ReactJS - HOOKS

1

What are HOOKS ?

Hooks are hooked into functional components in ReactJS.

Any function that starts with ‘use’ is a hook.

Difference between Local Variables and States?

Local Variable

- **Local variables don’t persist between renders.**
- **Changes to local variables won’t trigger renders.**

State Variable

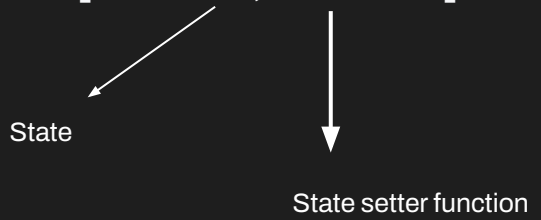
1. A **state variable** to remember the data between different renders.
2. A **state setter function** to update the variable and trigger React to render the component again.

Using State Variable !

```
import { useState } from 'react';
```

Syntax:

const [variable, function] = useState(initialValue)



State updations !

- `setCnt(cnt+1)`
- `setCount(prev => prev + 1);`
-
```
setCnt(prev=>{

    return prev+1;

})
```

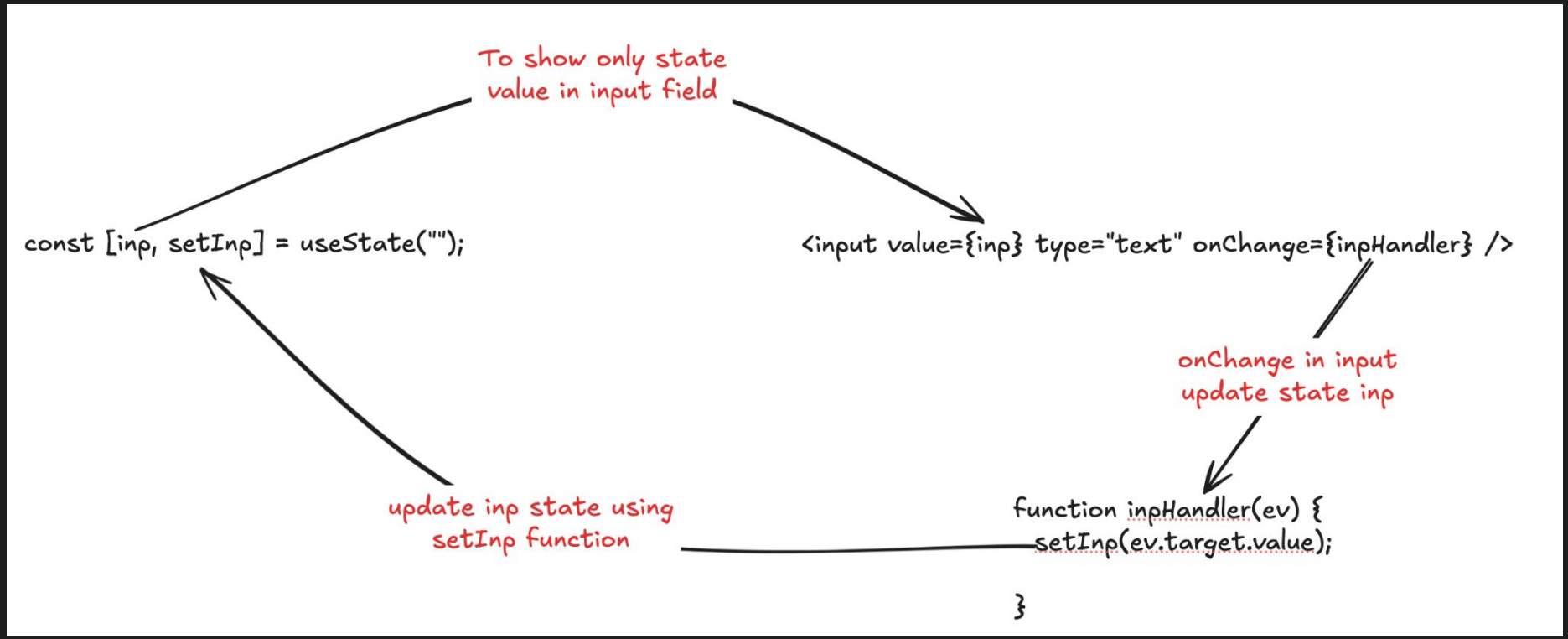
ReactJS - Taking Input

2

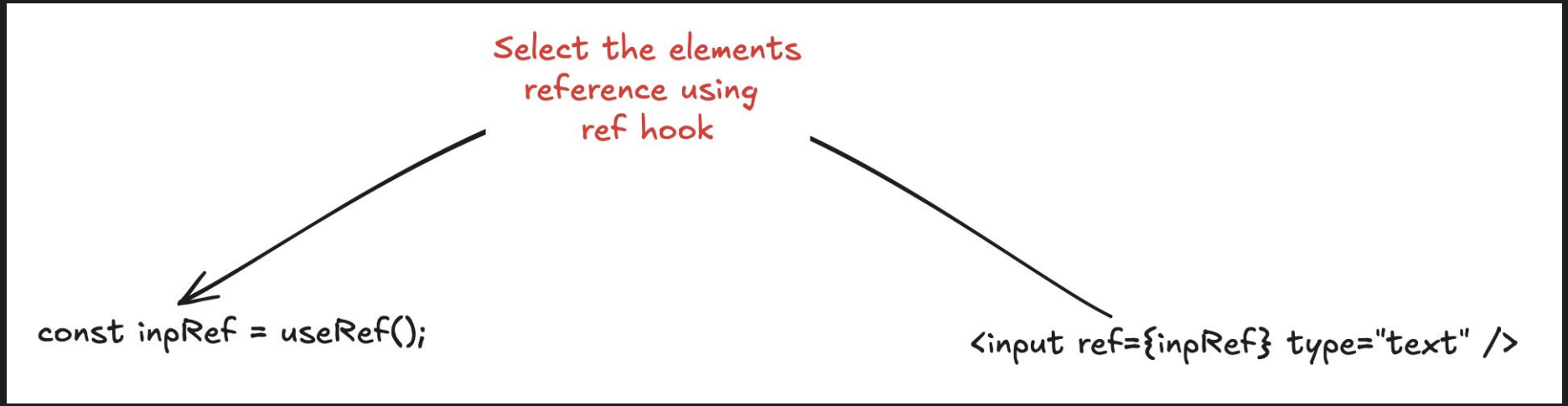
Two ways to take input:

1. `useState`
2. `useRef`

useState working with input!



useRef working with input!



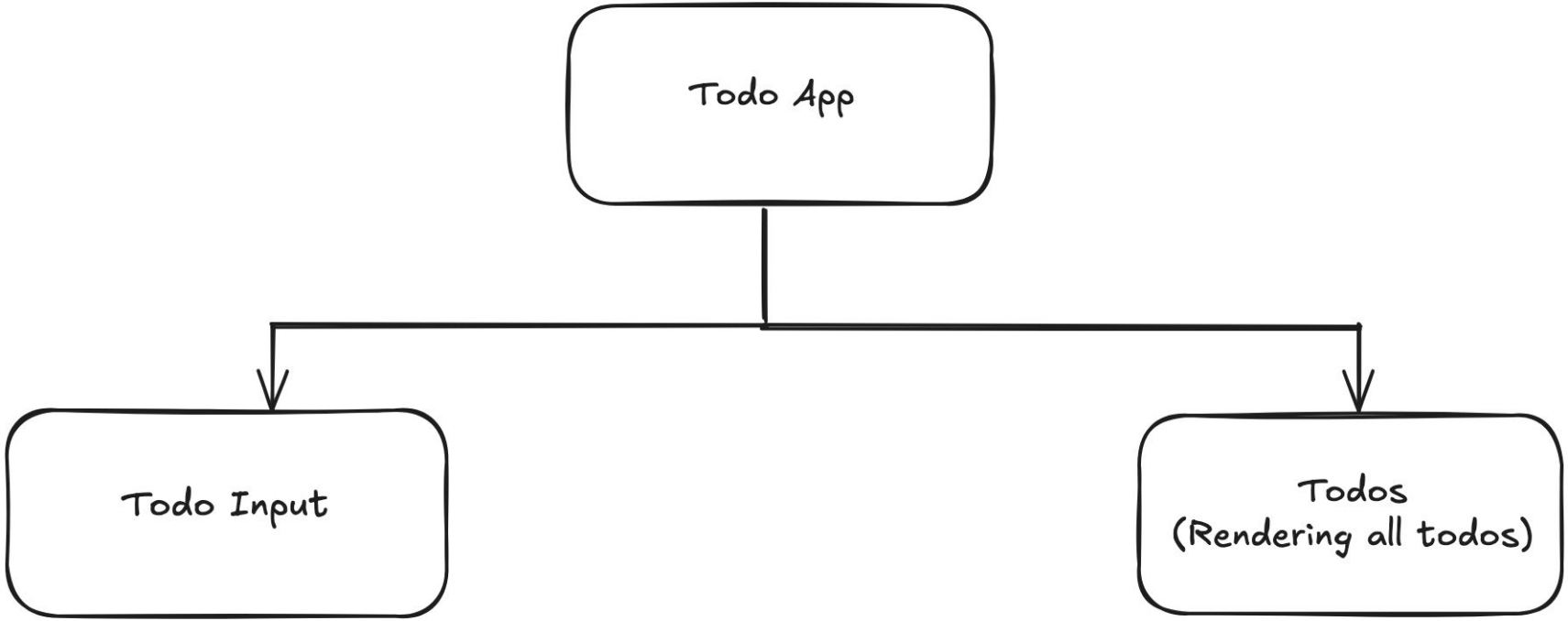
ReactJS - TodoApp

3

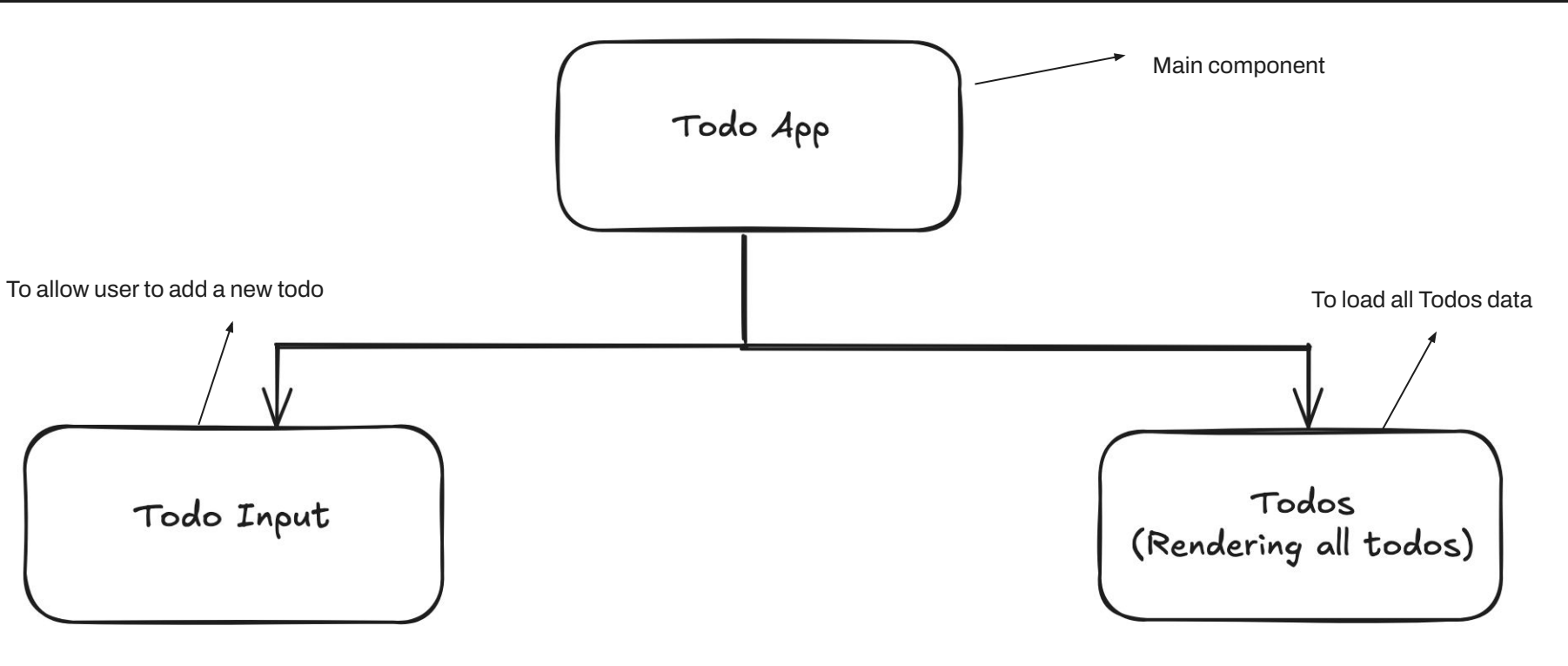
Tasks todo in application?

1. Take input of todos from the user.
2. Show the todos on the browser.
3. When a new todo is added or deleted update the data on browser.
4. Functionality to change the priority of Todos.

Component Structure

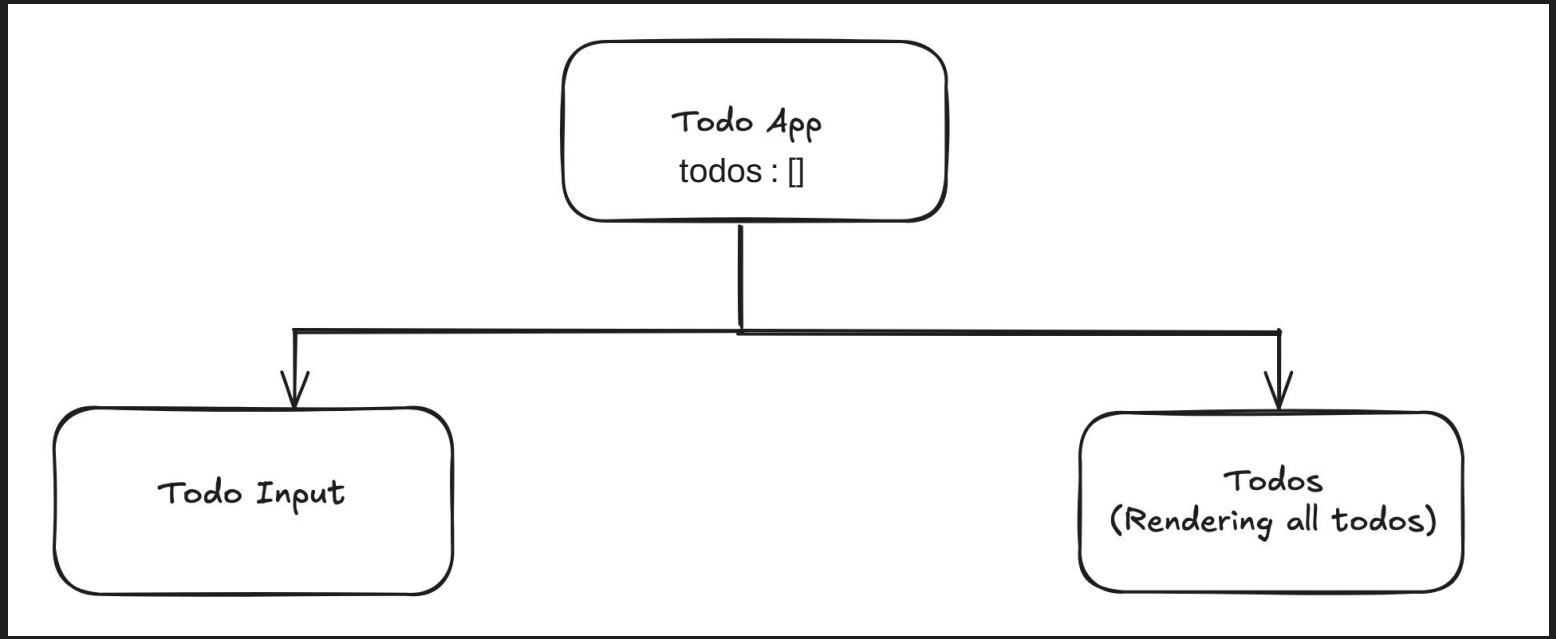


Component Structure



Todos State?

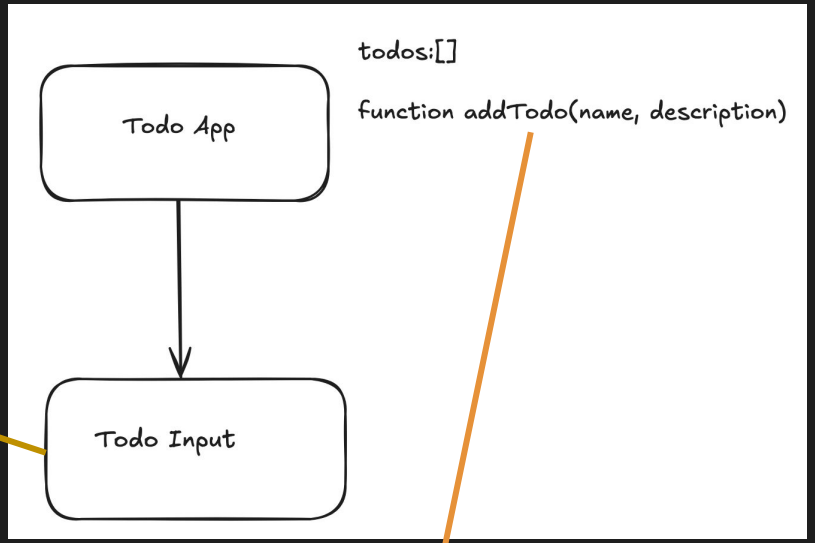
LCA can store the state, since we need to access state in Todo Input and Todos to render list as well.



Bidirectional Data Flow - Data from child to parent !

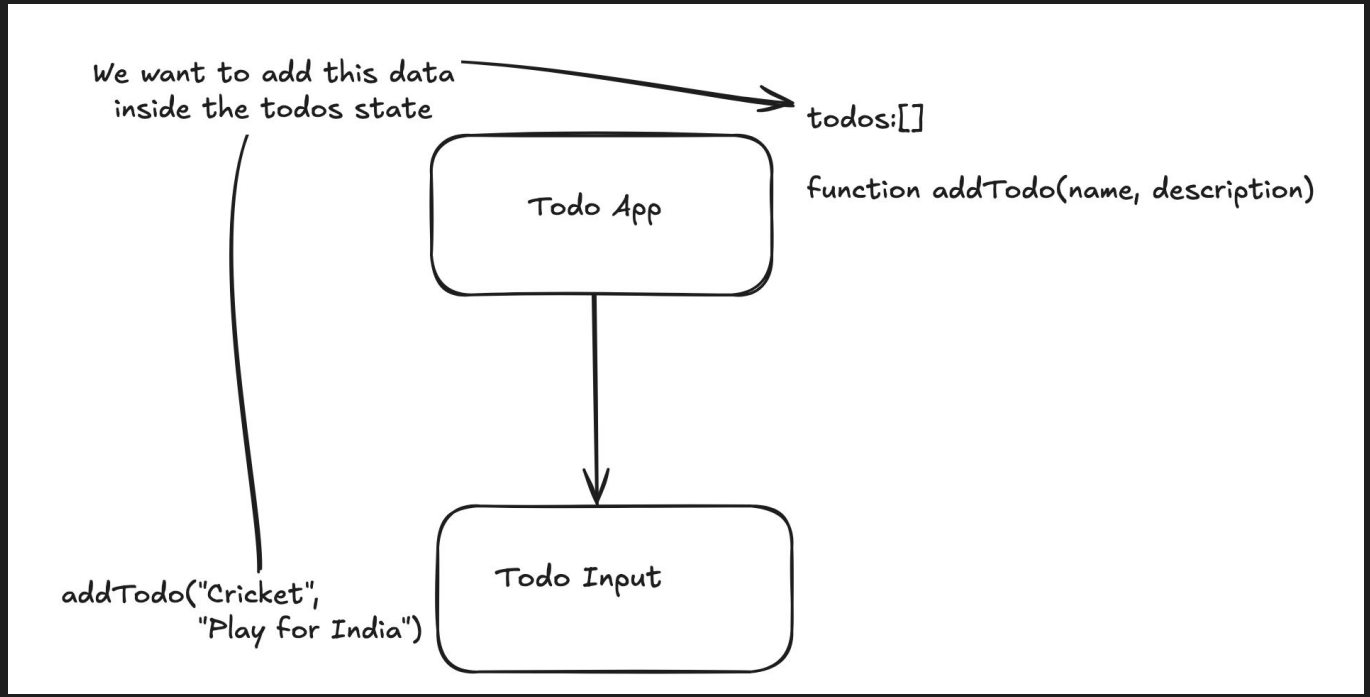
```
function TodoInput() {
 return (
   <div>
     <input type="text" placeholder="Enter task name" />
     <input type="text" placeholder="Enter description for task" />

     <button>Add Todo</button>
   </div>
 );
}
```

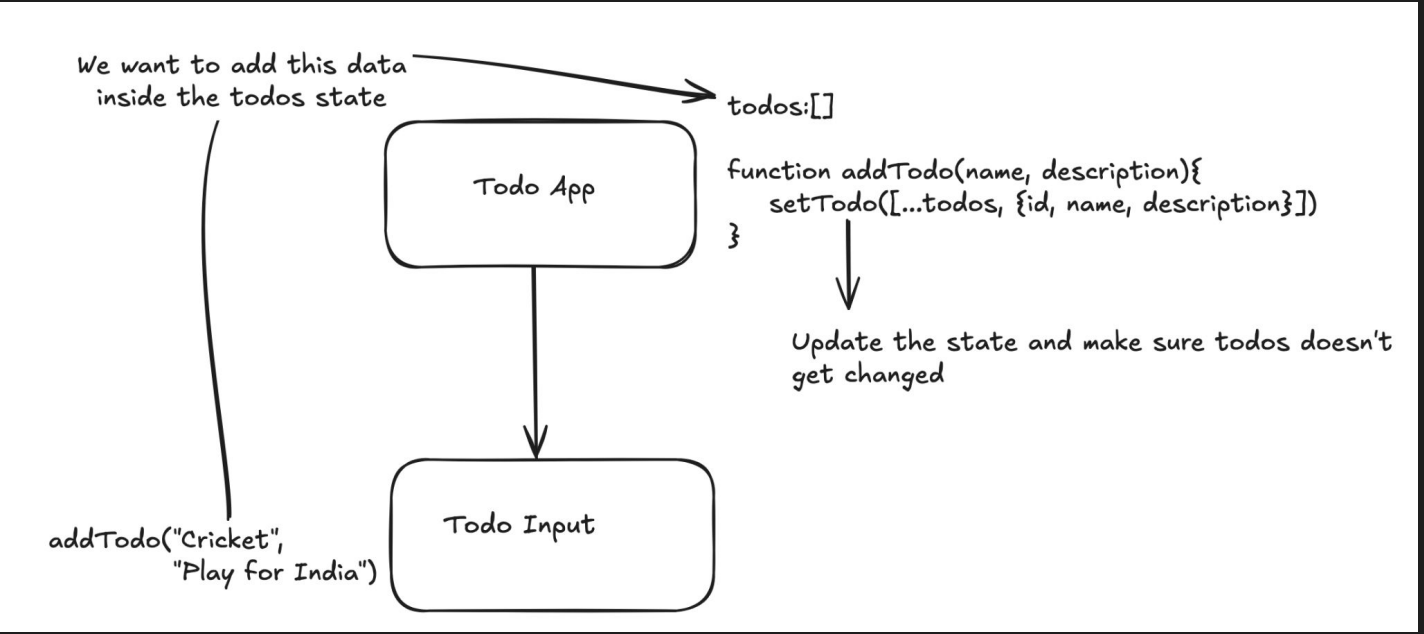


We create a function in the parent component only through which child can provide data back to parent.

Bidirectional Dataflow?



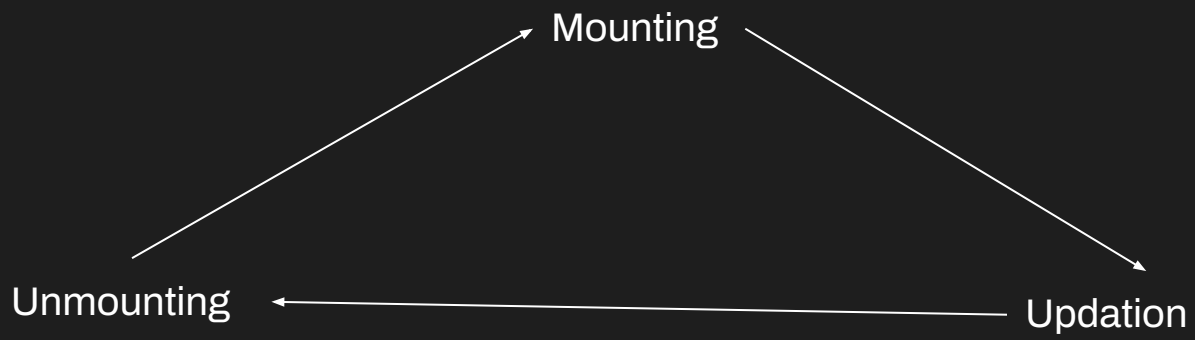
Bidirectional Dataflow?



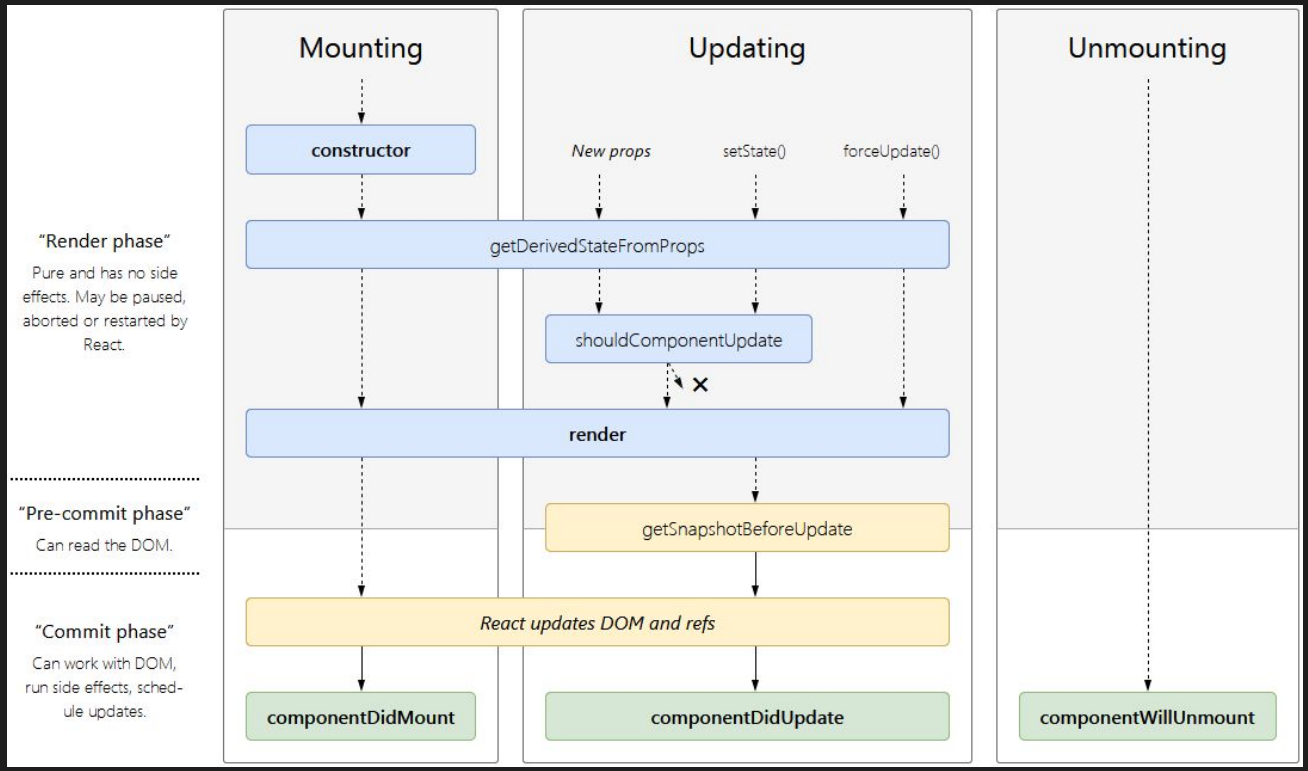
Understanding Life-Cycle, Side-Effects and Hooks

1. Life Cycle
2. Side-Effects

Life-Cycle



Life-Cycle Methods



Where does useEffect comes into the picture?

Lifecycle Methods	Hook	Renders
componentDidMount	`useEffect(()=>{ ... }, [])`	after first render only
componentDidUpdate	`useEffect(()=>{... }, [dependency1, dependency2])`	after first render AND subsequent renders caused by a change in a dependency
componentWillUnmount	`useEffect(() => { ... return ()=> {...cleanup}})`	
shouldComponentUpdate	no comparable hook, instead, wrap function component in `React.memo(List)`	renders only if a prop changes

Important hooks?

1. useEffect
2. useCallback
3. useMemo
4. useContext