HELLO REACT INDIA _

Reducing bugs in React codebase

Understanding anti-patterns in React

Darshita Chaturvedi
Co-Founder & CEO, Atri Labs

Shyam Swaroop
Co-Founder & CTO, Atri Labs

Darshita Chaturvedi

@darshitac_

- Maintainer of Atri engine, and Co-Founder & CEO of Atri Labs

  Creating a full-stack web development framework

- Prev: Graduate Research Assistant at MIT & Quant Researcher at BlackRock

Shyam Swaroop

@ShyamSwar

- Creator of Atri engine, and Co-Founder & CTO of Atri Labs

  Creating a full-stack web development framework

- Prev: Graduate Research Assistant at Columbia University & Software Engineer at EXL Services

Motivation

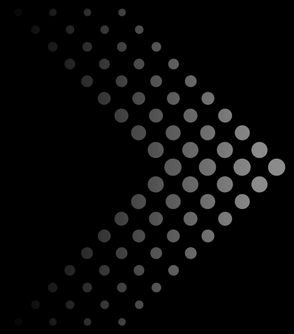
Many interactions in one page

Suboptimal code in one interaction can ruin the entire application

Code quality is paramount to support building production-grade applications

Requirement to build rigorous internal standards on code quality

Types of bugs



- Component does not update upon a user event
- Component updates partially upon a user event
- Component renders unexpectedly

Origin story of this talk

Central theme

React code should be independent of:

- sequence in which components will render
- when components will render.

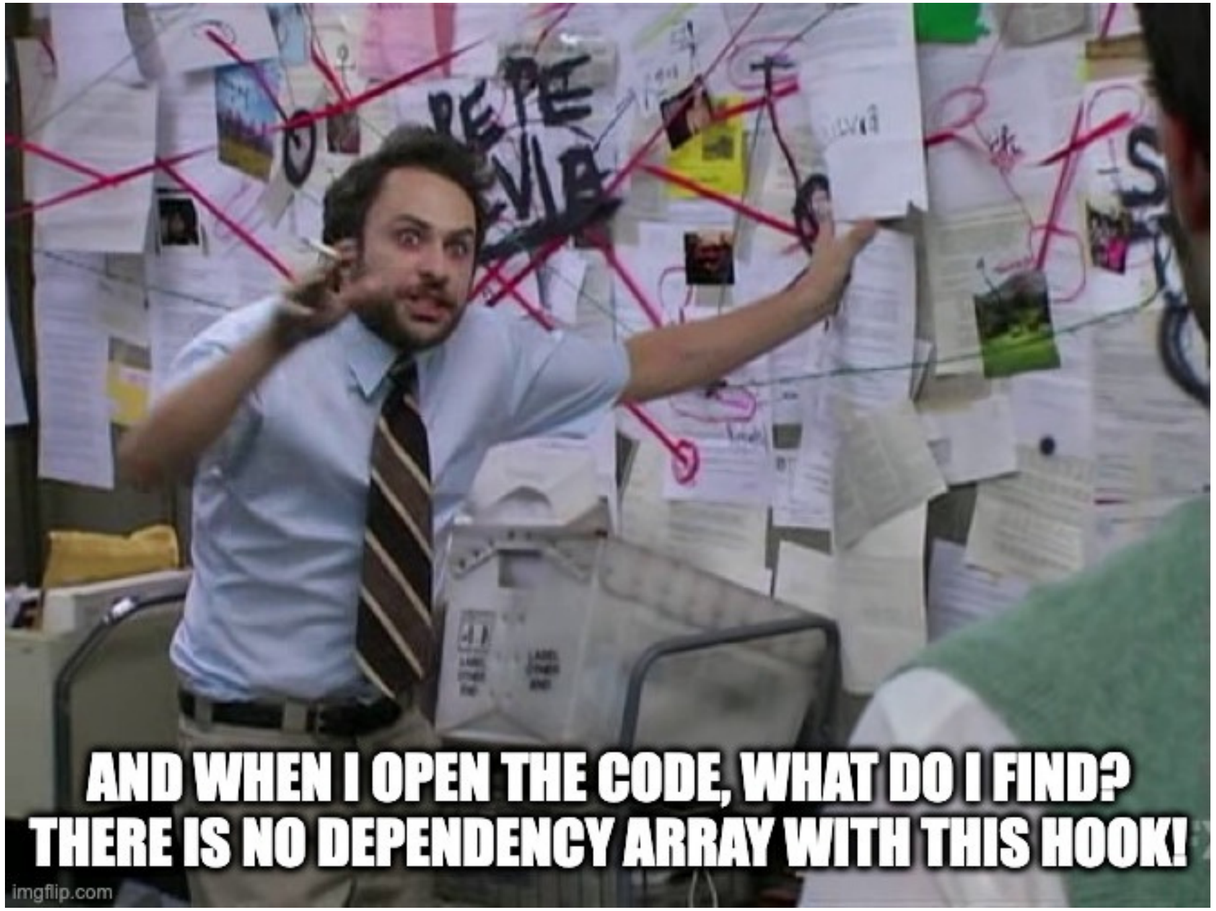
Hints

1 Hook without a dependency array

- In React, different pieces of code are linked to each other by dependencies.
- These pieces of interdependent code together keep the application state in its desired form.
- Therefore, if we use a hook that does not have a dependency, there is a high chance that it will result in bugs.

Be cautious while using hooks that do not take dependency arrays

Hints


AND WHEN I OPEN THE CODE, WHAT DO I FIND?
THERE IS NO DEPENDENCY ARRAY WITH THIS HOOK!
imgflip.com

What is a Pattern?

We characterize a React code as a good pattern if:

1. **Component is reusable**
2. **Code is easier to review and debug**

! Code is still considered a pattern if we wrote more lines of code or we (expectedly) introduced a few extra renders in order to achieve the above objectives.

Sandbox

https://codesandbox.io/s/react-anti-patterns-43w3sy?file=/src/pages/Incorrect.tsx

Moving away from useState

```tsx
const Slider: React.FC<SliderComponentTypes> = ({
  minValue,
  maxValue,
  givenValue,
}) => {
  const [startPosition, setStartPosition] = useState<{…
  } | null>(null);
  const [finishPosition, setFinishPosition] = useState({ x: 0, y: 0 });
  const [currentPosition, setCurrentPosition] = useState({ x: 0, y: 0 });
  const [thumbPosition, setThumbPosition] = useState("0%");
  const [value, setValue] = useState(0);

  function getPosition(e: React.MouseEvent) {…
  }

  const handleMouseMove = useCallback(
    (e: any) => {
      const { pageX, pageY } = getPosition(e);
      console.log(pageX, pageY);
      setCurrentPosition({ ...currentPosition, x: pageX, y: pageY });
      console.log("current posi", currentPosition);
    },
    [currentPosition]
  );
  const handleThumbMouseUp = useCallback(…
  const handleThumbMouseDown = useCallback(…

  useEffect(() => {…
```

```tsx
const Slider: React.FC<SliderProps> = (props) => {
  const onMouseDown = useCallback(
    (e: React.MouseEvent) => {
      const startPostion = { x: e.pageX, y: e.pageY };
      const onMouseMove = (e: MouseEvent) => {
        if (startPostion) {
          const delta = e.pageX - startPostion.x;
          props.onChange(props.value + delta);
        }
      };
      const onMouseUp = () => {
        // unsubscribe
        document.removeEventListener("mousemove", onMouseMove)
        document.removeEventListener("mouseup", onMouseUp);
      };
      // subscribe
      document.addEventListener("mousemove", onMouseMove);
      document.addEventListener("mouseup", onMouseUp);
    },
    [props]
  );
```

Atri framework

1 : 3

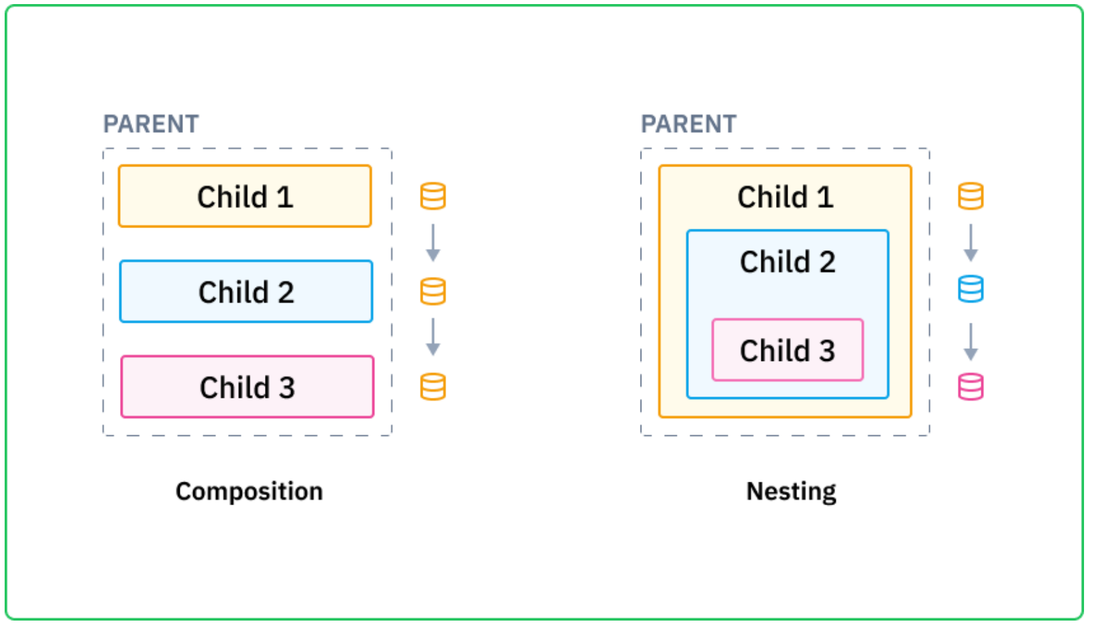
$$\frac{\#\ \text{useState}}{\#\ (\text{custom hooks} + \text{components})}$$

No. of components = 192
No. of custom hooks = 112
No. of useState in components = 55
No. of useState in custom hooks = 50

Hints

2 Nesting can lead to duplication of states

- While using nesting, we often provide duplicate states to the children. This can introduce bugs.
- Moreover, if we observe that there is a bug in *Child 3*, the time complexity of debugging in nesting is O (n) where n is the number of children above *Child 3*. By contrast, the time complexity of debugging in composition is O (1).



Avoid nesting since it may introduce bugs and make debugging harder.

Hints

3 Using uncontrolled components

- Using controlled components reduced issues in our codebase by more than 80%

Make use of controlled components wherever possible

So far...

Hooks without dependency array
Nesting approach to arrange components

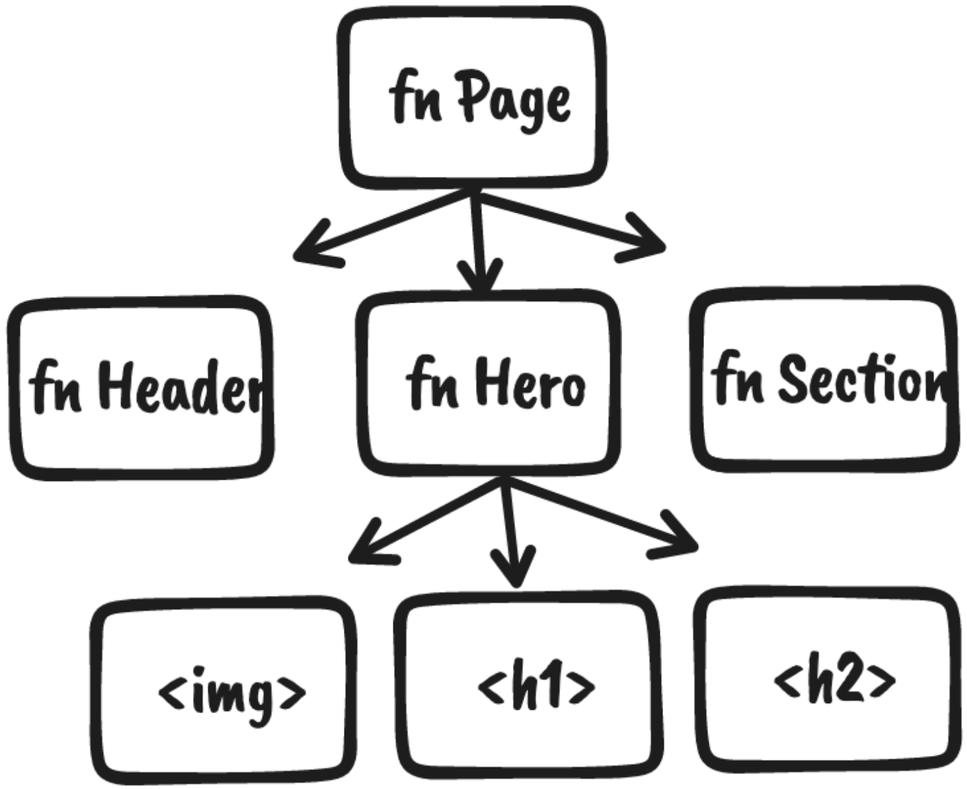
props or context as initial state
Destroy and Recreate

Use props in JSX
Use the context of the event handler to store state
props as a dependency in useMemo (recommended)
Use controlled component

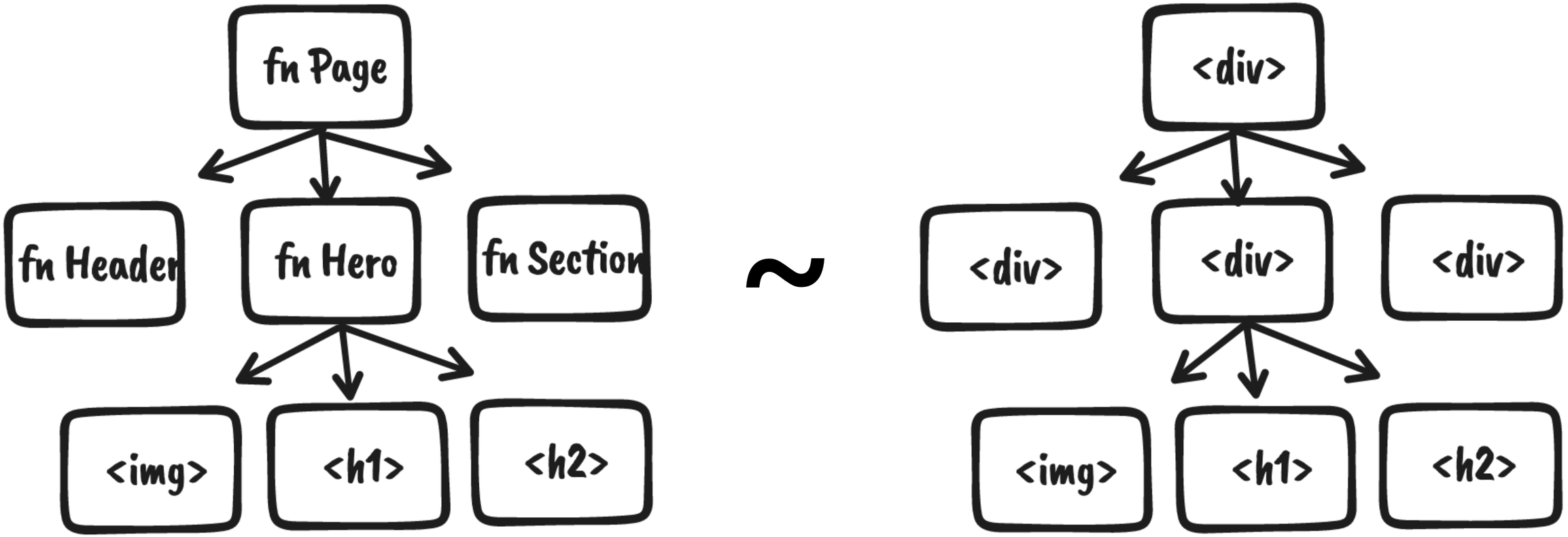
React code should be independent of:

- **sequence in which components will render**
- **when components will render.**

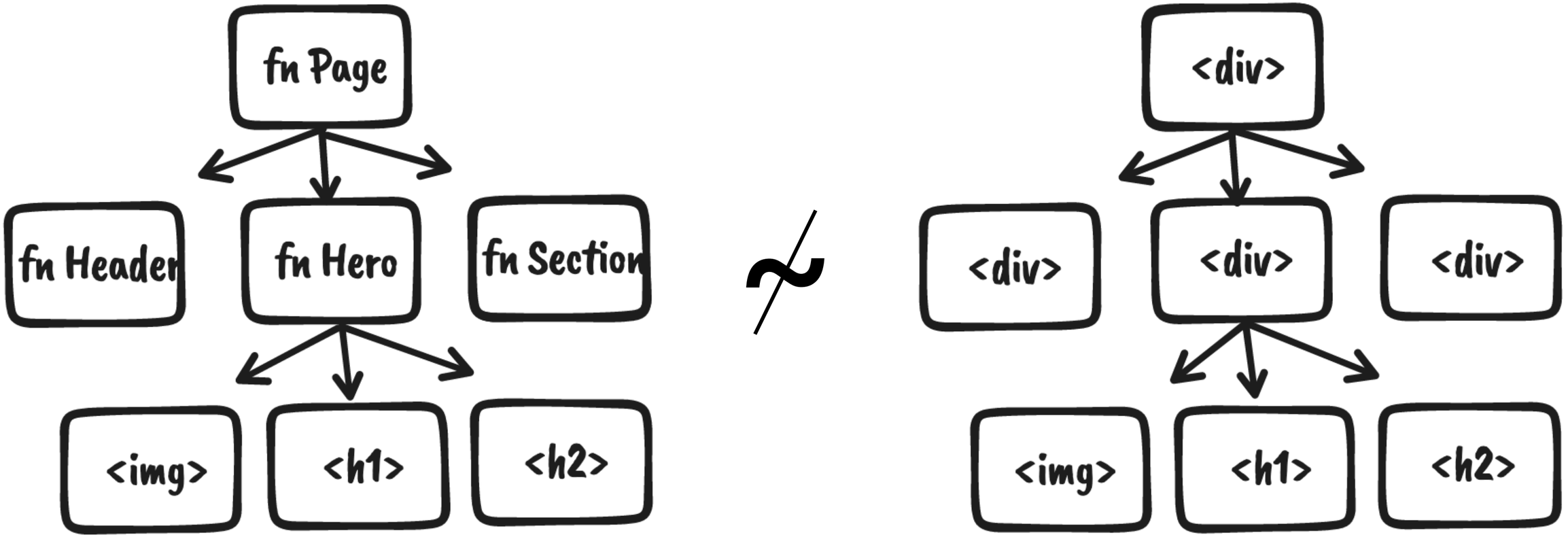
React Mental Model (1)



React Mental Model (2)



React Mental Model (3)



Why are they not similar?

- JSX Element creates React Nodes
- JSX Element gets rendered again, not React Node
- {props.children} won't render again when the parent renders assuming children is a ReactNode

Limited responsibility

- **Each component should have limited responsibility**

  For example, a component that fetches data should not be responsible for UI and vice versa

- **useEffect should be close to the top in the component hierarchy**

Thank you for listening!

 Darshita Chaturvedi

 @darshitac_

 Shyam Swaroop

 @ShyamSwar

https://github.com/Atri-Labs/atrilabs-engine