





**STATE MARKET BACK**

ET UNNEFRONN'DURDANTE=REDUCER

MAGNET 2-DUALS - THOUGHTS

```
const reducer: (state: State, action: Action) => State =
  (state: State, action: Action) => {
    {
      switch (state.type) {
        case "idle": {
          switch (action.type) {
            case "face": return idle();
            case "start": return moving();
            default: throw new Error("Impossible");// 🤮
          }
        }
        case "moving": {
          switch (action.type) {
            case "stop": return idle();
            default: throw new Error("Impossible");// 🤮
          }
        }
      }
    }
  }
}
```









```
import { Reducer } from 'redux';

const reducer: Reducer<State, Action> = (state: State | undefined,
action: Action) => {
  if (!state) return initialState //mandatory by Redux
  switch (state.type) {
    case "idle": {
      // ⚠️ Redux actions MUST be tagged on a `type` attribute
      switch (action.type) {
        case "face": return idle();
        case "start": return moving();
        case "stop": throw new Error("Impossible");// 🤢
      }
    }
    case "moving": {
      switch (action.type) {
        case "stop": return idle();
        default: throw new Error("Impossible");// 🤢
      }
    }
  }
}
```

# STATE MACHINE STRIKE BACK

ET UNE FONCTION D'UPDATE = REDUCER



```
import { Reducer } from 'redux';

const reducer: Reducer<State, Action> = (state: State | undefined,
action: Action) => {
  if (!state) return initialState //mandatory by Redux
  switch (state.type) {
    case "idle": {
      // ⚠ Redux actions MUST be tagged on a `type` attribute
      switch (action.type) {
        case "face": return idle();
        case "start": return moving();
        case "stop": throw new Error("Impossible");// 🤮
      }
    }
    case "moving": {
      switch (action.type) {
        case "stop": return idle();
        default: throw new Error("Impossible");// 🤮
      }
    }
  }
}
```

# ERREURS

## FP-TS

```
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);
// ⚠ Redux is js first, so state parameter must be a subtype of `any | undefined`
const reducer: Reducer<State, Action> = (state: State | undefined, action: Action) => {
  if (!state) return initialState //mandatory by Redux

  // ⚠ `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": {
          // ⚠ Redux action MUST be tagged on a `type` attribute
          switch (action.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 (action.type) {
            case "stop": return pipe(idle(), E.right);
            default: return pipe(new Error("Illegal Action from Moving"),
                                  E.left); // 😎
          }
        }
      }
    }
  )(state)
}
```