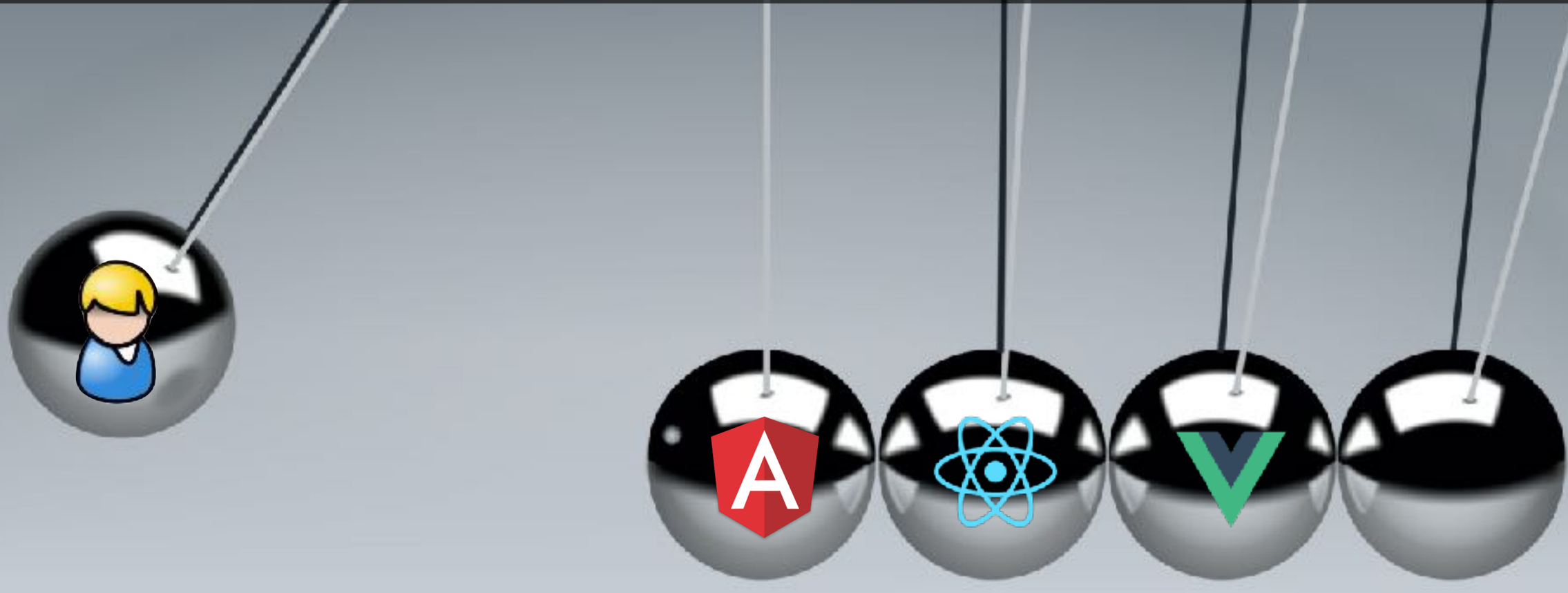
From User Action to Framework Reaction

Reactivity in modern Frontend Frameworks



Jonas Bandi
jonas.bandi@ivorycode.com
Twitter: @jbandi

- Freelancer, in den letzten 6 Jahren vor allem in Projekten im Spannungsfeld zwischen modernen Webentwicklung und traditionellen Geschäftsanwendungen.
- Dozent an der Berner Fachhochschule seit 2007
- In-House Kurse: Web-Technologien im Enterprise
  UBS, Postfinance, Mobiliar, SBB, BIT, BSI, Elca ...

JavaScript / Angular / React / Vue.js
Schulung / Beratung / Coaching / Reviews

jonas.bandi@ivorycode.com

http://ivorycode.com/#schulung

Agenda

Intro

Baseline:
"Out of the Box"-Reactivity of  ,  and 
(Slides and Demos)

Further Explorations:
Demos and Code Explorations

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

Reactivity ?

Reactive Programming?

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

- Wikipedia

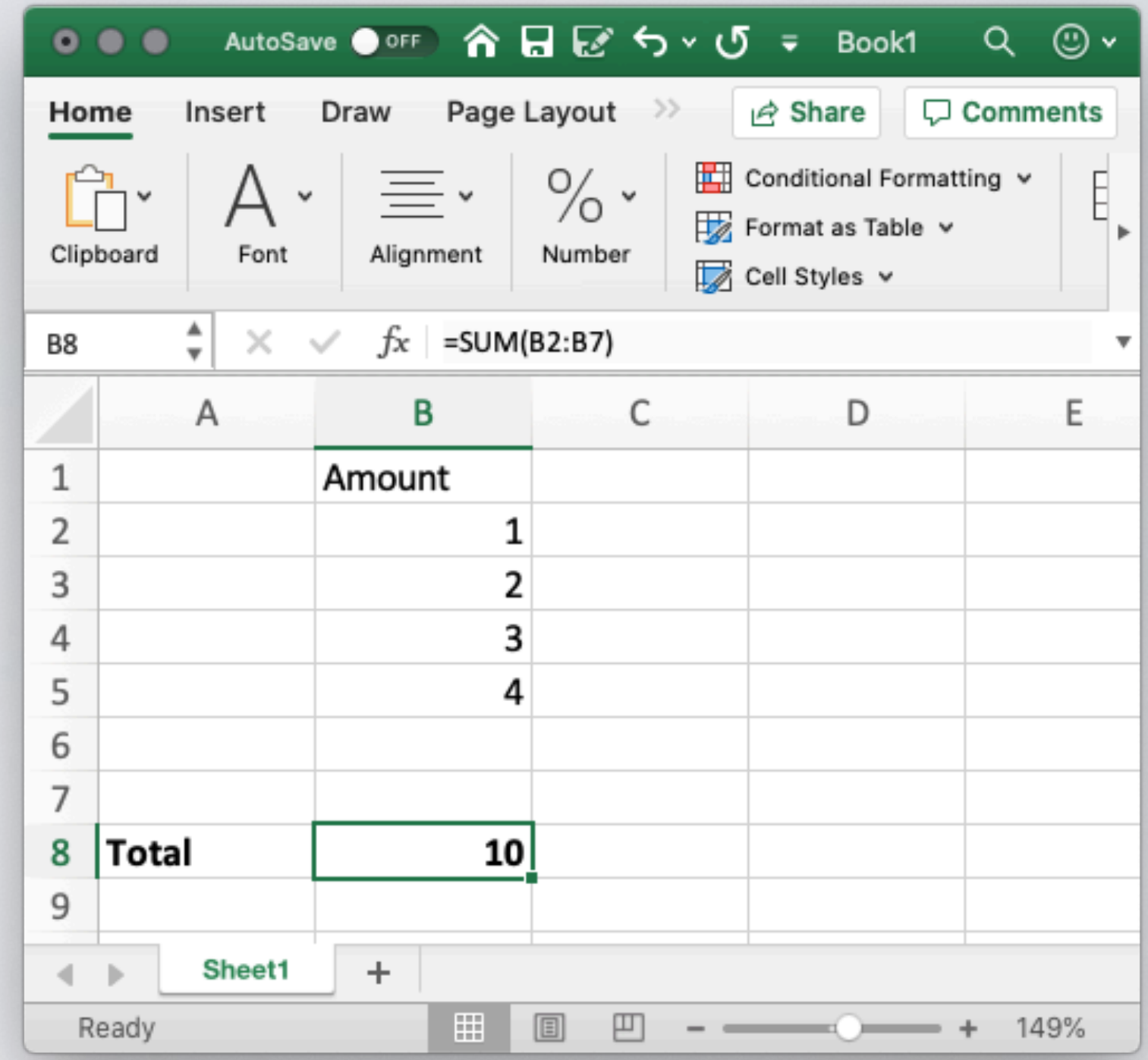
Reactive programming is programming with asynchronous data streams.

Andre Staltz

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`));
```


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%

In the Beginning there was Darkness ...

... then the DOM was created.

… and we manipulated the DOM …

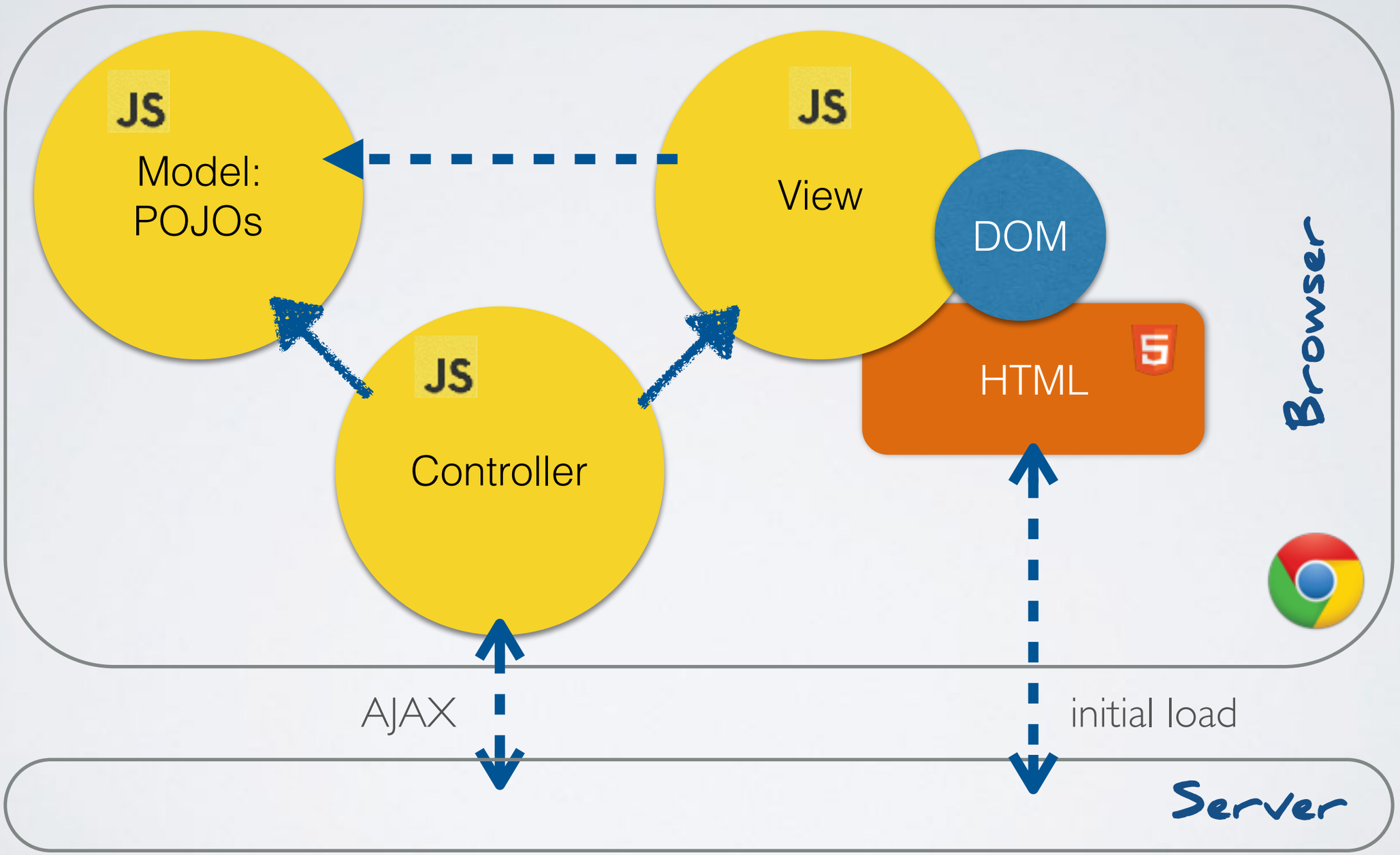
```
$(".menu-item")
  .removeClass("active")
  .addClass("inactive ")
  .css("padding-left", "0px")
  .find(".trigger")
  .click(function(ev) {
     // spaghetti carbonara?
  })
  .each(function () {
     // spaghetti napoli?
  });
```

… the Dark Ages of DOM …

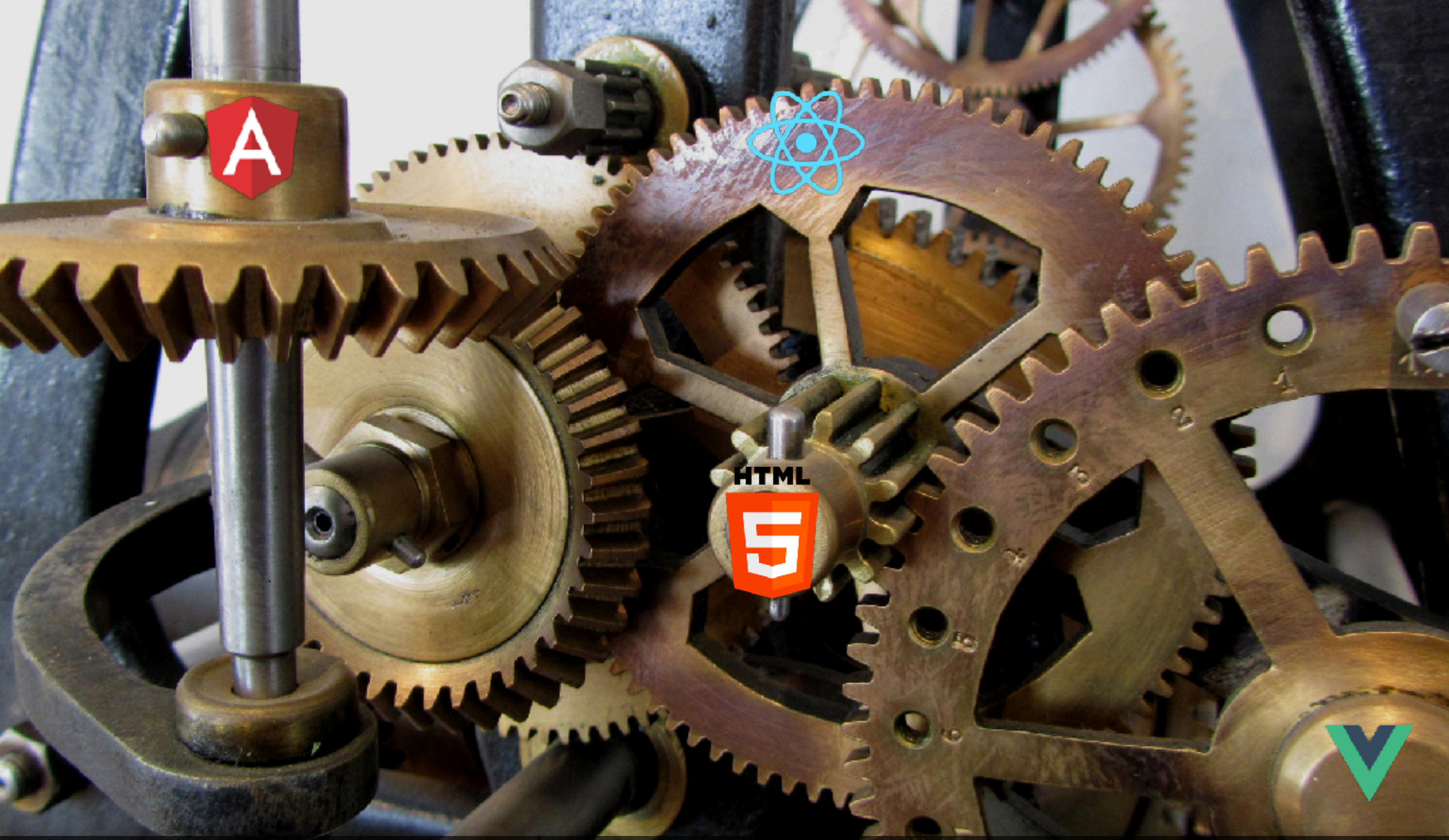

ember
BACKBONE.JS
Knockout.
… a new hope …

Model View Controller

Client Side MVC

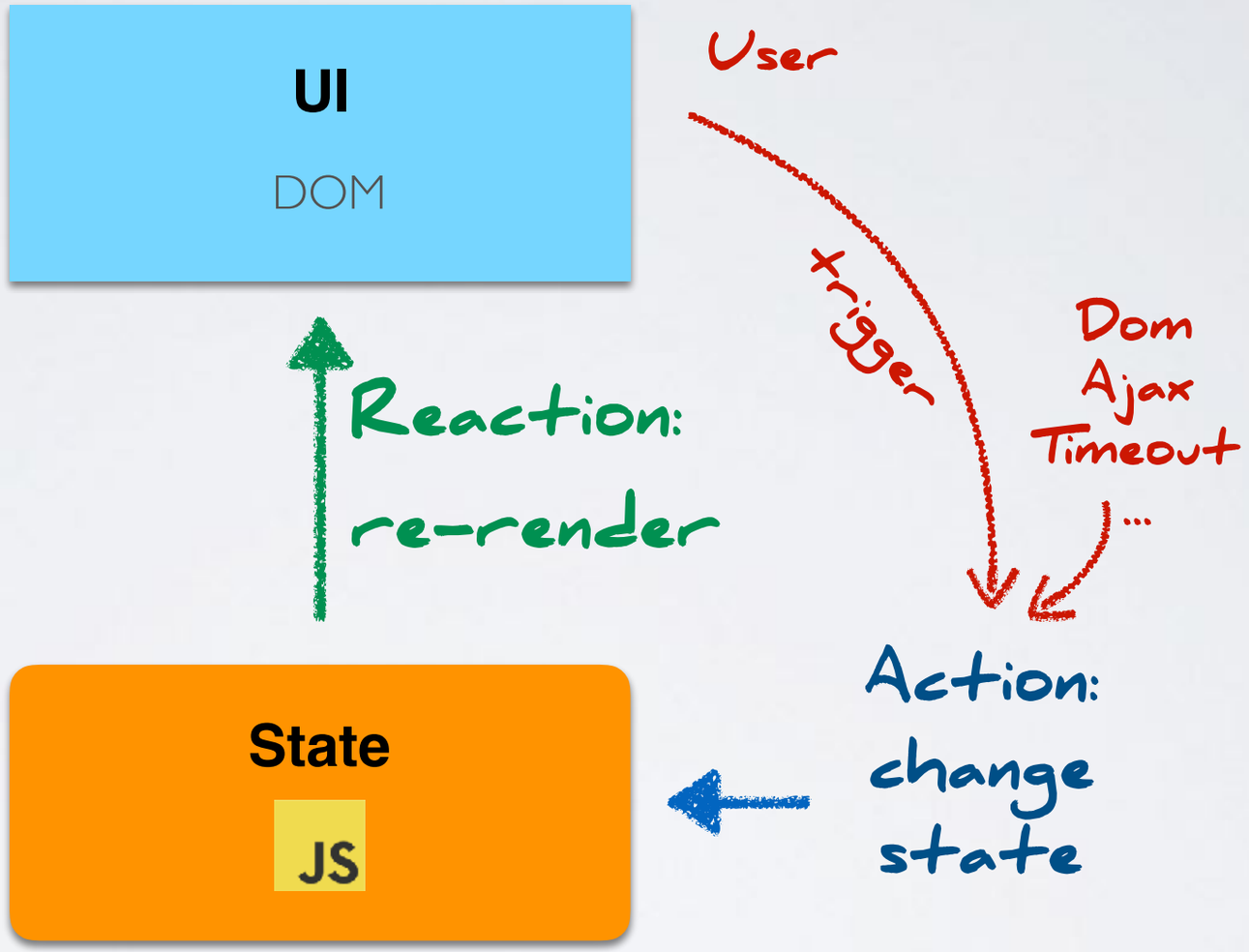


Thou shalt not manipulate the DOM!



the DOM *is* updated

State is Managed in JavaScript



Reactivity: The framework 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.

With modern SPA architectures (MVC, MVP, Components …) the client 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.

Reactive programming in general addresses the question: How to deal with change over time?

The UI should (automatically) re-render when the state changes.

https://blog.danlew.net/2017/07/27/an-introduction-to-functional-reactive-programming/

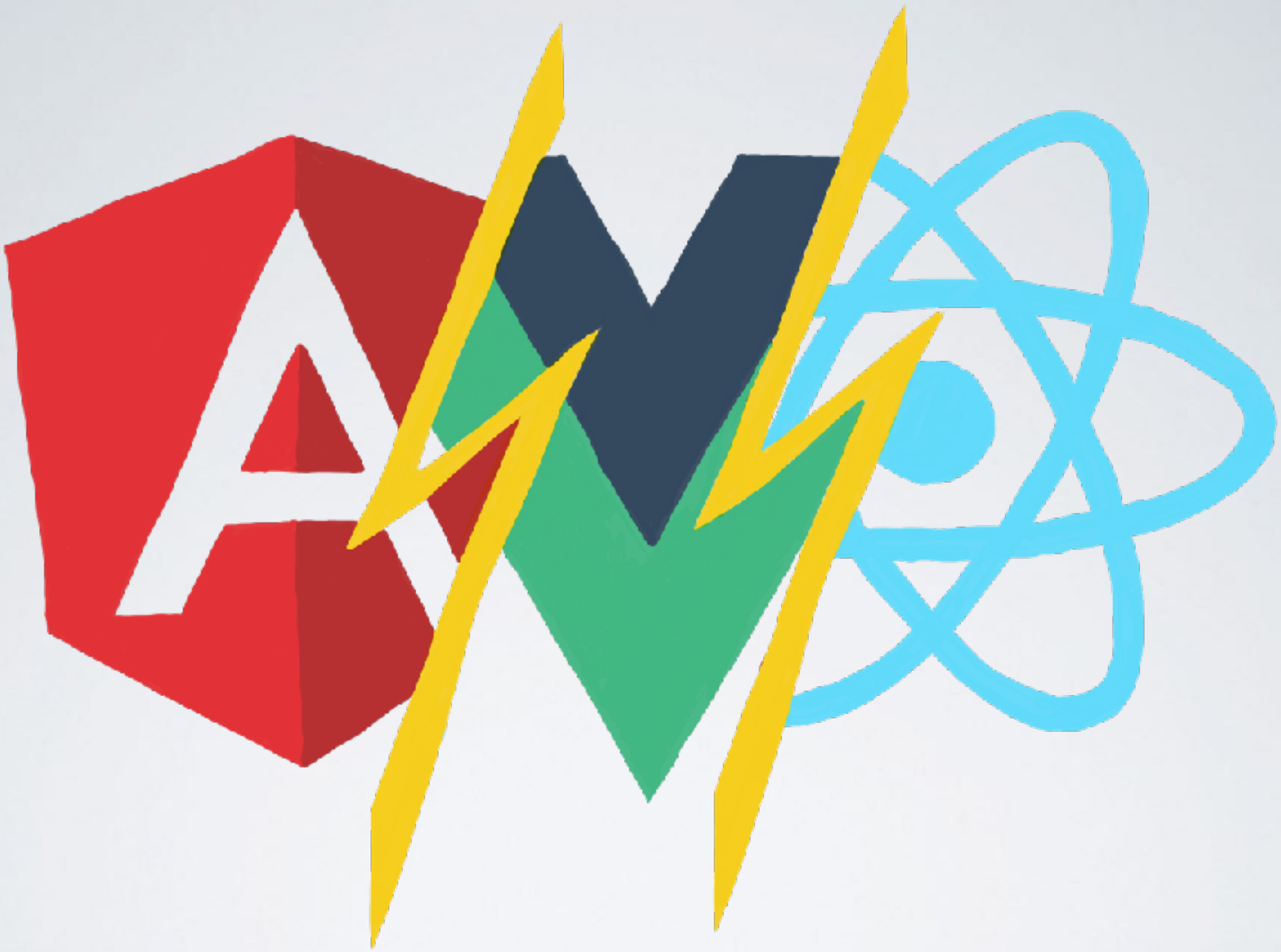
Rich Harris
@Rich_Harris

Following

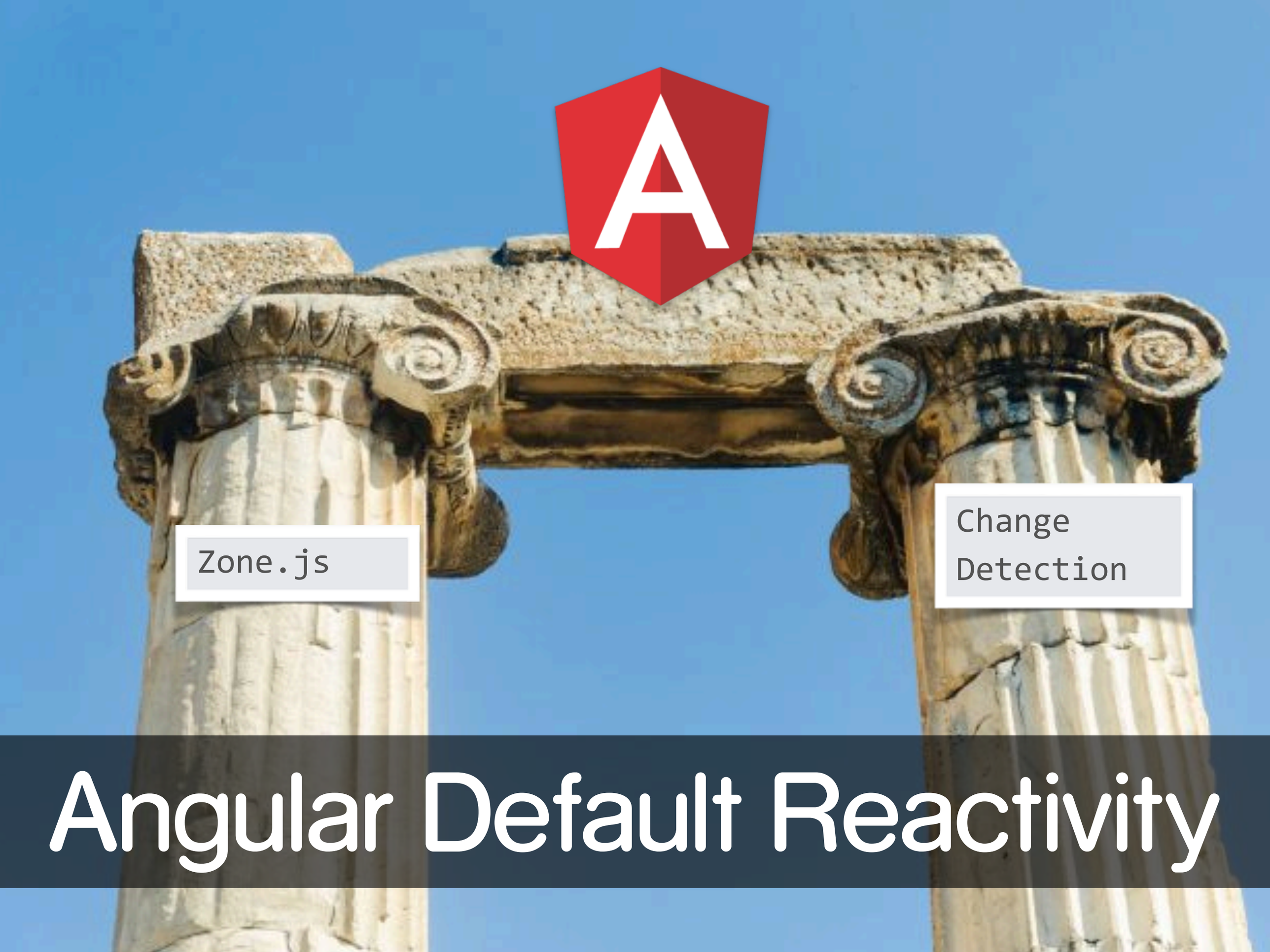
The problem all frameworks are solving is *reactivity*. How does the view react to change?

- 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 🤷)

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



Baseline: Core Reactivity



Angular Default Reactivity

```
@Component({
  selector: 'app-counter',
  template: `
    <div>
      {{count}}
    </div>
  `
})
export class CounterComponent implements OnInit {
  count = 0;

  ngOnInit() {
    setInterval(() => { this.count++; }, 1000);
  }
}
```

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.

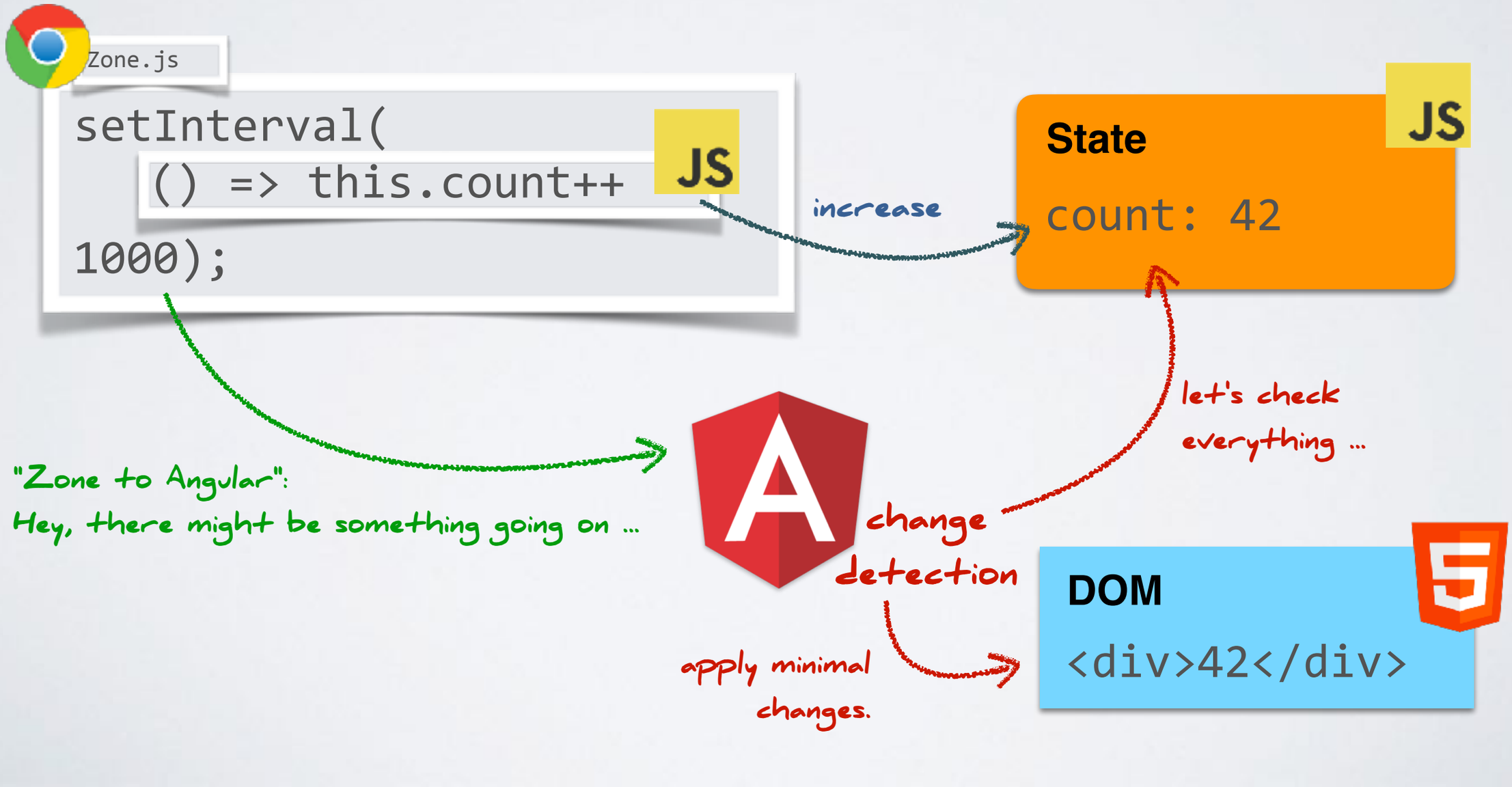
https://github.com/angular/zone.js/
https://medium.com/better-programming/zone-js-for-angular-devs-573d89bbb890

Default Reactivity in Angular

"simulated reactivity"

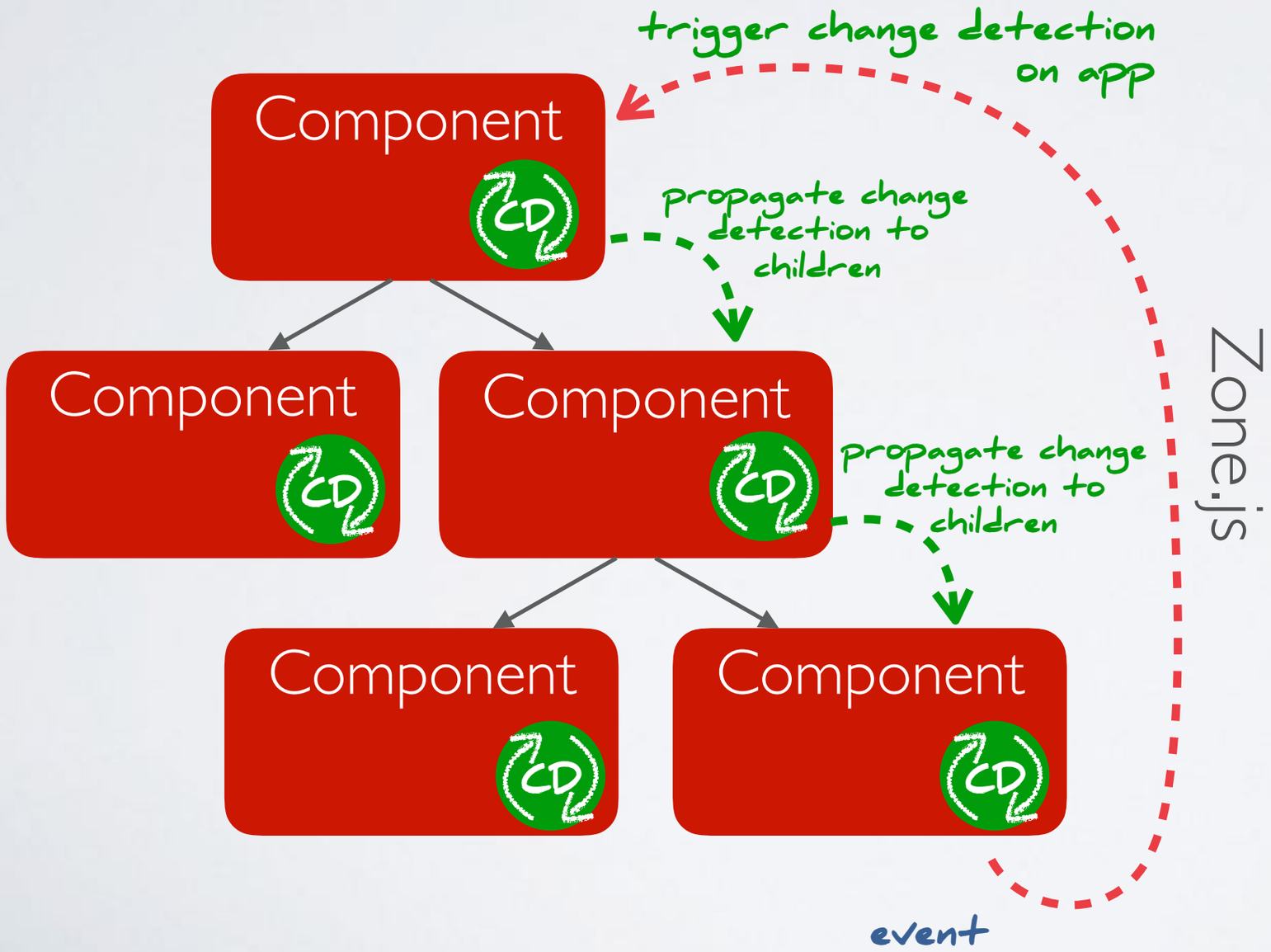
UI = f(state)

Triggered by Zone.js
"simulated reactivity"



Default Change Detection

As default Angular detects changes by inspecting the state of all components every time change detection is triggered.



Change detection is always triggered at the top of the component hirarchy and propagates down to each child.

Every value used in a template is inspected and compared to the previous value.

Checking all components on every possible event can be performance intense.

Default Reactivity in Angular

Zone.js with Default Change Detection:

- are a form of *simulated reactivity:* the framework does not react to changes but to events that might potentially resulted in changes
- are 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

Default Angular Reactivity

"Simulated Reactivity"

Strength

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

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 ...

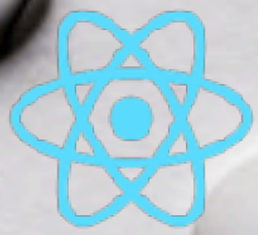
- avoid setter/getters
- unidirectional data-flow
- avoid async/await

Unidirectional Data Flow

Angular enforces *unidirectional data flow* from top to bottom of the component tree.

- A child is not allowed to change the state of the parent once the parent changes have been processed.
- This prevents cycles in the change detection.
  (to prevent inconsistencies between state and ui and for better performance)
- In development mode Angular performs a check (a second change detection) to ensure that the component tree has not changed after change detection. If there are changes it throws a `ExpressionChangedAfterItHasBeenCheckedError`.

https://indepth.dev/everything-you-need-to-know-about-the-expressionchangedafterithasbeencheckederror-error/



(Missing?) Reactivity in React

Function Components

A components is a plain JavaScript function.

properties

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

UI (DOM ...)

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

A component transforms JavaScript state into the DOM structure:

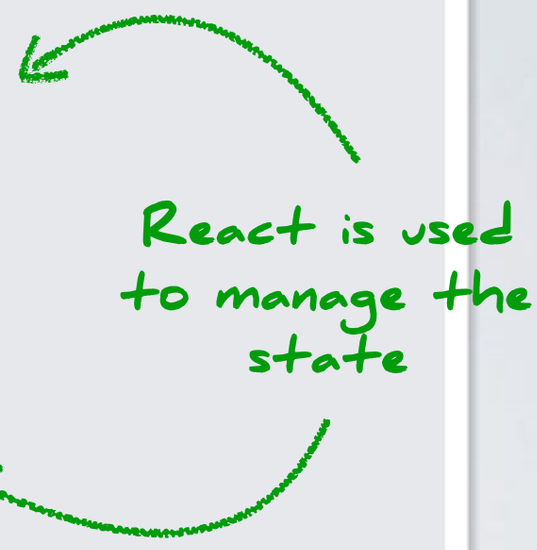
$$UI = f(state)$$

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

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

    useEffect(() => {
        setInterval(() => {
            setCount(count => count + 1);
        }, 1000)
    }, []);

    return <div>{count}</div>
}
```



Reactivity in React

UI = f(state)

triggered by the programmer

JS

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

State

```
count: 42
```

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

virtual DOM

update the state

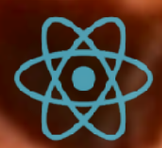
trigger re-rendering

apply minimal changes.

DOM

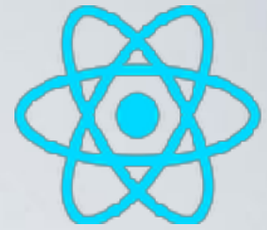
```
<div>42</div>
```

ONE DOES NOT SIMPLY



CHANGE STATE IN REACT

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.

Reactive State in Vue

```
import { reactive, watch } from 'vue'

const state = reactive({
  count: 0
})

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

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

changing state triggers re-rendering

The example is using the composition API of upcoming vue 3:
https://vue-composition-api-rfc.netlify.com/

This is available in Vue 2: https://github.com/vuejs/composition-api

Reactivity in Vue

UI = f(state)

triggered by reactive state

JS

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

increase

State (Vue proxy)

JS

```
{ count: 42 }
```

get/set count()

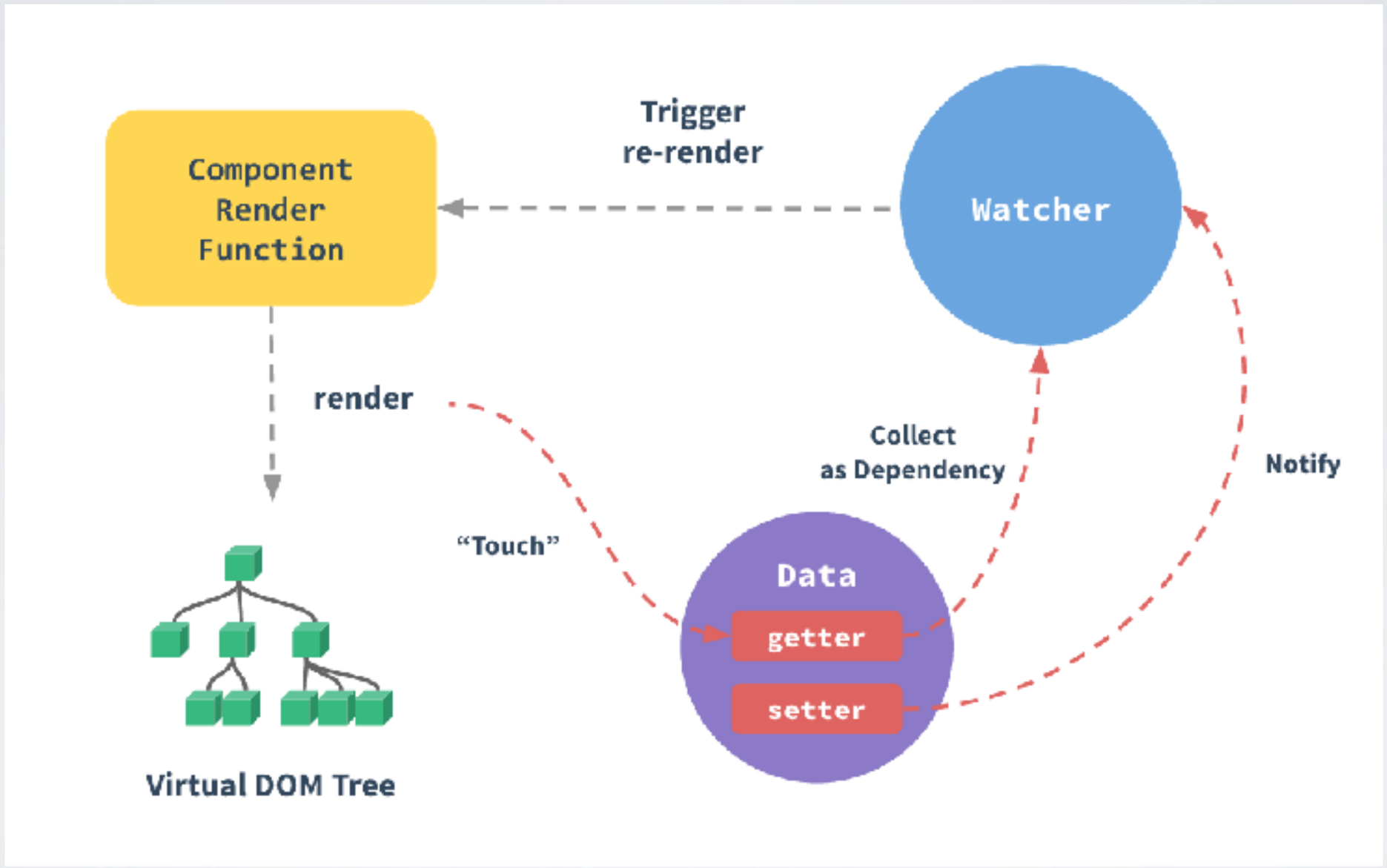
trigger re-rendering

apply minimal changes.

DOM

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

Change Tracking & Reactive State



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

Vue Reactivity

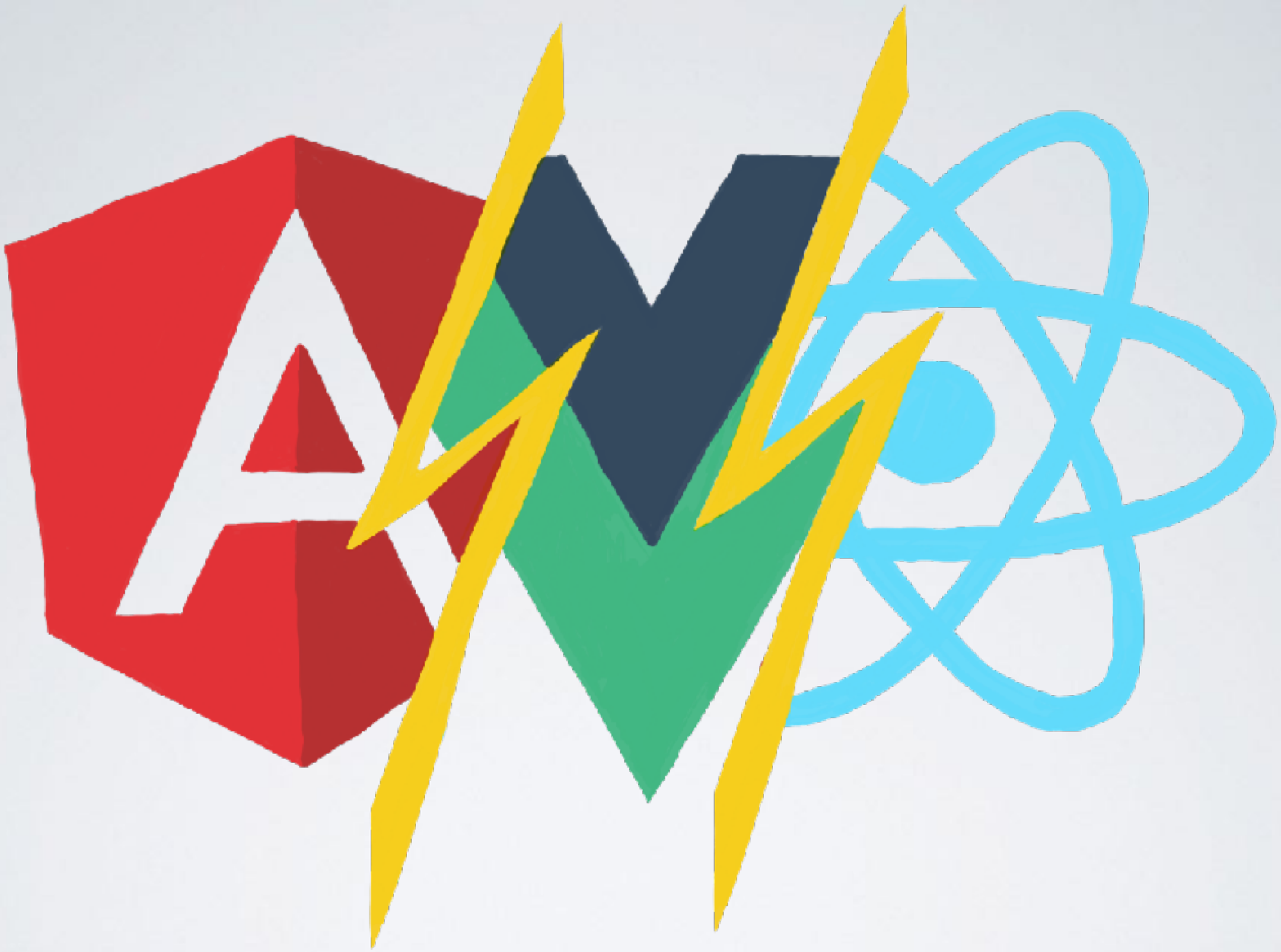
"Reactive State"

Strength

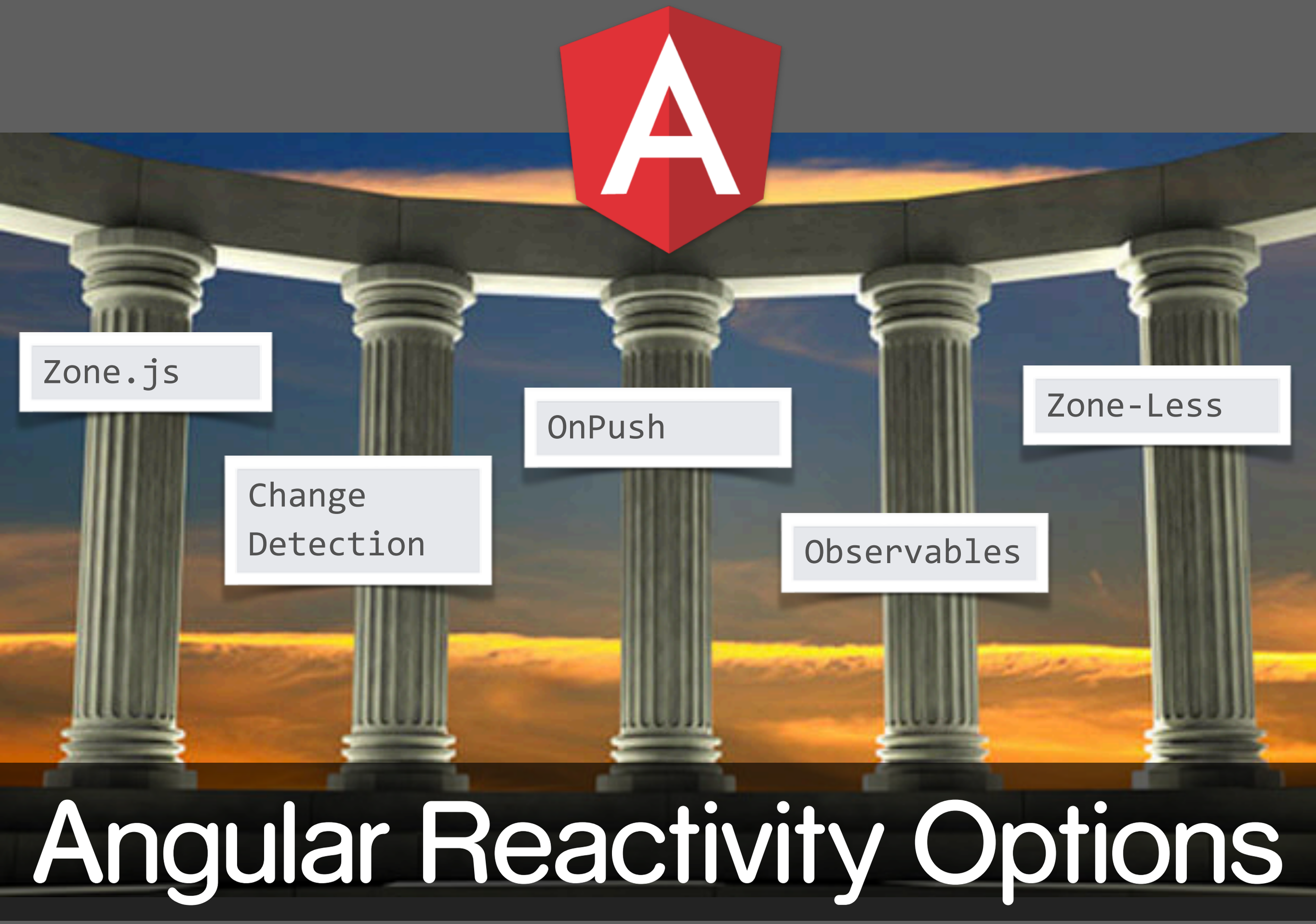
"True Reactivity": The state can be observed.

Weakness

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



Alternatives & Variations



Angular Reactivity Options

Observables & async Pipe

Reactivity realized with Streams

UI = f(state)

Triggered by Zone.js
"simulated reactivity"

UI$ = f(state$)

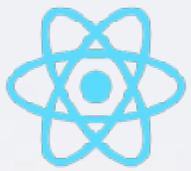
"streams: true reactive programming"

Computed State

Anything that can be derived from the application state, should be derived. Automatically.

- MobX Introduction

No ideomatic solution:
- getters/setters
- ngOnChanges
- Pure Pipe
- Observables
- (NgRx)

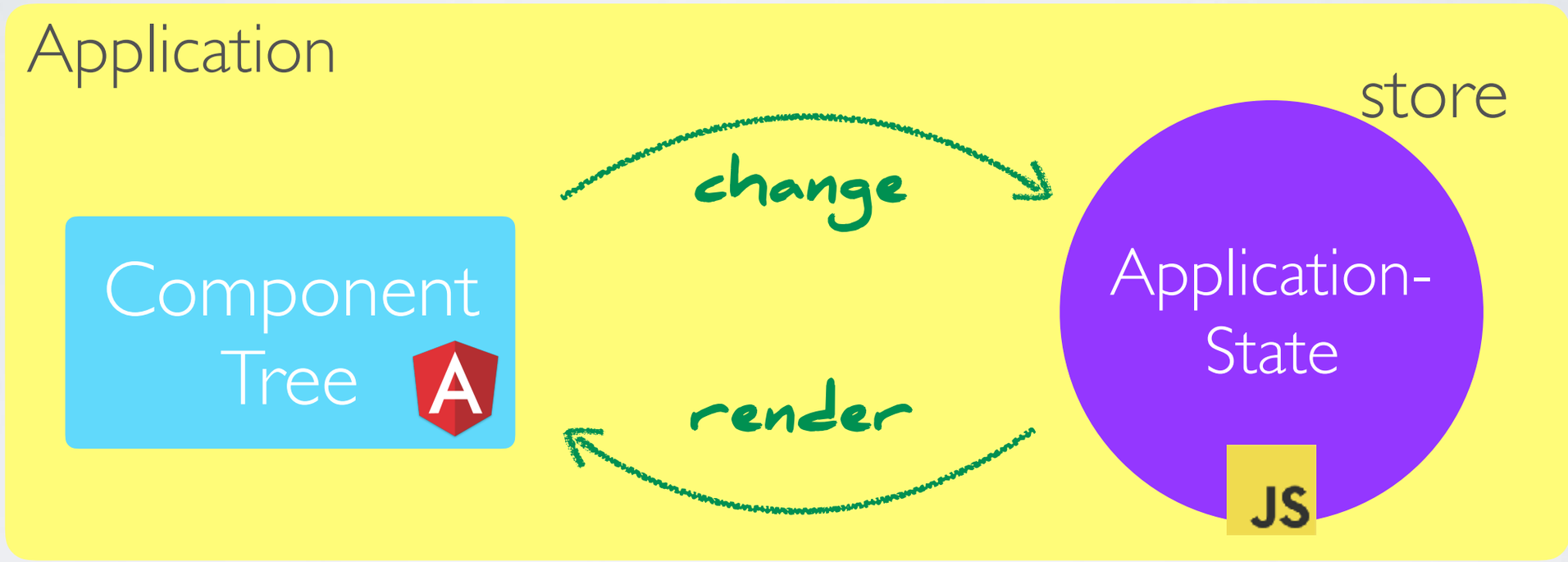


Derived state is calculated during rendering. Optmization with Memoization.

Computed properties.

State outside Components

A state container extracts the shared state out of the components, and manages it in a global singleton.



The component tree becomes a big "view", any component can access the state or trigger changes, no matter where they are in the tree!

Have Fun with the Framework of your Choice!

Twitter: @jbandi

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

Resources

- The Taxonomy of Reactive Programming
  https://vsavkin.com/the-taxonomy-of-reactive-programming-d40e2e23dee4
- Front end development and change detection (Angular vs. React):
  https://www.youtube.com/watch?v=1i8klHov3vA
- JS Roundabout, Reactivity in React and Vue, February 2018
  https://www.youtube.com/watch?v=HWZq_rIJU4o
- Why React Is *Not* Reactive - Shawn Wang @ ReactNYC
  https://www.youtube.com/watch?v=ZZoB5frIcnE
- Shift Dev 2019: "Rethinking Reactivity" - Rich Harris (New York Times)
  https://www.youtube.com/watch?v=gJ2P6hGwcgo
- The Return of 'Write Less, Do More' by Rich Harris | JSCAMP 2019
  https://www.youtube.com/watch?v=BzX4aTRPzno
- Reactivity: Vue 2 vs Vue 3
  https://www.vuemastery.com/blog/Reactivity-Vue2-vs-Vue3/