Diploma Web Application Development: Introduction

ICT50220 Diploma of Information Technology(Front-End Web Development)

Code	Title
ICTWEB517	Create web-based programs
ICTWEB546	Validate application design against specifications

North Metropolitan

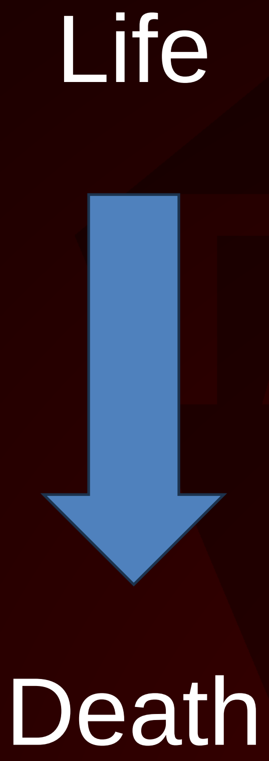
Web Application Development
React Core Concepts

Core Concepts

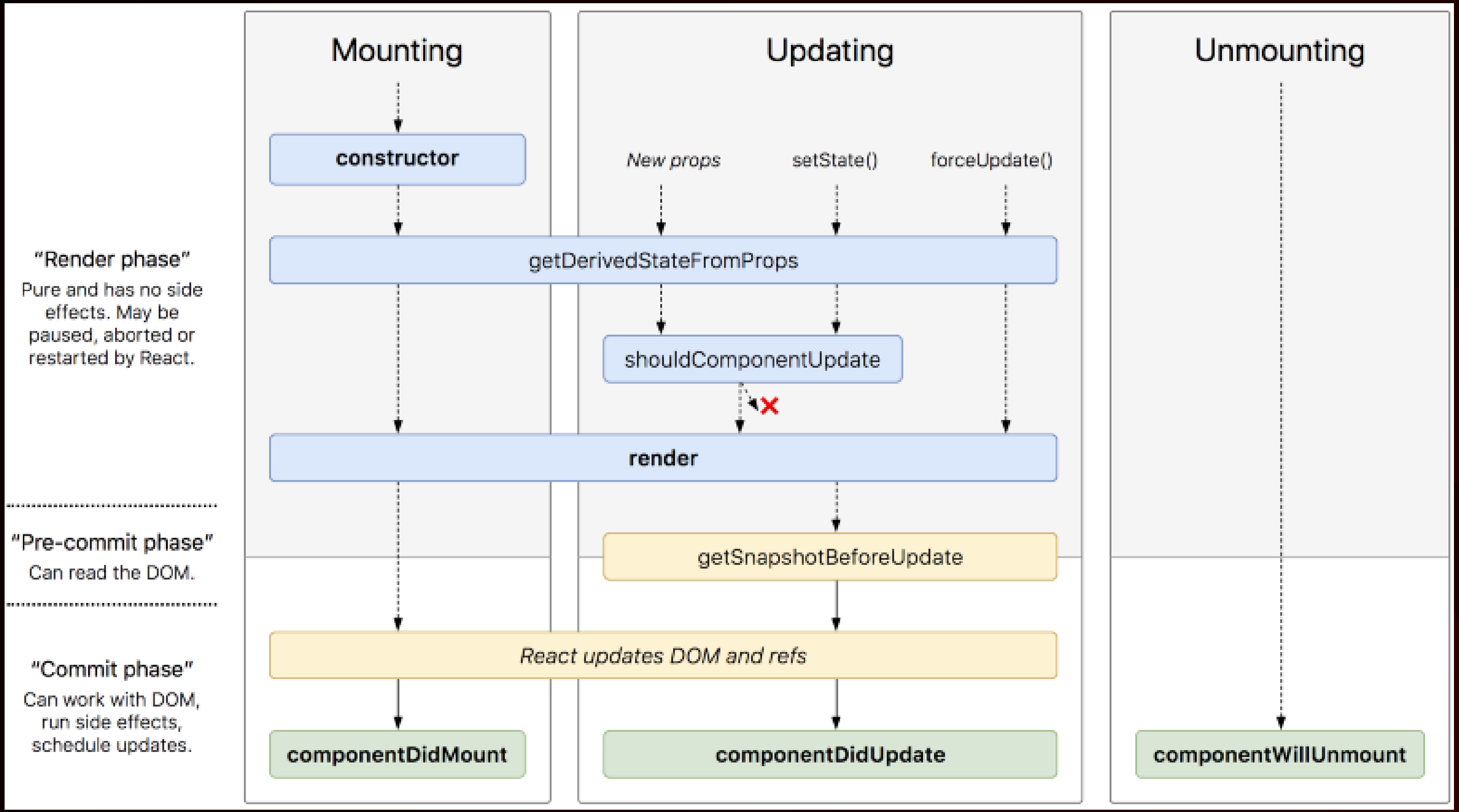
1. Component Lifecycle
2. State and setState()
3. Higher-Order Components
4. Context API
5. Data Flow
6. React Hooks
7. Props

React Component Lifecycle

1. **Mounting** - the component has been created and inserted into the DOM
2. **Re-rendered** - the component is reinserted into the DOM due to a change in the props or the state
3. **Unmounting** - the component is removed from the DOM



React Component Lifecycle



Life → Death

https://www.freecodecamp.org/news/these-are-the-concepts-you-should-know-in-react-js-after-you-learn-the-basics-ee1d2f4b8030/

State & React

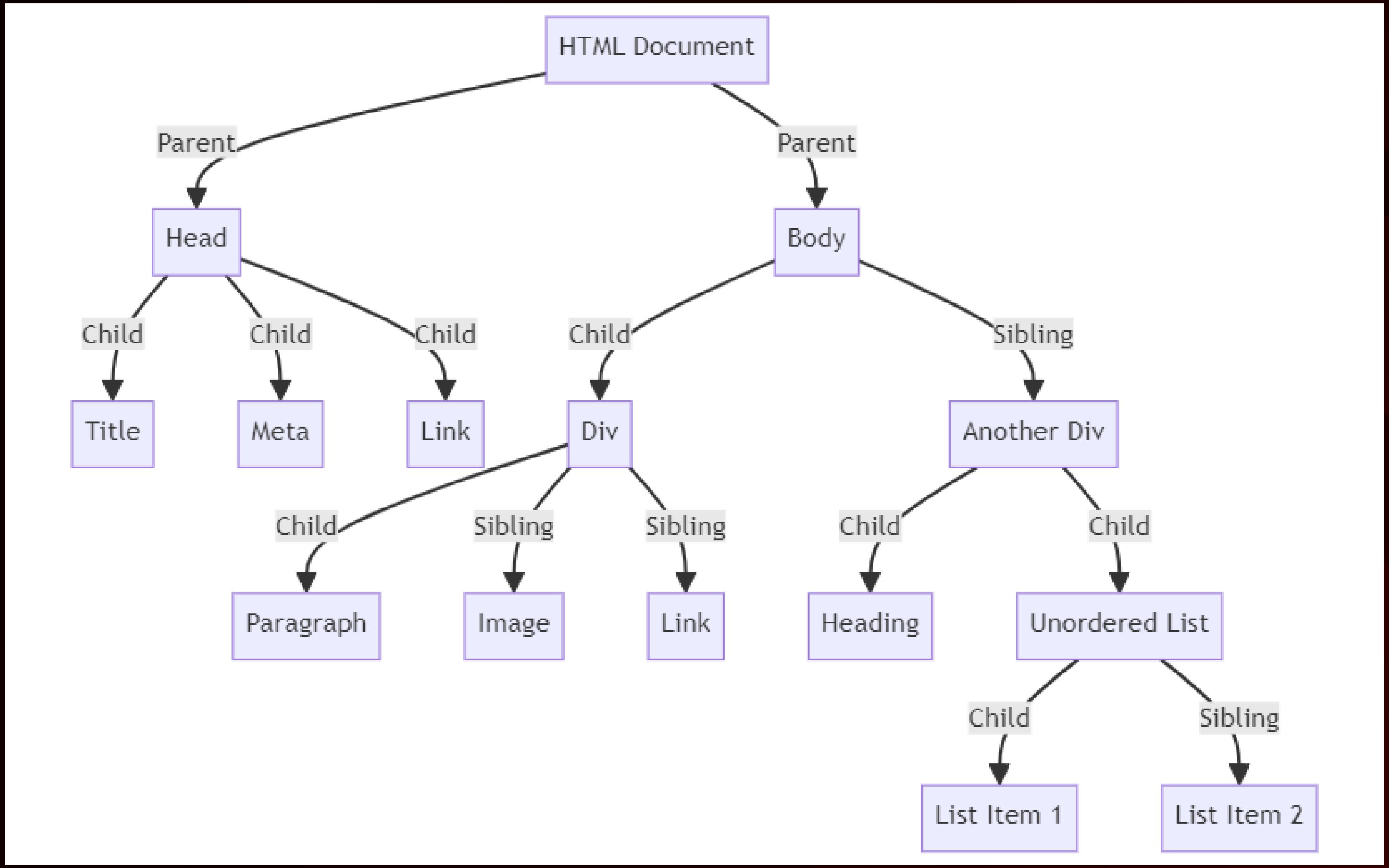
- State is a built-in React **object**
- this object contains data/information about our component
- we know that this data can change over time
- however, we can use & set this state
- our component and its children will be re-rendered with the updated state

```
constructor(props) {
  // Remember props is only data
  super(props);
  // Initialising state
  this.state = {
    count: 0
  };
}

// Method to update state
incrementCount = () => {
  this.setState((prevState) => ({
    count: prevState.count + 1
  }));
};
```

Remember: we discussed form state: empty, typing, submitting, success & error

DOM – the family tree



React Hooks

- hooks are in-built React functions
- we can use them in the lifecycle of function components.

useState - adds state to function components.
useEffect - lets you use side effects in function components.

```
function NasaApod() {
    const [apodData, setApodData] = useState(null);

    useEffect(() => {
          // NASA API key and URL
          const apiKey = 'DEMO_KEY';
          const apiUrl = `https://api.nasa.gov/planetary/apod?api_key=${apiKey}`;
          …
    }
}
```

React – Side Effects

Our code will run actions which run outside of the sequence of the current code.

1. Updating an external document (e.g. .csv file)
2. Adding event listeners
3. Animation of UI elements
4. *Fetching data from an API*
5. Synchronisation with a Cloud Storage service

```
function NasaApod() {
    const [apodData, setApodData] = useState(null);

    useEffect(() => {
        // NASA API key and URL
        const apiKey = 'DEMO_KEY';
        const apiUrl = `https://api.nasa.gov/planetary/apod?api_key=${apiKey}`;
        …
    }
}
```

Javascript Objects

- contain key, value pairs
- we can de-structure them
- we can use dot notation to get values
- can contain other objects & functions

```
const user = {
            firstName: "John",
            lastName: "Doe",
            age: 30,
            isStudent: false,
            address: {
                        street: "123 Main St",
                        city: "Perth",
                        zipCode: "6000",
            },
}
```

The Props Object

- the object is passed between Parents & Children
- the object *prop*erties are used but not changed by the children
- it is really a plain JavaScript object
- the *key: value* pairs can contain all types of values
- including functions, used to pass data back to the parent to use

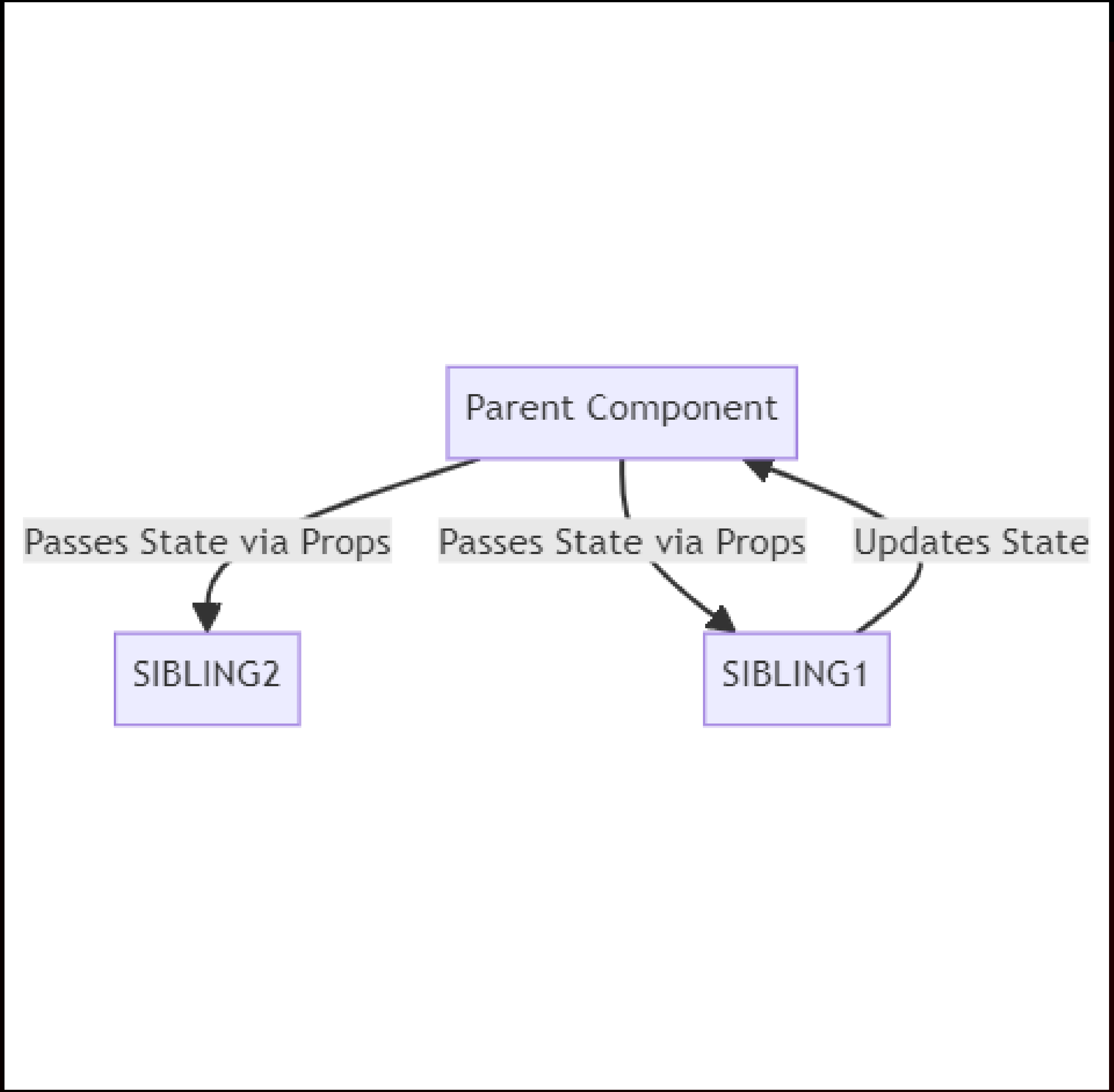
```
// An example props object
const props = {
  apodData: {
    title: "Astronomy Picture of the Day",
    date: "2024-08-21",
    url: "https://example.com/image.jpg",
    explanation: "This is a description of
                 the image.",
  },
    // We can pass function in props
    showAlert: () => alert("Title:
Astronomy Picture of     the Day"),
};
```

Higher Order Functions

- a function that takes a component and returns a new component
- use case: re-using components for different UI
- the original component is passed in the *props* object (*in this case ApodImageDisplay*)

```
// Component to display our NASA Image only
function ApodImageDisplay({ apodData }) {
  return (
    <div>
        <h1>{apodData.title}</h1>
        <img src={apodData.url} alt={apodData.title}/>
    </div>
  );
}

// The Higher order function uses our ApodImageDisplay Component
const updatedDisplay = withNasaApodData(ApodImageDisplay);
```



React High Order Functions

React Context

- if we want a particular state to be available throughout the application.
- creating a context means you can use it anywhere without having to pass it between components
- use case: creating a dark and light theme for the application

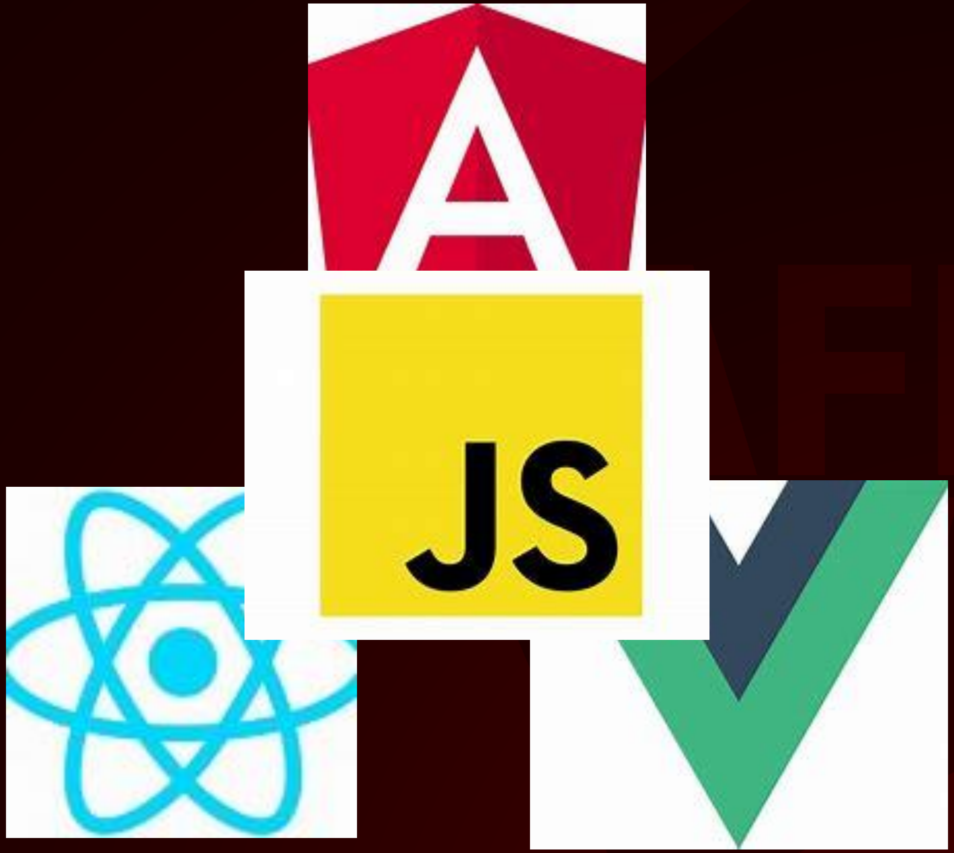
```
// Create a Theme Context with
// "light" as the default value
const ThemeContext = createContext('light’);

// Custom hook to use the Theme Context
function useTheme() {
    return useContext(ThemeContext);
}
// Use the custom hook
const theme = useTheme();
// Apply theme-based css styles
const themeStyles = {
    backgroundColor: theme === 'light' ? '#fff’ :
        '#333',
};

// Add a button to a component to switch the themes
<button onClick={() => setTheme(theme === 'light’ ?
        'dark' : 'light')}>
        Toggle Theme
</button>
```

Big 3 JS Frameworks & Components

- all use components to manager structure, behaviour and look (HTML/CSS & JS)
- they all promote modularity (DRY)
- all use the concept of parents & children in a tree structure
- data and functions are passed between these relations

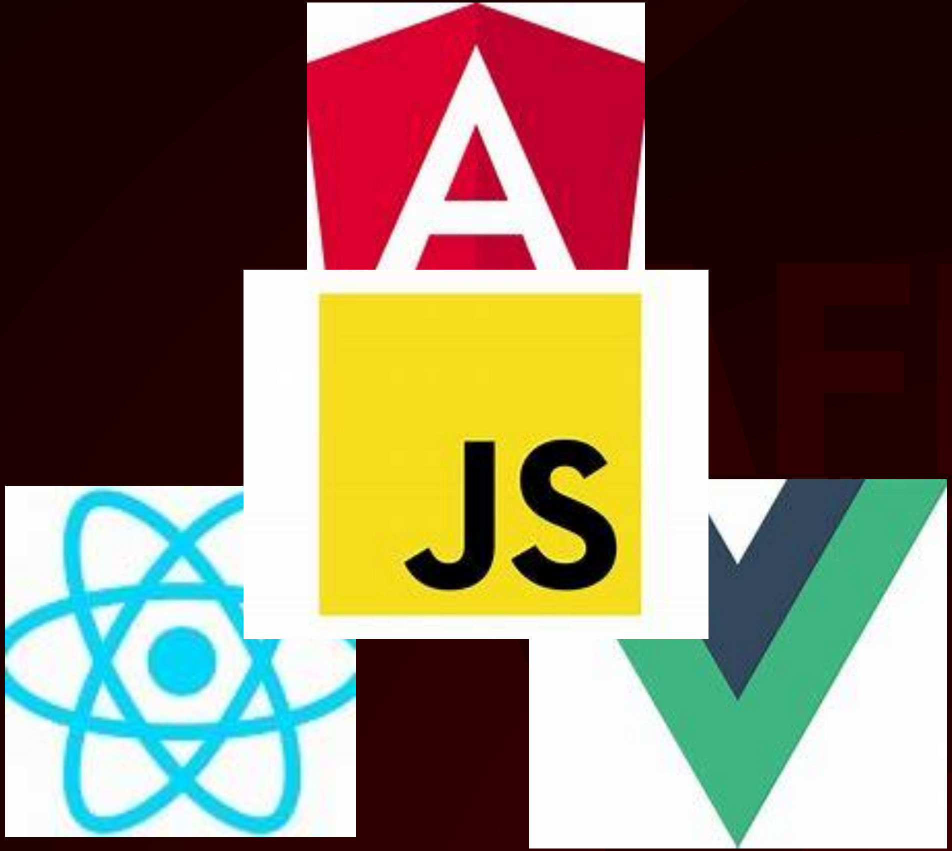


Big 3 JS Frameworks & Component Life

All three follow the sequence:

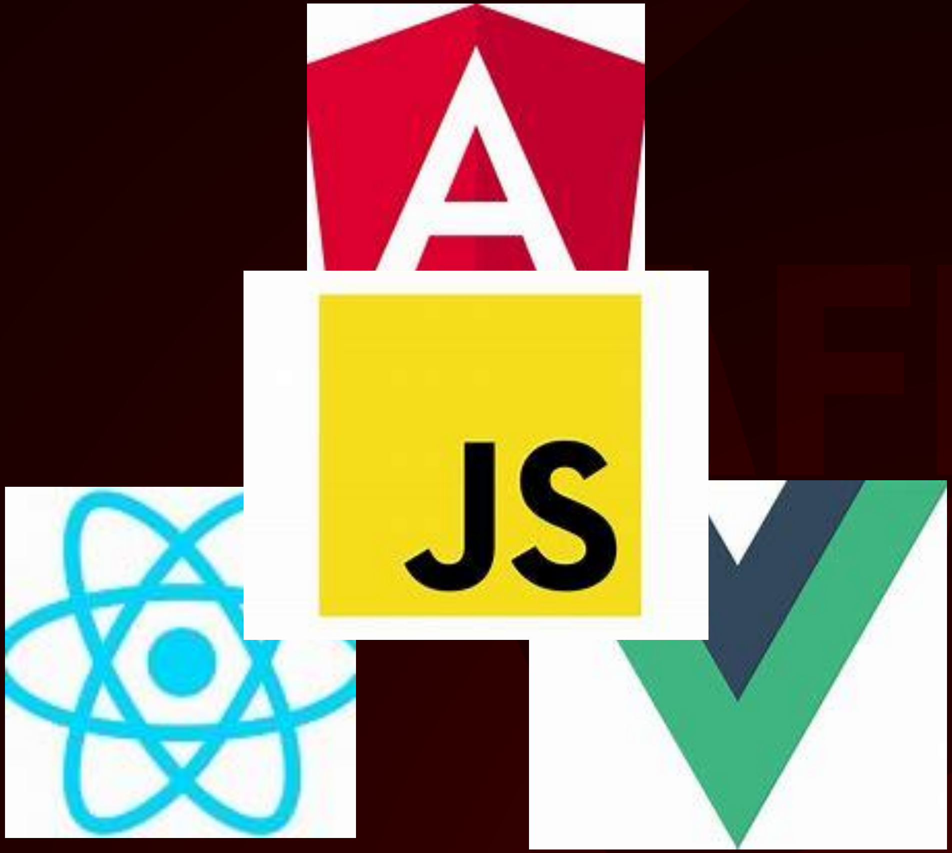
- birth: initialisation or mounting
- change: updated or onchange
- cleanup: beforeDestroy, willUnmount
- death: destroy

The final stage ensures state, events and requests from components don't conflict.



Big 3 JS Frameworks & Props

- the concept of a props object exists in all 3 frameworks (props/inputs/attributes)
- these props flow down from parents to child
- children can use but not change props
- in all 3 frameworks props are really ust JS objects.



Big 3 JS Frameworks & State

- all manage state at a component level

- changes in state re-render the DOM automatically

Challenge 1

- revise your JS Fundamentals
- fix the broken code in the 7 files for challenge 1

Challenge 2

- using the JS fundamentals
- follow the instructions for challenge 2
- create your react project
- fix the bugs

ICT40120 Certificate IV in Programming