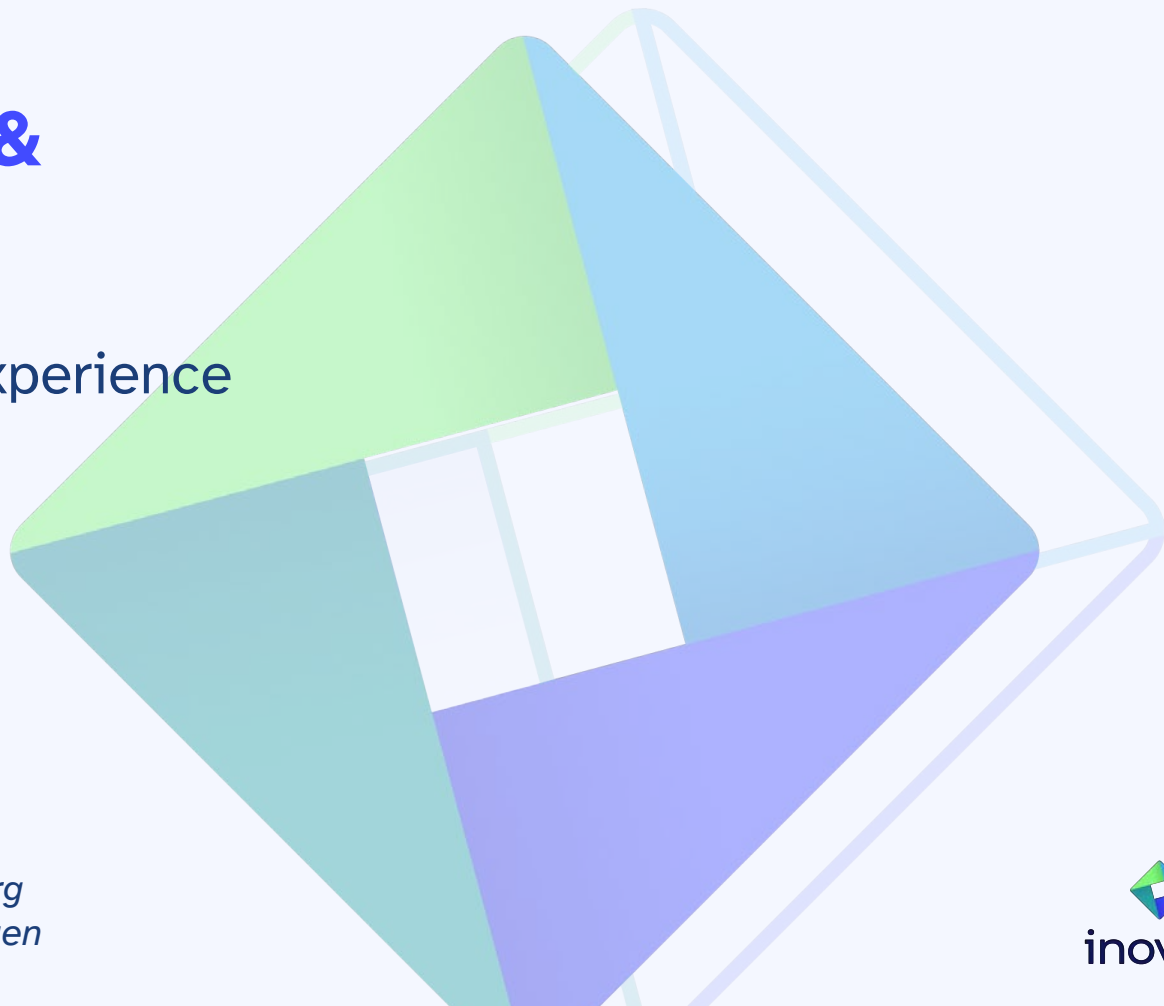


# React Compiler & React 19

Enhanced Developer Experience  
and Performance

## Team inovex

*Karlsruhe · Köln · München · Hamburg  
Berlin · Stuttgart · Pforzheim · Erlangen*



# Agenda

- What is React Compiler and how it helps us
- Unnecessary re-renders and memoization
- React Compiler: behind the scenes
- Rolling-out React Compiler on [scrumlr.io](https://scrumlr.io)
- Key takeaways on React Compiler
- React 19 : new APIs, hooks and improvements
- Summary



## What is a compiler?

"A compiler is a special program that translates a programming language's source code into machine code, bytecode or another programming language..."

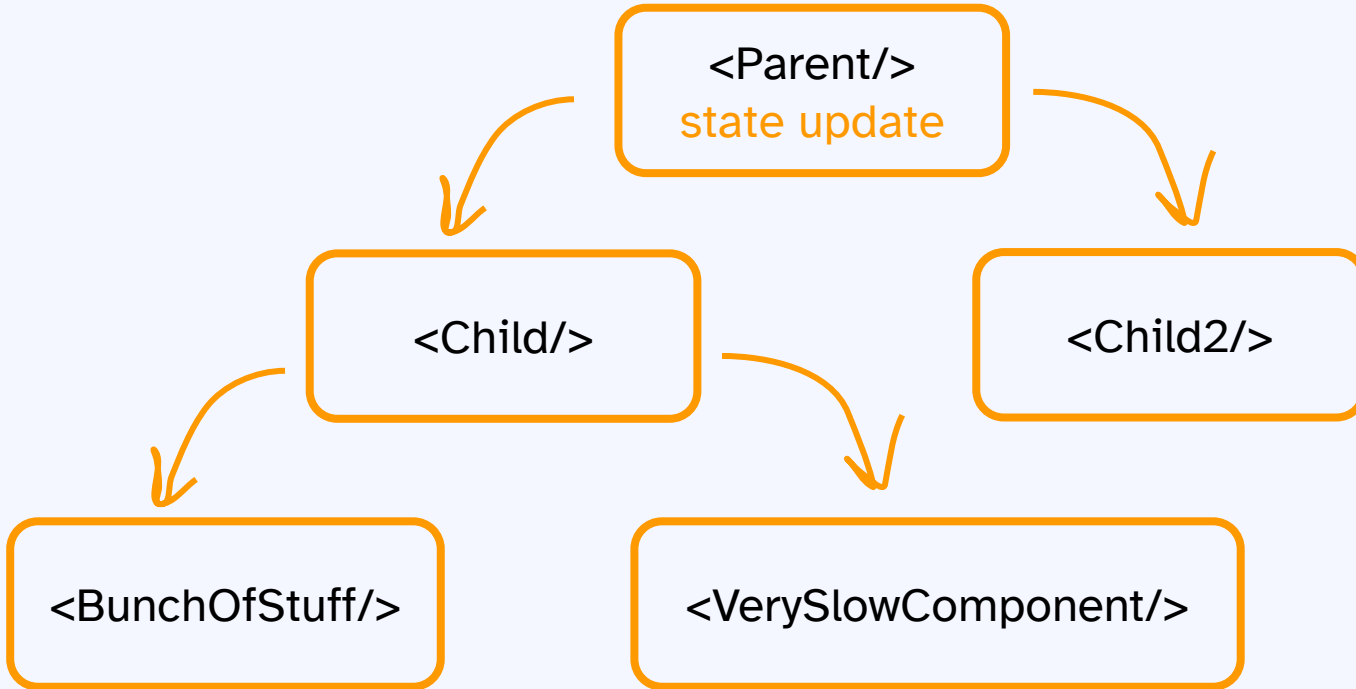
# React Compiler

"...is a build-time only tool that automatically optimizes your React app. It works with plain JavaScript, and understands the Rules of React, so you don't need to rewrite any code to use it...

In order to optimize applications, React Compiler **automatically memoizes your code...**"



# Re-renders in React are cascading



# How to avoid unnecessary re-renders

- Composition
  - "moving state down"
  - "components as props"
  - "elements as props"

- `React.memo()`
- `useCallback`
- `useMemo`

memoization



```
const Component = () => {
  const [inputValue, setInputValue] = useState('');

  const memoizedValue = useMemo(() => {
    // Some heavy computation or data processing
    return inputValue.toUpperCase();
  }, [inputValue]);

  const handleInputChange = useCallback((e) => {
    setInputValue(e.target.value);
  }, []);

  const handleClick = useCallback(() => {
    // Some action to be performed
  }, []);

  const handleFormSubmit = useCallback((e) => {
    e.preventDefault();
    // Some form submission logic
  }, []);

  return (
    <div>
      <input type="text" value={inputValue} onChange={handleInputChange} />
      <button onClick={handleClick}>Click me</button>
      <form onSubmit={handleFormSubmit}>
        <input type="submit" value="Submit" />
      </form>
      <p>{memoizedValue}</p>
    </div>
  );
};

export default Component;
```

## Manual memoization in React

- harder than it seems
- not straightforward
- not intuitive
- readability is lost
- easy to break





## Compiler to the rescue! as long as...

- you follow the rules of React
- code is valid, semantic JavaScript
- nullable/optional values and properties are defined before accessing them, a.k.a. { "strictNullChecks" : "true" }

**React Compiler will skip compilation when it detects an error**

[source code](#)





```
1 export default function MyApp() {  
2   const name = "Sirius Black"  
3   return (  
4     <div>  
5       <p>{name}</p>  
6     </div>  
7   )  
8 }  
9  
10  
11
```

-JS

```
function MyApp() {  
  const $ = _c(1);  
  
  let t0;  
  
  if ($[0] === Symbol.for("react.memo_cache_sentinel")) {  
    t0 = (  
      <div>  
        <p>{"Sirius Black"}</p>  
      </div>  
    );  
    $[0] = t0;  
  } else {  
    t0 = $[0];  
  }  
  
  return t0;  
}
```



```
const $empty = Symbol.for("react.memo_cache_sentinel");
```

```
/**
```

```
 * DANGER: this hook is NEVER meant to be called directly!
```

```
 **/
```

```
export function c(size: number) {
```

```
  return React.useState(() => {
```

```
    const $ = new Array(size);
```

```
    for (let ii = 0; ii < size; ii++) {
```

```
      $[ii] = $empty;
```

```
    }
```

```
    // This symbol is added to tell the react devtools that this array is from
```

```
    // useMemoCache.
```

```
    // @ts-ignore
```

```
    [$empty] = true;
```

```
    return $;
```

```
  })[0];
```

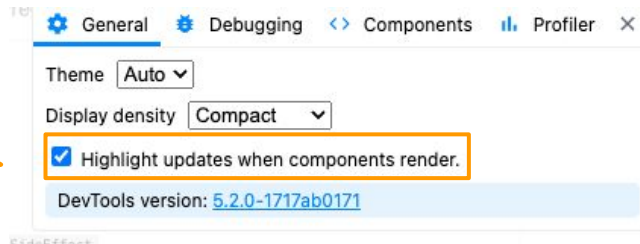
```
}
```

# React Compiler on scrumlr.io

- initial healthcheck with

```
npx react-compiler-healthcheck@latest
```

- update to React 19
- install eslint-plugin-react-compiler and configure it
- install Compiler
- analyze with React DevTools



## React Compiler on [scrumlr.io](https://scrumlr.io)

- ⚡ `npx react-compiler-healthcheck@latest`  
Successfully compiled 124 out of 135 components.  
StrictMode usage found.  
Found no usage of incompatible libraries.

**Health check:** approx.92% can be compiled

**Eslint check:** 25 skipped components

# React Compiler - Takeaways and Concerns

- Experiment and play around now, adopt later
- Follow The Rules of React
- Pay attention to libraries/packages
- Affects Memory, CPU ?
- Hard to debug ?



**2.5x**

faster interactions

**12%**

faster initial load  
faster navigations

**0%**

memory increase

# What is new in React 19

## Client API: use()

Lets you read the value of a Promise or a Context

```
const resolvedData = use(Promise)  
  
const context = use(Context)
```



## Promise

```
const fetchData = async() => {
  const response = await fetch("https://.....");
  return response.json()
}

const Component = () => {
  const data = use(fetchData);

  return (
    <ErrorBoundary fallback={<ErrorPage/>}>
      <Suspense fallback={<Loading/>}>
        {data}
      </Suspense>
    </ErrorBoundary>
  )
}
```

## Context

```
const Tooltip = ({ show }) => {
  if (show) {
    const theme = use(ThemeContext);
    return <ToolTip theme={theme} />;
  }

  return null;
};
```

# Client API: Actions

- **Perform a data mutation and then update state in response**
- **Automatically manage submitting data:**
  - Pending state
  - Optimistic updates
  - Error Handling
  - Forms

## React DOM: `<form>` Actions via **action** and **formAction** props

```
<form action={actionFunction}>
```

```
<button formAction={actionFunction} >
```

```
<input formAction={actionFunction} >
```

## React DOM: <form> Actions

```
<form action={formData => {  
  const email = formData.get("email");  
}}>
```

```
<form action={async formData => {  
  const email = formData.get("email");  
  await updateEmail(email);  
}}>
```

## Client API: useActionState()

update state based on the result of an action

```
const [state, formAction, isPending] = useActionState(actionFunction, initialState);
```

# Client API: useActionState()

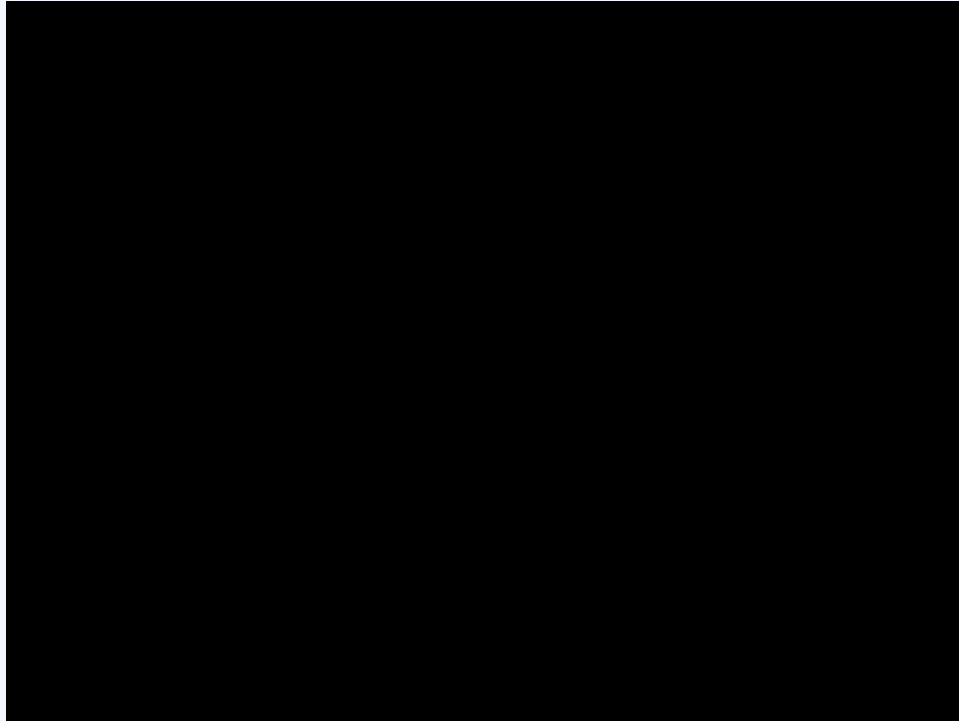
```
const Form = () => {
  const [ data, submitAction, isPending ] = useActionState(

    async (currentValue, formData) => {
      const newName = formData.get("name");
      const data = await updateName(newName);
      return data;
    },

    null,
  );


  return (
    <form action={submitAction}>
      <input name="name" ... />
      <button>
        {isPending ? "Sending" : "Sent"}
      </button>
      {data?.success && "Saved to database!"}
    </form>
  )
};
```

## Using action for form submission





## Traditional form data submission



Enter your name

Send

Name is ...

The image shows a rectangular form with a thick black border. Inside the form, there is a text input field on the left containing the placeholder text "Enter your name". To the right of the input field is a light gray button with the text "Send". Below the input field and button, the text "Name is ..." is displayed.

## Client API: useFormStatus()

get status information from the last form submission

```
const { data, pending, method, action } = useFormStatus()
```

```
const Form = () => {  
  
  const actionFunction = () => {  
    //some stuff to be done  
  };  
  
  return (  
    <form action={actionFunction}>  
      <input name="email" ... />  
      <CustomButton />  
    </form>  
  )  
};  
  
const CustomButton = () => {  
  
  const { pending } = useFormStatus();  
  
  return (  
    <button>  
      {pending ? "Sending" : "Sent"}  
    </button>  
  )  
};
```



```
const Form = () => {  
  
  const actionFunction = () => {  
    //some stuff to be done  
  };  
  
  const { pending } = useFormStatus();  
  
  return (  
    <form action={actionFunction}>  
      <input name="email" ... />  
      <button>  
        {pending ? "Sending" : "Sent"}  
      </button>  
    </form>  
  )  
};
```



## Client API: useOptimistic()

optimistically update UI while async request is underway

```
const [ optimisticValue, setOptimisticValue ] = useOptimistic(state, updateFunction)
```

# Client API: useOptimistic()

```
function ChangeName({currentName, onUpdateName}) {
  const [optimisticName, setOptimisticName] = useOptimistic(currentName);

  const submitAction = async formData => {
    const newName = formData.get("name");
    setOptimisticName(newName);
    const updatedName = await updateName(newName);
    onUpdateName(updatedName);
  };

  return (
    <form action={submitAction}>
      <p>Your name is: {optimisticName}</p>
      <p>
        <label>Change Name:</label>
        <input
          type="text"
          name="name"
        />
      </p>
    </form>
  );
}
```

Enter your name

Send

Name is ...

## Notes on Actions

- actions trigger transitions → concurrent rendering
- transitions schedules **TWO RENDERS**
  - one high priority render with old state
  - one low priority
- Transitions are not a tool for everyday use
- Specific use → transition from "nothing" to "very heavy stuff"



- **Improvements: ref as prop**

access ref as prop for function components

```
//This

function Input({ref}) {
  return <input ref={ref} ... />
};

//insted of

import {forwardRef} from 'react';

const Input = forwardRef((props, ref) =>{
  return <input ref={ref} ... />
});
```

- **Improvements: Cleanup functions for refs**

```
<input
  ref={({ref}) => {
    return () => {
      // ref cleanup
      ref.removeEventHandler(...)
    };
  }}
/>
```

## ● Improvements: Support for Document Metadata

use document metadata tags within components

```
function BlogPost({post}) {
  return (
    <article>
      <h1>{post.title}</h1>
      <title>{post.title}</title>
      <meta name="author" content={post.content} />
      <link rel="author" href="https://twitter.com/.../" />
      <meta name="keywords" content={post.keywords} />
      <p>
        .....
      </p>
    </article>
  );
}
```

- **Improvements: stylesheet support**

render stylesheets within components

```
function Component() {  
  return (  
    <Suspense fallback="loading...">  
      <link rel="stylesheet" href="foo" precedence="default" />  
      <link rel="stylesheet" href="bar" precedence="high" />  
      <article class="foo-class bar-class">  
        {...}  
      </article>  
    </Suspense>  
  )  
}
```

## ● Improvements: Preloading resources

new APIs for telling browser how to load resources

```
import { prefetchDNS, preconnect, preload, preinit } from 'react-dom'
function MyComponent() {
  preinit('https://.../path/to/some/script.js', {as: 'script' }) // loads and executes this script eagerly
  preload('https://.../path/to/font.woff', { as: 'font' }) // preloads this font
  preload('https://.../path/to/styleSheet.css', { as: 'style' }) // preloads this stylesheet
  prefetchDNS('https://...') // when you may not actually request anything from this host
  preconnect('https://...') // when you will request something but aren't sure what
}
```

## Additionally:

- React Server Components
- server actions with "use action"
- support for async `<script>` tags within components
- better error reporting
- support for custom elements
- `<Context>` as Provider

## Summary

- React Compiler is not React 19
- Forget about memoization soon? Not really...
  - Half of re-renders have negligible effect on performance
  - Composition
  - External state management
- Caution when migrating to React 19





**React 19 & React Compiler: Elevating Developer Experience Without Compromising Performance**