Online Multimedia

Winter Semester 2019/20

Tutorial 05 – React

Today's Agenda

- React Basics
- React Core Concepts
- React Hooks
- Roundup Quiz

Survey & Feedback

Please help us improve the tutorials and assignments by filling out this survey

https://survey.medien.ifi.lmu.de/index.php/752321/lang-en

Recap

Web Components

Custom Elements

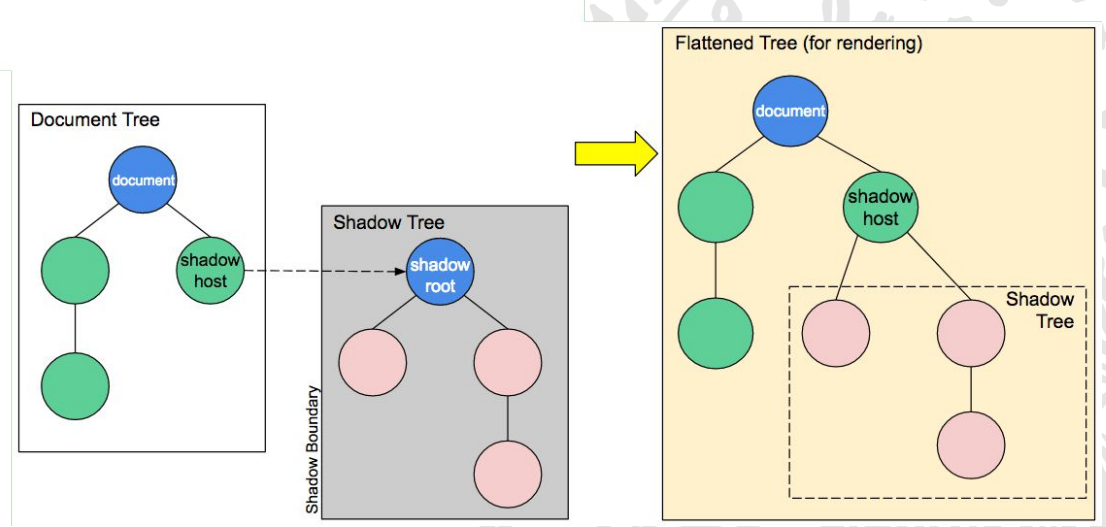
```
<my-element></my-element>
```

HTML Imports

```
<link rel="import"
  href="my-code.html">
```

HTML Templates

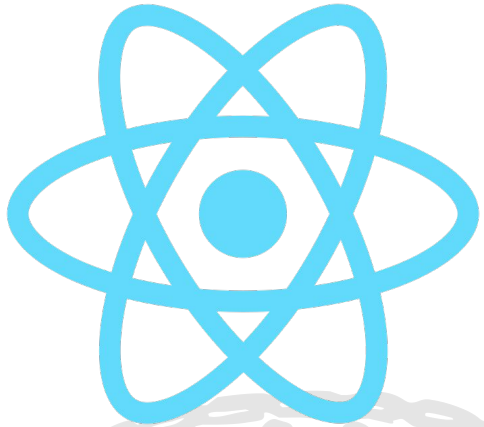
```
<template>
  This content is rendered
  later on.
</template>
```



https://mdn.mozillademos.org/files/15788/shadow-dom.png

React Introduction

React



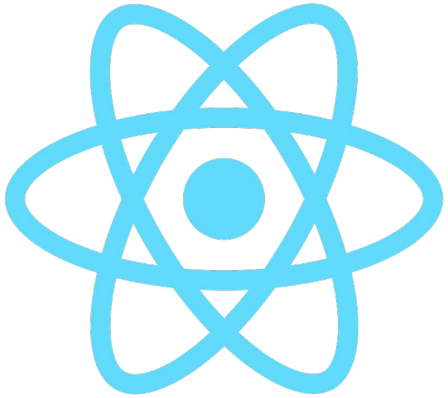
- “A JavaScript library for building **user interfaces**.”
- React is a library for developing single-page web applications
- It is lightweight and a complement solution to Web Components

- Currently on version 16+

Why didn't we choose XYZ?

- Angular vs. React vs. Vue
- There are no one greater than the others in all aspects, **obviously**
- We use React this year because we like its simplicity and try new things

Angular

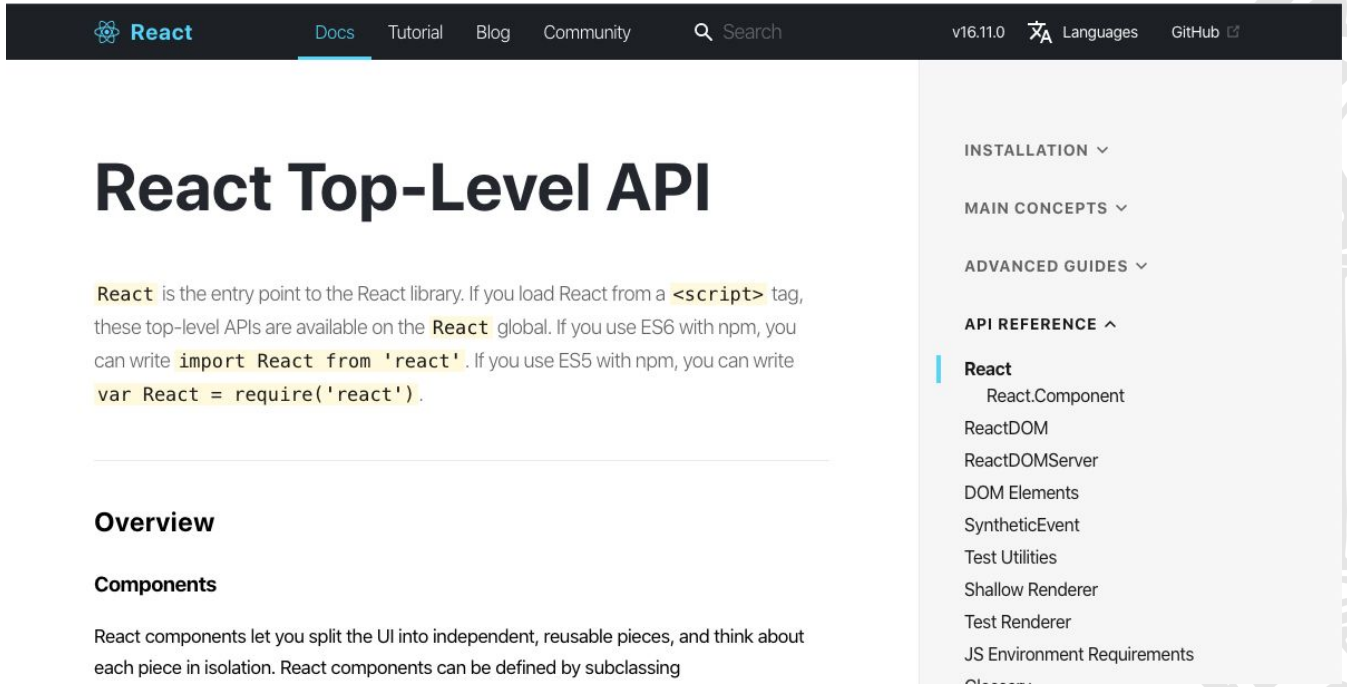


React

Vue

Read the Fantastic Manual

- The documentation to get you started: https://reactjs.org/docs/getting-started.html
- Detailed docs on library elements: https://reactjs.org/docs/react-api.html



React – Getting Started

- The easiest way to get started with React is via the Create React App
- With npm installed, execute:

```
npx create-react-app <app-name> --typescript
```

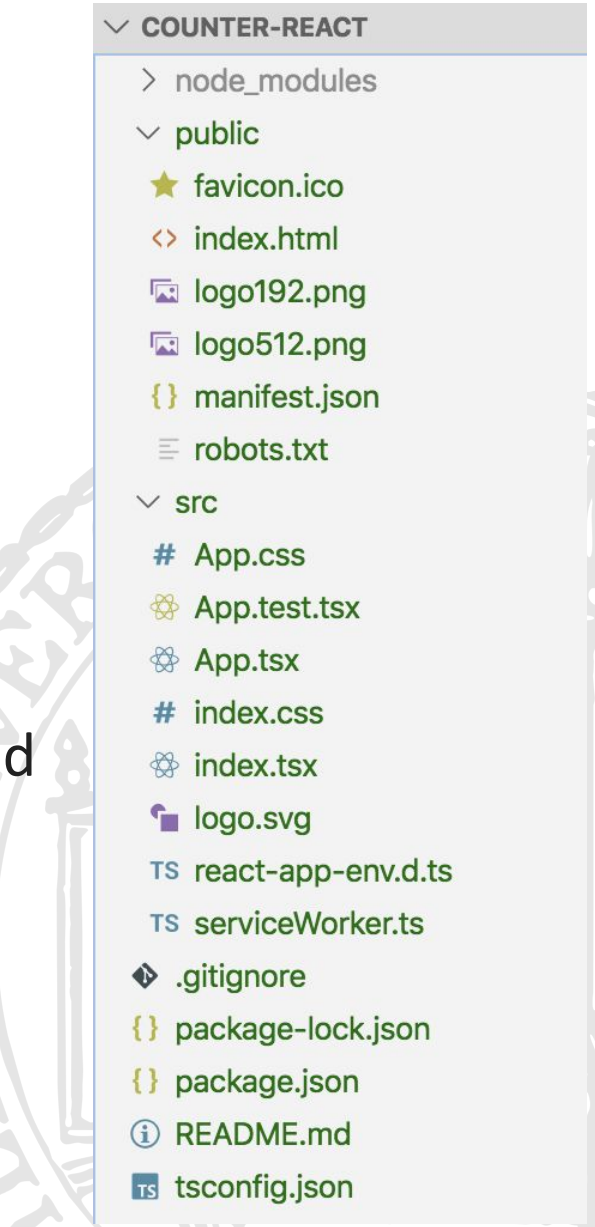
- The Tool will generate a complete React app for you to modify
  - This skeleton is a node application, and installed all dependencies for you
  - It comes with a simple development server that you can start with

    ```
    npm start
    ```

  - The tool generates something we will not cover, like service worker. Feel free to read up on them

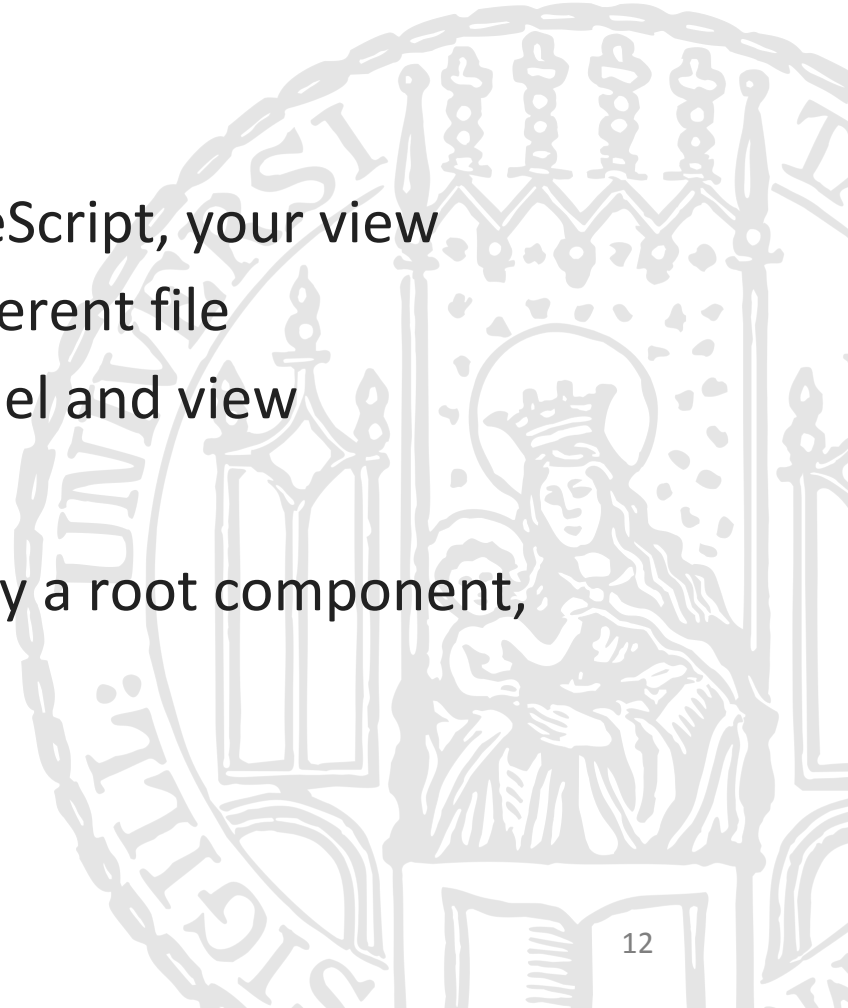
Structure of an React App

- Necessary config files
  - `tsconfig.json`: for the TypeScript compiler
  - `package.json`: for application management
- Public folder: static files
- You actual application in `/src`
  - `index.tsx, index.css` - Bootstrapping files, they bind your application together
  - `App.tsx` - A customized component
  - `App.css` - The component styles
  - `App.test.tsx` - Testing file regarding the component
  - `serviceWorker.ts` - We won't cover this



React – Your Part

- An React app is built from small building blocks
- Each component consists of three parts:
  - The component, written in TypeScript, your model
  - The component also mixes *HTML-ish* code in TypeScript, your view
  - Optional style information, can be organized in different file
- You'll notice that there is no controller to connect model and view
- We use state-management instead
- To use a component, one has to eventually imported by a root component, which will be rendered via `ReactDOM.render`.



Our First Component

- To create a new component, create a TypeScript file with `.tsx` extension
  - A good practice to structure your components is to place one component in a dedicated folder
- Create a class for your component
- Extends the class from `React.Component`

```tsx
import React, {Component} from 'react'

export class OmmCounter extends Component {
  state = { count: 0 }
  inc = () => {
     this.setState({count: (this.state.count + 1)}) }
  dec = () => {
     this.setState({count: (this.state.count - 1)}) }
  render() {
    return (
      <div>
        <span>{ this.state.count }</span>
        <div>
          <button onClick={this.inc}>+</button>
          <button onClick={this.dec}>-</button>
        </div>
      </div>
    )
  }
}
```

omm-counter.tsx

Using the Component

- Now you can use the counter like any other HTML tag
- Copy styles we prepared for you from GitHub

- When you inspect the page in the browser, you will notice a included script:
  - `main.chunk.js` -- Your bundled components

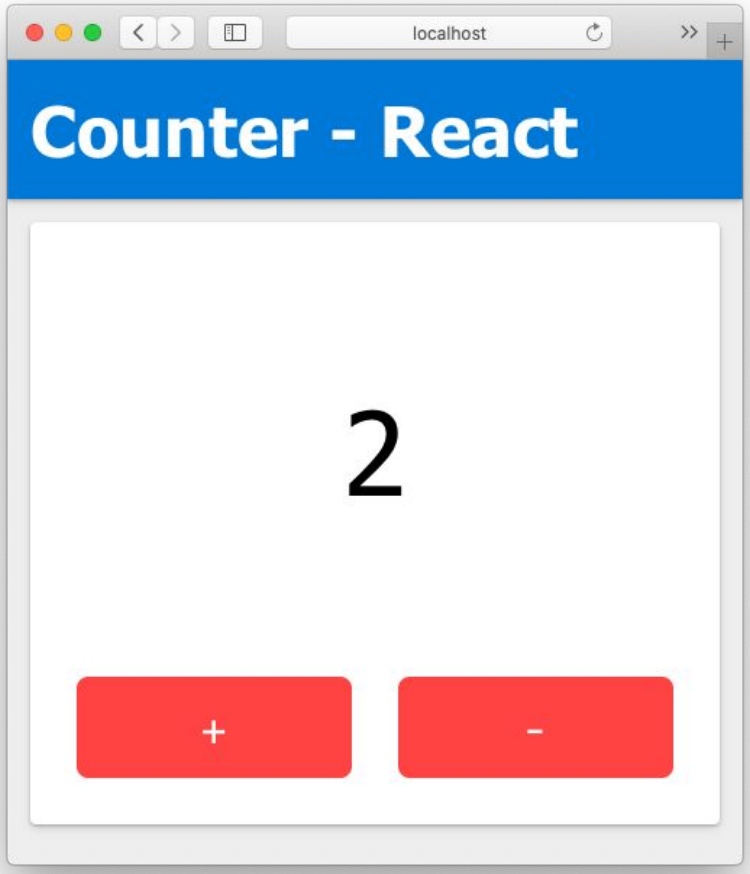
```tsx
import React from 'react';
import './App.css';
import OmmCounter from './components/omm-counter/omm-counter';

const App: React.FC = () => {
  return (
    <div className="app">
      <div className="header">
          <h1>Counter - React</h1>
      </div>
      <OmmCounter />
    </div>
  );
}

export default App;
```

App.tsx

Result



React Core Concepts


WAIT NOW I PUT MY HTML
IN JAVASCRIPT OBJECTS WTF
JSX LOOKS LIKE PERL
CGI ALL OVER AGAIN
imgflip.com

Understanding JSX/TSX

- JSX/TSX is just a syntax sugar to eliminate cumbersome JS calls
- React transpiles this syntax to JS calls

```
export default class OmmCounter extends Component {
  …
  render() {
    return
    React.createElement('div', null,
      React.createElement('span', null, this.state.count),
      React.createElement('div', null,
        React.createElement('button', {onClick: this.inc}, '+'),
        React.createElement('button', {onClick: this.dec}, '-')))
  }
}
```

omm-counter.tsx

```
export default class OmmCounter extends Component {
  …
  render() {
    return (
      <div>
        <span>{ this.state.count }</span>
        <div>
          <button onClick={this.inc}>+</button>
          <button onClick={this.dec}>-</button>
        </div>
      </div>
    )
  }
}
```

omm-counter.tsx

Rendering Element

- A *class-based component* App is registered and rendered in root element (`ReactDOM.render`)
- The App component uses our `OmmCounter`
- `OmmCounter` is declared in its class's `render` method

```tsx
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
ReactDOM.render(<App />, document.getElementById('root'));
```

index.tsx

```tsx
export default class OmmCounter extends Component {
  …
  render() {
    return (
      <div>
        <span>{ this.state.count }</span>
        <div>
          <button onClick={this.inc}>+</button>
          <button onClick={this.dec}>-</button>
        </div>
      </div>
    )
  }
}
```

omm-counter.tsx

State Management

- React use state to manage all UI elements, changes in state trigger React to re-render components and potentially update the DOM in browser
- To update the state, use `this.setState`
- `setState` merges new states and

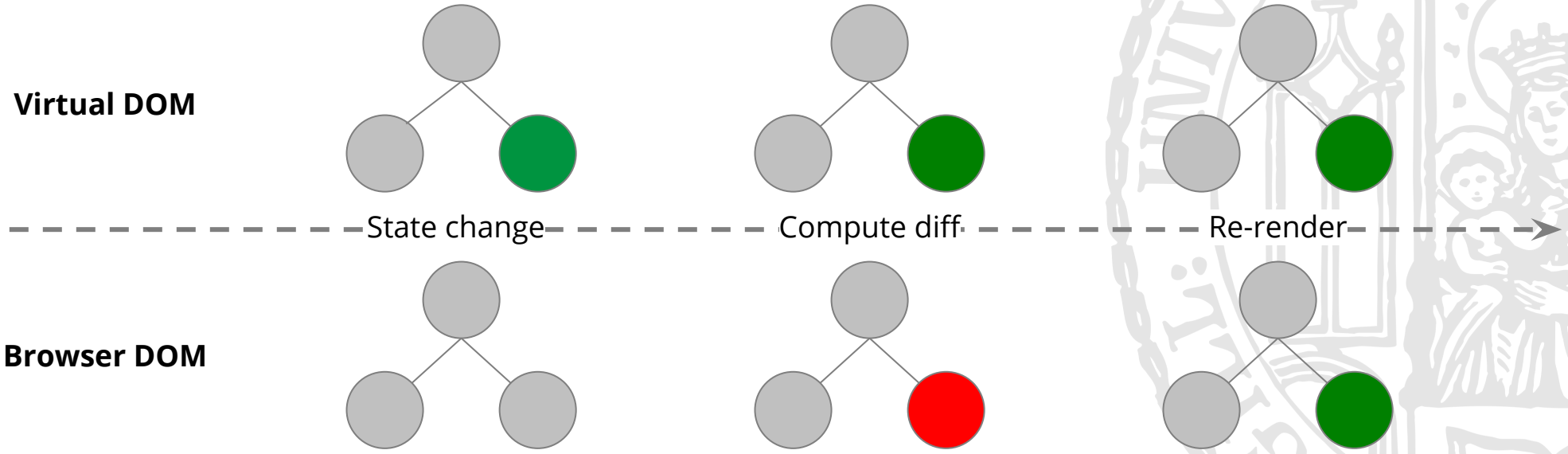
replaces corresponding old states

```
import React, {Component} from 'react'
export class OmmCounter extends Component {
  // state can only be access in class-based components
  state = { count: 0 }
  inc = () => {
     this.setState({count: (this.state.count + 1)}) }
  dec = () => {
     this.setState({count: (this.state.count - 1)}) }
  render() {
     …
  }
}
```

omm-counter.tsx

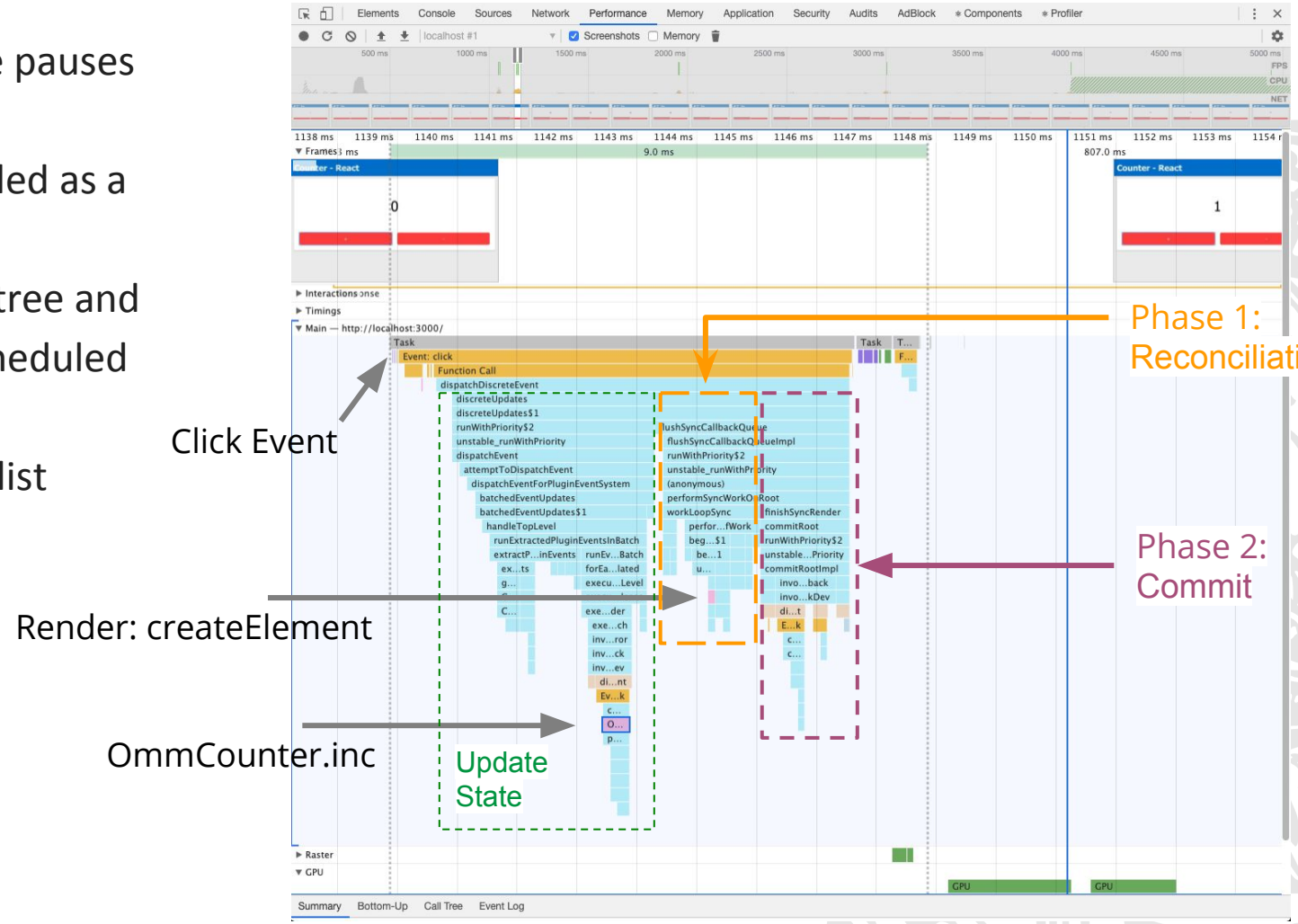
Reconciliation

- React keeps a "virtual" representation (*Virtual DOM*) of a UI in memory and synced with the "real" DOM by `ReactDOM`. The process of synchronization is called reconciliation.
- When the component state is updated, the `ReactDOM.render` will be recalled and compares the "real" DOM to patch their diff.



Fiber Engine: Two-Phase Update Processes

- Traverse update a large DOM tree pauses main thread (recall event loop)
- Hence, an update patch is scheduled as a fiber node
- ***Reconciliation phase*** builds fiber tree and ensures higher priority tasks is scheduled after a timeslice (interruptible)
- ***Commit phase*** commits all effect list (uninterruptible)



Event Handling

- React events are named using camelCase, rather than lower case.
- To evoke a event, pass a function reference rather than calling is in the brackets

```
export default class OmmCounter extends Component {
  …
  render() {
    return (
      <div>
        <span>{ this.state.count }</span>
        <div>
          <button onClick={this.inc}>+</button>
          <button onClick={this.dec}>-</button>
        </div>
      </div>
    )
  }
}
```

omm-counter.tsx

Breakout #1: The MemeMUC Component

Using the skeleton from GitHub, we create a new component for MemeMUC - The OMM Meme Generator

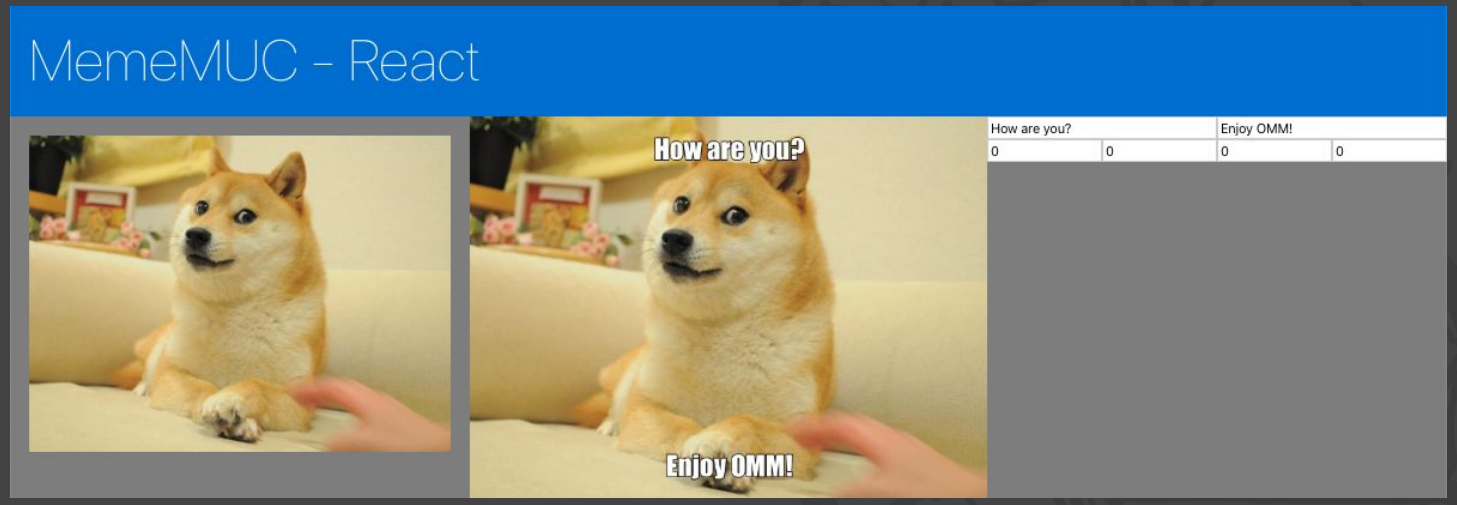
This component is a UI to create memes

The user should select a meme from a list of available images (hardcoded for now)

Some inputs allow changing the memes parameters

The meme-image is automatically updated. You can use API at http://mememuc.medien.ifi.lmu.de:3000/.

Timeframe: **15 Minutes**



Props

- `props` works like function parameters, which allows you pass data from a parent (wrapping) component to a child (embedded) component.
- Changes in props trigger React to re-render the components and potentially update the DOM in the browser.

```tsx
import React, { Component } from 'react';
import Child from './child'
export default class Parent extends Component {
  render() {
    return (
      <div><Child name={"Max"} age={18}/></div>
    )
  }
}
```

parent.tsx

```tsx
import React, { Component } from 'react';
interface ChildProps { name: string, age: number }
export default class Child extends Component<ChildProps> {
  render() {
    return (<div>
        <div>name: {this.props.name}</div>
        <div>age: {this.props.age}</div>
        {/* <div>role: {this.props.role} </div> */}
    </div>)
}}
```

child.tsx

Conditional Rendering

- Conditional rendering in React can be easily done in pure JS logic by using if statement

```
if (condition) {
   optionalRender = (<div>render under a condition</div>)
}
return (<div>{optionalRender}</div>)
```

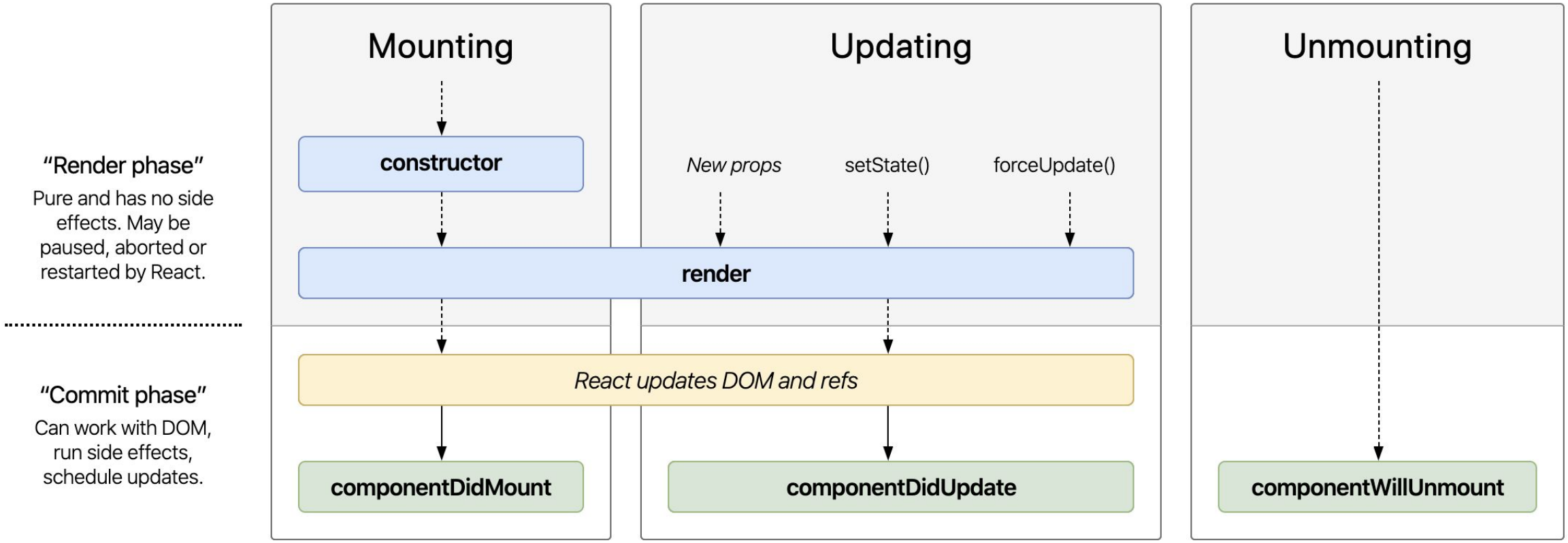
Repetitive Rendering

- Similarly, to render a list of components, one can use .map call or simply use for loop to construct repetitive rendering template
- It is more efficient to pass an unique id to key property in the mapped components

```
return (<div>{
   this.state.stateArray.map(item => {
      return <div key={item.id}>{item.property}</div>
   })
}</div>)
```

React Lifecycle Method Diagram

(only in class-based components)



http://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/

React Hooks

Functional Components

- Class components ("stateful") was the only way to create an component
- However, building UI is very rare in designing a class system
- Functional components ("stateless") simply use JS functions to create an component:

```
const c: React.FC = () => {
   return (<div>TSX</div>)
}
```

```
import React from 'react';
import './App.css';
import OmmCounter from './components/omm-counter/omm-counter'
const App: React.FC = () => {
  return (
    <div className="app">
      <div className="header"><h1>Counter - React
Hooks</h1></div>
      <OmmCounter />
    </div>
  );
}
export default App;
```

App.tsx

State Hook

- In functional components, there are no `state`, to manage states, we can involve `useState` hook:

```
const [currentState, setCurrentState] = useState('any data')
const F = () => { setCurrentState('new state') }
```

- `useState()` returns an array with exactly two elements:
  - Getter: Your current state value
  - Setter: A method to update your state value

State Hook

- Rewrite `omm-counter` using React Hooks:

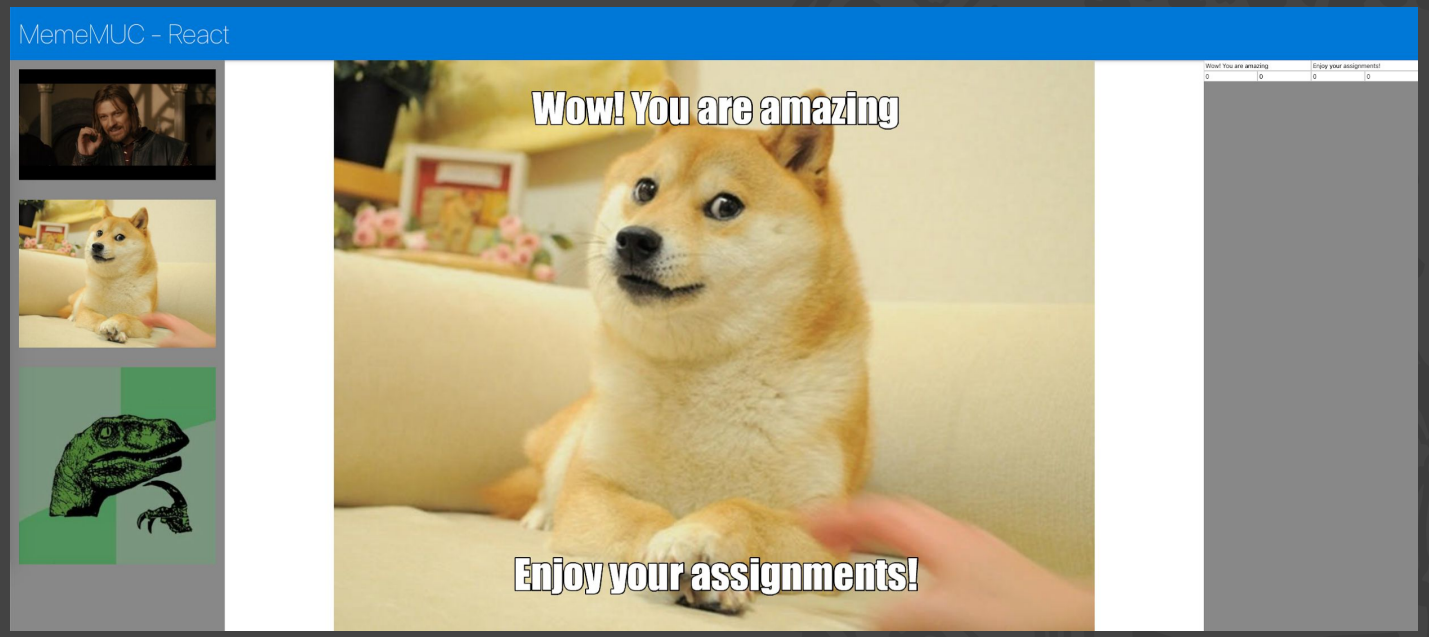
```
import React, { useState } from 'react';
const OmmCounter: React.FC = () => {
  const [getter, setter] = useState({counter: 0})
  const inc = () => {setter({counter: getter.counter + 1})}
  const dec = () => {setter({counter: getter.counter - 1})}
  return (
    <div><span className="counter-state">{ getter.counter }</span><div>
      <button onClick={inc}>+</button>
      <button onClick={dec}>-</button>
    </div></div>
  )}
export default OmmCounter
```

App.tsx

Breakout #2: Meme Generator with Hooks

1. Read https://reactjs.org/docs/hooks-effect.html#effects-without-cleanup, understanding how to replace `componentDidMount()` by `useEffect()`
2. Rewrite the `OmmMemeMUC` using `useState` and `useEffect` Hooks
3. Request Memes using fetch API and update memes when `componentDidMount`

Timeframe: **15 Minutes**



Round-up Quiz

1. Which argument you need in Create React App for using TypeScript?
2. What is the underlying function in JSX/TSX syntax to create an HTML tag in a component?
3. What function do you need to render a component in ReactDOM?
4. How to change states in a React component? Name two approaches.
5. What is the reconciliation in React? Explain in one sentence.
6. What are the two phases in React Fiber?
7. What is the naming convention to handle events in React?
8. What is the attribute to evoke a button click event?
9. Why we need props? Explain in one sentence.
10. How to check a component is updated? Name two approaches.

Confusing? 😕

- Don't worry, we'll do a recap next week.
- In the meantime, watch some tutorials and play around with React
- Examples:
  - https://www.youtube.com/watch?v=sBws8MSXN7A
  - https://www.youtube.com/watch?v=dpw9EHDh2bM
  - https://www.youtube.com/watch?v=ZCuYPiUIONs

Thanks!
What are your questions?

Appendix

Advanced React Topics

- Higher Order Components (HOC)
  - https://reactjs.org/docs/higher-order-components.html
- Context
  - https://reactjs.org/docs/context.html
- Refs
  - https://reactjs.org/docs/forwarding-refs.html
- More Hooks: useContext, …
  - https://reactjs.org/docs/hooks-reference.html
- Concurrent Mode & Suspense
  - https://reactjs.org/docs/concurrent-mode-intro.html
- Virtual DOM and Internals
  - https://reactjs.org/docs/faq-internals.html