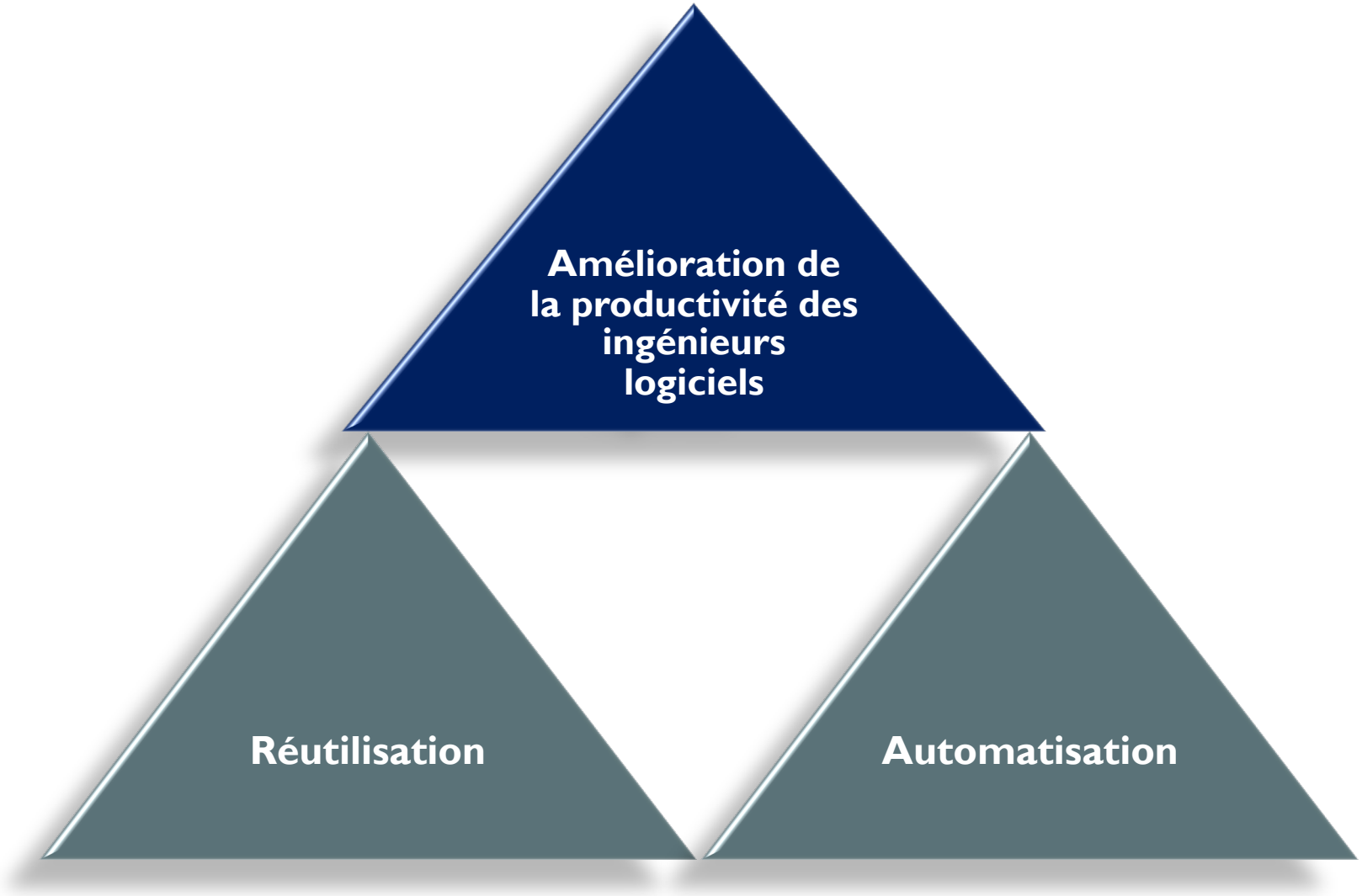
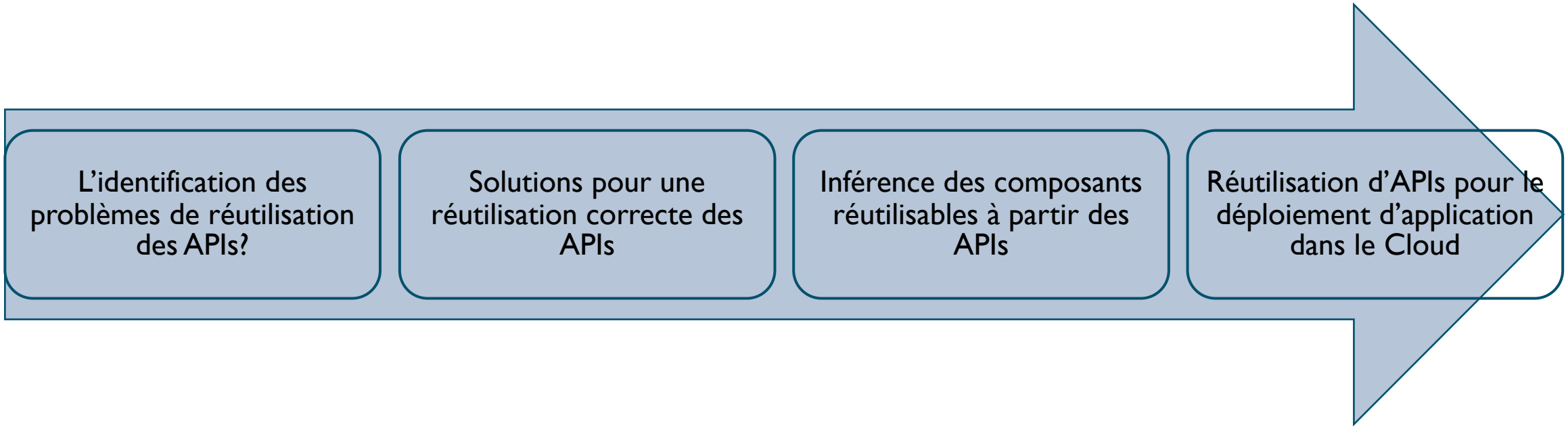
Faire face aux défis d'utilisabilité des APIs: du développement au déploiement des applications logicielles

Mohamed Aymen Saied

Portrait géneral de mes travaux de recherche



Portrait général de mes travaux de recherche



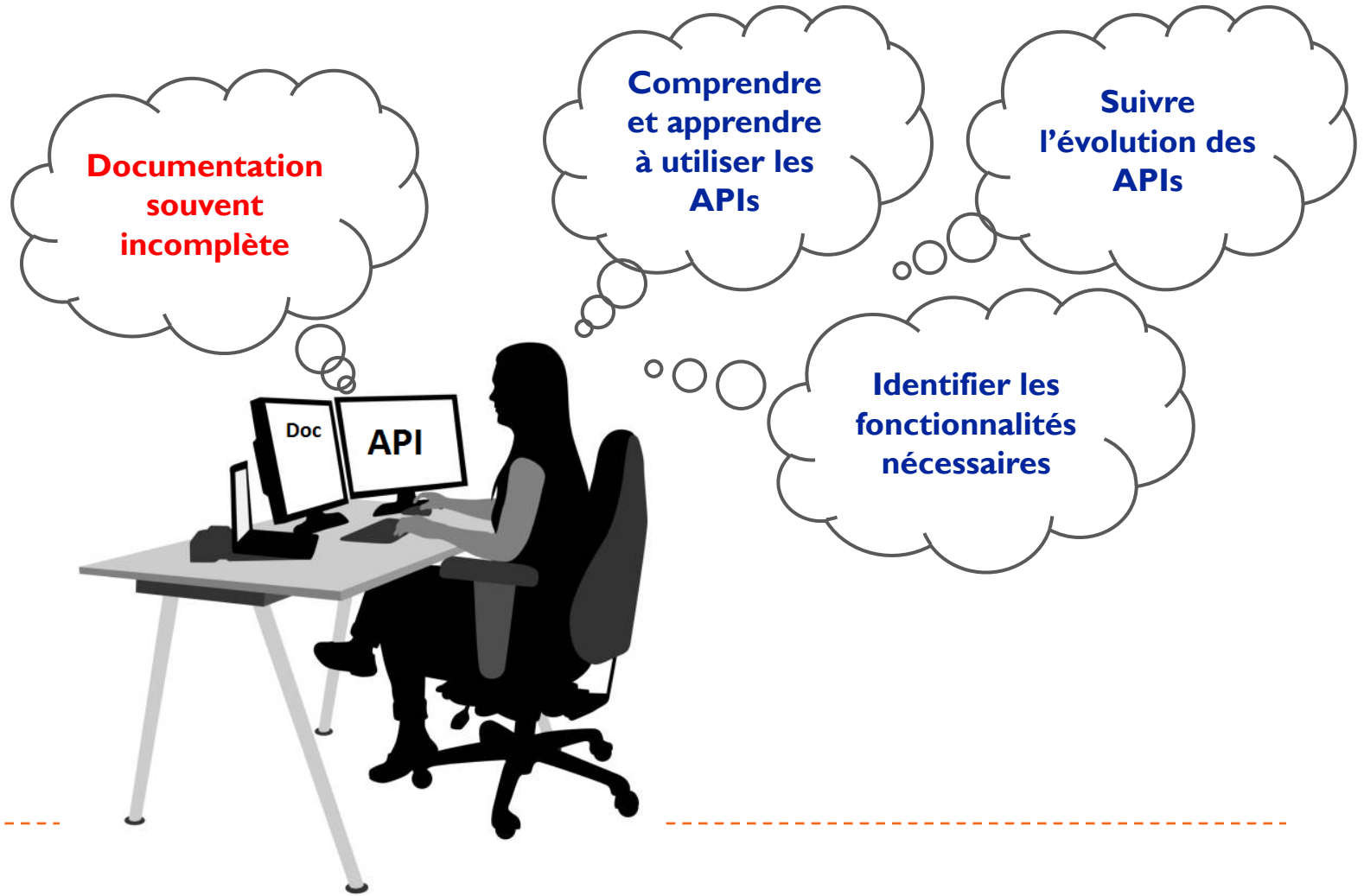
Application Programming Interface (APIs)

- Les logiciels réutilisent des fonctionnalités fournies par des librairies logicielles
- Les librairies exposent leurs fonctionnalités via des APIs
- Documentation décrit comment interagir avec l'API

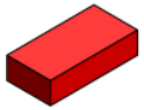
Method and Description
`add(E e)` Appends the specified element to the end of this list.
`add(int index, E element)` Inserts the specified element at the specified position in this list.
`addAll(Collection<? extends E> c)` Appends all of the elements in the specified collection to the end of this list,

Développement logiciels et APIs

- Les logiciels modernes utilisent intensivement des APIs
- L'utilisation des APIs demande beaucoup d'efforts
- La mauvaise utilisation des APIs entraîne différents types de problèmes
- Impact sur la productivité

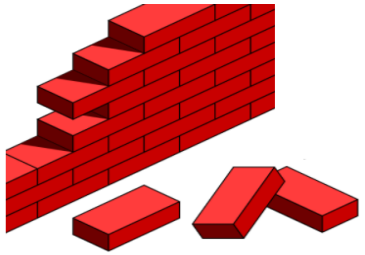


Comment améliorer l'utilisabilité des APIs?



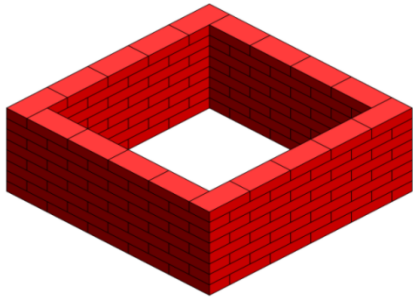
Méthode

- Pré-conditions (paramètres de la méthode)



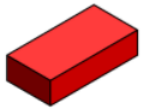
API : plusieurs méthodes d'une API

- Quelles méthodes utiliser ensemble
- Ordre et/ou contraintes d'appel

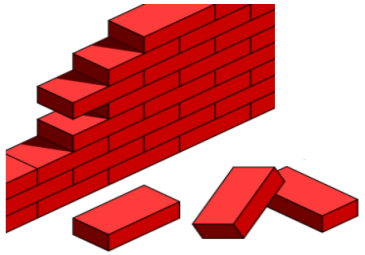


Plusieurs APIs

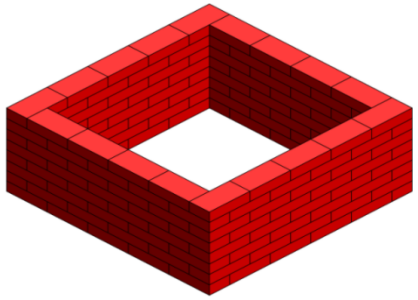
- Combinaisons possibles d'APIs
- APIs compatibles et faciles à intégrer



Méthode

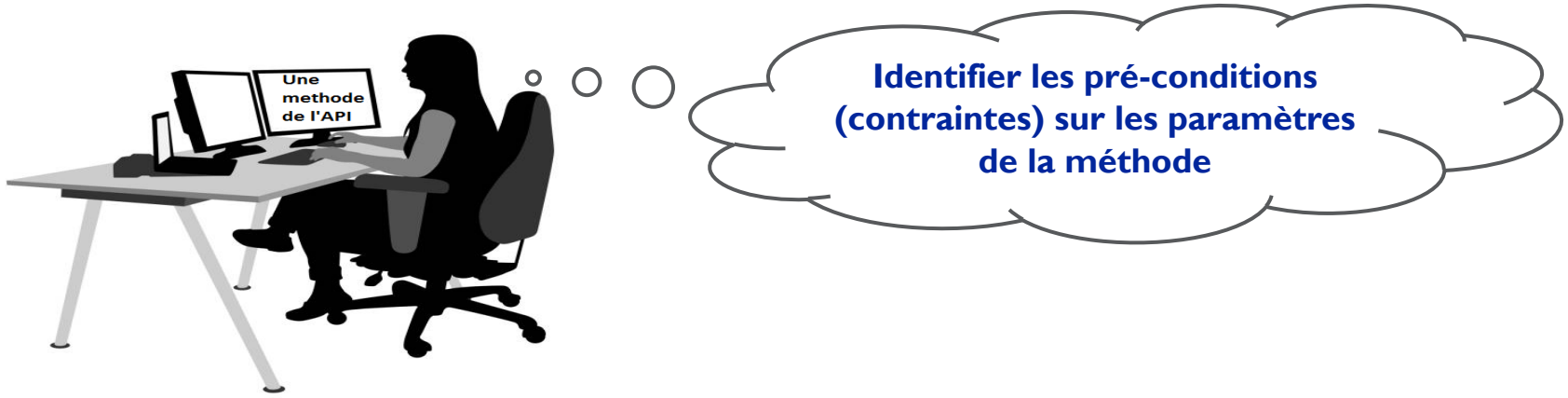


API : plusieurs méthodes d'une API



Plusieurs APIs

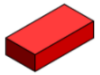
Exemples de contraintes



- "Range limitation"
  - La restriction sur les valeurs de paramètres va au-delà des restrictions définis par les types déclarés

```
public void setTimeToLive(int ttl) throws IOException {
    if (ttl < 0 || ttl > 255) {
        throw new IllegalArgumentException("ttl out of range");
    }
                    . . .
}
```

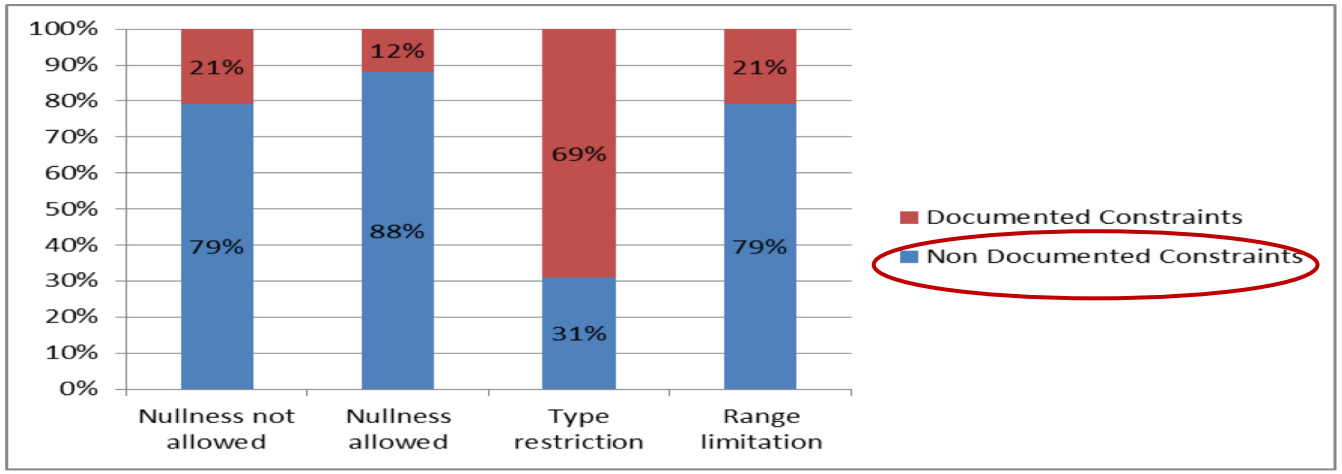
Motivation



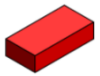
Fréquence dans 24 APIs (incluant JDK)

	Number of Constraints
Nullness not Allowed	3013
Nullness Allowed	778
Type Restriction	116
Range Limitation	401

Documentation de contraintes (excluant JDK)

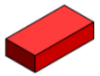


Inférence utilisant des programmes clients

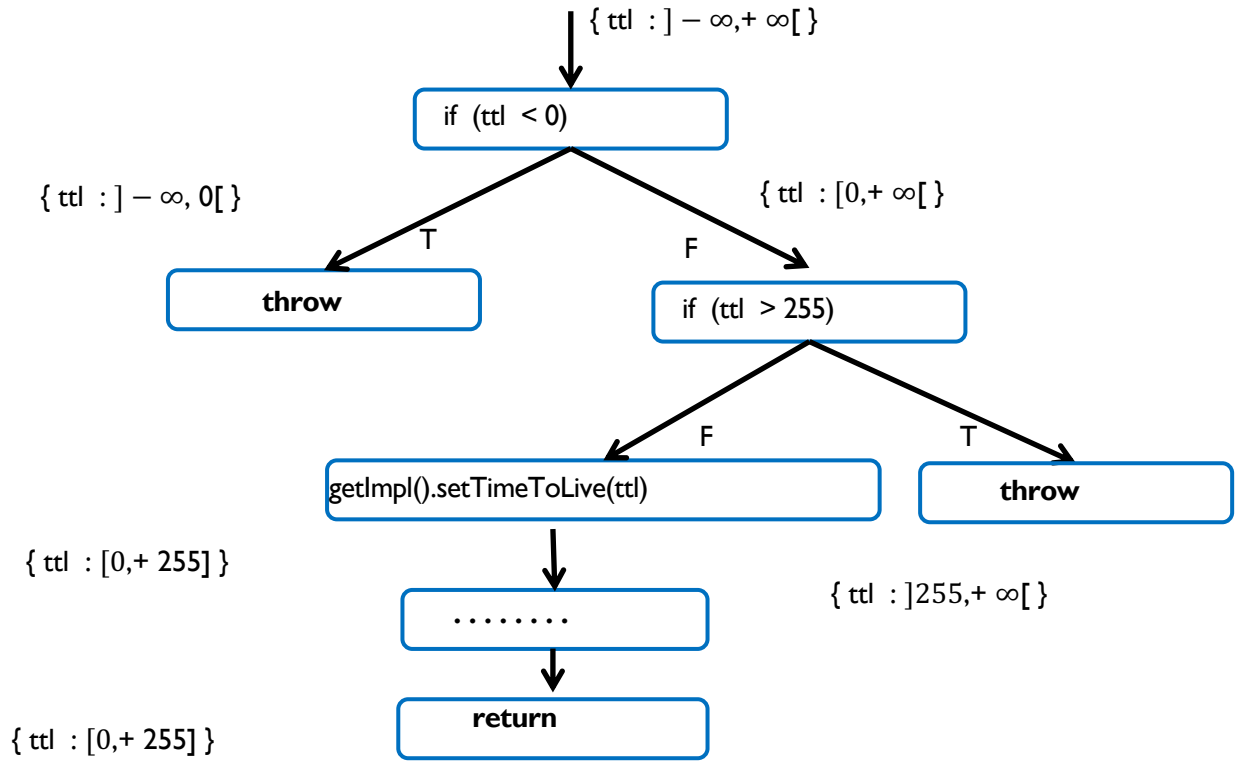


```
public clientMethod(int p)
    {    ...
             if (p >= 0 && p <= 100)
                 apiObj.apiMethod(p);
         ...
    }
```

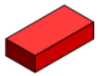
Inférence utilisant le code de la librairie



- Analyse intraprocédurale



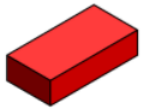
Inférence utilisant le code de la librairie



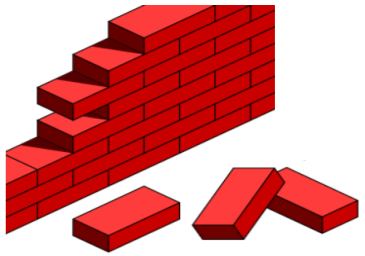
- Evaluation (intraprocedurale)
  - 13 APIs

	Précision
Range Limitation	100%
Nullness Allowed	96%
Nullness not Allowed	97%
Type Restriction	100%

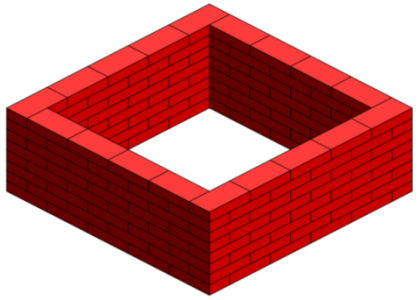
	Rappel
Range Limitation	85%
Nullness allowed	78%
Nullness not allowed	84%
Type restriction	94%



Méthode



API : plusieurs méthodes d'une API



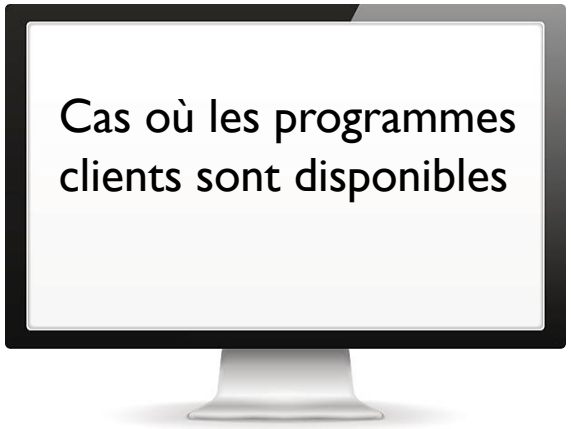
Plusieurs APIs

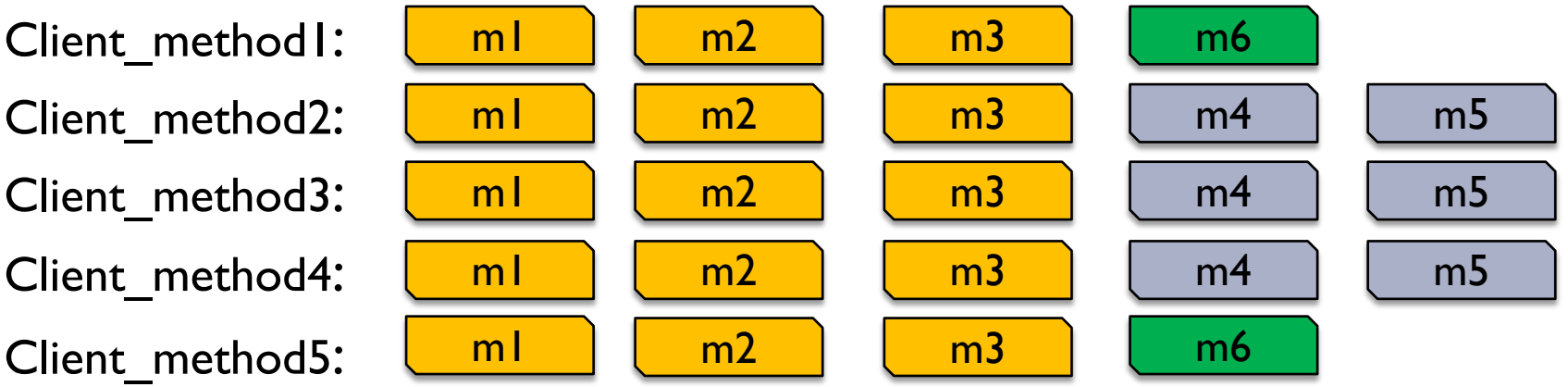
Inférence de patrons

- Patron d'utilisation
  - Un sous-ensemble (ordonné ou non) de méthodes pouvant être utilisées conjointement par les programmes clients

- Problème de clustering
  - Distance de co-utilisation entre les méthodes de l'API
  - Les données
    - Code client vs librairie

Inférence de patrons non ordonnés



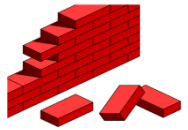


- Distance de co-utilisation des méthodes
  - Proportion de méthodes client partagées

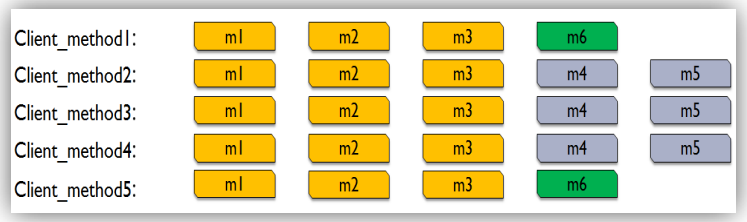
Inférence de patrons non ordonnés

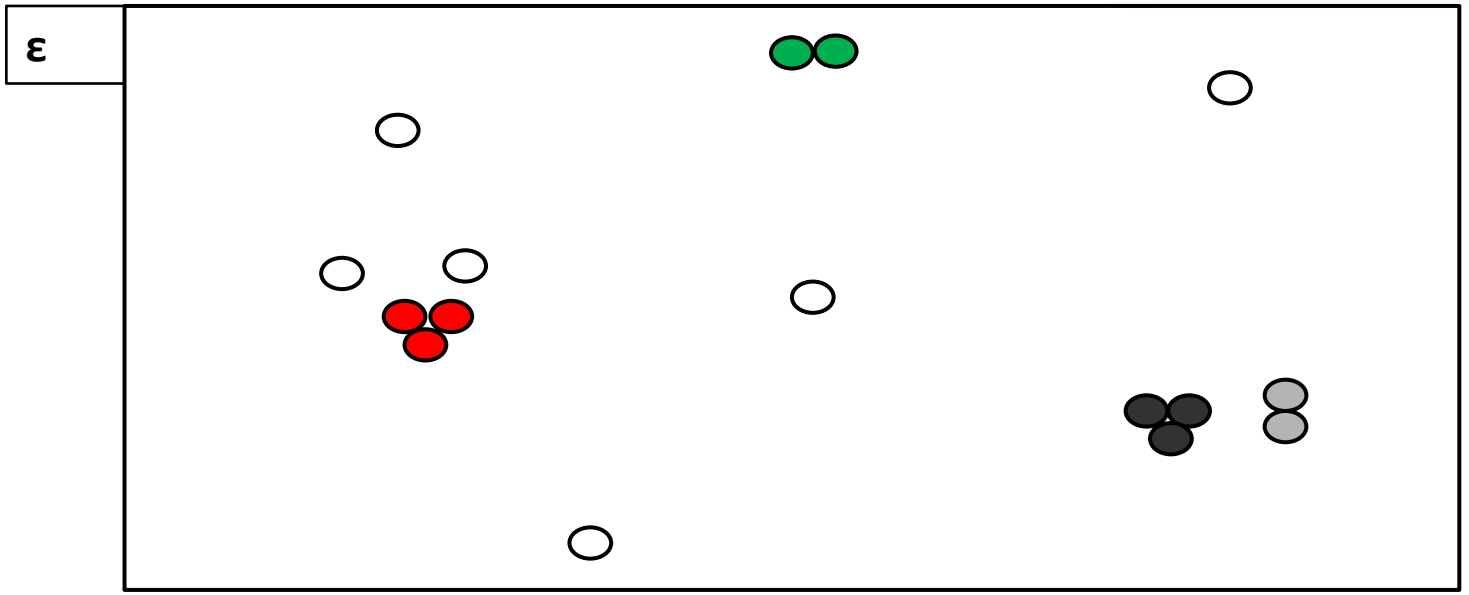
- Estimer la distance de co-utilisation entre les méthodes
  - Proximité structurelle: proportion d'objets de la librairie partagés entre les méthodes
  - Proximité sémantique: similarité cosinus (avec LSI) entre les vocabulaires des méthodes

Clustering des méthodes

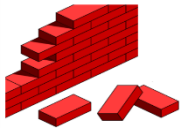


- Regroupement basé sur les distances/estimations de co-utlisation
- Seuil sur la distance



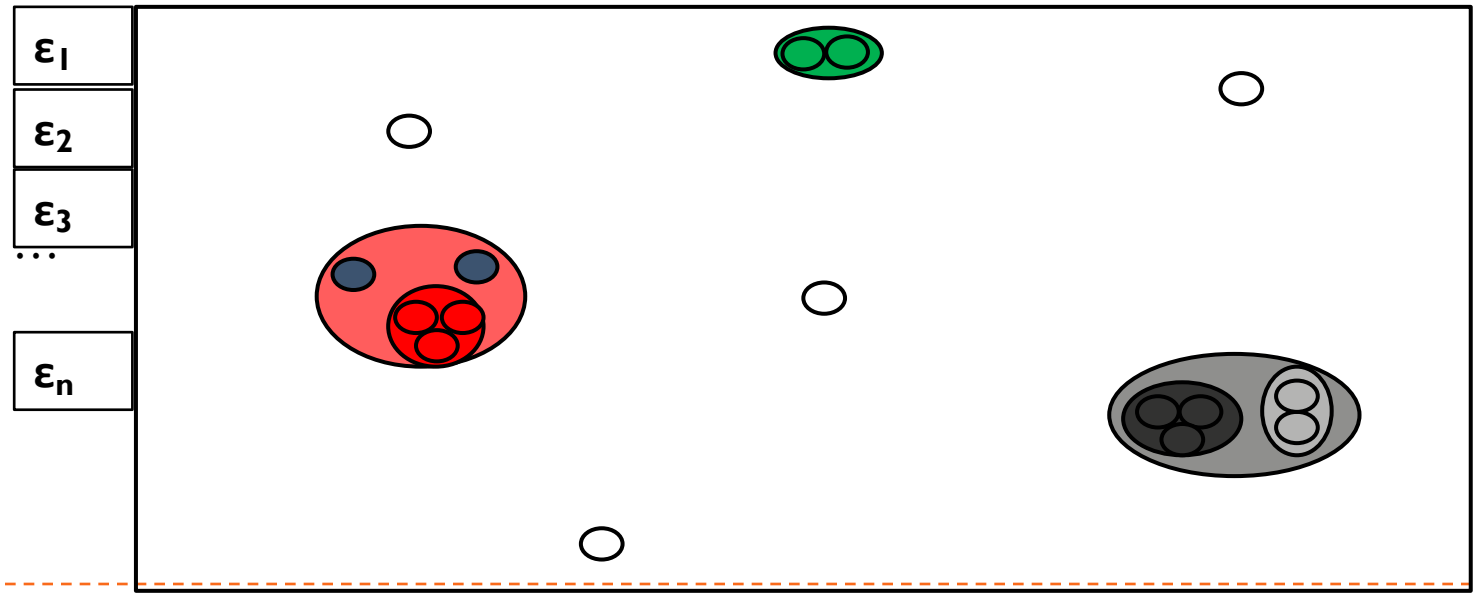


Clustering hiérarchique des méthodes

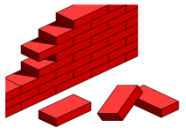


- Grouper les méthodes en plusieurs niveaux en fonction de leurs distances / estimations de co-utilisation

- Des patrons à différents niveaux de forces de co-utilisation



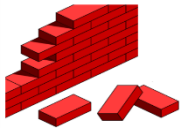
Inférence de patrons complexes



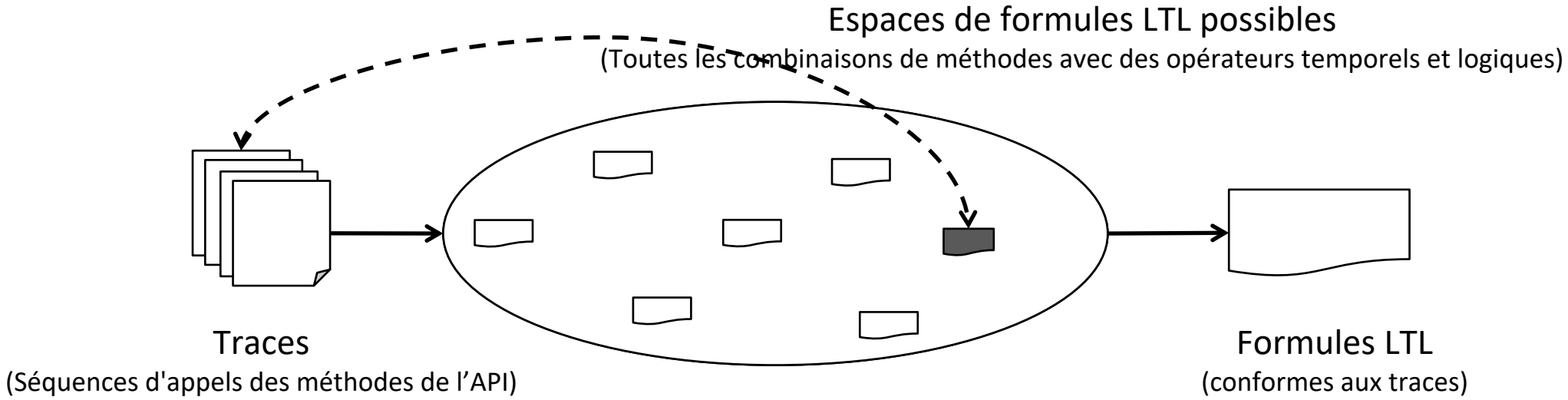
- Plus que de simples groupes de méthodes
- Exemple
  - Lorsque la méthode m1 est appelée, la méthode m2 est appelée juste après et la méthode m3 n'est jamais appelée plus tard, ou m2 n'est pas appelée juste après et m3 sera appelé
- Peut être exprimé en LTL (Linear Temporal Logic)

$$G(c \rightarrow XG(!\,b)) \oplus G(c \rightarrow X!\,a)$$

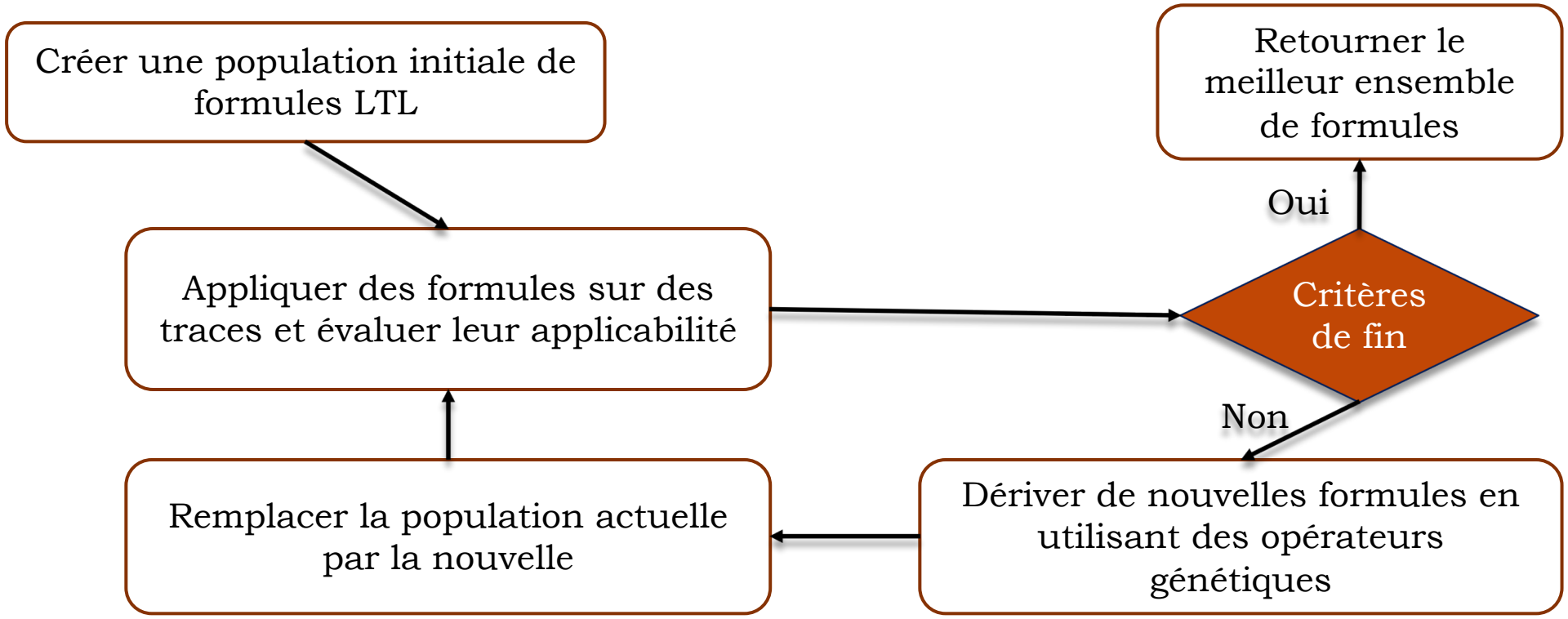
Inférence de patrons complexes



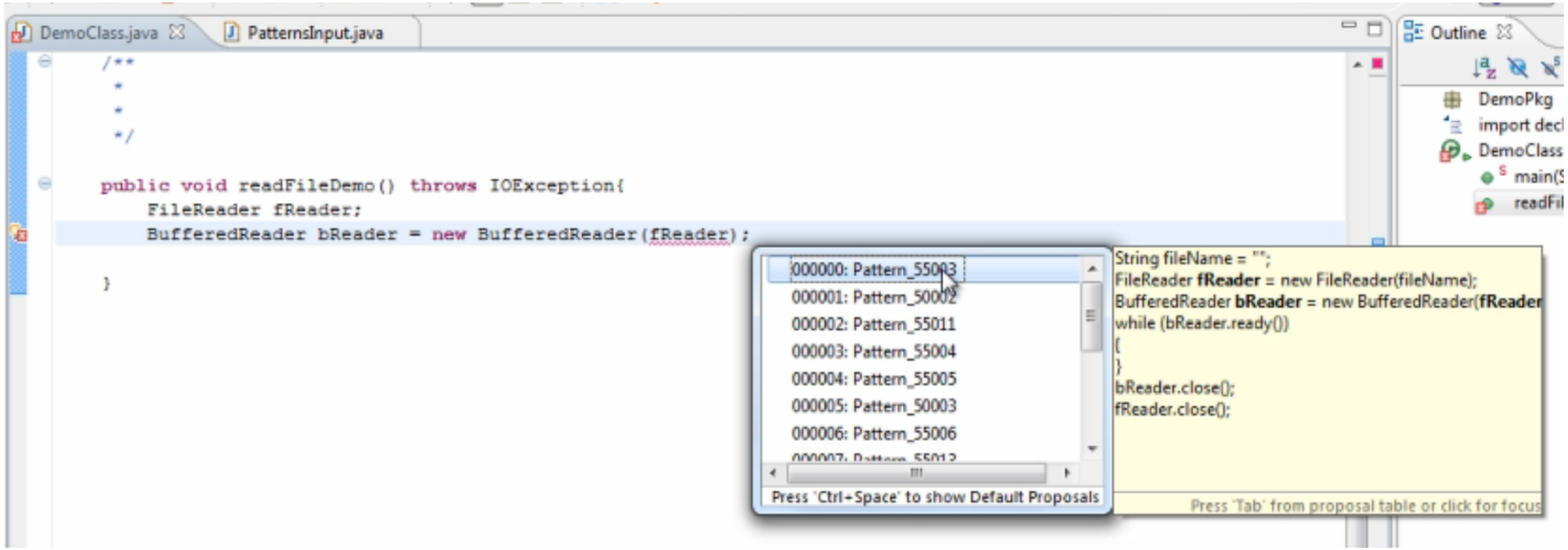
- Apprendre les formules LTL
- A partir des traces d'exécution
- Utilisation de la programmation génétique

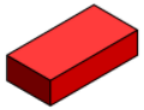


Inférence de patrons complexes

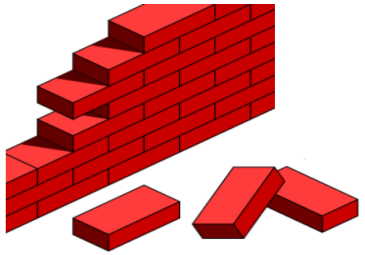


Assister les développeurs dans l'environnement de développement

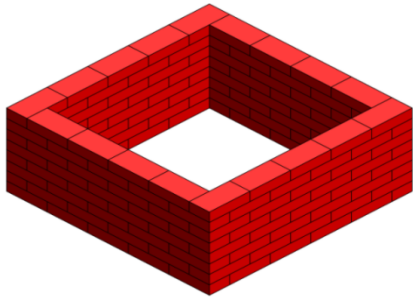




Méthode



API : plusieurs méthodes d'une API

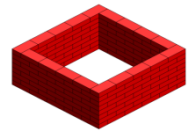


Plusieurs APIs

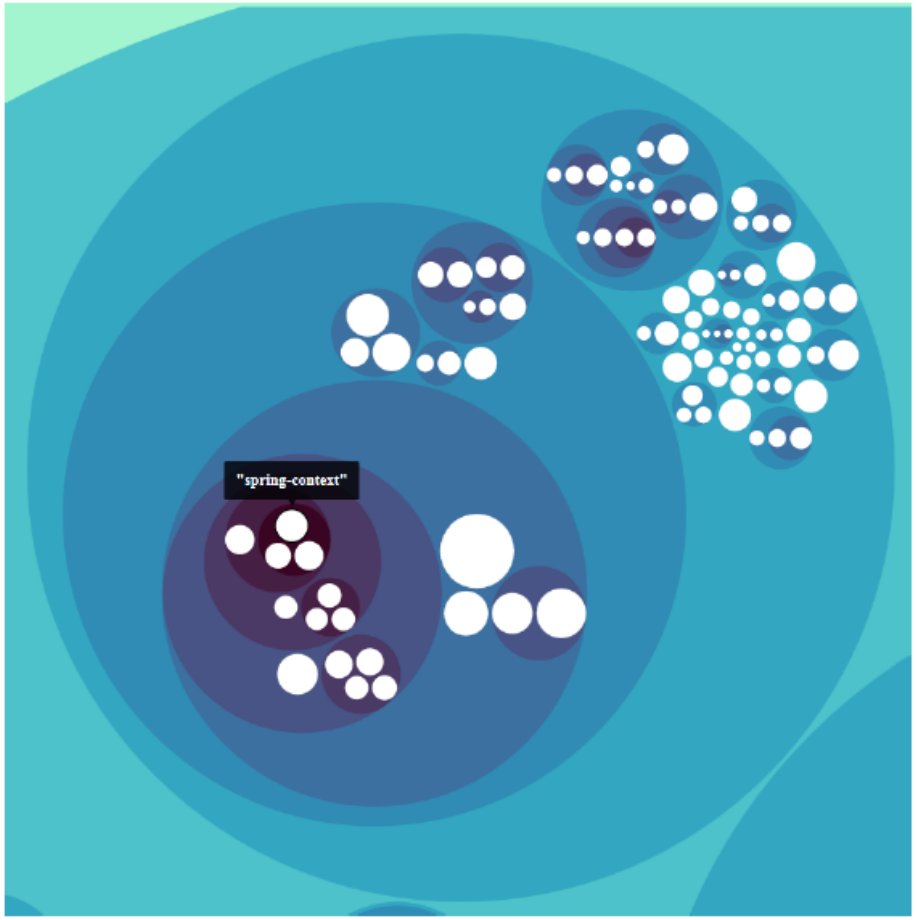
Patrons d'utilisation de librairies

- Intuition
  - Les librairies fréquemment utilisées ensembles ont des fonctionnalités complémentaires et sont faciles à intégrer

Patrons d'utilisation de librairies



- Inférence des patrons
  - Problème de clustering
  - Distance de co-utilisation
    - Proportion de clients partagés
  - Patrons = groupes en multi-niveaux



Les APIs du développement au déploiement

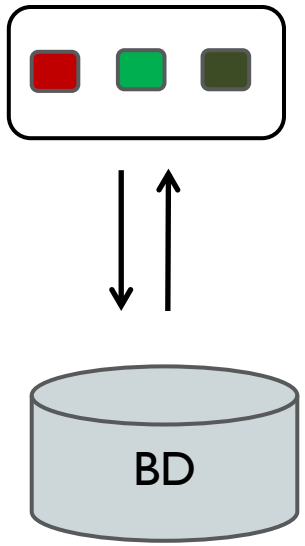
- les APIs sont importantes pour l'environnement de déploiement
  - Environnements infonuagiques (Cloud)
  - Infrastructure virtuelle
    - Créée, modifiée et supprimée par des APIs
  - les applications
    - Configurées et déployées par des APIs

- Appliquer l'inférence de patrons sur des APIs de déploiement d'applications
  - Inférence de patrons/schémas de déploiement

Application basée sur les Microservices (Cloud-Native)

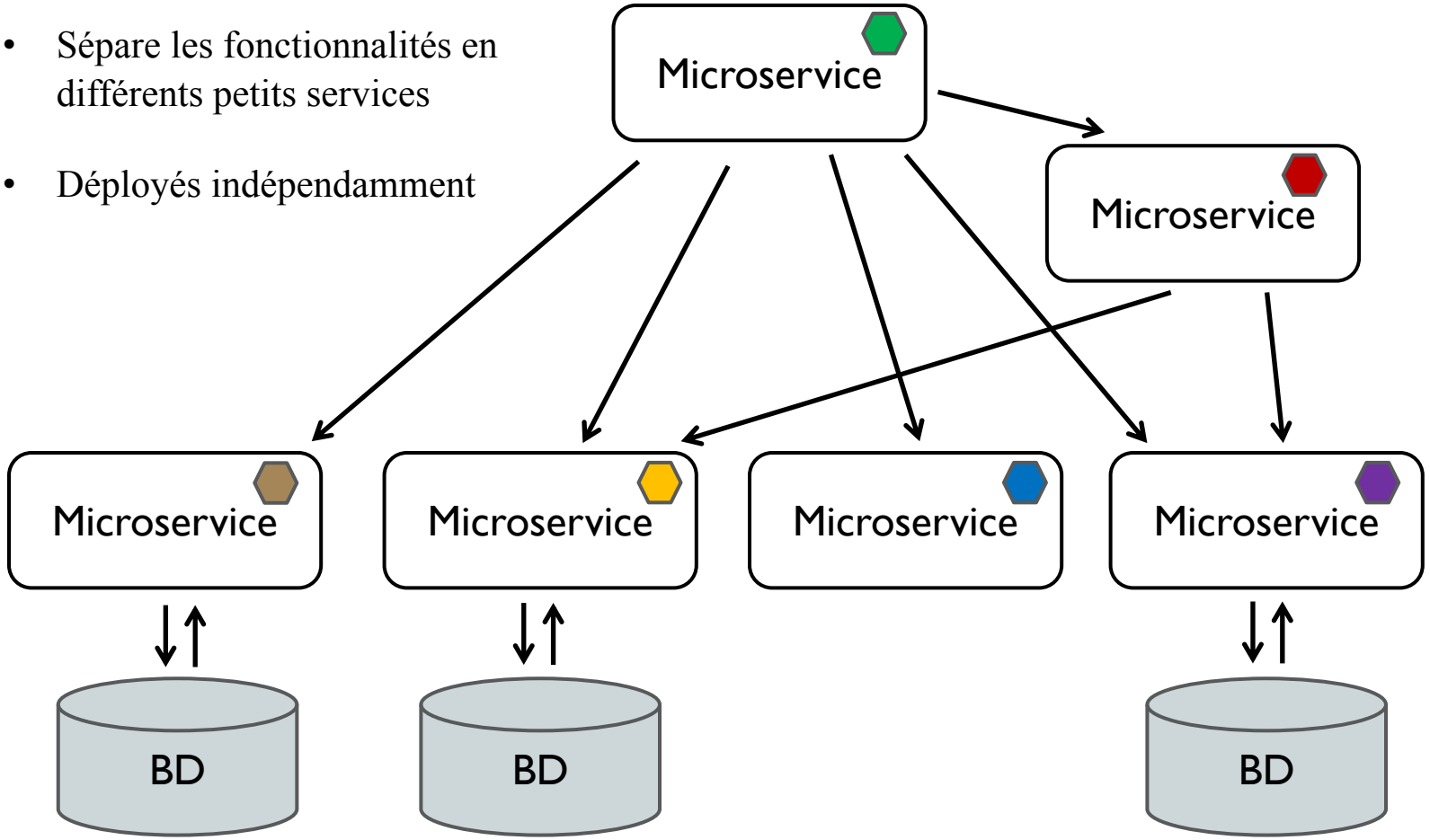
Application Monolithique

- Un seul bloc intégrant toutes les fonctionnalités
- Déployées et exécutées en un seul processus



Application Microservices

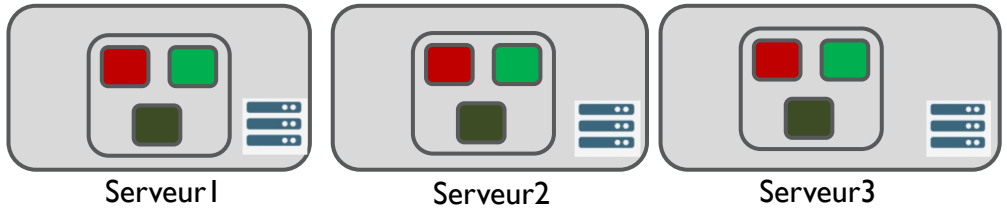
- Sépare les fonctionnalités en différents petits services
- Déployés indépendamment



Application basée sur les Microservices (Cloud-Native)

Application Monolithique

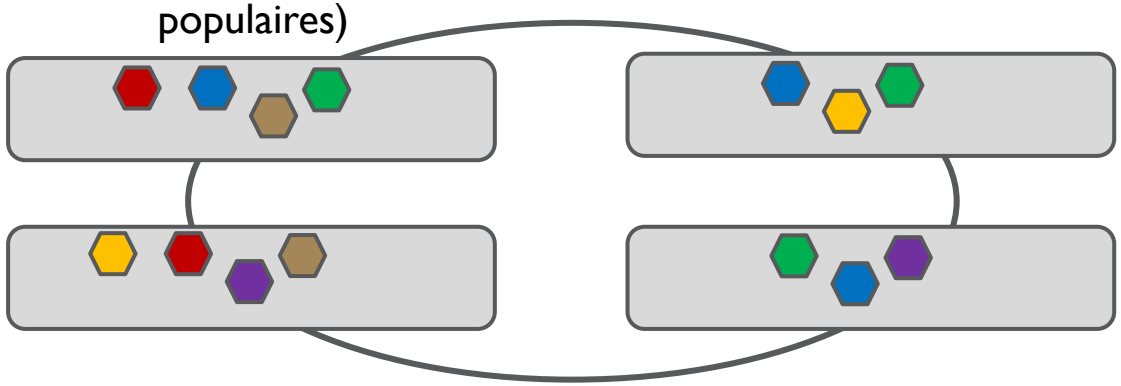
- Passage à l'échelle
  - Variation du trafic
  - Cloner toute l'application dans plusieurs serveurs



Pas suffisamment flexible

Application Microservices

- Passage à l'échelle
  - Individuellement (plus d'instances pour les services populaires)



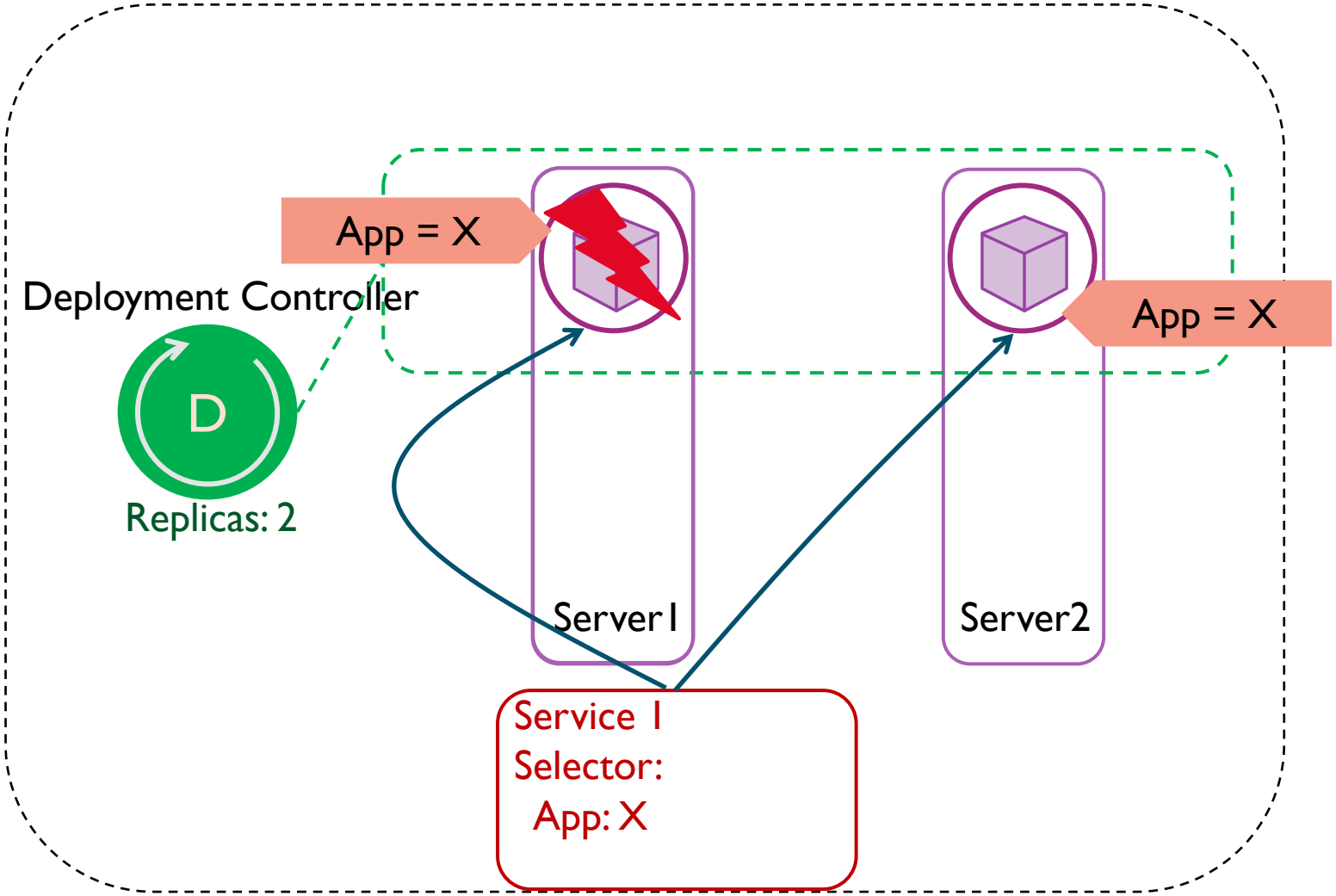
Plus de flexibilité (Cloud-Native)

Kubernetes et les microservices

kubernetes

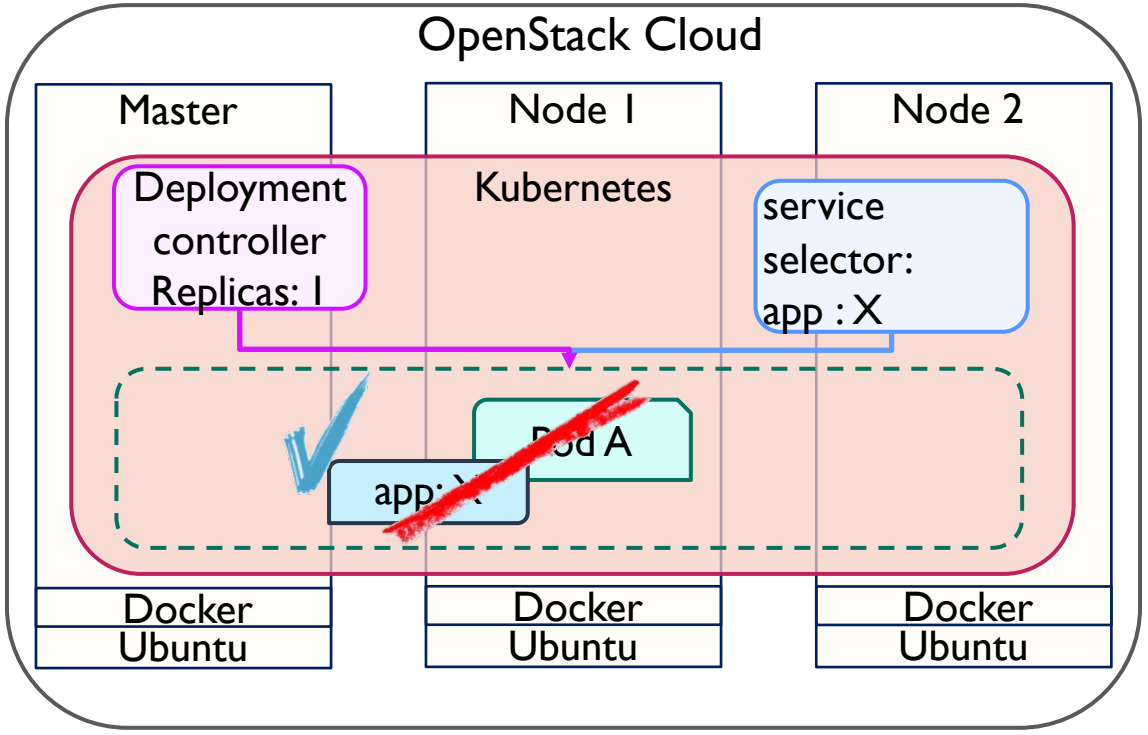
- Une plateforme pour l'orchestration de microservices
- Offre une API qui permet d'automatiser
  - ✓ le déploiement
  - ✓ la mise à l'échelle
  - ✓ la gestion du cycle de vie
- Nous avons appliqué l'inférence de patrons à l'API de Kubernetes

Microservices (stateless deployment pattern)

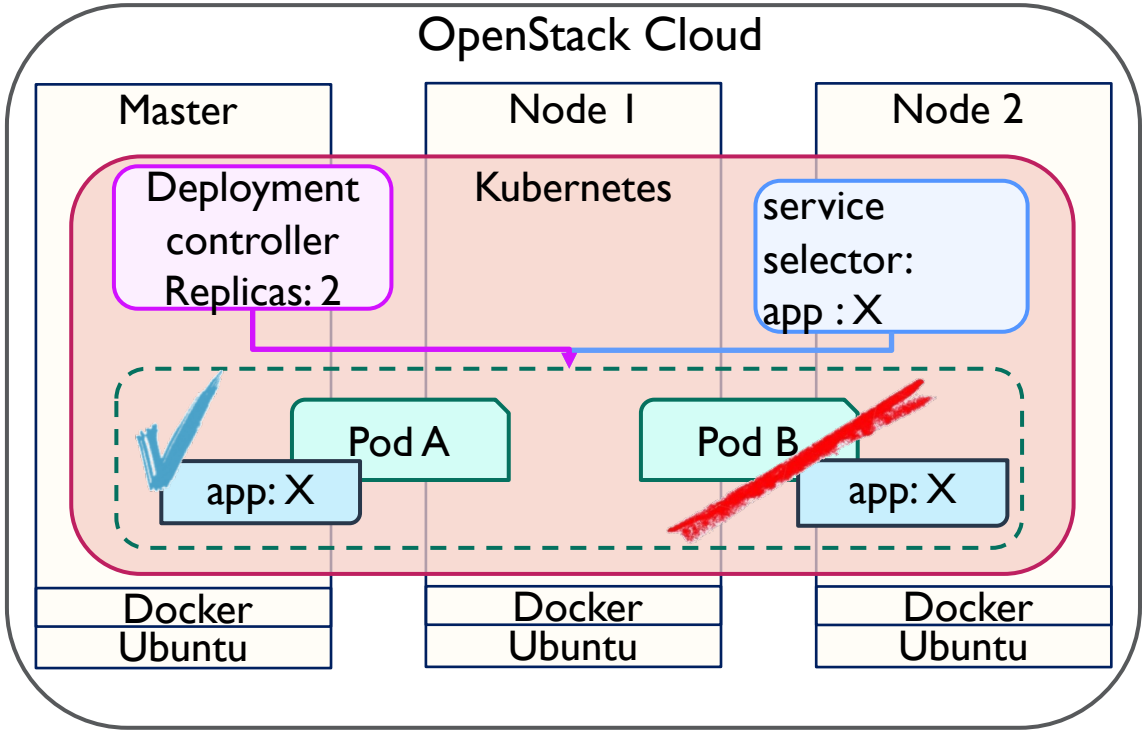


Disponibilité des microservices stateless avec Kubernetes

- Aptitude aux systèmes hautement disponibles
  - ~5 minutes d'indisponibilité par année
- Nous avons évalué le taux d'interruption des services pendant les pannes pour les patrons identifiés



No-Redundancy Redundancy Model

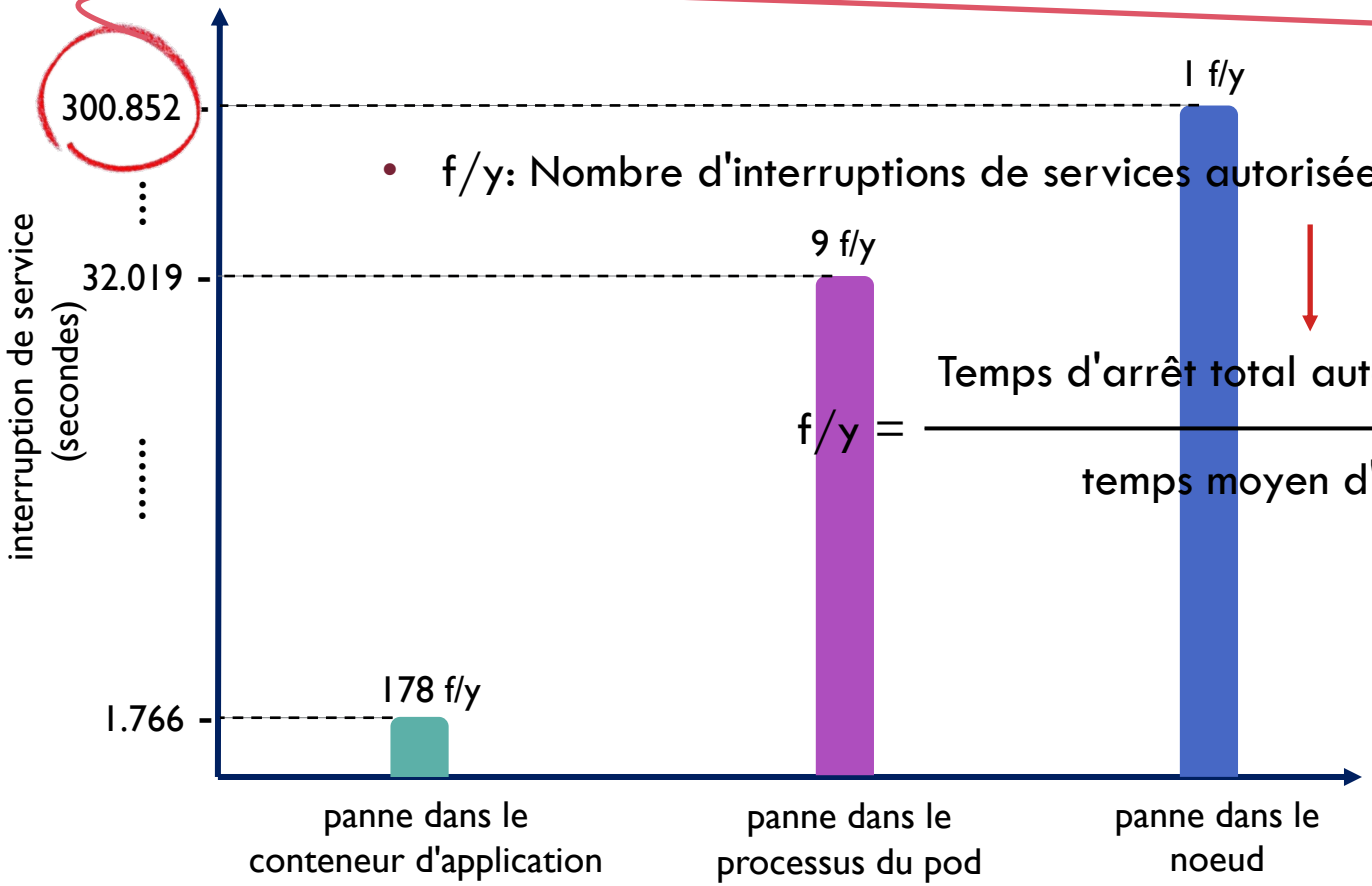


N-Way Active Redundancy Model

Disponibilité des microservices (expériences et mesures)

Kubernetes (configuration par défaut) !

- f/y: Nombre d'interruptions de services autorisées par année pour les systèmes HA (hautement disponibles)

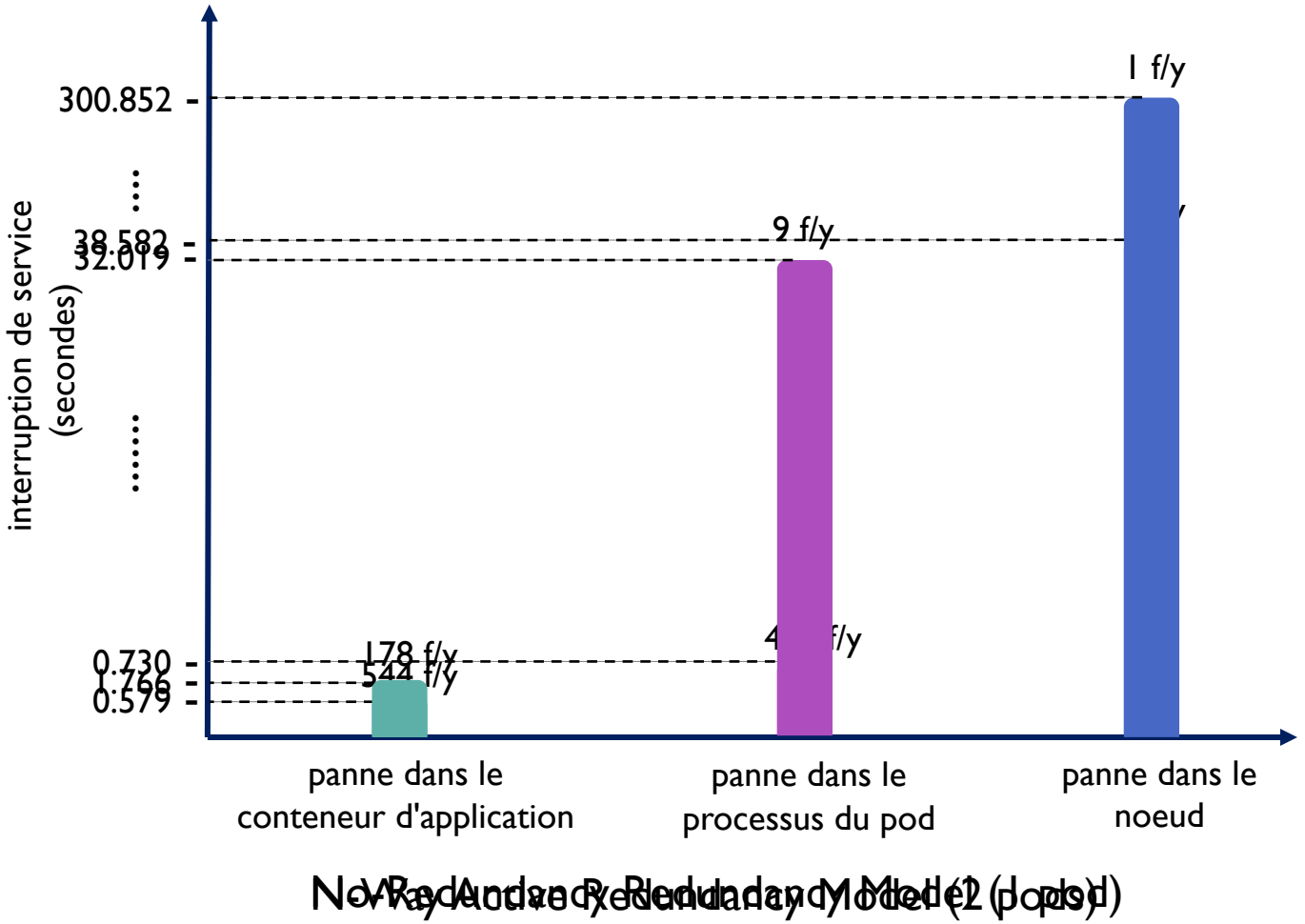


$$f/y = \frac{\text{Temps d'arrêt total autorisé (315,36 secondes)}}{\text{temps moyen d'interruption de service}}$$

No-Redundancy Redundancy Model (1 pod)

f/y: (failures per year) Nombre d'interruptions de services autorisées par année pour les systèmes HA (hautement disponibles)

Disponibilité des microservices (expériences et mesures)



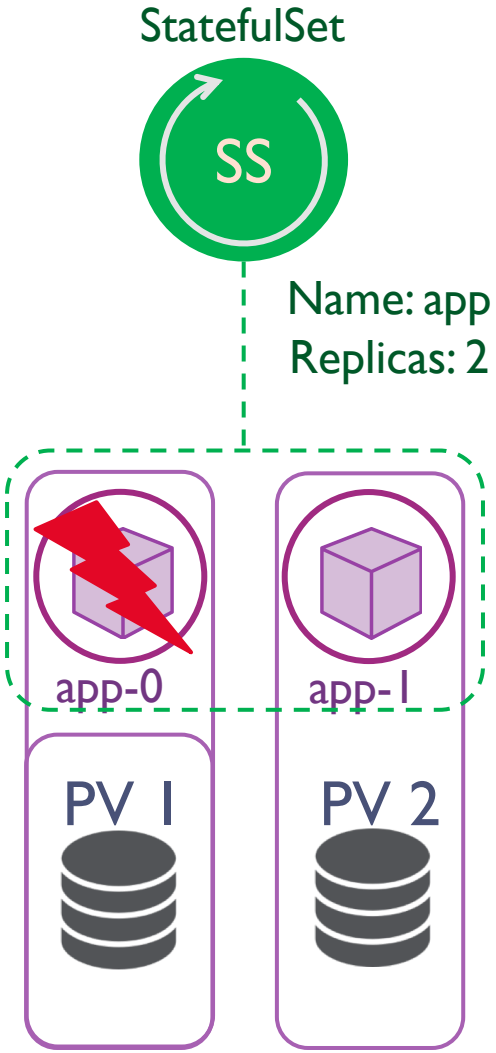
La redondance est un mécanisme important pour améliorer la disponibilité des services

f/y: (failures per year) Nombre d'interruptions de services autorisées par année pour les systèmes HA (hautement disponibles)

Microservices (stateful deployment pattern)

- Les clients recevant le service depuis app-0 connaîtront une interruption de service jusqu'à ce que app-0 soit redémarrée et récupère le dernier état enregistré de PV1.

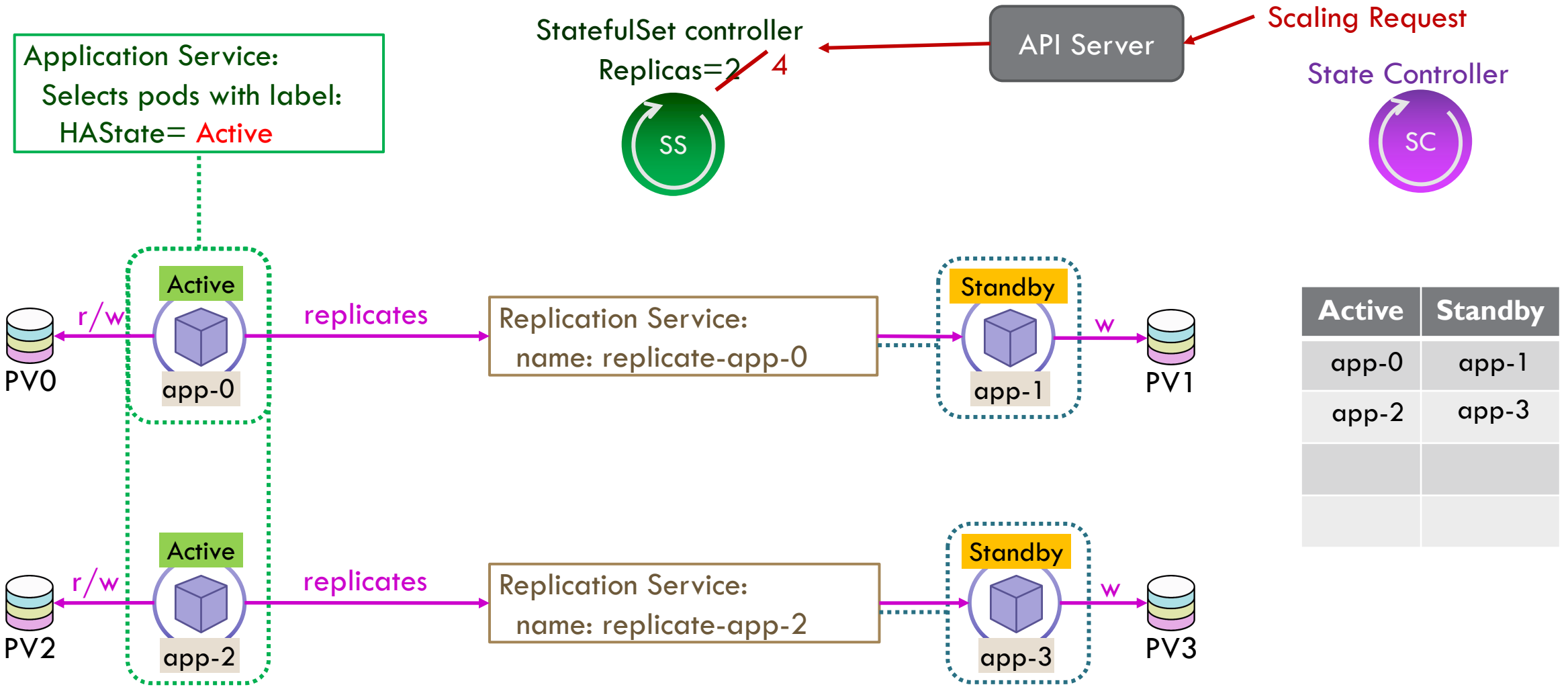
- Les instances de microservice ont des états différents
  - Non interchangeable



Enrichir Kubernetes avec un contrôleur d'état

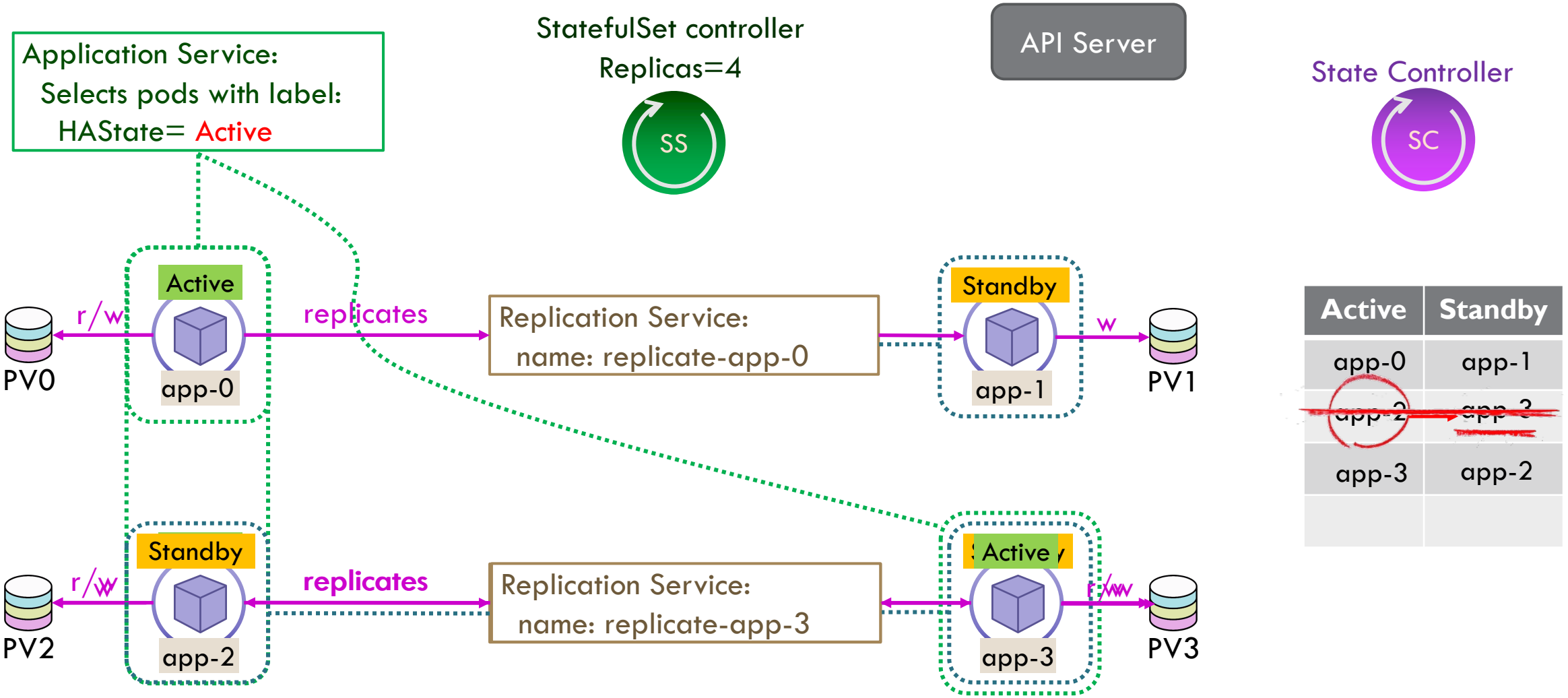
- Un standby pour chaque pod actif
- Répliquer l'état du pod actif dans le pod standby
- Gérer les cas où le nombre de pods change
  - Interruption de service
  - Passage à l'echelle (Scaling Request)
- Avec notre solution
  - Amélioration du taux d'interruption (40% - 98%)
  - Brevet provisoire déposé

Enrichir Kubernetes avec un contrôleur d'état



Active	Standby
app-0	app-1
app-2	app-3

Enrichir Kubernetes avec un contrôleur d'état



Travaux futures

Migration vers le Cloud-Native

- Objectifs à long terme
  - Supporter et automatiser (autant que possible) la migration vers les applications et les architectures cloud-native

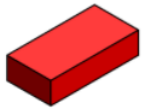
- Objectifs à court terme :
  - Considérer d'autres modèles de redondance plus efficaces
  - Proposer une méthode pour la prédiction et la prévention des interruptions des services
  - Proposer une méthode pour extraire des microservices à partir de la décomposition des monolithes
  - Proposer une méthode de test/ verification pour les systèmes migrés

Travaux futures

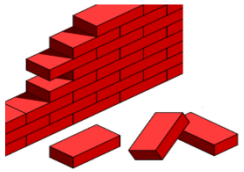
Défis d'utilisabilité des APIs

- Patrons d'API pour d'autres écosystèmes (App mobiles)
  - Tenir compte des permissions d'utilisation d'APIs (aspects de sécurité)
- Évolution des patrons
  - Étudier comment les patrons d'utilisation évoluent dans le temps
- Autres sources de données pour l'inférence de patrons d'API
  - Historique de changement, Q&A site stackoverflow

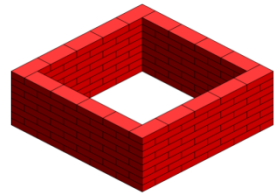
Conclusion



Inférence de contraintes d'utilisation des methodes



Inférence de patrons d'utilisation d'une API



Inference de patrons d'utilisation de plusieurs librairies

- Une vision holistique et pragmatique
  - Tous les niveaux d'utilisabilité sont importants et complémentaires
  - L'approche doit être pragmatique par rapport à la disponibilité des données

Merci

Inférence de contraintes (niveau méthode)

Identifier les pré-conditions (contraintes) sur les paramètre de la méthode

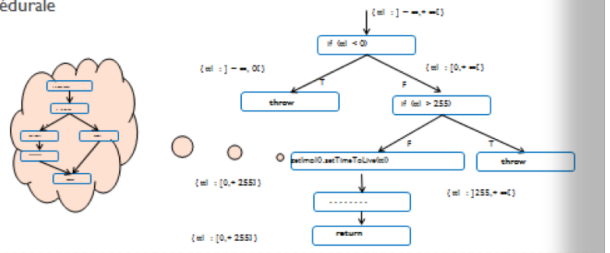
```
public void addChild( XTDetachedNode child,int index ) {
    // ....
    testRecursion( child );
    children.add( index, child );
    child.setParent( this ); }
```

- Exemples de contraintes
  - "Nullness not allowed"
    - Le paramètre null provoque un échec lors de l'exécution
  - "Nullness allowed"
    - La valeur nulle a une sémantique spécifique pour un paramètre

13

Inférence de contraintes (niveau méthode)

- Inférence utilisant le code de la librairie
  - Intraprocédurale
  - Interprocédurale

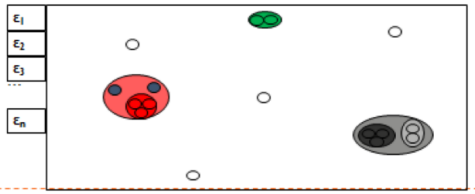


16

(Saied et al. SANER15a)

Inférence de patrons non ordonnés (niveau API)

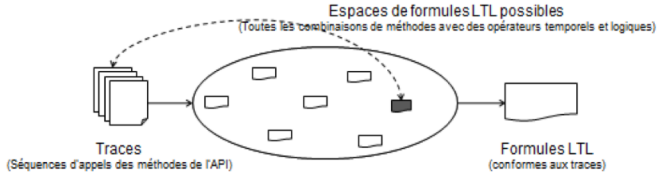
- Clustering hiérarchique
  - Grouper les méthodes en plusieurs niveaux en fonction de leurs distances / estimations de co-utilisation
  - Des patron à différents niveaux de forces de co-utilisation



23

Inférence de patrons complexes (niveau API)

- Apprendre les formules LTL
- A partir des traces d'exécution
- Utilisation de la programmation génétique



29

(Saied et al. TR17, DBS 18)

Patrons d'utilisation de librairies (niveau de plusieurs API)

- Inférence des patrons
  - Problème de clustering
  - Distance de co-utilisation
    - Proportion de clients partagés
  - Patrons = groupes en multi-niveaux

35

Microservices (stateful deployment pattern)

- Les clients recevant le service depuis app-0 connaîtront une interruption de service jusqu'à ce que app-0 soit redémarrée et récupère le dernier état enregistré de PV1.