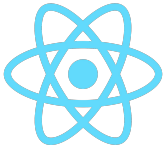
Soluto
by asurion

REACT NEW FEATURES

AND INTRO TO HOOKS

Nir Hadassi
Software Engineer @ Soluto



This talk was supposed to be about...

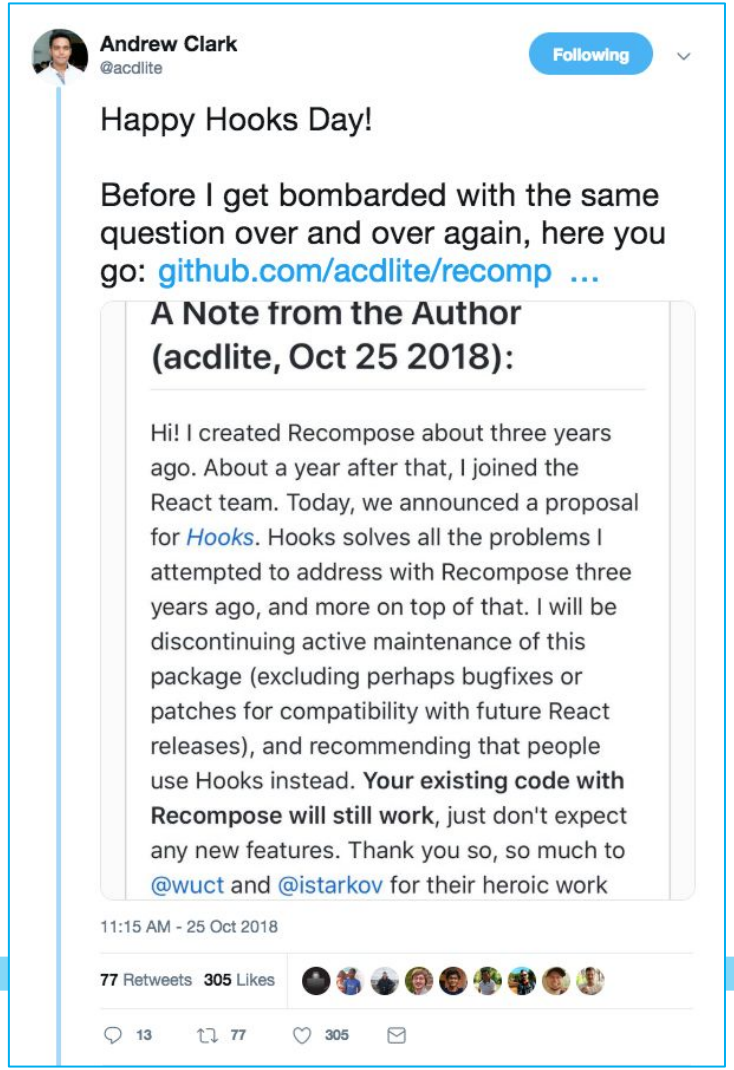
High-Order-Components

and Recompose

But then...


React
@reactjs
Following
just gonna leave this here
Introducing Hooks – React
A JavaScript library for building user interfaces
reactjs.org
10:25 AM - 25 Oct 2018
706 Retweets 1,728 Likes
44
706
1.7K
Tweet your reply

And then...



Andrew Clark
@acdlite

Following

Happy Hooks Day!

Before I get bombarded with the same question over and over again, here you go: github.com/acdlite/recomp ...

A Note from the Author (acdlite, Oct 25 2018):

Hi! I created Recompose about three years ago. About a year after that, I joined the React team. Today, we announced a proposal for *Hooks*. Hooks solves all the problems I attempted to address with Recompose three years ago, and more on top of that. I will be discontinuing active maintenance of this package (excluding perhaps bugfixes or patches for compatibility with future React releases), and recommending that people use Hooks instead. **Your existing code with Recompose will still work**, just don't expect any new features. Thank you so, so much to @wuct and @istarkov for their heroic work

11:15 AM - 25 Oct 2018

77 Retweets **305** Likes

13 77 305

About Myself

Nir Hadassi

4 years

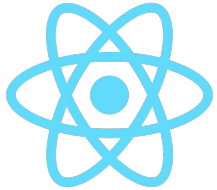
working at

Soluto

3 years

working with

React



nirsky

What are React Hooks?

"Hooks lets you use state and other React features without writing a class."

Introduced on React v16.7.0-alpha

Why?

CLASSES ARE

BAD

Why classes are bad?

- Complex components become hard to understand
- Classes confuse people (notorious `this`..)
- Classes confuse machines (don’t minify well)
- It’s hard to reuse stateful logic between classes

Agenda

1. Hooks Intro
    a. useState
    b. useRef
    c. useContext
    d. useEffect
2. Memo
3. Lazy

useState

0

Class with state

```
class CounterButton extends Component {
    constructor() {
        super();
        this.state = {
            count: 0
        }
    }
    render() {
        return <button onClick={() => this.setState(prevState => ({ count: prevState.count + 1
}))}>
            { this.state.count }
        </button>
    }
}
```

With useState

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


const Counter = props => {
    const [count, setCount] = useState(0);


    return <button onClick={() => setCount(count + 1)}>
            { count }
    </button>
}
```

With useState

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


const Counter = props => {
    const [count, setCount] = useState(0);


    return <button onClick={() => setCount(count + 1)}>
            { count }
    </button>
}
```

With useState

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


const Counter = props => {
    const [count, setCount] = useState(0);


    return <button onClick={() => setCount(count + 1)}>
            { count }
    </button>
}
```

With useState

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


const Counter = props => {
    const [count, setCount] = useState(0);


    return <button onClick={() => setCount(count + 1)}>
            { count }
    </button>
}
```

Multiple State Variables

```
const Player = props => {
    const [volume, setVolume] = useState(0);
    const [position, setPosition] = useState(0);
    const [paused, setPaused] = useState(true);

    const onClick = () => {
        setPosition(0);
        setPaused(false);
    }
}
```

Multiple State Variables

```
const Player = props => {
    const [state, setState] = useState({
        volume: 0,
        position: 0,
        paused: true
    });

    const onClick = () => {
        setState({
            ...state,
            position: 0,
            paused: false
        })
    }
}
```

useContext

Using context without hooks

```
import { ThemeContext } from './context';


const Counter = props => {
   const [count, setCount] = useState(0);


   return (
       <ThemeContext.Consumer>
           {theme => (
               <Button theme={theme} onClick={...}>
                   {count}
               </Button>
           )}
       </ThemeContext.Consumer>
   )
}
```

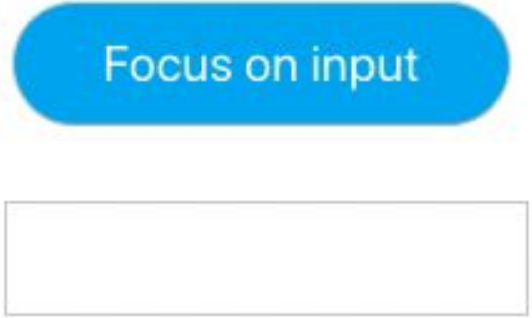
useContext hook

```
import React, { useContext } from 'react';
import { ThemeContext } from './context';

const Counter = props => {
   const [count, setCount] = useState(0);
   const theme = useContext(ThemeContext)

   return (
       <Button theme={theme} onClick={...}>
           {count}
       </Button>
   )
}
```

useContext hook

```
import React, { useContext } from 'react';
import { ThemeContext } from './context';

const Counter = props => {
   const [count, setCount] = useState(0);
   const theme = useContext(ThemeContext)

   return (
       <Button theme={theme} onClick={...}>
           {count}
       </Button>
   )
}
```

useRef



useRef

```
const TextInputWithFocusButton = (props) => {
    const inputRef = useRef();

    return (
     <>
        <input ref={inputRef} type="text" />
        <button onClick={() => inputRef.current.focus()}>
            Focus the input
        </button>
     </>
    );
}
```

useRef

```
const TextInputWithFocusButton = (props) => {
    const inputRef = useRef();

    return (
     <>
        <input ref={inputRef} type="text" />
        <button onClick={() => inputRef.current.focus()}>
            Focus the input
        </button>
     </>
    );
}
```

useRef

```
const TextInputWithFocusButton = (props) => {
    const inputRef = useRef();

    return (
     <>
        <input ref={inputRef} type="text" />
        <button onClick={() => inputRef.current.focus()}>
            Focus the input
        </button>
     </>
    );
}
```

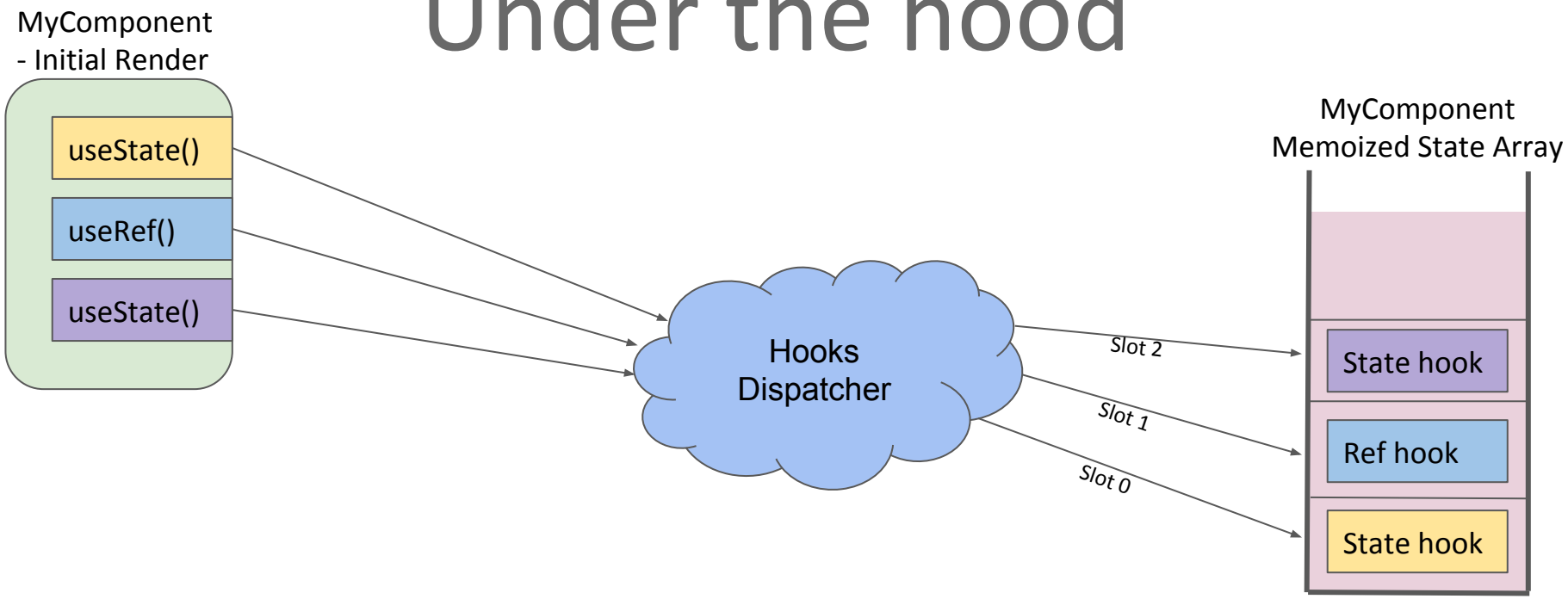
useRef

```
const TextInputWithFocusButton = (props) => {
    const inputRef = useRef();

    return (
     <>
        <input ref={inputRef} type="text" />
        <button onClick={() => inputRef.current.focus()}>
            Focus the input
        </button>
     </>
    );
}
```

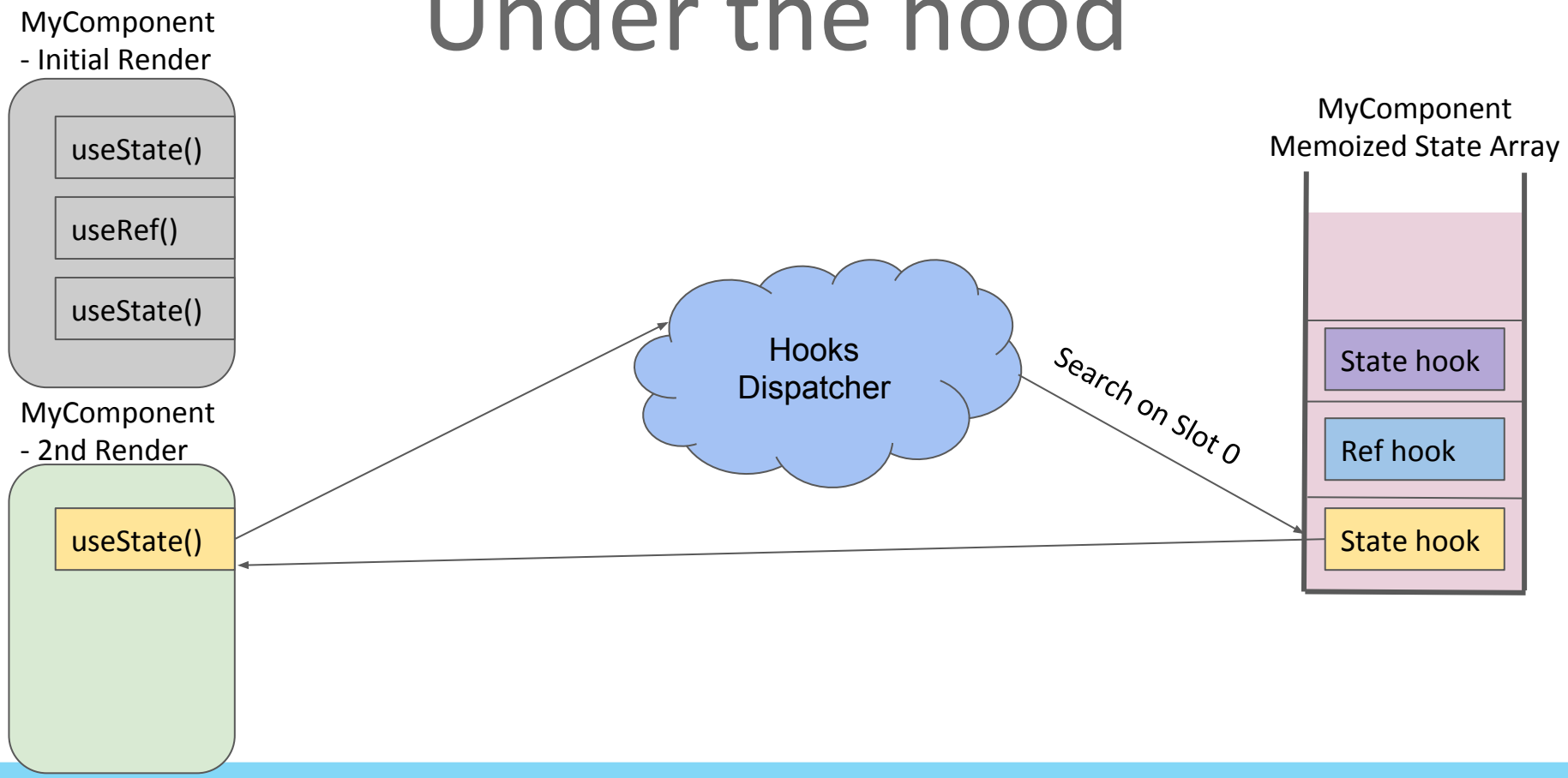
Rules of Hooks

- Only Call Hooks at the Top Level!
  - Don’t call Hooks inside loops, conditions, or nested functions
  - Order Matters!
- Only Call Hooks from React Functions
  - Or from custom hooks

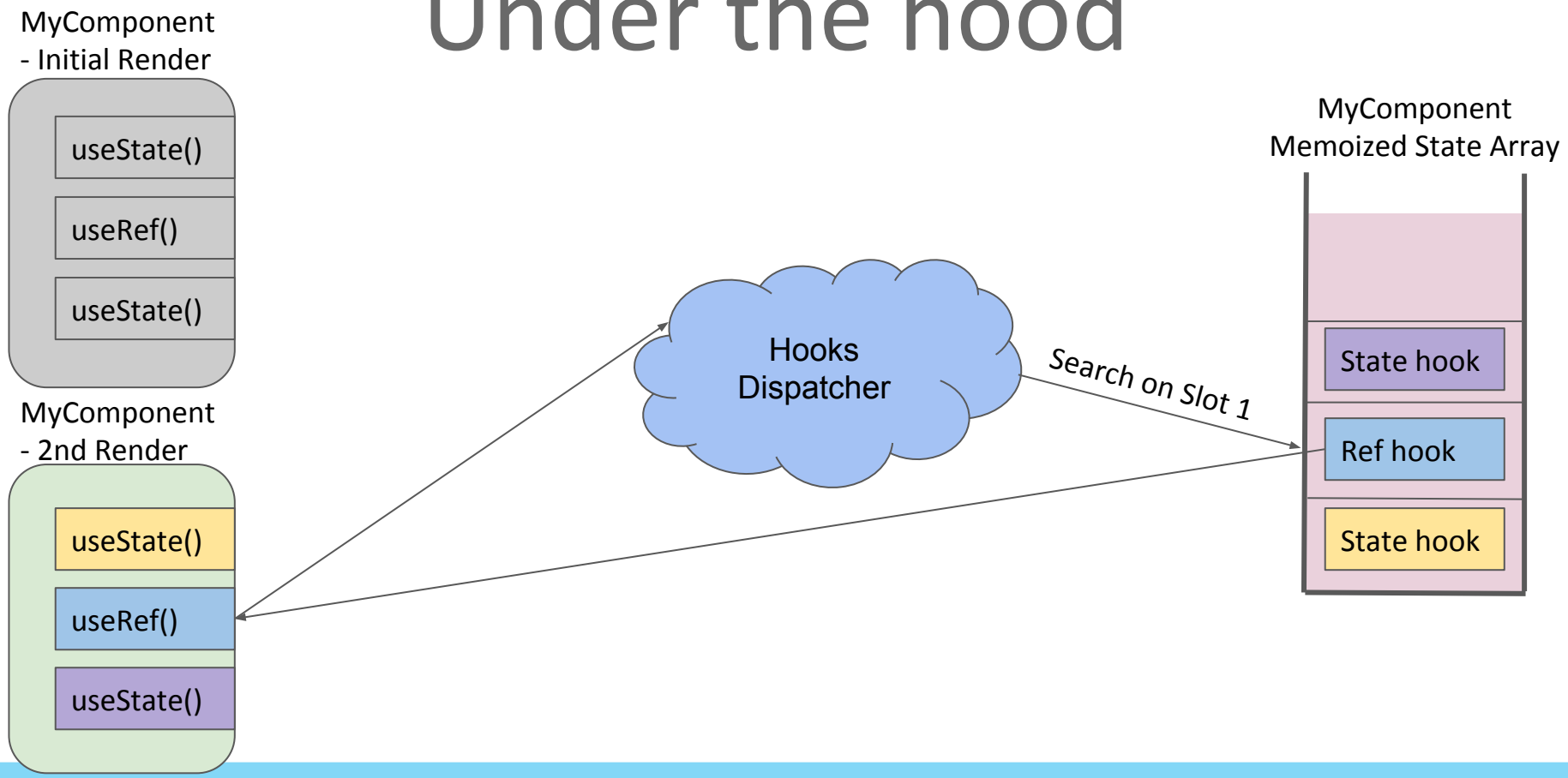
Under the hood



Under the hood



Under the hood



useEffect

Executing something on every render using lifecycle

```
class CounterButton extends Component {
    constructor() {
        super();
        this.state = {count: 0}
    }
    componentDidMount() {
        console.log(`The count is now ${this.state.count}`)
    }
    componentDidUpdate() {
        console.log(`The count is now ${this.state.count}`)
    }
    render() {
        return <button onClick={() => this.setState(prevState => ({ count: prevState.count + 1 }))}>
            { this.state.count }
       </button>
    }
}
```

Executing something on every render using useEffect

```
const Counter = props => {
    const [count, setCount] = useState(0);
    useEffect(() => {
        console.log(`The count is now ${count}`)
    });

    return <button onClick={() => setCount(count + 1)}>
            { count }
    </button>
}
```

Executing something on every render using useEffect

```
const Counter = props => {
    const [count, setCount] = useState(0);
    useEffect(() => {
        console.log(`The count is now ${count}`)
    });

    return <button onClick={() => setCount(count + 1)}>
            { count }
    </button>
}
```

Effects with Cleanup

```
useEffect(() => {
    console.log(`The count is now ${count}`);
    return function cleanup() {
        console.log('cleaning up');
    }
});
```

```
The count is now 0
--- Button onClick ---
cleaning up
The count is now 1
--- Button onClick ---
cleaning up
The count is now 2
--- Button onClick ---
cleaning up
The count is now 3
>
```

Effects with Cleanup

```
useEffect(() => {
    console.log(`The count is now ${count}`);
    return function cleanup() {
        console.log('cleaning up');
    }
});
```

```
The count is now 0
--- Button onClick ---
cleaning up
The count is now 1
--- Button onClick ---
cleaning up
The count is now 2
--- Button onClick ---
cleaning up
The count is now 3
>
```

Should my effect run on every render?

Consider the next scenario...

```
//timer changes every 100ms
const Counter = ({timer}) => {
    const [count, setCount] = useState(0);
    useEffect(() => {
        console.log(`The count is now ${count}`)
    });

    return <MyComponent onClick={() => setCount(count + 1)} timer={timer}>
            { count }
    </MyComponent>
}
```

```
31 The count is now 0
71 The count is now 0
   The count is now 1
   The count is now 1
46 The count is now 1
   The count is now 2
   The count is now 2
34 The count is now 2
   The count is now 3
100 The count is now 3
>
```

useEffect 2nd parameter

```
useEffect(() => {
    console.log(`The count is now ${count}`)
}, [count]);
```

componentWillReceiveProps

```
componentWillReceiveProps(nextProps) {
   if (this.props.timer !== nextProps.timer) {
        console.log(`The timer is now ${nextProps.timer}`)
   }
}
```

componentWillReceiveProps - hooks version

```
useEffect(() => {
    console.log(`Timer is now ${props.timer}`);
}, [props.timer]);
```

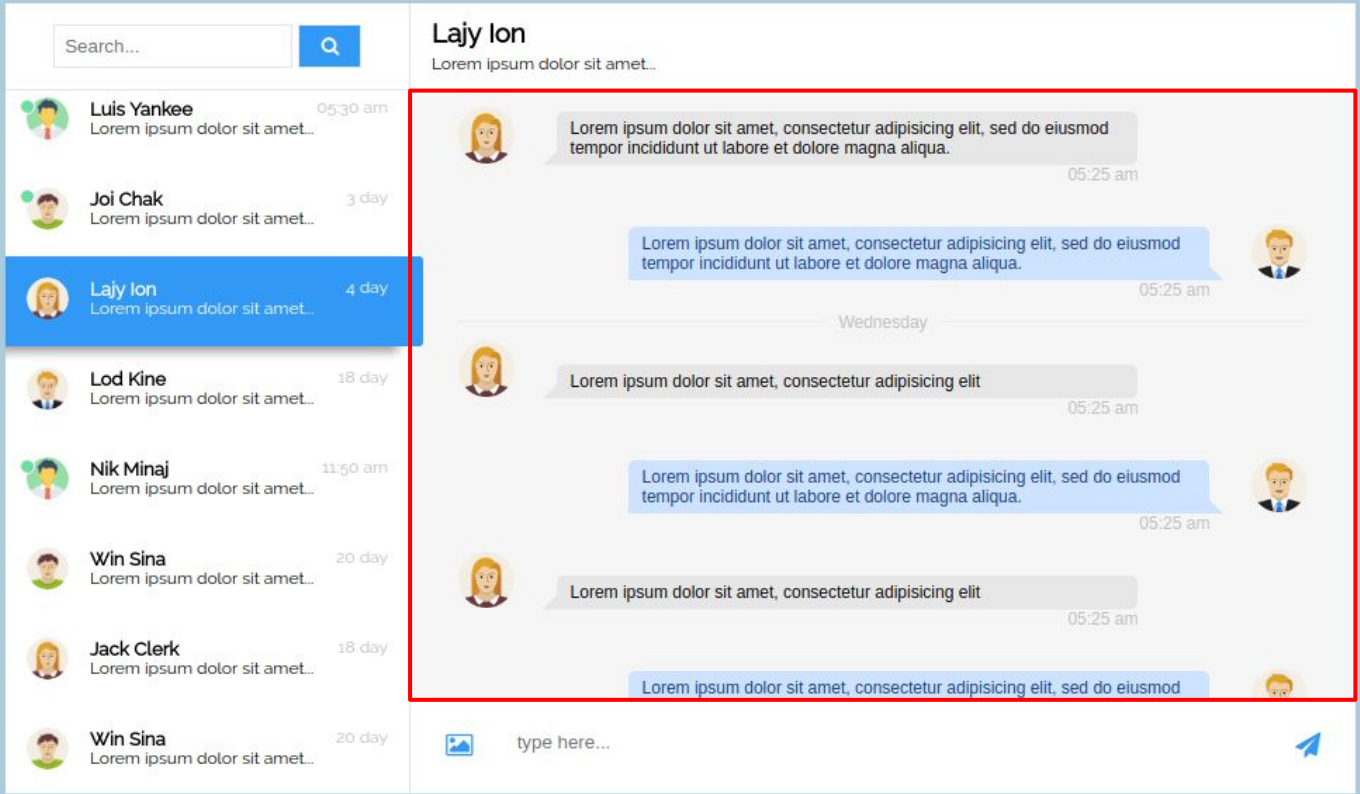
componentDidMount - hooks version

```
useEffect(() => {
    console.log(`I just mounted!`)
}, []);
```

componentWillUnmount - hooks version

```
useEffect(() => {
    return function cleanup(){
        console.log(`I'm unmounting!`)
    }
}, []);
```

Let’s combine what we learned so far


Search...
Luis Yankee
05:30 am
Lorem ipsum dolor sit amet...
Joi Chak
3 day
Lorem ipsum dolor sit amet...
Lajy Ion
4 day
Lorem ipsum dolor sit amet...
Lod Kine
18 day
Lorem ipsum dolor sit amet...
Nik Minaj
11:50 am
Lorem ipsum dolor sit amet...
Win Sina
20 day
Lorem ipsum dolor sit amet...
Jack Clerk
18 day
Lorem ipsum dolor sit amet...
Win Sina
20 day
Lorem ipsum dolor sit amet...
Lajy Ion
Lorem ipsum dolor sit amet...
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
05:25 am
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
05:25 am
Wednesday
Lorem ipsum dolor sit amet, consectetur adipisicing elit
05:25 am
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
05:25 am
Lorem ipsum dolor sit amet, consectetur adipisicing elit
05:25 am
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod
type here...

```
class ChatPage extends Component {
    constructor() {
        super();
        this.state = {
            messages: []
        }
        this.onNewMessage = this.onNewMessage.bind(this);
    }
    componentDidMount () {
        SocketClient.subscribeForNewMessages (this.props.roomId, this.onNewMessage);
    }
    componentWillUnmount () {
        SocketClient.unsubscribe (this.props.roomId);
    }
    componentWillReceiveProps (nextProps) {
        if (nextProps.roomId !== this.props.roomId) {
            this.setState({ messages: [] });
            SocketClient.unsubscribe (this.props.roomId);
            SocketClient.subscribeForNewMessages (nextProps.roomId, this.onNewMessage);
        }
    }
    onNewMessage (message) {
        this.setState({ messages: [...this.state.messages, message] })
    }
    render() {
        return this.state.messages.map((text, i) => <div key={i}>{text}</div>)
    }
}
```

```
class ChatPage extends Component {
    constructor() {
        super();
        this.state = {
            messages: []
        }
        this.onNewMessage = this.onNewMessage.bind(this);
    }
    componentDidMount () {
        SocketClient.subscribeForNewMessages (this.props.roomId, this.onNewMessa
    }
    componentWillUnmount () {
        SocketClient.unsubscribe (this.props.roomId);
    }
    componentWillReceiveProps (nextProps) {
        if (nextProps.roomId !== this.props.roomId) {
            this.setState({ messages: [] });
            SocketClient.unsubscribe (this.props.roomId);
            SocketClient.subscribeForNewMessages (nextProps.roomId, this.onNewMessage);
        }
    }
    onNewMessage (message) {
        this.setState({ messages: [...this.state.messages, message] })
    }
    render () {
        return this.state.messages.map((text, i) => <div key={i}>{text}</div>)
    }
}
```

```
constructor() {
    super();
    this.state = {
        messages: []
    }
    this.onNewMessage =
        this.onNewMessage.bind(this);
}
```

```
class ChatPage extends Component {
    constructor() {
        super();
        this.state = {
            messages: []
        }
        this.onNewMessage = this.onNewMessage.bind(this);
    }
    componentDidMount () {
        SocketClient.subscribeForNewMessages (this.props.roomId, this.onN
    }
    componentWillUnmount () {
        SocketClient.unsubscribe (this.props.roomId);
    }
    componentWillReceiveProps (nextProps) {
        if (nextProps.roomId !== this.props.roomId)
            this.setState({ messages: [] });
            SocketClient.unsubscribe (this.p        omId);
            SocketClient.subscribeForNe        ages (nextProps.roomId, this.onNewMessage);
        }
    }
    onNewMessage (message) {
        this.setState({ messages: [...this.state.messages, message] })
    }
    render() {
        return this.state.messages.map((text, i) => <div key={i}>{text}</div>)
    }
}
```

```
onNewMessage(message) {
    this.setState({
        messages: [...this.state.messages, message]
    })
}
```

```
class ChatPage extends Component {
    constructor() {
        super();
        this.state = {
            messages: []
        }
        this.onNewMessage = this.onNewMessage
    }
    componentDidMount () {
        SocketClient.subscribeForNewMessages (this.props.roomId, thi
    }
    componentWillUnmount () {
        SocketClient.unsubscribe (this.props.roomId);
    }
    componentWillReceiveProps (nextProps) {
        if (nextProps.roomId !== this.props.roomId) {
            this.setState ({ messages: [] });
            SocketClient.unsubscribe (this.props.roomId);
            SocketClient.subscribeForNewMessages (nextProps.roomId, this.onNewMessage);
        }
    }
    onNewMessage (message) {
        this.setState ({ messages: [...this.state.messages, message] })
    }
    render () {
        return this.state.messages.map((text, i) => <div key={i}>{text}</div>)
    }
}
```

```
componentDidMount () {
    SocketClient.subscribeForNewMessages (
        this.props.roomId,
        this.onNewMessage
    );
}
```

```
class ChatPage extends Component {
    constructor() {
        super();
        this.state = {
            messages: []
        }
        this.onNewMessage = this.onNewMessage.bind(this);
    }
    componentDidMount () {
        SocketClient.subscribeForNewMessages (this.props.roomId, this.onNewMessa
    }
    componentWillUnmount () {
        SocketClient.unsubscribe (this.props.roomId);
    }
    componentWillReceiveProps (nextProps) {
        if (nextProps.roomId !== this.props.roomId) {
            this.setState({ messages: [] });
            SocketClient.unsubscribe (this.props.roomId);
            SocketClient.subscribeForNewMessages (nextProps.roomId, this.onNewMessage);
        }
    }
    onNewMessage (message) {
        this.setState({ messages: [...this.state.messages, message] })
    }
    render() {
        return this.state.messages.map((text, i) => <div key={i}>{text}</div>)
    }
}
```

```
componentWillUnmount () {
    SocketClient.unsubscribe (
        this.props.roomId
    );
}
```

```
class ChatPage extends Component {
    constructor() {
        super();
        this.state = {
            messages: []
        }
        this.onNewMessage = this.onNewMessage.bind(this);
    }
    componentDidMount () {
        SocketClient.subscribeForNewMessages (this.props.roo
    }
    componentWillUnmount () {
        SocketClient.unsubscribe (this.props.roomId)
    }
    componentWillReceiveProps (nextProps) {
        if (nextProps.roomId !== this.props.roomId) {
            this.setState ({ messages: [] });
            SocketClient.unsubscribe (this.props.roomId);
            SocketClient.subscribeForNewMessages (nextProps.roomId, this.onNewMessage);
        }
    }
    onNewMessage (message) {
        this.setState ({ messages: [...this.state.messages, message] })
    }
    render () {
        return this.state.messages.map((text, i) => <div key={i}>{text}</div>)
    }
}
```

```
componentWillReceiveProps(nextProps) {
    if (nextProps.roomId !== this.props.roomId) {
        this.setState({ messages: [] });
        SocketClient.unsubscribe(this.props.roomId);
        SocketClient.subscribeForNewMessages(
            nextProps.roomId,
            this.onNewMessage
        );
    }
}
```

```
class ChatPage extends Component {
    constructor() {
        super();
        this.state = {
            messages: []
        }
        this.onNewMessage = this.onNewMessage.bind(this);
    }
    componentDidMount () {
        SocketClient.subscribeForNewMessages (this.props.roo
    }
    componentWillUnmount () {
        SocketClient.unsubscribe (this.props.roomId);
    }
    componentWillReceiveProps (nextProps) {
        if (nextProps.roomId !== this.props.roomId) {
            this.setState({ messages: [] });
            SocketClient.unsubscribe (this.props.
            SocketClient.subscribeForNewMessa          rops.roomId, this.onNewMessage);
        }
    }
    onNewMessage (message) {
        this.setState({ mess    s: [...this.state.messages, message] })
    }
    render () {
        return this.state.messages.map((text, i) => <div key={i}>{text}</div>)
    }
}
```

```
render() {
    return this.state.messages
      .map((text, i) =>
            <div key={i}>{text}</div>
      )
}
```

```
const ChatPage = ({ roomId }) => {
    const [messages, setMessages] = useState([]);

    useEffect(() => {
        setMessages([]);
        const onNewMessage = (message) => setMessages([...messages, message]);
        SocketClient.subscribeForNewMessages(roomId, onNewMessage);

        return () => SocketClient.unsubscribe(roomId);
    }, [roomId]);

    return messages.map((text, i) => <div key={i}>{text}</div>)
}
```

```
const ChatPage = ({ roomId }) => {
    const [messages, setMessages] = useState([]);

    useEffect(() => {
        setMessages([]);
        const onNewMessage = (message) => setMessages([...messages, message]);
        SocketClient.subscribeForNewMessages(roomId, onNewMessage);

        return () => SocketClient.unsubscribe(roomId);
    }, [roomId]);

    return messages.map((text, i) => <div key={i}>{text}</div>)
}
```

```
const ChatPage = ({ roomId }) => {
    const [messages, setMessages] = useState([]);

    useEffect(() => {
        setMessages([]);
        const onNewMessage = (message) => setMessages([...messages, message]);
        SocketClient.subscribeForNewMessages(roomId, onNewMessage);

        return () => SocketClient.unsubscribe(roomId);
    }, [roomId]);

    return messages.map((text, i) => <div key={i}>{text}</div>)
}
```

```
const ChatPage = ({ roomId }) => {
    const [messages, setMessages] = useState([]);

    useEffect(() => {
        setMessages([]);
        const onNewMessage = (message) => setMessages([...messages, message]);
        SocketClient.subscribeForNewMessages(roomId, onNewMessage);

        return () => SocketClient.unsubscribe(roomId);
    }, [roomId]);

    return messages.map((text, i) => <div key={i}>{text}</div>)
}
```

```
const ChatPage = ({ roomId }) => {
    const [messages, setMessages] = useState([]);

    useEffect(() => {
        setMessages([]);
        const onNewMessage = (message) => setMessages([...messages, message]);
        SocketClient.subscribeForNewMessages(roomId, onNewMessage);

        return () => SocketClient.unsubscribe(roomId);
    }, [roomId]);

    return messages.map((text, i) => <div key={i}>{text}</div>)
}
```

```
const ChatPage = ({ roomId }) => {
    const [messages, setMessages] = useState([]);

    useEffect(() => {
        setMessages([]);
        const onNewMessage = (message) => setMessages([...messages, message]);
        SocketClient.subscribeForNewMessages(roomId, onNewMessage);

        return () => SocketClient.unsubscribe(roomId);
    }, [roomId]);

    return messages.map((text, i) => <div key={i}>{text}</div>)
}
```

```
const ChatPage = ({ roomId }) => {
    const [messages, setMessages] = useState([]);

    useEffect(() => {
        setMessages([]);
        const onNewMessage = (message) => setMessages([...messages, message]);
        SocketClient.subscribeForNewMessages(roomId, onNewMessage);

        return () => SocketClient.unsubscribe(roomId);
    }, [roomId]);

    return messages.map((text, i) => <div key={i}>{text}</div>)
}
```

Custom Hooks

What are Custom Hooks?

- Basically functions that run hooks
- Like any other function - they can can take any args and return whatever you want
- By convention - custom hook name start with “use”
- Like any other hook - must be called on the top level of our components

This is the custom hook:

```
const useChatMessages = (roomId) => {
    const [messages, setMessages] = useState([]);

    useEffect(() => {
        setMessages([]);
        const onNewMessage = (message) => setMessages([...messages, message]);
        SocketClient.subscribeForNewMessages(roomId, onNewMessage);

        return () => SocketClient.unsubscribe(roomId);
    }, [roomId]);

    return messages;
}
```

This is the (very short) component:

```
const ChatPage = ({ roomId }) => {
    const messages = useChatMessages(roomId);


    return messages.map((text, i) =>
                <div key={i}>{text}</div>)
}
```

```
class ChatPage extends Component {
    constructor() {
        super();
        this.state = {
            messages: []
        }
        this.onNewMessage = this.onNewMessage.bind(this);
    }
    componentDidMount () {
        SocketClient.subscribeForNewMessages (this.props.roomId, this.onNewMessage);
    }
    componentWillUnmount () {
        SocketClient.unsubscribe (this.props.roomId);
    }
    componentWillReceiveProps (nextProps) {
        if (nextProps.roomId !== this.props.roomId) {
            this.setState({ messages: [] });
            SocketClient.unsubscribe (this.props.roomId);
            SocketClient.subscribeForNewMessages (nextProps.roomId, this.onNewMessage);
        }
    }
    onNewMessage (message) {
        this.setState({ messages: [...this.state.messages, message] })
    }
    render () {
        return this.state.messages.map((text, i) => <div key={i}>{text}</div>)
    }
}
```

Remember how it used to look?
28 lines vs. 5 with custom hook

Custom Hooks allow us to..

- Easily share code
- Keep our components clean and readable
- Use Hooks from npm packages

useHook

One new React

https://ikgraf.github.io/react-hoo

- react-i18next/hooks Internationalization for react done right.
- react-immer-hooks useState and useReducer using Immer to update state.
- react-intersection-visible-hook React hook to track the visibility of a functional component.
- react-pirate React lifecycle and utilities hooks.
- react-powerhooks Hooks api for react-powerplug.
- react-selector-hooks Collection of hook-based selector factories for declarations outside of render.
- react-use Collection of essential hooks.
- react-useFormless React hook to handle forms state.
- react-use-form-state React hook for managing form and inputs state.
- react-use-idb React hook for storing value in the browser using IndexDB.
- react-wait Complex Loader Management Hook for React Applications.
- react-window-communication-hook React hook to communicate among browser contexts (windows, iframes).
- react-with-hooks Ponyfill for the proposed React Hooks API.
- reaktion useState like hook for state management.
- redux-react-hook React hooks using mapped state from Redux store.
- react-hooks-visibility-sensor Tell whether an element has scrolled into view or not.
- resynced Multiple state management using React Hooks API.
- rxjs-hooks An easy way to use RxJS v6+ with react hooks.
- the-platform Browser API's turned into React Hooks and Suspense-friendly React elements for common situations.
- use-abortable-fetch React hook that does a fetch and aborts when the components is unloaded or a different request is made.
- use-events A set of React Hooks to handle mouse events.
- use-immer A hook to use immer to manipulate state.
- use-redux A hook to bind redux.
- use-simple-undo Simple implementation of undo/redo functionality.
- use-socketio React hooks to use with https://socket.io/.
- use-store React hook for subscribing to your single app state (works with your current Redux app).
- use-undo Hook to implement Undo and Redo functionality.
- usePosition Hook to get position top left of an element.
- usescroll Hook to automatically update navigation based on scroll position.

Other hooks

- useReducer
- useCallback
- useMemo
- useImperativeMethods
- useLayoutEffect

Memo

React.memo

PureComponent for function components

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

const MyComponent = props => { ... }

export default memo(MyComponent);
```

Introduced on React 16.6

React.memo

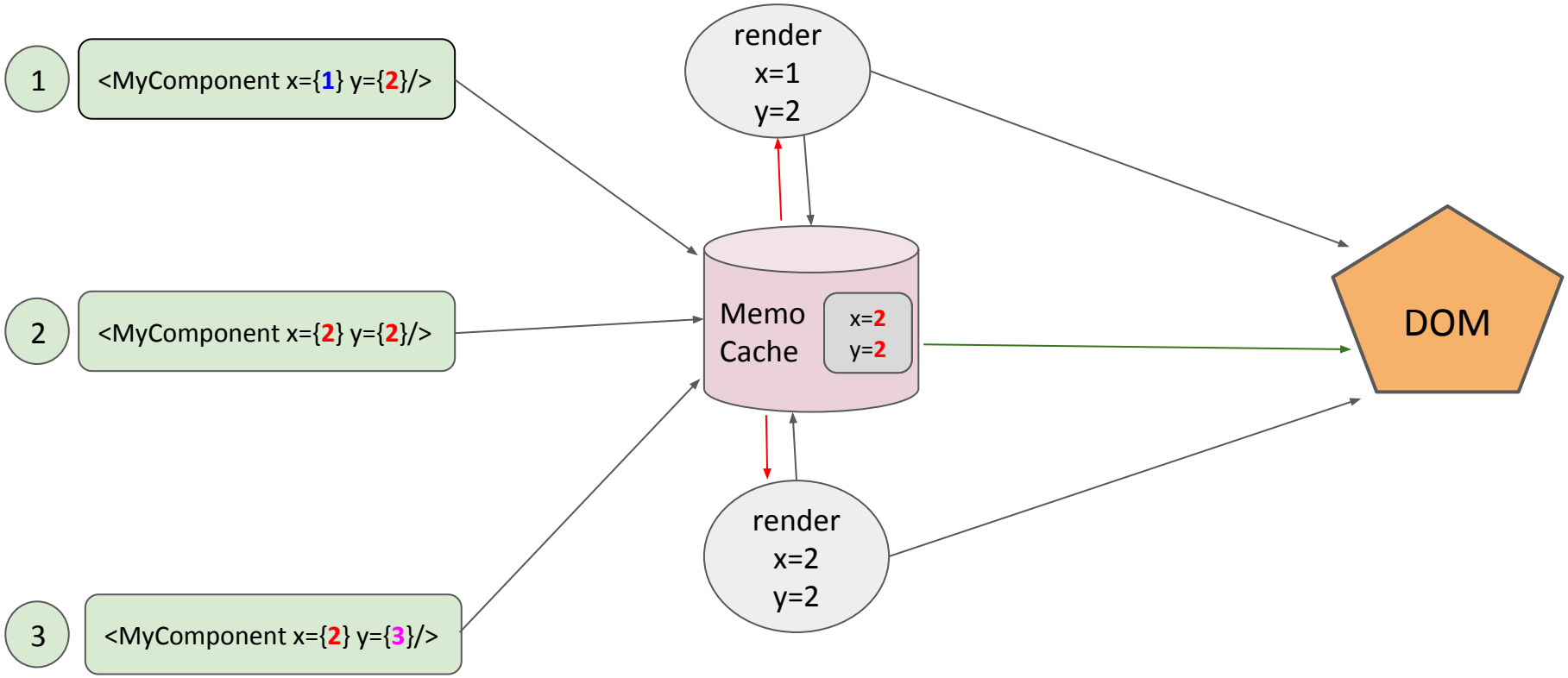
PureComponent for function components

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


const MyComponent = props => { ... }
const areEqual = (prevProps, nextProps) => { ... }


export default memo(MyComponent, areEqual);
```

Introduced on React 16.6

Rendering “MyComponent” with Memo

areEqual = (prev, next) => prev.x === next.x



Suspense + Lazy

Lazy

Code-Splitting

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

const OtherComponent = React.lazy(() => import('./OtherComponent'));

const MyComponent = (props) =>
   <div>
       <OtherComponent/>
   </div>
```

Introduced on React 16.6

Lazy

Code-Splitting

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

const OtherComponent = React.lazy(() => import('./OtherComponent'));

const MyComponent = (props) =>
   <div>
       <OtherComponent/>
   </div>
```

Introduced on React 16.6

Lazy

Code-Splitting

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

const OtherComponent = React.lazy(() => import('./OtherComponent'));

const MyComponent = (props) =>
   <div>
       <OtherComponent/>
   </div>
```

Introduced on React 16.6

Lazy

Code-Splitting

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

const OtherComponent = React.lazy(() => import('./OtherComponent'));

const MyComponent = (props) =>
   <div>
       <OtherComponent/>
   </div>
```

Introduced on React 16.6

Lazy + Suspense

Code-Splitting

```
import React, { lazy, Suspense } from 'react';

const OtherComponent = React.lazy(() => import('./OtherComponent'));

const MyComponent = (props) =>
   <div>
       <Suspense fallback={<div>Loading..</div>}>
           <OtherComponent />
       </Suspense>
   </div>
```

Introduced on React 16.6

Lazy + Suspense

Code-Splitting

```
import React, { lazy, Suspense } from 'react';


const OtherComponent = React.lazy(() => import('./OtherComponent'));


const MyComponent = (props) =>
   <div>
       <Suspense fallback={<div>Loading..</div>}>
           <OtherComponent />
       </Suspense>
   </div>
```

Introduced on React 16.6

Lazy + Suspense

Code-Splitting

```
import React, { lazy, Suspense } from 'react';

const OtherComponent = React.lazy(() => import('./OtherComponent'));

const MyComponent = (props) =>
   <div>
       <Suspense fallback={<div>Loading..</div>}>
           <OtherComponent />
       </Suspense>
   </div>
```

Introduced on React 16.6

```
const Home = lazy(() => import('./components/Home'));
const Posts = lazy(() => import('./components/Posts'));

const App = () => (
   <Router>
       <Suspense fallback={<Loading />}>
           <Switch>
               <Route exact path="/" component={Home} />
               <Route path="/posts" component={Posts} />
           </Switch>
       </Suspense>
   </Router>
)
```

Home Posts

App

What is Lorem Ipsum? Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

Why do we use it?

It is a long established fact that a reader will be distracted by the readable content of a page when looking at its layout. The point of using Lorem Ipsum is that it has a more-or-less normal distribution of letters, as opposed to using 'Content here, content here', making it look like readable English. Many desktop publishing packages and web page editors now use Lorem Ipsum as their default model text, and a search for 'lorem ipsum' will uncover many web sites still in their infancy. Various versions have evolved over the years, sometimes by accident, sometimes on purpose (injected humour and the like).

What is Lorem Ipsum? Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic

Elements Console Audits Sources Netw

View: Group by frame

Filter Hide data URLs

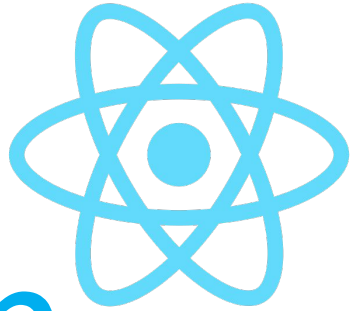
All XHR JS CSS Img Media Font Doc WS Manifest Other

100 ms 200 ms 300 ms 400 ms 500

Name	S...	Type	Initi...	Size	Time	P...	Waterfall
localhost	200	d...	Other	8...	579...	...	
main.0...	(...	st...	(ind...	0 B	Pen...	...	
main.bf...	(...	sc...	(ind...	0 B	Pen...	...	

3 requests | 841 B transferred | Finish: 579 ms

Soluto
by asurion

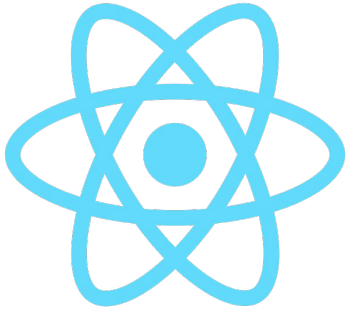


Hook me up with Questions

Nir_Hadassi

nirsky

Soluto
by asurion



THANKS!


Nir_Hadassi


nirsky