**🧠 What is useMemo?**

useMemo is a **React Hook** that **memoizes (remembers)** the **result of a function** — so React doesn’t re-run that function every time your component re-renders, **unless its dependencies change**.

Think of it like:  
"Only recalculate this value when it’s really needed."

**Parameters:**

* () => computeExpensiveValue(a, b) → a **function** that returns a value.
* [a, b] → the **dependency array**; only when a or b changes, the function runs again.

| **Hook** | **Returns** | **Use Case** |
| --- | --- | --- |
| useMemo | **Memoized value** | Caches results of **functions** that return values (e.g. computed arrays, numbers, objects). |
| useCallback | **Memoized function** | Caches the **function definition** itself (useful when passing functions as props). |

**🚀 When to Use useMemo**

✅ Use when:

* You have **expensive calculations** (e.g., sorting, filtering, big loops).
* You want to **avoid re-computation** when inputs haven’t changed.
* You need to **prevent unnecessary child re-renders** (if passing computed props).

❌ Don’t use when:

* Computation is simple — using useMemo itself adds a little overhead.
* You’re just trying to “optimize everything.”

**🧩 So what’s the main benefit?**

Without useMemo:

* Every render recalculates total (even when not needed).

With useMemo:

* It recalculates **only when dependencies change**.

If you had 10,000 numbers instead of 5, this saves a lot of processing.

**🔍 Scenario: Filtering a big list of users**

Imagine you have **a large list of users**, and you want to search by name.

🧱 Without useMemo

**⚙️ What happens here:**

1. When the component first renders → filter() runs once ✅
2. When you type in search → filter() runs again ✅ (expected)
3. When you toggle theme → ❌ filter() still runs again unnecessarily  
   (even though usersData and searchTerm didn’t change)

You’ll see "Filtering users..." printed **every time** in the console — that’s wasted work.

🧱 With useMemo

**🧠 Now what happens:**

* When the page loads → it filters once (you’ll see "Filtering users..." logged ✅)
* When you type → it filters again ✅ (because searchTerm changes)
* When you toggle the theme → ❌ it **does not filter again**

Even though the component re-renders (due to theme change), the filter logic is **skipped** because searchTerm didn’t change.

So in short:  
👉 useMemo is not for avoiding *renders*,  
👉 it’s for avoiding *heavy recalculations* during those renders.

⚙️ So to summarize:

| **Search Term Change** | **Does useMemo re-run?** | **Why** |
| --- | --- | --- |
| "" → "Veda" | ✅ Yes | searchTerm changed |
| "Veda" → "Reva" | ✅ Yes | searchTerm changed |
| "Reva" → "Veda" | ✅ Yes | searchTerm changed again |
| "Veda" → "Veda" | ❌ No | searchTerm didn’t change |

**🧠 Key takeaway:**

* useMemo **only caches the last result** based on its dependency list.
* It doesn’t store a “history” of old computations.
* So when you type "Veda" again after "Reva", React still re-filters — because the dependency changed

**💡 What useMemo actually does (in simple words):**

useMemo **remembers** the result of a calculation (like filtering, sorting, or computing) so React doesn’t have to **recalculate** it on every render.

It’s for **performance optimization**, not skipping renders —  
it skips **re-computation** of expensive logic.

**🧠 Example meaning:**

Let’s say you have a large data list of 10,000 users.

If you use a **filter**, **sort**, or **map** every render — React will run that heavy logic every time the component re-renders (like toggling a theme or typing in another field).

That’s wasteful ⚠️

So you wrap it in useMemo:

**🛍️ Scenario: E-Commerce Product List with Filters**

You have a list of 5,000 products.

You can:

* Search products by name 🕵️‍♀️
* Change app theme (light / dark) 🌗

Now, changing theme should **not** re-filter products —  
but React will still re-render the whole screen (because theme changed).

That’s why we’ll use **useMemo**.

**⚙️ Behavior:**

* When you type → it filters (✅ expected)
* When you toggle theme → it filters again (❌ unnecessary)

Every time you change theme, "Filtering products..." runs again in console  
because React re-renders the component, so .filter() runs again.

Imagine 5,000 products — that’s a big performance hit ⚠️

⚡ Step 2 — Optimized with useMemo

| **Action** | **Re-render?** | **Filter recalculates?** | **Why** |
| --- | --- | --- | --- |
| Page loads | ✅ Yes | ✅ Yes | First time |
| Type “iPhone” | ✅ Yes | ✅ Yes | search changed |
| Type “TV” | ✅ Yes | ✅ Yes | search changed |
| Toggle theme | ✅ Yes | ❌ No | search didn’t change |
| Toggle theme again | ✅ Yes | ❌ No | search didn’t change |

So, useMemo **doesn’t stop re-renders** (because theme still changes UI),  
but it **prevents re-running expensive filter logic** if dependencies didn’t change.

**🧩 Situation Recap**

Let’s say your e-commerce app:

* Fetches products from an **API** once.
* Lets users **filter/search** products.
* Lets users toggle **theme** (light/dark).

Now, let’s see how all this behaves with and without useMemo.

**⚙️ Step 1 — Without useMemo**

const filteredProducts = products.filter((product) =>

product.name.toLowerCase().includes(searchTerm.toLowerCase())

);

Here’s what happens:  
1️⃣ Page loads → API runs ✅  
2️⃣ You type in search → filters again ✅  
3️⃣ You change theme → component **re-renders**, so .filter() runs again ❌ (unnecessary extra filtering logic)  
4️⃣ If API call is *inside useEffect without []*, it’ll call API again ❌ (big problem)

So — every re-render runs the .filter() code (and possibly API),  
even when nothing related to products changed.

**⚡ Step 2 — Using useMemo**

const filteredProducts = useMemo(() => {

console.log("Filtering products...");

return products.filter((product) =>

product.name.toLowerCase().includes(searchTerm.toLowerCase())

);

}, [searchTerm, products]);

Now:

✅ Page loads → filters once  
✅ You type “iPhone” → filters again  
❌ You toggle theme → filters **not run again**, because neither searchTerm nor products changed.

So yes — the **component re-renders**,  
but the **filter logic (or API call)** is **skipped** because the dependencies didn’t change.

**🧠 If it was an API call**

Let’s say you do this:

useEffect(() => {

fetch("/api/products").then(...)

}, []); // ✅ Runs only once on mount

Even if you change theme, sidebar, etc. →  
React re-renders the component but **does not** call the API again.

👉 Why?  
Because useEffect with [] means “only run once at first mount.”

**🧩 So combined behavior looks like:**

| **Action** | **Component Re-renders?** | **API Runs Again?** | **Filter Logic Runs Again?** |
| --- | --- | --- | --- |
| Page loads | ✅ Yes | ✅ Yes | ✅ Yes |
| Search changes | ✅ Yes | ❌ No | ✅ Yes |
| Theme toggles | ✅ Yes | ❌ No | ❌ No |
| Resize window | ✅ Yes | ❌ No | ❌ No |

**✅ Final Summary**

* useEffect([]) → prevents API from running again unnecessarily.
* useMemo([searchTerm]) → prevents heavy logic like filtering/sorting from running again unnecessarily.
* React **still re-renders the UI** (e.g., to show dark mode),  
  but the *data-fetching* and *expensive logic* **stay cached and untouched**.