





**ERRURS**

F

R

-

T

S

MAGNÉT-QUALITÄT S-THOMAS & SÖHN

```

import { pipe } from 'fp-ts/function';
import * as E from 'fp-ts/Either';

type State = E.Either<Error, Idle | Moving>
const initialState: State = pipe(idle(), E.right);

const reducer = (state: State, command: Command) => {
  // ⚠️ `chain` is `flatMap` or `bind` name in fp-ts;
  // be carefull bind is js Function.prototype.bind ... naming are hard
  return E.chain(
    // 😞 TS inference is bad, you will often need to help the typer
    (state: Idle | Moving): E.Either<Error, Idle | Moving> => {
      switch (state.type) {
        case "idle": {
          switch (command.type) {
            case "face": return pipe(idle(), E.right); // idle () |> E.right
            case "start": return pipe(moving(), E.right);
            case "stop": return pipe(new Error("Illegal Action from Idle"),
                                     , E.left); // 😎
          }
        }
        case "moving": {
          switch (command.type) {
            case "stop": return pipe(idle(), E.right);
            default: return pipe(new Error("Illegal Action from Moving"),
                                 E.left); // 😎
          }
        }
      }
    }
  )(state)
}

```

# Design simpliste pour exemple

## Que faire quand le state est en erreur ?

*Action Init?*

*Reinit auto ?*

*Conserver le state avant erreur ?*

**Dans un state plus complexe,  
on ne veut pas forcément tout  
dans un Either : c'est ok!**

# ERREURS

## FP-TS

Design simpliste pour exemple

Que faire quand le state est en erreur ?

*Action Init?*

*Reinit auto ?*

*Conserver le state avant erreur ?*

Dans un state plus complexe,  
on ne veut pas forcément tout  
dans un Either : c'est ok!

```
import { pipe } from 'fp-ts/function';
import * as E from 'fp-ts/Either';

type State = E.Either<Error, Idle | Moving>
const initialState: State = pipe(idle(), E.right);

const reducer = (state: State, command: Command) => {
  // ⚠️ `chain` is `flatMap` or `bind` name in fp-ts;
  // be carefull bind is js Function.prototype.bind ... naming are hard
  return E.chain(
    // 😞 TS inference is bad, you will often need to help the typer
    (state: Idle | Moving): E.Either<Error, Idle | Moving> => {
      switch (state.type) {
        case "idle": {
          switch (command.type) {
            case "face": return pipe(idle(), E.right); // idle () |> E.right
            case "start": return pipe(moving(), E.right);
            case "stop": return pipe(new Error("Illegal Action from Idle"),
                                     E.left); // 😎
          }
        }
        case "moving": {
          switch (command.type) {
            case "stop": return pipe(idle(), E.right);
            default: return pipe(new Error("Illegal Action from Moving"),
                                  E.left); // 😎
          }
        }
      }
    }
  )(state)
}
```



# GESTION D'ÉTATS

## ELM / REDUX

**Redux** est la librairie de « state management » la plus connue de l'écosystème **React**. Elle est largement inspirée de la Elm Architecture.

On peut donc également faire le parallèle avec celle-ci :

- Un **modèle** est appelé **STORE** dans Redux.  
Le **STORE** comprends deux parties :
  - Un **STATE** qui stocke les données de l'application
  - Un **REDUCER** qui est l'équivalent de notre fonction **update**
- Un **message** est une **ACTION**
- La fonction pour **envoyer un message** (sendMessage) est appelée **DISPATCH**
- La **vue** est nommée de la même façon

