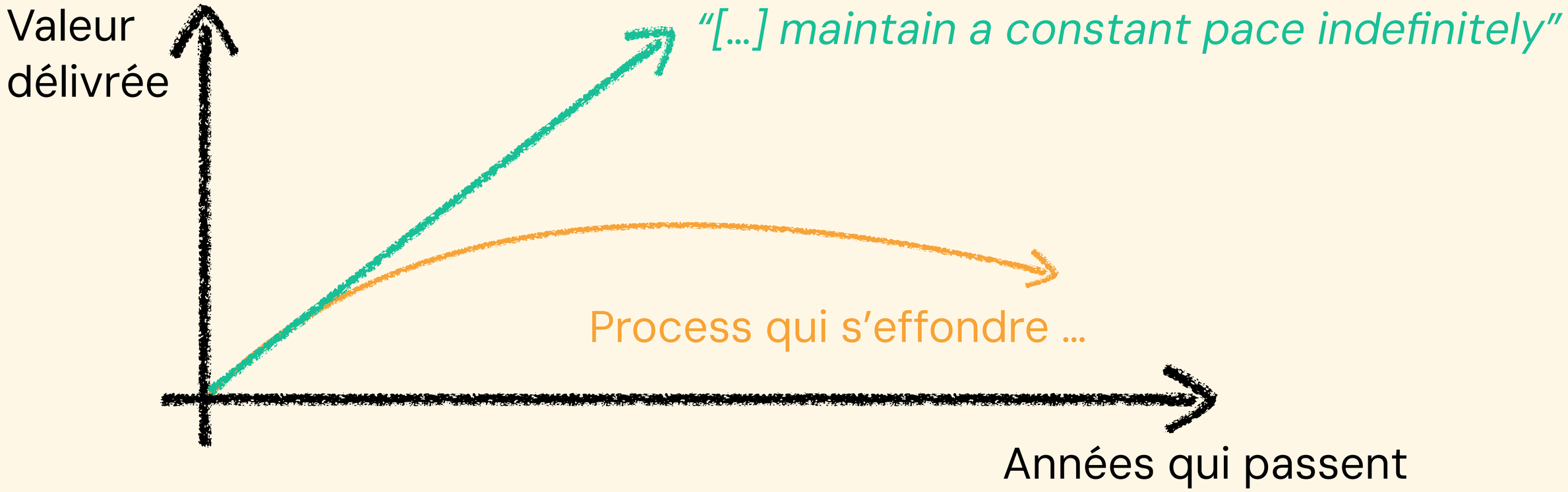
Comment les tests de validation *black-box* **renforcent** votre *Agilité*

Journée agile 2024 by PYXIS



enspirit

Plein d'open-source dev & devops

Du logiciel pour les humains

Approche 360°

enspirit

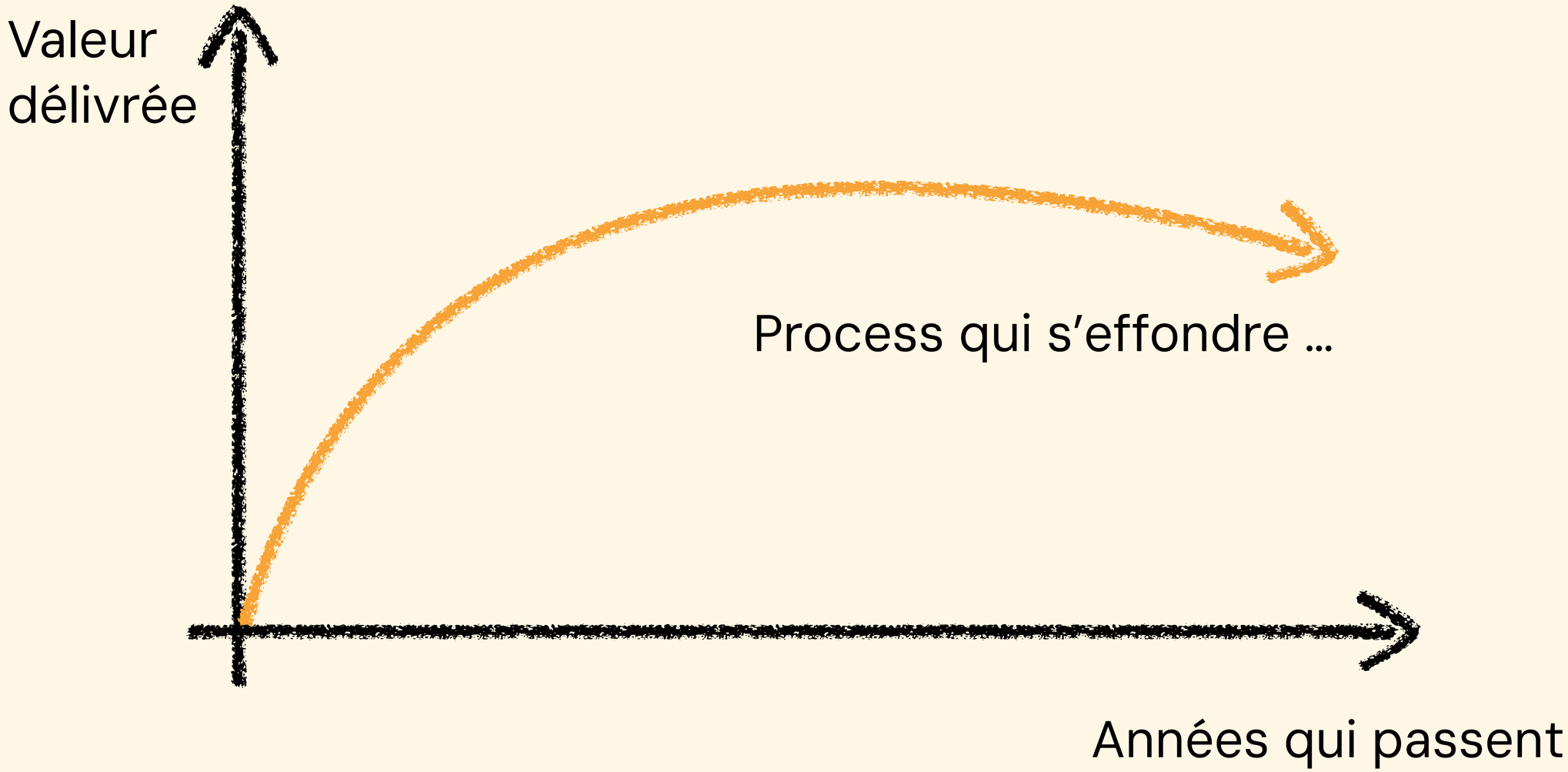
klaro.cards

L'info au bon endroit et au bon moment

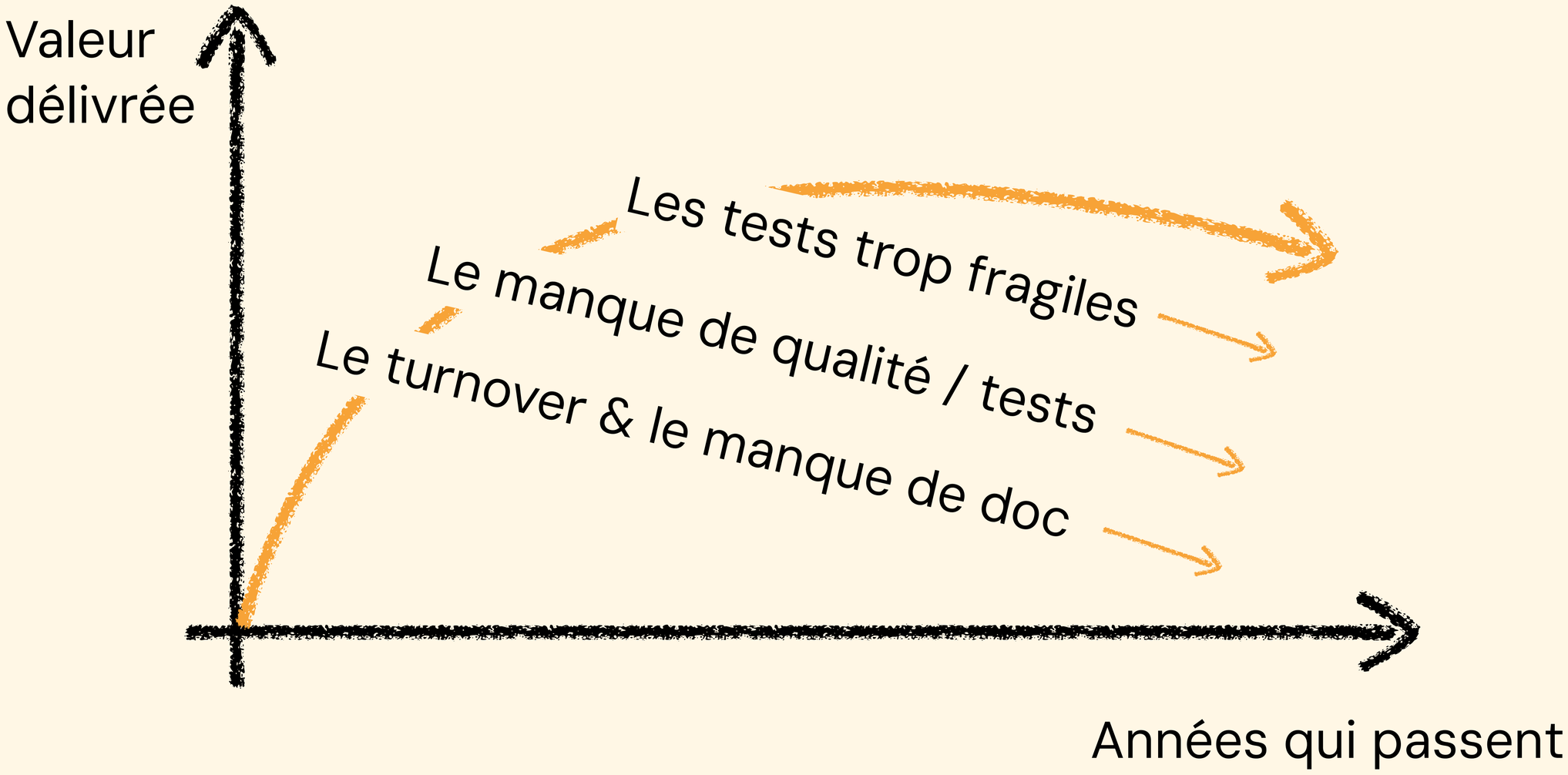
Et si on y voyait plus clair ?

@blambeau - Bernard Lambeau

Comment rester *Agile* dans la longueur... ?



Quelques root causes ...



Mon Saint Graal : une suite de tests …

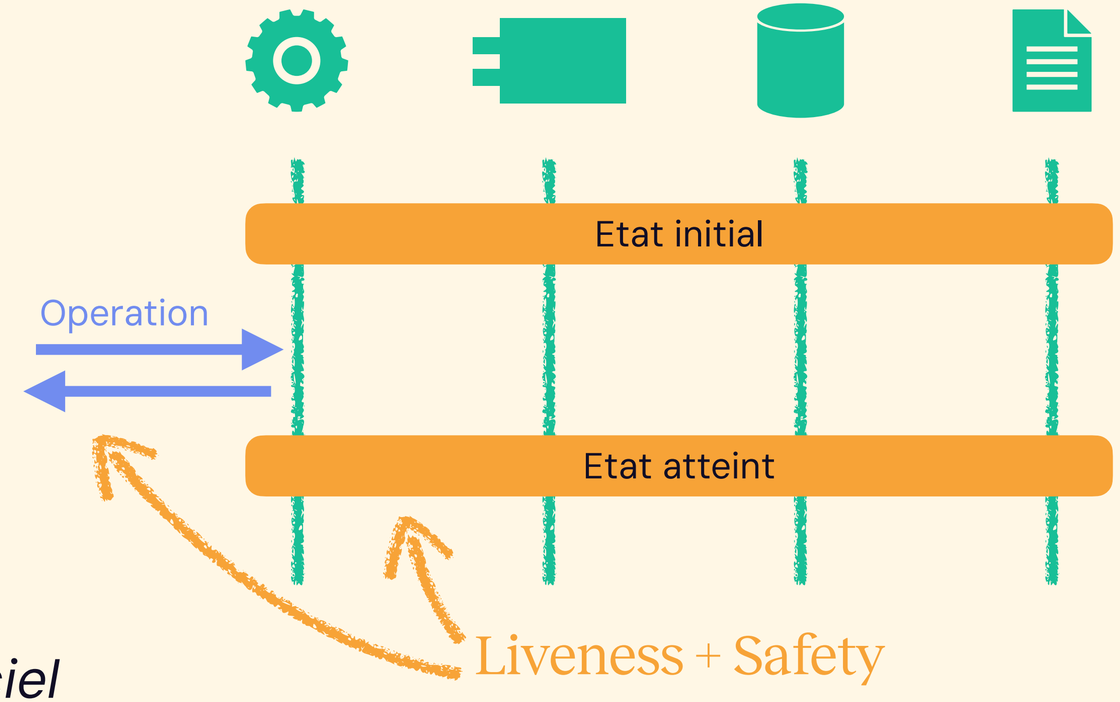
10 ans de R&D

- Stable dans la longueur, malgré les *refactorings*
- Technology agnostic, la plus déclarative possible
- Pas trop lente
- A charge du développeur, pas du QA ou de l’analyste
- Qui (re)-documente techniquement
- Qui permette de réfléchir aux requis (aussi non fonctionnels)
- Qui offre des effets leviers sur la qualité

Webspicy : black-box testing du backend (API)

Une API publique (p.ex. RESTFul) offre une belle opportunité qualité / testing :

- *un point stable du logiciel*
- *un point vulnérable du logiciel*
- *un point documentable du logiciel*
- *un point facilement appelable du logiciel*
- *un point facilement mesurable du logiciel*
- *un point facilement proxyable du logiciel*
- *un point technologiquement neutre du logiciel*



https://github.com/enspirit/webspicy

Quels méthodes de test utilisez vous ?

TYPES

- ✓ Unitaire
- ✓ Composant
- ✓ Intégration
- ✓ Système
- ✓ Acceptance
- ✓ Performance
- ✓ Sécurité
- ✓ Utilisabilité
- ✓ Conformité
- ✓ Compatibilité
- ✓ Smoke / Stress

PROCESSUS

- ✓ TDD
- ✓ BDD
- ✓ Cycle en V
- ✓ QA Manuel
- ✓ Break things

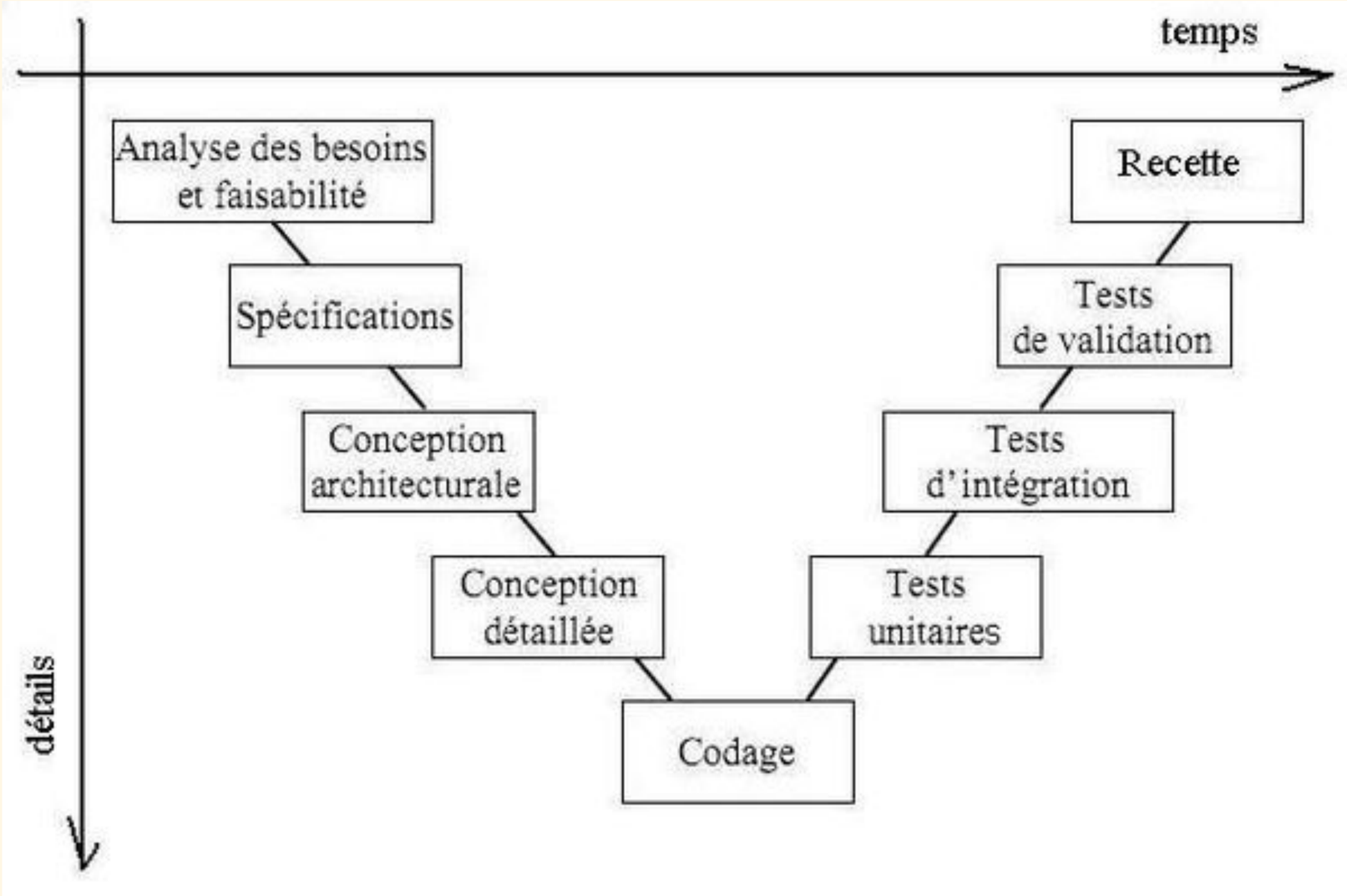
https://app.wooclap.com/DAQLFE

Journée agile 2024 by PYXIS

Slide intentionnellement laissé blanc

enspirit

Vade retro ; un bon vieux Cycle en V



D’où un framework avec un focus un peu inhabituel (?)

TYPES

- ✓ Unitaire
- ✓ **Composant**
- ✓ Intégration
- ✓ **Système**
- ✓ Acceptance

- ✓ Performance
- ✓ **Sécurité**
- ✓ Utilisabilité
- ✓ Conformité
- ✓ **Compatibilité**
- ✓ Smoke / **Stress**

PROCESSUS

- ✓ **TDD**
- ✓ BDD
- ✓ **Validation**
- ✓ QA Manuel
- ✓ **Black box**

https://app.wooclap.com/DAQLFE

Retour d'expérience

20

Projets logiciels “instrumentés”

Simple site web, systèmes complexes, SaaS, lib open-source, etc.

10

Stacks différentes, sans problème majeur

Dont Node.js, TypeScript, Ruby, PHP, Java, Scala, C#, …

15

Développeurs formés

Séniors & juniors (mais ⚠️)

Retour d’expérience

Ca aide, beaucoup

Demande un investissement initial et quelques shifts de mindset

Approche équilibrée & flexible

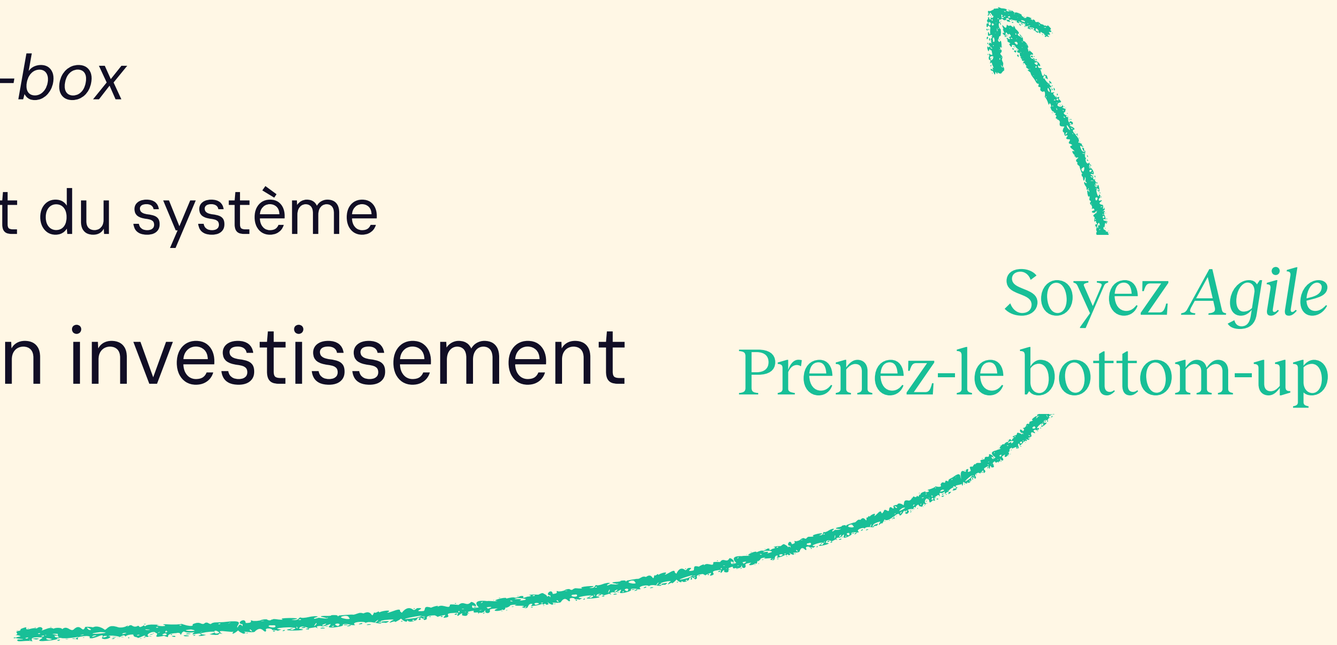
Tests composants & système, en + de l’unit., intégration & acceptance

Les développeurs apprécient (mais ⚠️)

Aide à l’architecture, aux requis, à la communication interne

Plan de la suite

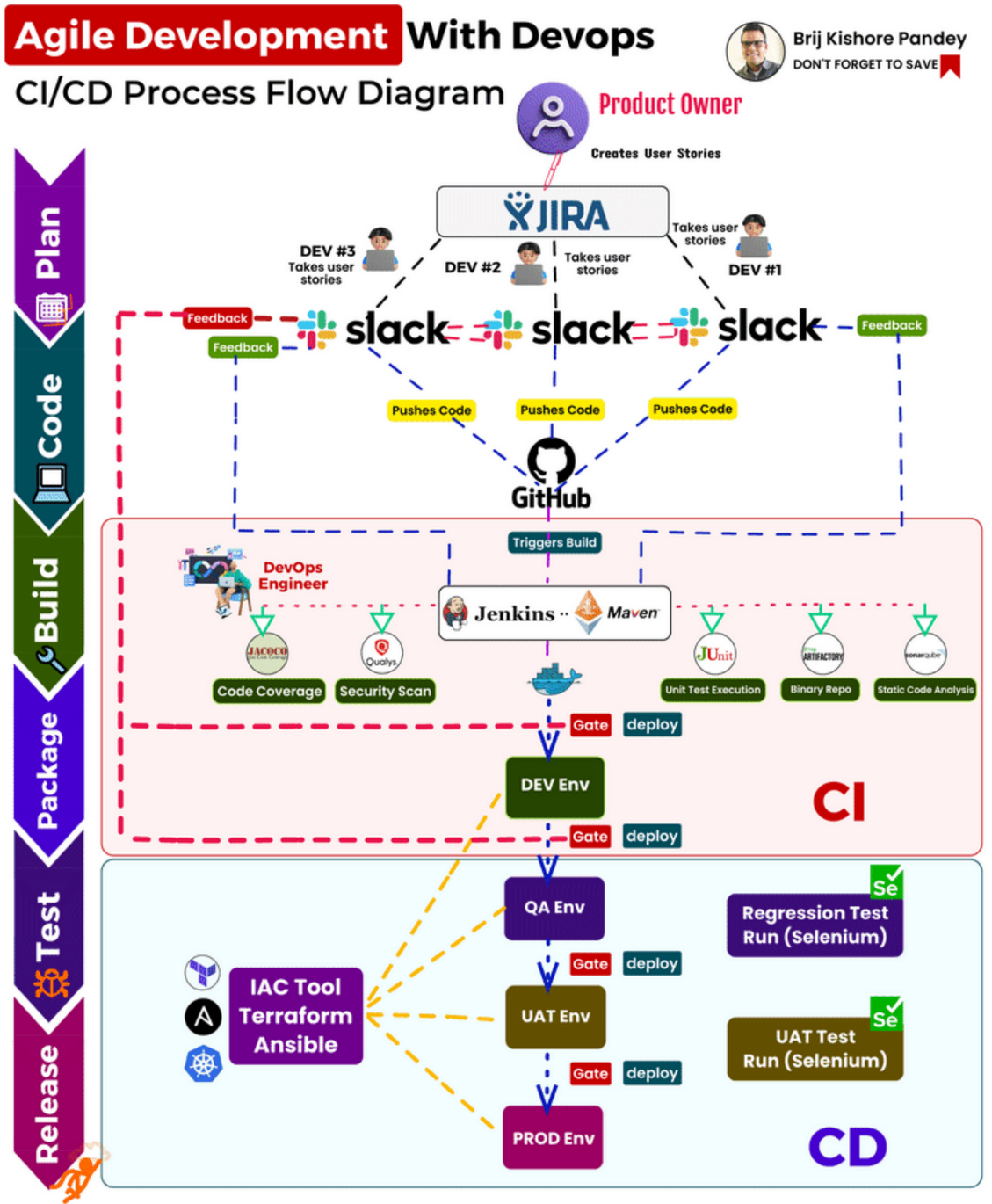
- Investissement initial dans l'architecture du système
  - Ingrédient n°1 : monorepo + make
  - Ingrédient n°2 : modèle *black-box*
  - Ingrédient n°3 : maîtriser l'état du système
- Profiter pleinement de son investissement
  - Tests d'acceptance
  - Tests de validation webspicy



Point de départ : *Agile* (?)

Individuals and interactions
Over processes and tools

Si un développeur a besoin de tout un cluster Amazon pour reproduire, tester puis fixer un bug système, on n'est pas rendus …



Ingrédient n°1 : *make up* and go

✓

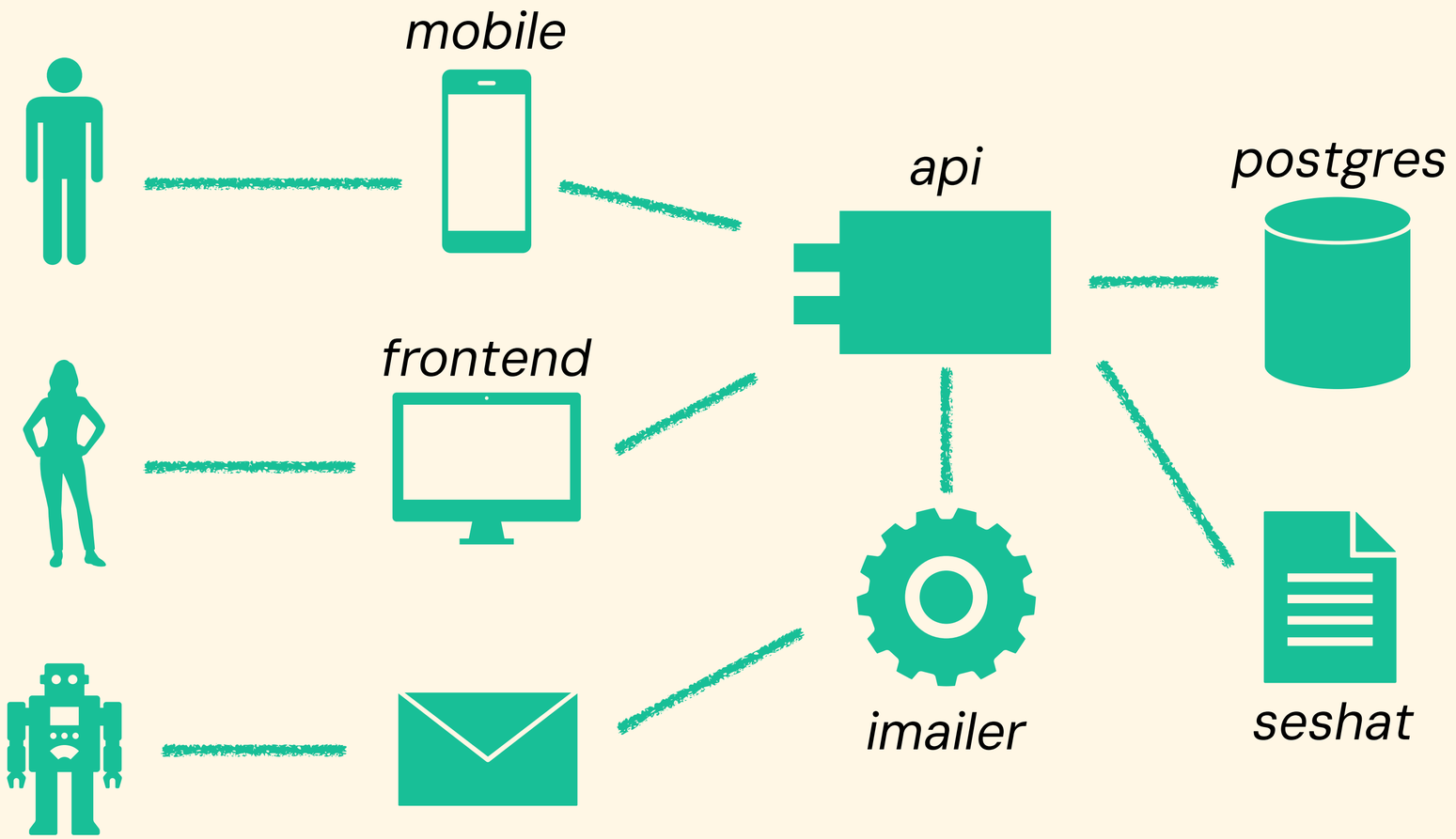
L’ensemble du système doit pouvoir s’exécuter sur la machine de n’importe quel développeur

```
git clone ; make up
```

Le développeur travaille en autonomie dans l’heure
Avec des commandes make standardisées pour tout le cycle de vie

https://github.com/enspirit/makefile-for-monorepos

Système logiciel : klaro.cards (simplifié)

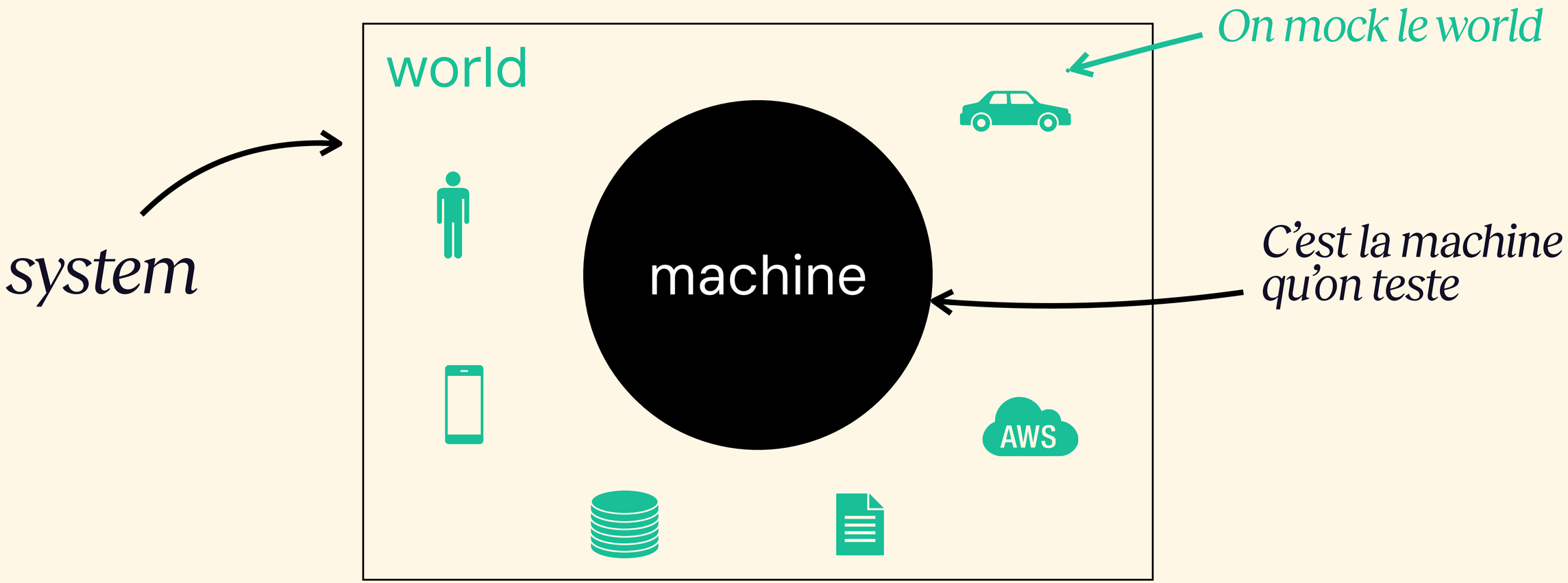


Ingrédient n°1 : monorepo + docker + make

```
monorepo
├── api                              # Un composant backend / api
│   ├── Dockerfile                   # ... chacun a son Dockerfile
│   ├── env                          #
│   │   └── devel.env                # ... variables d'env pour le développement
│   ├── makefile.mk                  # ... extensions make
│   └── ...                          # ... code source
│
├── frontend                         # Un autre composant, p.ex. frontend
│   ├── Dockerfile                   # ... structure similaire
│   └── ...
│
├── .env                             # Variables d'env globales (e.g. COMPOSE_FILE)
├── docker-compose.base.yml          #
├── docker-compose.devel.yml         # Orchestration avec docker-compose
├── docker-compose.testing.yml       #
├── Makefile                         # Makefile réutilisable
├── Jenkinsfile                      # CI & CD Jenkins/Gitlab super simple (make only)
└── config.mk                        # Configuration et cibles make globales
```

https://github.com/enspirit/makefile-for-monorepos

Ingrédient n°2 : modèle *black-box*



The World and the Machine, M. Jackson, 1995

Ingrédient n°2 : mocks utilisés chez nous

Mock de services HTTP
https://github.com/jmartin82/mmock

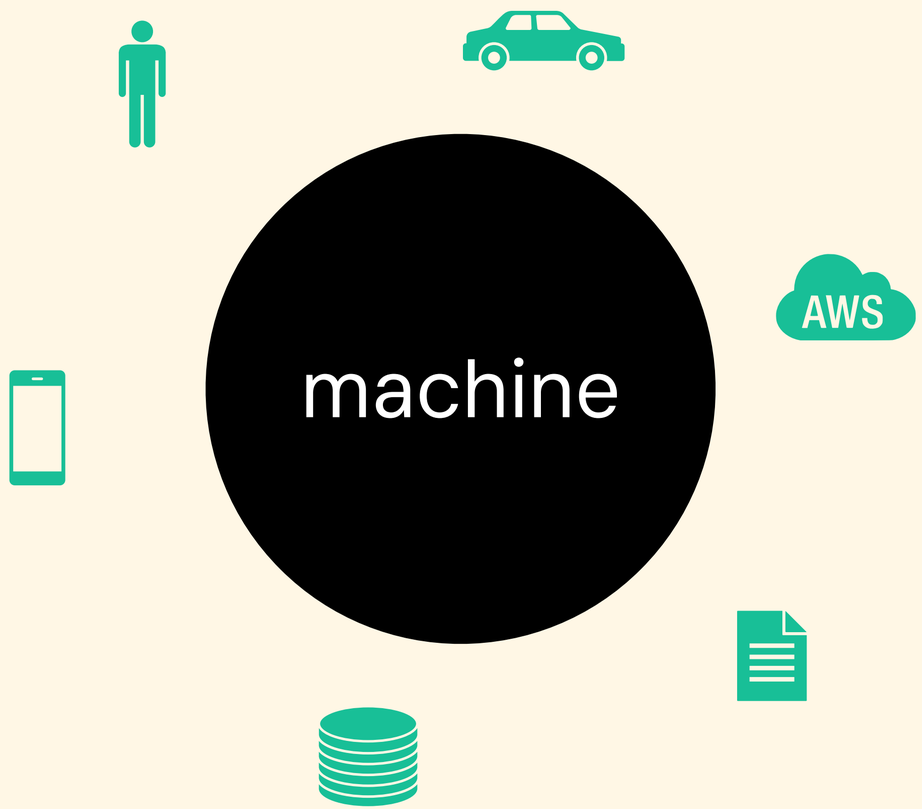
Un vrai faux serveur mail
https://github.com/gessnerfl/fake-smtp-server

Emulateur pour Firebase
https://github.com/firebase/firebase-tools

Simulateurs sur mesure
...

Dockerisations / servicisations chez Enspirit qu'on peut publier open-source

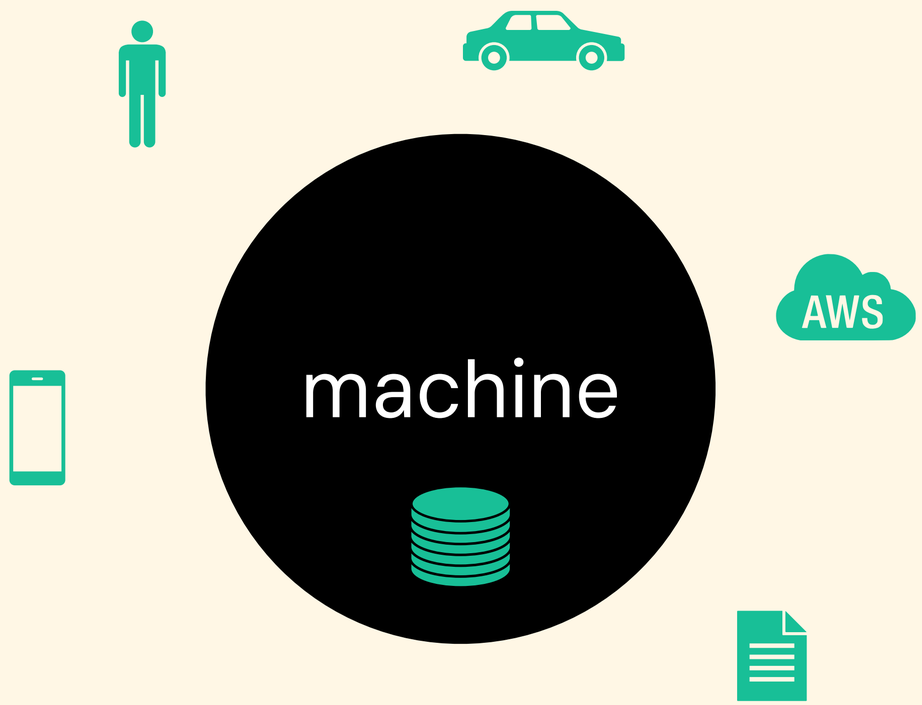
Ingrédient n°3 : *system state*



Tout ce joli petit monde a un état interne qu'il faut pouvoir

- Contrôler : état initial
- Inspecter : état atteint

Ingrédient n°3 : *system state*



Peut-être la machine a un état interne également …

… ça ne change rien dans le fond …

Vue de l’esprit des frontières de la boîte noire

Ingrédient n°3 : pour gérer l'état du système …

SQL database flush & seed as docker + http service
https://github.com/enspirit/dbagent

File storage as docker + http service (compatible S3 / GCS)
https://github.com/enspirit/seshat

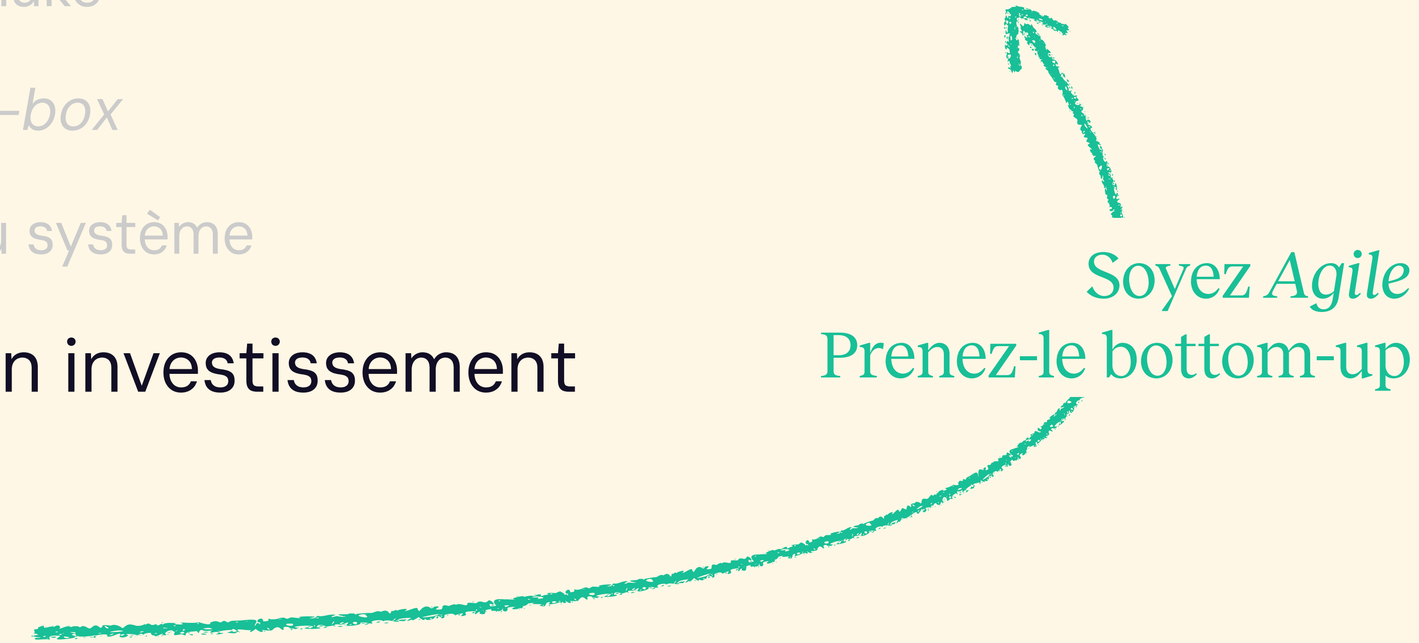
Magic API over PostgreSQL as docker + http service
https://postgrest.org/en/v12/explanations/install.html#docker

Gestionnaires d'état applicatif sur mesure
redis & autres nosql, reset de rabbitmq, …

Dockerisations / servicisations chez Enspirit qu'on peut publier open-source

On est arrivés là …

- Investissement initial dans l'architecture du système
  - Ingrédient n°1 : monorepo + make
  - Ingrédient n°2 : modèle *black-box*
  - Ingrédient n°3 : gérer l'état du système
- Profiter pleinement de son investissement
  - Tests d'acceptance
  - Tests de validation webspicy

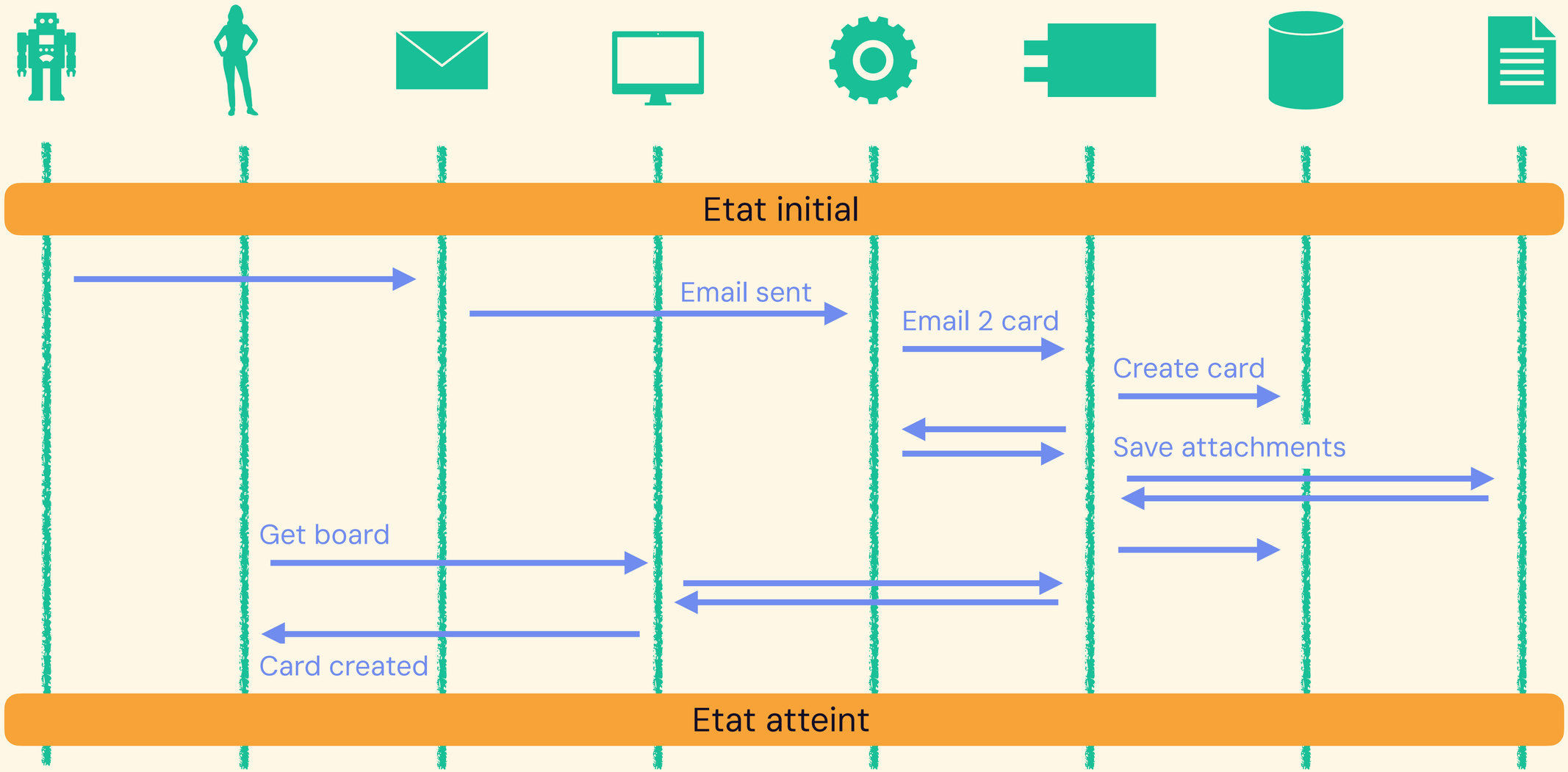


Exemple de test d’acceptance utilisateur

Feature: Email to Card

Si j’ai associé une adresse email à mon board Klaro Cards
Et qu’on m’envoie un email à cette adresse
Alors cet email donne lieu à une nouvelle carte
Le sujet de l’email est repris comme titre de la carte
Son contenu devient la description de la carte
Les filtres du board fournissent les valeurs de ses dimensions
Et l’ensemble des pièces jointes en sont les attachements

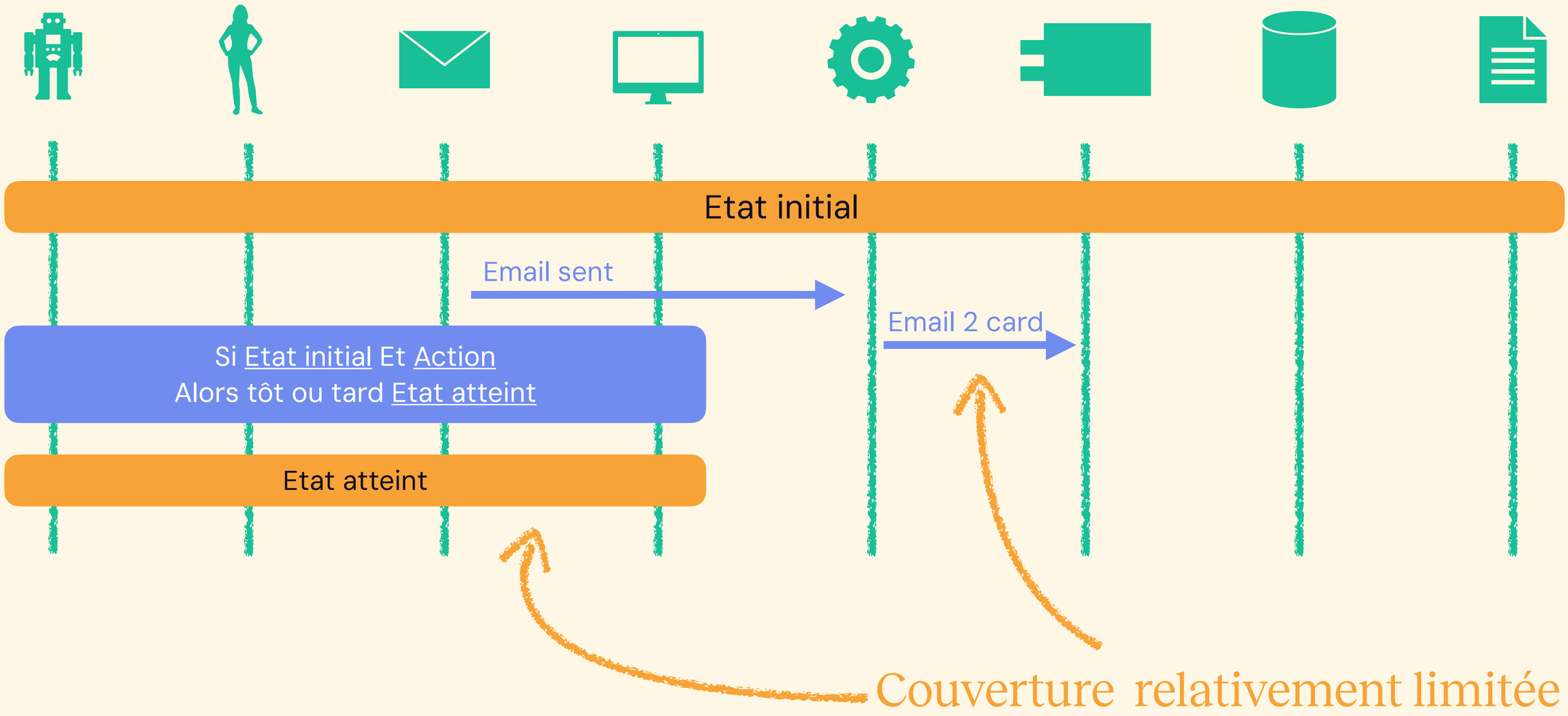
Interactions entre composants du système



Faiblesses des tests d’acceptance

- Lent(issime), donc couverture limitée
- Fragile (si mal codés ou interfaces/processus instables)
- Parfois difficile à mettre en oeuvre (p.ex. async / faux négatifs)
- Assertions trop souvent limitées à ce qu’on voit à l’interface
- Offre une documentation business, mais pas technique
- Gros gros focus sur les *happy paths* seulement

Happy paths : c'est quasi exclusivement du *liveness*



Propriétés *Liveness* vs. *Safety*

Liveness

Tôt ou tard, quelque chose de bien arrive

Safety

Les mauvaises choses n'arrivent pas

aka. *Good est toujours vrai* / Bad n'est jamais vrai

Comment exprimer cela dans des tests ?

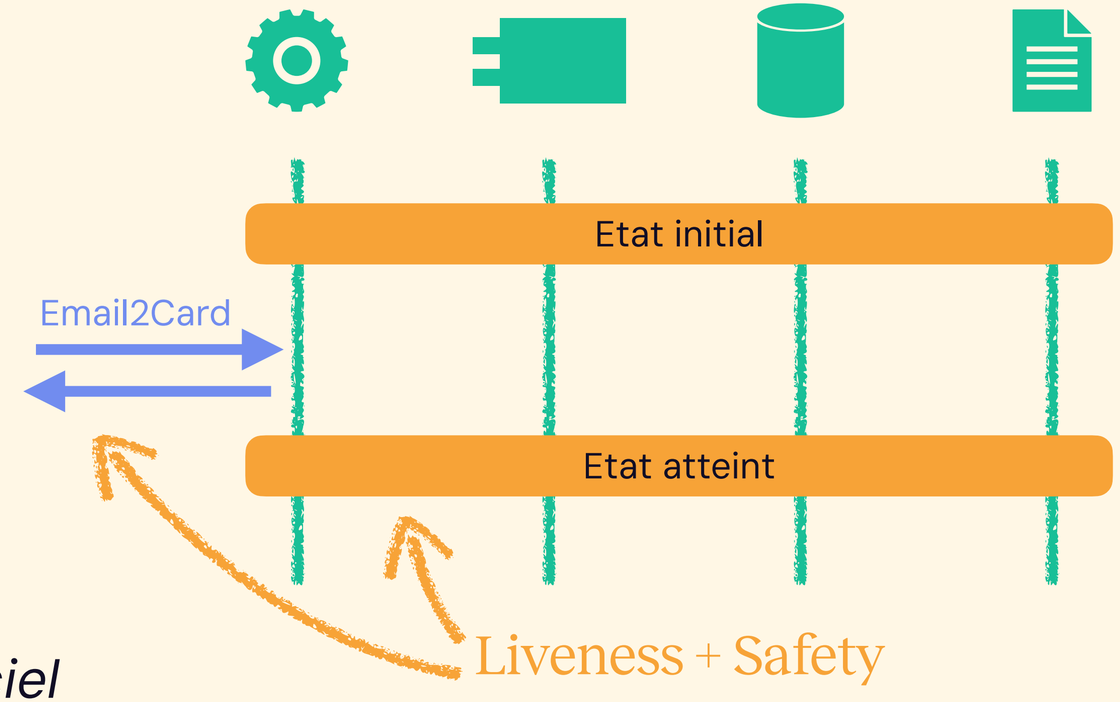
La *Safety* est omniprésente en qualité

- Toutes nos APIs valident leurs données d'entrée
- Un mot de passe, même encrypté, ne sort jamais de l'API
- L'utilisateur doit être authentifié pour toutes les opérations en écriture
- Un Content-Security-Policy strict est présent sur tous les sites hébergés
- Aucune requête synchrône ne prend plus de 250ms.
- Le refactoring de l'API est 100% backward compatible
- La base de données reste en permanence dans un état cohérent …

Webspicy : black-box testing du backend (API)

Une API publique (p.ex. RESTFul) offre une belle opportunité qualité / testing :

- *un point stable du logiciel*
- *un point vulnérable du logiciel*
- *un point documentable du logiciel*
- *un point facilement appelable du logiciel*
- *un point facilement mesurable du logiciel*
- *un point facilement proxyable du logiciel*
- *un point technologiquement neutre du logiciel*

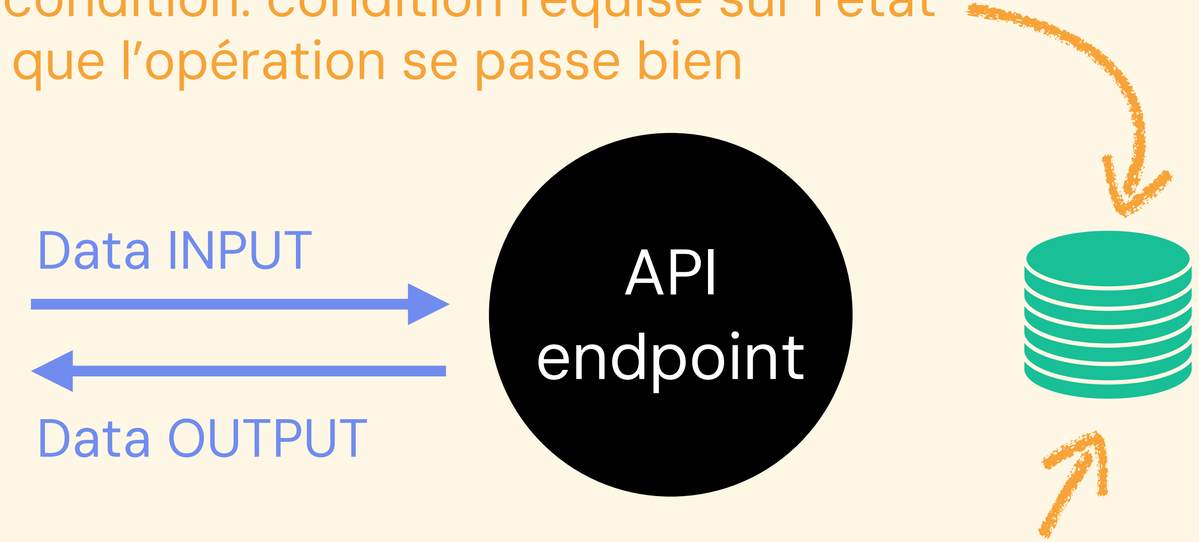


https://github.com/enspirit/webspicy

Webspicy : modèle de tests de validation black-box

POST /url, aka HighLevelOperation

PRE-condition: condition requise sur l'état pour que l'opération se passe bien



Example (Test Case)

Invocation avec un INPUT valide et sans violer la PRE

Counter-Example (Robustness)

Invocation avec un INPUT invalide et/ou en violant la PRE

POST-condition: condition requise sur l'état atteint si l'opération a réussi et est correcte

ERR-condition: condition requise sur l'état atteint si l'opération a échoué

https://github.com/enspirit/webspicy

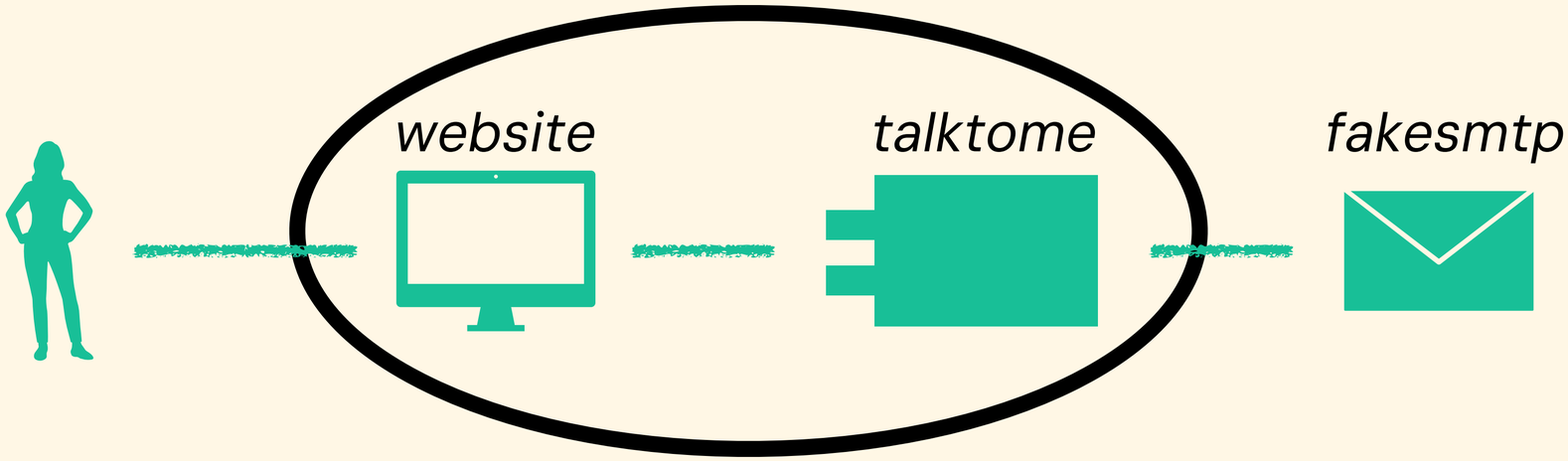
Webspicy : effets leviers

- ✓ Les schema de données permettent des contraintes arbitraires
- ✓ La validation des schema s’appliquent à tous les cas de test
- ✓ Les POST-conditions sont testées sur tous les cas de test
- ✓ Les PRE-conditions peuvent générer des cas de test (p.ex. par mutation)
- ✓ L’enforcement des PRE & POST conditions peut être déclaré et vérifié
- ✓ Collection open-source de POST-conditions réutilisables (WIP)

https://github.com/enspirit/webspicy

Démo

Système logiciel : enspirit.dev



- ✓ Le web service retourne le message attendu (+ effet levier)
- ✓ Un email est envoyé à info@enspirit.be si formulaire ok
- ✗ Aucun email n’est envoyé sinon
- ✗ Le web service valide ses données d’entrée (+ effet levier)
- ✗ L’internaute n’est pas un robot qui nous flood (+ effet levier)

Conclusion - A retenir

(piqure de rappel)

L'Agilité tient moins du super pipeline CI&CD de la plateforme de staging que de la capacité des développeurs à délivrer du code correct rapidement

(contribution)

Parmi les techniques de test, le TDD des API publiques (data in/out) est une approche viable (grâce au devops), complémentaire à l'unitaire & l'acceptance, avec des avantages propres (p.ex. effets leviers + safety + documentation)

Merci.
Questions ?