Building User Interfaces

React 3

Component Lifecycle

Professor Yuhang Zhao

Logistics

- Mid-term
  - March 11, Friday 7:15-8:45 pm
  - Remote exam on Zoom
  - An alternative session is available
- Quizzes
  - We will create a shared google document with all prior quizzes, answers, and explanations.

What we will learn today?

- The component lifecycle
- State update methods

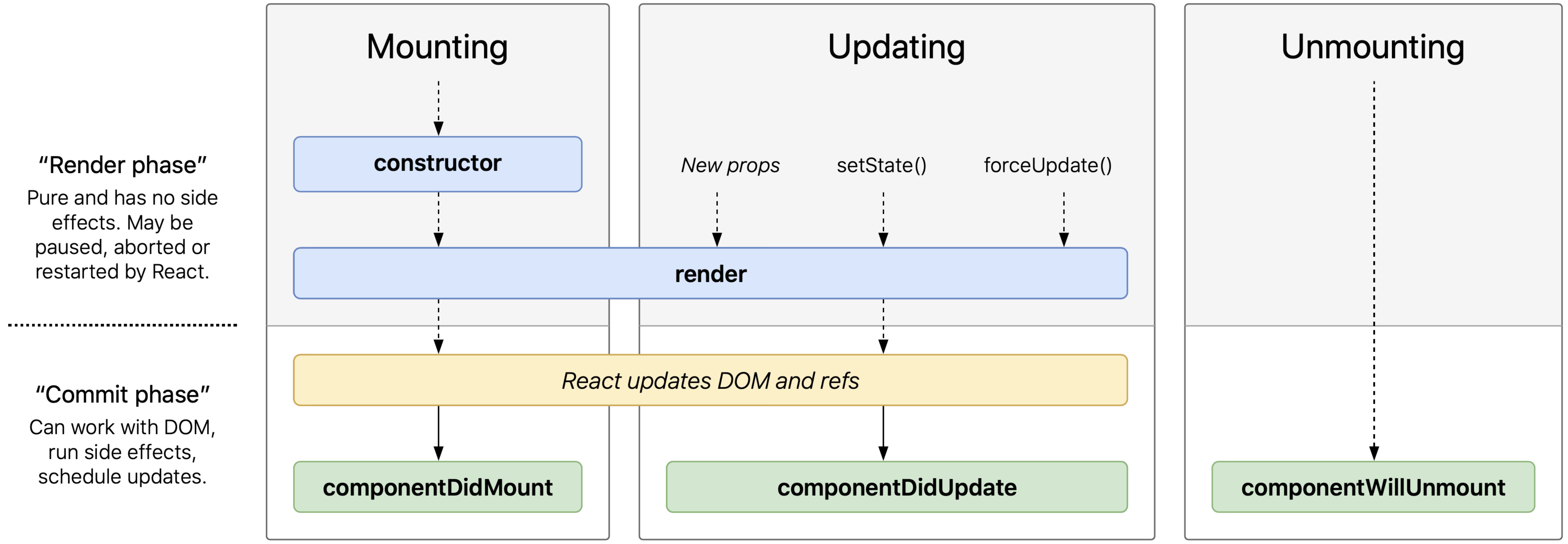
The Component Lifecycle

The Component Lifecycle

Definition: : The lifecycle of a React *component* can be defined as the series of methods that are invoked in different stages of the component's existence. There are four stages:

- Initialization
- Mounting
- Updating
- Unmounting

The Component Lifecycle[1]



[1] Wojciech Maj

Detour: What are side effects?

Definition: *Side effects* include anything that affects something outside the scope of the executed function, such as API requests (e.g., REST API).

Pure functions, e.g., `constructor()`, `render()`, have no side effects.

The Component Lifecycle[2] [3] is made up of three actions:

1. **Mounting**
2. **Updating**
3. **Unmounting**

Each action has a number of *lifecycle methods* associated with it in *render* and *commit* phases.

We will use a StackBlitz to illustrate all three actions.[4]

[2] ReactJS.org: React.Component

[3] The (new) React lifecycle methods in plain, approachable language

[4] See on StackBlitz

Mounting

Definition: *Mounting* is the process of creating an instance of a component and inserting it into the DOM.

Commonly used mounting lifecycle methods:

1. `constructor()`
2. `render()`
3. `componentDidMount()`

Mounting: `render()`

`render()` is the only required method within a class component, reading `this.props` and `this.state` and returning:

- **React elements**, adding a single element to the container
- **Arrays & fragments**, rendering multiple elements
- **Portals**, adding children to a DOM subtree
- **String & numbers**, rendering text nodes in the container
- **Booleans | null**, rendering nothing

```
return test && <Child />;
```

Mounting: `render()`, ***continued***

`render()` has to remain *pure*, executing exactly the same way every time:

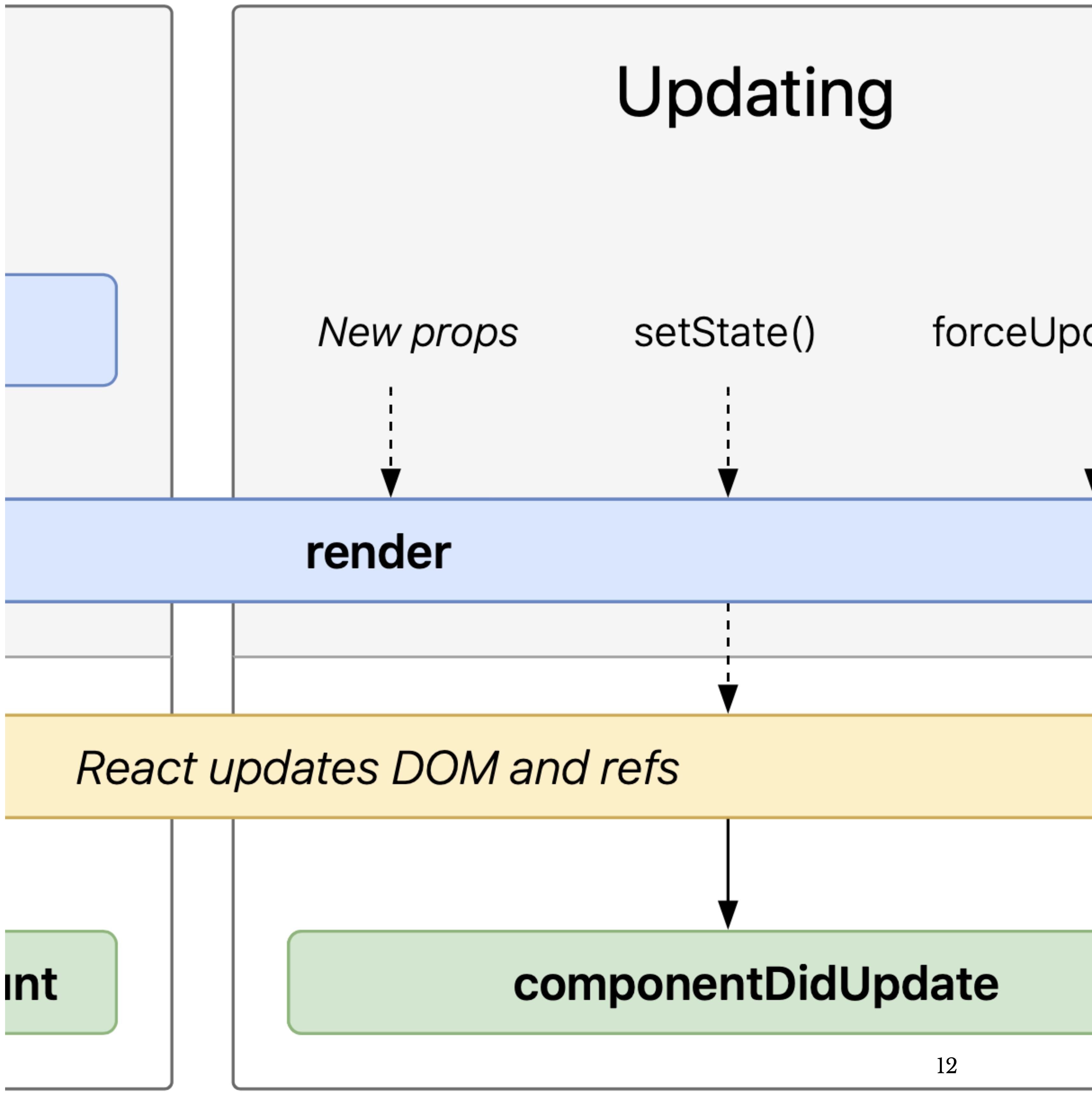
- no `state` updates are allowed within `render()`
- `render()` does not interact with the browser

Interactions with the browser should happen in other lifecycle methods.

Mounting: `render()`, ***continued***

What will happen if `setState()` is called in `render()`?

— Infinite loop > Stack overflow



Mounting: `constructor()`

`constructor()` is only needed to inherit `props`, to initialize `state`, and to bind event-handling functions.

`super(props)` should be called before any other statement, and all other statements should come after it.

`constructor()` is the only place where we should directly assign state using `this.state = { key: value }`, and `this.setState()` method should not be used here.

Mounting: `constructor()`, ***continued***

```
constructor(props) {
    // inherit props
    super(props);
    // set states
    this.state = { key: 'value' };
    // bind event-handling functions
    this.handleClick = this.handleClick.bind(this);
}
```

Mounting: `componentDidMount()`

`componentDidMount()` is automatically called as soon as the component is mounted following `render()`.

This give us an opportunity to do anything we did not want to do in `render()`, e.g., to initiate API calls, request data, etc, before the browser is updated.

Pro Tip: Unlike in `render()`, `setState()` method can be used in `componentDidMount()`. `setState()` will trigger a re-render before the browser reflects the update. State updates here should be used sparingly (e.g., to determine where a tooltip should be rendered) to maintain performance.

Updating

Definition: *Updating* involves re-rendering a component following changes to `props` or `state`.

Commonly used updating lifecycle methods:

1. `render()`
2. `componentDidUpdate()`

Updating: `componentDidUpdate()`

`componentDidUpdate(prevProps, prevState, snapshot)` is invoked as soon as there is an update.

Again, this is an opportunity to do anything we do not want to do within `render()`, e.g., placing network requests.

```
componentDidUpdate(prevProps) {
 if (this.props.userName !== prevProps.userName) {
   this.fetchData(this.props.userName);
 }
}
```

Unmounting

Definition: *Unmounting* involves removing a component from the DOM.

Unmounting lifecycle method:

1. `componentWillUnmount()`

Unmounting: `componentWillUnmount()`

`componentWillUnmount()` is invoked immediately before a component is unmounted — an opportunity to perform any necessary cleanup, e.g., resetting counters, invalidating timers, canceling network requests.

`setState()` method should not be called within `componentWillUnmount()` as it will never be rendered.

Key considerations in using `state`

Why is `state` so important?

Remember that a `state` update is how React knows that a component needs to be re-rendered. Once an application is loaded, all React does is to monitor changes to `state` and re-render components based on the changes.

How `state` should be updated

State should not be modified directly. The following will not re-render the component:

```
this.state.TAName = 'Andy';
```

We must use `setState()`:

```
this.setState({TAName: 'Andy'});
```

Considering asynchronous updates

Because React may batch-process state updates to improve performance, `state` and `props` may be updated asynchronously.

Because updates may be asynchronous, subsequent attempts to access the `state` may not provide updated information.

```
this.setState({ counter: this.state.counter + 1 });
console.log(this.state.counter);
```

May not increment, especially when you attempt to increment an item quantity more than once in the same cycle, use the `updater` argument

`this.setState(updater, callback)` will ensure the increment:

```
this.setState((state, props) => ({
  counter: state.counter + props.increment
}));
```

Complex state manipulations[5] [6]

— Adding to and removing from arrays
— State updates from children

[5] See in solutions CodePen

[6] A good article on managing `state` with arrays

3 Quizzes

Complete the Canvas quiz.

What did we learn today?

- The component lifecycle
- State update methods