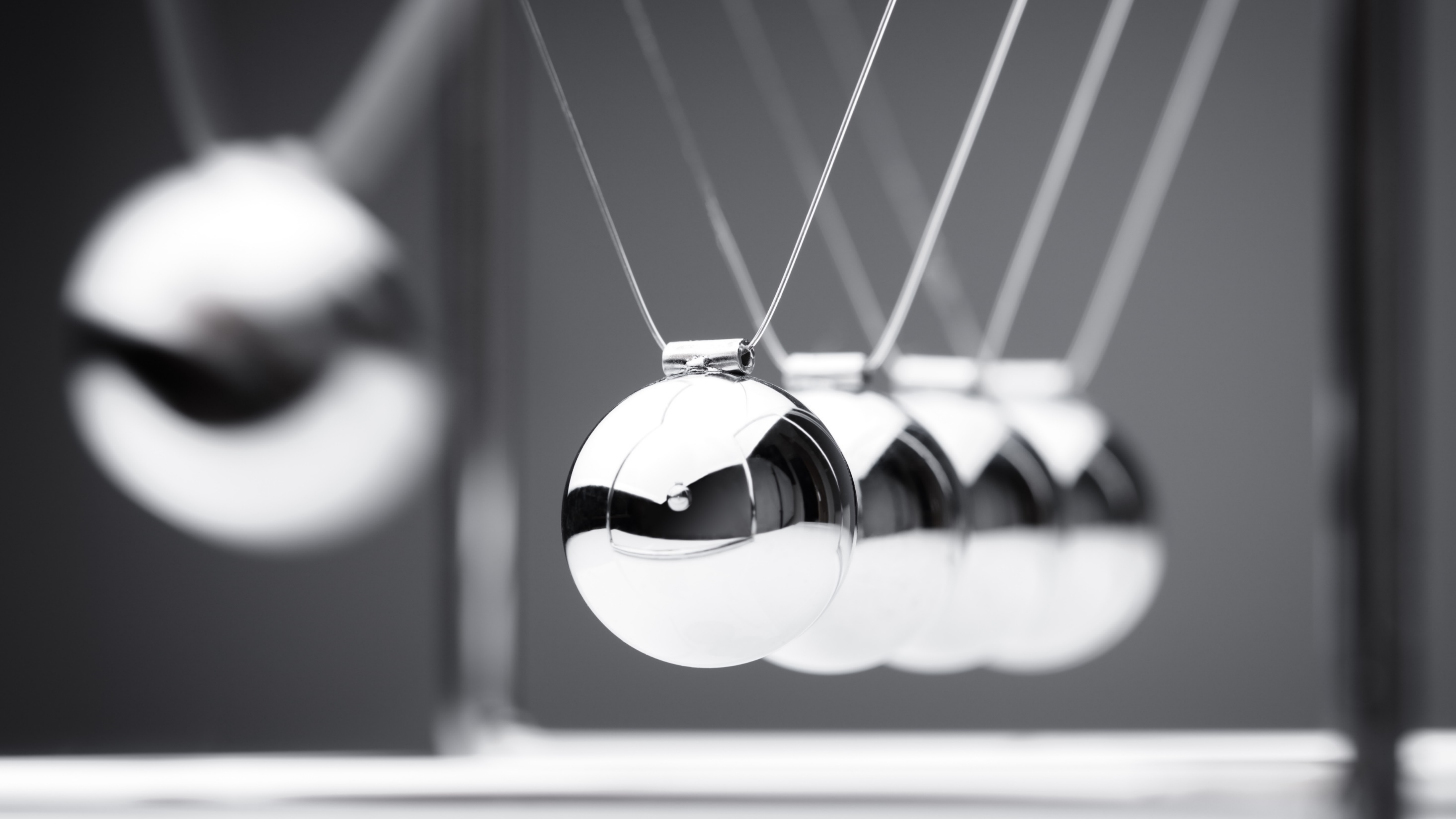
Working with Effects

Duncan Hunter
CONSULTANT | SPEAKER | AUTHOR

@dunchunter duncanhunter.com.au



Module Overview

Why use effects?

Add @ngrx/effects

Define an effect

Register an effect

Use an effect

Exception handling in effects

NgRx Effects Library

Manages side effects to keep components pure

Effects Keep Components Pure

Component

```
constructor(
   private store: Store<State>,
   private productService: ProductService
) { }


ngOnInit() {
   this.productService.getProducts().subscribe(
      products => this.store.dispatch(
        ProductActions.loadProducts()
      )
}
```

Reducers Are Pure Functions

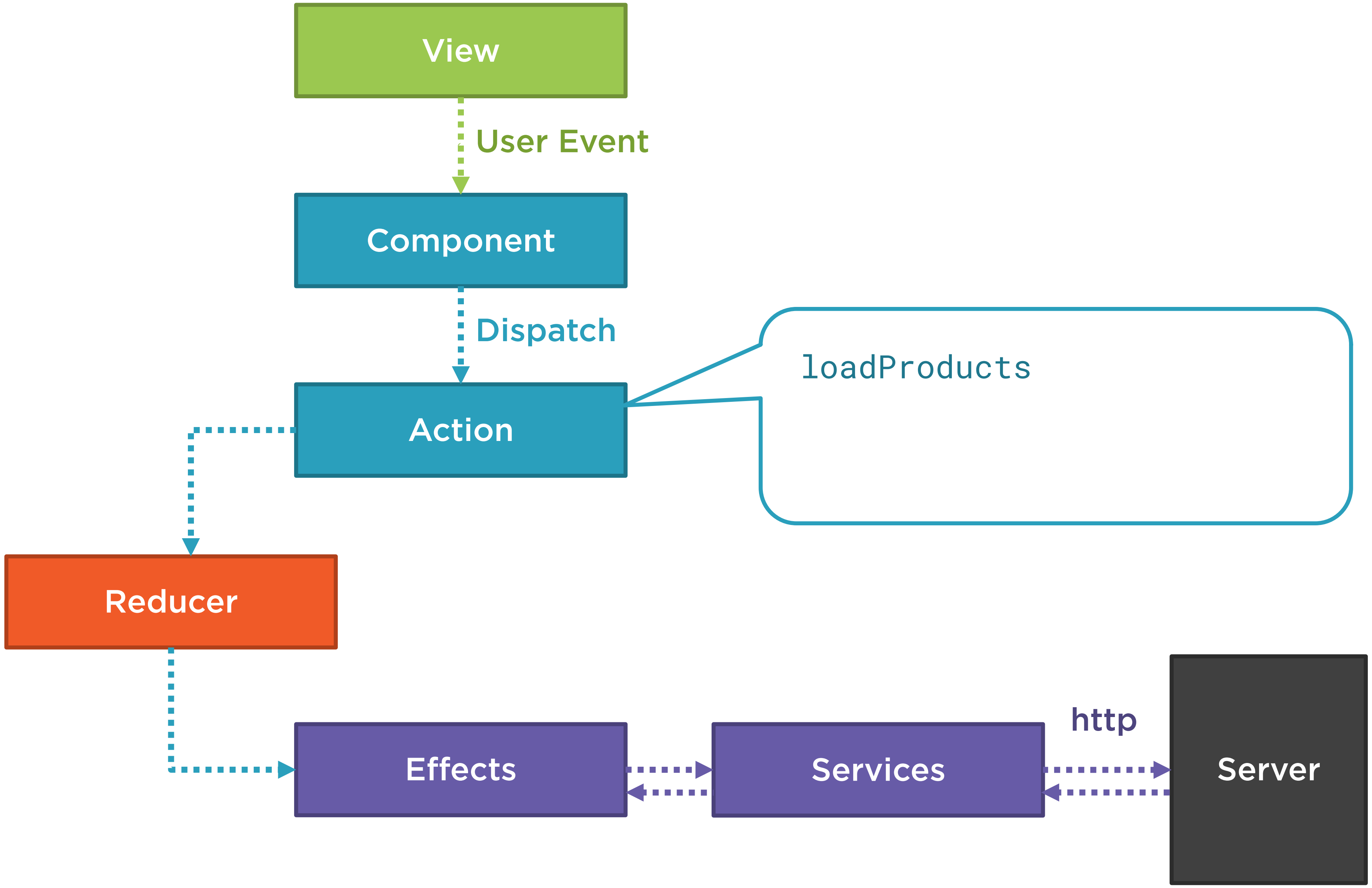
Reducer

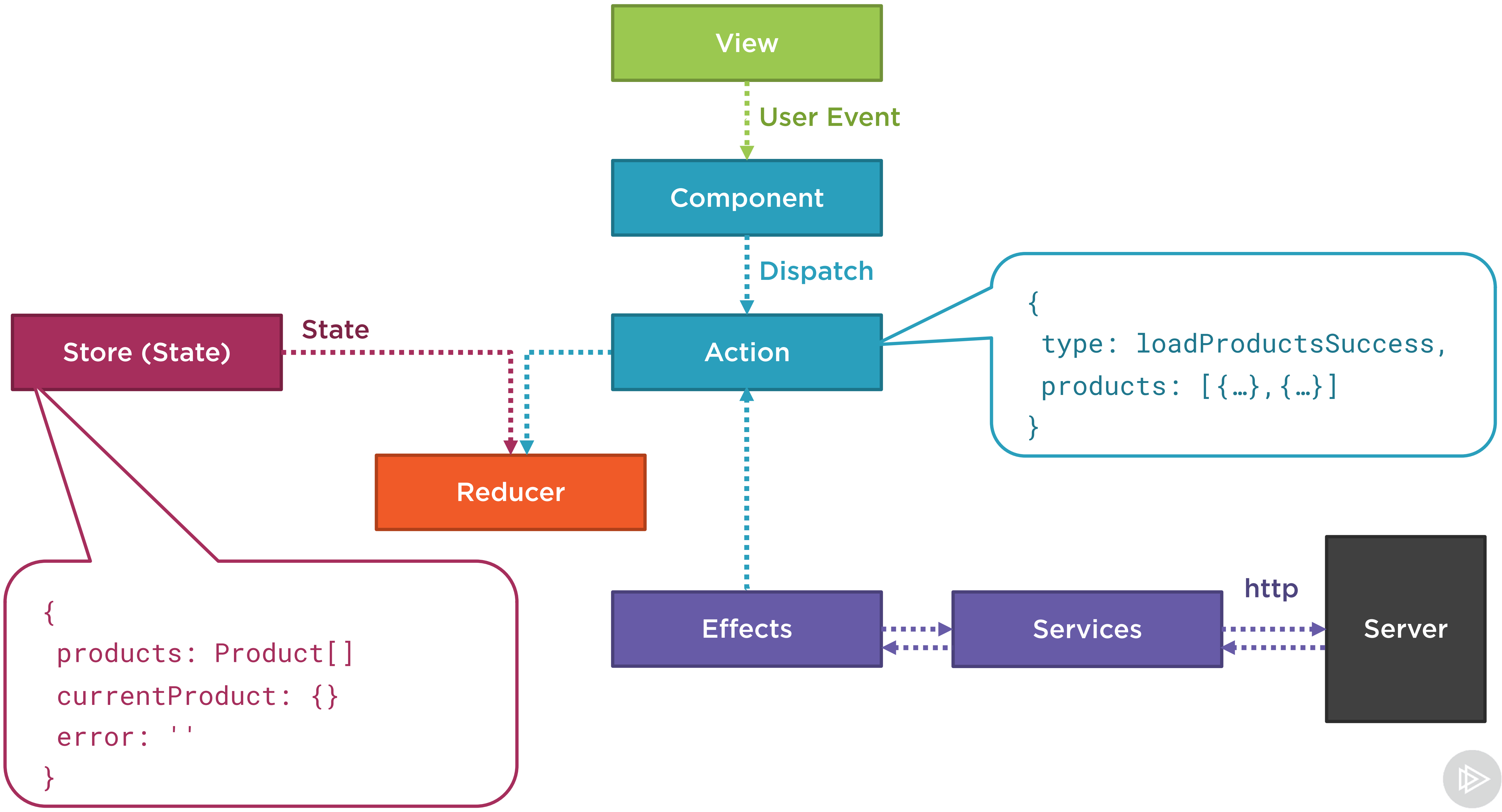
```
export const productReducer = createReducer<ProductState>
(

  initialState,
   return this.productService.getProducts().subscribe(
      products => this.store.dispatch(
        ProductActions.loadProducts()
   )

)
```

Effects Take Actions and Dispatch Actions

Effects

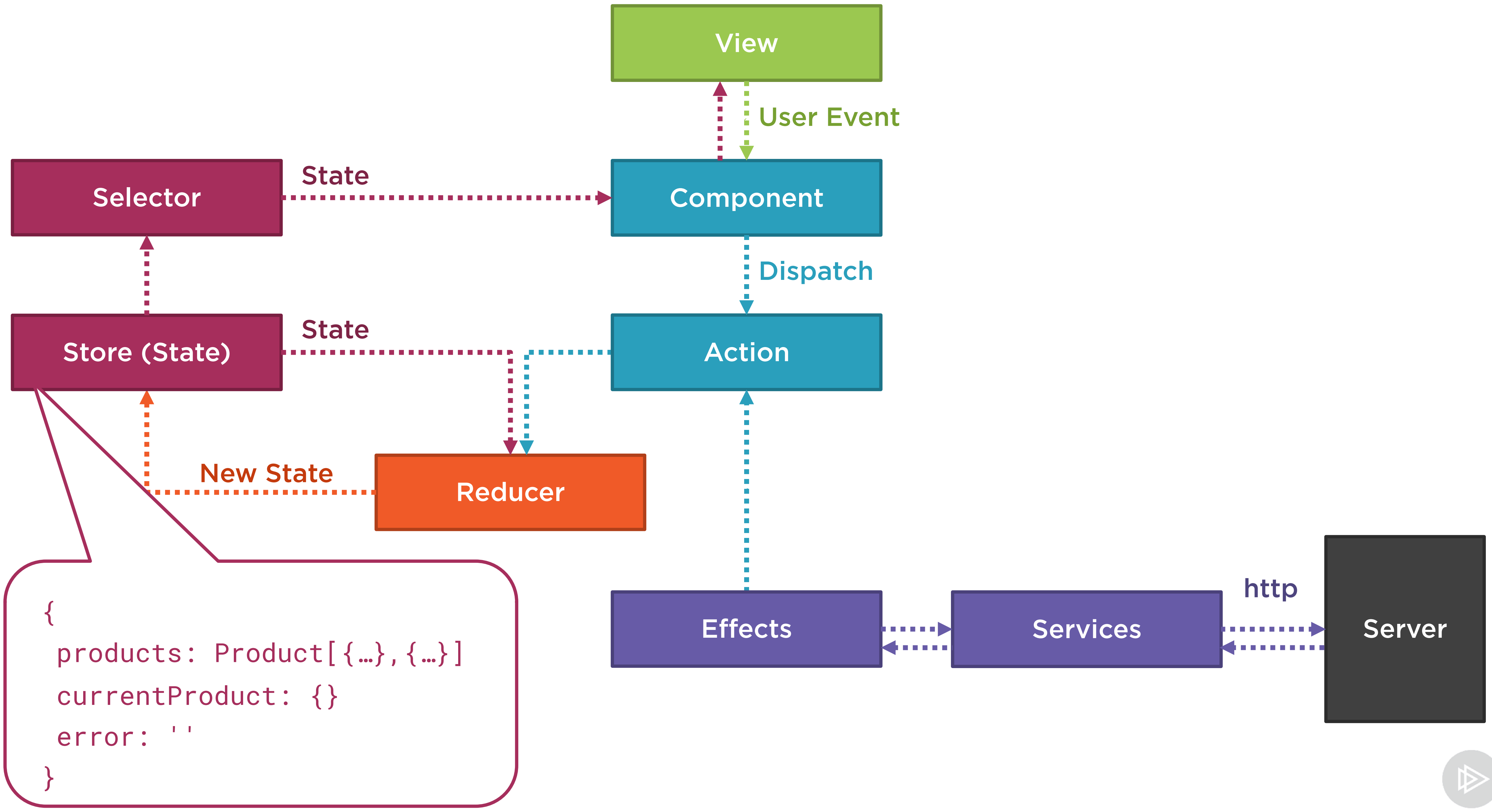
Effects take an action, do some work and dispatch a new action


View
User Event
Component
Dispatch
Action
loadProducts
Reducer
Effects
Services
http
Server



```
{
 type: loadProductsSuccess,
 products: [{…},{…}]
}
```

```
{
 products: Product[]
 currentProduct: {}
 error: ''
}
```



```
{
 products: Product[{…},{…}]
 currentProduct: {}
 error: ''
}
```

Benefits of Effects

Keep components pure

Isolate side effects

Easier to test

Defining an Effect

Create service

```
@Injectable()
export class ProductEffects {









}
```

Defining an Effect

Create service

```
@Injectable()
export class ProductEffects {

    constructor(private actions$:Actions,
                                                    ) { }
}
```

Defining an Effect

Create service

```
@Injectable()
export class ProductEffects {

    constructor(private actions$:Actions,
                private productService: ProductService) { }






}
```

Defining an Effect

```
@Injectable()
export class ProductEffects {

   constructor(private actions$:Actions,
               private productService: ProductService) { }

   loadProducts$ = createEffect(() => {









   });
}
```

Define effect

Defining an Effect

Define effect

```
@Injectable()
export class ProductEffects {

    constructor(private actions$:Actions,
                private productService: ProductService) { }

    loadProducts$ = createEffect(() => {
      return this.actions$





    });
}
```

Defining an Effect

```
@Injectable()
export class ProductEffects {

   constructor(private actions$:Actions,
               private productService: ProductService) { }

   loadProducts$ = createEffect(() => {
     return this.actions$.pipe(
       ofType(ProductActions.loadProducts),




     );
   });
}
```

Filter actions

Defining an Effect

```
@Injectable()
export class ProductEffects {

   constructor(private actions$:Actions,
               private productService: ProductService) { }

   loadProducts$ = createEffect(() => {
     return this.actions$.pipe(
       ofType(ProductActions.loadProducts),
        mergeMap(action =>


        )
     );
   });
}
```

Map

Defining an Effect

```
@Injectable()
export class ProductEffects {

   constructor(private actions$:Actions,
                         private productService: ProductService) { }

   loadProducts$ = createEffect(() => {
     return this.actions$.pipe(
       ofType(ProductActions.loadProducts),
        mergeMap(action =>
         this.productService.getProducts().pipe(
         )
        )
     );
   });
}
```

Call service

Defining an Effect

```
@Injectable()
export class ProductEffects {

   constructor(private actions$:Actions,
            private productService: ProductService) { }

   loadProducts$ = createEffect(() => {
     return this.actions$.pipe(
       ofType(ProductActions.loadProducts),
       mergeMap(action =>
        this.productService.getProducts().pipe(
          map(products =>
           ProductActions.loadProductsSuccess({products}))
        )
      );
   });
}
```

Return action

Defining an Effect

Create service

Define effect

Filter actions

Map

Call service

Return new action

```
@Injectable()
export class ProductEffects {

   constructor(private actions$:Actions,
               private productService: ProductService) { }

   loadProducts$ = createEffect(() => {
     return this.actions$.pipe(
       ofType(ProductActions.loadProducts),
       mergeMap(action =>
        this.productService.getProducts().pipe(
          map(products =>
           ProductActions.loadProductsSuccess({products}))
        )
      );
   });
}
```

Defining an Effect

```
@Injectable()
export class ProductEffects {

   constructor(private actions$:Actions,
               private productService: ProductService) { }

   loadProducts$ = createEffect(() => {
     return this.actions$.pipe(
       ofType(ProductActions.loadProducts),
       mergeMap(action =>
        this.productService.getProducts().pipe(
          map(products =>
           ProductActions.loadProductsSuccess({products}))
        )
      );
   })
}
```

Take an action

Do some work

Return a new action

Demo

Install and define an effect

RxJS Operators

```
loadProducts$ = createEffect(() => {
  return this.actions$.pipe(
    ofType(ProductActions.loadProducts),
    mergeMap(action =>
     this.productService.getProducts().pipe(
       map(products =>
        ProductActions.loadProductsSuccess({products}))
     )
   );
});
```

RxJS Operators

```
loadProducts$ = createEffect(() => {
  return this.actions$.pipe(
    ofType(ProductActions.loadProducts),
    mergeMap(action =>
     this.productService.getProducts().pipe(
       map(products =>
        ProductActions.loadProductsSuccess({products}))
     )
   );
});
```

RxJS Operators

```
loadProducts$ = createEffect(() => {
  return this.actions$.pipe(
    ofType(ProductActions.loadProducts),
    mergeMap(action =>
     this.productService.getProducts().pipe(
       map(products =>
        ProductActions.loadProductsSuccess({products}))
     )
    )
  );
});
```

RxJS Operators

```
loadProducts$ = createEffect(() => {
  return this.actions$.pipe(
    ofType(ProductActions.loadProducts),
    switchMap(action =>
     this.productService.getProducts().pipe(
       map(products =>
        ProductActions.loadProductsSuccess({products}))
     )
   );
});
```

RxJS Operators

switchMap	Cancels the current subscription/request and can cause race condition **Use for get requests or cancelable requests like searches**
concatMap	Runs subscriptions/requests in order and is less performant **Use for get, post and put requests when order is important**
mergeMap	Runs subscriptions/requests in parallel **Use for get, put, post and delete methods when order is not important**
exhaustMap	Ignores all subsequent subscriptions/requests until it completes **Use for login when you do not want more requests until the initial one is complete**

Registering an Effect

App Module

```
@NgModule({
 imports:[
  ...
  StoreModule.forRoot({}),
  EffectsModule.forRoot([]),
 ],
 declarations:[...],
 bootstrap:[...]
})
export class AppModule{ }
```

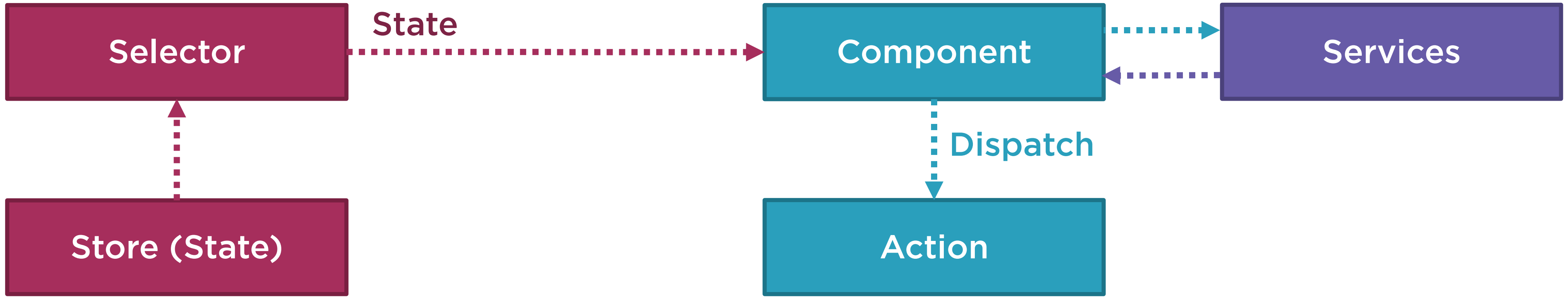
Product Module

```
@NgModule({
 imports:[
    ...
  StoreModule.forFeature('products',reducer),
  EffectsModule.forFeature([ProductEffects])
 ],
 declarations:[...],
 providers:[...]
})
export class ProductModule{ }
```

Demo

Registering an effect

Using Effects



Using Effects

Inject the store

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

Call the dispatch method

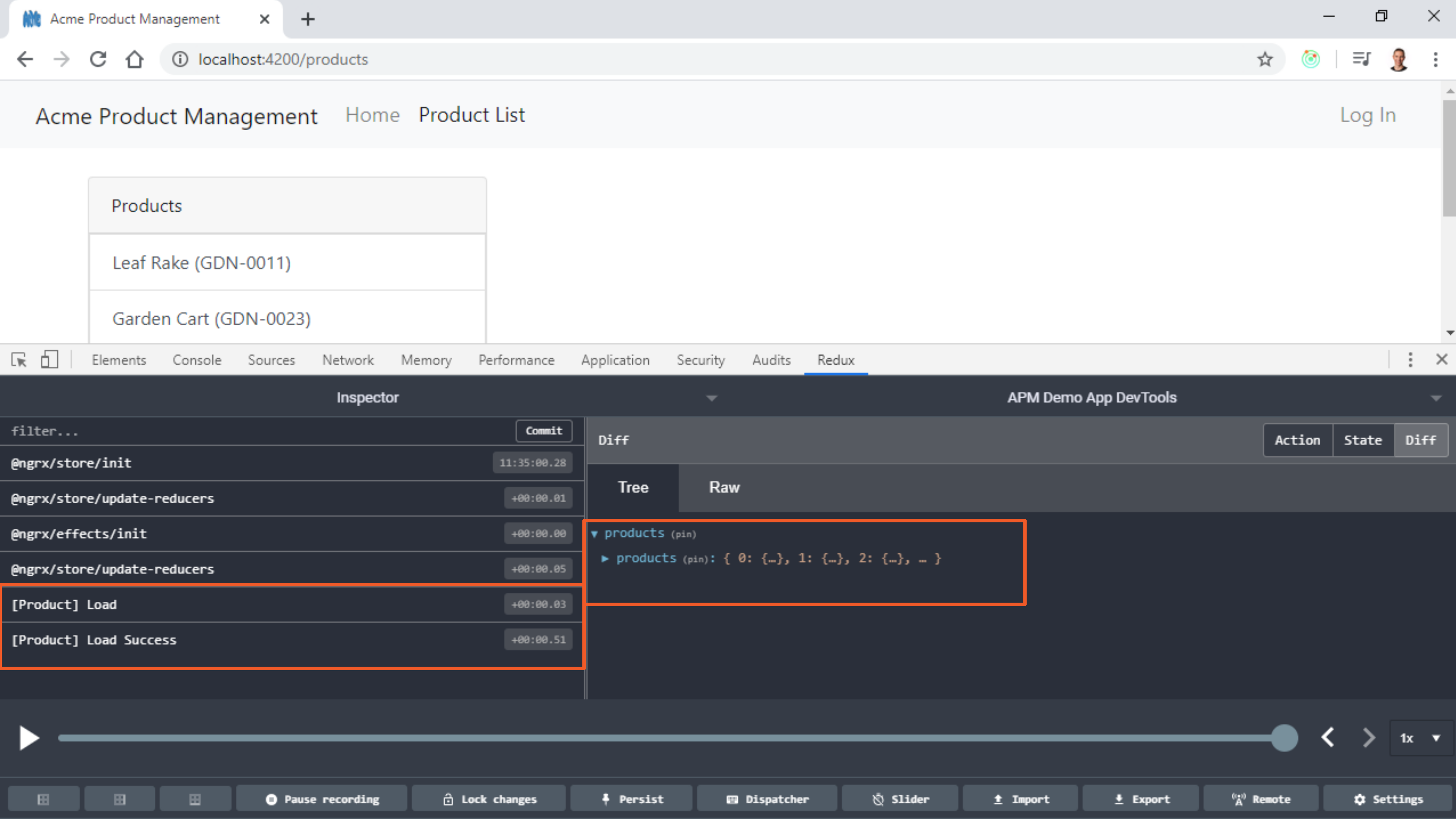
```
this.store.dispatch(ProductActions.loadProducts());
```

Select state with selector

```
this.products$ = this.store.select(getProducts);
```

Add async pipe

```
*ngIf="products$ | async as products"
```


Acme Product Management

localhost:4200/products

Acme Product Management Home Product List Log In

Products

Leaf Rake (GDN-0011)

Garden Cart (GDN-0023)

Elements Console Sources Network Memory Performance Application Security Audits Redux

Inspector

APM Demo App DevTools

```
@ngrx/store/init                 11:35:00.28
@ngrx/store/update-reducers      +00:00.01
@ngrx/effects/init               +00:00.00
@ngrx/store/update-reducers      +00:00.05
[Product] Load                   +00:00.03
[Product] Load Success           +00:00.51
```

Diff

Action State Diff

Tree Raw

```
▼ products (pin)
  ▶ products (pin): { 0: {…}, 1: {…}, 2: {…}, … }
```

Demo

Using the effect

Exception Handling in Effects

```
@Injectable()
export class ProductEffects {

    constructor(private actions$:Actions,
                private productService: ProductService) { }

    loadProducts$ = createEffect(() => {
      return this.actions$.pipe(
        ofType(ProductActions.loadProducts),
        mergeMap(action =>
          this.productService.getProducts().pipe(
            map(products =>
              ProductActions.loadProductsSuccess({products}))
          )
        );
    });
}
```

Exception Handling in Effects

```
@Injectable()
export class ProductEffects {

   constructor(private actions$:Actions,
               private productService: ProductService) { }

   loadProducts$ = createEffect(() => {
     return this.actions$.pipe(
       ofType(ProductActions.loadProducts),
       mergeMap(action =>
        this.productService.getProducts().pipe(
          map(products =>
           ProductActions.loadProductsSuccess({products}))
        )
      );
   });
}
```

Return new action

Exception Handling in Effects

```
@Injectable()
export class ProductEffects {

    constructor(private actions$:Actions,
                private productService: ProductService) { }

    loadProducts$ = createEffect(() => {
      return this.actions$.pipe(
        ofType(ProductActions.loadProducts),
        mergeMap(action =>
          this.productService.getProducts().pipe(
            map(products =>
              ProductActions.loadProductsSuccess({products})),
            catchError(error =>
              of(ProductActions.loadProductsFailure({error})))
      );
    });
}
```

Return new action

Exception Handling in Effects

```
@Injectable()
export class ProductEffects {

   constructor(private actions$:Actions,
                  private productService: ProductService) { }

   loadProducts$ = createEffect(() => {
     return this.actions$.pipe(
       ofType(ProductActions.loadProducts),
       mergeMap(action =>
        this.productService.getProducts().pipe(
          map(products =>
           ProductActions.loadProductsSuccess({products})),
          catchError(error =>
            of(ProductActions.loadProductsFailure({error}))))
       );
   });
}
```

Return
new action

Exception Handling in Effects

```
@Injectable()
export class ProductEffects {

   constructor(private actions$:Actions,
               private productService: ProductService) { }

   loadProducts$ = createEffect(() => {
     return this.actions$.pipe(
       ofType(ProductActions.loadProducts),
       mergeMap(action =>
        this.productService.getProducts().pipe(
          map(products =>
           ProductActions.loadProductsSuccess({products})),
          catchError(error =>
            of(ProductActions.loadProductsFailure({error})))
      );
   });
}
```

Return
new action

Exception Handling in Effects

Add to interface

```
export interface ProductState {
 ...
 error: string;
}
```

Initialize state

```
const initialState: ProductState = {
 ...
 error:''
};
```

Make selector

```
export const getError = createSelector(
 getProductFeatureState,
 state => state.error
);
```

Exception Handling in Effects

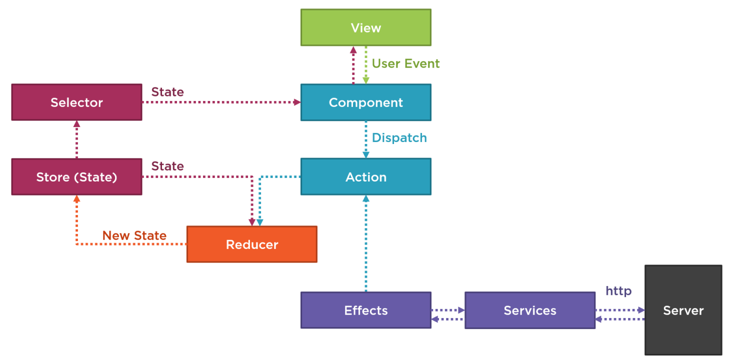
Add on handler

```
on(ProductActions.loadProductsFailure,
  (state, action):ProductState => {
    return {
      ...state,
      products: [],
      error: action.error
     };
}),
```

Demo

Add exception handling to effect

Checklist: Using Effects



Add @ngrx/effects

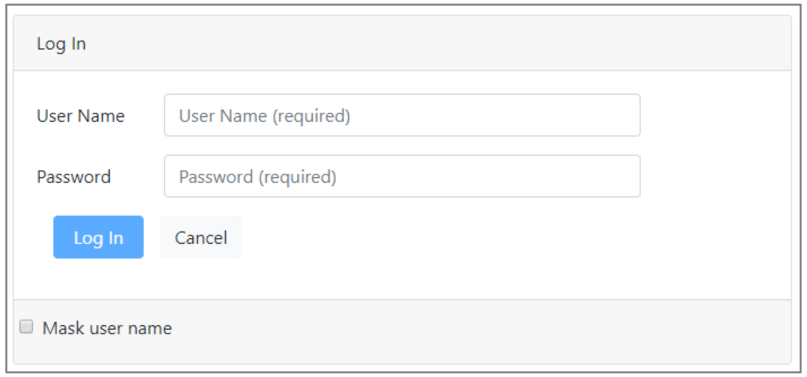
Build the effect to process that action and dispatch the success and fail actions

Initialize the effects module in your root module

Register effects in your feature modules

Process the success and failure actions in the reducer

Homework



Update the login component to use an async pipe

Add a maskUserName$ variable in the component

Subscribe in the template with an async pipe

https://github.com/DeborahK/Angular-NgRx-GettingStarted/tree/master/APM-Demo3