

# QUALITÉ DU SI

Sommaire

[Télécharger en version PDF](#)

Intro

PBT

Quête des machines états

Gestion d'erreurs & qualité du code

Frontends dataflow

Cloud & microservices

Communication Inter-process

Decentralized web

Final Boss

The background image shows a modern, multi-story library. The architecture features light blue walls and white railings. Stairs connect different levels, and bookshelves filled with books are arranged throughout the space. A person is walking down a staircase on the left, and another person is sitting on a blue sofa in the foreground on the right. The overall atmosphere is bright and spacious.

**QUALITÉ D'USI**

**IMAGE - INTRODUCTION**

# DISCLAIMER 1

## VOTRE DIPLÔME EST GLOBAL, LES UE NE SONT PAS DES SILOS

Ce cours a pour pré-requis vos cours de L3 :

- Compréhension typage statique / dynamique
- Savoir lire une stack d'erreur
- Savoir lire une doc d'API

Ce cours a pour pré-requis vos cours de M1/M2 :

- SGBDR et NoSQL (NDD)
- Serialization / Deserialization (ALOM, ARI)
- Single Page Application (ARI)
- Programmation async monothreadée aka fiber aka green thread aka light thread aka event loop aka coroutine (ARI)
- CORS et CSP (CAR, ARI, ALOM, SSI)
- API REST/JSON (ALOM, ARI)
- Management de projet (MP)
- Anglais

# **DISCLAIMER 2**

**ON ATTEND DE DIPLÔMÉS M2 D'APPORTER DE NOUVELLES CONNAISSANCES DANS L'ENTREPRISE**

**Ce cours va vous dérouter par rapport à ce que vous avez vu dans vos cours de programmation**

**Ce n'est pas contradictoire**

**Ce cours vise à vous ouvrir à des connaissances de programmation moderne**

**Ce cours vous aidera à structurer vos futurs apprentissages**

**Ce cours doit vous amener à raisonner au niveau d'un SI, mais on va repartir des bases depuis le logiciel**

# **DISCLAIMER 3**

**ÇA VA ÊTRE DENSE MAIS GUIDÉ**

**A condition que vous travaillez au fil de l'eau, vous n'aurez pas l'occasion de rattraper du retard pris**

**Cours en FR pour assurer le bonne compréhensions**

**TP en EN pour donner du vocabulaire et faciliter les recherches**

**7 Cours, incluant 4 TP, puis un projet de Système d'information**

**Vous avez accès à la première source d'information du monde : INTERNET**

- > **Vous ne trouverez pas la réponse copier-coller !**
- > **Mais vous avez accès au code source, docs d'API et communautés open source**

**Je réponds aux sollicitations entre 2 cours si vous posez des questions !**

# INTRO

## POURQUOI LA QUALITÉ EST UN ENJEU

-  Le SI est un actif valorisable de l'entreprise
-  Le SI de qualité procure un avantage compétitif
-  Le SI de qualité améliore la satisfaction client
-  Le SI de qualité améliore la satisfaction au travail
-  Le SI doit répondre aux besoins fonctionnels et non fonctionnels

# INTRO

## LES DIMENSIONS DE LA QUALITÉ

-  **Infrastructure** : matériel, réseau, OS, ... (*du baremetal au cloud*)
-  **Logiciel** : applications construites et maintenues
-  **Données** : données du SI (SQL / NoSQL)  **NDD**
-  **Information** : communications inter-applicative
-  **Administrative** : qualité de la fonction SI, incluant les processus d'élaboration du budget et d'élaboration du planning 
-  **Service** : valeur du service rendu « perçue » par le client 
-  **RH** : organisation des équipes SI  **Management de projets**

# INTRO

## La qualité optimale

**Attentes** : besoins exprimés par le client final (utilisateur)

**Spécification** : traduction du besoin utilisateur (MOA)

**Réalisation** : mise en oeuvre du service (MOE)



# LE TRILÈME

## L'impact de la gestion de projet

Les approches orientées « cycle en V » mettent l'accent sur l'axe spécification, dont vont découler les 2 autres

Les approches agiles mettent l'accent à l'origine sur la attentes↔réalisation (XP)

... mais ont dévié dans leurs versions récentes sur l'axe attentes↔spécification (scrum, safe)

... ce qui fit émerger le mouvement software craftsmanship sur l'axe réalisation↔attentes

Les approches R&D mettent souvent l'accent sur l'axe réalisation seul (innovations de rupture)



# INTRO

## ASSURANCE « QUALITÉ »

**La qualité est souvent présentée par une approche systémique orientée processus : l'assurance qualité**

**Vous manipulez peut être ces systèmes en entreprise : ISO, CMMi, ITIL, Safe, ...**

**Ces systèmes sont une partie de la dimension administrative, insuffisante prise seule**

**Dans un contexte de numérisation accélérée et totale de l'économie, les dimension Architecture et Management doivent être menées de concert**

**La dimension architecture est critique pour obtenir un avantage stratégique**

# CALENDRIER 2023

## PRÉVISIONNEL

3/01: Intro & Property based Testing - Troll of Fame Kata (TypeScript)

10/01: Quête des machines à états - Tennis Kata (TypeScript)

17/01 || 24/01 : Frontend dataflow - Front Kata (React)

24/01 || 17/01: Intro Blockchain & Smart contracts - Contract Kata (JsLigo)

31/01: Gestion des Erreurs & Présentation du projet

7/02 : Pas cours

21/02: Cloud et microservice

28/02: Communication Inter-process

# **QUALITÉ DU SI**

---

COURS 1 - PBT

# THE QUEST

PBT

Quand on reprend un projet, qu'on accueille un nouveau développeur, qu'on revient sur un code d'il y a 6 mois, qu'on effectue une migration technique, ... : comment avoir une documentation à jour.

Les écrits ne sont JAMAIS à jours

Comment documenter les règles de gestion d'un projet?



# CODE IS LAW

Code as specification ou spécification détaillée?

Moins bonne UX / Meilleure durabilité

1. Vérifier les invariants : preuves > types > tests
2. Faciliter la compréhension du code : idiomes, compétences
3. Uniformiser les pratiques : design patterns, guidelines
4. Améliorer la communication tech/business : Ubiquitous language, spécifications, UML
5. Améliorer l'expérience utilisateur : UX design, user doc, vidéos

Impact User Experience

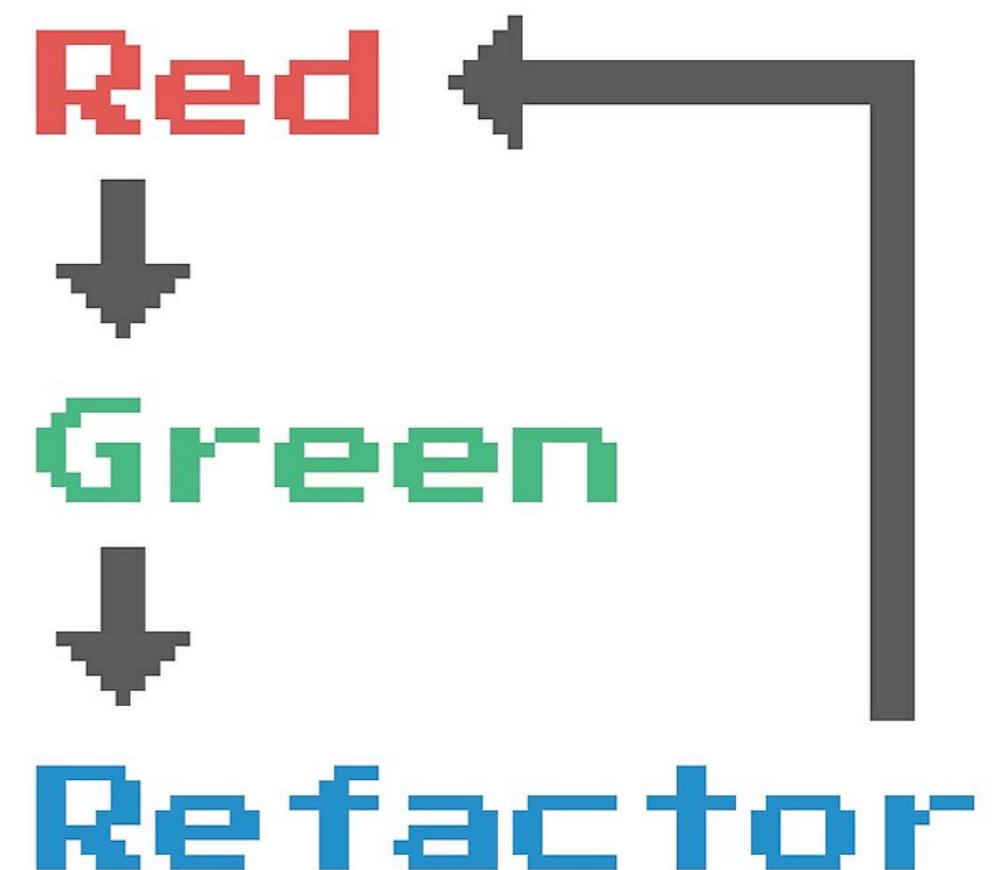
Impact Durabilité

Meilleures UX / Moins durabilité

# TEST DRIVEN DEVELOPMENT

## RED GREEN REFACTOR

1. Réception d'une demande de fonctionnalité
2. Écrire un test qui échoue
3. Écrire le code, jusqu'à ce que le test passe
4. Nettoyer le code
5. Itérer



# CARACTÉRISTIQUE D'UN TEST

## TEST UNITAIRE

Isolé: pas de dépendance à un composant externe (Base de données, système de fichier, ...)

Reproductible: retourne toujours le même résultat si vous ne modifiez rien entre les executions

Temps opportun: l'écriture d'un test doit prendre peu de temps au regard du code testé, si vous avez du mal à écrire un test, vous avez probablement un problème de couplage fort dans votre logiciel

Documentation exécutable: un test utile ne se contente pas de tester du code, il fournit une documentation à l'équipe technique. C'est ce qui doit aider à définir le bon niveau test, on évite les micro comme les méga tests

# PROPRIÉTÉ ?

Remember last Christmas !

Quand a été fêté noël en 2022 ?

Quand a été fêté noël en 2020 ?

Quand a été fêté noël en 1990 ?



Le 25 décembre est une PROPRIÉTÉ qui définit « Noël »

# LES TESTS

DU TU AU PBT

**Unit Tests**

**Property Based Tests**

Fixed input

Random input

One execution

Many executions

Assert result

Assert result or behavior

# TAKE AWAY

## NOUS AVONS VU

Les tests sont une documentation à jour

Les TU testent des cas particulier et peuvent être généralisé par des tests d'invariants

Les tests de propriétés permettent de valider les règles business et de découvrir des bugs qui passeraient à la trappe des TU classiques



# TROLL OF FAME KATA

LE ROI DES TROLLS,  
GNONPOM, A CODÉ LE  
TROLL OF FAME: UNE  
APPLICATION  
FABULEUSE QUI AIDE LES  
TROLLS A APPRENDRE LES  
NOMBRES QUAND ILS  
CHASSENT.



GNONPOM ÉTAIT UN ROI  
DÉVELOPPEUR, FÉRU DE  
TEST DRIVEN  
DÉVELOPPEMENT. IL A MIS  
EN PRODUCTION TOF  
QUAND TOUS LES TESTS  
ÉTAIENT VERTS.

MALHEUREUSEMENT, IL A ÉTÉ ABATTU PAR UN HORRIBLE ELFÉ.

VIVE LE NOUVEAU ROI, VIVE LE TROLL AKLASS!

CETTE FOIS C'EST DÉCIDÉ LE TOURNOI DE CHASSE À L'ELFÉ EST LANCÉ !

A LA FIN DE CHAQUE BATAILLE, LES TROLLS VEULENT COMPARER LES NOMBRES ET ATTRIBUTS DE CES ELFES DÉGOÛTANTS.

AVEC TOF ÇA DEVRAIT ÊTRE FACILE ... CA DEVRAIT.

# TRAVAUX PRATIQUES

## DÉBUTER UN TP

Les TP sont gérés avec Github Classroom

Cliquez sur le lien fourni et acceptez

MIAGE Lille

Accept the assignment —  
tof-group2

Once you accept this assignment, you will be granted access to  
the [tof-group2-oteku](#) repository in the [miage-lille](#) organization on  
GitHub.

[Accept this assignment](#)

Cela crée un repository privé personnel dans le groupe miage-lille

⚠ Vous avez jusqu'au mercredi suivant pour terminer

⚠ 1 étudiant = 1 TP = 1 repo

# TRAVAUX PRATIQUES

## ENVIRONNEMENT DE DÉVELOPPEMENT

Tous les TPs peuvent être réalisé:

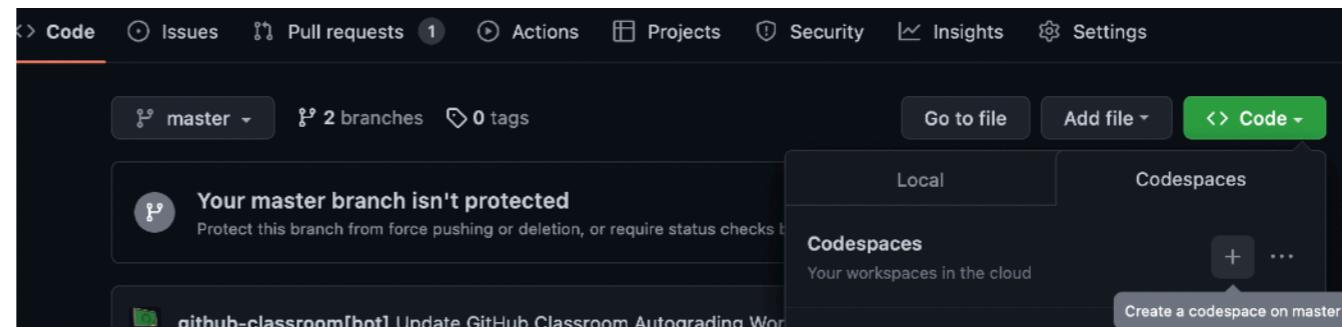
Via vscode + docker plugin (nécessite docker en local)

Via un webIDE:

Gitpod (possible de connecter un vscode local avec gitpod plugin)

ou

GitHub codespace



⚠️ Tout autre choix est sous votre responsabilité!

# TRAVAUX PRATIQUES

## LIENS

GROUPE 1 (Thomas) <https://classroom.github.com/a/1F9SU84Y>

GROUPE 2 (Quentin) <https://classroom.github.com/a/ngKNSmhq>

# THE BOSS

SI VOUS VOULEZ ALLER PLUS LOIN

Renforcement :

🎥 [Types VS Tests](#)

🎥 [Much Ado About Testing](#)



# **QUALITÉ DU SI**

---

COURS 2 - LA QUÊTE DES MACHINES ÉTATS

# THE QUEST

La plupart des bugs (i.e., dégâts financiers pour mon entreprise) rencontrés en prod (dans ma vie) sont liés à :

des MACHINES ÉTATS implicites

du code qui crash au RUNTIME

des EFFETS non maîtrisés

... et j'aime pas ça !!!

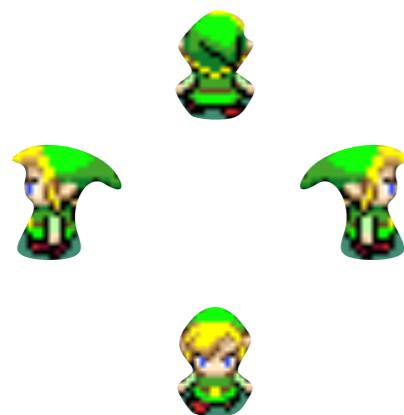
Peut-on améliorer la conception logicielle,  
puis SI pour éviter cela ?

OUI c'est le but de ce cours



# LINK TO THE PATH

## IMPLÉMENTATION NAÏVE EN TYPESCRIPT



```
enum Direction {  
    North,  
    East,  
    South,  
    West  
}  
  
const label = (d : Direction) : string => {  
    switch (d){  
        case Direction.North: return "north";  
        case Direction.South: return "south";  
        case Direction.East: return "east";  
        case Direction.West: return "west";  
    }  
}  
  
console.log(label(1));  
console.log(label(Direction.East));  
console.log(label(Direction.South));  
console.log(label(4));
```

# LINK TO THE PATH

## IMPLÉMENTATION NAÏVE EN TS



```
console.log(label(-35));
```

No ERROR

Undefined

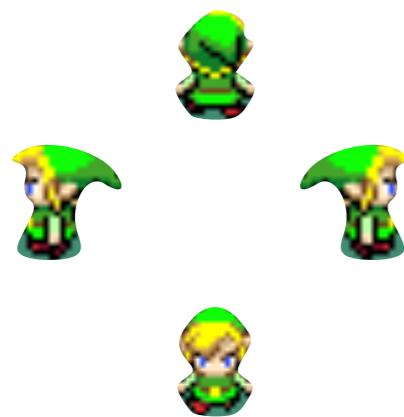
-35 n'est pas une Direction valide ... mais les valeurs d'un enum sont en réalité des numbers

Direction.North = 0 ... Direction.West = 3

Vu du système de type les enums sont des numbers 😱

# LINK TO THE PATH

# FAISONS CRASHER LE RUNTIME?



```
const label = (d : Direction) : string => {
    switch (d) {
        case Direction.North: return "north";
        case Direction.South: return "south";
        case Direction.East: return "east";
        case Direction.West: return "west";
    }
    throw new Error("Not a valid Direction")
}
```

Compilation ERROR

**Unreachable code detected.**

**TS croit qu'on traite tous les cas ... Les enums sont dangereux!**

# LINK TO THE PATH

## LES UNIONS



```
type Direction = "North" | "East" | "South" | "West";  
  
const label = (d : Direction) : string => {  
    switch (d){  
        case "North": return "north";  
        case "East": return "east";  
        case "South": return "south";  
        case "West": return "west";  
    }  
}
```

"North", "East", "South", "West" sont des types! (String literal type)

# LINK TO THE PATH

## LES UNIONS



```
console.log(label(-35));
```

Compilation ERROR

Argument of type '-35' is not assignable to parameter of type 'Direction'.

On peut valider dès la compilation que l'on utilise des valeurs valides !!!

# LINK TO THE PATH

## LES UNIONS



```
const label = (d : Direction) : string => {
  switch (d){
    case "North": return "north";
    case "East": return "east";
    case "South": return "south";
    //case "West": return "west";
  }
}
```

Compilation Error

Function lacks ending return statement and return type  
does not include 'undefined'.

On peut valider dès la compilation qu'on traite TOUTES LES valeurs valides !!!

# LINK TO THE PATH

ET EN JAVA? JAVA 15 (SEALED), JAVA 14 (RECORD) ET JAVA 17 (EXHAUSTIVITÉ)

```
sealed interface Direction {  
    record North() implements Direction {}  
    record East() implements Direction {}  
    record South() implements Direction {}  
    record West() implements Direction {}  
}  
public static final String label(Direction d) {  
    return switch(d){  
        case Direction.North n -> "north";  
        case Direction.East e -> "east";  
        case Direction.South s -> "south";  
        case Direction.West w -> "west";  
    }  
}
```

Enfin des interfaces scellées + record permettent de décrire correctement un type « OU »

Finalement JEP 406 <http://openjdk.java.net/jeps/8213076> Java 17 (Septembre 2021) apporte le « switch/case » pour l'exhaustivité d'un pattern matching, ainsi que c'est un nouveau « switch/case » expressif (parce que l'instruction switch java est quand même bien pourri)

Puis JEP 420 Java18 pour une amélioration des pattern de déconstruction au sein d'un switch/case expressif

Modèle similaire à Kotlin avec 10 ans de retard ou Scala avec 15 ans de retard ...

Mettre en prod un projet JAVA < 18 est une faute professionnelle 😱

# TAKE AWAY

AVEC UN LANGAGE STATIQUEMENT (BIEN) TYPÉ

On peut valider qu'on traite uniquement les valeurs valides

On peut valider qu'on traite toutes les valeurs valides

Dès la compilation

Dans les langages ML : OCaml, Haskell, F#, ...

Mais aussi les langages modernes qui s'en inspirent : TS, Scala, Kotlin, Swift, Rust, C++17, ...



# **LA PLUPART DES APP DE GESTION SONT DES MACHINES ÉTATS**

**LES TRANSITIONS DE LINK**



**Link regarde dans une direction, avance, s'arrête, etc...**

# MODÉLISER LA COMMANDE

UN TYPE « ET »

```
type Direction = "North" | "East" | "South" | "West";

type Command = {
    order: string,
    direction: Direction
}

const turnEast : Command = {
    order: "face",
    direction: "East"
}
```

Le type command est un order de type string ET une direction de type Direction

# MODÉLISER LA COMMANDE

UN TYPE « ET »



```
const yolo : Command = {  
    order: "triple_backflip",  
    direction: "East"  
}
```

Triple Backflip n'est pas un ordre valide

# MODÉLISER LA COMMANDE

ON A DÉJÀ VU COMMENT RÉSoudre CELA => ORDER : UN TYPE « OU »

```
type Direction = "North" | "East" | "South" | "West";  
  
type Order = "Face" | "Start" | "Stop"  
  
type Command = {  
    order: Order,  
    direction: Direction  
}
```

# MODÉLISER LA COMMANDE

ORDER : UN TYPE « OU »



```
const noSense : Command = {  
    order: "Start",  
    direction: "East"  
}
```

Start East n'est pas une commande valide

# MODÉLISER LA COMMANDE

ORDRE AND TYPE DE PARAMÈTRES → PARMI LESQUELS UN DISCRIMINANT

```
type Direction = "North" | "East" | "South" | "West";  
  
type Direction = "North" | "East" | "South" | "West";  
  
type Command = {kind: "Face", direction: Direction} | {kind: "Start"} | {kind: "Stop"}  
{}
```

Un discriminant est une propriété de type qui permet de distinguer les membres d'un type union

# MODÉLISER LA COMMANDE

COMMAND : UN TYPE « OU »



```
const stop_ : Command = {kind: "Stop"}      direction: "East"}
```

# TAKE AWAY

**LES TYPES « OU » SONT TRÈS UTILES**

Les types « ET » s'appellent aussi record ou types produit

*Les classes sont des types « ET »*

Les types « OU » s'appellent aussi variants ou types somme



# SCRIPT DE COMMANDE

ENCHAINER LES COMMANDES AVEC UN TYPE OU RÉCURSIF



```
type Direction = "North" | "East" | "South" | "West"

type Command =
| {kind: "Face", direction: Direction}
| {kind: "Start"}
| {kind: "Stop"}
| {kind: "Chain", first: Command, second: Command}

const moveEast = {
    kind:"Chain",
    first: {kind: "Face", direction: "East"} ,
    second: {
        kind:"Chain",
        first: {kind: "Start"} ,
        second: {kind:"Stop"}
    }
}
```

# SCRIPT DE COMMANDE

## UNE COMMANDE PLUS COMPLEXE

Est complexe à lire

```
const moveWestThenNorth = {
    "kind": "Chain",
    "first": {
        "kind": "Face",
        "direction": "West"
    },
    "second": {
        "kind": "Chain",
        "first": {
            "kind": "Start"
        },
        "second": {
            "kind": "Chain",
            "first": {
                "kind": "Stop"
            },
            "second": {
                "kind": "Chain",
                "first": {
                    "kind": "Face",
                    "direction": "North"
                },
                "second": {
                    "kind": "Chain",
                    "first": {
                        "kind": "Start"
                    },
                    "second": {
                        "kind": "Stop"
                    }
                }
            }
        }
    }
}
```

# SCRIPT DE COMMANDE

KEEP CALM & USE FUNCTIONS



```
const face = (d:Direction) : Command => ({kind: "Face", direction: d})
const start = () : Command => ({kind: "Start"})
const stop_ = () : Command => ({kind: "Stop"})
const chain = (first: Command, second: Command) : Command => ({kind: "Chain", first, second})

const pipe = (first: Command, ...rest: Array<Command>) : Command =>
  rest.length > 0
    ? chain(first, pipe(rest[0], ...rest.slice(1)))
    : first
```

```
const moveWestNorth = pipe(
  face("West"),
  start(),
  stop_(),
  face("North"),
  start(),
  stop_()
)
```

# PIPE OPERATION

## REMINDER

Beaucoup de langages utilisent l'opérateur infix `|>` pour la fonction pipe

```
pipe(x, foo) = x |> foo = foo(x)
```

Il existe une proposition TC39 en stage 2 pour JavaScript <https://github.com/tc39/proposal-pipeline-operator>

Stage 2 étant encore incertain, je préfère utiliser la fonction pipe en JS/TS, nous utiliserons l'implémentation de la librairie fp-ts. Mais vous pouvez utiliser `|>` avec un polyfill.

# TAKE AWAY

## NOUS AVONS VU

Le type **Command** est récursif : s'exprime en terme de {kind: "Chain", first **Command**, second **Command**}

Un système qui a des types « ET », des types « OU » et des types récursifs s'appelle un système de données algébriques (ADT)

Un ADT permet de représenter les valeurs autorisées d'un programme et leurs transitions



# SUIVRE LES TRANSITION

NOUS POUVONS ÉCRIRE DES TRANSITIONS INVALIDES



```
const invalid = pipe(  
  face("East"),  
  stop_  
)
```

# SUIVRE LES TRANSITION

## APPROXIMATION D'UN GADT

```
import { identity } from "fp-ts/lib/function";

type Direction = "North" | "East" | "South" | "West"

type Command<A, B, C> =
  { kind: "Face", direction: Direction, proof: (_a: A) => A }
  { kind: "Start", proof: (_a: A) => B }
  { kind: "Stop", proof: (_a: A) => B }
  { kind: "Chain", first: Command<A, B, any>, second: Command<B, C, any> }

const face = (d: Direction)
  : Command<"Idle", "Idle", void> => ({ kind: "Face", direction: d, proof: identity<"Idle"> })
const start = (): Command<"Idle", "Moving", void> => ({ kind: "Start", proof: (_: "Idle") => "Moving" })
const stop_ = (): Command<"Moving", "Idle", void> => ({ kind: "Stop", proof: (_: "Moving") => "Idle" })
const chain = <A, B, C>(first: Command<A, B, any>, second: Command<B, C, any>)
  : Command<A, C, any> => ({ kind: "Chain", first, second }) as Command<A, C, any>
```

On paramètre le type command avec 3 paramètres

Pour Face, B & C sont des types fantômes

Pour Start et Stop, C est un type fantôme

On ajoute des preuves :

Face conserve l'état de la machine: "**Idle**". La fonction Identity peut être utilisée comme preuve.

Start fait avancer la machine à états : "**Idle**" => "**Moving**"

Start fait avancer la machine à états : "**Idle**" => "**Moving**"

Chain fait avancer la machine à états en composant des commandes par associativité : la preuve est directement encodée dans leurs types

# SUIVRE LES TRANSITION

ON PEUT CRÉER UNIQUEMENT DES COMMANDES VALIDES

```
/* COMPILE */
let validCmd =
  chain(
    chain(
      face("East"),
      start()
    ),
    stop_()
  );

/* ERREUR DE COMPILEMENT */
let invalidCmd = chain(
  face("East"),
  stop_()
);
```

# TAKE AWAY

## NOUS AVONS VU

Les GADT sont des types « OU » qui possèdent des témoins de type

ADT + GADT permettent de valider qu'on ne représente que des états autorisés **et** qu'on effectue que des transitions d'états autorisés

En Java ou TS, leur encodage et le messages d'erreurs sont complexe; leur utilisation est discutablement intéressante mais possible

Beaucoup plus aisé en Haskell ou OCaml



# TENNIS KATA



# TRAVAUX PRATIQUES

## LIENS

GROUPE 1 (Thomas) <https://classroom.github.com/a/Nd8M0mh6>

GROUPE 2 (Quentin) [https://classroom.github.com/a/CSHbQ\\_xq](https://classroom.github.com/a/CSHbQ_xq)

# THE BOSS

SI VOUS VOULEZ ALLER PLUS LOIN

Renforcement :



[Comprendre les ADT](#)

🎥 [The power of composition](#)

Diversification :

🎥 [Categories for the Working Hacker](#)

🎥 [Writing Safer Code Using GADTs](#)



# **QUALITÉ DU SI**

---

COURS 3 - FRONTEND DATAFLOW

# HISTORIQUE

30 ANS DE WEB



AoL ALttP LA

OoT OoA OoS

TP SS

LoZ OoT MM

FS WW FSA

MC PH ST



# WEB DEVELOPMENT

90'S : L'ÈRE DU HTML

Pages statiques HTML

Applet Java, DHTML + CGI, Flash (RIP 2021)

# Apple



May 8, 1998

Hot News Headlines

Pro. Go. Whoa. A Strategy as Simple as the Macintosh.



Pro.

Creative professionals, meet your match.



Go.

We rewrote the book on mobile computing.



Whoa.

It's okay, you don't have to say anything.

The  
Apple Store

Hot News  
About Apple

Products  
Support

Design & Publishing  
Education

Developer  
Where to Buy



Find:

Shortcut Search

[Site Map](#) · [Search Tips](#) · [Index](#)

[The Apple Store](#) | [Hot News](#) | [About Apple](#) | [Products](#) | [Support](#)  
[Design & Publishing](#) | [Education](#) | [Developers](#) | [Where to Buy](#) | [Home](#)  
[Job Opportunities at Apple](#)

Visit other Apple sites around the world: Choose... Go

# WEB DEVELOPMENT

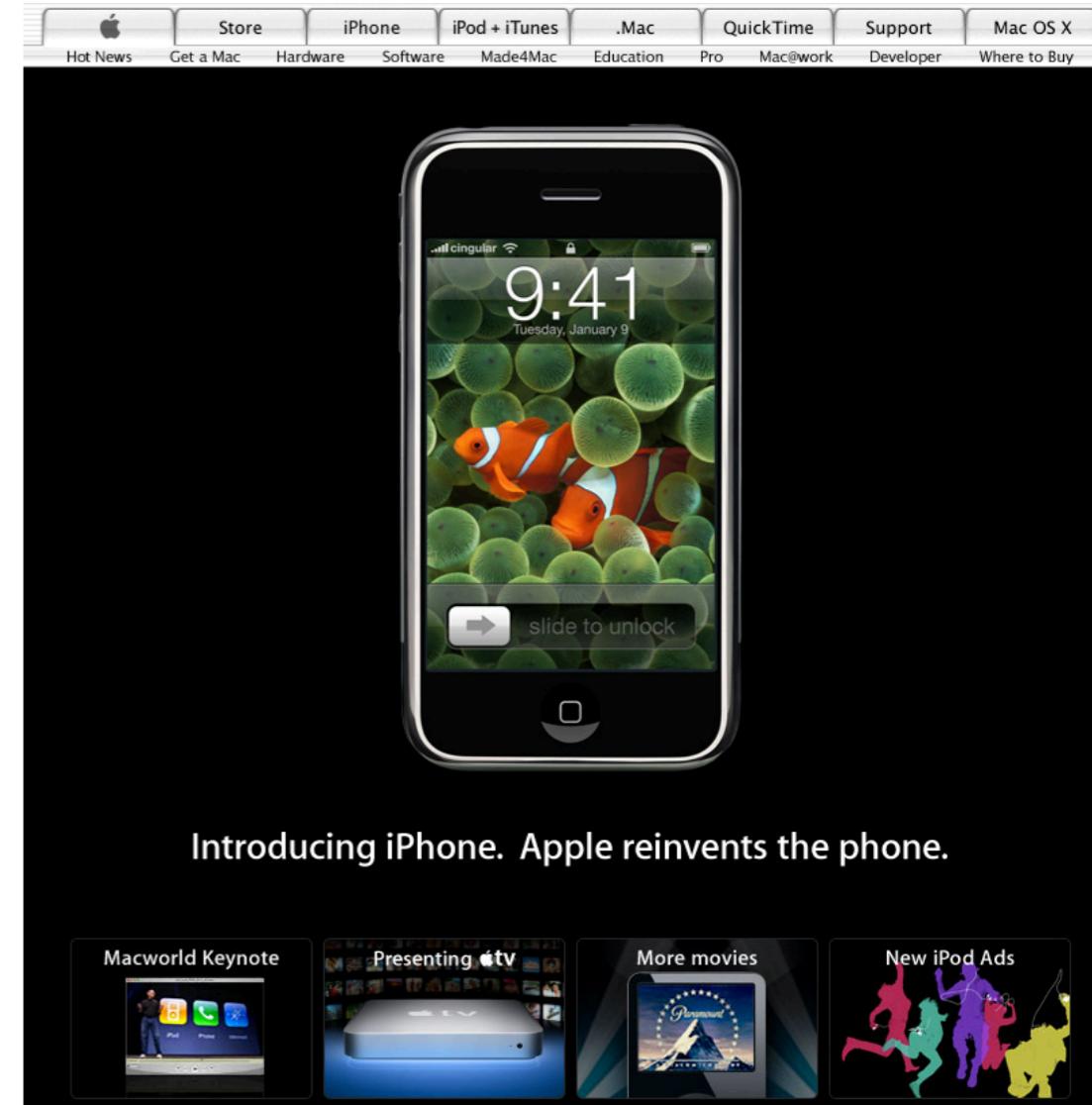
00'S : L'ÈRE DE LAMP

Server Side Rendering

Stack star LAMP : Linux Apache PHP MySql

Alternatives Spring MVC, ASP.net, Django ou Ruby on Rails

Succès du templating (Mustache, Jinja, Twig, ...)



# WEB DEVELOPMENT

10'S : L'ÈRE DE JAVASCRIPT

Scission tech entre App & website :  
App JS Front-end OU Static Site Generation

La « guerre » : React VS Angular VS Vue.js

La finalisation des Web Component (lit-element)

2 approches antagonistes : UI expressive (React  
... ou langages expressifs qui compilent vers js)  
VS UI Components (séparation template / logique  
: Angular, Vue, Web Component, Svelte)



# THE QUEST

Gestion des états en UI

Les architectures Front Back sont actuellement le standard

Les backends sont souvent des applications CRUD

La gestion de l'état applicatif s'opère dans le front end

Ce qui amène de nouvelles problématiques

- Comment éviter les états incohérents ?
- Comment avoir des états prévisibles ?
- Comment faire circuler l'information dans l'UI ?



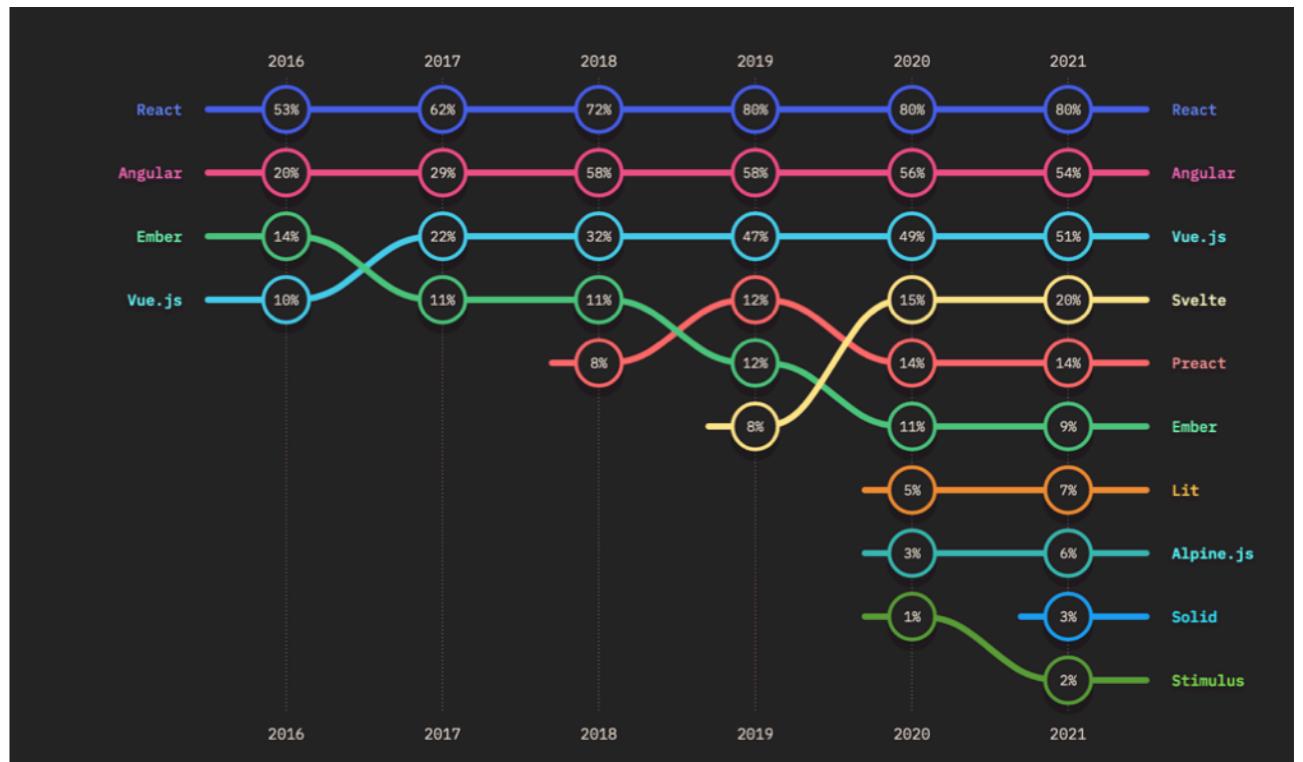
# DEV FRONT END

JS EVERYWHERE

Quelle approche ?

- Frameworks : Angular / Vue / React / Svelte

- Standard : Web Components  
(lit-element, lit, Stencil)



StateOfJS 2021 : Usage des frameworks frontend

# DEV FRONT END

JS EVERYWHERE

Javascript est un langage complexe

Demande l'utilisation de beaucoup lib tierces combler ses manques (Ramda, Immutable.js, Flow, eslint, ...)

Evolue très vite sans jamais faire table rase du passé (don't break the web)

=> Ça explique que JS soit de plus en plus utilisé comme un « bytecode » pour des langages qui compilent vers JS, avec Typescript en première ligne ! mais aussi Elm, Rescript, ClojureScript, Purescript, KotlinJs, Ocaml ...

# DEV FRONT END

CHOISIR UNE OPTION



Dans le cadre du cours, nous allons nous appuyer sur:

- Langage: Typescript (verbeux et mauvaise inférence, mais nécessaire à maîtriser en 2023)
- Framework : React
  - Seule lib front "mainstream" à avoir une approche expressive !
  - Un composant React = Une fonction qui prend en paramètre un objet props et retourne un objet de type React.Element
  - Un composant monté dans le DOM correspond donc à un objet instancié par un pattern Factory
  - Vous pouvez utiliser le DSL JSX pour décrire les Elements

```
import React from "react";  
  
const Welcome = ({ name } /* destructuring de props */) => <h1>Hello, {name}</h1>;
```

NE CREEZ PAS DE COMPOSANTS AVEC DES CLASS, N'UTILISEZ PAS LES LIFECYCLES

# POINT D'ATTENTION SUR REACT

C'EST PAS SIMPLE DE DÉBUTER REACT EN 2023

2014

```
var Component2014 = React.createClass({  
  render(props){  
    ...  
  }  
});
```

2016

```
const StatelessComponent2016 = (props) => <div> ... </div>;  
class StatefulComponent extends React.Component {  
  constructor(props) {  
    super(props);  
    ...  
  }  
  render(props) {  
    ...  
  }  
}
```

2017

```
class StatelessComponent2017 extends React.PureComponent {  
  render(props) {  
    ...  
  }  
}  
class StatefulComponent extends React.Component {  
  constructor(props) {  
    super(props);  
    ...  
  }  
  render(props) {  
    ...  
  }  
}
```

+ Mixins  
+ Flux

+ HOC  
+ REDUX

+ Render props  
+ Context

+ API changes

+ API changes

Les approches >2016 fonctionnent encore

Autant de « visions » que de projets ... c'est normal React est une lib qui laisse beaucoup de liberté dans l'architecture du projet

# POINT D'ATTENTION SUR REACT

C'EST PAS SIMPLE DE DÉBUTER REACT EN 2023

2019



2022

```
const futureComponent = (props) => {  
  ...  
  return <div> ... </div>  
}
```

```
const futureComponent = (props) => {  
  ...  
  return <div> ... </div>  
}
```

(Pas de changement sur la déclaration des composants)

+ Hook  
+ Context

+ Concurrent  
+ Automatic Batching

DANS LE CADRE DU COURS JE VOUS IMPOSE UNE « VISION » à respecter ... même si vous utilisez déjà React autrement

Documentation React : <https://beta.reactjs.org>

# POINT D'ATTENTION SUR REACT

## ÉTAT LOCAL OU ÉTAT GLOBAL ?

- Un état global unique facilite la gestion de la logique applicative
- Un état local est parfois utile pour une logique de composant réutilisable  
(datepicker, ...)

Documentation React : <https://beta.reactjs.org>

# GESTION D'ÉTATS

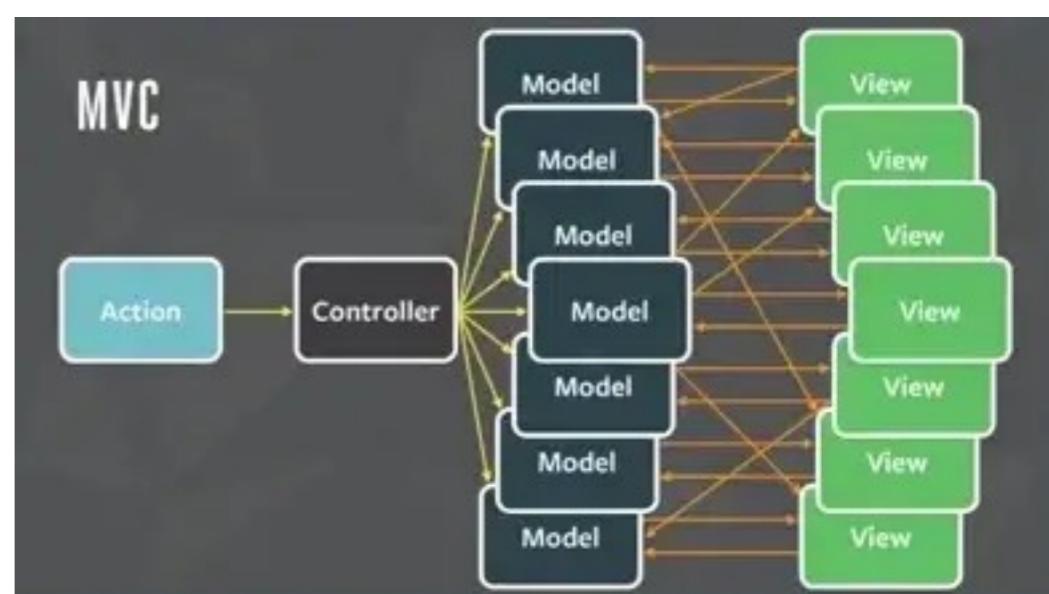
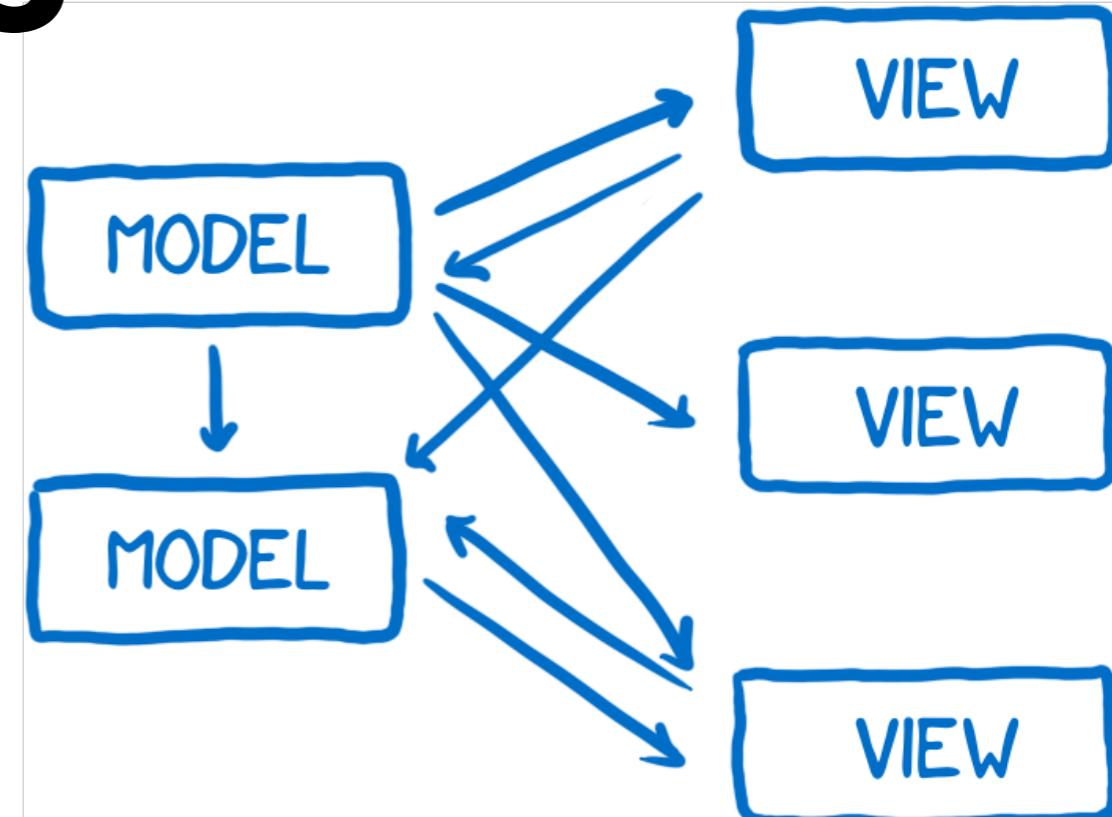
MVC / MVVM CÔTÉ CLIENT

1. Le modèle transmet des données à la vue
2. La vue met à jour le modèle sur la base d'interaction utilisateur
3. Le modèle mets à jours LES VUES qui l'utilisent

Chaque changement peut être asynchrone

Chaque changement peut en engendrer d'autres en cascade

Comment debugger un tel flux de données 🎲



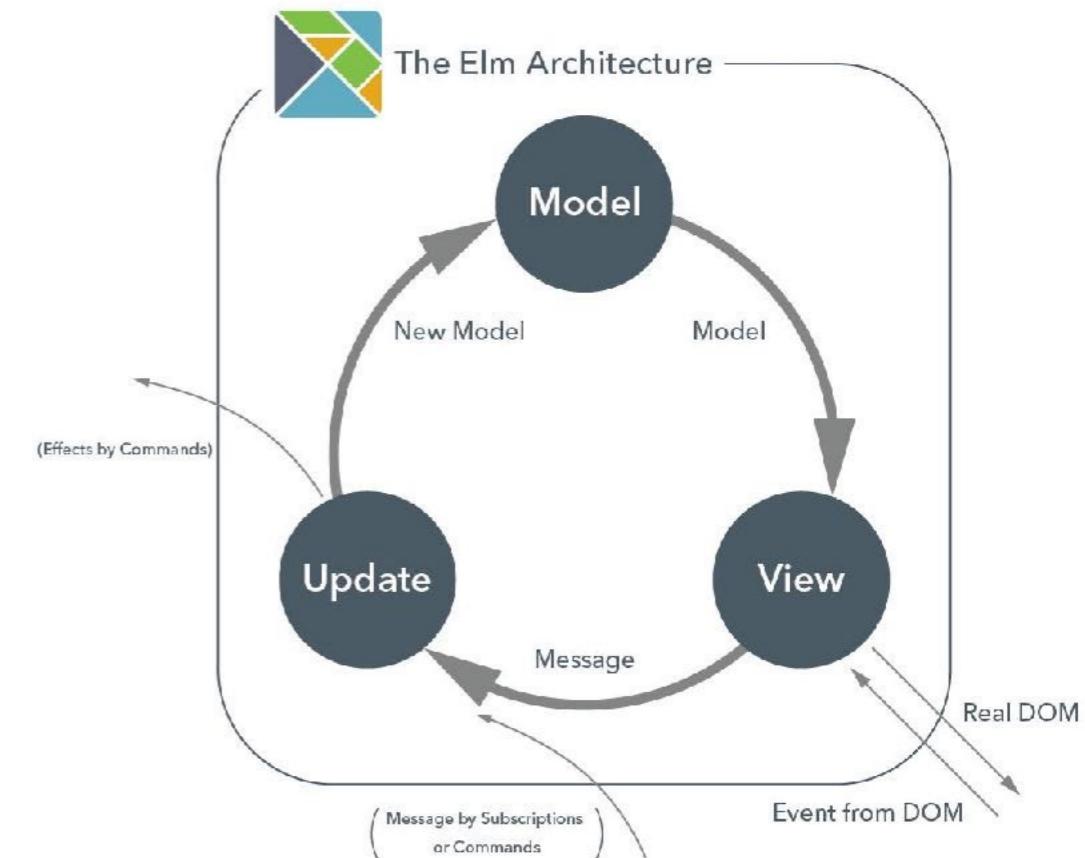
# GESTION D'ÉTATS

## ONE WAY DATA FLOW

MVC / MVVM la circulation des données est dans les deux sens (model <-> vue)

Le « One Way Data Flow » est un flux de circulation des données unidirectionnel, popularisé par Elm, démocratisé par Flux puis Redux.

Il est inspiré de la Elm Architecture

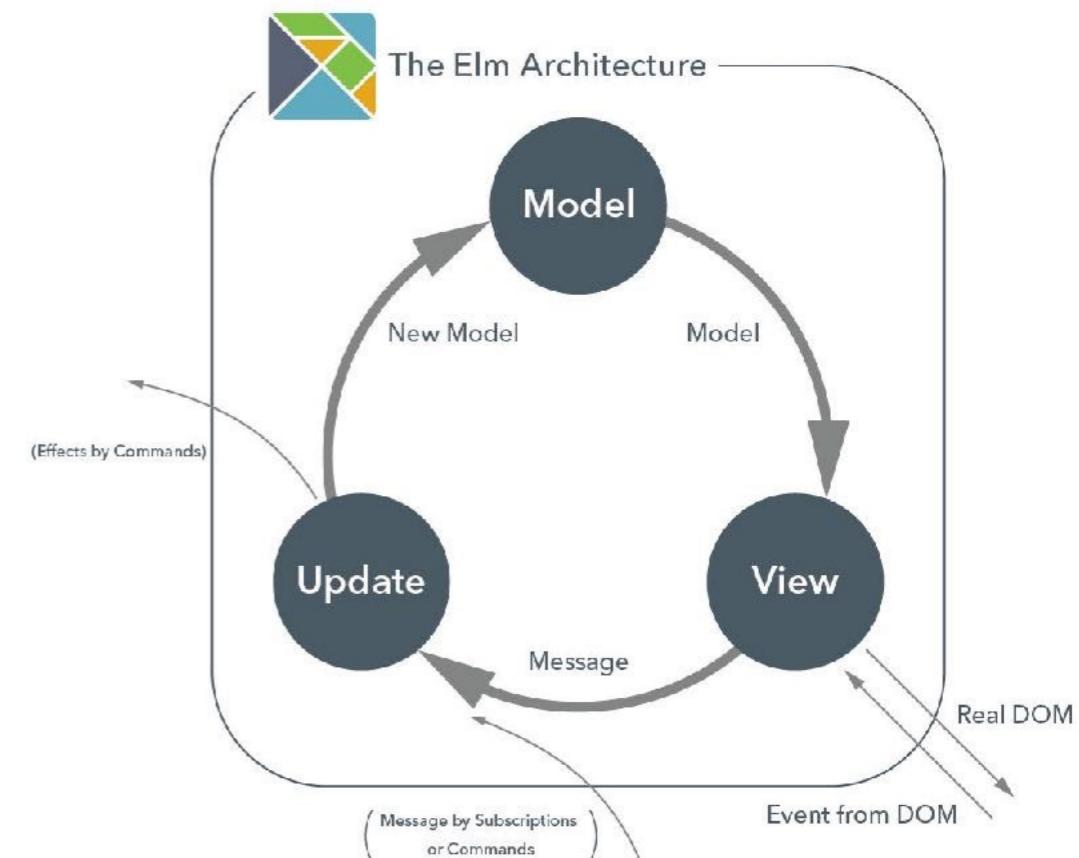


# GESTION D'ÉTATS

## ELM ARCHITECTURE

Implémentations :

- Elm, F# Elmish, Rust Yew.rs, ocaml-vdom
- redux-loop + \*js



A inspiré :

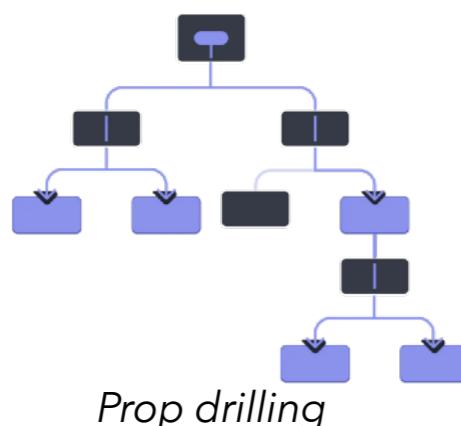
- Redux + \*js
- react useReducer hook (au niveau state local)

# GESTION D'ÉTATS

## ELM / CONTEXT

Pour éviter la complexité liée à Redux dans un premier temps, on peut implémenter la logique de la Elm Architecture à l'aide de l'API Context et d'un hook useReducer

Un contexte permet d'éviter le « prop drilling » et ainsi de pouvoir utiliser des données facilement, partout dans notre application.



```
const initialModel = {};
const ModelContext = createContext(null);
const SendMessageContext = createContext(null);

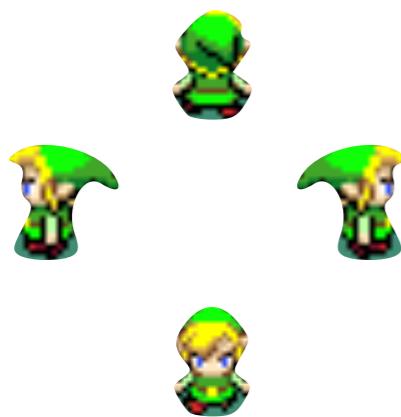
const update = (model, message) => {
  switch (message.type
    // Describe state machine here
  ) {
  }
  return model;
};

export const ModelProvider = ({ children }) => {
  const [model, sendMessage] = useReducer(update, initialModel);

  return (
    <ModelContext.Provider value={model}>
      <SendMessageContext.Provider value={sendMessage}>
        {children}
      </SendMessageContext.Provider>
    </ModelContext.Provider>
  );
};
```

# STATE MACHINE STRIKE BACK

SI ON ADAPTAIT NOTRE MACHINE



```
interface North {
    type: "north";
}
interface East {
    type: "east";
}
interface South {
    type: "south";
}
interface West {
    type: "west";
}

type Direction = North | East | South | West

const label = (d: Direction) => {
    switch (d.type) {
        case "north": return "North"
        case "east": return "East"
        case "south": return "South"
        case "west": return "West"
    }
}
```

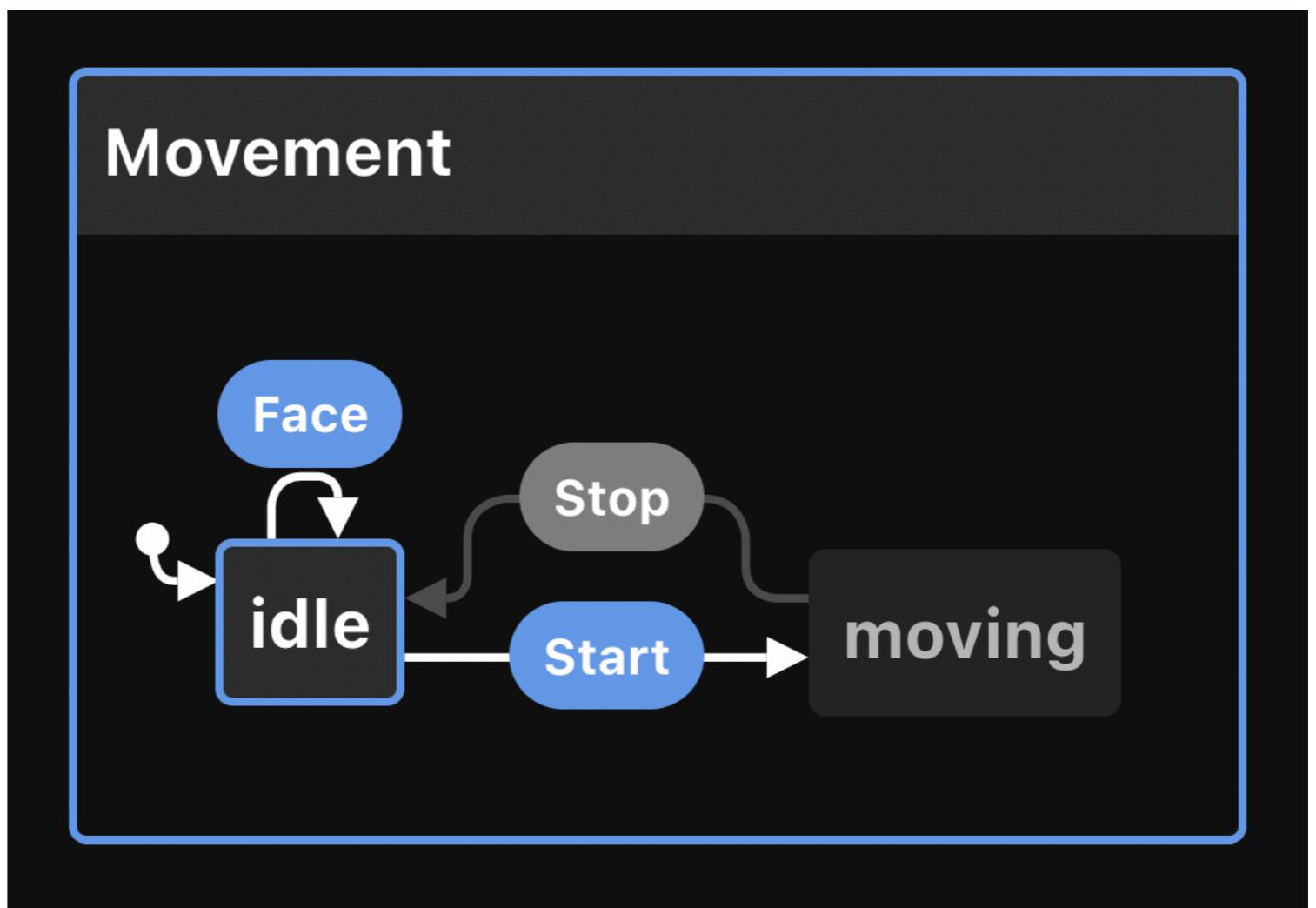


a des « tagged union » qui sont des types OU avec un pattern matching exhaustif

Il suffit que les interfaces partagent un « tag » = un attribut

# STATE MACHINE STRIKE BACK

RAPPEL DE NOTRE MACHINE A ÉTATS



# STATE MACHINE STRIKE BACK

ON ADAPTE UN PEU LES COMMANDES...



```
interface Face {
    type: "face";
    direction: Direction;
}
interface Start {
    type: "start";
}
interface Stop {
    type: "stop";
}
type Command =
    | Face
    | Start
    | Stop

const stop: () => Stop = () => ({ type: "stop" });
const start: () => Start = () => ({ type: "start" });
const face: (d: Direction) => Face = (d: Direction) => ({ type: "face",
direction: d });
```

Face, Start et Stop sont des types, on peut créer nos constructeurs de valeurs avec des fonctions.

# STATE MACHINE STRIKE BACK

DONC ON VEUT STOCKER L'ÉTAT



```
interface Idle {
  type: "idle";
}
interface Moving {
  type: "moving";
}
type State = Idle | Moving

const idle : () => Idle = () => ({
  type: "idle"
});
const moving : () => Moving = () => ({
  type: "moving"
});

const initialState : State = idle();
```

# STATE MACHINE STRIKE BACK

ET UNE FONCTION D'UPDATE = REDUCER



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

# 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)
    }
}
```

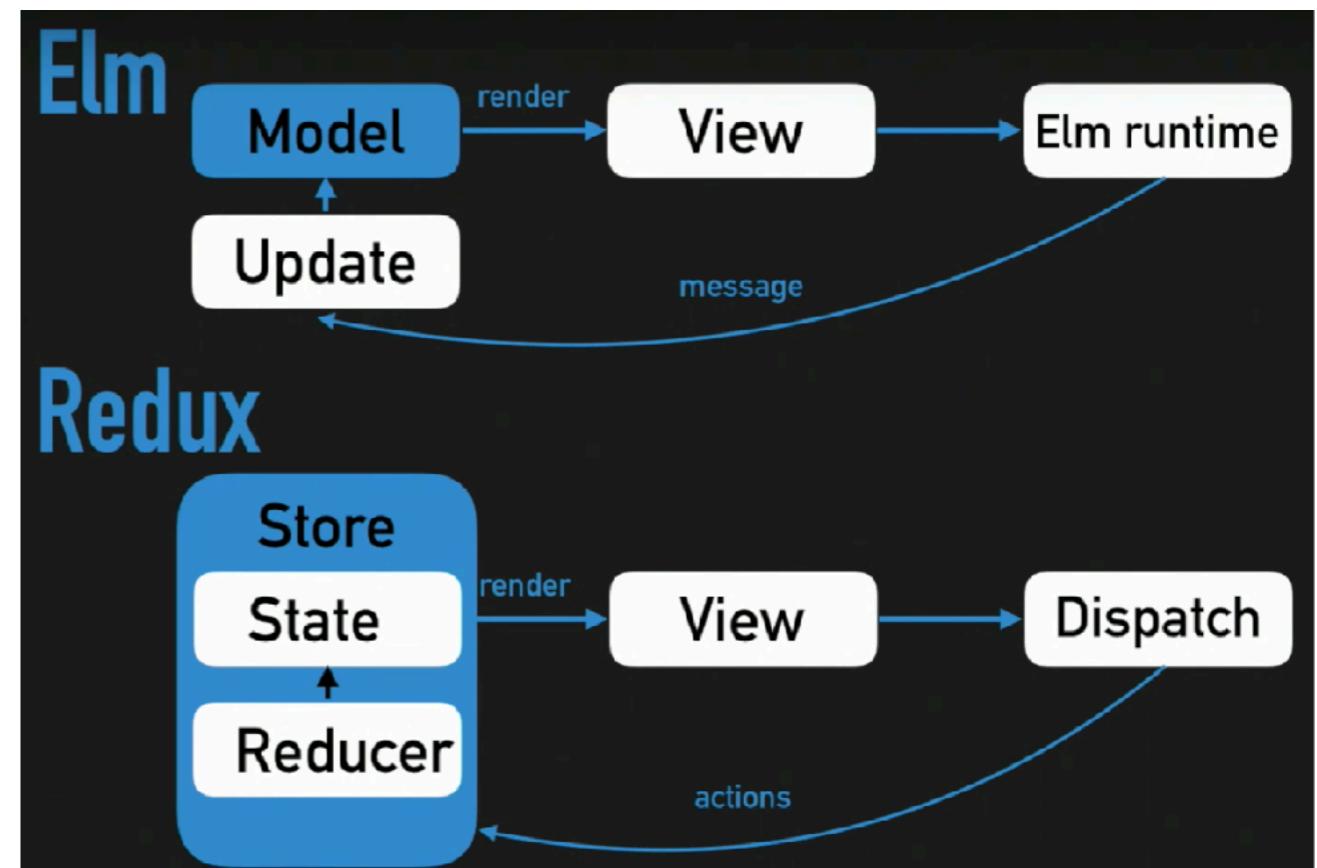
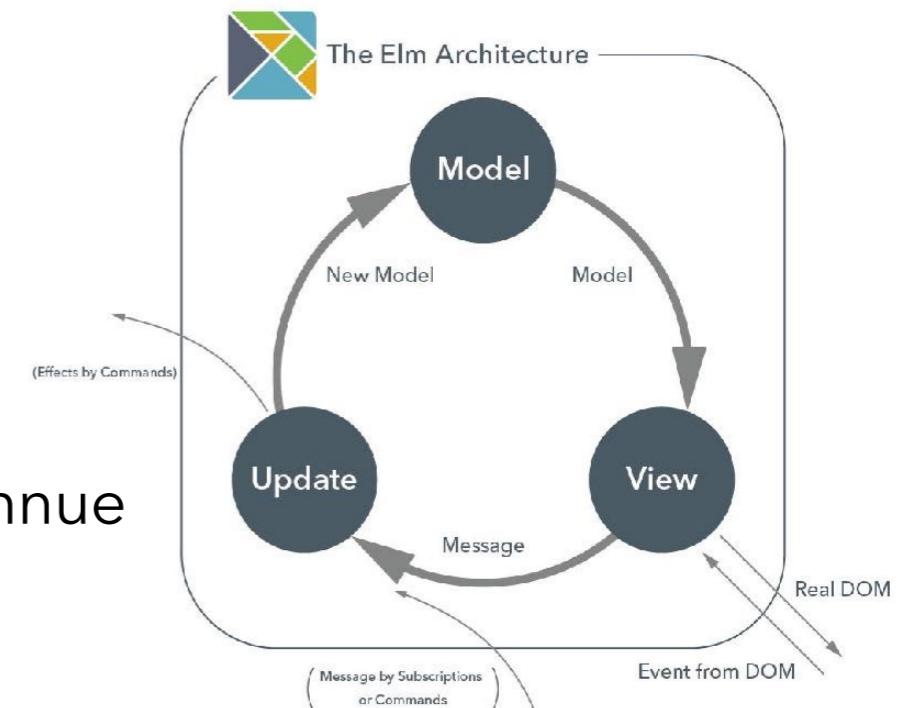
# GESTION D'ÉTATS

## ELM / REDUX

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

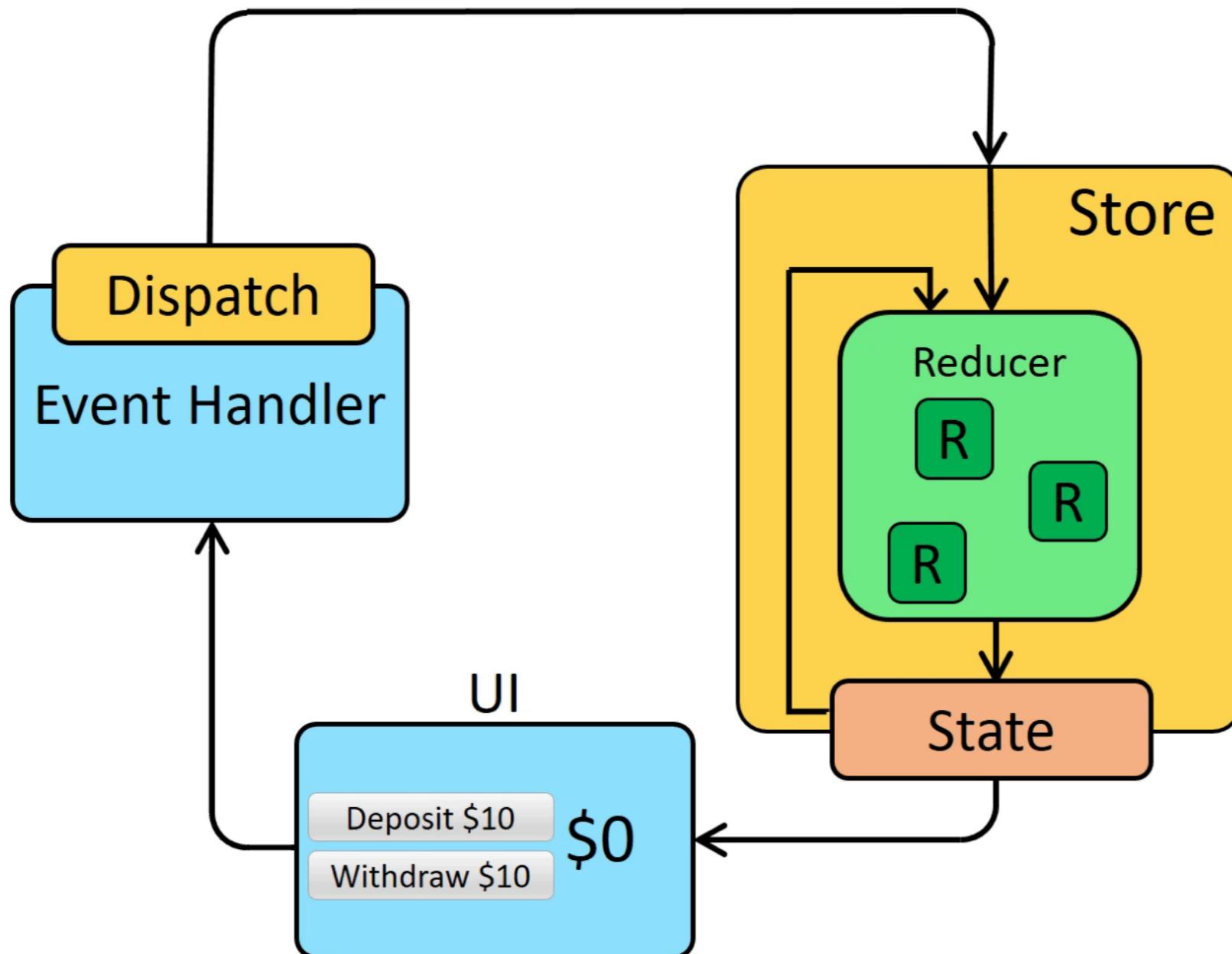
On peut donc également faire le parallèle avec celle-ci :

- 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



# GESTION D'ÉTATS

REDUX



# GESTION D'ÉTATS

## EXEMPLE AVEC REDUX : TODOLIST

```
// action.ts
type NewTodo = {
  type: 'NEW';
  todo: Todo;
};
type CheckAll = {
  type: 'CHECK_ALL';
};
type Action = NewTodo | CheckAll;

const newTodo = (): NewTodo => ({
  type: 'NEW',
  todo: { checked: false, content: '' },
});
const checkAll = (): CheckAll => ({ type: 'CHECK_ALL' });

// store.ts
import { Reducer, Store, legacy_createStore as createStore } from 'redux';
const store: Store<State, Action> = createStore(reducer);

// view.tsx
import { useDispatch, useSelector } from 'react-redux';
const Counter = () => {
  const dispatch = useDispatch();
  const todos = useSelector(todosSelector);
  return (
    <div>
      {todos.map(t => (
        <p>{t.content}</p>
      ))}
      <button onClick={() => dispatch(newTodo())} type="button"
title="+" />
    </div>
  );
};
```

```
// reducer.ts
type Todo = { content: string; checked: boolean };
type State = {
  todos: Array<Todo>;
};

const initialState: State = {
  todos: [],
};

const reducer: Reducer<State, Action> = (
  state: State | undefined,
  action: Action
) => {
  if (!state) return initialState;
  switch (action.type) {
    case 'NEW':
      return { ...state, todos: [...state.todos, action.todo] };
    case 'CHECK_ALL':
      return {
        ...state,
        todos: state.todos.map(t => {
          t.checked = true;
          return t;
        }),
      };
  }
};

export const todosSelector = (state: State) =>
  state.todos;
```

# GESTION D'ÉTATS

## REDUX

Remarques :

- **N'utilisez pas Redux Tools Kit** avant d'avoir complètement maîtrisé les concepts de Redux.  
RTK a été conçu pour éviter le boilerplate pour mettre en place Redux dans une application.
- Lisez la **documentation officielle** qui explique bien les concepts de Redux :  
<https://redux.js.org>
- Le lien entre Redux et React se fait notamment via la librairie react-redux. Elle apporte notamment les hooks **useDispatch** et **useSelector**

# GESTION D'ÉTATS

## ALTERNATIVES

React fournit nativement des hooks pour la gestion de l'état local (useState, useReducer) = **spaghetti code pour gérer l'état d'une application réelle**

Redux Toolkit : encodage initial + configuration = **0 safety**

XState : configuration ... mais des outils de qualité ... **mais 0 safety**

Redux Saga : séparation entre description et execution de programmes avec des effets algébriques (simulés grâce aux generators).

**C'est un très bon pattern également !!!**

# TAKE AWAY

## ONE WAY DATA FLOW ROCKS

Avantages :

- **Moins d'erreurs** car vous avez un control plus fin des données
- Plus **facile à debugger** car vous connaissez la provenance et la destination des données: il est facile de faire du « time travel »
- Plus **efficace** car les librairies maîtrisent les limites de chaque partie du système

Ce sont des machines de Moore !



# NOTRE ENVIRONNEMENT DE TP

Une application complexe nécessite de pouvoir chaîner les actions et de gérer des actions asynchrones.

Nous utiliserons:

- redux-loop qui implémente le pattern Elm dans une vision puriste
- fp-ts pour disposer des types Option et Either
- Fast Check pour les PBT

Interdiction d'utiliser l'état local !

# THE BOSS

SI VOUS VOULEZ ALLER PLUS LOIN

Renforcement :

 [Doc React à jour](#) (futur site encore en beta)

 [Elm and Redux Join forces](#) (10 min intro to redux loop)

Diversification:

 [Elm guide](#)

 [Elm in action](#)



# CATSTAGRAM KATA



# TRAVAUX PRATIQUES

## LIENS

GROUPE 1 (Thomas) <https://classroom.github.com/a/ib2wGIKY>

GROUPE 2 (Quentin) [https://classroom.github.com/a/a4D\\_G9dC](https://classroom.github.com/a/a4D_G9dC)

# **QUALITÉ DU SI**

---

COURS 4 - BLOCKCHAINS, SMART CONTRACTS

# THE QUEST

Blockchain

Le web3 est-il une révolution?

Il est nécessaire de comprendre ce qu'est une blockchain, un smart contract et comment ces technologies s'intègrent dans les SI

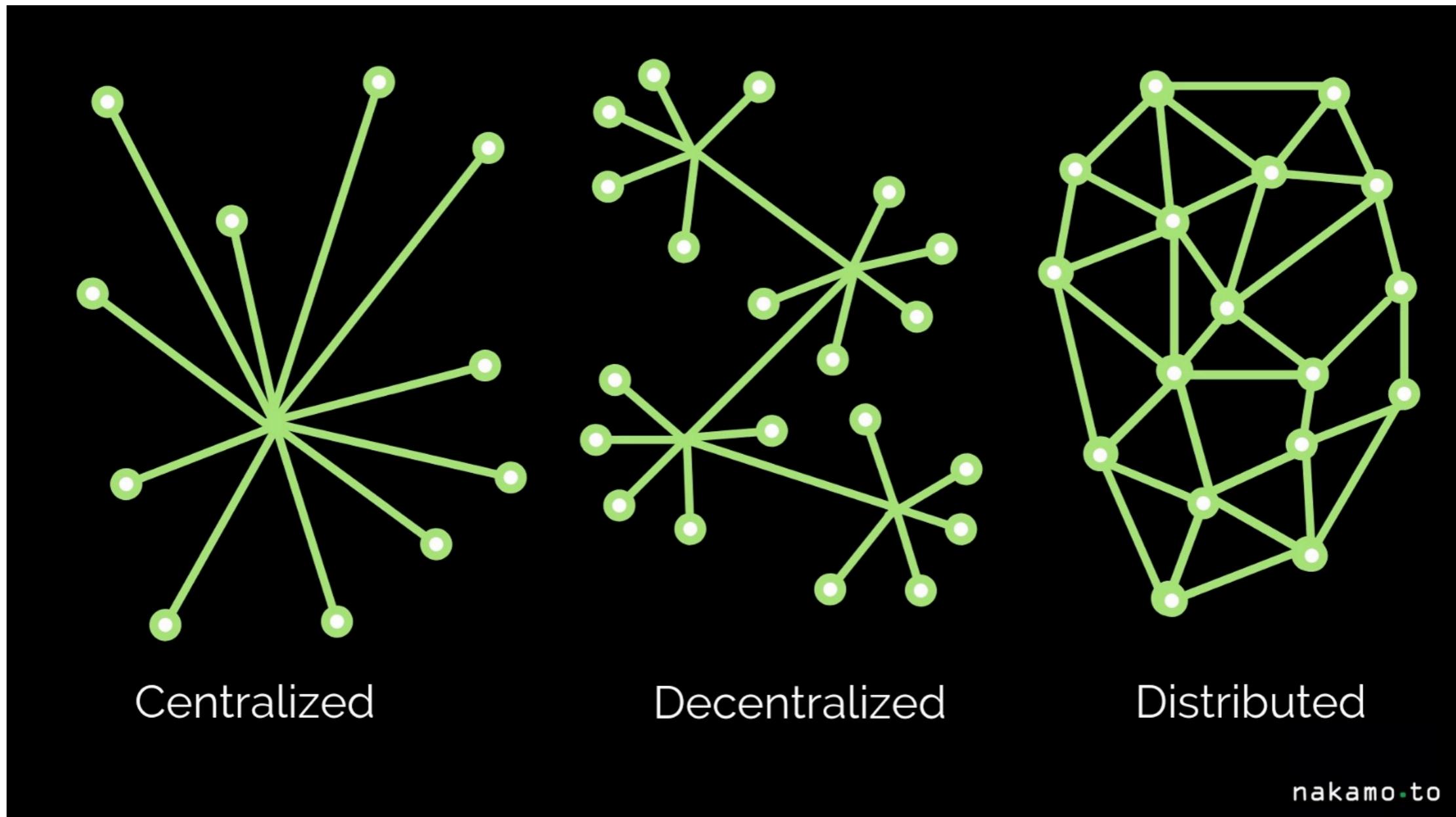
Quelles sont les nouvelles opportunités ?

Quels sont les nouveaux risques?



# SYSTÈMES RÉPARTIS

## TOPOLOGIES



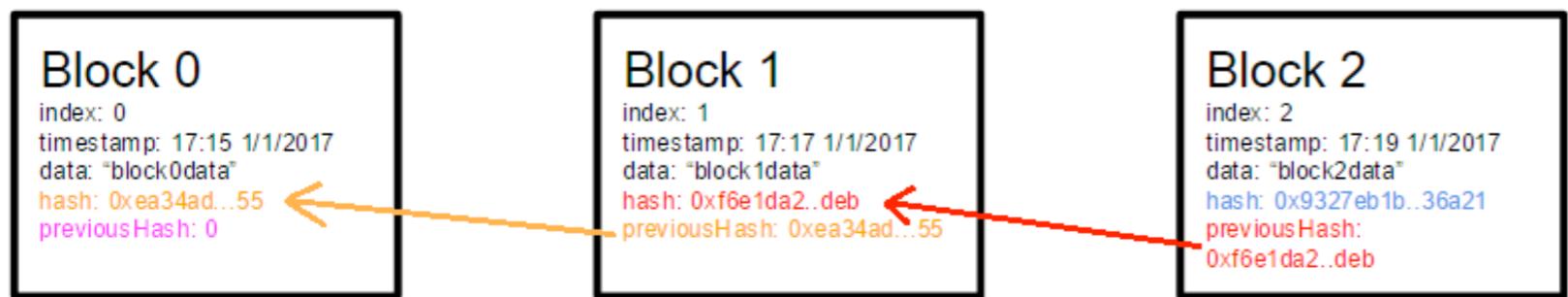
# **SYSTÈMES DISTRIBUÉS**

**QUEL SERVICE PEUT RÉSISTER À**

- un câble fibre optique atlantique coupé
- une législation aggressive US ou Européenne ou Française
- un arrêt d'exportation de composants hardware par son pays producteur
- une attaque de multinationales (GAFA / BATX)
- une attaque/hack d'un gouvernement étranger hostile
- un gouvernement autoritaire contre sa population
- une prise d'intérêt d'un actionnaires majoritaires du système

# BLOCKCHAIN

## CHAIN DE BLOCK



# BLOCKCHAIN

## QUELLE INNOVATION?

- Un registre distribué ❌
  - Comme Redis, Git, ...
- Un registre immutable ❌
  - Seules les opérations & code sont immutables
  - Peut s'obtenir par configuration sur une BDD



# BLOCKCHAIN

## DÉFINITION

- L'invention de la rareté numérique ✨
- Un système résistant à la censure... 🤔
  - BitTorrent aussi
- ... et garantissant l'intégrité du registre grâce à un modèle économique ✨
  - La révolution Bitcoin!



# CENSURE != MODÉRATION

## QU'EST-CE QUE LA CENSURE?

N'importe qui peut créer une adresse sur une blockchain publique et ainsi envoyer ou recevoir des actifs, sans permission spécifique (contrairement à VISA ou AWS).

Aucune autorité centrale ne peut fermer l'adresse ou prendre le contrôle des actifs sans posséder la clé privée liée; ni même interdire de déployer un service sur une blockchain; et de l'utiliser en respectant les règles d'utilisation définient par l'algorithme du service.

C'est la résistance à la censure (en réalité variable selon les blockchains en fonction de leur propriétés de décentralisation).

Cependant rien n'empêche de mettre des règles de modération ou d'accès à un service sur blockchain, c'est de la modération!

# DÉCENTRALISATION

## UN MOT VALISE AUX MULTIPLES FACETTES

- Décentralisation géographiques des nœuds
- Décentralisation des personnes/entités qui possèdent les nœuds logiciel
- Décentralisation des personnes/entités qui possèdent les hardware sur lequel tourne les nœuds
- Décentralisation de la typologie hardware : quelle diversité, quelle facilité à l'obtenir, à quel prix, quel coût à opérer
- Décentralisation du capital : combien de personnes possèdent des tokens, quelle est la capitalisation médiane, quel écart entre les plus 'riche' et les plus 'pauvres'
- Décentralisation de la capacité à dégrader la chaîne : quelle est la Supérminorité aka coefficient de Nakamoto
- Décentralisation du capital initial

# **DES BLOCKCHAINS**

## **UN DOMAINE EN ÉVOLUTION**

**Bitcoin est une technologies qui apporte un use case : une monnaie résistante à la censure et programmable**

**Ethereum est une plateforme de computing qui réutilise la technologie blockchain pour créer des programmes résistants à la censure**

**Beaucoup d'autres ont suivi, soit pour apporter quelque chose de nouveau:**

- Tezos a démontré que le PoS pouvait être aussi décentralisé que le PoW, est la première chain avec auto-amendement et staking liquide
- Monero est une monnaie programmable anonyme

**Soit pour améliorer un problème mis en avant par les technologie précédentes:**

- Avalanche est une blockchain EVM plus rapide grâce à un consensus statistique
- Near est une chaine plus rapide grâce à l'intégration du sharding de données

# BLOCKCHAIN

## QU'EST-CE?

- Un système de calcul,
  - Comme un VPS ou le Cloud PaaS
- Déterministe
  - Code is Law
- résistant à la censure
  - Code is Law



# SMART CONTRACT

## QU'EST-CE?

- Code stocké dans une blockchain
- Collection d'opérations (entrypoints/functions)
- Collection de données (storage/state)
- Immutable une fois déployé

Smart contract

Parameter Type

Storage Type

Set of instructions

# AVANTAGES INTRASÈQUES

RÉSISTANCE À LA SÉGRÉGATION (MONNAIE)

38% de la population mondiale n'a pas de compte bancaire

Selon les régimes, les services bancaires peuvent être refusés sur base d'opinion politique, pour la nature du business, pour des raisons ethniques, de nationalité, de genre, d'orientation sexuelle, ...

... qui sait à quoi ressemblera son gouvernement dans 10 ans ?

# AVANTAGES INTRASÈQUES

## PAIEMENTS (MONNAIE)

### Internationaux:

- frais inférieurs aux frais de change
- pas de contrôle des changes

### Shopping:

- pas de tiers qui détient vos informations (meilleure sécurité)
- pas de métadonnées (meilleure privacy)

### Limites :

- finalité: les paiements ont des latences pouvant atteindre plusieurs minutes
- Micro-paiements: Bitcoin & Ethereum ont des frais trop élevé pour le permettre... mais ce n'est pas le cas de toute chaîne

# AVANTAGES INTRASÈQUES

## ÉCHANGES ANONYMES

**Les principales blockchains sont pseudonymisées**

- Les transactions & métadonnées sont publiques
- L'entrée en crypto par un échange centralisé requiert un KYC (know your customer = identification)
- Si on connaît l'adresse de quelqu'un ...

**Mais certaines garantissent l'anonymat: Monero, ZCash**

- Ce qui requiert un ramp up en P2P

**Certains protocoles « mixer » ou « sappling » permettent des transactions anonymes sur une chaîne pseudonymes:**

- Tezos Sappling
- Ethereum Tornado : attention ce protocole est censuré sur 70% des blocs  
<https://www.mevwatch.info/>

# AVANTAGES INTRASÈQUES

## IDENTIFICATION VIA WALLET

**Il est facile de prouver qu'on est le propriétaire d'une adresse (identifiant unique)**

**Sans jamais révéler d'information personnelle au service**

**Meilleure « Privacy »  
... et destruction de l'économie de l'attention!**

# AVANTAGES INTRASÈQUES

Tokenization

**Standards de données = Interopérabilité**

- ERC20/721/1155 sur EVM
- FA1.2/2/2.1 sur Tezos

**Liquide et non custodial**

**Peut représenter n'importe quel actif**

- Fongible : monnaie, action, item dans un MMO, ...
- Non fongible : art, collectible, foncier, item unique dans un MMO, ...
- Semi-fongible : personnage dans un MMO

**Limite : cadre légal dépendant de la géographie et en évolution rapide**

# AVANTAGES INTRASÈQUES

Decentralized Autonomous Organisation (DAO)

Pas d'autorité centrale qui peut changer les règles à son grés

Mais des règles immuables dans le code

Ou pouvant évoluer démocratiquement (self amendment Tezos)

Dans les DAO, les tokens sont de droits de votes!

# AVANTAGES OPPORTUNISTES

Un meilleur cloud

Les smart contracts sont des Function as a Services (FaaS), disposant d'un espace de stockage (DBaaS), disponibles 24/7, géo-réparti

Mieux résistant à la plupart des attaques conventionnelles

Les transactions sont un moyen d'enregistrer des preuves immuables

Les frais sont payés par l'utilisateur du service (vs le fournisseur du services dans le cloud)

Frais = paiement du run (gas), du stockage (burn) et prime à la rapidité d'execution (bonus pour le validateur du bloc)

Tout le monde peut observer ce qui se passe dans une chaîne : il est facile de créer des outils de statistiques ou de supervision performants

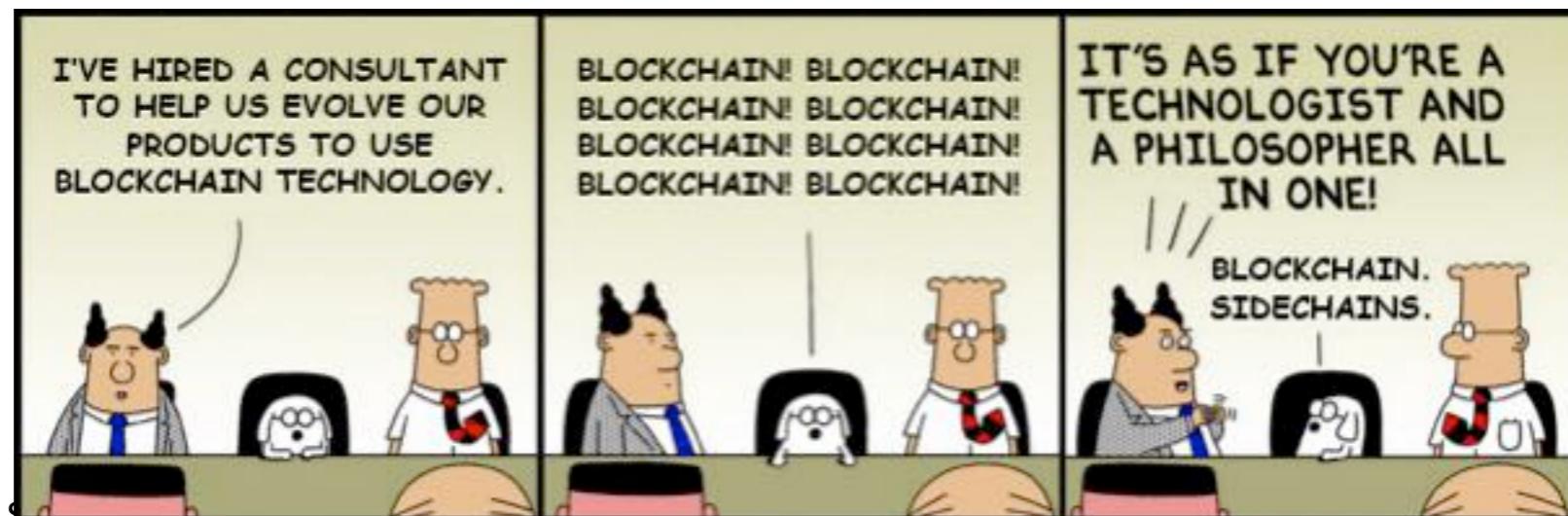
# AVANTAGES CONTINGENT

Un momentum d'opportunités

Les DSI s'intéressent aux blockchains pour:

- Ne pas rater le coche
- Comprendre la technologie

Les projets de blockchains privées ou hybrides ne sont pas résistants à la censure  
Ce sont néanmoins des projets importants pour la diffusion et l'appropriation



# UN DOMAINE NOUVEAU

## PROPICE AUX PROPAGANDES

Les smart contracts sont des Function as a Services (FaaS), disposant d'un espace de stockage (DBaaS), disponibles 24/7, géo-réparti

Mieux résistant à la plupart des attaques conventionnelles

Les transactions sont un moyen d'enregistrer des preuves immuables

Les frais sont payés par l'utilisateur du service (vs le fournisseur du services dans le cloud)

Frais = paiement du run (gas), du stockage (burn) et prime à la rapidité d'execution (bonus pour le validateur du bloc)

Tout le monde peut observer ce qui se passe dans une chaîne : il est facile de créer des outils de statistiques ou de supervision performants

# TAKE AWAY

## BLOCKCHAIN

Une innovation technologique qui apporte de nombreuses innovations d'usage

La décentralisation amène de nouvelles questions économiques, politiques et éthiques

Ceci est propice à la propagande des différentes parties prenantes (insiders/outsiders/central organizations)

Mais offre de nombreuses opportunités pour les futurs SI



# TEZOS

## SPÉCIFICITÉS

Lancée en 2017

Un processus d'auto-amendement du protocol piloté par une gouvernance onchain permet d'éviter les hardfork

Liquid Proof of Stake

Michelson VM, prouvée en Coq

Token natif \$XTZ (prononcé « tez »)

Standards FA1.2 (FT) & FA2 (multi-assets)



# JSLIGO

## HELLO WORLD

**Langage de smart contrat inspiré par Typescript qui compile vers Michelson**

```
type storage = string;
type parameter = unit ;
type return_ = [list<operation>, storage];

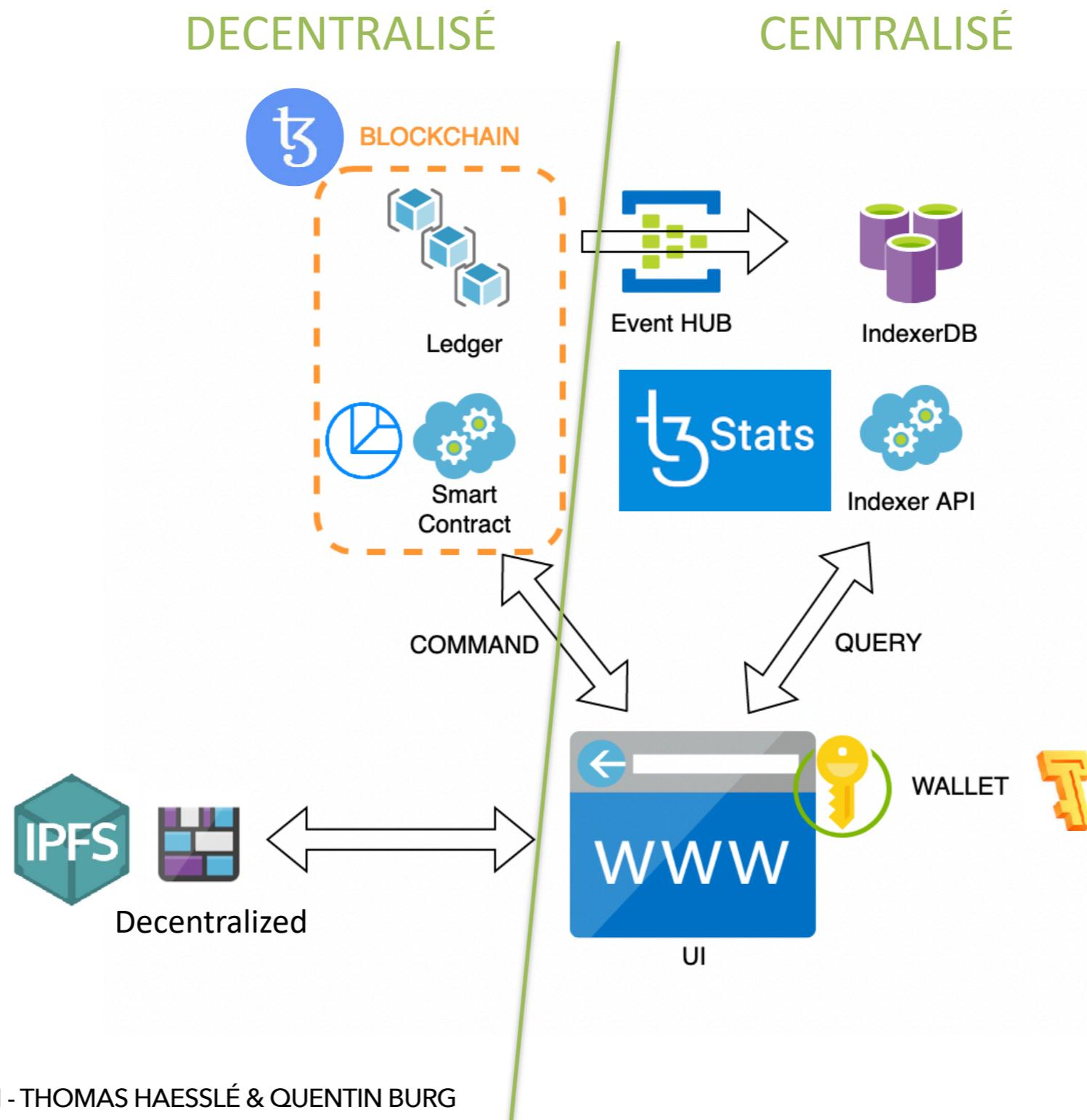
const main = (action: parameter, store: storage) : return_ =>
  [list([]), "Hello Tezos!"];
```

# JSLIGO

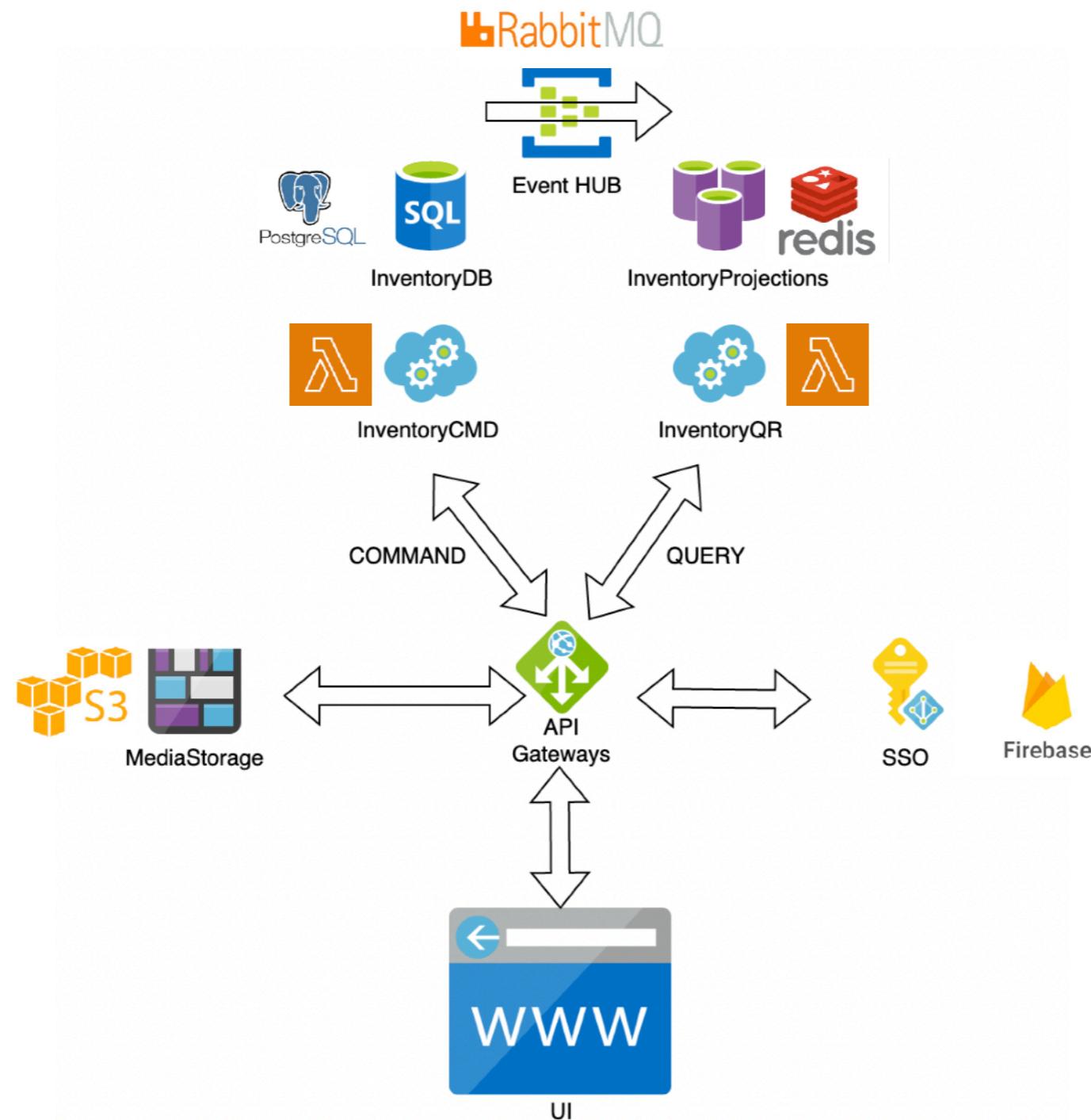
## MOVEMENTS DE LINK

```
type direction = {kind : "North" } | {kind:"East"} | {kind:"South"} | {kind:"West"};  
type parameter = {kind: "Face", direction: direction}  
| {kind: "Start"}  
| {kind:"Stop"};  
  
type state = {kind : "Idle" } | {kind:"Moving"};  
type storage =  
{  
    state: state,  
    orientation: direction  
};  
  
type return_ = [list <operation>, storage];  
  
const main = (action: parameter, store: storage) : return_ => {  
    const noop = list([]);  
    switch(action.kind){  
        case "Face": return [noop, {state: {kind : "Idle"}, orientation: : action.direction}];  
        case "Start": return [noop, {...store, state: {kind:"Moving"} }];  
        case "Stop": return [noop, {...store, state: {kind : "Idle"} }];  
    }  
};
```

# ARCHITECTURE MICROSERVICES



# ARCHITECTURE WEB3



**STAR LORDS KATA**

# TRAVAUX PRATIQUES

## LIENS

GROUPE 1 (Thomas) [https://classroom.github.com/a/1bVc\\_x07](https://classroom.github.com/a/1bVc_x07)

GROUPE 2 (Quentin) [https://classroom.github.com/a/VZ-F4E\\_R](https://classroom.github.com/a/VZ-F4E_R)