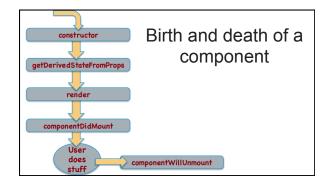
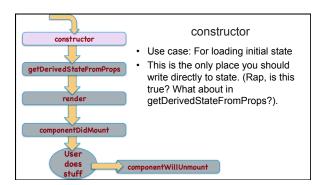


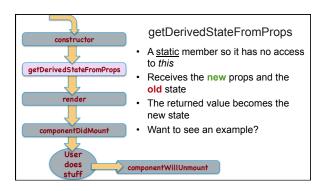
Lifecycle	events	fire	when	
-----------	--------	------	------	--

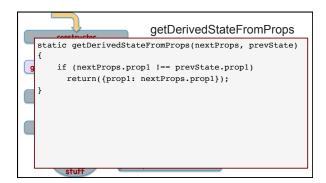
- 1. A component is born or dies
- 2. When state changes

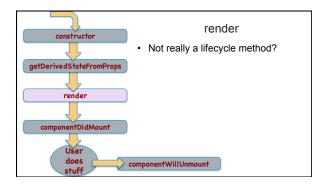
Component birth and death

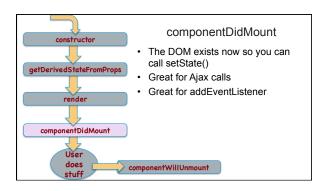


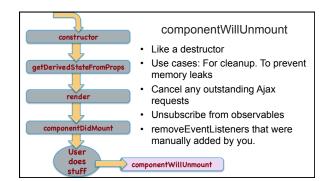




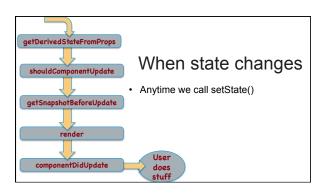


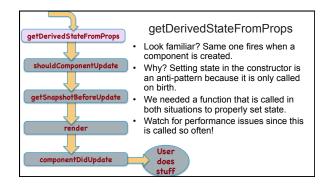


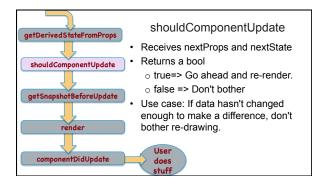




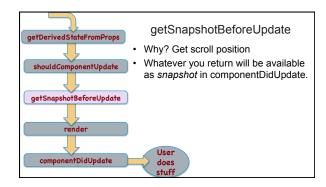
## When state changes

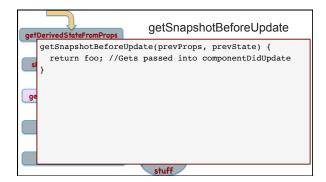


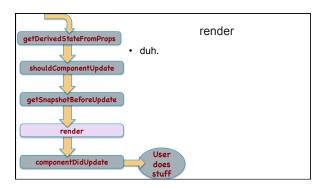


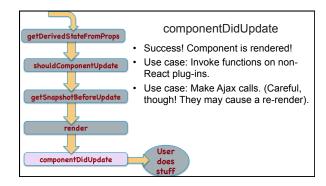


get	shouldComponentUpdate					
	<pre>shouldComponentUpdate(nextProps, nextState) {</pre>					
	if (state.val - nextState.val < smallNumber)					
sl	return false;					
	return true;					
ge	}					
3-						
stuff						









getDerivedStateFromProps componentDidUpdate					
ge	<pre>componentDidUpdate(prevProps, prevState, snapshot) {    //snapshot came from getSnapshotBeforeUpdate    if (this.props.foo === prevProps.foo)       console.log("foo has not changed"); }</pre>				
stuff					

- Use these sparingly
- They add cognitive friction to the component and they're often not necessary -- the component will do it anyway.
- Rap, put an example of no-op/default behaviors.

## tl;dr

- When you extend React.Component, you can override certain methods and they'll be called as your component ...
  - o is born or dies
  - o updates its internal state
- With them we have great control over setups, cleanups, updates, and more