

Test et Maintenance de Logiciel

Chapitre 4

Test fonctionnel

Ileana Ober

Maître de conférences

Université Paul Sabatier

IRIT

<http://www.irit.fr/~Ileana.Ober/>



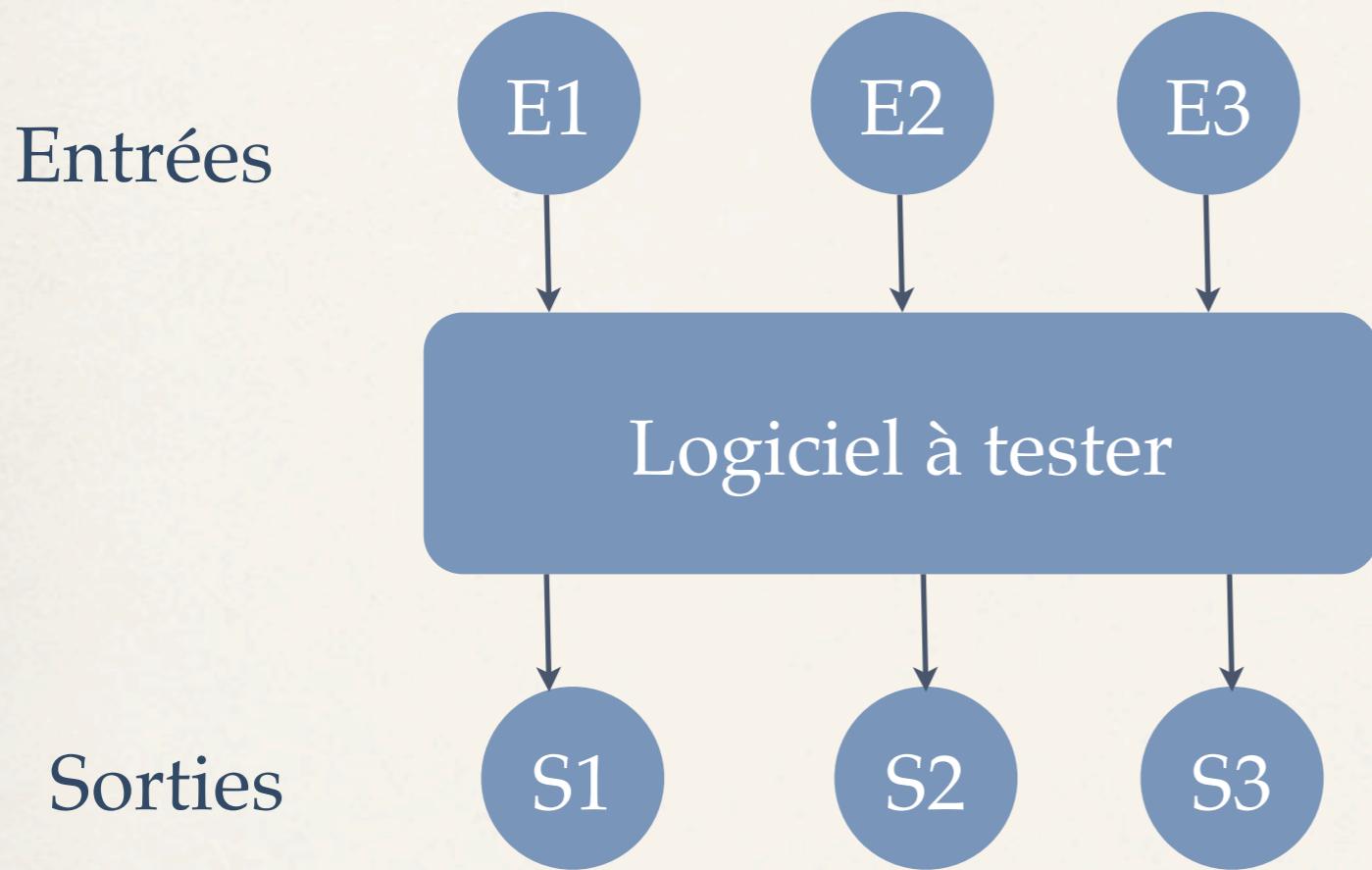
Année Universitaire 2012-2013

©Ileana Ober

Plan du cours

- ❖ Partition des domaines des entrées
- ❖ Test aux limites
- ❖ Graphes de causes à effets

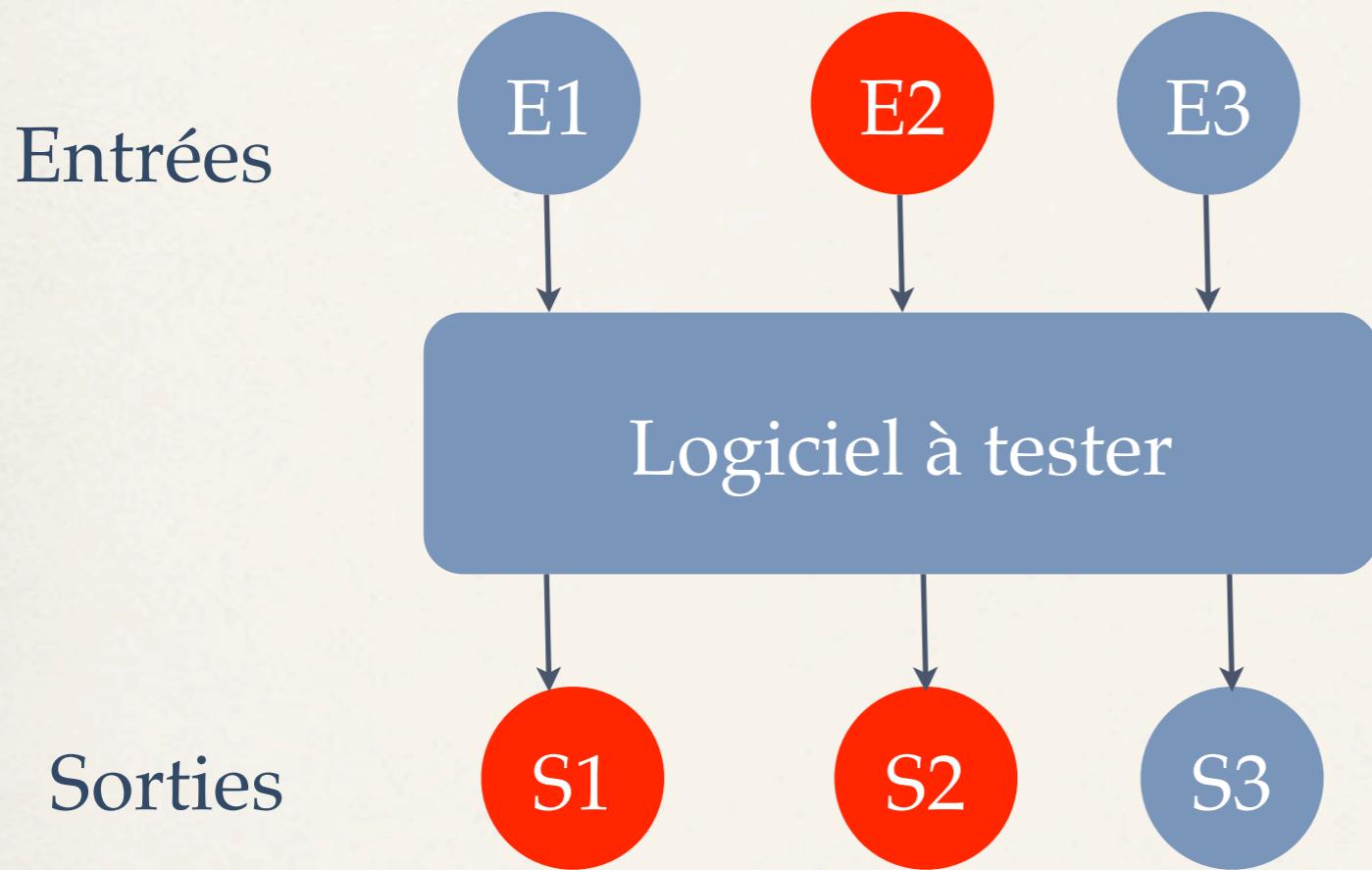
Principe du test



On ne peut pas tester toutes les **entrées/sorties**
=> on doit être malins dans la méthodologie de test

On doit trouver des dépendances entre entrées et sorties

Principe du test



On ne peut pas tester toutes les **entrées/sorties**
=> on doit être malins dans la méthodologie de test

On doit trouver des dépendances entre entrées et sorties

Partition des domaines des entrées

- ❖ **Principe:**
 - ❖ diviser le domaine des entrées en un nombre fini de classes tel que le programme réagisse pareil (en principe) pour toutes valeurs d'une classe
 - ❖ conséquence : il ne faut tester qu'une valeur par classe !
 - ❖ permet de se ramener à un petit nombre de CTs
- ❖ **Exemple**
 - ❖ tester la fonction valeur absolue `abs : int 7 → int`
 - ❖ 2^{32} ($= 4\ 294\ 967\ 296$) entrées
 - ❖ seulement 3 classes intéressantes $<0, =0, >0$
 - ❖ conséquence: on teste une seule valeur par classe, exemple -2, 0, 7

Classe d'équivalence

Définition: Deux entrées sont dans la même **classe d'équivalence**, si, d'après la spécification, elles sont traitées de la même façon.

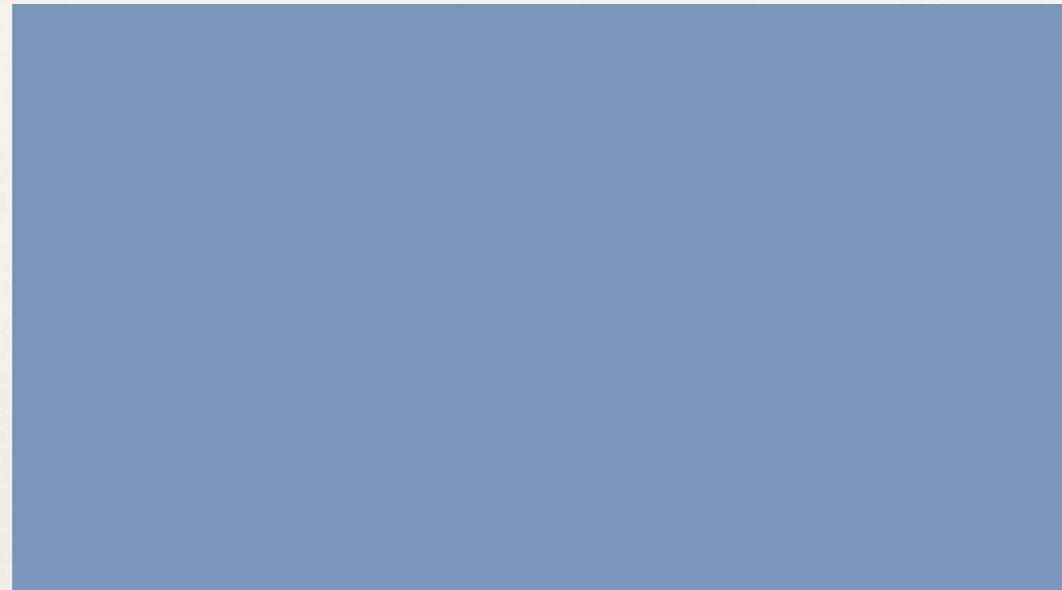
Intérêt de la stratégie des classes d'équivalence

- ✿ Tester le moins de valeurs d'entrées possibles
- ✿ Identifier et éliminer les test redondants
- ✿ Estimer le nombre de cas de test nécessaires par rapport à une spécification
- ✿ Stratégie appelée aussi : **méthode des partition des données**

Test avec des classes d'équivalence

- ❖ Besoin d'effectuer un seul test par classe d'équivalence; tester plus est redondant
- ❖ La partition du domaine des entrées en classes d'équivalence permet de trouver des tests
 - ❖ complets
 - ❖ non-redondants

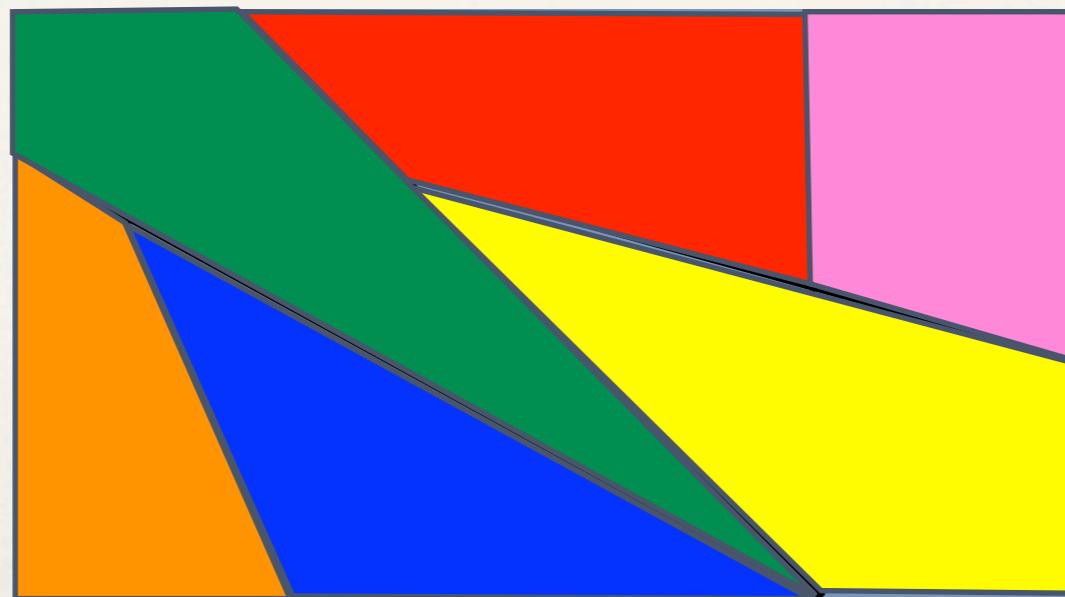
Domaine
d'une
entrée



Test avec des classes d'équivalence

- Besoin d'effectuer un seul test par classe d'équivalence; tester plus est redondant
- La partition du domaine des entrées en classes d'équivalence permet de trouver des tests
 - complets
 - non-redondants

D - Domaine
d'une entrée



Complétude:
$$D = D_1 \cup D_2 \cup \dots \cup D_6$$

Non redondance:
$$D_i \cap D_j = \emptyset, \text{ pour tout } i \neq j$$

Démarche

- ✿ Identifier les classes d'équivalence
 - ✿ Dans la spécification, identifier les exigences sur les entrées
 - ✿ Traduire chaque exigence en domaine de valeurs possibles
 - ✿ Partager le domaine en sous-domaines d'entrées disjoints, recouvrant le domaine
- ✿ Identifier les cas de test, pour couvrir
 - ✿ les *classes valides* non encore couvertes : établir un nouveau cas de test qui en recouvre le plus possible, jusqu'à ce que toutes soient couvertes
 - ✿ les *classes invalides* non encore couvertes (test de robustesse): établir un nouveau cas de test qui en recouvre une (et une seule), jusqu'à ce que toutes soient couvertes dans le domaine
- ✿ Spécifier les valeurs de sorties correspondantes

Heuristiques pour créer des partitions

Conseil 1: Attention à en faire

Conseil 2: Attention à ne pas en faire trop

| Exigence | Classes valides | Classes invalides |
|--|---|--|
| intervalle de valeurs | une classe pour les valeurs de l'intervalle | - les valeurs inférieures (si possible) - les valeurs supérieures (si possible) |
| nombre limité de valeurs | une classe pour un bon nombre de valeurs | - une classe pour trop peu de valeurs - une classe pour trop de valeurs |
| ensemble de valeurs, traitées différemment | une classe par valeur valide | - une classe pour tous les valeurs invalides |
| contrainte (obligation) | une classe pour contrainte satisfaite | - une classe pour contrainte non satisfaire |

Deux grandes types de partitions

- ❖ Crées sur la base de la spécification textuelle (cahier de charges)
 - ❖ prend en compte les relations entre variables d'entrées
 - ❖ plus pertinent
 - ❖ peu automatisable
- ❖ Crées à partir des interfaces
 - ❖ automatisable
 - ❖ basée uniquement sur les types des données d'entrée

Exemple 1

Tester une fonction qui calcule la valeur absolue d'un entier.

- type d'entrée interface textuelle

| Condition | Classe(s) valide(s) | Classe(s) invalide(s) |
|-----------------|---------------------|-----------------------|
| nb. entrées | 1 | 0, >1 |
| type entrée | integer | string |
| valeurs valides | <0, >=0 | |

Données de test:

valides: -10, 100

invalides: “xyz”, \emptyset , (1,2)

Exemple 1 - bis

Tester une fonction qui calcule la valeur absolue d'un entier.

- * type d'entrée interface “informatique”
=> on est sûrs que les valeurs sont du bon type en bon nombre

| Condition | Classe(s) valide(s) | Classe(s) invalide(s) |
|-----------------|---------------------|-----------------------|
| nb. entrées | 1 | |
| type entrée | integer | |
| valeurs valides | <0, >=0 | |

Données de test:

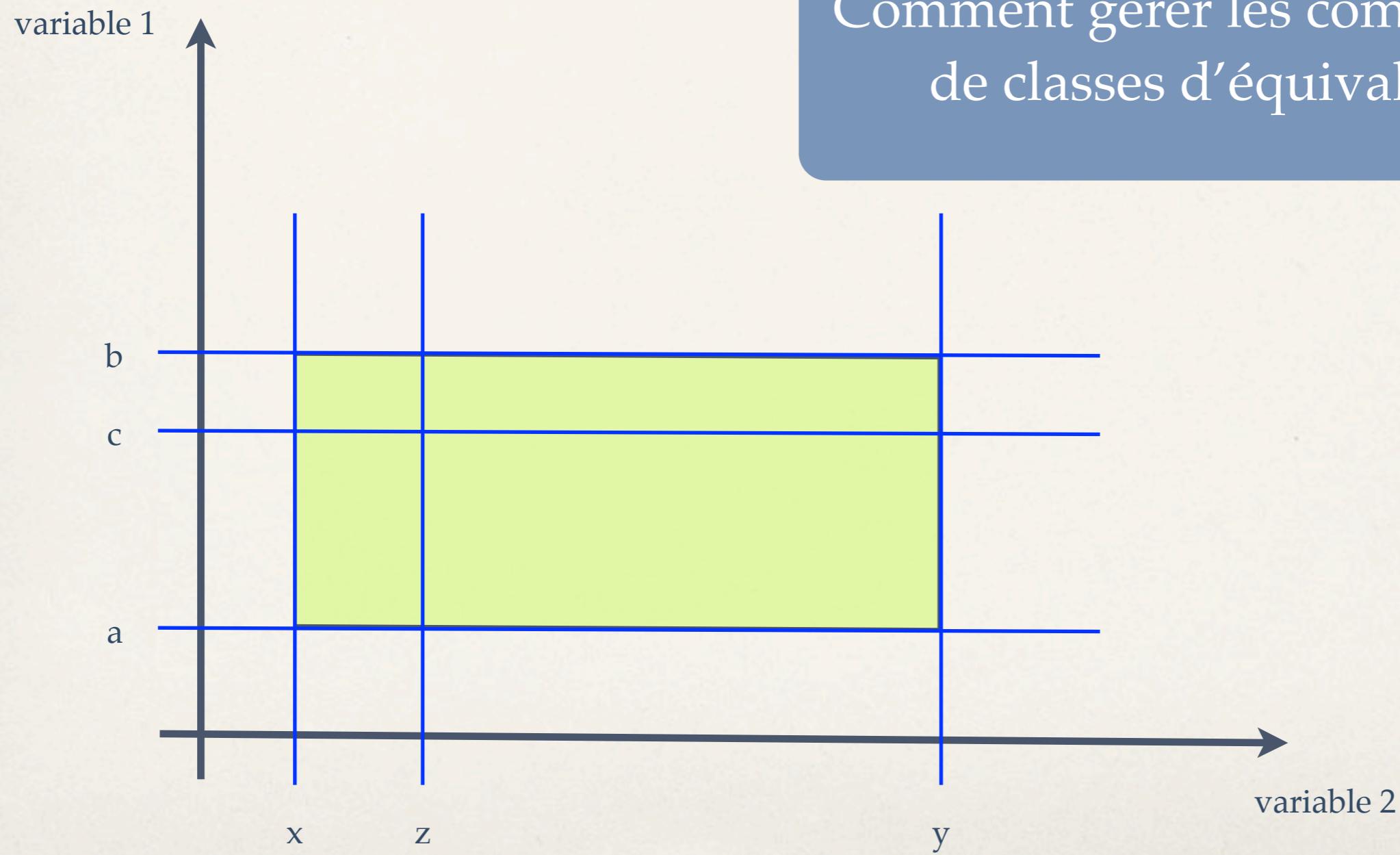
valides: -10, 100

invalides: aucune

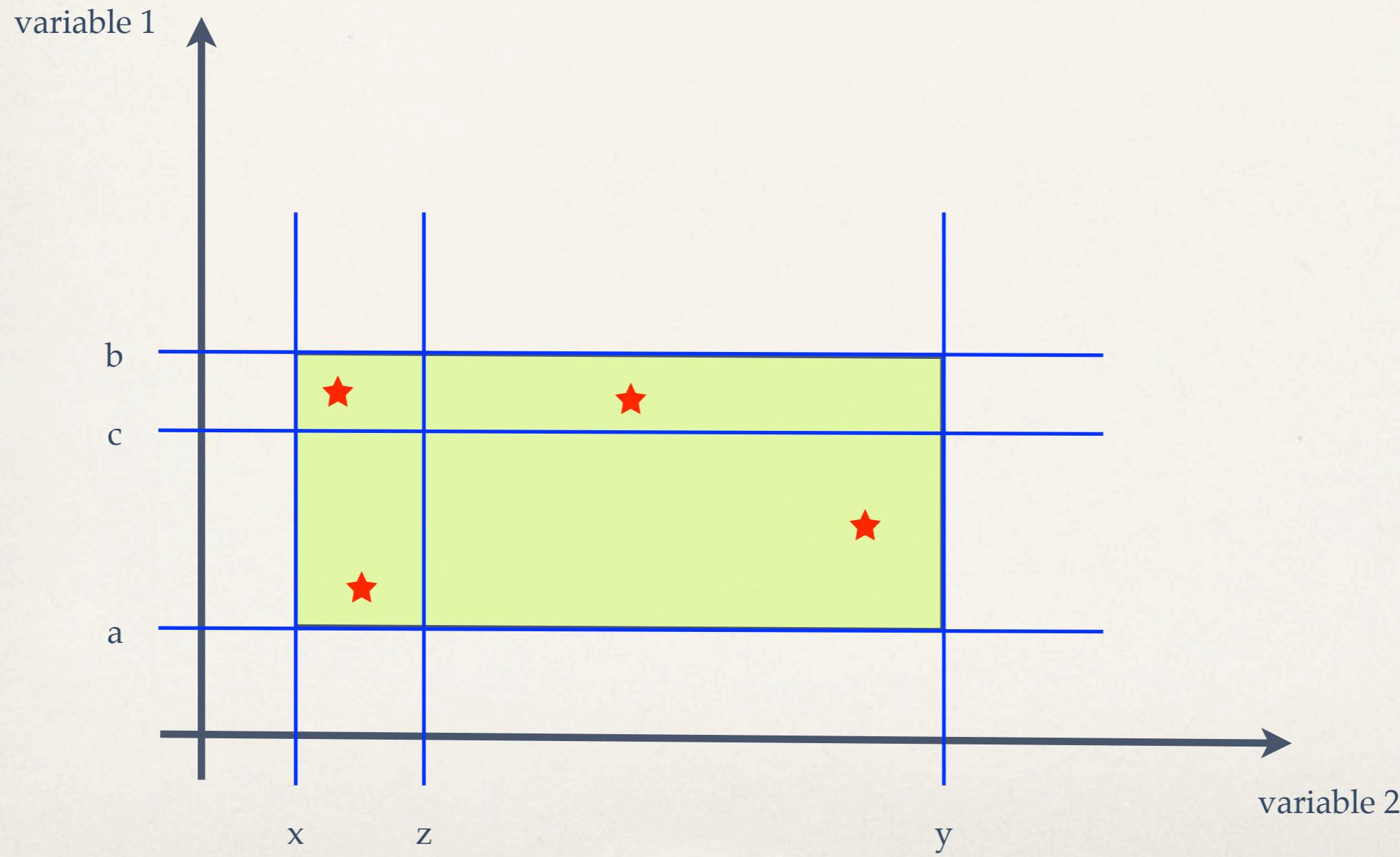
Combien cas de test?

- ❖ Un par classe d'équivalence valide
- ❖ Un par classe d'équivalence invalide (en test de robustesse)
- ❖ Que faire avec les “combinaisons” de variables?

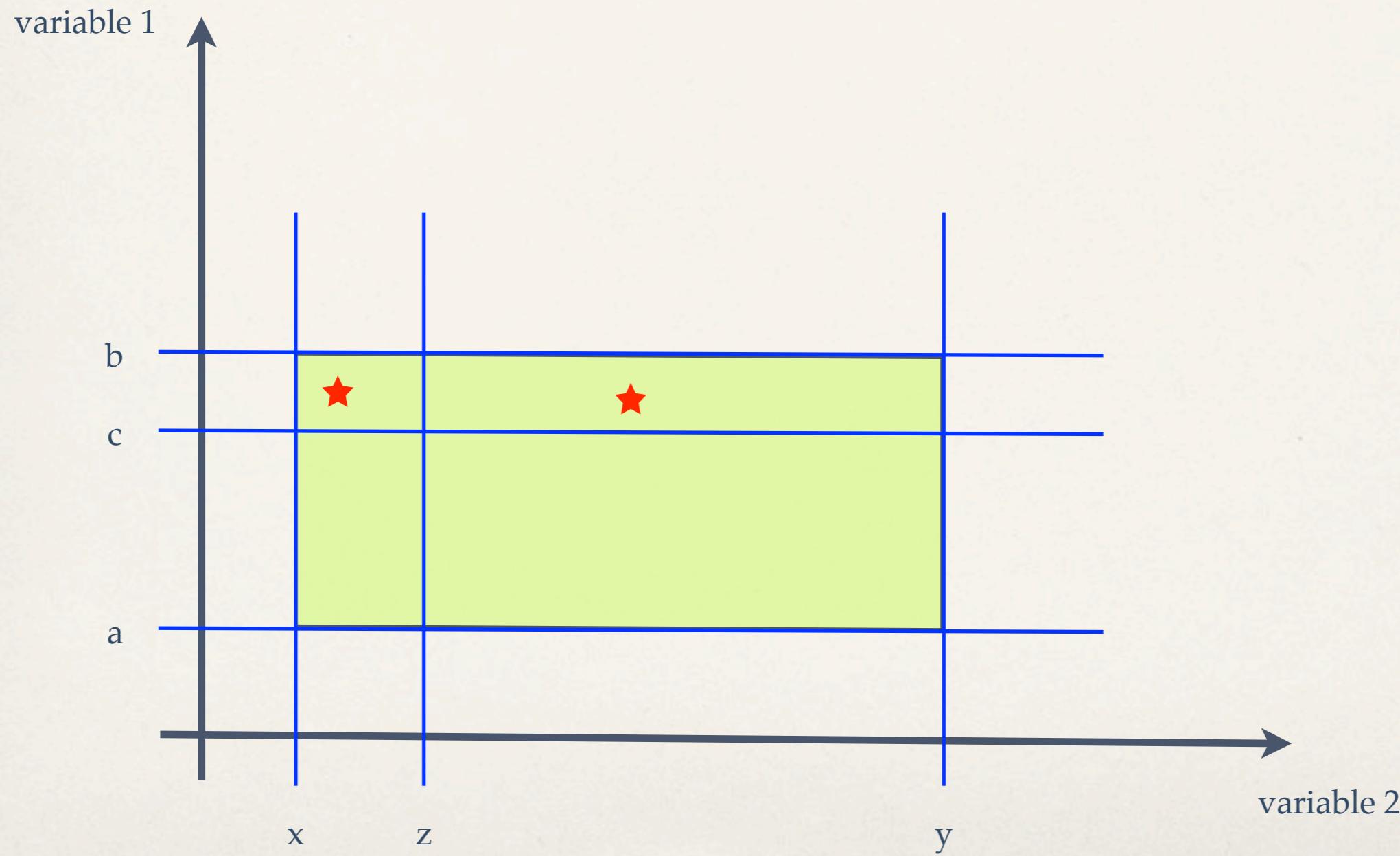
Cas de test - classes d'équivalences



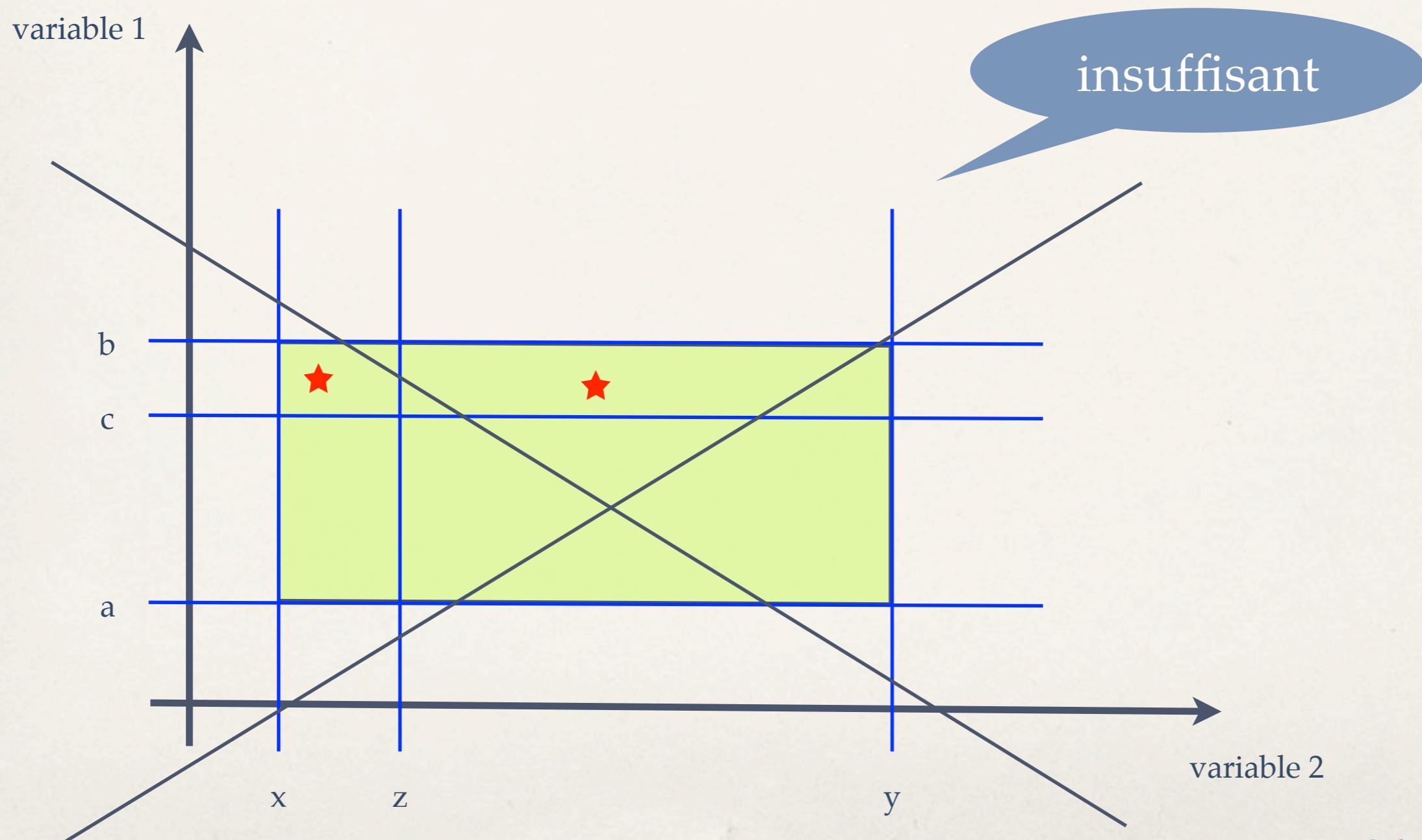
Couverture “forte” des classes d'équivalence valides (non-robuste)



Couverture “faible” des classes d'équivalence valides (non-robuste)

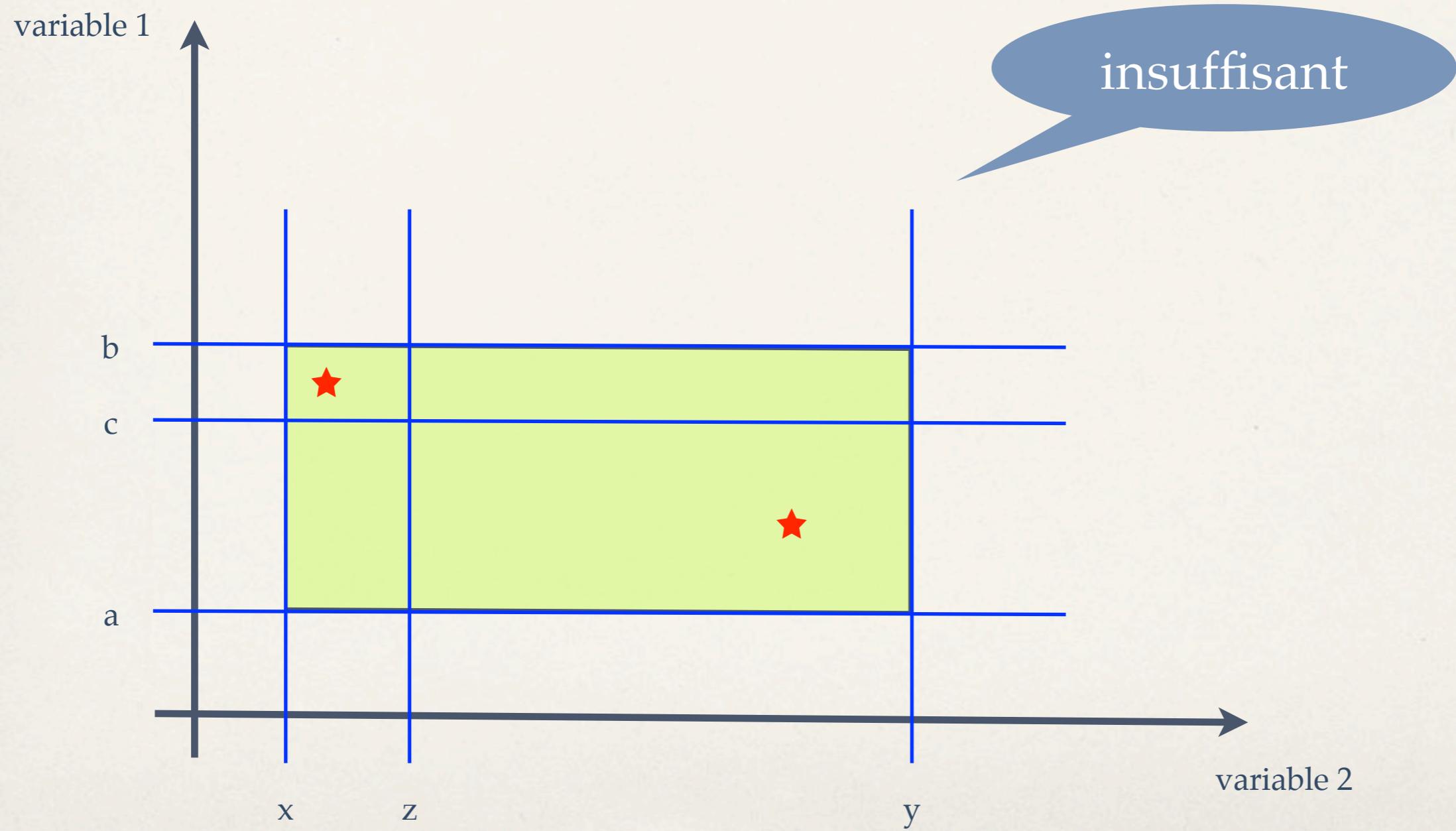


Couverture “faible” des classes d'équivalence valides (non-robuste)

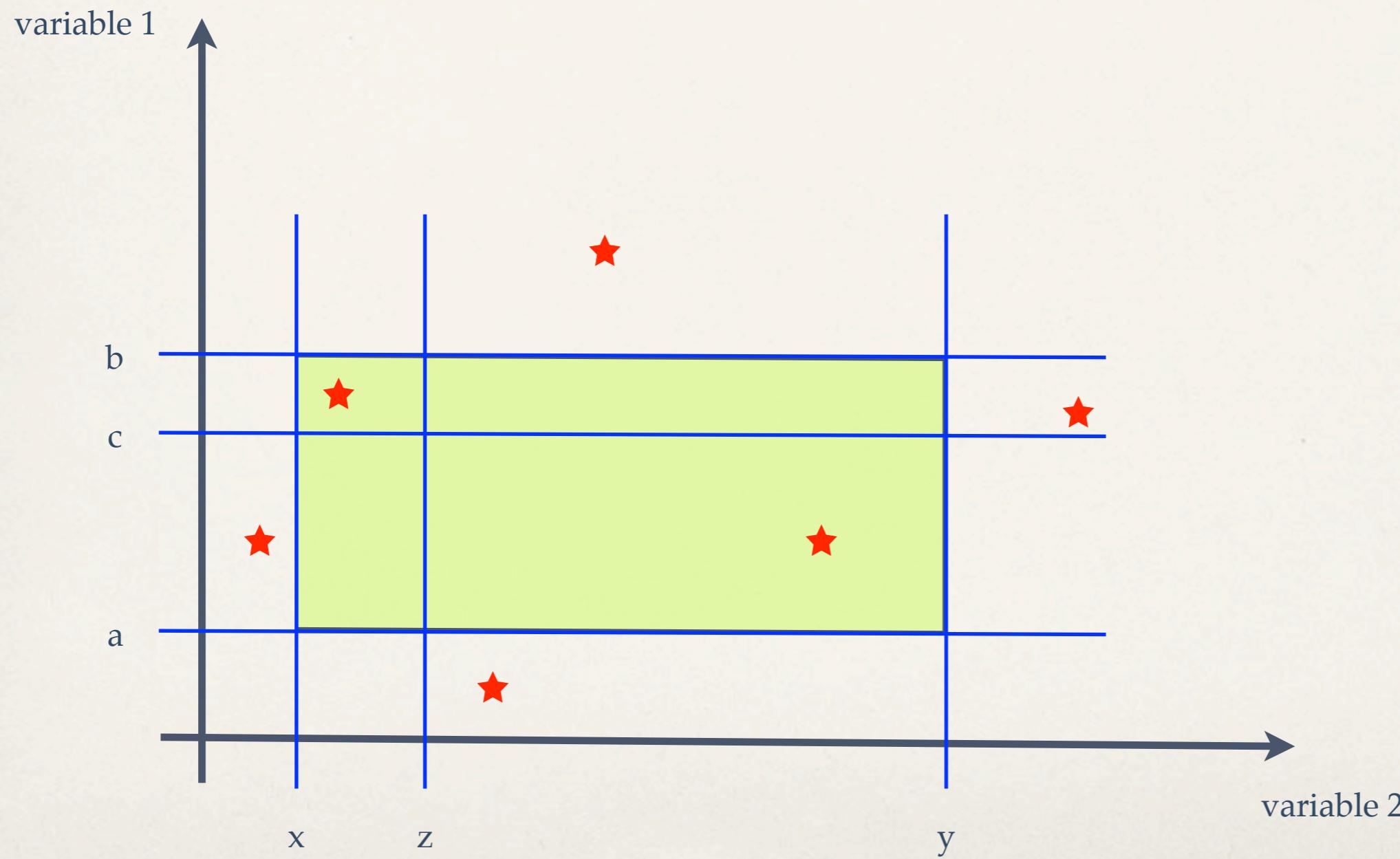


©Ileana Ober, 2012

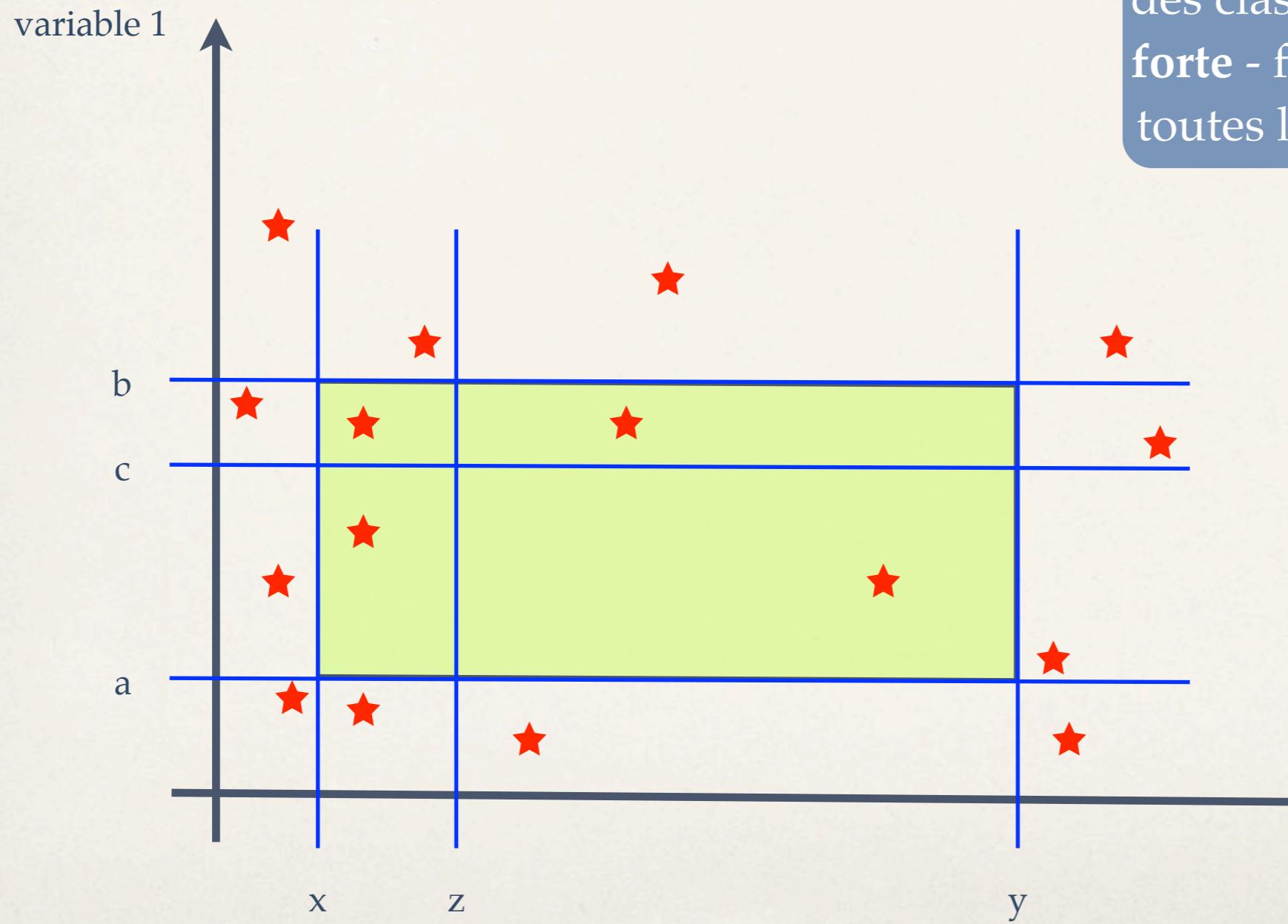
Couverture “faible” des classes d'équivalence valides (non-robuste)



Couverture robuste et faible des classes d'équivalence



Couverture robuste et forte des classes d'équivalence



robuste - fait appel à la couverture des classes invalides
forte - fait appel à la gestion de toutes les combinaisons

Exemple - compagnie d'assurances voiture

Le logiciel d'une compagnie d'assurances propose des devis à partir des informations suivantes:

- type de véhicule (voiture, moto, camion, bus)
- âge

Dans le calcul du devis, il y a 3 groupes d'âge: moins de 20, 20-65, plus de 65 ans.

Donner les jeux de test la couverture des classes d'équivalence:

- forte et faible
- robuste et non-robuste

Exemple - compagnie d'assurances voiture

Classes d'équivalence

Type de véhicule

Voiture

Camion

Moto

Bus

Age

< 20

20 - 65

>65

Combien de test
il faut prévoir?

Exemple - compagnie d'assurances voiture

Classes d'équivalence

Type de véhicule

Voiture

Age

< 20

Camion

20 - 65

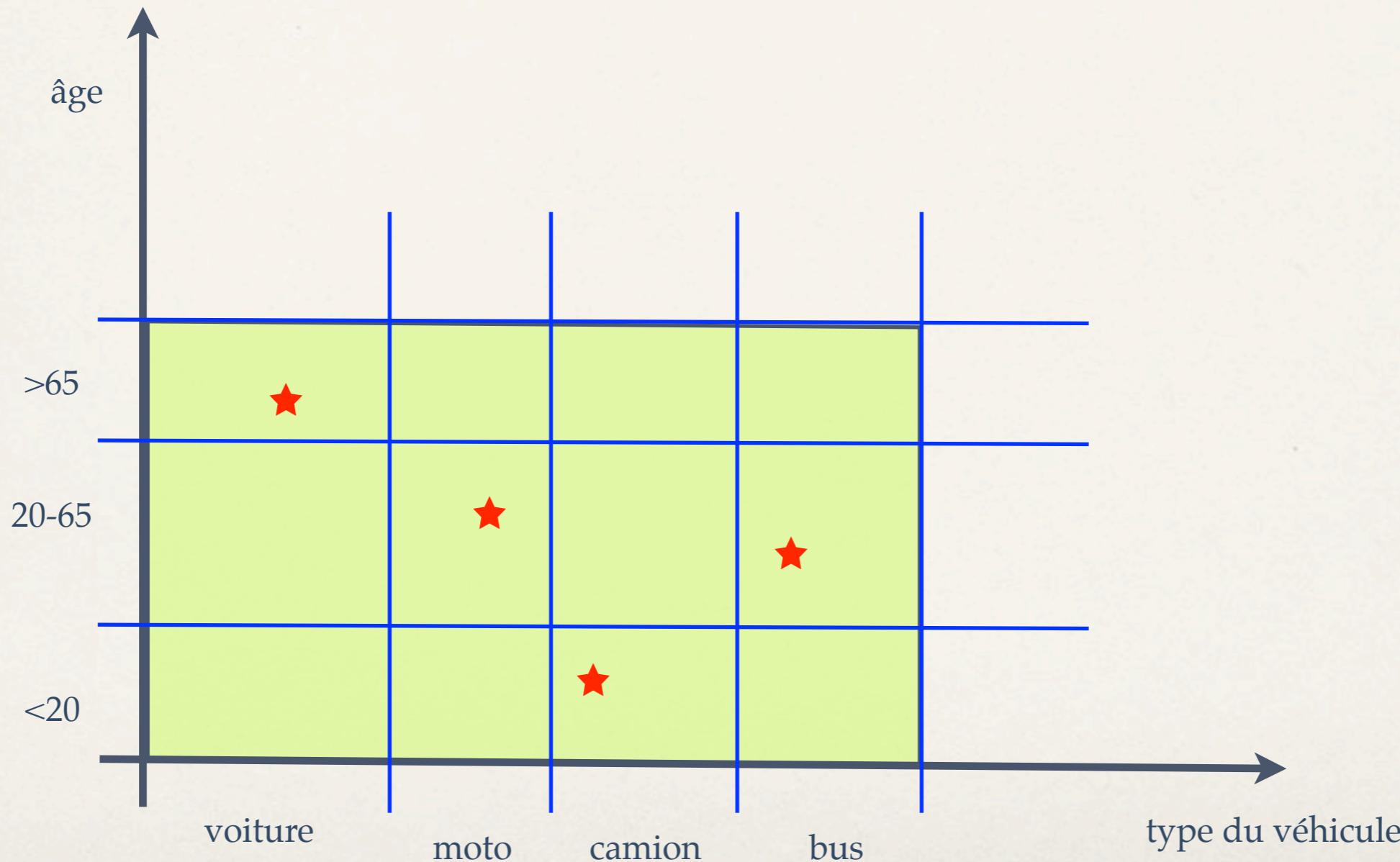
Moto

>65

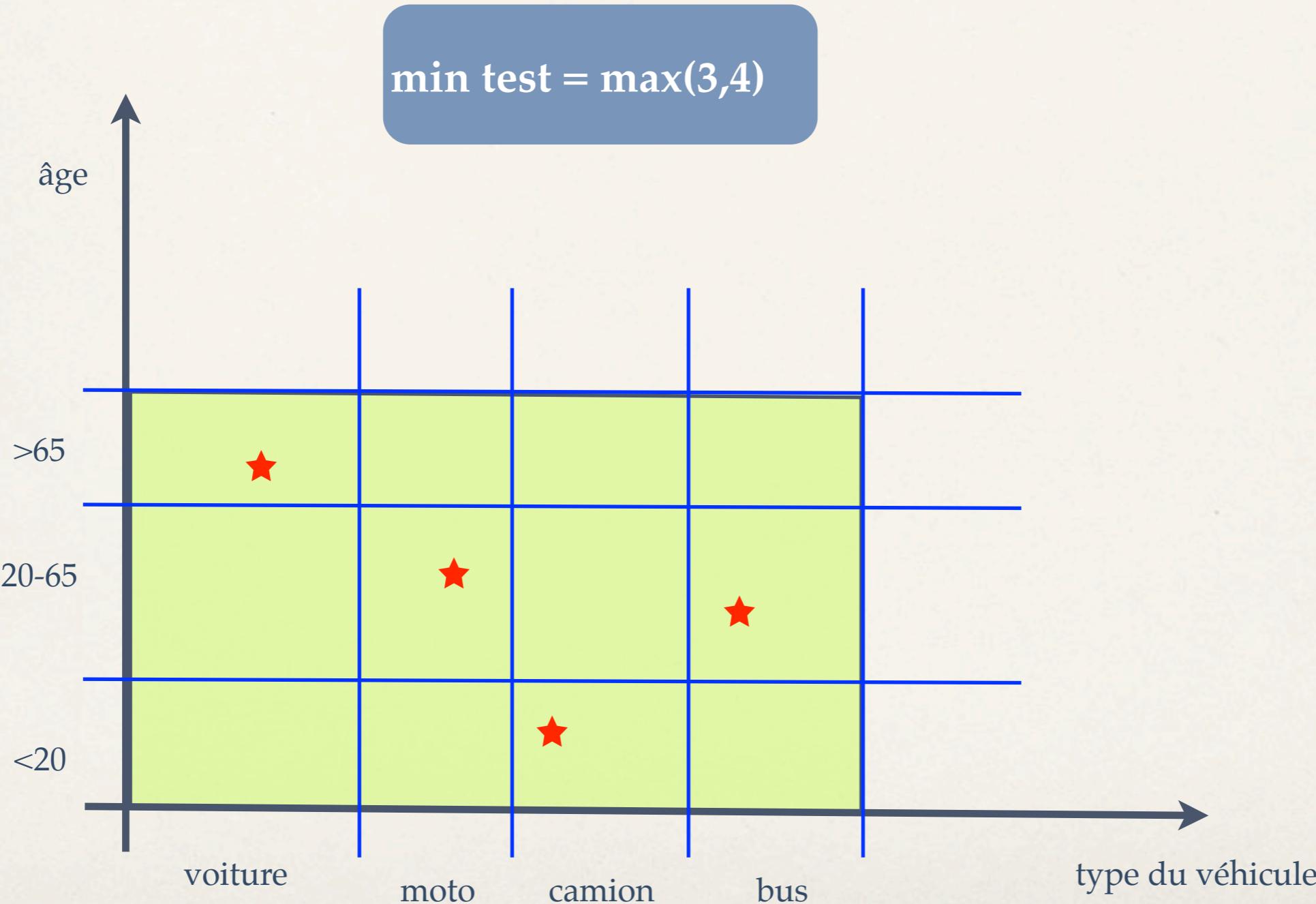
Bus

Minimum 4 tests

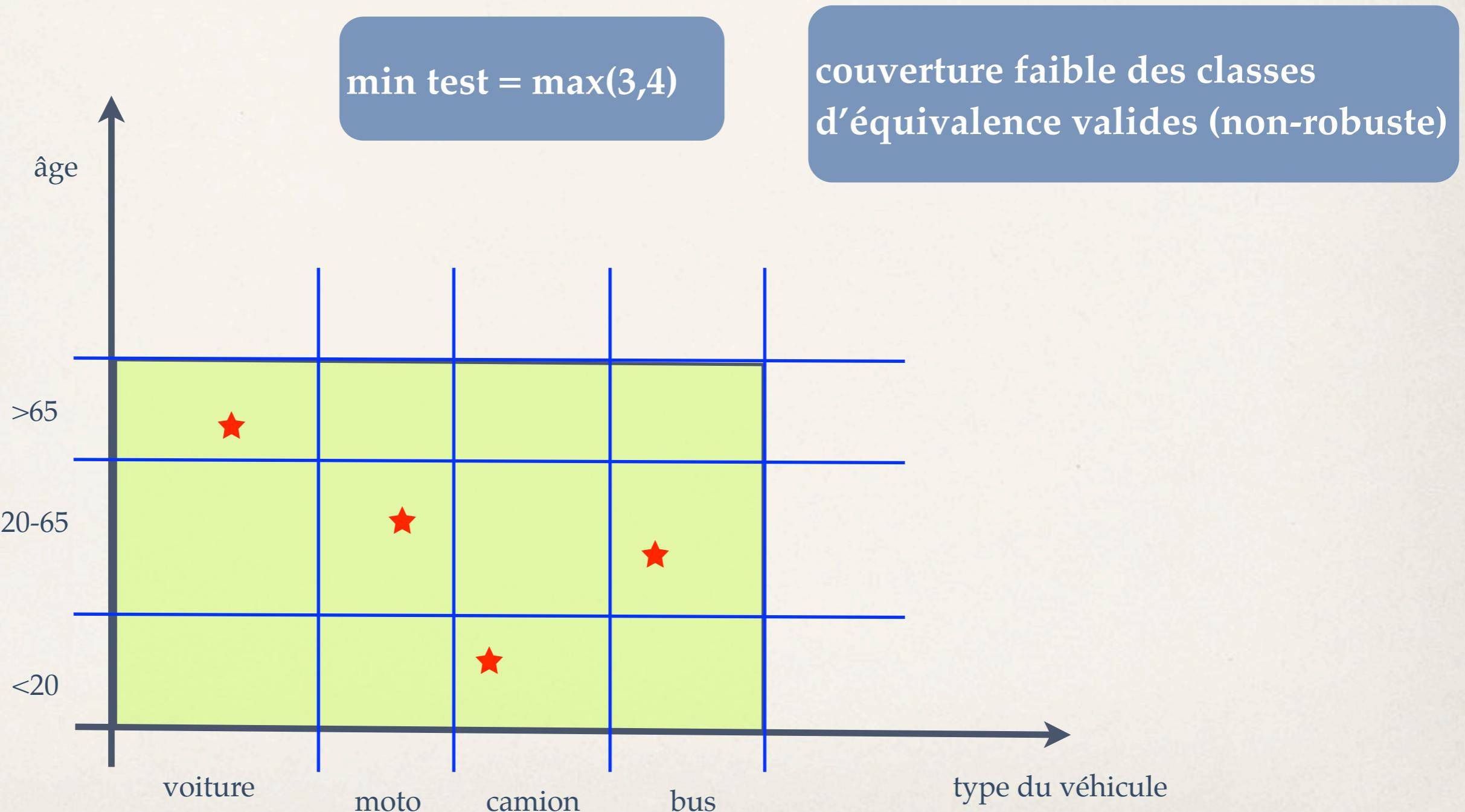
Exemple - compagnie d'assurances voiture



Exemple - compagnie d'assurances voiture



Exemple - compagnie d'assurances voiture



Exemple - compagnie d'assurances voiture

Classes d'équivalence

Type de véhicule

Voiture

Camion

Moto

Bus

Age

< 20

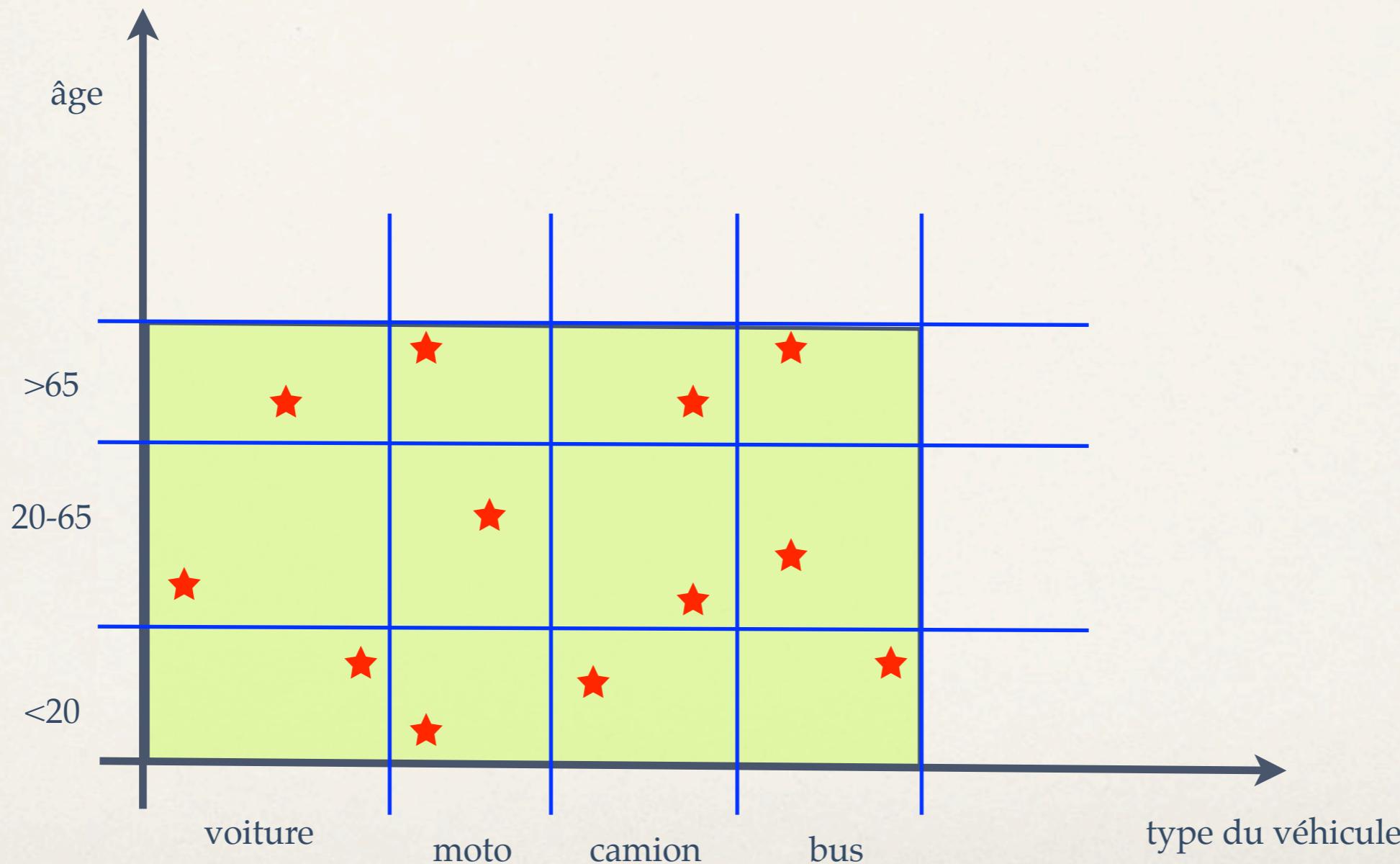
20 - 65

>65

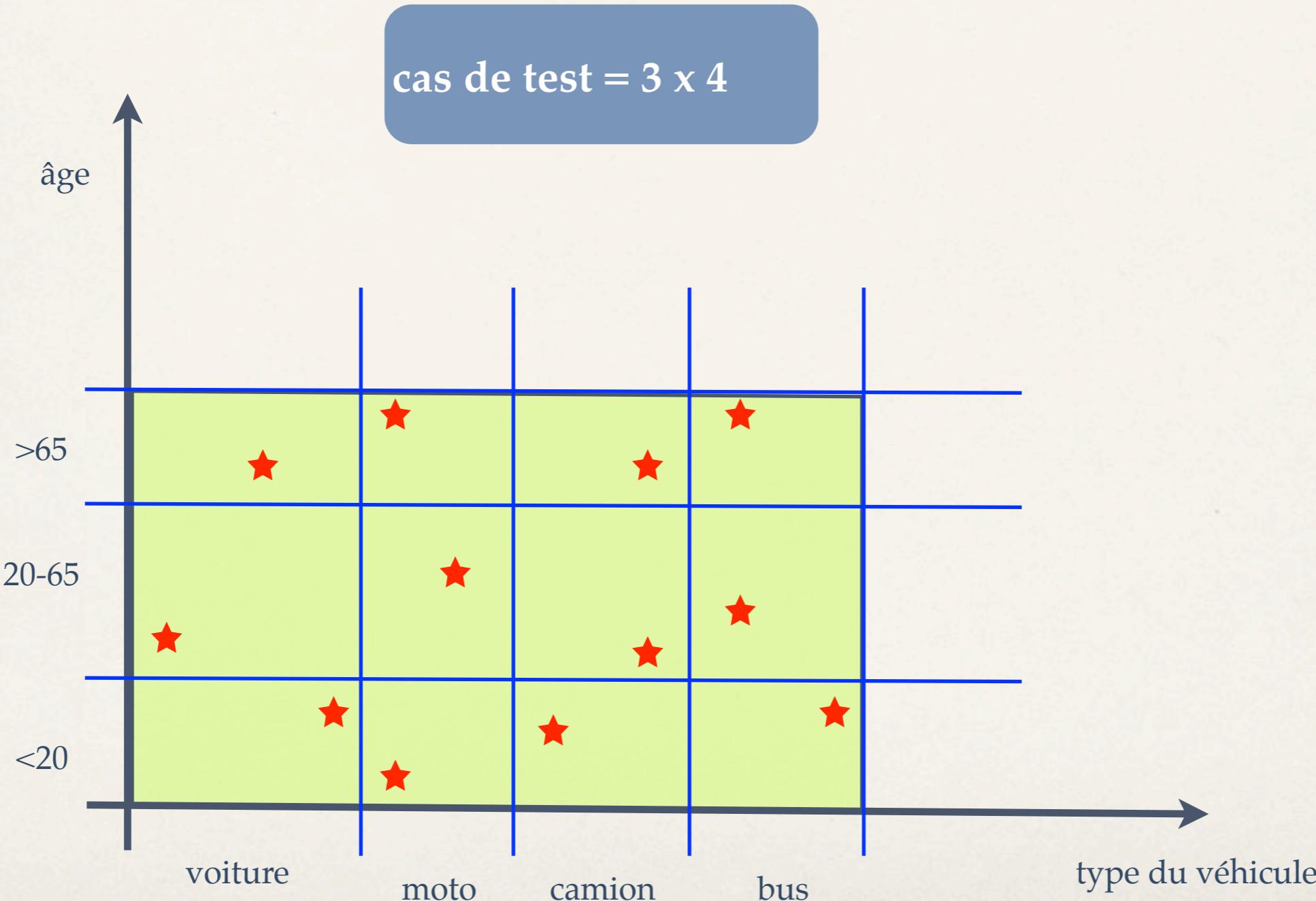
Exemple - compagnie d'assurances voiture

| Classes d'équivalence | |
|---|--|
| Type de véhicule | Age |
| Voiture <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> < 20 |
| Camion <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> 20 - 65 |
| Moto <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> >65 |
| Bus <input checked="" type="checkbox"/> | couverture faible des classes d'équivalence valides (non-robuste) |

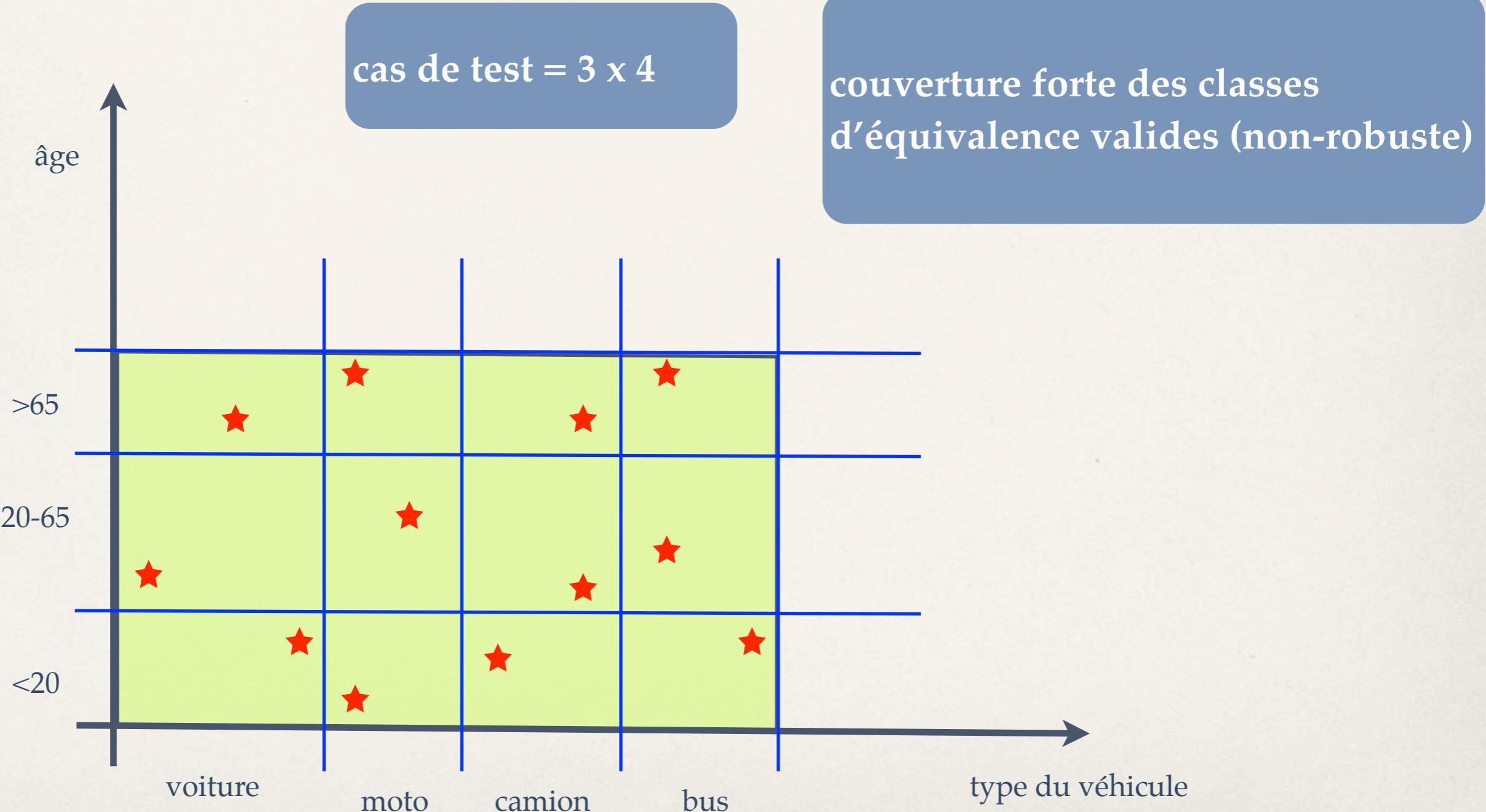
Exemple - compagnie d'assurances voiture



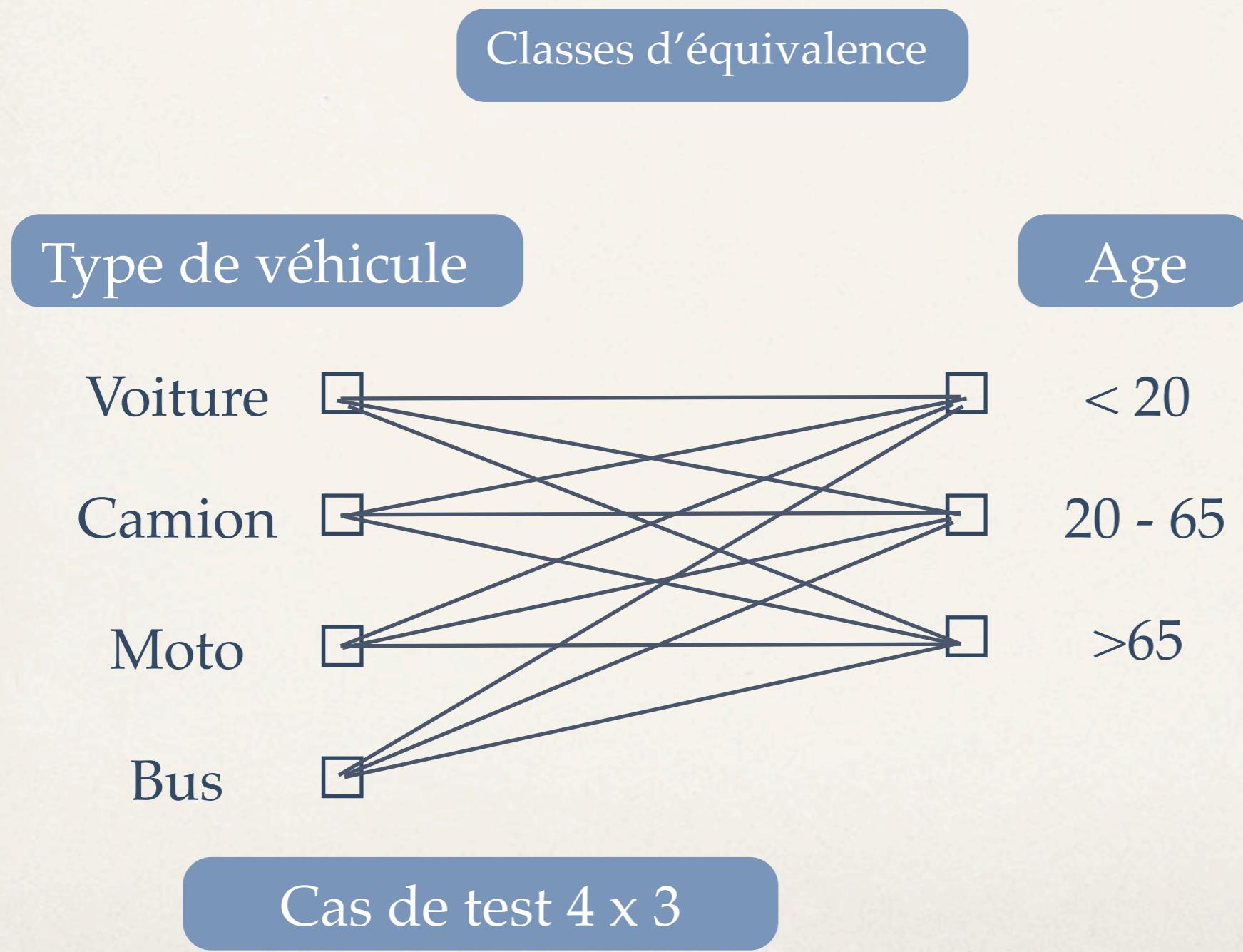
Exemple - compagnie d'assurances voiture



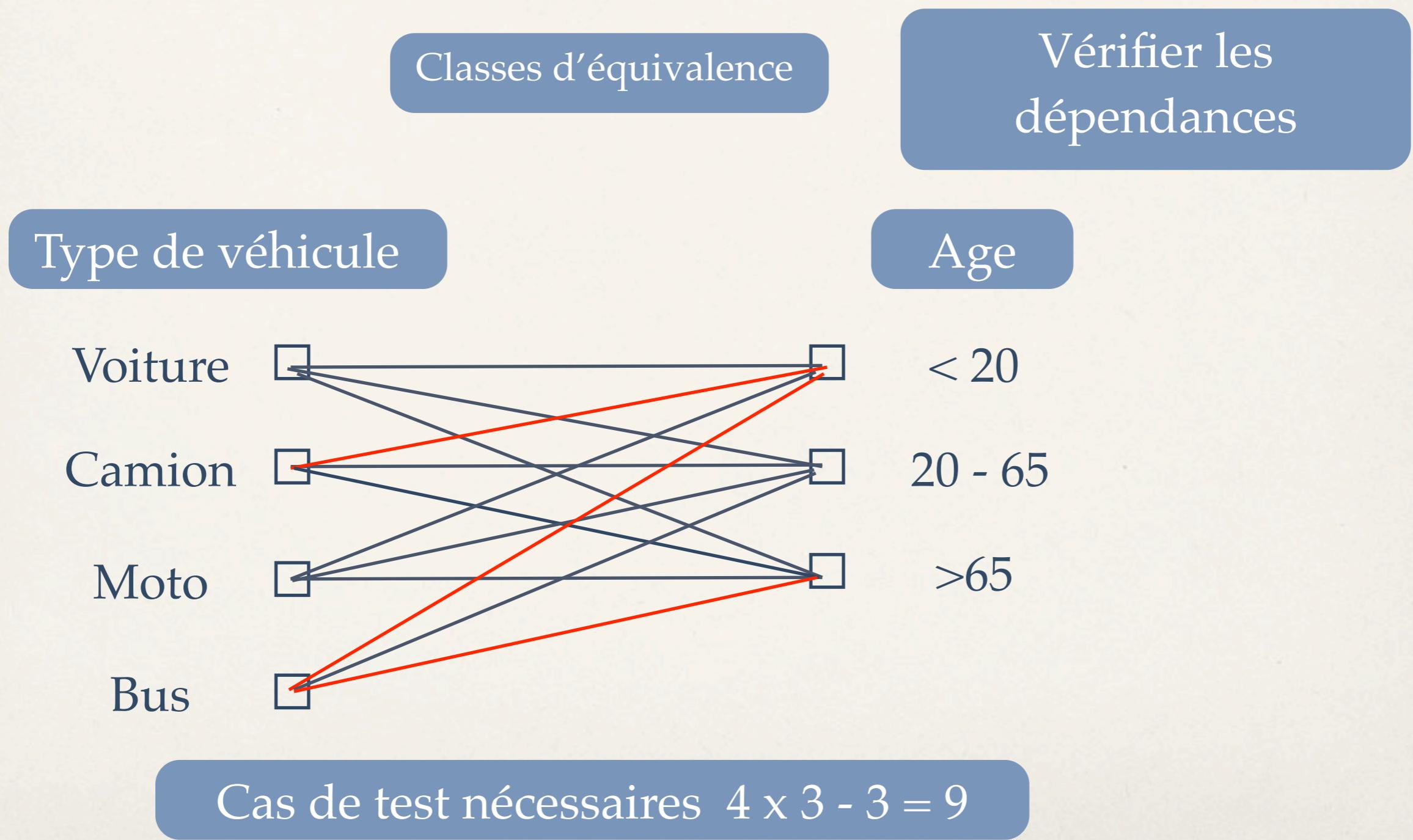
Exemple - compagnie d'assurances voiture



Exemple - compagnie d'assurances voiture



Exemple - compagnie d'assurances voiture



Test avec des classes d'équivalence

- ❖ **Avantages:**
 - ❖ couvre bien les domaines des entrées
- ❖ **Inconvénients**
 - ❖ n'exploré pas les combinaisons d'exigences sur les entrées
 - ❖ ne permet pas de détecter certains défauts de codage courants : \leq à la place de < ...

Plan du cours

- ❖ Partition des domaines des entrées
- ❖ Test aux limites
- ❖ Graphes de causes à effets

Test aux limites

- ❖ Une tactique pour améliorer l'efficacité des tests produits par d'autres familles.
- ❖ *Idée : les erreurs se nichent dans les cas limites, donc tester aussi les valeurs aux limites des domaines ou des classes d'équivalence.*
- ❖ Une forme plus agressive de test partitionnel
- ❖ Fondé sur l'étude des limites des domaines de définition des entrées mais aussi des sorties
- ❖ Conditions aux limites : situations exactement sur, juste avant et juste après les frontières des classes d'équivalence correspondantes

Stratégie de test

- ✿ pour chaque classe d'équivalence, identifier les conditions aux limites
- ✿ Etablir un nouveau cas de test qui en recouvre une (et une seule), jusqu'à ce que toutes soient couvertes
 - ✿ Tester les bornes des classes d'équivalence, et juste à côté des bornes
 - ✿ Tester les bornes des entrées et des sorties

Heuristique

| Exigence | Classes valides | Classes invalides |
|---|---|--|
| intervalle de valeurs a..b | a b succ(a) pred(b) | pred(a) succ(b) |
| nombre limité de valeurs | minimum maximum minimum+1 maximum -1 | minimum-1 maximum+1 |
| ensemble de valeurs (fichier, liste) | premier élément deuxième élément avant-dernier élément dernier élément | avant premier élément après dernier élément |

Single Fault Assumption

- ✿ Dans le test aux limites on utilise l'hypothèse d'une seule erreur
- ✿ On cherche les valeurs de test à la limite d'une seule variable à la fois
- ✿ Raison:
Les défaillances sont presque jamais causées par l'occurrence simultanée de plusieurs fautes.

Test aux limites pour une variable

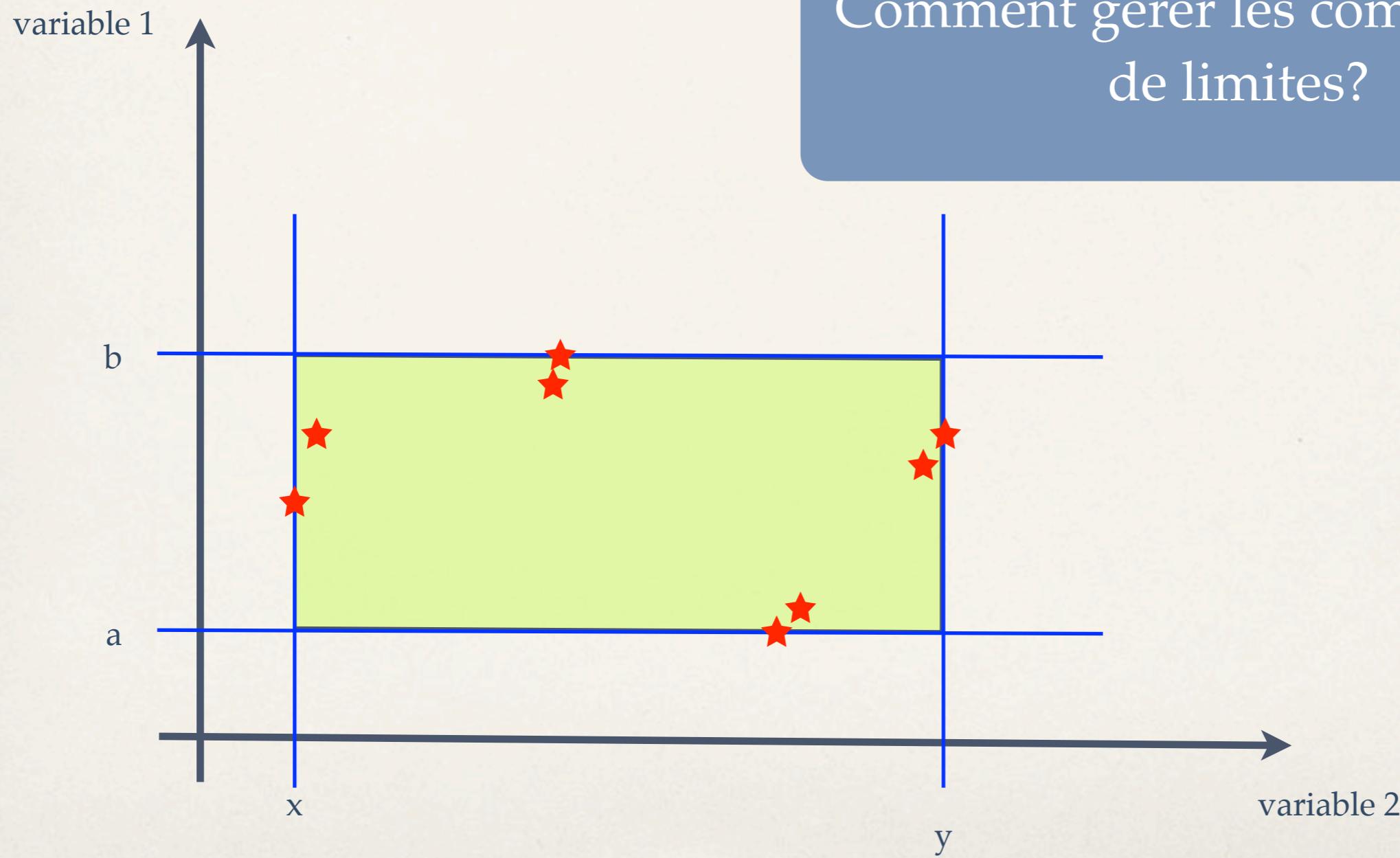


Test aux limites pour une variable

Comment gérer les combinaisons
de limites?

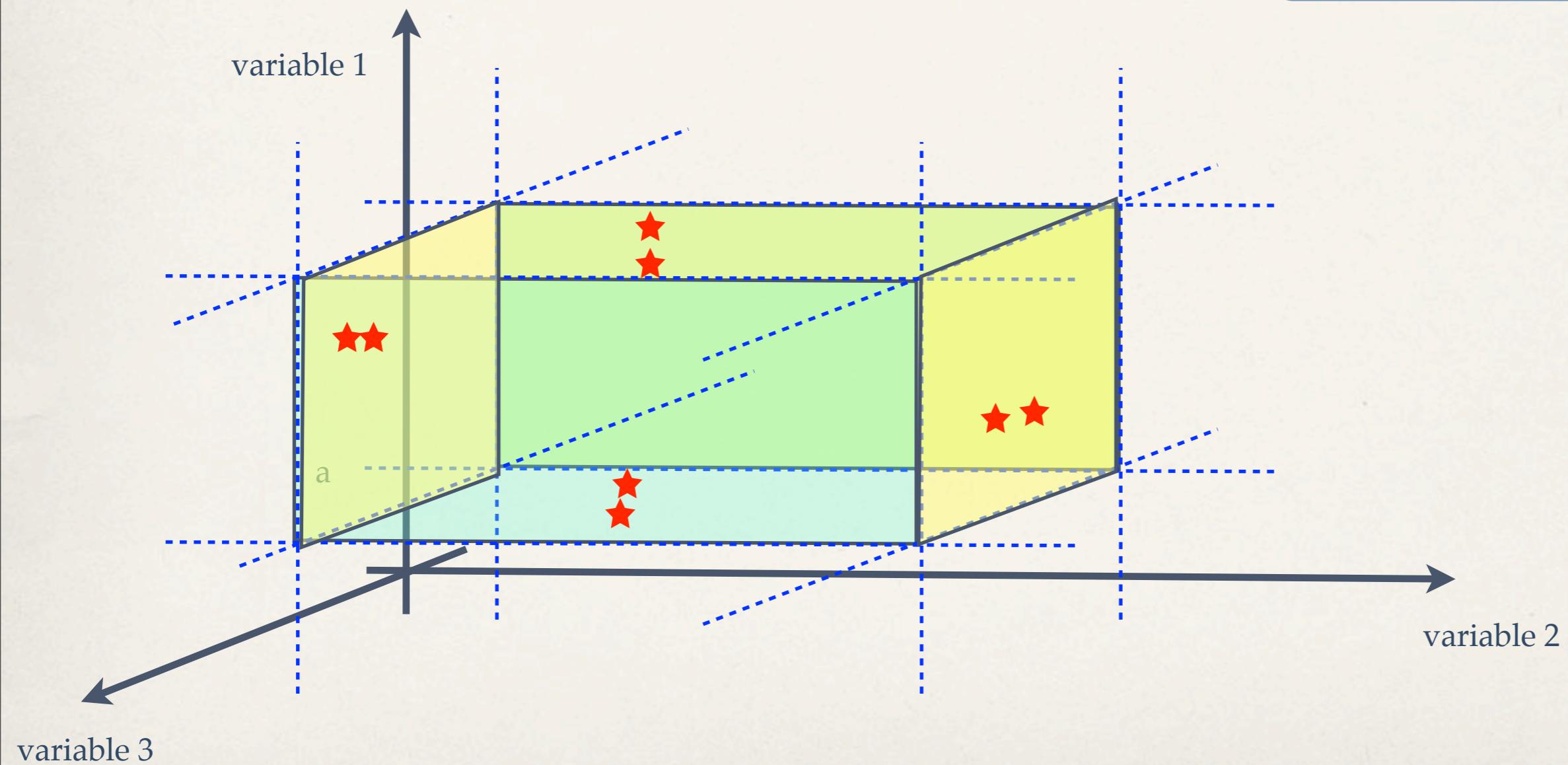


Test aux limites - deux variables



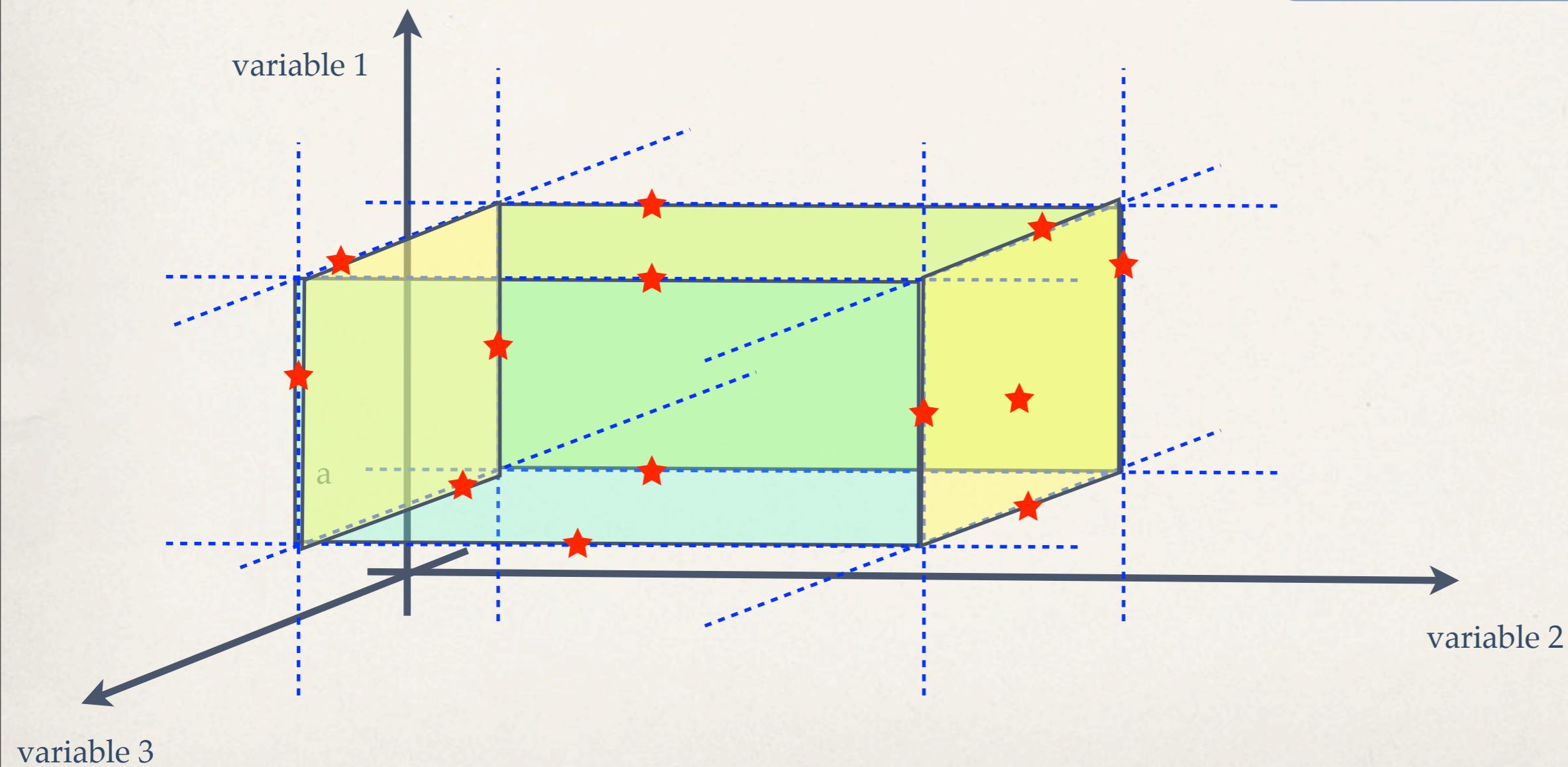
Test aux limites - trois variables

Comme ça?

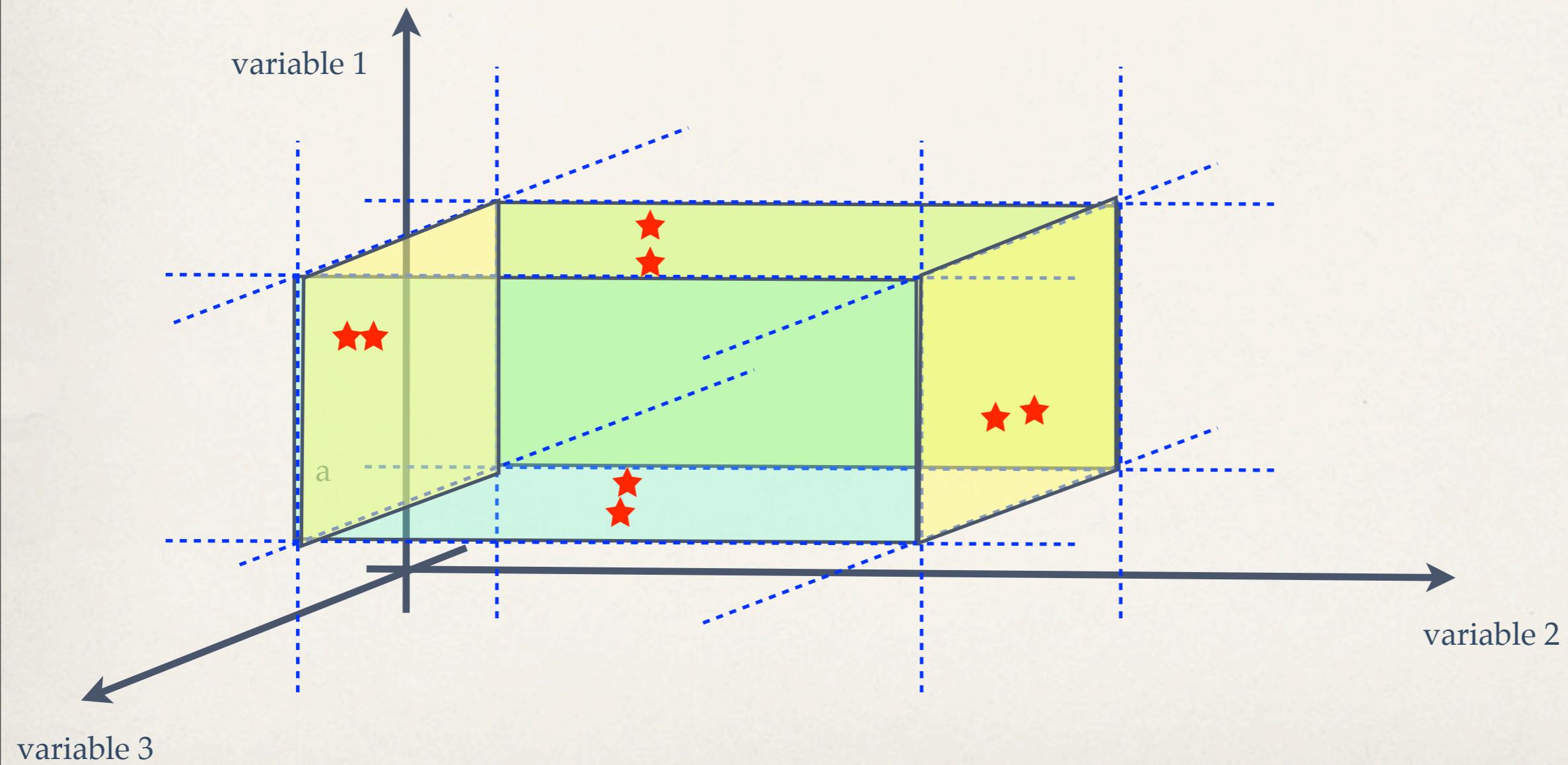


Test aux limites - trois variables

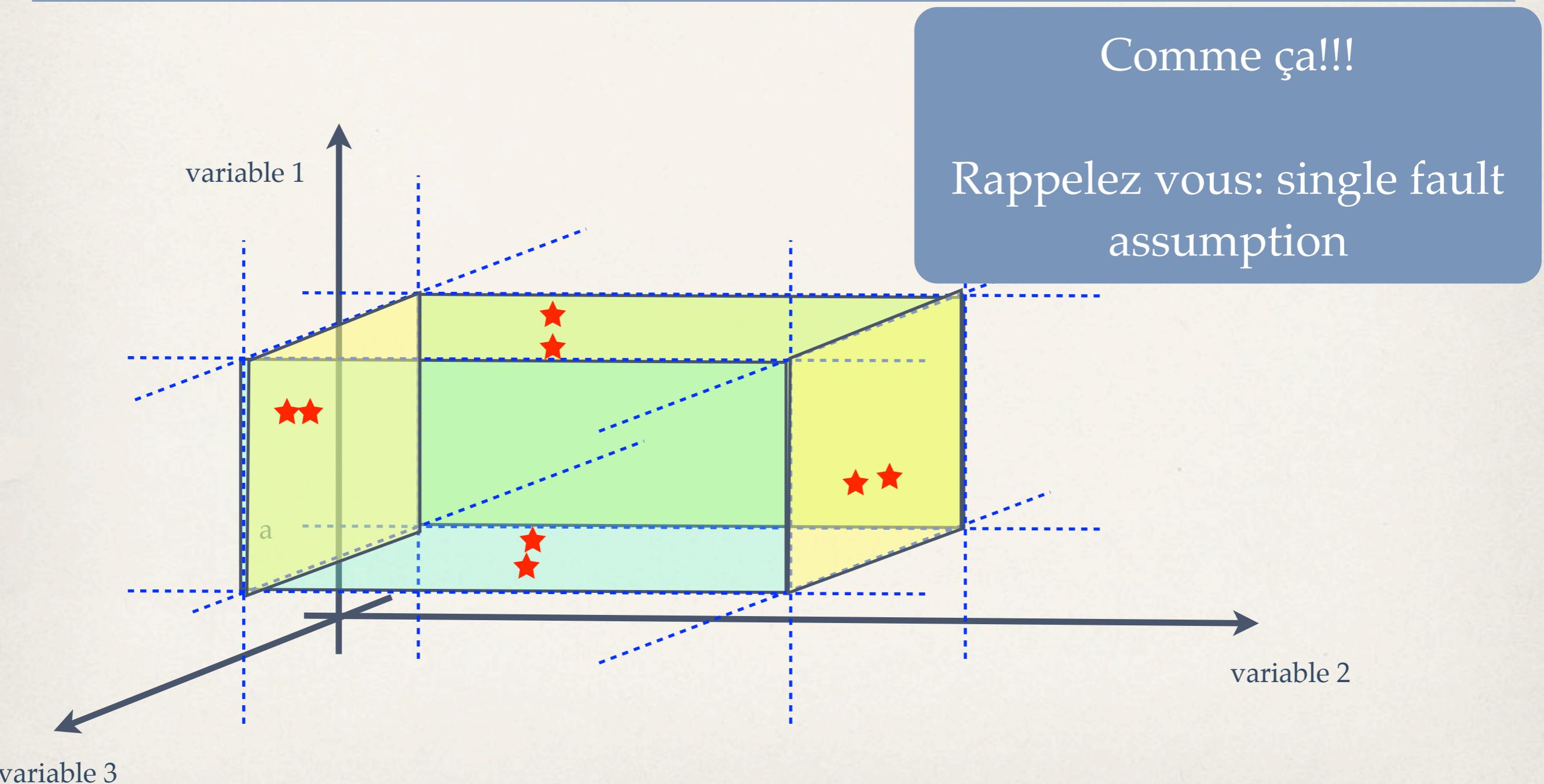
Ou, comme ça?



Test aux limites - trois variables



Test aux limites - trois variables



Test aux limites

- ✿ **Avantages:**
 - ✿ l'une des méthodes fonctionnelles les plus efficaces
 - ✿ couvre une large gamme d'erreurs
 - ✿ utilisable dans toute phase de test
 - ✿ utilisable en tests de charge, de performances, de précision

- ✿ **Inconvénients**
 - ✿ difficulté de formaliser la notion de limite
 - ✿ n'exploré pas les combinaisons d'exigences sur les entrées

Plan du cours

- * Partition des domaines des entrées
- * Test aux limites
- * Test de robustesse
- * Graphes de causes à effets

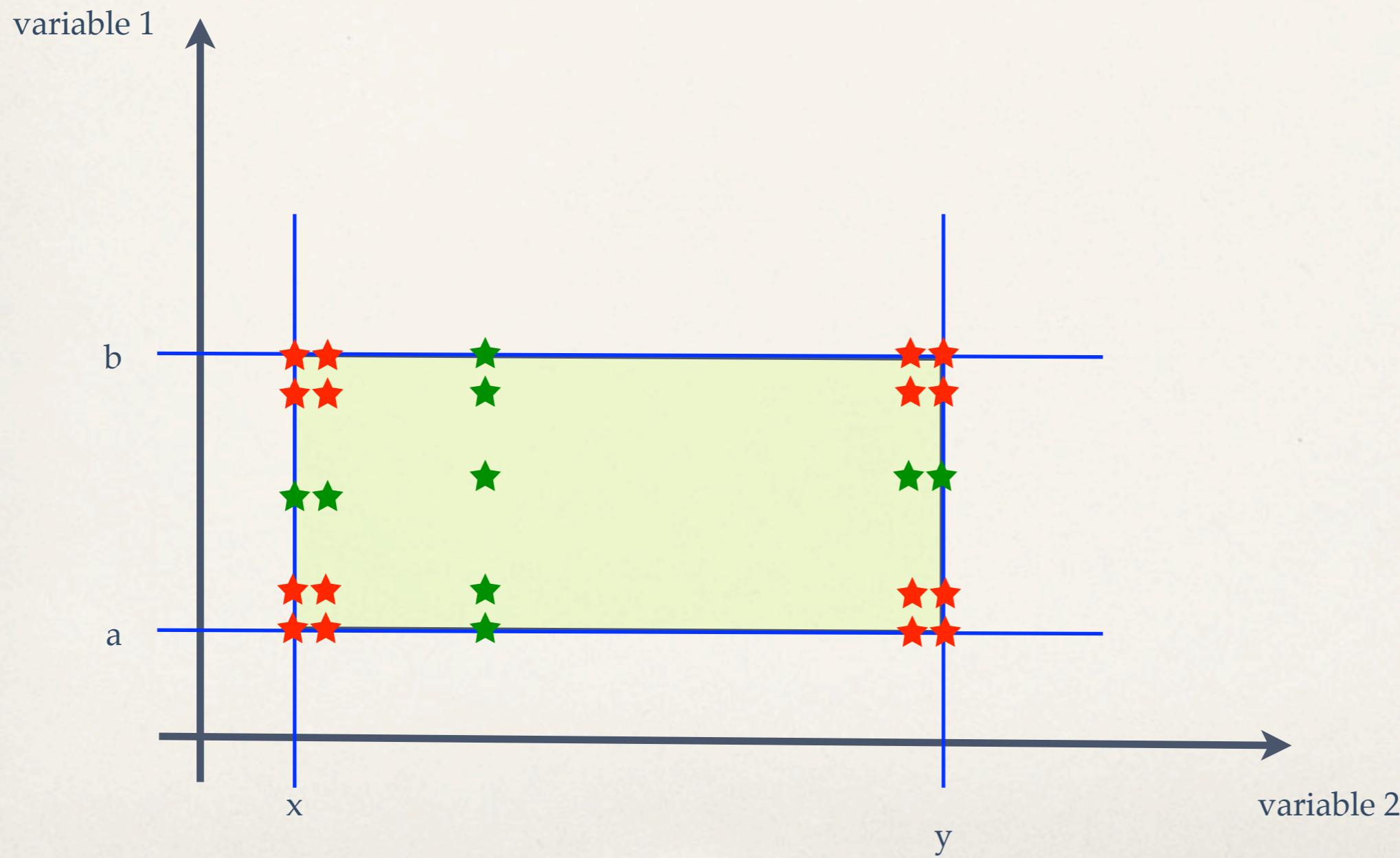
Test de robustesse

- ❖ Extension du teste aux limites
- ❖ Examine ce qui se passe si les valeurs sont légèrement dépassées (test négatif)
- ❖ Pourquoi? Pour tester la gestion des exception, pour s'assurer que les exigences sont bien prises en compte

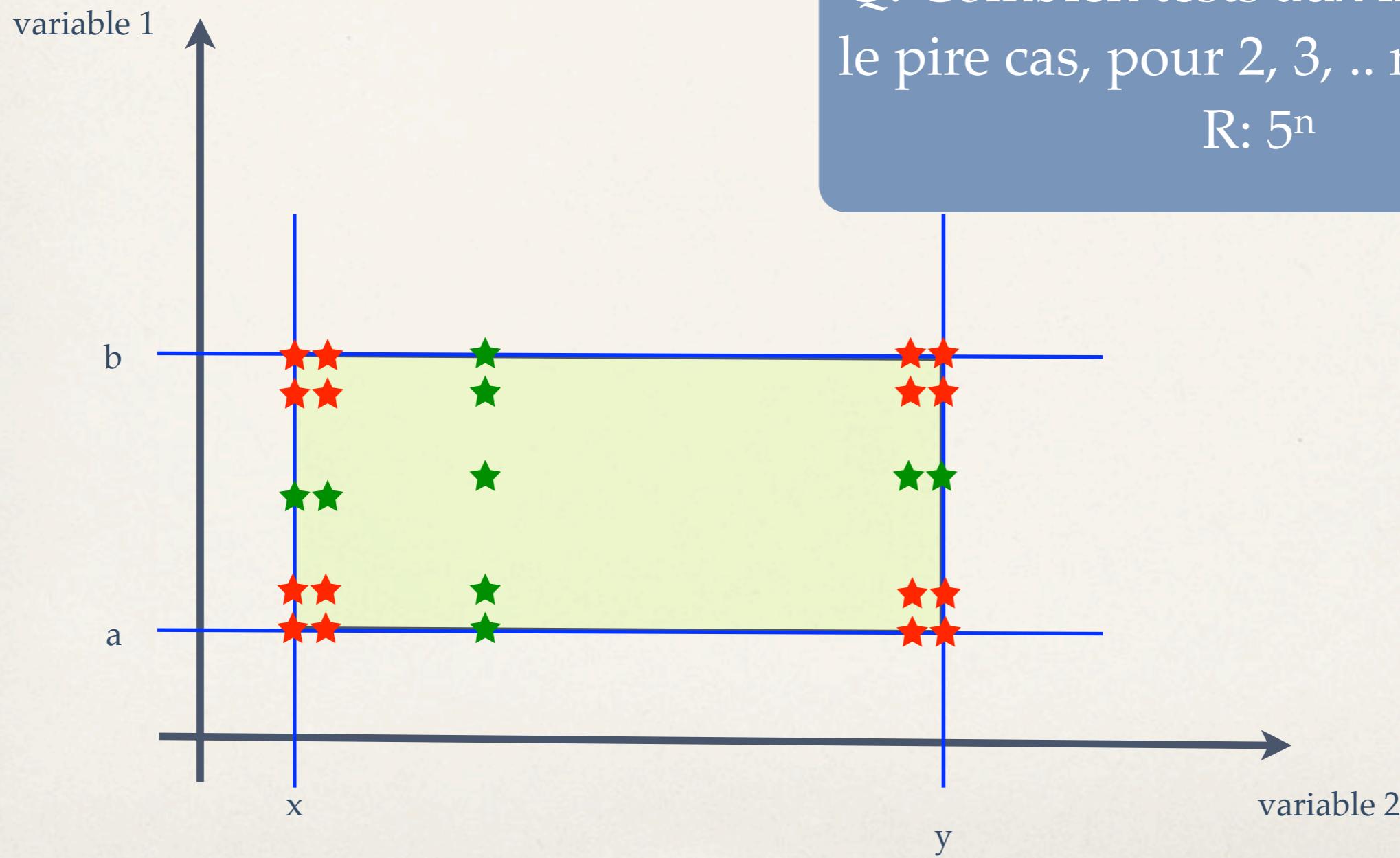
Test au pire-cas (worst case testing)

- Le test aux limites utilise comme hypothèse la “single fault assumption”
- Le test au pire cas s’intéresse à ce qui se passe si plus d’une variable a une valeur extrême
- Utile si les variables ont des interactions complexes ou si les défaillances ont des gros impacts

Test au pire cas



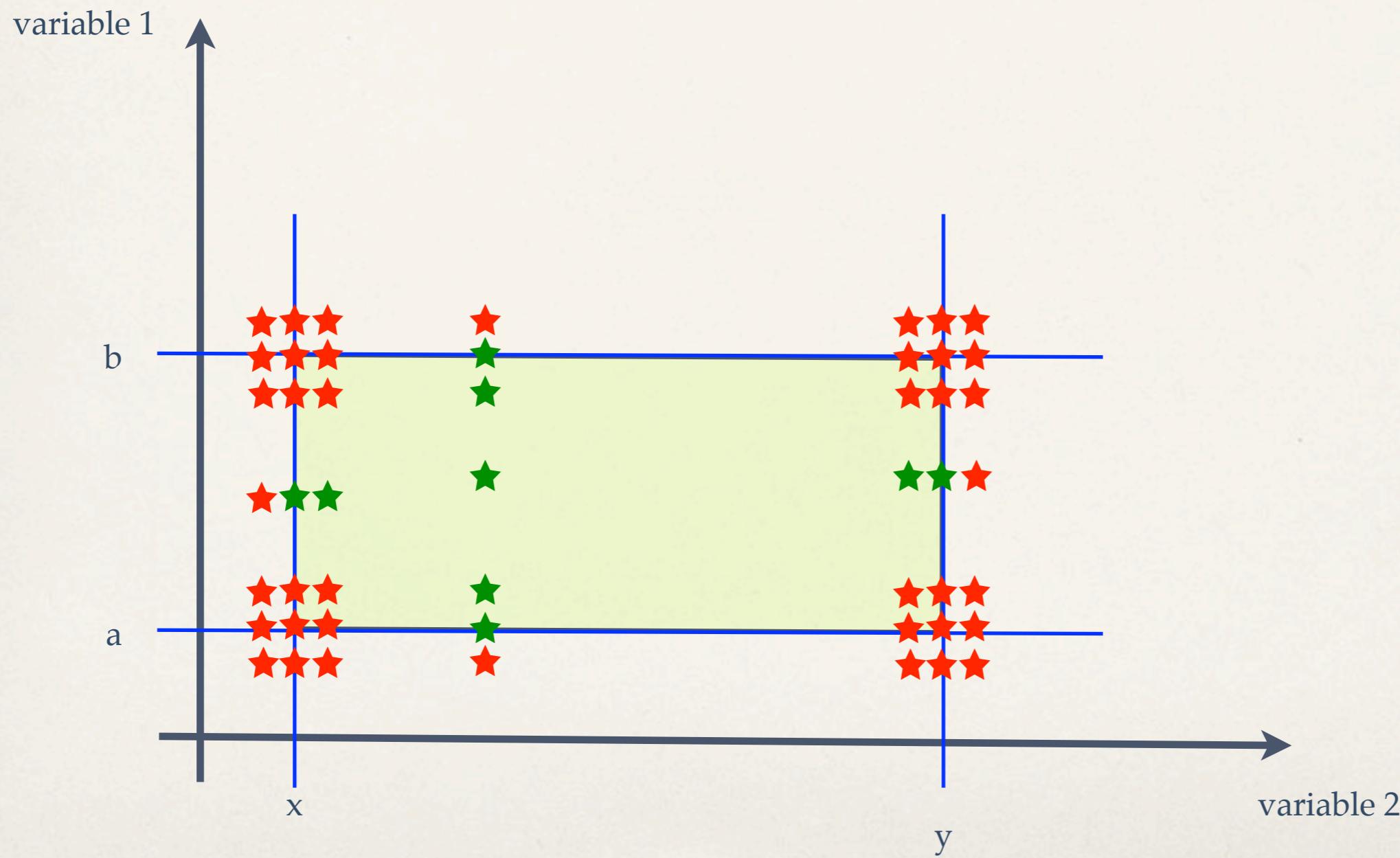
Test au pire cas



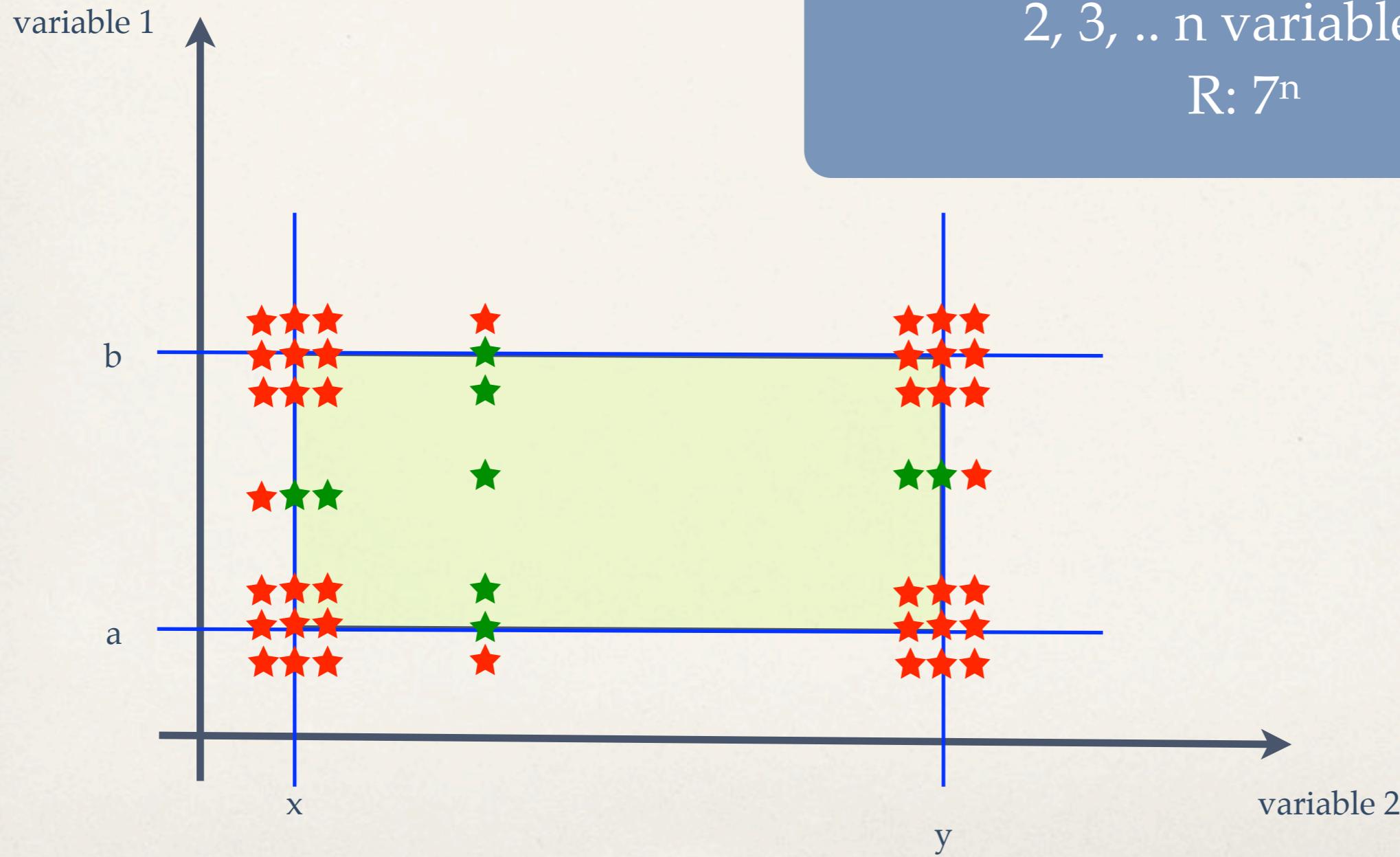
Q: Combien tests aux limites dans le pire cas, pour 2, 3, .. n variables?

R: 5^n

Test au pire cas robuste (paranoïaque)



Test au pire cas robuste (paranoïaque)



Q: Combien tests aux limites, pour
2, 3, .. n variables?

$$R: 7^n$$

Remarques

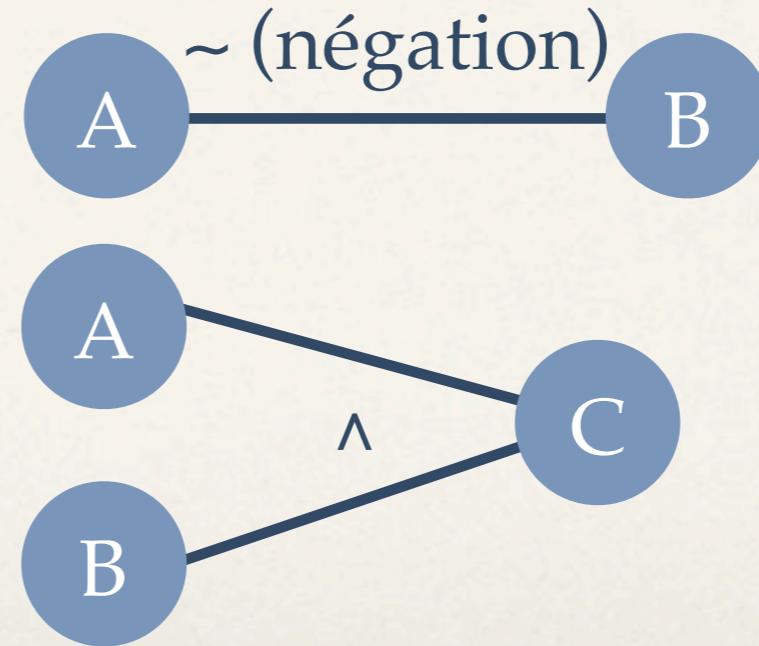
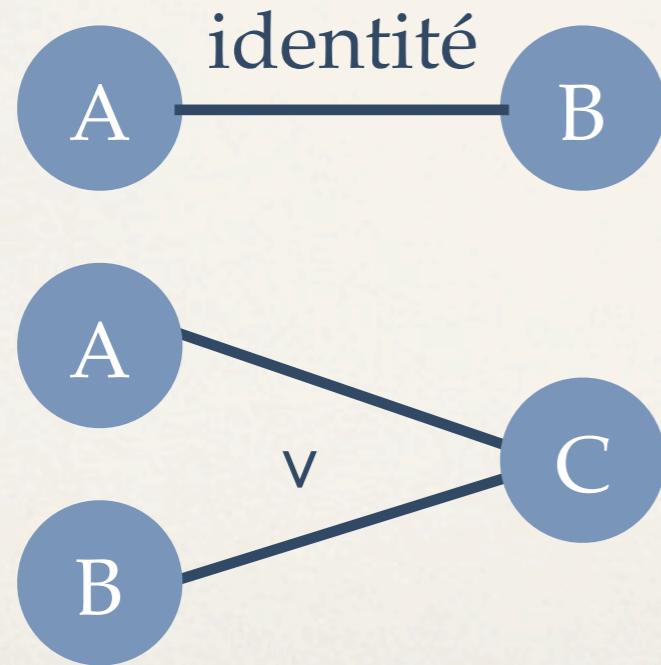
- ✿ Q: A-t-on besoin de test négatifs?
- ✿ R: Oui, ça permet de s'assurer que la spécification a été prise en compte de manière précise et de vérifier la gestion des exceptions.
- ✿ Q: Est-ce qu'il nous faut plus de test négatifs ou positifs?
- ✿ R: En général il y a plus de tests positifs. Les tests négatifs prolifèrent s'il y a un besoin important de fiabilité ou si l'environnement est perçu comme hostile.

Plan du cours

- * Partition des domaines des entrées
- * Test aux limites
- * Test de robustesse
- * Graphes de causes à effets

Graphe de cause à effet

- * graphes combinatoires logiques (Myers 79)
- * relient les effets (sorties) aux causes (entrées)
 - * nœuds : causes, effets ou intermédiaires
 - * arcs : connecteurs logiques



Démarche

- ✿ identifier les causes
- ✿ identifier les effets
- ✿ construire le graphe de causes à effets
- ✿ indiquer les contraintes logiques et annoter le graphe
- ✿ en déduire la table de décision
- ✿ simplifier la table

Exemple 1- Traitement d'un fichier de mouvements

- ✿ le 1er caractère doit être un D ou un C
- ✿ le 2ème caractère doit être un chiffre
- ✿ le mouvement est alors effectué
- ✿ si le 1er caractère est erroné, le message M1 doit être affiché
- ✿ si le 2ème caractère est erroné, le message M2 doit être affiché

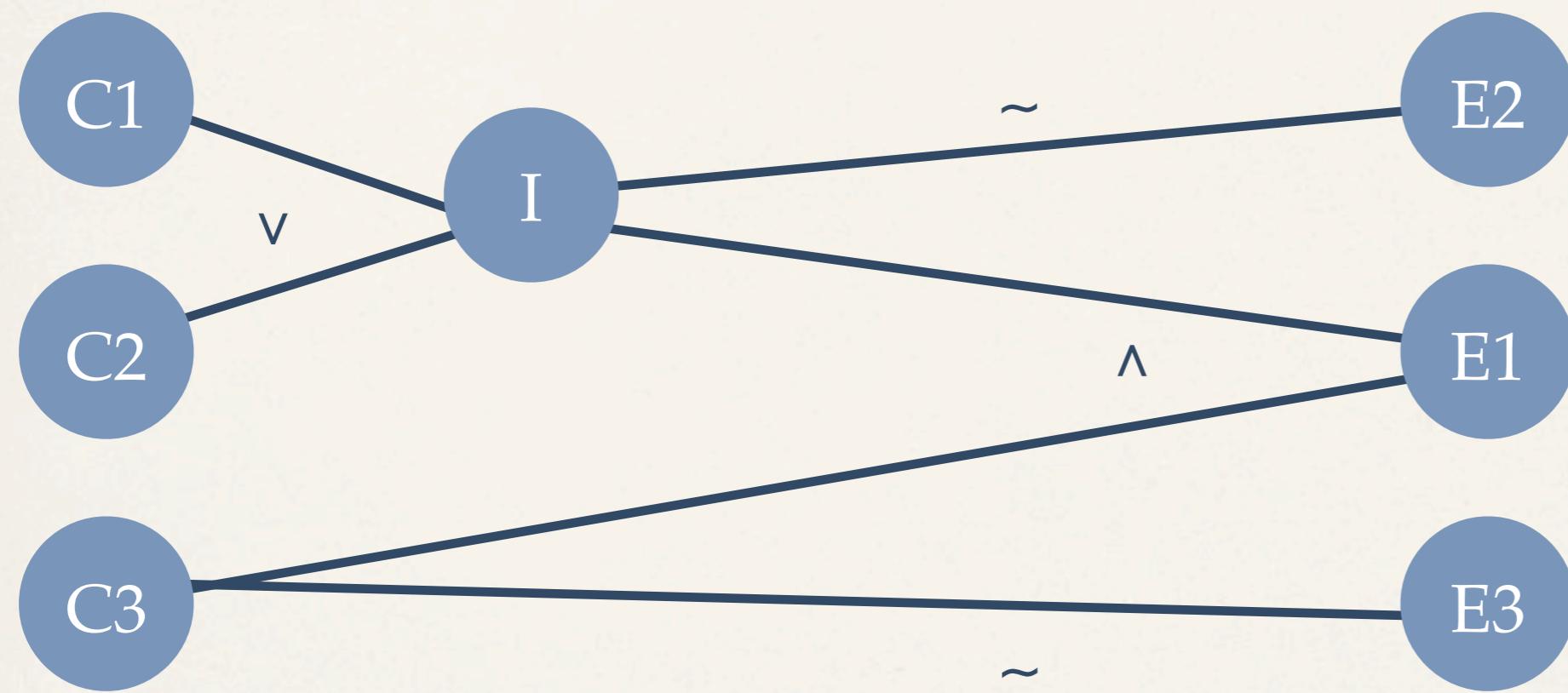
Exemple 1

Identifier causes et effets

- ❖ Causes
 - ✿ C1 : le 1er caractère est un D
 - ✿ C2 : le 1er caractère est un C
 - ✿ C3 : le 2ème caractère est un chiffre

- ❖ Effets
 - ✿ E1 : le mouvement est effectué
 - ✿ E2 : le message M1 est affiché
 - ✿ E3 : le message M2 est affiché

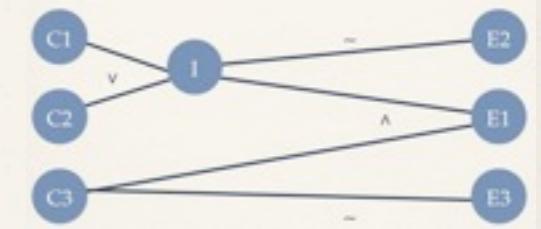
Exemple 1 - construire le graphe



Exemple 1

Déduire la table des décisions

| No | Causes | | | Effets | | |
|----|--------|----|----|------------|----|----|
| | C1 | C2 | C3 | E1 | E2 | E3 |
| 1 | F | F | F | F | F | V |
| 2 | F | F | V | F | V | F |
| 3 | F | V | F | F | F | F |
| 4 | F | V | V | V | F | F |
| 5 | V | F | F | F | F | V |
| 6 | V | F | V | V | F | F |
| | V | V | | IMPOSSIBLE | | |



Exemple 1

Simplification de la table

| No | Causes | | | Effets | | |
|-----|--------|----|----|--------|----|----|
| | C1 | C2 | C3 | E1 | E2 | E3 |
| 1.2 | F | F | - | F | V | - |
| 3.5 | - | - | F | F | - | V |
| 4 | F | V | V | V | F | F |
| 6 | V | F | V | V | F | F |

Graphe de cause à effet

- ❖ **Avantages:**
 - ❖ ceux du teste fonctionnel +
 - ❖ explore les combinaisons d'exigences sur les entrées
 - ❖ complet, précis, peut être en partie automatisé
 - ❖ relecture des spécifications et analyse minutieuse
 - ❖ oracle : manuel, au moins
- ❖ **Inconvénients**
 - ❖ fondé sur la compréhension des fonctionnalité
 - ❖ complexité du graphe
 - ❖ exactitude du graphe
 - ❖ mise-à-jour difficile du graphe
 - ❖ lourd

Comment combiner les différentes stratégies de test?

- ✿ Pour limiter le nombre de tests: test incrémental et prioritisation
 - ✿ ordonner les méthodes de test en fonction du nombre de tests générés
 - ✿ commencer avec celles qui génèrent moins de test
 - ✿ choisir les données de test et lancer les tests
 - ✿ pour la prochaine famille rajouter **uniquement** les tests manquants
 - ✿ itérer
- ✿ Ordre typique
 - ✿ Couverture des exigences
 - ✿ Couverture des interfaces
 - ✿ Couverture du code (test structurel)
- ✿ Le test est une activité créative
 - ✿ pas de critère parfait
 - ✿ combiner les critères pour plus d'efficacité

Stratégie typique pour le test dynamique

