ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Professional NgRx

3 - Best Practices

Caching

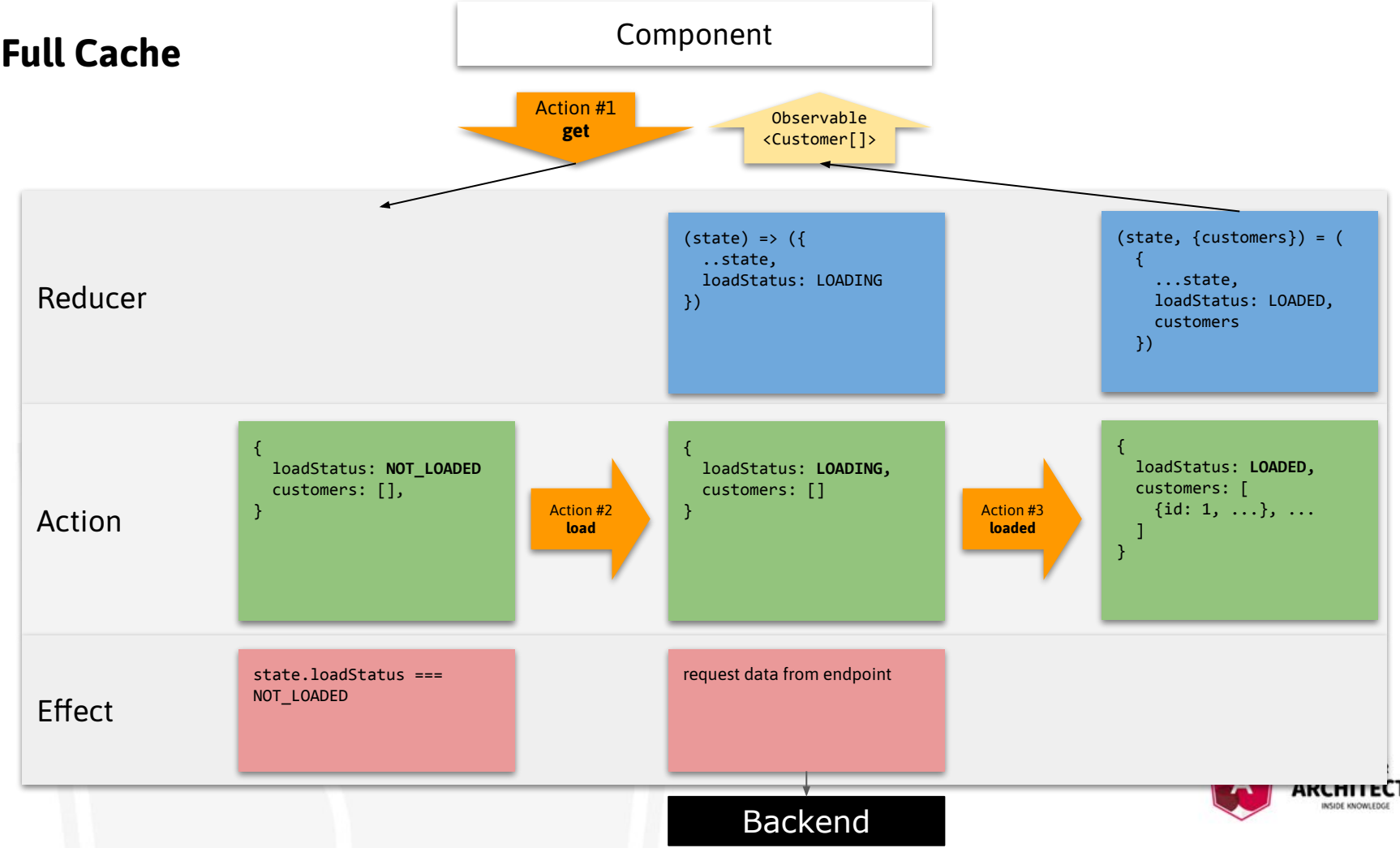
It is "State **Management**"

⇓

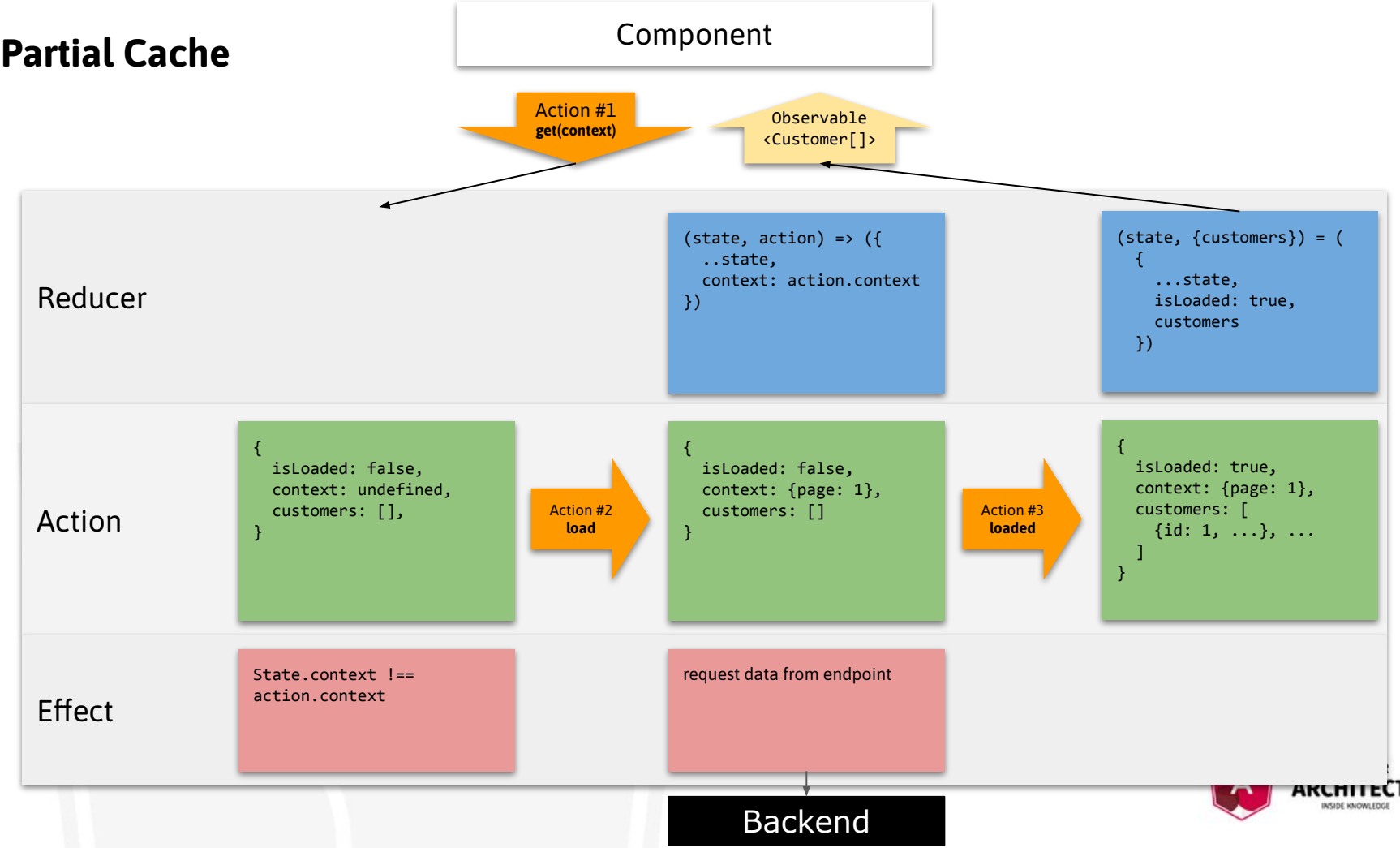
Caching Logic is part of NgRx

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Full Cache


Component
Action #1
get
Observable
<Customer[]>
Reducer
(state) => ({
..state,
loadStatus: LOADING
})
(state, {customers}) = (
{
...state,
loadStatus: LOADED,
customers
})
Action
{
loadStatus: NOT_LOADED
customers: [],
}
Action #2
load
{
loadStatus: LOADING,
customers: []
}
Action #3
loaded
{
loadStatus: LOADED,
customers: [
{id: 1, ...}, ...
]
}
Effect
state.loadStatus ===
NOT_LOADED
request data from endpoint
Backend

Partial Cache

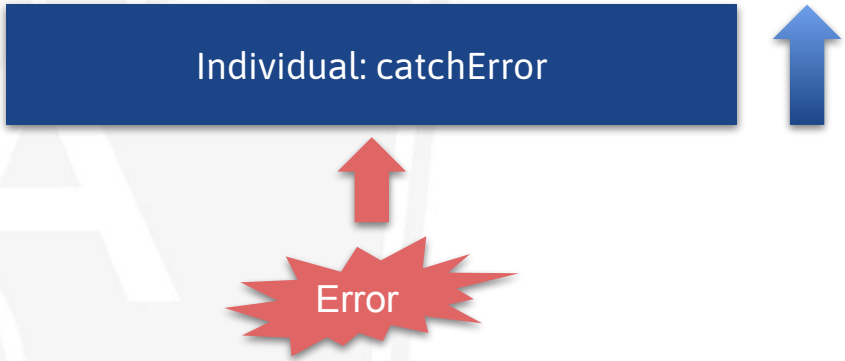


Caching

- Real Caching via operators
- Caching via Logic
- Caching with Invalidation

Error Handling

Error Bubbling



Error Handling

- Error handling is built-in in NgRx Effects
- Careful with `catchError` operator
  - Replaces the original Observable with an alternative
  - Overrides NgRx's handler

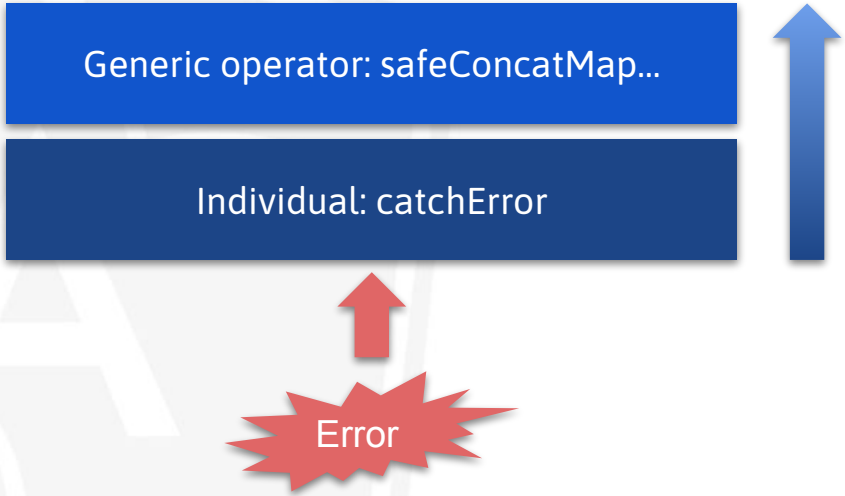
Don't do this!

```
this.actions$.pipe(
  ofType(update),
  concatMap(({ customer }) =>
    this.http.put<Customer[]>(this.#baseUrl, customer)
  ),
  map(() => updateSuccess()),
  catchError(() => of(updateFailed())
);
```

Use catchError only in "completing" Observables

```
this.actions$.pipe(
  ofType(update),
  concatMap(({ customer }) =>
    this.http.put<Customer[]>(this.#baseUrl, customer).pipe(
      map(() => updateSuccess()),
      catchError(() => of(updateFailed()))
    )
  )
);
```

Error Bubbling



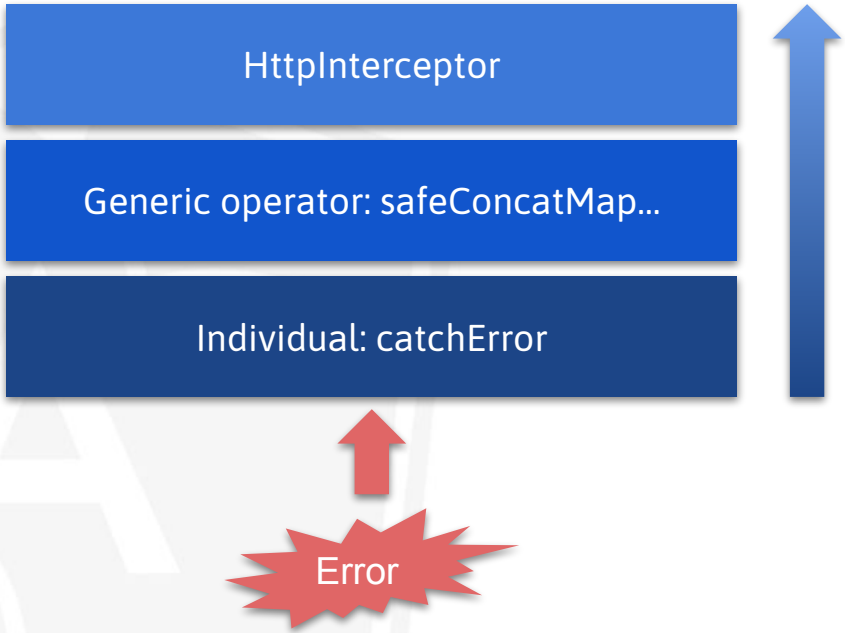
Generic Operator 1/2

```
function safeConcatMap<S, T extends string>(
  project: (value: S) => Observable<TypedAction<T>>
): OperatorFunction<S, TypedAction<T | "NOOP">> {
  return (source$: Observable<S>): Observable<TypedAction<T | "NOOP">> =>
    source$.pipe(
      concatMap((value) =>
        project(value).pipe(catchError(() => of(noopAction())))
      )
    );
}
```

Generic Operator 2/2

```
this.actions$.pipe(
  ofType(update),
  safeConcatMap(({ customer }) =>
    this.http
      .put<Customer[]>(this.#baseUrl, customer)
      .pipe(map(() => updateSuccess()))
  )
);
```

Error Bubbling



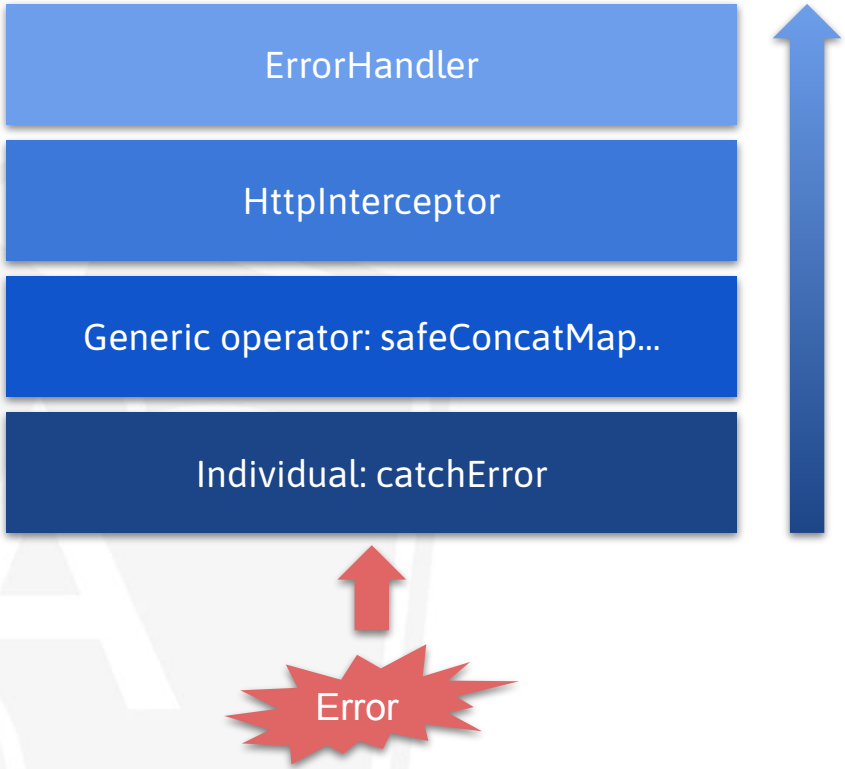
Combination with HttpInterceptor 1/2

```
export const ERROR_MESSAGE_CONTEXT = new HttpContextToken(
  () => "Sorry, something went wrong on our side."
);

export class ErrorInterceptor implements HttpInterceptor {
  // ...

  intercept(
    req: HttpRequest<unknown>,
    next: HttpHandler
  ): Observable<HttpEvent<unknown>> {
    return next.handle(req).pipe(
      catchError((err) => {
        const errorMessageContext = req.context.get(ERROR_MESSAGE_CONTEXT);
        this.uiMessage.error(errorMessageContext);
        return throwError(() => err);
      })
    );
  }
}
```

Combination with HttpInterceptor 2/2

```
this.actions$.pipe(
  ofType(update),
  safeConcatMap(({ customer }) =>
    this.http
      .put<Customer[]>(this.#baseUrl, customer, {
        context: withErrorMessageContext("Customer could not be updated"),
      })
      .pipe(
        tap(() => this.uiMessage.info("Customer has been updated")),
        map(() => load())
      )
  )
);
```

Error Bubbling



ANGULAR ARCHITECTS
INSIDE KNOWLEDGE

Global Error Handling

(because it's not always in the API communication...)

```
@Injectable()
export class ErrorHandlerService implements ErrorHandler {
  constructor(private injector: Injector) {}

  handleError(error: unknown): void {
    const messageService = this.injector.get(MessageService);
    messageService.error("We are sorry. An error happened.");
    console.error(error);
  }
}
```

Can a State be in an "Error State"?

ANGULAR ARCHITECTS
INSIDE KNOWLEDGE

Error State

- Error property for each Action questionable
  - add / addSuccess / addFailed
  - remove / removeSuccess / removeFailed
  - update / updateSuccess / updateFailed
- Focus on error property for central state element (often entities)

Error State

```
export interface CustomerState {
  customers: Customer[];
  hasError: boolean;
}

export const initialState: CustomerState = {
  customers: [],
  hasError: false
};
```

Deferred Actions

Examples

- Navigating to a different route
- Showing a message
- Showing a local loading indicator

Does this look right?

```
remove$ = createEffect(() =>
  this.actions$.pipe(
    ofType(remove),
    concatMap(({ customer }) =>
      this.http.delete<Customer[]>(`${this.                mer.id}`)
    ),
    tap(() => this.router.navigateByUrl("/customer")),
    map(() => removed())
  )
);
```

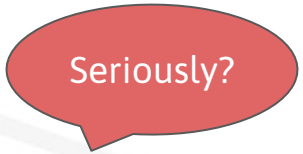
Tight Coupling

Better now?

```
@Component({
  // ...
})
export class EditCustomerComponent {
  // ...

  submit(customer: Customer) {
    this.store.dispatch(
      update({
        customer: { ...cust                omerId },
      })
    );
    this.router.navigate(["/customers"]);
  }
}
```

And this?

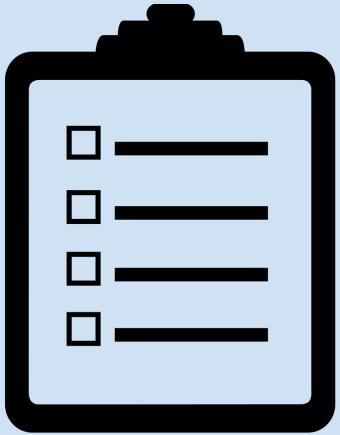
```
@Component({
  // ...
})
export class EditCustomerComponent {
  constructor(private router: Router, actions$: Actions) {
    actions$
      .pipe(ofType(removed))
      .subscribe(() => this.router.navigate(["/customers"]));
  }
}
```



Distribution of Tasks

Component

Specify Redirection

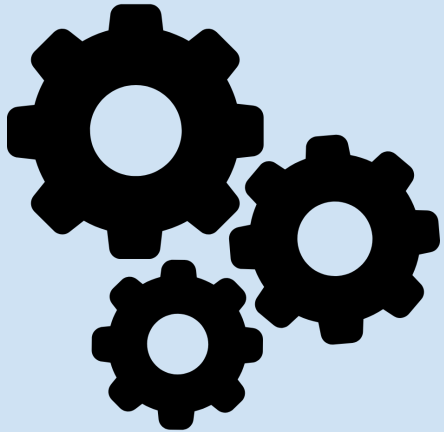


Url
(Action payload)

```
this.store.dispatch(
  load({forward: "/customer"}
)
```

State Management

Execute Redirection



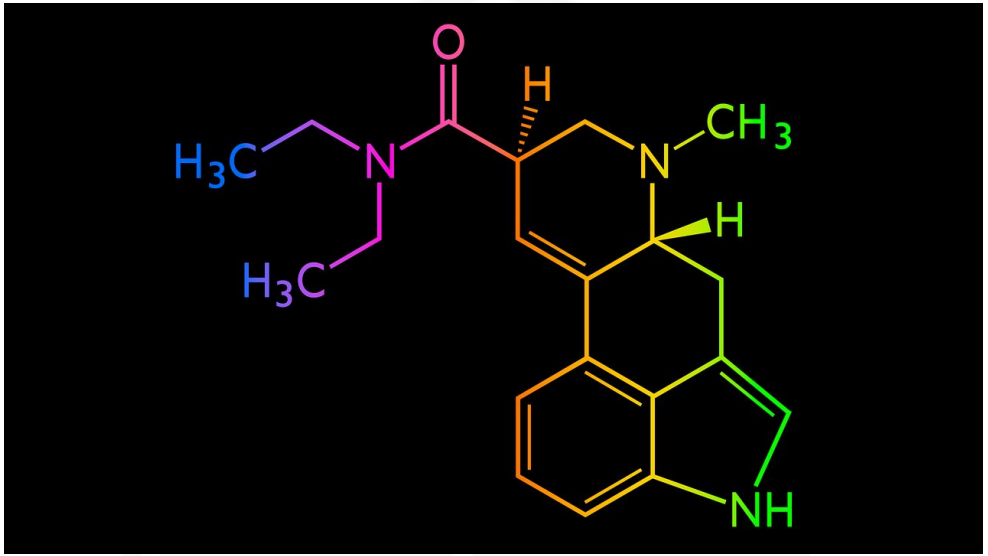
```ts
// customer.actions.ts
export const remove = createAction(
  "[Customer] Remove",
  props<{ customer: Customer; forward: string; message: string }>()
);

// customer.effects.ts
remove$ = createEffect(() =>
  this.actions$.pipe(
    ofType(remove),
    concatMap(({ customer, forward, message }) =>
      this.http.delete<Customer[]>(`${this.#baseUrl}/${customer.id}`).pipe(
        tap(() => this.router.navigateByUrl(forward)),
        tap(() => this.uiMessage.info(message))
      )
    ),
    map(() => removed())
  )
);
```

Strict Decoupling?

- Depends on type of State
  - Entity: Managing Entities
  - UI: Coordinating multiple UI components
- Consequences for
  - Selectors using multiple feature states
  - Action per Component
  - Simplicity
  - Architecture

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Dependent Feature States



Resetting States / "Inter-Feature-State" Actions

- Including Trigger via the URL
- Always via API
- In Exercise explain the usage of Routing and SubModules if possible

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Scenarios

- Passive dependency
  - Other feature state is read-only → no actions required
  - Selectors in simple use cases
  - combineLatest when Facades act as API
- Active dependency
  - Action triggers change or side-effect in state
  - Direct usage of action in effect/reducer
  - Use Observable from Facade

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Active Dependency 1/2

```
// customer.facade.ts

@Injectable({
  providedIn: "root",
})
export class CustomerFacade {
  #selectedCustomerId$: Observable<number | undefined>;

  getSelectedCustomerId() {
    return this.#selectedCustomerId$;
  }

  constructor(private store: Store) {
    this.#selectedCustomerId$ = this.store.select(
      fromCustomer.selectSelectedId
    );
  }
}
```

Active Dependency 2/2

```
// booking.effects.ts

@Injectable()
export class BookingEffects {
  constructor(private store: Store, private customerFacade: CustomerFacade) {}

  load$ = createEffect(() =>
    this.customerFacade.getSelectedCustomerId().pipe(
      filter(Boolean),
      map((customerId) => loaded({ bookings: bookings.get(customerId) || [] }))
    )
  );
}
```

Architectural Considerations

- Mirror dependencies via routing
  - Important for lazy loaded modules
  - RouterGuard acts as resetting trigger
- Eagerly initialise only data module
- Is "trigger logic" part of a component or state management?

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE