

# 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 white walls and blue railings. Stairs connect different levels, and bookshelves are filled with books. A person is walking down a staircase on the left, and another person is sitting on a blue sofa in the foreground. 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 2024

## PRÉVISIONNEL

9/01 : Intro & Property based Testing - Troll of Fame Kata (TypeScript)

16/01 : Quête des machines à états - Tennis Kata (TypeScript)

23/01 : Frontend dataflow - Front Kata (React)

30/01 : Intro Blockchain & Smart contracts - Contract Kata (JsLigo)

6/02 : Gestion des Erreurs & Présentation du projet

20/02 : Cloud et microservice

27/02 : Communication Inter-process

19/03 : Séance projet

# **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

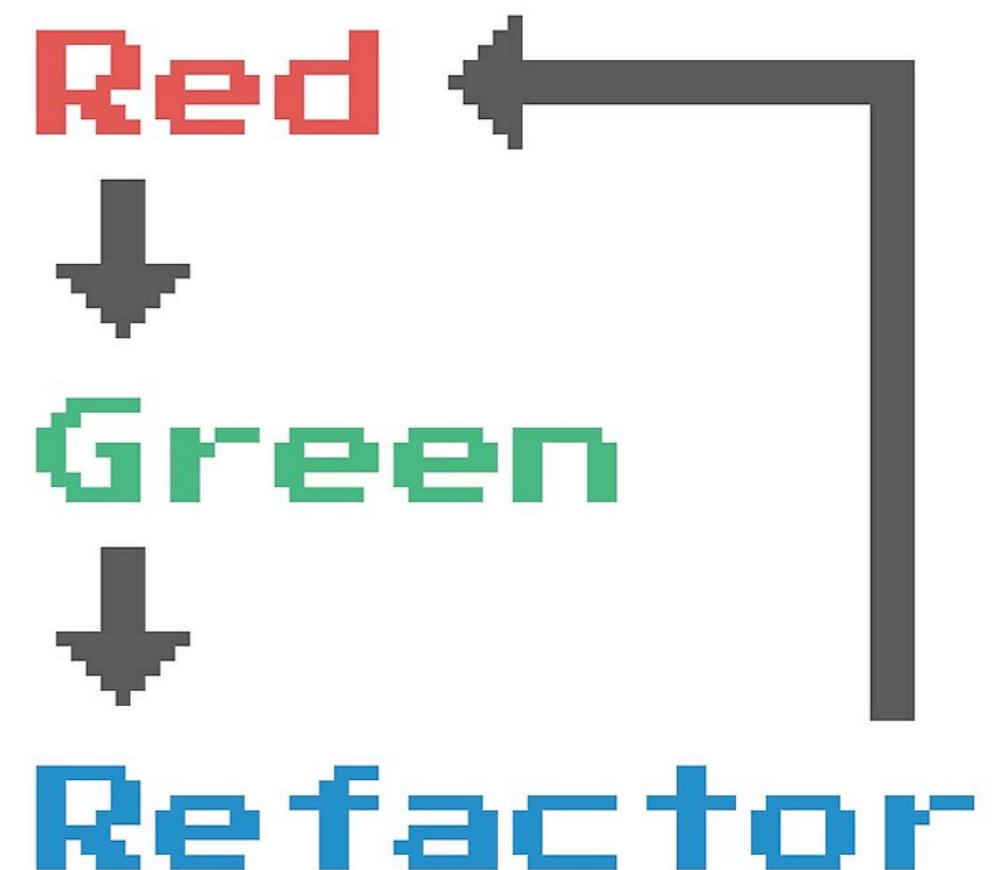
Impact Durabilité

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 2023 ?

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 RÔI  
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 RÔI, 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:

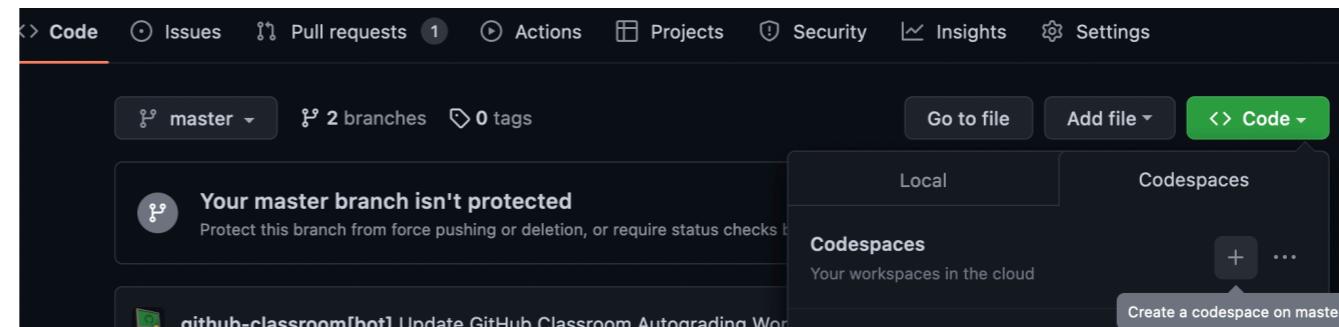
Via vscode + docker plugin (nécessite docker en local)

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/cGRWz5of>

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

# 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 !!!

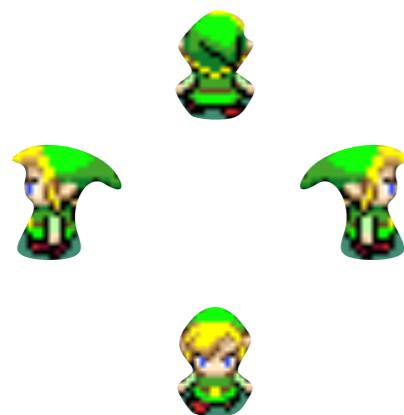
Peut-on améliorer la conception logicielle,  
puis SI pour éviter cela ?

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

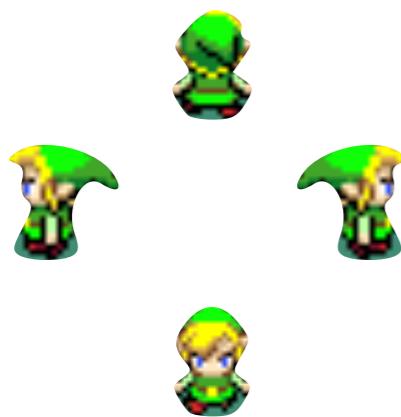
-35 n'est pas une Direction valide ... mais les valeurs d'un enum sont en réalité des numbers

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

LoZ OoT MM

OoT OoA OoS

FS WW FSA

TP SS

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

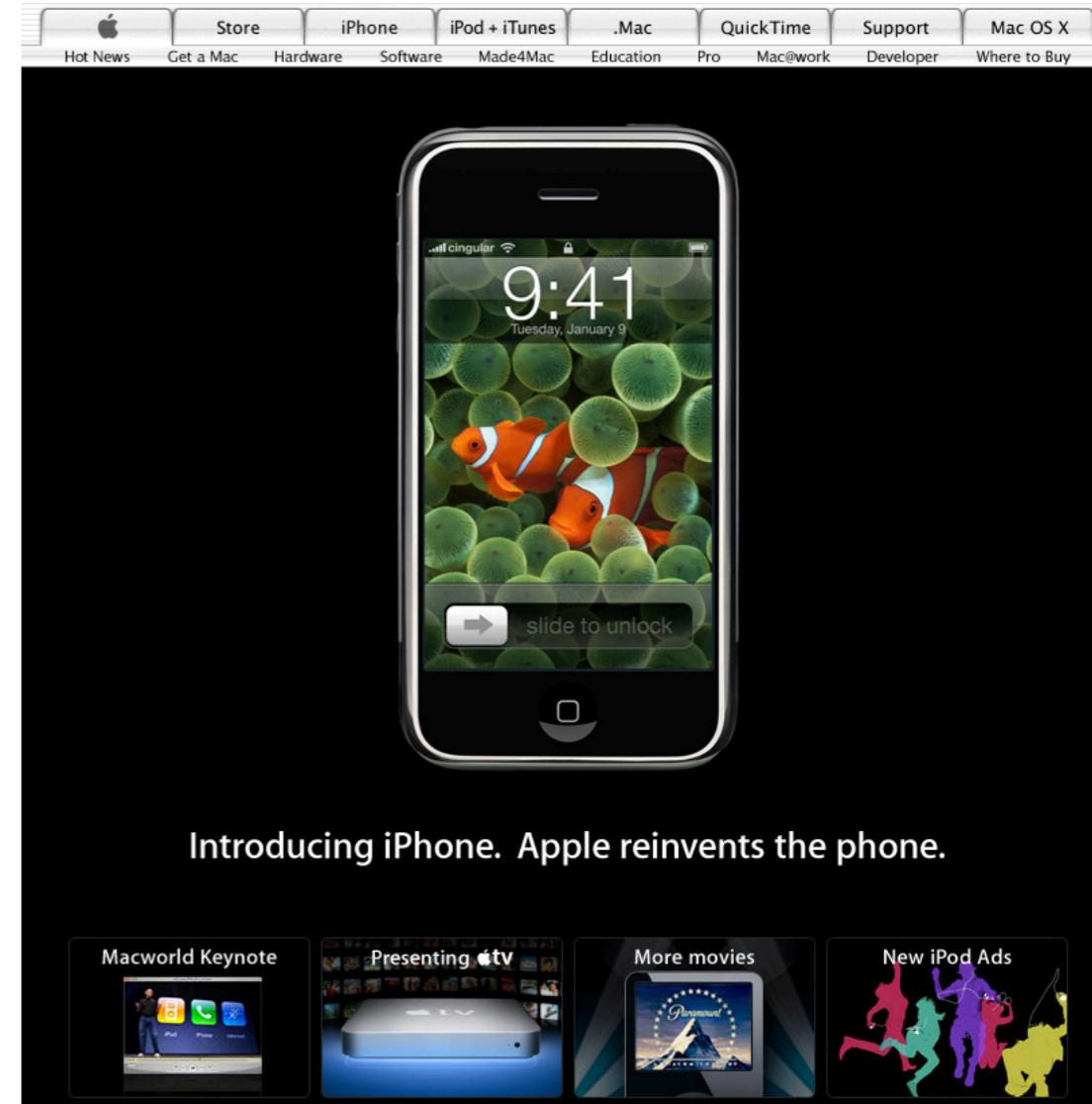
00'S : L'ÈRE DE LAMP

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 ?



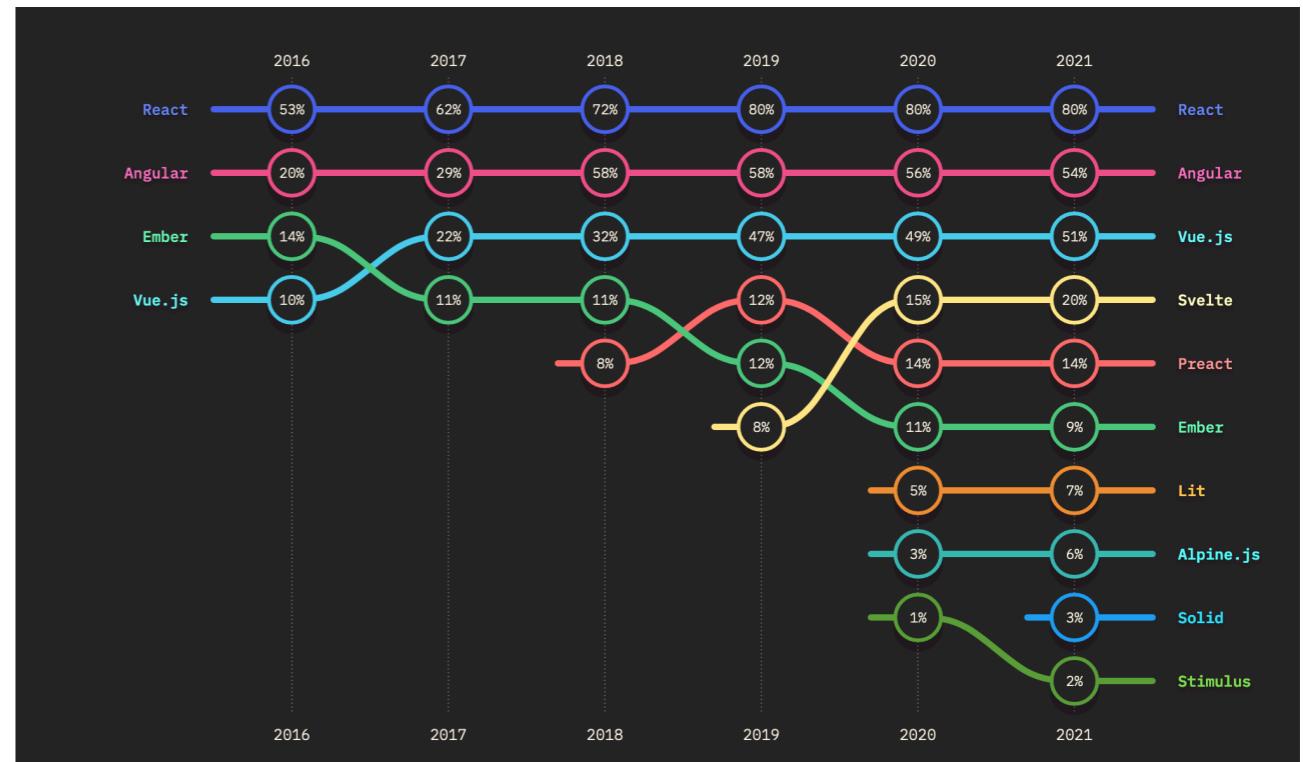
# DEV FRONT END

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

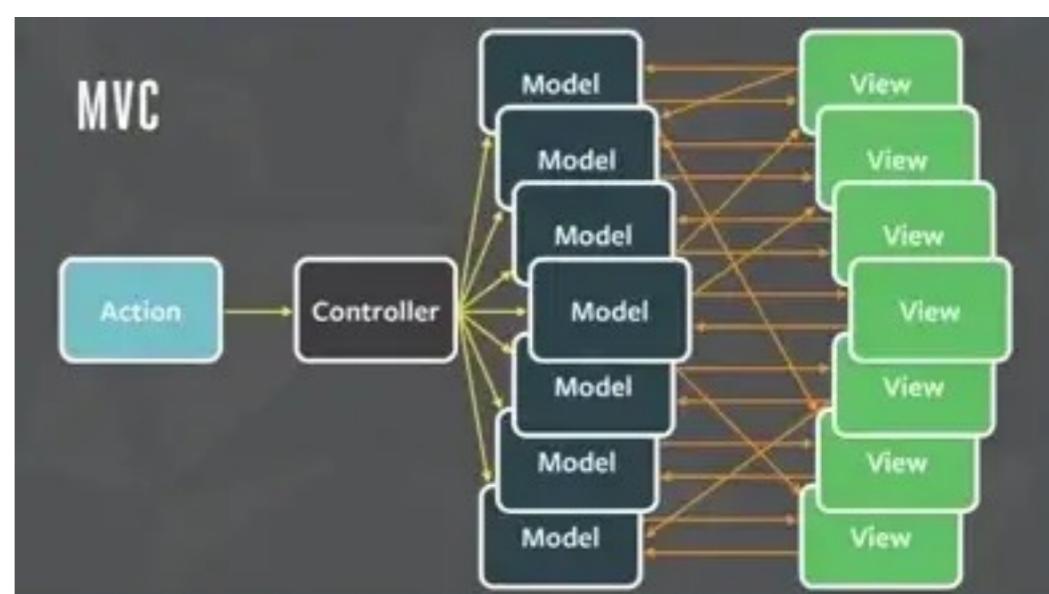
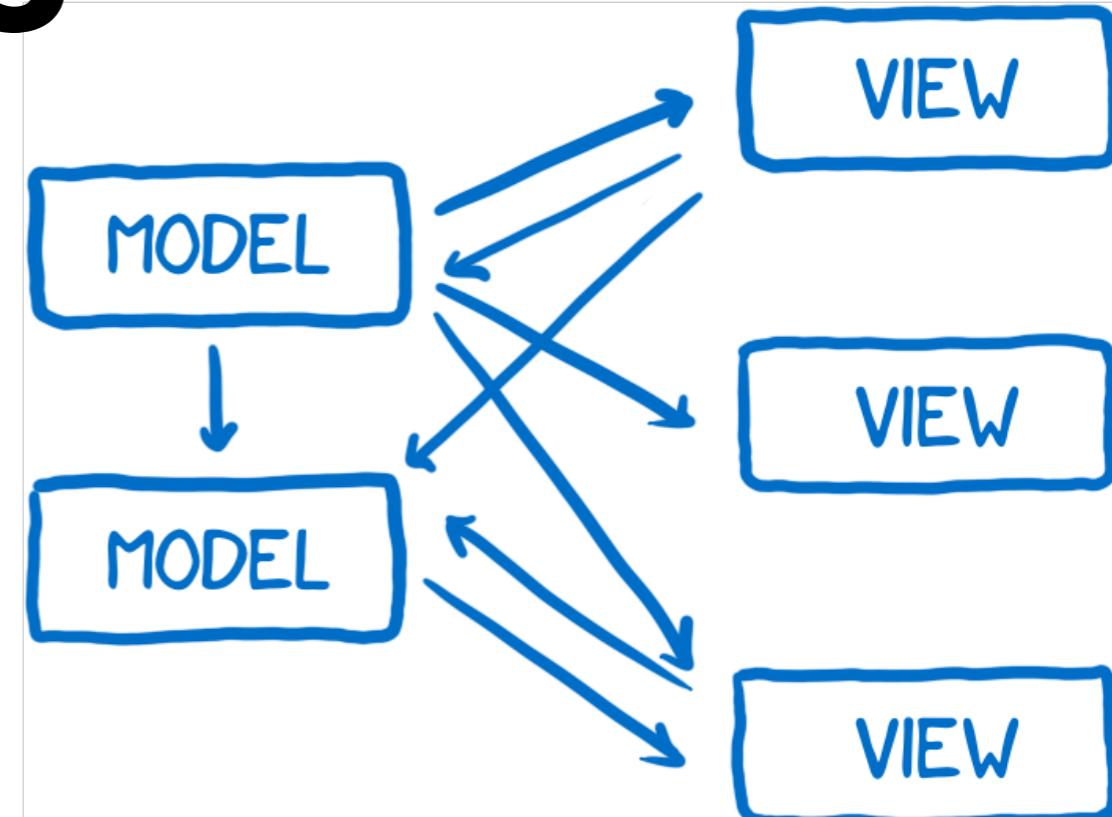
MVC / MVVM CÔTÉ CLIENT

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 🎲



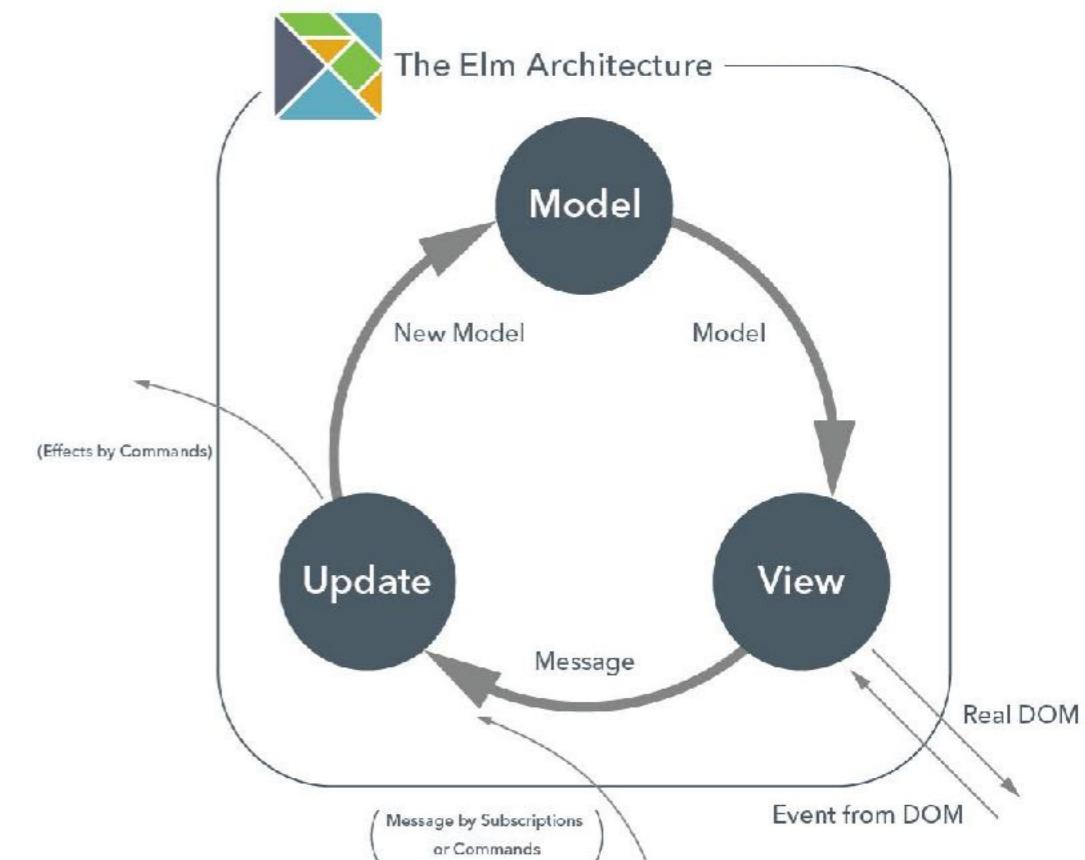
# GESTION D'ÉTATS

## 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

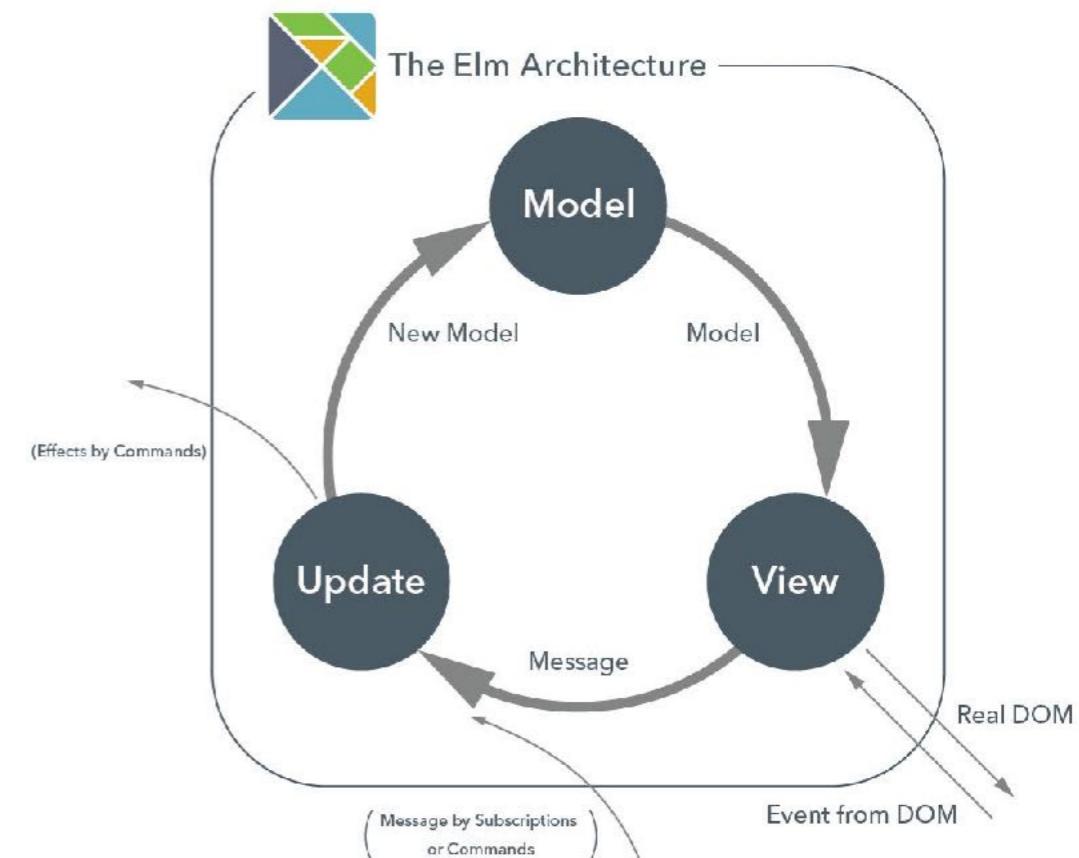


# GESTION D'ÉTATS

## ELM ARCHITECTURE

Implémentations :

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



A inspiré :

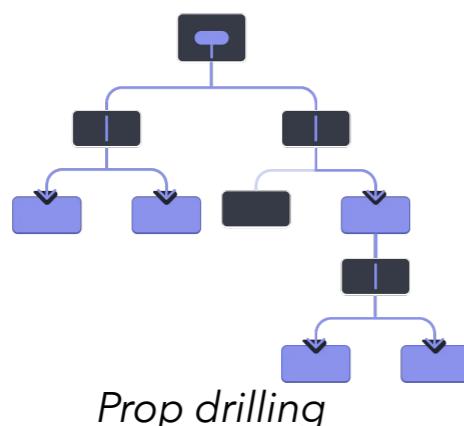
- Redux + \*js
- react `useReducer` hook (au niveau state local)

# 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);

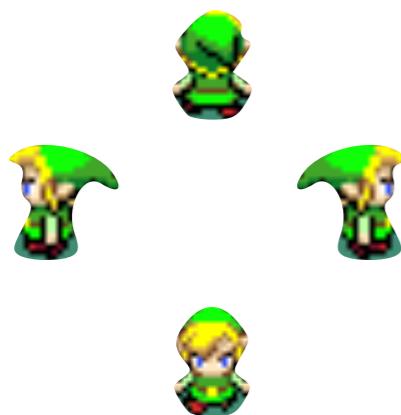
const update = (model, message) => {
  switch (message.type
    // Describe state machine here
  ) {
  }
  return model;
};

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"
    }
}
```

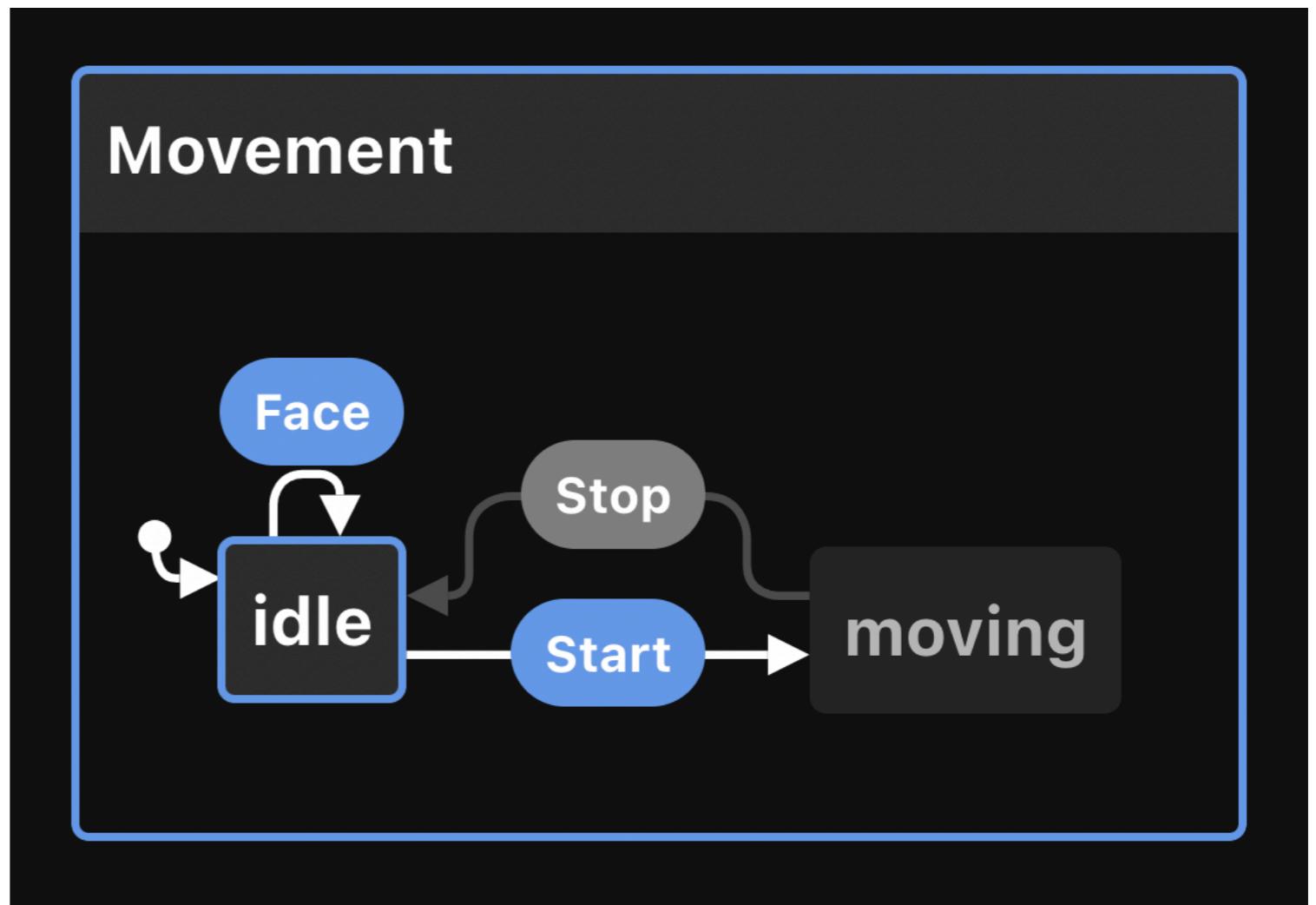


a des « tagged union » qui sont des types OU avec un pattern matching exhaustif

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)
    }
}
```

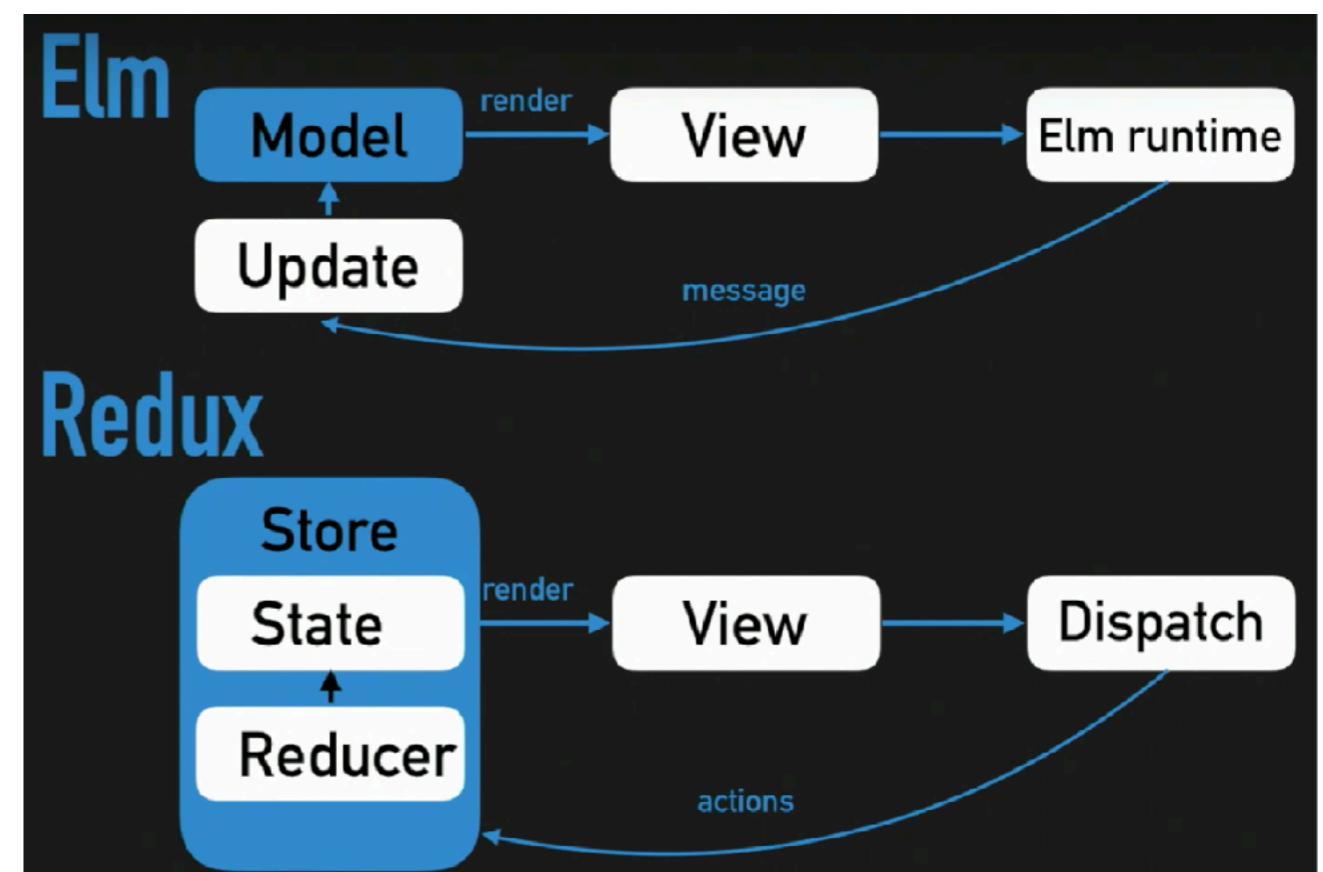
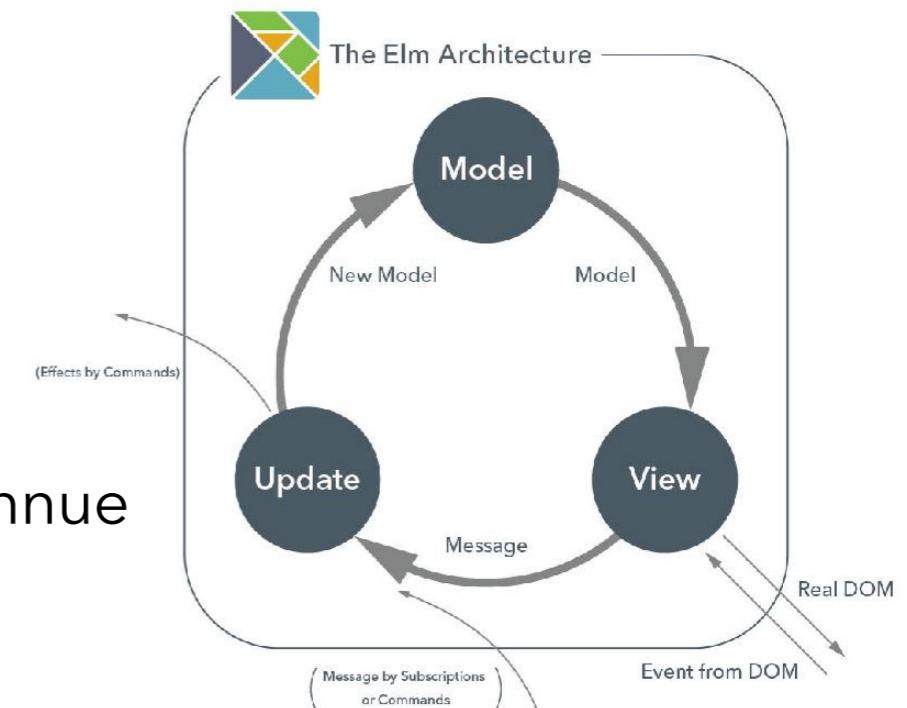
# GESTION D'ÉTATS

## ELM / REDUX

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

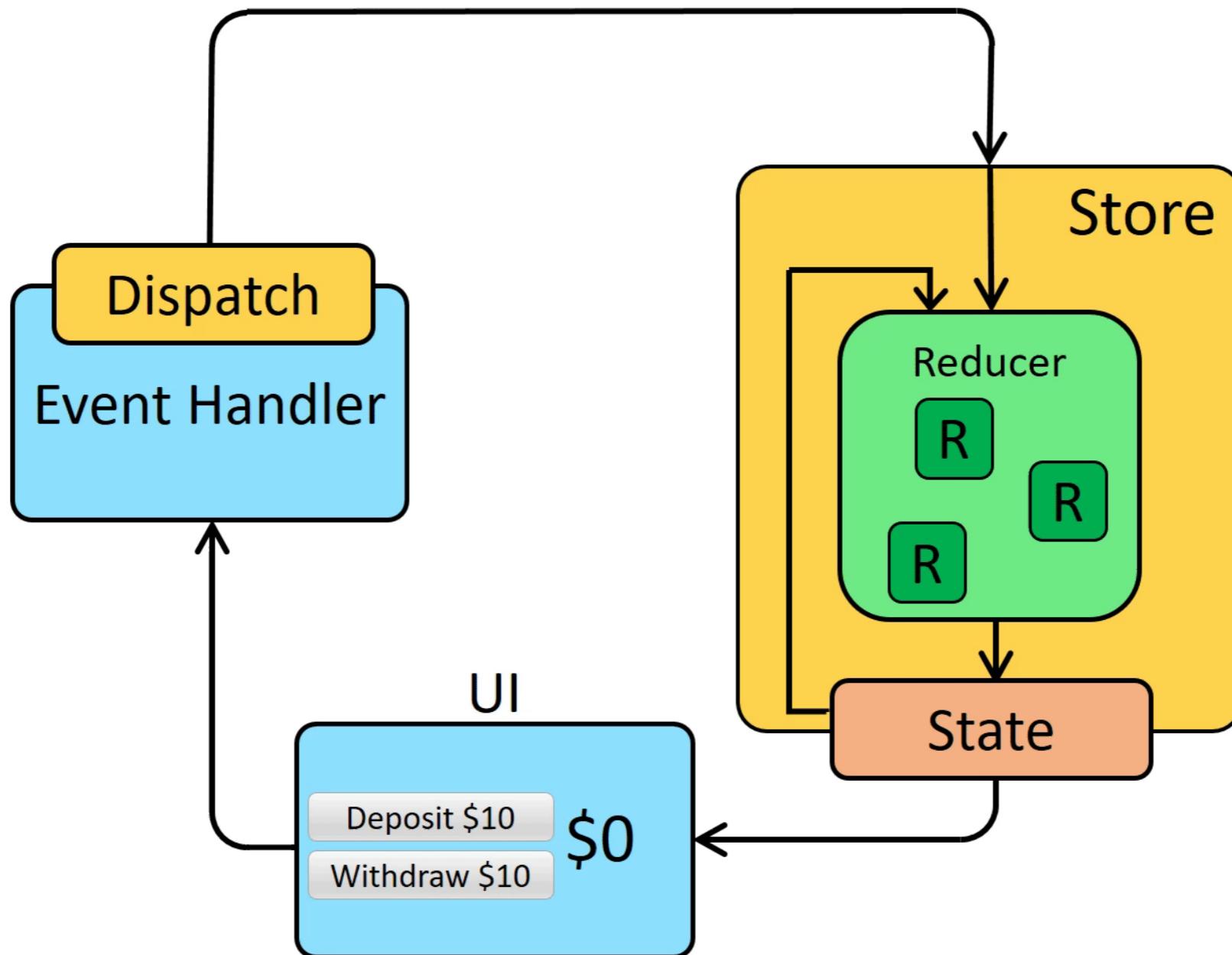
On peut donc également faire le parallèle avec celle-ci :

- 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

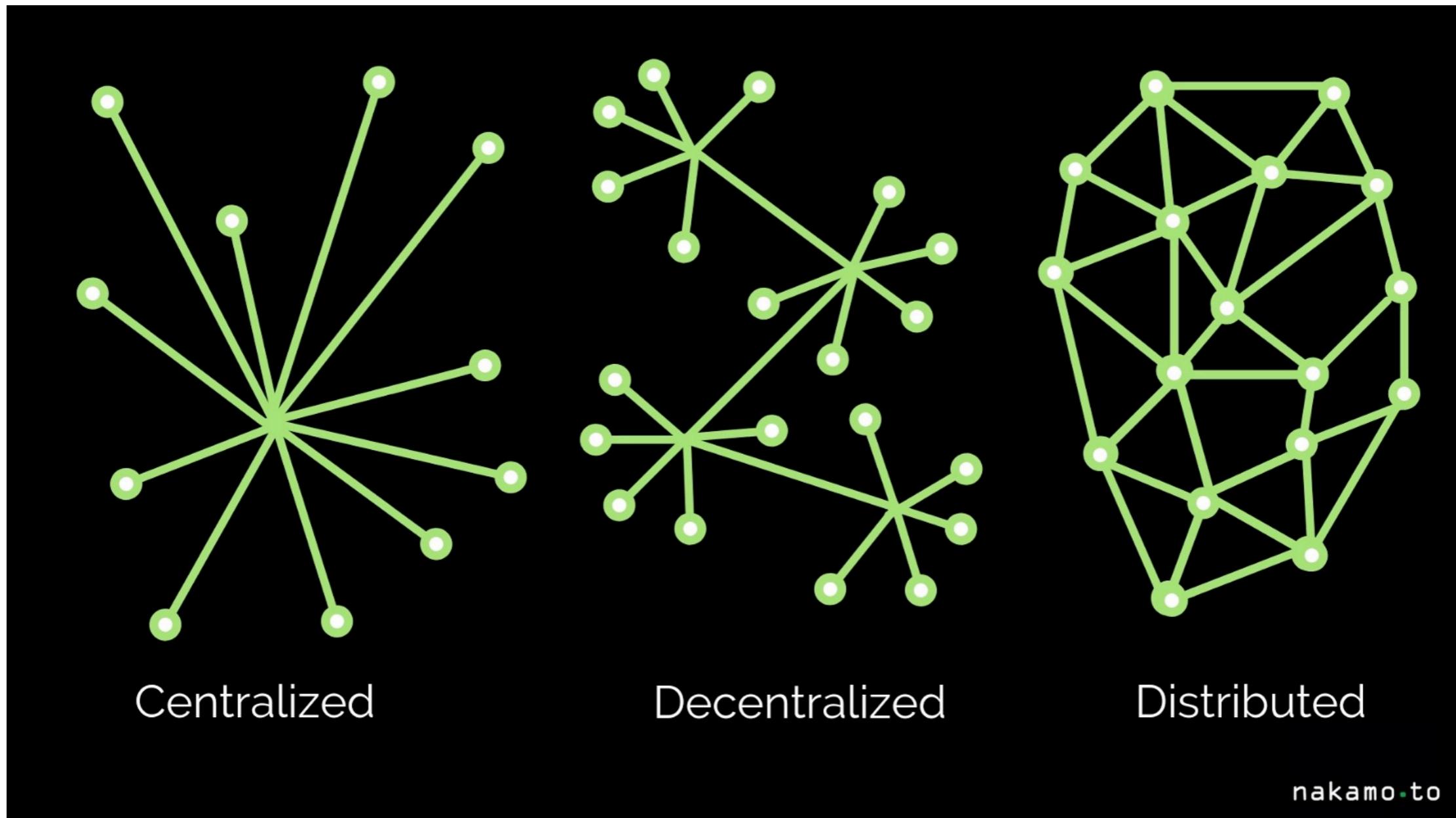
Quelles sont les nouvelles opportunités ?

Quels sont les nouveaux risques?



# SYSTÈMES RÉPARTIS

## TOPOLOGIES



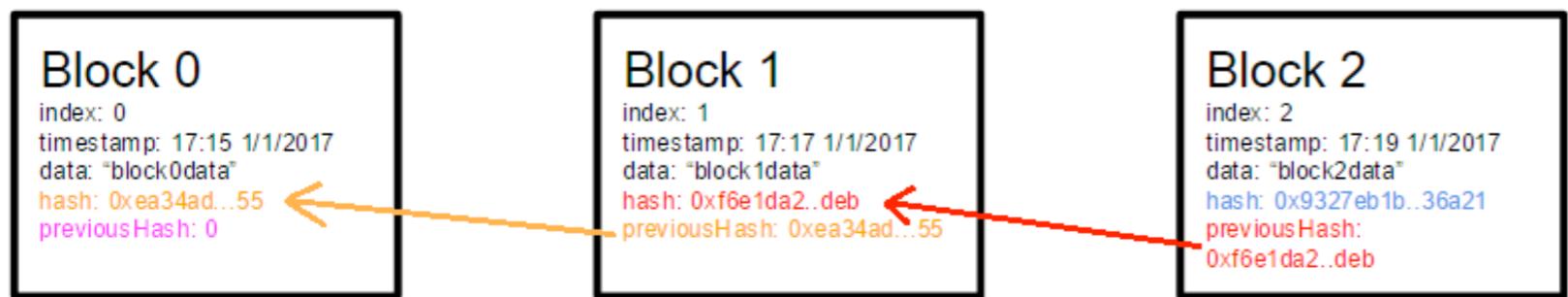
# **SYSTÈMES DISTRIBUÉS**

**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 pseudonyme:**

- 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

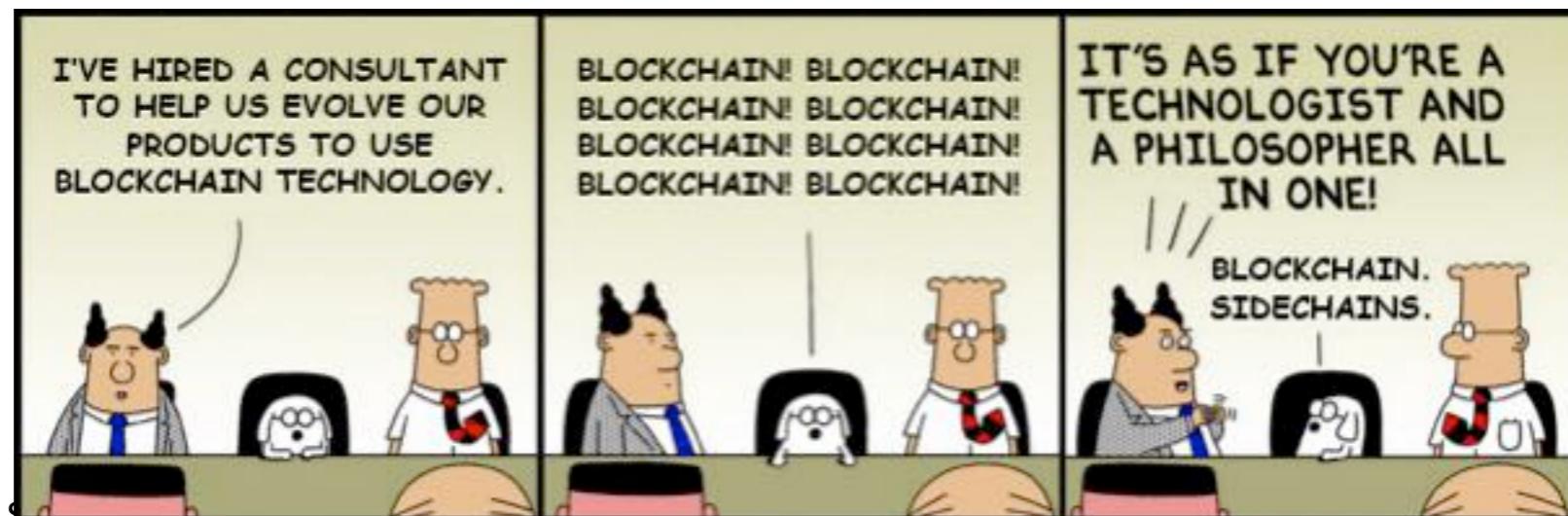
# AVANTAGES CONTINGENT

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)



# CONTRATS

TEZOS

Tout contrat dispose d'un compte de XTZ (account model != UTXO model)

Deux types d'adresses:

- Les contrats implicites, liés à un manager qui possède la clé privée. Le hash de la clé publique correspond à l'adresse, précédé par "tz1" (Ed25519 curve), "tz2" ([Secp256k1](#) curve), or "tz3" (P256 curve), en fonction de la Signature Algorithm's elliptic curve (see [ECDSA](#)). Ces contrats disposent d'un opération "transfert" implicitement
- Les smart contracts crée par une opération "origination". Ils n'ont pas de pair de clés publique/privée. Leur adresse débute par "KT1"

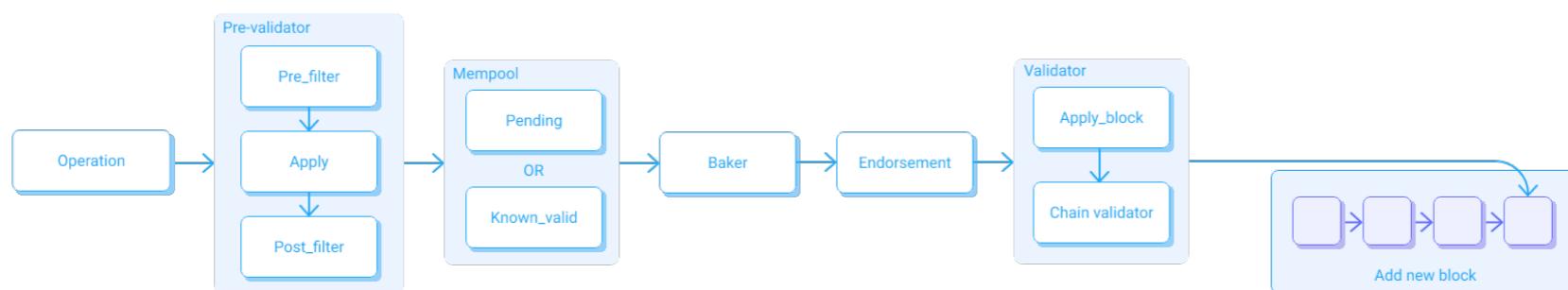
# OPERATIONS

## TEZOS

Une opération, appelée aussi transaction, correspond à un message envoyé à une adresse

```
type operation = {  
    amount: amount; // amount being sent  
    parameters: data list; // parameters passed to the script  
    counter: int; // invoice id to avoid replay attacks  
    destination: contract hash;  
}
```

Avec un workflow spécifique



# 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];

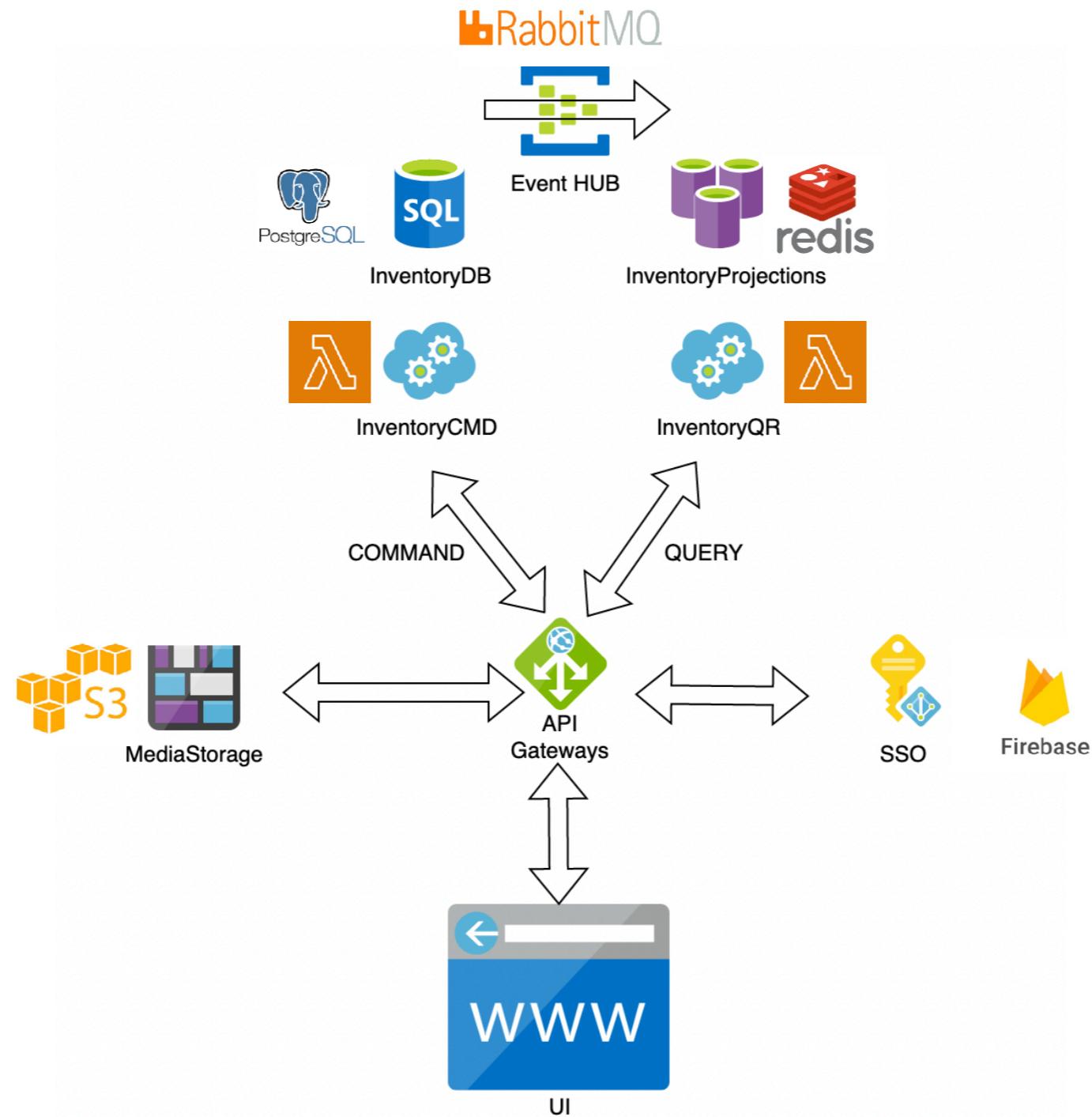
const main = (action: parameter, store: storage) : return_ =>
  [list([]), "Hello Tezos!"];
```

# JSLIGO

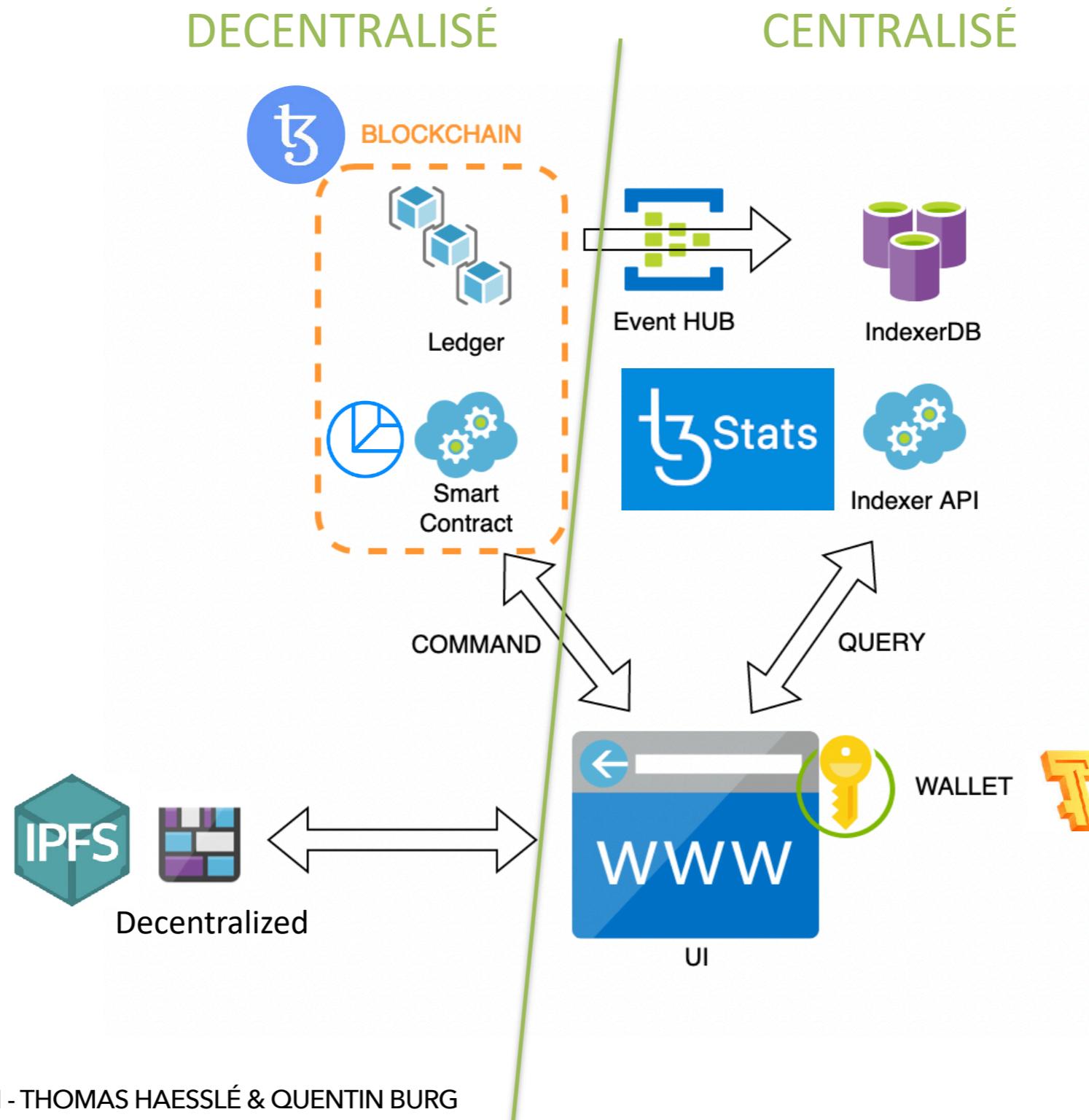
## MOVEMENTS DE LINK

```
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 MICROSERVICE CQRS



# 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)

# THE BOSS

SI VOUS VOULEZ ALLER PLUS LOIN

Blockchain :

 [Bitcoin whitepaper](#) (la base)

 [Grokking Bitcoin](#) (très bon livre pour approfondir)

Tezos :

 [Open Tezos](#) (learning)

 [Ligolang](#)

 [Taqueria](#) (app workflow tooling)



# **QUALITÉ DU SI**

---

**COURS 5 - GESTION D'ERREURS & PRÉSENTATION PROJET**

# THE QUEST

## Gestion d'erreurs

### ⚠ JAVA Inside

Les crash au **RUNTIME** sont très coûteux ...

*... s'ils impactent le système : certains langages dynamiques ont une stratégie du laisser planter (Erlang/Elixir), ou d'une manière générale les implémentations du modèle acteur (Akka, Actix, ...) ... hors scope du cours*

Que représente une erreur ?

- ➡ Une absence de valeur
- ➡ Un chemin alternatif dans le flux d'exécution
- ➡ Une exception au runtime



# SPÉCIFICATION

## TIRER À L'ARC SUR UN MONSTRE

1. Armer son arc d'une flèche
2. Viser le monstre
3. Tirer sur une flèche et blesser l'ennemi



# EXCEPTIONS

## THROWS ARE UGLY GOTOS

```
class Weapon {
    Weapon(){
        throw new RuntimeException("NoMoreArrow");
    }
}
class Target {
    Target(){
        throw new RuntimeException("TooMuchFog");
    }
}
class Impacted{}

public class Main {
    Weapon armYouBow = new Weapon();
    Target targetMonster = new Target();
    Impacted hitMonster(Weapon w, Target t) {
        return new Impacted();
    }
}
```

*Exemples en Java*

# EXCEPTIONS

THROW ARE UGLY GOTOS LOST IN TIME OF ASYNC / FUTURE

```
import java.util.concurrent.*;

class Weapon {
    Weapon() { throw new RuntimeException("NoMoreArrow"); }
}

class Target {
    Target() { throw new RuntimeException("TooMuchFog"); }
}

class Impacted {}

public class Main {
    static Impacted hitMonster(Weapon w, Target t) {
        return new Impacted();
    }

    public static FutureTask<Void> attack = new FutureTask<>(new Runnable() {
        @Override
        public void run() {

            Weapon armYouBow = new Weapon();
            Target targetMonster = new Target();
            hitMonster(armYouBow, targetMonster);

        }
    }, (Void) null);

    public static void main(String[] args) {
        ExecutorService es = Executors.newSingleThreadExecutor();
        es.execute(attack);
        es.shutdown();
    }
}
```

# EXCEPTIONS

TRY / CATCH SONT LES RACINES DU MAL

```
class Weapon {  
    Weapon() {  
        throw new RuntimeException("NoMoreArrow");  
    }  
}  
public class Main {  
    try {  
        doSomethingThatMayThrow();  
        Weapon armYouBow = new Weapon();  
    } catch (Exception e) {  
    }  
}
```

On casse aussi la hiérarchie d'héritage en OOP Fragile Base Classe problem

# TAKE AWAY

MAUVAIS CHOIX POUR ...

Modéliser l'absence de valeur

Modéliser une erreur fonctionnelle

Modéliser des erreurs asynchrones

OK SI ...

Vous n'espérez pas que quelqu'un les « catch »

Vous voulez semer le chaos sur Hyrule

Vous savez ce que vous faites (contributeur VM)



# TAKE TAKE AWAY

SÉRIEUSEMENT, QUAND LANCER UNE EXCEPTION ?

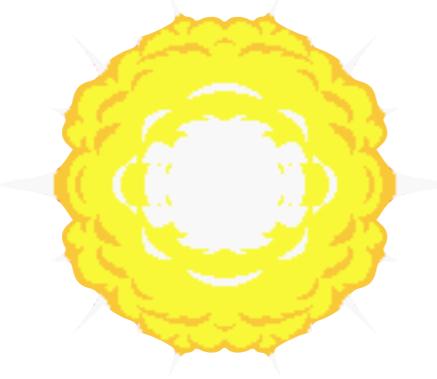
Quand vous voulez faire « crasher » le programme, par exemple :

- La configuration minimum n'est pas fournie au démarrage
- Une erreur n'est pas récupérable par le système et nécessite une intervention humaine (*Les **Error** de Java ... donc si vous écrivez une VM*)



# MODÉLISER UNE ABSENCE POTENTIELLE DE VALEUR

## UNE VALEUR POUR SEMER LA DÉSOLATION : NULL



Exception in thread "main" java.lang.NullPointerException

```
class Weapon {}
class Target {}
class Impacted {}

public class Main {
    static Weapon armYouBow = null;
    static Target targetMonster = null;
    static Impacted hitMonster(Weapon w, Target t) {
        return null;
    }
    public static void main(String[] args) {
        hitMonster(armYouBow, targetMonster).toString();
    }
}
```



Qui accepterait de travailler avec un langage qui autorise la compilation de ce programme? Sérieusement?

OCaml, Rust, Haskell : null n'existe pas

Kotlin, Swift, F# : n'autorisent null que si explicitement autorisé à la déclaration

TS (bien configuré) : n'autorise null que si explicitement typé

# MODÉLISER UNE ABSENCE POTENTIELLE DE VALEUR

## OPTION EN JAVA

```
class Weapon {}
class Target {}
class Impacted{}

import java.util.Optional;
class Weapon {}
class Target {}
class Impacted{}

Optional<Weapon> armYouBow = Optional.empty();
Optional<Target> targetMonster = Optional.empty();
Optional<Impacted> hitMonster(Optional<Weapon> w,
Optional<Target> t) {
    return Optional.empty();
}

.
```

**Optional<A>** est un type à décrire l'existence ou non d'une valeur!

**None<A>** n'a pas de valeur pour le type Option<A>. Ce type a exactement 1 valeur pour l'ensemble A donné.

Vous pouvez vous prémunir en utilisant `java.util.Objects.requireNonNull` dans vos constructeurs

**Some<A>** représente une valeur de type A, ce type a autant de valeurs que l'ensemble A

# MODÉLISER UNE ABSENCE POTENTIELLE DE VALEUR

## TRAITER LES VALEURS OPTIONNELLES EN JAVA

```
Optional<Impacted> hitMonsterIf(Optional<Weapon> w, Optional<Target> t) {  
    if (w.isPresent() && t.isPresent()) {  
        return Optional.of(new Impacted());  
    }  
}  
  
Optional<Impacted> hitMonsterflatMap(Optional<Weapon> w, Optional<Target> t) {  
    return w.flatMap( sw -> t.map(st -> new Impacted()));  
}
```

Optional n'a pas évolué en sealed interface/record en Java17

La manipulation de Optional est fastidieuse et demande de la vigilance (pas de pattern matching)

**flatmap (aka bind) et map facilitent la manipulation des valeurs optionnelles et garantissent un bon traitement des cas**

# MODÉLISER UNE ABSENCE POTENTIELLE DE VALEUR

## TRAITER LES VALEURS OPTIONNELLES EN TS

```
import * as O from 'fp-ts/Option'
import { Option } from 'fp-ts/Option'
const calculateDamage = (w: Weapon, t: Target): number => 42
const damageMonster = (w: Option<Weapon>, t: Option<Target>): Option<number> =>
O.chain(
  (ow : Weapon) => O.map(
    (ot : Target) =>
      calculateDamage(ow, ot)
  )(t))
(w)
```

chain est flatMap en TS

La librairie fp-ts fournit Option<A>

Dans cet exemple typique l'annotation limitée à <A> (mais O.Option<A>) et O.Option<B>

initialisée) & null (absence de valeur)

map: <A, B>(f: (a: A) => B) => (fa: O.Option<A>) => O.Option<B>

Les manipulation de type nullable ou leur expressivité est perfectible

# TAKE AWAY

## LES OPTION / MAYBE

A utiliser pour modéliser l'absence de valeur

Sécurisant, surtout quand on dispose d'ADT

Dans certains langages Option s'appelle  
Optional (Java) ou Maybe (Scala, Haskell)



# **MODÉLISER UNE ERREUR POTENTIELLE**

**UNE ERREUR?**

**Le traitement réussit à produire une valeur**

**OU**

**Le traitement produit une erreur**

**Les données qui peuvent être en erreurs sont des types SOMME**

# MODÉLISER UNE ERREUR POTENTIELLE

## RESULT

```
type Error<E> = { kind: "Error", error: E }
type Ok<A> = { kind: "Ok", ok: A }
type Result<E, A> = Error<E> | Ok<A>
const isNat = (x: number): Result<string, number> =>
  x > 0
    ? { kind: "Ok", ok: x }
    : { kind: "Error", error: "Not a natural number" }

const foldNumber = (y: Result<string, number>, defaultValue: number): number => {
  switch (y.kind) {
    case "Ok": return y.ok;
    case "Error": return defaultValue;
  }
}
```

Les données qui peuvent être en erreurs sont des types SOMME

# MODÉLISER UNE ERREUR POTENTIELLE

## EITHER = RESULT

```
type Left<E> = { kind: "Left", left: E }
type Right<A> = { kind: "Right", right: A }
type Either<E, A> = Left<E> | Right<A>
const isNat = (x: number): Either<string, number> =>
  x > 0
    ? { kind: "Right", right: x }
    : { kind: "Left", left: "Not a natural number" }

const foldNumber = (y: Either<string, number>, defaultValue: number): number => {
  switch (y.kind) {
    case "Right": return y.right;
    case "Left": return defaultValue;
  }
}
```

Either est en réalité plus générique car représente n'importe quel arbre binaire

Par convention, il est souvent utilisé avec Right (correct) contenant la valeur ok, et Left (abandon) contenant la valeur error

# MODÉLISER UNE ERREUR POTENTIELLE

## AVEC FP-TS

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

const isNat = (x: number): Either<string, number> =>
  x > 0
    ? E.right(x)
    : E.left("Not a natural number")

const foldNumber = (y: Either<string, number>, defaultValue: number): number =>
  pipe(
    y,
    E.match(
      (_: string) => defaultValue, // onLeft handler
      (value: number) => value // onRight handler
    )
  )
```

FP-TS fournit des fonctions utilitaires telles que le match expressif

# MODÉLISER UNE ERREUR POTENTIELLE

## CHAIN (AKA FLATMAP AKA BIND) / MAP

```
type weapon = string
type target = string
type impacted = { impacted: target }

let must_be_carried = (w : target) : Either<string,target> =>
  w === "bow" ? E.right(w) : E.left("not carried")

let hit_monster = (w: Either<string, weapon>, t: Either<string, target>): Either<string, impacted> =>
  pipe(
    w,
    E.chain(
      (_carried) => pipe(
        t,
        E.map(
          (targeted) => ({ impacted: targeted })
        )
      )
    )
  )
)
```

chain: <E, A, B>(f: (a: A) => E.Either<E,B>) => (ma: E.Either<E,A>) => E.Either<E,B>

map: <A, B>(f: (a: A) => B) => (fa: E.Either<E,A>) => E.Either<E,B>

# MODÉLISER UNE ERREUR POTENTIELLE

```
sealed interface Result<T,E> {
    record Ok<T,E>(T ok) implements Result<T,E> {
        public Ok {
            java.util.Objects.requireNonNull(ok);
        }
    }
    record Err<T,E>(E error) implements Result<T,E> {
        public Err {
            java.util.Objects.requireNonNull(error);
        }
    }
    public static<T> Ok ok(T ok){
        return new Ok(ok);
    }
    public static<E> Err err(E error){
        return new Err(error);
    }
}

record Weapon(String name){
    public Weapon {
        java.util.Objects.requireNonNull(name);
    }
    public static Weapon weapon(String name){
        return new Weapon(name);
    }
}

public class Main
{
    public static Result<Weapon, Exception> mustCarryABow(Weapon w){
        return w.name().equals("bow") ? Result.ok(w) : Result.err(new Exception("bow not carried"));
    }
    public static void main(String[] args) {
        var myWeapon = Weapon.weapon("bow");
        switch (mustCarryABow(myWeapon)){
            case Result.Ok<Weapon, Exception> o -> System.out.println("Cool you carry a ".concat(o.ok().name()));
            case Result.Err<Weapon, Exception> e -> System.out.println("Error:".concat(e.error().getMessage()));
        }
    }
}
```

# MODÉLISER UNE ERREUR POTENTIELLE

## EN JAVA

N'existe pas dans la lib standard

Peut être encodé avec les génériques (vous avez vu comment) ... c'est pas compliqué mais il manque quelques fonctions pour être réellement utilisable (pure, map, flatmap, fold)

Si vous voulez faire du Java « PRO » vous pouvez utiliser VAVR  
[https://www.vavr.io/vavr-docs/#\\_either](https://www.vavr.io/vavr-docs/#_either) ...

Ou passez à Scala ou Kotlin + Arrow-Kt

Ou abandonnez la JVM 😈 pour OCaml, Rust, F# ou Haskell

# TAKE AWAY

LES RESULT / EITHER

A utiliser pour modéliser les cas d'erreur

Sécurisant

Facile à manipuler

Parfois encodé avec un seul paramètre, pour homogénéiser les erreurs, s'appelle souvent Try dans ce cas.

```
interface MyDomainErrors extends Error { };
type Try<A> = Either<MyDomainErrors, A>;
```



# MORE

## ACCUMULATEURS D'ERREURS

On souhaite parfois remonter l'ensemble des erreurs rencontrées (formulaire, json, csv, compilation, ...) : nous avons besoin d'accumulateurs d'erreurs !

Problème Either va contenir la première erreur qui sera propagée par chain ou map

On aimeraient quelque chose qui ressemble à

```
type AccumulateErrors<E, A> = Either<NonEmptyArray<E>, A>;
```

... Ce type s'appelle parfois VALIDATION

# TAKE AWAY

## DU TAKE AWAY

**Utilisez Option<A> pour représenter la possibilité d'une absence de valeur**

**Utilisez Either<E, A> pour représenter la possibilité d'une erreur**

**Utilisez Either<NonEmptyArray<E>, A> si vous voulez un accumulateur d'erreur pour valider des données**

**Dans la majorité des cas ce que vous voulez c'est Either<E, A> !!!**



# THE BOSS

SI VOUS VOULEZ ALLER PLUS LOIN

Renforcement :

🎥 [Error handling Isn't All About Errors by Jane Lusby](#)



# PROJET 2023

JUSTICKET



**Projet de promotion : Chaque groupe va créer un Système d'Information**

**4 projets constituent ce SI, la DSI veillera à leur bonne intégration**

**Nous sommes dans le cadre du cours, je m'attends à ce que vous mettiez en oeuvre des compétences acquises durant ce cours : utilisation des ADT, PBT, Gestion d'erreurs, etc...**

**Vous êtes (presque) des professionnels, vous devez être capable de réutiliser des compétences acquises dans le cadre d'autres enseignements (cf disclaimer 1 en intro de ce cours)**

**Je suis CLIENT, ce qui m'intéresse c'est le résultat. Livrables:**

- Repository GIT contenant code source, tests, documentation (1 par projet)
- Déploiement du SI mis à disposition (1 par Système d'information)

**Rendu le 30 Mars à Minuit.**

# PROJET 2023

DSI



**La DSI a un rôle particulier.**

**Elle doit veiller à l'intégration de l'ensemble des projets**

**Attention à bien identifier le chemin critique**

**Pensez itératif et incrémentés**

# PROJET 2023

CONSEILS



**Justicket**

**Ce projet représente peu de travail individuel, si vous appliquer une stratégie gestion de projet cohérente : lisser et répartir l'effort.**

**Vous aurez du temps sur chaque cours de QSI restant pour avancer sur le projet, si vous utilisez correctement ce temps 50% du projet sera déjà fait!**

**C'est un travail de groupe : ne travaillez pas à 1!**

# PROJET 2023

LES SOUS-PROJETS DU POC



**Justicket**

Périmètre : démontrer la logique d'émission et gestion des billets de concerts

Projets :

1. Application API de gestion des concerts (Prestataire 1)
2. Application de tokenization des billets (Prestataire 2 )
3. Application Web utilisateurs (Prestataire 3)
4. Application dashboard (DSI)

# **QUALITÉ DU SI**

---

**COURS 6 - COMMUNICATION INTERPROCESS**

# OBJECTIFS

## LES DIMENSIONS DE LA QUALITÉ

-  **Infrastructure** : matériel, réseau, OS, ... (*du baremetal au cloud*) ✓
-  **Logiciel** : applications construites et maintenues ✓
-  **Données** : données du SI (SQL / NoSQL) ✓
-  **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*

# THE QUEST

## LES COMMUNICATIONS SONT PARTOUT

Nous l'avons vu précédemment avec les micro-services le réseau est le point de faiblesse

Comment faire communiquer de manière robuste nos services et nos UI ?

Comment choisir le bon protocol ?

... en réalité ce ne sont pas des problèmes nouveaux, c'est ce qu'on appelle la communication inter-process

1 service = 1 process

La nouveauté vient du passage par le réseau

Mais l'IoT remet en avant le besoin de communiquer entre process au sein d'un même système



# COMMENT ÉVOLUE UN SYSTÈME

TEMPS

+1j : stock a bomb



# COMMENT ÉVOLUE UN SYSTÈME

REQUÊTE RÉPONSE



*I want a bomb*

Send a bomb



# COMMENT ÉVOLUE UN SYSTÈME

## MESSAGE ORIENTED



# COMMENT ÉVOLUE UN SYSTÈME

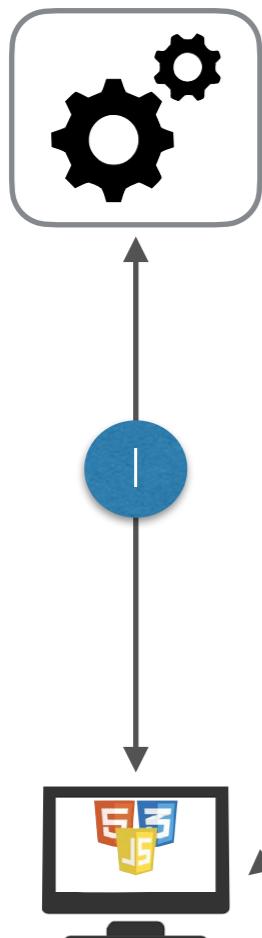
EVENT ORIENTED



# DANS UN SI

## EXEMPLE D'UNE APPLICATION BOURSIÈRE

Application back-end  
« portefeuille d'actions »



Application back-end  
« place de marché »

- 1 Lister les actions dans un portefeuille
- 2 Envoyer un ordre d'achat
- 3 Rafraîchir le cours d'une action

Quel stimuli (request / event / timer) ?

Quelle résilience (bloquant / always on) ?

Quelle « temporalité » ?

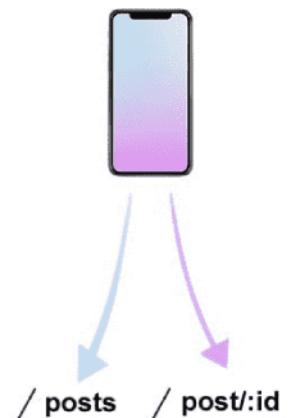
# CHOISIR LE BON OUTIL

Temps réel VS résilience	Client - server : request / response	Server - clients : event broker	Peer-to-peer	Batch
TR > résilience	<b>REST/JSON</b> <b>GraphQL</b> <b>gRPC</b>	Websocket	webRTC ipfs scuttlebutt	<i>BAD IDEA</i>
Résilience > TR	<i>RPC sur Message Queue</i>	<b>Message Queue</b> (RabbitMQ)	zeroMQ	<i>Timer process</i>
Pas de TR	<i>BAD IDEA</i>	Agent de transfert (Fluentbit)	Blockchain	<b>ETL (Talend)</b> <b>Data pipeline</b> (Fluentd)
IPC (no networking)	<i>RPC sur Unix Socket</i>	Unix Socket	Named pipe (FIFO)	crontab +.sh

# WEBSERVICES

REST/JSON LE STANDARD DES ANNÉES 2010'S

{...} REST



**REST = URI + VERBE + TYPE de média** (*application/JSON, application/XML, ...*)

*Pas de typage des ressources en standard mais possible via Json Schema ou Json-LD + schema.org*

**REST « RPC » ... a des problèmes :**

- Adherence back-front
- Empêche la mise en cache

**REST « HATEOAS » ... a des problèmes :**

- Orienté « ressources » quand les bonnes pratiques poussent à l'orienté « domaine »
- Problème de N+1 requêtes

# WEBSERVICES

## GRAPH LE NOUVEAU STANDARD



Langage de requête et serveur d'exécution

Transport : HTTP POST

Statiquement typé (Schema)

3 opérations :

- Query **Système de requête qui retourne exactement ce qui est demandé, évite la multiplication des end-points**
- Mutation **pour exécuter des opérations qui modifient les données côté serveur**
- Subscription **push des mises à jour de données du serveur vers le client en temps réels**

Très pratique pour construire un brique d'API Management (ex : [hasura.io](https://hasura.io))

Documentation toujours à jour avec GraphiQL

Créé par Facebook en 2012 ; géré par GraphQL Foundation depuis 2019

En 2021, vous devez connaitre GraphQL !

# WEBSERVICES

## GRPC : THE GOOGLE THING

Remote Procedure Call (RPC) léger et très rapide

Transport : http/2 + protocole buffers (format binaire)

Statiquement typé (proto file)

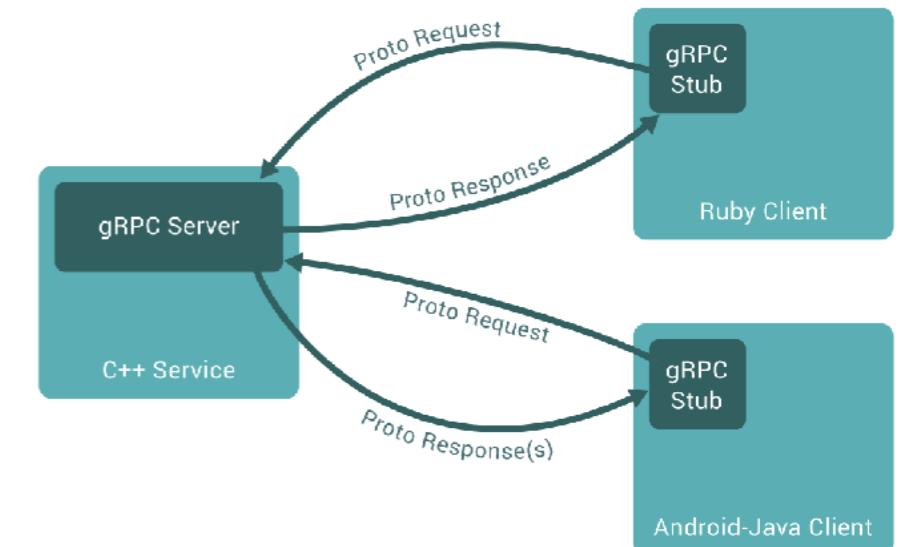
4 types de services :

- Unary RPC **le client envoie une requête et le serveur retourne une réponse**
- Server streaming RPC **le client envoie une requête et le serveur renvoi un stream**
- Client streaming RPC **le client stream un séquence de message et le serveur les lit puis retourne une réponse**
- Bidirectional streaming RPC **le client et le serveur s'échange un stream read-write**

Intéressant pour les cas où vous devez transmettre « beaucoup » de données (services IA audio/vidéo) ou quand vous avez besoin de faible latence (back-to-back)

Publié par Google en 2015 ; maintenant le format par défaut des API Google

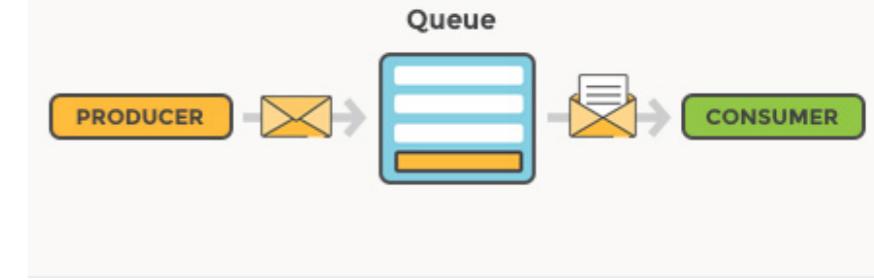
C'est surtout protobuf qui est intéressant ... vous pouvez tout à fait l'utiliser pour des messages sur message queue ou même API REST



# MESSAGE QUEUE

## POURQUOI ?

- Permet de mettre en oeuvre des patterns plus avancés que le simple RPC (pub/sub, routing, topics, ...)
- Permet de mettre en place des stratégies back-to-back!  
*Quand il n'y a pas d'Humain qui clique:*
  - Que faire quand le service n'est pas disponible ?
  - Que faire quand le service ne répond pas ?
- Facilite l'architecture backend
  - La découverte des services est simplifiée
  - Permet un couplage faible entre les services
  - Simplifie grandement les problèmes de gestion d'identité & droits



### Solutions:

RabbitMQ est sûrement la solution leader open source. C'est une très bonne solution, complète et avec un coût assez faible pour débuter

Apache Kafka est une autre solution populaire, le ticket d'entrée est beaucoup plus coûteux

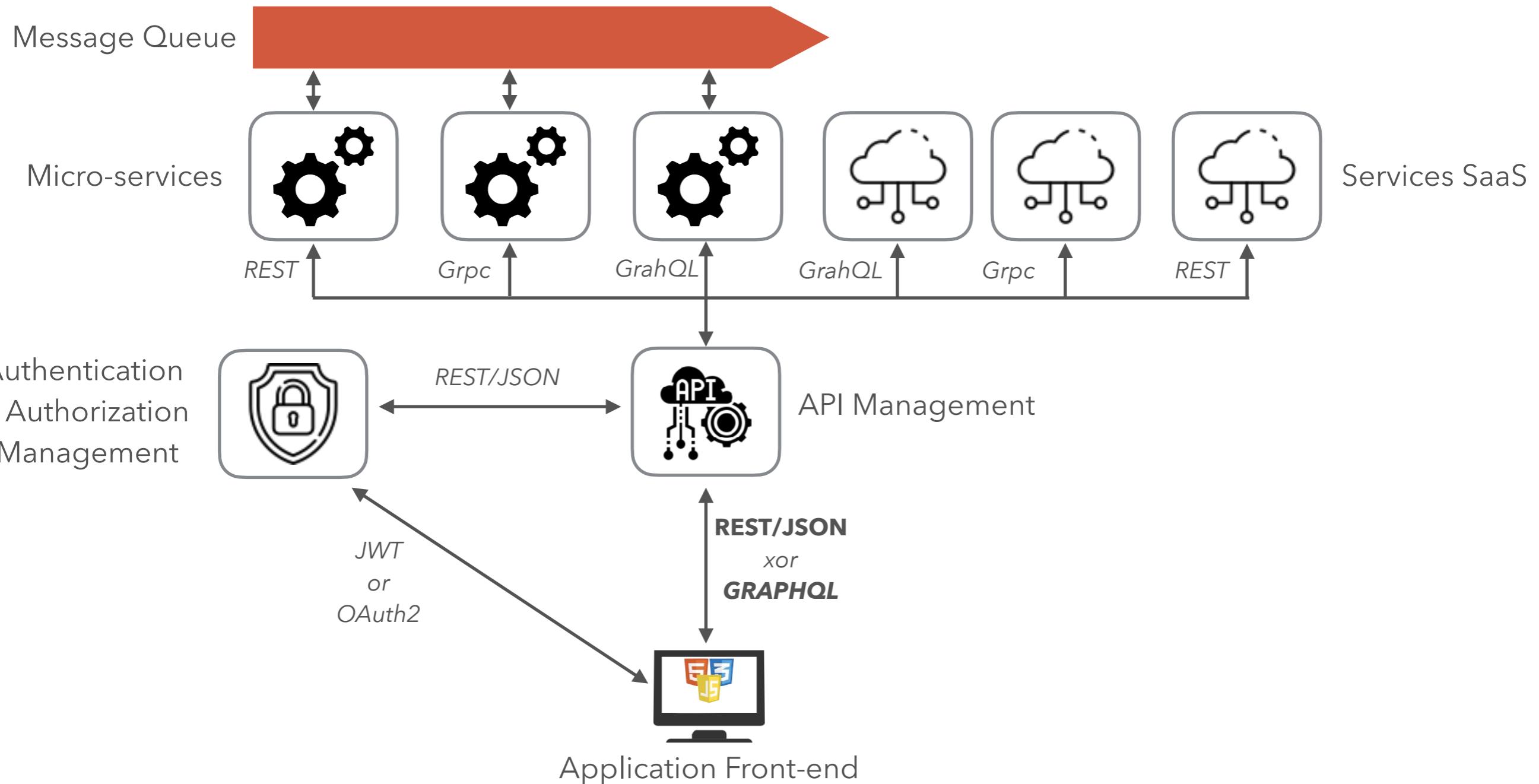
zeroMQ est une solution de niche assez intéressante et en croissance, y compris chez de grands comptes, mais plus bas niveau en terme d'UX

Tous les cloud providers GCP, AWS, Azure ont leur solution vendor lock

Les principales solutions ESB (IBM, TIBCO, Software AG, Oracle, Red Hat) intègrent une message queue

# LE VRAI VISAGE D'UN SI MICROSERVICES

UNE ARCHITECTURE FRONT-BACKS ZERO-TRUSTED NETWORK (LA PLUS COURANTE)



# TRANSPORT ET SERIALIZATION

NE FAITE PAS LA CONFUSION

- HTTP REST, GraphQL, grpc, amqp,... sont des protocoles de transport
- XML, JSON, protobuf, capnp, ... sont des formats de sérialisation

Vous pouvez faire n'importe quelle combinaison

- Privilégier JSON pour une interface utilisateur (API publique ou API backend destinée à une consommation Frontend)
- Privilégier un format binaire pour des communication back-to-back
  - Protobuf est leader mais garder un œil sur capnp

# TAKE AWAY

## UTILISER LE BON OUTIL

Webservice pour la com Front/Back :

- REST : pour les WS micro service
- GraphQL : pour l'API management et les notification

Message queue pour la com Back-to-Back

Body JSON est souvent la norme mais pas une obligation

Body binaire pour les streams (api audio/video) ...  
REST ou GRPC n'est pas vraiment important !



# **QUALITÉ DU SI**

---

COURS 7 - CLOUD

# OBJECTIFS

## LES DIMENSIONS DE LA QUALITÉ

-  **Infrastructure** : matériel, réseau, OS, ... (*du baremetal au cloud*) ←
-  **Logiciel** : applications construites et maintenues
-  **Données** : données du SI (SQL / NoSQL) ✓
-  **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*

# SIDE QUEST

Cloud computing

Avec les nouveaux modèles économiques dominants (paiement à l'usage, abonnement sans engagement, freemium), on cherche à transférer au maximum le CAPEX<sub>(1)</sub> vers de l'OPEX<sub>(2)</sub>

Les utilisateurs veulent un SERVICE plutôt qu'un produit, avec une mise à jour en continue et automatique...

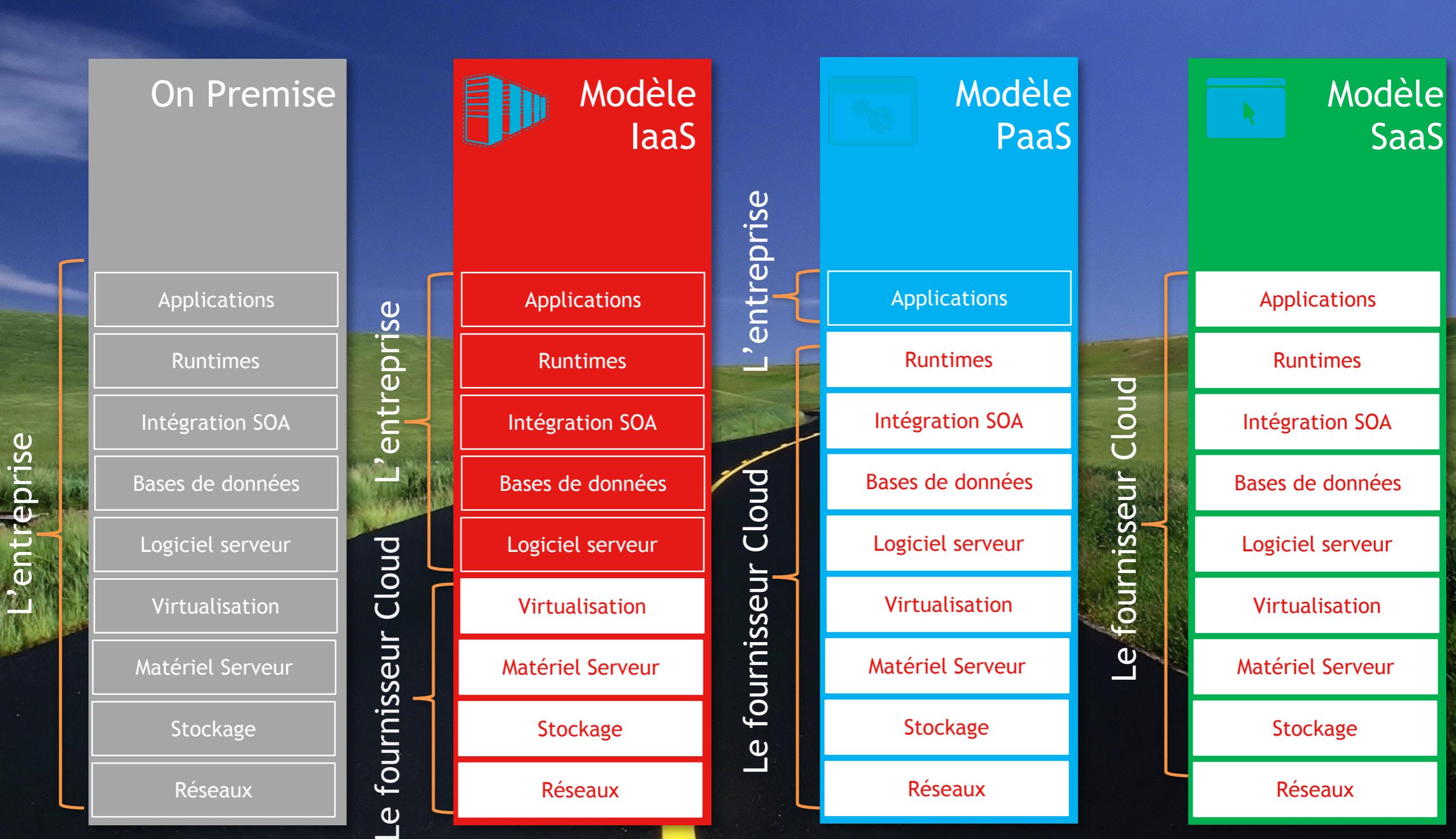
... Y compris pour le hardware (modèle Tesla)

<sup>(1)</sup> CApacity EXPenses : coûts fixes

<sup>(2)</sup> OPerational EXPenses : coûts variables



# LES POSITIONNEMENTS



# ON PREMISE

Infrastructure dans ses propres locaux

Contrôle absolu sur les données et les applications mais ...

... Demande de gérer l'installation, la sécurité, la maintenance, la sauvegarde, la disponibilité

Peu d'intérêt car n'apporte pas de valeur ... sauf si vous êtes offreur de cloud !

Dans des cas de SI « secret », un cloud « privé » peut être stratégique. Il doit être traité comme un cloud pour les équipes projets ... simplement l'offreur est interne

On-site

Applications

Data

Runtime

Middleware

O/S

Virtualization

Servers

Storage

Networking

You manage

Service provider manages

# IAAS

## INFRASTRUCTURE AS A SERVICE

Abstraction de toute l'infrastructure : on loue une machine virtuelle (VM)

Service dominant dû au « phénomène Docker »

Demande des compétences Ops pour gérer correctement les environnements (sécurité, maintenance, ...)

=> Peut-être coûteux si les compétences ne sont pas suffisantes en interne

Automatisation du provisionnement et du déploiement avec des outils comme Terraform et Ansible  
=> Infrastructure as a Code

IaaS

Applications

Data

Runtime

Middleware

O/S

Virtualization

Servers

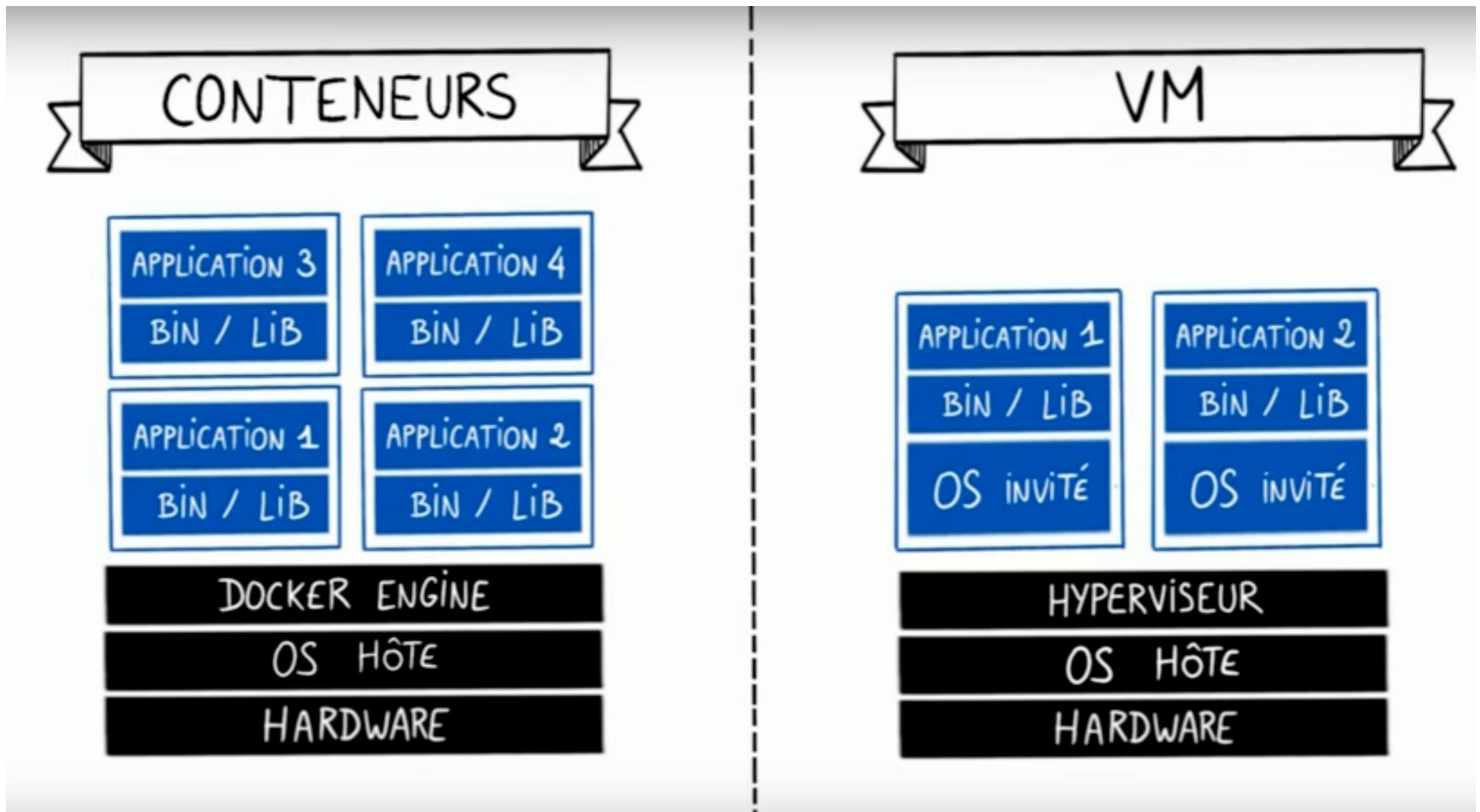
Storage

Networking

You manage

Service provider manages

# CONTENEURS VS VM



# CAAS

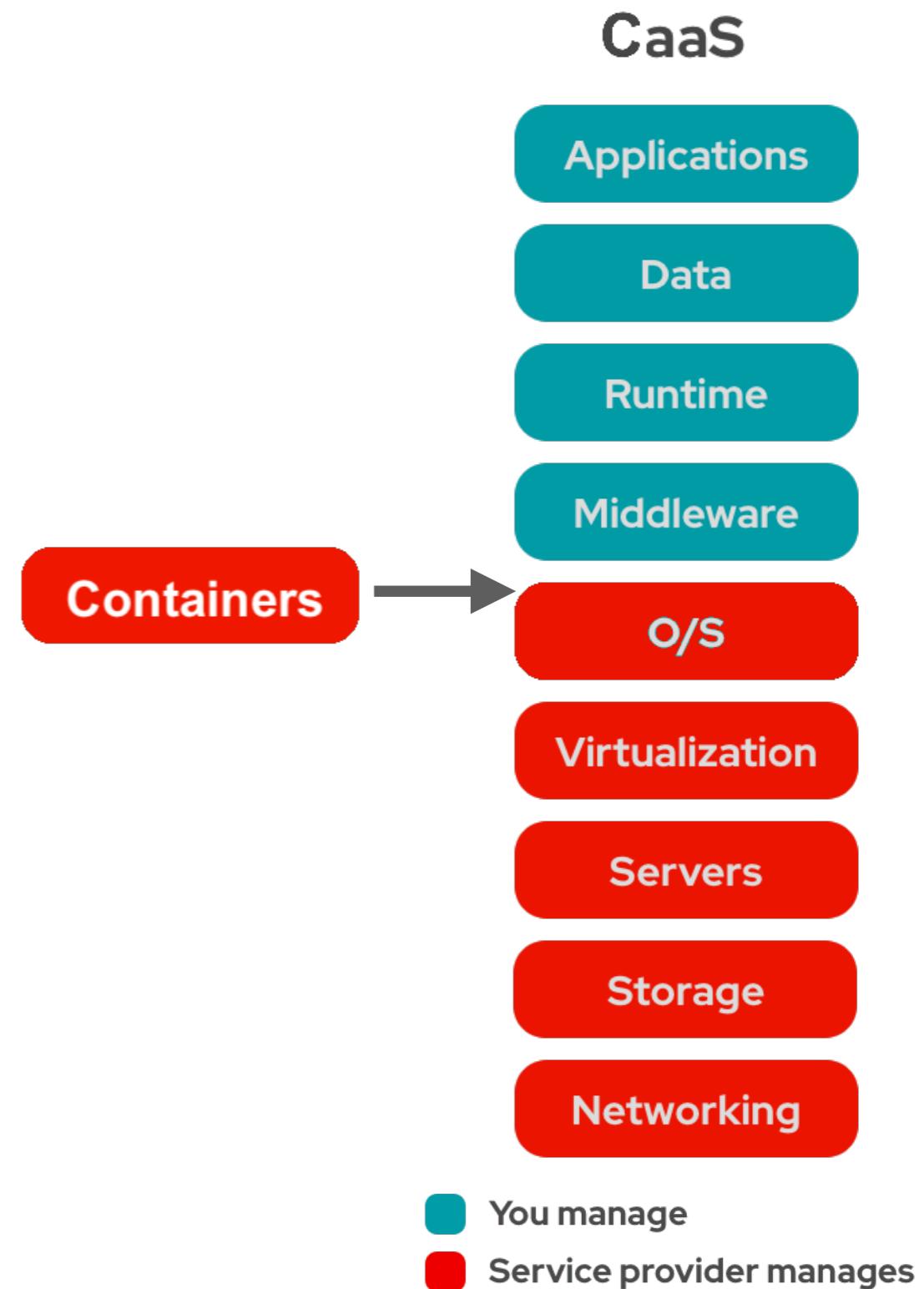
## CONTAINER AS A SERVICE

Le CaaS est un IaaS particulier

L'offreur de cloud fournit un environnement pour gérer des conteneurs (démarrage, arrêt, mise à jour...)

Permet aux développeurs de gérer des applications conteneurisées facilement

Mais demande aussi des compétences Ops pour gérer correctement les environnements notamment en prod !



# PAAS

PaaS

## PLATFORM AS A SERVICE

Le fournisseur de cloud gère les couches logicielles « basses » comme l'OS

=> Fournit un runtime prêt à l'emploi et des composants logiciels comme une base de données, des messages brokers, de l'object storage, etc...

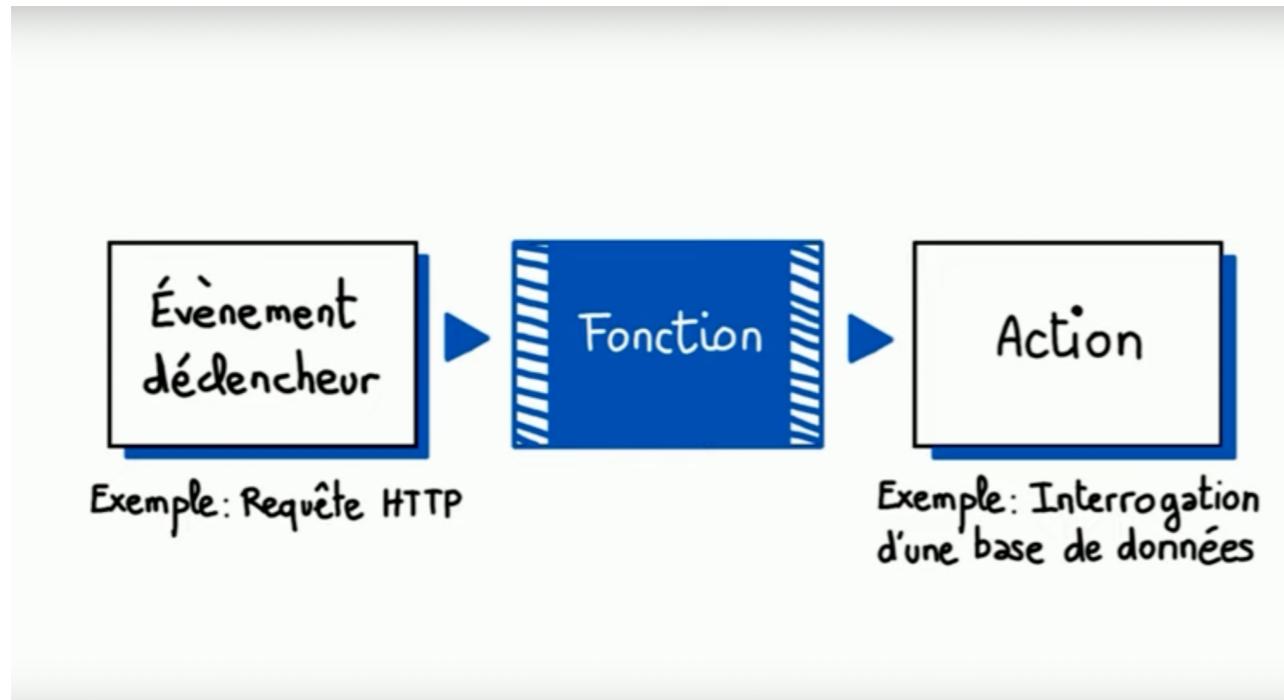
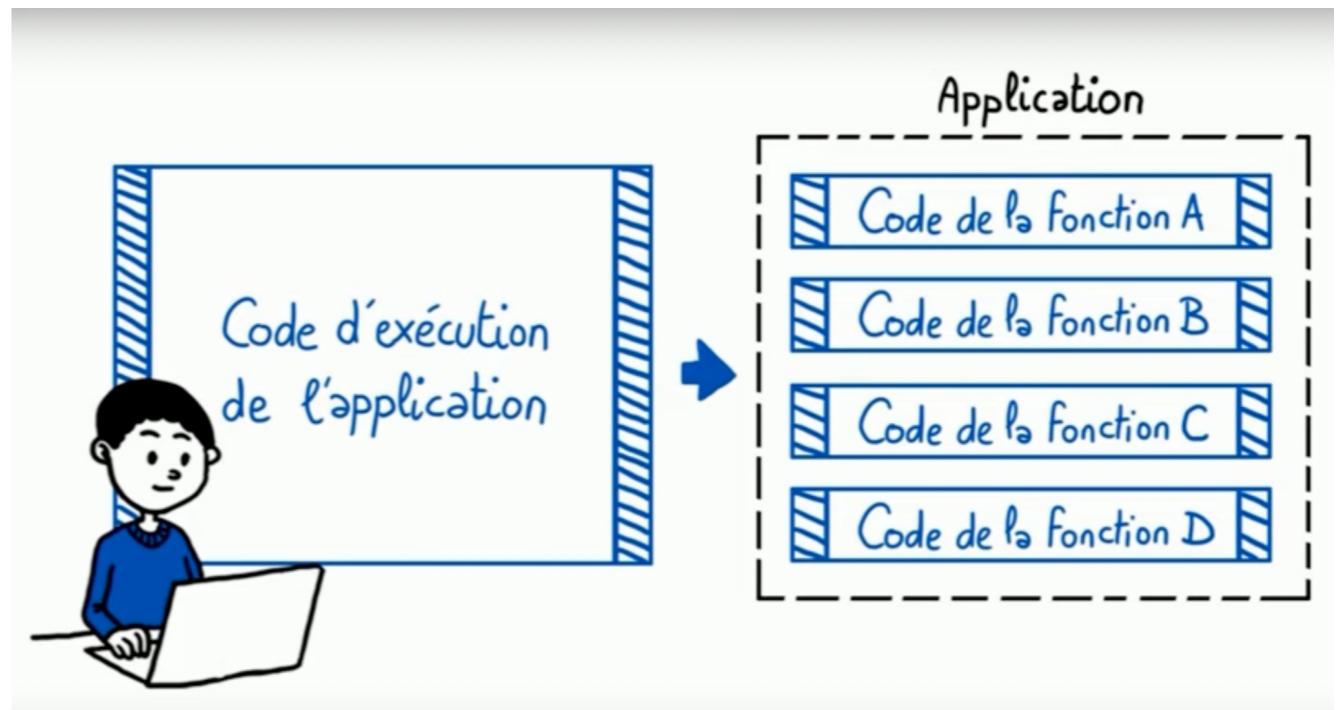
Déploiement facilité => permet de se focus sur l'apport de valeur



- You manage
- Service provider manages

# FAAS

## FUNCTION AS A SERVICE



# FAAS

## FUNCTION AS A SERVICE

L'offreur de cloud fournit un environnement d'exécution pour des morceaux de code (fonctions)

Il adapte automatiquement le nombre d'instances en fonction de la charge sur la fonction

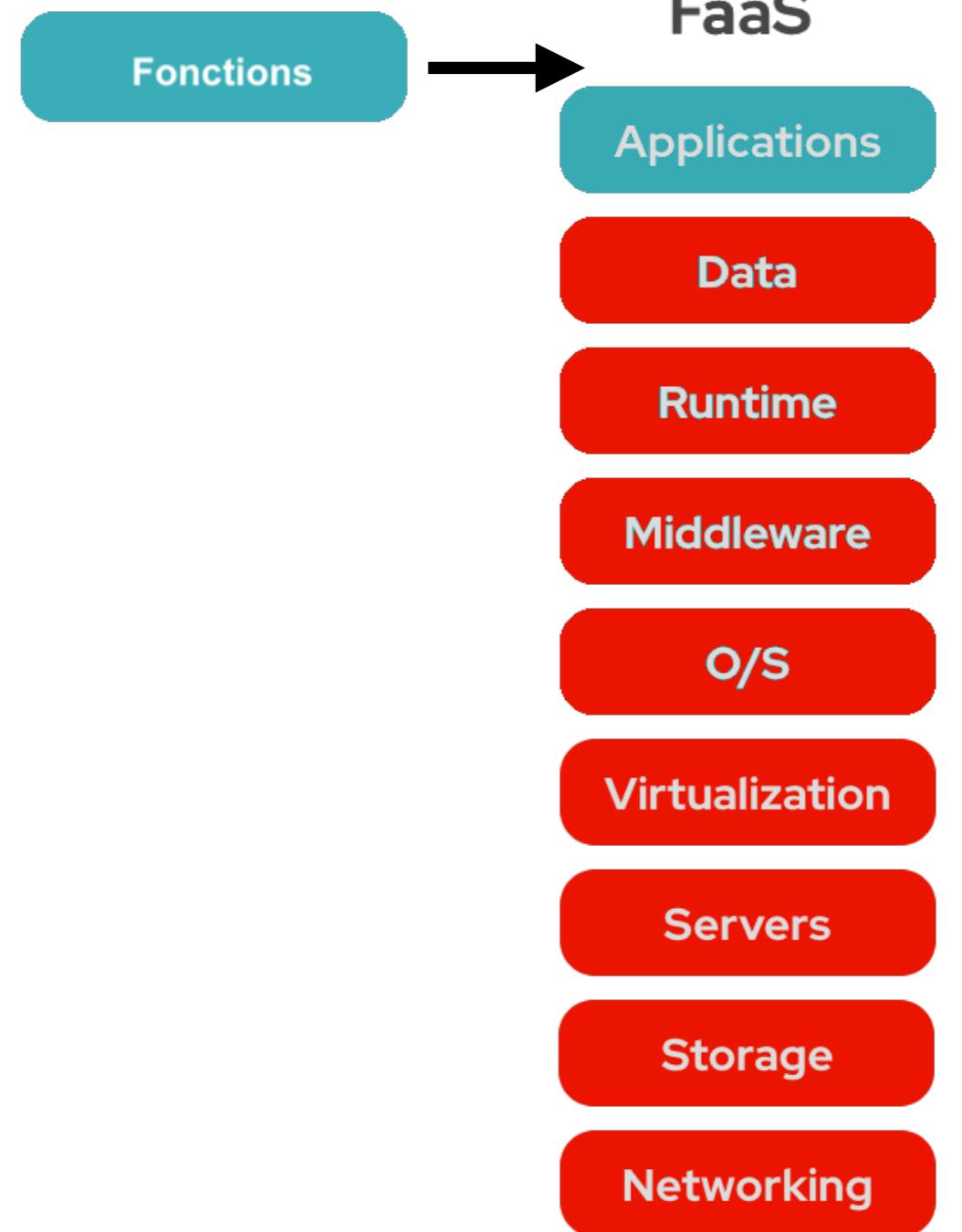
Paiement au temps d'exécution

MAIS :

Peu adapté en cas de trafic constant

Le cloud provider peut limiter les langages compatibles et imposer son framework

=> Forte dépendance au fournisseur de cloud !



- You manage
- Service provider manages

# SAAS

SOFTWARE AS A SERVICE

Mise à disposition d'applications complètes

Ex : Gmail, Notion, Evernote, etc...

SaaS

Applications

Data

Runtime

Middleware

O/S

Virtualization

Servers

Storage

Networking

-  You manage
-  Service provider manages

# POSITIONNEMENT DES OFFRES

Software As a Service  
SaaS

salesforce.com  
Success. Not Software.<sup>®</sup>

Office 365



Plateforme As a  
Service  
PaaS

Blockchain VM



Serveless  
FaaS



PaaS  
Platform



CLOUD  
FOUNDRY

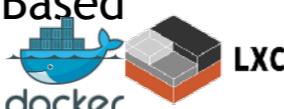


Microsoft Azure



Infrastructure As a  
Service  
IaaS

Container  
Based



LXC



amazon  
web services

Microsoft Azure



VM  
Based



openstack<sup>™</sup>



amazon  
web services

Microsoft Azure



# TENDANCES

## CONSTAT

**IaaS** domine sous le poids du **phénomène Docker** face au **PaaS** :  
c'est dommage car c'est (souvent) plus cher et les dev se mettent à gérer un OS ... ce qu'ils ne savent pas toujours faire pour de la **prod** : **risques de sécurité, risque de stabilité** (combien de Dockerfile avec `apt update -qy`?)

**Parts de marché en 2022** : ~33% AWS ; ~22% Azure ; ~9% Google ; ~5% Alibaba

**AWS est dominant et tente d'imposer des produits à Vendor Locks**

**Les outsiders (Azure, Google Cloud, IBM, Ovh, Clever Cloud, ...)** ont du coup une stratégie donnant une plus grande part à l'**open source** ou aux **technologies interopérables** (en général, Google a par exemple une stratégie vendor locks autour de sa gamme Firebase)

**Avenir incertain de Google Cloud (Top 2 en 2023 ou arrêt ... ou revue d'objectifs)**  
**et diversification des stratégies Google (Editeur de logiciel pour les offreurs de cloud avec Anthos)**

# IMPACTS

## ORGANISATION, COMPÉTENCES, SÉCURITÉ

Plus on abstrait l'infrastructure :

- Plus on doit abstraire les effets I/O dans l'architecture (stockage disk vs cloud object comme S3 ou OpenStack Swift)
- Plus on bascule d'une sécurité périmétrique (Firewall strategy) à une sécurité de la donnée (Zero-trust policy). Cela devient nécessaire : DevSecOps
- Plus le travail d'OPS devient un travail de DEV, il devient SRE (Site Reliability Engineer) - importance de l'automatisation



# TAKE AWAY

## CLOUD COMPUTING

Plus on monte dans l'abstraction plus on maximise l'OPEX et on diminue le CAPEX

Il y a des effets bénéfiques induits comme l'infrastructure as code ou l'infrastructure immutable

Attention au marketing : docker c'est du IaaS, le FaaS c'est du PaaS propriétaire

Cela impact les organisations RH :

- noOPS, OPS-dev, DEV & OPS ou DEVops ?
- SRE (software reliability engineer) ?



# DUNGEON BOSS

SI VOUS VOULEZ ALLER PLUS LOIN

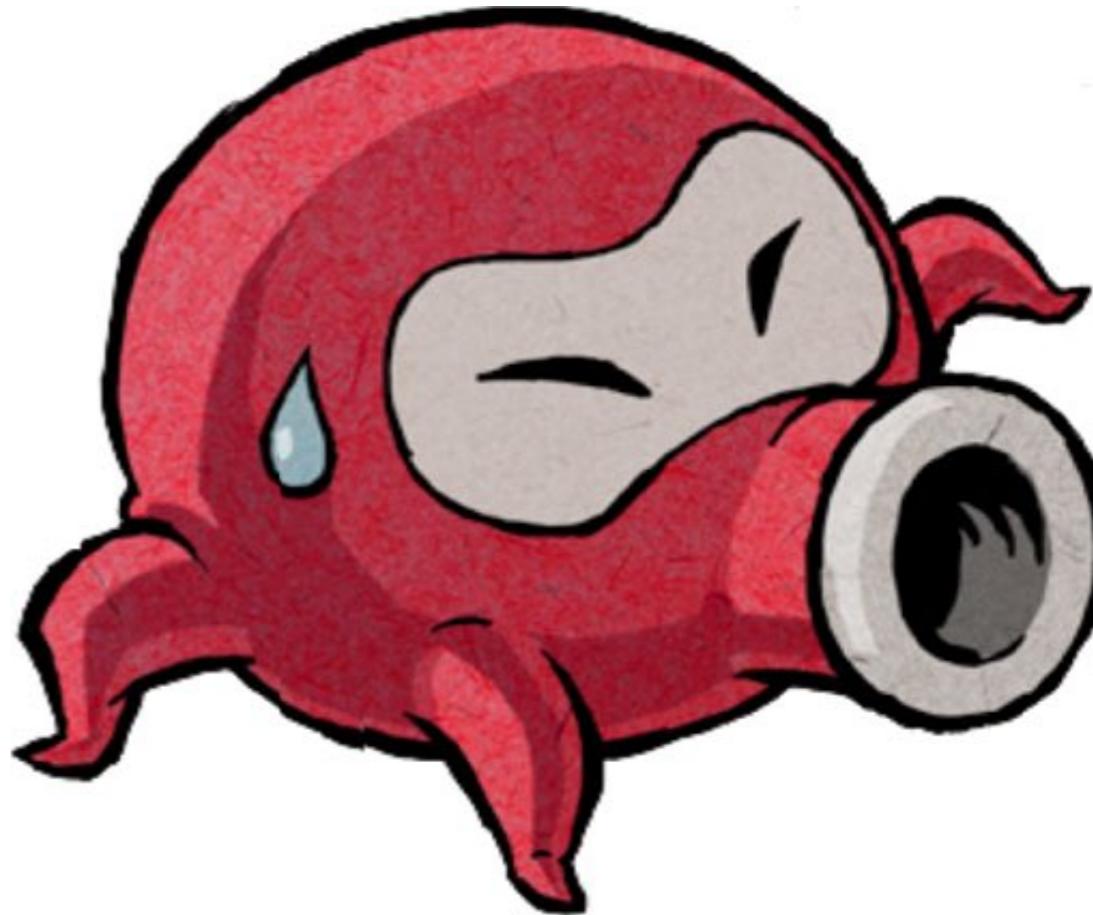
Renforcement :



12 factor app



Qu'est ce que S3



# OBJECTIFS

## LES DIMENSIONS DE LA QUALITÉ

-  **Infrastructure** : matériel, réseau, OS, ... (*du baremetal au cloud*) ✓
  -  **Logiciel** : applications construites et maintenues ←
  -  **Données** : données du SI (SQL / NoSQL) ✓
  -  **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*

# **QUALITÉ DU SI**

---

**COURS 8 - MICROSERVICES & MICROFRONTEND**

# THE QUEST

Du monolith au micro services

**Presque tout est web ! Au moins en informatique de gestion**

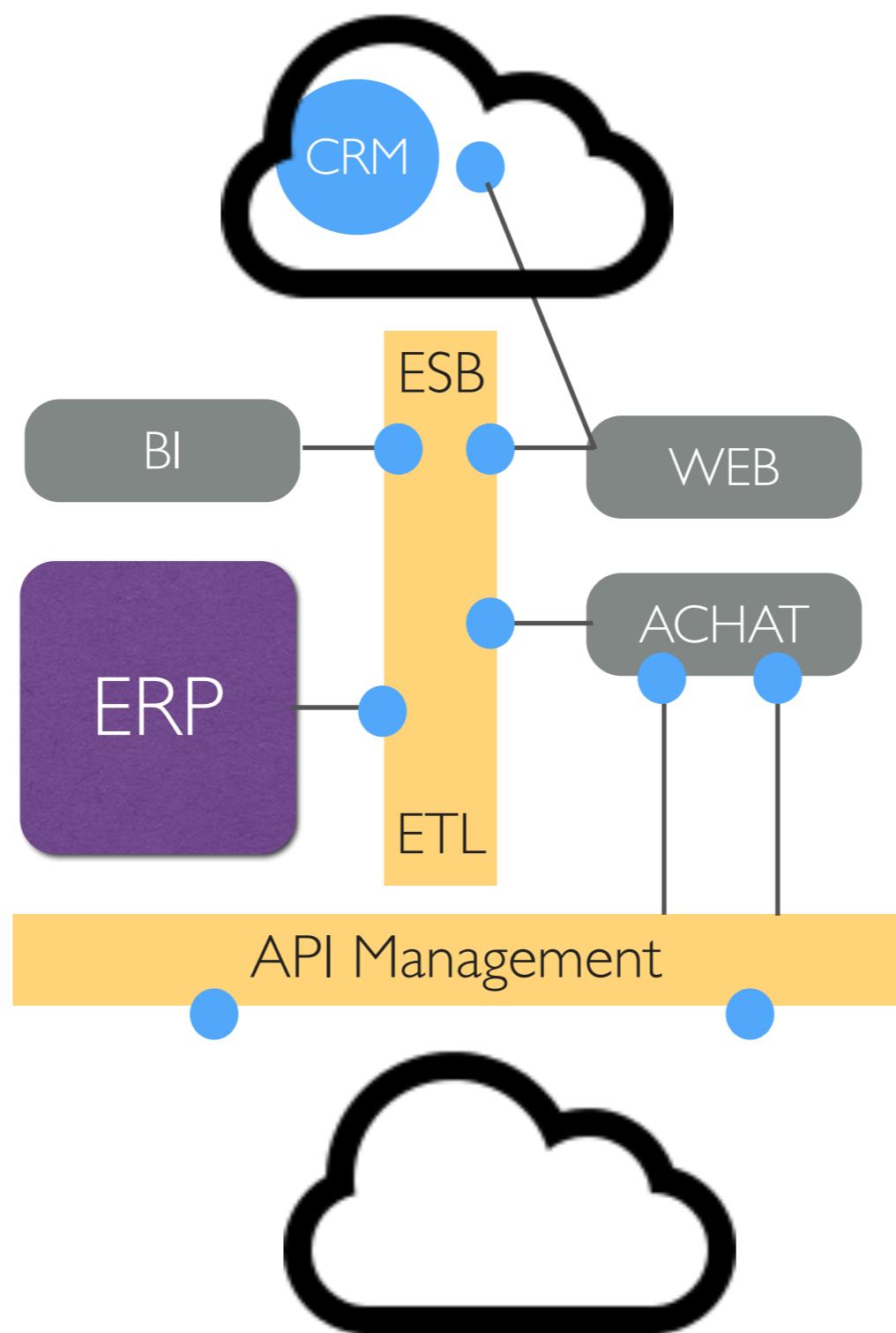
**Comprendre les architectures actuelles des web applications, leurs évolutions, leurs limites**

**Eviter le hype driven développement**

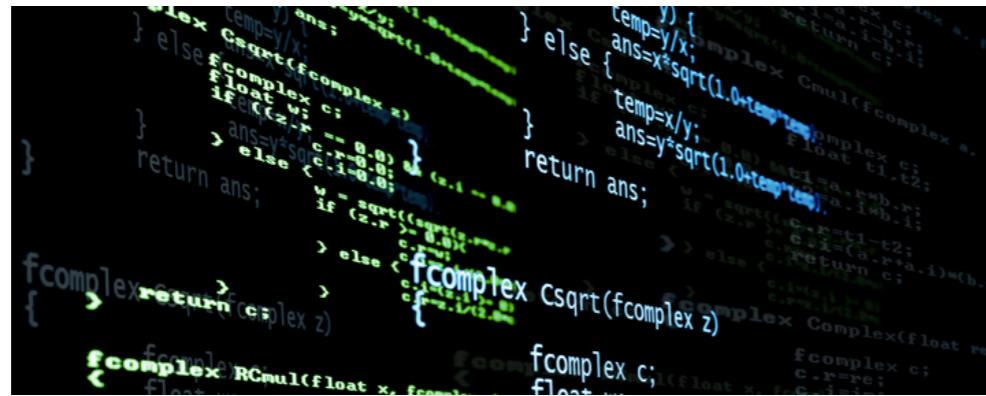
**Adopter une posture d'architecte**



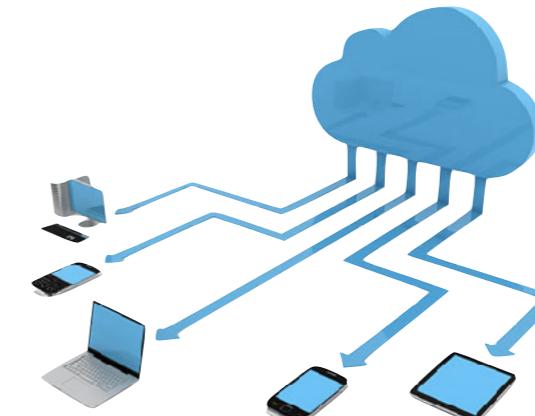
# LES SI ACTUELS



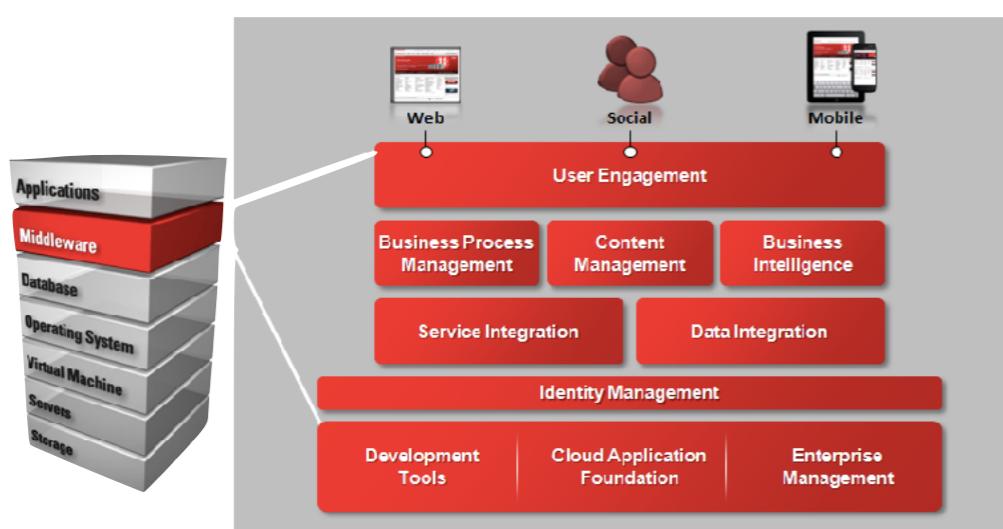
# LA COMPOSITION DU SI



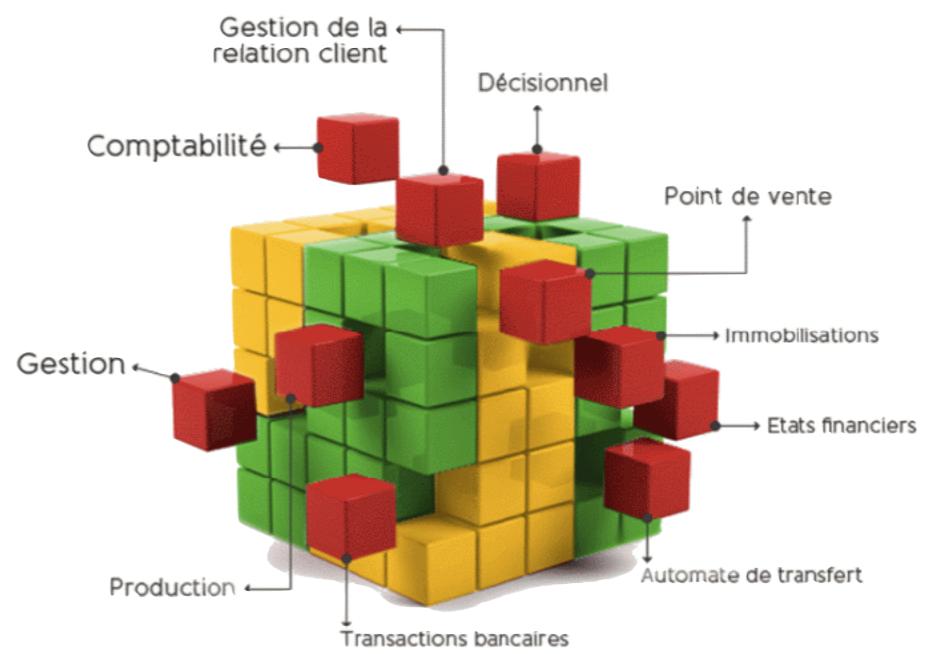
Logiciels spécifiques (App)



Services tiers (SaaS)



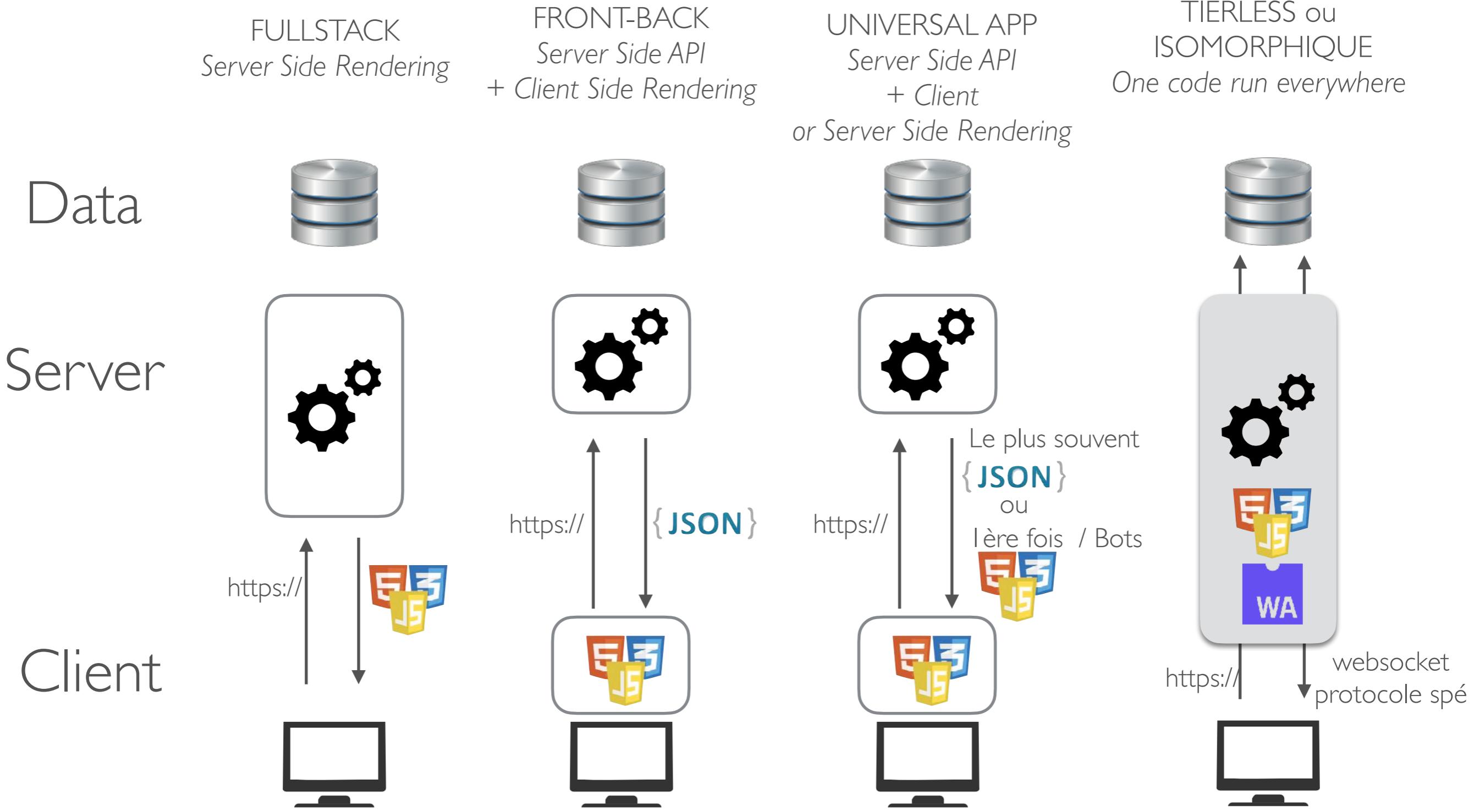
Progiciels



ERP

# QU'EST CE QU'UNE APP ?

## EVOLUTION DES ARCHITECTURES DES WEB APPS



# QU'EST CE QU'UNE APP ?

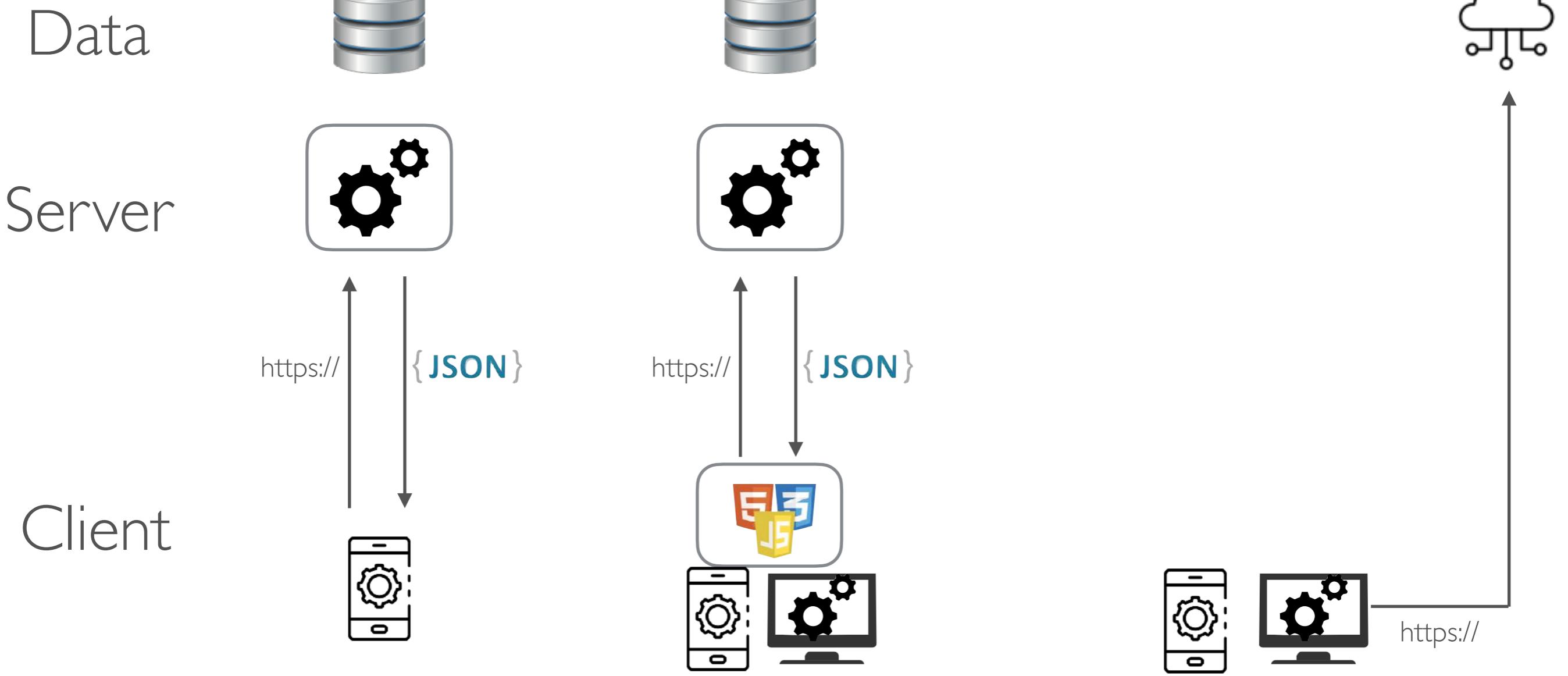
## LES APP CLIENTS LOURDS

CLIENT LOURD  
Server Side API  
+ Client lourd natif

HYBRIDE  
Server Side API  
Client lourd en JS

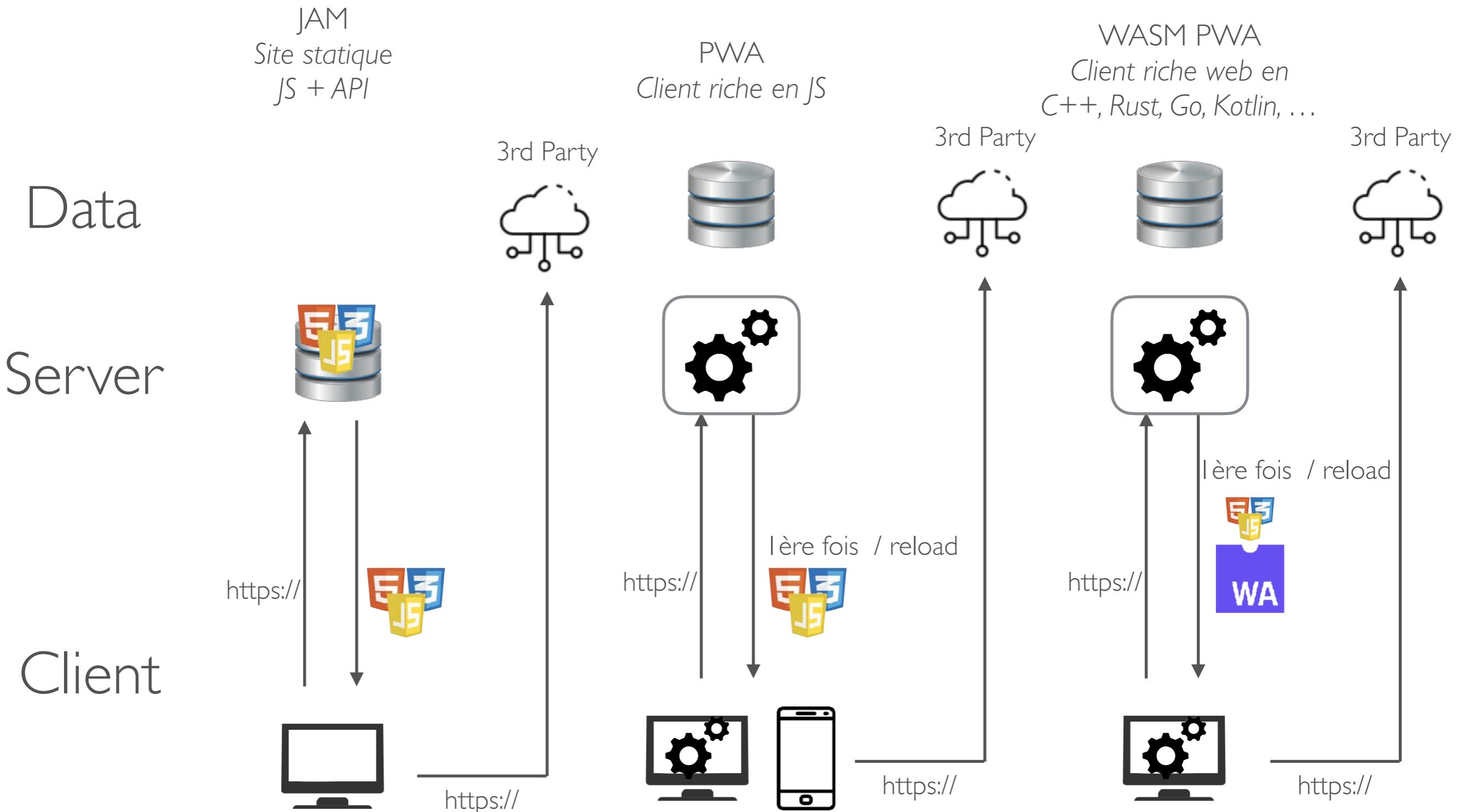
SERVERLESS  
*Client lourd +*  
Someone else server

3rd Party

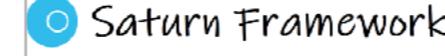


# QU'EST CE QU'UNE APP ?

## LES APP CLIENTS LOURDS « LÉGERS »



# LES FRAMEWORKS EXEMPLES

Langage	Tierless	Fullstack	Micro-Framework	CS Web UI	GUI
Java		 			
Scala					
Kotlin				Kotlin multiplatform	Kotlin multiplatform
C#	<b>Blazor</b>	<b>ASP.NET</b>		* <b>Blazor</b>	
F#	fsbolero				
OCaml			Opium	ocaml-vdom	<b>REVERY</b>
Javascript			express		
Python					
Rust				*	

\* WASM

# BACKEND API

## PRATIQUES USUELLES

La programmation monothreadée asynchrone est la norme (sauf en Java) à base de Fiber\* vs thread pool

=> à la base du langage : webAPI (js browser) / event-loop (node.js) ;  
=> pleinement intégré : Coroutine (Kotlin); go-routine (Go); asyncio (python) ;  
projet Loom (WIP - Java)  
=> fourni par lib tierce : FutureImpl Vert.x (Java); Lwt (OCaml) ; Cats-effect, ZIO (Scala) ; Tokio (Rust)

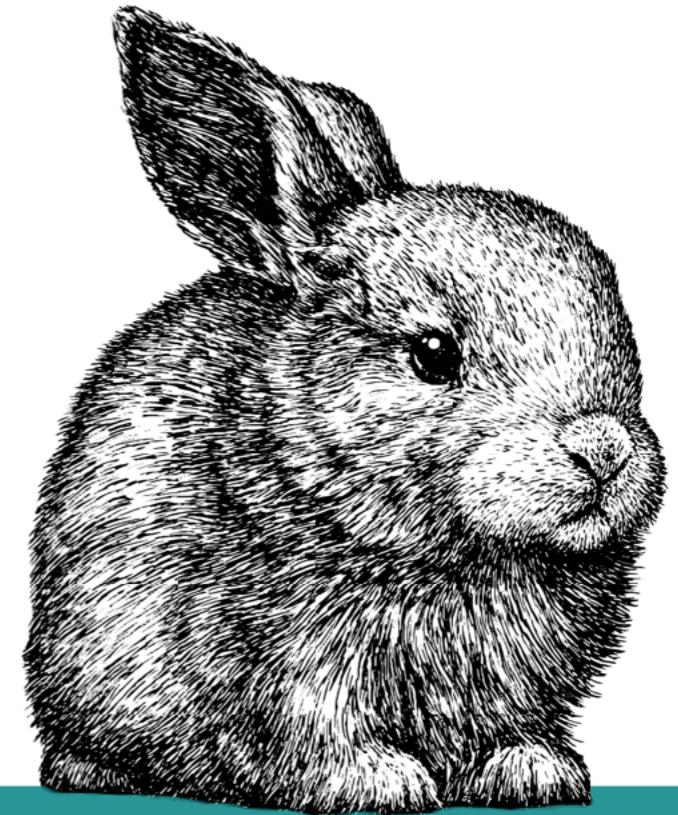
Montée en puissance des architectures « middleware » face aux « vieux » MVC ; dans certains cas d'autres patterns sont intéressants (machines de Moore, CQRS) ;

=> express (node.js) ; Vert.x (Java) ; Opium (OCaml), Flask (python); Ktor (Kotlin); Saturn (F#); actix (Rust)

Organisation du code en architecture hexagonale ou en oignon courante (structure de projet vu en ALOM)

\*a.k.a green thread a.k.a light thread a.k.a co-routine a.k.a event-loop

*Feigning knowledge of a word you've heard a few times*



*Expert*

Pretending to Know  
About Stuff

O RLY?

@ThePracticalDev

# CRUD API : QUELLE VALEUR ?

*Perfecting the parts that don't matter*

AVONS NOUS VRAIMENT BESOIN D'UN FRAMEWORK ?

REST/JSON :

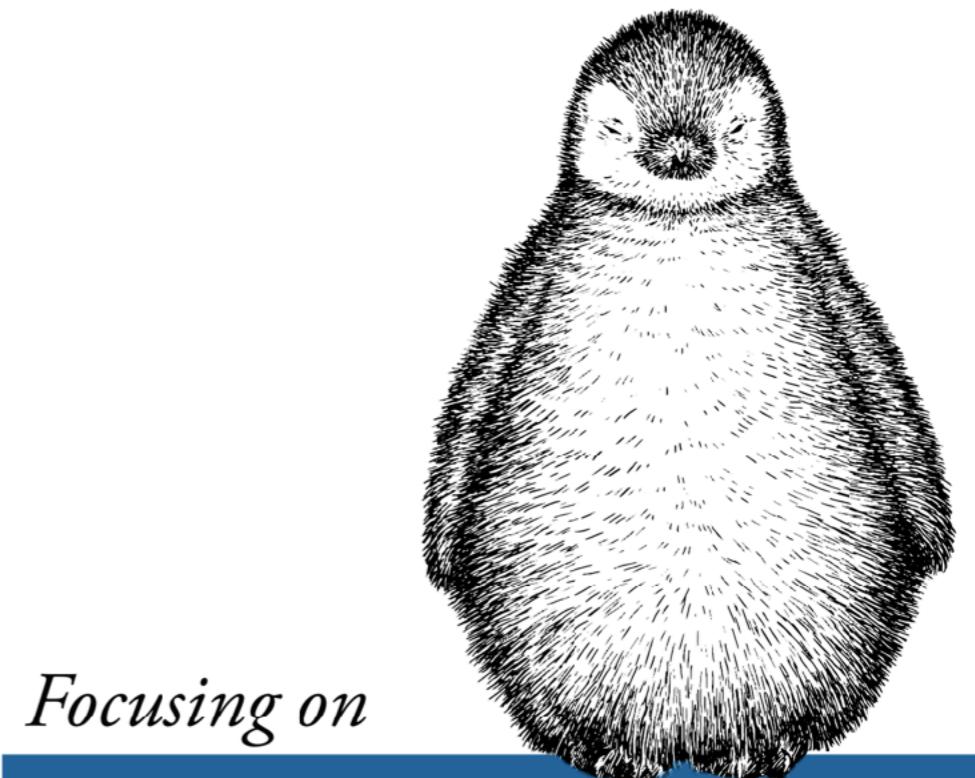
- [PostgREST](#) : plugin postgresql
- [CouchDB](#) : DB document

GraphQL :

- [PostGraphile](#) : plugin postgresql
- [irmin.io](#) : DB Key/Value

Realtime DB :

- [RethinkDB](#)
- [ParsePlatform](#)



O RLY?

@ThePracticalDev

# LEARN JS !

DE FACTO LE BYTE CODE DU WEB ... ET (PRESQUE) TOUT EST WEB

D'excellents moteurs : v8, Servo

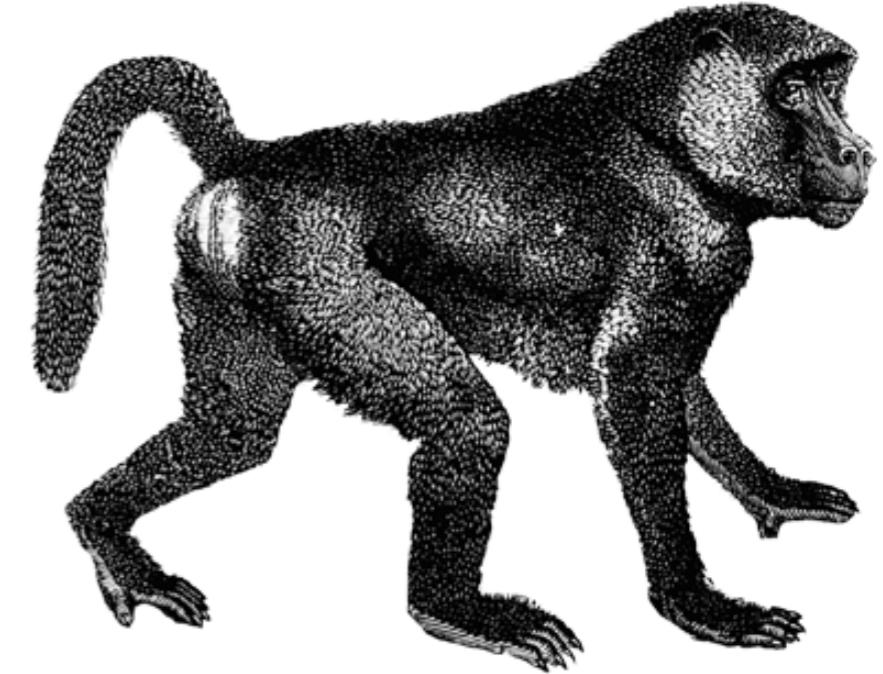
Langage « intéressant » mais « dangereux »

Certains langages statiquement typés ont d'excellents compilateurs vers JS : *TypeScript*, *Purescript*, *Js\_of\_OCaml*, *scala.js*, *Nim*, *Haxe*, ...

...Même utilisé comme byte code vous avez besoin de comprendre JS

Comprendre la WebAPI / Event loop est indispensable

*Because a good screwdriver fix everything*



Javascript  
Everything  
*Understand this*

O RLY?

*A. Good-Enough*

# LEARN WASM ?

## LE BYTE CODE DU WEB

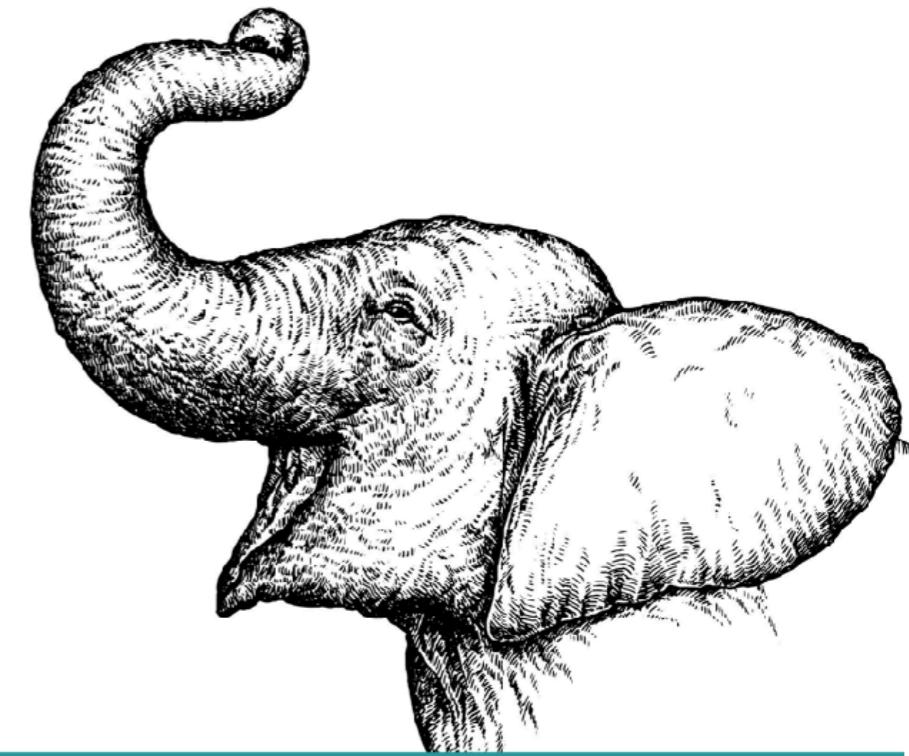
Le W3C ne voulait pas de JS comme byte code

Les cas où la performance est supérieure à JS sont limités, principalement:

- Si le coup de calcul est significativement supérieur au chargement/parsing (cryptographie, jeux vidéos, 3D)
- Si votre langage a un bon compilateur WASM : C, C++, Rust, C#, F# (ou lang avec LLVM backend)

C'est un vrai byte code : avez vous appris le byte code JVM ? Intéressant mais pas nécessaire !

*The answer to every programming question ever conceived*



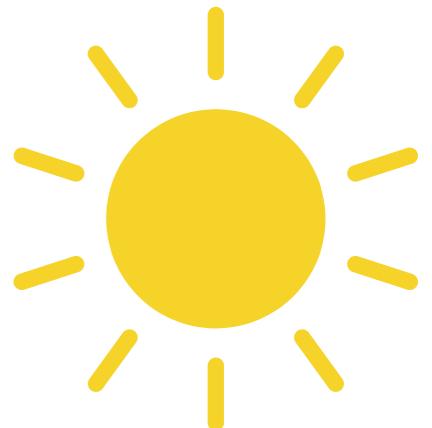
# It Depends

*The Definitive Guide*

O RLY?

@ThePracticalDev

# UN SI MICROSERVICES



## GAINS ESCOMPTÉS

Agnostique du langage

Agilité → Limité à un périmètre délimité domaine métier

Résilience → Facilement testable et gérable dans le temps

Extensibilité → Peut passer à l'échelle par domaine : « horizontal scale »

# DES MICROSERVICES

DE NOUVELLES CONTRAINTES SUR LE SI



**Quelle est la limite du service ?**



**Le réseau ça tombe, c'est lent, ... il faut le gérer !**



**CI/CD mandatory. Il faut être dev et ops, finit l'amateurisme !**



**Le monitoring est complexe**



**La communication inter-process c'est complexe**



**Les policies de sécurité sont complexes**



**L'infra ça coûte un bras ... si ma société ne me permet pas de PaaS provider**



**Le front devient adhérent de tous les services !**

# LE MONOLITH

CE FAMEUX CROQUEMITAINE

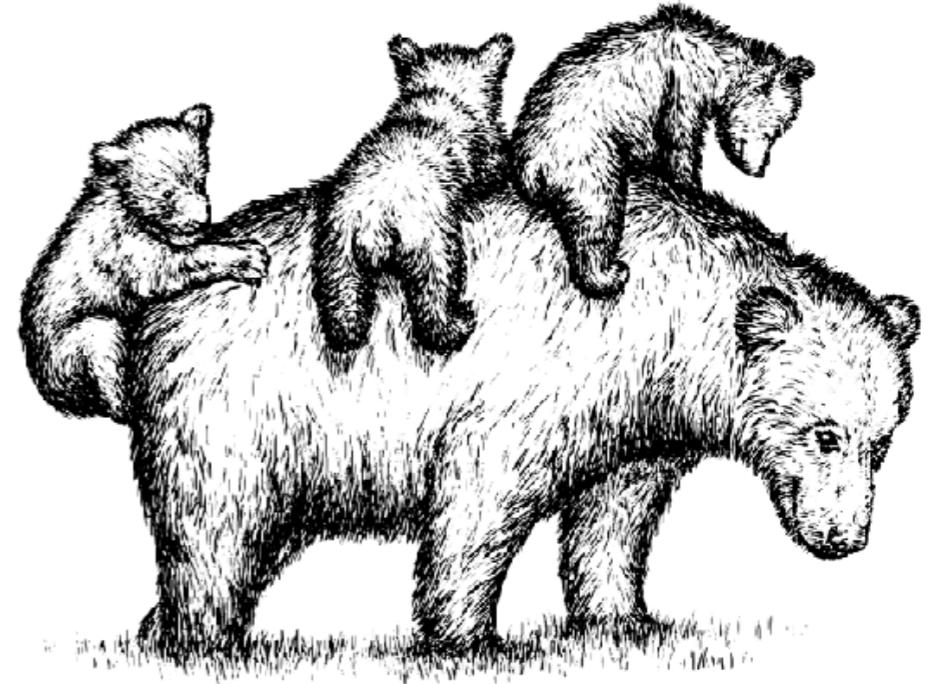
Désigne un logiciel qui n'est pas microservice

Dans les faits des logiciels « legacy », mal conçus, mal gérés dans le temps et qui provoquent de l'attrition

Bref de MAUVAIS INVESTISSEMENTS ...

... mais on peut aussi faire des erreurs avec des microservices

*Getting the wrong idea from that conference talk you attended*



Solving Imaginary  
Scaling Issues

*At Scale*

O RLY?

@ThePracticalDev

# CONCEPTION DE SERVICES

AVONS NOUS BESOIN DE MICROSERVICE ? DÉCONSTRUIRE LE MYTHE !

**Il n'est pas nécessaire de faire des « micro service web » pour :**

**Agnostique du langage** : la plupart des langages permettent soit de l'interopérabilité de bytecode (JVM, .Net), soit de l'interopérabilité de librairie C (.dll, .so, .a), soit de l'interopérabilité par un pivot de langage (JS) ou un assembly (Wasm)

**Agilité** : 1 service ~ = 1 librairie => 1 équipe ; c'est le fonctionnement des communautés open sources

**Résilience** : Typage statique + fonctions

**Extensibilité** : L'« horizontal scale » est parfois plus cher que le « vertical scale » sur le cloud et dans certains cas moins performant (puissance de calcul VS parallélisation)

**Tout cela a l'avantage de produire des logiciels valides à la compilation et de réduire l'adhérence au réseau !**

# HYPE DRIVEN DEVELOPMENT

???

« Quand une équipe décide d'une technologie, d'une architecture ou d'un design en fonction de sa popularité » ... souvent influencé par les Google / AWS / Facebook / Netflix / AirBnB / Uber / Spotify ces dernières années

Les anti HDD ont souvent un discours conservateur qui vise à justifier l'immobilisme de leur entreprise !

Mais sont souvent dans une forme de (elderly)-Hype

Syndrome de « c'était mieux avant » ? Pas que...

*Looking for love in all the wrong frameworks*



## Hype Driven Development

*Life on the Bandwagon*

O RLY?

@ThePracticalDev

# HYPE DRIVEN DEVELOPMENT

QU'EST-CE VRAIMENT QUE LA HYPE ?

```
#define P(a,b,c) a##b##c
#include/*+*****+*/<curses.h>
int          c,h,          v,x,y,s,          i,b; int
main         () {           initscr(          ); P(cb,
rea,          k)()          ;///          );
P(n,          oec,          ho)(          );
/*          ;for          (curs_set(0); s=          x=COLS/2
; P(      flu,          shi,          np)()) { timeout(y=c=
P(c,          lea,          r)()          ;for          (P(
mva,          d,          dstr          )2,
G) ;          ; P(          usl,          3+x,
          P(m,          vad,          dstr          eep,
          "      "); for(i=LINES; /*          )( y      >>8,x,//
; mvinsch(i,0,0>(~c|i-h-H          */ i
:(i-          h|h-          * / i
if((          i=( y          &h-i          -->0
          A)>>8)>=LINES | |mvinch(i*= 0<i,
!=mvinch(i,3+x))break/*&%  &*/;
          >>8,          x,0>v          )? ' '
          /-W;          P(m,          i+H)
          COLS-9," %u/%u ",(0<i)*          i+H)
          b); refresh(); if(++          <0?' '|'
          --W; h=rand()% (LINES-H-6          >0?I:v+
          )+2; } } flash(); }}
```

# TAKE AWAY

## ARCHITECTURE DES SERVICES

Difficile de faire le tri entre hype, marketing et besoin : soyez pragmatique

Vous devez être capable d'argumenter un choix

La limite d'un service est souvent un client ou la capacité de le gérer avec une pizza team

Il n'y a pas de mal à faire des « migroservices »

Être architecte c'est faire des choix



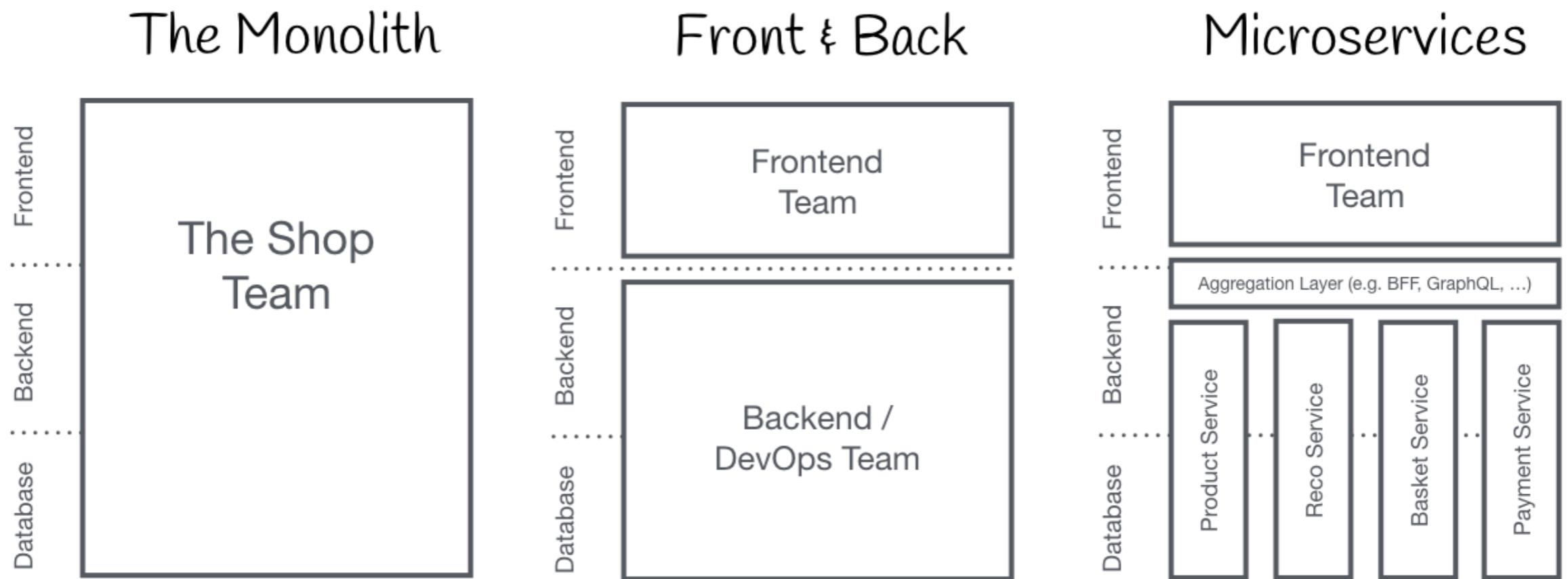
# MICROFRONTEND

DES MICROSERVICES AUX MICROFRONTEND



# LE PROBLÈME

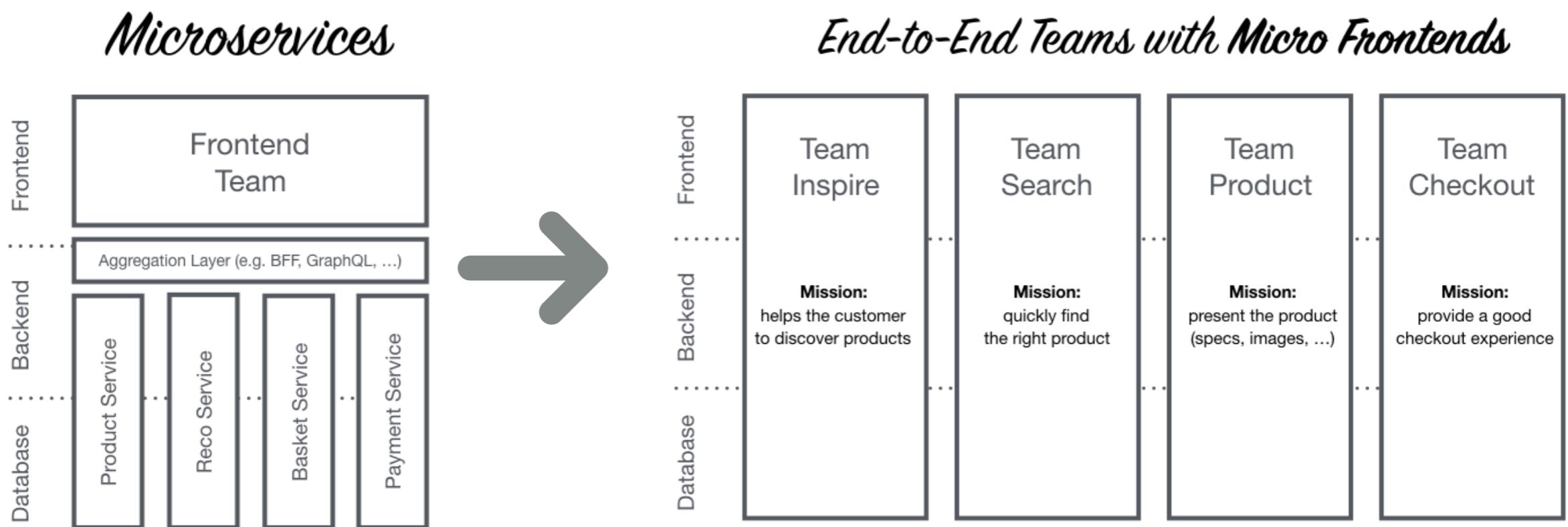
AVEC LES MICROSERVICES L'ÉQUIPE FRONT EST DEVENU LE GOULOT D'ÉTRANGLEMENT



pictures from <https://micro-frontends.org/>

# L'OBJECTIF

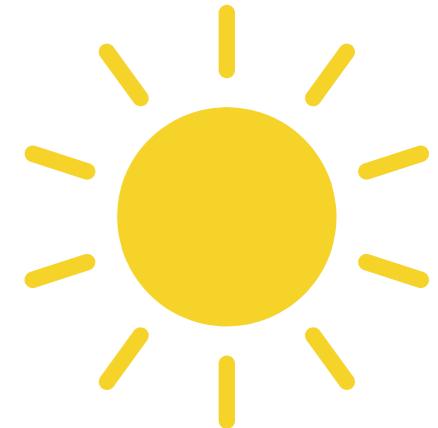
APPLIQUER LES PRINCIPES DES MICRO SERVICES AUX FRONT-ENDS



pictures from <https://micro-frontends.org/>

# MICROFRONTEND

GAINS ESCOMPTÉ, COMME LES MICROSERVICES



Agnostique du langage

Agilité → Peut être buildé et déployé de manière isolée

Résilience → Spécialisé pour une partie bornée de l'UX

Extensibilité → Assemblé avec d'autres micro-frontend pour créer une UX

# DES MICROFRONTEND



ON A SEULEMENT RÉSOLU 1 TRADEOFF DES MICROSERVICES



**Quelle est la limite du service ?**



**Le réseau ça tombe, c'est lent, ... il faut le gérer !**



**CI/CD mandatory. Il faut être dev et ops, finit l'amateurisme !**



**Le monitoring est complexe**



**La communication inter-process c'est complexe**



**Les policies de sécurité sont complexes**



**L'infra ça coûte un bras ... si ma société ne me permet pas de PaaS provider**



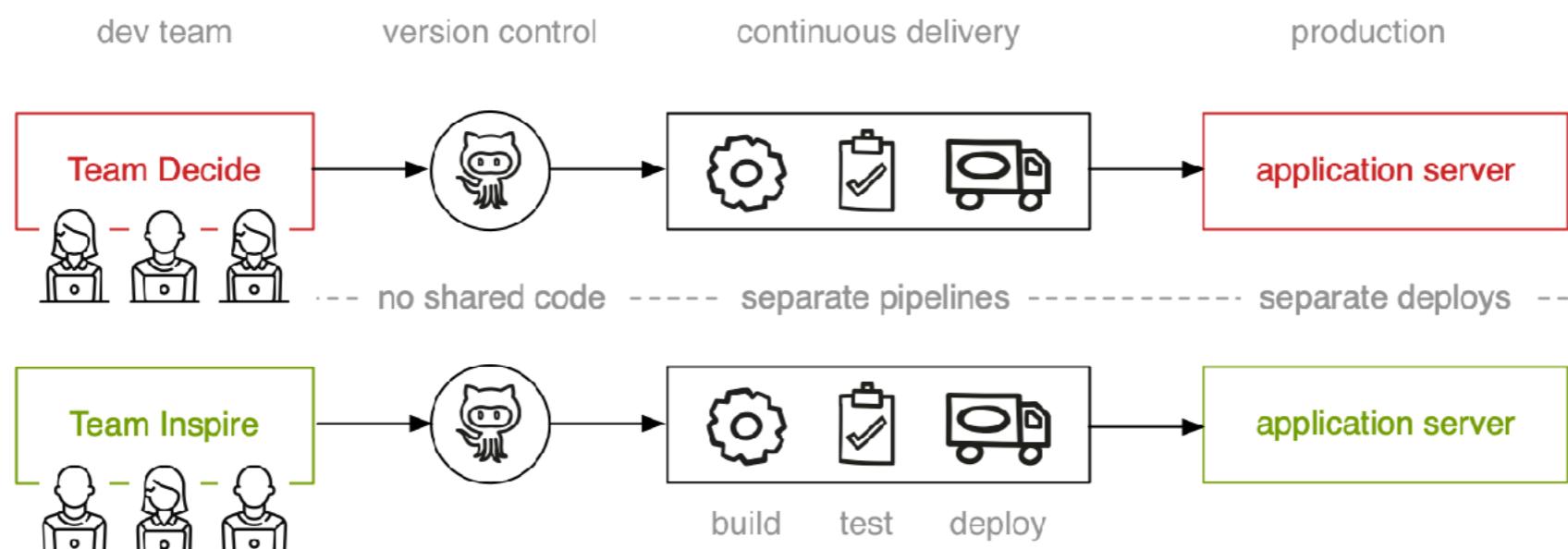
**~~Le front devient adhérent de tous les services !~~**

# MICROFRONTEND

RECHERCHE DE L'AUTONOMIE COMPLÈTE D'UNE ÉQUIPE

C'est l'équipe qui décide

- De sa stack technologique
- De déployer de manière indépendante des autres équipes



pictures from the book « Microfrontend in Action », manning publication

# MICROFRONTEND

## UN CHANGEMENT D'ORGANISATION

Une organisation micro-services est avant tout une réorganisation de la DSI autour de spécialistes (produit, reco, panier, payement, front), regroupés autour d'une compétence métier ou d'une technologie

Une organisation micro-frontend repose sur des organisations transversales (Inspiration, Décision, Achat), regroupées autour d'un besoin client

Une page est donc la propriété d'une équipe et peut intégrer des fragments d'autres équipes

# EXEMPLE

Team  
Décision  
(fragment)

The screenshot shows the Zalando homepage with a red header banner. The banner features a photo of a man holding a child and the text "Ensemble, bientôt. Idées cadeaux pour les fêtes". Below the banner is a green dashed-line grid containing five product cards:

- Créateurs**: A man in a black kimono-style outfit.
- Eco-responsabilité**: A man in a blue turtleneck sweater.
- 30 % Créateurs**: A man in black pants and a white shirt.
- Hot Drop**: A pair of Salomon Speedcross 3 ADV UNISEX shoes.
- 20 % Créateurs**: A man in a grey blazer and black pants.

Each card includes the brand name, product name, original price, discounted price, and a green progress bar at the bottom.

**Faites-leur plaisir**  
Pour tout budget

Team Inspiration (page)

# EXAMPLE

**Ensemble, bientôt.**  
Idées cadeaux pour les fêtes

Découvrir →

**Faites-leur plaisir**  
Pour tout budget

- Moins de 25 €
- Entre 25 & 50 €
- Entre 50 & 100 €
- Sélection créateurs

**Zign**  
**Pull-over**  
19,99 € TVA incluse

★ ★ ★ ★ ★ 6

Couleur: royal blue

XL (en taille Z)

Ajouter au panier

**MON PANIER**

Zign Pull-over - royal blue Couleur: bleu roi Taille: XL Quantité: 1

19,99 € TVA incluse

3,50 € Livraison

Total: 23,49 €

Peut toute commande à un montant supérieur à 24,99 € vous pouvez bénéficier de la livraison gratuite.

Retour et remboursement

**Mon panier (1 article)**

Zign Pull-over - royal blue Couleur: bleu roi Taille: XL

1 19,99 €

Les articles dans le panier ne sont pas réservés.

**Livraison estimée**

Me, 23.12. - Lu, 04.01.

**Nous acceptons**

Voir plus >

Total	19,99 €
Sous-total	19,99 €
Livraison	3,50 €
Total (TVA incluse)	23,49 €

**COMMANDER**

Ajouter un code (facultatif)

**Vous aimerez sûrement aussi**  
**Inspiré par vos choix**

- à partir de 27,99 € JIMARCO JJCONNOR CHECK - Chino - d...
- à partir de 38,49 € MARCO BIONI - Chino - black
- à partir de 27,99 € à partir de 39,99 € ONSMARK PANT STRIPE - Pantalon classi...

Team Inspiration  
(page)

Team Décision  
(Page)

Team Achat  
(Page)

# COMMENT FAIRE TECHNIQUEMENT ?

INCLUDE LES FRAGMENTS ET GÉRER LE ROUTAGE : HTTP OU SPA

Hyperlink : routage par reverse proxy +



Intégration client par iFrame  Spotify



Intégration serveur SSI : DHTML so 90's ... et pourtant  zalando 

Single Page Application : routage client +



Utiliser une JAM Stack : générer un site statique à partir des pages de chaque team (très bien adapté au sites web, moins au applications web)



Architecture App Shell 

# COMMENT FAIRE TECHNIQUEMENT ?

## DÉVELOPPER LES FRAGMENTS

钐 Abandonner le critère « agnostique » et choisir un framework d'entreprise  
(ex react) 

钐 Utiliser le web components comme un langage commun d'intégration  
 clever cloud

🚀 Utiliser des framework tierless

# POINT D'ATTENTION SUR LES WEB COMPONENT

UN STANDARD W3C QUI A MIS TRÈS LONGTEMPS À ARRIVER ... ET PLUS FORCÉMENT ADAPTÉ AUX ATTENTES ACTUELLES

Web components = Shadow DOM + Custom Element + HTML Template

Une API assez bas niveau plus destinée à construire des frameworks qu'à être utilisée directement

Approche de templates dirigés par les données (à la angular) préférée à une approche expressive (à la react)

Framework star : lit-element

Recyclage de vieux frameworks : ionic, vue.js

Tous les frameworks SPA mainstream (react, angular, vue.js, svelte) permettent d'inclurent des web components dans leurs composants propres

Mais c'est un standard 🤔

# POINT D'ATTENTION SUR LE TIERLESS

BACK TO THE FUTURE

Issu de la recherche : Links, Eliom

Dissémination : Meteor, Elixir Live Views ... mais surtout Microsoft avec Blazor

Des publications prometteuses sur les SESSION TYPES\*

\* Session types are a type discipline for communication channel endpoints which allow conformance to protocols to be checked statically.

Probablement trop « futuriste » pour devenir mainstream rapidement

Impact RH : après 10 ans à séparer dev front / back, il va falloir expliquer que c'était une mauvaise idée

# TAKE AWAY

## UN SUJET ORGANISATIONNEL & TECHNIQUE

Les années 20's voient se généraliser les approches microfrontend (au moins dans les grands groupes)

D'abord un sujet d'organisation, aboutissement de l'agilité/lean dans la DSI

Augmente la complexité : il faut maîtriser les architectures front-back techniquement pour pouvoir opérer une architecture microfrontend ...  
... à moins de voir croître les app tierless

On voit aussi les entreprises qui n'ont pas traité le sujet organisationnel faire machine arrière



# THE BOSS

SI VOUS VOULEZ ALLER PLUS LOIN

Renforcement :

 [Long short story : micro-frontends.org](https://micro-frontends.org)

 [Microfrontend in action](#)

Diversification :

 [Sessions Types in programming languages](#)



# SIDE QUEST

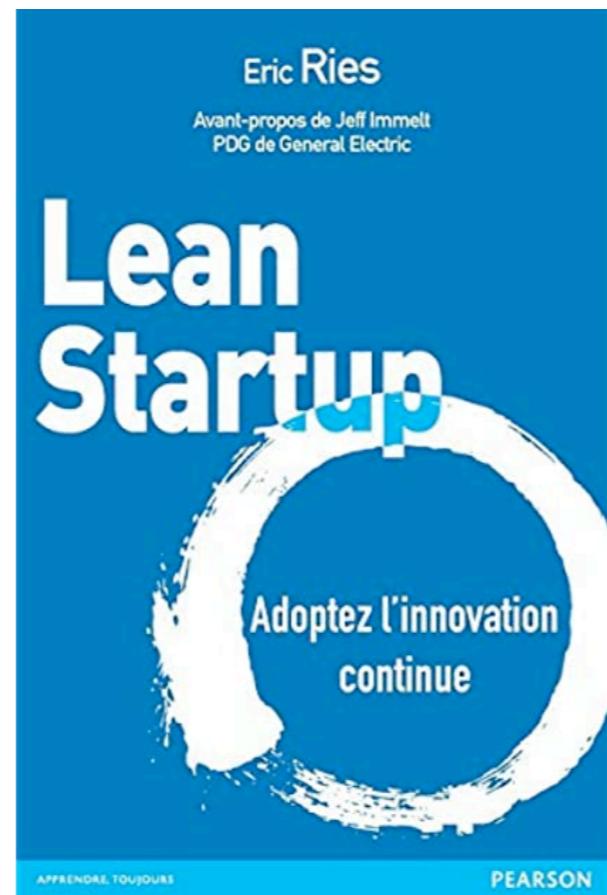
Fast learning tactics

Des actions : lire, pratiquer, créer son réseau

L'apprentissage commence après la MIAGE

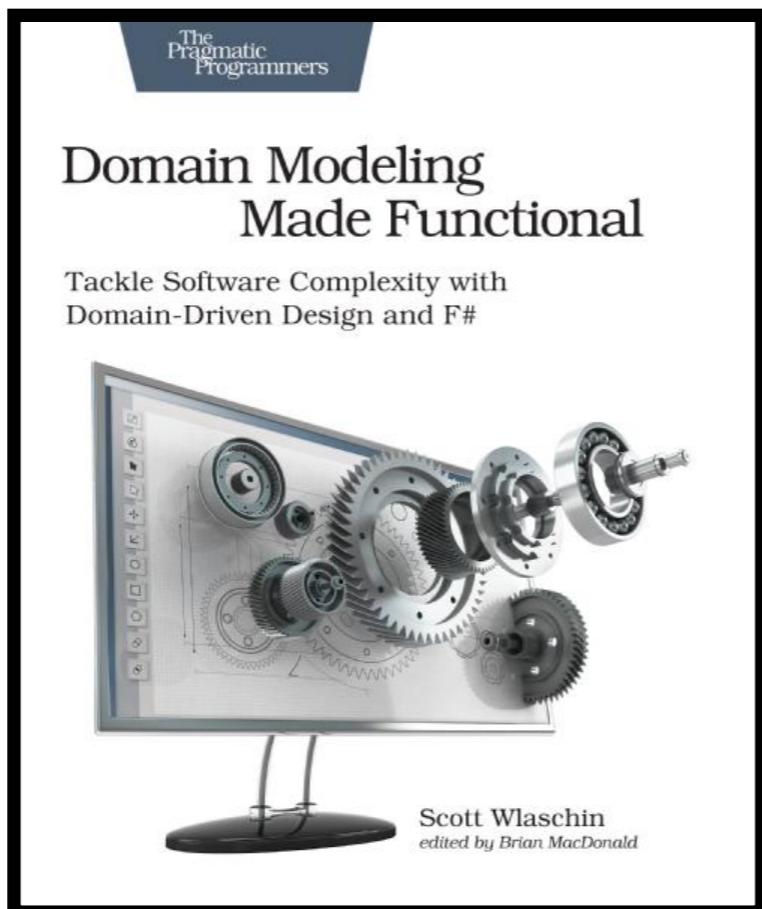


# PRODUCT MANAGER ENTREPRENEUR·E



<https://www.amazon.fr/Lean-Startup-Adoptez-linnovation-continue/dp/2744066400/>

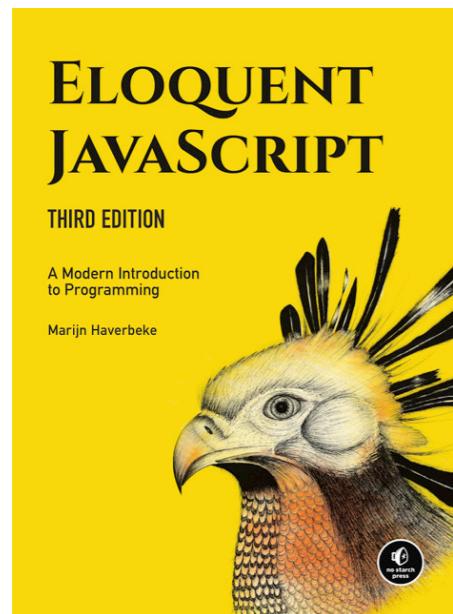
# PRODUCT OWNER CHEF·FE DE PROJET ARCHITECTE



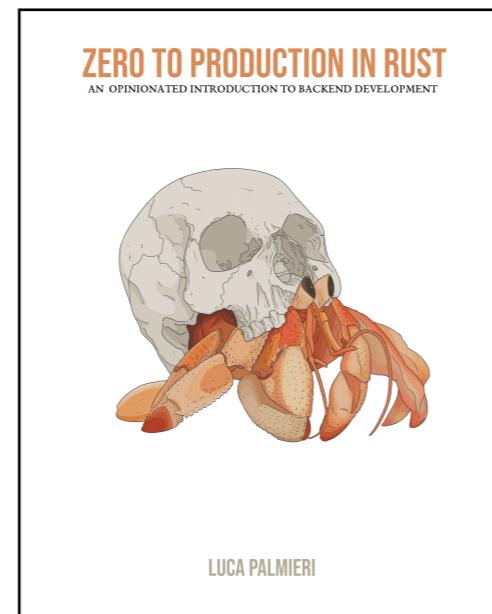
<https://pragprog.com/titles/swdddf/domain-modeling-made-functional/>

# DEVELOPPEUR·EUSE

Orienté sur un langage

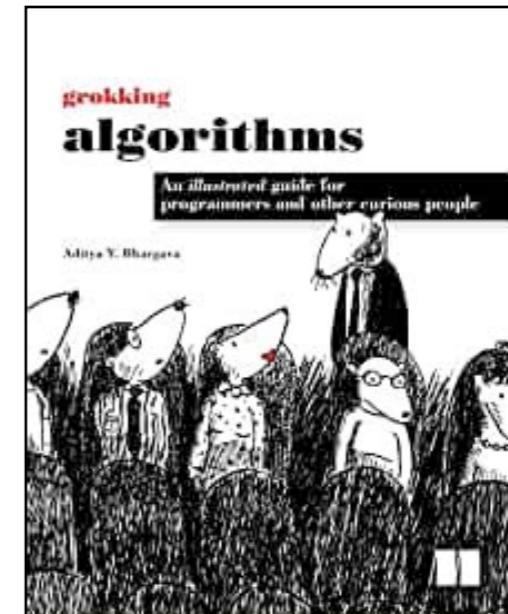


<https://eloquentjavascript.net/>



<https://www.zero2prod.com>

Générique



<https://www.manning.com/books/grokking-algorithms>

• • •

# DEVELOPPEUR·EUSE

S'entrainer et progresser



<https://www.codingame.com>



<https://www.codewars.com/>

Contribuer à des projets open source

- Good for 1st Issue
- Documentation

# NETWORK

## Meetups

- Frontend Beers : <https://www.meetup.com/fr-FR/frontendbeers/>
- Nord Agile: <https://www.meetup.com/fr-FR/nord-agile/>
- French Produit Nord : <https://www.meetup.com/fr-FR/frenchproduit-nord/>

## Conventions

- Dev Fest
- Agile Tour
- FOSDEM (Bruxelles)

# **VISIBILITÉ**

**CV -> LinkedIn**

**Portfolio -> Github**

# SIDE QUEST

Qualité du code

Raisonner sur un logiciel et à plus large échelle sur un SI est complexe

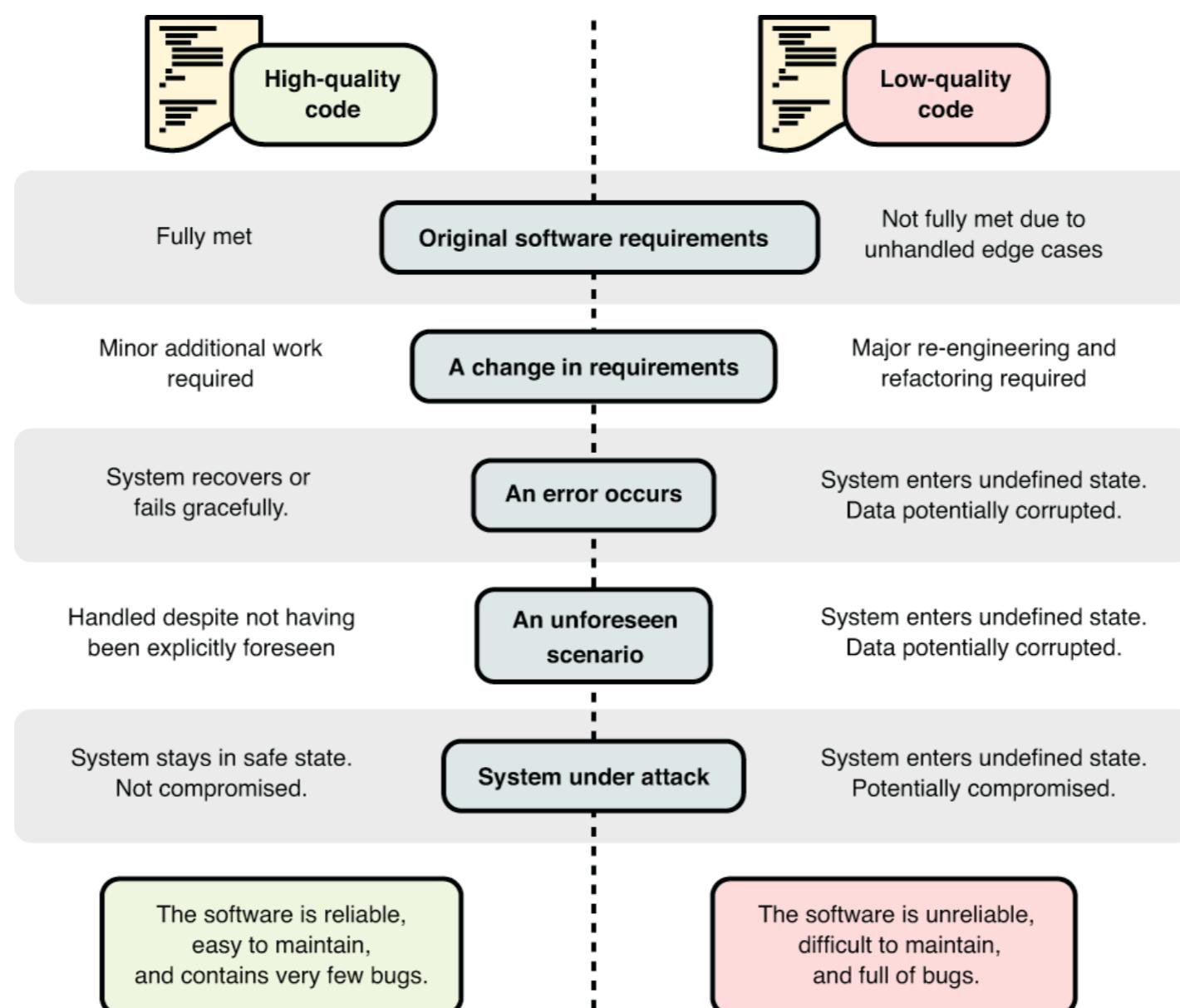
Jusqu'à présent nous nous sommes concentré sur des tactiques (ADT, Option/Result, PBT) pour résoudre des problèmes récurrents (machine états implicites, null pointer errors, exemples mal choisis, ...) ... mais spécifiques !

Quelles sont les stratégies d'architecture généralisables ?



# QUALITÉ

## QU'EST CE QUI CARACTÉRISE UN CODE DE QUALITÉ

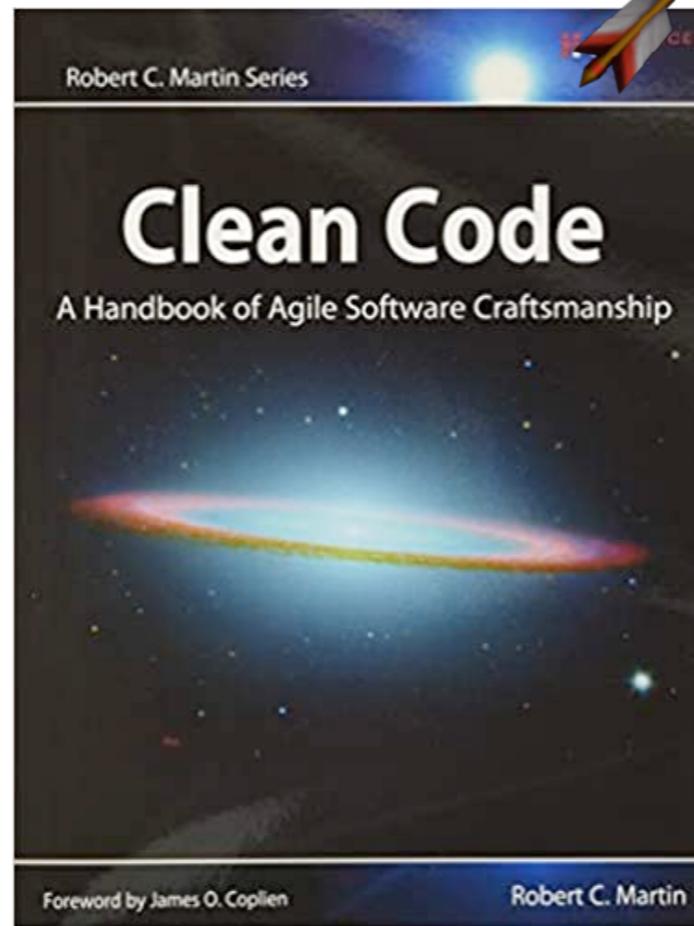


Extrait de *Good Code, Bad Code* par Tom Long

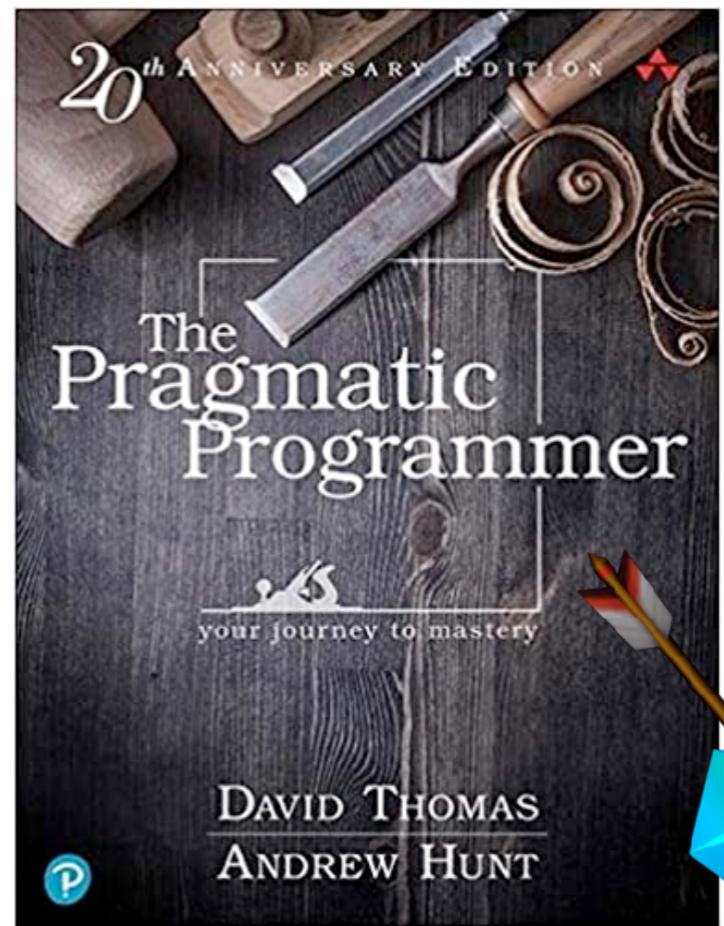
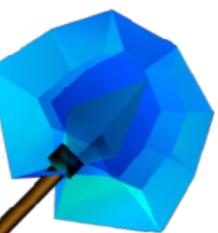
# GÉNIE LOGICIEL

QUELS SONT LES OUVRAGES DE RÉFÉRENCE

SOLID



CLEAN ARCHITECTURE

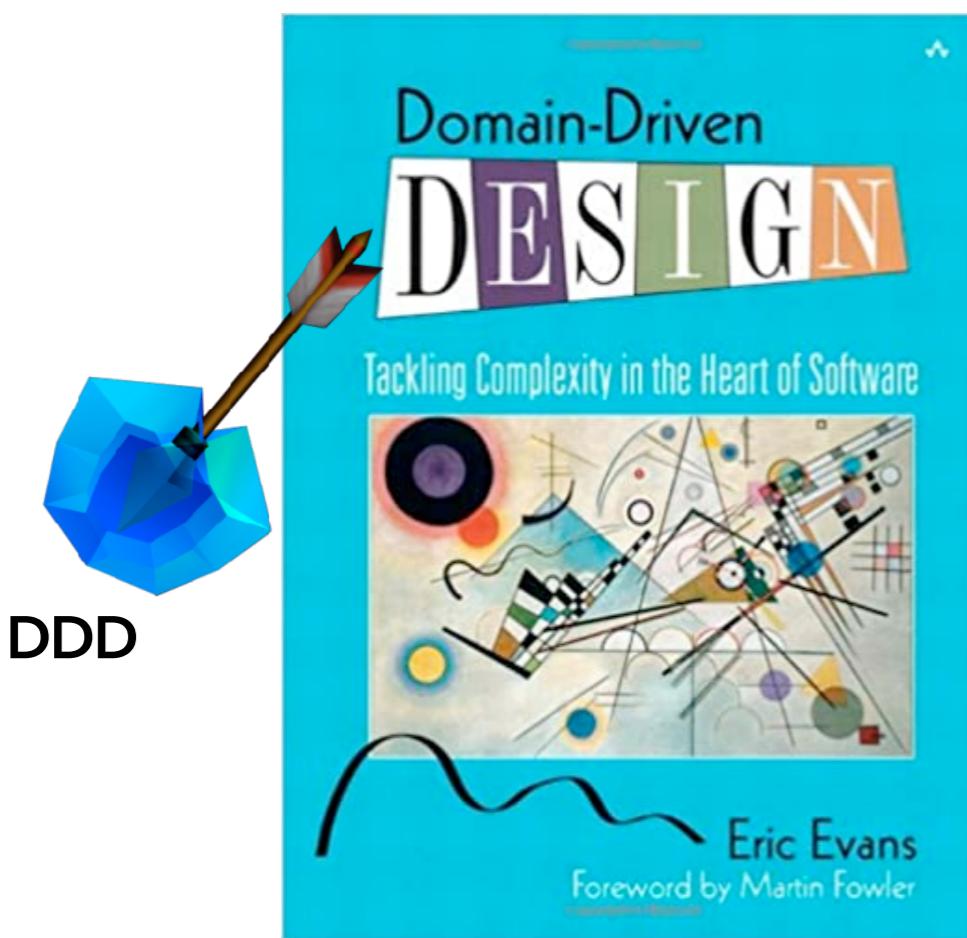


DRY

SONT CENTRÉS SUR DES EXEMPLES OOP

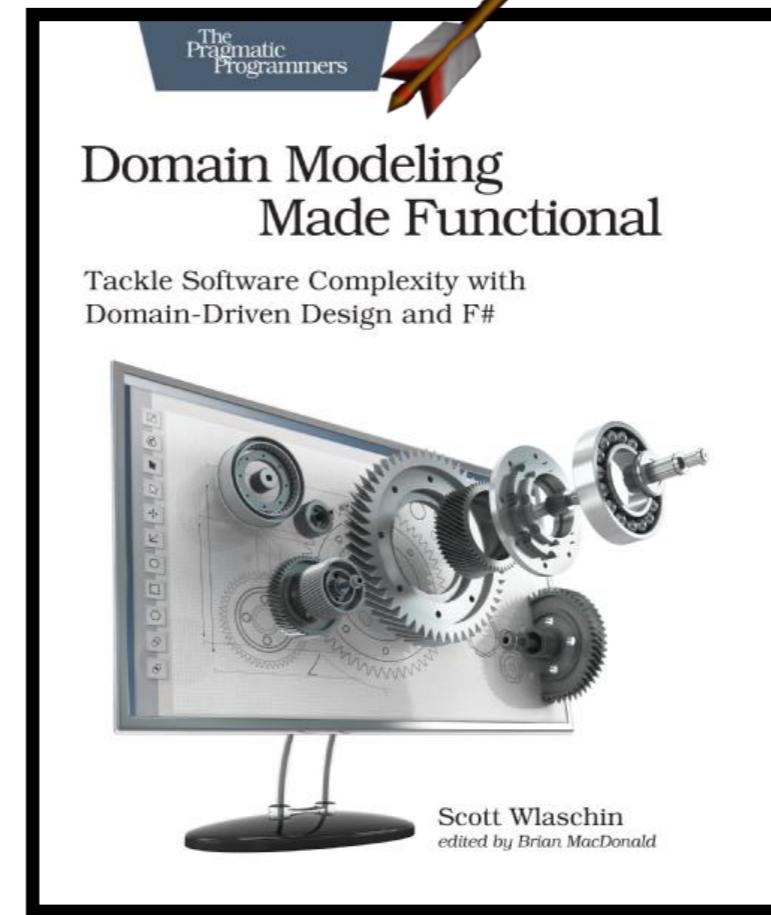
# ARCHITECTURE

QUELS SONT LES OUVRAGES DE RÉFÉRENCE



DDD

OOP



DDD + ONION

FP

# **LES IDIOMES LIMITENT LE GÉNIE LOGICIEL?**

**Spoiler : NON !**



# DRY

## DON'T REPEAT YOURSELF

« Every piece of knowledge must have a single, unambiguous, authoritative representation within a system », *Andy Hunt*

Cela se traduit dans les logiciels par:

- L'utilisation d'abstractions: **class, interface, package ... mais aussi type, module, typeclass, trait, function, s-expression ...**
- La normalisation des données

## INDÉPENDANT DES IDIOMES

# SOLID

## SINGLE-RESPONSIBILITY PRINCIPLE

« There should never be more than one reason for a **class** to change. »,  
*Robert C. Martin a.k.a Uncle Bob*

Peut être modifiée en

« There should never be more than one reason for an **abstraction** to change. »

Cela se traduit dans les logiciels par:

- Créer des abstraction qui ont un faible couplage entre elles
- Séparer les données et les comportements: *Visitor pattern, Iterators, Functional programming, Modular programming*

INDÉPENDANT DES IDIOMES

# SOLID

## OPEN-CLOSED PRINCIPLE

« Software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification », *Bertrand Meyer*

« You should be able to extend the behavior of a system without having to modify that system. », *Uncle Bob*

Indissociable du S, cela se traduit dans les logiciels par:

- La définition de contrats publics / implémentations privées
- L'extension grâce au polymorphisme : sous-typage, paramétrique (a.k.a générique) ou ad-hoc

## INDÉPENDANT DES IDIOMES

# SOLID

## (BARBARA) LISKOV SUBSTITUTION PRINCIPLE

« *Subtype Requirement*: Let  $\phi(x)$  be a property provable about objects  $x$  of type T. Then  $\phi(y)$  should be true for objects  $y$  of type S where S is a subtype of T. », Barbara Liskov

Cette propriété vise à garantir l'interopérabilité sémantique des types dans une hiérarchie de type:

- La définition de contrats publics / implémentations privées
- L'extension grâce au polymorphisme : sous-typage, paramétrique (a.k.a générique) ou ad-hoc

## INDÉPENDANT DES IDIOMES

# SOLID

## INTERFACE SEGREGATION PRINCIPLE

« *No client should be forced to depend on methods it does not use* », Robert C. Martin a.k.a Uncle Bob

Indissociable du S et du L, le I vise la suppression des « God Classes », cela se traduit par :

- L'extension grâce au polymorphisme : sous-typage, paramétrique (a.k.a générique) ou ad-hoc
- Les capacités sont décrites dans des abstractions
- Ces abstractions sont le plus limitées possibles

## INDÉPENDANT DES IDIOMES

# SOLID

## DEPENDENCY INVERSION PRINCIPLE

Vise toujours à atteindre un couplage faible dans les logiciels.

Les abstractions de « haut niveau » ne doivent pas dépendre des abstractions « bas niveau »

Les abstractions ne doivent pas dépendre de détails, mais les détails d'implémentation doivent dépendre des abstractions.

Indissociable du I, cela se traduit par :

- La mise en oeuvre de l'injection de dépendance

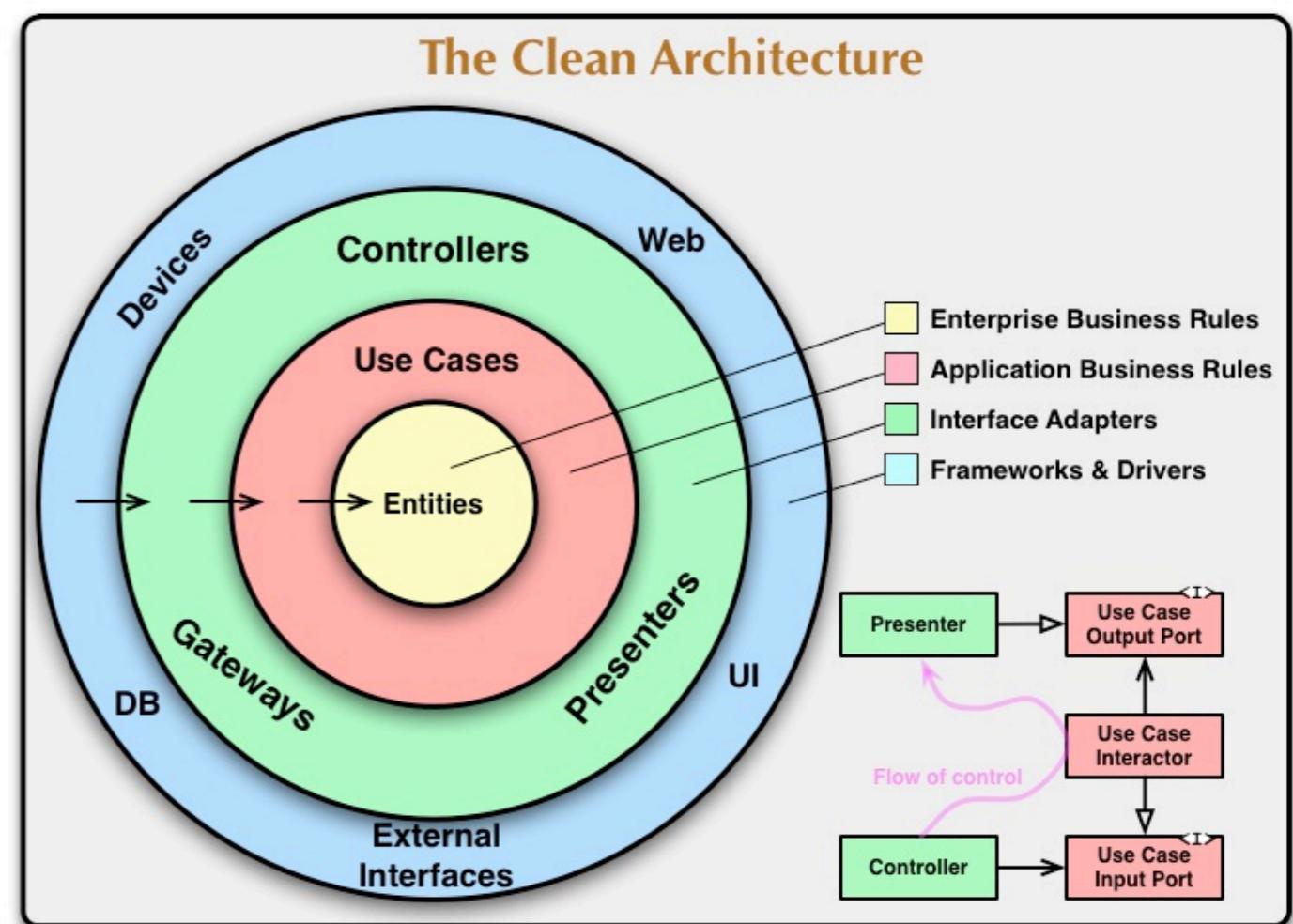
## INDÉPENDANT DES IDIOMES

# CLEAN / ONION

UNCLE BOB ARCHITECTURE / JEFFREY PALERMO ARCHITECTURE

Objectifs est d'avoir un logiciel :

- Indépendant des frameworks
- Testable
- Indépendant de l'UI
- Indépendant de la DB
- Indépendant des API externes



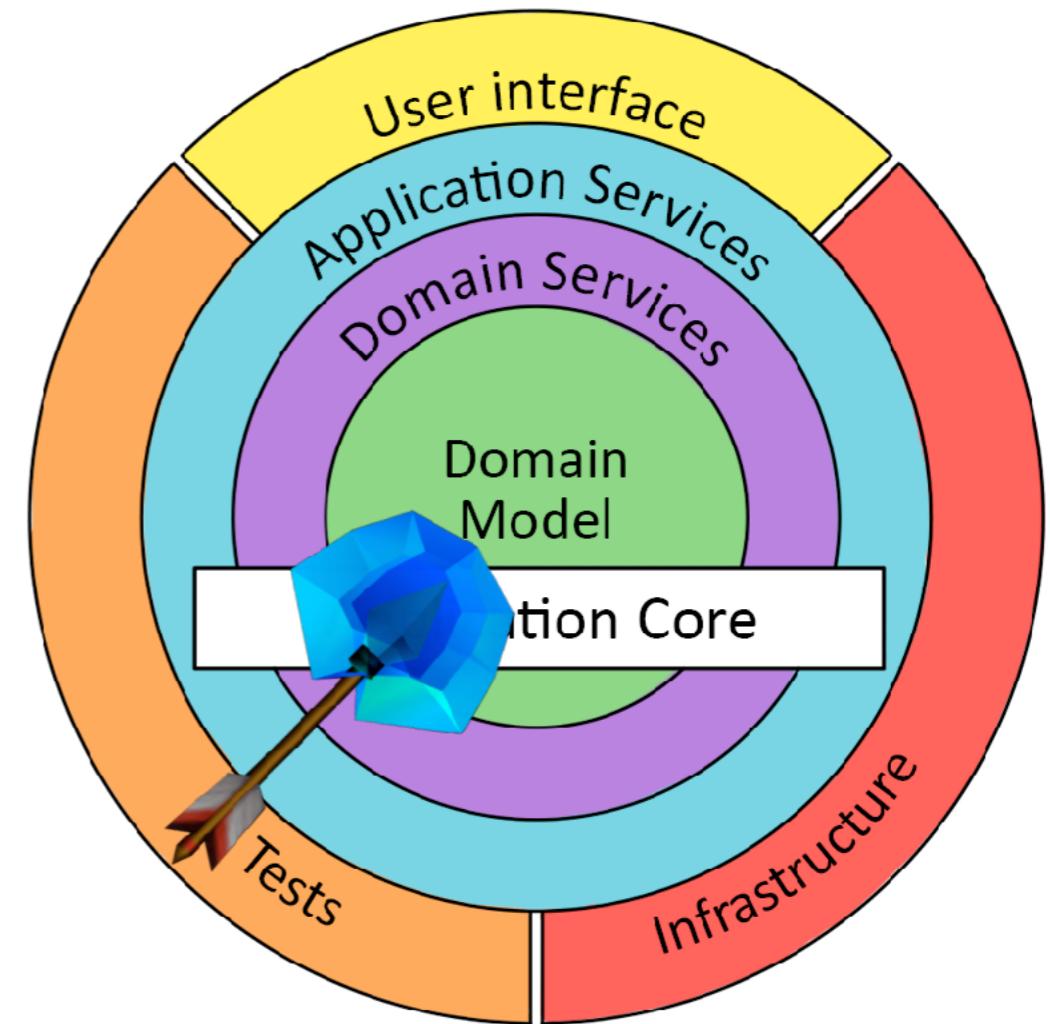
BREF : DRY + SOLID + EFFETS AUX FRONTIÈRES DU PROGRAMME

# DDD

## DOMAIN DRIVEN DESIGN - A.K.A ONION MADE BUSINESS FRIENDLY

Une approche top-down :

- Le périmètre d'un logiciel (service) est borné par un domaine business
- Ubiquitous langage (celui du business)
- En appliquant SOLID
- Dans une onion architecture



TOUJOURS INDÉPENDANT DES IDIOMES

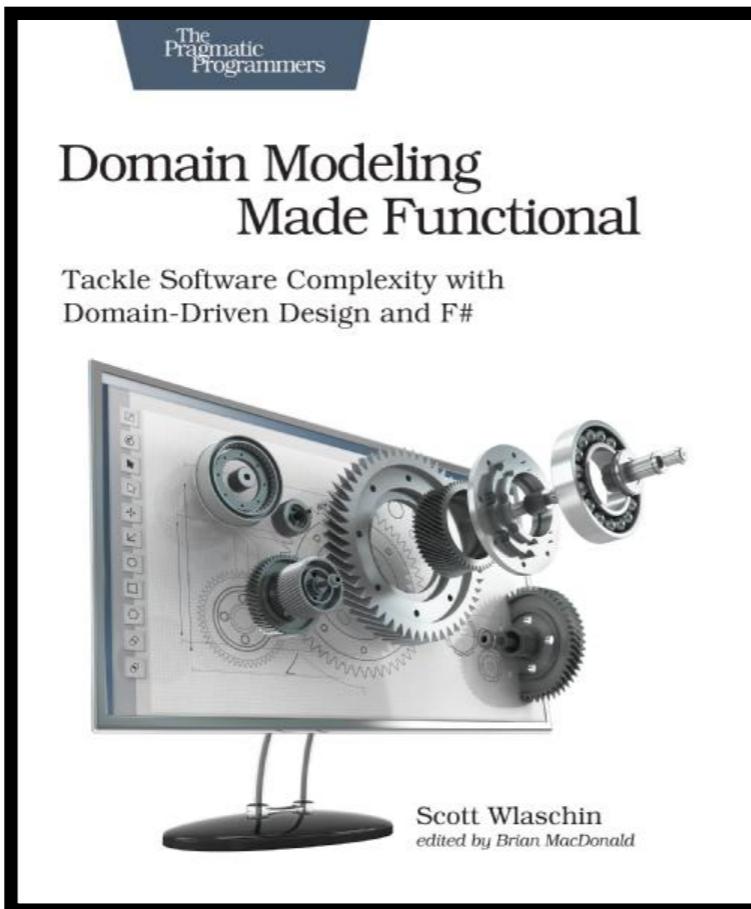
# FINAL BOSS

LISEZ ! A METTRE ENTRE TOUTES LES MAINS

**Product Owner :**

**Architectes :**

**Chefs de projets :**



<https://pragprog.com/titles/swdddf/domain-modeling-made-functional/>

