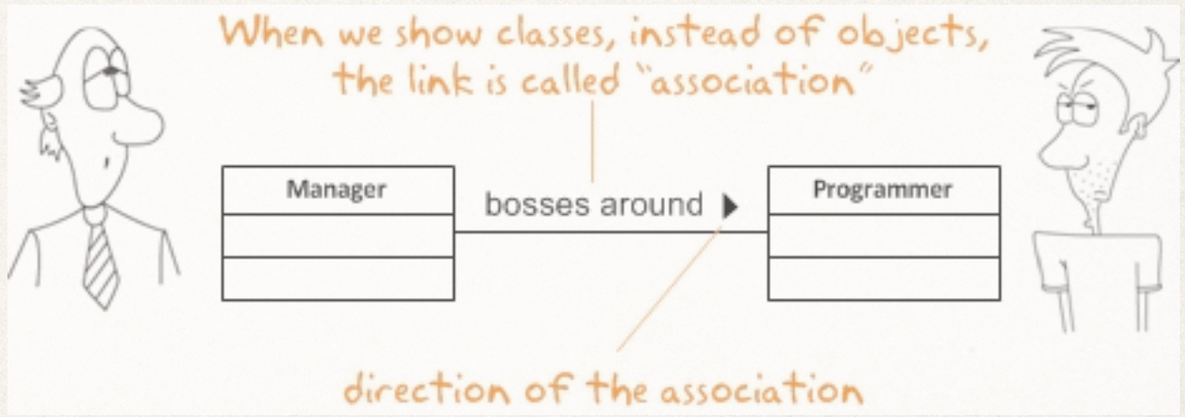
MCO -

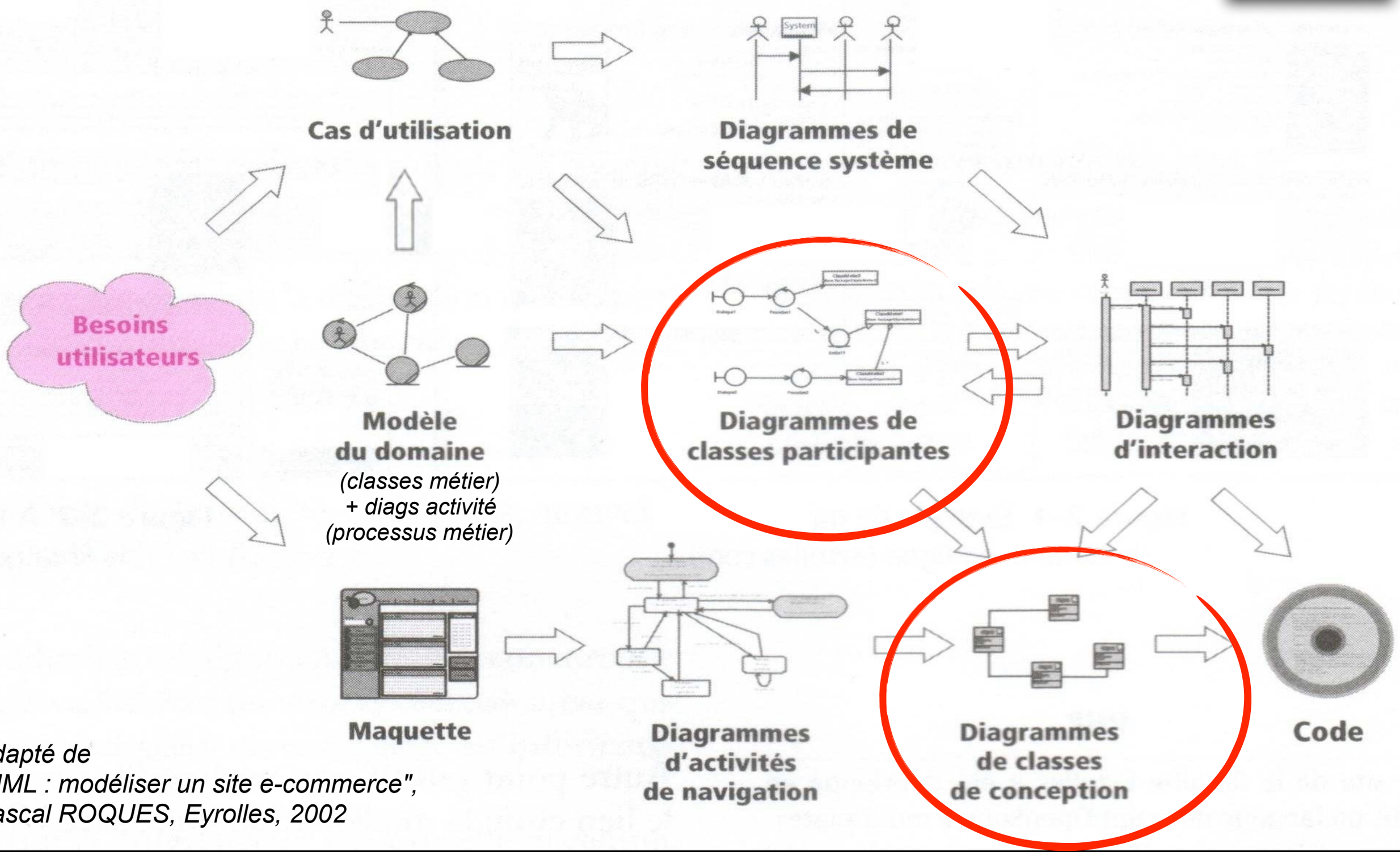
Diagramme de classes

Ileana Ober
Université Paul Sabatier
IRIT
http://www.irit.fr/~Ileana.Ober/



Année Universitaire 2017-2018

Une démarche



Adapté de
"UML : modéliser un site e-commerce",
Pascal ROQUES, Eyrolles, 2002

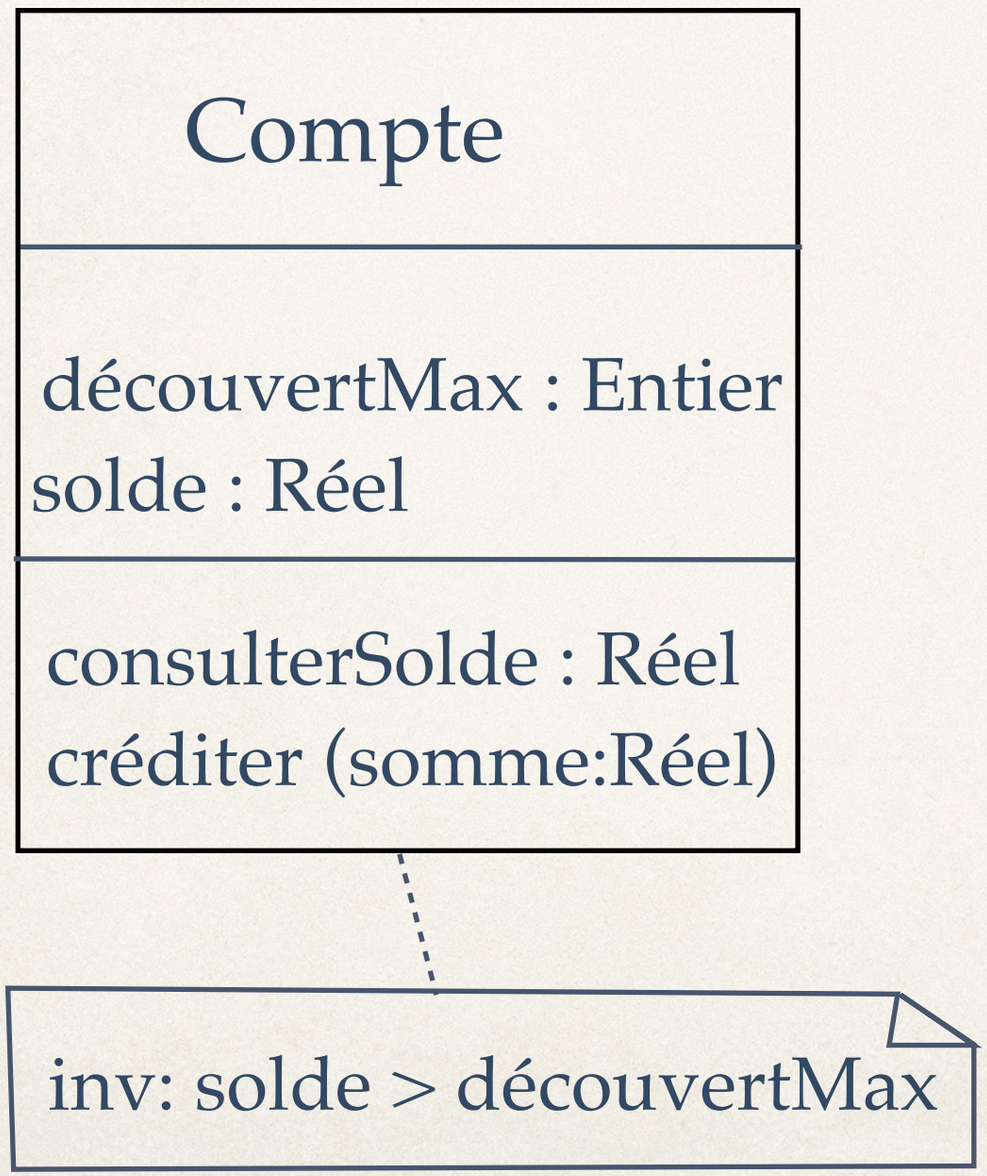
Plan

- Notions de base
- Conformité
  diagramme de classes - système modélisé,
  diagramme d'instances - diagramme de classes
- Concepts avancés
- Usage des diagrammes de classes

Notions de base

- Classe
- Objet
- Association
- Héritage
- Contrainte

Classe UML



Nom de la classe

Attributs

nom : type

Opérations

nom

liste paramètres

(avec leur types)

type de retour

Notations pour la classe

Compte

Compte
découvertMax : Entier solde : Réel

Compte
consulterSolde : Réel créditer (somme:Réel)

- noms de classes (et types) en majuscule
- nom d'attributs et opérations en minuscule

Objet UML

comptePaul:Compte

:Compte

comptePaul

comptePaul:Compte
découvertMax = 1000 solde =456

- noms d'objets commencent par une minuscule
- nom soulignés = noms d'instance

Classe et Objet

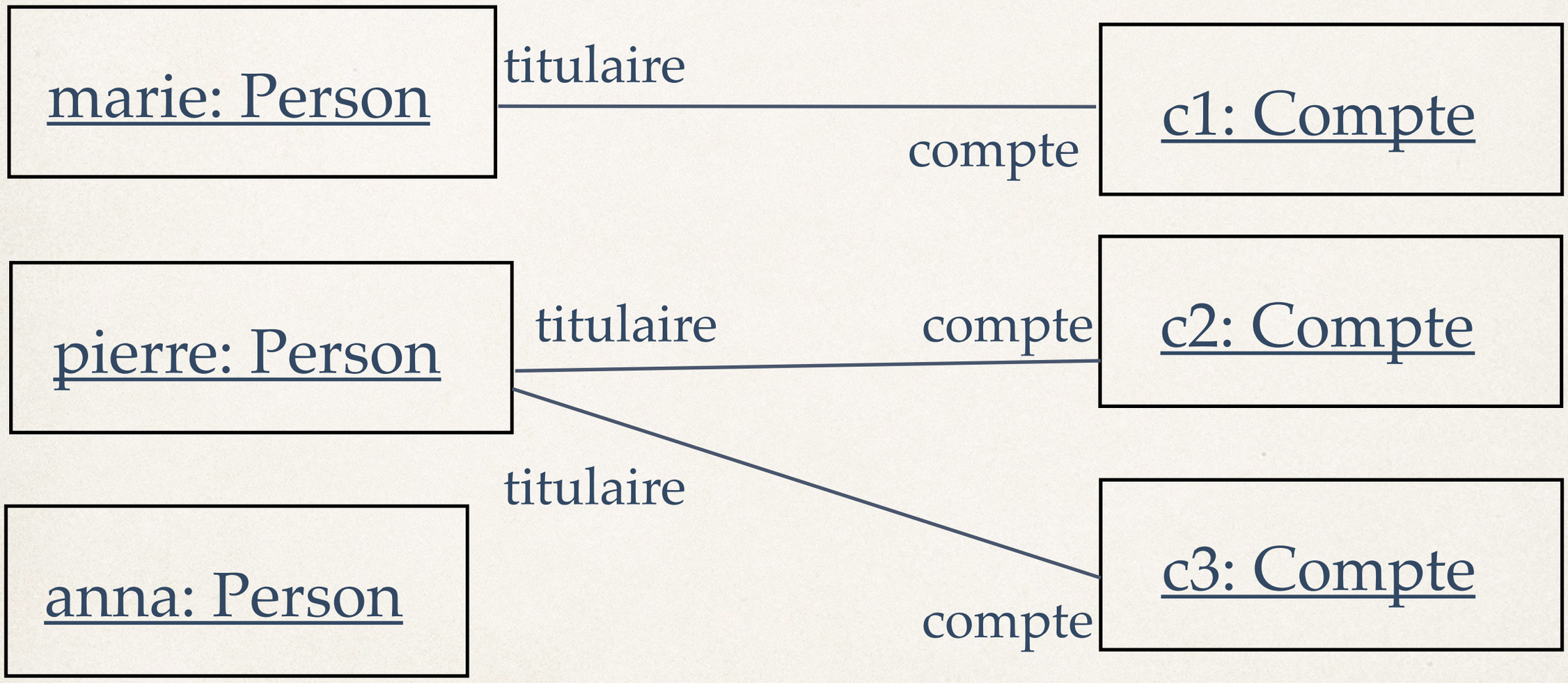
- Une classe spécifie la structure et le comportement d'un ensemble d'objets
- La structure d'une classe est constante
- Une classe existe pendant toute l'exécution
- Les objets sont crées ou détruits pendant l'exécution
- La valeur des attributs des objets varie

Compte
découvertMax : Entier solde : Réel
consulterSolde : Réel créditer (somme:Réel)

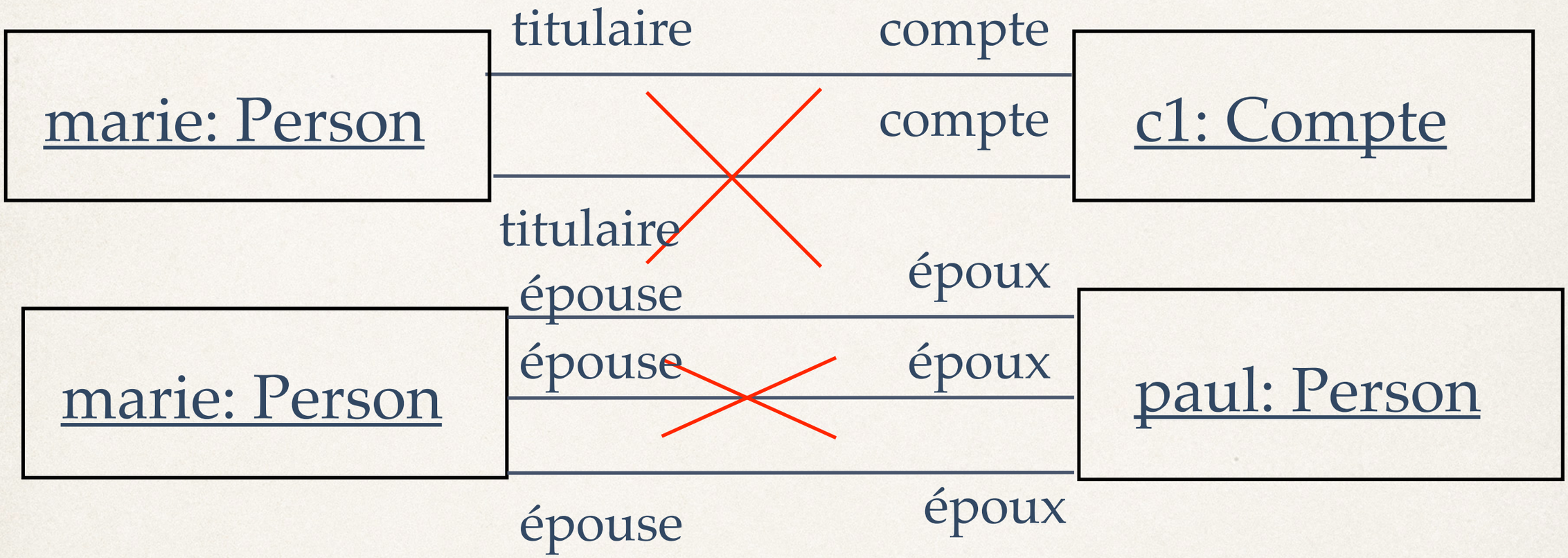
comptePaul:Compte
découvertMax = 1000 solde =456

compteMarie:Compte
découvertMax = 1500 solde =345

Lien - connexion entre objets



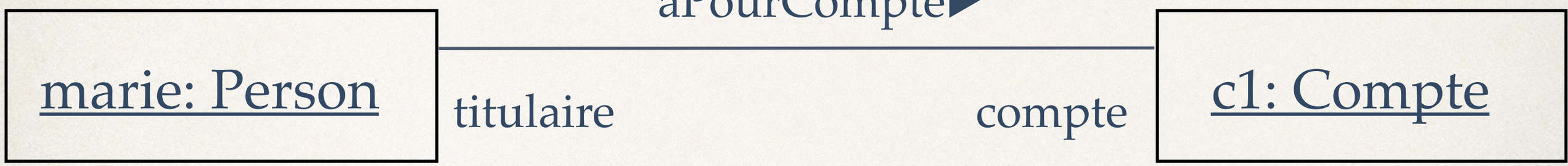
Liens



Au maximum un lien d'un type donné entre les mêmes deux objets

(contrainte qui pourra être relachée plus tard)

Rôles



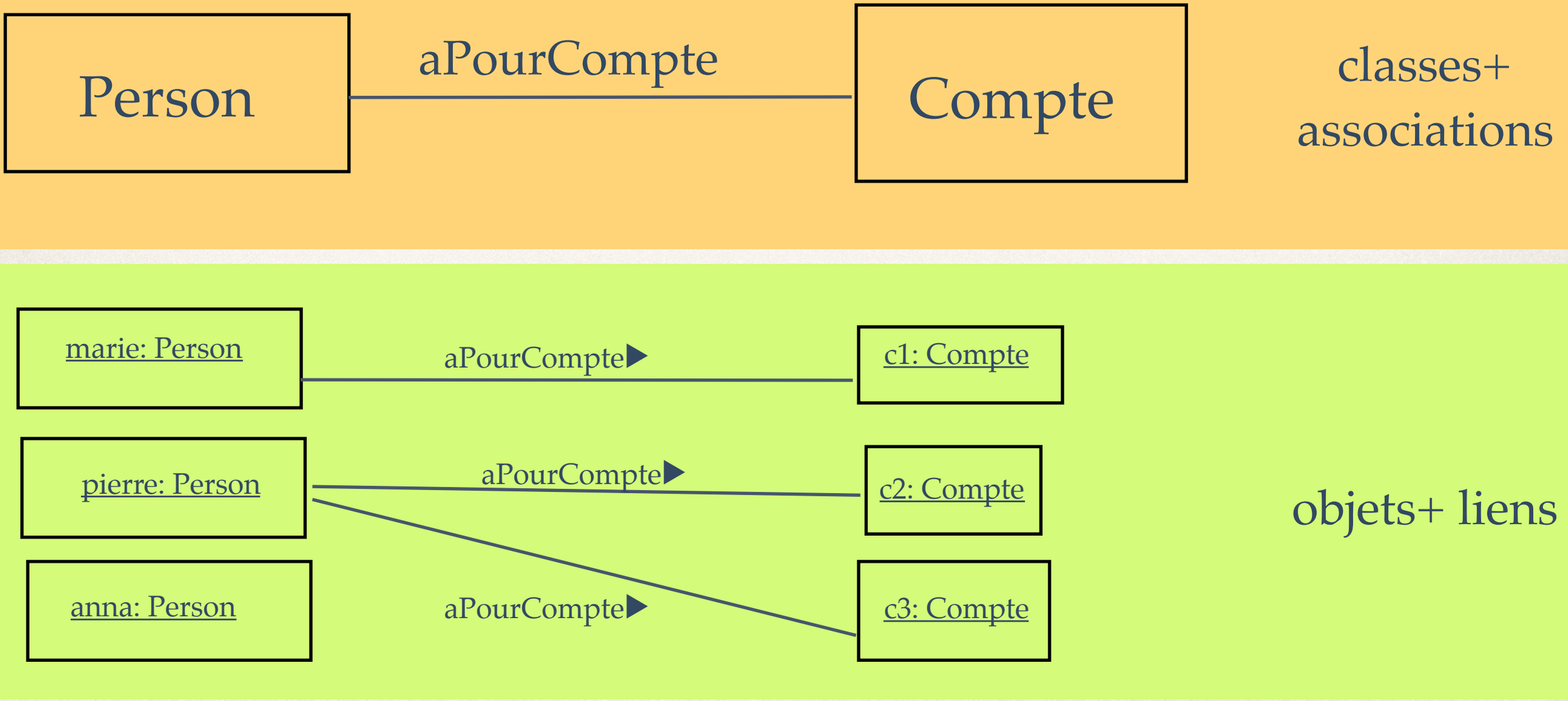
- Chacun des deux objets joue un rôle dans le lien
- Le nom de ce rôle apparaît sur le lien du côté de l'objet

  marie est **titulaire** du compte **c1**
  c1 est le **compte** de **marie**

- Si le nom de rôle est omis on utilise le nom de la classe

Associations

- Une association regroupe un ensemble de liens



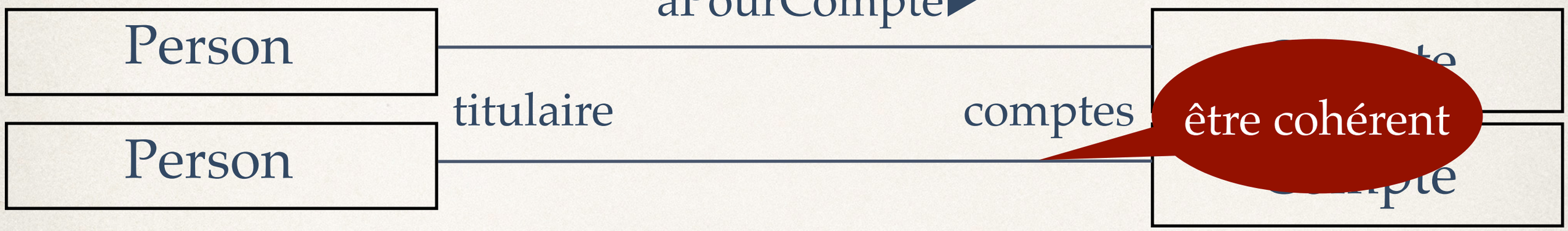
Association - Lien

- Un lien relie 2 objets
- Une association relie 2 classes

- Un lien est une instance d'association
- Une association donne le type d'un ensemble de liens

- Un lien peut être crée détruit pendant l'exécution
- Une association ne varie pas à l'exécution

Nom des associations



Rôles et navigation

marie.comptes = {c1}

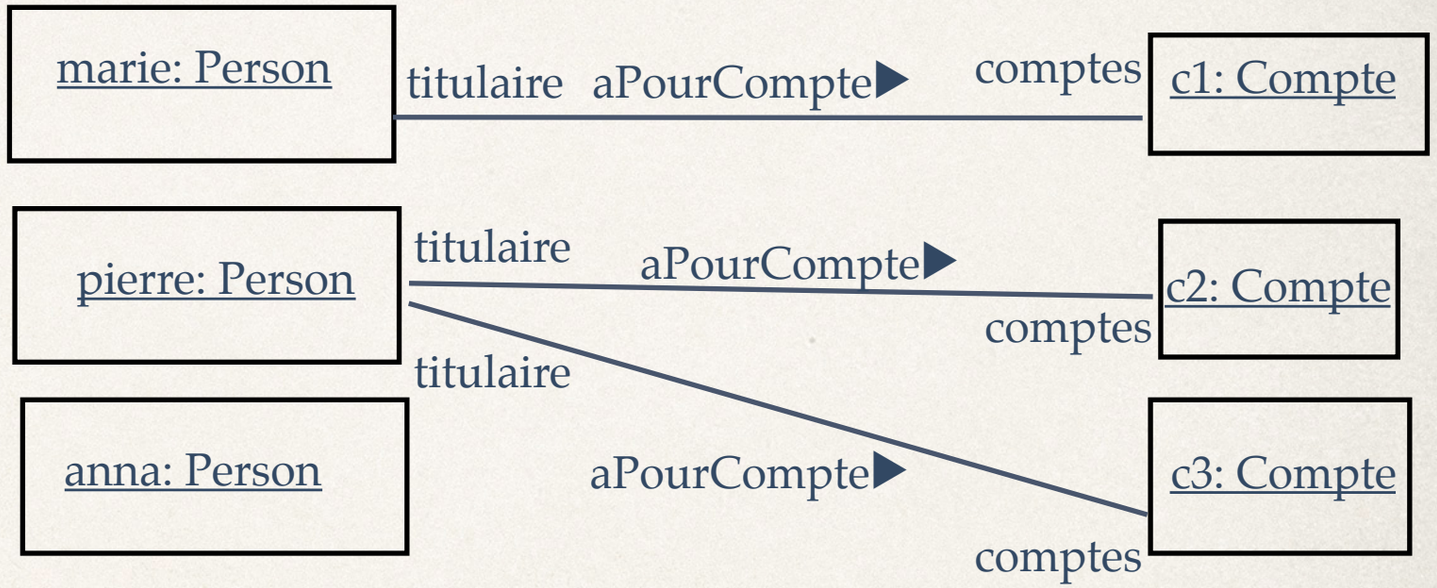
pierre.comptes = {c2, c3}

anna.comptes = ∅

c1.titulaire = {marie}

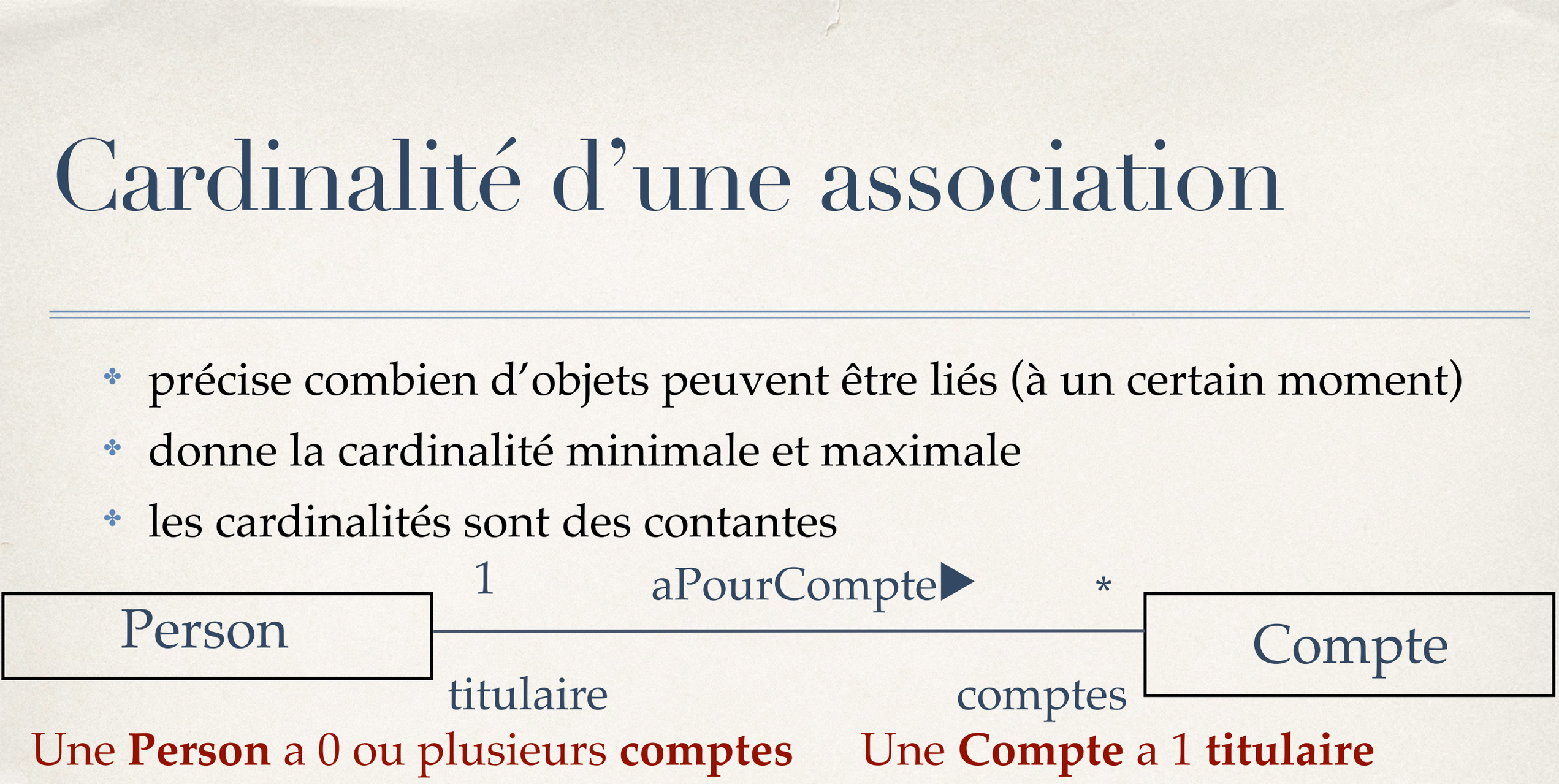
c2.titulaire = {pierre}

c3.titulaire = {anna}

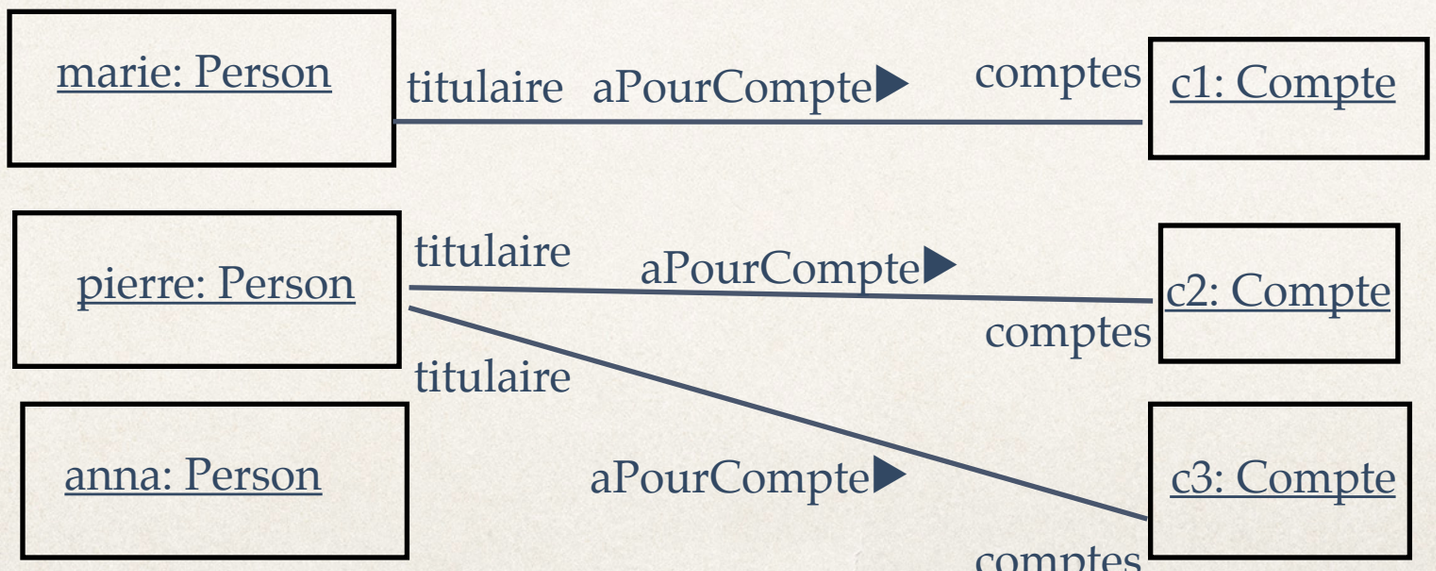


Cardinalité d'une association

- précise combien d'objets peuvent être liés (à un certain moment)
- donne la cardinalité minimale et maximale
- les cardinalités sont des contantes

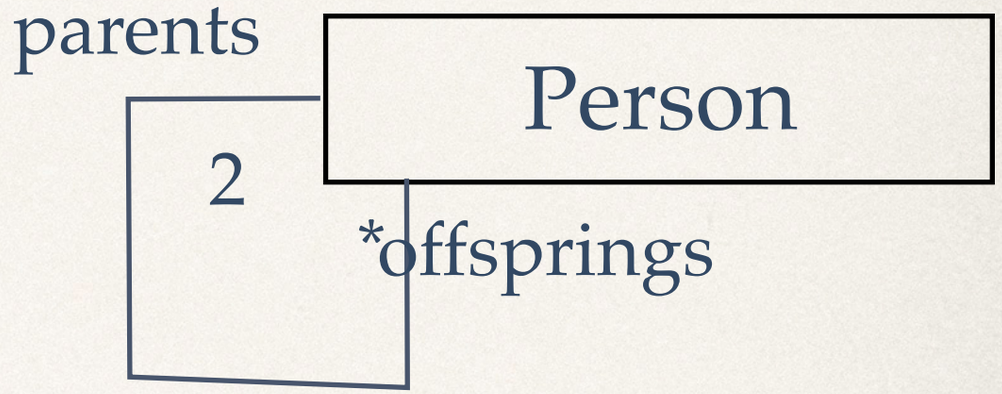


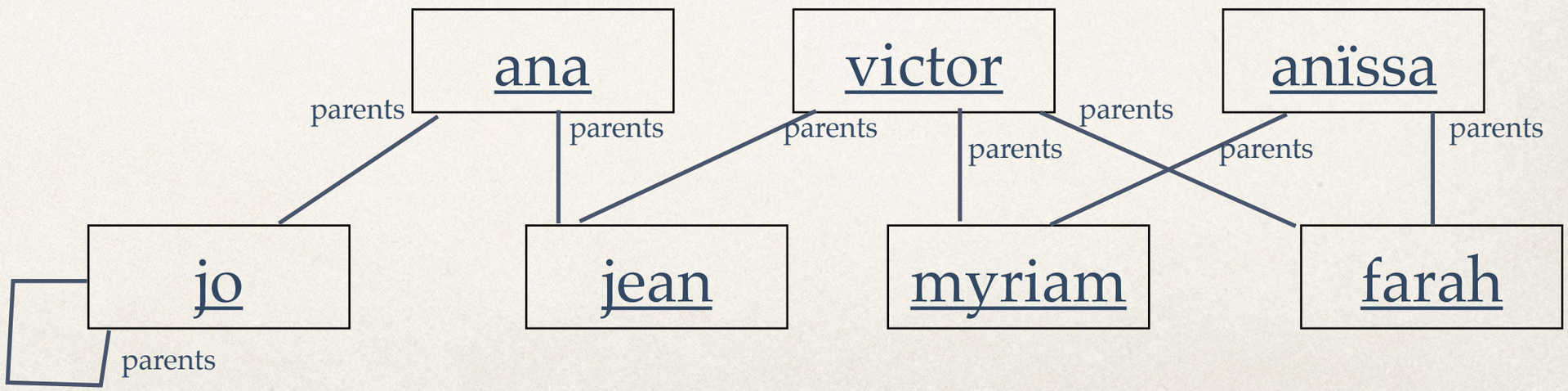
Une **Person** a 0 ou plusieurs **comptes** Une **Compte** a 1 **titulaire**



Associations réflexives

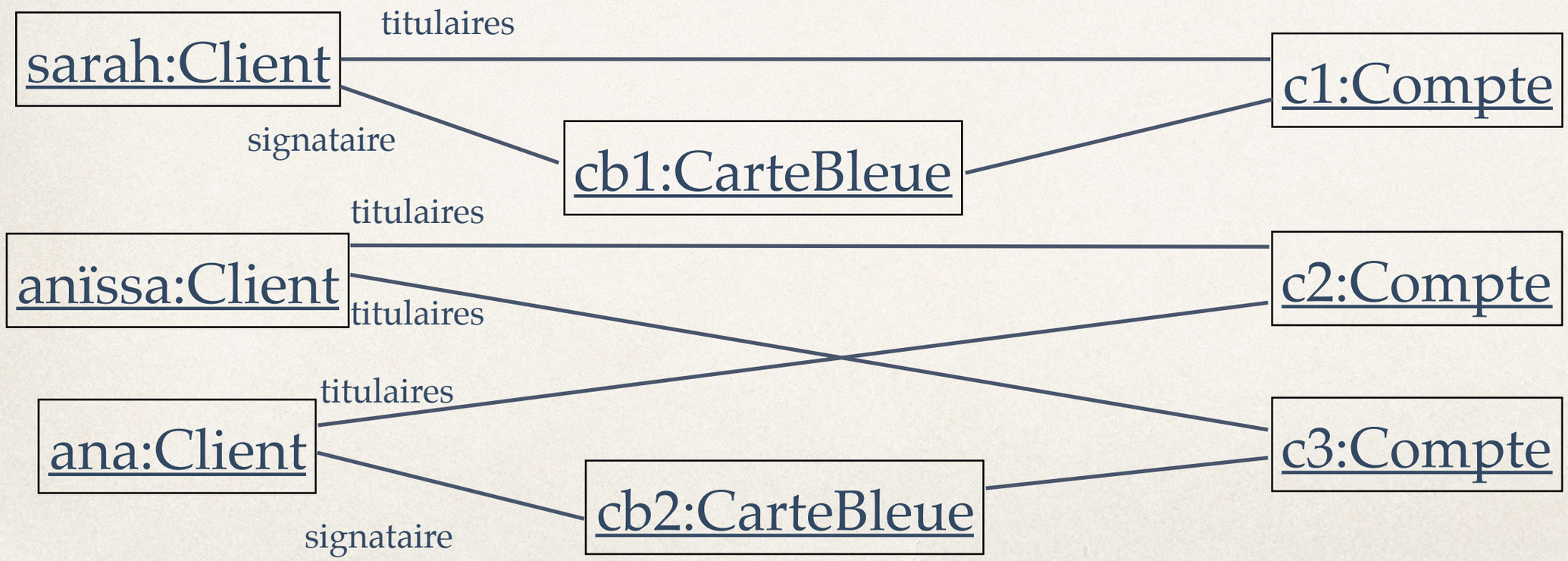
- Rien n'empêche les associations réflexives
- Une association réflexive n'implique pas un lien réflexif





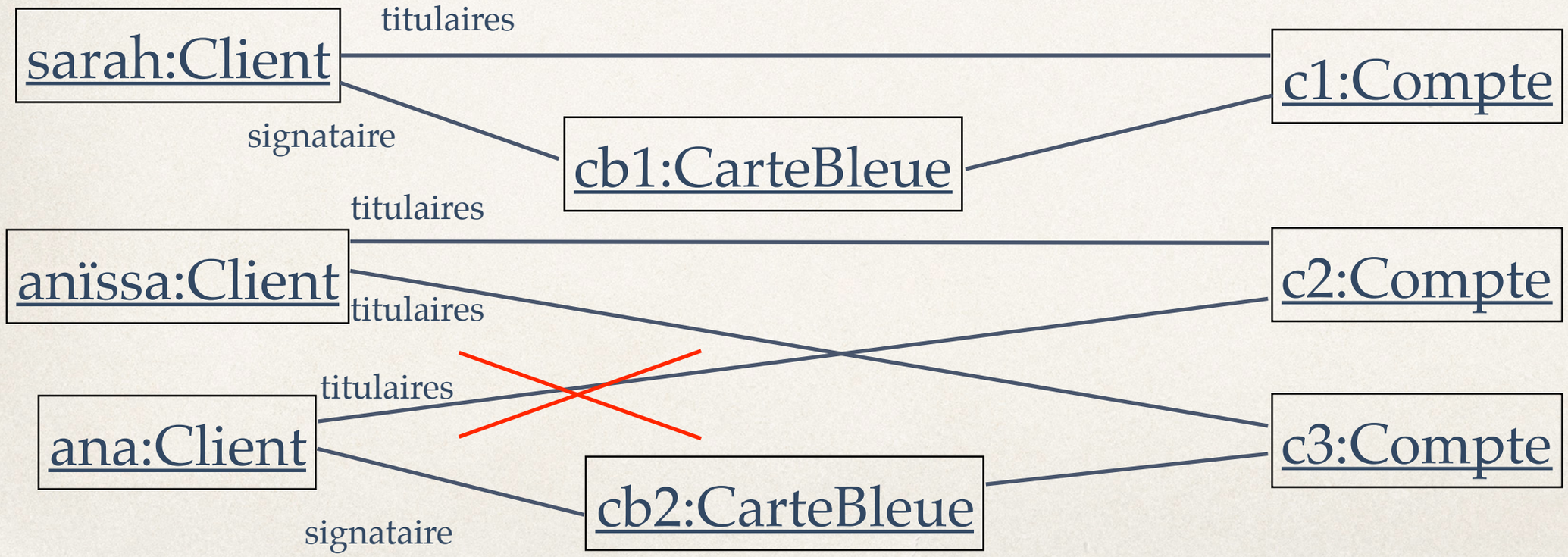
Constraints entre des associations

Attention aux Constraints évidentes qui peuvent mener à des mauvais modèles



Constraints entre des associations

Le signataire de la carte bleue associée à un compte est l'un des titulaires de ce compte



Constraints entre des associations

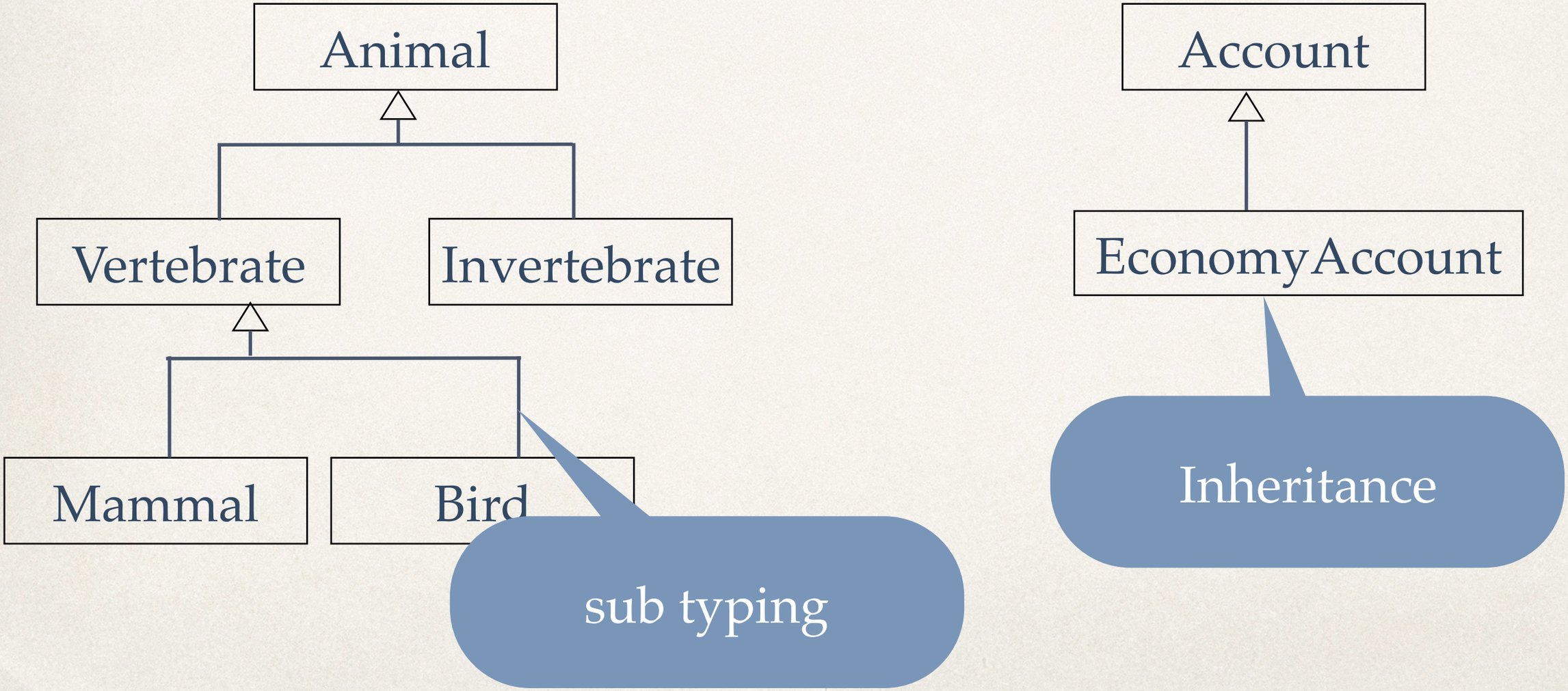
Le signataire de la carte bleue associée à un compte est l'un des titulaires de ce compte

- Les Constraints peuvent s'exprimer de manière précise avec des langages de contrainte (OCL - Object Constraint Language)

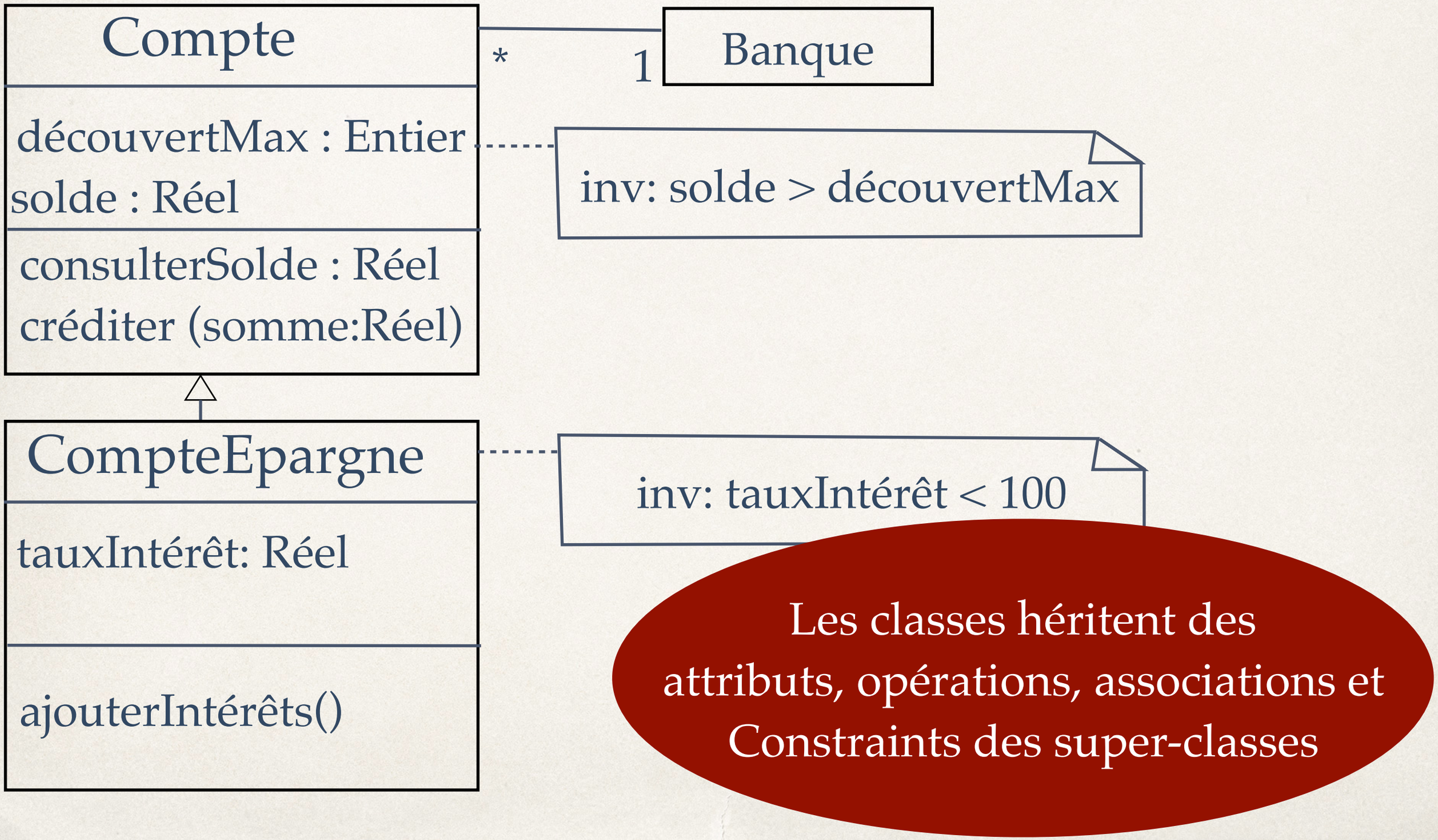
```
context CarteBleue
inv: self.Compte.titulaires -> includes self.signataire
```

Specialization

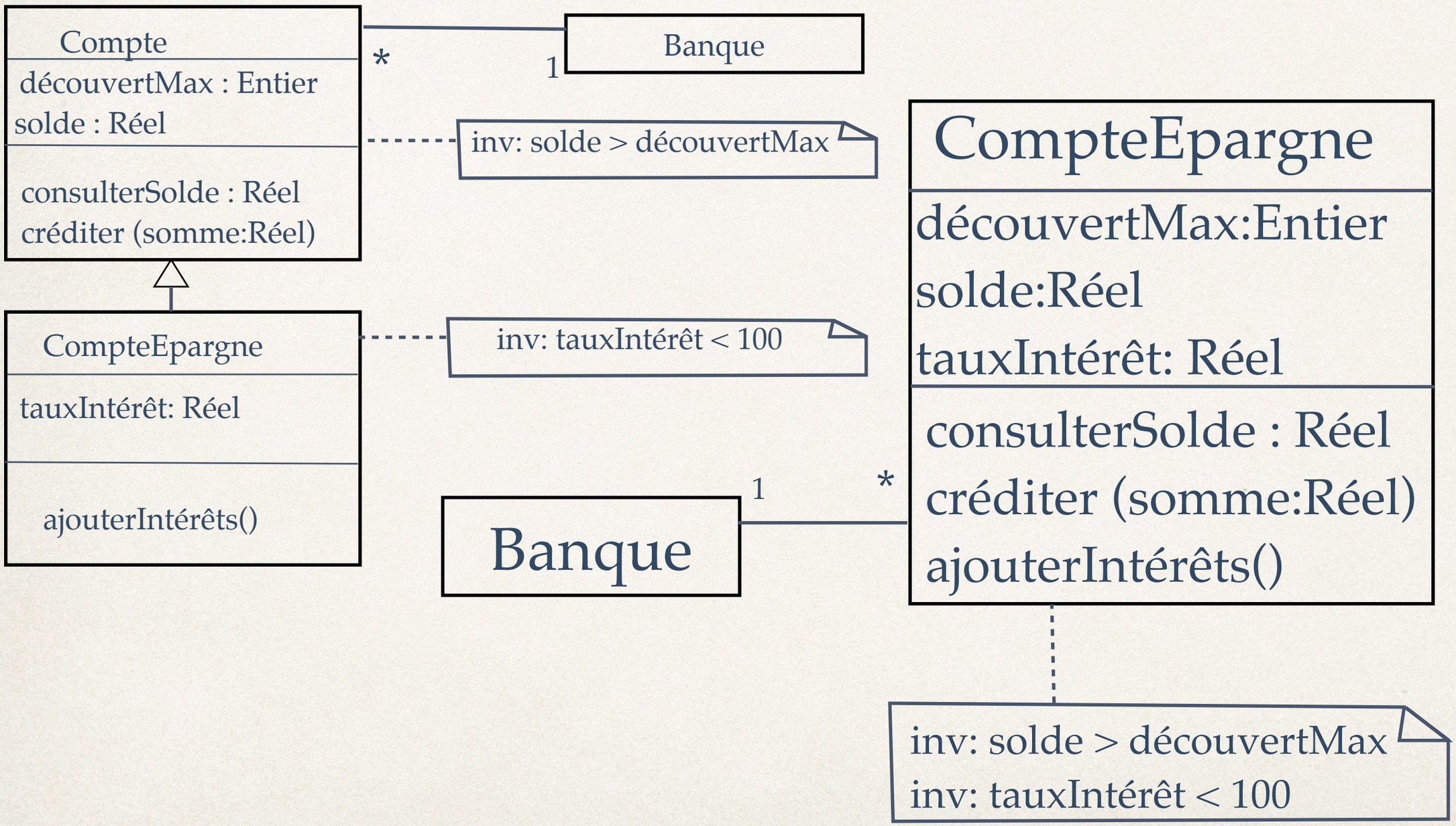
- One class can specialize another



Inheritance

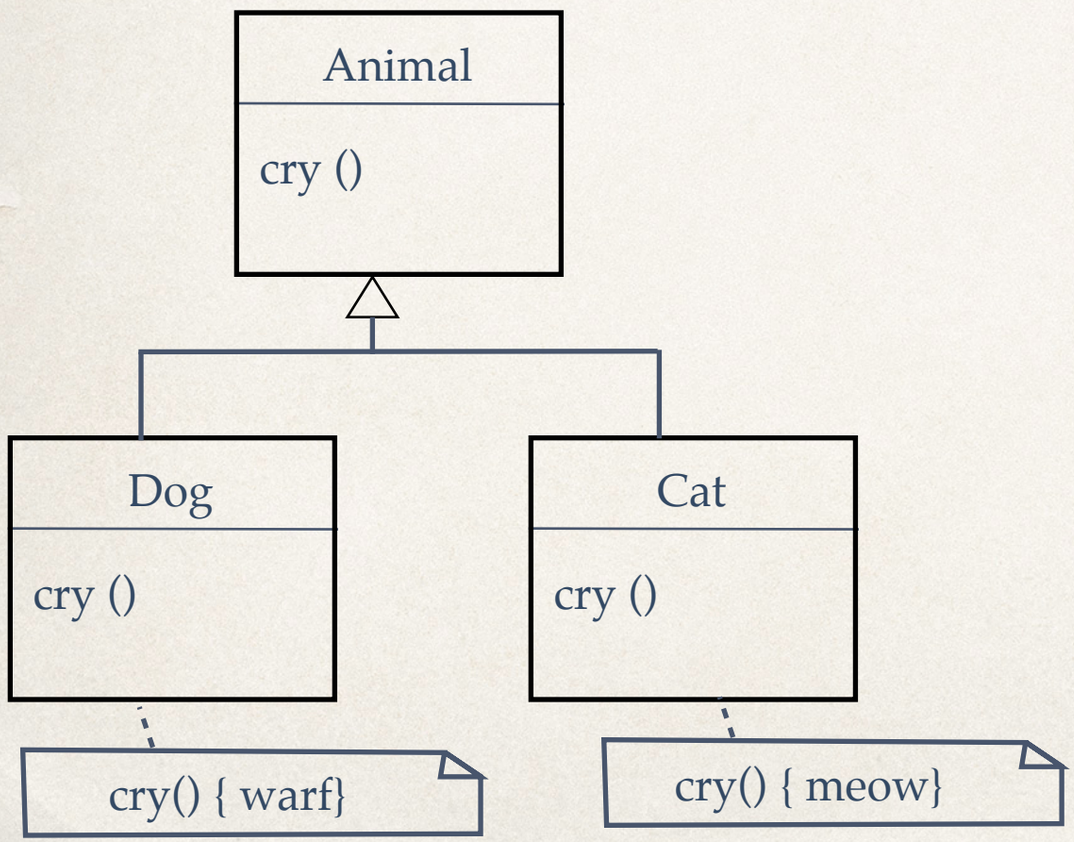


Héritage


Compte
découvertMax : Entier
solde : Réel
consulterSolde : Réel
créditer (somme:Réel)
*
1
Banque
inv: solde > découvertMax
CompteEpargne
tauxIntérêt: Réel
ajouterIntérêts()
inv: tauxIntérêt < 100
Banque
1
*
CompteEpargne
découvertMax:Entier
solde:Réel
tauxIntérêt: Réel
consulterSolde : Réel
créditer (somme:Réel)
ajouterIntérêts()
inv: solde > découvertMax
inv: tauxIntérêt < 100

Polymorphism

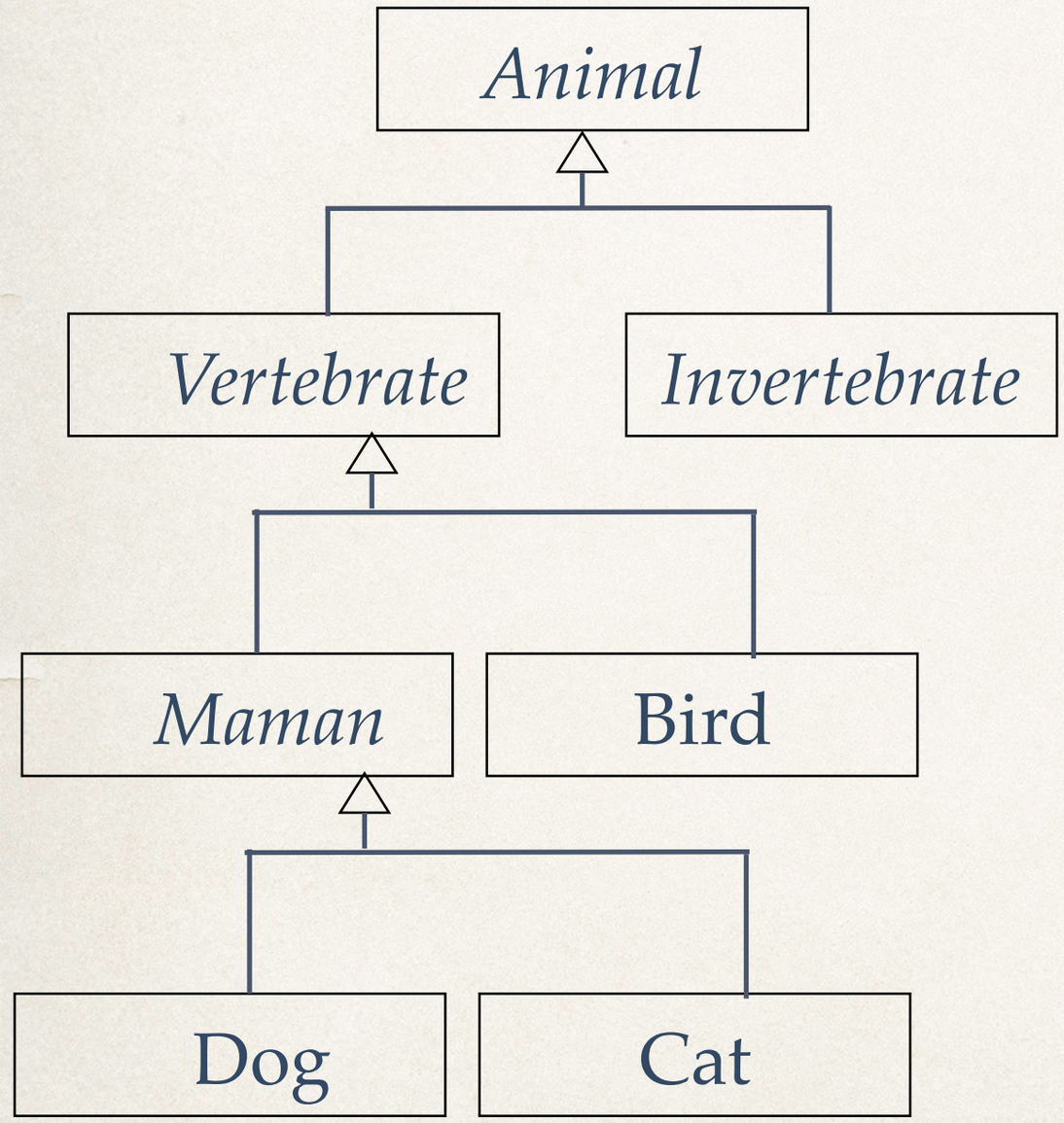
- One operation can be refined in a sub-class
- Allows to have methods whose behavior is specific to the particular type of class




```
Animal a;
if <condition>
        a = new (Dog)
else
        a=new (Cat)
endif
a.cry();
```

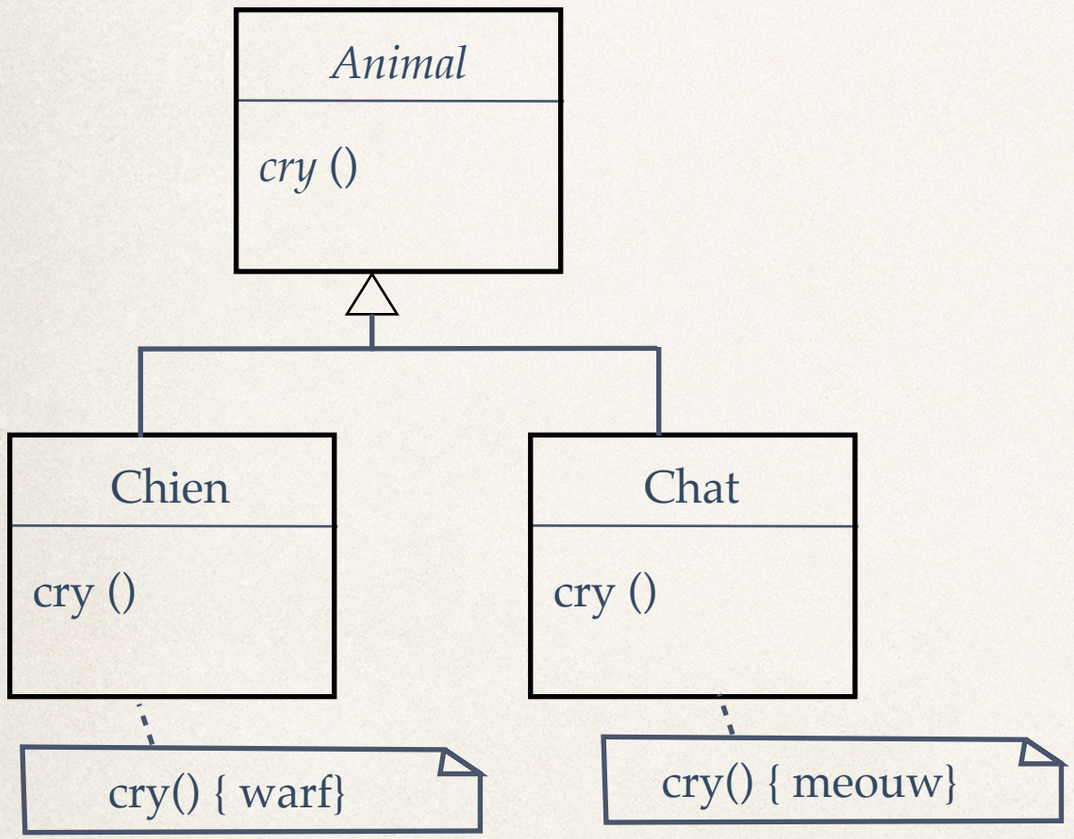
The body of this operation is linked dynamically

Abstract classes



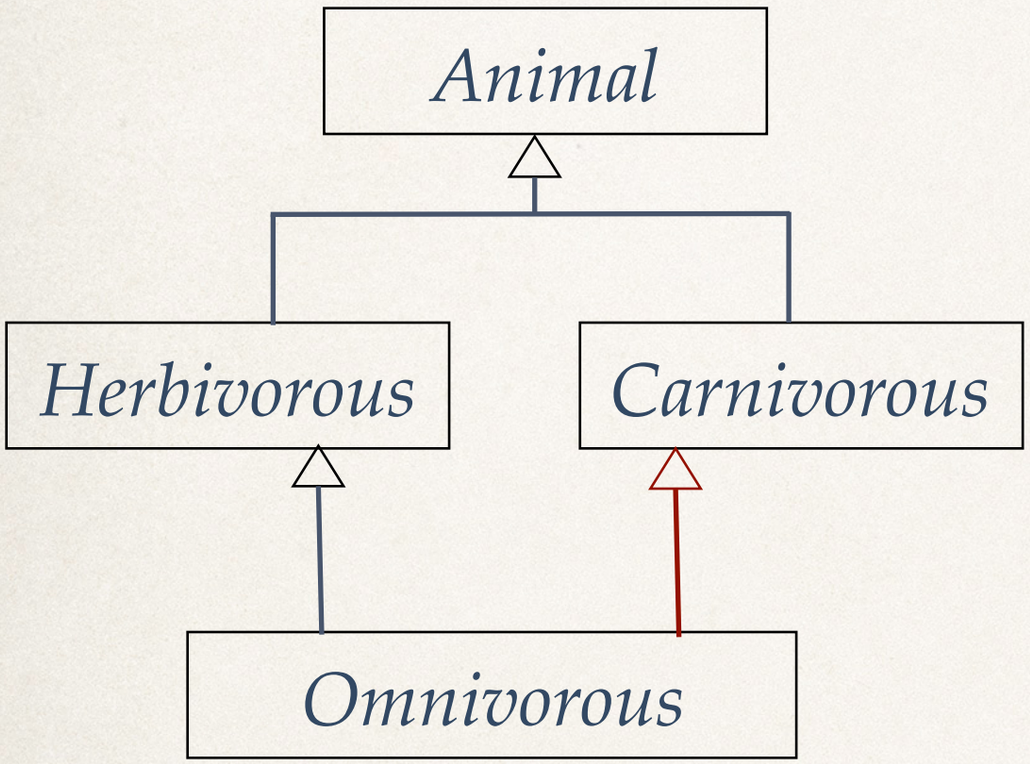
- Some classes exist for structuring the inheritance hierarchy
  abstract class (as opposed to concrete class)
- An abstract class **cannot be instantiated**
- **Useful to define an abstract behavior**
- **Notation: name in italic**

Abstract methods



- Some methods define only an abstract behavior
- **Notation: name in italic**
- **They are part of abstract classes**
- **Need to be refined to be actually used**
-

Multiple inheritance



- One class can inherit more than one super-classes
- Impossible in some programming languages (Java, C#)
- Allowed in UML

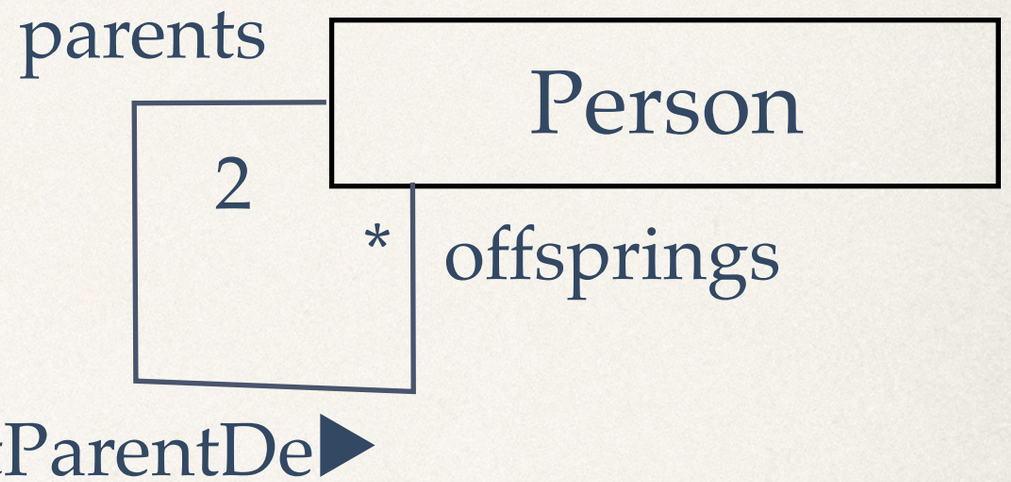
Plan

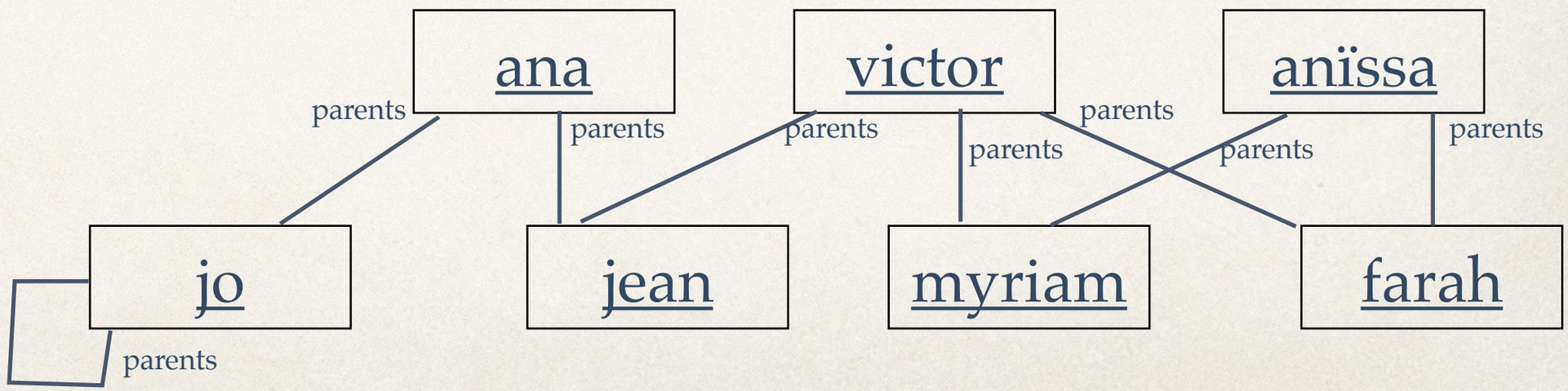
- Notions de base
- **Conformité**
  diagramme de classes - système modélisé,
  diagramme d'instances - diagramme de classes
- Concepts avancés
- Usage des diagrammes de classes

Conformity
Instances - class diagrams

- Are they conforming?

Does not conform to the class diagram



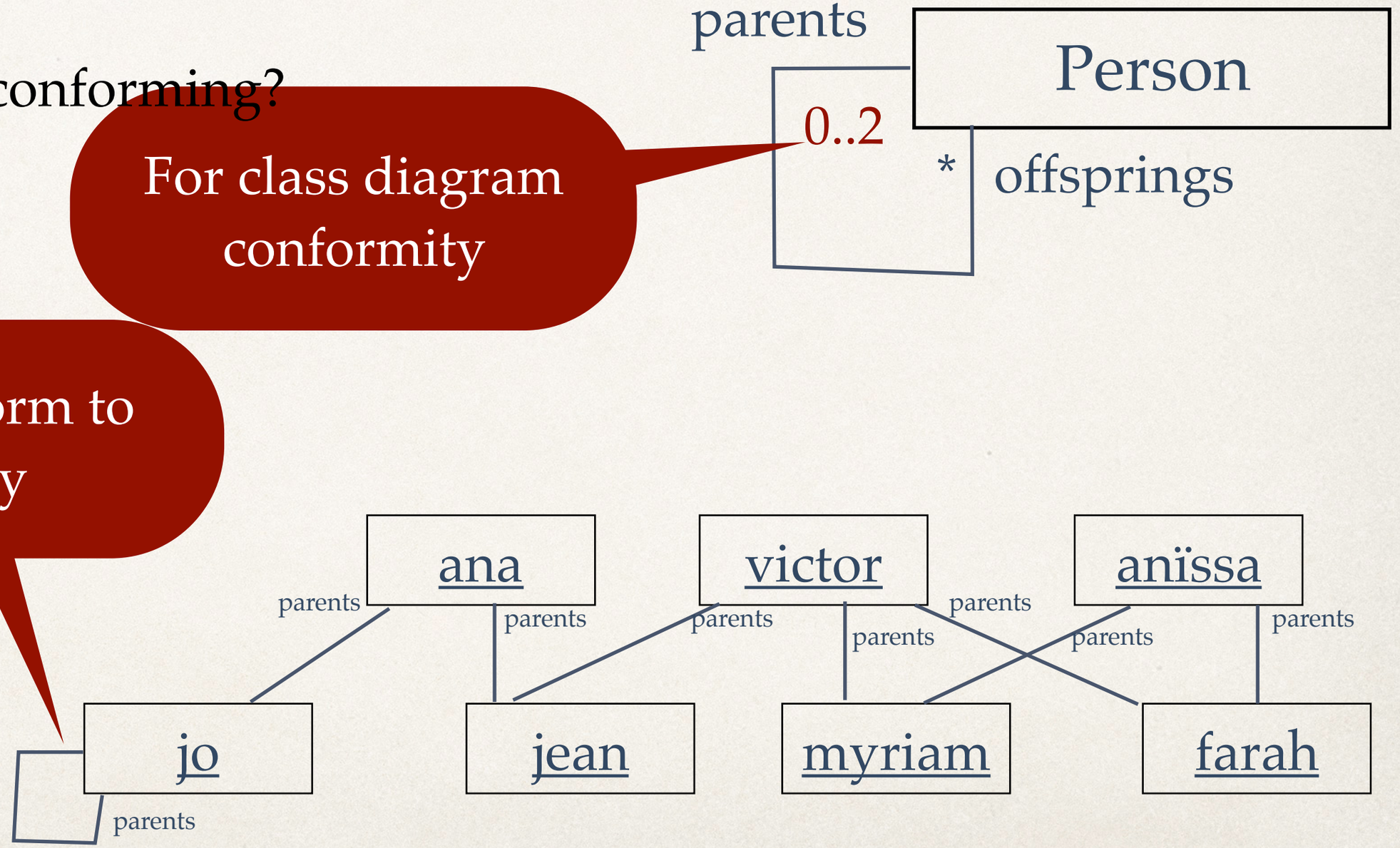


Conformity
Class diagram vs modeled system

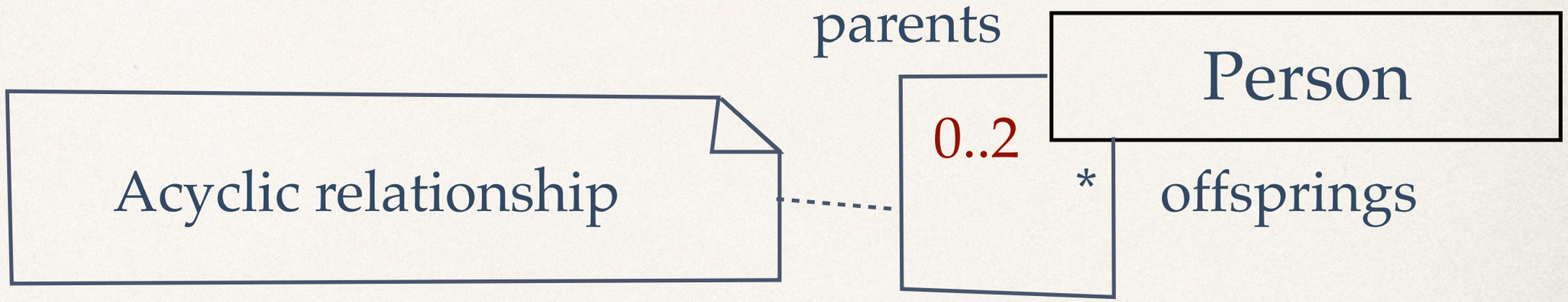
- Are they conforming?

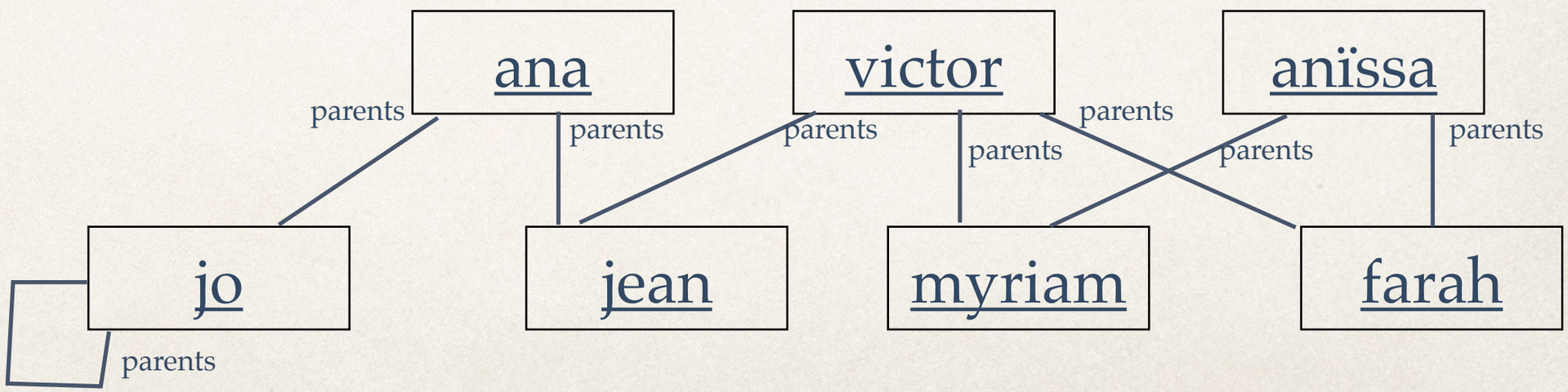
For class diagram conformity

Not conform to reality



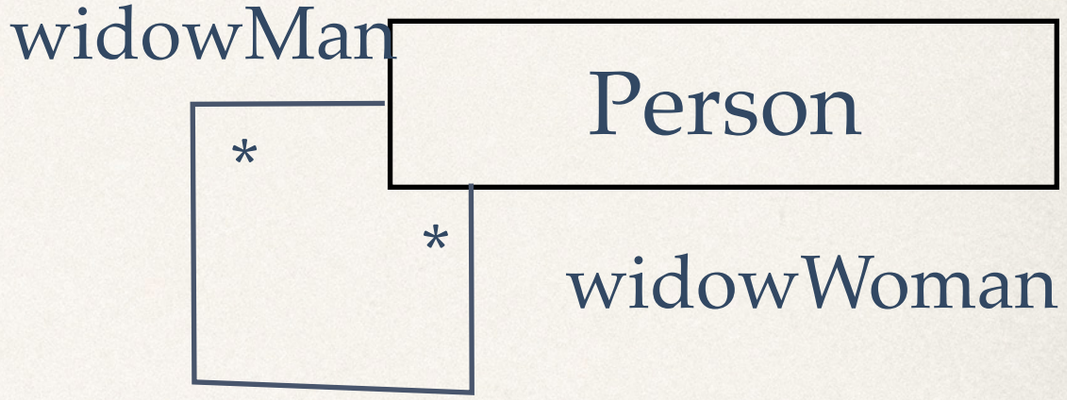
Add constraints





Testing the model is important

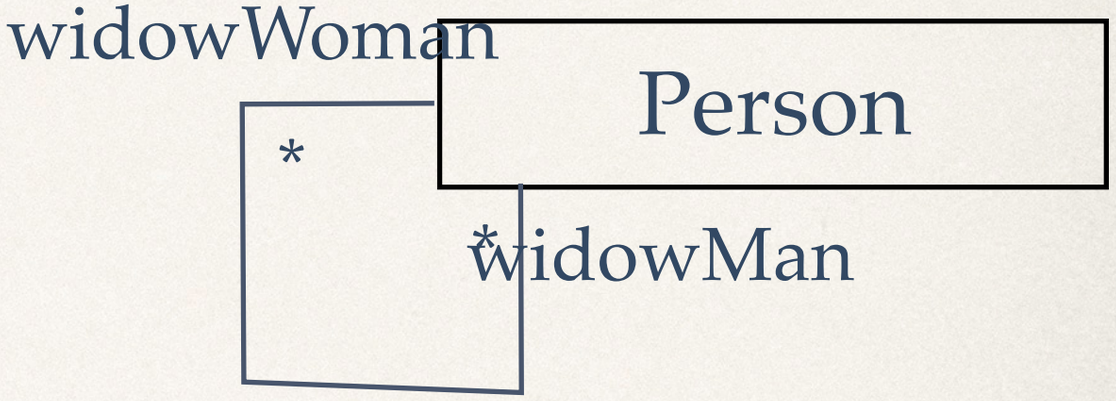
- test cardinalities
- tester la terminology
- identify missing constraints



Un untested model is most likely erroneous

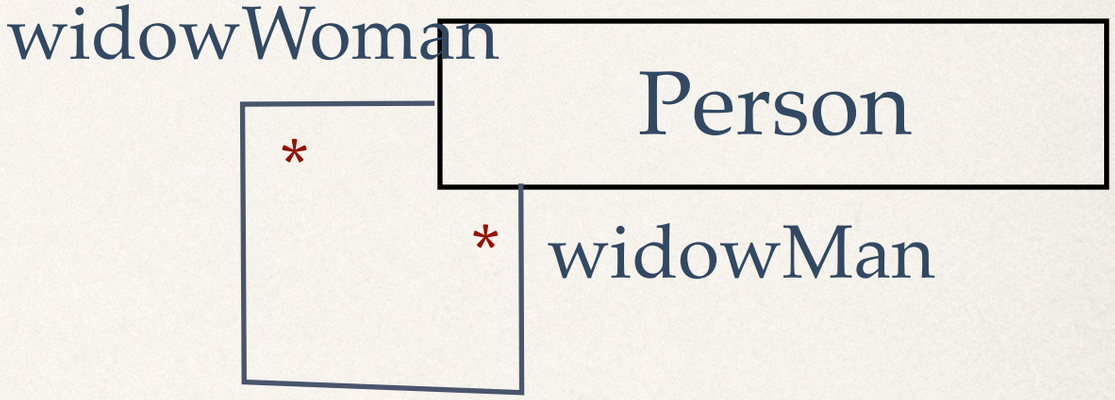
Testing the model is important

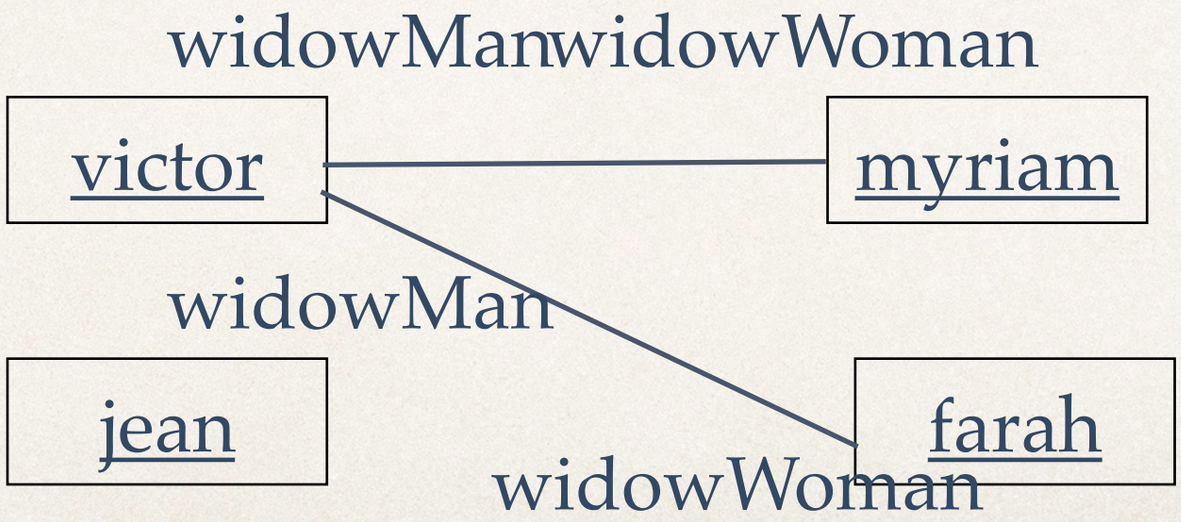
- **test cardinalities**
- tester la terminology
- identify missing constraints



Testing the model is important

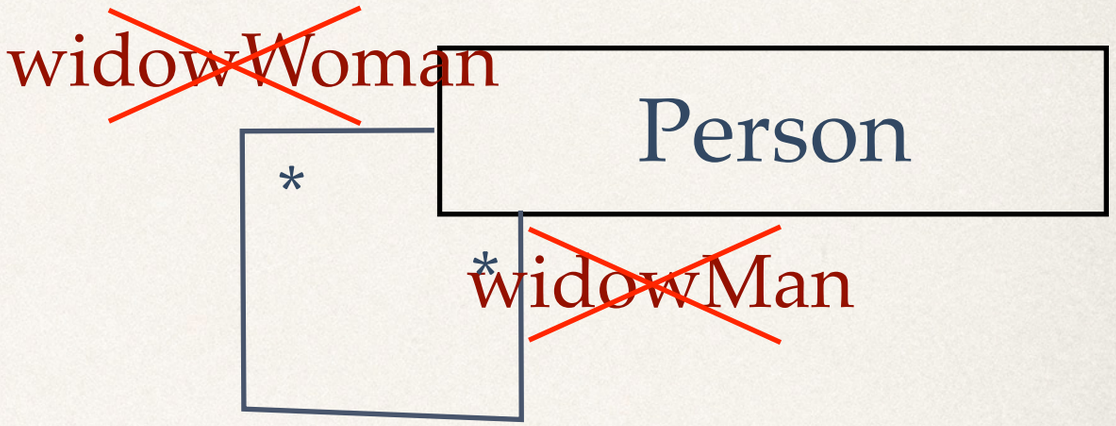
- **test cardinalities**
- tester la terminology
- identify missing constraints

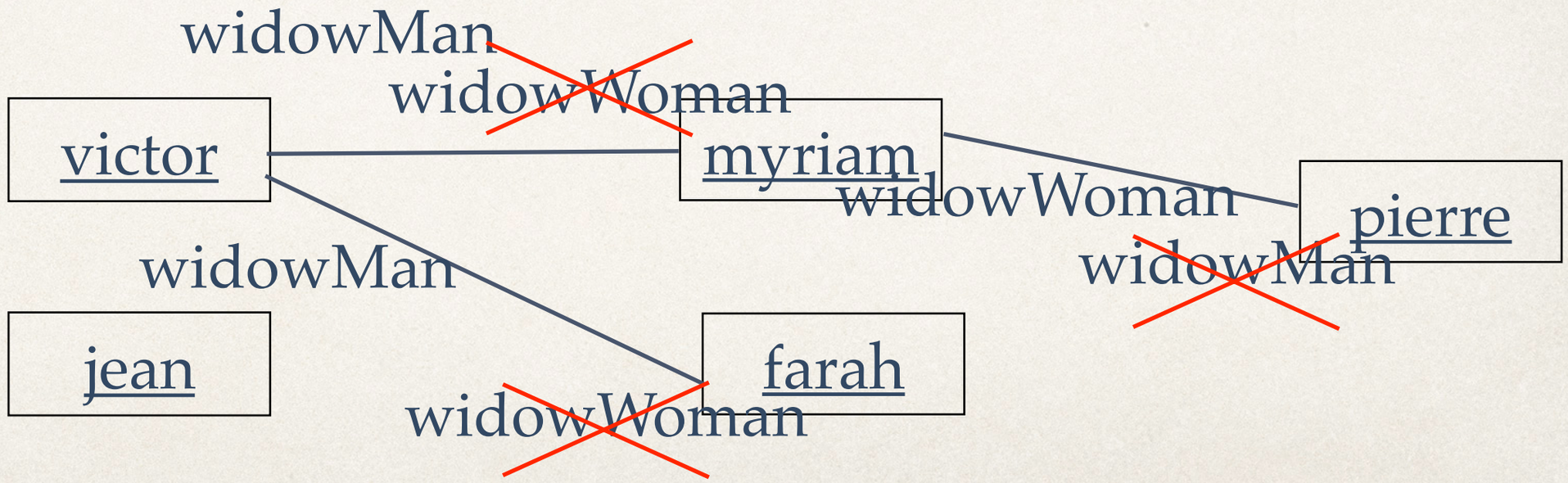




Testing the model is important

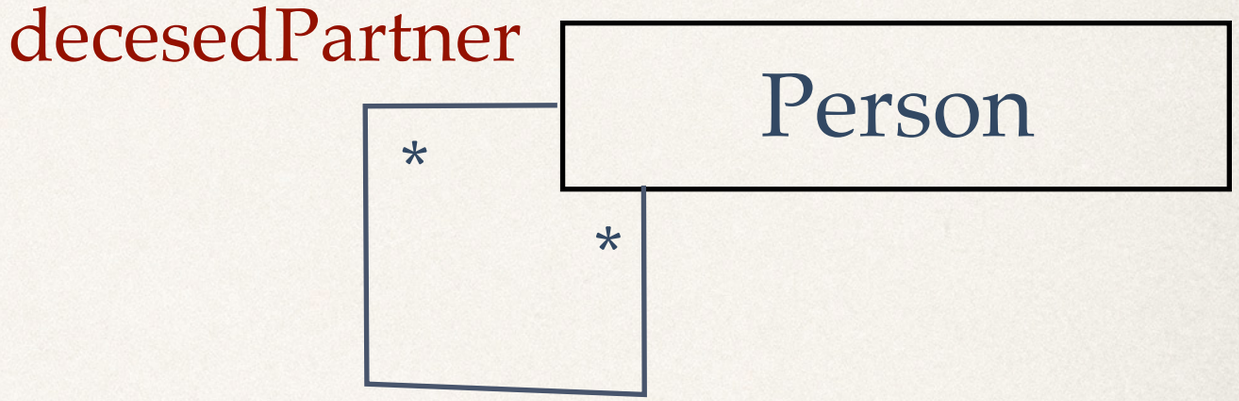
- test cardinalities
- **tester la terminology**
- identify missing constraints

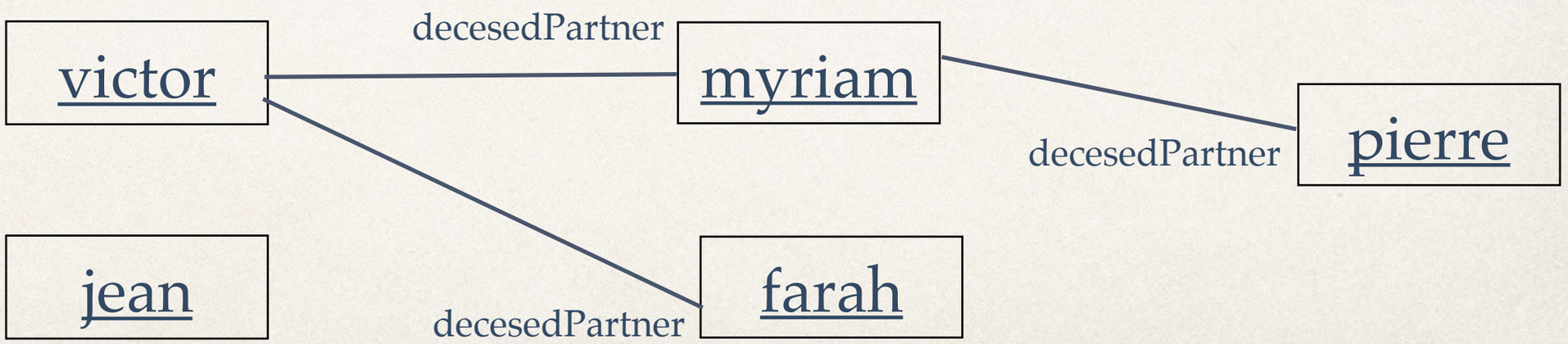




Testing the model is important

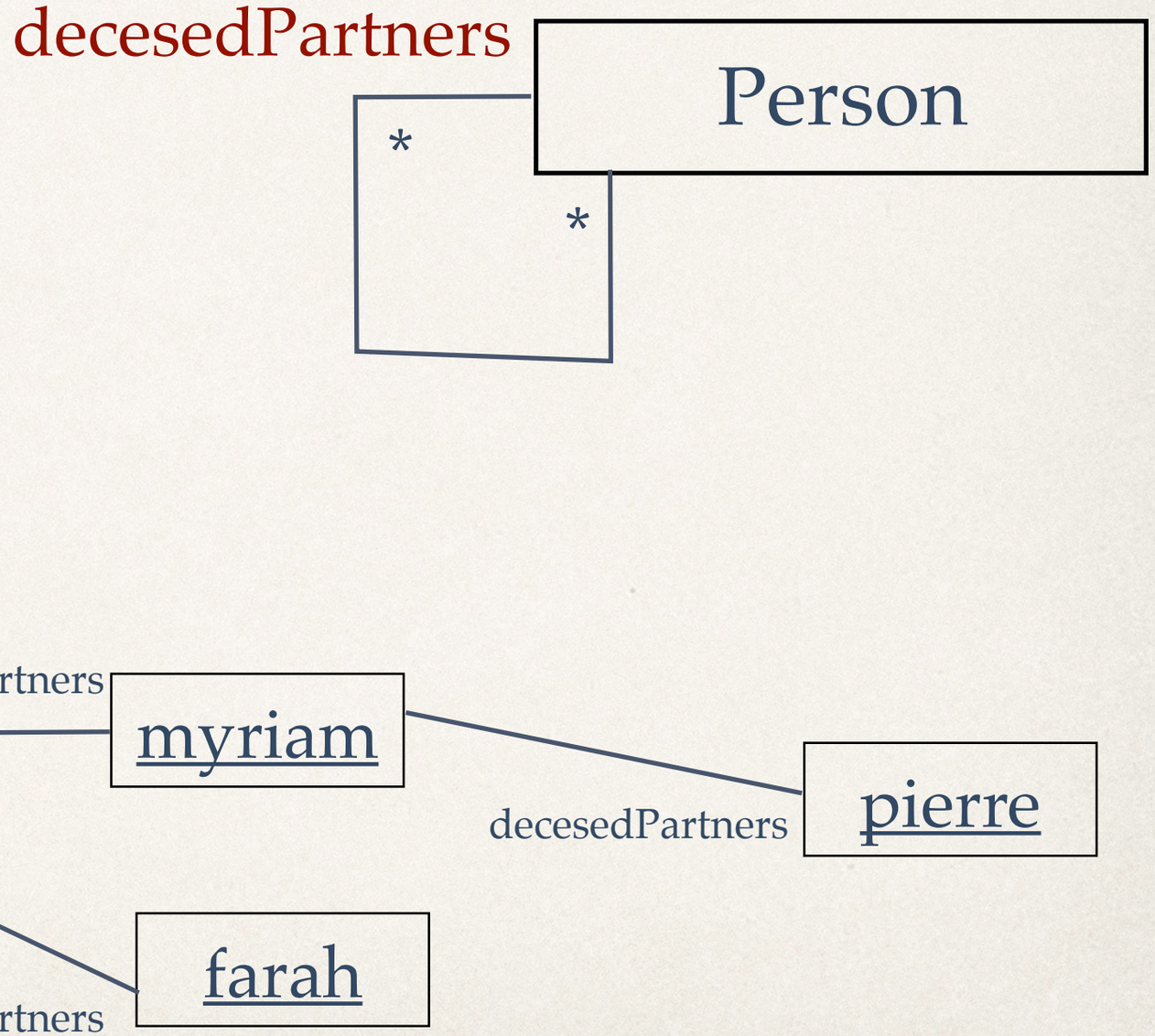
- test cardinalities
- **tester la terminology**
- identify missing constraints





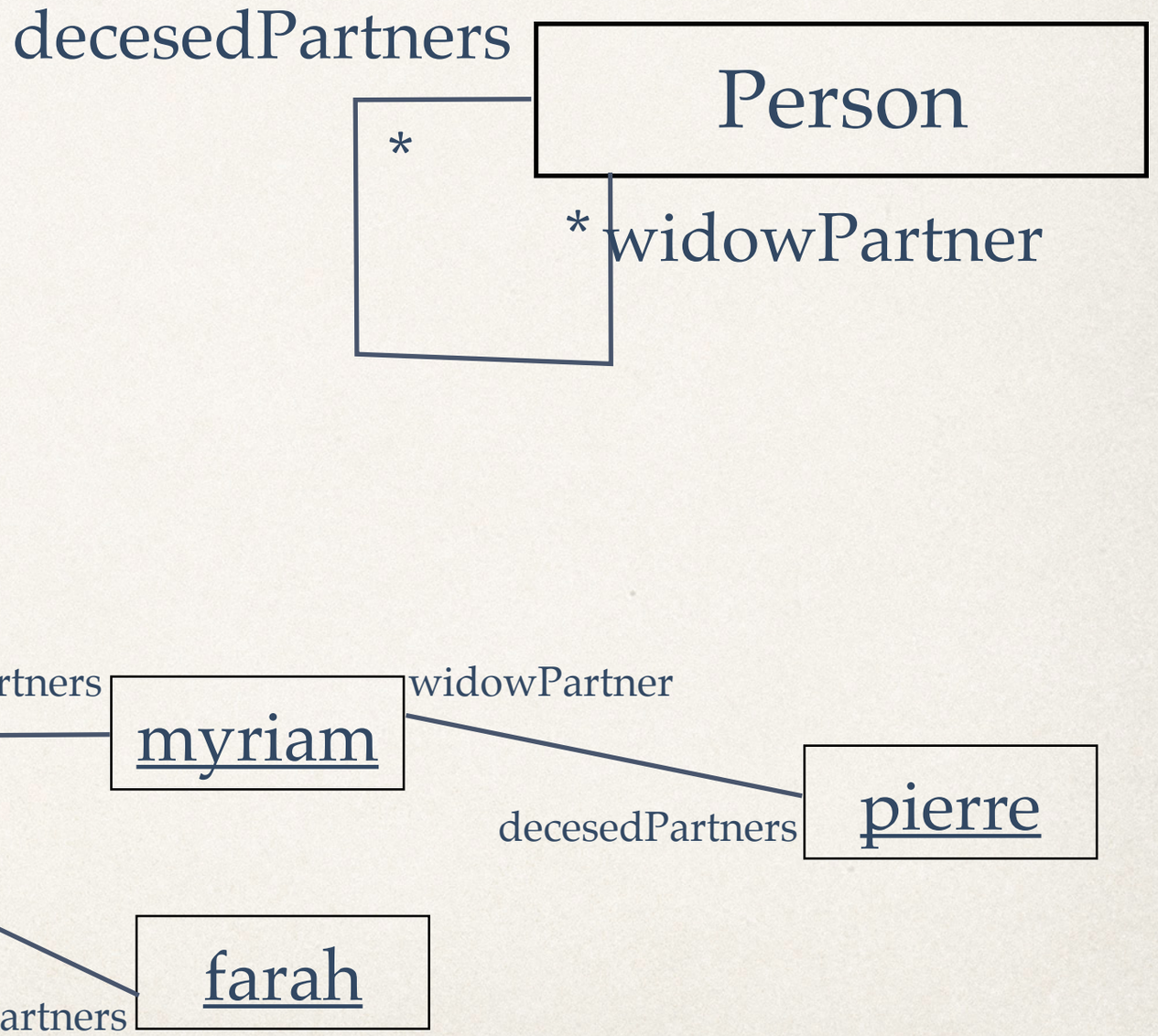
Testing the model is important

- test cardinalities
- **tester la terminology**
- identify missing constraints



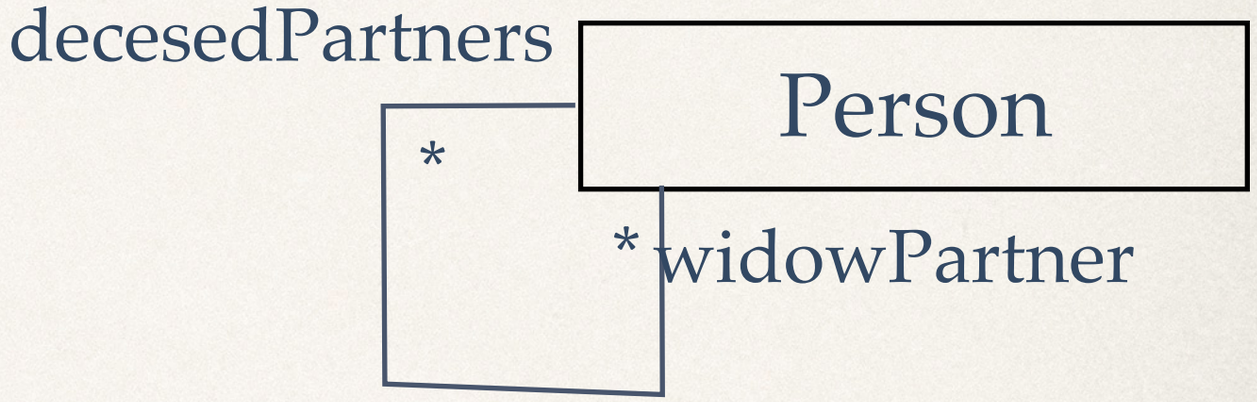
Testing the model is important

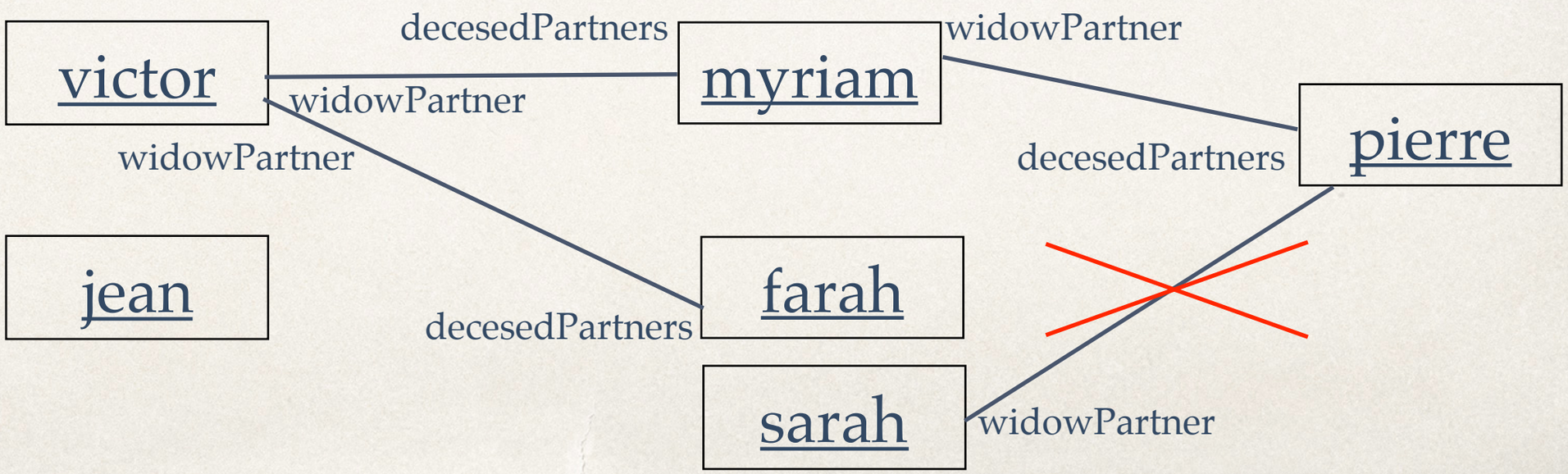
- test cardinalities
- **tester la terminology**
- identify missing constraints



Testing the model is important

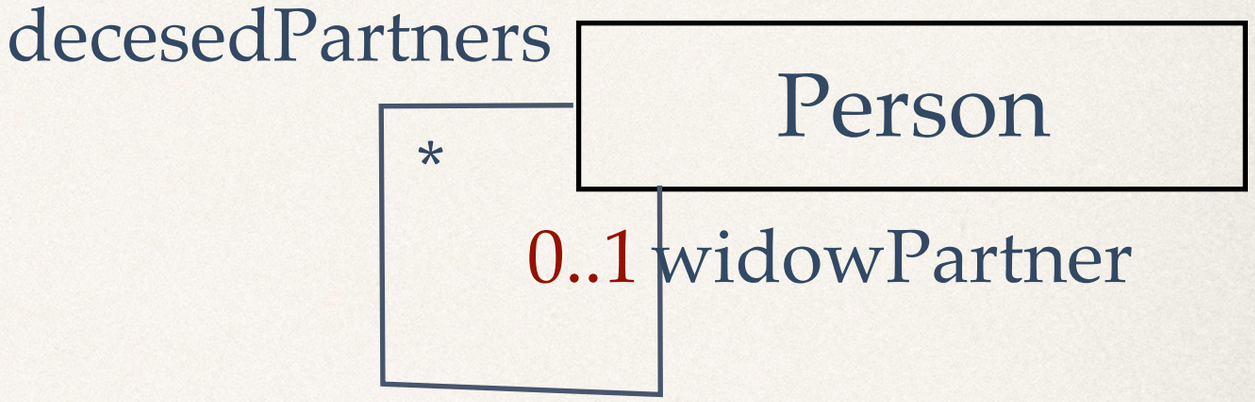
- **test cardinalities**
- tester la terminology
- identify missing constraints

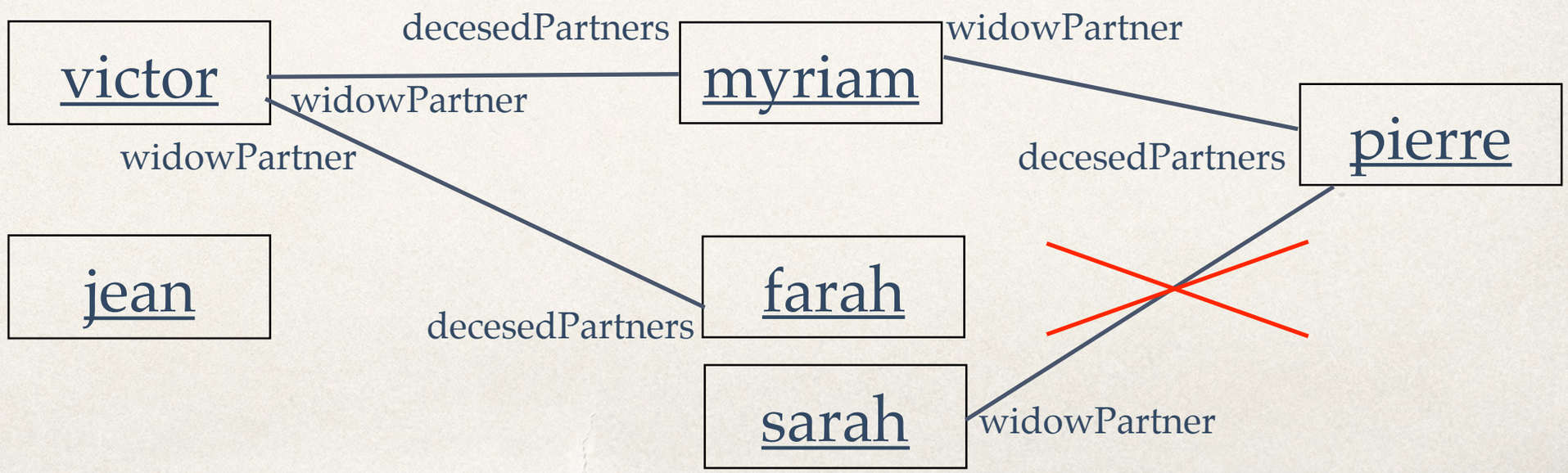




Testing the model is important

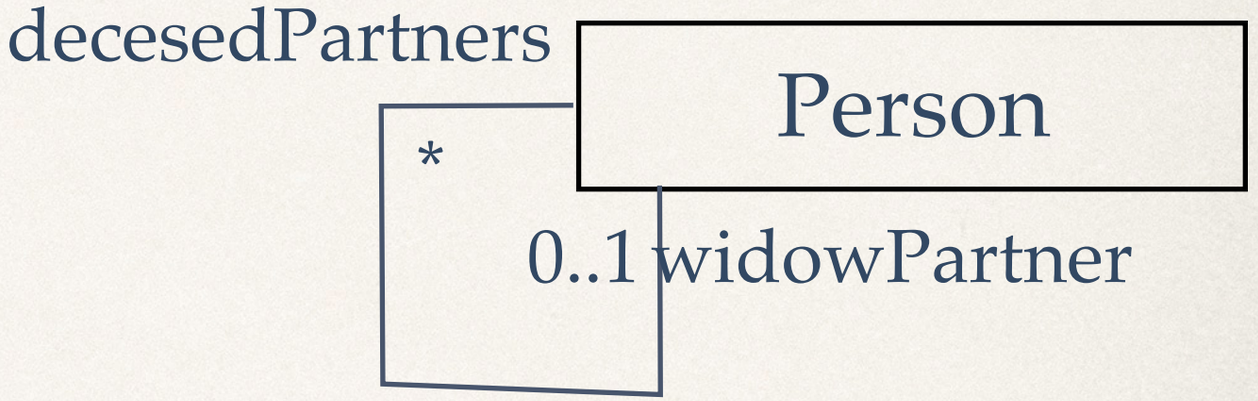
- **test cardinalities**
- tester la terminology
- identify missing constraints

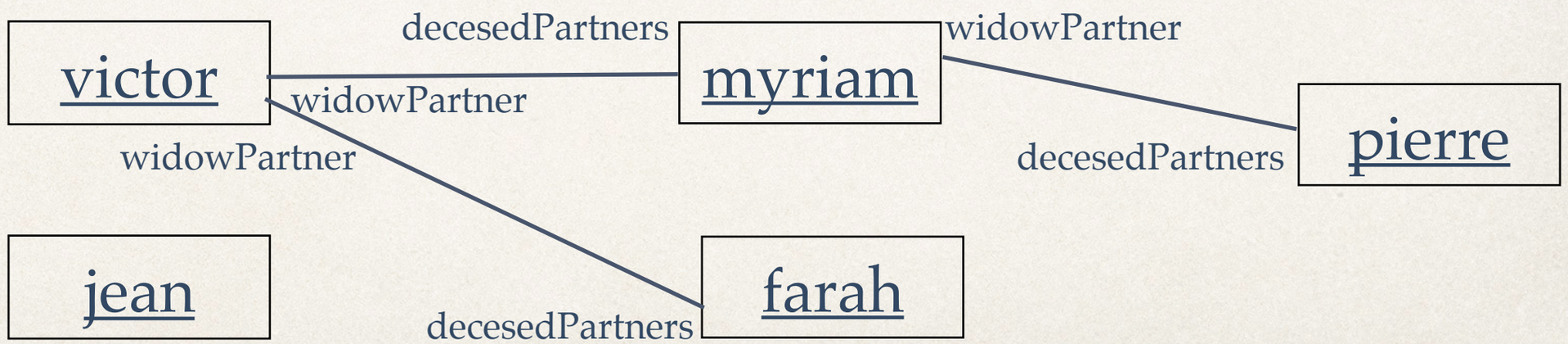




Testing the model is important

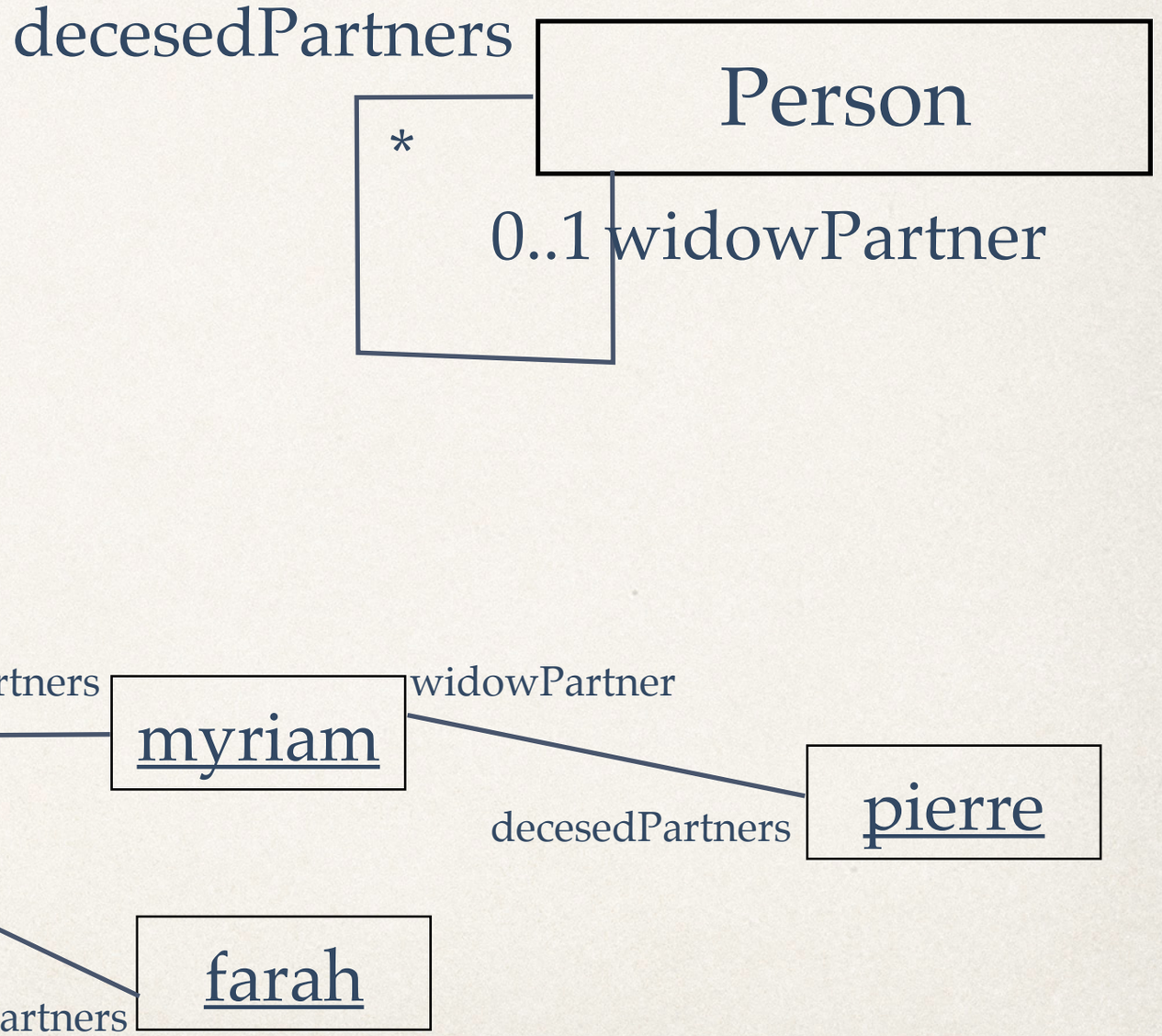
- **test cardinalities**
- tester la terminology
- identify missing constraints





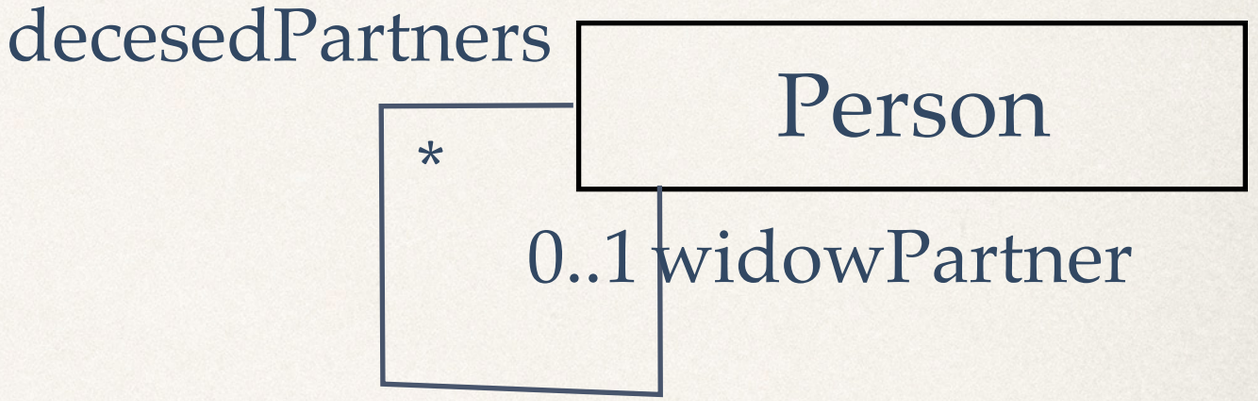
Testing the model is important

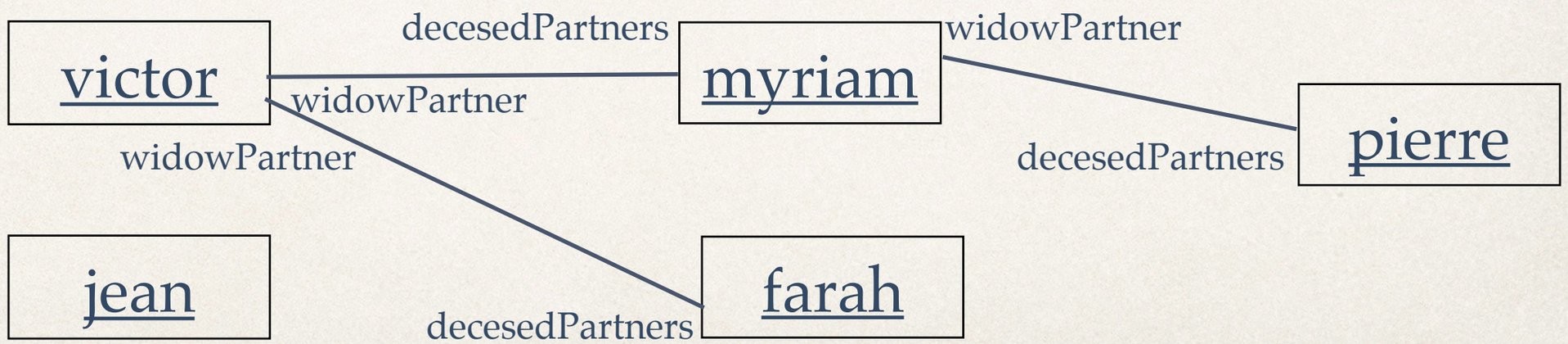
- test cardinalities
- **tester la terminology**
- identify missing constraints



Testing the model is important

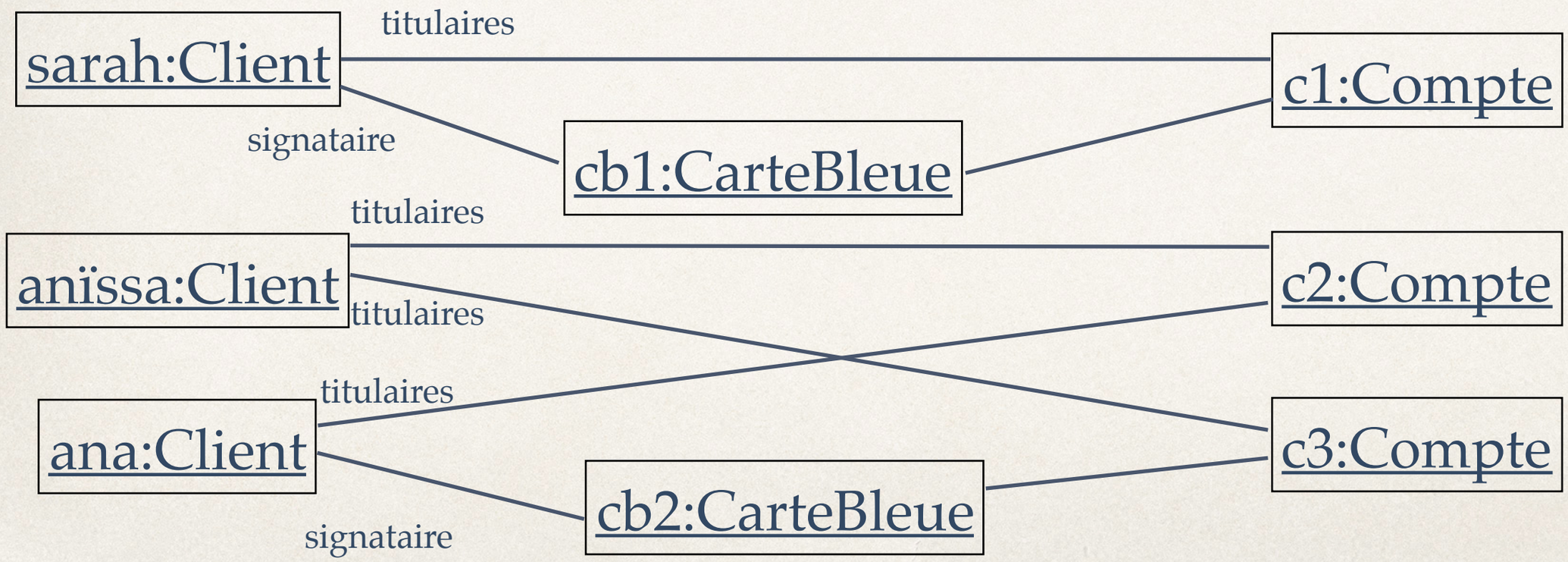
- test cardinalities
- tester la terminology
- **identify missing constraints**





Constraints and associations

take care to constraints too obvious that are not expressed



Plan

- Notions de base
- Conformité
  diagramme de classes - système modélisé,
  diagramme d'instances - diagramme de classes
- **Concepts avancés**
- Usage des diagrammes de classes

Concepts avancés

- visibilité
- énumération
- association - notions avancée
  (composition, aggregation, classe

A utiliser en
fonction de l'étape de
développement, la nature du projet,
les outils de modélisation, ...

nnée)

Visibility

- A technique allowing to restrict the access to some information

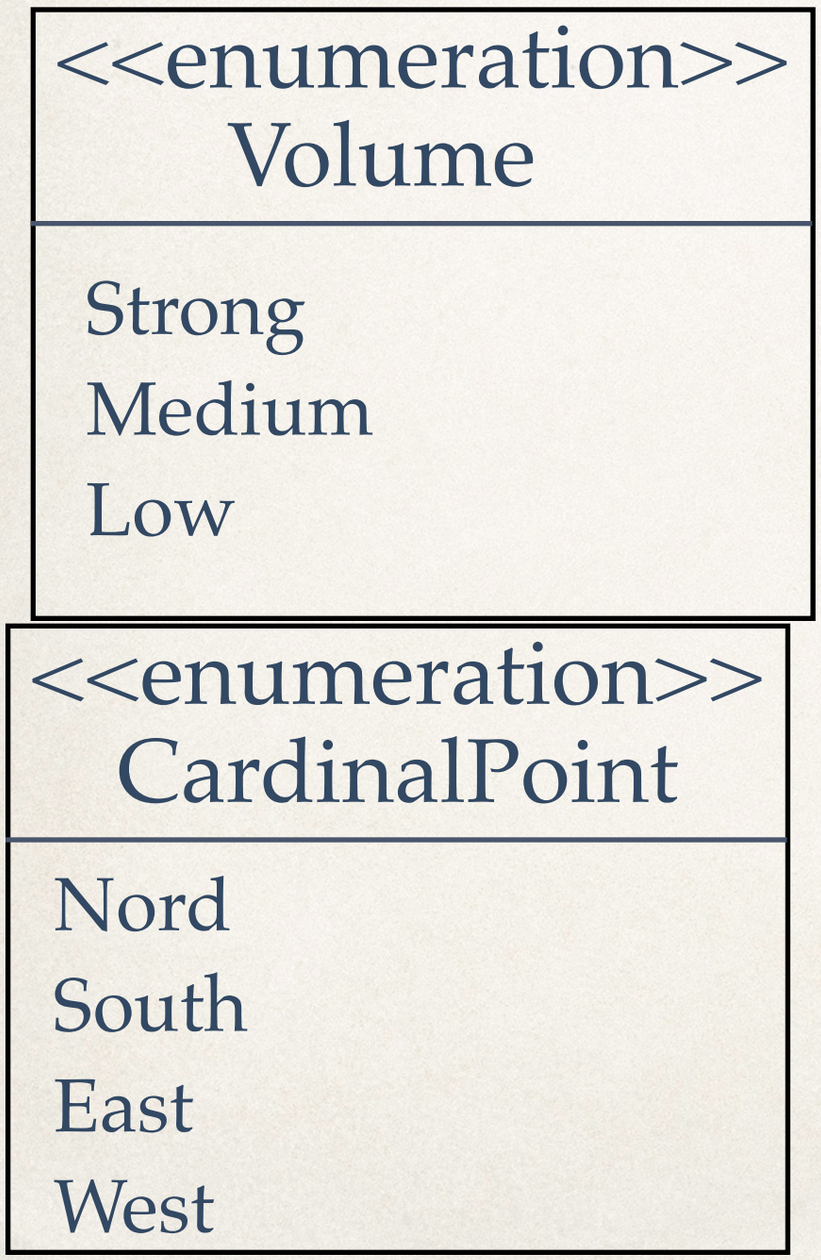
  + public visible
  # protected visible within the class and its sub-classes
  - private visible only within the class
  ~ package visible only within the package

- Useful in the detailed design and during implementation (not before!)

- No point in the conceptual model (and business design)

- Its semantics depends on the target programming language

Enumeration



- no ordering between values

Usage:

Association - advanced notions

- Navigation
- Composition, aggregation
- Predefined constraints
- Association class

Navigation

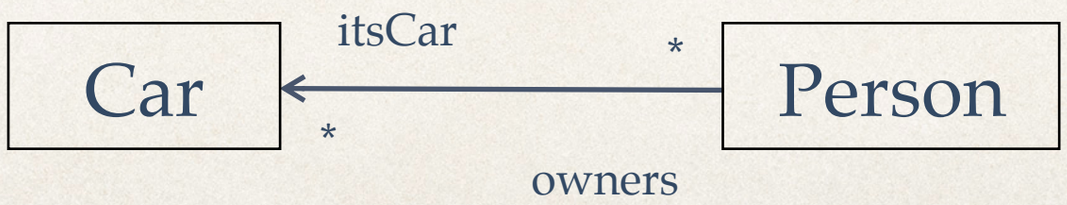
- Association uni-directional

- One can navigate in one direction

- Adds a constraint

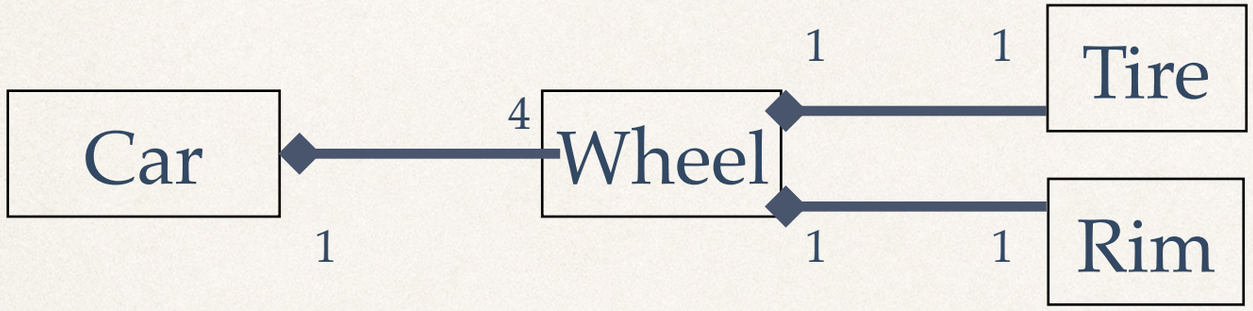
- If you hesitate, don't add it!

- Useful in design and implementation
  (not during analysis of business modeling)

- 

Composition

- Whole-part relationship



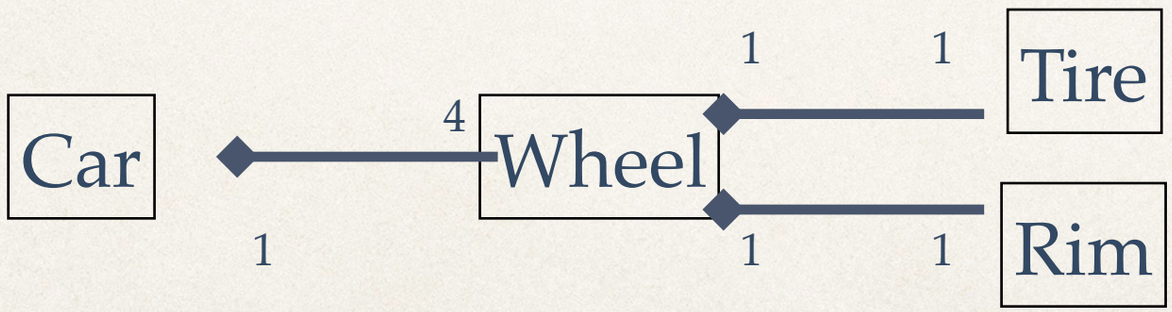
Composition - Constraints

Composition related constraints

1. An object can only be composed **within** one other object

2. A composed object cannot exists without its « whole »

3. If a composed object is destroyed its parts should be also destroyed



may depend of the situation to be modeled (car selling vs. car breaks)

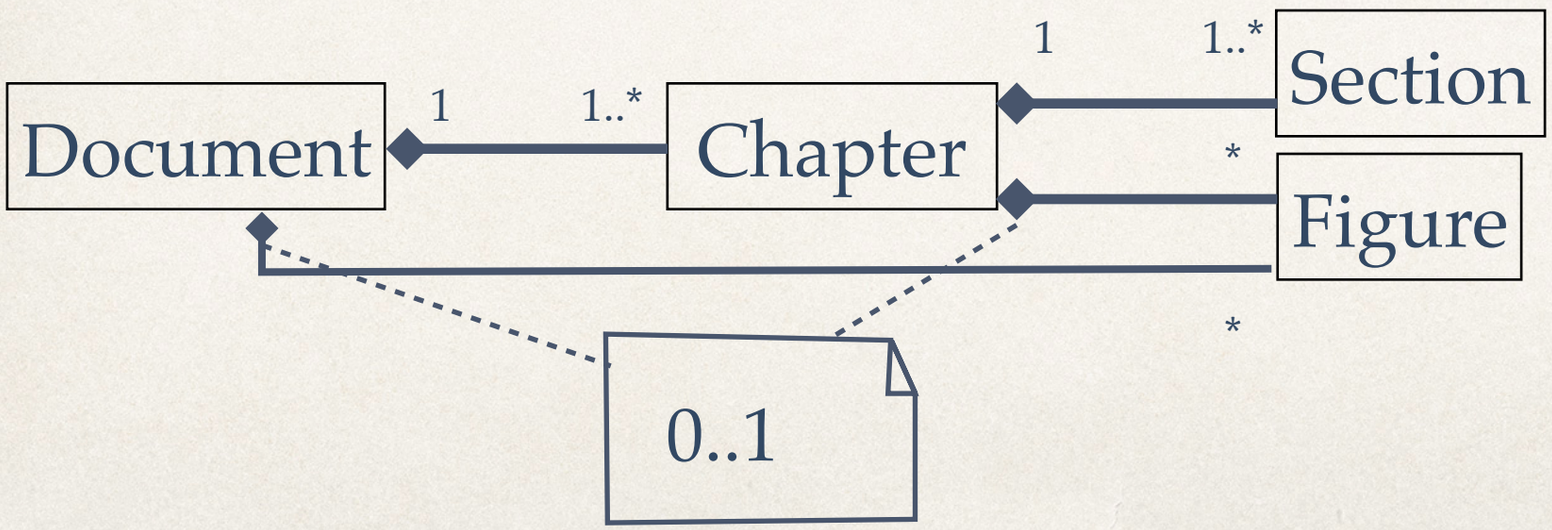
Composition - example

Composition related constraints

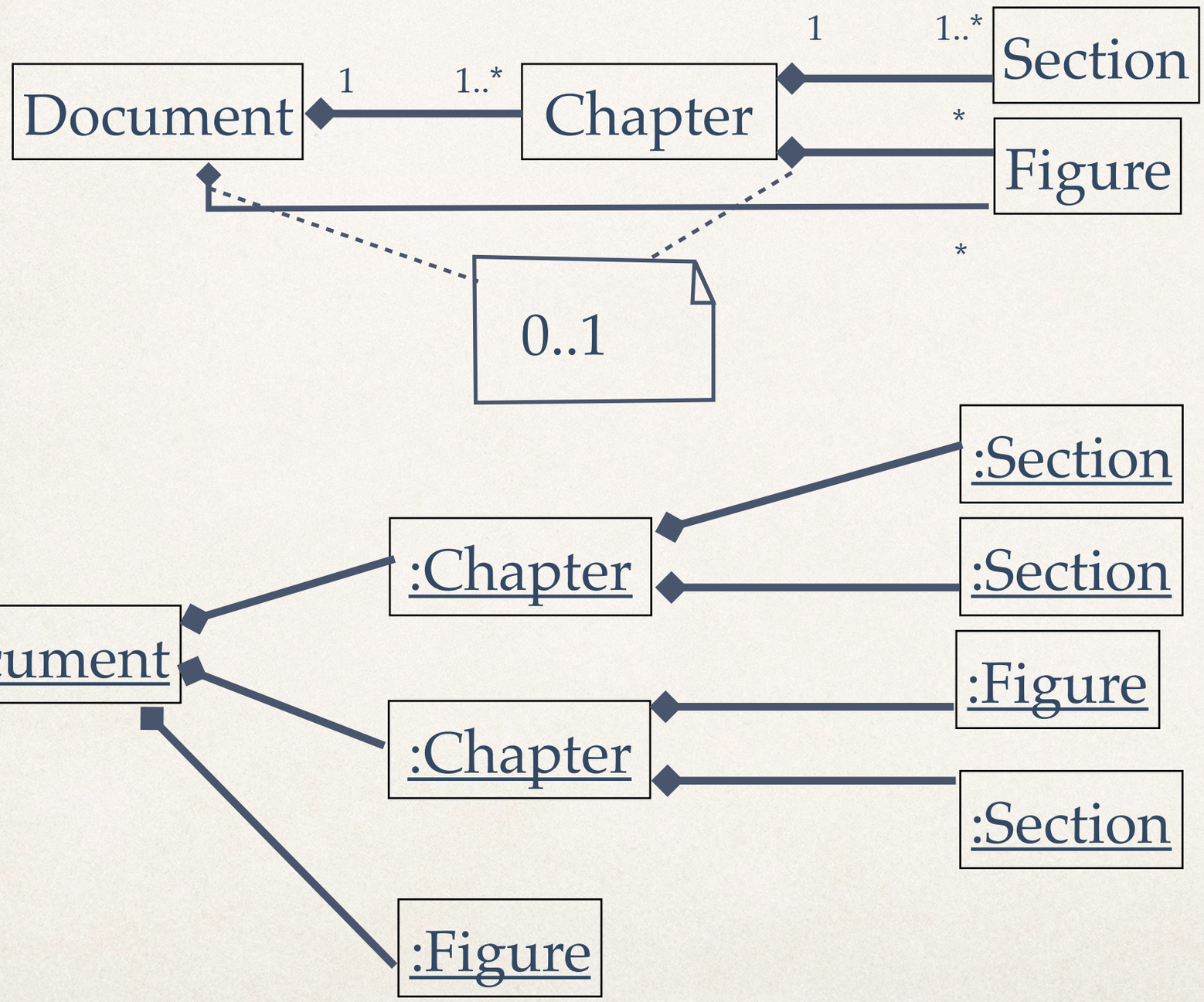
1. An object can only be composed **within** one other object

2. A composed object cannot exists without its « whole »

3. If a composed object is destroyed its parts should be also destroyed

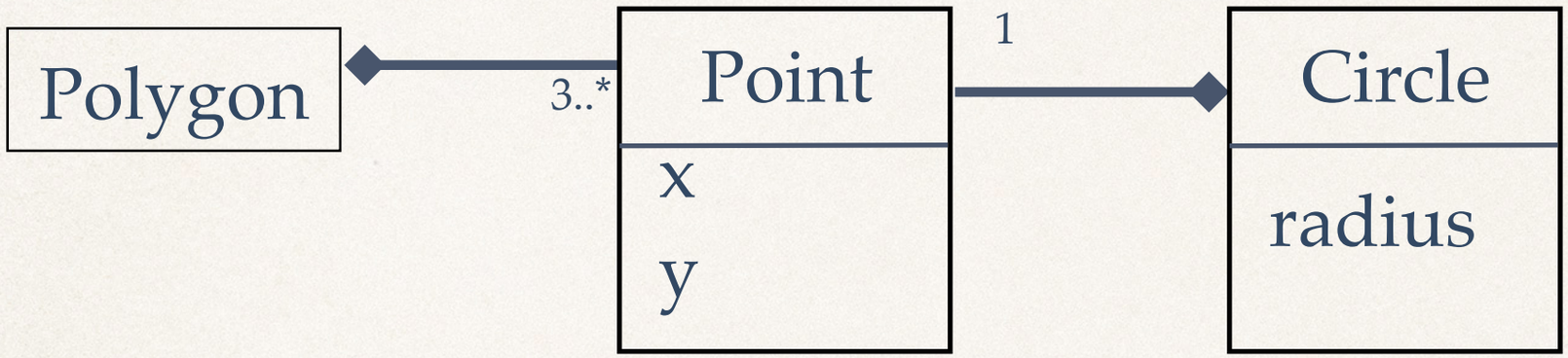


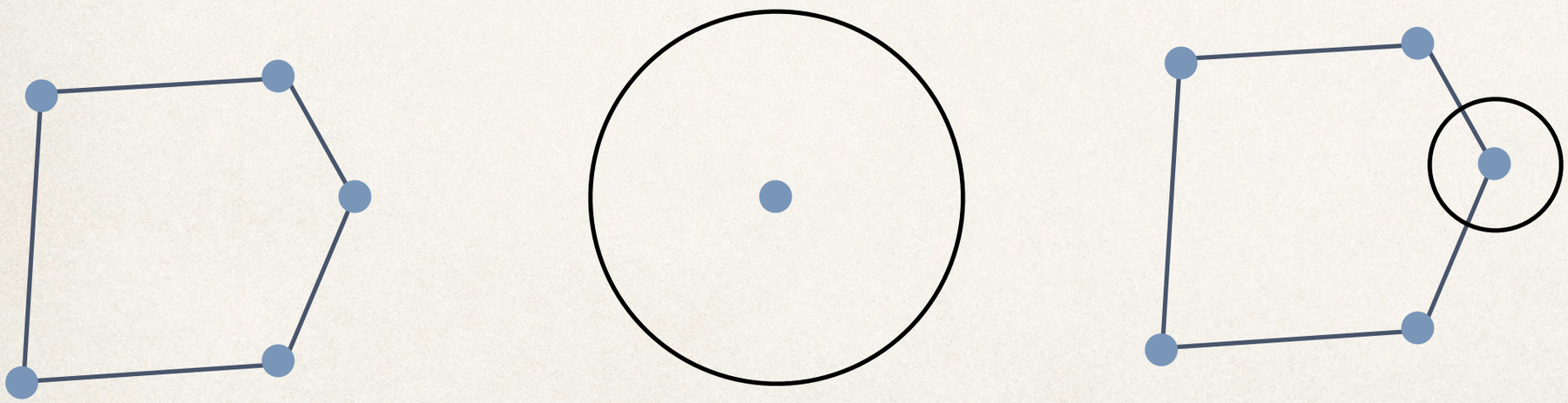
Composition



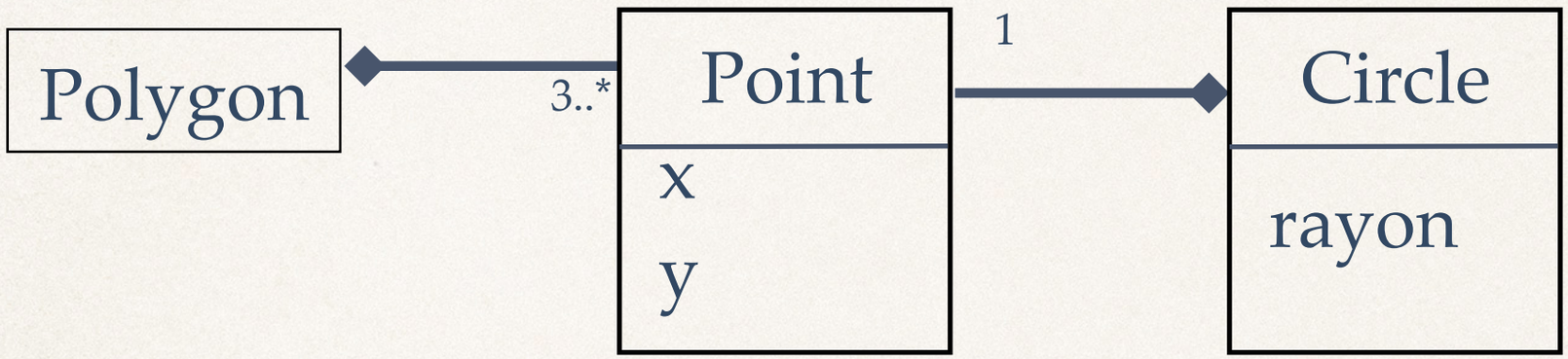
Les composants forment un arbre

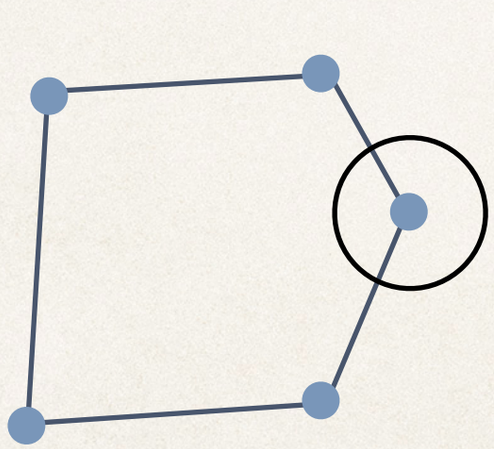
Composition -

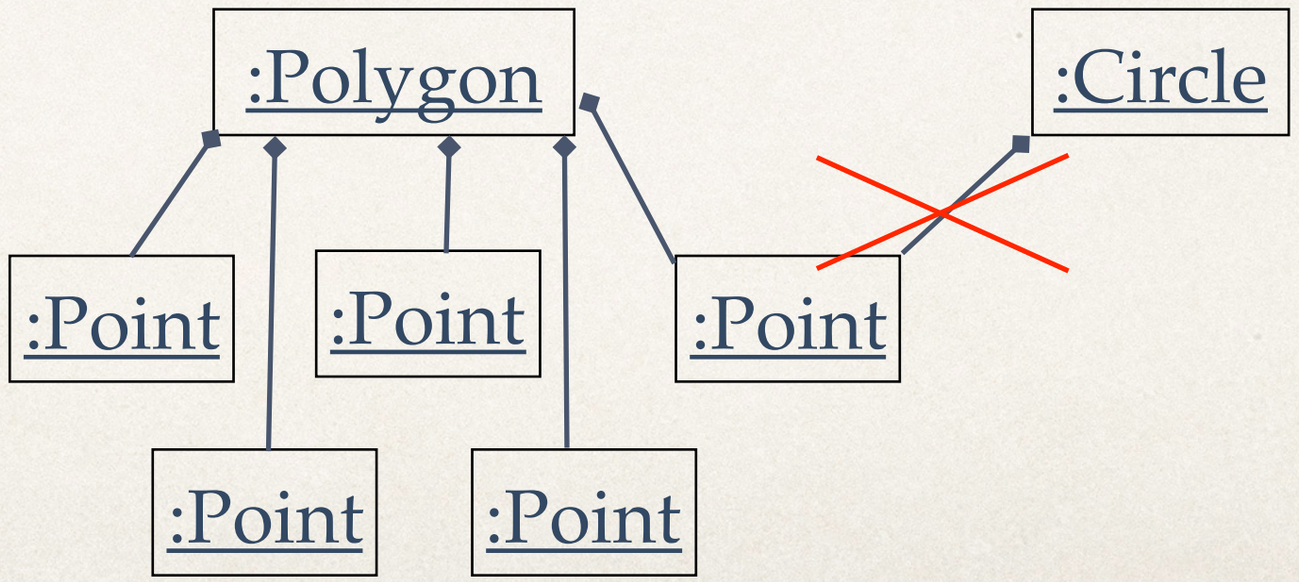




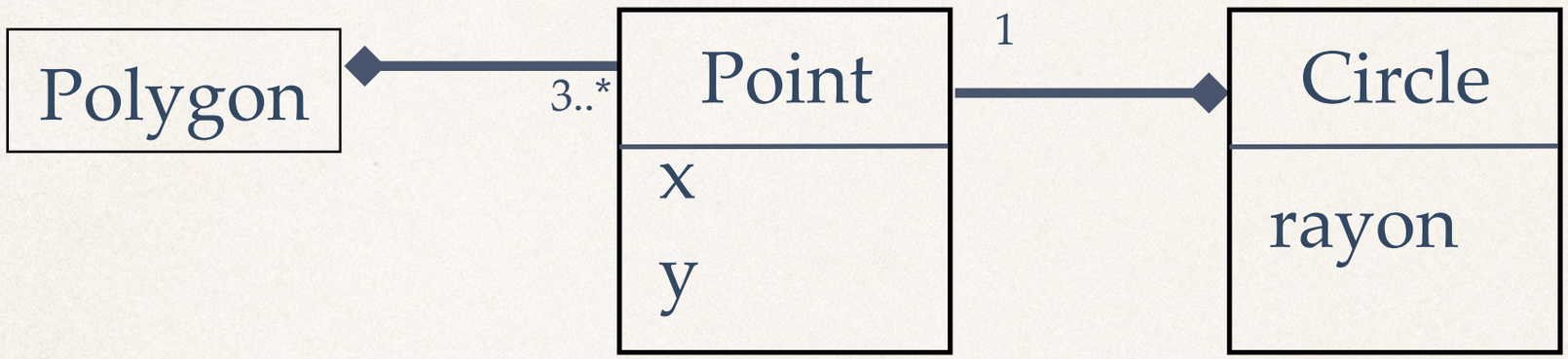
Composition -

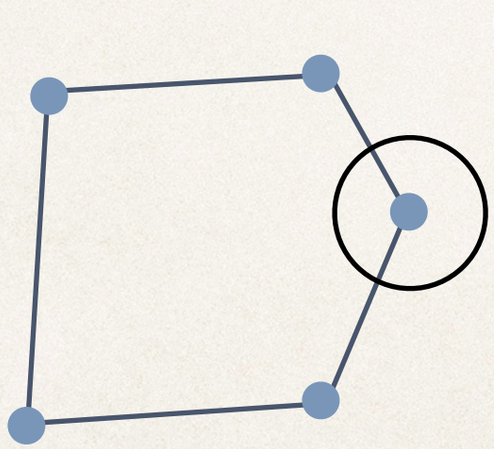


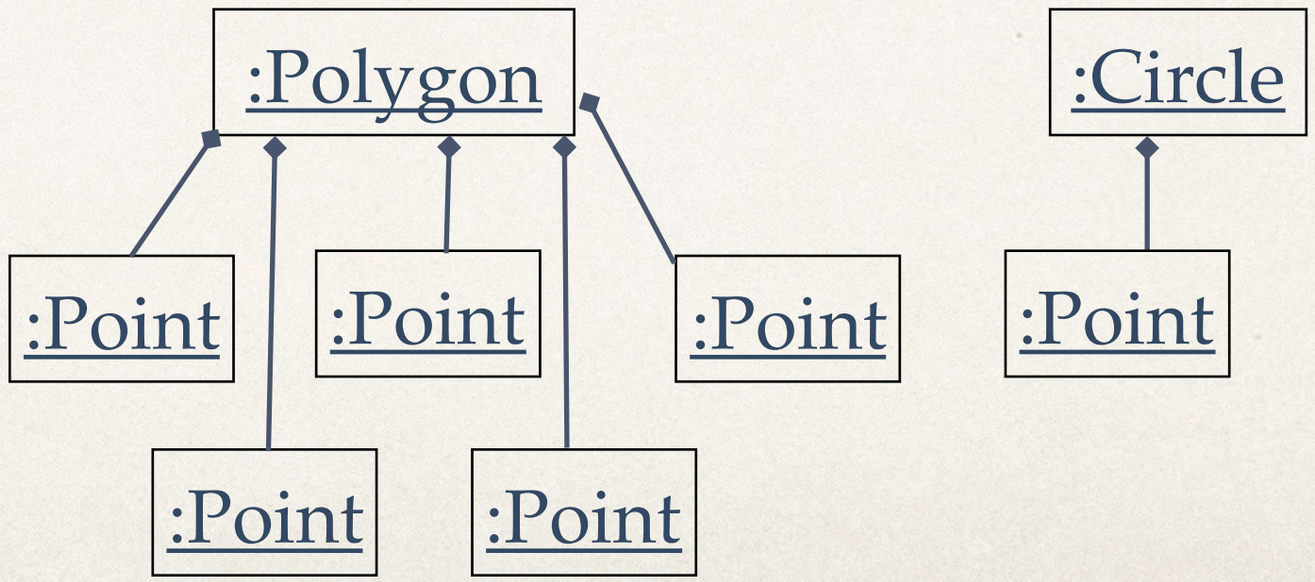




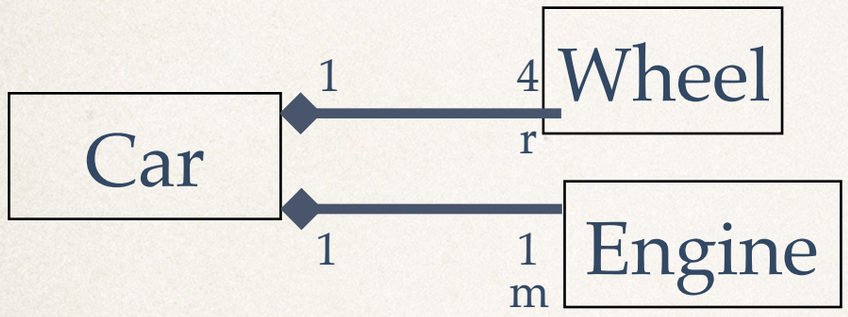
Composition -

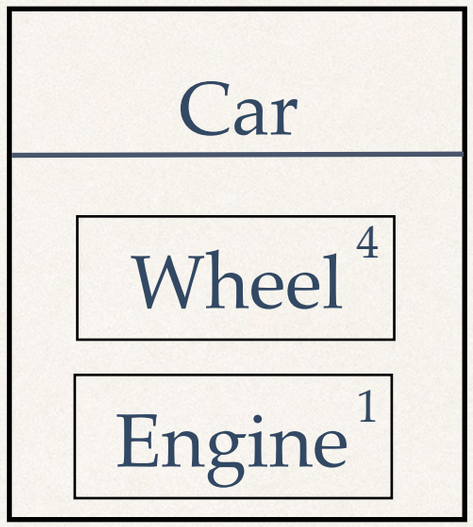


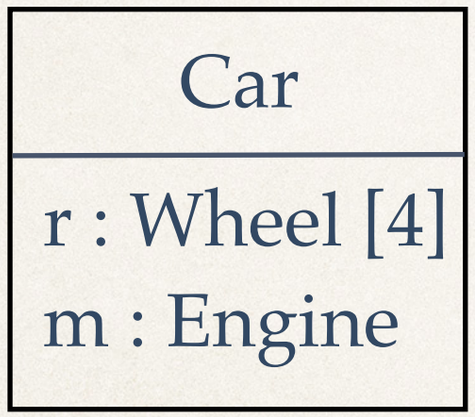




Composition - alternative notations

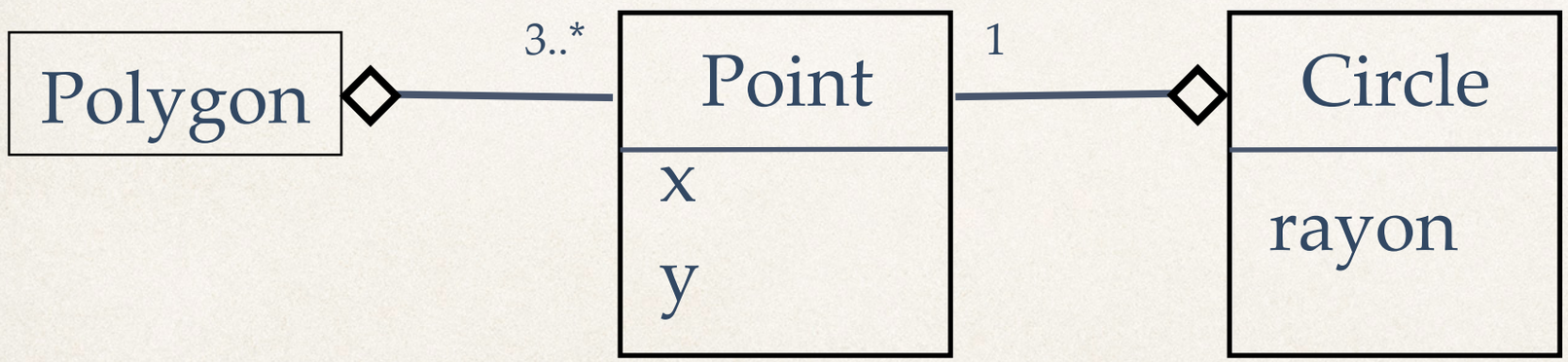




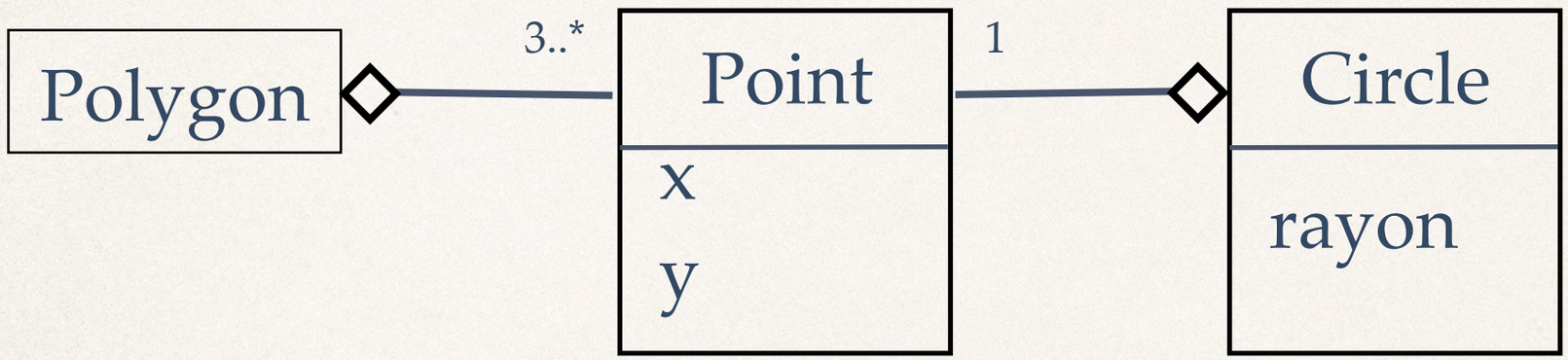


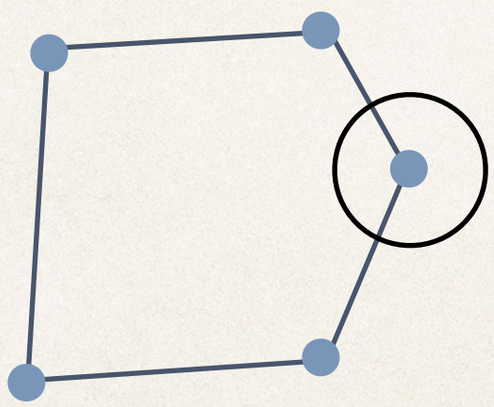
Aggregation

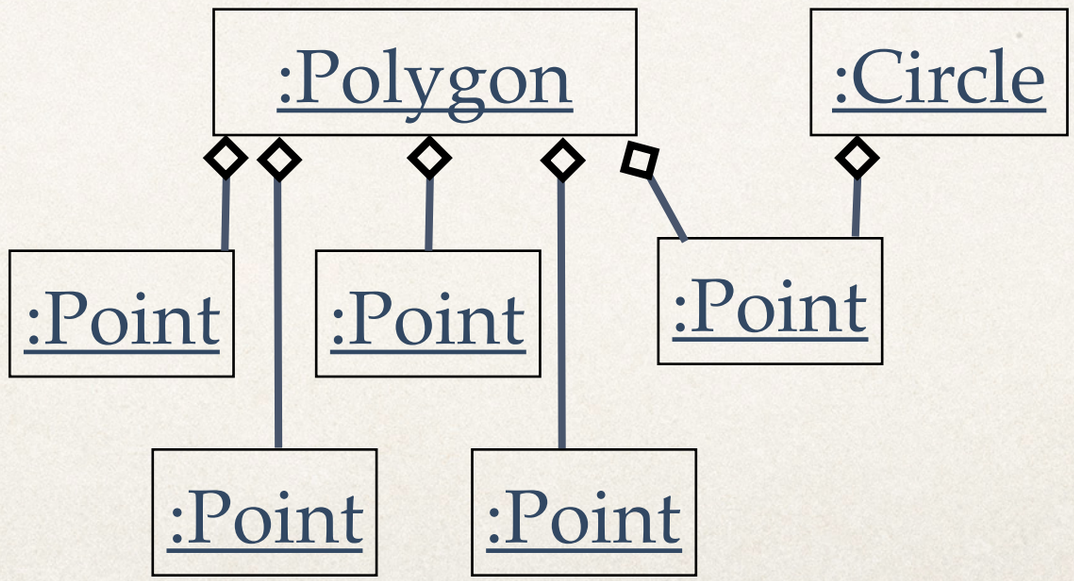
- particular case of an association plus some notions of containment
- no clear semantics
- use carefully (not use it)



Aggregation





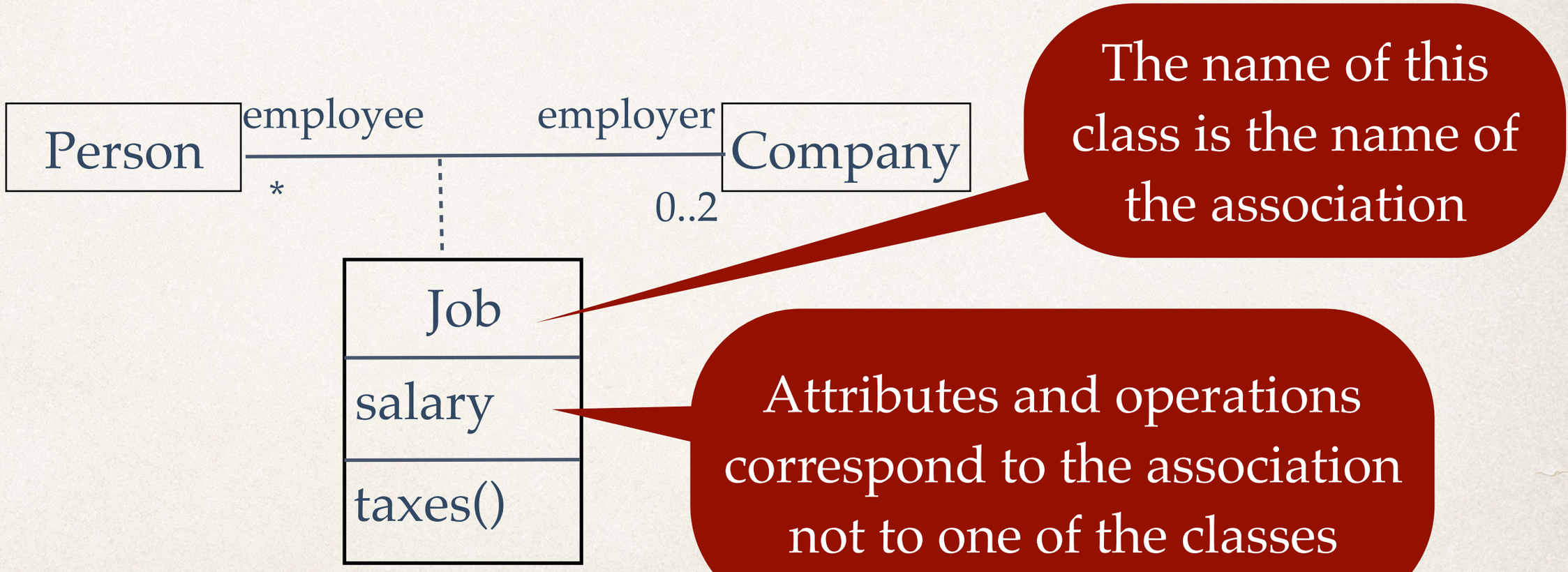


Predefined association constraints

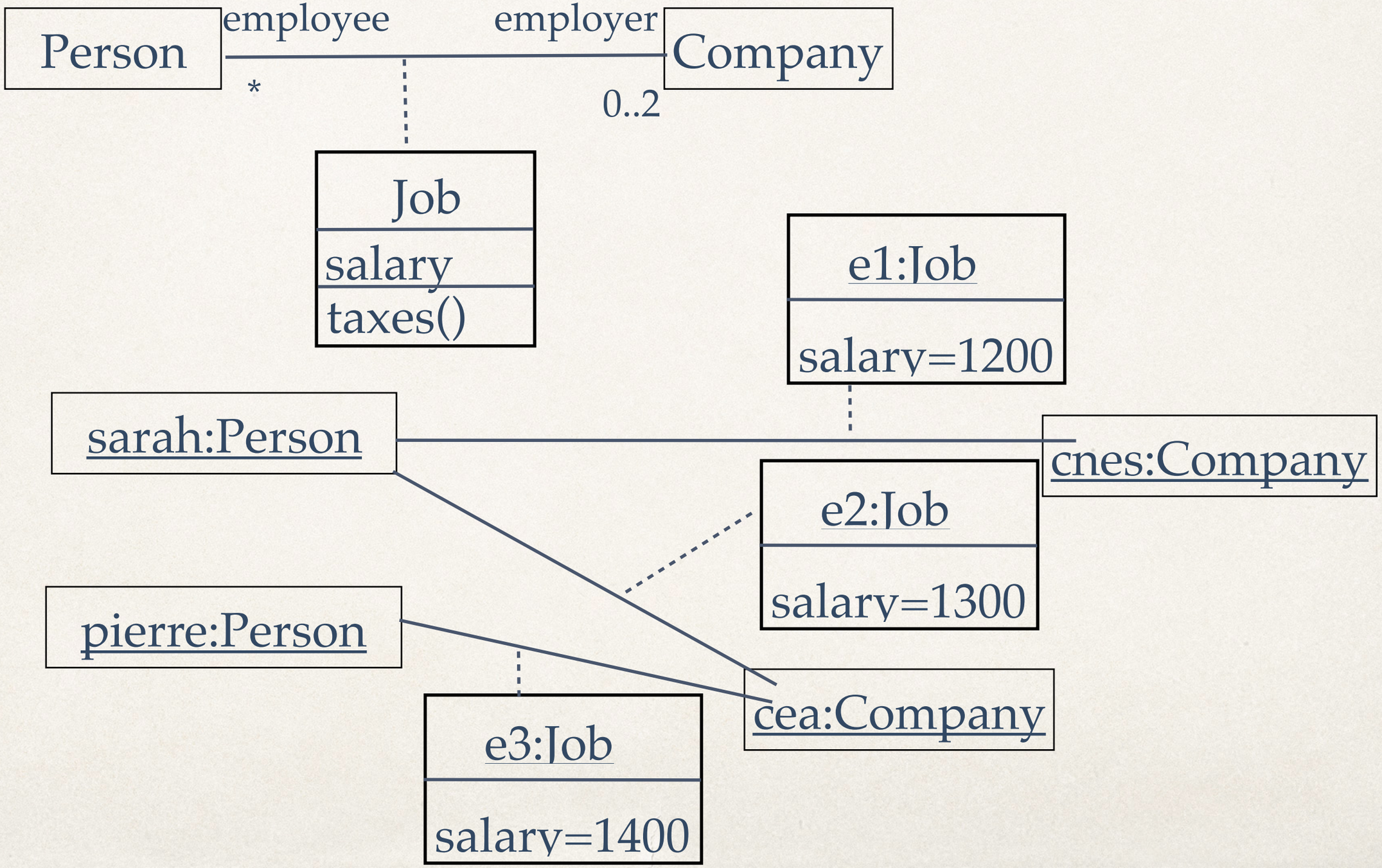
- { frozen } : fixed at the object creation can’t change afterwards
- { ordered } : ordered set (not sorted!!)
- { addOnly } : can’t delete elements from this set

Association class

- Allows to add more information (attributes, associations) to classes

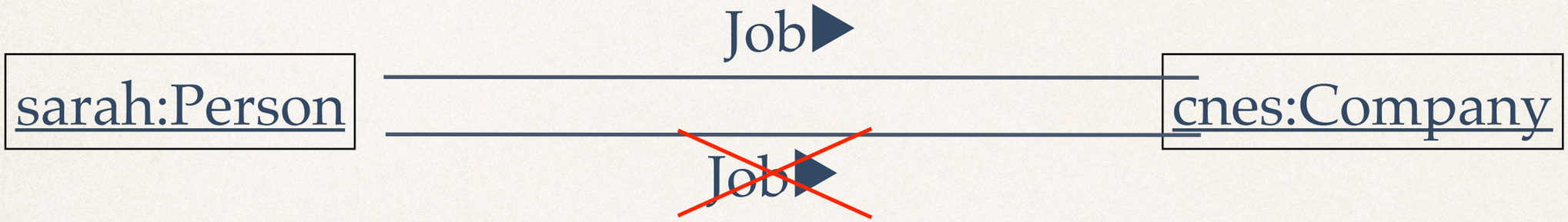


Association classes - example

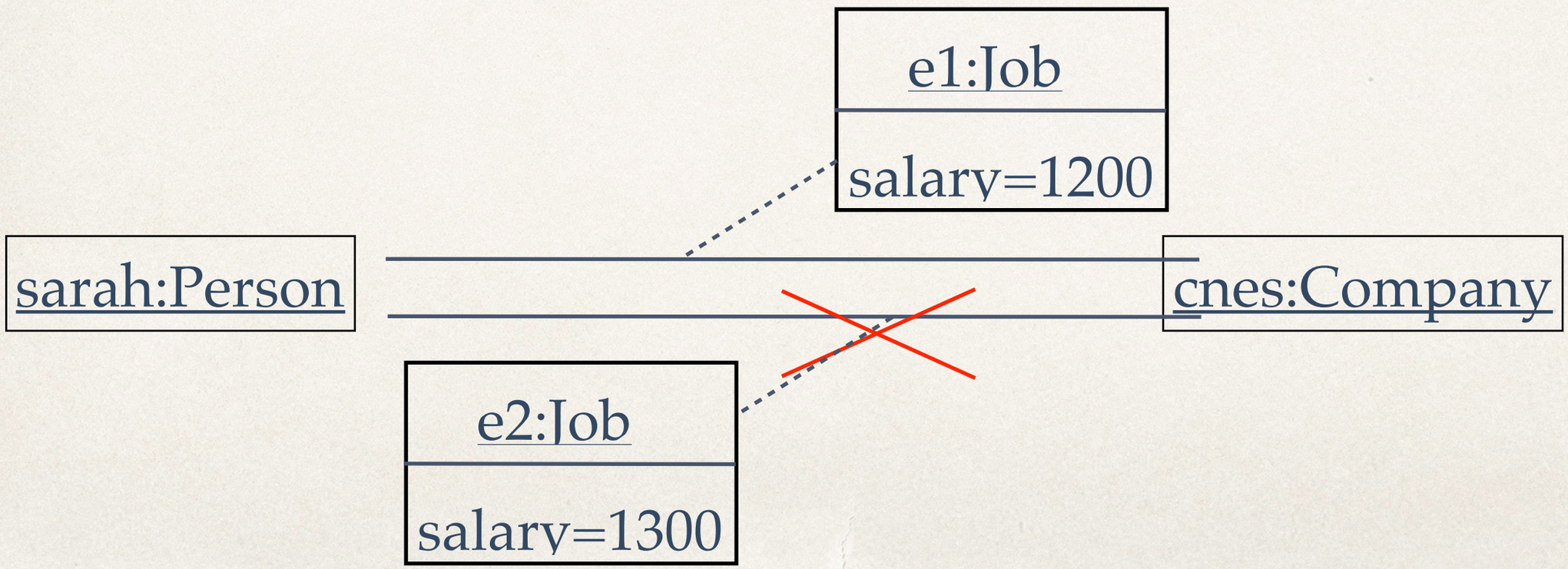


Association classes

- A pair of objects cannot be connected by more than a link of the same kind

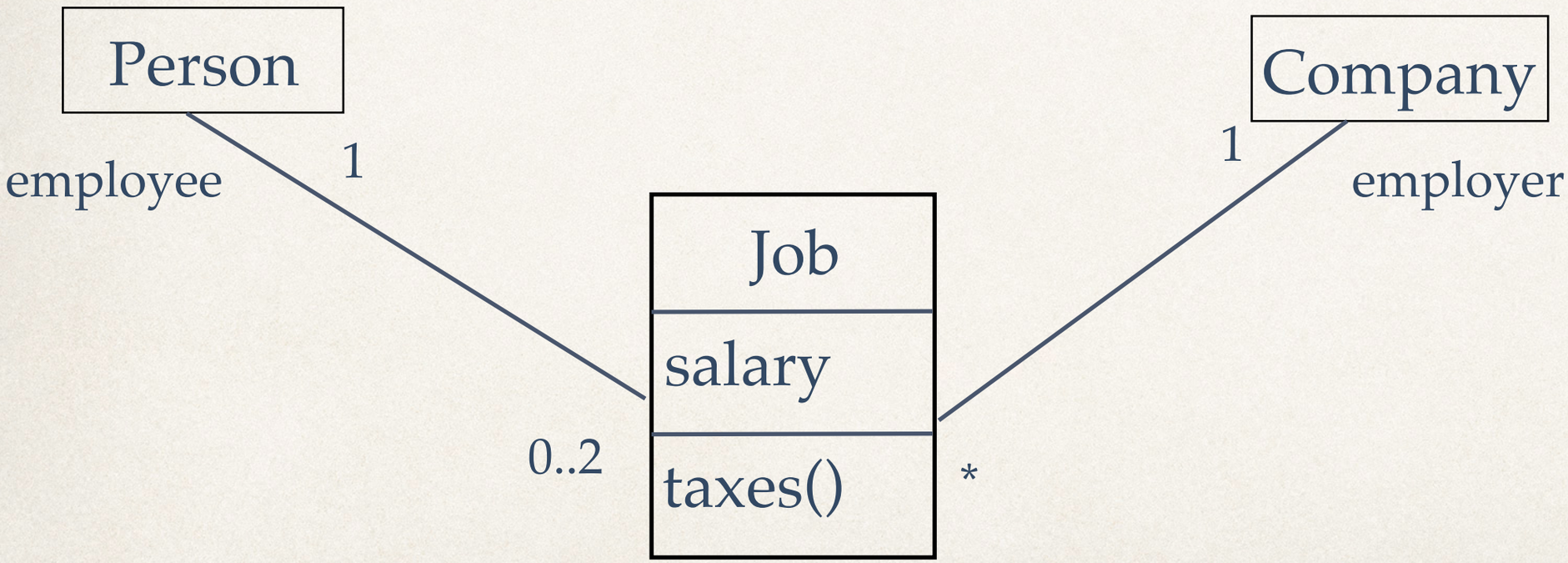


- This constraint holds for associations



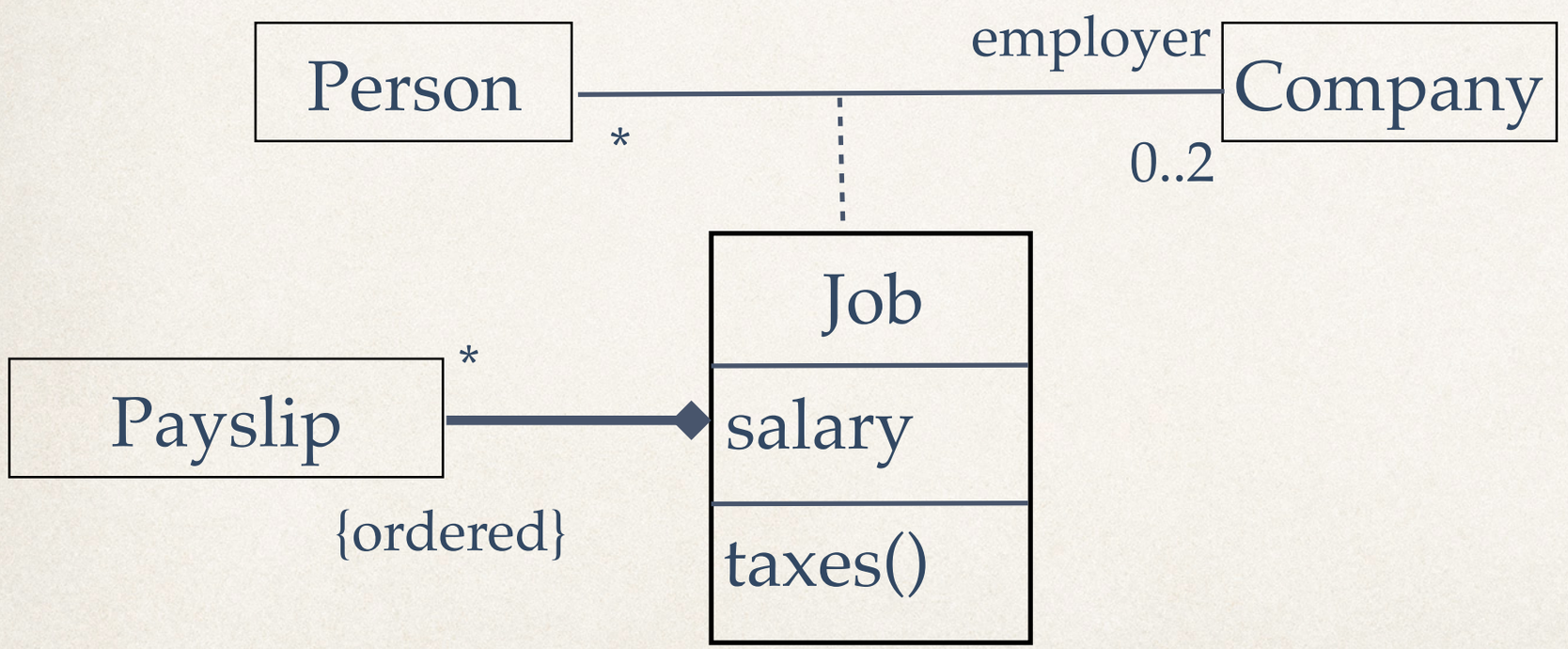
Association classes

- If we need to have several relationships btw the very same instances then we should not use an association class

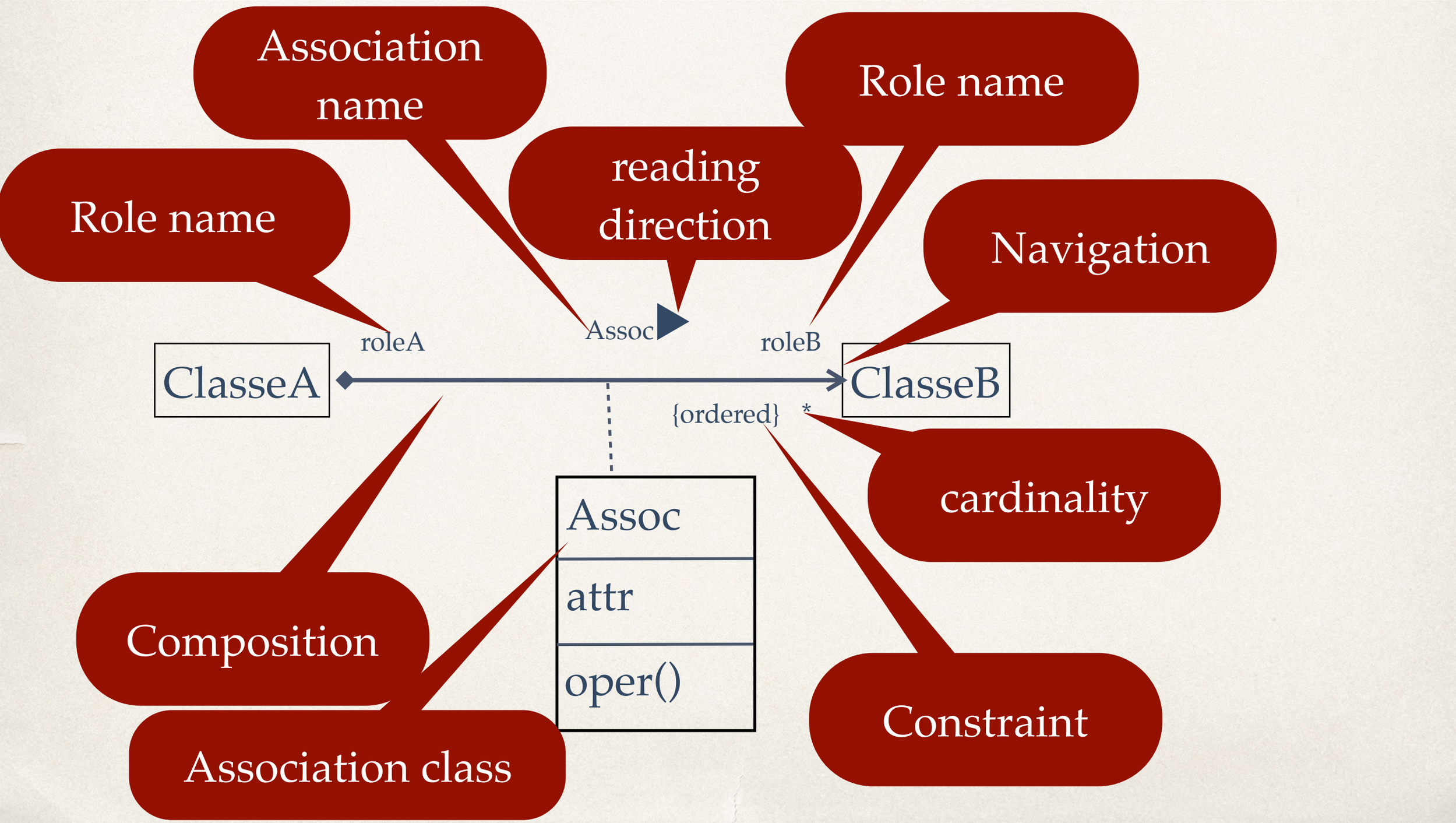


Association classes

- Can be used as regular classes - be part of associations



Associations



Plan

- Notions de base
- Conformité
  diagramme de classes - système modélisé,
  diagramme d'instances - diagramme de classes
- Concepts avancés
- **Usage des diagrammes de classes**

Usage des diagrammes de classes

Syntaxe identique

- En modélisation métier
  Diagramme de classes du domaine

- En conception objet préliminaire
  Diagramme de classes participantes - pour chaque réalisation des cas d'utilisation

- En conception objet détaillée
  Diagramme de classes de la conception

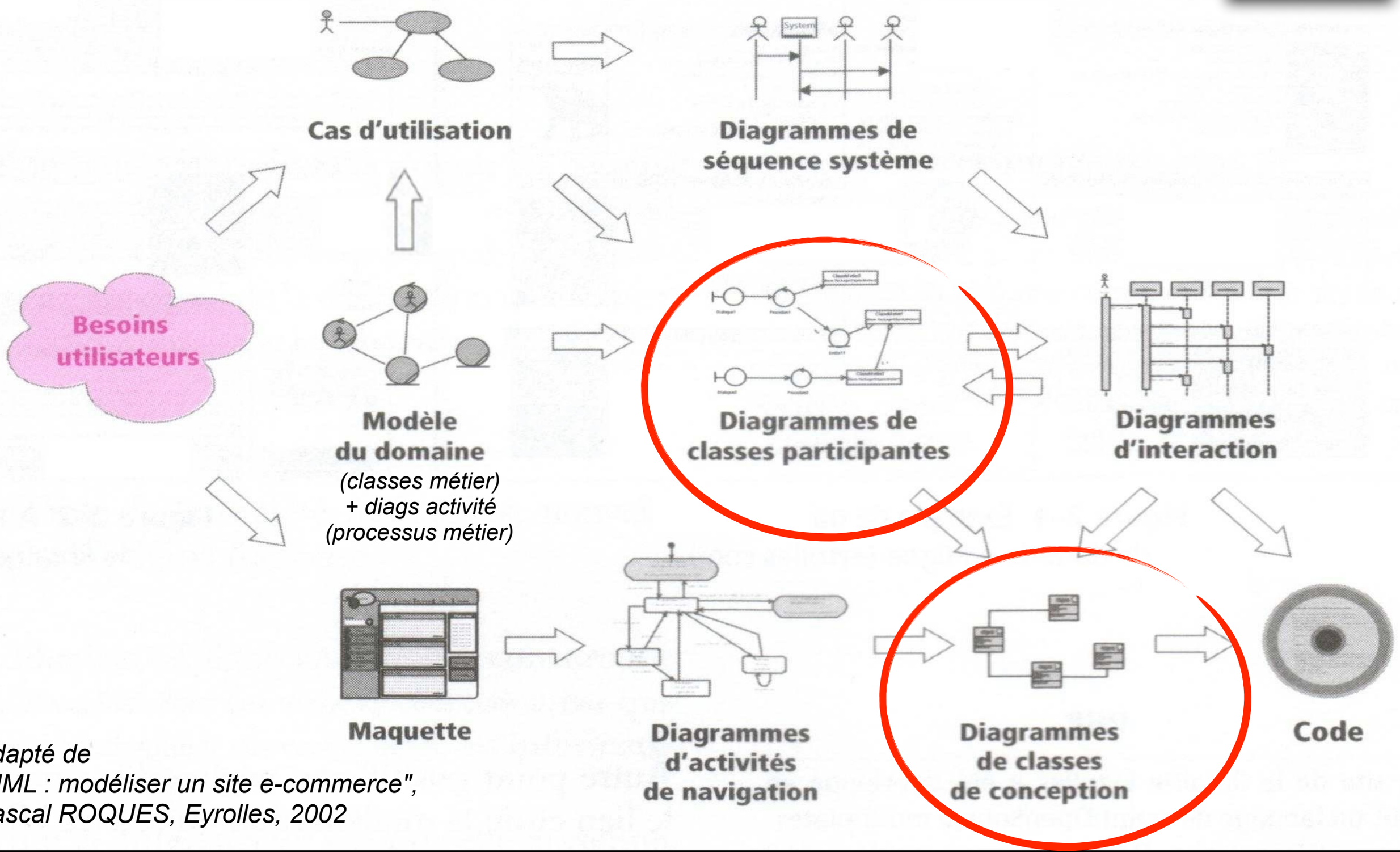
Niveau de détail différent

Modélisation métier

Buts:

- Comprendre la structure statique et dynamique d'une organisation
- S'assurer que les clients, utilisateurs et développeurs ont la même compréhension de l'organisation
- Déduire des besoins système pour supporter l'organisation

Une démarche



Adapté de
"UML : modéliser un site e-commerce",
Pascal ROQUES, Eyrolles, 2002

Transparents élaborés à partir des supports réalisés par

Jean-Marie Favre, Université Joseph Fourier, Grenoble
Henri Massié, Université Paul Sabatier, Toulouse
Joannis Parissis, Université Joseph Fourier, Grenoble