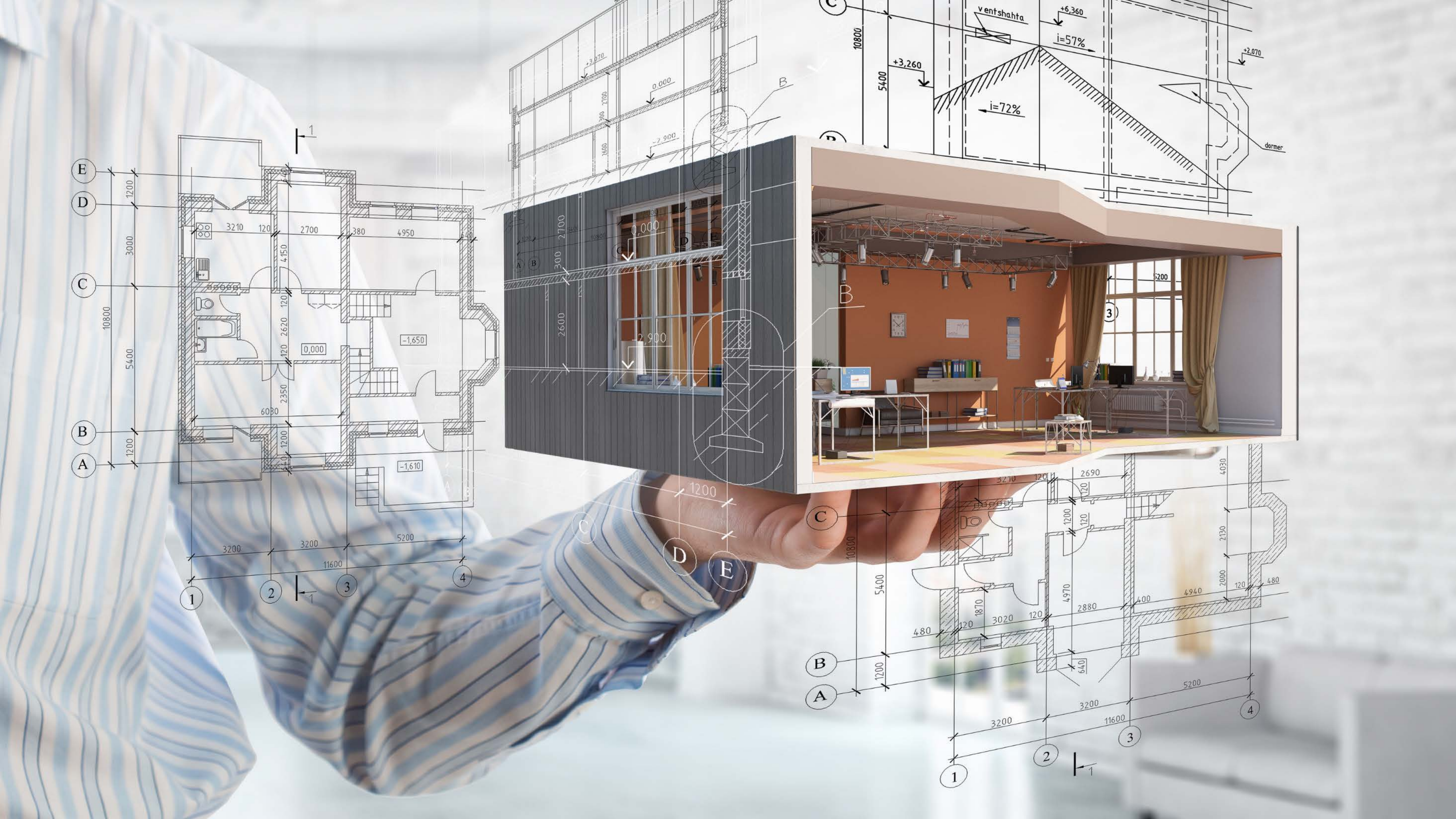
Architectural Considerations

Duncan Hunter
CONSULTANT | SPEAKER | AUTHOR

@dunchunter duncanhunter.com.au


ventshahta
i=57%
i=72%
dormer

Module Overview

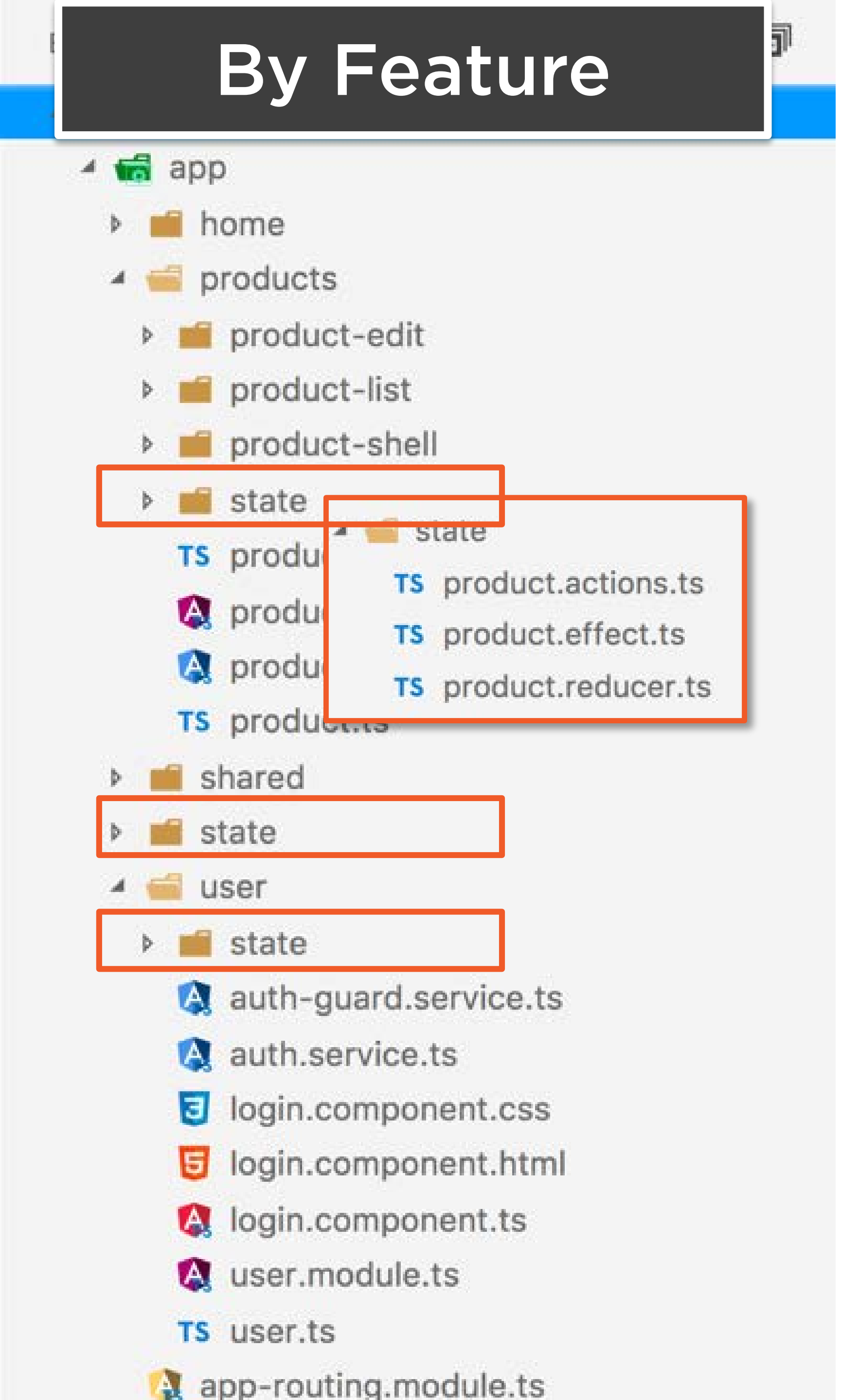
Folder structure

Presentational and container components

Change detection strategy OnPush

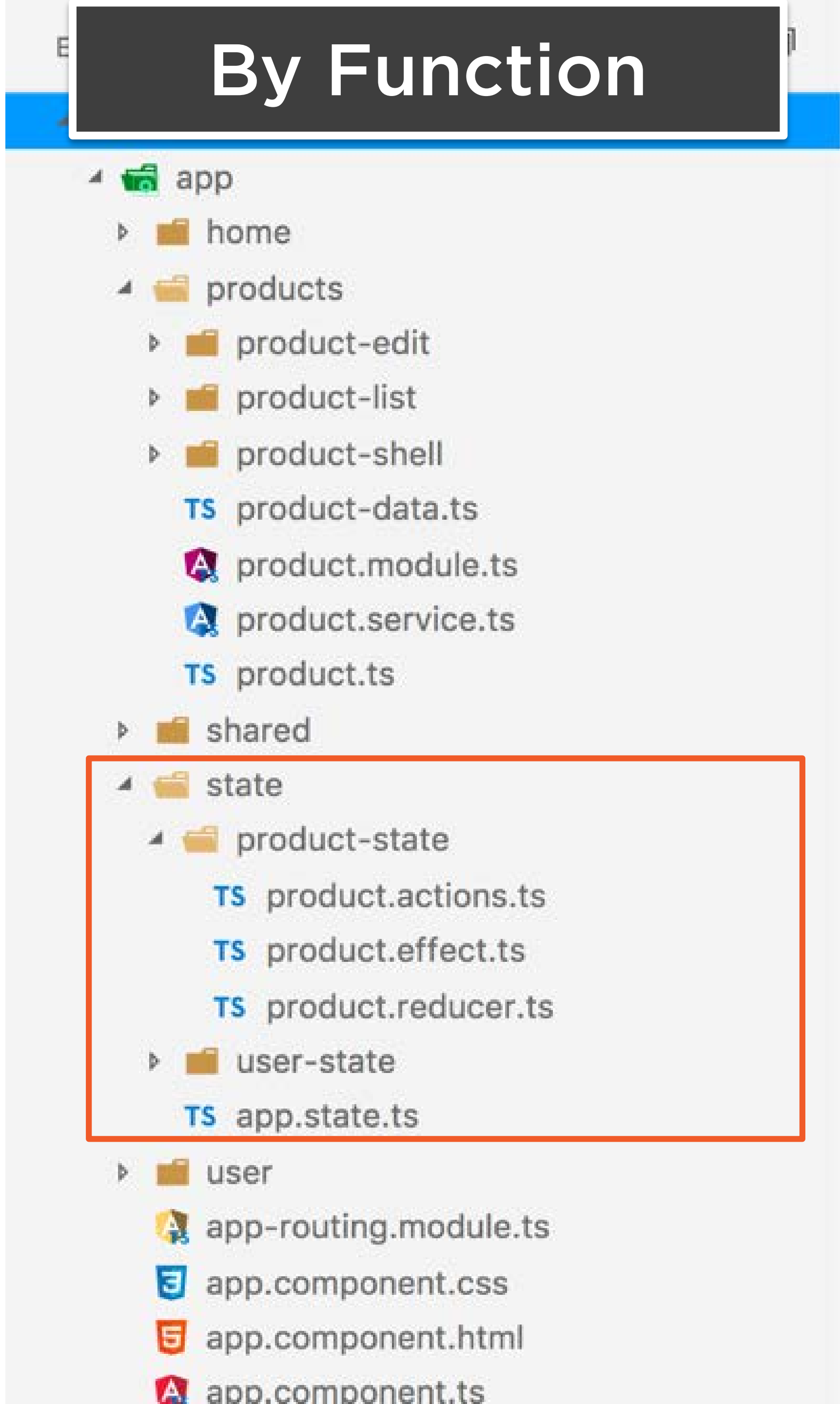
Adding an index.ts file to our state folders

Folders by feature or function?



By Feature

```
app
  home
  products
    product-edit
    product-list
    product-shell
    state
      state
        product.actions.ts
        product.effect.ts
        product.reducer.ts
    produ
    produ
    produ
    product.ts
  shared
  state
  user
    state
    auth-guard.service.ts
    auth.service.ts
    login.component.css
    login.component.html
    login.component.ts
    user.module.ts
    user.ts
  app-routing.module.ts
```



By Function

```
app
  home
  products
    product-edit
    product-list
    product-shell
    product-data.ts
    product.module.ts
    product.service.ts
    product.ts
  shared
  state
    product-state
      product.actions.ts
      product.effect.ts
      product.reducer.ts
    user-state
    app.state.ts
  user
  app-routing.module.ts
  app.component.css
  app.component.html
  app.component.ts
```

Benefits of Folders by Feature

Follows Angular style guide

Easy to find related files

Less cluttered

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

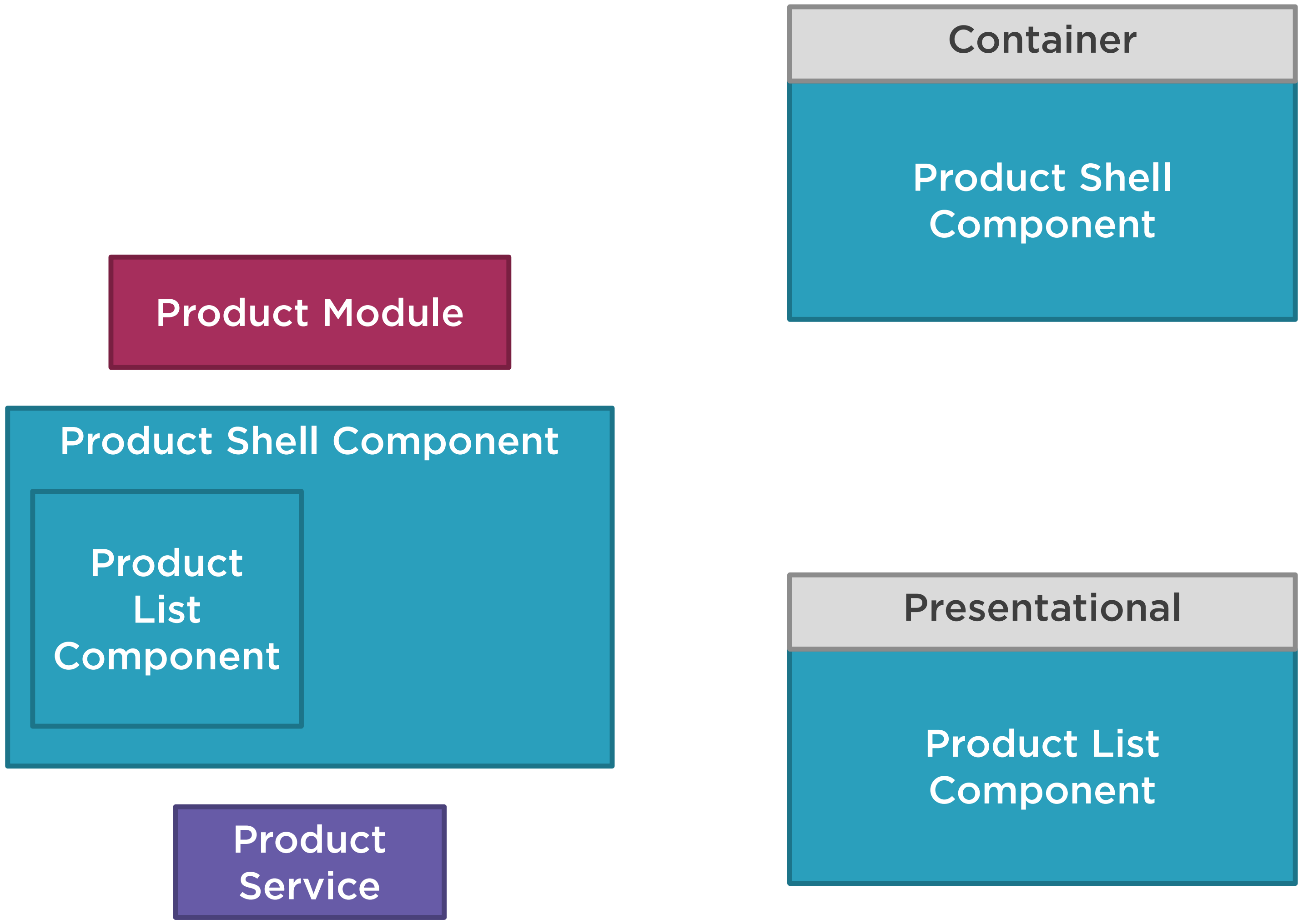
Benefits of Presentational and Container Components

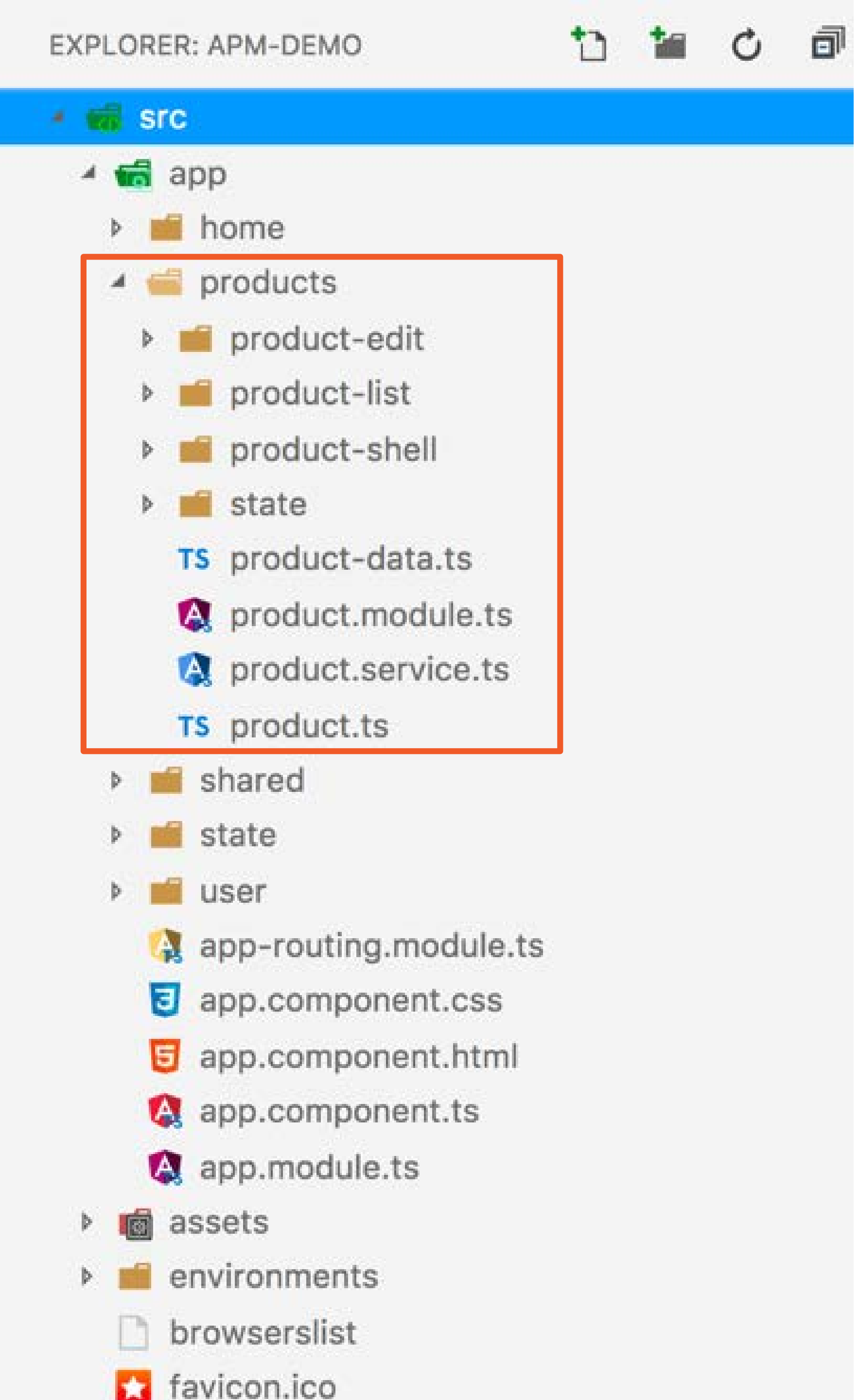
Performance

Composability

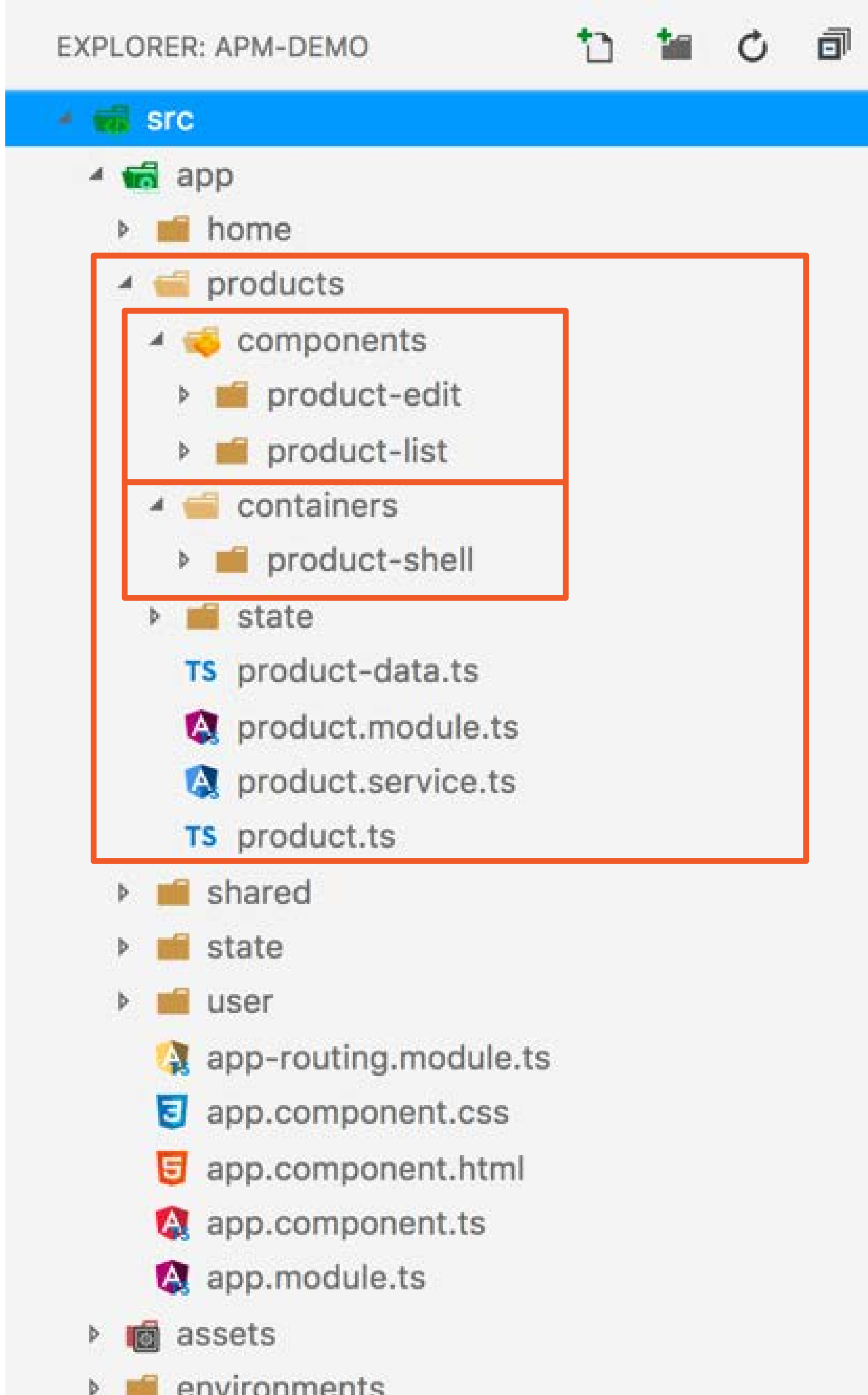
Easier to test

Sample Application Architecture





```
EXPLORER: APM-DEMO
src
  app
    home
    products
      product-edit
      product-list
      product-shell
      state
      product-data.ts
      product.module.ts
      product.service.ts
      product.ts
    shared
    state
    user
    app-routing.module.ts
    app.component.css
    app.component.html
    app.component.ts
    app.module.ts
  assets
  environments
  browserslist
  favicon.ico
```



```
EXPLORER: APM-DEMO
src
  app
    home
    products
      components
        product-edit
        product-list
      containers
        product-shell
      state
      product-data.ts
      product.module.ts
      product.service.ts
      product.ts
    shared
    state
    user
    app-routing.module.ts
    app.component.css
    app.component.html
    app.component.ts
    app.module.ts
  assets
  environments
```

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<fromProduct.State>
  ) { }

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

  checkChanged(value:boolean) {
   this.store.dispatch(...);
  }
  ...

}
```

Presentational and Container Components

Product Shell Component

```
export class ProductShellComponent
implements OnInit {

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

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

 checkChanged(value) {
   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() checked = new EventEmitter<boolean>();
 @Output() selected = new EventEmitter<Product>();
 @Output() initializeNewProduct =
           newEventEmitter<void>();

 checkChanged(value ) {
   this.checked.emit(value);
 }

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

 productSelected(product) {
   this.selected.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"
        (checked)="checkChanged($event)”
        (initializeNewProduct)="newProduct()"
        (selected)="productSelected($event)">
    </pm-product-list>
  </div>
</div>
```

Demo

Container components

Demo

Presentational components

Benefits of Container and Presentational Components

Performance

Composability

Easier to test

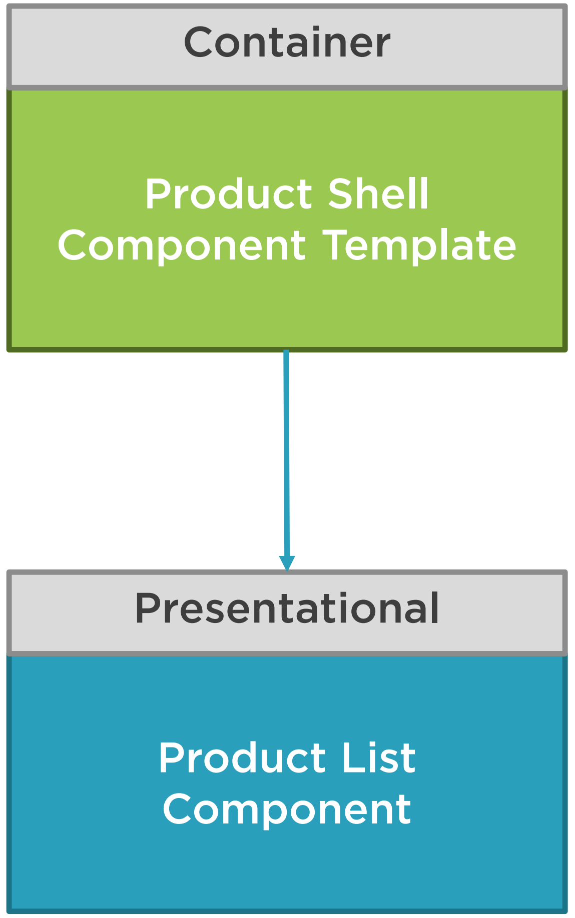
ChangeDetectionStrategy.OnPush

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

ChangeDetectionStrategy.OnPush

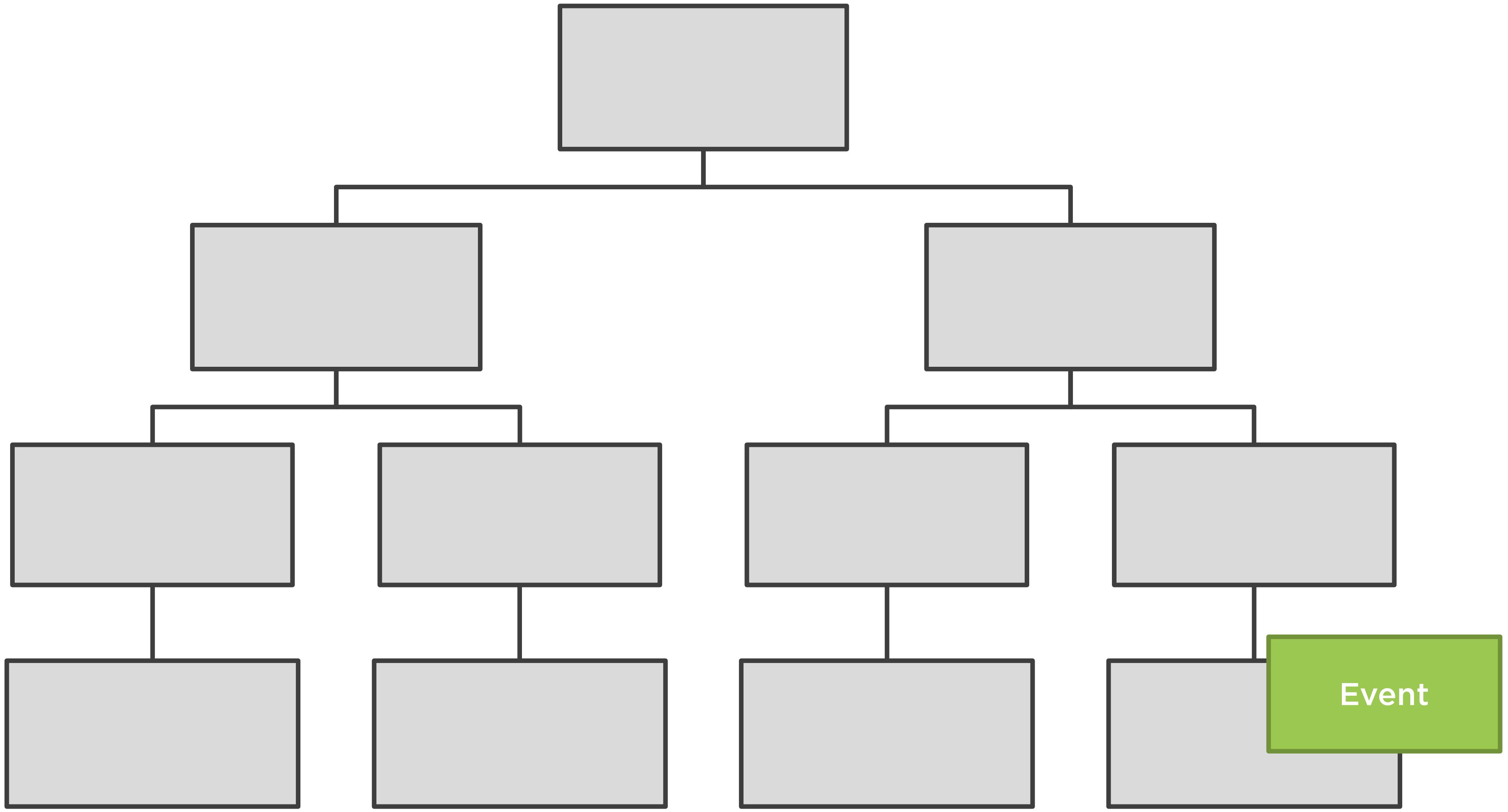
```
<pm-product-list
   [products]="products$ | async">
</pm-product-list>
```

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

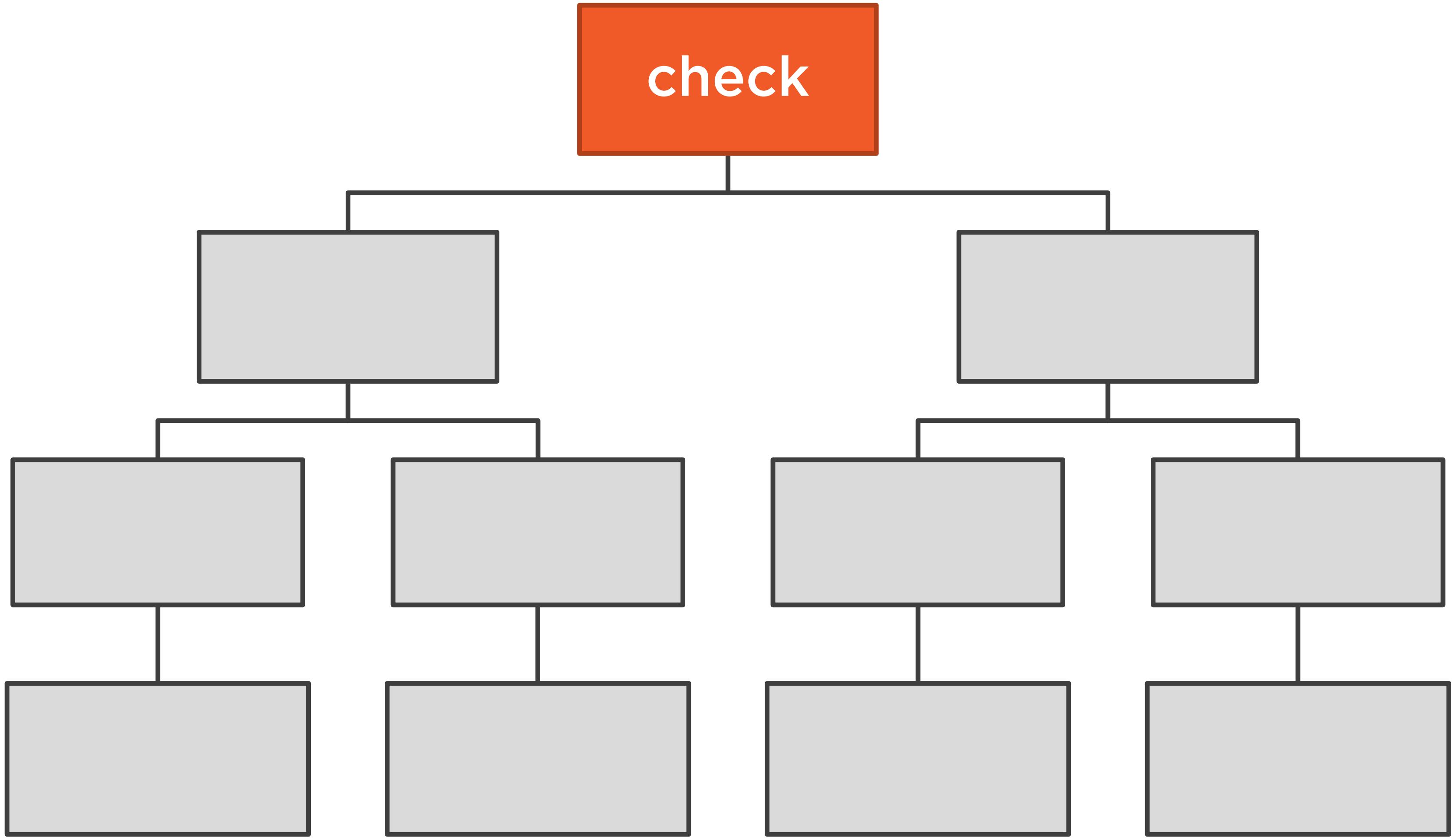


ChangeDetectionStrategy.Default

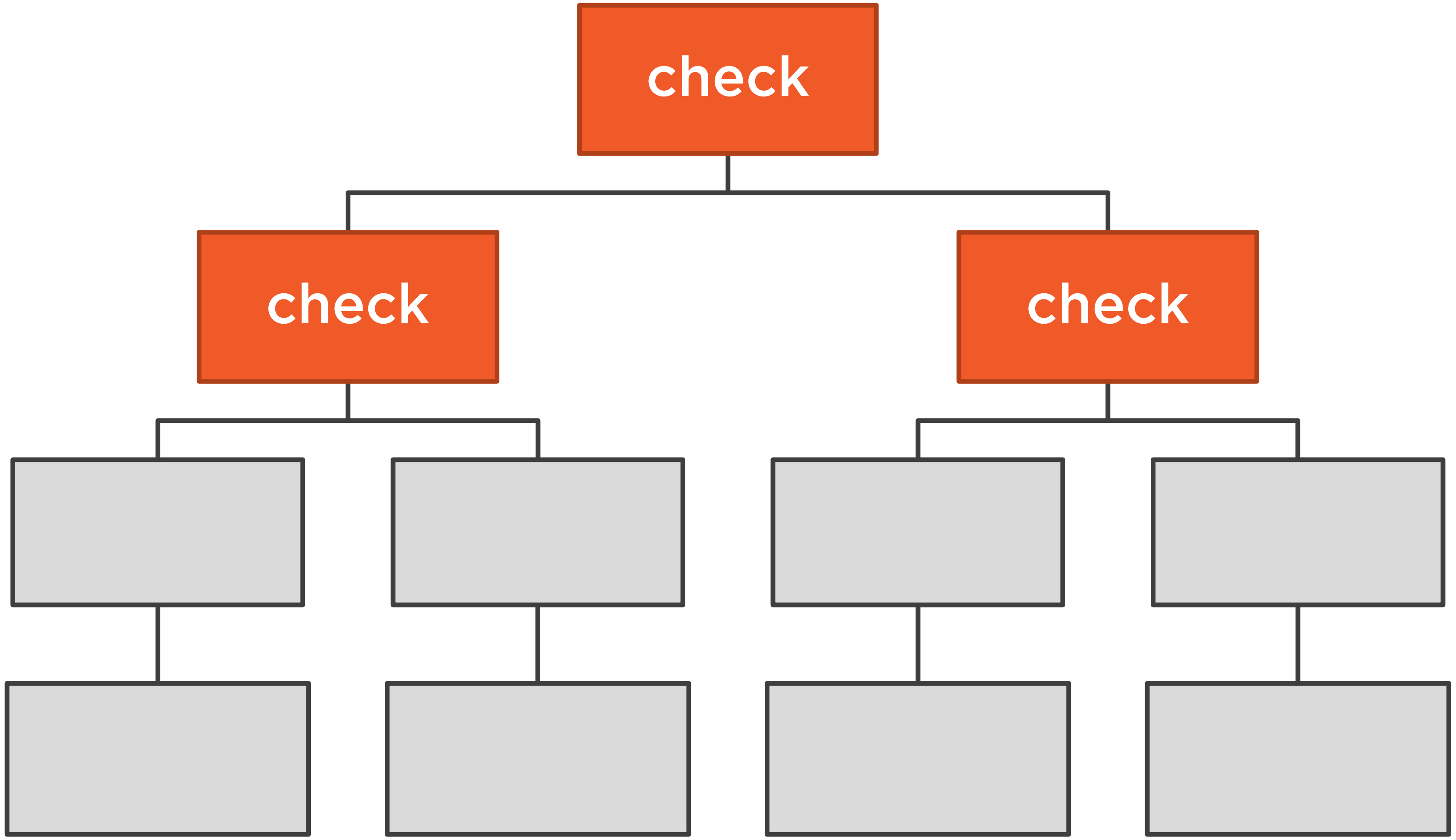
Change Detection



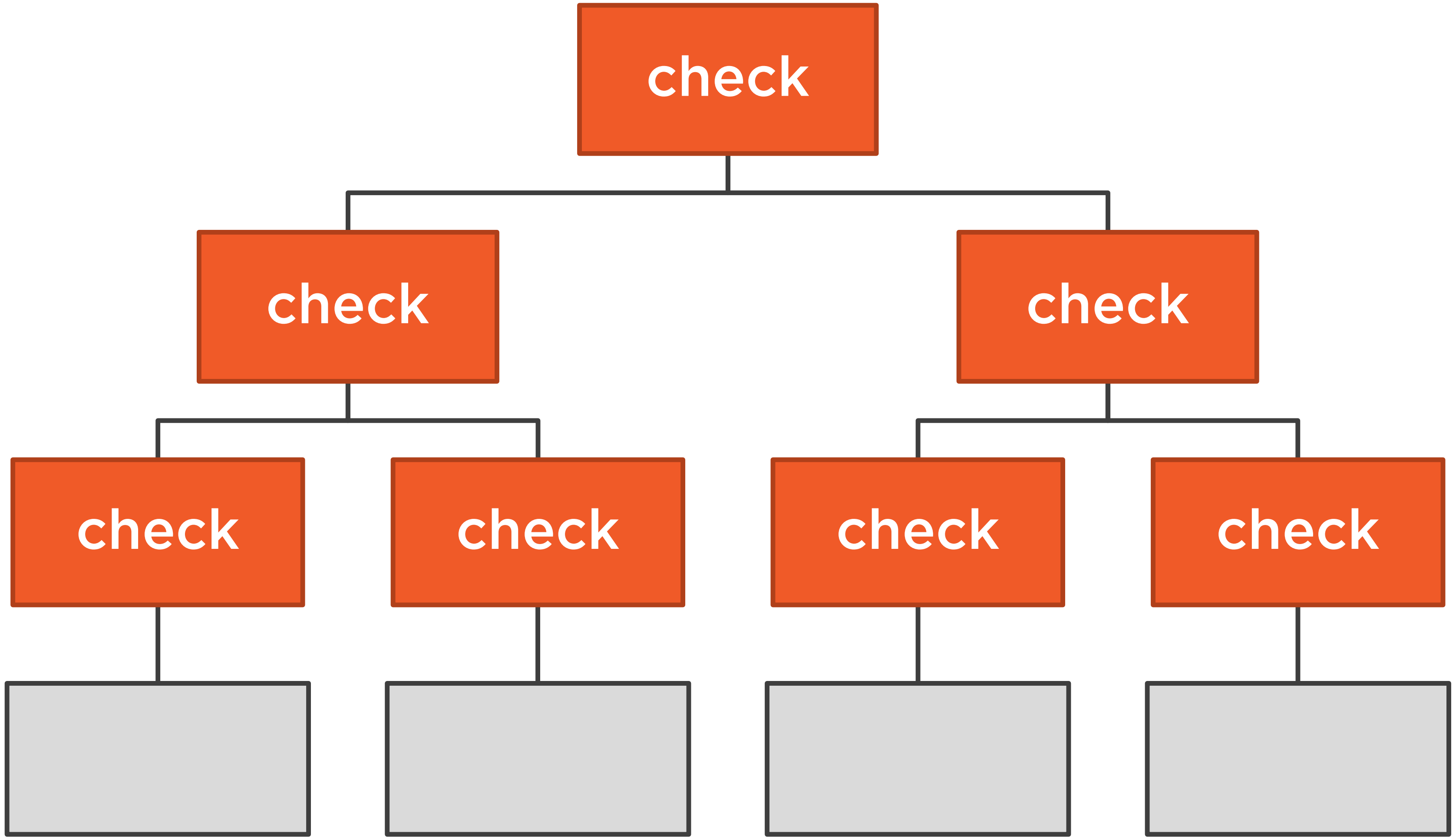
Change Detection



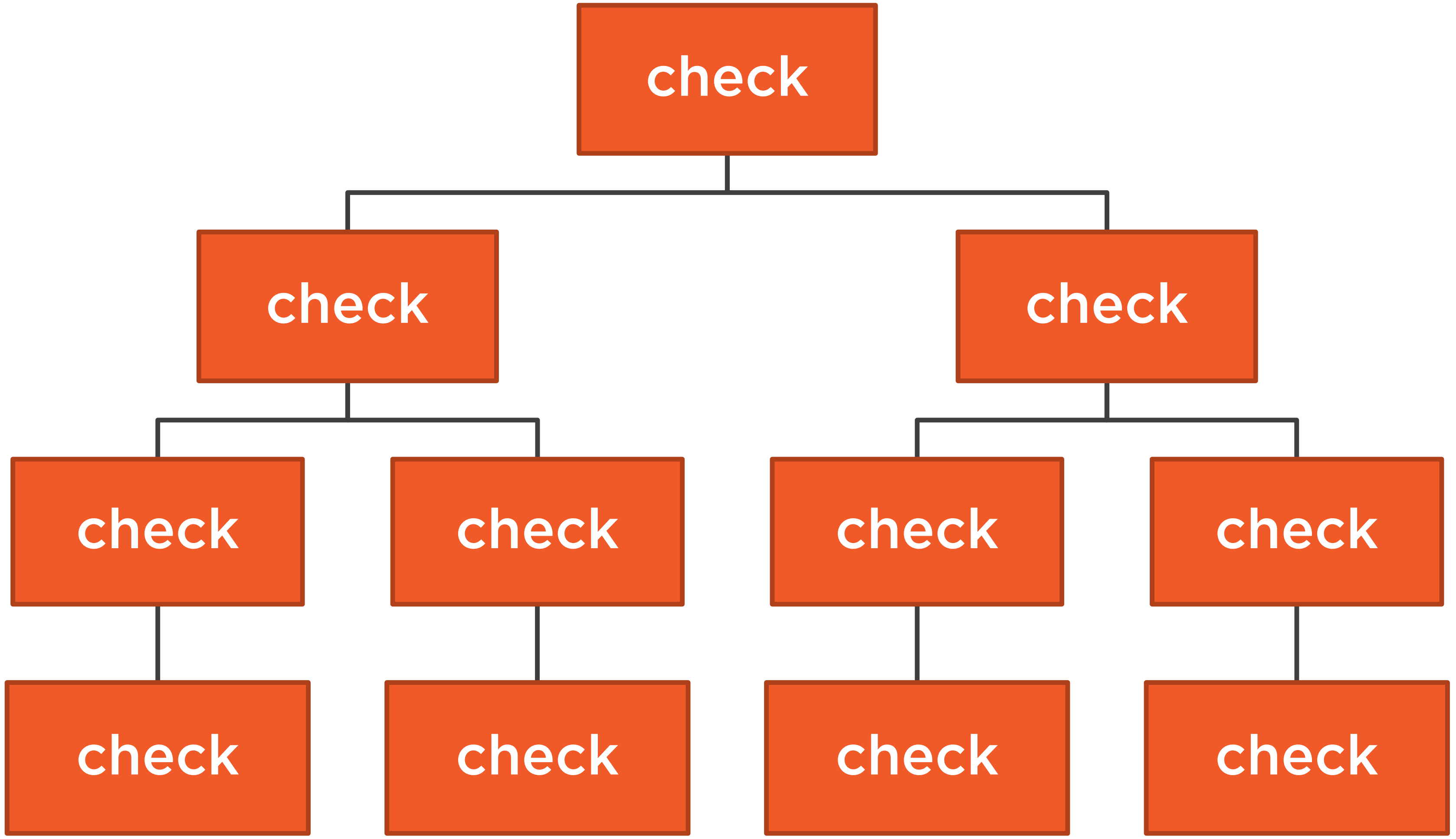
Change Detection



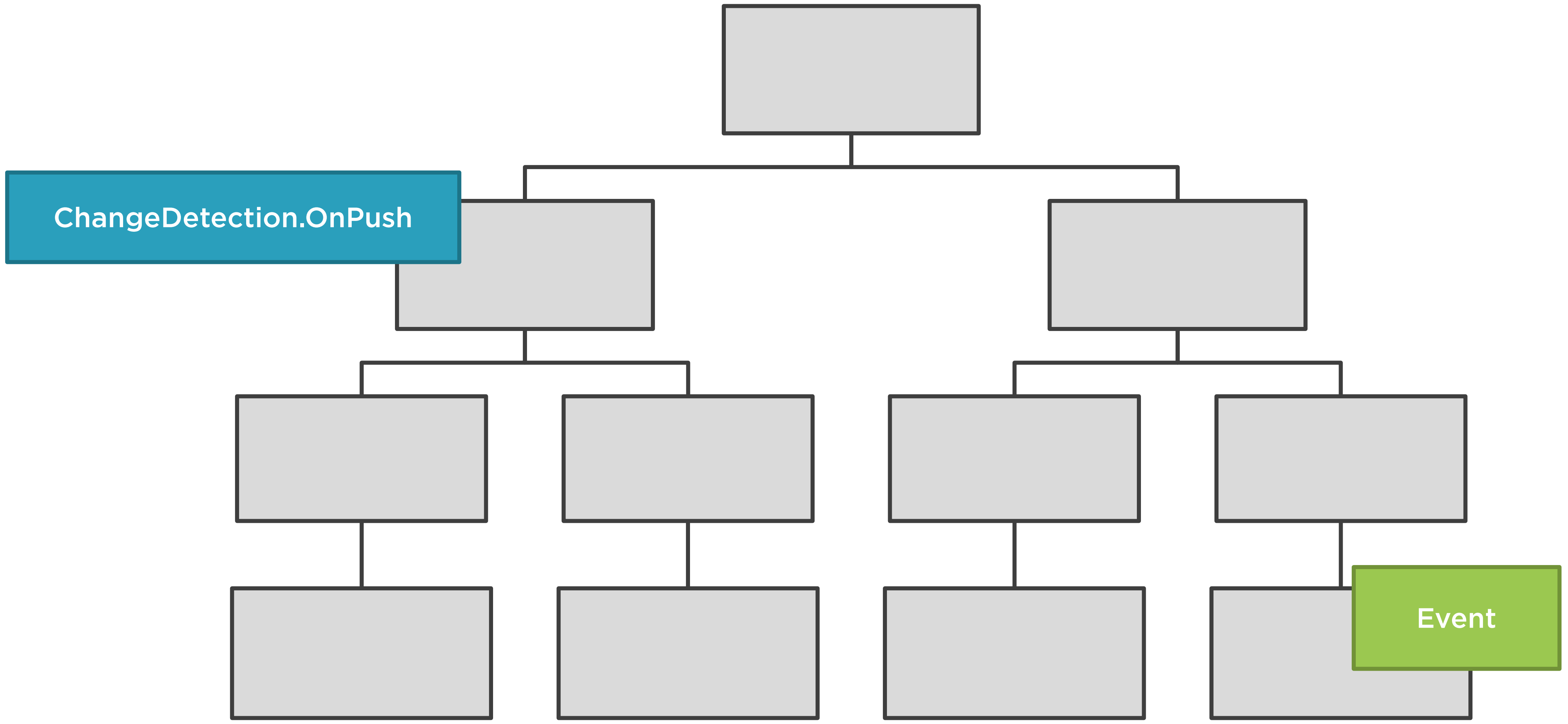
Change Detection



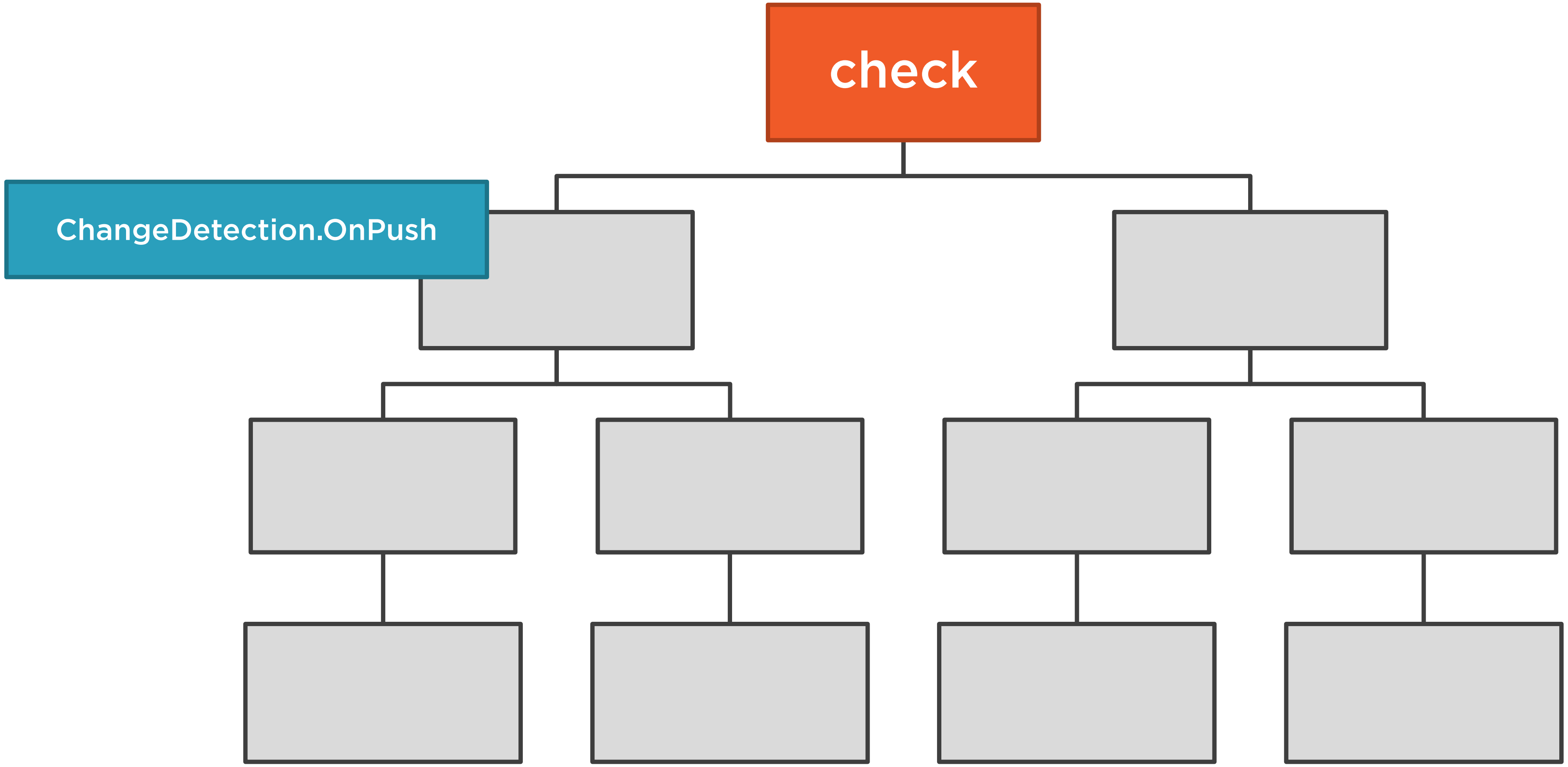
Change Detection



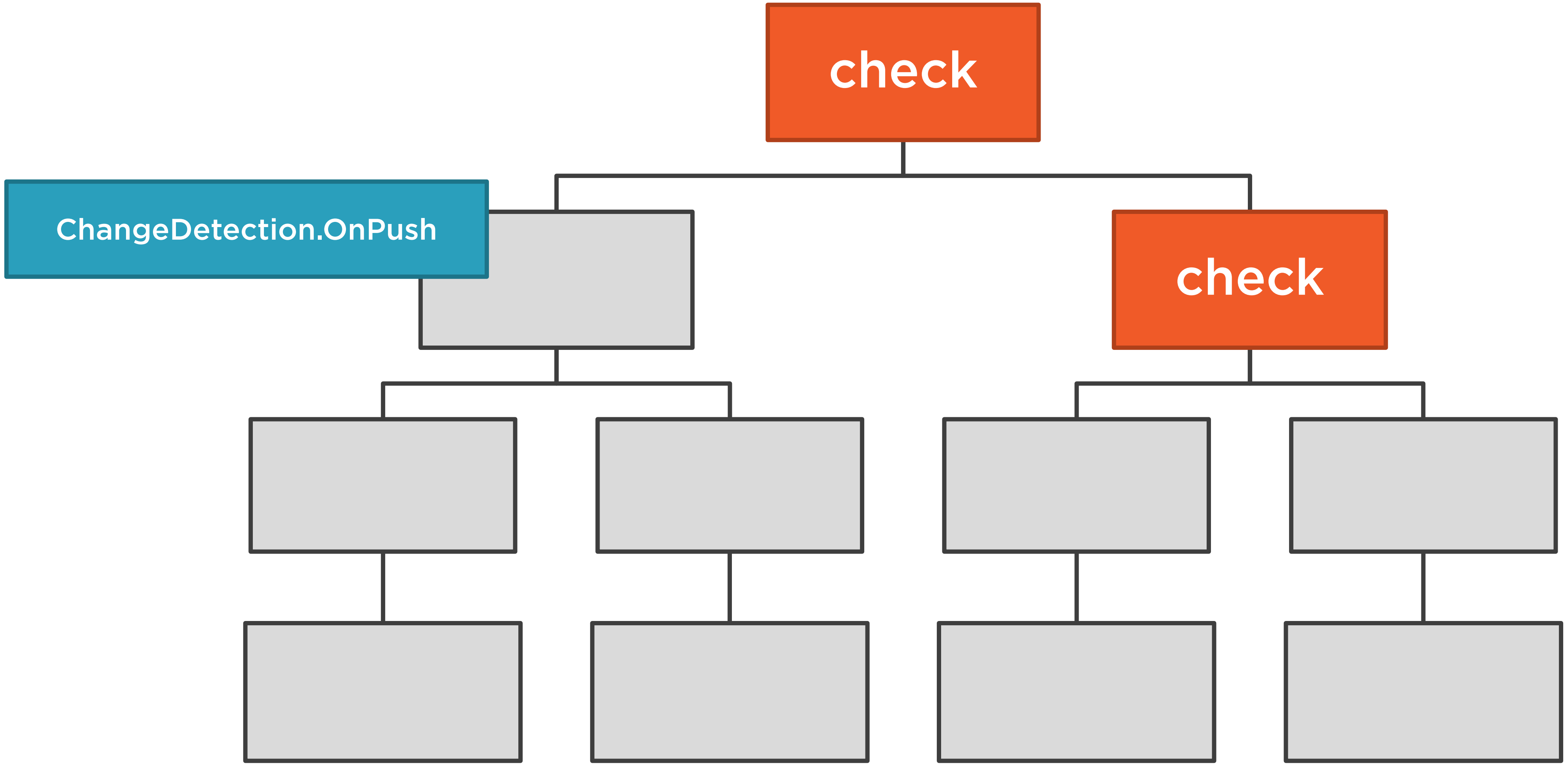
Change Detection - OnPush



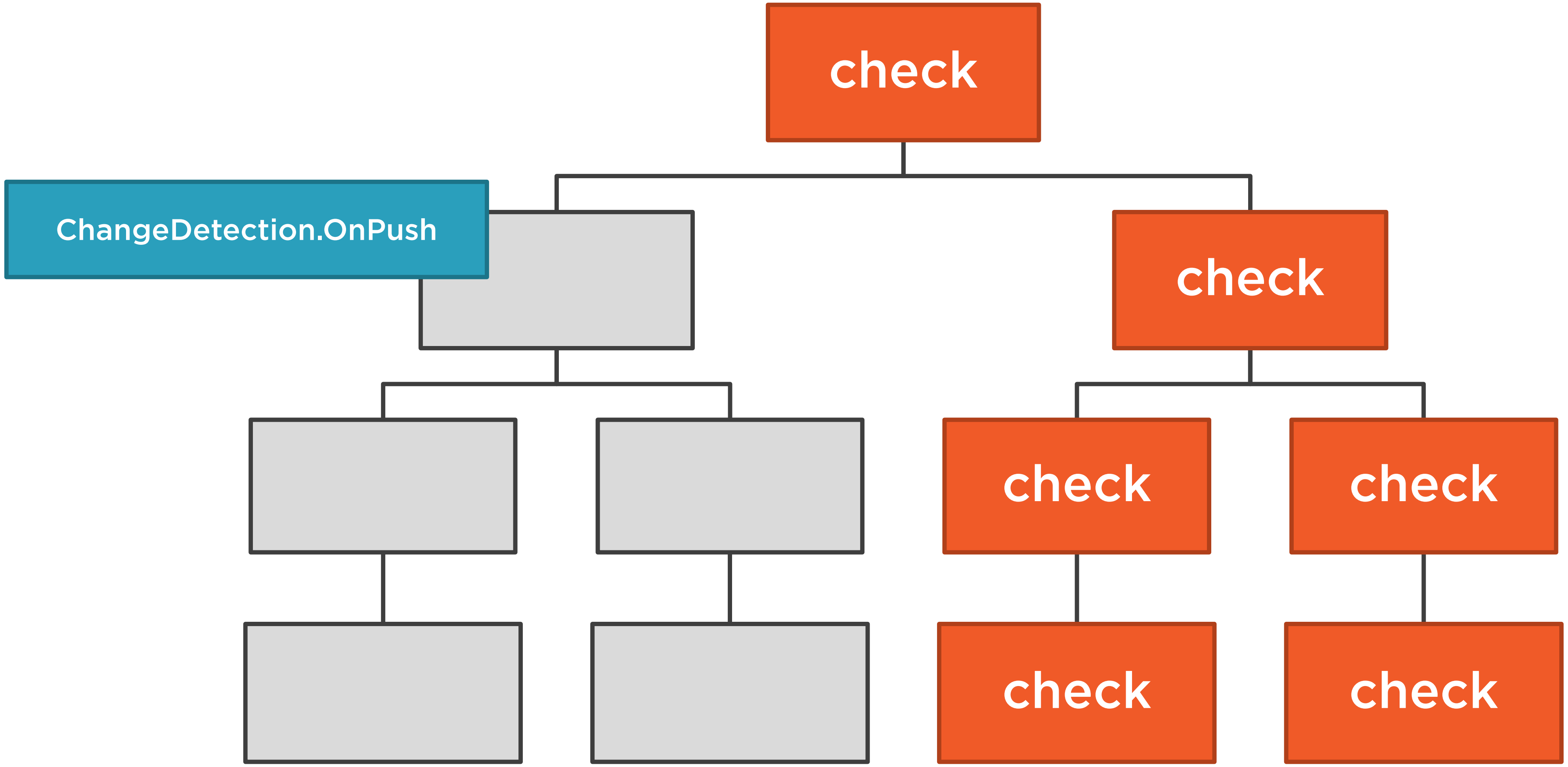
Change Detection - OnPush



Change Detection - OnPush



Change Detection - OnPush



OnPush Change Detection

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


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

OnPush Change Detection

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

OnPush Change Detection Strategy

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

OnPush Change Detection

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


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

Default Change Detection Strategy

Demo

Change detection OnPush

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.

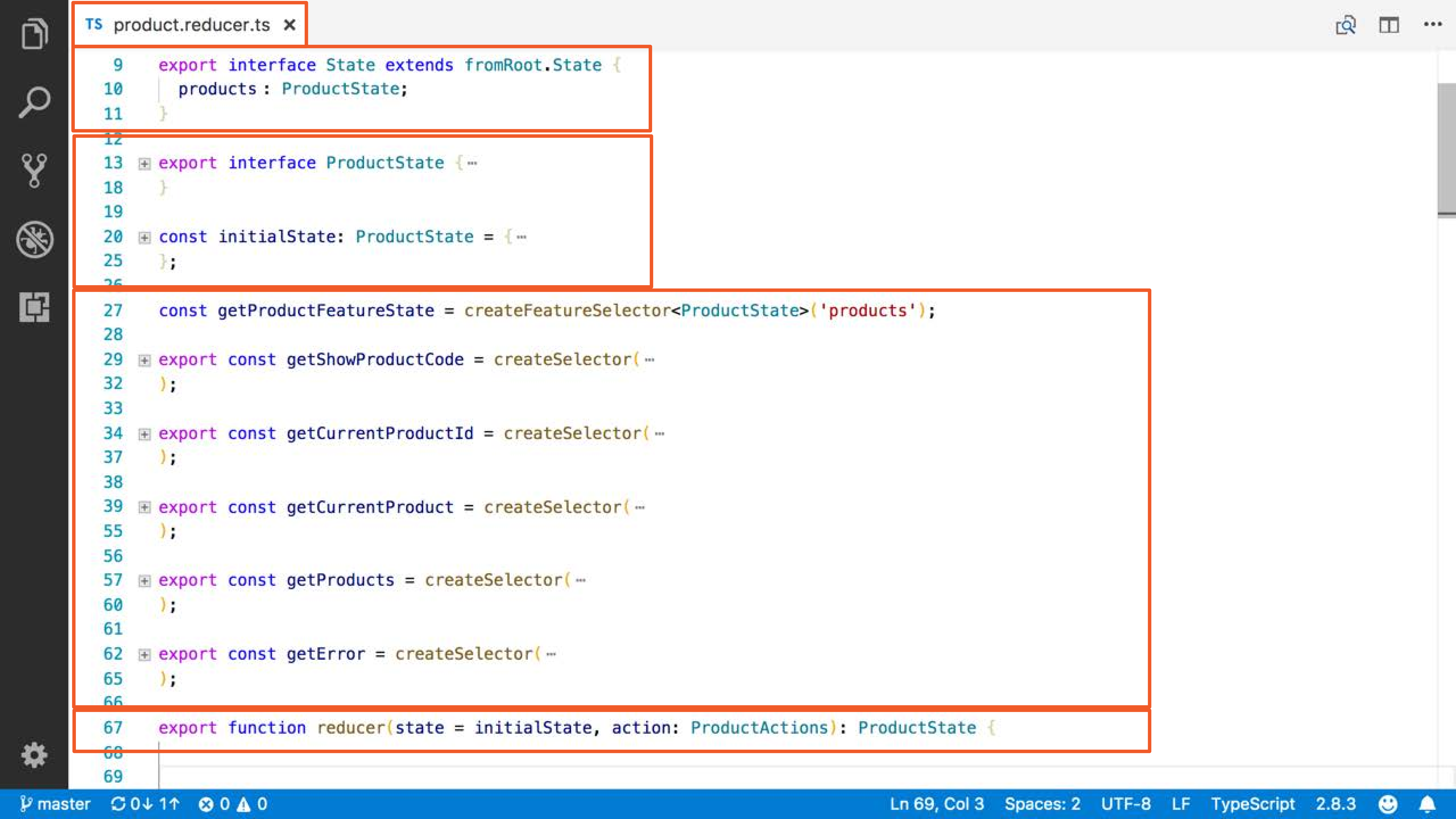
Re-exporting with a Index.ts File

app/index.ts

```
export { Foo } from './app/foo';

export { Bar } from './app/bar';

export * as Baz from './app/baz';
```

Consumer

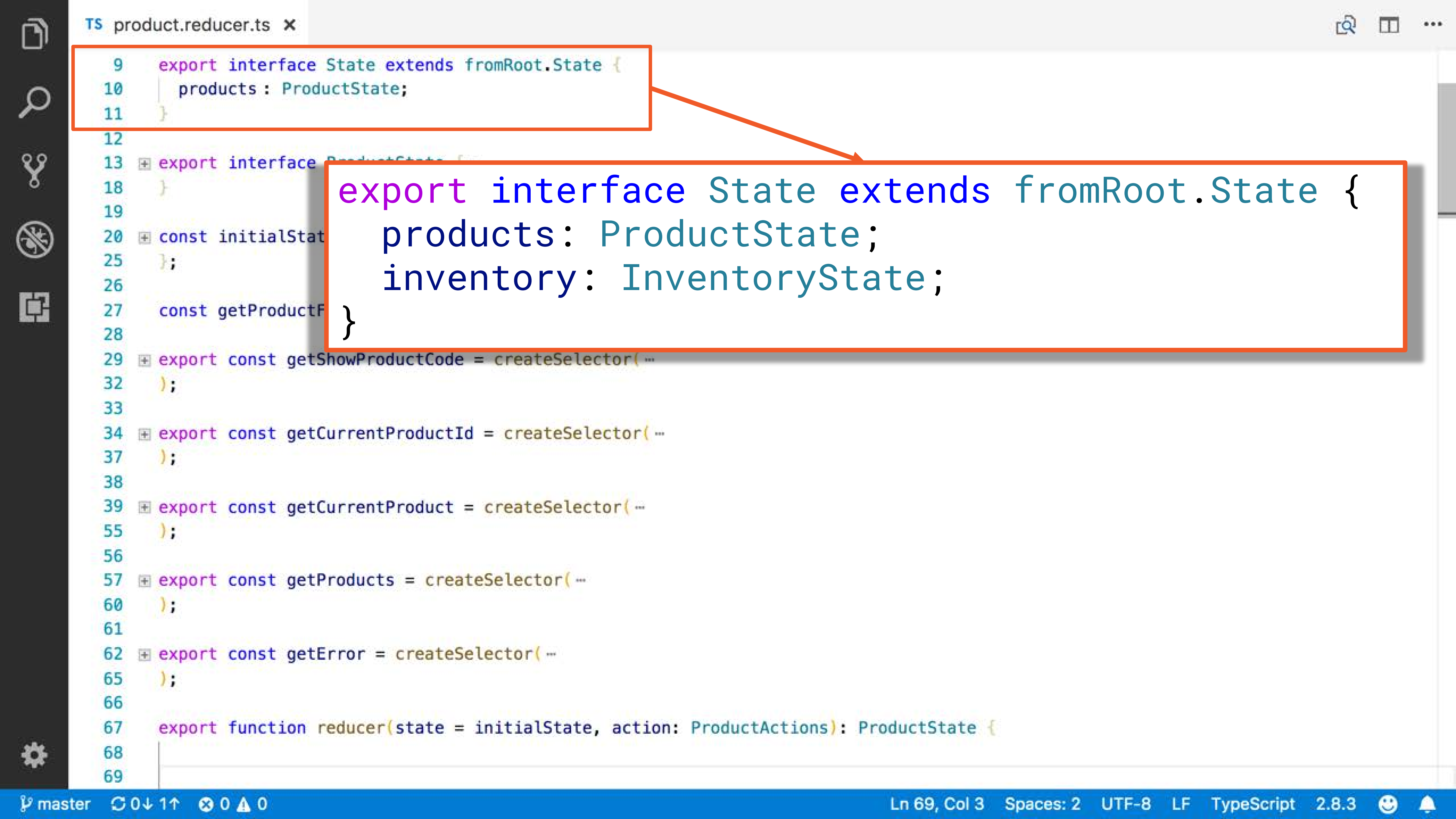
```
import { Foo, Bar, Baz } from './app'; // index.ts implied by convention
```



product.reducer.ts

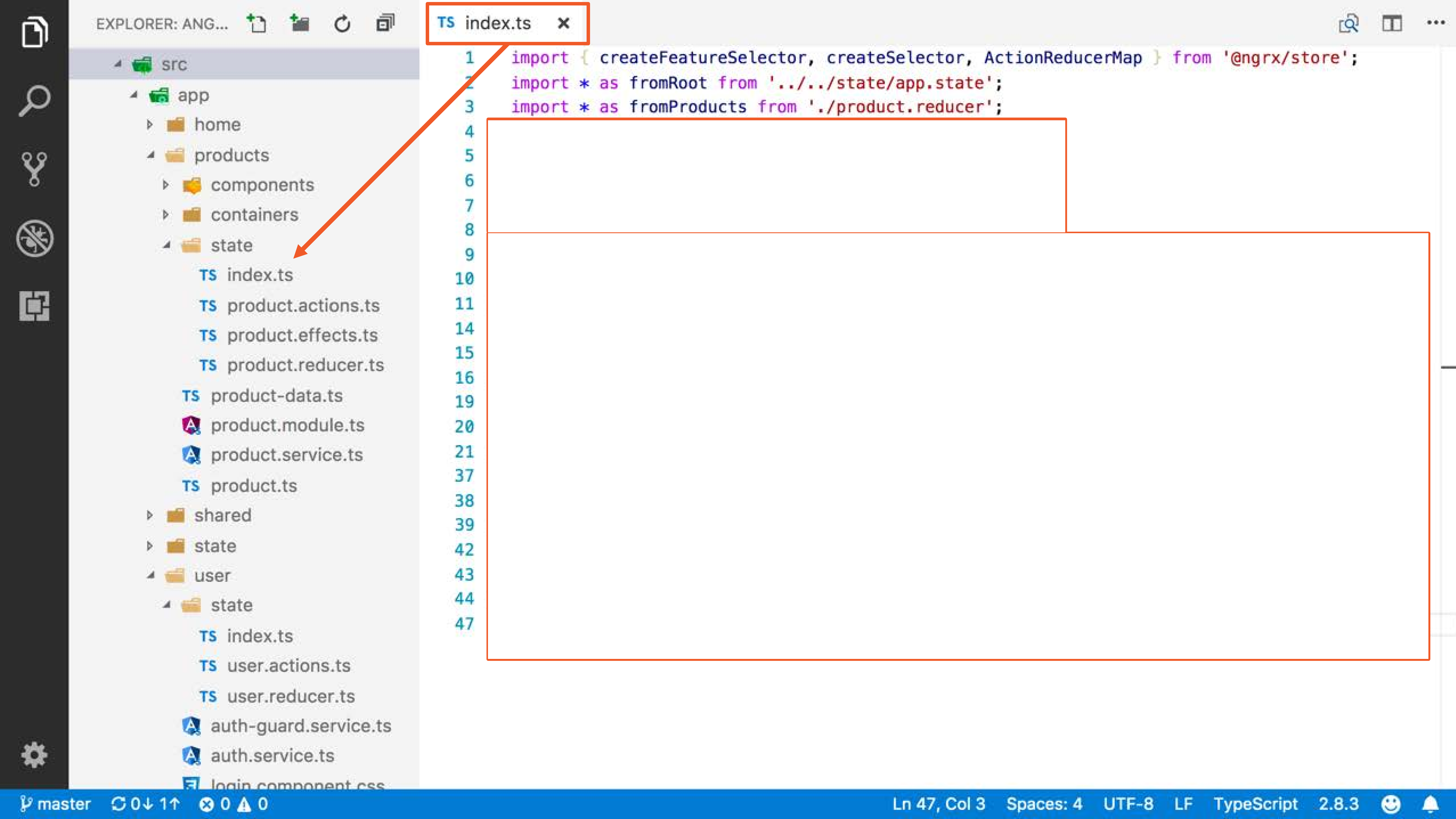
```
export interface State extends fromRoot.State {
  products : ProductState;
}

export interface ProductState {⋯
}

const initialState: ProductState = {⋯
};

const getProductFeatureState = createFeatureSelector<ProductState>('products');

export const getShowProductCode = createSelector(⋯
);

export const getCurrentProductId = createSelector(⋯
);

export const getCurrentProduct = createSelector(⋯
);

export const getProducts = createSelector(⋯
);

export const getError = createSelector(⋯
);

export function reducer(state = initialState, action: ProductActions): ProductState {
```

master   0↓ 1↑   0 ⚠ 0   Ln 69, Col 3   Spaces: 2   UTF-8   LF   TypeScript   2.8.3



```
export interface State extends fromRoot.State {
  products: ProductState;
}

export interface ProductState {
}

const initialStat
};

const getProductF

export const getShowProductCode = createSelector(…
);

export const getCurrentProductId = createSelector(…
);

export const getCurrentProduct = createSelector(…
);

export const getProducts = createSelector(…
);

export const getError = createSelector(…
);

export function reducer(state = initialState, action: ProductActions): ProductState {
```

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



```typescript
import { createFeatureSelector, createSelector, ActionReducerMap } from '@ngrx/store';
import * as fromRoot from '../../state/app.state';
import * as fromProducts from './product.reducer';
```

Benefits of State Index.ts Files

Public API for state

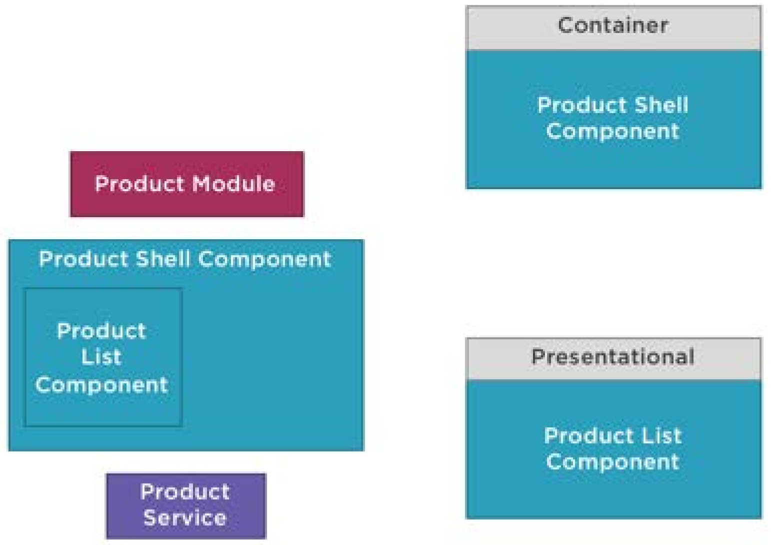
Separation of concerns

Cleaner code

Demo

Adding an index.ts to the state folder

Checklist: Container and Presentational Components



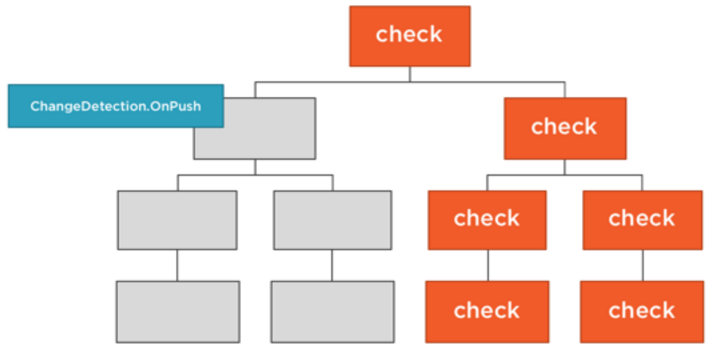
View performance

Separation of concerns

Composability

Easier testing

Checklist: ChangeDetection OnPush

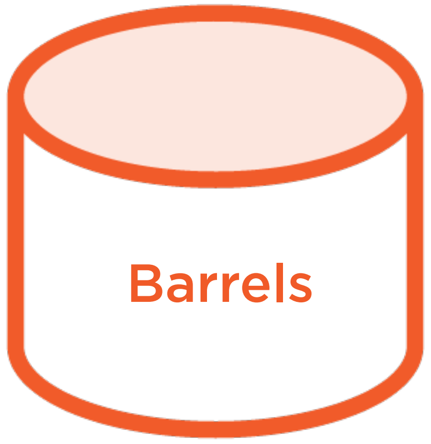


Skip change detection unless an @Input receives a new value or object reference

Add 'ChangeDetectionStrategy.OnPush' to all container component decorators

Easier when categorizing components into presentational or container components

Checklist: Barrels



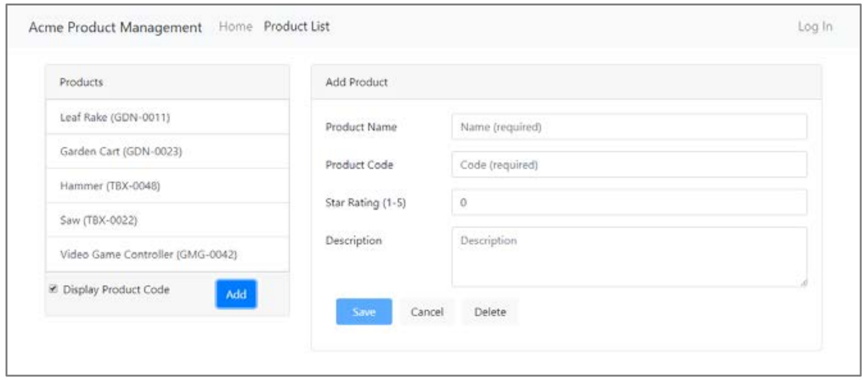
Rollup exports from several ECMAScript modules into a single module

Public APIs for feature state modules

To use barrels:

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

Homework: Presentational Component


Acme Product Management · Home · Product List · Log In

Products
- Leaf Rake (GDN-0011)
- Garden Cart (GDN-0023)
- Hammer (TBX-0048)
- Saw (TBX-0022)
- Video Game Controller (GMG-0042)

☑ Display Product Code · Add

Add Product

Product Name	Name (required)
Product Code	Code (required)
Star Rating (1-5)	0
Description	Description

Save · Cancel · Delete

Move Product Edit component into the components folder

Change the import file paths

Remove the injected store

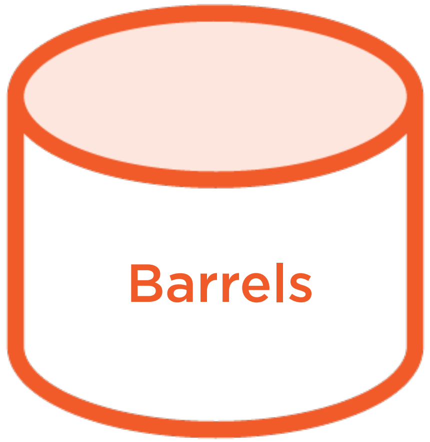
Pass all store state properties in as inputs

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