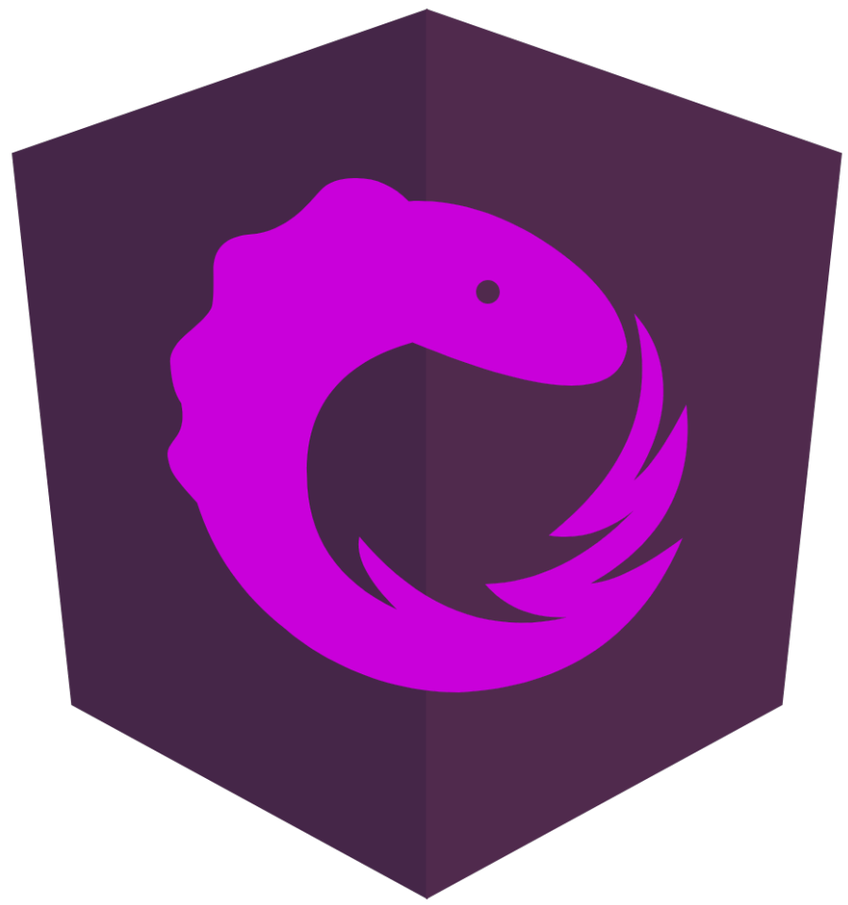
Angular NgRx

Ing. Luigi Brandolini



NgRx

https://ngrx.io/

Ing. L.Brandolini

State Management

- Lo stato è considerato l'insieme delle proprietà che devono essere mantenute da un'applicazione
- Queste sono legate ai dati provenienti da servizi RESTFul e impostazioni utente
- La gestione dello stato, viene effettuata su due livelli:
  - Componente
  - Applicazione

State Management

- La gestione dello **stato applicativo** viene effettuata mediante un framework chiamato NgRx

- Si utilizza in applicazioni complesse, al crescere delle user interactions e con sorgenti multiple di dati

- L'idea centrale consiste nel separare la User Interface dalla Data Architecture

- NgRx consente di sviluppare applicazioni reactive, garantendo un insieme di proprietà

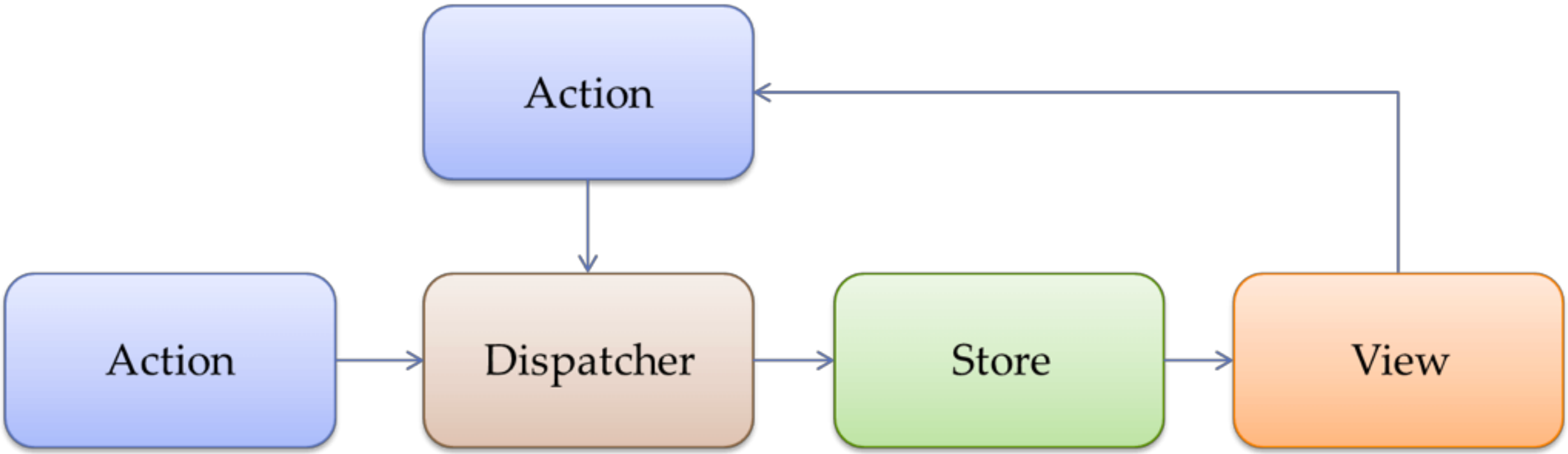
Reactive Programming

- Con il termine "reactive programming", si intende un modello basato su:
  - programmazione funzionale
  - chiamate asincrone dei servizi
  - data streaming
  - propagazione del cambiamento
  - pattern: Iterator e Observables

NgRx

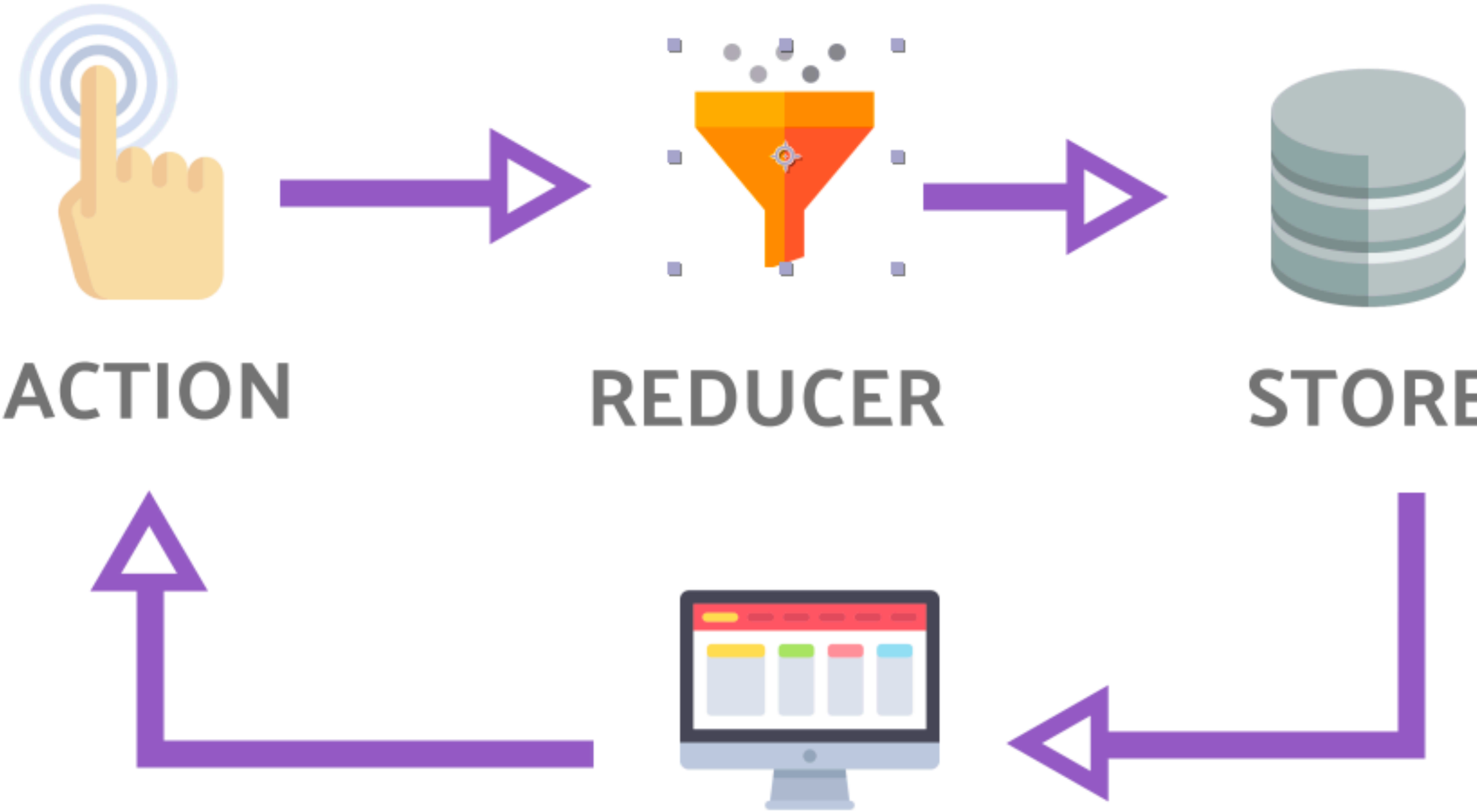
Flux Pattern

- Introdotto da Angular (2+) e React (Facebook)



- Attori principali
  - Azioni
  - Dispatcher
  - Store (possono essere multipli e includono: dati + logica)
  - View

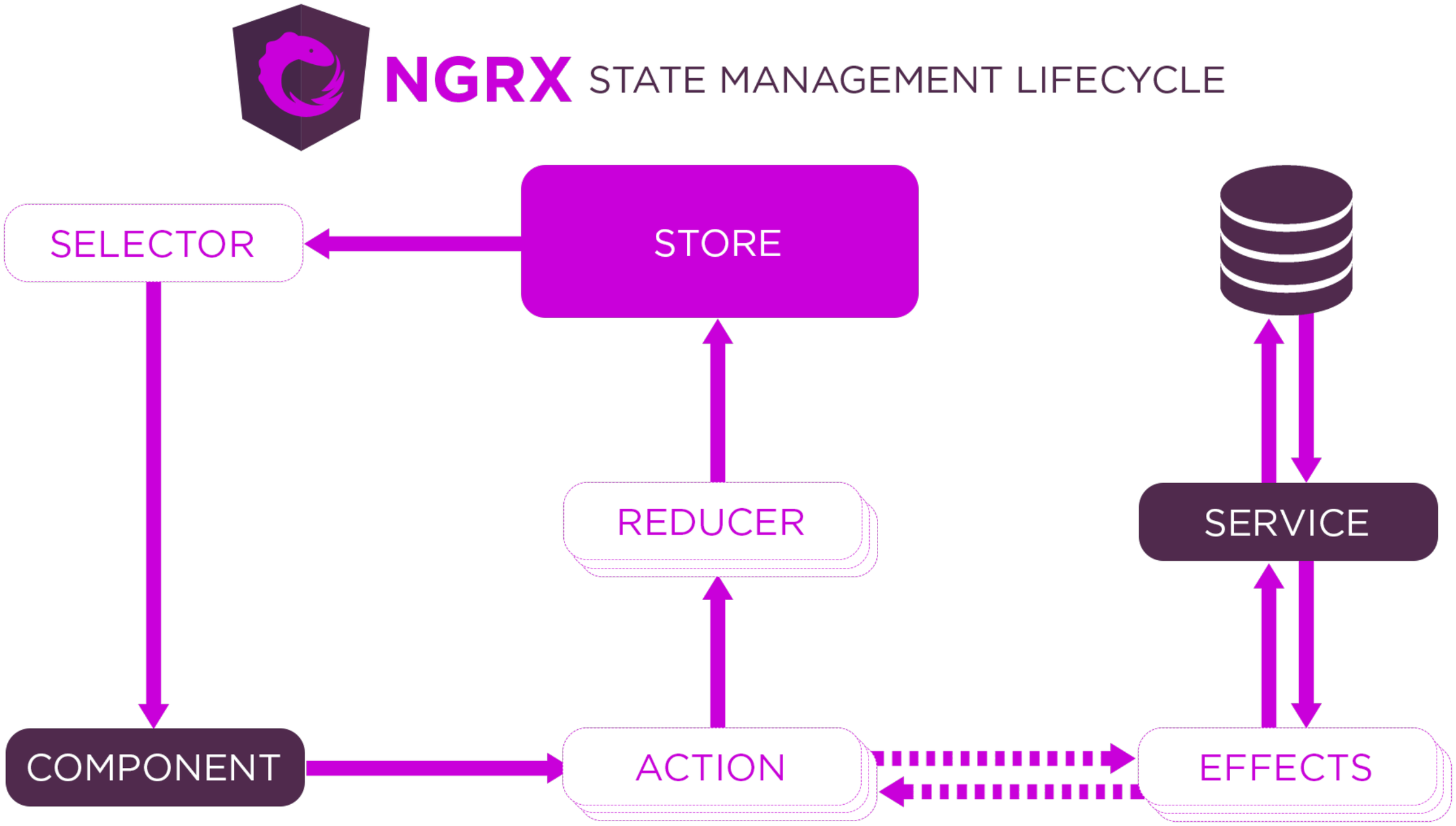
Redux Pattern



Redux Pattern

- Non c'è un nodo dispatcher
- Store
  - incapsula lo stato applicativo (senza logica di business)
  - *single-source-of-truth*
  - genera un nuovo stato
- Reducer
  - Funzioni che incapsulano logica di business

NgRx - Lifecycle



NgRx - Proprietà

- Serializability: normalizzazione e passaggio dello stato attraverso Observables alle altre componenti, con serializzazione finale (eg: localStorage)

- Type Safety: basata interamente su TypeScript

- Encapsulation: mediante NgRx Effects e Store

- Testability: isolation testing, mediante librerie offerte come `provideMockStore` e `provideMockActions`

- Performance: lo Store è un immutabile e la change detection basata su strategia "onPush"

Lifecycle

- Al top dell'architettura c'è lo **Store**, ispirato da Redux, che rappresenta lo state container

- Le **actions** sono eseguite attraverso le componenti e innescano cambiamenti allo Store mediante l'attivazione di funzioni chiamate **reducers**

- I **selectors**, invece, sono funzioni che servono ad accedere allo Store per recuperare ed elaborare i dati presenti nello stato applicativo

Installazione

- Via Npm:

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

- Via ng add (Angular 6+):

```
ng add @ngrx/store
```

Effects

- Opzionalmente è possibile aggiungere i cosiddetti Effects all'applicazione

- Sono classi decorate con `@Injectable()` che consentono di eseguire *side effect* legati alle chiamate asincrone verso sistemi esterni

- Vengono iniettate a livello di componente

- Riducono il livello di accoppiamento tra componenti e servizi

Effects

- Isolano i *side effects* dalle altre componenti
- Rappresentano servizi long-running che ascoltano, attraverso un Observable, ogni azione partita dallo Store
- Filtrano quelle azioni basate sul tipo a cui sono interessati (attraverso un operatore)
- Eseguono task, sincroni o asincroni, restituendo una nuova azione

Installazione

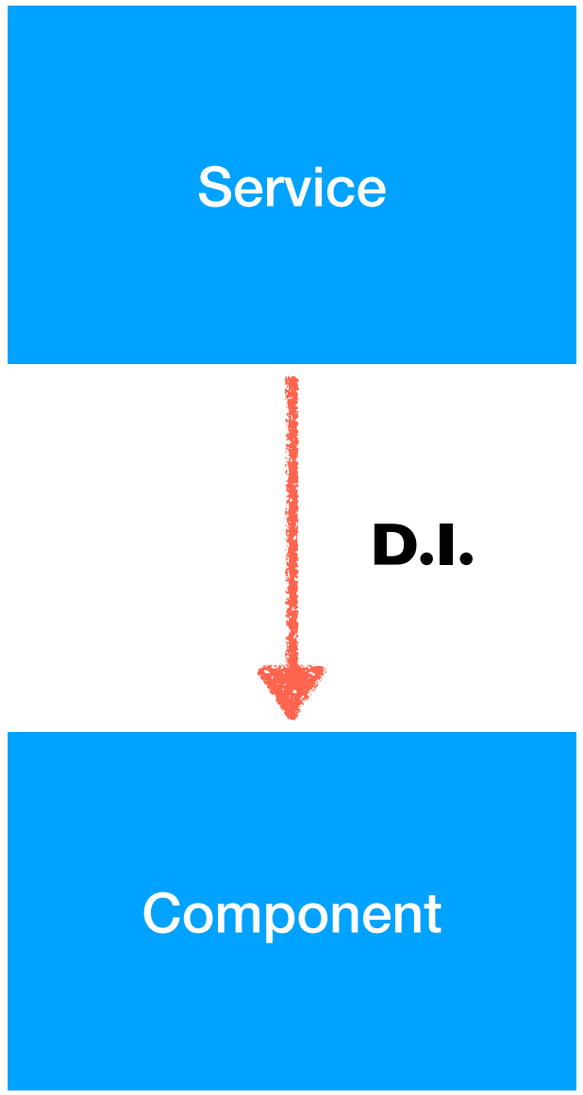
- Via Npm:

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

- Via `ng add` (Angular 6+):

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

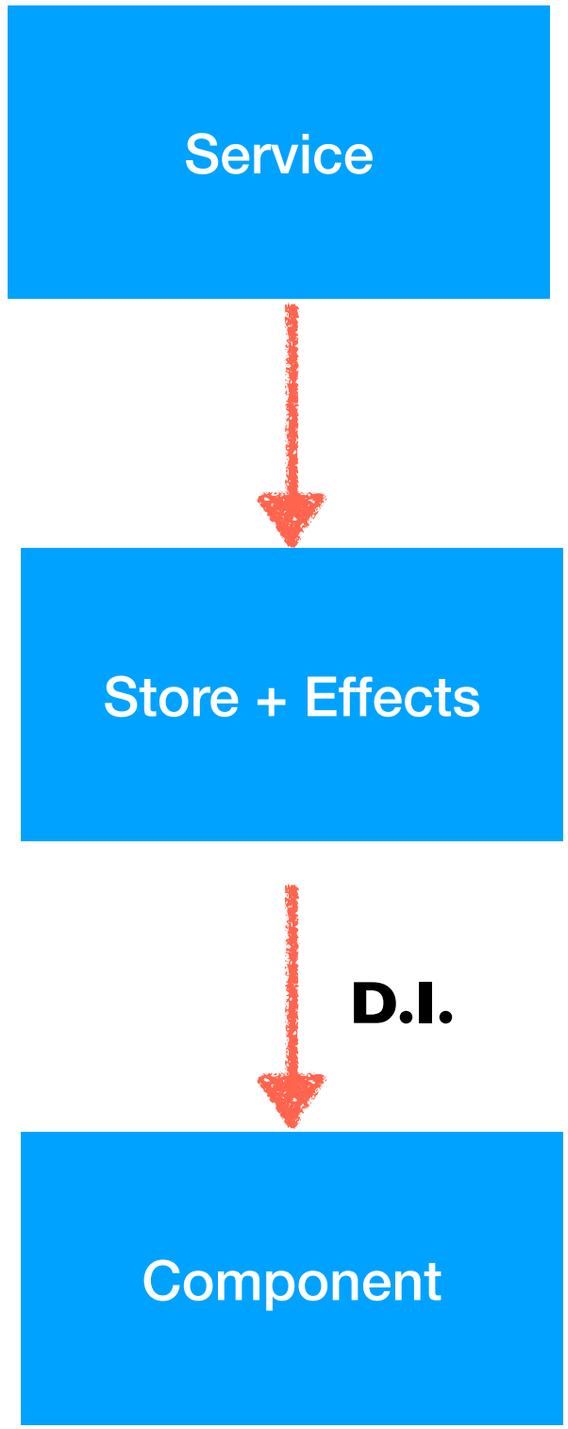
Vecchia architettura



Vecchia architettura

- Il componente ha molteplici responsabilità:
  - Gestione stato
  - Effettuare il fetching dei dati attraverso il corrispondente servizio
  - Modificare lo stato al suo interno

Nuova architettura



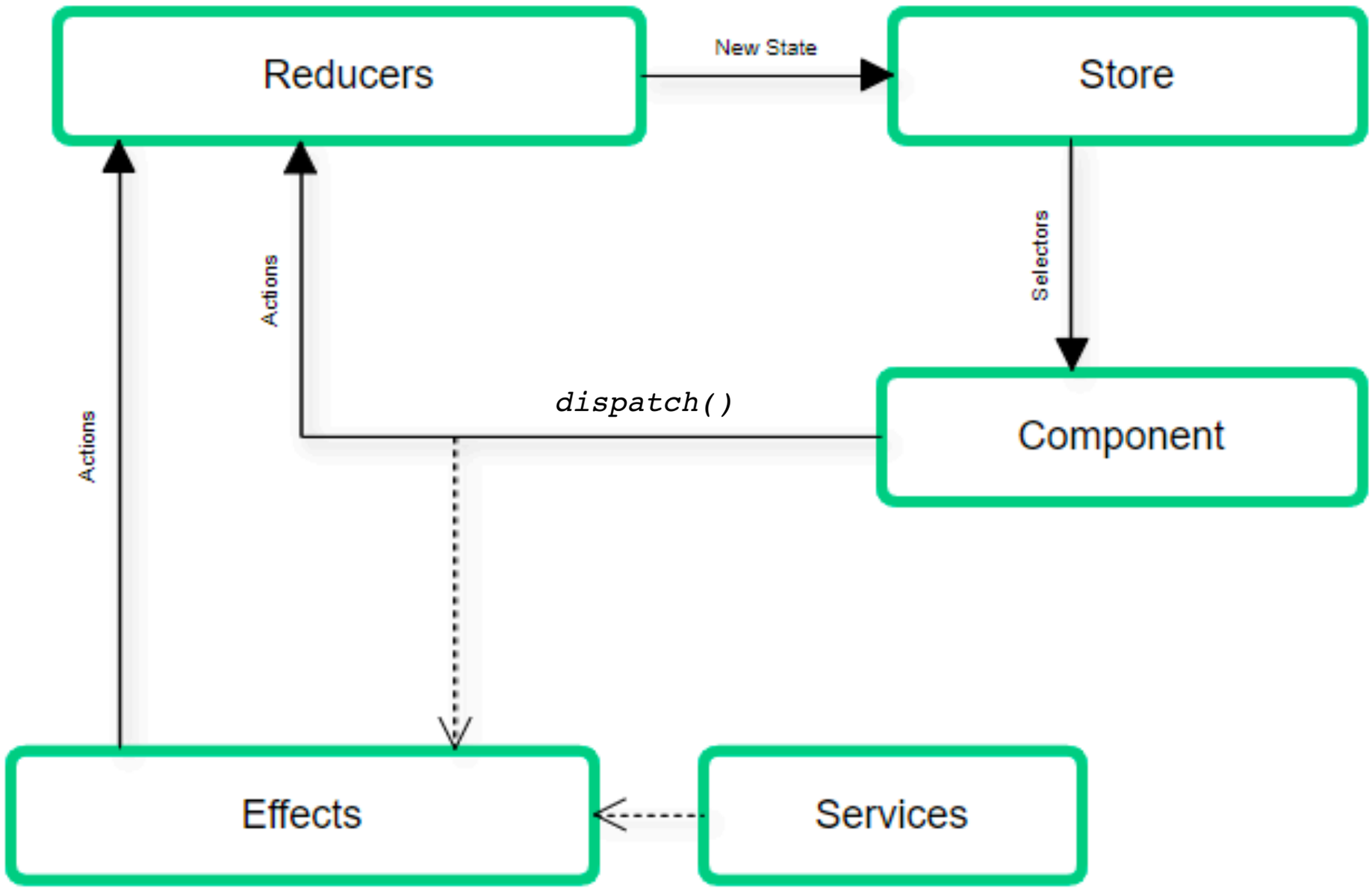
Nuova architettura

- Gli Effects usati assieme allo Store, diminuiscono la responsabilità del componente

- Questa architettura diventa particolarmente utile quando ci sono molteplici sorgenti informative che recuperano dati, attraverso più servizi che invocano altri servizi

- Gli Effects consentono ai services di essere meno stateful ed eseguono interazioni esterne.

Nuova architettura



Credits

Dott. Ing. Luigi Brandolini

Software Engineer, IT Professional Instructor

Contacts:

E-Mail: luigi.brandolini@gmail.com