**useMemo**

useMemo is a React Hook that lets you cache the result of a calculation between re-renders.

**Parameters**

* calculateValue: The function calculating the value that you want to cache. It should be pure, should take no arguments, and should return a value of any type. React will call your function during the initial render. On next renders, React will return the same value again if the dependencies have not changed since the last render. Otherwise, it will call calculateValue, return its result, and store it so it can be reused later.
* dependencies: The list of all reactive values referenced inside of the calculateValue code. Reactive values include props, state, and all the variables and functions declared directly inside your component body. If your linter is [configured for React](https://react.dev/learn/editor-setup#linting), it will verify that every reactive value is correctly specified as a dependency. The list of dependencies must have a constant number of items and be written inline like [dep1, dep2, dep3]. React will compare each dependency with its previous value using the [Object.is](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/is) comparison.
* On the initial render, useMemo returns the result of calling calculateValue with no arguments.
* During next renders, it will either return an already stored value from the last render (if the dependencies haven’t changed), or call calculateValue again, and return the result that calculateValue has returned.

const cachedValue = useMemo(calculateValue, dependencies)

* [Usage](https://react.dev/reference/react/useMemo#usage)
  1. [Skipping expensive recalculations](https://react.dev/reference/react/useMemo#skipping-expensive-recalculations)

useMemo in the first rendering run the caculation function and cach (store) the result and return it ,then in any rendering of the component react check if the useMemo dependency changed it will re-run the caculation function and rechache it else if the dependency not changed react return the chached value wich stores in the prev rendering .

ex: By default, React will re-run the entire body of your component every time that it re-renders. For example, if this TodoList updates its state or receives new props from its parent, the filterTodos function will re-run:

function TodoList({ todos, tab, theme }) {

const visibleTodos = filterTodos(todos, tab);

// ...

}

Usually, this isn’t a problem because most calculations are very fast. However, if you’re filtering or transforming a large array, or doing some expensive computation, you might want to skip doing it again if data hasn’t changed. If both todos and tab are the same as they were during the last render, wrapping the calculation in useMemo like earlier lets you reuse visibleTodos you’ve already calculated before.

* + [Skipping re-rendering of components](https://react.dev/reference/react/useMemo#skipping-re-rendering-of-components)

Component take aprops as theme props or any props if one of this props changed this component eill rerendering so all the children component under it will rerender whether it is props changed or no ,

Let this children component props not changed why make rerender of it ?

So we can use useMemo to prevent rerender this child component eleseif it is props changed

**👩‍💻 عندي كومبوننتين:**

1. **TodoList** (الأب)
2. **List** (الإبن اللي بياخد todos ويعرضهم)

**🟢 الحالة بدون useMemo وبدون memo**

function TodoList({ todos, theme }) {

const visibleTodos = todos.filter(t => !t.done); // بيرجع Array جديد كل مرة

return (

<div className={theme}>

<List items={visibleTodos} />

</div>

);

}

function List({ items }) {

console.log("List rerender"); // للتوضيح

return <ul>{items.map(t => <li key={t.id}>{t.text}</li>)}</ul>;

}

🔎 لما تغيّري theme (مثلاً dark → light):

* TodoList rerender ✅ (طبيعي).
* visibleTodos بيعمل Array جديد في كل render.
* React يفتكر إن props لـ List اتغيرت (لأن Array جديد) → فيرندر List كمان.

**🟡 نضيف memo فقط**

const List = React.memo(function List({ items }) {

console.log("List rerender");

return <ul>{items.map(t => <li key={t.id}>{t.text}</li>)}</ul>;

});

* هسا React بفحص props:
  + إذا نفس القيمة = ما يرندر.
  + إذا اختلفت = يرندر.
* لكن! Array اللي جاي من filter كل مرة جديد → props مش نفسها → لسا List يرندر كل مرة.

**🔴 نضيف useMemo**

function TodoList({ todos, theme }) {

const visibleTodos = React.useMemo(() => {

return todos.filter(t => !t.done);

}, [todos]); // فقط لو todos تغيرت

return (

<div className={theme}>

<List items={visibleTodos} />

</div>

);

}

* الآن لو غيرتي theme بس:
  + visibleTodos يضل نفس الـ reference (ما ينشأ Array جديد).
  + props اللي رايحة لـ List نفسها.
  + memo يمنع rerender.

🔹 النتيجة:  
List ما يرندر إلا إذا فعلاً تغيرت todos.

**✨ الخلاصة**

* memo = يقول لReact: "لا ترندرني إلا لو props اتغيروا".
* useMemo = يخلي القيمة (array, object, function) نفسها بين renders عشان props يبينوا "ما تغيروش".
* الاتنين مع بعض = تحسين أداء.

2.[Preventing an Effect from firing too often](https://react.dev/reference/react/useMemo#preventing-an-effect-from-firing-too-often)

Sometimes, you might want to use a value inside an [Effect:](https://react.dev/learn/synchronizing-with-effects)

Ex:

function ChatRoom({ roomId }) {

const [message, setMessage] = useState('');

const options = {

serverUrl: 'https://localhost:1234',

roomId: roomId

}

useEffect(() => {

const connection = createConnection(options);

connection.connect();

Note :

**👇 المشكلة**

* كل ما الكومبوننت يعمل re-render، بينبني **object جديد** (مرجع جديد).
* حتى لو القيم نفسها، React بشوف إنه اختلف (لأنه reference مختلف).
* النتيجة: الـ useEffect يشتغل كأنه في dependency تغير.

**✅ الحل**

* نحط إنشاء الـ object داخل **useMemo**، بحيث ينشأ مرة وحدة فقط (أو يتغير بس إذا الـ dependencies تغيرت).
* هيك React بيعطيك **نفس الـ reference** بين الـ renders.
* النتيجة: الـ useEffect ما يتنفذ إلا عند التغيير الفعلي للقيم.

يعني

إذا أنشأت كائن (object) داخل الكومبوننت واستخدمته في useEffect، React رح يبني نسخة جديدة من الكائن في كل إعادة render—even لو القيم ما تغيرت. هذا يخلي الـ useEffect يشتغل بشكل غير ضروري. عشان نتجنب هالشي، من الأفضل نستخدم useMemo لتوليد الكائن مرة وحدة وإعادة استخدامه ما دام الـ dependencies ما تغيرت. بهيك، React يحتفظ بنفس المرجع (reference)، وما يشغّل الـ useEffect إلا لما يحصل تغيير حقيقي.

3 .[Memoizing a dependency of another Hook](https://react.dev/reference/react/useMemo" \l "memoizing-a-dependency-of-another-hook)

تقربا نفس فكرة اللي قبل انو مثلا

الـ **useEffect** بيعتمد على **dependency** معيّن. لكن هذا الـ dependency في كل re-render بينبنى من جديد بمرجع (reference) مختلف، حتى لو القيم داخله نفسها. React مش ذكية كفاية لتفهم إنه القيمة نفسها، لأنها بتقارن الـ reference فقط. النتيجة: React بتفكّر إنه الـ dependency تغيّر، وبالتالي بتعيد تنفيذ الـ useEffect كل مرة، مع إنه فعليًا ما صار أي تغيير حقيقي.

4 .[Memoizing a function](https://react.dev/reference/react/useMemo" \l "memoizing-a-function)