Applikationsentwicklung mit JavaScript & HTML5

React

Berner Fachhochschule
CAS Applikationsentwicklung
mit JavaScript & HTML5

Jonas Bandi
IvoryCode GmbH
jonas.bandi@ivorycode.com

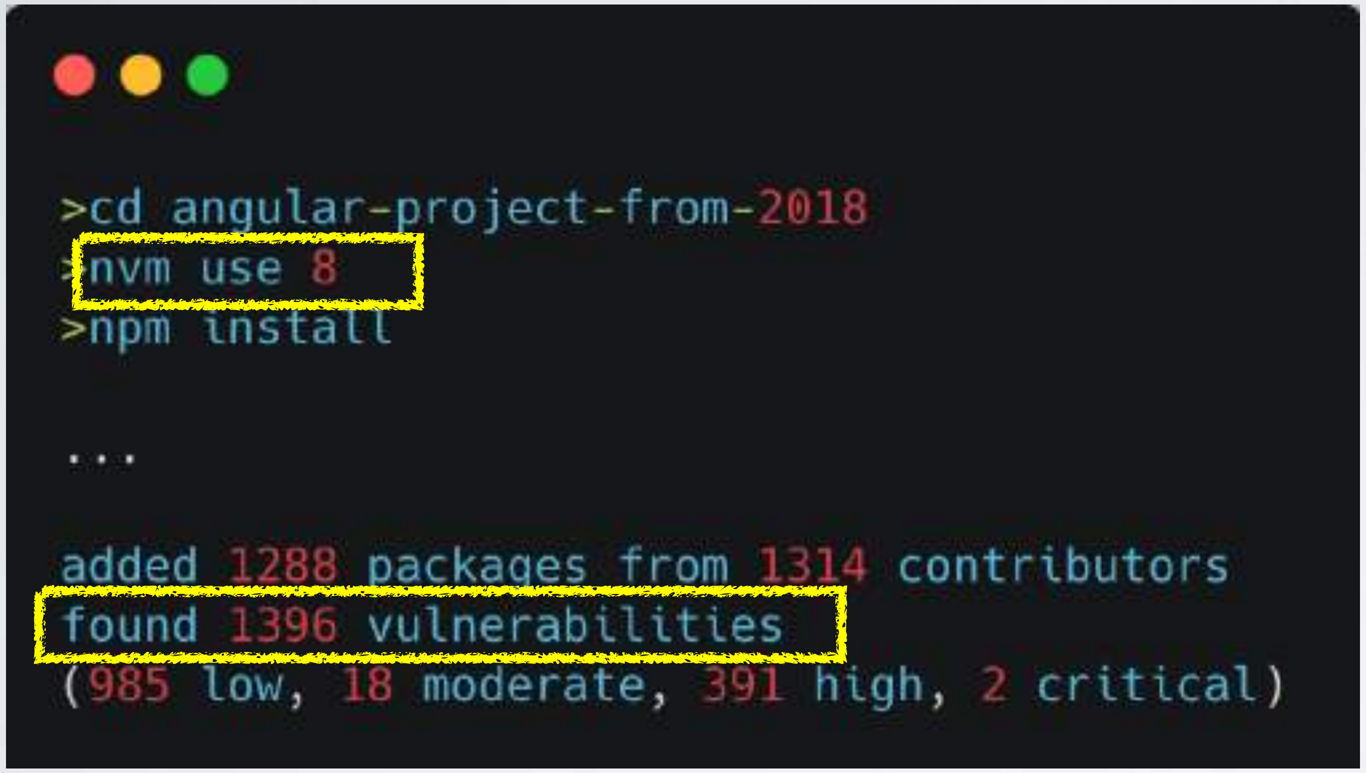
Recap Immutable Statemanagement

- `structuredClone` creates a deep copy, which is not recommended. Render Optimizations with *memoizaition* are not possible with deep cloning ...
  React docs don't mention `structuredClone` but just the "spread" copying:
  https://react.dev/learn/updating-objects-in-state

- mutating state is not recommended:
  https://react.dev/learn/updating-objects-in-state#why-is-mutating-state-not-recommended-in-react
  - newer features of React like concurrent rendering / `startTransition` might fall back and render an previous version of the state ...
    recommendation: *treat state as a imutable snapshot*
    https://react.dev/learn/state-as-a-snapshot

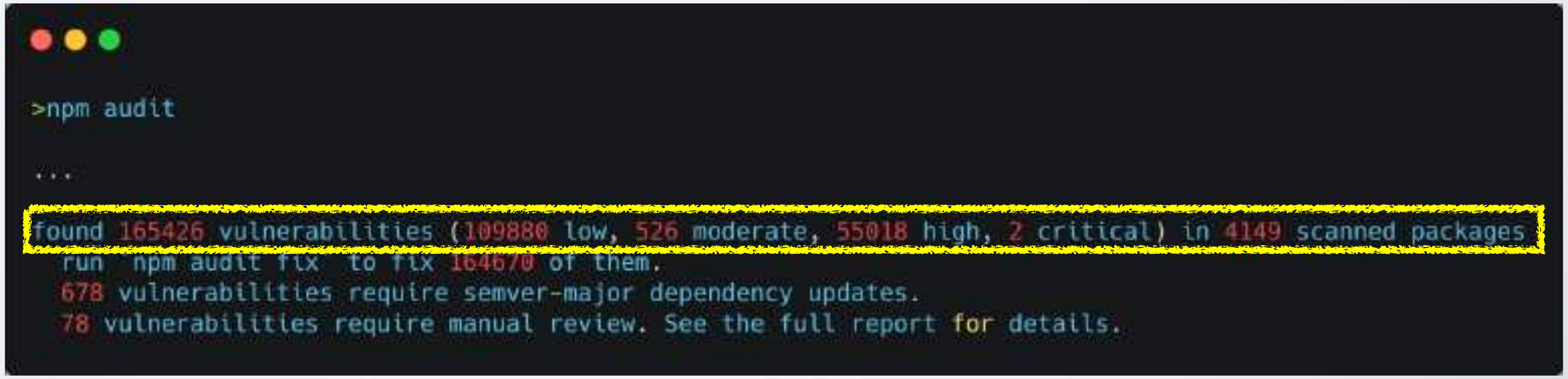
Side Topic: Maintenance

A "small" Angular project from 2018:



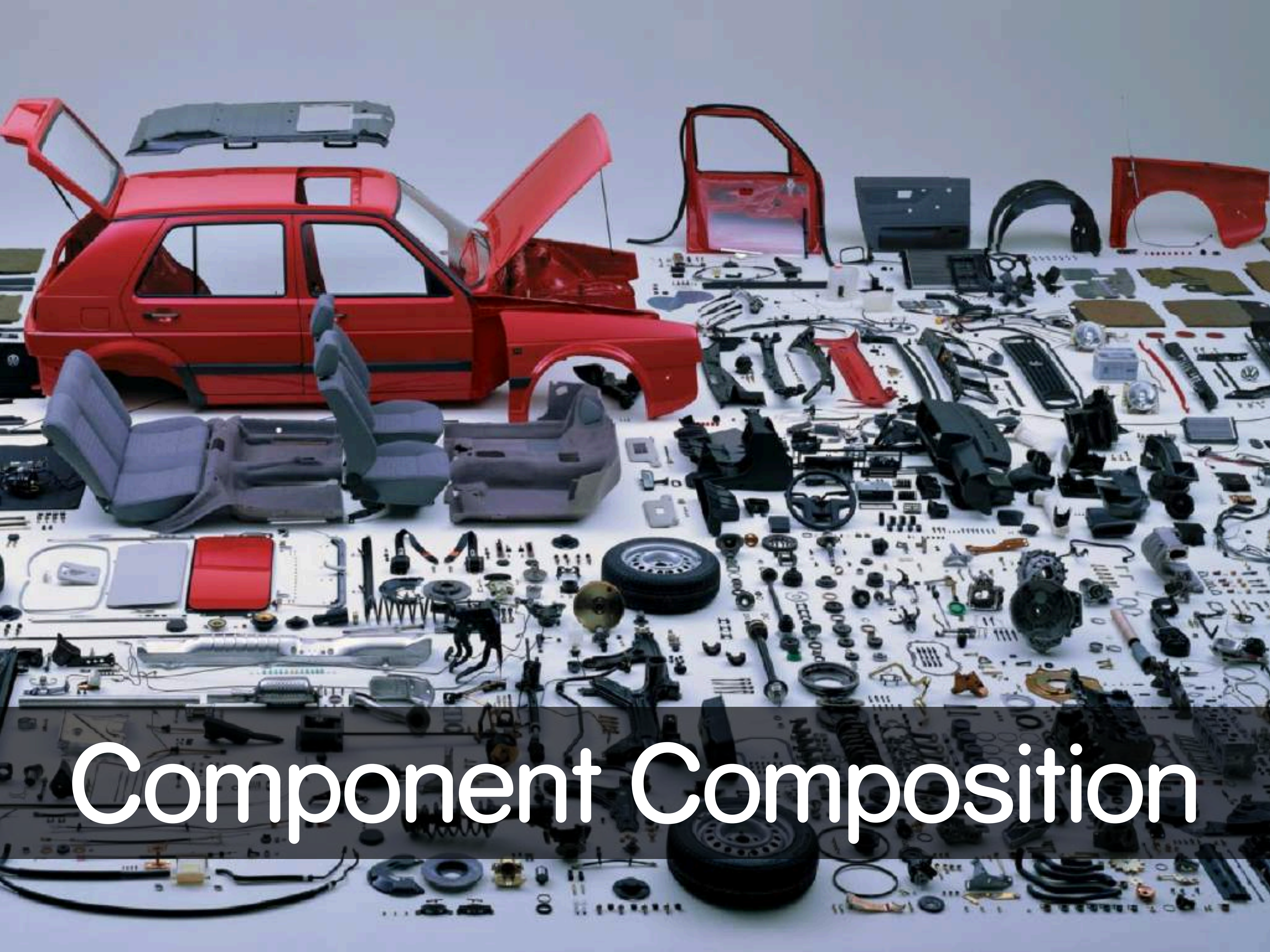
```
>cd angular-project-from-2018
>nvm use 8
>npm install

...

added 1288 packages from 1314 contributors
found 1396 vulnerabilities
(985 low, 18 moderate, 391 high, 2 critical)
```

Real-World example:
In-house React component library, several years untouched:



```
>npm audit

...

found 165426 vulnerabilities (109880 low, 526 moderate, 55018 high, 2 critical) in 4149 scanned packages
  run `npm audit fix` to fix 164670 of them.
  678 vulnerabilities require semver-major dependency updates.
  78 vulnerabilities require manual review. See the full report for details.
```

However `npm audit` vulnerabilities might not be a good metric: https://overreacted.io/npm-audit-broken-by-design/



Component Composition

Container vs. Presentation Components

"Separtion of Concerns"

Application should be decomposed in container- and presentation components:

Container	**Presentation**
Little to no markup	Mostly markup
Pass data and actions down	Receive data & actions via props
typically stateful / manage state	mostly stateless
	better reusability

aka: Smart- vs. Dumb Components

https://medium.com/@dan_abramov/smart-and-dumb-components-7ca2f9a7c7d0

Separation of Concerns

Separation of concerns is not equal to separation of file types!

Keep things together that change together.

You can split a component into a controller and a view:

```
import {View} from './View';

export function Controller {
  ... // state & behavior
  return (
    <View data={...}
          onEvent={...} />
  );
}
```

Controller.js

```
export function View({data, onEvent}){
  return (
    <div>
      {data.message}
      <button onClick={()=>onEvent()}>
        Go!
      </button>
    </div>
  );
}
```

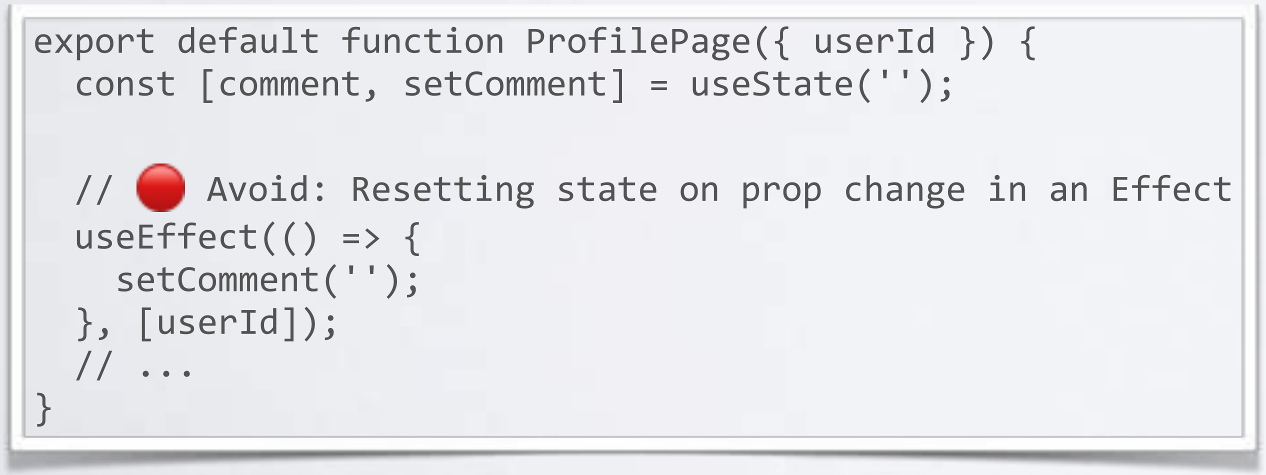
View.js

https://codesandbox.io/s/NxqMqyxlD
https://medium.com/styled-components/component-folder-pattern-ee42df37ec68

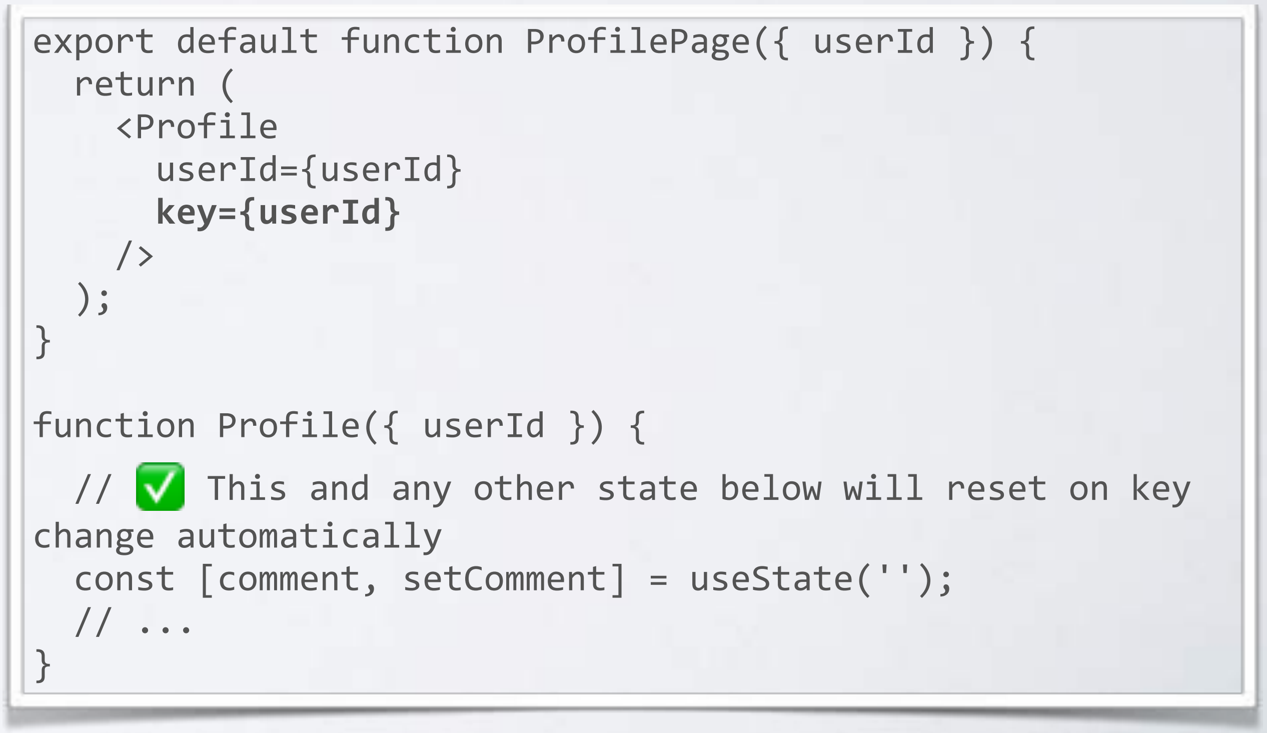
Controlling Component Lifecycle with a Key

React is preserving state in a component "instance".
By passing a **key** to a component, you can control the lifecycle of the "instance".

Example:



```
export default function ProfilePage({ userId }) {
  const [comment, setComment] = useState('');

  // 🔴 Avoid: Resetting state on prop change in an Effect
  useEffect(() => {
    setComment('');
  }, [userId]);
  // ...
}
```



```
export default function ProfilePage({ userId }) {
  return (
    <Profile
      userId={userId}
      key={userId}
    />
  );
}

function Profile({ userId }) {
  // ✅ This and any other state below will reset on key
change automatically
  const [comment, setComment] = useState('');
  // ...
}
```

https://react.dev/learn/you-might-not-need-an-effect#resetting-all-state-when-a-prop-changes

Composition with `props.children`

```
import type {ReactNode} from 'react';
type Props = {children: ReactNode}

function WrapperComponent(props: Props) {
    return (
      <div className="container">
        <hr/>
        {props.children}
        <hr/>
      </div>
    )
  }
}
```

```
function App() {
    return (
      <WrapperComponent>
        <ChildComponent>
      </WrapperComponent>
    );
}
```

TypeScript: There are several possibilities to type children. Popular choices are:

- `React.ReactNode`
- `JSX.Element`
- `React.PropsWithChildren`

https://www.totaltypescript.com/jsx-element-vs-react-reactnode

https://react.dev/learn/passing-props-to-a-component#passing-jsx-as-children

Components as props

```
import type {ReactNode, FunctionComponent} from 'react';
type Props = {leftComponent: ReactNode,
           middleComponent: ReactNode, rightComponent: FunctionComponent}

function Layout({leftComponent, middleComponent, rightComponent}: Props) {
  return (
    <div className="row">
      <div>{leftComponent}</div>
      <div>{middleComponent}</div>
      <div style={{display: 'flex', flexDirection: 'column'}}>
        {[1,2,3].map((i) => (
          <div key={i}>
            {React.createElement(rightComponent)}
          </div>)
        )}
      </div>
    </div>
  );
}
```

```
function App() {
  return (
    <Layout
      leftComponent={<Navigation/>}
      middleComponent={<MainContent/>}
      rightComponent={Advertisement}/>
  );
}
```

Passing component instance

passing component type

Higher Order Components

```
function withBackground(Component){
  return () => (
    <div className="container">
      <hr/>
        <Component/>
      <hr/>
    </div>
  )
}
```

Higher Order Component: takes a component as argument & returns a new component

```
function Content() {
    return (
      <div>Test!</div>
    )
}
const WrappedComponent = withBackground(Content);
```

applying the higher order component

`lazy()` and **`memo()`** are two examples of higher order components.

For TypeScript typing see: https://react-typescript-cheatsheet.netlify.app/docs/hoc/full_example/

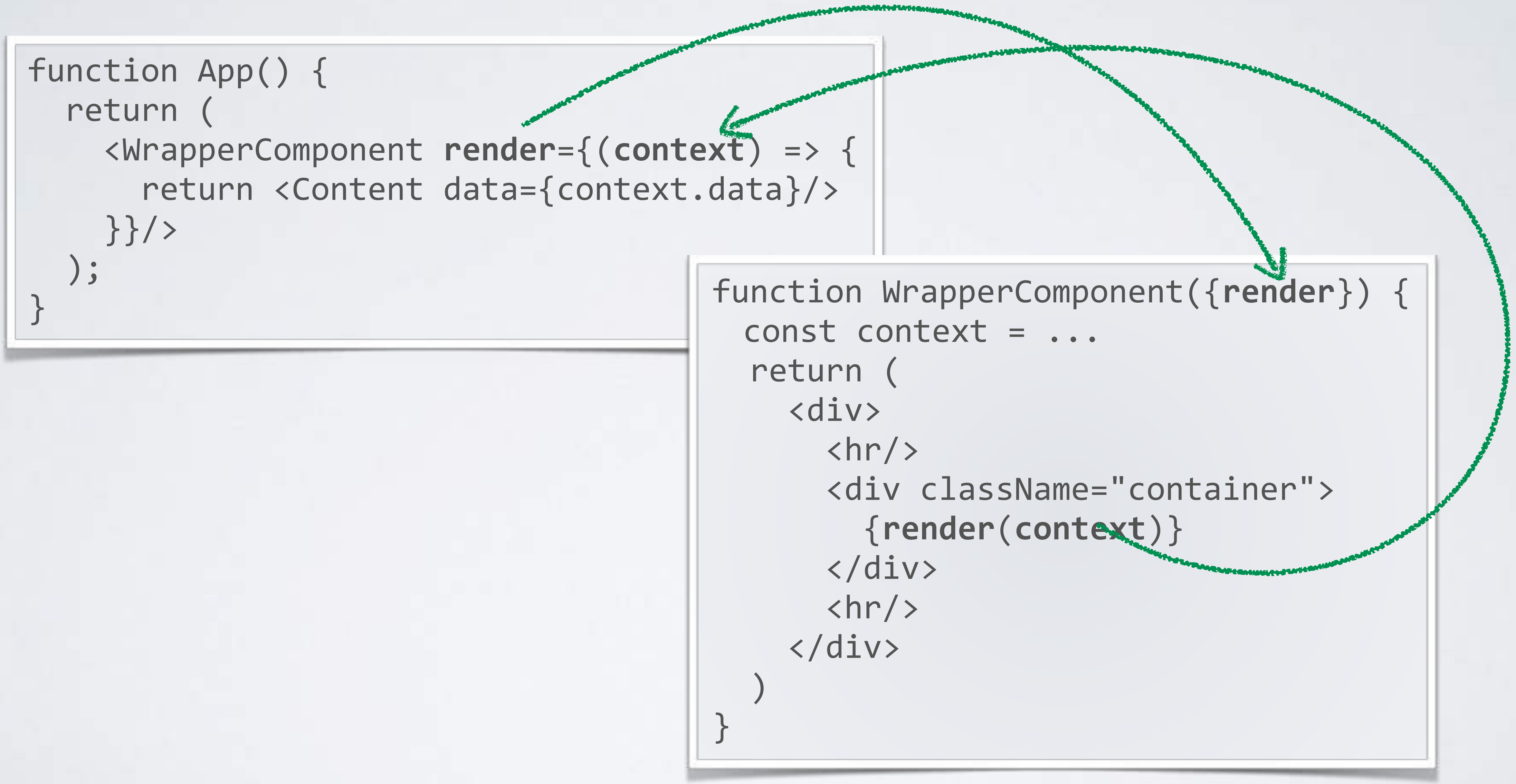
Note: HOCs were very popular before hooks. Hooks did replace many scenarios for HOCs!

Recompose: Higher-Order Component Library
https://github.com/acdlite/recompose (not maintained any more)

https://reactjs.org/docs/higher-order-components.html
https://www.robinwieruch.de/react-hooks-higher-order-components

Render Props

"render props" is a technique for sharing code beteween components.



```
function App() {
  return (
    <WrapperComponent render={(context) => {
      return <Content data={context.data}/>
    }}/>
  );
}
```

```
function WrapperComponent({render}) {
  const context = ...
  return (
    <div>
      <hr/>
      <div className="container">
        {render(context)}
      </div>
      <hr/>
    </div>
  )
}
```

Note: many use-cases for render props can be replaced with hooks in a more elegant way!

(but hooks can't render anything, cant' set values on a context and cant' implement error boundaries)

https://reactjs.org/docs/render-props.html

https://frontarm.com/james-k-nelson/hooks-vs-render-props/

Lazy Loading & Suspense

Lazy Loading of Components

React makes it very easy to load components on demand with **`React.lazy`** & **`<Suspense>`**

https://react.dev/reference/react/lazy#suspense-for-code-splitting

```
import React, {lazy, Suspense} from 'react';
const OtherComponent = lazy(() => import('./OtherComponent'));

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

`<Suspense>` will render the `fallback` if a contained component is not yet loaded.

Note: `React.lazy()` is built on top of dynamic `import()` specified in ECMAScript 2020:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/import

Suspense

https://react.dev/reference/react/Suspense

```
function App() {
    return (
        <>
            <Suspense fallback={<div>Loading...</div>}>
                <Content/>
            </Suspense>
        </>
    )
}
```

```
let loaded = false;
const promise = new Promise((resolve) => {
    setTimeout(() => {
        loaded = true;
        resolve(loaded);
    }, 1000);
});


function Content() {
    if (!loaded) {
        throw promise;
    }
    return <h1>Content</h1>;
}
```

Suspense

https://github.com/pmndrs/suspend-react

```
function App() {
    return (
        <>
            <Suspense fallback={<div>Loading...</div>}>
                <Content/>
            </Suspense>
        </>
    )
}
```

```
async function loadData() {
    const promise: Promise<string> = new Promise((resolve) => {
        setTimeout(() => {
            resolve("Test");
        }, 1000);
    });
    return promise;
}

function Content() {
    const data = suspend(loadData);
    return <h1>{data}</h1>;
}
```

Error Boundaries

Error Boundary

An Error Boundary is a component that can catch errors that happen during render or lifecycle methods in the components below them in the tree.

```
class **ErrorBoundary** extends React.Component {

  static getDerivedStateFromError(error) {
     // return a state update object with error data
     // next render can then use that error data
  }

  componentDidCatch(error, info) {
     // perform an action based on the error
  }

  render() {
   //
  }
}
```

"pure" function, no side-effects, just updating the state

side-effects like logging are executed here

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

Errors in render or during the component lifecycle, can't be handled with try/catch since React is calling these functions, based on declarative JSX.

https://react.dev/reference/react/Component#catching-rendering-errors-with-an-error-boundary

react-error-boundary

Small wrapper library providing a JSX element for an error boundary.
(so you don't need to write a class component)

```
npm install react-error-boundary
```

```
<ErrorBoundary
    onError={myErrorHandler}
    fallback={<CustomErrorComponent/>}>

  <ComponentThatMayError />

</ErrorBoundary>
```

https://github.com/bvaughn/react-error-boundary

React Error Handling

imperative vs. declarative code paths

```
export function Counter() {
  const [count, setCount] = useState(0);

  function increaseCount() {
   try {
    const val = maybeThrowError();
    return () => setCount((count) => count + val);
   } catch(e) { console.log('Errror!') }
  }

  const val = maybeThrowError();

  return (
    <div>
      <div>Display of Counter: {val}</div>
      <button onClick={increaseCount}>count is {count}</button>
    </div>
  );
}
```

event handling is imperative
=> handling with try-catch

rendering is declarative
=> handling with error-boundary

```
function App() {
  return (
    <div className="App">
        <ErrorBoundary fallback={<h1>Error ...</h1>}>
          <Counter />
        </ErrorBoundary>
    </div>
  );
}
```

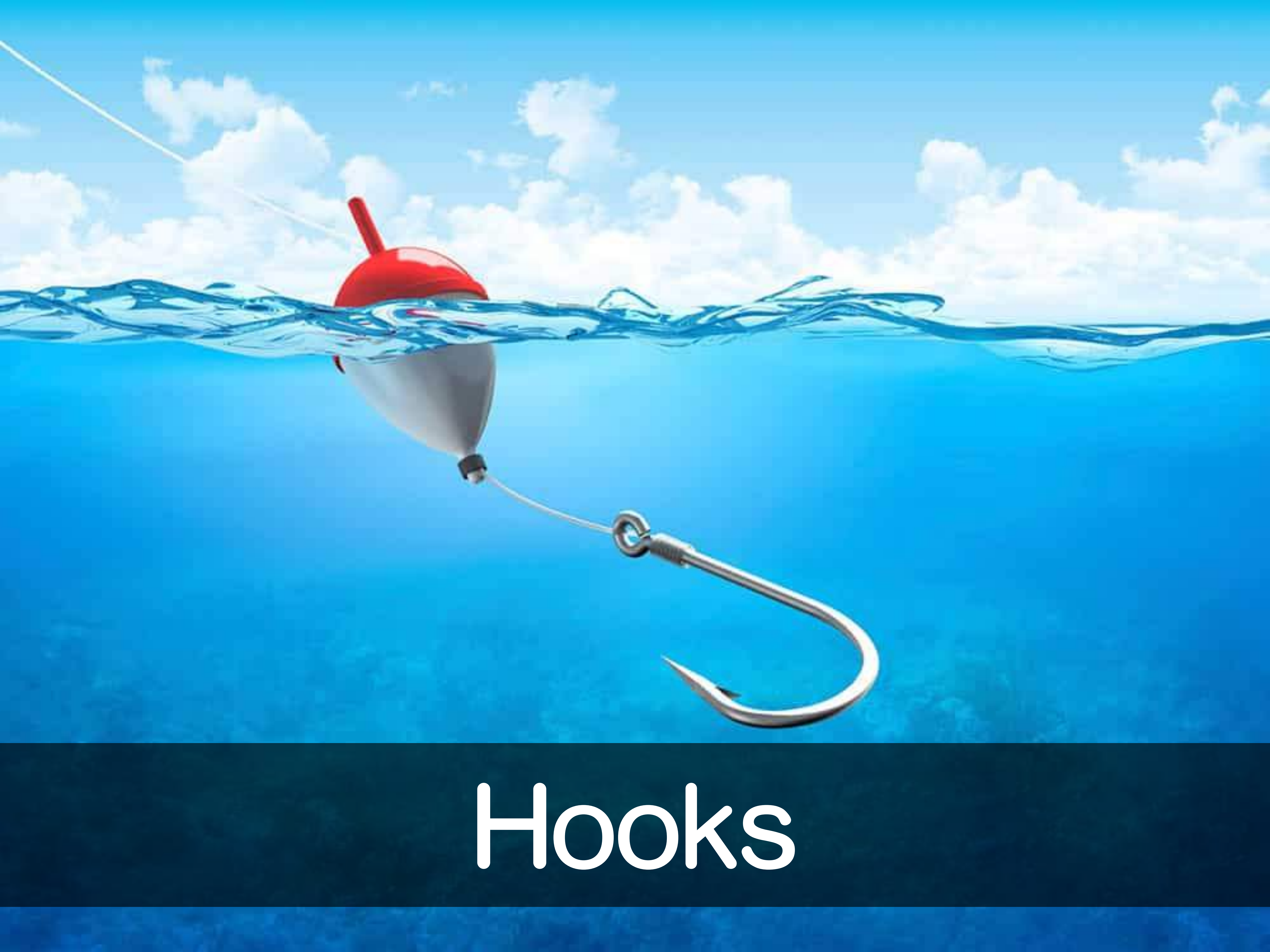
React Error Handling

bridging from imperative to declarative error handling

```
export function Counter() {
  const [count, setCount] = useState(0);
  const [error, setError] = useState(null);

  function increaseCount() {
    try {
     const val = maybeThrowError();
     return () => setCount((count) => count + val);
    } catch(e) { e => setError(e); }
  }

  if (error) throw error;

  return (
    <div>
      <div>Display of Counter: {val}</div>
      <button onClick={increaseCount()}>count is {count}</button>
    </div>
  );
}
```

EXERCISE

Exercise: Component Composition



Hooks

Advantages of Hooks

Hooks *embrace JavaScript* closures and avoid introducing React-specific APIs where JavaScript already provides a solution.

Reduce complexity in components.

Enable reuse:
Logic can easily be decoupled from components and shared among components.
Hooks are composable: new Hooks can be created by composing other Hooks.

Favoring *composition* over inheritance.

Why Hooks (2024)?

Because the React ecosystem has embraced Hooks.

All modern React libraries expose their API via Hooks:

- ReactRouter / ReactLocation
- MaterialUI
- MobX, ReactRedux, Recoil ...
- ReactQuery
- react-i18next
- ...

Basic Hooks

useState
useEffect
useRef
useContext
useReducer

useMemo
useCallback

useImperativeHandle
useSyncExternalStore

useTransition
useDeferredValue

useLayoutEffect
useInsertionEffect
useDebugValue
useId

useActionState
useOptimistic
useFormStatus

https://react.dev/reference/react/hooks
https://react.dev/reference/react-dom/hooks

The Rules of Hooks

(a part of the *Rules of React*)

Hooks are JavaScript functions, but you need to follow two rules when using them.

1. Only Call Hooks at the Top Level
   (don't call Hooks inside loops, conditions, or nested functions)
2. Only Call Hooks from React Functions
   (from function components or custom hooks)

Use the ESLint plugin to enforce these two rules:
https://www.npmjs.com/package/eslint-plugin-react-hooks

Rules of Hookes: https://react.dev/reference/rules/rules-of-hooks
Rules of React: https://react.dev/reference/rules
https://medium.com/@ryardley/react-hooks-not-magic-just-arrays-cd4f1857236e

EXERCISE

Exercises: Hooks - 1 to 5

DOM Access with `useRef`

React abstracts the real DOM behind the component tree and the virtual DOM.
But sometimes you need access to real DOM elements.

```
function TextInputWithFocusButton() {
  const inputEl = useRef(null);
  function buttonClicked() {
    // `current` points to the mounted text input element
    inputEl.current.focus();
  };
  return (
    <>
      <input ref={inputEl} type="text" />
      <button onClick={buttonClicked}>Focus the input</button>
    </>
  );
}
```

`useRef()` combined with the `ref` attribute can be used to access DOM nodes.

https://react.dev/reference/react/useRef#examples-dom

However, `useRef()` can be used for keeping any mutable value for the full lifetime of the component (a "replacement" for instance fields in classes)

https://react.dev/reference/react/useRef

React Context & useContext

Context provides a way to pass data through the component tree without having to pass props down manually at every level.

Providing a context

```
export const MyContext = React.createContext('my-context');
```

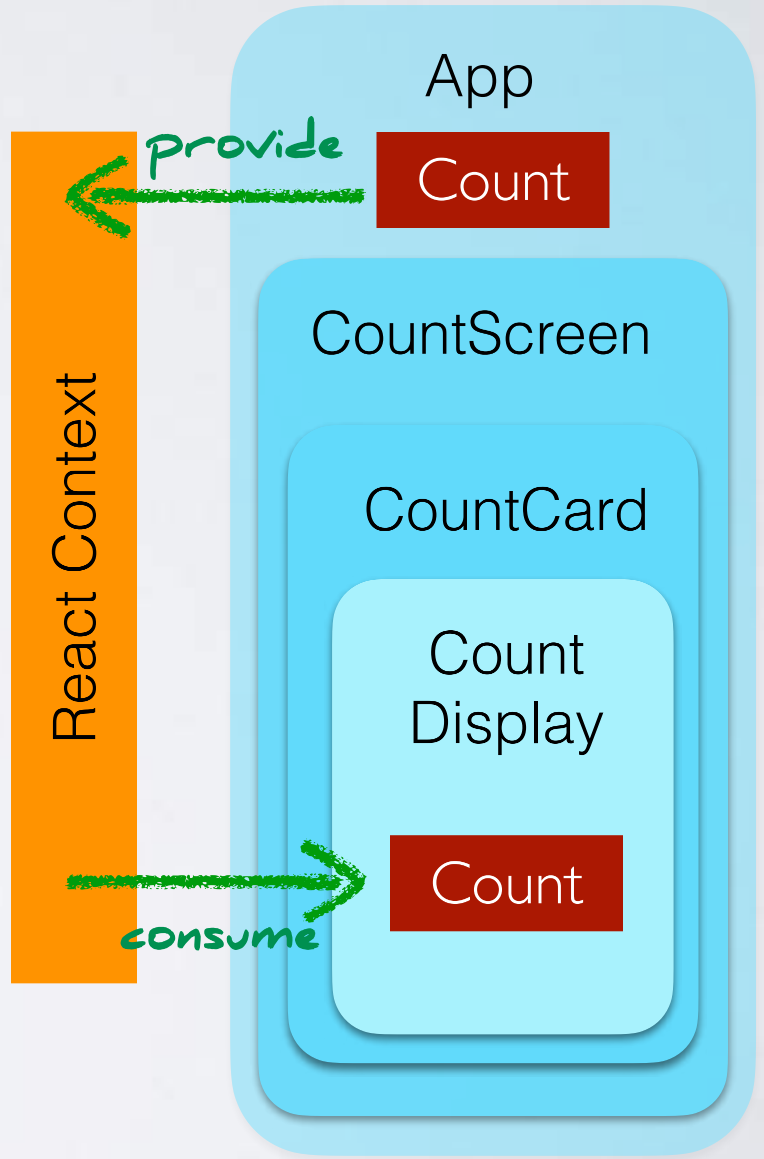
```
import {MyContext} from '../MyContext';
...
const contextObject = ...
<MyContext.Provider value={contextObject}>
    <App />
</MyContext.Provider>
```

an object is provided somewhere in the component tree

consuming a context

```
import React, {useContext} from 'react';
import {MyContext} from '../MyContext';
...
function MyComponent(){
   const contextObject = useContext(MyContext);
   // do something with the context object
}
```

the object can be consumed deeper down in the component tree



Note: in React v19 a context can be consumed with use (a new API which is not a Hook) https://react.dev/reference/react/use

https://react.dev/learn/passing-data-deeply-with-context
https://react.dev/learn/scaling-up-with-reducer-and-context

Note: historically there have been other ways to consume a context via render props and "injection" https://reactjs.org/docs/context.html

useReducer

https://react.dev/reference/react/useReducer

```
const initialState = {count: 0};

function reducer(state, action) {
  switch (action.type) {
    case 'increment':
      return {count: state.count + 1};
    case 'decrement':
      return {count: state.count - 1};
    default:
      throw new Error();
  }
}

function Counter() {
  const [state, dispatch] = useReducer(reducer, initialState);
  return (
    <>
      Count: {state.count}
      <button onClick={() => dispatch({type: 'decrement'})}>-</button>
      <button onClick={() => dispatch({type: 'increment'})}>+</button>
    </>
  );
}
```

https://react.dev/learn/extracting-state-logic-into-a-reducer#comparing-usestate-and-usereducer

Background: Reduce

```
var a = [1,2,3,4,5];

var result = a.reduce(
  // reducer function
  (acc, val) => {
    const sum = acc.sum + val;
    const count = acc.count + 1;
    const avg = sum/count;
    return {sum, count, avg};
  },
  // state object
  {sum:0, count:0, avg:0}
);

console.log('Statistics:', result);
```

The reducer function is a pure function.

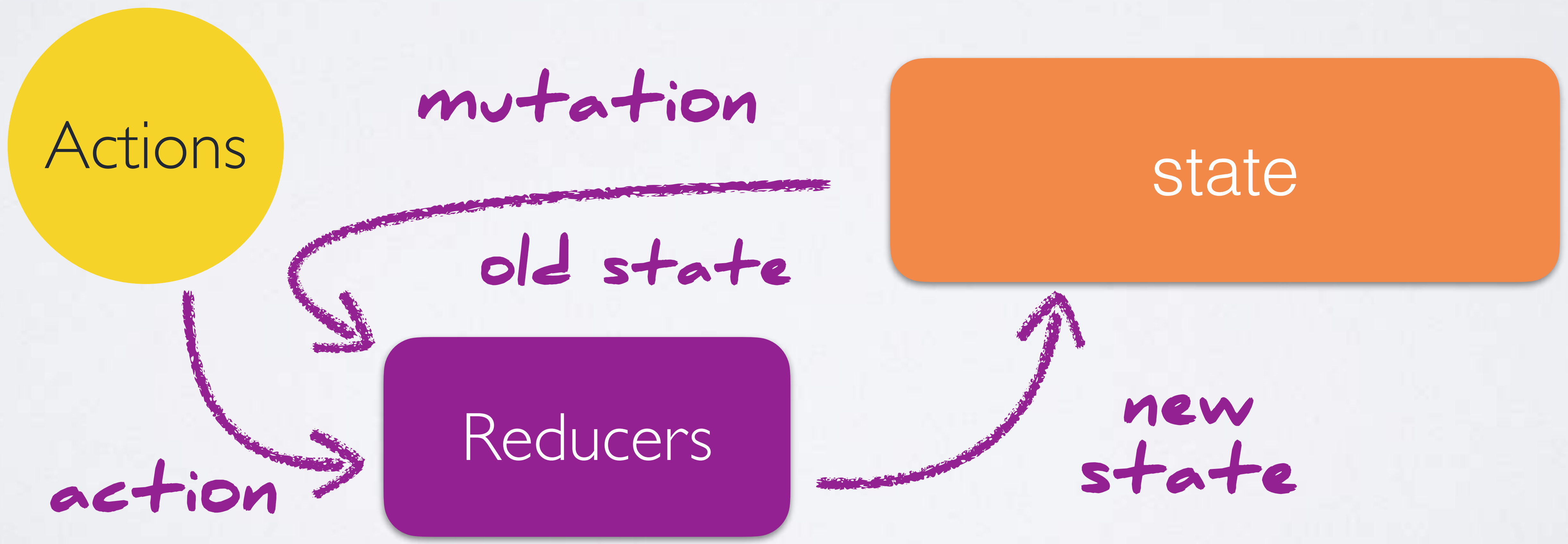
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/reduce

State Reducer Pattern

State changes are modelled with a reducer function.
With this pattern state updates can be consolidatedoutside of components.

(old state , action) => new state

aka: "reducer function"



Redux made this pattern popular: https://redux.js.org/

EXERCISE

Exercise: Hooks

Custom Hooks

"Hooks are just functions."
"Hooks can be composed from other hooks."

DEMO:

- simple wrapper around useState
- useToggle
- useState with localStorage persistence
- useTodoId in ToDo app router solution

Hook Libraries

https://github.com/streamich/react-use
https://usehooks.com/
https://github.com/antonioru/beautiful-react-hooks
https://usehooks-ts.com/
https://github.com/palmerhq/the-platform
https://github.com/rehooks/awesome-react-hooks
https://github.com/alibaba/hooks
https://nikgraf.github.io/react-hooks/
https://observable-hooks.js.org/
https://crutchcorn.github.io/rxjs-use-hooks/
...

Examples of custom Hooks:

- https://rangle.io/blog/simplifying-controlled-inputs-with-hooks/
- https://overreacted.io/making-setinterval-declarative-with-react-hooks/
- https://upmostly.com/tutorials/using-custom-react-hooks-simplify-forms/

3rd Party Hook Demos

useInterval

https://github.com/streamich/react-use/blob/master/docs/useInterval.md

useImmer

https://github.com/immerjs/use-immer

useAxios

https://github.com/simoneb/axios-hooks

Custom Hook

Hooks are just function. Custom Hooks can call other hooks.

usage in component:

```
function App() {
  const [name, setName] = useLocalStorageState('name');
  const handleChange = event => setName(event.target.value);
 return <input value={name} onChange={handleChange} id="name" />
}
```

custom hook:

```
function useLocalStorageState(key, defaultValue = '') {
  const [state, setState] = React.useState(
    () => window.localStorage.getItem(key) || defaultValue
  );

  React.useEffect(() => {
    window.localStorage.setItem(key, state);
  }, [key, state]);

  return [state, setState];
}
```

Custom Hook

custom hook:

```
function useForm(initialValues, submitHandler){

  const [formValues, setFormValues] = useState(initialValues);

  function handleChange(event) {
    const target = event.target;
    const value = target.type === "checkbox" ? target.checked : target.value;
    const name = target.name;
    setFormValues({...formValues, [name]: value});
  }

  function handleSubmit(e) {
    e.preventDefault();
    submitHandler(formValues)
  }

  return {formValues, handleChange, handleSubmit};
}
```

usage in component:

```
function AppComponent() {
  const initialValues = {isGoing: true, numberOfGuests: 0};
  let {formValues, handleChange, handleSubmit} = useForm(initialValues, submitForm);
  ...
```

https://reactjs.org/docs/hooks-custom.html

Custom Hooks for Data Fetching

```
function useUsers(){

  const [data, setData] = useState(initialValue);
  const [error, setError] = useState(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
      let cancelled = false;
      async function loadData(){
        try {
          const res = await fetch('https://jsonplaceholder.typicode.com/users');
          const resJson = await res.json();
          if (!cancelled) setData(resJson);
        } catch (err) {
          if (!cancelled) setError(err);
        } finally {
          if (!cancelled) setLoading(false);
        }
      }
      loadData();
      return () => { cancelled = true }
    }, []);

  return {loading, data, error};
}
```

custom Hook

```
function AppComponent() {
  const {loading, data, error} = useUsers();

  if (loading) return <Spinner/>;
  if (error) return <>...</>;
  ...
  return <>...</>
}
```

Consider using **axios-hooks**, **use-http**, **react-fetch-hook** or *TanStack Query* or *SWR* instead of implementing the low-level fetch.

EXERCISE

Exercise: Custom Hooks

Performance

React Performance

How fast is a "render-cycle"?
How many render cycles do we have?

Mechanisms from React:

- `memo`: higher order component ro prevent re-renders by memoizing a component
  https://reactjs.org/docs/react-api.html#reactmemo
- `useMemo`: hook to memoize a value to prevent expensive calculations on every render
  https://reactjs.org/docs/hooks-reference.html#usememo
- `useCallback`: hook to memoize a function to make callbacks
  https://reactjs.org/docs/hooks-reference.html#usecallback

https://kentcdodds.com/blog/fix-the-slow-render-before-you-fix-the-re-render
https://reactjs.org/docs/profiler.html

React Compiler

"under construction"

In the future the "React Compiler" will hopefully make manual optimization with memo, useMemo and useCallback obsolete ...

https://react.dev/learn/react-compiler

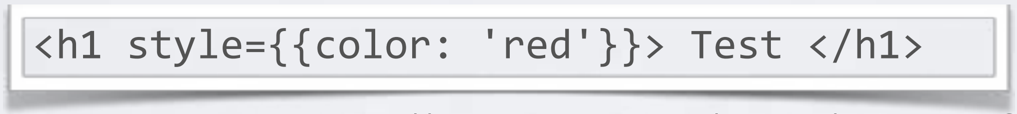
Styling React

Styling React Components

- Traditional CSS

  Optional with a CSS preprocessor: SASS, Less, Stylus

- Inline Styles



```
<h1 style={{color: 'red'}}> Test </h1>
```

  Are Inline Styles Faster than CSS? https://danielnagy.me/posts/Post_tsr8q6sx37pl

- CSS Modules

  CSS classes are scoped to components

- CSS-in-JS Library

  Generate styles with JavaScript

  - Emotion: https://github.com/emotion-js/emotion
  - Styled Components: https://github.com/styled-components/styled-components
  - Stitches: https://stitches.dev/
  - Vanilla Extract: https://vanilla-extract.style/
  - TSS React: https://www.tss-react.dev/

- Tailwind: https://tailwindcss.com/

https://nextjs.org/docs/app/building-your-application/styling/css-in-js

```
npm i node-sass @emotion/core @styled-components
```

```
/** @jsx jsx */
import React from 'react';
import {css, jsx} from '@emotion/core'
import styled from 'styled-components'
import styles from './Greeter.module.scss';

const Title = styled.h1`
  color: brown;
`;

export default function Greeter() {
  return (
    <div>
      <h1 className={styles.title}>Styled with CSS module</h1>
      <h1 css={css`
        color: pink;
      `}>Styled with Emotion</h1>
      <Title>Styled with Styled components</Title>
    </div>
  )
}
```

css modules

emotion

styled components

Note: typically it does not make sense to use different styling libraries!

TailwindCSS

utility-first CSS framework

```
<button class="px-4 py-1 text-sm text-purple-600 font-semibold rounded-full border
                 border-purple-200 hover:text-white hover:bg-purple-600
                 hover:border-transparent focus:outline-none focus:ring-2
                 focus:ring-purple-600 focus:ring-offset-2">
    Message
</button>
```

https://tailwindcss.com/docs/utility-first#why-not-just-use-inline-styles

Framework agnostic but especially popular in React and other JSX-based frameworks:
Installation for any framework:
https://tailwindcss.com/docs/installation/framework-guides

Tailwind is traditionally strong for styling raw html elements. Typically it can't be used to style a traditional component library.
But the rise of headless component libraries open a new usage-scenario for Tailwind.

https://tailwindcss.com/

Tailwind is very controversial

Tailwind CSS is the worst:

https://www.youtube.com/watch?v=lHZwlzOUOZ4

The Tailwind CSS Drama Your Users Don't Care About

https://www.builder.io/blog/the-tailwind-css-drama-your-users-don't-care-about

Why I don't like Tailwind:

https://www.aleksandrhovhannisyan.com/blog/why-i-dont-like-tailwind-css/

Unstyled Components + Tailwind = shadcn

shadcn/ui https://ui.shadcn.com/

"Code generator for components"
"Components are included as source code not as npm packages"

Component Libraries

Component Libraries

Material UI

https://mui.com/

- Chackra UI:
  https://chakra-ui.com/
- Ant Design of React
  https://ant.design/docs/react/introduce
- Semantic UI
  https://react.semantic-ui.com/

And more:
Rainbow UI, Cloudscape Design System, react-bootstrap, reactstrap ...

- KendoReact
  https://www.telerik.com/kendo-react-ui/
- PrimeReact
  https://www.primefaces.org/primereact/
- Infragistics / Ignite UI:
  https://www.infragistics.com/products/ignite-ui-react
- DevExtreme
  https://js.devexpress.com/
- Syncfusion:
  https://www.syncfusion.com/react-ui-components
- jQWidgets:
  https://www.jqwidgets.com/react/react-js-components.htm
- agGrid
  https://www.ag-grid.com/

https://github.com/brillout/awesome-react-components

Headless Components

Headless Components

Components with minimal or no UI.

Radix https://www.radix-ui.com/

React Aria https://react-spectrum.adobe.com/react-aria/index.html

headless UI https://headlessui.com/

base ui https://ark-ui.com/

https://base-ui.com/

Often combined with Tailwind: https://tailwindcss.com/

shadcn/ui https://ui.shadcn.com/

TanStack Table: https://tanstack.com/table
ReactRanger: https://github.com/tannerlinsley/react-ranger

TanStack Form: https://tanstack.com/form
HouseForm: https://houseform.dev/

```
npm install @mui/material @emotion/react @emotion/styled
```

```
import Button from "@mui/material/Button";

<Button variant="contained" onClick={increment}>Click Me!</Button>

<Button sx={{ color: "red" }} onClick={increment}>Click Me!</Button>
```

Radix

first install tailwind: https://tailwindcss.com/docs/guides/vite

```
npm install @radix-ui/react-switch
```

unstyled:

```
<div className=" items-center">
  <div>Switch</div>
  <div>
    <Switch.Root>
      <Switch.Thumb />
    </Switch.Root>
  </div>
</div>
```

styled:

```
<Switch.Root className="w-[42px] h-[25px] bg-blackA6 rounded-full relative shadow-[0_2px_10px]
              shadow-blackA4 focus:shadow-[0_0_0_2px] focus:shadow-black data-[state=checked]:bg-
              black outline-none cursor-default">
  <Switch.Thumb className="block w-[21px] h-[21px] bg-white rounded-full shadow-[0_2px_2px]
            shadow-blackA4 transition-transform duration-100 translate-x-0.5 will-change-transform
            data-[state=checked]:translate-x-[19px]" />
</Switch.Root>
```