Killing bugs with kindness

Understanding anti-patterns in React

Darshita Chaturvedi

Co-Founder & CEO, Atri Labs

React Advanced London 2022

October 24, 2022

About Me

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

  Creating a new full-stack web development framework

- **Previously, Graduate Research Assistant at MIT and Quant Researcher at BlackRock**

  Experiences in academia and quantitative finance industry

- **Editor of novels written by my grandfather over the last 50 years**

  Favorite past times: painting and learning art history

Motivation

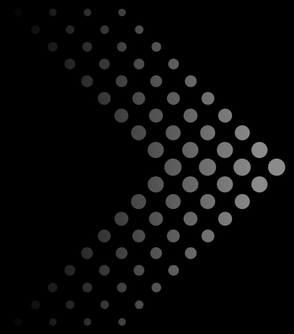
Many interactions in one page

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

Debugging like a developer!

Central theme

React code should be independent of:

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

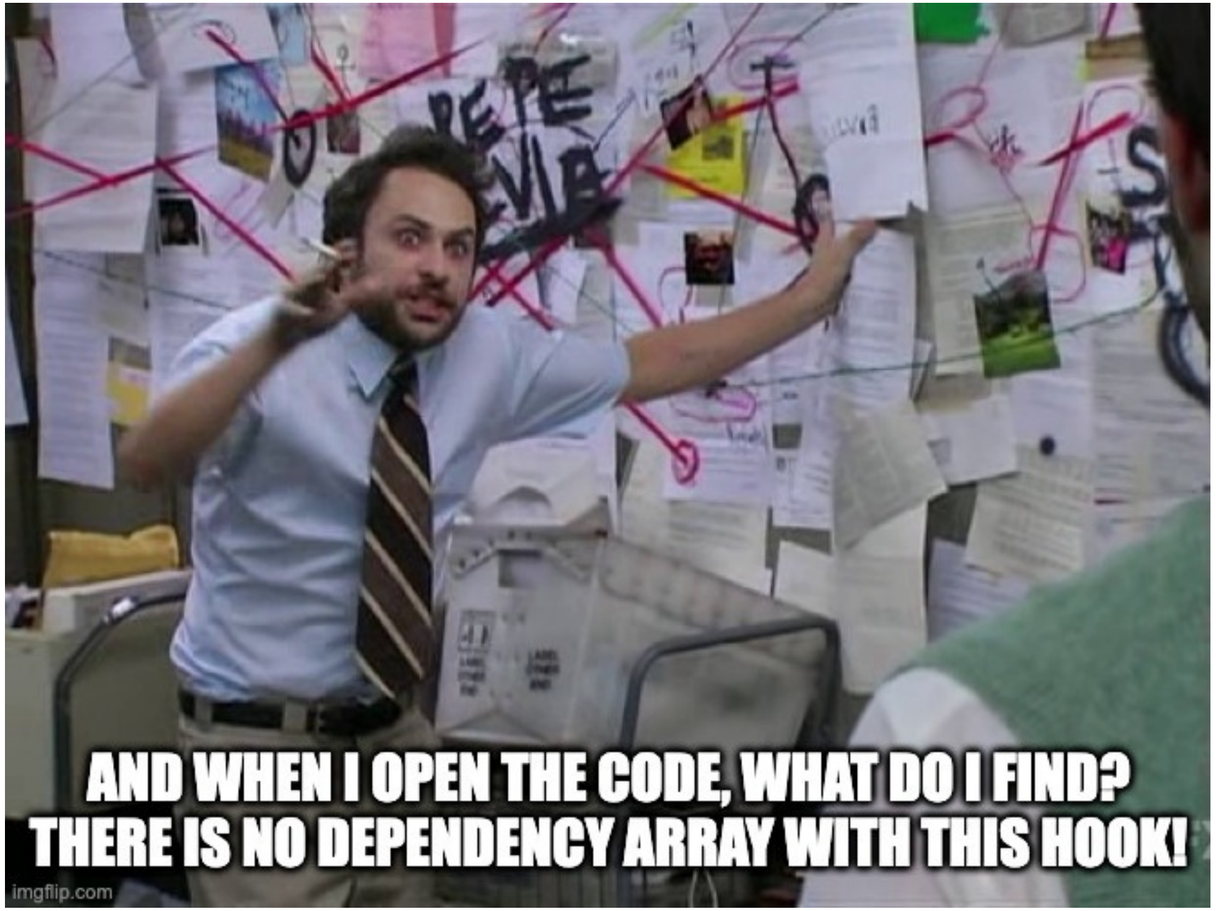
Hints

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

Caaarol! I gotta talk to you about React hooks!



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

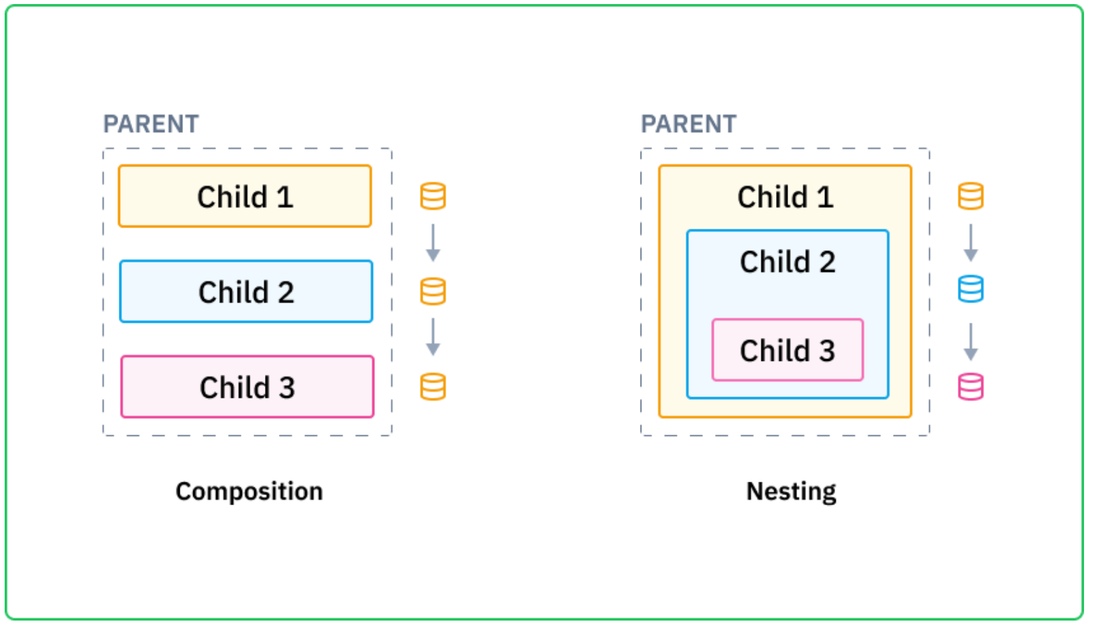
$$\frac{\#\ \text{useState}}{\#\ (\text{custom hooks} + \text{components})}$$

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

Key Takeaways

- Hooks without dependency array
- Nesting approach to arrange components

- props in context or 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.**

Thank you!

Darshita Chaturvedi

@darshitac_

Blog: https://medium.com/better-programming/how-we-reduced-bugs-in-our-react-code-base-9a7a979b4442

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

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