EcomLabs

STEP-BY-STEP GUIDE

useEffect for derived state

Five before-and-after examples for removing the useEffect calls that only copy data: cart total, filters, full name, sorting and counts. Each one measured in Chromium with React 19, plus when useMemo is worth it.

FOR	React 18 or 19, with or without Next.js
TESTED WITH	React 19.0 · Chromium 151 · Playwright 1.62 · Windows 11
UPDATED	October 30, 2026
DOCS	react.dev/learn/you-might-not-need-an-effect
ARTICLE	ecomlabs.dev/g/effect-en
AUTHOR	Luis Morales · @luismoralesdev

Contents

Before you start	The rule, and how we measured it
Step 01	Find the Effects that only copy data
Step 02	Cart total
Step 03	Filters
Step 04	Full name
Step 05	Sorting
Step 06	Counts
Step 07	Add useMemo only after measuring
When you finish	Checklist

BEFORE YOU START

The rule, and how we measured it

If a value can be calculated from props or state, do not store it in more state or update it with `useEffect`: compute it during render. With the Effect, React renders with the stale value, commits that to the DOM, runs the Effect and renders again.

The five examples in this guide are real components. We bundled them with esbuild and React 19.0, ran them in Chromium 151 with Playwright and counted renders after a real click or keystroke, without StrictMode. In all five, the Effect version took 2 renders on mount and 2 on update; the version computed during render took 1 and 1. The final text was the same: the difference is the extra render and the in-between commit, which shows stale data.

STEP 01

Find the Effects that only copy data

Search for every `useEffect` and look at the next line. We ran it on the examples file; in your project, use `grep -rn -A1 "useEffect(() => {" src`:

```
$ grep -n -A1 "useEffect(() => {" examples.jsx
6:  useEffect(() => {
7-    setTotal(items.reduce((sum, item) => sum + item.price * item.qty, 0));
--
21:  useEffect(() => {
22-    setVisible(products.filter((p) => p.category === category && p.stock > 0));
```

If the Effect only calls a setter with something calculated from props or state, it is a candidate. If it talks to the network, the DOM or an outside widget, leave it.

STEP 02

Cart total

```
// 1. Cart total — before
export function CartTotalBefore({ items }) {
  const [total, setTotal] = useState(0);
  useEffect(() => {
    setTotal(items.reduce((sum, item) => sum + item.price * item.qty, 0));
  }, [items]);
  return <p>{items.length} items · total {(total / 100).toFixed(2)}</p>;
}

// 1. Cart total — after
export function CartTotal({ items }) {
  const total = items.reduce((sum, item) => sum + item.price * item.qty, 0);
```

```
  return <p>{items.length} items · total {(total / 100).toFixed(2)}</p>;
}
```

With the Effect, adding an item made the first commit show "3 items · total 12.50" when the total was already 15.00.

STEP 03

Filters

```
// 2. Filters — before
export function ProductListBefore({ products, category }) {
  const [visible, setVisible] = useState([]);
  useEffect(() => {
    setVisible(products.filter((p) => p.category === category && p.stock > 0));
  }, [products, category]);
  return <ul>{visible.map((p) => <li key={p.id}>{p.name}</li>)}</ul>;
}

// 2. Filters — after
export function ProductList({ products, category }) {
  const visible = products.filter((p) => p.category === category && p.stock > 0);
  return <ul>{visible.map((p) => <li key={p.id}>{p.name}</li>)}</ul>;
}
```

With the Effect, the list starts empty and the first commit on mount shows no products.

STEP 04

Full name

```
// 3. Full name — before
  const [firstName, setFirstName] = useState("Ana");
  const [lastName, setLastName] = useState("López");
  const [fullName, setFullName] = useState("");
  useEffect(() => {
    setFullName(firstName + " " + lastName);
  }, [firstName, lastName]);
// 3. Full name — after
  const [firstName, setFirstName] = useState("Ana");
  const [lastName, setLastName] = useState("López");
  const fullName = firstName + " " + lastName;
```

This is the example from React's guide. The JSX with the two inputs is the same in both versions. With the Effect, typing "Beatriz" left "Ana López" in the first commit.

STEP 05

Sorting

```
// 4. Sorting — before
export function SortedProductsBefore({ products, sortBy }) {
  const [sorted, setSorted] = useState([]);
  useEffect(() => {
    setSorted(products.toSorted((a, b) => a[sortBy] - b[sortBy]));
  }, [products, sortBy]);
  return <ol>{sorted.map((p) => <li key={p.id}>{p.name}</li>)}</ol>;
}

// 4. Sorting — after
export function SortedProducts({ products, sortBy }) {
  const sorted = products.toSorted((a, b) => a[sortBy] - b[sortBy]);
  return <ol>{sorted.map((p) => <li key={p.id}>{p.name}</li>)}</ol>;
}
```

> **CAUTION**
> Do not call `sort()` on arrays that come from props or state: it sorts them in place. `toSorted()` returns a new array.

STEP 06

Counts

```
// 5. Counts — before
export function OrderCountsBefore({ orders }) {
  const [pending, setPending] = useState(0);
  const [shipped, setShipped] = useState(0);
  useEffect(() => {
    setPending(orders.filter((o) => o.status === "pending").length);
    setShipped(orders.filter((o) => o.status === "shipped").length);
  }, [orders]);
  return <p>{pending} pending · {shipped} shipped</p>;
}

// 5. Counts — after
export function OrderCounts({ orders }) {
  const pending = orders.filter((o) => o.status === "pending").length;
  const shipped = orders.filter((o) => o.status === "shipped").length;
  return <p>{pending} pending · {shipped} shipped</p>;
}
```

The Effect's two setters are batched into one extra render, but they are still two copies to keep in sync.

STEP 07

Add useMemo only after measuring

Time the calculation with `console.time` in a production build. If it adds up to 1 ms or more and its dependencies rarely change, cache it with `useMemo`:

```
const sorted = useMemo(() => products.toSorted((a, b) => a.price - b.price), [products]);
```

In our test, sorting 20,000 products took 3 to 5 ms each time. Without `useMemo`, every keystroke in the search box sorted again; with `useMemo`, not once. `useMemo` does not speed up the first render.

> **CAUTION**
> In development, StrictMode calls every component (and the `useMemo` function) twice. Do not measure performance there: use production.

WHEN YOU FINISH

Checklist

- [ ] No state holds something that can be calculated from props or other state.
- [ ] No Effect is left whose only job is calling a setter.
- [ ] Sorting uses `toSorted()` or a copy.
- [ ] `useMemo` only wraps calculations measured at 1 ms or more.
- [ ] Whatever the user causes runs in the event handler.
- [ ] To reset a component when a prop changes, you pass a `key`.
- [ ] You compared the visible text before and after the change.