State management in Angular

Mario Fernandez

What is state management?

Disclaimer

Disclaimer

I am a declared *React* fan

Disclaimer

I am a declared *React* fan

Probably unqualified to provide too many opinions on *Angular*

Disclaimer

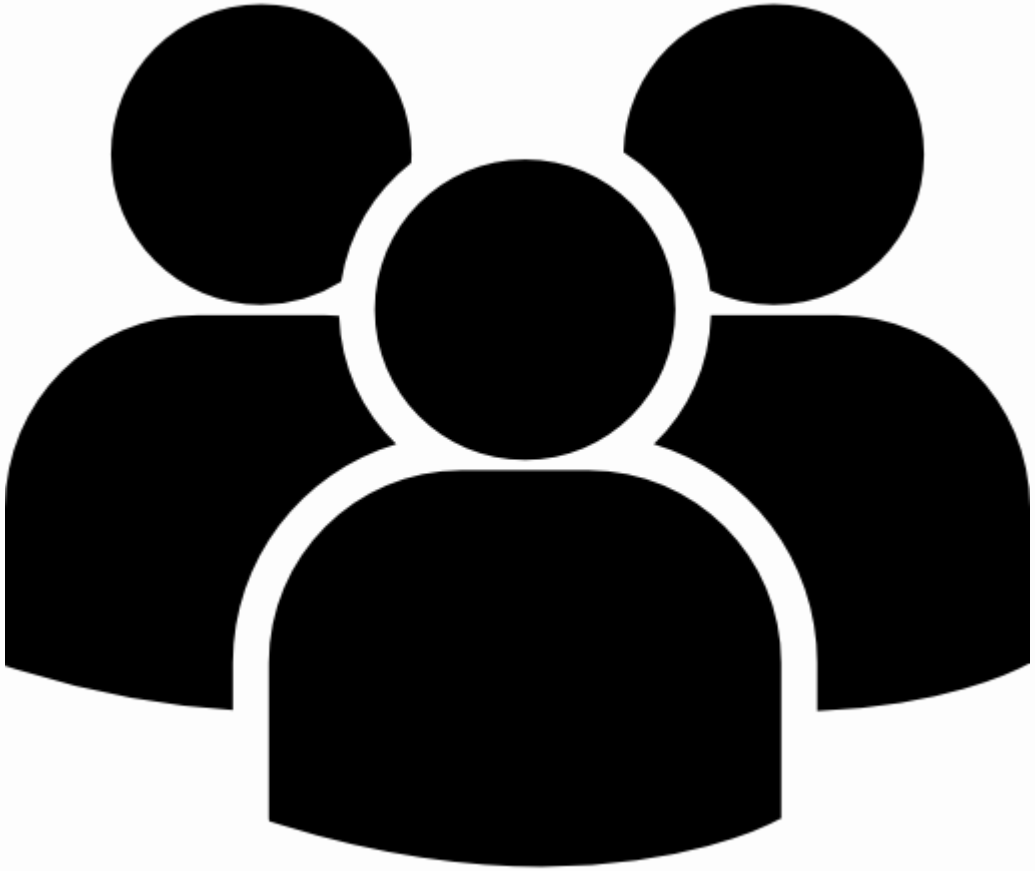
I am a declared *React* fan

Probably unqualified to provide too many opinions on *Angular*

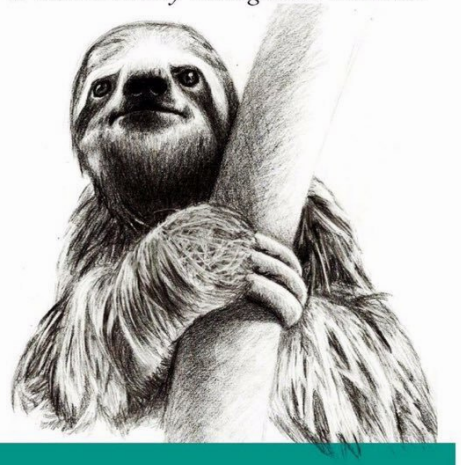
Will do so anyway

Context

```
commit 4db543c897d44849eb073facfe1562a156b02137
Author: Frontend Dev
Date:   Sun Oct 30 21:10:48 2016 +0100

    Initial commit
```



Cutting corners to meet arbitrary management deadlines



Essential

Copying and Pasting from Stack Overflow

O'REILLY®

The Practical Developer
@ThePracticalDev

Patterns

Input/Output

What

What

Most components do not hold their own state

What

Most components do not hold their own state

Instead, they receive their data as `@Input`, and trigger changes upstream with `@Output`

What

Most components do not hold their own state

Instead, they receive their data as `@Input`, and trigger changes upstream with `@Output`

Presentational Components, Dumb Components, Functional Components

Example

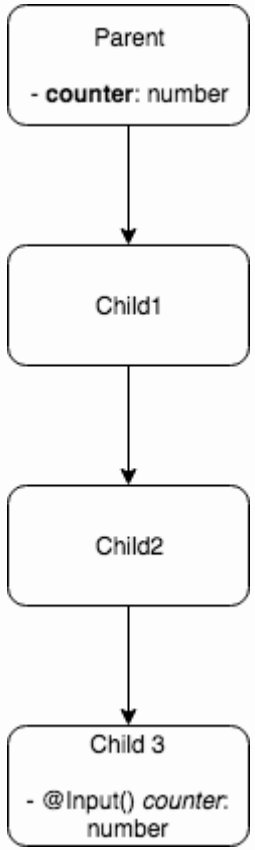
```
class SimpleComponent {
  @Input() selectedDate: Date;
  @Output() onJump = new EventEmitter<Date>();

  jumpBack(date: Date) {
    this.onJumpto.emit(date);
  }
}
```

That's it, problem solved

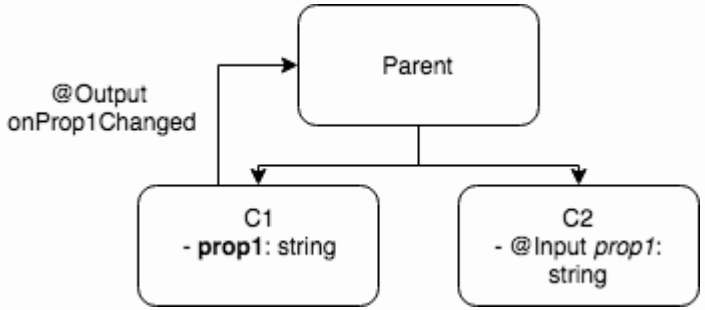
End

Maybe not

The input train



The sideways state



Let's take a break to talk about bad ideas

ViewChild

What

What

A way to bind a child component to a variable

What

A way to bind a child component to a variable

Access the child's component state directly

```
class Evil {
  @ViewChild(PoorChildrenComponent)
  sayGoodbyeToEncapsulation: PoorChildrenComponent;

  beEvil() {
    this.sayGoodbyeToEncapsulation.changeStuffExternally();
  }
}
```

Break encapsulation

Couple components forever


WTF?

Nested Forms

What

What

Combine multiple forms

What

Combine multiple forms

Treat them as one

```
<!-- Parent component template -->
<child-component
  [parentForm]="formGroup">
```

```
class ChildComponent implements OnInit {
  @Input() parentForm: FormGroup

  ngOnInit() {
    this.parentForm.setControl('stuff', this.myControl);
  }
}
```

```
class ParentComponent implements OnInit {
  formGroup: FormGroup

  youWontLikeThis() {
    this.formGroup.get('stuff').setValue('surprise!');
  }
}
```

Service as a Global

```
@Injectable()
export class GlobalService {
  stuff: string
  moreStuff: string
}
```


Shortcut

Back to more reasonable practices

ngrx-store

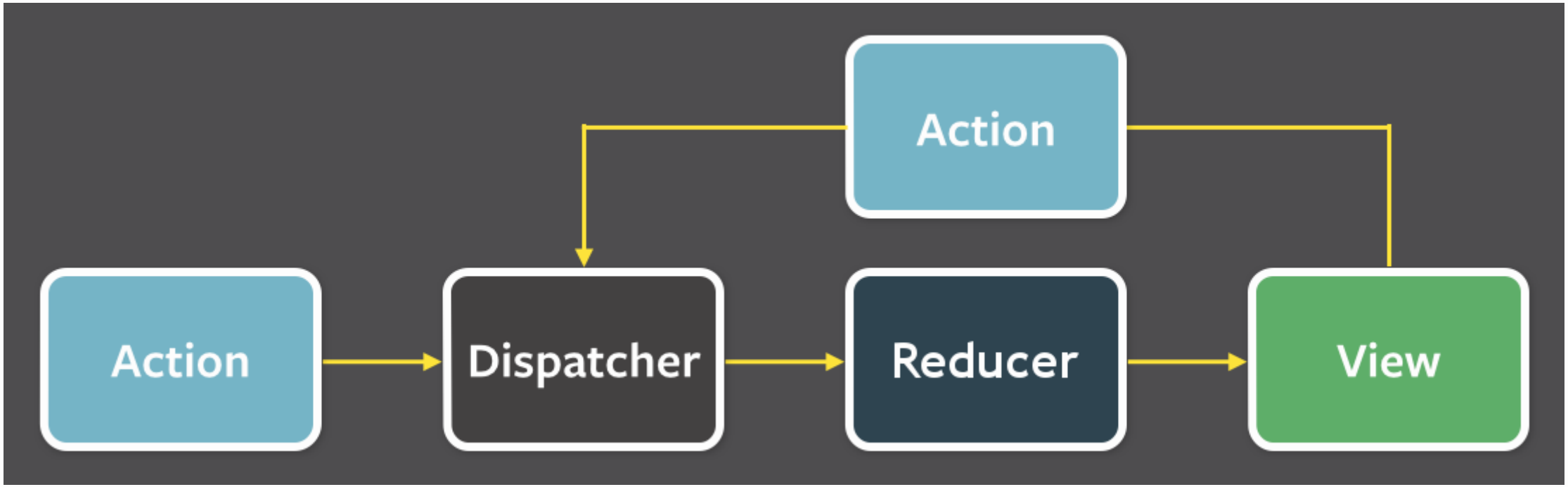
What

What

Redux for angular

What

Redux for angular



This will be the final solution, I thought

The harsh reality of introducing *Redux*

- *Redux* is not the easiest library to explain, specially to developers without a frontend background
- Incremental adoption is problematic
- Not as established as a pattern in the community

Data Services

What

A *Service* that:

- holds data
- offers an API to manipulate it
- provides a way to get notified of updates

A simple service

```
@Injectable()
export class Store {
  private timeslotSubject = new BehaviorSubject<string>(null);
  timeslot$ = this.timeslotSubject.asObservable();

  selectTimeslot(timeslot: string) {
    this.timeslotSubject.next(timeslot);
  }
}
```

Changing the state

```
class Publisher implements OnInit {
  constructor(private store: Store) {}

  onSelection(ts: string) {
    this.store.selectTimeslot(ts);
  }
}
```

Getting updates

```
class Subscriber implements OnInit {
  timeslot: string

  constructor(private store: Store) {}

  ngOnInit() {
    this.store.timeslot$
      .filter(ts => !!ts)
      .subscribe(ts => this.timeslot = ts)
  }
}
```

Caveats

- Only partially adopted
- Not always clear which components should subscribe
- Should a publisher hold its own state, or get it through the service as well?

Summary

Learnings

- `Angular` offers many shortcuts that can be easily misused
- **Discipline** is needed to keep your app under control
- *Data Services* seem to be the most accepted pattern
- *Input/Output* work as well, specially for simple apps

Links

- https://hceris.com/angular-from-react-part1/
- https://angular.io/guide/component-interaction#parent-and-children-communicate-via-a-service
- https://blog.angular-university.io/how-to-build-angular2-apps-using-rxjs-observable-data-services-pitfalls-to-avoid/

Thank you!

Questions?