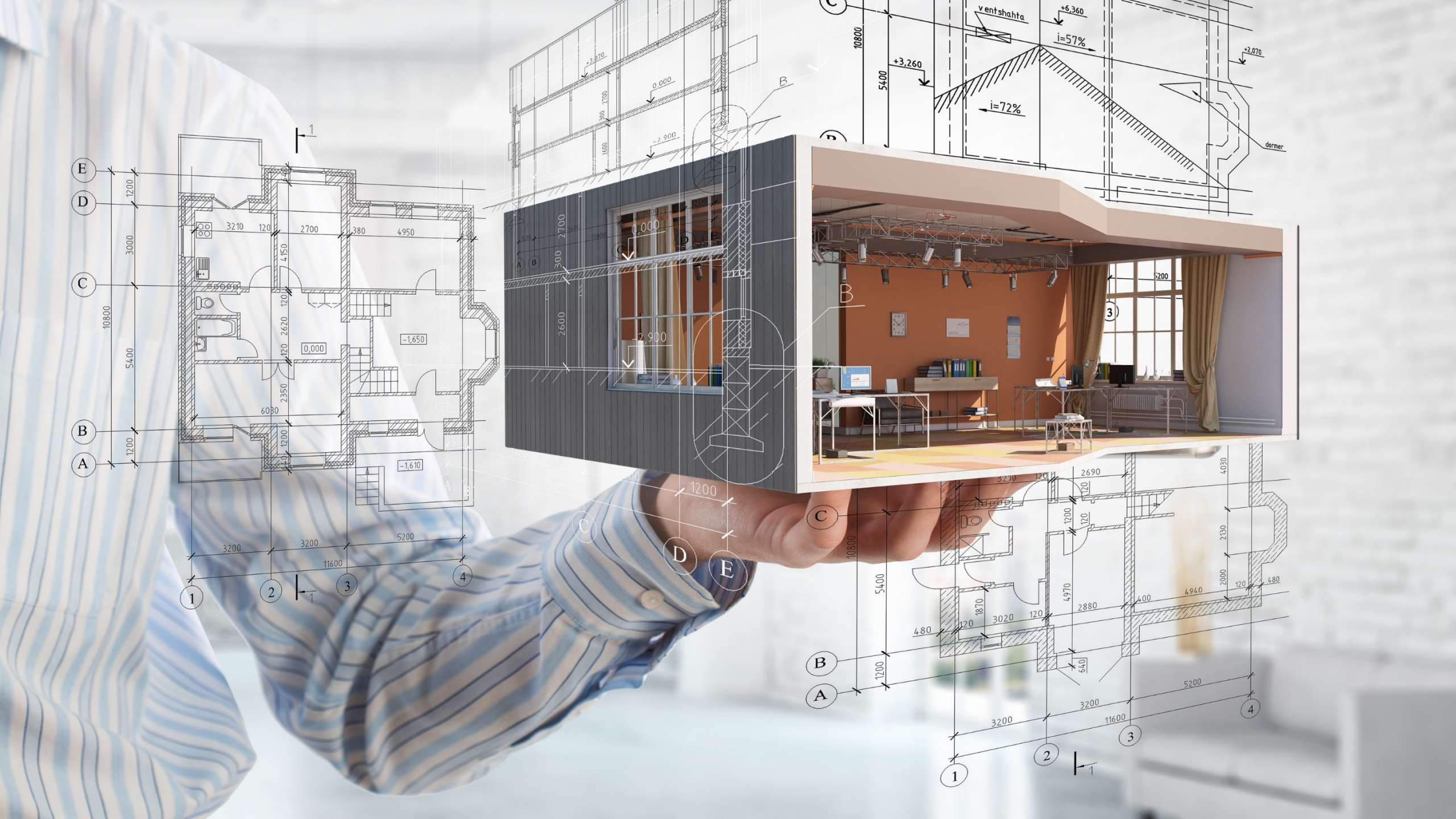
Architectural Considerations

Duncan Hunter
CONSULTANT | SPEAKER | AUTHOR

@dunchunter duncanhunter.com.au


ventshahta
i=57%
i=72%
+6,360
+3,260
+2,070
dormer
0,000
-1,650
-1,610
11600
10800
5400

Module Overview

- Presentational and container components
- Change detection strategy OnPush
- Adding an index.ts file to our state folders
- Moving related actions into separate files
- Creating composable state modules

NgRx takes logic out of components.

Presentational and Container Components

Division of components into two categories

Presentational and Container Components

Presentational	Container
Concerned with how things look	Concerned with how things work
HTML markup and CSS styles	Have little to no HTML and CSS styles
No dependencies on the rest of the app	Have injected dependencies
Don't specify how data is loaded or changed but emit events via @Outputs	Are stateful and specify how data is loaded or changed
Receive data via @Inputs	Top level routes
May contain other components	May contain other components

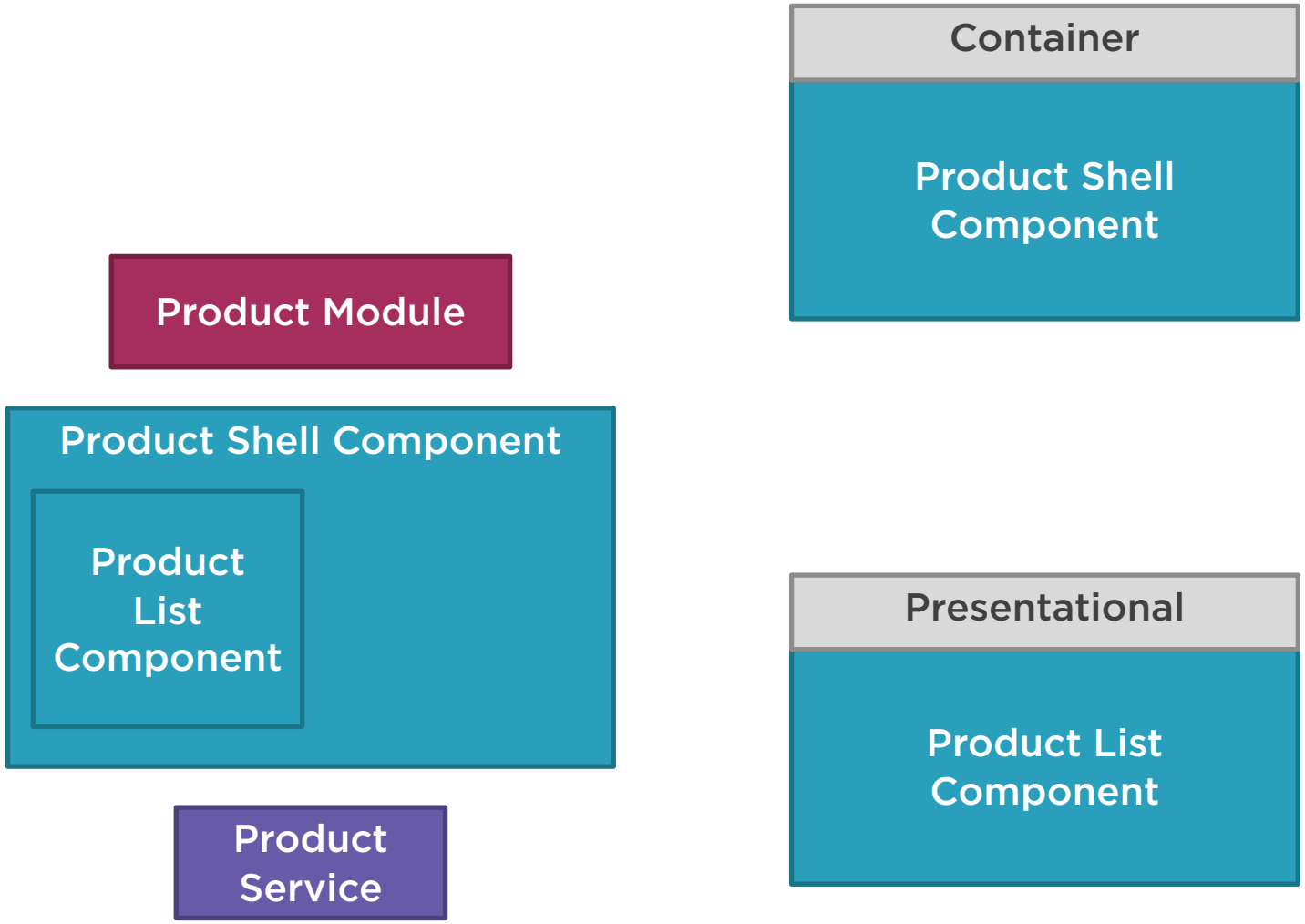
Benefits of Presentational and Container Components

Performance

Composability

Easier to test

Sample Application Architecture



Presentational and Container Components

Product Shell Component

```
export class ProductShellComponent
implements OnInit {

  constructor() { }

  ngOnInit() {}
}
```

Product List Component

```
export class ProductListComponent
implements OnInit, OnDestroy {

 constructor(
   private store: Store<State>
 ) { }

 ngOnInit() {
  this.products$ = this.store.selectedProduct(...);
  ...
 }

 checkChanged() {
  this.store.dispatch(...);
 }
 ...

}
```

Presentational and Container Components

Product Shell Component

```
export class ProductShellComponent
implements OnInit {

 constructor(
   private store: Store<State>
 ) {}

 ngOnInit() {
  this.store.dispatch(...);
  this.products$ = this.store.select(...);
   ...
 }

 checkChanged() {
   this.store.dispatch(...);
 }

 newProduct() {
   this.store.dispatch(...);
 }
}
```

Product List Component

```
export class ProductListComponent {
 @Input() errorMessage: string;
 @Input() products: Product[];
 @Input() displayCode: boolean;
 @Input() selectedProduct: Product;
 @Output() displayCodeChanged = new EventEmitter<void>();
 @Output() initializeNewProduct = new EventEmitter<void>();
 @Output() productWasSelected = new EventEmitter<Product>();

 checkChanged() {
   this.displayCodeChanged.emit();
 }

 newProduct() {
   this.initializeNewProduct.emit();
 }

 productSelected(product) {
   this.productWasSelected.emit(product);
 }

}
```

Presentational and Container Components

Product Shell Template

```
<div class="row">
  <div class="col-md-4">
    <pm-product-list
        [displayCode]="displayCode$ | async"
        [products]="products$ | async"
        [selectedProduct]="selectedProduct$ | async"
        [errorMessage]="errorMessage$ | async"
        (displayCodeChanged)="checkChanged()”
        (initializeNewProduct)="newProduct()"
        (productWasSelected)="productSelected($event)">
    </pm-product-list>
  </div>
</div>
```

Demo

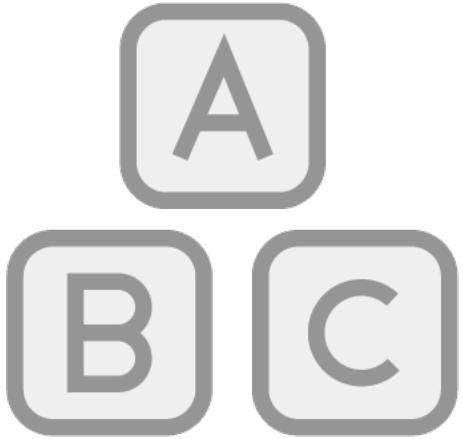
Container components

Demo

Presentational components

Benefits of Container and Presentational Components

Performance



Composability

Easier to test

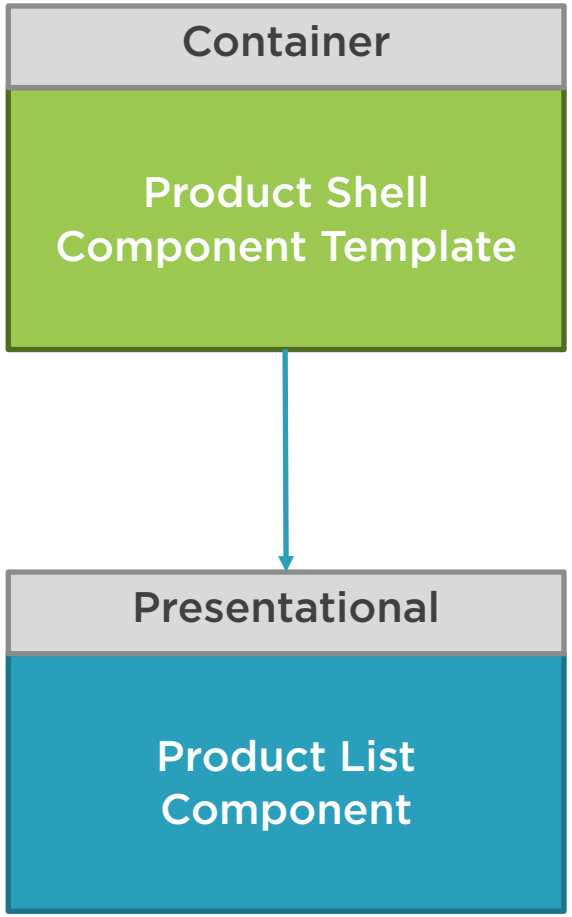
ChangeDetectionStrategy.OnPush

OnPush means that the change detector's mode will be initially set to CheckOnce

ChangeDetectionStrategy.OnPush

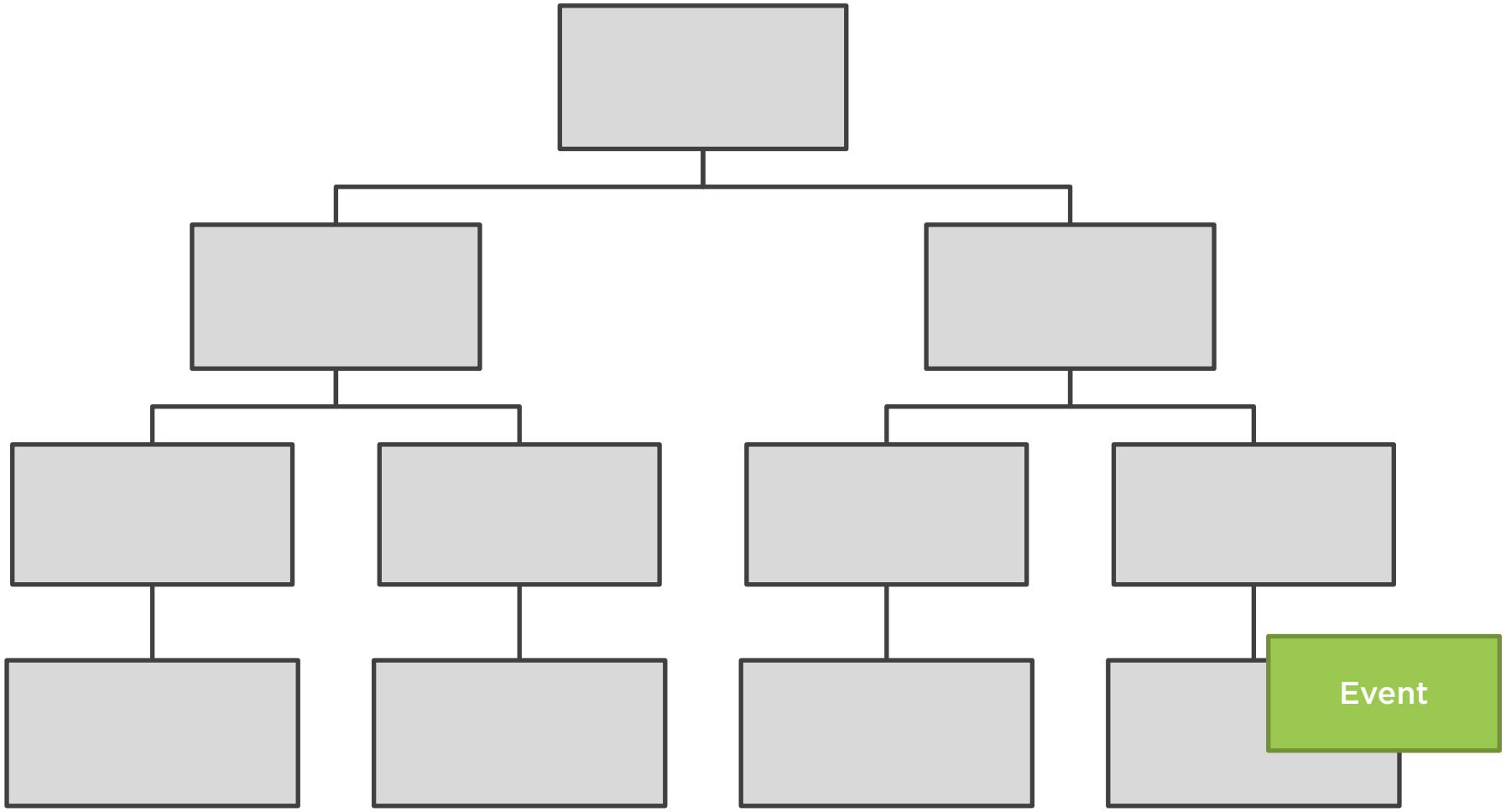
```
<pm-product-list
   [products]="products$ | async">
</pm-product-list>
```

```
export class ProductListComponent {
   @Input() products: Product[];
}
```

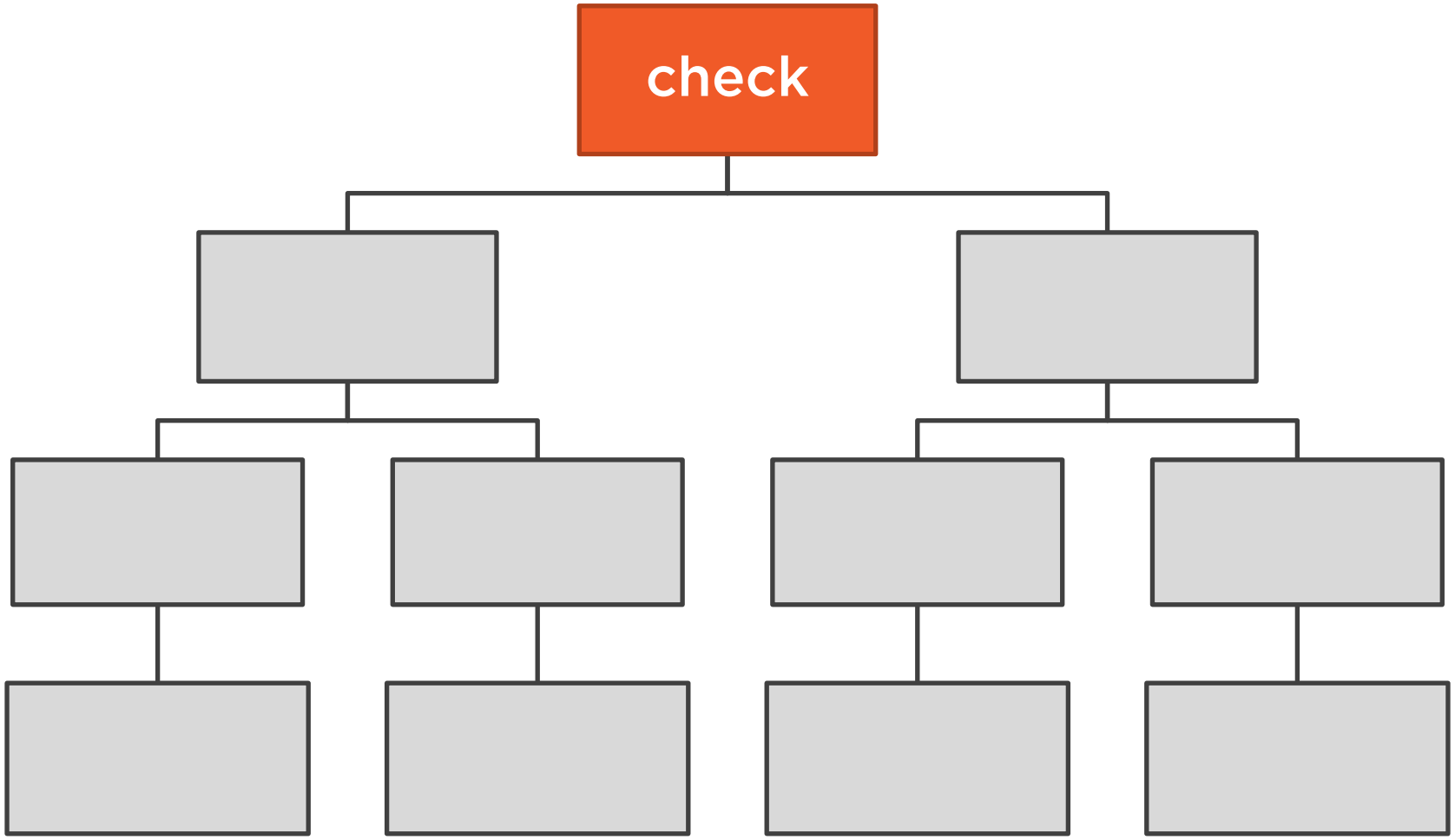


ChangeDetectionStrategy.Default

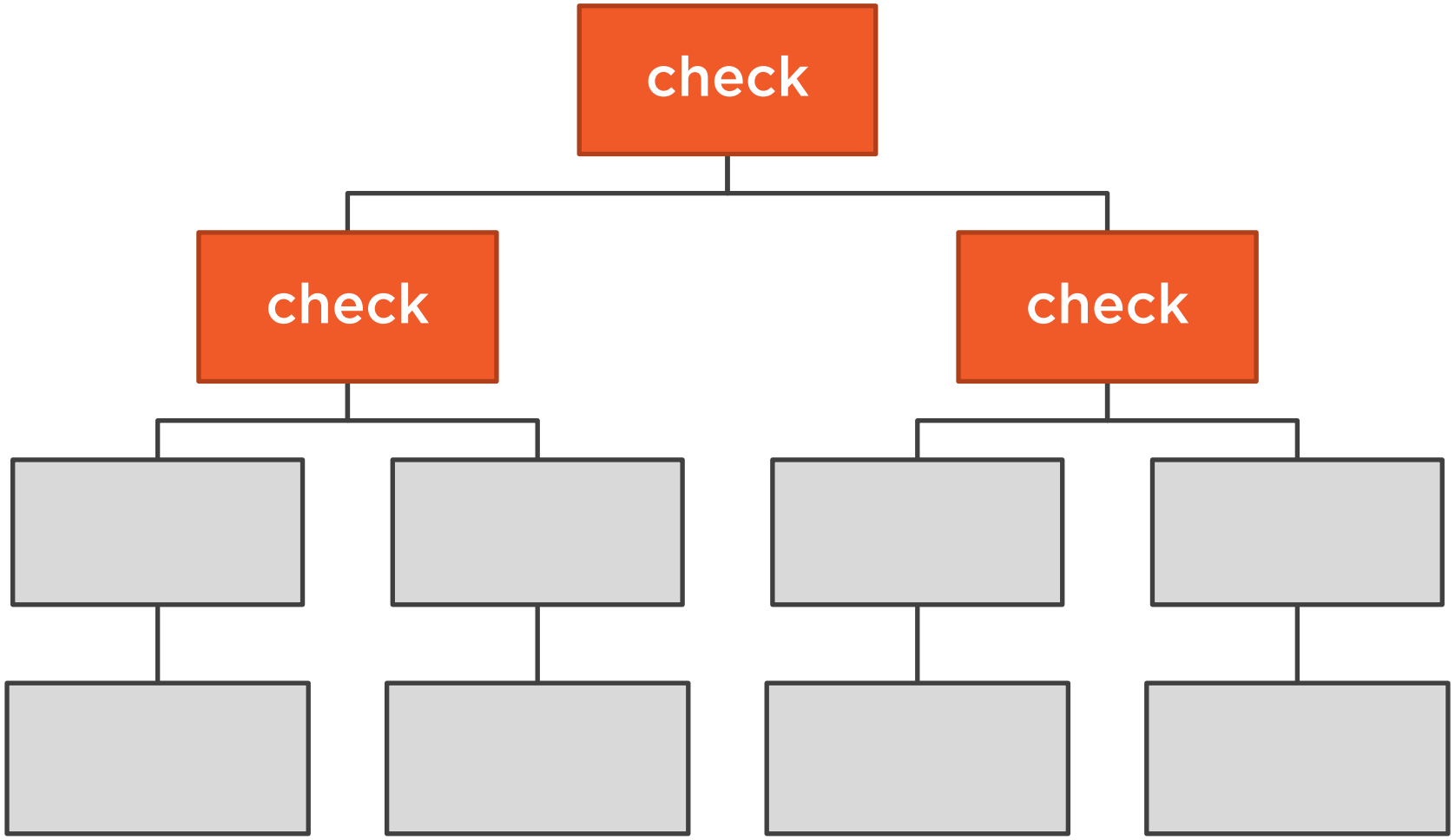
Change Detection



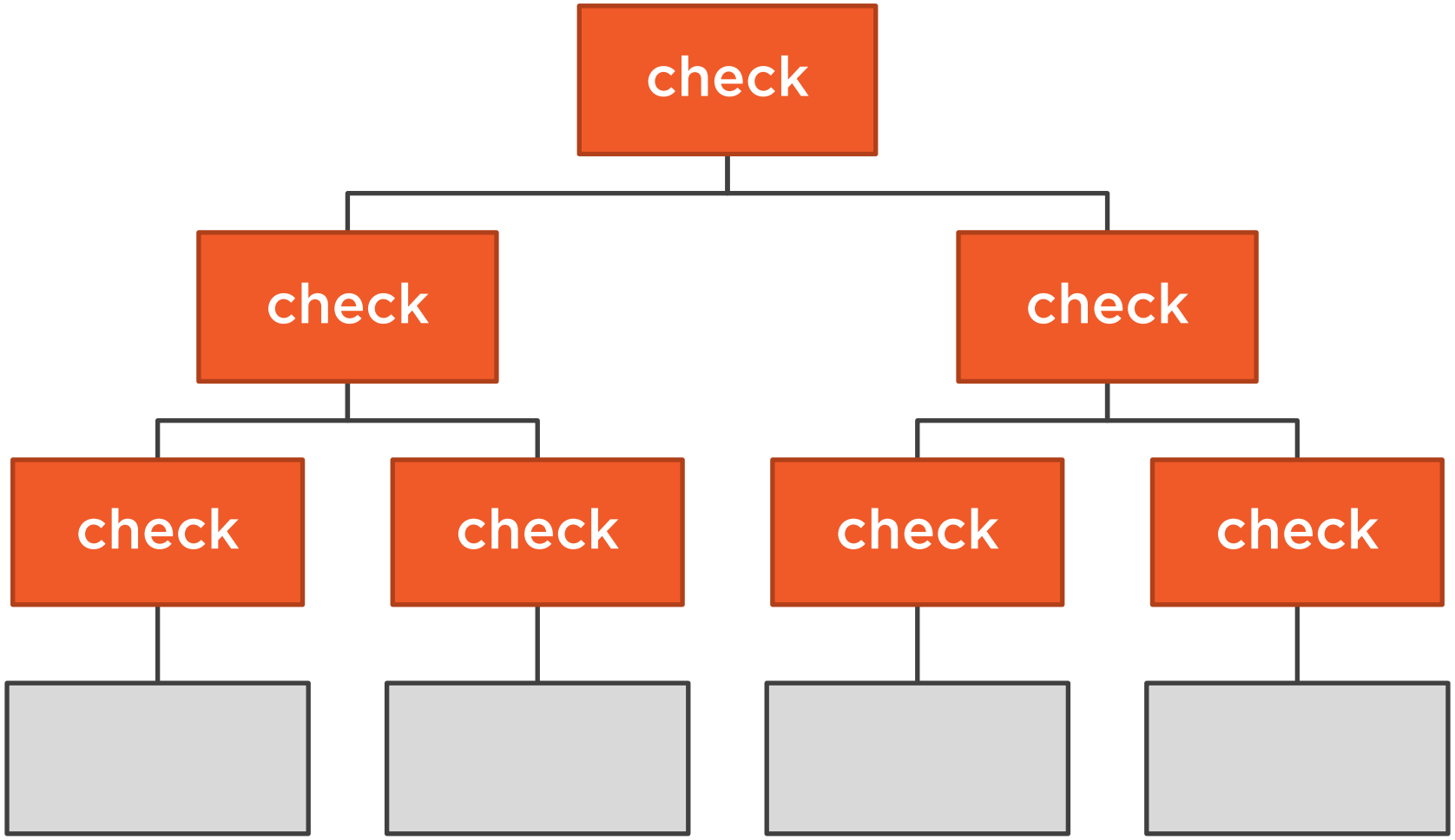
Change Detection



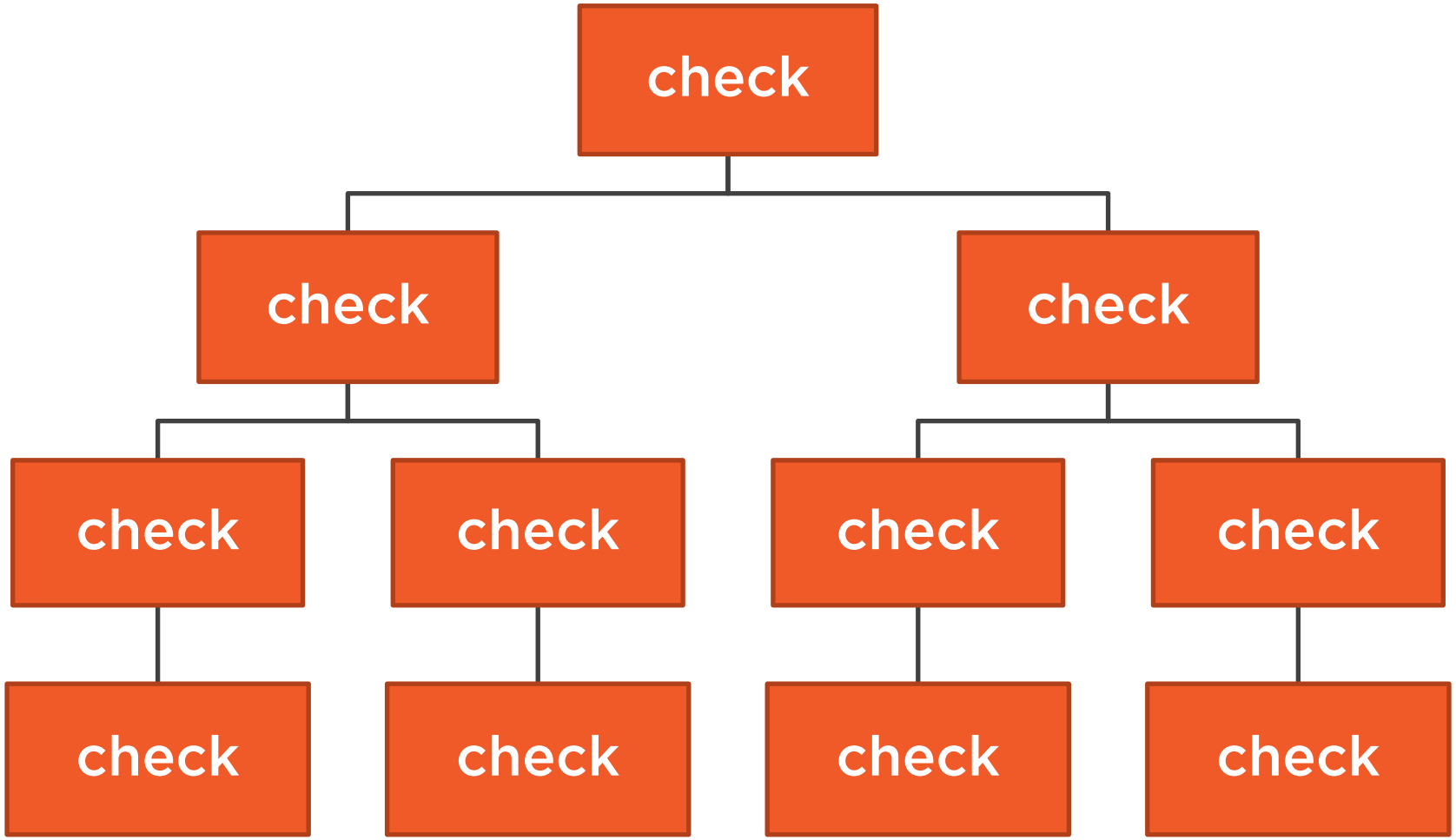
Change Detection



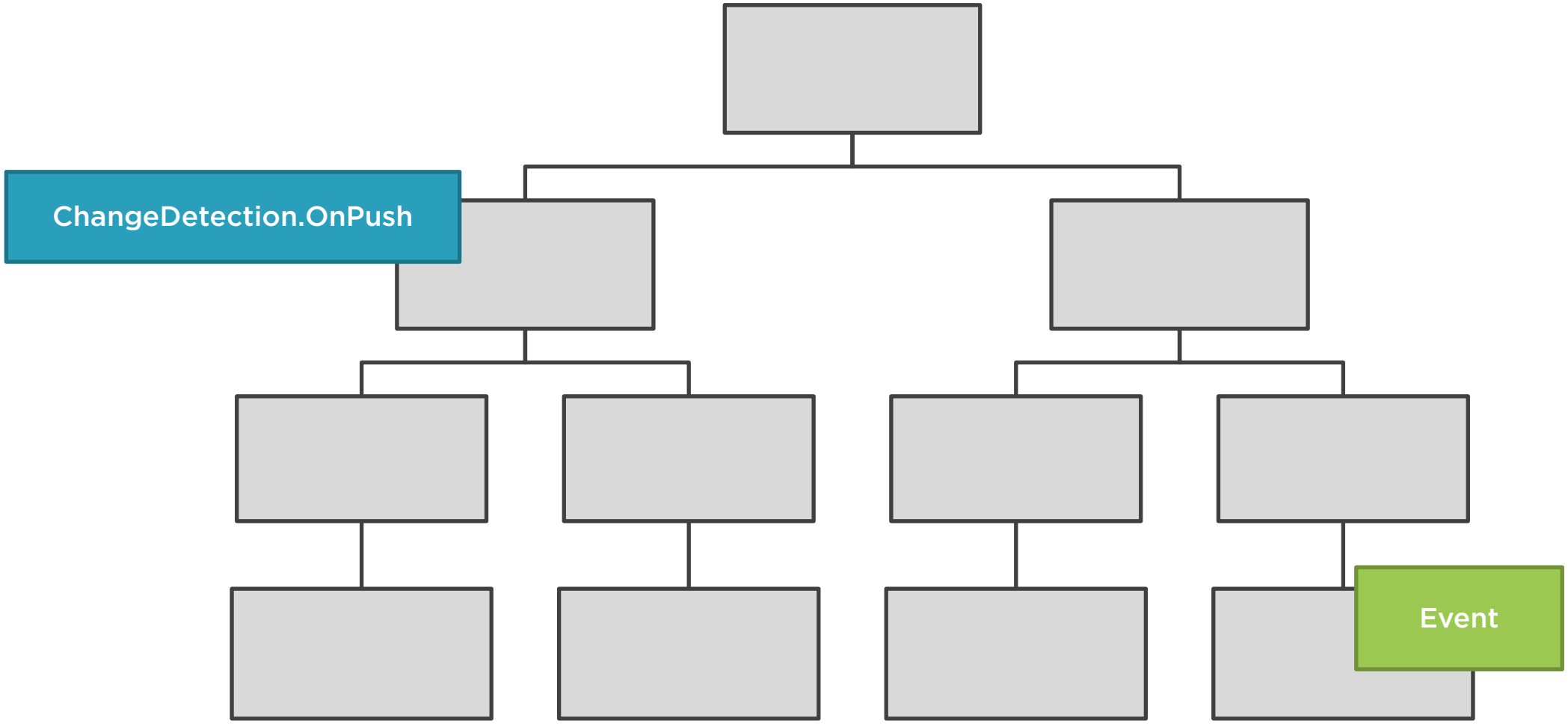
Change Detection



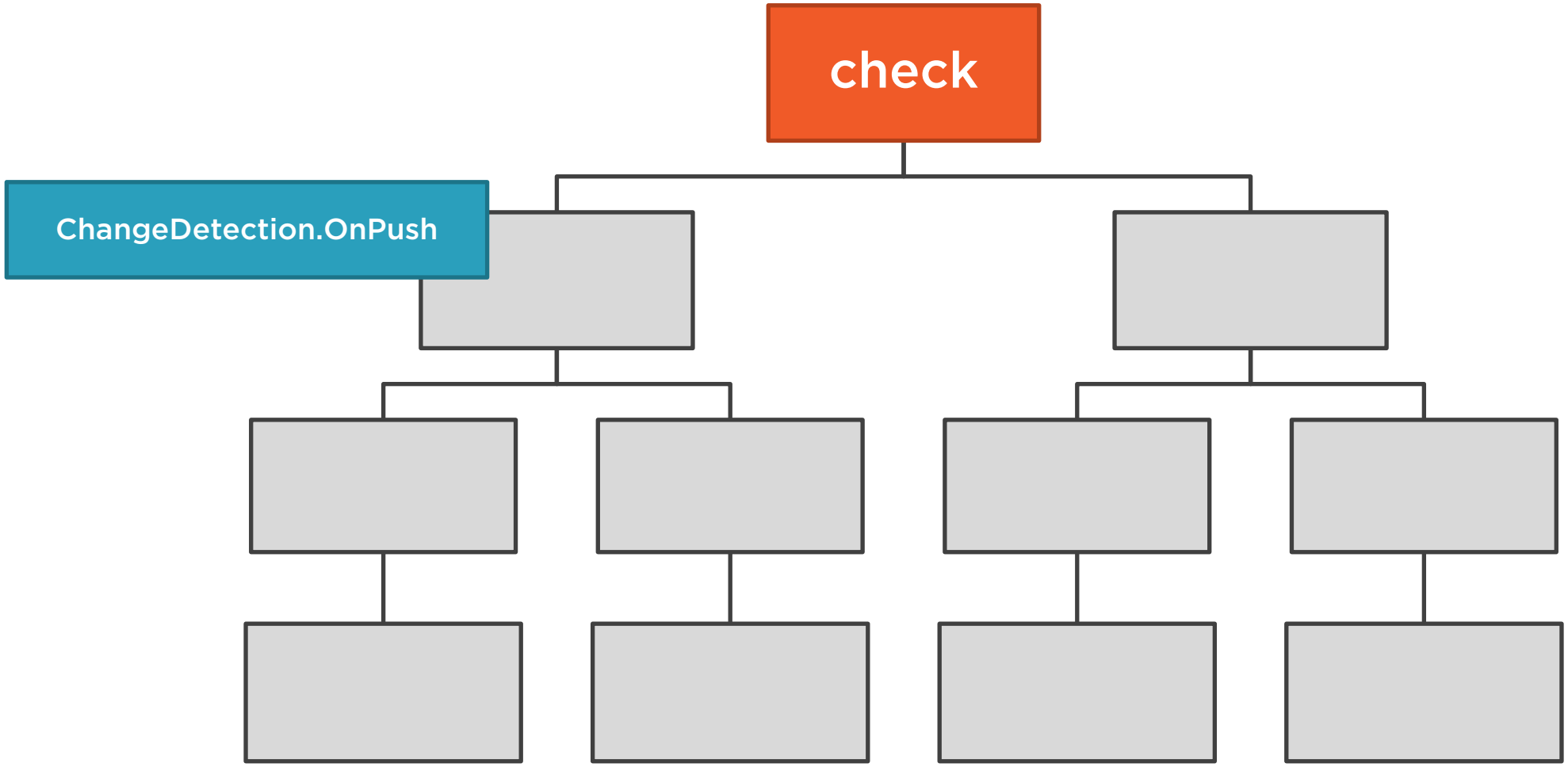
Change Detection



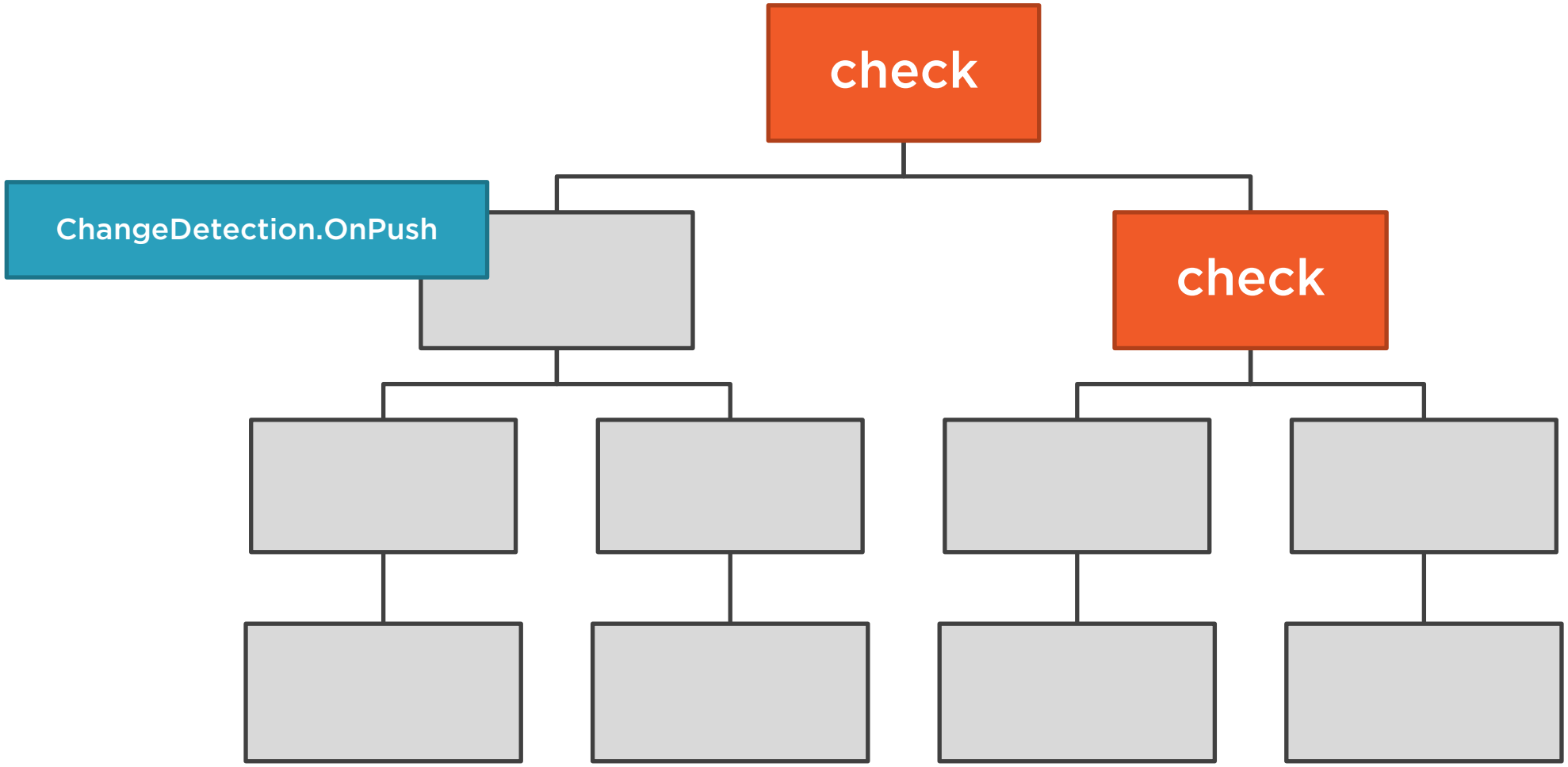
Change Detection - OnPush



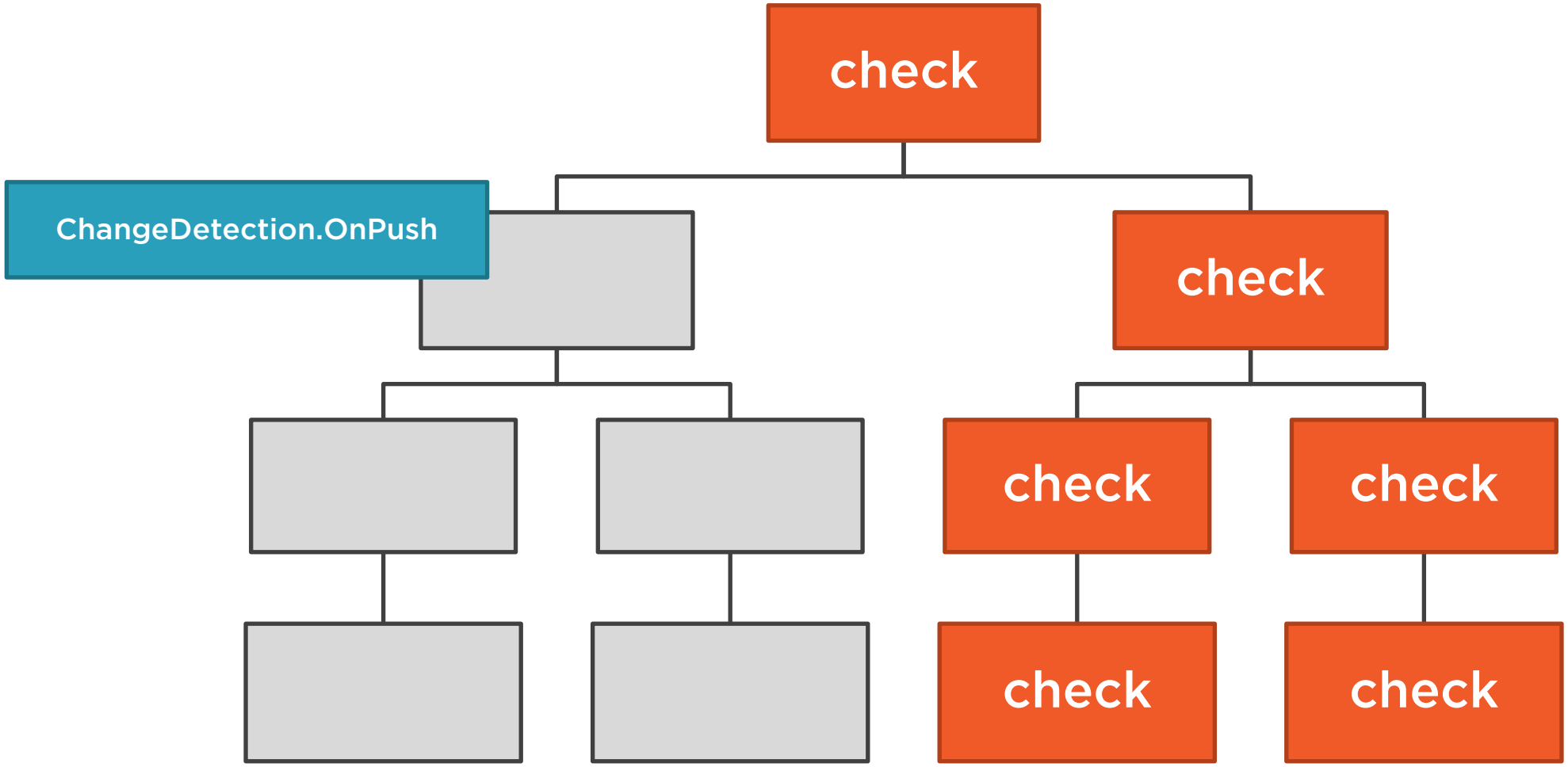
Change Detection - OnPush



Change Detection - OnPush



Change Detection - OnPush



OnPush Change Detection

```
import {Component,OnInit,ChangeDetectionStrategy} from '@angular/core';


@Component({
  templateUrl:'./product-list.component.html',
  styleUrls:['./product-list.component.css'],
})
export class ProductListComponent implements OnInit{}
```

OnPush Change Detection

```
import {Component,OnInit,ChangeDetectionStrategy} from '@angular/core';

@Component({
  templateUrl:'./product-list.component.html',
  styleUrls:['./product-list.component.css'],
  changeDetection: ChangeDetectionStrategy.OnPush
})
export class ProductListComponent implements OnInit{}
```

OnPush Change Detection Strategy

OnPush Change Detection

```
import {Component,OnInit,ChangeDetectionStrategy} from '@angular/core';


@Component({
  templateUrl:'./product-list.component.html',
  styleUrls:['./product-list.component.css'],
  changeDetection: ChangeDetectionStrategy.Default
})
export class ProductListComponent implements OnInit{}
```

Default Change Detection Strategy

Demo

Change detection OnPush

product.reducer.ts

APM-Demo4 > src > app > products > state > product.reducer.ts > getProducts > createSelector() callback

```
1  import { Product } from '../product';…
7
8  export interface State extends AppState.State {
9    products: ProductState;
10 }
11
12 export interface ProductState {…
17 }
18
19 const initialState: ProductState = {…
24 };
25
26 const getProductFeatureState = createFeatureSelector<ProductState>('products');
27
28 export const getShowProductCode = createSelector(
29   getProductFeatureState,
30   state => state.showProductCode
31 );
32
33 export const getCurrentProductId = createSelector(
34   getProductFeatureState,
35   state => state.currentProductId
36 );
37
38 export const getCurrentProduct = createSelector(
39
40   getCurrentProductId,
```

0 0 Ln 58, Col 26 Spaces: 2 UTF-8 LF TypeScript 3.8.3

product.reducer.ts

APM-Demo4 > src > app > products > state > product.reducer.ts > getProducts > createSelector() callback

```
1  import { Product } from '../product';…
7
8  export interface State extends AppState.State {
9    products: ProductState;
10 }
11
12 export interface
17 }
18
19 const initialStat
24 };
25
26 const getProductF
27
28 export const getShowProductCode = createSelector(
29   getProductFeatureState,
30   state => state.showProductCode
31 );
32
33 export const getCurrentProductId = createSelector(
34   getProductFeatureState,
35   state => state.currentProductId
36 );
37
38 export const getCurrentProduct = createSelector(
39   getProductFeatureState,
40   getCurrentProductId,
```

```
export interface State extends AppState.State {
  products: ProductState;
  inventory: InventoryState;
}
```

0 0 Ln 58, Col 26 Spaces: 2 UTF-8 LF TypeScript 3.8.3

Barrel

A way to rollup exports from several modules into a single convenience module. The barrel itself is a module file that re-exports selected exports of other modules.

Re-exporting with a Index.ts File

app/index.ts

```
export { Foo } from './app/foo';
export { Bar } from './app/bar';
export * as Baz from './app/baz';
```

Consumer

```
import { Foo, Bar, Baz } from './app'; // index.ts implied by convention
```

EXPLORER: PLURALSI...

- src
  - app
    - home
    - products
    - shared
    - state
      - product
        - actions
        - index.ts
        - product-data.ts
        - product-state.module.ts
        - product.effects.ts
        - product.reducer.ts
        - product.service.ts
        - product.ts
      - user
      - app.state.ts
    - user
    - app-routing.module.ts
    - app.component.css
    - app.component.html
    - app.component.ts
    - app.module.ts
  - assets
  - environments

TS index.ts

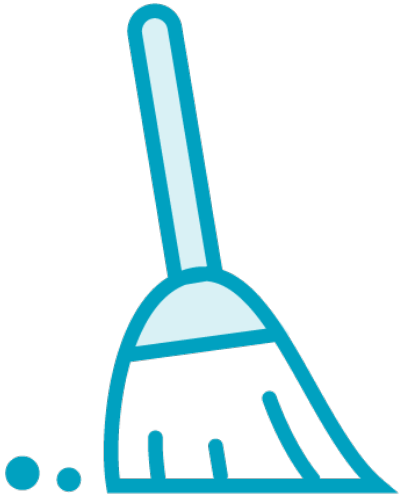
```
import { createFeatureSelector, createSelector } from '@ngrx/store';
import * as AppState from '../../state/app.state';
import { ProductState } from './product.reducer';
```

0 0 Ln 26, Col 21 Spaces: 4 UTF-8 CRLF TypeScript 3.8.3

Benefits of State Index.ts Files

Public API for state

Separation of concerns



Cleaner code

Demo

Adding an index.ts to the state folder

Actions should capture events not commands

TS product.actions.ts

```
6   );
7
8   export const setCurrentProduct = createAction(
9     '[Product] Set Current Product',
10    props<{ currentProductId: number }>()
11  );
12
13  export const clearCurrentProduct = createAction(
14    '[Product] Clear Current Product'
15  );
16
17  export const initializeCurrentProduct = createAction(
18    '[Product] Initialize Current Product'
19  );
20
21  export const loadProducts = createAction(
22    '[Product] Load'
23  );
24
25  export const loadProductsSuccess = createAction(
26    '[Product] Load Success',
27    props<{ products: Product[] }>()
28  );
29
30  export const loadProductsFailure = createAction(
31    '[Product] Load Fail',
32    props<{ error: string }>()
33  );
34
```

⊗ 0 ⚠ 0 Ln 29, Col 1 Spaces: 2 UTF-8 CRLF TypeScript 3.8.3 Prettier: ✓

product-api.actions.ts

```
6   );
7
8   export const setCurrentProduct = createAction(
9     '[Product Page] Set Current Product',
10    props<{ currentProductId: number }>()
11  );
12
13  export const clearCurrentProduct = createAction(
14    '[Product Page] Clear Current Product'
15  );
16
17  export const initializeCurrentProduct = createAction(
18    '[Product Page] Initialize Current Product'
19  );
20
21  export const loadProducts = createAction(
22    '[Product Page] Load'
23  );
24
25  export const loadProductsSuccess = createAction(
26    '[Product API] Load Success',
27    props<{ products: Product[] }>()
28  );
29
30  export const loadProductsFailure = createAction(
31    '[Product API] Load Fail',
32    props<{ error: string }>()
33  );
34
```

product-page.actions.ts

```
import { Product } from '../product';
import { createAction, props } from '@ngrx/store';

export const toggleProductCode = createAction(
  '[Product Page] Toggle Product Code'
);

export const setCurrentProduct = createAction(
  '[Product Page] Set Current Product',
  props<{ product: Product }>()
);

export const clearCurrentProduct = createAction(
  '[Product Page] Clear Current Product'
);

export const initializeCurrentProduct = createAction(
  '[Product Page] Initialize Current Product'
);

export const loadProducts = createAction(
  '[Product Page] Load'
);

export const updateProduct = createAction(
  '[Product Page] Update Product',
  props<{ product: Product }>()
);
```

product-api.actions.ts

```
import { Product } from '../product';
import { createAction, props } from '@ngrx/store';

export const loadProductsSuccess = createAction(
  '[Product API] Load Success',
  props<{ products: Product[] }>()
);

export const loadProductsFailure = createAction(
  '[Product API] Load Fail',
  props<{ error: string }>()
);

export const updateProductSuccess = createAction(
  '[Product API] Update Product Success',
  props<{ product: Product }>()
);

export const updateProductFailure = createAction(
  '[Product API] Update Product Fail',
  props<{ error: string }>()
);

export const createProductSuccess = createAction(
  '[Product API] Create Product Success',
  props<{ product: Product }>()
);

export const createProductFailure = createAction(
```

EXPLORER: PL...

- app
  - home
  - products
    - components
    - containers \ product-sh...
    - product.module.ts
  - shared
  - state
    - product
      - actions
        - index.ts
        - product-api.actions.ts
        - product-page.action...
      - index.ts
      - product-data.ts
      - product-state.module.ts
      - product.effects.ts
      - product.reducer.ts
      - product.service.ts
      - product.ts
    - user
    - app.state.ts
  - user
  - app-routing.module.ts

index.ts

```
1  import * as ProductPageActions from './product-page.actions';
2  import * as ProductApiActions from './product-api.actions';
3
4  export { ProductPageActions, ProductApiActions };
5
```

0 0 Ln 5, Col 1 Spaces: 4 UTF-8 CRLF TypeScript 3.8.3 Prettier

Demo

Grouping related actions into separate files

State Module

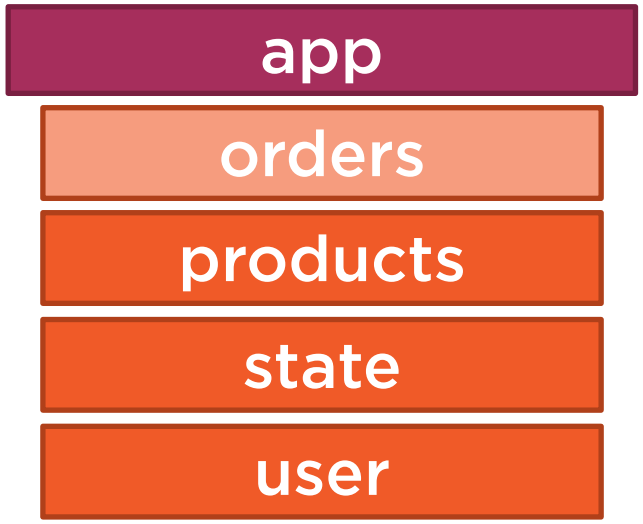
An Angular module that has just state logic and no presentational logic like components.

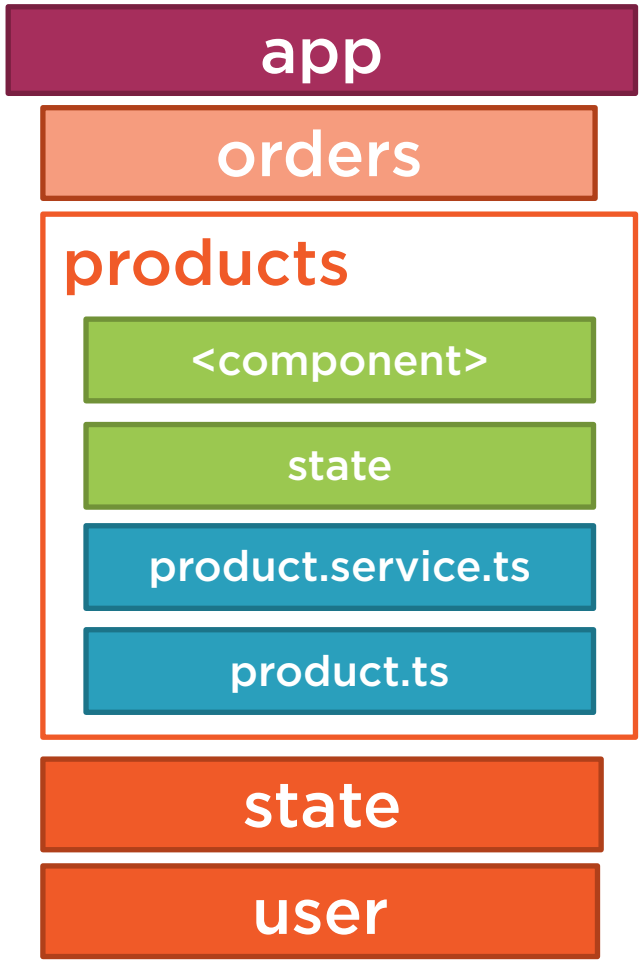
app

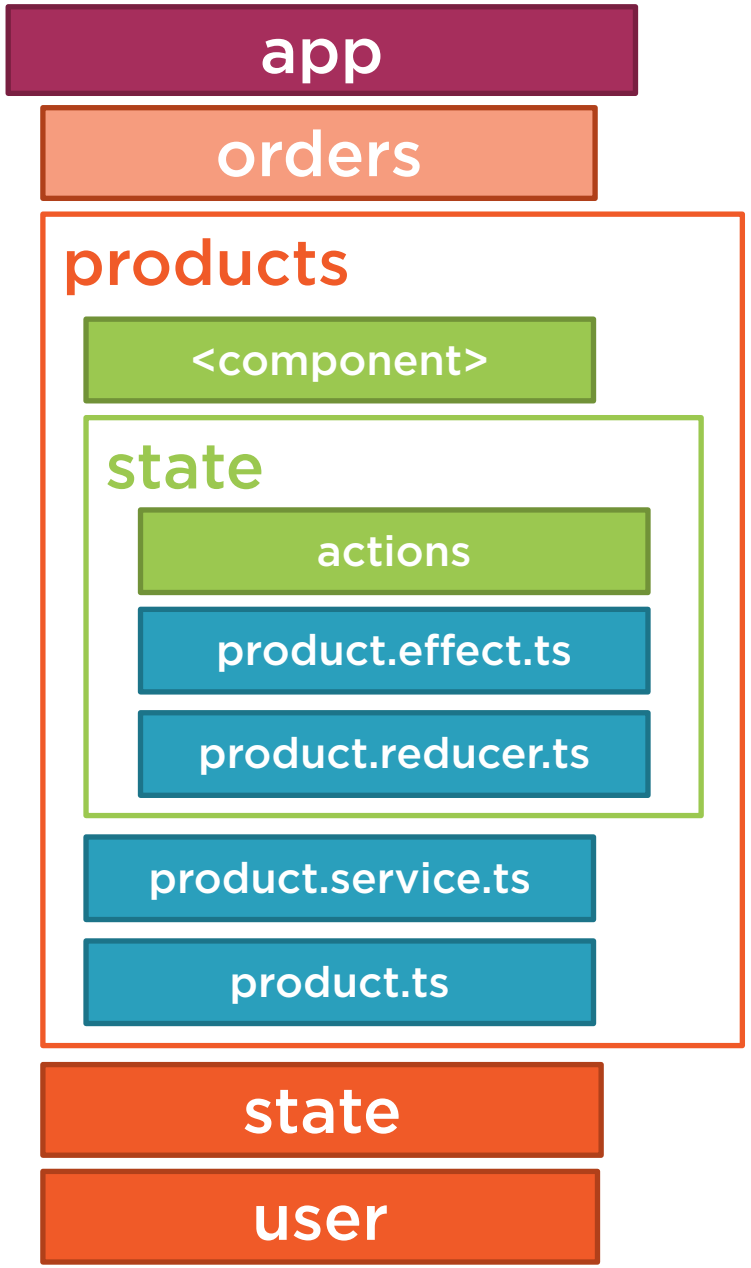
products

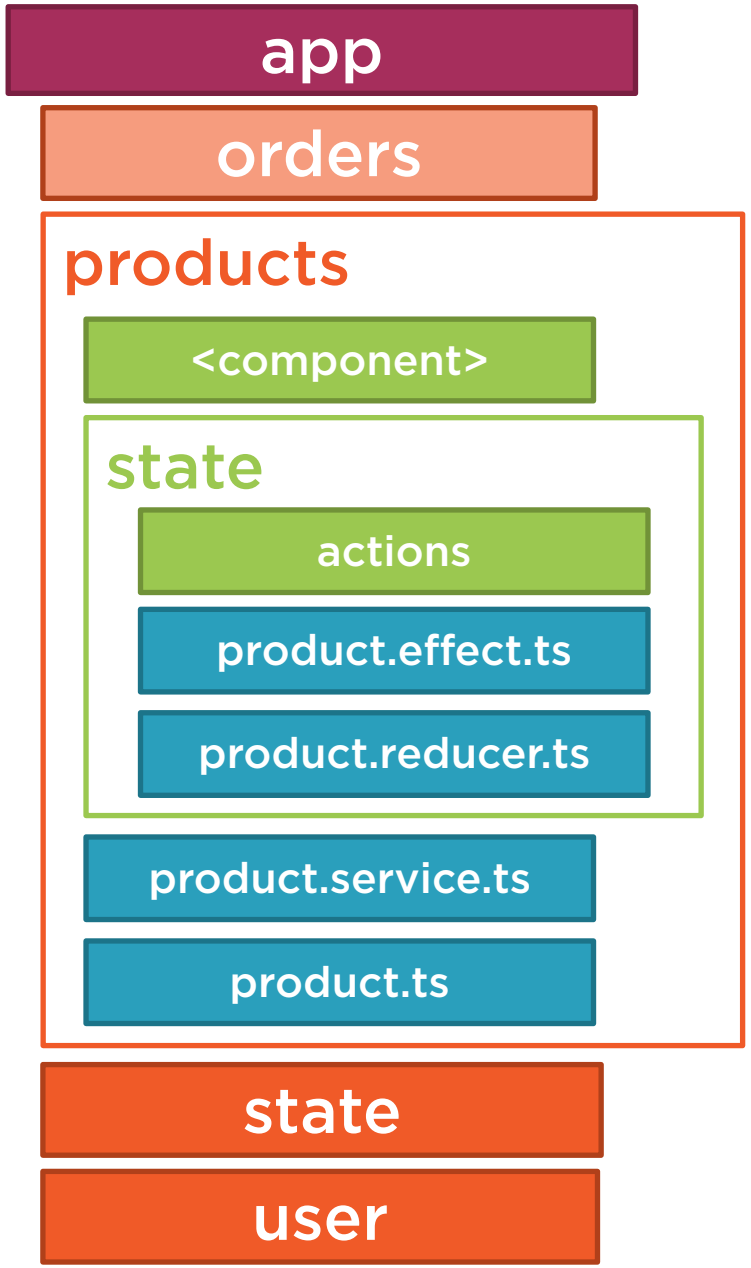
state

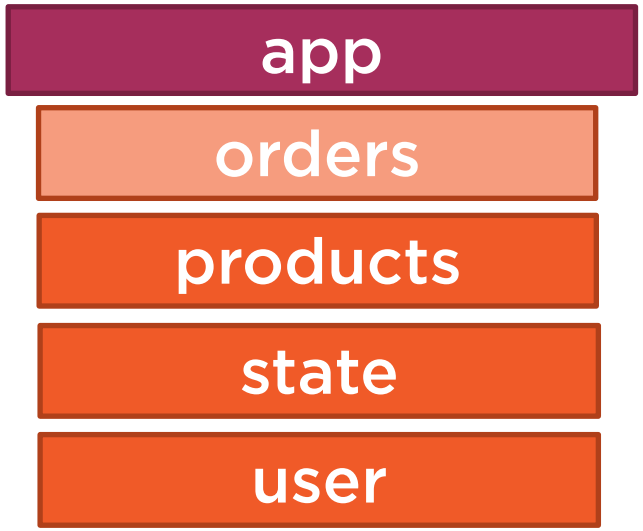
user

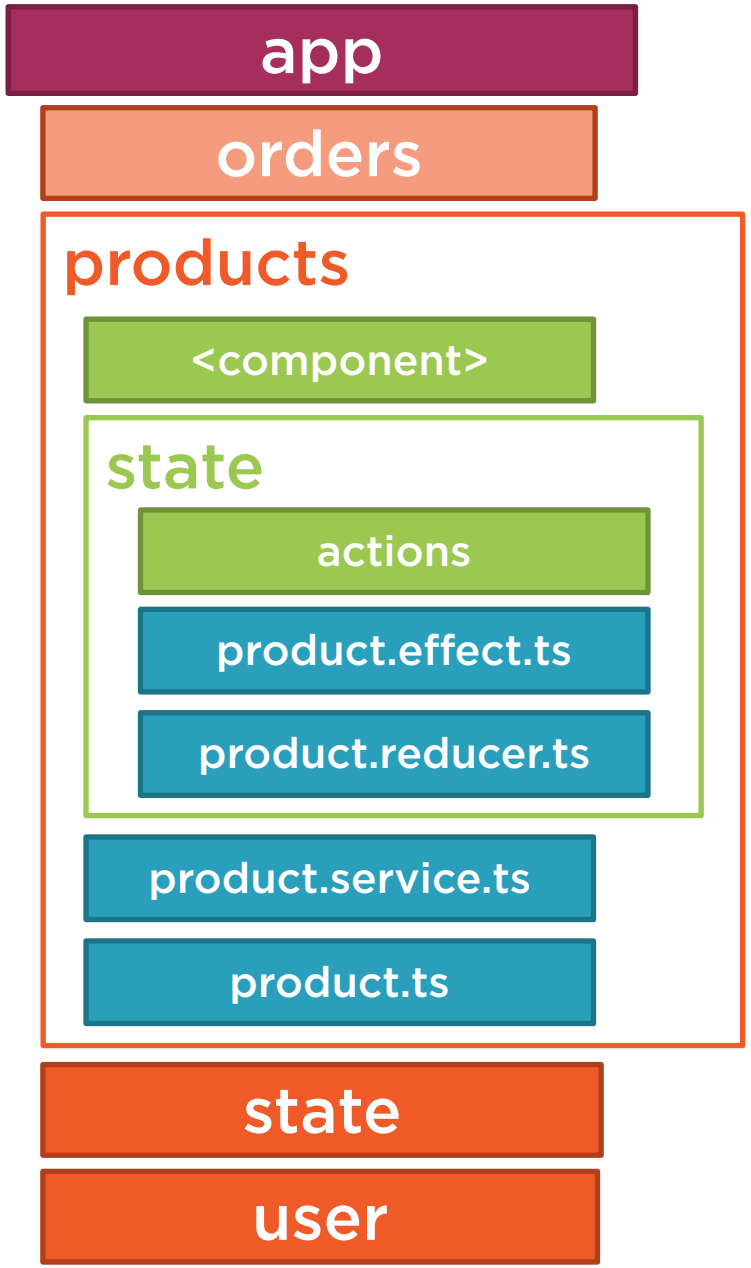

app
orders
products
state
user

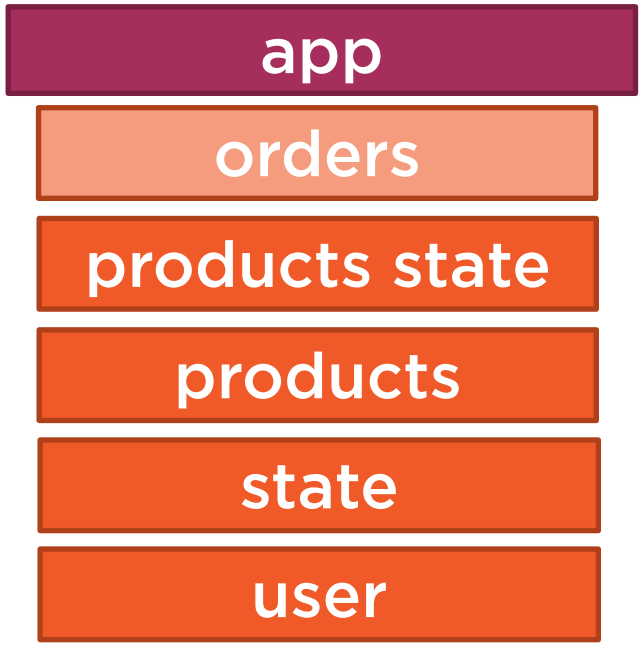

app
orders
products
<component>
state
product.service.ts
product.ts
state
user

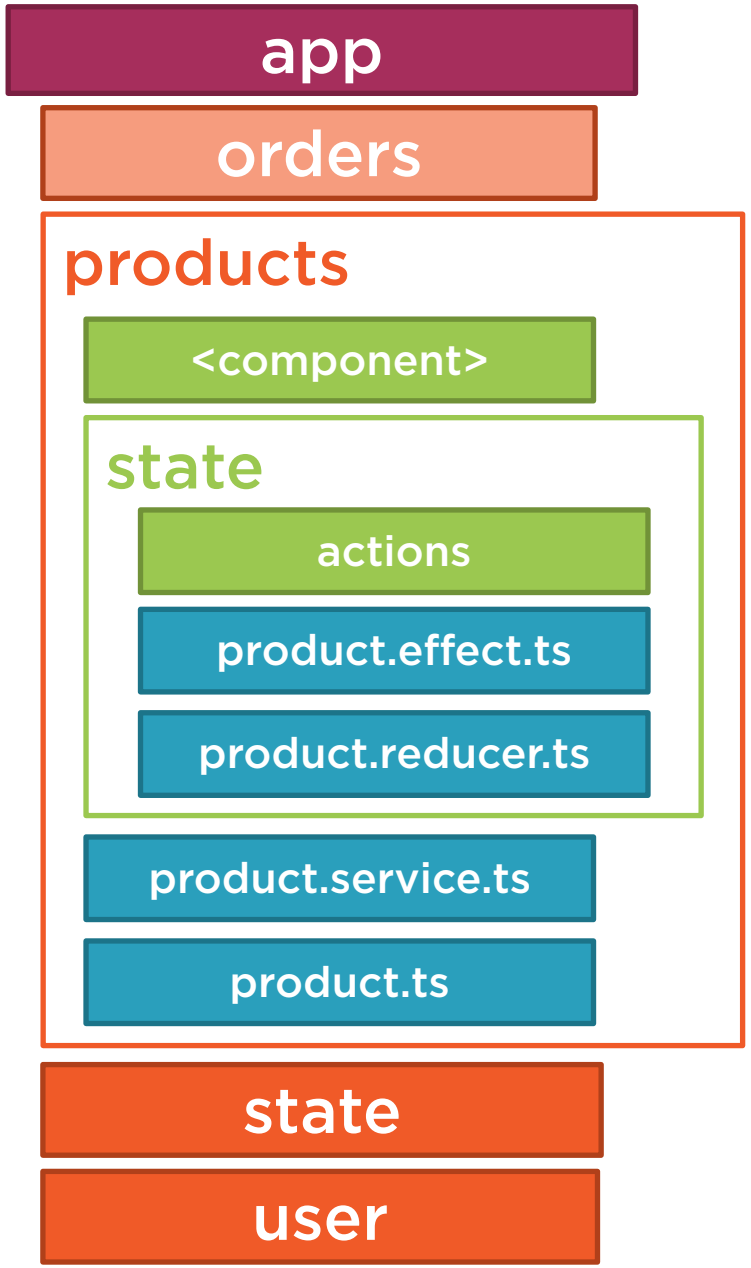

app
orders
products
<component>
state
actions
product.effect.ts
product.reducer.ts
product.service.ts
product.ts
state
user


app
orders
products
<component>
state
actions
product.effect.ts
product.reducer.ts
product.service.ts
product.ts
state
user

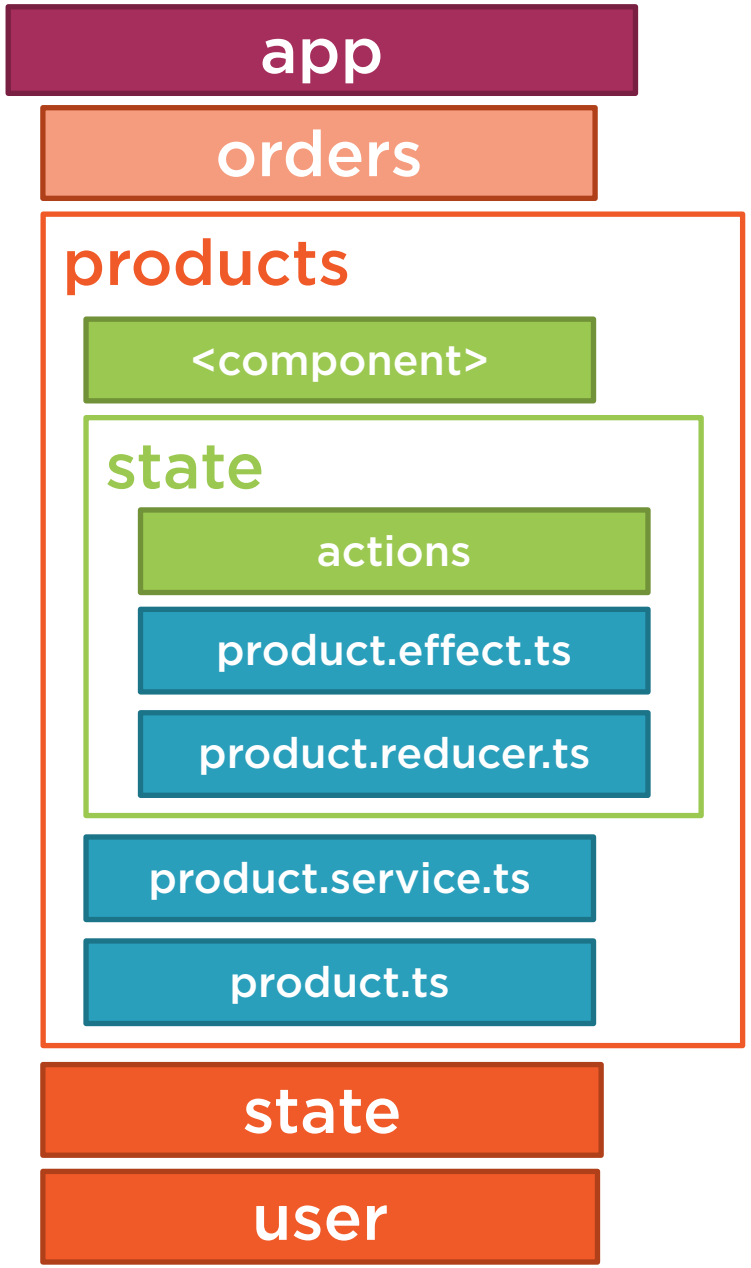

app
orders
products
state
user

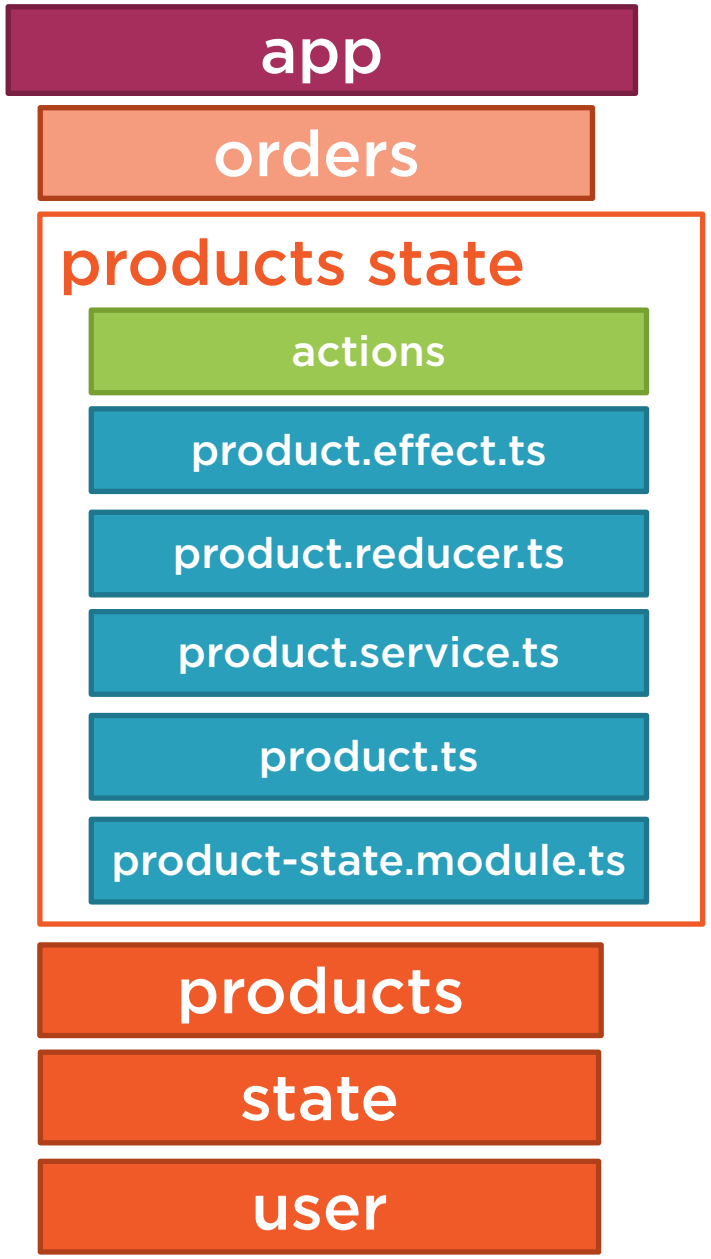

app
orders
products
<component>
state
actions
product.effect.ts
product.reducer.ts
product.service.ts
product.ts
state
user

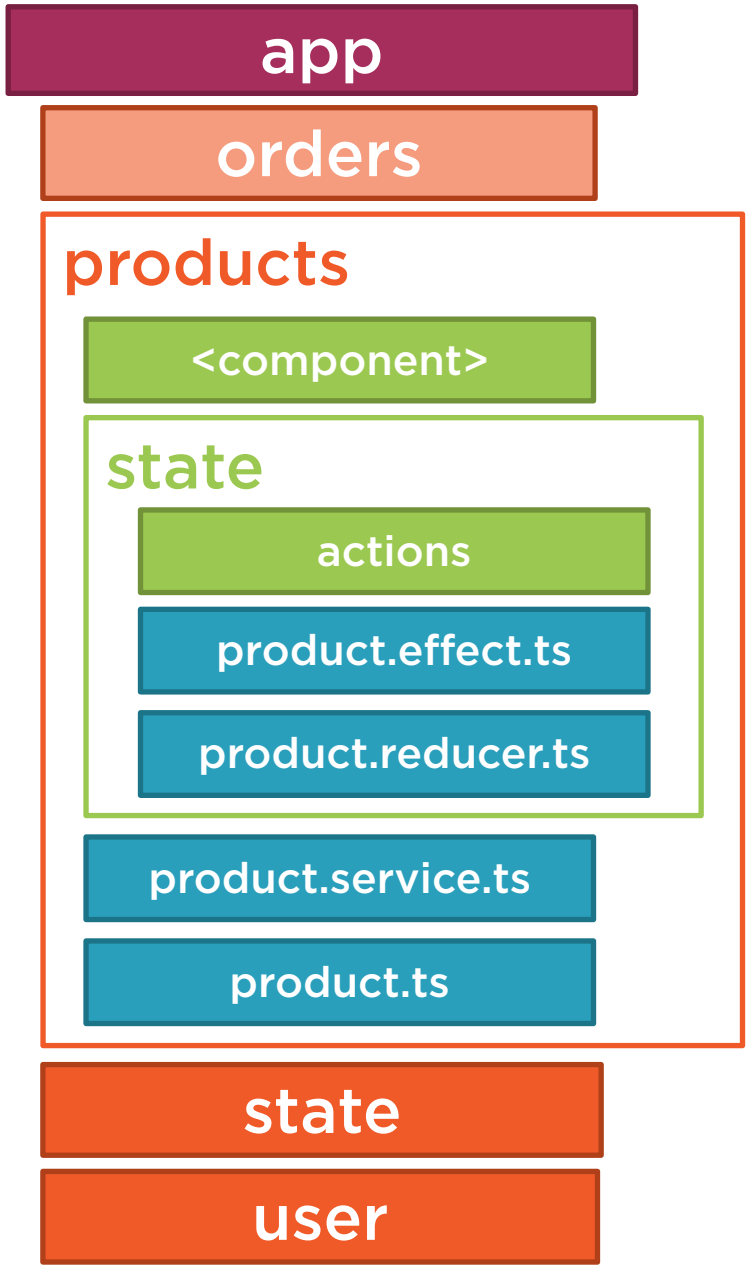

app
orders
products state
products
state
user

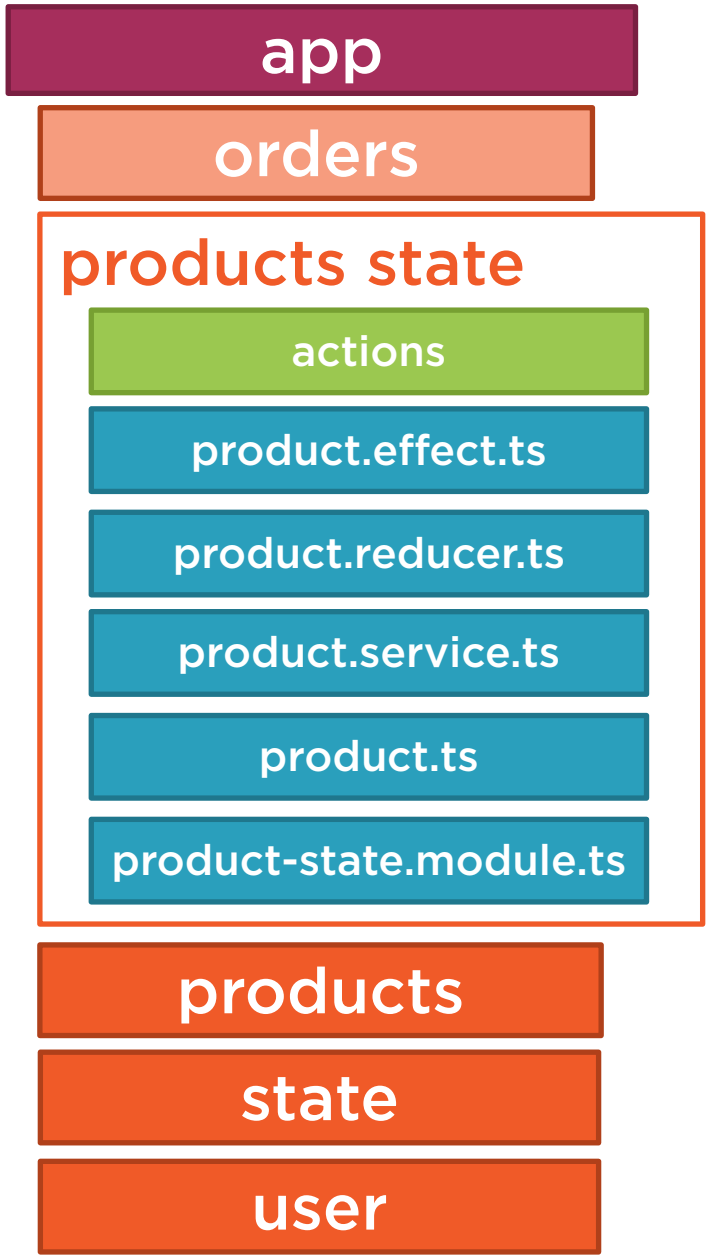

app
orders
products
<component>
state
actions
product.effect.ts
product.reducer.ts
product.service.ts
product.ts
state
user

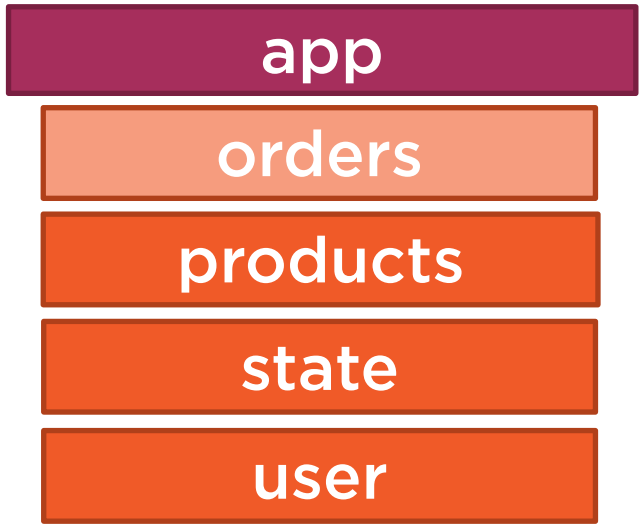

app
orders
products state
products
state
user

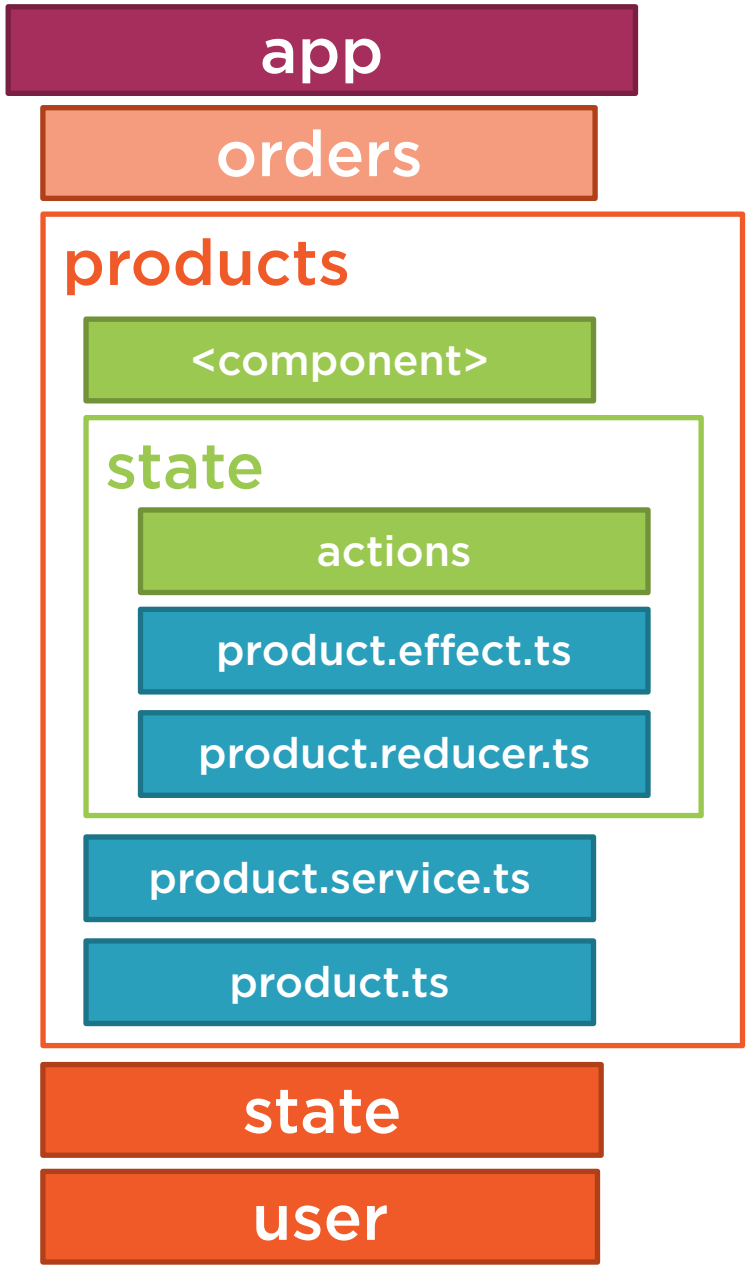

app
orders
products
<component>
state
actions
product.effect.ts
product.reducer.ts
product.service.ts
product.ts
state
user

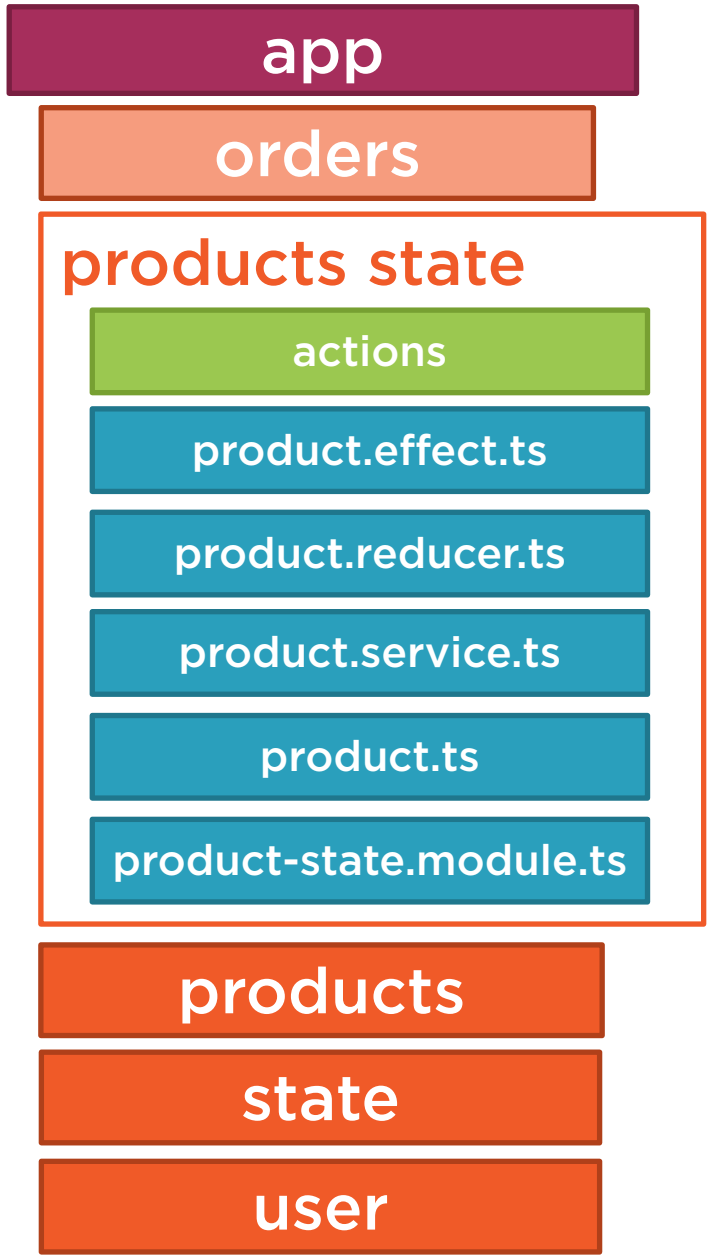

app
orders
products state
actions
product.effect.ts
product.reducer.ts
product.service.ts
product.ts
product-state.module.ts
products
state
user

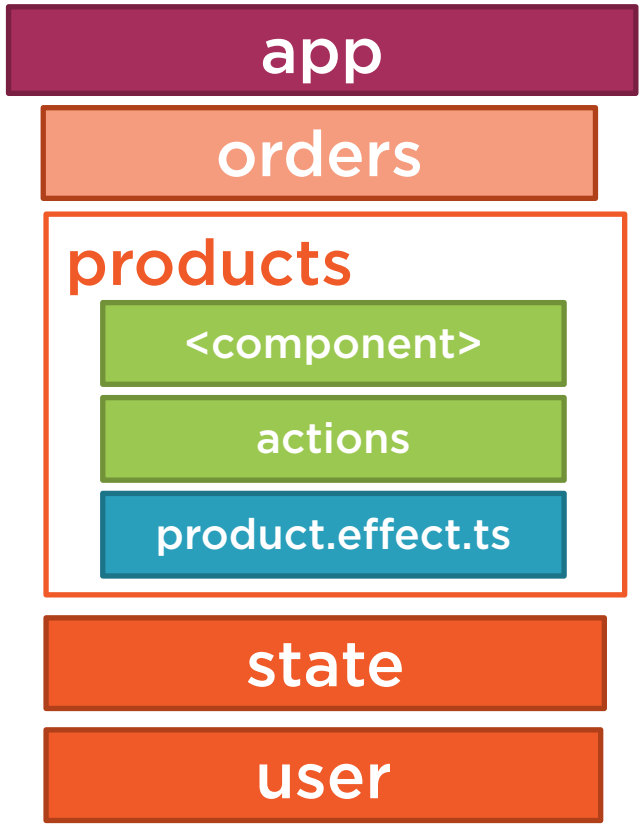

app
orders
products
<component>
state
actions
product.effect.ts
product.reducer.ts
product.service.ts
product.ts
state
user

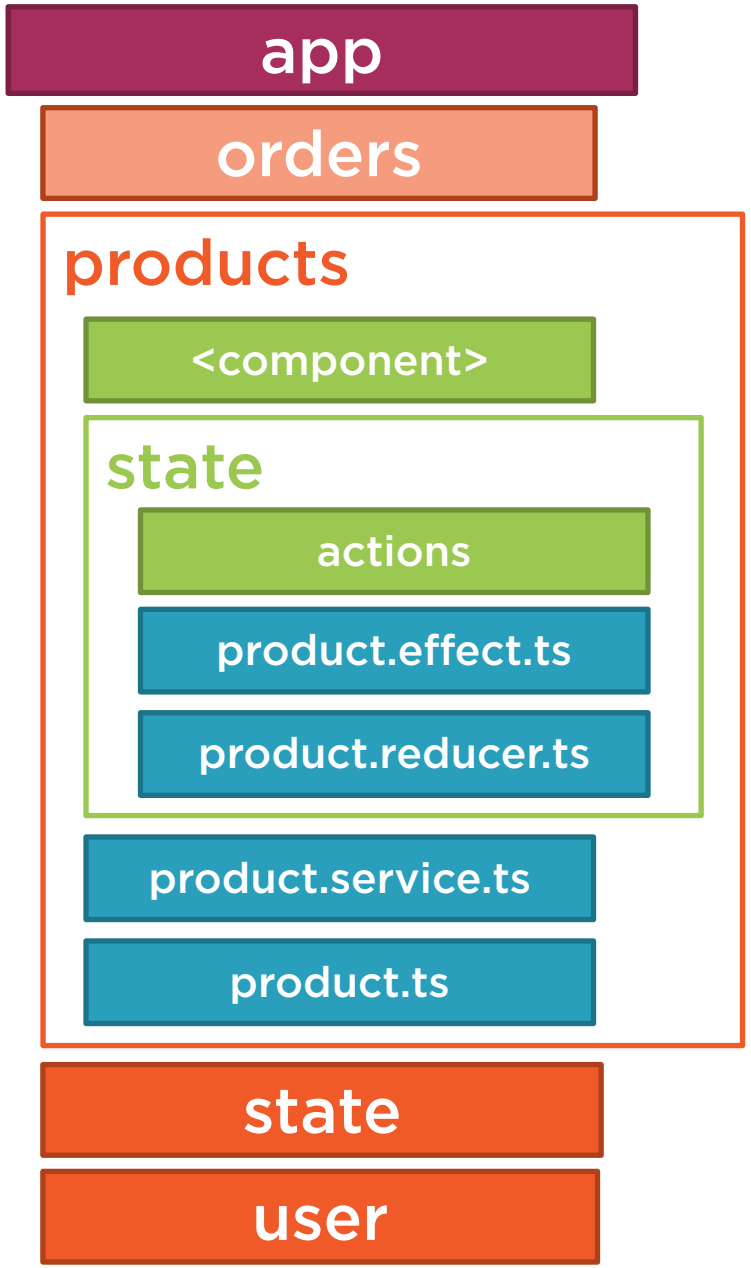

app
orders
products state
actions
product.effect.ts
product.reducer.ts
product.service.ts
product.ts
product-state.module.ts
products
state
user

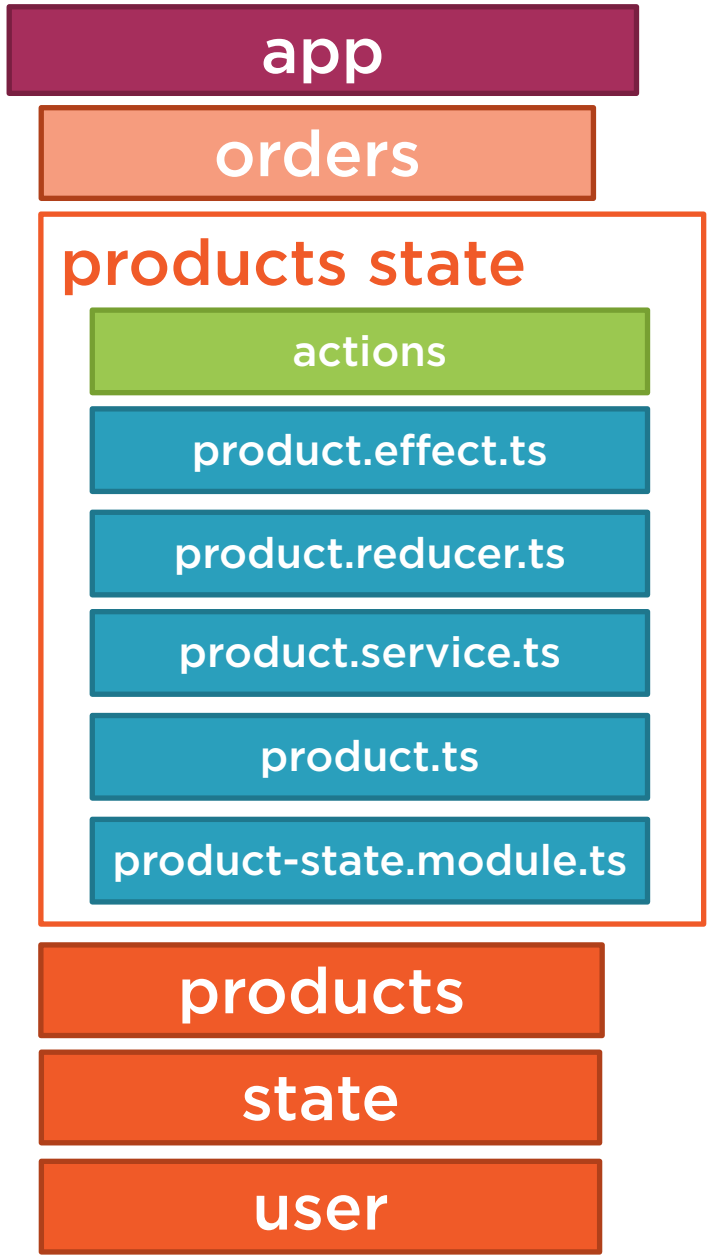

app
orders
products
state
user

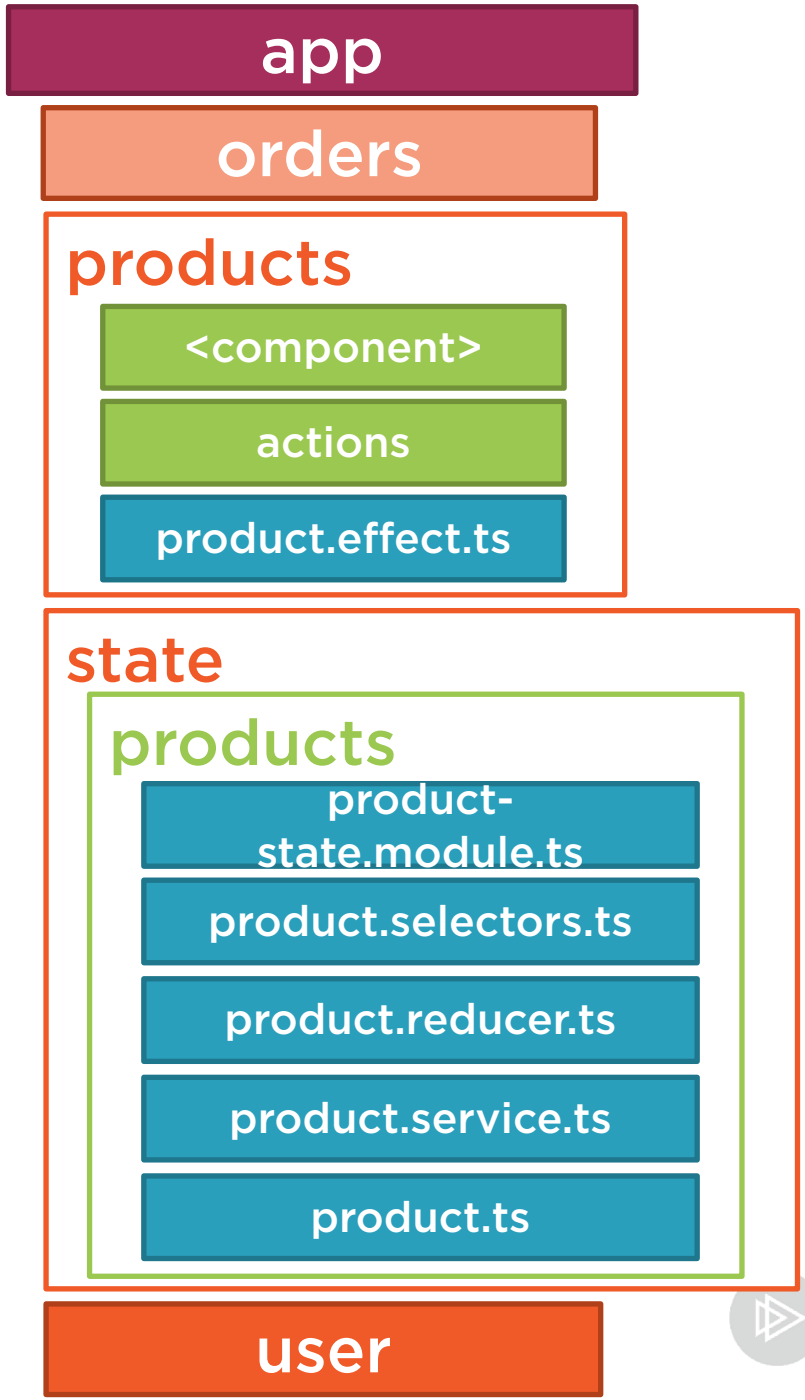

app
orders
products
<component>
state
actions
product.effect.ts
product.reducer.ts
product.service.ts
product.ts
state
user


app
orders
products state
actions
product.effect.ts
product.reducer.ts
product.service.ts
product.ts
product-state.module.ts
products
state
user

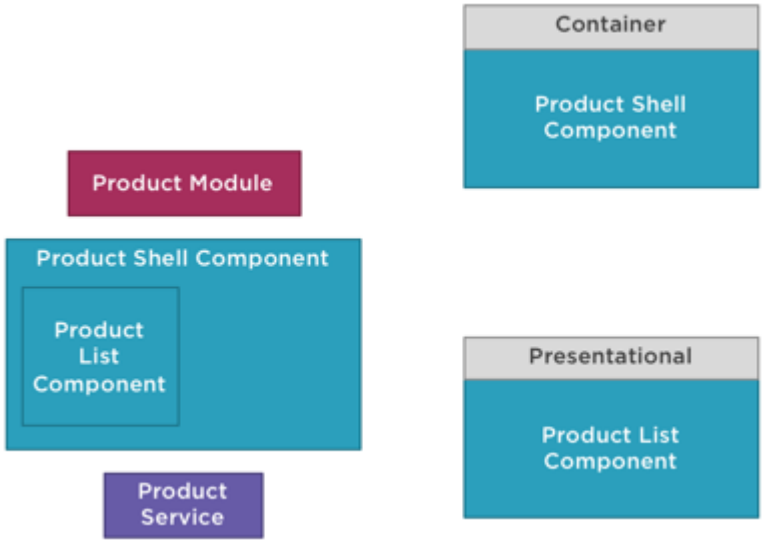

app
orders
products
<component>
actions
product.effect.ts
state
user


app
orders
products
<component>
state
actions
product.effect.ts
product.reducer.ts
product.service.ts
product.ts
state
user


app
orders
products state
actions
product.effect.ts
product.reducer.ts
product.service.ts
product.ts
product-state.module.ts
products
state
user


app
orders
products
<component>
actions
product.effect.ts
state
products
product-state.module.ts
product.selectors.ts
product.reducer.ts
product.service.ts
product.ts
user

Checklist: Container and Presentational Components



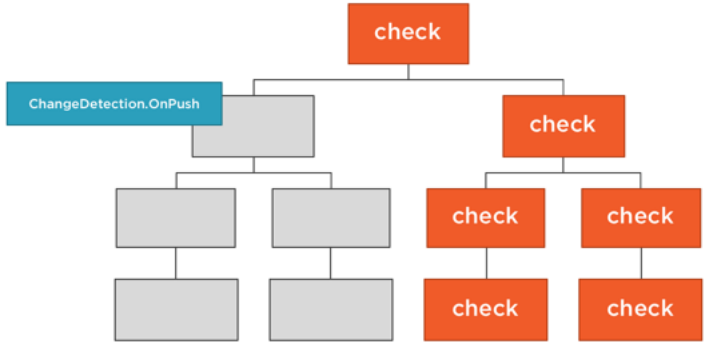
View performance

Separation of concerns

Composability

Easier testing

Checklist: ChangeDetection OnPush

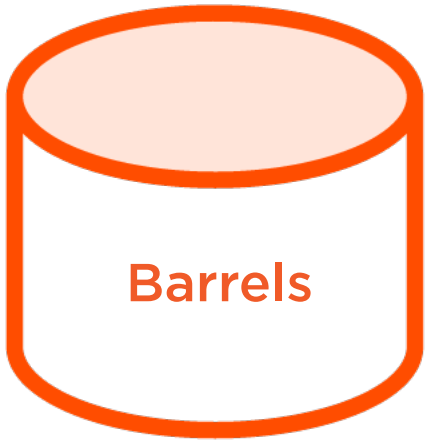


Skip change detection unless an @Input receives a new value or object reference

Add 'ChangeDetectionStrategy.OnPush' to all container component decorators

Easier when categorizing components into presentational or container components

Checklist: Barrels



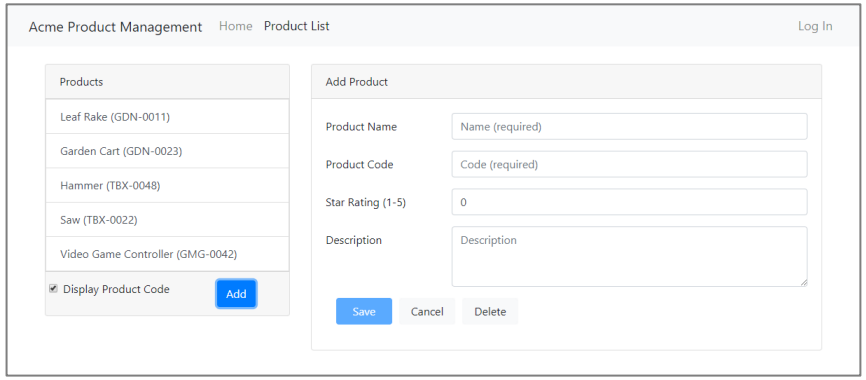
Rollup exports from several ECMAScript modules into a single module

Public APIs for feature state modules

To use barrels:

- Make index.ts file in each state folder
- Add selectors and state interfaces to index.ts
- Re-export other feature state for other modules

Homework: Presentational Component



Remove the injected store

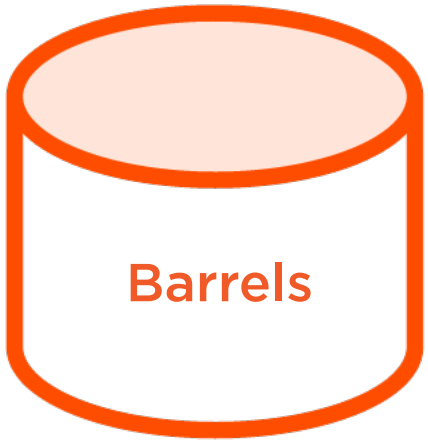
Pass all store state properties in as inputs

Move all dispatched actions to the Product Shell, called via emitted events

Add an OnChanges life cycle hook to listen for and call the patch form method on changes

https://github.com/DeborahK/Angular-NgRx-GettingStarted/tree/master/APM-Demo5

Homework: User Index.ts File



Add an index.ts file to the User state folder

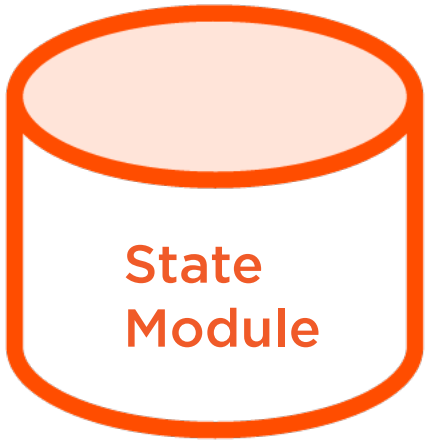
Copy the State interface and selectors to the index.ts file

Add back any missing import statements

Change any files import statements that use the state interface or selectors

https://github.com/DeborahK/Angular-NgRx-GettingStarted/tree/master/APM-Demo5

Homework: User State Module



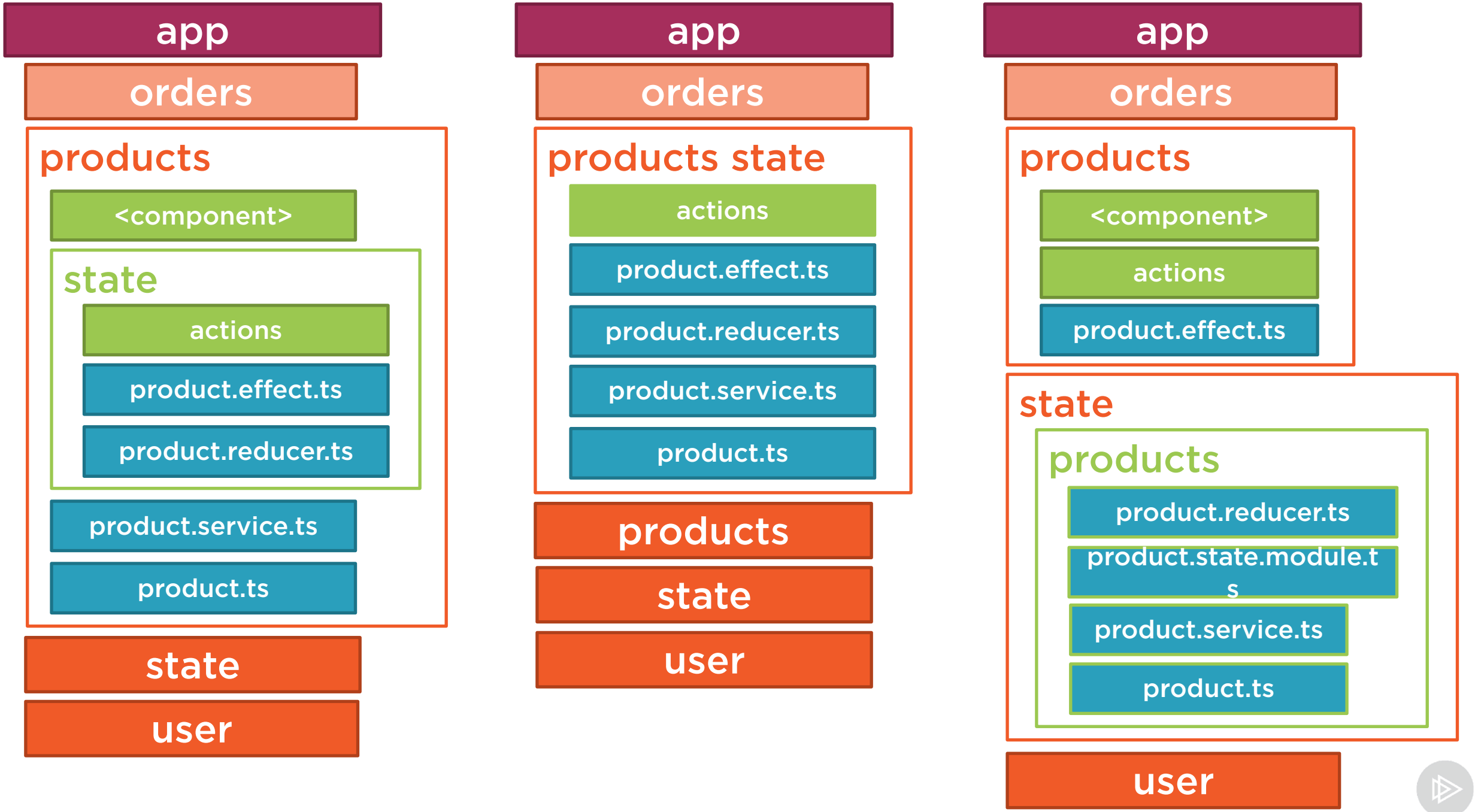
a

b

c

d

https://github.com/DeborahK/Angular-NgRx-GettingStarted/tree/master/APM-Demo5


app
orders
products
<component>
state
actions
product.effect.ts
product.reducer.ts
product.service.ts
product.ts
state
user
app
orders
products state
actions
product.effect.ts
product.reducer.ts
product.service.ts
product.ts
products
state
user
app
orders
products
<component>
actions
product.effect.ts
state
products
product.reducer.ts
product.state.module.ts
product.service.ts
product.ts
user