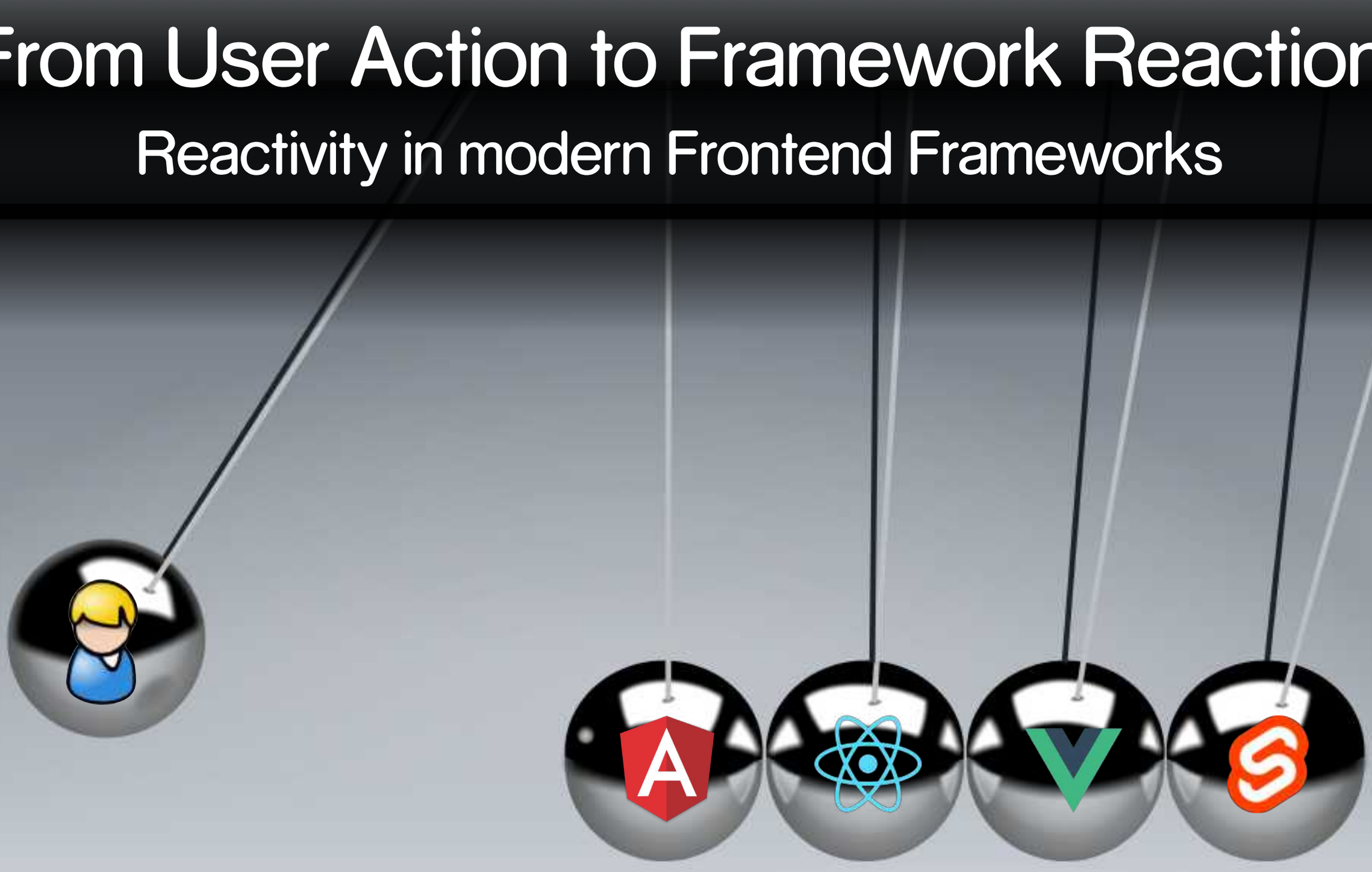
# From User Action to Framework Reaction

## Reactivity in modern Frontend Frameworks



# Jonas Bandi

@jbandi

- Freelancer, in den letzten 8 Jahren tätig in Projekten im Spannungsfeld zwischen modernen Webentwicklung und traditionellen Geschäftsanwendungen.
- Dozent an der Berner Fachhochschule seit 2007
- In-House Kurse & Beratungen zu Web-Technologien im Enterprise: UBS, Postfinance, Mobiliar, AXA, BIT, SBB, Elca, Adnovum, BSI ...


JavaScript / Angular / React / Vue / Vaadin
Schulung / Beratung / Coaching / Reviews

jonas.bandi@ivorycode.com

# Reactivity ?

"There are as many definitions of reactive programming as there are reactive programmers."

# Reactive Programming?

In computing, reactive programming is a declarative programming paradigm concerned with data streams and the propagation of change.

- Wikipedia

*reactive programming* is a paradigm in which declarative code is issued to construct asynchronous processing pipelines.

- Defining the term "reactive"
https://developer.ibm.com/articles/defining-the-term-reactive/

Reactive programming is programming with asynchronous data streams.

- The introduction to Reactive Programming you've been missing
https://gist.github.com/staltz/868e7e9bc2a7b8c1f754

```
click$
  .pipe(scan(count => count + 1, 0))
  .subscribe(count => console.log(`Clicked ${count} times`));
```

"The essence of functional reactive programming is to specify the *dynamic behavior* of a value completely at the *time of declaration*"

- Heinrich Apfelmus

https://apfelmus.nfshost.com/blog/2011/03/28-essence-frp.html

AutoSave OFF Book1

Home Insert Draw Page Layout Share Comments

Clipboard Font Alignment Number Conditional Formatting Format as Table Cell Styles

B8 =SUM(B2:B7)

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | | Amount | | | |
| 2 | | 1 | | | |
| 3 | | 2 | | | |
| 4 | | 3 | | | |
| 5 | | 4 | | | |
| 6 | | | | | |
| 7 | | | | | |
| 8 | Total | 10 | | | |
| 9 | | | | | |

Sheet1 +

Ready 149%

# Agenda

Intro

Reactivity - What are we talking about?

Exploration

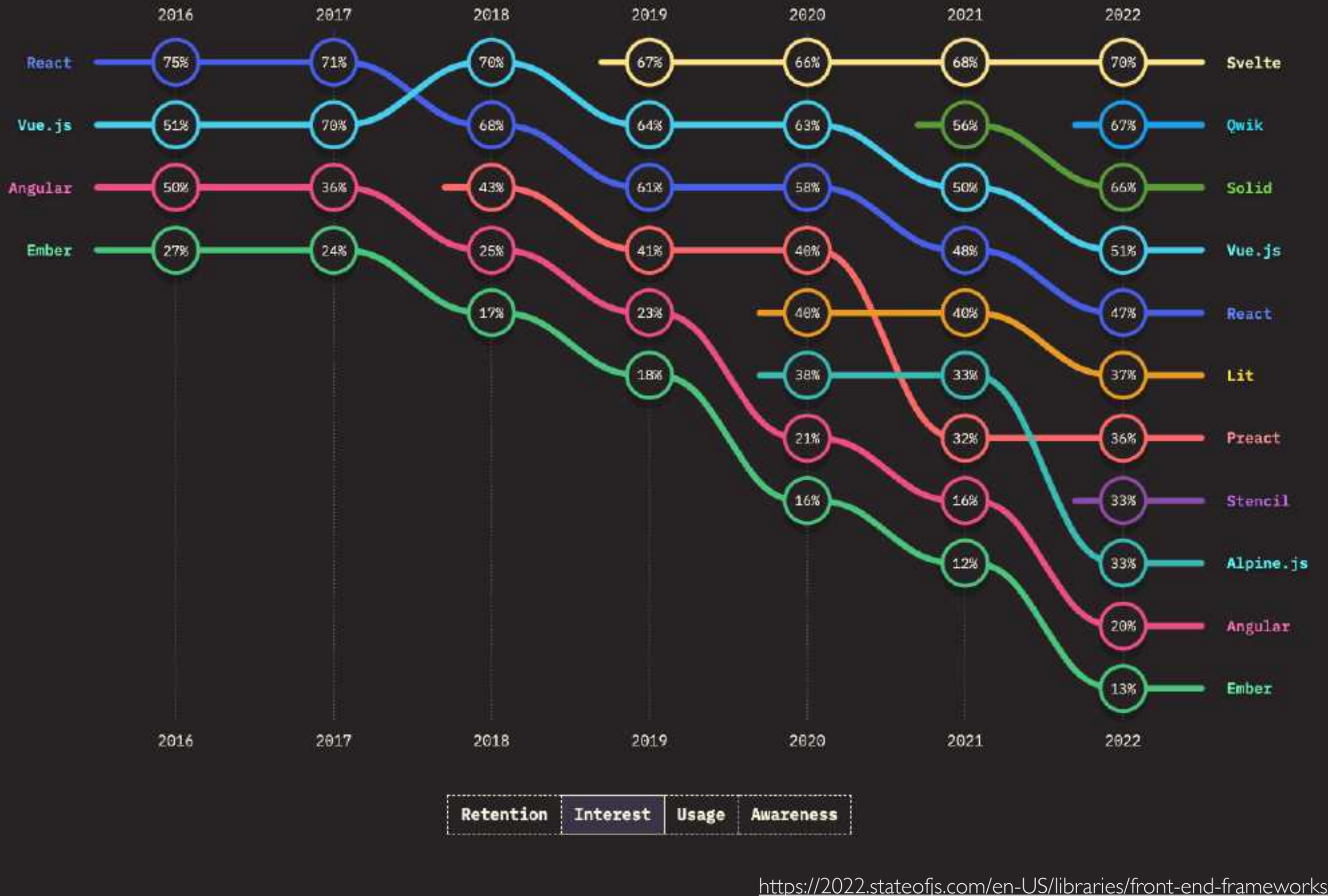
"Out of the Box"-Reactivity of

- Code Example
- How does it work?
- Implications

Takeaway

A glimpse into each framework.
A "feeling" how the framework works.

# State of JavaScript Survey 2022:

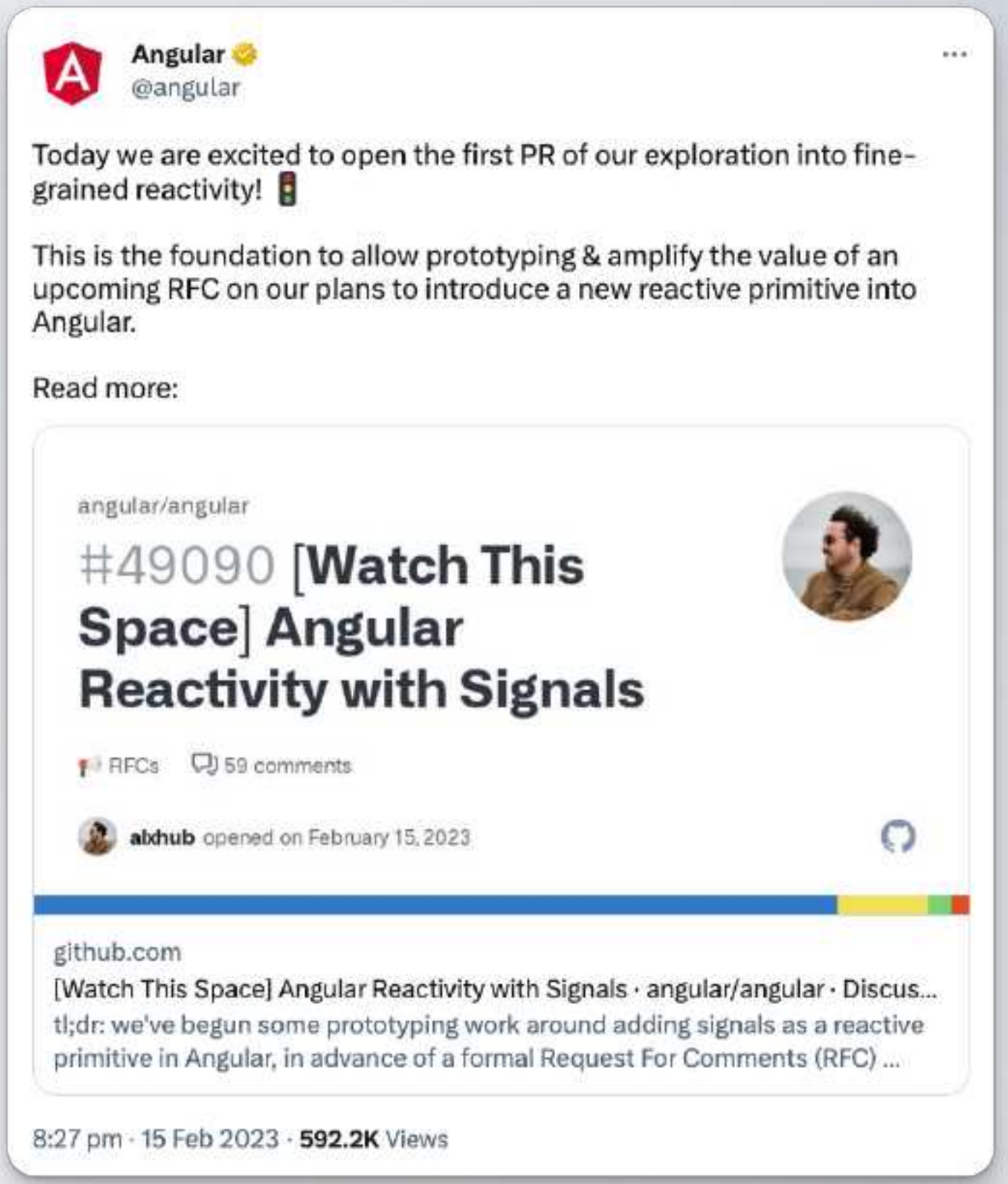



https://2022.stateofjs.com/en-US/libraries/front-end-frameworks/

# In the News ...

New reactive primitives ...

```
import {signal, computed, effect}
                    from '@angular/core';
```




https://github.com/angular/angular/discussions/49090
https://dev.to/this-is-angular/i-changed-my-mind-angular-needs-a-reactive-primitive-n2g

In the Beginning there was Darkness ...


... then the DOM was created.

# … and we manipulated the DOM …


```
$(".menu-item")
    .removeClass("active")
    .addClass("inactive ")
    .css("padding-left", "0px")
    .find(".trigger")
    .click(function(ev) {
        // spaghetti carbonara?
    })
    .each(function () {
        // spaghetti napoli?
    });
```


# … the Dark Ages of DOM …


# … a new hope …

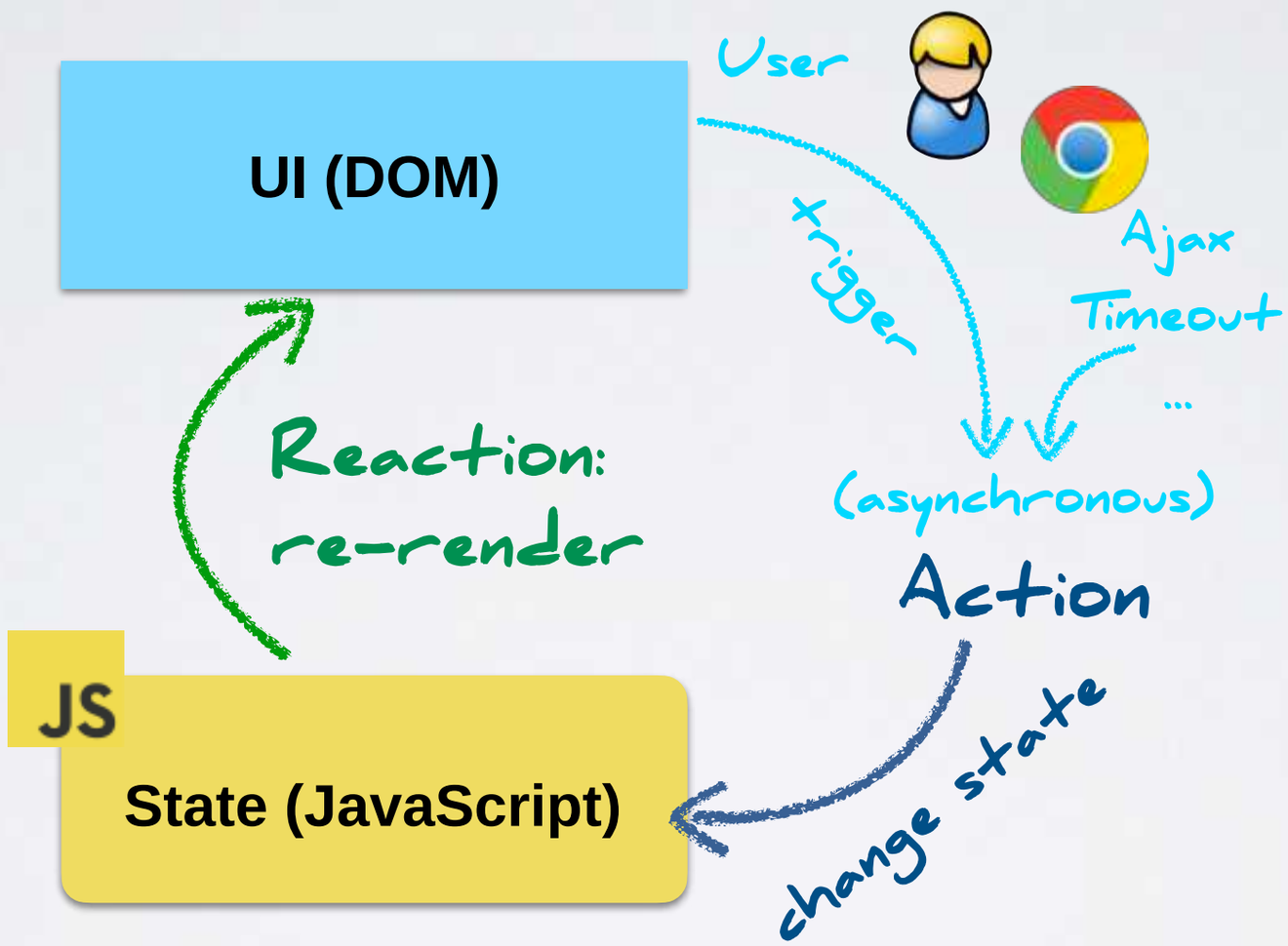
# Model View Controller

# Thou shalt not manipulate the DOM!


# the DOM *is* updated

# State is Managed in JavaScript

The UI renders the state and "emits" events.




Reactivity in a SPA: The application reacts on state changes and updates the UI.

Evan You - Reactivity in Frontend JavaScript Frameworks: https://www.youtube.com/watch?v=r4pNEdIt_I4
http://teropa.info/blog/2015/03/02/change-and-its-detection-in-javascript-frameworks.html

# Reactivity: What and Why?

Traditional "DOM-centric" applications

## UI = state

Browsers have "built-in" reactivity: If the DOM is changed, the UI is re-rendered.

Problem: the same state might be displayed at several places in the DOM.

With client-side Single-Page-Applications, the state is represented as JavaScript objects.

## UI = f(state)


The UI that you can see and manipulate on screen is the result of painting a visual representation of data.

This is the *Reactivity* we are investigating:
How do frameworks deal with state changes over time?
*When the state changes then the UI should (automatically) update ...*

**Rich Harris**
@Rich_Harris

The problem all frameworks are solving is *reactivity*. How does the view react to change?

- React: 'we re-render the world'
- Vue: 'we wrap your data in accessors'
- Svelte: 'we provide an imperative set() method that defeats TypeScript'
- Angular: 'zones' (actually idk 🤷‍♂️)

5:01 PM · Nov 3, 2018 · Twitter Web App

https://twitter.com/Rich_Harris/status/1058751085706383362

# Framework Reactivity


# Angular Reactivity

```
@Component({
  selector: 'app-counter',
  template: `
    <div>
      <h2>Display of Counter</h2>
      <h1>{{ state }}</h1>
      <button (click)="increment()">Increment</button>
    </div>
  `,
  styles: [],
})
export class CounterComponent {
  state = 0;

  increment() {
    this.state++;
  }
}
```

# setInterval

*... it's not what you think it is ...*

# Zone.js:

## The "Magic" in Angular Change Detection

Zone.js is a JavaScript library provided by the Angular project that patches many asynchronous browser APIs. Listeners can then be triggered when these APIs are executed.

Patched APIs (examples): `setInterval`, `Promise`, `XMLHttpRequest`, `prompt` and DOM events.

More details: https://github.com/angular/angular/blob/master/packages/zone.js/STANDARD-APIS.md

Angular relies on Zone.js to trigger automatic change detection. Angular is running inside the NgZone (a zone created via Zone.js). When async APIs are executed Angular gets notified when the execution has finished and triggers change detection.

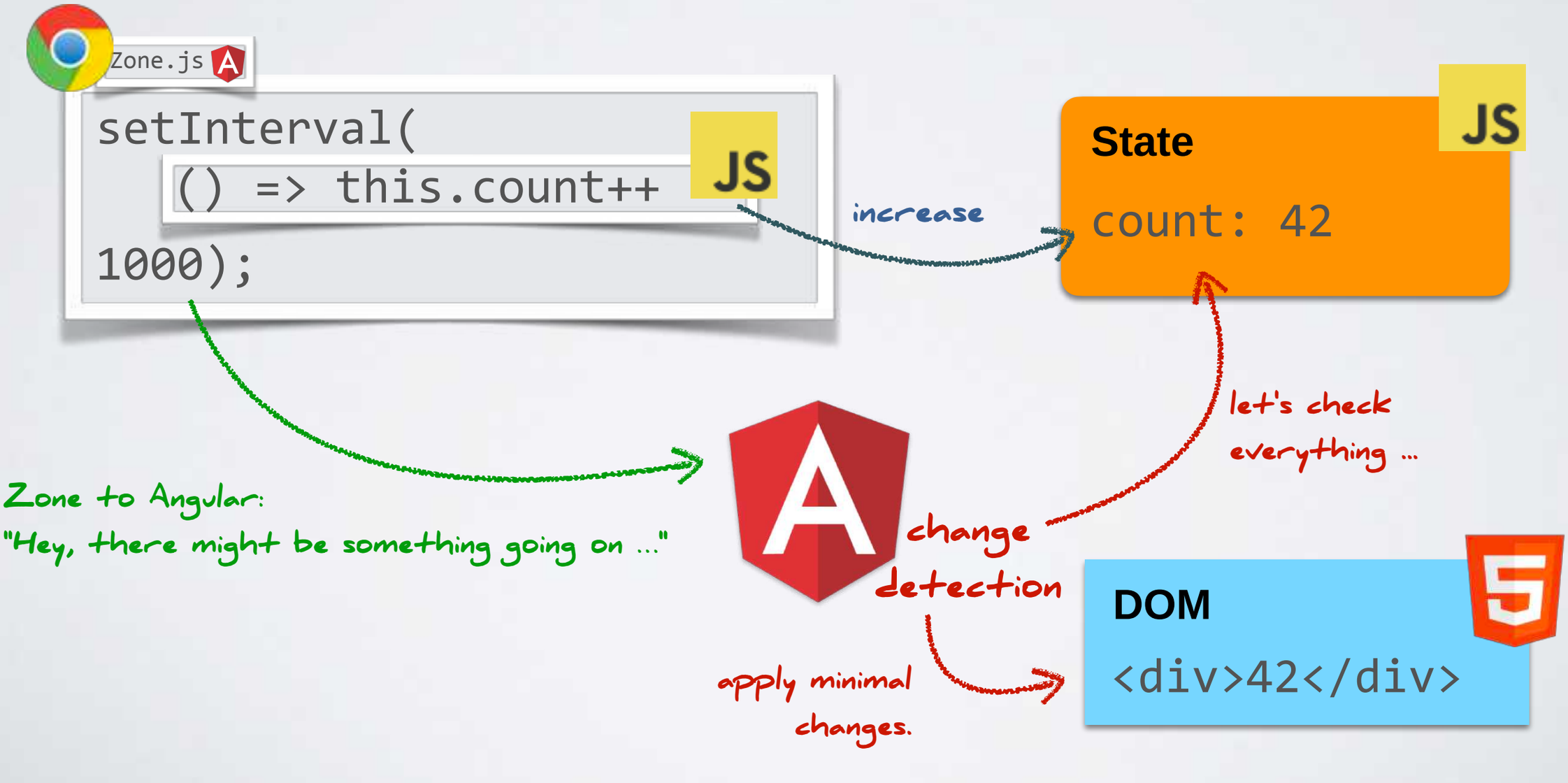
https://github.com/angular/zone.js/
https://medium.com/better-programming/zone-js-for-angular-devs-573d89bbb890

# Default Reactivity in Angular

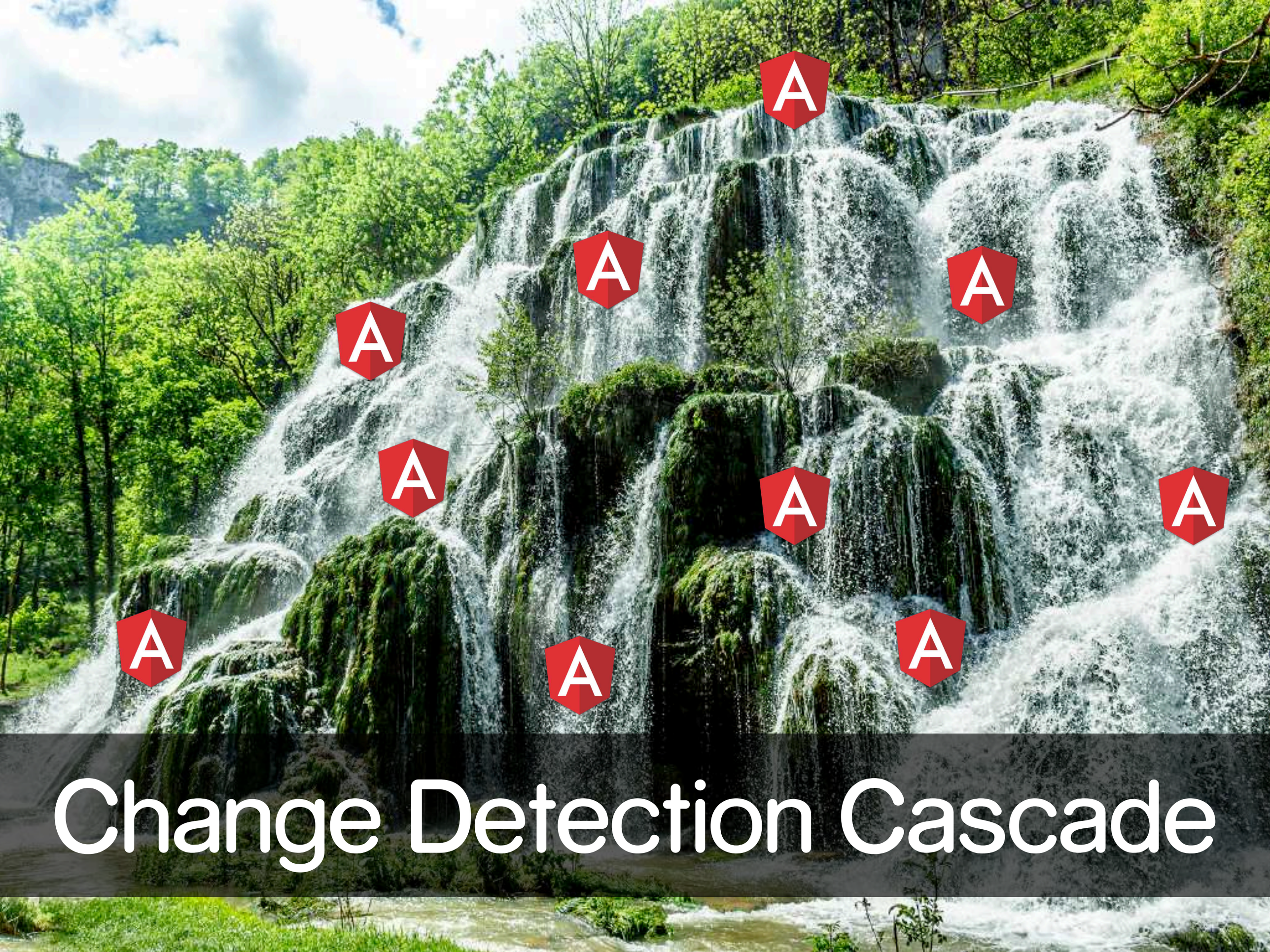
"simulated reactivity"

UI = f(state)

Triggered by Zone.js
"simulated reactivity"



# Mutability



# Change Detection Cascade

# Default Angular Reactivity

"Simulated Reactivity"

## Strength

Transparent Reactivity:
The programmer should be able to use ideomatic JavaScript, the Framework does the rest.

Programming model based on mutations.

## Weakness

Zone.js: Patching the browser is problematic on many levels.

Brute-force approach of default change detection is not optimal in regard to performance.

Change Detection imposes constraints / rules ...

- avoid setter/getters?
- no native async/await
- implicit unidirectional data-flow
  (`ExpressionChangedAfterItHasBeenCheckedError`)

# Default Reactivity in Angular

Zone.js with Default Change Detection:

- is a form of *simulated reactivity:* the framework does not react to changes but to events that might potentially have caused changes
- is a form of *transparent reactivity:* It makes reactivity an *implicit characteristic* of your program.

A common alternative in Angular is to model Reactivity explicitly with RxJS, this is a form of *explicit reactivity*.

The Taxonomy of Reactive Programming: https://vsavkin.com/the-taxonomy-of-reactive-programming-d40e2e23dee4
https://github.com/meteor/docs/blob/version-NEXT/long-form/tracker-manual.md

# Angular Reactivity Variations

`ChangeDetectionStrategy.OnPush`

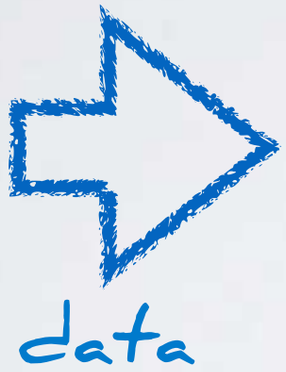
RxJS Observables

Zone-Less

# (Missing?) Reactivity in React

# Function Components

Components are written as plain JavaScript functions.



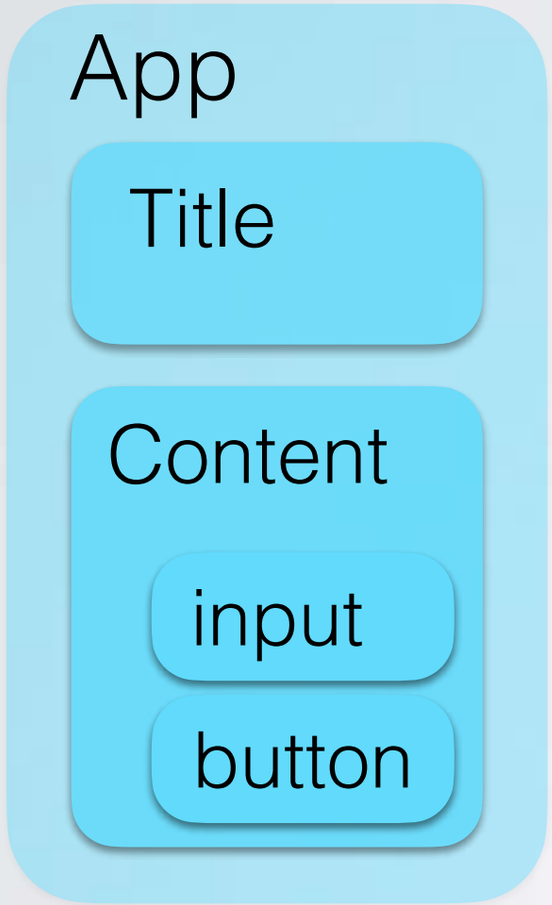

```
function AppComponent(props) {
    return (
        <div>
            <h1>{props.title}</h1>
            <p>{props.message}</p>
        </div>
    );
}
```


The function is called each time the UI is rendered (i.e. with *every* data-change)

A Visual Guide To React Mental Models:
https://obedparla.com/code/a-visual-guide-to-react-mental-models/

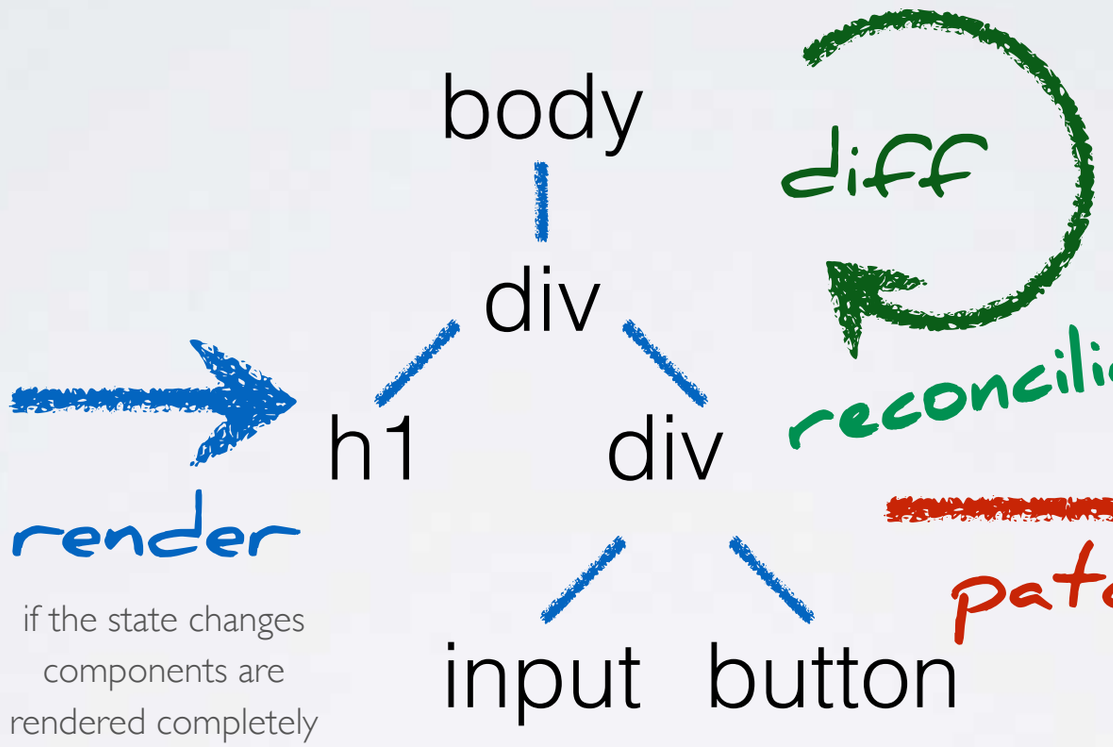
# The Virtual DOM

## Components




## Virtual DOM

In-Memory, implemented in JavaScript




if the state changes components are rendered completely including all their children

## Browser DOM

```
<body>
  <div>
    <h1></h1>
    <div>
      <input/>
      <button>
        </button>
    </div>
  </div>
</body>
```

The Virtual DOM also enables server-side rendering and rendering to iOS/Android UIs.

https://reactjs.org/docs/reconciliation.html

ONE DOES NOT SIMPLY

CHANGE STATE IN REACT

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

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

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

  return (
    <div>
      <h2>Display of Counter.</h2>
      <h1>{count}</h1>
      <button onClick={increment}>Increase</button>
    </div>
  )
}
```

# Reactivity in React

UI = f(state)

triggered by the programmer

JS

```
setInterval(() =>
  setCount(count => count + 1),
1000);
```

**State**

```
count: 42
```

update the state

Programmer to React:
"Please change the state for me ... "

virtual DOM

trigger re-rendering

apply minimal changes.

**DOM**

```
<div>42</div>
```

# Immutability

Render Cascade

# React Reactivity

"Everything is rendered on every state change"

## Strength

Functional Mindset:

- Rendering is a side-effect of state changes.
- Components transform state to ui.

## Weakness

"Render everything" approach is wasteful.

State is managed by React: we have to use the APIs and concepts of React.

Programming model enforces "immutable state management".

https://www.joshwcomeau.com/react/why-react-re-renders/

https://blog.isquaredsoftware.com/2020/05/blogged-answers-a-mostly-complete-guide-to-react-rendering-behavior/

# Reactive State in Vue

```vue
<template>
  <h3>Display of Counter!</h3>
  <h1>{{ state.count }}</h1>
  <button @click="increment">Increase</button>
</template>

<script setup lang="ts">
  import { reactive } from "vue";

  const state = reactive({ count: 0 });

  function increment() {
    state.count++;
  }
</script>
```

# "Naked" Reactive State in Vue:

```
const { reactive, watchEffect } = Vue;

const state = reactive({
  count: 0
});

watchEffect(() => {
  document.body.innerHTML = `count is ${state.count}`
});

setInterval(() => state.count++, 1000);
```

changing state
triggers re-rendering

The example is using the Composition API of Vue 3:
https://vuejs.org/api/reactivity-core.html

# Reactivity in Vue

$$UI_x = f_x(state_x)$$

$$UI_Y = f_Y(state_Y)$$

$$UI_Z = f_Z(state_Z)$$

fine-grained

triggered by reactive state

JS

```
setInterval(
   () => state.count ++,
1000);
```

increase

**State** (Vue proxy)

JS

```
{ count: 42 }
get/set count()
```

trigger fine-grained re-rendering

apply minimal changes.

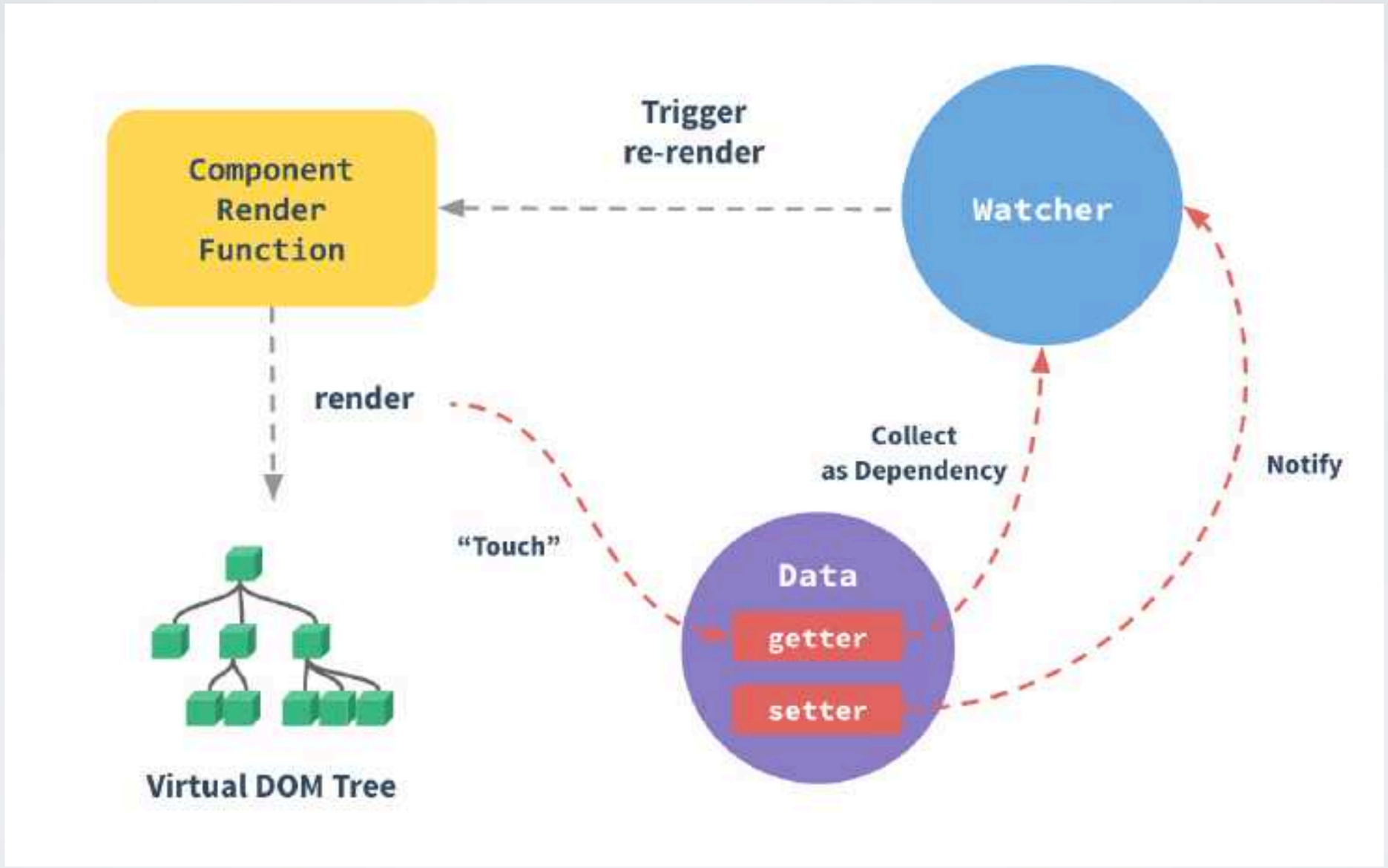
**DOM**

```
<div>42</div>
```

Note: Some statemanagment libraries implement the same concept for other frameworks (MobX, Jotai, Signals ...)

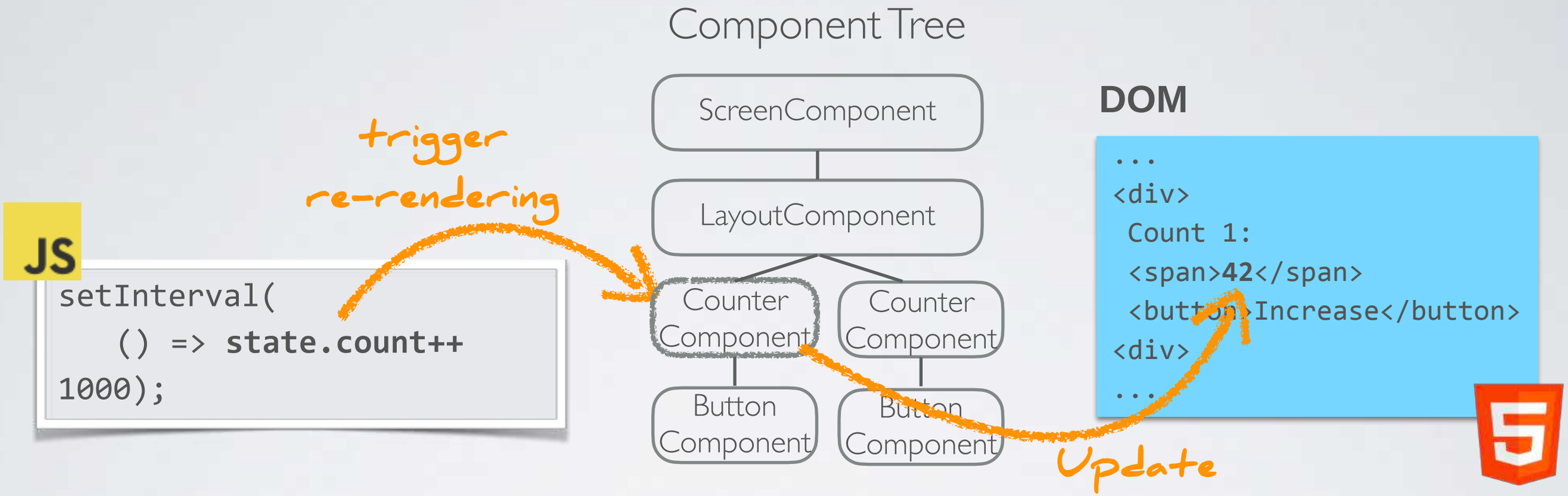
A Hands-on Introduction to Fine-Grained Reactivity: https://dev.to/ryansolid/a-hands-on-introduction-to-fine-grained-reactivity-3ndf

# Change Tracking & Reactive State




https://vuejs.org/v2/guide/reactivity.html

# Fine Grained Reactive State




Fine grained reactive state only triggers re-rendering on components that depend on changed state.

Vue's fine-grained dependency detection just feeds into a less fine-grained Virtual DOM and Component system. (from the documentation of Solid.js - https://www.solidjs.com/guides/comparison#vue )

Vue 3.2 introduced v-memo which allows to implement more fine-grained rendering.
https://vuejs.org/api/built-in-directives.html#v-memo

# Vue Reactivity

"Reactive State"

## Strength

"True Reactivity": The state can be observed.

Fine-Grained Reactivity: only runs the code that need to be run.

Programming model embraces mutability.

## Weakness

State is not "plain" JavaScript, which comes with its own limitations.

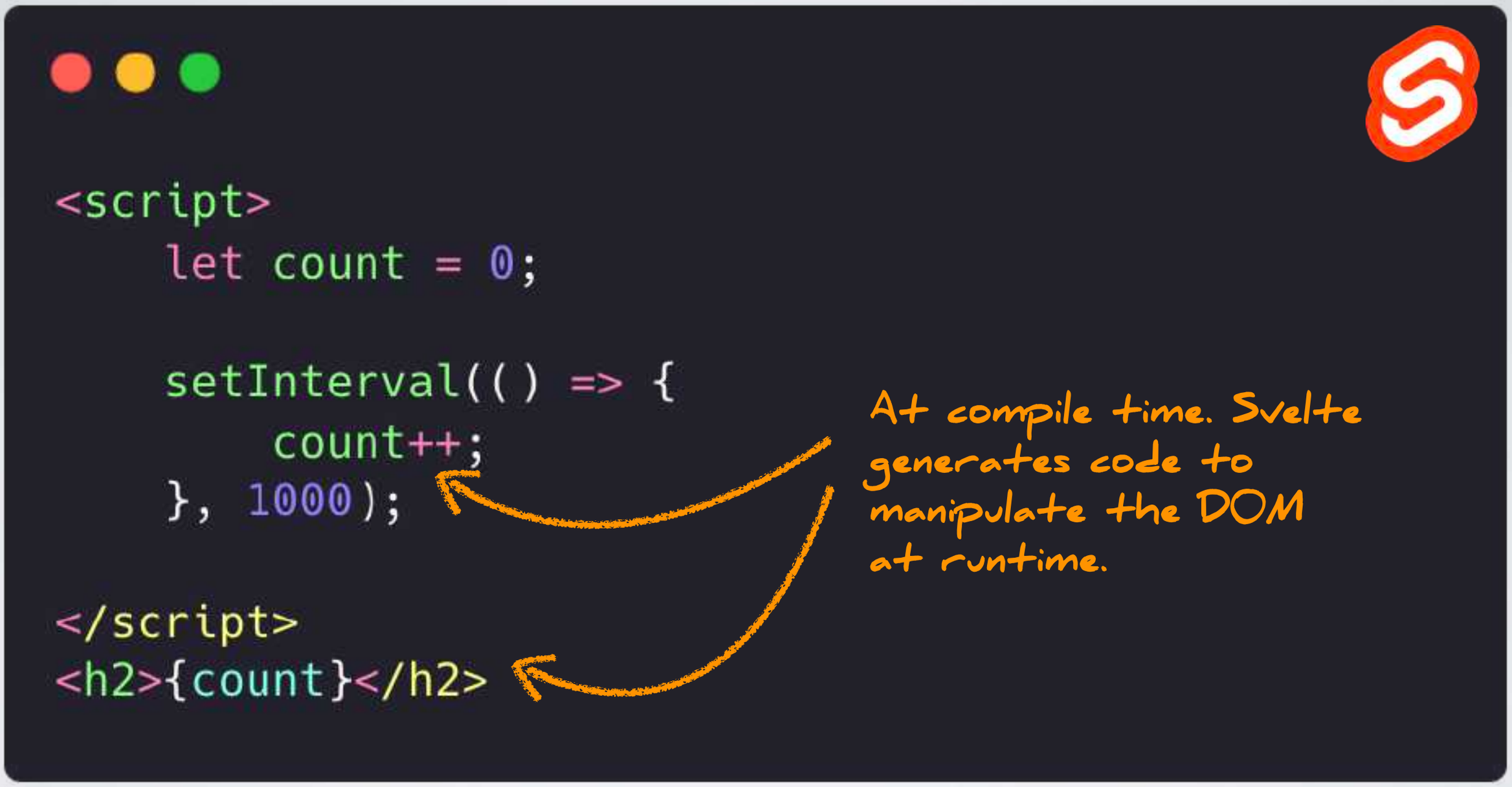
https://dev.to/ryansolid/a-hands-on-introduction-to-fine-grained-reactivity-3ndf

# Svelte

## "Embrace the Compiler!"


Cybernetically enhanced
web apps

aka: Abandon JavaScript?


<script>
let count = 0;
setInterval(() => {
count++;
}, 1000);
</script>
<h2>{count}</h2>
At compile time. Svelte generates code to manipulate the DOM at runtime.

# Reactivity in Svelte

$$UI_x = f_x(state_x)$$

$$UI_Y = f_Y(state_Y)$$

$$UI_Z = f_Z(state_Z)$$

fine-grained

triggered by compile-time generated code

JS

```
setInterval(
    () => state.count++
1000);
```

compile

generate

```
setInterval(
    () => {
      state.count++;
      $invalidate(state);
    }
1000);
```

JS

call

```
function $invalidate(args){
  ...
  updateElement(el, newVal)
}
```

JS

svelte helper functions

apply minimal changes.

**DOM**

```
<div>42</div>
```

build-time

run-time

# Compile-Time-Generated Reactivity

```
<script>
   let name = 'Web';
   let number = 0;

   function update(e) {
      name = 'Svelte';
      number = 42;
   }
</script>

<h1 on:click={update}>
   Hello {name}, {number}!
</h1>
```


helper functions

```
import { SvelteComponent, append, detach, element, init, insert,
   listen, noop, safe_not_equal, set_data, space, text} from "svelte/internal";
```

life-cycle

```
function create_fragment(ctx) {
   let h1;
   let t0;
   let t1;
   let t2;
   let t3;
   let t4;
   let mounted;
   let dispose;

   return {
      c() {
         h1 = element("h1");
         t0 = text("Hello ");
         t1 = text(/*name*/ ctx[0]);
         t2 = space();
         t3 = text(/*number*/ ctx[1]);
         t4 = text("!");
      },
      m(target, anchor) {
         insert(target, h1, anchor);
         append(h1, t0);
         append(h1, t1);
         append(h1, t2);
         append(h1, t3);
         append(h1, t4);
         if (!mounted) {
            dispose = listen(h1, "click", /*update*/ ctx[2]);
            mounted = true;
         }
      },
      p(ctx, [dirty]) {
         if (dirty & /*name*/ 1) set_data(t1, /*name*/ ctx[0]);
         if (dirty & /*number*/ 2) set_data(t3, /*number*/ ctx[1]);
      },
      i: noop,
      o: noop,
      d(detaching) {
         if (detaching) detach(h1);
         mounted = false;
         dispose();
      }
   };
}
```

create

mount

update

unmount

"reactivity"

```
function instance($$self, $$props, $$invalidate) {
   let name = 'Jonas';
   let number = 0;

   function update(e) {
      $$invalidate(0, name = 'Bandi');
      $$invalidate(1, number = 42);
   }

   return [name, number, update];
}
```

instance scope

becomes ctx

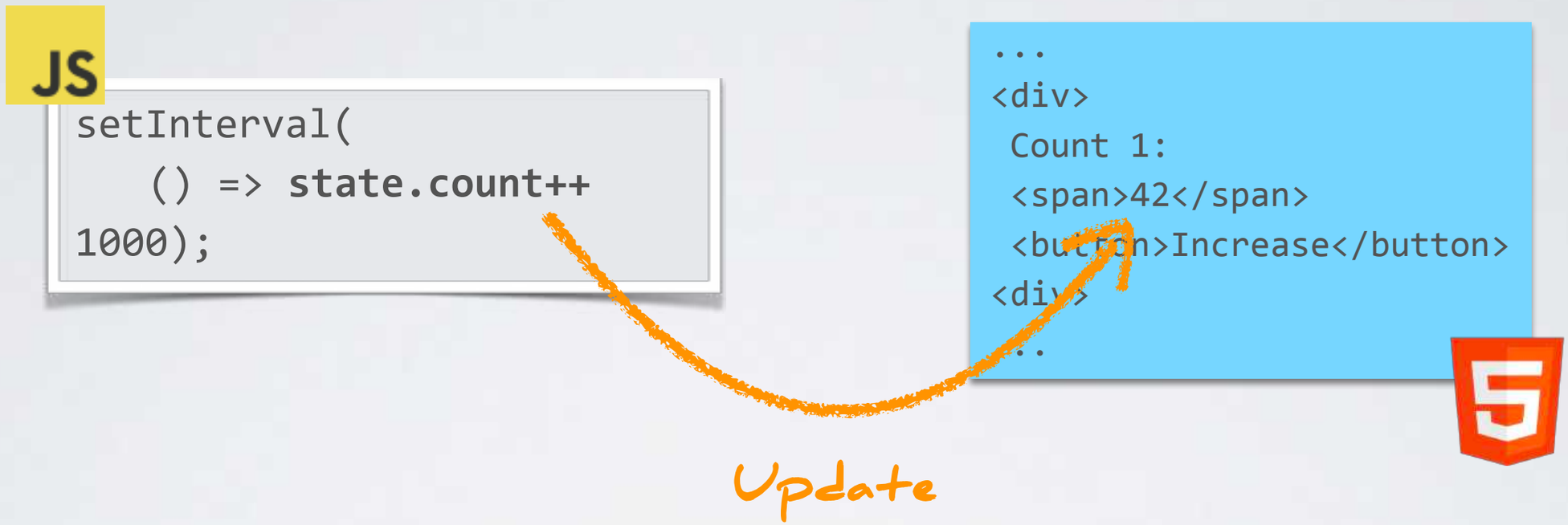
```
class App extends SvelteComponent {
   constructor(options) {
      super();
      init(this, options, instance,
         create_fragment, safe_not_equal, {});
   }
}

export default App;
```

initialization

fine grained reactivity

https://svelte.dev/repl/

# True Fine-Grained Reactivity




Components are not relevant for reactivity at runtime!
Reactive primitives directly update the DOM.

Reasoning: components are useful for code-organization but not needed for optimized UI-updated.

Frameworks featuring true fine-grained reactivity: Solid, Marko, Preact, Qwick, Svelte ...

Components are Pure Overhead: https://dev.to/this-is-learning/components-are-pure-overhead-hpm

# Svelte Reactivity

"Compile-Time-Generated Reactivity"

## Strength

Very compact and intuitive code.

Fine-Grained Reactivity: only runs the code that need to be run.

Claims to be significantly faster than the other mainstream frameworks.

## Weakness

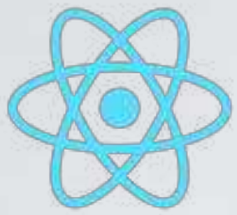
changing ("extending") the semantics of JavaScript

# *On a tangent:*
# All Modern Frontend Frameworks are Compilers!

| | |
|---|---|
| Angular | • TypeScript<br>• Template<br>• (workarounds) |
| React | • JSX<br>• TypeScript |
| Vue | • SFC<br>• Template<br>• TypeScript |

... but Svelte goes one step further by changing the semantics of JavaScript ...

# Fun Fact:



React has plans for a future compiler that is changing the semantics of JavaScript ...
(reducing boilerplate for convenience with auto-memoization)

"React Forget" - A Memoizing Compiler
https://www.youtube.com/watch?v=lGEMwh32soc
https://reactjs.org/blog/2022/06/15/react-labs-what-we-have-been-working-on-june-2022.html

Vue has an optional compiler macro that changes the sematics of JavaScript (reducing boilerplate for convenience with auto-generating value-accessors)

Vue Reactivity Transforms
https://vue-macros.sxzz.moe/features/reactivity-transform.html
https://vuejs.org/guide/extras/reactivity-transform.html
https://github.com/vuejs/rfcs/discussions/369

# In the News ...

**Angular** ✅
@angular

Today we are excited to open the first PR of our exploration into fine-grained reactivity! 🚦

This is the foundation to allow prototyping & amplify the value of an upcoming RFC on our plans to introduce a new reactive primitive into Angular.

Read more:

angular/angular

#49090 [Watch This Space] Angular Reactivity with Signals

RFCs 59 comments

**alxhub** opened on February 15, 2023

github.com
[Watch This Space] Angular Reactivity with Signals · angular/angular · Discus...
tl;dr: we've begun some prototyping work around adding signals as a reactive primitive in Angular, in advance of a formal Request For Comments (RFC) ...

8:27 pm · 15 Feb 2023 · **592.2K** Views

https://github.com/angular/angular/discussions/49090

**Theo - t3.gg** ✅
@t3dotgg

Feels like Angular progressed 5 years in the last 4 months. Presence, sentiment, tech, etc.

Happy for them but also I'm terrified

2:41 AM · Mar 1, 2023 · **198.8K** Views

https://twitter.com/t3dotgg/status/1630744988823982082

Please squint your eyes ... 👀

```vue
<template>
    <h3>Count: {{ count }}</h3>
    <h3>Double: {{ doubleCount }}</h3>
    <button @click="increment">Icrement</button>
</template>
<script setup lang="ts">
  import { computed, effect, ref } from "vue";

  const count = ref(0);

  const doubleCount = computed(() => {
    console.log('Computing double count');
    count.value * 2:
  });


  watchEffect(() => {
    console.log("count changed", count.value);
  });


  function increment() {
    count.value = count.value + 1;
  }

</script>
```

```ts
import { Component, signal, computed, effect, OnInit }
                                        from '@angular/core';
@Component({
  selector: 'app-counter',
  template: `
    <h3>Count {{ count() }}</h3>
    <h3>Double {{ doubleCount() }}</h3>
    <button (click)="increment()">Increment</button>
  `,
})
export class CounterComponent implements OnInit {

  count = signal(0);

  doubleCount = computed(() => {
    console.log('Computing double count');
    return this.count() * 2;
  });

  ngOnInit() {
    effect(() => {
      console.log('Effect: Count is now: ', this.count());
    });
  }

  increment() {
    this.count.set(this.count() + 1);
  }

}
```

using `@angular/core@16.0.0-next.0`

# Signals ... old wine in new skins?

(aka Fine-Grained Reactivity)

| | |
|---|---|
| Vue (Composition API in 2020) | `ref` / `reactive` |
| Solid (2019) | `createSignal` |
| Svelte (v3 in 2019) | compiler generates reactive code similar to signals |
| Preact (signals in 2022) | `signal` |
| Recoil (2020) | `atom` |
| MobX (2016) | `observable` |
| Knockout (2010) | `observable` |

A Hands-on Introduction to Fine-Grained Reactivity: https://dev.to/ryansolid/a-hands-on-introduction-to-fine-grained-reactivity-3ndf

FrontendMasters - Reactivity with SolidJS: https://frontendmasters.com/courses/reactivity-solidjs/

# Signals in Angular

Fine Grained Reactivity ... ?

Getting rid of Zone.js ... ?

Future role of RxJS in the Angular ecosystem?

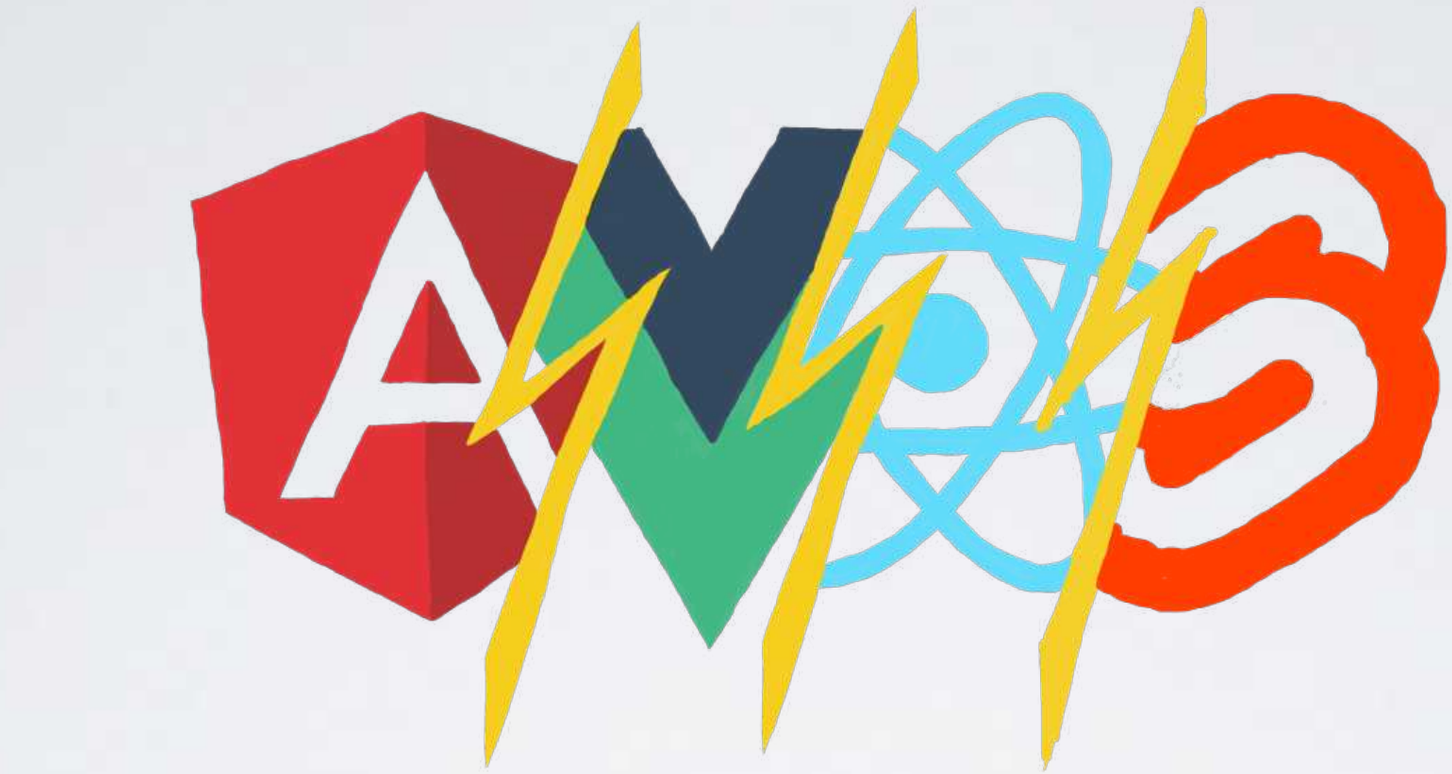
Relevance of `ChangeDetectionStrategy.OnPush`?

# Have Fun with the Framework of your Choice!

@jbandi

Code: https://github.com/jbandi/framework-reactivity-2023

# QUESTIONS?




JavaScript / Angular / React / Vue / Vaadin
Schulung / Beratung / Coaching / Reviews

jonas.bandi@ivorycode.com