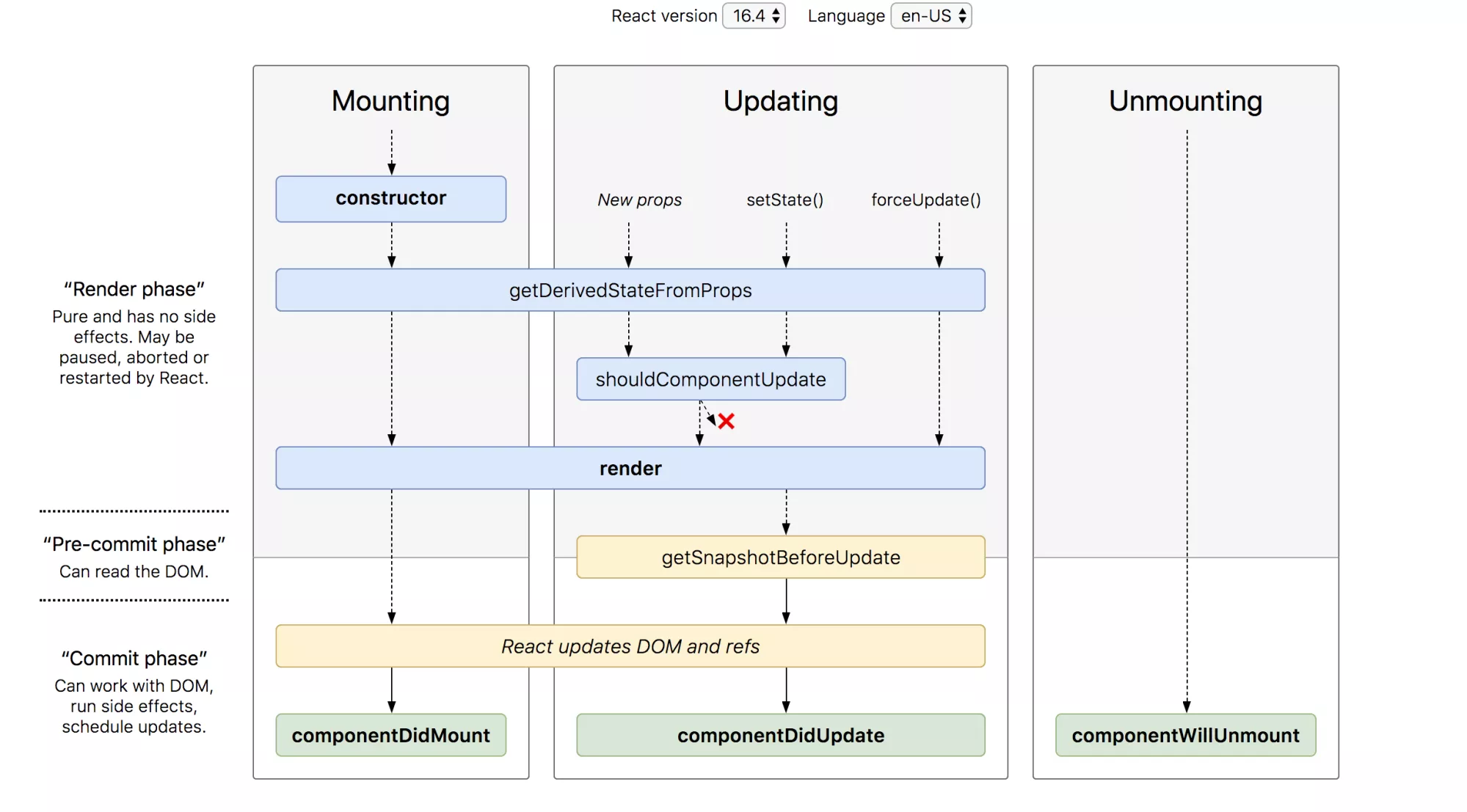
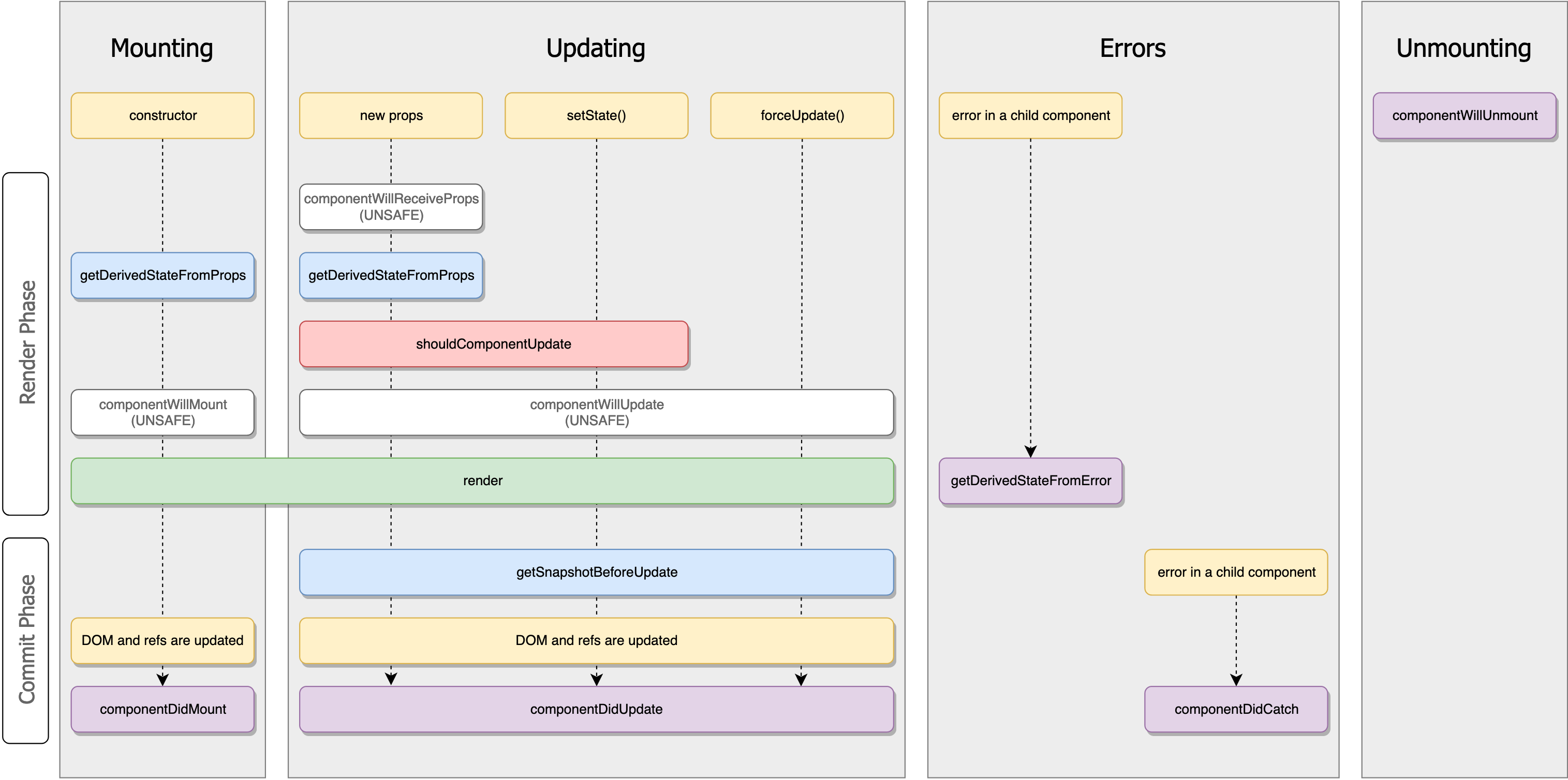
**Table of Contents**

Latest Life Cycle



LifeCycle with obselete methods -



# Life Cycle before 16.4

# 

# React LifeCycle Method

## Mounting

### defaultProps

defaultProps can be defined as a property on the component class itself, to set the default props for the class. This is used for undefined props, **but not for null props**. For example:

class CustomButton extends React.Component {

// ...

}

CustomButton.defaultProps = {

color: 'blue'

};

### getInitialState (OBSOLETE)

set intial state of the componet.

This is methos is replaced by constructor.

### Constructor

you should call super(props) before any other statement

Typically, in React constructors are only used for two purposes:

* Initializing [local state](https://reactjs.org/docs/state-and-lifecycle.html) by assigning an object to this.state.
  + You should not call setState() in the constructor(). Instead, if your component needs to use local state, assign the initial state to this.state directly in the constructor:
* Binding [event handler](https://reactjs.org/docs/handling-events.html) methods to an instance.

### Static getDerivedStateFromProps

getDerivedStateFromProps is invoked right before calling the render method, both on the initial mount and on subsequent updates. It should return an object to **update the state, or null** to update nothing.

[Make sure you’re familiar with simpler alternatives:](https://reactjs.org/blog/2018/06/07/you-probably-dont-need-derived-state.html)

* If you need to perform a side effect (for example, data fetching or an animation) in response to a change in props, use [componentDidUpdate](https://reactjs.org/docs/react-component.html" \l "componentdidupdate) lifecycle instead.
* If you want to re-compute some data only when a prop changes, [use a memoization helper instead](https://reactjs.org/blog/2018/06/07/you-probably-dont-need-derived-state.html" \l "what-about-memoization).
* If you want to “reset” some state when a prop changes, consider either making a component [fully controlled](https://reactjs.org/blog/2018/06/07/you-probably-dont-need-derived-state.html" \l "recommendation-fully-controlled-component) or [fully uncontrolled with a](https://reactjs.org/blog/2018/06/07/you-probably-dont-need-derived-state.html" \l "recommendation-fully-uncontrolled-component-with-a-key) key instead.

### ComponentWillMount (OBSOLETE)

This is method is replaced by getDerivedStateFromProps.

### Render()

### componentDidMount

Initialization that requires DOM nodes should go here. this is a good place to instantiate the **network request**.

This method is a good place to **set up any subscriptions**

You may call setState() immediately in componentDidMount(). 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.

## Updating

### componentWillReceiveProps (OBSOLETE)

This is method is replaced by getDerivedStateFromProps.

### GetDerivedStateFromProps

Explained in Mounting section.

### ShouldComponentUpdate

Use shouldComponentUpdate() to let React know if a component’s output is not affected by the current change in state or props. The default behavior is to re-render on every state change, and in the vast majority of cases you should rely on the default behavior.

shouldComponentUpdate() is invoked before rendering when new props or state are being received. Defaults to true. This method is **not called for the initial render or when forceUpdate() is used.**

### componentWillUpdate (OBSOLETE)

This is method is replaced by getDerivedStateFromProps.

### render()

### getSnapshotBeforeUpdate

is invoked right before the most recently rendered output is committed to e.g. the DOM. It enables your component to capture some information from the DOM (e.g. scroll position) before it is potentially changed.

### componentDidUpdate

## Errors

### geyDerivedStateFromError

This lifecycle is invoked after an error has been thrown by a descendant component. It receives the error that was thrown as a parameter and should return a value to update state.

### ComponentDidCatch

This lifecycle is invoked after an error has been thrown by a descendant component. It receives two parameters:

1. error - The error that was thrown.
2. info - An object with a componentStack key containing [information about which component threw the error](https://reactjs.org/docs/error-boundaries.html" \l "component-stack-traces).

## Unmounting

### componentWillUnmout

# Function and Class Components

The simplest way to define a component is to write a JavaScript function:

function Welcome(props) { return <h1>Hello, {props.name}</h1>;

}

const element = <Welcome name="Sara" />;

ReactDOM.render(

element,

document.getElementById('root')

);

You can also use an [ES6 class](https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Classes) to define a component:

class Welcome extends React.Component {

render() {

return <h1>Hello, {this.props.name}</h1>;

}

}

# React Hooks

## Basic

### useState

### The useState hook lets you add state to function components.

By calling React.useState inside a function component, you create a single piece of state associated with that component. (every hook starts with the word “use”; a call to useState literally lets you “use state” in a function component)

### useEffect

### useContext

### [Additional Hooks](https://reactjs.org/docs/hooks-reference.html" \l "additional-hooks)

1. [useReducer](https://reactjs.org/docs/hooks-reference.html" \l "usereducer)
2. [useCallback](https://reactjs.org/docs/hooks-reference.html" \l "usecallback)
3. [useMemo](https://reactjs.org/docs/hooks-reference.html" \l "usememo)
4. [useRef](https://reactjs.org/docs/hooks-reference.html" \l "useref)
5. [useImperativeHandle](https://reactjs.org/docs/hooks-reference.html" \l "useimperativehandle)
6. [useLayoutEffect](https://reactjs.org/docs/hooks-reference.html" \l "uselayouteffect)
7. [useDebugValue](https://reactjs.org/docs/hooks-reference.html" \l "usedebugvalue)

# Lists and Keys

taking care of List rendering and it’s keys. All lists should be render with unique keys otherwise there will be a warning message on console and misfunction is some cases.