RxJS - Advanced Patterns:

Operating Heavily Dynamic UIs

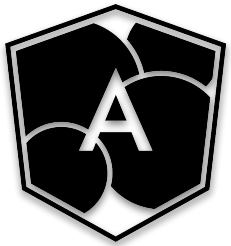
Agenda

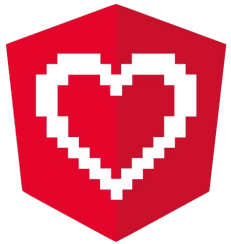
- The Problem
- Reactive Micro Architecture
- Event Sourcing, CQRS and their relation
- Orchestrate rendering and UI interaction
- Where and when to optimize performance

Angular by heart and code

Development, Workshops, Community

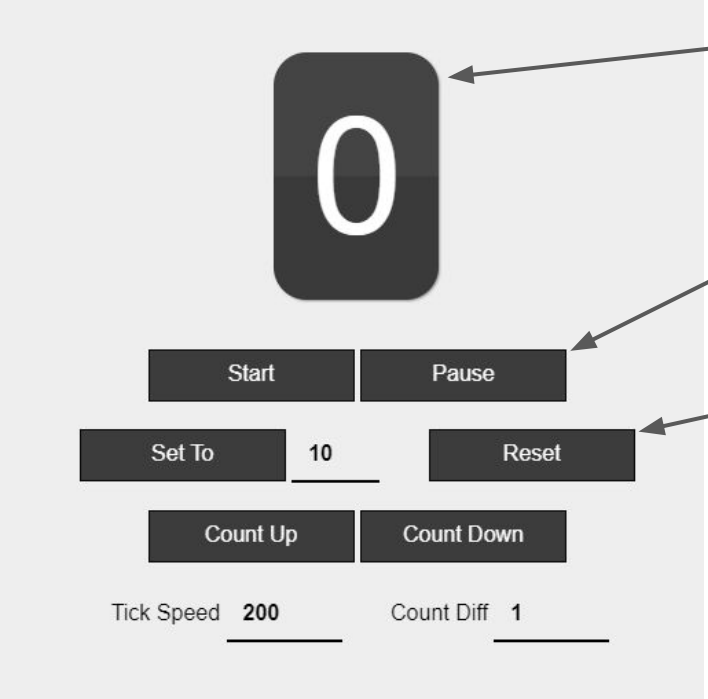
Michael Hladky




The Problem

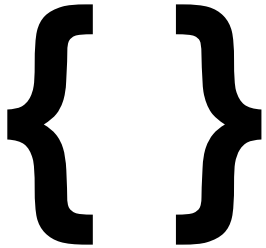
The Problem



- State

- Background processes

- User interactions

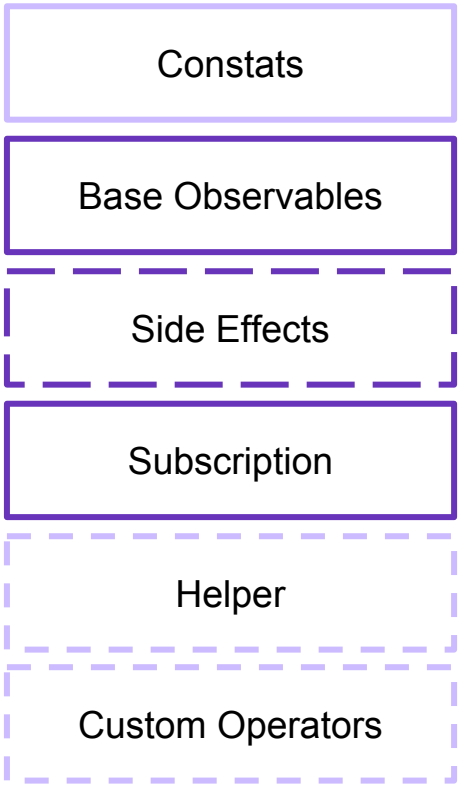


Showcase Problem

Micro Architecture

Architecture Sections

C O E S H P

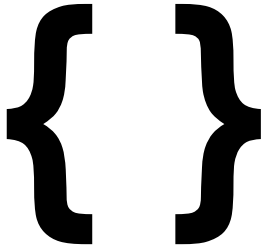


Divide code into 6 main groups

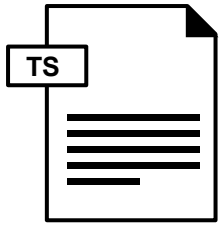
Structure code with this groups makes code
- maintainable
- extensible
- easy to orientate

Constants

C O E S H P



Static data i.e. JSON files
Constants i.e. interval rate
UI elements i.e. Elem. ref. to button



In **some cases** a you will extract these things into a **separate file**.

Base Observables

C O E S H P

Source Observables

Interaction Observables

State Observables

Intermediate Observables

Source observables are the purest observables in you architecture. Here we separate into state and interaction.

Interaction observables are mostly UI related. (i.e. btn click) Could be abstracted into a component.

State observables represent the state of your application. In most casts it is in a separate file.

Intermediate observables are a combination of state and interaction observables.

Side Effects

C O E S H P

UI Updates

UI Interaction

Background Processes

UI Input are all observables that trigger i.e. a renderView() function.

UI Outputs are all events from user interaction that trigger something else.

All actions triggered from automated processes. (i.e. intervals, http, web-socket msg's)

Subscriptions

C O E S H P

```
merge([
    renderValue$,
    updateState$
])

.pipe(
  takeUntil(trigger$)
)

.subscribe();



{{ state$ | async }}
```

Separate subscriptions into inputs and outputs.

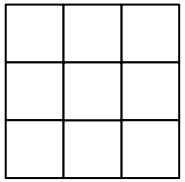
Subscription handling should be done declarative.
i.e. takeUntil

In best case we maintain only a single subscription.

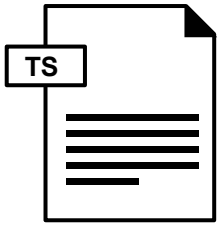
Some frameworks even take over subscription handling for us.

Helper

C O E S H P



Functions that perform common, often reused logic.



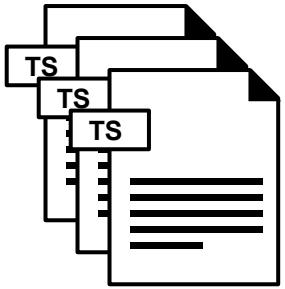
In many cases a you will extract these functions into a separate file.

Custom Operators

C O E S H P

Creation methods

Operators



Creation methods are all functions that return a new observable.

Operators are all functions that take an observable and return an observable.

In most cases a you will extract these functions
into separate files.

{ }

Implement Micro Architecture

Event Sourcing

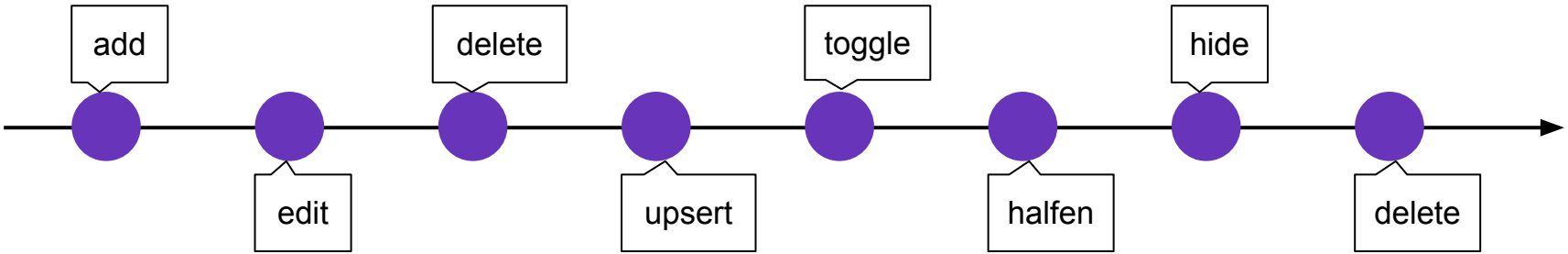
“ ***Event Sourcing:***
Capture all changes to an application state as a sequence of events. ”

Martin Fowler

Event Sourcing

Modeling state changes as an **immutable sequence** of **events**.

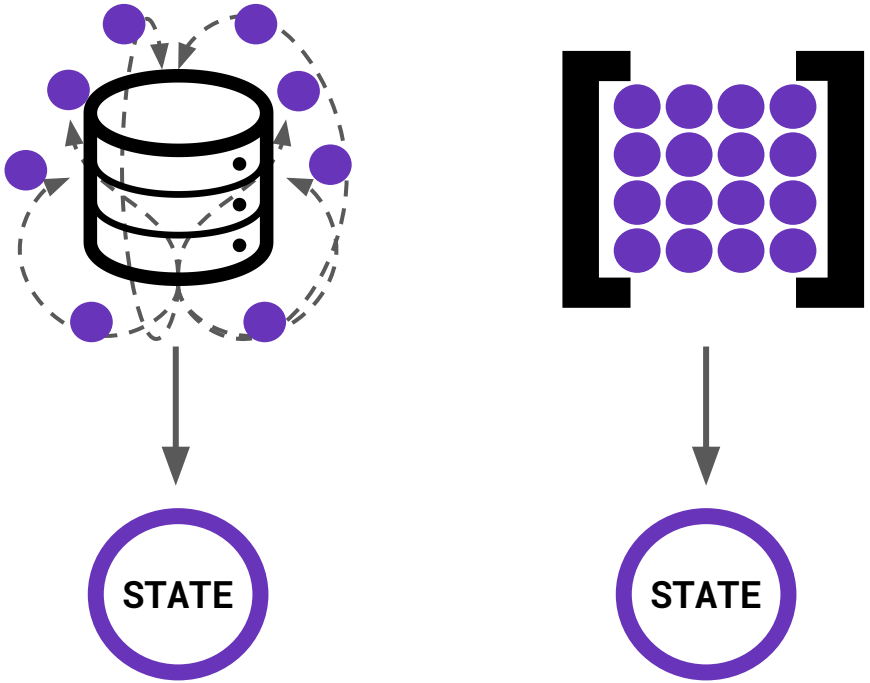
Every event describes it's changed to the state.



Event Sourcing

Instead of mutating the state,

derive (query) it from the
immutable sequence of changes



Command Query Responsibility Segregation (CQRS)

CQRS provides separation of concerns for reading and writing.

CQRS:

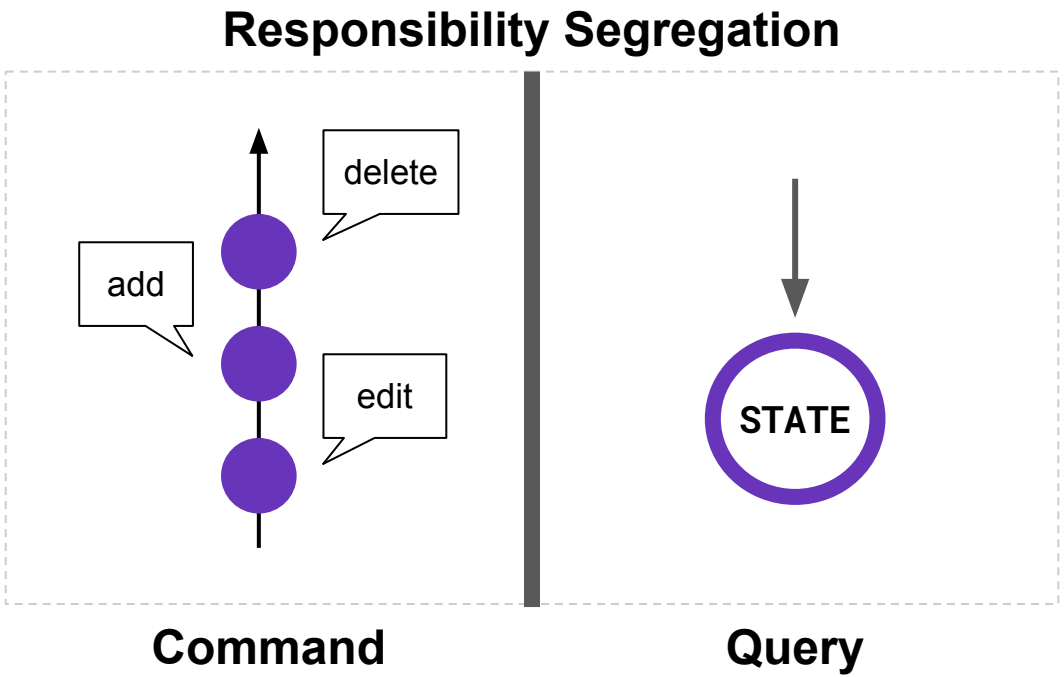
Every method should either be a command, or a query, but not both.

Bertrand Meyer

CQRS

Separating an application **responsibilities** into two parts:

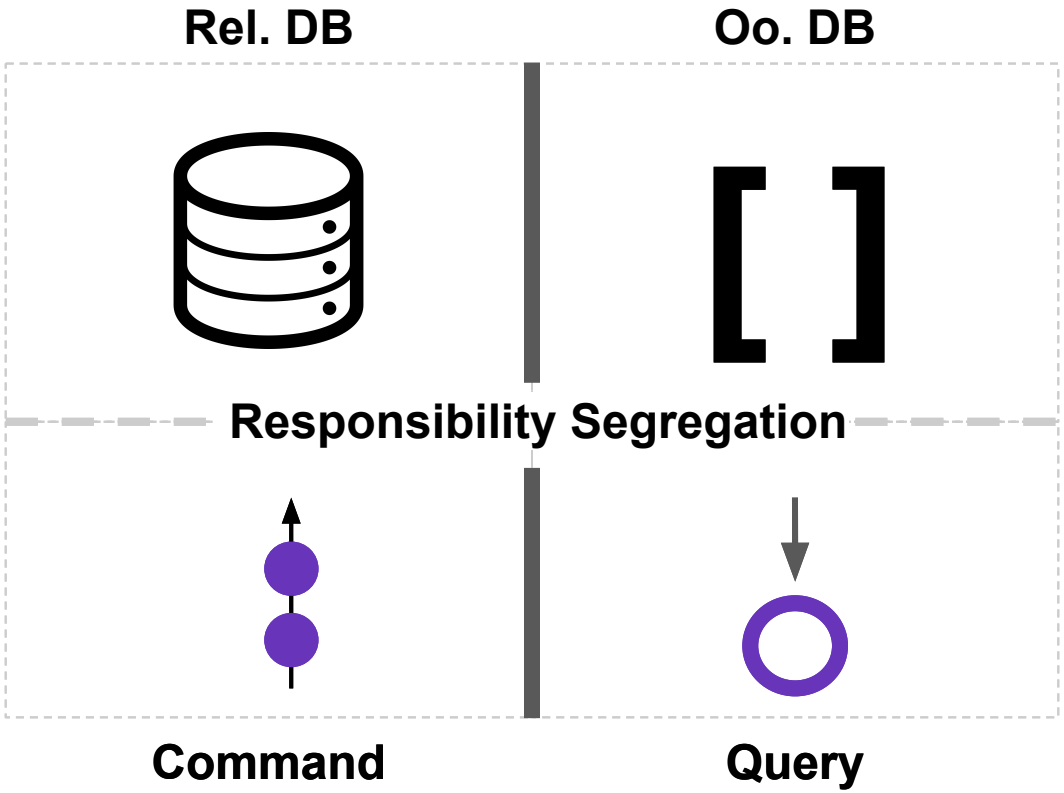
- The **command** side witch update state
- The **query** side which reads state



CQRS

Enables a combination of i.e:

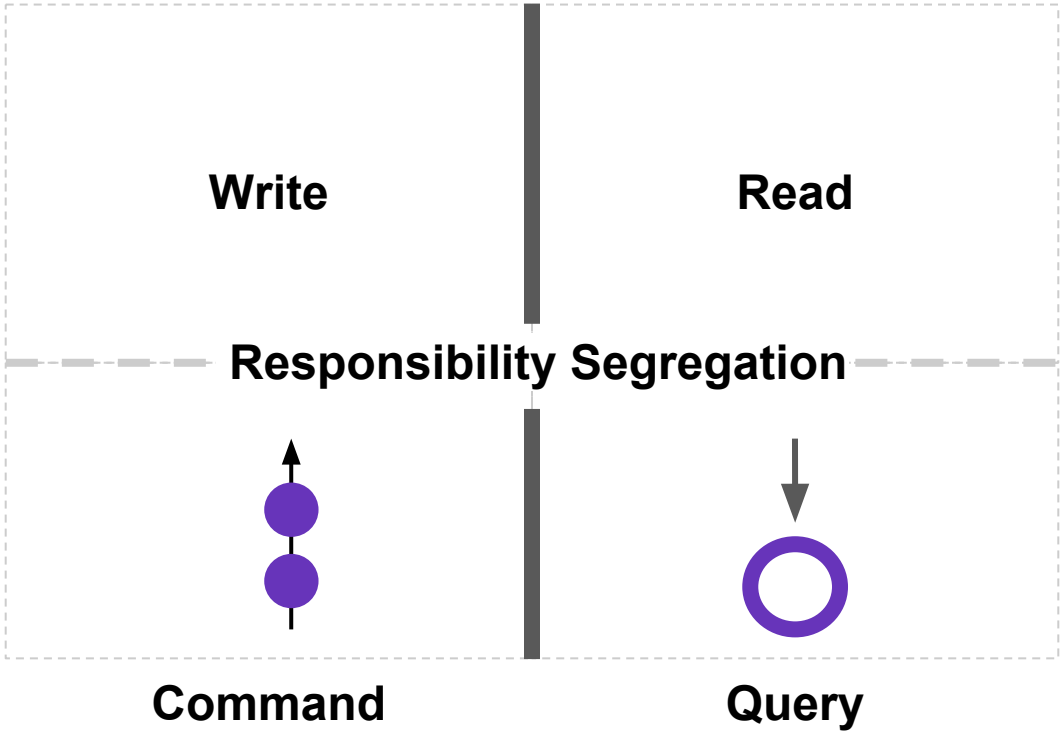
- normalization (**faster writes**)
- denormalization (**faster reads**)

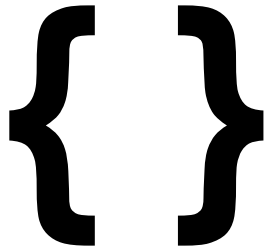


CQRS

Let's **apply it to** the **frontend** and by

separating writing and **reading** strictly





Separate
State Management and Side Effects

Orchestrate UI interaction and rendering

Orchestrate rendering and UI interaction

- First render
- Than interaction

Where and when
to optimize performance

Where and when to optimize performance

- Do it **at the end** of your task
- Before trigger a **render side effect**
- Use the **AnimationFrame**Scheduler
- **Sample** frequent commands
- **debounce** typing
- Use **standard operators** to work with **arrays**

Thanks for your time

I'm Michael,
If you have any questions
just ping me!

Michael Hladky