Strongly Typing Actions with Action Creators

Duncan Hunter
Lead Frontend Developer

@dunchunter | duncanhunter.com.au

Module Introduction

How to write good actions

Using action creators

Update the demo to load products data

Actions express unique events that happen throughout your application

Action Interface

```
{
    type: string;
}
```

Example Action

Event source and catego

Action to happen

```
{
    type: '[Products Page] Toggle Show Product Code'
}
```

Example Action

Event source and catego

Action to happen

```
{
      type: '[Products API] Load Products Success',
      products: [{ id: 1, name : 'Hammer' }]
}
```

Example Action with Data

```
{
    type: '[Products API] Load Products Success',
    products: [{ id: 1, name : 'Hammer' }]
}
```

Optional metadata

Rules to Writing Good Actions

Write them upfront

Divide by event source

Many specific actions

Capture events not commands

Descriptive debug logs

Two Action Creator Helper Functions

```
import { createAction} from ‘@ngrx/store’;

createAction();
```

Two Action Creator Helper Functions

```
import { createAction, createActionGroup } from '@ngrx/store';

createAction();

createActionGroup();
```

Action Creator - createAction

```
import { createAction } from '@ngrx/store';

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

Action Creator - createAction

```
import { createAction, emptyProps, props } from '@ngrx/store';

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

export const productsLoadedSuccess = createAction(
    '[Products API] Products Loaded Success'
    props<{ products: Product[] }>()
);
```

Action Creator - createAction

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

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

export const productsLoadedSuccess = createAction(
    '[Products API] Products Loaded Success'
    props<{ products: Product[] }>
);

export const productsLoadedFail = createAction(
    '[Products API] Products Loaded Fail'
    props<{ message: string }>
);
```

Action Creator - createActionGroup

```
{
  type: '[Products Page] Load Products'
}
```

```
import {                                        ore';

export const Product    geActions = createActionGroup({
    source: 'Products Page',
    events: {
        'Load Products': emptyProps(),
    },
});
```

Action Creator - createActionGroup

```
import {

export co
    sour
    even

    },
});
```

```
{
  type: '[Products API] Products Loaded Success',
  products: [{id: 1, name: 'Hammer'}]
}

{
  type: '[Products API] Products Loaded Fail',
  message: 'something went wrong getting products'
}
```

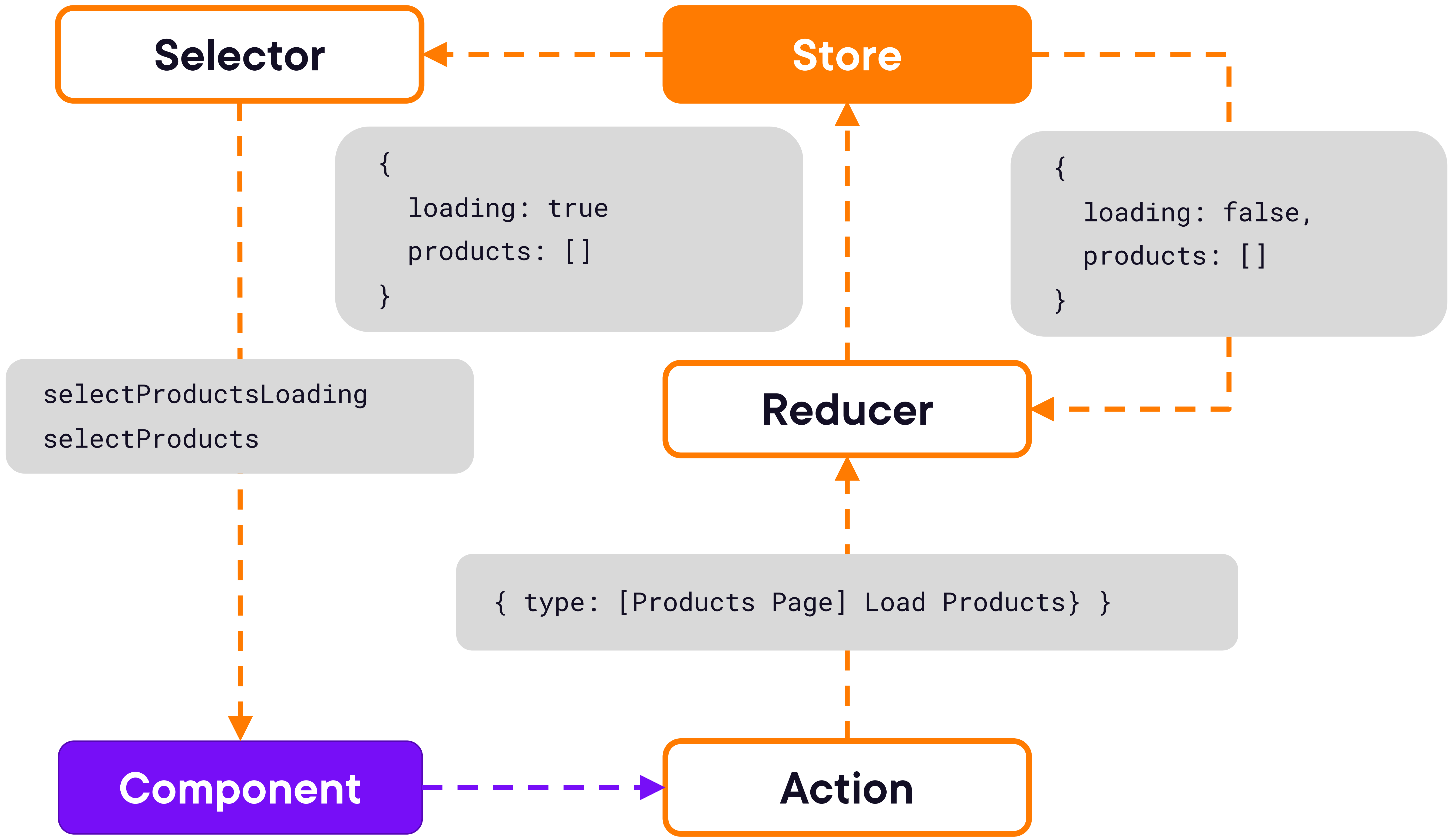
```
export const ProductsAPIActions = createActionGroup({
    source: 'Products API',
    events: {
        'Products Loaded Success': props<{ products: Product[] }>(),
        'Products Loaded Fail': props<{ message: string }>()
    },
});
```

Using Action Creators

```
import { ProductPageActions } from '../state/products.actions';

toggleShowProductCode() {
  this.store.dispatch(ProductPageActions.loadProducts());
}
```

State Lifecycle – Load Products



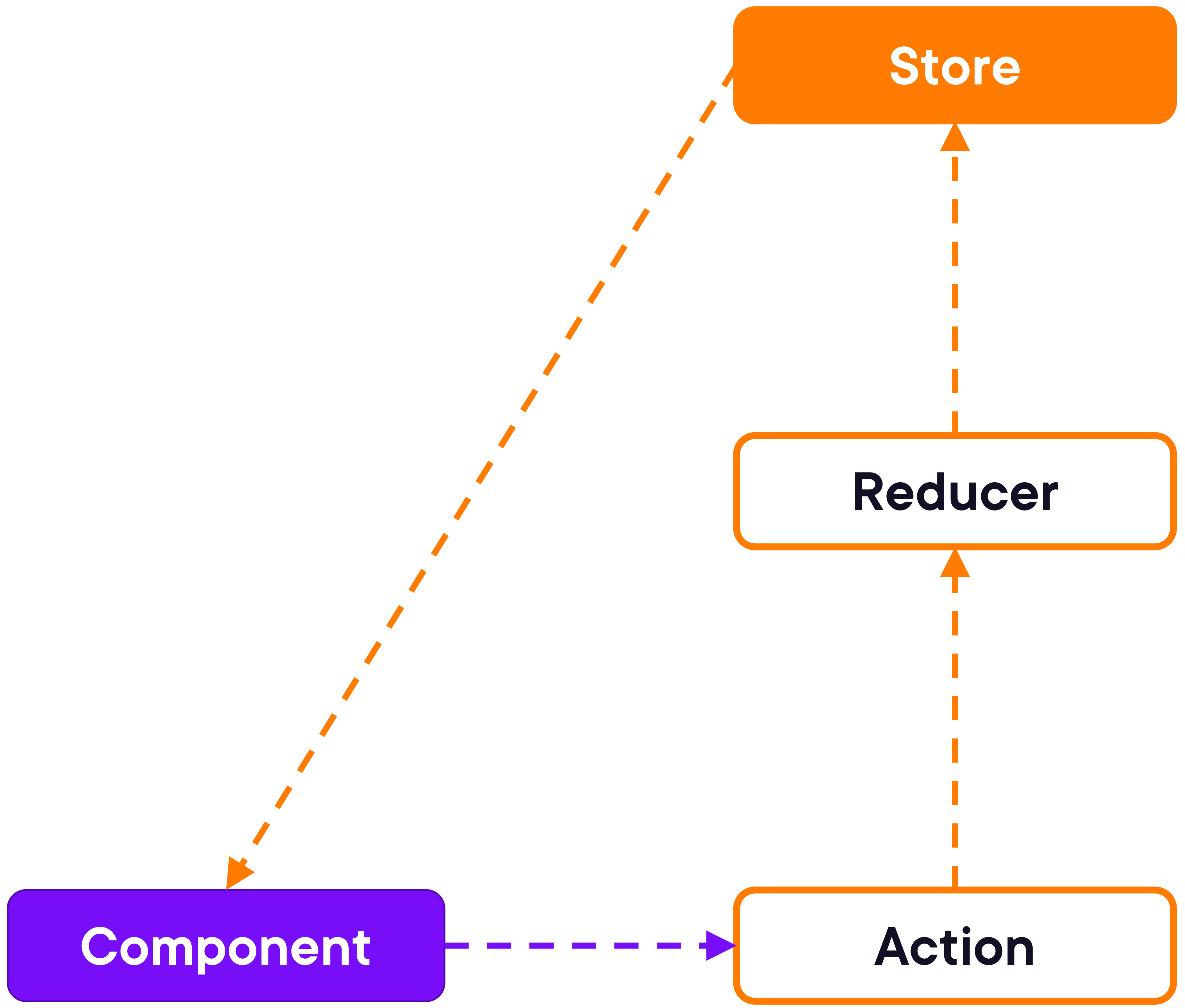
```
{
  loading: true
  products: []
}
```

```
{
  loading: false,
  products: []
}
```

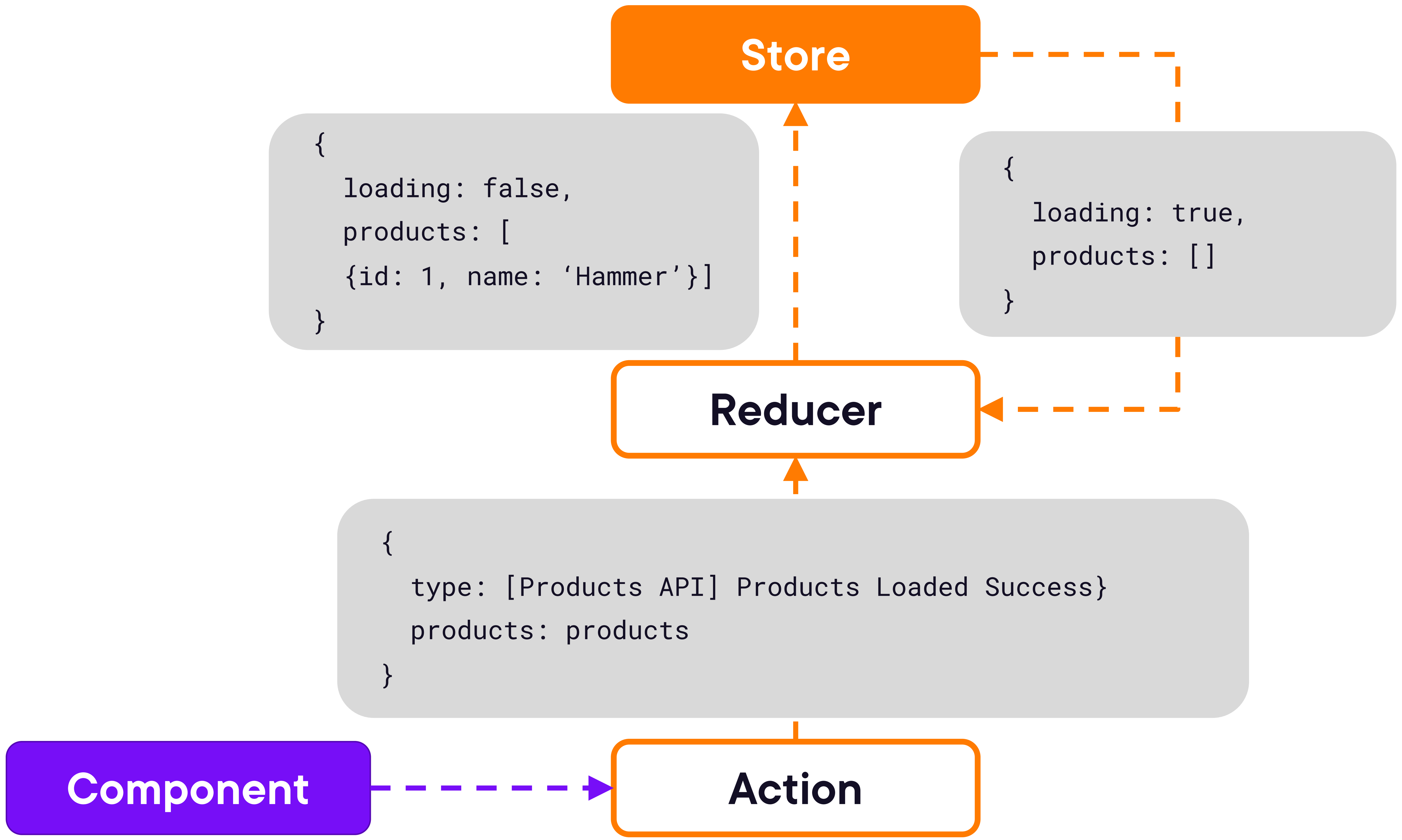
```
selectProductsLoading
selectProducts
```

```
{ type: [Products Page] Load Products} }
```

State Lifecycle – Load Products



State Lifecycle – Products Loaded Success

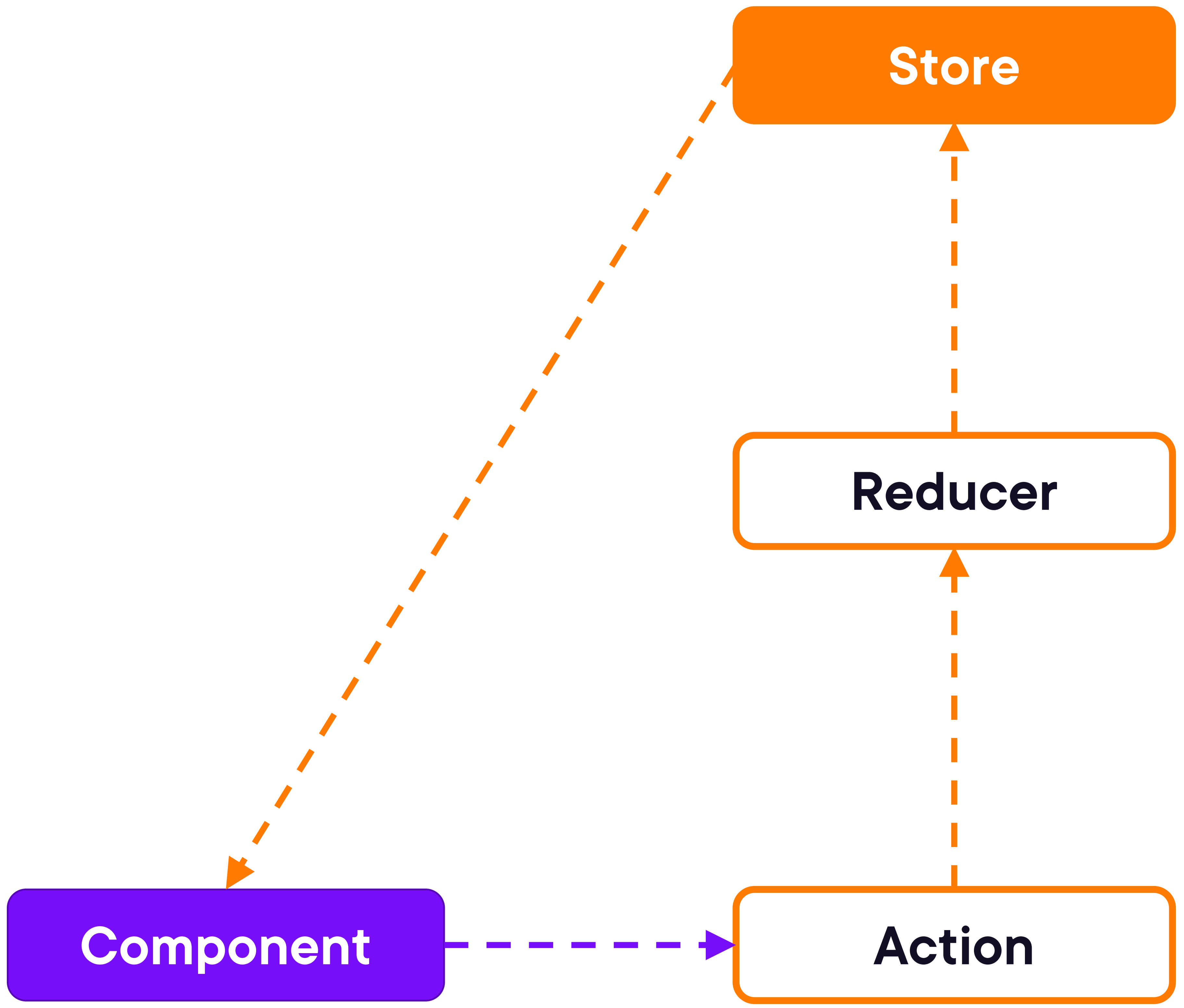


```
{
  loading: false,
  products: [
  {id: 1, name: 'Hammer'}]
}
```

```
{
  loading: true,
  products: []
}
```

```
{
  type: [Products API] Products Loaded Success}
  products: products
}
```

State Lifecycle – Load Products



Demo

Adding action creators

- Create products.actions.ts file
- Add createActionGroup methods
- Swap inline to shared actions
- Update our reducer

Demo

Use load products actions

- Update products page component
- Update products reducer