IT'S TIME TO TALK ABOUT

SIGNALS

and the carcinisation of the web

```
let counter = 1;
let isEven = counter % 2 === 0;
let parity = isEven ? "even" : "odd";
```

HTTPS://EN.WIKIPEDIA.ORG/WIKI/REACTIVE_PROGRAMMING

```
let counter = 1;
let isEven = counter % 2 === 0;
let parity = isEven ? "even" : "odd";

// Whenever counter changes
counter = 6;
```

HTTPS://EN.WIKIPEDIA.ORG/WIKI/REACTIVE_PROGRAMMING

```
let counter = 1;
let isEven = counter % 2 === 0;
let parity = isEven ? "even" : "odd";

// Whenever counter changes
counter = 6;
isEven = counter % 2 === 0;
parity = isEven ? "even" : "odd";
```

HTTPS://EN.WIKIPEDIA.ORG/WIKI/REACTIVE_PROGRAMMING

```
let counter = 1;
let isEven = counter % 2 === 0;
let parity = isEven ? "even" : "odd";

// Whenever counter changes
counter = 6;
isEven = counter % 2 === 0;
parity = isEven ? "even" : "odd";
```

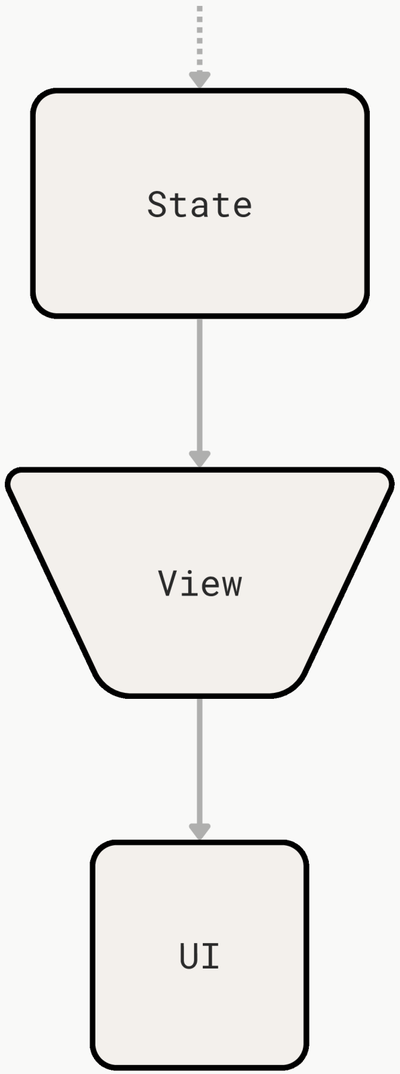
```
let counter = 1;
const isEven $= counter % 2 === 0;
const parity $= isEven ? "even" : "odd";
```

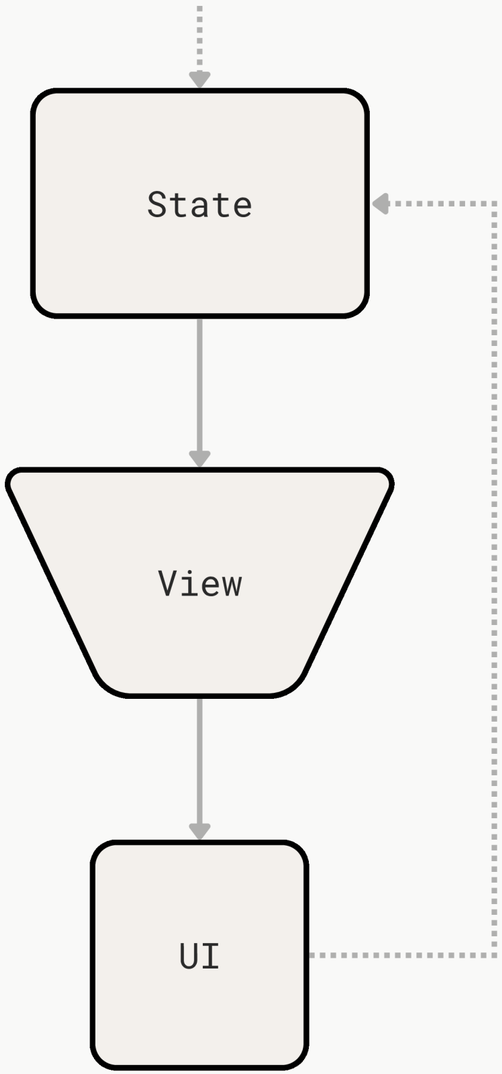
HTTPS://EN.WIKIPEDIA.ORG/WIKI/REACTIVE_PROGRAMMING

```
let counter = 1;
let isEven = counter % 2 === 0;
let parity = isEven ? "even" : "odd";

// Whenever counter changes
counter = 6;
isEven = counter % 2 === 0;
parity = isEven ? "even" : "odd";
```

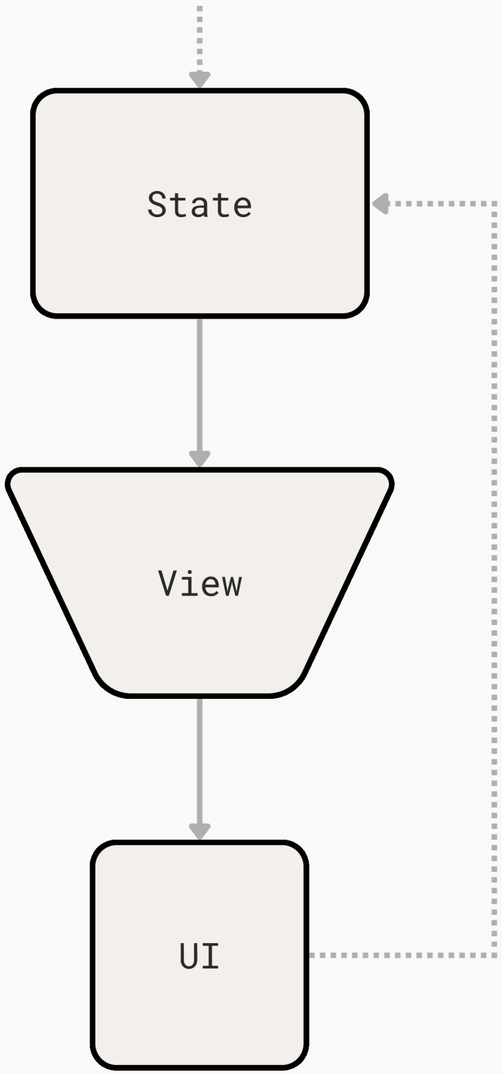
```
let counter = 1;
const isEven $= counter % 2 === 0;
const parity $= isEven ? "even" : "odd";

// Changing `counter` triggers re-calc
counter = 6;
// → isEven = true
// → parity = "even"
```

HTTPS://EN.WIKIPEDIA.ORG/WIKI/REACTIVE_PROGRAMMING

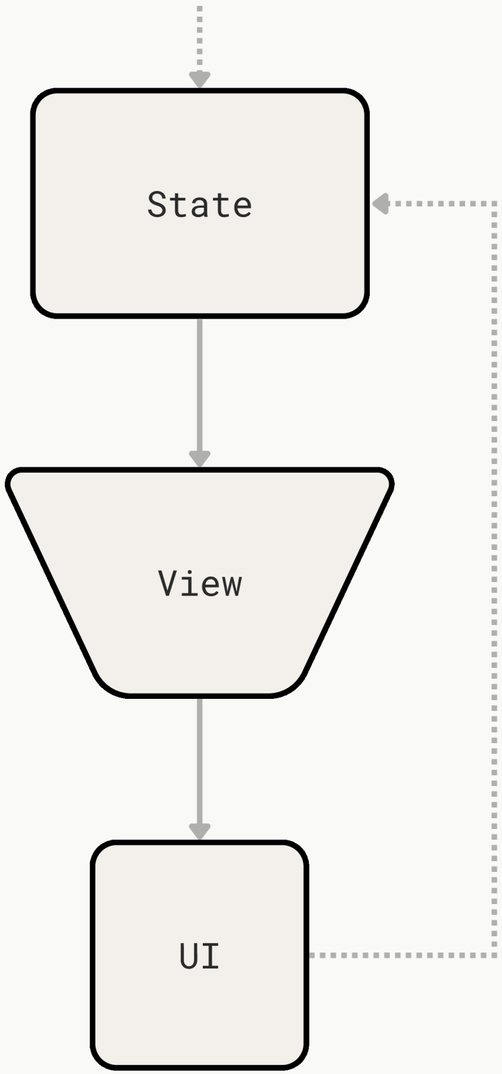

State
View
UI



Your View is a pure function of the Model.



Your UI is the result of a pure function of the State.



Your UI is the result of a pure function of the State.

For a given state, no matter how you arrived at that state, the output of the system is always the same

HTTPS://WWW.PZURAQ.COM/BLOG/HOW-AUTOTRACKING-WORKS

A BRIEF HISTORY OF REACTIVITY ON THE WEB

1995 – HTML

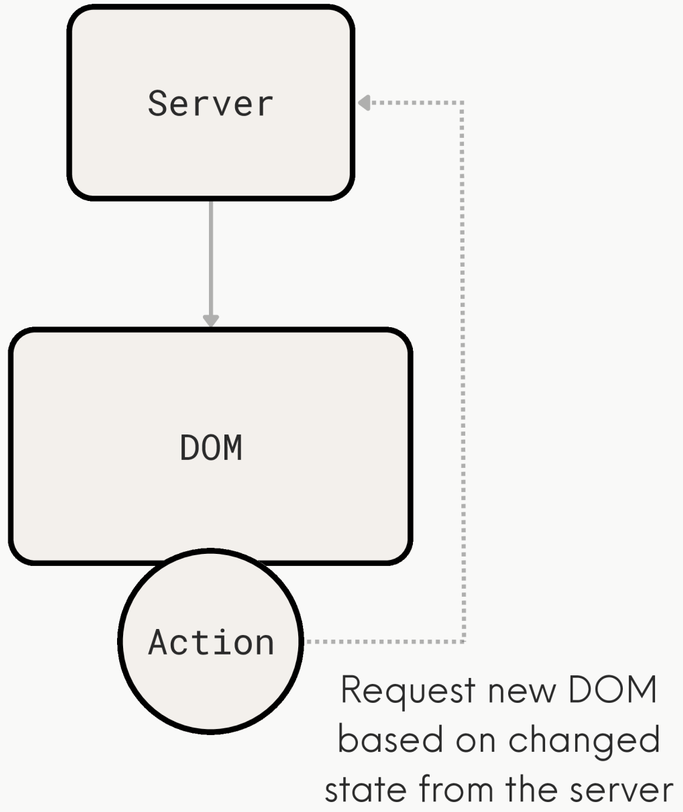
& server side, e.g. PHP

```php
<?php
  $counter = $_GET["counter"];
?>

<html>
<body>
  <a href="?counter=<?php echo $counter + 1; ?>">👍</a>
  <a href="?counter=<?php echo $counter - 1; ?>">👎</a>
  <p>Counter: <?php echo $counter; ?></p>
  <p>Parity: <?php echo $counter % 2 ? "odd" : "even"; ?></p>
</body>
</html>
```

1995 – HTML

& server side, e.g. PHP

```php
<?php
  $counter = $_GET["counter"];
?>

<html>
<body>
  <a href="?counter=<?php echo $counter + 1; ?>">👍</a>
  <a href="?counter=<?php echo $counter - 1; ?>">👎</a>
  <p>Counter: <?php echo $counter; ?></p>
  <p>Parity: <?php echo $counter % 2 ? "odd" : "even"; ?></p>
</body>
</html>
```

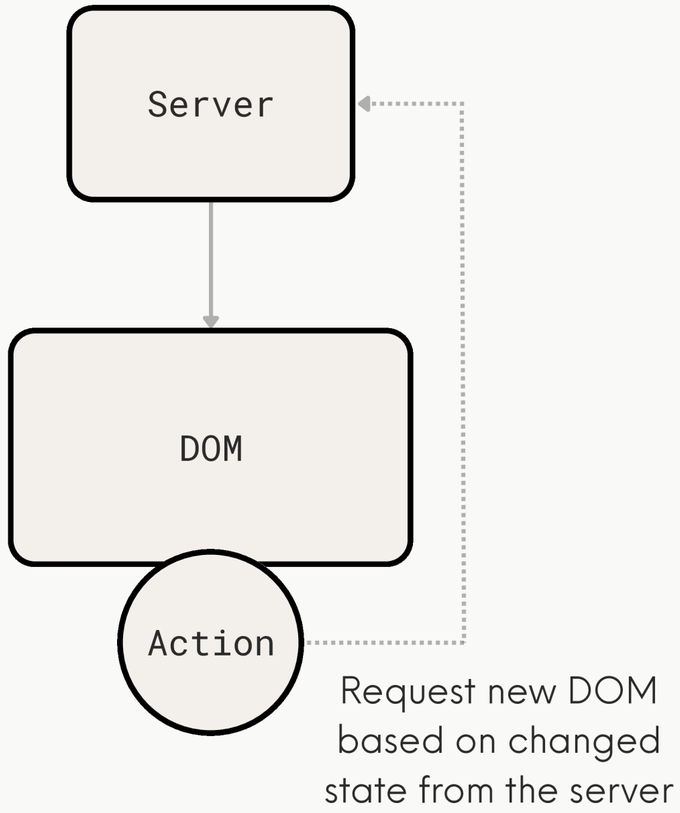


1995 – HTML

& server side, e.g. PHP

```php
<?php
  $counter = $_GET["counter"];
?>

<html>
<body>
  <a href="?counter=<?php echo $counter + 1; ?>">👍</a>
  <a href="?counter=<?php echo $counter - 1; ?>">👎</a>
  <p>Counter: <?php echo $counter; ?></p>
  <p>Parity: <?php echo $counter % 2 ? "odd" : "even"; ?></p>
</body>
</html>
```

“PULL”



2006 – jQuery and AJAX

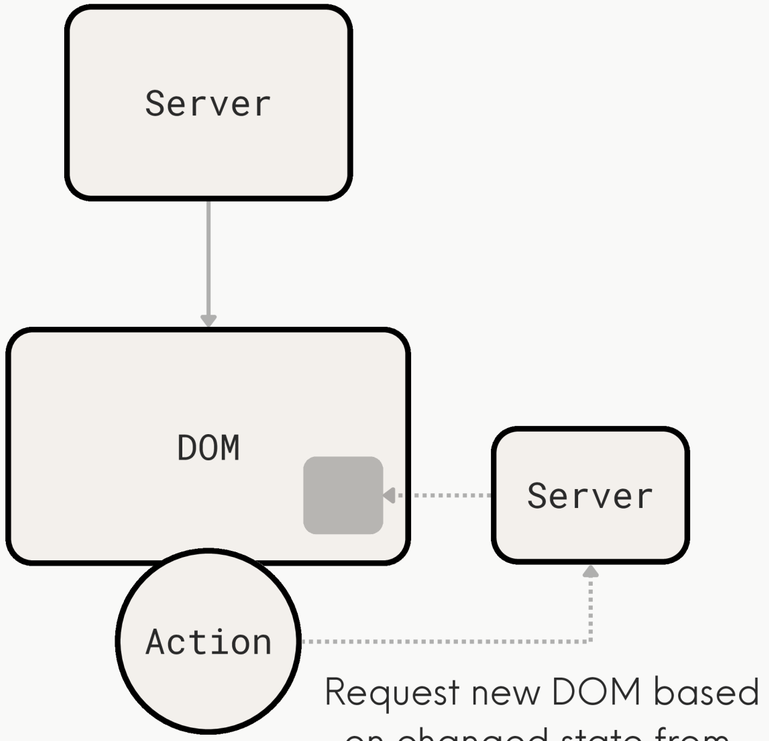
```
<button id="increase">👍</button>
<button id="decrease">👎</button>
<div id="content" data-count="0">
  <p>Counter: 0</p>
  <p>Parity: even</p>
</div>

function update (val) {
  $content.attr("data-count", val);
  $.ajax(serverUrl, { val }).done((html) => {
    $content.html(html);
  });
}

$increase.click(() => update($content.attr("data-count") - 1));
$decrease.click(() => update($content.attr("data-count") + 1));
```

2006 – jQuery and AJAX

```
<button id="increase">👍</button>
<button id="decrease">👎</button>
<div id="content" data-count="0">
  <p>Counter: 0</p>
  <p>Parity: even</p>
</div>

function update (val) {
  $content.attr("data-count", val);
  $.ajax(serverUrl, { val }).done((html) => {
    $content.html(html);
  });
}

$increase.click(() => update($content.attr("data-count") - 1));
$decrease.click(() => update($content.attr("data-count") + 1));
```

Improved

“PULL”



Request new DOM based on changed state from the server, **specifically for certain part of the UI**

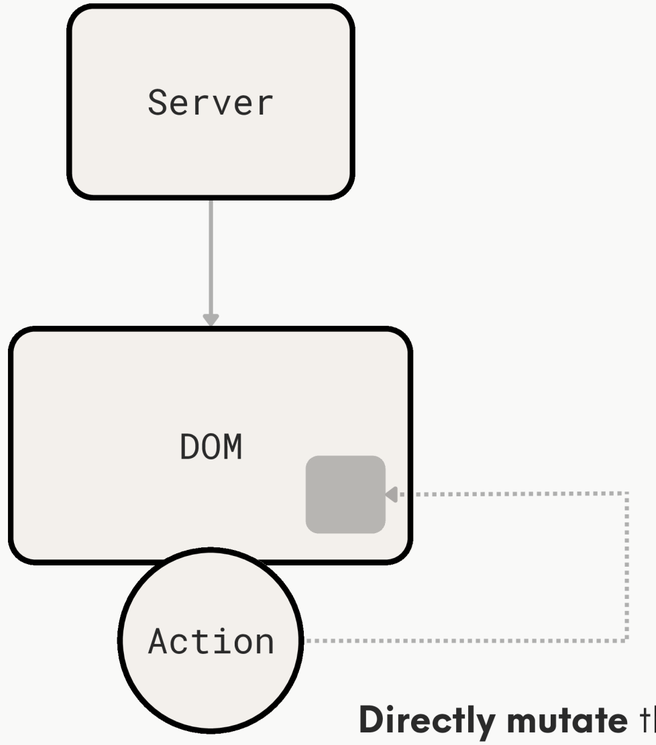
2006 – jQuery

```
<button id="increase">👍</button>
<button id="decrease">👎</button>
<div id="content" data-count="0">
  <p>Counter: <span id="counter"></span></p>
  <p>Parity: <span id="parity"></span></p>
</div>

function update (val) {
  const parity = val % 2 ? "odd" : "even";
  $content.attr("data-count", val);
  $counter.text(val);
  $parity.text(parity);
}

update($content.attr("data-count"));
$increase.click(() => update($content.attr("data-count") - 1));
$decrease.click(() => update($content.attr("data-count") + 1));
```

2006 – jQuery

```
<button id="increase">👍</button>
<button id="decrease">👎</button>
<div id="content" data-count="0">
  <p>Counter: <span id="counter"></span></p>
  <p>Parity: <span id="parity"></span></p>
</div>

function update (val) {
  const parity = val % 2 ? "odd" : "even";
  $content.attr("data-count", val);
  $counter.text(val);
  $parity.text(parity);
}

update($content.attr("data-count"));
$increase.click(() => update($content.attr("data-count") - 1));
$decrease.click(() => update($content.attr("data-count") + 1));
```

“PUSH”

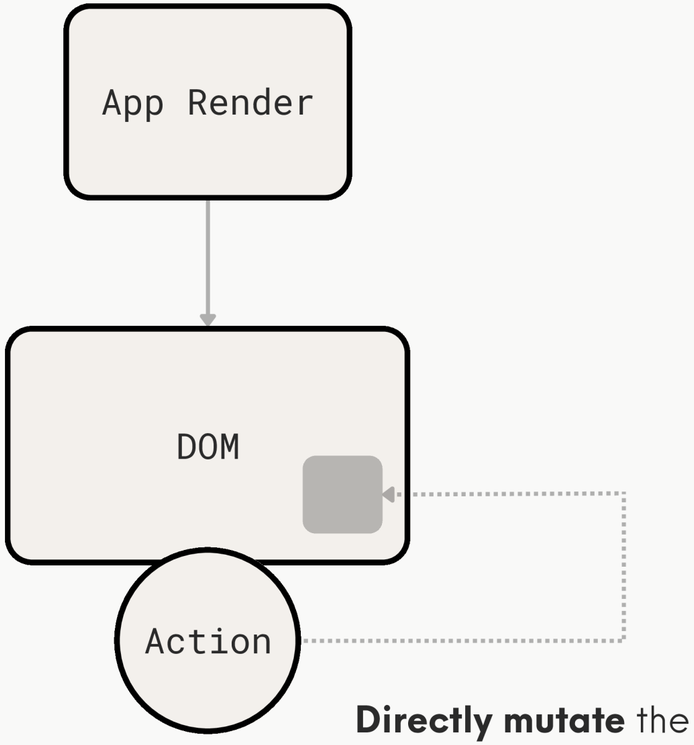


Directly mutate the DOM specifically where needed

2010 – Knockout

```
<button data-bind="click: increase">👍</button>
<button data-bind="click: decrease">👎</button>
<p>Counter: <span data-bind="text: counter"></span></p>
<p>Parity: <span data-bind="text: parity"></span></p>

function ViewModel () {
  this.counter = ko.observable(0);
  this.parity = ko.computed(() =>
    this.counter() % 2 ? "odd" : "even"
  );
  this.decrease = () => this.counter(this.counter() - 1);
  this.increase = () => this.counter(this.counter() + 1);
}
ko.applyBindings(new ViewModel());
```

“PUSH”

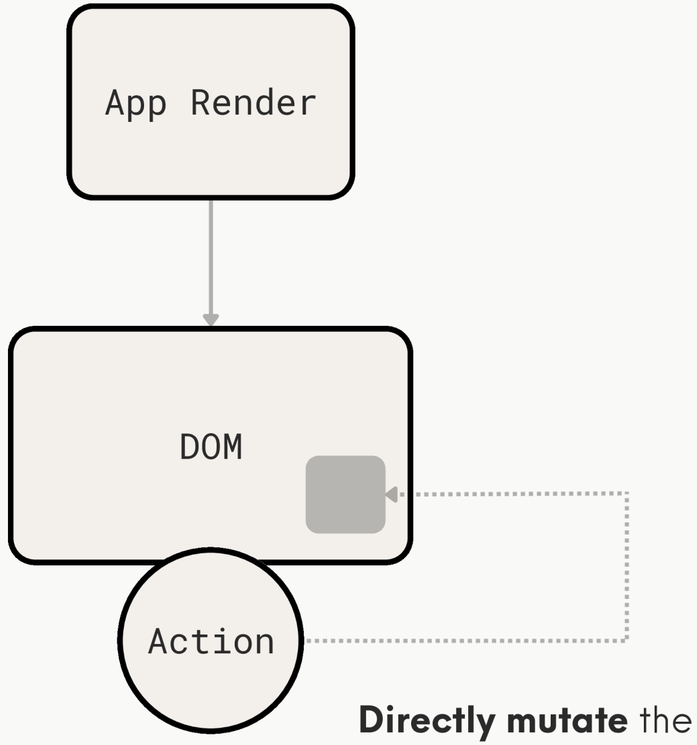


Directly mutate the DOM specifically where needed

2014 – Vue

```
<button v-on:click="count++">👍</button>
<button v-on:click="count--">👎</button>
<p>Counter: {{ counter }}</p>
<p>Parity: {{ parity }}</p>

var vm = new Vue({
  data: {
    counter: 0
  },
  computed: {
    parity: function () {
      return this.counter % 2 ? "odd" : "even";
    }
  }
});
```

“PUSH”

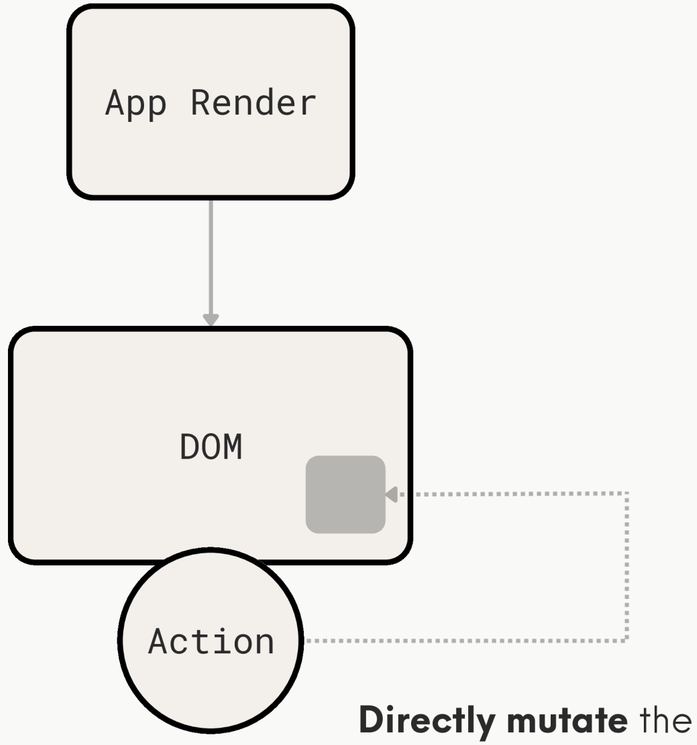


Directly mutate the DOM specifically where needed

2016 – Svelte

```svelte
<button on:click={increase}>👍</button>
<button on:click={decrease}>👎</button>
<p>Counter: {counter}</p>
<p>Parity: {parity}</p>

let counter = 0;
const decrease = () => {
  counter -= 1;
};
const increase = () => {
  counter += 1;
};

const parity = counter % 2 ? "odd" : "even";
```

“PUSH”



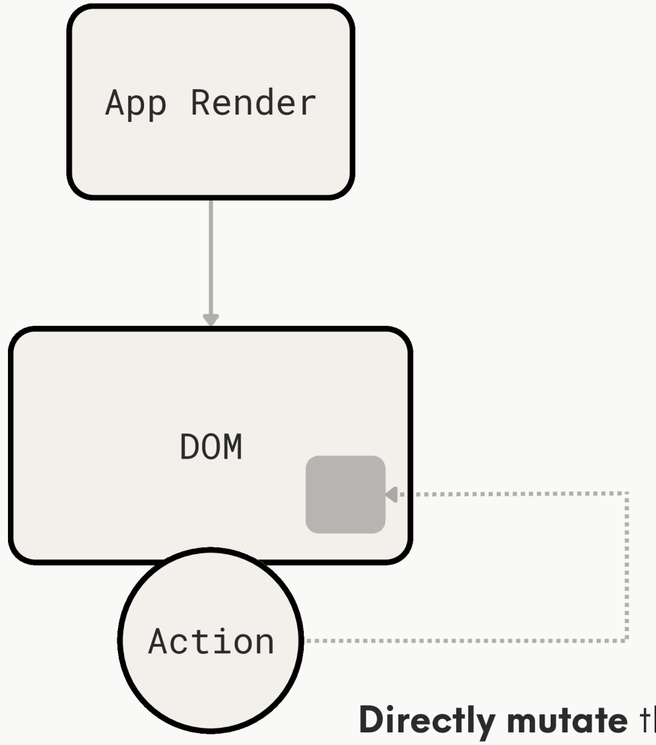
Directly mutate the DOM specifically where needed

2016 – Svelte

*since runes were introduced in 2019

```
<button on:click={increase}>👍</button>
<button on:click={decrease}>👎</button>
<p>Counter: {counter}</p>
<p>Parity: {parity}</p>

let counter = $state(0);
const decrease = () => {
  counter -= 1;
};
const increase = () => {
  counter += 1;
};

const parity = $derived(counter % 2 ? "odd" : "even");
```

“PUSH”

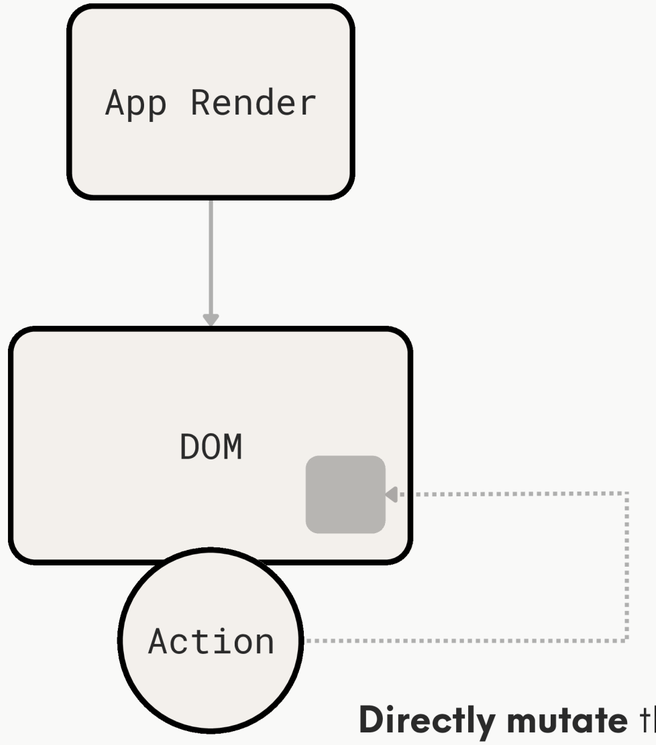


Directly mutate the DOM specifically where needed

2019 – Solid

```
function App () {
  const [counter, setCounter] = createSignal(0);
  const decrease = () => setCounter(counter() - 1)
  const increase = () => setCounter(counter() + 1)

  const parity = createComputed(() =>
    counter() % 2 ? "odd" : "even");

  return (
    <>
      <button onClick={increase}>👍</button>
      <button onClick={decrease}>👎</button>
      <p>Counter: {counter}</p>
      <p>Parity: {parity}</p>
    </>
  );
}
```

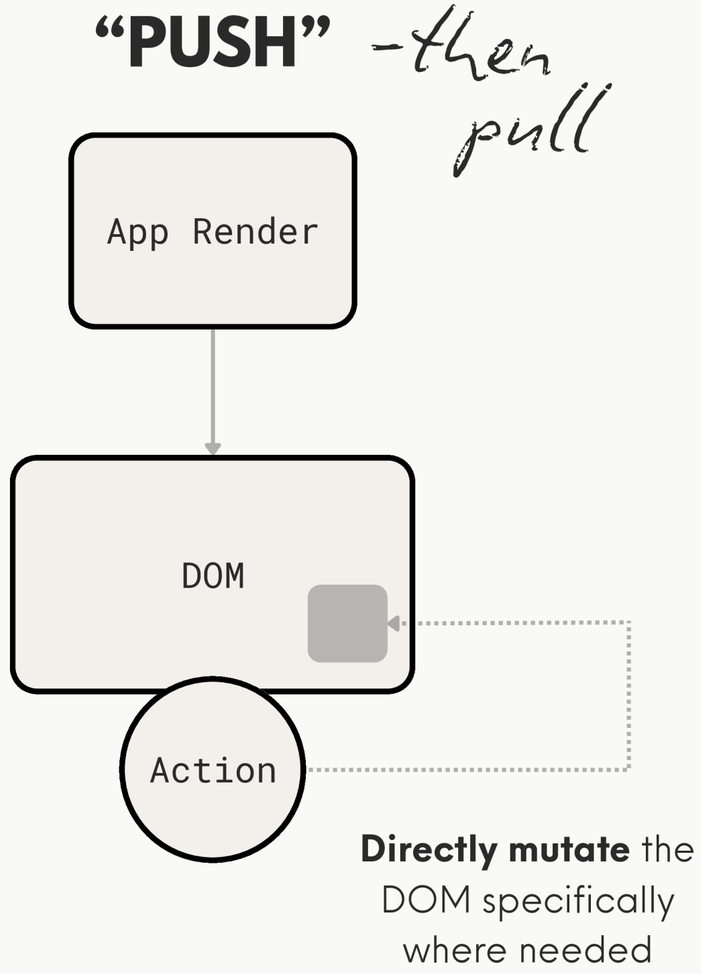
“PUSH”



Directly mutate the DOM specifically where needed

2019 – Solid

```
function App () {
  const [counter, setCounter] = createSignal(0);
  const decrease = () => setCounter(counter() - 1)
  const increase = () => setCounter(counter() + 1)

  const parity = createComputed(() =>
    counter() % 2 ? "odd" : "even");

  return (
    <>
      <button onClick={increase}>👍</button>
      <button onClick={decrease}>👎</button>
      <p>Counter: {counter}</p>
      <p>Parity: {parity}</p>
    </>
  );
}
```

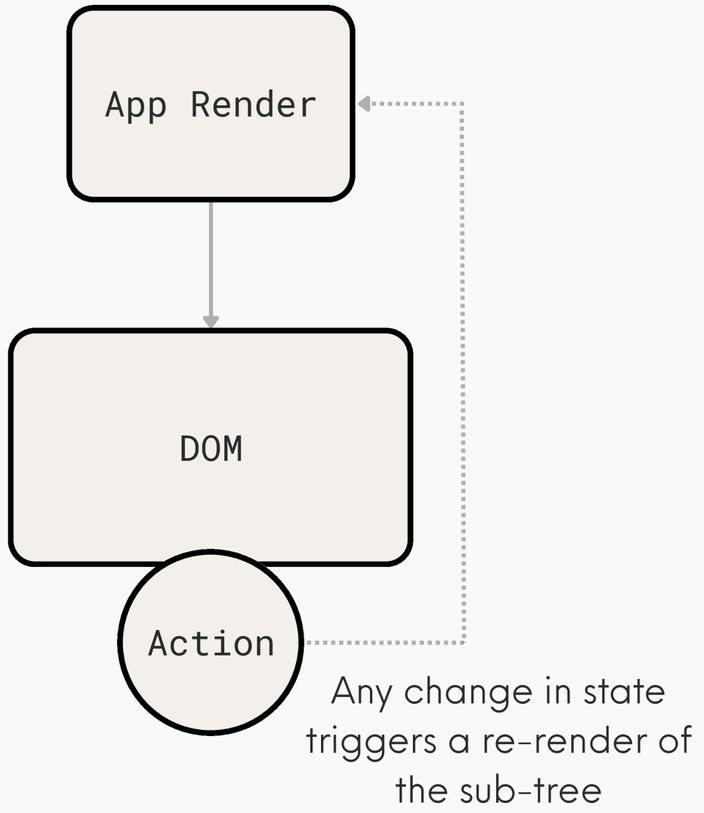


WHAT ABOUT REACT?

2013 – React

```
function App () {
  const [counter, setCounter] = useState(0);
  const decrease = () => setCounter(counter - 1)
  const increase = () => setCounter(counter + 1)

  const parity = useMemo(() => counter % 2 ? "odd" : "even");

  return (
    <>
      <button onClick={decrement}>👎</button>
      <button onClick={increment}>👍</button>
      <p>Counter: {counter}</p>
      <p>Parity: {parity}</p>
    </>
  );
}
```

“PULL”



Andrew Clark
@acdlite

We might add a signals-like primitive to React but I don't think it's a great way to write UI code. It's great for performance. But I prefer React's model where you pretend the whole thing is recreated every time. Our plan is to use a compiler to achieve comparable performance.

12:34 AM · February 18, 2023

HTTPS://X.COM/ACDLITE/STATUS/1626590880126889984

John Lindquist
@johnlindquist

@dan_abramov @reactjs "React" is a terrible name for @reactjs.

2:54 AM · March 24, 2019

HTTPS://X.COM/JOHNLINDQUIST/STATUS/1109498707475488768

HOW DOES IT WORK?

HOW DOES IT WORK?

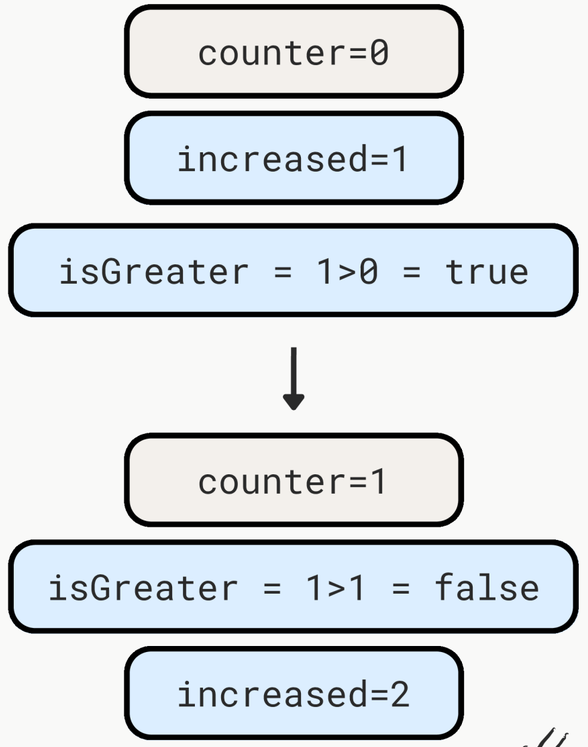
Let's look at some code

THERE ARE STILL CHALLENGES

```
const counter = signal(0);
const increased = computed(() =>
  counter.get() + 1);
const isGreater = computed(() =>
  increased.get() > counter.get());

counter.set(1);

// Depending on the order the computed
// values will bere-calculated, you can
// end up in a glitch state where
// isGreater.get() equals false 🙈
```

HTTPS://EN.WIKIPEDIA.ORG/WIKI/REACTIVE_PROGRAMMING#GLITCHES

```
const counter = signal(0);
const increased = computed(() =>
  counter.get() + 1);
const isGreater = computed(() =>
  increased.get() > counter.get());

counter.set(1);

// Depending on the order the computed
// values will bere-calculated, you can
// end up in a glitch state where
// isGreater.get() equals false 🙈
```



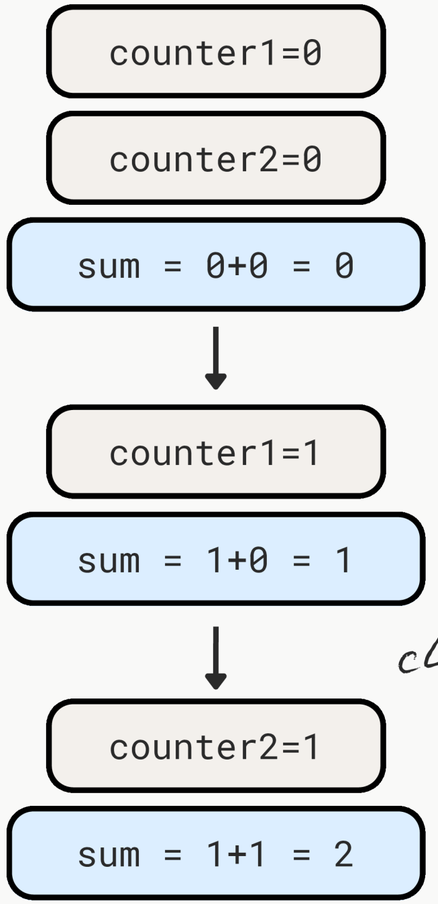
the order of re-calculating matters

HTTPS://EN.WIKIPEDIA.ORG/WIKI/REACTIVE_PROGRAMMING#GLITCHES

```
const counter1 = signal(0);
const counter2 = signal(0);
const sum = computed(() =>
  counter1.get() + counter2.get());

const increaseBoth = (amount) => {
  counter1.set(counter1.get() + amount);
  counter2.set(counter2.get() + amount);
}

effect(() => console.log(`Sum: ${sum.get()}`);

increaseBoth(2);
// Sum: 2
// Sum: 4
```

HTTPS://EN.WIKIPEDIA.ORG/WIKI/REACTIVE_PROGRAMMING#GLITCHES

```
const counter1 = signal(0);
const counter2 = signal(0);
const sum = computed(() =>
  counter1.get() + counter2.get());

const increaseBoth = (amount) => {
  counter1.set(counter1.get() + amount);
  counter2.set(counter2.get() + amount);
}

effect(() => console.log(`Sum: ${sum.get()}`);

increaseBoth(2);
// Sum: 2
// Sum: 4
```



un-batched changes lead to intermediate glitches

HTTPS://EN.WIKIPEDIA.ORG/WIKI/REACTIVE_PROGRAMMING#GLITCHES

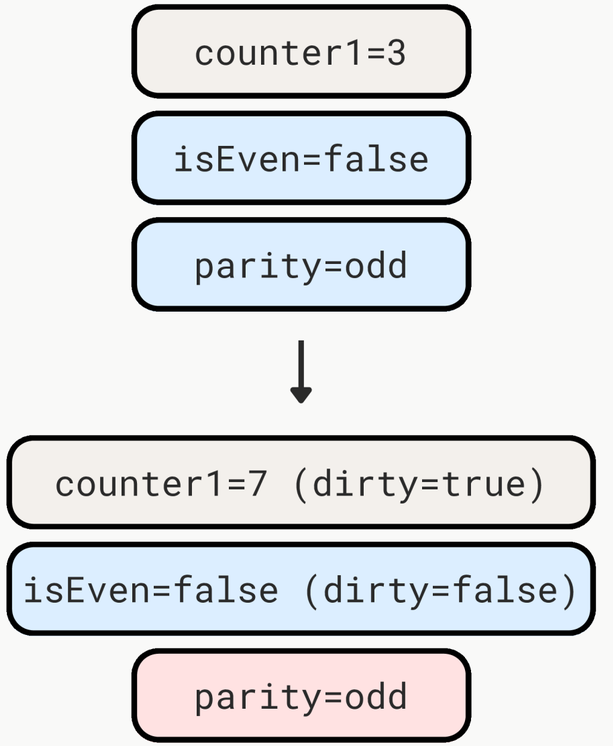
```
const counter = signal(0);
const isEven = computed(() =>
  counter.get() % 2 === 0);
const parity = computed(() =>
  isEven.get() ? "even" : "odd");

counter.set(3);
// isEven = false
// parity = "odd"
```

HTTPS://EN.WIKIPEDIA.ORG/WIKI/REACTIVE_PROGRAMMING#GLITCHES

```
const counter = signal(0);
const isEven = computed(() =>
  counter.get() % 2 === 0);
const parity = computed(() =>
  isEven.get() ? "even" : "odd");

counter.set(3);
// isEven = false
// parity = "odd"

counter.set(7);
// isEven = false
// parity = "odd" *
```

HTTPS://EN.WIKIPEDIA.ORG/WIKI/REACTIVE_PROGRAMMING#GLITCHES

```
const counter = signal(0);
const isEven = computed(() =>
  counter.get() % 2 === 0);
const parity = computed(() =>
  isEven.get() ? "even" : "odd");

counter.set(3);
// isEven = false
// parity = "odd"

counter.set(7);
// isEven = false
// parity = "odd" *
```



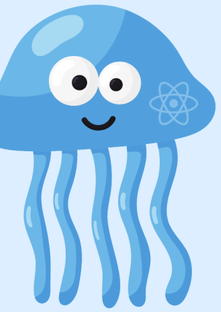
re-calculation can be skipped if none of the dependencies are dirty

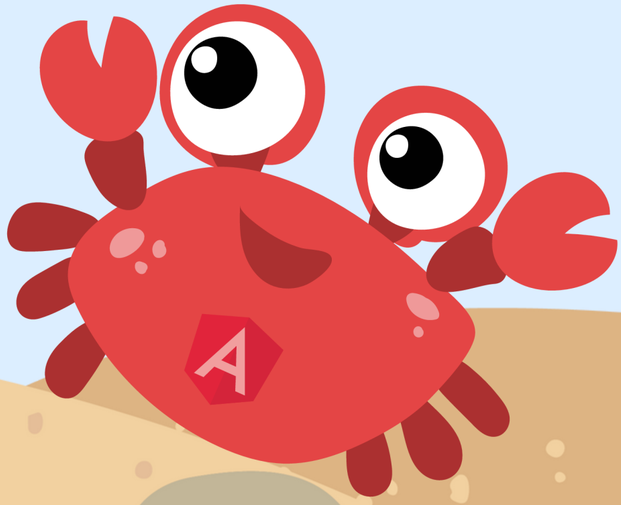
HTTPS://EN.WIKIPEDIA.ORG/WIKI/REACTIVE_PROGRAMMING#GLITCHES

THE TC39 PROPOSAL

Frontend frameworks have been evolving and converging towards the same solution for the same problem

*we're all becoming crabs





```
const counter = new Signal.State(0);
const parity = new Signal.Computed(
  () => counter.get() % 2 ? "odd" : "even"
);

const decrease = () => counter.set(counter.get() - 1);
const increase = () => counter.set(counter.get() + 1);

// A library or framework defines effects using low-level
// primitives
declare function effect(cb: () => void): (() => void);

effect(() => window.counter.innerHTML = counter.get());
effect(() => window.parity.innerHTML = parity.get());
```

HTTPS://GITHUB.COM/TC39/PROPOSAL-SIGNALS

SOME GOOD RESOURCES

https://www.pzuraq.com/blog/what-is-reactivity

https://milomg.dev/2022-12-01/reactivity

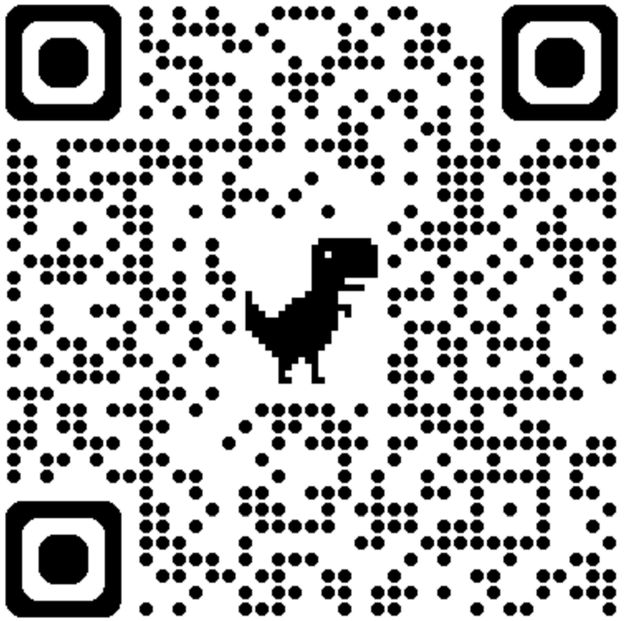
https://dev.to/ryansolid/a-hands-on-introduction-to-fine-grained-reactivity-3ndf

https://dev.to/this-is-learning/the-evolution-of-signals-in-javascript-8ob

https://github.com/milomg/reactively/blob/main/Reactive-algorithms.md#reactively

https://www.youtube.com/watch?v=nYkdrAPrdcw – Rethinking Web App Development at Facebook

https://www.youtube.com/watch?v=uA0ClYdC0xA – Daniel Ehrenberg – Standardizing Signals in TC39



Link to the slides:

https://www.julianburr.de/web-directions-code-2025-slides.pdf

https://www.linkedin.com/in/julianburr/

https://bsky.app/profile/julianburr.de

https://github.com/julianburr