

FULLSTACK DEVELOPMENT REACT

REACT

- » Component based library to build composable UIs
- » OpenSourced in 2013
- » Implemented and Maintained by Facebook
- » Learn once, write anywhere
 - » React-Native
 - » React-Native-Desktop
 - » React-Native-Windows

COMPONENTS

“Components let you split the UI into independent, reusable pieces.”¹

- » Main Building block of a React App
- » Describe the look and feel of one section in the UI

¹ <https://reactjs.org/docs/components-and-props.html>

REACT COMPONENTS

```
const Button = () => {  
  return (  
    <button type='button'>  
      A button  
    </button>  
  )  
}
```

// Usage

```
React.renderComponent(<Button />, document.body)
```

REACT CLASS COMPONENTS

» Alternative syntax for components

```
class Button extends React.Component {  
  render() {  
    return (  
      <button type='button'>  
        A button  
      </button>  
    )  
  }  
}
```

// Usage

```
React.renderComponent(<Button />, document.body)
```

JSX

» JavaScript XML

» extension to write XML in JS

» Allows to combine data preparation with render logic

```
const Button = () => {  
  return [  
    <button type='button'>  
      A button  
    </button>  
  ]  
}
```

REACT WITHOUT JSX

» React can be used without JSX

```
const Button = () => {  
  return React.createElement(  
    'button',  
    { type: 'button' },  
    'A button'  
  )  
}
```

WHICH COMPONENTS DO YOU SEE

Sign In

Email

Password

Sign in

[Sign Up](#)

WHICH COMPONENTS DO YOU SEE

App

Button

I owe somebody Somebody owes me

User

Amount

Select Create

A user	10,40\$	Paid
A user	10,40\$	
A user	10,40\$	
A user	-10,40\$	Paid
A user	10,40\$	

BUILDING THE FIRST REACT COMPONENT

EMBEDDING EXPRESSIONS

```
const CurrentTime = () => {  
  return (  
    <h1>  
      {(new Date()).toLocaleDateString()}  
    </h1>  
  )  
}
```

EMBEDDING EXPRESSIONS

```
const FagoMenu = () => {  
  return (  
    <a href={` /menu/${(new Date()).toLocaleDateString()}`}>  
      Go to todays menu  
    </a>  
  )  
}
```

CONDITIONAL RENDERING

```
const CurrentTime = () => {  
  // ...  
  return (  
    <h1>  
      {isToday  
        ? 'Today'  
        : 'Not Today'}  
    </h1>  
  )  
}
```

CONDITIONAL RENDERING

```
const CurrentTime = () => {  
  // ...  
  return (  
    <h1>  
      {isToday && 'Today'}  
      {!isToday && 'Not today'}  
    </h1>  
  )  
}
```

LOOP OVER ARRAYS

```
const UserList = ({ users }) => {  
  return (  
    <ul>  
      {users.map((user) => {  
        return (<li key={user.id}>{user.name}</li>)  
      })}  
    </ul>  
  )  
}
```

FRAGMENTS

» Groups a list of children without adding a dom element

```
const AComponent = () => {  
  return (  
    <>  
      <label>An input</label>  
      <input type='text' />  
    </>  
  )  
}
```


KEYED FRAGMENTS

» Same as fragment but a key can be provided (eg.: definition list)

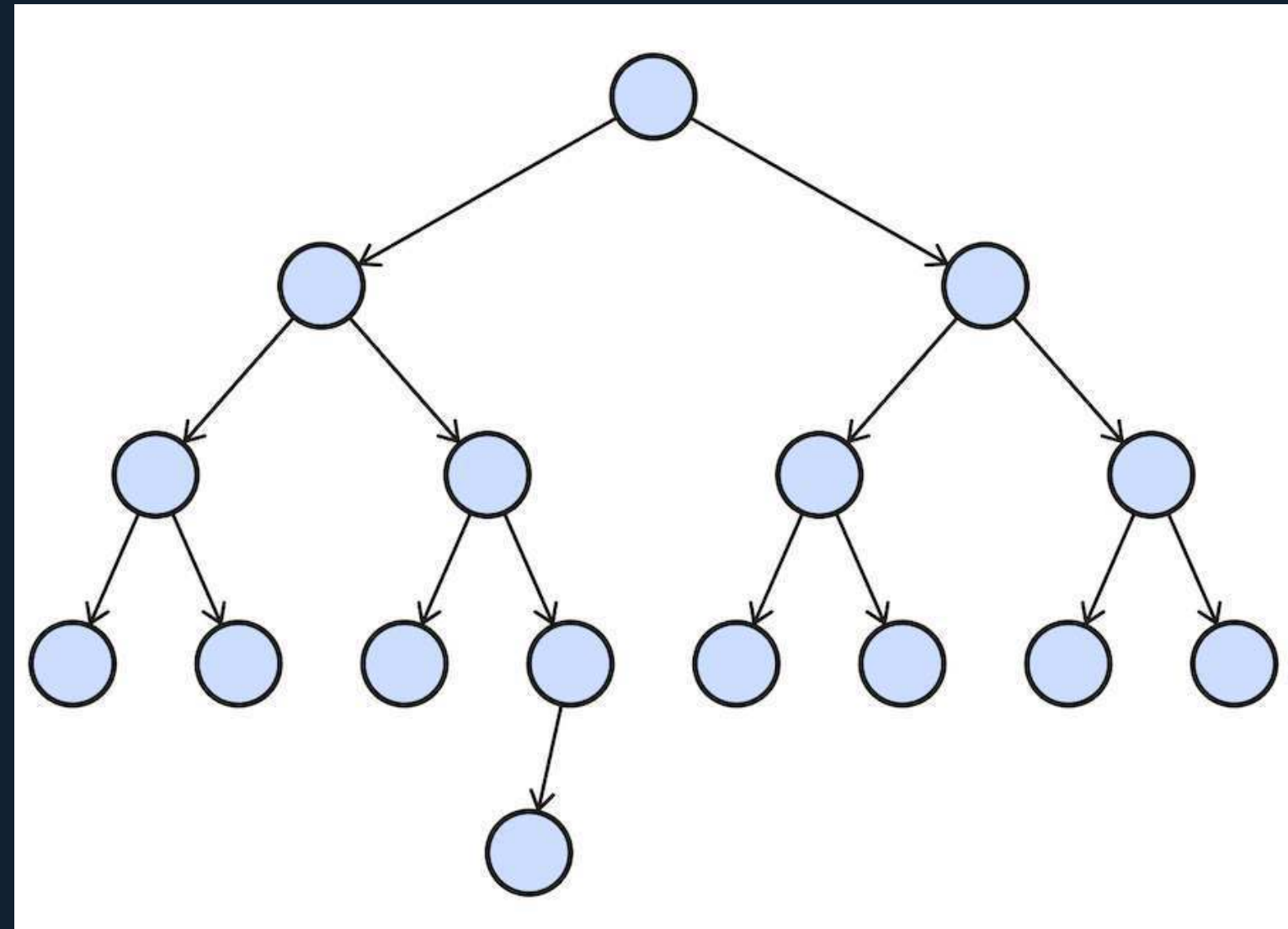
```
const AComponent = ({ items }) => {  
  return (  
    <dl>  
      {items.map(item => (  
        // Without the `key`, React will fire a key warning  
        <React.Fragment key={item.id}>  
          <dt>{item.term}</dt>  
          <dd>{item.description}</dd>  
        </React.Fragment>  
      ))}  
    </dl>  
  )  
}
```

KEY PROPERTY IN LOOPS

- » Is required when iterating over lists
- » Helps react to decide if an element needs to be rerendered
- » Video explanation
- » Detailed explanation

COMPONENT COMPOSITION

» Components can be nested and composed together



REACT PROPS

- » Possibility to customize components
 - » Can be seen as component configuration
- » Props are passed to the component
 - » A component at a lower level of the tree can't modify given props directly

REACT PROPS

```
const Button = ({ children, disabled = false }) => {  
  //      ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^  
  // props which are passed to the component  
  
  return (  
    <button disabled={disabled} className='button'>  
      {children}  
    </button>  
  )  
}
```

```
const usage = <Button disabled>A button</Button>  
// 1)      ^^^^^^^  
// 2)      ^^^^^^^  
// 1) shortcut for disabled={true}  
// 2) child components/nodes passed to a component
```

EXERCISE: 30MIN

- » I'll create breakout rooms
- » Fork/clone the following <https://github.com/webpapaya/fhs-react-redux-starter-kit>
- » execute `npm run start:storybook`

EXERCISE: 30MIN

- » extend the sign in component to display
 - » Input with username
 - » Input with password
 - » A Sign In Button (reuse existing button)
- » Extract an Input Component
 - » make it configurable via props

STATE IN REACT

- » What we've seen so far:
 - » Components can render chunks of UI
 - » Components can be nested

STATE IN REACT

“How can we interact with components?”

STATE IN REACT

“The State of a component is an object that holds some information that may change over the lifetime of the component ⁵”

⁵ [geeksforgeeks.com](https://www.geeksforgeeks.com/react-state/)

REACT STATE (WITHOUT HOOKS)

```
class ToggleButton extends React.Component {
  state = { backgroundColor: 'red' };
  // define a default value for background color

  toggleBackgroundColor = () => {
    const nextBackgroundColor = backgroundColor === 'red' ? 'blue' : 'red'
    this.setState({ backgroundColor: nextBackgroundColor })
    //      ^^^^^^^^^
    // setState calls render method with updated state
  }
  render() {
    return (
      <button
        onClick={() => this.toggleBackgroundColor() }
        style={{ backgroundColor: this.state.backgroundColor }}
      >
        {children}
      </button>
    );
  }
}
```

REACT STATE (WITH HOOKS)

» Alternative syntax with hooks

```
const ToggleButton = () => {
  const [backgroundColor, setBackground] = useState('red')
  // 1)                                     ^^^^^^^^^
  // 2) ^^^^^^^^^^^^^^^^^^^^^^^^^
  // 3)                                     ^^^^^^^^^^^^^^^^^^^^^
  // 1) define a state with a default value "red"
  // 2) the current value of the state
  // 3) function to set the state to something else

  return (
    <button
      onClick={() => setBackground(backgroundColor === 'red' ? 'blue' : 'red')}
      style={{ backgroundColor }}
    >
      {children}
    </button>
  )
}
```

REACT HOOKS

“Hooks allow you to reuse stateful logic without changing your component hierarchy. **React Docs**”

REACT HOOKS

- » Introduced recently to reduce boilerplate
- » Makes it possible to use state in functional components
 - » Previously one had to convert between functional/class components when state introduced
- » hooks are prefixed with use
- » Can't be called inside loops, conditions or nested

USESTATE

```
const App = () => {  
  const [count, setCount] = useState(0)  
  const handleIncrement = () => setCount(count + 1)  
  
  return (  
    <div>  
      <div>{count}</div>  
      <button onClick={handleIncrement}>Increment by 1</button>  
    </div>  
  )  
}
```

EXTRACT INTO CUSTOM HOOK

```
const useCounter = () => {
  const [count, setCount] => useState(0);
  const handleIncrement = () => setCount(count + 1);
  return { count, handleIncrement };
}

const App = () => {
  const {count, handleIncrement} => useCounter();

  return (
    <div>
      <div>{count}</div>
      <button onClick={handleIncrement}>Increment by 1</button>
    </div>
  );
}
```


STATE VS. PROPS

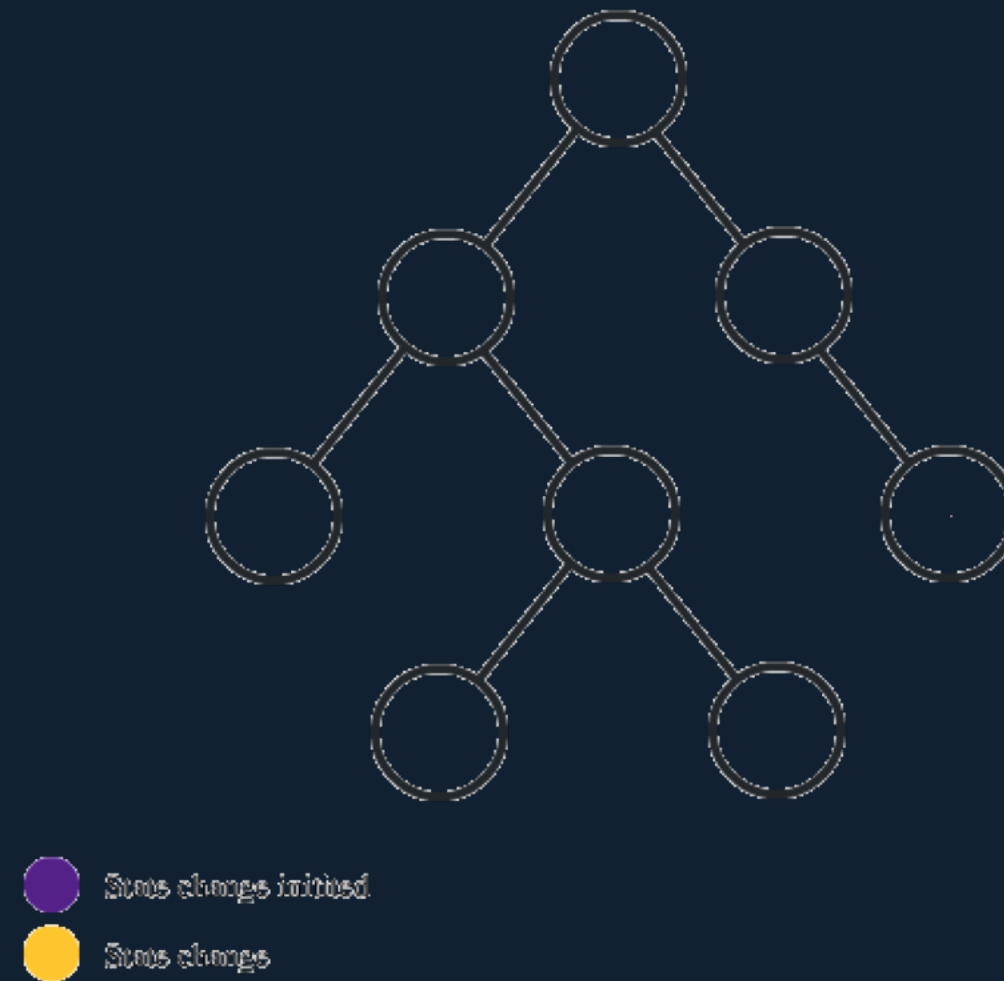
PROPS	STATE
Can get initial value from parent Component?	Yes
Can be changed by parent Component?	Yes
Can set default values inside Component?*	Yes
Can change inside Component?	No
Can set initial value for child Components?	Yes
Can change in child Components?	Yes

source

UNIDIRECTIONAL DATAFLOW

- » Props only flow from parent to children
- » Parent is responsible to update data
 - » might provide callbacks to do so
- » set state rerenders all children of component

UNIDIRECTIONAL DATAFLOW



“Source”

VIRTUAL DOM

- » makes DOM updates faster
- » after setState subtree is rerendered in memory
- » compares DOM to in memory representation
- » applies DOM changes when needed

FORMS WITH REACT HOOKS

```
const App = () => {
  const [username, setUsername] => useState('');
  //                               ^^^^^^^^^^^^^^^
  // define a new state with an initial value of empty string

  return (
    <div>
      <input onChange={(evt) => setUsername(evt.target.value)} value={username}>
        { /*                               ^^^^^^^^^^^^^^^^^^^^^^^^^ * / }
        { /* set the state of the username */ }
      <button onClick={() => console.log({ username })}>Submit form</button>
    </div>
  );
}
```


EXERCISE 1/2 (20 MINUTES)

- » Enhance the SignIn form
 - » store username/password in a component state
- » once the submit button is clicked log all values to the console

EXERCISE 2/2 (20 MINUTES)

» Bonus: extract a useForm hook

```
const [values, setValue] =  
useForm({  
  username: '',  
  password: ''  
})  
  
return <Input  
  onChange={setValue('username')} />
```


OTHER HOOKS

USEEFFECT⁴

```
// Executed on every rerender  
useEffect(() => {})
```

```
// Executed when component rendered initially  
useEffect(() => {}, [])
```

```
// Executed when component rendered initially  
// and when variable changes.  
useEffect(() => {}, [variable])
```

```
// Cleanup when component unmounts (eg. eventHandlers, setInterval/setTimeout)  
useEffect(() => {  
  // do something fancy  
  return () => { console.log('cleanup') }  
}, [variable])
```

⁴ this will be covered in more detail in the side effect lecture

PREVIOUS EXAMPLE

```
const useCounter = () => {
  const [count, setCount] => useState(0);
  const handleIncrement = () => setCount(count + 1);
  return { count, handleIncrement };
}

const App = () => {
  const {count, handleIncrement} => useCounter();

  return (
    <div>
      <div>{count}</div>
      <button onClick={handleIncrement}>Increment by 1</button>
    </div>
  );
}
```

UPDATE TITLE WITH COUNTER

```
const useCounter = () => {
  const [count, setCount] => useState(0);
  const handleIncrement = () => setCount(count + 1);
  return { count, handleIncrement };
}

const App = () => {
  const {count, handleIncrement} => useCounter();

  // Is executed when component is rendered for the first time
  // And when the counter variable changes.
  useEffect(() => {
    document.title = `Counter clicked ${count} times`;
  }, [count]);

  return (
    <div>
      <div>{count}</div>
      <button onClick={handleIncrement}>Increment by 1</button>
    </div>
  );
}
```

EXTRACT TO CUSTOM HOOK

```
const useCounter = () => {
  const [count, setCount] => useState(0);
  const handleIncrement = () => setCount(count + 1);
  useEffect(() => {
    document.title = `Counter clicked ${count} times`;
  }, [count]);
  // ^^^^^^ moved to hook

  return { count, handleIncrement };
}

const App = () => {
  const {count, handleIncrement} => useCounter();

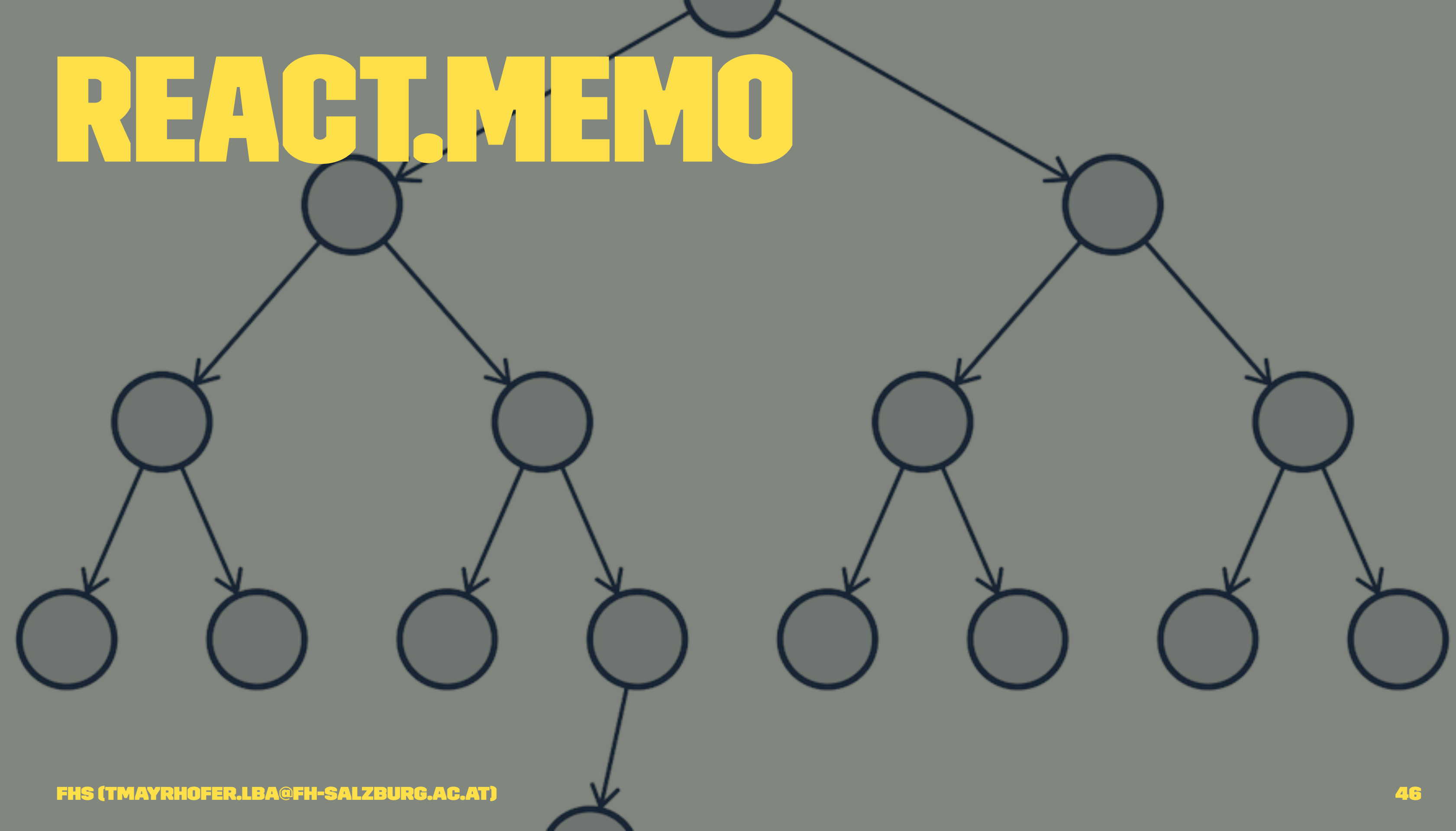
  return (
    <div>
      <div>{count}</div>
      <button onClick={handleIncrement}>Increment by 1</button>
    </div>
  );
}
```

REACT.MEMO

“Memoizing a function makes it faster by trading space for time. It does this by caching the return values of the function in a table.”⁷

⁷ <https://metacpan.org/pod/Memoize>

REACT.MEMO



REACT.MEMO

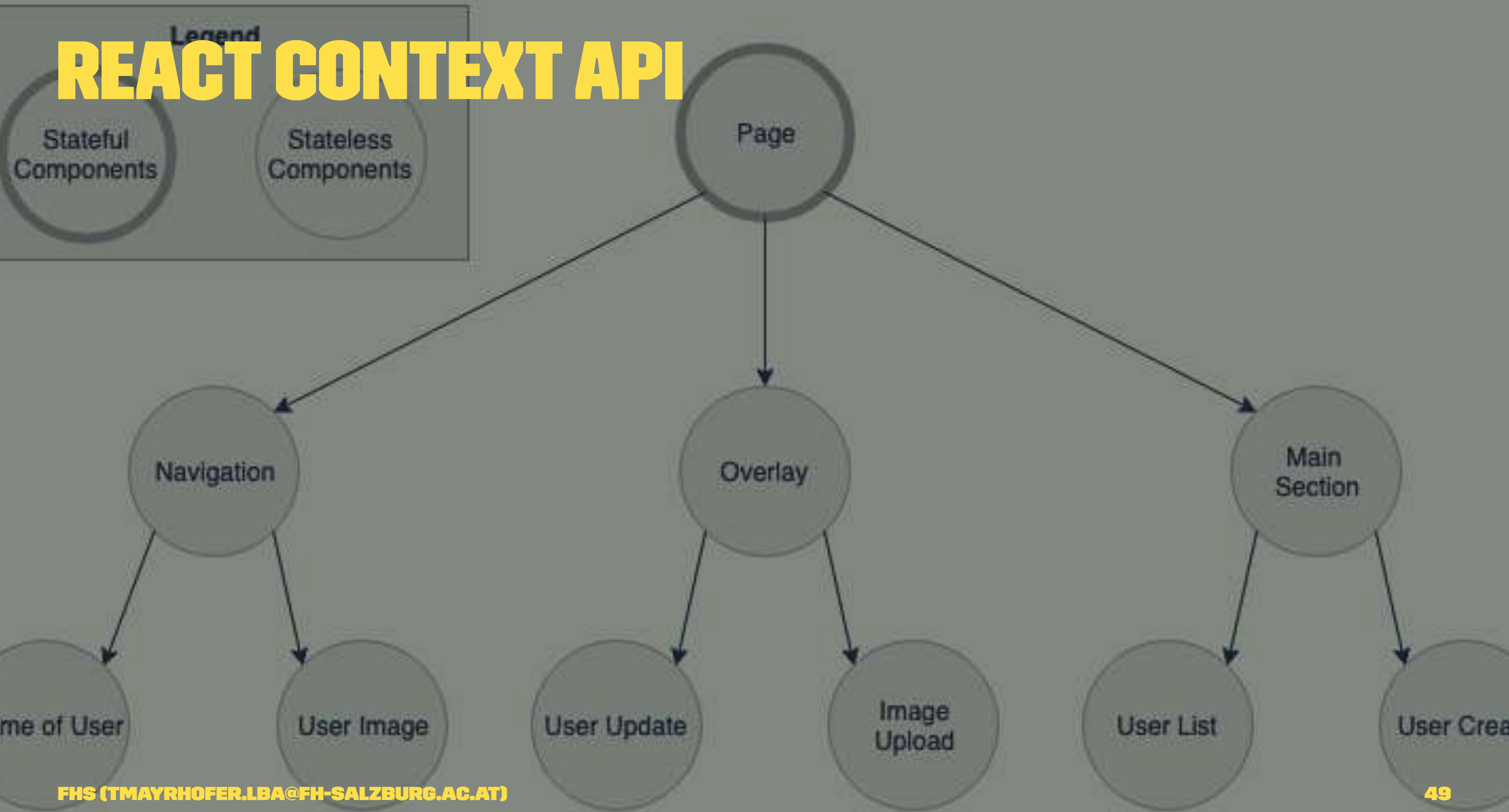
- » Caches the rendered component
- » Only rerenders when one of the props changes
 - » shallow comparison

```
const MyComponent = React.memo(function MyComponent(props) {  
  /* render using props */  
});
```

REACT CONTEXT API

- » Available since the beginning of React
- » Prevent "prop drilling"

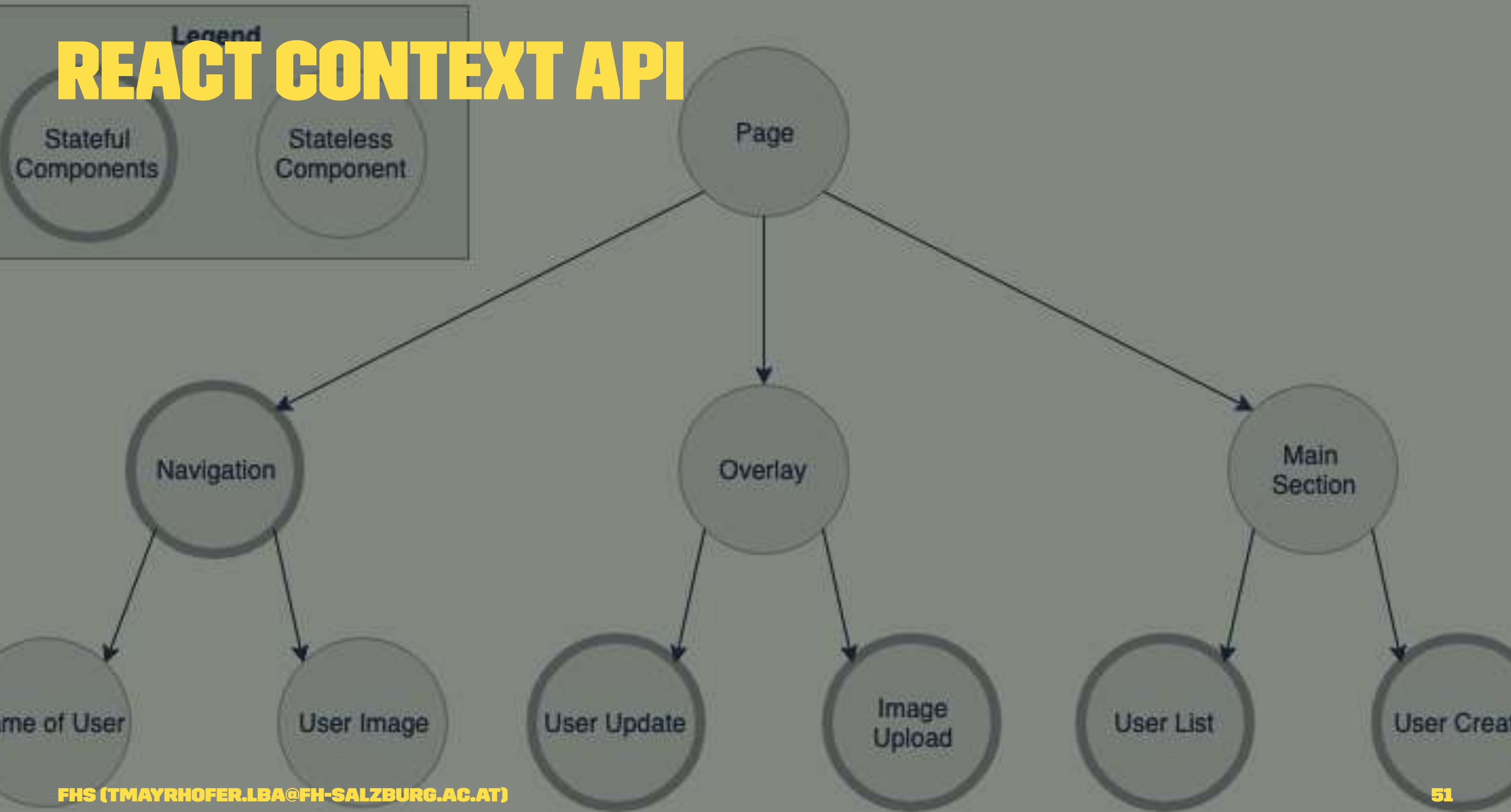
REACT CONTEXT API



REACT CONTEXT API

```
▼ <View pointerEvents="box-none" style={281}>
  ▼ <div className="css-1dbjc4n r-13awgt0 r-12vffkv">
    ▼ <View key="1" pointerEvents="box-none" style={281}>
      ▼ <r>
        ▼ <Context.Consumer>
          ▼ <Context.Provider>
            ▼ <Connect(t)>
              ▼ <t language="de" loggedInUserId="253431163">
                ▼ <t>
                  ▼ <Router.Consumer.Provider>
                    ▼ <withRouter(n)>
                      ▼ <t>
                        ▼ <Router.Consumer.Consumer>
                          ▼ <Router.Consumer.Provider>
                            ▼ <n>
                              ▼ <t>
                                ▼ <Router.Consumer.Consumer>
                                  ▼ <t>
                                    ▼ <Router.Consumer.Consumer>
                                      ▼ <Router.Consumer.Provider>
                                        ▼ <Unknown>
                                          ▼ <t>
                                            ▼ <withRouter(t)>
                                              ▼ <t>
                                                ▼ <Router.Consumer.Consumer>
                                                  ▼ <Router.Consumer.Provider>
                                                    ▼ <t>
                                                      ▼ <Connect(t)>
                                                        ▼ <t scale="normal">
                                                          ▼ <t>
                                                            ▼ <t showReload={true}>
                                                                ► <SideEffect(t) title="Twitter">...</SideEffect(t)>
                                                                ► <withRouter(Connect(t))>...</withRouter(Connect(t))>
                                                                ► <t zIndex={1}>...</t>
                                                            ▼ <View>
                                                                ▼ <div className="css-1dbjc4n r-1pi2tsx r-sa2ff0 r-13qz1uu r-417010">
                                                                    ► <withRouter(Connect(i))>...</withRouter(Connect(i))>
                                                                    ▼ <@twitter/Responsive>
                                                                        ▼ <View accessibilityRole="main" style={245}>
                                                                            ▼ <main role="main" className="css-1dbjc4n r-16y2uox r-1wbh5a2">
```

REACT CONTEXT API



CREATING A CONTEXT

```
const DEFAULT_VALUE = 1
const MyContext = React.createContext(DEFAULT_VALUE)
```

```
const RootComponent = () => {
  return (
    <MyContext.Provider value={2}>
      <ANestedComponent />
    </MyContext.Provider>
  )
}
```

```
const ANestedComponent = () => {
  const value = useContext(MyContext)
  return (
    <h1>The value from context is {value}</h1>
  )
}
```

PITFALLS 1

» fine granular context



PITFALLS/TIPS

- » Prefer passing props down to components
 - » prefer explicit (pass down) vs implicit (context)
- » only use when multiple components need to access same data
 - » if possible pass data down
- » don't overuse

OTHER HOOKS

» API Reference

» `useReducer`

» `useCallback`

» `useMemo`

» `useRef`

» `useImperativeHandle`

» `useLayoutEffect`

TASK ADVANCED HOOKS TASK

- » build a clock component
 - » component displays current time in seconds
 - » automatically updates itself
 - » remove `setInterval` when component unmounts
- » You'll need
 - » `useEffect`, `useState`
 - » `setInterval` or `setTimeout`

FEEDBACK

» Questions: tmayrhofer.lba@fh-salzburg.ac.at

» <https://de.surveymonkey.com/r/8TW92LL>