SOLIDJS

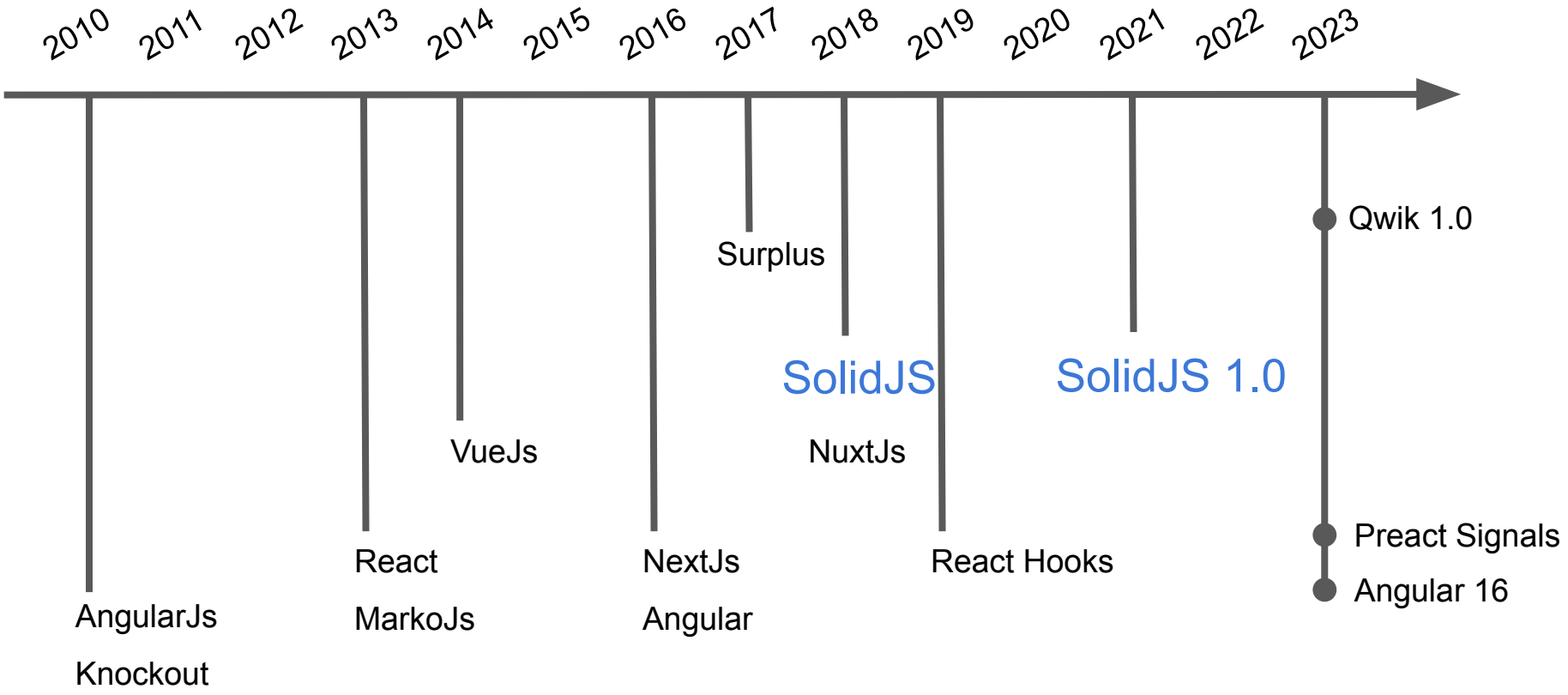
Simple and performant reactivity for building user interfaces.

Nawfel Bengherbia

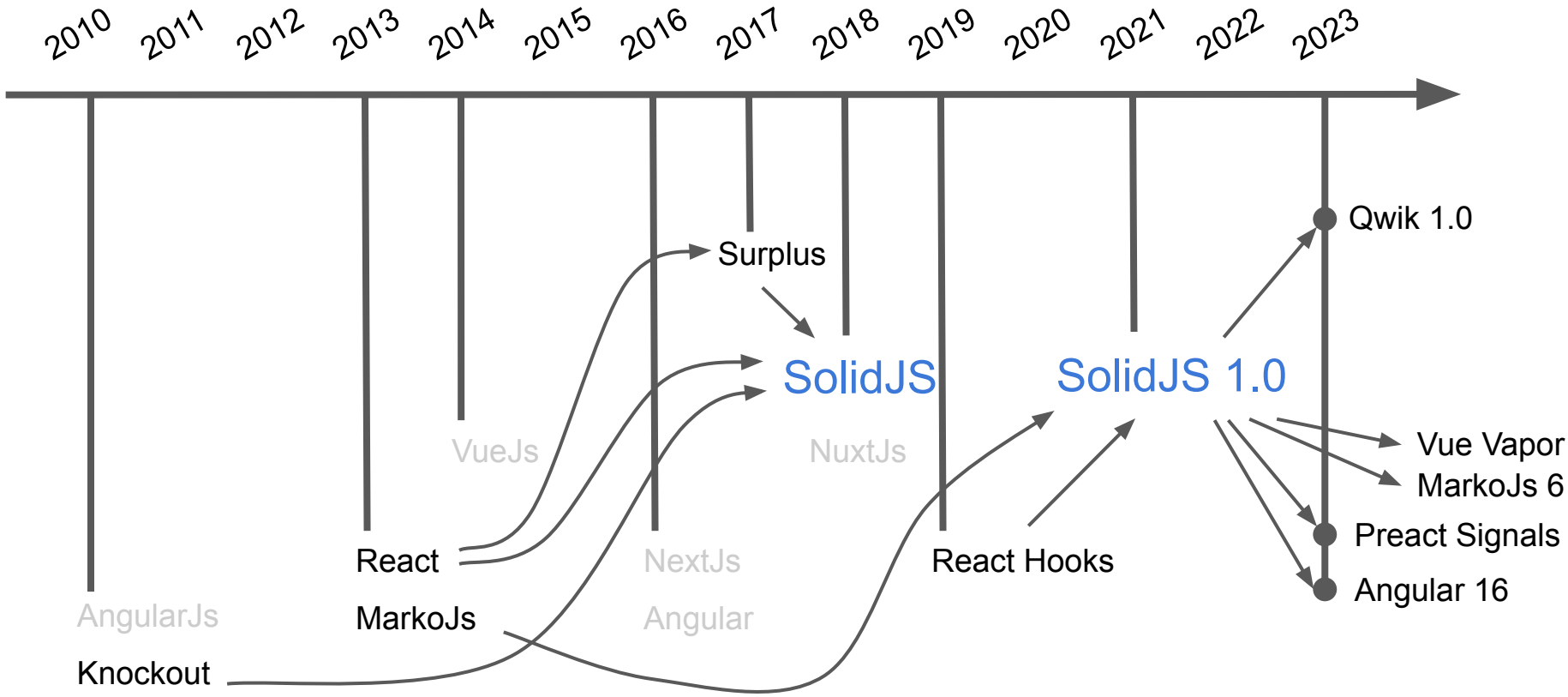
Agile Web Developer

[sfeir]

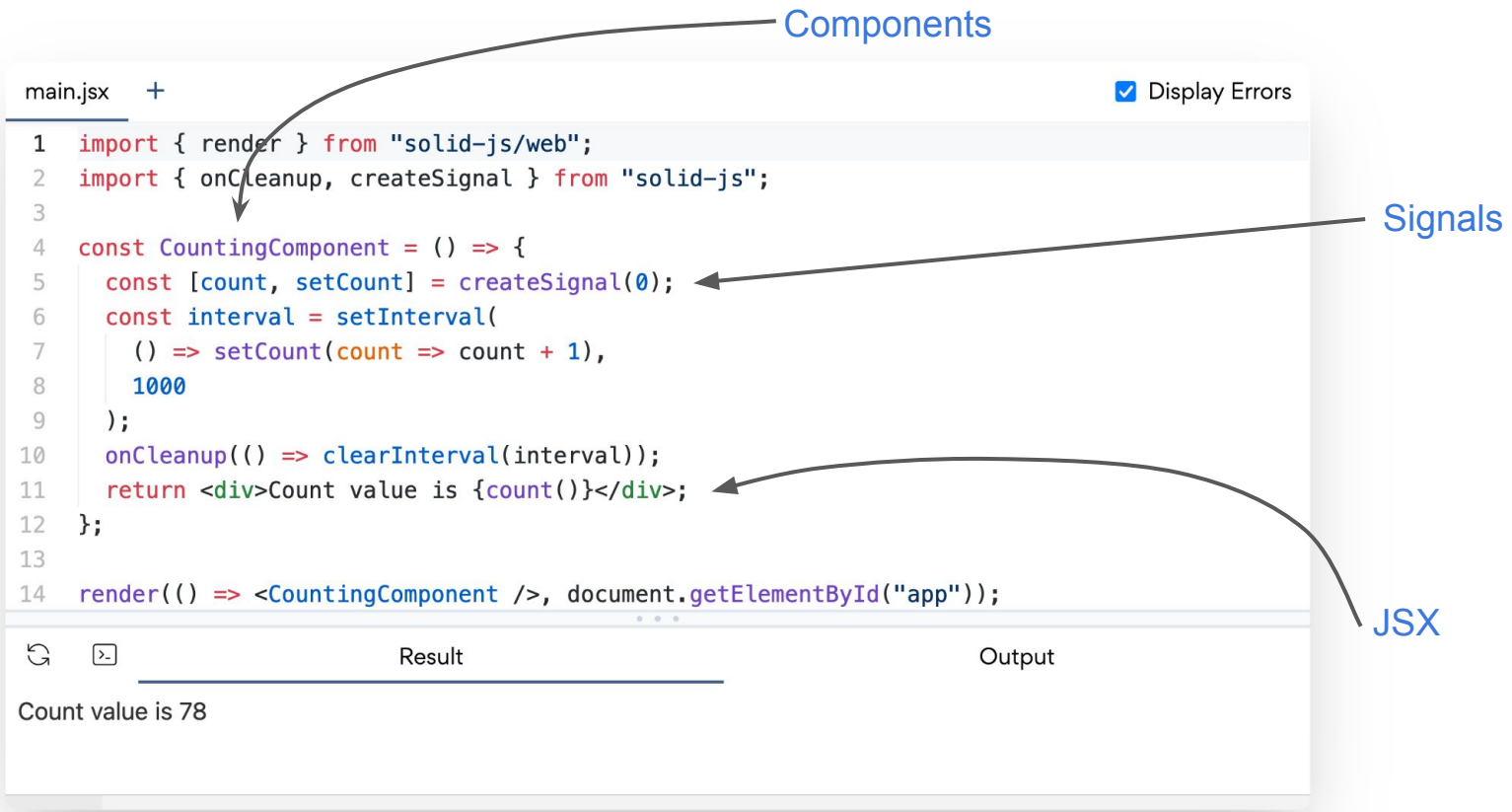
Timeline


2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
AngularJs
Knockout
React
MarkoJs
VueJs
NextJs
Angular
Surplus
SolidJS
NuxtJs
React Hooks
SolidJS 1.0
Qwik 1.0
Preact Signals
Angular 16

Influences de SolidJs



Ça ressemble à quoi ?



Avantages de SolidJS

- JSX
- Patterns simples et explicites

- Signals
  - Gestion simple du state
  - Rapidité
- 7,9KB gzipped

- SSR optimisé

https://medium.com/@dan_abramov/youre-missing-the-point-of-react-a20e34a51e1a

Avantages de SolidJS

- JSX
- Patterns simples et explicites

- Signals
  - Gestion simple du state
  - Rapidité
- 7.9KB gzipped

> Now that we're not surprised by virtual DOM anymore and it is being adopted by other frameworks and libraries, we can focus on examining React's true strengths: **composition, unidirectional data flow, freedom from DSLs, explicit mutation and static mental model.**

Avantages de SolidJS

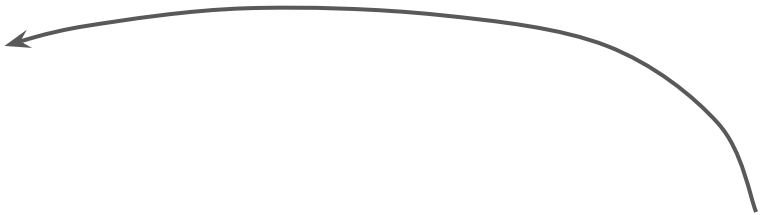
- JSX
- Patterns simples et explicites

- Signals
  - Gestion simple du state
  - Rapidité
- 7,9KB gzipped

- SSR optimisé

Gestion de state (Exemple 1)

```jsx
import { useState } from 'react';

export default function Counter() {
  const [count, setCount] = useState(0);

  function handleClick() {
    setCount(count + 1);
  }

  return (
    <button onClick={handleClick}>
      You pressed me {count} times
    </button>
  );
}
```

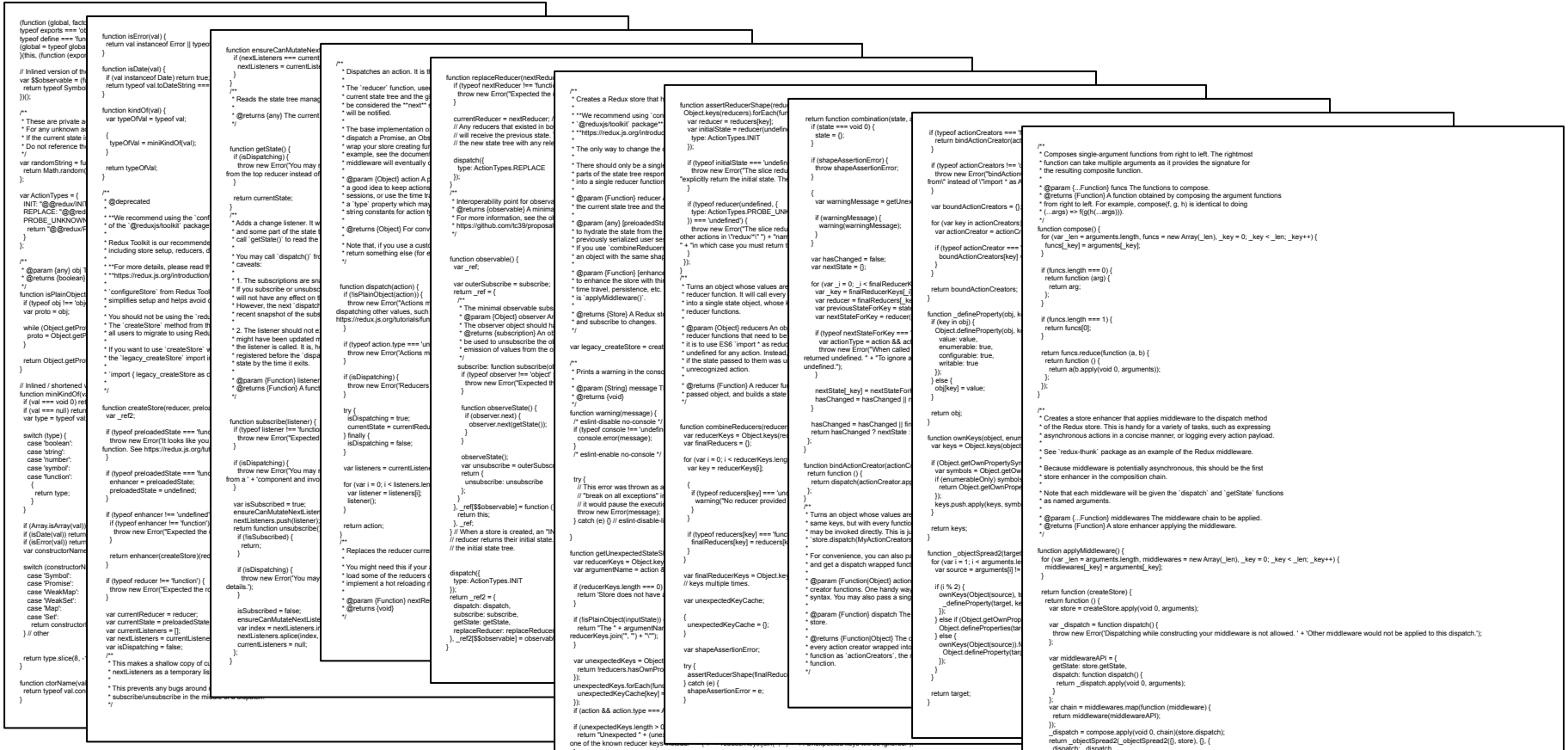


Comment rendre ce state global ?

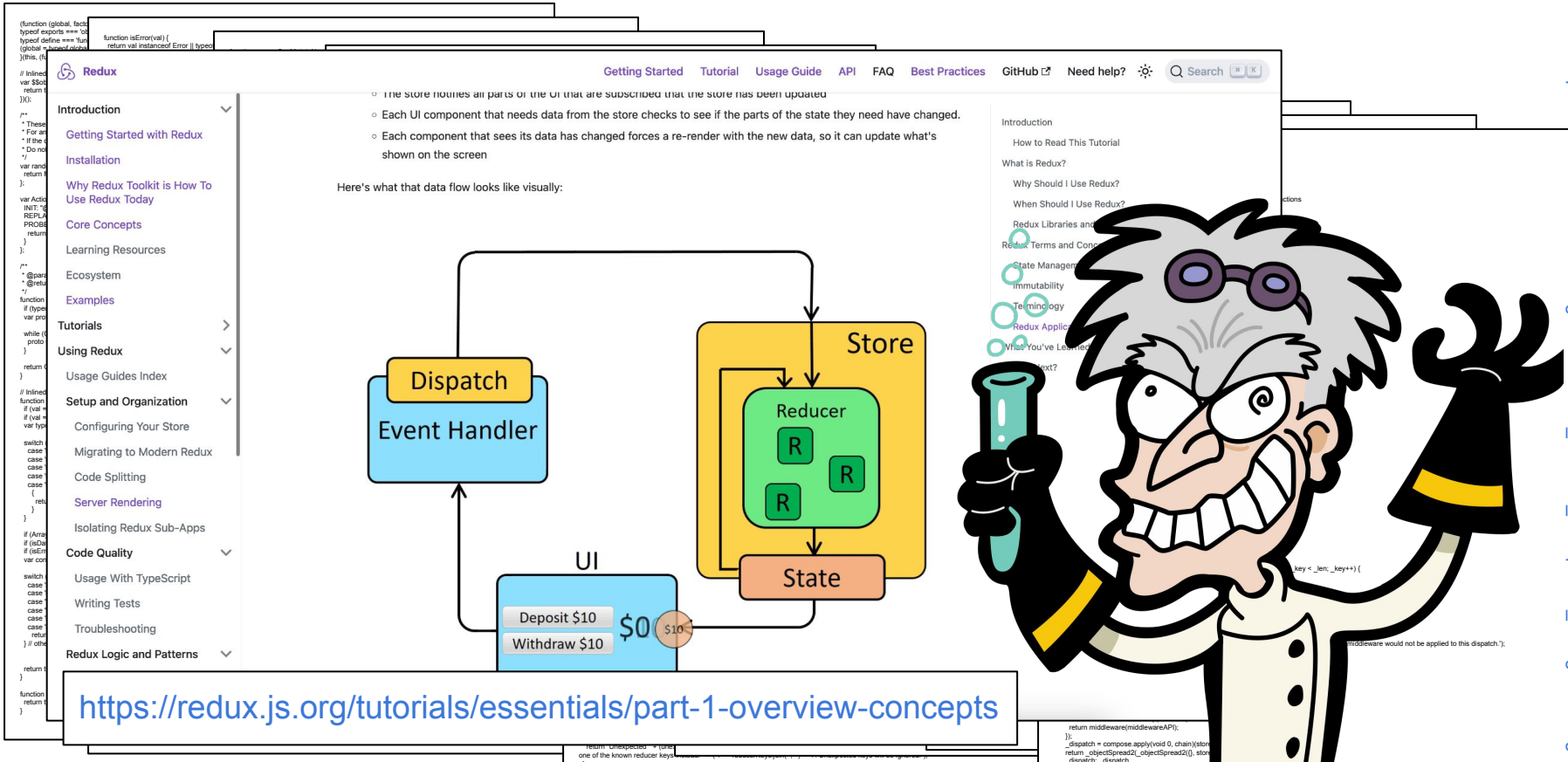
https://react.dev/reference/react/useState

Gestion de state (Exemple 1)

https://www.unpkg.com/redux@4.2.1/dist/redux.js



Gestion de state (Exemple 1)



Gestion de state

https://redux.js.org/introduction/examples

https://codesandbox.io/s/competent-frog-4kklt3?file=/src/features/counter/Counter.js

Counter.js

```jsx
import React from "react";
import { useSelector, useDispatch } from "react-redux";
import { increment, selectCount } from "./counterSlice";

export function Counter() {
  const count = useSelector(selectCount);
  const dispatch = useDispatch();

  return (
    <button onClick={() => dispatch(increment())}>
      You pressed me {count} times
    </button>
  );
}
```

counterSlice.js

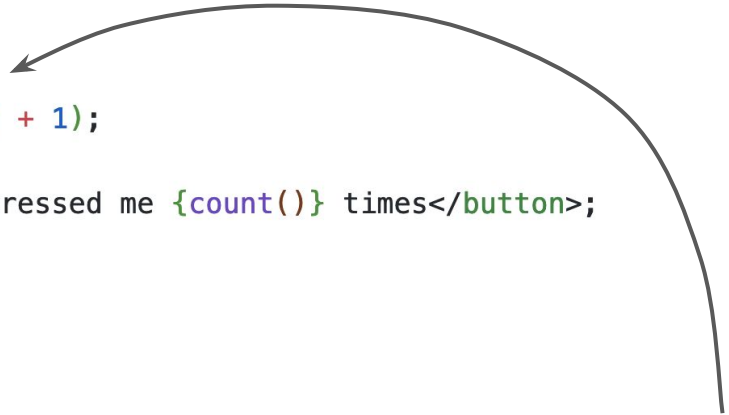
```jsx
import { createSlice } from "@reduxjs/toolkit";

const initialState = {
  value: 0,
  status: "idle"
};

export const counterSlice = createSlice({
  name: "counter",
  initialState,
  reducers: {
    increment: (state) => {
      state.value += 1;
    }
  }
});

export const { increment } = counterSlice.actions;

export const selectCount = (state) => state.counter.value;

export default counterSlice.reducer;
```

Code Splitting
Server Rendering
Isolating Redux Sub-Apps
Code Quality
Usage With TypeScript
Writing Tests
Troubleshooting
Redux Logic and Patterns
Structuring Reducers
Reducing Boilerplate

UI
Deposit $10
Withdraw $10
$0
$10

Gestion de state (Exemple 1)

```
import { createSignal } from "solid-js";

function Counter() {
  const [count, setCount] = createSignal(1);
  const handleClick = () => setCount(count() + 1);

  return <button onClick={handleClick}>You pressed me {count()} times</button>;
}
```



Comment rendre ce state global ?

Gestion de state (Exemple 1)

```
import { createSignal } from "solid-js";

function Counter() {
  const [count, setCount] = createSignal(1);
  const handleClick = () => setCount(count() + 1);

  return <button onClick={handleClick}>You pressed me {count()} times</button>;
}
```

```
import { createSignal } from "solid-js";

const [count, setCount] = createSignal(1);
function Counter() {
  const handleClick = () => setCount(count() + 1);

  return <button onClick={handleClick}>You pressed me {count()} times</button>;
}
```

Gestion de state (Exemple 2)

```
function Counter() {
  let [count, setCount] = useState(0);

  useEffect(() => {
    let id = setInterval(() => {
      setCount(count + 1);
    }, 1000);
    return () => clearInterval(id);
  });

  return <h1>{count}</h1>;
}
```

https://overreacted.io/making-setinterval-declarative-with-react-hooks/

Gestion de state (Exemple 2)

Overreacted

Making setInterval Declarative with React Hooks

February 4, 2019 · 16 min read

Translated by readers into: Français · 简体中文

If you played with React Hooks for more than a few hours, you probably ran into an intriguing problem: using `setInterval` just doesn't work as you'd expect.

In the words of Ryan Florence:

> *I've had a lot of people point to setInterval with hooks as some sort of egg on React's face*

Honestly, I think these people have a point. It *is* confusing at first.

But I've also come to see it not as a flaw of Hooks but as a mismatch between the React programming model and `setInterval`. Hooks, being closer to the React programming model than classes, make that mismatch more prominent.

There *is* a way to get them working together very well but it's a bit unintuitive.

In this post, we'll look at *how* to make intervals and Hooks play well together, *why* this solution makes sense, and which *new* capabilities it can give you.

Disclaimer: this post focuses on a *pathological case*. Even if an API simplifies a hundred use cases, the discussion will always focus on the one that got harder.

If you're new to Hooks and don't understand what the fuss is about, check out this introduction and the documentation instead. This post assumes that you worked with Hooks for more than an hour.

Just Show Me the Code

Without further ado, here's a counter that increments every second:

```
import React, { useState, useEffect, useRef } from 'react';

function Counter() {
  let [count, setCount] = useState(0);

  useInterval(() => {
    // Your custom logic here
    setCount(count + 1);
  }, 1000);

  return <h1>{count}</h1>;
}
```

(Here's a CodeSandbox demo.)

This `useInterval` isn't a built-in React Hook; it's a custom Hook that I wrote:

```
import React, { useState, useEffect, useRef } from 'react';

function useInterval(callback, delay) {
  const savedCallback = useRef();

  // Remember the latest callback.
  useEffect(() => {
    savedCallback.current = callback;
  }, [callback]);

  // Set up the interval.
  useEffect(() => {
    function tick() {
      savedCallback.current();
    }
    if (delay !== null) {
      let id = setInterval(tick, delay);
      return () => clearInterval(id);
    }
  }, [delay]);
}
```

(Here's a CodeSandbox demo in case you missed it earlier.)

My `useInterval` Hook sets up an interval and clears it after unmounting. It's a combo of `setInterval` and `clearInterval` tied to the component lifecycle.

Feel free to copy paste it in your project or put it on npm.

If you don't care how this works, you can stop reading now! The rest of the blog post is for folks who are ready to take a deep dive into React Hooks.

Wait What?! 🤔

I know what you're thinking:

> *Dan, this code doesn't make any sense. What happened to "Just JavaScript"? Admit that React has jumped the shark with Hooks!*

I thought this too but I changed my mind, and I'm going to change yours. Before explaining why this code makes sense, I want to show off what it can do.

Why `useInterval()` Is a Better API

To remind you, my `useInterval` Hook accepts a function and a delay:

```
useInterval(() => {
  // ...
}, 1000);
```

This looks a lot like `setInterval`:

```
setInterval(() => {
  // ...
}, 1000);
```

So why not just use `setInterval` directly?

This may not be obvious at first, but the difference between the `setInterval` you know and my `useInterval` Hook is that **its arguments are "dynamic"**.

I'll illustrate this point with a concrete example.

Let's say we want the interval delay to be adjustable:

17

Delay: 100

While you wouldn't necessarily control the delay with an *input*, adjusting it dynamically can be useful — for example, to poll for some AJAX updates less often while the user has switched to a different tab.

So how would you do this with `setInterval` in a class? I ended up with this:

```
class Counter extends React.Component {
  state = {
    count: 0,
    delay: 1000,
  };

  componentDidMount() {
    this.interval = setInterval(this.tick, this.state.delay);
  }
  componentDidUpdate(prevProps, prevState) {
    if (prevState.delay !== this.state.delay) {
      clearInterval(this.interval);
      this.interval = setInterval(this.tick, this.state.delay);
    }
  }
  componentWillUnmount() {
    clearInterval(this.interval);
  }
  tick = () => {
    this.setState({
      count: this.state.count + 1
    });
  }

  handleDelayChange = (e) => {
    this.setState({ delay: Number(e.target.value) });
  }

  render() {
    return (
      <>
        <h1>{this.state.count}</h1>
        <input value={this.state.delay} onChange={this.handleDelayChange} />
      </>
    );
  }
}
```

(Here's a CodeSandbox demo.)

This is not too bad!

What's the Hook version looking like?

```
function Counter() {
  let [count, setCount] = useState(0);
  let [delay, setDelay] = useState(1000);

  useInterval(() => {
    // Your custom logic here
    setCount(count + 1);
  }, delay);

  function handleDelayChange(e) {
    setDelay(Number(e.target.value));
  }

  return (
    <>
      <h1>{count}</h1>
      <input value={delay} onChange={handleDelayChange} />
    </>
  );
}
```

(Here's a CodeSandbox demo.)

Yeah, *that's all it takes*.

Unlike the class version, there is no complexity gap for "upgrading" the `useInterval` Hook example to have a dynamically adjusted delay:

```
// Constant delay
useInterval(() => {
  setCount(count + 1);
}, 1000);

// Adjustable delay
useInterval(() => {
  setCount(count + 1);
}, delay);
```

When `useInterval` Hook sees a different delay, it set

Instead of writing code to *set* and *clear* the interval, I c with a particular delay — and our `useInterval` Hook

What if I want to temporarily *pause* my interval? I can

```
const [delay, setDelay] = useState(1000);
const [isRunning, setIsRunning] = useState(true);

useInterval(() => {
  setCount(count + 1);
}, isRunning ? delay : null);
```

(Here is a demo!)

This is what gets me excited about Hooks and React all the existing imperative APIs and create declarative AP more closely. Just like with rendering, we can **describe in time simultaneously** instead of carefully issuing co

I hope by this you're sold on `useInterval()` Hook be when we're doing it from a component.

But why is using `setInterval()` and `clearInterval` Hooks? Let's go back to our counter example and try t

First Attempt

I'll start with a simple example that just renders the in

```
function Counter() {
  const [count, setCount] = useState(0);
  return <h1>{count}</h1>;
}
```

Now I want an interval that increments it every second. It's a side effect that needs cleanup so I'm going to `useEffect()` and return the cleanup function:

+ les autres ⅔ de l'article

https://www.reddit.com/r/reactjs/comments/7y2b7h/my_office_takes_the_dan_knows_best_meme_too_far/

Gestion de state (Exemple 2)

```
function Counter() {
  const [count, setCount] = useState(0);

  useInterval(() => {
    setCount(count + 1);
  }, 1000);

  return <h1>{count}</h1>;
}

function useInterval(callback, delay) {
  const savedCallback = useRef();

  useEffect(() => {
    savedCallback.current = callback;
  });

  useEffect(() => {
    function tick() {
      savedCallback.current();
    }

    let id = setInterval(tick, delay);
    return () => clearInterval(id);
  }, [delay]);
}
```

Il suffit de mettre des callbacks dans des Refs

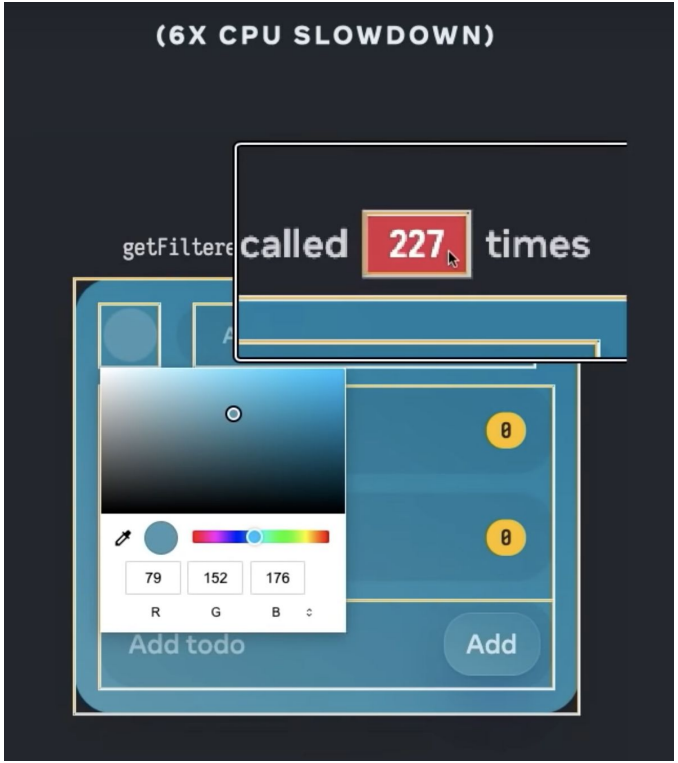
Gestion de state (Exemple 2)

```
function Counter() {
  const [count, setCount] = createSignal(1);
  const timer = setInterval(() => setCount(count() + 1), 1000);
  onCleanup(() => clearInterval(timer));

  return <h1>{count()}</h1>;
}
```

Gestion de state (Exemple 3)

```jsx
function TodoList({ visibility, themeColor }) {
  const [todos, setTodos] = useState(initialTodos);
  const handleChange = todo => setTodos(todos => getUpdated(todos, todo));
  const filtered = getFiltered(todos, visibility);

  return (
    <div>
      <ul>
        {filtered.map(todo => (
          <Todo key={todo.id} todo={todo} onChange={handleChange} />
        ))}
      </ul>
      <AddTodo setTodos={setTodos} themeColor={themeColor} />
    </div>
  );
}
```

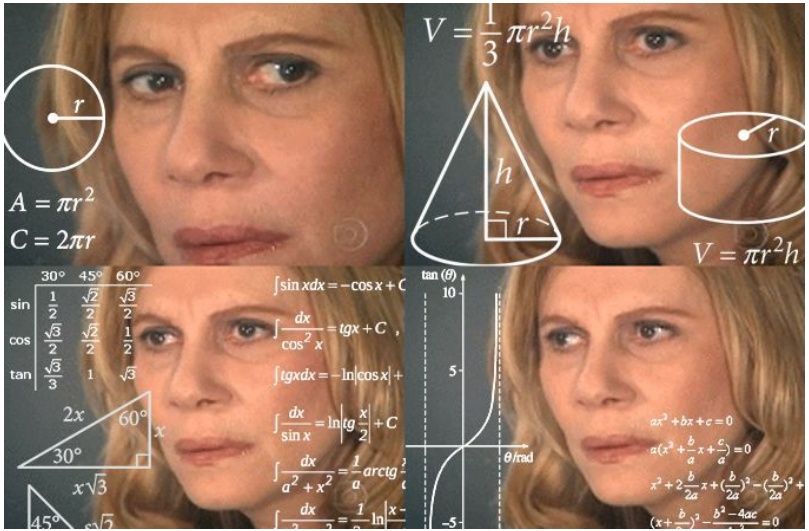
https://www.youtube.com/watch?v=lGEMwh32soc



Gestion de state (Exemple 3)

```
const Todo = React.memo(UnmemoizedTodo);

function TodoList({ visibility, themeColor }) {
  const [todos, setTodos] = useState(initialTodos);
  const handleChange = useCallback(
    todo => setTodos(todos => getUpdated(todos, todo)),
    []
  );
  const filtered = useMemo(
    () => getFiltered(todos, visibility),
    [todos, visibility]
  );
  return (
    <div>
      <ul>
        {filtered.map(todo => (
          <Todo key={todo.id} todo={todo} onChange={handleChange} />
        ))}
      </ul>
      <AddTodo setTodos={setTodos} themeColor={themeColor} />
    </div>
  );
}
```

useMemo(), memo(),
useCallback(),
[Dependencies]



https://www.youtube.com/watch?v=lGEMwh32soc

Gestion de state (Exemple 3)

React Forget
Annoncé en 2021
Pas encore sorti (en juin 2023)

```jsx
function TodoList({ visibility, themeColor }) {
  const [todos, setTodos] = useState(initialTodos);
  const handleChange = todo => setTodos(todos => getUpdated(todos, todo));
  const filtered = getFiltered(todos, visibility);

  return (
    <div>
      <ul>
        {filtered.map(todo => (
          <Todo key={todo.id} todo={todo} onChange={handleChange} />
        ))}
      </ul>
      <AddTodo setTodos={setTodos} themeColor={themeColor} />
    </div>
  );
}
```

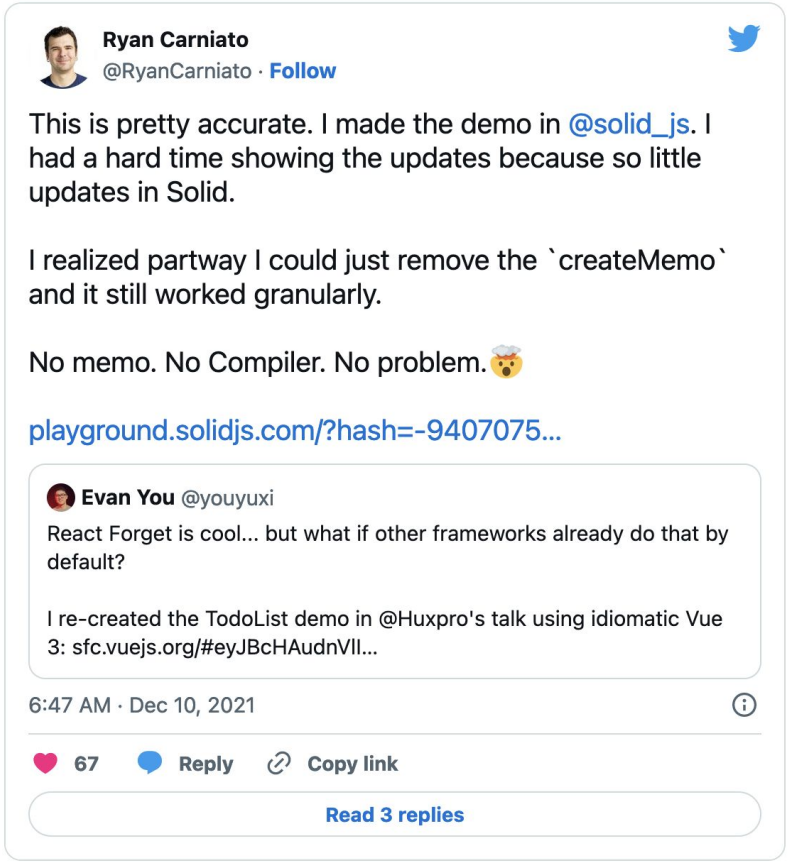
```jsx
function TodoList({ visibility, themeColor }) {
  const [todos, setTodos] = useState(initialTodos);

  let hasVisibilityChanged, hasThemeColorChanged, hasTodosChanged, memoCache;

  if (hasVisibilityChanged || hasThemeColorChanged || hasTodosChanged) {
    const handleChange =
      memoCache[0] ||
      (memoCache[0] = todo => setTodos(todos => getUpdated(todos, todo)));

    let filtered, jsx_todos;
    if (hasVisibilityChanged || hasTodosChanged) {
      filtered = memoCache[1] = getFiltered(todos, visibility);
      jsx_todos = memoCache[2] = (<ul>{filtered.map(...)}</ul>);
    } else {
      filtered = memoCache[1];
      jsx_todos = memoCache[2];
    }

    const jsx_addTodo = hasThemeColorChanged
      ? (memoCache[3] = <AddTodo setTodos={setTodos} themeColor={themeColor} />)
      : memoCache[3];

    return (memoCache[4] = <div>{jsx_todos}{jsx_addTodo}</div>);
  } else {
    return memoCache[4];
  }
}
```

https://www.youtube.com/watch?v=lGEMwh32soc

Gestion de state (Exemple 3)

```
export default function TodoList(props) {
  const [state, setState] = createStore({ todos: initialTodos });
  const handleChange = (todo) => updateTodo(todo, setState);

  return (
    <>
      <ul>
        <For each={getFiltered(state.todos, props.visibility)}>
          {(todo) => <Todo todo={todo} onChange={() => handleChange(todo)} />}
        </For>
      </ul>
      <AddTodo addTodo={addTodo(setState)} themeColor={props.themeColor} />
    </>
  );
}
```

https://playground.solidjs.com/anonymous/80e0c4b5-705e-4abd-9b33-e5ced64d52bf



https://twitter.com/RyanCarniato/status/1469181959955836931

Avantages de SolidJS

- JSX
- Patterns simples et explicites

- Signals
  - Gestion simple du state
  - Rapidité
- 7,9KB gzipped

- SSR optimisé

Local State

```
function Counter() {
  const [count, setCount] = createSignal(0);
  return <div>{count()}<div>
}
```

Global State

```
const [count, setCount] = createSignal(0);
function Counter() {
  return <div>{count()}<div>
}
```

https://dev.to/this-is-learning/making-the-case-for-signals-in-javascript-4c7i

	Solid	Autres		
select row highlighting a selected row. (5 warmup runs). 16 x CPU slowdown.	13.1 ±1.0 (1.37)	22.1 ±1.3 (2.31)	15.6 ±1.0 (1.64)	21.9 ±1.6 (2.30)
swap rows swap 2 rows for table with 1,000 rows. (5 warmup runs). 4 x CPU slowdown.	28.7 ±0.5 (1.16)	29.0 ±0.8 (1.17)	166.0 ±1.0 (6.71)	160.9 ±1.0 (6.50)
remove row removing one row. (5 warmup runs). 4 x CPU slowdown.	39.6 ±1.1 (1.03)	45.8 ±1.1 (1.20)	42.3 ±1.2 (1.10)	43.6 ±1.4 (1.14)
create many rows creating 10,000 rows. (5 warmup runs with 1k rows).	420.5 ±3.5 (1.06)	475.3 ±1.5 (1.19)	474.2 ±1.9 (1.19)	634.0 ±2.4 (1.59)

https://krausest.github.io/js-framework-benchmark/2023/table_chrome_114.0.5735.90.html

Avantages de SolidJS

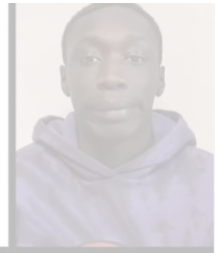
- JSX
- Patterns simples et explicites

- Signals
  - Gestion simple du state
  - Rapidité
- 7,9KB gzipped

- SSR optimisé

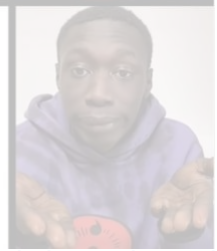
Local State

```
function Counter() {
  const [count, setCount] = createSignal(0);
  return <div>{count()}<div>
}
```



Global State

```
const [count, setCount] = createSignal(0);
function Counter() {
  return <div>{count()}<div>
}
```



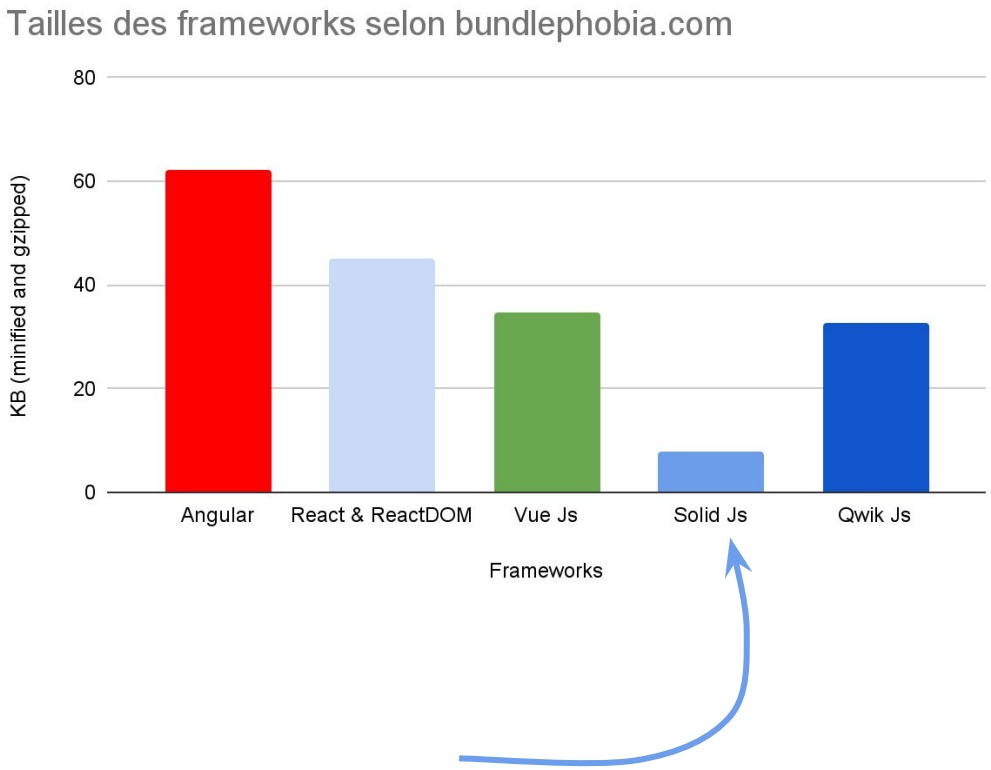
https://dev.to/this-is-learning/making-the-case-for-signals-in-javascript-4c7i

	Solid		Autres	
select row highlighting a selected row. (5 warmup runs). 16 x CPU slowdown.	13.1 ±1.0 (1.37)	22.1 ±1.3 (2.31)	15.6 ±1.0 (1.64)	21.9 ±1.6 (2.30)
swap rows swap 2 rows for table with 1,000 rows. (5 warmup runs). 4 x CPU slowdown.	28.7 ±0.5 (1.16)	29.0 ±0.8 (1.17)	166.0 ±1.0 (6.71)	160.9 ±1.0 (6.50)
remove row removing one row. (5 warmup runs). 4 x CPU slowdown.	39.6 ±1.1 (1.03)	45.8 ±1.1 (1.20)	42.3 ±1.2 (1.10)	43.6 ±1.4 (1.14)
create many rows creating 10,000 rows. (5 warmup runs with 1k rows).	420.5 ±3.5 (1.06)	475.3 ±1.5 (1.19)	474.2 ±1.9 (1.19)	634.0 ±2.4 (1.59)

https://krausest.github.io/js-framework-benchmark/2023/table_chrome_114.0.5735.90.html

Avantages de SolidJS

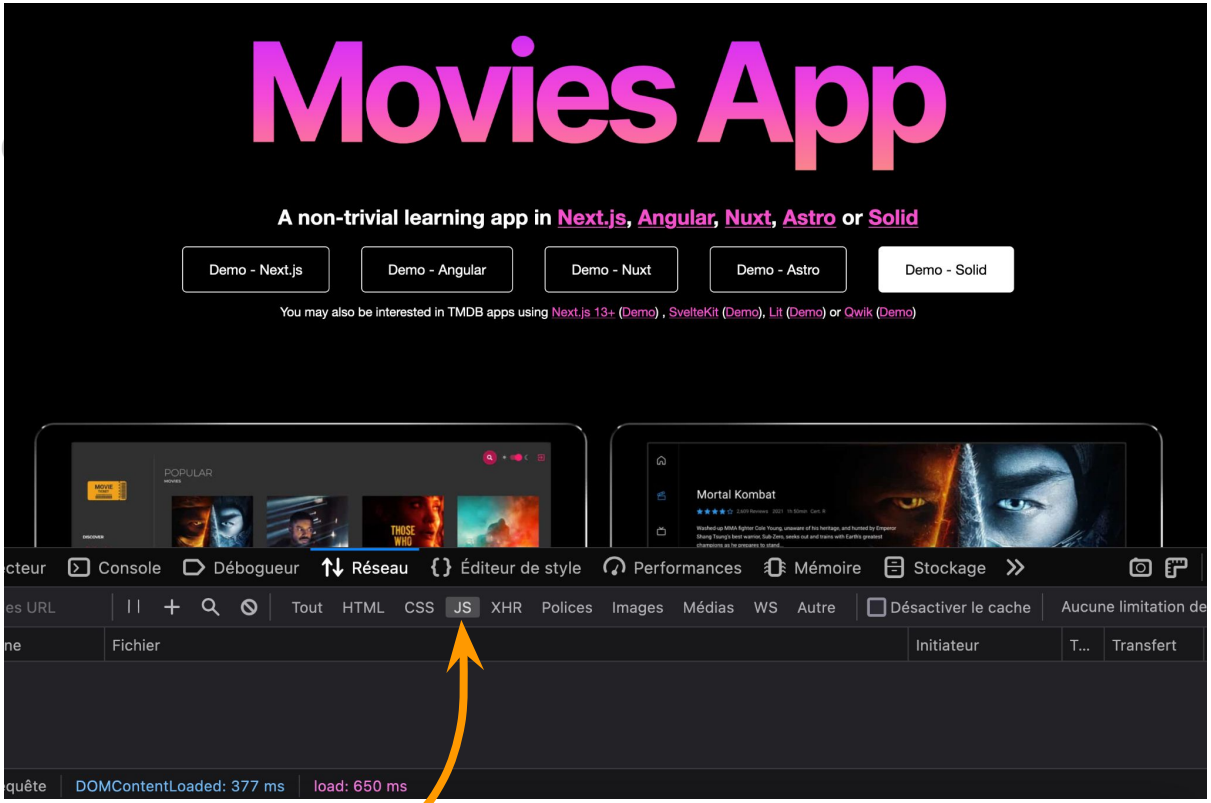
- JSX
- Patterns simples et explicites
- Signals
  - Gestion simple du state
  - Rapidité
- 7,9KB minified & gzipped
- SSR optimisé



Avantages de SolidJS

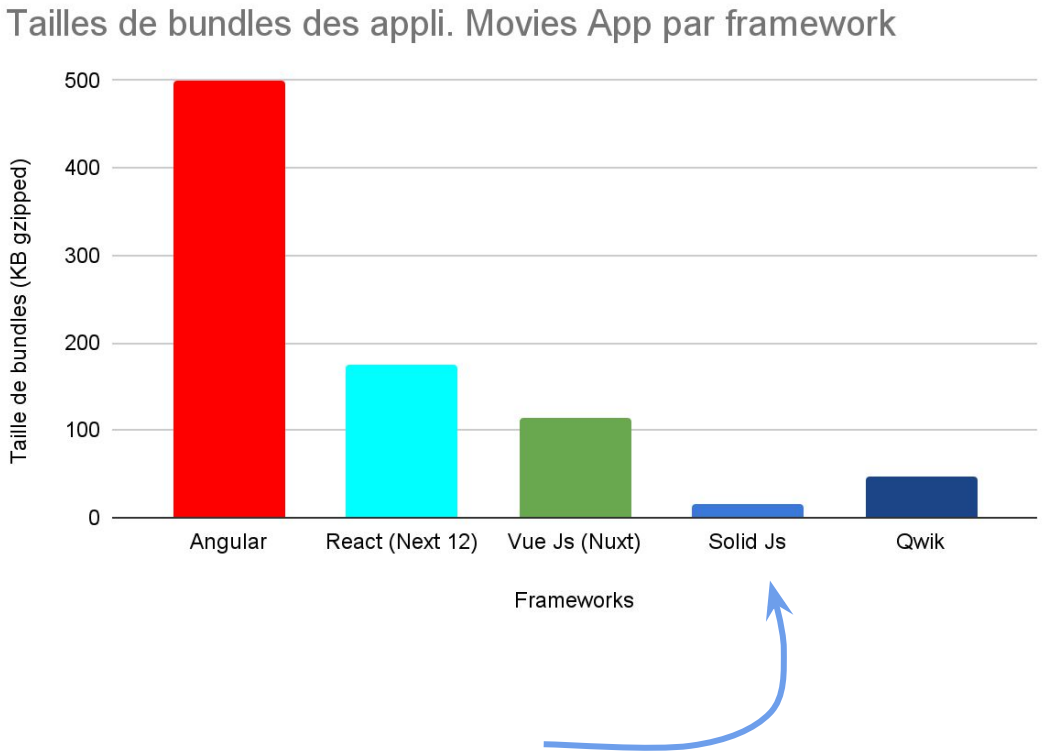
https://tastejs.com/movies/

- JSX
- Patterns simples et expli
- Signals
  - Gestion simple du state
  - Rapidité
- 7,9KB gzipped
- SSR optimisé



Avantages de SolidJS

- JSX
- Patterns simples et explicite
- Signals
  - Gestion simple du state
  - Rapidité
- 7,9KB gzipped
- SSR optimisé



Ma présentation de SolidJS (2023-01-25)
https://nawfelbgh.github.io/links/2023-01-25-SolidJS-talk.pdf

Avantages de SolidJS

- JSX
- Patterns simples et explicites

- Signals
  - Gestion simple du state
  - Rapidité
- 7,9KB gzipped

- SSR optimisé

React's true strengths: **composition, unidirectional data flow, freedom from DSLs, explicit mutation and static mental model.**

Local State

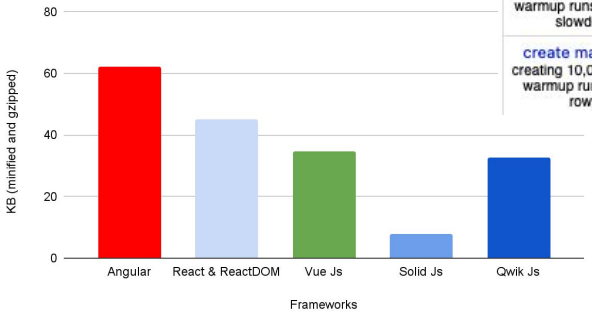
```
function Counter() {
  const [count, setCount] = createSignal(0);
  return <div>{count()}<div>
}
```

Global State

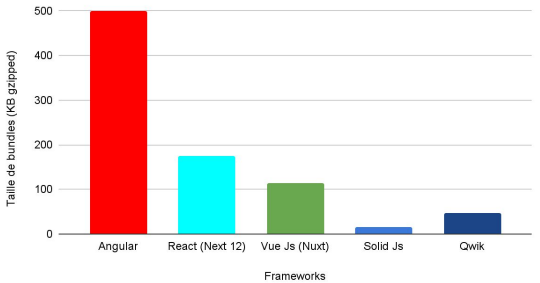
```
const [count, setCount] = createSignal(0);
function Counter() {
  return <div>{count()}<div>
}
```

select row highlighting a selected row. (5 warmup runs). 16 x CPU slowdown.	13.1 ± 1.0 (1.37)	22.1 ± 1.3 (2.31)	15.6 ± 1.0 (1.64)	21.9 ± 1.6 (2.30)
swap rows swap 2 rows for table with 1,000 rows. (5 warmup runs). 4 x CPU slowdown.	28.7 ± 0.5 (1.16)	29.0 ± 0.8 (1.17)	166.0 ± 1.0 (6.71)	160.9 ± 1.0 (6.50)
remove row removing one row. (5 warmup runs). 4 x CPU slowdown.	39.6 ± 1.1 (1.03)	45.8 ± 1.1 (1.20)	42.3 ± 1.2 (1.10)	43.6 ± 1.4 (1.14)
create many rows creating 10,000 rows. (5 warmup runs with 1k rows).				

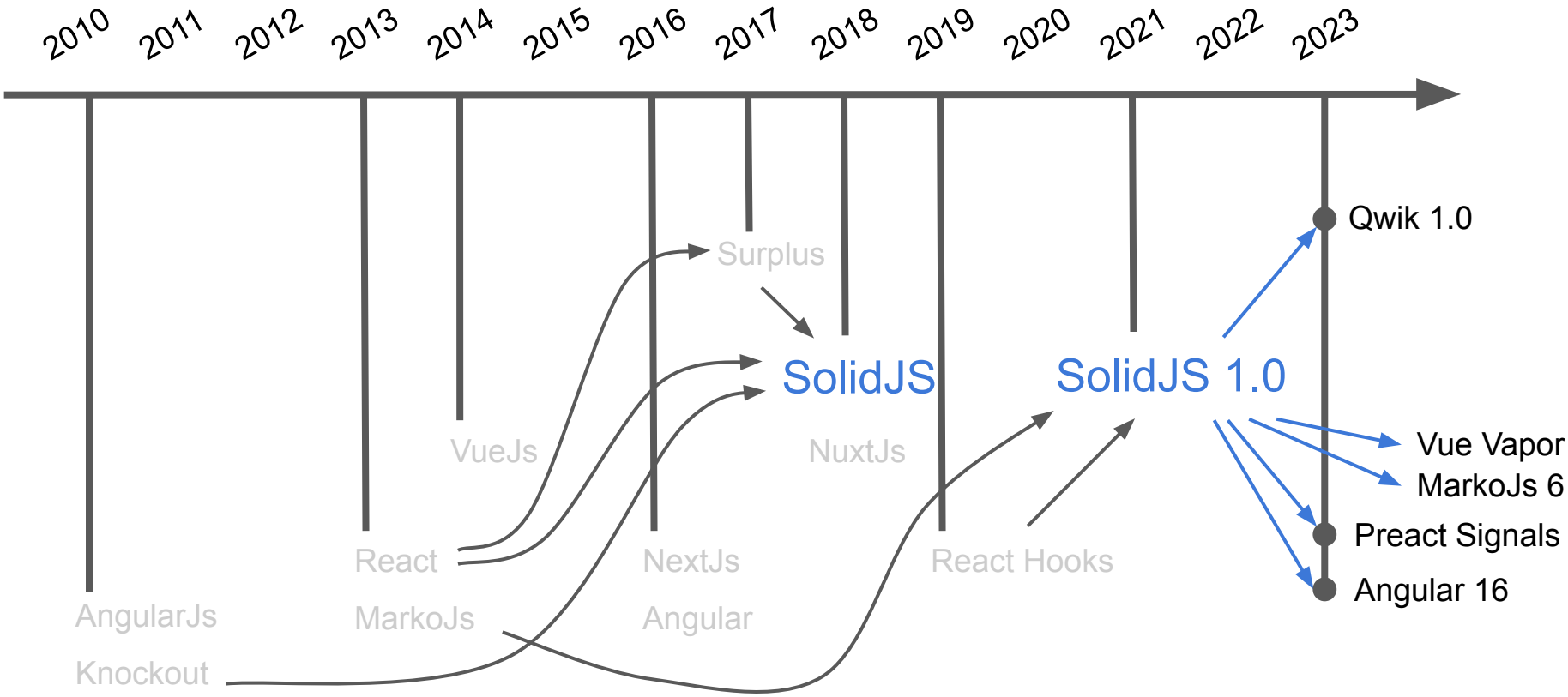
Tailles des frameworks selon bundlephobia.com



Tailles de bundles des appli. Movies App par framework



Influencés par SolidJS



Influencés par SolidJS

Angular v16 is here! 03/05/2023

https://blog.angular.io/angular-v16-is-here-4d7a28ec680d

Signals

```
@Component({
  selector: 'my-app',
  standalone: true,
  template: `
    {{ fullName() }} <button (click)="setName('John')">Click</button>
  `,
})
export class App {
  firstName = signal('Jane');
  lastName = signal('Doe');
  fullName = computed(() => `${this.firstName()} ${this.lastName()}`);

  constructor() {
    effect(() => console.log('Name changed:', this.fullName()));
  }
```

Influencés par SolidJS

2022 Year In Review - VueJs

https://blog.vuejs.org/posts/2022-year-in-review

Vapor Mode

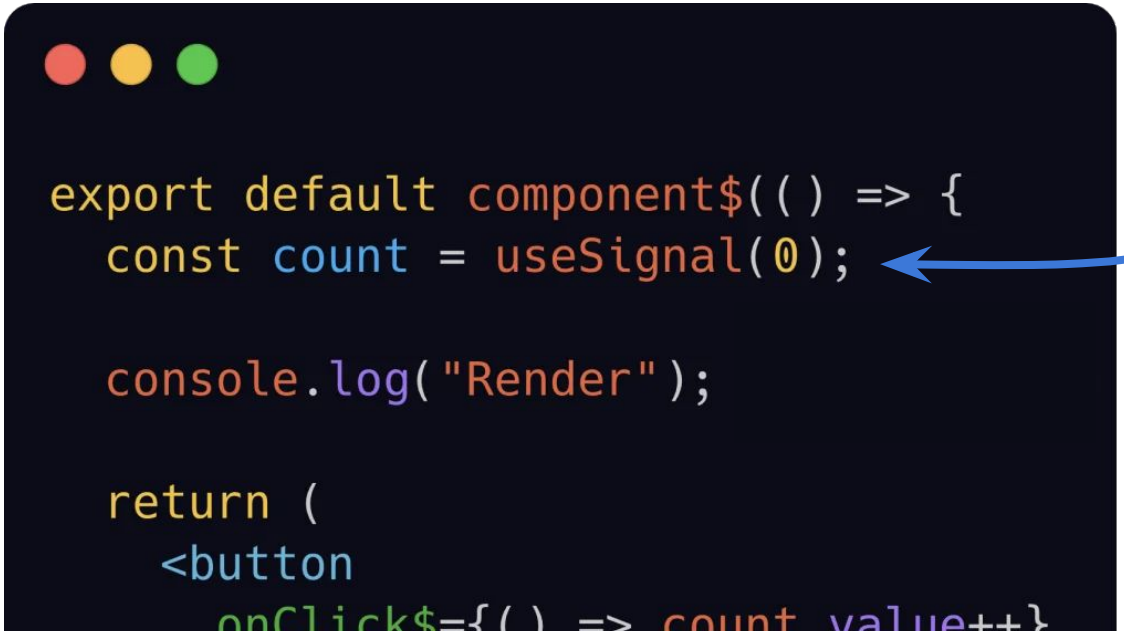
Vapor Mode is an alternative compilation strategy that we have been experimenting with, inspired by Solid. Given the same Vue SFC, Vapor Mode compiles it into JavaScript output that is more performant, uses less memory, and requires less runtime support code compared to the current Virtual DOM based output. It is still in early phase, but here are some high level points:

- Vapor Mode is intended for use cases where performance is the primary concern. It is opt-in and does not affect existing codebases.
- At the very least, you will be able to embed a Vapor component subtree into any existing Vue 3 app. Ideally, we hope to achieve granular opt-in at the component level, which means freely mixing

Influencés par SolidJS

feat: useSignal() - Qwik 05/10/2022

https://github.com/BuilderIO/qwik/pull/1363



Points négatifs de SolidJS

- Communauté réduite
- Ressemble beaucoup à React
  - Pas motivant pour les gens qui n’aiment pas les patterns de React
  - Beaucoups des utilisateurs de React pensent que SolidJS c’est un peu la même chose
- Moins connu dans le monde de l’entreprise

Conclusion

Votez pour un framework simple et performant

Votez pour SolidJS !