

REACT ADVANCED TOPICS

ABOUT/CONTACT

- » Thomas Mayrhofer (@webpapaya)
- » E-Mail: tmayrhofer.lba@fh-salzburg.ac.at

ROADMAP

- » Performance in Tips
- » Portals
- » Refs
- » Error Boundaries
- » Code Splitting

REACT PERFORMANCE TIPS

REACT PERFORMANCE TIPS

USE THE PRODUCTION BUILD

- » react adds development checks/warnings
- » in the production build warnings are stripped
- » with CRA⁶ run npm run build

⁶ Create React App (<https://create-react-app.dev/>)

REACT PERFORMANCE TIPS

IMMUTABILITY

“An immutable data structure is an object that doesn't allow us to change its value. (Remo H. Jansen)”

REACT PERFORMANCE TIPS

IMMUTABLE OBJECTS IN JS

```
const immutableObject = Object.freeze({ test: 1 })
immutableObject.test = 10
console.log(immutableObject) // => { test: 1 }
```

REACT PERFORMANCE TIPS

CHANGING AN IMMUTABLE VALUE

```
const immutableObject = Object.freeze({ a: 1, b: 2 })
const updatedObject = Object.freeze({ ...immutableObject, a: 2 })
console.log(updatedObject) // => { a: 2, b: 2 }
```

REACT PERFORMANCE TIPS

UNFREEZE AN OBJECT

```
const immutableObject = Object.freeze({ test: 1 })
const unfrozenCopy = { ...immutableObject }
```

REACT PERFORMANCE TIPS

WHY IMMUTABILITY

- » race conditions impossible
- » state of the application is easier to reason about
- » easier to test
- » easier to compare
- » shallow compare enough

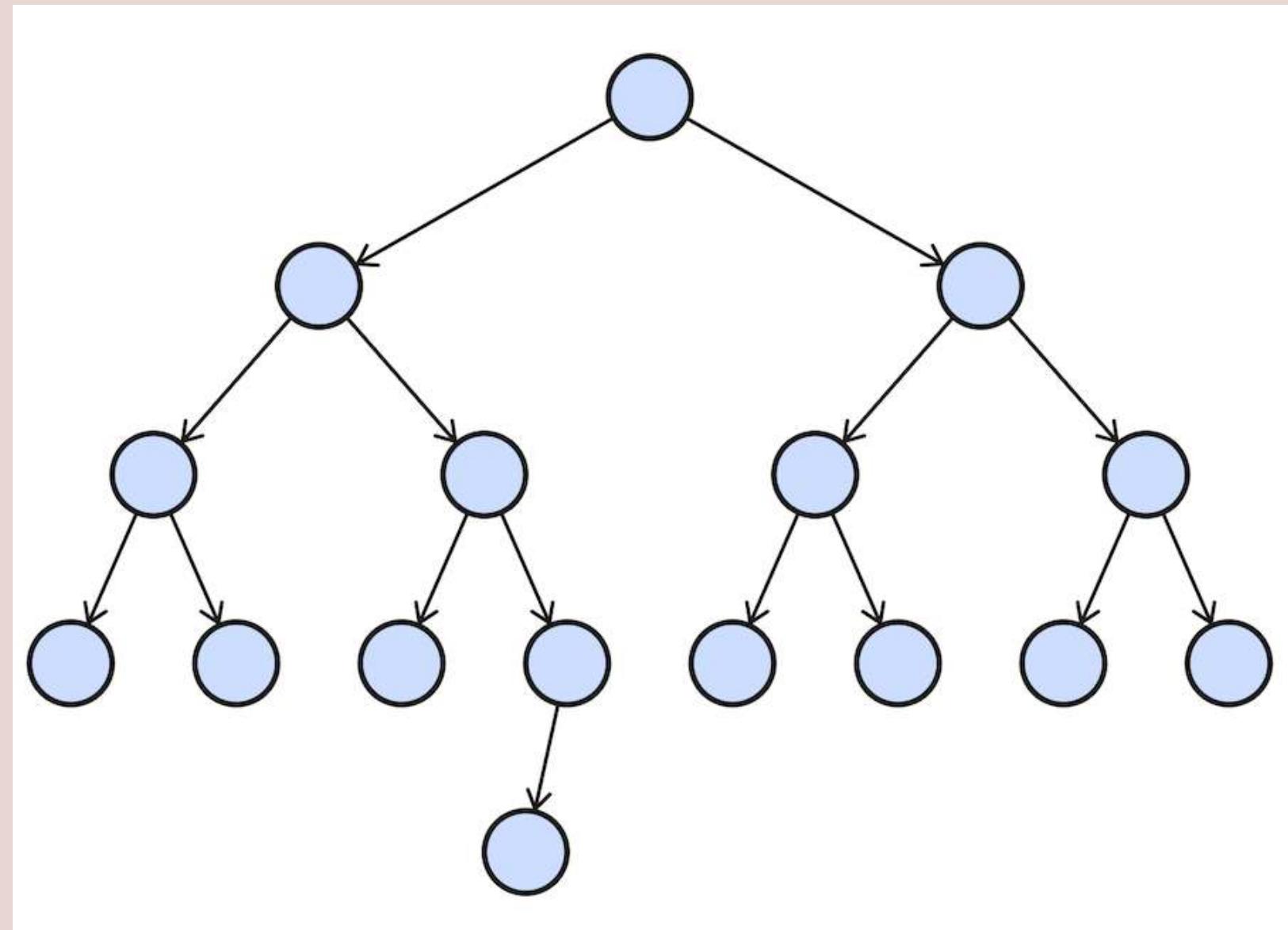
REACT PERFORMANCE TIPS

MEMOIZATION

“`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. (<https://metacpan.org/pod/Memoize>)”

REACT PERFORMANCE TIPS

REACT COMPONENT TREE



REACT PERFORMANCE TIPS

REACT.MEMO

```
import React from 'react'

const MyComponent = React.memo(({ users }) => {
//                                     ^^^^^^
// memoize the component, all sub-components will
// be rerendered initially and only when the pointer
// to the user array changes

  return (
    <section>
      <Users users={users}>
    </section>
  )
})
```

REACT PERFORMANCE TIPS

FILTERING LARGE LISTS

```
import React from 'react'

const MyComponent = ({ users }) => {
  const activeUsers = users.map(user => !user.inactive)

  return (
    <Users users={activeUsers} />
  )
}
```

REACT PERFORMANCE TIPS

FILTERING LARGE LISTS (MEMOIZATION)

```
import React, useMemo from 'react'

const MyComponent = ({ users }) => {
  const activeUsers = useMemo(
    // ^^^^^^
    // add the useMemo hook
    () => users.map(user => !user.inactive),
    // ^
    // define a callback so react can decide when to filter
    [users]
    // ^^^^
    // define the "dependencies"
    // = the filter function should be executed
    // when users array changes

  return (
    <Users users={activeUsers} />
  )
}
```

REACT PERFORMANCE TIPS

ARRAYS/OBJECTS AS DEFAULT VALUES

```
import React, useMemo from 'react'

const MyComponent = ({ users = [] }) => {
  const activeUsers = useMemo(
    () => users.map(user => !user.inactive),
    [users]
  )

  return (
    <Users users={activeUsers} />
  )
}
```

REACT PERFORMANCE TIPS

ARRAYS/OBJECTS AS DEFAULT VALUES

```
import React, useMemo from 'react'

const MyComponent = ({ users = [] }) => {
// ^^^
// every time undefined is passed to the component
// a new array will be created.

  const activeUsers = useMemo(
    () => users.map(user => !user.inactive),
    [users])
// ^^^^^^
// users change with each rerender
// => empty array will be filtered all the time

  return (
    <Users users={activeUsers} />
  )
}
```

REACT PERFORMANCE TIPS

ARRAYS/OBJECTS AS DEFAULT VALUES

```
import React, useMemo from 'react'

const DEFAULT_USERS = []
// ^^^^^^^^^^^^^^^^^^^^^^
// this array gets instantiated once when the code
// is parsed initially. The same array will be reused
// with each rerender.

const MyComponent = ({ users = DEFAULT_USERS }) => {
// ^^^^^^^^^^
// use the global array

  const activeUsers = useMemo(
    () => users.map((user) => !user.inactive),
    [users]
  )

  return (
    <Users users={activeUsers} />
  )
}
```

REACT PERFORMANCE TIPS

CALLBACKS

» Do you find anything problematic here?

```
import React, useMemo from 'react'

const Users = React.memo(({ users, onUserClick }) => {
  // ...
})

const MyComponent = ({ users = DEFAULT_USERS }) => {
  return (
    <Users
      users={activeUsers}
      onUserClick={() => console.log('test')}
    />
  )
}
```

REACT PERFORMANCE TIPS

CALLBACKS

```
import React, useMemo from 'react'

const Users = React.memo(({ users, onUserClick }) => {
  // ...
})

const MyComponent = ({ users = DEFAULT_USERS }) => {
  return (
    <Users
      users={activeUsers}
      onUserClick={() => console.log('test')}
    // ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    // Each time the component is rerendered a new function
    // will be created. This makes React.memo think that the
    // prop changed and will trigger a rerender of the Users
    // component.
    />
  )
}
```

REACT PERFORMANCE TIPS

CALLBACKS

```
import React, { useCallback } from 'react'

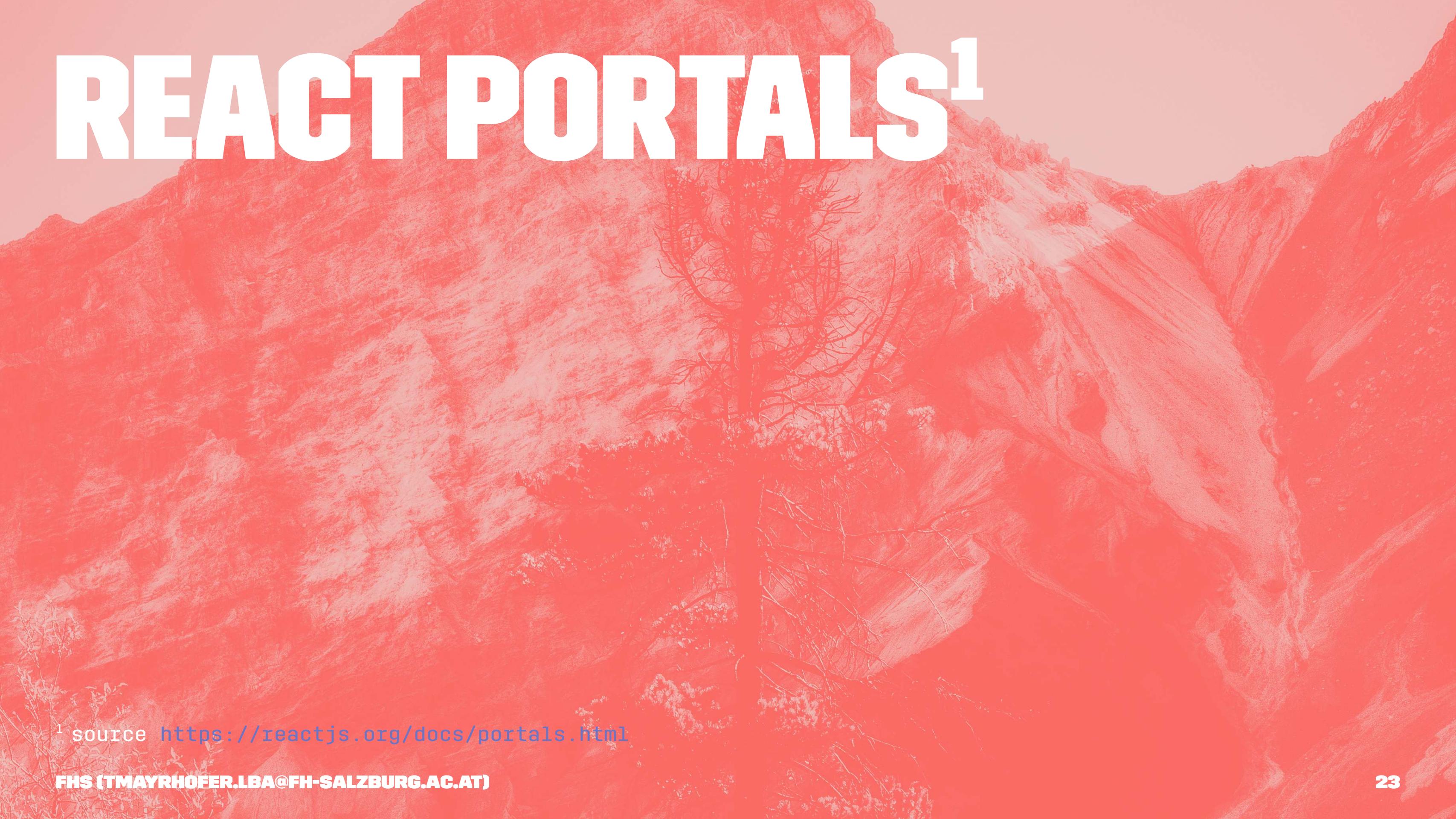
const Users = React.memo(({ users, onUserClick }) => {
  // ...
})

const MyComponent = ({ users = DEFAULT_USERS }) => {
  const onUserClick = useCallback(
    //                                     ^^^^^^^^^^
    // cache/memoize the callback function
    () => console.log('test'),
    []
  //^^
  // dependencies (when should the function be recreated)
  )
  return (
    <Users users={activeUsers} onUserClick={onUserClick} />
  )
}
```

TASK (15 MIN)

- » Go to your application
- » Create a new branch
- » Find performance optimizations
 - » Default Arrays
 - » Arrays which could be memoized

REACT PORTALS¹

A landscape photograph showing a range of mountains with distinct horizontal sedimentary rock layers. In the foreground, a large, leafless tree stands prominently on a rocky outcrop. The sky is overcast with soft, grey clouds.

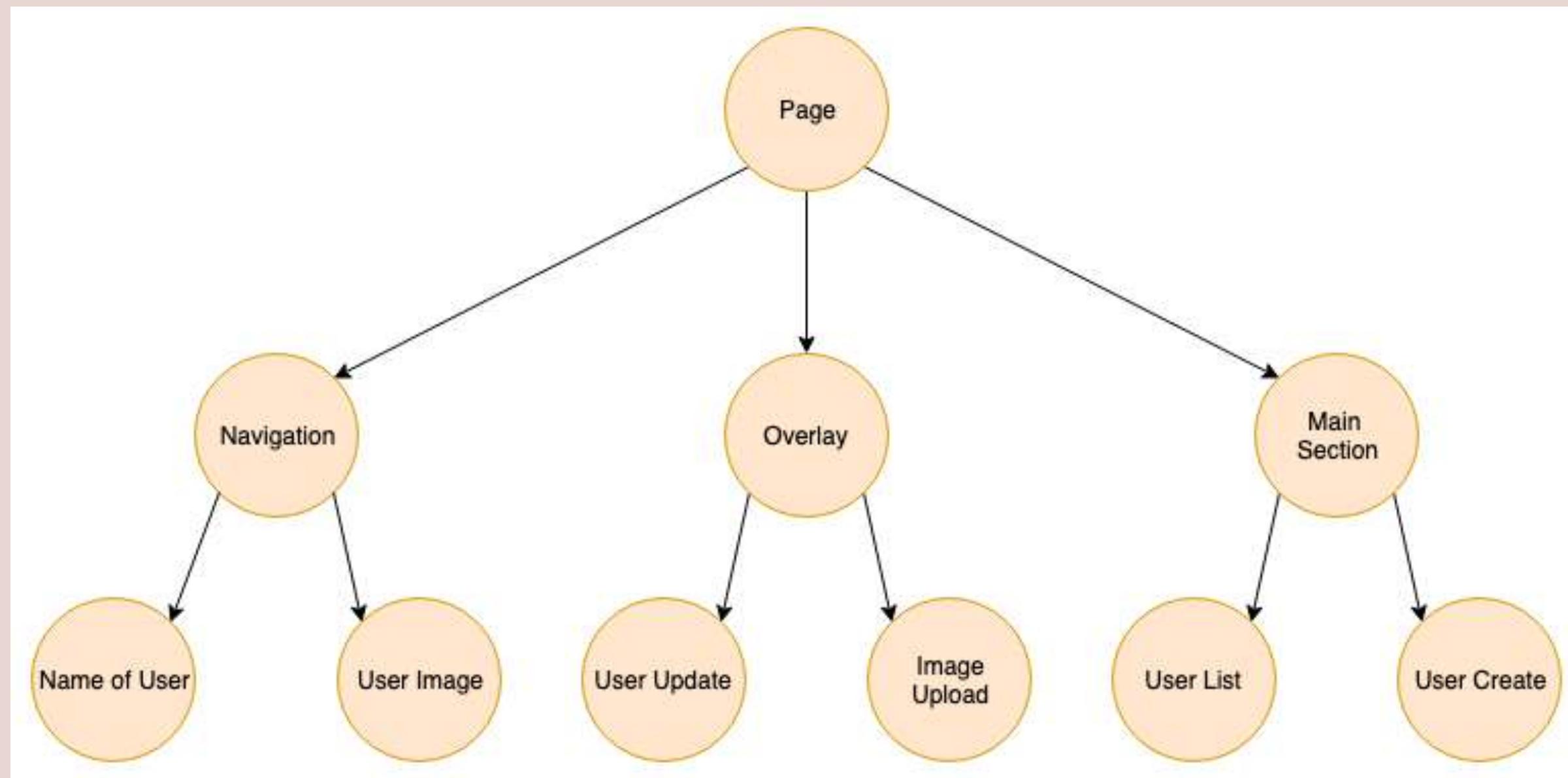
¹ source <https://reactjs.org/docs/portals.html>

REACT PORTALS¹

“Portals provide a first-class way to render children into a DOM node that exists outside the DOM hierarchy of the parent component.”

¹ source <https://reactjs.org/docs/portals.html>

REACT PORTALS¹



¹ source <https://reactjs.org/docs/portals.html>

REACT PORTALS¹

- » Components are rendered inside nearest parent DOM

```
const MyComponent = ({ children }) => {  
  return (  
    <div>  
      {children}  
    </div>  
  );  
}
```

¹ source <https://reactjs.org/docs/portals.html>

REACT PORTALS¹

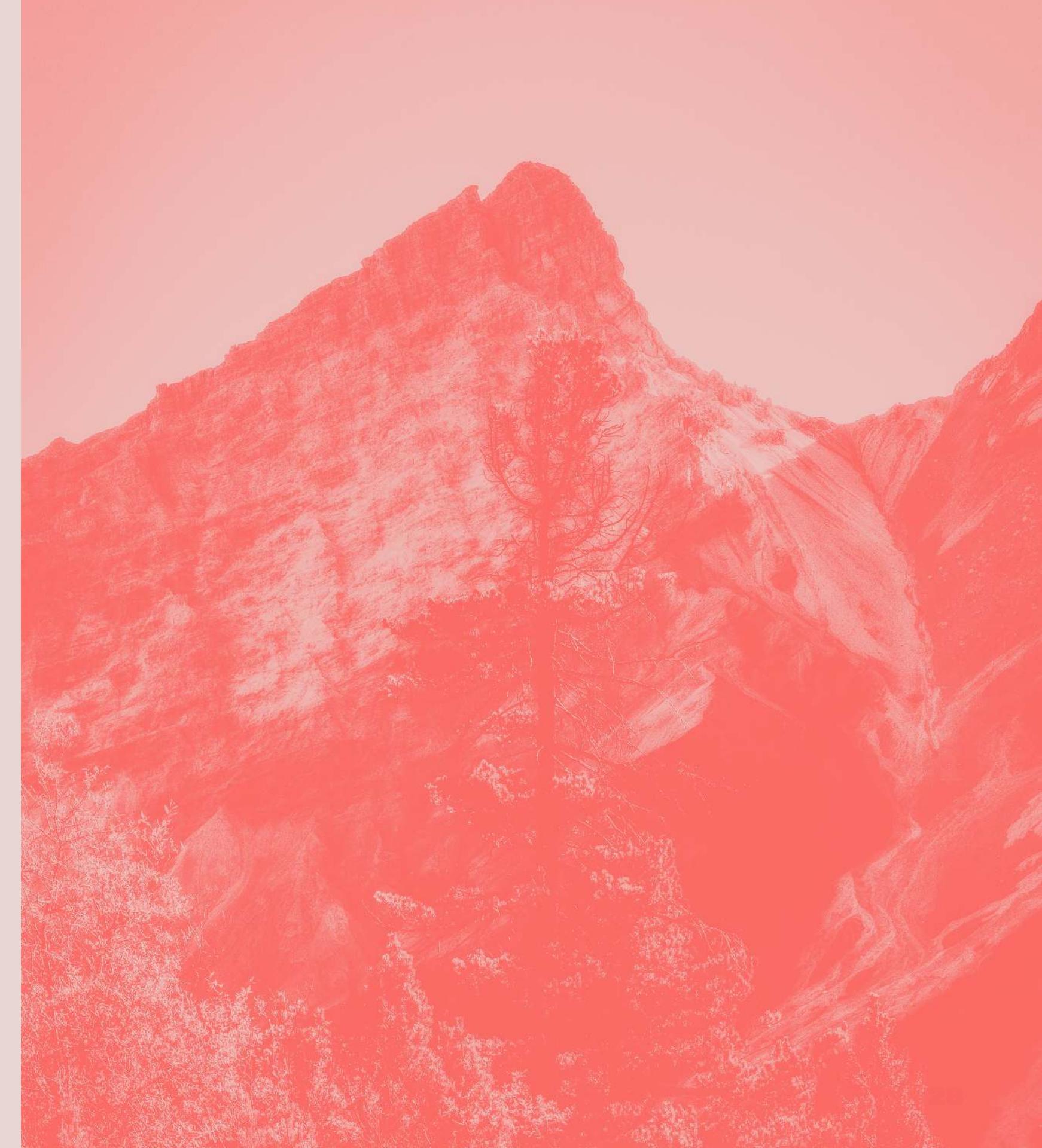
- » Sometimes components need to render outside of DOM nodes
- » eg. modals

```
const MyComponent = ({ children }) => {  
  return ReactDOM.createPortal(  
    children,  
    // specify elements to be rendered  
  
    document.getElementById("myComponentRoot")  
    // children will be rendered inside #myComponentRoot  
  );  
}
```

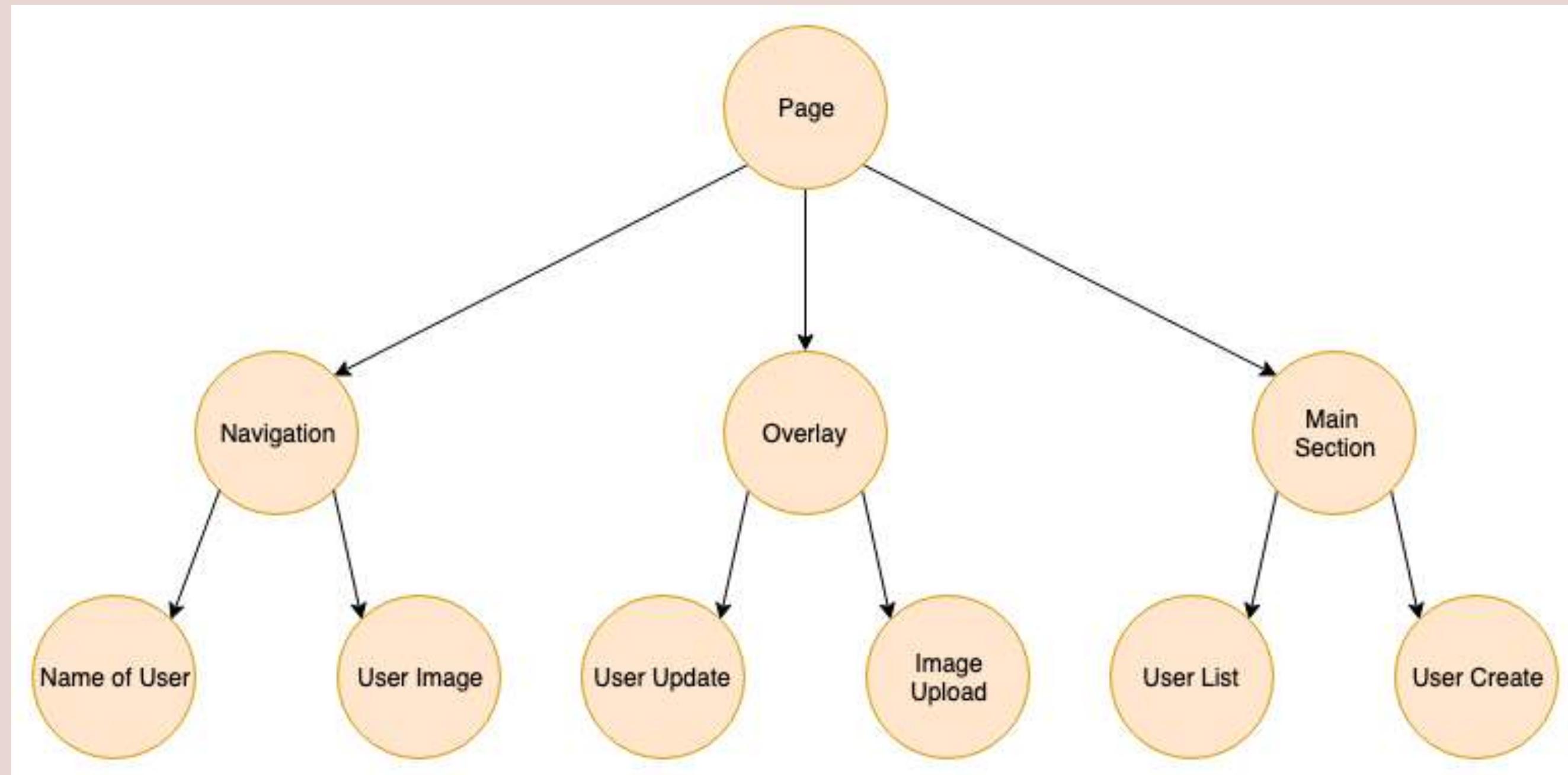
¹ source <https://reactjs.org/docs/portals.html>

REFS

“Refs provide a way to access DOM nodes or React elements created in the render method.”



REFS



REFS²

- » props are used to interact with child components
- » to modify children rerender with different props
- » might be triggered via `useState/useState`
- » refs are used to access DOM elements directly
- » setting focus
- » trigger imperative animations

² source <https://reactjs.org/docs/refs-and-the-dom.html>

REFS²

```
const MyInputComponent = ({ autofocus }) => {
  const inputRef = useRef(null);
  //           ^^^^^^^^^^
  // specify a reference with an initial value of null

  useEffect(() => {
    inputRef.current?.focus();
    //           ^^^^^^
    // current might be <input /> or null
    // when <input /> then focus input
  }, [inputRef.current])

  return <input ref={inputRef} />
  //           ^^^^^^^^^^
  // specify to which element the ref should be bound
}
```

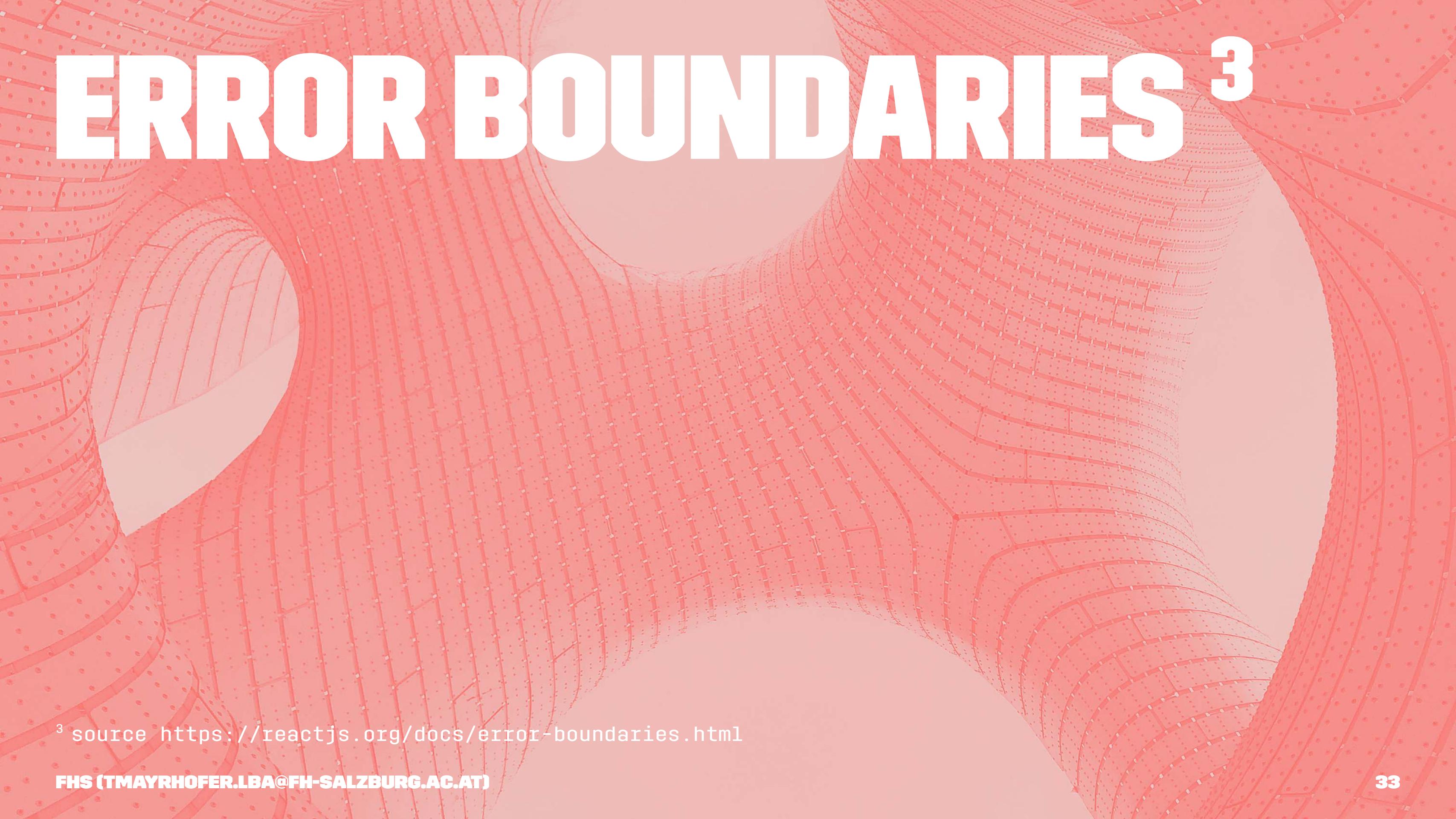
² source <https://reactjs.org/docs/refs-and-the-dom.html>

REFS²

- » Don't Overuse Refs
- » prefer using props and lift up state

² source <https://reactjs.org/docs/refs-and-the-dom.html>

ERROR BOUNDARIES³



³ source <https://reactjs.org/docs/error-boundaries.html>

ERROR BOUNDARIES³

- » an error in a component might break the whole app
- » react mixes up state
- » reloading page is the only way to recover
- » bad UX
- » Error boundaries handle errors of components

³ source <https://reactjs.org/docs/error-boundaries.html>

ERROR BOUNDARIES³

- » static getDerivedStateFromError
 - » invoked after an error in child has been thrown
 - » should return new component state
- » componentDidCatch
 - » invoked after an error in child has been thrown
 - » used to trigger side effects (eg. server logging)

³ source <https://reactjs.org/docs/error-boundaries.html>

ERROR BOUNDARIES³

```
class ErrorBoundary extends React.Component {
  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;
  }
}
```

³ source <https://reactjs.org/docs/error-boundaries.html>

ERROR BOUNDARIES³

USING ERROR BOUNDARIES

- » using boundaries is up to the developer
- » place boundaries around use-cases⁴ eg:
 - » sign-up form
 - » create money-transactions
 - » money-transaction list

³ source <https://reactjs.org/docs/error-boundaries.html>

⁴ personal opinion

CODE SPLITTING

CODE SPLITTING BUNDLING

“combine multiple JS modules into a single bundle”

- » when apps grow the bundle will grow
- » large libraries are added to bundle
- » increases download/parse time

CODE SPLITTING⁵

“split bundle into multiple bundles which can be loaded lazily during runtime”

⁵ source <https://reactjs.org/docs/code-splitting.html>

CODE SPLITTING⁵

DYNAMIC IMPORTS

```
const add = async (a, b) => {
  const math = await import("./math")
  //
  // load "./math" module only when `add` is executed
  // for the first time. Bundlers will split the bundle
  // automatically. (2 files will be generated)

  return math.add(a, b)
};
```

⁵ source <https://reactjs.org/docs/code-splitting.html>

CODE SPLITTING⁵

React.lazy

“The React.lazy function lets you render a dynamic import as a regular component.”

```
const OtherComponent = React.lazy(() => import('./OtherComponent'));  
//  
// dynamically import `./OtherComponent` when the component is rendered  
// for the first time.  
// `./OtherComponent` needs to be exported as default
```

⁵ source <https://reactjs.org/docs/code-splitting.html>

CODE SPLITTING⁵

React.Suspense

- » importing modules dynamically is an `async` operation
- » a loading screen needs to be displayed

“React.Suspense lets components “wait” for something before rendering.”

⁵ source <https://reactjs.org/docs/code-splitting.html>

CODE SPLITTING⁵

React.Suspense

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

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

function MyComponent() {
  return (
    <Suspense fallback=<div>Loading...</div>>
      <OtherComponent />
    </Suspense>
  );
}

}
```

⁵ source <https://reactjs.org/docs/code-splitting.html>

FEEDBACK

- » Questions: tmayrhofer.lba@fh-salzburg.ac.at
- » <https://s.surveyplanet.com/xlibwm85>