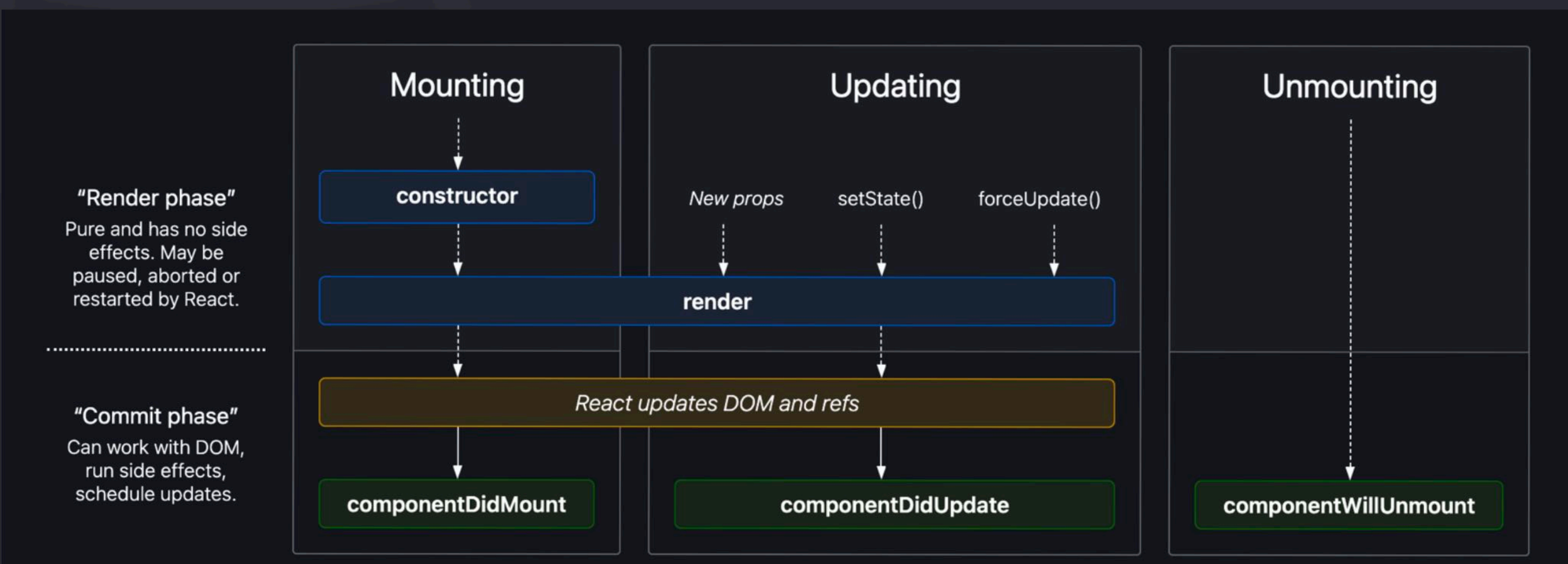
# Cheatsheet Lifecycle / Hooks




Source: projects.wojtekmaj.pl/react-lifecycle-methods-diagram/

## useEffect():

- passing no array: effects fire after every rerender
- passing empty array: effect fires only once (on mount), will not re-run, will get cleaned up (on unmount)
- passing variable(s): dependency array -> effect will fire if dependency variables change

A cleanup function will have to be returned inside of the useEffect (for example disconnect observers)

„Think of effects as an escape hatch from React's purely functional world into the imperative world." (Docs)

Management of side effects in functional components - like data fetch, setting up subscriptions, updating the DOM manually (whatever changes should happen on mount/update)
A side effect is if a function creates some kind of effect outside of its scope! (= impure function)

## useState():

- useState() is a hook, which allows storing a variable and a setter function for changing it (it uses array destructuring which allows setting values to variables inside of an array)
- every state variable that needs to be managed, has to get their own useState hook. Each hook only hooks into one value

## Functional Components:

- In functional components the whole function is going to be rerun every time a rerender happens - every time a statevalue (!) / prop changes
- this is the reason, why a fetch outside of a useEffect will cause an eternal rerender. The response from the fetch will be always different from the one in memory (even if the actual content is the same, it points to a different reference in memory), and if the state updates, it will also be different every time, hence the constant rerenders