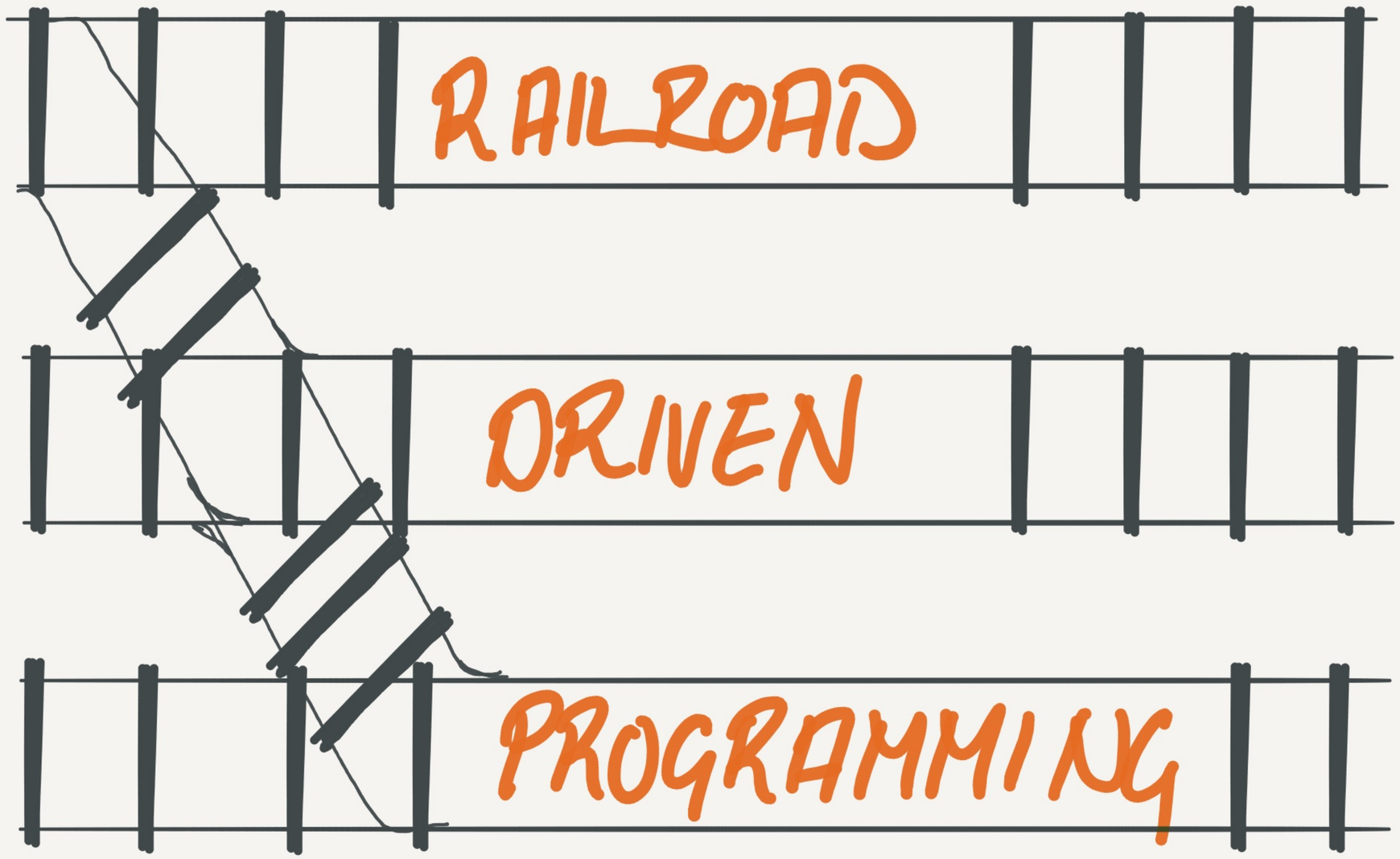
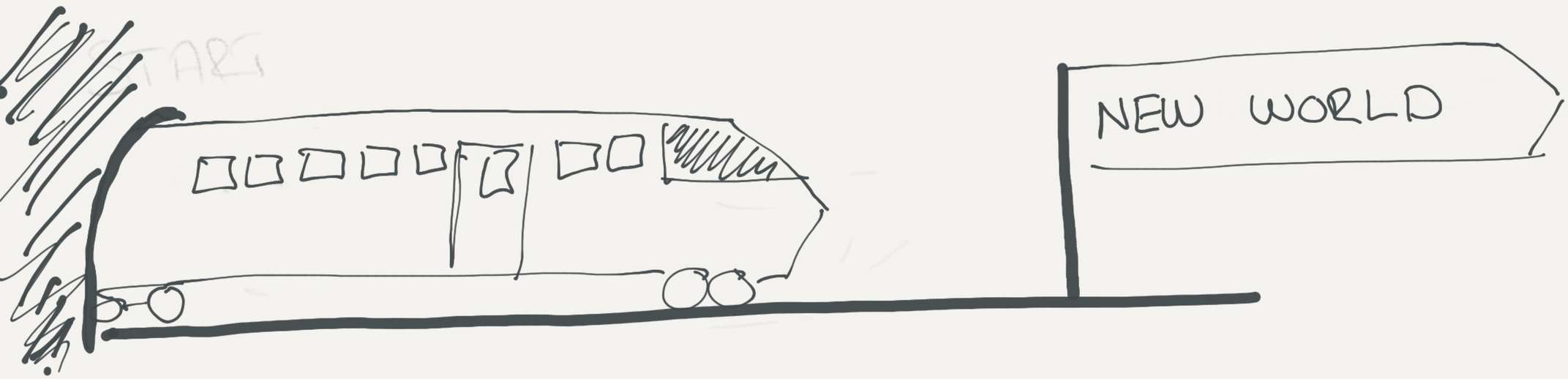


RINKKASATIAINEN

RAILROAD - DRIVEN PROGRAMMING



PRESENT

MONADS

fsharpforfunandprofit.com

HOW EITHER MONAD CHANGED MY CODING

Aki Salmi
@rinkkasatiainen
ambientia

Aki Salmi

@rinkkasatiainen

aki@rinkkasatiainen.fi

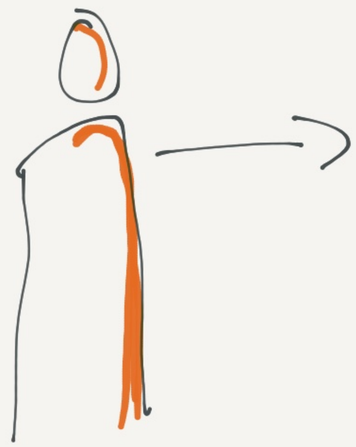
Ambientia

COME AND TALK W/ ME

- CODE
- AWS (WANT TO LEARN MORE)
- NVC
- LISTENING
- EMPATHY

EXAMPLE CODE

NODE JS
EXPRESS
TYPESCRIPT



POST /register
GET /messages/:streamId
POST /messages/:streamId

- CAN CREATE A MESSAGE THREAD
- CAN READ MESSAGE & RESPONSE
- CAN SEND NEW MSG ON RESPONDED MSG

I WILL BE SHOWING CODE

HERE IS FEW ES6 METHOD SHORTHANDS

```
// @ts-ignore
function a1() {
    return "a1";
}

// @ts-ignore
const a2 = () => "a2";
```

{
return "a2";
}

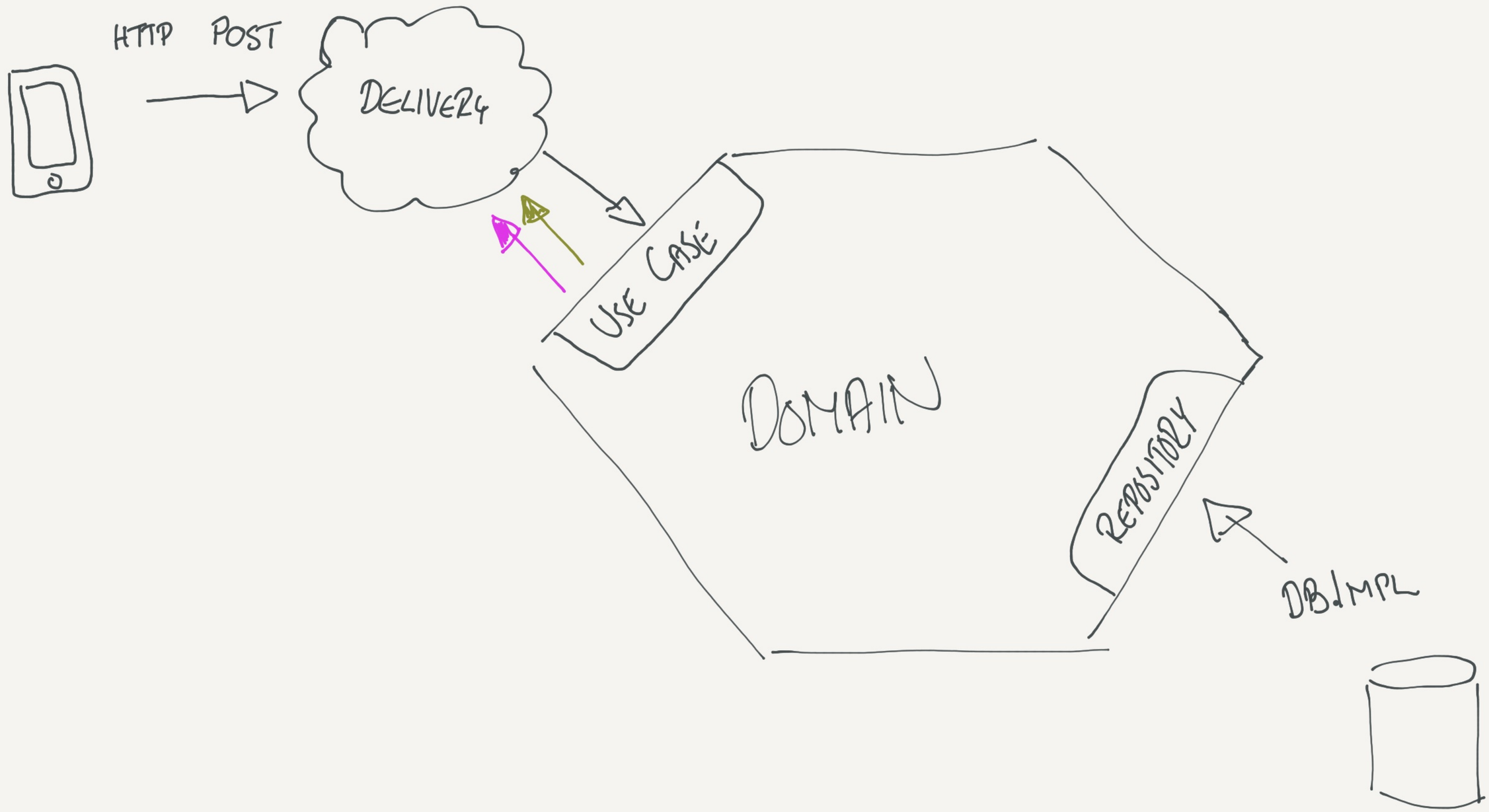
FUNCTION RETURNS A FUNCTION

```
// @ts-ignore
function b1( arg1 ) {
    return function ( arg2 ) {
        return `${arg1} ${arg2}`;
    };
}

// @ts-ignore
const b2 = arg1 => arg2 => `${arg1} ${arg2}`;
```

2-PARAM FUNC TO 2 ONE-PARAM FUNCTIONS

```
const withMultipleParams = (arg1, arg2) => `${arg1} ${arg2}`;
const withMultipleParams2 = arg1 => arg2 => `${arg1} ${arg2}`;

console.log(withMultipleParams( arg1: "foo",  arg2: "bar"))

const curriedFn = withMultipleParams2( arg1: "foo")
console.log(curriedFn( arg2: "bar"))
console.log(withMultipleParams2( arg1: "foo")("bar"))
```

PORTS - AND - ADAPTERS



Use Case: Post a message

CREATE A COMMAND FROM REQUEST → IS REQUEST DATA VALID

→ AM I AUTHORIZED

↓

FIND ENTITY → DOES ENTITY EXIST

↓

UPDATE ENTITY → DOES DOMAIN LOGIC ALLOW UPDATES NOW?

↓

RETURN RESULT TO USER

SOMETHING LIKE THIS

USE CASE HANDLER!

```
// tslint:disable-next-line:no-any
export const postMessageTheOldWay: (x: any) => (y: any) => (z: any) => (u?: any) => any =
    repository => streamId => command => userAccountDetails => {
        const user = userAccountDetails as UserAccountDetails;

        if (!isAuthenticated(user) || !isAuthorized(streamId)(user)) {
            throw new Error( message: "Describing UNAUTHORIZED");
        }
        if (!isValidMessage(command)(user)) {
            throw new Error( message: "Describing BAD REQUEST");
        }
        // all following throws exception if message is not found, or handle null here
        const message = findMessage(repository)(streamId);
        // @ts-ignore
        const modifiedMessage = postAMessage(command)(message);
        // @ts-ignore
        saveEntity(repository)(streamId)(modifiedMessage);

        return modifiedMessage;
    };
```

SOMETHING LIKE THIS

HANDLE AUTHENTICATION & AUTHORIZATION

```
// tslint:disable-next-line:no-any
export const postMessageTheOldWay: (x: any) => (y: any) => (z: any) => (u?: any) => any =
    repository => streamId => command => userAccountDetails => {
        const user = userAccountDetails as UserAccountDetails;
        if (!isAuthenticated(user) || !isAuthorized(streamId)(user)) {
            throw new Error( message: "Describing UNAUTHORIZED");
        }
        throw new Error( message: "Describing BAD REQUEST");
    }
    // all following throws exception if message is not found, or handle null here
    const message = findMessage(repository)(streamId);
    // @ts-ignore
    const modifiedMessage = postAMessage(command)(message);
    // @ts-ignore
    saveEntity(repository)(streamId)(modifiedMessage);

    return modifiedMessage;
};
```

SOMETHING LIKE THIS

VALIDATE INCOMING DATA

```
// tslint:disable-next-line:no-any
export const postMessageTheOldWay: (x: any) => (y: any) => (z: any) => (u?: any) => any =
    repository => streamId => command => userAccountDetails => {
        const user = userAccountDetails as UserAccountDetails;

        if (!isAuthenticated(user) || !isAuthorized(streamId)(user)) {
            throw new Error( message: "Describing UNAUTHORIZED");
        }
```

```
if (!isValidMessage(command)(user)) {
    throw new Error( message: "Describing BAD REQUEST");
}
```

```
        message = findMessage(repository)(streamId);
        // @ts-ignore
        const modifiedMessage = postAMessage(command)(message);
        // @ts-ignore
        saveEntity(repository)(streamId)(modifiedMessage);

        return modifiedMessage;
    };
```

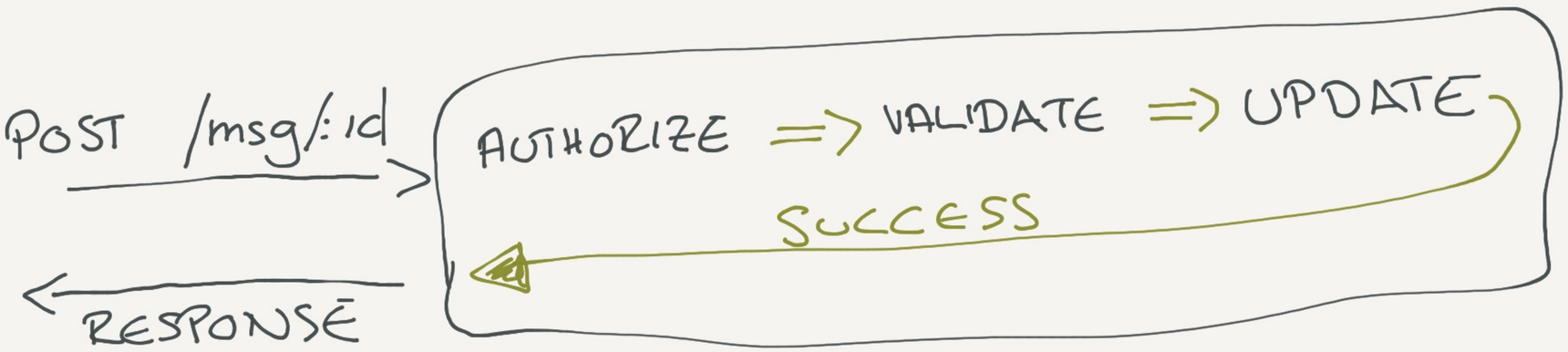
SOMETHING LIKE THIS

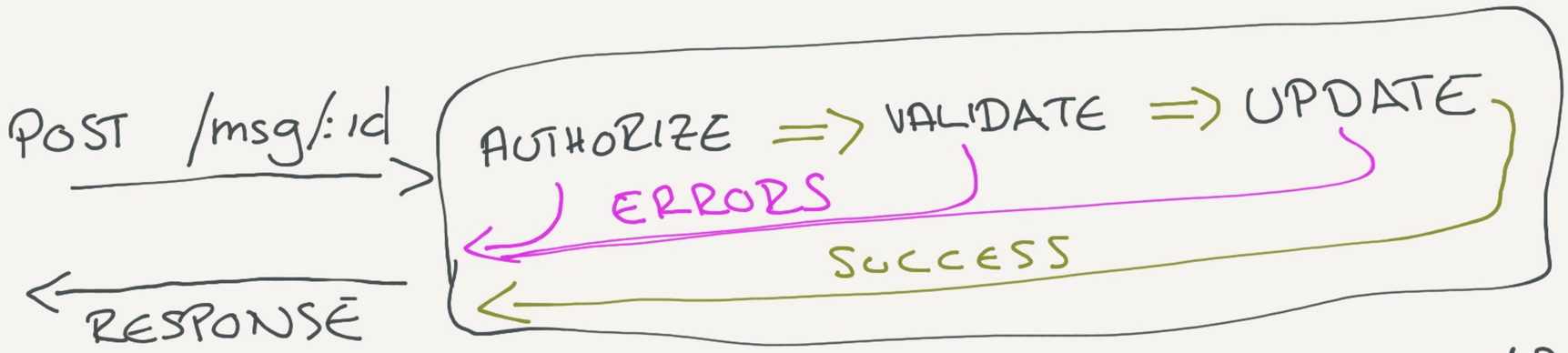
DO A BIT OF DOMAIN LOGIC

```
// tslint:disable-next-line:no-any
export const postMessageTheOldWay: (x: any) => (y: any) => (z: any) => (u?: any) => any =
    repository => streamId => command => userAccountDetails => {
        const user = userAccountDetails as UserAccountDetails;

        if (!isAuthenticated(user) || !isAuthorized(streamId)(user)) {
            throw new Error( message: "Describing UNAUTHORIZED");
        }
                                    (command)(user)) {
```

```
// all following throws exception if message is not found, or handle null here
const message = findMessage(repository)(streamId);
// @ts-ignore
const modifiedMessage = postAMessage(command)(message);
// @ts-ignore
saveEntity(repository)(streamId)(modifiedMessage);

return modifiedMessage;
```

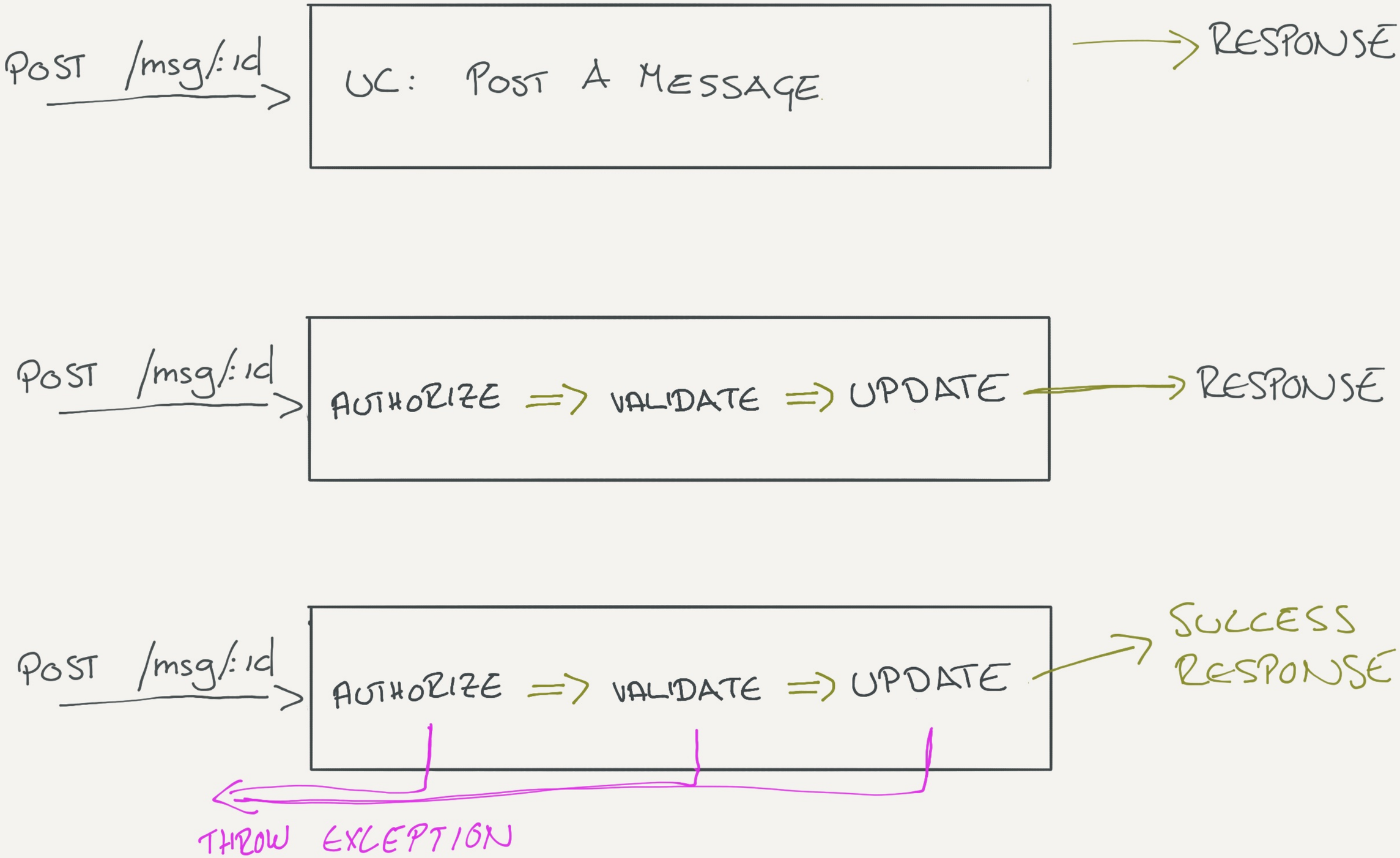
DATA FLOW IN DOMAIN





ERRORS HANDLED BY MIDDLEWARE / ROUTES

DATA FLOW IN PRESENT



ERRORS AS 1ST CLASS CITIZENS

ERRORS CONTAIN IMPORTANT INFORMATION

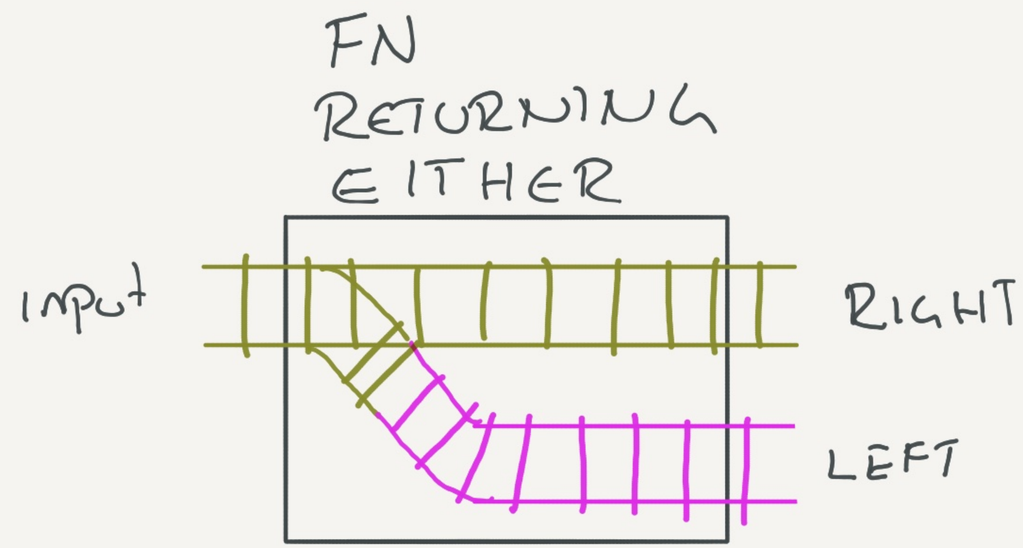
- AUTHENTICATION / AUTHORIZATION
- INPUT VALIDATION
- DOMAIN SPECIFIC VALIDATION
  - CANNOT POST A NEW MESSAGE IF LATEST IS NOT ANSWERED TO

INTRODUCE EITHER!

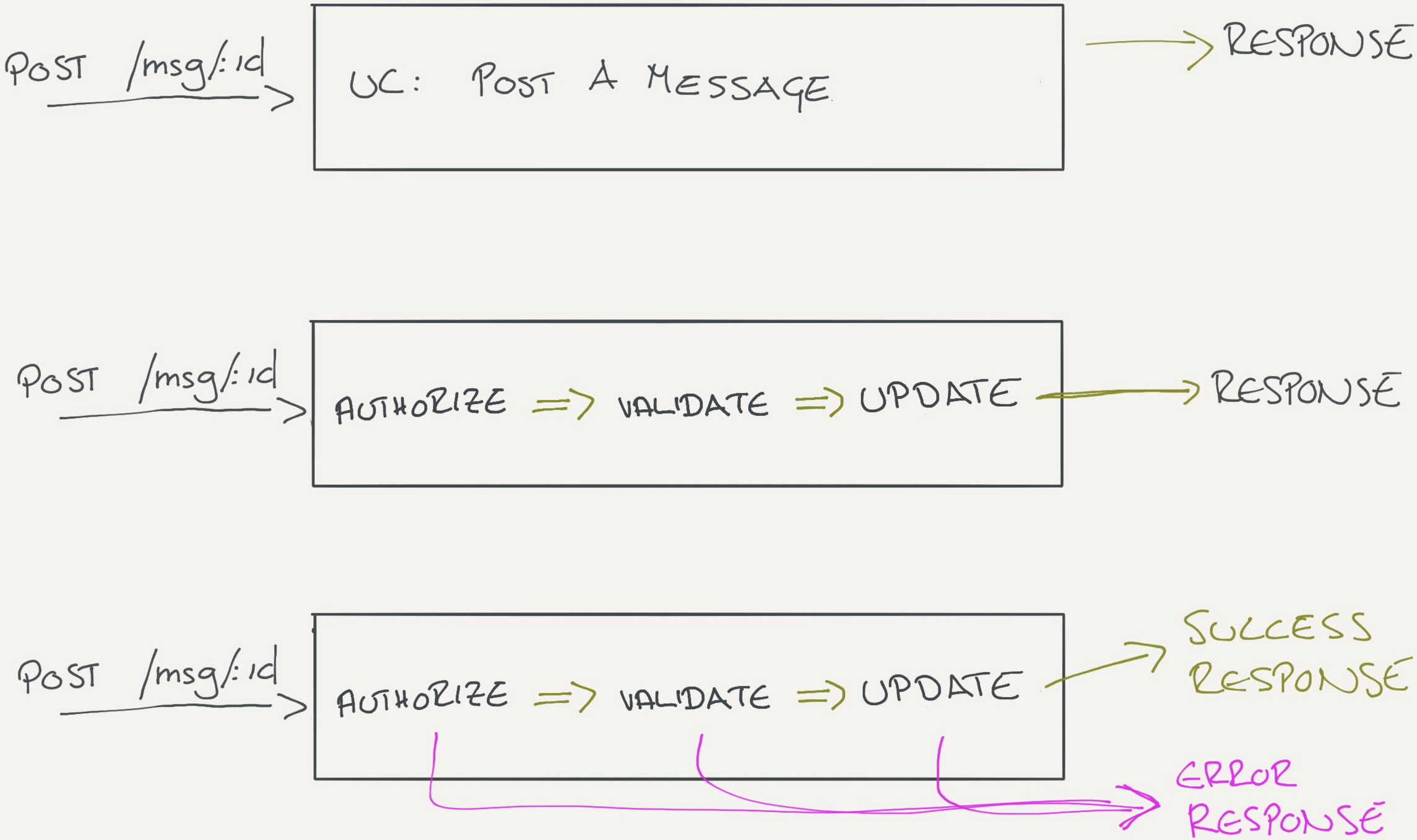
RIGHT?

Either <L, R>

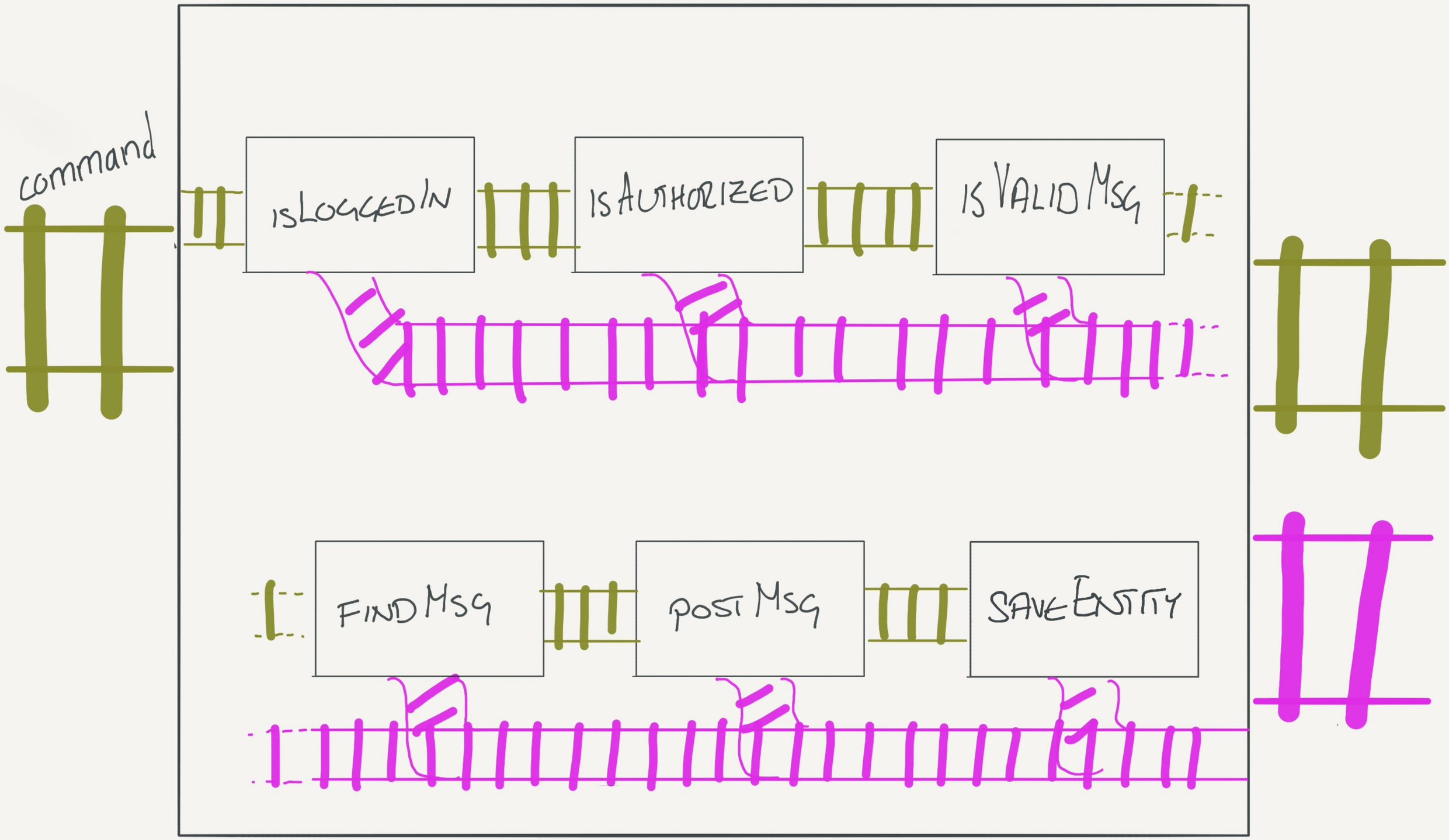
- LEFT IS FOR ERRORS
- RIGHT IS FOR SUCCESS



DATA FLOW

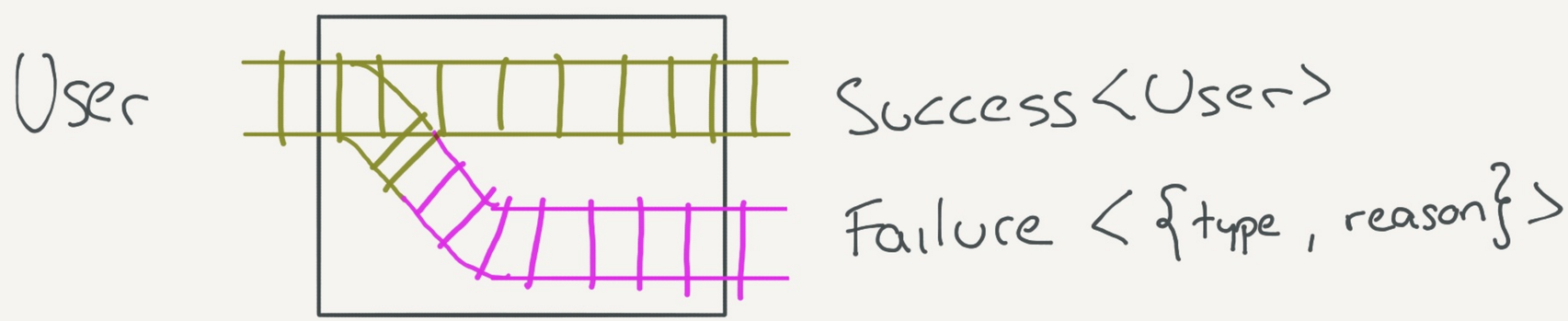

POST /msg/:id
UC: POST A MESSAGE
RESPONSE
POST /msg/:id
AUTHORIZE => VALIDATE => UPDATE
RESPONSE
POST /msg/:id
AUTHORIZE => VALIDATE => UPDATE
SUCCESS RESPONSE
ERROR RESPONSE

Use Case: Post a Message

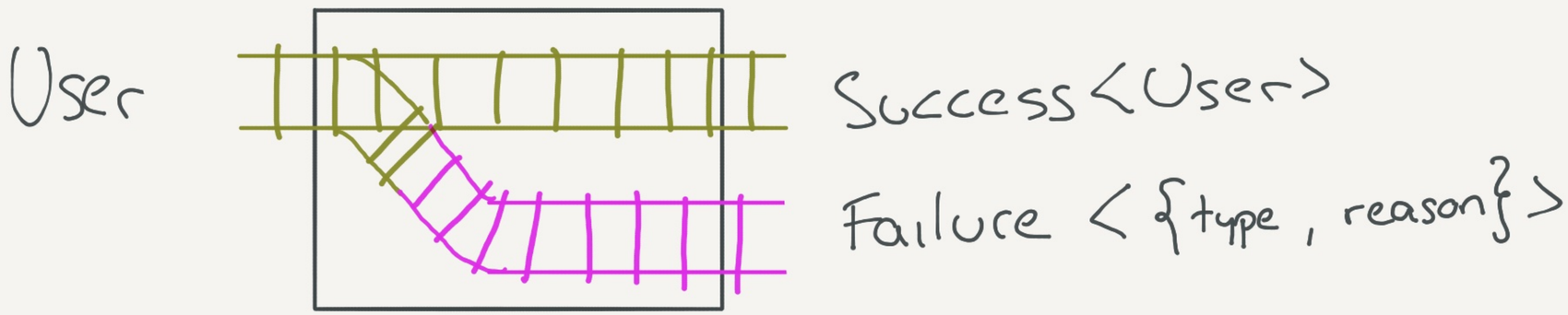


isLoggedIn

User => a function => Either<
Failure, Success<User>
>



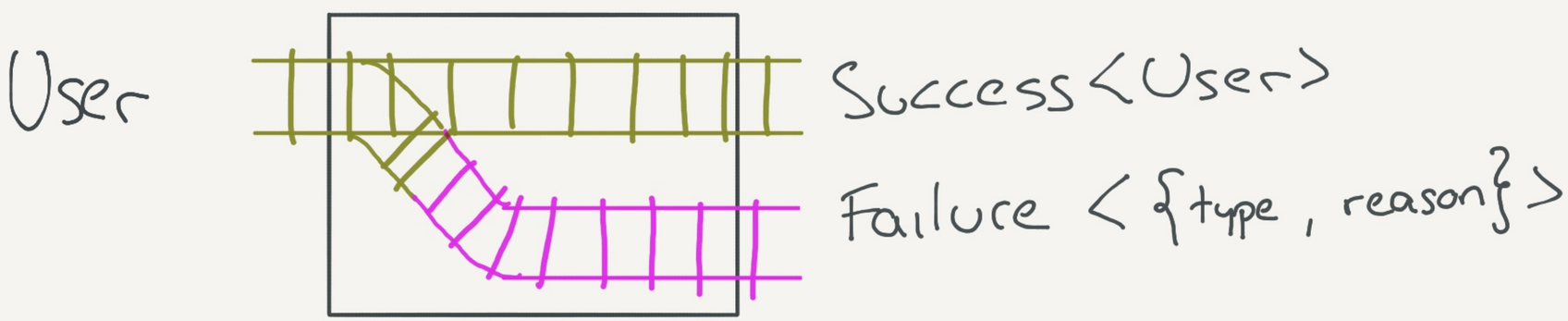
isLoggedIn



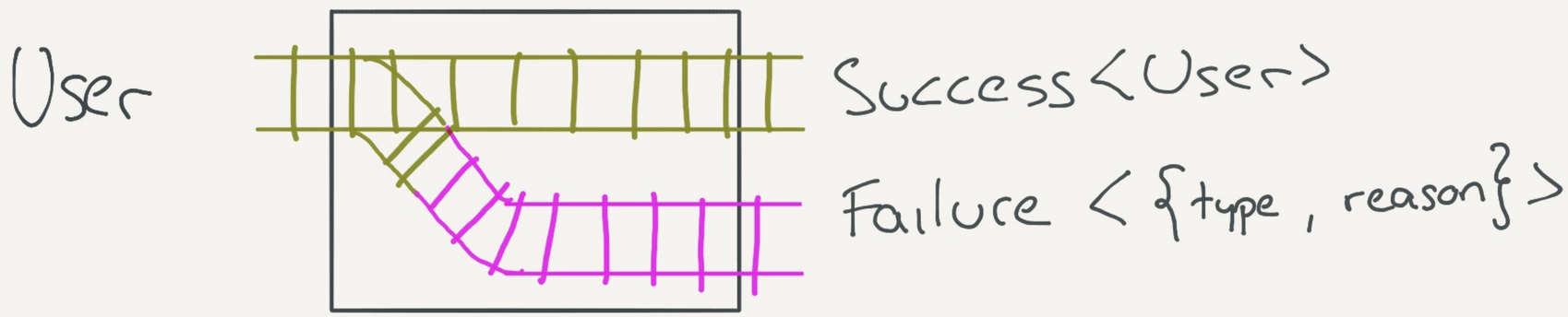

```
export const isAuthenticated: ChainableFunction<UserAccountDetails> =
    userOrUndefined => toEither(toError( x: "401")("forbidden"))(userOrUndefined);
```

IsAuthorized(streamId)

User => a function => Either<
Failure, Success<Users>
>

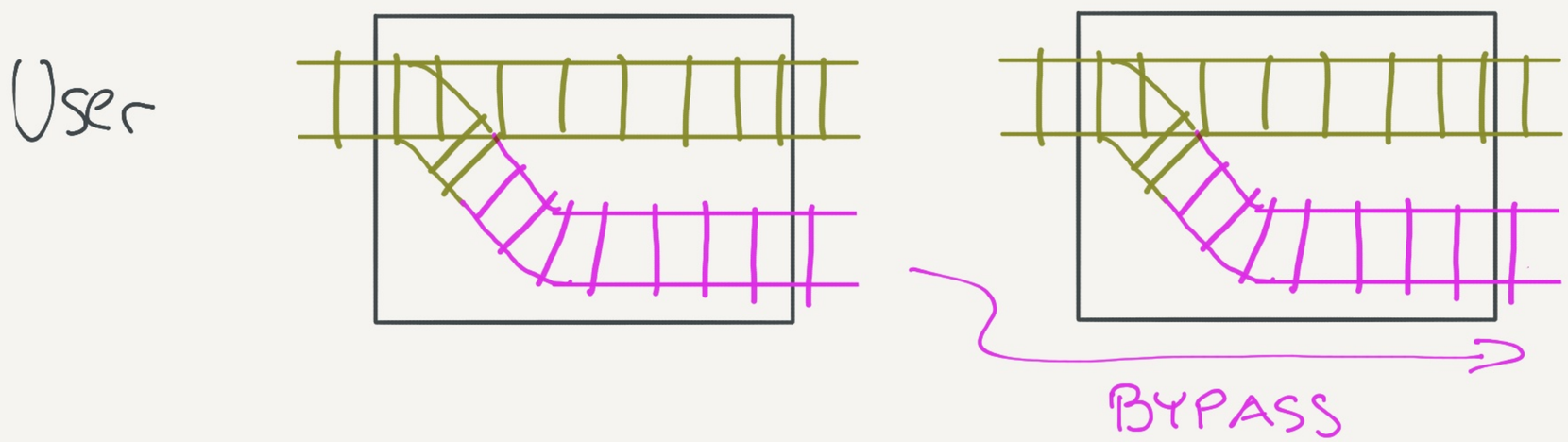


IsAuthorized(streamId)



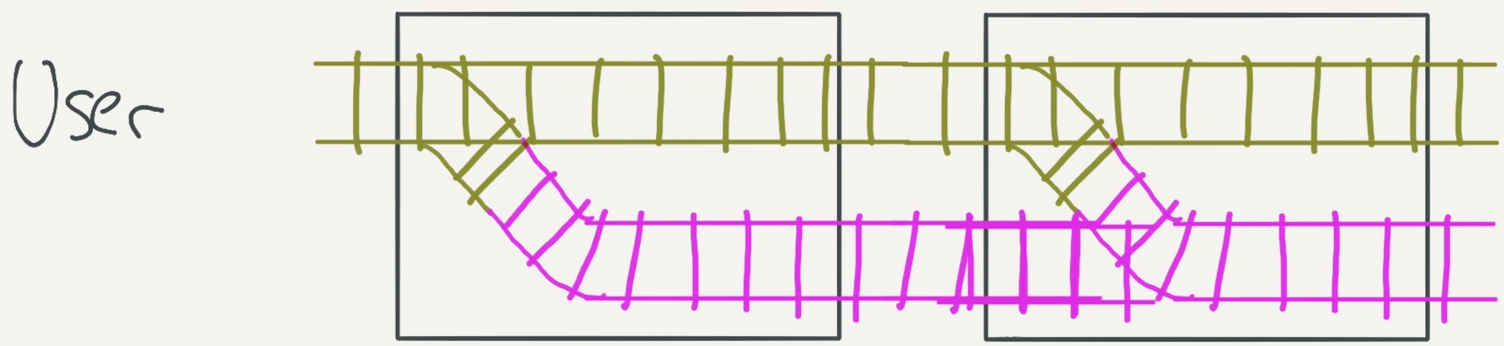

```
export const isAuthorized: (page: string) => ChainableFunction<UserAccountDetails> =
    page => user => {
        if (user && page === user.userId) {
            return S.Right(user);
        }
        return S.Left(toError( x: "403")("unauthorized"));
    };
```

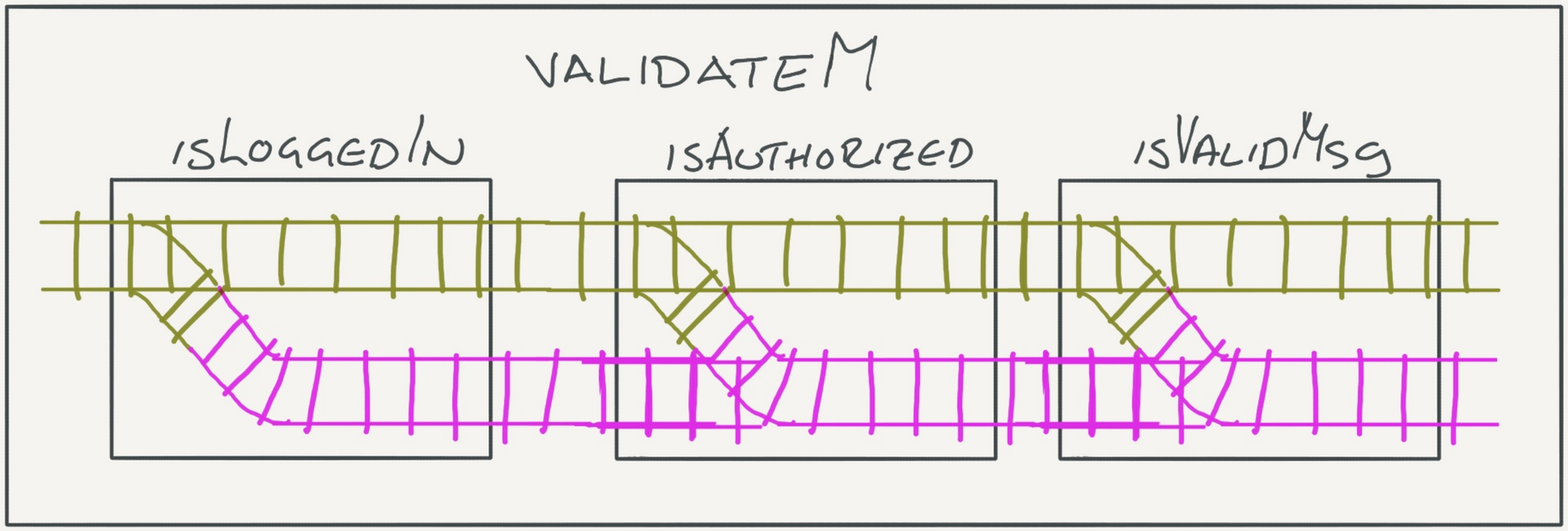
ISLOGGEDIN >> ISAUTHORIZED(STREAMID)



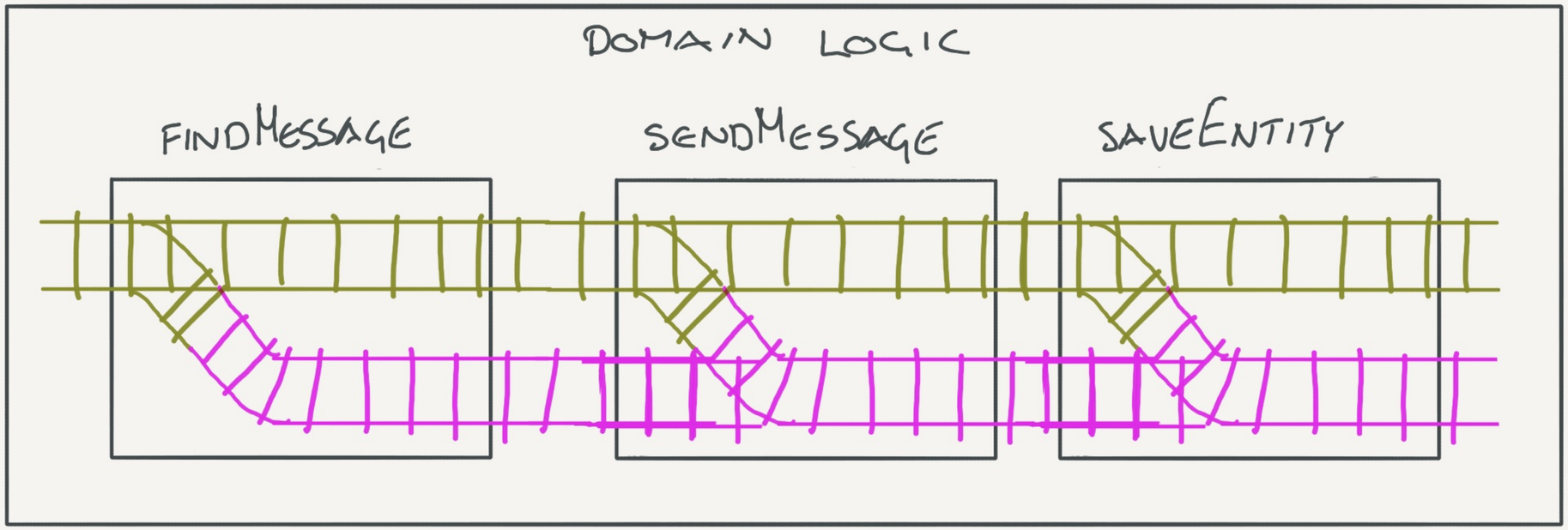
ISLOGGEDIN >> ISAUTHORIZED(STREAMID)

ON 2-TRACK-MODEL




VALIDATEM
ISLOGGEDIN
ISAUTHORIZED
ISVALIDMSG

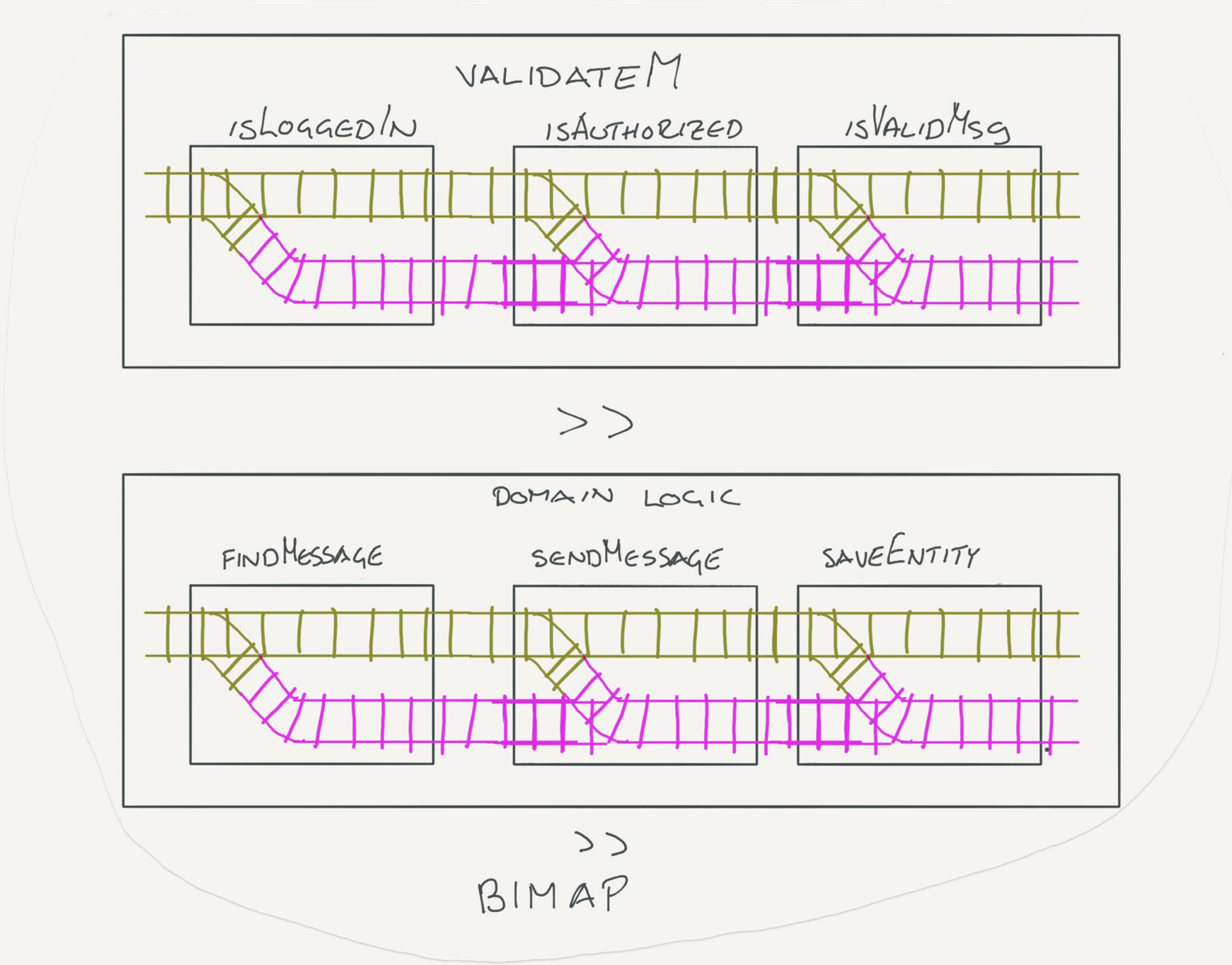
>>


DOMAIN LOGIC
FINDMESSAGE
SENDMESSAGE
SAVEENTITY

>>

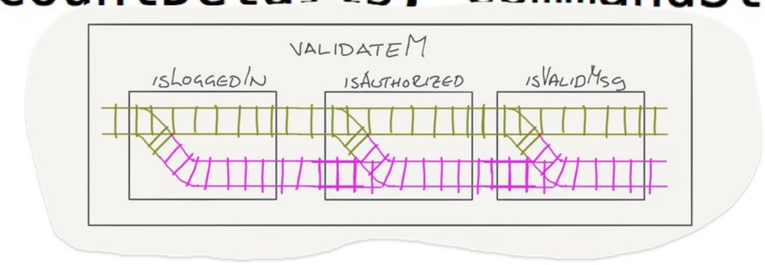
BIMAP

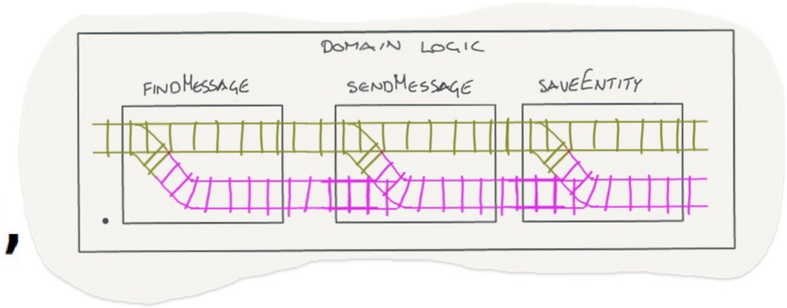
postAMessage, in code



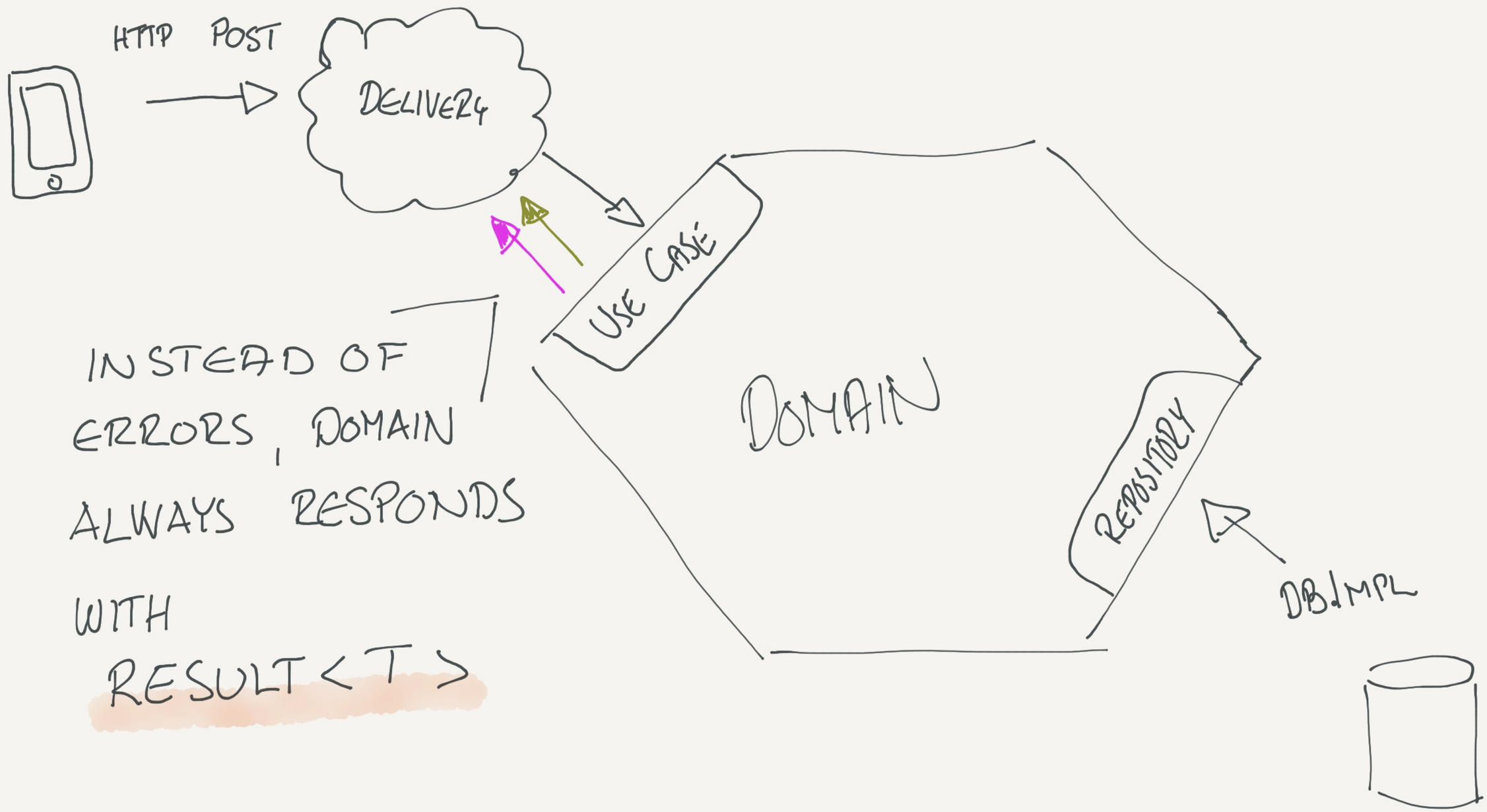
IN CODE

```
export const postMessage: (x: MessageRepository) => PostMessageCommandHand
    repository => streamId => command => userAccountDetails => {
        const isValid = validateM<UserAccountDetails, CommandStatus>( funcs
            isAuthenticated,
            isAuthorized(streamId),
            isValidMessage(command),
        ])(toMaybe(userAccountDetails));

        return isValid.andThen([
                findMessage(repository),
                postAMessage(command),
                saveEntity(repository)(streamId),
            ])(streamId)
            .bimap(
                convertError( x: {type: "501"}),
                convertToResponse,
            );
    };
```





PORTS - AND - ADAPTERS

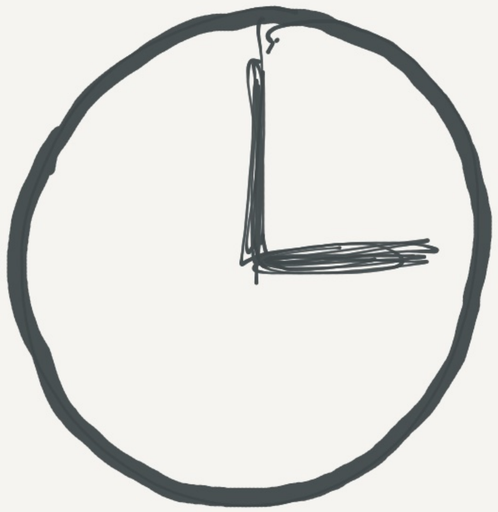


```typescript
// tslint:disable-next-line:no-any
export const postMessageTheOldWay: (x: any) => (y: any) => (z: any) => (u?: any) => any =
    repository => streamId => command => userAccountDetails => {
        const user = userAccountDetails as UserAccountDetails;

        if (!isAuthenticated(user) || !isAuthorized(streamId)(user)) {
            throw new Error( message: "Describing UNAUTHORIZED");
        }
        if (!isValidMessage(command)(user)) {
            throw new Error( message: "Describing BAD REQUEST");
        }
        // all following throws exception if message is not found, or handle null here
        const message = findMessage(repository)(streamId);
        // @ts-ignore
        const modifiedMessage = postAMessage(command)(message);
        // @ts-ignore
        saveEntity(repository)(streamId)(modifiedMessage);

        return modifiedMessage;
    };
```

THE CHANGE IS SOMEWHAT AS:

```typescript
export const postMessage: (x: MessageRepository) => PostMessageCommandHandler =
    repository => streamId => command => userAccountDetails => {
        const isValid = validateM<UserAccountDetails, CommandStatus>( funcs: [
            isAuthenticated,
            isAuthorized(streamId),
            isValidMessage(command),
        ])(toMaybe(userAccountDetails));

        return isValid.andThen([
                findMessage(repository),
                postAMessage(command),
                saveEntity(repository)(streamId),
            ])(streamId)
            .bimap(
                convertError( x: {type: "501"}),
                convertToResponse,
            );
    };
```

EVENTUALLY, THINGS HAPPEN!



HOW IS FUTURES HANDLED DIFFERENTLY?

HOW IS FUTURES HANDLED DIFFERENTLY?

```
export const postMessage: (x: MessageRepository) => PostMessageCommandHand
    repository => streamId => command => userAccountDetails => {
        const isValid = validateM<UserAccountDetails, CommandStatus>( funcs
            isAuthenticated,
            isAuthorized(streamId),
            isValidMessage(command),
        ])(toMaybe(userAccountDetails));

        return isValid.andThen([
                findMessage(repository),
                postAMessage(command),
                saveEntity(repository)(streamId),
            ])(streamId)
            .bimap(
                convertError( x: {type: "501"}),
                convertToResponse,
            );
    };
```

NOW

IN FUTURE

HAPPENS ANYWAY!

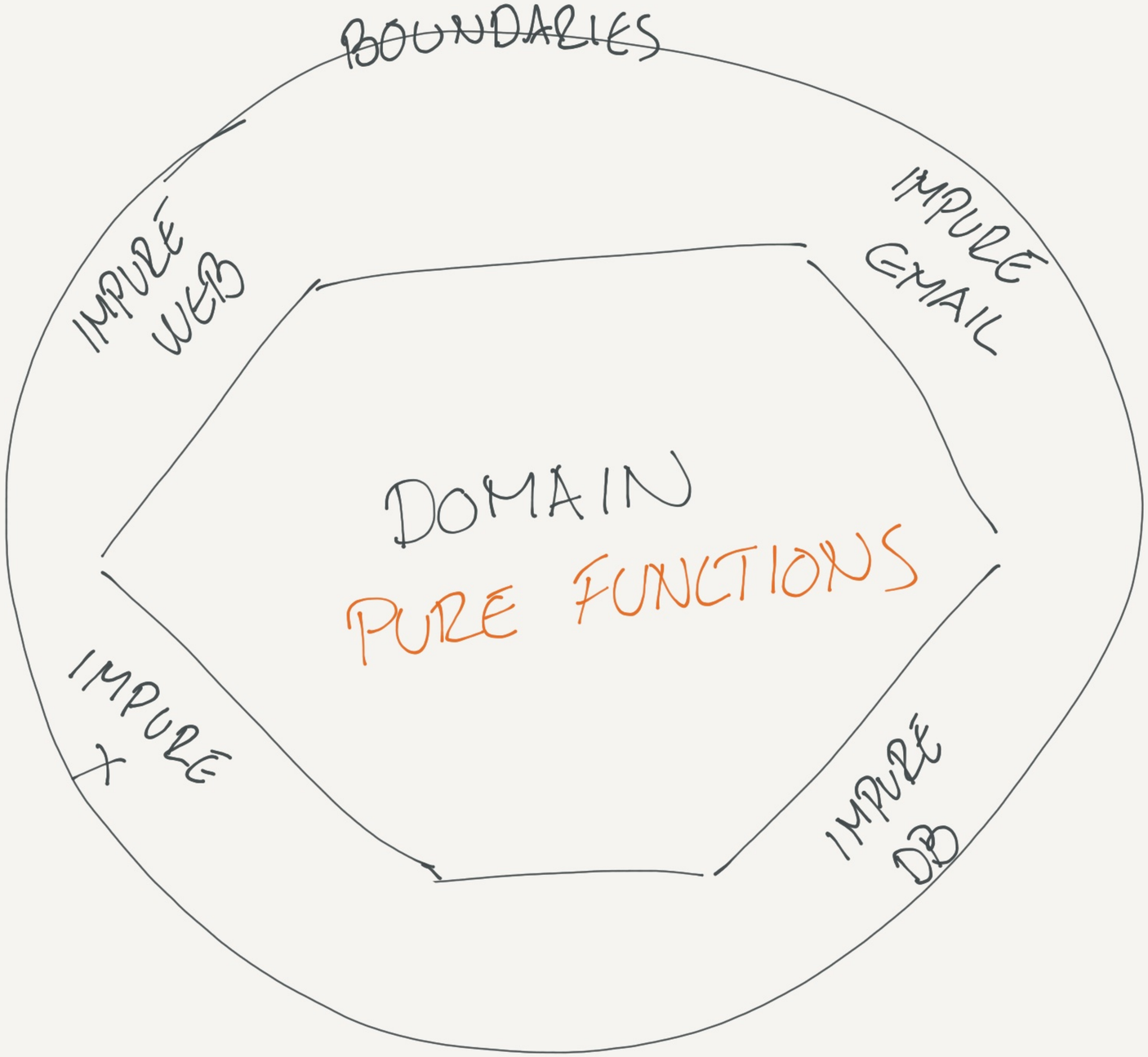
DOMAIN LOOKS COOL

WHAT ABOUT ROUTES?

```
export const postMessageRouteMethod: PostMessageRouteMethod =
    commandHandler => pageUuid => (req: Request, res: Response) =>  {
        const user = req.user
        const message = createMessageFromRequest(req)(pageUuid);

        const onError = handleErrors( funcs: [
                logentry,
                reportErrors,
                redirectTo( url: `/message/${pageUuid}`),
            ]);

        const onSuccess = handleSuccess<CommandStatus>( funcs: [
                logentry,
                reportCommandSuccess,
                redirectOnSuccess( fun: () => `/message/${pageUuid}`),
            ]);

        const result: FutureResult<CommandStatus> = commandHandler(pageUuid)(message)(user);

        const handle = handleFuture<CommandStatus>( params: {req, res});
        handle(onError)(onSuccess)(result);
    };
```

THAT'S IT!

- COMPOSE DOMAIN LOGIC FROM SMALL (PURE) FUNCTIONS
- RETURN EITHER<FAILURE, SUCCESS<T>>
- DOMAIN LOGIC IS EASY TO TEST
- ROUTES ARE EASY TO TEST
- WHERE ARE ALL SIDE-EFFECTS AND IMPURE FUNCTIONS?


BOUNDARIES
IMPURE WEB
IMPURE GMAIL
DOMAIN
PURE FUNCTIONS
IMPURE X
IMPURE DB