# Component Lifecycle

***CS568 – Web Application Development I***

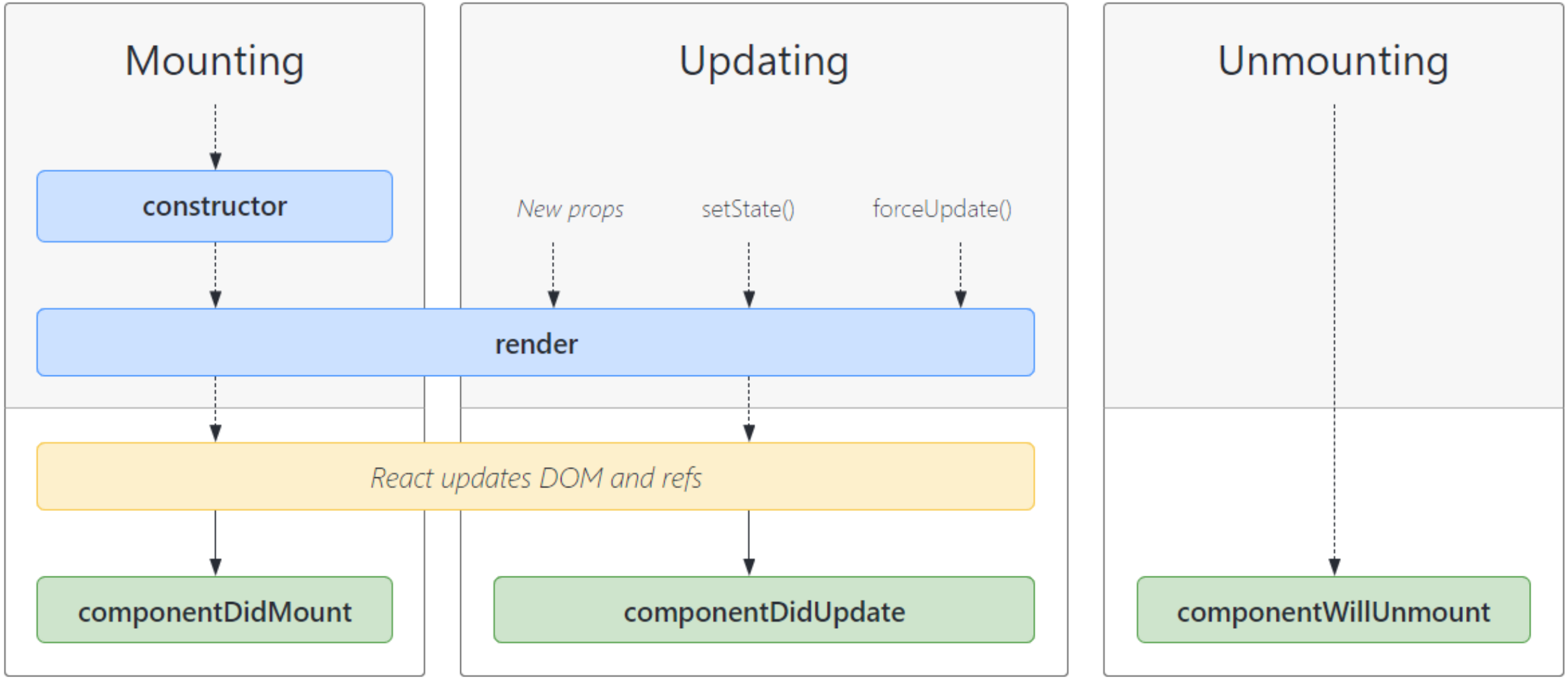
***Computer Science Department***

***Maharishi International University***

# Maharishi International University - Fairfield, Iowa

# Content

- Component lifecycle phases
  - Render phase
  - Commit phase
- Lifecycle methods
  - **ComponentDidMount**
  - ComponentDidUpdate
  - ComponentWillUnmount
  - And other.




Refer: [Common lifecycle methods]

# Component lifecycle phases

1. Render phase – Pure and has no side effects. Can be called multiple times by React. Must be stable and return the same result.
   - Render
2. Commit phase – Run side effects. Will be called once.
   - ComponentDidMount
   - ComponentDidUpdate
   - ComponentWillUnmount

Pure function – the function return values are identical for identical arguments e.g, sum(1,2) always returns 3 no matter how many times you called it.

# Side effects

Side effects are an action that impinges on the outside world. Examples:

- Making API calls for data
- Updating global variables from inside a function
- Working with timers like setInterval or setTimeout
- Setting or getting values in local storage
- Measuring the width or height or position of elements in the DOM
- Logging messages to the console or other service

# componentDidMount

- This is invoked immediately after a component is mounted (inserted into the DOM tree).
- Here is the right place to initialize the DOM node. For example, **load data from the back-end service**.
- Here is the right place to set up any subscriptions (don’t forget to unsubscribe in componentWillUnmount())
- You may call setState() immediately in componentDidMount() (Most times, developers do that!). It will trigger an **extra rendering**, but it will happen before the browser updates the screen. This guarantees that even though the render() will be called twice in this case, **the user won’t see the intermediate state**. Use this pattern with caution because it often causes performance issues.

# componentDidUpdate

- componentDidUpdate() is invoked immediately after updating occurs. This method is not called for the initial render.
- You may call setState() immediately in componentDidUpdate() but note that it must be wrapped **in a condition** like in the example below, or you'll cause an infinite loop.
- Don't copy props into state that causes bugs! Directly use props.

```
componentDidUpdate(prevProps, prevState) {
  // Typical usage (don't forget to compare props):
  if (this.props.userID !== prevProps.userID) {
    this.fetchData(this.props.userID);
  }
}
```

# componentWillUnmount

- componentWillUnmount() is invoked immediately before a component is unmounted and destroyed.
- Perform any necessary **cleanup** in this method, such as invalidating timers, canceling network requests, or cleaning up any subscriptions that were created in componentDidMount().
- You should not call setState() in componentWillUnmount() because the component will never be re-rendered.

# Component Lifecycle - Creation

Invoking Order

| | |
|---|---|
| constructor(props) | Call super(props) |
| render | Prepare JSX code |
| componentDidMount | Can be used for making HTTP requests |

# Component Lifecycle - Update (for Props Changes)

Invoking Order

For performance optimization

shouldComponentUpdate(nextProps,nextState)

render

componentDidUpdate

May cancel update process

Last minute DOM operations

# shouldComponentUpdate for Optimization

- Since ‘students’ component is a child element of App component when something is changed in App, students component is re-rendered.
- How can we avoid this?
- render() will not be invoked if shouldComponentUpdate() returns false!

# shouldComponentUpdate for Optimization

```
//students.js
shouldComponentUpdate(nextProps,nextState){
        if(nextProps.students !== this.props.students){
            return true; //re-render
        }
        return false; //do not re-render
    }
```

# Pure Components

- Instead of implementing shouldComponentUpdate method we can extend to PureComponent**.**
- Pure components prevent from re-rendering if props or state is the same.
- **It takes care of “shouldComponentUpdate” implicitly.**
- Pure Components are more performant.
- State and Props are shallowly compared.
- Shallow comparison checks the reference (address) not the value.

# Unnecessary render

React re-renders when there is change in state and props.

When the parent component renders, React will recursively render all its child components, regardless of whether their props have changed or not. In other words, child components go through the render phase but not the commit phase. It is also known as **unnecessary render**.

There are techniques to prevent from unnecessary renders. One of them is to implement the shouldComponentUpdate or extend to PureComponent.

# UseEffect hook

UseEffect is called every time component renders. You can control it so it acts as componentDidUpdate, componentDidMount, componentWillUnmount.

- useEffect(() => {…}); - similar to componentDidUpdate. Also includes componentDidMount.
- useEffect(() => {…}, **[]**); - similar to componentDidMount.
- useEffect(() => {… **return () => {…}** }, []); - similar to componentWillUnmount.

Refer: The effect hook.

# React.memo

- Similar to should component update for functional components. But only checks props.
- If your component renders the same result given the same props, you can use React.memo for performance optimization. That means React will skip rendering the component with the same prop.

*const MyComponent = React.memo(function MyComponent(props) {…});*

- React.memo shallowly compares props. You can overwrite the “areEqual” function for custom props comparison.