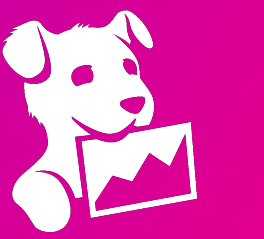
A scientific approach to performance issues

React Native Connection
June 1st, 2023

Louis Zawadzki
@zawadzkil



DATADOG

The Datadog platform

Infrastructure Monitoring

- Containers
- Serverless
- Network Performance Monitoring
- Network Device Monitoring
- Cloud Cost Management

Application Performance Monitoring

- Distributed Tracing
- Error Tracking
- Continuous Profiler
- Database Monitoring
- Universal Service Monitoring

Digital Experience Monitoring

- Synthetics
- Real User Monitoring
- Session Replay

Log Management

- Observability Pipelines
- Sensitive Data Scanner
- Audit Trails
- Log Forwarding

Security

- Cloud Security Management
- Application Security Management
- Cloud SIEM

Developer Experience

- CI Visibility
- Continuous Testing

Watchdog AI

Insights • Impact Analysis • Root Cause Analysis • Anomaly Detection • Alerts • Correlation • Optimizations

Shared Platform Services

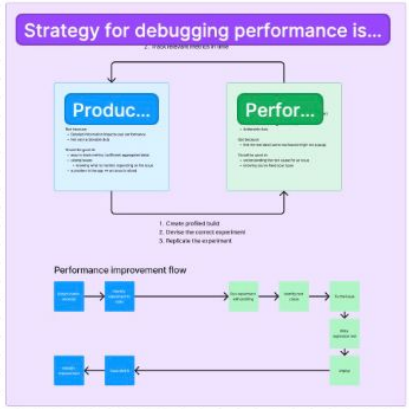
Collaboration • Dashboards • Mobile • Agents • Notebook • Workflows • Open Telemetry • Service Catalog

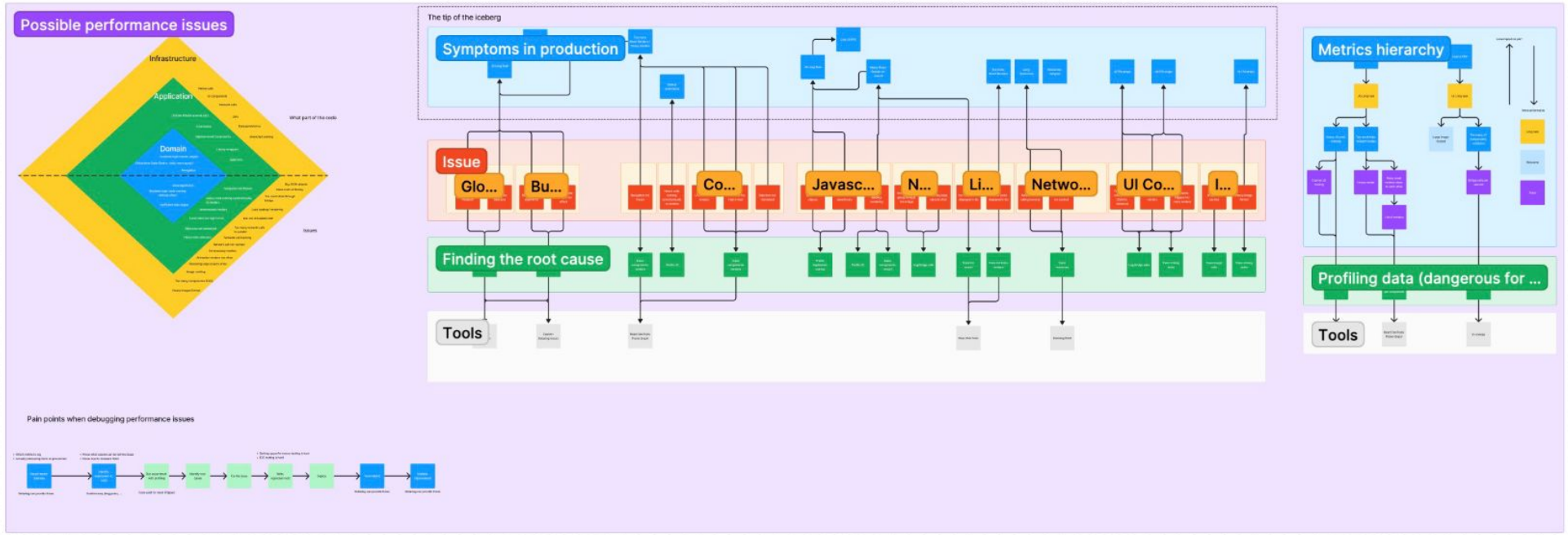
UNIFIED METRICS, LOGS, TRACES

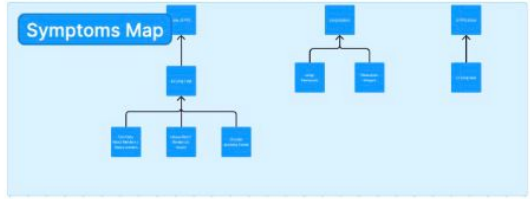
600+ INTEGRATIONS

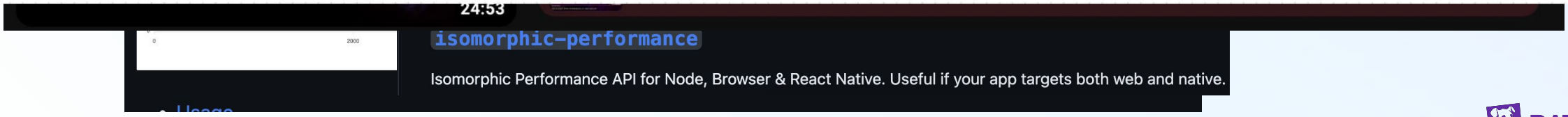
How can we improve performance tracking?

How can we improve performance tracking?





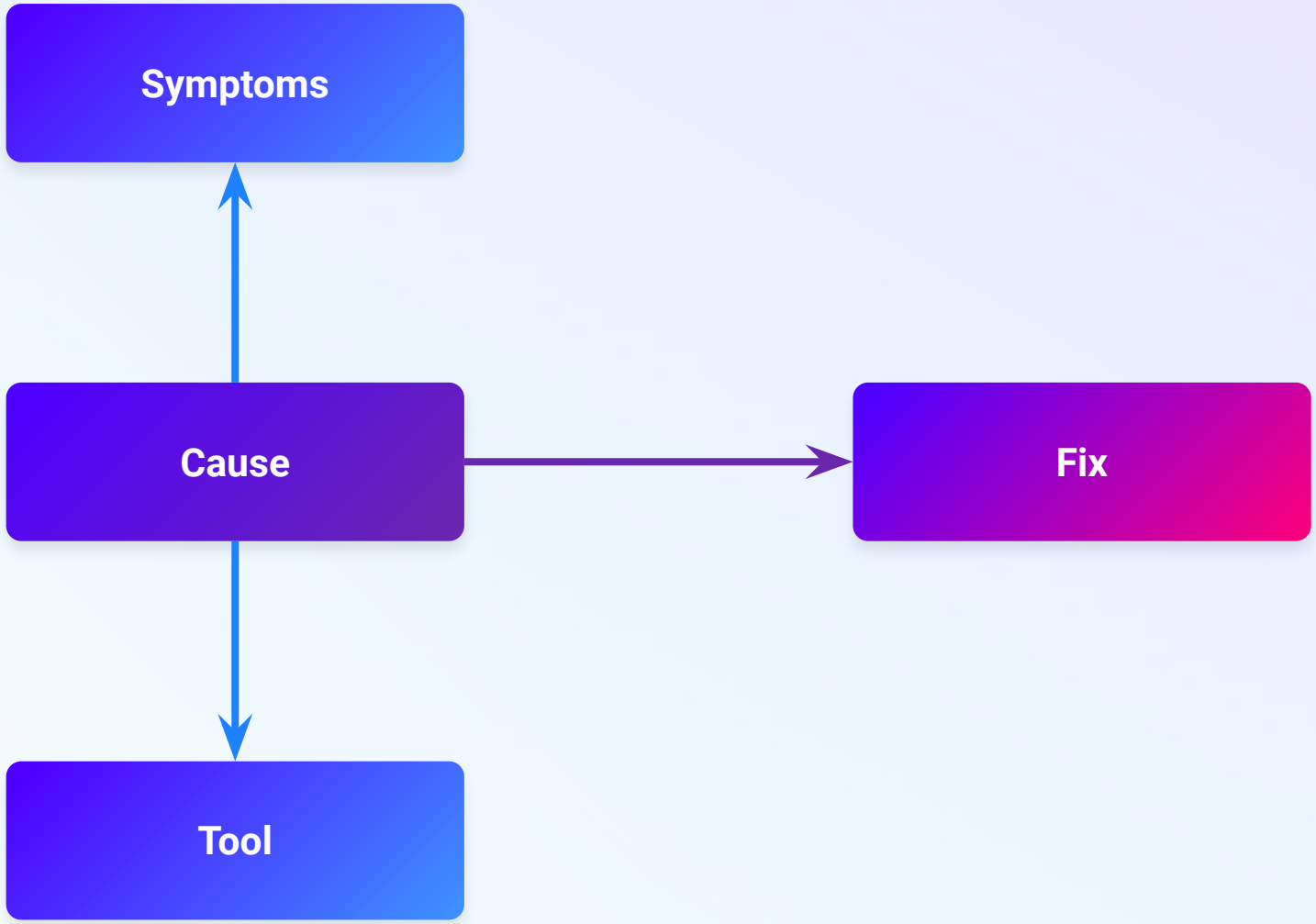




99%* of perf issues come from 25 causes

*Not scientifically proven

The model



Observing symptoms

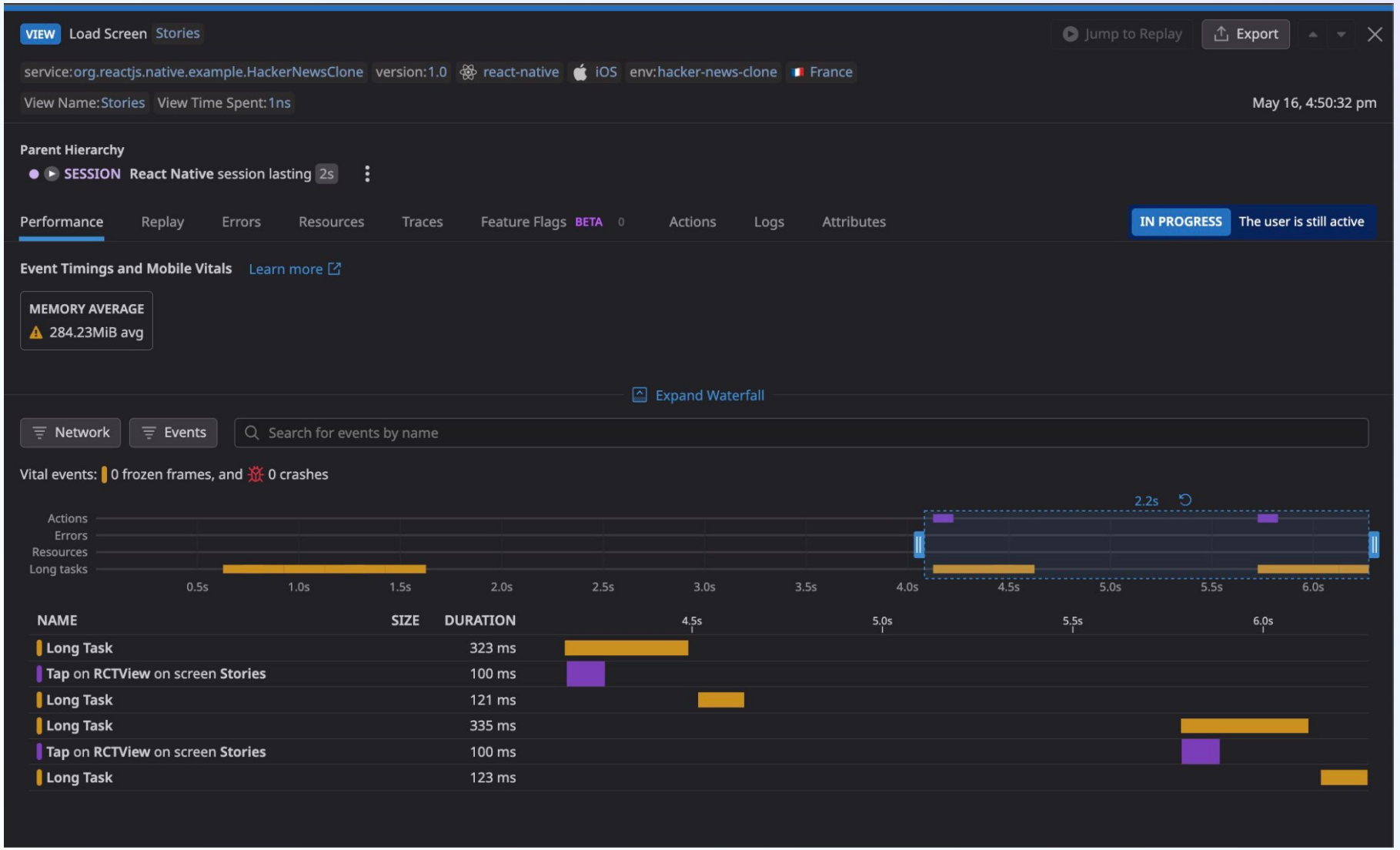
Symptoms

UI Framerate

JS Framerate

Adding observability

Adding observability



Large selectors

Too many renders

Useless renders

Selecting a group of causes

```
4   export const selectTopStories = (state: RootState): Story[] => {
5     if (state.stories.selectorType === 'top') {
6       return state.stories.ids
7         .map(id => {
8           return state.stories.stories.find(story => story.id === id);
9         })
10        .filter(story => story !== undefined)
11        .map(story => ({
12          ...story,
13          timestamp: new Date((story as Story).time * 1000).toDateString(),
14        })) as Story[];
15    }
16
17    return state.stories.stories
18      .map(story => ({
19        ...story,
20        timestamp: new Date(story.time * 1000).toDateString(),
21      }))
22      .sort((storyA, storyB) => storyA.time - storyB.time);
23  };
```

- Large selectors
- Too many renders
- Useless renders

Validate causes with experiments

What’s an experiment

Protocol

- Click on button
- Observe execution time with `console.log`

Expectations

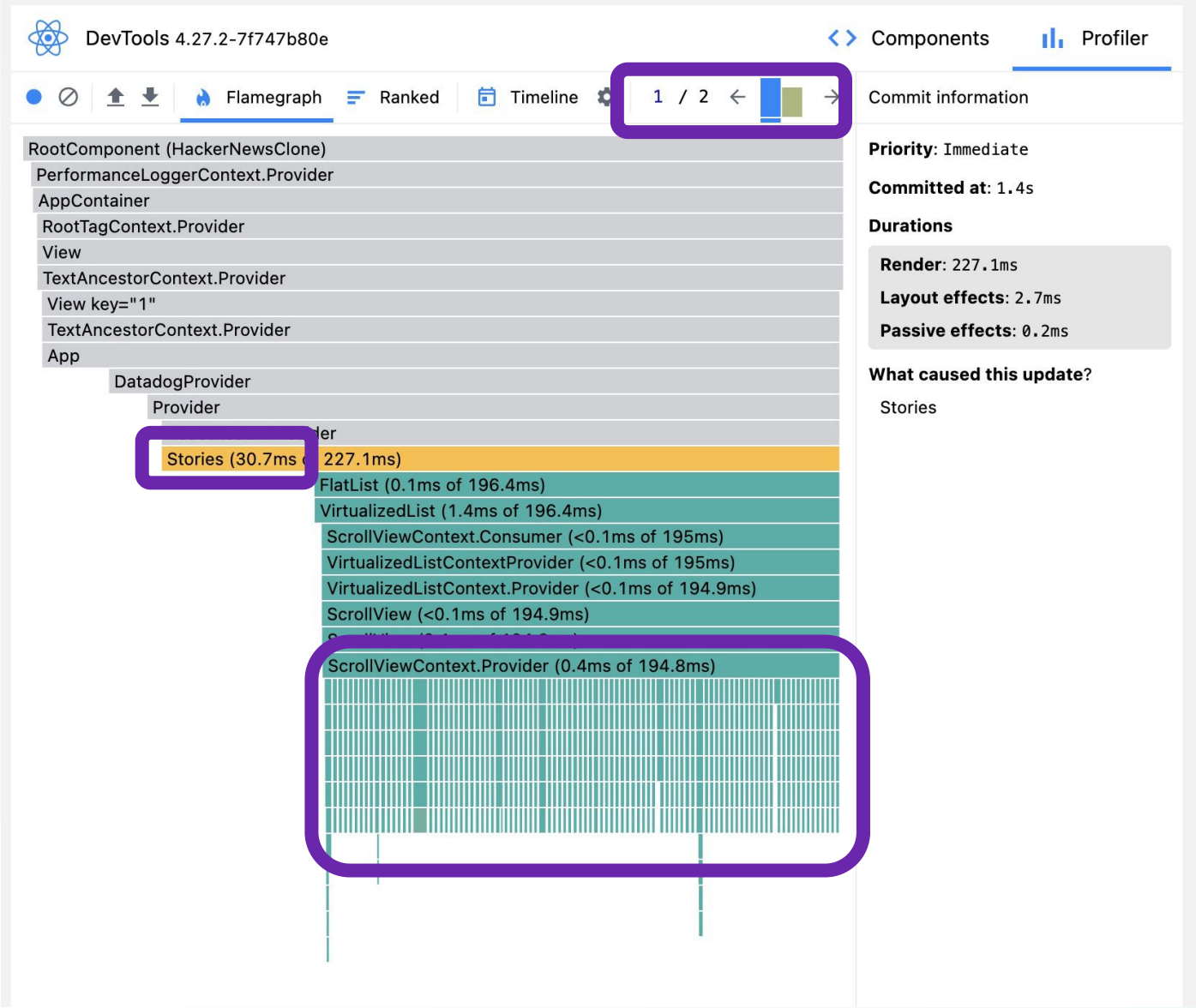
Execution time > 400ms

Running the experiment

```
export const selectTopStories = (state: RootState): Story[] => {
  const start = Date.now();

  let result: Story[] = [];
  if (state.stories.selectorType === 'top') {…
  } else {…
  }

  const end = Date.now();
  console.log('selector time', end - start);

  return result;
};
```

```
LOG  selector time 49
LOG  selector time 35
```

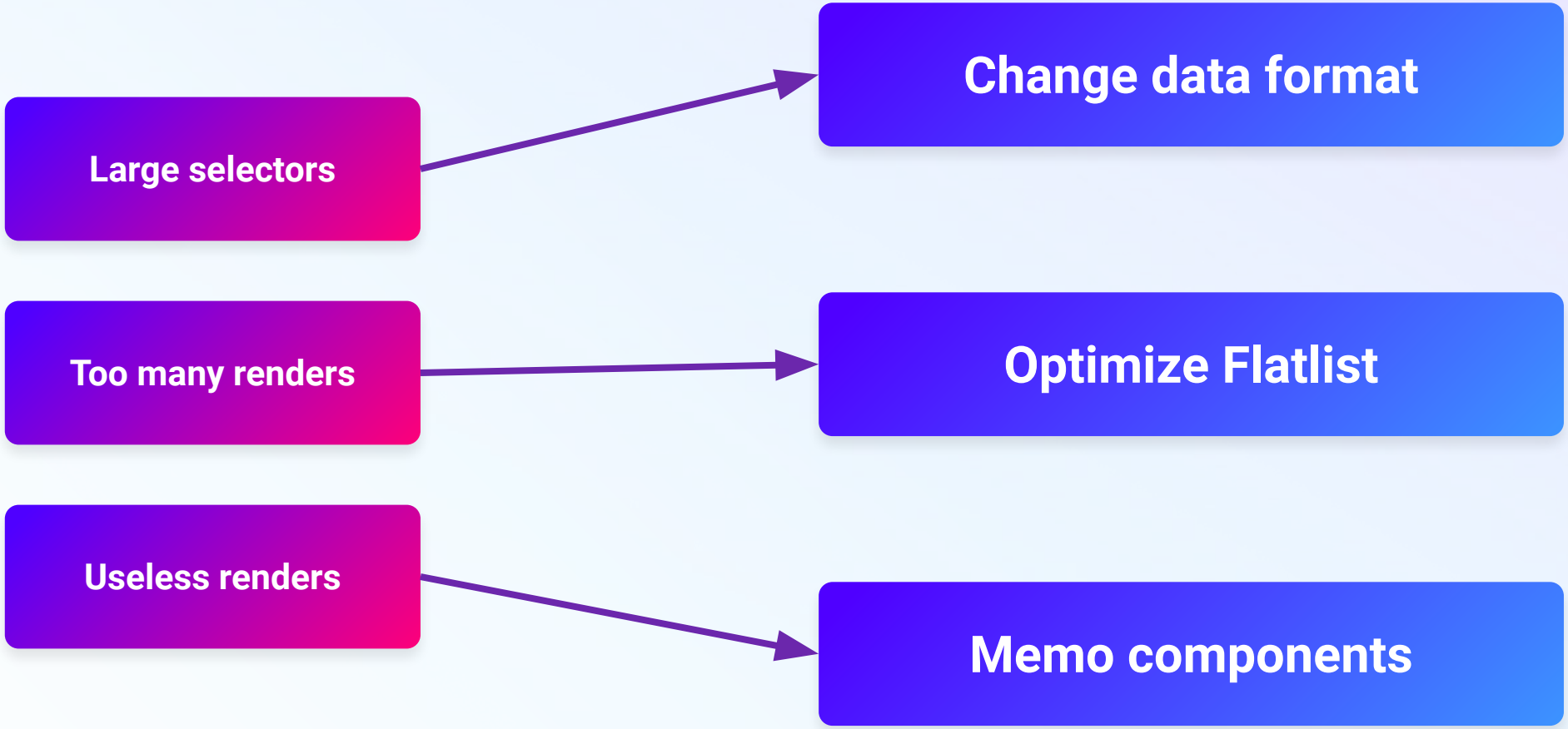
Repeat until you find all the causes



- Large selectors
- Too many renders
- Useless renders

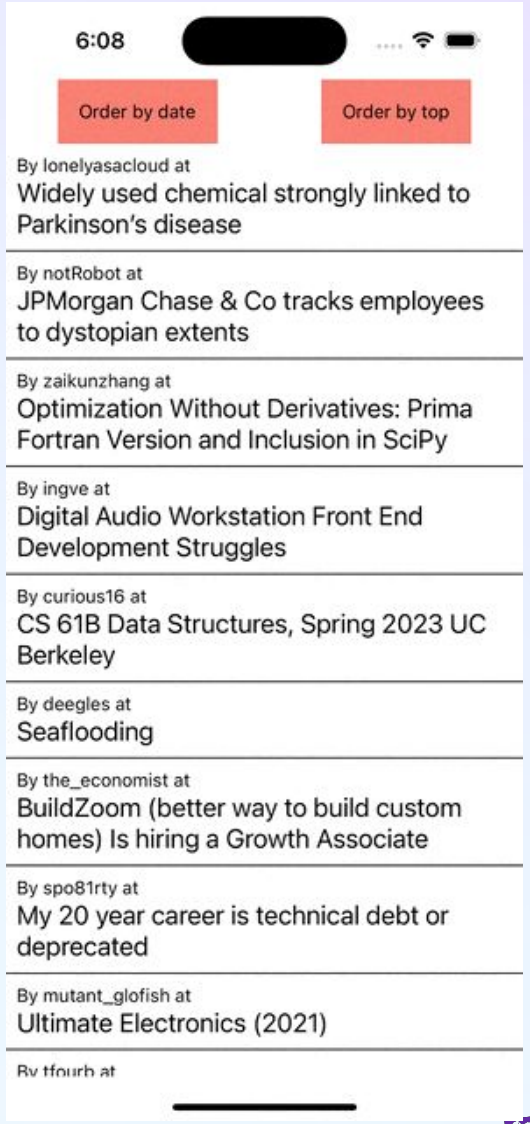
Apply fix

Apply fixes

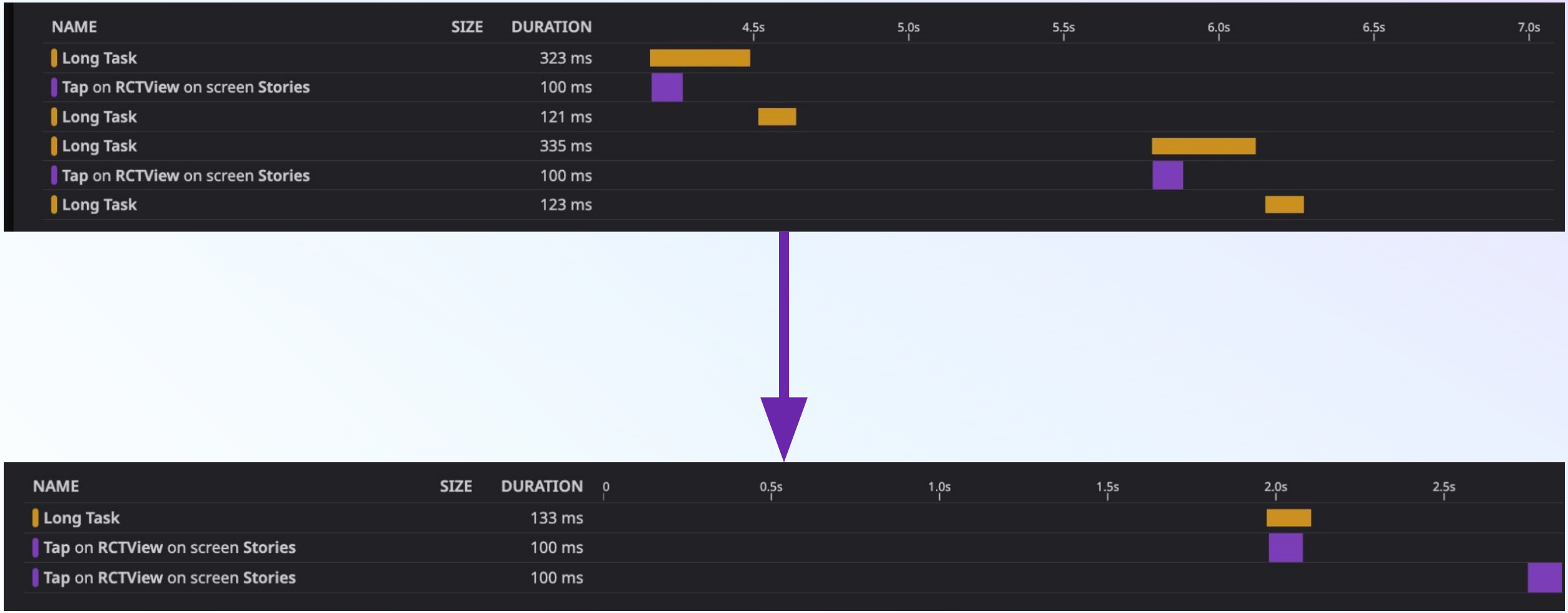

Large selectors
Too many renders
Useless renders
Change data format
Optimize Flatlist
Memo components

Check improvements

Check improvements



Check improvements



Wrapping it up

Wrapping it up

Add observability → Observe symptoms → Run experiments → Fix found causes → Check improvement

List of causes

Filter the list of causes below by the symptoms you see.

Table

↗ Causes

↗ Symptoms

Aa Name	Category	↗ Symptoms
Heavy data mutation	State data	Low JS framerate JS Long tasks JS code blocking thread
Heavy data selectors	State data	Low JS framerate

Thank you!

Louis Zawadzki
@zawadzkil

DATADOG