Smart / Dumb Components

Angela Damaso &
Yong Choi

KAPLAN TEST PREP

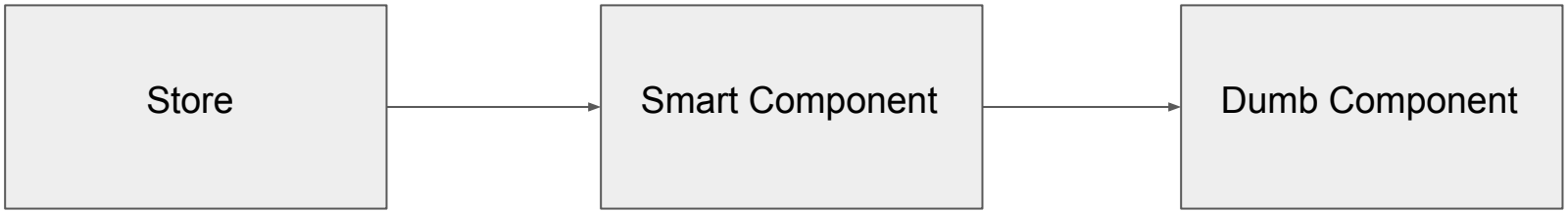
Smart & Dumb Components

Why did we choose this architecture?

- Easy to create reusable components
- Gives us a consistent architecture across each project
  - Example: Global navigation bar is structured the same way in TestUI repo and Student Reporting repo
- Works great with NGRX
- Best Practice (many talks and articles about this topic)
  - https://medium.com/@dan_abramov/smart-and-dumb-components-7ca2f9a7c7d0
  - https://redux.js.org/docs/basics/UsageWithReact.html

Data flows from

Store ➡ Smart Component ➡ Dumb Component



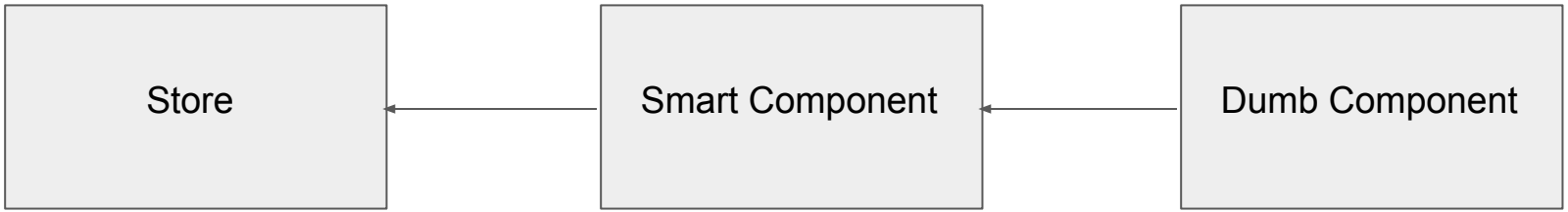
Contains the global state of the application

Hooks up the Store with the Dumb Component

Just outputs the state

Store changes from

Dumb Component ➡ Smart Component ➡ Store

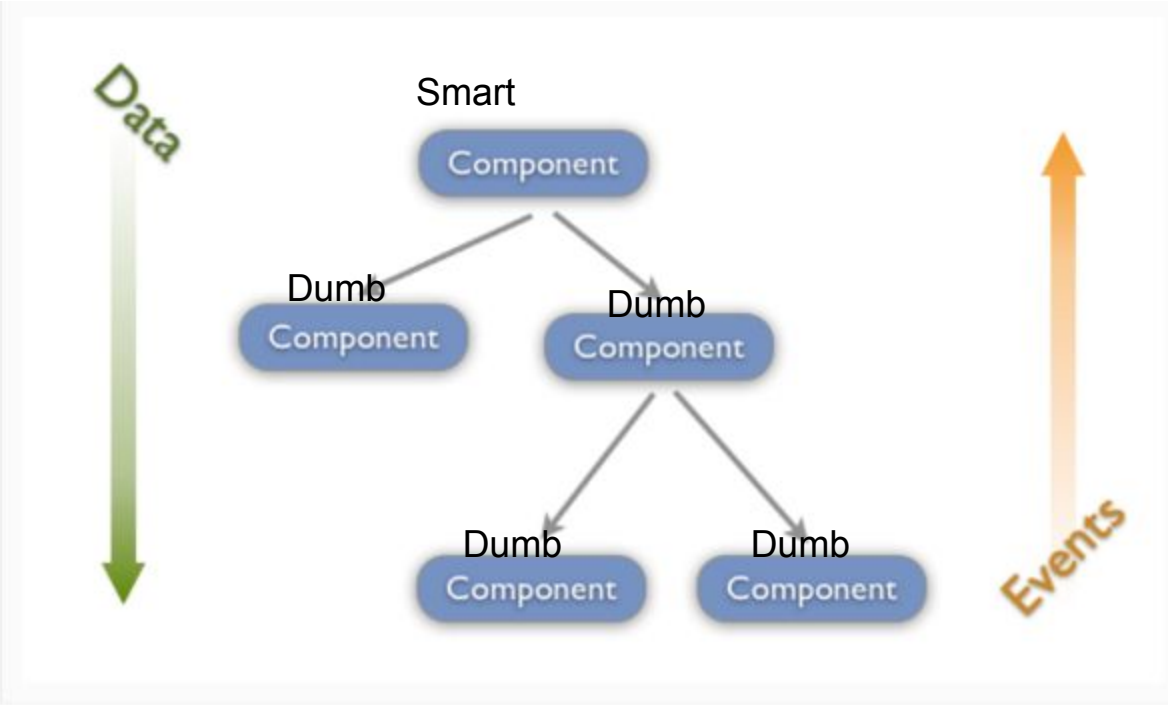


Contains the global state of the application

Tells the Store to update itself

Tells the smart component to modify the state

The components just reflect the Store


Data
Smart
Component
Dumb
Component
Dumb
Component
Dumb
Component
Dumb
Component
Events

Smart & Dumb Architecture

- Using the smart/dumb architecture allows us to use a variety of different Stores.
- The dumb components have no idea what type of Store is being used.
- One site could use simple services, another ngrx, and another a future state management solution that has yet to be written.
    - **Example:** TestUI repo is using NGRX for state management. Student Reporting is using an Angular Simple Service. Both repos use the same Higgs components and talk to the different stores.
- We use @Input and @Output to communicate between components
    - https://toddmotto.com/component-events-event-emitter-output-angular-2
    - https://angular.io/guide/component-interaction

Good Dumb Component Examples

```
@Component({
    selector: 'cart-item',
    template: `
    <li class="margin-t-20">
        {{cartItem.title}} - \${{cartItem.price}} x {{cartItem.quantity}}
    </li>
    `,
    changeDetection: ChangeDetectionStrategy.OnPush
})
export class CartItem {
    @Input() cartItem: any;
}
```

```
@Component({
    selector: 'refresh-button',
    template: `
    <button (click)="invalidateReddit.emit(selectedReddit)">
        Refresh {{selectedReddit}} Posts
    </button>
    `,
    changeDetection: ChangeDetectionStrategy.OnPush
})
export class RefreshButton {
    @Input() selectedReddit : string;
    @Output() invalidateReddit: EventEmitter<string> = new EventEmitter<string>();
}
```

Bad Dumb Component Examples

What's wrong with this dumb component?

```
@Component({
    selector: 'product-item',
    template: `
    <li class="margin-t-20">
        <div>{{product.title}} - {{product.price}}</div>
        <div>{{product.inventory}} left</div>
        <button class="pure-button pure-button-primary"
            (click)="add(product)"
        </button>
    </li>
    `,
    changeDetection: ChangeDetectionStrategy.OnPush
})
export class ProductItem {
    @Input() product: IProduct;
    @Output() addToCart: EventEmitter<IProduct>
      = new EventEmitter<IProduct>();

    add(product: IProduct) {
      product.inventory = product.inventory - 1;
      this.addToCart.emit(product);
    }
}
```

What's wrong with this dumb component?

Dumb components should not modify state!!!

```
@Component({
    selector: 'product-item',
    template: `
    <li class="margin-t-20">
        <div>{{product.title}} - {{product.price}}</div>
        <div>{{product.inventory}} left</div>
        <button class="pure-button pure-button-primary"
            (click)="add(product)"
        </button>
    </li>
    `,
    changeDetection: ChangeDetectionStrategy.OnPush
})
export class ProductItem {
    @Input() product: IProduct;
    @Output() addToCart: EventEmitter<IProduct>
      = new EventEmitter<IProduct>();

    add(product: IProduct) {
      product.inventory = product.inventory - 1;
      this.addToCart.emit(product);
    }
}
```

What's wrong with this dumb component?

```
@Component({
    selector: 'cart-list',
    template: `
        <ul *ngIf="cartList">
            <cart-item *ngFor="let cartItem of cartList"
                [cartItem]="cartItem">
            </cart-item>
        </ul>
        <button class="pure-button pure-button-primary"
            (click)="checkout())"> Checkout</button>
    `
})
export class CartList {
    @Input() cartList: any;

    constructor(private checkoutService: CheckoutService) {
      checkout() {
        this.checkoutService.dingDangDo();
      }
    }
}
```

What's wrong with this dumb component?

Dumb components should not modify state!!!

Presentational components shouldn't be doing work! It should only have logic related to the UI.

```
@Component({
    selector: 'cart-list',
    template: `
        <ul *ngIf="cartList">
            <cart-item *ngFor="let cartItem of cartList"
                [cartItem]="cartItem">
            </cart-item>
        </ul>
        <button class="pure-button pure-button-primary"
            (click)="checkout())"> Checkout</button>
    `
})
export class CartList {
    @Input() cartList: any;

    constructor(private checkoutService: CheckoutService) {
      checkout() {
        this.checkoutService.dingDangDo();
      }
    }
}
```

Good Smart Component Examples

This smart component is hooking up to the state service which is the Store.

Smart component doesn't directly update the Store. It runs a function on the Store to do it.

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {

  counter = 0;

  constructor(
    private state: State
  ) {
  }

  ngOnInit() {
    this.counter = this.state.counter;
  }

  increment() {
    this.state.add();
  }
```