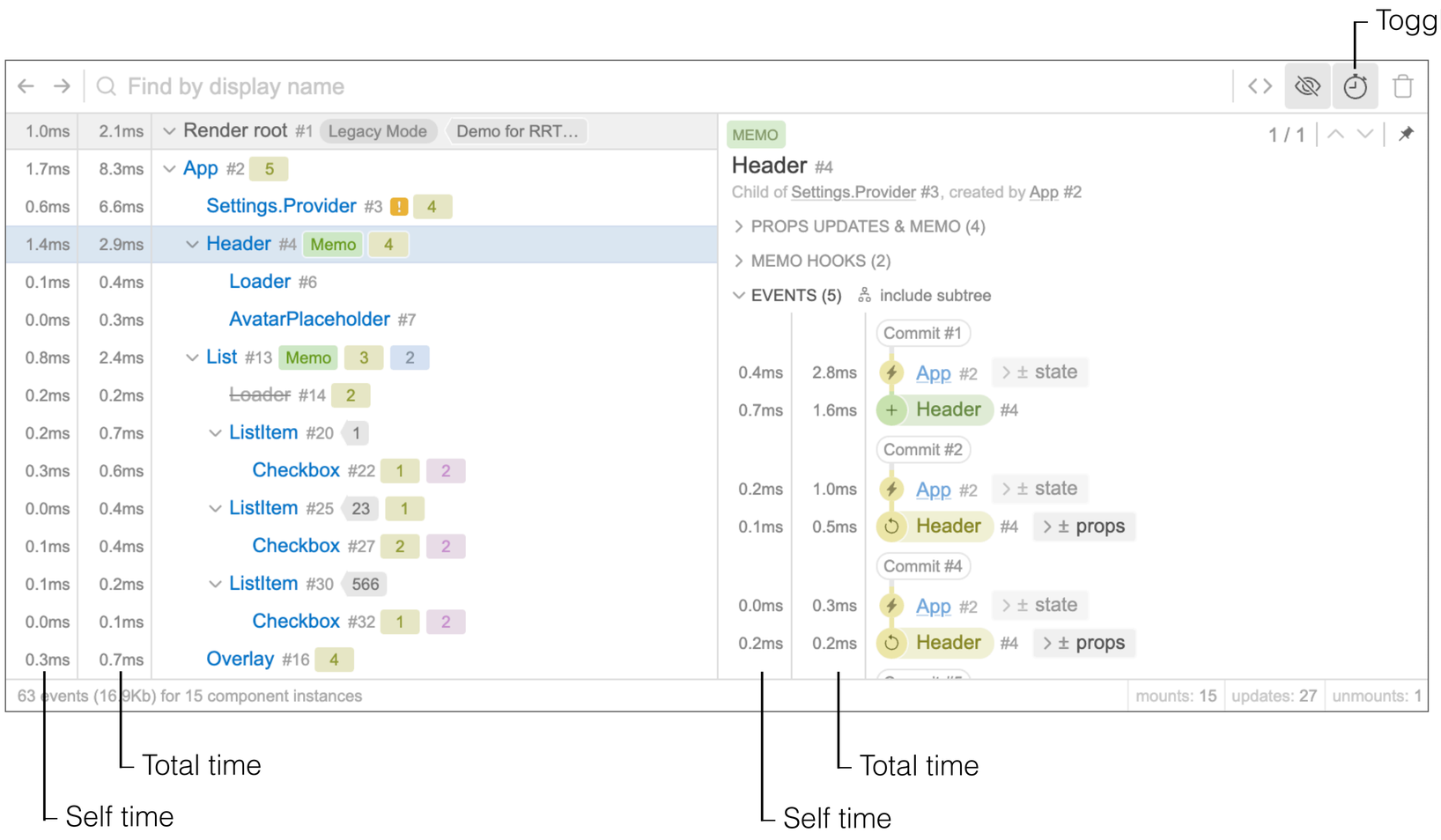


1. Click on button with trash bin icon to reset loaded events. This will also remove any unmounted components.

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.

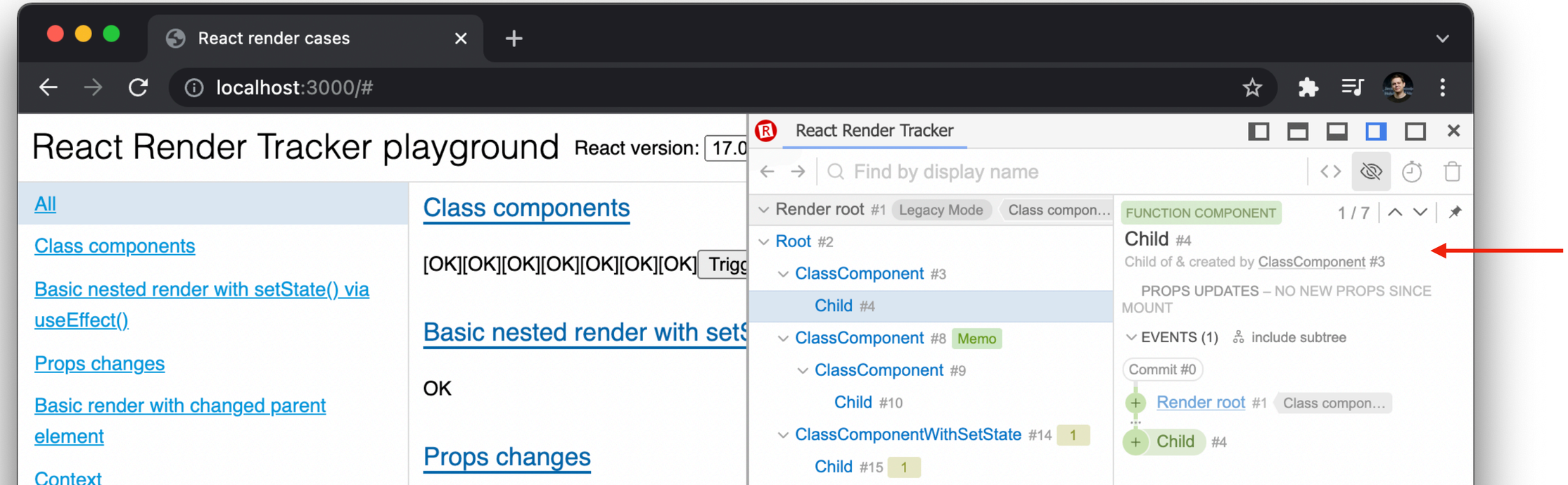
The next step is to open UI in one of the ways that works best for your case.

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>
```

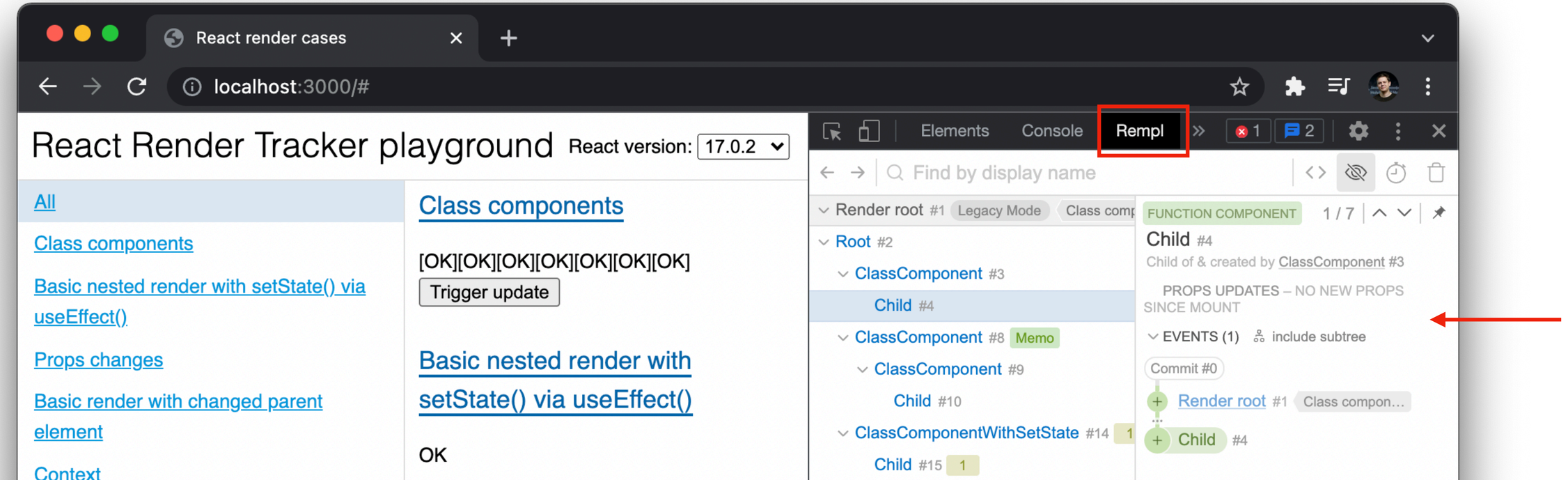


Open UI in browser's devtools

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

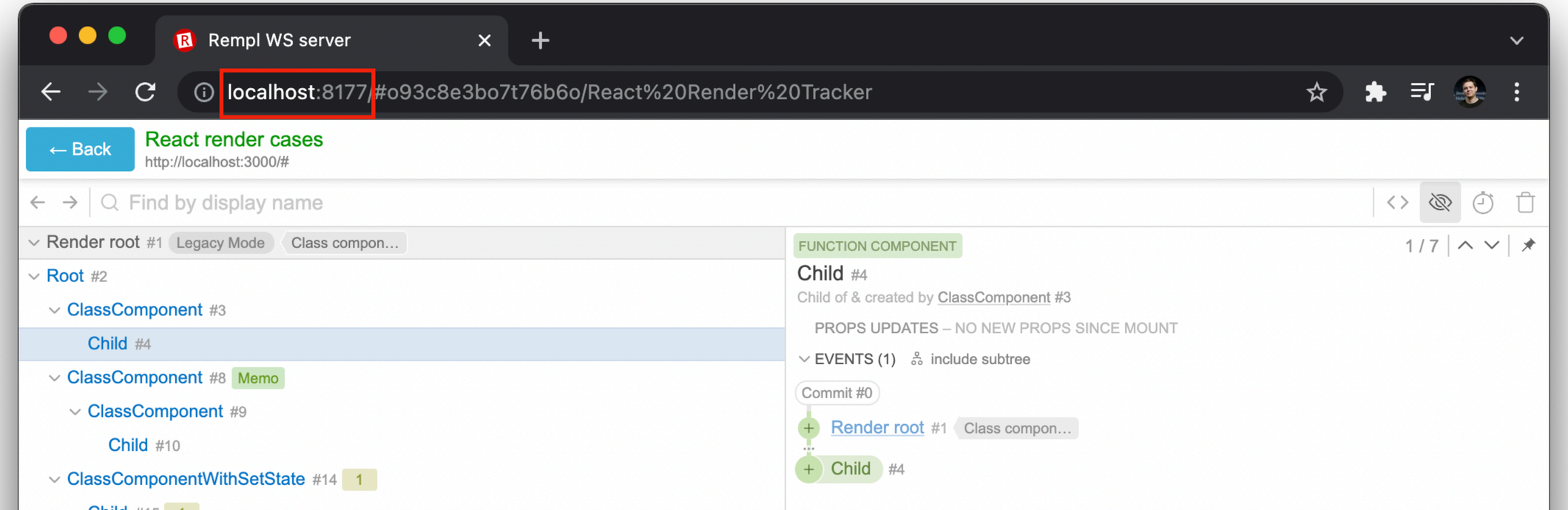
1. Install and launch Rempl server

```
> npm install --global rempl-cli
> rempl
Init standalone version of rempl
Server run at http://localhost:8177
```

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

```
<meta name="rempl: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)