RXJS IS FAR FROM DEAD
LONG LIVE MOBX

@_MAXGALLO

HI 👋

I'M MAX* GALLO

ABOUT ME: 🍝 💻 🇬🇧 🎶 🏍 📷 ✈️ ✍️

PRINCIPAL ENGINEER @ **DAZN**

TWITTER: **@_MAXGALLO**
MORE: **MAXGALLO.IO**

OR MASSIMILIANO, IF YOU LIKE ITALIAN SPELLING CHALLENGES

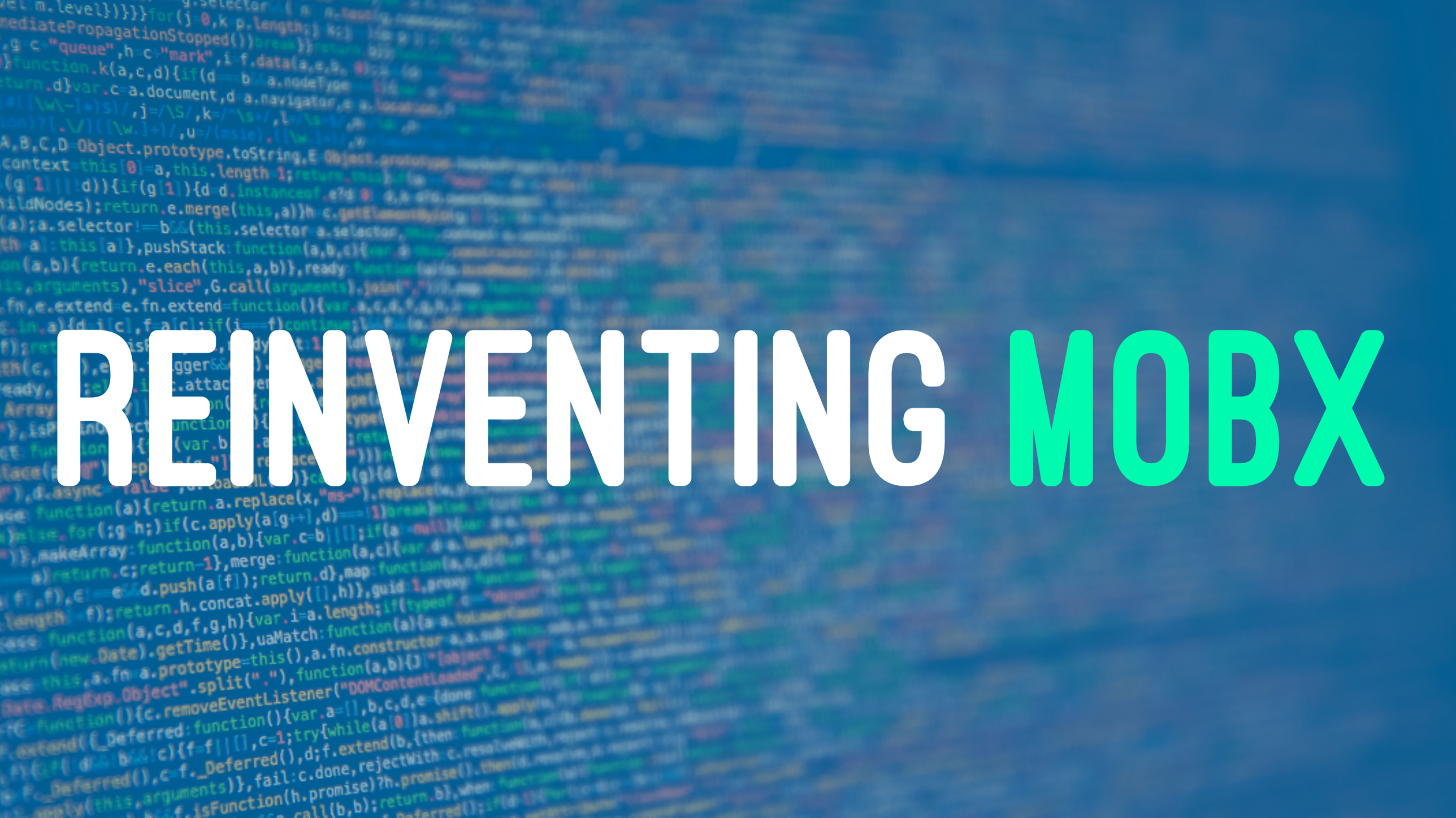
HERE'S THE PLAN

1. REINVENTING **MOBX**
2. REINVENTING **RXJS**
3. **ALL** FOR **ONE** AND **ONE** FOR **ALL**

REINVENTING THE WEEL

BY

TAKING THINGS APART


REINVENTING MOBX

MOBX CODE

```
const { observable, autorun } = require('mobx');

const okComputer = observable({
    title: "OK Computer",
    year: 1997,
    playCount: 0
});

autorun(() => {
    console.log(`Ok Computer PlayCount: ${okComputer.playCount}`)
}); // Ok Computer PlayCount: 0

okComputer.playCount = 2;  // Ok Computer PlayCount: 2
okComputer.playCount = 20; // Ok Computer PlayCount: 20
```

MOBX CODE FIRST IMPRESSIONS

- **SYNTAX** IS CLOSE TO THE LANGUAGE
- NO EXPLICIT **SUBSCRIPTION**
- **TRANSPARENT** FUNCTIONAL REACTIVE PROGRAMMING

LET'S REINVENT **MOBX**

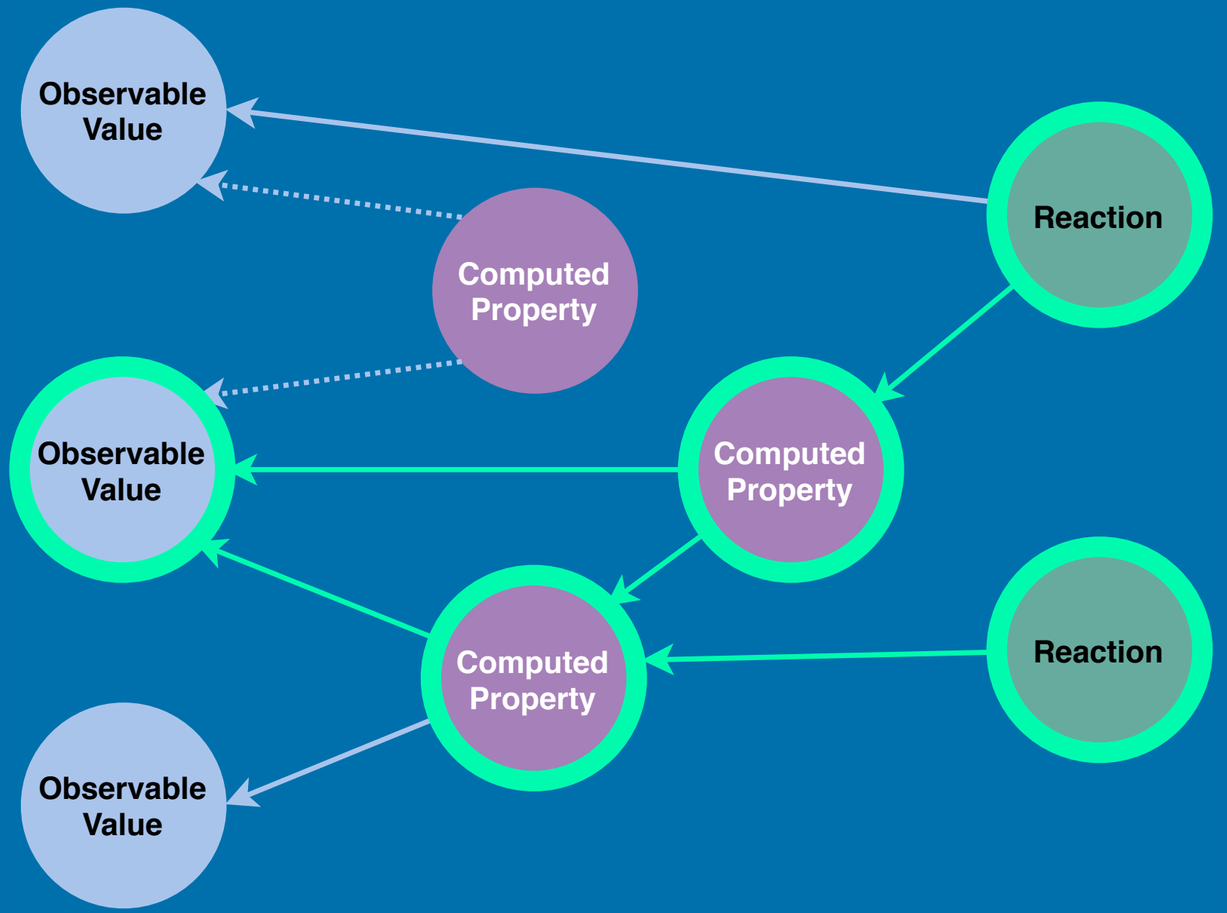
MOBX FROM THE INSIDE

- DOESN'T CARE ABOUT THE **PAST**
- ACT AS A **PROXY IN FRONT OF JAVASCRIPT**
- ALL REACTIONS ARE **SYNCHRONOUS**
- **DERIVATION GRAPH**

MOBX DEEP DIVE COMPUTED PROPERTIES

```
const { observable, autorun, computed } = require('mobx');

const okComputer = observable({
    title: "OK Computer",
    year: 1997,
    playCount: 0
});

const allInfo = computed(() => okComputer.title + okComputer.playCount);

autorun(() => { console.log(allInfo) }); // Ok Computer0

okComputer.playCount = 2;  // Ok Computer2
okComputer.playCount++;    // Ok Computer3
```

MOBX DEEP DIVE DERIVATION GRAPH




REINVENTING RXJS

RXJS CODE

```
const { from } = require('rxjs');
const { map, filter } = require('rxjs/operators');

const observable = from([1, 2, 3, 4, 5])
    .pipe(
        map(x => x + 1),
        filter(x => x % 2 === 0),
        map(x => x - 1),
    );

observable.subscribe(
    val => console.log('odd: ', val),
    error => console.error(error),
    () => console.log('Completed!'),
);
// odd: 1, odd: 3, odd: 5, Completed!
```

RXJS CODE FIRST IMPRESSIONS

- **SYNTAX** IS LESS FAMILIAR THAN MOBX
- **EXPLICIT SUBSCRIPTION**
- **OBSERVABLE** TC39 STAGE 1
- **PIPELINE OPERATOR** TC39 STAGE 1

LET'S REINVENT **RXJS**

RXJS FROM THE INSIDE

- MADE OF **REUSABLE PARTS** > **STREAMS**
- **CUSTOM OPERATORS**
- **LAZY EVALUATION**
- **SYNCHRONOUS** BY DEFAULT > **SCHEDULERS**

RXJS DEEP DIVE SCHEDULERS

SCHEDULERS IN RXJS ARE THINGS THAT CONTROL THE ORDER OF EVENT EMISSIONS (TO OBSERVERS) AND THE SPEED OF THOSE EVENT EMISSIONS.

– ANDRÉ STALTZ

QUEUE / ASAP / ASYNC / ANIMATIONFRAME / VIRTUALTIME


ALL FOR ONE AND ONE FOR ALL

SIDE BY SIDE

	PARADIGM	EXECUTION	SYNTAX	OBSERVABLES
MOBX	**TRANSPARENT** FUNCTIONAL REACTIVE PROGRAMMING	SYNCHRONOUS	PLAIN JAVASCRIPT	OBSERVABLE VALUES
RXJS	**EVENT STREAM** FUNCTIONAL REACTIVE PROGRAMMING	SYNCHRONOUS & ASYNCHRONOUS	CUSTOM*	OBSERVABLE EVENTS

WHEN SHOULD I USE MOBX ?

- **LEARNING CURVE**
- **VALUES.** NOT EVENTS
- EASY REPRESENTATION OF **APPLICATION STATE**
- **STATE = DERIVATION (** PREVIOUS STATE **)**

WHEN SHOULD I USE RXJS ?

- EVENTS & VALUES
- WORK WITH TIME
- HEAVY I/O TASKS
- LOW-LEVEL CONTROL

WHEN SHOULD I USE BOTH ?

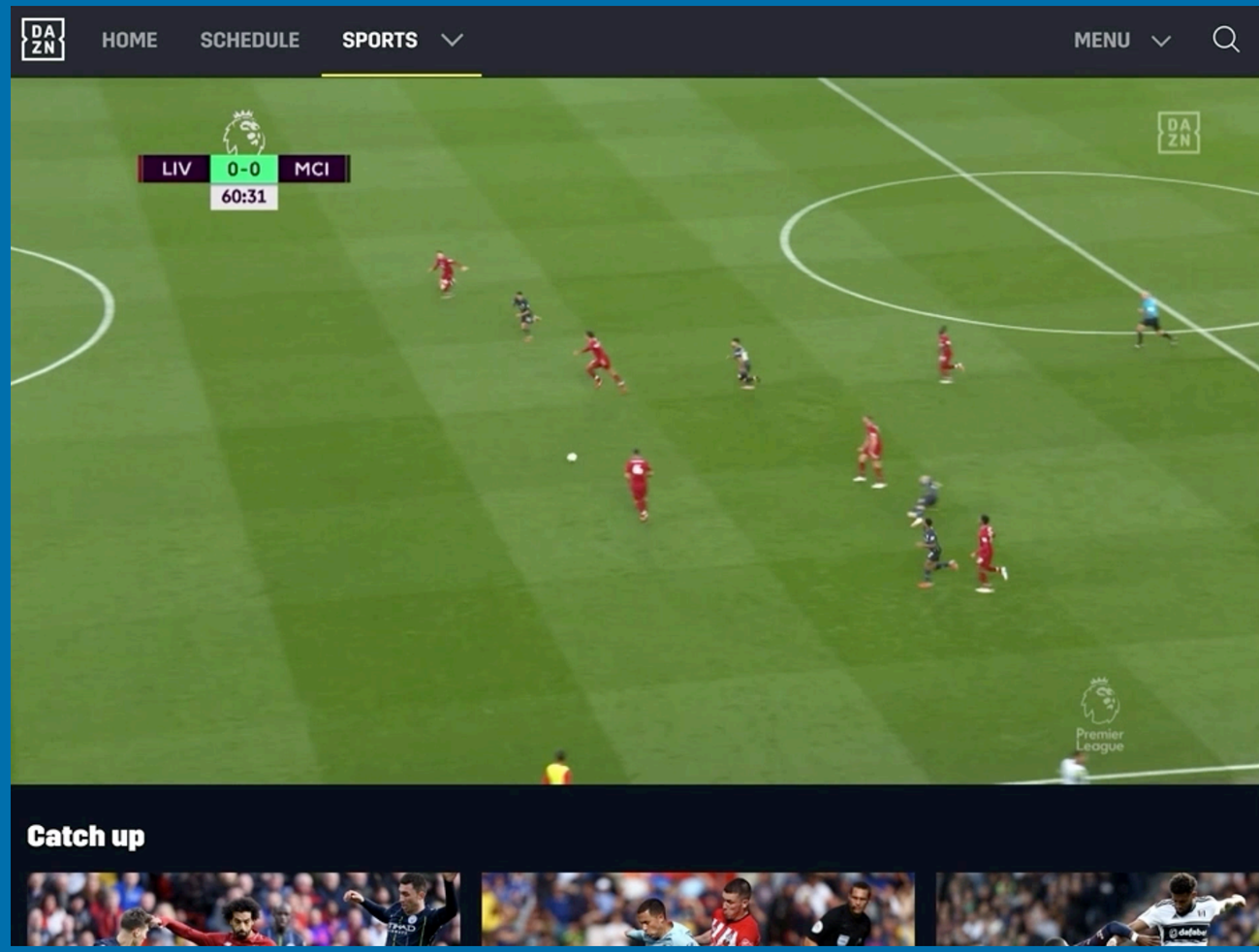
1. RXJS HANDLES AN **HEAVY TASK**
2. IT CHANGES THE **APPLICATION STATE.** MANAGED BY **MOBX**
3. **REACTION:** THE VIEW IS UPDATED

WHEN SHOULD I USE BOTH ?

REAL LIFE EXAMPLE

APPLICATION STATE > MOBX

SCROLL BASED ANIMATIONS > RXJS



THANK YOU

SLIDES **GITHUB.COM/MAXGALLO/TALK-RXJS-MOBX**

TWITTER **@_MAXGALLO**
OTHER **MAXGALLO.IO**