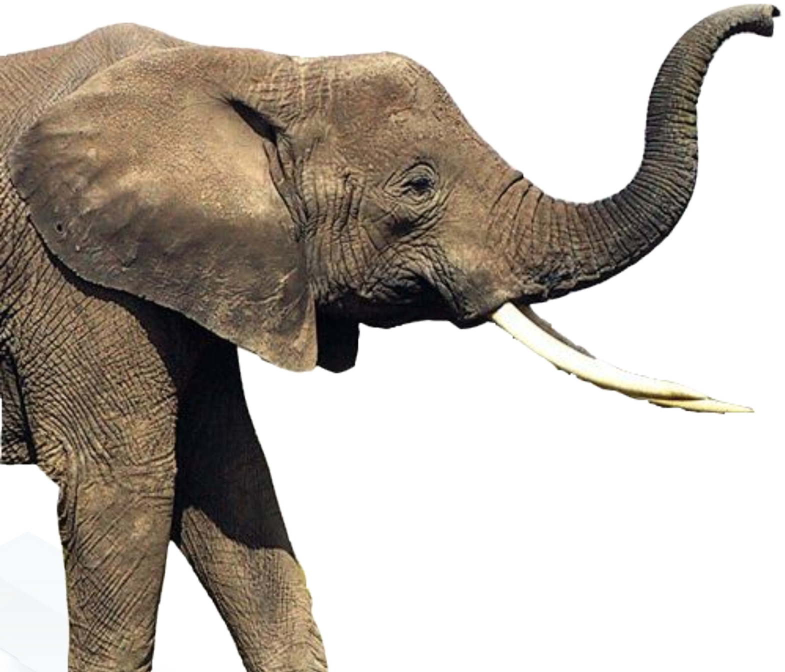
SOLIDJS

Getting Started

Docs and information available at **www.solidjs.com**

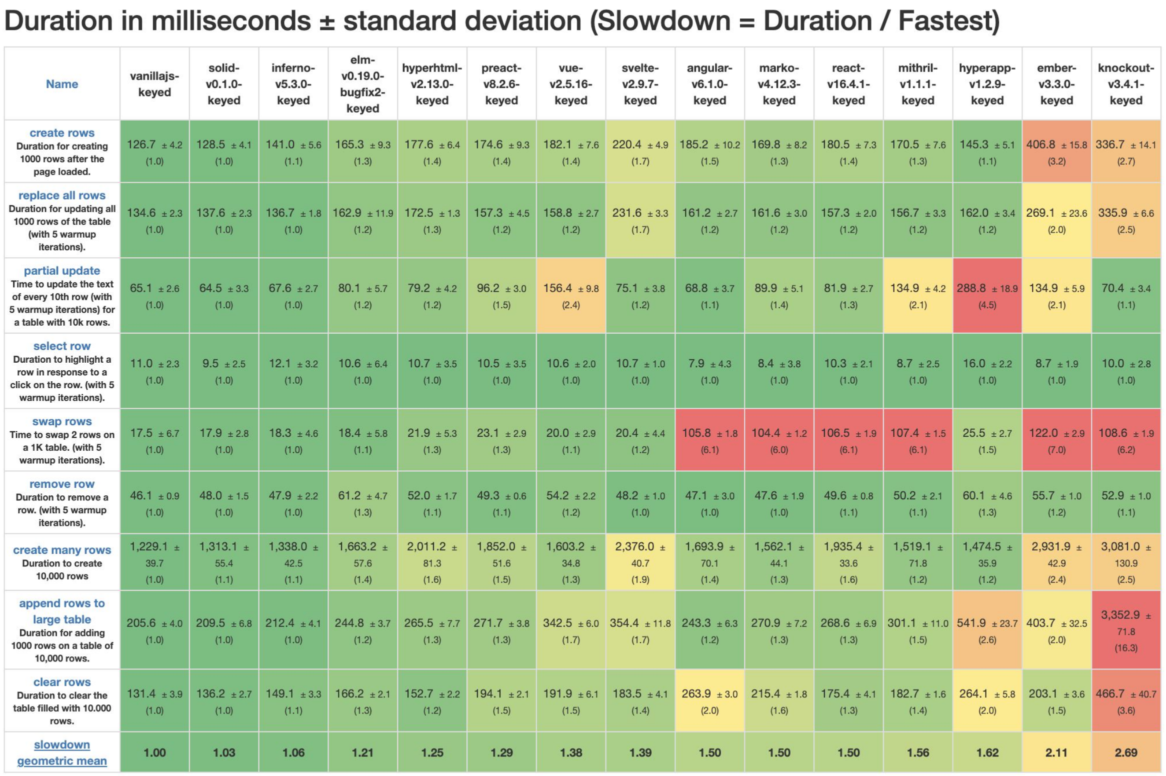
SolidJS: Origins

SolidJS: Yet another JavaScript Framework



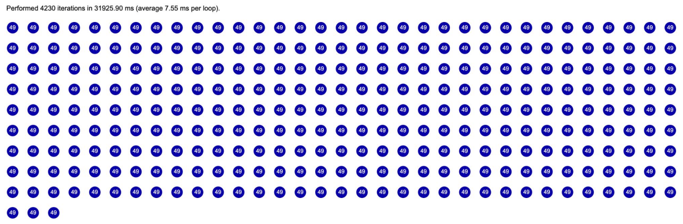
- Started development in 2016
- A return to fine-grained reactivity
- Performance without a Virtual DOM

SolidJS: Performance Champion

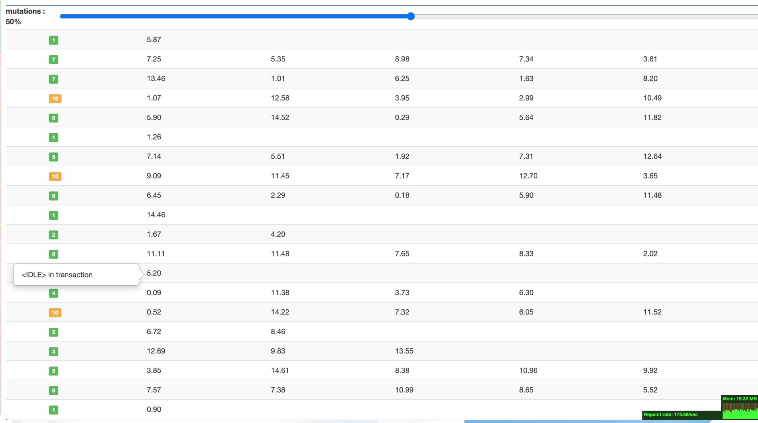


Duration in milliseconds ± standard deviation (Slowdown = Duration / Fastest)

Name	vanillajs-keyed	solid-v0.1.0-keyed	inferno-v5.3.0-keyed	elm-v0.19.0-bugfix2-keyed	hyperhtml-v2.13.0-keyed	preact-v8.2.6-keyed	vue-v2.5.16-keyed	svelte-v2.9.7-keyed	angular-v6.1.0-keyed	marko-v4.12.3-keyed	react-v16.4.1-keyed	mithril-v1.1.1-keyed	hyperapp-v1.2.9-keyed	ember-v3.3.0-keyed	knockout-v3.4.1-keyed
create rows Duration for creating 1000 rows after the page loaded.	126.7 ± 4.2 (1.0)	128.5 ± 4.1 (1.0)	141.0 ± 5.6 (1.1)	165.3 ± 9.3 (1.3)	177.6 ± 6.4 (1.4)	174.6 ± 9.3 (1.4)	182.1 ± 7.6 (1.4)	220.4 ± 4.9 (1.7)	185.2 ± 10.2 (1.5)	169.8 ± 8.2 (1.3)	180.5 ± 7.3 (1.4)	170.5 ± 7.6 (1.3)	145.3 ± 5.1 (1.1)	406.8 ± 15.8 (3.2)	336.7 ± 14.1 (2.7)
replace all rows Duration for updating all 1000 rows of the table (with 5 warmup iterations).	134.6 ± 2.3 (1.0)	137.6 ± 2.3 (1.0)	136.7 ± 1.8 (1.0)	162.9 ± 11.9 (1.2)	172.5 ± 1.3 (1.3)	157.3 ± 4.5 (1.2)	158.8 ± 2.7 (1.2)	231.6 ± 3.3 (1.7)	161.2 ± 2.7 (1.2)	161.6 ± 3.0 (1.2)	157.3 ± 2.0 (1.2)	156.7 ± 3.3 (1.2)	162.0 ± 3.4 (1.2)	269.1 ± 23.6 (2.0)	335.9 ± 6.6 (2.5)
partial update Time to update the text of every 10th row (with 5 warmup iterations) for a table with 10k rows.	65.1 ± 2.6 (1.0)	64.5 ± 3.3 (1.0)	67.6 ± 2.7 (1.0)	80.1 ± 5.7 (1.2)	79.2 ± 4.2 (1.2)	96.2 ± 3.0 (1.5)	156.4 ± 9.8 (2.4)	75.1 ± 3.8 (1.2)	68.8 ± 3.7 (1.1)	89.9 ± 5.1 (1.4)	81.9 ± 2.7 (1.3)	134.9 ± 4.2 (2.1)	288.8 ± 18.9 (4.5)	134.9 ± 5.9 (2.1)	70.4 ± 3.4 (1.1)
select row Duration to highlight a row in response to a click on the row. (with 5 warmup iterations).	11.0 ± 2.3 (1.0)	9.5 ± 2.5 (1.0)	12.1 ± 3.2 (1.0)	10.6 ± 6.4 (1.0)	10.7 ± 3.5 (1.0)	10.5 ± 3.5 (1.0)	10.6 ± 2.0 (1.0)	10.7 ± 1.0 (1.0)	7.9 ± 4.3 (1.0)	8.4 ± 3.8 (1.0)	10.3 ± 2.1 (1.0)	8.7 ± 2.5 (1.0)	16.0 ± 2.2 (1.0)	8.7 ± 1.9 (1.0)	10.0 ± 2.8 (1.0)
swap rows Time to swap 2 rows on a 1K table. (with 5 warmup iterations).	17.5 ± 6.7 (1.0)	17.9 ± 2.8 (1.0)	18.3 ± 4.6 (1.0)	18.4 ± 5.8 (1.1)	21.9 ± 5.3 (1.3)	23.1 ± 2.9 (1.3)	20.0 ± 2.9 (1.1)	20.4 ± 4.4 (1.2)	105.8 ± 1.8 (6.1)	104.4 ± 1.2 (6.0)	106.5 ± 1.9 (6.1)	107.4 ± 1.5 (6.1)	25.5 ± 2.7 (1.5)	122.0 ± 2.9 (7.0)	108.6 ± 1.9 (6.2)
remove row Duration to remove a row. (with 5 warmup iterations).	46.1 ± 0.9 (1.0)	48.0 ± 1.5 (1.0)	47.9 ± 2.2 (1.0)	61.2 ± 4.7 (1.3)	52.0 ± 1.7 (1.1)	49.3 ± 0.6 (1.1)	54.2 ± 2.2 (1.2)	48.2 ± 1.0 (1.0)	47.1 ± 3.0 (1.0)	47.6 ± 1.9 (1.0)	49.6 ± 0.8 (1.1)	50.2 ± 2.1 (1.1)	60.1 ± 4.6 (1.3)	55.7 ± 1.0 (1.2)	52.9 ± 1.0 (1.1)
create many rows Duration to create 10,000 rows	1,229.1 ± 39.7 (1.0)	1,313.1 ± 55.4 (1.1)	1,338.0 ± 42.5 (1.1)	1,663.2 ± 57.6 (1.4)	2,011.2 ± 81.3 (1.6)	1,852.0 ± 51.6 (1.5)	1,603.2 ± 34.8 (1.3)	2,376.0 ± 40.7 (1.9)	1,693.9 ± 70.1 (1.4)	1,562.1 ± 44.1 (1.3)	1,935.4 ± 33.6 (1.6)	1,519.1 ± 71.8 (1.2)	1,474.5 ± 35.9 (1.2)	2,931.9 ± 42.9 (2.4)	3,081.0 ± 130.9 (2.5)
append rows to large table Duration for adding 1000 rows on a table of 10,000 rows.	205.6 ± 4.0 (1.0)	209.5 ± 6.8 (1.0)	212.4 ± 4.1 (1.0)	244.8 ± 3.7 (1.2)	265.5 ± 7.7 (1.3)	271.7 ± 3.8 (1.3)	342.5 ± 6.0 (1.7)	354.4 ± 11.8 (1.7)	243.3 ± 6.3 (1.2)	270.9 ± 7.2 (1.3)	268.6 ± 6.9 (1.3)	301.1 ± 11.0 (1.5)	541.9 ± 23.7 (2.6)	403.7 ± 32.5 (2.0)	3,352.9 ± 71.8 (16.3)
clear rows Duration to clear the table filled with 10.000 rows.	131.4 ± 3.9 (1.0)	136.2 ± 2.7 (1.0)	149.1 ± 3.3 (1.1)	166.2 ± 2.1 (1.3)	152.7 ± 2.2 (1.2)	194.1 ± 2.1 (1.5)	191.9 ± 6.1 (1.5)	183.5 ± 4.1 (1.4)	263.9 ± 3.0 (2.0)	215.4 ± 1.8 (1.6)	175.4 ± 4.1 (1.3)	182.7 ± 1.6 (1.4)	264.1 ± 5.8 (2.0)	203.1 ± 3.6 (1.5)	466.7 ± 40.7 (3.6)
slowdown geometric mean	**1.00**	**1.03**	**1.06**	**1.21**	**1.25**	**1.29**	**1.38**	**1.39**	**1.50**	**1.50**	**1.50**	**1.56**	**1.62**	**2.11**	**2.69**



Performed 4230 iterations in 31925.90 ms (average 7.55 ms per loop).

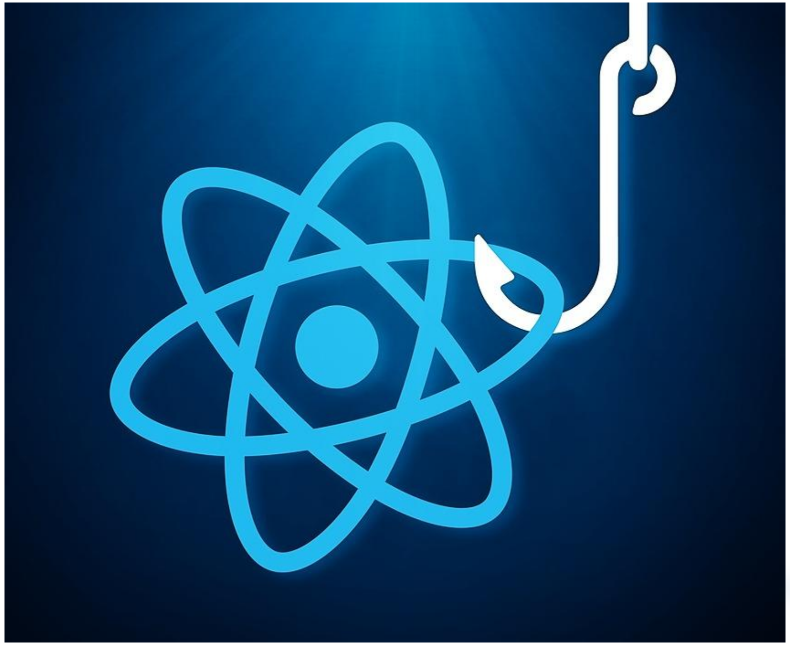


mutations : 50%

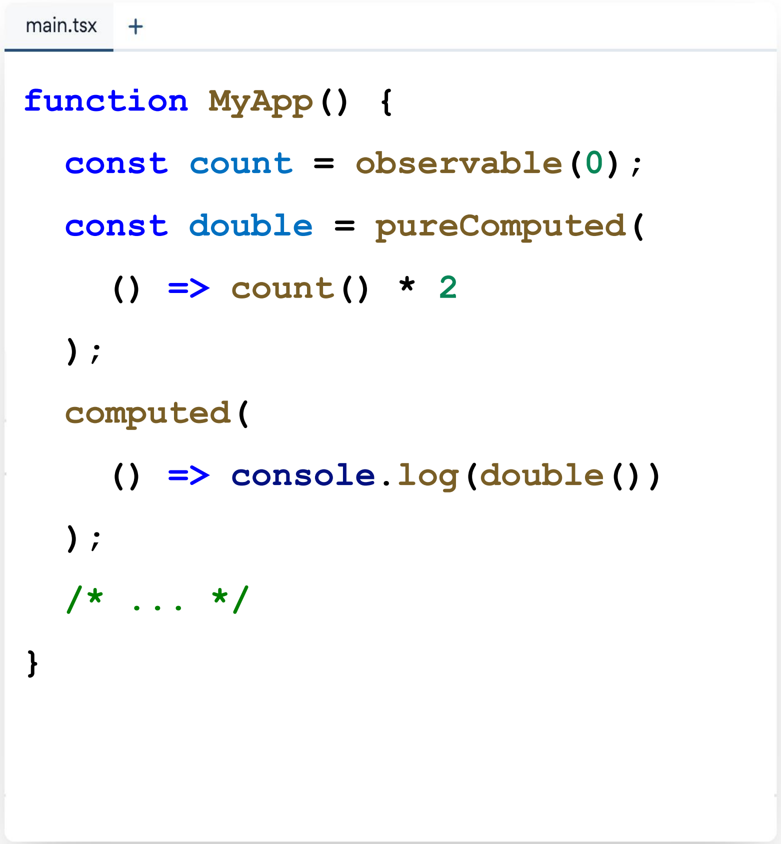
1	5.87				
7	7.25	5.35	8.98	7.34	3.61
7	13.46	1.01	6.25	1.63	8.20
10	1.07	12.58	3.95	2.99	10.49
6	5.90	14.52	0.29	5.64	11.82
1	1.26				
5	7.14	5.51	1.92	7.31	12.64
10	9.09	11.45	7.17	12.70	3.65
6	6.45	2.29	0.18	5.90	11.48
1	14.46				
2	1.67	4.20			
8	11.11	11.48	7.65	8.33	2.02
<IDLE> in transaction	5.20				
4	0.09	11.38	3.73	6.30	
10	0.52	14.22	7.32	6.05	11.52
2	6.72	8.46			
3	12.69	9.83	13.55		
5	3.85	14.61	8.38	10.96	9.92
6	7.57	7.38	10.99	8.65	5.52
1	0.90				

Enter React Hooks

- Return to primitives
- Adopted almost in every framework
- They look a lot like reactive primitives

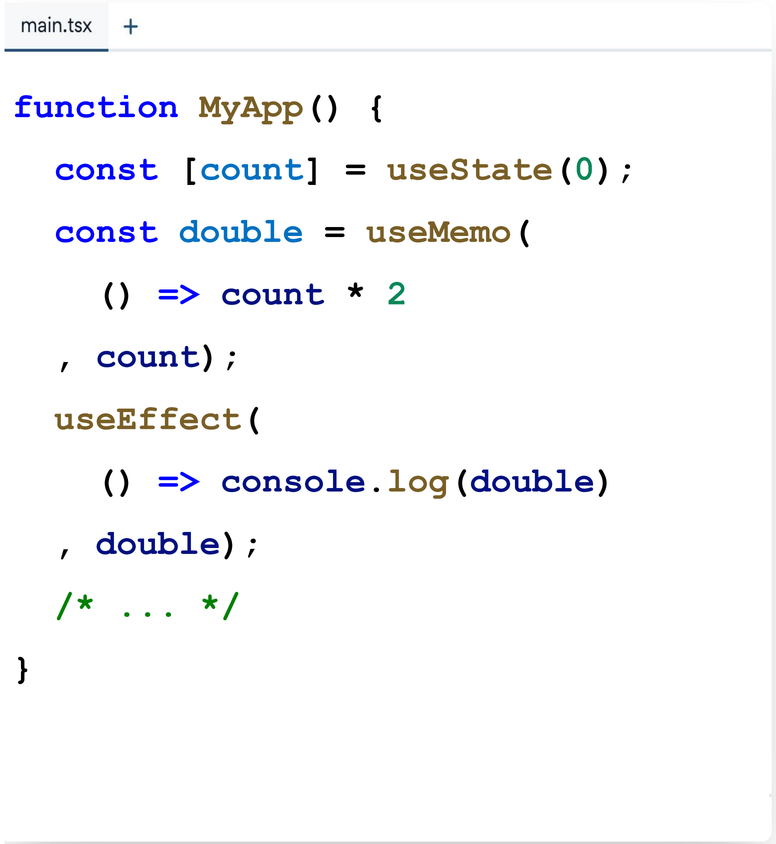


Reactivity vs Hooks



main.tsx +

```
function MyApp() {
  const count = observable(0);
  const double = pureComputed(
    () => count() * 2
  );
  computed(
    () => console.log(double())
  );
  /* ... */
}
```

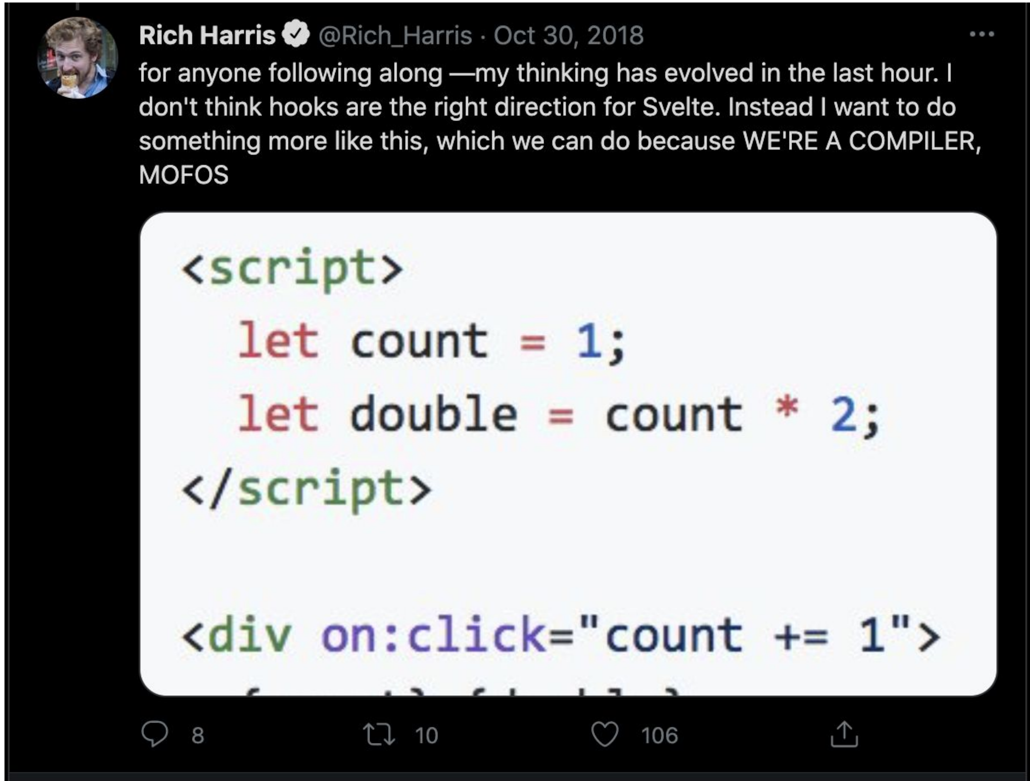


main.tsx +

```
function MyApp() {
  const [count] = useState(0);
  const double = useMemo(
    () => count * 2
  , count);
  useEffect(
    () => console.log(double)
  , double);
  /* ... */
}
```

Primitives everywhere

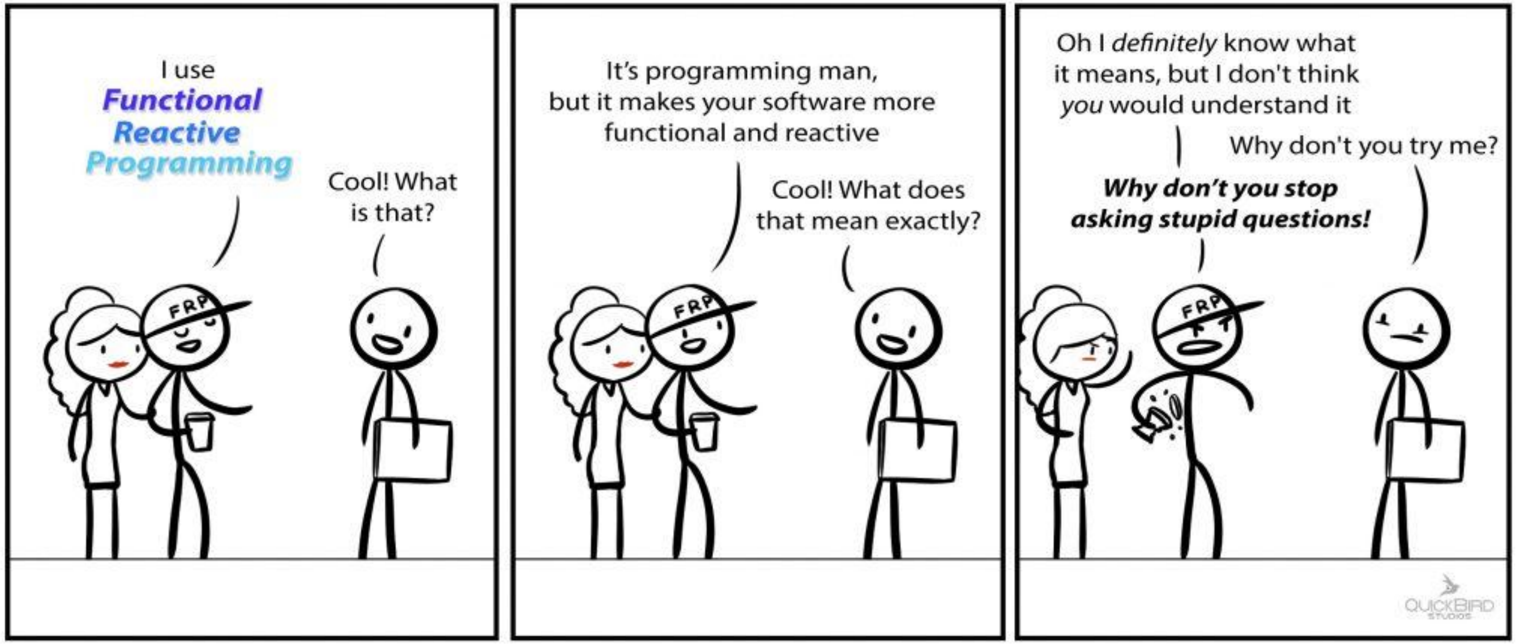
- React Hooks
- Reactivity as a language
- Composition API
- Solid's primitives
- Common Hooks for Web Components



Rich Harris @Rich_Harris · Oct 30, 2018

for anyone following along —my thinking has evolved in the last hour. I don't think hooks are the right direction for Svelte. Instead I want to do something more like this, which we can do because WE'RE A COMPILER, MOFOS

```
<script>
  let count = 1;
  let double = count * 2;
</script>

<div on:click="count += 1">
```


I use Functional Reactive Programming
Cool! What is that?
It's programming man, but it makes your software more functional and reactive
Cool! What does that mean exactly?
Oh I *definitely* know what it means, but I don't think *you* would understand it
Why don't you try me?
Why don't you stop asking stupid questions!
QUICKBIRD STUDIOS

SolidJS: Reactivity

What’s Reactive Programming?

$$a = b + c$$

* where the value of **a** updates whenever the value of **b** or **c** changes.

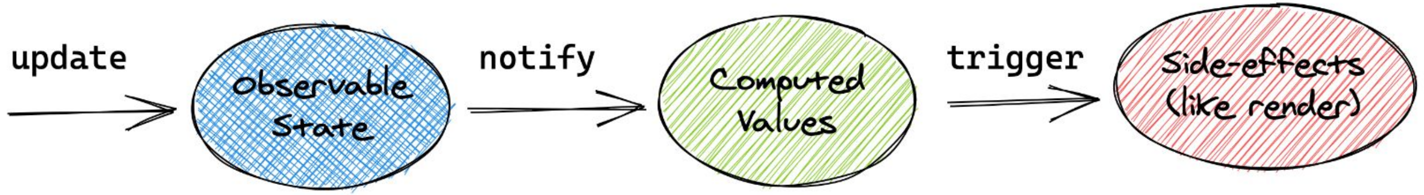
Why Reactive Programming

Declarative

Composable

Simple model consists of only 3 concepts:

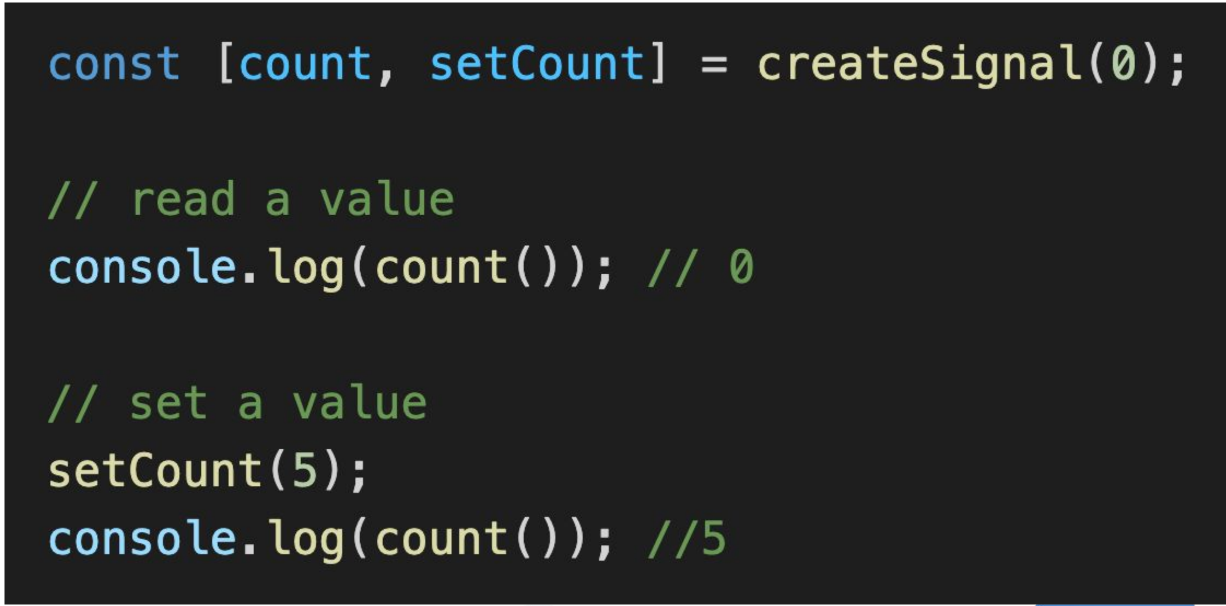
- Signals
- Derivations
- Effects



Signals

Getter, Setter, and a value

Also known as Observable, Ref, Atom, Behavior

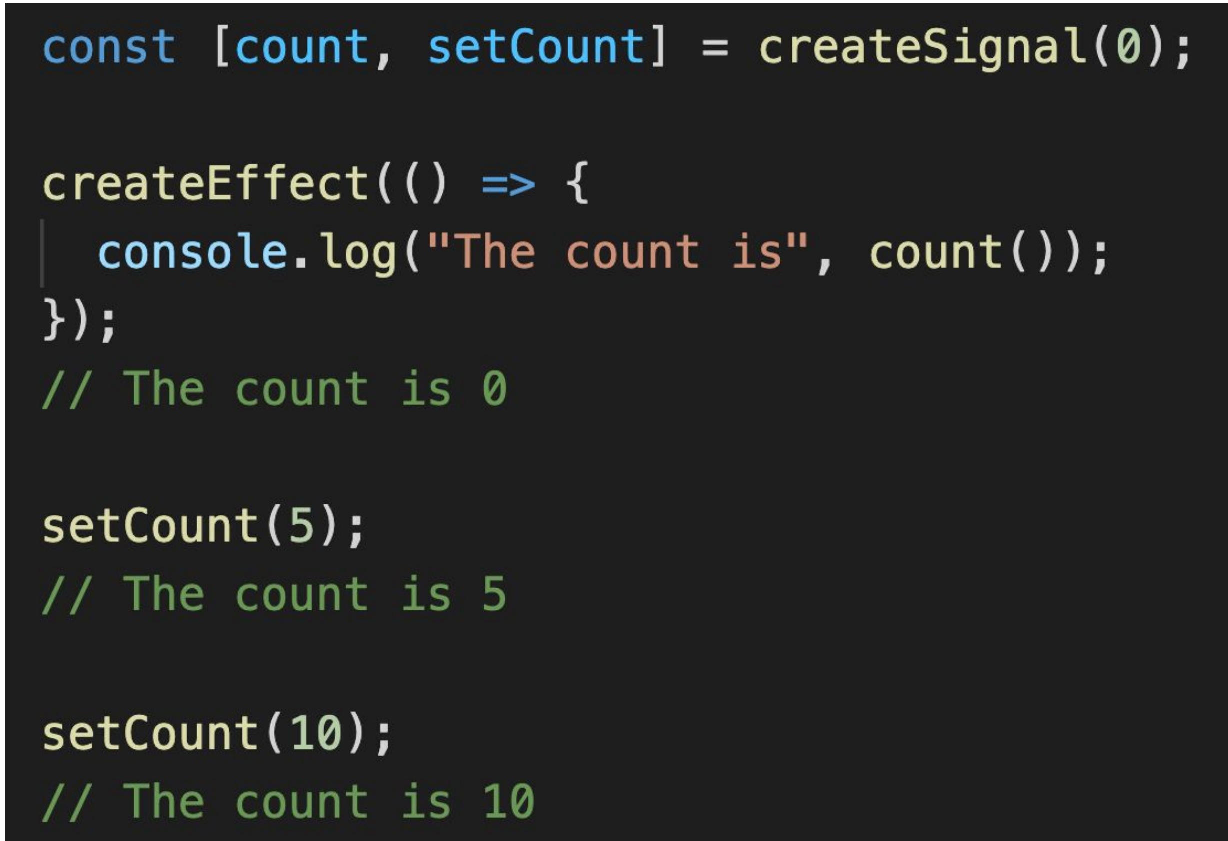


```
const [count, setCount] = createSignal(0);

// read a value
console.log(count()); // 0

// set a value
setCount(5);
console.log(count()); //5
```

Effects

Creates Side Effects

Also known as: Reactions, Autoruns, Watches, Computeds



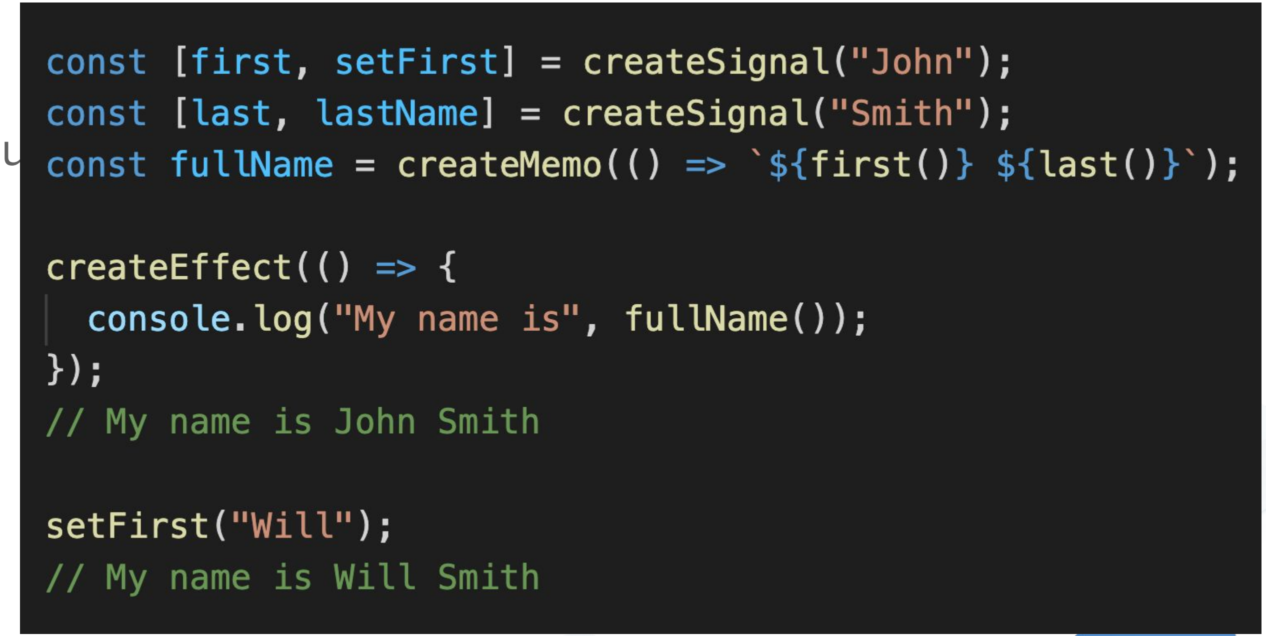
```js
const [count, setCount] = createSignal(0);

createEffect(() => {
  console.log("The count is", count());
});
// The count is 0

setCount(5);
// The count is 5

setCount(10);
// The count is 10
```

Derivations

Both observer and a signal

Only re-calculates when valu
of dependencies change

Also known as Computeds,
Memos, Selectors



```
const [first, setFirst] = createSignal("John");
const [last, lastName] = createSignal("Smith");
const fullName = createMemo(() => `${first()} ${last()}`);

createEffect(() => {
  console.log("My name is", fullName());
});
// My name is John Smith

setFirst("Will");
// My name is Will Smith
```

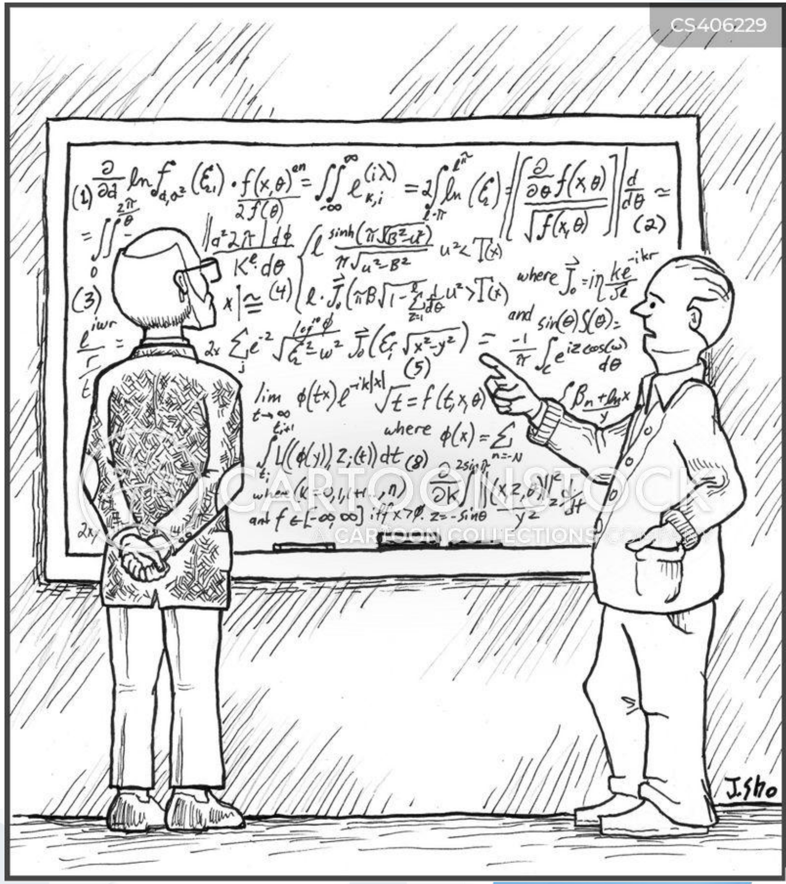
Why Derivations?

Cache work from expensive computations

Used in more than one computation

One of multiple dependencies in computation

What can be derived, should be derived

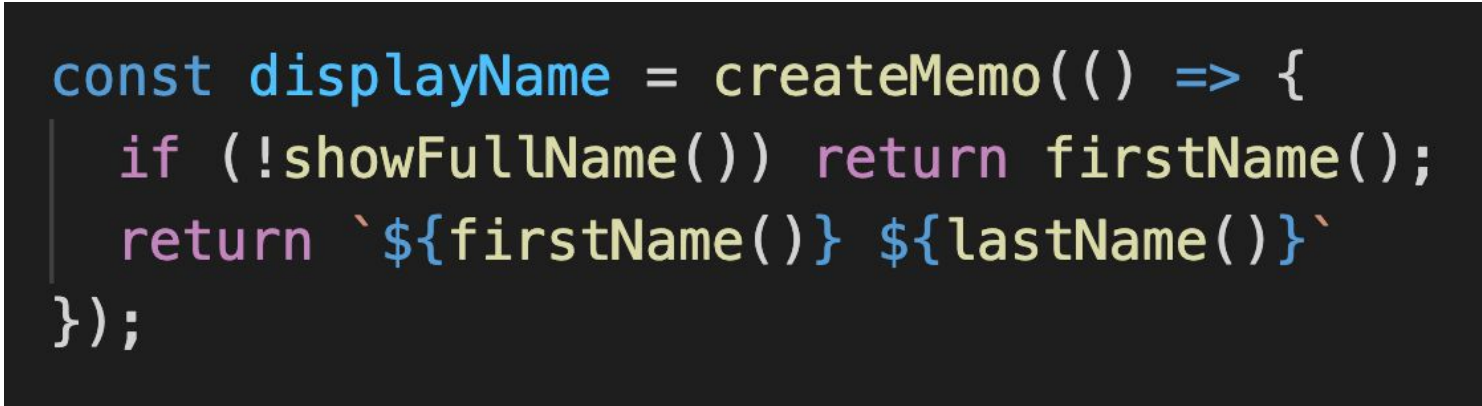


Dynamic Tracking

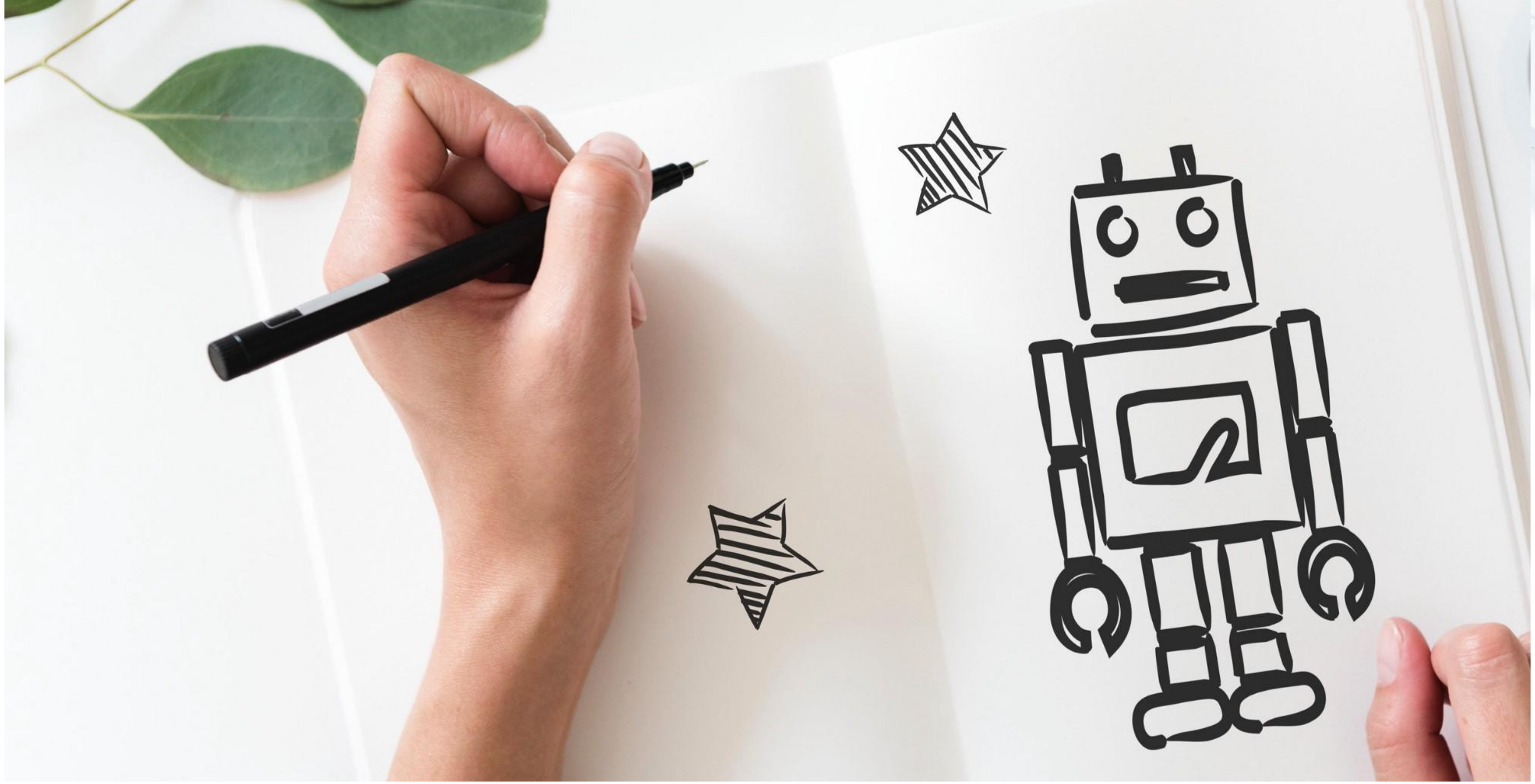
Every execution dependencies are cleaned up and collected again.

This ensures that only currently dependencies are tracked.

This is something that can only feasibly be done at runtime.



```
const displayName = createMemo(() => {
  if (!showFullName()) return firstName();
  return `${firstName()} ${lastName()}`
});
```



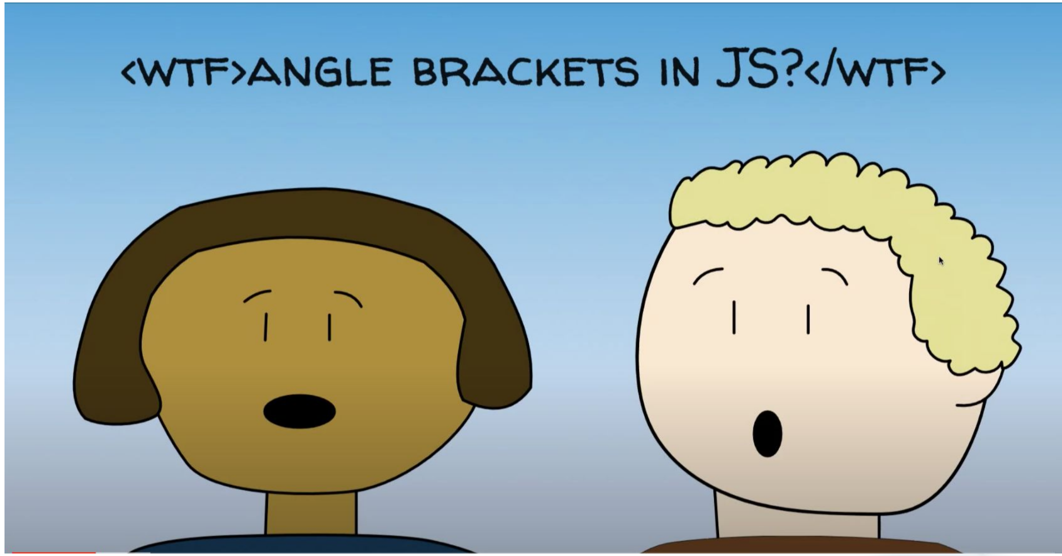
SolidJS: Rendering

Introducing JSX

JSX is a XML syntax in JavaScript popularized by React.

Describe your view inside your JavaScript.

Convenient syntax sugar for the DOM.



React’s JSX

```
function Counter() {
  const [count, setCount] = createSignal(0);
  return <h2>{count()}</h2>;
}
```

```
function Counter() {
  const [count, setCount] = createSignal(0);
  return createElement("h2", {}, count());
}
```

Reactive JSX

```
function Counter() {
  const [count, setCount] = createSignal(0);
  return <h2>{count()}</h2>;
}
```

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

  const el = document.createElement("h2");
  createEffect(() => {
    el.textContent = count();
  });
  return el;
}
```

Making a Counter in Solid

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

function Counter() {
  const [count, setCount] = createSignal(0);
  const id = setInterval(() => {
    setCount(count() + 1)
  }, 1000);
  onCleanup(() => clearInterval(id));

  return <h2>{count()}</h2>;
}
```

Controlling Flow

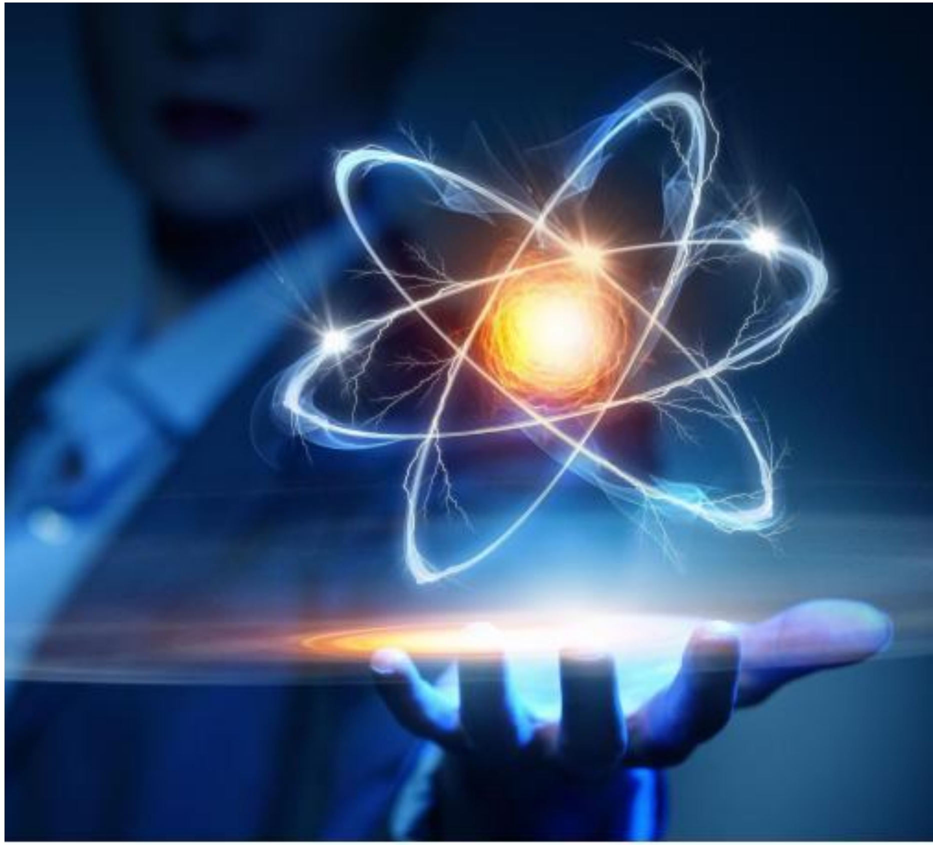
```
<ul>{
  <Paginated each={list()}>{
    (item) => <li>{item}</li>}
  </Paginated>
</ul>>
```

Reactive Advantage

Components Run Once

Templates compile to Real DOM Nodes

State Independent of Component



SolidJS: Getting Started

Single Page App Starters

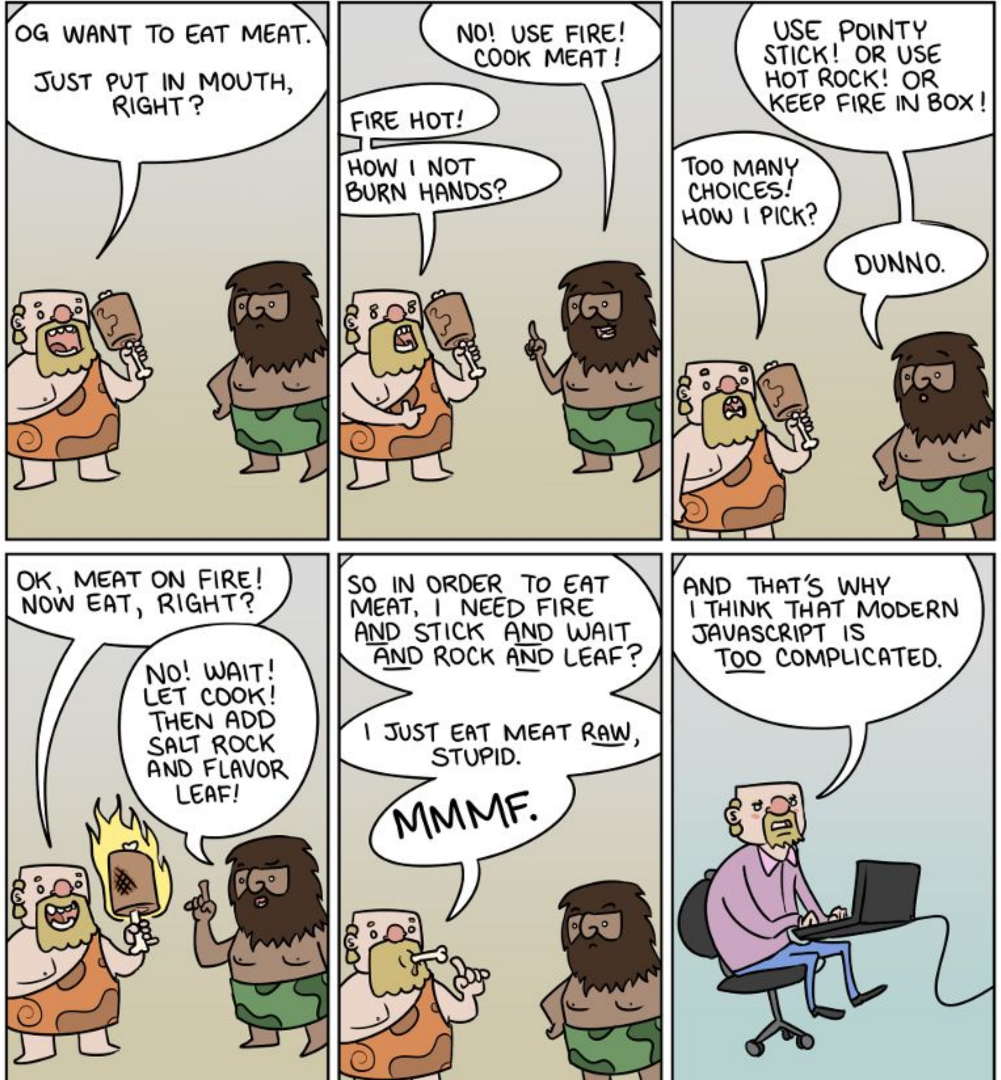
```
> npx degit solidjs/templates/js my-app
> cd my-app
> npm i # or yarn or pnpm
> npm run dev # or yarn or pnpm
```

Static Site Generation with Astro

```
> cd my-app
> npm init astro # select Solid
> npm i
> npm run dev
```

SolidStart: Adaptive Server Side Rendering

```
> cd my-app
> npm init solid@next
> npm i # or yarn or pnpm
> npm run dev # or yarn or pnpm
```


OG WANT TO EAT MEAT.
JUST PUT IN MOUTH, RIGHT?
NO! USE FIRE! COOK MEAT!
FIRE HOT!
HOW I NOT BURN HANDS?
USE POINTY STICK! OR USE HOT ROCK! OR KEEP FIRE IN BOX!
TOO MANY CHOICES! HOW I PICK?
DUNNO.
OK, MEAT ON FIRE! NOW EAT, RIGHT?
NO! WAIT! LET COOK! THEN ADD SALT ROCK AND FLAVOR LEAF!
SO IN ORDER TO EAT MEAT, I NEED FIRE AND STICK AND WAIT AND ROCK AND LEAF?
I JUST EAT MEAT RAW, STUPID.
MMMF.
AND THAT'S WHY I THINK THAT MODERN JAVASCRIPT IS TOO COMPLICATED.

site: https://solidjs.com
twitter: @solid_js
github: https://github.com/solidjs