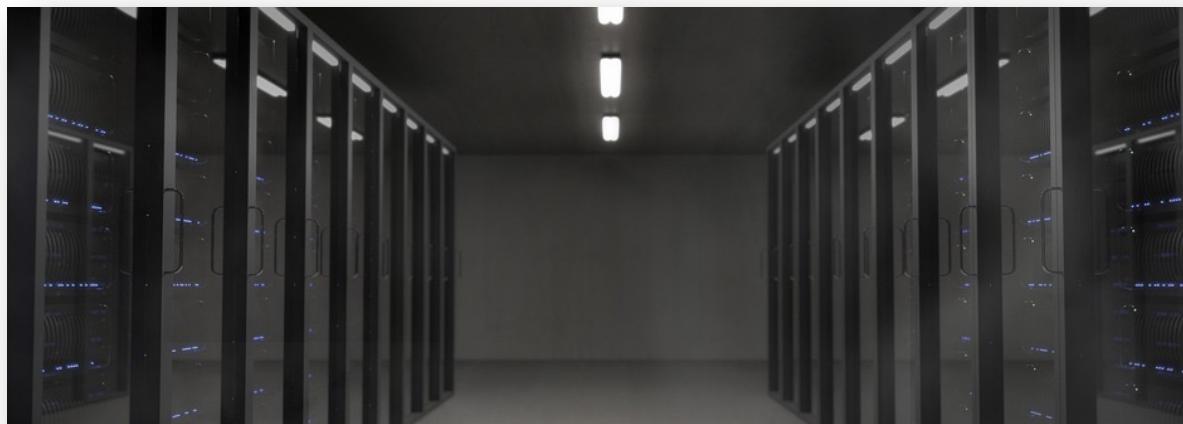


Méthodes Aléatoires

<https://codeshare.io/kmRJ1L>

LOG3430 Méthodes de test et de validation du logiciel

Certains tests ne sont pas basés sur une théorie rigide, mais plutôt sur des preuves empiriques.



Il sont plutôt axé
sur les données

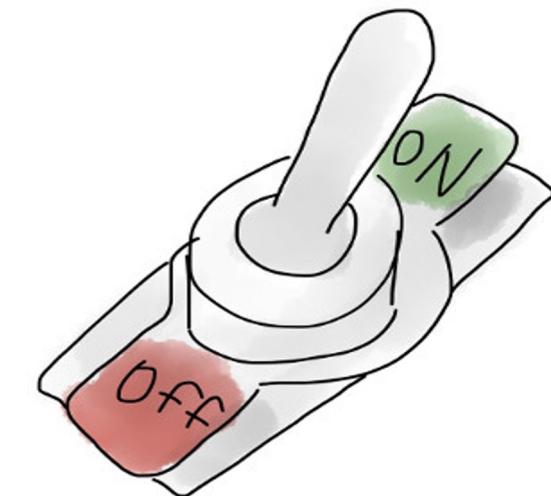
Il existe de nombreuses façons d'utiliser les données pour effectuer des tests



A/B Testing



Fuzzing



Feature Toggles

Fuzzing



Tests: ex: Unit Tests--> on utilise le code source pour voir si nos tests vont passer ou non

Fuzzer: On modifie les valeurs d'entrée et on n'a pas nécessairement accès aux Unit Tests. On utilise un Fuzzer pour voir si un programme va faire un crash ou non. Aucun test spécifique, juste vérifier si mon programme plante

Un fuzzer peut être classé comme suit

programme qui crée des entrées aléatoires

Note: Si j'ai accès au code source mais qu'on ne l'utilise pas, on reste en boîte noire!!!

- Basé sur la génération : les entrées sont générées à partir de rien
- Basé sur la mutation : en modifiant les entrées existantes
- Dumb ou smart selon qu'il utilise ou connaît la structure d'entrée
- Boîte blanche, grise, ou noire, selon qu'il connaît ou non la structure du programme.

Fuzzing de boîte noire : Traite le programme comme une boîte noire et ignore la structure interne du programme. Par exemple, un outil de test aléatoire qui génère des entrées au hasard est considéré comme un fuzzer de boîte noire. Si absolument aucune intelligence et aucun accès au code source

Fuzzing de boîte blanche : S'appuie sur l'analyse du programme pour augmenter systématiquement la couverture du code ou pour atteindre certains emplacements critiques du programme. Par exemple, on peut exécuter le programme (en commentant). On est guidés par le code source

par quelques entrées aléatoires), rassembler des contraintes sur les entrées `à des instructions conditionnelles, utiliser un solveur de contraintes pour générer de nouvelles entrées de test.

Fuzzing de boîte grise : Teste le programme avec une connaissance partielle de son fonctionnement interne. Par exemple, un fuzzer de boîte grise pourrait tirer parti des retours de couverture d'autres instrumentations ou bibliothèques pour apprendre à approfondir le programme. Si une entrée générée augmente la couverture, elle sera apprise par le fuzzer pour améliorer davantage le fuzzing.

Objectif des Fuzzers

1. Trouvez de vrais bugs
2. Réduire le nombre de faux positifs
 - a. Générer des entrées raisonnable car on va essayer de réduire le nb de Faux Positifs. Ex à chaque fois que je teste une valeur, si je ne trouve pas de bogues/défauts c'est du temps perdu, temps de CPU et serveur perdu
 - b. Si nous attendons une chaîne, passer un fichier (File object) sera rejeté avant même qu'il n'arrive dans notre code

Entrée générée

1. Générer une entrée complètement aléatoire **boite noire**
 - a) Ne contrôle pas nécessairement le type d'entrée **on génère n'importe quoi**
 - b) "Lait, 3,99" -> 9620
int générés de façon aleatoire
2. Comprendre le type d'entrée **boite noire**
 - a) « Lait, 3,99 » -> (est une chaîne) -> « %&\$# » **donne une erreur**
3. Comprendre la structure d'entrée **on sait que c'est une chaîne, mais en quoi elle ressemble**
 - a) « Lait, 3,99 » -> « \w+, \d\.\d\d » -> « HFSDMEX, 8,43 »
mots virgule espace 3 chiffres
4. Approches formelles **utilie pr un programme structuré qui ne change pas souvent**
 - a) Fuzz basé sur des modèles, grammaires et protocoles
 - b) Utile lorsque le problème est bien structuré
 - c) Souvent peu pratique pour les grands programmes du monde réel

Fuzz avec mutation

1. Prendre l'entrée existante
2. Modifiez-la au hasard
3. Passez-la au programme

La plupart des fuzzers en boîte noire et grise mais il y a en boîte blanche aussi

Exemples

1. Un ensemble de fichiers image qui seront mutés de manière aléatoire
2. Un ensemble d'entrées pré-enregistrées qui seront modifiées de manière aléatoire
 - a) "Lait, 3,99" -> "Zait, 2,99"

Problèmes

- Un nombre écrasant de faux positifs
 - Les faux positifs sont très coûteux car ils nécessitent un effort manuel
 - On utilise n'importe quoi, à quoi s'attendre ?
- Focus sur la couverture du code
 - Surtout les approches de la méthode formelle
 - La couverture est moins importante que des apports raisonnables
- Nettoyage/Cleaning
 - Sanitizer : rendre la saisie aléatoire plus raisonnable éliminer le + possible de générations inutiles
 - Minimisation : éliminez les échecs de test redondants avec des “diff”
 - Triage : trouver des sorties/stackdumps similaires et les regrouper dans le même rapport de bogue

Résumé de fuzzing

- Test avec une entrée aléatoire raisonnable
 - Objectif : trouver de vrais bugs
 - Problème : la plupart des échecs sont des faux positifs qui coûtent cher
- Utilisé dans la pratique
 - Google, Microsoft, Apple, etc. l'utilisent surtout dans des environnements bien spécifiés/contrôlés
- Faut-il l'utiliser ?
 - Au niveau de base, il est simple d'ajouter un Fuzzer
 - Exemple : au lieu de toujours tester le même entier, testez un entier aléatoire dans une plage
- Pourquoi générer une entrée aléatoire si vous avez une entrée réelle enregistrée ?
 - Utiliser l'entrée enregistrée qui a provoqué des échecs sur le terrain
 - Transformez cela en cas de test

Chaos Monkey

La théorie (à travers une métaphore)

Avez-vous une voiture?

- Votre pneu de secours est gonflé ?
- Avez-vous des outils dans le coffre?
- Savez-vous comment changer un pneu?
- La solution
 - Demandez à un singe de crever votre pneu une fois par semaine
 - Réparer le problème
 - Ainsi, lorsque vous aurez un pneu crevé sur l'autoroute, vous serez prêt!

Les logiciels sont artificiels

Le chaos coûte cher dans l'environnement physique réel,

Mais

peut être (presque) gratuit et automatisé dans le cloud."

Éteindre un serveur et voir comment les autres serveurs vont réagir et garder le service stable, rembarquer et rerouter le traffic et les utilisateurs ne devraient même pas se rendre compte qu'il y a un problème

Chaos Monkey

- Désactive de manière aléatoire les instances de production pour s'assurer que le système est suffisamment robuste pour survivre à ce type de panne sans aucun impact sur le client.
 - 1) On teste notre système et on vérifie s'il répond comme on veut
 - 2) On ne fait pas de façon complètement aléatoire, il est vrai qu'on désactive un serveur aléatoirement mais on ne le fait pas à minuit par ex, on le fait quand que les ingénieurs de Netflix sont dispos et prêts à répondre au cas où il y a qdc qui arriverait et qu'il n'y a pas bcp d'heure de pointe avec bcp d'utilisateurs (10h a.m)

Le chaos est ajouté en production

“By running Chaos Monkey in the **middle of a business day**, in a **carefully monitored environment** with **engineers standing by** to address any problems, we can still learn the lessons about the weaknesses of our system, and build **automatic recovery mechanisms** to deal with them. **So next time an instance fails at 3 am on a Sunday, we won’t even notice.**”

Types de Chaos

on fait ces tests pendant que les ingénieurs sont là

Chaos Monkey:

- Arrête des instances de production aléatoires

Latency Monkey:

- induit des retards artificiels dans la couche de communication

rajouter des délais entre les utilisateurs et la réponse du système pour voir si le système va être capable de savoir s'il y a un délai sur une instance spécifique et rerouter l'utilisateur vers une autre instance

Janitor Monkey:

- trouve les ressources inutilisées et s'en débarrasse

voir que quand on se débarrasse des ressources inutilisées cela cause un problème

Security Monkey: voir si les certificats de sécurité sont à jour, pour la localisation du serveur également ex: si on est en Europe est-ce que le certificat est valide

- trouve des violations de sécurité ou des vulnérabilités et termine leurs instances

10–18 Localization Monkey: vérifier que la langue est correcte pr l'environnement ex: Netflix au Canada et fr et en Europe eng, mais pas des mots en espagnol

- utilise différentes langues et jeux de caractères selon la région géographique

Chaos Gorilla: ex: On simule une panne de tout le Qc, et on va voir si on va se rerouter vers les serveurs d'Ontario

- simule une panne de toute une zone de disponibilité

Sommaire

- Le chaos est ajouté en production au milieu de la journée de travail avec un suivi attentif et des ingénieurs prêts à résoudre les problèmes
 - pour que:
- les faiblesses soient identifiées
- des mécanismes de récupération automatique soient construits
 - pour que
- une panne de 3 heures du matin ne nécessite pas d'interventions manuelles

A/B Testing



Exemple: Campagne par e-mail

"L'offre se termine ce samedi ! Utilisez le code A1"

Vs

courriel envoyé à la moitié des gens en classe

"L'offre se termine bientôt ! Utilisez le code B1"

envoyé à la moitié des gens en classe

Pk différence de réponse entre code A1 et code B1?

- Une des deux versions est différente ce qui force les gens à utiliser un plus que l'autre
- Possibilité qu'on n'a pas envoyé le courriel à des groupes différents. Ex: Pour un courriel de promotion de souliers de course, on a envoyé le courriel A1 à des personnes qui font le marathon et B1 à des personnes aléatoires. On doit avoir une population de gens le + possible aléatoire si on veut comparer les résultats de façon utile.



Project name Home About Contact Dropdown - Default Static top Fixed top

Project name Home About Contact Dropdown - Default Static top Fixed top

Welcome to our website

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Learn more

Welcome to our website

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

→ Learn more

Click rate:

52 %

72 %

Maxime Lorant –

https://commons.wikimedia.org/wiki/File:A-B_testing_simple_example.png,
CC BY-SA 4.0, <https://commons.wikimedia.org/w/index.php?curid=52602931>

Differentes versions, logging et analyse

- Versions et essais sur le terrain
 - Certains utilisateurs voient la version A ex: Facebook
Ils vont montrer deux versions à des gens et voir celle qui est la plus populaire
 - Certains utilisateurs voient la version B
- Logging
 - Mesurer les clics/conversions/ventes
- Analytique
 - Comparer statistiquement A et B pour déterminer s'il y a une différence significative

Test d'hypothèse statistique

hypothèse nulle = "pas de différence entre A et B" But: prouver qu'il y a une différence entre A et B

Exécuter un test statistique

- T-Test suppose une distribution normale
- Test de Wilcoxon : sommes de rang (pas d'hypothèse de normalité)

Rejeter l'hypothèse nulle avec une confiance particulière, par exemple 95 %,
 $p \leq 0,05$ On veut qu'il y ait une différence entre A et B, si pas de différence, on garde la version A qui est la version originale

Direction

- $A > B$ ou $B > A$

Magnitude de la différence (taille d'effet)

- Par exemple, dans quelle mesure B est-il meilleur que A ?

Essais randomisés

- Utilisez un générateur de nombres aléatoires pour déterminer quels utilisateurs voient chaque fonctionnalité
 - ex: 1000 utilisateurs on leur offre un bouton différent et de manière aléatoire on voit si les gens préfèrent une version plutôt qu'une autre
 - Si ce n'est pas vrai dans la méthode aléatoire, vous avez un biais systématique
- Vous n'êtes pas obligé d'envoyer le même nombre d'utilisateurs pour utiliser un test statistique
- Vous aurez besoin d'un échantillon statistiquement significatif
 - par exemple, au moins 100 dans chaque condition
 - Minimum 30 personnes! en bas de 30 c'est très difficile de faire des tests statistiques
 - Cela prend 400 personnes choisies aléatoires pour un bon test
- Si vous testez plusieurs hypothèses, vous voudrez peut-être faire une correction
 - par exemple, correction de Bonferroni

Nous pouvons tester les fonctionnalités progressivement

Commencez avec 1 % des utilisateurs voyant une nouvelle fonctionnalité
ex: Netflix offre l'option de choix aléatoire d'un film/série et tester pour 100 personnes. S'ils l'utilisent, on augmente le nb de personnes, jusqu'à ce que l'option B remplace l'option A.

```
int num = rand.nextInt(100) + 1;  
  
if (num <= 99) {  
    ServerRunningA()  
}  
  
else {  
    ServerRunningB()  
}
```

Une fois que vous en avez assez vu pour B, faites un test

- if $\text{countB} \geq 100$
 - //comparer un échantillon de taille similaire de résultats A et B
 - Wilcoxon(bResults(), rand(countB, AResults()))
- if $p > 0.05$ voir si notre erreur est plus petite que 5%. Si plus grande que 5%, cela veut dire que notre erreur est +grande que 5%
 - Attendez plus de résultats B
 - if $\text{countB} \geq 400 \ \& \ p > 0.05$ probablement qu'il n'y a pas de différence
 - on fait un test avec 400 users, et qu'on a plus de 5% de chances d'obtenir le mm résultat, que $p > 5\%$, le résultat n'est pas significatif
- Attention! 5% de 400 c'est pas énorme, mais 1 million peut devenir significatif
 - Avec un très grand nombre de résultats, de petites différences peuvent encore être statistiquement significatives avec 1 millions de users, une différence de 0.2 peut devenir significative
 - Besoin de déterminer les tailles d'effet pour connaître la différence

Quelle approche est la meilleure?

Quelle option choisir? Celle de gauche (50/1000 clics) ou celle de droite (30/1000 clics)?

"L'offre se termine ce samedi!"

$50/1000 = 5\%$ le % de clics

"L'offre se termine bientôt!"

$30/1000 = 3\%$

```
clicks <- c(50, 30)
```

```
essais <- c(1000, 1000)
```

```
prop.test(clicks, trials)
```

$0.4 = 20/50$

- p-value = 0.03015
- Ordre de grandeur
 - 20 ventes supplémentaires. Soit 1,4 fois plus de ventes pour la version à gauche

$1 + 0.4$

Quel test statistique ?

Répartition supposée	Type de données	Exemple de cas	Test
Gaussien/Normal	Continue	Revenu moyen par utilisateur payant	Welch's t-test
Binomiale	Ratio (pourcentage de succès)	Taux de clics entre 2 éléments (bouton bleu/ bouton vert)	Fisher's exact (petit) Chi-squared (grand)
Poisson	Compte	Nombre de transactions par utilisateur payant	E-test
Multinomiale	Compte par catégorie	Nombre de chaque produit acheté	Test du chi-carré (Chi-squared test)
Aucune hypothèse <small>test qui peut être utilisé avec n'importe quel répartition</small>	Continue	Délai entre les demandes pour le même fichier	Wilcoxon ou Mann-Whitney

Si on a un test qui peut être utilisé avec n'importe quelle autre répartition, pk on a besoin des autres tests?
 - Car quand on connaît la répartition, si on utilise le test qui utilise la répartition, on a + de PRÉCISION!

Essais sur le terrain chez Google

Toutes les nouvelles fonctionnalités doivent être déployées dans le cadre d'un essai

- Un ensemble d'indicateurs contrôlés par le serveur qui modifient le comportement de Chrome de manière dynamique, lors de l'exécution, sans envoyer de nouvelle version
 - Affiche les essais en cours. `chrome://version/`
 - Les essais contrôlent un ensemble de bascules/flags/toggles. `chrome://flags/`
- En tant que développeur, votre fonctionnalité fait partie d'un essai
- Les données de télémétrie sont collectées pour garantir le succès de la nouvelle fonctionnalité
- Les forums sont surveillés pour s'assurer qu'il n'y a pas de problèmes

Définir la mesure/les métriques à l'avance

- Clics
- Ventes
- Temps sur place
- Performance
- ...

Que mesurer ?

Remplissage automatique en un seul clic

ex: quand on achète qqc ça remplit automatiquement notre adresse, no de téléphone...

Faites des suggestions de remplissage automatique au premier clic de souris sur un élément de formulaire. – Mac, Windows, Linux, Chrome OS, Androïd

#enable-single-click-autofill

- Nombre moyen de champs de remplissage automatique qui sont corrigés ultérieurement.
- Vitesse moyenne pour remplir un champ.
- Seuil : avec la saisie automatique, en moyenne, les utilisateurs corrigent moins de 25 % des champs.
- Hypothèse : Avec la saisie automatique, en moyenne, les utilisateurs rempliront un champ en 50 % moins de temps. si on ne remplit pas le champ en moins de 50% de temps, on laisse de côté notre fonctionnalité

Que mesurer ?

Activer l'image dans l'image.

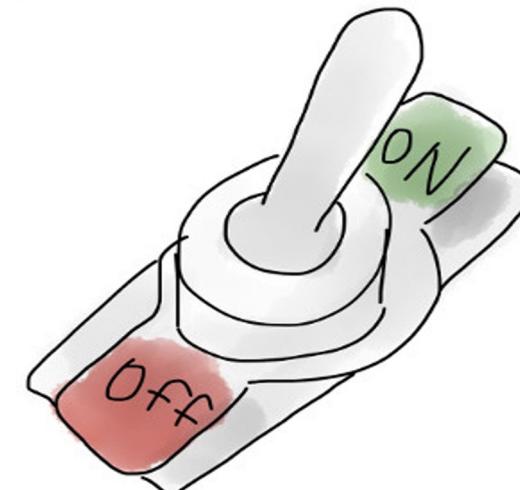
Activez la fonction image dans l'image pour les vidéos. – Mac, Windows, Linux, ChromeOS

#enable-picture-in-picture

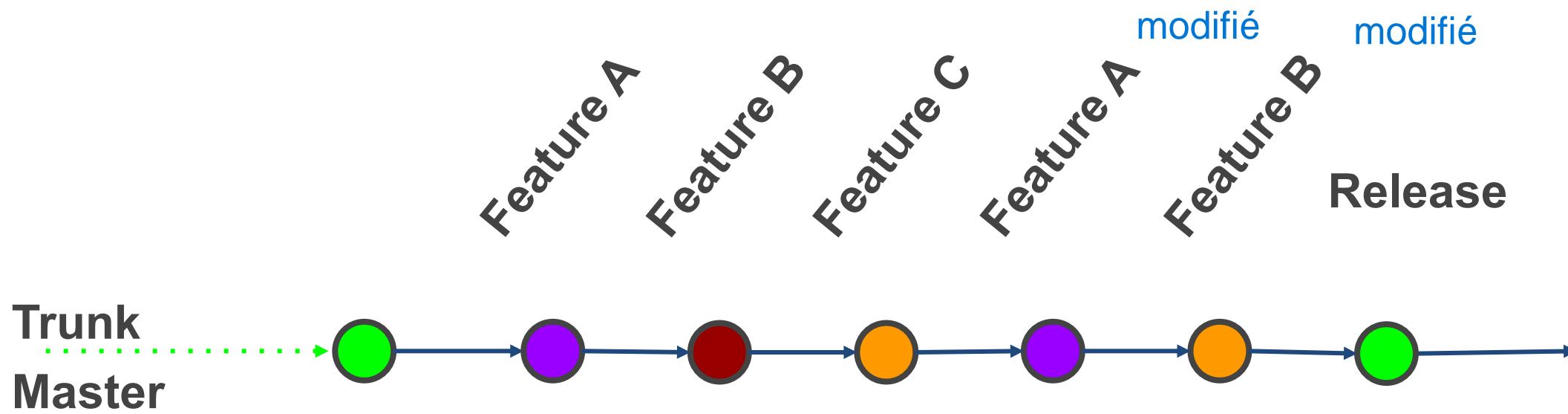
- Le nombre brut d'utilisateurs, s'il est inférieur au seuil, peut être en mesure d'éliminer la fonctionnalité
- Seuil : avec p-in-p activé, 2 % des utilisateurs utiliseront la fonction p-in-p

Bascules entre fonctionnalités

Feature Toggles

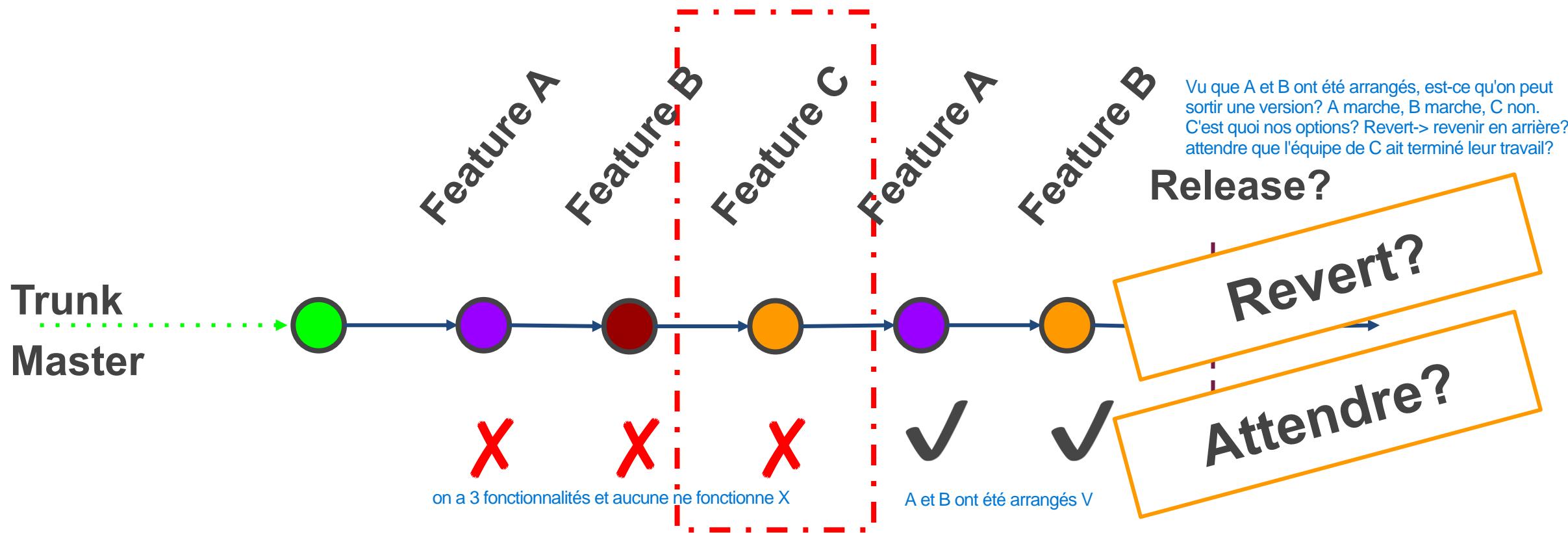


Google et Facebook/Meta ont un tronc unique pour tout leurs code

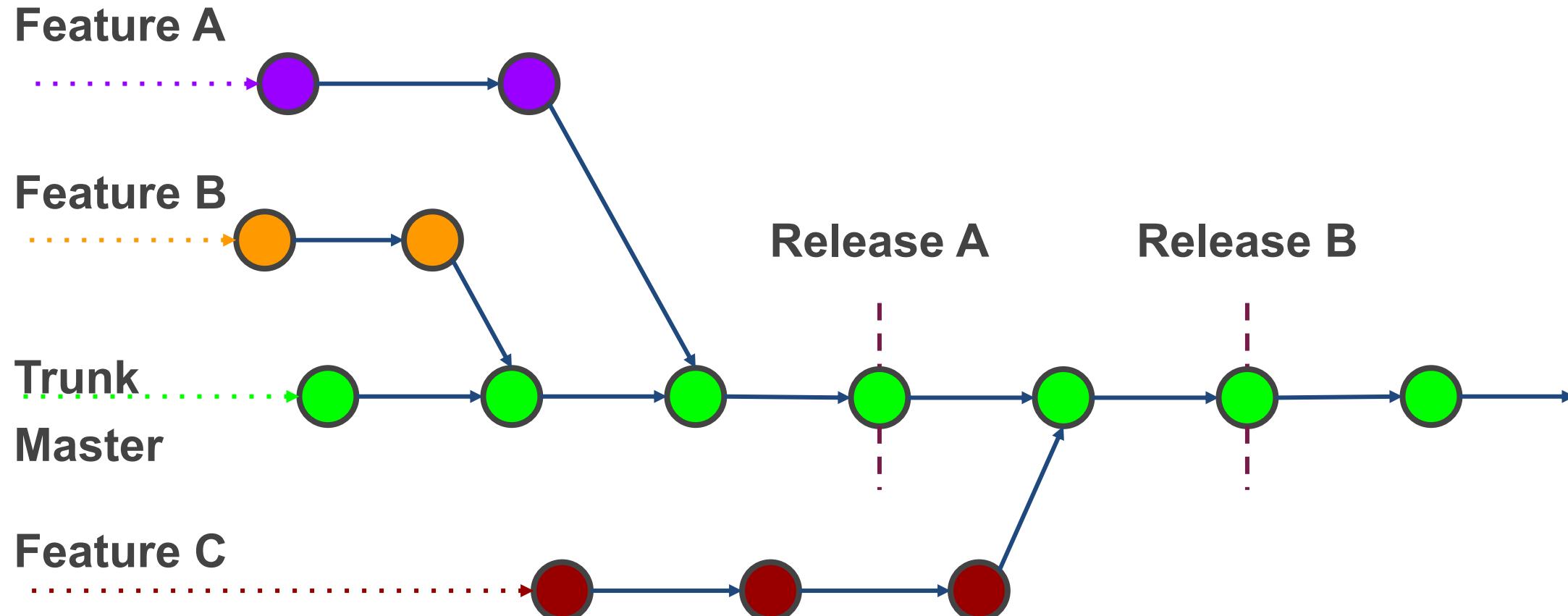


Que faire lorsqu'une fonctionnalité est intégrée et n'est pas prête à être publiée ?

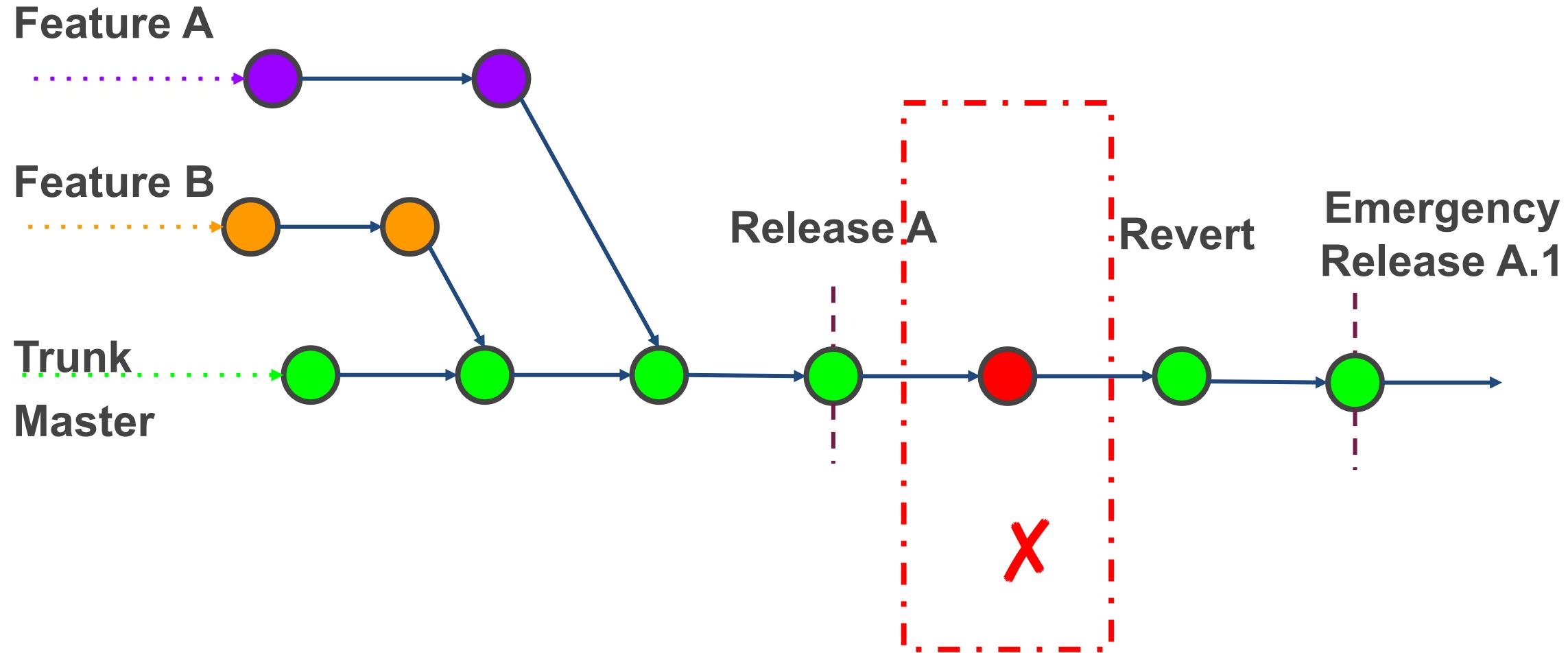
Trunk Based Development

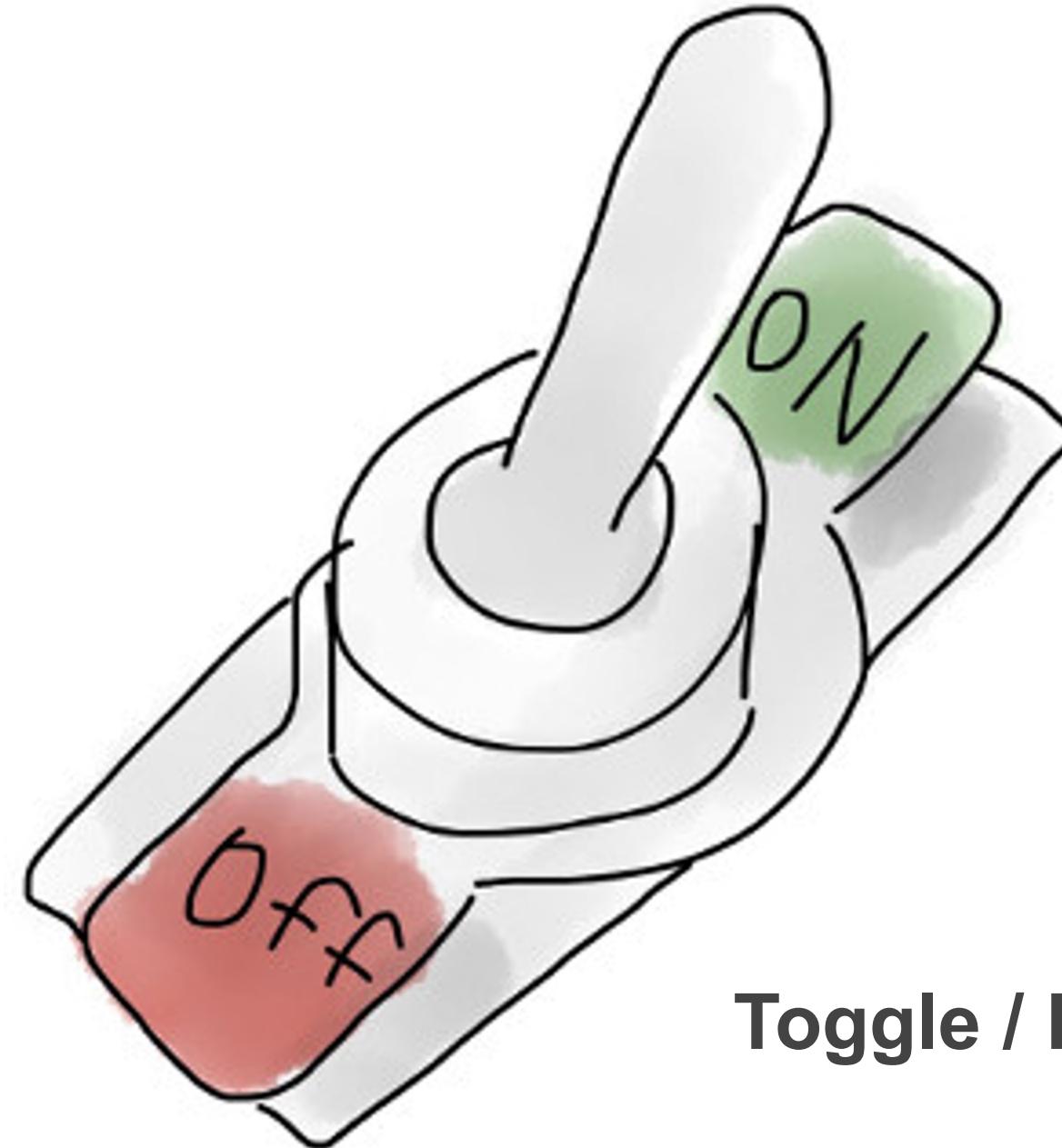


Gestion des fonctionnalités avec branches

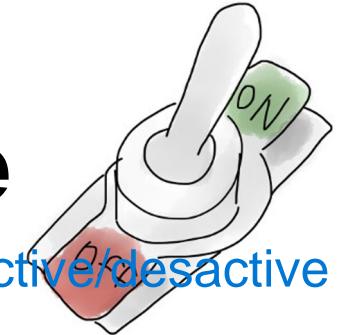


Les branches entraînent des problèmes de “merge” (merge hell) et sont difficiles à détruire





Toggle / Flag / Switch



Les “toggles” de fonctionnalités de Chrome

chaque nouvelle fonctionnalité va être caché derrière un if, et on active/désactive le if si on veut que la fonctionnalité soit active ou non.

```
452● if (command_line->HasSwitch(switches::kDisableFullscreen3d))  
453     return;
```

Bloc conditionnel les if

```
107 // Disable 3D inside of flapper.  
108 const char kDisableFlash3d[]  
109  
110 // Disable using 3D to present fullscreen  
111● const char kDisableFullscreen3d[]
```

= "disable-flash-3d";

Mécanisme de changement

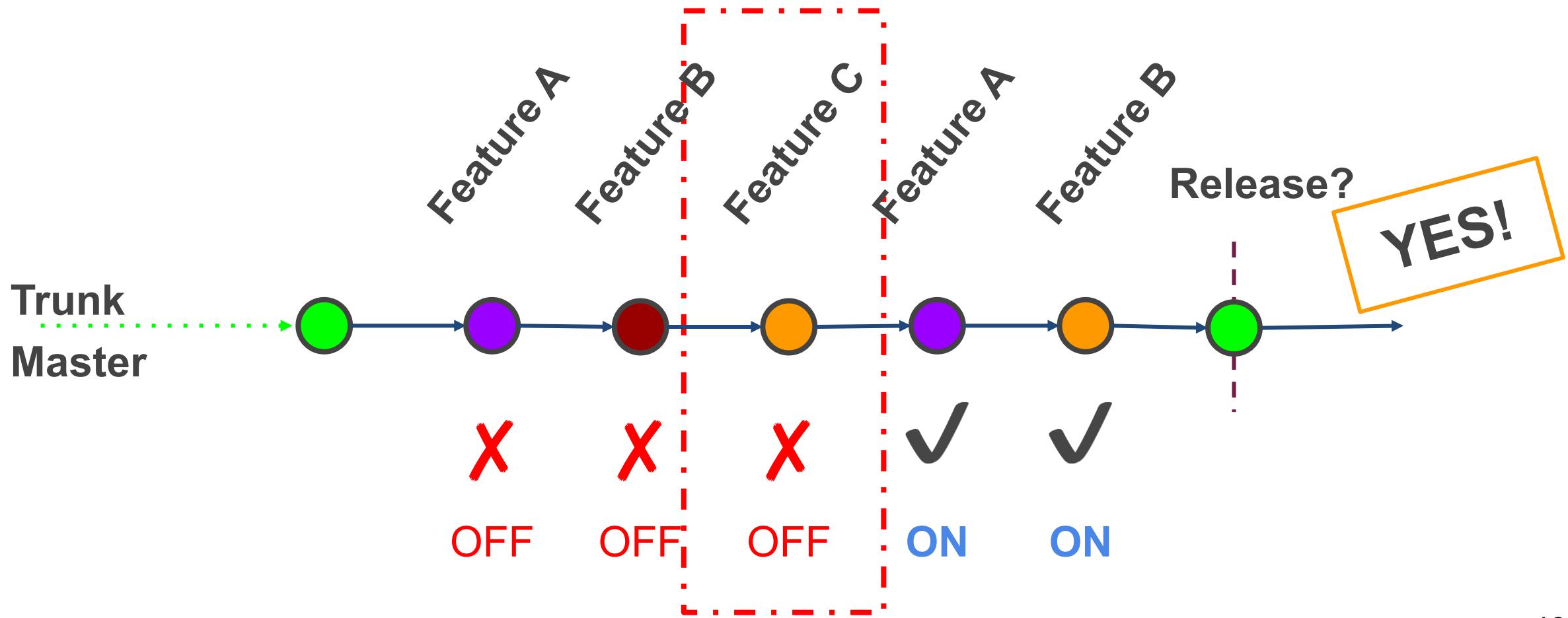
compositor_switches.cc (chrome/ui/compositor)

content_switches.cc (chrome/content)

gaia_switches.cc (chrome/google_ap)

Fichier de configuration

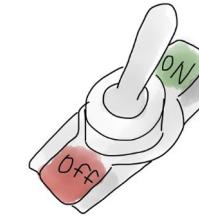
Trunk Development avec Feature Toggles



Technologie critique pour prendre en charge le DevOps

- Test A/B, Essais sur le terrain

X% des utilisateurs

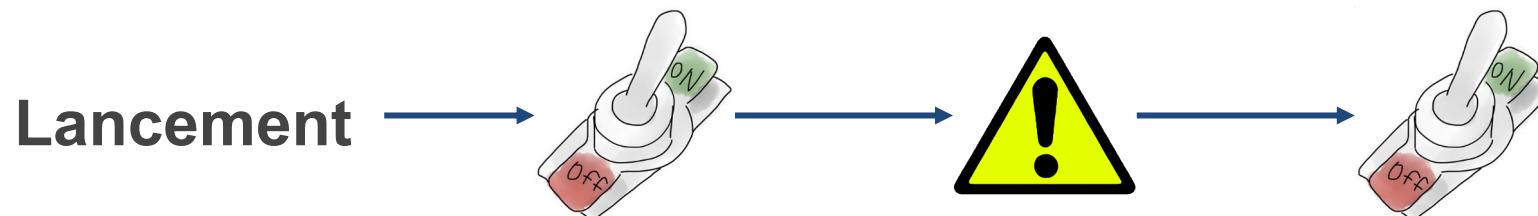


- Darklaunching, Version canary

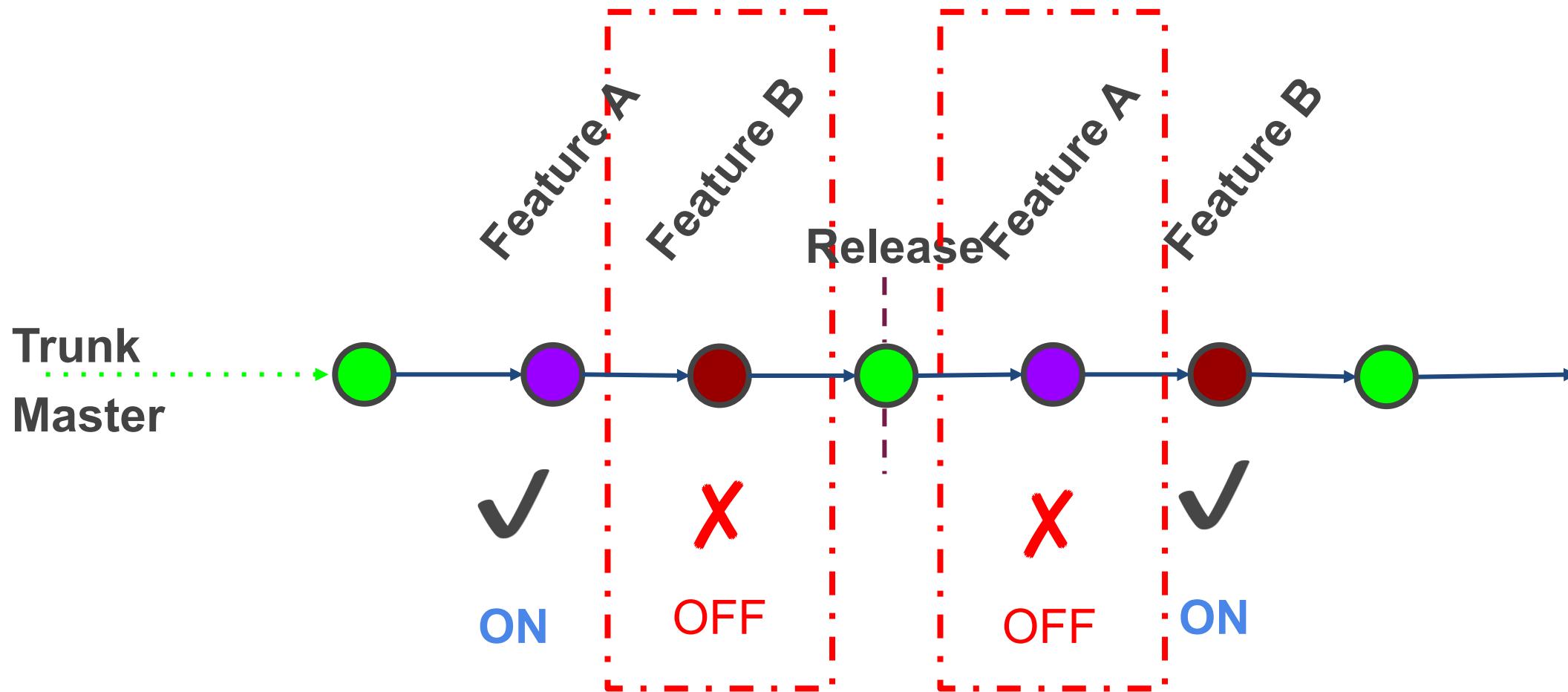
0% Users 50% Users 100% Users

Gens utilisant le logiciel sont les canary. Ce qu'on fait--> Activer la switch pr certaines activités pr un % d'utilisateurs. Ensuite, on va voir si la version a des problèmes. On va vérifier si plus de bugs créés, voir si + d'utilisateurs se plaignent dans les forums.... Si tt est normal, on augmente le nb d'utilisateurs

- Annulations de fonctionnalités



Trunk Development avec Feature Toggles



Erreur après une version avec des “Feature Toggles”

- Désactiver la nouvelle fonctionnalité [qui ne fonctionne pas bien](#)
- Vous n'avez pas besoin de créer une nouvelle version ou de déployer une nouvelle version
- Résoudre le problème et déployer à nouveau, progressivement

**DevOps est presque
impossible sans
développement basé sur
le tronc**

Changement de mentalité de développeur

- L'ancien code de fonctionnalité et le nouveau code de fonctionnalité sont compilés dans la même version
- Doit pouvoir basculer à l'ancienne version
- Les fonctionnalités sont écrites de manière plus isolée car elles doivent être derrière des “toggles”

Des recherches ont été effectuées sur les bascules sur



Source Code
39 Versions



Conférences, présentations et Blogs
13 Entreprises

Avantages des “Toggles”

- Concilier lancement rapide et développement de fonctionnalités à long terme *très flexible dès qu'il y a un changement, on peut désactiver s'il y a un problème, on peut faire des tests A/B (une version pr une personne et une autre version différente pr l'autre personne). Fonctionnalités concues de façon indépendantes, donc pas trop de conflits.*
- Déploiement flexible des fonctionnalités
- Activation des changements de contexte rapides
- Les fonctionnalités sont conçues pour être basculables

Pièges, dettes et catastrophes causées par des “Toggles”

- “Toggle Debt” temps perdu de compilation
 - C'est comme commenter du code mais garder l'ancien code compilé et “live”
- Test de fonctionnalité combinatoire
 - Les fonctionnalités peuvent changer pendant que le système est en cours d'exécution
 - Quelles fonctionnalités devons-nous tester ?

Courbes de survie pour les Toggles

toggles survivent longtemps

3 types de Toggles

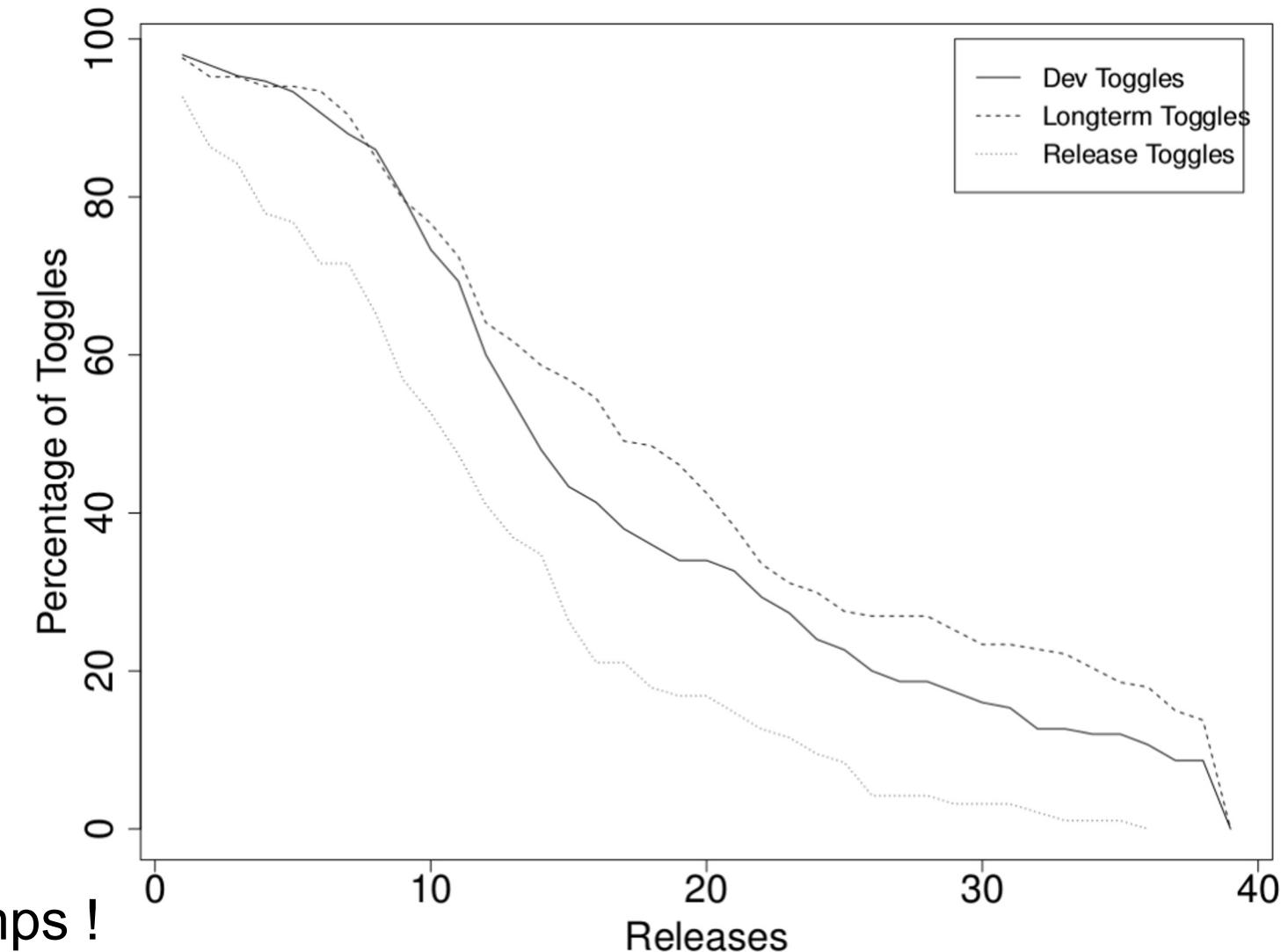
- Dev
 - Debugging
 - Tests

• Longterm

pour des clients qui demandes caractéristiques spécifiques

Règles commerciales : Différents clients obtiennent des caractéristique différente

- Release
 - Sorties de fonctionnalités

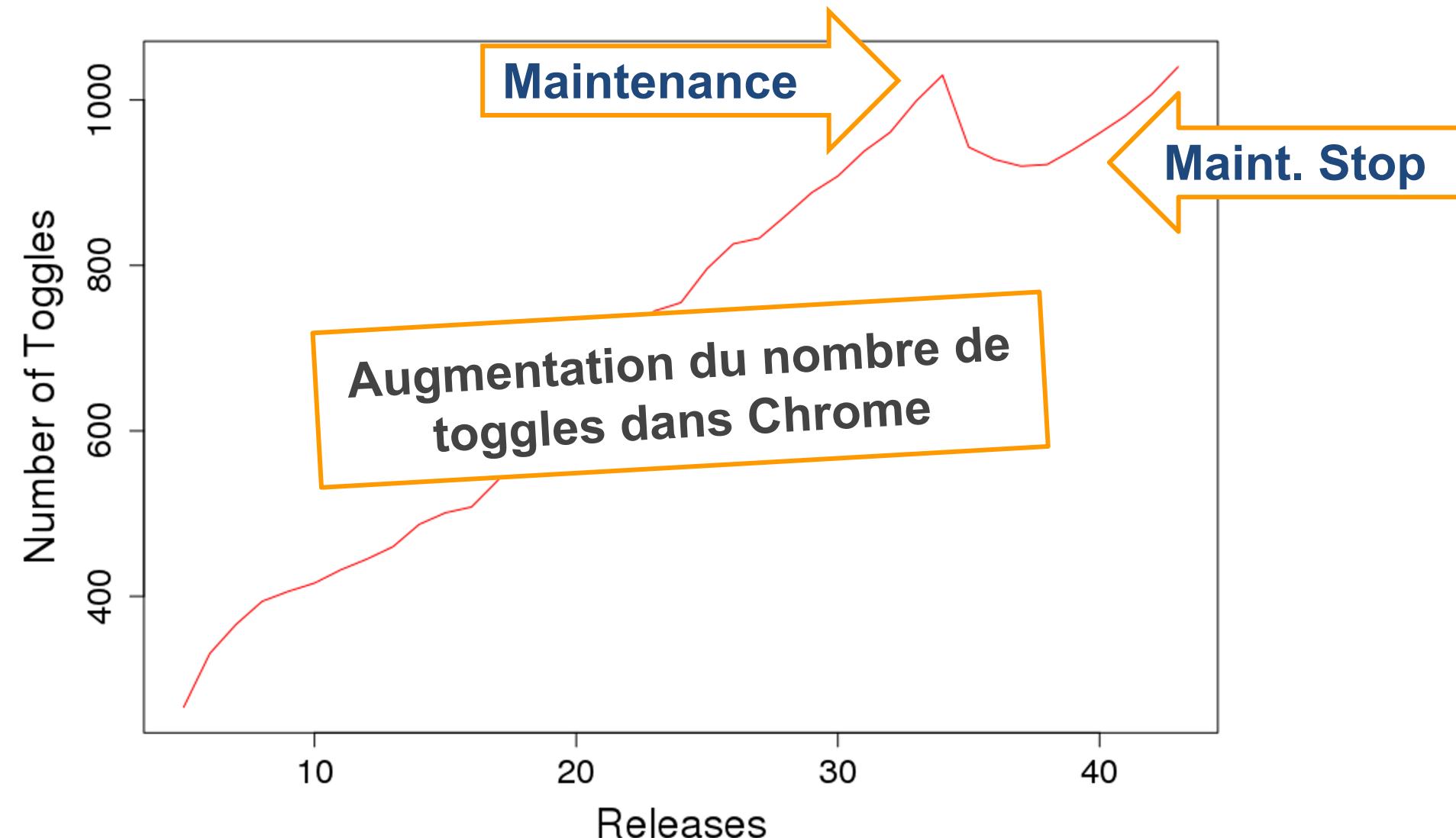


Les Toggles survivent longtemps !

Toggle inertie/dette

- Impossible de supprimer les Toggles tant que vous n'êtes pas sûr que la nouvelle fonctionnalité/le nouveau code fonctionne
 - Essentiellement, l'ancien code est commenté, mais il est toujours “live” !
- Ce retard signifie
 - On oublie souvent ce qu'on doit supprimer exactement
 - Nous ne voulons pas le faire parce que nous sommes déjà passés à autre chose

Une des pires dettes techniques



Knight capital group

- Gestion d'un volume quotidien moyen de transactions de plus de 3,3 milliards de transactions
- Négociant plus de 21 milliards de dollars... quotidiennement.
- 31 juillet 2012, Knight disposait d'environ 365 millions de dollars en espèces et équivalents.
- Un "**flag**" réutilisé a ramené de la mort l'ancien code "Power Peg".
- A déclenché une boucle sans fin qui:
 - Knight a commencé à négocier contre lui-même
 - Knight a perdu 460 millions de dollars en 45 minutes.

Combinatorial Hell

problèmes: on oublie cmt utiliser les switch

```

if (command_line->HasSwitch(switches::kDisableInstantExtendedAPI) ||
    command_line->HasSwitch(switches::kEnableLocalOnlyInstantExtendedAPI) ||
    command_line->HasSwitch(switches::kEnableLocalFirstLoadNTP)) {
    return false;
}
if (command_line->HasSwitch(switches::kDisabl
    return true;
```

Combinaison de Toggles

```

if (command_line.HasSwitch(switches::kDisableDistanceFieldText)) {
    is_distance_field_text_enabled_ = false;
} else if (command_line.HasSwitch(switches::kEnableDistanceFieldText)) {
    is_distance_field_text_enabled_ = true;
} else {
    is_distance_field_text_enabled_ = false;
```

Propagation des effets avec
des méthodes utilitaires

Gérer les combinaisons

Notre responsabilité que notre fonctionnalité fonctionne. Ne jamais envoyer en production qqc qui n'est pas testé. Tester progressivement et envoyer en production(déployer) progressivement est la bonne approche

- Testez et déployez progressivement toute combinaison de fonctionnalités que vous utiliserez en production
- N'utilisez pas de combinaisons en production que vous n'avez pas déjà testées et déployées progressivement

Diapositives basées sur le travail de :

Prof. Weiyi Shang

Prof. Peter Rigby

Localisation des défauts à l'aide de données historiques

LOG3430 Méthodes de test et de validation du logiciel

La plupart des tests n'échouent pas



**Sur 850 000 tests
exécutés chez Google,
0,29 % des tests ont
échoué**



**Sur 120 millions
d'exécutions de tests
sur Firefox, 1,6 %
échouent**



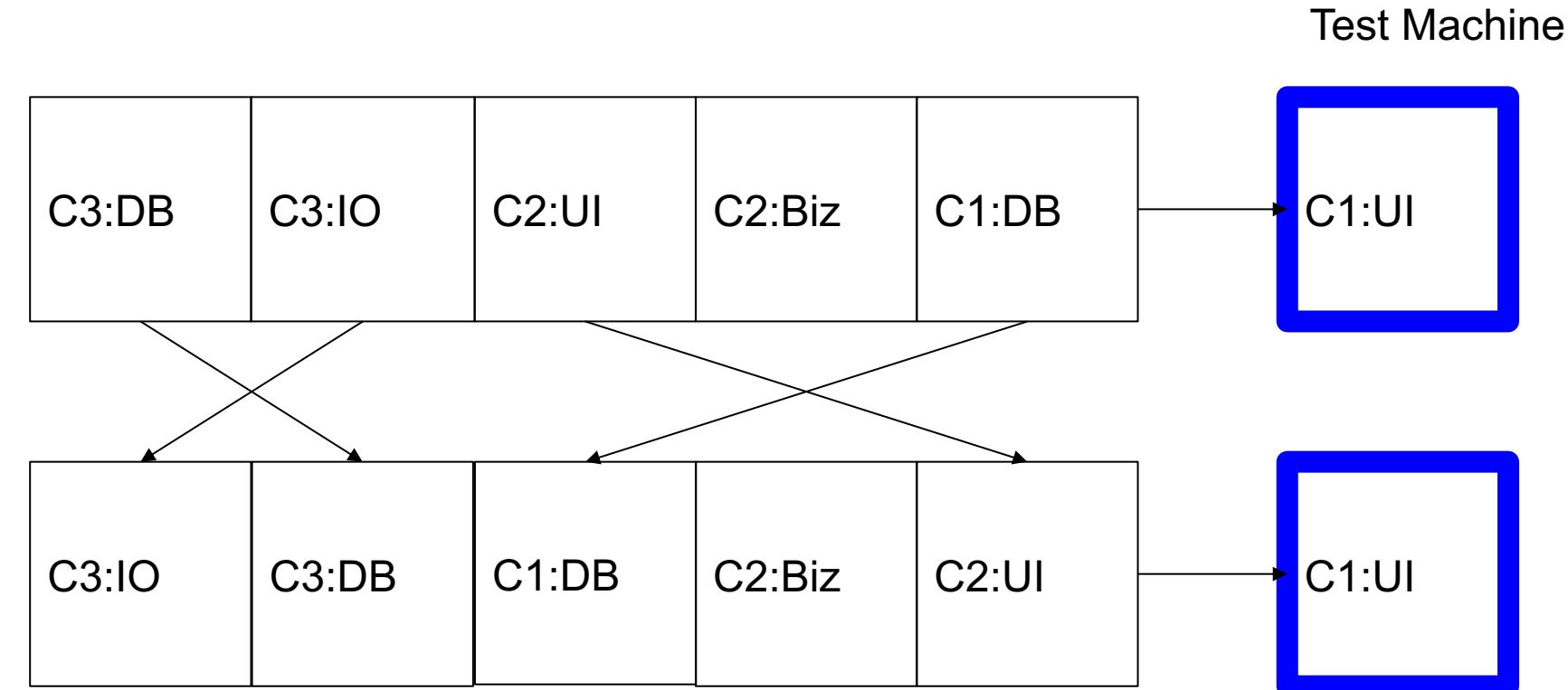
Historical Failure Probability

(S. Elbaum, G. Rothermel, and J. Penix)

Historique de tests

Test Name	Historical Failure rate
UI _{test}	10%
Biz _{test}	5%
DB _{test}	2%
IO _{test}	1%

C1, C2, C3 ce sont des critères de tests. C1 est le test le + critique, C3 est le test le + critique
 Critique veut dire c'est quoi les chances que le test va échouer basé sur l'historique



Après chaque test, nous avons de nouvelles informations

Test	Test	Co-failure
UI _{test}	DB _{test}	60% Test UI avec Test BD, 60% de proba d'échouer
UI _{test}	Biz _{test}	50% Test UI avec Test business logic
UI _{test}	IO _{test}	10%

Test Machine

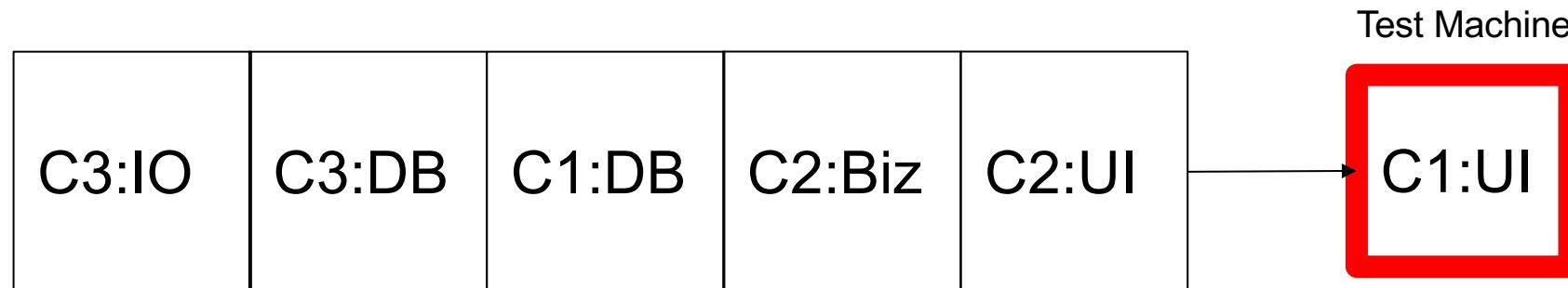


Étant donné que l'interface utilisateur a échoué, ajustez la probabilité d'échec des tests en file d'attente

Test	Test	Co-failure
UI _{test}	DB _{test}	60%
UI _{test}	Biz _{test}	50%
UI _{test}	IO _{test}	10%

Probabilité conditionnelle

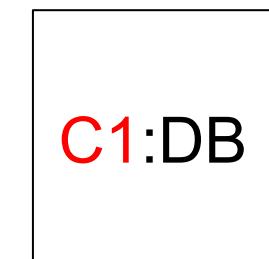
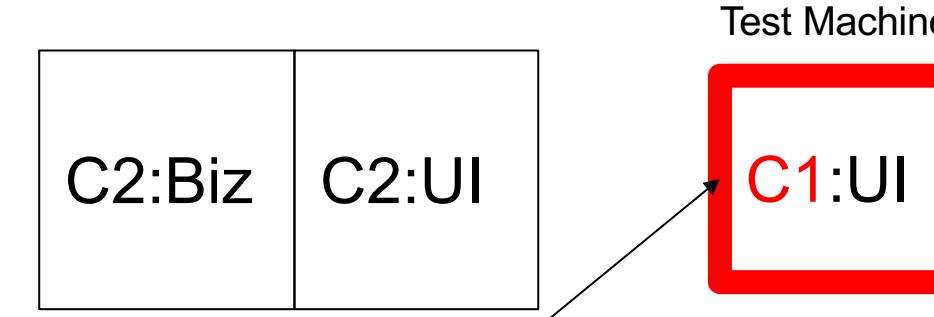
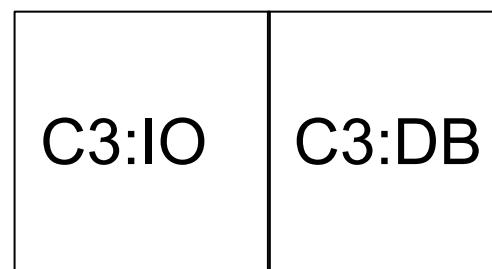
La chance que 2 tests se causent une interference et augmentent les probas qu'un des 2 tests échoue



Test	Test	Co-failure
DB _{test}	Biz _{test}	80%
UI _{test}	DB _{test}	60%
Biz _{test}	UI _{test}	1%

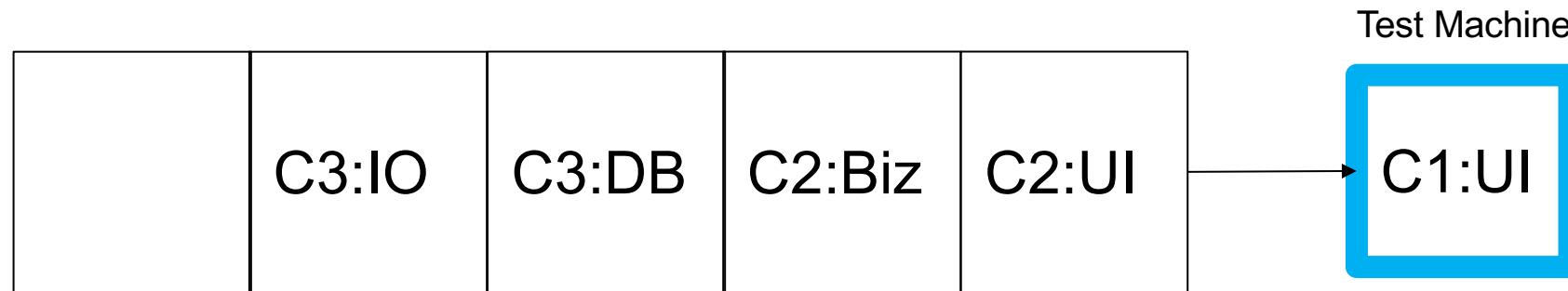
Probabilité conditionnelle

Si on sait que le test de BD a + de chances après mon test de UI, on va les faire ensemble au lieu de faire les tests dans un autres ordre



Test	Test	Co-failure
DB _{test}	Biz _{test}	80%
UI _{test}	DB _{test}	60%
Biz _{test}	UI _{test}	1%

Probabilité conditionnelle



La réalité est plus complexe

Nous aimerais réorganiser globalement tous les tests.

Cependant,

- 1,000+ demandes de test par minute pendant les heures de travail
- Des milliers de tests x des millions d'exécutions de tests
 - Énorme distribution de co-échec

Probabilité conditionnelle de réorganiser les tests pour **un seul changement**

- Utiliser une fonction (avec score) pour ordonner globalement les tests

Proba que 2 tests échouent ensemble

Scoring Function

Étant donné
l'échec de l'IU,
probabilité que la
BD échoue

Adjuster score
existant

$$new_sc = previous_sc + (P(t_2 = fail | t_1 = fail) - 0.5)$$

Réduire score si
relation < 0.5

Accélération des résultats de simulation par rapport à la baseline FIFO

	Google dataset	CoFailureProbability organiser les tests pour augmenter les chances que les tests échouent ensemble avec historique	PastFailureProbability on a les données antécédentes de chaque test, et on voit si un test a + de chances d'écouler que les autres
1,2,3, x,x,x	FirstFail On arrête de tester dès qu'on a une faute, donc dès qu'un test échoue	31% pourcentage d'accélération 31% de proba de détecter le 1er échec	4%
1,x,2, x...3	AllFail On va attendre d'avoir vu tt les échecs possibles pour arrêter	39% 39% de proba de détecter tous les échecs	5%

Conclusion sur la sélection et la priorisation

Des approches simples basées sur les échecs passés fournissent des prédictions précises sur les tests qui échoueront

- Voir [l'article](#) (Test Re-Prioritization in Continuous Testing Environments, Zhu et al., 2018)

Les approches historiques simples surpassent les approches plus complexes basées sur les coûts et les règles d'association

- Plus de détails dans cet [article](#) (Improving Test Effectiveness Using Test Executions History: An Industrial Experience Report, Najafi et al. , 2019)

Quantifier les Flaky tests pour trouver des instabilités de test



Un « Flaky Test » réussit et échoue sur le même build

(mm version du code)

Des tests peuvent échouer des fois et passer d'autres fois

Ex: la rapidité de la machine, l'environnement de la machine, dépendance envers d'autres systèmes externes

Pourquoi garder un « Flaky test »?

Certains tests ont un non-déterminisme inhérent

- Bruit matériel/environnemental
 - Propriétés asynchrones
 - La décision n'est pas basée sur des règles, mais sur un modèle de données d'IA
- test qui donne des réponses un peu différentes, mais c'est mieux d'avoir un test que de ne pas avoir de test du tout

Tests défectueux

- Certains tests peuvent être très coûteux à réparer

Selon Google

Ex: on a 10 tests, 10 de mes 10 tests n'ont pas échoué car mon code correspondait exactement au comportment voulu. Si un test échoue parmi les 10, il y a 84% de chances que ca soit dû à qqc d'irrégulier , donc quand un test échoue pr la 1ere fois faut pas nécessairement lui faire confiance. S'il échoue pls fois, on peut lui faire confiance.

84 % des tests qui échouent pour la première fois sont des échecs irréguliers

1,5 % des tests présenteront un « faux » résultat 1,5% des tests qui sont Flaky

Malgré de nombreux efforts, le taux de « Flaky test » reste constant

- c'est-à-dire que pour chaque « Flaky test » fixé, un nouveau est ajouté

[1 sur 7 des tests écrits par nos ingénieurs de classe mondiale échouent parfois d'une manière qui n'est pas causée par des modifications du code ou des tests] - Google



Facebook (Machalica et al.)

Réexécutez les tests de Facebook jusqu'à 10 fois!

- Flaky : s'il y a au moins un « pass »
- Échec : si les 10 exécutions sont des échecs

Inclure les « Flaky test » dans le modèle de sélection des tests conduirait à "trois fois plus d'échecs de test". Ceux-ci gaspilleraient l'effort du développeur car ils sont « Flaky »



La solution de Microsoft

Exécutez le test 1000 fois et déterminez le résultat

- Si en dessous d'un seuil irrégulier
 - Mettre le test en quarantaine et signaler un bogue



La solution de Google:

Exécutez un test trois fois, ne signalez un défaut que s'il échoue trois fois de suite

Les développeurs commencent à ignorer les « Flaky test » à moins qu'ils n'échouent trois fois



Coûteux!

- Si un test d'intégration s'exécute en 15 minutes
 - Nécessite trois essais ou 45 minutes pour déterminer le résultat!



Étapes de quantification des « Flaky » tests

nb d'essais pour vérifier notre test, le nb de fois qu'il a donné des faux résultats. S'il était supposé passer, le nb de fois qu'il a échoué et le nb de fois qu'il était supposé échouer mais qu'il a passé

1. Mesure du degré de « Flakiness » d'un test ($\text{FlakeRate} = \text{flake}/\text{essais}$)
2. Établir la ligne de base FlakeRate sur une version stable
3. À quel point mes tests sont-ils « Flaky » ?
 - a) Nombre d'exécutions pour avoir StableFlakeRate statiquement fiable
4. Probabilité de changement à l'état stable (distribution binomiale)
5. Prioriser les relances pour trouver des instabilités (probabilité d'un ensemble de relances)

Résultats des tests

x: test échoue

V: test passe

Bogue

T1: x (TP, bogue détecté)

T2 V (FN)

Pas de bogue

V (TN, pas de bogue)

X (FP)

Bons résultats

- Le test échoue et trouve un défaut (TP)
- Le test réussit et il n'y a pas de défaut (TN)

Résultats de « Flaky » test

- Un test échoue, mais aucun défaut n'est trouvé dans le logiciel (FP)
 - Efforts inutiles pour enquêter sur une défaillance irrégulière
- Un test réussit, mais une erreur passe à la production (FN)
 - Crash potentiel de l'utilisateur final (slipthrough)
crash de l'utilisateur,
l'utilisateur détecte le bogue

Quelle est l'exactitude d'un test?

$$\begin{aligned}\text{Exactitude} &= (\text{TP} + \text{TN})/\text{essais} \\ &= (\text{fail_with_fault} + \text{pass_no_fault})/\text{exécutions}\end{aligned}$$

C'est les deux T1

$$\begin{aligned}\text{FlakeRate} &= (\text{FP} + \text{FN})/\text{Runs} = 1 - \text{precision} \\ &= (\text{fails_no_fault} + \text{slipthroughs})/\text{exécutions}\end{aligned}$$

C'est les deux T2

Nous pouvons dire qu'un test est exacte à 99% ou nous pouvons dire qu'il « Flake » 1/100 fois

Nous pouvons mesurer le FlakeRate sur une période de temps

Ligne de base : quels tests sont « Flaky »

Définition : un test « Flaky » s'il réussit et échoue sur le même build

Exemples de versions stables

- Une version qui a été exécutée avec succès en production
- La dernière version stable donnée à un client

Besoin d'une version stable pour que

- Tout échec de test est un FlakyFailure (c'est-à-dire un faux positif)
- Toute réussite au test est une bonne réussite (c'est-à-dire vrai négatif)

Stable Build FlakeRate

Quand les bogues ont été corrigés, ma situation est stable

Pas de bogue

V (TN)
X (FP)

StableAccuracy = $\frac{TP+TN}{\text{exécutions}}$

= $\frac{\cancel{\text{fail_with_fault}} + \text{pass_no_fault}}{\text{runs}} = \text{passes/ exécutions}$

StableFlakeRate = $\frac{FP + FN}{\text{exécutions}} = 1 - \text{exactitude}$

= $\frac{\cancel{\text{fails_no_fault}} + \cancel{\text{slipthroughs}}}{\text{runs}} = \text{fails/ exécutions}$

Nombre d'exécutions pour déterminer le FlakeRate de base

Nombre d'échantillons dont nous aurions besoin pour avoir une confiance statistique dans une proportion (c'est-à-dire le FlakeRate)

Confiance statistique (estimation binomiale)

- 99 % de confiance = 680 exécutions
- 99,9 % de confiance = 1 097 exécutions

Exécutez un test plus de 1000 fois sur une version stable

Microsoft le font 1000 fois

$$n = \frac{Z^2 p(1 - p)}{e^2}$$

Un test réussi ne suffit pas !

Les Flaky tests ajoutent des exécutions supplémentaires, ce qui entraîne des dépenses supplémentaires.

**Les tests très “Flaky”
nécessitent de nombreuses
répétitions (95%)**

FLAKERATE	FLAKYFAILURES %	pass %	re-runs
0	0.00	100.00	0
1/2000	0.05	99.95	1
1/1000	0.1	99.99	2
1/500	0.20	99.80	3
1/100	1.00	99.00	15
5/100	5.00	95.00	72
1/10	10.00	90.00	138
2/10	20.00	80.00	249
3/10	30.00	70.00	322
4/10	40.00	60.00	369
5/10	50.00	50.00	384

La distribution binomiale peut réduire les exécutions

$$P_t(f, r, \text{FLAKERATE}) = \binom{r}{f} * \text{FLAKERATE}^f * (1 - \text{FLAKERATE})^{r-f}$$

5% de 1000 ex: 1000

- Quelle est la probabilité d'obtenir f échecs avec r tests
- P_t indique utiliser la probabilité d'obtenir l'ensemble de résultats de test à partir de répétitions
- FlakeRate est calculé à partir d'un build stable

Les tests n'ont plus de résultat binaire

Ils deviennent instables par rapport à la version de base FlakeRate

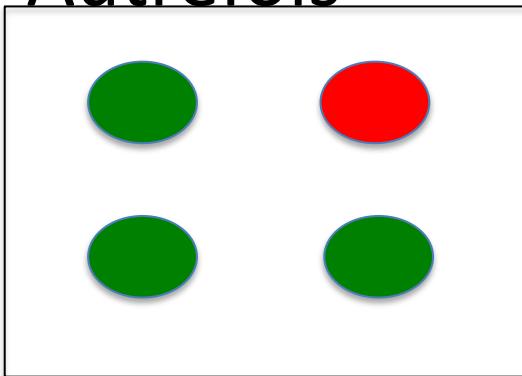
Est-ce que les tests échouent + que dans le passé?

Est-ce que les tests passent + souvent que dans le passé?

Si le test a changé de comportement, on doit aller voir le code src pour voir pk il a changé de comportement

Example:

Autrefois



4 tests

1/4 qui échoue

On veut être certain à 95% que c'est notre ratio

On cherche la valeur de 95%, donc 72 executions par test

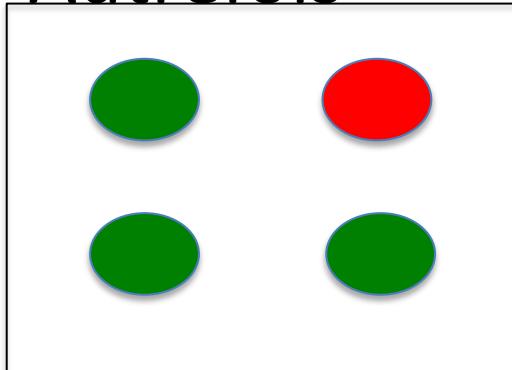
4 tests * 72 = 288 executions

-  = Pass = $\frac{3}{4} = 75\%$
-  = Fail = $\frac{1}{4} = 25\%$

Besoin de 288 executions
pour déterminer ratio avec
95 % de confiance

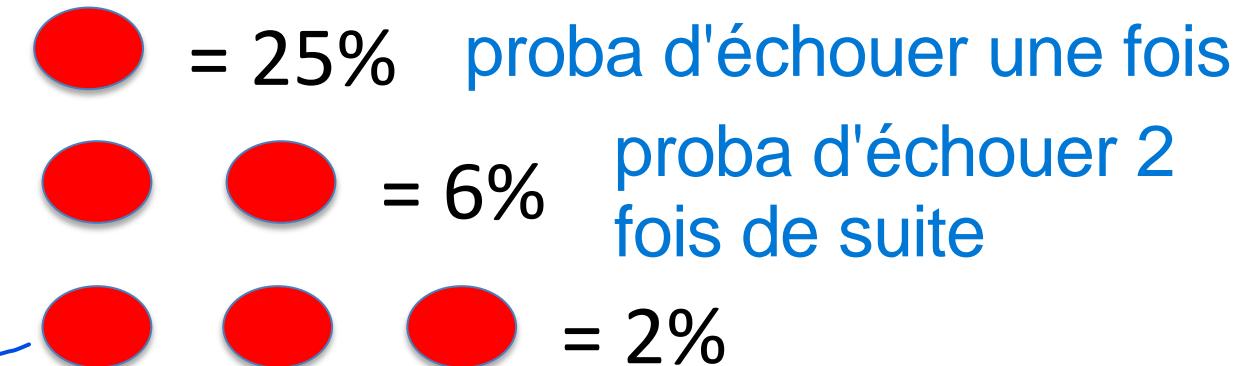
Quelle est la probabilité que tout échoue?

Autrefois



$$\begin{aligned}1/4 &= 25\% \\1/4^2 &= 6\% \\1/4^3 &= 2\%\end{aligned}$$

Aujourd'hui



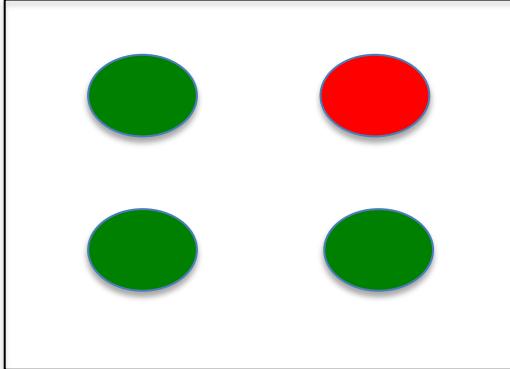
Google signale un échec

25 % d'échecs attendus, ont obtenu 100 % d'échecs

Coût 3 exécutions pour être en bas de 5%

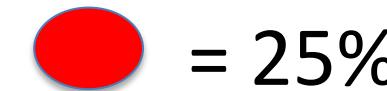
Une réussite (avec bruit)

Autrefois



Google signale à tort aucun
échec à 3 exécutions

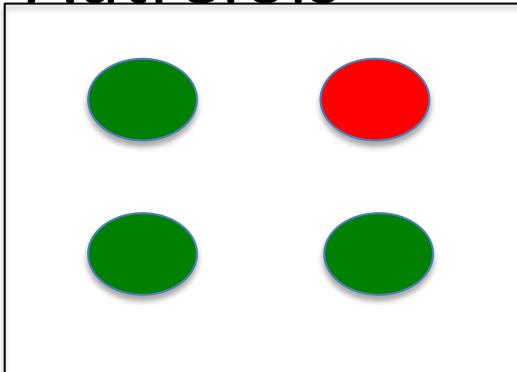
Aujourd'hui



25% d'échecs attendus, 75%
échecs obtenu
Coût: 4 executions pour être en bas
du 5%

Alternance réussite/échec

Autrefois



Aujourd’hui



= 75%



= 38%



= 42%



= 21%

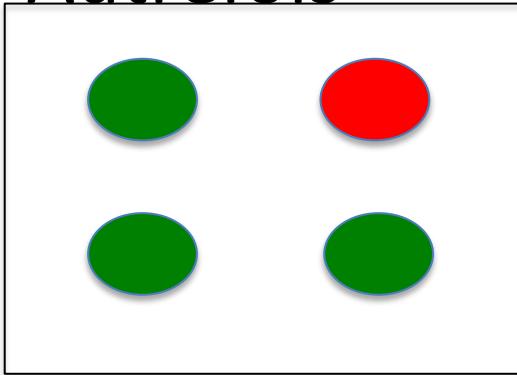
8 passes et 8 fails pour < 2%

25% d'échecs attendus, 50% échec obtenu

Coût 16 executions pour être en bas du 5%

Tout Passe

Autrefois



Aujourd'hui



= 75%



= 56%



= 42%

12 passes pour < 4%

25% d'échecs attendus, 0% échec obtenu
Coût 12 executions

Prioriser les relances

Quel test recommenceriez-vous en premier ?

1. T1 : FlakeRate = 10 %
2. T2 : FlakeRate = 5 %
3. T3 : FlakeRate = 15 %

Rép: T2-->On va refaire le test avec la situation la - probable
 .On va voir si on modifie notre Flake-Rate (72) ou on garde notre
 Flake-Rate

Ensuite on fait T1. À la fin on fait T3 car la proba la + grande en
 dernier

FLAKERATE	FLAKYFAILURES %	pass %	re-runs
0	0.00	100.00	0
1/2000	0.05	99.95	1
1/1000	0.1	99.99	2
1/500	0.20	99.80	3
1/100	1.00	99.00	15
5/100	5.00	95.00	72
1/10	10.00	90.00	138
2/10	20.00	80.00	249
3/10	30.00	70.00	322
4/10	40.00	60.00	369
5/10	50.00	50.00	384

Prioriser les relances

Quel test recommenceriez-vous en premier ?

1. T2 : FlakeRate = 5 %
2. T1 : FlakeRate = 10 %
3. T3 : FlakeRate = 15 %

Bruit vs Signal

Signal

- Le test commence à échouer beaucoup plus que prévu
 - Quelqu'un a probablement changé le code et il y a une erreur

Bruit

- Le test échoue aux niveaux attendus
 - Probablement juste l'interférence normale (problèmes asynchrones ou environnementaux)

Un "algorithme" pour trouver des changements dans le FlakeRate

données historiques

$$P_t(f, r, \text{FLAKERATE}) = \binom{r}{f} * \text{FLAKERATE}^f * (1 - \text{FLAKERATE})^{r-f}$$

1. Exécutez chaque test une fois
2. Calculer P_t
3. Ordonnez des tests par P_t
4. Exécutez le test avec le P_t le plus bas
5. Enquêter les tests qui montrent des FlakeRates hautement improbables (c'est-à-dire instables) Is situation la - probable est la plus étrange
6. Aller à l'étape 5 jusqu'à ce qu'il n'y ait plus de tests budgétés
 - ou confiance élevée atteinte (c'est-à-dire que FlakeRate est stable et que le comportement est stable)

Exemple

On sait historiquement:

T1 : FlakeRate = 10 % (10 % d'échec du test)

T2 : FlakeRate = 5 % (5 % d'échec du test)

T3 : FlakeRate = 15 % (15 % d'échec du test)

Nous avons un budget de test de: 5 fois pour chaque test maximum

Nous exécutons d'abord chaque test une fois.

Exemple

Le résultat du test est :

T1=> échec

T2=> échec

T3=> passe

Sur la base des résultats des tests, nous vérifions que c'est la probabilité de ce qui vient de se passer

T1=> P(échec) : 10 %

Proba de réussite = 1 - Proba échec

T2=> P(échec) : 5 %

1 - 15%

T3=> P(réussi) : 85 %

La situation la - probable est le T2, donc on répète le test numéro 2

Exemple

Le résultat du test est :

T1=> échec

T2=> échec

T3=> passe

Sur la base des résultats des tests, nous vérifions que c'est la probabilité de ce qui vient de se passer

T1=> P(échec) : 10 %

T2=> P(échec) : 5 %

T3=> P(réussi) : 85 %

Nous devons donner la priorité au test qui a le résultat le plus improbable, c'est-à-dire T2. Puis T1, T3

Exemple

Le résultat du test est :

T2=> passe

T1=> dans la queue

T3=> dans la queue

Sur la base des résultats du test, nous vérifions la probabilité de ce qui vient de se passer

T2=> P(échec, passe) : 9.5 %

9,5% est très plus bas que 10%, donc on refait T2

Nous devons donner la priorité au test qui a le résultat le plus improbable, c'est-à-dire toujours T2. Puis T1, T3

Exemple

Ordre jusqu'à date: T1,T2,T3
T2,T2

Le résultat du test est :

T2=> passe

T1=> dans la queue

T3=> dans la queue

Sur la base des résultats du test, nous vérifions la probabilité de ce qui vient de se passer

T2=> P(échec, passe, passe) : 13.5 % > 10%

13,5% est + élevée que la chance que T1 échoue, donc on va modifier l'ordre et on va faire T1

Nouvel ordre:

T1,T2,T3
T2,T2,T1

Nous devons donner la priorité au test qui a le résultat le plus improbable, c'est-à-dire T1. Puis T2, T3

Exemple

Le résultat du test est :

T1=> passe

T2=> dans la queue (13.5% chances de passer)

T3=> dans la queue (85% chances de passer)

Sur la base des résultats du test, nous vérifions la probabilité de ce qui vient de se passer

T1=> P(échec, passe) : 18 % On va faire T2, qui est la situation la - probable

Nouvel ordre:

T1,T2,T3

T2,T2,T1

T2

Nous devons donner la priorité au test qui a le résultat le plus improbable, c'est-à-dire T2. Puis T1, T3

Exemple

Le résultat du test est :

T2=> passe

T1=> dans la queue (18% chances de passer)

T3=> dans la queue (85% chances de passer)

Sur la base des résultats du test, nous vérifions la probabilité de ce qui vient de se passer

T2=> $P(1x$ échec, $3x$ passe) : 17.1%

On refait T2 car en dessous du seuil

Nouvel ordre:

T1,T2,T3

T2,T2,T1

T2,T2

Nous devons donner la priorité au test qui a le résultat le plus improbable, c'est-à-dire T2. Puis T1, T3

Exemple

Le résultat du test est :

T2=> passe

T1=> dans la queue (18% chances de passer)

T3=> dans la queue (85% chances de passer)

Sur la base des résultats du test, nous vérifions la probabilité de ce qui vient de se passer

T2=> P(1x échec, 4x passe) : 20.4%

On doit refaire T1

Nouvel ordre:

T1,T2,T3

T2,T2,T1

T2,T2,T1

Nous devons donner la priorité au test qui a le résultat le plus improbable, c'est-à-dire T1. Mais nous avons atteint le max pour T2 (5 fois pour T2). Alors nous faisons T1, T3 (mais pas T2).

Exemple

Après avoir terminé l'exécution de tous les tests, nous comparons la distribution des réussites/échecs de chaque test avec son historique.

Par exemple, le test T2 échoue historiquement à 5 %. Le test de cette version est dans une situation qui a 20.4 % de ce produire (test a échoué 20% du temps).

On peut décider s'il s'agit d'une déviance significative (normalement basée sur le test binomial des statistiques.)

Si cela est considéré comme une déviance significative, c'est un indicateur de "quelque chose de mauvais (par exemple, un bogue)".

Sinon, ne faites rien.

Dois-je corriger mon « Flaky » test ?

Les tests ne sont plus binaires, nous faisons de la détection d'anomalies

- Changements dans le FlakeRate stable (instabilités)

Tant que le comportement du système est stable, alors peut-être pas

- Vous obtenez toujours le signal du test en réexécutant et en examinant P_t

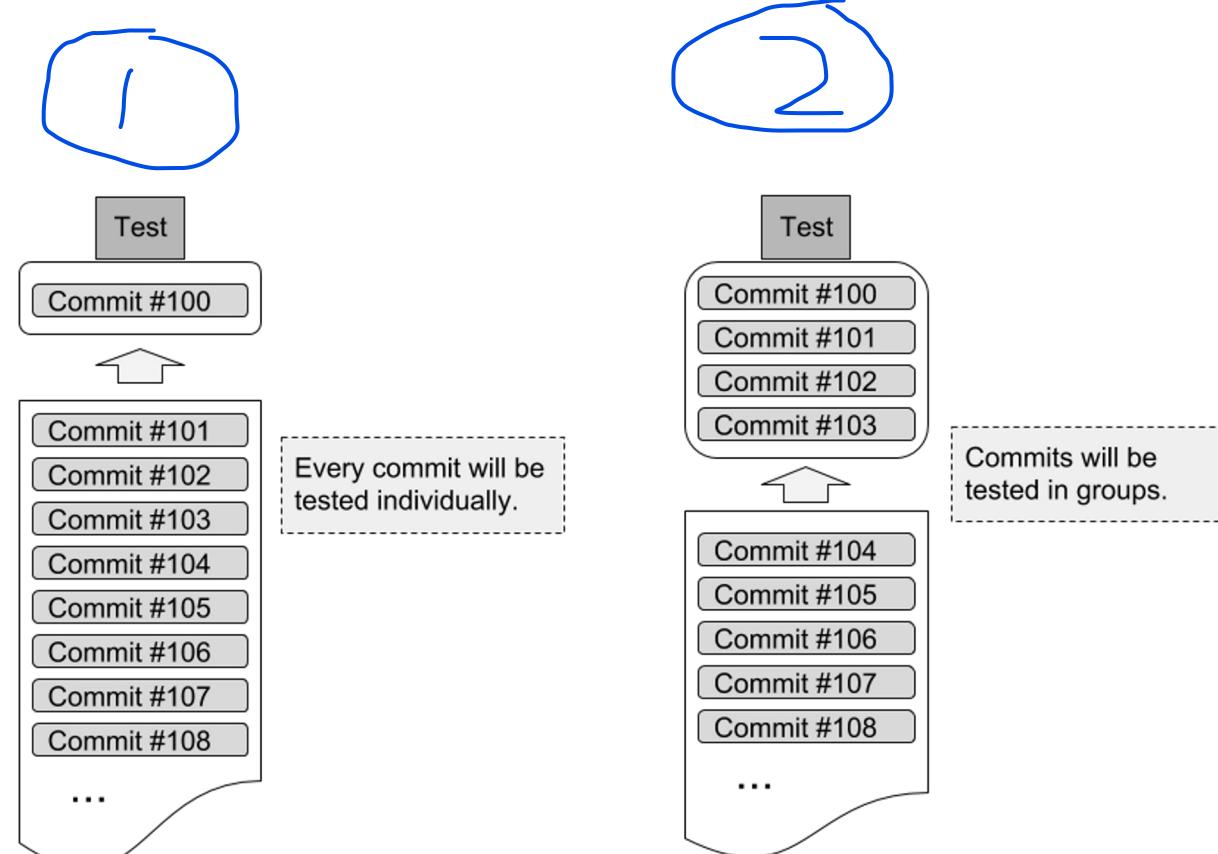
Cependant, rendre un test moins « Flaky » signifiera moins de bruit

- Tout échec deviendra plus improbable et nécessitera moins de relances

Compromis entre le coût de réparation du test et le coût de réexécution et de calcul de P_t

Tests par lots et prédition des coupables

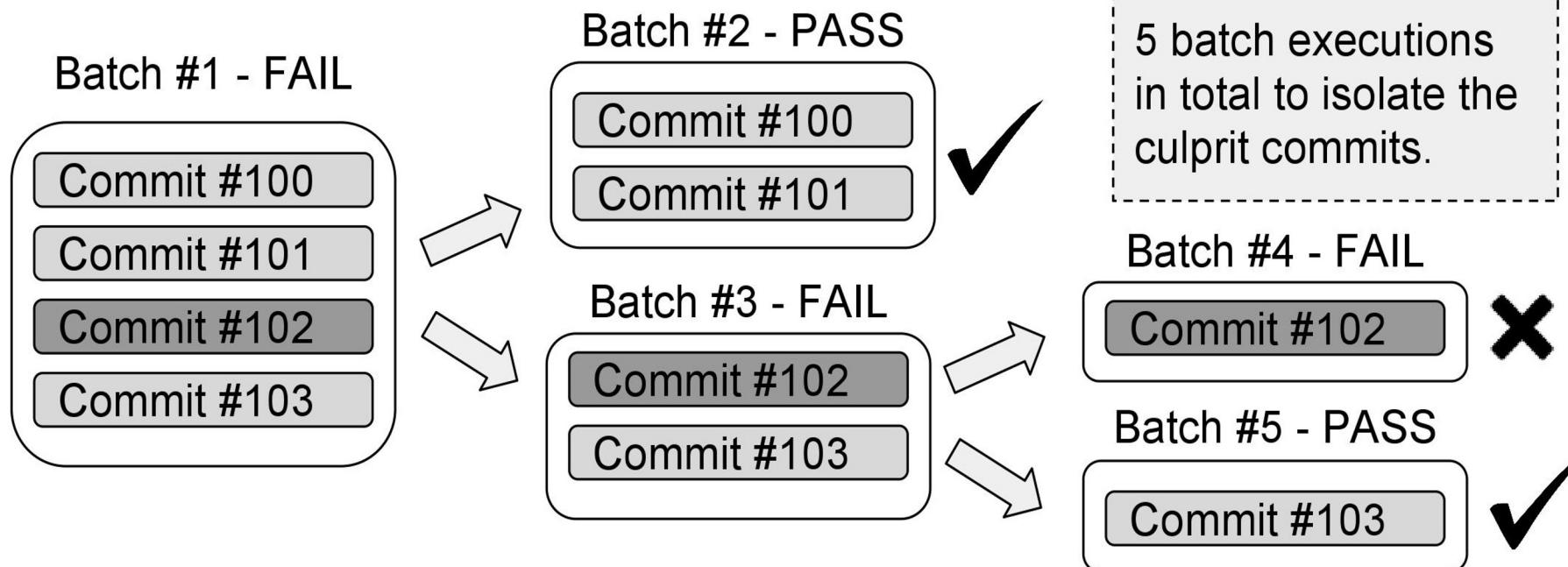
Contexte : Tester tout ou tester par lots



On utilise la procédure de tests sur chaque commit. À chaque fois que je soumets sur Git, je vérifie mon commit et le commit seul

Sln: regrouper les commit et tester chaque lot qui est complèt.
Par contre, on perd une utilité de tester en groupe, on perd le fait de détecter la src d'un bogue précisément, dans quel commit

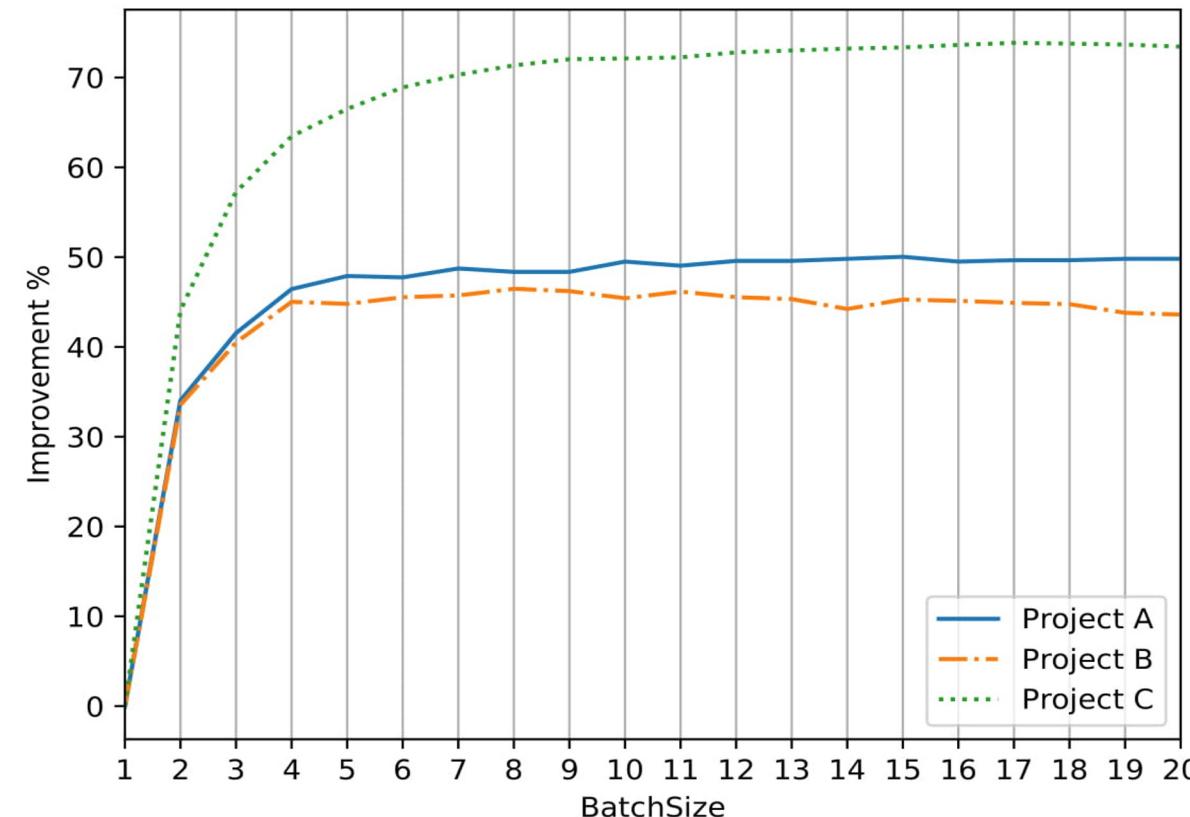
Bissection



**Dans quelle mesure
pouvons-nous réduire
les exécutions de tests
en testant par lots ?**

Moins de coupables, meilleure efficacité

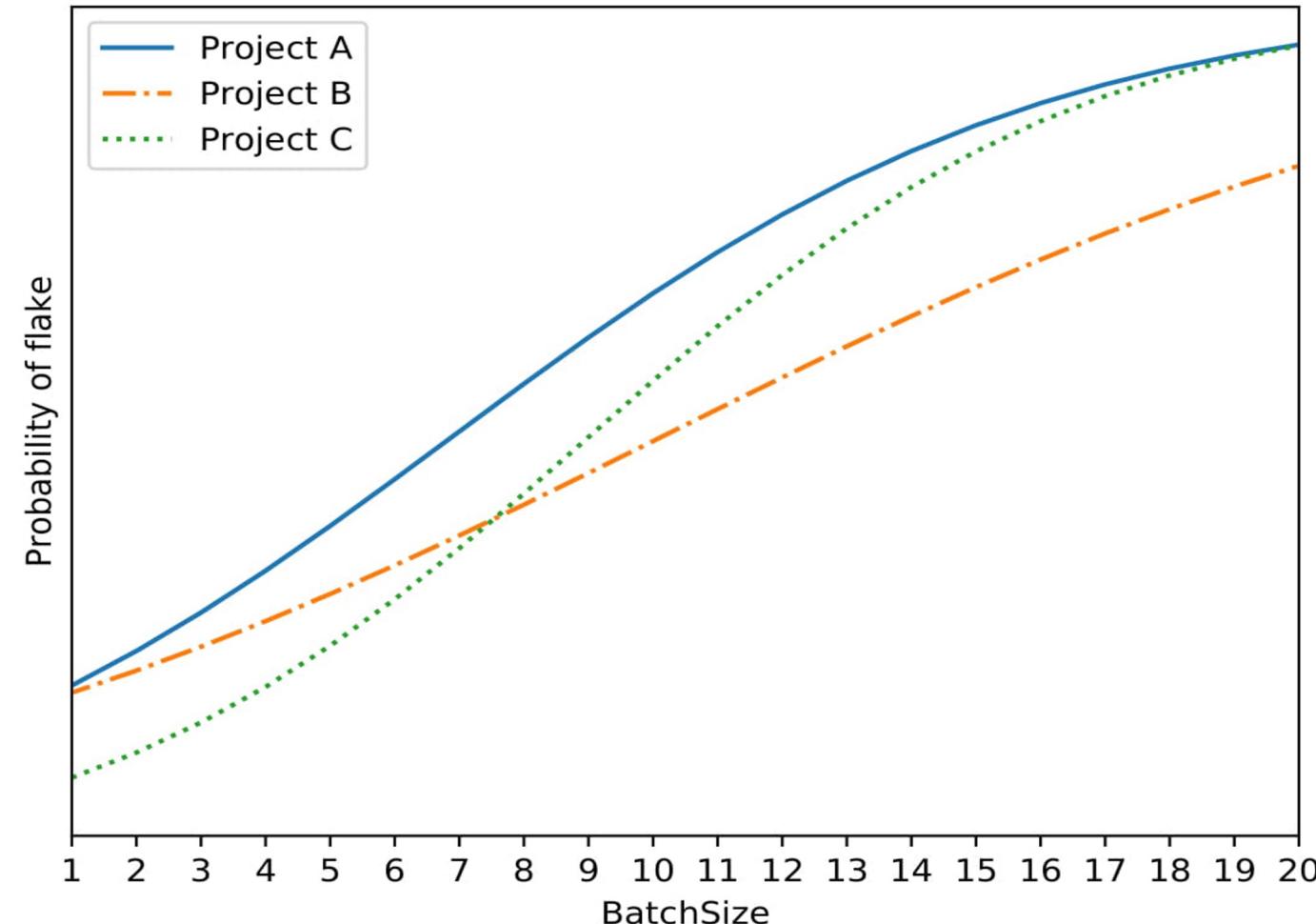
J'ai moins besoin de faire des tests si je n'ai pas bcp de bogues



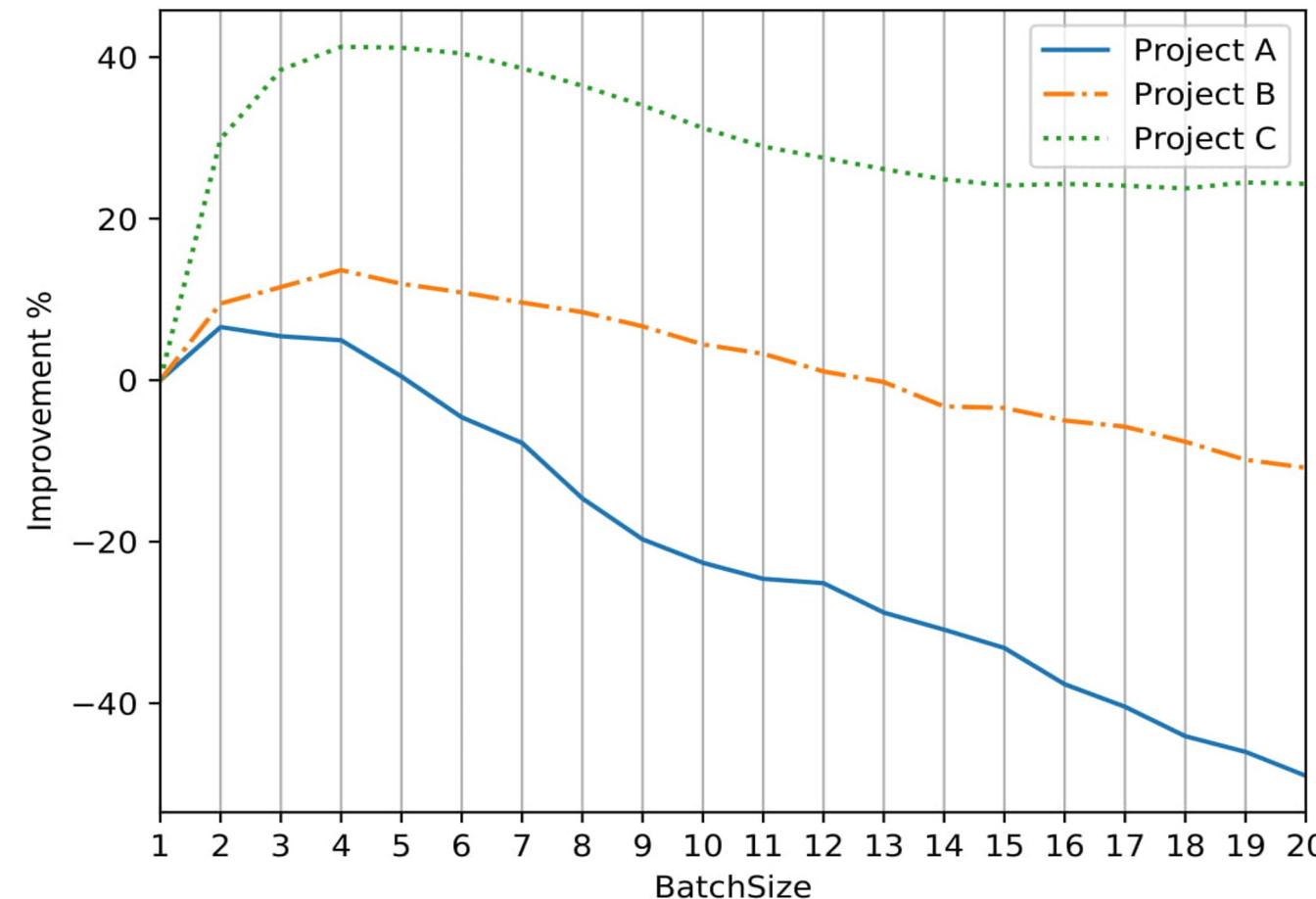
le nb de commits dans mon lot de commits

Comment la « flakiness » des tests affecte-t-elle les tests par lots ?

FlakeRate vs BatchSize



En considérant le FlakeRate



Git Bisect

Git bisect

Git viens de Linus Torvalds, le créateur de Linux
bisect = “diviser en deux parties égales”

Origines

- Linux n'exécute normalement pas de tests sur des serveurs
- Lorsqu'une régression se produit (c'est-à-dire que quelqu'un signale un problème), ils doivent déterminer où il a été introduit
- Git bisect effectue une recherche binaire pour trouver le commit défectueux

En pratique:

- Les tests prennent trop de temps, c'est-à-dire pas assez de temps pour tester chaque commit
- Les tests sont trop chers, c'est-à-dire qu'il n'y a pas assez d'argent pour tester chaque commit
- Ne testez que chaque commit X.
 - Si réussi, merge all
 - Si fail, bisect pour trouver le problème

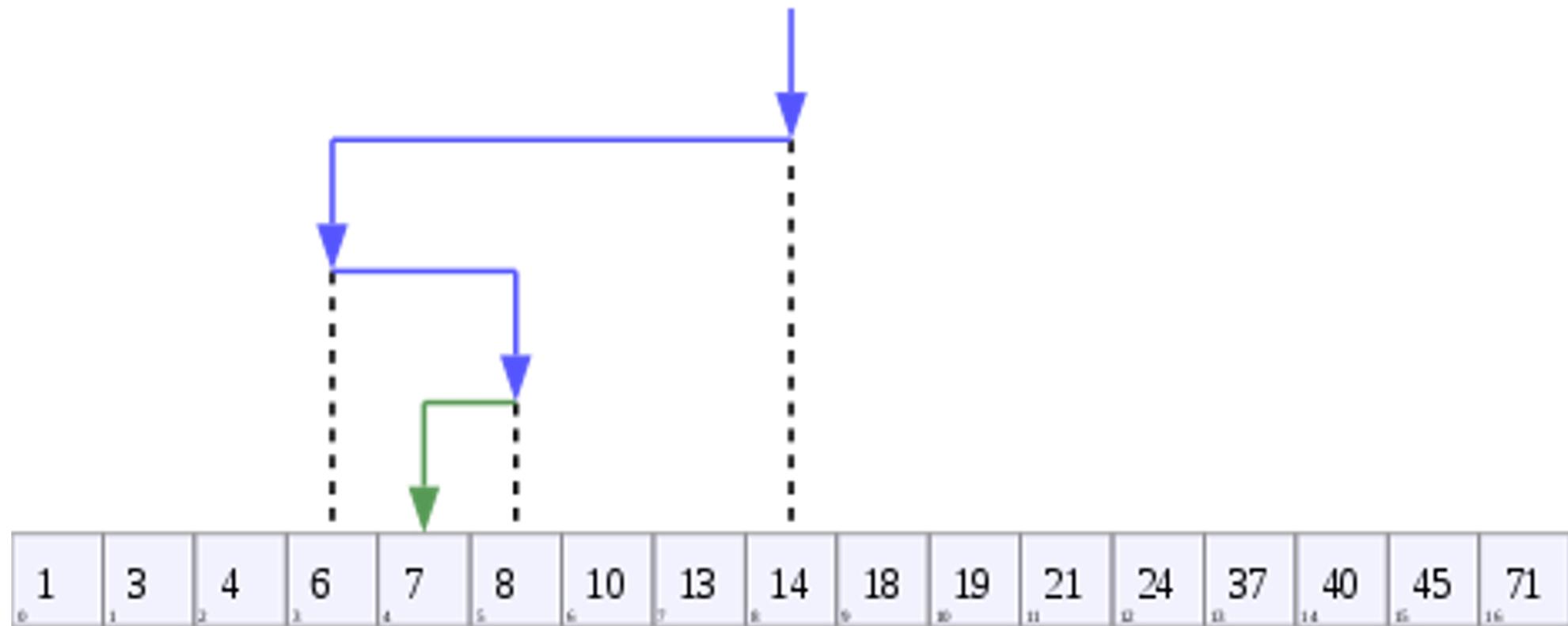
Exercice de recherche binaire



Objectif : trouver le chiffre 7 dans une liste triée

- Diviser en deux
- if $7 == \text{valeur courante}$: stop
- else if $7 < \text{valeur courante}$, considérer la moitié gauche
- else if $7 > \text{valeur courante}$, considérer la moitié droite

Recherche binaire pour trouver 7



Pourquoi une recherche binaire est-elle efficace ?

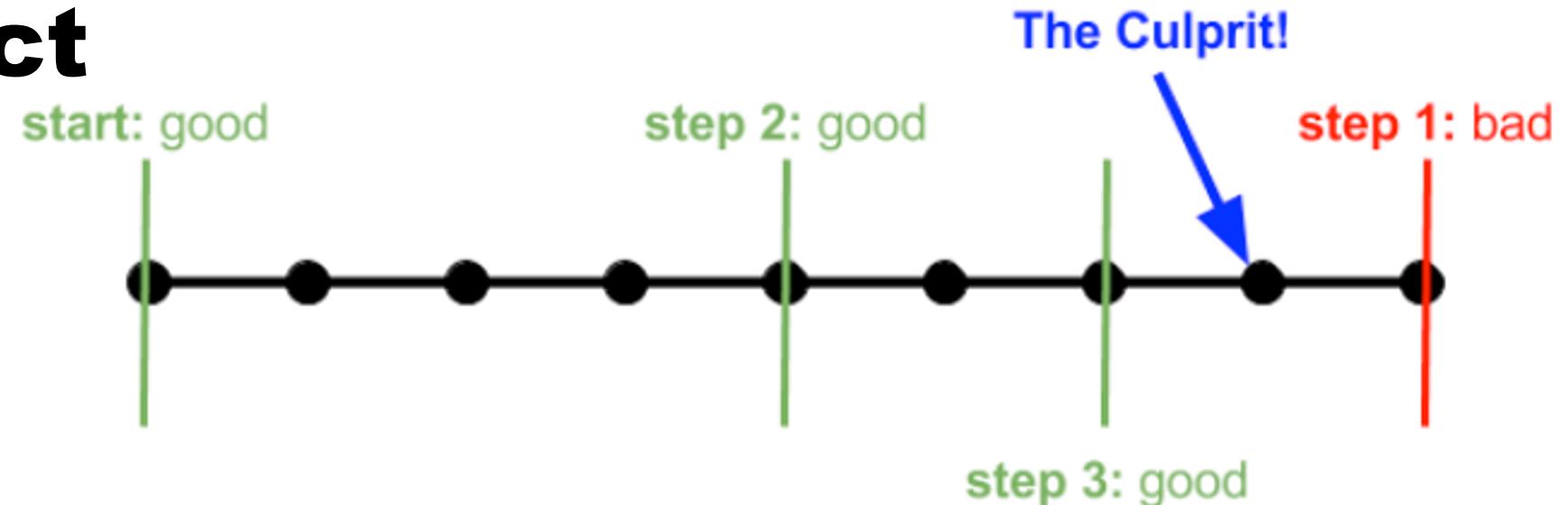
Il réduit l'espace de recherche de moitié à chaque fois

- Dans le pire des cas, vous devez faire des tests $\ln(n)$, $n = \text{nombre de commits}$
- Dans notre exemple, nous devons faire quatre vérifications
- Si nous avions une liste de 1000 dans le pire des cas, nous devrions faire 10 vérifications

Qu'est-ce que cela signifie pour les logiciels ?

- Si nous avons une régression et que nous avons 1000 commits, nous n'avons qu'à faire un maximum de 10 vérifications pour trouver le bug
- Très important lorsque la construction ou le test prend beaucoup de temps à s'exécuter

Git bisect

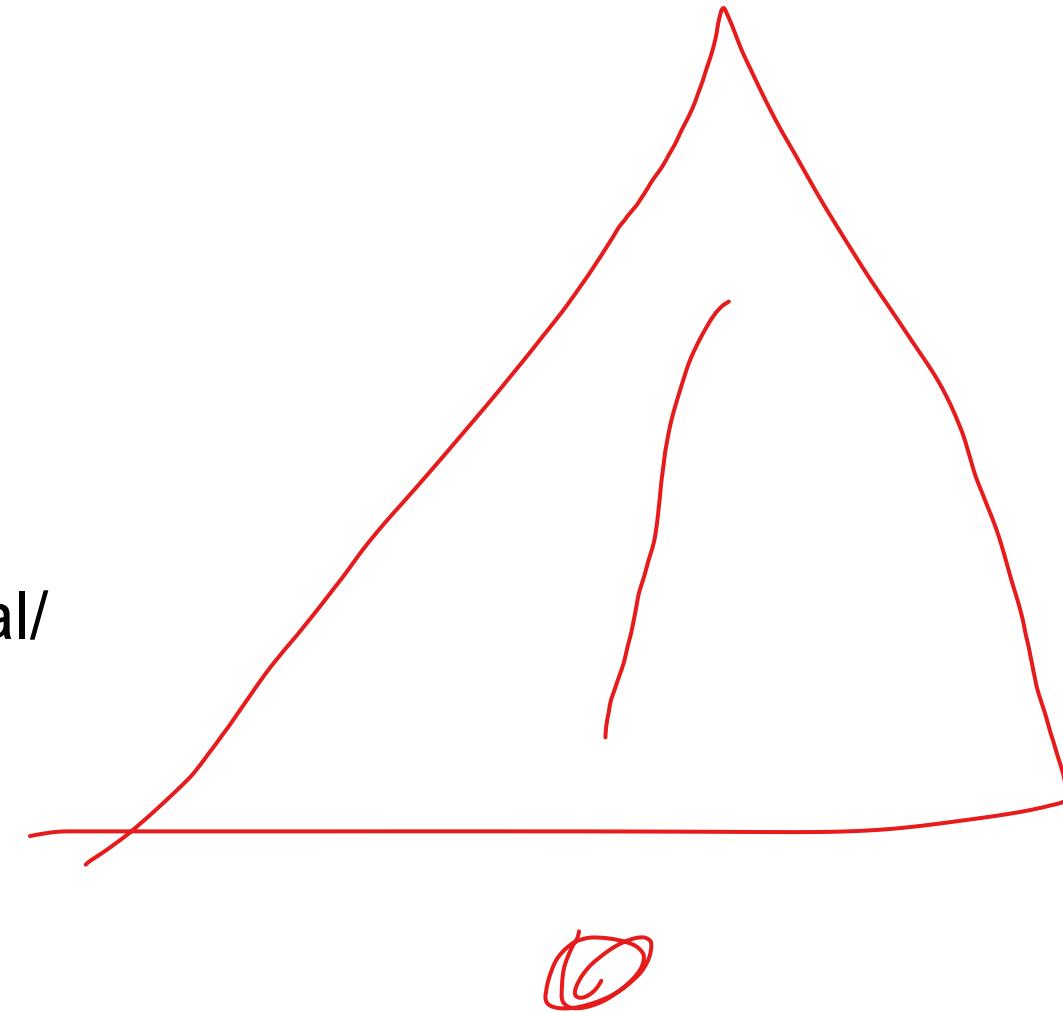


Objectif : trouver le premier commit qui échoue à un test

- Diviser les commits suspects en deux
- Si le test échoue, le bogue est dans la moitié gauche
- Si le test réussit, alors le bogue est dans la moitié droite
- (répéter)

TUTORIEL GIT BISECT

<https://linuxhint.com/git-bisect-tutorial/>



Bisect

- `git bisect start`
 - Dites à git que nous allons commencer à bissecter
- `git bisect bad`
 - lui indique l'emplacement du premier commit dont vous êtes sûr qu'il est mauvais
 - Par défaut c'est `head`
 - Vous pouvez spécifier un commit plus ancien (par exemple, le code que vous venez de publier sur le client)

Dernier bon commit

- Nous devons dire à git la dernière fois que nous sommes sûrs que le code a fonctionné
- Si nous ne savons pas, nous pouvons simplement choisir le premier commit
- Regardons les messages de commit du dépôt
 - git bisect good *****

Test itératifs

- rafraîchir l'espace de travail
- Exécuter les tests
- Si réussi, tapez
 - git bissec good
- En cas d'échec, tapez
 - git bisect bad
- Répétez jusqu'à ce que Git vous indique le premier mauvais et le dernier bon commit
- Sortir de bisect
 - git bisect reset
 - Corrigez le code

Bisect peut être automatiser

- Dépend des valeurs de retour d'un processus
- git bisect run <cmd>

Conclusion

Git bisect utilise une recherche binaire et des tests de régression pour trouver l'introduction d'un bogue

Dans le pire des cas, il regarde $\ln(n)$ commits

Vous exécutez les tests en lui indiquant si le commit actuel est bon ou mauvais

Au final, vous savez quel commit a introduit le bug !

Attributions

Matériel tiré de recherche de:

- Prof Peter Rigby
- Prof Weiyi Shang

Tests d'intégration et regression

1) Est-il possible d'utiliser des outils de tests unitaires pour faire des tests d'intégration?

Vrai, car les outils de tests unitaires sont utilisés pour être testés à l'intérieur des fonctions spécifiques et on a accès au code source, on va vérifier exactement telle ligne de code/branche (if) fait exactement ce qu'on pensait ou pas

LOG3430 Méthodes de test et de validation du logiciel

2) Les tests d'intégration sont en boîte blanche ou en boîte noire?

Boîte noire, car j'ai pas accès au code src qui implémente la fonctionnalité pour la tester

Intégration ascendante (Bottom-Up Integration) ↗

Avantages :

- 1) Le code peut ne pas être complet ou entièrement fonctionnel, c-à-d qu'on commence avec le bas niveau, on commence avec des composantes qui n'ont pas de connexions avec le reste du logiciel
- 2) Permet de tester la performance de nos sous-modules. Avantage principal de Bottom-Up est que vu qu'on commence avec les composantes de bas niveau ont un effet élevée sur la performance. Vu qu'ils sont au + bas niveau, tout ce qui est en haut, tout ce qui utilise cette logique dépend du bas. Donc, si on a une basse performance à notre bas niveau, on va avoir une basse performance à notre haut niveau aussi.
- 3) Pas de STUBS(+dur à faire), mais de DRIVERS!
- 4) Plus facile de cibler les erreurs que l'approche Big-Bang. Moins de risques d'erreur globale, on cible l'erreur, on sait ce que l'on teste, contrairement à big-bang
- 5) Utile pour les systèmes en temps réel qui ont des contraintes de performance

Désavantages:

- 1) Nécessite des DRIVERS
- 2) Lorsqu'on cible les erreurs, processus est + long, on doit y aller étape par étape pour isoler l'erreur contrairement à l'approche incrémentale
- 3) Aucune notion du squelette du programme, défauts de conception. Lié à l'architecture! Vu qu'on est obligé de commencer les tests par le bas, on va être obligé de commencer le développement par le bas aussi. Donc, on ne peut pas créer le système sachant quelle va être l'architecture finale. L'architecture peut être un peu bizarre à la fin. Il va être tard pour détecter les erreurs de conception.

Intégration descendante (Top-Down Integration) ↗

Avantages :

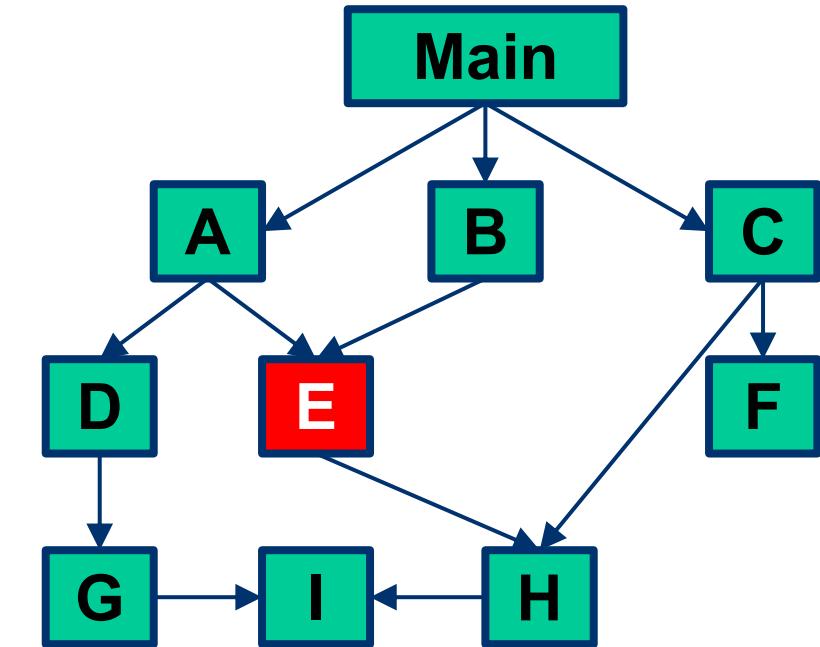
- 1) Prototypes rapides! Pratique à l'approche agile ou on n'a pas une architecture concue.
On peut donner un prototype au client et le client va vérifier si le logiciel correspond aux attentes. Si ça correspond pas aux attentes on modifie dès le départ la conception que le client veut.
- 2) Nécessite de peu(pas) de Drivers. La fonctionnalité du haut appelle une fonctionnalité du bas et celle du bas appelle une plus bas, on arrive jusqu'en bas du programme, on n'a pas besoin de Driver. Fonctionnalité de bas niveau va être appelée par celle de haut niveau
- 3) Plus facile de cibler les erreurs que l'approche Big-Bang. Moins de risques d'erreur globale, on cible l'erreur, on sait ce que l'on teste, contrairement à big-bang

Désavantages:

- 1) Nécessite des STUBS!
- 2) Permet de tester la performance de nos sous-modules très tard

Exercice: Integration

- Déterminer l'ordre dans lequel les modules à droite doivent être développés et testés
- Bottom-up
- Top-down
- Criticité :  



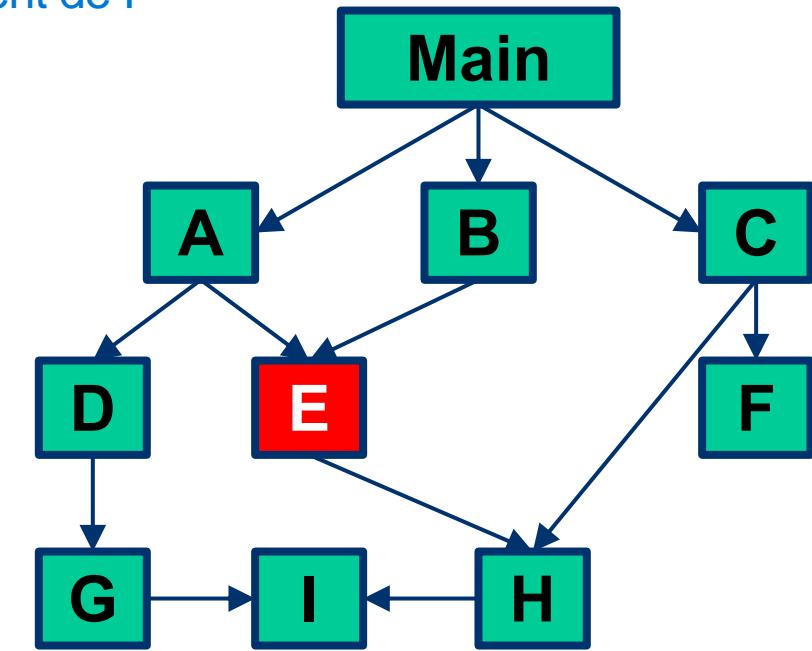
- Expliquez pourquoi votre ordre est l'ordre préférée en fonction de
 - Criticité des modules
 - Nombre de stubs et de drivers requis
 - Nombre de fois qu'un module est testé

Exercice: Bottom-Up

On commence avec I plutôt que F car il y a plus choses qui appellent I , dépendant de I
 Si F avait été en rouge (critique), on aurait commencé par F

- 1) I
- 2) G+I, H+I
- 3) D+G+I, E+H+I
- ...
 On ne peut tester C, car C dépend de F et on n'a pas encore testé F. Aussi E est + critique que F, donc c'est aussi le chemin le + rapide pour tester E
- 4) A+D+E+G+I+H et B+E+H+I
- 5) F
- 6) C+F+H+I
- Tester C, qui a besoin de F et H. H a besoin de I.
- 7) Main + le reste

- (driver pour I)
 I n'appelle rien, donc je n'ai pas besoin de tester qqc plus bas que I
- (drivers pour G, H)
 (drivers pour D, E)
- (drivers pour A et B)
- (driver pour F)
- (driver pour C)
- (driver pour Main, ou un tester qui teste le main directement)



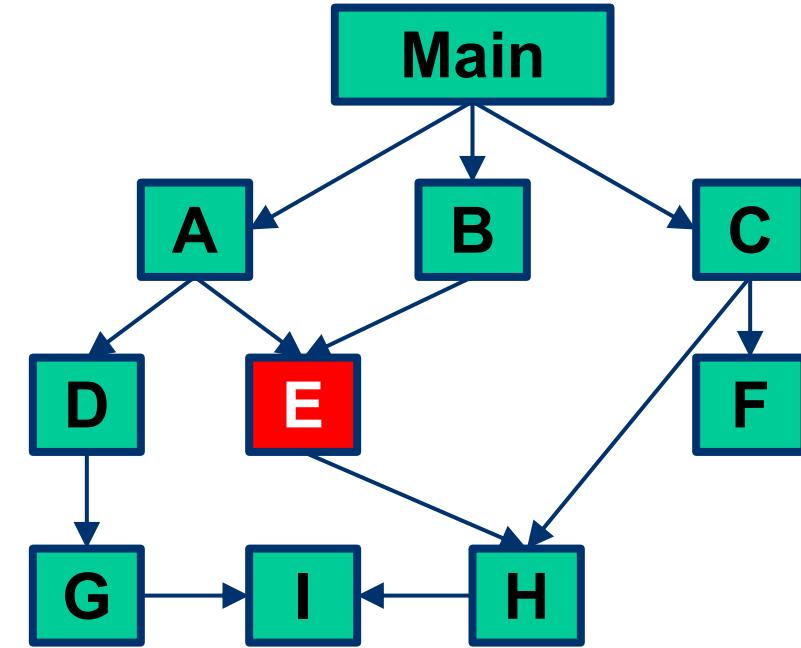
I--> a été testé 9 fois
 H --> testé 6 fois
 G --> testé 4 fois
 E --> testé 4 fois
 D --> testé 3 fois
 ...

Exercice: Top-Down

Main
A+Main
B+Main

À CONTINUER

stubs car A,B,C n'existent pas
(stubs pour A,B,C)
(stubs pour B,C,D,E)
(stubs pour A,C,E)



Exercice: Risk-Driven

Tester le + critique en premier (ici E)

1) E (driver pour E, stub pour H)

2) A+E (driver pour A, stub pour H et D) car E dépend de H et A dépend de D

- Pk on irait vers le haut? Pk on irait vers le bas?

*Si on va vers le bas, c'est prc on veut tester la performance de notre système critique. Aussi car on veut éliminer les STUBS le + rapidement possible(car difficile à faire)

*Si on va vers le haut, c'est prc on veut voir cmt E fonctionne avec l'interface utilisateur et dans le concept global du système. Aussi, on veut tester le + de composantes dans ce cas. Donc, ici on a décidé d'aller vers le haut

3) B+E (driver pour B, stub pour H)

4) Main+A+B+E (driver pour Main/testeur, stub pour H ,C, D)

5) Main+A+B+E+H (driver pour Main/testeur, stub pour I ,C, D)

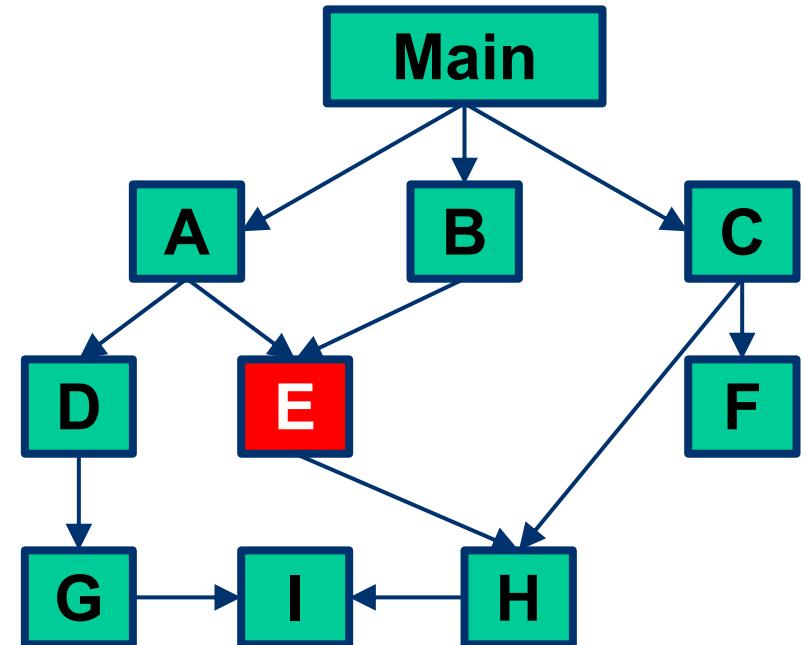
6) Main+A+B+E+H+I (driver pour Main/testeur, stub pour C, D)

7) Main+A+B+E+H+I+C (driver pour Main/testeur, stub pour F, D) on a choisi entre C et D par ordre alphabétique

8) Main+A+B+E+H+I+C+F (driver pour Main/testeur, stub pour D)

9) Main+A+B+E+H+I+C+F+D (driver pour Main/testeur, stub pour G)

10) Tout E --> testé 10 fois ici (et en 1er) vs Bottom-Up testé 4 fois



Lequel tester mtn? D,H ou C et pk?

- H, car on est dans un Risk-Driven, donc on devrait tester E qui est le + critique

on pourrait avoir un driver pour I si on dit que G et H utilisent la fonctionnalité de I différemment. Si on dit que G et H utilisent la mm fonctionnalité pareillement, on n'a pas besoin de driver de I car on utilise déjà I+H

Ordre d'intégration

- Plus de techniques!

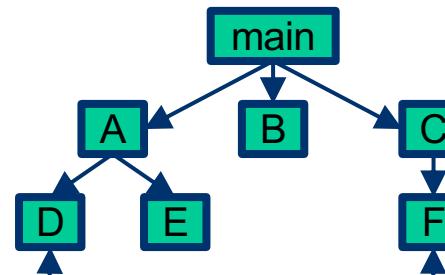
Ordre d'intégration

Il n'est pas conseillé d'exécuter une intégration de classe big-bang. L'intégration devrait être faite par étape, mais cela mène à des stubs.

- Il n'est pas toujours faisable de construire un stub qui est plus simple que le code réel (*Surtout en OO, qui favorise des méthodes très petites*)
- La génération de stub ne peut pas être automatisée (*cela requiert une compréhension des sémantiques de fonctions simulées*)
- Le potentiel de fautes pour certains stubs peut être le même ou pire que celle de la fonction réelle
- Minimiser le nombre de stubs développés génère d'importantes économies

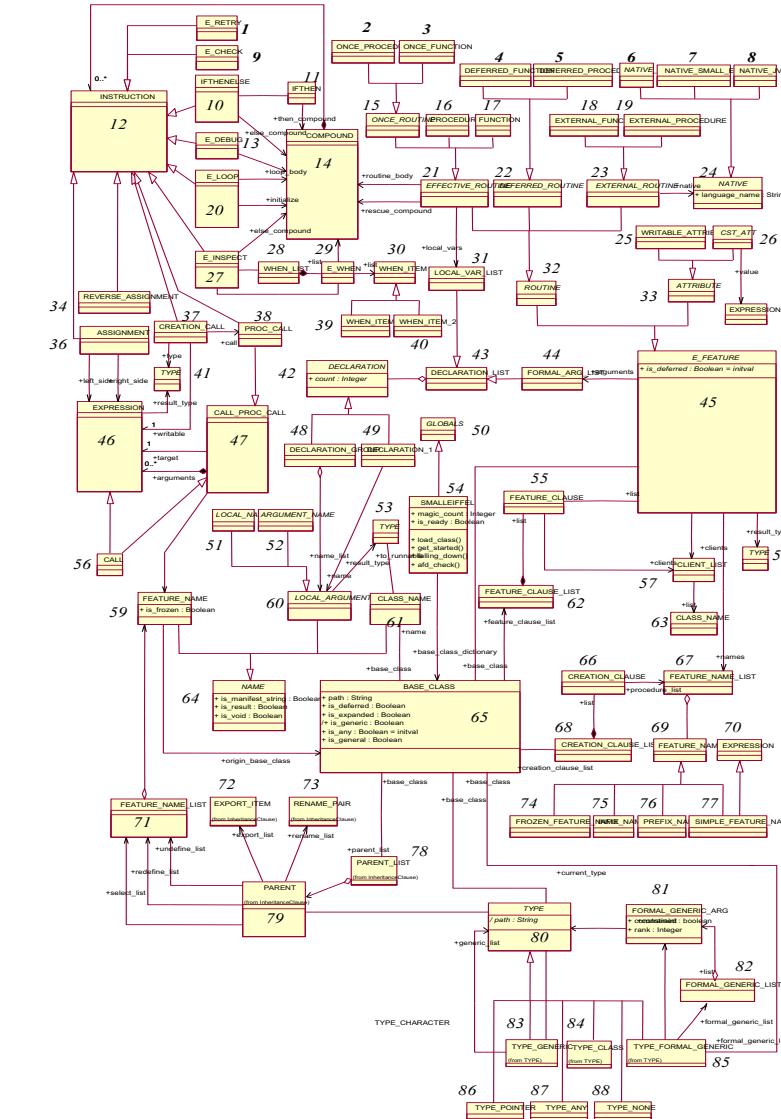
Ordre d'intégration

En général, les graphes de dépendance des classes ne forment pas des hiérarchies simples



Mais des réseaux complexes avec:

- Forte Connectivité
- Interdépendances cycliques



Test ORD de Kung et al.

Vise à produire un ordre partiel des niveaux de tests basé sur les diagrammes de classes

Les classes en test à un certain niveau devraient dépendre seulement des classes testées précédemment

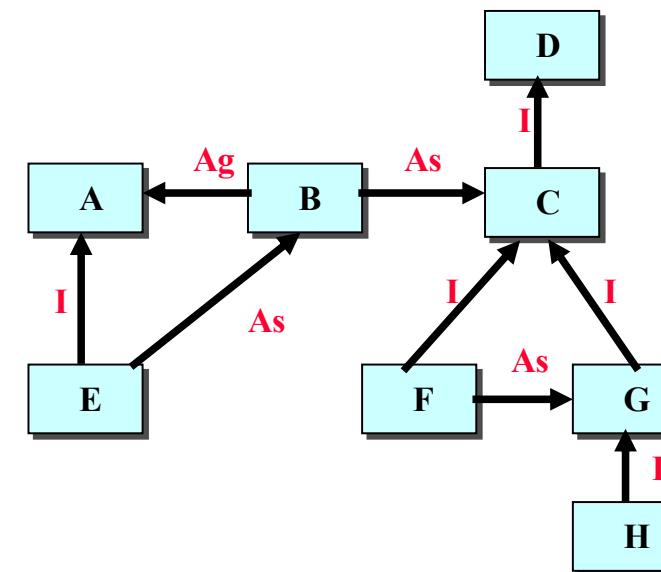
En utilisant des classes précédemment testées dans nos cas de test, nous n'avons pas besoin de stubs pour les remplacer

L'information de dépendance de classe peut soit provenir de l'ingénierie inverse (*reverse engineering*) du code, soit de la documentation de conception (ex. : diagrammes UML).

Test ORD de Kung et al.

Basé sur le diagramme de relation objet (Object Relation Diagram, ORD) → version simplifiée du diagramme de classe qui se concentre sur les relations entre classes.

- L'ORD pour un programme P est un digraphe (graphe orienté) à branches étiquetées où les nœuds représentent les classes de P, et les branches représentent les relations d'héritage, d'agrégation et d'association. ORD ne fait aucune distinction entre composition et agrégation



I: Héritage (hérite des propriétés d'une autre classe)

Ag: Agrégation (une classe contient des instances d'une autre classe)

As: Association (quand on appelle une méthode ex: static d'une autre classe, juste un appel à la méthode d'une classe et pas à la classe complète)

Test ORD de Kung et al.

Pour chaque paire de classes C1 et C2

Une branche est étiquetée « I » de C1 à C2

- si C1 est un enfant de C2 (héritage)

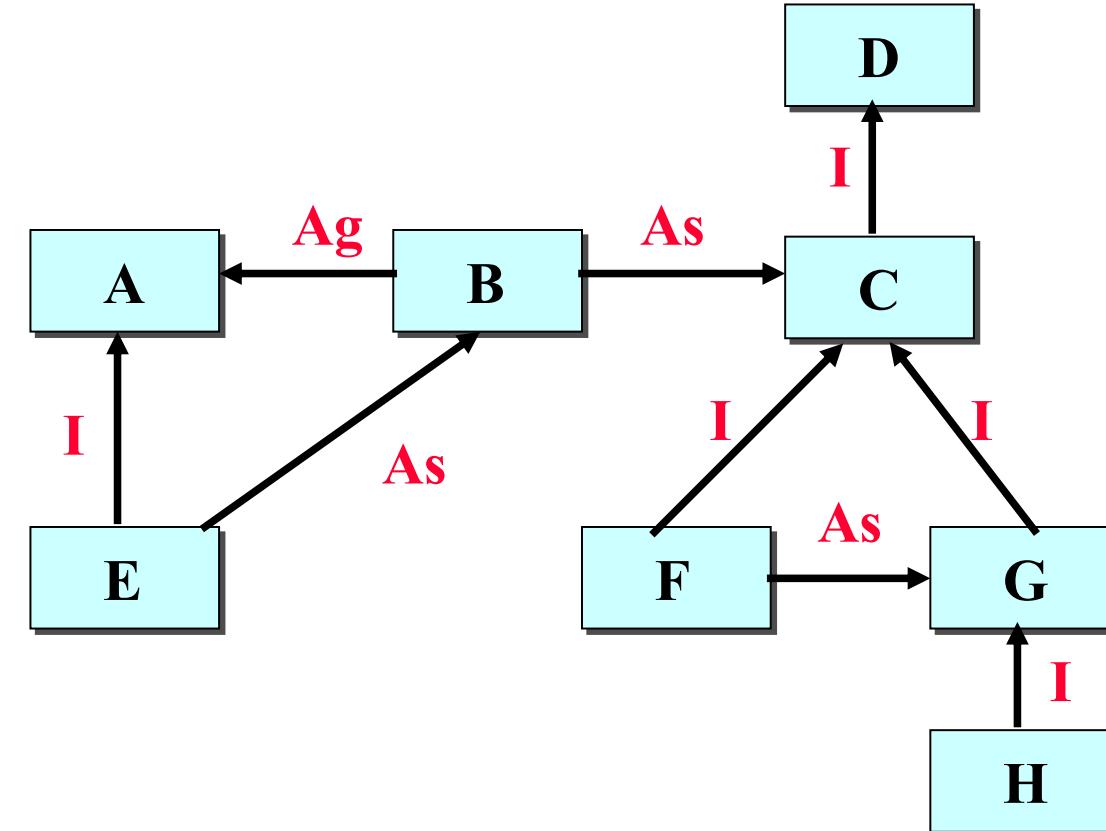
Une branche est étiquetée « Ag » de C1 à C2

- si C1 contient une ou plusieurs instances de C2 (agrégation).

Une branche est étiquetée « As » de C1 à C2

- si C1 dépend de C2. Ex. : C1 appelle une méthode de C2, utilise des données de C2, ou a des méthodes qui prennent en paramètre C2 (association)

Exemple d'ORD



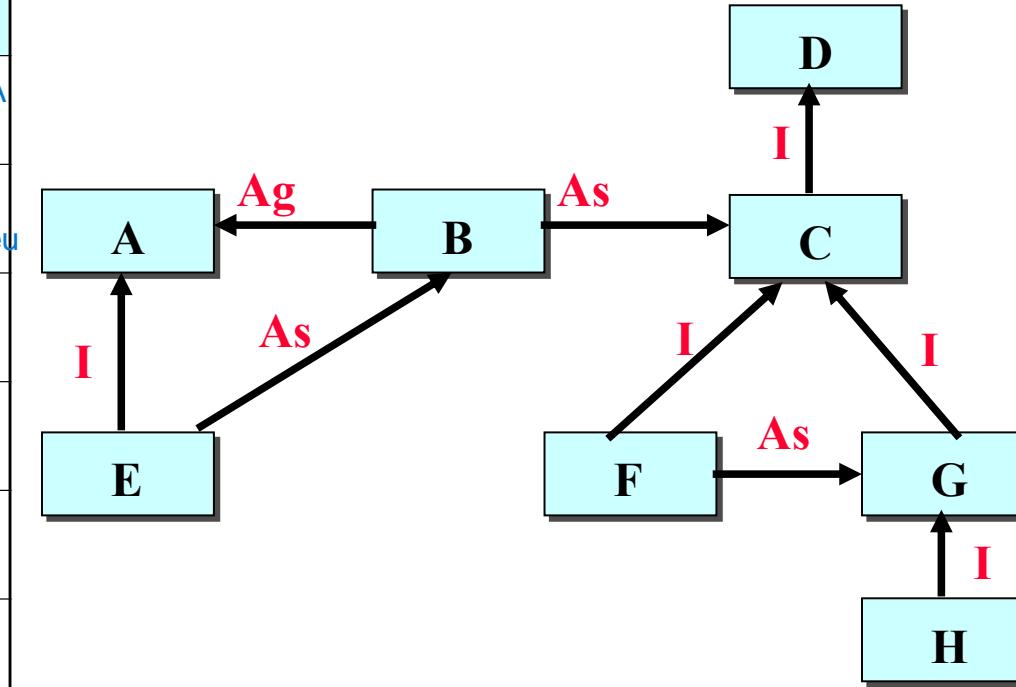
Classe coupe-feu (class firewall, CFW)

Soit une classe X : à quel point cette classe est isolée du reste du système?

- **CFW(X)** pour une classe X : ensemble de classes qui dépendent de la classe X.
 - Ensemble de classes qui *peuvent* être affectées par un changement à la classe X.
 - Donc, ensemble de classes qui devraient être testées de nouveau quand la classe X est changée → tests de régression.
- Supposons que l'ORD n'est pas modifié par le changement fait à la classe X.
 - Donc pas de changement à l'héritage, aux agrégations et associations.
- Un test adéquat de CFW(X) doit inclure :
 - les classes enfants de X (relation I),
 - les classes agrégées de X (relation Ag),
 - les classes associées avec X (relation As).

Exemple coupe-feu

Classe X	CFW(X)
A	B, E <small>E est un enfant de A B est une Ag de A</small>
B	E <small>J'ai besoin de E pour tester B pour le coupe-feu</small>
C	B, E, F, G, H
D	B, C, E, F, G, H
E	
F	
G	F, H
H	



On peut donc voir qu'il est possible d'intégrer les classes dans un certain ordre afin d'éviter d'écrire des stubs.

Ordre (partiel) de test

- Fournit au testeur une feuille de route pour mener un test d'intégration.
- L'ordre désirable est celui qui minimise le nombre de stubs ou l'effort de création des stubs.
- Teste les classes indépendantes en premier, puis teste les classes dépendantes en se basant sur leurs relations (I, Ag ou As).
 - Les parents sont donc toujours testés avant les enfants.

Ordre d'intégration pour des ORDs acycliques

Peut générer un ordre de test qui assure que X soit testé *avant* toutes les classes de $CFW(X)$

- Les stubs ne sont pas requis
- Plusieurs classes peuvent se trouver au même niveau → ordre *partiel*

Exemple de test de régression : le code change dans les classes A et D :

$$CFW(A, D) = \{B, C, E, F, G, H\}$$

Donc, *toutes* les classes doivent être testées de nouveau.

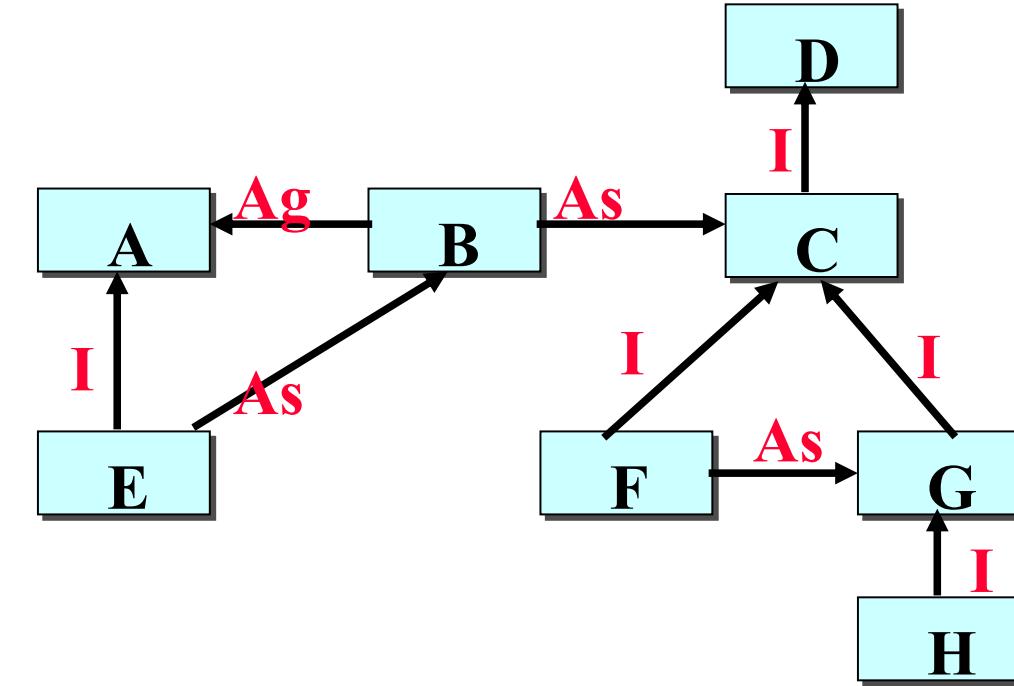
En terme de théorie des graphes, la méthode (de Kung et al.) revient à faire un tri topologique.

i.e. visiter chaque sommet avant ses successeurs.

Exemple : test si on modifie A et D

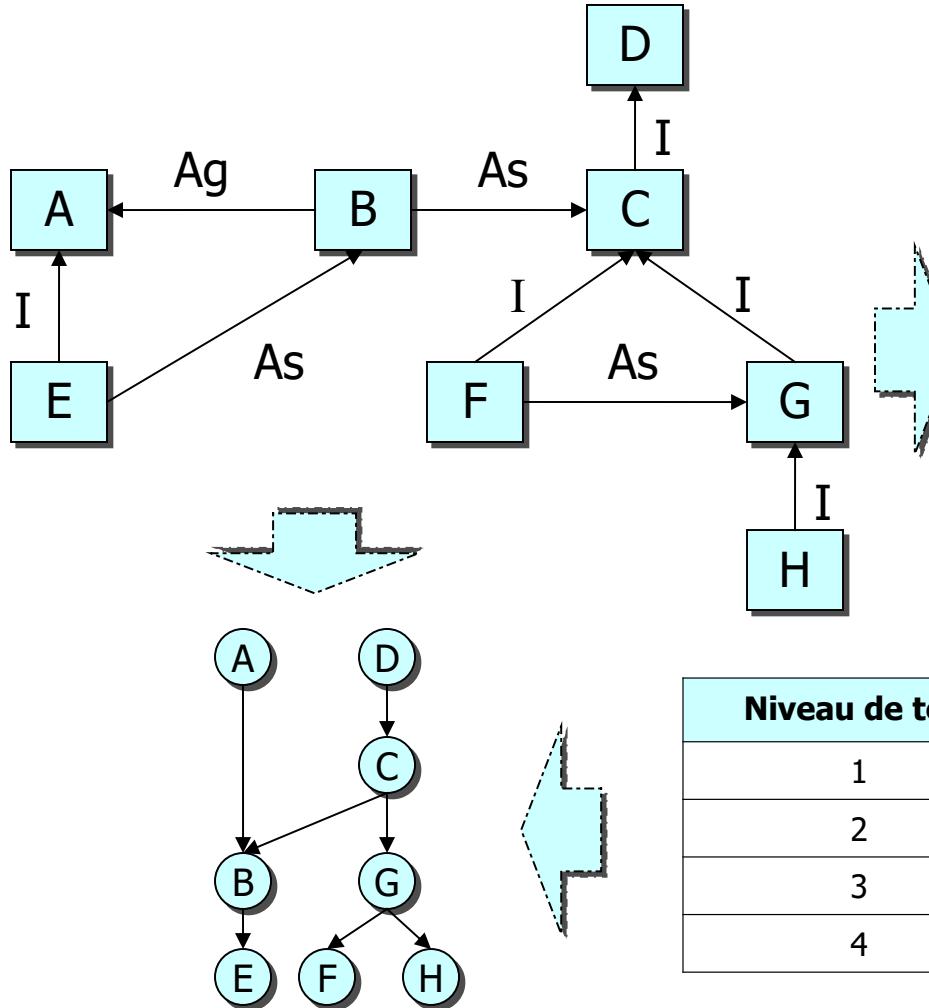
Tester les systèmes qui ont le + de dépendances (flèches entrantes) en premier

Ordre (partiel) de test	Classe(s)
1	A, D <small>A et D au sommet car tout pointe vers A et D. Ils ne pointent vers rien</small>
2	C
3	B, G
4	E, F, H



Le test s'exécute dans l'ordre : On fait le niveau 1 en premier (classes indépendantes) et on descend par la suite.

Étapes



Classe X	CFW(X)
A	B, E
B	E
C	B, E, F, G, H
D	B, C, E, F, G, H
E	
F	
G	F, H
H	

Niveau de test	Classe(s)
1	A, D
2	C
3	B, G
4	E, F, H

Les ORDs cycliques

En pratique, les ORDs sont souvent cycliques.

- Cela est dû aux besoins de performance, à une mauvaise conception, à des changements mal faits ...
- La méthode de Kung et al. (tri topologique) ne peut pas être appliquée.

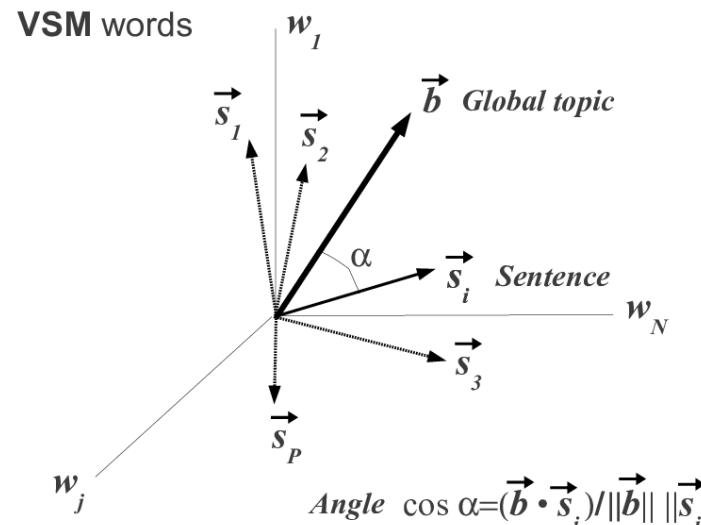
Solution : Cycle interrompu : identifier et éliminer une branche du *cluster* et répéter ceci jusqu'à ce que le graphe (dans le *cluster*) devienne acyclique.

- Quelle relations (branches) doit-on éliminer ?
 - *Associations*
 - En OO classique (ex. : C++, Java), tous les cycles dans le graphe orienté contiennent toujours au moins une branche pour une association.
- Chaque suppression d'association résulte en au moins un stub.

Regression

Approche VSM (Vector Space Model)

- Que fait-on si tout ce qu'on a c'est une grosse pile de tests unitaires mélangés ?
 - Ex.: pas d'étiquetage, pas d'indication claire sur ce qui est testé dans chaque test ...
- Il existe une application de l'approche VSM (*vector space model*, modèle vectoriel) permettant de mesure la différence entre deux textes.
 - Après tout le code source est un texte, et les tests unitaires sont un autre texte.



Note : il existe d'autres approches, mais l'approche VSM est la plus simple à comprendre et implémenter. Ex.:

- Latent Semantic Analysis,
- Latent Dirichlet Allocation,
- Okapi BM25F,
- etc.

Approche VSM (*Vector Space Model*)

- Soit un dictionnaire de termes qui définissent les différentes composantes de notre logiciel.
- Prenons un espace de t dimensions orthogonales qui devient notre espace vectoriel.
- Analysons notre code source :
 - À chaque composante sera associée un vecteur, formé des termes du dictionnaire et de leur fréquence.
- Analysons nos cas de tests :
 - À chaque cas de tests sera associé un vecteur, formé des termes du dictionnaire et de leur fréquence.
- Plus un vecteur-composante sera proche d'un vecteur-test, plus la probabilité que le test est lié à cette composante est élevée.

Approche VSM (*Vector Space Model*)

- Soit le dictionnaire de $t=3$ termes suivants :
 - Dictionnaire = {copy, file, print}.
- Soit la composante Q :
 - Le code de la composante Q possède deux fois le terme « print » et aucun des deux autres.
 - $Q = 0T1 + 0T2 + 2T3$.
- Soit le cas de tests D1 :
 - Le code du cas de tests D1 possède deux fois le terme « copy », trois fois le terme « file » et cinq fois le terme « print ».
 - $D1 = 2T1 + 3T2 + 5T3$.
- Soit le cas de tests D2 :
 - Le code du cas de tests D2 possède trois fois le terme « copy », sept fois le terme « file » et une fois le terme « print ».
 - $D2 = 3T1 + 7T2 + 1T3$.
- Quel cas de test est le plus probablement lié à la composante Q ?

Dictionnaire = {copy, file, print}

Composante {print, print}

Test1 = $2T1 + 3T2 + 5T3$

Test2 = $3T1 + 7T2 + 1T3$

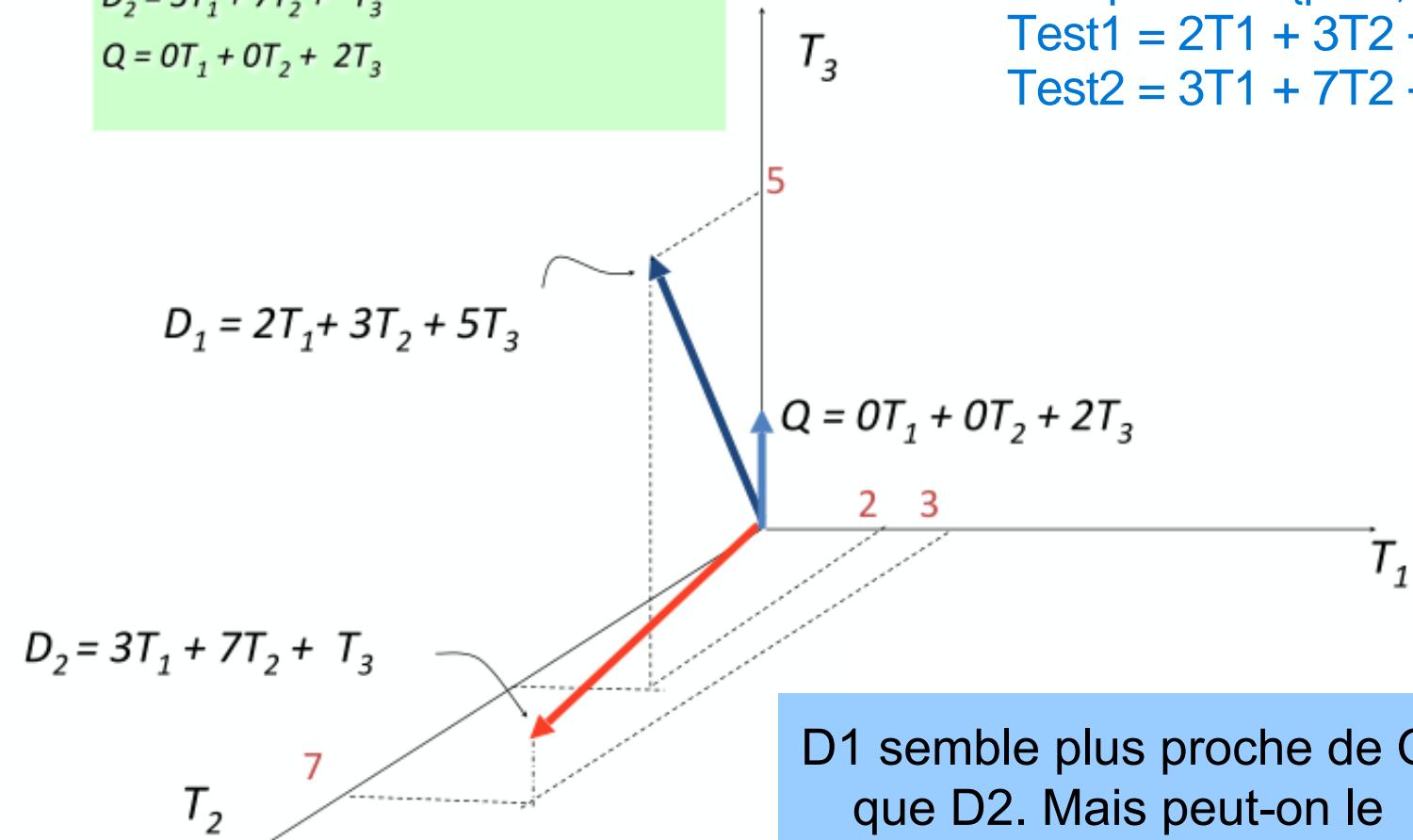
Approche VSM : Représentation graphique

$$D_1 = 2T_1 + 3T_2 + 5T_3$$

$$D_2 = 3T_1 + 7T_2 + T_3$$

$$Q = 0T_1 + 0T_2 + 2T_3$$

Dictionnaire = {copy, file, print}
 Composante {print, print}
 $T_{Test1} = 2T_1 + 3T_2 + 5T_3$
 $T_{Test2} = 3T_1 + 7T_2 + 1T_3$



D1 semble plus proche de Q que D2. Mais peut-on le mesurer automatiquement ?

Approche VSM : Mesure de la distance

- On peut mesurer la distance entre deux vecteurs en mesurant l'angle entre les deux.
 - Plus l'angle est petit, plus les vecteurs sont similaires.
- Pour ce faire il faut :
 - Normaliser les vecteurs pour que les deux soient de taille unitaire.
 - Calculer le cosinus de l'angle entre les deux vecteurs.

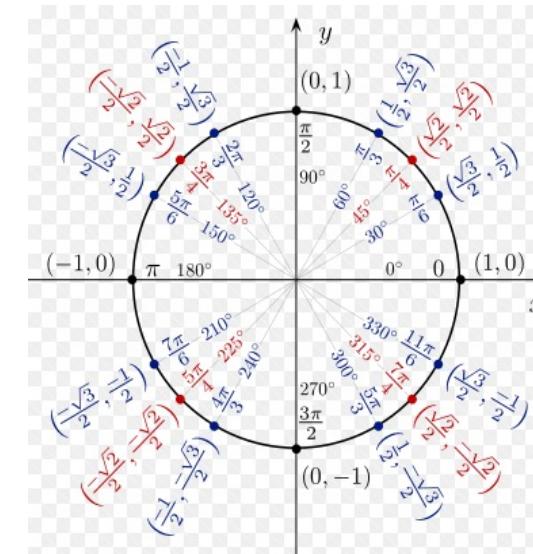
j et k sont les deux vecteurs qu'on veut comparer

$$\cos\theta = \frac{j \cdot k}{\|j\| \|k\|} = \frac{\sum_{i=1}^t (w_{i,j} * w_{i,k})}{\sqrt{\sum_{i=1}^t w_{i,j}^2} * \sqrt{\sum_{i=1}^t w_{i,k}^2}}$$

- Où j correspond à un vecteur et k à l'autre.
- Où w correspond au poids du terme i pour le vecteur j.
- Où w correspond au poids du terme i pour le vecteur k.
- Où t correspond au nombre de termes.

Approche VSM : Retour à l'exemple

- Rappel :
 - Cosinus $180^\circ = -1,000$: même direction, sens opposé.
 - Cosinus $120^\circ = -0,500$.
 - Cosinus $90^\circ = 0,000$: vecteurs orthogonaux. notre test et notre composante n'ont aucun terme en commun, 0 chance qu'ils soient reliés
 - Cosinus $5^\circ = 0,996$.
 - Cosinus $0^\circ = 1,000$: même direction, même sens.
- Dans notre cas, on ne peut pas avoir de nombre de termes négatifs, donc les angles restent limités entre 0° et 90° .



Approche VSM : Retour à l'exemple

- On veut mesurer la distance entre Q et D1, et entre Q et D2, avec :

- $Q = 0T_1 + 0T_2 + 2T_3,$
- $D_1 = 2T_1 + 3T_2 + 5T_3,$
- $D_2 = 3T_1 + 7T_2 + 1T_3.$

$$j^*k / \|j\| \|k\|$$

$$Q * D_1 / \|Q\| \|D_1\| = 0*2 + 0*3 + 2*5 / (\sqrt{0^2+0^2+2^2}) * \sqrt{(2^2+3^2+5^2)} = 10 / 2*\sqrt{38} \\ = 0.81$$

$$Q * D_2 / \|Q\| \|D_2\| = 0*3 + 0*7 + 2*1 / (\sqrt{0^2+0^2+2^2}) * \sqrt{(3^2+7^2+1^2)} = 2 / 2*\sqrt{59} \\ = 0.13$$

On peut conclure que D1 est 81% similaire à notre composante et que D2 est 13% similaire à notre composante.

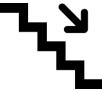
Approche VSM : Limites

Avec l'approche VSM, il n'y a aucune notion d'ordre, slm la fréquence des termes, donc on peut avoir des FP

Technique de filtration plutôt qu'une technique exacte

- C'est une approche qui ne cherche pas à comprendre le texte rédigé
→ basé sur les termes exacts trouvés.
 - Ex.: « A hérite de B » est identique à « B hérite de A » selon l'approche VSM.
 - Ne considère pas les synonymes. Ex.: composante écrit « utilisateur », test écrit « usager ». **le VSM ne sait pas lequel est le bon (user ou utilisateur)**
 - Ne considère pas les homonymes. Ex.: « serveur » d'une architecture client-serveur et « serveur » d'un restaurant.
 - Etc.
- On peut ajuster le calcul du poids pour corriger certains problèmes.
 - Ex.: donner plus d'importance à la présence d'un terme que d'un autre.

Exercices supplémentaires



Intégration descendante (Top-Down Integration)

Exemple:

Supposons que nous ayons une idée pour un système de feuilles de calcul (par exemple, Google Sheets).

- 1) Nous avons besoin d'un composant pour afficher les feuilles de calcul.
- 2) Le composant d'affichage doit obtenir ses données d'un composant qui gère la représentation, stockage et les conversions des données de/des feuille(s). Il doit également s'appuyer sur un composant qui peut faire des calculs, et un autre composant pour gérer des données externes.
- 3) Le composant de représentation/stockage/conversion doit créer des représentations de données à partir de deux composants, l'un stockant les données dans des fichiers binaires et l'autre en XML.
- 4) Le composant qui gère les données externes doit s'appuyer sur une base de données de devises.

Générer l'ordre dans lequel les tests doivent être effectués/séparés pour une approche d'intégration descendante.

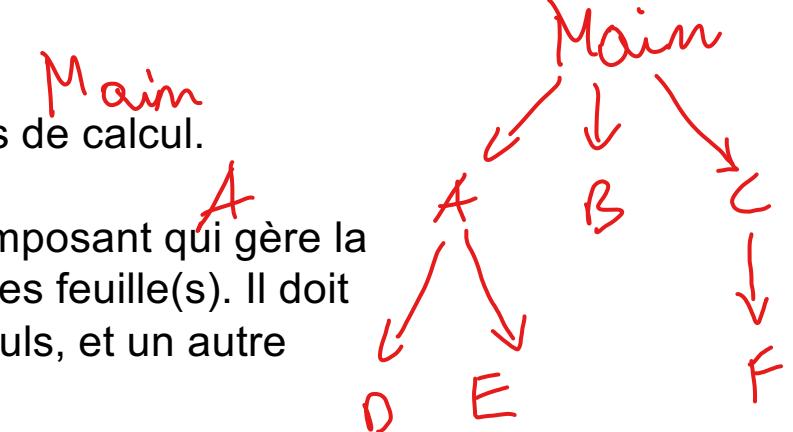
Intégration ascendante (Top-Down Integration)



Exemple:

Supposons que nous ayons une idée pour un système de feuilles de calcul (par exemple, Google Sheets).

- 1) Nous avons besoin d'un composant pour afficher les feuilles de calcul.
- 2) Le composant d'affichage doit obtenir ses données d'un composant qui gère la représentation, stockage et les conversions des données de/des feuille(s). Il doit également s'appuyer sur un composant qui peut faire des calculs, et un autre composant pour gérer des données externes.
- 3) Le composant de représentation/stockage/conversion doit créer des représentations de données à partir de deux composants, l'un stockant les données dans des fichiers binaires et l'autre en XML.
- 4) Le composant qui gère les données externes doit s'appuyer sur une base de données de devises.

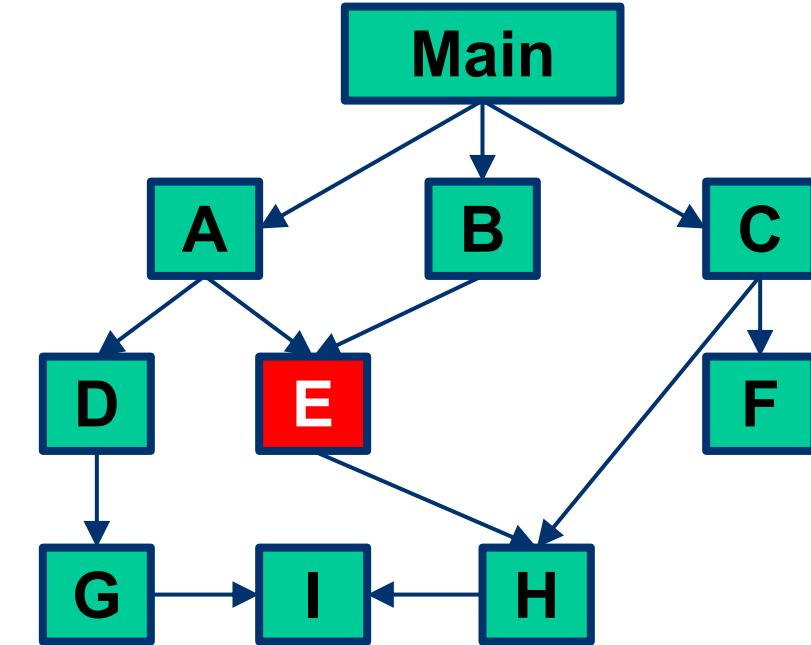


Générer l'ordre dans lequel les tests doivent être effectués/séparés pour une approche d'intégration ascendante.

Exercice: Top-Down

Main
 A+Main
 B+Main
 C+Main
 D+A+Main
 E+A+B+Main
 F+C+Main
 G+D+A+Main
 H+E+C+A+B+Main
 I + le reste

(stubs pour A,B,C)
 (stubs pour B,C,D,E)
 (stubs pour A,C,E)
 (stubs pour A,B,F,H)
 (stubs pour B,C,E,G)
 (stubs pour D,C,H)
 (stubs pour A,B,H)
 (stubs pour B,C,E,I)
 (stubs pour D,F,I)



E est testé similairement par rapport à la stratégie bottom-up.

Seulement des driver pour Main!

Stubs: 12 (Réutiliser si possible!)

Pourrait également utiliser des modules réels au lieu de **stubs bleus**

Fois testé: Main (10), A (6), B (4), C (4), D (3), **E (3 ou 4)**, F (2), G (2), H (2), I (1)

LOG3430

Méthodes de test

Tests d'Intégration

*LOG3430 Méthodes de test et de
validation du logiciel*

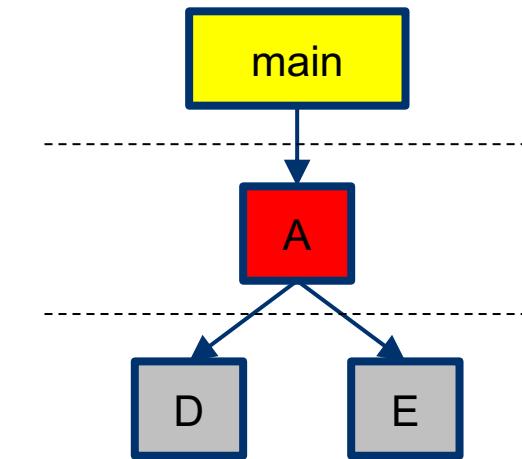
Tests d'intégration

Objectif :

- s'assurer que les composants fonctionnent de manière isolée **et** une fois assemblé



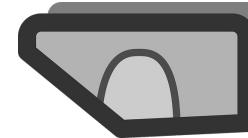
Une fois que les composants ont été testés à un niveau satisfaisant, il faut les combiner avec des systèmes opérationnels



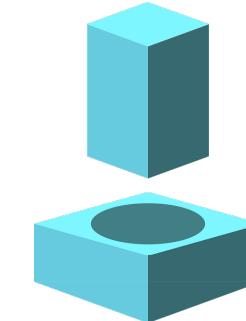
Défauts principalement lié à l'interface

Types de défauts d'interface

Fonctionnalité inadéquate ou incomplète (~20%)



Structure des données (~9 %)



Gestion inadéquate des erreurs (~15 %)



Post-traitement inadéquat (~11 %)

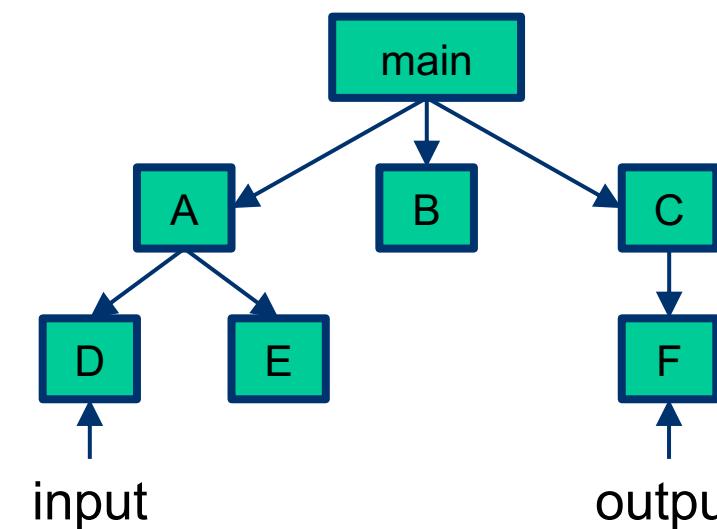


Quelle est la stratégie pour effectuer des tests d'intégration?

Nous allons vérifier/tester comment les composants individuels sont assemblés pour former un programme

- Problème 1 : tester l'interaction des composants
- Problème 2 : déterminer l'ordre optimal d'intégration

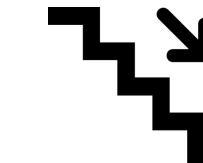
Les tests d'intégration **nécessitent** donc une structure de dépendance des composants



Comment effectuer des tests d'intégration ?

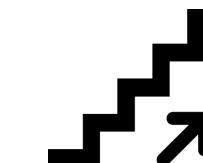
L'approche Big Bang 

Intégration descendante



L'approche Sandwich

Intégration ascendante



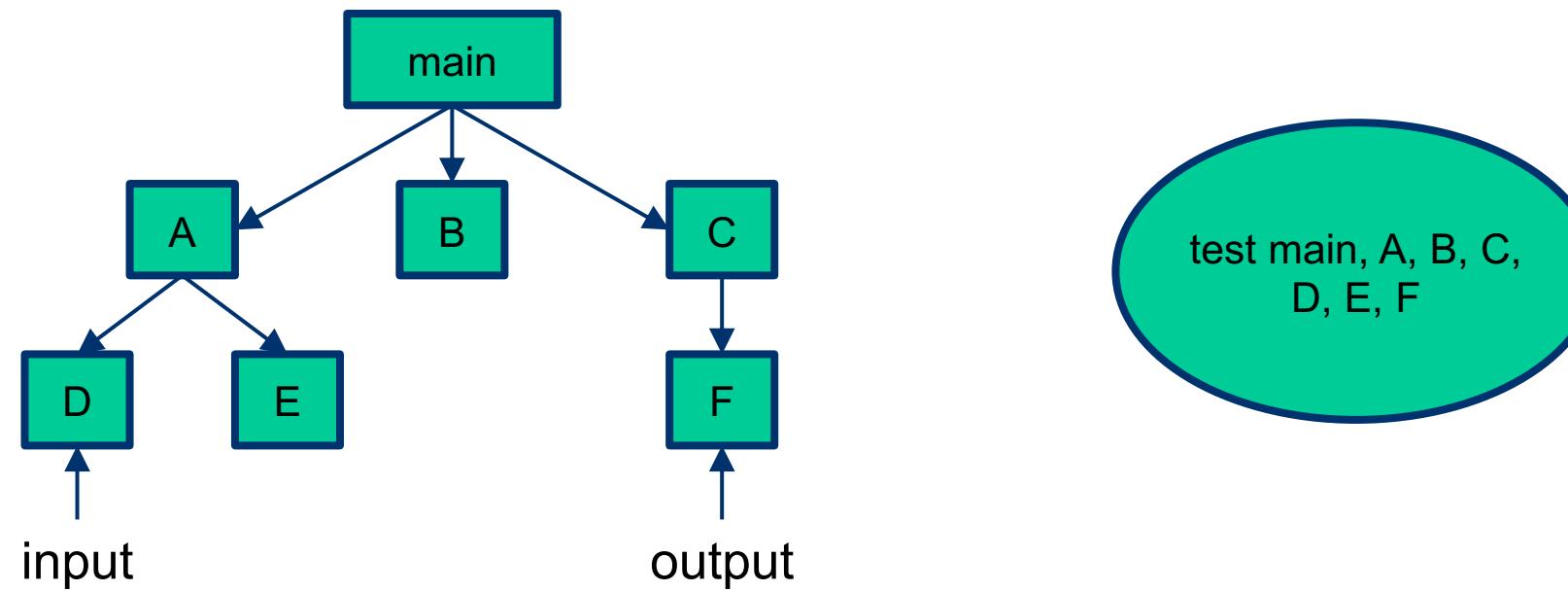
Versions incrémentielles (Incremental Builds)



L'approche Big Bang



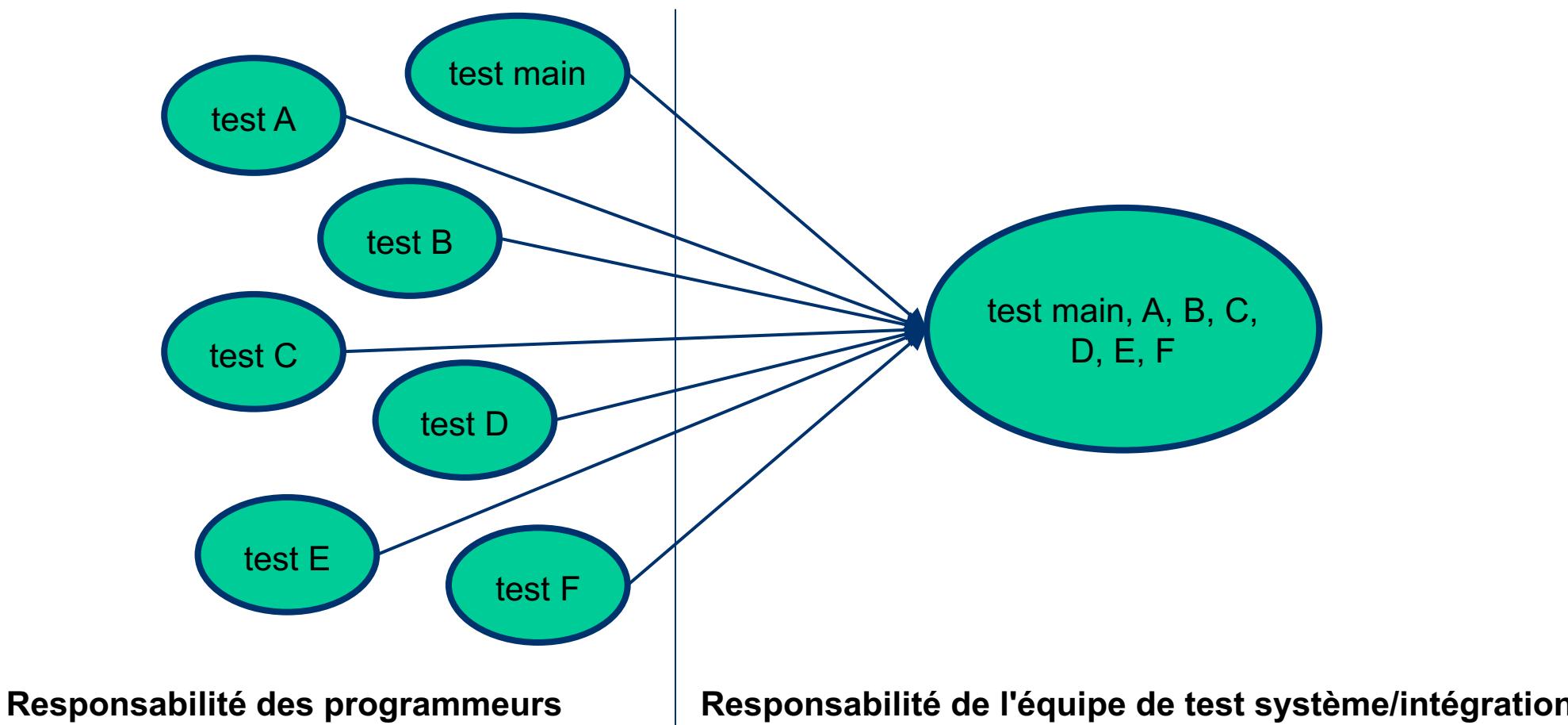
Non incrémentiel (Tous les composants intégrés dans un produit en même temps)

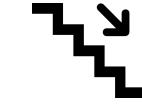


L'approche Big Bang



Suppose que les composants sont initialement testés de manière isolée (c'est la responsabilité des programmeurs)

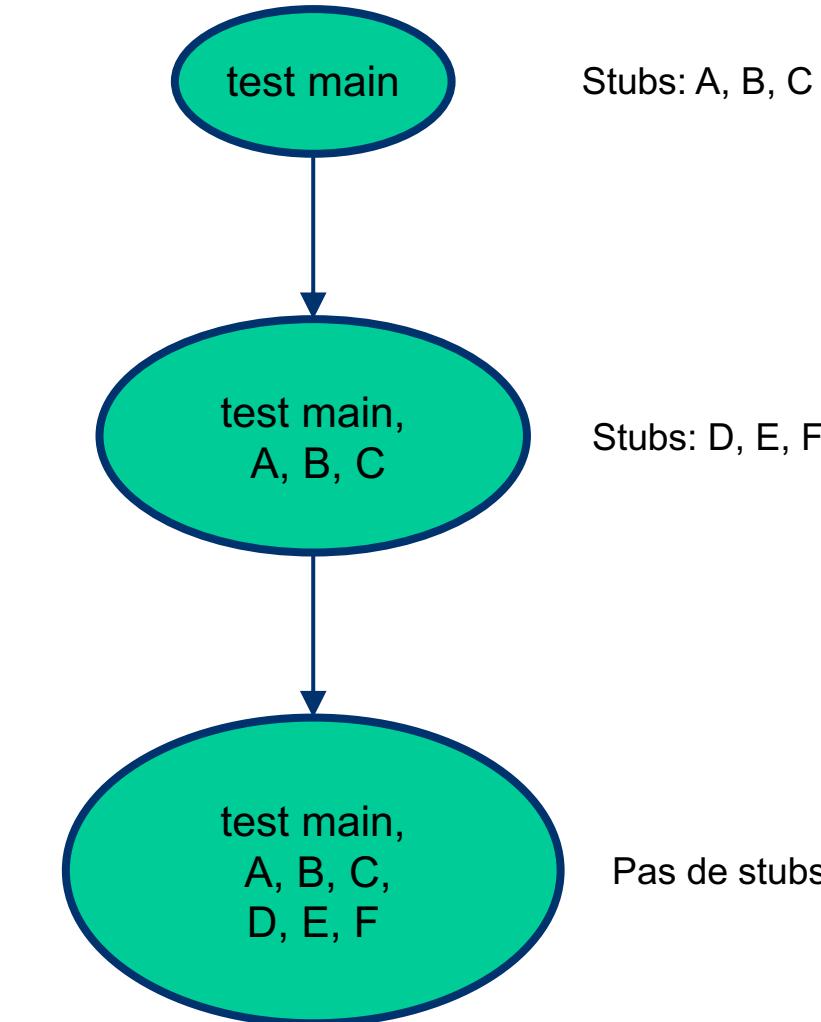
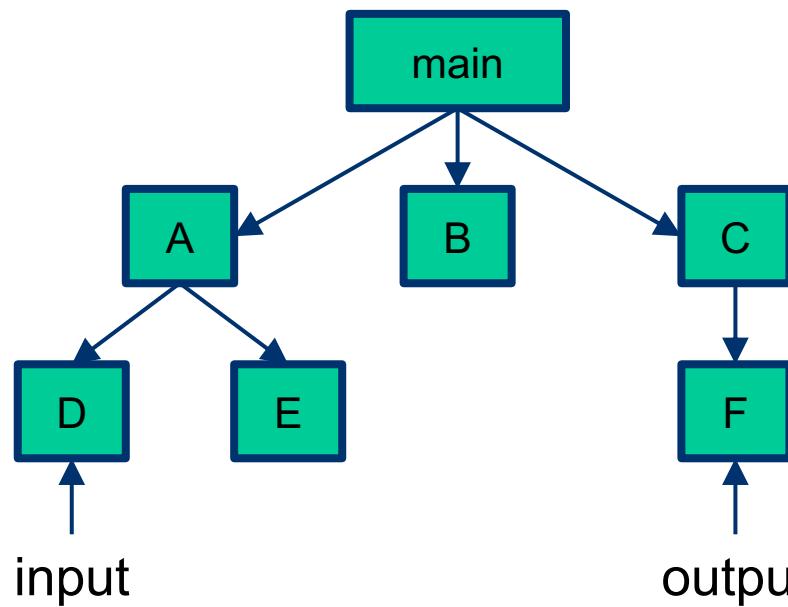




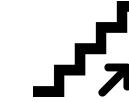
Intégration descendante (Top-Down Integration)

Stratégie incrémentale

Teste les composants de haut niveau, puis les composants appelés jusqu'aux composants de plus bas niveau

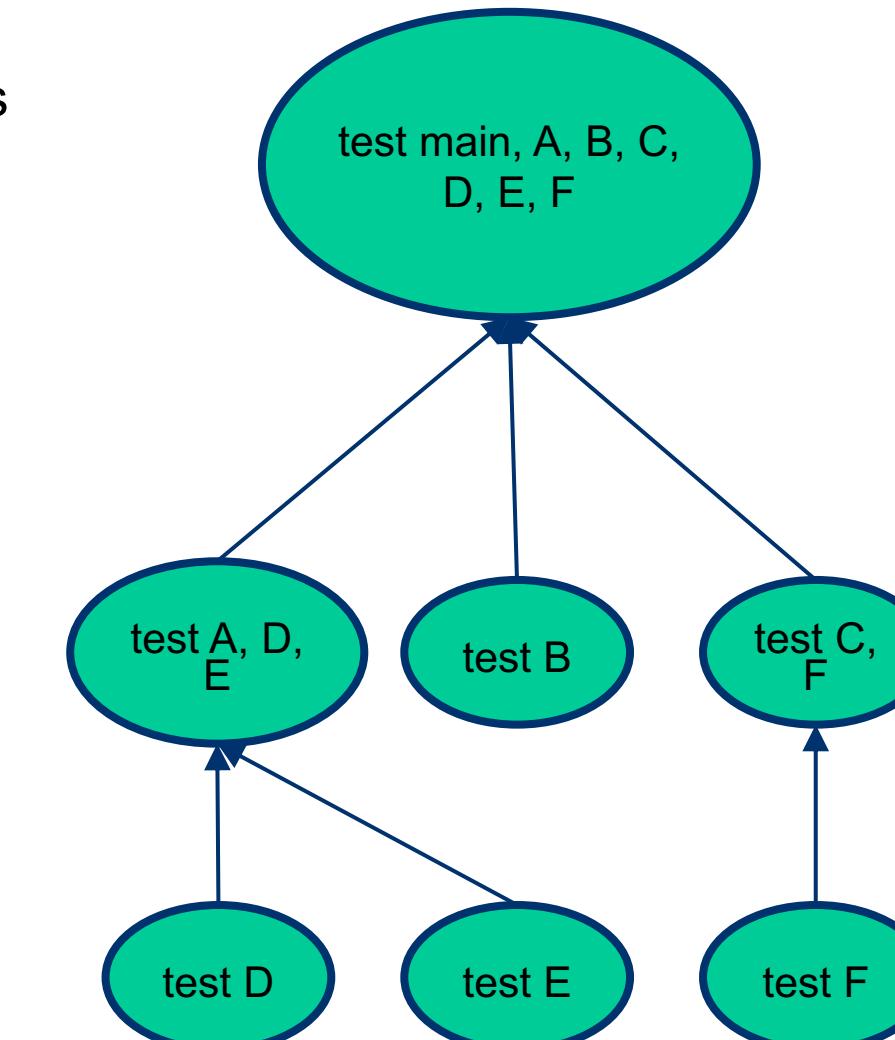
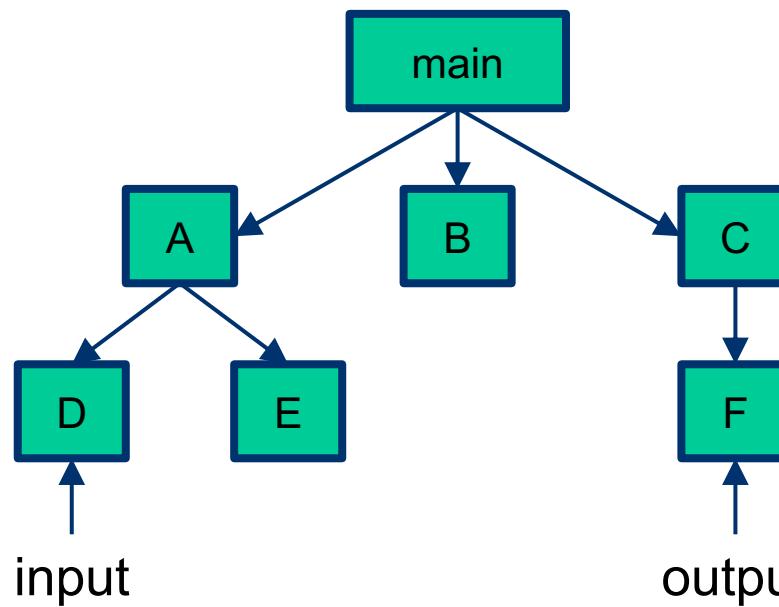


Intégration ascendante (Bottom-Up Integration)



Stratégie incrémentale

Teste les composants de bas niveau, puis les composants qui les utilisent jusqu'aux composants de plus haut niveau

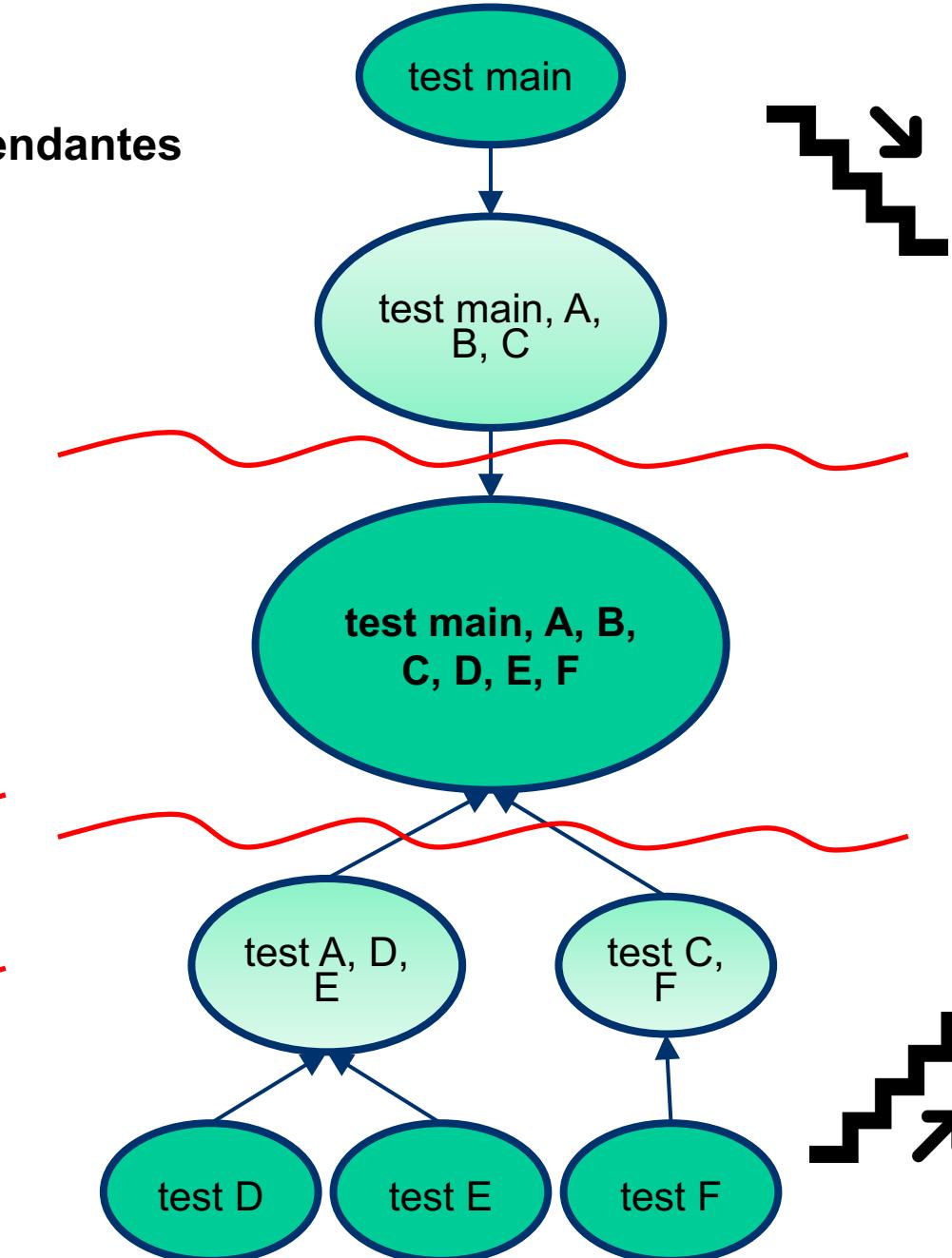
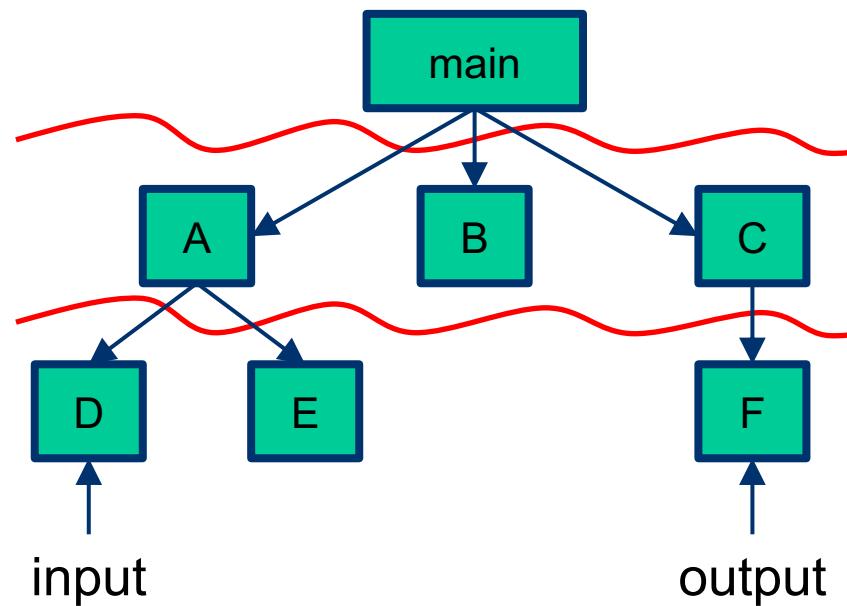


L'approche Sandwich

Combine les approches descendantes et ascendantes

Distingue trois couches à tester :

- Logique (haut) → testé de haut en bas
- Milieu (couche cible)
- Opérationnel (bas) → testé de bas en haut



Stratégies d'intégration

	Ascendant	Descendant	Big-Bang	Sandwich
Intégration Commence	Tôt	Tôt	Tard	Tôt
Délai de livraison du système	Tard	Tôt	Tard	Tôt
Pilotes (drivers)	Oui	Non	Non	Oui
Stubs	Non	Oui	Non	Oui
Capacité de tester les chemins	Facile	Difficile	Facile	Moyenne
Capacité de planifier et contrôler	Facile	Difficile	Facile	Difficile

Versions incrémentielles (Incremental Builds)

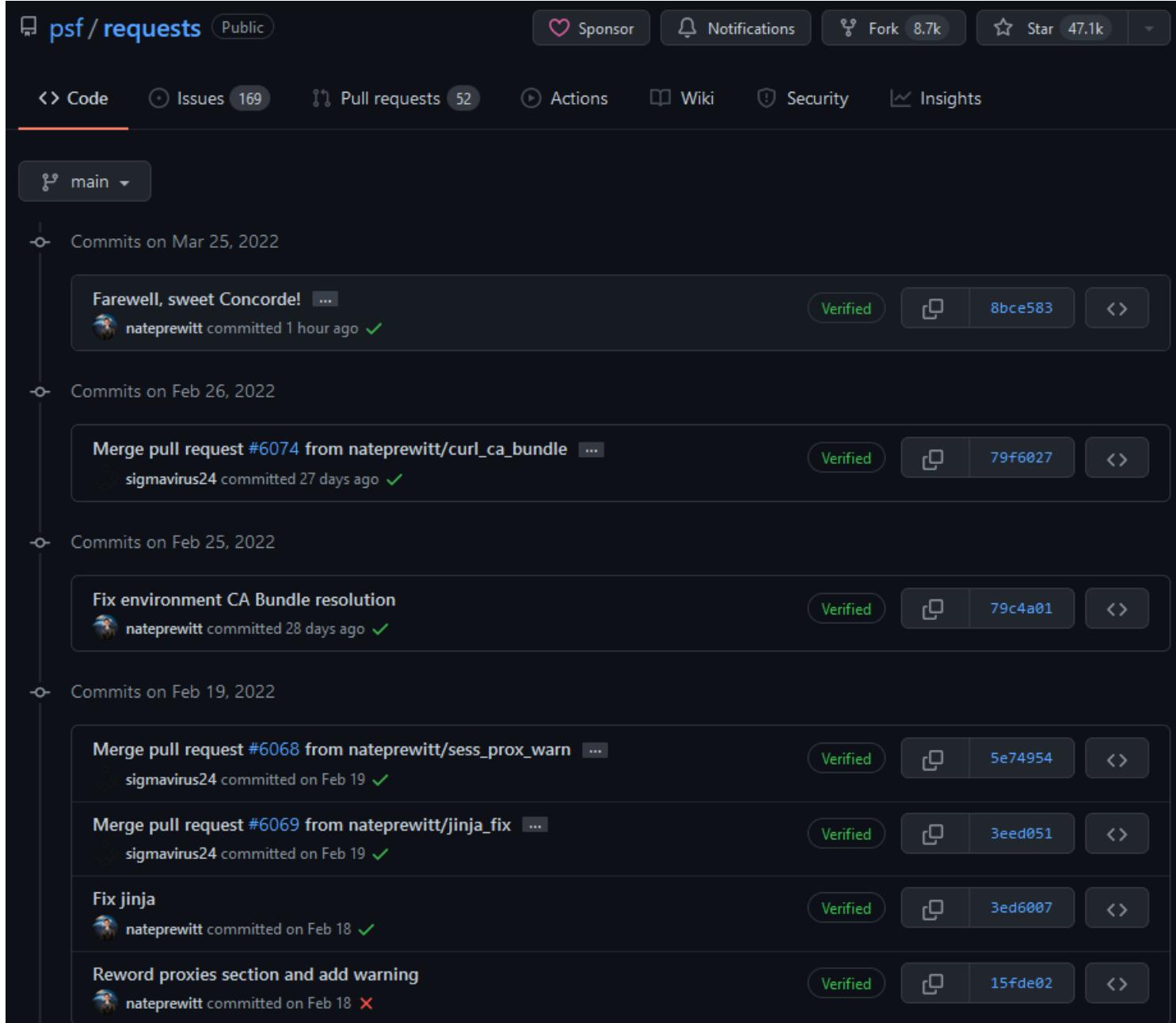
Idée de base:

- (1) Construire une version **de base** initiale «« Version initiale »»
 - Pour fournir des fonctionnalités de bas niveau
 - Peut utiliser diverses techniques : big bang, top-down, ...
- (2) Ajouter de nouveaux modules à la version existante
 - Choisissez les nouveaux modules pour fournir plus de fonctionnalités
- (3) Répétez (2)





Versions incrémentielles (Incremental Builds)



The screenshot shows a GitHub repository named `psf/requests` with a dark theme. The main navigation bar includes links for Code, Issues (169), Pull requests (52), Actions, Wiki, Security, and Insights. The `Code` tab is selected. A dropdown menu for the `main` branch is open, showing commit history. The commits are grouped by date:

- Commits on Mar 25, 2022:**
 - Farewell, sweet Concorde!** (Verified) by `nateprewitt` committed 1 hour ago. Status: ✓
- Commits on Feb 26, 2022:**
 - Merge pull request #6074 from nateprewitt/curl_ca_bundle** (Verified) by `sigmavirus24` committed 27 days ago. Status: ✓
- Commits on Feb 25, 2022:**
 - Fix environment CA Bundle resolution** (Verified) by `nateprewitt` committed 28 days ago. Status: ✓
- Commits on Feb 19, 2022:**
 - Merge pull request #6068 from nateprewitt/sess_prox_warn** (Verified) by `sigmavirus24` committed on Feb 19. Status: ✓
 - Merge pull request #6069 from nateprewitt/jinja_fix** (Verified) by `sigmavirus24` committed on Feb 19. Status: ✓
 - Fix jinja** (Verified) by `nateprewitt` committed on Feb 18. Status: ✓
 - Reword proxies section and add warning** (Verified) by `nateprewitt` committed on Feb 18. Status: ✗

**Vous êtes maintenant prêt à
faire l'activité sur Moodle**

LOG3430

Méthodes de test

Tests de regression

*LOG3430 Méthodes de test et de
validation du logiciel*

Pourquoi faire des tests de régression ?

Tester la partie du logiciel qui vient d'être modifiée.

Les modifications ont tendance à introduire de nouveaux défauts.

- Ex.: Étude empirique sur le code de Firefox (An & Khomh 2015) : 25% des *commits* introduisent des défauts causant des défaillances.
- Ex.: Étude empirique sur le d'Eclipse (2005) : 25% des corrections sur des *commits* défectueux introduisent d'autres défauts ...

Réparer les défauts... ?

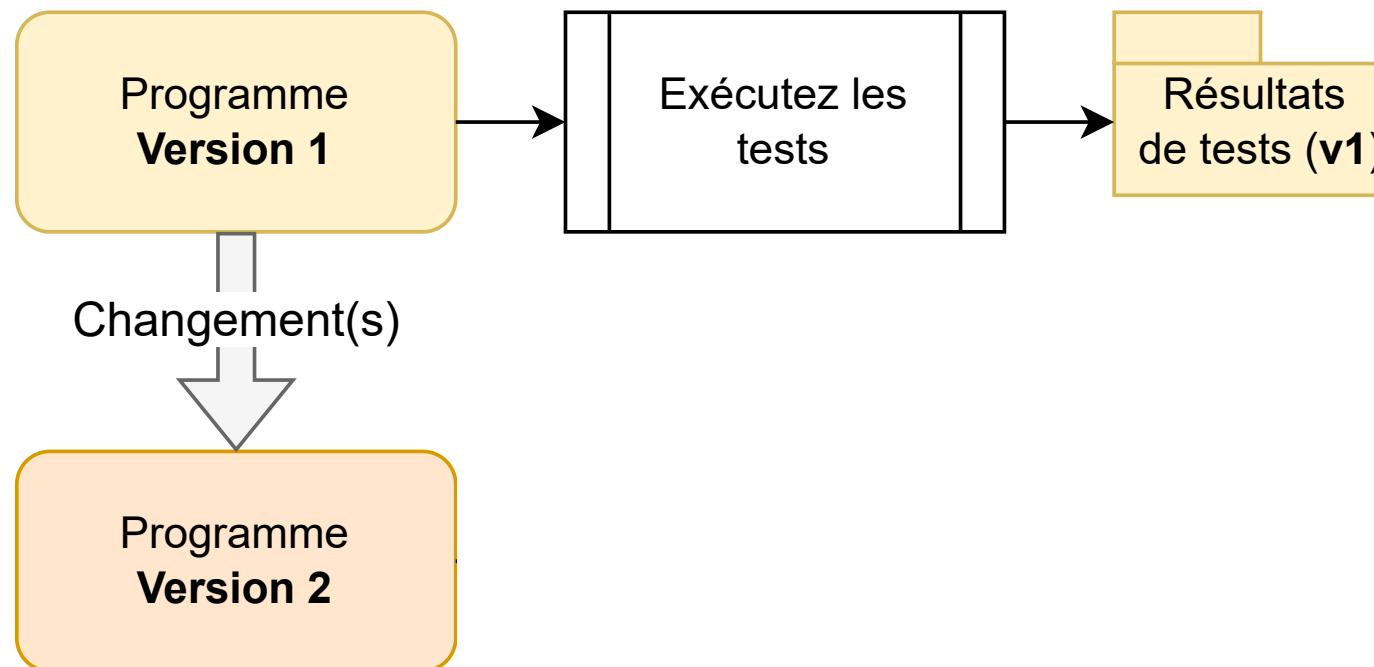
```
public int[] reverse(int[] origin){  
    int[] target = new int[origin.length];  
    int index = 0;  
    while(index < origin.length - 1){      //bug, missing origin[0]  
        index++;  
        target[origin.length-index] = origin[index];  
    }  
    return target;  
}
```

Régression!

Plante maintenant quand
la longueur d'origin est 0

```
public int[] reverse(int[] origin){  
    int[] target = new int[origin.length];  
    int index = 0;  
    while(index < origin.length - 1){  
        index++;  
        target[origin.length-index] = origin[index];  
    }  
    target[origin.length-1] = origin[0];  
    return target;  
}
```

Idée générale



Changements de code

Comment définir les changements :

- supposons que nous supprimons la différence d'espacement et/ou de commentaires
- Avec des « diff » de text très simple tout changement est repéré

Si nous déplaçons une méthode du bas d'une classe vers le début de la classe, est-ce vraiment une différence ?

Pourquoi réfléchir à notre approche de régression ?

« J'ai déjà une bonne quantité de tests rédigés, pas de problèmes, non ? »

Comment déterminer quels tests exécuter ?

Comment déterminer quels tests mettre à jour ?

Une solutions: traçabilité de tests et couverture de code

	Test d1	Test d2	Test d3	Test d4	...	Test dY
Fonction1	1	0	1	1
Fonction2	1	1	0	1
Fonction3	1	1	0	0
Fonction4	0	1	1	1
...
FonctionX

Si je modifie la Fonction2, je sais que je dois mettre-à-jour les tests d1, d2 et d4 et les exécuter.

Traçabilité de tests et couverture de code avec critères de tests

	Instructions	Branches	MC/DC	Catégorie-partition
Jeu de tests T1	90%	85%	60%	30%
Jeu de tests T2	20%	15%	10%	80%
Jeu de tests T3	15%	10%	5%	60%
Jeu de tests T4	75%	90%	85%	50%
...
Jeu de tests TX

Si le critère le plus important est la couverture des branches,
alors le jeu de tests T4 est le plus approprié

Priorisation

Certains jeux de tests coûtent plus cher que d'autres.



Certains tests sont plus pertinents que d'autres.

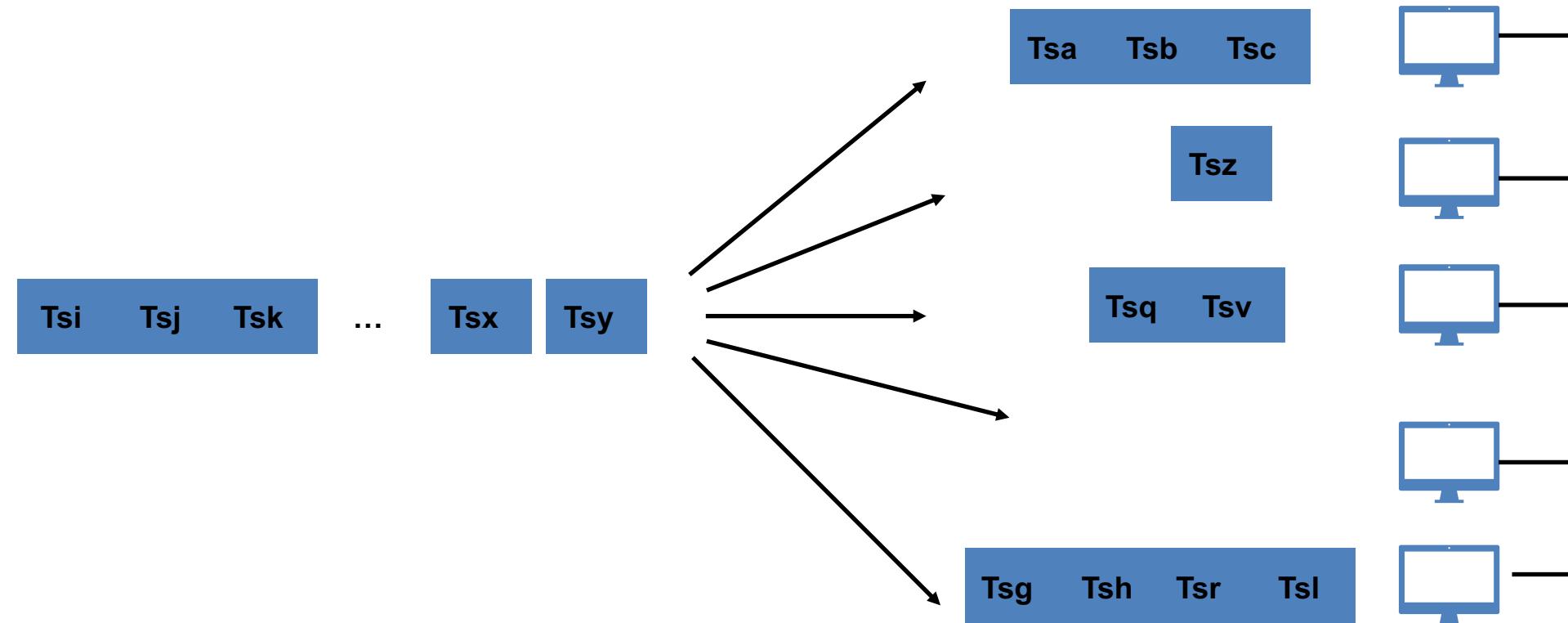
On peut donc prioriser sur la base du ratio bénéfice/coût.

- Trouver une manière de calculer qui fait du sens peut être difficile, mais c'est possible.

Priorisation: exemple

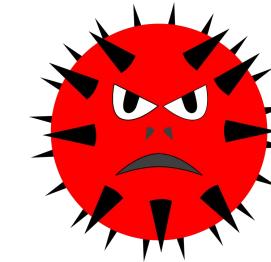
	Temps d'exécution (s)	Cohérence avec le profil opérationnel (%)	%/s
Jeu de test T1	35s	10%	0,29
Jeu de test T2	28s	25%	0,89
Jeu de test T3	50s	60%	1,2
Jeu de test T4	400s	95%	0,24

Pourquoi prioriser ? Pourquoi ne pas simplement paralléliser ?



Fault Exposing Potential (FEP)

Il est possible de mesurer la capacité d'un jeu de tests à trouver des défauts en évaluant sa capacité à tuer des mutants.



- Le FEP correspond au score de mutation du jeu de tests.
 - Rappel : il s'agit du pourcentage de mutants tués par le jeu de tests.
 - Un jeu de test ayant un FEP plus élevé offre donc un meilleur bénéfice.

Average Percentage of Fault Detection (APFD)

Utiliser les données historiques de la capacité des cas de test pour révéler les défauts

- Nous utilisons une moyenne pondérée du pourcentage de défauts détectés sur la durée de vie de la suite de tests -- pourcentage moyen de détection de défauts (APFD)
- Nous pensons qu'un APFD plus élevé signifie un taux de détection de défauts plus rapide (meilleur)

Ordre des tests

	1	2	3	4	5	6	7	8	9	10
TC1	x				x					
TC2	x				x	x	x			
TC3	x	x	x	x	x	x	x			
TC4				x						
TC5							x	x	x	

Ordre lexicographique

TC	% Défauxts détectés
TC1	20
TC2	40
TC3	70
TC4	70
TC5	100

Ordre des tests

	1	2	3	4	5	6	7	8	9	10
TC1	x				x					
TC2	x				x	x	x			
TC3	x	x	x	x	x	x	x			
TC4				x						
TC5							x	x	x	

Ordre de % de détections

TC	% Défauxts détectés
TC3	70
TC5	100
TC2	100
TC1	100
TC4	100

Ordre des tests

	1	2	3	4	5	6	7	8	9	10
TC1	x				x					
TC2	x				x	x	x			
TC3	x	x	x	x	x	x	x			
TC4				x						
TC5							x	x	x	

Ordre de % de détections

TC	% Défauxts détectés
TC3	70
TC5	100

Améliorer le test de régression

Une fois que nous avons un test de régression, il est important de le mettre à jour chaque fois que nous

- corrigéons un bug
- ajoutons, modifions ou supprimons des fonctionnalités
- changeons de plateforme

Si nous créons des variantes du système, nous devons également créer des variantes parallèles de la suite de tests de régression.

You are now ready to do the activity on Moodle

Tests de Charge

Tests de performances --> Déterminer si la performance est adéquate ou si elle est pire qu'avant

LOG3430 Méthodes de test et de validation du logiciel

Ex: Tremblement de Terre

Si tout est normal, puis qu'un tremblement arrive au Qc, tt le monde va se mettre à envoyer des textos ce qui va augmenter l'utilisation pendant les 20 minutes après le tremblement. Cela, on n'a pas pu le prédire car on ne prédit pas les tremblements

Systèmes logiciels (ultra) à grande échelle



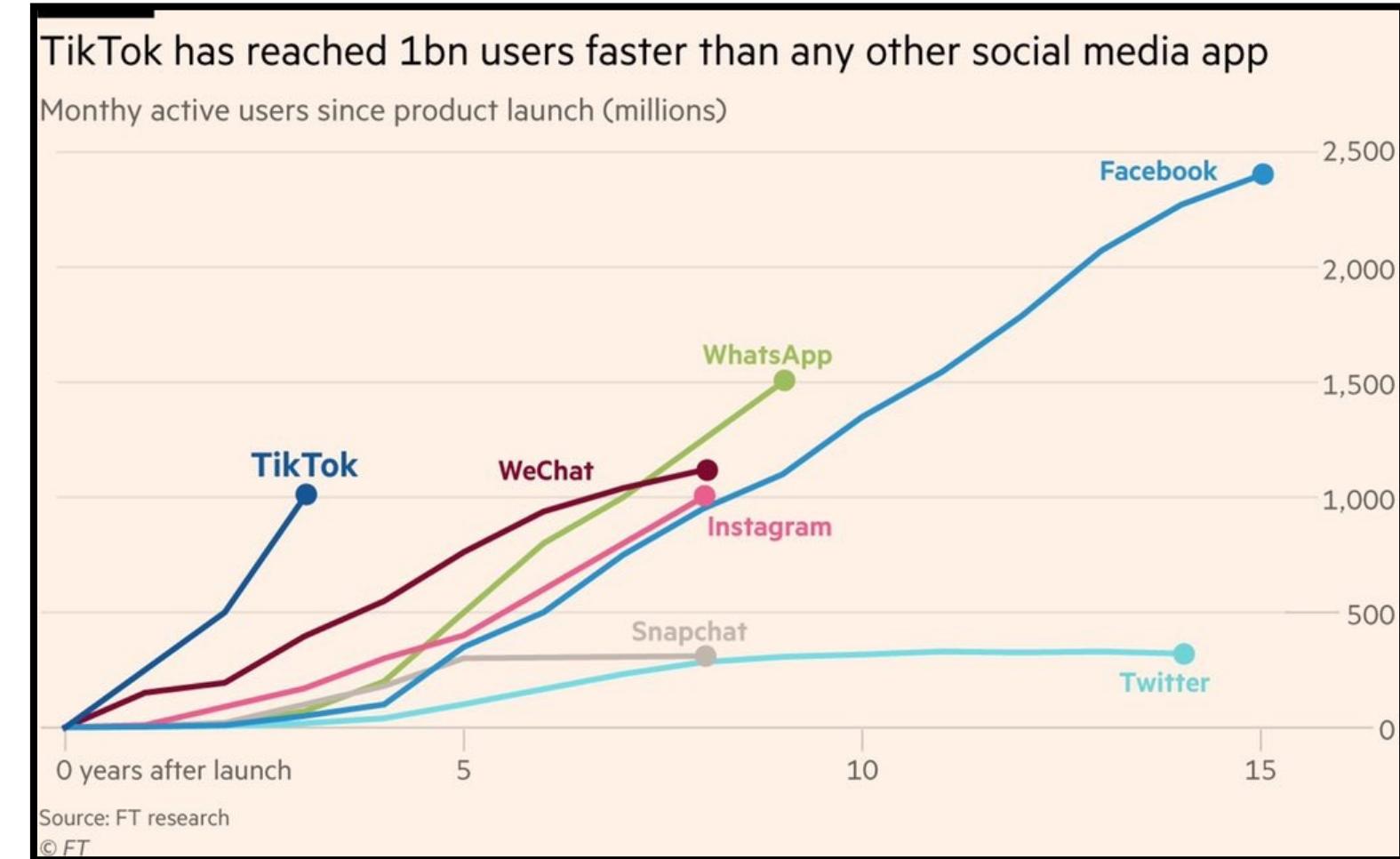
4 millions d'utilisateurs
2600-3000 req/sec la plupart des jours



450 millions d'utilisateurs actifs
> 50 milliards de messages chaque jour



Croissance rapide et modèles d'utilisation variés



Les problèmes des systèmes à grande échelle sont rarement à fc

HealthCare.gov



We have a lot of visitors on the site

Please stay on this page

We're working to make the experience better. Please stay in your place in line. We'll send you to the front of the line as soon as we can. Thanks for your patience!

lost connection
(02/05/13)

over 8 million users
(05/24/13)

Test de charge nécessaire !

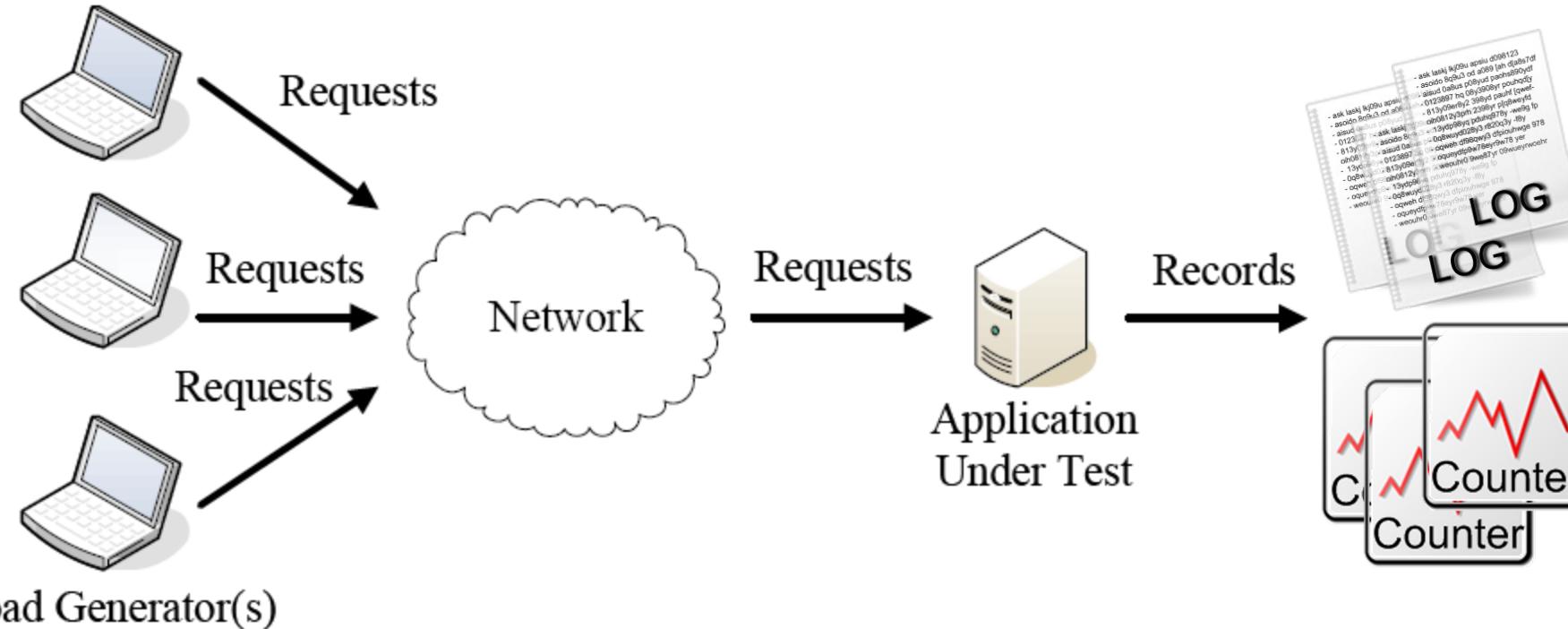


Hi-Tech Technology

Facebook down due to "degraded performance" across the world

January 26, 2020 0 Comments technology

Test de charge



Conception de tests

Imite plusieurs utilisateurs effectuant à plusieurs reprises les mêmes tâches

Exécution des tests

Peut prendre des heures voire des jours

Analyse des tests

Conception d'un test

Supposons qu'un système dispose de 5 paramètres de configuration utilisateur.

- Deux paramètres ont 3 valeurs possibles.
- Trois paramètres ont 2 valeurs possibles.
- Il y a donc $2^3 \times 3^2 = 72$ configurations possibles à tester.

Le serveur Web Apache dispose de 172 paramètres de configuration utilisateur (158 options binaires). Ce système a $1,8 \times 10^{55}$ configurations possibles à tester !

L'objectif de la conception de test de charge
est d'obtenir le maximum d'informations
avec le nombre minimum de tests.

Terminologies

Le résultat d'une expérience est appelé la **variable de réponse** (response variable).

- Par exemple, le débit et le temps de réponse pour les tâches.

Chaque variable qui affecte la variable de réponse et a plusieurs alternatives est appelée un **facteur** (factor).

- Par exemple, pour mesurer les performances d'un poste de travail, il existe quatre facteurs : le type de processeur, la taille de la mémoire, le nombre de disques durs et la charge de travail.

Les valeurs que peut avoir un facteur sont appelées **niveaux** (levels).

- Par exemple, la taille de la mémoire a 3 niveaux : 2 Go, 6 Go et 12 Go

La répétition de tout ou partie des expériences est appelée **réPLICATION** (replication).

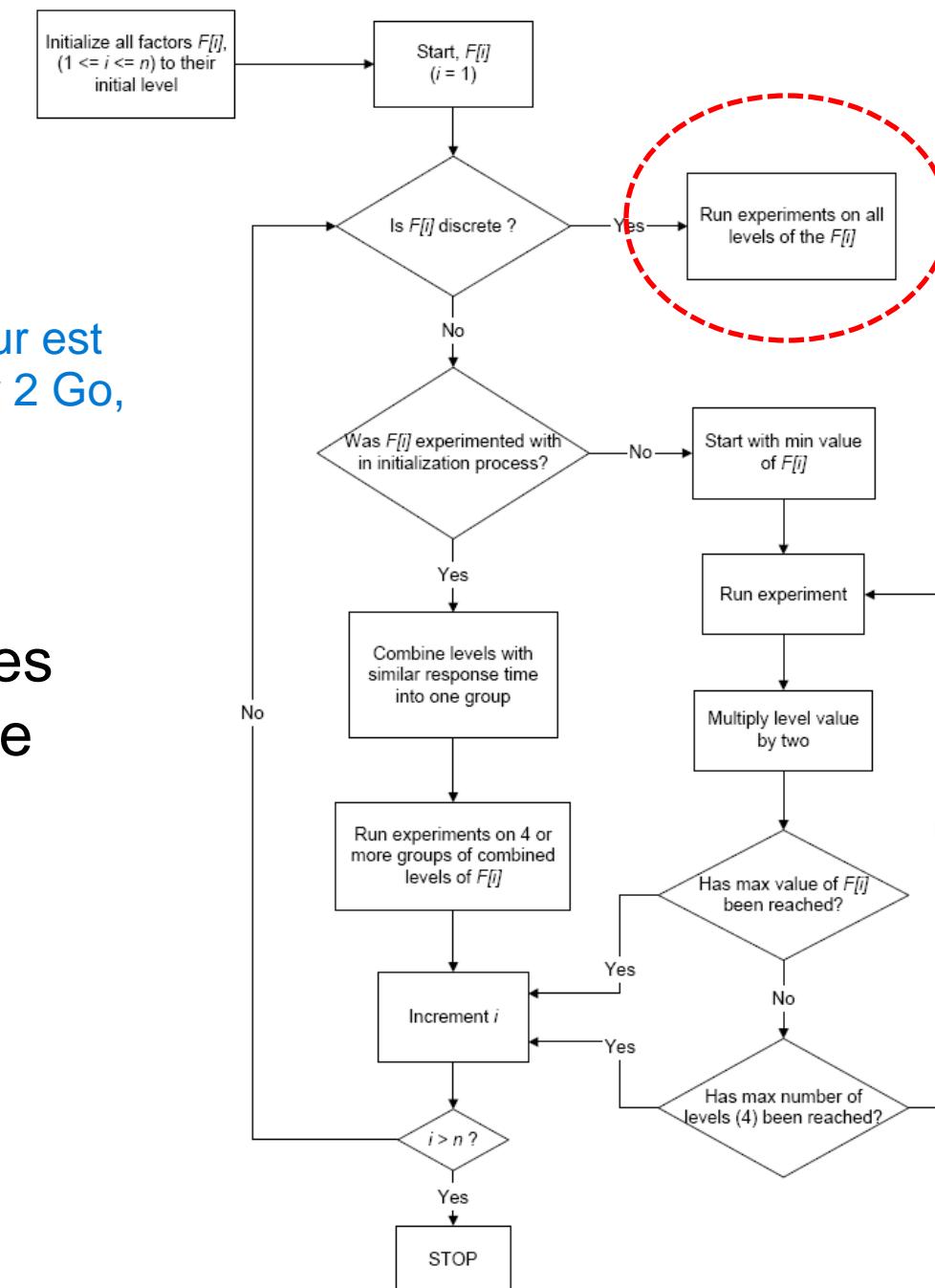
Effets d'interaction (Interaction effects): On dit que deux facteurs A et B interagissent si l'effet de l'un dépend de l'autre.

Approche Ad-hoc

Je commence avec 1 facteur, je détermine si le facteur est discrèt ou non(réponses concrètes). Ex: je peux avoir 2 Go, 4Go, 8 Go, mais on ne peut avoir 2.2 Go

Revoir, de manière itérative, chacun des facteurs (discrets et continus) et des facteurs d'identité qui ont un impact sur les performances d'un système de commerce électronique à trois niveaux.

4 tests par facteurs sont acceptables, on va couvrir la majorité des situations



Covering Array

Un tableau pour un modèle d'espace d'entrées (t -way covering array) est un ensemble de configurations dans lequel chaque combinaison valide de valeurs-de-facteurs pour chaque combinaison de facteurs t apparaît au moins une fois.

Supposons qu'un système dispose de 5 paramètres de configuration utilisateur. Trois paramètres sur cinq ont 2 valeurs possibles (0, 1) et les deux autres paramètres ont 3 valeurs possibles (0, 1, 2). Il y a au total $2^3 \times 3^2 = 72$ configurations possibles à tester.

Un réseau de couverture à 2 voies
(2-way covering array)

A	B	C	D	E
0	1	1	2	0
0	0	0	0	0
0	0	0	1	1
1	1	1	0	1
0	1	0	0	2
1	0	1	1	0
1	1	1	1	2
1	0	0	2	1
1	0	0	2	2

Covering Array et CIT

Il existe de nombreux autres types de tableaux de couverture, tels que : tableau de couverture à force variable, tableau de couverture sensible aux cas de test, etc.

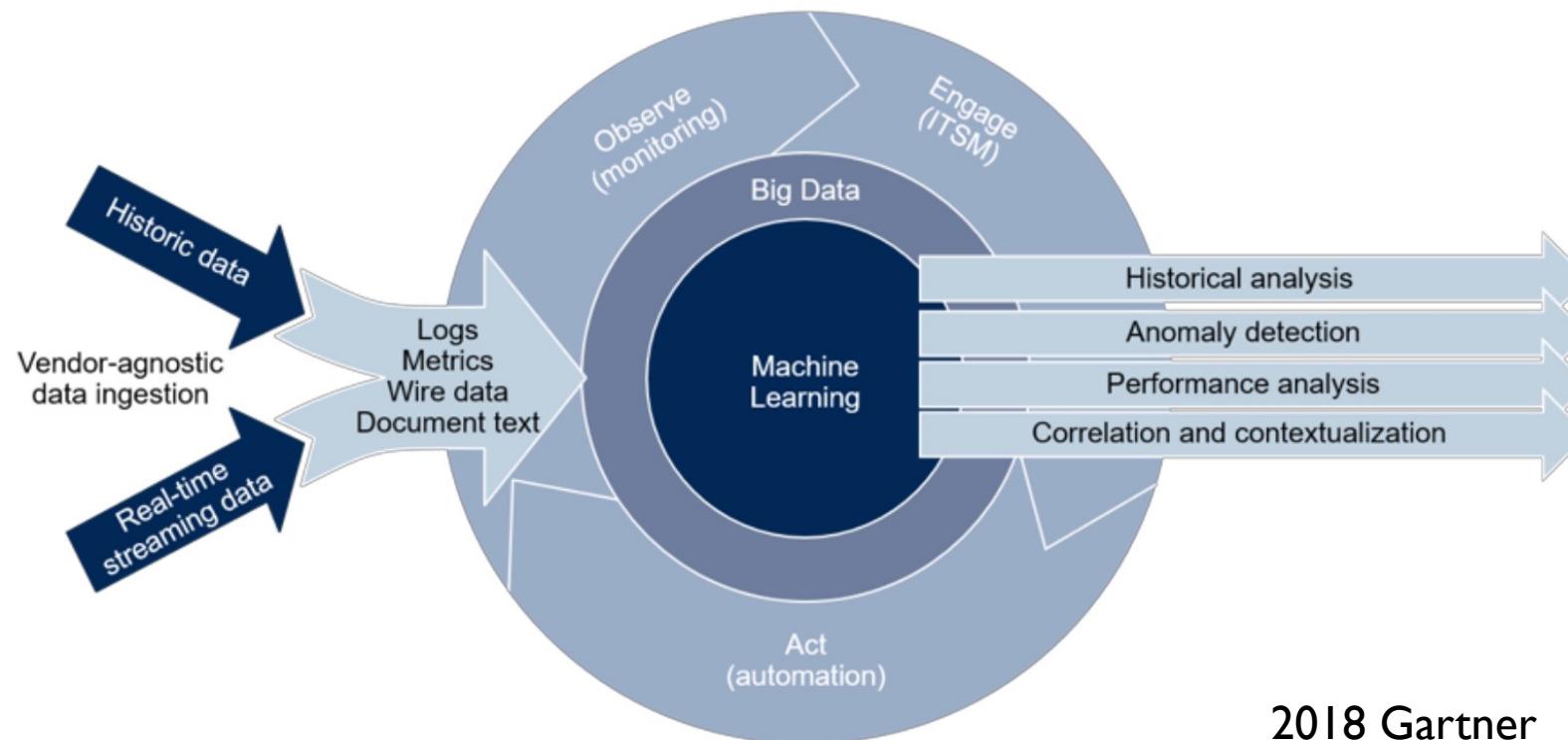
Le test d'interaction combinatoire (Combinatorial Interaction Testing: CIT) modélise un système testé comme un ensemble de facteurs, dont chacun tire ses valeurs d'un domaine particulier. CIT génère un échantillon qui répond aux critères de couverture spécifiques (par exemple, couverture à 3 facteurs).

De nombreux outils commerciaux et gratuits existent : <http://pairwise.org/tools.asp>

Appliquer AIOps pour trouver des configurations optimales

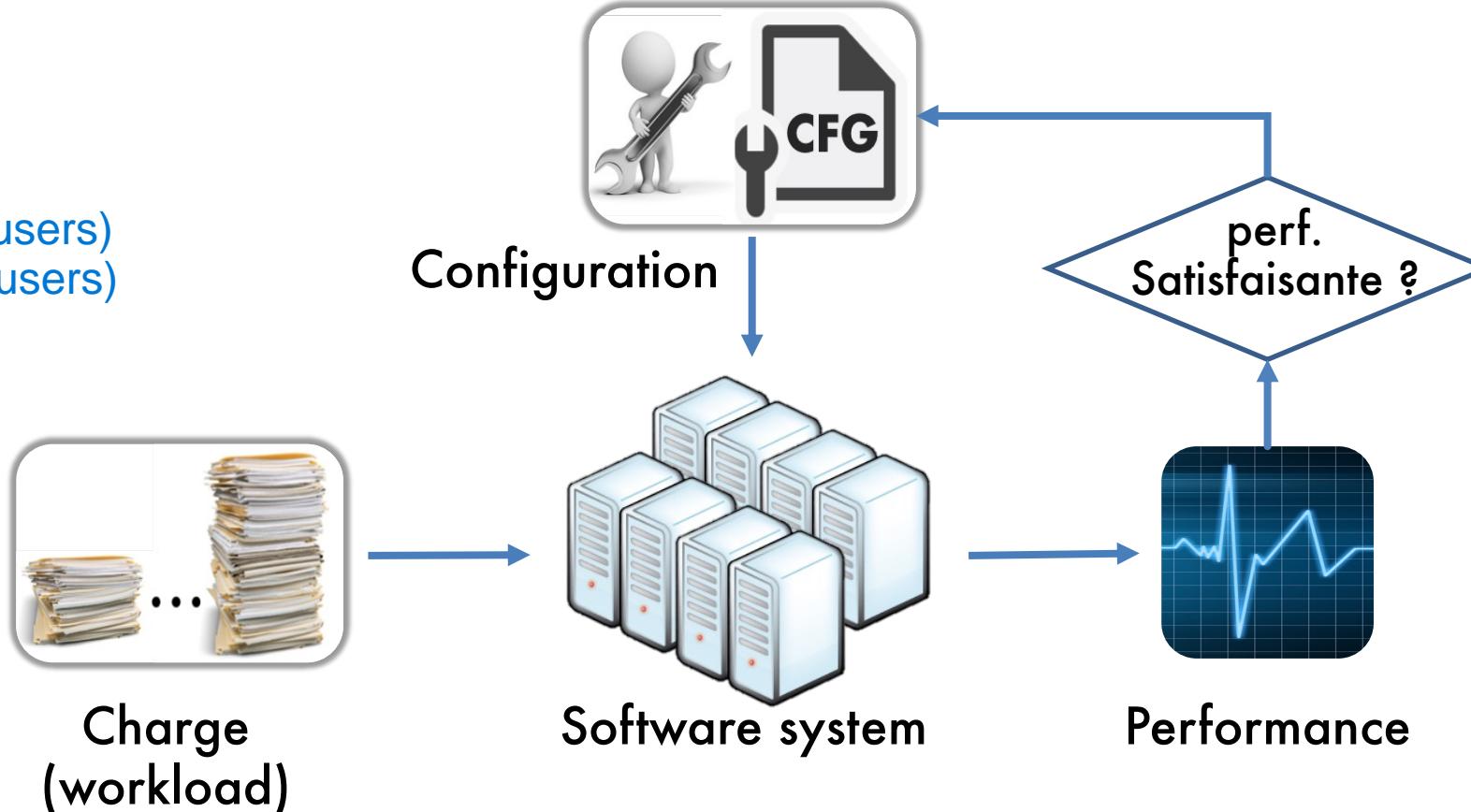
AIOps améliore les opérations informatiques grâce à de meilleures informations en combinant le « Big Data », l'apprentissage automatique et la visualisation (Gartner)

AIOps relève les défis DevOps avec l'IA (Microsoft)



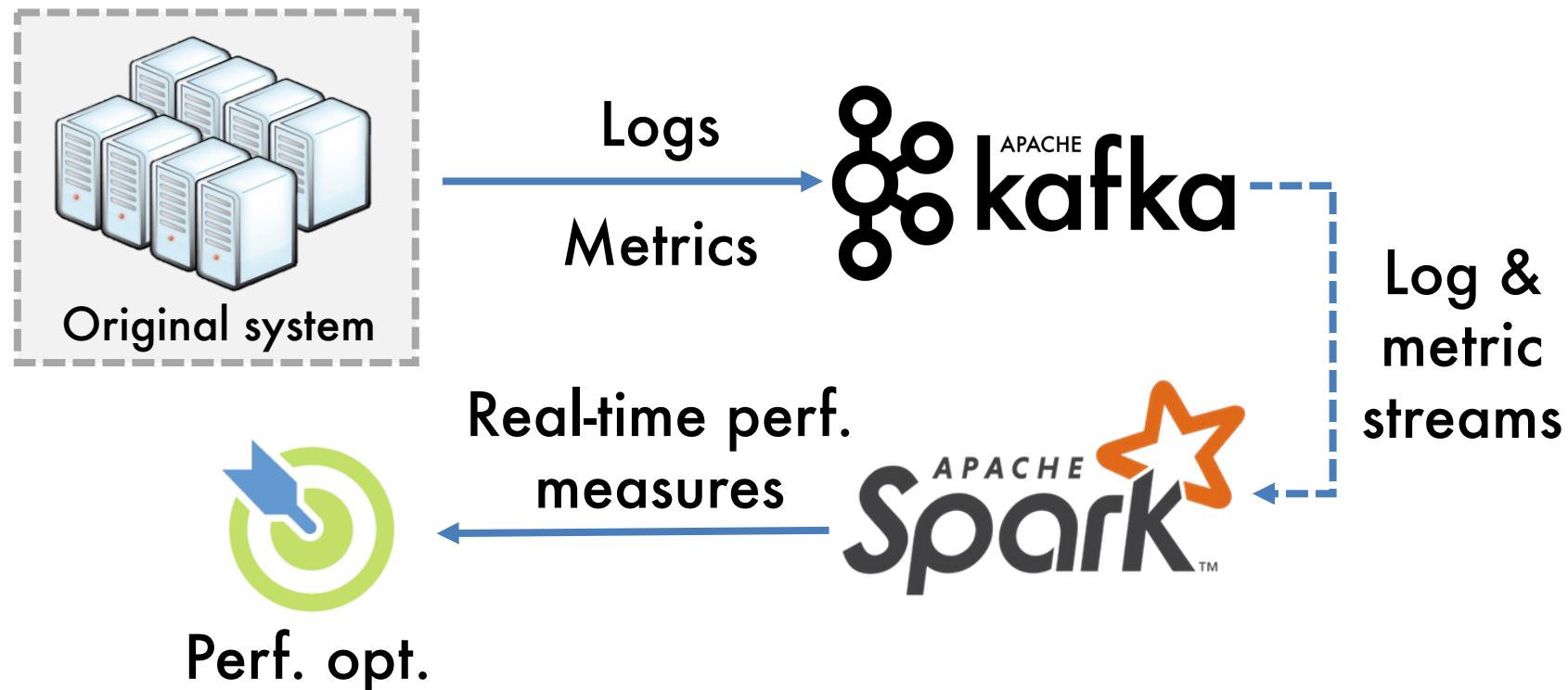
La configuration manuelle de systèmes logiciels à grande échelle est coûteuse et sujette aux erreurs

Lundi-Jeudi (500 users)
Ven-Dim (50,000 users)



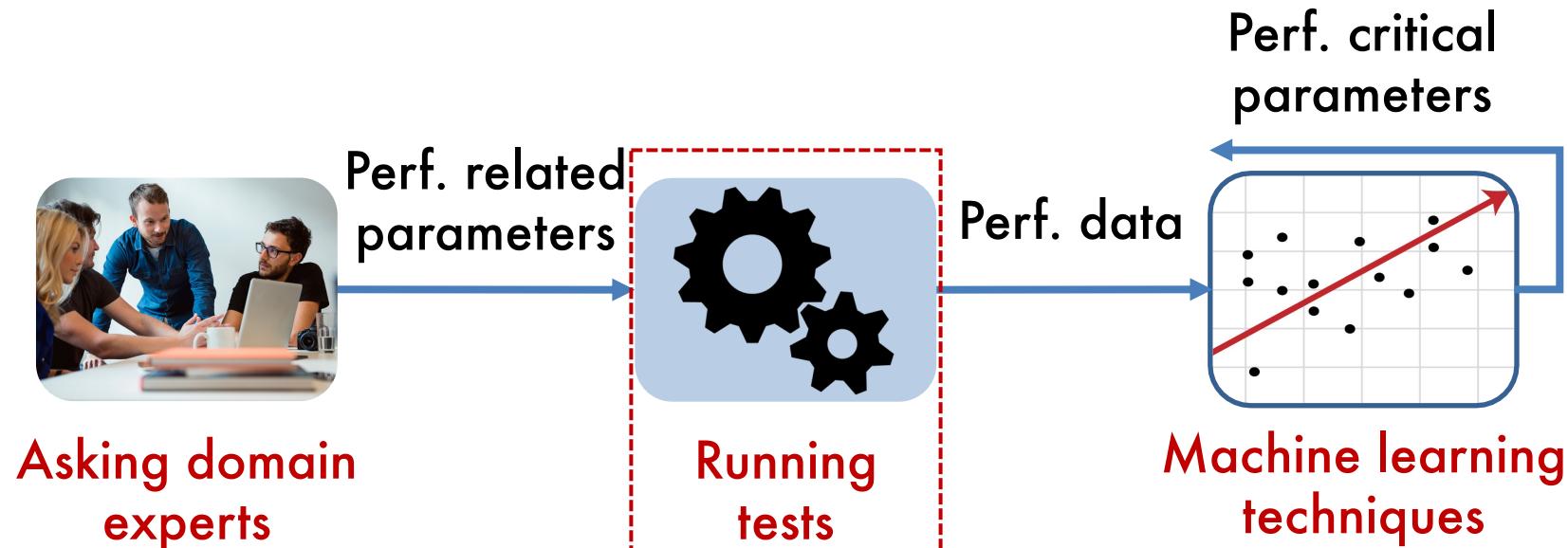
Les charges de travail évoluent constamment, nécessitant une **intervention humaine constante** pour garantir des performances optimales

Tirer parti des logs et de métriques pour surveiller le comportement du système



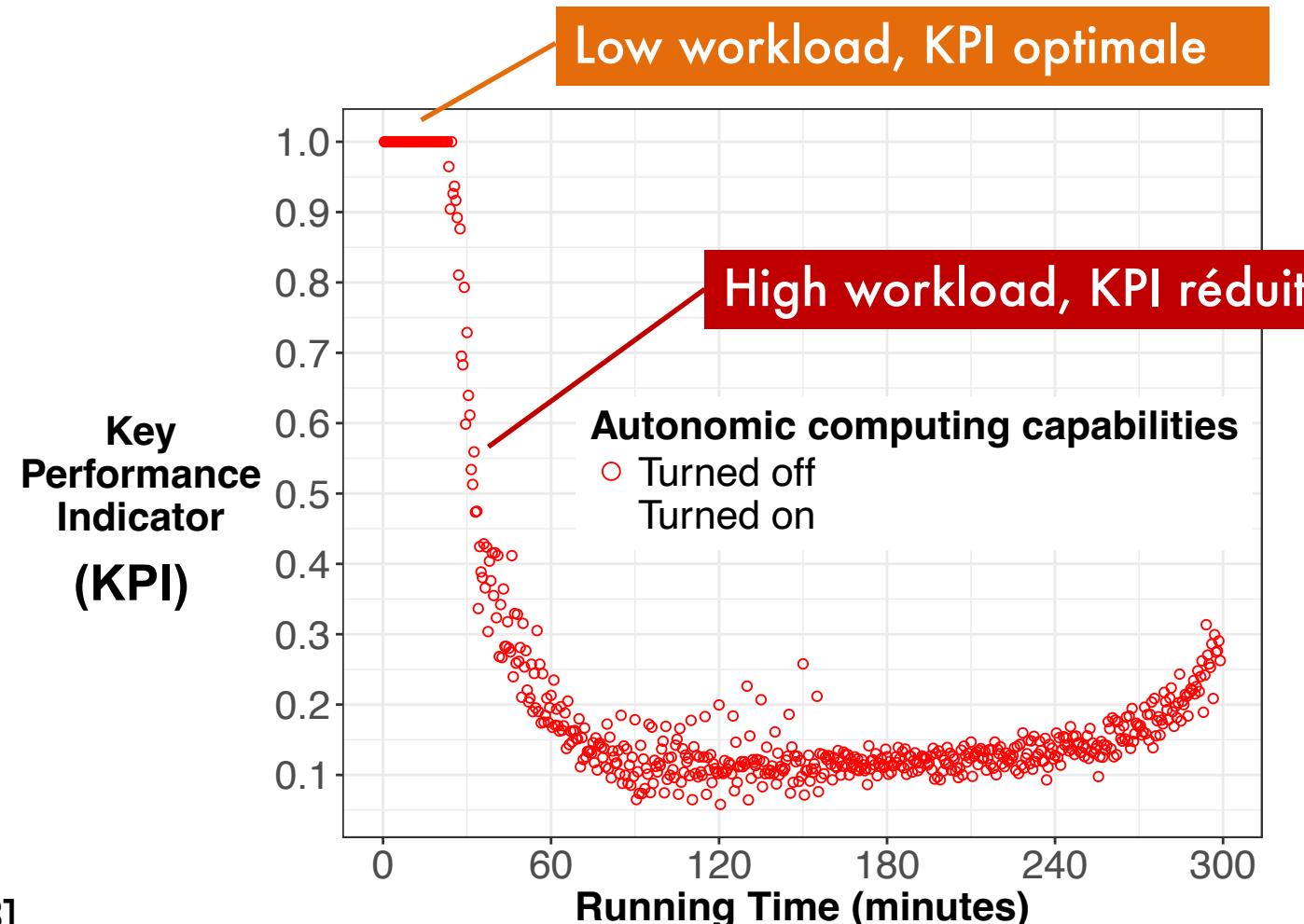
Réponse rapide aux changements de charge de travail(en quelques secondes)

La relation entre les paramètres de configuration et la performance

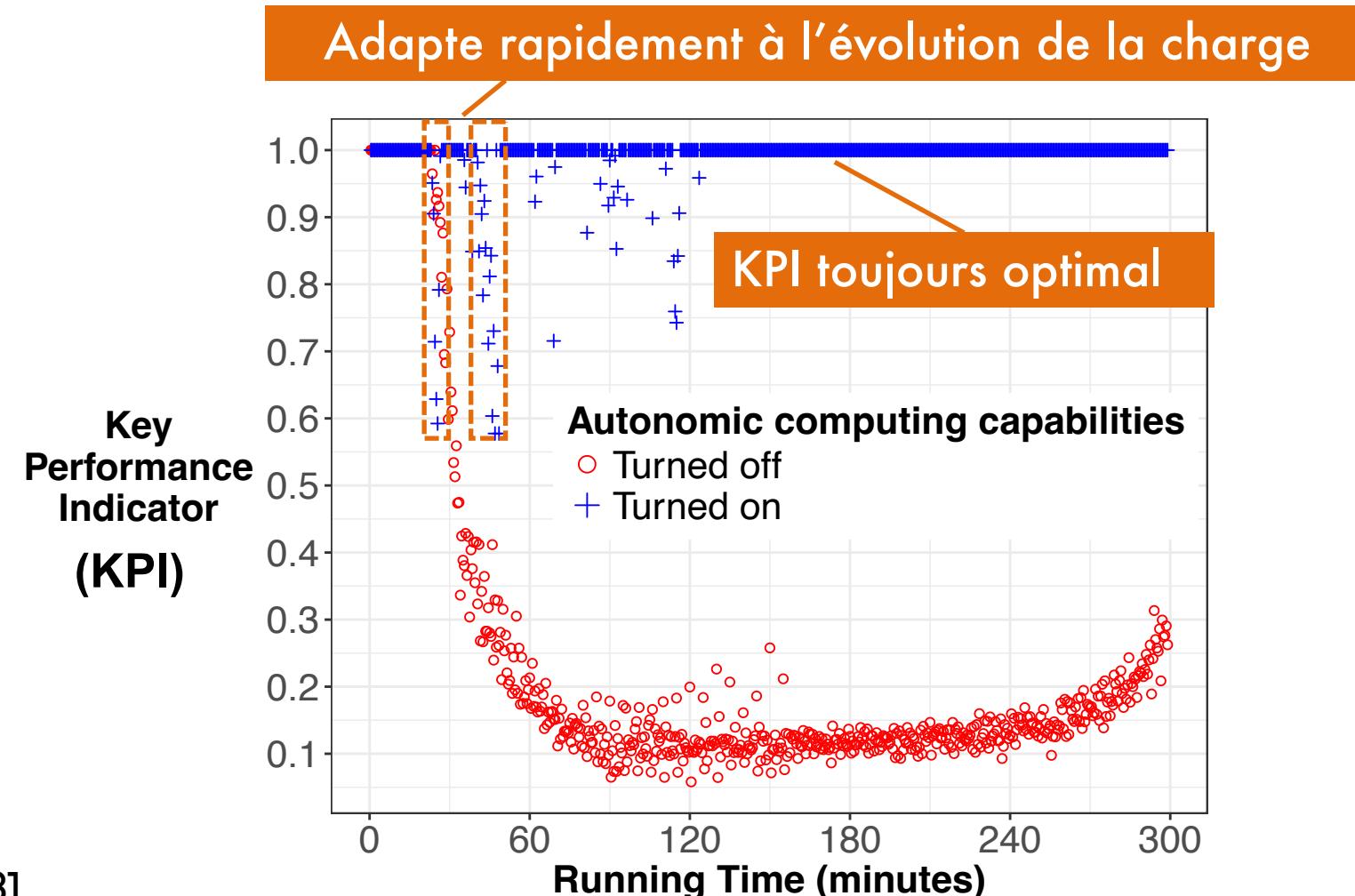


Seuls quelques-uns des nombreux paramètres ont un impact significatif sur la performance d'un système

La configuration autonome améliore considérablement la performances d'un système



La configuration autonome améliore considérablement la performances d'un système

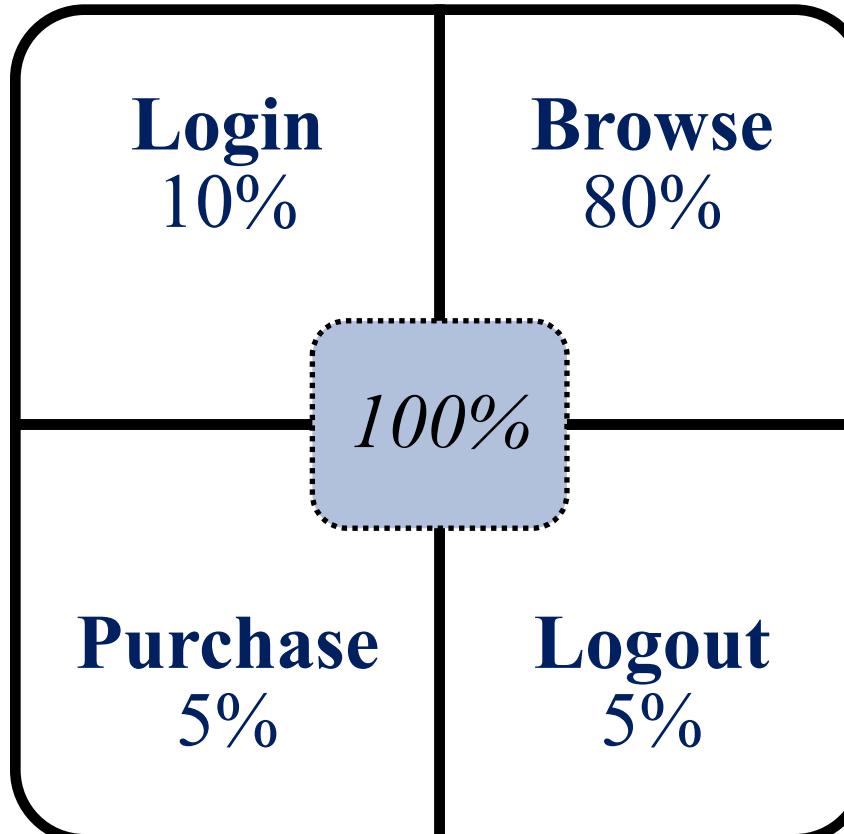


Conception d'un test de charge

Conception de charges réalistes

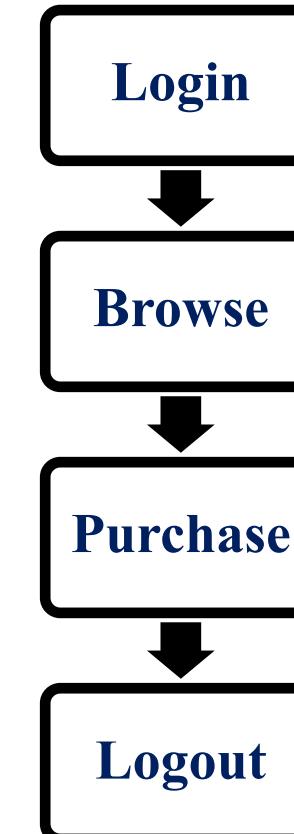
Système de commerce électronique

Charge de travail globale



Steady Load, Step-wise load,
Extrapolated load

Use-Case



Load Dérivé de UML, Markov et Stochastic Form-oriented Models

Caractériser une charge de travail globale

Mélange de charge de travail

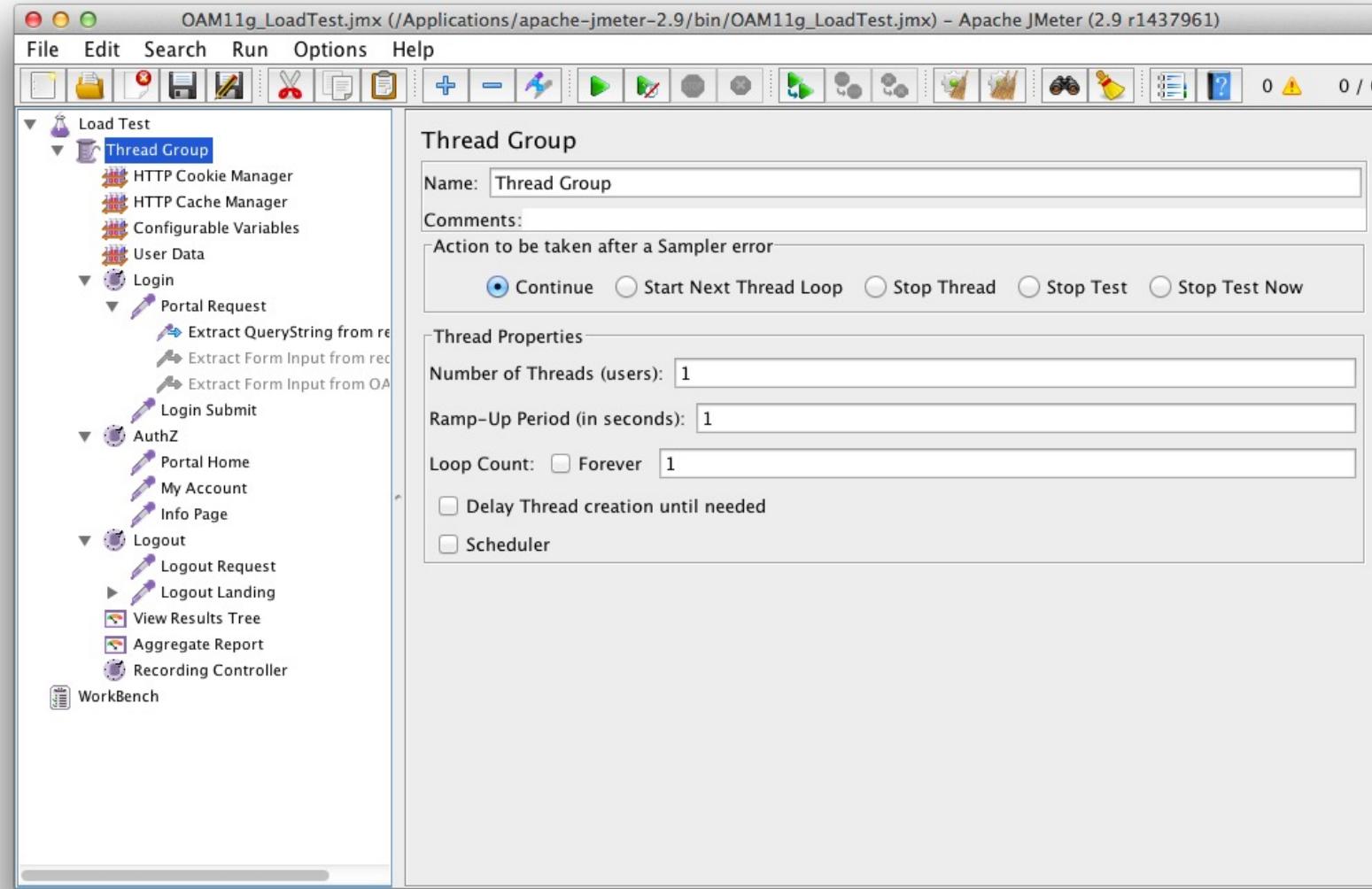
- navigation (30%), achat (10%) et recherche (60%)

Intensité de la charge de travail

- Taux de requêtes (5 requêtes/sec)



Spécification d'un test de charge dans Apache JMeter



Utilisation d'outils open source pour les tests de performances chez Google

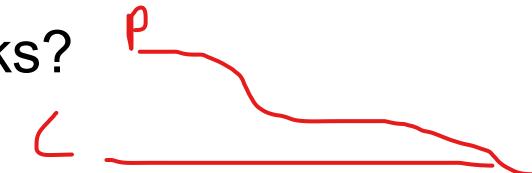
- <https://www.youtube.com/watch?v=k9h51BM2h4w>

Charge de travail globale (1)

le nb de requêtes ne change pas tout au long de nos tests

Charge stable (Steady Load)

- Facile à mesurer
- Memory leaks?



1000 users qui utilisent le système

[Bondi, CMG 2007]

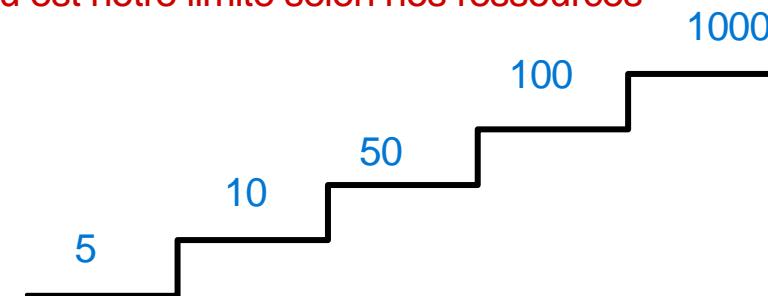
Si on a une charge constante, mais que la performance dégrade avec le temps, on va avoir des memory leaks

Charge pas-à-pas (Step-wise Load)

Charge pas-à-pas vs Charge stable: charge p-à-p permet de terminer ou est notre limite selon nos ressources

- Même mélange de charge de travail
- Intensité différente

5 users qui sont différents, ces 5 mm profils on va les multiplier



[Hayes, CMG 2000]

Ici, on a de + en + de users, si la performance baisse c'est normal car on a plus de users

Dérivation des charges de test à partir des données historiques

Charge de travail globale (2)

En cas de données d'utilisation passées manquantes, les charges de test peuvent être extrapolées à partir des sources suivantes :

- Données d'utilisation bêta *ex: joueurs testeurs avec des profils d'utilisateurs réguliers*
- Entretiens avec des experts du domaine
- Données des concurrents



Approche de base : mesurer la fréquence des événements



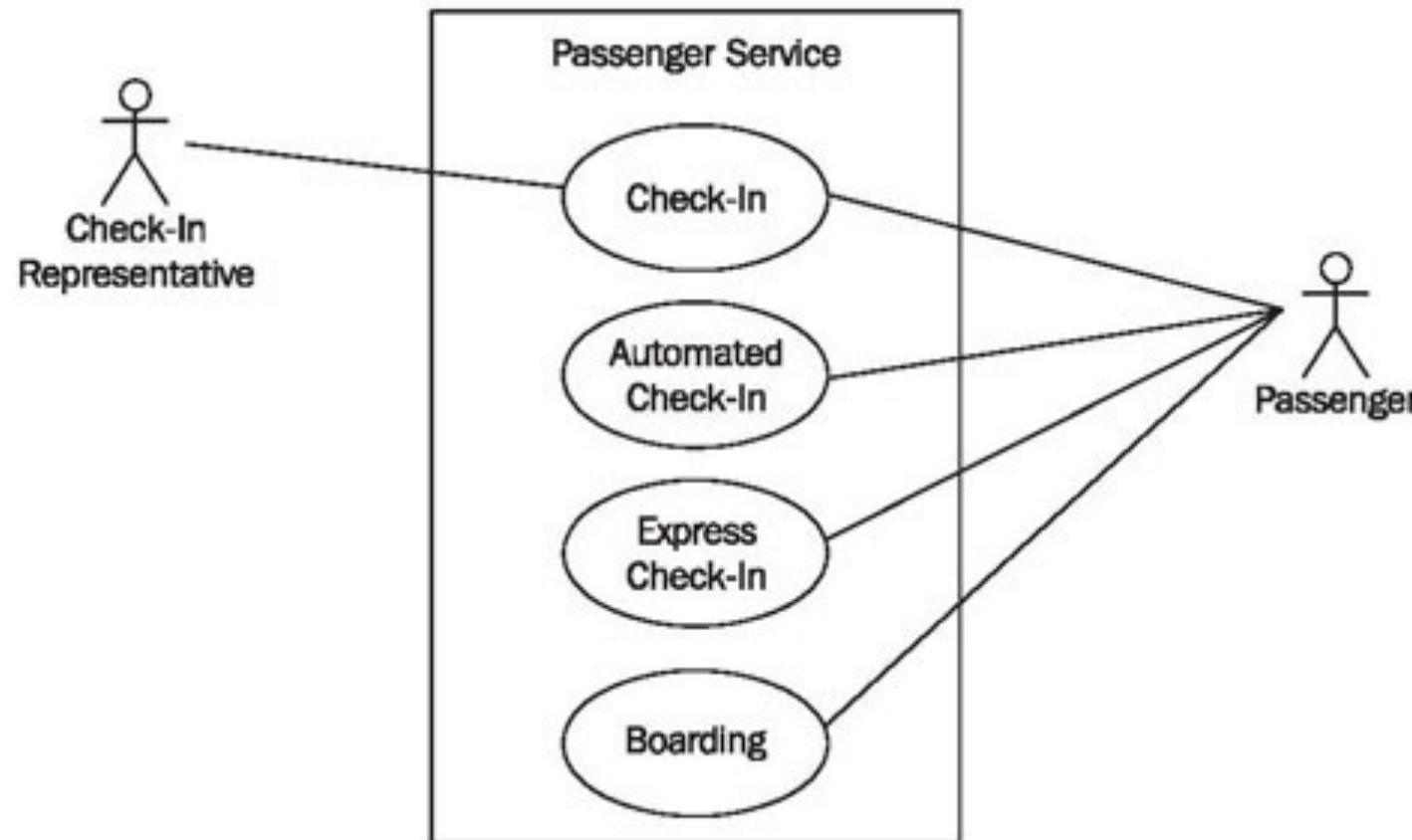
Connaissance du
domaine des analystes
de performance

Plan de test

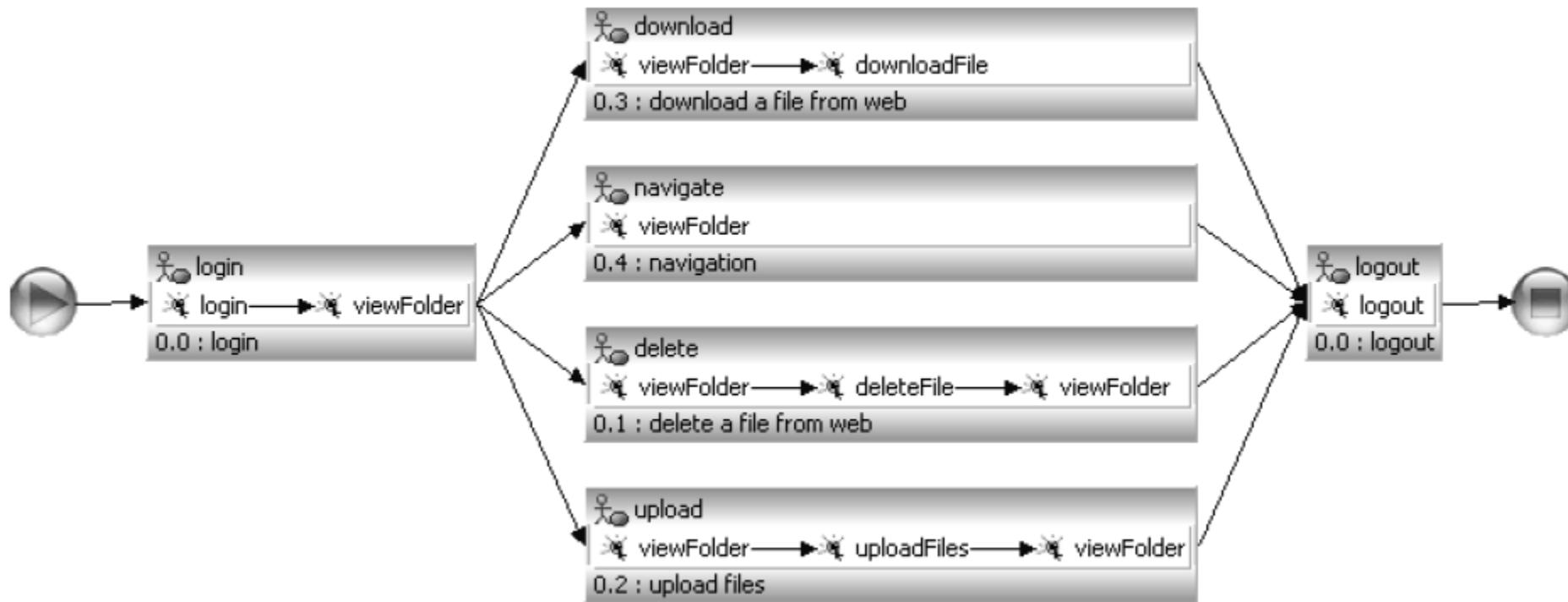
Événement	Fréquence
Login	50/min
Purchase	200/min
Browse	1000/min
...	

*Les actions de l'utilisateur
se produisent en séquences
et ne sont pas seulement
des événements discrets !*

Se souvenir des diagrammes de cas d'utilisation (Use-case diagram)



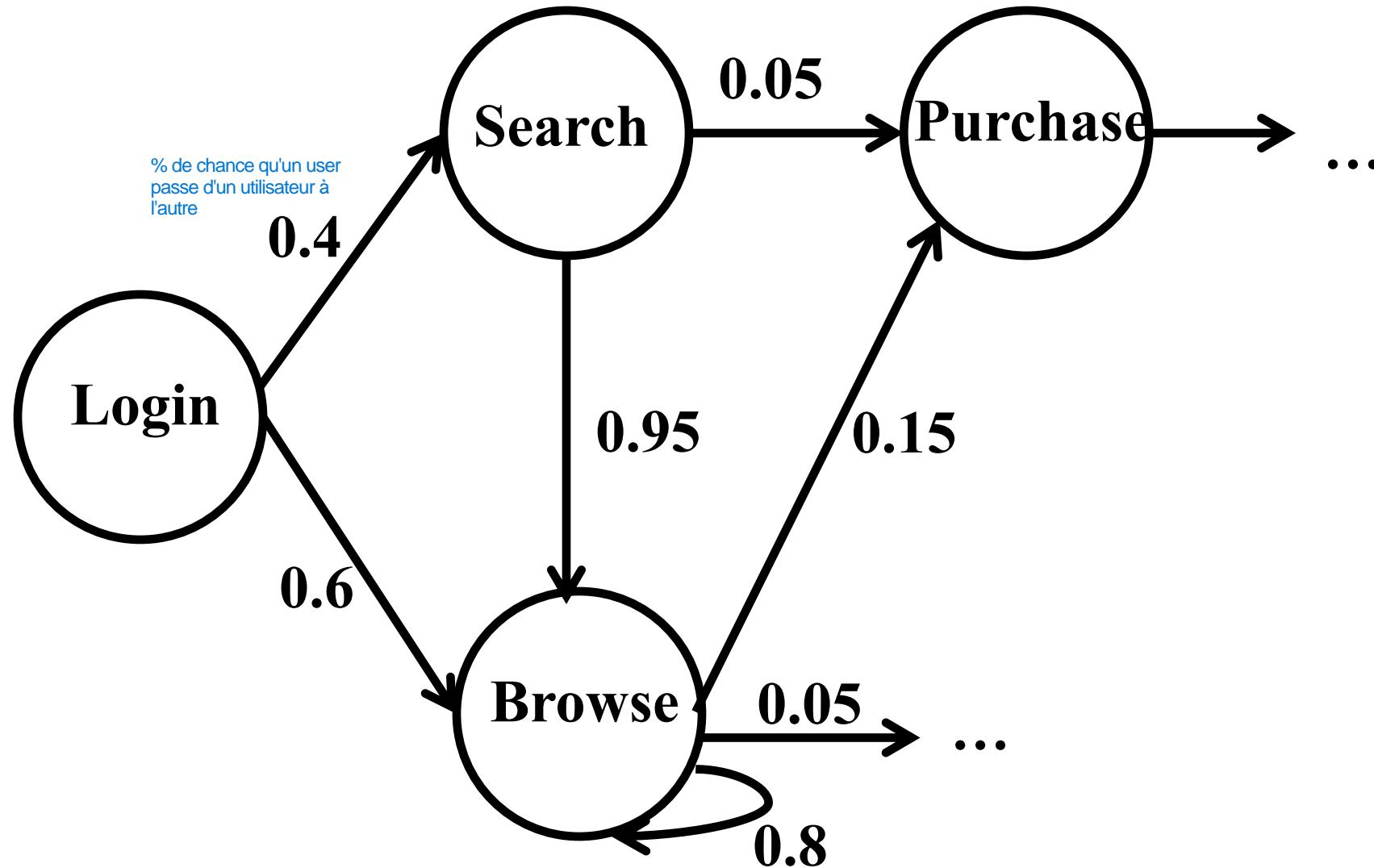
Use-Case (1) - UML



The RUG (Realistic Usage Model)
- derived based on UML use case diagrams

Use-Case (2)

- Markov Chain



Use-Case (2) - Markov Chain

```
192.168.0.1 - [22/Apr/2014:00:32:25 -0400] "GET /dsbrowse.jsp?browsetype=actor&browse_category=&browse_actor=ANTHONY%20BENING&browse_title=&limit_num=2&c
192.168.0.1 - [22/Apr/2014:00:32:25 -0400] "GET /dsbrowse.jsp?browsetype=category&browse_category=11&browse_actor=&browse_title=&limit_num=2&customerid=5
192.168.0.1 - [22/Apr/2014:00:32:25 -0400] "GET /dslogin.jsp?username=user41&password=password HTTP/1.1" 200 2539 16
192.168.0.1 - [22/Apr/2014:00:32:25 -0400] "GET /dsbrowse.jsp?browsetype=actor&browse_category=&browse_actor=WILLIAM%20BRODY&browse_title=&limit_num=5&cu
192.168.0.1 - [22/Apr/2014:00:32:25 -0400] "GET /dsbrowse.jsp?browsetype=category&browse_category=15&browse_actor=&browse_title=&limit_num=3&customerid=4
192.168.0.1 - [22/Apr/2014:00:32:25 -0400] "GET /dsbrowse.jsp?browsetype=actor&browse_category=&browse_actor=HILARY%20GOLDBERG&browse_title=&limit_num=3&
192.168.0.1 - [22/Apr/2014:00:32:25 -0400] "GET /dsbrowse.jsp?browsetype=category&browse_category=6&browse_actor=&browse_title=&limit_num=8&customerid=41
192.168.0.1 - [22/Apr/2014:00:32:25 -0400] "GET /dsbrowse.jsp?browsetype=title&browse_category=&browse_actor=&browse_title=HOLY%20AUTUMN&limit_num=8&cust
192.168.0.1 - [22/Apr/2014:00:32:25 -0400] "GET /dspurchase.jsp?confirmpurchase=yes&customerid=5961&item=646&quan=3&item=2551&quan=1&item=45&quan=3&item=
192.168.0.1 - [22/Apr/2014:00:32:25 -0400] "GET /dspurchase.jsp?confirmpurchase=yes&customerid=41&item=4544&quan=1&item=6970&quan=3&item=5237&quan=2&item
192.168.0.1 - [22/Apr/2014:00:32:29 -0400] "GET /dslogin.jsp?username=user3614&password=password HTTP/1.1" 200 728 6
192.168.0.1 - [22/Apr/2014:00:32:29 -0400] "GET /dsbrowse.jsp?browsetype=title&browse_category=&browse_actor=&browse_title=GUNFIGHTER%20MALTESE&limit_num
192.168.0.1 - [22/Apr/2014:00:32:29 -0400] "GET /dsbrowse.jsp?browsetype=actor&browse_category=&browse_actor=ELLEN%20GARCIA&browse_title=&limit_num=5&cus
192.168.0.1 - [22/Apr/2014:00:32:29 -0400] "GET /dsbrowse.jsp?browsetype=category&browse_category=9&browse_actor=&browse_title=&limit_num=1&customerid=36
192.168.0.1 - [22/Apr/2014:00:32:29 -0400] "GET /dsbrowse.jsp?browsetype=actor&browse_category=&browse_actor=ANGELINA%20DIETRICH&browse_title=&limit_num=
192.168.0.1 - [22/Apr/2014:00:32:29 -0400] "GET /dsbrowse.jsp?browsetype=actor&browse_category=&browse_actor=JULIA%20TAUTOU&browse_title=&limit_num=7&cus
192.168.0.1 - [22/Apr/2014:00:32:29 -0400] "GET /dspurchase.jsp?confirmpurchase=yes&customerid=3614&item=4717&quan=2&item=9118&quan=1&item=4793&quan=3&it
192.168.0.1 - [22/Apr/2014:00:32:31 -0400] "GET /dslogin.jsp?username=user13337&password=password HTTP/1.1" 200 1960 9
192.168.0.1 - [22/Apr/2014:00:32:31 -0400] "GET /dsbrowse.jsp?browsetype=title&browse_category=&browse_actor=&browse_title=CHILL%20MOONWALKER&limit_num=5
192.168.0.1 - [22/Apr/2014:00:32:31 -0400] "GET /dspurchase.jsp?confirmpurchase=yes&customerid=13337&item=322&quan=2&item=2535&quan=2&item=5157&quan=3&it
192.168.0.1 - [22/Apr/2014:00:32:35 -0400] "GET /dslogin.jsp?username=user5414&password=password HTTP/1.1" 200 2579 10
192.168.0.1 - [22/Apr/2014:00:32:35 -0400] "GET /dsbrowse.jsp?browsetype=actor&browse_category=&browse_actor=GRACE%20BRIDGES&browse_title=&limit_num=9&cu
192.168.0.1 - [22/Apr/2014:00:32:35 -0400] "GET /dspurchase.jsp?confirmpurchase=yes&customerid=5414&item=198&quan=3&item=5697&quan=2&item=732&quan=1 HTTP
192.168.0.1 - [22/Apr/2014:00:32:35 -0400] "GET /dsnewcustomer.jsp?firstname=RHUSQS&lastname=EBFMQDBUNM&address1=9098231627%20Dell%20Way&address2=&city=G
192.168.0.1 - [22/Apr/2014:00:32:35 -0400] "GET /dsbrowse.jsp?browsetype=title&browse_category=&browse_actor=&browse_title=GRINCH%20WILD&limit_num=5&cust
192.168.0.1 - [22/Apr/2014:00:32:35 -0400] "GET /dspurchase.jsp?confirmpurchase=yes&customerid=20001&item=7868&quan=3&item=3824&quan=2&item=4142&quan=2 H
192.168.0.1 - [22/Apr/2014:00:32:36 -0400] "GET /dslogin.jsp?username=user13713&password=password HTTP/1.1" 200 729 6
192.168.0.1 - [22/Apr/2014:00:32:36 -0400] "GET /dsbrowse.jsp?browsetype=category&browse_category=9&browse_actor=&browse_title=&limit_num=2&customerid=13
192.168.0.1 - [22/Apr/2014:00:32:36 -0400] "GET /dspurchase.jsp?confirmpurchase=yes&customerid=13713&item=493&quan=3&item=8890&quan=1&item=10000&quan=1&i
192.168.0.1 - [22/Apr/2014:00:32:41 -0400] "GET /dslogin.jsp?username=user9011&password=password HTTP/1.1" 200 728 6
```

web access logs depuis quelques mois

Use-Case (2)

- Markov Chain

```
192.168.0.1 - [22/Apr/2014:00:32:25 -0400] "GET
/dsbrowse.jsp?browsetype=title&browse_category=&browse_actor=&browse_title=
HOLY%20AUTUMN&limit_num=8&customerid=41 HTTP/1.1" 200 4073 10
```

```
192.168.0.1 - [22/Apr/2014:00:32:25 -0400] "GET
/dspurchase.jsp?confirmpurchase=yes&customerid=5961&item=646&quan=3&ite
m=2551&quan=1&item=45&quan=3&item=9700&quan=2&item=1566&quan=3&i
tem=4509&quan=3&item=5940&quan=2 HTTP/1.1" 200 3049 177
```

```
192.168.0.1 - [22/Apr/2014:00:32:25 -0400] "GET
/dspurchase.jsp?confirmpurchase=yes&customerid=41&item=4544&quan=1&item
=6970&quan=3&item=5237&quan=2&item=650&quan=1&item=2449&quan=1
HTTP/1.1" 200 2515 113
```

Use-Case (2) - Markov Chain

192.168.0.1 - [22/Apr/2014:00:32:25 -0400] "GET
/dsbrowse.jsp?browsetype=title&browse_category=&browse_actor=&browse_title=HOLY%20AUTUMN&limit_num=8&customerid=41 HTTP/1.1" 200 4073 T0

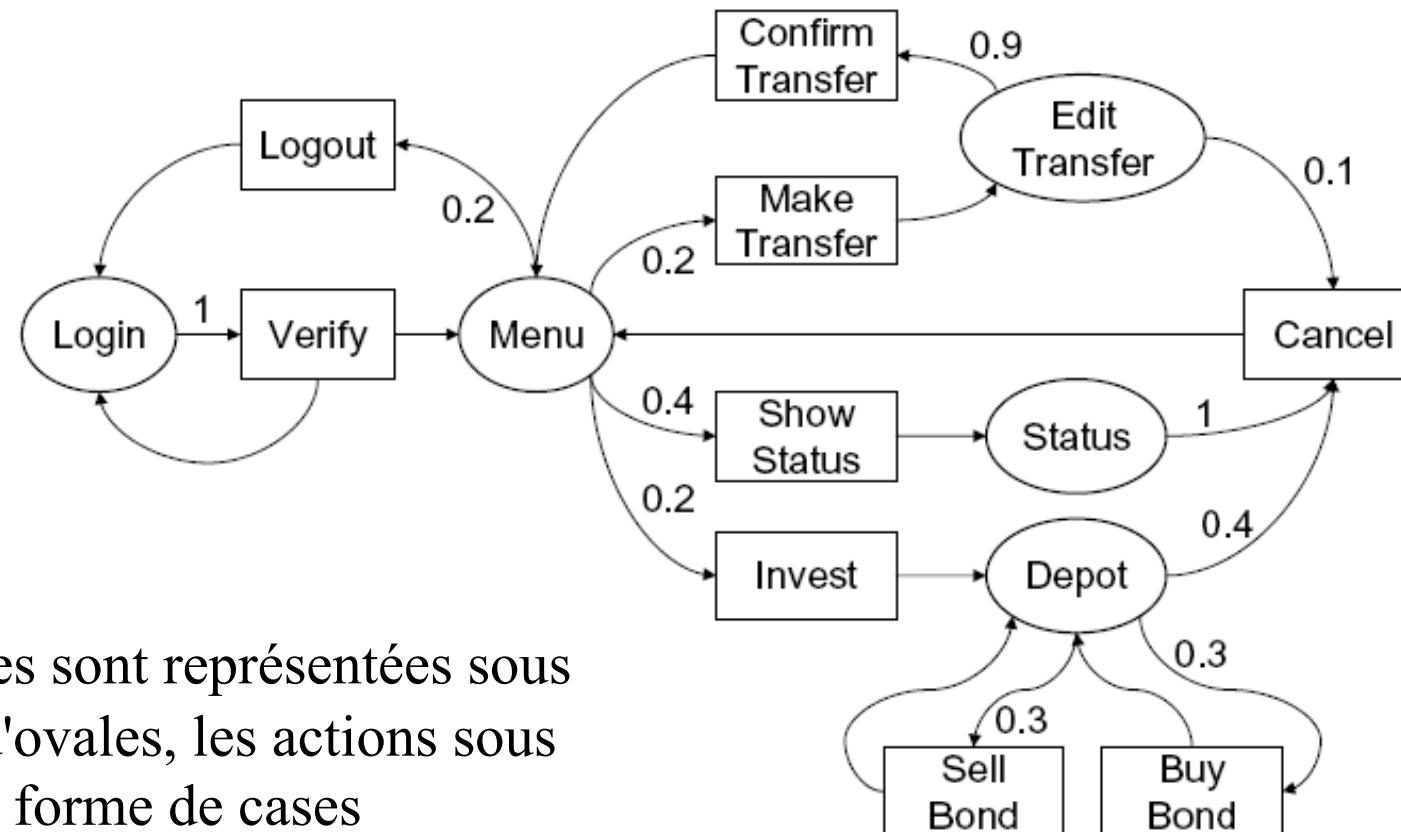
192.168.0.1 - [22/Apr/2014:00:32:25 -0400] "GET
/dspurchase.jsp?confirmpurchase=yes&customerid=5961&item=646&quan=3&item=2551&quan=1&item=45&quan=3&item=9700&quan=2&item=1566&quan=3&item=4509&quan=3&item=5940&quan=2 HTTP/1.1" 200 3049 177

192.168.0.1 - [22/Apr/2014:00:32:25 -0400] "GET
/dspurchase.jsp?confirmpurchase=yes&customerid=41&item=4544&quan=1&item=6970&quan=3&item=5237&quan=2&item=650&quan=1&item=2449&quan=1 HTTP/1.1" 200 2515 113

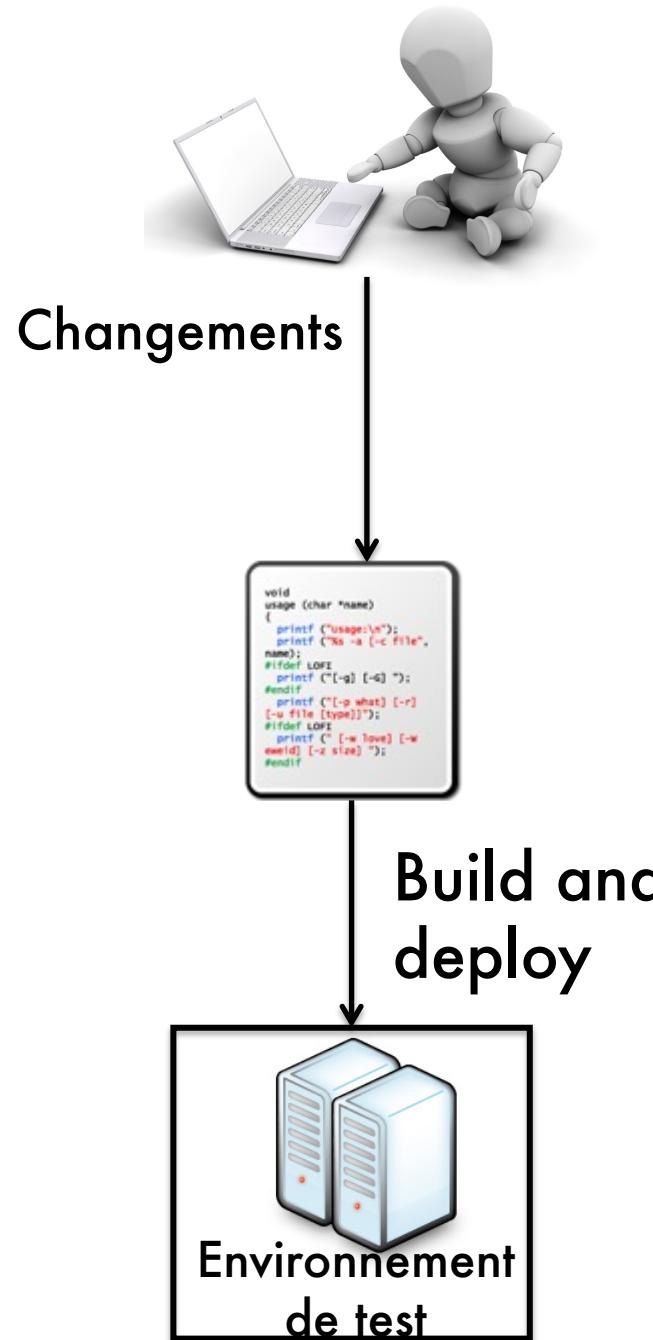
customer 41: browse -> purchase

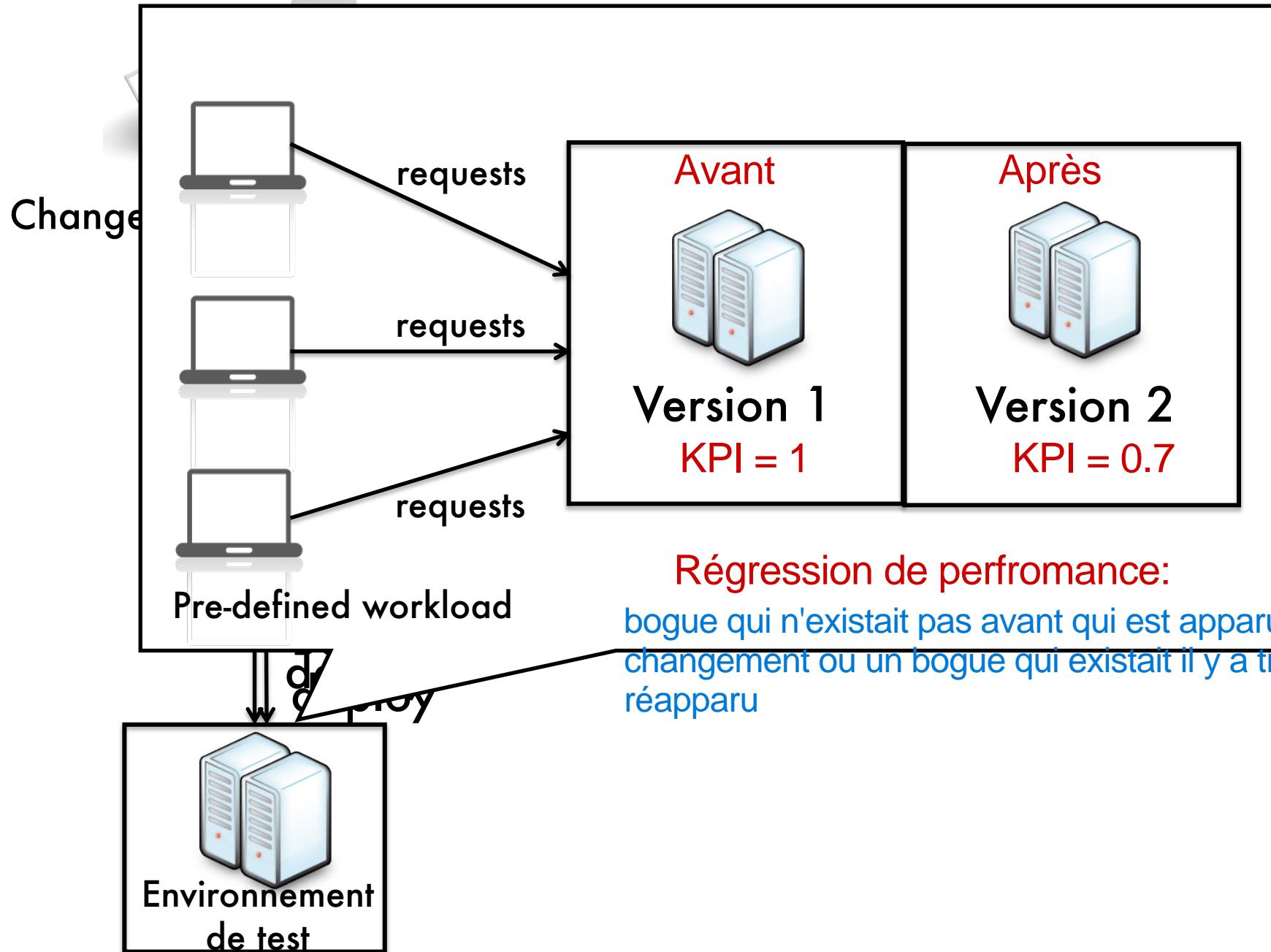
Use-Case (3)

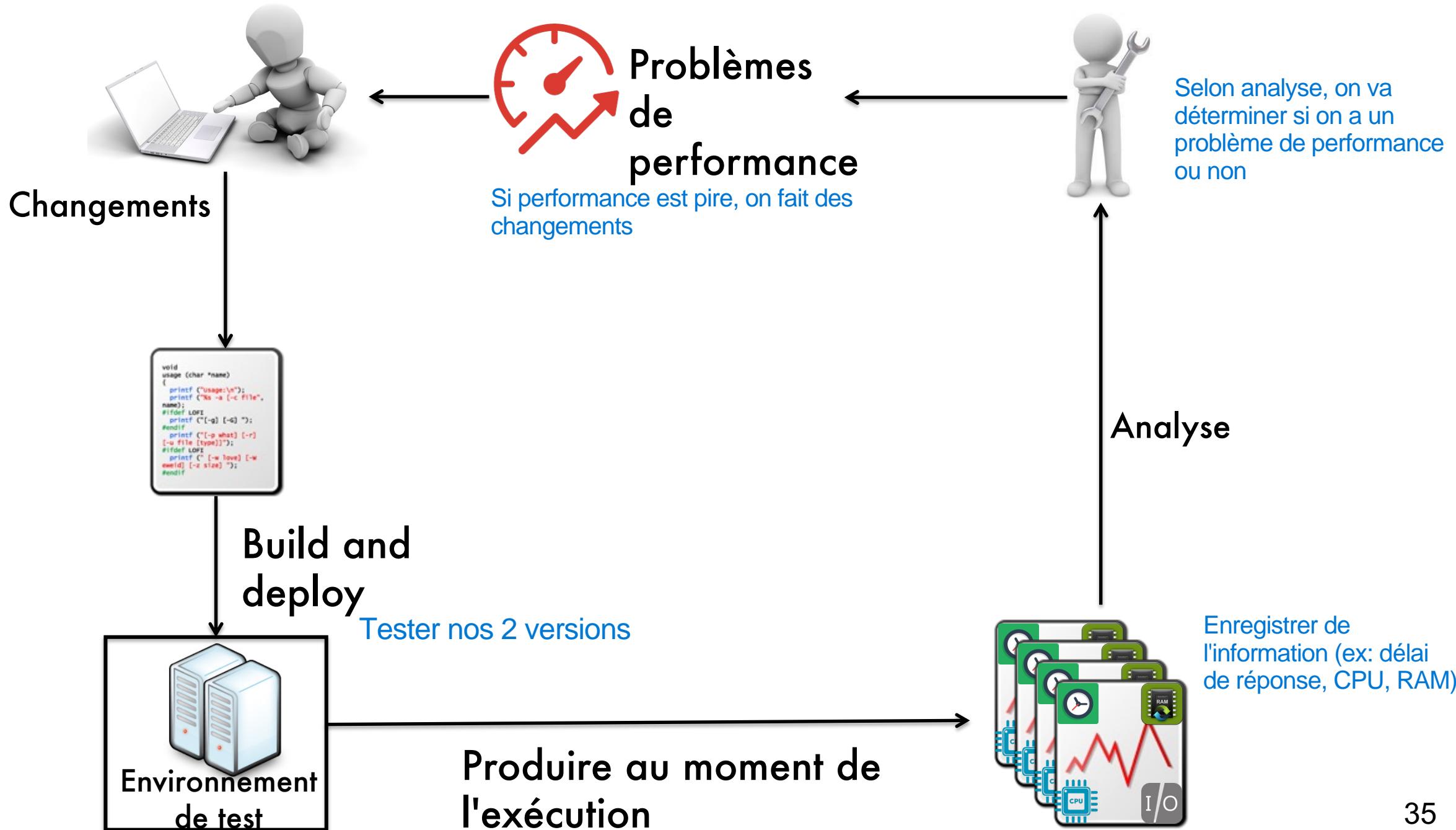
- Stochastic Form-Oriented Model



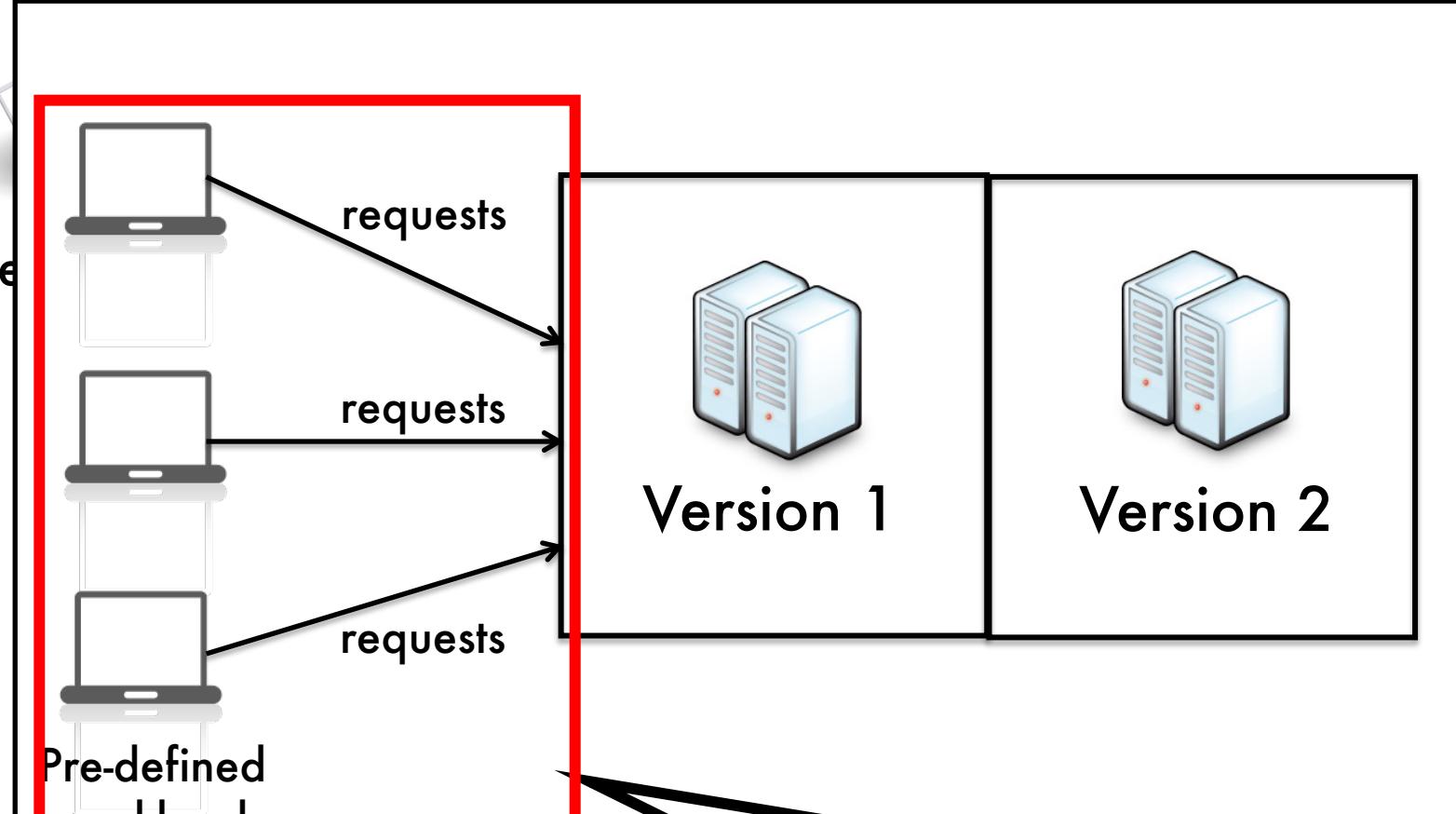
Les pages sont représentées sous forme d'ovales, les actions sous forme de cases







Change



Conception de tests de performance



Environnement
de test

Comment concevoir des tests de performance ?

Les fichiers de moins de 1 Ko représentent 35 % de toutes les demandes.

[SPECweb96]

Relisez les actions des utilisateurs enregistrées dans les logs du serveur au cours des dernières 24 heures.



Faible granularité

On n'a aucune idée de c'est quoi la différence entre 0.5 Ko et 1 Ko et qu'on ne sait pas ce qui se passe en haut de 1Ko
Informations détaillées manquantes

Haute granularité

Difficile à gérer

Charge de travail globale

Timestamp	Log line
00:00	Load data by Alice with size 32768 bytes
00:02	Read data by Alice with size 2048 bytes
00:03	Read data by Dan with size 1024 bytes
00:05	Read data by Alice with size 16384 bytes
00:06	Write data by Alice with size 8192 bytes

Envoyer en continu:
Load, read et write

Faible granularité

Haute granularité

Charge de travail globale en tenant compte de la distribution

Timestamp	Log line
00:00	Load data by Alice with size 32768 bytes
00:02	Read data by Alice with size 2048 bytes
00:03	Read data by Dan with size 1024 bytes
00:05	Read data by Alice with size 16384 bytes
00:06	Write data by Alice with size 8192 bytes



Load (20%), read (60%) et write (20%)

Faible granularité

Haute granularité

Charge de travail globale en tenant compte de la distribution et des utilisateurs

Timestamp	Log line
00:00	Load data by Alice with size 32768 bytes
00:02	Read data by Alice with size 2048 bytes
00:03	Read data by Dan with size 1024 bytes
00:05	Read data by Alice with size 16384 bytes
00:06	Write data by Alice with size 8192 bytes



User comme Alice:
Load (25%), read
(50%) and write (25%)

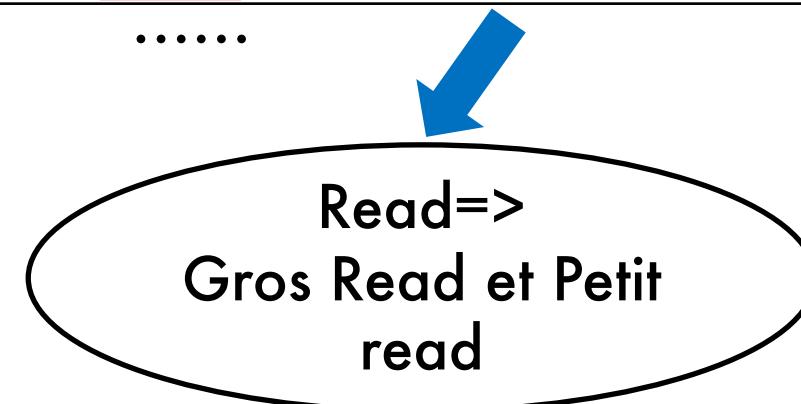
Faible granularité

Haute granularité

Agréger la charge de travail en tenant compte de la distribution, des utilisateurs et du contexte

Timestamp	Log line
00:00	Load data by Alice with size 32768 bytes
00:02	Read data by Alice with size 2048 bytes
00:03	Read data by Dan with size 1024 bytes
00:05	Read data by Alice with size 16384 bytes
00:06	Write data by Alice with size 8192 bytes

.....



Faible granularité

Haute granularité

Agréger la charge de travail en tenant compte de la distribution, des utilisateurs, du contexte et de la séquence

Timestamp	Log line
00:00	Load data by Alice with size 32768 bytes
00:02	Read data by Alice with size 2048 bytes
00:03	Read data by Dan with size 1024 bytes
00:05	Read data by Alice with size 16384 bytes
00:06	Write data by Alice with size 622 bytes

Ultramoderne

User comme Alice:

Séquence 1 (x%): Load=>gros read=>
petit read=> write

Faible granularité

Haute granularité

Les tests de charge réalistes sont basés sur des charges de travail (historiques), mais les charges de travail changent avec le temps

Ex: Le tremblement de Terre
Voir page 1



Conception de charges induisant des défauts

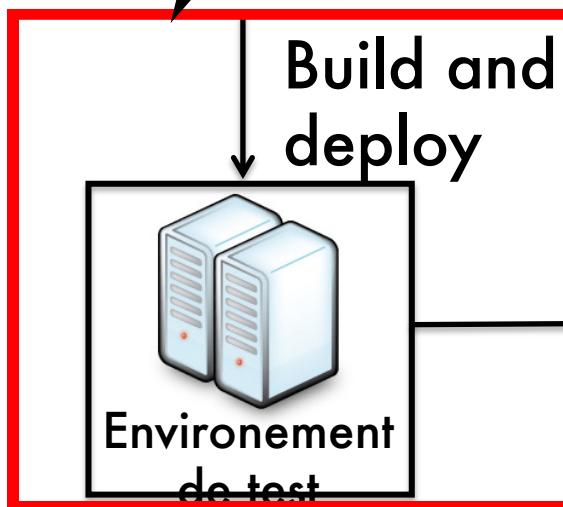
Analyse du code source - Analyse du flot de données

- Identifie les modules et les régions sensibles à la charge potentiels pour les défauts sensibles à la charge (par exemple, les fuites de mémoire et l'allocation de mémoire dynamique incorrecte)
- Annotation du graphe de flux de contrôle des appels malloc()/free() et de leurs tailles
- Load Sensitivity Index (LSI) indique l'augmentation/diminution nette de l'espace utilisé pour chaque itération
- Écrire des cas de test qui exercent les régions de code avec des valeurs LSI élevées

Les tests de performance nécessitent beaucoup de temps



Si on essaye de simuler ce qui s'est passé hier, on a besoin d'une autre journée pour simuler ce qui s'est passé hier



Produce at run-time

Réductions de la charge de conception - Extrapolation

Question: Pouvons-nous réduire l'effort et les coûts des tests de charge, lorsque le temps et les ressources matérielles/logicielles sont limités ?

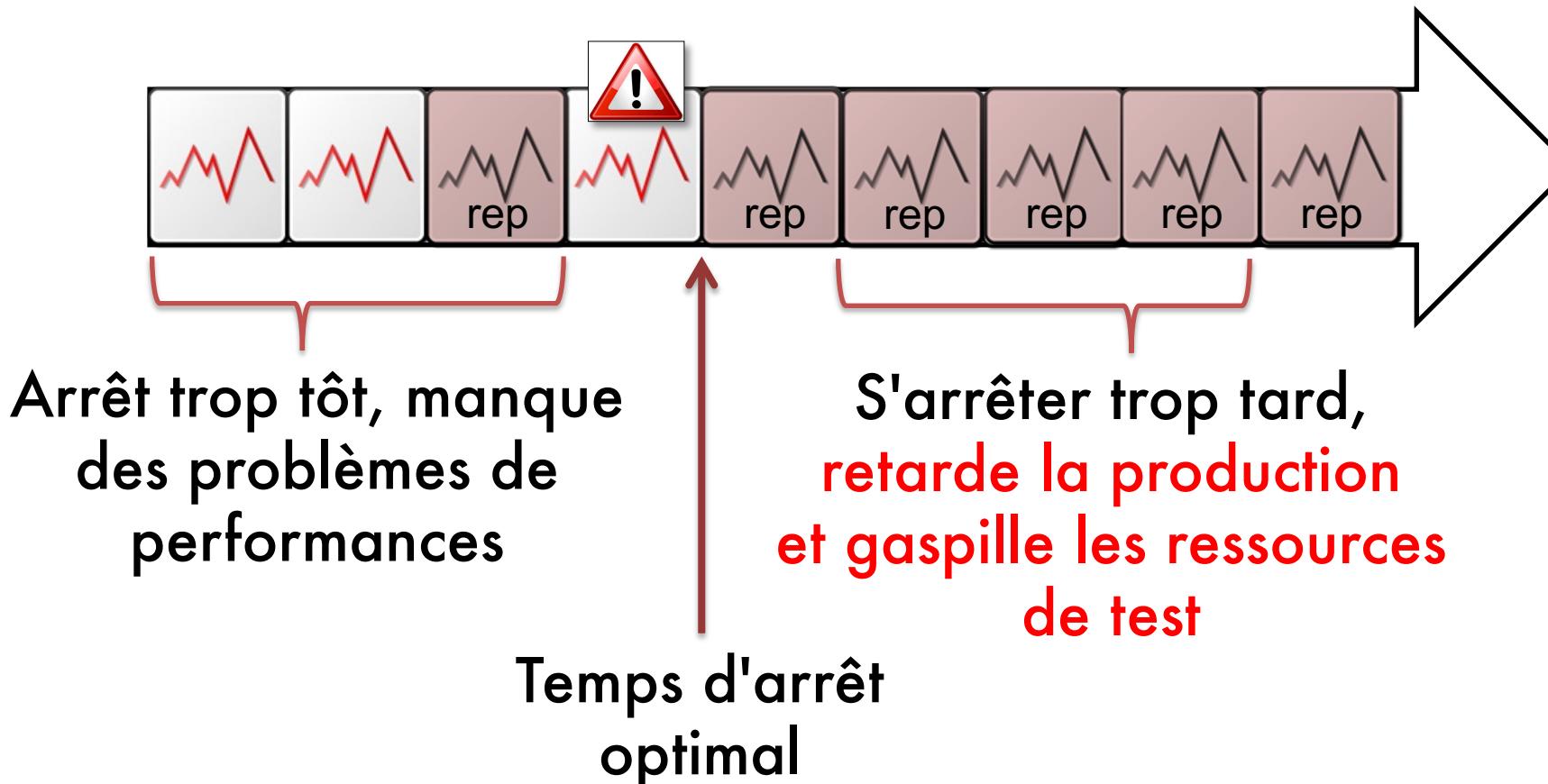
Extrapolation pour les tests de charge par étapes

- N'examinez que quelques niveaux de charge
- Extrapoler les performances du système pour les autres niveaux de charge

Ou d'une manière plus intelligente : arrêtez le test de charge lors de l'affichage de données répétitives



Approches automatisées pour réduire les tests de performance de longue durée



Mesurer la probabilité de répétitivité

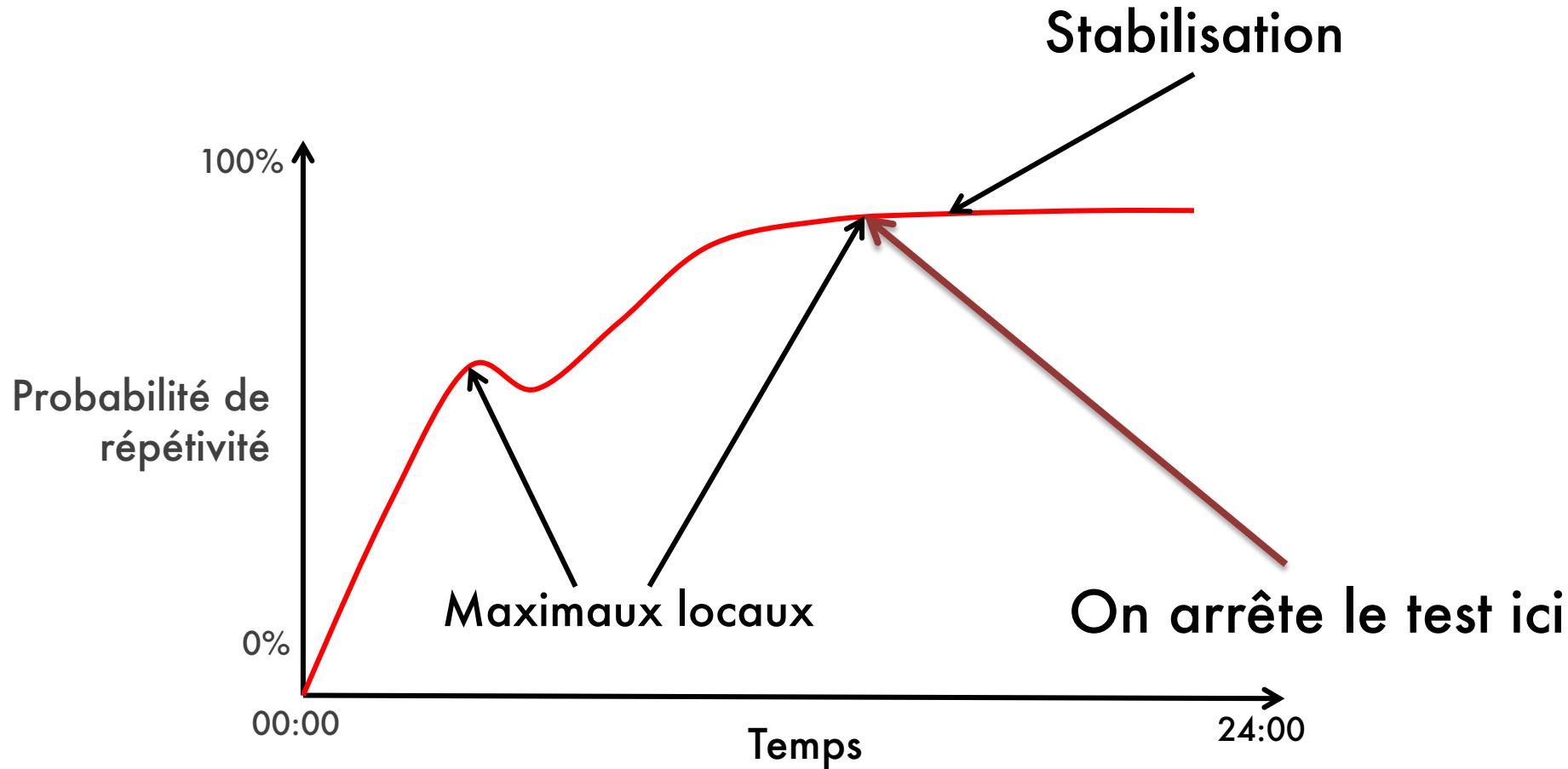
Répétez ce processus plusieurs fois
(par exemple, 1 000) pour calculer **la**
probabilité de répétitivité

Sélectionnez au hasard une
période de temps (par
exemple, 10 minutes)

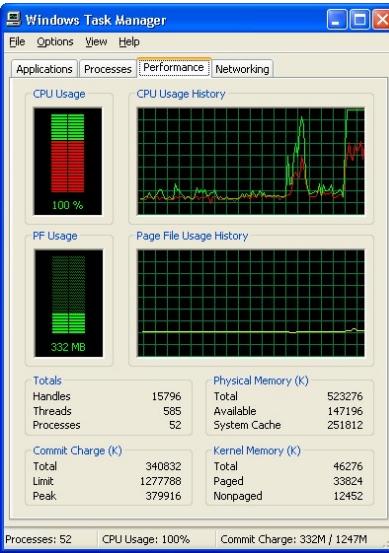
Heure actuelle

Rechercher une autre
période non chevauchante et
répétitive

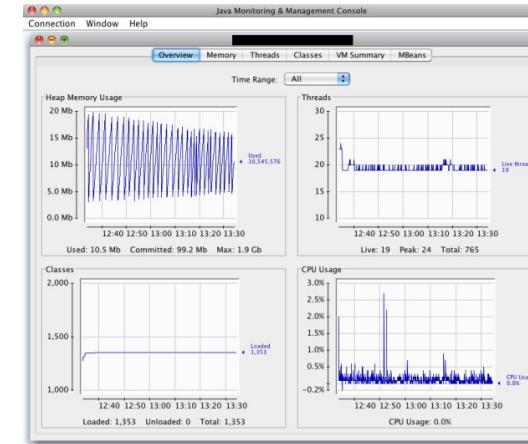
Search for the optimal stopping time during a test run



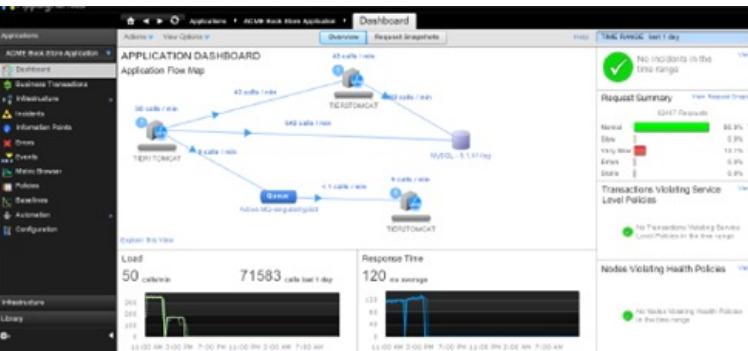
Outils de surveillance des tests



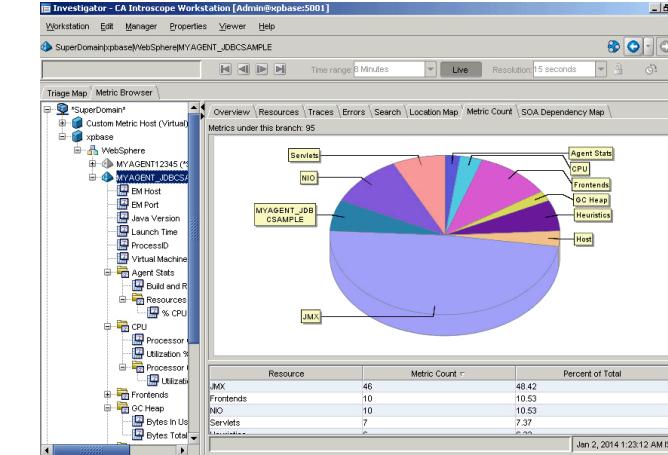
Task Manager



JConsole



App Dynamics



CA Willy

```

baban@hashprompt:~$ pidstat -u -r 1 1
Linux 3.2.0-36-generic (hashprompt)           Friday 15 February 2013   _x86_64_ (4 CPU)

03:24:49 IST      PID    %usr %system %guest   %CPU   CPU  Command
03:24:50 IST      1554   0.00   0.99   0.00   0.99   2 Xorg
03:24:50 IST      2708   2.97   0.00   0.00   2.97   0 cinnamon
03:24:50 IST      2731   0.00   0.99   0.00   0.99   0 gkrellm
03:24:50 IST      6231   0.99   0.00   0.00   0.99   3 chromium-browse
03:24:50 IST     17570   0.99   0.00   0.00   0.99   3 pidstat

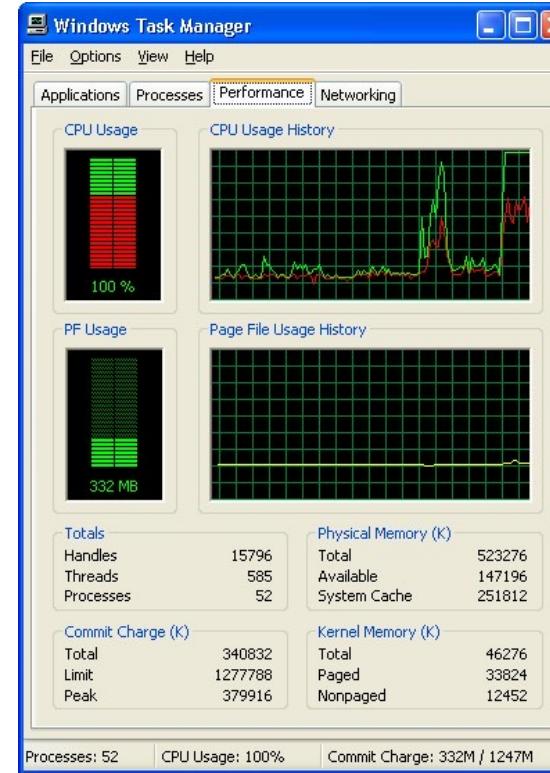
03:24:49 IST      PID minflt/s majflt/s   VSZ   RSS  %MEM  Command
03:24:50 IST     1939   114.85   0.00 107088 1260  0.03 cpufreqd
03:24:50 IST     2708     1.98   0.00 1673804 139240 3.63 cinnamon
03:24:50 IST     2731    11.88   0.00 578060 14500  0.38 gkrellm
03:24:50 IST     6231   100.99   0.00 1088184 165708 4.32 chromium-browse
03:24:50 IST    17570   369.31   0.00  95436 1072  0.03 pidstat

Average:      PID    %usr %system %guest   %CPU   CPU  Command
Average:      1554   0.00   0.99   0.00   0.99   - Xorg
Average:      2708   2.97   0.00   0.00   2.97   - cinnamon
Average:      2731   0.00   0.99   0.00   0.99   - gkrellm
Average:      6231   0.99   0.00   0.00   0.99   - chromium-browse
Average:     17570   0.99   0.00   0.00   0.99   - pidstat

Average:      PID minflt/s majflt/s   VSZ   RSS  %MEM  Command
Average:     1939   114.85   0.00 107088 1260  0.03 cpufreqd
Average:     2708     1.98   0.00 1673804 139240 3.63 cinnamon
Average:     2731    11.88   0.00 578060 14500  0.38 gkrellm
Average:     6231   100.99   0.00 1088184 165708 4.32 chromium-browse
Average:    17570   369.31   0.00  95436 1072  0.03 pidstat
baban@hashprompt:~$
```

pidstat

Exemples de surveillance sans agent



Task Manager



JConsole

PerfMon (Windows), sysstat (Linux), top

Exemples de surveillance programmable : PSUtil

```
>>> import psutil  
>>>  
>>> psutil.cpu_times()
```

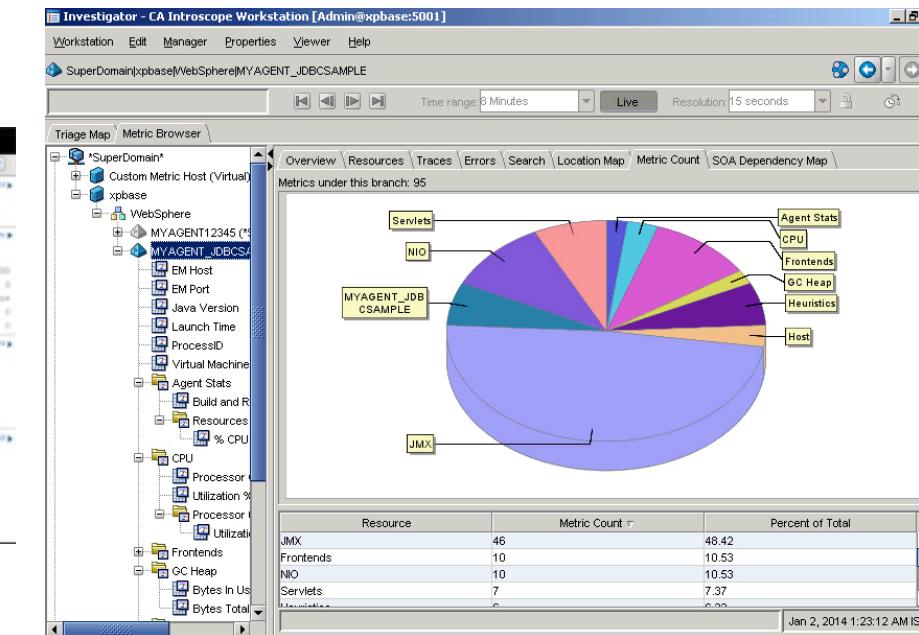
```
p = psutil.Process(7055)  
p.cpu_times()  
p.memory_info()
```

```
p = psutil.Popen(["/usr/bin/python", "-c", "print('hello')"],  
stdout=PIPE)
```

Exemples de surveillance basée sur un agent



App Dynamics



CA Willy

Dell FogLight, New Relic

Instrumentation

Instrumentation au niveau du code source

- Instrumentation manuelle ad-hoc,
- Instrumentation automatisée (par exemple, AspectJ), et
- Cadre d'instrumentation des performances (par exemple, l'API Application Response Time)

Cadre d'instrumentation binaire

- DynInst (<http://www.dyninst.org/>),
- PIN (<https://software.intel.com/en-us/articles/pin-a-binary-instrumentation-tool-downloads/>), et
- Valgrind (<http://valgrind.org/>)

Trace de système d'opération

- LTTng (lttng.org)

Biais de mesure

Le biais de mesure est difficile à éviter et imprévisible.

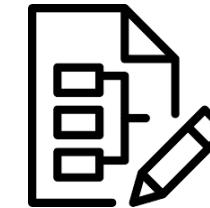
Exemple 1: Comment se fait-il que la même application s'exécute aujourd'hui plus rapidement qu'hier ? [Hier mon système utilisait le réseau](#)

Exemple 2 : Pourquoi le temps de réponse est très différent lors de l'exécution du même binaire sous différents comptes utilisateurs de la machine ? [Les différents utilisateurs sont en train de faire des choses différentes](#)

Exemple 3 : Pourquoi l'optimisation du code ne fonctionne que sur mon ordinateur ? [Mac vs Windows](#)

- Mesure répétée
- Randomiser la configuration des tests

Instrumentation vs logs



Instrumentation
Overhead

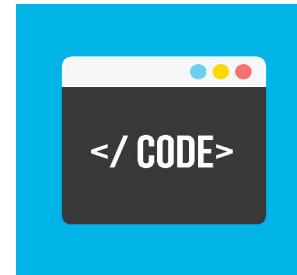
versus



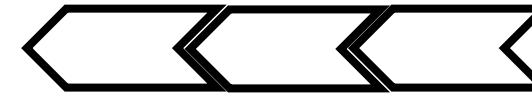
Informations
manquantes dans
les logs

Générez automatiquement
du code de log léger pour la
surveillance des
performances

Approche générale

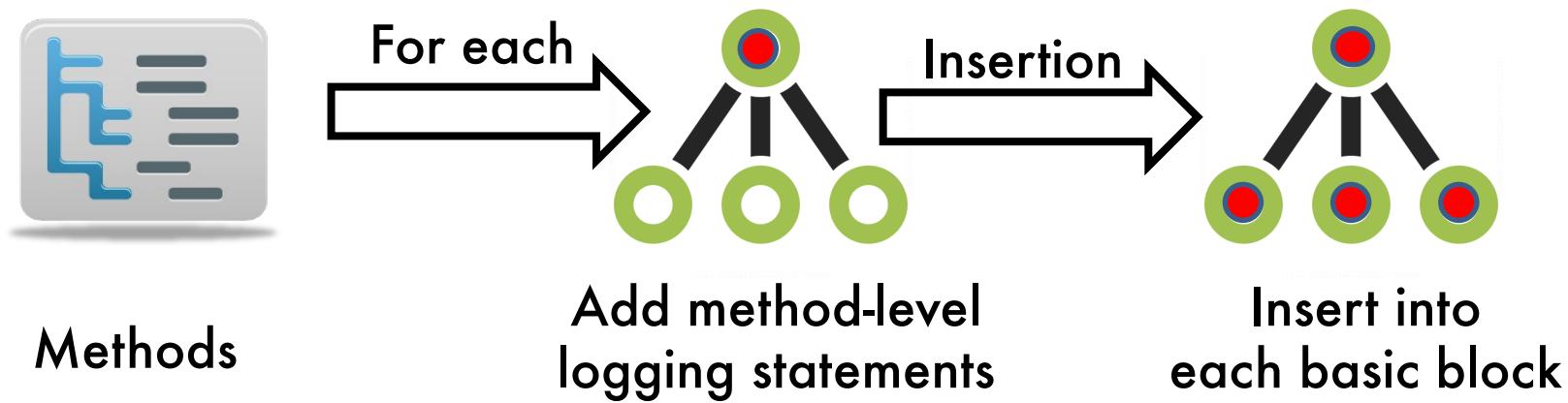


Source code



Logging
statements

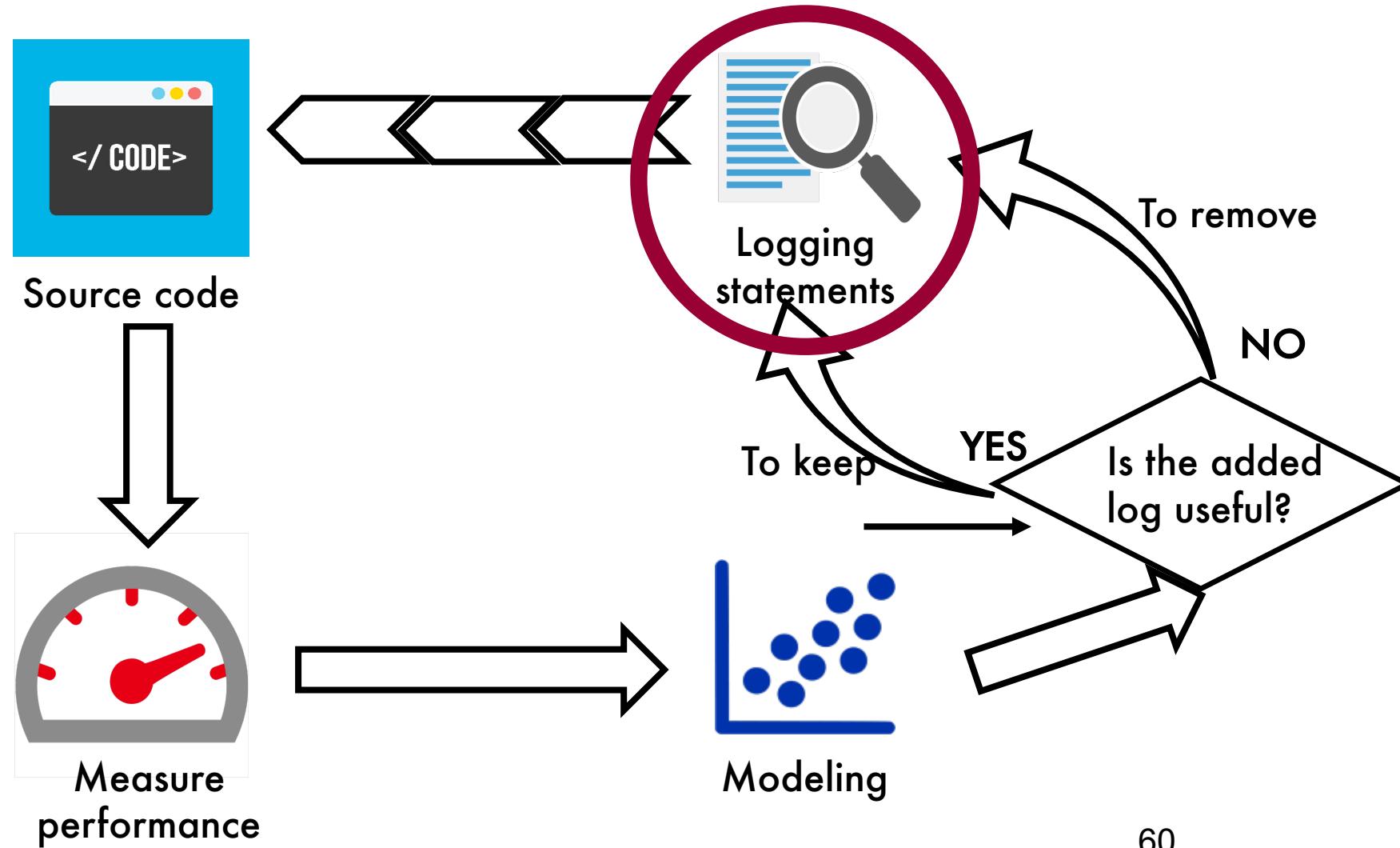
Insérer des instructions de logs dans des blocs de base

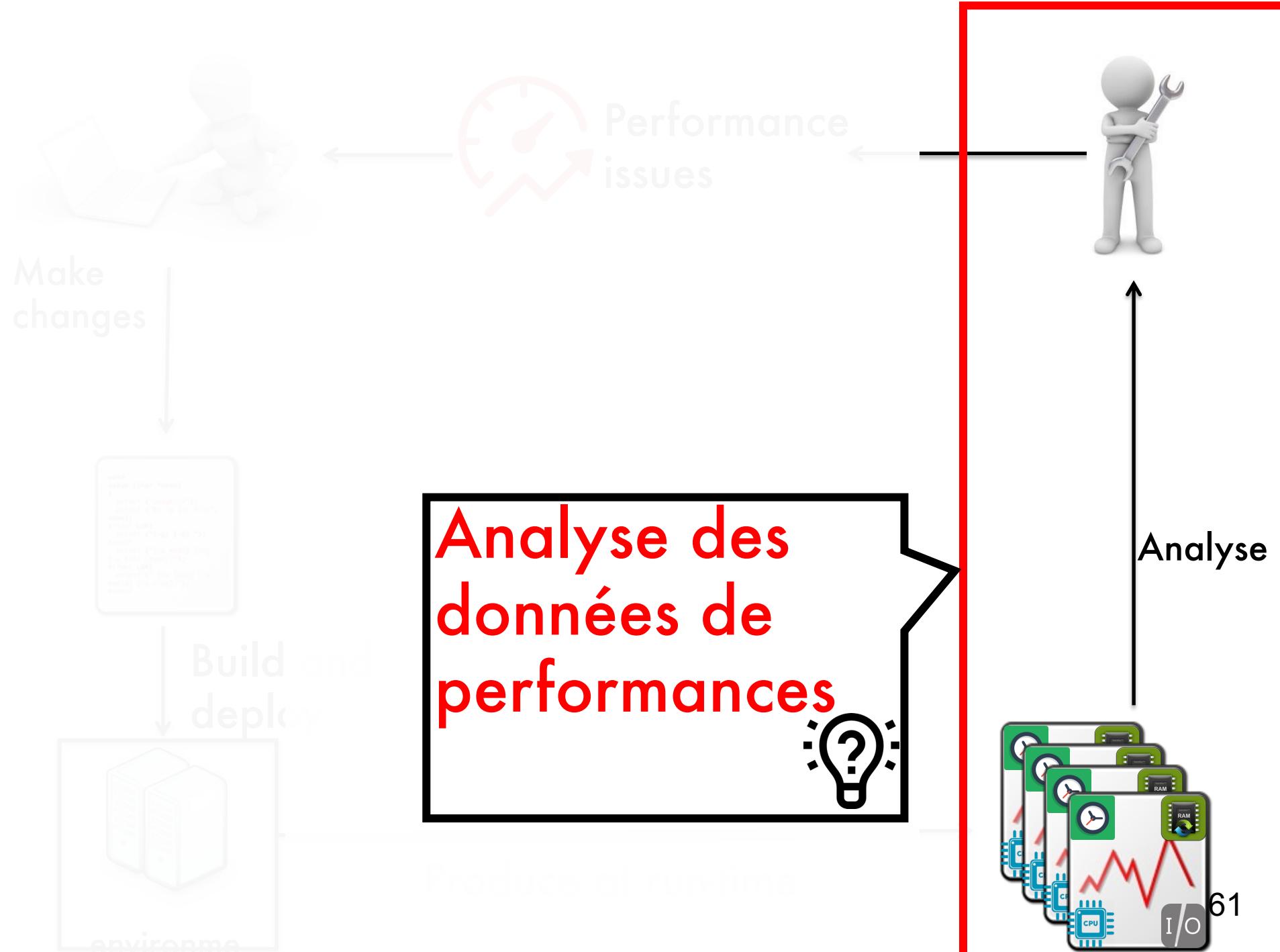


```

def foo()
{
    log()
    for index in range(x, y):
        if isEven():
            methodSay("Even")
}
// Method Body
    
```

Approche générale





Les systèmes logiciels génèrent de grandes quantités de données de performance



Compteurs d'échantillons

	A	B	C	D	E
1	Time	Disk Reads/sec	Disk Writes/sec	Page Faults/sec	Memory
2	2/29/08 16:58	0.049986394	0.000723659	0.003876542	3534848
3	2/29/08 17:01	0	0	0	3534848
4	2/29/08 17:04	0.060612225	0.027551011	0.016530607	3534848
5	2/29/08 17:07	0	0	0	3534848
6	2/29/08 17:10	0	0	0	3534848
7	2/29/08 17:13	0.060733302	0.027606046	0.016563628	3534848
8	2/29/08 17:16	0	0	0	3534848
9	2/29/08 17:19	0.060727442	0.027603383	0.01656203	3534848
10	2/29/08 17:22	0	0	0	3534848
11	2/29/08 17:25	0	0	0	3534848
12	2/29/08 17:28	0	0	0	3534848
13	2/29/08 17:31	0	0	0	3534848
14	2/29/08 17:34	0.121368621	0.055167555	0.038617289	3534848
15	2/29/08 17:37	0	0	0	3534848
16	2/29/08 17:40	0	0	0	3534848
17	2/29/08 17:43	0	0	0	3534848
18	2/29/08 17:46	0	0	0	3534848
19	2/29/08 17:49	0	0	0	3534848
20	2/29/08 17:52	0	0	0	3534848
21	2/29/08 17:55	0.121392912	0.055178596	0.033107158	3534848
22	2/29/08 17:58	0.060592703	0.027542138	0.02203371	3534848

Exemples de logs d'exécution

#	Log Lines
1	time=1, thread=1, session=1, receiving new user registration request
2	time=1, thread=1, session=1, inserting user information to the database
3	time=1, thread=2, session=2, user=Jack, browse catalog=novels
4	time=1, thread=2, session=2, user=Jack, sending search queries to the database
5	time=3, thread=1, session=1, user=Tom, registration completed, sending confirmation email to the user
6	time=3, thread=2, session=2, database connection error: session timeout
7	time=4, thread=1, session=1, fail to send the confirmation email, number of retry = 1
8	time=6, thread=2, session=2, user=Jack, successfully retrieved data from the database
9	time=7, thread=2, system health check
10	time=8, thread=1, session=1, registration email sent successfully to user=Tom
11	time=9, thread=2, session=3, user=Tom, browse catalog=travel
12	time=10, thread=2, session=3, user=Tom, sending search queries to the database
13	time=10, thread=3, session=4, user=Jim, updating user profile
14	time=11, thread=3, session=4, user=Jim, database error: deadlock

Vérification par rapport aux valeurs de seuil

Comparaison directe

- Par exemple, les valeurs de débit correspondent-elles à la cible ?

Comparaison avec les données traitées

- Maximum
- Médiane ou moyenne
- Valeur à 90 centiles

Comparaison avec les données dérivées

- Dérivation des seuils
 - Quel est le temps de réponse pour les versions précédentes ?
- Dérivation des données cibles
 - Quelle sera la fiabilité estimée ?



Détection des problèmes connus à l'aide de modèles

Modèles d'utilisation de la mémoire

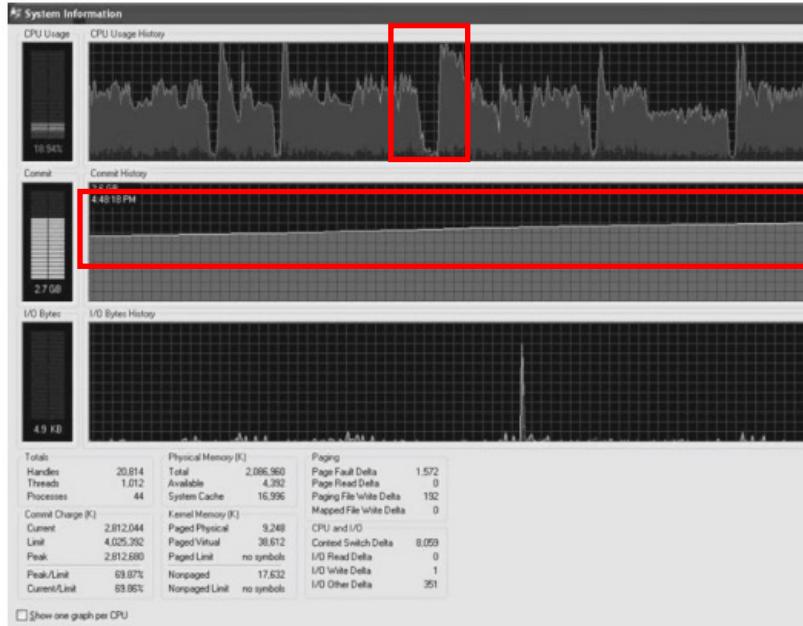
- détection de “Memory leak”

Modèles dans les logs

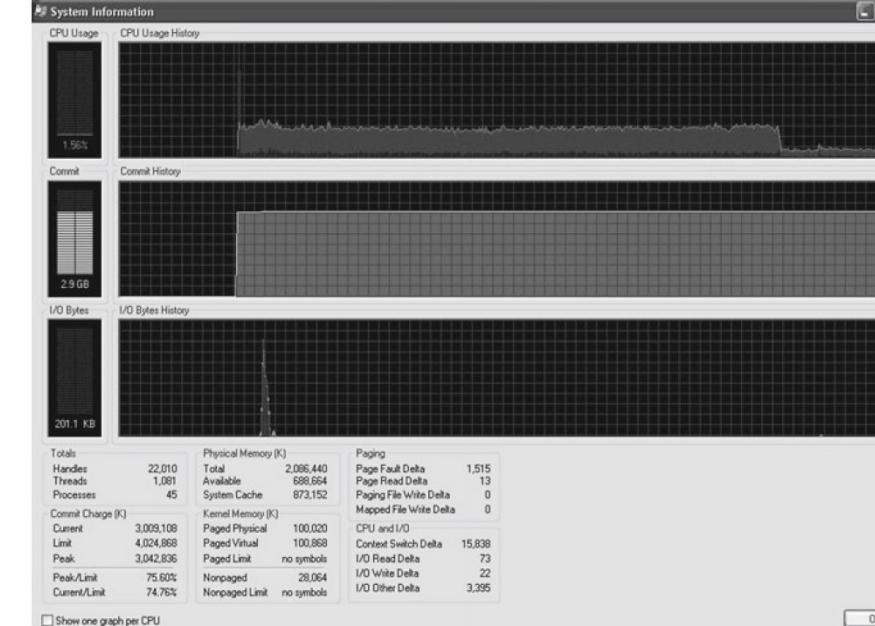
- Error keywords



Correspondance avec un modèle connu : interblocages et fuite de mémoire



Avant le correctif : baisse périodique du processeur et augmentation de la mémoire



Après réparation

[Avritzer et al., 2012]

Mots clés d'erreur

Un système d'entreprise à grande échelle peut générer 1,6 million de lignes de logs dans un test de charge de 8 heures

- 23,000 lignes contiennent “fail” ou “failure”
 - Combien de types d'échecs y a-t-il dans ce test ?
Il y a 3 de types d'échecs

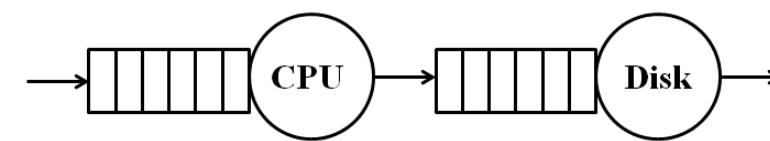
Événements	Fréquence
Error occurred during purchasing, item= <i>\$v</i>	500
Error! Cannot retrieve catalogs for user= <i>\$v</i>	300
Authentication error for user= <i>\$v</i>	100

Détection automatisée des comportements anormaux

Dérivez automatiquement le comportement "attendu/normal" et signalez les comportements anormaux

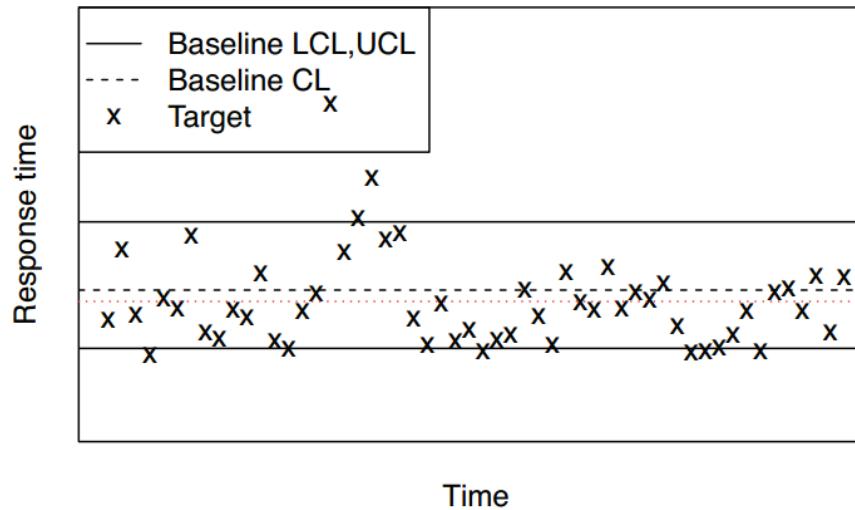


Data Mining

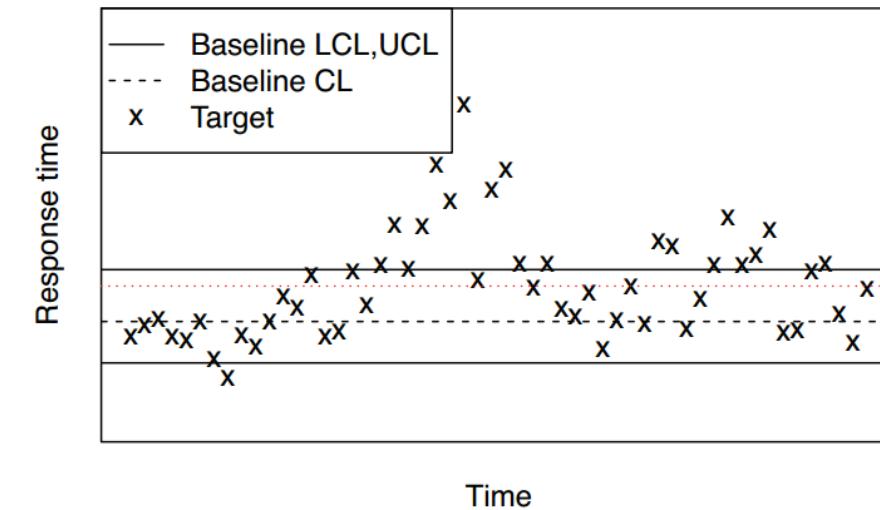


Théorie des files d'attente
(Queueing theory)

Dérivation des plages de performances à l'aide de graphiques



Fonctionnement Normal

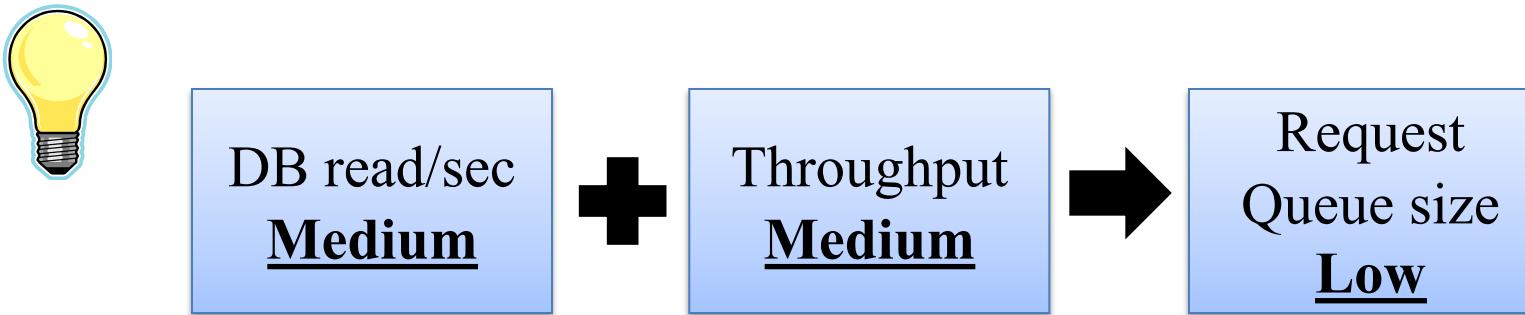


Test Suspect

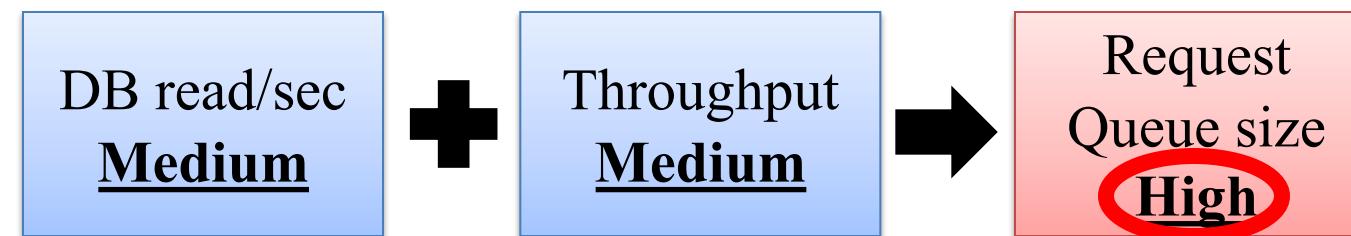
- Dériver des données des bons tests passés
- Signaler les nouveaux tests comme anormaux s'il y a de nombreuses violations

Dérivation automatisée des règles de performance

- Des tests passés, on peut extraire des règles telles que :



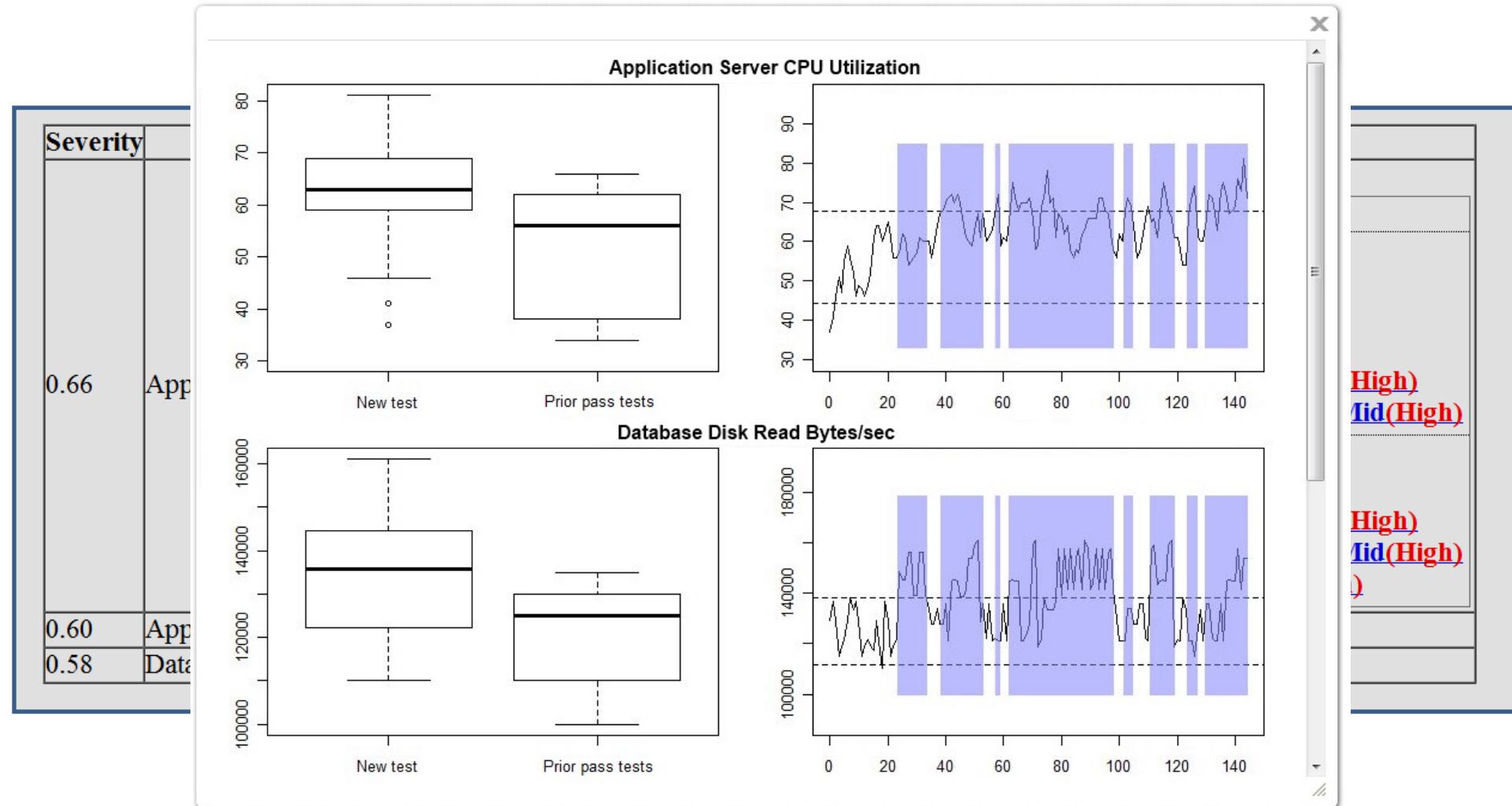
- Signale les tests où la règle ne tient pas



Rapport d'analyse de compteur

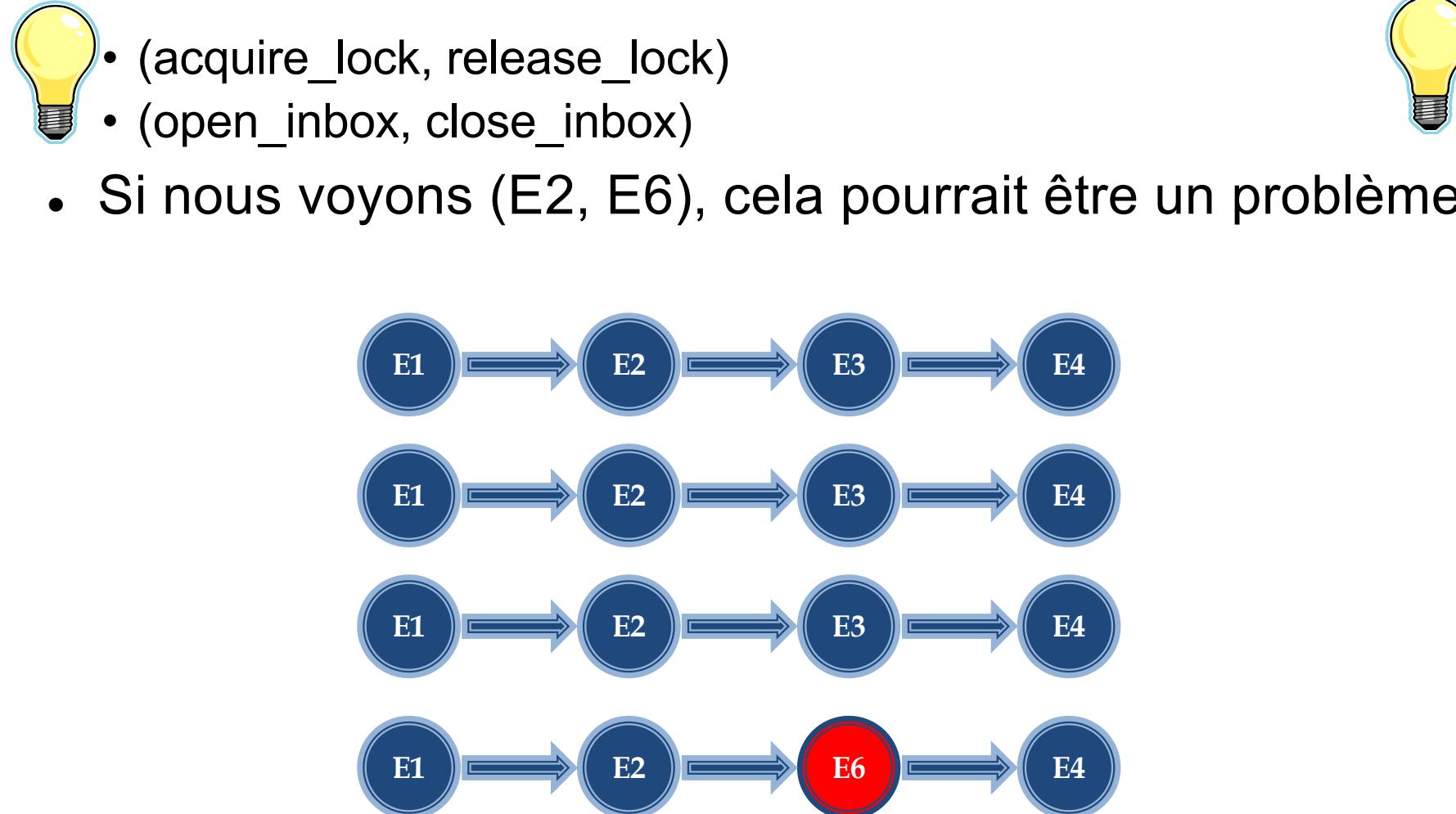
Severity	Performance Regression	Symptoms		
0.66	Application Server CPU Utilization	Hide Rules	Graph Conf. Change	Expected Correlation
		 0.79		Database Logical Disk Reads/sec=Mid Database Memory Page Writes/sec=Mid Database CPU Utilization=Mid Database Memory Page Reads/sec=Mid <u>Application Server CPU Utilization=Mid(High)</u> <u>Application Server Memory Utilization=Mid(High)</u>
		 0.65		Database Logical Disk Reads/sec=Mid Database Memory Page Reads/sec=Mid <u>Application Server CPU Utilization=Mid(High)</u> <u>Application Server Memory Utilization=Mid(High)</u> <u>Database Disk Read Bytes/sec=Mid(High)</u>
0.60	Application Server Memory Utilization	Show Rules		
0.58	Database Disk Read Bytes/sec	Show Rules		

Rapport d'analyse de compteur - Examiner la



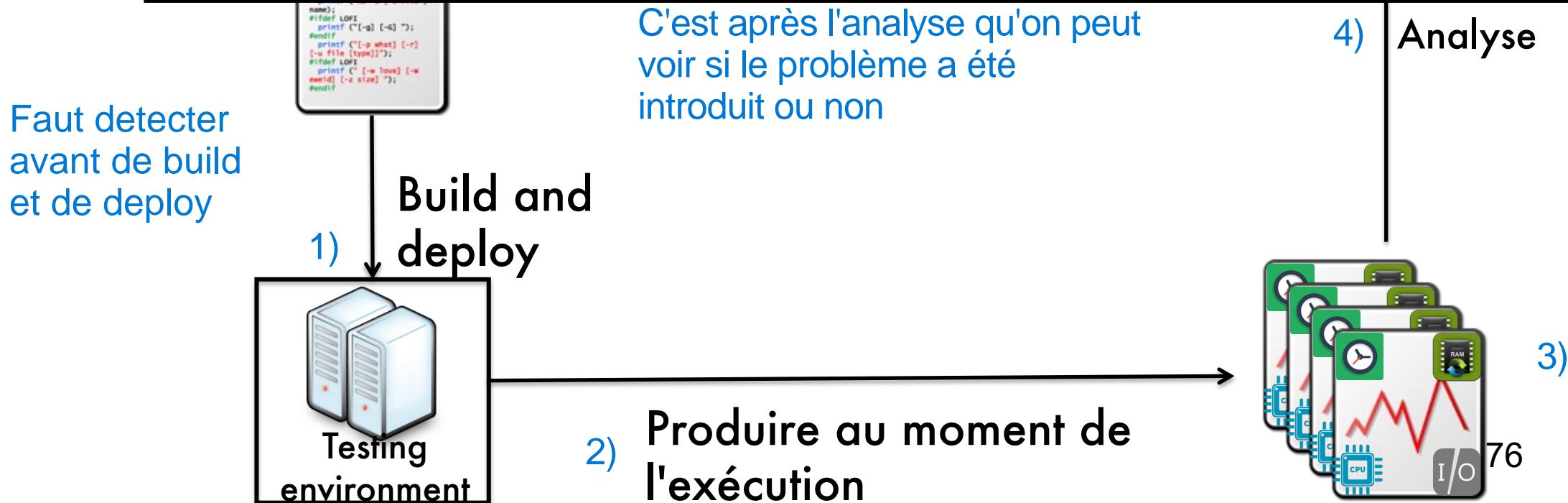
Analyse fonctionnelle automatisée

- (E2, E3) sont toujours ensemble :
 - (acquire_lock, release_lock)
 - (open_inbox, close_inbox)
- Si nous voyons (E2, E6), cela pourrait être un problème



Dérivation d'un comportement fonctionnel anormal

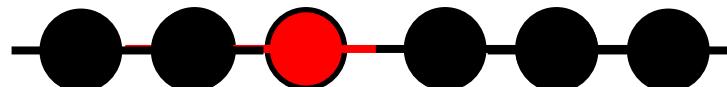
#	Z-Stat	Kinds	Min	Max	Total		Event
<u>SessionID=19420, Entering purchase for simple quantity queries</u>							
Freq		Sample		Details (Sort by Freq)			
87,528 (98%)		ds2logs.txt 688 ds2logs.txt 689		E13 --> SessionID=19420, Entering purchase for simple quantity queries E14 --> SessionID=19420, Initial purchase, update cart			
1,436 (<1%)		ds2logs.txt 2,484 ds2logs.txt 2,488		E13 --> SessionID=16242, Entering purchase for simple quantity queries E13 --> SessionID=16242, Entering purchase for simple quantity queries			
358 (<1%)		ds2logs.txt 10,020 ds2logs.txt 10,021		E13 --> SessionID=13496, Entering purchase for simple quantity queries E15 --> SessionID=13496, Finish purchase before commit			
E19	34.73	2	317	16,273	16,590	<u>SessionID=14128, End of purchase process</u>	
E22	20.65	2	1	3,857	3,858	<u>SessionID=12067, Purchase complete</u>	



Il est trop tard quand on détecte les problèmes de performances au niveau du build



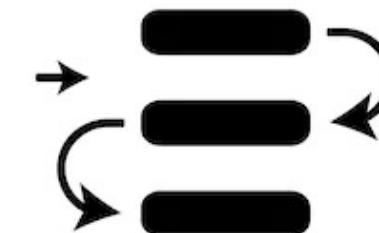
Difficile d'identifier la cause



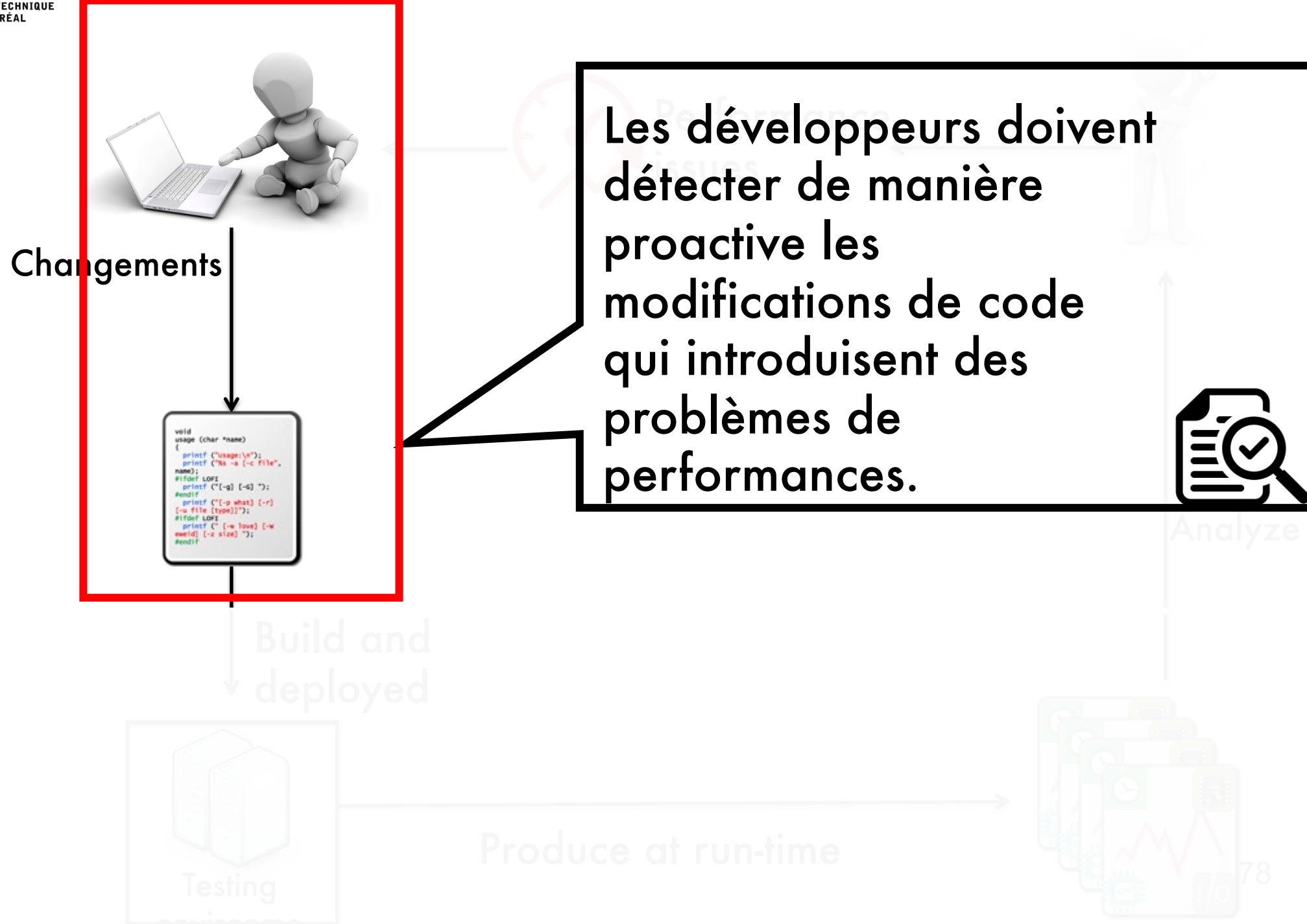
Trop de changements de code



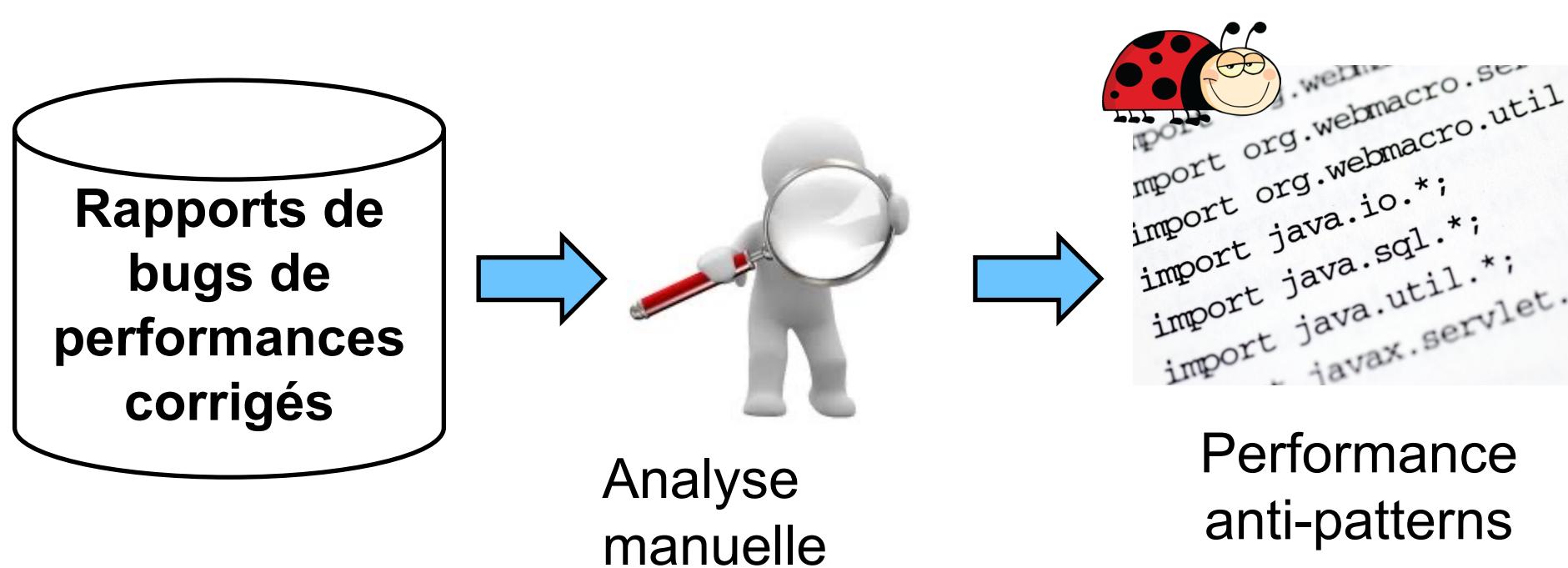
Difficile de trouver un expert pour enquêter



D'autre code peut déjà dépendre du code bogué



Autres moyens de détecter les problèmes de performances



Performance anti-patterns

Source code

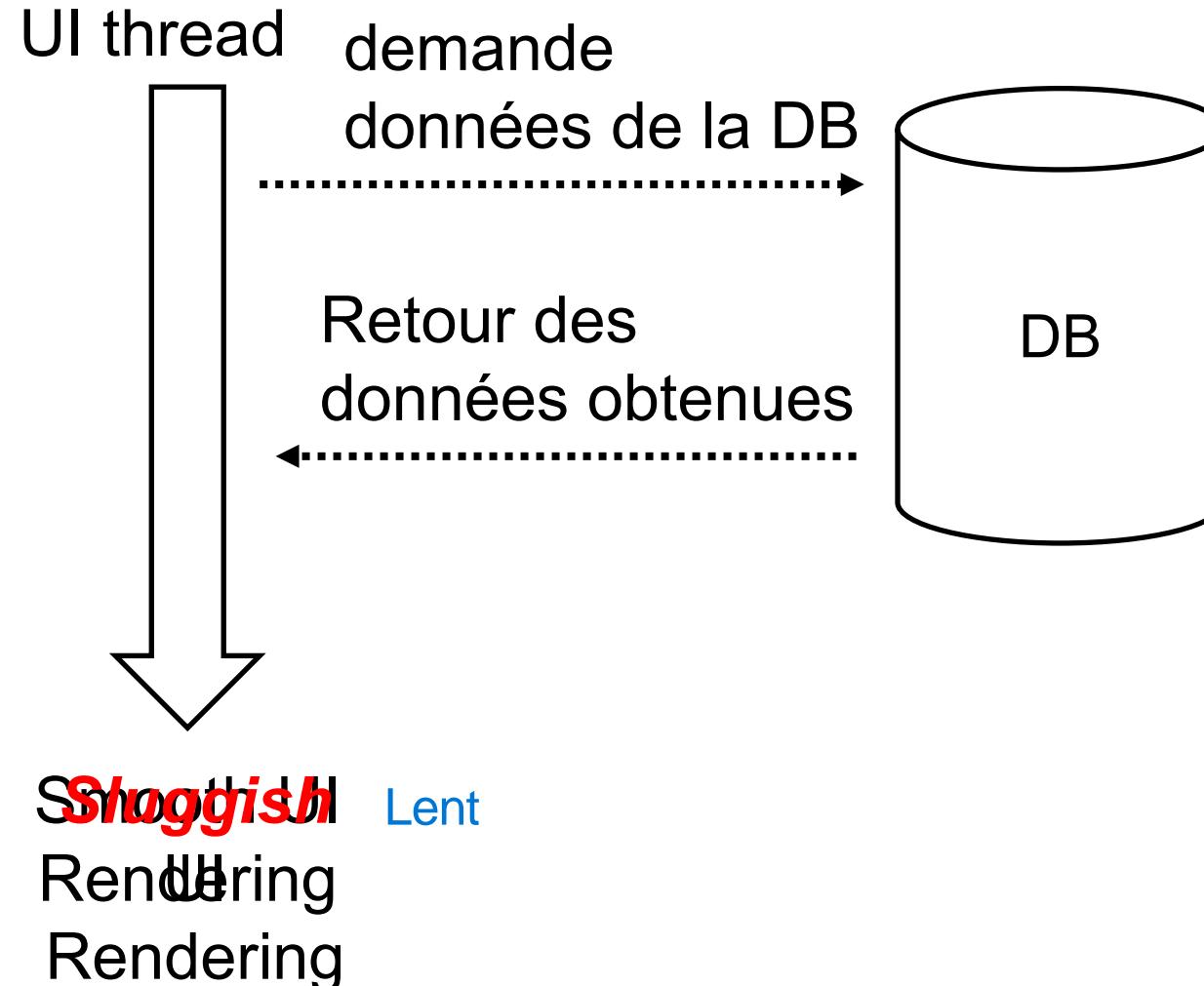
```
for (Company c: companyList){  
    for (Department d:c.getDepartments()){  
        ...  
        String query= "select department as d  
        where      d.companyID=''" +c.id+ "' and  
        d.name'"=  d.getDepartmentName()+"'";  
        ...  
        stmt.executeQuery(query);  
    }  
}
```

SQL

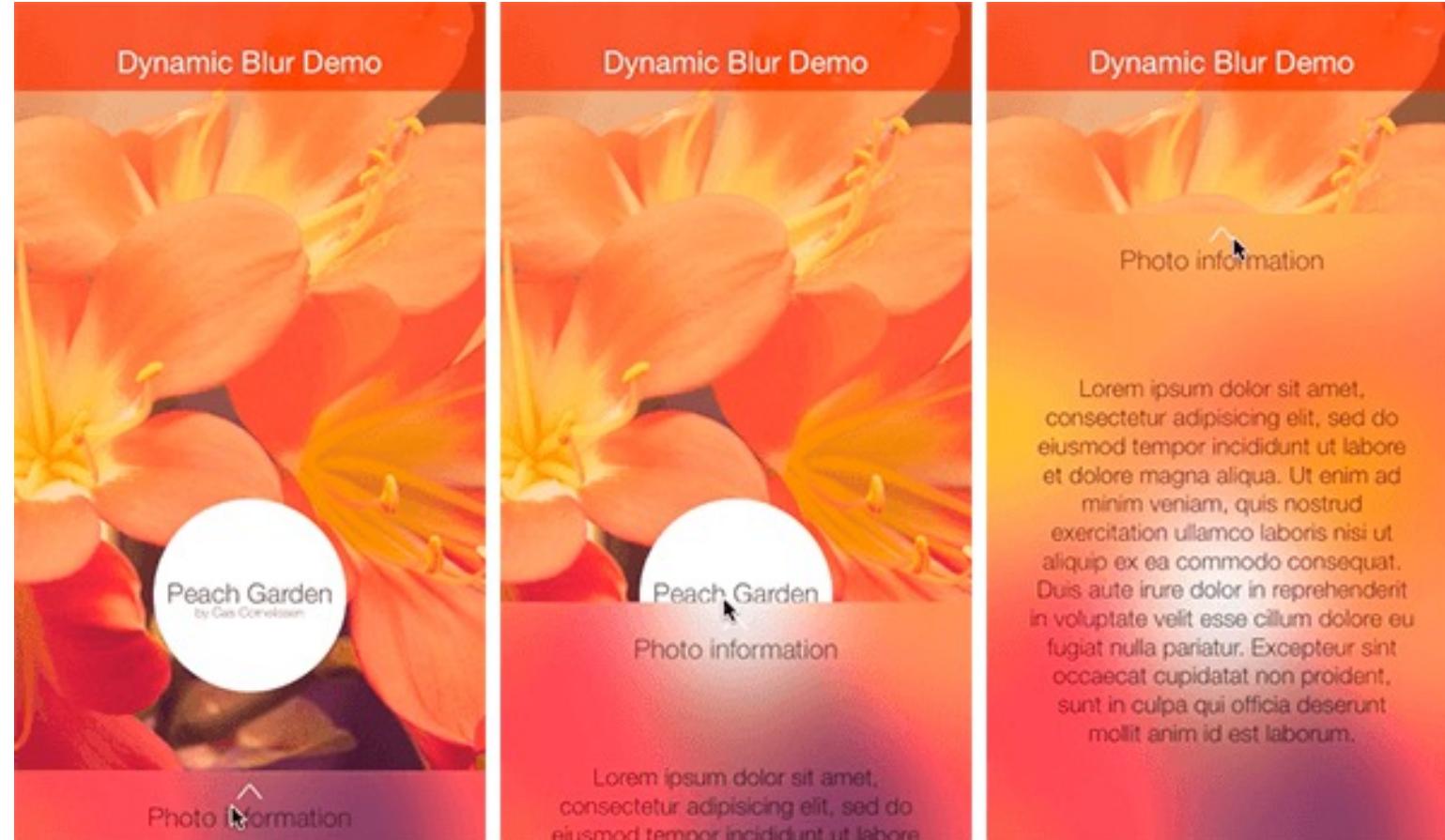
select department as d where d.companyID=1
select department as d where d.companyID=2
....



EXEMPLE : Accéder aux données sur le thread d'interface utilisateur



Application inefficace des effets de IU



Les effets de “blur” peuvent être coûteux!

Application inefficace des effets de IU

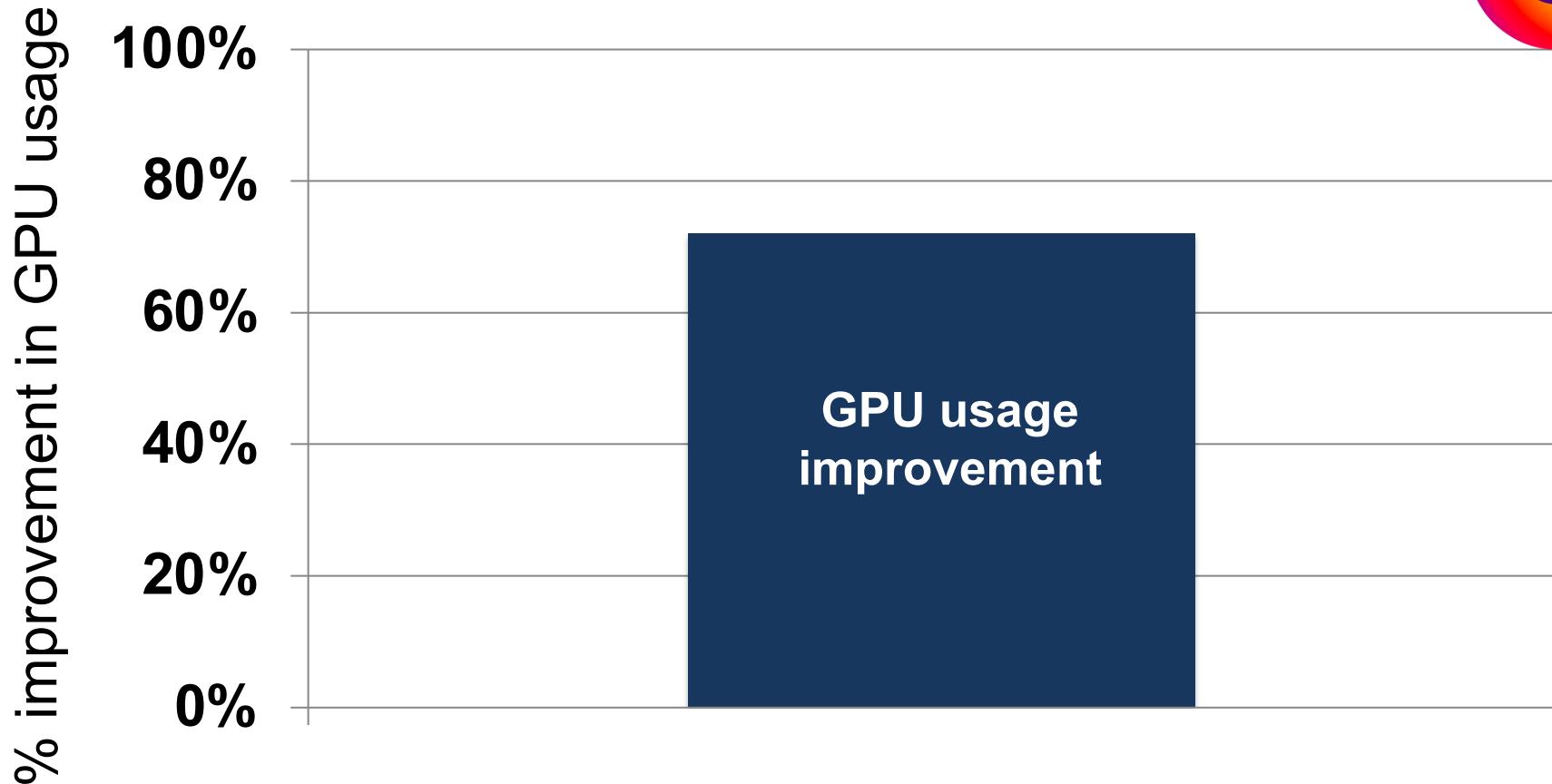
Configuring blur effects

```
• lazy var backgroundEffect: UIVisualEffectView? = {  
•     let blur = UIBlurEffect(style: UIBlurEffectStyle.Light)  
•     let vib = UIVibrancyEffect(forBlurEffect: blur)  
•     let effect: UIVisualEffectView? = DeviceInfo.isBlurSupported() ?  
    UIVisualEffectView(effect: blur) : nil  
•     return effect  
• }()
```

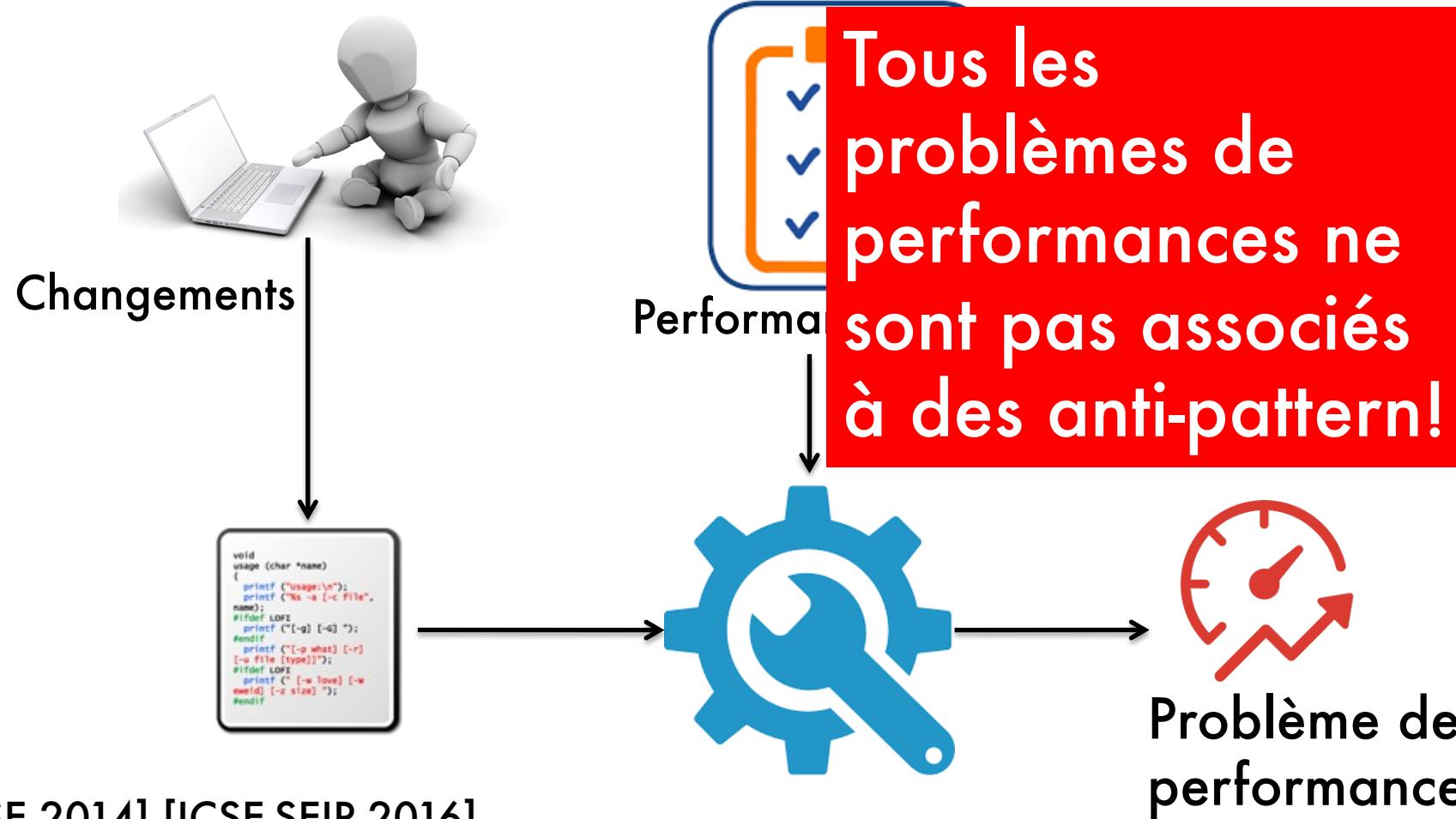
```
override init(frame: CGRect) {  
    ...  
    imageWrapper.addSubview(backgroundEffect)  
    ...  
    contentView.addSubview(imageWrapper)  
    ...  
}
```

Ajouter des effets de flou à chaque cellule d'un tableau

Exemple d'amélioration : suppression de l'effet lourd de l'IU sur l'utilisation du GPU



Détecter automatiquement les anti-pattern de performance dans le code source



A close-up photograph showing a person's hand wearing a white nitrile glove. The hand is holding a red plastic test tube that is tilted diagonally. Inside the tube, there is a dark, viscous liquid, likely blood or a similar sample. The background is dark and out of focus.

**Testing is still needed to detect
performance issues.**

Ce serait bien si nous pouvions nous permettre d'exécuter des tests de performance pour chaque commit



**Micro-performance
benchmarks!**

Les matériaux sont inspirés du travail de:

Weiyi Shang

Zhen Ming Jiang

Ahmed Hassan

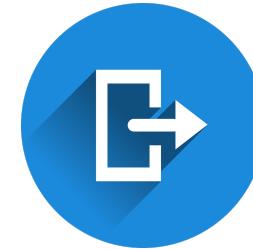
Derek Foo

Haroon Malik

Mark Syer

Parminder Flora

Journalisation de logiciel (Software Logging)



Quel est le profil des utilisateurs les applications, cmt il utilisent

LOG3430 Méthodes de test et de validation du logiciel

Qu'est-ce que la journalisation et à quoi sert-elle ?

Utilise du code pour afficher dynamiquement les informations d'un système

```
if (logger.isDebugEnabled()) {  
    logger.debug("While processing UpdateAttributes message ignored incoming profile: {}",  
    this);  
}
```

Qu'est-ce que la journalisation et à quoi sert-elle ?

Exemple de log pour « info » d'un système

```
2021-06-10 21:31:02,527 INFO FSNamesystem.audit: allowed=true ugi=hadoop (auth:SIMPLE) ip=/192.168.8.176 cmd=getfileinfo src=/tmp  
2021-06-10 21:31:02,558 INFO FSNamesystem.audit: allowed=true ugi=hadoop (auth:SIMPLE) ip=/192.168.8.176 cmd=listStatus src=/tmp  
2021-06-10 21:31:02,563 INFO FSNamesystem.audit: allowed=true ugi=hadoop (auth:SIMPLE) ip=/192.168.8.176 cmd=delete src=/tmp
```



Avantages de la journalisation

- Aide à diagnostiquer les pannes d'exécution
- On peut utiliser les « logs » comme débogueur
- Peut être utilisée pour assistance d'utilisateur/client
- Aide à la compréhension du système



Désavantages de la journalisation

- Impact sur la performance du système
- Peut exposer des informations sensibles
- On doit traiter de grandes quantités de données
- Peut provoquer des erreurs (exception NullPointerException) si géré incorrectement
- Rend le code plus difficile à lire
- A un coût d'entretien



Les « logs » peuvent avoir plusieurs formes

- CSV, HTML, JSON, Text Pattern, Syslog ...
- Stockage de données
 - Mongo, CouchDB, Cassandra
- Kibana and Elasticsearch

Les « logs » peuvent avoir plusieurs formes

Peuvent être non structuré

- Phrases textuelles non structurées qui nécessitent un traitement ultérieur
- Grep, sed, awk, perl = beurk
- Difficile de trouver des informations/champs similaires dans les données

```
unstructured_data = ["Unstructured message", "Hello Python  
World", str(datetime.now(timezone("EST"))).isoformat()]
```

Les « logs » peuvent avoir plusieurs formes

Peuvent être structurés

- Contiennent les "données réelles" du contexte au lieu de simples messages
- Tailles de fichiers, latences, etc. (tout est structuré)
- Peut rechercher facilement des données structurées similaires, par exemple toutes les tailles de fichiers

```
2021-06-10 21:31:02,527 INFO FSNamesystem.audit: allowed=true ugi=hadoop (auth:SIMPLE) ip=/192.168.8.176 cmd=getfileinfo src=/tmp
2021-06-10 21:31:02,558 INFO FSNamesystem.audit: allowed=true ugi=hadoop (auth:SIMPLE) ip=/192.168.8.176 cmd=listStatus src=/tmp
2021-06-10 21:31:02,563 INFO FSNamesystem.audit: allowed=true ugi=hadoop (auth:SIMPLE) ip=/192.168.8.176 cmd=delete src=/tmp
```

Format et modèles de « logs »

- ```
<Pattern>%d %p %c{1.} [%t] %m%n</Pattern>
```
- %d = date
  - %p = level (eg. ERROR)
  - %c = name of logger
    - %c{1.} = org.apache.commons.Foo -> o.a.c.Foo
  - %t = name of the thread
  - %m = the message
  - %n = new line

# Format et modèles de « logs »

**Log4j**  
**Levels**

| Event Level | LoggerConfig Level |       |      |      |       |       |     |
|-------------|--------------------|-------|------|------|-------|-------|-----|
|             | TRACE              | DEBUG | INFO | WARN | ERROR | FATAL | OFF |
| ALL         | YES                | YES   | YES  | YES  | YES   | YES   | NO  |
| TRACE       | YES                | NO    | NO   | NO   | NO    | NO    | NO  |
| DEBUG       | YES                | YES   | NO   | NO   | NO    | NO    | NO  |
| INFO        | YES                | YES   | YES  | NO   | NO    | NO    | NO  |
| WARN        | YES                | YES   | YES  | YES  | NO    | NO    | NO  |
| ERROR       | YES                | YES   | YES  | YES  | YES   | NO    | NO  |
| FATAL       | YES                | YES   | YES  | YES  | YES   | YES   | NO  |
| OFF         | NO                 | NO    | NO   | NO   | NO    | NO    | NO  |

# « Logging » à l'ancienne

## Utiliser le système de fichiers pour la journalisation

- + Le système de fichiers est robuste
- + Le système de fichiers est bien connu
  
- l'utilisation à grande échelle est difficile
- Si la machine meurt, vous perdrez tous les « logs »
- Nécessite une stratégie de « rotation » et expiration

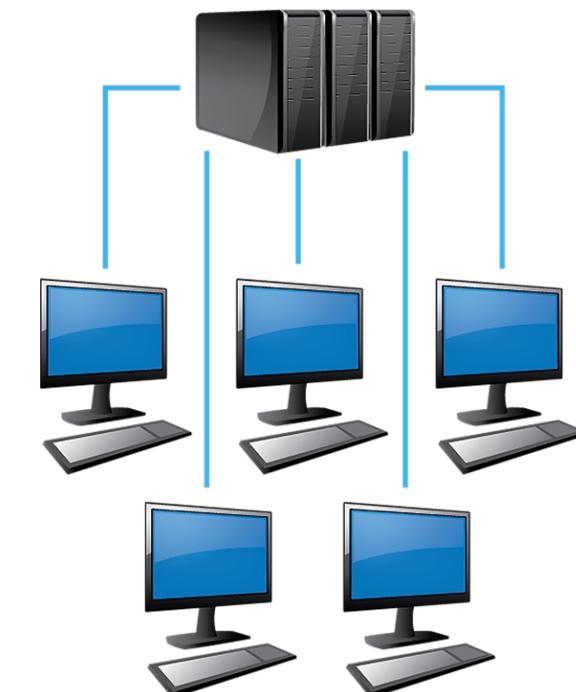


# « Logging » moderne

## Syslog (ancien)

Transférer les journaux vers un agrégateur

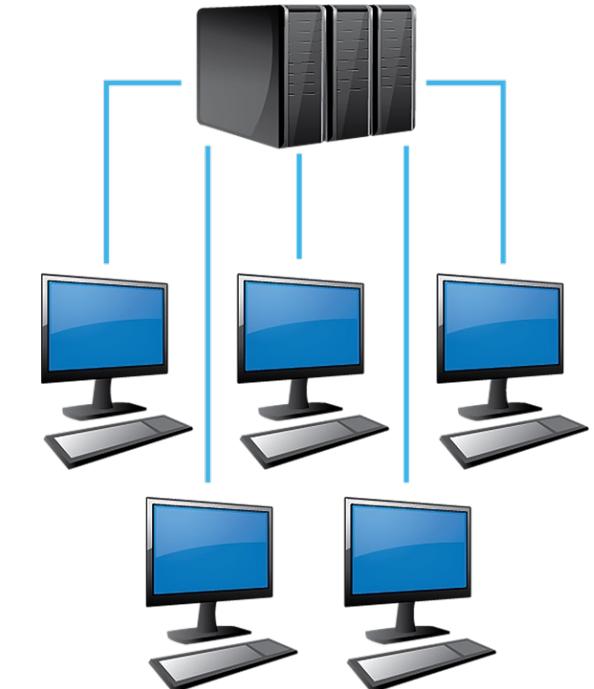
- + Lieu central unique, vous n'avez pas à surveiller que les machines individuelles manquent de stockage
- Fichiers fixes et outils unix "primitifs"



# « Logging » moderne

Agréger dans un moteur de recherche (e.g., Lucene)

- +Interface unique pour toutes les données de journal
- +La langue de recherche est plus expressive que grep/regexp  
*(La plupart des gens ne comprennent pas réellement les regex)*
- +Les analyses ne se contente plus de simplement « trouver » des choses
  - ex., Lucene a des statistiques de base intégrées
  - Index de texte intégral



# Que devons-nous enregistrer dans nos « logs »?

**enregistrez tout ce que vous pensez être important**

Aucune différence entre les données importantes et non importantes

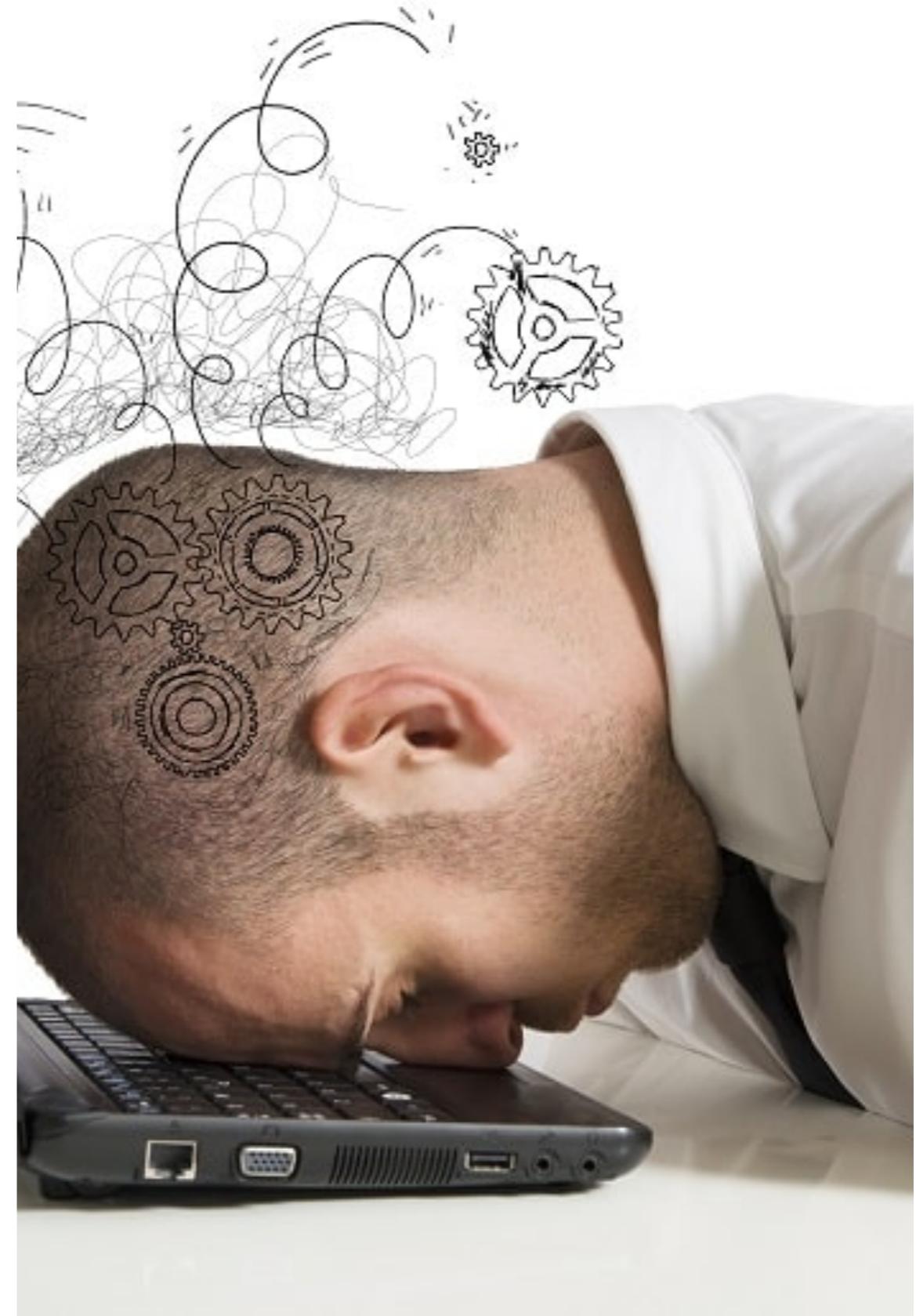
- + Tous les journaux ont un « timestamp »
- + Temps réel
- + On utilise des systèmes d'analyse statistiques et pas seulement recherche de mots clés
- Données semi-structurées

**Vous êtes maintenant  
prêt à faire l'activité sur  
Moodle**

# **LOG3430: Tests Métamorphiques**

Basée sur le matériel de G.  
Antoniol,

traduction de “Testing Definition  
and Context” par Maxime  
Lamothe



# Tester

---

- Ensemble d'activités pour vérifier si le(s) résultat(s) réel(s) d'un système/composant/programme correspondent aux résultat(s) attendu(s)
- Tests de logiciels:
  - Exécuter un système/composant/programme avec des données d'entrée et **vérifier le(s) résultat(s) obtenu(s) par rapport au(x) résultat(s) attendu(s)** dans un environnement pré-déterminé.

# Que se passe-t-il dans cette image?

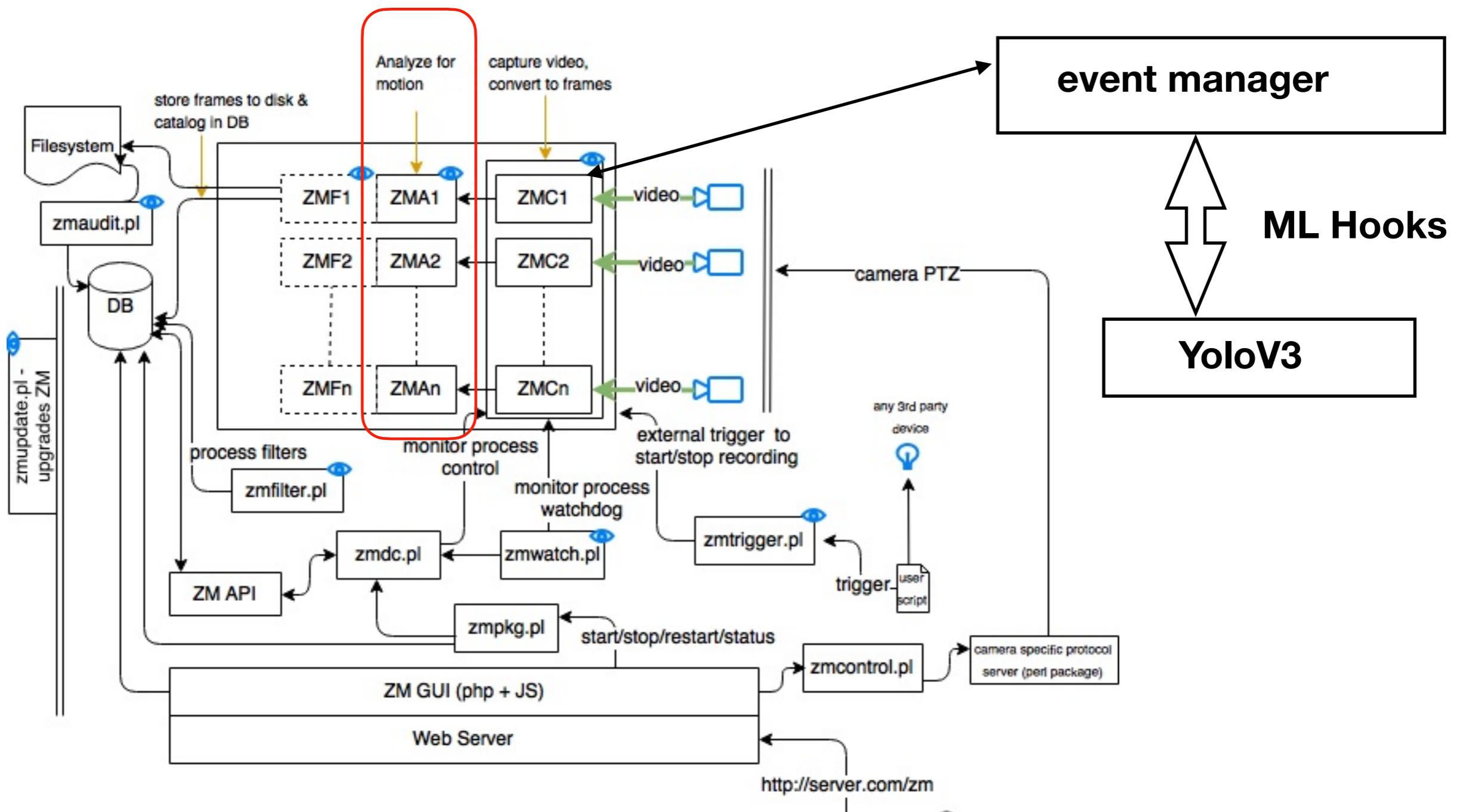
Un homme marche, fait demi-tour et se rassoit



# Un test



# Tester...quoi?



# **Tester : contexte et notions de base**

---

# Éléments clé

- Exécuter: un système/composant/programme

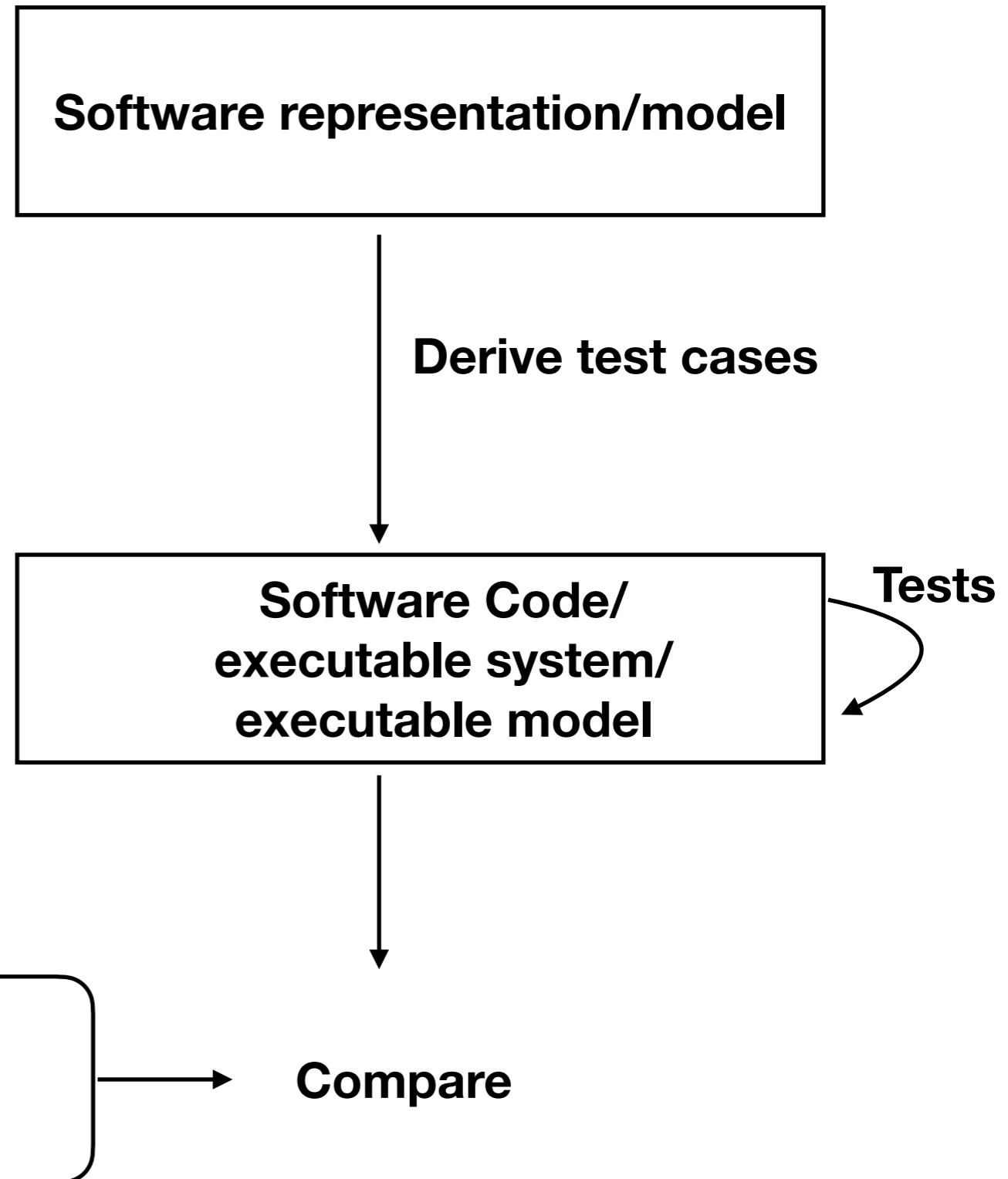
- Dans un environnement prédéterminé

- Données d'entrée

- Écrites manuellement ou générées automatiquement ou ..*générées par les users*

- Comparer les résultats aux résultats attendus

- Nous devons avoir un oracle



# Qu'est-ce qu'on teste?

---

**Tout système/composant/programme a plusieurs attributs de qualité**

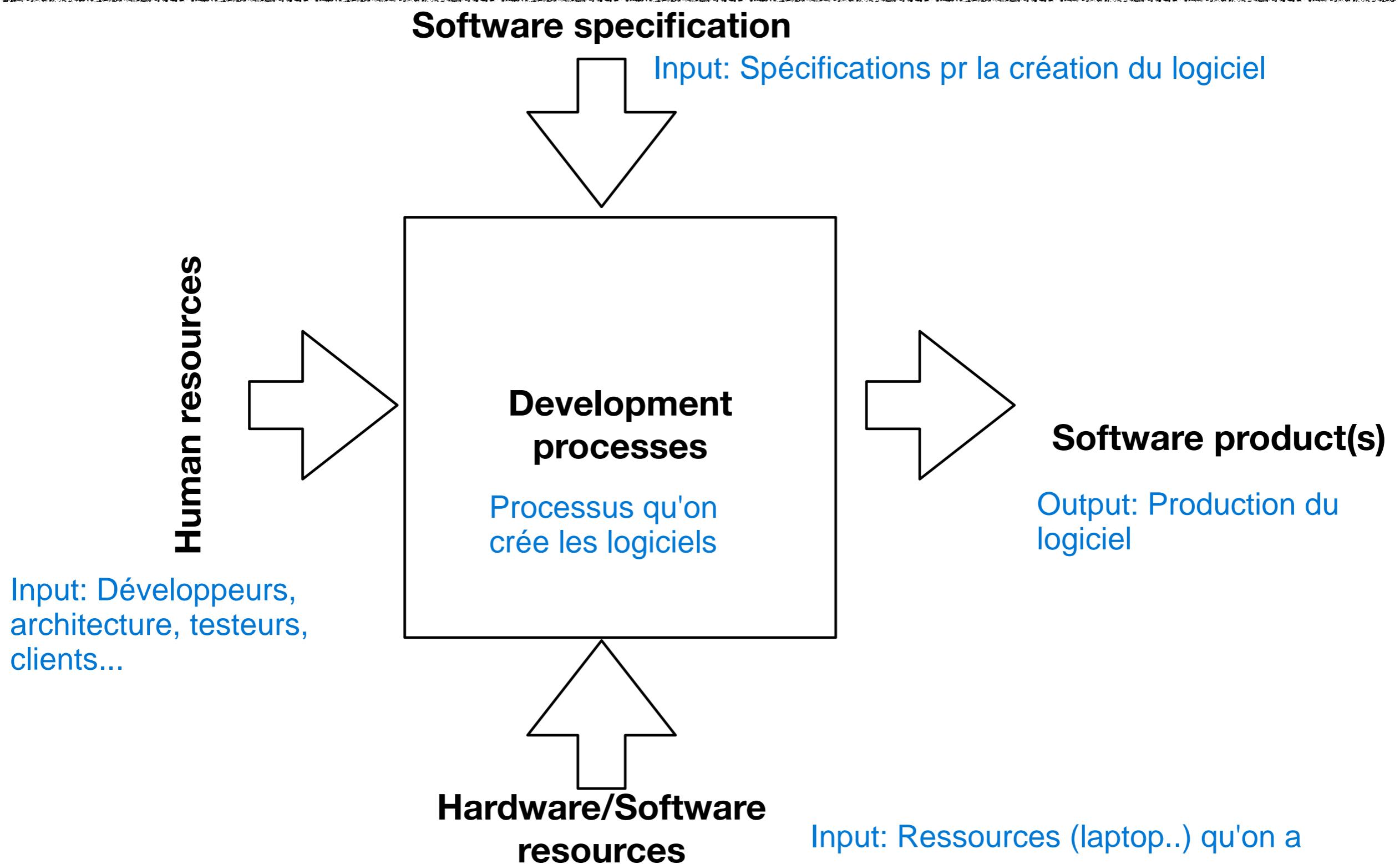
- Exactitude
- Fiabilité
- Robustesse
- Performance
- Convivialité
- Vérifiabilité
- Maintenabilité
- Etc.
- Repairability
- Evolvability
- Reusability
- Portability
- Understandability
- Interoperability

# Tester pour trouver des défauts

---

- Tester est souvent synonyme de tester pour identifier des défauts ou des comportements inattendus (Exactitude)
  - Tester nécessitent des spécifications précises, complètes et non ambiguës
- Autres types de tests: test de performance, utilisation de mémoire, portabilité, robustesse, sécurité, ...

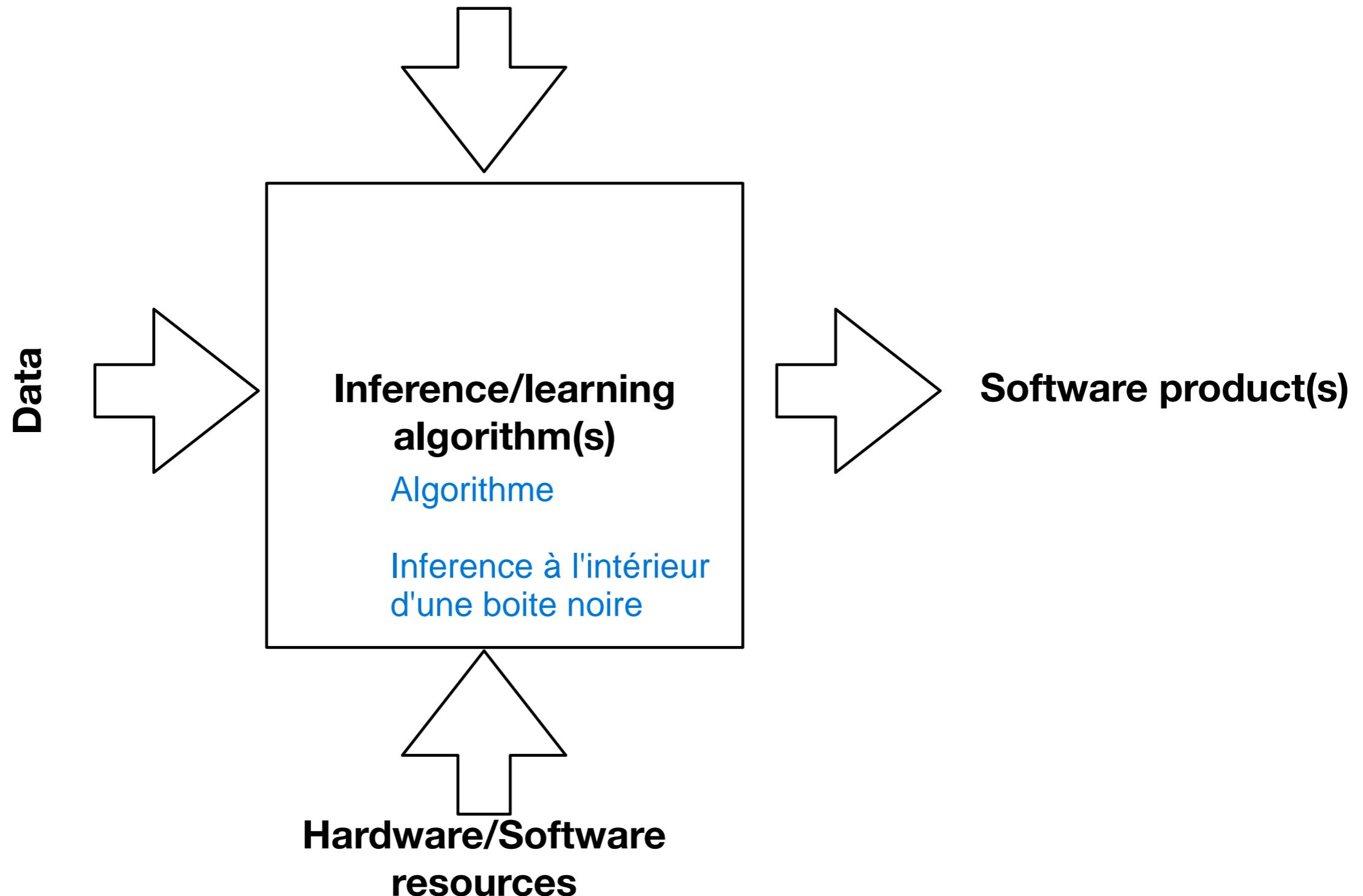
# Méta-modèle de développement log. traditionnel



# Méta-modèle de développement ML

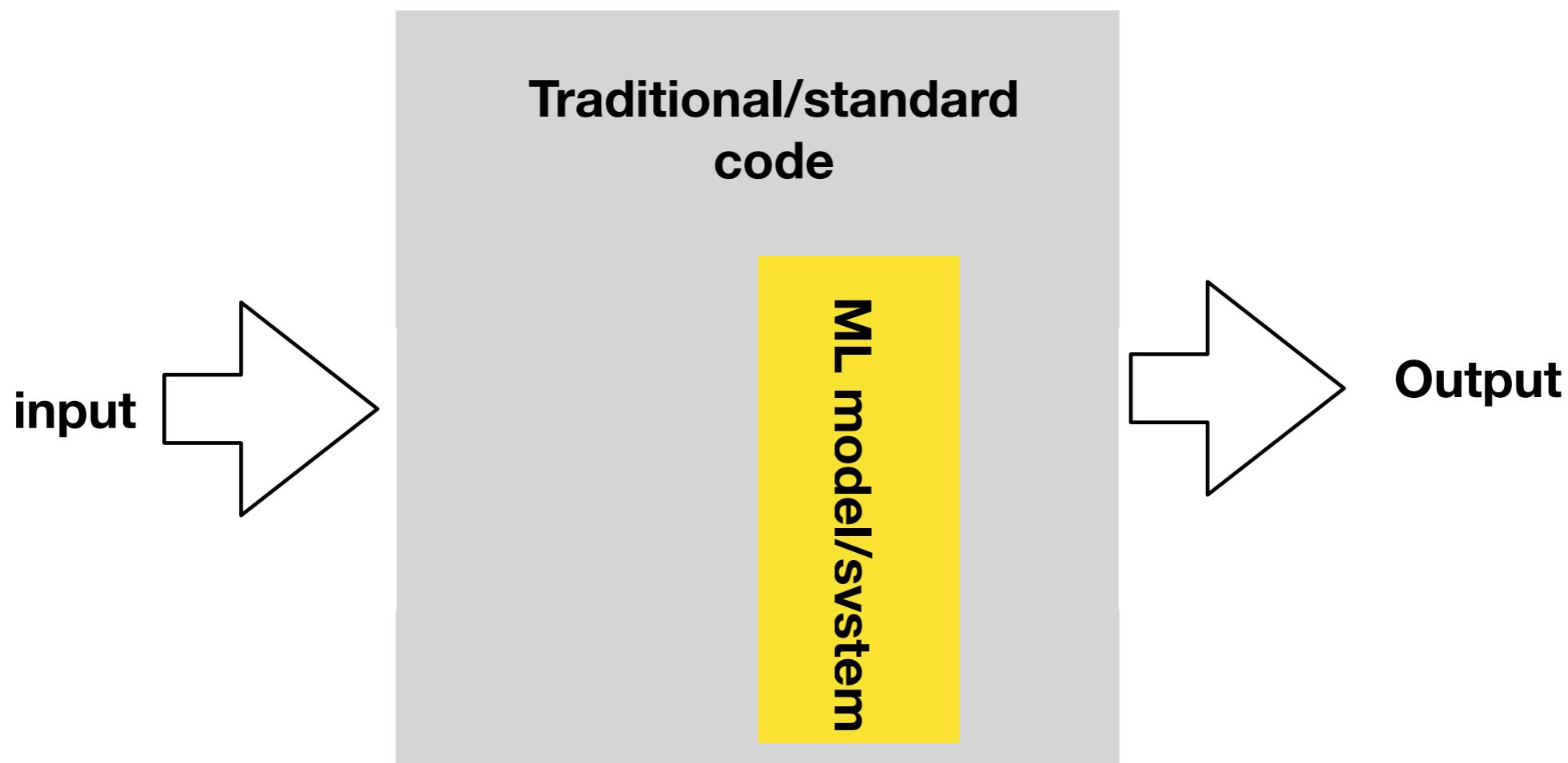
**ML Model**

Machine Learning Modèle d'apprentissage



# Le défi

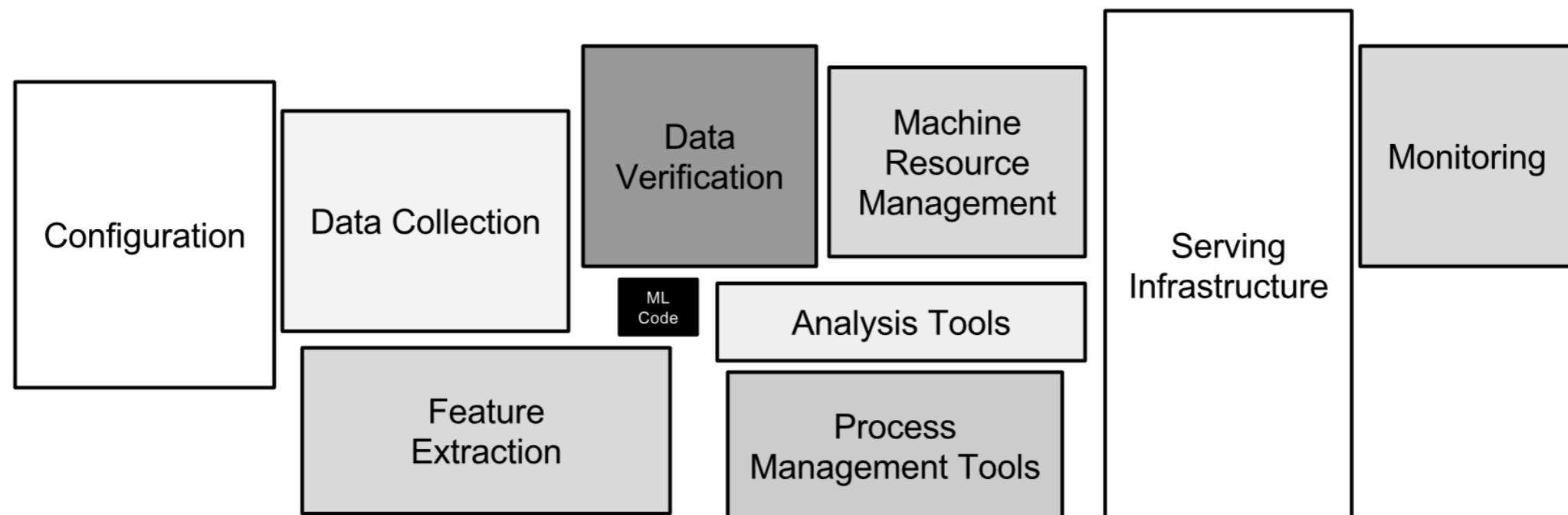
**Assurance qualité pour le système hybride intégrant des composants ML**



**Assurance qualité pour les composants ML seul ?**

# Quelle est la part du ML ?

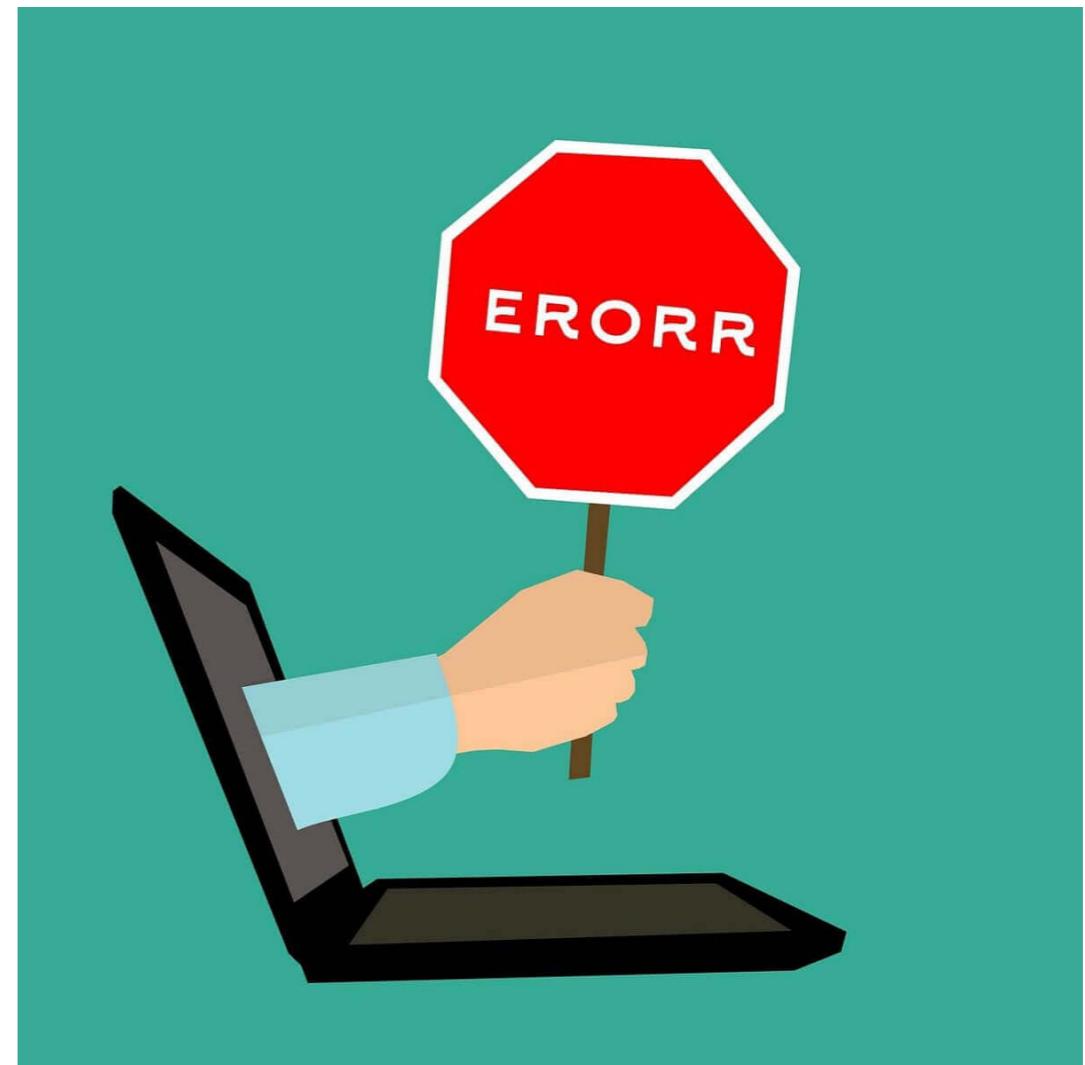
---



**Hidden Technical Debt in Machine Learning Systems (Google)**

# Définitions de base (traditionnel !)

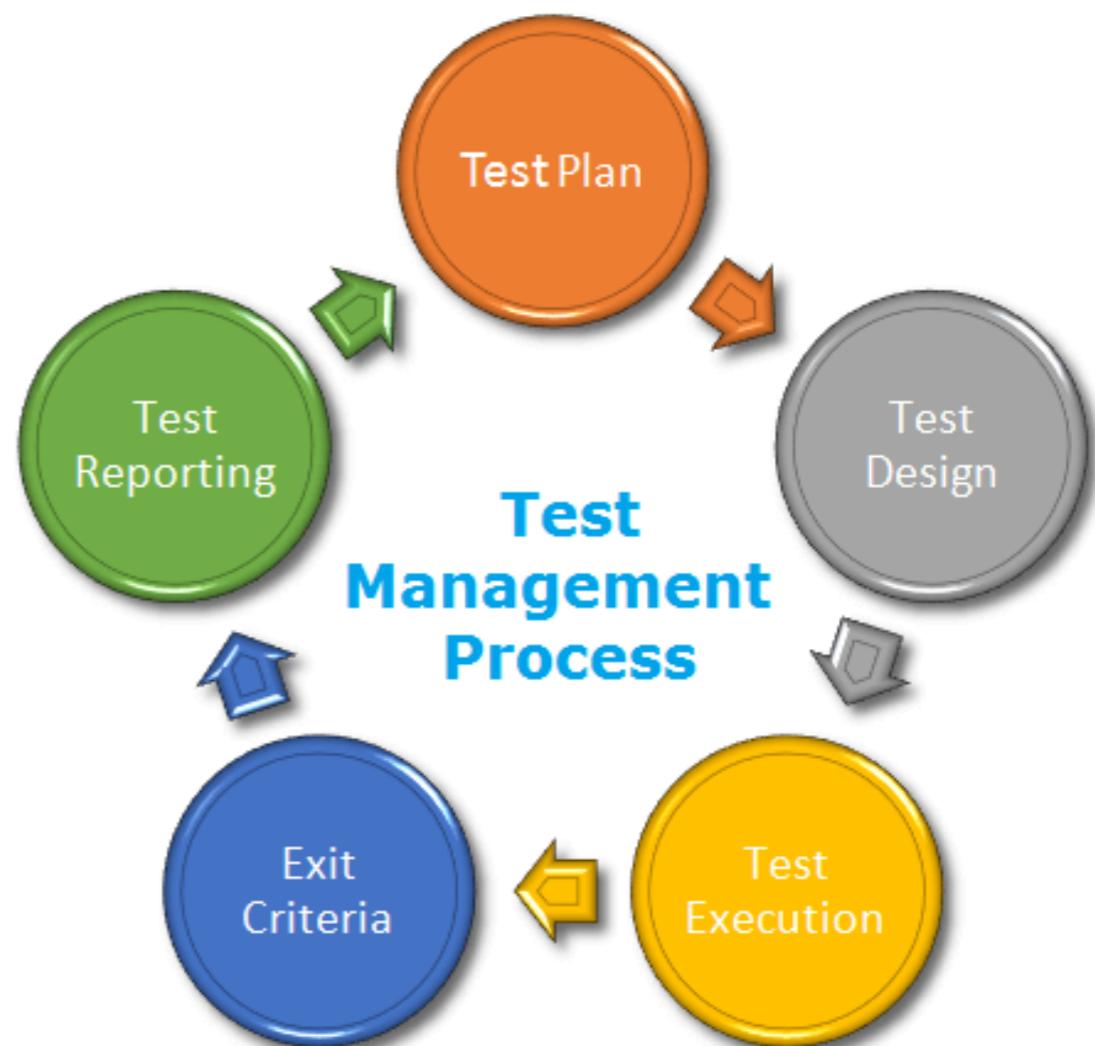
- **Erreurs** : les gens commettent des erreurs
- **Défaut**: Un défaut est le résultat d'une erreur dans la documentation du logiciel, le code, etc.
- **Échec**: Un échec se produit lorsqu'un défaut s'exécute
- **Incident**: Conséquences des échecs - L'occurrence d'un échec peut ou non être apparente pour l'utilisateur
- **Tester** : Exercer le logiciel avec des cas de test pour trouver des défauts ou gagner en confiance dans le système
- **Cas de Test**: Ensemble d'entrées et une liste de sorties attendues (parfois omises)



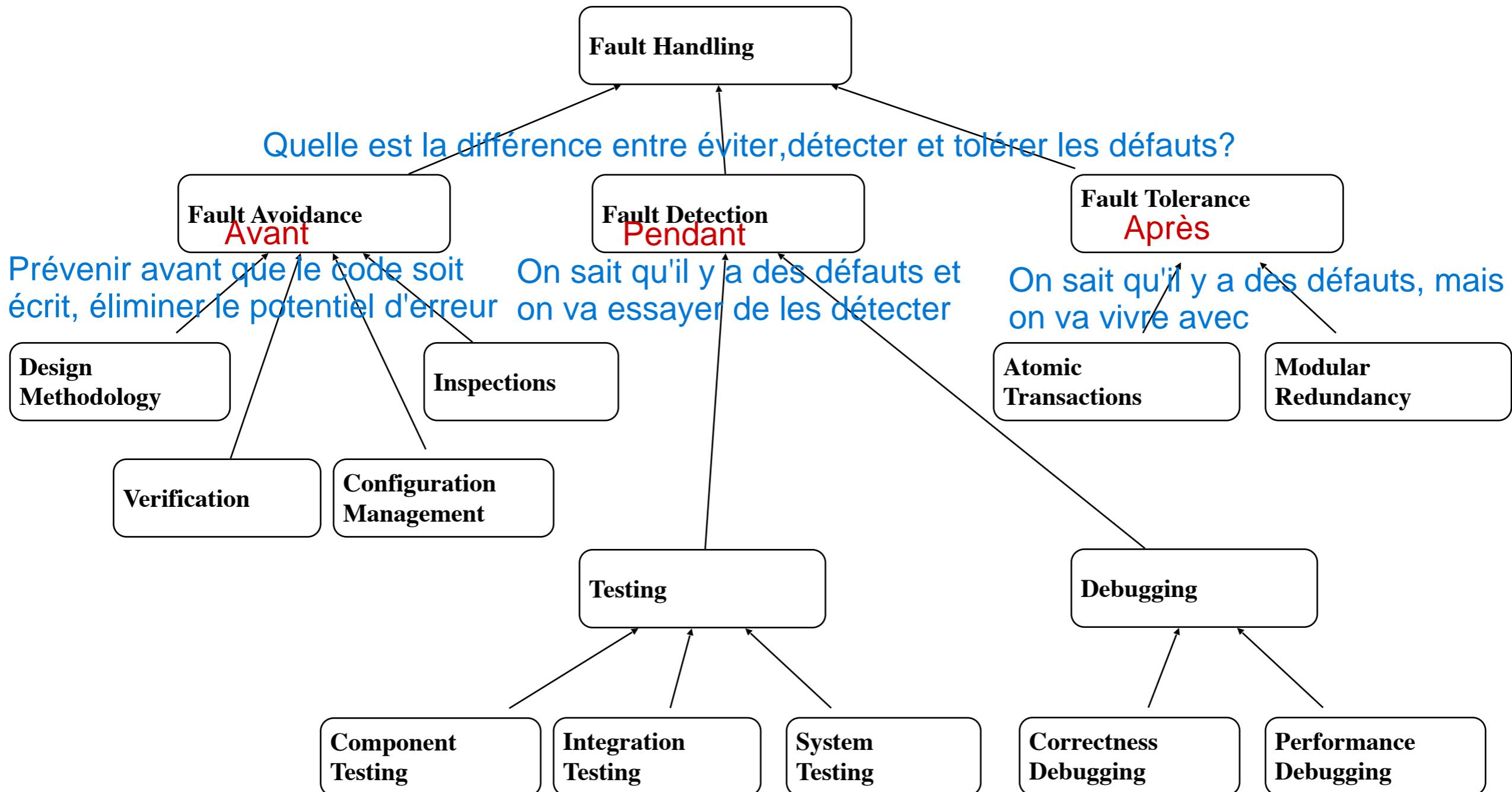
# Organisation des tests traditionnels

- Nombreuses causes potentielles de défaillance différentes,  
Grands systèmes : les tests comportent plusieurs étapes

- Module, component, ou unit testing
- Integration testing
- Function test
- Performance test
- Acceptance test
- Installation test

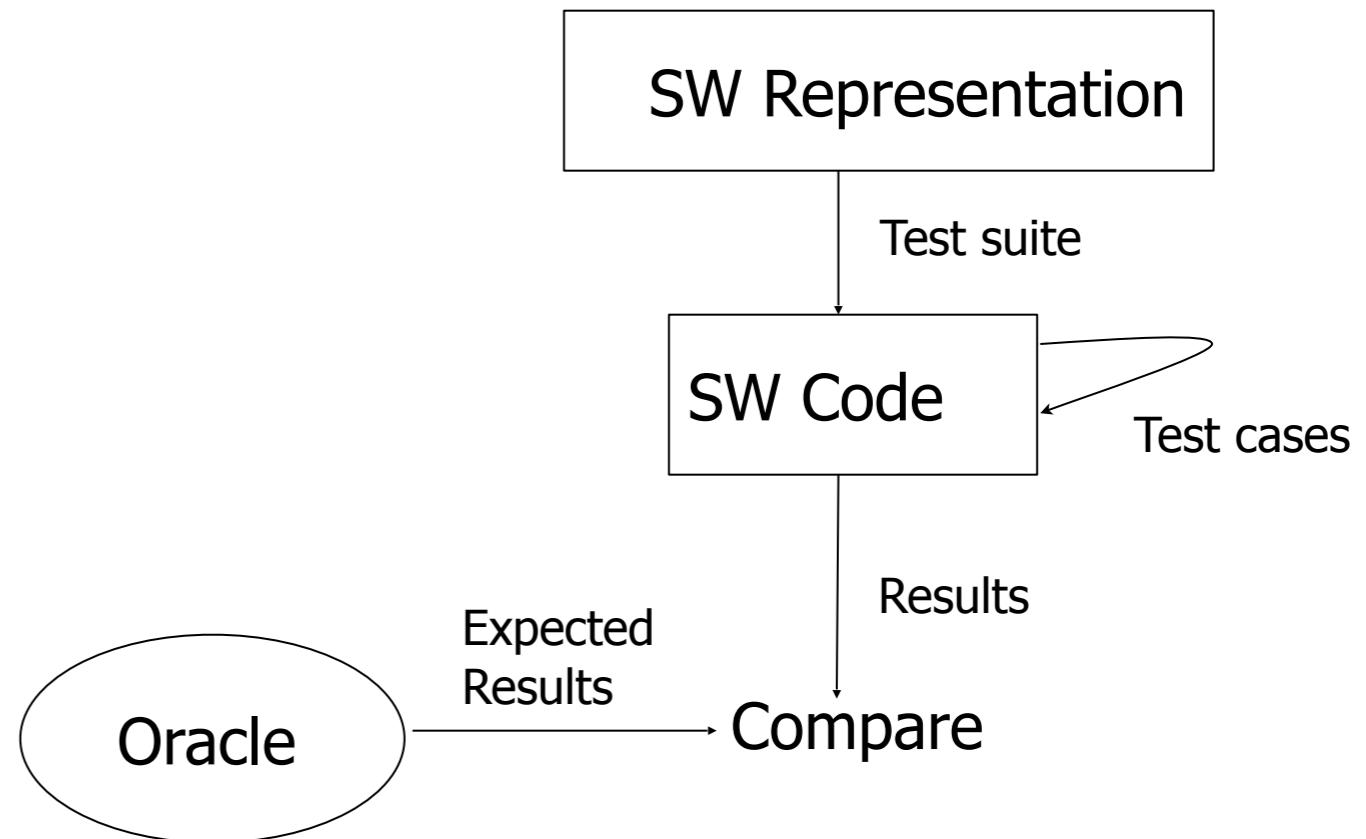


# Traiter les défauts



# Un modèle de test générique

- Nous avons quelques représentations logicielles
  - exigences, code source, cas d'utilisation, conception, architecture
- La ou les représentations logicielles sont utilisées pour dériver des cas de test (suite de tests)
- Nous exécutons le SdT avec des données d'entrée de test
- Le(s) résultat(s) est(sont) comparé(s) à l'Oracle



# Tests exhaustifs

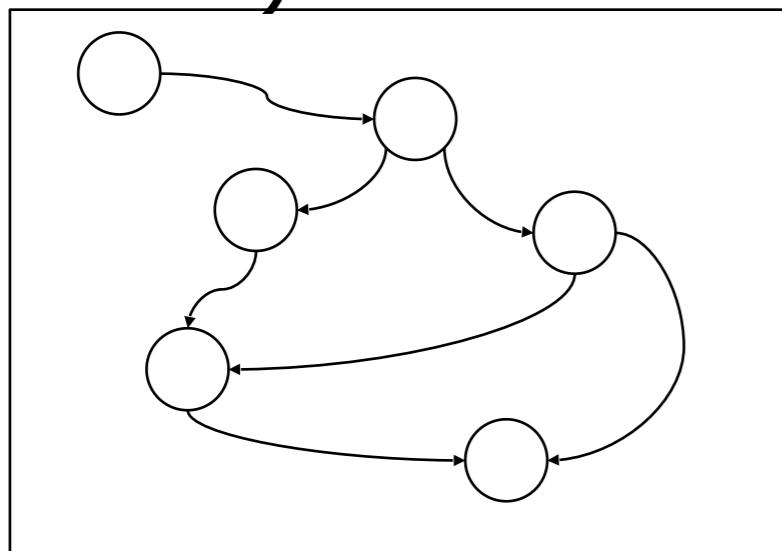
---

Dans les systèmes moyens ou gros, les tests exhaustifs ne fonctionnent pas

- ❑ Les tests exhaustifs, c'est-à-dire tester un système logiciel en utilisant toutes les entrées possibles, sont la plupart du temps impossibles.
- ❑ Examples:
  - programme qui calcule la fonction factorielle ( $n!=n.(n-1).(n-2)\dots 1$ )
    - ✓ Tests exhaustifs = exécuter le programme avec 0, 1, 2, ..., 100, ... comme entrée
  - un compilateur (e.g., javac)
    - ✓ Test exhaustif = exécuter le compilateur (Java) avec n'importe quel programme (Java) possible (c'est-à-dire le code source)
  - Technique utilisée pour réduire le nombre d'entrées
    - Les critères de test regroupent les éléments d'entrée en classes (d'équivalence)
    - Une entrée est sélectionnée dans chaque classe (notion de couverture des données de test)

# Couverture de test

Représentation logicielle  
(modèle)



Critères associés

Les cas de test doivent couvrir  
tous les ... du modèle

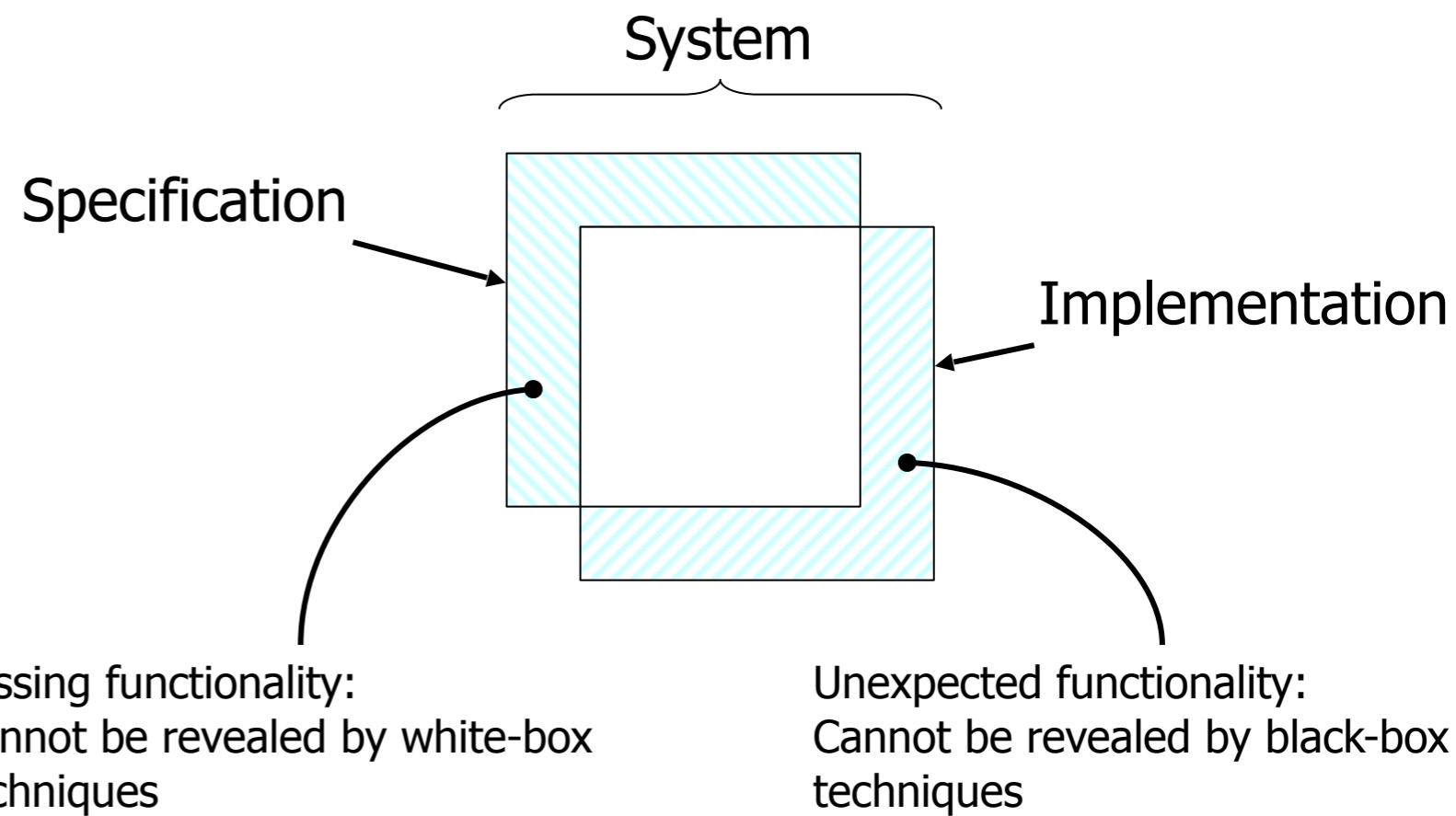


La représentation de

- the specification ==> tests en boîte noire
- the implementation ==> tests en boîte blanche

# Modèles de test courants

- Boîte noire : documents textuels
  - test de partitions de classe ou de catégorie d'équivalence
  - **metamorphic testing**
- Boîte blanche : un modèle, un graphe de flux de contrôle, un CFG inter-procédural annoté
  - déclaration ou couverture en succursale, MCDC, ...



# Oracles

---

*Le résumé de cinq minutes*

# Retour à la base

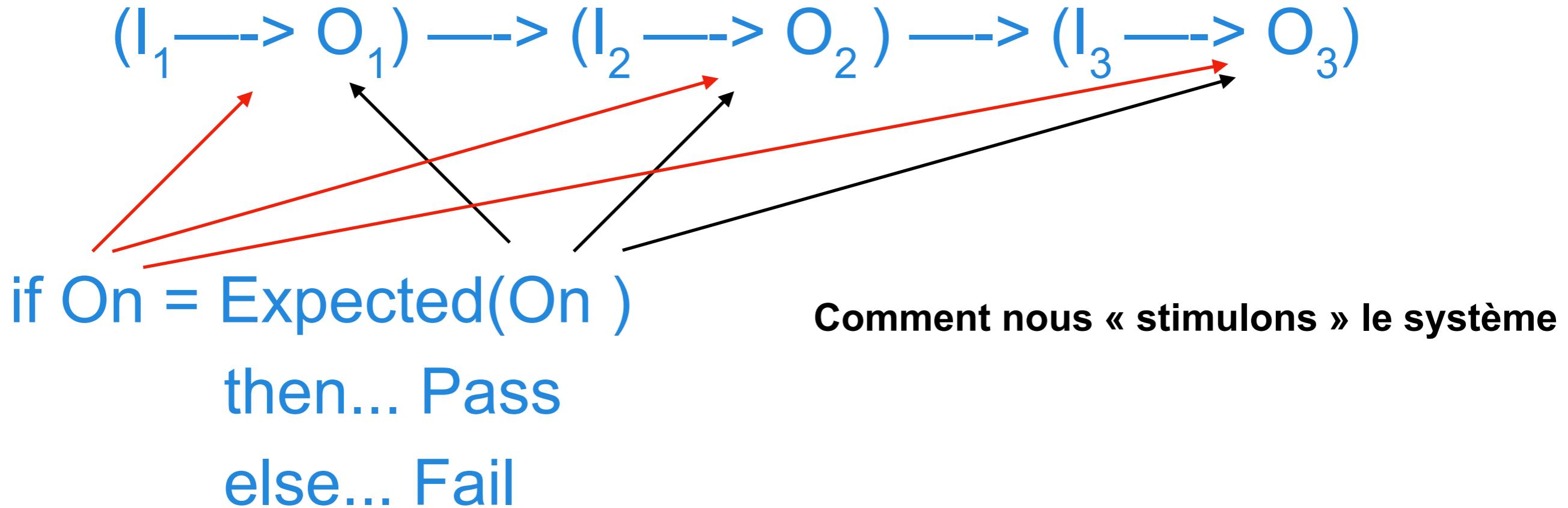
Les tests sont des séquences de **stimuli et d'observations**. Nous nous soucions de l'entrée et de la sortie.



$$(I_1 \longrightarrow O_1) \longrightarrow (I_2 \longrightarrow O_2) \longrightarrow (I_3 \longrightarrow O_3)$$

Input 1 mène à  
Output 1

# De quoi avons-nous besoin pour tester



## Test Oracle

Comment nous vérifions l'exactitude de l'observation résultante.

# Tester: définition d'Oracle

---

Si un test logiciel est une séquence d'activités (stimuli et observations), un **oracle** est un prédicat qui détermine si une séquence donnée est acceptable ou non.

Un oracle répondra par un verdict de réussite ou d'échec sur l'acceptabilité de toute séquence de test pour laquelle il est défini

# Tester les composants

## Oracle

---

- **Information**

- Les informations utilisées par l'oracle pour juger de l'exactitude de l'implémentation, compte tenu des entrées.
- Une spécification, sous une forme qui peut être utilisée directement par le code de test.

- **Procédure d'utilisation d'un Oracle**

- Code qui utilise ces informations pour arriver à un verdict.
- Une forme de vérification automatisée.
- Communément aussi simple que :
- **(value(system output) == value(expected output))**

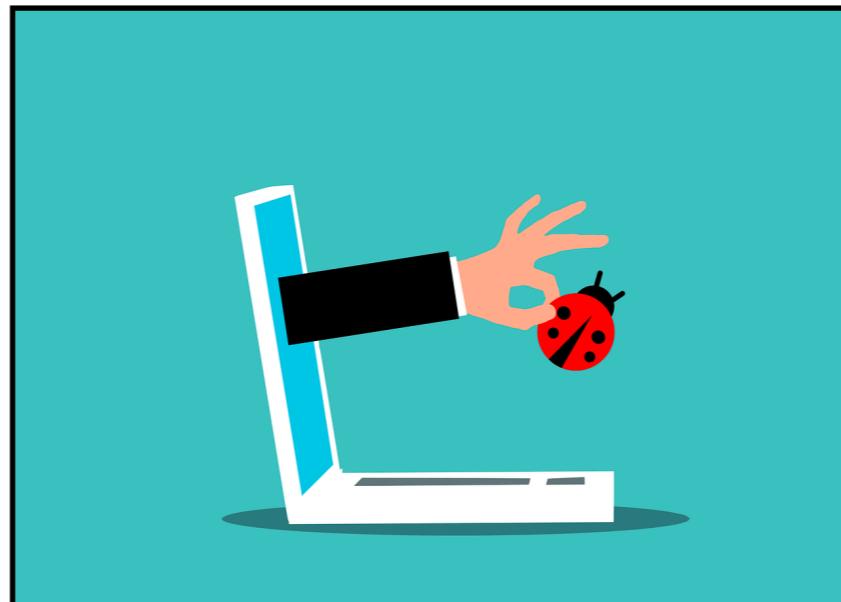
# Les oracles sont du code

---

- Les oracles doivent être développés.
  - Comme le projet, un oracle est construit à partir du cahier des charges.
  - ... et est sujet à interprétation par le développeur
  - ... et peut contenir des défauts
- Un oracle défectueux peut être un problème.
  - Peut entraîner des faux positifs - detection de bogue(s) lorsqu'il n'y avait pas de défaut dans le système.
  - Peut entraîner des faux négatifs - pas de detection de bogue(s) alors qu'il y avait un/des défaut(s) dans le système.

# Faux positif? Faux négatif?

Lorsque nous testons (nous cherchons pour un/des bogue(s)),  
nous avons deux possibilités



Nos tests on “trouvé” un/des bogue(s),  
(Test positif)

ex: On a la Covid,  
si on est + mais  
c'est pas une  
bonne chose

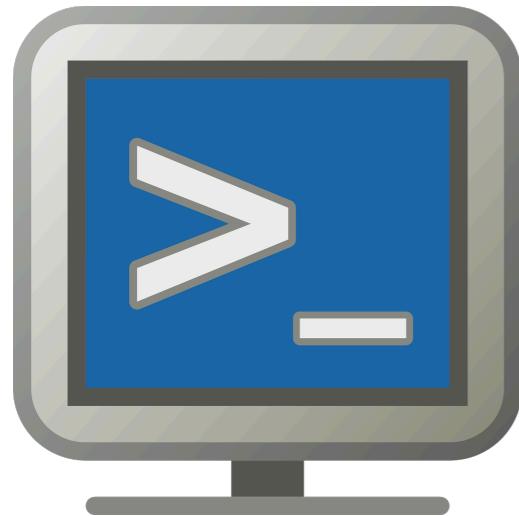


Nos tests n'ont trouvé aucun bogue(s)  
(Test négatif)

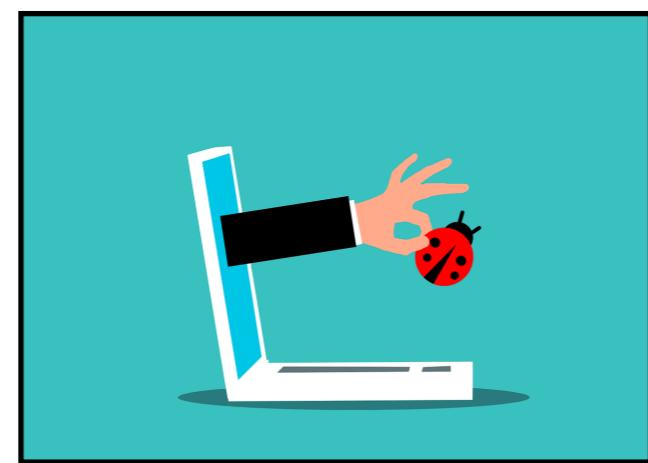


# Faux positif? Faux négatif?

1000 méthodes



Tests



5 bogues?



$f(x)$

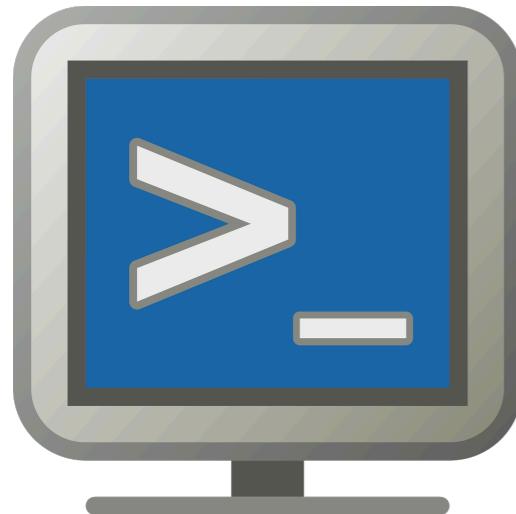
Ici, nous avons “détecté” 5 bogues,  
mais un des bogues “détecté”, n’était pas un bogue du tout.

Nous avons donc 1 faux positif dans notre détection

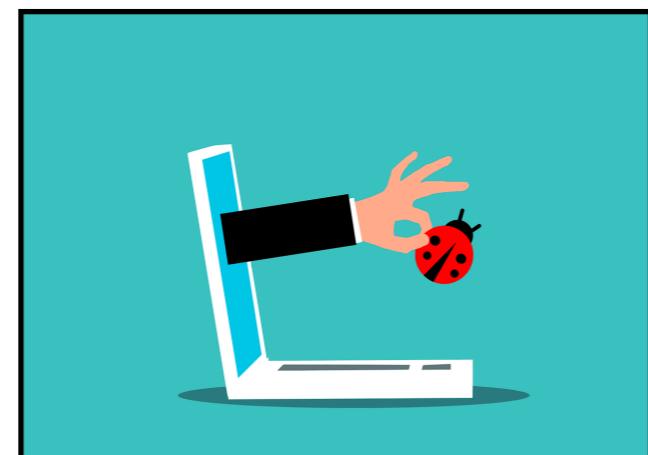
# Faux positif? Faux négatif?

Ca se peut dans un de ces bogues, il y a un qui est Flaky (des fois ça me dit qu'il y en a et des fois ça me dit qu'il y en a pas)  
On pourrait repasser mes tests qq fois pr voir si un de mes tests est Flaky

1000 méthodes (4 défauts)



Tests



3 bogues?



Si nous savons que 4/1000 méthodes sont défectueuses,  
nous devrions “détecté” 4 bogues.



Si nous ne détectons que 3 bogues,  
une des méthodes que nous avons déclarées comme étant correcte ne l'est pas.  
Nous avons donc 1 faux négatif dans notre détection 1 FN

# Où obtenons-nous des oracles de test

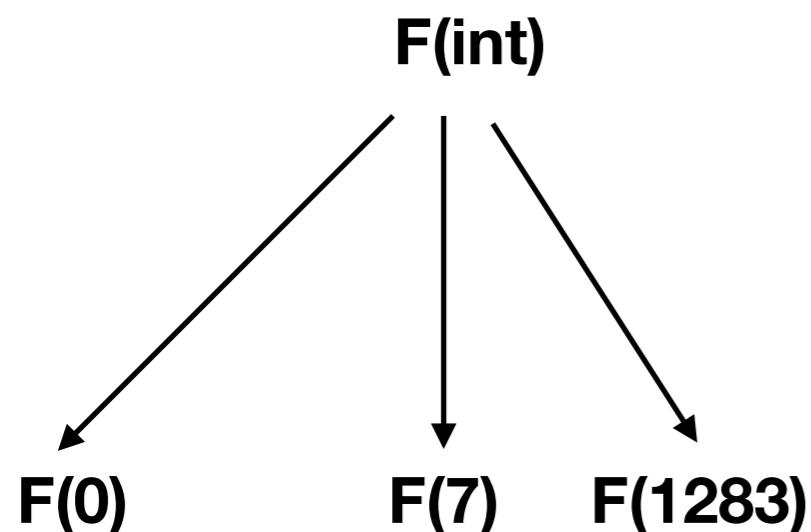
**Le plus souvent :** les développeurs écrivent des oracles à la main, liés à un cas de test particulier.

- Beaucoup d'efforts manuels et de temps requis pour créer des tests
- Ne sera pas en mesure d'exécuter de nombreux tests.  
Avez-vous choisi les bonnes entrées ?



# Le problème d'Oracle de test

Nous sommes doués pour proposer de nouvelles entrées...



Mais, il est **beaucoup plus difficile** de vérifier automatiquement les résultats de plusieurs tests.

$$F(0) = ?$$

$$F(7) = ?$$

$$F(1283) = ?$$

# Types d'oracles

---

## Oracles spécifiés

- Les développeurs, à l'aide des exigences, spécifient formellement les propriétés que le comportement correct doit suivre.

## Oracles dérivés

- Un oracle est dérivé d'artefacts de développement ou d'exécutions système.

## Oracles implicites

- Un oracle juge l'exactitude en utilisant les propriétés attendues de nombreux programmes.

## Oracles humains

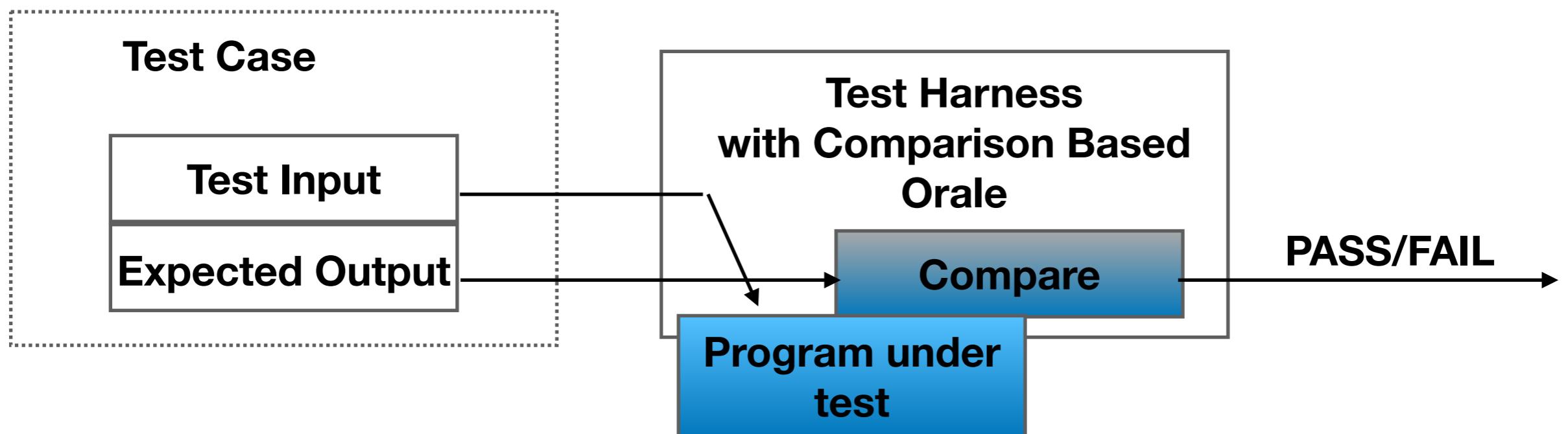
La qualité des données n'est jamais garantie

- Comment gérer l'absence d'oracle ?

Ex: Captcha, où on doit rentrer manuellement les images correspondant à un feu de circulation, se base sur ce que la majorité des gens ont rentré et va nous laisser passer si on coche une image par erreur

# Oracles spécifiés

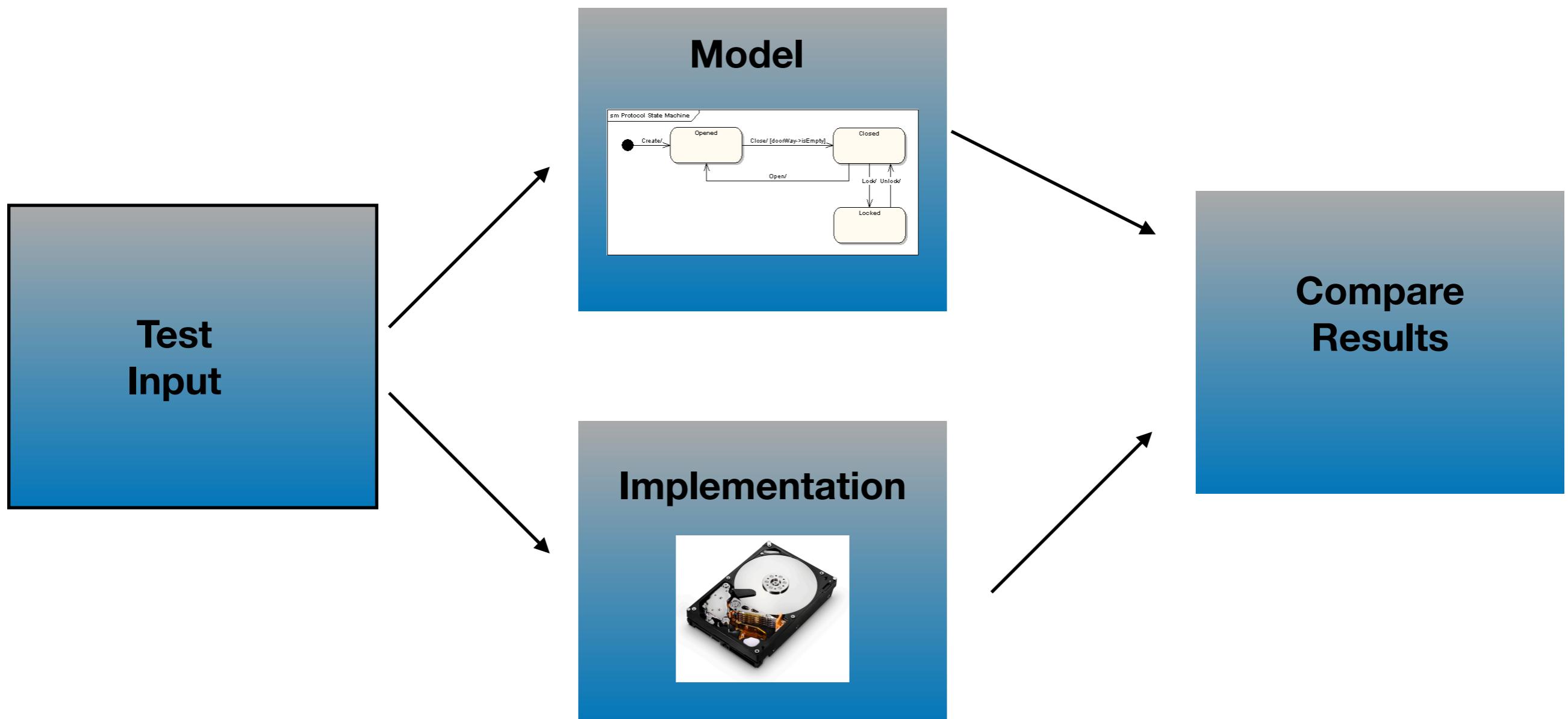
Les oracles spécifiés jugent le comportement à l'aide d'une spécification d'exactitude créée par l'homme



La plupart des oracles sont des oracles spécifiés : tout cas de test écrit manuellement a un oracle spécifié.

# Modèles de système comme oracles

Les modèles pourraient potentiellement servir d'oracle de test "universel"





## “On Testing Untestable Programs”

---

*Elaine Weyuker in The Computer Journal N 4 Viol 25, 1982*

# Une taxonomie de programmes non testables

- Programme écrit pour déterminer la réponse
- Programme qui produit tellement de résultats qu'il est impossible de tout vérifier
- Programmes pour lesquels le testeur a une idée fausse

Programmes de simulations, donc  
pour déterminer la réponse des  
systèmes il faut faire des  
simulations



# How to

---

- Si nous écrivons un programme pour calculer une réponse, cela implique que nous n'avons pas une telle réponse
- Si nous ne savons pas quelle est la réponse, comment pouvons-nous écrire un oracle et tester le programme ?



# Pseudo-Oracles

---

- Si nous ne pouvons pas connaître la « bonne » réponse, nous ne pouvons pas tester le logiciel
- Au lieu d'un oracle, utilisez un pseudo-oracle
- obtenez deux systèmes développés à partir des mêmes spécifications et comparez les résultats
- Spécification ou données du système ML ?
  - use “*simple inputs*”

# Exemples

---

- Programmes de simulation
- Compilateurs
- Optimisations combinatoires
- PNL (NLP en anglais)

Z. Q. Zhou, D. H. Huang, T. H. Tse, Z. Yang, H. Huang, and T.Y. Chen. Metamorphic testing and its applications. In Proc. of the 8th International Symposium on Future Software Technology (ISFST 2004), 2004.

# Proto-AI code excerpt

---

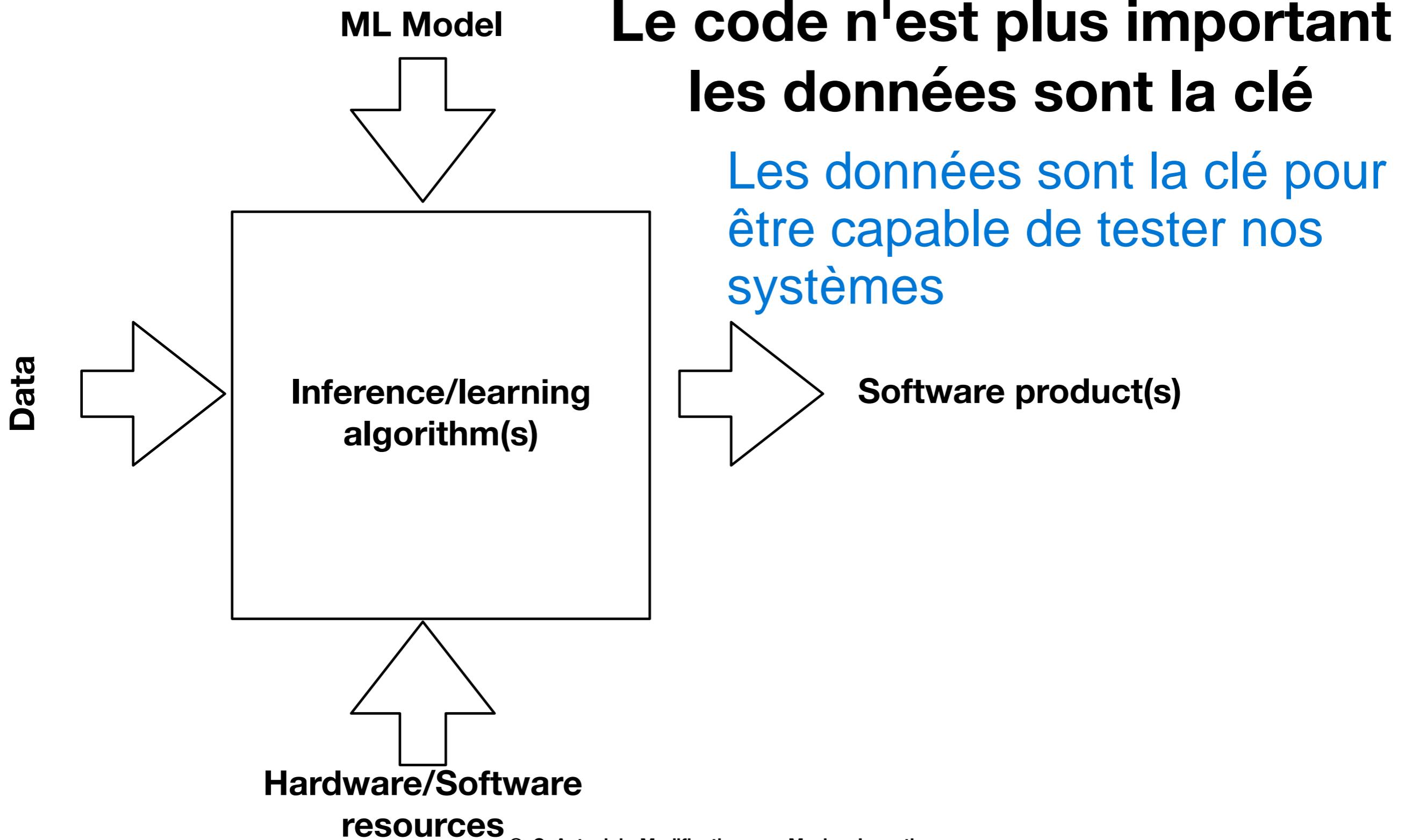
```
public static double YYY(double[] s, double[] t){
 double[][] matr = new double[s.length + 1][t.length + 1];

 matr[0][0] = 0;
 for (int i = 1; i < s.length + 1; i++) {
 matr[i][0] = inf;
 }
 for (int i = 1; i < t.length + 1; i++) {
 matr[0][i] = inf;
 }

 for (int i = 1; i < s.length + 1; i++) {
 for (int j = 1; j < t.length + 1; j++) {
 double cost = distanceD(s[i - 1], t[j - 1]);
 matr[i][j] = cost + minimum(matr[i - 1][j], matr[i][j - 1], matr[i - 1][j - 1]);
 }
 }
 return matr[s.length][t.length];
}
```

Hermann Ney. 1990. The use of a one-stage dynamic programming algorithm for connected word recognition. In *Readings in speech recognition*, Alex Waibel and Kai-Fu Lee (Eds.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA 188-196.

# Complication

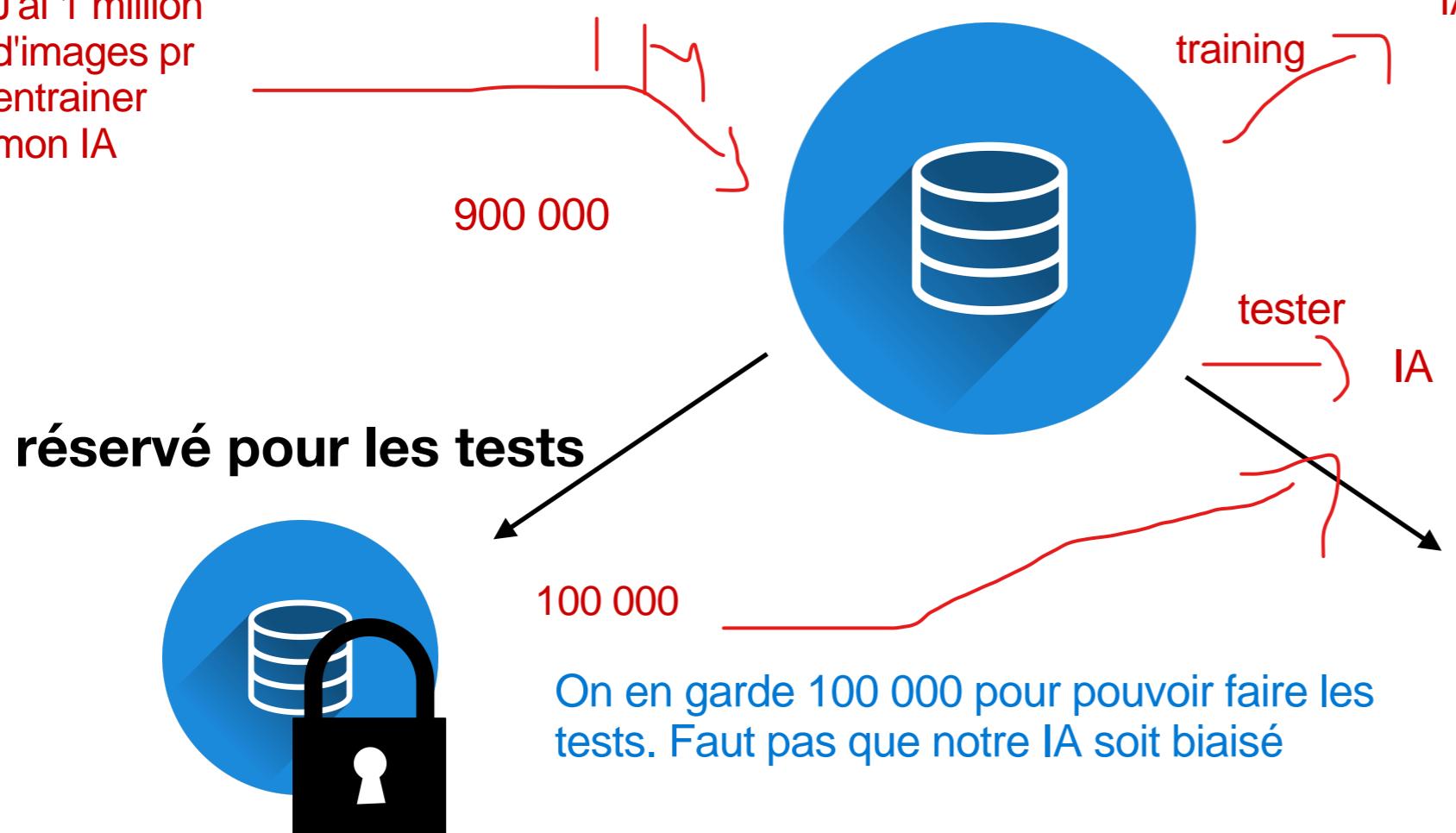


# Comment testons-nous les modèles de ML?

Si nous avons beaucoup de données :

- Nous n'utilisons pas toutes nos données pour entraîner notre modèle. Nous réservons certaines de nos données à des fins de test.

J'ai 1 million d'images pr entraîner mon IA

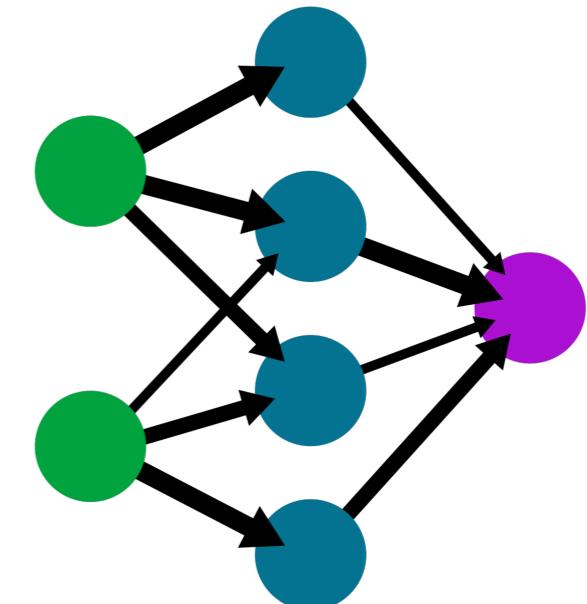


On en garde 100 000 pour pouvoir faire les tests. Faut pas que notre IA soit biaisé

Problème:  
tester le système, je dois avoir des images  
Je ne peux pas réutiliser les 1 M d'images fournies pr entraîner mon IA pour tester mon système.

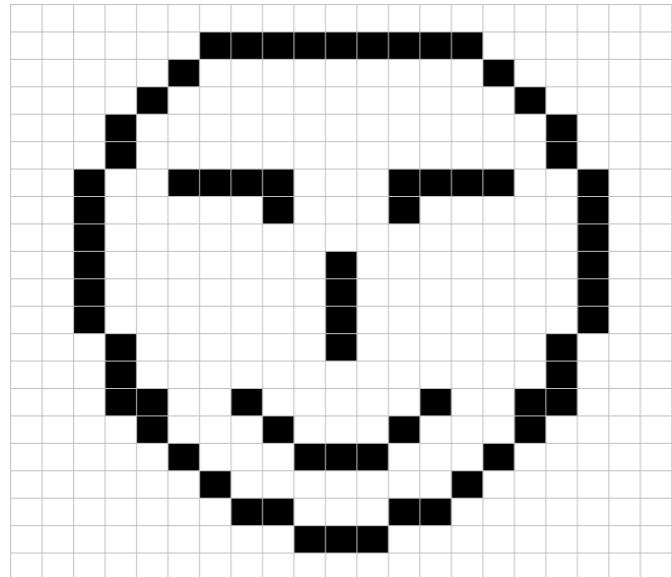
Il faut tester avec des images inconnues, car notre système devrait servir à reconnaître des images qu'il n'a jamais vu.

A simple neural network  
input layer      hidden layer      output layer

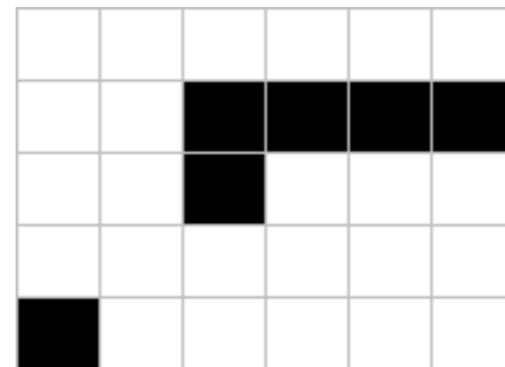


# Faux positif? Faux négatif?

---

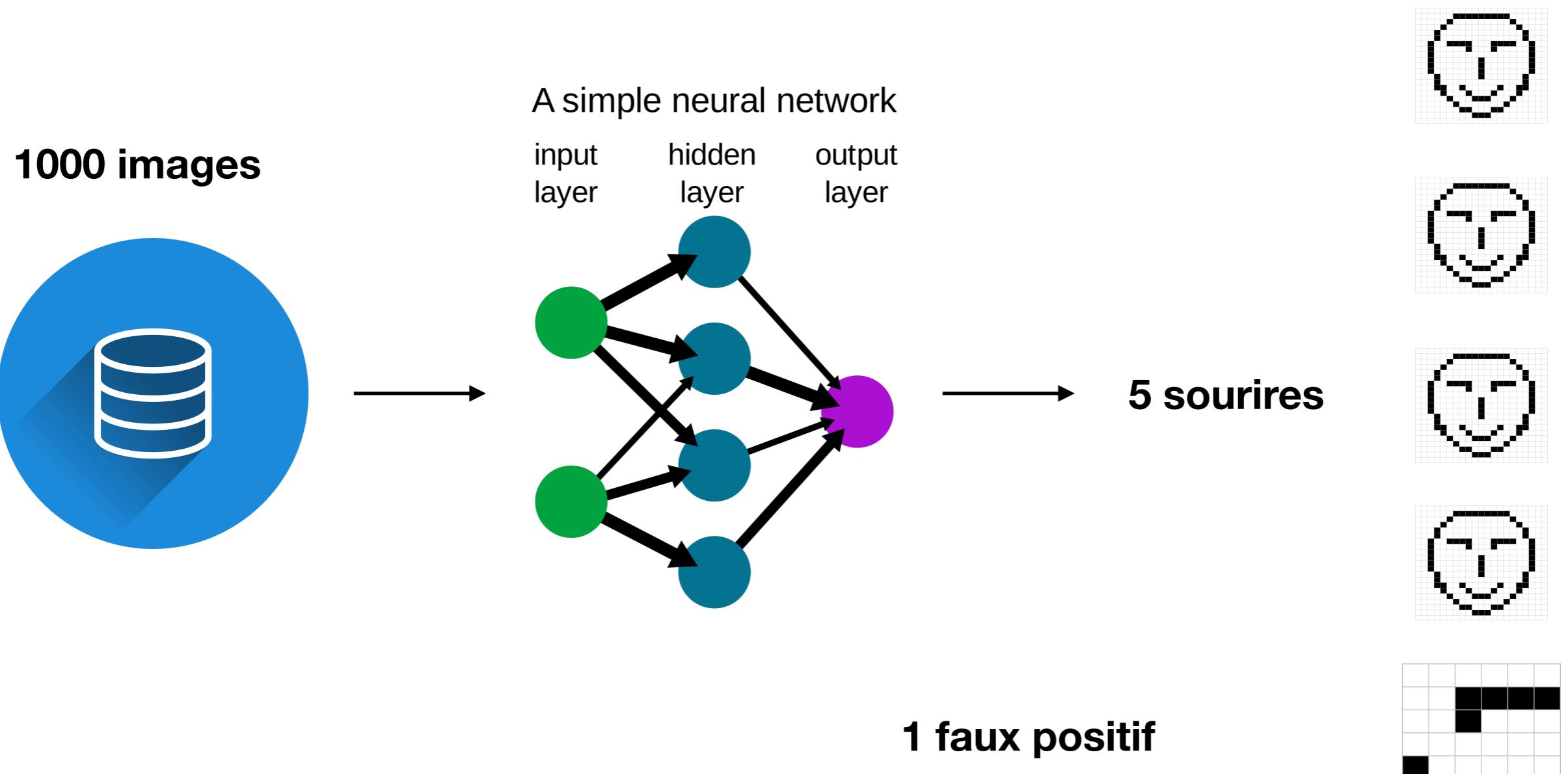


**Visage avec un sourire  
(Positif)**

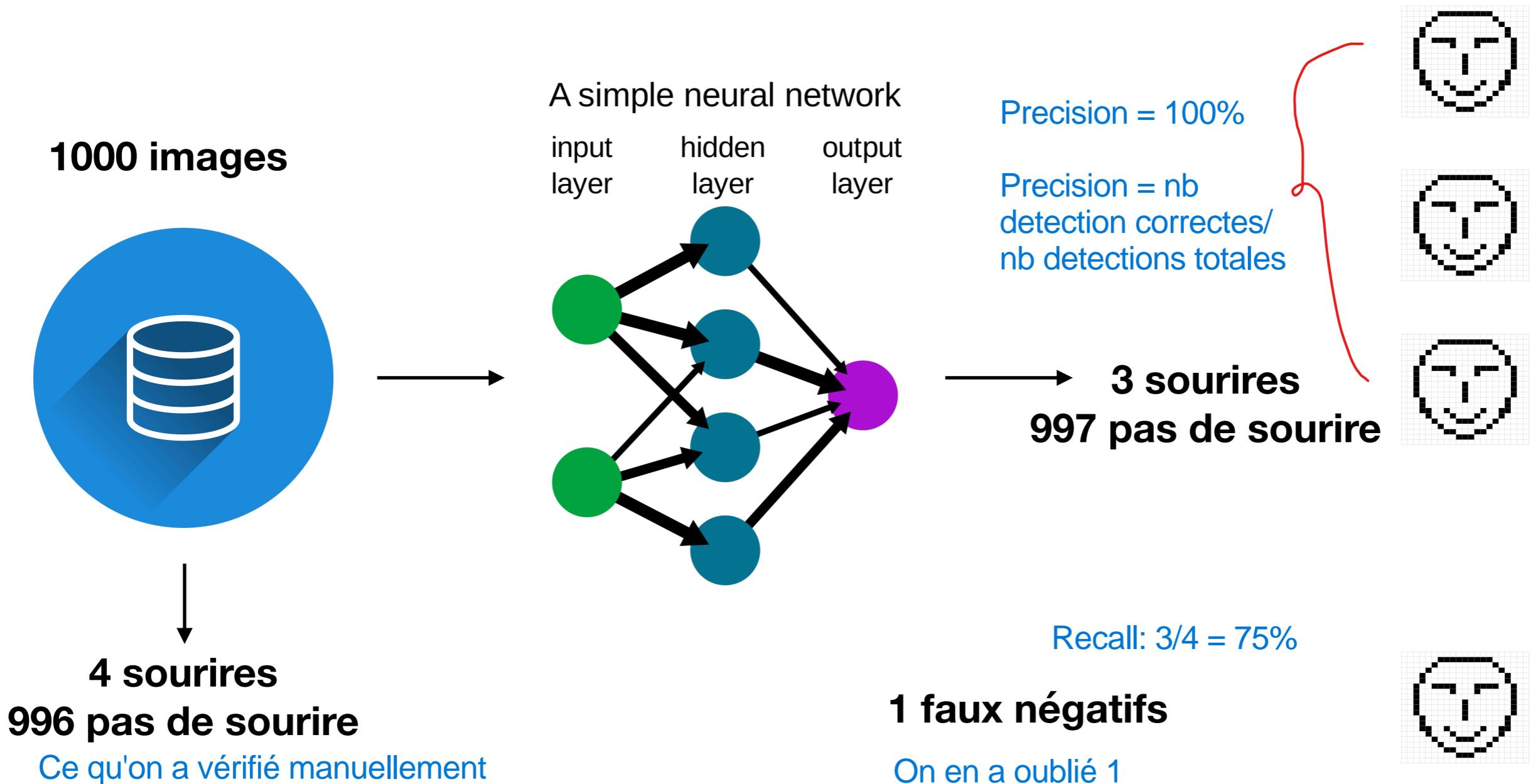


**Pas un sourire  
(Négatif)**

# Faux positif? Faux négatif?



# Faux positif? Faux négatif?



# Problème?

---

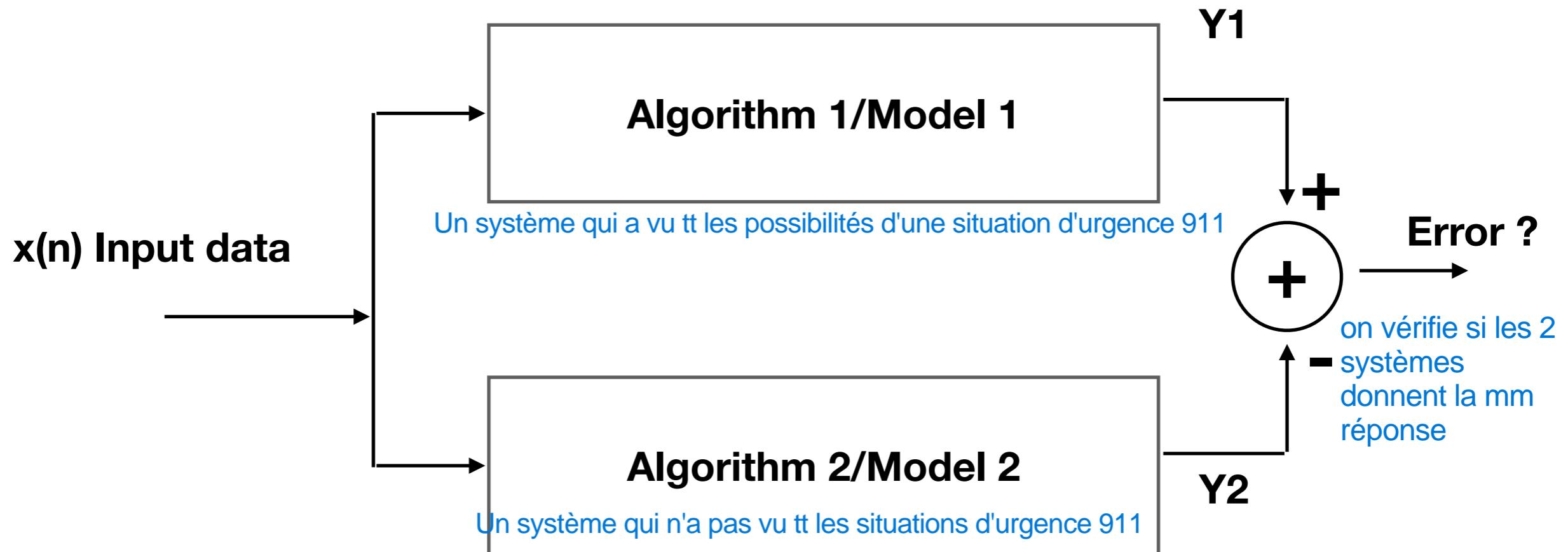
Nos tests sont aussi précis que nos données

Si nous ne disposons pas de données qui représentent toutes les situations dans lesquelles le modèle ML sera utilisé, nous ne pouvons pas tester le modèle complètement.

# Pseudo-oracle how-to

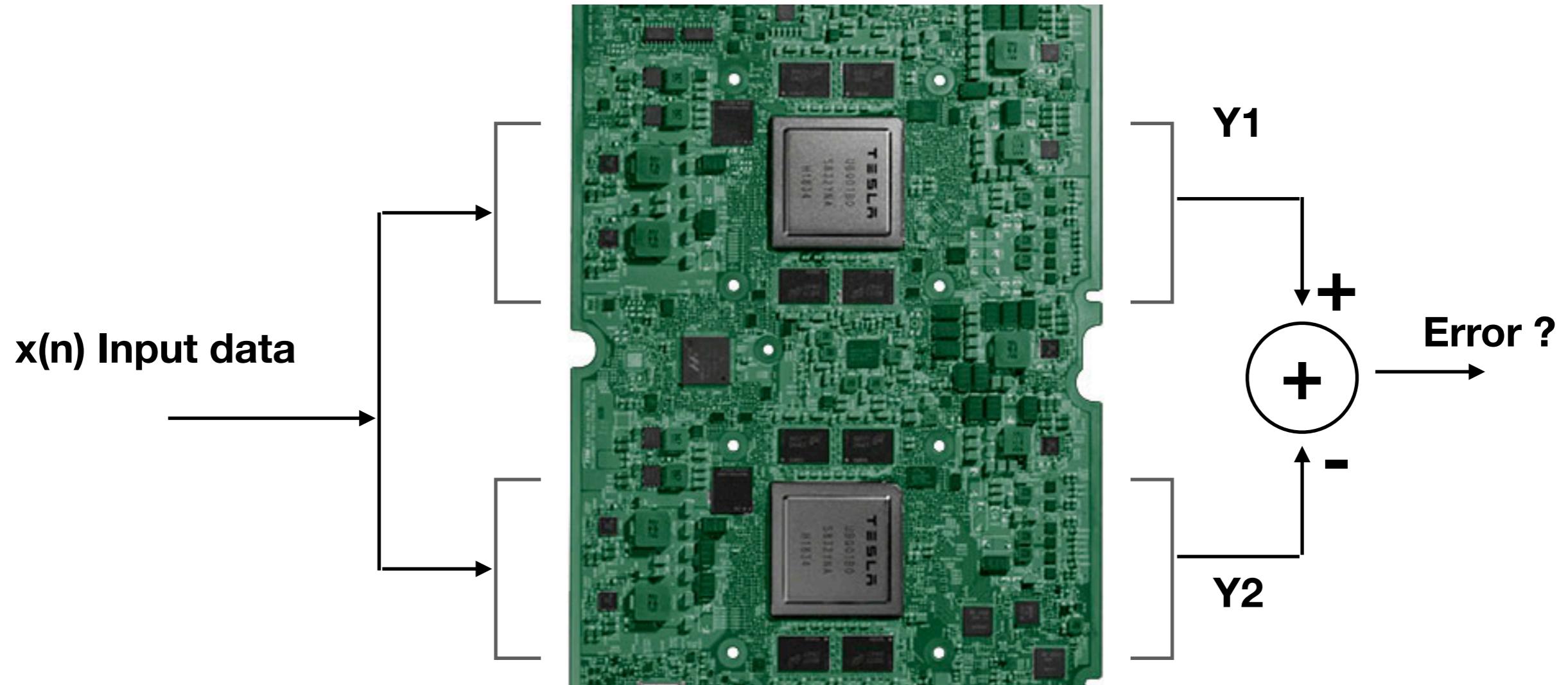
Deux systèmes identiques entraînés avec des données un peu différentes.

Leurs résultats sont comparés pour voir si le système 1 va donner des résultats différents du système 2



**Si  $Y_1$  et  $Y_2$  sont différents, nous savons que l'un est faux**

# Pseudo-oracle how-to



**Si  $Y_1$  et  $Y_2$  sont différents, nous savons que l'un est faux**

# Chances de se tromper?

---

**Soit la Probabilité qu'un système soit erroné => X**

$$x = 0.10$$

**Probabilité que deux systèmes soit erronés  
en même temps =>  $X^2$**

$$\begin{aligned} x_1 &= 0.10 \\ x_2 &= 0.10 \end{aligned} \quad = 0.10 * 0.10 = 0.01$$

**Si nous avons Y nombres de systèmes concurrent,  
la probabilité d'une erreur simultané est =>  $X^Y$**

Si l'erreur est la mm dans tous les cas, on a juste à mettre la proba en exposant du nb de cas

# Observation

---

- De nombreux programmes sans oracles ont des propriétés telles que certaines modifications apportées à l'entrée entraînent des modifications prévisibles de la sortie  
Si je donne à mon système une image et que je tourne cette image, est-ce que le système va me donner la mm réponse?  
Que j'aille un être humain dans les 2 images  

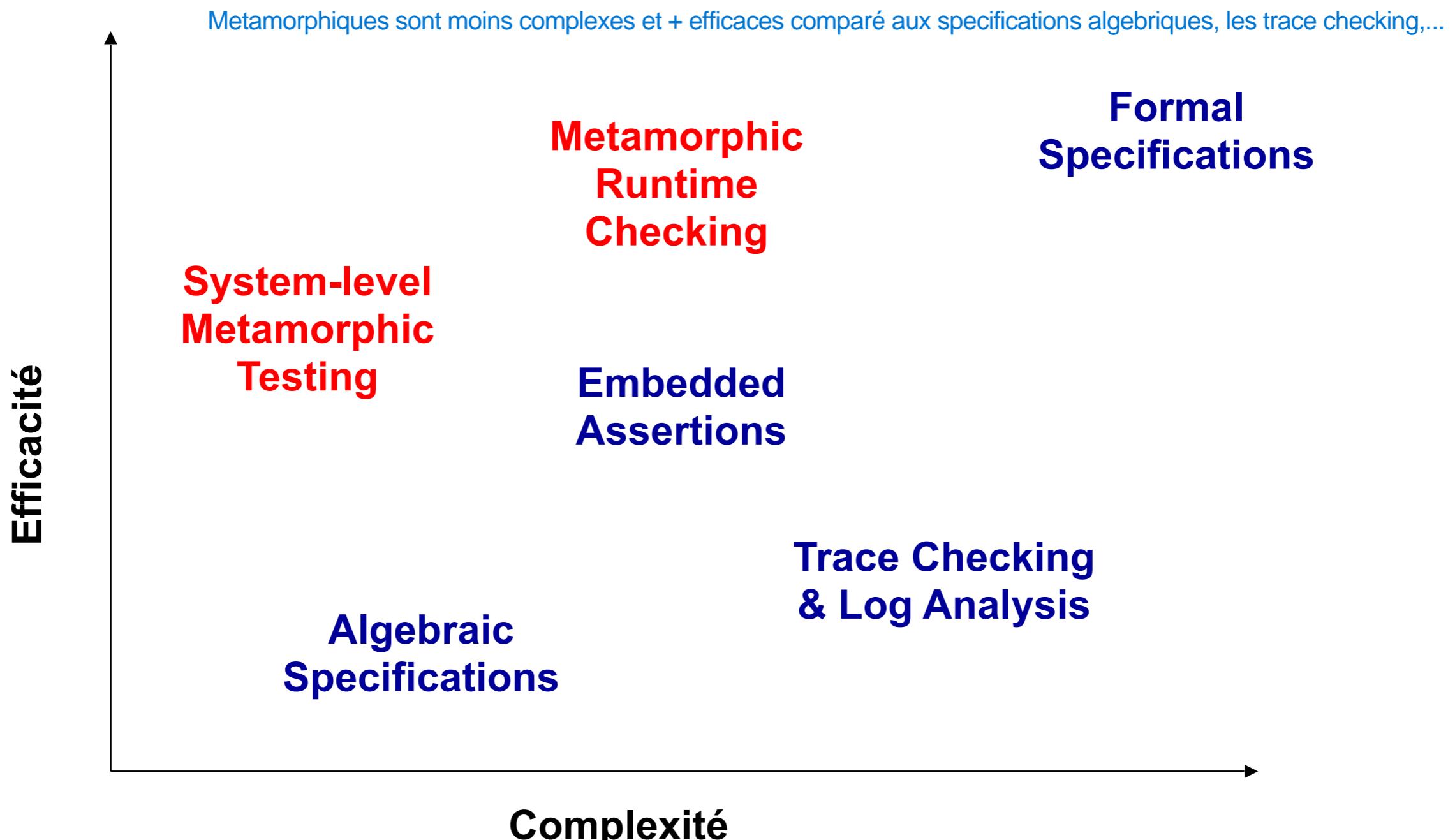
- Nous pouvons détecter des défauts dans ces programmes en recherchant toute violation de ces "propriétés métamorphiques"
- C'est ce qu'on appelle les "**tests métamorphiques**"
  - [T.Y. Chen et al., *Info. & Soft. Tech* vol.4, 2002]

# Hypothèses (C. Murphy )

---

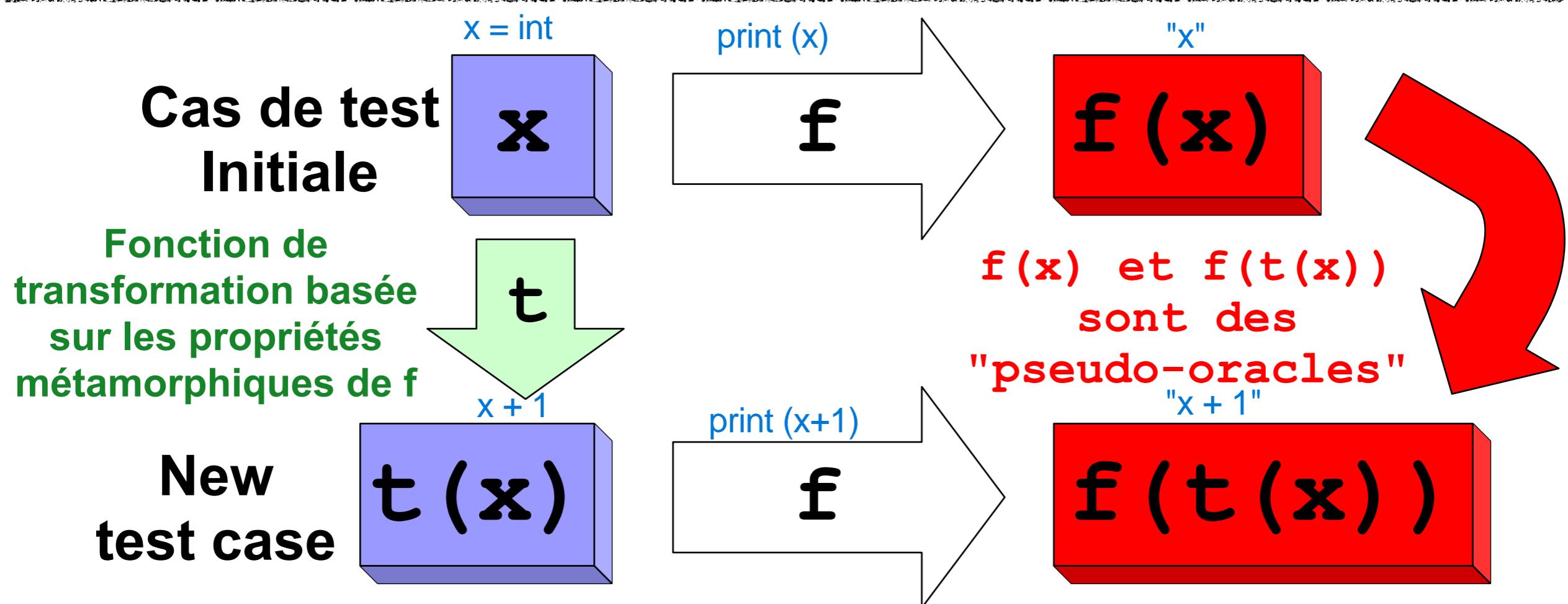
- Pour les programmes qui n'ont pas d'oracle de test, une approche automatisée des **tests métamorphiques est plus efficace** pour détecter les défauts que les autres approches
- Une approche qui **effectue des tests métamorphiques au niveau des fonctions dans le contexte d'une application en cours d'exécution** augmentera encore l'efficacité
- Il est **possible de poursuivre ce type de test dans l'environnement de déploiement**, avec un impact minimal sur l'utilisateur final

# Complexité vs Efficacité - C. Murphy



# Test métamorphique [Chen et al., 2002]

On va tester l'input  $x$  dans la fonction  $f$



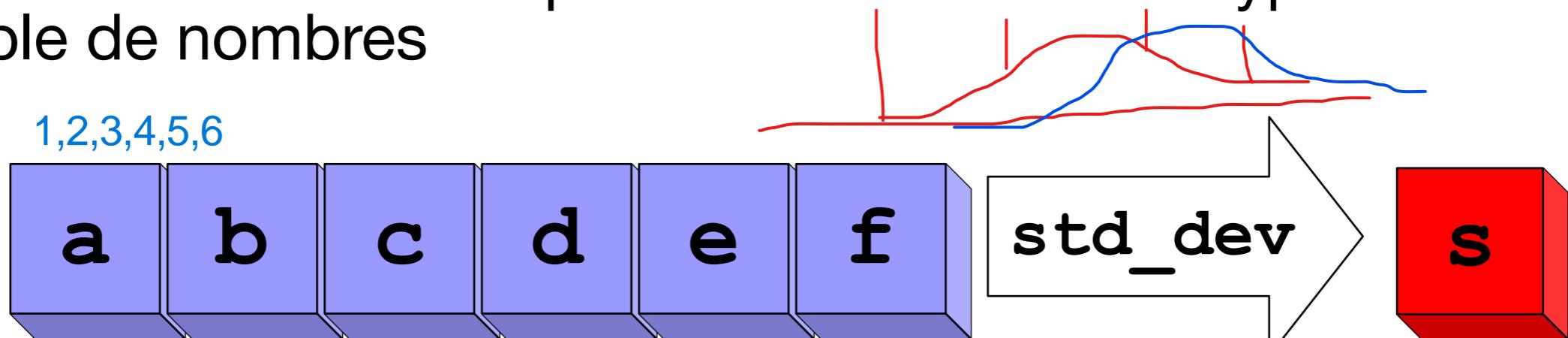
- Si la nouvelle sortie de cas de test  $f(t(x))$  est comme prévu, elle n'est pas nécessairement correcte
- Cependant, si  $f(t(x))$  n'est pas comme prévu,  $f(x)$  ou  $f(t(x))$  – ou les deux ! - est faux

# Exemple de test métamorphique

Si j'ai une liste (ex: a,b,c,d,e,f), ma propriété métamorphique est d'ajouter des constantes aux items de ma liste, je dois l'ajouter à tous les items de ma liste

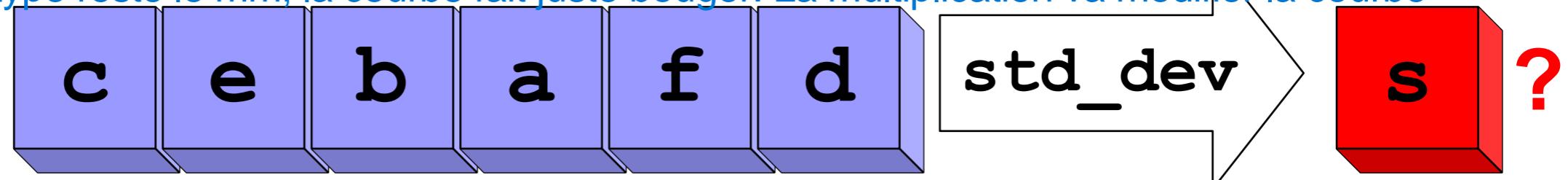
- Considérons une fonction pour déterminer l'écart type d'un ensemble de nombres

Initial input

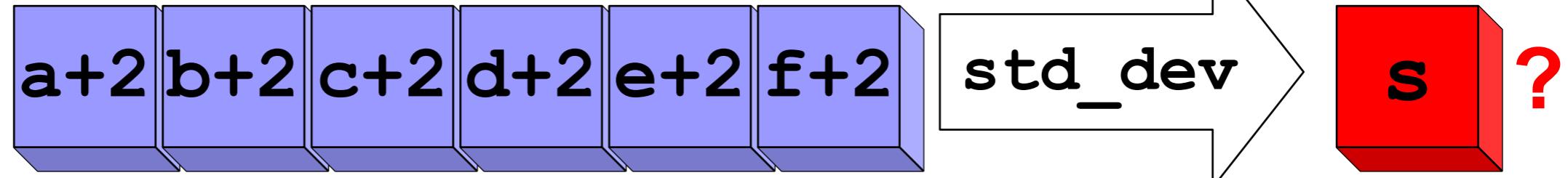


Dans le cas de l'écart-type, je px avoir une liste dans n'importe quel ordre et ca ne va pas affecter le résultat de l'écart-type. L'écart-type reste le mm, la courbe fait juste bouger. La multiplication va modifier la courbe

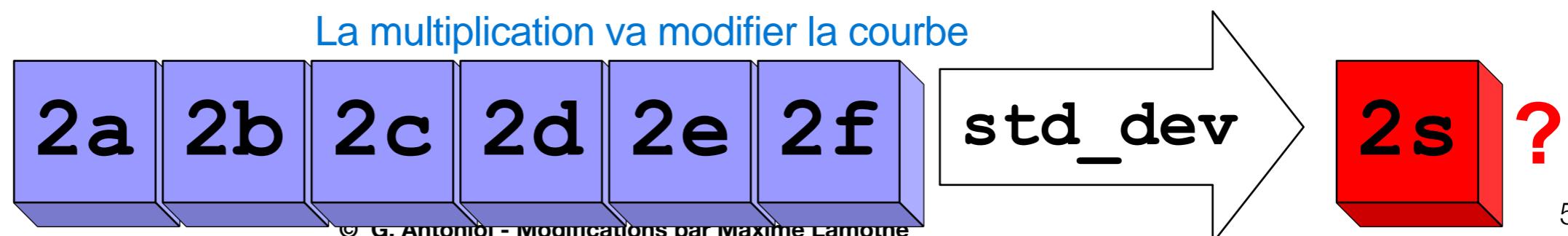
New test case #1



New test case #2



New test case #3



La multiplication va modifier la courbe

# **Propriétés métamorphiques**

---

# C. Murphy - Catégories de propriétés métamorphiques

---

## BRUIT

- **Additive:** Augmenter (ou diminuer) les valeurs numériques par une constante
- **Multiplicative:** Multiplier des valeurs numériques par une constante
- **Permutative:** Permuter aléatoirement l'ordre des éléments d'un ensemble
- **Invertive:** Nier les éléments d'un ensemble
- **Inclusive:** Ajouter un nouvel élément à un ensemble
- **Exclusive:** Supprimer un élément d'un ensemble
- **Compositional:** Composer un ensemble

# C. Murphy - Classifiers

## propriétés métamorphiques

1. *La permutation de l'ordre des exemples dans les données d'apprentissage ne devrait pas affecter le modèle*
2. *Si toutes les valeurs d'attribut dans les données d'apprentissage sont augmentées d'une constante positive, le modèle doit rester le même*
3. *La mise à jour d'un modèle avec un nouvel exemple devrait produire le même modèle créé avec des données d'entraînement contenant à l'origine cet exemple*
4. *Si toutes les valeurs d'attribut dans les données d'apprentissage sont multipliées par -1 et qu'un exemple à classer est également multiplié par -1, la classification doit être la même*
5. *La permutation de l'ordre des exemples dans les données de test ne devrait pas affecter leur classification*
6. *Si toutes les valeurs d'attribut dans les données d'apprentissage sont multipliées par une constante positive et qu'un exemple à classer est également multiplié par la même constante positive, la classification doit être la même*
7. *Si toutes les valeurs d'attribut dans les données d'apprentissage sont augmentées d'une constante positive et qu'un exemple à classer est également augmenté de la même constante positive, la classification doit être la même*

# Autres catégories de propriété

- |

---

- Statistique
  - Même moyenne, variance, etc. que l'original  
*si on ajoute/enlève une constante*  
donc on ne va pas modifier la courbe, SAUF en multiplication
- Heuristique
  - A peu près égal à l'original  
*si on fait une approximation, petite modification, on va être à peu près égal à l'original*
- Sémantiquement équivalent
  - Spécifique au domaine  
*dépend de l'application*

# Autres catégories de propriété -

## II

---

- Basé sur le bruit
  - Ajouter/Modifier des données (ou de petites variations/bruits) qui ne devraient pas affecter le résultat
- Partiel
  - La modification d'une partie de l'entrée n'affecte qu'une partie de la sortie
- Composition
  - La nouvelle entrée repose sur la sortie d'origine
    - $\text{CheminLePlusCourt}(a, b) =$ 
      - $\text{CheminLeplusCourt}(a, c) + \text{CheminLe PlusCourt}(c, b)$

# Étant donné une image

---

## Objets

- Ne changent pas leur nature si:
  - Nous les faisons pivoter
  - Nous changeons la couleur
  - Nous changeons les ombres
  - Nous superposons des effets tels que pluie, brouillard, ombre(s)
  - deux ou plusieurs objets du même type sont dans une image
  - ils restent toujours du même type
  - si un objet différent est placé dans la scène, les objets d'origine sont toujours là et inchangés

# Graphes

---

- Étant donné un graphe orienté
  - le chemin le plus court reste le même si on inverse les bords
  - le débit maximal reste le même si nous inversons les arêtes
  - les composants fortement connectés restent les mêmes si nous inversons les arêtes

# Strings

---

- Si une chaîne X contient la sous-chaîne Y
  - Inverser X signifie que nous devrions trouver l'inverse de Y

# Fonction mathématique

---

$$\text{SqRt}(X) * \text{SqRt}(Y) == \text{SqRt}(X * Y)$$

- Test 1:  $\text{SqRt}(9) == 3$
- Gen Test 2:  $\text{SqRt}(9) * \text{SqRt}(2) == \text{SqRt}(9 * 2)$

$$\log(k * X) == \log(X) + \log(k)$$

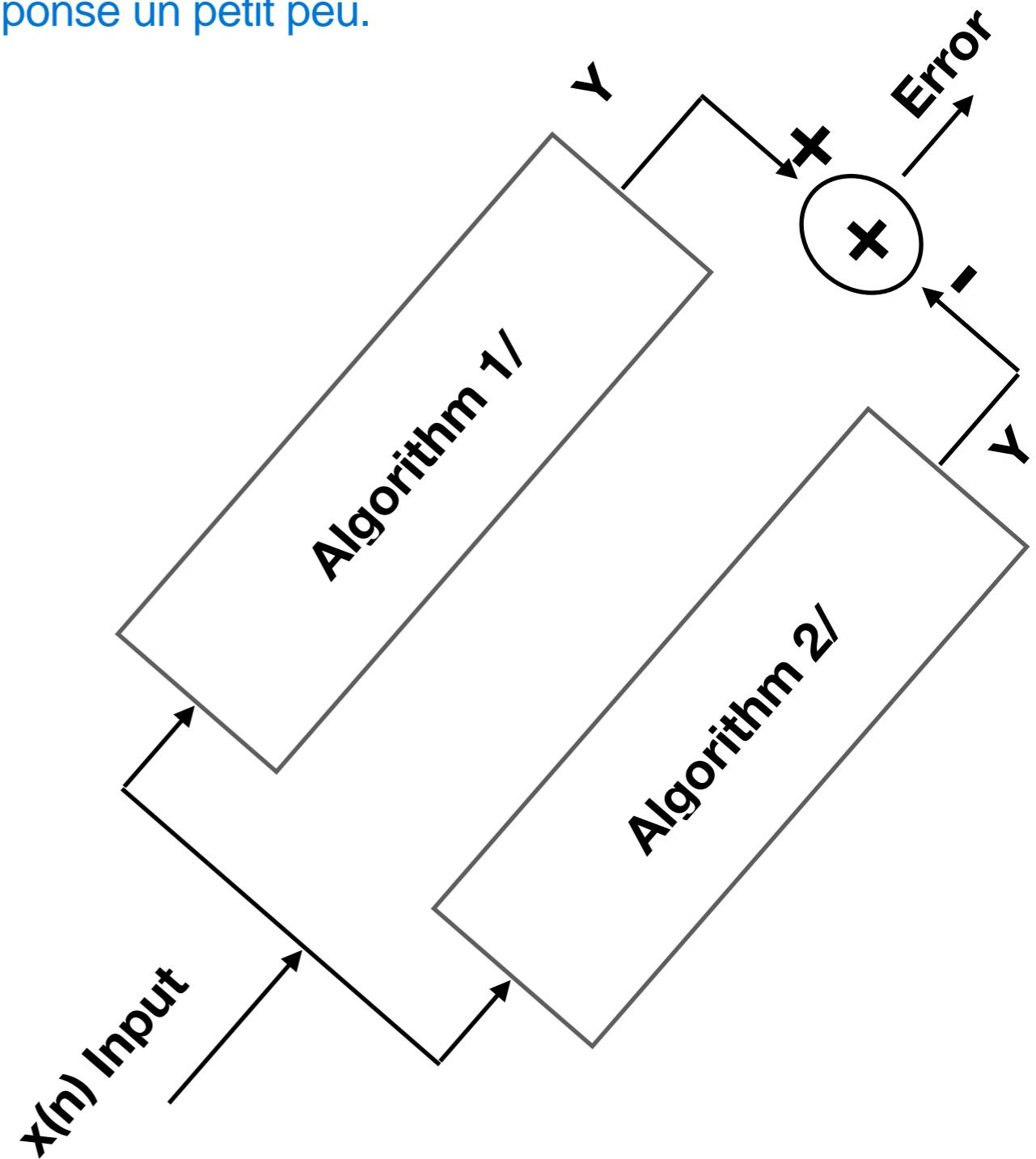
$$\exp(k * X) == \exp(X) * \exp(k)$$

$$(x+k)^2 = x^2 + k^2 + 2kx$$

# But

- Faire diverger deux classificateurs
  - une fois que les classificateurs donnent des sorties différentes, nous savons que l'un est faux
  - lancer une solution aléatoire
    - appliquer de petits changements ou exploiter les propriétés métamorphiques
    - si les propriétés sont métamorphiques, nous savons qu'elles ne devraient pas changer

On a 2 algorithmes, et on va essayer de perturber la réponse un petit peu.



# Exemples avec des images

---

- Opération sur une seule image
  - bit flip    changement de blanc à noir, on a la mm image
  - area flip
  - blur
  - rotation    rotation de l'image, ne devrait pas changer la reconnaissance d'image
  - re-scaling
  - changement de couleur ....



# Reconnaissance d'objets : augmentation du jeu de données

- Toute opération métamorphique ne changera pas l'objet

- bit flip
- area flip
- blur
- rotation
- re-scaling
- color change ....

**Calculer automatiquement les boîtes englobantes**



# Concepts clés

---

- Tester des programmes non testables
  - Une taxonomie
  - Pseudo-oracle
- Propriétés métamorphiques
  - pour les classificateurs, quelques-uns sont connus
  - pouvons-nous les apprendre .. oui ... mais ...

# Fiabilité du logiciel

*LOG3430 Méthodes de test et de validation du logiciel*

# Supprimer x% des fautes du logiciel conduira nécessairement à x% d'amélioration de la fiabilité?

FAUX, car:

- une faute peut être responsable d'un % énorme de non fiabilité (ex: un Launcher. Le Launcher a 2 fonctionnalités, le lancement de l'app et l'app en tant que tel après. Si on a un bogue dans le lancement, on n'aura pas accès à l'app par après vu qu'on ne peut pas lancer l'app)
- corriger un bogue en amène un autre différent par exemple

V/F?

# AVAIL, MTTF, POFOD, ROCOF?

AVAIL: temps de disponibilité d'utilisation d'un système (Availability)

MTTF: temps moyen entre deux erreurs (Mean Time To Failure)

POFOD: proba d'avoir un échec lors d'une requête quelconque (Probability Of Failure On Demand) --> on ne voudrait pas mesurer que le système 911 n'est pas dispo, mais plutôt mesurer que sur un x nb d'appels dans 1h, à comb d'appels on a pu répondre. Si 1000 appels, on voudrait avoir le POFOD le + bas possible

ROCOF: Fréquence d'erreurs dans un TEMPS donné (Rate Of Occurrence Of Failure)

*Mesure du nombre de défaillances du système pour un certain nombre d'entrées de données.*

Utilisée pour estimer: POFOD

Mesure du temps (ou nombre de transactions) entre les défaillances du système.

Utilisée pour estimer: MTTF et ROCOF

*Mesure du temps de non-disponibilité causée par la défaillance.*

Utilisée pour estimer: AVAIL

Un système logiciel est utilisé pour les prises de rendez-vous. Le système a reçu 25 676 requêtes le mois dernier. 4 de ces demandes n'ont pu être traitées en raison de diverses défaillances.

Quelle métrique de fiabilité pouvons-nous utiliser avec ces informations ? Quelle est la valeur de cette métrique ?

**POFOD**

**4 / 25 676**

99,9%

ex: si five nines, ca serait  
99,999%

Un système logiciel est en cours de construction avec les exigences suivantes. Une disponibilité de « three nines » est demandée avec un MTTF minimum de sept jours. Combien de temps en moyenne le système peut-il être hors ligne pour un seul événement ? Temps moyen entre 2 erreurs: 7jrs

On veut que notre système soit disponible 99,9% du temps et un MTTF minimum de 7 jrs (on ne veut pas avoir d'échec en dedans des 7 jrs). Faut attendre 7jrs pour voir le prochain échec.

$$99,9\% = 0,999$$

$$0,999 = 7 \text{ jrs} \rightarrow 0,999 * 7$$

$$= 6,993 \text{ jrs} = 167,832 \text{ heures} = 10069,92 \text{ minutes} \rightarrow \text{temps de disponibilité}$$

Three Nines: Temps d'indisponibilité:  
 $7 \text{ jrs} - 6,9993 \text{ jrs} = 0,007 \text{ jrs}$   
 $= 0,168 \text{ hrs} = \sim 10 \text{ min.}$  Donc syst  
 peut être indisponible 10 min tous  
 les 7 jrs

MTTF Temps Indisponibilité:

Une année contient 365 jrs = 52 sem

7 jrs dans une semaine

$52 * 7 = 364$  jrs qu'on pourrait avoir des échecs

Donc, on a 1 journée entre 365 et 364 qu'on pourrait avoir des échecs.

On prend cette journée, on la sépare en hrs, min, sec et on l'étale sur l'année au complet. **Approches statistique**

1 jrs = 24hrs = 1440 min / 52 échecs = ~27 min

Selon le MTTF, sur une base annuelle on a un échec à tt les 7jrs d'une durée de 27 min

Quel critère choisir entre Three nines et MTTF?

- On choisit le Three Nines, car il est + petit. Il est + robuste et + sévère et satisfait les DEUX critères (Three nines et MTTF)

# Approche statistique

- On ne sait pas quand aura lieu la prochaine défaillance de notre logiciel.
  - ⇒ Cela pourrait être dans cinq ans ... comme dans cinq secondes !
- Le temps entre les défaillances (MTBF) peut donc être vu comme un phénomène aléatoire.
- Ce n'est pas purement aléatoire par contre ... certains paramètres peuvent nous aider à prédire les prochaines défaillances.
  - ⇒ A : Un système qui possède beaucoup de défauts devrait avoir des défaillances plus fréquentes.
  - ⇒ B : Un système qui a eu des défaillances fréquentes dans le passé devrait avoir plus de défaillances qu'un système qui n'en avait que rarement.
  - ⇒ C : L'ajout de nouveau code, la modification du code existant tend à introduire de nouveaux défauts, tandis que les activités de correction de défauts fait normalement diminuer le nombre de défauts.

# *Approche statistique*

- Des centaines de modèles de fiabilités ont été développés depuis les années ‘70s afin de prédire les défaillances futures.
- Beaucoup de recherches sont faites en génie industriel afin de prédire les défaillances mécaniques.
  - ⇒ Afin de prévenir des catastrophe (ex. : défaillance d'une aile d'avion après X heures de vol).
  - ⇒ Afin de gérer les coûts de maintenance (ex. : cette machine a besoin d'un nouveau piston à tous les X heures d'opération).
  - ⇒ Afin de prévoir la durée de vie utile (ex. : cette laveuse est garantie deux ans, car ses pièces vont commencer à se briser après deux ans).
- Certains modèles ont été développés spécifiquement pour les logiciels et ... sont mieux que rien.

## *Un autre point de vue*

Au lieu de la variable aléatoire  $T$ , le temps avant la prochaine défaillance, nous pouvons modéliser en utilisant le nombre de défaillances qu'un système éprouve avant un temps  $t$ .

De manière évidente, le nombre de défaillances éprouvées avant un temps  $t$  est une variable aléatoire; c'est en fait un ***processus aléatoire***, puisqu'une fois le temps  $t$  fixé, le nombre de défaillances éprouvées est une variable aléatoire.

# *Intensité de la défaillance*

On veut diminuer l'intensité de la défaillance

Disons que  $M(t)$  est un processus aléatoire représentant le nombre cumulatif de défaillances au temps  $t$ , cela signifie que la valeur moyenne est :

$$m(t) = E[M(t)]$$

La fonction  $\lambda(t)$  , l'intensité de la défaillance, est :

$$\lambda(t) = \frac{dm(t)}{dt} = \frac{d}{dt}(E[M(t)])$$

# *Propriétés de l'intensité de la défaillance*

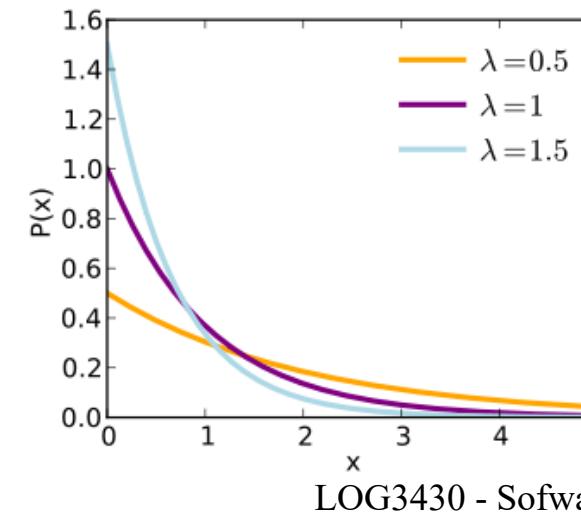
L'intensité de la défaillance représente le nombre anticipé de défaillances par unité de temps.

Le nombre de défaillances qui se produiront entre  $t$  et  $t+\Delta t$   
peut être estimé par  $\lambda(t)Dt$

L'intensité de la défaillance, devrait diminuer afin d'obtenir une croissance dans  $R(t)$ .

# *Exemple typique : Musa*

- Les modèles Musa (1975) et Musa-Okumoto (1984), malgré leurs âges, sont encore populaires pour l'estimation du nombre de défaillance à prévoir en fonction du temps.
- L'objectif est d'estimer le nombre probable de défaillances en fonction du temps d'exécution, et se basent donc sur la distribution exponentielle, avec une probabilité de défaillance  $\lambda < 1$ .



Distribution exponentielle canonique  
(source: wikipedia)

$$P(x) = \lambda e^{-\lambda x}$$

Où  $P(x)$  est la probabilité que l'événement ce produise au temps  $x$ .

# Exemple typique : Musa

- Les modèles Musa adaptent la distribution exponentielle pour leurs besoins :

$$R(t) = e^{-\lambda t}$$

⇒ Où  $R(t)$  est la *reliability function*, et représente la **probabilité qu'il n'y ait aucune défaillance** après  $t$  temps d'exécution.

C'est une fonction de **probabilité** : même si la probabilité est très faible, rien n'empêche d'avoir quatorze défaillances le même jour ...

⇒ Où  $\lambda$  est estimé à travers le type de langage de programmation, le nombre de défauts estimé restants dans le code, la taille du logiciel, le nombre d'instructions exécutées, etc.

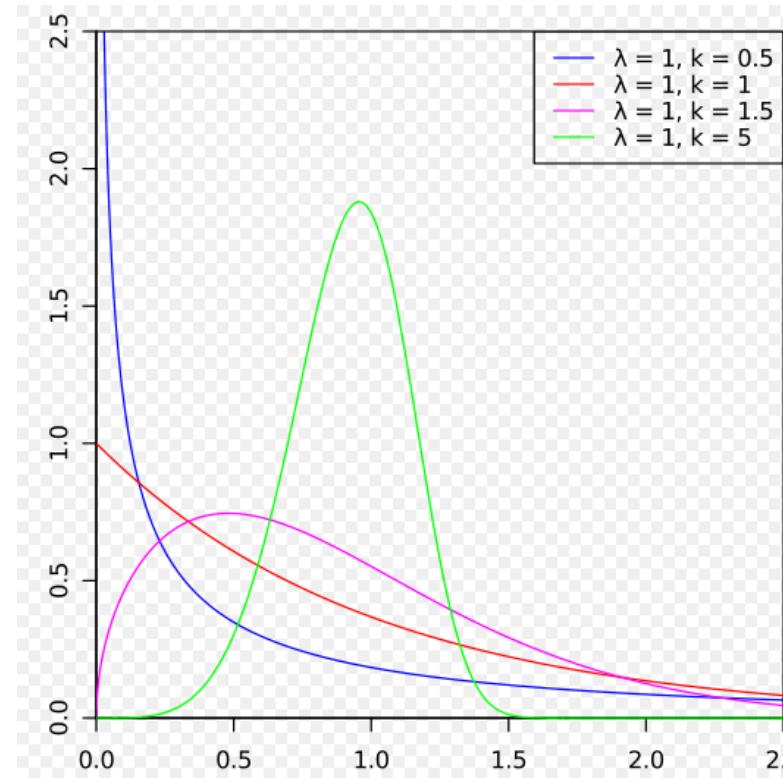
Notez qu'on ignore le nombre de défauts restants, mais qu'il existe des tables basées sur les activités d'assurance-qualité réalisées (ex. : la NASA peut obtenir un taux de 0,1 défaut / kLOC).

# *Exemple plus avancé : Weibull*

- Vous travaillez pour un jeu vidéo en ligne. Votre jeu existe depuis plusieurs années et ses défaillances sont connues.
- Vous prévoyez cependant sortir dans quatre semaines une nouvelle expansion.
- D'ici quatre semaines, le nouveau code de l'expansion sera progressivement ajouté au jeu en ligne afin de s'assurer qu'il est compatible avec le code existant.
  - ⇒ Malgré vos meilleurs efforts, il est fort probable que ce nouveau code ajoute des défauts au code existant.
  - ⇒ Le nombre de défauts va donc augmenter pendant quatre semaines, avant de diminuer à mesure que ces défauts seront découverts et éliminés.
  - ⇒ Conséquemment, la probabilité qu'une défaillance apparaisse dans les quatre prochaines semaines devrait augmenter, avant de diminuer par la suite.
  - ⇒ Y a-t-il moyen de modéliser cette période de rodage qui suit une mise-à-jour majeure ?

# Exemple plus avancé : Weibull

- La distribution de Weibull a été développée pour représenter ce genre de phénomène :



Distribution de Weibull canonique  
(source: wikipedia)

$$F(x) = \frac{k}{\lambda} \left(\frac{x}{\lambda}\right)^{k-1} e^{-(x/\lambda)^k}$$

Où  $F(t)$  est la probabilité qu'une défaillance (*failure*) se soit produite au temps  $t$ .

Où  $k$  représente le taux d'augmentation des défaillances ( $k>1$ ) ou de diminution des défaillances ( $k<1$ ).

Où  $\lambda$  représente l'intensité des défaillances précédentes.

## Exemple plus avancé : Weibull

- Dans notre cas, le taux de défaillance va augmenter, donc  $k > 1$ .
  - ⇒ Il existe des tables permettant d'estimer la valeur de forme (*shape parameter*)  $k$ . Ce facteur permet d'indiquer sur combien de temps ces défauts seront ajoutés.
- Les défaillances précédentes connues de notre jeu et le nombre de défauts injectés par les mises-à-jour serviront à déterminer le facteur d'échelle (*scale parameter*)  $\lambda$ .

# *Exemple plus avancé : Weibull*

- Votre équipe d'assurance-qualité a produit trois modèles :

| Modèles                                                                                                                                               | Facteur de forme k | Facteur d'échelle $\lambda$ |
|-------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------|-----------------------------|
| <u>Modèle A</u><br>On commence la première mise-à-jour immédiatement, on teste normalement après le lancement.                                        | 2,0                | 20                          |
| <u>Modèle B</u><br>On retarde le lancement de 2 semaines afin de corriger plus de défauts, et on met plus de budget de tests durant les mises-à-jour. | 1,5                | 100                         |
| <u>Modèle C</u><br>On retarde le lancement de 4 semaines et on maximise le personnel sur les tests durant les mises-à-jour.                           | 1,1                | 200                         |

Votre objectif est qu'il y ait moins d'une chance sur quatre d'avoir une défaillance pendant les quatre semaines de mises-à-jour : les joueurs et joueuses s'attendent à ce qu'il y ait des problèmes, mais il ne faudrait pas qu'il y en ait trop.

Fonction de probabilité  
(PDF, Probability  
Distribution Function)

$$F(x) = \frac{k}{\lambda} \left(\frac{x}{\lambda}\right)^{k-1} e^{-(x/\lambda)^k}$$

Fonction cumulative  
(CDF, Cumulative  
Distribution Function)

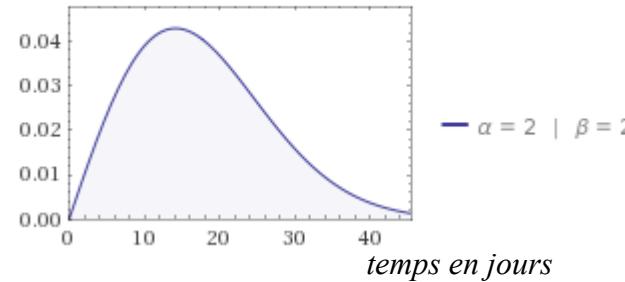
$$F(x) = 1 - e^{-(x/\lambda)^k}$$

Fonction de probabilité  
(PDF, Probability  
Distribution Function)

$$F(x) = \frac{k}{\lambda} \left(\frac{x}{\lambda}\right)^{k-1} e^{-(x/\lambda)^k}$$

Probabilité  
d'une défaillance

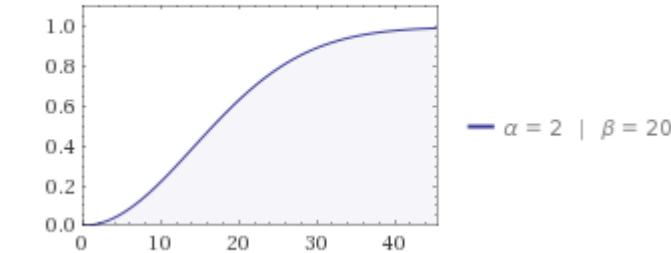
Plot of PDF:



Fonction cumulative  
(CDF, Cumulative  
Distribution Function)

$$F(x) = 1 - e^{-(x/\lambda)^k}$$

Plot of CDF:



Après 28 jours :  
 $\approx 80\%$  défaillance

Après 28 jours :  
 $<5\%$  défaillance

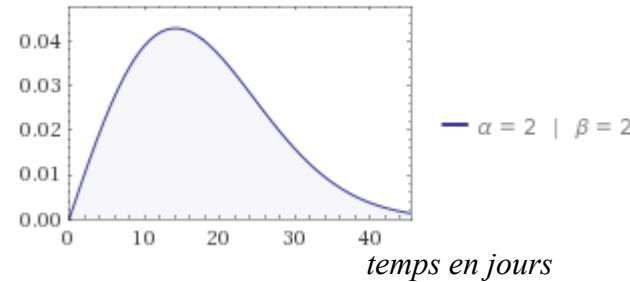
Fonction de probabilité  
(PDF, Probability  
Distribution Function)

$$F(x) = \frac{k}{\lambda} \left(\frac{x}{\lambda}\right)^{k-1} e^{-(x/\lambda)^k}$$

Fonction cumulative  
(CDF, Cumulative  
Distribution Function)

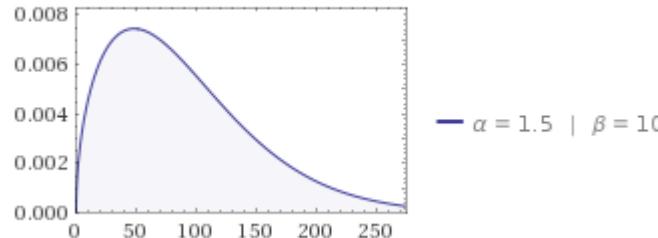
$$F(x) = 1 - e^{-(x/\lambda)^k}$$

Plot of PDF:

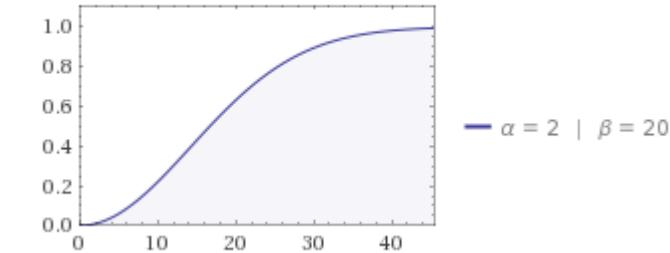


Probabilité  
d'une défaillance

Plot of PDF:

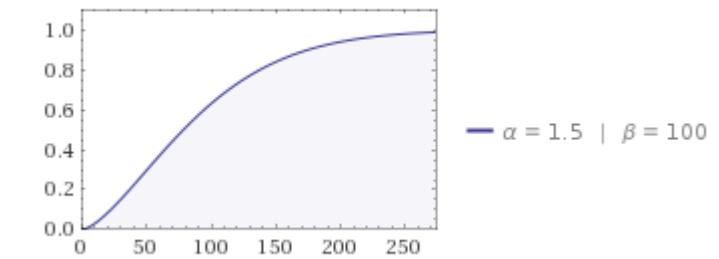


Plot of CDF:



Après 28 jours :  
 $\approx 80\%$  défaillance

Plot of CDF:



Après 28 jours :  
 $\approx 15\%$  défaillance

Fonction de probabilité  
(PDF, Probability Distribution Function)

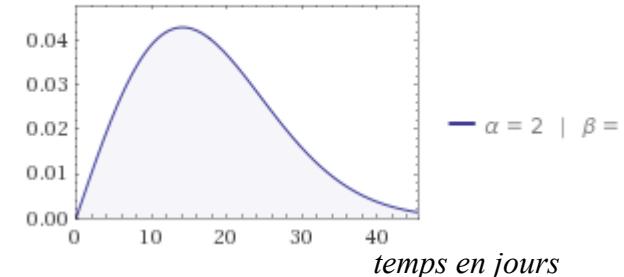
$$F(x) = \frac{k}{\lambda} \left(\frac{x}{\lambda}\right)^{k-1} e^{-(x/\lambda)^k}$$

Fonction cumulative  
(CDF, Cumulative Distribution Function)

$$F(x) = 1 - e^{-(x/\lambda)^k}$$

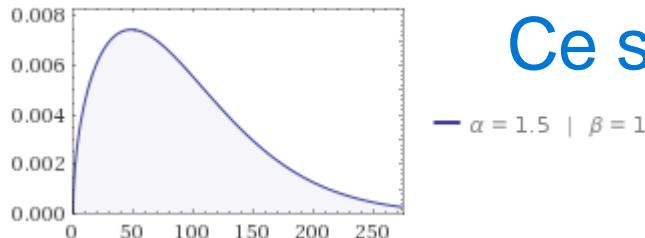
Plot of PDF:

Probabilité d'une défaillance



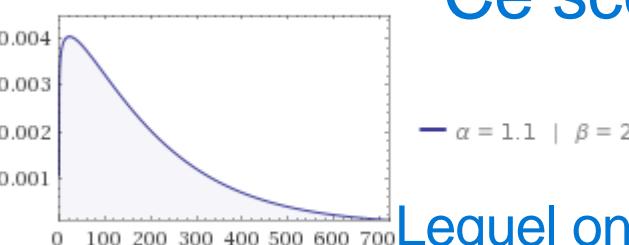
Ce scénario A ne fonctionne pas car > 25% après 28 jrs

Plot of PDF:



Ce scénario B fonctionne

Plot of PDF:



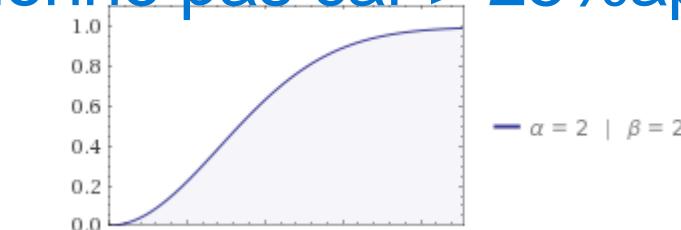
Ce scénario C fonctionne

Lequel on choisit? B ou C?

- On choisit B car nécessite - d'efforts et réponds aux exigences<sup>21</sup> et coûte - cher

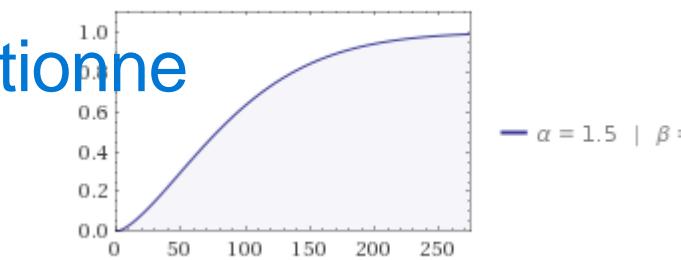
Source : modélisation faite avec Wolfram-Alpha

Plot of CDF:



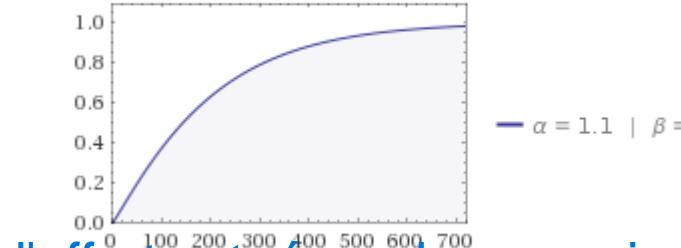
Après 28 jours :  
 $\approx 80\%$  défaillance

Plot of CDF:



Après 28 jours :  
 $\approx 15\%$  défaillance

Plot of CDF:



Après 28 jours :  
 $< 5\%$  défaillance

21

## *Exemple plus avancé : Weibull*

- Sur la base de ces graphes, le modèle B semble le plus approprié :
  - ⇒ Le modèle A n'atteint pas le niveau de fiabilité demandé.
  - ⇒ Le modèle C dépasse nos attentes, mais coûtera beaucoup plus cher et le retard de livraison pourrait avoir un impact sur notre clientèle.



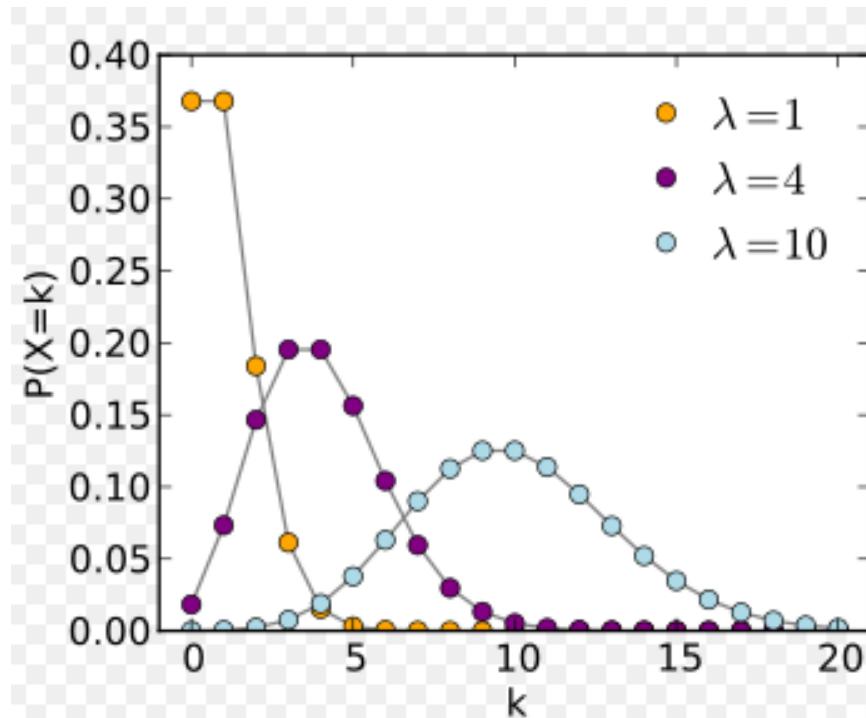
...time to change the  
company's name... 22

# Exemple : Poisson

- La distribution de Poisson est utile si nos événements sont discrets
- Une autre utilité de la distribution de Poisson est si au lieu d'estimer le temps d'exécution  $t$  avant la prochaine défaillance, on veut estimer le nombre de défaillances probables au temps  $t$ .
  - ⇒ L'apparition des défaillances est un phénomène aléatoire et discret, donc la distribution de Poisson peut être utile.
  - ⇒ C'est une approche populaire en ingénierie de la fiabilité.

# Exemple : Poisson

- La distribution de Poisson représente une suite d'événements discrets.



Distribution de Poisson canonique  
(source: wikipedia)

$$P(k) = \frac{\lambda^k e^{-\lambda}}{k!}$$

Où  $P(k)$  est la probabilité que  $k$  défaillances soient observées durant la période de temps donnée.  
Où  $k$  représente le nombre de défaillances à observer.  
Où  $\lambda$  représente le taux moyen de défaillances pour un intervalle de temps donné (ex.: une défaillance par mois).

# *Exemple : Poisson*

- Encore une fois, l'utilisation d'une distribution de Poisson pour l'estimation du temps moyen pour atteindre  $k$  défaillance se base sur certaines hypothèses :
  - ⇒ Les défaillances sont indépendantes les unes des autres.
  - ⇒ Le taux moyen de défaillance est constant. Cela implique qu'on ne corrige plus de défauts durant la période de temps estimée → utile pour évaluer ce qui va se passer si on arrête de tester maintenant.
- Si le taux moyen de défaillance est constant, nous disons qu'il s'agit d'un processus homogène de Poisson (Homogeneous Poisson Process, HPP).
- Si le taux moyen de défaillance n'est pas constant, alors le modèle change un peu, et on dit qu'il s'agit d'un processus non-homogène de Poisson (Non-Homogeneous Poisson Process, NHPP).

# *Exemple : Poisson*

- Prenons le cas suivant :  $\lambda = 0,001$  défaillances par mois.

$$P(k) = \frac{0,001^k e^{-0,001}}{k!}$$

- Durant un mois, quelle est la probabilité d'avoir :
  - ⇒ Aucune défaillance : 99,9% de chance.
  - ⇒ Une défaillance : 1 chance sur 1 000 (équivalent à  $\lambda$ ).
  - ⇒ Deux défaillances : 1 sur 20 000.
  - ⇒ Trois défaillances : 1 sur 50 000 000.

# *Simulation de la fiabilité*

- Beaucoup des modèles de fiabilité ont besoin de données sur le logiciel
  - ⇒ Ex. : le taux de défaillance.
- Or, bien souvent, on veut estimer la fiabilité d'un système qui n'est pas encore distribué.
  - ⇒ Comment connaître le taux de défaillance d'un logiciel qui n'a jamais été utilisé ?
- Solution : Simuler l'utilisation du logiciel.
- Problème : Comment définir l'utilisation normale du logiciel, son « profil opérationnel » ?