Angular - Maatwerk

Peter Kassenaar
info@kassenaar.com

NgRx Effects

Talking to external resources and dispatching a new action upon result

For Instance (when to use ngrx/effects):

- Talking to a RESTFul server == a *side effect* with implications on the store.
  - Hence the name, `ngrx/effects`
  - So, it is a side effects model for `ngrx/store`
- Listen for `ngrx/store` actions
- Isolate effects from components and reducers
- Communicate outside of Angular, notify the store when changes are complete
  - Perfect for Asynchrounous operations!

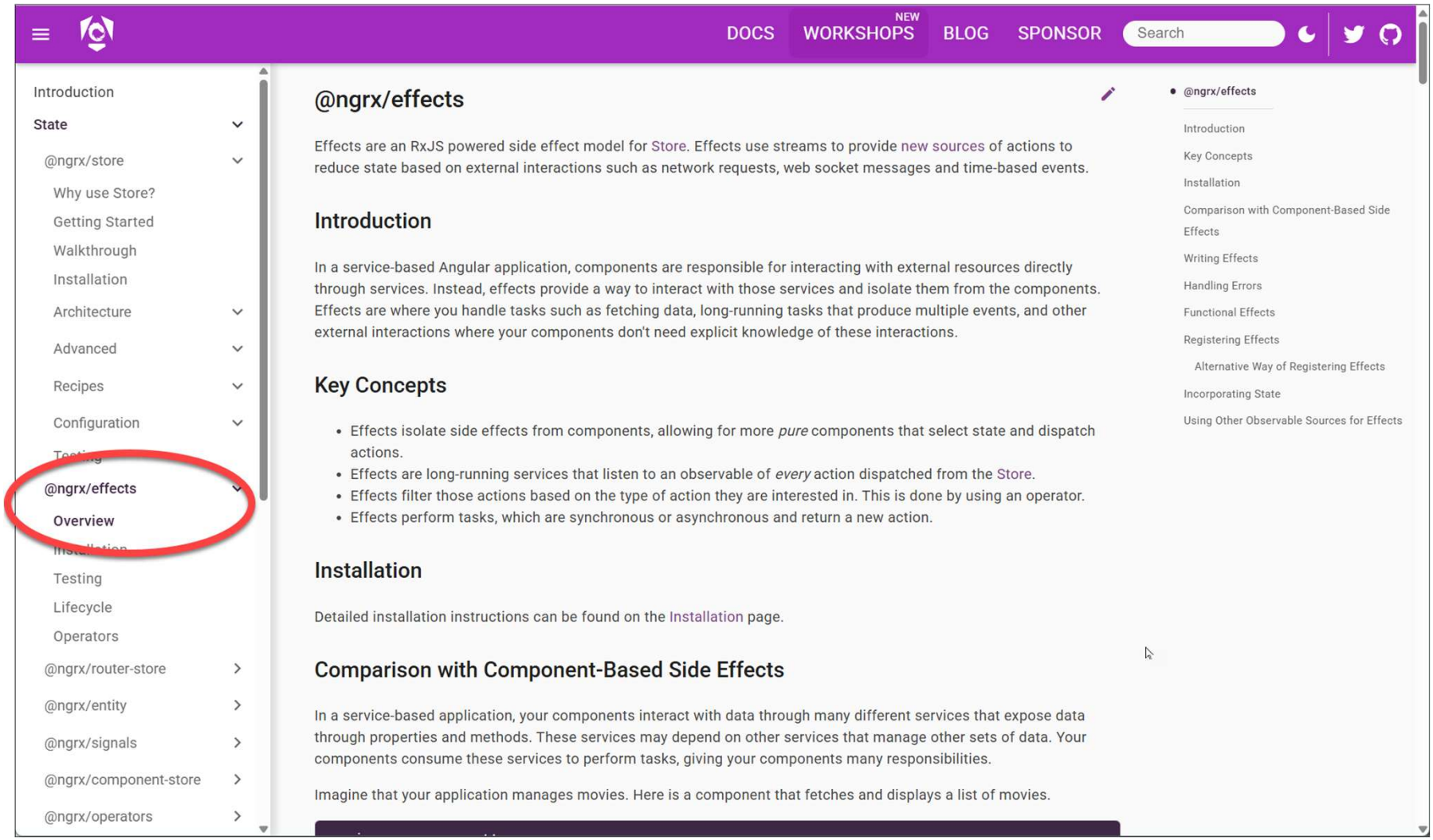
What is `@ngrx/effects`?

"@ngrx/effects is part of the NgRx state management library.

It lets you listen for dispatched actions and perform side effects (like HTTP requests, logging, navigation, etc.) outside of components and reducers. Reducers must be pure functions—no side effects allowed.

Effects solve the problem of handling async logic in a clean, testable, and reactive way."

Latest information: @ngrx.io

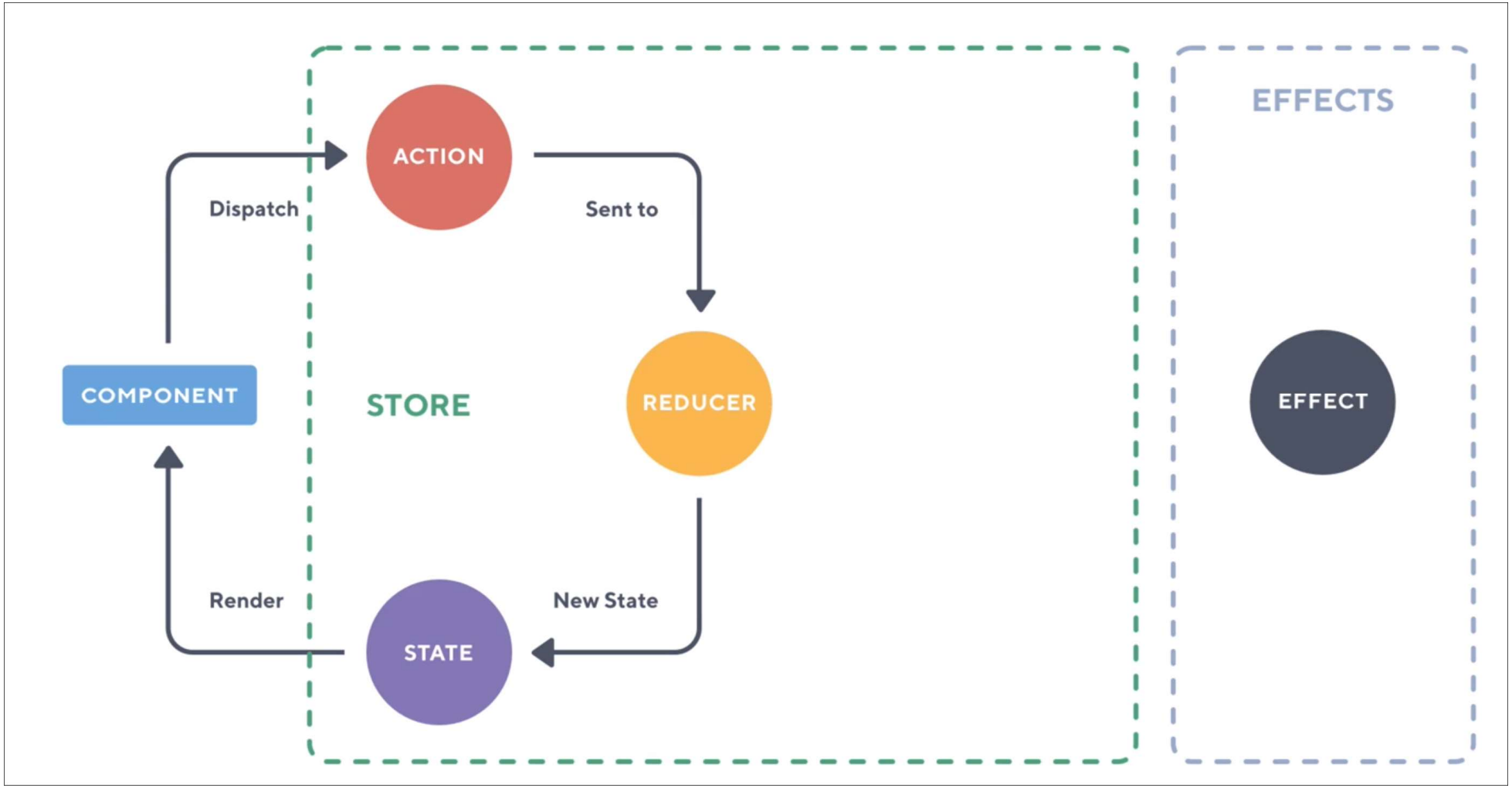


https://ngrx.io/guide/effects

Basic flow

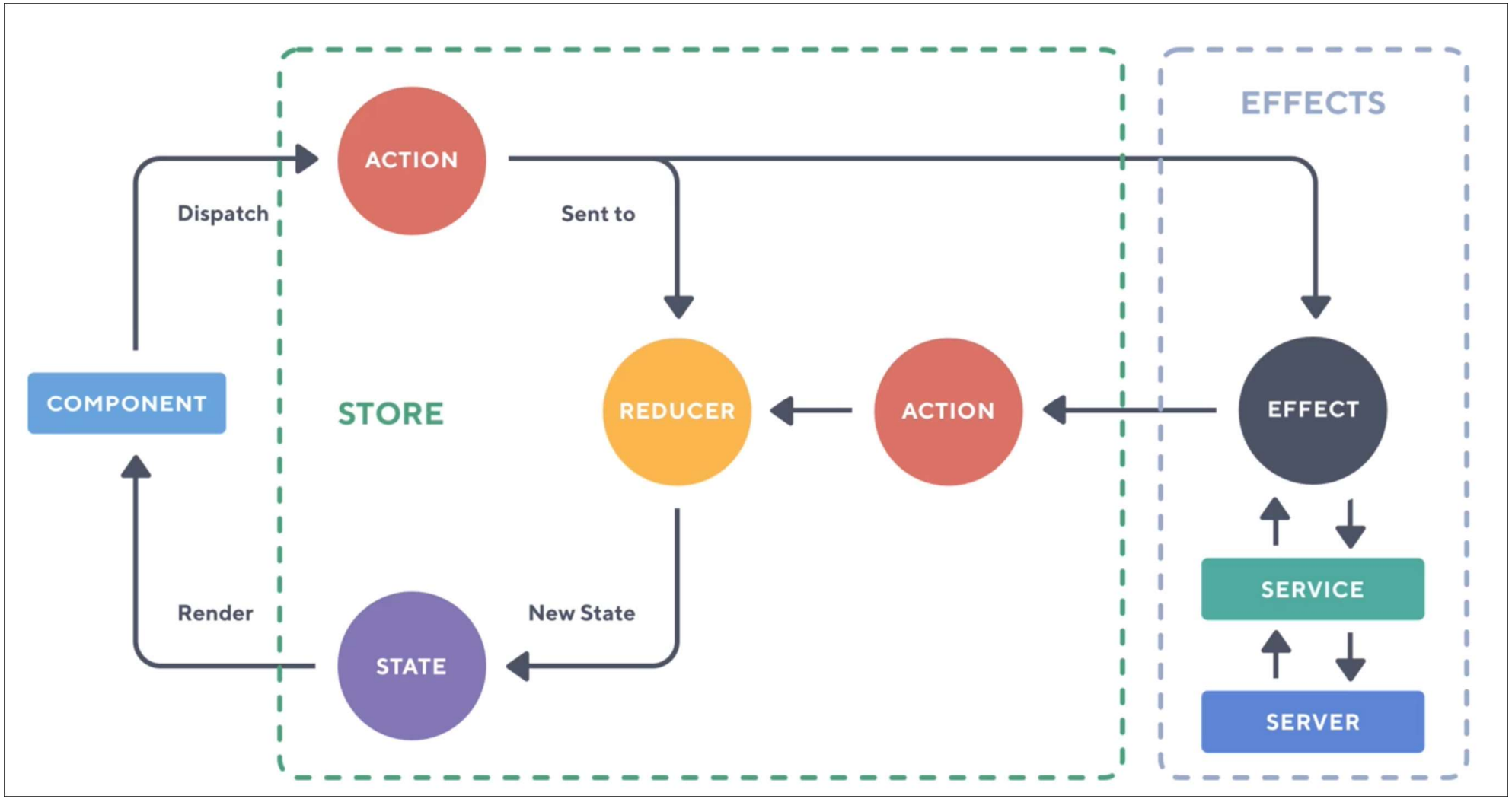
1. User dispatches an action (for example `loadCities`).
2. Effect listens to it, runs some async code (like HTTP call)
3. On success/failure, effect dispatches another action
   - For instance `loadCitiesSuccess` with the cities as payload, or `loadCitiesFailure` with the error as payload

Effects flow



https://platform.ultimateangular.com/courses/ngrx-store-effects/lectures/3919211

Effects flow



https://platform.ultimateangular.com/courses/ngrx-store-effects/lectures/3919211

So, an effect...

1. Listens to store `Actions`.
   - YOU define which action an effect listens to
2. Performs an action (such as talking to a webserver)
3. Dispatches the result to the reducer as a new `Action`.
4. The reducer in turn updates the store/state

Adding @ngrx/effects

```
npm install @ngrx/effects --save
```

OR

```
ng add @ngrx/effects
```

This also :

- Updates `package.json` and `npm install`
- Create a `src/app/app.effects.ts` file with an empty `AppEffects` class.
- Create a `src/app/app.effects.spec.ts` file with a basic unit test.
- Update your `src/app/app.module.ts`
  - imports array with `EffectsModule.forRoot([AppEffects])`.

We're doing this manually here, so use

```
npm install @ngrx/effects
```

Adding effects to the module

- Import `EffectsModule` from `@ngrx/effects`
- Use `EffectsModule.forRoot()` for root module
- Use `EffectsModule.forFeature()` for feature module

```
// Effects
import {EffectsModule} from '@ngrx/effects';

// exported effects from general index file
import { effects } from './effects/index';

@NgModule({
   …
   imports     : [
      …
      EffectsModule.forRoot(effects),    // array of effects
   ],
   …
})
export class AppModule {
}
```

Example effect - /260-ngrx-effects

```
@Injectable()
export class CitiesEffects {

  constructor(private actions$: Actions,
              private cityService: CityService) {
  }

  loadCities$ = createEffect(() => this.actions$.pipe(
    // 1. Listen to this specific event (fired from app.component.ts)
    ofType(LoadCitiesViaEffect),
    mergeMap(() => {
      return this.cityService.loadCities()  // 2. talk to API
        .pipe(
          map((cities: City[]) => loadCitiesSuccess({cities})), // 3. Dispatch new action
          catchError(() => of(loadCitiesFail())) // 4. catch error and dispatch failure action
        );
    })
  ));
}
```

Difference with service-based approach

- The Service *does not* subscribe.
- Instead it just fetches content from an URL and returns it to the effect
- …which in turn dispatches a new Action to update the store.

```
// this is now called from the Effect
loadCities() {
  return this.http.get(BASE_URL)
    .pipe(
      tap(res => console.log('We talked to json-server and received: ', res)),
      finalize(() => 'Getting cities complete...')
    );
  // Note: when using effects, no more subscriber in the service!
}
```

Workshop

- Start from `/260-ngrx-effects`
  - Use `cities.json`, or another `.json`-file you create yourself
  - Implement the `Effect()`'s for Adding and Removing a city

- OR: Create a blank project:
  - Add `@ngrx/effects` to the project and to the module
  - Use `createEffect()` to load an external resource (your `.json`-file)
  - Notify the store once the resource is loaded and update the UI.