

React in 2020

- URL: <https://pavestru.github.io/react-bbs-2020-slides>
- Source:
<https://github.com/pavestru/react-bbs-2020-slides>
- This presentation was generated using [mdx-deck](#).

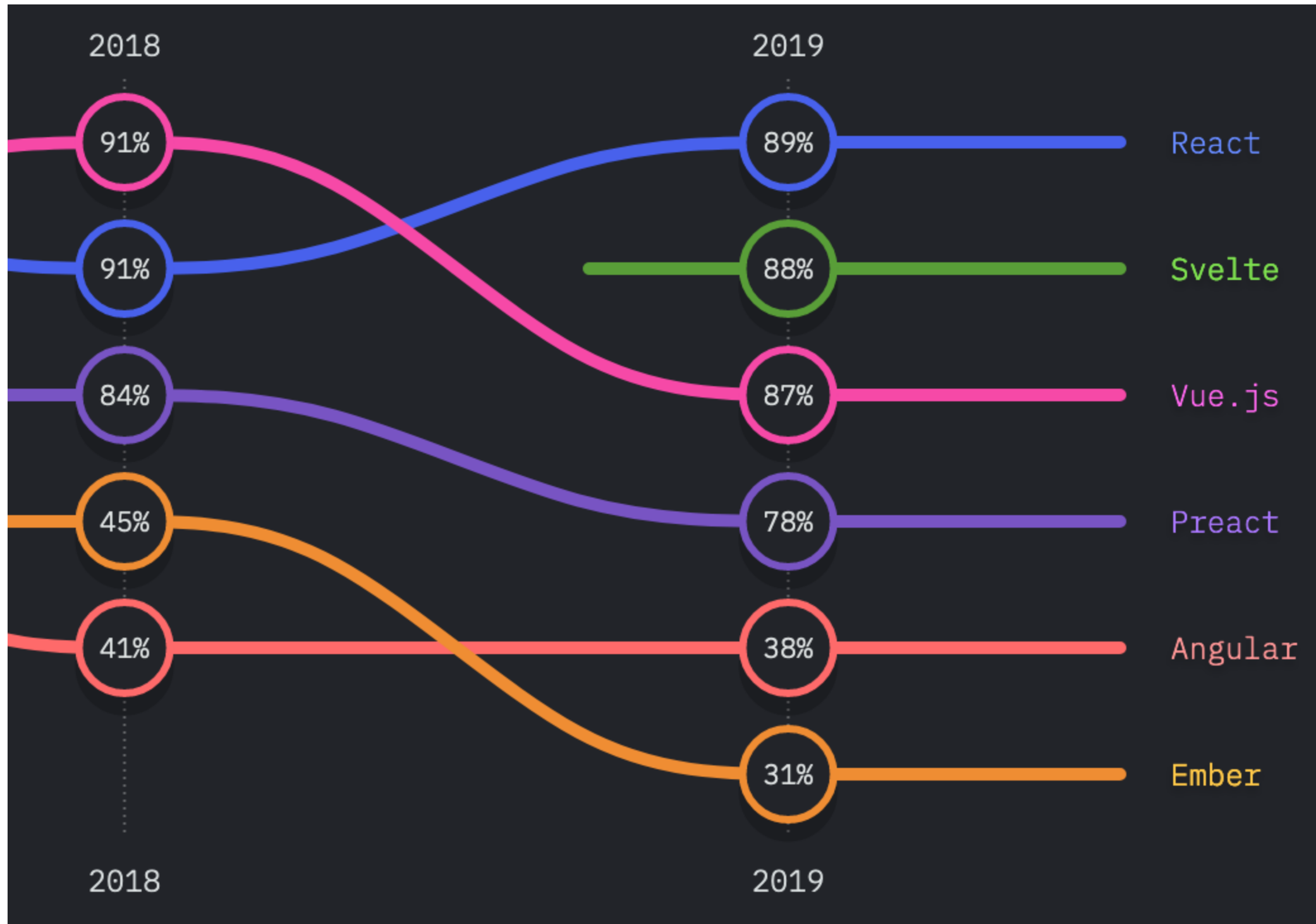
Speakers

- Pavel Struhar
- Jan Capiak

State of JS 2019

<https://2019.stateofjs.com/>

State of JS - Satisfaction survey



Svelte? Who is Svelte?

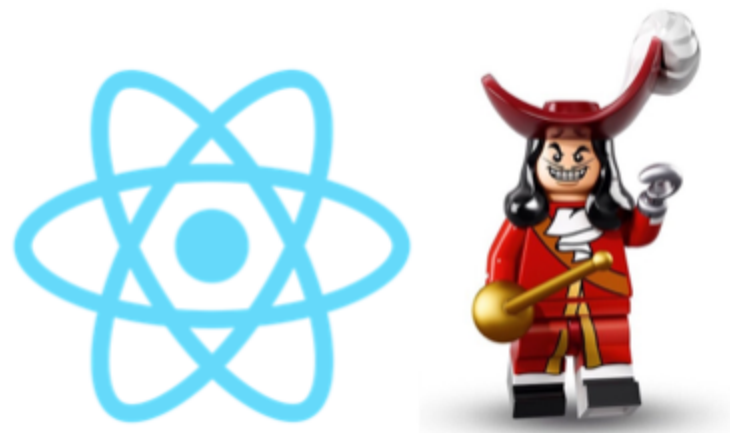
Are those graphs relevant?

Why React is winning

- Uni-directional data flow
- Component model
- Just JavaScript
- Just UI component library
- Community
- Investments of large companies
- ...
- Innovative

Topics

- Hooks
- Component Life-cycle API updates
- Error Boundaries
- Concurrent rendering and Suspense



REACT HOOKS

What are React Hooks?

- JS functions
- \geq v16.8.0
- Hooks don't work inside classes

Which Hooks does React offer?

- Basic Hooks (**useState, useEffect, useContext**)
- Additional Hooks (**useCallback, useMemo, useRef ...**)
- Custom Hooks

Function component

```
import React from 'react';  
  
export function Example(props) {  
  return <div>Hello {props.name}!</div>;  
}
```

```
import React, { useState } from 'react';

export function Example(props) {
  const [count, setCount] = useState(0);
  return (
    <div>
      Hello {props.name}!
      <button onClick={() => setCount(count + 1)}>
        Click me (clicked {count} times)
      </button>
    </div>
  );
}
```

React Class vs. useState Hook

```
class Example extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0
    };
  }

  render() {
    return (
      <div>
        <p>You clicked {this.state.count} times</p>
        <button onClick={() => this.setState({ count: this.state.count + 1 })}>
          Click me
        </button>
      </div>
    );
  }
}
```

```
import React, { useState } from 'react';

function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);

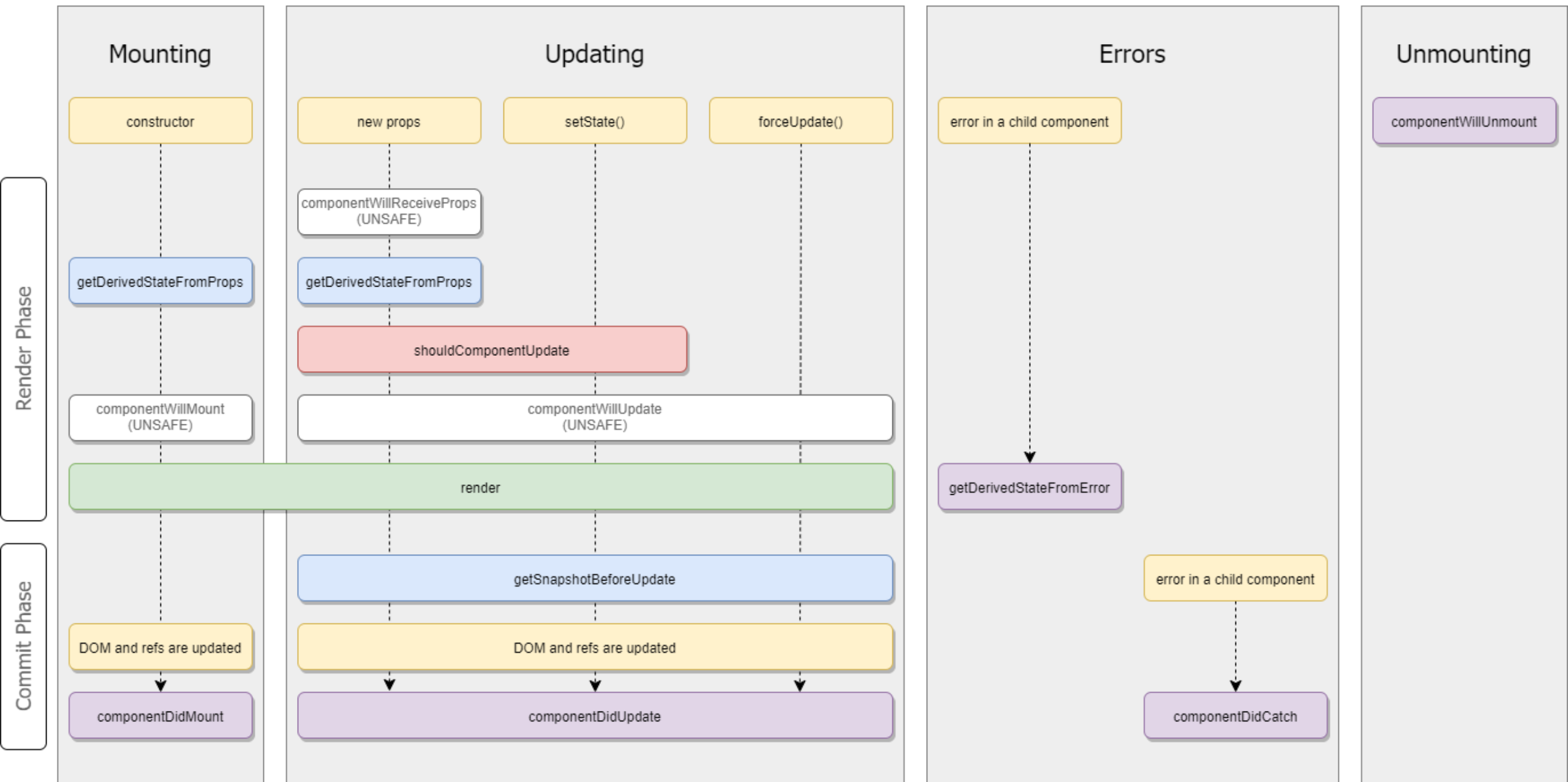
  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

React Life-cycle methods - Legacy

- `componentWillMount` → **`UNSAFE_componentWillMount`**
- `componentWillReceiveProps` →
`UNSAFE_componentWillReceiveProps`
- `componentWillUpdate` → **`UNSAFE_componentWillUpdate`**

React Life-cycle methods - New

- **getDerivedStateFromProps** $\geq 16.3.0$
- **getSnapshotBeforeUpdate** $\geq 16.3.0$
- **getDerivedStateFromError** $\geq 16.6.0$



Rewriting to Hooks

- Live coding session
- Radio Stations deployed app: <https://pavestru.github.io/radio-stations>
- Source: <https://github.com/pavestru/radio-stations>
- See the [diff](#) between old and Hooks
- See the [diff](#) for custom hook



Mark Dalgleish
@markdalgleish



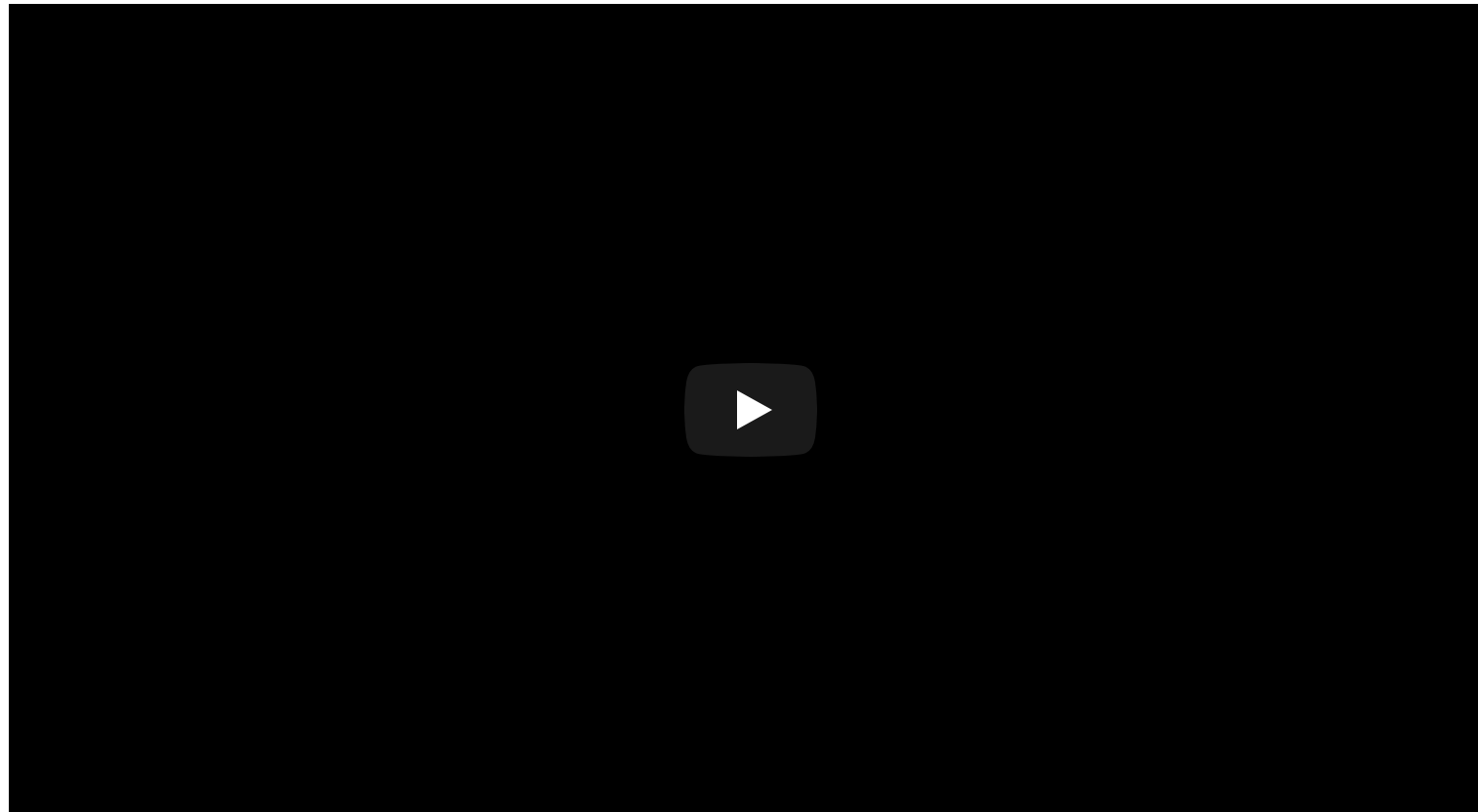
"With hooks, beginners no longer need to learn about 'this' to avoid shooting themselves in the foot."

Closures:



[Tweet source](#)

JavaScript Closure



Getting Closure on React Hooks

<https://www.youtube.com/watch?v=KJP1E-Y-xyo>

Recommended articles

- [Writing Resilient Components](#) by Dan Abramov
- [Making setInterval Declarative with React Hooks](#) by Dan Abramov

Error Boundary

- components that catch JavaScript errors and display a fallback UI
- `getDerivedStateFromError()`, `componentDidCatch()`
- works like a JavaScript catch block, but for components
- catch only errors in the components below them in the tree

```
class ErrorBoundary extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = { hasError: false };  
  }  
  
  static getDerivedStateFromError(error) {  
    // Update state so the next render will show the fallback UI.  
    return { hasError: true };  
  }  
  
  componentDidCatch(error, errorInfo) {  
    // You can also log the error to an error reporting service  
    logErrorToMyService(error, errorInfo);  
  }  
  
  render() {  
    if (this.state.hasError) {  
      // You can render any custom fallback UI  
      return <h1>Something went wrong.</h1>;  
    }  
  
    return this.props.children;  
  }  
}
```

```
<ErrorBoundary>  
  <MyWidget />  
</ErrorBoundary>
```


Error Boundary

"try-catch" for `throw new Error()`

React Suspense

"try-catch" for `throw new Promise()`

React Suspense

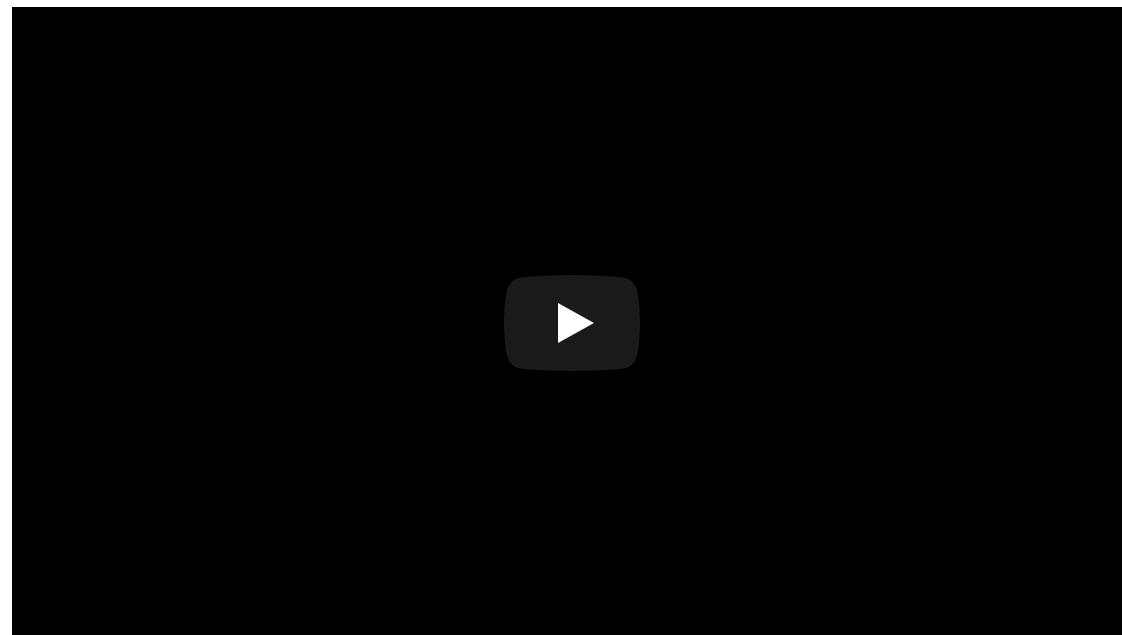
API still experimental

```
const ProfilePage = React.lazy(() => import("./ProfilePage"))  
  
const fallback = {<Spinner />}</>  
const ProfilePage = lazy(fallback)  
return <div>  
  <ProfilePage />  
</div>  
</Suspense>;
```

Concurrent rendering

Putting User Experience First

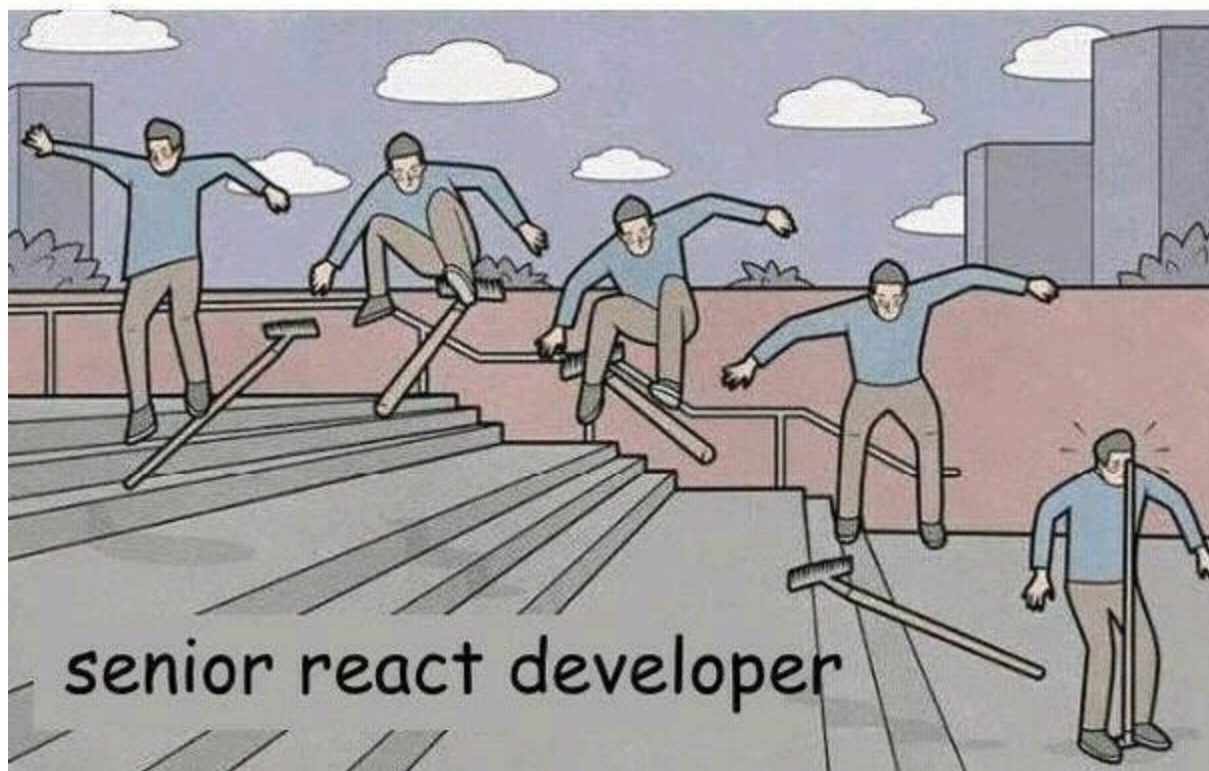
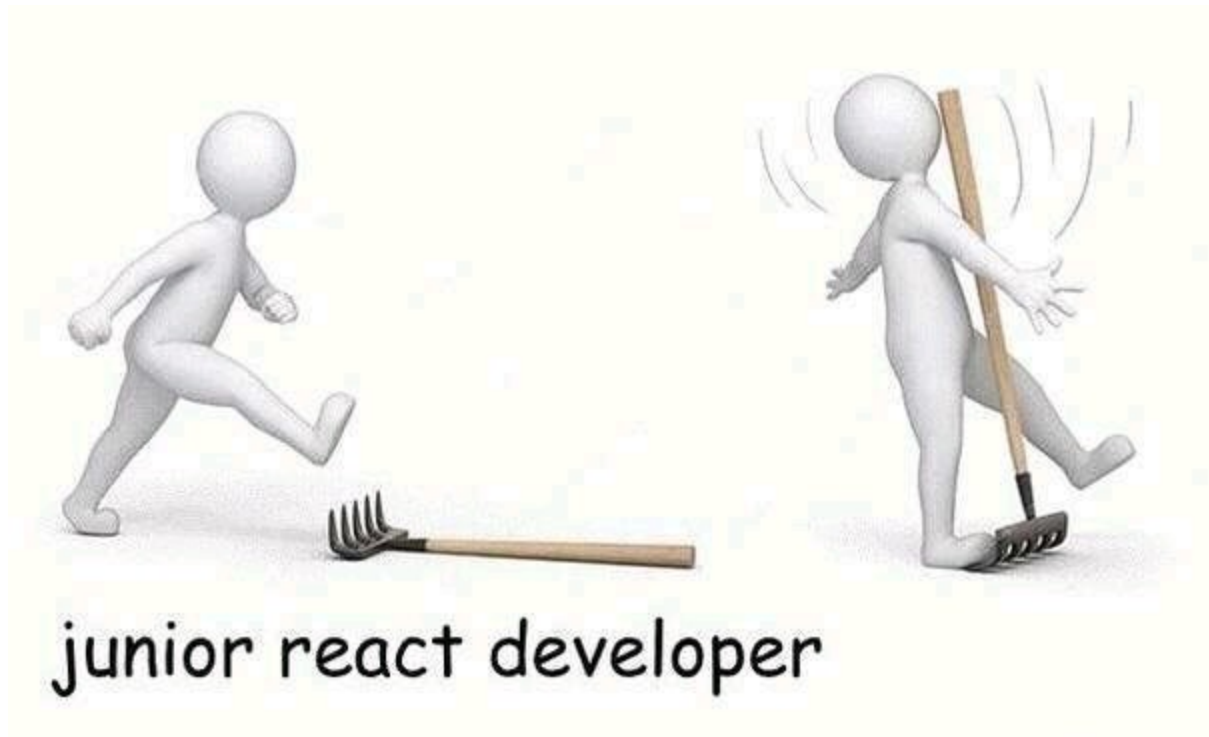
Understanding Event Loop



Event Loop playground: <http://latentflip.com/loupe>

Concurrent rendering

- Splits rendering into chunks
- Puts them into batch
- Able to interrupt, cancel rendering



Future talks?

Deep dive



SVELTE



Thank you for your attention

