Typescript Bootcamp 2023

React 3

Andre Siggia
10 Feb 2023

Topics

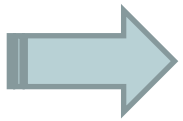
- Class components
- Class component lifecycle
- Function component lifecycle
- React error handling
- Synthetic events
- Working with forms

Class components

https://beta.reactjs.org/reference/react/Component#migrating-a-simple-component-from-a-class-to-a-function

Component written as a function

```
function GreetingsFn(props) {
  return (
    <div>
      Hello {props.name}!
    </div>
  );
}
```



Component written as a JavaScript/ES6 Class

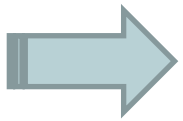
?

Class components

https://beta.reactjs.org/reference/react/Component#migrating-a-simple-component-from-a-class-to-a-function

Component written as a function

```
function GreetingsFn(props) {
  return (
    <div>
      Hello {props.name}!
    </div>
  );
}
```



Component written as a JavaScript/ES6 Class

```
class GreetingsClass extends React.Component {
  render() {
    return (
      <div>
        Hello {this.props.name}!
      </div>
    );
  }
}
```

“render” is its only **required** method

</> https://codesandbox.io/s/component-lifecycle-i9r7w8?file=/src/pages/Greetings.js

Class component lifecycle

https://beta.reactjs.org/reference/react/Component

 **Mounting**

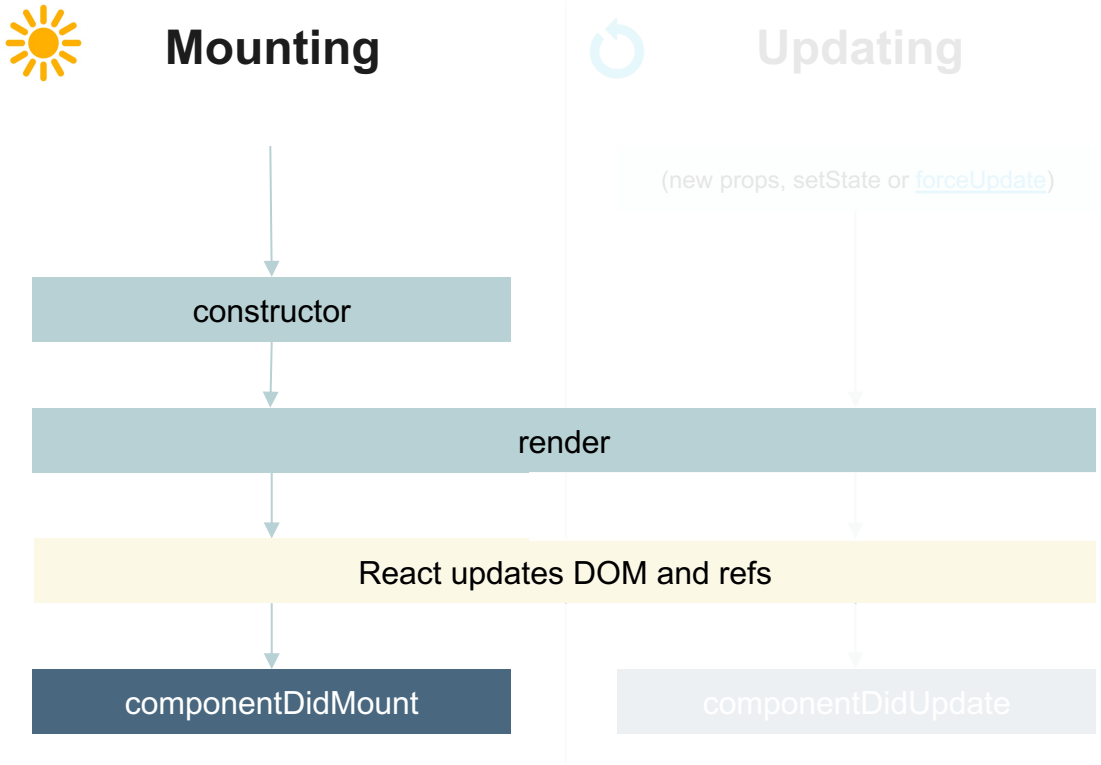
 **Updating**

 **Unmounting**

 **Error Handling**

Class component lifecycle

https://beta.reactjs.org/reference/react/Component



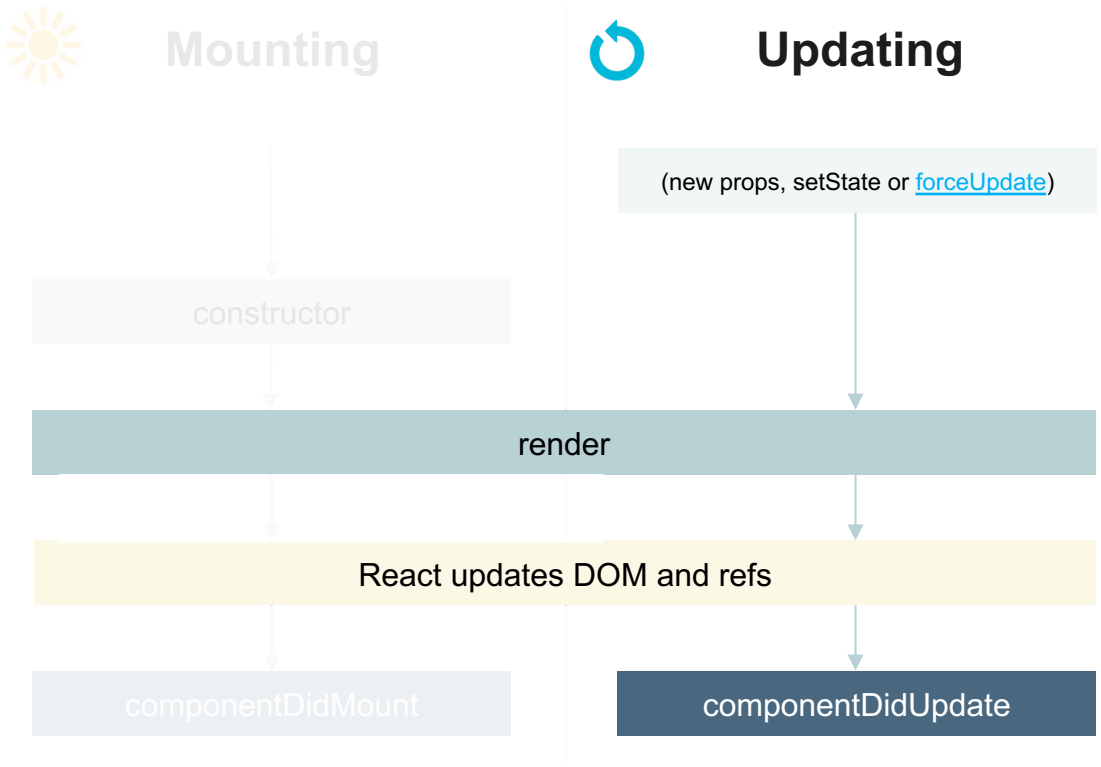
The component is being created and inserted into the DOM

1. constructor() is called before component is mounted
2. render() returns what you want to appear on the screen; it is the only required method in a class component
3. componentDidMount() is called right after component is inserted into the DOM tree

</> https://codesandbox.io/s/component-lifecycle-i9r7w8?file=/src/pages/Mounting.js

Class component lifecycle

https://beta.reactjs.org/reference/react/Component



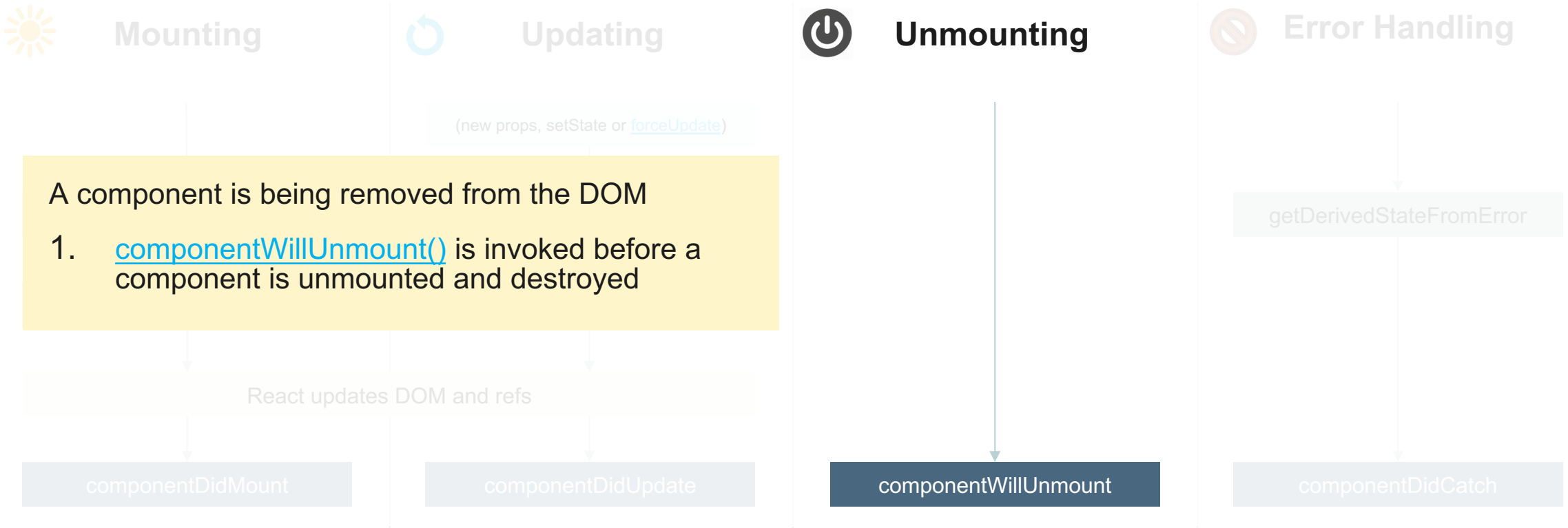
A component is being re-rendered

1. render() (same as in mounting phase)
2. componentDidUpdate() is invoked immediately after updating occurs

</> https://codesandbox.io/s/component-lifecycle-i9r7w8?file=/src/pages/Count.js

Class component lifecycle

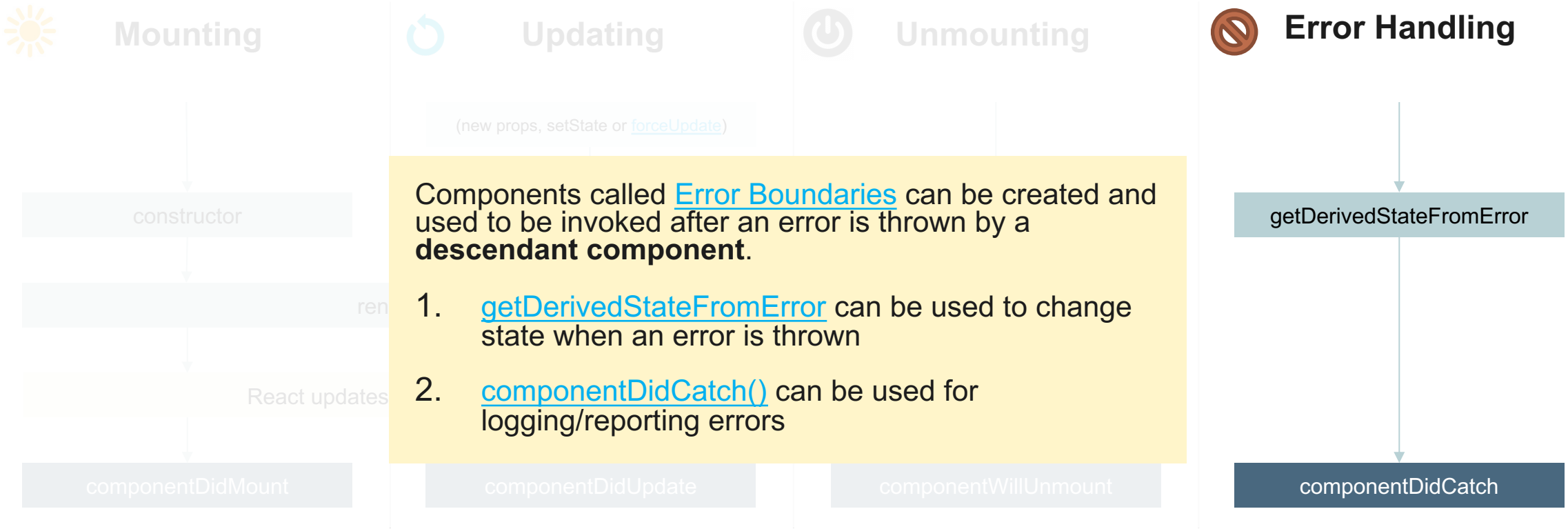
https://beta.reactjs.org/reference/react/Component



https://codesandbox.io/s/component-lifecycle-i9r7w8?file=/src/pages/Unmounting.js

Class component lifecycle

https://beta.reactjs.org/reference/react/Component

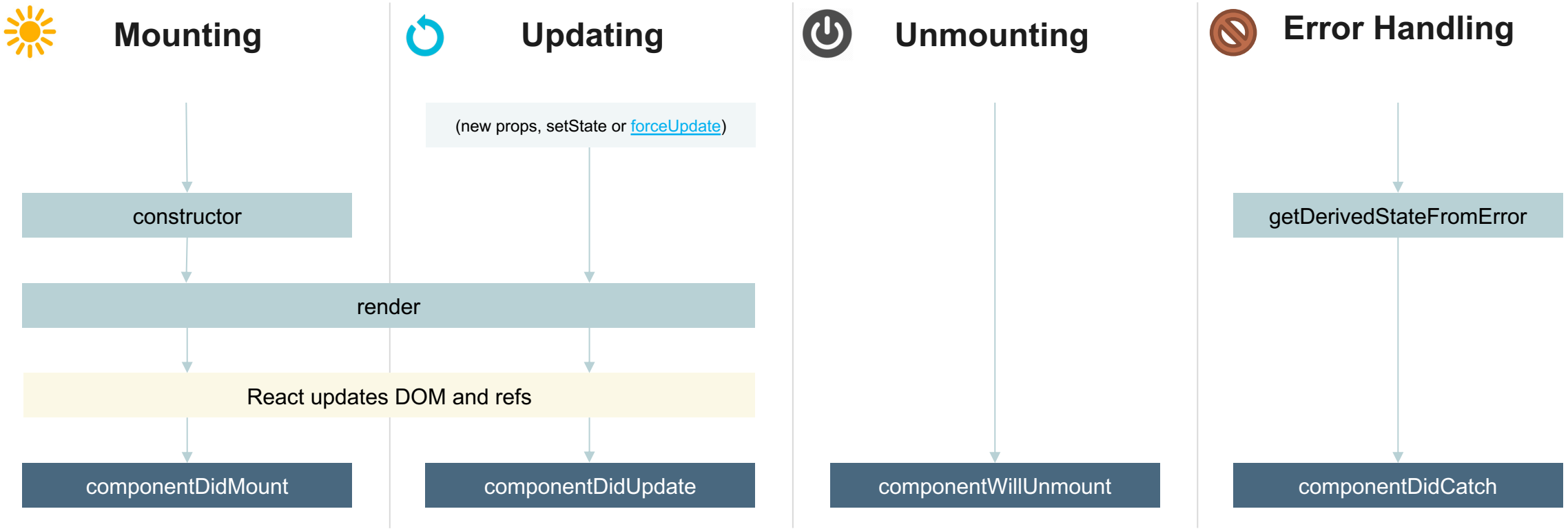


Components called Error Boundaries can be created and used to be invoked after an error is thrown by a **descendant component**.

1. getDerivedStateFromError can be used to change state when an error is thrown
2. componentDidCatch() can be used for logging/reporting errors

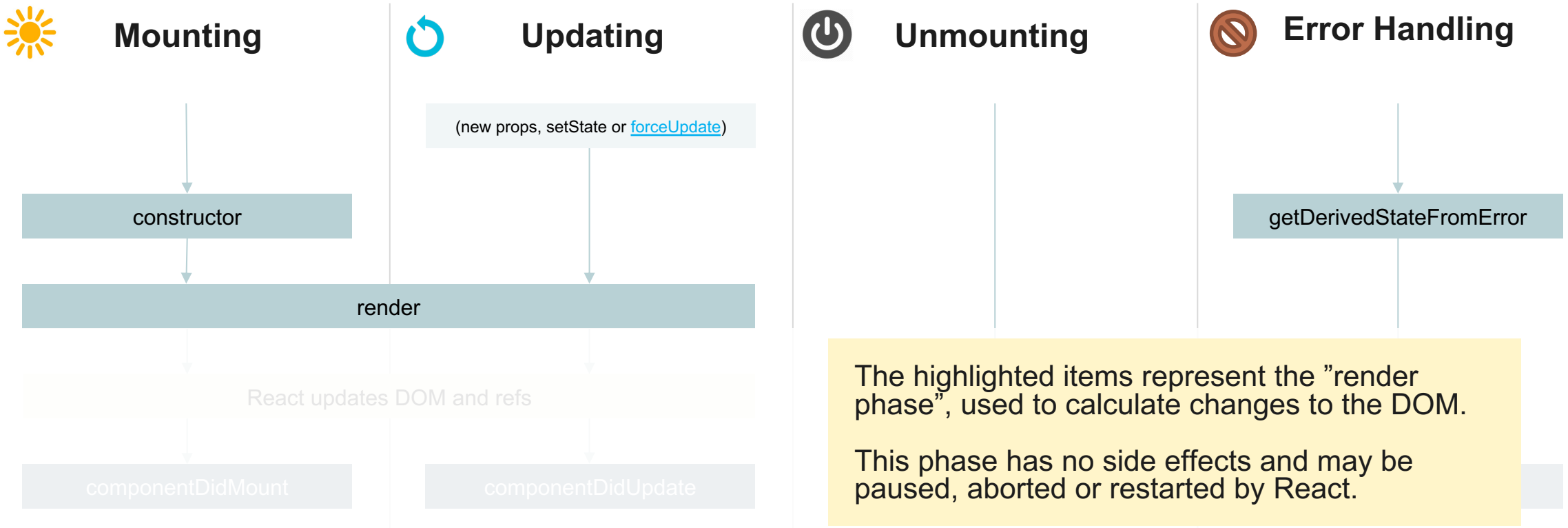
Class component lifecycle

https://beta.reactjs.org/reference/react/Component



Class component lifecycle

https://beta.reactjs.org/reference/react/Component

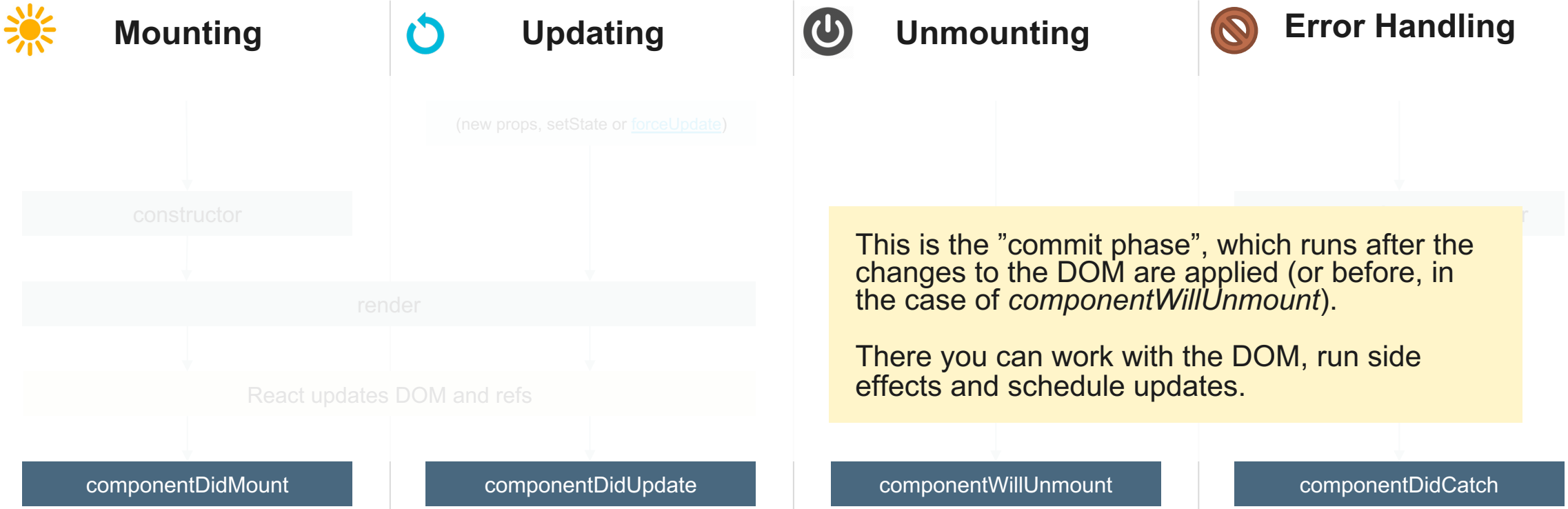


The highlighted items represent the ”render phase”, used to calculate changes to the DOM.

This phase has no side effects and may be paused, aborted or restarted by React.

Class component lifecycle

https://beta.reactjs.org/reference/react/Component

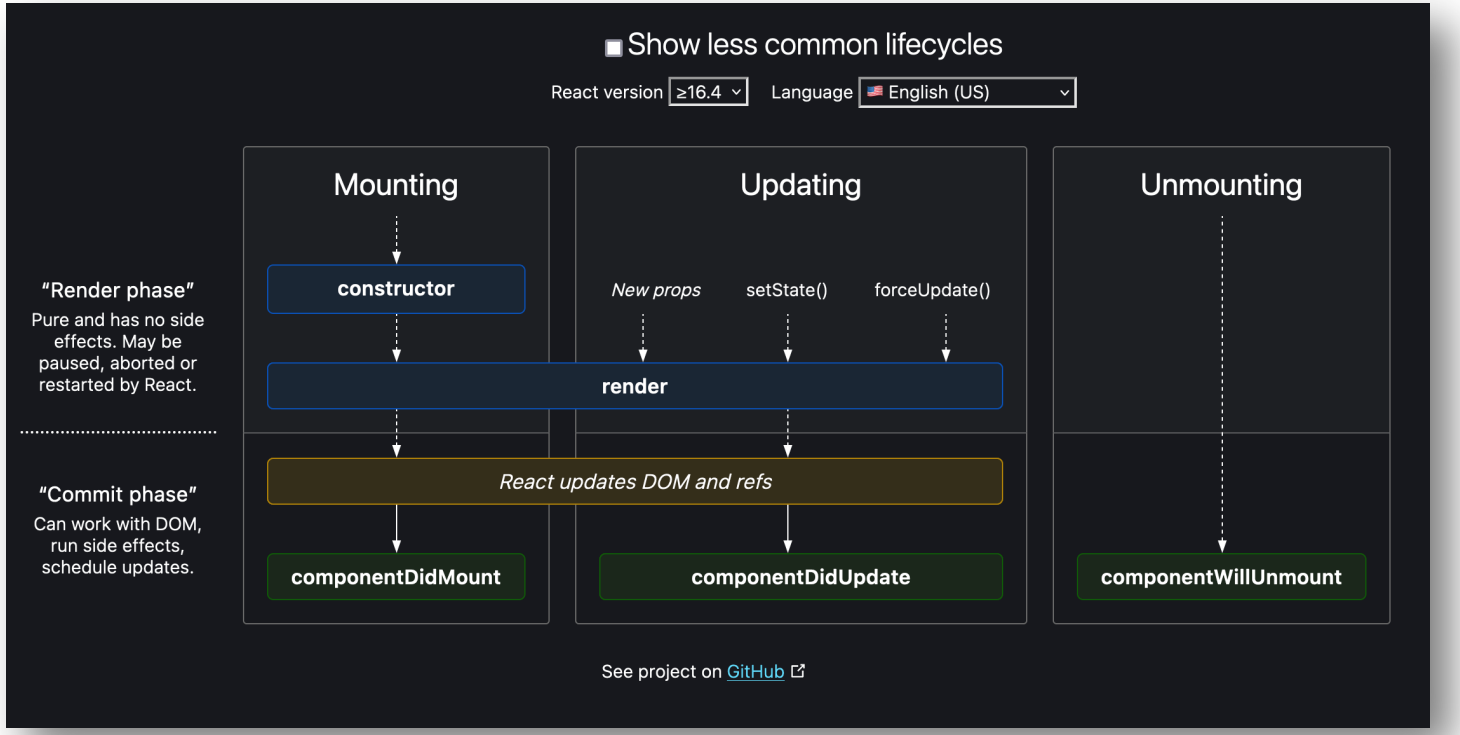


This is the ”commit phase”, which runs after the changes to the DOM are applied (or before, in the case of *componentWillUnmount*).

There you can work with the DOM, run side effects and schedule updates.

Class component lifecycle

https://beta.reactjs.org/reference/react/Component

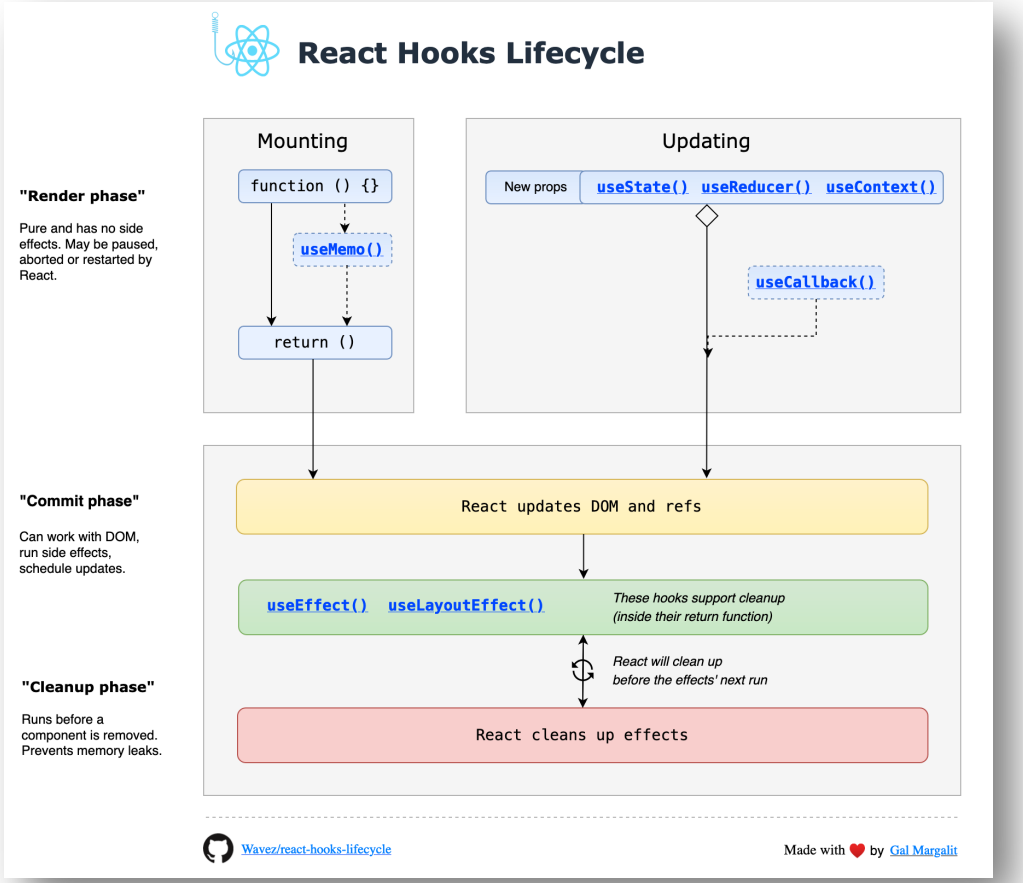


Interactive diagram

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

Function component lifecycle

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



Interactive diagram

https://wavez.github.io/react-hooks-lifecycle/

Function component lifecycle

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

 **Mounting**

 **Updating**

 **Unmounting**

 **Error Handling**

Function component lifecycle

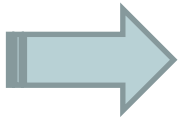
https://reactjs.org/docs/hooks-effect.html

Mounting

useEffect with [ ]

```
state = {
  isOnline: false
};

componentDidMount() {
  ExampleAPI.getStatus().then(({ isOnline }) => {
    this.setState({ isOnline });
  });
}
```



```
const [isOnline, setIsOnline] = React.useState(false);

React.useEffect(() => {
  ExampleAPI.getStatus().then(({ isOnline }) => {
    setIsOnline(isOnline);
  });
}, []);
```

For once-off effects, just call useEffect with an empty array ([ ]) as second parameter.

Updating

useEffect with state(s) or prop(s)

Unmounting

return from useEffect

Error Handling

Error Boundary

</> https://codesandbox.io/s/component-lifecycle-i9r7w8?file=/src/pages/Mounting.js

Function component lifecycle

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

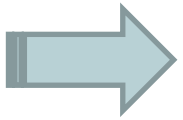
Mounting | **Updating** | Unmounting | Error Handling

useEffect with [ ] | useEffect with state(s) or prop(s) | return from useEffect | Error Boundary

```
state = {
  count: 0
};

componentDidMount() {
  document.title = `You clicked ${this.state.count} times`;
}

componentDidUpdate() {
  document.title = `You clicked ${this.state.count} times`;
}
```



```
const [count, setCount] = React.useState(0);

React.useEffect(() => {
  document.title = `You clicked ${count} times`;
}, [count]);
```

For effects that need refreshing whenever one or more variables change, just call useEffect with an array of the variables as second parameter.

</> https://codesandbox.io/s/component-lifecycle-i9r7w8?file=/src/pages/Count.js

Function component lifecycle

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

Mounting

useEffect with [ ]

Updating

useEffect with state(s) or prop(s)

Unmounting

return from useEffect

Error Handling

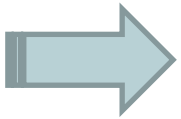
Error Boundary

```
state = { width: 0 };

componentDidMount() {
  window.addEventListener("resize", this.onResize);
}

componentWillUnmount() {
  window.removeEventListener("resize", this.onResize);
}

onResize = () => {
  this.setState({ width: window.innerWidth });
};
```



```
const [width, setWidth] = React.useState(0);

function onResize() {
  setWidth(window.innerWidth);
}

React.useEffect(() => {
  window.addEventListener("resize", onResize);
  return () => window.removeEventListener("resize", onResize);
}, []);
```

To invoke a clean up function before unmounting, return a function from a useEffect call.

</> https://codesandbox.io/s/component-lifecycle-i9r7w8?file=/src/pages/Unmounting.js

Function component lifecycle

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

Mounting

useEffect with [ ]

Updating

useEffect with state(s) or prop(s)

Unmounting

return from useEffect

Error Handling

Error Boundary

From React documentation: “There are no direct equivalent hooks for error handling in function components yet”.

You can either create an Error Boundary class component or use some external package (like the react-error-boundary package).

React error handling

https://reactjs.org/docs/error-boundaries.html

```
export default class ErrorBoundary extends React.Component {
  state = { hasError: false };

  static getDerivedStateFromError(error) {
    return { hasError: true };
  }

  componentDidCatch(error, info) {
    console.log(error, info.componentStack);
  }

  render() {
    if (this.state.hasError) {
      return <p>An error occurred</p>;
    }

    return this.props.children;
  }
}
```

A JavaScript error in a part of the UI shouldn't break the whole app.

You can use an Error Boundary as a wrapper to your components to catch errors during rendering, in a lifecycle method, or in the constructor.

Example:

</> https://codepen.io/andre-siggia/pen/YzjJQrL

Synthetic events

https://beta.reactjs.org/reference/react-dom/components/common#react-event-object

Also known as “React event object”, it is a wrapper over the native browser events to smooth out cross-browser differences.

The original event is also sent in the “nativeEvent” property.

```
<form onSubmit={(e) => e.preventDefault()}>
  <button type="submit">Send</button>
</form>
```

Synthetic events

https://beta.reactjs.org/reference/react-dom/components/common#react-event-object

Also known as “React event object”, it is a wrapper over the native browser events to smooth out cross-browser differences.

The original event is also sent in the “nativeEvent” property.

```
<form onSubmit={(e) => e.preventDefault()}>
  <button type="submit">Send</button>
</form>
```

```
interface BaseSyntheticEvent<E = object, C = any, T = any> {
    nativeEvent: E;
    currentTarget: C;
    target: T;
    bubbles: boolean;
    cancelable: boolean;
    defaultPrevented: boolean;
    eventPhase: number;
    isTrusted: boolean;
    preventDefault(): void;
    isDefaultPrevented(): boolean;
    stopPropagation(): void;
    isPropagationStopped(): boolean;
    persist(): void;
    timeStamp: number;
    type: string;
}
```

https://github.com/DefinitelyTyped/DefinitelyTyped/blob/master/types/react/index.d.ts#L1185

Working with forms

https://reactjs.org/docs/forms.html

HTML form elements work a bit differently from other DOM elements in React, because form elements naturally keep some internal state.

For example, inputs keep its “value” stored.

```
<input placeholder="Name" />
```

Working with forms

https://reactjs.org/docs/forms.html

Uncontrolled

```
<input ref={this.nameRef} placeholder="Name" />
```

A form element can be used as “uncontrolled”, where the value is stored in the DOM. To retrieve it, you can use a ref.

You can use the “defaultValue” prop to set an initial value to the form element.

Working with forms

https://reactjs.org/docs/forms.html

Uncontrolled

```
<input ref={this.nameRef} placeholder="Name" />
```

A form element can be used as “uncontrolled”, where the value is stored in the DOM. To retrieve it, you can use a ref.

You can use the “defaultValue” prop to set an initial value to the form element.

Controlled (most common)

```
<input
  value={this.state.name}
  onChange={(e) => {
    this.setState({
      team: e.target.value
    });
  }}
  placeholder="Name"
/>
```

A form element becomes "controlled" if you set its value via a prop.

You must then either set it as “read only” (using the readOnly prop) or listen to the onChange method for changes.

</> https://codesandbox.io/s/working-with-forms-pvprpw?file=/src/ControlledForm.js

</> https://codesandbox.io/s/working-with-forms-pvprpw?file=/src/UncontrolledForm.js

Homework

</> https://github.com/codescreen/Evolution_Bootcamp_TypeScript_Bootcamp/tree/main/React_Typescript_1-3

Additional materials

React Hooks: Lifecycle Diagram
https://medium.com/@galmargalit/react-function-components-hooks-lifecycle-diagram-14f76e0a5988

The React Lifecycle, step by step [class component]
https://medium.com/@vmarchesin/the-react-lifecycle-step-by-step-47c0db0bfe73

Everything about event bubbling/capturing:
https://transang.me/everything-about-event-bubbling/

Properly using bind in React
https://medium.com/webmonkeys/properly-using-bind-in-react-2e5c7e62bdb8

Controlled and uncontrolled form inputs in React don't have to be complicated
https://goshacmd.com/controlled-vs-uncontrolled-inputs-react/

React Strict Mode
https://beta.reactjs.org/reference/react/StrictMode
https://www.heissenberger.at/en/blog/react-components-reder-twice/

Thank you!