

# QUALITÉ DU SI

Sommaire

[\*Télécharger en version PDF\*](#)

Intro

Quête des machines états

Langages & Property based Testing

Gestion d'erreurs & qualité du code

Frontends dataflow

Cloud & microservices

Communication Inter-process

Decentralized web

Final Boss

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

**QUALITÉ D'USI**

**IMAGE - INTRODUCTION**

# DISCLAIMER 1

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

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

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

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

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

# **DISCLAIMER 2**

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

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

**Ce n'est pas contradictoire**

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

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

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

# **DISCLAIMER 3**

**ÇA VA ÊTRE DENSE MAIS GUIDÉ**

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

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

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

**7 Cours/TP incluant 3 Kata, 3 TP « socles » d'un mini-projet**

**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

## La qualité optimale

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

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

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



# CALENDRIER 2022

## PRÉVISIONNEL

4/01: Intro & quête des machines états - Tennis Kata

11/01: Langages & Property based Testing - Troll of Fame Kata

18/01: Gestion d'erreurs & qualité du code - DnD Kata

25/01: Frontends dataflow - Catstagram Kata

01/02: Cloud & microservices - FINALISEZ LES TPS PRECEDENTS

22/02: Communication Inter-process - TP auth-service

01/03: Communication Inter-process - Dapp Kata

08/03: Présentation du Mini-projet -> Livraison le 30/03 à minuit au plus tard.

# 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

## 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 menée de concert**

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

# **QUALITÉ DU SI**

---

**COURS 1 - 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 !!!

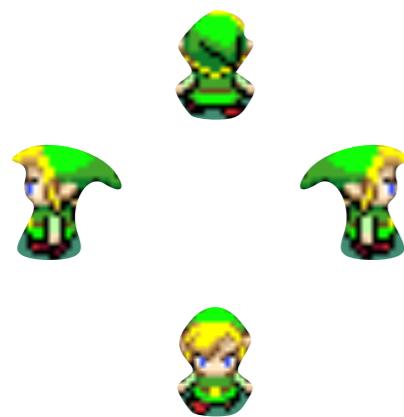
Peut-on améliorer la conception logicielle,  
puis SI pour éviter cela ?

OUI c'est le but de ce cours



# LINK TO THE PATH

## IMPLÉMENTATION NAÏVE EN JAVA



```
class Direction {  
    public static final int North = 1;  
    public static final int East = 2;  
    public static final int South = 3;  
    public static final int West = 4;  
}  
  
public class Main {  
    public static final String label(int d) {  
        switch (d) {  
            case Direction.North : return "north";  
            case Direction.East : return "east";  
            case Direction.South : return "south";  
            case Direction.West : return "west";  
        }  
        throw new IllegalArgumentException("not a valid direction");  
    }  
  
    public static void main(String args[]) {  
        System.out.println(label(1));  
        System.out.println(label(Direction.East));  
        System.out.println(label(Direction.South));  
        System.out.println(label(4));  
    }  
}
```

# LINK TO THE PATH

## IMPLÉMENTATION NAÏVE EN JAVA



```
public static void main(String args[]) {  
    System.out.println(label(-35));  
}
```

Runtime ERROR

Exception in thread "main" java.lang.IllegalArgumentException: not a valid direction

-35 est un Int valid mais pas une Direction valide : peut-on faire mieux ?

# LINK TO THE PATH

## UTILISATION DES ENUM EN JAVA



```
enum Direction { North, East, South, West }

public class Main {
    public static final String label(Direction d) {
        switch (d) {
            case North : return "north";
            case East : return "east";
            case South : return "south";
            case West : return "west";
        }
        throw new IllegalArgumentException("not a valid direction");
    }

    public static void main(String args[]) {
        System.out.println(label(Direction.East));
    }
}
```

# LINK TO THE PATH

## UTILISATION DES ENUM EN JAVA



```
public static void main(String args[]) {  
    System.out.println(label(-35));  
}
```

Compilation ERROR

```
error: incompatible types: int cannot be converted to Direction  
System.out.println(label(-35));
```

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

# LINK TO THE PATH

## UTILISATION DES ENUM EN JAVA ... MAIS EN MODIFIANT LE CODE



Indice sur le problème

```
enum Direction { North, East, South, West }

public class Main {
    public static final String label(Direction d) {
        switch (d) {
            case North : return "north";
            case East : return "east";
            case South : return "south";
            //case West : return "west";
        }
        throw new IllegalArgumentException("not a valid direction");
    }

    public static void main(String args[]) {
        System.out.println(label(Direction.East));
        System.out.println(label(Direction.West));
    }
}
```

Runtime ERROR

Exception in thread "main" java.lang.IllegalArgumentException: not a valid direction

Le compilateur ne sait pas détecter si nous avons traité tous les cas !!!

# LINK TO THE PATH

## ON EST EN OOP : ON A DU POLYMORPHISME PAR HÉRITAGE



Le polymorphisme par héritage donne une garantie sur le fait de toujours avoir une implémentation de `toString`

Il ne fournit aucune garantie sur l'exhaustivité du domaine : on pourrait ajouter une classe NordEast par exemple ! Ou TripleBackflip !

Connaitre l'ensemble des implémentations peut être complexe (impossible avant Java 17)

```
interface Direction{
    public int toInt();
    public String toString();
};

class North implements Direction{
    public int toInt(){ return 1;};
    public String toString(){ return "north";};
};

class East implements Direction{
    public int toInt(){ return 2;};
    public String toString(){ return "east";};
};

class South implements Direction{
    public int toInt(){ return 3;};
    public String toString(){ return "south";};
};

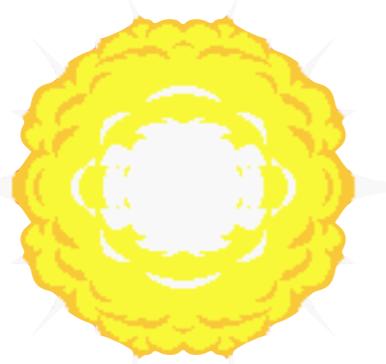
class West implements Direction{
    public int toInt(){ return 4;};
    public String toString(){ return "west";};
};

public class Main {
    public static final String label(Direction d) {
        return d.toString();
    }

    public static void main(String args[]) {
        System.out.println(label(new East()));
        System.out.println(label(new West()));
    }
}
```

# LINK TO THE PATH

## QUE SE PASSE-T-IL SI ON RÉCUPÈRE LES DIRECTIONS D'UNE LIB



Les encodages initiaux sont complexes à maintenir!

*Composition over inheritance: "Favor 'object composition' over 'class inheritance'."*  
*(Gang of Four 1995:20)*

```
/** lib tierce */
interface Direction{
    public int toInt();
};

class North implements Direction{
    public int toInt(){ return 1;};
};

class East implements Direction{
    public int toInt(){ return 2;};
};

class South implements Direction{
    public int toInt(){ return 3;};
};

class West implements Direction{
    public int toInt(){ return 4;};
};

/** notre code */
interface DirectionImprove{
    public String toString();
};

class NorthImprove extends North implements DirectionImprove{
    public String toString(){ return "north";};
};

class EastImprove extends North implements DirectionImprove{
    public String toString(){ return "east";};
};

class SouthImprove extends North implements DirectionImprove{
    public String toString(){ return "south";};
};

class WestImprove extends North implements DirectionImprove{
    public String toString(){ return "west";};
};
```

# LINK TO THE PATH

EN PASSANT PAR 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)

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

... mais incomplet (pas de support des primitives float, boolean, double), pas de pattern de déconstruction dans le case (prévu en Java 18)

... Vous n'êtes pas prêt de voir ça en production

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

# LA MINUTE HINOX

... VOUS POUVEZ DÉJÀ UTILISER

Java 20 a.k.a Kotlin

Java 30 a.k.a Scala

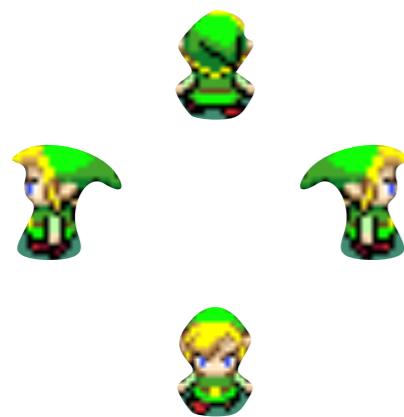
Java 100 a.k.a OCaml



# LINK TO THE PATH

UTILISATION D'UN TYPE « OU » DANS UN SYTÈME DE TYPE ML

MetaLangage (1970) trouve toi un acronyme original Machine Leaning



```
type direction = | North | East | South | West

let label d = match d with
| North -> "north"
| East -> "east"
| South -> "south"
| West -> "west"

let _ = print_endline(label North);;
let _ = print_endline(label East);;
let _ = print_endline(label South);;
let _ = print_endline(label West);;
```



# APARTÉ



## POURQUOI OCAML

French tech : made in INRIA

Qui l'utilise ? Facebook, Docker, Citrix, Tarides, Bloomberg, Janestreet, LexiFi, ANSSI, Dassault Systèmes, Shiro Games, DGFiP, Airbus, Microsoft, Tezos : (Nomadic Labs, Marigold, Trilli), ...

Très utilisé en finance, blockchain, infrastructure management, sécurité, compilateurs

Multi paradigme : fonctionnel, impératif, objet

Multi target : natif, byte code, js, unikernel ... php, java, Erlang

Un ML : très utile pour ce que je veux illustrer et inspiration pour des langages modernes (Scala, Kotlin, Swift, Rust)

Vous en avez fait en M1

... Le meilleur langage du monde

... Les mauvaises langues diront qu'il manque de Higher Kinded Types ... et c'est vrai

# LINK TO THE PATH

UTILISATION D'UN TYPE « OU » DANS UN SYTÈME DE TYPE ML



```
let _ = print_endline(label -35);;
```

Compilation ERROR

Error: This expression has type direction -> string  
but an expression was expected of type int

On peut valider dès la compilation qu'on utilise UNIQUEMENT des valeurs valides !!!

# LINK TO THE PATH

## UTILISATION D'UN TYPE « OU » DANS UN SYTÈME DE TYPE ML



```
let label d = match d with
| North -> "north"
| East -> "east"
| South -> "south"
(* / West -> "west" *)
```

### Compilation WARNING

Warning : this pattern-matching is not exhaustive.  
Here is an example of a case that is not matched:  
West

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

Warn par défaut mais on peut en faire une erreur bloquant la compilation en ajoutant le paramètre `-warn-error -a+31+8`

# 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 : 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;
}

let turn_east = {
    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 »



```
let yolo = {  
    order="triple_backflip";  
    direction=East;  
}
```

# MODÉLISER LA COMMANDE

ON A DÉJÀ VU COMMENT RÉSOUTRE 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 »



```
let no_sense = {  
    order=Start;  
    direction=East;  
}
```

# MODÉLISER LA COMMANDE

ORDRE AND TYPE DE PARAMÈTRE

```
type direction = | North | East | South | West  
type command = | Face of direction | Start | Stop
```

# MODÉLISER LA COMMANDE

COMMAND : UN TYPE « OU »



```
let stop = Stop
```

# 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

**Start** est un constructeur du type command à 0 paramètre => Start est une valeur

**Face of direction** est un constructeur du type command à 1 paramètre => Face East est une valeur



# SCRIPT DE COMMANDE

ENCHAINER LES COMMANDES AVEC UN TYPE OU RÉCURSIF



```
type direction = | North | East | South | West

type command =
| Face of direction
| Start
| Stop
| Chain of command * command

let move_east = Chain(
  Face East, Chain (
    Start, Stop
  )
)
```

# SCRIPT DE COMMANDE

## UNE COMMANDE PLUS COMPLEXE

```
let move_east =
  Chain(
    Chain(Face East,
      Chain (
        Start, Stop
      )),
    Chain(Face South,
      Chain (
        Start, Stop
      ))
  )
```

Est complexe à lire

# SCRIPT DE COMMANDE

ÉCRIVONS UN DSL



```
let (--)> cmd1 cmd2 = Chain(cmd1, cmd2)

let _ = Face East
    --> Start
    --> Stop
    --> Face South
    --> Start
    --> Stop
```

# TAKE AWAY

## NOUS AVONS VU

Le type `command * command` se lit **commande ET commande**, c'est un tuple, donc un type « ET »

Le type `command` est récursif : s'exprime en terme de **Chain of command \* 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



```
let _ = Face East  
      --> Stop
```

# SUIVRE LES TRANSITION

## DESCRIPTION D'UN GADT

```
type direction = | North | East | South | West
type idle
type moving
type ('before, 'after) command =
| Face : direction -> (idle, idle) command
| Start : (idle, moving) command
| Stop : (moving, idle) command
| Chain : ('a, 'b) command * ('b, 'c) command -> ('a, 'c) command

let (--) cmd1 cmd2 = Chain(cmd1, cmd2)
```

On paramètre le type **command** avec 2 paramètres de type polymorphes (similaire aux génériques) '**before** et '**after**

On décrit explicitement le type de chaque constructeur de type **command** avec des types témoins

Face **prend en paramètre une** direction **et donne une valeur de type** (idle, idle) command

Start **est une valeur de type** (idle, moving) command

Stop **est une valeur de type** (moving, idle) command

Chain **permet maintenant de composer une** ('a, 'c) command **à partir d'un tuple** ('a, 'b) command \* ('b, 'c) command

# SUIVRE LES TRANSITION

## DESCRIPTION D'UN GADT

```
let _ = Start --> Start
```

### Compilation ERROR

Error: This expression has type (idle, moving) command  
but an expression was expected of type (moving, 'a) command  
Type idle is not compatible with type moving

```
let _ = Face West --> Start --> Stop
```

On peut créer uniquement des commandes valides

# TAKE AWAY

## NOUS AVONS VU

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

Ils permettent (entre autre) de valider à la compilation les transitions d'états

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

Nous avons une machine à états finis, validée par compilation ❤️❤️❤️

Seulement en Haskell, OCaml, Scala



# TENNIS KATA



# 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 2 - LANGAGES & PROPERTY BASED TESTING**

# THE QUEST

Comment et pourquoi choisir un langage ?

En terme de popularité ?

En terme d'employabilité ?

En terme de paradigme ?

En terme d'exploitation ?

En terme de connaissances acquises ?

En terme de connaissances à acquérir ?

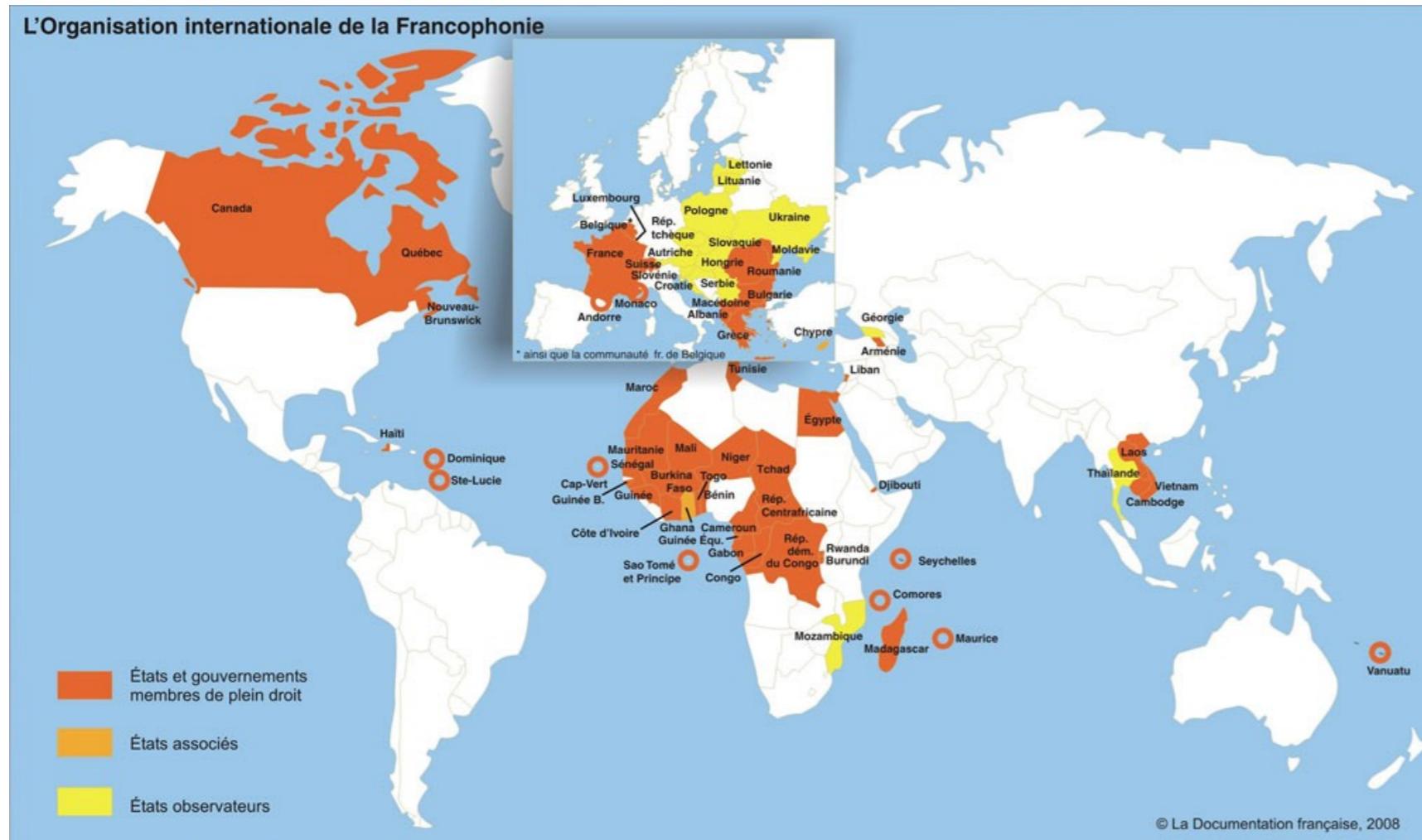
Tout développeur devrait se poser ces questions !

Peut-on se passer de développeurs (no-code) ?



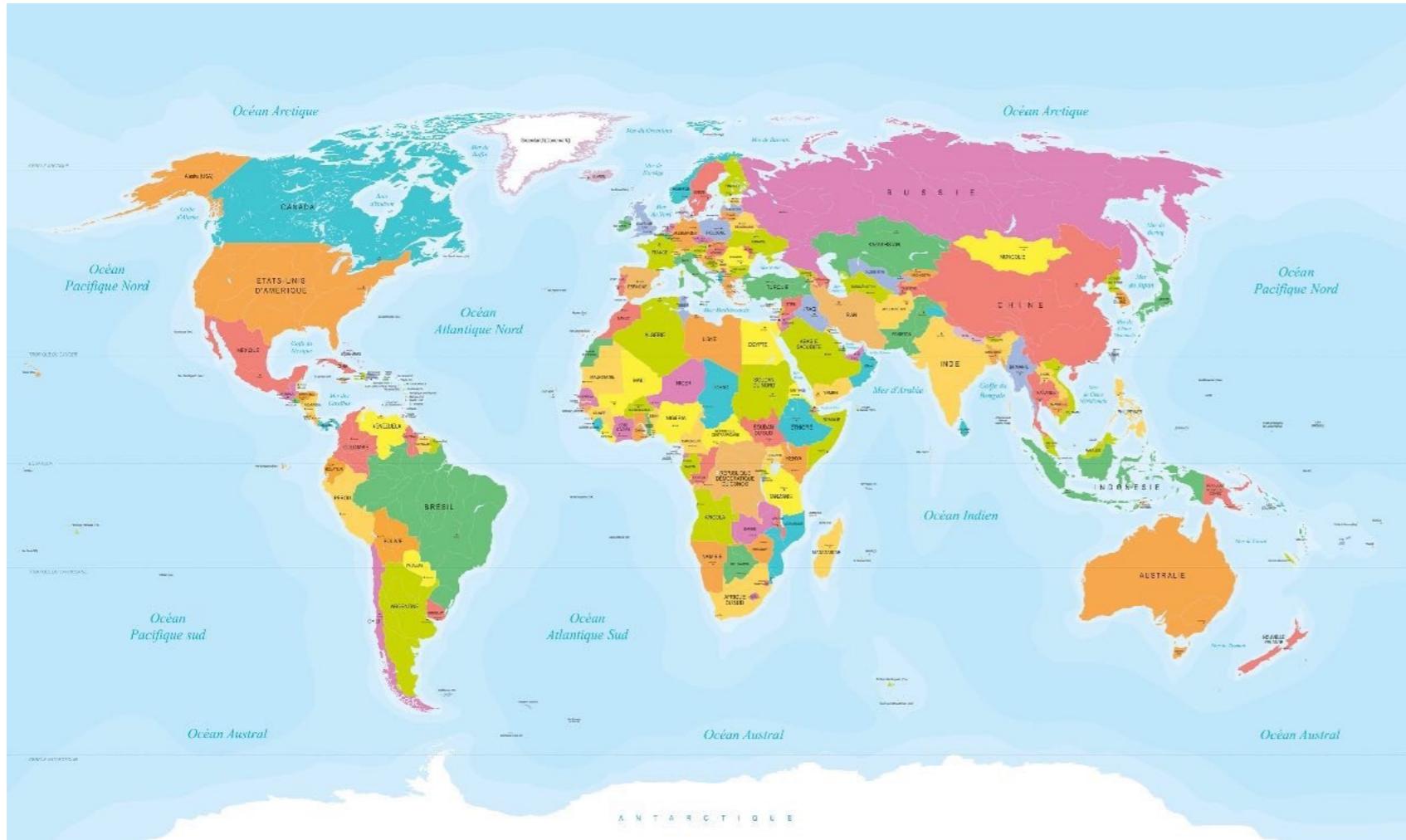
# PARLEZ VOUS FRANÇAIS ?

OU BIEN : Belge ? Suisse ? Sénégalais ? Québécois ? Congolais ?



# LE MONDE EST VASTE

7 000 langues



**Anglais 1ère L2 / 3ème L1**

**Mandarin 5ème L2 / 1ère L1**

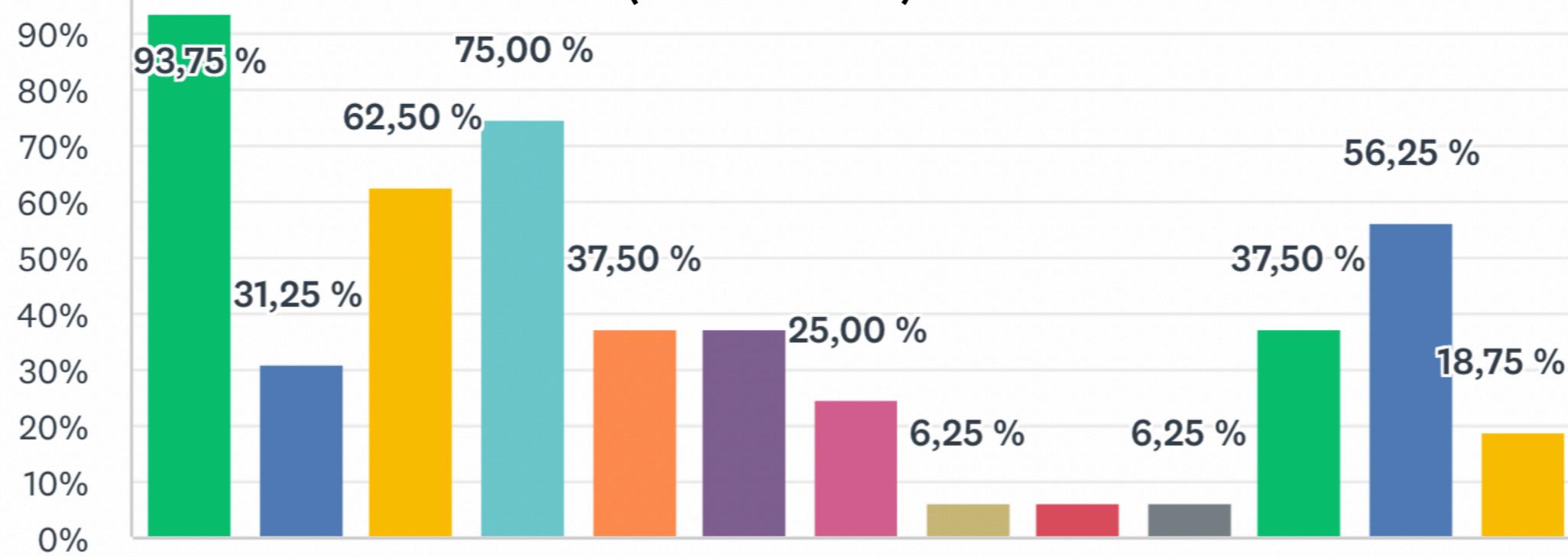
**Français 3ème L2 / 15ème L1**

**Swahili 9ème L2 / 31ème L1  
(la plus parlée en Afrique de l'est)**

**Esperanto uniquement L2,  
1M locuteurs (c'est peu),  
pourtant utilisé par Croix-Rouge**

# ETAT DE LA PROMO

VOUS VOUS ESTIMEZ AUTONOMES (DÉCLARATIF)



Java, C#, Python, javascript, TypeScript, PHP, Scala, Rust, Go, C++, Kotlin, PL/SQL, Autre (veuillez préciser...)

Tous ces langages sont des dialectes d'une même langue : C++

En moyenne vous êtes autonomes sur 4 langages et 2 idiomes !

# **REMEMBER THE M1**

C3P - 3 IDIOMES

**Programmation Fonctionnelle (FP) : OCaml**

**Programmation Procédurale : C**

**Programmation Orientée Object (OOP) : Python**

# REMEMBER THE MIAGE

VOUS AVEZ QUAND MÊME VU D'AUTRES LANGAGES

Programmation Fonctionnelle (FP) : OCaml, Haskell

Programmation Procédurale : C, bash

Programmation Orientée Object (OOP) : Python, Java, PHP, Javascript, Blockly

# REMEMBER THE MIAGE

VOUS AVEZ QUAND MÊME VU D'AUTRES LANGAGES

Programmation Fonctionnelle (FP) : OCaml, Haskell

Programmation Procédurale : C, bash

Programmation Orientée Object (OOP) : Python, Java, PHP, Javascript, Blockly

Où range-t-on SQL ?

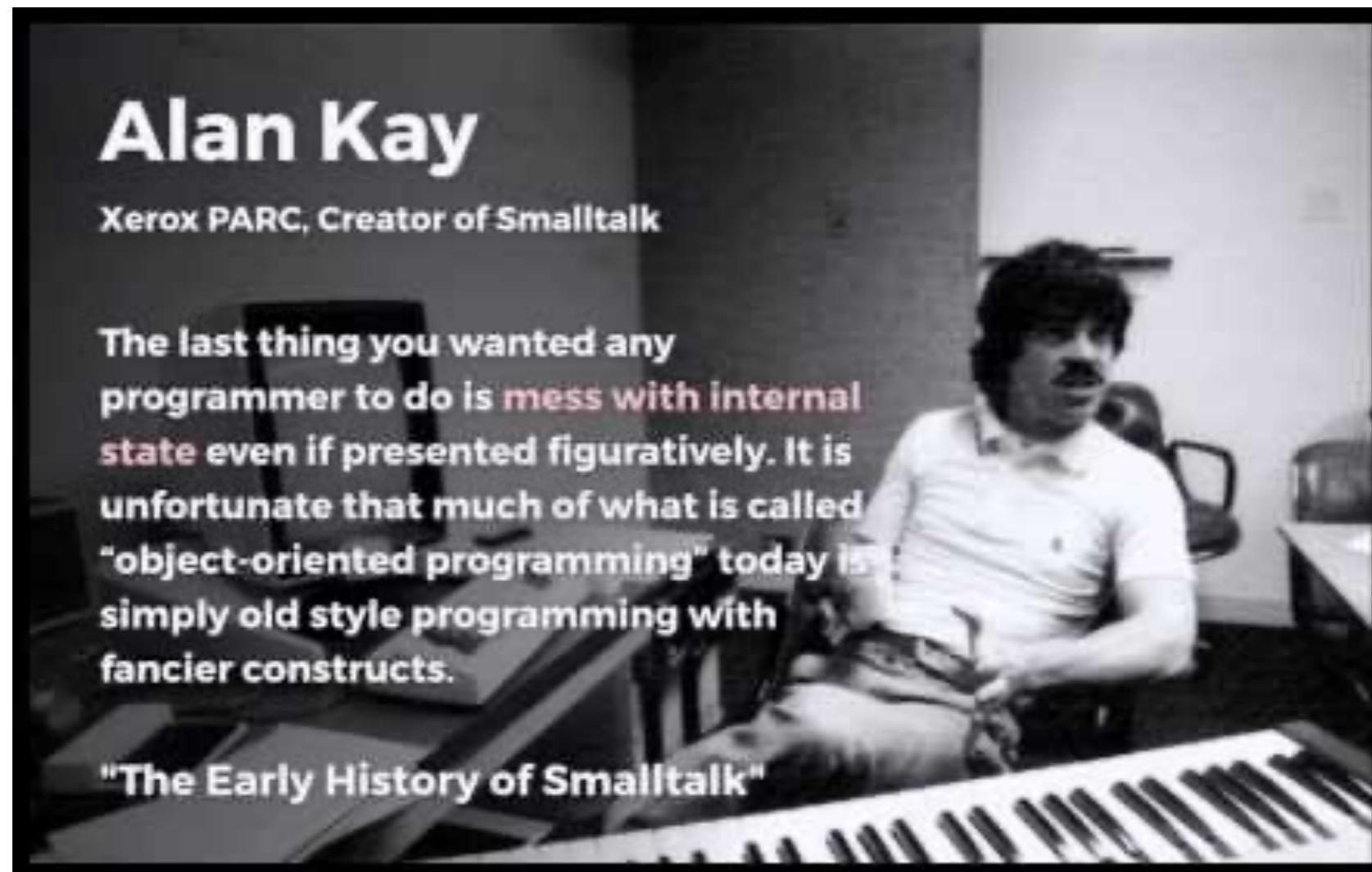
Où range-t-on les assembleurs ?

L'OOP Javascript est très différente des autres

Cette vision n'est pas suffisante !

# C'EST QUOI L'OOP ?

QUE DIT LE PÈRE DE L'OOP ... DÈS LES ANNÉES 80

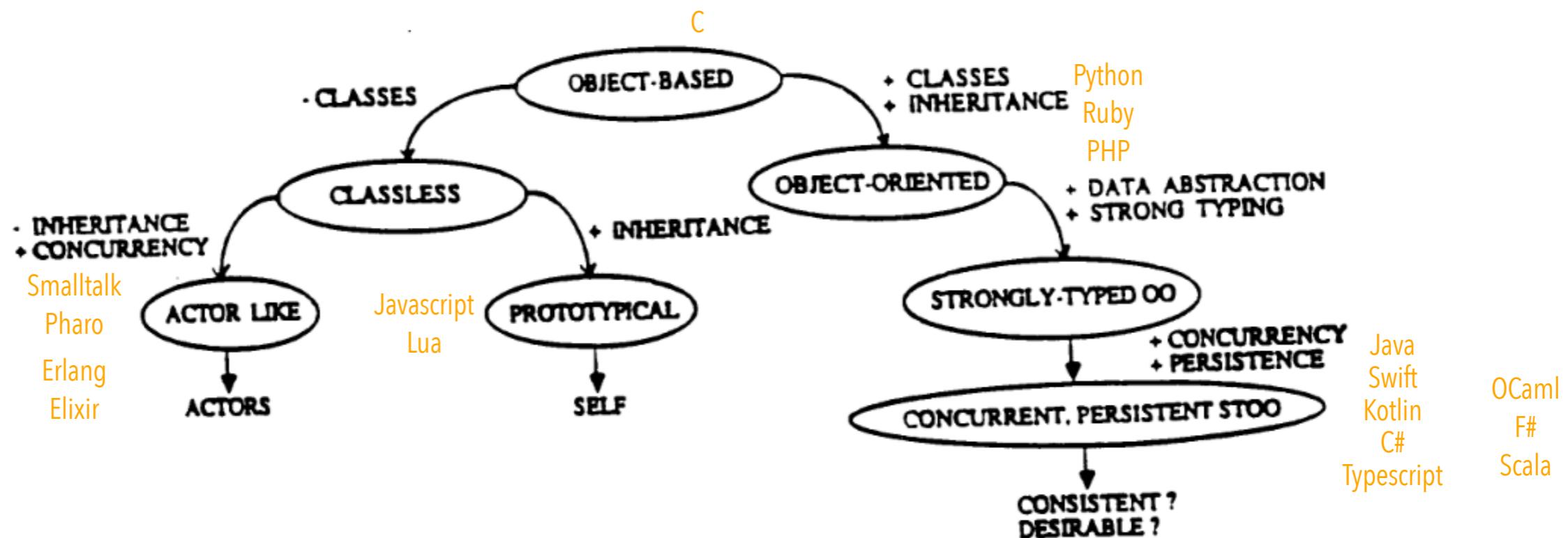


Java et Python ne seraient pas des langages orientés objets 😠

# C'EST QUOI L'OOP ?

Pas de consensus académique réel

La meilleure que j'ai trouvée (Peter Wegner, 1989)



Source : <https://pdfs.semanticscholar.org/48a6/7e434d764769ad66eddd8c4989364a88d708.pdf>

# C'EST QUOI LA FP ?

Question posée à un développeur

Disposer de fonctions d'ordre supérieur

= une fonction est une valeur

=> une fonction peut prendre en arguments des fonctions OU retourner une fonction

Programmer avec des fonctions :

- **TOTALES** (toute valeur unique dans le type des arguments a un et un seul résultat dans le type du résultat)
- **DÉTERMINISTES** (les mêmes arguments produisent toujours le même résultat)

Nécessite donc d'avoir des *Lambdas* (fonctions anonymes) et des *Closures* (permet l'*application partielle*)

Certains langages favorisent la FP : OCaml, F#, Scala, Clojure, Erlang/Elixir, ...

La plupart des langages permettent la FP : JS, Python, Java, C++, C#, Kotlin, Swift, ...

# C'EST QUOI LA FP ?

Question posée à un développeur HASKELL

Programmer avec des fonctions :

- **TOTALES** (toute valeur unique dans le type des arguments à un et un seul résultat dans le type du résultat)
- **DETERMINISTES** (les mêmes arguments produisent toujours le même résultat)
- **PURES** (son évaluation n'a pas d'effet de bord)

Un logiciel pur n'a pas beaucoup d'utilité business, cela nécessite donc d'avoir une évaluation paresseuse et un moyen de produire les effets à l'exécution, comme Monade IO ou Effets Algébriques, d'en parler en QSI, si ça vous intéresse RDV à Lambda Lille 😊

Le code Haskell est pur à la compilation mais impur à l'exécution

# BREF ...

## FUNCTIONS



TOTAL  
(NOT PARTIAL)



DETERMINISTIC  
(NO RANDOMNESS)



PURE  
(NO SIDE EFFECT)



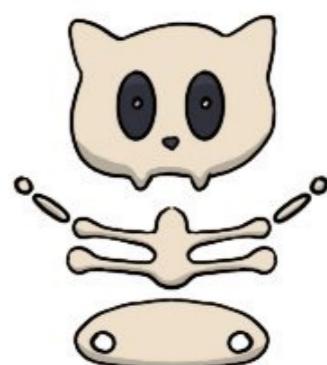
NO MUTATION



NO NULL



NO REFLECTION



NO EXCEPTION

Source : <https://impurepics.com/>

# PRINCIPAUX IDIOMES

**6 REQUIS + 3 DIVERSIFICATIONS + 3 SPÉCIFIQUES**

1 Langage impératif : C, Ada, Rust, Go

1 Langage objet statiquement typé : Kotlin, Swift, Java, C++, C#

1 Langage objet dynamiquement typé : PHP, ruby, python

1 Langage objet à prototype : Javascript, Lua

1 Langage ML : OCaml, Haskell

1 Langage orienté expression : SQL, LINQ

1 Lisp : Clojure, Racket, CommonLisp

1 Programmation parallèle : Erlang, Elixir

1 Objet orienté acteurs / messages : Smalltalk, Pharo

1 Assistant de preuve : Idris, Coq, F-star

1 Langage à pile : les assembleurs, WASM, WrapScript

1 Langage logique : Prolog

# OUI MAIS LE MARCHÉ

GÉRER MON EMPLOYABILITÉ ? ÊTRE ENGAGÉ ? ÊTRE TRENDY ?

1	Python	🌐	⌚	⚙️	100.0
2	Java	🌐	⌚	⌚	97.5
3	C	⌚	⌚	⚙️	96.0
4	C++	⌚	⌚	⚙️	85.7
5	JavaScript	🌐			83.1
6	C#	🌐	⌚	⌚	77.2
7	HTML,CSS	🌐			75.6
8	Swift	⌚	⌚		69.4
9	Matlab		⌚		69.4
10	SQL		⌚		69.3
11	PHP	🌐			68.8
12	R		⌚		67.7
13	Ruby	🌐	⌚		67.7
14	Shell		⌚		65.9
15	Go	🌐	⌚		62.1
16	Scala	🌐	⌚		59.7
17	Visual Basic		⌚		56.7
18	SAS		⌚		55.4
19	Dart	🌐	⌚		53.4
20	Perl	🌐	⌚		53.2

Job Ranking

1	Python	🌐	⌚	⚙️	100.0
2	Java	🌐	⌚	⌚	93.7
3	C		⌚	⚙️	87.3
4	C++		⌚	⚙️	85.6
5	JavaScript	🌐			85.3
6	C#	🌐	⌚	⌚	76.3
7	R		⌚		74.2
8	HTML,CSS	🌐			73.2
9	Swift		⌚		71.9
10	Dart	🌐	⌚		70.6
11	PHP	🌐			69.7
12	Ruby	🌐	⌚		69.2
13	Go	🌐	⌚		67.2
14	Shell		⌚		65.9
15	Matlab		⌚		62.8
16	Visual Basic		⌚		58.9
17	Julia		⌚		57.6
18	Kotlin	🌐	⌚		56.4
19	Scala	🌐	⌚		55.0
20	Rust	🌐	⌚	⚙️	52.5

Open Source Ranking

1	Python	🌐	⌚	⚙️	100.0
2	Java	🌐	⌚	⌚	94.9
3	C		⌚	⚙️	91.4
4	C++		⌚	⚙️	87.5
5	JavaScript	🌐			74.9
6	C#	🌐	⌚	⚙️	74.1
7	Go	🌐	⌚		69.3
8	R		⌚		68.8
9	Ruby	🌐	⌚		67.1
10	Dart	🌐	⌚		67.1
11	Swift		⌚		65.7
12	Matlab		⌚		65.2
13	HTML,CSS	🌐			65.1
14	Rust	🌐	⌚	⚙️	62.0
15	Shell		⌚		59.9
16	Kotlin	🌐	⌚		58.2
17	Scala	🌐	⌚	⌚	57.0
18	PHP	🌐			55.9
19	SQL		⌚		54.6
20	Julia		⌚		52.7

Trend Ranking

# LES CLASSEMENTS SONT DES PHOTOS

## UN CONSTAT

Python N°1 tient son trust du machine learning ... beaucoup de POC, Google cherche à en sortir

Le trust de Java sur l'info de gestion en France pendant 20 ans fait qu'il y a un besoin de maintenance énorme : principalement du travail de TMA sur des applis Java 8

Les langages proprio comme COBOL ou ABAP n'apparaissent pas dans ce genre de photos

Dart a un unique cas d'usage : le framework mobile Flutter

Beaucoup de build Kotlin, Scala, Typescript dans les grands groupes

Beaucoup de Rust, Go, Elixir, Node.js dans les startup tech

Beaucoup de PHP, Ruby dans les startup non tech

A l'international .net est autant utilisé que JVM

Une tendance générale à plus de typage statique après 10 ans de JS everywhere

# LES LANGAGES MAINSTREAM

COMPOSEZ VOS APPRENTISSAGES

Haskell + Java => Scala

Haskell + Js => Purescript ou Elm

OCaml + Ada + C++ => Rust

OCaml + C# => F#

OCaml + Java => Kotlin

OCaml + ruby => Swift

OCaml + Js => Rescript

C# + Js => Typescript

Erlang + ruby => Elixir

Lua + C + Ada => Go

common Lisp + Java => Clojure

ML et Haskell ont beaucoup influencé les évolutions des langages ces dernières années 

# LES LANGAGES RÉSOLVENT DE PROBLÈMES

## QUELLES SONT VOS PRIORITÉS

Être résilient par compilation : Haskell, OCaml, F#, Scala, Rust ... dans une certaine mesure Kotlin, Swift

Être résilient par parallélisation : Erlang / Elixir

Valider les états (ADT) : Kotlin, Scala, F#, Swift, Rust, OCaml, Haskell

Valider les transitions (GADT) : OCaml, Haskell, Scala

Écrire des DSL génériques (Higher Kinded Types) : Haskell, Scala

Faire de l'embarqué : Go, Rust, OCaml

Faire des jeux vidéos AAA+ : C#, C++ ; AA+ : Lua, Haxe, Javascript, Rust

Faire du calcul parallèle : Rust, C++, Erlang/Elixir

Faire du micro-frontend : F#, C#, Elixir, Typescript

Faire un OS : C, OCaml, Rust

Faire une blockchain : OCaml, Haskell, Rust

Faire de la data science : python, julia, lua, C++, swift

Faire du prototype rapide : javascript, Ruby

« Tournevis multifonction » : F#, Kotlin, OCaml ... Tout langage avec un bon compilateur Javascript et FFI C

# TAKE AWAY

## NOUS AVONS VU

You devriez maîtriser au moins un langage pour chacun des 6 idiomes principaux

Par composition d'idiome, l'apprentissage de nouveaux langages est facilité

Pas beaucoup d'innovation en 40 ans mais ...

... Les langages s'influencent les uns, les autres, la plupart des langages modernes sont multi-paradigmes

Plus que l'idiome c'est le système de types qui prime pour la qualité, et l'écosystème / les ressources pour la diffusion

Vous devez savoir dire pourquoi vous avez choisi un langage pour un projet !



# THE BOSS

SI VOUS VOULEZ ALLER PLUS LOIN

Renforcement :

🎥 Object Oriented Programming is not what I thought

Diversification :

🎥 Four langages from forty years ago

🎥 One kata, three langage



# THE QUEST

## Property based Testing

**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**

**Les ADT sont une très bonne documentation de domaines métiers... mais insuffisants pour documenter certaines catégories de règles de gestion**



# PROPRIÉTÉ ?

Remember last Christmas !

Quand a été fêté noël en 2021 ?

Quand a été fêté noël en 2020 ?

Quand a été fêté noël en 1990 ?



Le 25 décembre est une PROPRIÉTÉ qui définit « Noël »

# LES TYPES DE TESTS

DRIVEN BY WHAT ? TROP D'ACRONYME !

TDD : Test Driven Development (red-green-refactor cycle)

PBT : Property Based Testing (on en parle dans 5 min), teste sur base de valeurs aléatoire déduites de propriétés

Mutation testing : tester les tests par mutation automatique du code

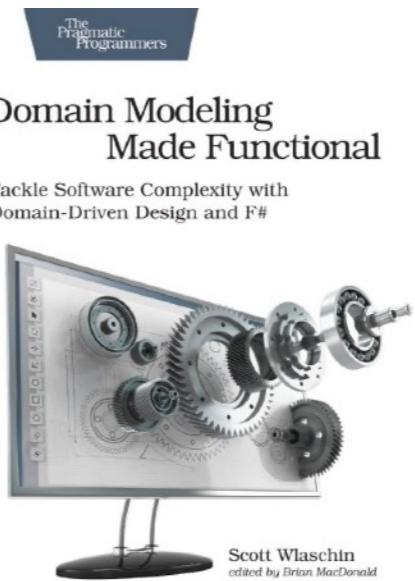
Fuzz testing : algorithmes génétiques qui génèrent des test PBT ou mutation

... D'autres types spécifiques (load, penetration, ...)

# NOT TESTS

DRIVEN BY WHAT ? TROP D'ACRONYME !

TDD : Type Driven Development (pensez au Tennis Kata)



Types as specification : thèse de ce livre (lisez le)

DDD : Domain Driven Design, un concept qui vise à faire se rapprocher l'architecture du métier, plutôt que des contraintes techniques

# LES TESTS

QUELLES SONT VOS PRIORITÉS

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

Écrits (SFD, Wiki, xDoc) < TU < PBT < ADT < GADT

Choisir le meilleur niveau possible et maîtrisé selon le point à documenter



# **TAKE TAKE AWAY**

**POUR DES LOGICIELS DE QUALITÉ**

Faites du

Type Driven Development

then

Test Driven Refactoring



# TROLL OF FAME KATA

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



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

MALHEUREUSEMENT, IL A ÉTÉ ABATTU PAR UN HORRIBLE ELFÉ.

VIVE LE NOUVEAU ROI, VIVE LE TROLL AKLASS!

CETTE FOIS C'EST DÉCIDÉ LE TOURNOI DE CHASSE À L'ELFE 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.

# THE BOSS

SI VOUS VOULEZ ALLER PLUS LOIN

Renforcement :

🎥 [Types VS Tests](#)

🎥 [Much Ado About Testing](#)

Diversification :

💻 [Le kata dans d'autres langages \(Java, Haskell, Rust, JS\)](#)

🔬 [Energy Efficiency across Programming Languages](#)



# **QUALITÉ DU SI**

---

**COURS 3 - GESTION D'ERREURS & QUALITÉ DU CODE**

# THE QUEST

## Gestion d'erreurs

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



# EXCEPTIONS

RAISE / THROW ARE UGLY GOTOS

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

```
type weapon  
type target  
type impacted = | Impacted  
  
let arm_your_bow : weapon = raise (Failure "NoMoreArrow")  
let targeted_monster : target = raise (Failure "TooMuchFog")  
let hit_monster : weapon -> target -> impacted =  
    fun w t -> Impacted
```

On remarquera le manque d'expressivité de Java

# EXCEPTIONS

RAISE / THROW ARE UGLY GOTOS LOST IN TIME OF ASYNC / FUTURE

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

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

# EXCEPTIONS

TRY / CATCH SONT LES RACINES DU MAL

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

```
let arm_your_bow : weapon = try raise (Failure "NoMoreArrow") with  
Failure s -> let _ = print_endline s in raise (Failure « unarmed»)
```

Dans une instruction on peut s'arrêter là

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

Dans une expression on peut passer la main ou traiter définitivement

# 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 ?

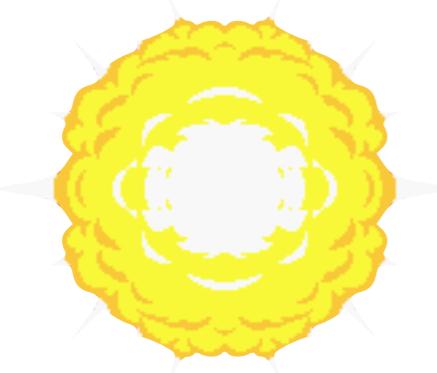
Quand vous voulez faire « crasher » le programme, par exemple :

- 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

# MODÉLISER UNE ABSENCE POTENTIELLE DE VALEUR

## OPTION

```
class Weapon {}

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();
}
----- ----- ,,,
```

```
type weapon
type target
type weapon
type target
type impacted = | Impacted

let arm_your_bow : weapon option = None
let targeted_monster : target option = None
let hit_monster : weapon -> target -> impacted option =
  fun w t -> None
let hit_monster : weapon -> target -> impacted option =
  fun w t -> None
```

# MODÉLISER UNE ABSENCE POTENTIELLE DE VALEUR

## TRAITER LES VALEURS OPTIONNELLES

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

```
let (let*) = Option.bind  
let (>>=) = Option.bind (* Infix operator for bind *)  
let (>>|) x f = Option.map f x (* Infix operator for map with reverse  
parameters *)  
  
let hit_monster_point_free w t =  
    w >>= fun _ -> t >>| (fun _ -> Impacted)
```

Optional n'a pas évolué en sealed interface/record en Java 17  
La manipulation de Optional est fastidieuse et demande de la vigilance

Les patterns infatigable sont facilités à plus aisée  
... mais peut être fastidieux en temps d'apprendre ces nouveaux opérateurs

**flatmap (aka bind) et map facilitent la manipulation des valeurs optionnelles et Les opérateur let\*/let+ rendent plus visible l'extraction de valeur de l'Option garantissent un bon traitement des cas**

# TAKE AWAY

## LES OPTIONS

A utiliser pour modéliser l'absence de valeur

Sécurisant, surtout quand on dispose d'ADT

Facile à manipuler avec syntaxe spécifique

(*let\* -> OCaml, let! -> F#, do notation ->*

*Haskell, for comprehension -> Scala*)

... ou à défaut flatMap / bind

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



# MODÉLISER UNE ERREUR POTENTIELLE

## RESULT

```
type weapon = string
type target = string
type impacted = | Impacted of target

let must_be_carried w = if w = "bow" then Ok w else Error (Failure "not carried")
let (let*) = Result.bind

let hit_monster_let_star w t =
  let* used = w in
  let* targeted = t in
  Ok (Impacted targeted)

let foo = hit_monster_let_star (must_be_carried "bow") (Ok "moblin");;
```

Depuis OCaml 4.08, la lib standard dispose de modules Option et Result satisfaisant

S'ils ne sont pas suffisant, regardez les lib Preface, Base ou Bastet

# 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 17

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 » en 2022 utilisez au moins VAVR

[https://www.vavr.io/vavr-docs/#\\_either](https://www.vavr.io/vavr-docs/#_either) ...

Ou attendez Java18 pour faire une lib propre sans hacks (il y a du boulot!)

Ou passez à Scala ou Kotlin + Arrow-Kt

Ou abandonnez la JVM 😈

# TAKE AWAY

## LES RESULTS

A utiliser pour modéliser les cas d'erreur

Sécurisant, surtout quand on dispose d'ADT

Facile à manipuler avec une syntaxe spécifique

Parfois encodé avec un seul paramètre

```
type 'a result =  
| Ok of 'a  
| Error of exn
```

Dans certains langages Result est remplacé par Either (Scala, Haskell), c'est un type plus générique qui par « convention » représente les erreurs dans la valeur Left

```
type ('a, 'b) either =  
| Left of 'a  
| Right of 'b
```



# 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** `Result.t` va contenir la première erreur qui sera propagée par `bind` ou `map`

On aimeraient quelque chose qui ressemble à

```
type ('ok, 'error) result =
| Ok of 'ok
| Error of 'error list
```

(ce type n'existe pas dans `Stdlib` c'est une illustration)

Avec **OCaml Base (Stdlib alternative)**, il existe une fonction `combine_errors`

```
val combine_errors : ('ok, 'err) t list -> ('ok list, 'err list) t
```

... mais cela devient vite fastidieux à manipuler

# VALIDATION

ACCUMULATEUR D'ERREUR POUR VALIDER LES DATA STRUCTURES

- En Haskell, il existe le module Control.Monad.Validate

Dans les autres langages, il faut existe souvent des lib complémentaires :

- En OCaml, la lib Base fournit un module validate
- En Scala, la lib Cats fournit un type de donnée Validated
- En Kotlin, la lib Arrow-kt fournit un type de donnée Validated

# TAKE AWAY

## DU TAKE AWAY

**Utilisez Option.t pour représenter la possibilité d'une absence de valeur**

**Utilisez Result.t pour représenter la possibilité d'une erreur**

**Utilisez Base.Validate si vous voulez un accumulateur d'erreur pour valider des données**

**Dans la majorité des cas ce que vous voulez c'est Result.t !!!**



# SIDE QUEST

Qualité du code

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

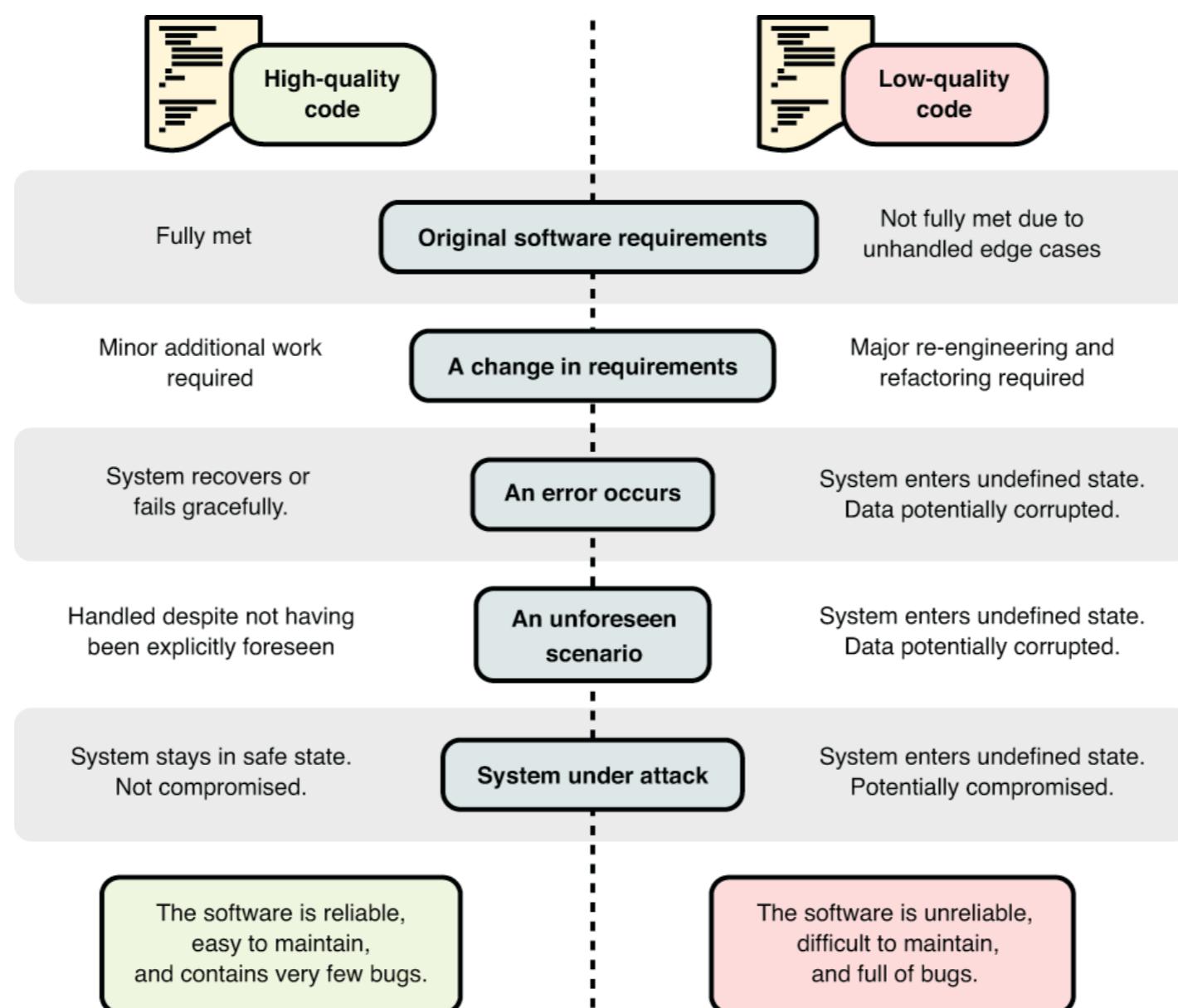
Jusqu'à présent nous nous sommes concentré sur des tactiques (ADT, Option/Result, PBT) pour résoudre des problèmes récurrents (machine états implicites, null pointer errors, exemples mal choisis, ...) ... mais spécifiques !

Quelles sont les stratégies d'architecture généralisables ?



# QUALITÉ

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

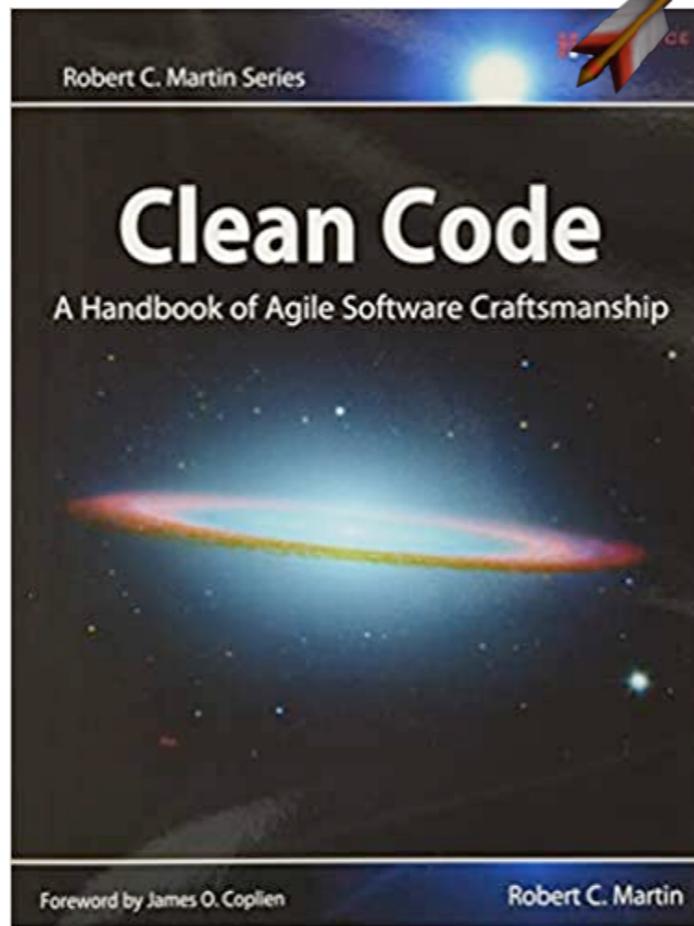


Extrait de *Good Code, Bad Code* par Tom Long

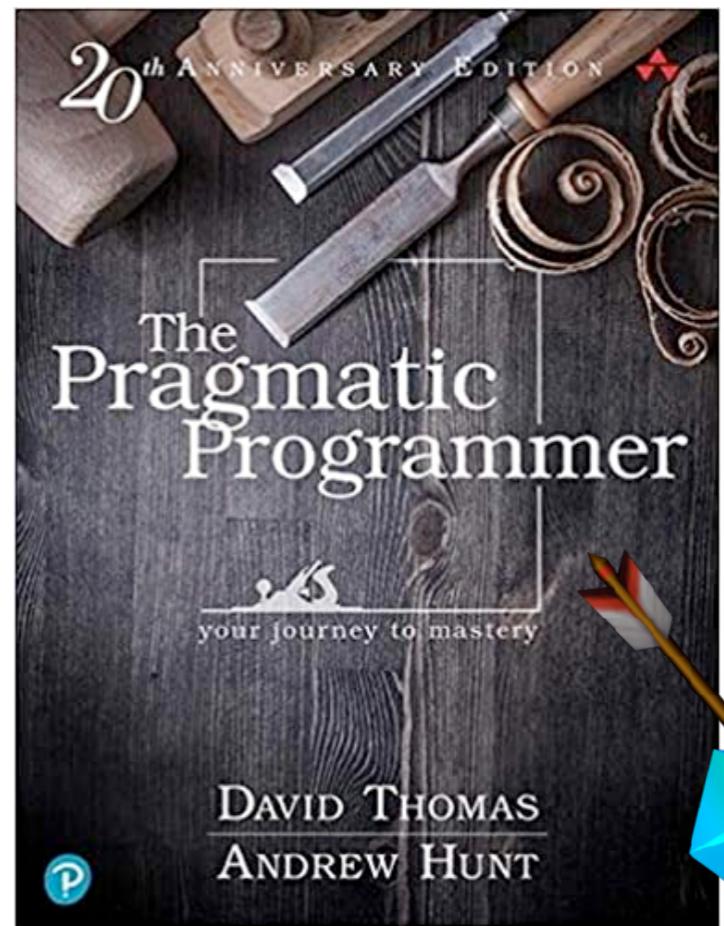
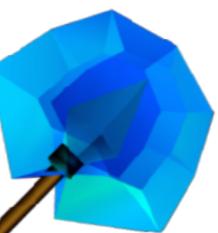
# GÉNIE LOGICIEL

QUELS SONT LES OUVRAGES DE RÉFÉRENCE

SOLID



CLEAN ARCHITECTURE

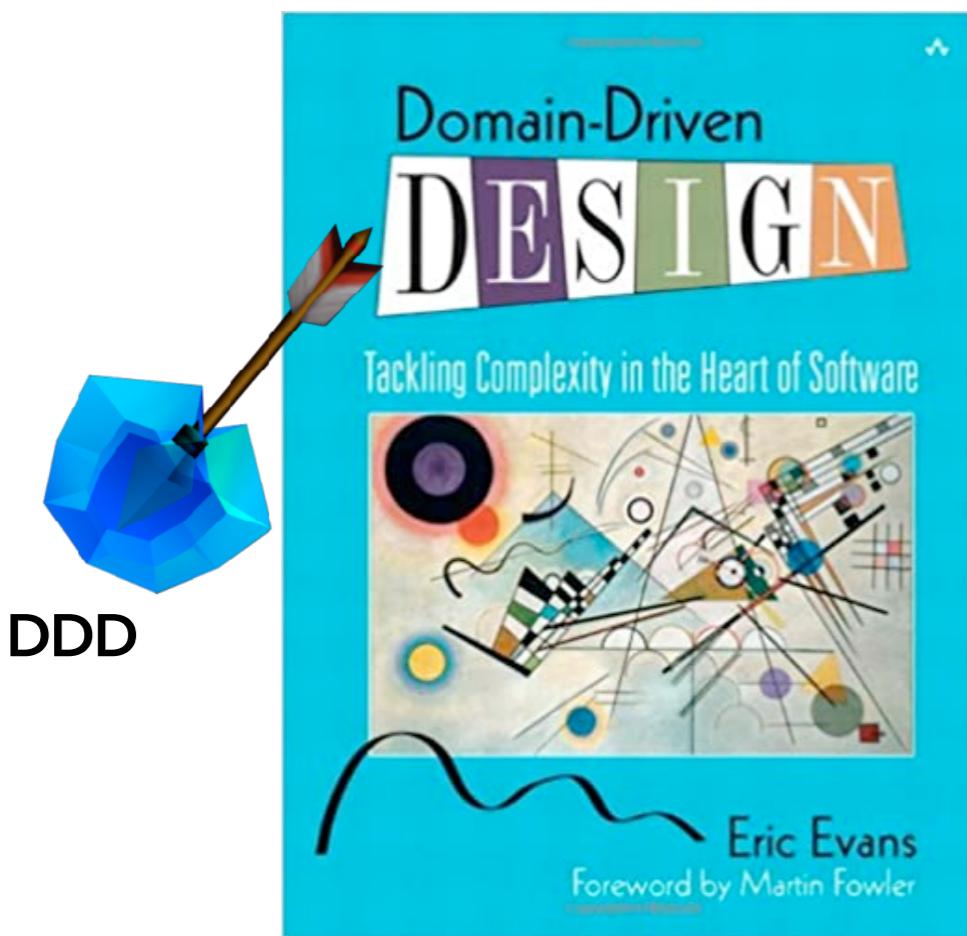


DRY

SONT CENTRÉS SUR DES EXEMPLES OOP

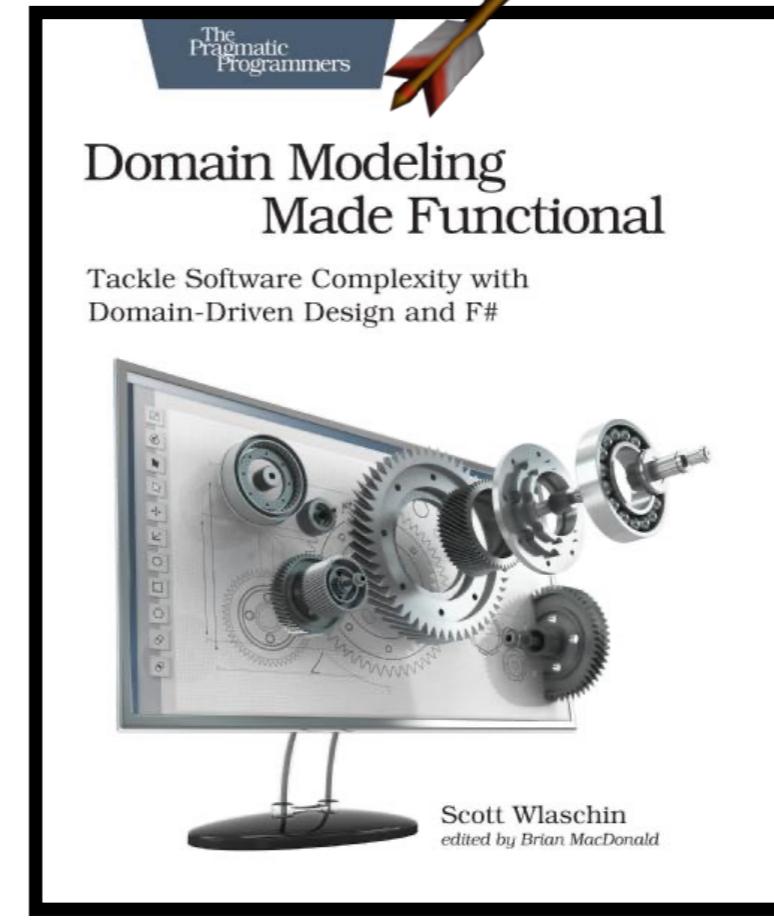
# ARCHITECTURE

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

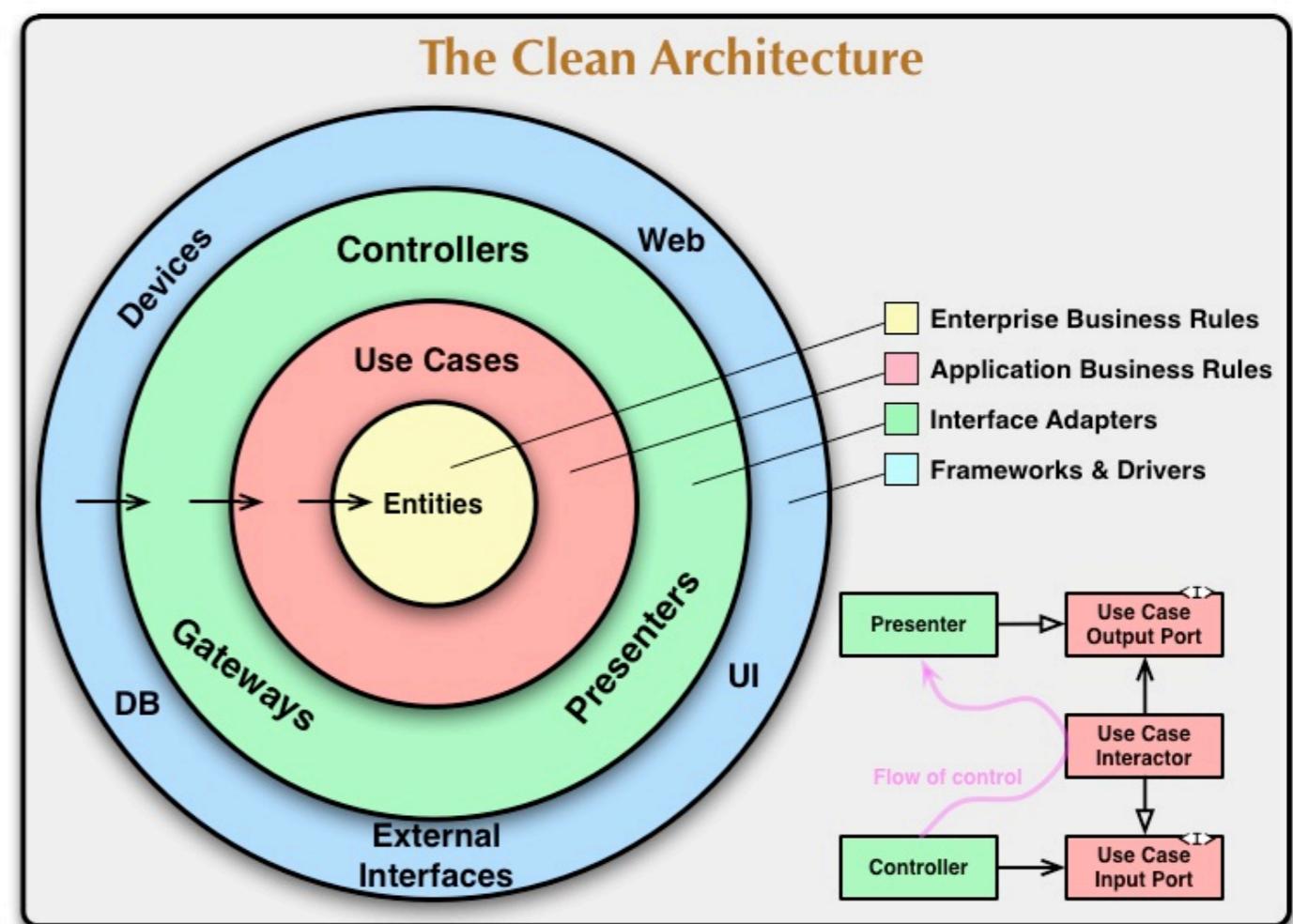
## INDÉPENDANT DES IDIOMES

# 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



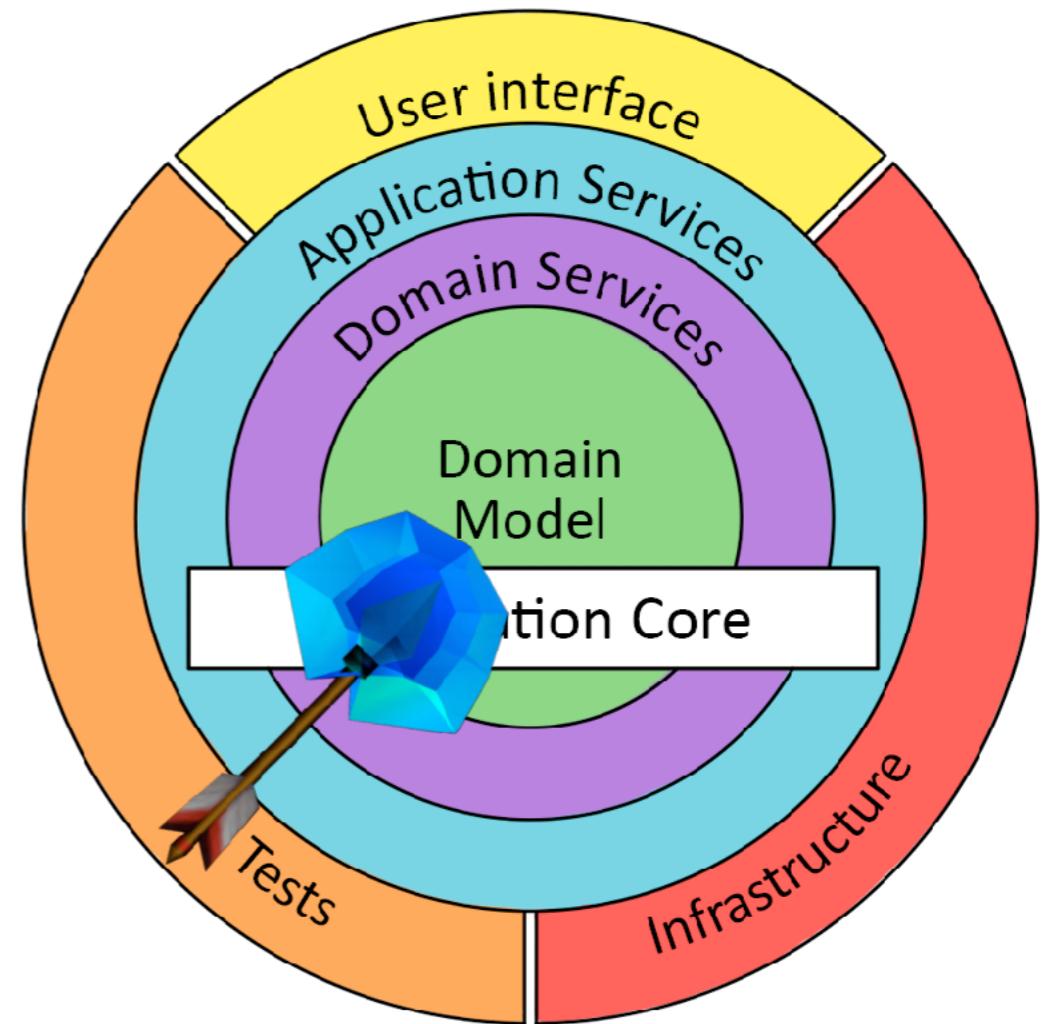
BREF : DRY + SOLID + EFFETS AUX FRONTIÈRES DU PROGRAMME

# 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

# D&D KATA



# THE BOSS

SI VOUS VOULEZ ALLER PLUS LOIN

Renforcement :

🎥 [Solid : au-delà de la POO](#) par Caroline Gaudreau & Gaël Deest

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

🎥 [Le design de l'erreur](#) par François Teychene

Diversification :

 [Error handling in Java](#)

 [Error handling in Scala](#)

 [Error handling in Kotlin](#)

 [Error handling in TypeScript](#)



# **QUALITÉ DU SI**

---

COURS 4 - FRONTENDS DATAFLOW

# HISTORIQUE

30 ANS DE WEB



AoL ALttP LA

OoT OoA OoS

TP SS

LoZ OoT MM

FS WW FSA

MC PH ST

# WEB DEVELOPMENT

90'S : L'ÈRE DU HTML

Pages statiques HTML

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

# Apple



May 8, 1998

Hot News Headlines

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



Pro.

Creative professionals, meet your match.



Go.

We rewrote the book on mobile computing.



Whoa.

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

The  
Apple Store

Hot News  
About Apple

Products  
Support

Design & Publishing  
Education

Developer  
Where to Buy



Find:

Shortcut Search

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

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

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

# WEB DEVELOPMENT

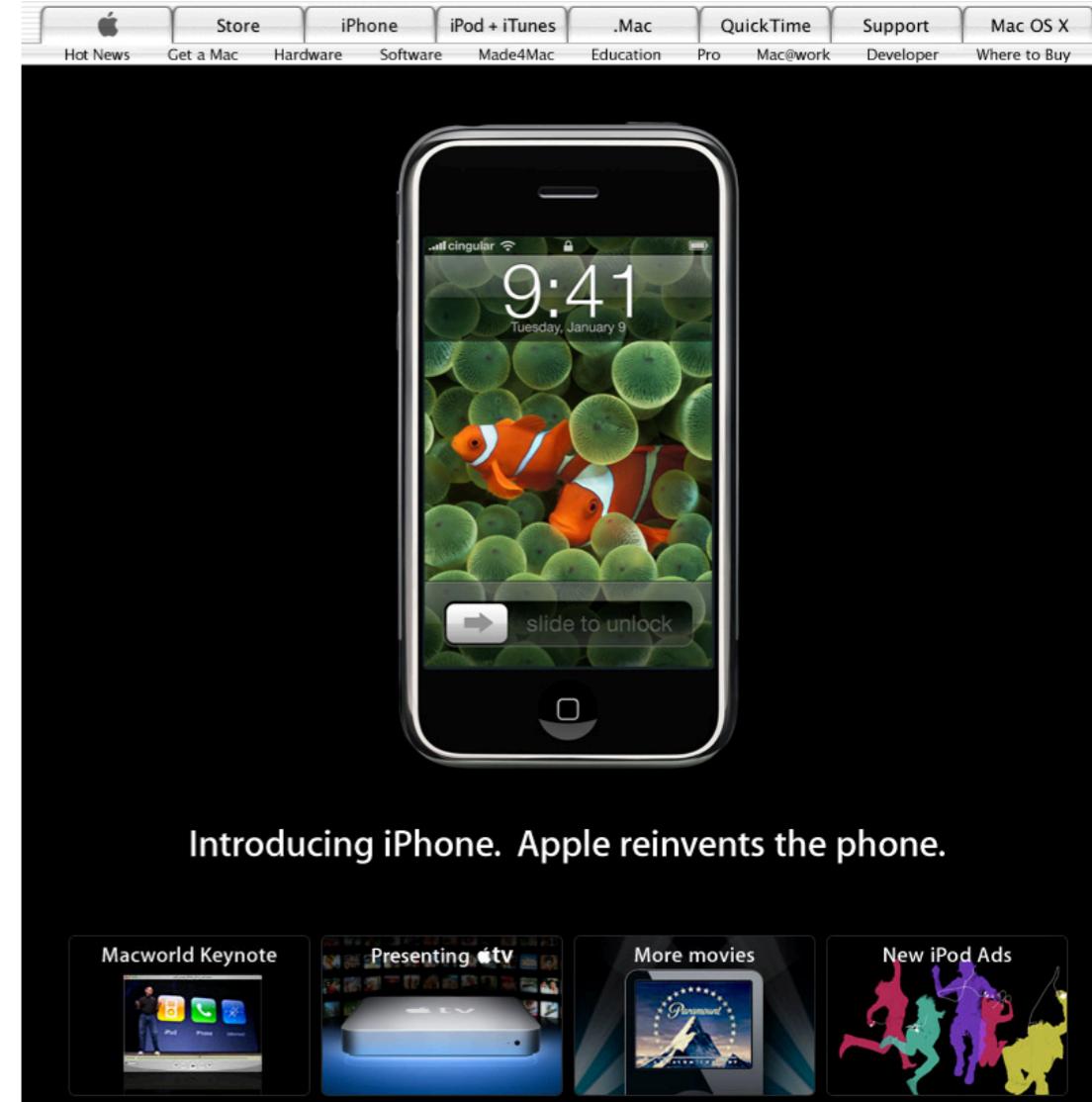
00'S : L'ÈRE DE LAMP

Server Side Rendering

Stack star LAMP : Linux Apache PHP MySql

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

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



# WEB DEVELOPMENT

10'S : L'ÈRE DE JAVASCRIPT

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

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

La finalisation des Web Component (lit-element)

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



# THE QUEST

Gestion des états en UI

Les architectures Front Back sont actuellement le standard

Les backends sont souvent des applications CRUD

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

Ce qui amène de nouvelles problématiques

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



# FRONT END DEV

JS EVERYWHERE

Quelle approche ?

- Frameworks : Angular / Vue / React / Svelte
- Standard : Web Components

Javascript est un langage complexe  
qui demande l'utilisation de beaucoup lib tierces combler ses manques (Ramda,  
Immutable.js, Flow, eslint, ...)  
qui évolue 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, Typescript en première ligne !  
mais aussi Elm, Rescript, ClojureScript, Purescript, KotlinJs, Ocaml ...

# **GESTION D'ÉTATS FRONT**

**ÉTAT LOCAL OU ÉTAT GLOBAL ?**

**Un état global unique facilite la gestion de la logique applicative**

**Il est parfois utile d'avoir un état local pour une logique de composant réutilisable  
(datepicker, ...)**

# GESTION D'ÉTATS

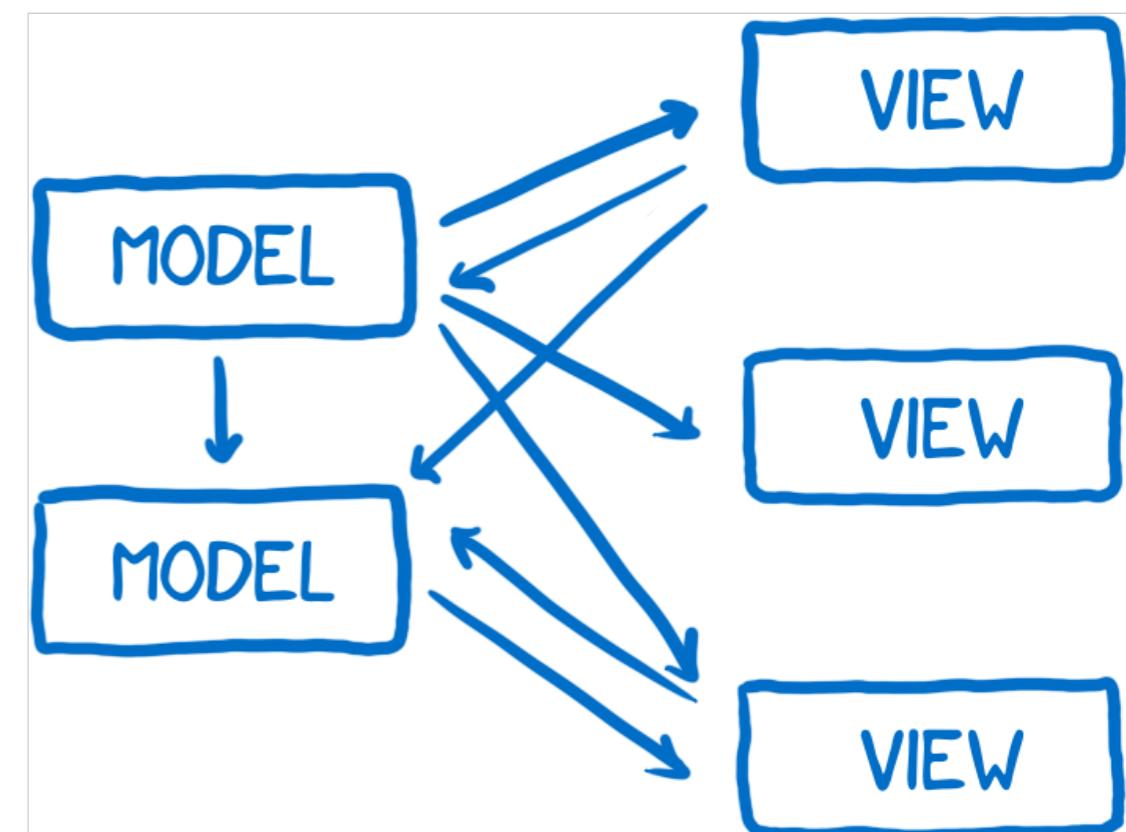
## MVC / MVVM CÔTÉ CLIENT

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 🎲

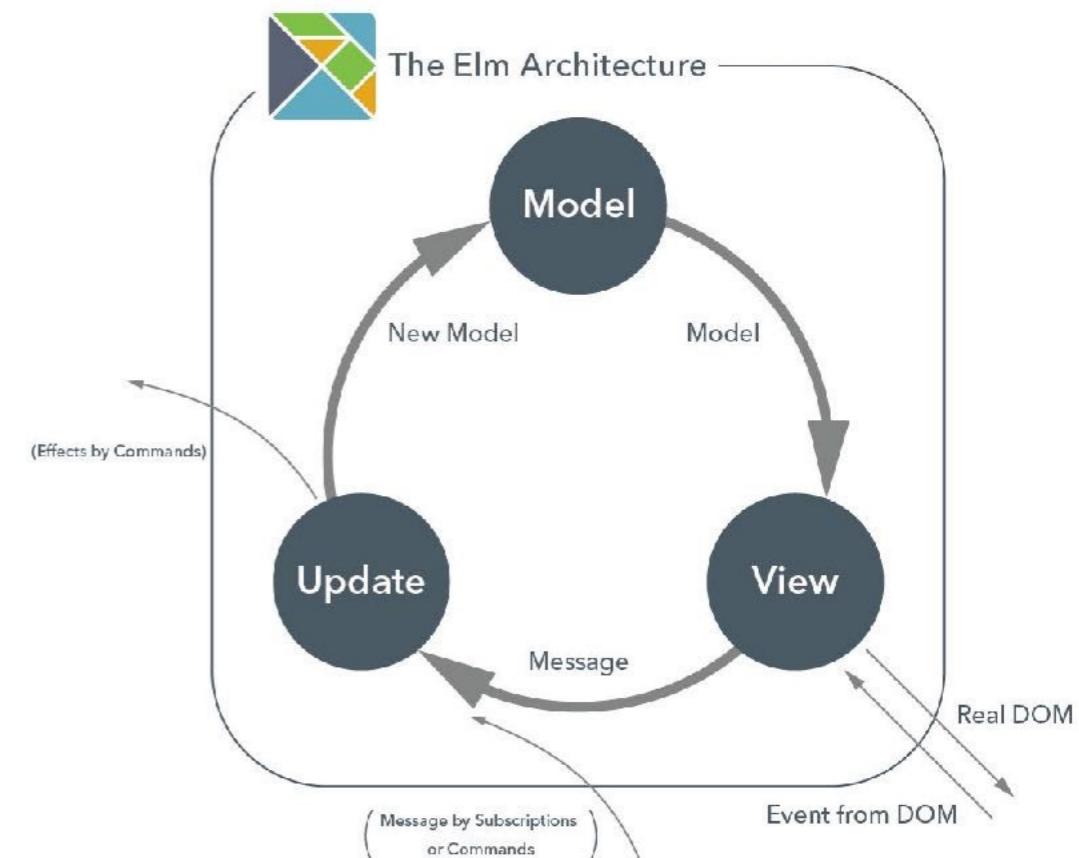


# GESTION D'ÉTATS

ONE WAY DATA FLOW

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

Un flux de circulation des données unidirectionnel, popularisé par Elm, démocratisé par Flux puis Redux.

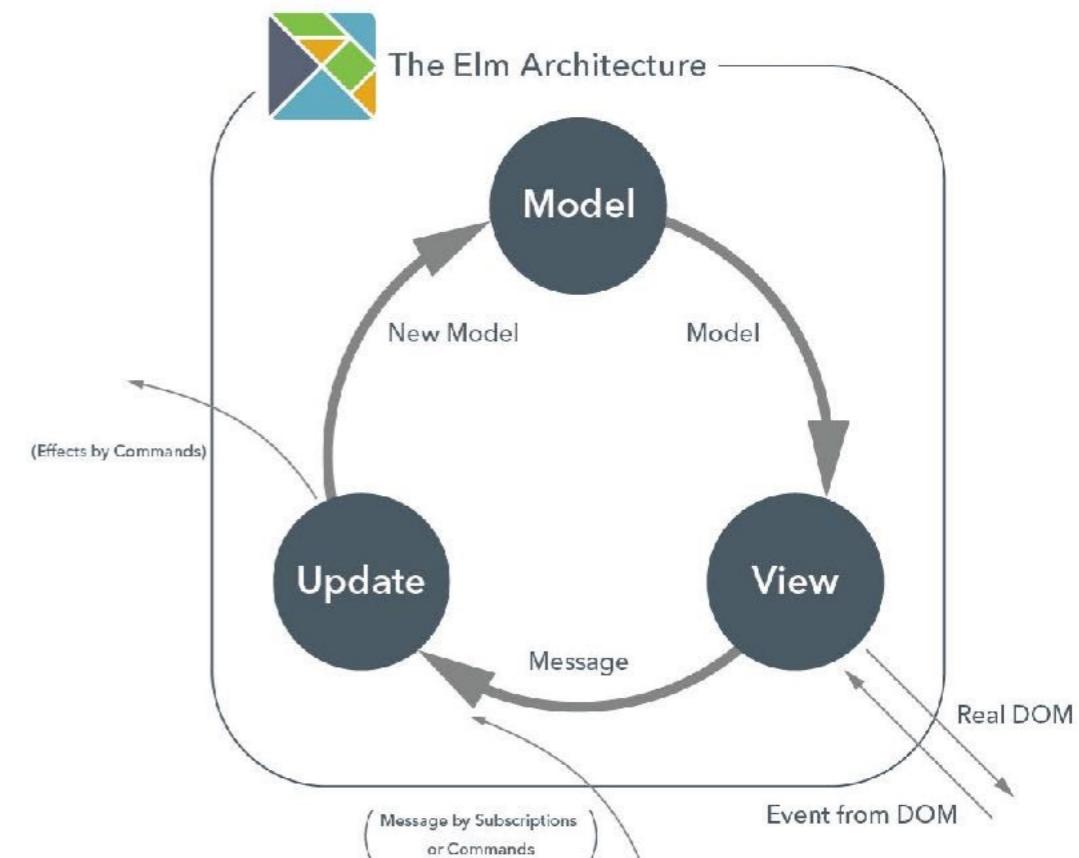


# GESTION D'ÉTATS

## ELM ARCHITECTURE

Implémentations :

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

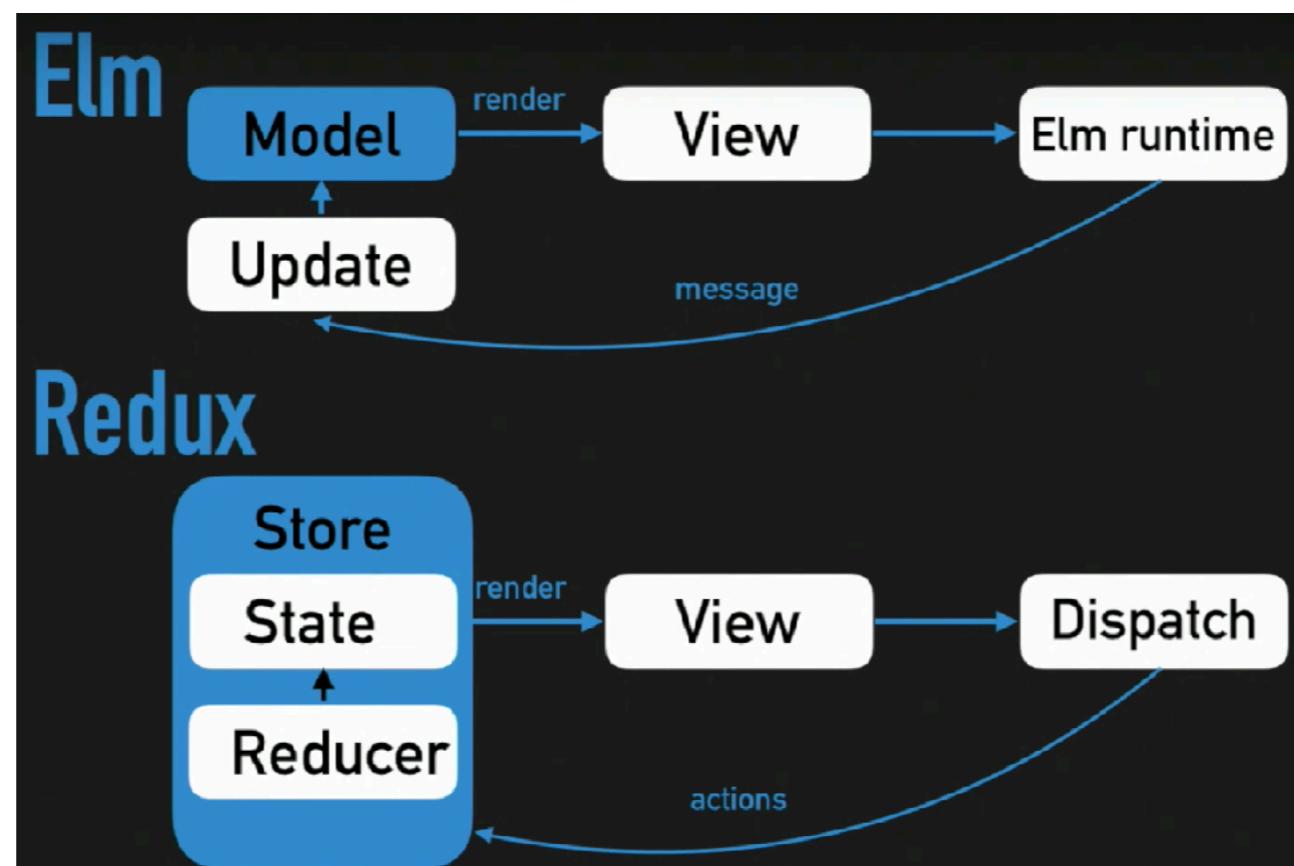


A inspiré :

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

# GESTION D'ÉTATS

ELM / REDUX



# FRONT END DEV

## CHOISIR UNE OPTION

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

- Langage: Typescript (verbeux et mauvaise inférence, mais nécessaire à maîtriser en 2021)
- Frameworks : 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 2021 ... SURTOUT QUE [REACTJS.ORG](#) EST DATÉ

2014

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

2016

```
const StatelessComponent2016 = (props) => <div> ... </div>;  
class StatelessComponent 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) {  
    ...  
  }  
}
```

2019

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

+ Mixins  
+ Flux

+ HOC  
+ REDUX

+ Render props  
+ Context

+ Hook  
+ Context

+ API changes

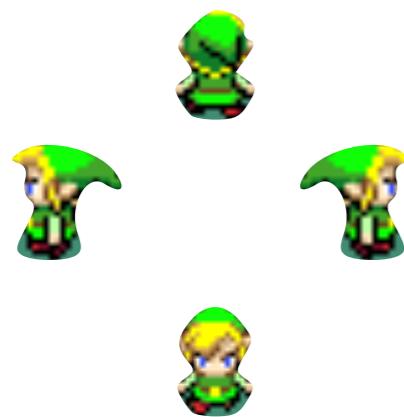
+ API changes

DANS PÉRIODE D'Urgence, il IMPOSE UNE « VISION » à respecter

Au contraire des « visionaries », dans d'autres projets autrefois normal react est une lib qui laisse beaucoup de liberté dans l'architecture du projet

# STATE MACHINE STRIKE BACK

SI ON ADAPTAIT NOTRE MACHINE EN TYPESCRIPT



```
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"
    }
}
```

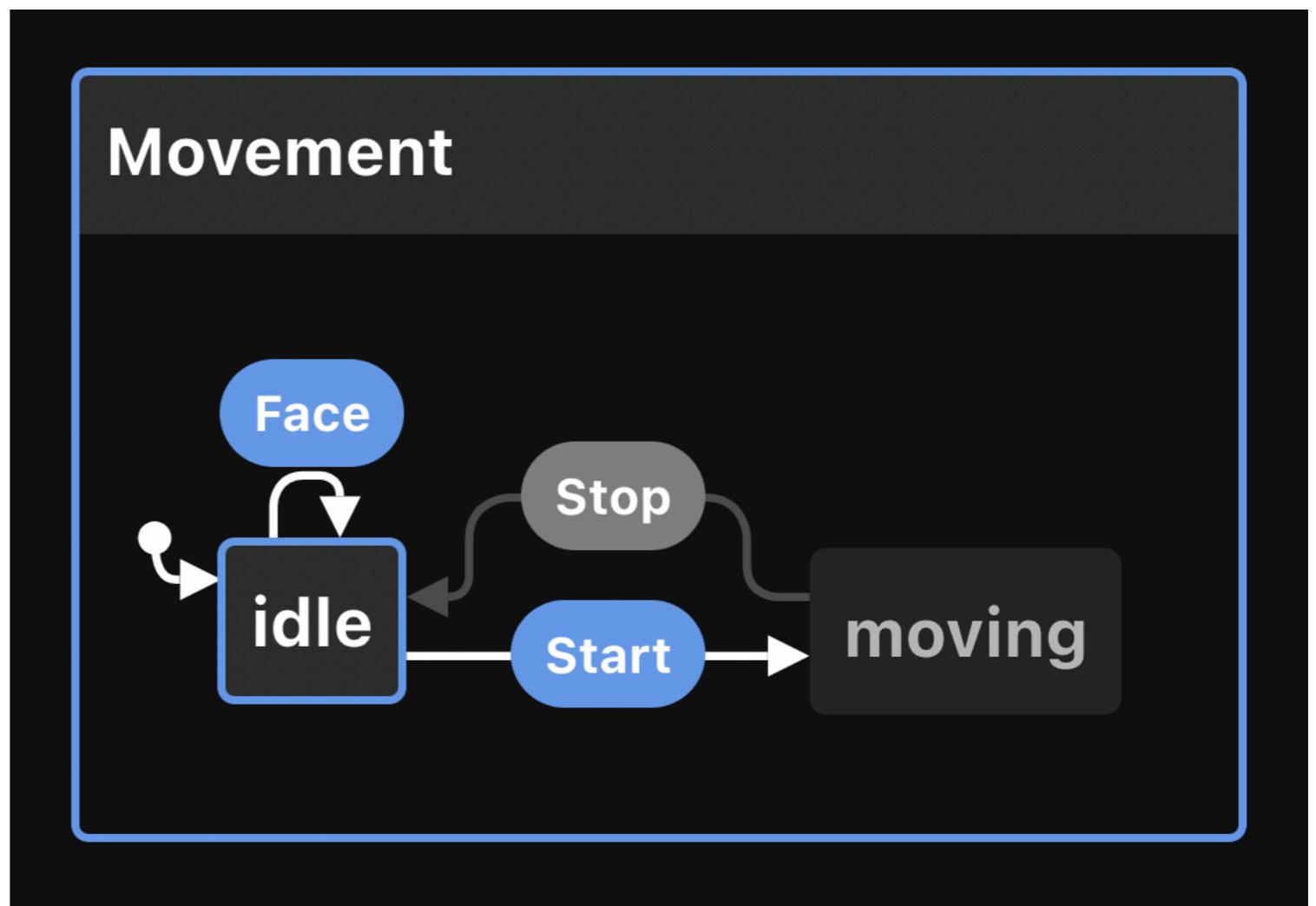


a des « tagged union » qui sont des types OU avec un pattern matching exhaustif

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

# STATE MACHINE STRIKE BACK

RAPPEL DE NOTRE MACHINE A ÉTATS



# STATE MACHINE STRIKE BACK

SI ON ADAPTAIT LES COMMANDES... EN ACTIONS



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

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

Rien de surprenant, on met simplement les paramètres (direction) dans l'interface

Face, Start et Stop sont des types (vs des constructeurs en OCaml), on peut créer nos constructeurs de valeurs

# 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

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

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



# ERREURS

FP-TS

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);

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

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

# ÉCOSSYSTÈME REACT

## NOTRE ENVIRONNEMENT DE TP

Une application plus complexe nécessite de pouvoir chainer 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 !

# ÉCOSSYSTÈME REACT

## 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 !



# CATSTAGRAM KATA



# 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](#)

 [ocaml-vdom](#)



# **QUALITÉ DU SI**

---

**COURS 5 - CLOUD & MICROSERVICES**

# 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...

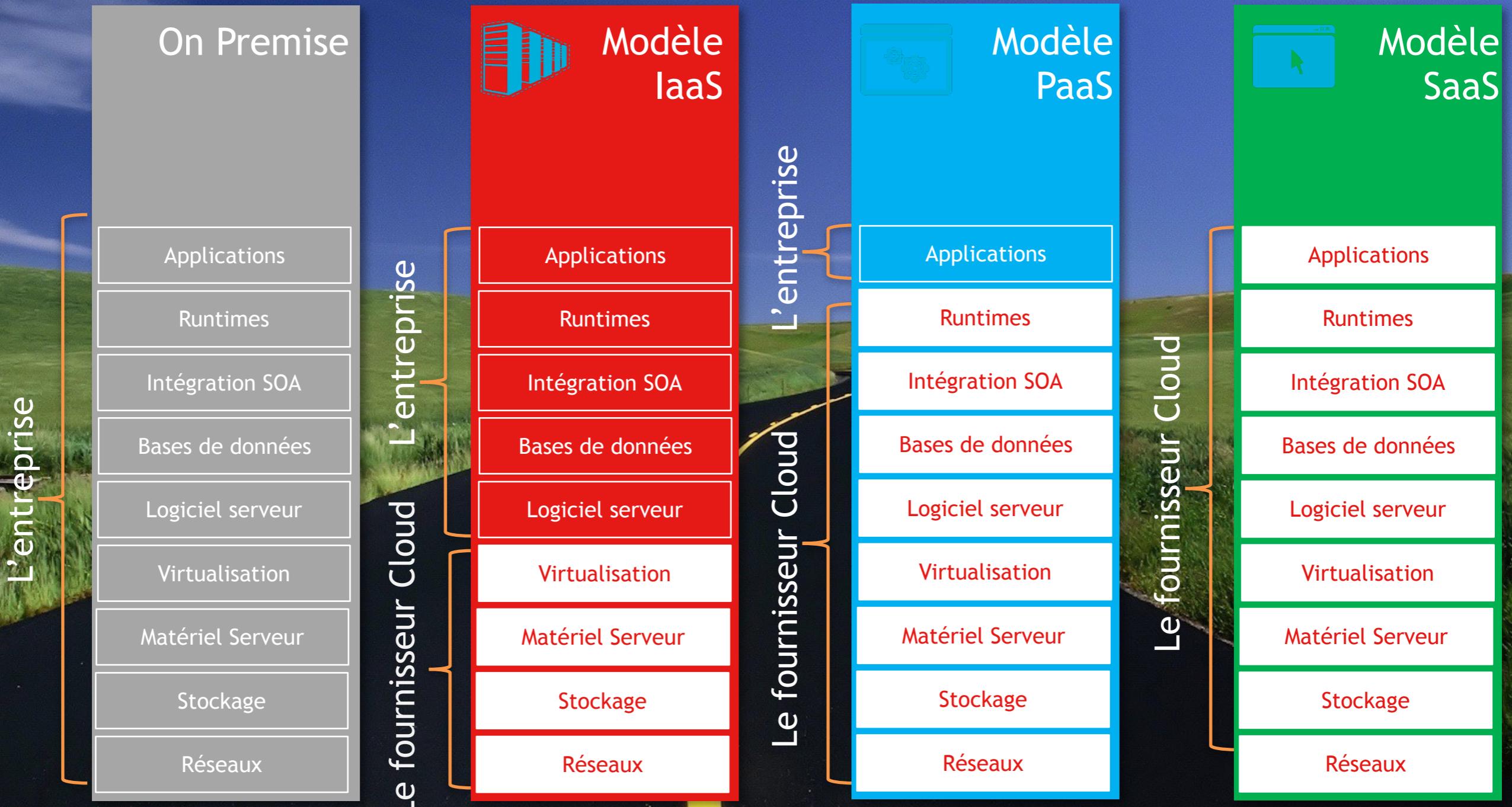
... Y compris pour le hardware (modèle Tesla)

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

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



# LES POSITIONNEMENTS



# 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



Microsoft Azure



Infrastructure As a  
Service  
IaaS

Container  
Based



Microsoft Azure



VM  
Based



Microsoft Azure



# TENDANCES

## CONSTAT

Gérer du baremetal n'a 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

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)

Parts de marché en 2020, ~32% AWS ; ~20% Azure ; ~6% 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)  
et diversification des stratégies Google (Anthos + partenariat OVH)

# 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 le CAPEX et diminue l'OPEX

Il y a des effets bénéfiques induit 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 RH : noOPS, OPS-dev, DEV & OPS ou DEVops ? SRE ?



# DUNGEON BOSS

SI VOUS VOULEZ ALLER PLUS LOIN

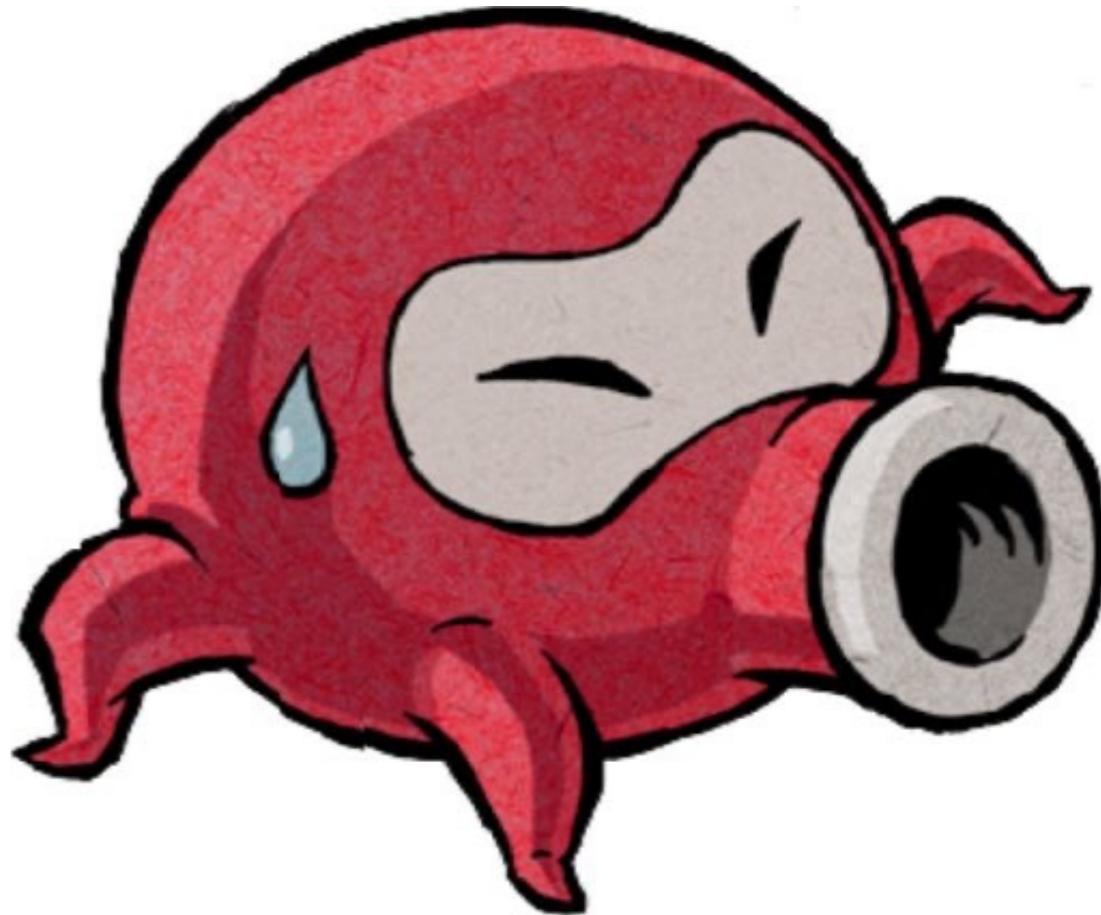
Renforcement :



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*

# THE QUEST

Du monolith au micro services

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

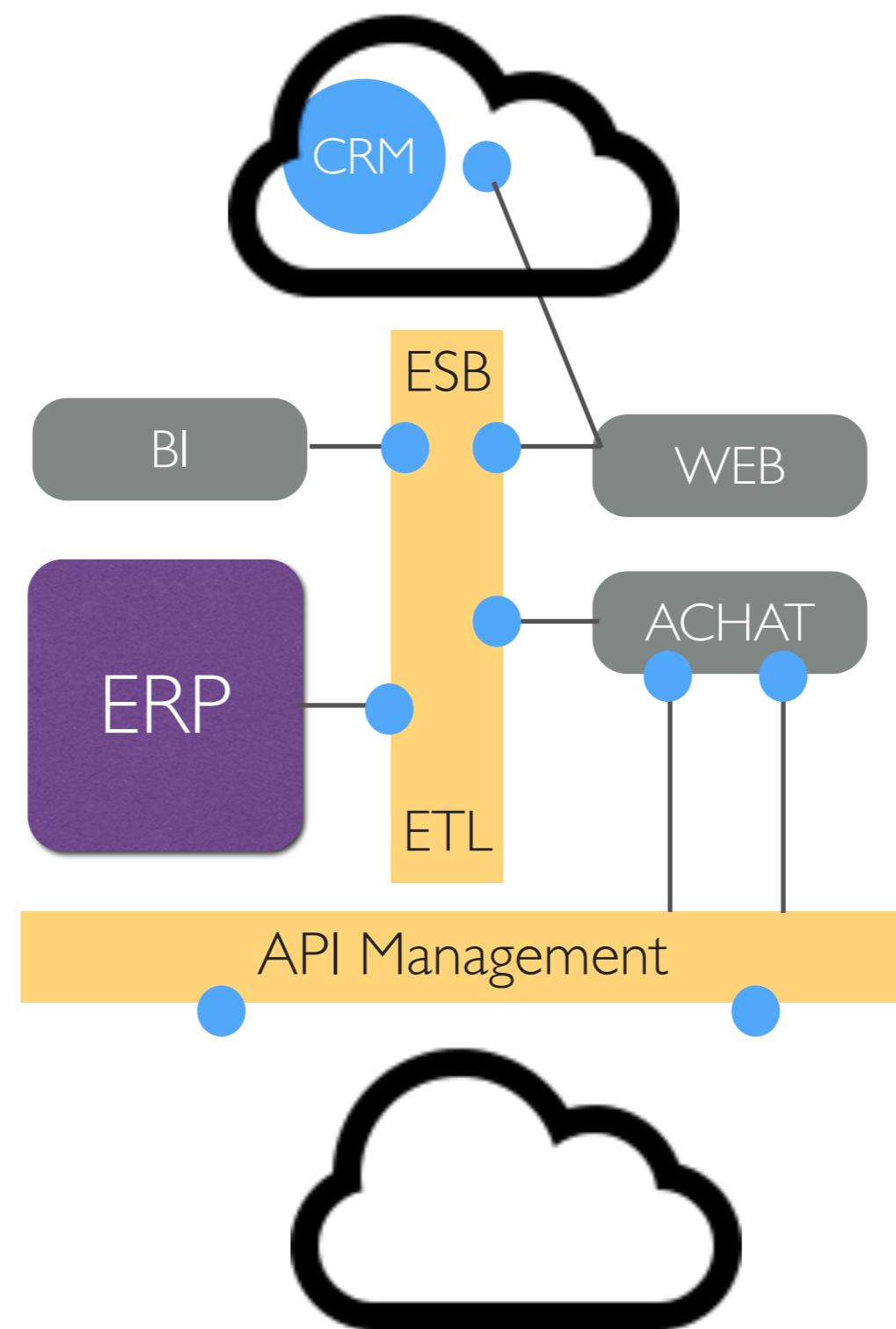
**Comprendre les architectures actuelles des web applications, leurs évolutions, leurs limites**

**Eviter le hype driven développement**

**Adopter une posture d'architecte**



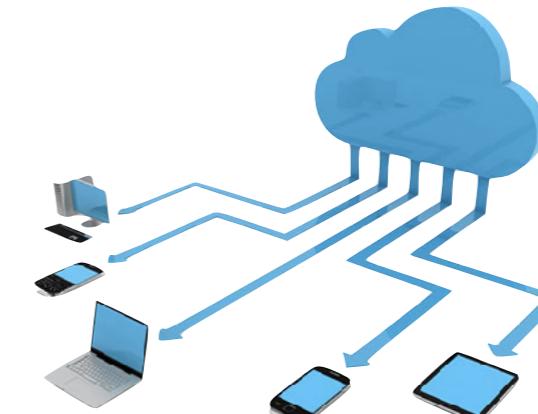
# LES SI ACTUELS



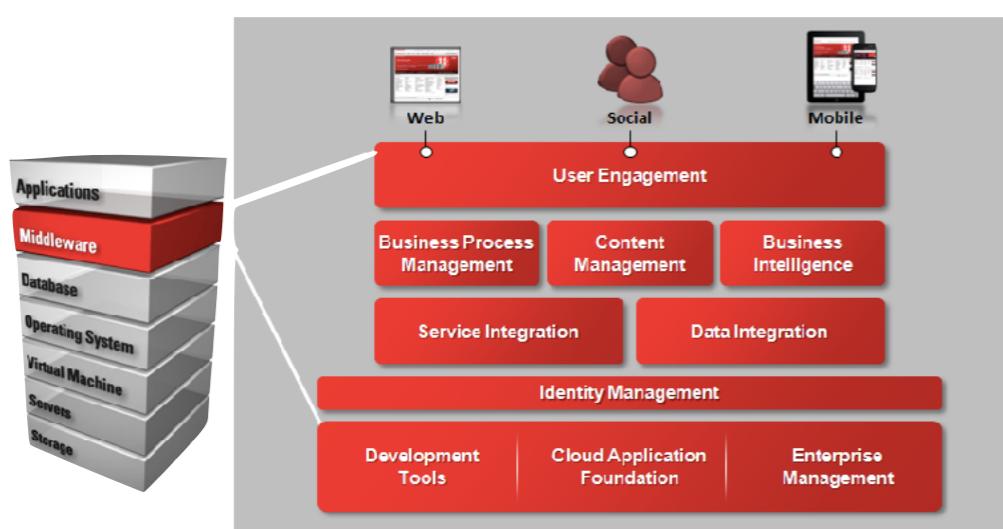
# LA COMPOSITION DU SI



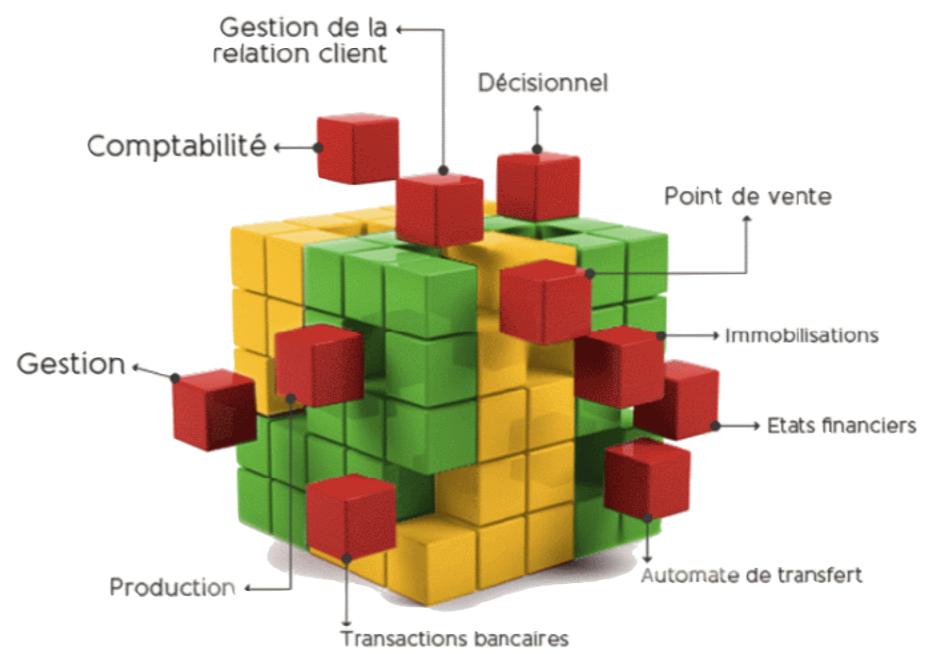
Logiciels spécifiques (App)



Services tiers (SaaS)



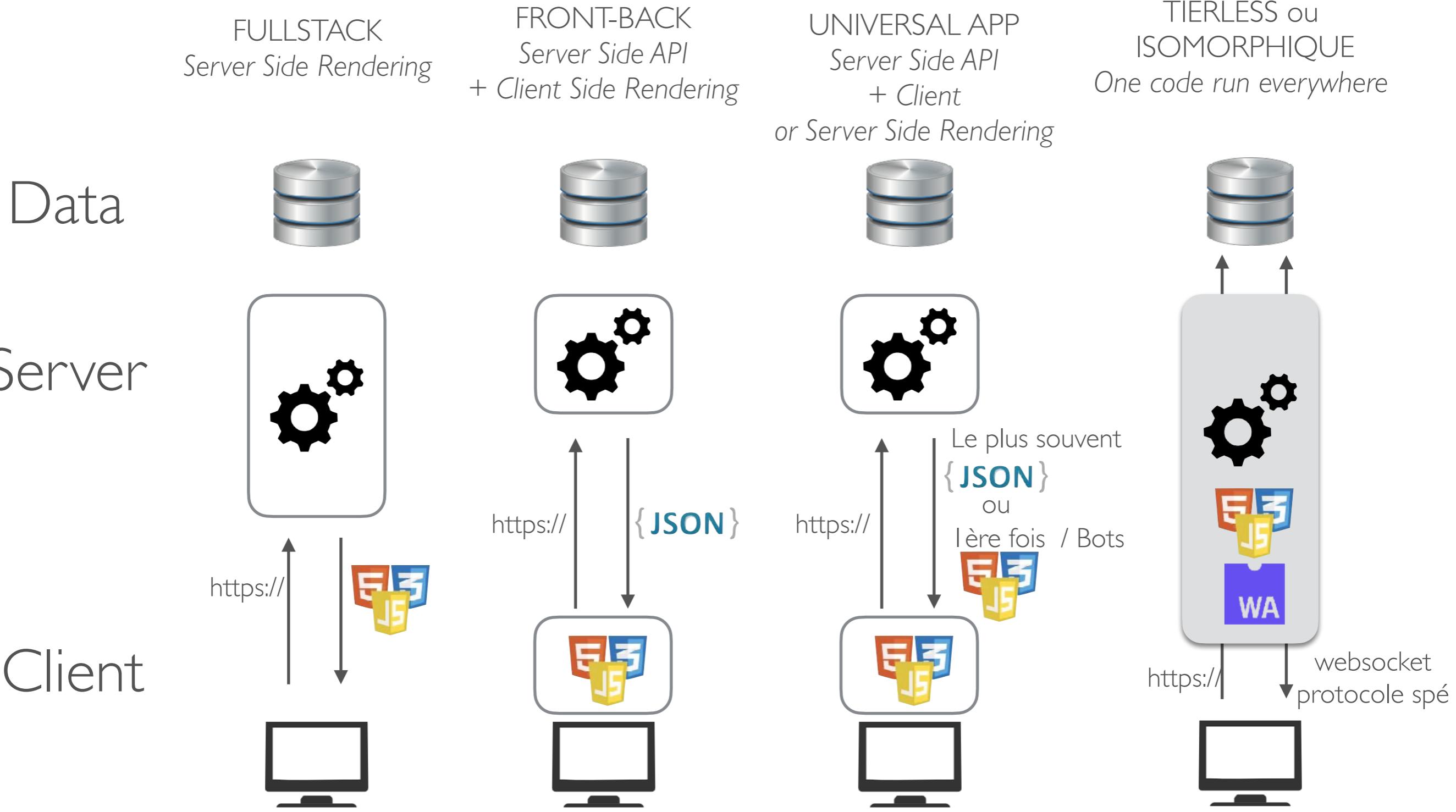
Progiciels



ERP

# QU'EST CE QU'UNE APP ?

## EVOLUTION DES ARCHITECTURES DES WEB APPS



# QU'EST CE QU'UNE APP ?

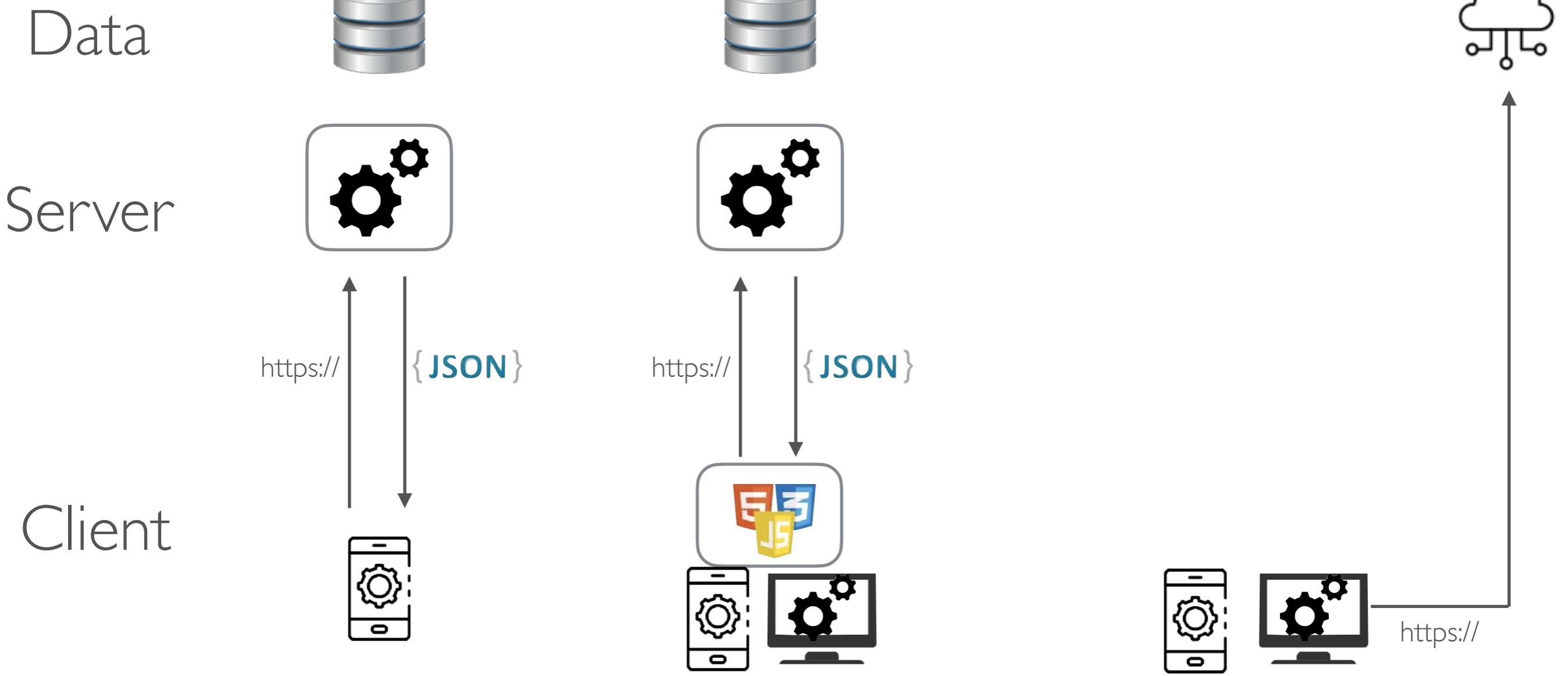
## LES APP CLIENTS LOURDS

CLIENT LOURD  
Server Side API  
+ Client lourd natif

HYBRIDE  
Server Side API  
Client lourd en JS

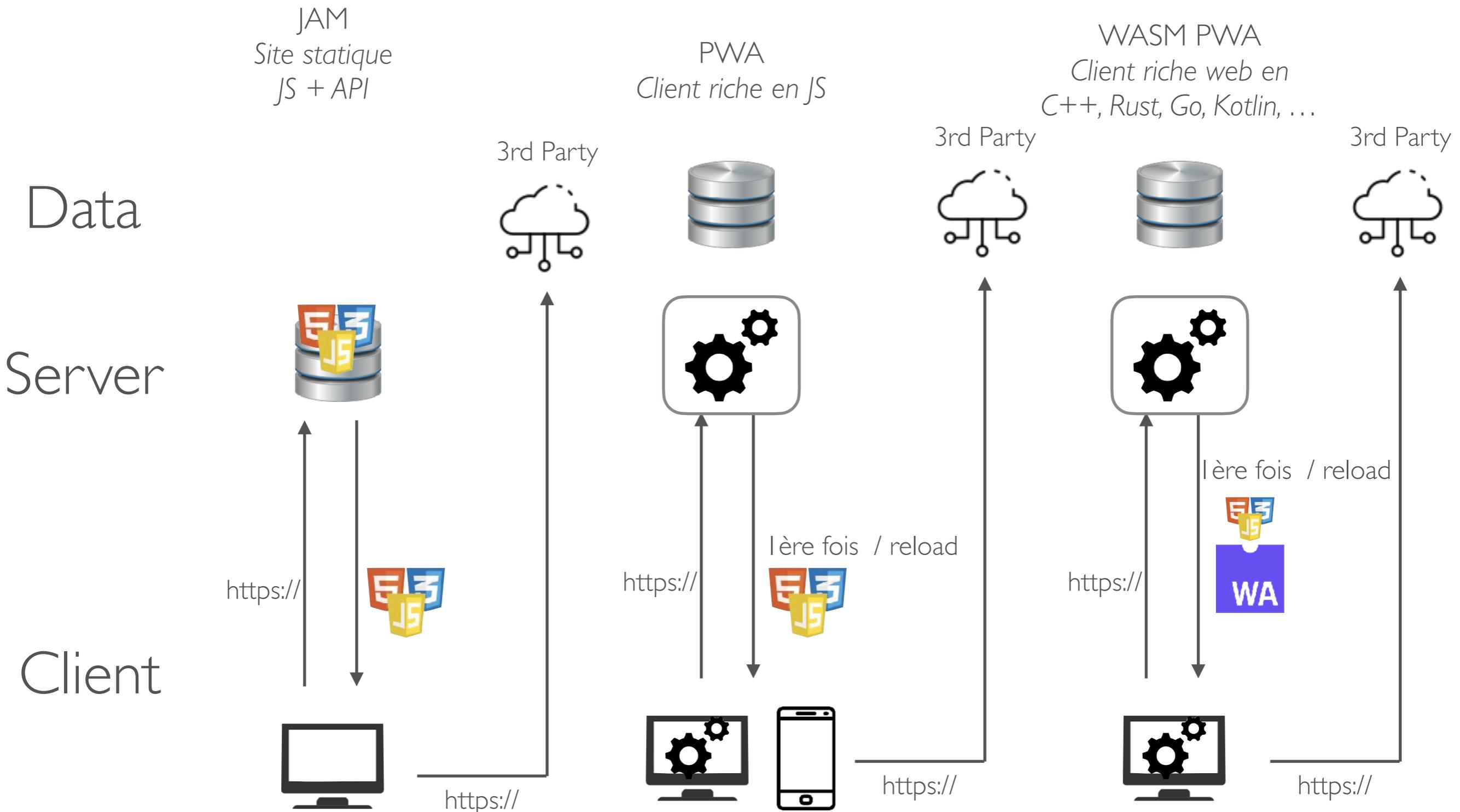
SERVERLESS  
*Client lourd +*  
Someone else server

3rd Party

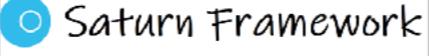


# QU'EST CE QU'UNE APP ?

## 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)

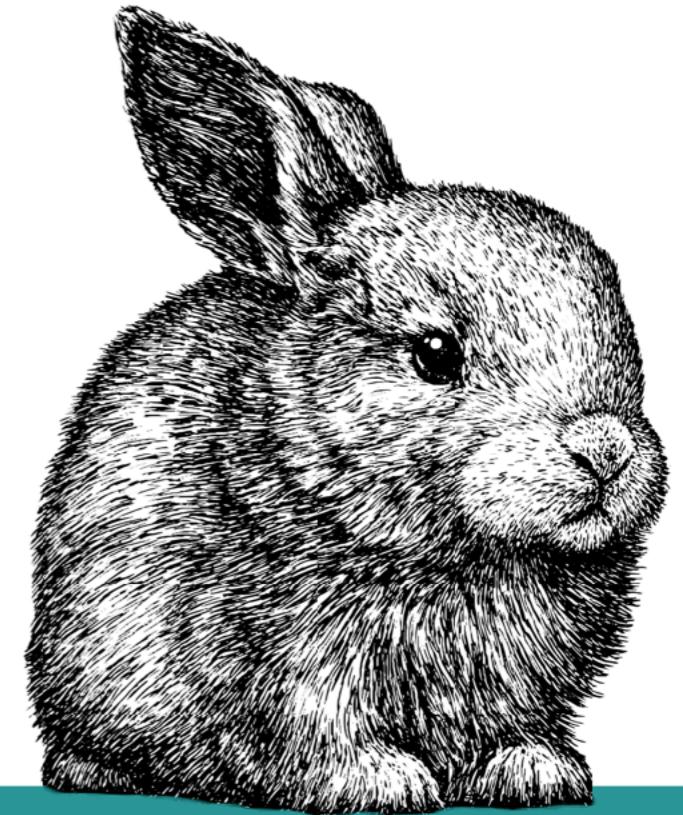
Montée en puissance des architectures « middleware » face aux « vieux » MVC ; dans certains cas d'autres patterns sont intéressants (machines de Moore, CQRS) ;

=> 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 :

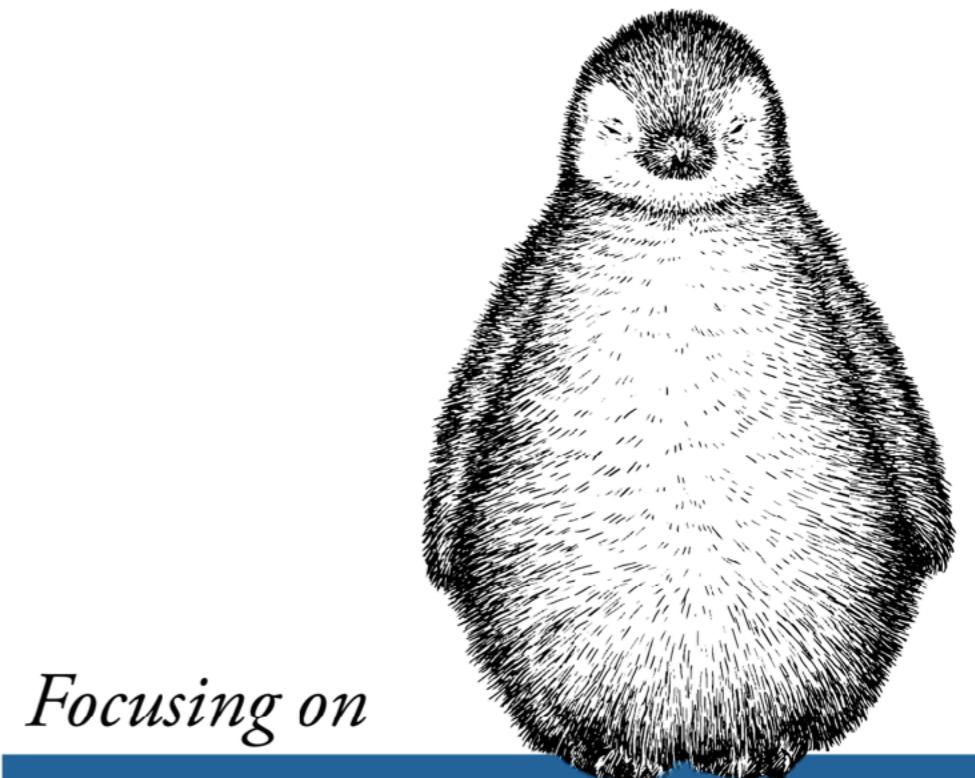
- [PostgREST](#) : plugin postgresql
- [CouchDB](#) : DB document

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

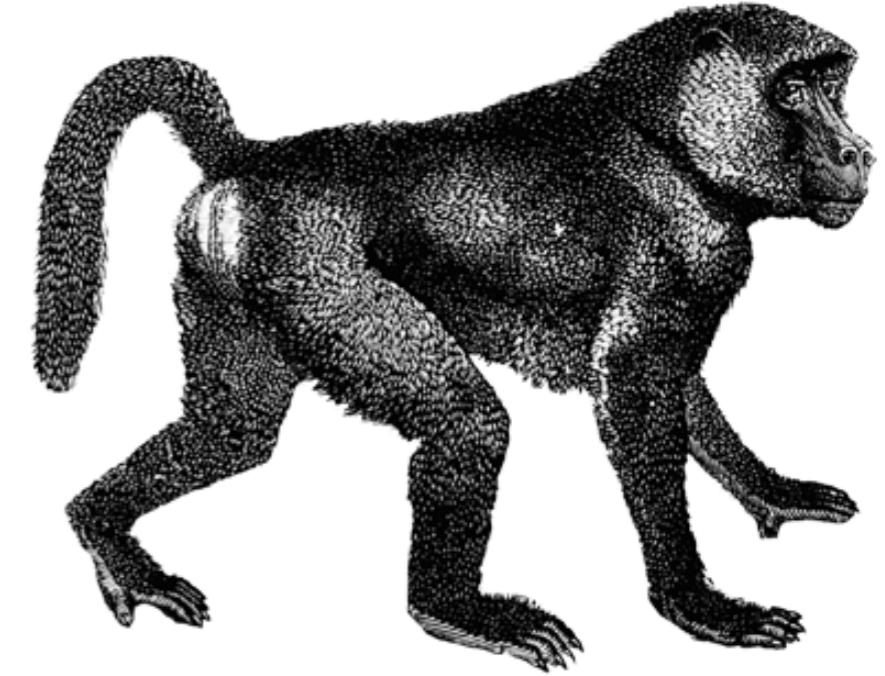
Langage « intéressant » mais « dangereux »

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

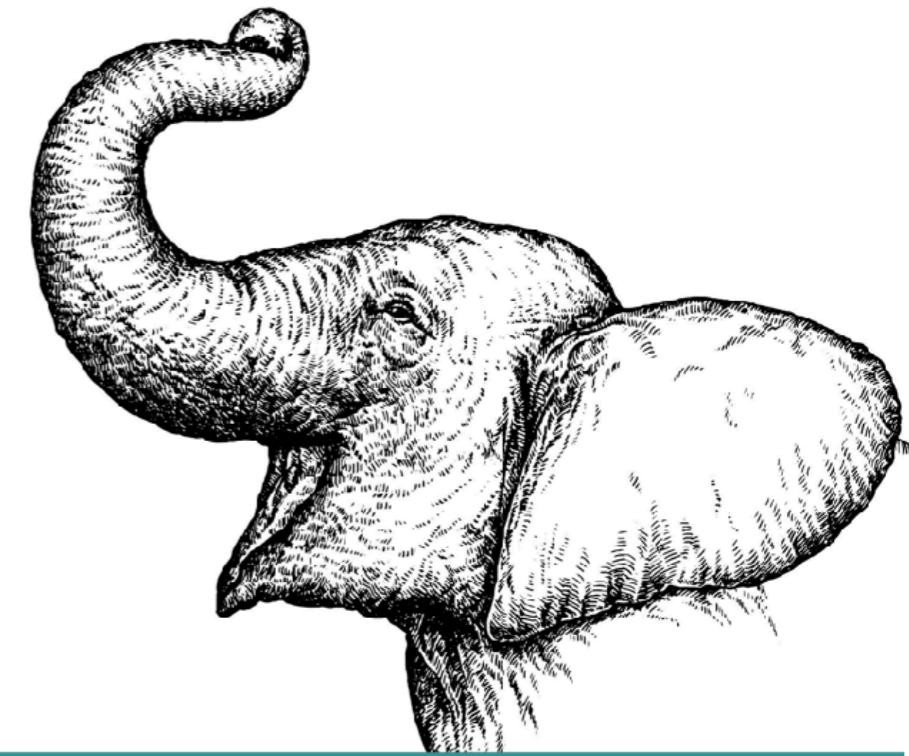
Le W3C ne voulait pas de JS comme byte code

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*



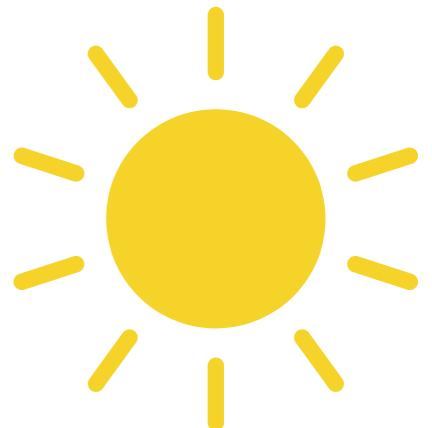
# It Depends

*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

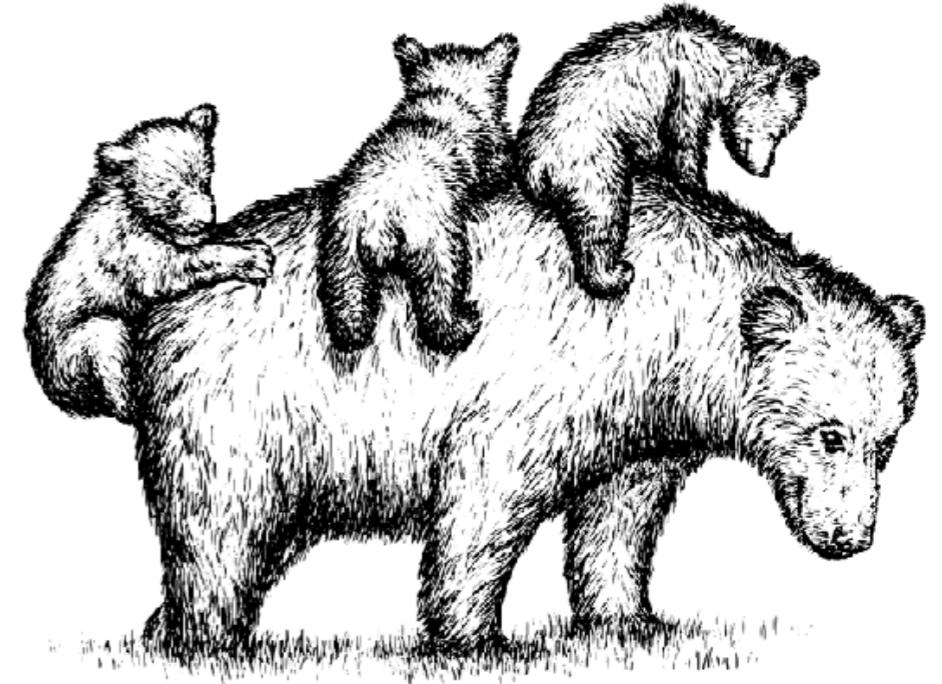
Désigne un logiciel qui n'est pas microservice

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



# THE BOSS

SI VOUS VOULEZ ALLER PLUS LOIN

Renforcement :

🎥 [Des microservices aux migroservices](#)

🎥 [Problèmes rencontrés en microservice](#)

🎥 [Hype Driven development](#)

2 talks / 3 de Quentin Adams 🙏

Diversification :

📚 [Eloquent Javascript \(free ebook\)](#)

🎥 [La Novlang dans 1984](#)



**TERMINEZ VOS TP  
PRENEZ LE TEMPS DE CONSOLIDER LE  
NOTIONS VUES PRÉCÉDEMMENT**

**LISEZ !!!  
REGARDEZ DES VIDEOS**

# **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

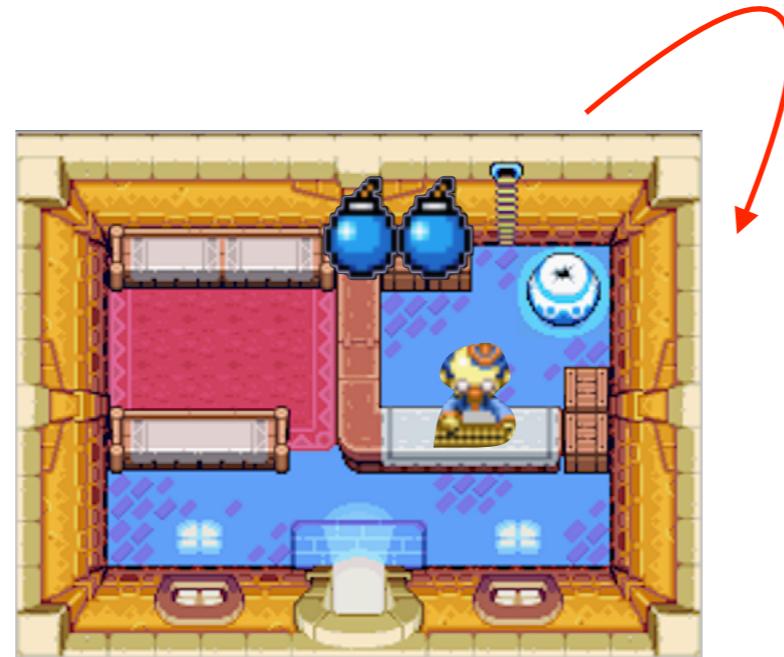
Mais l'IoT remet en avant le besoin de communiquer entre process au sein d'un même système



# COMMENT ÉVOLUE UN SYSTÈME

TEMPS

+1j : stock a bomb



# COMMENT ÉVOLUE UN SYSTÈME

REQUÊTE RÉPONSE



*I want a bomb*

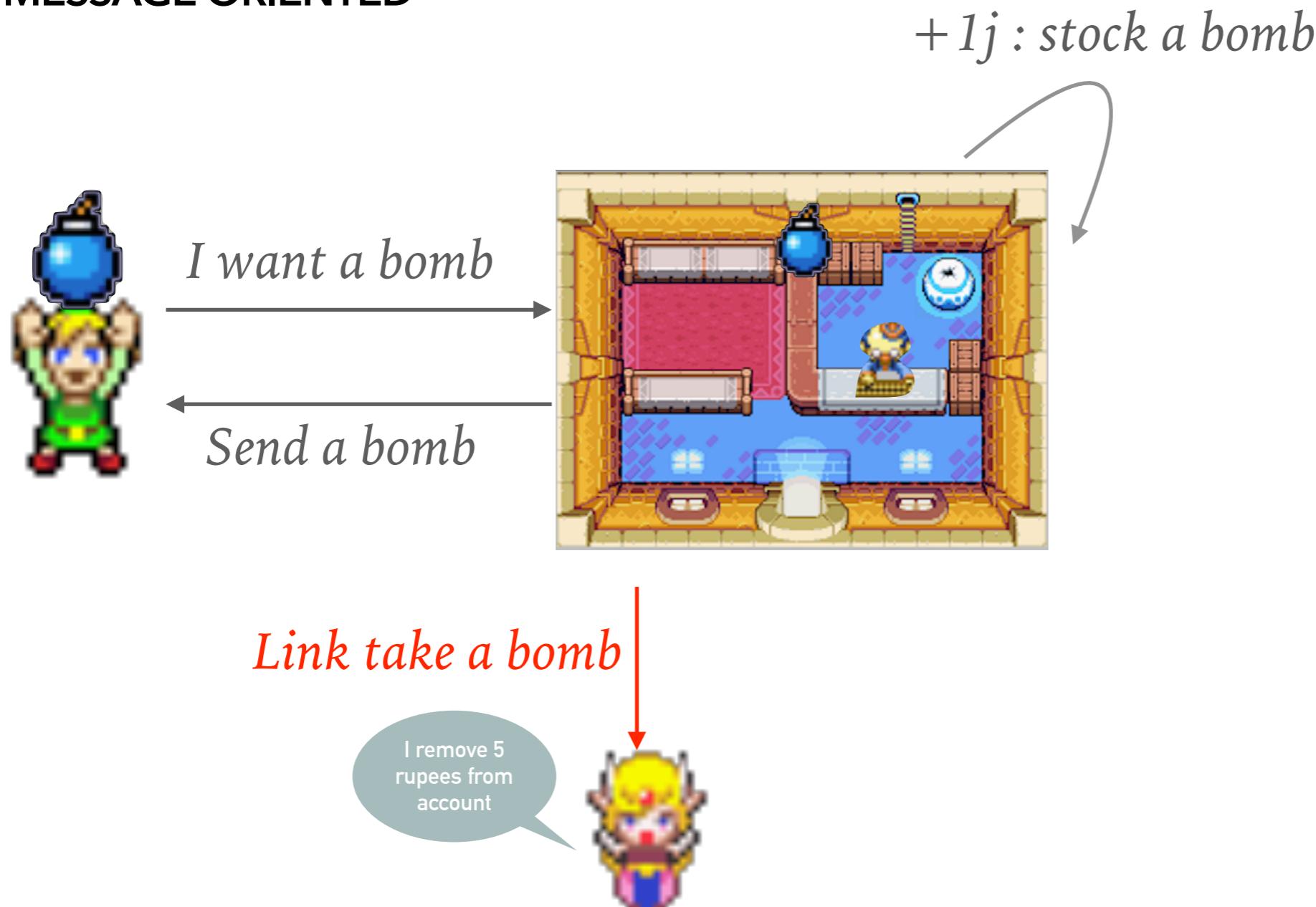
Send a bomb



+1j : stock a bomb

# COMMENT ÉVOLUE UN SYSTÈME

## MESSAGE ORIENTED



# COMMENT ÉVOLUE UN SYSTÈME

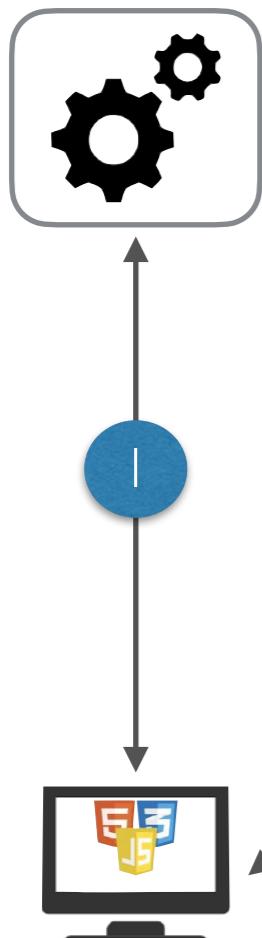
EVENT ORIENTED



# 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 » ?

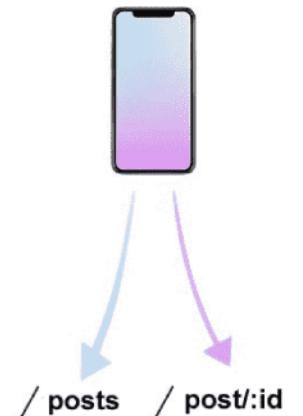
# CHOISIR LE BON OUTIL

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)

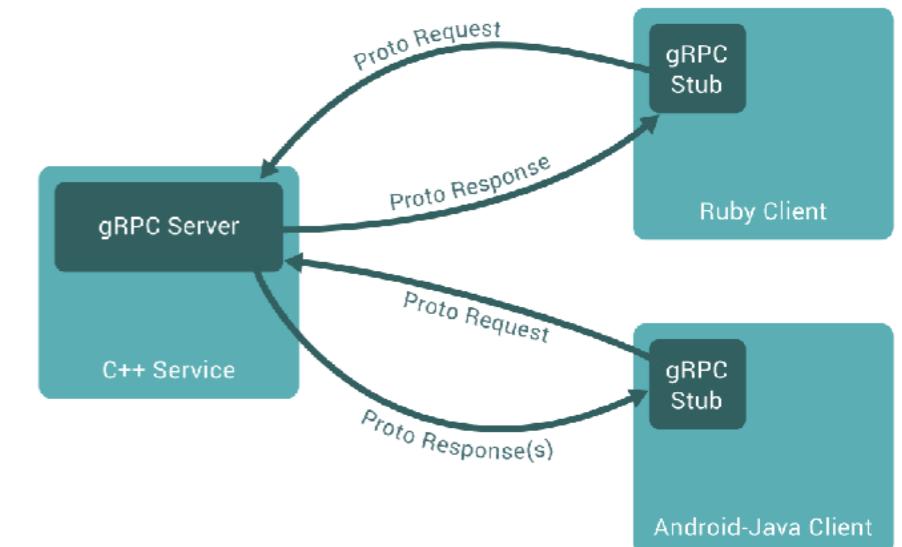
4 types de services :

- 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)

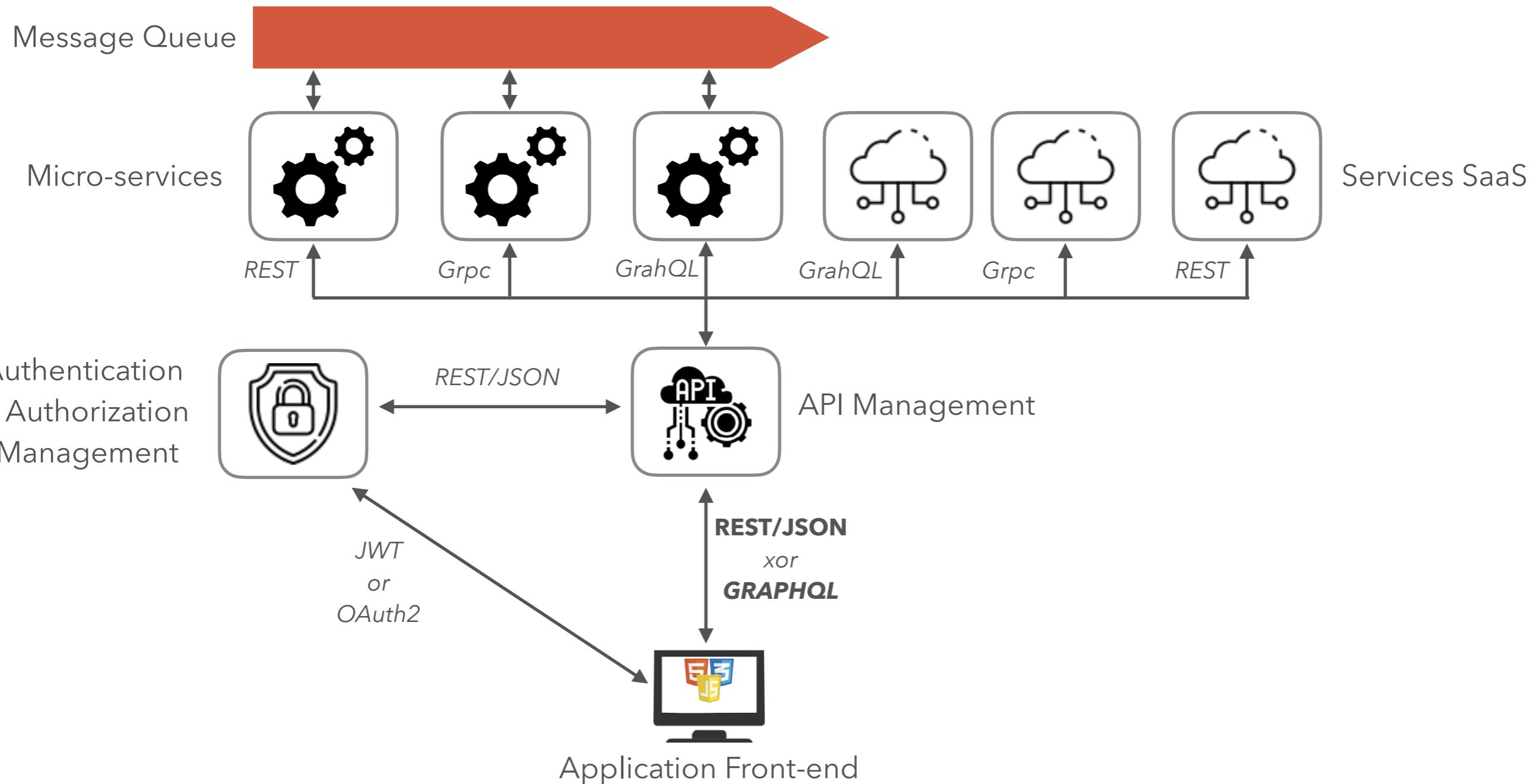
Publié par Google en 2015 ; maintenant le format par défaut des API Google

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



# LE VRAI VISAGE D'UN SI MICROSERVICES

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



# 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 !



# **TP : AUTH API**

# THE BOSS

SI VOUS VOULEZ ALLER PLUS LOIN

Renforcement :

🎥 [Build Pokemon cards with react / GraphQL](#)

📝 [REX Coursera migration GraphQL](#)



# **QUALITÉ DU SI**

---

COURS 7 - DECENTRALIZED WEB

# THE QUEST

XXX

XXX



# TAKE AWAY

XXX

XXX



**TP : DAPP**

# THE BOSS

SI VOUS VOULEZ ALLER PLUS LOIN

Renforcement :

XXX



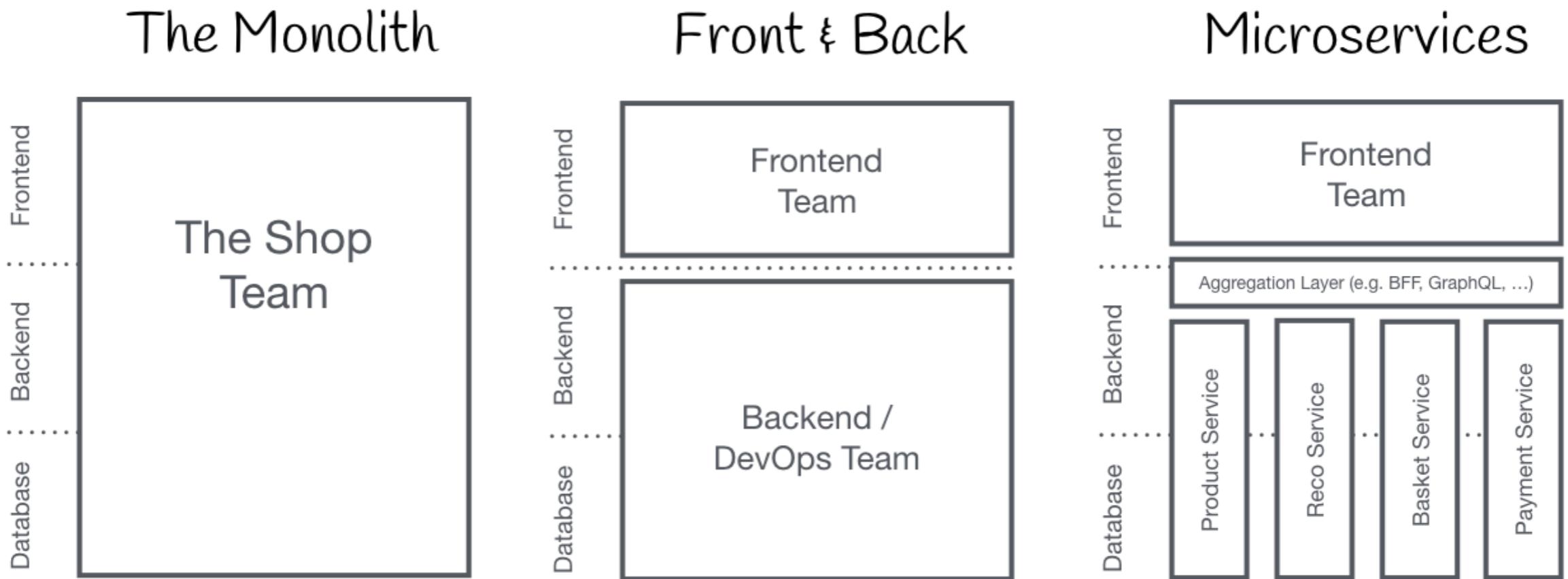
# MICROFRONTEND

DES MICROSERVICES AUX MICROFRONTEND



# LE PROBLÈME

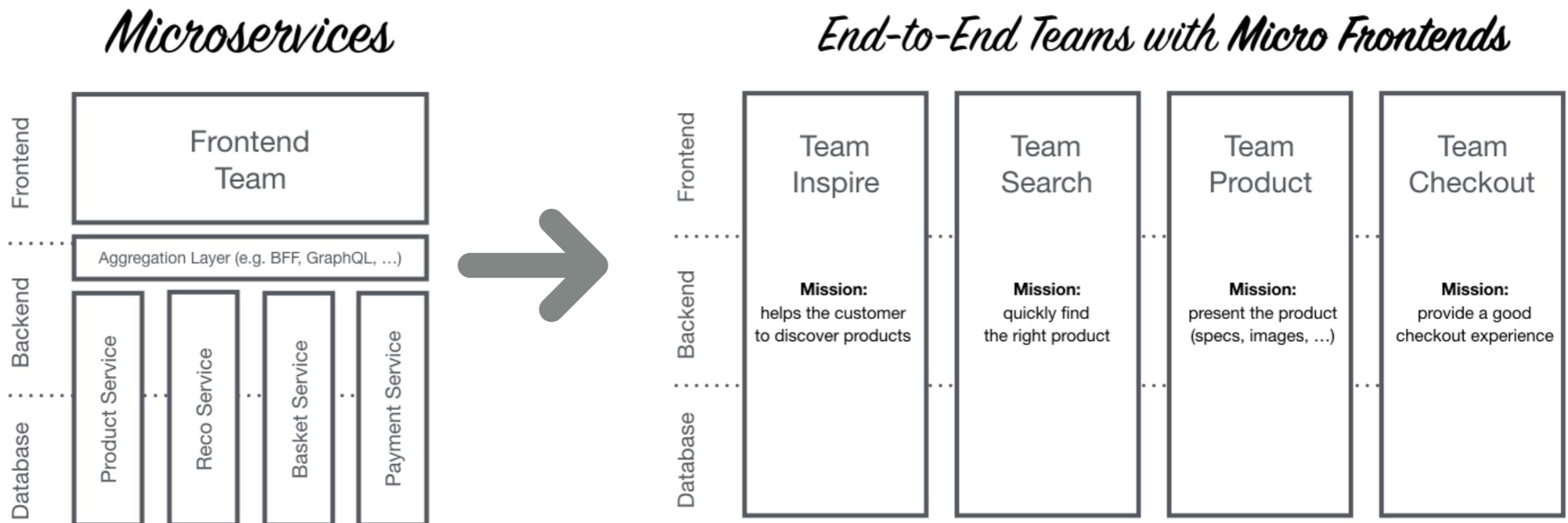
AVEC LES MICROSERVICES L'ÉQUIPE FRONT EST DEVENU LE GOULOT D'ÉTRANGLEMENT



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

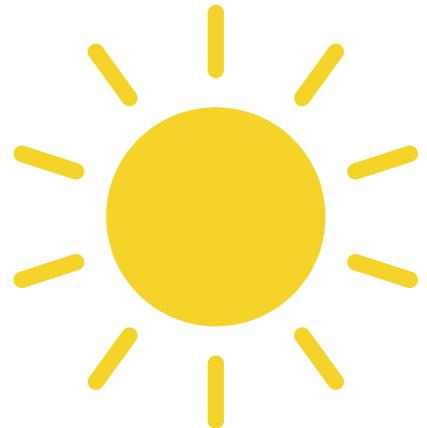
# L'OBJECTIF

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**



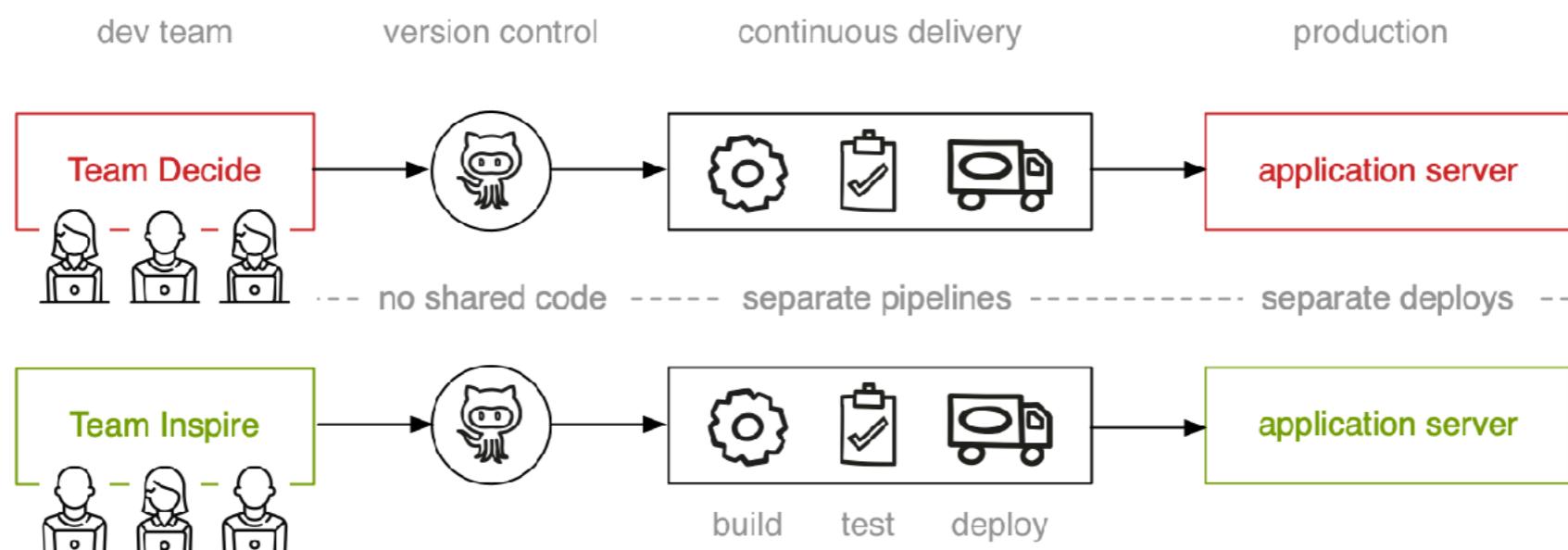
**~~Le front devient adhérent de tous les services !~~**

# 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 man holding a child and the text "Ensemble, bientôt. Idées cadeaux pour les fêtes". Below the banner is a grid of 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

- Under 25 €
- Entre 25 & 50 €
- Entre 50 & 100 €
- Sélection créateurs

31 Phillip Lim KIMONO STYLE TRAPUNTO STITCH - Ch...  
Zign Pullover - royal blue  
31 Phillip Lim SERGE - Pantalon de survêtement - black  
HaltDog Salmons G-HOED-GHEEDEROS 3 ADY UNDEX - E...  
Tiger of Sweden JOSEPH - Pantalon de survêtement - black

**Zign Pullover**  
19,99 € TVA incluse

★★★★★ 6

Couleur: royal blue

XL (en taille Z)

Acheter au panier

MON PANIER

19,99 €  
3,50 €  
Total : 23,49 €

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

Livraison : Total : 3,50 €

Retour garantie sous 180 jours

Échange possible

Matière et envergure

Détails du produit

Taille & coupe

Eco-responsabilité

Zign

4.3/5

Je donne mon avis

**Mon panier (1 article)**

Zign Pullover - 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

Total : 23,49 €

COMMANDER

Voir plus >

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

- à partir de 27,99 € JIMARCO JJCONNOR CHECK - Chino - dark grey
- à partir de 38,49 € JIMARCO JJCONNOR CHECK - Chino - dark grey
- à partir de 27,99 € à partir de 39,99 € MARCO BIONI - Chino - black
- à partir de 27,99 € à partir de 39,99 € ONSMARK PANT STRIPE - Pantalon classique - dark grey

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 SPA

DES SPA PARTOUT ... FRAMEWORK DEPENDENT ... ÉVOLUTION TRÈS RAPIDE

2014

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

2016

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



2017

```
class StatelessComponent2017 extends React.PureComponent {  
  render(props) {  
    ...  
  }  
}  
class StatefulComponent2017 extends React.Component {  
  constructor(props) {  
    super(props);  
    ...  
  }  
  render(props) {  
    ...  
  }  
}
```

2019

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

+ Mixins  
+ Flux

+ HOC  
+ REDUX

+ Render props  
+ Context

+ Hook  
+ Context

+ API changes

+ API changes

# 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 vont voir 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



# 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](#)



# TP : AUTH FRONTEND

## OBJECTIF

Website Page Title  
http://www.example.com/register

### Mon service

Email

Mot de passe

Déjà de compte ? [Se connecter](#)



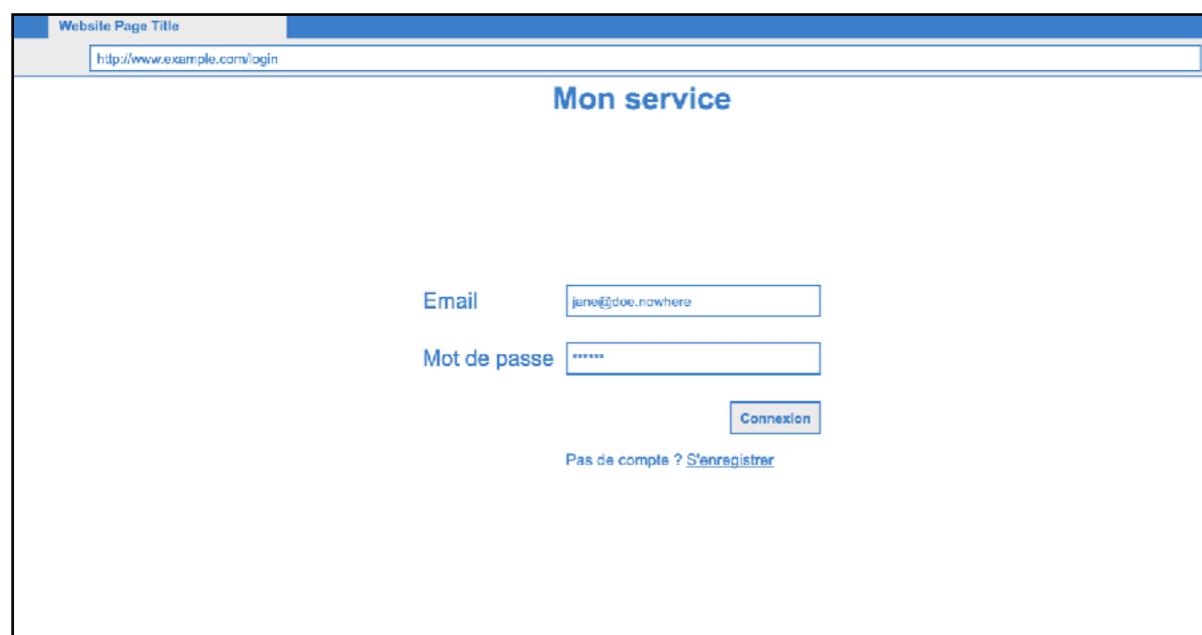
Website Page Title  
http://www.example.com/login

### Mon service

Email

Mot de passe

Pas de compte ? [S'enregistrer](#)



Website Page Title  
http://www.example.com/profile

### Mon service - connecté

#### Informations

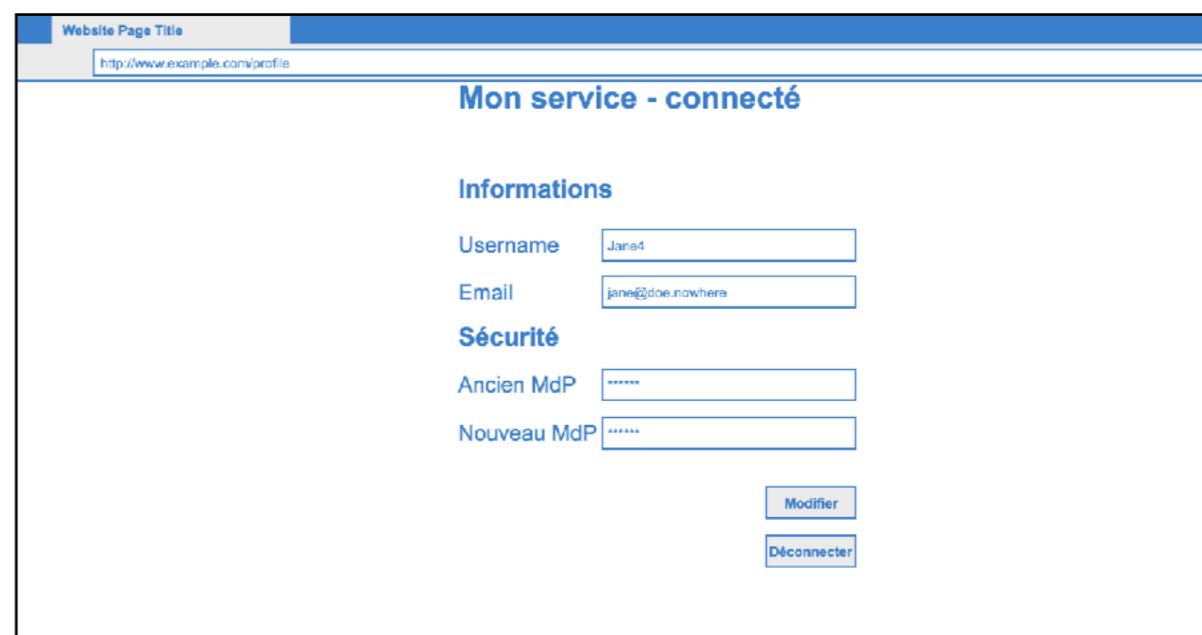
Username

Email

#### Sécurité

Ancien Mdp

Nouveau Mdp



# TP : AUTH FRONTEND

TEA + REACT

Nous avons vu avec Catstagram comment ADT + TEA permettent de faire des UI plus sûre.

Vous n'aurez pas toujours la possibilité de travailler avec un langage offrant un ADT : la majorité des projets FRONT sont codés en JS ou TS !

Au prix d'un outillage important et de bonnes pratiques, il est néanmoins possible de réduire les risques

Ce TP vise à vous montrer comment faire des applications FRONT (plus) sûres

Le TP vise à présenter certains outils et à mettre en oeuvre des bonnes pratiques

Le setup des outils est hors-scope, vous pourrez aisément trouver des tuto à ce sujet si nécessaire

# TP : AUTH FRONTEND

## ÉCOSYSTÈME REACT : UNE MINUTE DE LUCIDITÉ

**Vous avez fait un TP react ... mais en mode 2016 ... difficile de trouver de l'information de formation à jour tant le framework a évolué depuis 2014**

**Il a gagné le marché des frameworks, FB a pris l'ascendant en terme de « softpower », ce n'est plus sa priorité de former correctement les devs**

**Des leaders d'opinions « bénévoles » qui s'en servent pour leur personal branding ... et qui changent tous les 2 ans ... avec peu de culture de dev ... même sur la maintenances des grosses libs**

**Beaucoup de choses ont évolué pour faciliter la vie de nouveaux arrivant dans le développement UI ... mais sont une horreur à maintenir après 2 ans de vie d'application : les influenceurs ne feront pas votre TMA**

**Ce n'est pas propre à react, c'est même assez courant sur toutes les technos frontend, vu le rythme d'innovation, il est facile de « percer »**

**Autant de « vision » que de projet ... c'est normal react est une lib qui laisse beaucoup de liberté dans l'architecture du projet (vs framework opiniated à la Angular)**

**C'est pas simple de débuter React en 2021**

# TP : AUTH FRONTEND

## COMPOSANT REACT

Seul 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 COMPOSANT AVEC DES CLASS, N'UTILISEZ PAS LES LIFECYCLES

# TP : AUTH FRONTEND

## STATE MANAGEMENT REACT

A la différence d'une fonction graphique (ocaml-vdom / elm), un composant peut disposer d'un état local.

Nous utiliserons Redux pour gérer l'état application avec le middleware Redux-loop qui reprend le pattern Elm comme vu avec ocaml-vdom

Nous appliquerons la stratégie vu au cours 4:

- La gestion de la logique applicative doit être stockée dans l'état global (Redux)
- La logique de composant réutilisable peut être stockée dans l'état local (datepicker, info sur un champs, drag&drop, ...)

*A priori vous n'avez pas besoin d'état local pour ce TP, sauf si vous voulez vous amuser à mettre indicateur de complexité sur un mot de passe*

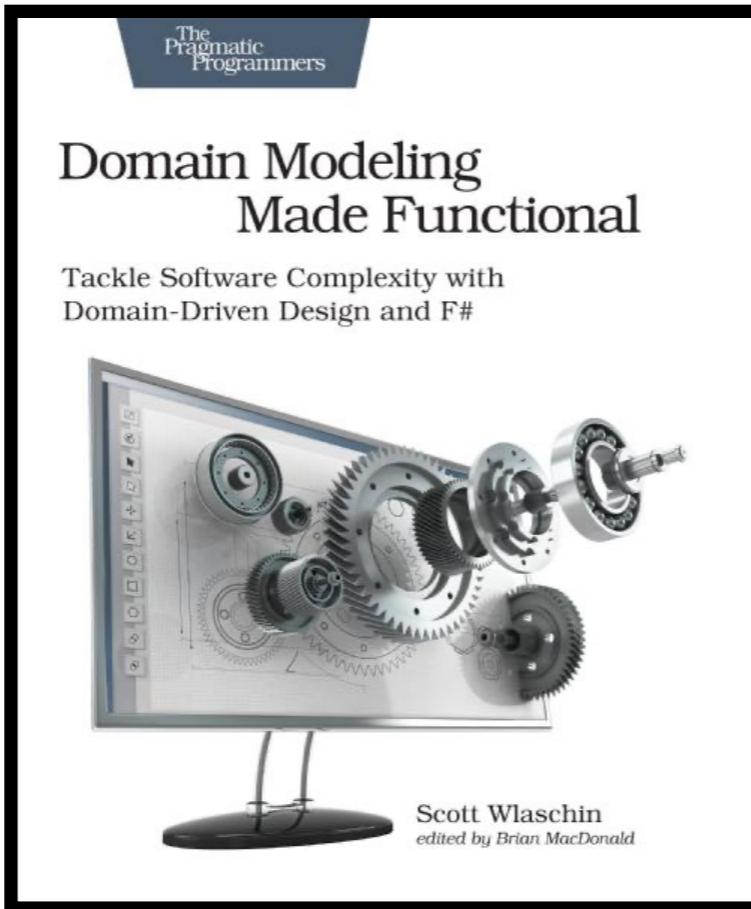
# FINAL BOSS

LISEZ ! A METTRE ENTRE TOUTES LES MAINS

**Product Owner :**

**Architectes :**

**Chefs de projets :**



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

