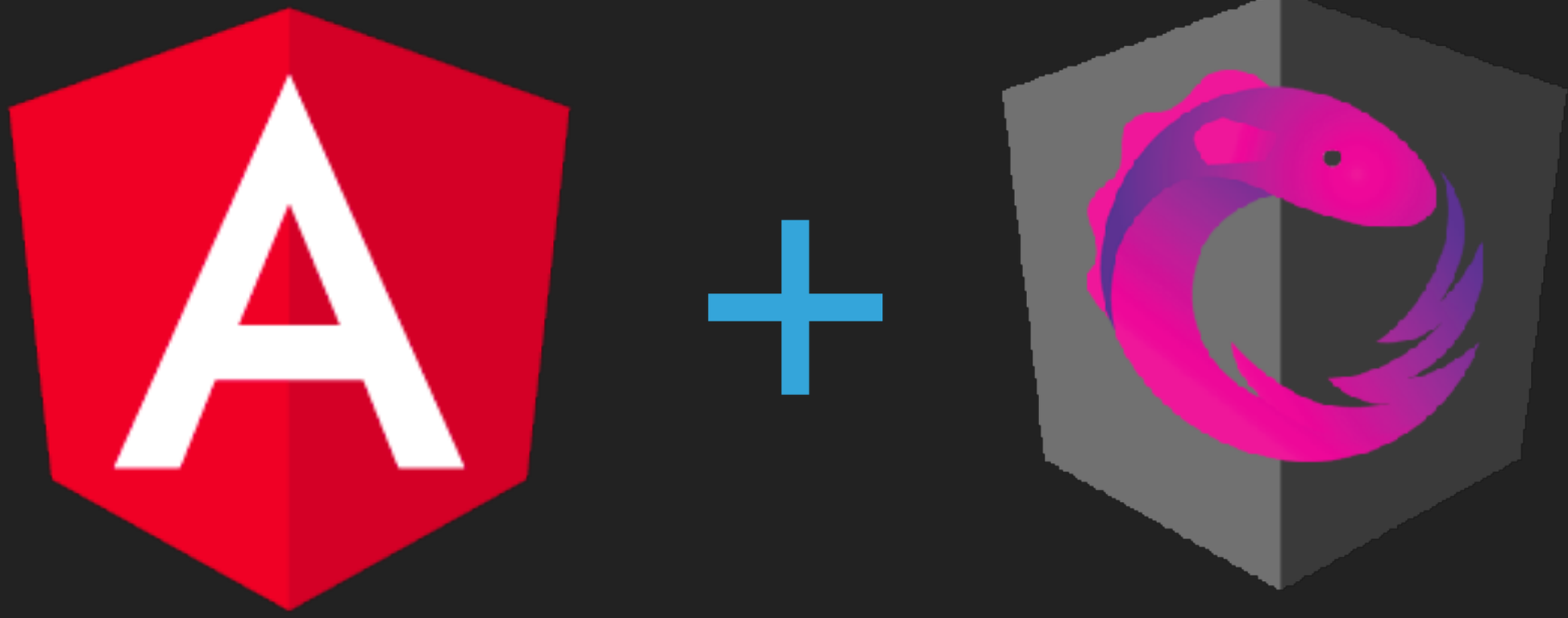


IMPLEMENTING THE REDUX PATTERN TO MAKE SCALABLE APPS

ANGULAR + NGRX

ANGUAR

(NOT TO BE CONFUSED WITH ITS EVIL OLDER SIBLING ANGULARJS)

What is Angular

ANGULAR VERSION HISTORY

- AngularJS / Angular 1.x.x
- Angular 2 : Complete rewrite, breaking change
- Angular 3: Skipped since @angular/router was ahead
- Angular 4
- Angular 5: Current Stable Release

WHY ANGULAR

ADVANTAGES

- Front end decoupled from back end code: Replaces server side templating engines plus JQuery approach (Java - JSTL, .Net Razor, Python Jinja)

- Code Organization (Components) & Productivity: More Declarative than Imperative

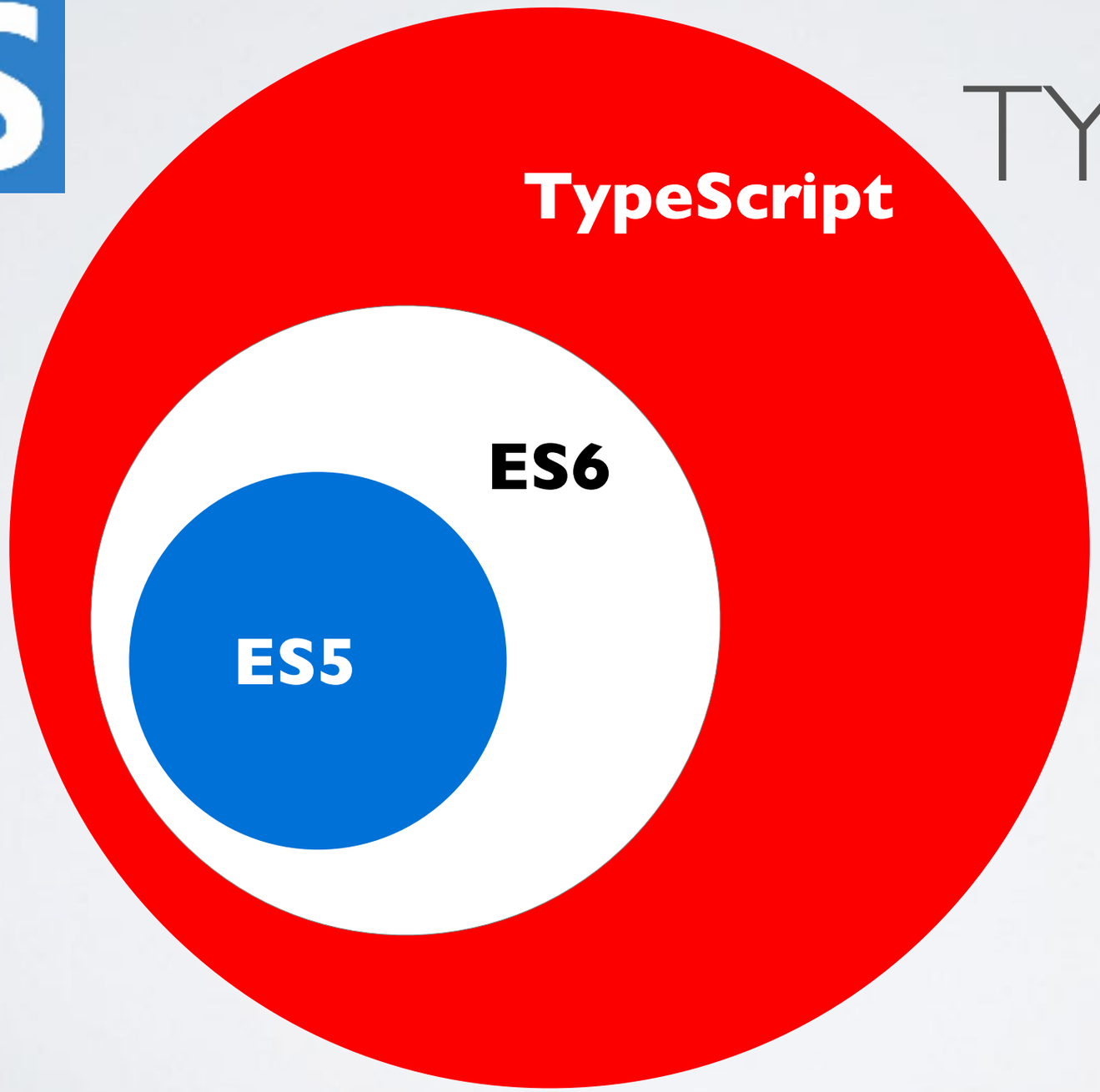
- Rapid Development & CLI code generation

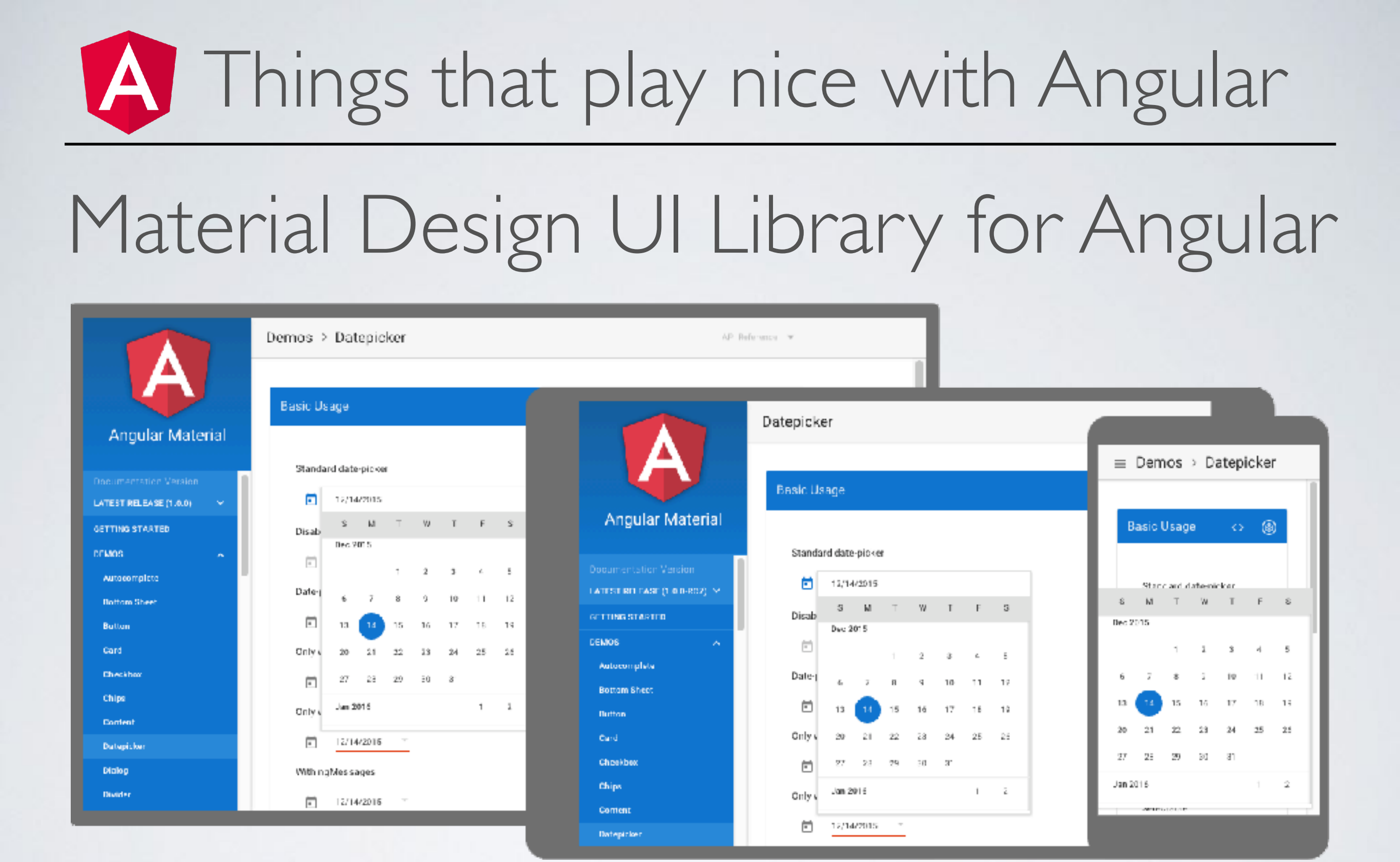
- Dynamic Content in HTML template

- Unit Testing Ready

- Cross Platform (browsers, device)

WHAT IS TYPESCRIPT?





Things that play nice with Angular

Material Design UI Library for Angular

material.angular.io

Things that play nice with Angular

AngularFire

github.com/angular/angularfire2

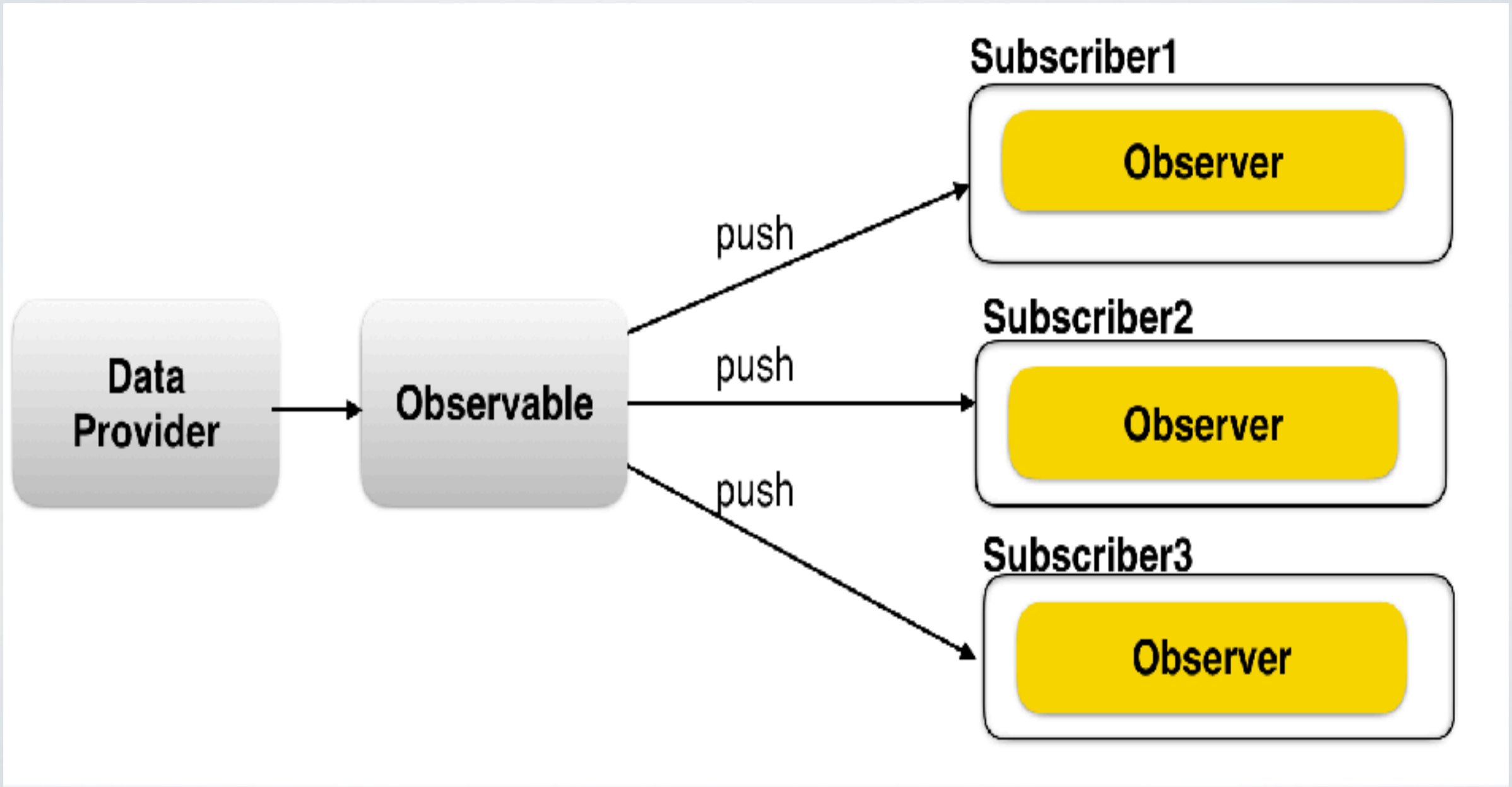
RFP - Observable Streams

ReactiveX

Javascript implementation: Rxjs

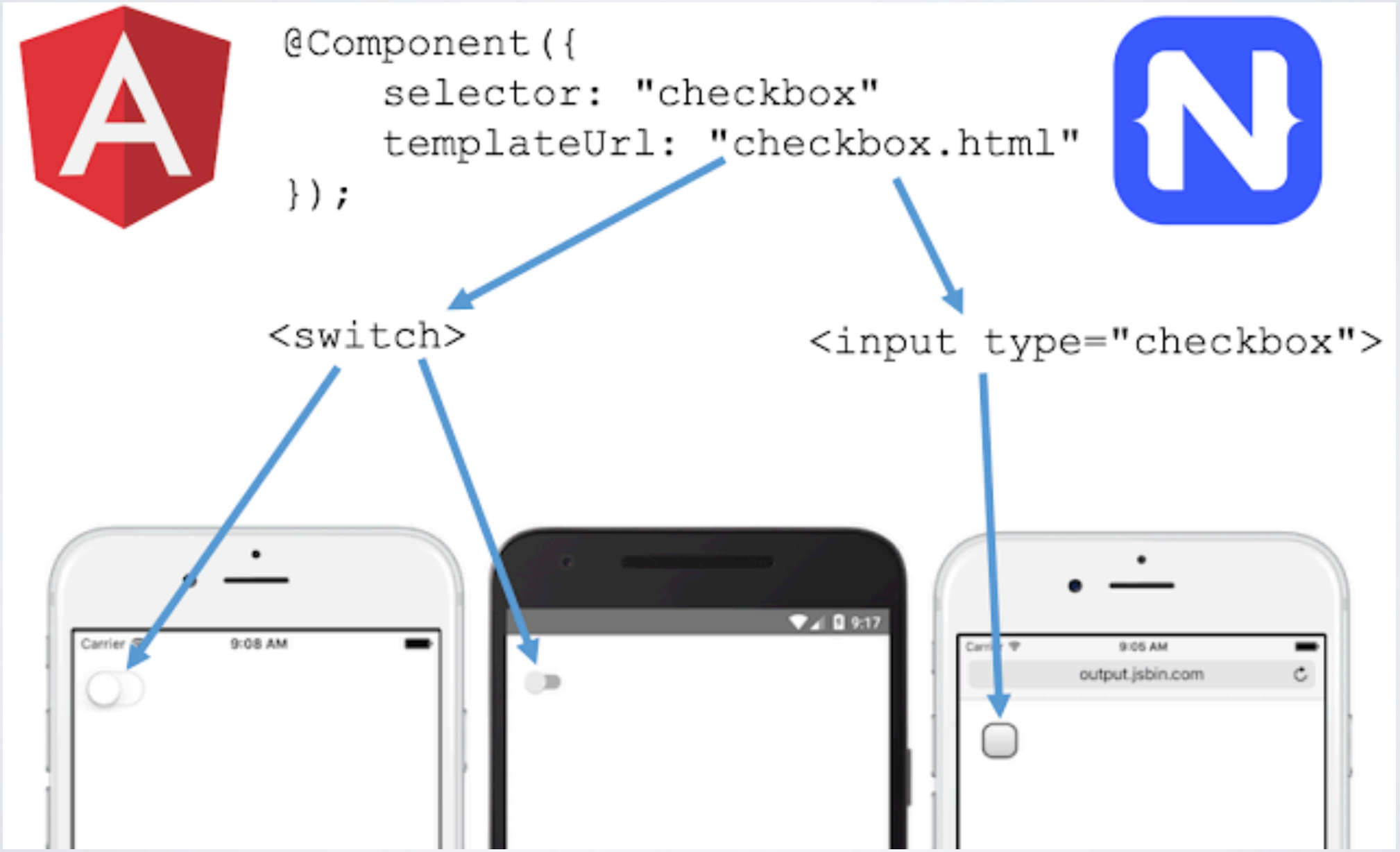
http://reactivex.io/rxjs/

Observable Streams



Things that play nice with Angular

Angular + NativeScipt = Cross Mobile



Redux Pattern

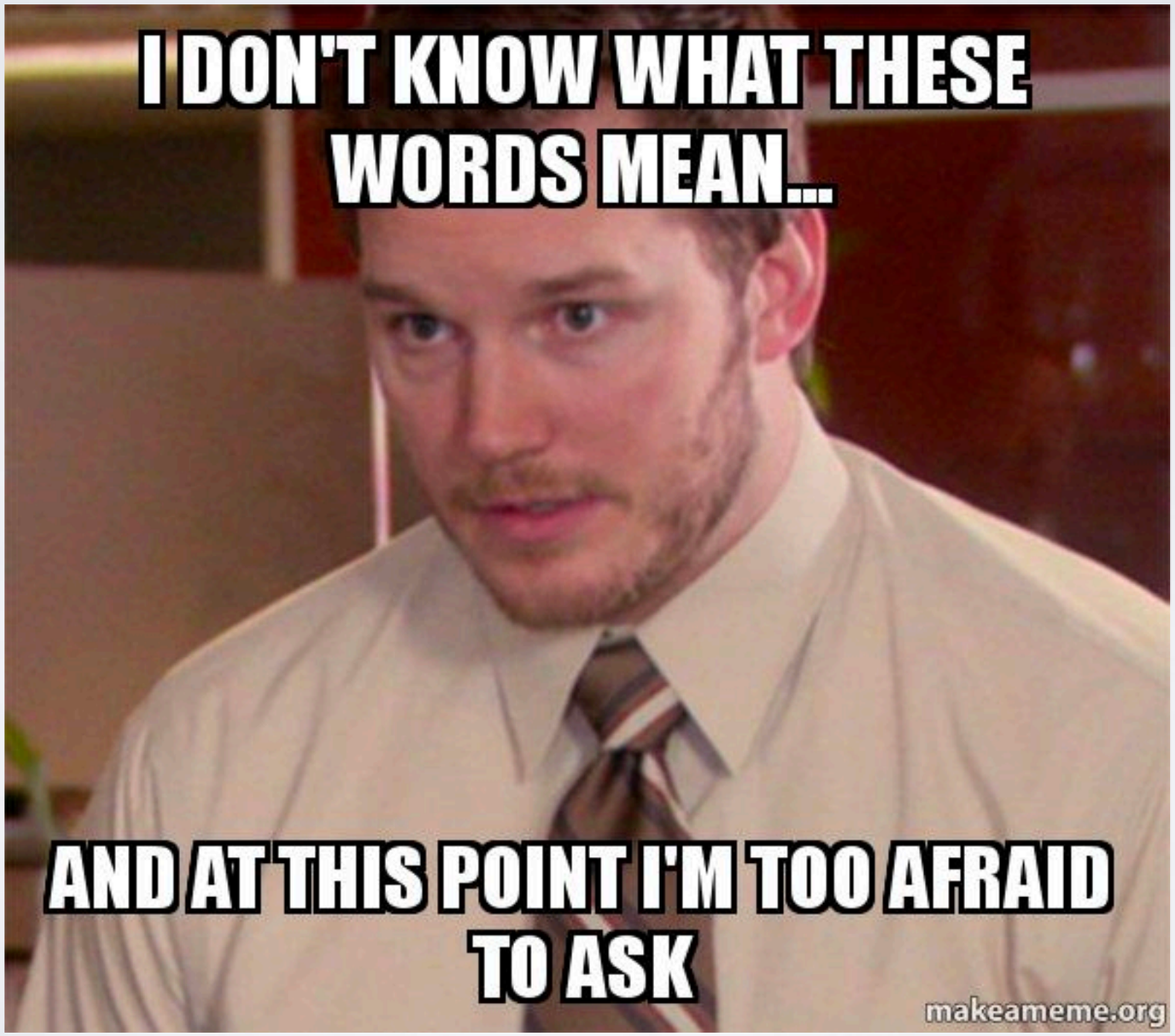
Ngrx:

"RxJS powered state management for Angular applications, inspired by Redux"

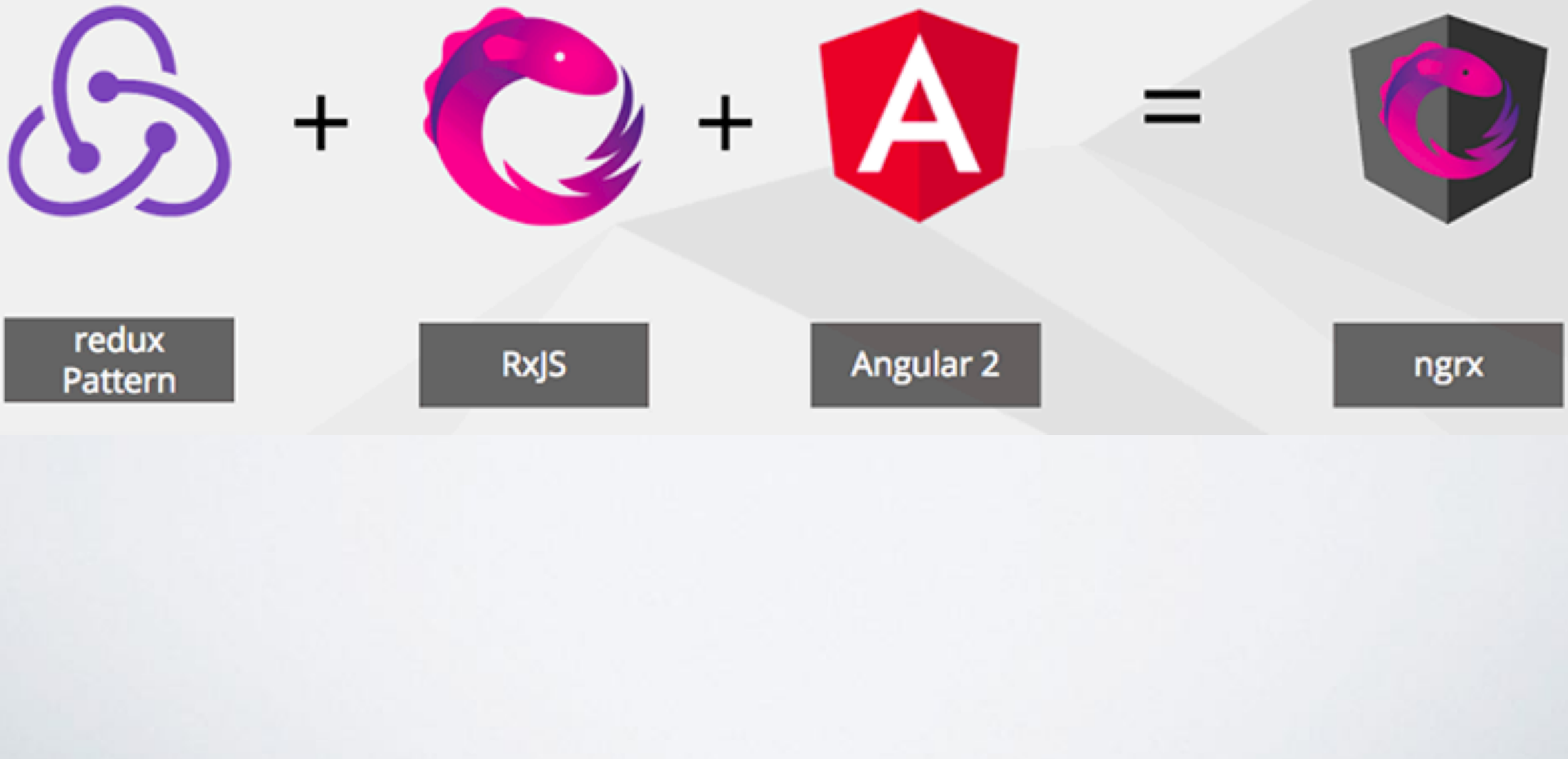
Redux Pattern

Ngrx:

"RxJS powered state management for Angular applications, inspired by Redux"



ngrx supercharges the redux pattern with RxJS



redux Pattern + RxJS + Angular 2 = ngrx

NGRX MIGHT BE THE RIGHT TOOL IF...

NGRX MIGHT BE THE RIGHT TOOL IF . . .

- User workflows are complex

NGRX MIGHT BE THE RIGHT TOOL IF . . .

- User workflows are complex
- Your app has a large variety of user workflows (ex: both regular users and administrators)

NGRX MIGHT BE THE RIGHT TOOL IF . . .

- User workflows are complex
- Users can collaborate
- Your app has a large variety of user workflows (ex: both regular users and administrators)

NGRX MIGHT BE THE RIGHT TOOL IF . . .

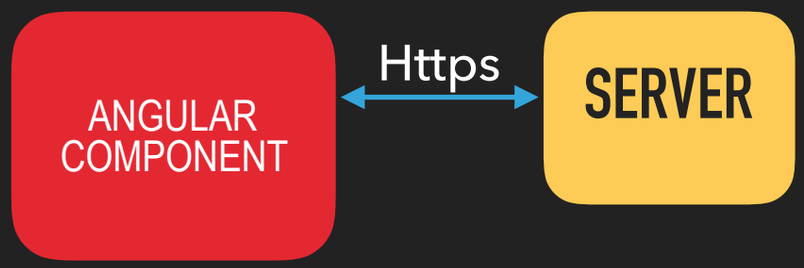
- User workflows are complex
- Your app has a large variety of user workflows (ex: both regular users and administrators)
- Users can collaborate
- You're using web sockets or Server Sent Events

NGRX MIGHT BE THE RIGHT TOOL IF . . .

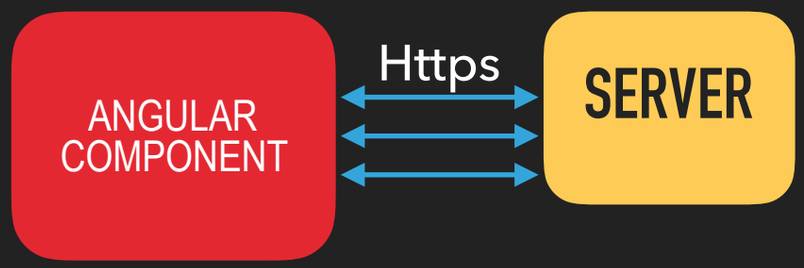
- User workflows are complex
- Your app has a large variety of user workflows (ex: both regular users and administrators)
- Users can collaborate
- You're using web sockets or Server Sent Events
- You're loading data from multiple endpoints to serve a single view

THE PROBLEM

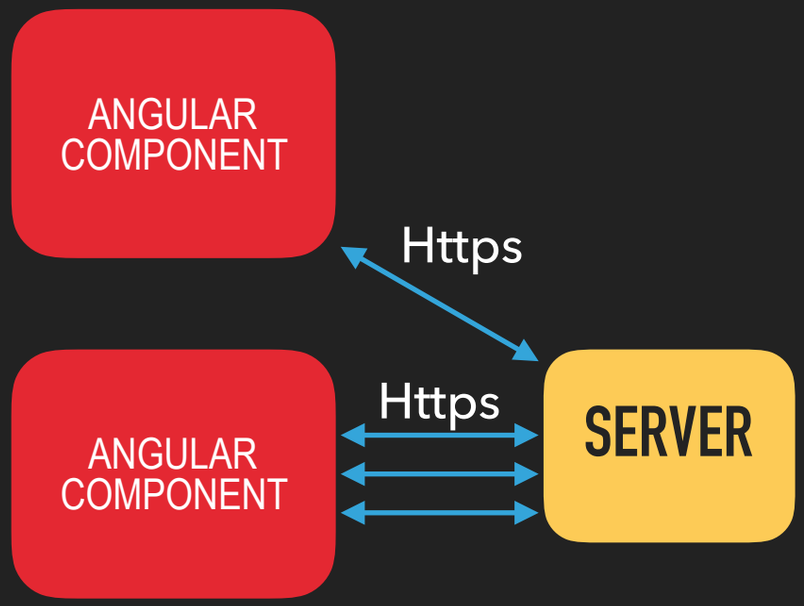
THE PROBLEM



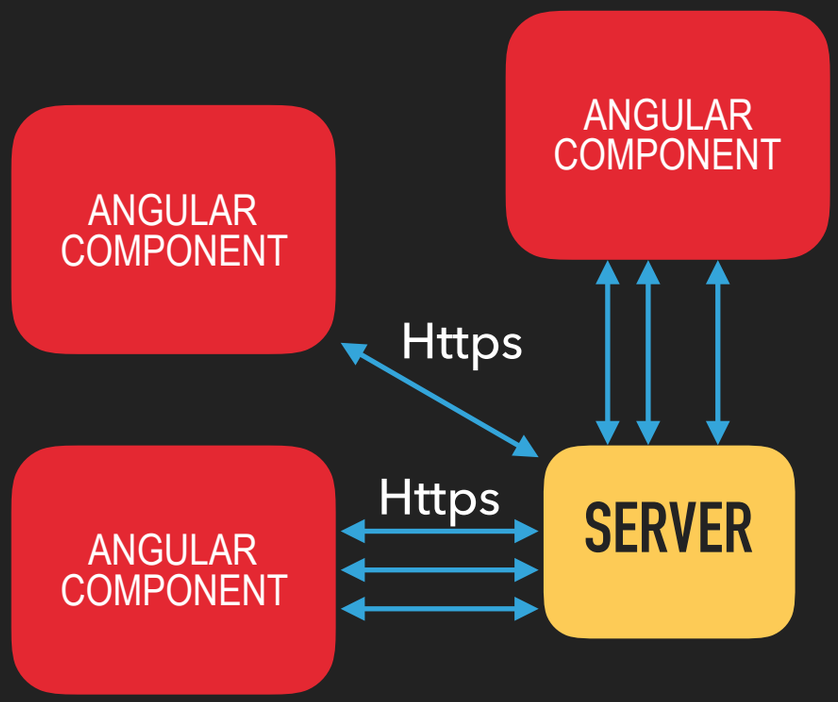
THE PROBLEM



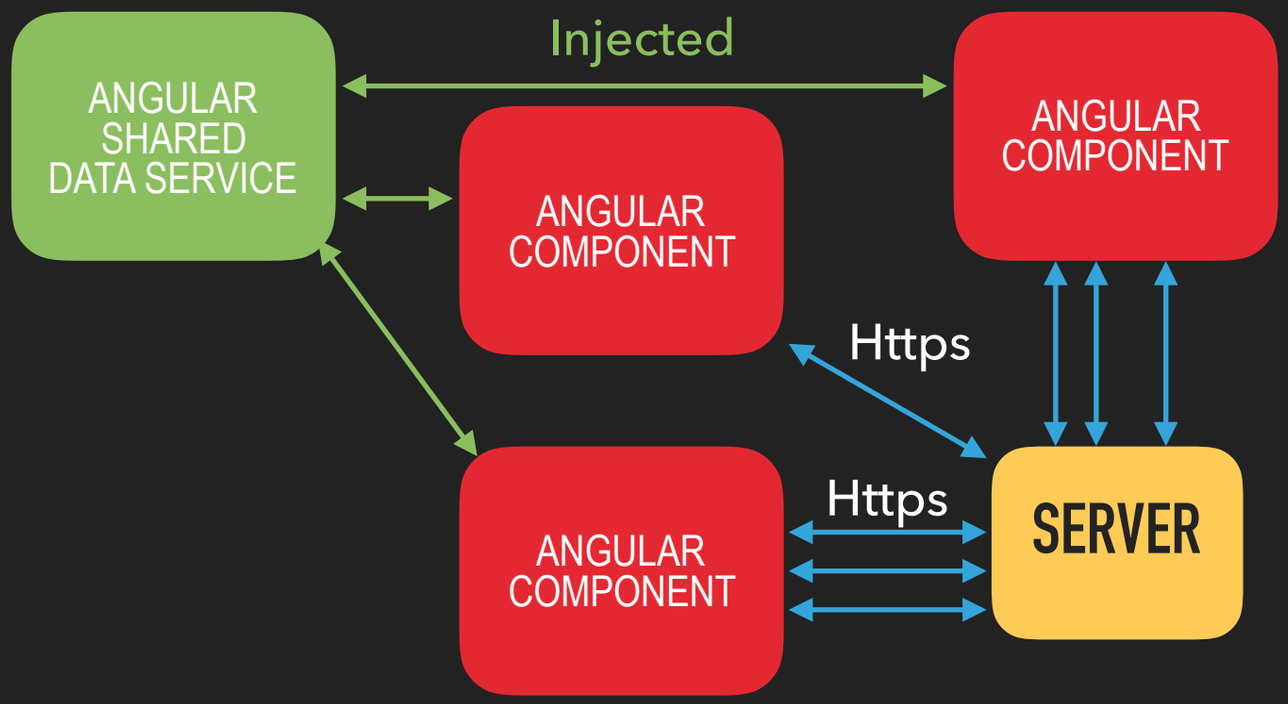
THE PROBLEM



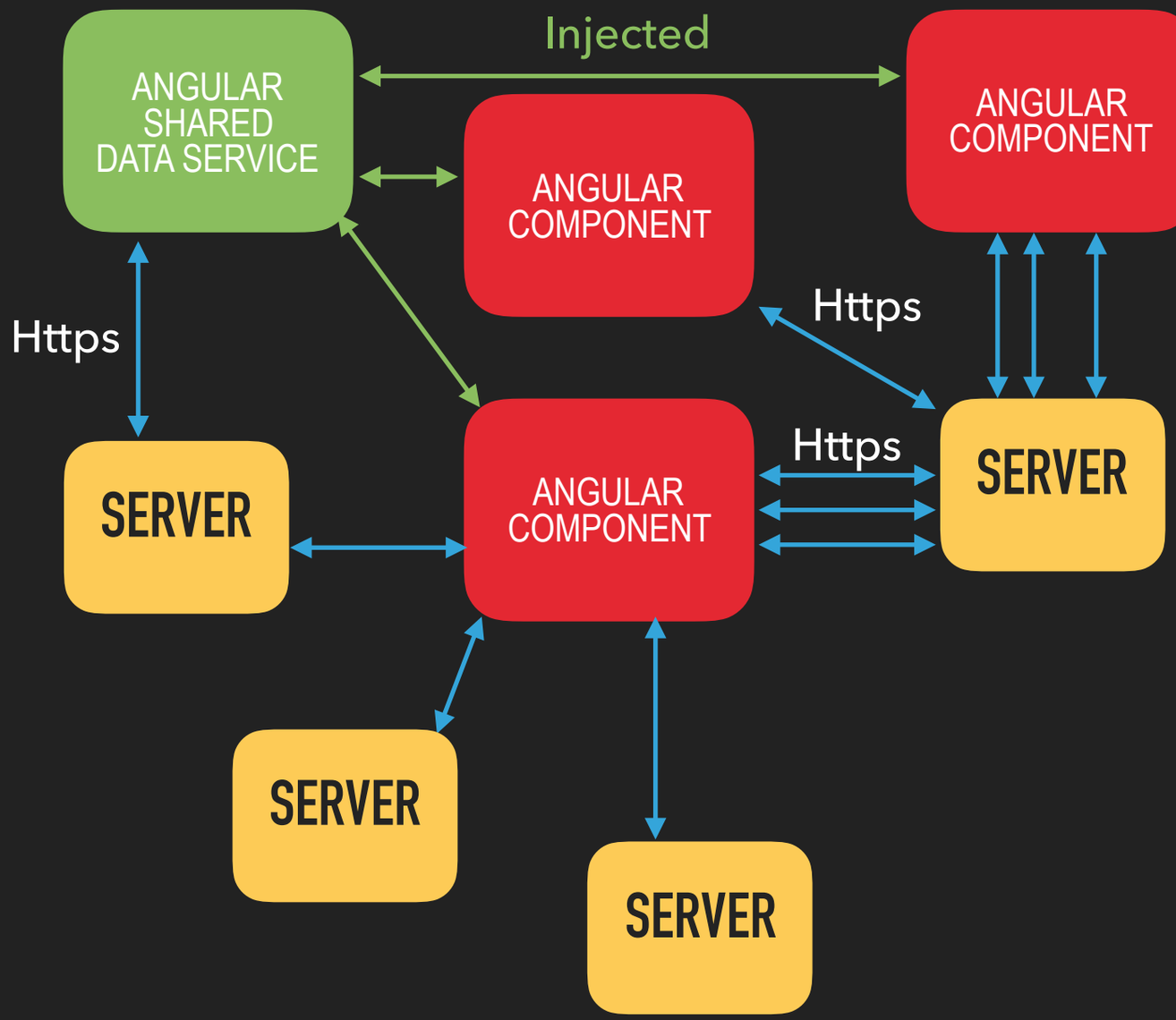
THE PROBLEM


ANGULAR COMPONENT
ANGULAR COMPONENT
Https
Https
ANGULAR COMPONENT
SERVER

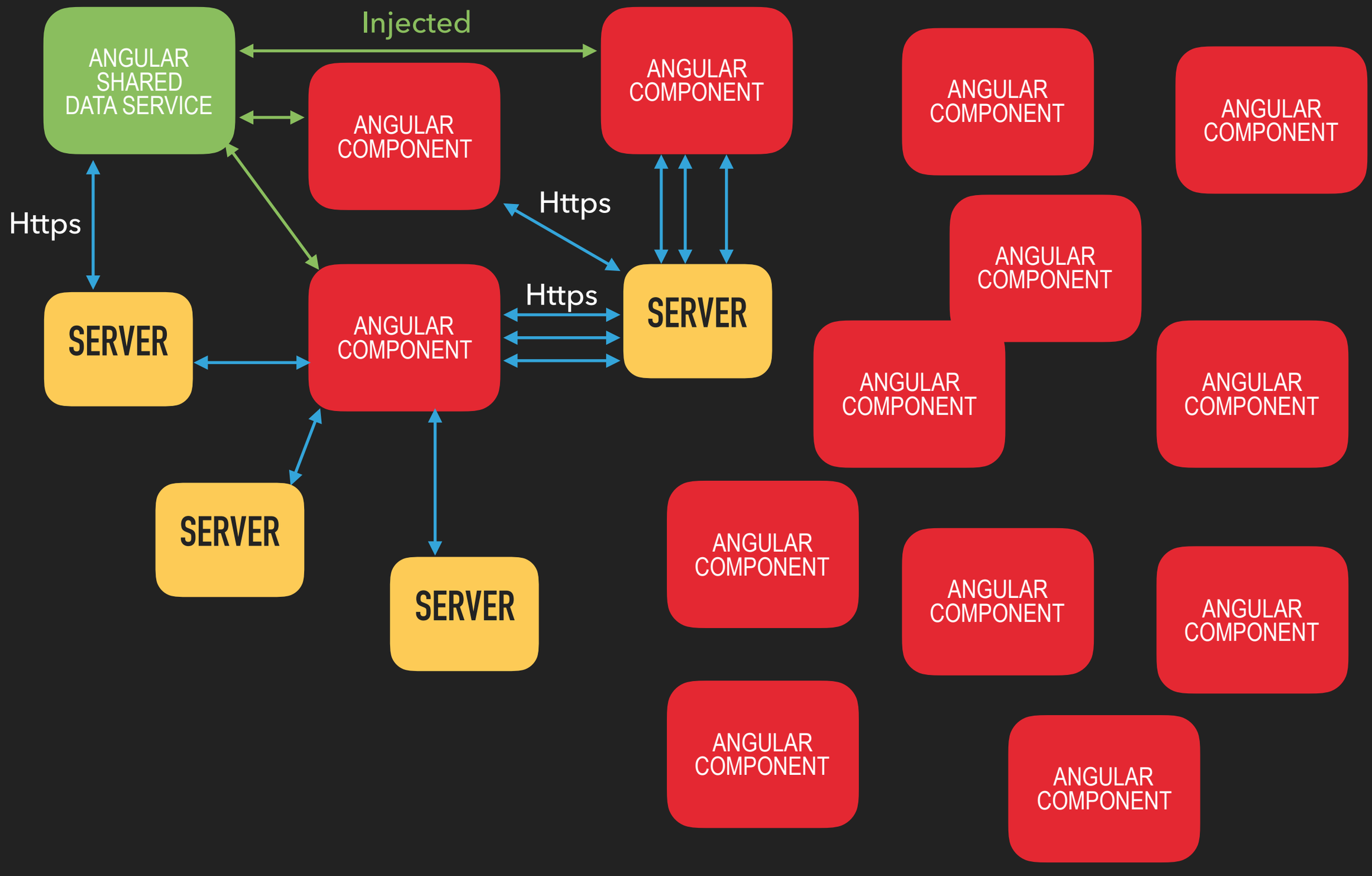
THE PROBLEM


Injected
ANGULAR SHARED DATA SERVICE
ANGULAR COMPONENT
ANGULAR COMPONENT
Https
ANGULAR COMPONENT
Https
SERVER

THE PROBLEM


Injected
ANGULAR SHARED DATA SERVICE
ANGULAR COMPONENT
ANGULAR COMPONENT
Https
Https
Https
SERVER
ANGULAR COMPONENT
SERVER
SERVER
SERVER

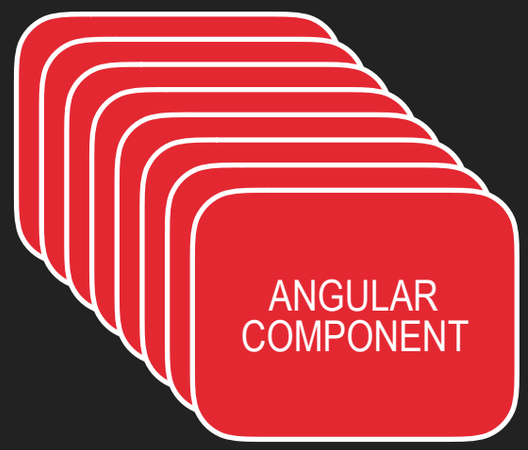
THE PROBLEM


ANGULAR SHARED DATA SERVICE
Injected
ANGULAR COMPONENT
ANGULAR COMPONENT
ANGULAR COMPONENT
ANGULAR COMPONENT
ANGULAR COMPONENT
Https
Https
Https
SERVER
SERVER
ANGULAR COMPONENT
SERVER
SERVER
ANGULAR COMPONENT
ANGULAR COMPONENT
ANGULAR COMPONENT
ANGULAR COMPONENT
ANGULAR COMPONENT
ANGULAR COMPONENT
ANGULAR COMPONENT

THE SOLUTION:  DATA FLOW

THE SOLUTION:

DATA FLOW

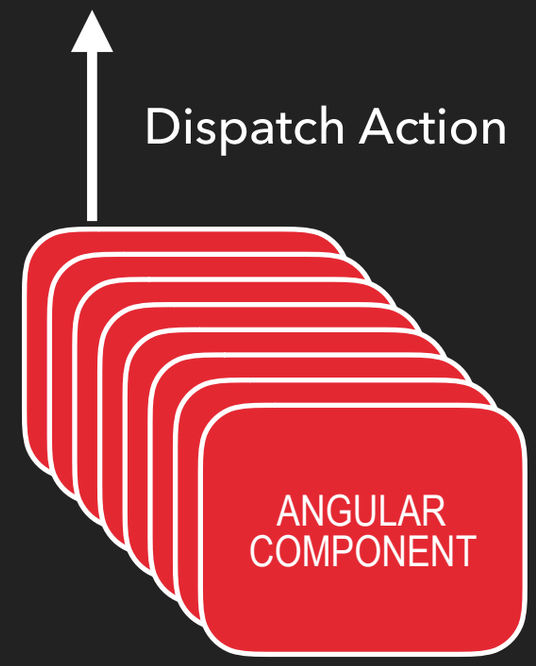


THE SOLUTION:  DATA FLOW

ACTION

{ type: addToDo

payload: ski }



THE SOLUTION:

DATA FLOW

@ngrx/STORE

Action Stream

ACTION
{ type: addToDo
payload: ski }

Dispatch Action

ANGULAR COMPONENT

THE SOLUTION:

DATA FLOW

@ngrx/STORE

Action Stream

REDUCER

(state, action) =>
state

ACTION

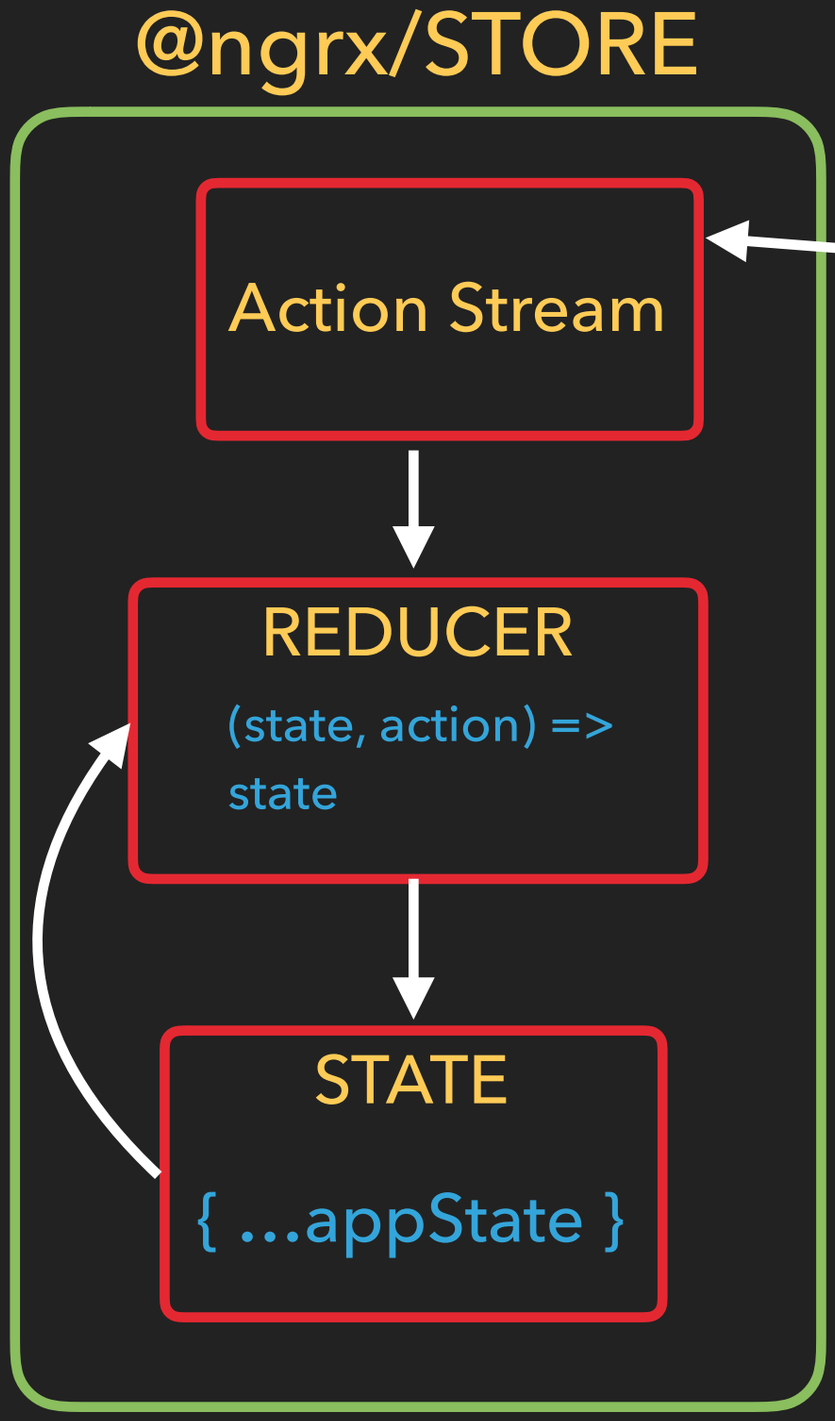
{ type: addToDo
payload: ski }

Dispatch Action

ANGULAR COMPONENT

THE SOLUTION:

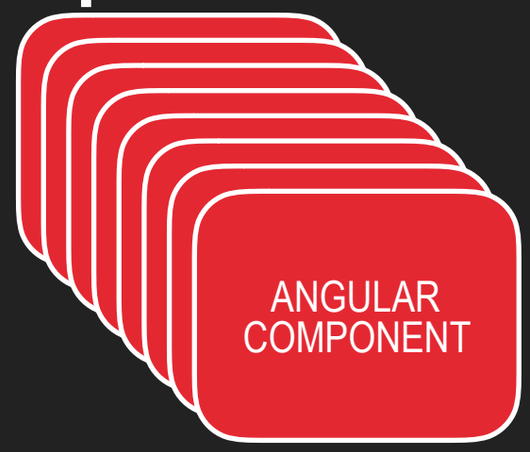
DATA FLOW



ACTION

{ type: addToDo
payload: ski }

Dispatch Action



THE SOLUTION:

DATA FLOW

@ngrx/STORE

Action Stream

REDUCER

(state, action) =>
state

STATE

{ ...appState }

ACTION

{ type: addToDo
payload: ski }

Dispatch Action

ANGULAR COMPONENT

Components
Subscribes to
State

THE SOLUTION:

DATA FLOW

@ngrx/STORE

Action Stream

REDUCER

(state, action) =>
state

STATE

{ ...appState }

@ngrx/
Effects

ACTION

{ type: addToDo
payload: ski }

Dispatch Action

ANGULAR
COMPONENT

Components
Subscribes to
State

THE SOLUTION:

DATA FLOW

@ngrx/STORE

Action Stream

REDUCER

(state, action) => state

STATE

{ ...appState }

@ngrx/ Effects

Https

SERVER

ACTION

{ type: addToDo payload: ski }

Dispatch Action

ANGULAR COMPONENT

Components Subscribes to State

THE SOLUTION:

DATA FLOW

@ngrx/STORE

Action Stream

REDUCER

(state, action) =>
state

STATE

{ ...appState }

@ngrx/
Effects

Https

SERVER

ACTION

{ type: addToDo
payload: ski }

Dispatch
Action

Dispatch Action

ANGULAR
COMPONENT

Components
Subscribes to
State

BENEFITS OF THE REDUX PATTERN

THESE ARCHITECTURAL DECISIONS:

PROVIDE THESE BENEFITS:

BENEFITS OF THE REDUX PATTERN

THESE ARCHITECTURAL DECISIONS:

- Single Source of Truth: The Store - centralized client side state

PROVIDE THESE BENEFITS:

BENEFITS OF THE REDUX PATTERN

THESE ARCHITECTURAL DECISIONS:

- Single Source of Truth: The Store - centralized client side state
- One Way Data Flow

PROVIDE THESE BENEFITS:

BENEFITS OF THE REDUX PATTERN

THESE ARCHITECTURAL DECISIONS:

- Single Source of Truth: The Store - centralized client side state
- One Way Data Flow
- State is only updated through pure functions (reducers)

PROVIDE THESE BENEFITS:

BENEFITS OF THE REDUX PATTERN

THESE ARCHITECTURAL DECISIONS:

- Single Source of Truth: The Store - centralized client side state
- One Way Data Flow
- State is only updated through pure functions (reducers)

PROVIDE THESE BENEFITS:

- Provides insight into race conditions bugs

BENEFITS OF THE REDUX PATTERN

THESE ARCHITECTURAL DECISIONS:

- Single Source of Truth: The Store - centralized client side state
- One Way Data Flow
- State is only updated through pure functions (reducers)

PROVIDE THESE BENEFITS:

- Provides insight into race conditions bugs
- Deterministic State Reproduction

BENEFITS OF THE REDUX PATTERN

THESE ARCHITECTURAL DECISIONS:

- Single Source of Truth: The Store - centralized client side state
- One Way Data Flow
- State is only updated through pure functions (reducers)

PROVIDE THESE BENEFITS:

- Provides insight into race conditions bugs
- Deterministic State Reproduction
- Deterministic View Renders

BENEFITS OF THE REDUX PATTERN

THESE ARCHITECTURAL DECISIONS:

- Single Source of Truth: The Store - centralized client side state
- One Way Data Flow
- State is only updated through pure functions (reducers)

PROVIDE THESE BENEFITS:

- Provides insight into race conditions bugs
- Deterministic State Reproduction
- Deterministic View Renders
- State updates are transactional

BENEFITS OF THE REDUX PATTERN

THESE ARCHITECTURAL DECISIONS:

- Single Source of Truth: The Store - centralized client side state
- One Way Data Flow
- State is only updated through pure functions (reducers)

PROVIDE THESE BENEFITS:

- Provides insight into race conditions bugs
- Deterministic State Reproduction
- Deterministic View Renders
- State updates are transactional
- Testing is easier

BENEFITS OF THE REDUX PATTERN

THESE ARCHITECTURAL DECISIONS:

- Single Source of Truth: The Store - centralized client side state
- One Way Data Flow
- State is only updated through pure functions (reducers)

PROVIDE THESE BENEFITS:

- Provides insight into race conditions bugs
- Deterministic State Reproduction
- Deterministic View Renders
- State updates are transactional
- Testing is easier
- More performant - Angular change detection OnPush setting

MORE PERFORMANT CHANGE DETECTION: "ONPUSH"

```
@Component({
  // ...
  changeDetection: ChangeDetectionStrategy.OnPush
})
```

```
let squirl = {
    tail: 'bushy',
    color: 'red-brown'
    age: 2,
    energyLevel: 'high',
    favoriteFood: 'pizza'
};
```

MORE PERFORMANT CHANGE DETECTION: "ONPUSH"

```
@Component({
  // ...
  changeDetection: ChangeDetectionStrategy.OnPush
})
```

```
let squirl = {
    tail: 'bushy',
    color: 'red-brown'
    age: 2,
    energyLevel: 'high',
    favoriteFood: 'pizza'
};
```

!! Mutating the state, same reference

```
squirl.age = 12;
squirl.energyLevel = 'low';
```

MORE PERFORMANT CHANGE DETECTION: "ONPUSH"

```
@Component({
  // ...
  changeDetection: ChangeDetectionStrategy.OnPush
})
```

```
let squirl = {
    tail: 'bushy',
    color: 'red-brown'
    age: 2,
    energyLevel: 'high',
    favoriteFood: 'pizza'
};
```

!! Mutating the state, same reference

```
squirl.age = 12;
squirl.energyLevel = 'low';
```

Replacing state, new reference

```
squirl = {
    tail: 'bushy',
    color: 'red-brown'
    age: 12,
    energyLevel: 'low',
    favoriteFood: 'pizza'
};
```

DETERMINISTIC STATE/VIEWS WITH ACTIONS, TRANSACTIONAL LOG

- Show redux dev tools, in demo app

DISPATCHING AN ACTION

```
<bc-toolbar (openMenu)="openSidenav()">
  Book Collection
</bc-toolbar>
```

```
openSidenav() {
  this.store.dispatch(new layout.OpenSidenav());
}
```

THE ACTION

```ts
import { Action } from '@ngrx/store';

export const OPEN_SIDENAV = '[Layout] Open Sidenav';
export const CLOSE_SIDENAV = '[Layout] Close Sidenav';

export class OpenSidenav implements Action {
  readonly type = OPEN_SIDENAV;
}

export class CloseSidenav implements Action {
  readonly type = CLOSE_SIDENAV;
}

export type Actions = OpenSidenav | CloseSidenav;
```

```
{
  type: '[Layout] Open Sidenav',
}
```

THE SOLUTION:

DATA FLOW

@ngrx/STORE

Action Stream

REDUCER

(state, action) =>
state

STATE

{ ...appState }

@ngrx/
Effects

Https

SERVER

ACTION

{ type: addToDo
payload: ski }

Dispatch
Action

Dispatch Action

ANGULAR
COMPONENT

Components
Subscribes to
State

THE REDUCER

```
import * as layout from '../actions/layout';

export function reducer(state = initialState, action: layout.Actions): State {
  switch (action.type) {
    case layout.CLOSE_SIDENAV:
      return {
        showSidenav: false,
      };

    case layout.OPEN_SIDENAV:
      return {
        showSidenav: true,
      };

    default:
      return state;
  }
}
```

THE SOLUTION:

DATA FLOW

@ngrx/STORE

Action Stream

REDUCER

(state, action) =>
state

STATE

{ ...appState }

@ngrx/
Effects

Https

SERVER

ACTION

{ type: addToDo
payload: ski }

Dispatch
Action

Dispatch Action

ANGULAR
COMPONENT

Components
Subscribes to
State

STATE

```
import * as layout from '../actions/layout';
```

```
export interface State {
  showSidenav: boolean;
}

const initialState: State = {
  showSidenav: false,
};
```

THE SOLUTION:

DATA FLOW

@ngrx/STORE

Action Stream

REDUCER

(state, action) =>
state

STATE

{ ...appState }

@ngrx/
Effects

Https

SERVER

ACTION

{ type: addToDo
payload: ski }

Dispatch
Action

Dispatch Action

ANGULAR
COMPONENT

Components
Subscribes to
State

SELECTOR & COMPONENT

```ts
export const getShowSidenav = (state: State) => state.showSidenav;
```

```ts
export class AppComponent {
  showSidenav$: Observable<boolean>;
  loggedIn$: Observable<boolean>;

  constructor(private store: Store<fromRoot.State>) {
    this.showSidenav$ = this.store.select(fromRoot.getShowSidenav);
    this.loggedIn$ = this.store.select(fromAuth.getLoggedIn);
  }
```

TEMPLATE

```
<bc-sidenav [open]="showSidenav$ | async">
  <bc-nav-item (navigate)="closeSidenav()" *ngIf="loggedIn$ | async" routerLink="/"
    My Collection
  </bc-nav-item>
  <bc-nav-item (navigate)="closeSidenav()" *ngIf="loggedIn$ | async" routerLink="/b
    Browse Books
  </bc-nav-item>
  <bc-nav-item (navigate)="closeSidenav()" *ngIf="!(loggedIn$ | async)">
    Sign In
  </bc-nav-item>
  <bc-nav-item (navigate)="logout()" *ngIf="loggedIn$ | async">
    Sign Out
  </bc-nav-item>
</bc-sidenav>
```

THE SOLUTION:

DATA FLOW

@ngrx/STORE

Action Stream

REDUCER

(state, action) =>
state

STATE

{ ...appState }

@ngrx/
Effects

Https

SERVER

ACTION

{ type: addToDo
payload: ski }

Dispatch
Action

Dispatch Action

ANGULAR
COMPONENT

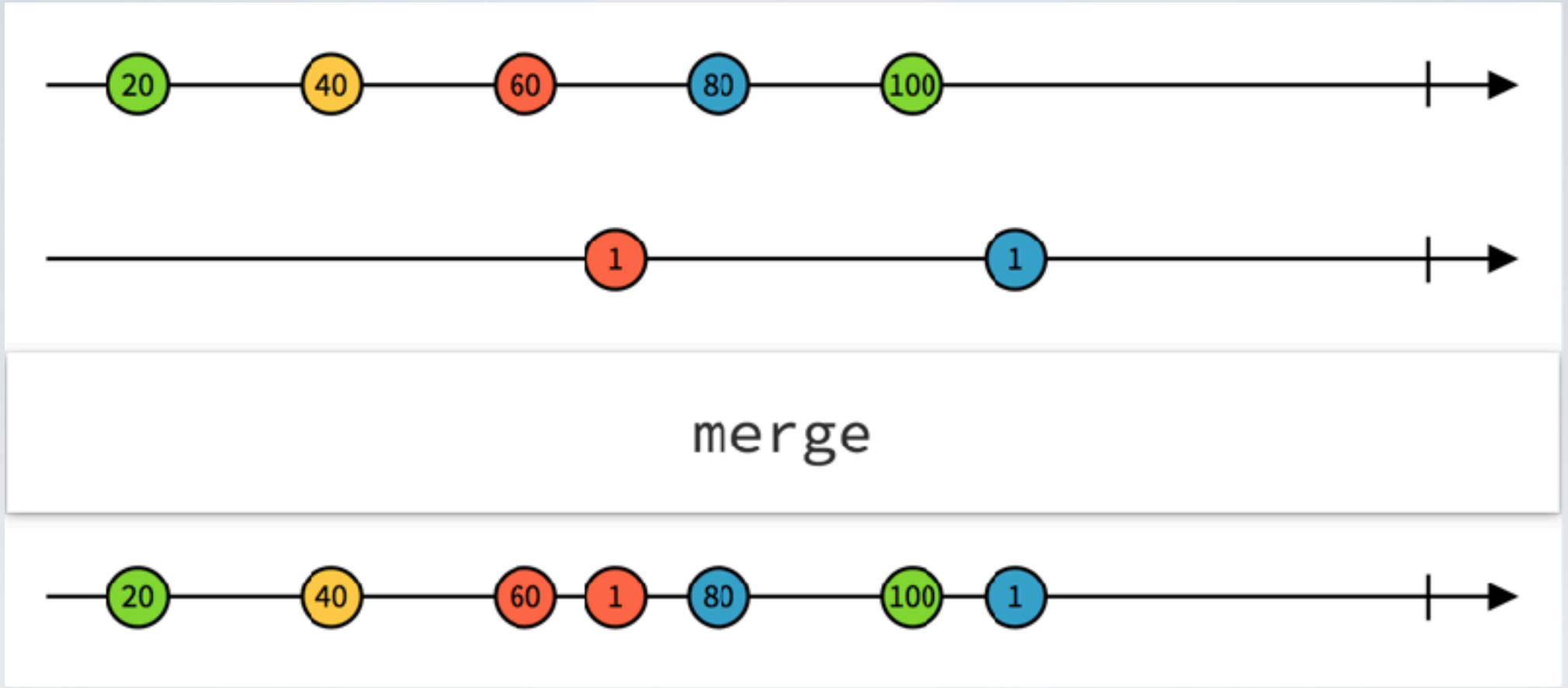
Components
Subscribes to
State

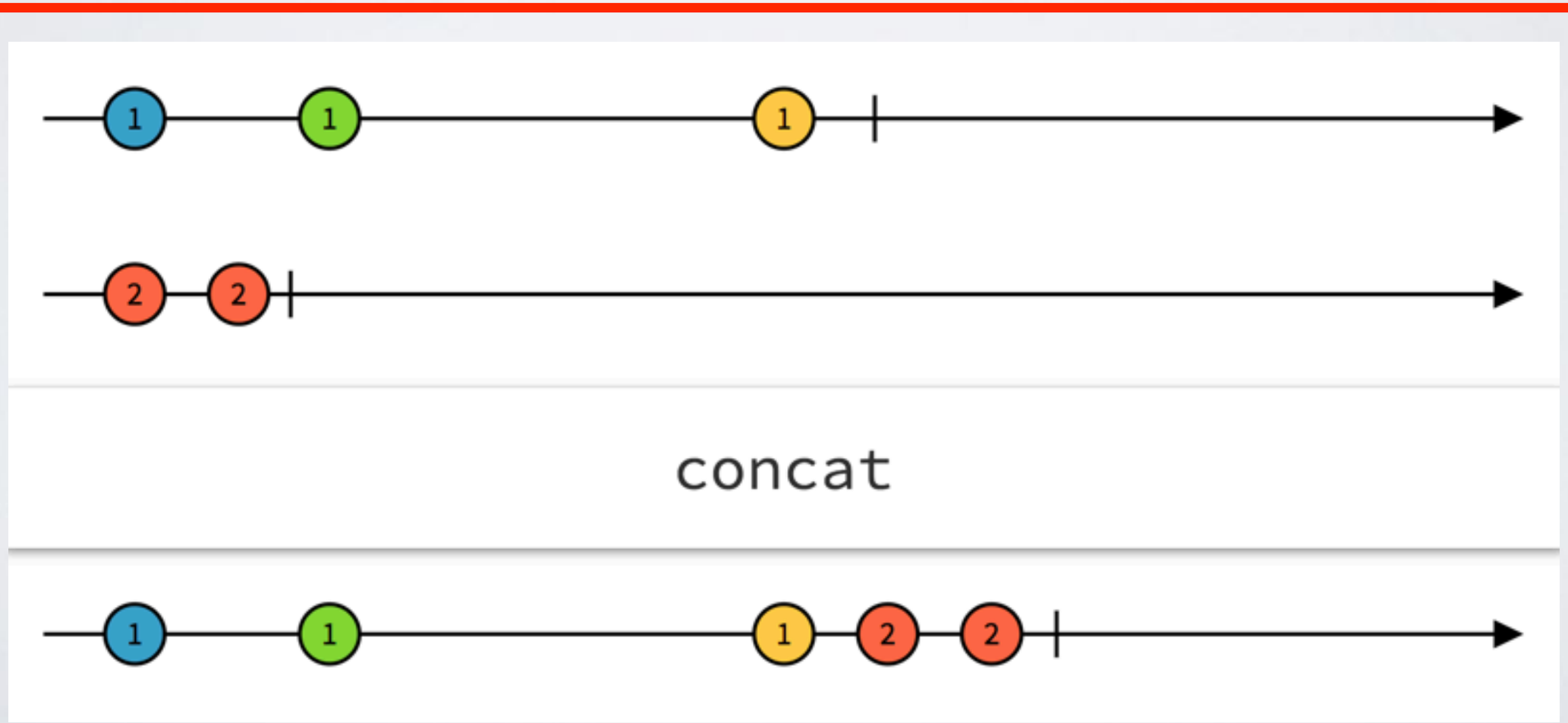
EFFECT

```
@Injectable()
export class BookEffects {
  @Effect()
  search$: Observable<Action> = this.actions$
    .ofType<book.Search>(book.SEARCH)
    .debounceTime(this.debounce, this.scheduler || async)
    .map(action => action.payload)
    .switchMap(query => {
      if (query === '') {
        return empty();
      }

      const nextSearch$ = this.actions$.ofType(book.SEARCH).skip(1);

      return this.googleBooks
        .searchBooks(query)
        .takeUntil(nextSearch$)
        .map((books: Book[]) => new book.SearchComplete(books))
        .catch(() => of(new book.SearchComplete([])));
    });
```

EFFECT

```
@Injectable()
export class DisclosuresEffects {

  @Effect() disclosuresContinue$ = this.actions$
    .ofType(LAST_PAGE_BEFORE_SOMETHING_AWESOME_CONTINUE)
    .map(toPayload)
    .exhaustMap(payload => concat(
        this.updateSecurityCert(toApiSecurityCertification(payload)),
        this.updateAccounts(),
        this.updateBillPay(),
        this.updateAccountConfigs(),
        this.orderTeddyBear(),
        this.updateDebitCardApps(),
        this.applyCoupon(),
        this.updateUserRelationships(),
        this.createWelcomeEmail(payload.emailDisclosures),
        this.sendWelcomeEmail(),
        [new CompleteAction()]
    )
    .catch(error => [new DisclosuresContinueErrorAction(error)])
  );
```


20
40
60
80
100
1
1
merge
20
40
60
1
80
100
1


1
1
1
2
2
concat
1
1
1
2
2

Additional resources:

https://github.com/apasternack/Presentations/tree/master/WestervilleWebMeetup_2.6.2018

THE END

STAY IN TOUCH

| Adam Pasternack

| Twitter: @AJPasternack

| adam.pasternack@gmail.com