REACT - INTERVIEW PREP

Agenda:

- Class based Components
- Lifecycle of Components
- Functional v/s Class based
- Higher Order Components (HOC)
- Virtual DOM & React rendering

```
class Welcome extends React.Component {
    render() {
        return <div> ... </div>.
    }
}
```

render() method in class method returns JSX from it

Component [React.Component]

↳ by extending this, Welcome Class

Becomes a Class Component

Functional Comp.

```
Const [btnText, setBtnText] = useState('Click')
Const [Count, setCount] = useState(0)
```

Function to update.

```
Constructor() {
    this.State = {
        Count : 0,
        btnText : 'Click'
    }
}
```

To update State in Class Based Component,

```
this.setState({count : this.state.Count + 1})
```

```
Class Welcome extends React.Component {
```

```
Constructor (props) {
    super (props)
    this.state = {count : 0}
}

render ()  {

}

}
```

Constructor → Special method used to initialize instance of Welcome class

Constructor(props) → Props' Parameter is used to access properties passed from parent.

Super(props) → Calls Constructor of the parent class

this.state = State initialization

{
  ;
}

render → return JSX.

State

this.state : {count: 0}

1) State Initialization

2) State Access → this.state.count

3) State Update

this.setState({count: this.state.count + 1})

Lifecycle Methods

Functional. ← Lifecycle

```
useEffect(() => {

  ...

  return () => { ... }

}, [count]

)
```

useEffect

[
1. onMount (init)
2. onUpdate
3. onUnMount (Destroy)
]

1. componentDidMount
2. componentDidUpdate
3. componentWillUnmount

★★★ • State update in React is Asynchronous.

- React batches state updates to be effective & minimize re-rendering.

Sequence & Lifecycle Event

1. Constructor — Initialize the state
2. Render — Render initial UI
3. ComponentDidMount — Fetches todo items after 1s, state updates.

4. render → runs again due to state change. & updates UI
5. Component Did Update → runs on state change. Logs.
6. render → runs again on new todo added.
7. Component Will Unmount → leave from DOM.

	Class	Functional
State mgmt	this.state this.setState	useState
Side effects	Lifecycle methods CDM, CDU, CWU	useEffect
Syntax	ES6 syntax	Functions
Structure	More boilerplate	More concise

More verbose | Less boilerplate

Why Functional Components are better?

1. Simplicity / Readability
2. Hooks
3. Avoid 'this' keyword
4. Better performance
   - they are Stateless by default
   - with hooks, manage state
5. Logic encapsulated
   - ↳ Hooks allows to reuse & encapsulate logic easily

Higher Order Components

[HOCs]

HOC → is a function that takes a Component and returns a new Component.

```
Const EnhancedComponent = hOC(WrappedCompone
```

returns a new component

fn

Component as a parameter

Advantages

HOC
- Code reusability
- Separation of concerns

When not to use HOC ⇒

(1) Overuse leads to complexity

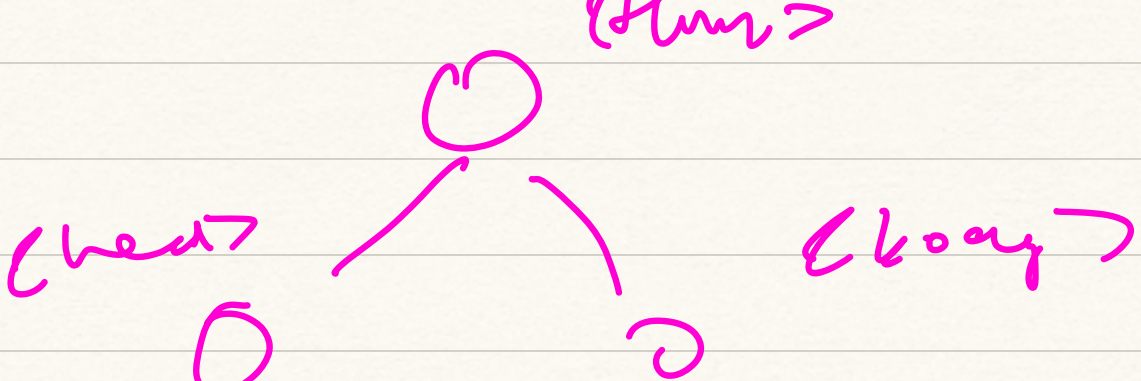
(2) Performance Concerns

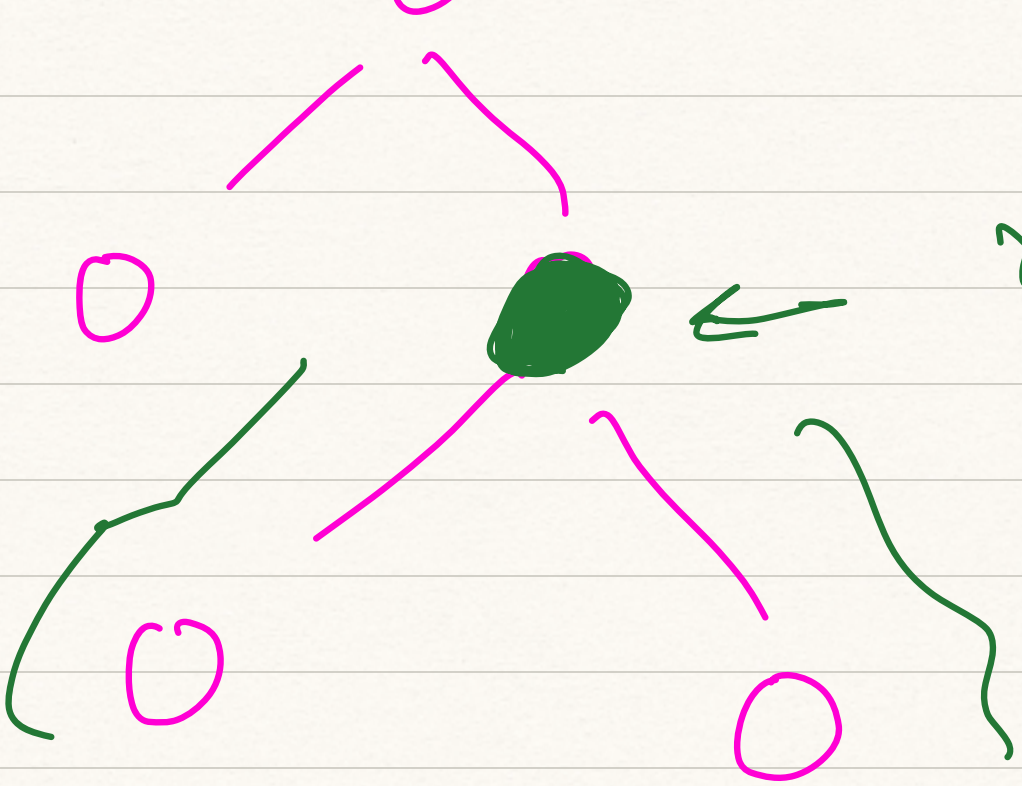
VIRTUAL DOM

Real DOM : Document Object Model

↓

represent UI State





re-renders

to

update UI

Re-rendering makes UI slow.

Inefficient DOM updates.

Virtual DOM

Javascript

Every DOM object ⇒ Virtual

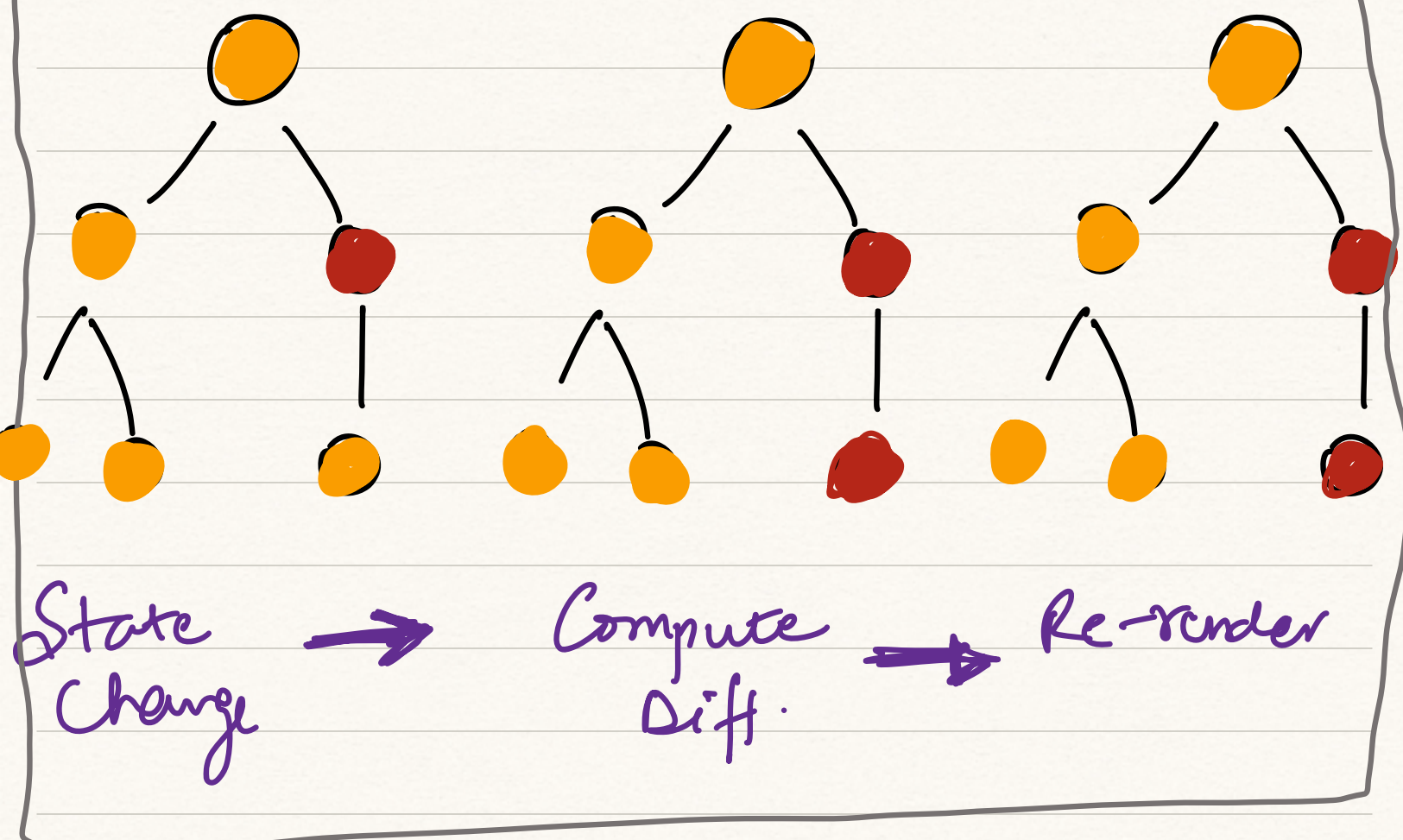
DOM object

(lightweight copy)

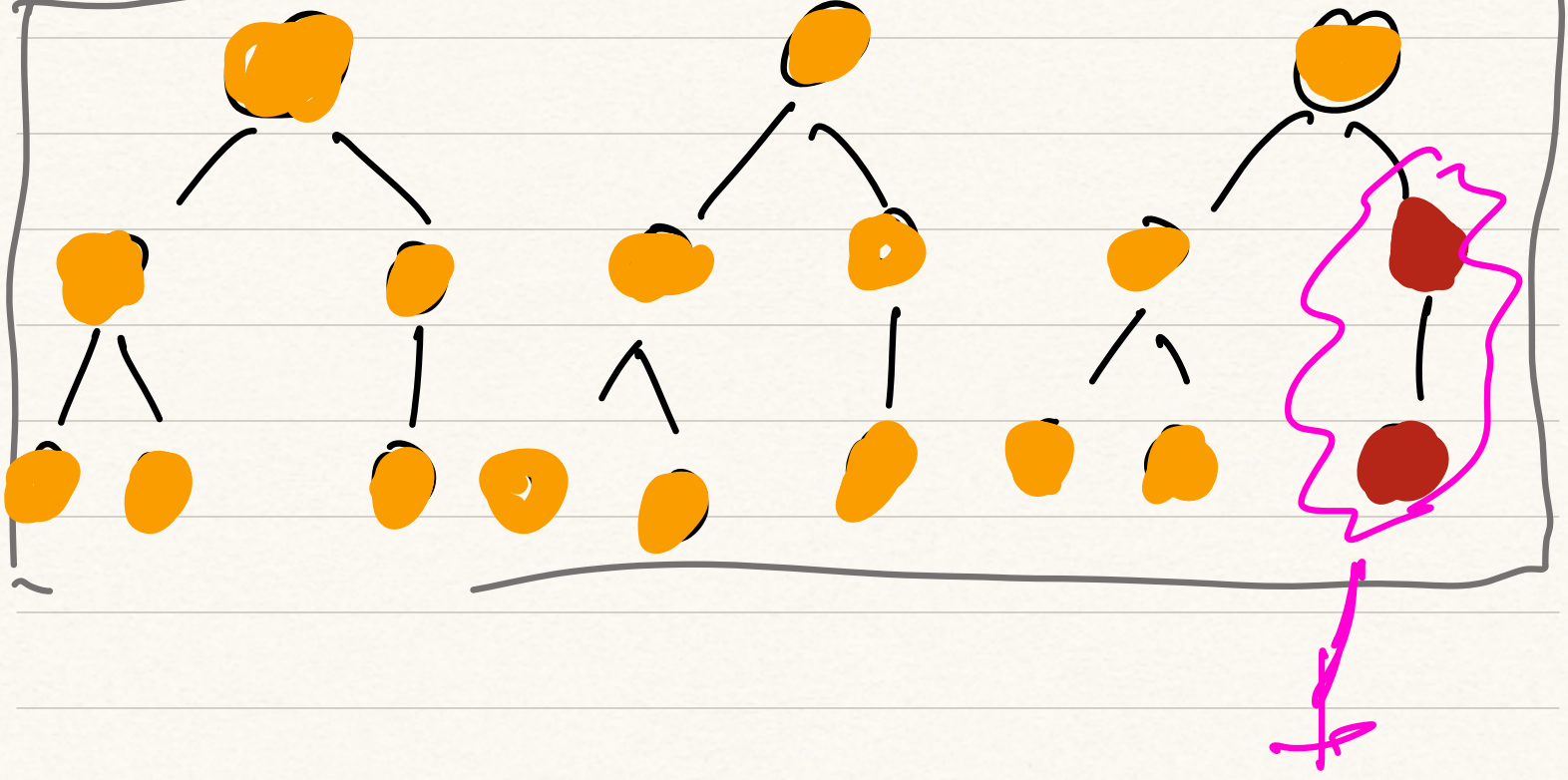
* Manipulating Virtual DOM is fast

- Nothing gets rendered on screen.

VIRTUAL DOM

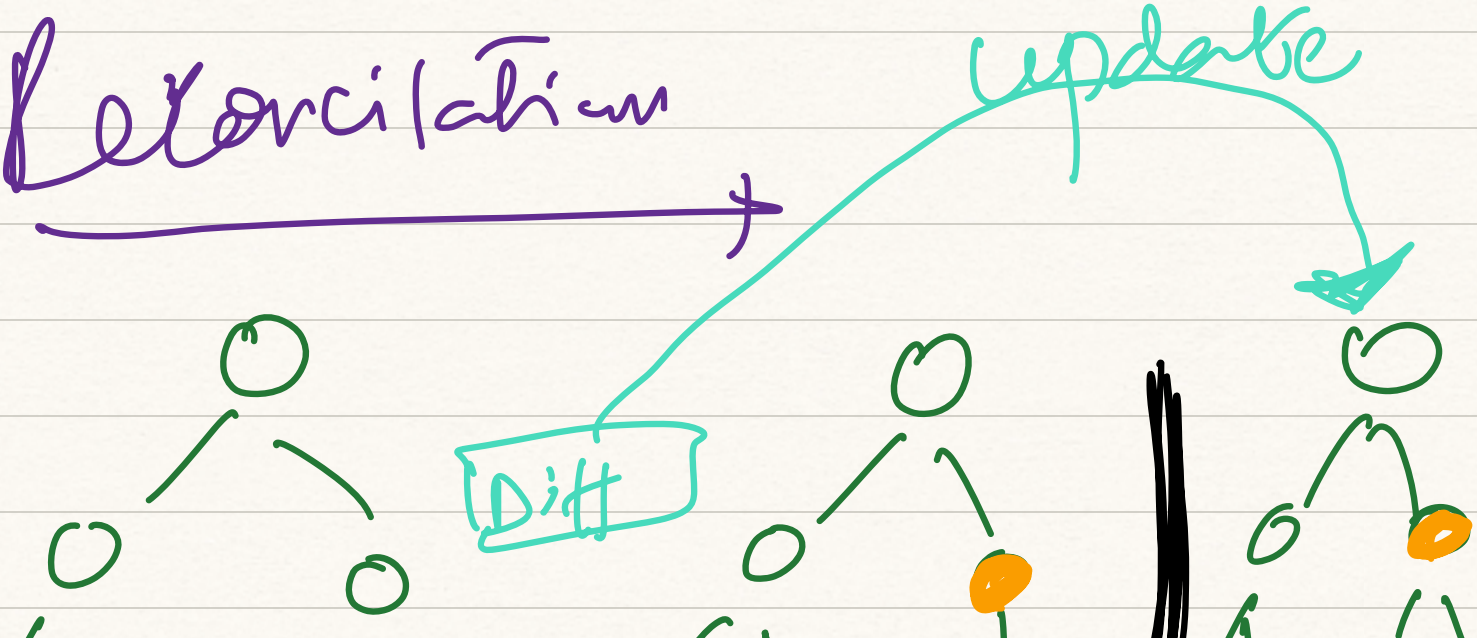


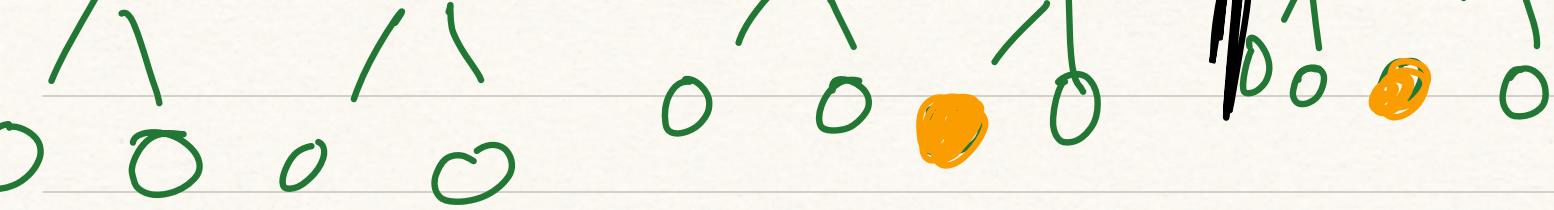
BROWSER DOM



Whole parent subtree gets re-rendered to give updated UI

Reconciliation





V. DOM

New VDOM

Browser DOM

(Process of Reconciliation)