Performance in React (Native)

Anna Doubkova @ Hive

@lithinn

Patterns vs pre-optimisation

Issues

The actual ones

Slow startup time

Time to first interaction

Animation jagged/slow

Slow response to user action

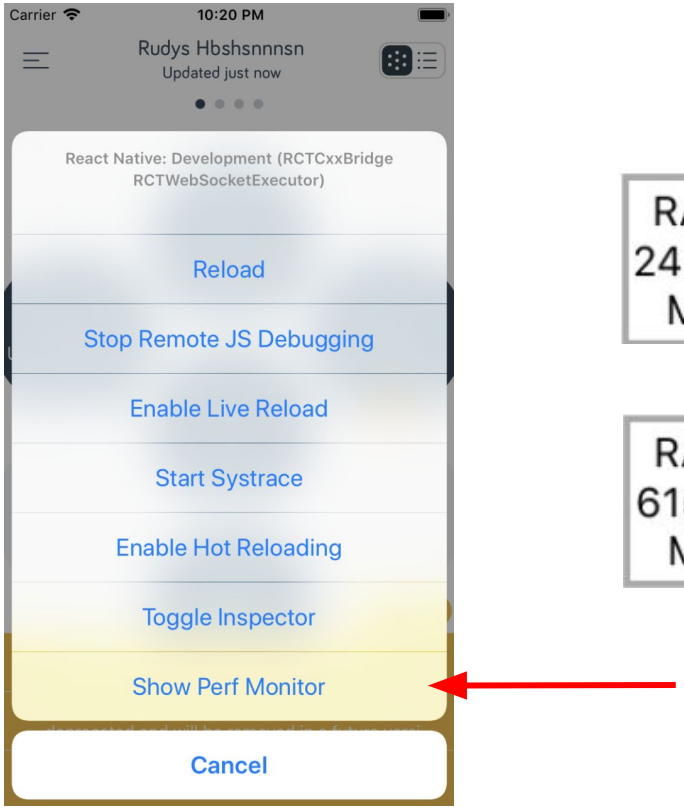
The weak link?

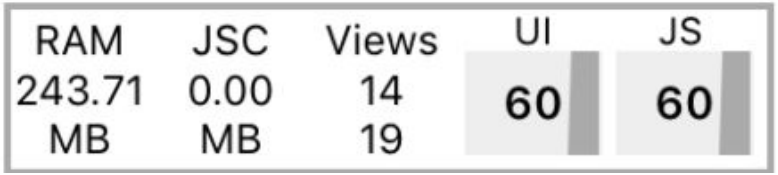
UI Thread

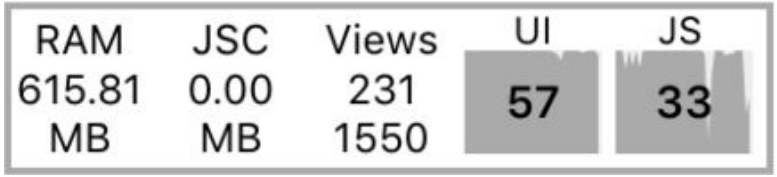
RN Bridge

JS Thread

Looking for the culprit





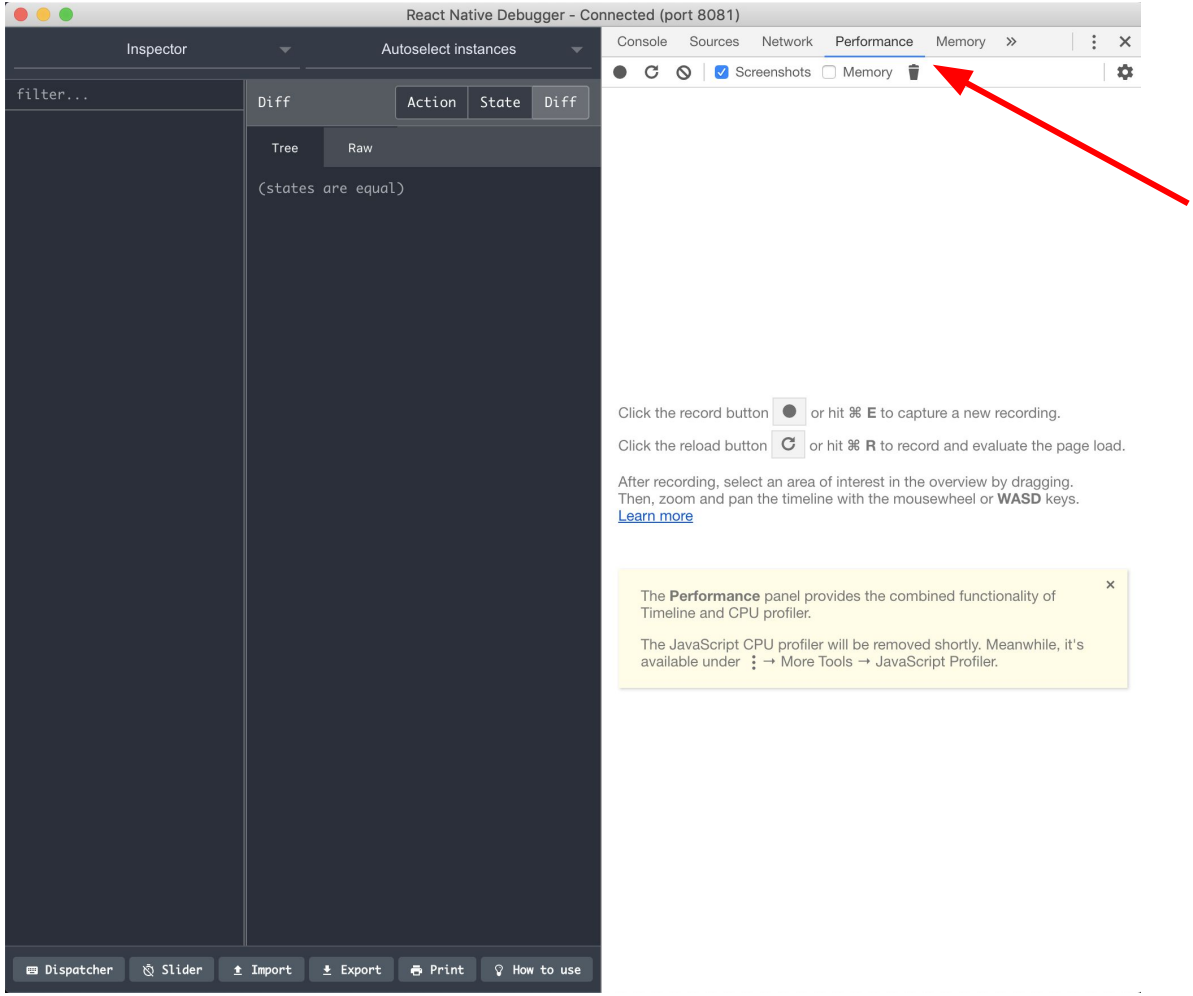


@lithinn

Measuring JS performance

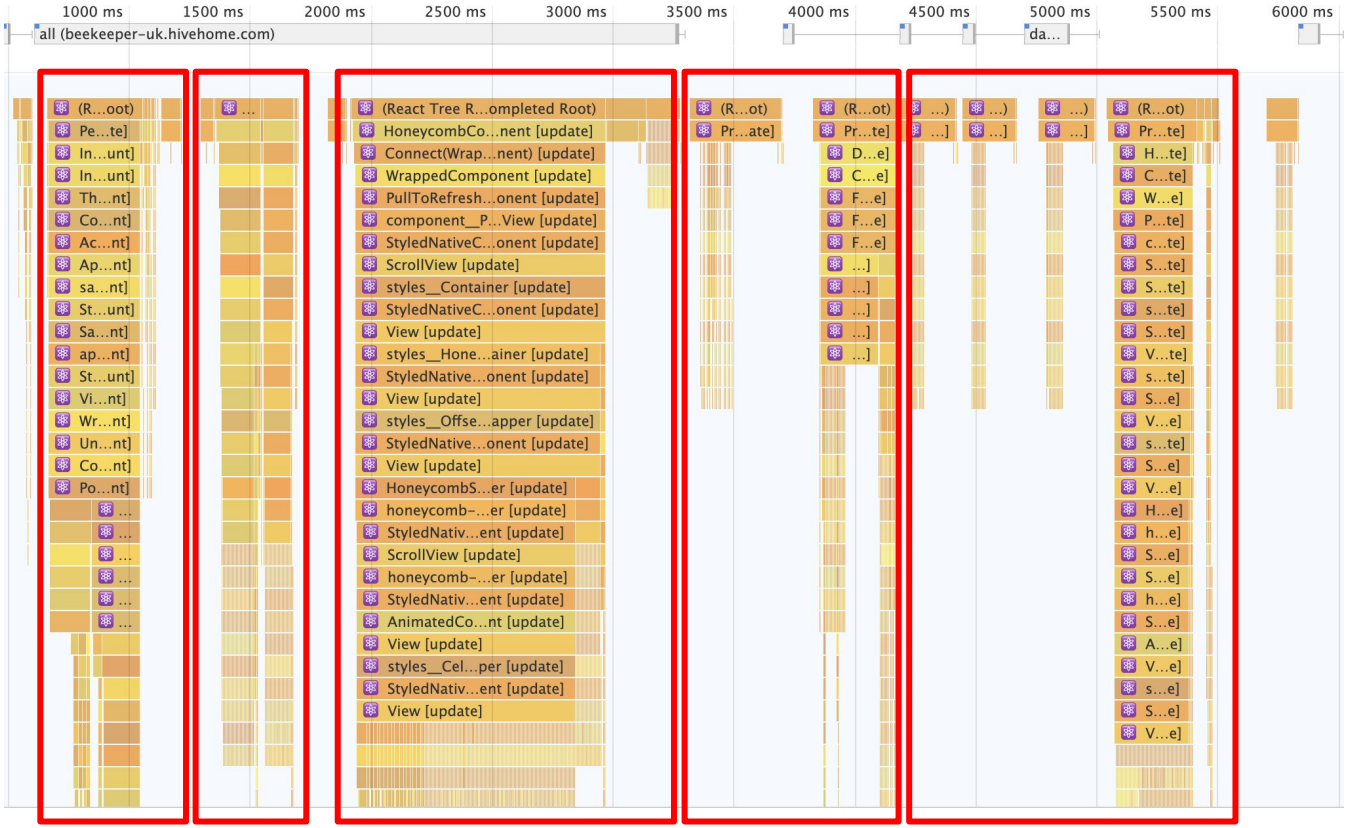

Carrier
9:49 AM
Rudys Home
Updated 19 minutes ago
Plug 2 charles
22.3°
Upgraded zone...
Hot Water
20.9°
Zone 3
Warm White Li...
Offline
VIEW WITH 2...
Edit
Add

@lithinn


React Native Debugger - Connected (port 8081)
Inspector
Autoselect instances
filter...
Diff
Action
State
Diff
Tree
Raw
(states are equal)
Dispatcher
Slider
Import
Export
Print
How to use
Console
Sources
Network
Performance
Memory
Screenshots
Memory
Click the record button or hit ⌘ E to capture a new recording.
Click the reload button or hit ⌘ R to record and evaluate the page load.
After recording, select an area of interest in the overview by dragging.
Then, zoom and pan the timeline with the mousewheel or WASD keys.
Learn more
The Performance panel provides the combined functionality of Timeline and CPU profiler.
The JavaScript CPU profiler will be removed shortly. Meanwhile, it's available under ⋮ → More Tools → JavaScript Profiler.

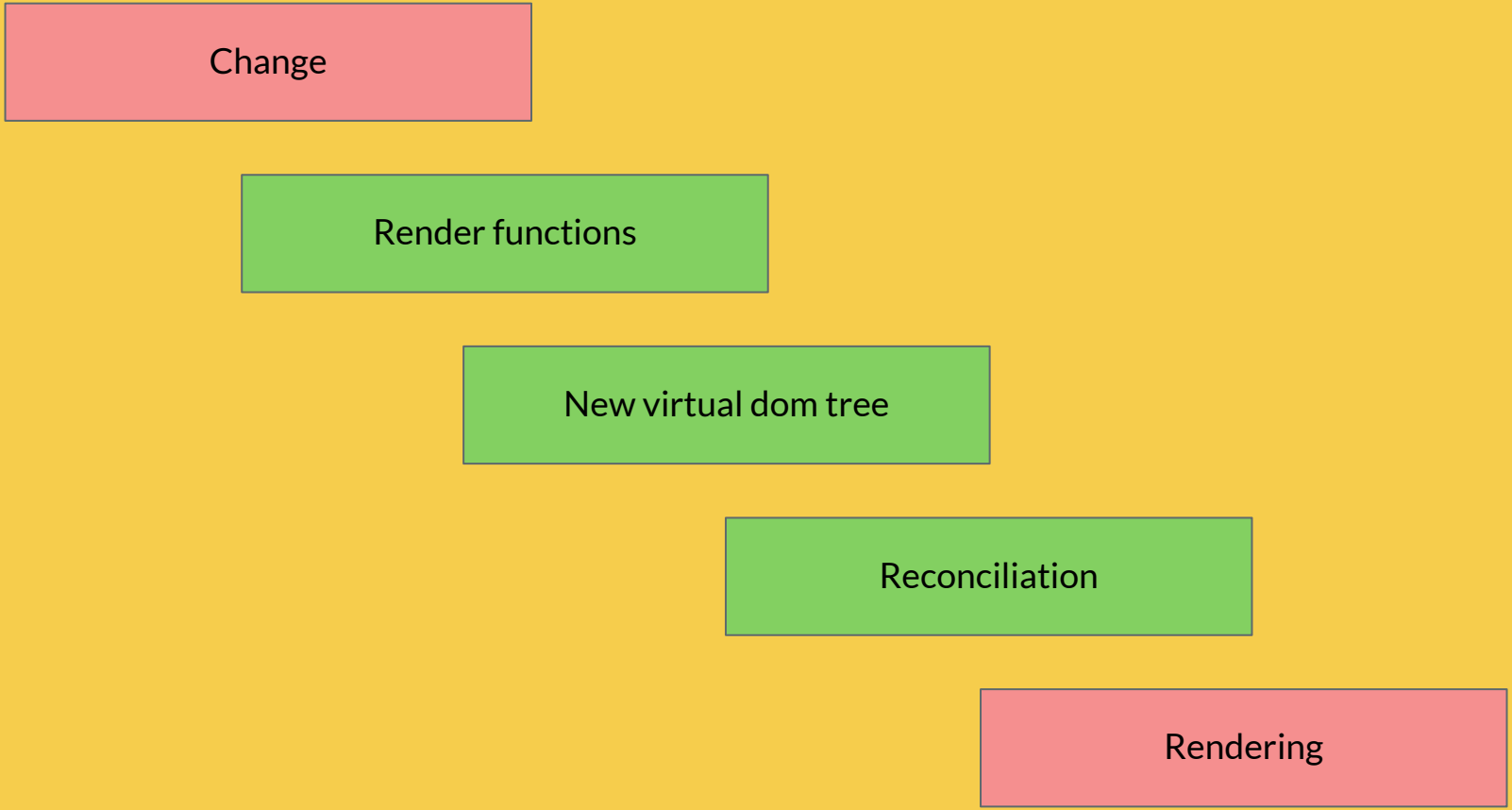
@lithinn

Measuring JS performance



@lithinn

State change


Change
Render functions
New virtual dom tree
Reconciliation
Rendering

@lithinn

Shou

date

@lithinn

```
// @flow
import * as React from 'react';
import { Text } from 'react-native';

type Props = {
    name: string,
    value: string,
};

class Component extends React.Component<Props> {
    render() {
        const { name, value } = this.props;
        return (
            <Text>
                {name} - {value}
            </Text>
        );
    }
}

export default Component;
```

@lithinn

```
shouldComponentUpdate(nextProps) {
    return nextProps.name !== this.props.name;
}

render() {
    const { name, value } = this.props;
    return (
        <Text>
            {name} - {value}
        </Text>
    );
}
```

@lithinn

```
type Props = {
    name: string,
    value: string,
    description: string,
};

class Component extends React.Component<Props> {
    shouldComponentUpdate(nextProps) {
        return nextProps.name !== this.props.name;
    }

    render() {
        const { name, value, description } = this.props;
        return (
            <Text>
                {name} - {value} - {description}
            </Text>
        );
    }
}
```

@lithinn

Reconciler

- Change node type -> re-mount
- Change node prop/state -> re-render self & children
  - Shallow equal
- Change propagates to all children

Node type change

```
import * as React from 'react';

type WrapperProps = { children: React.Node };
const Wrapper1 = ({ children }: WrapperProps) => <div>{children}</div>;
const Wrapper2 = ({ children }: WrapperProps) => <div class="two">{children}</div>;

type ComponentProps = { type: string };
const Component = ({ type }: ComponentProps) => {
    const Wrapper = type === 'one' ? Wrapper1 : Wrapper2;
    return <Wrapper>Hello</Wrapper>;
};

export const Wrapped = () => <Component type="one" />;
```

@lithinn

Prop change

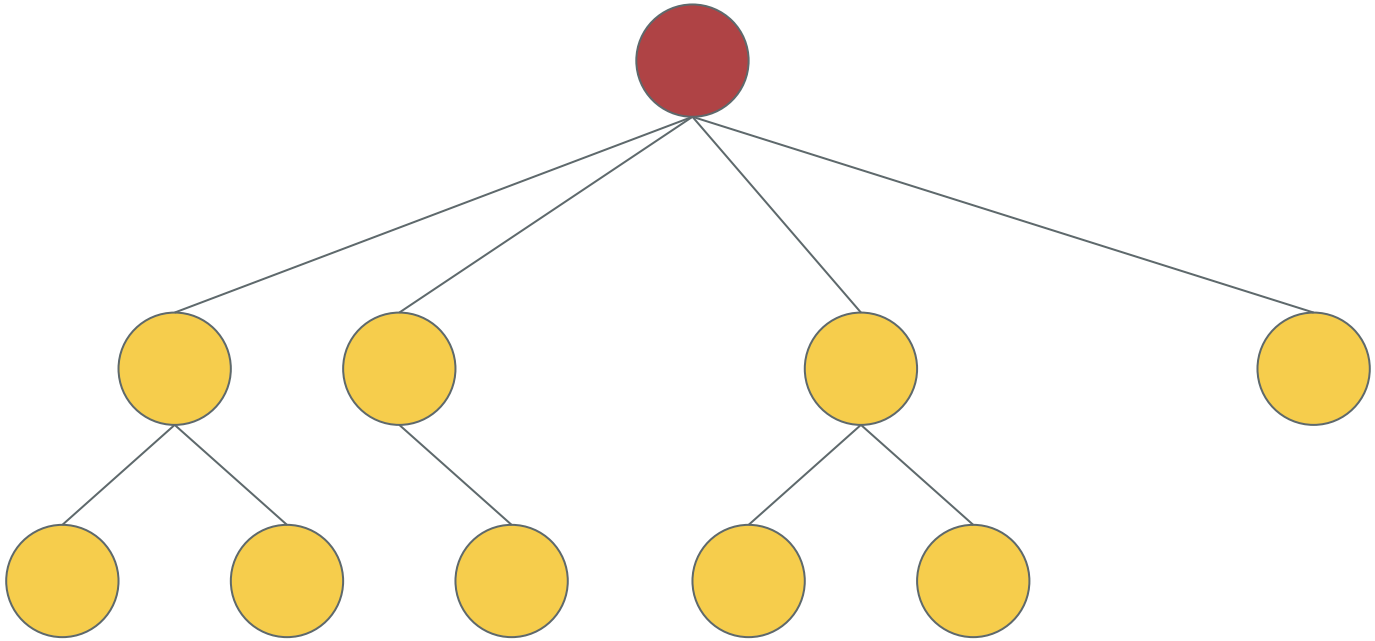
```
import * as React from 'react';

type Props = { text: string };

const Component = ({ text }: Props) => <button>{text}</button>;

export const Button = () => <Component text="Hello" />;
```

Recursive updates



@lithinn

Components

Examples - anonymous functions

```
const Component = ({ onClick }) => <button onClick={onClick}>Hello</button>;

export const Parent = () => <Component onClick={() => {}} />;
```

Examples - anonymous functions

```
const Component = ({ onClick }) => <button onClick={onClick}>Hello</button>;

const onClick = () => {};
export const Parent = () => <Component onClick={onClick} />;
```

Examples - react-redux

```
const mapStateToProps = state => ({ id: getProductId(state) });

const mapDispatchToProps = dispatch => {
    return { trackView: id => () => dispatch(track('product-screen', id)) };
};

const mergeProps = (stateProps, dispatchProps) => {
    return { track: dispatchProps.trackView(stateProps.id) };
};

export default connect(
    mapStateToProps,
    mapDispatchToProps,
    mergeProps
)(Component);
```

@lithinn

Examples - react-redux

```
const mapStateToProps = state => ({ id: getProductId(state) });

const mapDispatchToProps = { trackView: track };

export default connect(
    mapStateToProps,
    mapDispatchToProps
)(Component);
```

@lithinn

Examples - selectors

```
export const getProductId = state => {
    const product = state.product.items.find(item => item.type === 'sense');
    const child = product ? product.children.find(child => child.hasId) : {};
    return child.id || '';
};
```

Examples - selectors

```
import { createSelector } from 'reselect';

export const getProductId = createSelector(
    state => state.product.items.find(item => item.type === 'sense'),
    product => (product ? product.children.find(child => child.hasId) : {}),
    child => child.id || ''
);
```

State - best practices

Local state - setting state based on props

```
componentDidUpdate(prevProps, prevState) {
    if (this.props.selected) {
        this.setState({ updated: true });
    }
}
```

```
static getDerivedStateFromProps(props, state) {
    if (props.selected) {
        return { updated: true };
    }
}
```

@lithinn

Local state - setting state based on props

```
const mapStateToProps = state => {
    return { selected: state.selected, updated: !!state.selected };
};
```

Local state - necessary?

```
onClick = () => {
    this.setState({ updated: true });
};
```

```
onClick = () => {
    if (!this.state.updated) {
        this.setState({ updated: true });
    }
};
```

@lithinn

Redux state - local

```
toggle = isOpen => {
    this.props.dispatch({ payload: !isOpen, type: 'OPEN_MODAL' });
};
```

```
toggle = () => {
    this.setState({ openModal: !this.state.openModal });
};
```

@lithinn

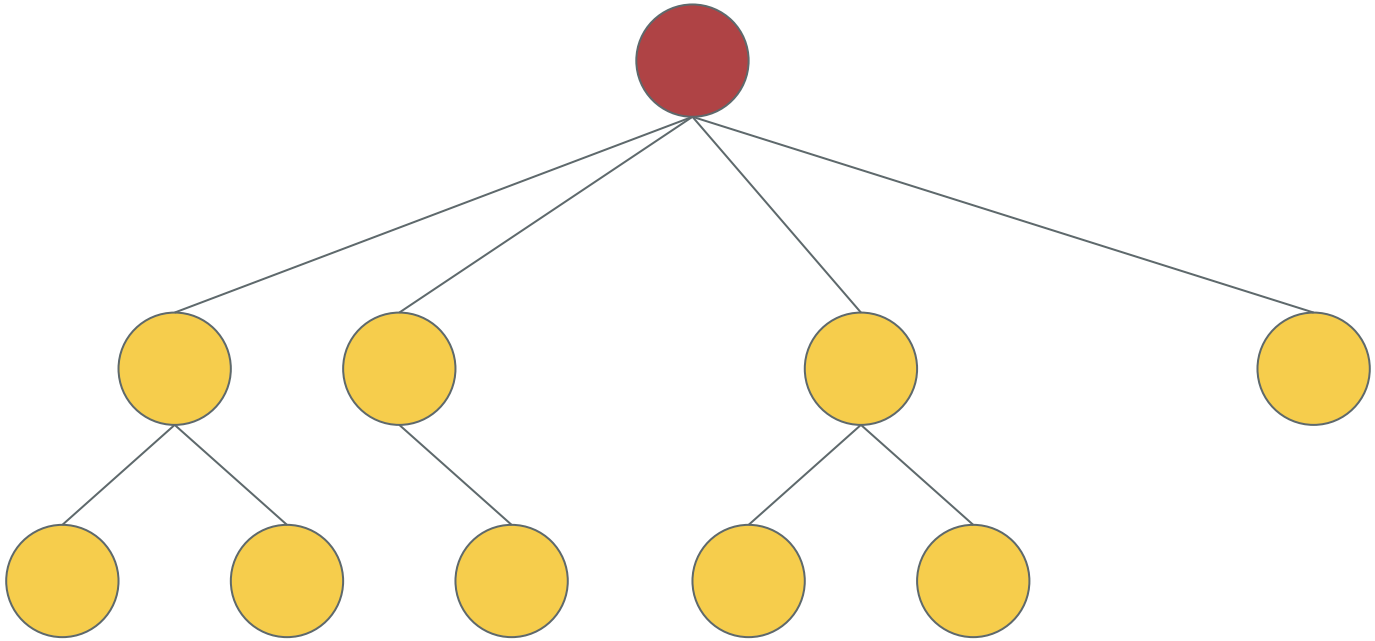
Redux state - batching

```
onClick = () => {
    this.props.dispatch({ type: 'GET_CONFIG' });
    this.props.dispatch({ type: 'FETCH_UPDATES' });
    this.props.dispatch({ type: 'PREPARE_FORM' });
};
```

```
onClick = () => {
    this.props.dispatch({ type: 'UPDATE' });
};
```

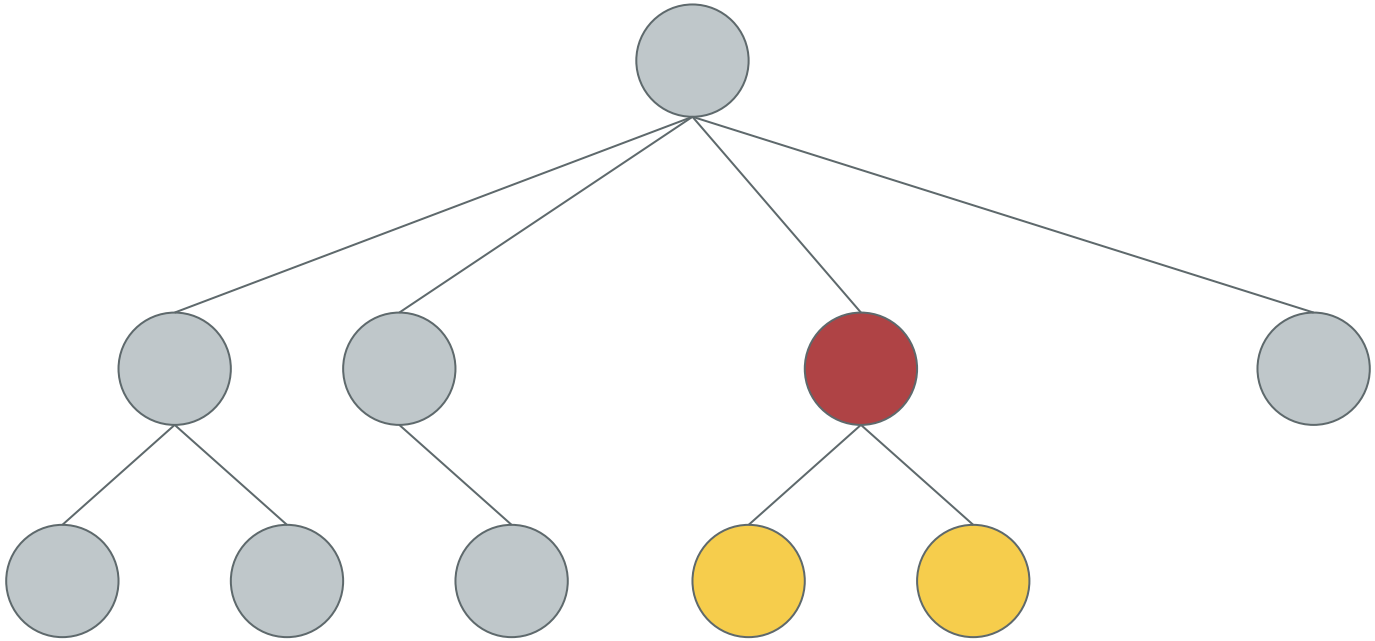
@lithinn

Connecting state



@lithinn

Connecting state



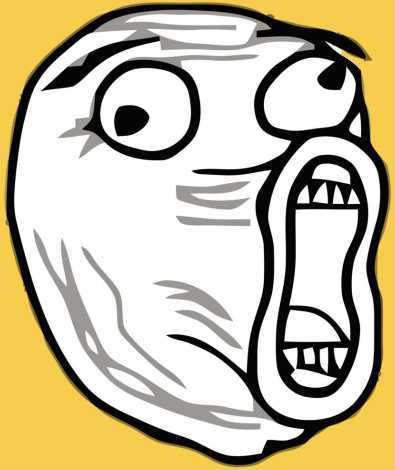
@lithinn

General conclusions

For react web/RN

- It’s always the JS
- Measure before acting
- Think about reconciliation & state updates
- Memoise

...or just use MobX



@lithinn

Native only

Native only - animations

```
Animated.timing(bounceValue, {
    toValue: 1,
    duration: 300,
    easing: Easing.linear,
    useNativeDriver: true,
}).start();
```

Native only - inline requires

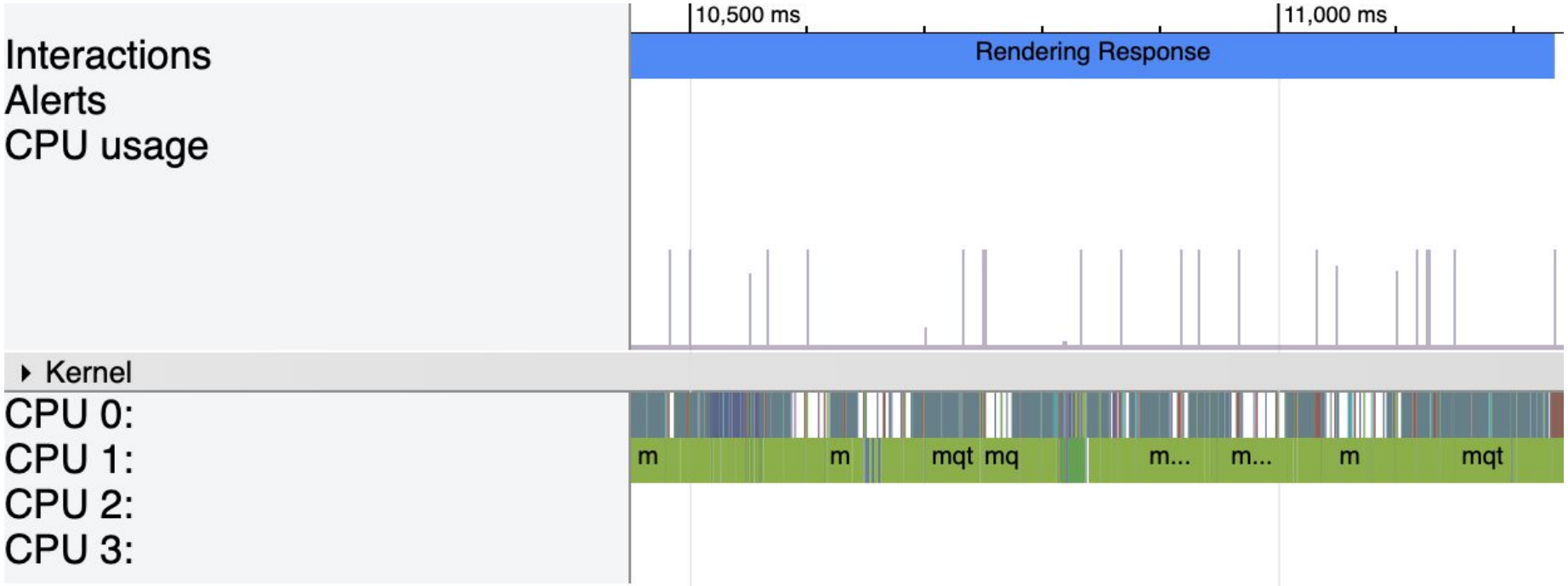
```
setTimeout(() => {
    const codePush = require('react-native-code-push');
    codePush.restartApp(false);
}, 1000);
```

```
<HeroCover coverImage={require('./upgrade-hero.jpg')} />
```

RAM bundler & Lazy Package

Native only - low-level measuring

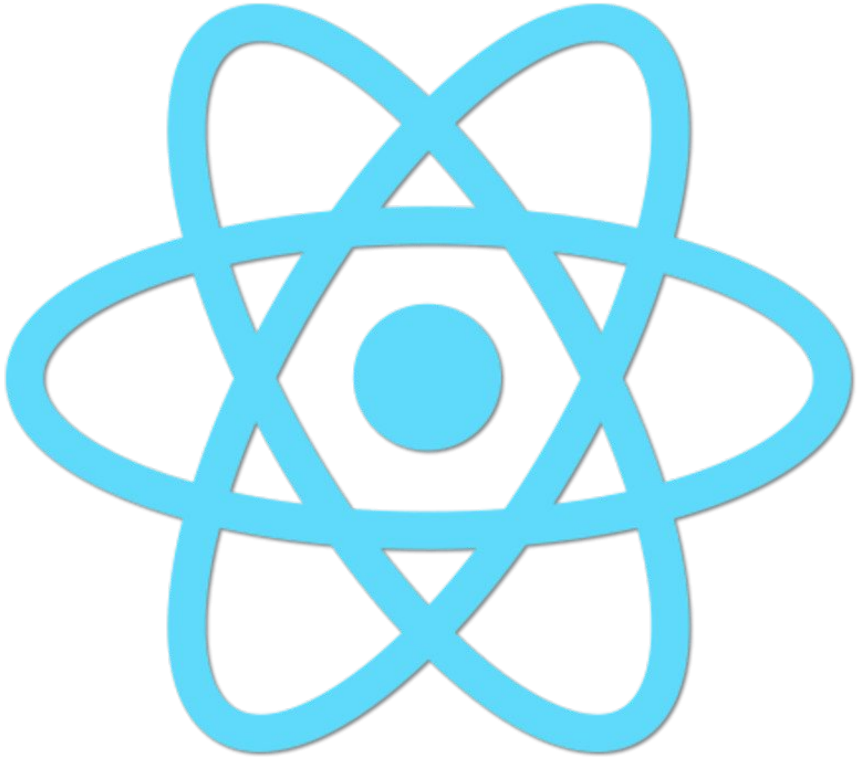
- Use profilers & systrace to check how your app is doing on low level



@lithinn

Native only - routing

- React router native
- React navigation
- React native navigation



Optimise wisely

Thank you!

Anna Doubkova @ Hive