

Conception orientée objet

UML

Michel SALA



Université Montpellier 1

LIRMM

michel.sala@univ-montp1.fr

programme

- I Les méthodes d'analyse**
- II Présentation des modèles OMT-UML**
- III La modélisation objet avec UML**
- IV La vue statique**
- V La vue de conception**
- VI La vue de cas d'utilisation**
- VII La vue dynamique**
- VIII La vue physique**

I Les méthodes d'analyse

1.1 Généralités

1.2 Initiation aux objets

1.2 Les approches existantes

1.3 Motivations des nouvelles approches

1.4 Position des méthodes objets

1.5 Concepts généraux

- 1.1 Généralités
 - Système d 'information
 - Pourquoi une méthode de conception ?

- 1.2 Initiation aux objets

[Rumbaugh et al]

Un **objet** est un concept, une abstraction ou une chose ayant des limites très claires et un sens précis dans le contexte du problème étudié.

Définition des classes

✎ [Masini G. et all]

Une **classe** est la description d'une famille d'objets ayant même structure et même comportement. Elle regroupe un ensemble de données et un ensemble de procédures ou de fonctions.

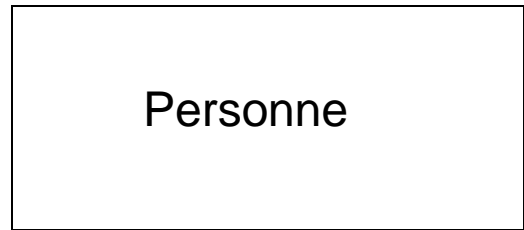
Chaque classe possède une double composante :

- statique : les données qui sont des champs nommés possédant une valeur. Les champs caractérisent l'état des objets pendant le déroulement du programme.
- dynamique : les procédure ou méthodes, qui représentent le comportement commun des objets appartenant à la classe. Les méthodes manipulent les champs des objets et caractérisent les actions pouvant être effectuées sur ces objets.

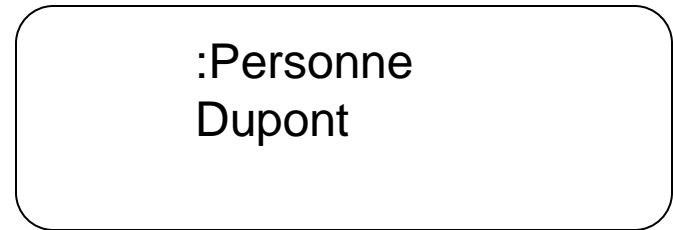
Définition d'une instance

∞ [Masini G. et all]

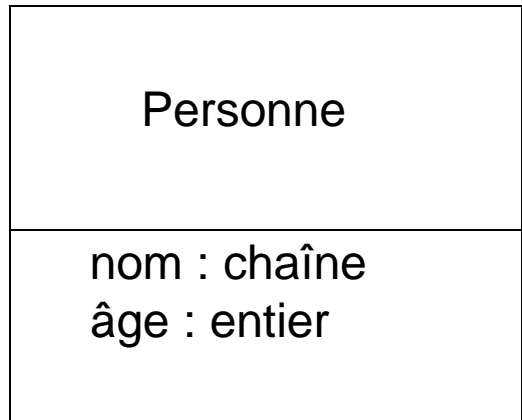
La classe est un modèle abstrait qui décrit des objets. Les représentants physiques d'une classe sont les *instances*. Une instance est donc un objet particulier qui est créé en respectant le "plan" donné par la classe. La classe joue donc le rôle de moule pour créer autant d'exemplaires que nécessaire.



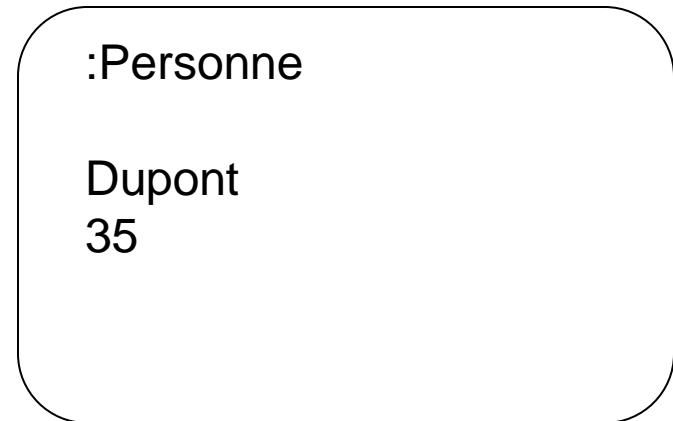
Classe



Objet
Instance d'une classe



Classe avec attributs

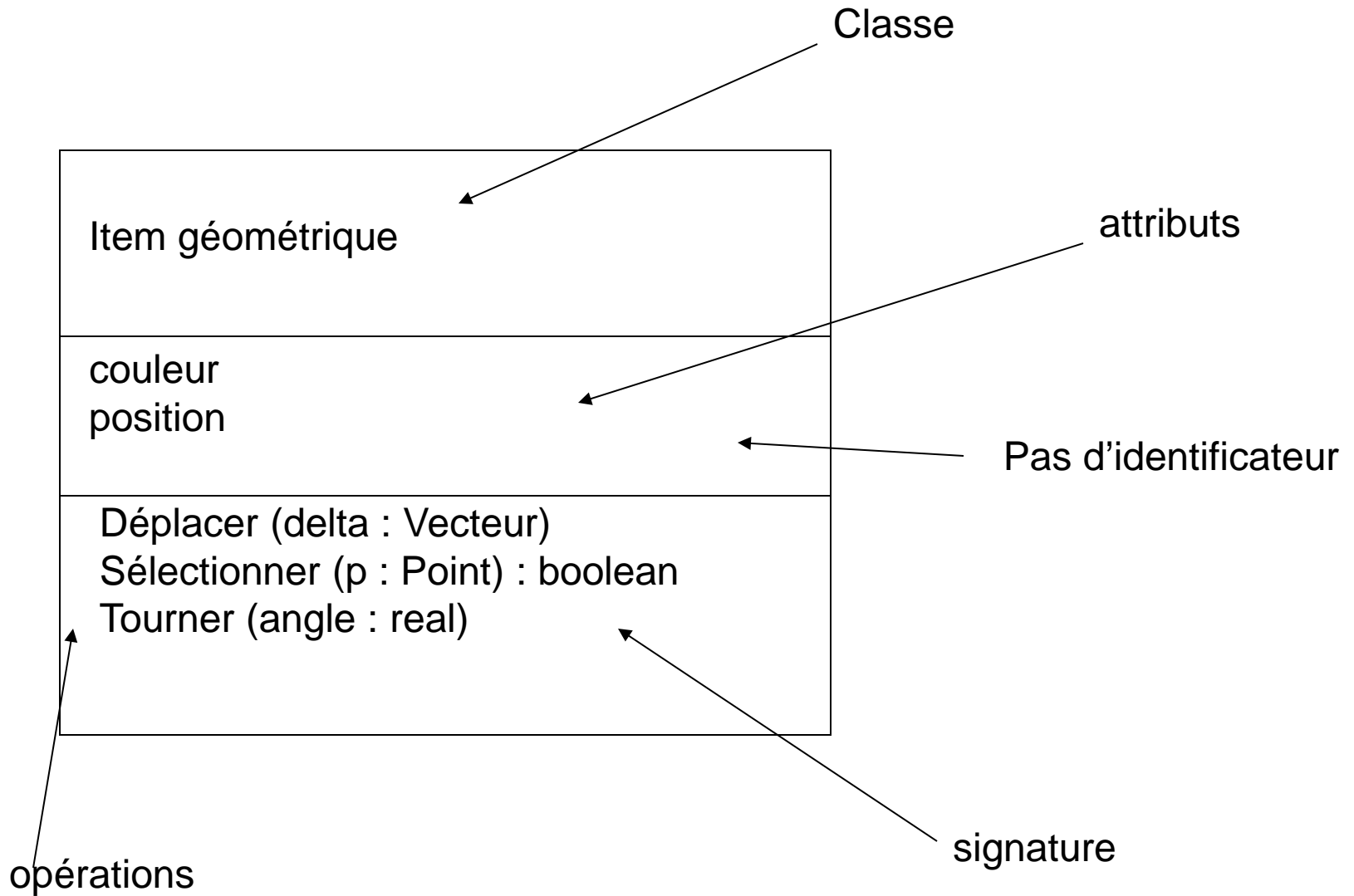


Objet avec valeurs

Définition d'une opération

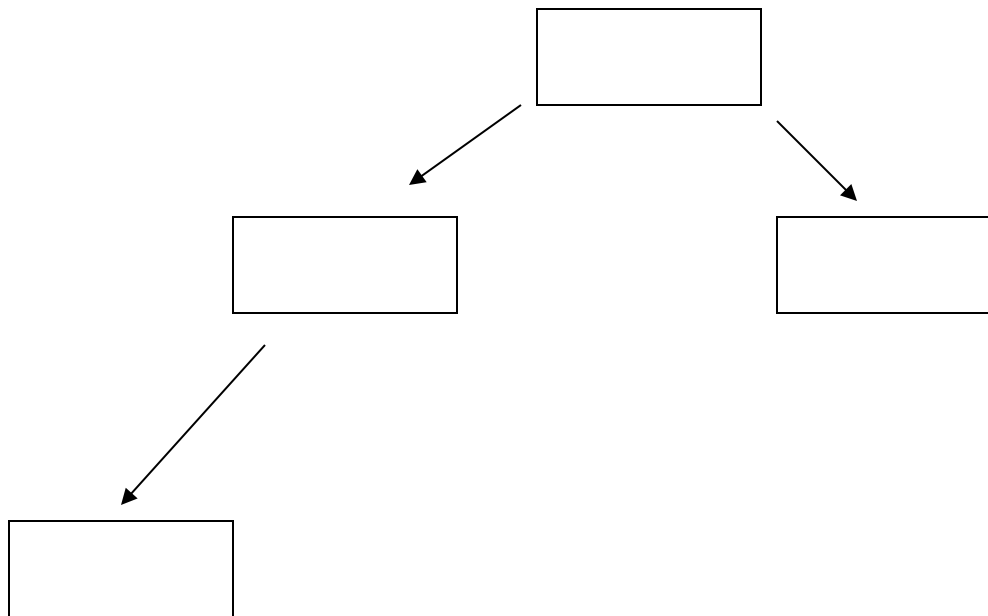
☞ [Rumbaugh et al]

- Une **opération** est une fonction ou une transformation qui peut être appliquée aux objets ou par les objets dans une classe.
- Une **méthode** est l'implémentation d'une opération dans une classe.



Héritage

Mécanisme de transmission de propriétés d'une classe vers une sous-classe



Notions de :

- super classe
- classe
- sous-classe
- classe abstraite
- héritage multiple

Définition de polymorphisme

- Une opération est polymorphe quand elle peut prendre différentes formes dans des classes différentes.

- 1.3 Motivations des nouvelles approches
 - Forte interactivité avec les utilisateurs
 - Manipulation d 'objets complexes
 - Evolution fréquentes d 'objets

- 1.4 Les approches existantes
 - 1.4.1 Les approches cartésiennes
 - Proviennent de la programmation structurée
 - SADT

- 1.4.2 Les approches systémiques
 - Théorie systémique des organisations
 - MERISE (Tardieu 83)
 - AXIAL (Pellaumail 86)

- 1.3.3 Les approches objets
 - Evolution des approches systémiques
 - OOD
 - HOOD
 - OOA
 - OOSE

OOD - Object Oriented Design (Grady Booch)

- modèle précurseur
- défini par le DOD pour rationaliser les développements Ada
- basé sur le modèle statique
- préconisait SADT
- introduit le concept de paquetage

- pas de décomposition des objets
- pas de type pour les attributs
- représentation graphique peu conviviale
- pas de prise en compte des SGBD OO

HOOD - Hierarchical Object Oriented Design

- appel d 'offre de l 'ESA
- étend OOD à du temps réel
- basé uniquement sur le modèle statique
- capacité de représenter des objets complexes
- pas de relation de généralisation
- pas de prise en compte des SGBD OO

OOA - Object Oriented Analysis

- initialement conçu pour le temps réel
- utilise le schéma relationnel étendu
- étude détaillée du cycle de vie des objets
- état la notion relationnelle
 - identificateur lié à la valeur
 - pas d 'opérations attachées aux objets
 - pas d 'encapsulation

OOSE - Object Oriented Software Engineering (Ivar Jacobson)

- méthode suédoise, Ericsson
- inclus avec les objets et leur comportement une description des interfaces utilisateur
- les règles de construction du niveau logique ne sont pas claires

- 1.5 Positionnement

- 1.5.1 Approche objet vs approche cartésienne

- Danger de la séparation des données et des traitements
- Identification des briques de base
- Le SI est un objet complexe
- Le SI interagit avec son environnement
- Approche sur les objets et leur comportement

- 1.5.2 Approche cartésienne vs approche objet
 - Répond plus directement aux besoins
 - Approche fonctionnelle plus intuitive
 - Identification des briques de base
 - Le SI est un tout cohérent

- 1.6 Concepts généraux des méthodes objet
 - 1.6.1 Les niveaux de conception
 - Avantages de l'approche objet
 - regrouper l'analyse des données et des traitements
 - simplifier les transformations du niveau conceptuel au niveau physique

I Les méthodes d'analyse

- Hiérarchie des niveaux de conception

Niveau externe	Visibilité des objets par rapport à chaque classe d'utilisateurs
Niveau conceptuel	Niveau global intégrant l'ensemble des vues
Niveau logique	Raffinage du niveau conceptuel
Niveau physique	Raffinage du niveau logique

- 1.6.2 Etape du cycle de développement
 - Modèle en cascade
 - Modèle en V
 - Modèle en W
 - Modèle en escargot
 - Modèle tridimensionnel

I Les méthodes d'analyse

	OOD	HOOD	OOA	OMT	OOSE	OOM
Planification						oui
Besoins					oui	oui
Spécif formelles		oui	oui	oui	oui	oui
Spécif techniques		oui	oui	oui	oui	oui
Implémentations	oui	oui		oui	oui	oui
Intégration et test		oui	oui	oui		oui
Validation/Maintenance	oui				oui	

- 1.6.3 Dimensions

- Statique décrit les objets du système
- Dynamique comment les objets évoluent
- Fonctionnelle flux d'informations

	OOD	HOOD	OOA	OMT	OOSE	OOM
Statique	oui	oui	oui	oui	oui	oui
Dynamique	oui		oui	oui	oui	oui
Fonctionnelle			oui	oui		oui

programme

II De OMT à UML

2.1 Historique

2.2 La notion orienté objet

2.3 La méthodologie OMT

2.4 Les modèles

2.5 Arrivée d'UML

- **2.1 Historique de OMT**

- inventé par le Centre de Recherche et de Développement de General Electric
- ouvrage de Rumbaugh 1991
- succès en France et en Europe
- fusionne avec la méthode OOD de Booch

- **2.2 La notion d 'orienté objet**

identité

données dans une entité appelé objet

classification

même structure de données

même comportement

—————→ classe

un objet est une instance de classe

polymorphisme

une même opération se comporte différemment sur différentes classes

augmentation de la généricité du code

héritage

partage des attributs et des opérations

encapsulation

définir une interface cachant l'implémentation
garantir l'intégrité des données

Un objet :

- une entité précise qui possède un nom
- un ensemble d'attributs qui la caractérise
- un ensemble d'opérations qui définit son comportement
- un objet est une instance d'une classe

Une classe :

- un type de données abstrait
- un ensemble de propriétés qui la caractérise
 - attributs
 - méthodes

- **2.3 La méthodologie OMT**

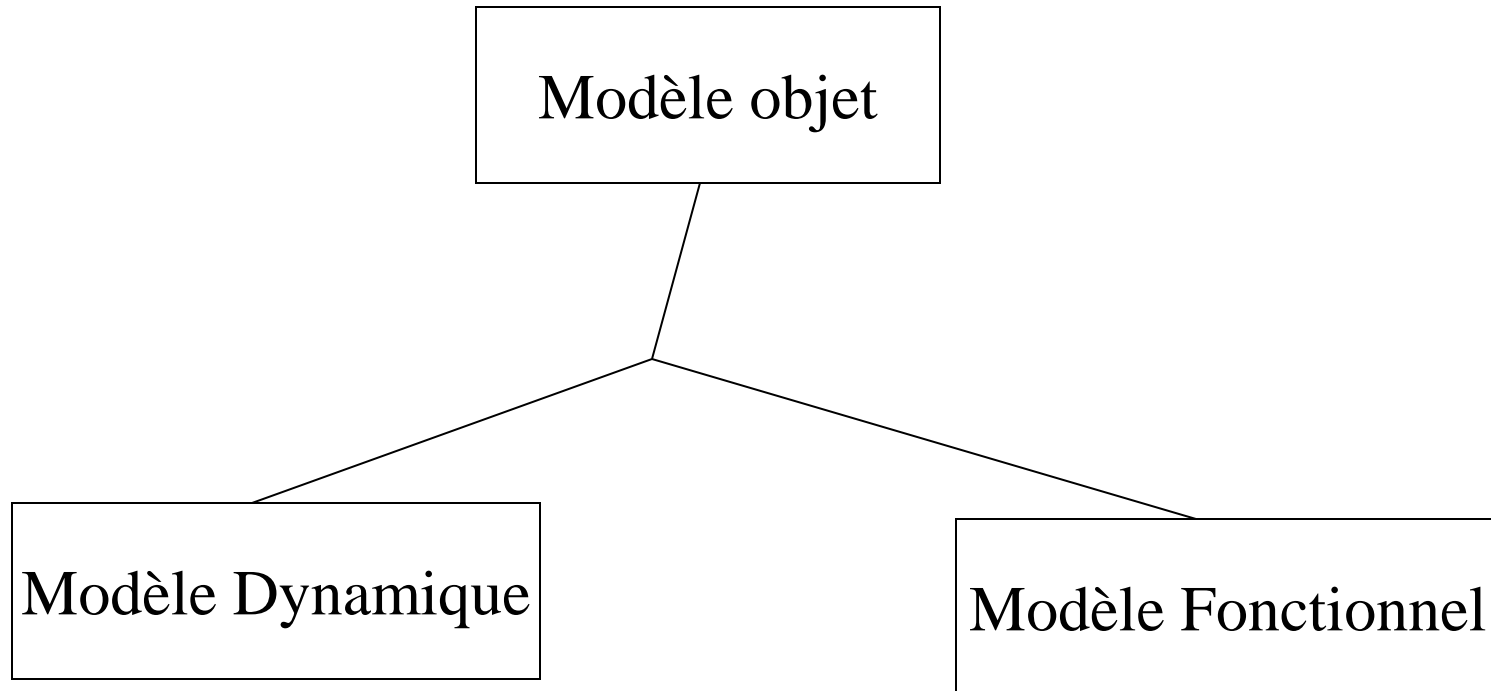
- *Analyse ou spécification*
 - CE QUE le système doit faire
- *Conception du système*
 - L 'architecture générale du logiciel
- *Conception des objets*
- *Implémentation*

- **2.4 Les modèles de OMT**

- Objet
- Fonctionnel
- Dynamique

**Ces 3 modèles
expriment des
points de vue
différents sur le
système**

Types d 'objets et
leurs relations



Stimuli des objets et
leurs réponses

Calculs effectués
par les objets

Types d 'objets et
leurs relations

Modèle objet

Décrit les structures de données sur lesquelles
les modèles dynamique et fonctionnel opèrent

Décrit les changements dans le temps et les aspects de contrôle du système

Modèle Dynamique

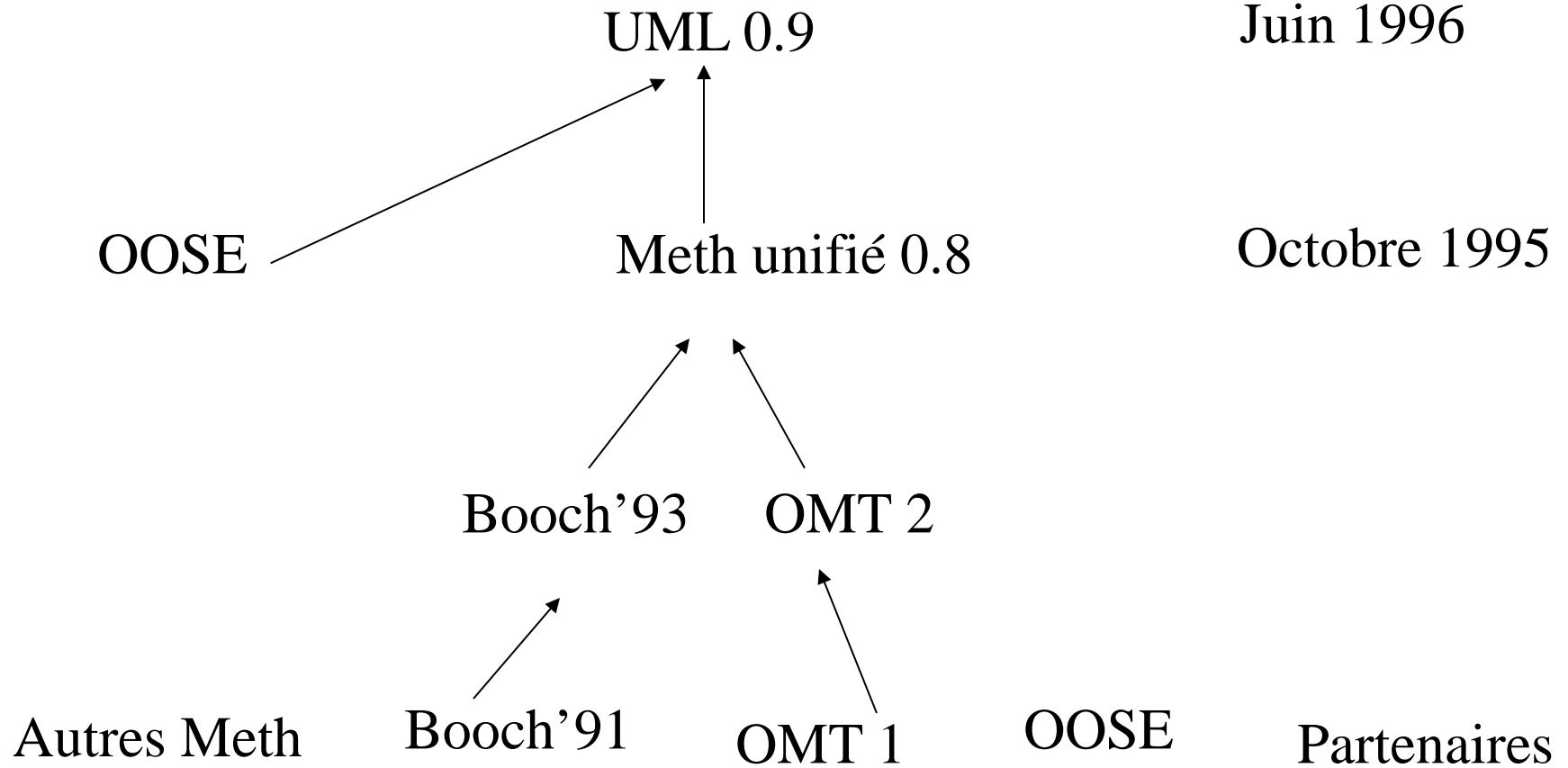
Stimuli des objets et leurs réponses

Décrit les transformations des données dans le système

Modèle Fonctionnel

Calculs effectués
par les objets

• 2.5 Arrivée d'UML



UML 2.0



UML 1.3



UML 1.2



UML 1.1



UML 1.0

Début 2005

Juin 1999

Juin 1998

Septembre 1997

Janvier 1997

Fin 97 UML devenu norme OMG (Object Management Group)

OMG (www.omg.org):



- créé en 89 à l'initiative d'entreprises
- CORBA (Common Object Request Broker Architecture)

- **UML 1.3 définit 9 types de diagrammes**
 - Vues statiques
 - Les diagrammes de classe**
 - Les diagrammes d'objet**
 - Les diagrammes de cas d'utilisation**
 - Les diagrammes de composants**
 - Les diagrammes de déploiement**
 - Vues dynamiques
 - Les diagrammes de séquence**
 - Les diagrammes de collaboration**
 - Les diagrammes d'états-transitions**
 - Les diagrammes d'activités**

- **UML 2.0 définit 4 domaines**
 - **Structurel**
 - Vue statique
 - Vue de conception
 - Vue de cas d'utilisation
 - **Dynamique**
 - Vue de machine d'états
 - Vue d'activité
 - Vue ensemble des interactions
 - **Physique**
 - Vue de déploiement
 - **Gestion du modèle**
 - Vue de gestion du modèle
 - Profil

Domaine	Vue	Diagramme
Structurel	Vue statique	Diagramme de classes
	Vue de conception	Structure interne
		Diagramme de collaboration
		Diagramme de composants
	Vue de cas d'utilisation	Diagramme de cas d'utilisation
Dynamique	Vue de machine d'états	Diagramme de machines d'états
	Vue d'activité	Diagramme d'activités
	Vue ens. des interactions	Diagramme de séquence
		Diagramme de communication
Physique	Vue de déploiement	Diagramme de déploiement
Gestion du modèle		
	Vue de gestion du modèle	Diagramme de package
	Profil	„

programme

III La modélisation objet avec UML

3.1 Présentation

3.2 Classes

3.3 Relations

- Introduction

- Caractéristiques de l'approche OO
 - Dotés d'un identifiant
 - Agissent en déclenchant leurs propres opérations
 - Etat à un instant donné
- Extension du modèle E/A
- Modèle près de celui de Merise

- 3.1 Généralités

- UML : Unified Modeling Language
- Unification des travaux de :
 - Rumbaugh (OMT)
 - Booch (Booch 93)
 - Jacobson (Use Case)

- Objectifs
 - Représentation objet
 - Smalltalk 1976
 - Compilateur C++ 1980
 - Langage de modélisation

- Les éléments clés
 - + langage précis
 - + compréhension de représentations abstraites
- période d'adaptation
- pas de processus d'élaboration de modèle

UML support de communication

- notation graphique
- aspect formel limite les erreurs sémantiques
- indépendance par rapport aux langages
- chaque diagramme possède une sémantique du même problème

3.2 Les classes

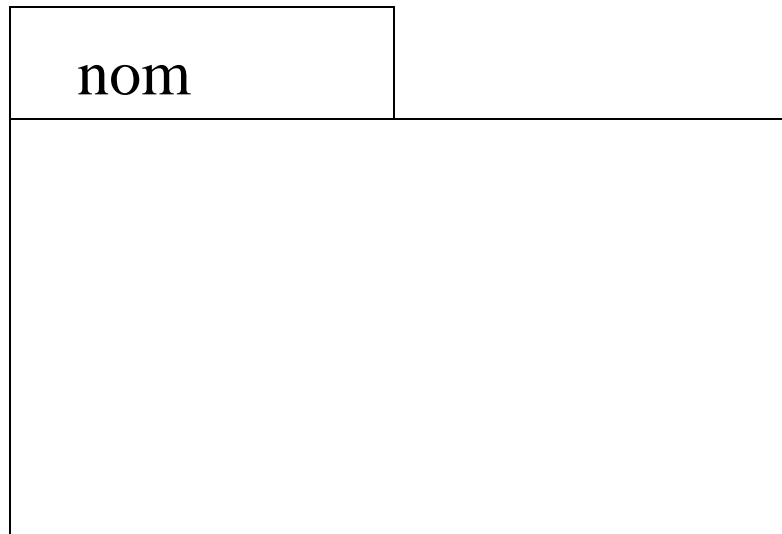
3.2.1 Les classeurs

3.2.2 Les classes

3.2.3 Les attributs

- 3.2.1 Les paquetages
 - Permet de regrouper et de d'isoler
 - Des classes
 - Des associations
 - Des paquetages

- notation



- 3.2.2 Les classes
 - Composées de trois parties
 - Le nom
 - Les attributs
 - Les opérations

Les parties non pertinentes peuvent ne pas être représentées

- Présentation graphique

Nom de la classe
Attribut1 : type1 = valeur-initiale1 Attribut2 : type2 = valeur-initiale2 ...
Opération1():Nom-classe Opération2(param1:type,...) Suppression()

- 3.2.3 Les attributs

- Visibilité

+	public	visible par tous
#	protégé	visible par les classes et amies
-	privé	visible par les classes amies

Dérivé / attribut provenant d'un autre attribut ou
d'une combinaison d'attributs

- Attribut
 - Valeur pour chaque instance d'objet
 - Domaine atomique
 - Nom d'attribut unique à l'intérieur de la classe
 - Pas d'attribut identifiant

- Exemple

Rectangle
+Longueur : entier
+Largeur : entier
+ /surface : entier

/surface = longueur X largeur

- Opération
 - Fonction appliquée aux objets de la classe
 - Peut être polymorphique
- Méthode
 - Implémentation d'une opération pour une classe

3.3 Les relations

3.3.1 Les associations

3.3.2 Les cardinalités

3.3.3 Les contraintes

3.3.4 Les agrégations

3.3.5 Les généralisations

3.3.6 Exemple de diagrammes statiques

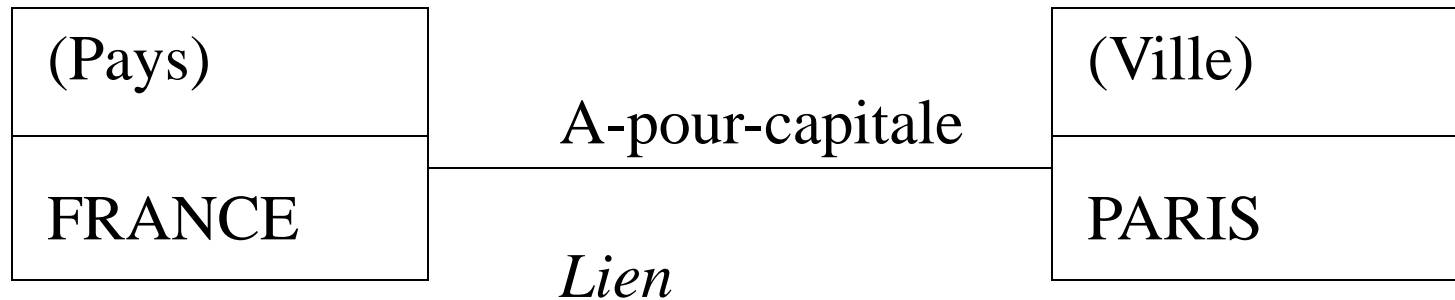
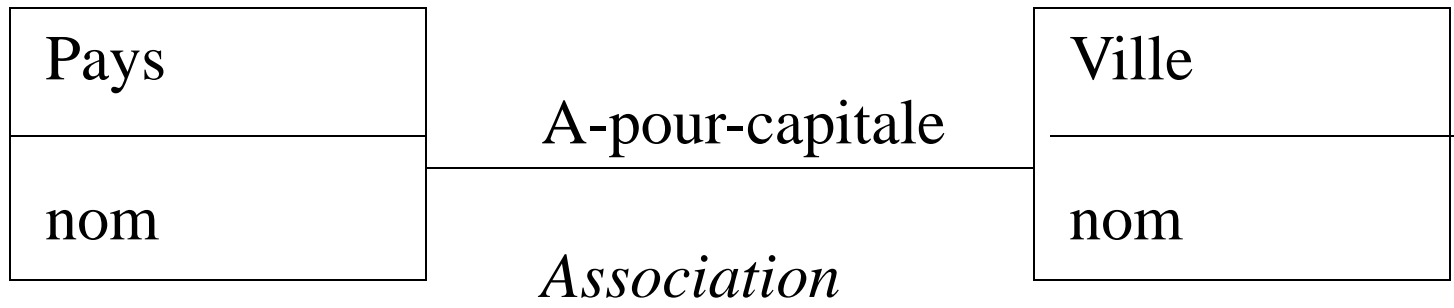
- 3.3.1 Les associations

- Lien

- Décrit une relation entre objets existant dans le réel
- Défini comme un tuple (identifiants des objets liés)

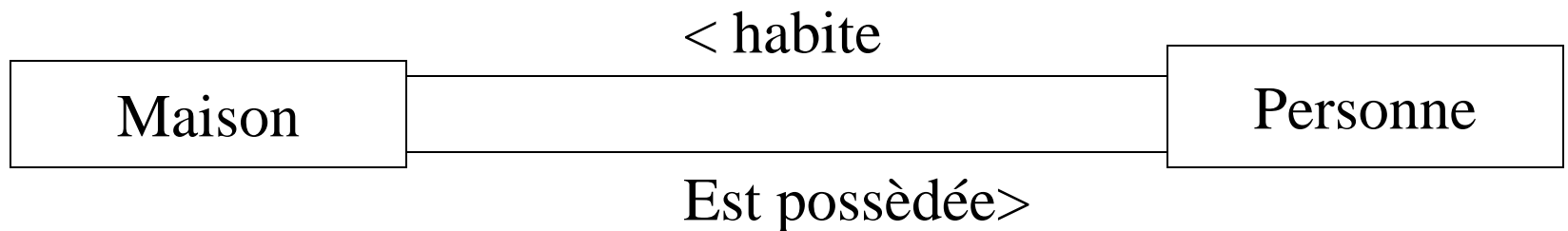
- Association

- Décrit un ensemble de liens similaires

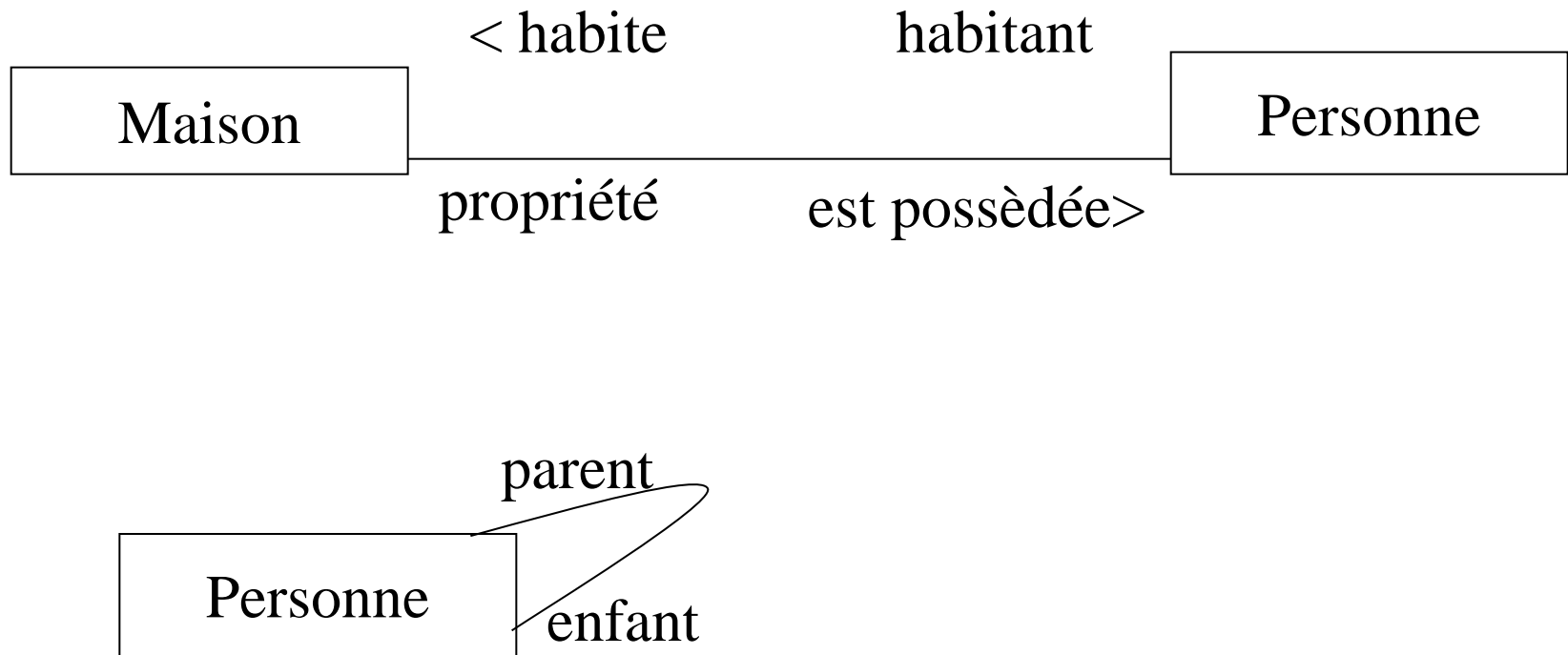


- Une association

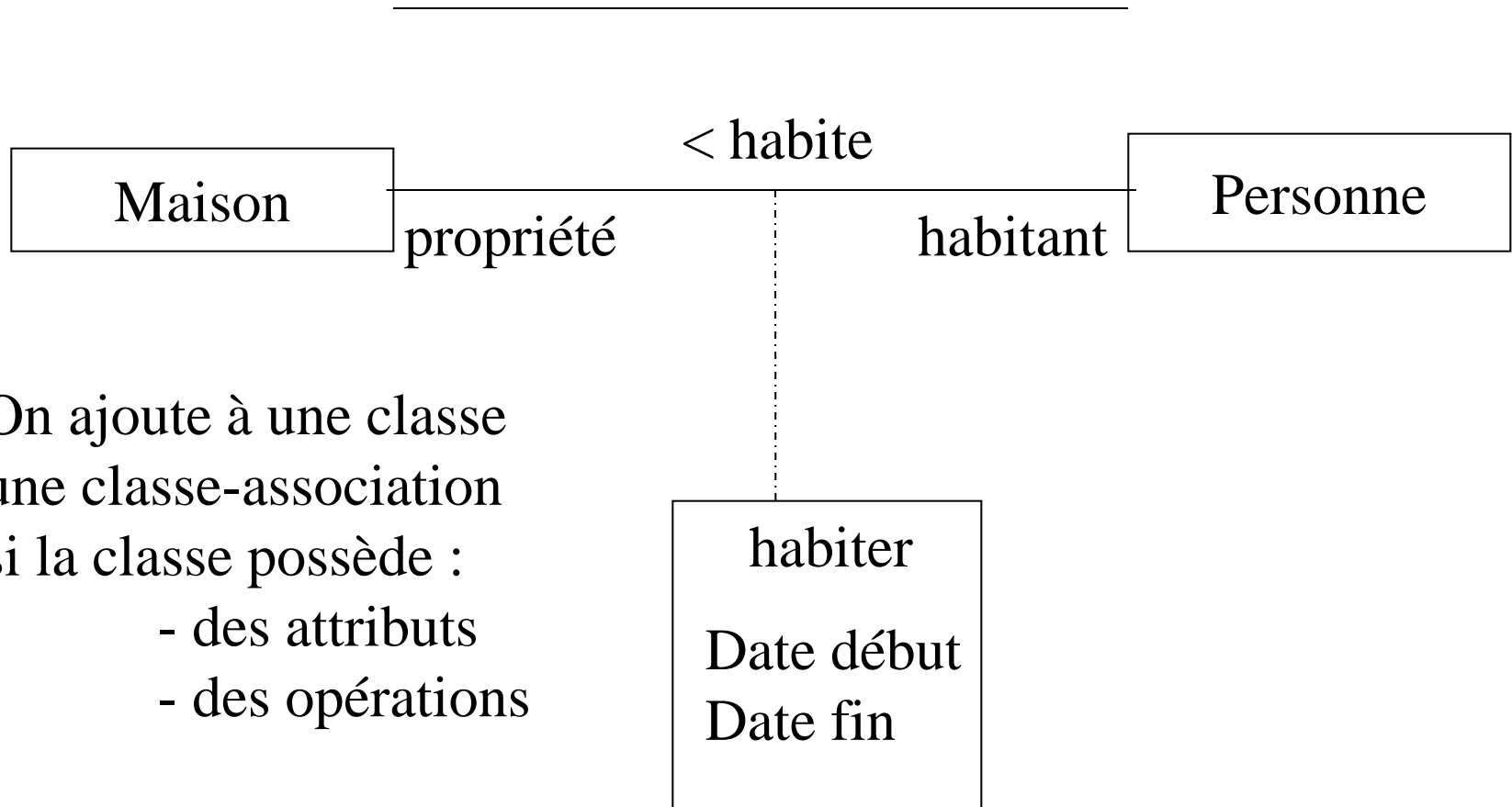
- Représente une relation structurale
- Le sens de lecture est indiqué par un petit triangle
- Peut-être récursive



- Rôle



- **Classe-association**



On ajoute à une classe
une classe-association
si la classe possède :

- des attributs
- des opérations

● 3.3.2 Les cardinalités

Indique le nombre minimal et maximal d'objets de la classe :

1 un et un seul

0..1 zéro ou 1

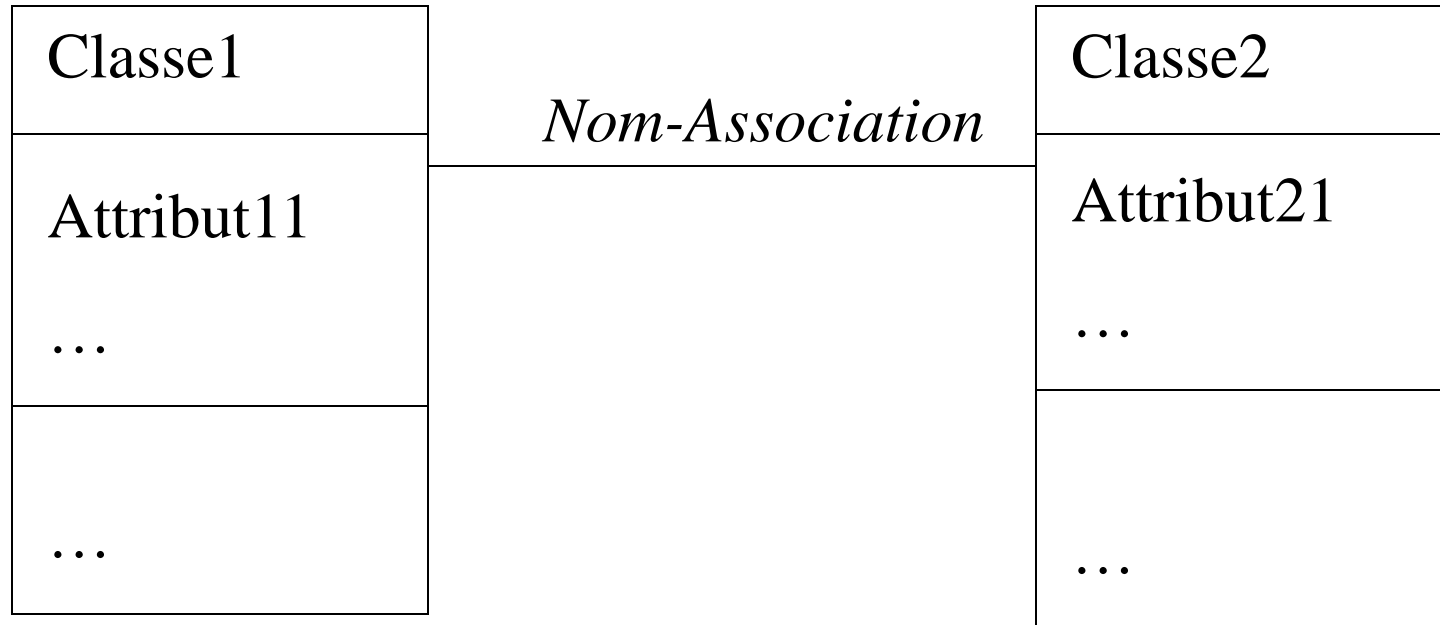
M..N de M à N (entiers naturels)

* de zéro à plusieurs

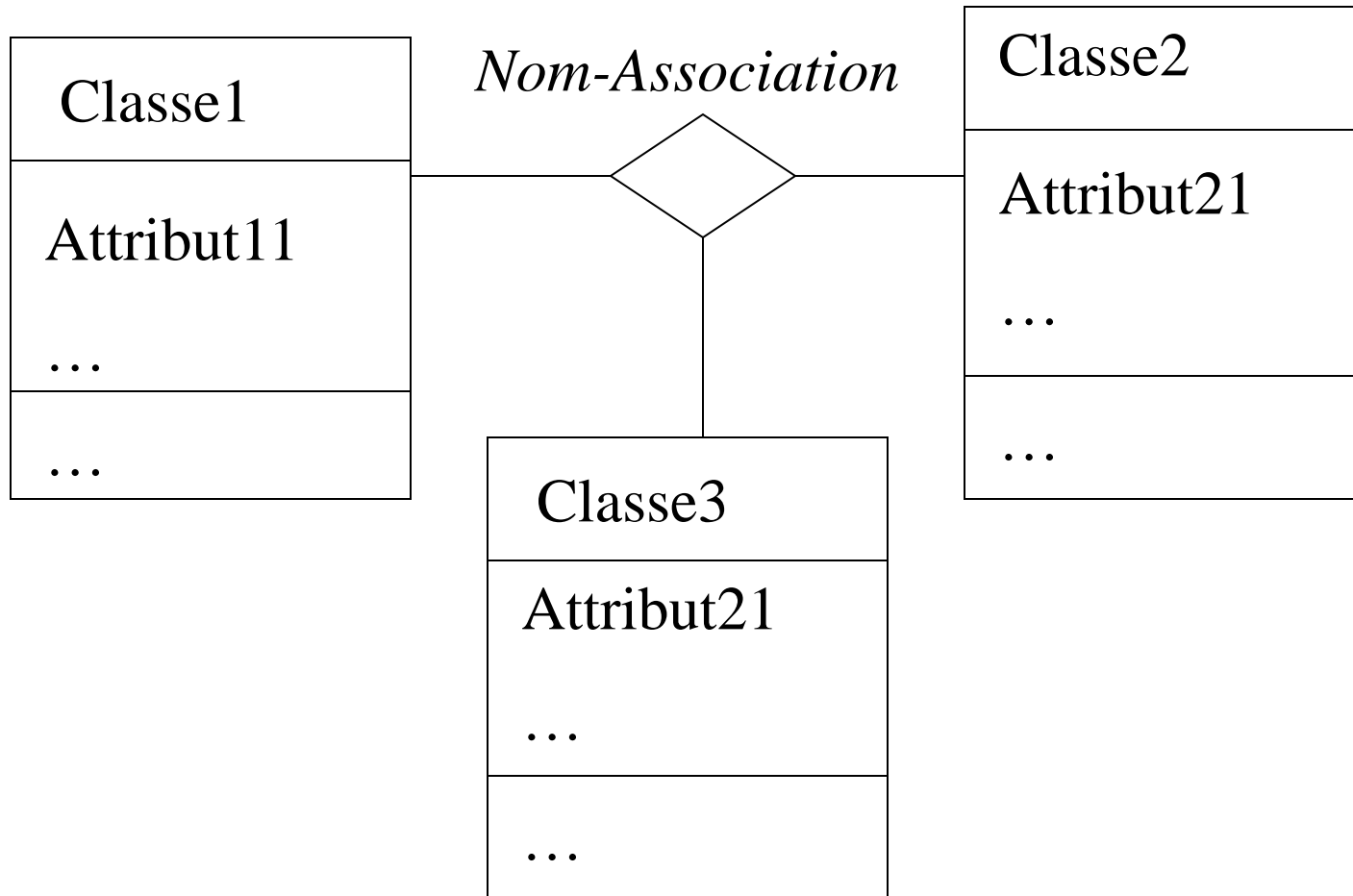
0..* de zéro à plusieurs

1..* de un à plusieurs

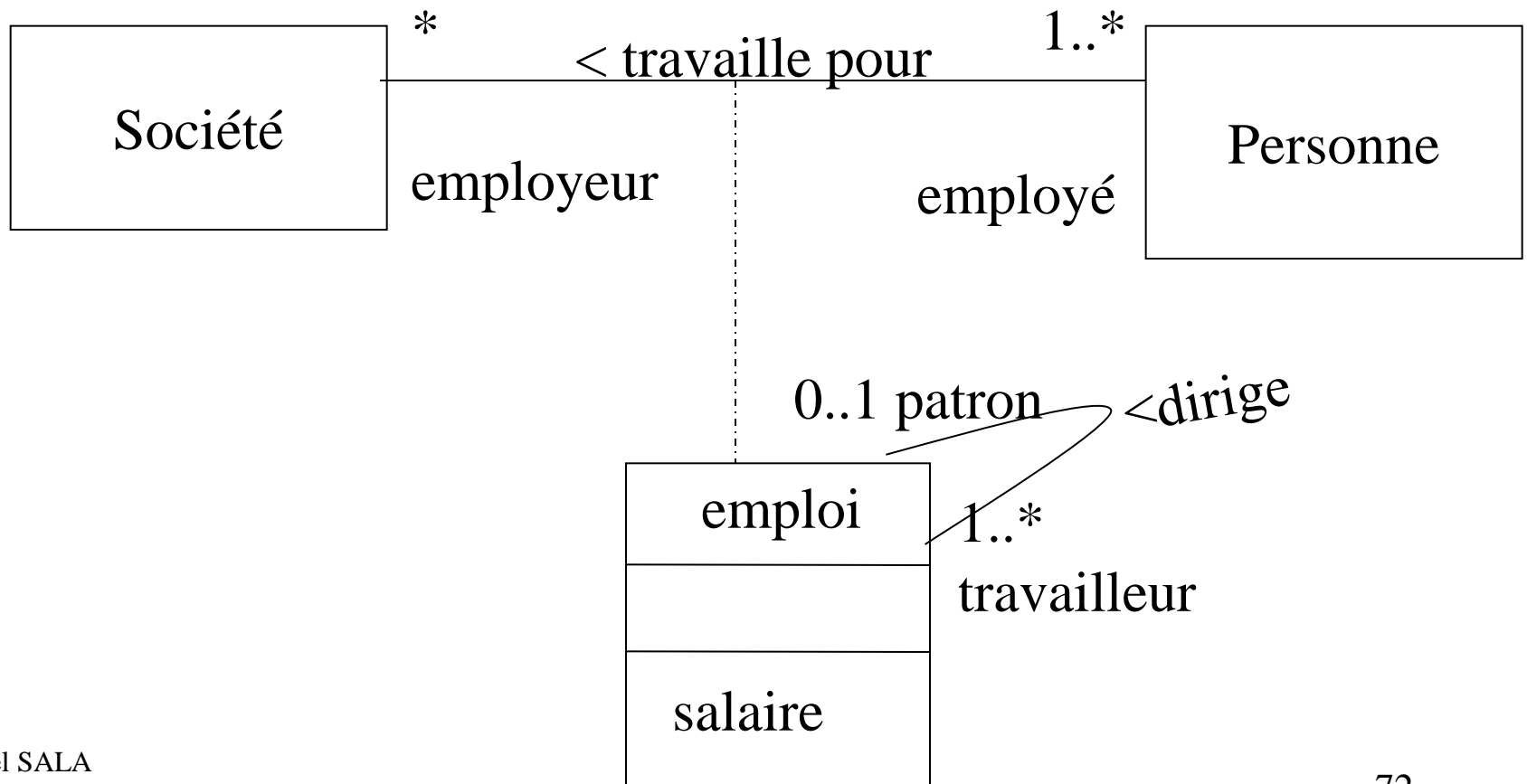
Degré < 2 association unaire ou binaire



Degré > 2 association ternaire

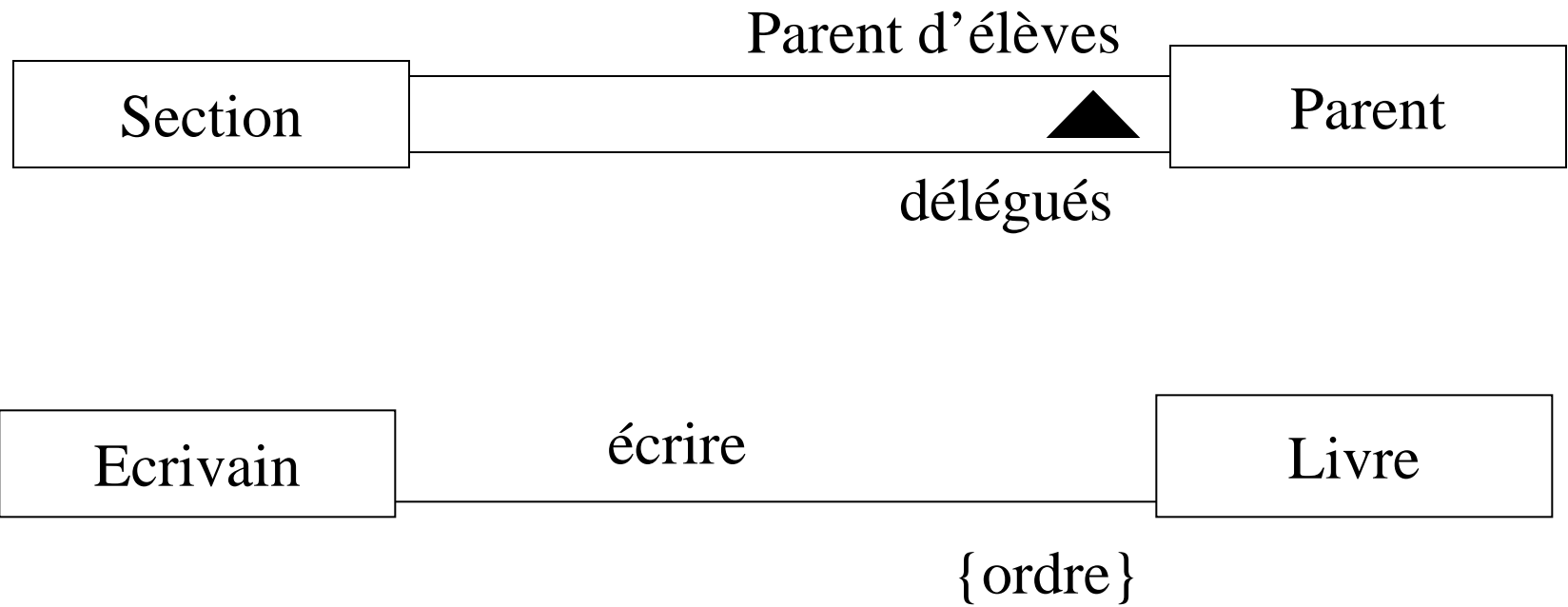


- exemple

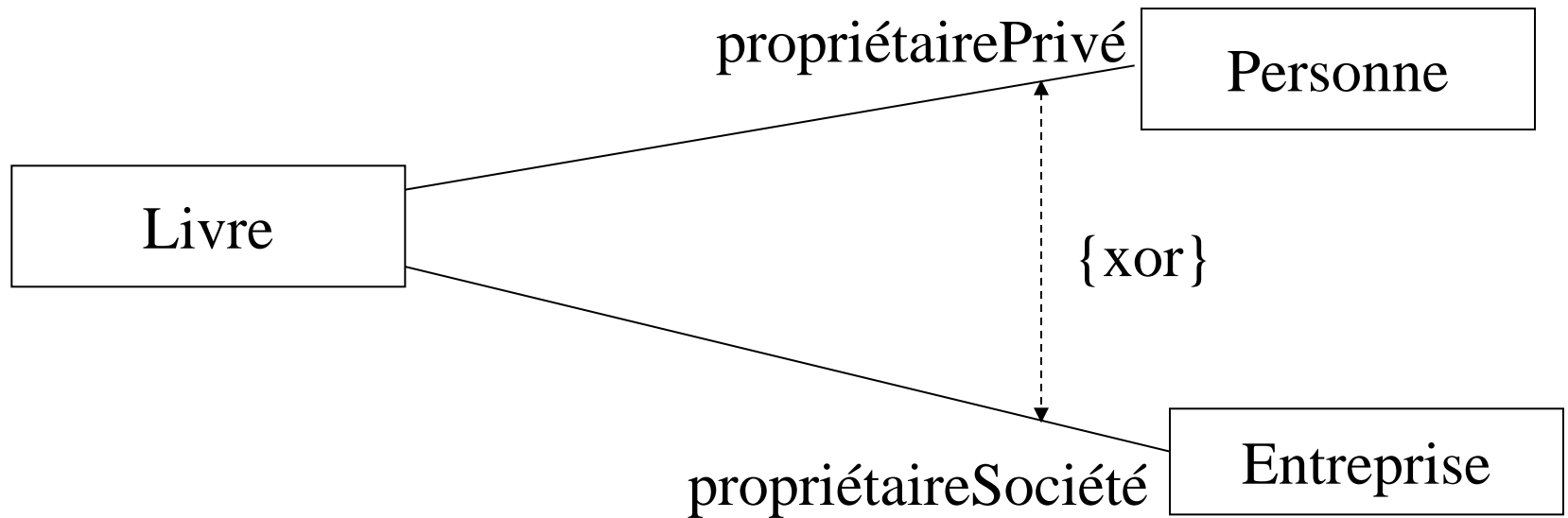


- 3.3.3 Les contraintes
 - Peut affecter une relation ou un groupe de relations
 - Spécifié par :
 - Chaîne de caractères en accolades {}
 - Si nécessaire un triangle noir indique le sens de la contrainte

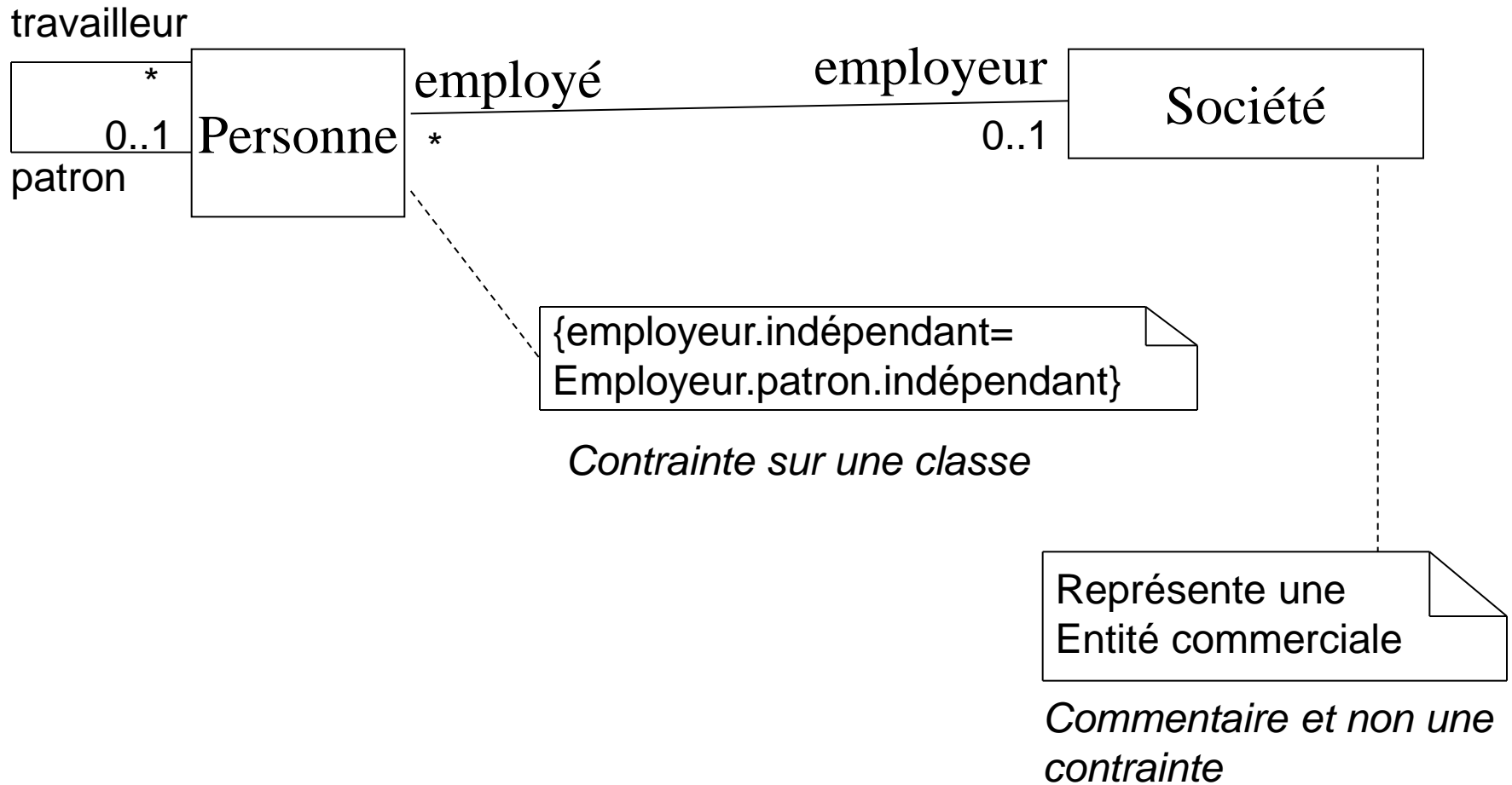
- **exemple**



- **exemple de contrainte**



- exemple

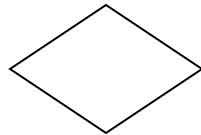


- 3.3.4 Les agrégations

- Relation entre deux classes
- Les objets d'une classe sont des composants d'une autre classe
- Construction d'objets complexes en assemblant des objets de base

- 3.3.4 Les agrégations

- Association non symétrique dans laquelle une des extrémités joue un rôle prédominant (agrégat)
- Concerne un seul objet de l'association
- Symbole

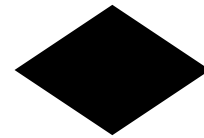


- Agrégation
 - Relation « Composant/Composé »
 - Relation « Partie de »

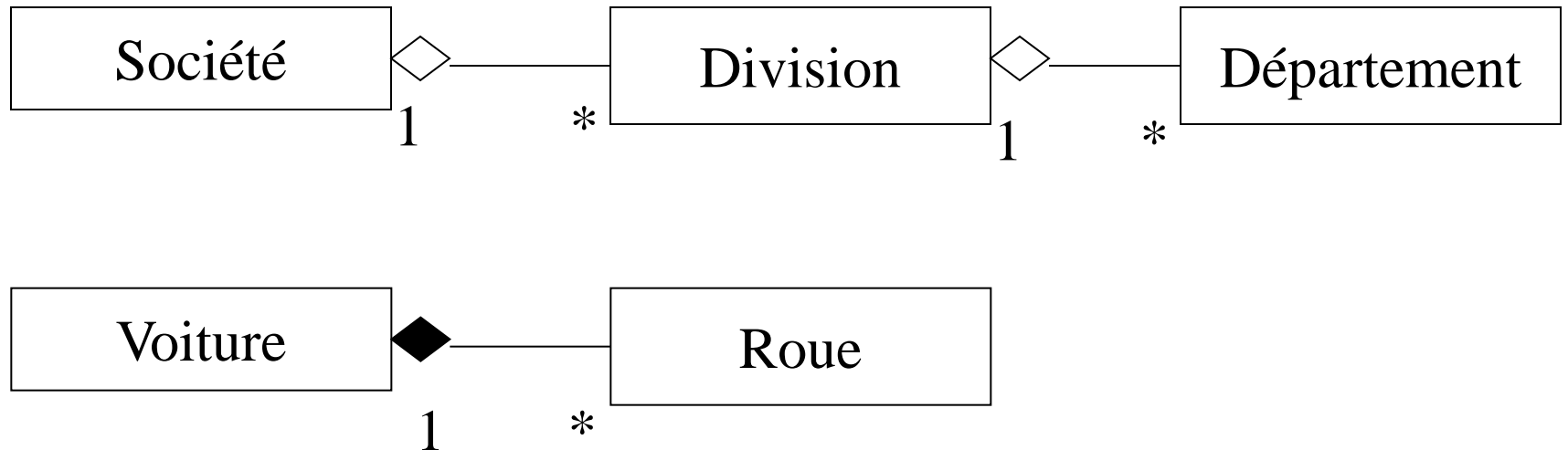
Si A est l'assemblage de B, C, D
B, C, D est une partie de A

Relation :
Transitive
Antisymétrique

La composition est une agrégation forte



- **exemple**



- 3.3.5 Les généralisations

- Désigne une relation de classification entre un élément plus général (super-classe) et un plus spécifique (sous-classe)

- Symbole 

- Abstractions puissantes pour partager des similarités entre classes tout en préservant leurs différences

- Généralisation / Spécialisation
 - Relation entre une classe et une ou plusieurs versions raffinées ou approfondies de celle-ci

La classe à approfondir est dite « super-classe »

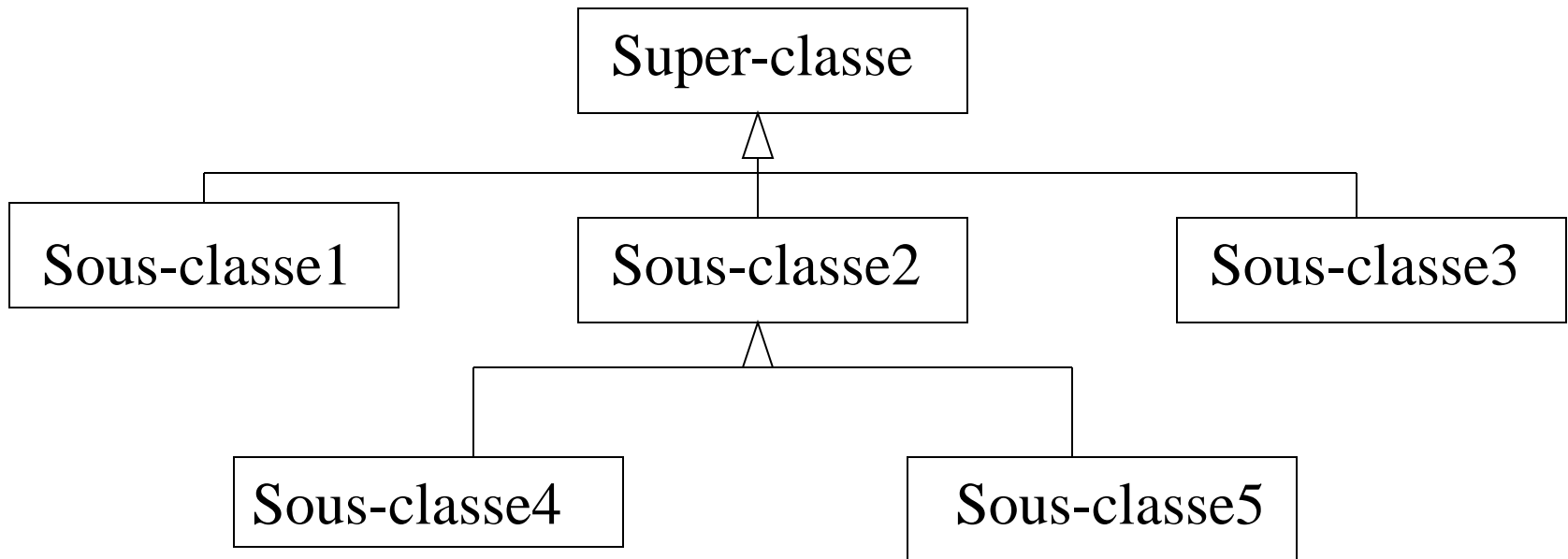
Les classes détaillées sont dites « sous-classes »

- Héritage
 - Simplification conceptuelle qui résulte de la réduction du nombre de caractéristiques indépendantes dans le système

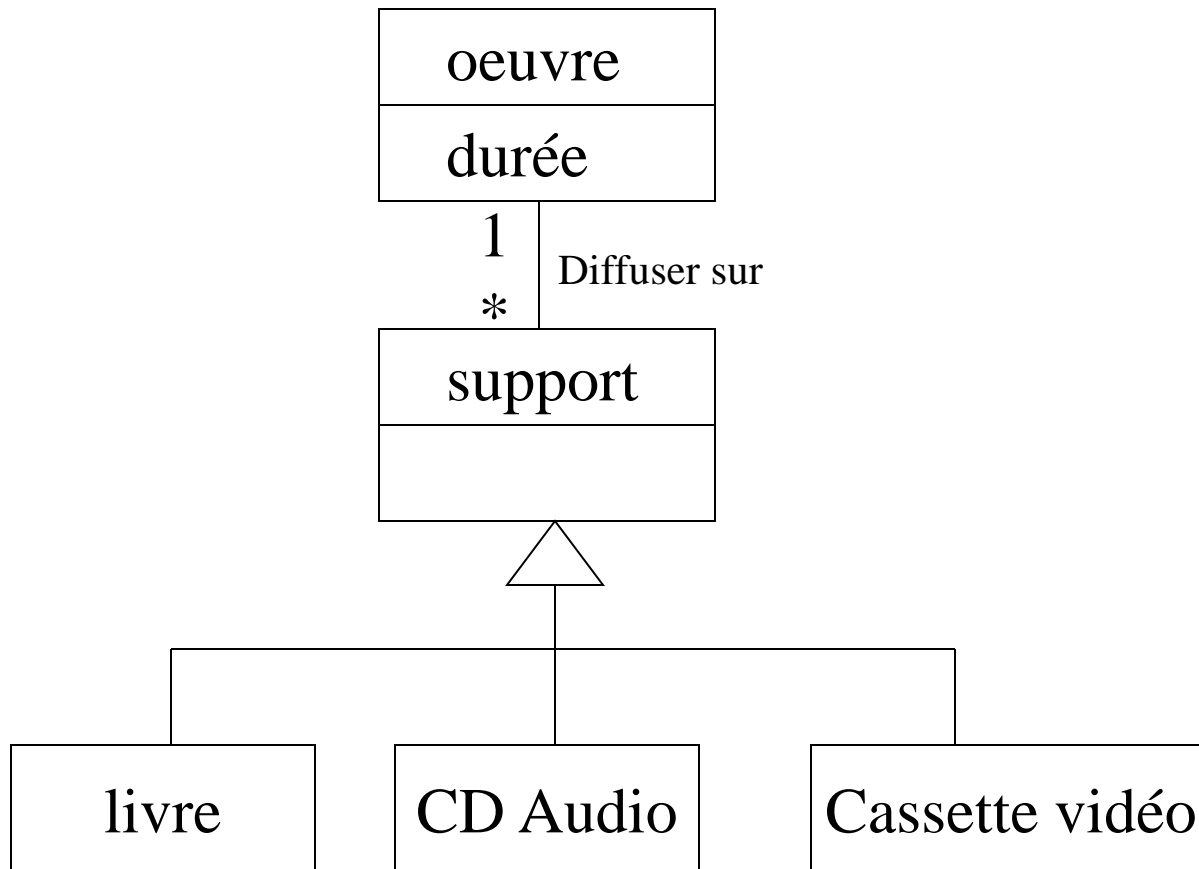
Les sous-classes héritent des attributs et des méthodes de la super-classe

Possibilité d'héritage multiple et de surcharge

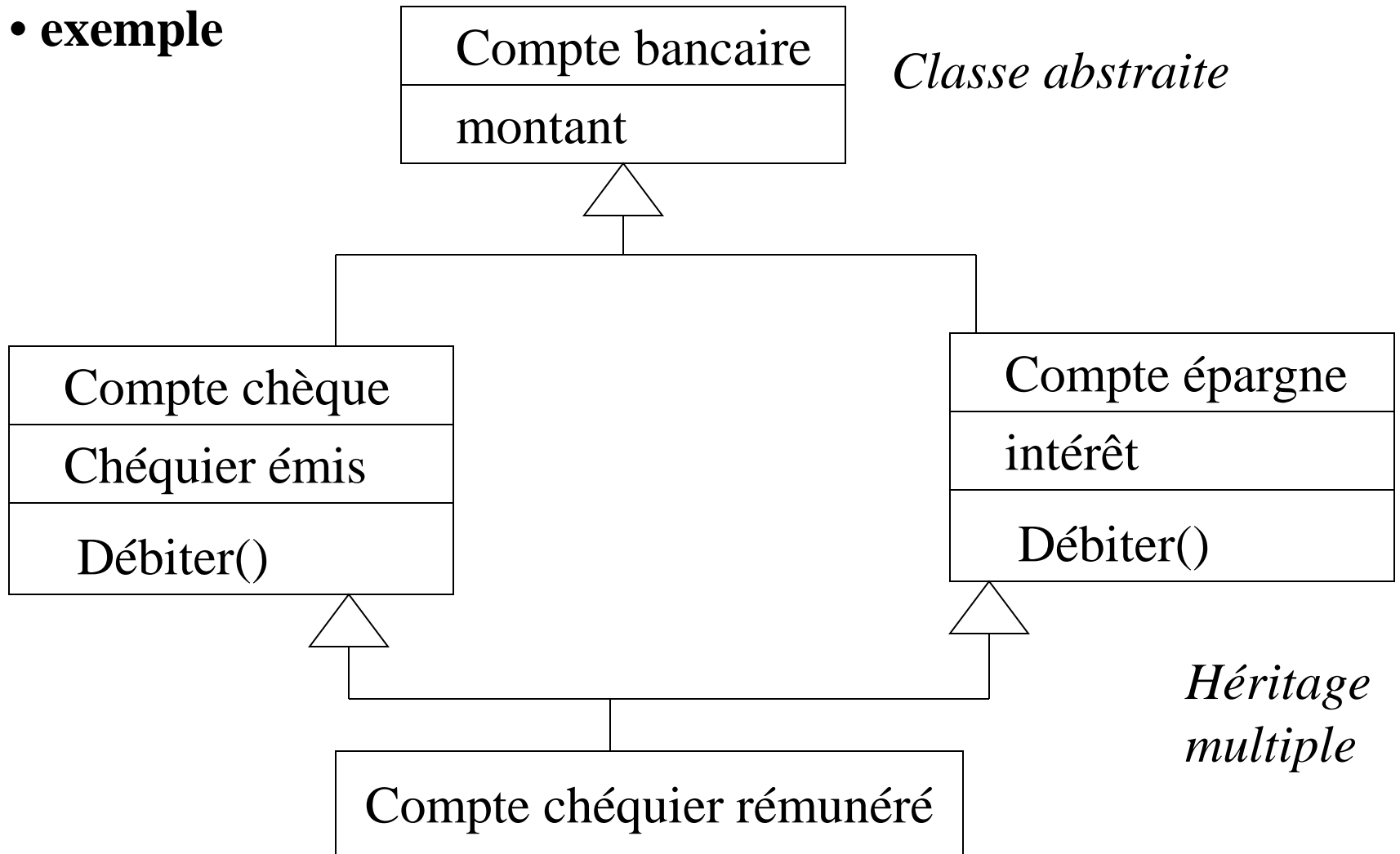
- Représentation graphique



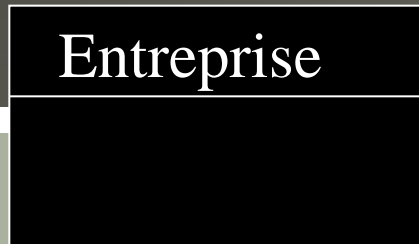
- **exemple**



- **exemple**



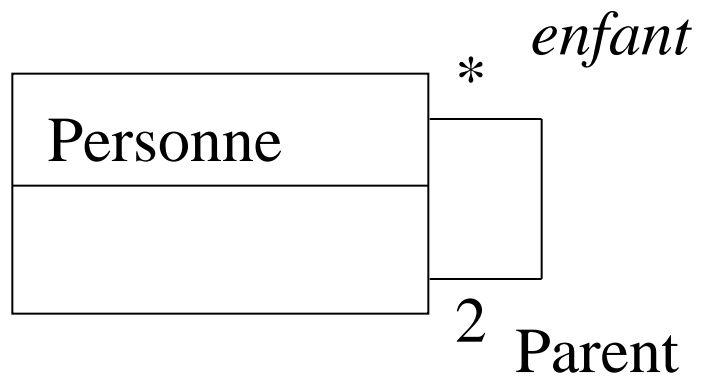
- 3.3.6 Exemple de diagrammes statiques

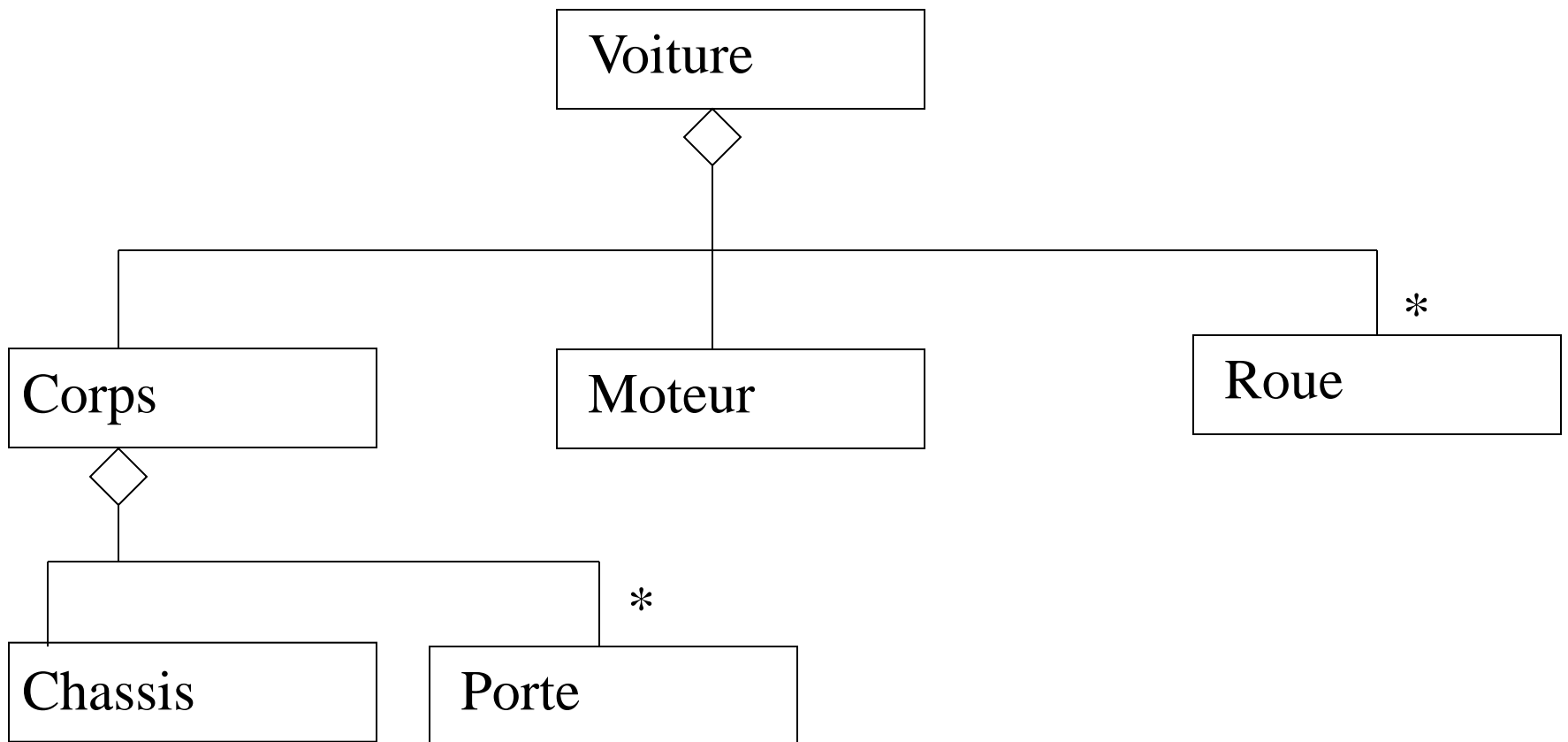


* *Possède des actions* > *

capital actionnaire







Véhicule

```
graph BT; Auto --> Terre[Terrestre]; Amphibie --> Terre; Terre --> Vehicule[Véhicule]; Bateau --> Aquatique[Aquatique]; Aquatique --> Vehicule;
```

Terrestre

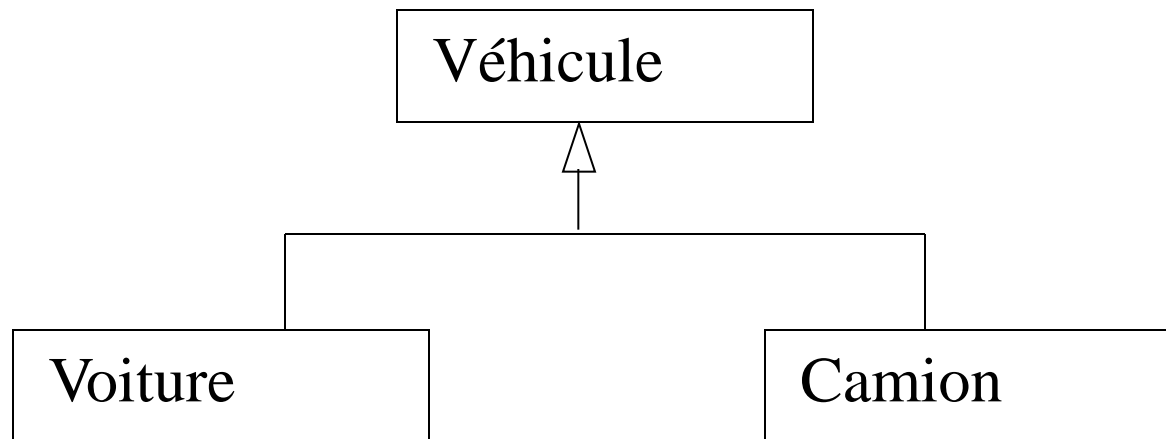
Aquatique

Auto

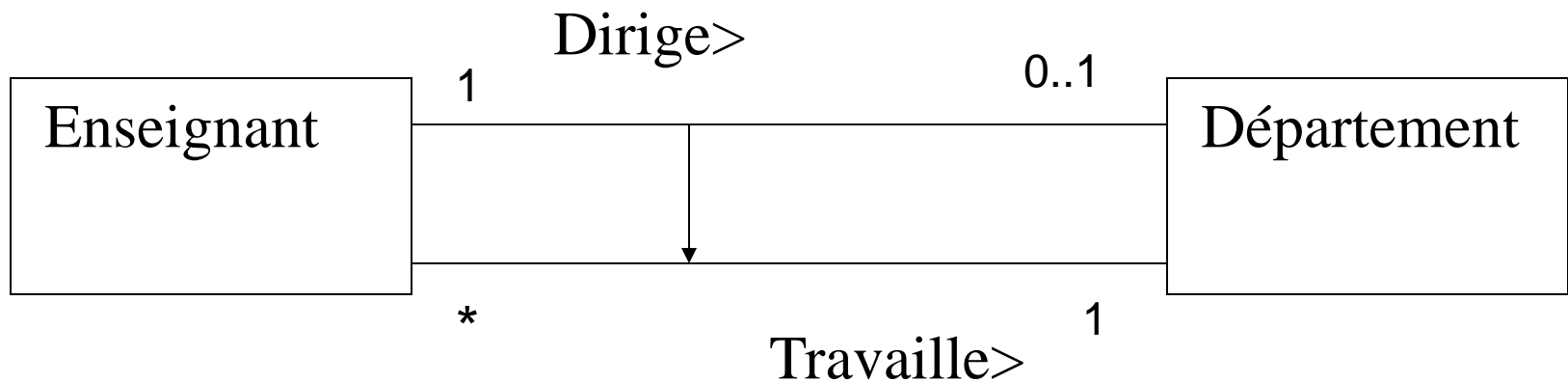
Amphibie

Bateau

Classe abstraite, sans instance



– Contrainte dynamique



programme

IV La vue statique sous UML

4.1 Les diagrammes de classes

4.2 Les diagrammes objets

4.1 Les diagrammes de classes

4.1.1 Les diagrammes de classes

4.1.2 Les classeurs

4.1.3 Les relations

4.1.4 Les dépendances

● 4.1.1 Présentation

- Ils expriment la structure statique d'un système, en terme de :
 - Classes
 - Relations inter classes
- Un diagramme de classe décrit de manière abstraite les liens potentiels d'un objet vers un autre objet
- Le diagramme de classe fait abstraction des aspects dynamiques et temporels

- 4.1.2 Les classeurs
 - Englobe tous les concepts
 - Classes
 - Interfaces
 - Les types de données

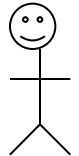
Classeur

Fonction

Notation

Acteur

Utilisateur d'un système



Artefact

Elément d'information physique d'un système

« artifact »
Nom

Classe

Concept d'un système modélisé

Nom

Collaboration

Relation contextuelle entre des objets qui jouent des rôles

Nom

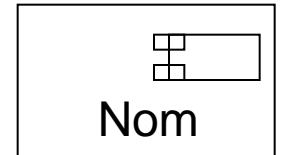
Classeur

Composant

Fonction

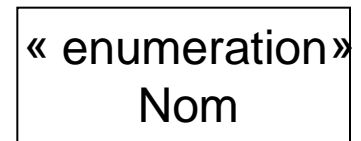
Elément modulaire d'un système avec des interfaces bien définies

Notation



Enumération

Type de données avec des valeurs littérales prédéfinies



Type primitif

Descripteur d'un jeu de valeurs primitives dépourvues d'identité

Nom

Interface

Jeu nommé d'opérations qui caractérise le comportement



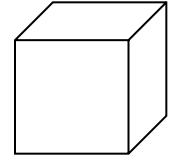
Classeur

Fonction

Notation

Nœud

Ressource informatique



Rôle

Élément interne dans le contexte d'une collaboration ou d'un classeur structuré

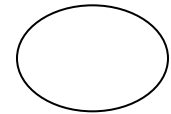
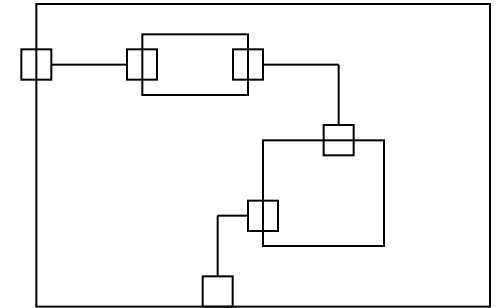
Role:Nom

Signal


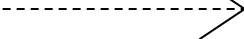
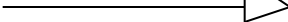
Communication asynchrone entre des objets

« signal »
Nom

Notation



● 4.1.3 Les relations

<i>Relation</i>	<i>Fonction</i>	<i>Notation</i>
Association	Description d'une connexion entre les instances des classes	
Dépendance	Relation entre deux éléments du modèle	
Généralisation	Relation entre une description plus spécifique et plus générale	

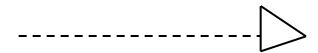
Relation

Fonction

Notation

Réalisation

Relation entre une spécification
et son implémentation



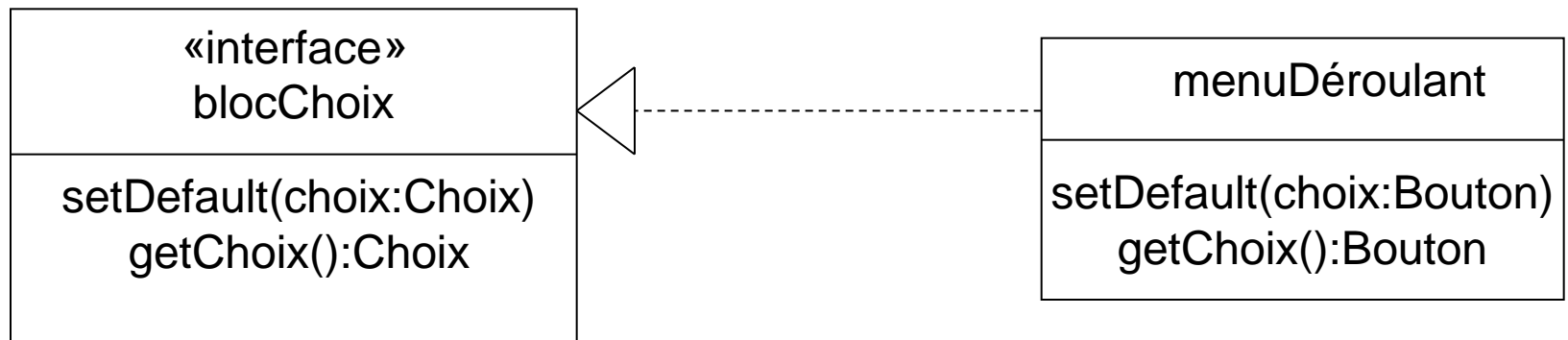
Utilisation

Situation dans laquelle un élément
a besoin d'un autre élément pour
fonctionner correctement



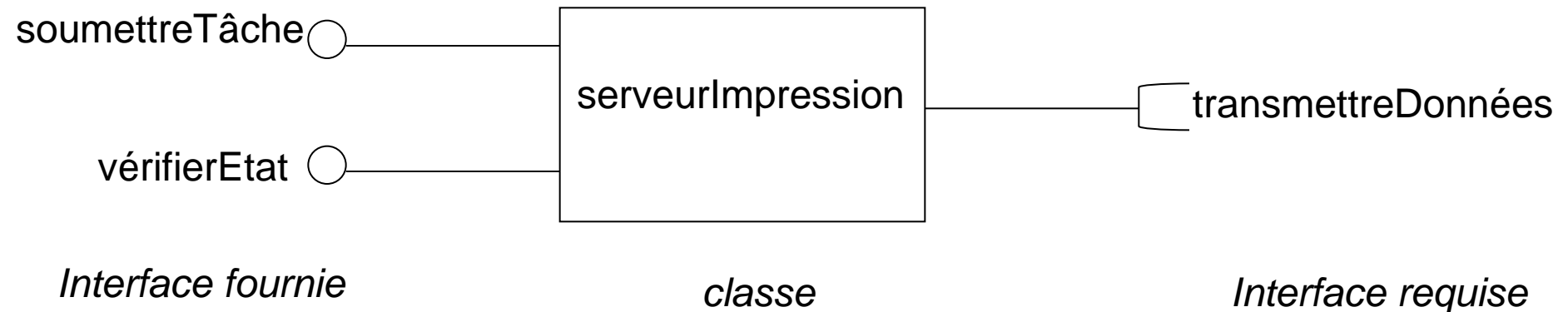
● *Relation de réalisation*

- Connecte l'élément d'un modèle, à un autre élément du modèle qui fournit sa spécification comportementale mais pas sa structure ou son implémentation
- Le client doit prendre en charge (par héritage ou par déclaration directe) au moins toutes les opérations du fournisseur



Interface fournie / interface requise

- Interface fournie : interface réalisée par une classe. Elle fournit les services aux appelants extérieurs
- Interface requise : interface qu'une classe utilise pour implémenter son comportement interne



● 4.1.4 Relation de dépendance

- Une dépendance indique une relation entre deux éléments du modèle ou plus.
- Elle indique une situation dans laquelle un changement apporté à l'élément fournisseur peut nécessiter ou indiquer un changement de signification de l'élément client dans la dépendance

Dépendance	Fonction	Mot clé
accès	Importation privée du contenu d'un autre package	access
paramétrisation (binding)	Attribution de valeurs aux paramètres d'un template pour générer un nouvel élément de modélisation	bind
appel	Une méthode d'une classe appelle une opération d'une autre classe	call
création	Une classe crée des instances d'une autre classe	create
dérivation	Une instance peut calculer une autre instance	derive
instanciation	La méthode d'une classe crée des instances d'une autre classe	instantiate
permission	Permission accordée à un élément d'utiliser le contenu d'un autre élément	permit

Dépendance	Fonction	Mot clé
réalisation	Mappage entre une spécification et une de ses implémentations	realize
raffinement	Un mappage existe entre des éléments situés à deux niveaux de sémantique différente <i>C'est une relation entre deux versions d'un concept à des étapes de développement ou à des d'abstraction différents qui s'expriment comme deux éléments de modélisation distinct</i>	refine
envoi	Relation entre l'expéditeur d'un signal et son destinataire	send
substitution	La classe source prend en charge les interfaces et les contrats de la classe cible qu'elle peut remplacer	substitute

Dépendance Fonction Mot clé

dépendance de trace	Une connexion existe entre les éléments de différents modèles, mais moins précise qu'un mappage	trace
------------------------	---	-------

*C'est une connexion conceptuelle entre des éléments de modèles différents à
différentes étapes de développement.*

utilisation	Un élément requiert la présence d'un autre élément pour fonctionner correctement (comprend les dépendances d'appel, de création, d'instanciation, d'envois...)	use
-------------	---	-----

4.2 Les diagrammes d'objets

- Les diagrammes de classe et les diagrammes d'objets appartiennent à deux vues complémentaires du domaine
- Un diagramme d'objets représente plutôt un cas particulier du diagramme de classes, une situation concrète à un instant donné

– Les règles de transition entre les 2 diagrammes sont :

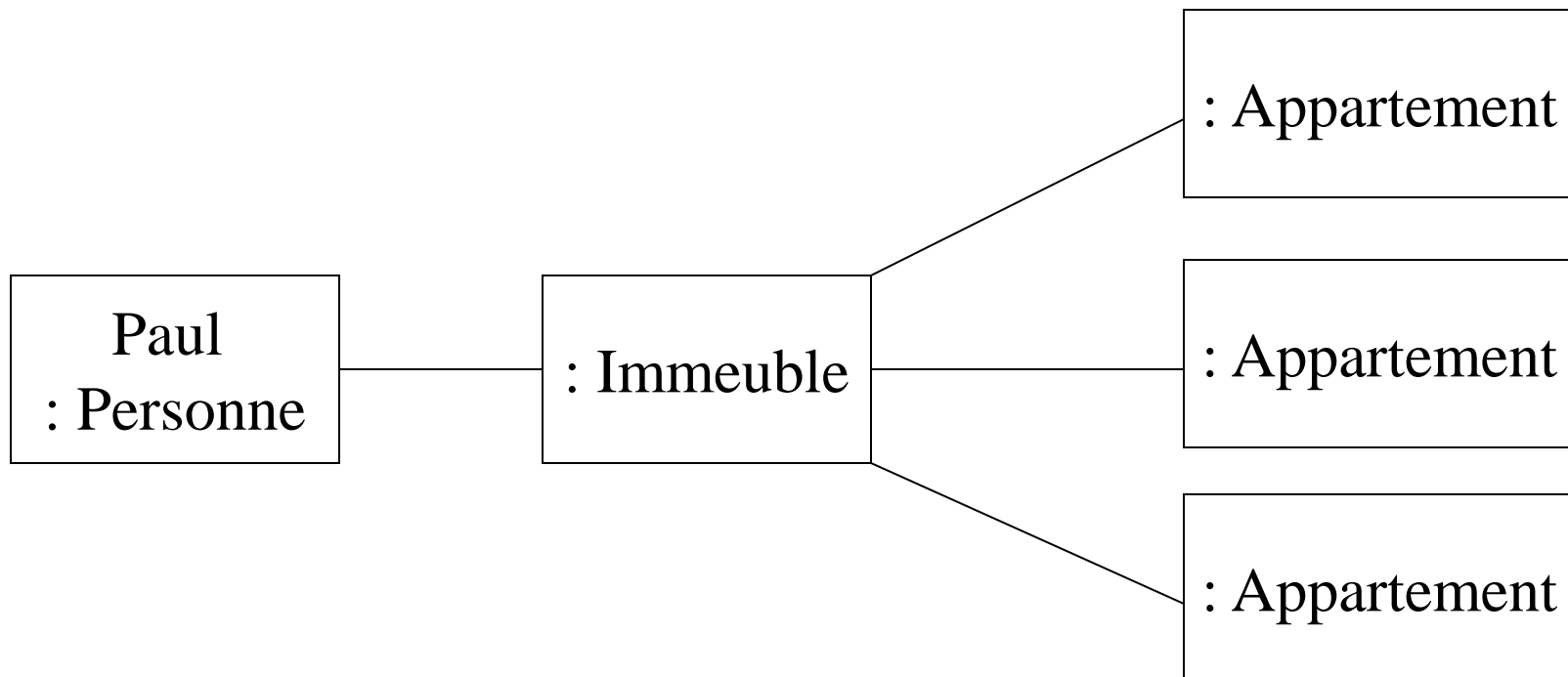
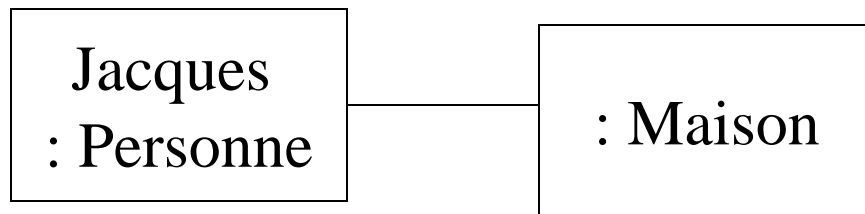
- Chaque objet est instance d'une classe, la classe ne change pas durant la vie de l'objet
- Les classes abstraites ne peuvent pas être instanciées
- Chaque lien est instance d'une relation (association, agrégation ou composition)
- Un lien entre 2 objets implique une relation entre les classes
- Les diagrammes d'objets sont des instances des diagrammes de classes

Représentation des objets

nom de l'objet

nom de l'objet : Classe

: Classe



programme

V La vue de conception sous UML

5.1 Les diagrammes de Collaboration

5.2 Les diagrammes de Composants

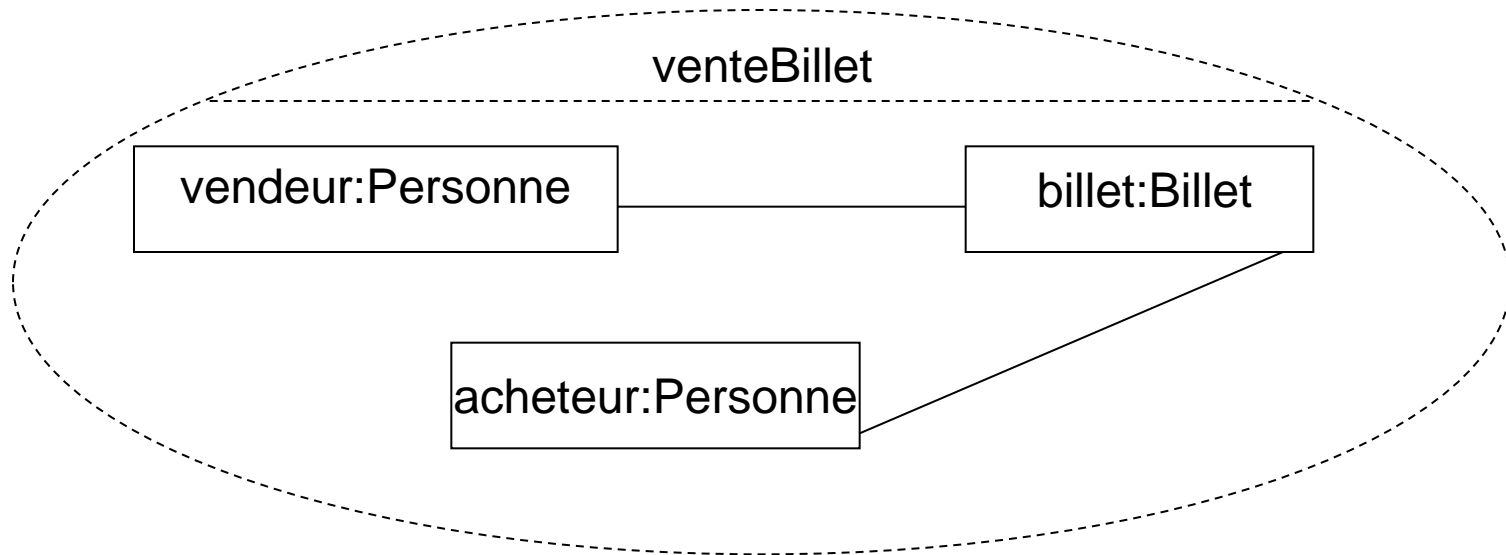
5.1 Diagramme de collaboration

5.1.1 Présentation

5.1.2 Messages

● 5.1.1 Présentations

- Montrent des interactions entre objets
- Une collaboration est un type de classeur structuré



Une interaction :

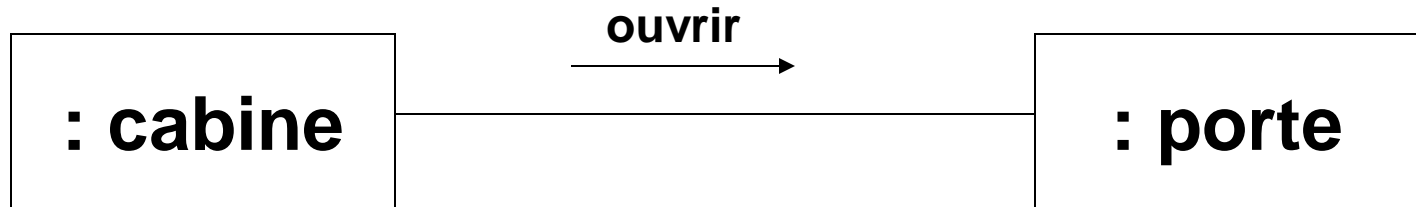
- **Les arguments**
- **Les variables locales créées pendant l'exécution**
- **Les liens entre objets qui participent à l'interaction**

- **Spécifie de manière précise l'ordre et les conditions d'envoi de messages**
 - **Pour chaque message, il est possible d'indiquer :**
 - **Les clauses qui conditionnent son envoi**
 - **Son rang**
 - **Sa récurrence**
 - **Ses arguments**

- 5.1.2 Messages

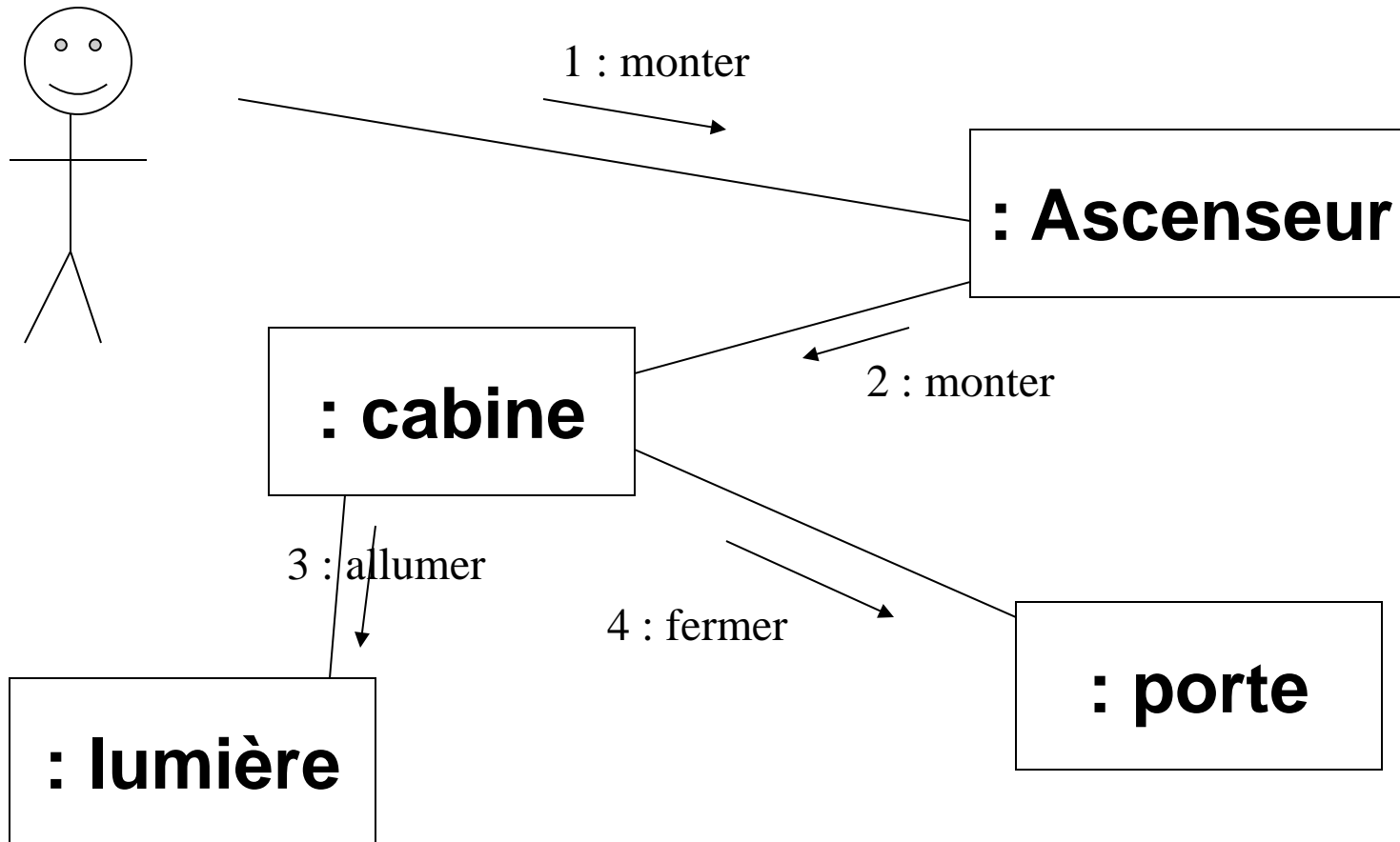
- **Un message est représenté par une flèche orientée vers le destinataire**

La cabine demande à la porte de s'ouvrir

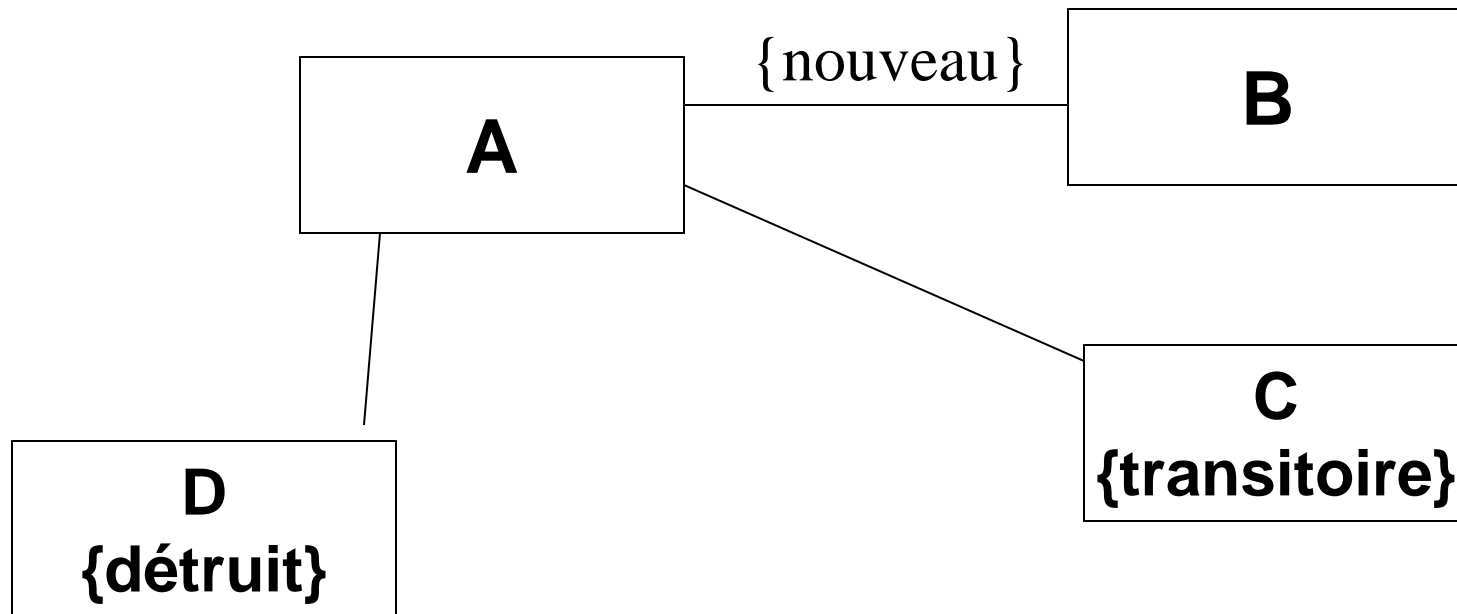


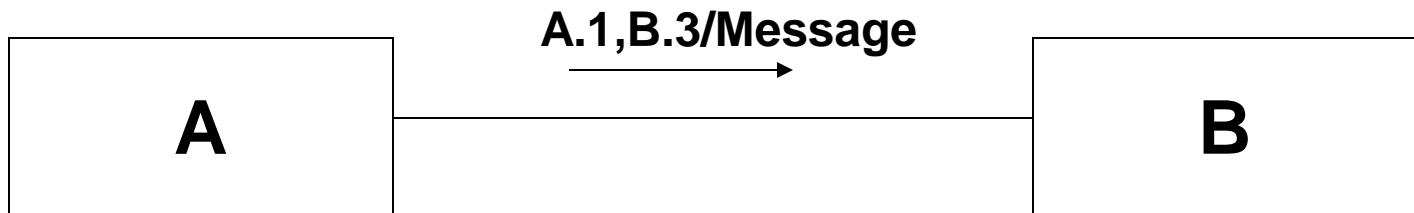
La cabine demande à la porte de s'ouvrir

– Ordre des envois de messages



- **Objets et liens créés ou détruits au cours d'une interaction peuvent porter les contraintes {détruit}, {nouveau}, {transitoire}**





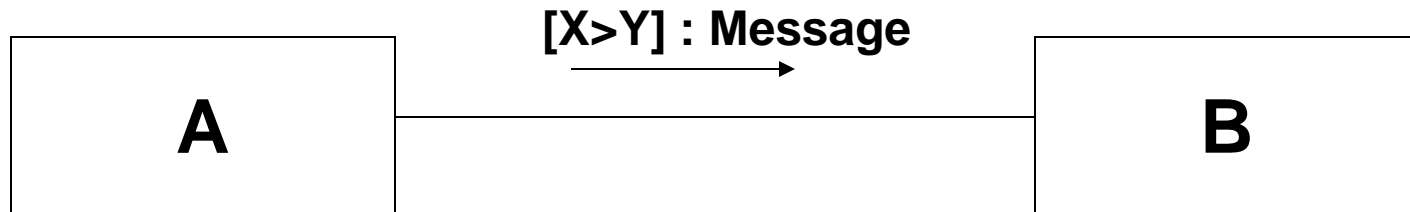
Le message “Message” est envoyé lorsque les envois A.1 et B.3 ont été réalisés



Envoi de message séquentiel (*) avec itération



Envoi de message parallèle (*||) avec itération



Envoi de message conditionnel

5.2 Diagramme de composants

5.2.1 Présentation

5.2.2 Les composants

5.2.3 Les modules

5.2.4 Les dépendances entre composants

● 5.2.1 Présentations

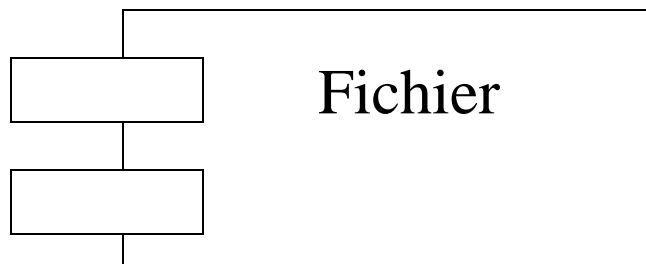
- Identifie « physiquement » les classes
- Décrit l'organisation des composants
- Définit les contraintes du développement

- Les diagrammes de composants décrivent les composants et leurs dépendances dans l'environnement de réalisation
- Ce sont des vues statiques de l'implémentation des systèmes qui montrent les choix de réalisation
- Ils ne sont utilisés que pour des systèmes complexes

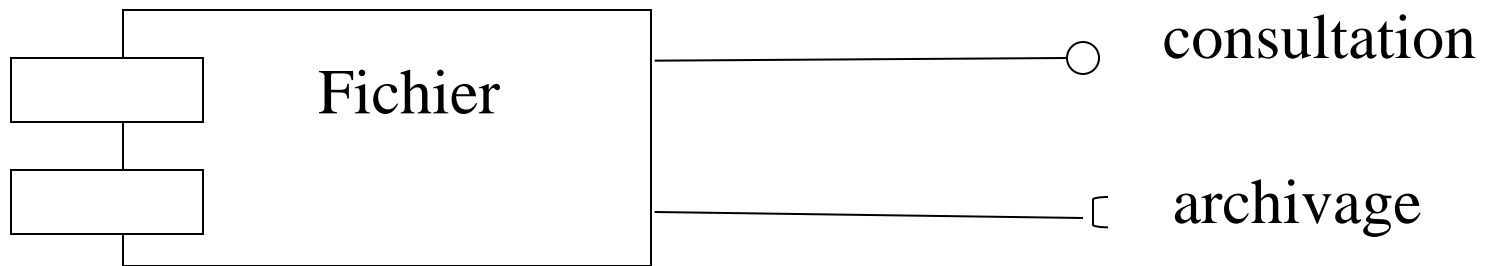
● 5.2.2 Les composants

- Un composant est un élément physique ou logique qui représente une partie implémentée du système
- Les composants possèdent des interfaces qu'ils prennent en charge (interfaces fournies) et des interfaces qu'ils demandent à d'autres composants (interfaces requises)

- Un composant est représenté par un rectangle principal avec sur son côté gauche deux plus petits rectangles
- Le nom du composant est placé dans le rectangle principal

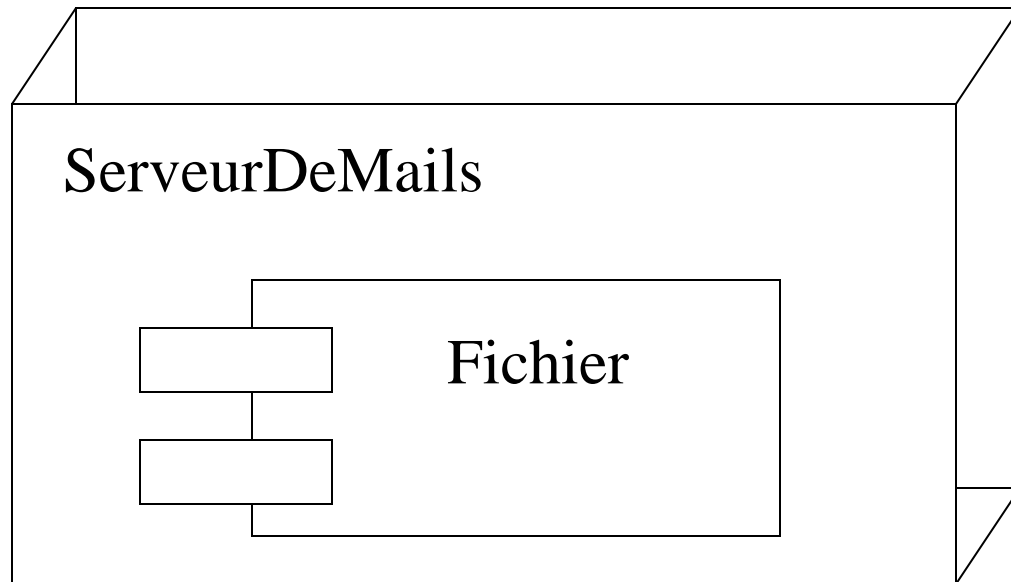


🌀 Un composant fichier et deux de ses interfaces

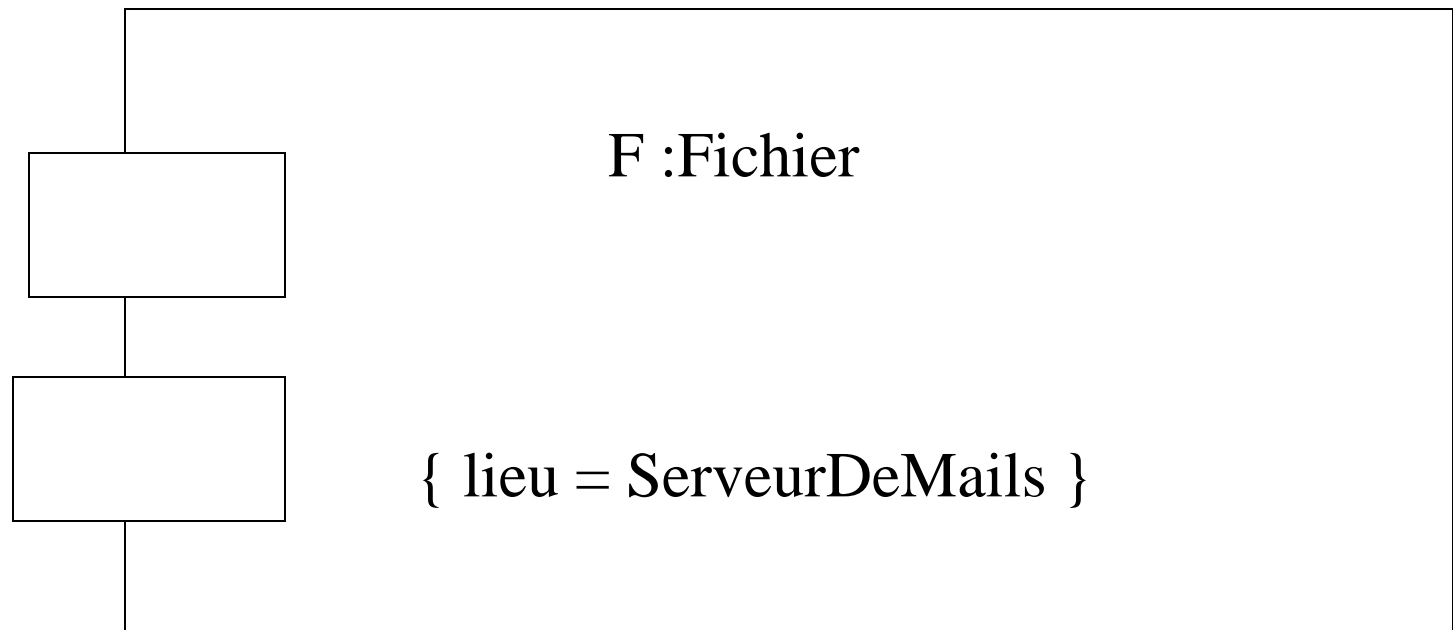


- ∞ Un composant est défini comme une sous-classe de **Classificateur**
- ∞ Il peut ainsi avoir des attributs, des opérations et participer à des relations
- ∞ Pour montrer les instances des composants, un diagramme de déploiement doit être utilisé

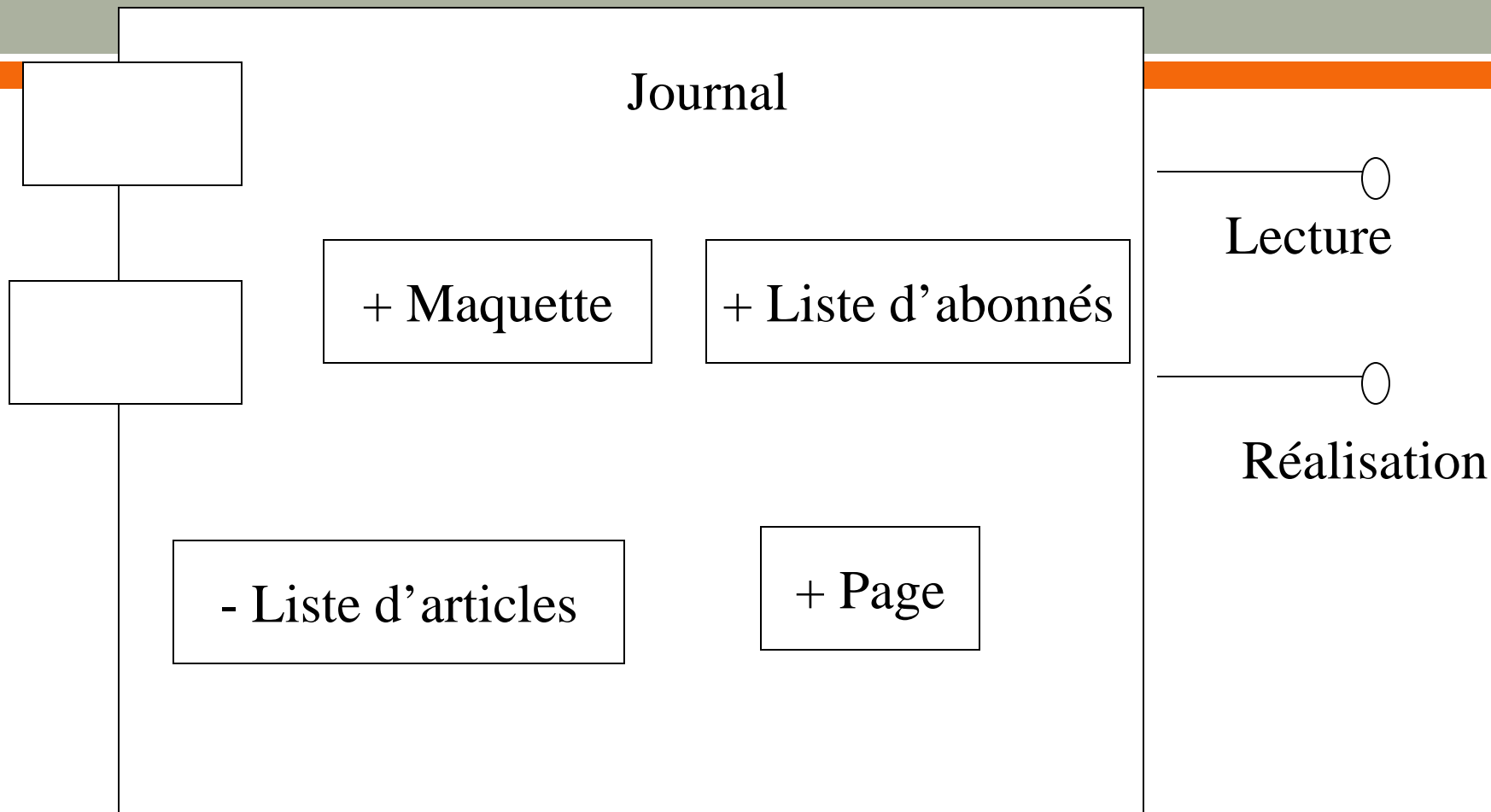
- Représentation d'une instance du composant **Fichier** dans un noeud




- Représentation du processeur où réside l'instance du composant **Fichier**



- ✎ Les composants d'un composant plus global sont représentés dans le symbole de ce dernier
- ✎ Les classes implémentées par un composant sont également représentées dans le symbole du composant



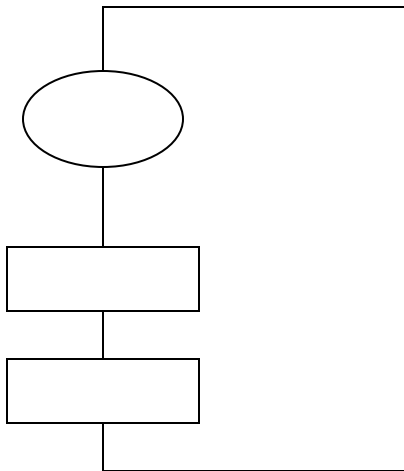
- 
- ∞ UML définit divers stéréotypes aux composants :
- « **documents** » : un document quelconque
 - « **exécutable** » : un programme qui peut s'exécuter sur un nœud
 - « **fichier** » : un document contenant du code source ou des données
 - « **bibliothèque** » : une bibliothèque
 - « **table** » : une table d'une base de données relationnelle

- **5.2.3 Les modules**

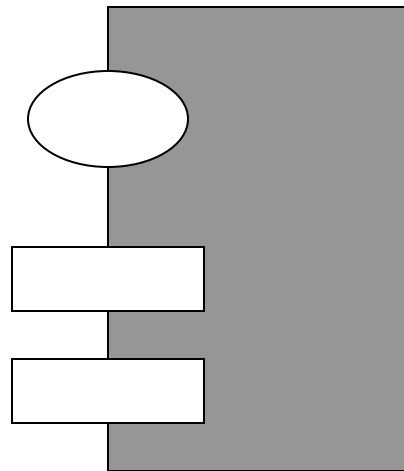
- Les modules représentent une unité pour la manipulation et le stockage de toutes les sortes d'éléments physiques qui entrent dans la fabrication des applications informatiques

- Exemples de représentation graphique de différentes sortes de modules à l'aide de composants stéréotypés

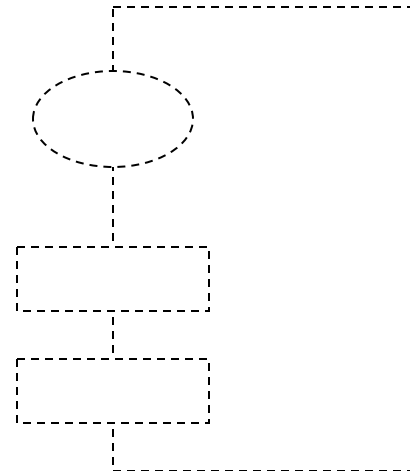
Spécification
Interface



Corps
Implémentation



Générique



∞ En C++ :

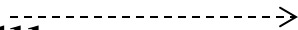
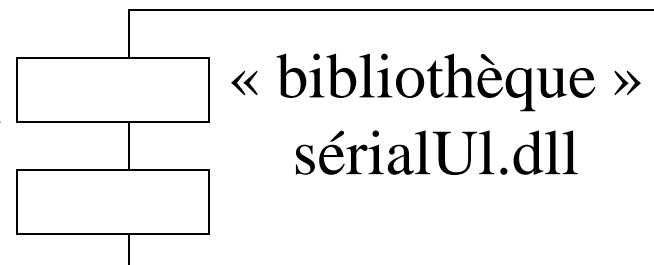
- Une spécification correspond à un fichier .h
- Un corps à un fichier .cpp

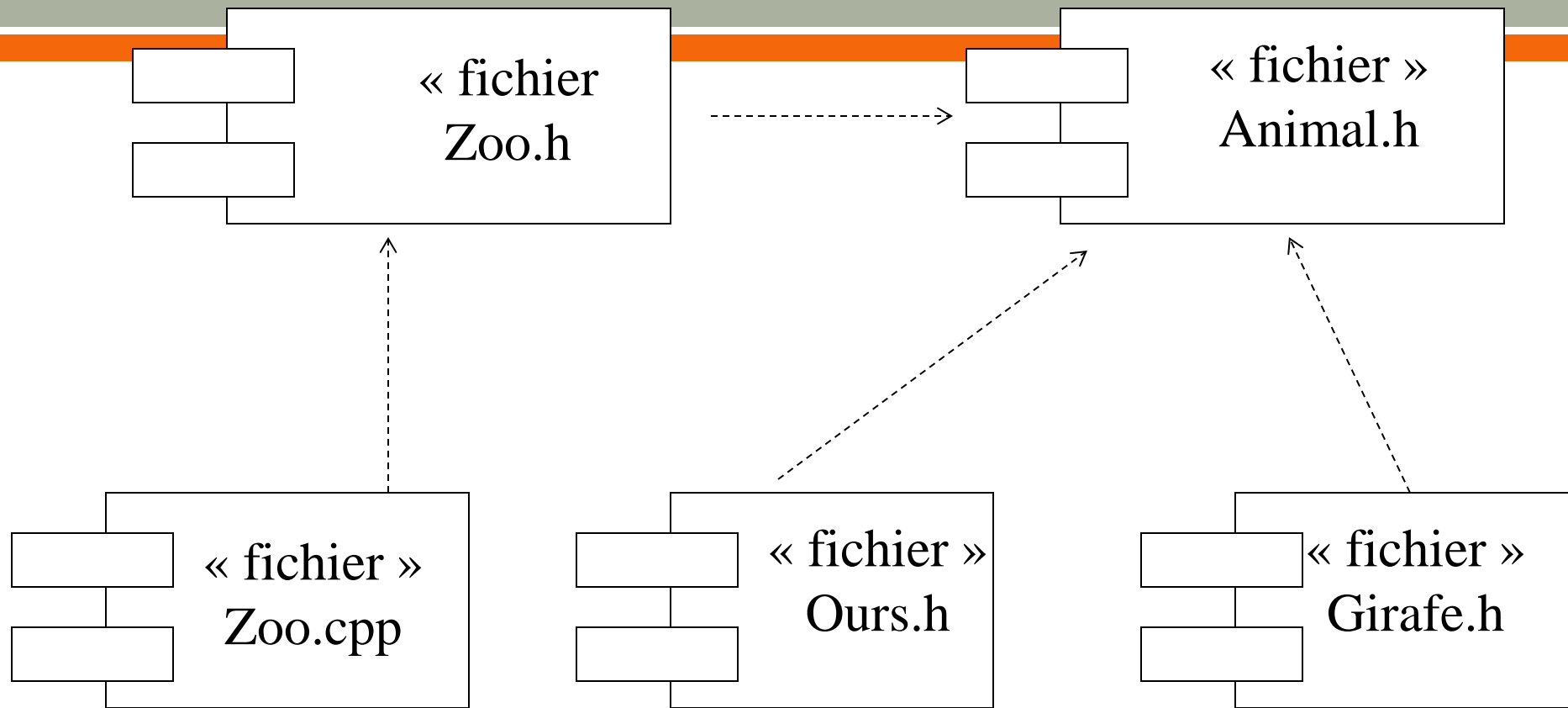
∞ En Ada :

- Les modules existent sous le nom de paquetages

● 5.2.4 Les dépendances entre composants

- pour indiquer qu'un élément d'implémentation d'un composant fait appel aux services offerts par les éléments d'implémentation d'un autre composant
- Une telle relation de dépendance est représentée par une flèche en pointillée orientée du composant utilisateur vers le composant fournisseur





programme

VI Vue de cas d'utilisation

6.1 Généralités

6.2 Les acteurs

6.3 Les relations entre cas d'utilisation

● 6.1 Généralités

- Les diagrammes de cas d'utilisation permettent :
 - D'exprimer les besoins
 - De définir les limites du système
 - De définir les relations entre le système et son environnement
 - De décrire le comportement du système du point de vue de l'utilisateur à partir d'actions et de réactions

- Ils identifient les utilisateurs du système : les acteurs
 - Ils classent les acteurs et structurent les objectifs
- Jacobson identifie les caractéristiques :
- Un modèle est une simplification de la réalité
 - Permet de mieux comprendre le système
 - Permet de modéliser les besoins du client
 - Surtout ne pas décrire des solutions d'implémentation

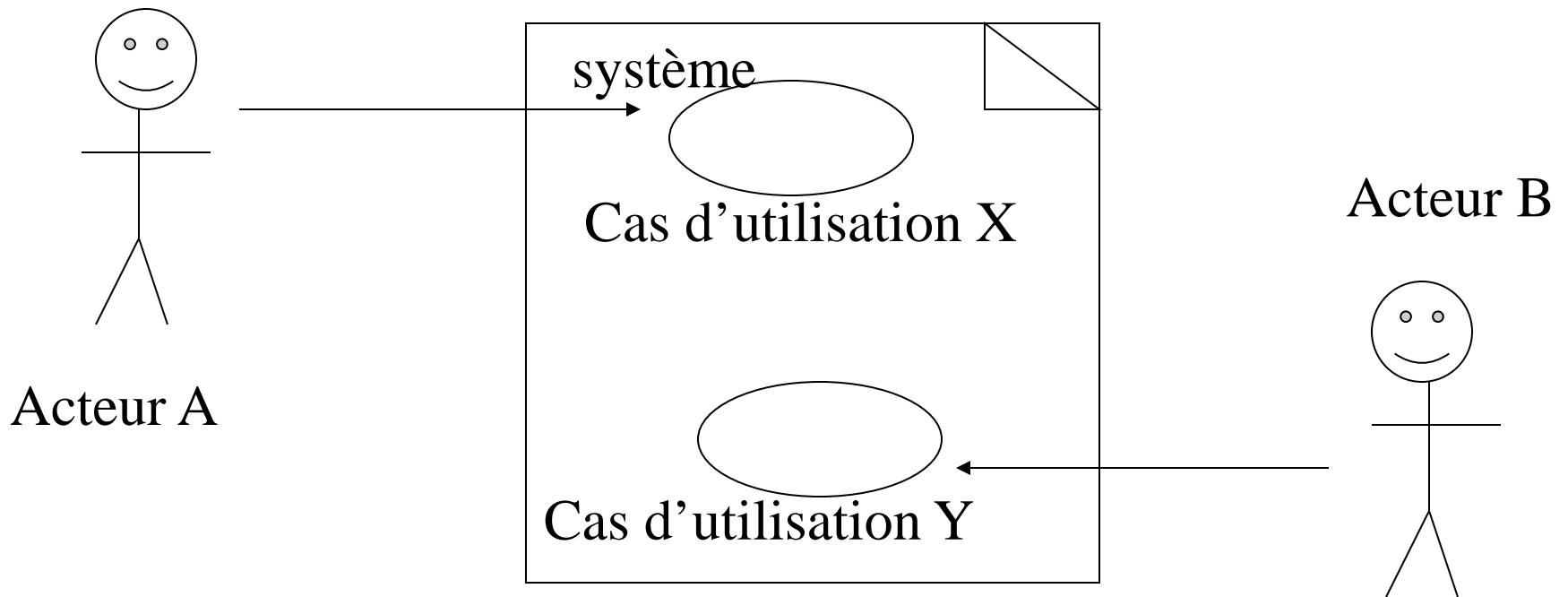
- Les diagrammes de cas d'utilisation servent de fil conducteur à toutes les étapes du projet :
 - L'utilisateur exprime le cas
 - L'analyste comprend le cas
 - Le concepteur conçoit le cas
 - Le programmeur réalise le cas
 - Le testeur vérifie le cas

- **6.2 Les acteurs**

- **Entité externe qui agit sur le système**

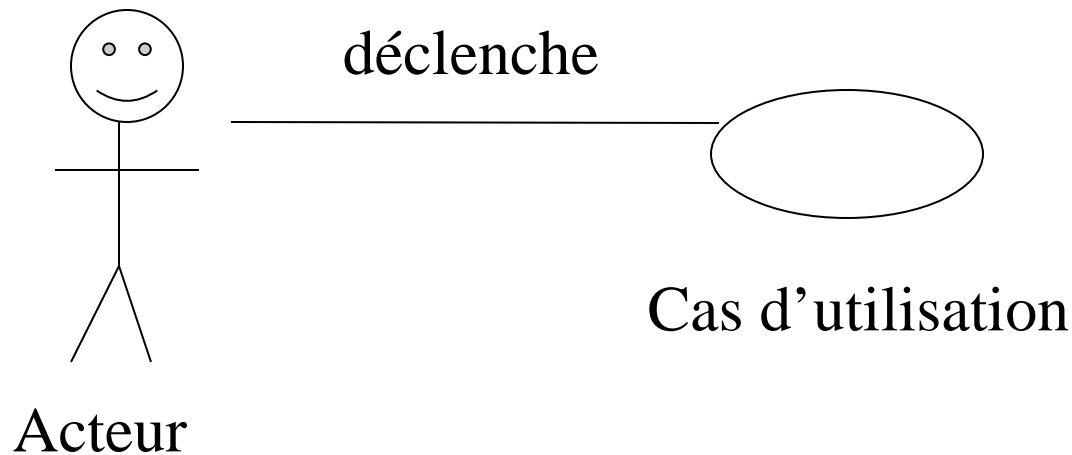
- Il peut consulter ou modifier l'état du système
 - A l'action d'un acteur, le système fournit un service
 - Les acteurs peuvent être classés
 - Un même utilisateur peut jouer différents rôles d 'acteurs

- Représente un rôle joué par une personne ou une chose qui interagit avec un système



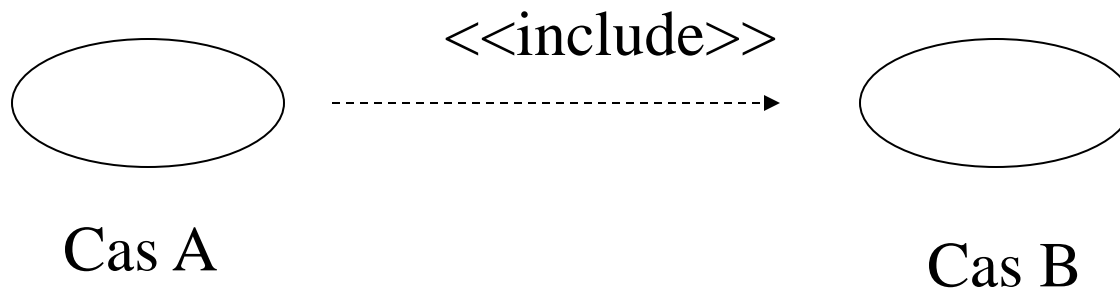
● 6.3 Les relations entre cas d'utilisation

- association



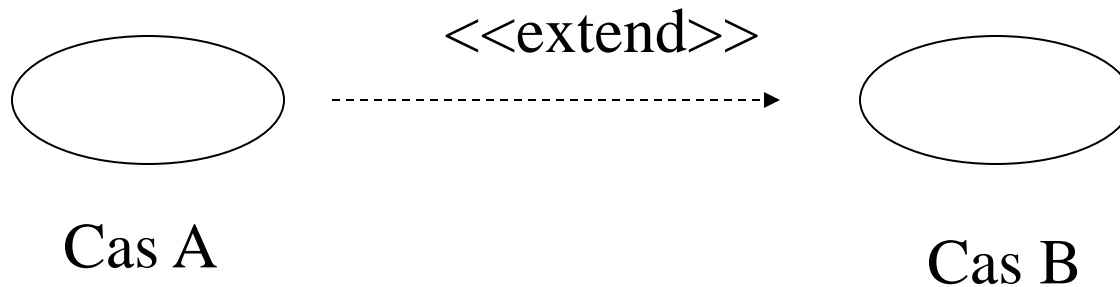
Chemin de communication entre un acteur et un cas d'utilisation auquel il participe

- Inclusion (include)



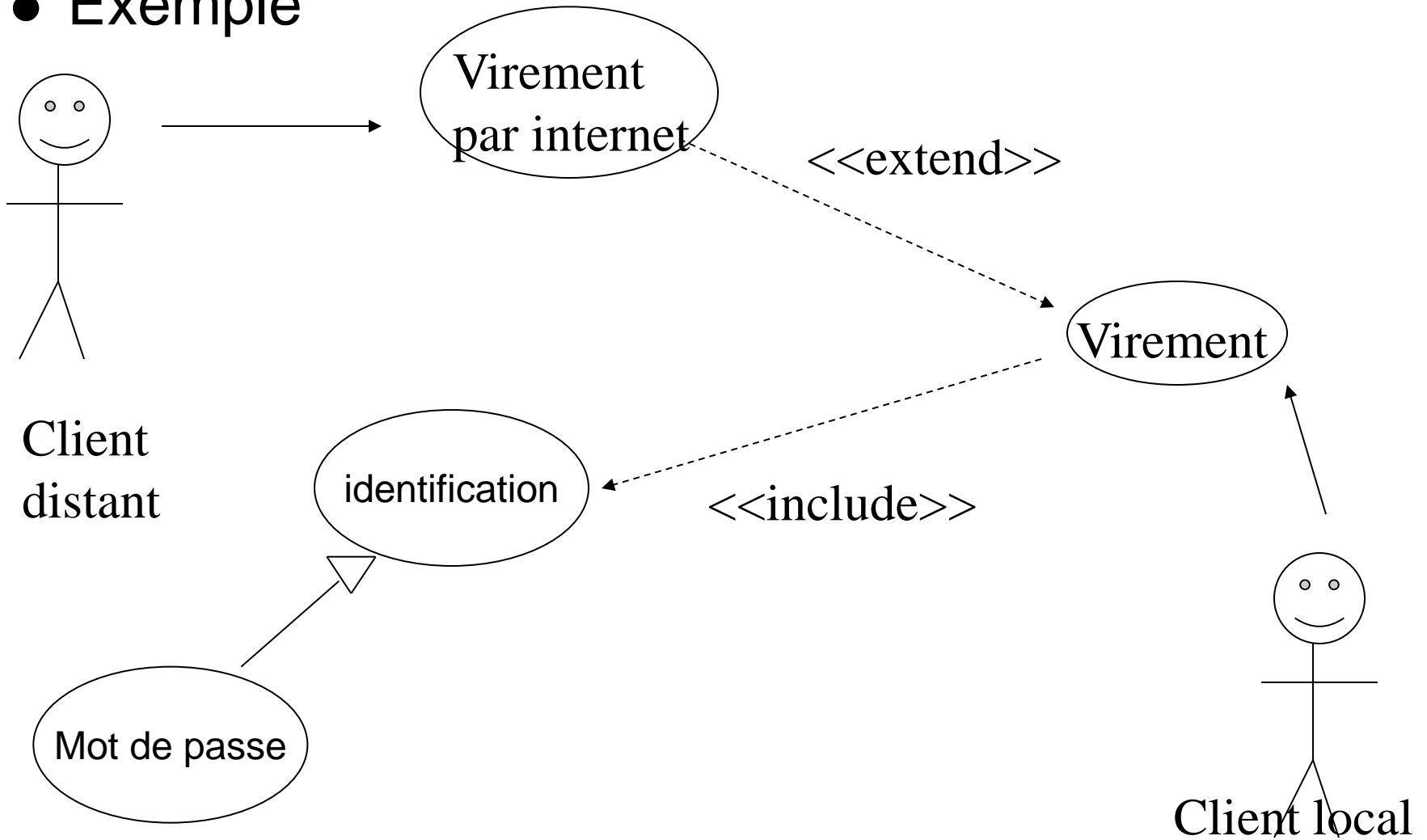
Insertion d'un comportement supplémentaire dans un cas d'utilisation de base qui décrit explicitement l'insertion

- Relation d'extension (extend)



Insertion d'un comportement supplémentaire dans un cas d'utilisation de base dont il ne sait rien

● Exemple



programme

VII La vue dynamique sous UML

7.1 Vue de machine d'état

7.2 Vue d'activité

7.3 Vue des interactions

7.1 Vue de machine d'états

7.1.1 Présentation

7.1.2 Événement

7.1.3 La notion d'état

7.1.4 La notion de transition

7.1.5 Etat composite

7.1.6 Diagramme de machine d'états

● 7.1.1 Présentation

- Une machine d'états est un graphique présentant des états et des transitions
- Elle est habituellement reliée à une classe
- Elle peut être rattacher à des cas d'utilisation ou de collaboration

● 7.1.2 Evènement

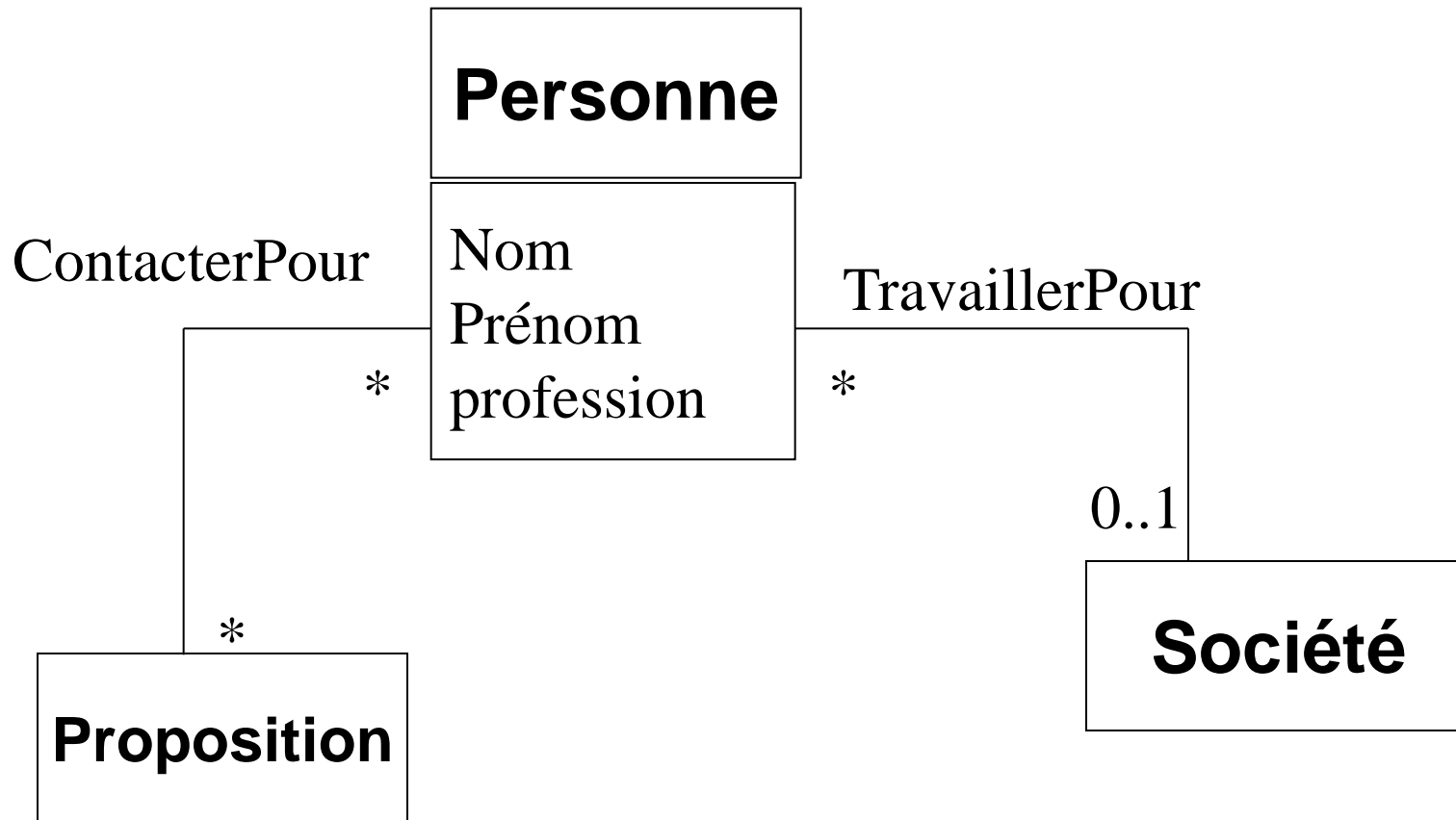
- Un événement est un type d'occurrence remarquable localisé dans le temps
- Il se produit à un moment précis et il n'a pas de durée
- Il existe 4 types d'évènement :
 - appel
 - changement
 - signal
 - temps

Type d'événement	description	syntaxe
appel	Réception d'une demande d'appel explicite synchrone par un objet	op(a:T)
changement	Changement dans la valeur d'une expression booléenne	when(exp)
signal	Réception d'une communication explicite, nommée, asynchrone entre des objets	sname(a:T)
temps	Arrivée d'un temps absolu ou passage d'un laps de temps relatif	after(time)

● 7.1.3 La notion d'état

- L'état d'un objet est défini :
 - Par les valeurs de ses variables d'instances
 - Par les valeurs de ses liens avec d'autres objets
- L'état d'un objet représente une durée, un intervalle de temps, un espace de temps séparé par deux évènements.

– Exemple



-L'objet *Personne* passe par 3 états différents dépendant de l'attribut *profession* et du lien avec l'objet *Société* et le ou les objets *Proposition* :

-L 'état Employé : l'objet *Personne* passe dans cet état quand l'attribut *profession* possède une valeur différente de “sans profession” et d'autre part lorsqu'il possède un lien *travaillerPour* avec un objet *Société*

- L 'état *DemandeurEmploi* : l'objet *Personne* passe dans cet état quand l'attribut *profession* possède une valeur différente de "sans profession" et lorsqu'il n'a pas de lien *travaillerPour* avec un objet *Société* ni de lien *contacterPour* avec un objet *Proposition*

- L 'état *EnPhaseEmbauche* : l'objet *Personne* passe dans cet état quand l'attribut *profession* possède une valeur différente de "sans profession" et lorsqu'il n'a pas de lien *travaillerPour* avec un objet *Société* mais qu'il a un lien *contacterPour* avec un ou plusieurs objets *Proposition*

-L'état d'un objet est représenté de la manière suivante :

Employé

DemandeurEmploi

En PhaseEmbauche

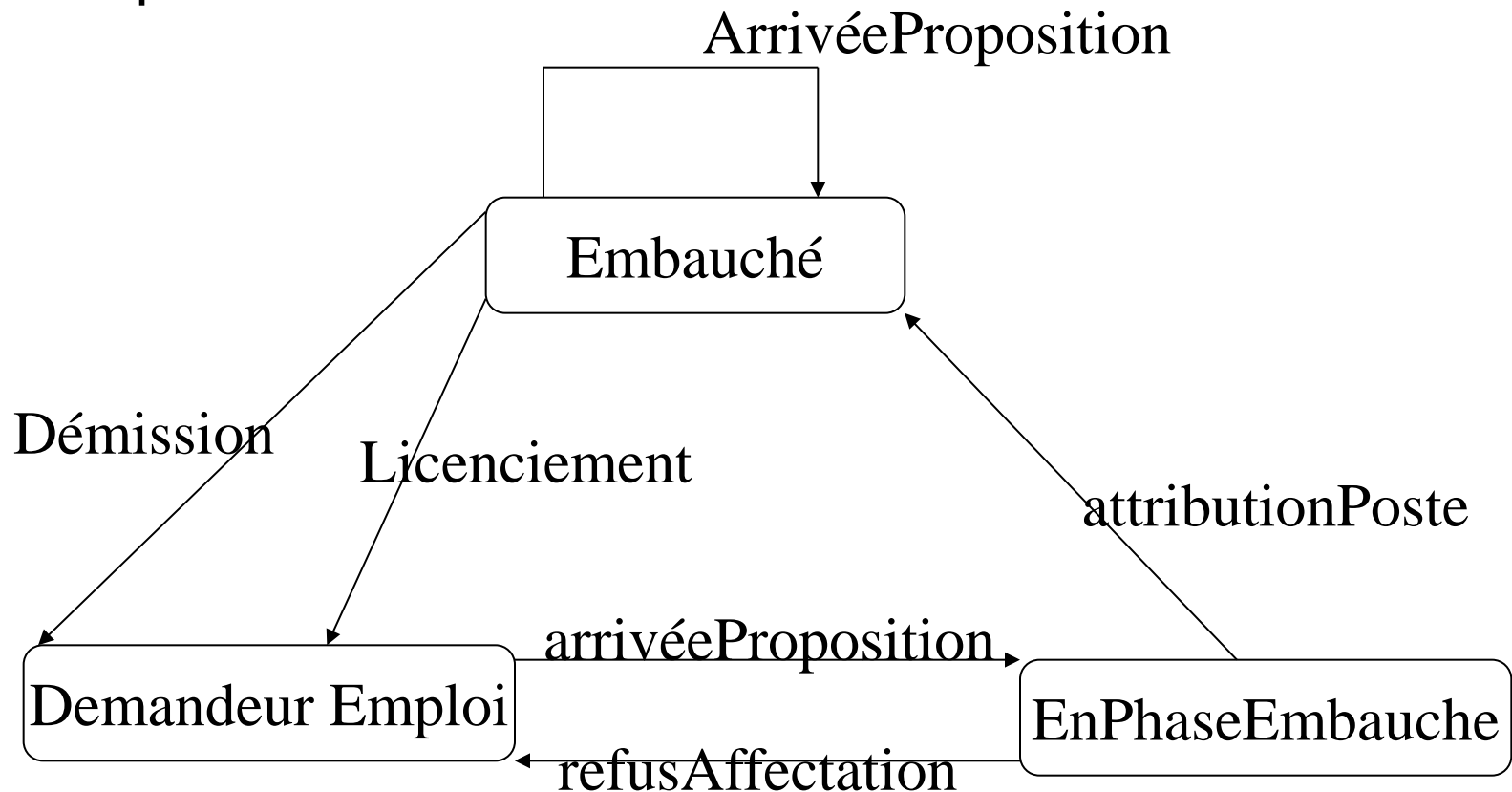
● 7.1.4 La notion de transition

-Une transition possède un déclencheur d'évènement, une condition de garde, un effet et un état cible

Exemple :

- le licenciement effectué par une société
- l'arrivée d'une proposition en provenance d'un organisme (ANPE, cabinet de recrutement)
- l'attribution d'un poste dans une société
- la démission décidée par l'employé

-Une transition est le changement d'état d'un objet causé par un évènement.



Type de transition	description	syntaxe
entry	Spécification d'une activité d'entrée qui s'exécute lorsqu'on saisit un état	entry/activity
exit	Spécification d'une activité de sortie qui s'exécute lorsqu'on quitte un état	exit/activity

Type de transition	description	syntaxe
transition externe	Réponse à un évènement qui engendre un changement d'état ainsi qu'un effet spécifié ou entry et/ou exit	(a:T) [guard] / activity
transition interne	Réponse à un évènement qui entraine l'exécution d'un effet mais pas d'un changement d'état, ni exit, ni entry	e(a:T) [guard]/activity

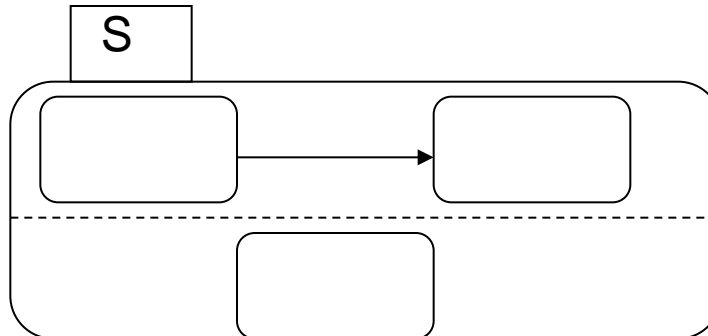
● 7.1.5 Etat composite

-Un état composite est un état décomposé en régions contenant chacune un ou plusieurs sous-états.

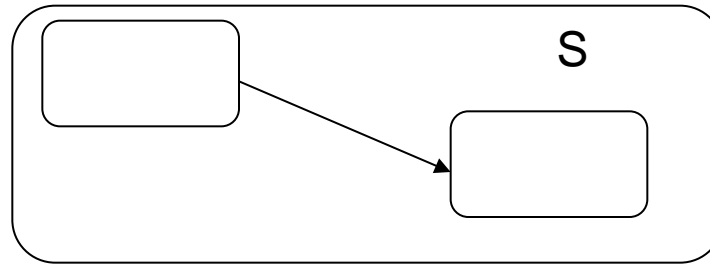
État simple



État orthogonal



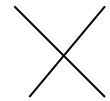
État non orthogonal



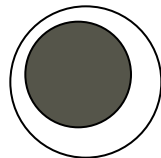
état initial



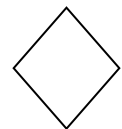
terminaison



état final



choix



- 7.16 Diagramme de machine d'états

- Il fait intervenir des évènements et des états, il est donc composé d'un ensemble de transition
- Les noeuds représentent les états
- Les flèches représentent les transitions
- Sur les flèches il y a les paramètres et les noms d'évènements

Imbrication d'états :

- Plusieurs sous-états peuvent être associés à un état

Exemple :

- l'état Embauché des objets Personne peut être précisé par 2 sous-états :

- AuTravail
- EnCongés

- Les attributs correspondent à des informations ou des paramètres portés par des évènements

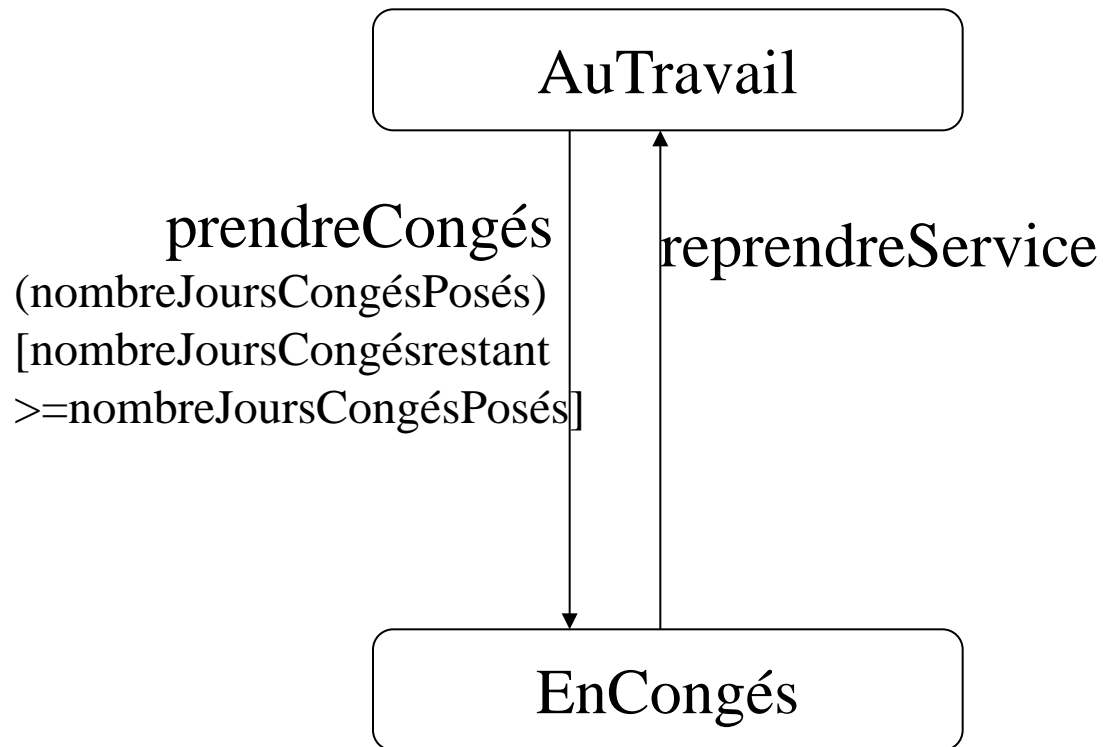
Ils sont représentés entre parenthèses après le nom de l'évènement

- Les conditions correspondent sont des fonctions booléennes

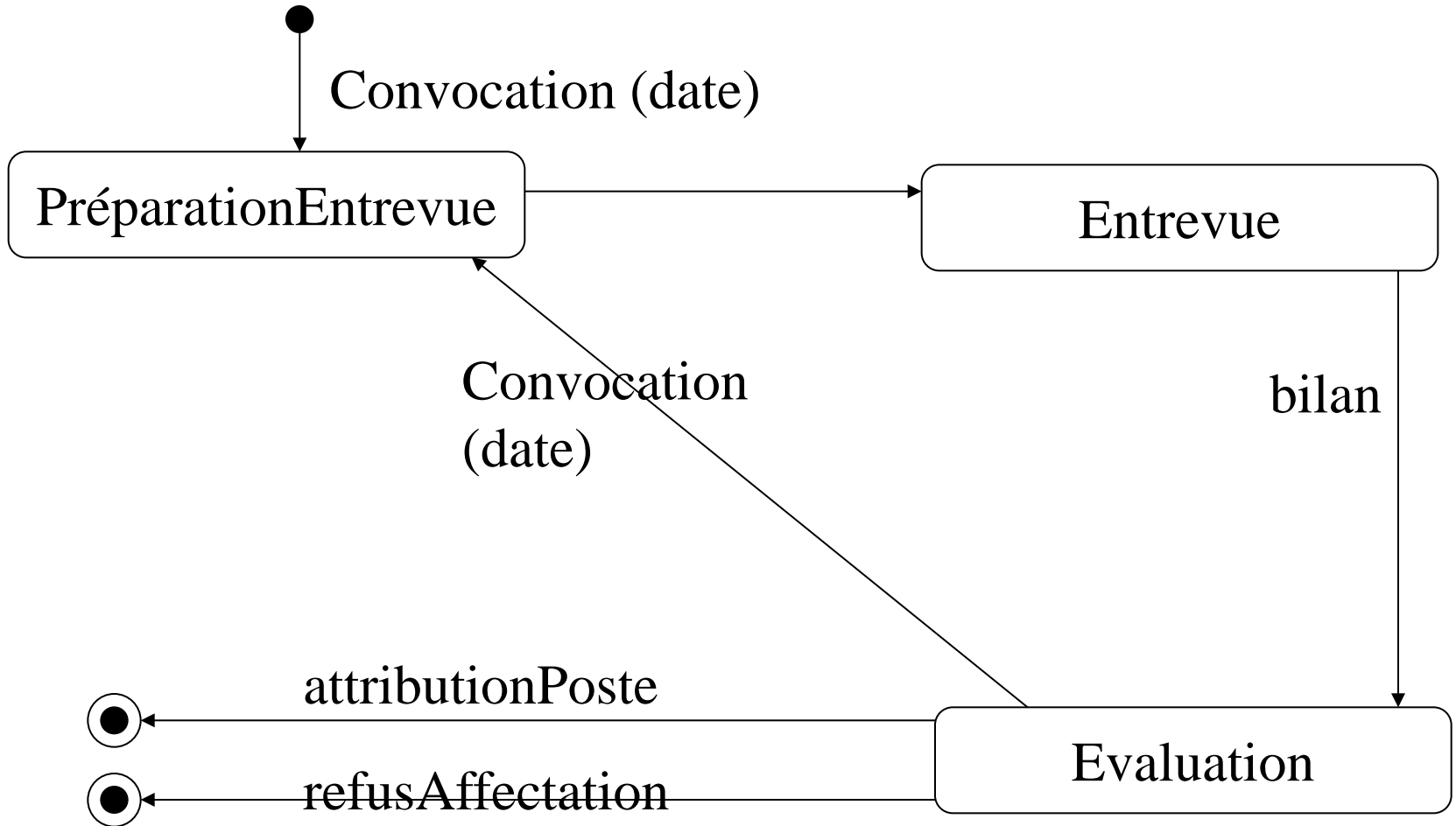
Ils sont représentés entre crochets après la liste des attributs. Une transition portant des conditions ne peut être effectuée que si la condition est vérifiée

-Il est propre à une classe donnée

PersonneEmbauché



Personne EnPhaseEmbauche



Etat1

-entry/ action en entrée

-do : activité pendant l'état

-évènement1 / action1

-évènement2 / action2

...

-exit/ action en sortie d'état

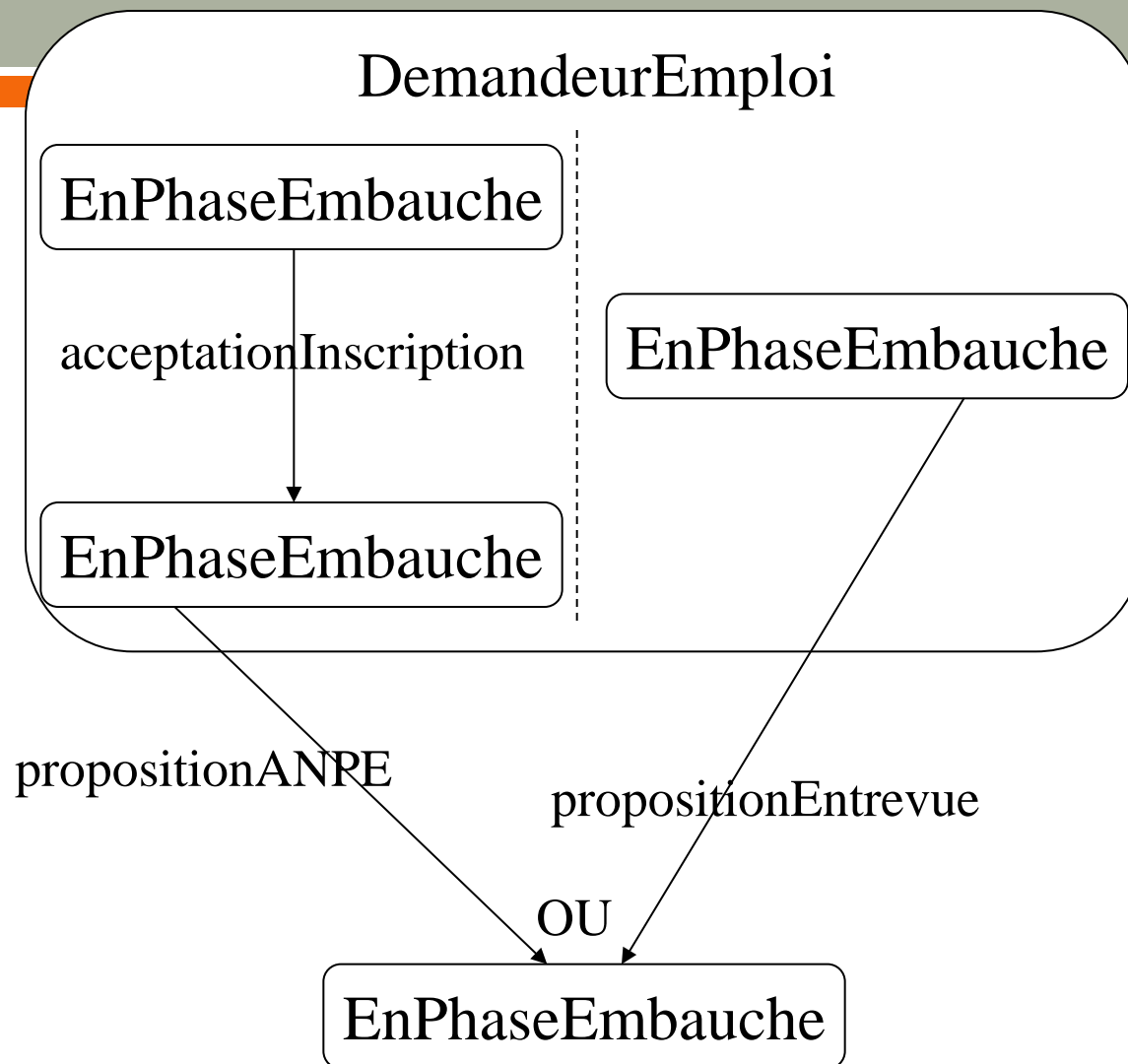
Évènement (attributs) [gardien]/action

Etat2

Exemple

Embauché

- entry**/ signer contrat de travail
- do** : assurer fonction
- arrivée proposition / répondre à la proposition
- mutation / changer d'affectation
- exit**/ rompre contrat de travail



7.2 Vue d'activités

7.2.1 Présentation

7.2.2 Les nœuds d'activité

7.2.3 Le flot des objets

7.2.4 Diagramme d'activités

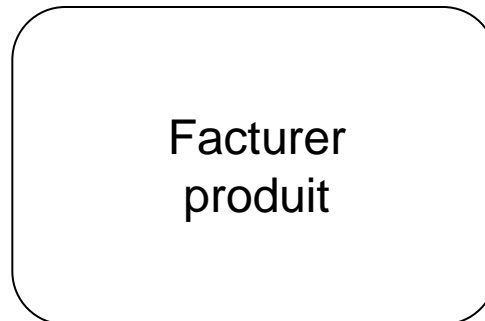
● 7.2.1 Présentation

- Une activité est un graphique de noeuds et de flots qui matérialise le flot de contrôle à travers les étapes de calcul
- Lorsqu'un objet réagit à un évènement il déclenche une réponse à cet évènement une ou plusieurs opérations appelées activités

- 7.2.2 Les nœuds d'activité

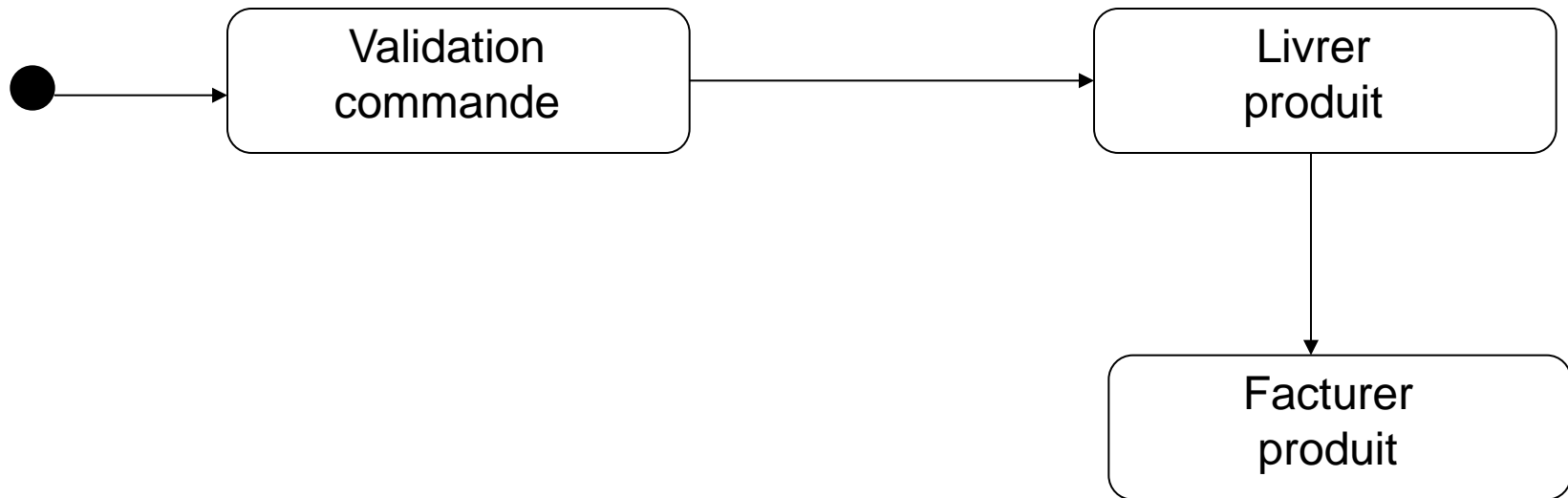
Un noeud d'activité :

- opération continue dans le temps
- elle prend un certain temps pour se réaliser

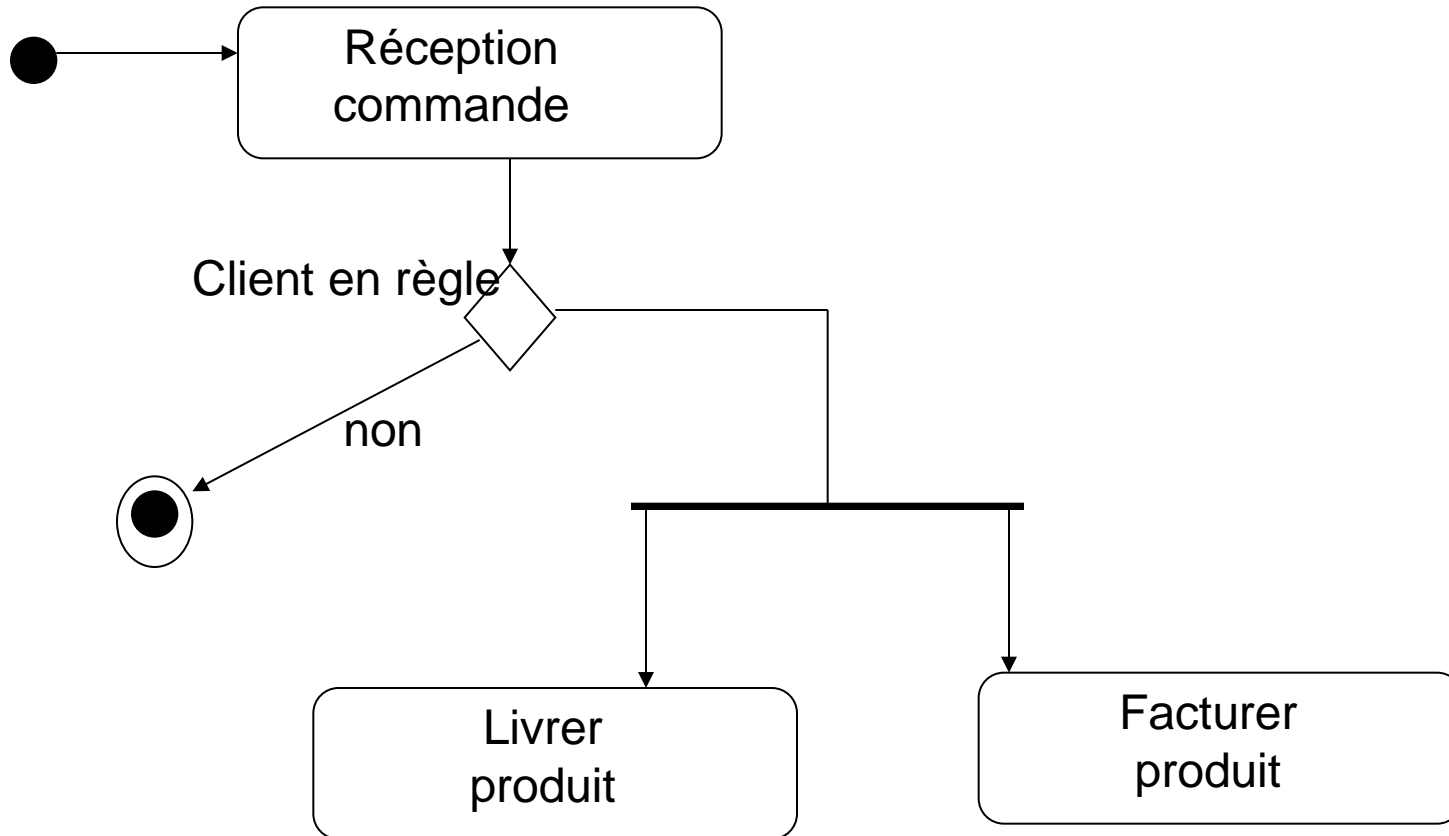


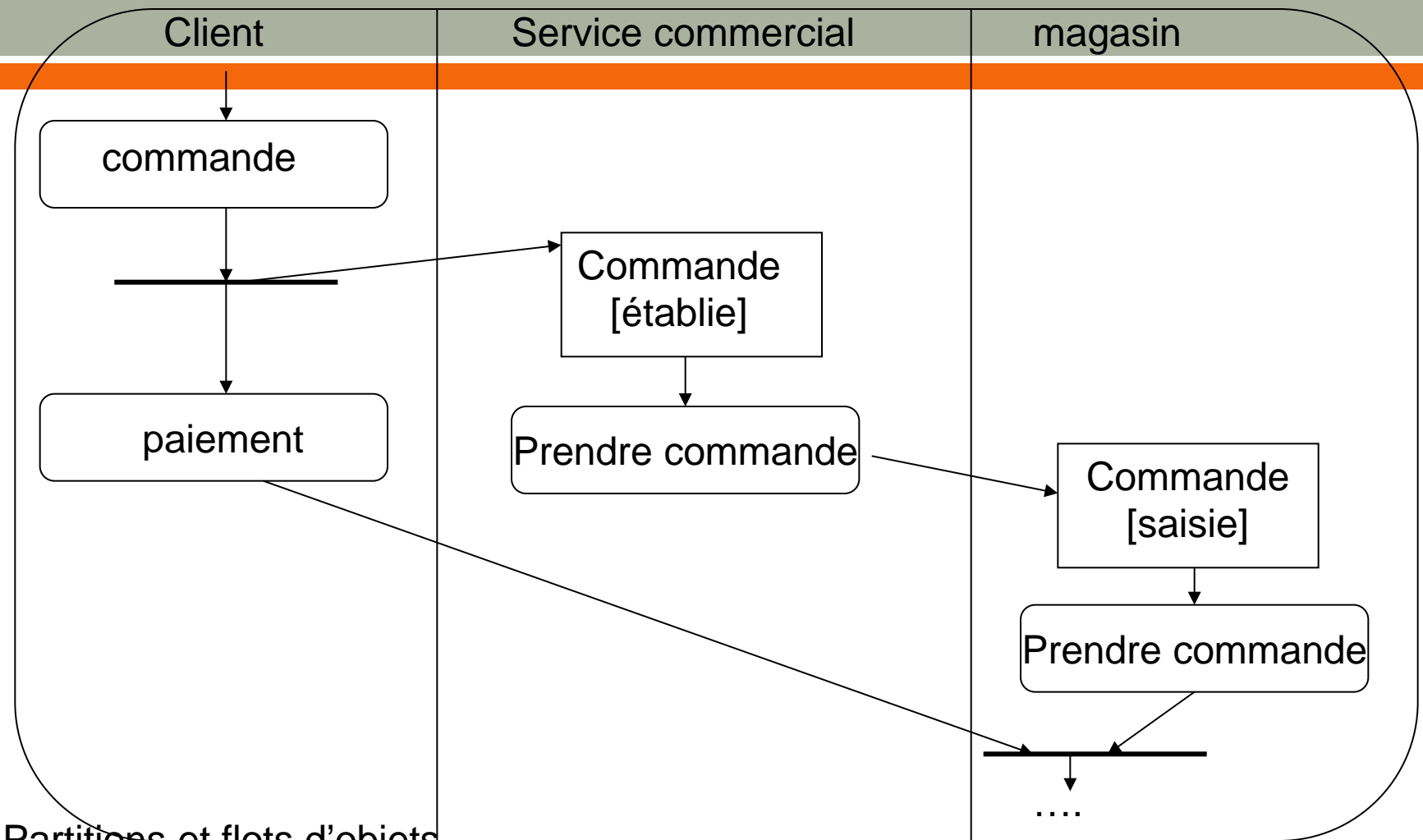
- 7.2.3 Le flot des objets

Un flot d'objet, représente un objet qui se trouve en entrée ou en sortie d'activité



● 7.2.4 Diagramme d'activités





Partitions et flots d'objets

7.3 Vue des interactions

7.3.1 Diagramme de séquence

7.3.2 Opérations sur les objets

7.3.3 Activations

- 7.3.1 Diagramme de séquence

- Représentent des interactions entre objets de manière temporelle

- Représentation d'un objet : objet + ligne de vie de l'objet

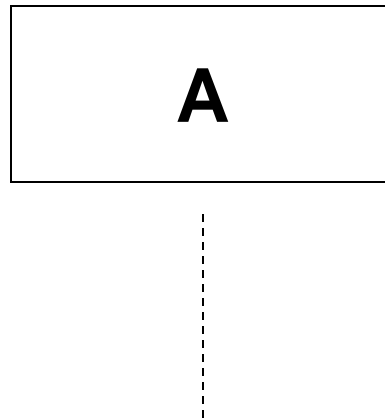


Diagramme de séquence

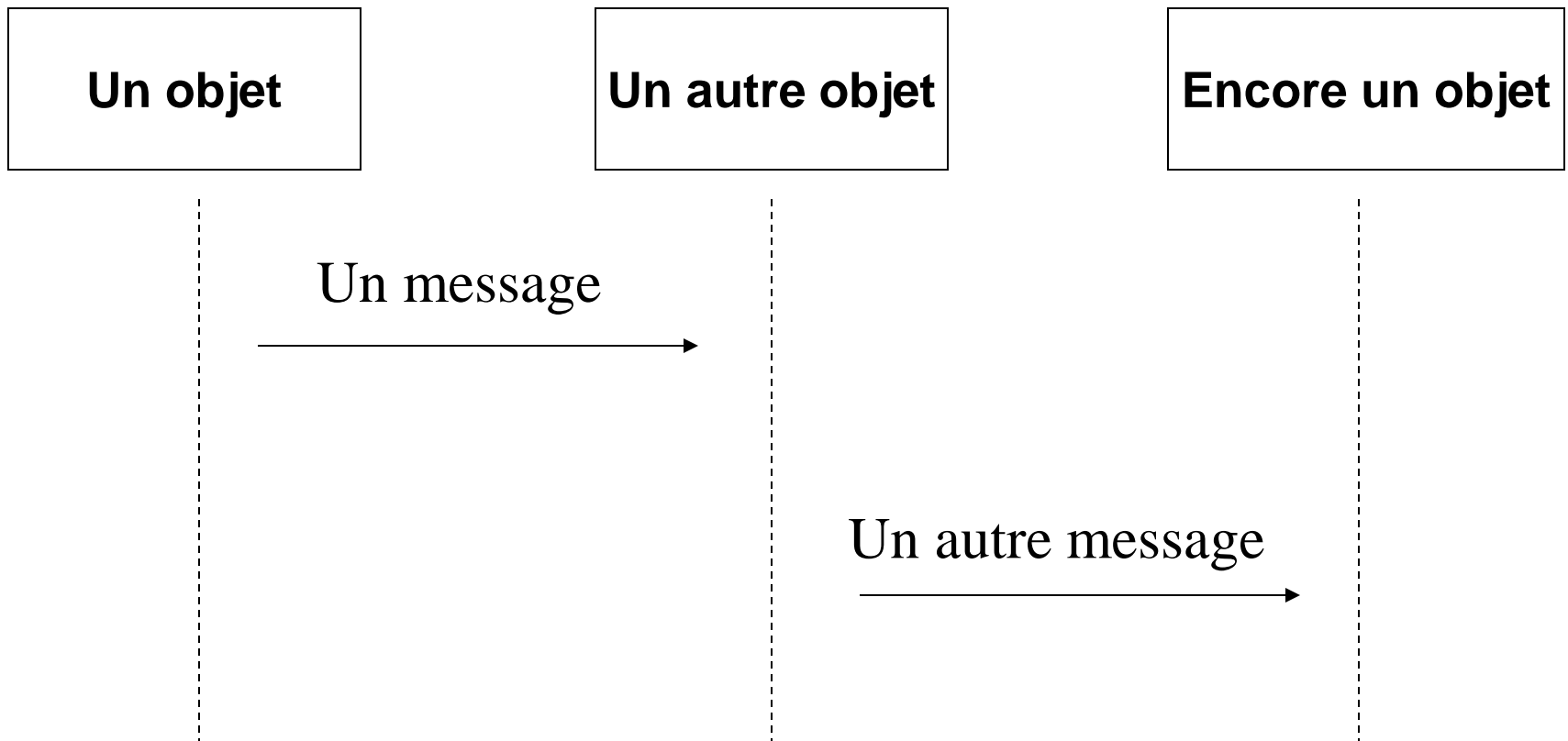
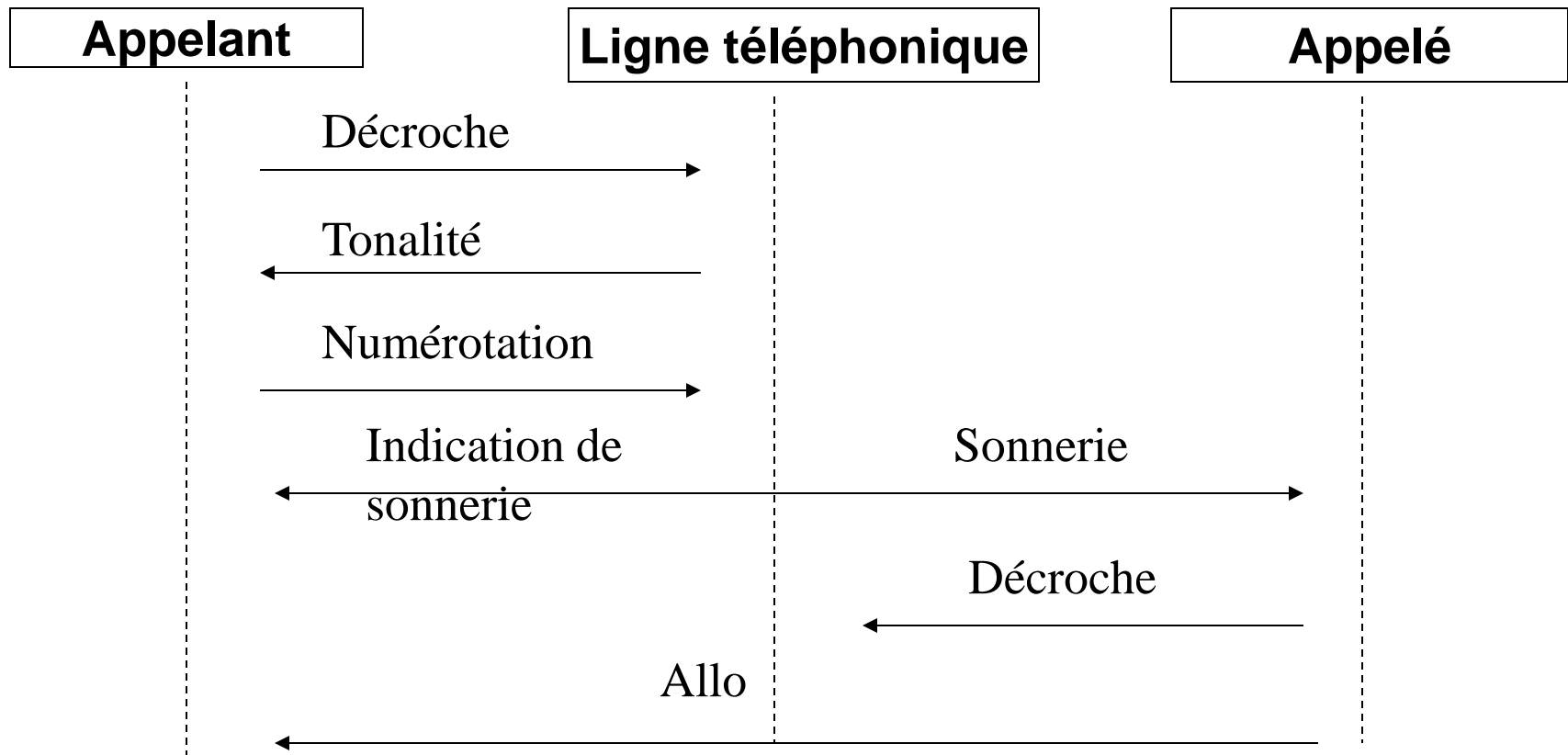
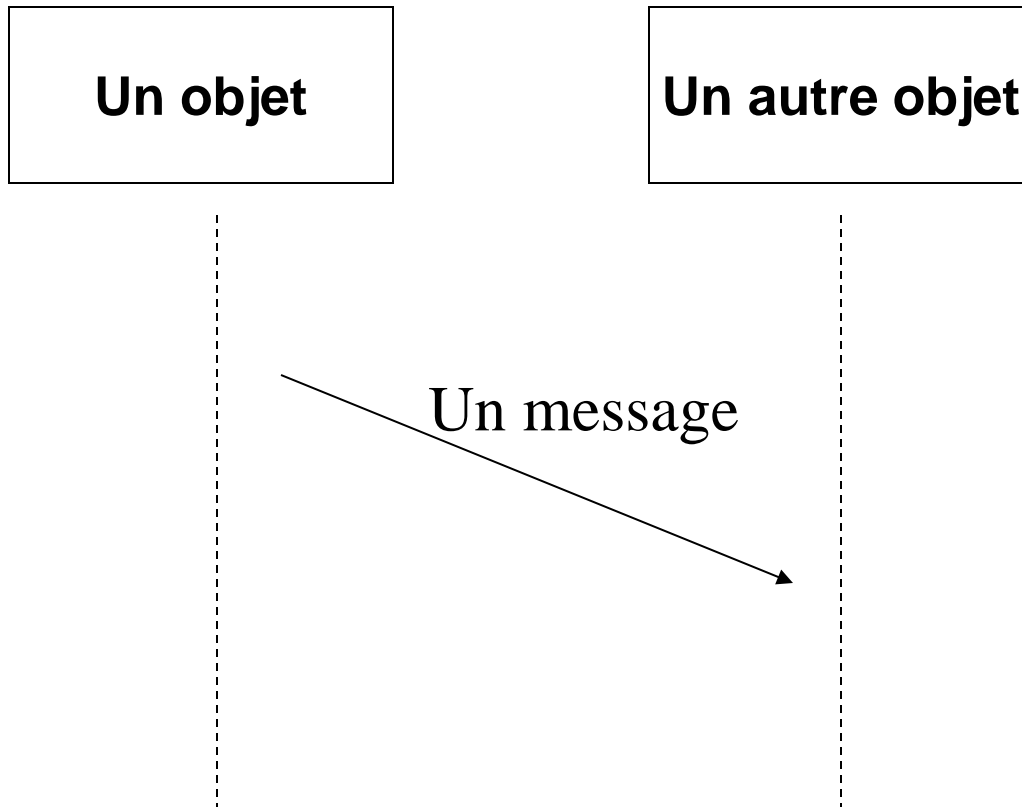


Diagramme de séquence utilisé pour documenter des cas d'utilisation



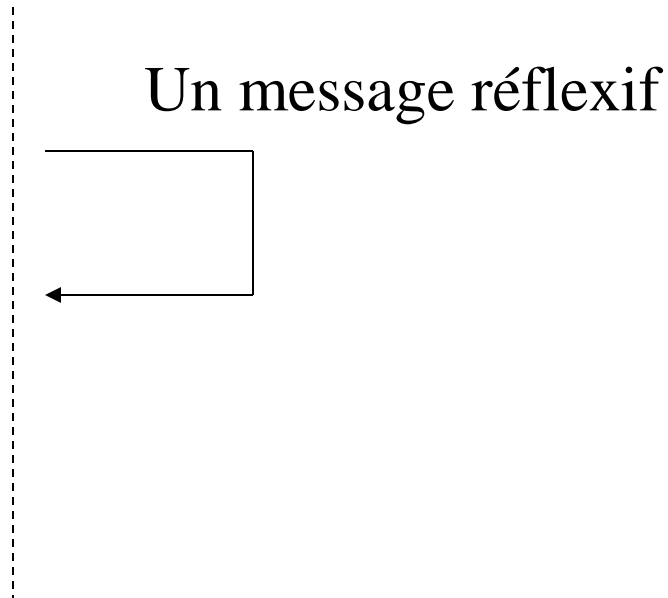
La flèche peut être représentée en oblique pour représenter un délai de transmission



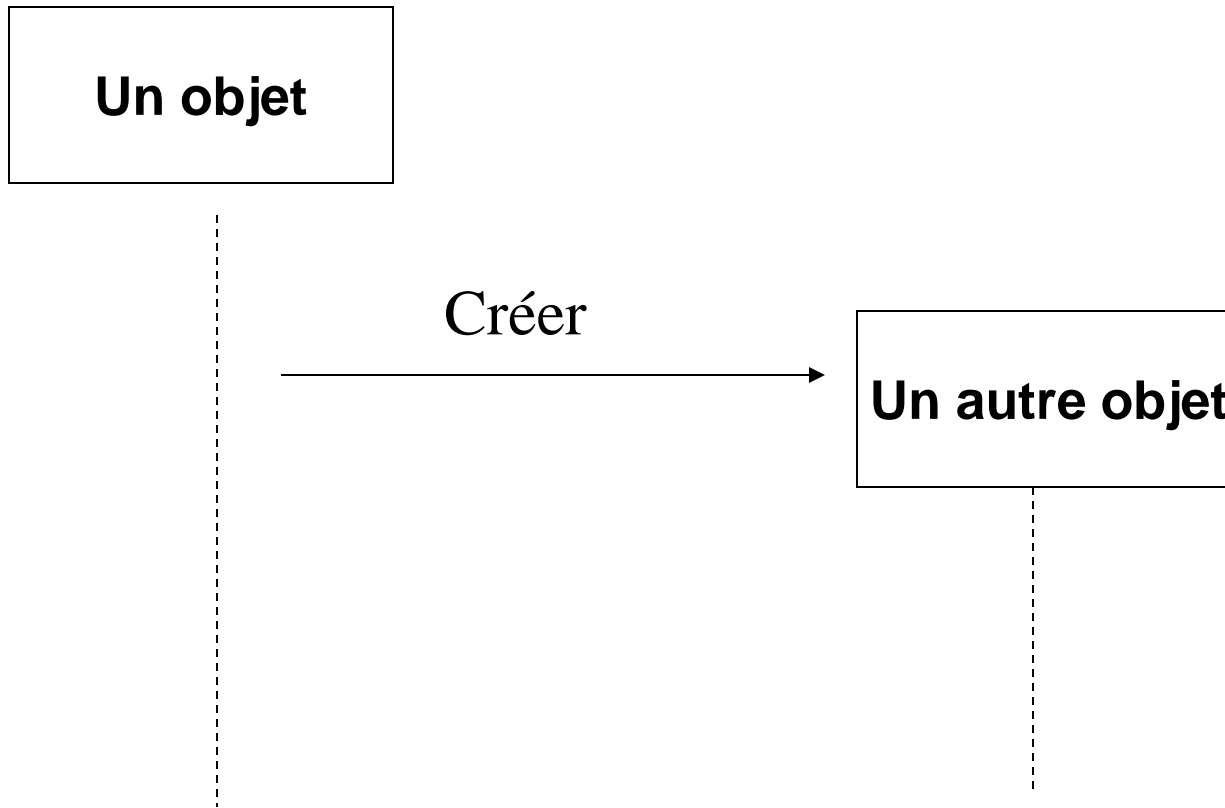
- 7.3.2 Opérations sur mes objets



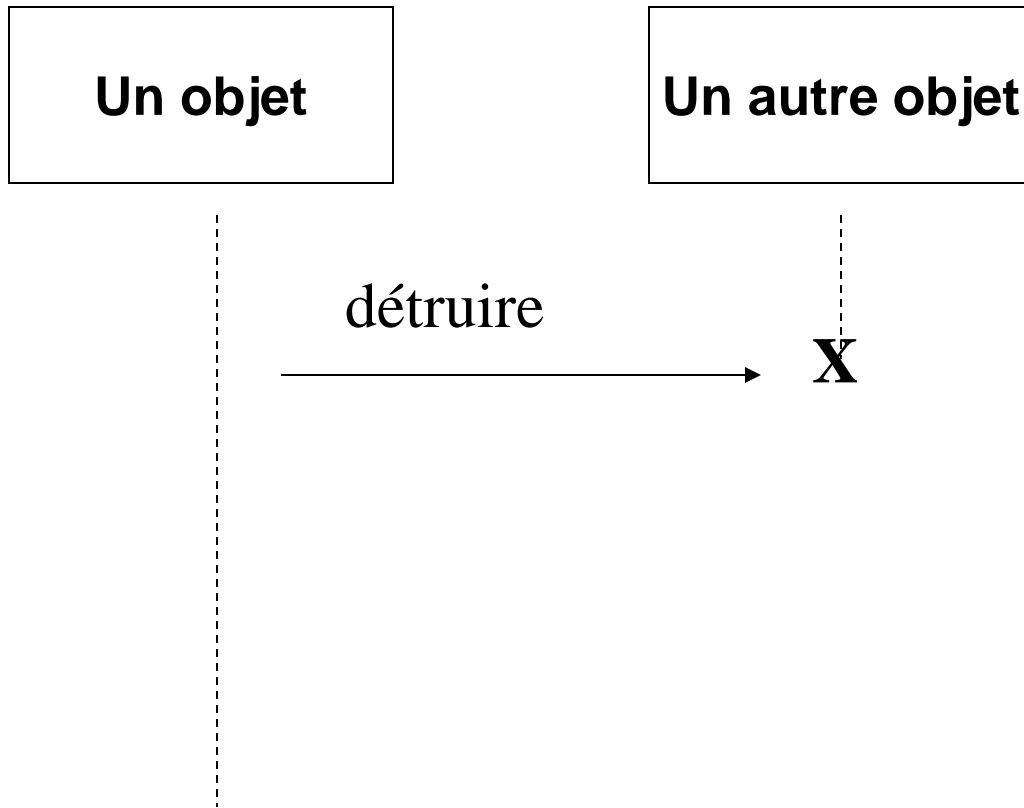
**Un objet peut s'envoyer
un message**



Création d'un objet

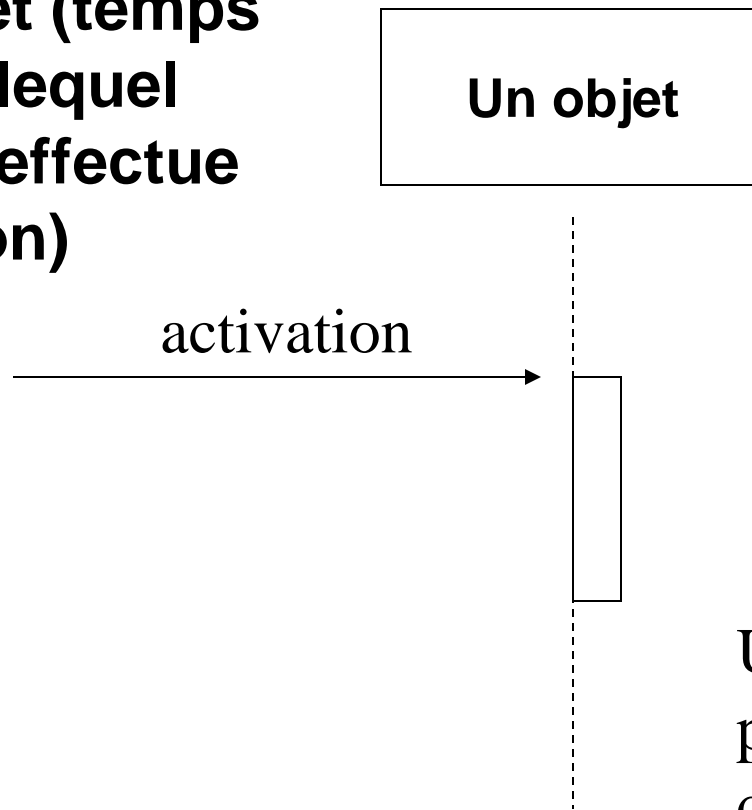


Destruction d'un objet



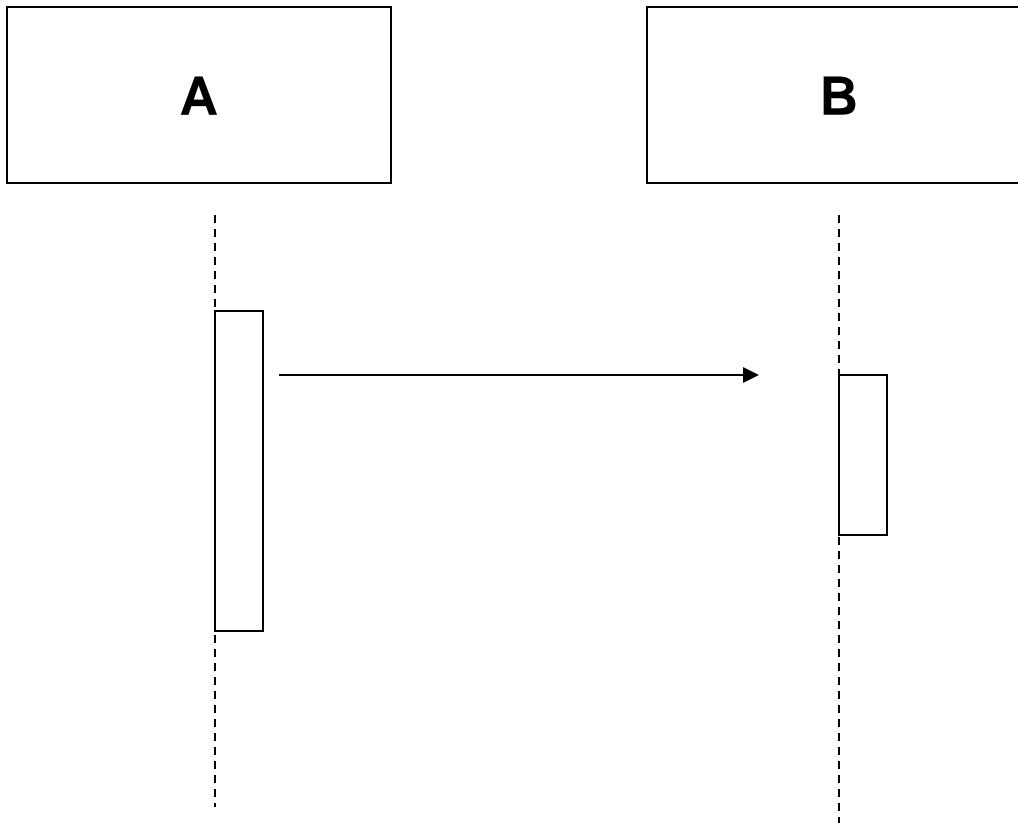
● 7.3.3 Activations

**Période d'activité
d'un objet (temps
pendant lequel
un objet effectue
une action)**



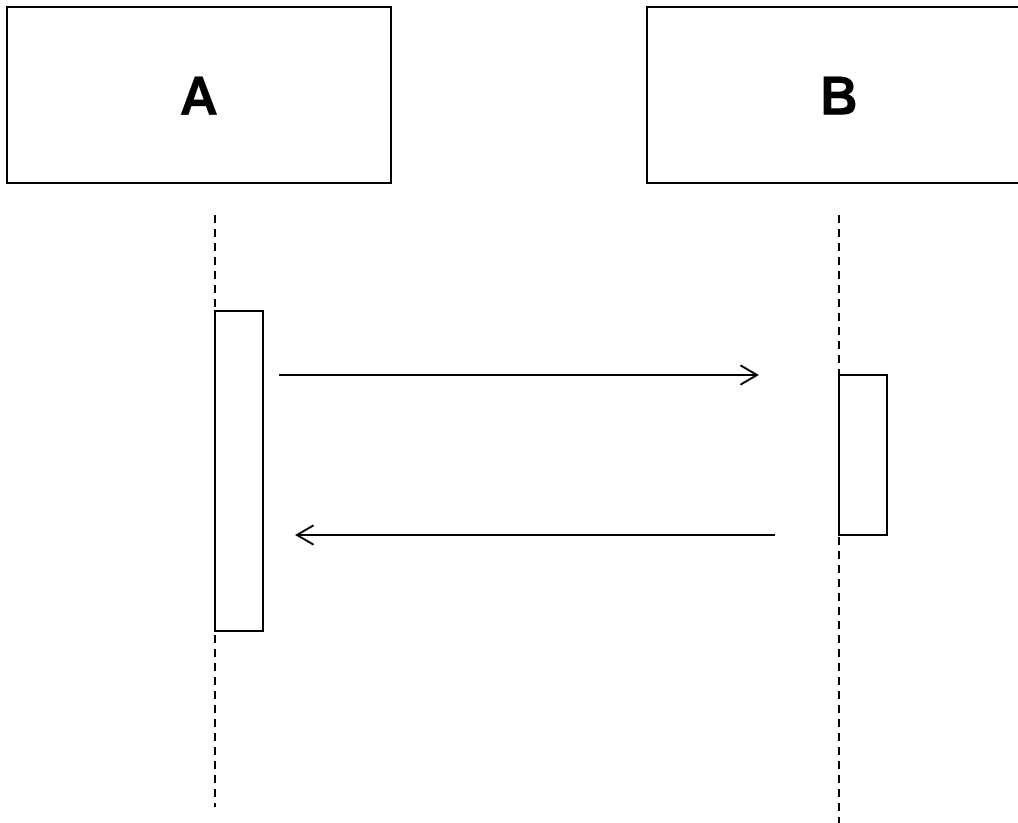
Une bande rectangulaire
placée sur la ligne de vie
de l'objet

Un objet A qui active un objet B



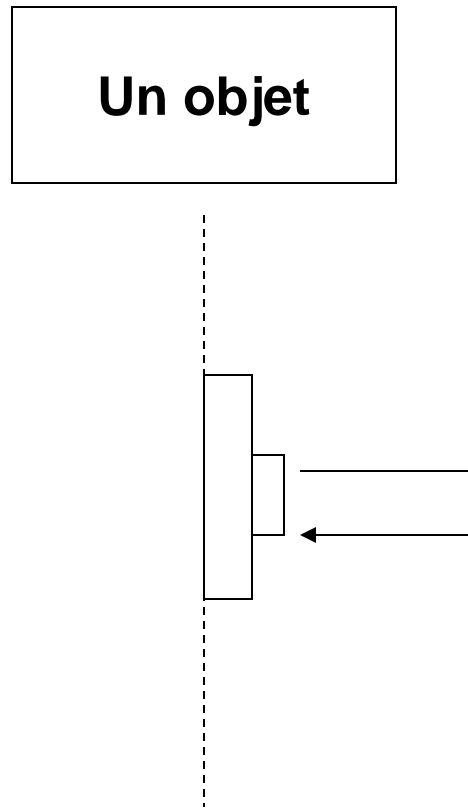
Si l'envoi est
synchrone, le
retour est implicite
en fin d'activité

Un objet A qui active un objet B

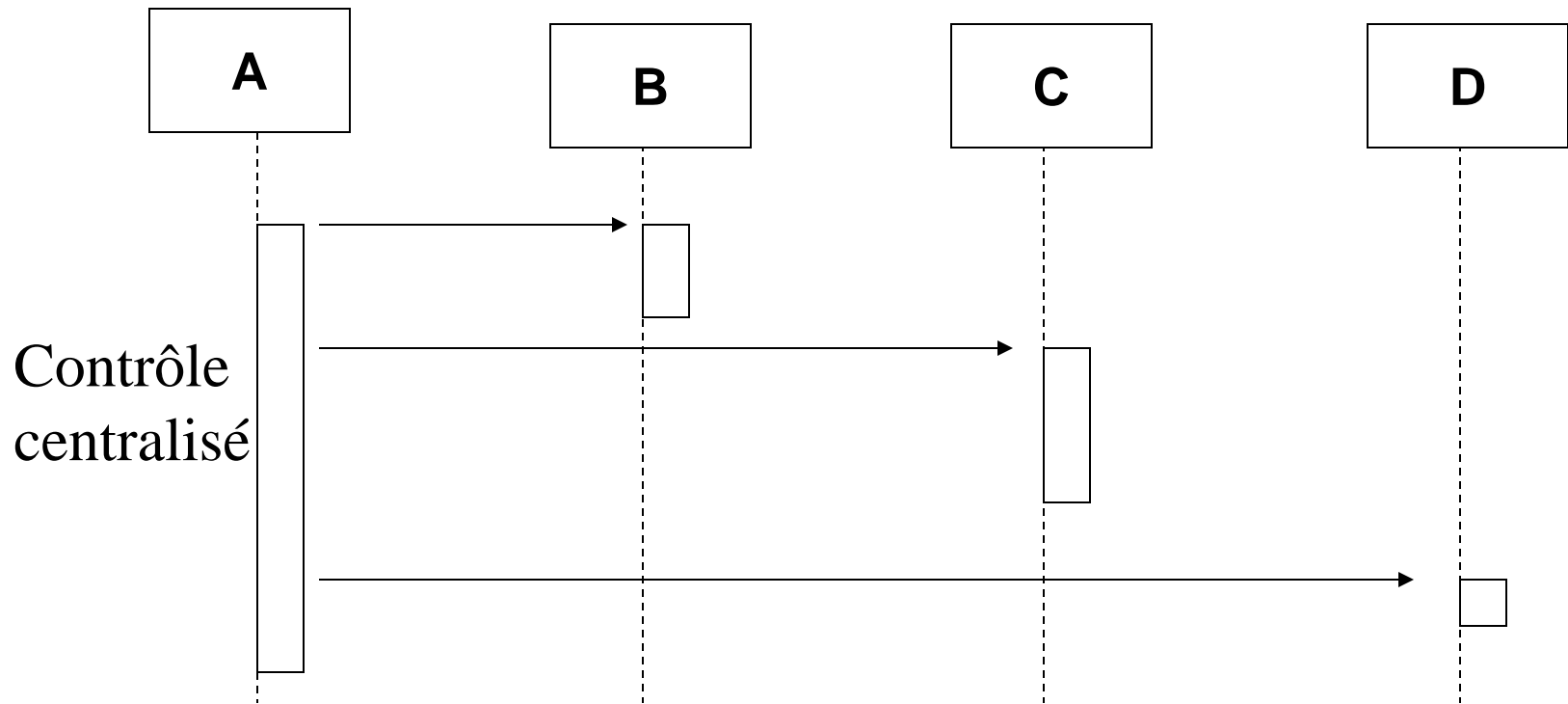


Si l'envoi est asynchrone, le retour doit être représenté s'il existe

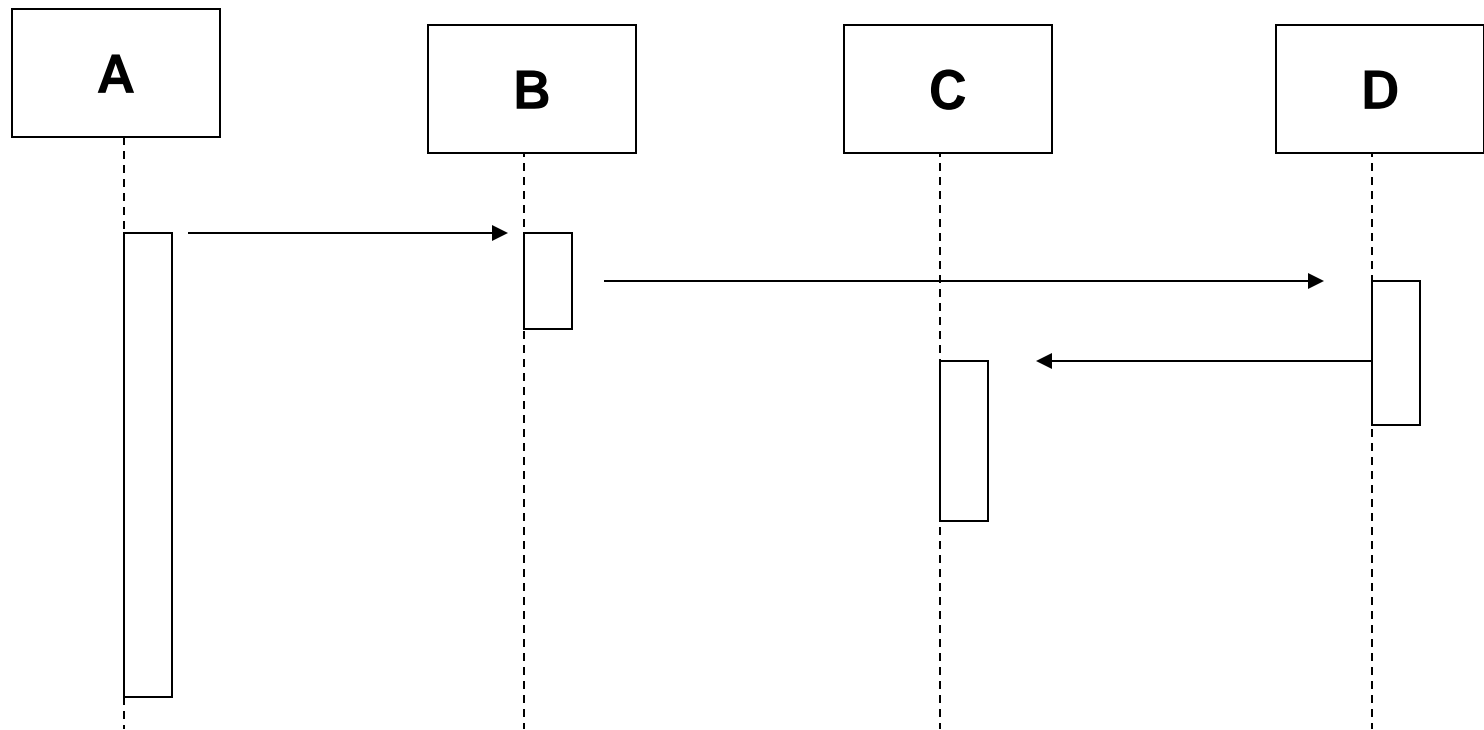
récurtivité



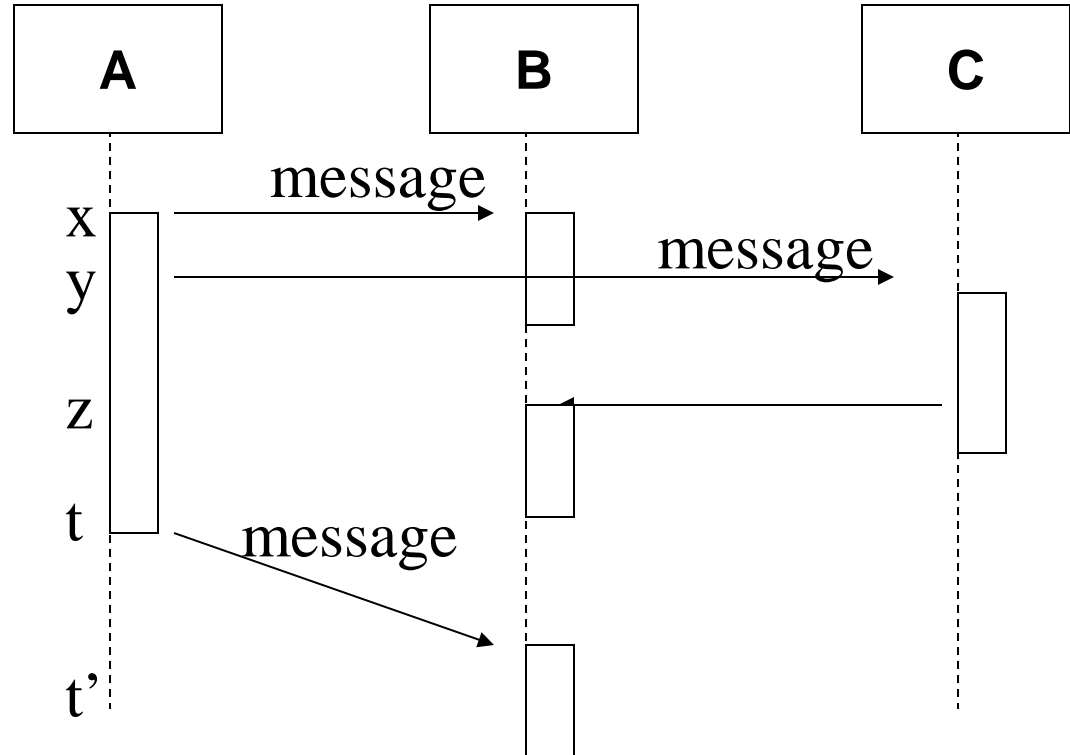
La forme du diagramme de séquence reflète le mode de contrôle de l'interaction



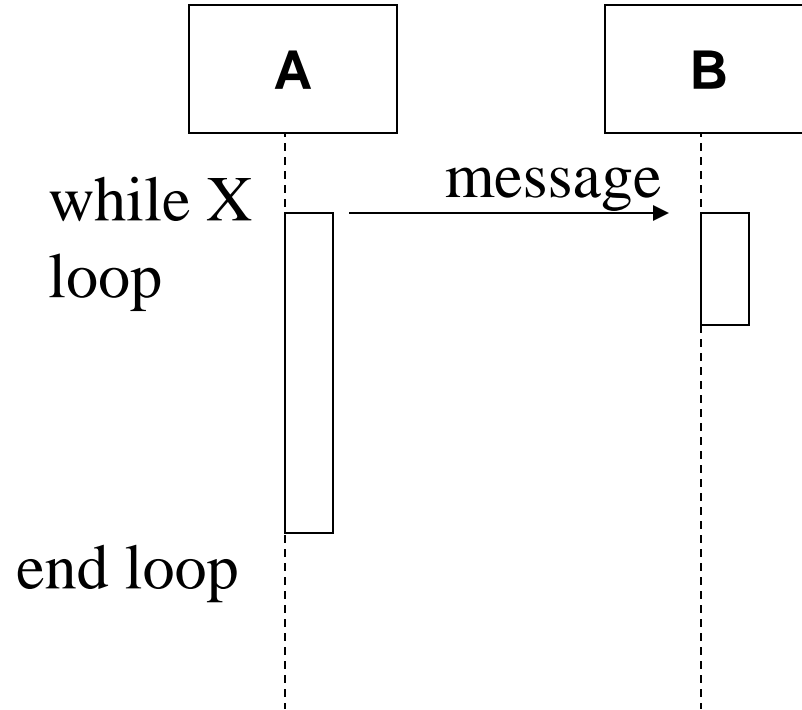
Contrôle décentralisé



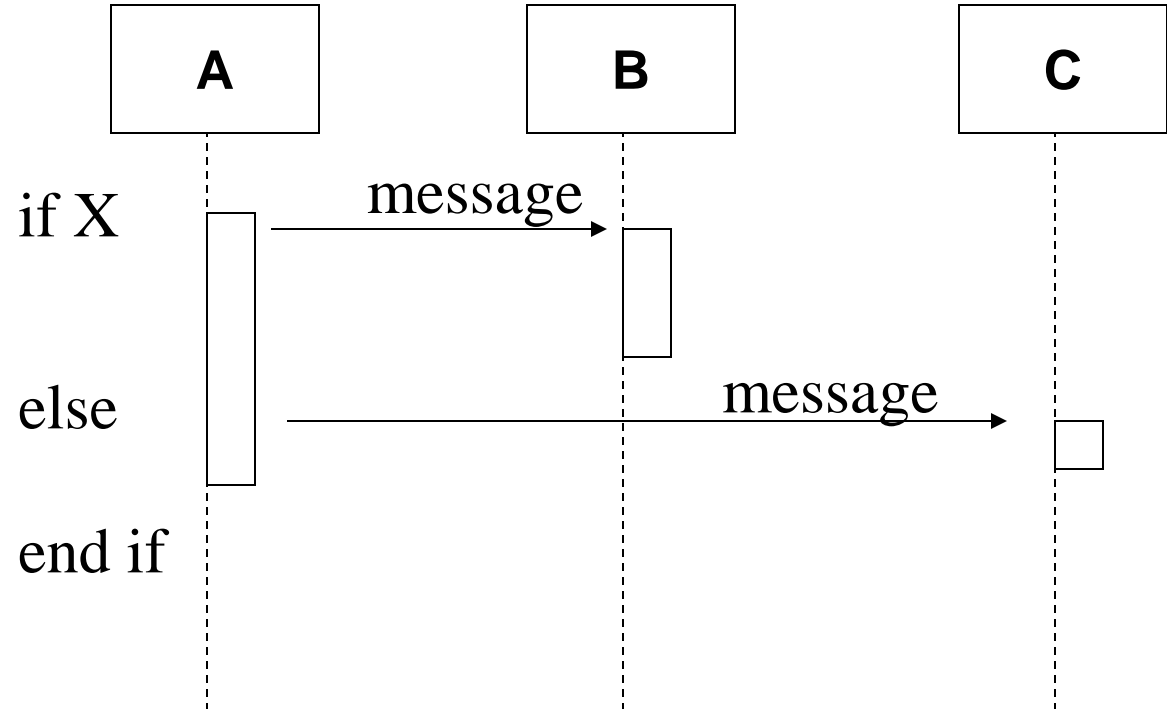
Transition : instant d'émission d'un message



Boucle While

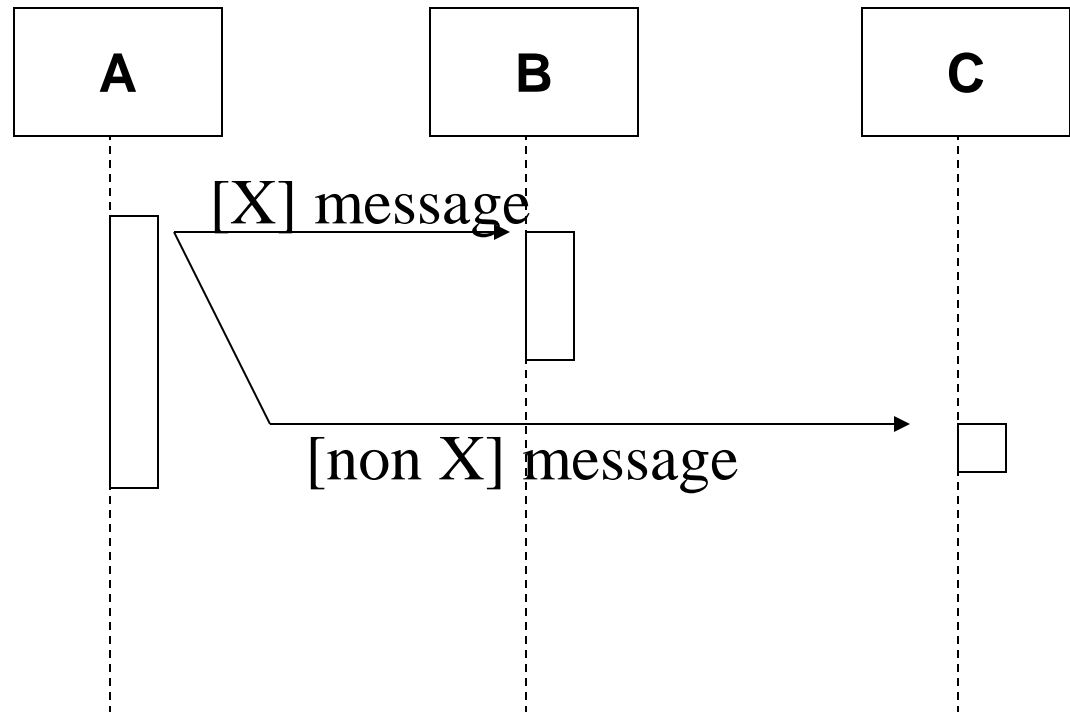


Branchement conditionnel



Branchement conditionnel

Soit



programme

VIII La vue physique sous UML

8.1 Présentation

8.2 Les noeuds

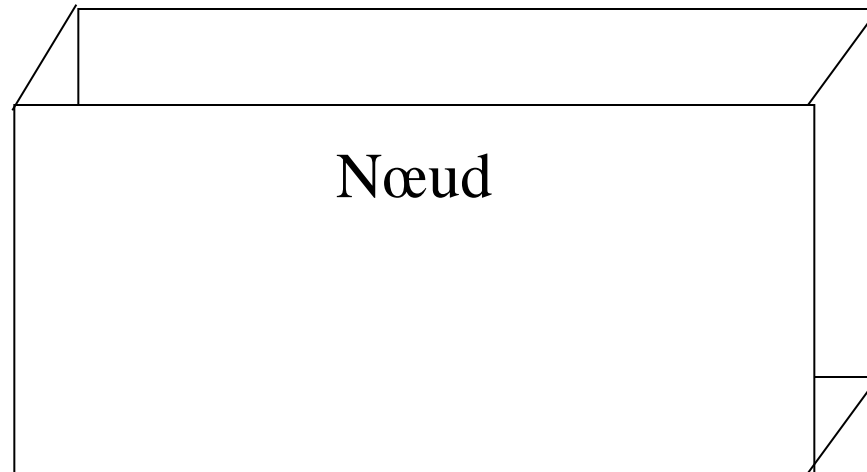
● 8.1 Représentation

- Ces diagrammes montrent la disposition physique des différents matériels (noeuds)
- Dans un environnement multi-tache définit :
 - les tâches : la décomposition entere de processus
 - Les communications : interaction entre processus
 - La parallelisation : synchronisation

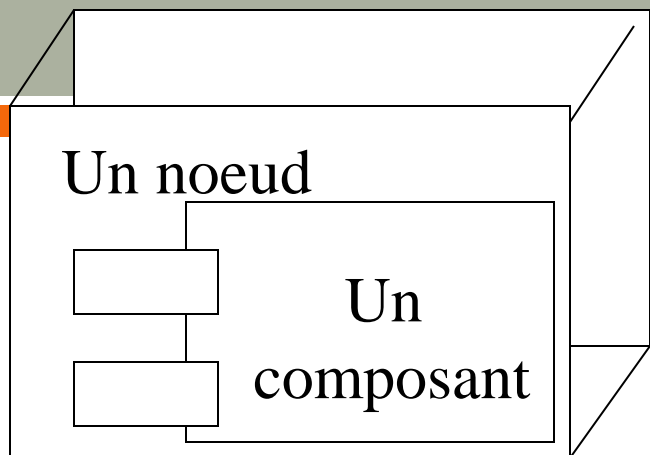
- 8.2 Noeuds

- Chaque ressource matérielle est représentée par un noeud

- Un noeud est représenté par un cube dessiné en 3D

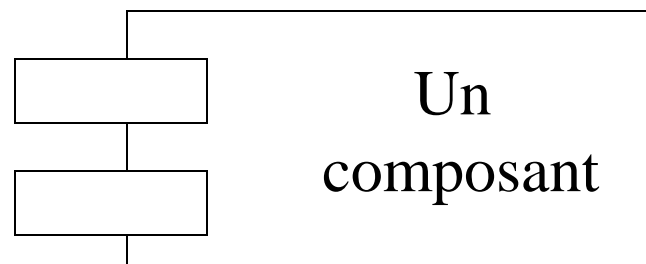


- ✎ Un nœud est défini comme une sous-classe de **Classificateur**
- ✎ Il peut aussi avoir des attributs et participer à des relations
- ✎ Pour montrer qu'un composant réside sur un nœud, 2 possibilités :
 - Un symbole de composant est emboîté dans le symbole du nœud
 - Une dépendance avec le mot clé « **support** » orientée du composant vers le noeud



« support »

A dashed arrow points from the component diagram below to the node diagram above, indicating a relationship or support.



programme

IX Vocabulaire

Domaine Structurel

Vue statique

Diagramme de classes

Association, classe, dépendance,
généralisation, interface

Vue de conception

Structure interne

Connecteur, interface, partie, rôle

Diagramme de collaboration

Connecteur, collaboration, rôle

Diagramme de composants

Composant, dépendance, interface

Vue de cas d'utilisation

Diagramme de cas d'utilisation

Acteur, association, extension,
inclusion, cas d'utilisation,

Dynamique

Vue de machine d'états

Diagramme de machines d'états

Complétude, transition, activité, effet,
événement, état, transition, déclencheur

Vue d'activité

Diagramme d'activités

Action, activité, contrôle des flots,
nœud de contrôle, flots de données, exception,
expansion, région, débranchement, jonction,
nœud objet, pin

Dynamique

Vue ensemble des interactions

Diagramme de séquence

Spécifications des occurrences, de l'exécution, interaction, ligne de vie, message, signal

Diagramme de communication

Collaboration, condition de garde, message, rôle, numéro de séquence

Physique

Vue de déploiement

Diagramme de déploiement

Artefact, dépendance, manifestation, noeud

Gestion du modèle

Vue de gestion du modèle

Diagramme de package

Importation, modèle package

Profil

Diagramme de package

Contrainte, profil, stéréotype,
valeur étiquetée

programme

X Dérivation du diagramme de classes

10.1 Vers un schéma relationnel

10.2 Vers une BDOO native

10.1 Diagramme de classes UML vers schéma relationnel



A Les classes

- Toute classe devient une relation
- Identificateur
 - En objet : identifiant interne et non explicité sur le diagramme de classes
 - Il faut déterminer une propriété ou un ensemble de propriétés qui vont jouer le rôle d'identifiant naturel et donc de clé de la relation.
 - Si il n'y en a pas, on rajoute un attribut de type compteur qui joue le rôle d'identifiant artificiel

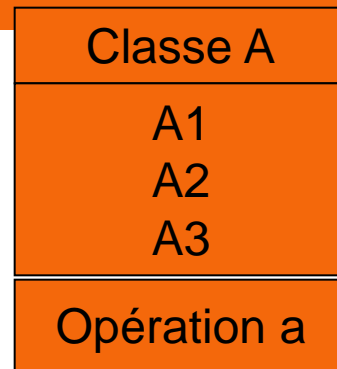
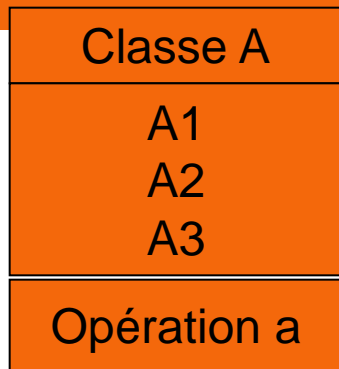
Classe A
A1 A2 A3
Opération a



entité A
Id A A1 A2 A3

B Les agrégations

- On remplace les agrégations par des associations
- On rajoute les multiplicités

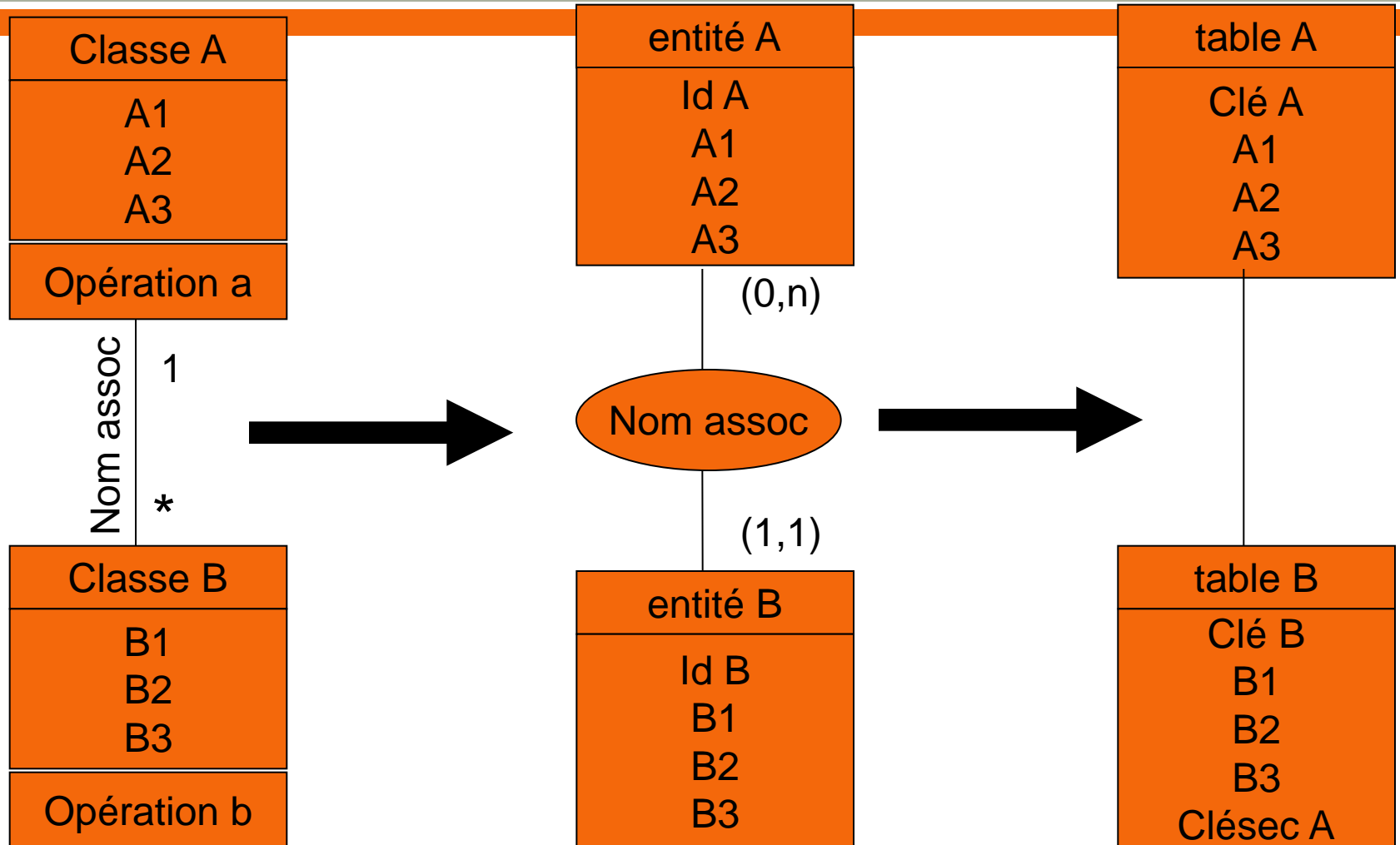


C Les associations (ou classe association)

- Toute association ou classe-association devient une relation
- Si multiplicité 1 ou 0..1 la relation est supprimée et l'on ne fait qu'une entité
- On fait migrer l'attribut ou les attributs identifiants (+ propriétés de l'association si il y en a) avec contrainte de clé étrangère

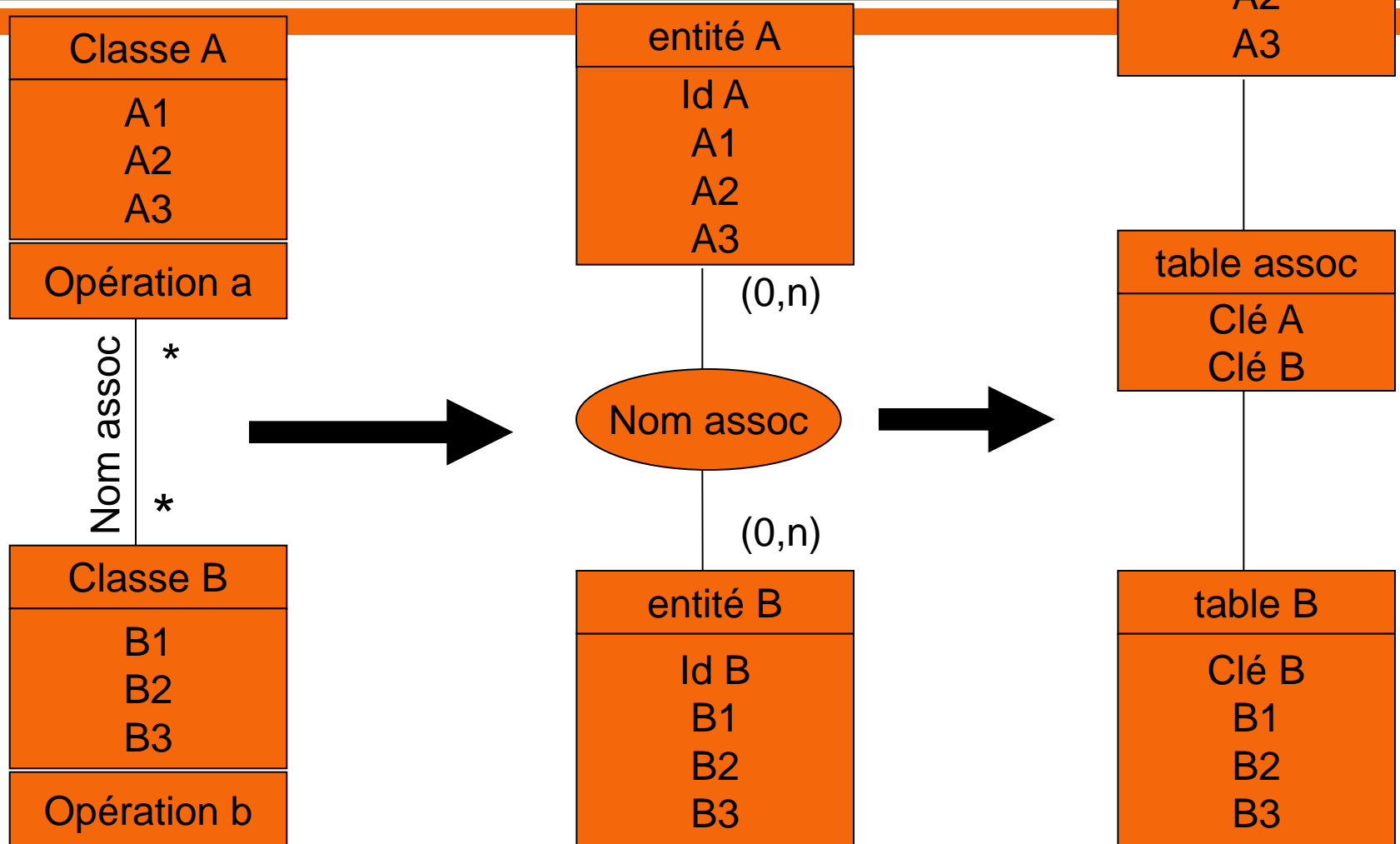
C Les associations (ou classe association)

Cardinalité 1 *



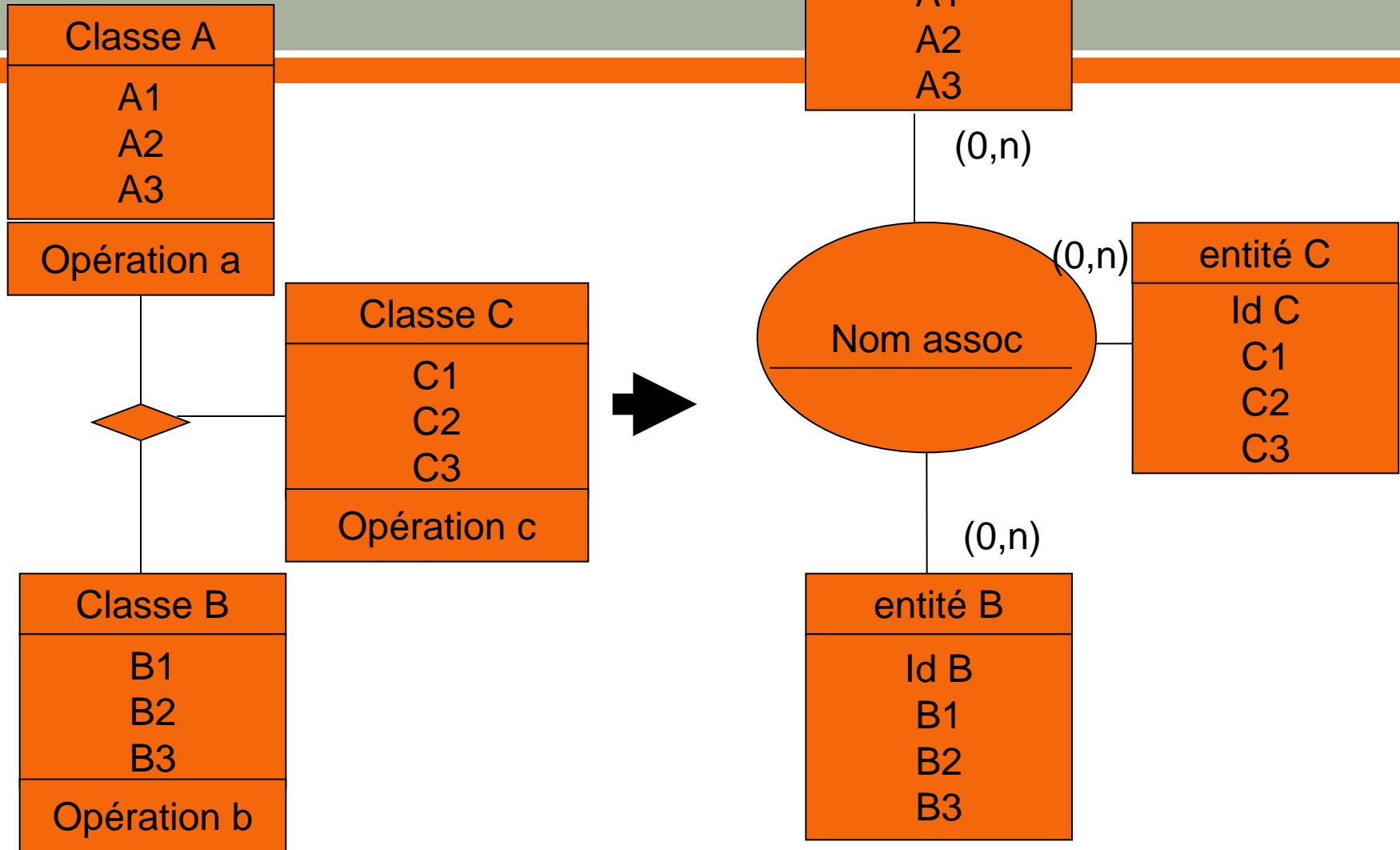
C Les associations (ou classe association)

Cardinalité * *

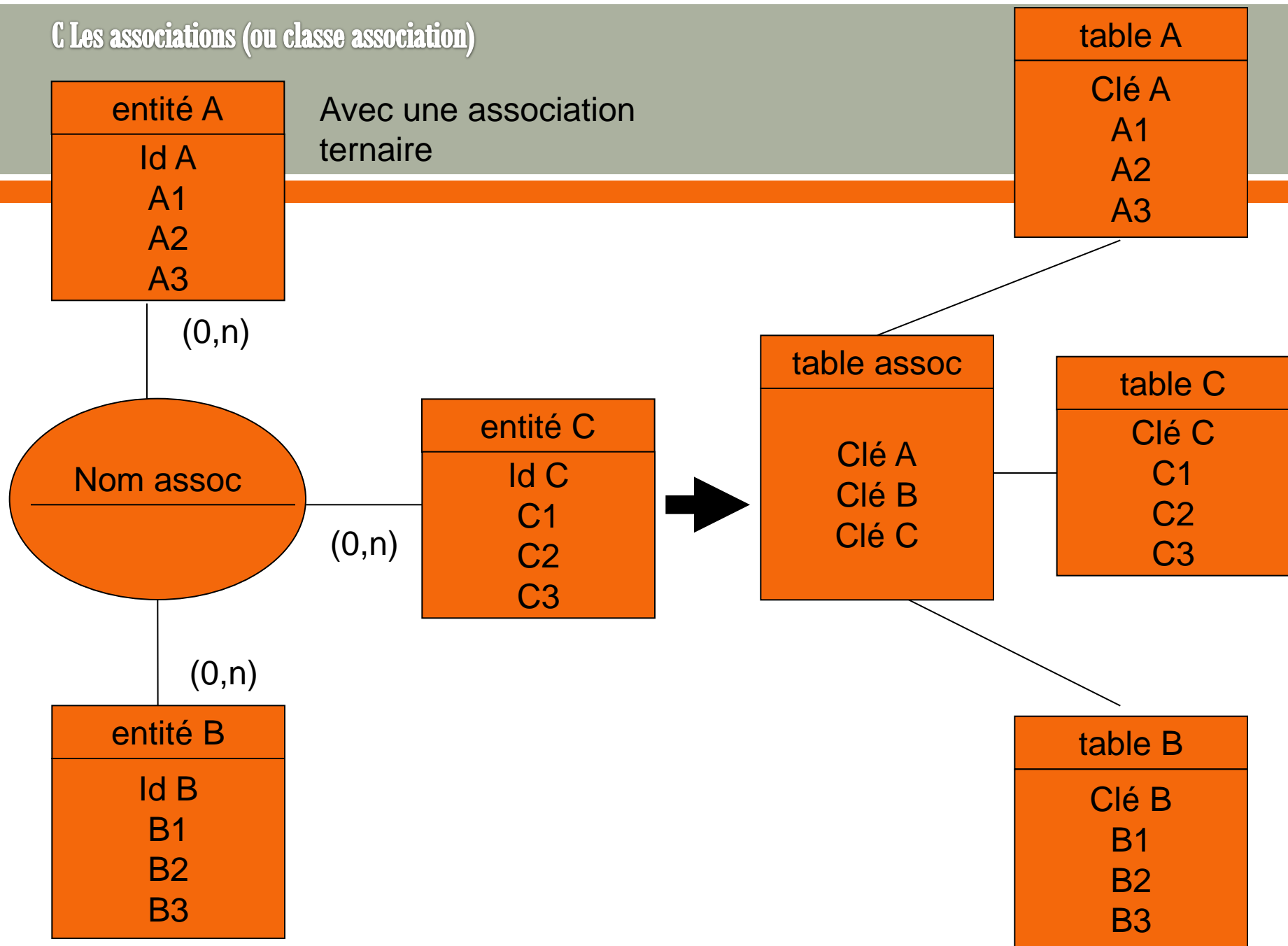


C Les associations (ou classe association)

Avec une association ternaire

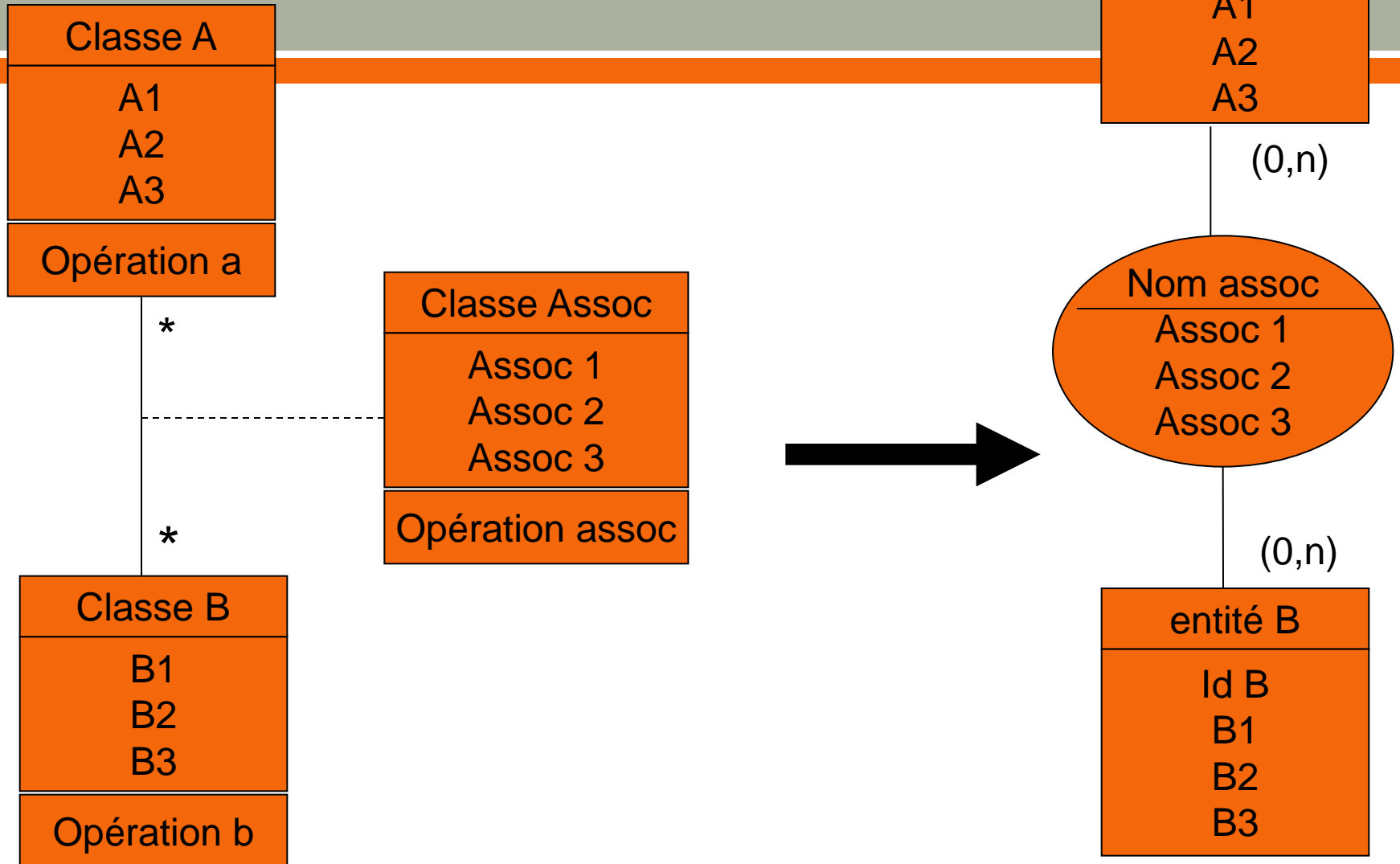


C Les associations (ou classe association)

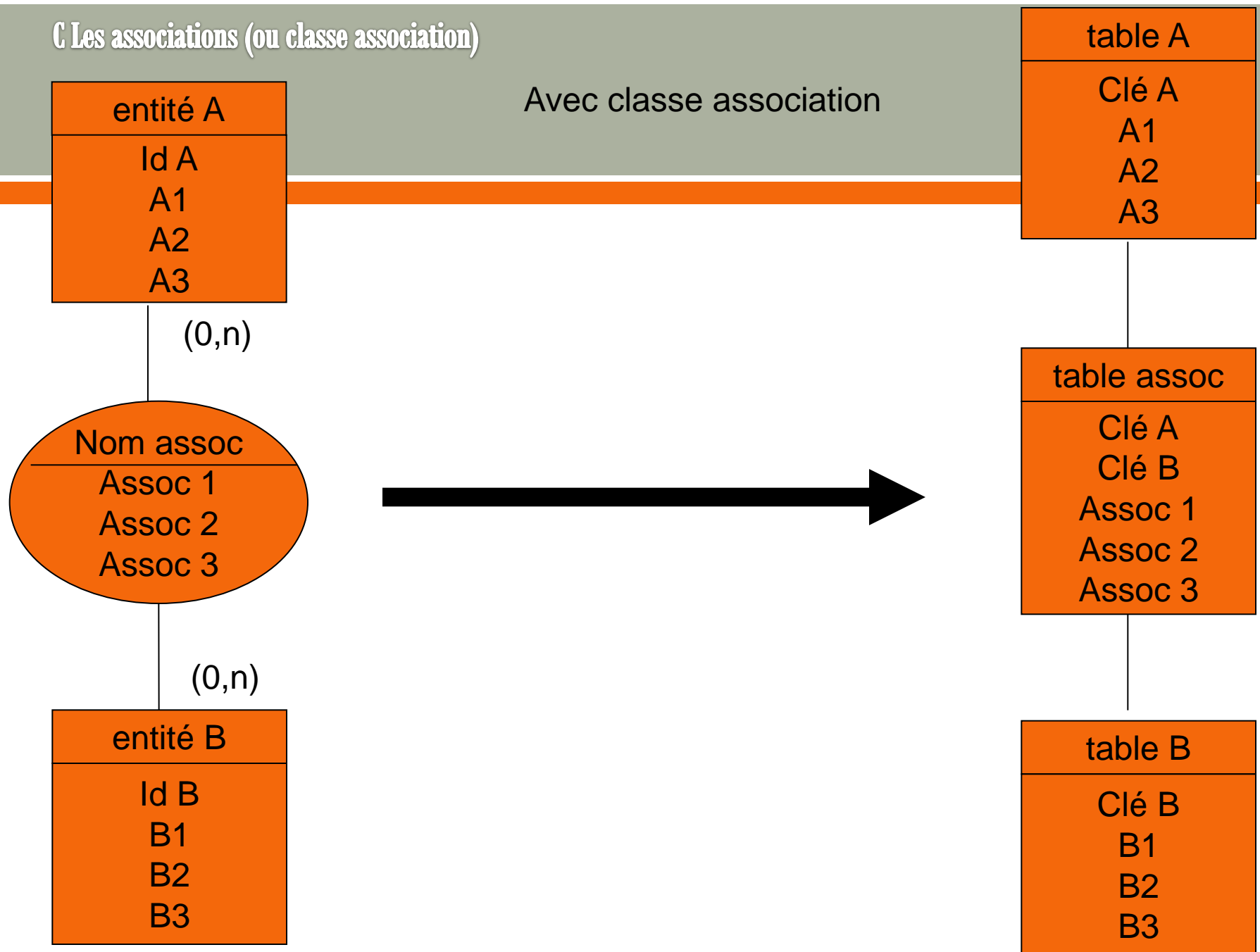


C Les associations (ou classe association)

Avec classe association

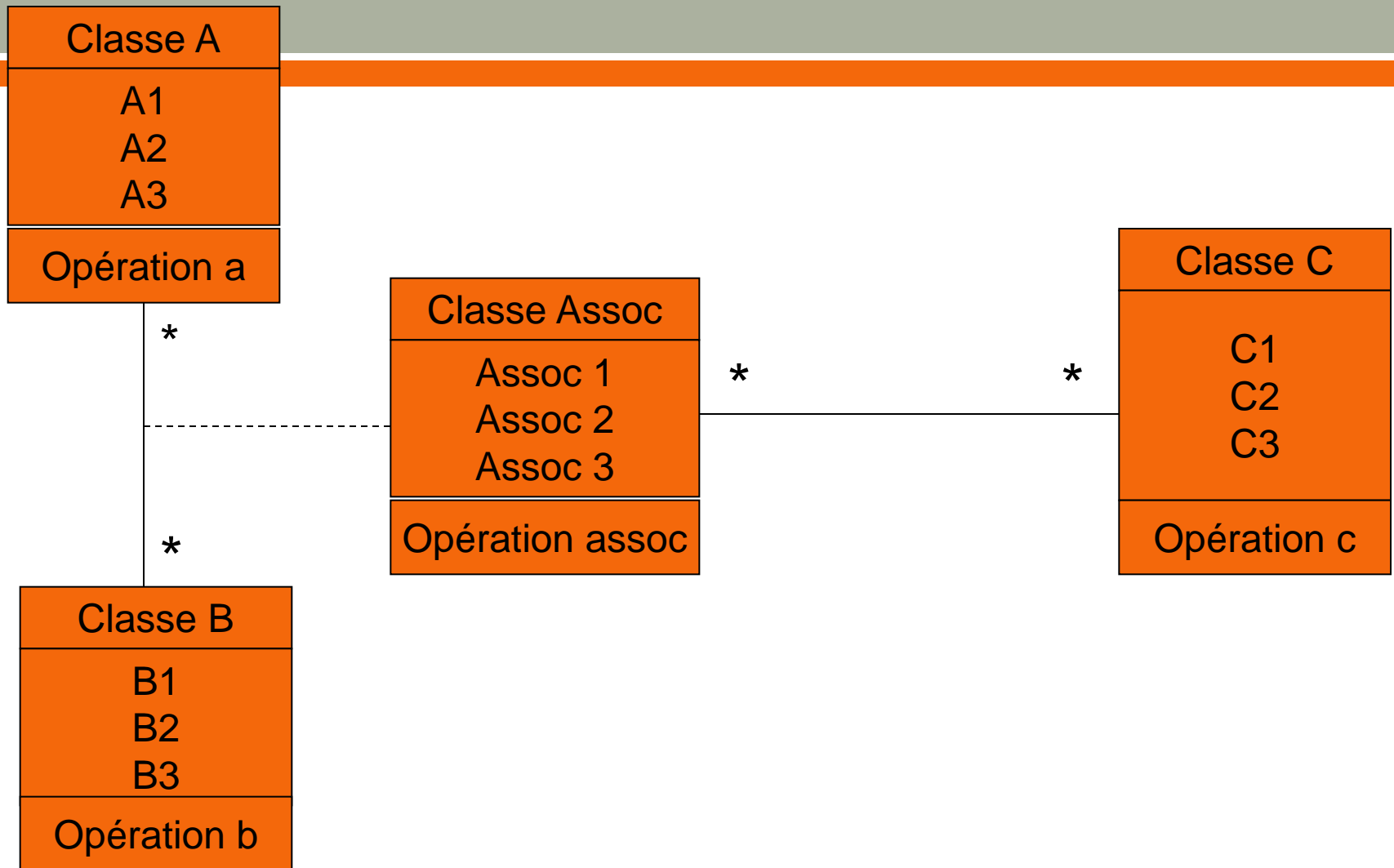


C Les associations (ou classe association)



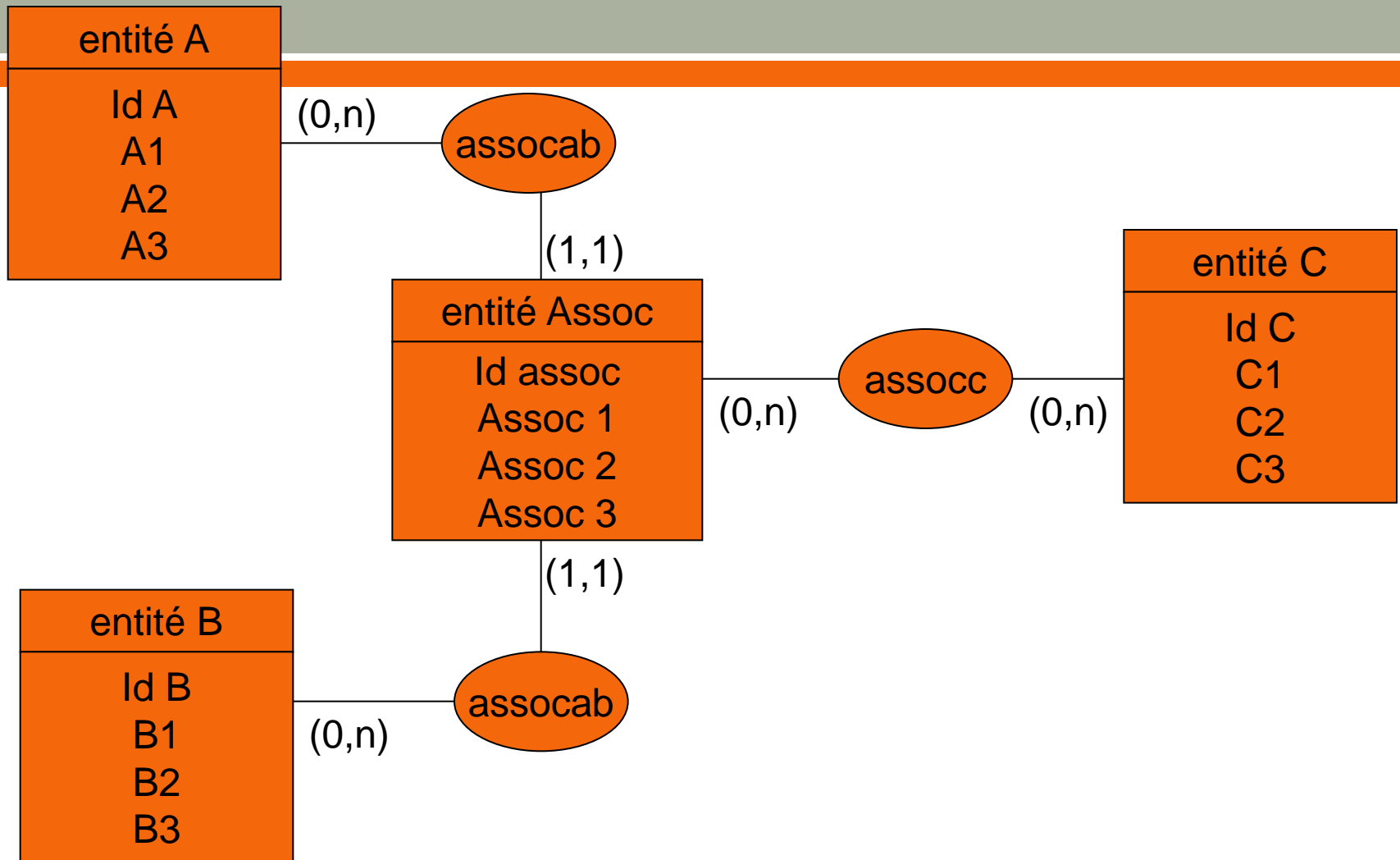
C Les associations (ou classe association)

Avec association sur classe association



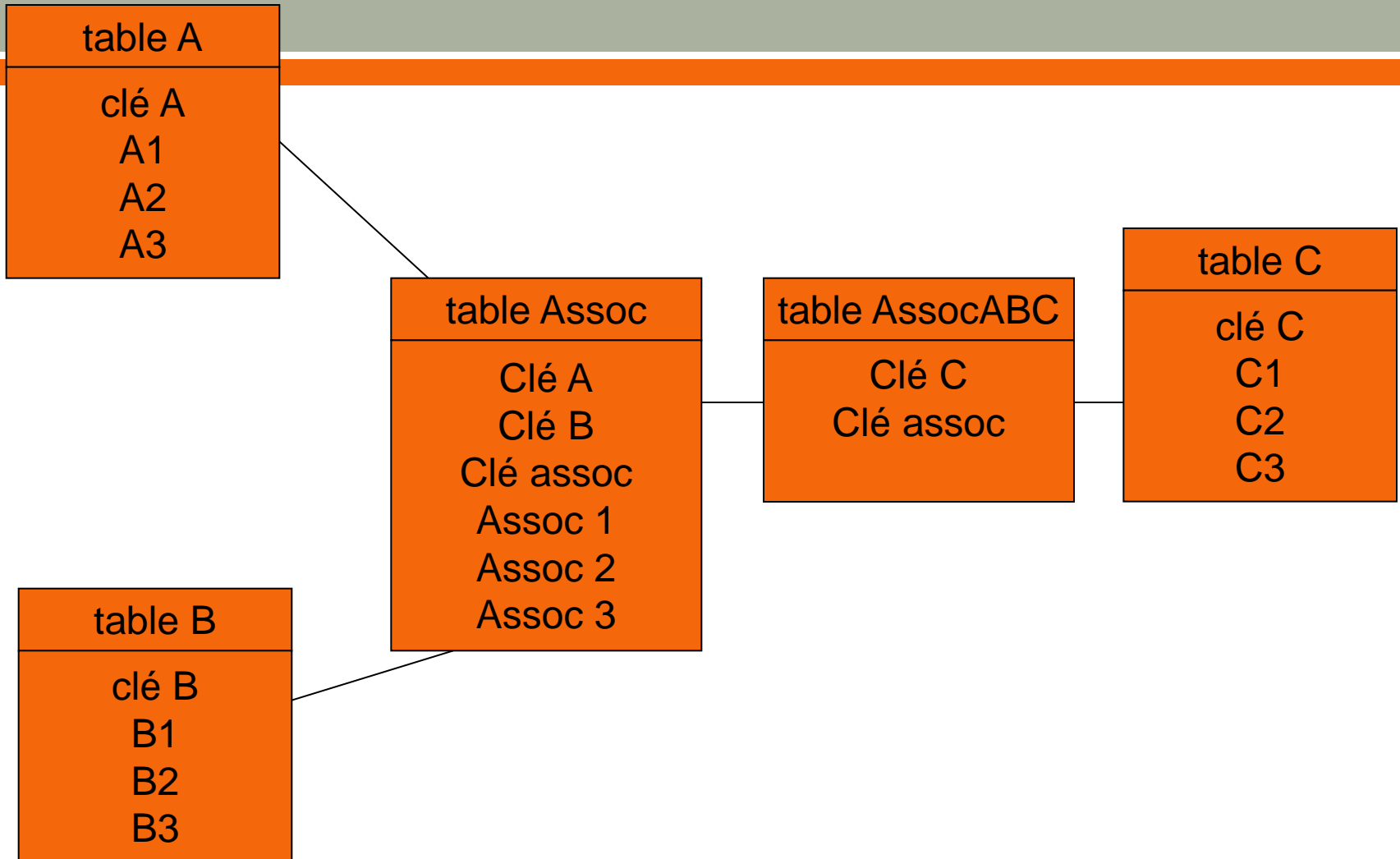
C Les associations (ou classe association)

Avec association sur classe association



C Les associations (ou classe association)

Avec association sur classe association

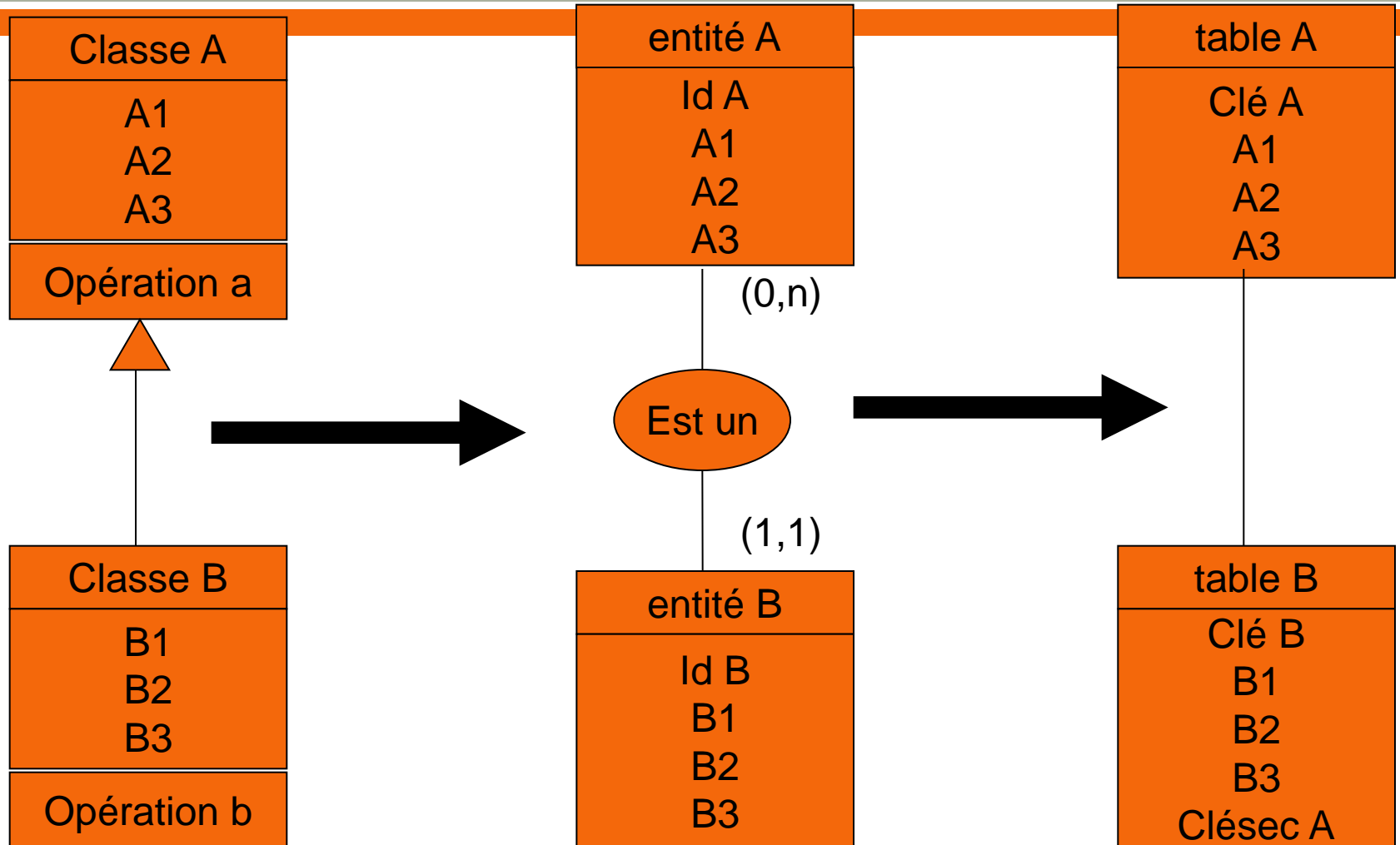


D Les généralisations

- Ensuite il y a les règles pour traduire l'héritage 3 possibilités :
 - tout garder
 - tout aplatir vers le haut
 - tout aplatir vers le bas.

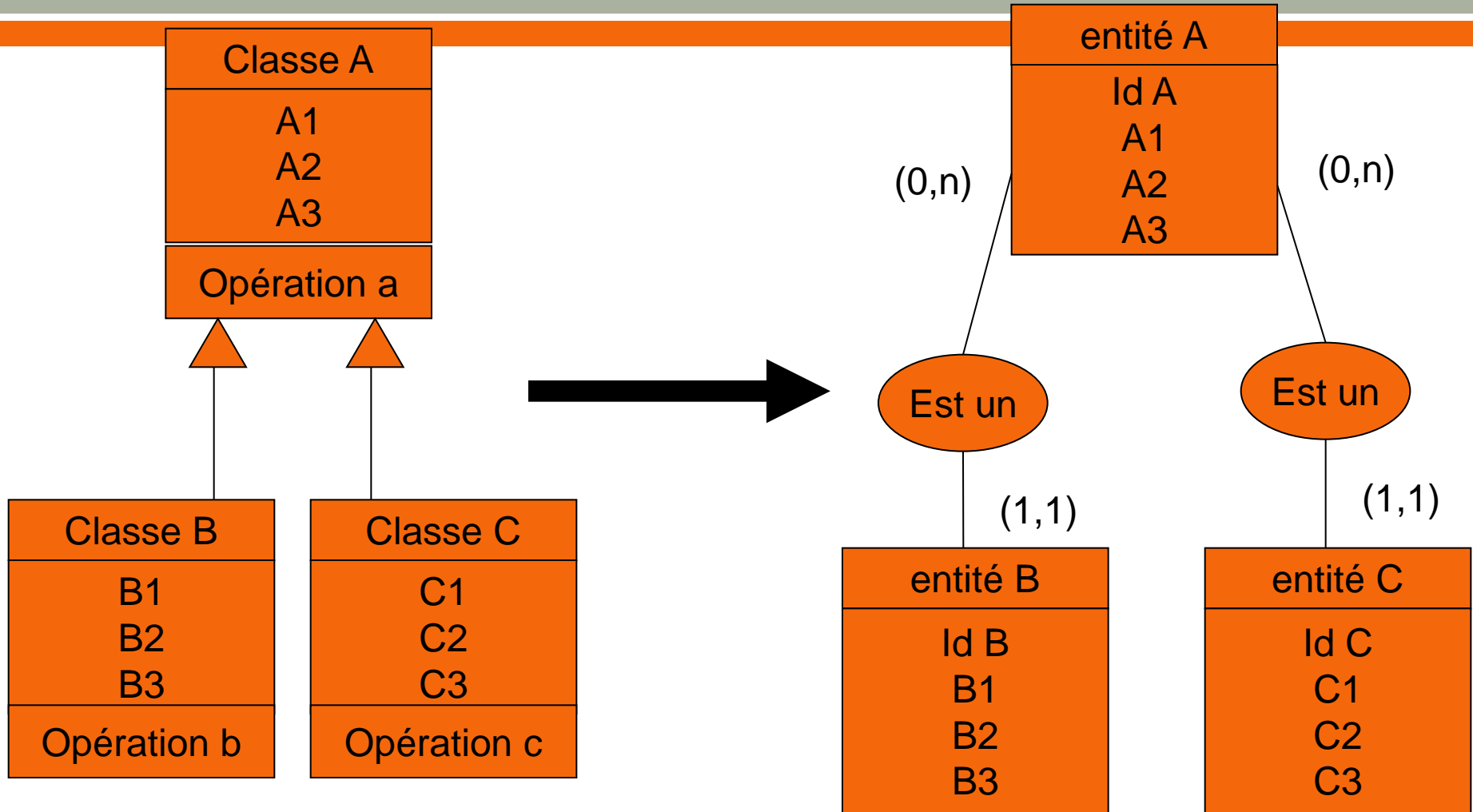
- Possibilité 1 : tout garder

Avec une seule sous-classe



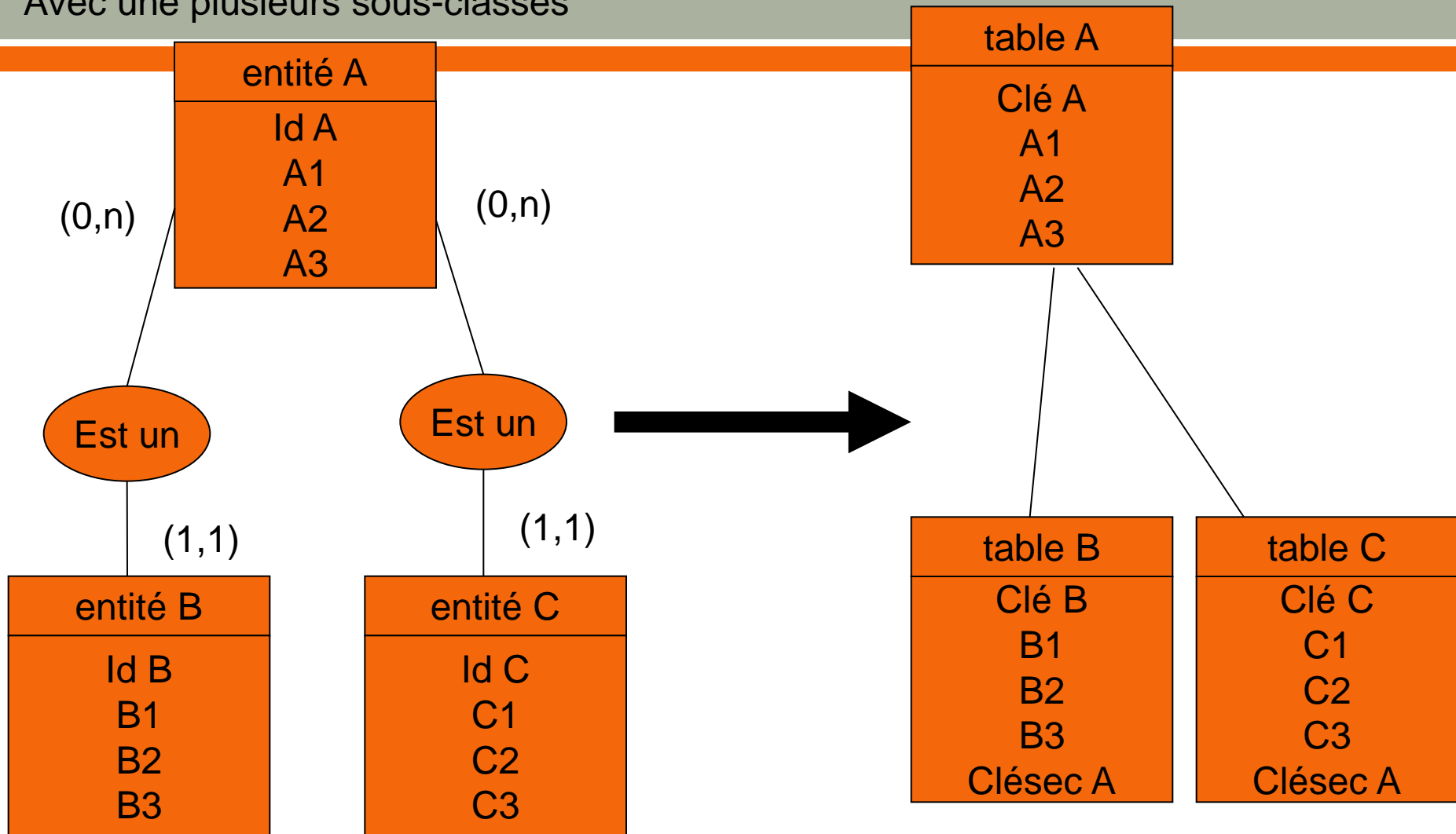
- Possibilité 1 : tout garder

Avec une plusieurs sous-classes



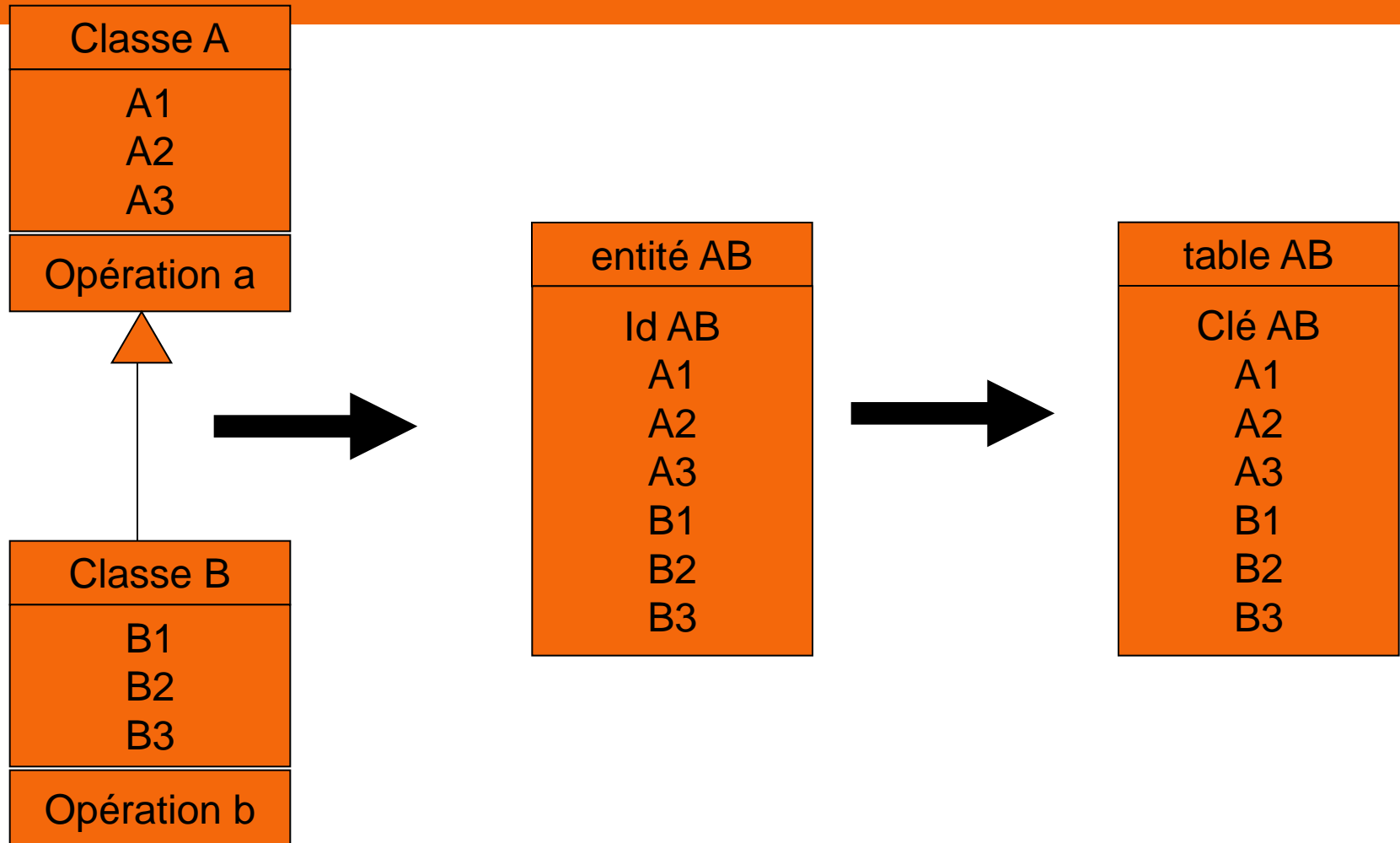
- Possibilité 1 : tout garder

Avec une plusieurs sous-classes



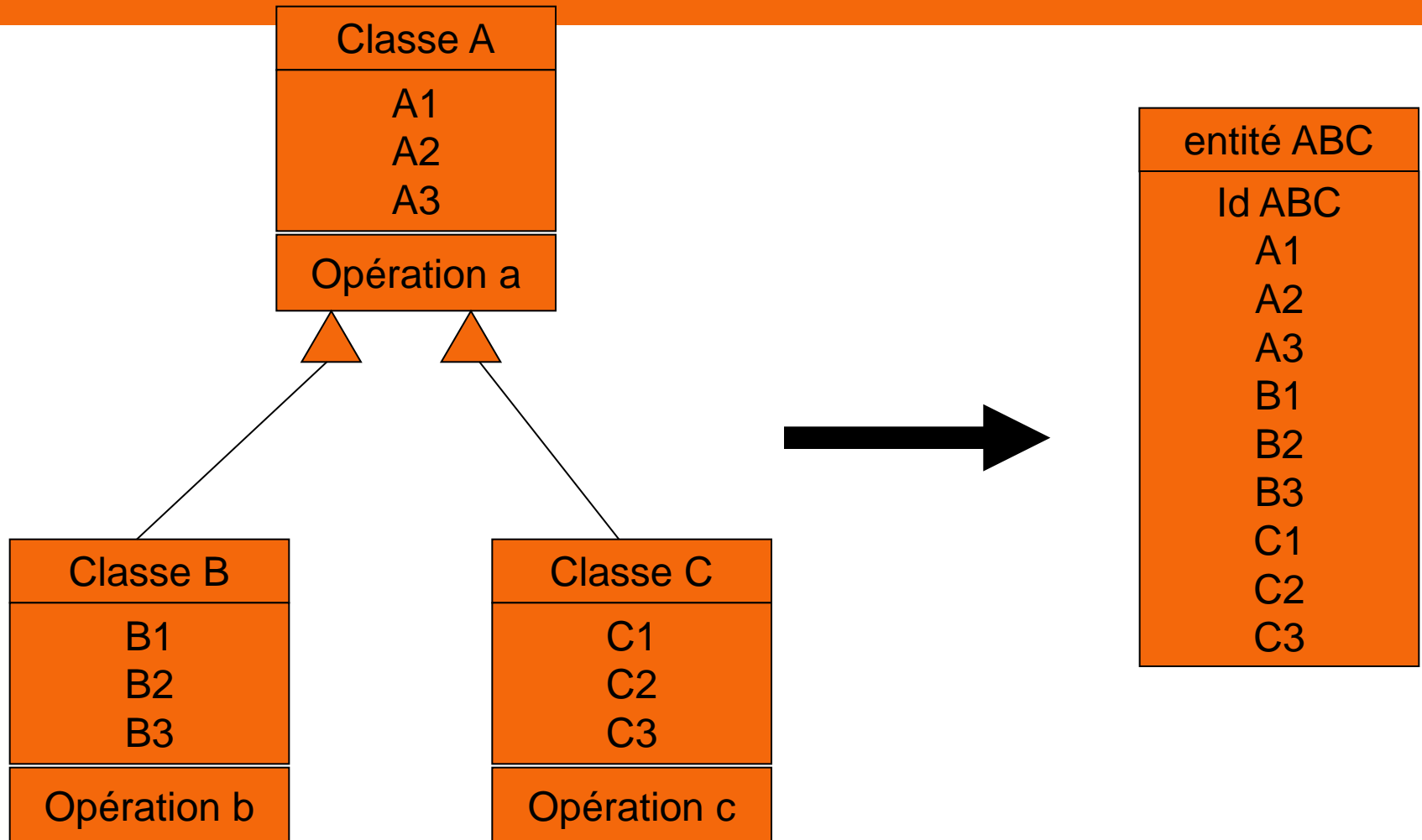
- Possibilité 2 : tout aplatir vers le haut

Avec une seule sous-classe



- Possibilité 2 : tout aplatir vers le haut

Avec plusieurs sous-classes



- Possibilité 2 : tout aplatir vers le haut

Avec plusieurs sous-classes

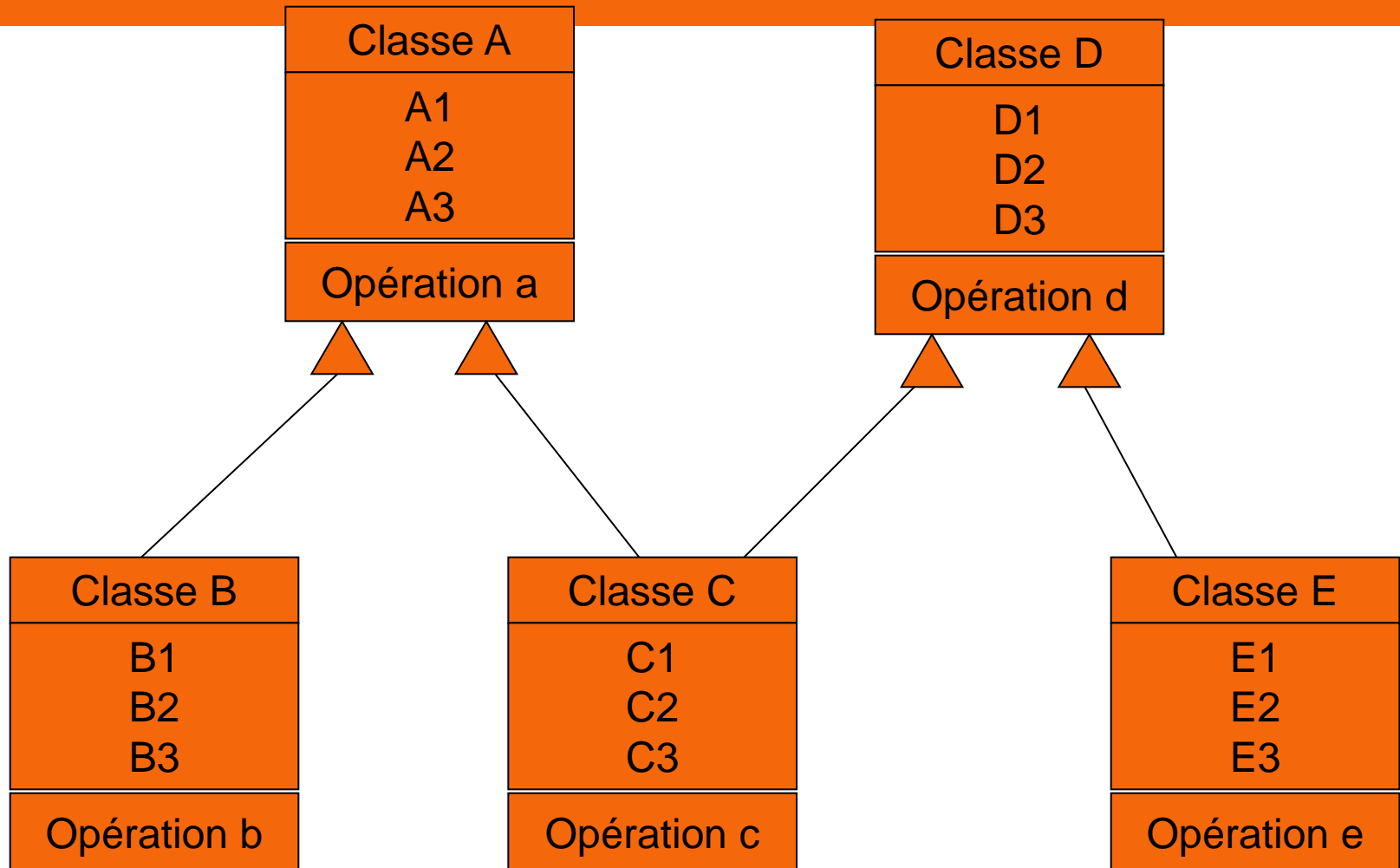
entité ABC
Id ABC
A1
A2
A3
B1
B2
B3
C1
C2
C3



table ABC
clé ABC
A1
A2
A3
B1
B2
B3
C1
C2
C3

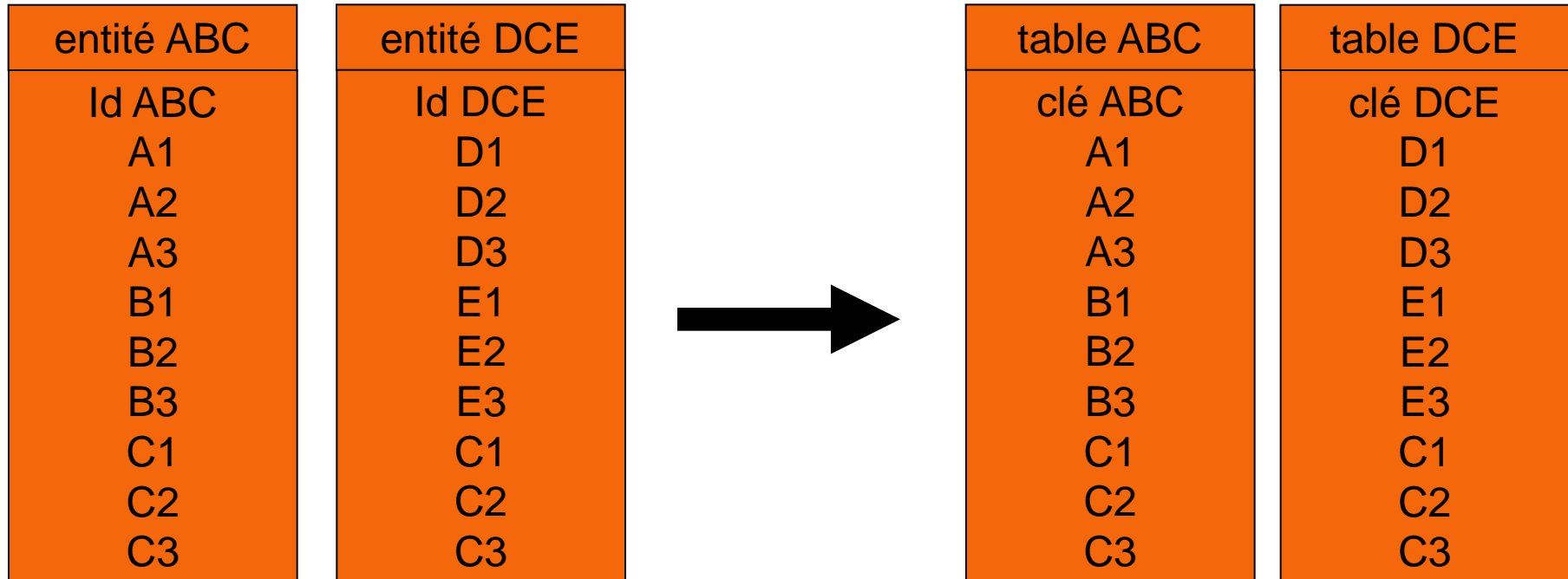
- Possibilité 2 : tout aplatir vers le haut

Avec héritage multiple



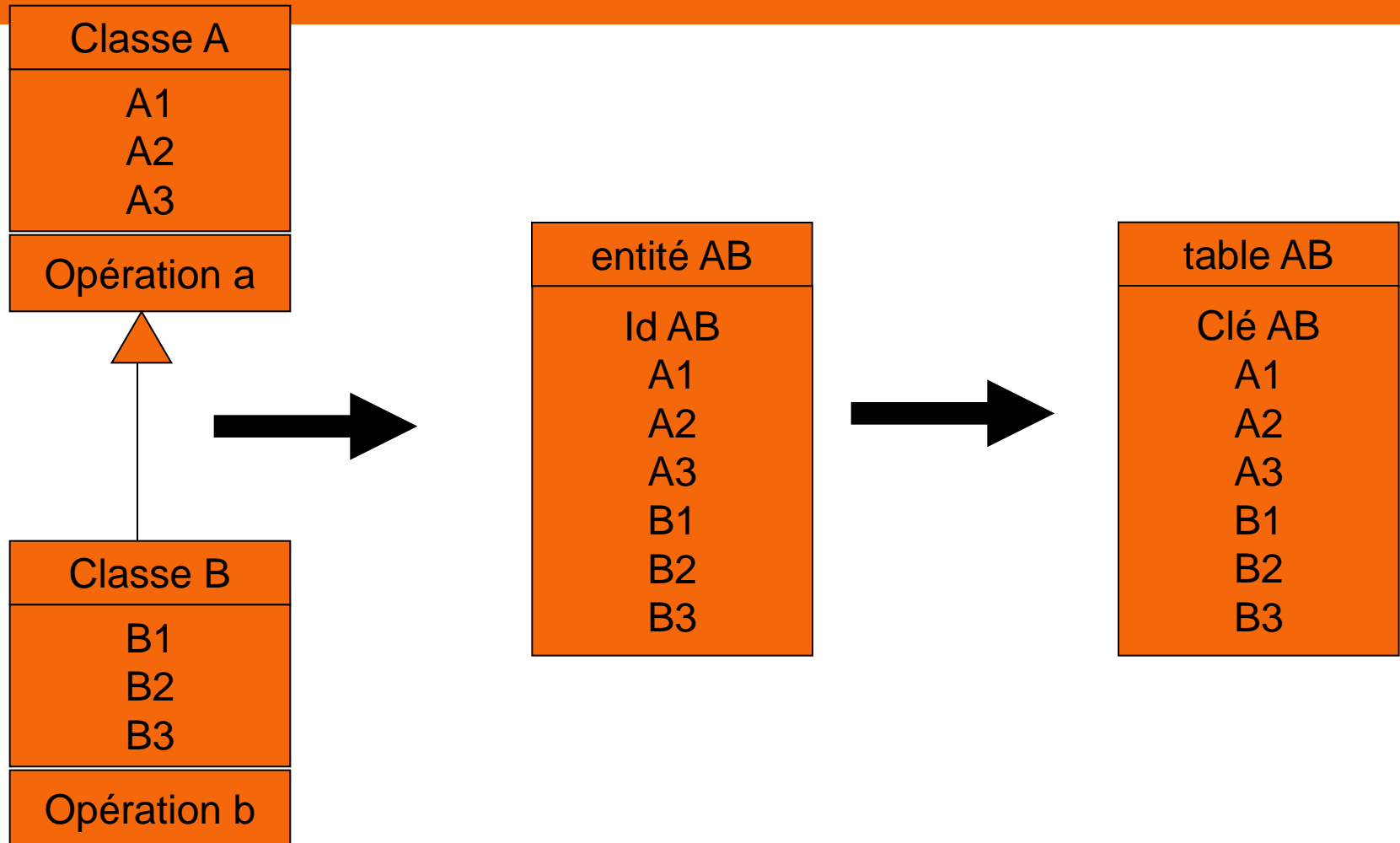
- Possibilité 2 : tout aplatir vers le haut

Avec héritage multiple



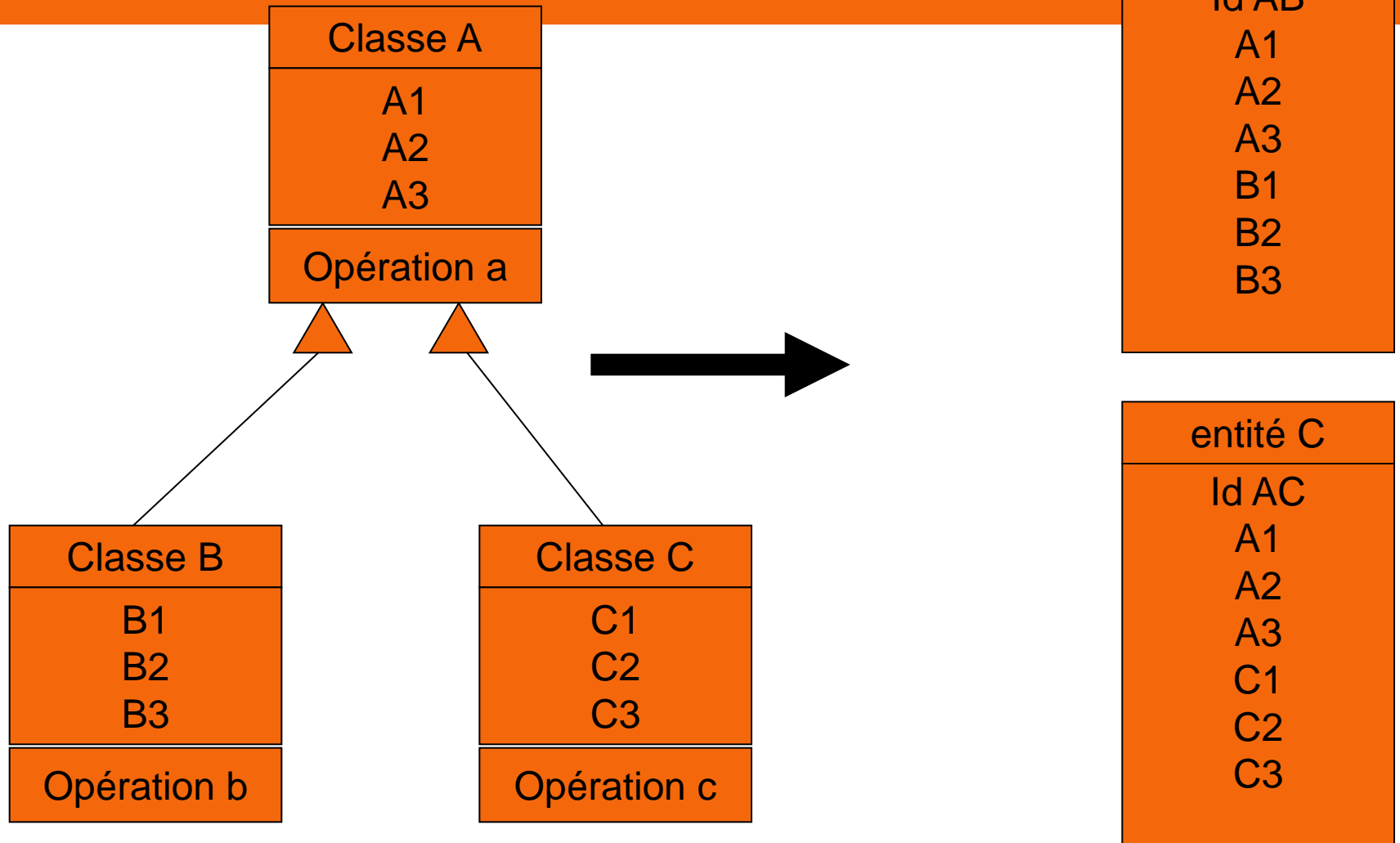
- Possibilité 3 : tout aplatir vers le bas.

Avec une seule sous-classe



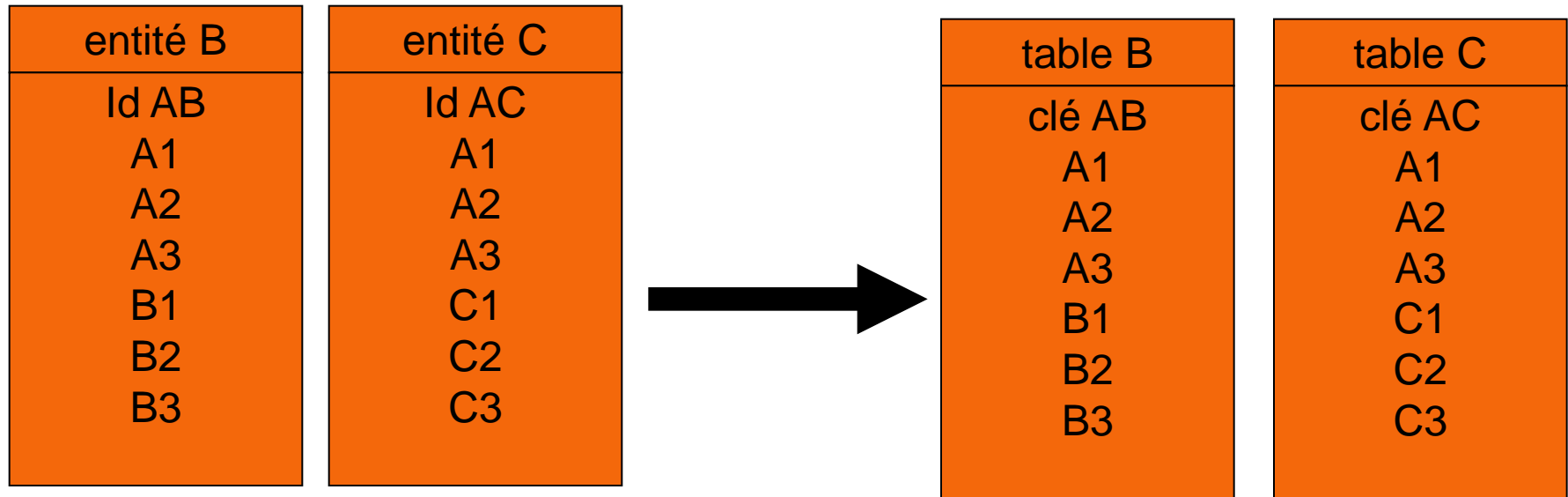
- Possibilité 3 : tout aplatir vers le bas

Avec plusieurs sous-classes



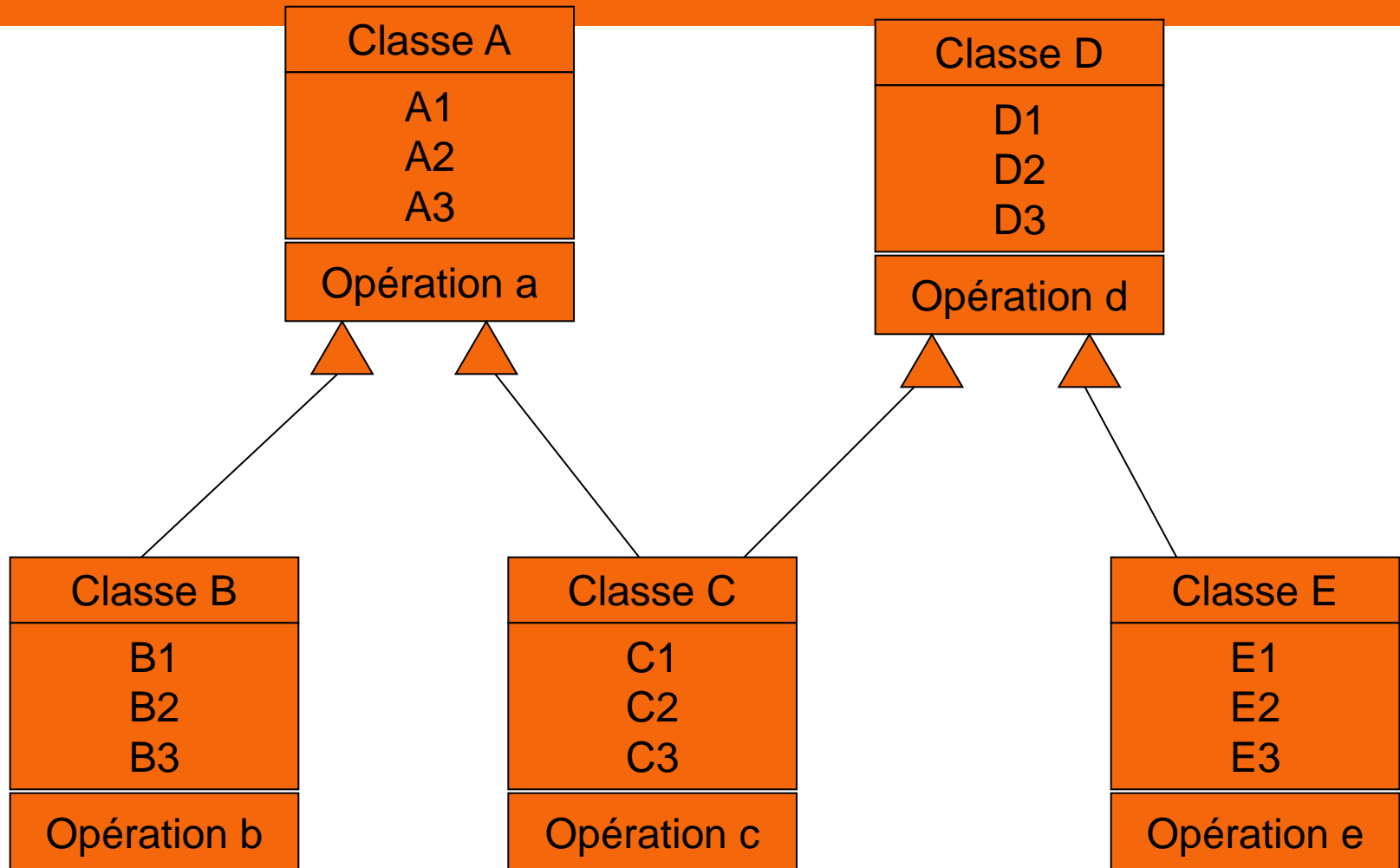
- Possibilité 3 : tout aplatir vers le bas

Avec plusieurs sous-classes



- Possibilité 3 : tout aplatir vers le bas

Avec un héritage multiple



- Possibilité 3 : tout aplatir vers le bas

Avec un héritage multiple

entité B
Id AB
A1
A2
A3
B1
B2
B3

entité E
Id DE
D1
D2
D3
E1
E2
E3

entité C
Id ADC
A1
A2
A3
C1
C2
C3
D1
D2
D3

- Possibilité 3 : tout aplatir vers le bas

Avec un héritage multiple

table B
clé AB
A1
A2
A3
B1
B2
B3

table E
clé DE
D1
D2
D3
E1
E2
E3

table C
clé ADC
A1
A2
A3
C1
C2
C3
D1
D2
D3

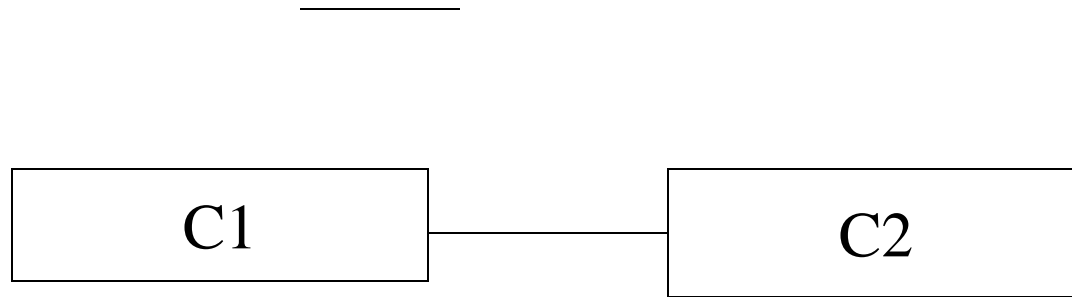
10.2 Diagramme de classes UML vers UNE BASE DE DONNEES ORIENTEE OBJET NATIVE



A Les agrégations

- On remplace les agrégations par des associations
- On rajoute les multiplicités

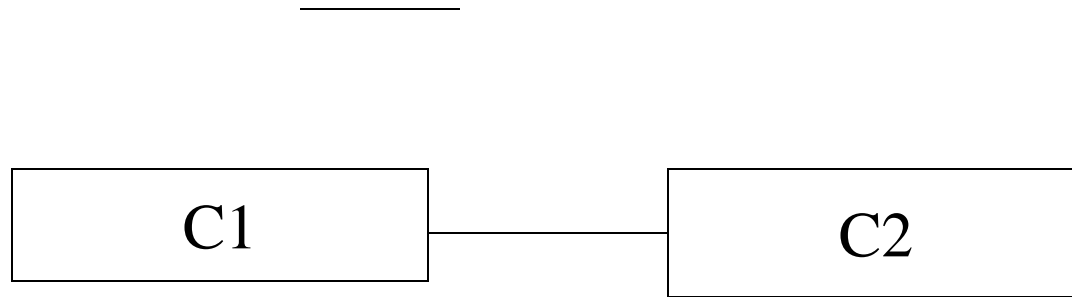
B Association 1-1



Class C1
tuple (...
 attribut
 ...
 classe1 : C2)

Class C2
tuple (...
 attribut
 ...)

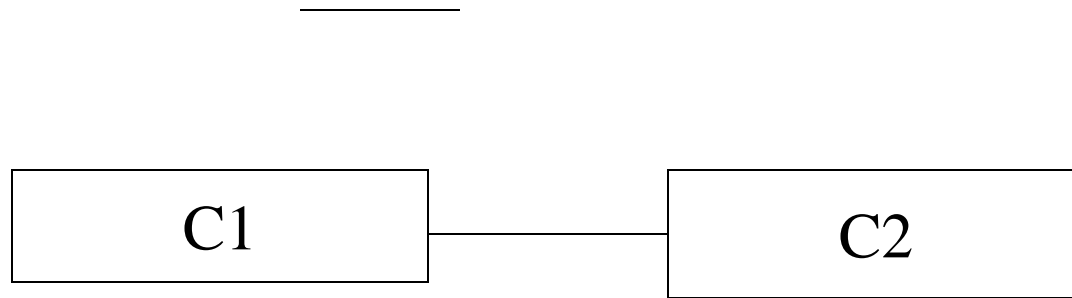
B Association 1-1



Class C1
tuple (...
 attribut
 ...)

Class C2
tuple (...
 attribut
 ...
 classe2 : C1)

B Association 1-1

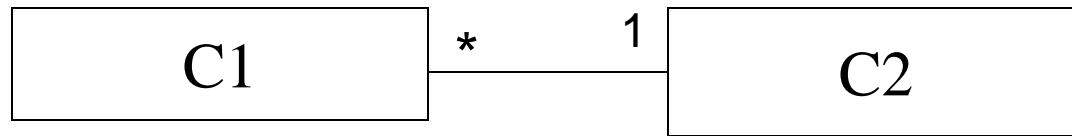


Class C1
tuple (...
 attribut
 ...
 classe1 : C2)

Class C2
tuple (...
 attribut
 ...
 classe2 : C1)

C Association 1-*

Possibilité 1

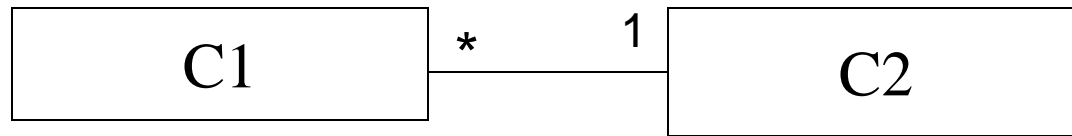


Class C1
tuple (...
 attribut
 ...)

Class C2
tuple (...
 attribut
 ...
 classe2 : set(C1))

C Association 1-*

Possibilité 2

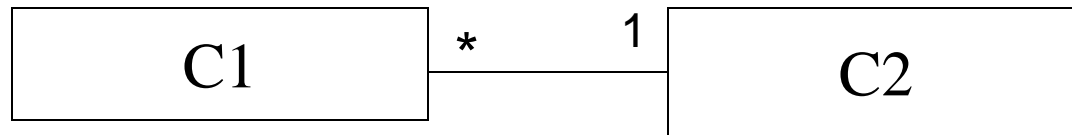


Class C1
tuple (...
 attribut
 ...
 classe1 : C2)

Class C2
tuple (...
 attribut
 ...)

C Association 1-*

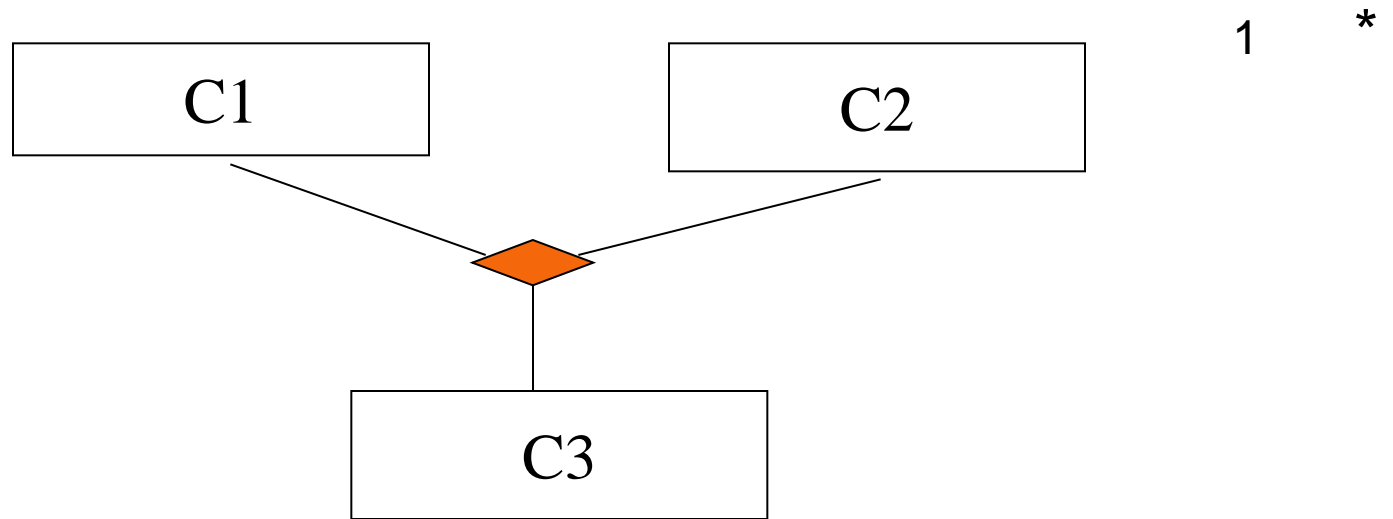
Possibilité 3



Class C1
tuple (...
 attribut
 ...
 classe1 : C2)

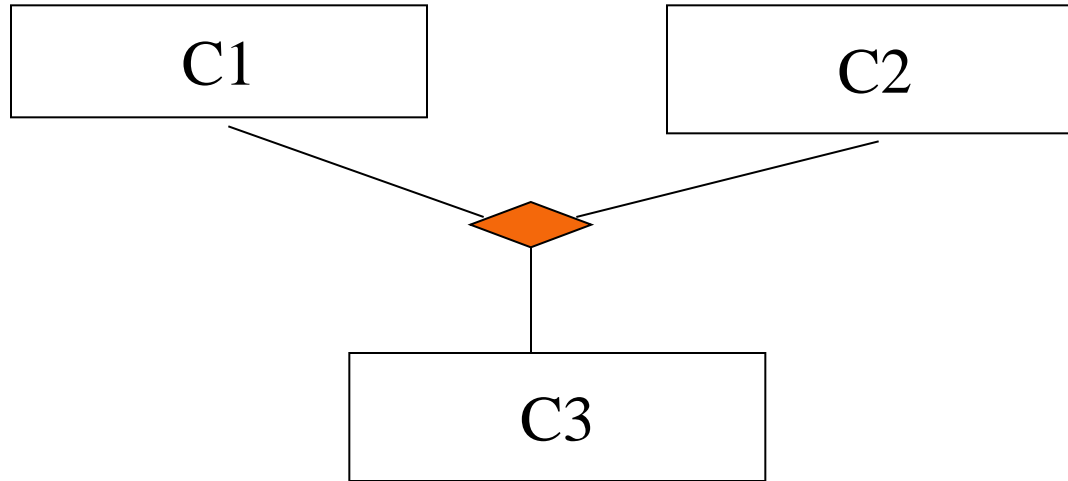
Class C2
tuple (...
 attribut
 ...
 classe2 : set(C1))

D Association ternaire



D Association ternaire

Possibilité 1



1

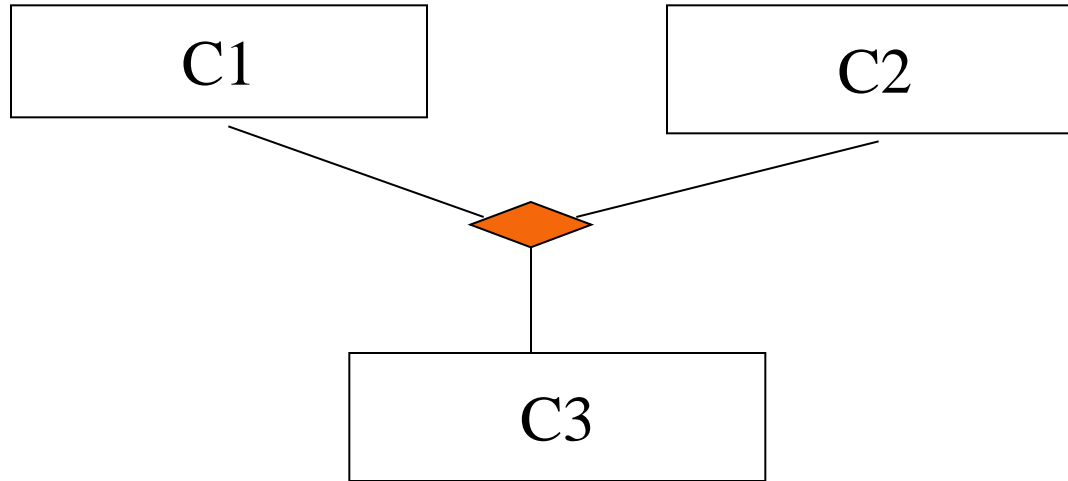
*

Class Assoc

```
tuple ( classe2 : C1
        classe1 : C2
        classe3 : C3))
```

D Association ternaire

Possibilité 2



1

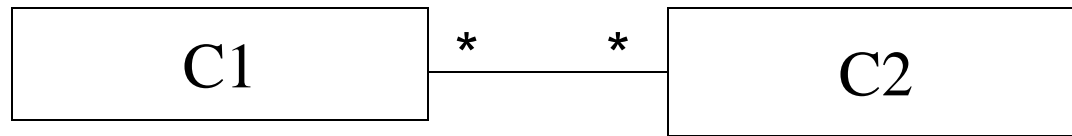
*

Class C1

```
tuple ( ...
    attribut
    ...
    set (tuple: (c2 :C2,
                  c3 :C3))
```

E Association *-*

Possibilité 1



Class C1

tuple (...

attribut

...

classe1 : set(C2))

Class C2

tuple (...

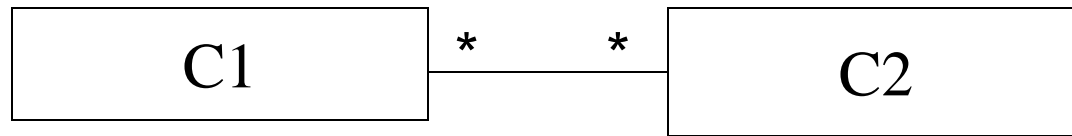
attribut

...)

1 classe privilégiée

E Association *-*

Possibilité 2



Class C1

tuple (...

attribut

...

classe1 : set(C2))

Class C2

tuple (...

attribut

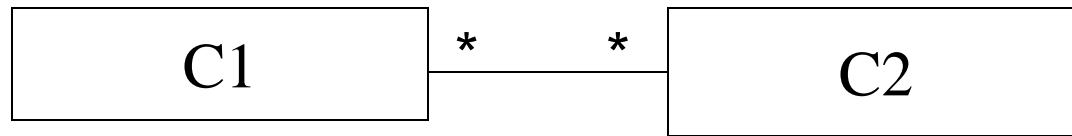
...

classe2 : set(C1))

Les 2 classes se référencent

E Association *-*

Possibilité 2



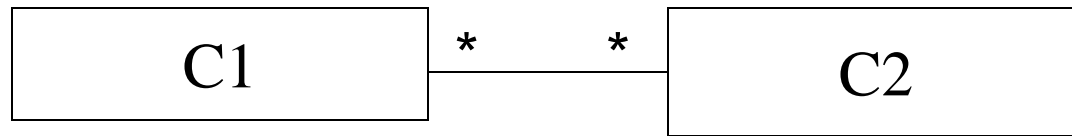
Class C1
tuple (...
 attribut
 ...)

Class C2
tuple (...
 attribut
 ...)

Une nouvelle classe Class Assoc
tuple (classe2 : C1
 classe1 : C2))

E Association *-*

Possibilité 2



Class C1

tuple (...

attribut

...

classe1 : set(C2))

Class C2

tuple (...

attribut

...

classe2 : set(C1))

Une nouvelle classe

Class Assoc

tuple (classe2 : C1

classe1 : C2))