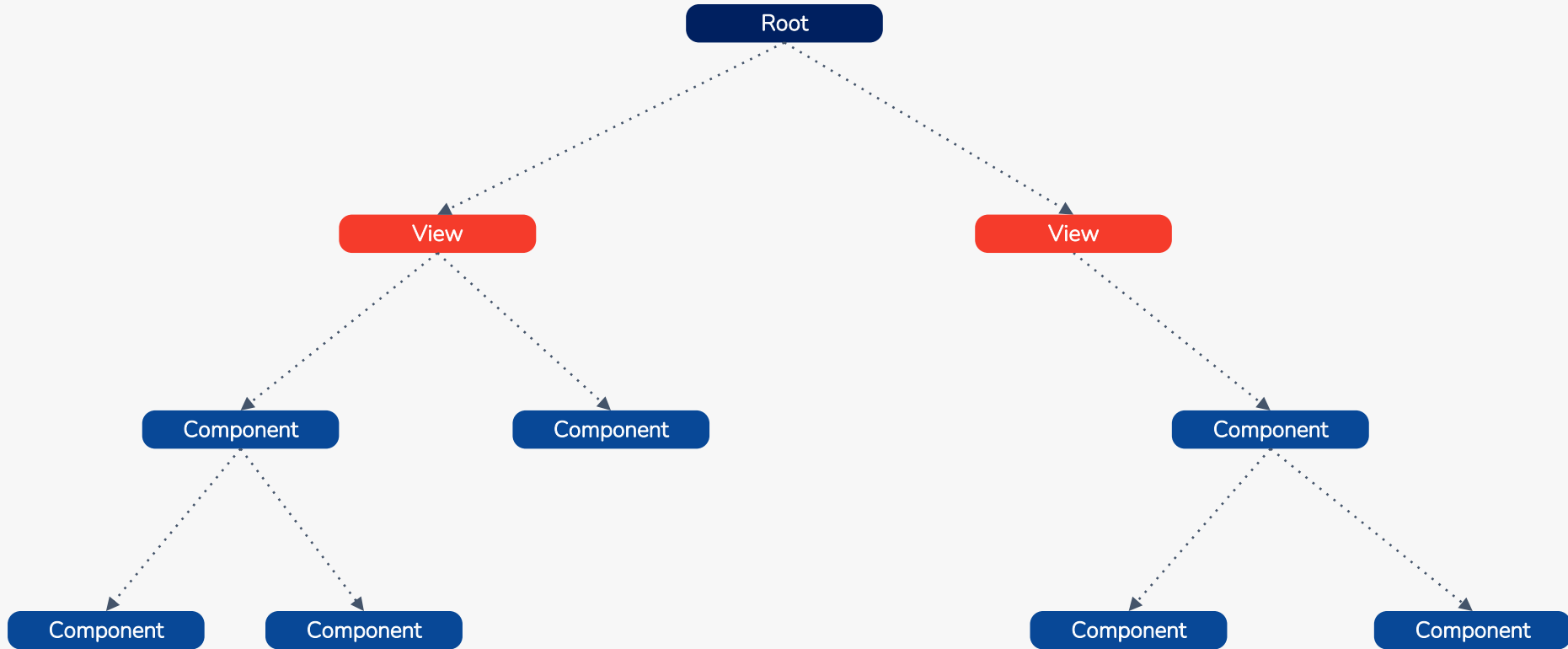




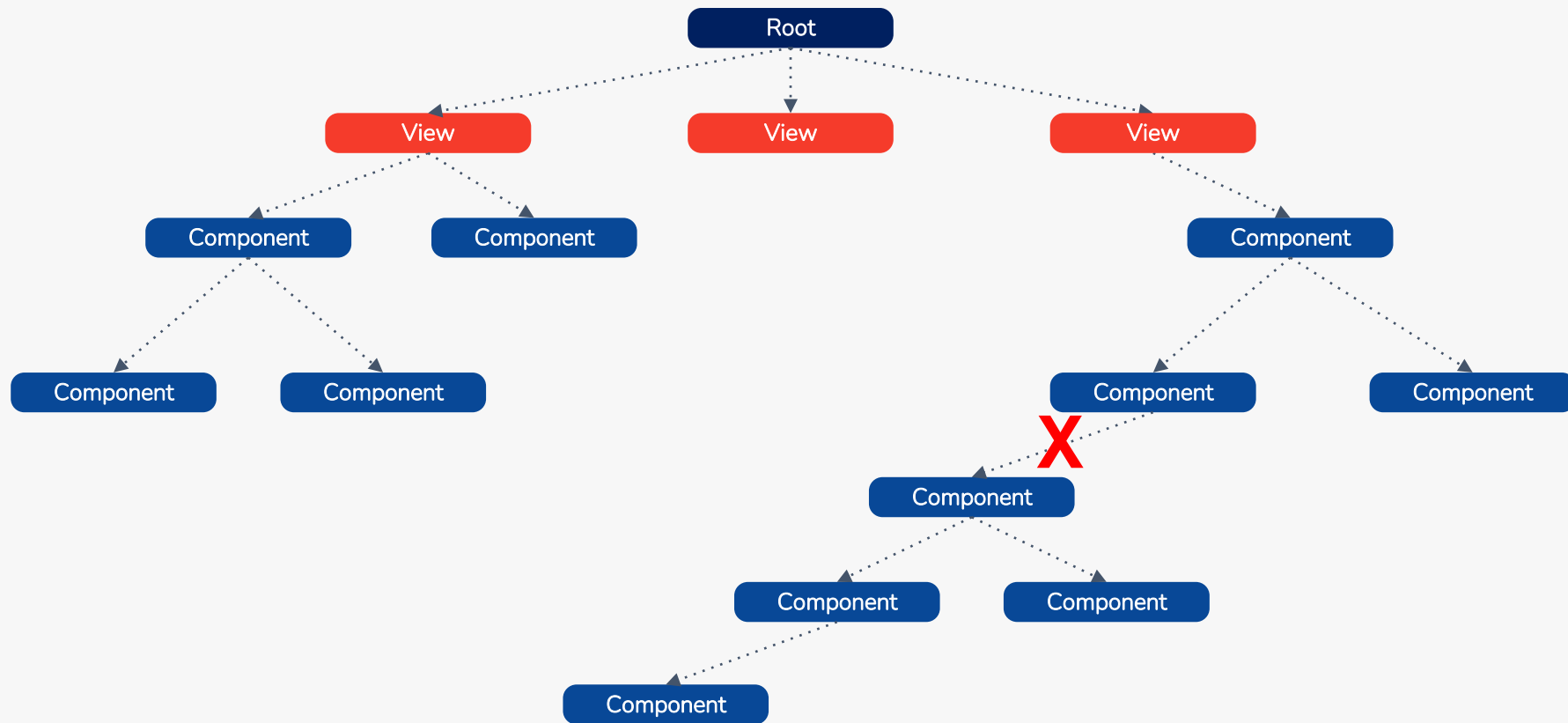
State Manager

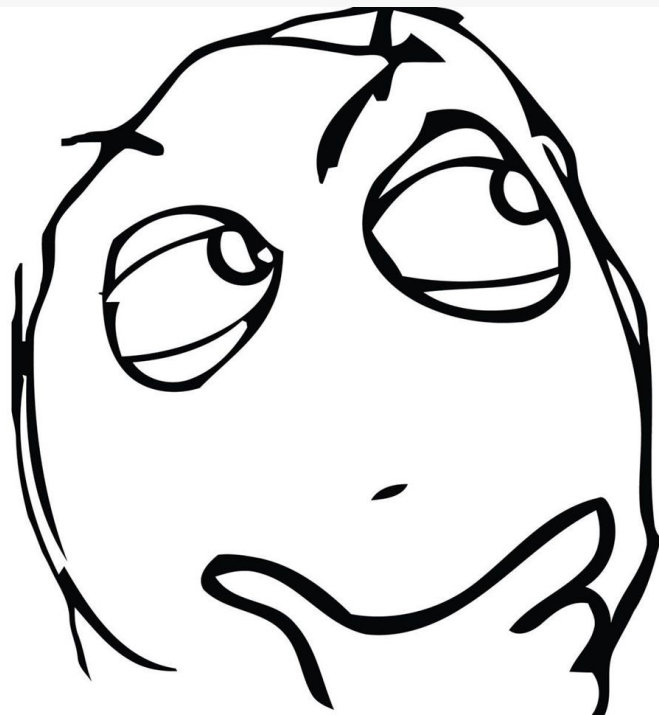
Fravezzi Mattia
m.fravezzi@almaviva.it

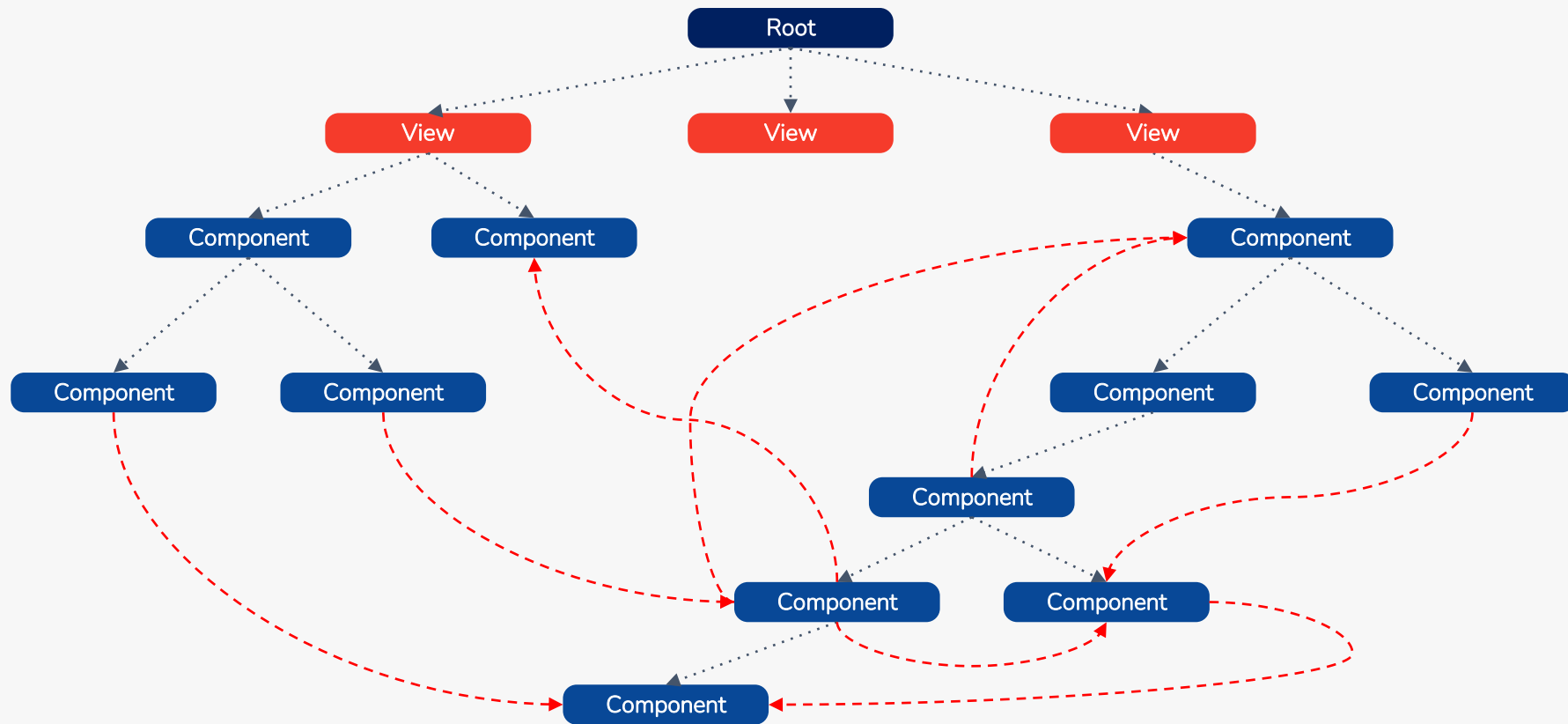
**ABBIAMO SEMPRE BISOGNO DI
UNO STATE MANAGER?**











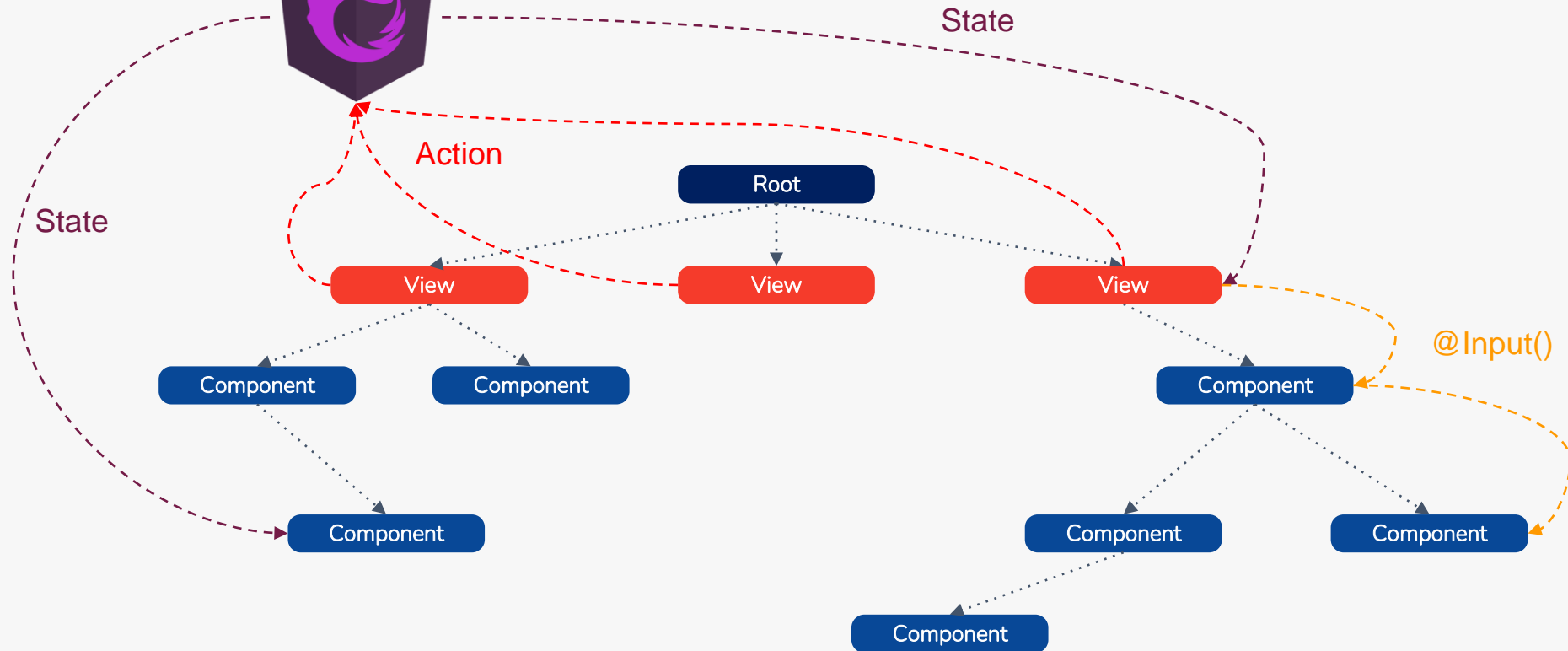


COME GESTIAMO LO STATO APPLICATIVO?

NGRX



STORE



NGRX - Fasi

- **Actions:** le azioni, effettuate nella maggior parte dei casi dall'utente, che potranno essere tracciate e identificate
- **Reducers:** si occupano di manipolare lo stato sulla base delle azioni
- **Effects:** gestione separata di side effect come XHR, gestione router, sync con localStorage...
- **Selectors:** recuperare una porzione di stato

Principi Redux

- STORE: singola sorgente di verità
- State è readonly
- Solo le actions possono modificare lo stato (usando i reducer)

Vantaggi

- Separazione di responsabilità
- State prevedibile
- Facile da testare
- Codice Manutenibile
- Type Safe (TS)
- Reactive store (RXJS)
- Separazione UI/STATE
- Non si perde il DataFlow
- Sync Router & LocalStorage
- Time Travel Debugging
- Import / Export State
- Più difficile creare problemi su sezioni diverse

Inspector

filter...

Commit

@ngrx/store/init	12:59:09.90
[Invoice] Create success	+00:00.32
[Invoice] Open	+00:06.61
[ui] Close Invoices panel	+00:00.95
[ui] Open Client panel	+00:02.06
[Invoice] Set Client	+00:01.97
[ui] Close Client panel	+00:01.25
[Invoice] Save	+00:04.19
[Invoice] Edit	+00:00:00
[Invoice] Edit success	+00:00.02
[ui] Close Invoices panel	+00:00:00
[ui] Close Client panel	+00:00.00

NgRx Store DevTools

Action

Action

State

Diff

Tree

Chart

Raw

▼ invoice (pin)

▶ client (pin): { name: "ACME COMPA_", address: null }

subject (pin): "Consulenza Sviluppo"

date (pin): "2019-07-18"

total (pin): 359

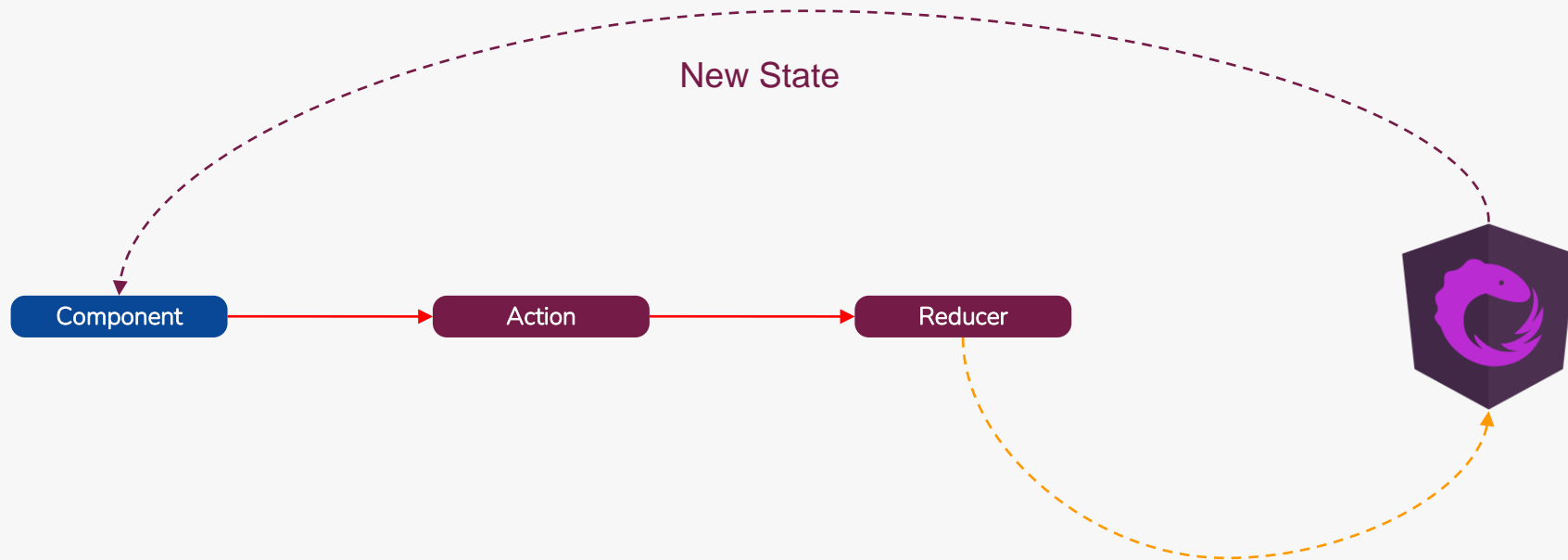
▶ items (pin): [{-}]

type (pin): "[Invoice] Save"

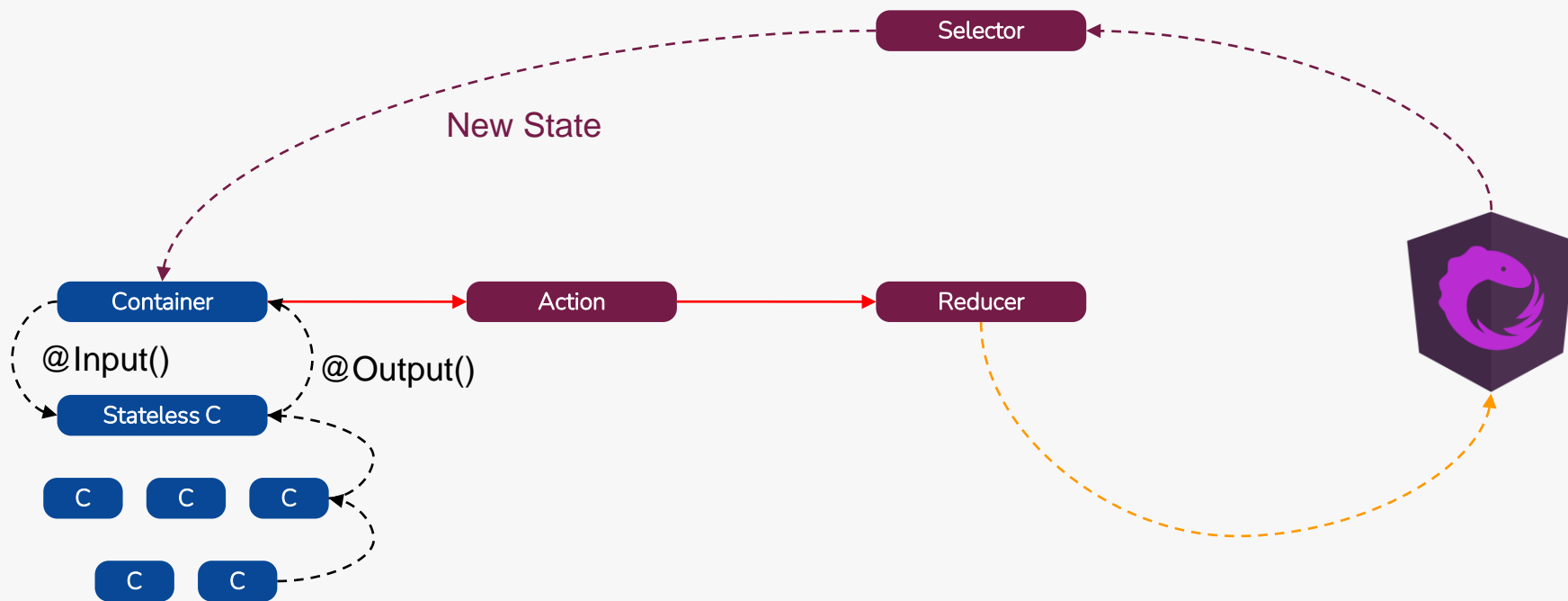
Svantaggi

- Non è facile da imparare
- Conoscere bene RXJS
- Richiede di scrivere molto più codice
- Task più lunghi da svolgere

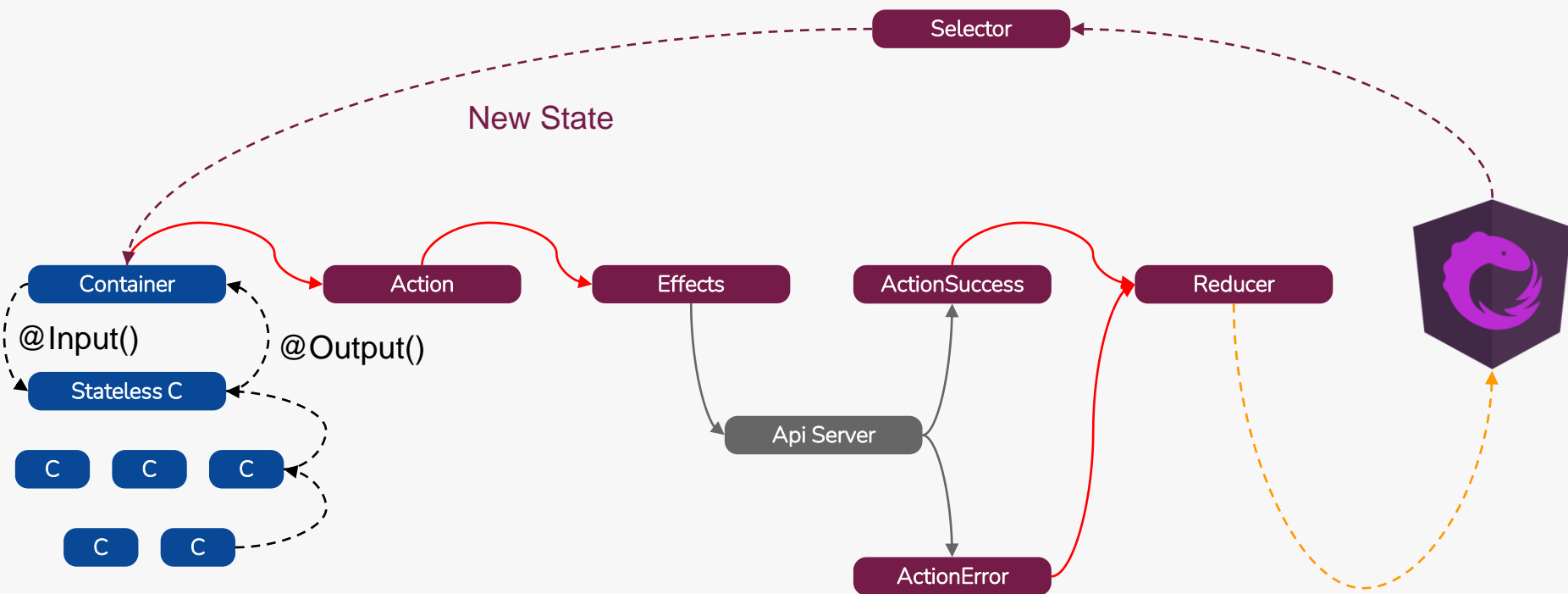
Redux Flow



Ngrx Flow



Ngrx Flow - SideEffect



Store

```
export interface AppState {
  profile: ProfileState;
  ...
}

export const reducers: ActionReducerMap<AppState> = {
  profile: profileReducer,
  ...
};
```

```
export interface ProfileState {
  user: User | undefined,
  error: boolean
};
```

```
@NgModule({
  declarations: [
    ...
  ],
  imports: [
    ...
    StoreModule.forRoot(reducers),
  ]
})
export class CoreModule { ... }
```

npm install @ngrx/store --save

Actions

```
export const getProfile = createAction(  
  '[Profile] Get',  
  props<{ id: number }>()  
)  
export const getProfileSuccess = createAction(  
  '[Profile] Get Success',  
  props<{ user: User }>()  
)  
export const getProfileError = createAction(  
  '[Profile] Get Error'  
)
```

Effects

```
@Injectable()
export class ProfileEffects {
  constructor(
    private actions$: Actions,
    private profileService: ProfileService
  ) { }

  getProfile$ = createEffect(() =>
    this.actions$.pipe(
      ofType(getProfile),
      switchMap((action: { id: number }) => this.profileService.getProfile(action.id))
        .pipe(
          map((user: User) => getProfileSuccess({ user })),
          catchError((e) => of(getProfileError()))
        )
    )
  )
}
```

```
@NgModule({
  imports: [
    StoreModule.forRoot(reducers),
    EffectsModule.forRoot([ProfileEffects])
  ]
})
export class CoreModule { ... }
```

npm install @ngrx/effects --save

Service

```
export class ProfileService {  
  constructor(  
    private http: HttpClient,  
    private appConfig: AppConfig  
  ) { }  
  
  getProfile(id: number): Observable<User> {  
    var url = `${this.appConfig.baseUrl}${this.appConfig.endpoints.profiles.baseUrl}/${id}`;  
    return this.http.get<User>(url)  
  }  
}
```

Reducer

```
const initialState: ProfileState = {  
  user: undefined,  
  error: false  
};  
  
export const profileReducer = createReducer(  
  initialState,  
  on(  
    getProfileSuccess,  
    (state, action) => ({ user: action.user, error: false }),  
  ),  
  on(  
    getProfileError,  
    (state, action) => ({ user: undefined, error: true }),  
  )  
);
```


Selector

```
export const getProfile = (state: AppState) => state.profile;

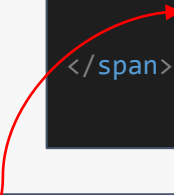
export const getDisplayName = createSelector(
  getProfile,
  (state: ProfileState) => {
    return !!state.user
      ? `Ciao ${state.user?.nome}!`
      : `---`;
  }
)
```

Come si usa?

```
@Component({
  selector: 'app-navbar',
  templateUrl: './navbar.component.html',
  styleUrls: ['./navbar.component.scss']
})
export class NavbarComponent implements OnInit {
  public profileName$: Observable<any> = this.store.pipe(select(getDisplayName) as any);

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

```
<span>
  {{ (profileName$ | async) }}
</span>
```



DEMO



almaviva.it