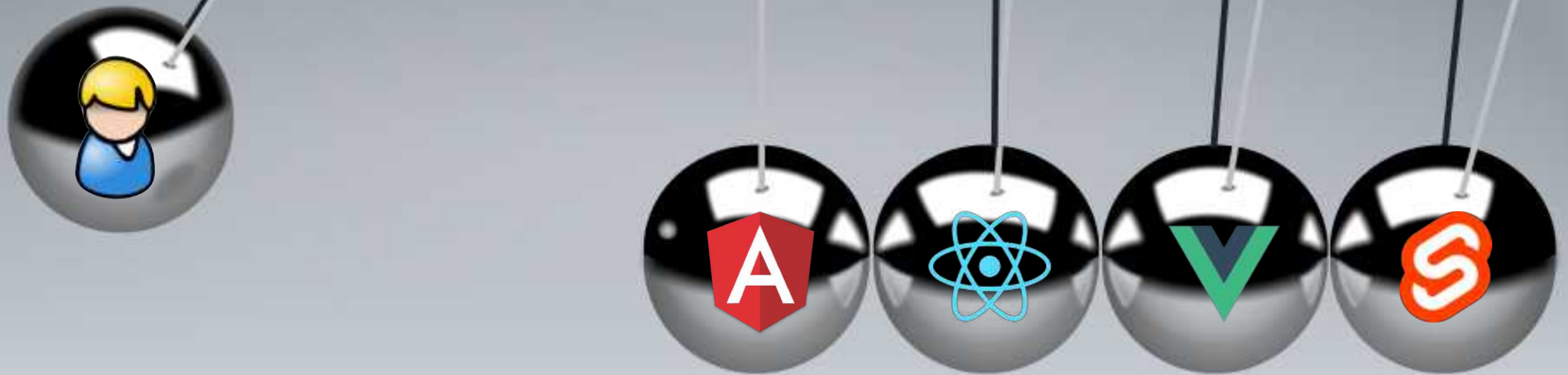


From User Action to Framework Reaction

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.js
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, via Michel Westrate

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 here?

Exploration

"Out of the Box"-Reactivity of

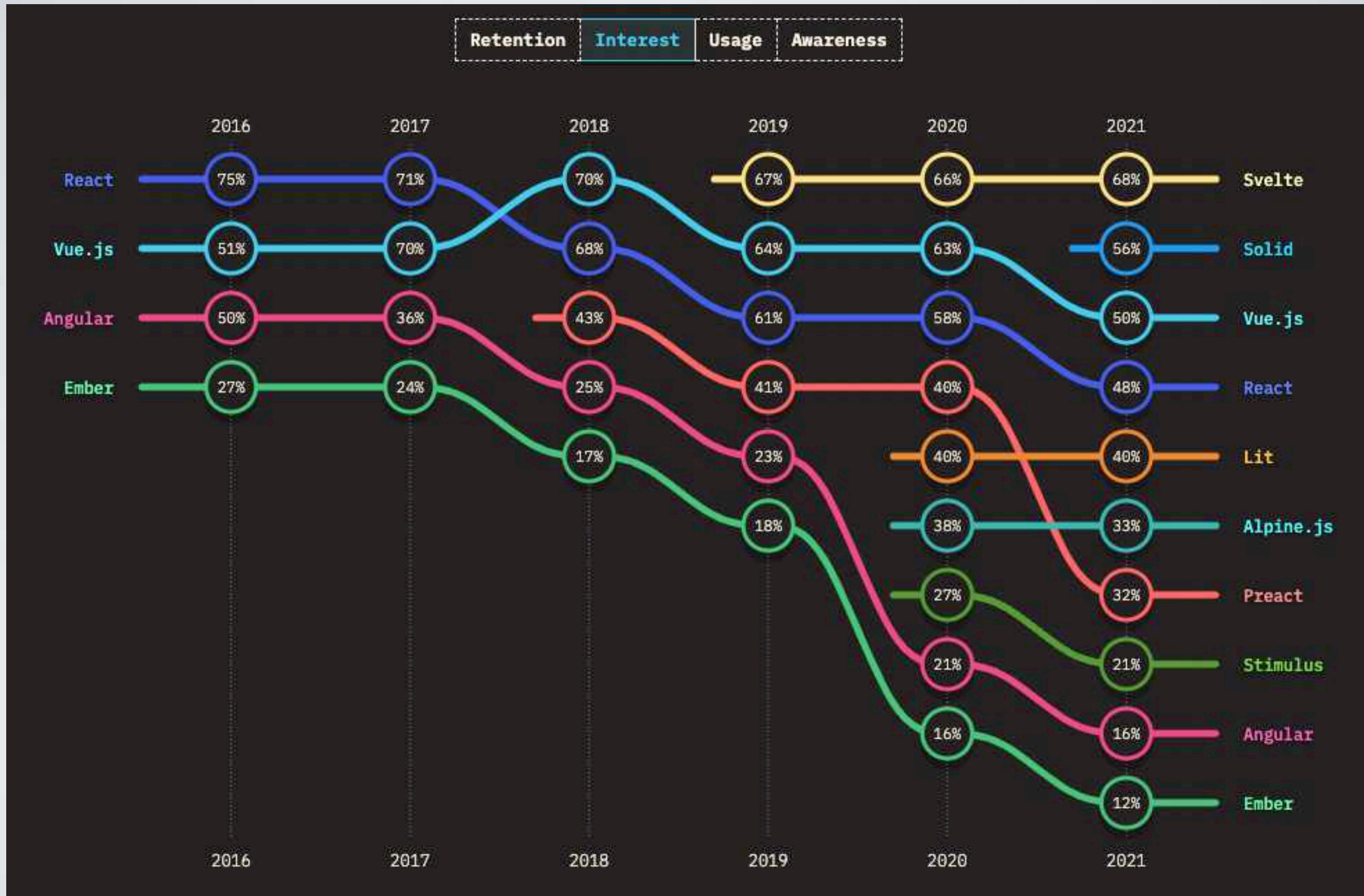


- Code Example
- How does it work?
- Implications
- One Advantage
- One Problem

Takeaway

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

State of JavaScript Survey 2021:



In the Beginning there
was Darkness ...



... then the DOM was created.

... and then 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 ...

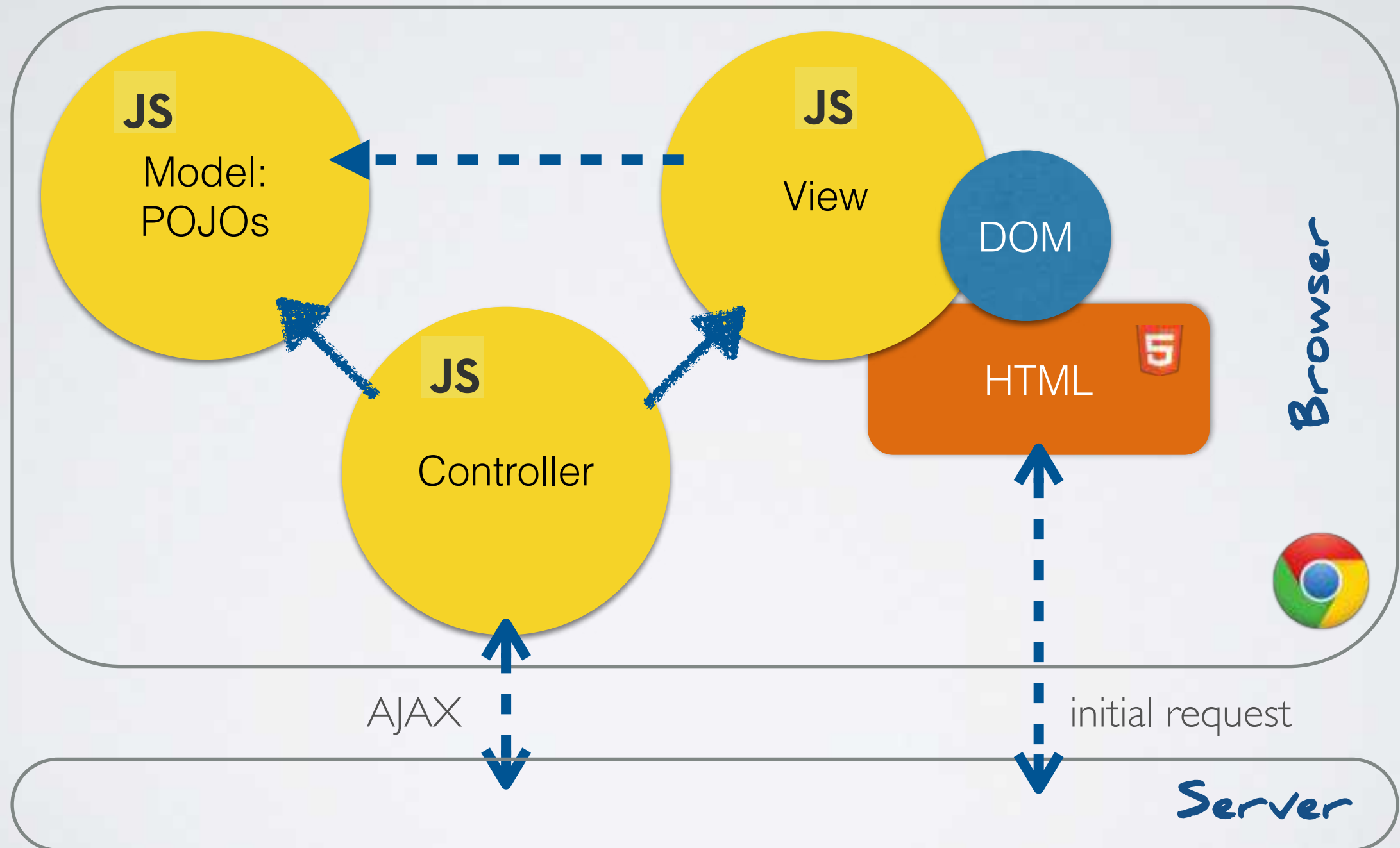


... a new hope ...

Model View Controller



Client Side MVC





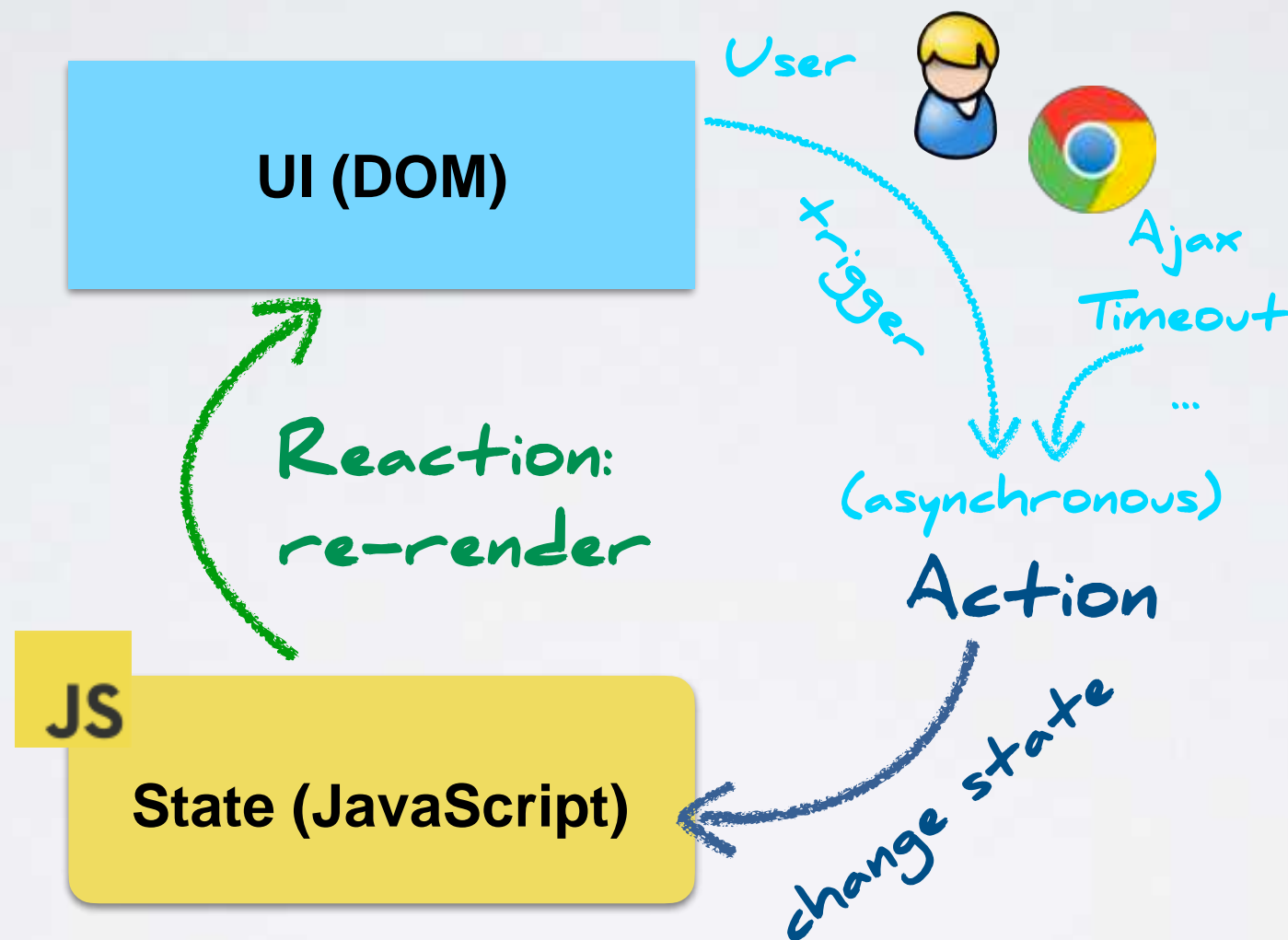
Thou shalt not manipulate
the DOM!



the DOM *is* updated

State is Managed in JavaScript

The UI renders the state and "signals" events.



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

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.



$UI = f(state)$

When to call?

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?
The UI should (automatically) update when the state changes.




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



Framework Reactivity



Zone.js

Change
Detection

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): `setTimeout`, `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.

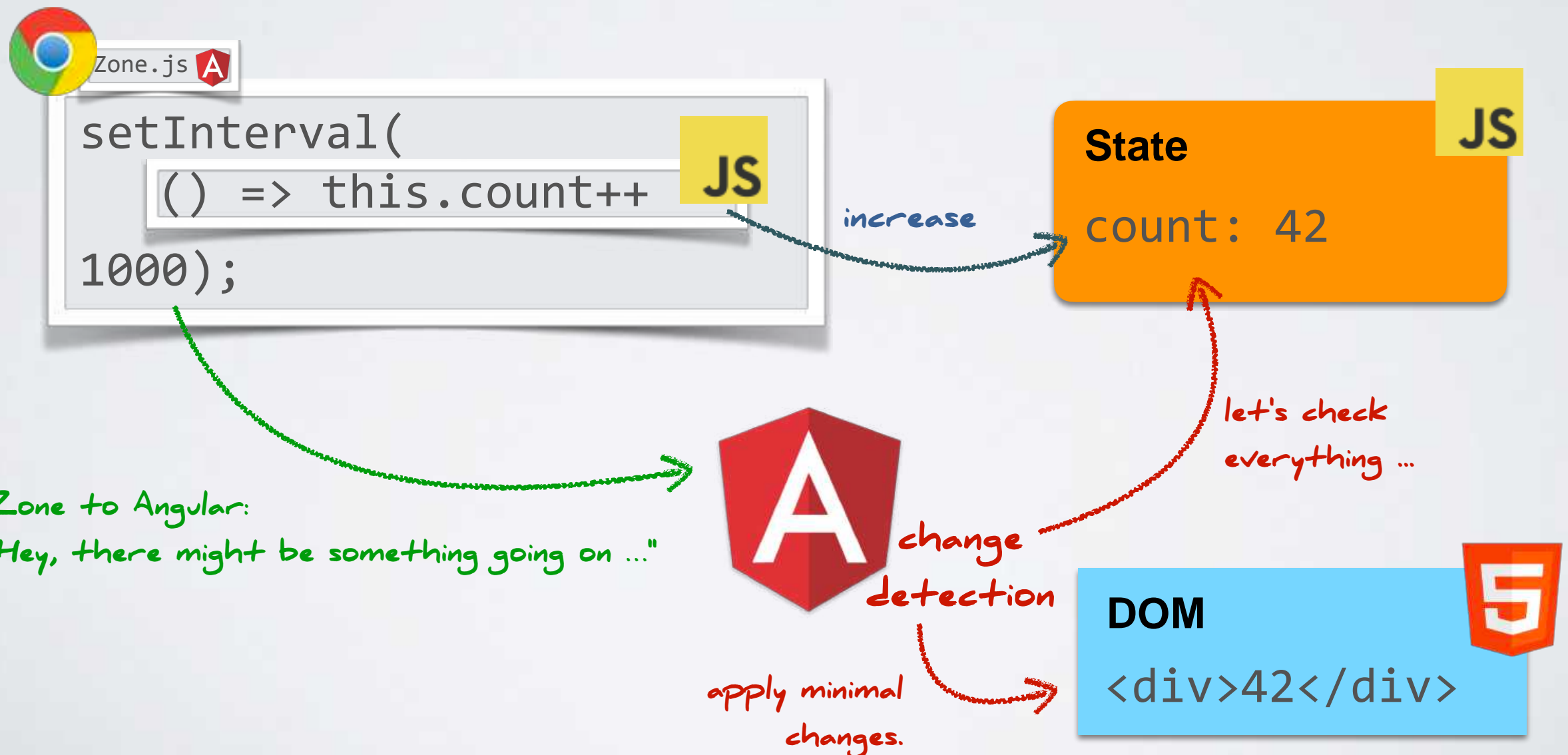
Angular is running inside the NgZone (a zone created via Zone.js). When async APIs are executed Angular gets notified when the execution has finished and triggers change detection.

Default Reactivity in Angular

"simulated reactivity"

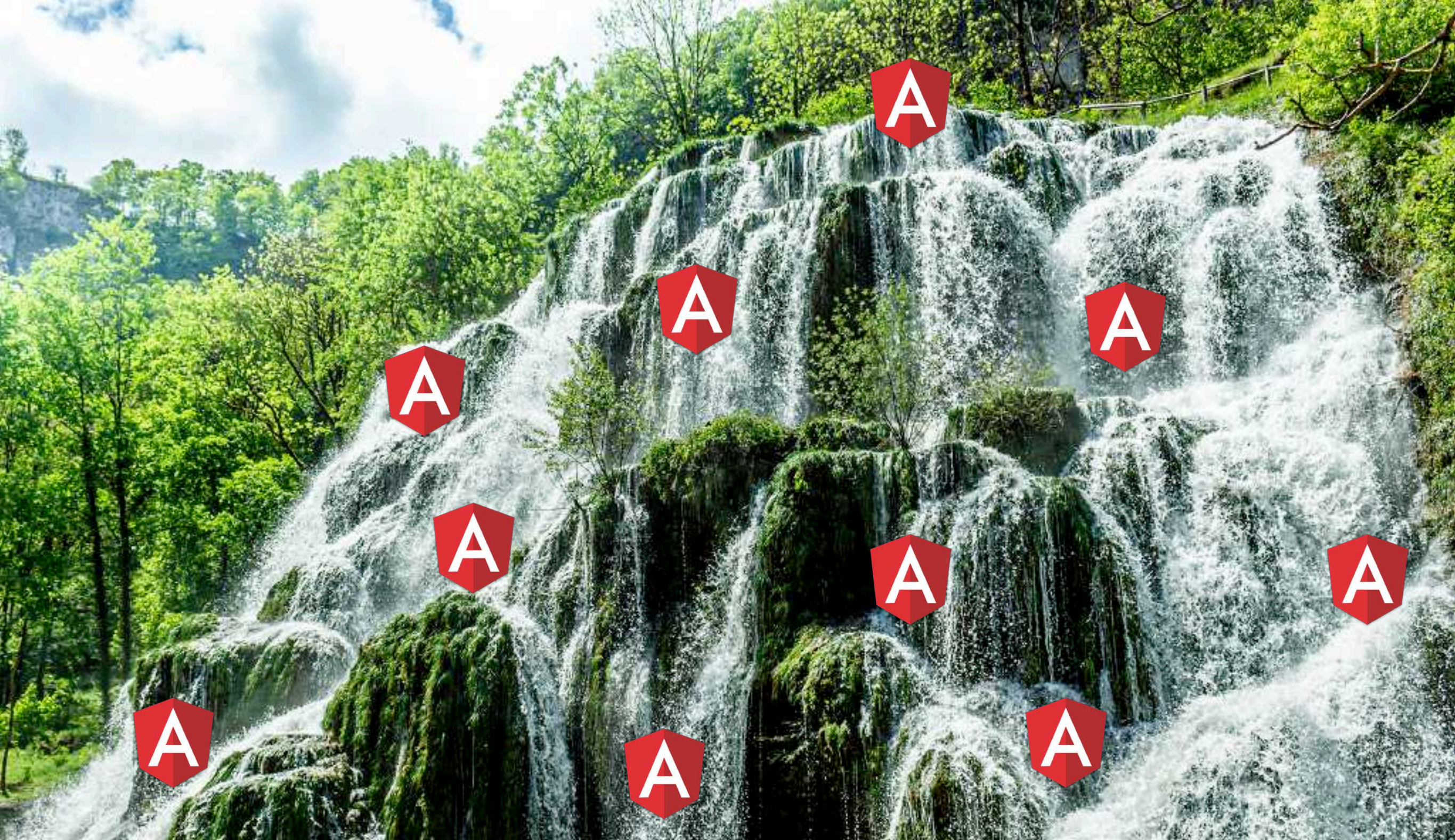
UI = f(state)

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 idiomatic 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 ...

- unidirectional data-flow
- avoid setter/getters?
- no native async/await

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.

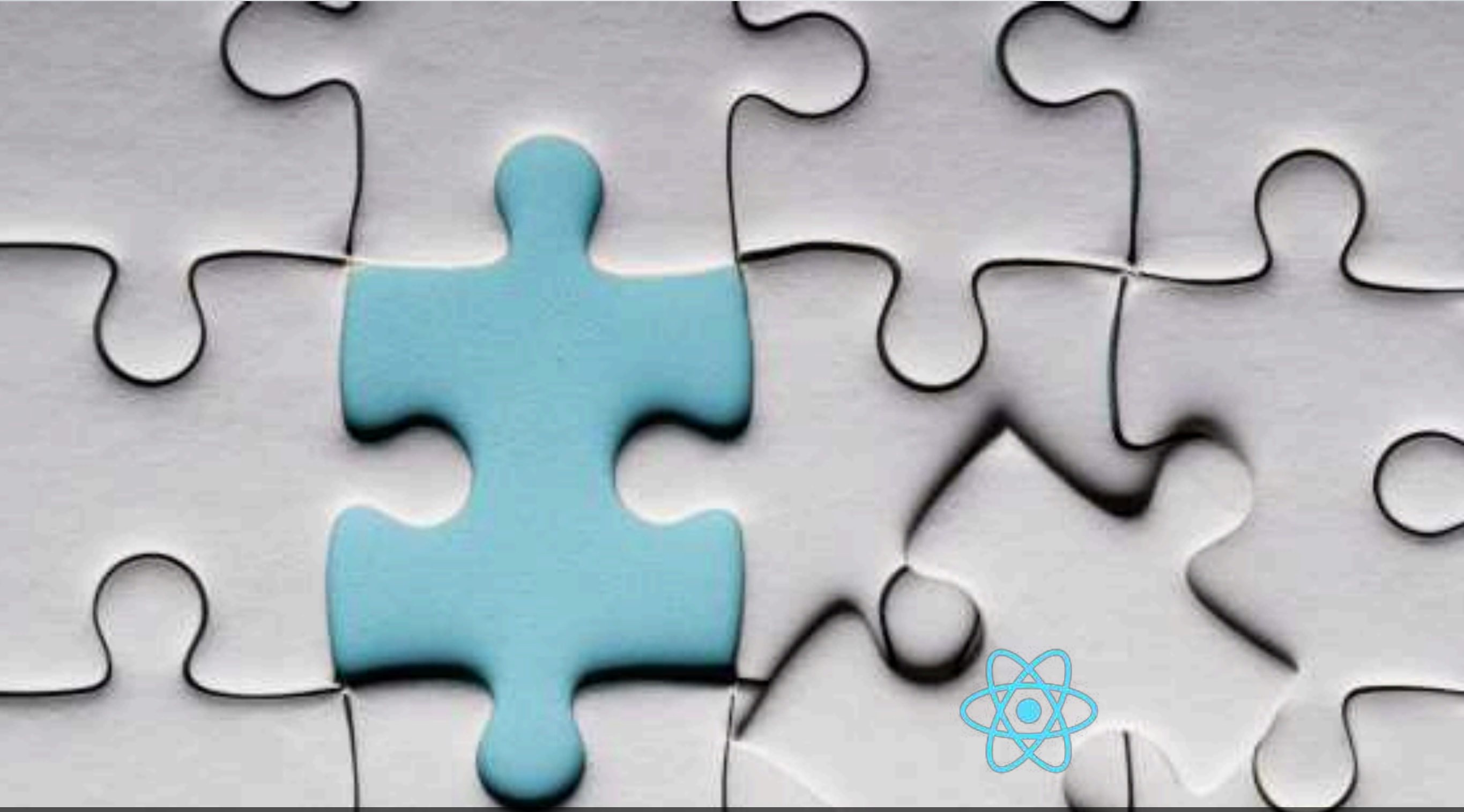
A common alternative in Angular is to model Reactivity explicitly with RxJS, this is a form of *explicit reactivity*.

Angular Reactivity Variations

`ChangeDetectionStrategy.OnPush`



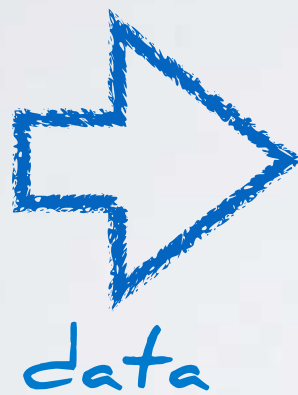
Zone-Less



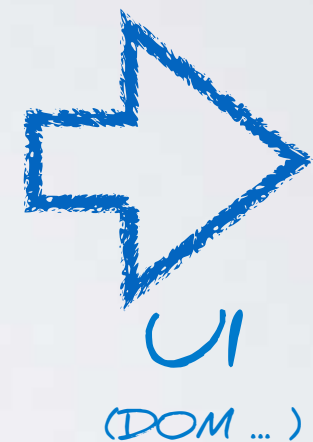
(Missing?) Reactivity in React

Function Components

Components are written as plain JavaScript functions.



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

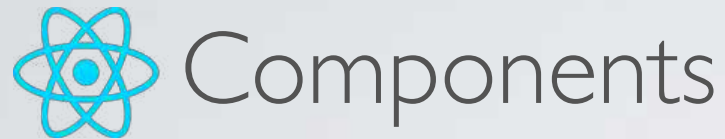


The function is called each time the UI is rendered (i.e. with every data-change)

A Visual Guide To React Mental Models:

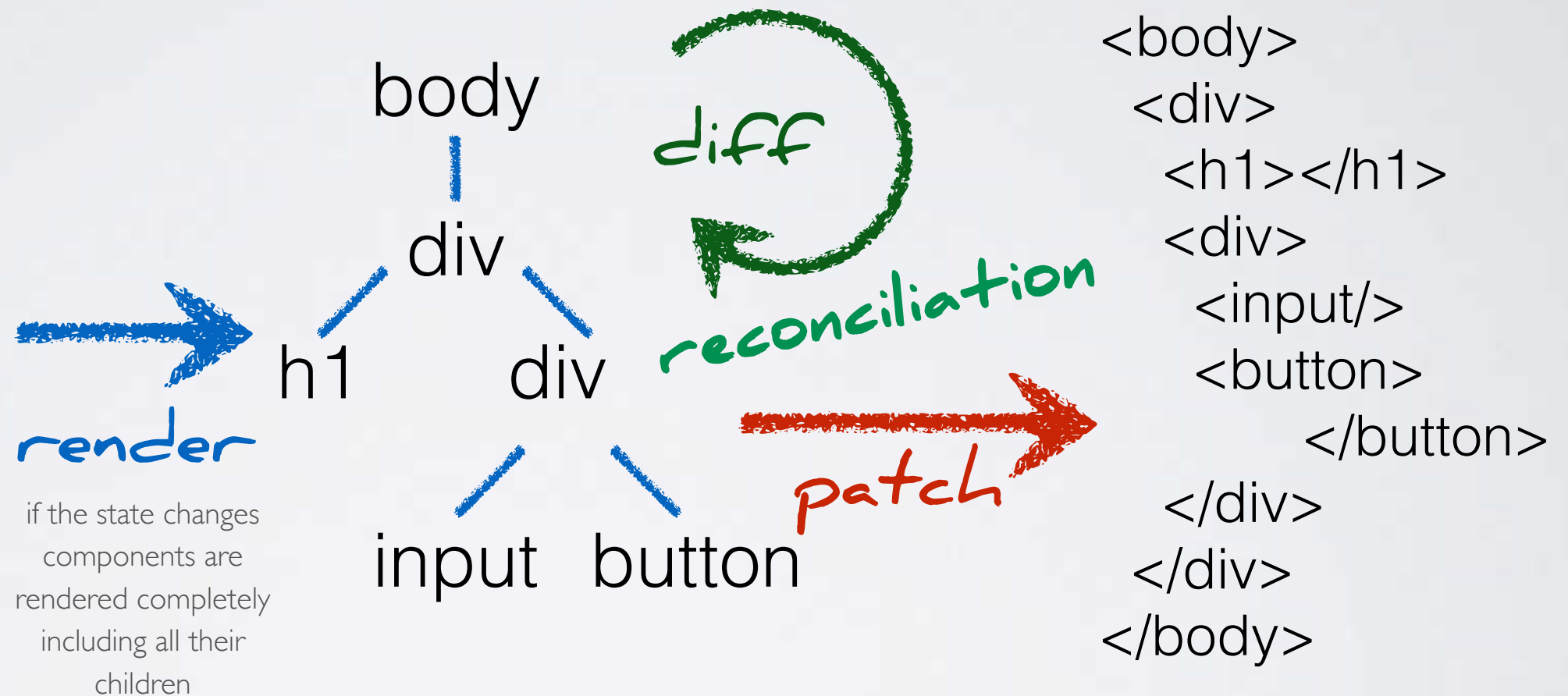
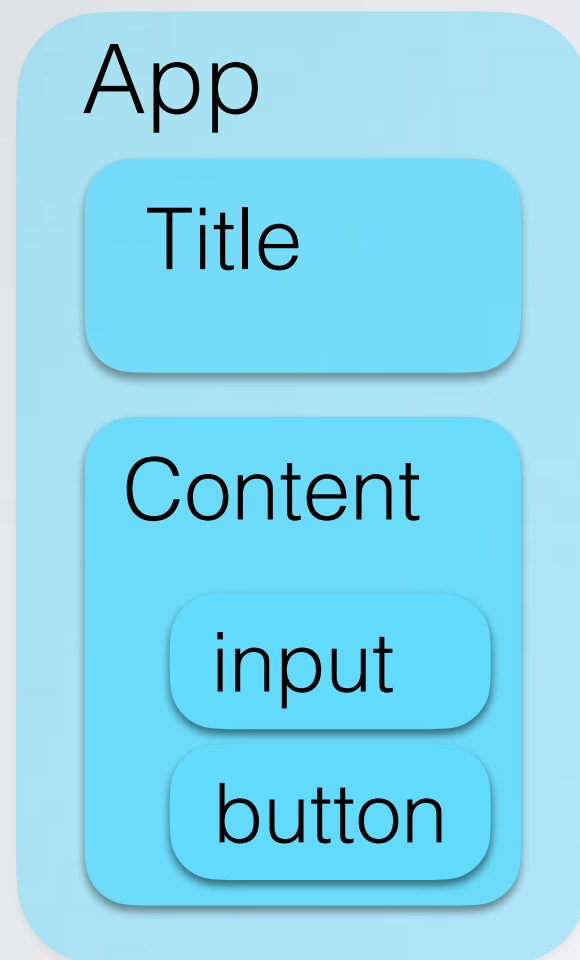
<https://obedparla.com/code/a-visual-guide-to-react-mental-models/>

The Virtual DOM




Virtual DOM

In-Memory, implemented in JavaScript



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



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

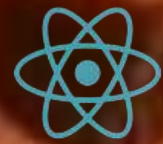
```
export function Counter() {  
  const [count, setCount] = useState(0);
```

React is used
to manage the
state

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

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

ONE DOES NOT SIMPLY



CHANGE STATE IN REACT

Reactivity in React

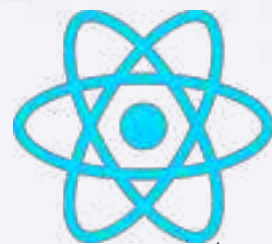
$$UI = f(state)$$

triggered by
the programmer

JS

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

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

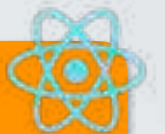


virtual
DOM

apply minimal changes.

State

count: 42



update the state

trigger re-rendering

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".

Reactive State in 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

fine-grained $UI_x = f_x(state_x)$ *triggered by reactive state*

$UI_y = f_y(state_y)$

$UI_z = f_z(state_z)$

JS

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

State (Vue proxy)

{ count: 42 }

JS

get/set count()

increase

trigger fine-grained re-rendering

apply minimal changes.

DOM

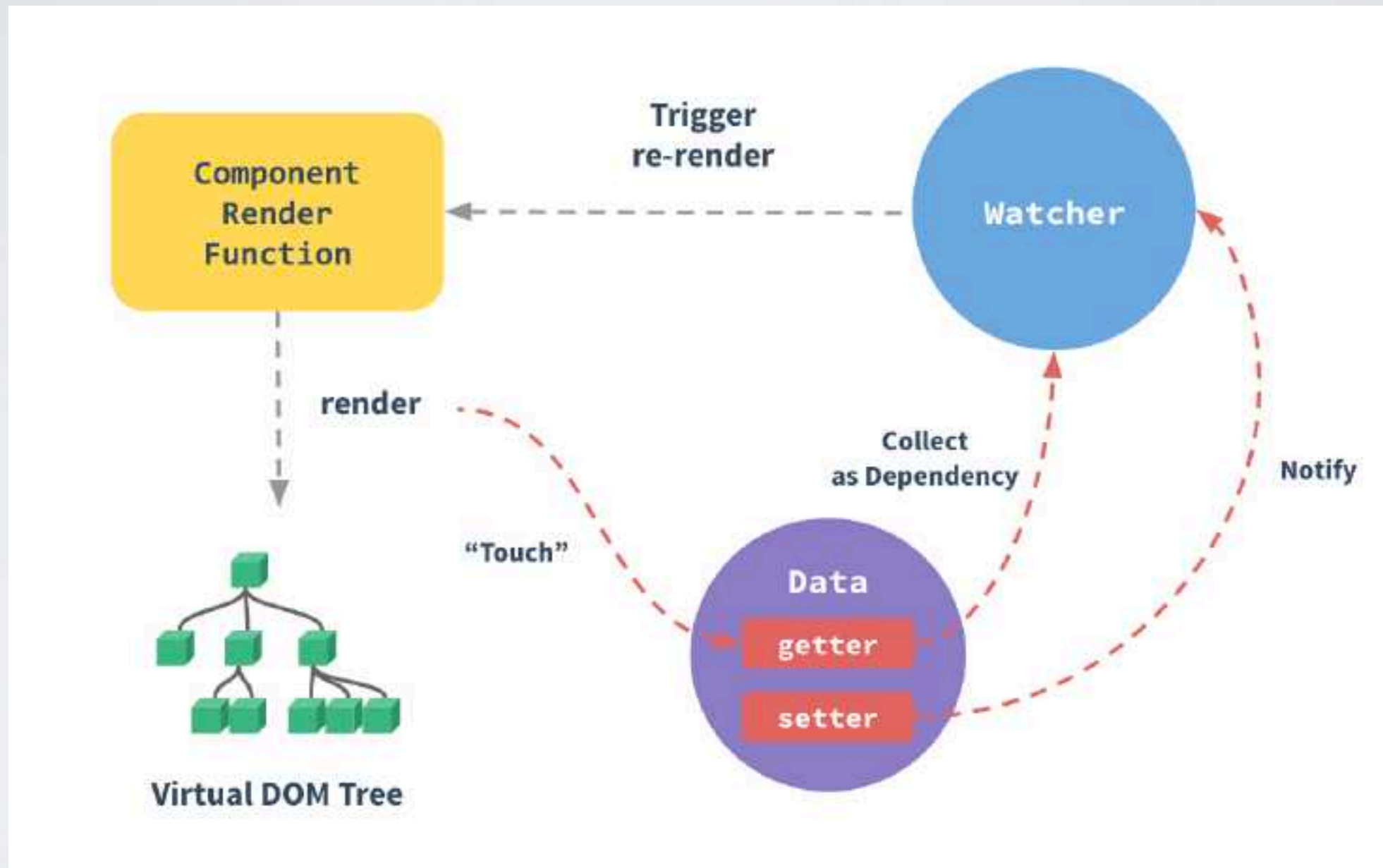
<div>42</div>



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



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.

Svelte

"Embrace the Compiler!"



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.

<h2>{count}</h2>

Reactivity in Svelte

fine-grained $UI_x = f_x(state_x)$ *triggered by compile-time generated code*

$UI_y = f_y(state_y)$

$UI_z = f_z(state_z)$

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 = 'Jonas';
  let number = 0;

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

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



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

helper functions

```
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();
    }
  };
}
```

life-cycle

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

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

export default App;
```

initialization

Svelte Reactivity



"Compile-Time-Generated Reactivity"

Strength

Very compact and intuitive code.

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




Significantly faster than the other mainstream frameworks.

Weakness

"extending" the semantics of JavaScript

Circling Back:

All Modern Frontend Frameworks are Compilers!

 Angular	<ul style="list-style-type: none">• TypeScript• Template• (workarounds)
 React	<ul style="list-style-type: none">• JSX• TypeScript
 Vue	<ul style="list-style-type: none">• SFC• Template• TypeScript

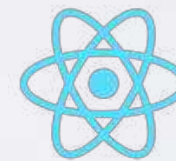


... but Svelte goes one step further.

Fun Fact:

React and Vue have plans for future compiler features that are changing the semantics of JavaScript ...
(reducing boilerplate for convenience)

"React Forget" - A Memoizing Compiler



<https://www.youtube.com/watch?v=IGEMwh32soc>

<https://reactjs.org/blog/2022/06/15/react-labs-what-we-have-been-working-on-june-2022.html>

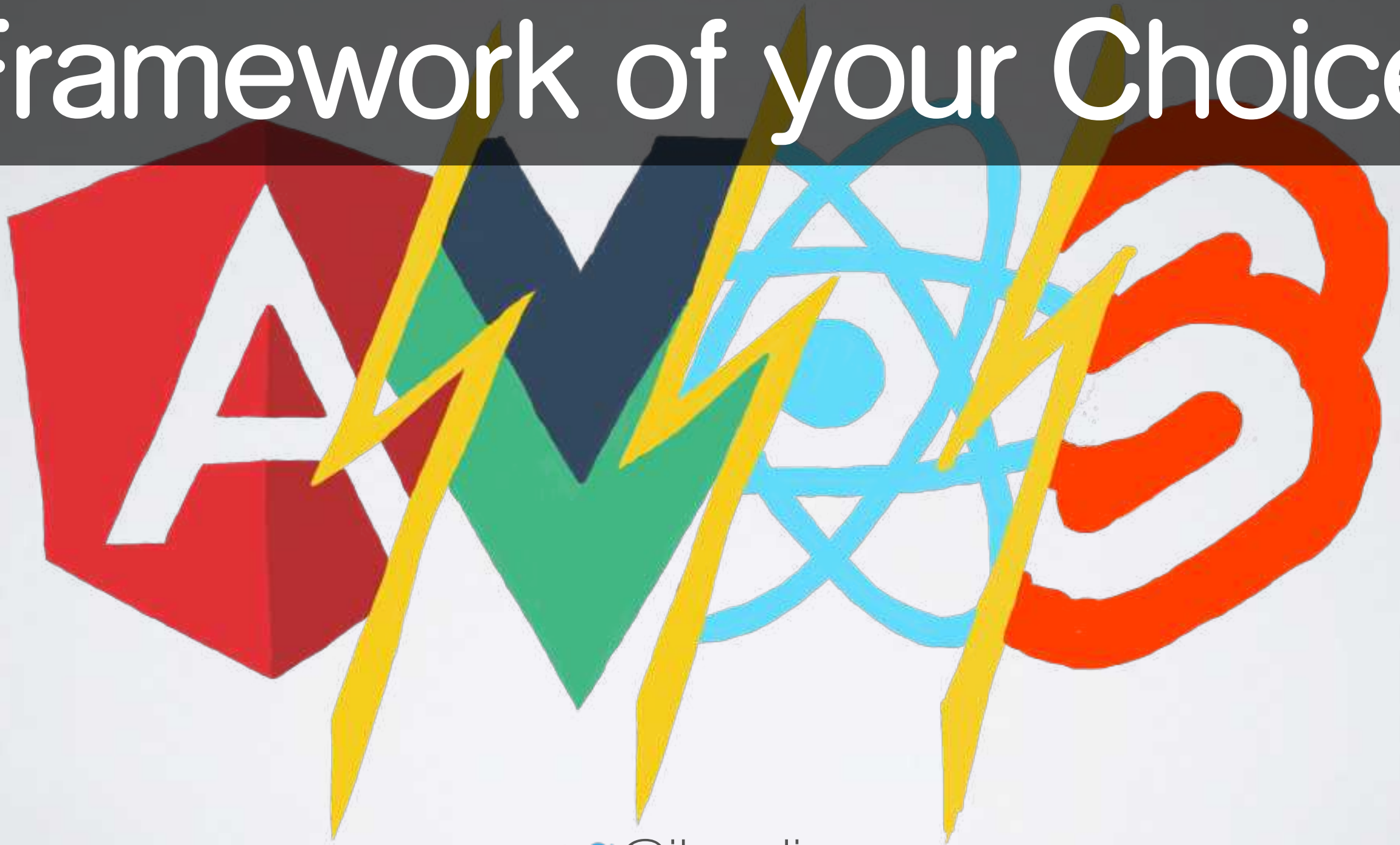
Vue Reactivity Transforms



<https://vuejs.org/guide/extras/reactivity-transform.html>

<https://github.com/vuejs/rfcs/discussions/369>

Have Fun with the Framework of your Choice!



 @jbandi

Code: <https://github.com/jbandi/framework-reactivity-2022>

QUESTIONS?



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

jonas.bandt@ivorycode.com