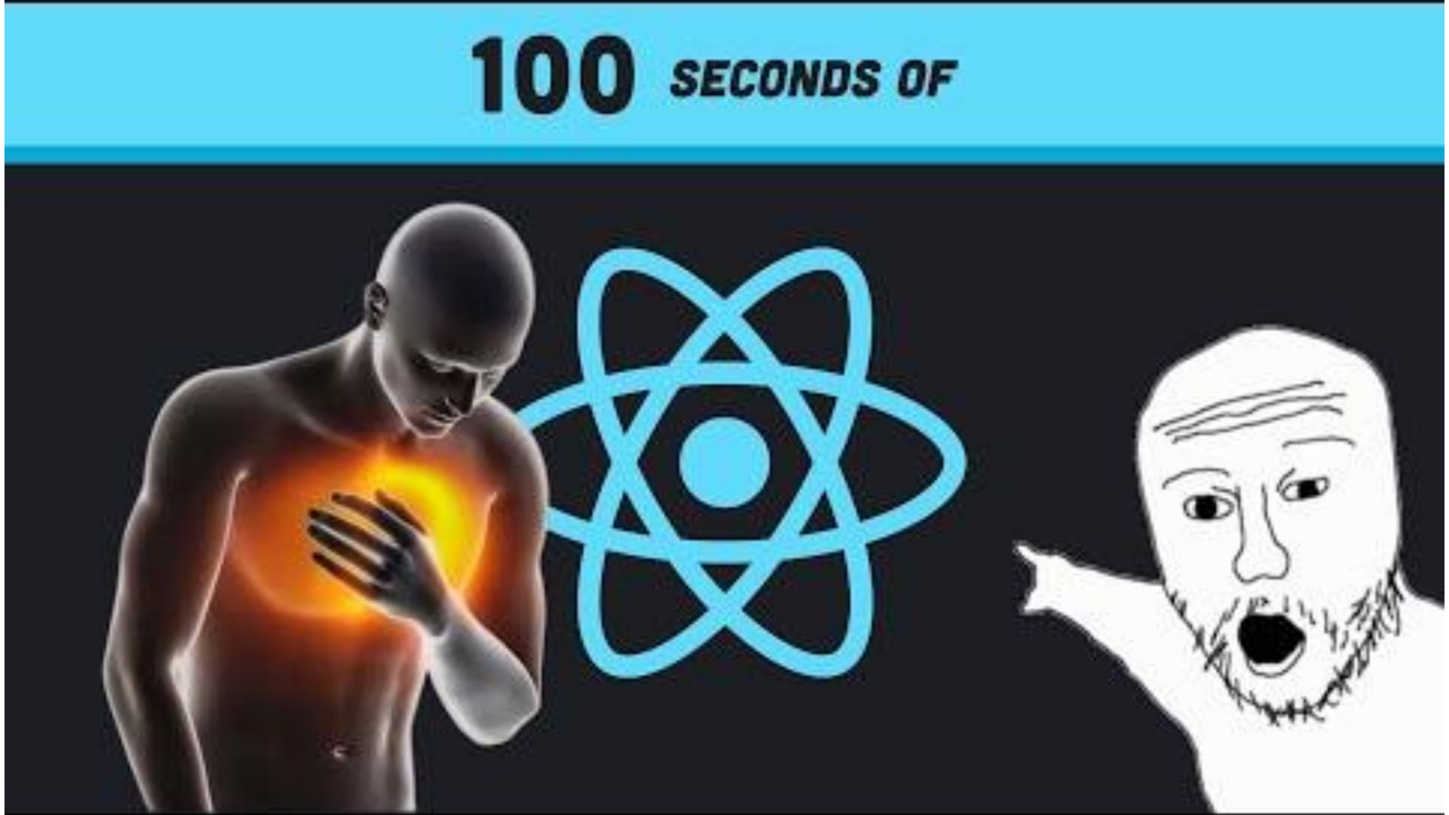
Lecture 7
React

Frontend Web Development


100 SECONDS OF

React is

- A UI library (not a framework) developed by Facebook in 2013
- Declarative
- Component-based
- Very popular (the #1 framework professionally)
- Free and open-source
- Currently at version 18.2 (released in June 2022)

JSX

JSX

```
class Button extends React.Component {
   state = { color: 'red' }

   handleChange = () => {
       const color = this.state.color === 'red' ? 'blue' : 'red';
       this.setState({color});
   }

   render() {
       return (<div>
           <button
               className={`btn ${this.state.color}`}
               onClick={this.handleChange}>
           </button>
       </div>);
   }
}
```

JSX

```
class Button extends React.Component {
   state = { color: 'red' }

   handleChange = () => {
       const color = this.state.color === 'red' ? 'blue' : 'red';
       this.setState({color});
   }

   render() {
       return (<div>
           <button
               className={`btn ${this.state.color}`}
               onClick={this.handleChange}>
           </button>
       </div>);
   }
}
```

Huh?

JSX
HTML
JS
What the hell is this?

JSX

JSX (**JavaScript Syntax Extension** and occasionally referred as **JavaScript XML**) is a React extension to the JavaScript language syntax which provides a way to structure component rendering using syntax familiar to many developers.

It is similar in appearance to HTML.

— Wikipedia

JSX

```
const element1 = <h1>Hello, world!</h1>;

const element2 = (<Component prop={value}>
    <h1>Hello, world!</h1>
</Component>);

const name = `My Name`;
const element3 = <h1>Hello, {name}!</h1>;
```

Introducing JSX

JSX

```
const element1 = React.createElement('h1', null, 'Hello, world!');

const element2 = React.createElement(
   Component,
   { prop: value },
   React.createElement(
       'h1',
       null,
       'Hello, world!'
   )
);

const name = `My Name`;
const element3 = React.createElement('h1', null, 'Hello, ', name, '!');
```

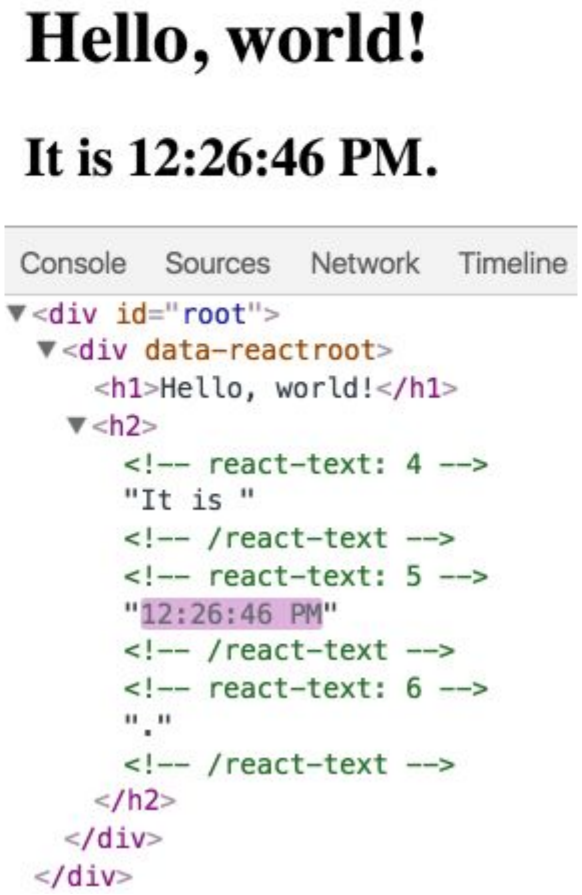
React
Top-Level API

Rendering

```
import ReactDOM from 'react-dom';

const root = ReactDOM.createRoot(
   document.getElementById('root')
);

const element = <h1>Hello, world</h1>;

root.render(element);
```

Rendering

```
const rootNode =
    document.getElementById('root');
const root =
    ReactDOM.createRoot(rootNode);

function tick() {
   const element = (<div>
           <h1>Hello, world!</h1>
           <h2>
    It is {new Date().toLocaleTimeString()}.
           </h2>
       </div>);
   root.render(element);
}

setInterval(tick, 1000);
```



Hello, world!

It is 12:26:46 PM.

```
▼<div id="root">
  ▼<div data-reactroot>
      <h1>Hello, world!</h1>
    ▼<h2>
        <!-- react-text: 4 -->
        "It is "
        <!-- /react-text -->
        <!-- react-text: 5 -->
        "12:26:46 PM"
        <!-- /react-text -->
        <!-- react-text: 6 -->
        "."
        <!-- /react-text -->
      </h2>
    </div>
  </div>
```

Virtual DOM

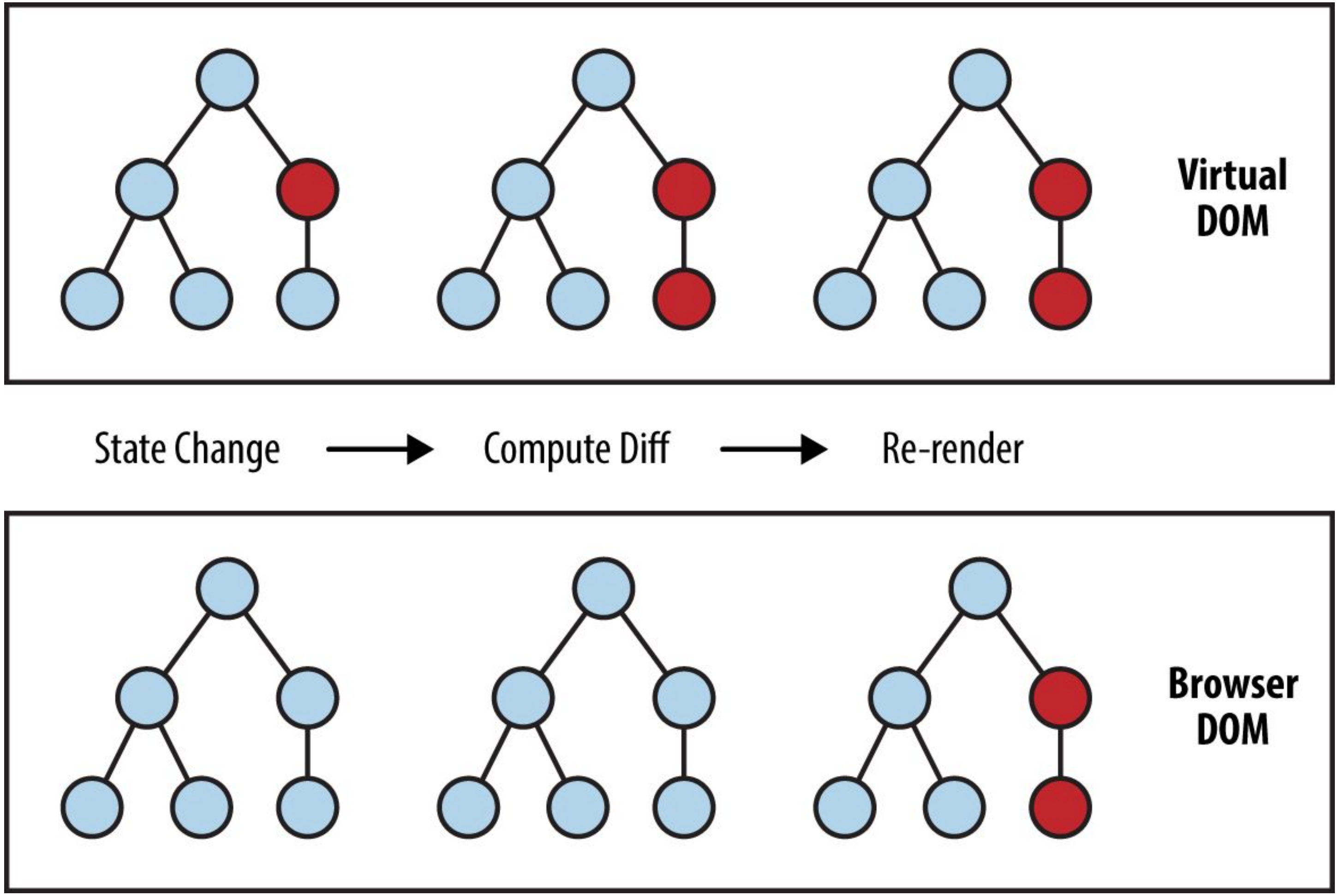
Perhaps it’s better to think of the virtual DOM as *React’s local and simplified copy of the HTML DOM*.

It allows React to do its computations within this abstract world and skip the “real” DOM operations, often slow and browser-specific.

The difference between Virtual DOM and DOM

How to write your own Virtual DOM

The Inner Workings Of Virtual DOM



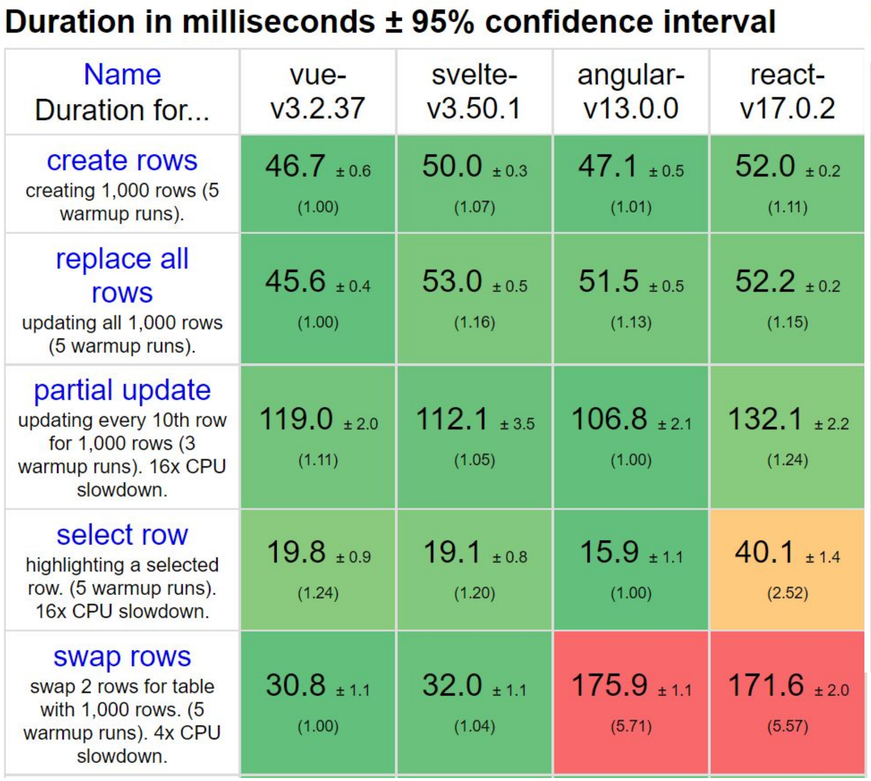
https://programmingwithmosh.com/react/react-virtual-dom-explained/

Virtual DOM

```
<ul class="list">
   <li>item 1</li>
   <li>item 2</li>
</ul>
```

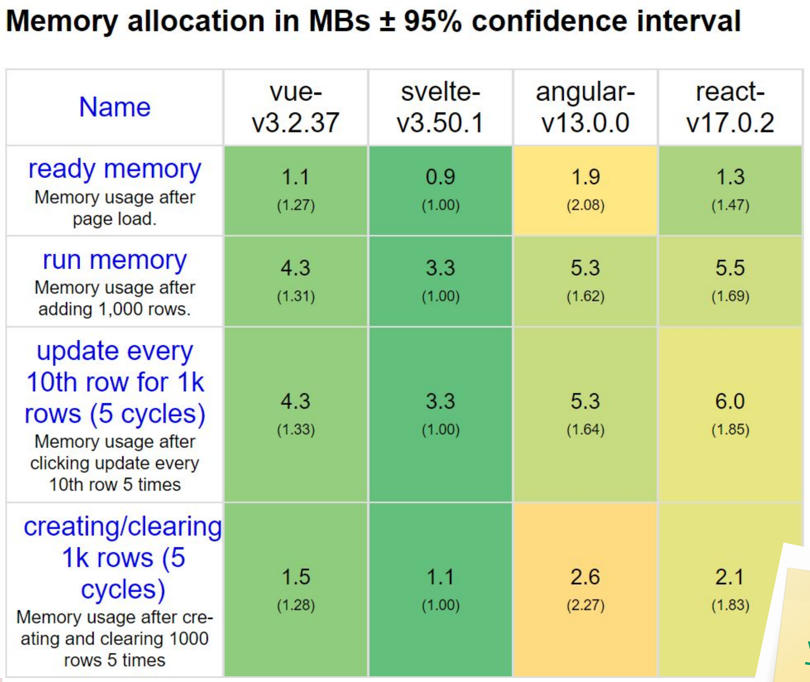
```
{ type: 'ul', props: { 'class': 'list' }, children:
   [
       { type: 'li', props: {}, children: ['item 1'] },
       { type: 'li', props: {}, children: ['item 2'] }
   ]
}
```

Does it really improve the performance?



Duration in milliseconds ± 95% confidence interval

Name Duration for...	vue-v3.2.37	svelte-v3.50.1	angular-v13.0.0	react-v17.0.2
create rows creating 1,000 rows (5 warmup runs).	46.7 ± 0.6 (1.00)	50.0 ± 0.3 (1.07)	47.1 ± 0.5 (1.01)	52.0 ± 0.2 (1.11)
replace all rows updating all 1,000 rows (5 warmup runs).	45.6 ± 0.4 (1.00)	53.0 ± 0.5 (1.16)	51.5 ± 0.5 (1.13)	52.2 ± 0.2 (1.15)
partial update updating every 10th row for 1,000 rows (3 warmup runs). 16x CPU slowdown.	119.0 ± 2.0 (1.11)	112.1 ± 3.5 (1.05)	106.8 ± 2.1 (1.00)	132.1 ± 2.2 (1.24)
select row highlighting a selected row. (5 warmup runs). 16x CPU slowdown.	19.8 ± 0.9 (1.24)	19.1 ± 0.8 (1.20)	15.9 ± 1.1 (1.00)	40.1 ± 1.4 (2.52)
swap rows swap 2 rows for table with 1,000 rows. (5 warmup runs). 4x CPU slowdown.	30.8 ± 1.1 (1.00)	32.0 ± 1.1 (1.04)	175.9 ± 1.1 (5.71)	171.6 ± 2.0 (5.57)



Memory allocation in MBs ± 95% confidence interval

Name	vue-v3.2.37	svelte-v3.50.1	angular-v13.0.0	react-v17.0.2
ready memory Memory usage after page load.	1.1 (1.27)	0.9 (1.00)	1.9 (2.08)	1.3 (1.47)
run memory Memory usage after adding 1,000 rows.	4.3 (1.31)	3.3 (1.00)	5.3 (1.62)	5.5 (1.69)
update every 10th row for 1k rows (5 cycles) Memory usage after clicking update every 10th row 5 times	4.3 (1.33)	3.3 (1.00)	5.3 (1.64)	6.0 (1.85)
creating/clearing 1k rows (5 cycles) Memory usage after cre-ating and clearing 1000 rows 5 times	1.5 (1.28)	1.1 (1.00)	2.6 (2.27)	2.1 (1.83)

VDOM is pure overhead (Svelte blog)

https://krausest.github.io/js-framework-benchmark/2022/table_chrome_106.0.5249.61.html

Anatomy of a Component
Class Components

Class Component

```
class Button extends React.Component {
   state = { color: 'red' }

   handleChange = () => {
       const color = this.state.color === 'red' ? 'blue' : 'red';
       this.setState({ color });
   }

   render() {
       return (<div>
           <button
               className={`btn ${this.state.color}`}
               onClick={this.handleChange}>
           </button>
       </div>);
   }
}
```

Render Props

```
class Button extends React.Component {
    // ...

    render() {
        return (<div>
            <button
                className={`btn ${this.state.color}`}
                onClick={this.handleChange}>
                {this.props.text}
            </button>
        </div>);
    }
}
// ...
<Button text="Hello, world!" />
```

Render Props (slots)

```
class Button extends React.Component {
    // ...
    render() {
        return (<div>
            <button
                className={`btn ${this.state.color}`}
                onClick={this.handleChange}>
                {this.props.children}
            </button>
        </div>);
    }
}
// ...
<Button>Hello, world!</Button>
```

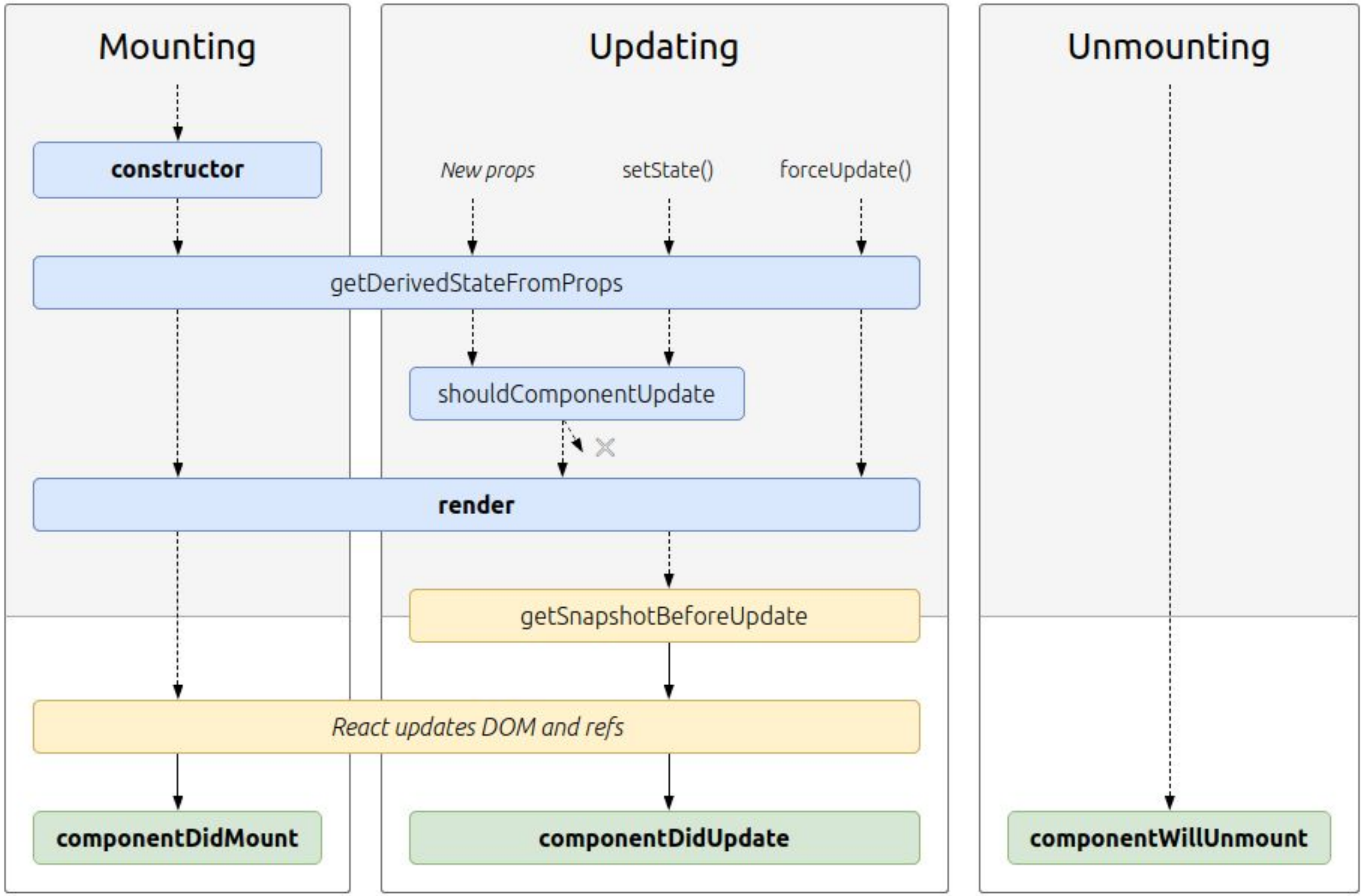
State

```
class Button extends React.Component {
    state = { color: 'red' }

    handleChange = () => {
        const color = this.state.color === 'red' ? 'blue' : 'red';
        this.setState({ color });
    }

    render() {
        return (<div>
            <button
                className={`btn ${this.state.color}`}
                onClick={this.handleChange}>
            </button>
        </div>);
    }
}
```

State

State and Lifecycle

```
// Usual update
this.setState(value);

// Update based on the previous state
this.setState(oldValue => ({ ...oldValue, counter: counter + 1}));

// Update based on the previous state and props
this.setState((oldValue, props) =>
                     ({ ...oldValue, counter: counter + props.increment }));
```



“Render phase”
Pure and has no side effects. May be paused, aborted or restarted by React.

“Pre-commit phase”
Can read the DOM.

“Commit phase”
Can work with DOM, run side effects, schedule updates.

Lifecycle

```
const socket = io(URL);

class Dashboard extends React.Component {
    ...
    componentDidMount() {
        socket.connect();
        socket.on('new_data', data => this.setState(data));
    }

    componentWillUnmount() {
        socket.disconnect();
    }
    ...
}
```

Lifecycle

```
class Dashboard extends React.Component {
    ...

    shouldComponentUpdate(nextProps, nextState) {
        return this.state.color !== nextState.color;
    }

    ...
}
```

Functional Components
Hooks

Functional Component

```
function Comment(props) {
   return (<div className="comment">
              <div className="user-info">
                  <img className="avatar"
                      src={props.author.avatarUrl}
                      alt={props.author.name}
                  />
                  <div className="user-info__name">
                     {props.author.name}
                  </div>
              </div>
              <div className="comment-text">
                  {props.text}
              </div>
          </div>);
}
```

Render Props

```
const Button = (props) => {
     return (<div>
       <button>
           {props.children}
       </button>
   </div>);
}

...

<Button>Hello, world!</Button>
```

Hooks

- Introduced in v16.8 (February 2019)
- Allow for state (& other React features) without a class
- Became the recommended way of writing components
- Allow you to reuse stateful logic without changing your component hierarchy
- Don’t work inside classes
- They are simple JS functions, but must be called only inside React functional components and at the top-level

useState()

```
const Button = () => {
   const [styles, setStyles] = useState({color: 'red'});

   const handleChange = () => {
       const color = styles.color === 'red' ? 'blue' : 'red';
       setStyles({ ...styles, color });
   }

   return (<div>
       <button
           className={`btn ${styles.color}`}
           onClick={handleChange}>
       </button>
   </div>);
}
```

useState()

```
// Usual creation with default value
const [styles, setStyles] = useState({ color: 'red' });

// Creation based on the function
// The function is called only once on mount
const [styles, setStyles] = useState(() => {
   // Expensive computations...
   return { color: 'red' };
});

// Usual update of data
setStyles({ color: 'blue' });

// Update based on previous state
setStyles(prevState => ({ ...prevState, color: 'blue' }));
```

useEffect()

```
const socket = io(URL);

const Dashboard = () => {
   // ...

   useEffect(() => {
       socket.connect();
       socket.on('new_data', data => setData(data));

       // Will be called when the effect is replaced
       return () => socket.disconnect();
   });

   ...
}
```

useEffect()

```
// Will be called during each rerender
useEffect(() => {
   socket.connect();
});

// Will be called only if elements of array are changed
useEffect(() => {
   socket.connect();
}, [socket]);

// Will be called only after first render
useEffect(() => {
   socket.connect();
}, []);
```

Other hooks

Basic Hooks

- useState
- useEffect
- useContext
- useId

Additional Hooks

- useReducer
- useCallback
- useMemo
- useRef
- useImperativeHandle
- useLayoutEffect
- useDebugValue
- useDeferredValue
- useTransition

Hooks API Reference

React Ecosystem

Styles

Usual Styles

```
// Somewhere in index.html...
<link rel="stylesheet" href="style.css" />

// style.css
.example {
    display: flex;
    ...
}

// Somewhere in React app
const component = () => (
   <div className="example">
       ...
   </div>
);
```

Problem with Usual Styles

```
// Ok, what if we want to split styles into separate units…

<link rel="stylesheet" href="style.css" />
<link rel="stylesheet" href="button/style.css" />
<link rel="stylesheet" href="header/user/style.css" />
<link rel="stylesheet" href="some/other/important/style.css" />
...
```

Imported Styles

```
// Somewhere in a React component
import 'Button.css';

const Button = (props) => (
   <div className="btn-wrapper">
       ...
   </div>
);
```

```
// Somewhere in another component
import 'Icon.css';

const Icon = (props) => (
   <div className="icon-wrapper">
       ...
   </div>
);
```

```
// Works with create-react-app out of the box
// In your own setup you need to configure something
// like style-loader for webpack
```

Problem with Imported Styles

```
// Somewhere in a React component
import 'Button.css';

const Button = (props) => (
   <div className="wrapper">
       ...
   </div>
);
```

```
// Somewhere in another component
import 'Icon.css';

const Icon = (props) => (
   <div className="wrapper">
       ...
   </div>
);
```

```
// Works with create-react-app out of the box
// In your own setup you need to configure something
// like style-loader for webpack
```

Problem with Imported Styles

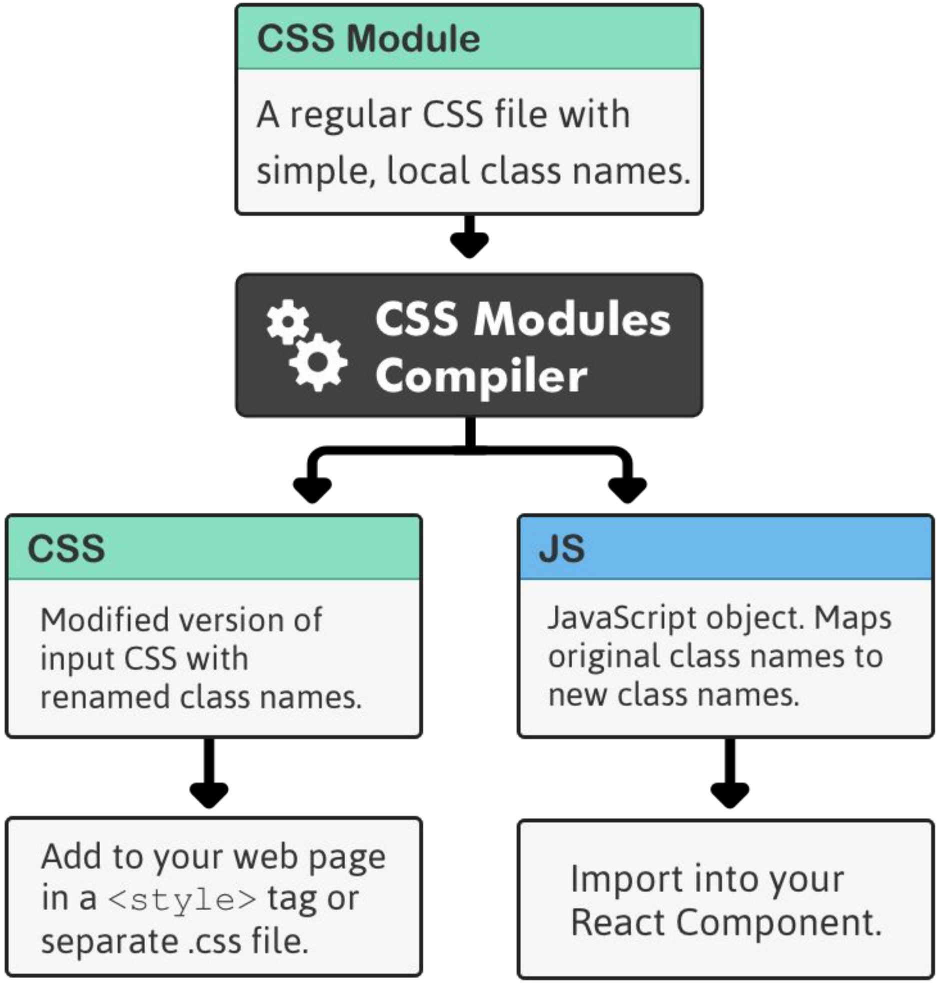
```css
/* Somewhere in final css-file... */
/* Button.css */
.wrapper {
   ...
   background-color: red;
}

/* Icon.css */
.wrapper {
   ...
   background-color: black;
}
```

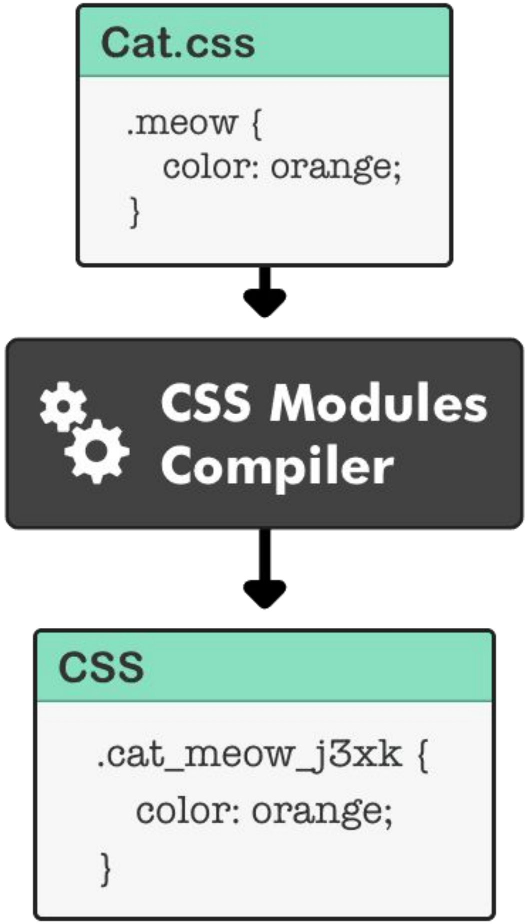
CSS Modules

```
// Somewhere in a React component
import classes from 'Button.module.css';

const Button = (props) => (
   <div className={classes.wrapper}>
       ...
   </div>
);

// Works with create-react-app out of the box
// In your own setup you need to configure something
// like css-loader for webpack
```

Using CSS Modules In React App

How to configure CSS and CSS modules in webpack


CSS Module
A regular CSS file with simple, local class names.
CSS Modules Compiler
CSS
Modified version of input CSS with renamed class names.
JS
JavaScript object. Maps original class names to new class names.
Add to your web page in a <style> tag or separate .css file.
Import into your React Component.



```
.meow {
  color: orange;
}
```

```
.cat_meow_j3xk {
  color: orange;
}
```

CSS-in-JS

```
const Button = styled.button`
 background: transparent;
 border-radius: 3px;
 border: 2px solid palevioletred;
 color: palevioletred;
 padding: 0.25em 1em;

 ${props => props.primary && css`
   background: palevioletred;
   color: white;
 `}
`;

const Container = styled.div`
 text-align: center;
`
```

```
const Example = () => (
    <Container>
        <Button>
            Normal Button
        </Button>
        <Button primary>
            Primary Button
        </Button>
    </Container>
);
```

Styled components

Routing

React Router

React Router

```
const App = () => (
   <BrowserRouter>
       <Routes>
           <Route path="/"        element={<Home />}      />
           <Route path="expenses" element={<Expenses />} />
           <Route path="invoices" element={<Invoices />} />
       </Routes>
   </BrowserRouter>
);

// Somewhere in another component
<Link to="/expenses">Expenses</Link>
```

Form Handling

The React Way

```
export const NewItem = ({ onCreate }) => {
   const [text, setText] = useState('');

   return (
       <div>
           <input
               type="text"
               value={text}
               onInput={(event) => setText(event.target.value)}
           />
           <button onClick={() => onCreate(text)}>Create</button>
       </div>
   )
};
```

Main Problem

```
const [text, setText] = useState('');
const [name, setName] = useState('');
const [surname, setSurname] = useState('');
const [gender, setGender] = useState('');
const [birthDate, setBirthDate] = useState('');
const [email, setEmail] = useState('');
// or
const [data, setData] = useState({
   text: '',
   name: '',
   surname: '',
   gender: '',
   birthDate: '',
   email: '',
   ...
});
```

react-hook-form

```
const App = () => {
   const { register, handleSubmit } = useForm();
   const onSubmit = data => console.log(data);

   return (
     <form onSubmit={handleSubmit(onSubmit)}>
       <input {...register("firstName", { required: true, maxLength: 20 })} />
       <input {...register("lastName", { pattern: /^[A-Za-z]+$/i })} />
       <input type="number" {...register("age", { min: 18, max: 99 })} />
       <input type="submit" />
     </form>
   );
};
```

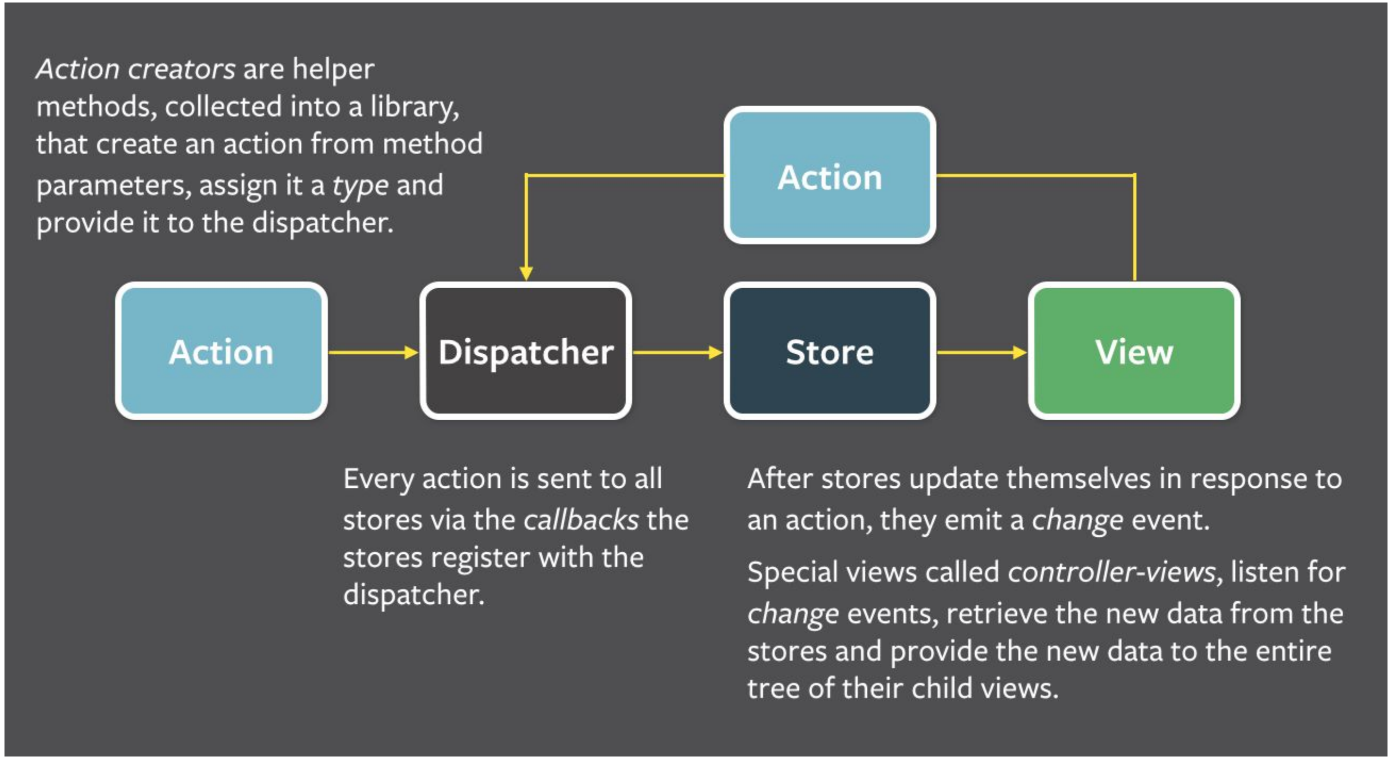
State Management

Custom Hooks

```
function useFriendStatus(friendID) {
   const [isOnline, setIsOnline] = useState(null);

   useEffect(() => {
       function handleStatusChange(status) {
           setIsOnline(status.isOnline);
       }

       ChatAPI.subscribeToFriendStatus(friendID, handleStatusChange);
       return () => {
           ChatAPI.unsubscribeFromFriendStatus(friendID, handleStatusChange);
       };
   });

   return isOnline;
}
```

Building Your Own Hooks

Flux Pattern



Action creators are helper methods, collected into a library, that create an action from method parameters, assign it a *type* and provide it to the dispatcher.

Every action is sent to all stores via the *callbacks* the stores register with the dispatcher.

After stores update themselves in response to an action, they emit a *change* event.

Special views called *controller-views*, listen for *change* events, retrieve the new data from the stores and provide the new data to the entire tree of their child views.

Scalable Frontend The State Layer

6 things I wish I knew about state management when I started writing React apps

Flux: In-Depth Overview

Hacker Way: Rethinking Web App Development at Facebook

Meta-frameworks

Next.js

- The “SvelteKit” of React
- Developed by Vercel in 2016
- All the features you’d expect from a production-ready framework
  - Hybrid SSR/SSG
  - File-system routing
  - TypeScript support
  - API routes
  - Internationalization
  - …

Remix

Couldn’t explain it better than their website: https://remix.run/

References and useful links

- https://github.com/illright/react-for-svelte-devs
- https://redux.js.org/
- https://nextjs.org/
- https://create-react-app.dev/