React

A JAVASCRIPT LIBRARY FOR BUILDING USER INTERFACES

Table of Content

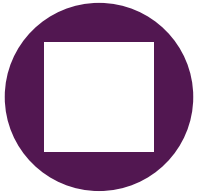
Introduction to JS frameworks, SPA, Webpack, Advantage of React.js over other frameworks

Real DOM vs Virtual DOM

Environment Setup, create-react-app cli

JSX, Props, State

Stateful vs Stateless Components

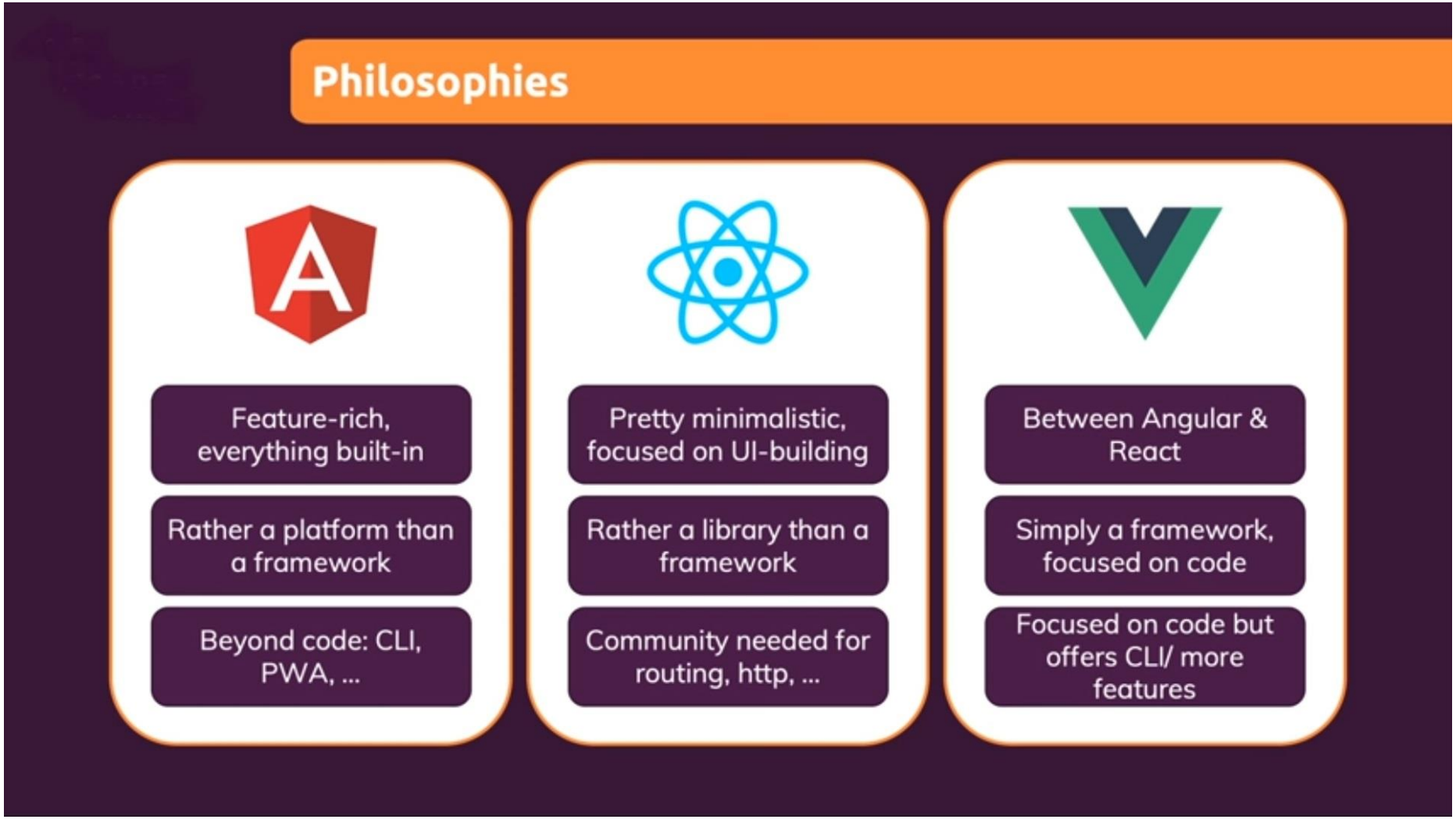


Refs & Keys

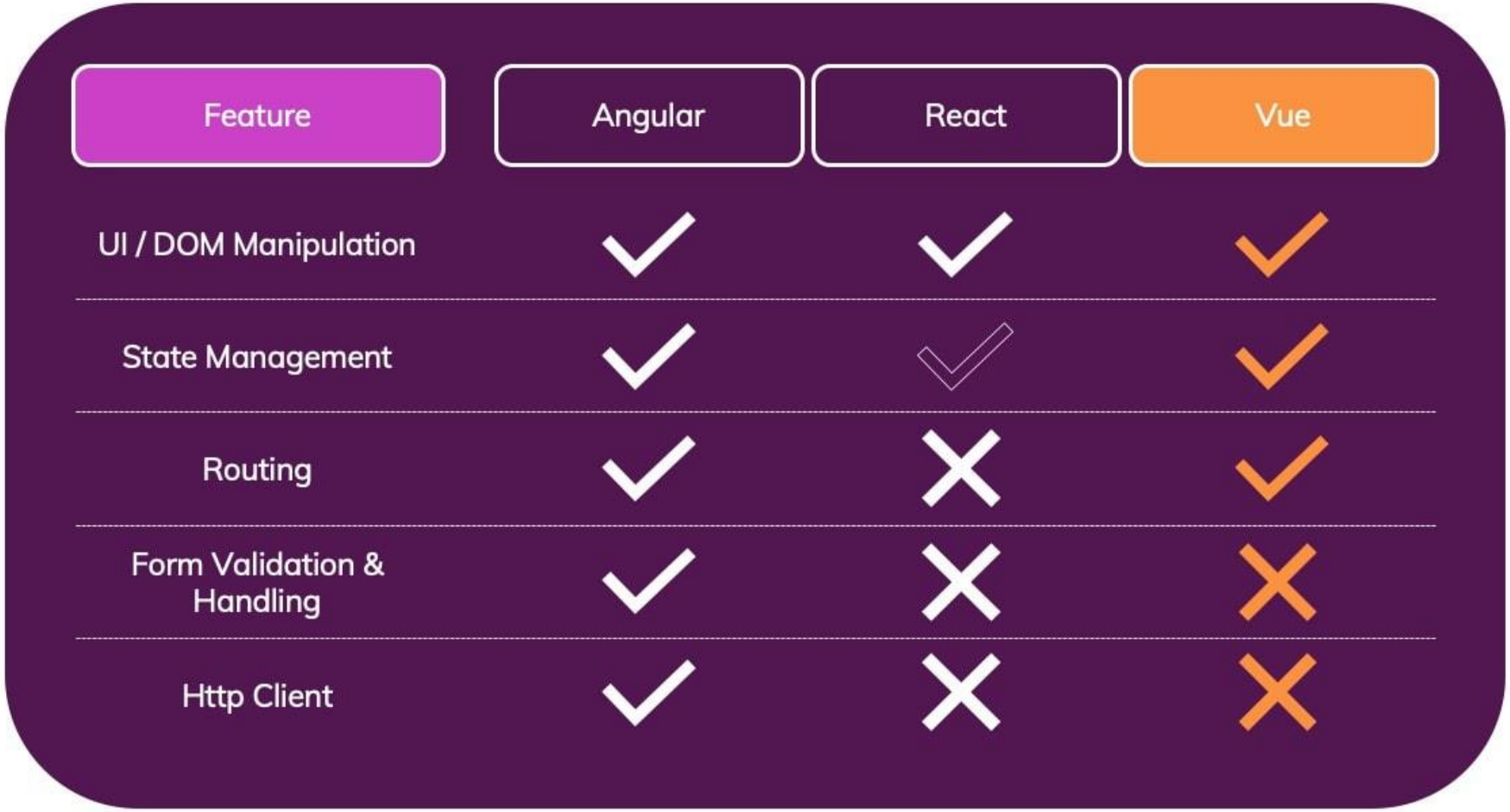
Component Lifecycle

Introduction to JS frameworks, SPA, Webpack & Advantage of React.js over other frameworks

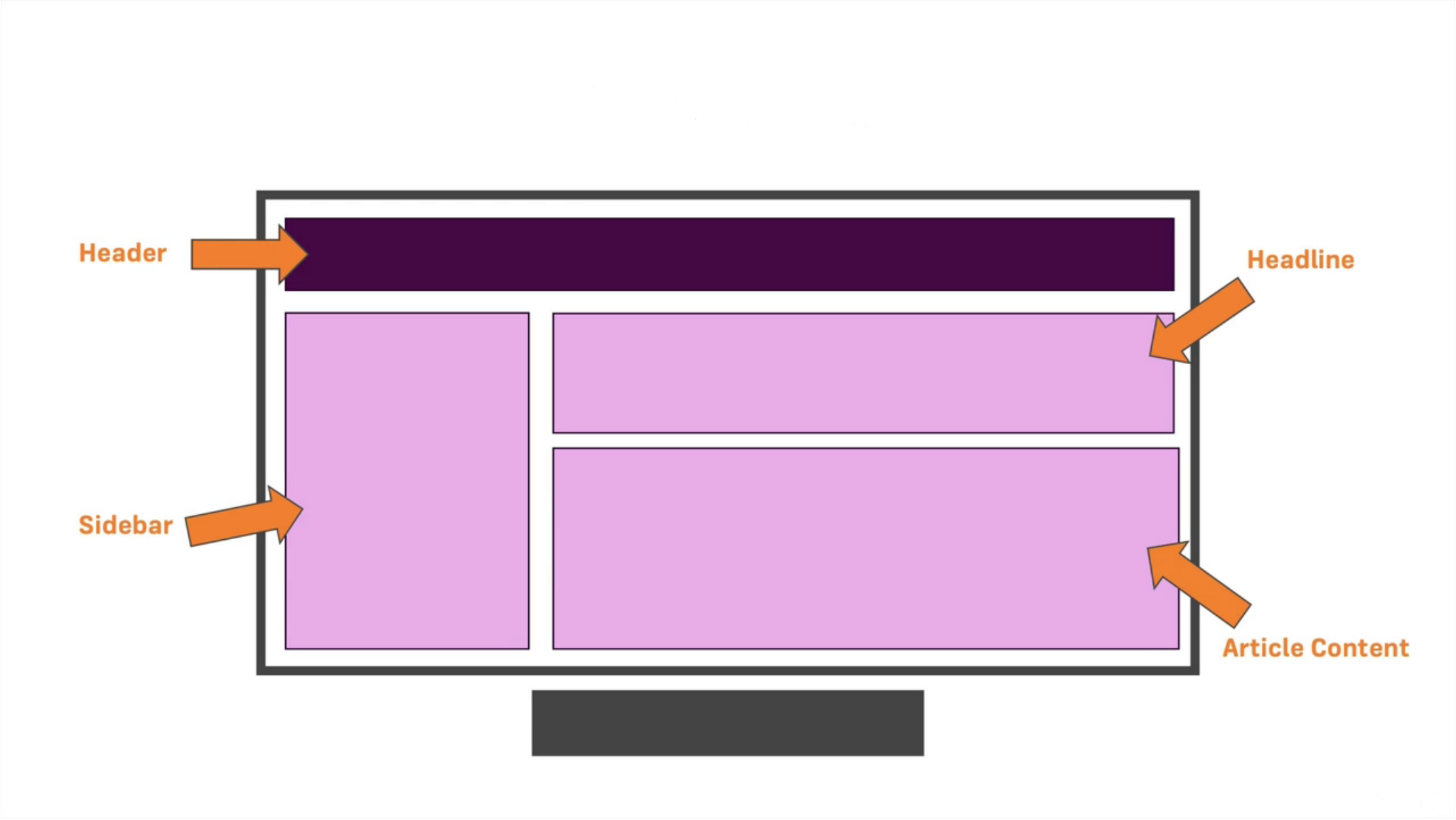
Popular JavaScript Frameworks



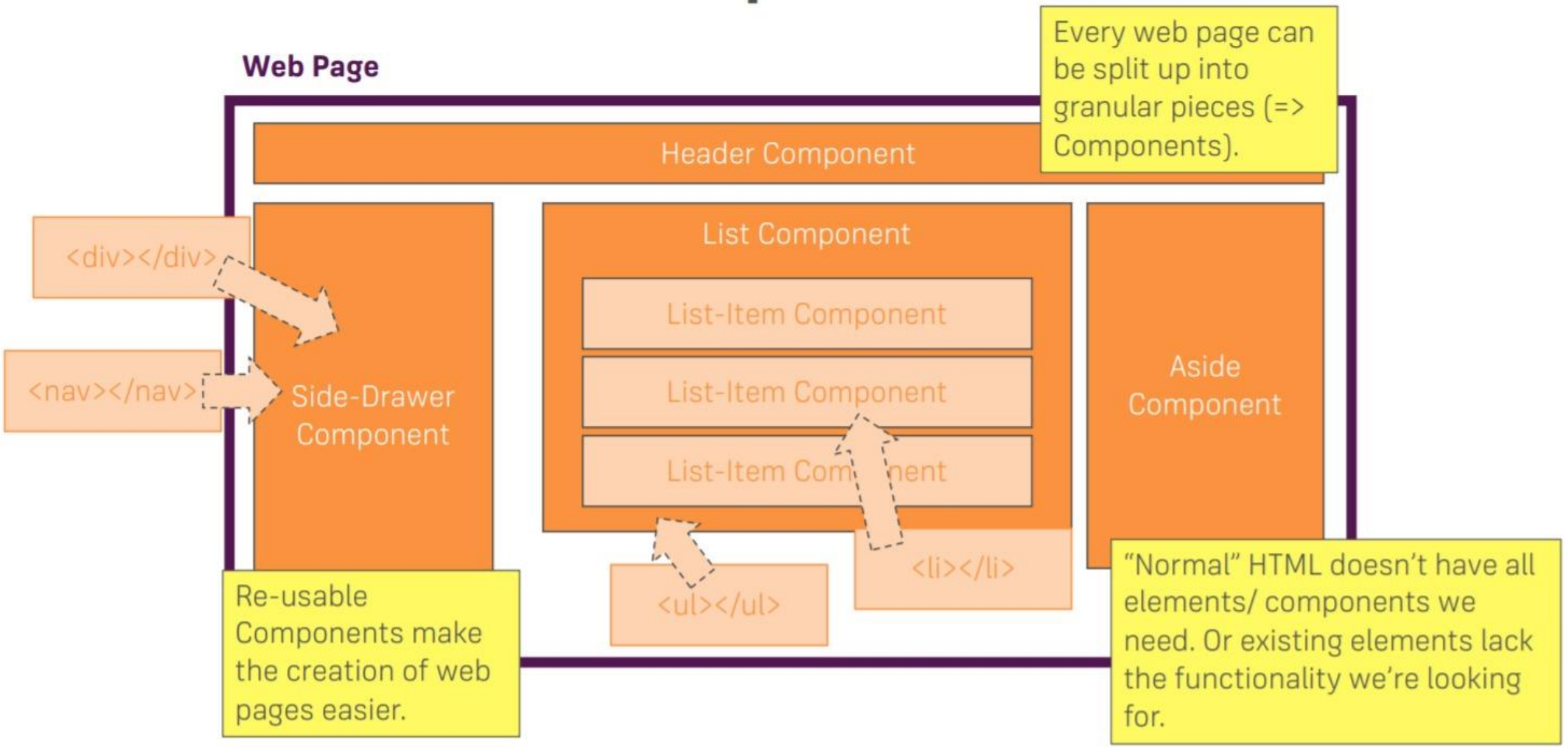
Feature Comparison



What are Components?



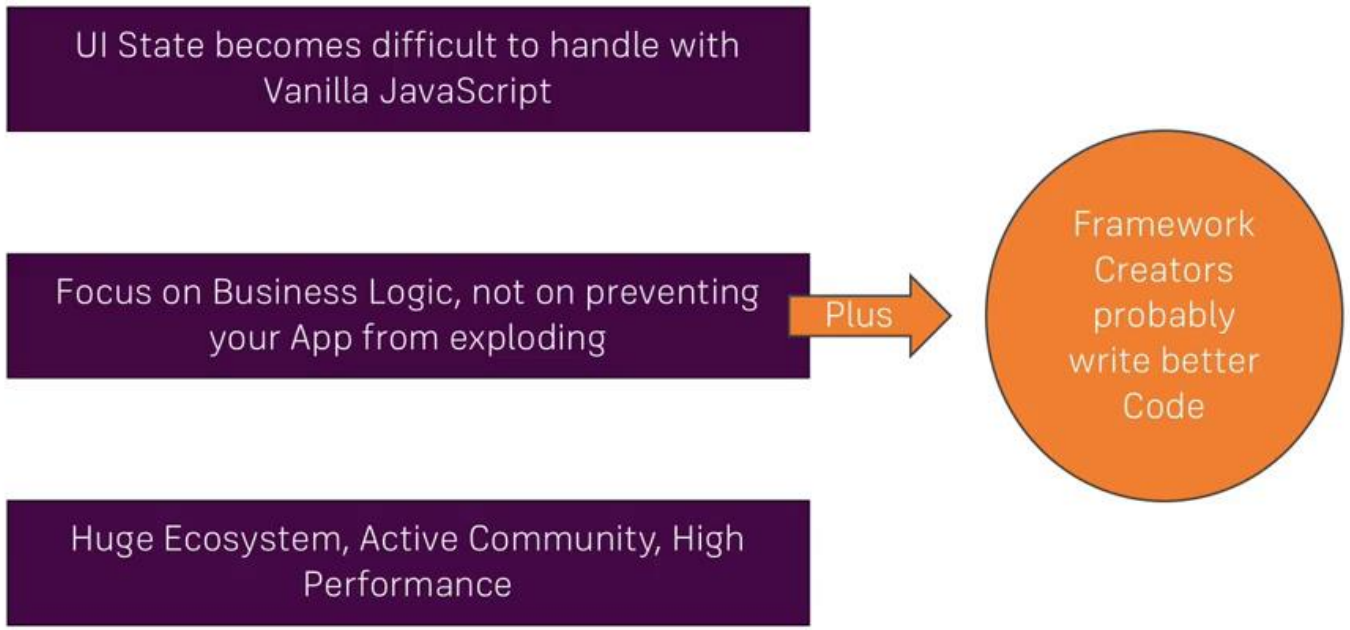
Components


Web Page
Every web page can be split up into granular pieces (=> Components).
Header Component

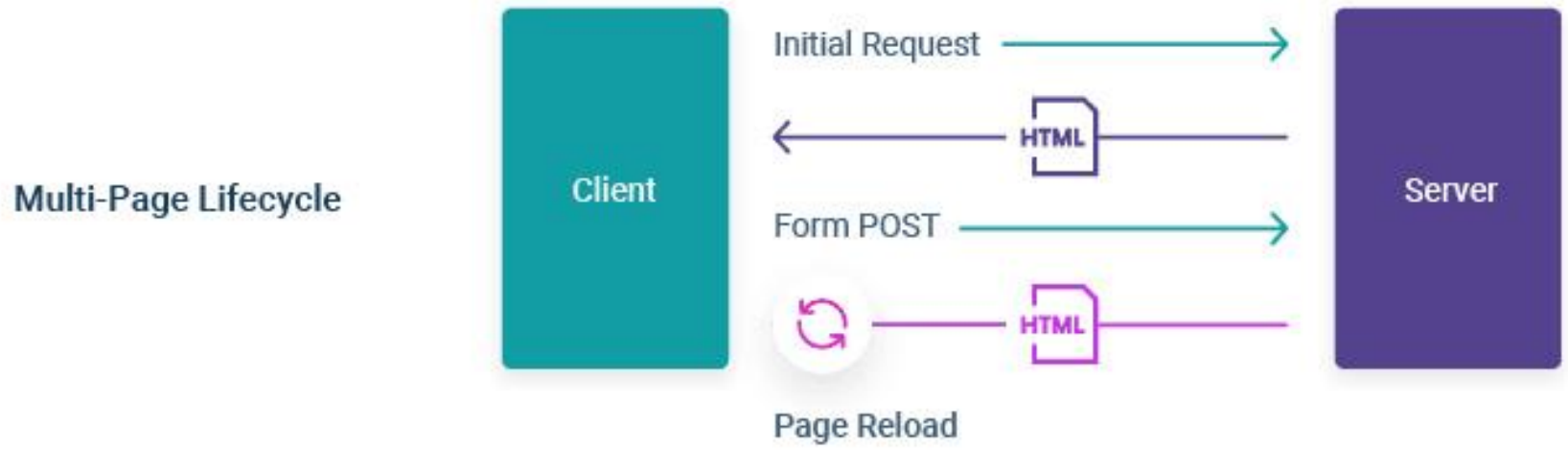
<nav></nav>
Side-Drawer Component
List Component
List-Item Component
List-Item Component
List-Item Component
Aside Component

Re-usable Components make the creation of web pages easier.
"Normal" HTML doesn't have all elements/ components we need. Or existing elements lack the functionality we're looking for.

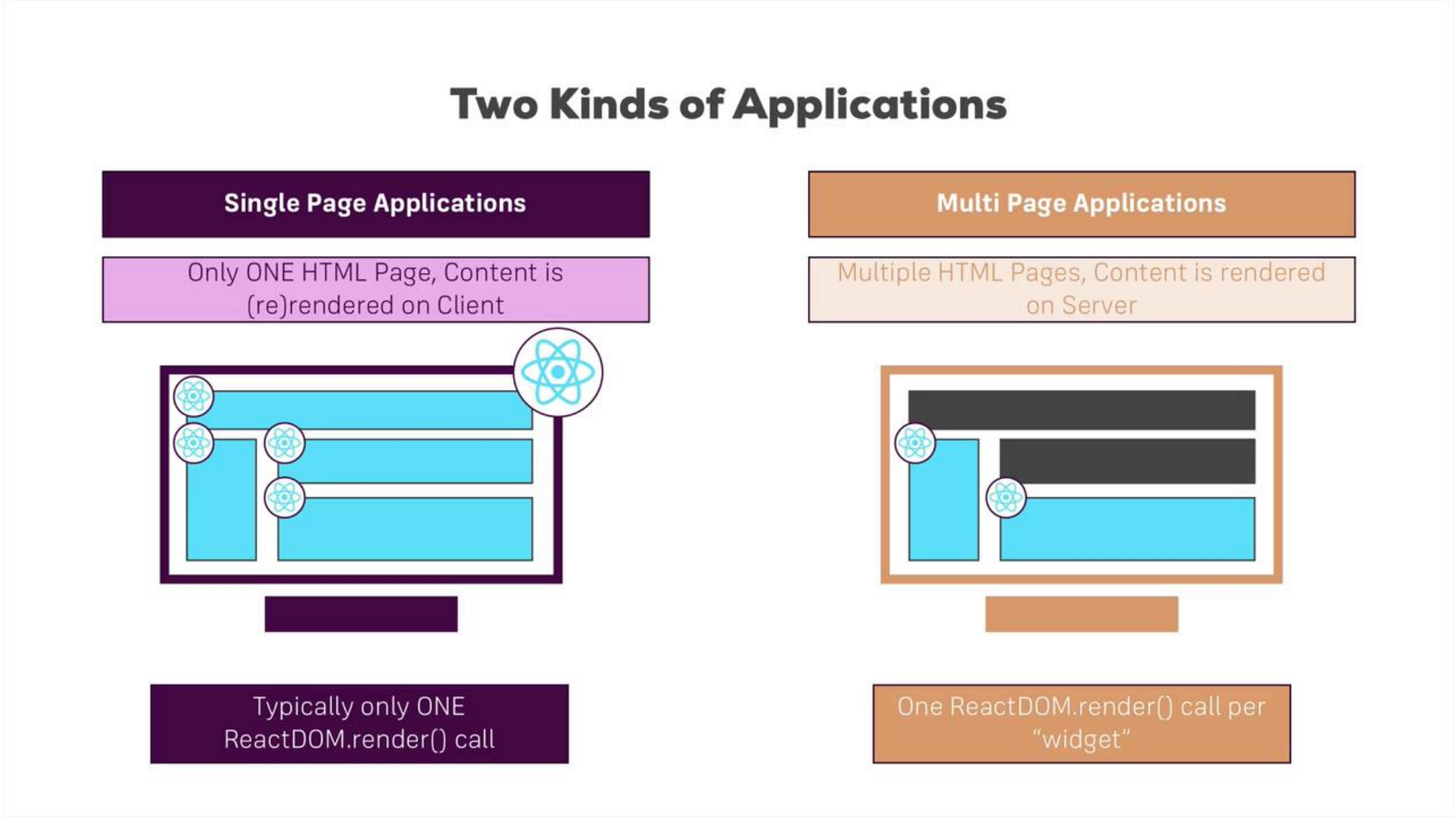
Why do we need JavaScript Frameworks?



What is a Single Page Application(SPA)?



SPA & MPA in React



What is Webpack?

Webpack is a tool that lets you **compile** JavaScript modules, also known as **module bundler.** Given a large number of files, it generates a **single file (or a few files)** that run your app.

It can perform many operations:

- helps you **bundle your resources**.
- watches for **changes** and **re-runs** the tasks.
- can run **Babel transpilation to ES5**, allowing you to use the latest JavaScript features without worrying about browser support.
- can transpile CoffeeScript to JavaScript
- can convert **inline images to data URIs**.
- allows you to use **require() for CSS files**.
- can run a **development webserver.**
- can handle **hot module replacement.**
- can split the **output files into multiple files**, to avoid having a huge js file to load in the first page hit.
- can perform **tree shaking**.

bundle your scripts

TOP BENEFITS OF REACT FOR FRONT-END DEVELOPMENT

A ~~STEEP~~ GENTLE LEARNING CURVE

FLEXIBILITY

FAST RENDERING

TESTABLE RICH UIS

UPLIFTED PRODUCTIVITY

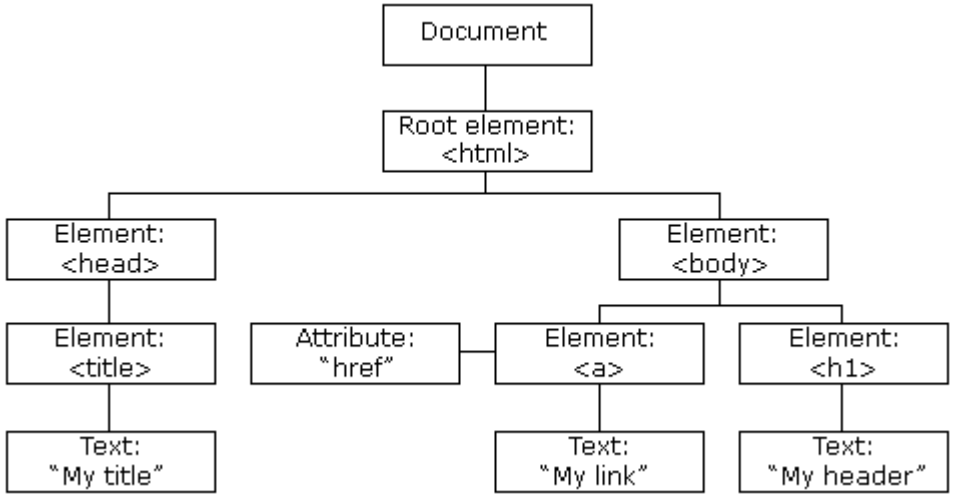
SEO FRIENDLINESS

A STRONG COMMUNITY

Real DOM vs Virtual DOM

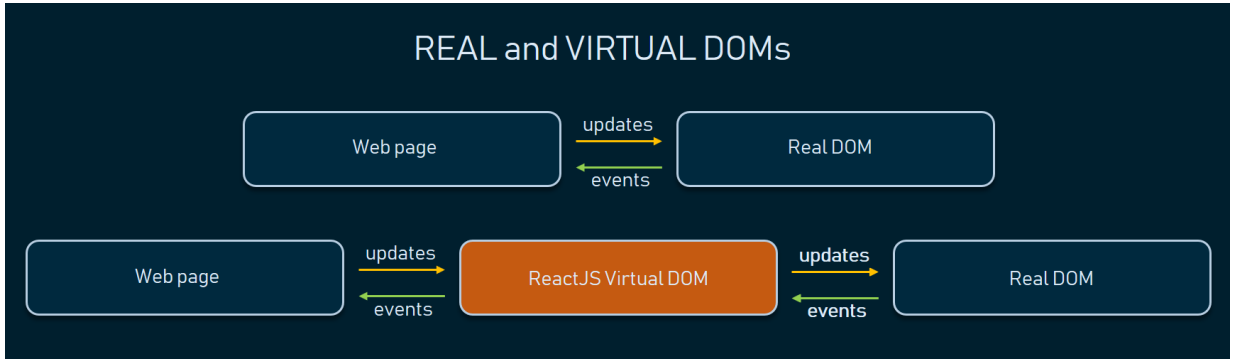
What is Real DOM?

The **Document Object Model** (DOM) is a programming interface for HTML and XML documents. It represents the page so that programs can change the document structure, style, and content. The DOM represents the document as nodes and objects. That way, programming languages can connect to the page.

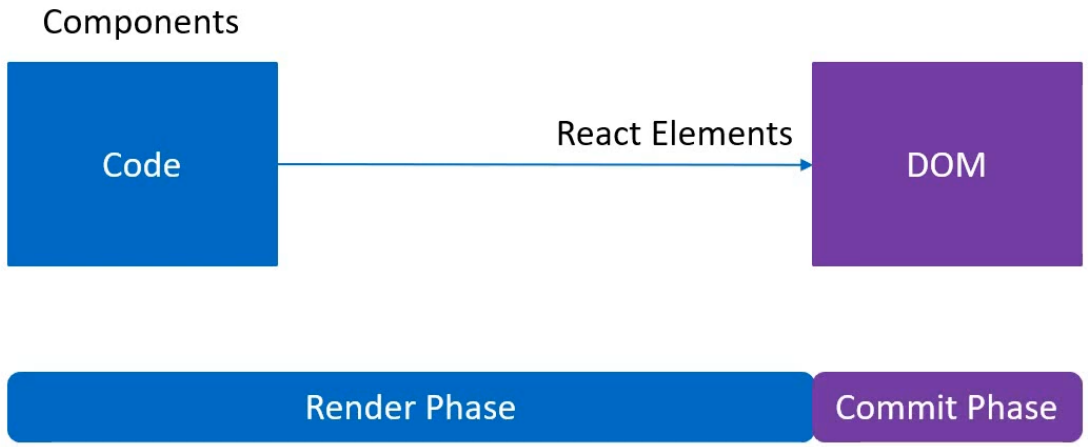


What is Virtual DOM?

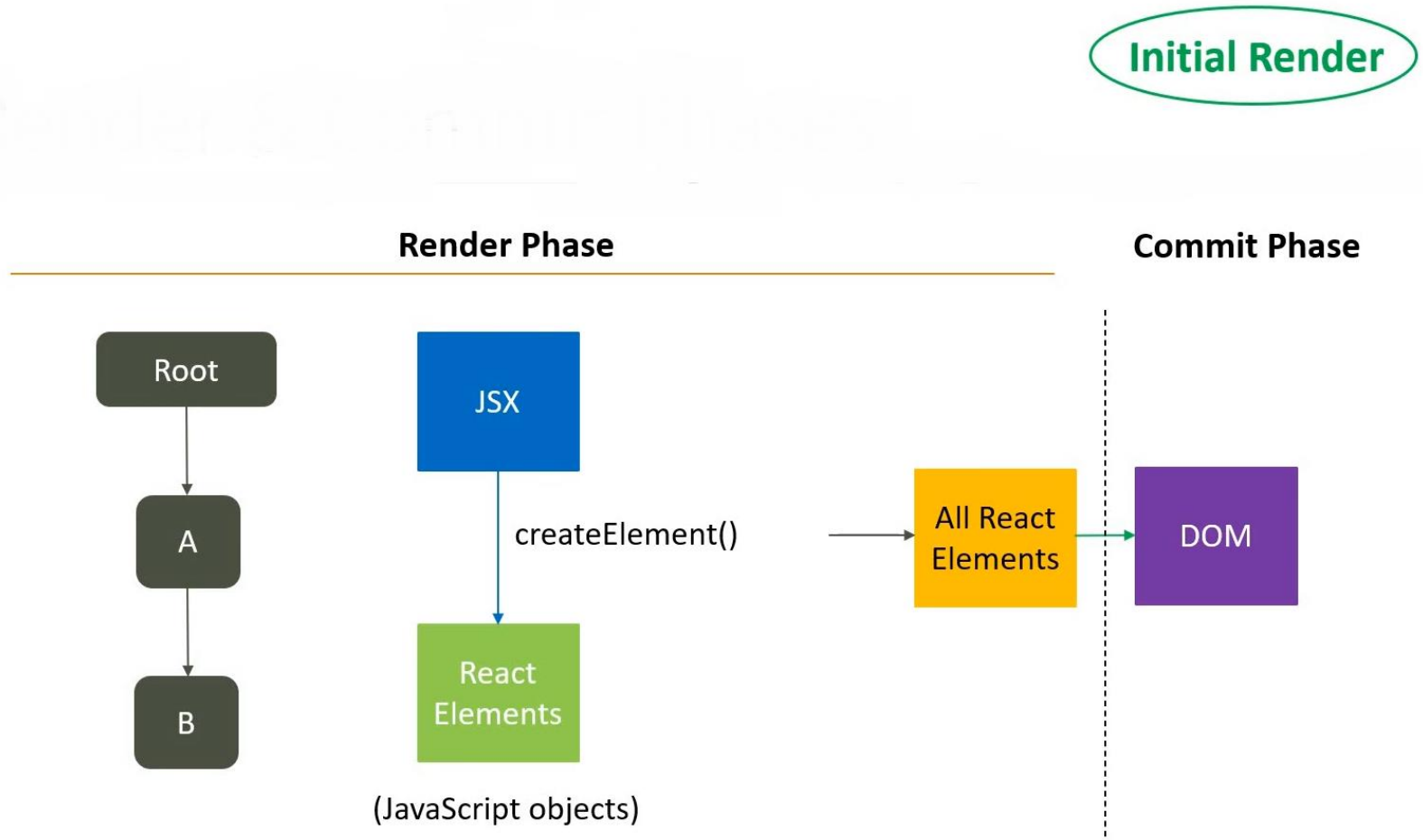
The Virtual DOM (VDOM) is a programming concept where an ideal, or "**virtual**", **representation of a UI is kept in memory and synced with the "real" DOM** by a library such as ReactDOM. This process is called **reconciliation**.



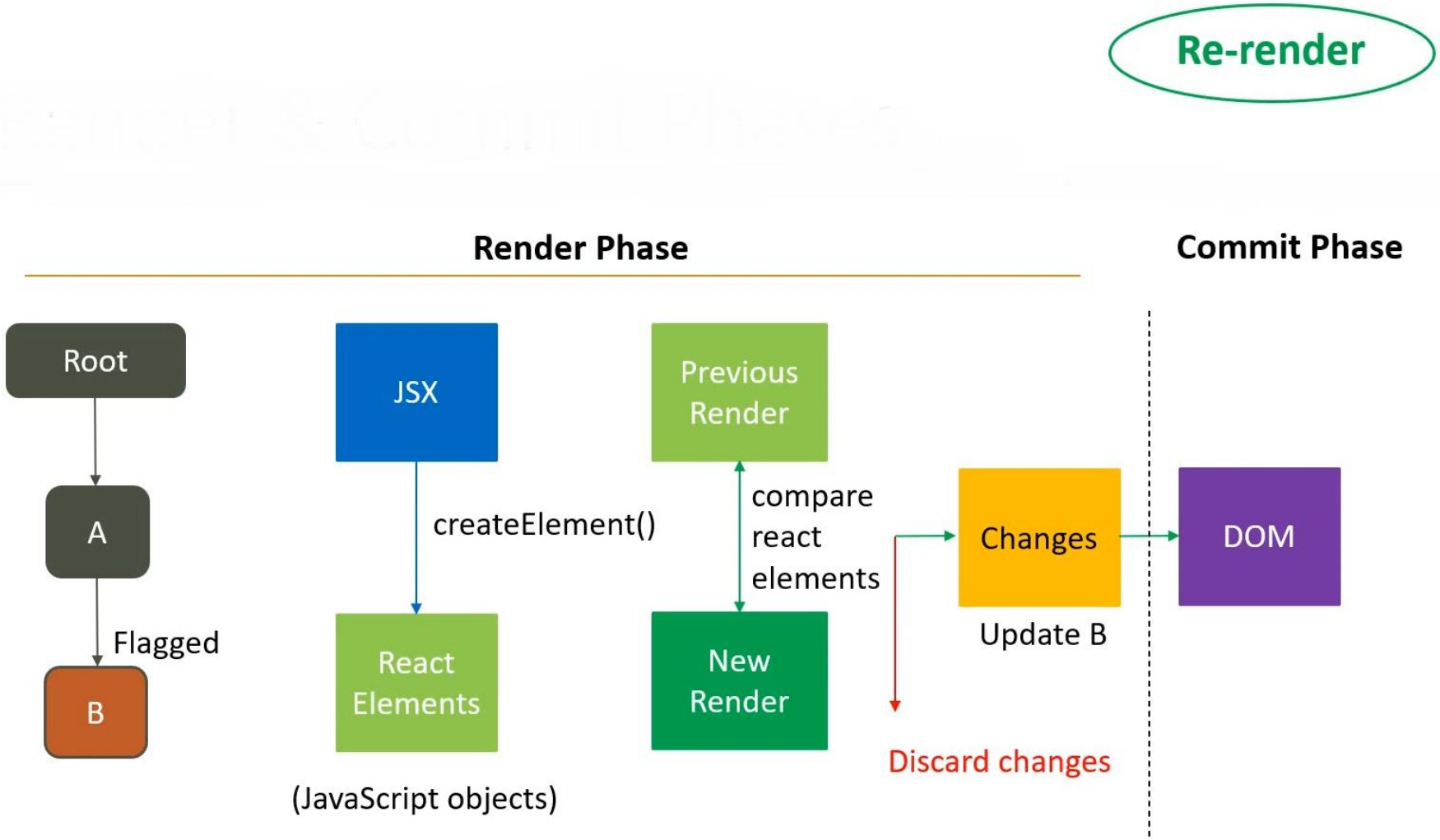
Rendering in React



Phases: Render & Commit (Initial)



Phases: Render & Commit (Update)



Re-rendering Scenario

- **Render Phase:-**
  1. Find all elements flagged for updates.
  2. For each flagged component, convert JSX to React elements and store the results.
  3. Perform reconciliation- Diff old & new tree of react elements (aka Virtual DOM).
  4. Hand over the changes to the next phase.

- **Commit Phase:-**
  - Apply changes to the Real DOM.

Environment Setup & create-react-app CLI

- **Step 1:** Download **NodeJS**. Link: https://nodejs.org/en/download/

- **Step 2:** Open cmd prompt or terminal and enter the following commands:

  > **npx create-react-app hello-react** (OR) **yarn create react-app hello-react**
  > **cd my-app**

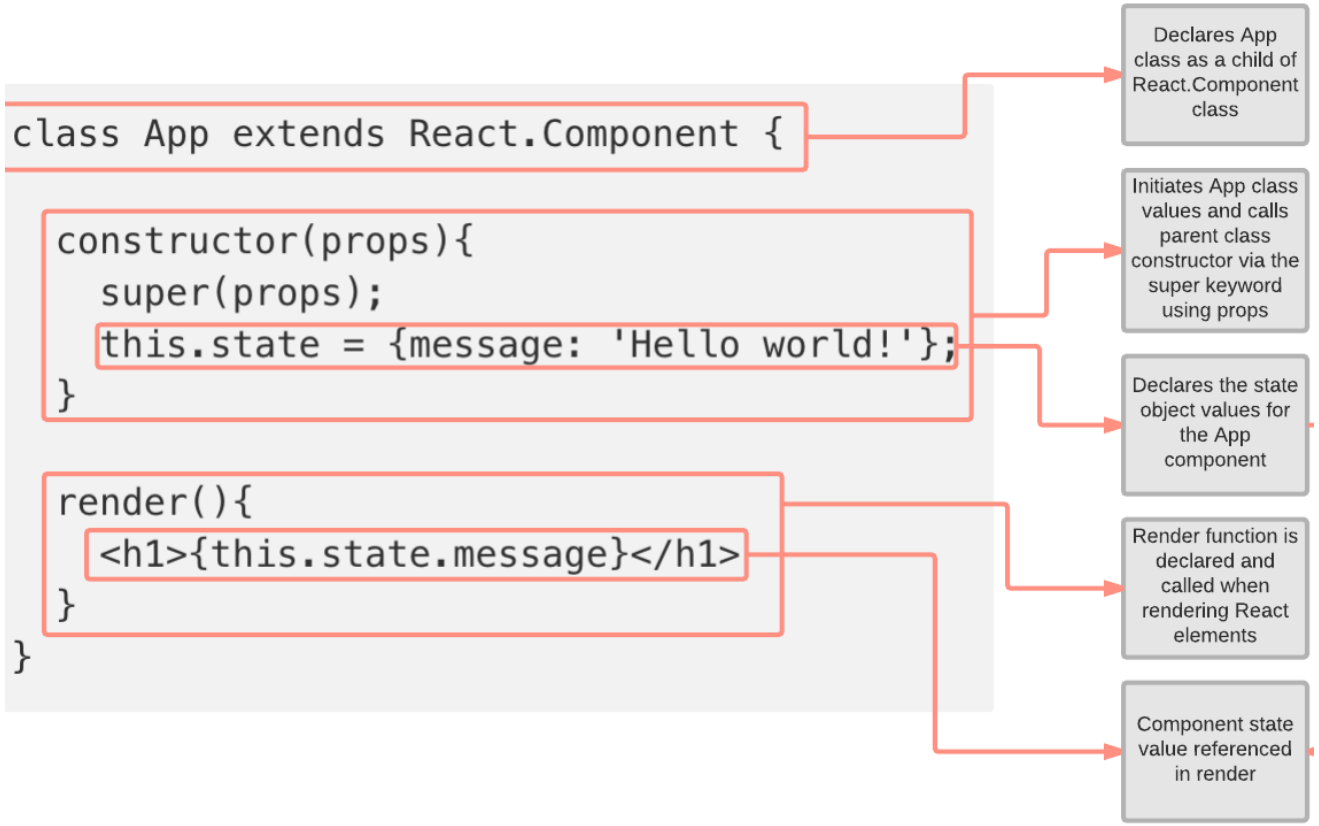
- **Step 3:** Then enter the command: > **npm start** (OR) **yarn start**
  - This will launch the development server at http://localhost:**3000**.

JSX, Props & State

- **JSX**
  JSX Allows us to include '**HTML**' in the same file along with '**JavaScript**' **(HTML+JS=JSX).** Each component in React generates some HTML which is rendered by the DOM.

- **Props**
  Props are the **read-only** data which is passed from one component to another. Props are **immutable**.

- **State**
  State is an **updatable structure** that is used to **contain data or information** about the component. It is the **heart** of the react component which determines the behavior of the component and how it will render.



JSX Transform

- Browsers don’t understand JSX out of the box, so most React users rely on a compiler like Babel or TypeScript to **transform JSX code into regular JavaScript**.
- When you use JSX, the compiler transforms it into React function calls that the browser can understand. The JSX transform turns JSX in to **React.createElement(…)** calls.
- Every JSX expression must have **one Root element.**

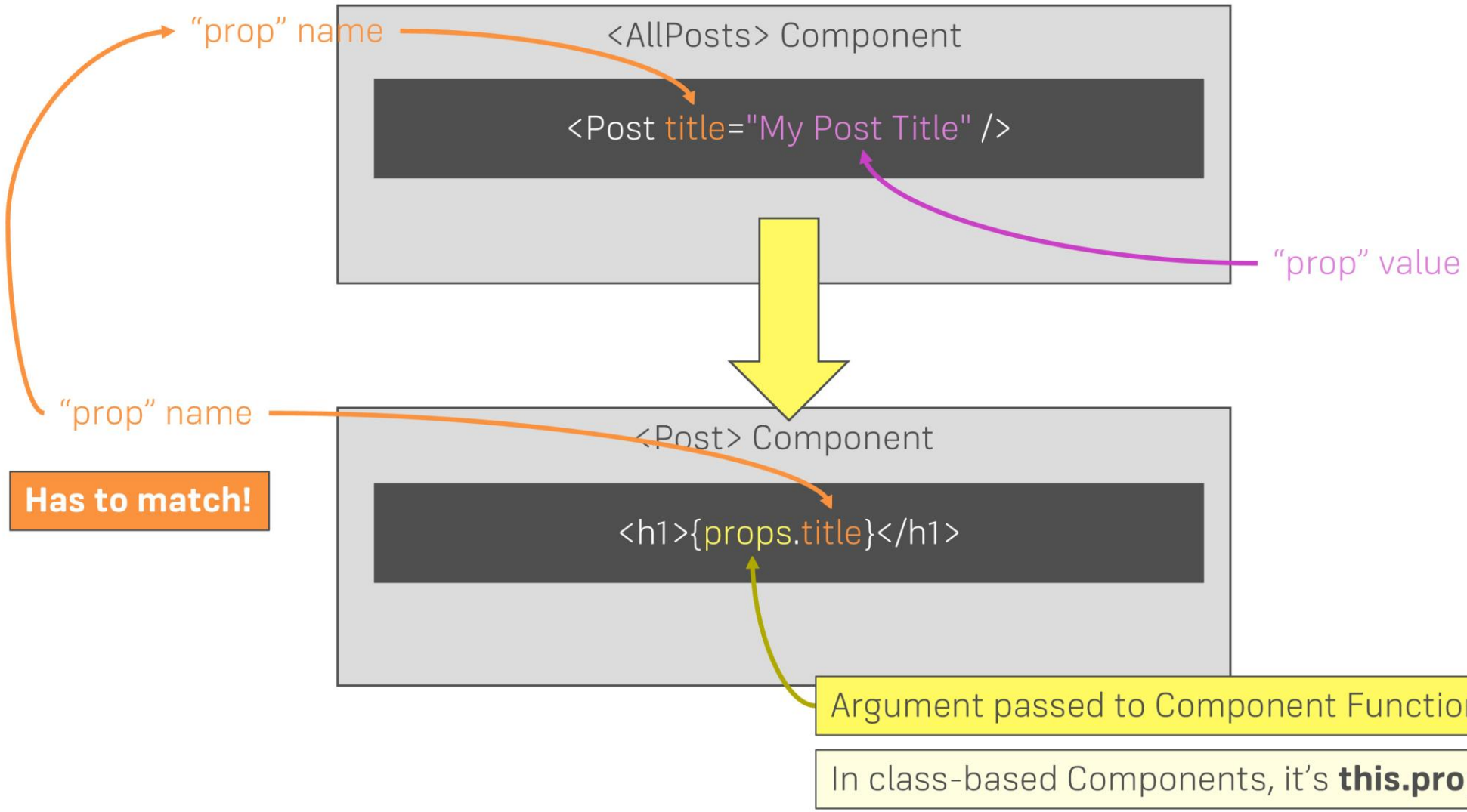
```
<h1 className="wowzers">Yay!</h1>
```

Into this:

```
React.createElement('h1', { className: 'wowzers' }, 'Yay!')
```

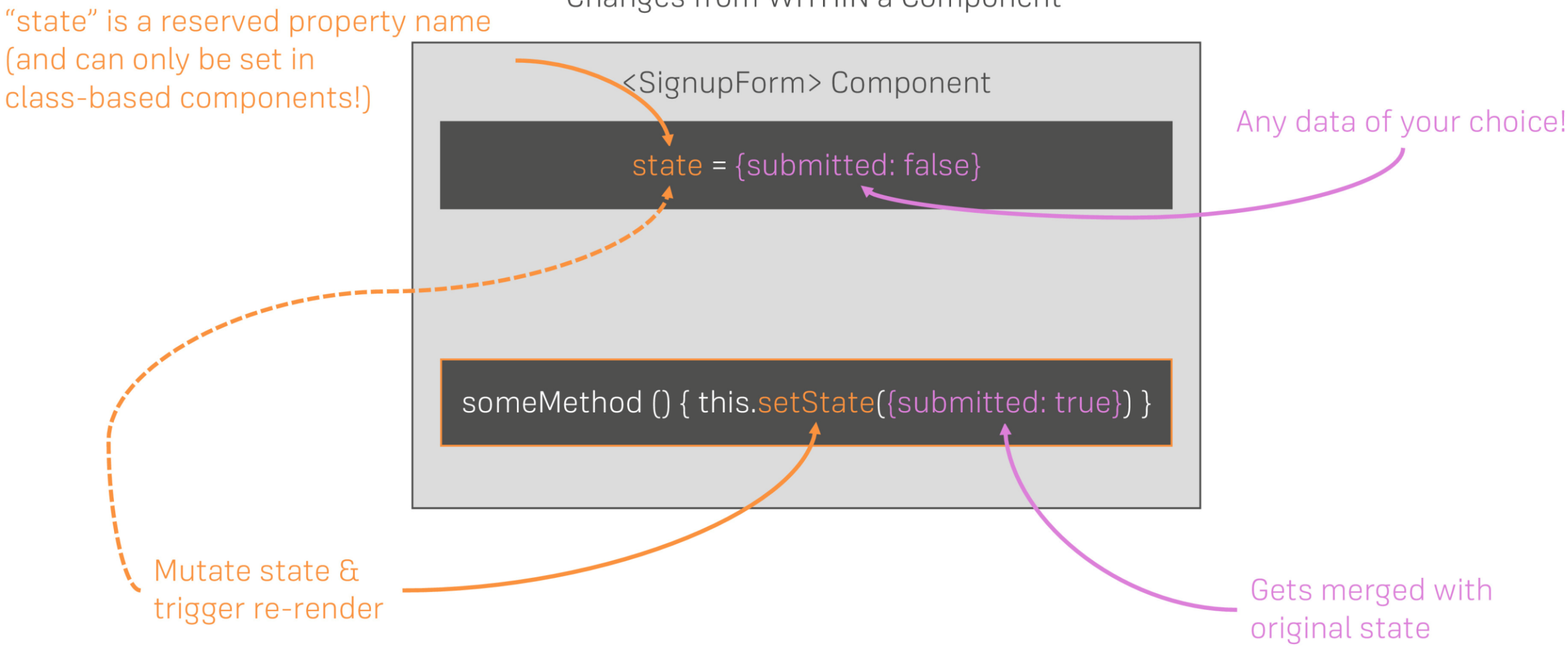
Props

Changes from OUTSIDE a Component (data passed into component)

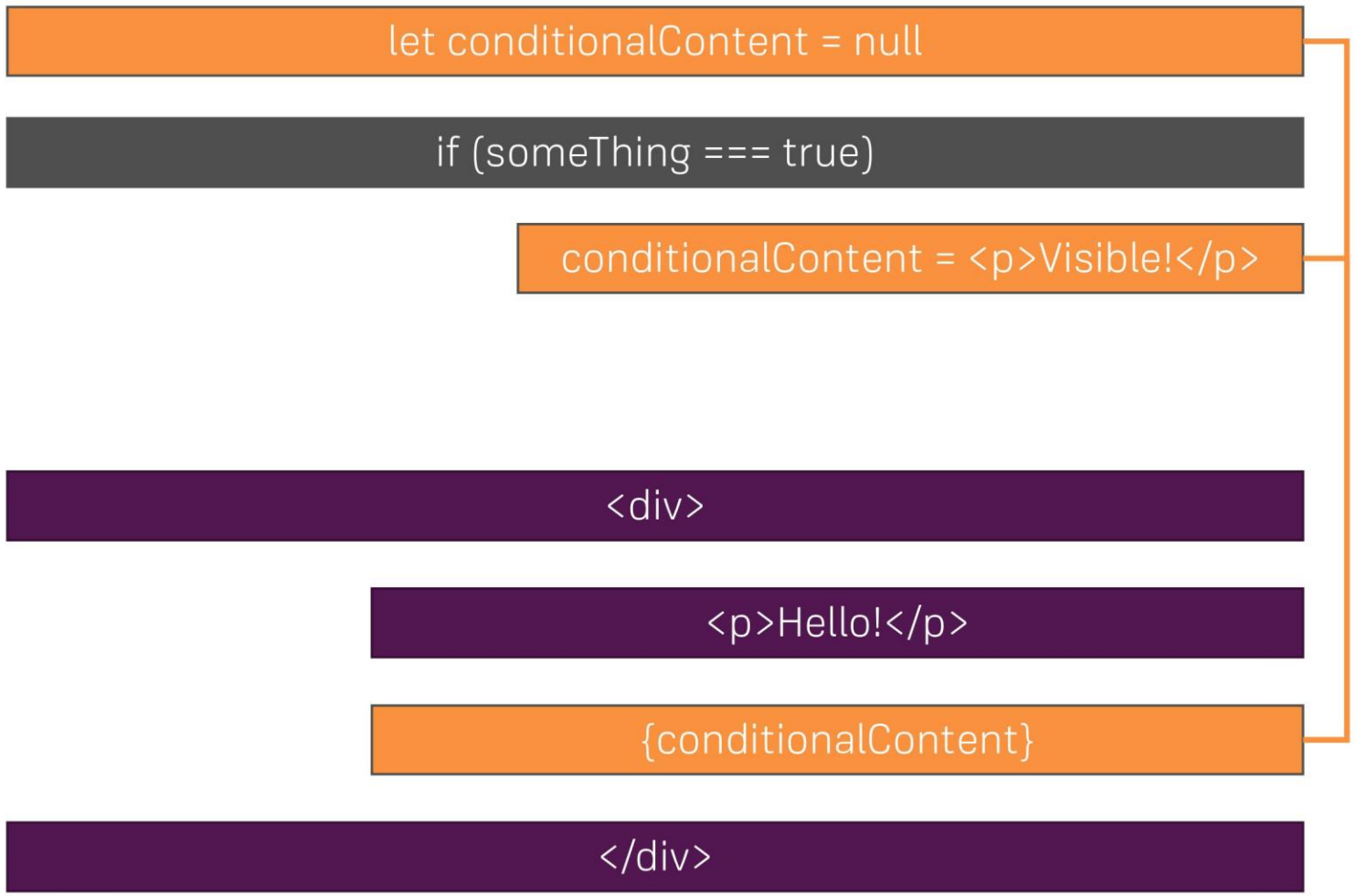


State

Changes from WITHIN a Component



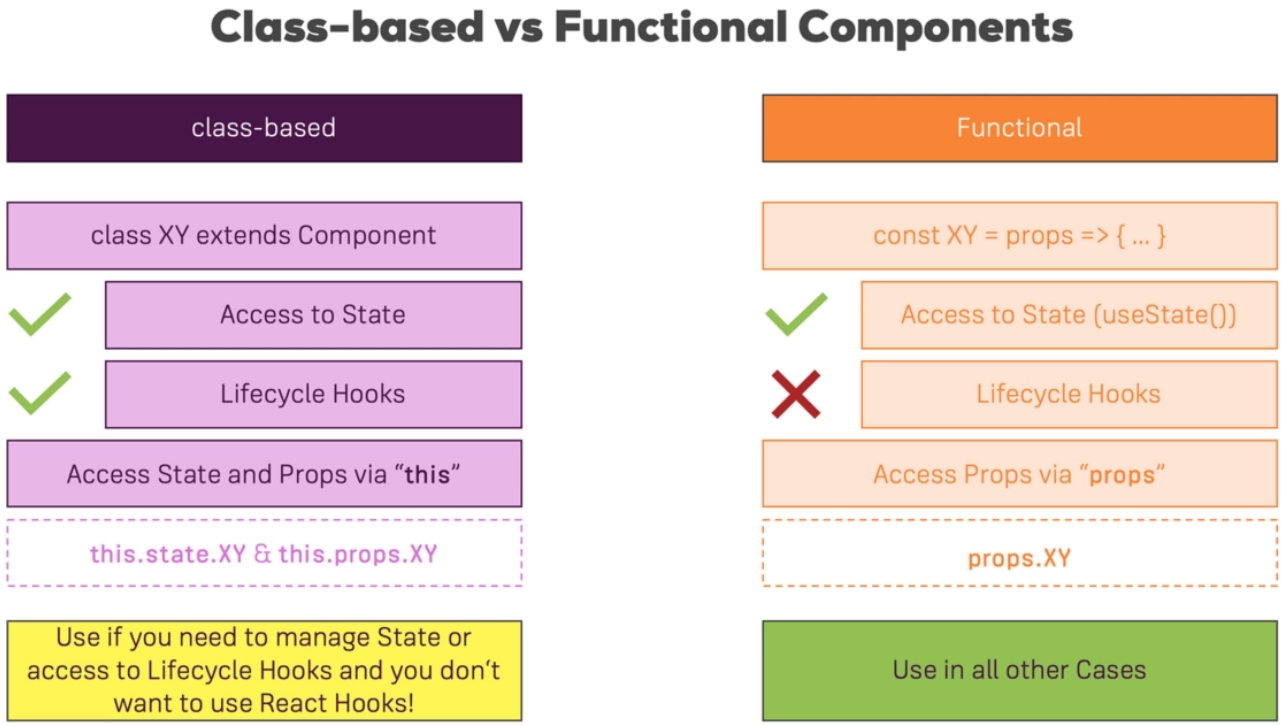
JSX & Conditionals



Stateful vs Stateless Components

- A **Stateful** component is a component that **manages its internal state**. Typically, a ***Class-based*** component is Stateful component. They are also called **Intelligent** or **Container** components.

- A **Stateless** component is a component that **does not manage its internal state**. Typically, a ***Functional*** component is a Stateless component. They are also called **Dumb** or **Presentational** components.



Refs & Keys

Refs

- Refs provide a way to **access DOM nodes or React elements** created in the render method.

- There are a few cases where you need to imperatively modify a child outside of the typical dataflow. The child to be modified could be an instance of a React component, or it could be a DOM element. For both of these cases, React provides an escape hatch.

- **When to use Ref:**
    1. Managing focus, text selection, or media playback.
    2. Triggering imperative animations.
    3. Integrating with third-party DOM libraries.

- **Avoid** using refs for anything that can be done declaratively.

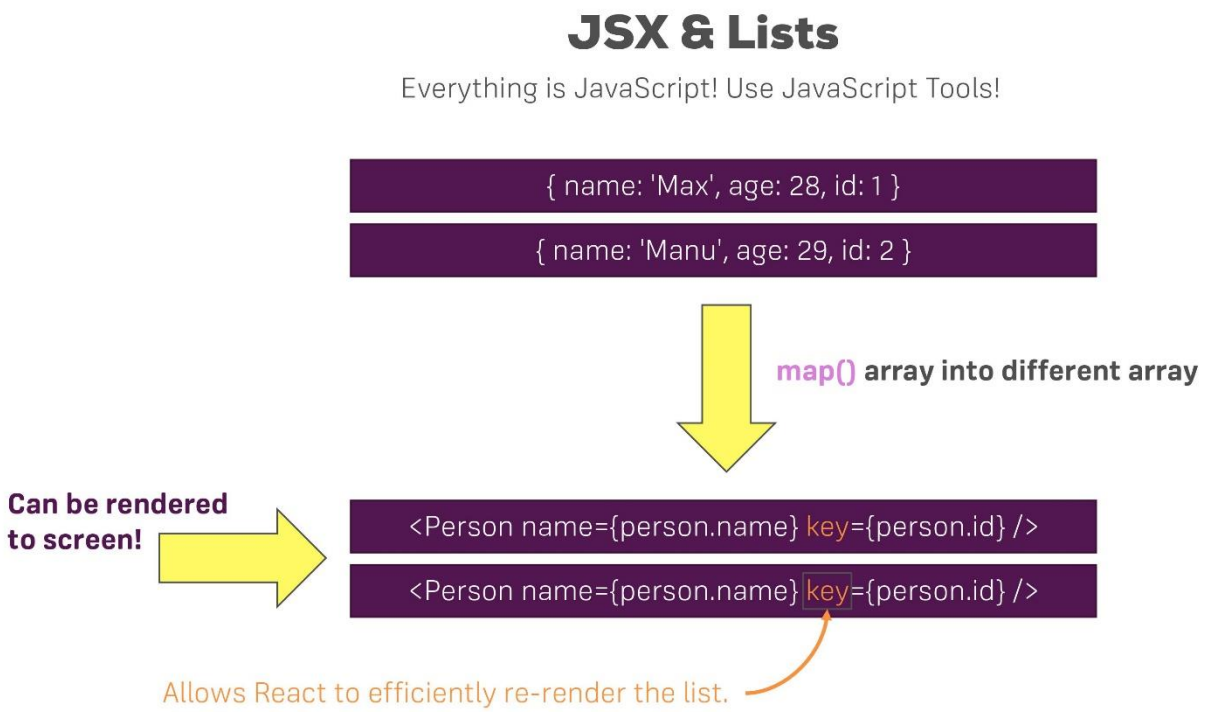
List & Keys

- Keys help React **identify** which items have changed, are added, or are removed.

- Key should be provided to **every list item** while rendering a **list** of HTML elements of React components to give them a **stable identity.**

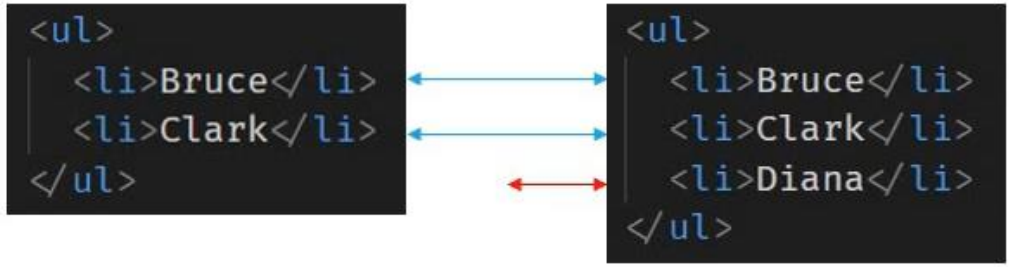
- A key should always be **unique**.

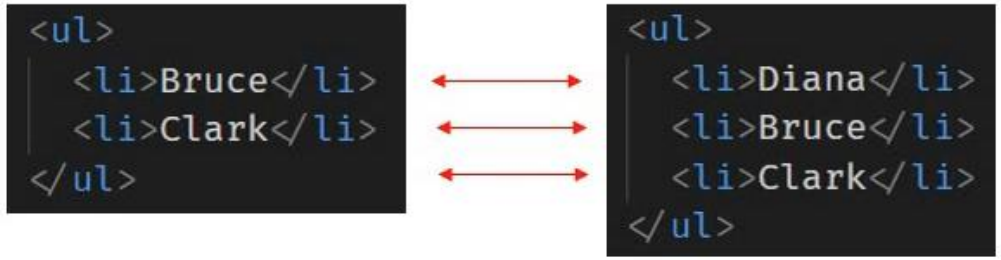
- Keys help in **efficient update** of the User Interface.



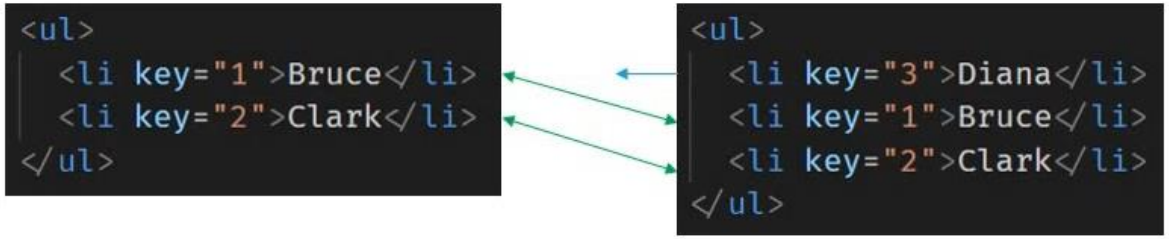
Why to use Key?

List without key attribute

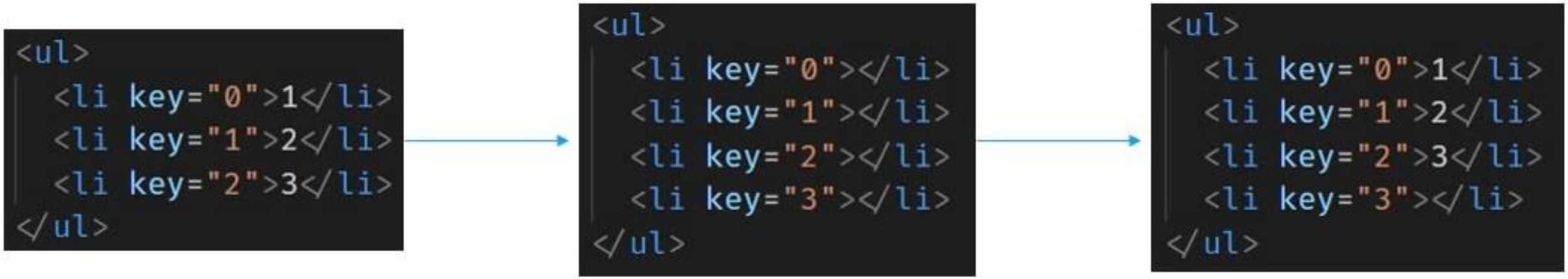




List with key attribute



Why NOT to use Index as Key?



When to use Index as Key?

- The Item in your list **do not have a unique id**.
- The list is static and **will not change**.
- The list will never be **re-ordered** or **filtered**.

Component Lifecycle

- Each component has several "**lifecycle methods**" that you can override to **run code at particular times in the process**.

- **Mounting:** These methods are called in the following order when an instance of a component is being created and inserted into the DOM.
  - **constructor()**
  - static getDerivedStateFromProps()
  - **render()**
  - **componentDidMount()**

- **Updating:** An update can be caused by changes to props or state. These methods are called in the following order when a component is being re-rendered.
  - static getDerivedStateFromProps()
  - shouldComponentUpdate()
  - **render()**
  - getSnapshotBeforeUpdate()
  - **componentDidUpdate()**

- **Unmounting:** This method is called when a component is being removed from the DOM.
  - **componentWillUnmount()**

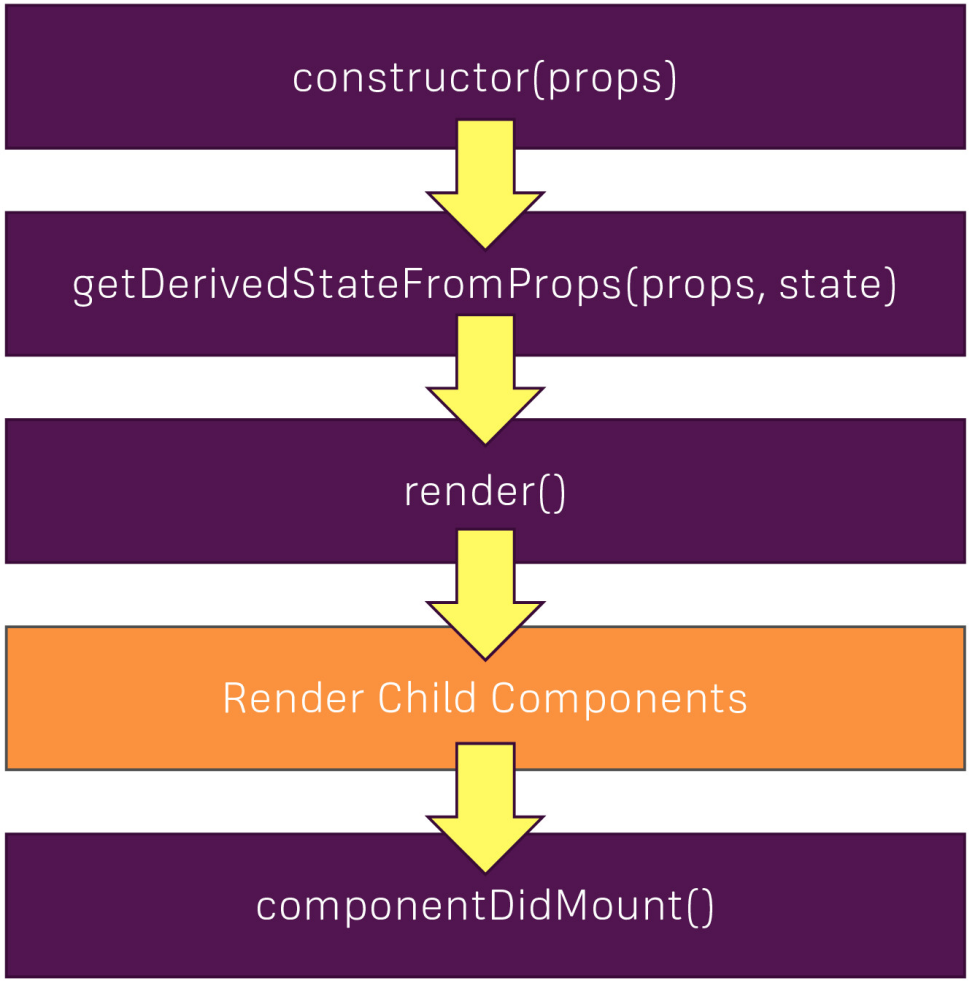
Component Lifecycle

Only available in Class-based Components!

- constructor()
- getDerivedStateFromProps()
- getSnapshotBeforeUpdate()
- componentDidCatch()
- componentWillUnmount()
- shouldComponentUpdate()
- componentDidUpdate()
- componentDidMount()
- render()

Component Lifecycle – Creation

Default ES6 class Feature



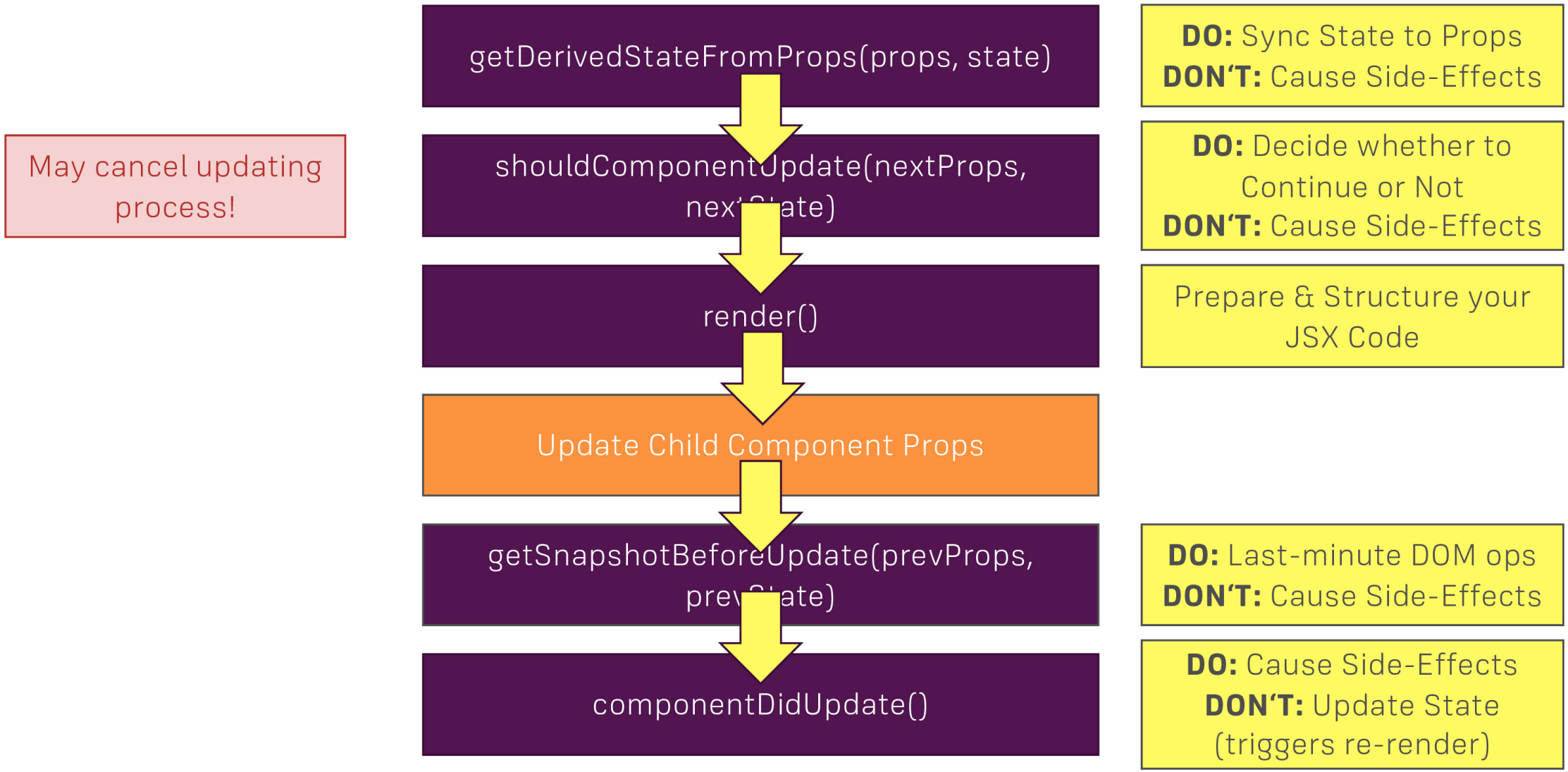
Call super(props)
DO: Set up State
DON'T: Cause Side-Effects

DO: Sync state
DON'T: Cause Side-Effects

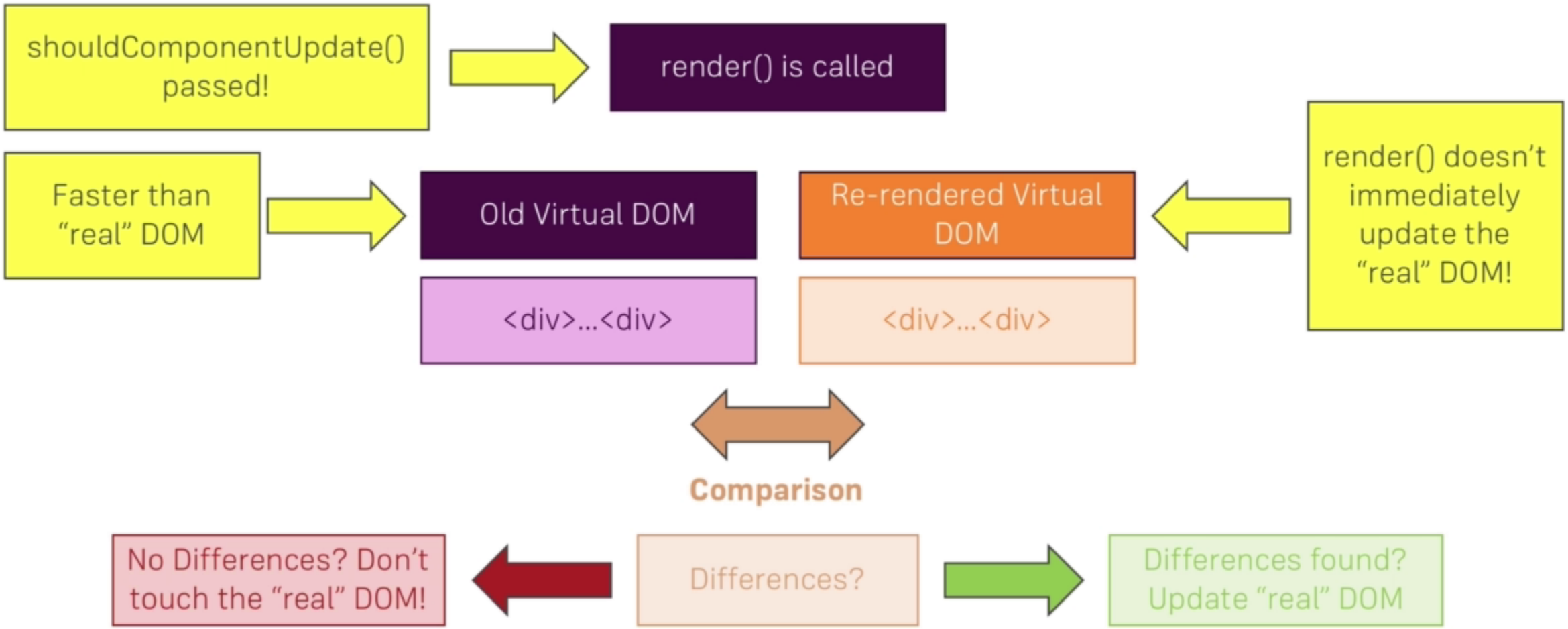
Prepare & Structure your JSX Code

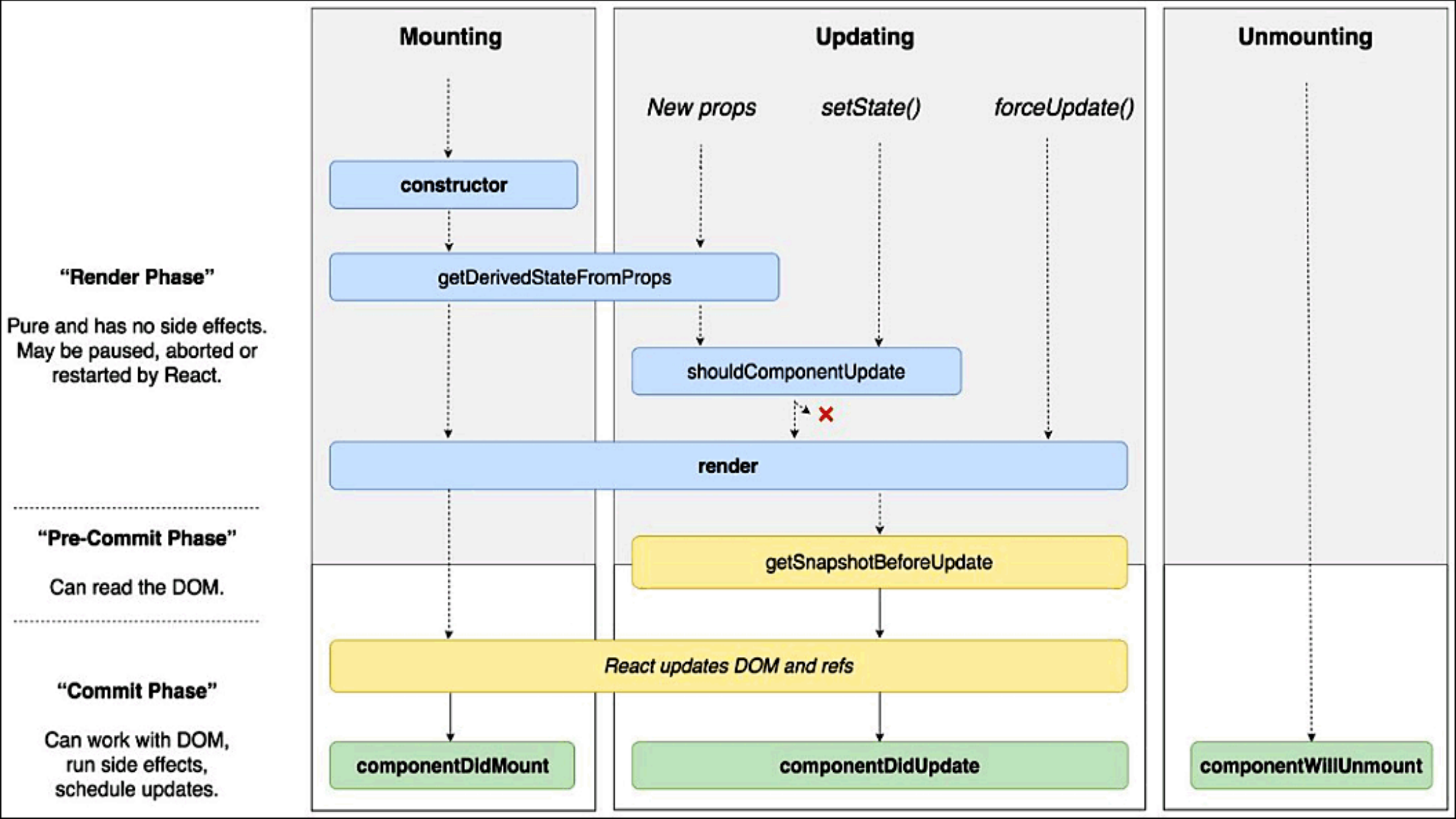
DO: Cause Side-Effects
DON'T: Update State (triggers re-render)

Component Lifecycle – Update



How React Updates The DOM




Mounting
Updating
Unmounting
New props
setState()
forceUpdate()
constructor
getDerivedStateFromProps
shouldComponentUpdate
render
getSnapshotBeforeUpdate
React updates DOM and refs
componentDidMount
componentDidUpdate
componentWillUnmount
"Render Phase"
Pure and has no side effects. May be paused, aborted or restarted by React.
"Pre-Commit Phase"
Can read the DOM.
"Commit Phase"
Can work with DOM, run side effects, schedule updates.

Please visit iEvolve Course: 63518

Thank You ☺