## ✅ Core Deep Concepts to Master

### 1. ****Virtual DOM & Reconciliation****

* **Virtual DOM**: A lightweight JS representation of the real DOM. React updates this first and then **diffs** the new vs old VDOM (a process called **reconciliation**) to compute the minimal set of changes to the real DOM.
* **Misconception**: React doesn’t always only update the changed node—it **re-renders the entire component** if its props/state change. Only the **DOM update** is optimized.
* **Why whole form re-renders**:
  + If the Form component has internal state or props that depend on the input, changing an input can trigger a re-render of the form.
  + React does **component-level re-renders**, not element-level.
  + If you don’t isolate the Input with React.memo or split into child components, the parent Form re-renders.

**Interview Q:** If React uses virtual DOM, why do components still re-render so frequently?  
**Answer:** Because reconciliation determines what to update on the DOM, but rendering a component (i.e., running its function) is different. React still has to re-run components' render logic to compare virtual DOMs.

### 2. ****Component Re-renders & Optimization****

* **React.memo()**: Prevents re-rendering of a functional component unless its props change.
* **useCallback & useMemo**:
  + useCallback: Returns a memoized callback.
  + useMemo: Returns a memoized **value**.
  + Both help avoid unnecessary child re-renders.

js

CopyEdit

const memoizedFn = useCallback(() => doSomething(), [deps]);

**Interview Q:** When would you use useMemo and when React.memo?

### 3. ****Cleanup in**** useEffect

* **Purpose**: Prevent memory leaks or stale effects.
* Examples:
  + Clearing timeouts/intervals.
  + Unsubscribing from events or WebSockets.

js

CopyEdit

useEffect(() => {

const id = setInterval(logData, 1000);

return () => clearInterval(id); // cleanup

}, []);

**Interview Q:** What would happen if we forget to add a cleanup in a useEffect with an event listener?

### 4. ****State Batching and Event Loop****

* React batches state updates for performance (in synthetic events and lifecycle methods).
* **Non-synthetic async code (e.g., setTimeout)** may not be batched unless you're using **React 18+ automatic batching**.

js

CopyEdit

setTimeout(() => {

setCount(c => c + 1);

setFlag(f => !f);

}, 1000);

**Interview Q:** How does React’s event loop interact with state batching?

### 5. ****React 18 Features****

* **Automatic Batching**: setTimeout/fetch state changes are now batched.
* **Concurrent Rendering**: React can interrupt rendering and work on multiple tasks simultaneously.
* **useTransition**: For marking state updates as non-urgent.

### 6. ****Keys in Lists (And Why They're Important)****

* React uses key to identify which items have changed, added, or removed.
* Bad keys (like index) lead to bugs in dynamic lists (e.g., losing focus on input fields).

**Interview Q:** Why shouldn’t we use array index as a key?

### 7. ****Controlled vs Uncontrolled Components****

* Controlled: value is managed via React state.
* Uncontrolled: Use ref to get value from DOM directly.

js

CopyEdit

<input value={input} onChange={e => setInput(e.target.value)} />

**Interview Q:** Which is better for performance and why?

### 8. ****Context API & Prop Drilling****

* Context avoids prop drilling but may cause unnecessary re-renders.
* Use useContextSelector or memoization to optimize.

**Interview Q:** What are the downsides of the Context API for deeply nested trees?

### 9. ****Custom Hooks****

* Abstract repeated logic across components.

js

CopyEdit

function useLocalStorage(key, defaultValue) {

const [value, setValue] = useState(() => localStorage.getItem(key) || defaultValue);

useEffect(() => {

localStorage.setItem(key, value);

}, [key, value]);

return [value, setValue];

}

**Interview Q:** When would you write a custom hook instead of a utility function?

### 10. ****Error Boundaries****

* Used to catch JavaScript errors in the component tree.
* Only works with **class components** (you can wrap functional components).

jsx

CopyEdit

class ErrorBoundary extends React.Component {

state = { hasError: false };

static getDerivedStateFromError(error) { return { hasError: true }; }

componentDidCatch(error, info) { logError(error, info); }

render() { return this.state.hasError ? <Fallback /> : this.props.children; }

}

### 11. ****React Suspense & Lazy Loading****

* Load components or data **asynchronously**.

js

CopyEdit

const LazyComponent = React.lazy(() => import('./HeavyComponent'));

**Interview Q:** How does Suspense help improve performance and user experience?

### 12. ****Refs and Forwarding Refs****

* useRef stores mutable values and DOM references.
* Forwarding refs allows parent to access child's DOM node.

js

CopyEdit

const MyInput = React.forwardRef((props, ref) => <input ref={ref} {...props} />);

## 🧠 Bonus Topics (Senior-level)

* **Render Props vs HOC vs Hooks**
* **React Fiber architecture** (scheduler for concurrent rendering)
* **React DevTools performance profiling**
* **Server-Side Rendering (Next.js)**
* **Hydration**
* **React Testing Library best practices**
* **How React handles re-renders at the scheduler level**