CSE 331
Software Design & Implementation

Winter 2021

Section 8 – HW8: React Canvas + Common Bugs

Administrivia

- HW7 due tonight, 11 pm (+ late day if you have & need one)
  - Any last-minute questions?

- HW8 (connect the dots) out shortly – React/TS warmup project, completely independent of previous hw assignments
  - Due next Thursday night

- Lots of demo code from yesterday's lectures and today's sections on the web (will finish lecture examples tomorrow)
  - Find links on the course calendar for yesterday and today
  - Download and run the code. Experiment and try things – you will have a much better feel for how everything works if you play with it (slides and reading alone likely aren't enough)

Agenda

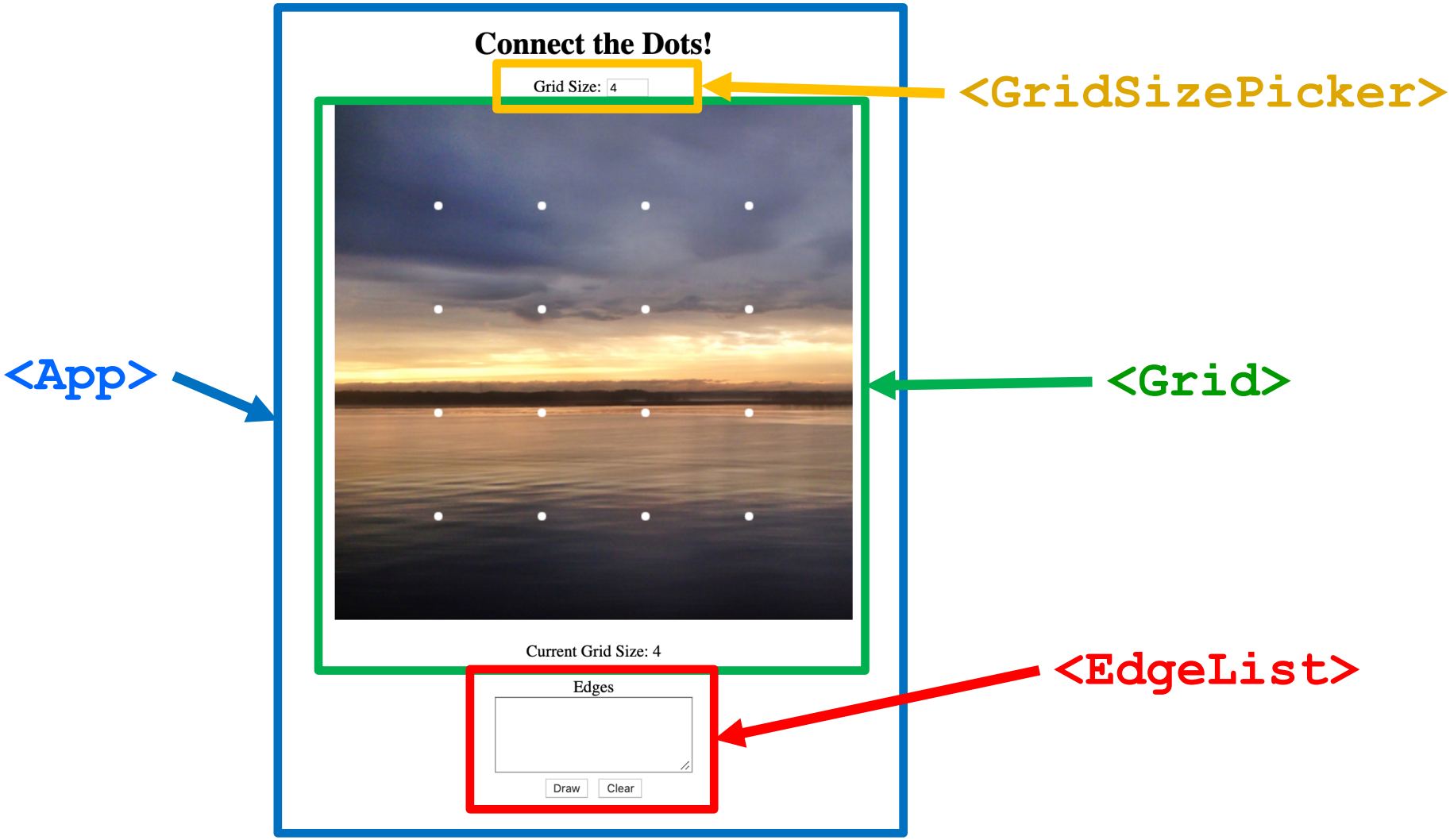
- Questions?
- Overview of HW8 – “Connect the Dots”
- Common React bugs & how to fix them
- Debugging Tools + Tips

HW8

HW8 Overview

- Starter code has (most of) the pieces, but not much functionality.
  - Lots of hard-coded values, placeholders (console.log instead of actually doing stuff), etc..
- Your job: "wire all the pieces together"
  - Accept user input
  - Process/parse the data
  - Error check – users do weird stuff, make sure you can't crash
  - Move data between components as necessary
  - Add the actual functionality in response to user input.
- Structure:
  - Top-level <App> component, with three child components.

HW8 Component Structure



Running a React App

`npm`: Similar to gradle, but we need to install manually the first time.

In the terminal, change directory until you're in the same place as the "**package.json**" file for the project you want to run.

To Install (first time): **`npm install`**

To Run (every time): **`npm start`**

Once started, you can edit and save files and the page will automatically reload – no need to restart. Use Control-C to shut down when you're done developing.

REACT BUGS

Common React Bugs

- Most common bugs in React are:
  - Reading from React state before the data has been populated.
  - Not properly understanding the React life cycle (the order that things happen within your app).

- This is because of specific **asynchronous updates** to React's internal representation of the webpage.
  - **Note:** There may be a very slight delay to updating your React components.

- **IMPORTANT:** You need to be careful when updating your React component's state and trying to access data!

Debugging React Strategies

When you hit a bug…

1. Walk step by step through the order that your code runs, checking how the state should be populated.
   - Use documentation about the React lifecycle to help you figure out which things happen in which order.

2. Put a `console.log()` in your methods if needed, and in `componentDidUpdate()` to check when your state was updated.

3. **Last resort:** Googling may be useful! Be ***very*** careful about this.

Section Demo Code

`package.json`	Tells `npm` what to install (don't need to modify)
`README`	Read it, has lots of explanations.
`public/index.html`	React "starter template"
`src/index.tsx`	Contains the `ReactDOM.render` call
`src/1-lifecycle/*`	First example, 1 buggy and 1 fixed versions
`src/2-state/*`	Second example, 2 buggy and 1 fixed version
`src/3-desync/*`	Third example, 1 buggy and 1 fixed version

Modify the import statement in **`src/index.tsx`** to change which demo you're currently running. (See **`README`** for more info).

Bug 1 – “Read before Write”

Expected Functionality:

- Adds a canvas to the page and displays a blue rectangle immediately.

Current Functionality:

- **`TypeError`** when the page is loaded.

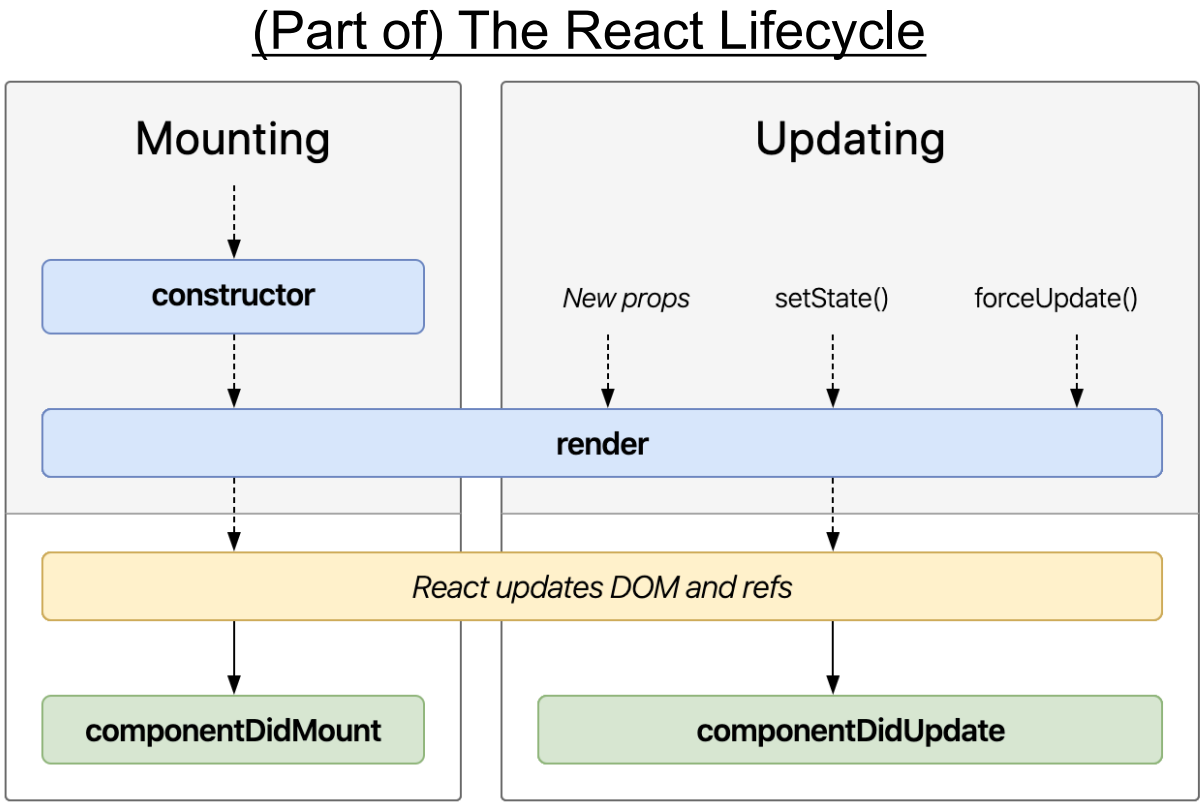
Bug 1 –The Problem

It seems like `this.canvasRef.current` is null, when it's supposed to be our Canvas object.

- Why doesn't the Canvas object exist yet? Let's think about how the `<canvas>` is eventually inserted into the page...

1. Our component is created and inserted into the page (in this case by `ReactDOM.render()`)
2. React constructs the component and then calls the component's `render()` to get the HTML tags we want.
3. React inserts those tags into the webpage **and then** sets up all the reference objects.

Bug 1 – "Read before Write"



- Order that React calls methods.
- We're accessing the reference during the constructor
- React doesn't update the refs (yellow box) until after `render()` – so they don't exist when the constructor is running!

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

Bug 1 – The Fix

(Part of) The React Lifecycle

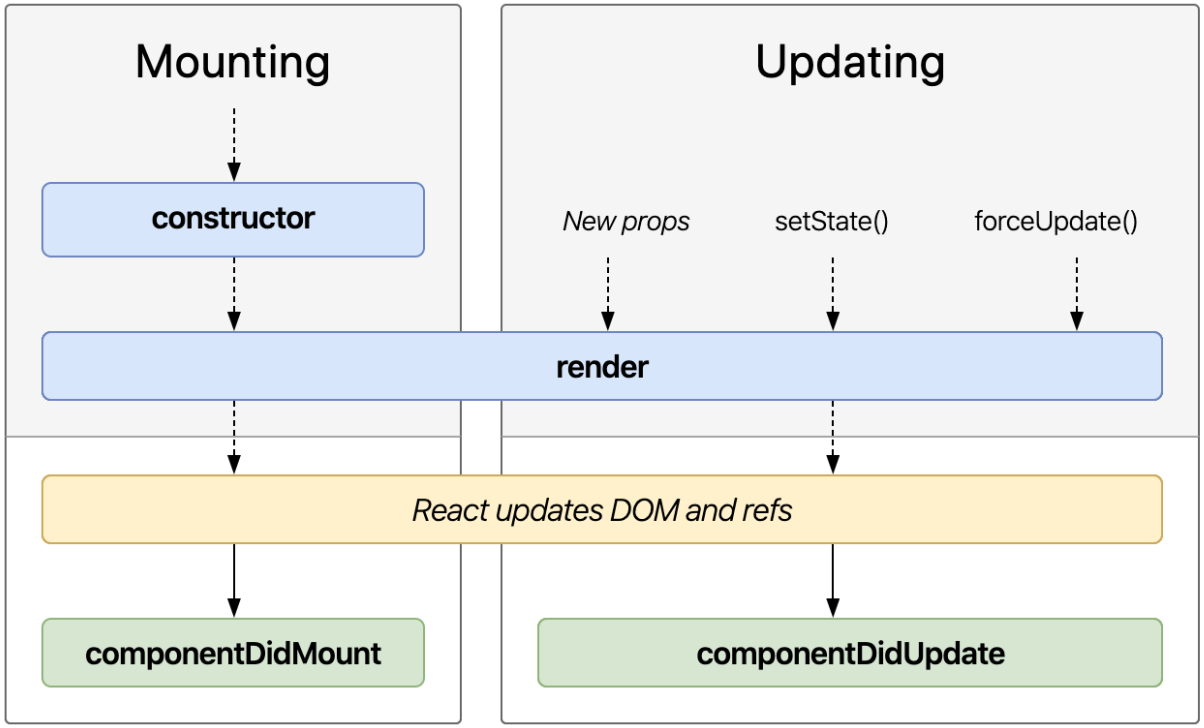


Image: http://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/

Solution

Override `componentDidMount`: called when React is done inserting all the DOM nodes and updating refs.

In `componentDidMount`, we know it's safe to use the ref (the "read"), since it's guaranteed to happen after the updating the refs (the "write") has finished.

Bug 1 – The Fix

- Move the `updateCanvasImage` call into `componentDidMount`
  - Still called during the component "mounting" phase – so we're able to set up the "first look" of the canvas like we wanted.
  - Happens after React sets up our refs, so we know we'll have a valid Canvas object to work with.

- Common idea in React:
  - Set something up (like the <canvas> tag) and give it to React (by returning from `render`)
  - Some time later, React will do its job.
  - React makes a *callback* (like `componentDidMount`) to let us know that it's done and we can use whatever we set up (like accessing the Canvas through its ref).

Bug 2 – “React Doesn’t Know”

Expected Functionality:

- When the button is clicked, the message on the page changes to "I've been clicked!"

Current Functionality:

- The message on the page never changes.
- We know that the button event is working because the `console.log()` inside the listener is being run, so the bug must be somewhere else.

Bug 2 – The Problem

- The `this.clicked` variable is being updated correctly
  - You can print it out to double check, if you'd like.

- The only place we can modify what text is being put in the `<p>` is during the render() method – we need to return a different `<p>` element to change what's on the page.
  - But React doesn't know it's supposed to call `render` again!
  - More accurately: React doesn't know that the contents of the `this.clicked` variable matters for `render`.

Bug 2 – The Solution?

- React has a special place for variables that affect how a component renders: `this.state`.
  - Store an object (`{...}`) inside `this.state`, put whatever properties we want in that object to track the data we need.
  - Instead of `this.clicked`, we write `this.state.clicked`
- "`clicked`" isn't a special name here – just a variable name. Could easily call it "`this.state.pizza`"
- "`state`" **is** a special name: part of the React convention, React code expects that you store your state variables in `this.state`

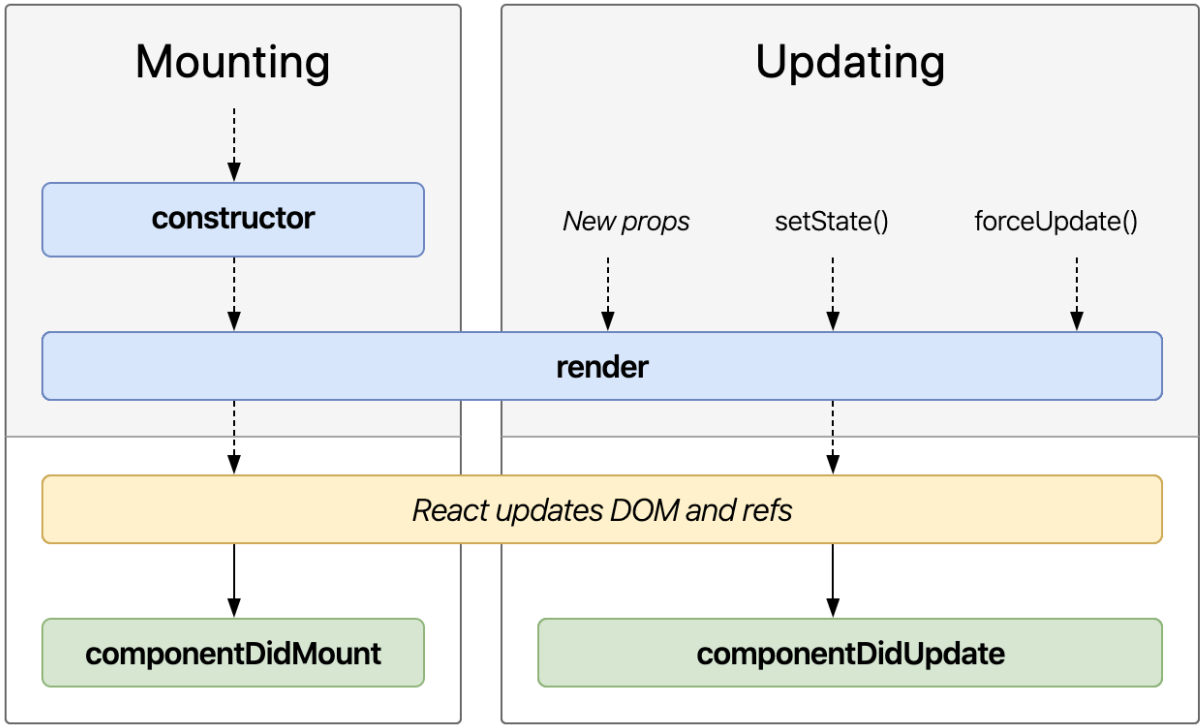
- Not quite a fix yet, but a step in the right direction.

Bug 2 – The *New* Problem

- We're now storing our data in the right place, but we still aren't telling React when we change it.
  - React requires that you notify it when you *want* to change the data, *instead* of changing it yourself.
  - This is why we get a TypeScript compiler error when we try to change it manually.
- To request a state change, call `this.setState` and pass it an object representing *the changes you want to make*.
  - You should never directly modify the contents of `this.state` (except for constructor initialization). (Impossible with TS).
- Since you use `this.setState` (which is React code) to update the state, React knows that you'll need things to be updated based on what changed. (So, React will re-do the render).

Bug 2 – The Fix

(Part of) The React Lifecycle



Solution

By calling `setState` to update our state, we trigger a "component update cycle". During an update, React will change the state and then re-call `render()`. In `render()`, we can return the new text to be displayed on the page.

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

Bug 3 – "Read before Write (is done)"

Expected Functionality:

- When a button is clicked, a square of that color appears in the canvas.
- The current color is displayed in the text above the buttons.

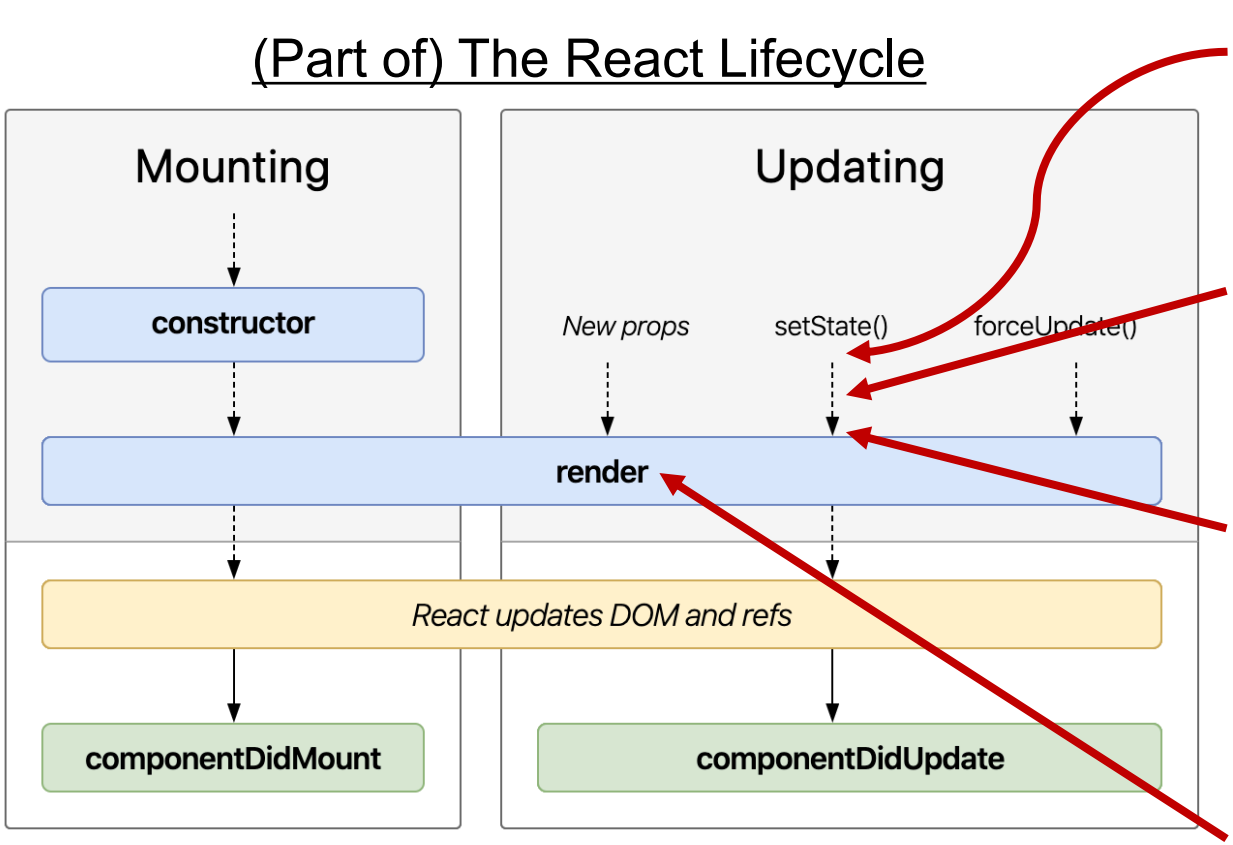
Current Functionality:

- The text above the buttons seems to be working correctly.
- The canvas is lagging behind one click – displays the color from two clicks ago instead of the most recent click.

Bug 3 – The Problem

- Remember that, in React, setState is a *request* for a *future change* to state. **When setState returns, the state has not yet been updated**.
  - React delays state changes for performance reasons.
  - Means we need to be careful about reading state: when do we know that it's guaranteed to be up-to-date?
- The problem is that we're trying to access the state immediately after calling setState – React hasn't gotten around to updating the state yet, so we're seeing the old value.
  - This is why the canvas is "lagging behind" by one: when we draw the canvas, we're seeing the value of state from the previous button press.

Bug 3 – What's Actually Happening



`setState` returns, our code keeps going

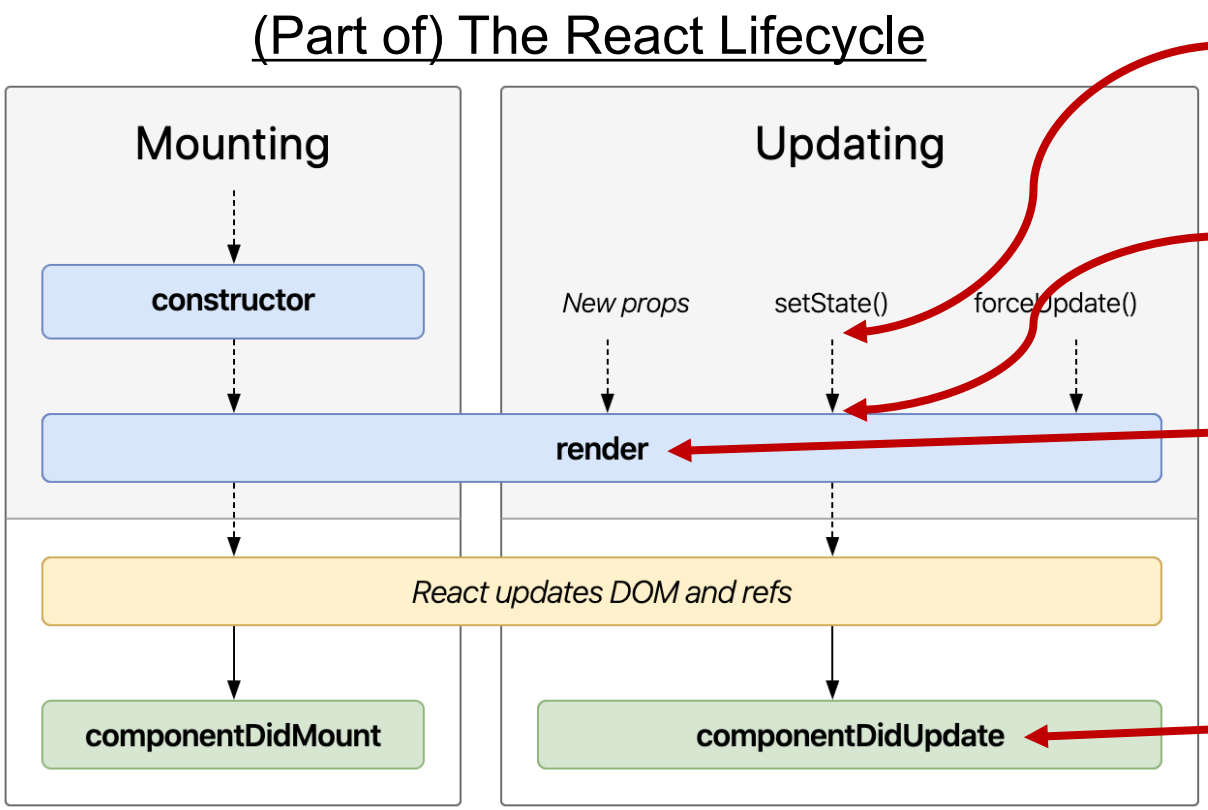
Our code calls `this.drawSquare`, which sees the old value still in `this.state`

`this.state` gets changed later (by React)

The `<p>` gets updated here, which is why it sees the correct state (state is guaranteed to be updated before render is called)

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

Bug 3 – The Fix



`setState` returns, our code keeps going

`this.state` gets changed later (by React)

`<p>` gets updated here, so it sees the correct state

Solution

Should call `drawSquare`, here, since it is guaranteed to happen after the `this.state` value has been updated.

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

DEBUGGING TIPS + TOOLS

Chrome Developer Tools

- View > Developer > Developer Tools
  - or: Right Click on Page > Inspect

- You can:
  - See the browser console (output of console.log)
  - See the HTML being used to display the page

- Pro Tip: When using console.log to print an object/array/value, pass it to console.log without appending any strings.
  - Gives you interactive output in the console!

```
const obj = {school: "UW", colors: ["purple", "gold"]};
console.log(obj)  // Not console.log("school: " + obj);
```

React Developer Tools

- Installed as Chrome extension (link in the HW8 spec)
- Adds new React-specific tabs to the top of Chrome Developer Tools window

- You can:
  - See the tree of your components
  - See the props + state in each component
  - Change state in components and watch components React to changes

- Also: React Developer Tools will look for common bugs in your code & warn you about them in the console