BASES DE DONNEES ORIENTEES OBJETS

Trois chapitres

- Principes et modèles
  - 2 approches :
    - langage de programmation OO

      => nouveaux SGBD "purs orientés-objets"

      norme **ODMG**
    - extension des bd relationnelles

      => relationnel-objet

      SQL 3
  - ODMG, la partie modèle de données
- Langage de manipulation de données d'ODMG : OQL
- Relationnel-Objet : un exemple, Oracle

Principes des SGBD OO

Plan

- Evolution des applications et des SGBD
- Structure complexe
- Lien de composition
- Identité
- Hiérarchie de généralisation / spécialisation
- Population et persistance
- Méthodes et encapsulation
- Un exemple: FormaPerm en BD OO
- Conclusion

Rappel : Fonctions des SGBD

- BD = ensemble de données permanentes, intégrées, partagées, en accès simultané
- Intégrité de la base de données
- Sécurité de la base de données
  - protection contre les accès non autorisés
- Atomicité des transactions
- Fiabilité de la base de données
  - protection conte les pannes
- Langages de requêtes et de mises à jour déclaratifs
- Performances
  - techniques de stockage
  - optimisation des requêtes

Nouvelles applications

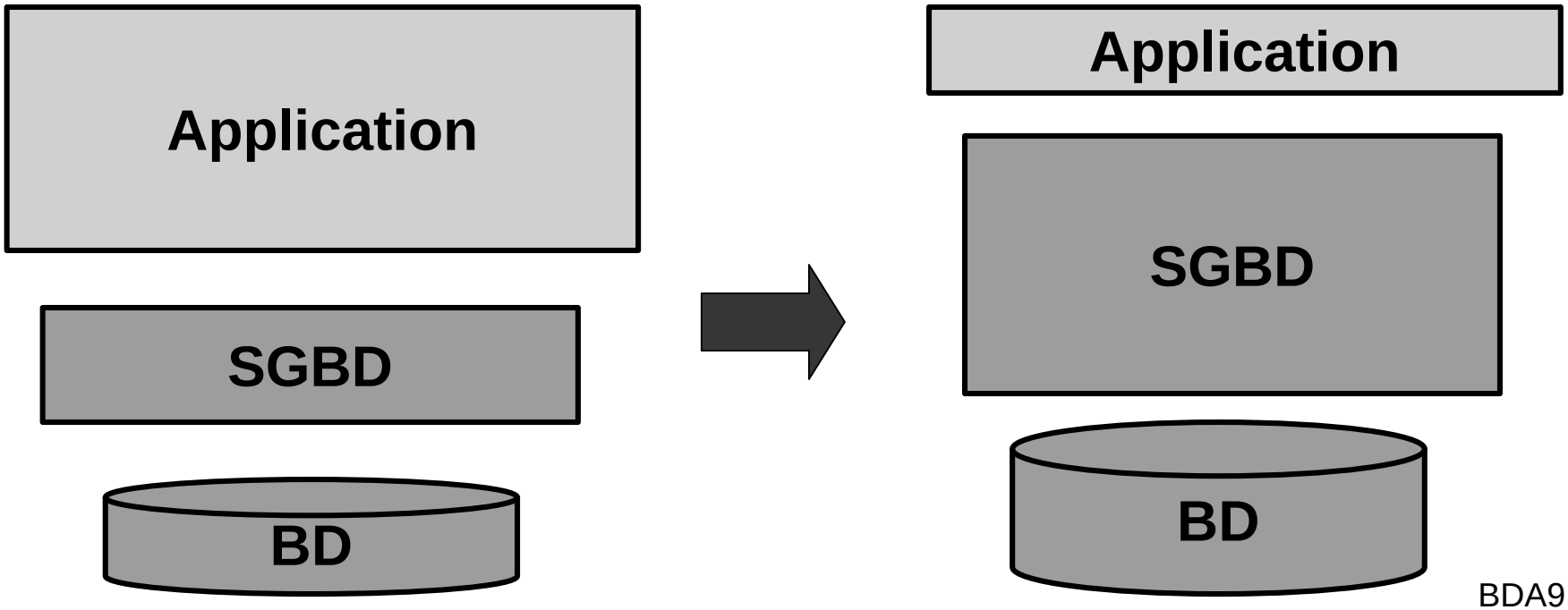
- conception assistée par ordinateur
- production assistée par ordinateur
- génie logiciel
- systèmes d'informations géographiques
- systèmes multi-média
- recherche et intégration de données de la toile
- …

- Nouveaux besoins
  - objets structurés, volumineux
  - nouveaux types de données
  - transactions longues
  - …
  - développement des SI non satisfaisant

Evolution des SGBD

- Applications plus complexes
- Coût du développement des applications

=> en faire faire plus au SGBD



Evolution des SGBD

1960’

- SGBD hiérarchique (IMS)
- SGBD réseau (CODASYL)
- ☺ schéma
  - langage navigationnel

1970’

- SGBD relationnel
- ☺ structure physique cachée aux utilisateurs
- ☺ modèle simple
- ☺ formalisation => normalisation
- ☺ langages déclaratifs

Evolution des SGBD (2)

1980’

Modèles sémantiques (EA)

☺ meilleure représentation du réel

☹ outils de conception uniquement

- 1986 : premiers SGBD OO

☺ meilleure représentation du réel au niveau logique

☺ réutilisation

☺1993 première norme ODMG pour SGBD OO

(Object Database Management Group)

☺1998 norme UML pour conception d'applications OO

☺1999 norme SQL3 pour SGBD relationnel-objet

ODMG

- Groupe de normalisation des SGBD OO
- Norme finale publiée en 2001
- A regroupé de nombreux vendeurs de SGBO OO
  - Poet
  - Ardent
  - Objectivity
  - Versant
  - GemStone
  - …

et des constructeurs, des utilisateurs, des chercheurs …

- www.odmg.org

Le relationnel : avantages

- approche formellement définie (=> normalisation, algèbre)
- modèle simple
- langage standard (SQL 2), déclaratif
- niveau logique (essentiellement)
- technologie la plus répandue
- efficace pour les applications de gestion classique

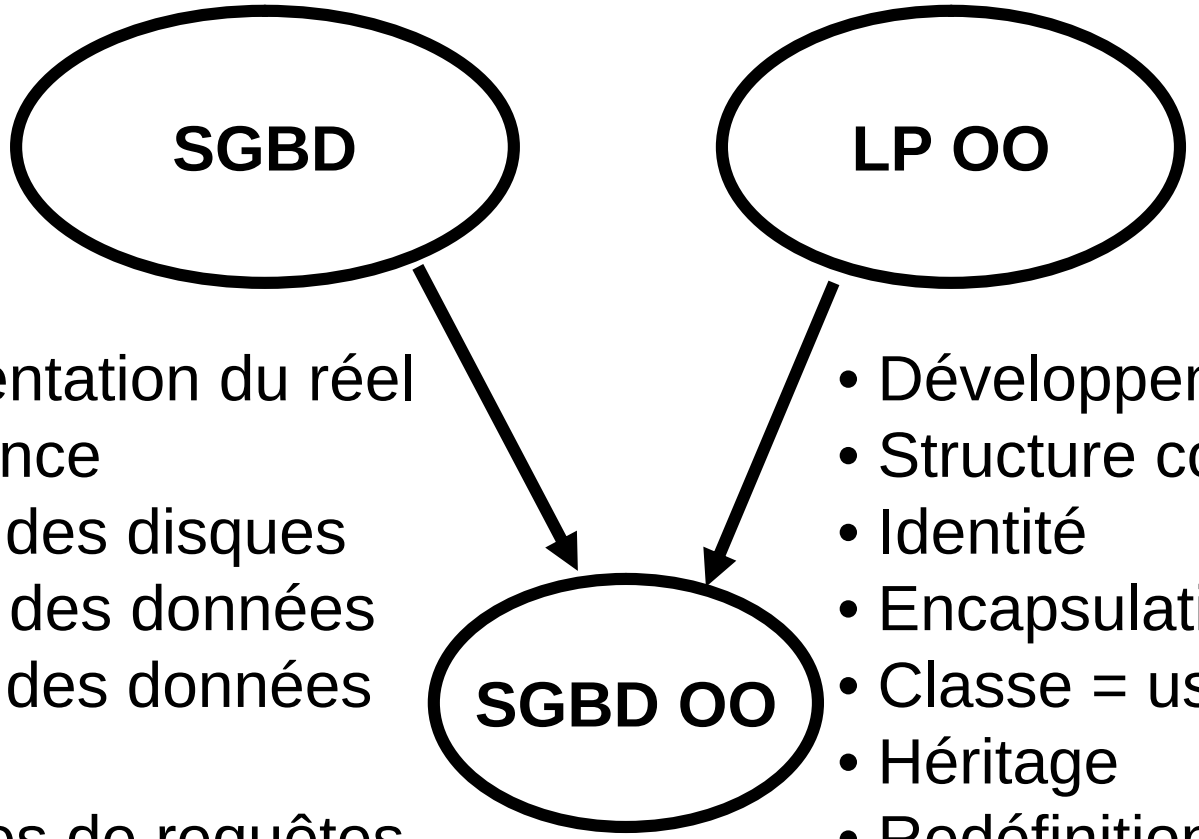
Le relationnel : faiblesses

- structure de données trop simple
  - pas d'attribut complexe, ni multivalué
    ==> entités réelles éclatées, jointures
  - un seul type de lien (clé externe)
- pas de niveau conceptuel
- peu compatible avec les langages de programmation
  - ensemble <--> élément
  - déclaratif <--> impératif
  - types de données
- données alphanumériques uniquement
  - images, sons, vidéo, espace …
- performances problématiques en cas de jointures
- développement et maintenance des SI insatisfaisant
- mécanisme de transactions inadapté aux nouvelles applications

Approche OO

- Ensemble de méthodologies et d'outils pour concevoir et réaliser des logiciels structurés et réutilisables, par composition d'éléments indépendants. [Khoshafian + Boral]
- Objectif : productivité des programmeurs
  - Moyen : réutilisation
- Concepts essentiels
  - objet encapsulé
    - interface visible : opérations (méthodes)
    - implémentation cachée : structure et code
  - héritage
- Langages de programmation OO
  - Eiffel, Smalltalk, C++, Java …

SGBD OO = LPOO + BD



- Représentation du réel
- Persistence
- Gestion des disques
- Partage des données
- Fiabilité des données
- Sécurité
- Langages de requêtes
- Indépendance logique / physique

- Développement
- Structure complexe
- Identité
- Encapsulation
- Classe = usine
- Héritage
- Redéfinition
- Bibliothèques de classes

Intérêt d'un SGBD OO / LP OO

C'est un SGBD (mieux qu'un LP):

- persistence des données
- indépendance modèles logique et physique
- LMD déclaratif
  - optimisation par le SGBD
- intégrité des données
- confidentialité, fiabilité, concurrence, gestion de transactions, …

Intérêt d’un SGBD OO / SGBDR

C’est mieux qu’un SGBD relationnel :

- permet la manipulation d’objets à structure complexe
- interface compatible avec les LP-OO
- nouveaux types de données (image, son…)
- versions, historiques, nouvelles transactions
- performances

Différences entre SGBDO

- toutes les fonctions d'un SGBD ?
- modèles de données différents
- langage sous-jacent différent (C++, Smalltalk, Lisp …)
- interprété ou compilé
- couplage fort ou faible avec le(s) langage de programmation
- performances
- bibliothèque de classes ± complète
- autres fonctions (versions, évolution du schéma, temps, extensibilité …)

Bases de données orientées objets

Modélisation

Diversité des modèles

- Norme ODMG
  mais de nombreux SGBDO ne la suivent pas.

- Ce cours définit :
  - les principes communs aux SGBD OO
  - les alternatives importantes

- Ce cours emploie une syntaxe tirée de celle d'ODMG

- Le relationnel-objet (SQL 3) sera présenté dans le chapitre 3.

Concepts principaux

Monde réel	**BD OO**
objet	objet, classe d'objets
propriété	attribut méthode
lien	lien de composition binaire sans attribut orienté
représentation multiple	hiérarchie de généralisation/ spécialisation, héritage

OBJETS A STRUCTURE COMPLEXE

- Objectif : représentation directe des objets du monde réel
- Monde réel : Personne

  nom

  prénoms

  adresse (rue, n°, ville, codeNPA)

  enfants (prénoms, sexe, dateNais)

  …

- En relationnel : **4** relations, **N** tuples

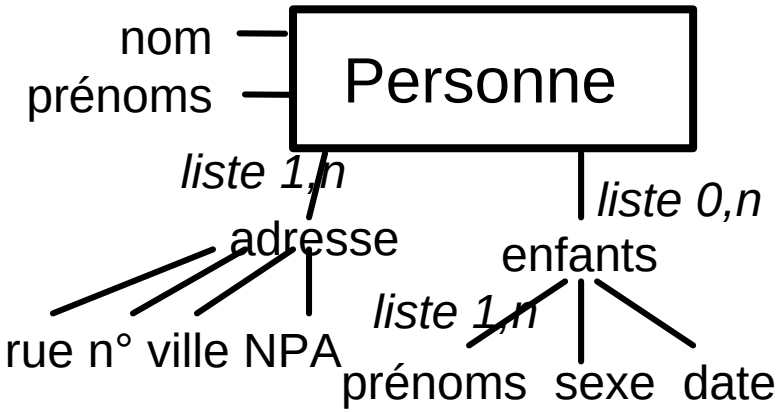
  Personne (n°, nom, adresse_rue, adresse_n°, adresse_ville, adresse_codeNPA)

  Personne_prénom (n°P, n°prénom, prénom)

  Personne_enfant (n°P, n°enfant, sexe, dateNais)

  Person_enfant_prénom (n°P, n°enfant, n°prénom, prénom)

Structure complexe



En OO : **un** seul objet

CLASS **Personne**

{ ATTRIBUTE **nom** : STRING ,

ATTRIBUTE **prénoms** : LIST STRING ,

ATTRIBUTE **adresse** : STRUCT **adr**

{ **rue** : STRING ,

n° : STRING ,

ville : STRING ,

codeNPA : INT }

ATTRIBUTE **enfants** : LIST STRUCT **enfant**

{ **prénoms** : LIST STRING ,

sexe : ENUM {'M', 'F'} ,

date : DATE }

}

Structure complexe (suite)

- Constructeurs de structure complexe :
  - attribut complexe : STRUCT
  - attribut multivalué => constructeur de collection
    - ensemble : SET
    - liste : LIST
    - multi-ensemble : BAG
    - tableau à une dimension : ARRAY
- Impact sur le SGBD :
  - LMD : comment accéder aux valeurs ?
    - notation pointée
    - variables sur les attributs multivalués
  - stockage d'objets complexes, gros, de taille variable

Types définis par l'application

- Les constructeurs de structure complexe servent à:
  - définir des **classes d'objets** à structure complexe
  - définir des **types de données** adaptés à l'application
    - type T-Adresse
    - types Point, Ligne, Polygone
    - types Image, Son …
- Comme les classes d'objets, les types de données définis par l'application ont :
  - une structure complexe
  - des opérations (méthodes)

Types de données - Exemple

TYPEDEF **T-Adresse** STRUCT
{ ATTRIBUTE **rue** : STRING ,
ATTRIBUTE **n°** : STRING ,
ATTRIBUTE **ville** : STRING ,
ATTRIBUTE **codeNPA** : INT }

CLASS **Personne**
{ ATTRIBUTE **nom** : STRING ,
ATTRIBUTE **prénom** : LIST STRING ,
ATTRIBUTE **adresse** : **T-Adresse** ,
ATTRIBUTE **enfants** : LIST STRUCT **enfant**
{ **prénoms** : LIST STRING ,
sexe : ENUM {'M', 'F'} ,
date : DATE } }

OBJET AVEC IDENTITE

- Objectif : Identifier les objets indépendamment de leur valeur et de leur adresse (MC ou disque)

  =>? ? ? ? ? ? ? ? ? ? ? ? ? aux changements de valeur

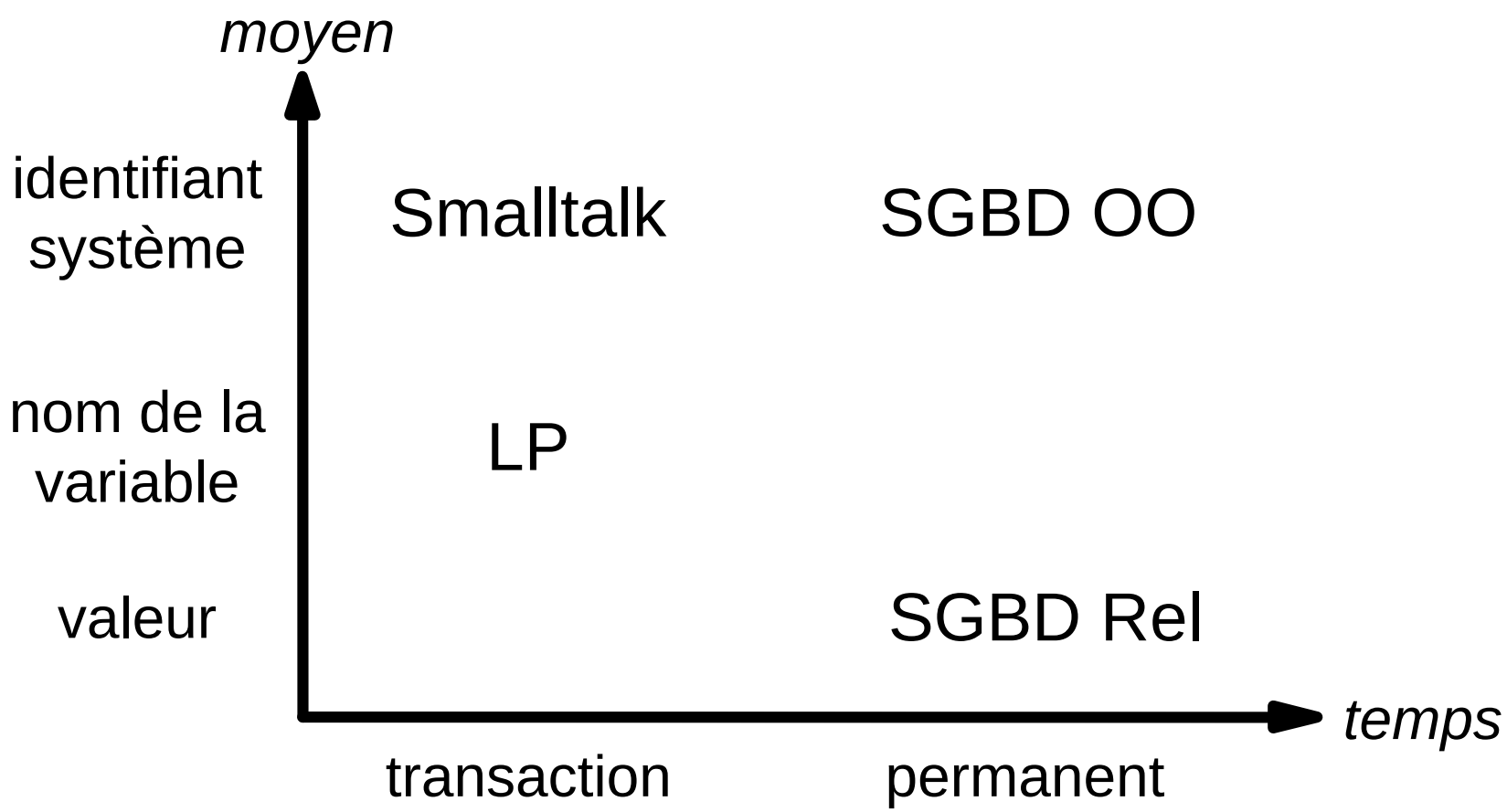
  => insensibilité aux déplacements internes

- Chaque objet possède une identité propre qui ne peut être changée durant toute sa vie
- L'identification des objets est gérée par le système (allocation).
- Intérêt de l'identité d'objet
  - Représentation directe du monde réel
  - Permet de représenter des doubles
  - Moyen efficace pour référencer un objet

Identités , clés , noms …

- SGBD relationnels :
  clé = un ensemble minimum d'attributs
  - Danger lors des :
    - mises à jour de la clé
    - changements d'attribut clé
  - Identité dépendante de la valeur
- Langages de programmation :
  noms des variables
  - Attention :
    - pas de test d'identité : X == Y ?
    - temporaire
  - Identité dépendante des accès

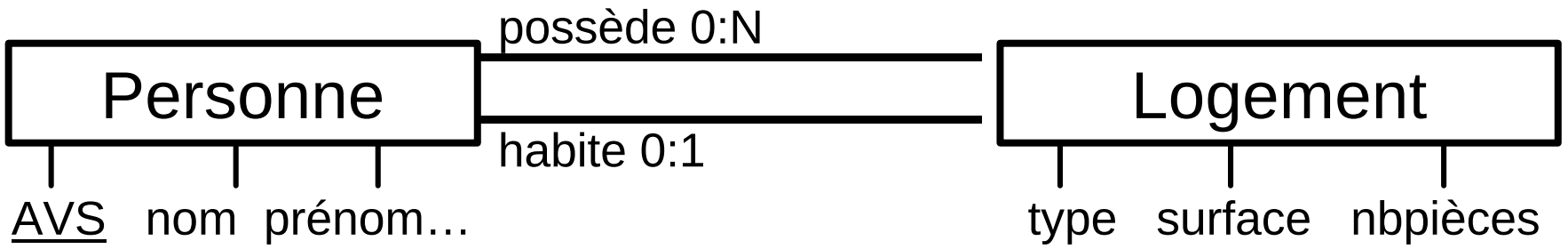
Approches de l’identité d’objet



Identité en orienté objet

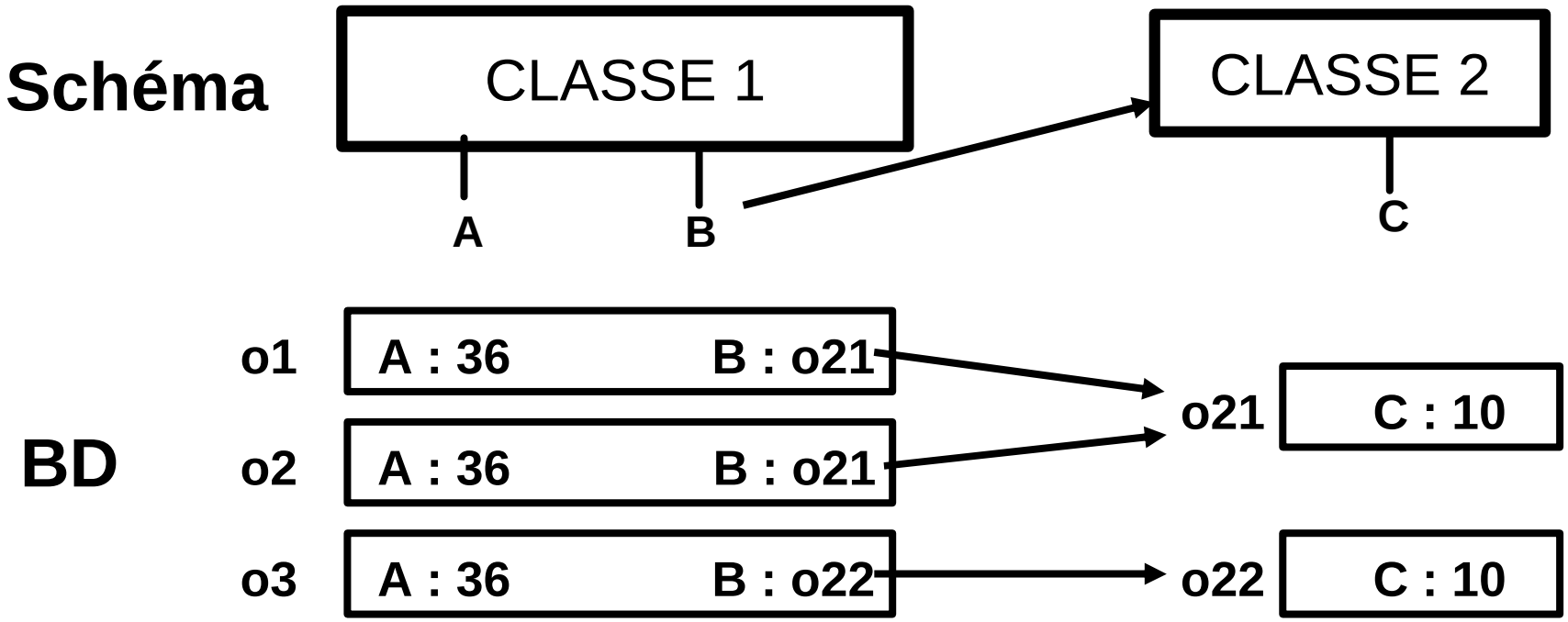
- oid (object identifier) géré par le SGBD OO
  - unique
  - permanent
  - immuable
- objet : (oid, valeur)
- Trois test d'égalité !
  - test d'identité ==
    même oid
  - test d'égalité en surface =
    même valeur
  - test d'égalité en profondeur = *
    feuilles composantes de même valeur

Tests d’identité / d’égalité



- Qui possède le logement qu’il habite ?
- Paul et Pierre habitent-ils des logements identiques ?
- Paul et Pierre habitent-ils le même logement ?

Tests d'identité / d'égalité



identité : o1.B == o2.B

égalité surface : o1 = o2

égalité profonde : o1 =* o3

o21 = o22

o1 =/= o2

o1 ? o3

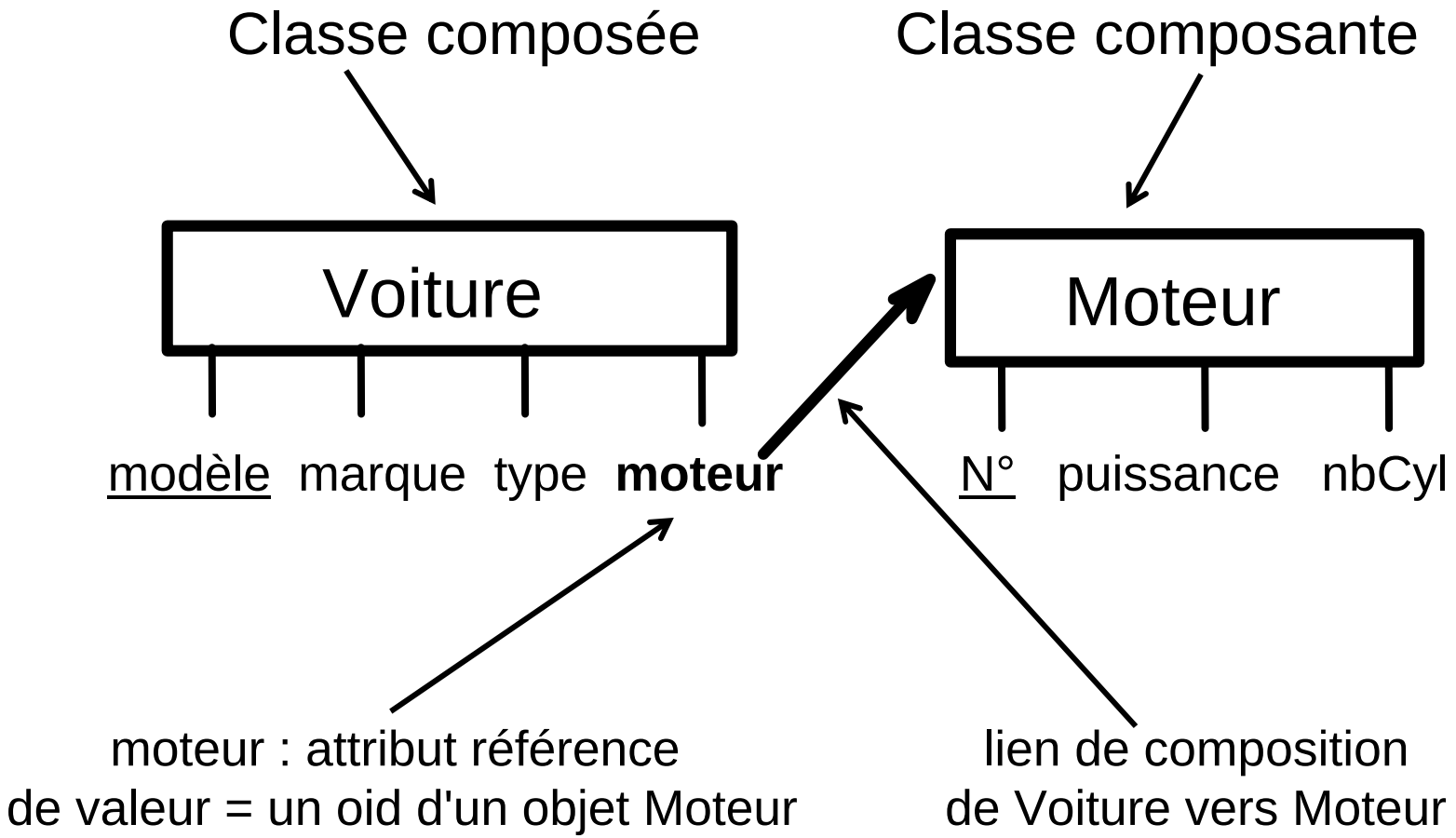
o21 =/= o22

Identité : impact sur le SGBD

- Implémentation :
  - adresse disque ou MC
  - un numéro logique
    - Exemple : n° de classe + n° de séquence
- LMD
  - différents tests
  - opérations ensemblistes selon :
    - les valeurs ?
    - les oids ?

LIEN DE COMPOSITION

- Objectif : représenter les liens de composition qui existent entre objets du monde réel



Lien de composition

CLASS Voiture

{ modèle : STRING ,
marque : STRING ,
type : STRING ,
moteur : Moteur }

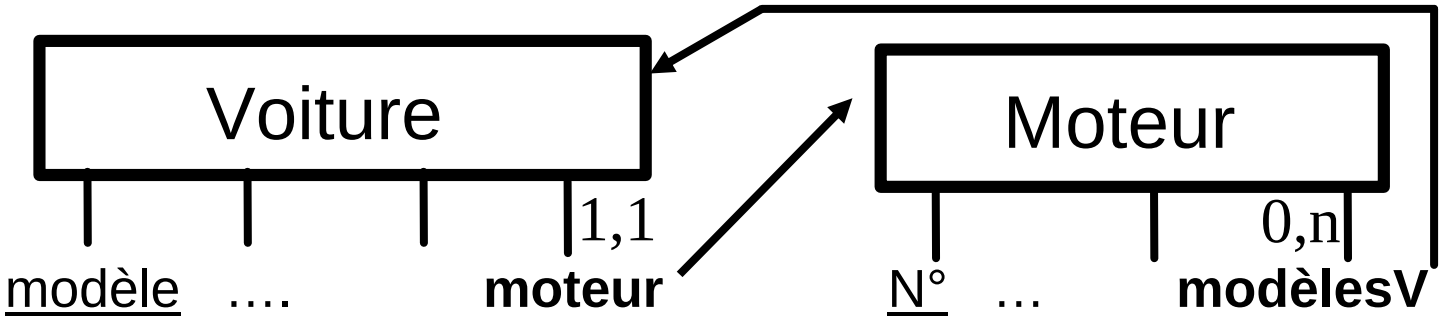
CLASS Moteur

{ N° : STRING ,
puissance : FLOAT ,
nbCyl : INT }

- Attention : 2 types d'attributs :
  - attribut valeur (domaine = STRING, INT… ou complexe)
  - attribut référence (domaine = une classe d'objets)

Contraintes de composition

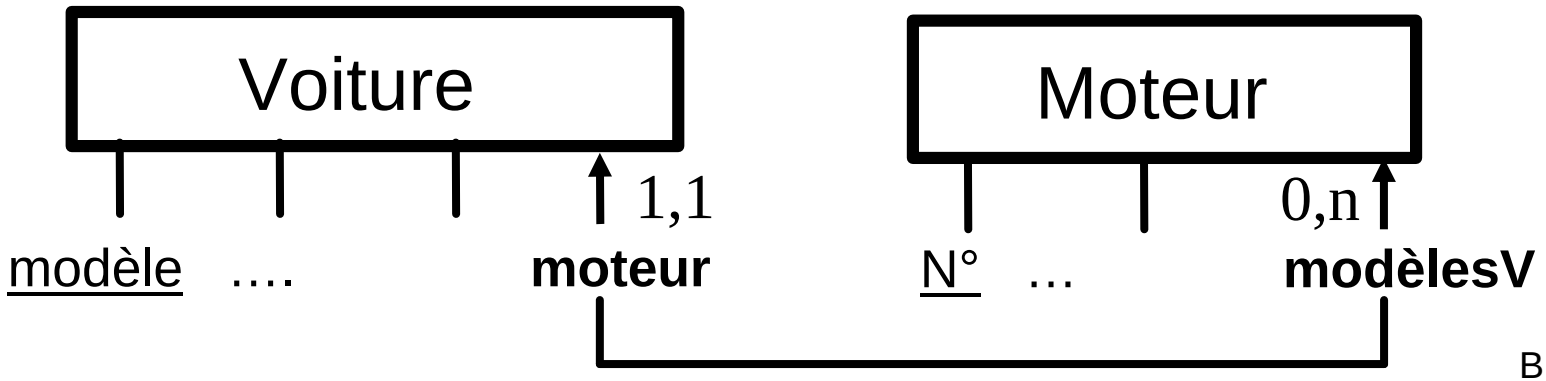
- objet composant : partagé / non partagé
- objet composant : dépendant / non dépendant
  - destruction composite => destruction composant
- cardinalités :
  - minimale, maximale
  - inverses (=> partagé / dépendant)
- lien inverse



Liens inverses gérés par le SGBD OO

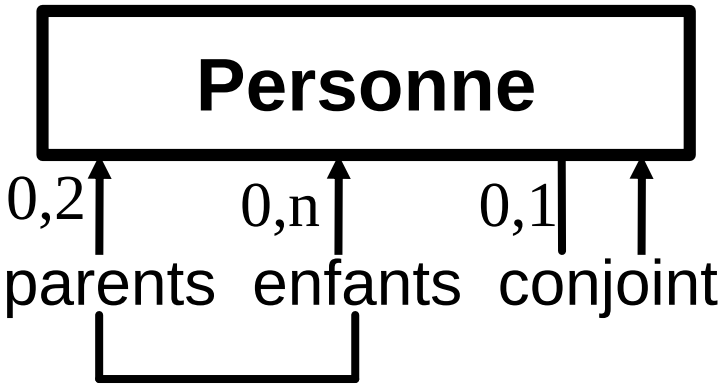
- Certains SGBD OO gèrent les liens de composition inverses
  - maj du lien inverse assurée par le SGBD OO
- CLASS Voiture
  { modèle : STRING ,
  ….. ,
  moteur : Moteur INVERSE Moteur.modèlesV }

CLASS Moteur
  { N° : STRING ,
  ….. ,
  modèlesV: SET Voiture INVERSE Voiture.moteur }

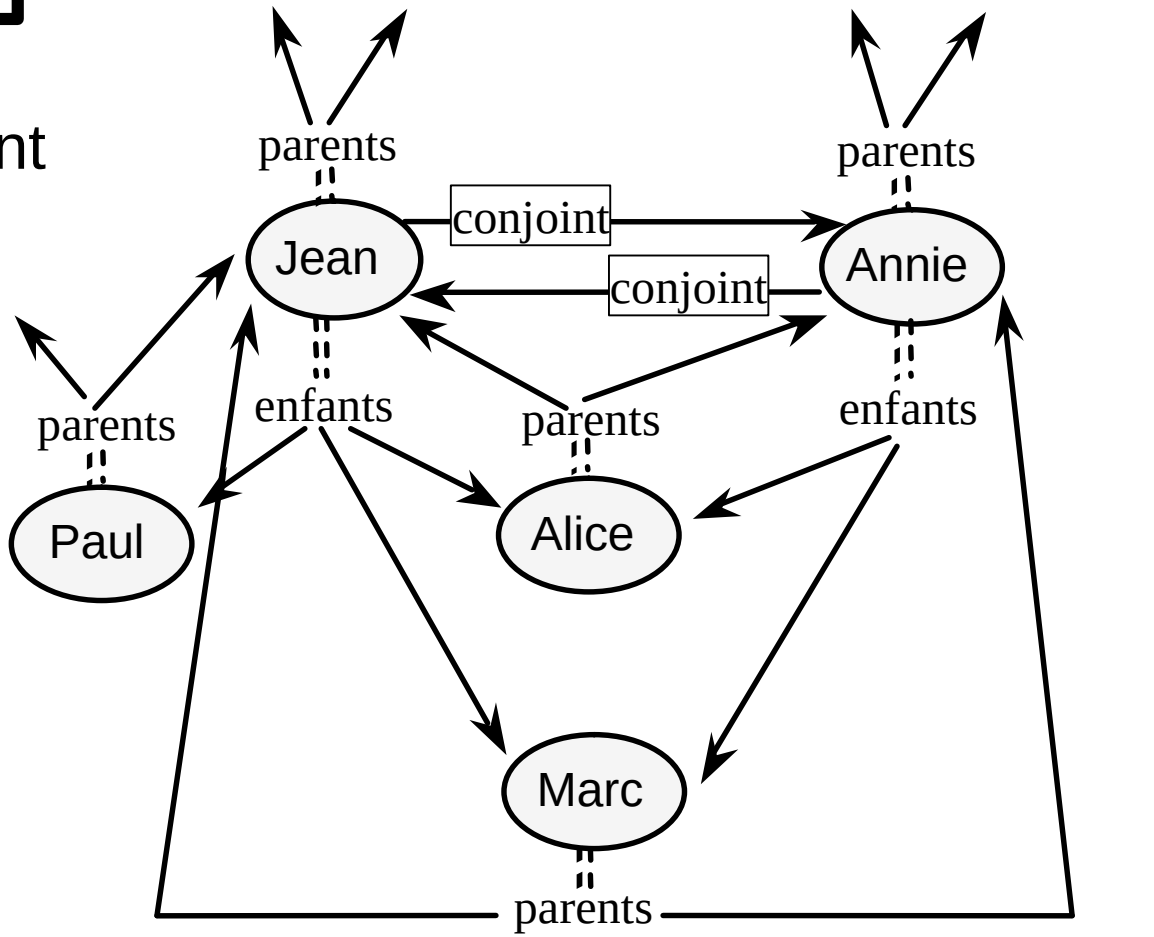


Base d'objets : réseaux d'instances

Schéma



BD



Intégrité référentielle

- Les SGBD OO vérifient les affectations :
  - attribut-référence = x
  - UPDATE Voiture
    WHERE modèle = 'Golf GTI'
    SET moteur = x
  - => x doit être un (des) oid de la classe référencée
- Suppression d'un objet composant
  - Le SGBD OO devrait mettre NULL dans les attributs référence des objets composites
  - MAIS c'est rarement fait …
  - SELECT v.moteur.N°
    FROM v IN Voiture
    WHERE modèle = 'Golf GTI' peut planter !

Impact sur le SGBD des liens de composition :

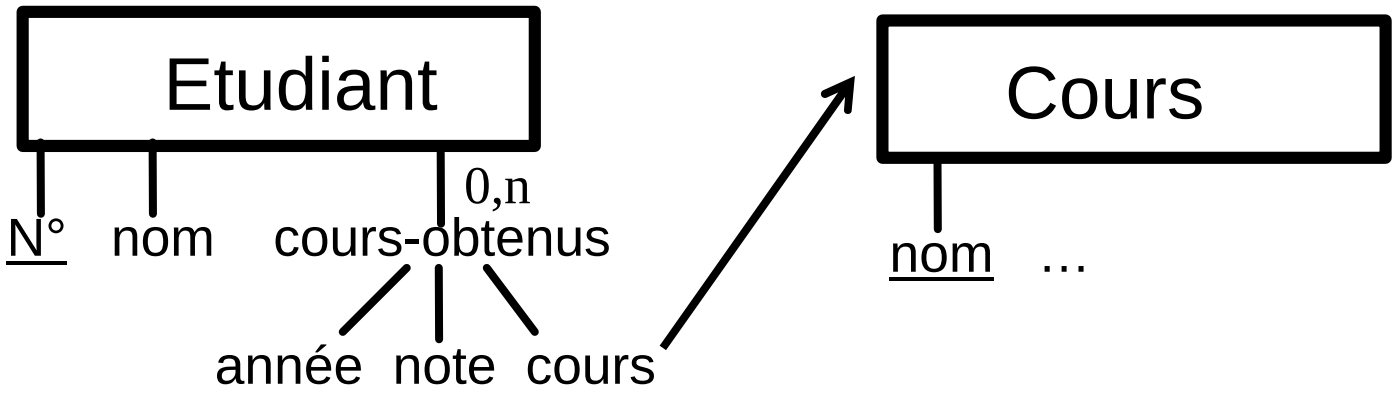
- Assurer l’intégrité référentielle
- Stockage des objets composants par rapport à leur objet composé
- Unité de verrouillage : objet composé / objet composant
- Transactions emboîtées

Lien de composition / association

BD OO	Entité Association
■ Sémantique :	
"composition"	association générique
Voiture –> Moteur	Etudiant --inscription-- Cours
■ orienté	non orienté
accès facile objet composé –> objet composant	
accès **difficile** objet composant –> objet composé	
■ binaire	n-aire
■ sans attribut	avec attribut
■ card. quelconques	card. quelconques

Lien de composition / association (2)

- Certains SGBD OO permettent les attributs référence en attributs composants



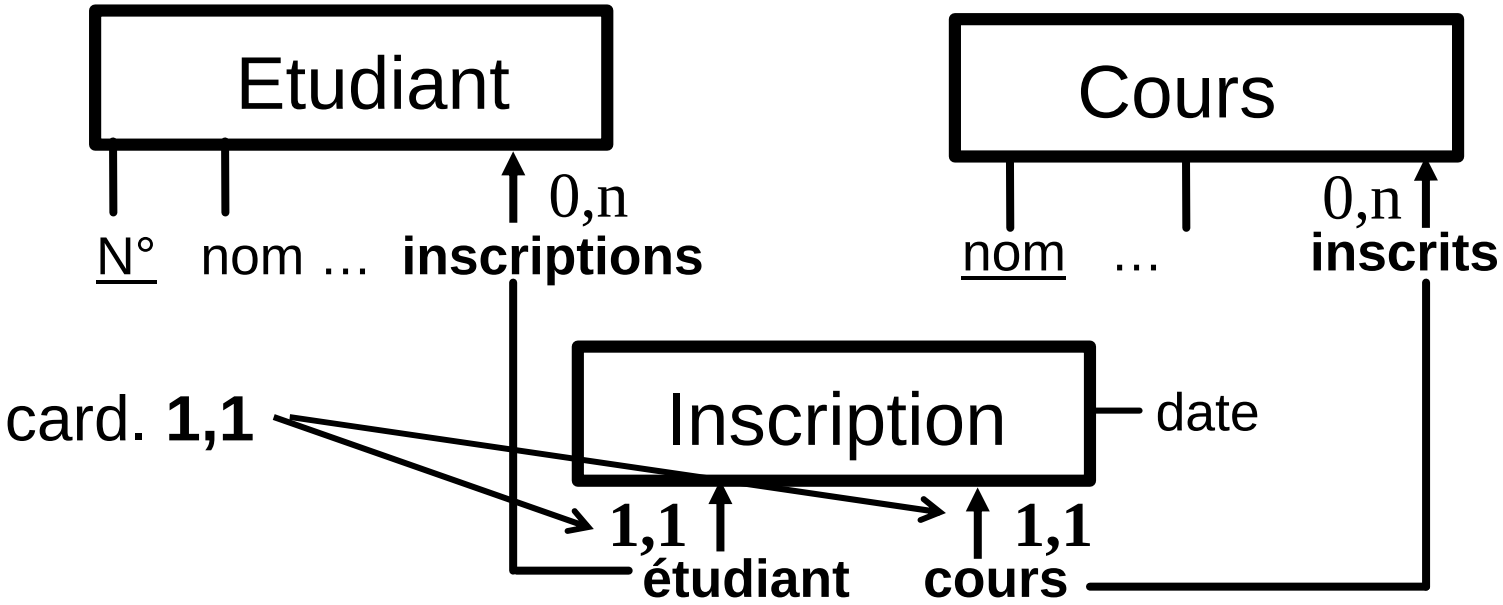
- En fait c'est un lien attribut — classe d'objet
  Lien inverse ?
- ODMG n'autorise les attributs référence qu'au premier niveau

  syntaxe :

  RELATIONSHIP nom-att-ref : [SET | LIST] nom-classe
  [INVERSE nom-classe.nom-att-ref2]

Représentation des associations

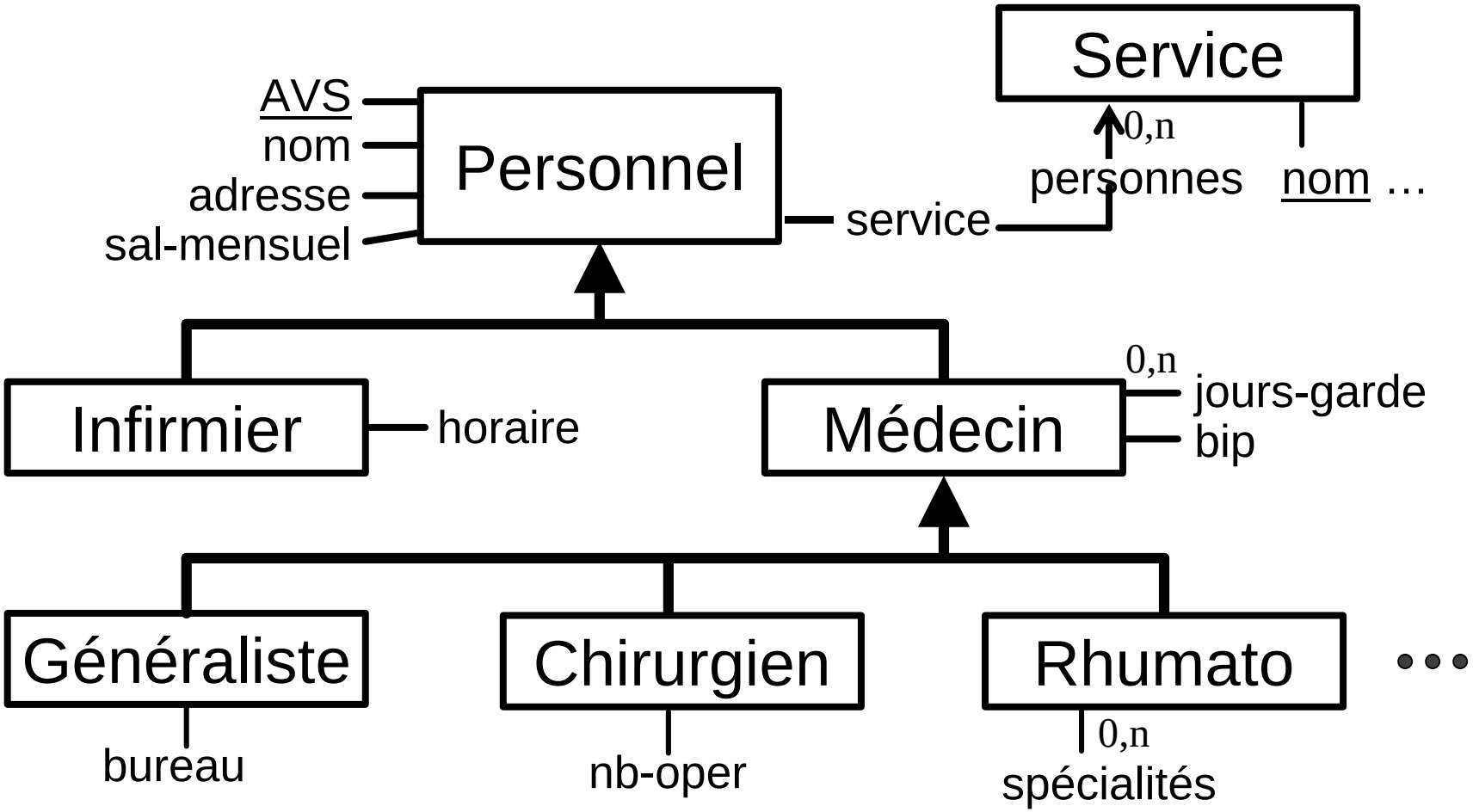
- Associations binaires sans attribut
  lien(s) de composition dans le sens des requêtes
- Associations n-aire et/ou avec attributs
  une classe d'objets avec un lien de composition par rôle (dans le sens des requêtes)
- Exemple : inscription (avec date) d'un étudiant à un cours



HIERARCHIE D'HERITAGE

- Objectif des LP OO : réutilisation (réduire le coût de développement)
  ==> Héritage des propriétés
  Redéfinition des propriétés pour les adapter
- Objectif des BD OO : représentions multiples du même objet
  - Annie est :
    - membre du personnel de l'hôpital
    - médecin
    - chirurgien
    - et en ce moment un patient
- "lien is-a" ou "lien de généralisation / spécialisation" ou "lien d'héritage"

Exemple : le personnel d'un hôpital



Attention : 2 types de flèches : flèches minces : composition
flèches épaisses : is-a

Propriétés des liens is-a

- Inclusion des populations
  - Tout objet d'une sous-classe est aussi objet de sa (ses) sur-classe
  - Exemple : un objet de la classe Médecin est aussi un objet de la classe Personnel
- Héritage des propriétés
  - La sous-classe hérite des :
    - attributs valeur
    - attributs référence
    - et des méthodes

    de sa (ses) sur-classe(s)
  - Exemple : Infirmier a pour attributs : AVS, nom, adresse, sal-mensuel, service et horaire

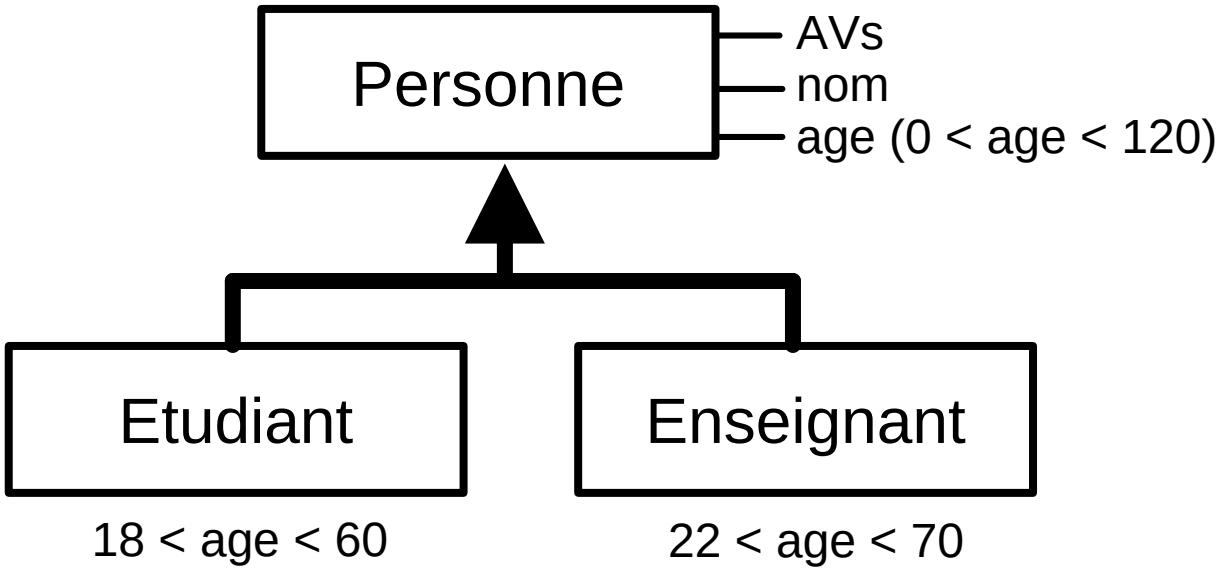
Propriétés des liens is-a (suite)

- Substituabilité
  - On peut toujours employer un objet spécifique à la place d'un objet générique
  - Exemple : ajouter au Service de réanimation un infirmier, un médecin…
- Sous-typage

  Une sous-classe peut avoir des :
  - propriétés supplémentaires
    - Exemple : Infirmier a l'attribut horaire
  - des propriétés redéfinies
    - domaine d'un attribut hérité plus spécifique dans la sous-classe
    - code d'une méthode héritée adapté à la sous-classe

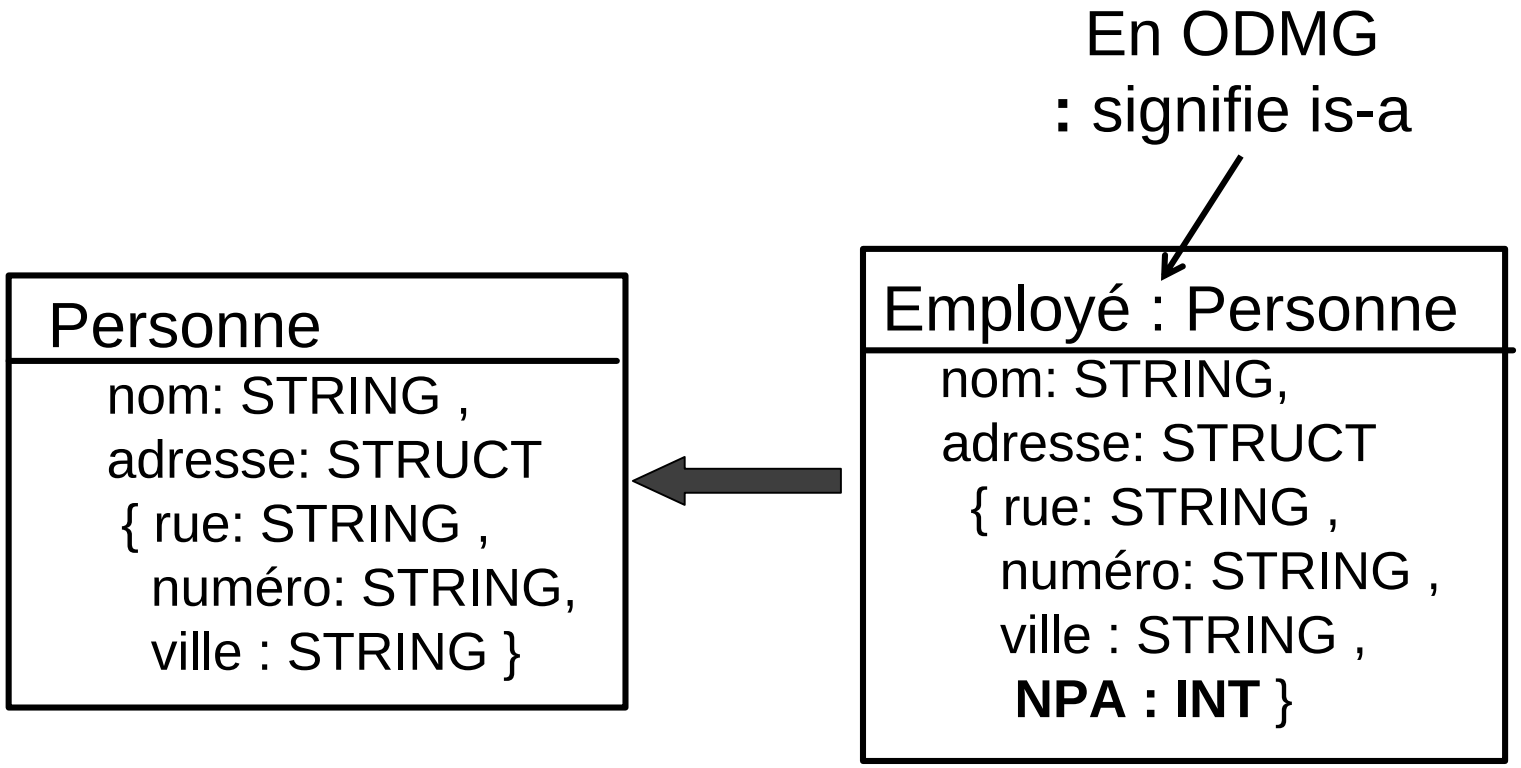
Redéfinition des attributs

- Redéfinition d'un attribut dans une sous-classe
  - nouvelle définition pour l'attribut
  - type de l'attribut redéfini doit être un sous-type
    - domaine et/ou cardinalites restreints
    - attribut complexe complèté
  - n'existe pas dans tous les SGBD OO
- Exemple de domaine restreint :



Redéfinition d'attribut

- Exemple d'attribut complexe complété



- Il existe d'autres types de redéfinition, plus souvent employés pour les méthodes (voir § Méthodes)

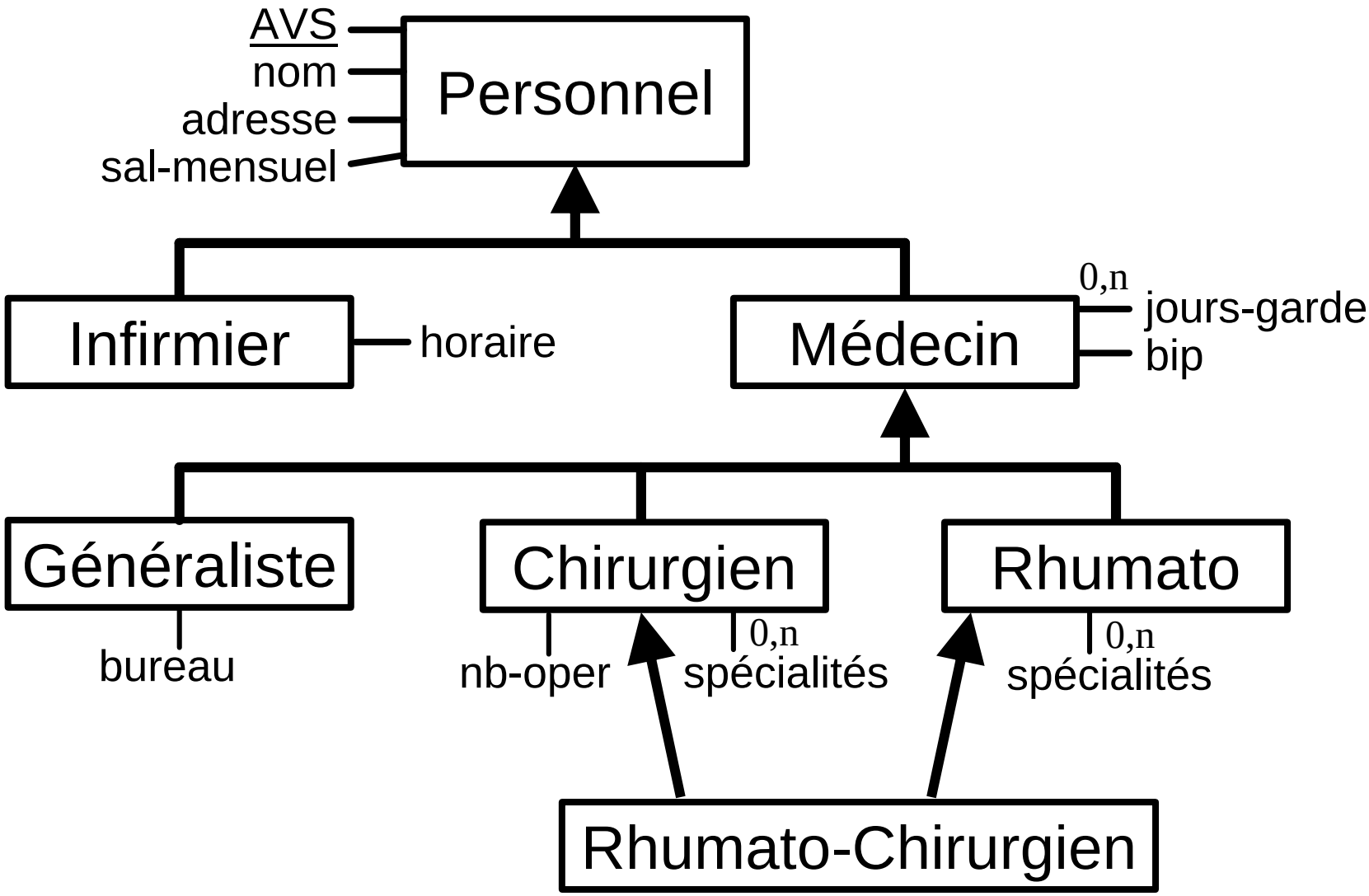
Restrictions à la hiérarchie

- Dynamique ?
  - Un objet peut-il changer de classe ?
    - un infirmier devient médecin
    - on apprend le type d'un personnel: c'est un médecin
  - Implémentation plus complexe (instances de formats différents)

  => Les SGBD OO offrent en général des hiérarchies statiques
- Instanciations multiples ?
  - Un objet du monde réel peut-il être décrit par plusieurs instances de classes différentes (non sur/sous-classes)
  - Exemple : Annie est Rhumatologue et Chirurgien
  - Implémentation plus complexe

  => En général non : sous-classe commune obligatoire

Héritage multiple



Conflits d’héritage multiple

- Quelles spécialités pour les Rhumato-Chirurgiens ?
- Solutions employées par les SGBD OO
  - Interdiction

    => renommer l’attribut / méthode qui pose problème
  - préfixage automatique des noms des attributs ou méthodes par le nom de la sur-classe
  - choix par le système (toujours la première sur-classe dans la déclaration textuelle)
  - choix par l'utilisateur
    - statique : à la définition du schéma
    - dynamique : lors des accès

Implémenter les hiérarchies

- LMD :
  - accès à la population propre / globale d'une classe
    - SELECT * FROM Personnel
      - les personnels qui ne sont ni infirmier ni médecin
      - tous les personnels

        selon quel format :
        - Personnel
        - ou : Personnel, Médecin, Chirugien…
  - changement de classe
- Stockage d'un objet :
  - avec héritage effectué : 1 objet = 1 enregistrement (dans la sous-classe la plus spécifique)
  - sans héritage : 1 objet = 1 enregistrement par classe (sa classe et ses sur-classes)

POPULATION ET PERSISTANCE

- Objectifs :
  - BD : gérer des ensembles d'objets permanents : "populations"
  - LPOO : permettre aux utilisateurs de manipuler de la même façon des objets temporaires et des objets permanents

  => Persistance et classification peuvent être indépendants
- SGBD classiques :
  - Relation, record type, type d'entité … =

    1) définition de la structure des occurrences potentielles

    2) récipient contenant toutes les occurrences existantes, permanentes par définition
- LPOO :
  - Classe = 1) usine pour fabriquer des objets de même type
  - Les objets sont temporaires
    - durée de vie = celle de leur programme (sauf s'ils sont stockés dans un fichier)

Deux approches : BD , LP

- SGBD OO issu du monde BD
  - classe = 1) + 2)
  - annie := Médecin (AVS : 123456 , nom : 'Rochat' , adresse : …, bip : 222)
    - Médecin(...) : chaque classe a une méthode (constructeur) du nom de la classe qui crée un objet
    - création d'un objet permanent stocké dans la population de la classe
    - rend l'oid de l'objet créé

SGBD OO issu du monde LP

- Objectif : disposer de manière souple de données permanentes ou non
- classe = 1) uniquement
  - annie := Médecin (AVS : 123456 , nom : 'Rochat' , adresse : …)
    - création d'un objet temporaire
    - rend l'oid de l'objet créé
- Le SGBDO fournit des outils aux utilisateurs pour gérer eux-mêmes
  - les populations des classes
    - (où mettre les objets pour les retrouver ?)
    - une classe peut avoir 0, 1 ou plusieurs populations
  - la persistance des objets

Exemple de gestion de population

- Via les collections (SET, LIST ...)
- L'utilisateur crée une (ou des) collection et y insère les objets
- Exemple : les médecins de l'hôpital

m : Médecin ;

lesmédecins : SET Médecin ; *déclaration*

...

m:= Médecin(AVS:123456, nom: 'Rochat', …., bip : 222);

lesmédecins.insert_élément(m) ; *insertion*

...

SELECT x.nom FROM x IN lesmédecins
WHERE x.AVS=123456 *utilisation*

Qualités de la persistance

- Orthogonale aux classes : pour la même classe, on peut avoir des objets permanents et d'autres temporaires
- Orthogonale aux opérations : les mêmes opérations peuvent être appliquées à des objets permanents ou temporaires
- Cohérente : un objet permanent ne peut pas référencer des objets temporaires
- Dynamique : le statut permanent / temporaire peut être changé à n’importe quel moment

Techniques de persistance

Différents modèles de persistance :

- Statique
  - systématique : tout est permanent
  - classe : persistance spécifiée à la déclaration de la classe
  - instance: persistance spécifiée lors de la création de l'instance
- Dynamique
  - explicitement par une commande à n'importe quel moment

    lesmédecins.**save()**
  - par accessibilité à partir de racines de persistance

    "Tout objet composant d'un objet permanent est permanent"

    PersistList.insert_last_élément(lesmédecins)

ODMG - persistance et population

- Approche type BD classique
  - Les objets sont tous toujours permanents
  - Chaque classe a 1 (ou 0) population
  - Si la population existe, les objets sont automatiquement stockés dedans
- CLASS nom-classe
  [EXTENT nom-population]
- En plus, l'utilisateur peut associer des noms permanents à certains objets

  NAME directeur : Personnel

  déclaration d'une variable permanente nommée

  directeur := Personnel (AVS: 1111, nom: 'Muller'…)

  création de l'objet directeur

METHODES ET ENCAPSULATION

- Objectif des méthodes : décrire dans le SGBD :
  - la structure des objets
  - **et les opérations** (méthodes) usuelles sur les objets
  - Même chose pour les types de données définis par l'application
- Intérêt : écrire les opérations une fois pour toutes
- A chaque classe (et type de données) sont associées les méthodes permettant de :
  - accéder
  - mettre à jour
  - manipuler

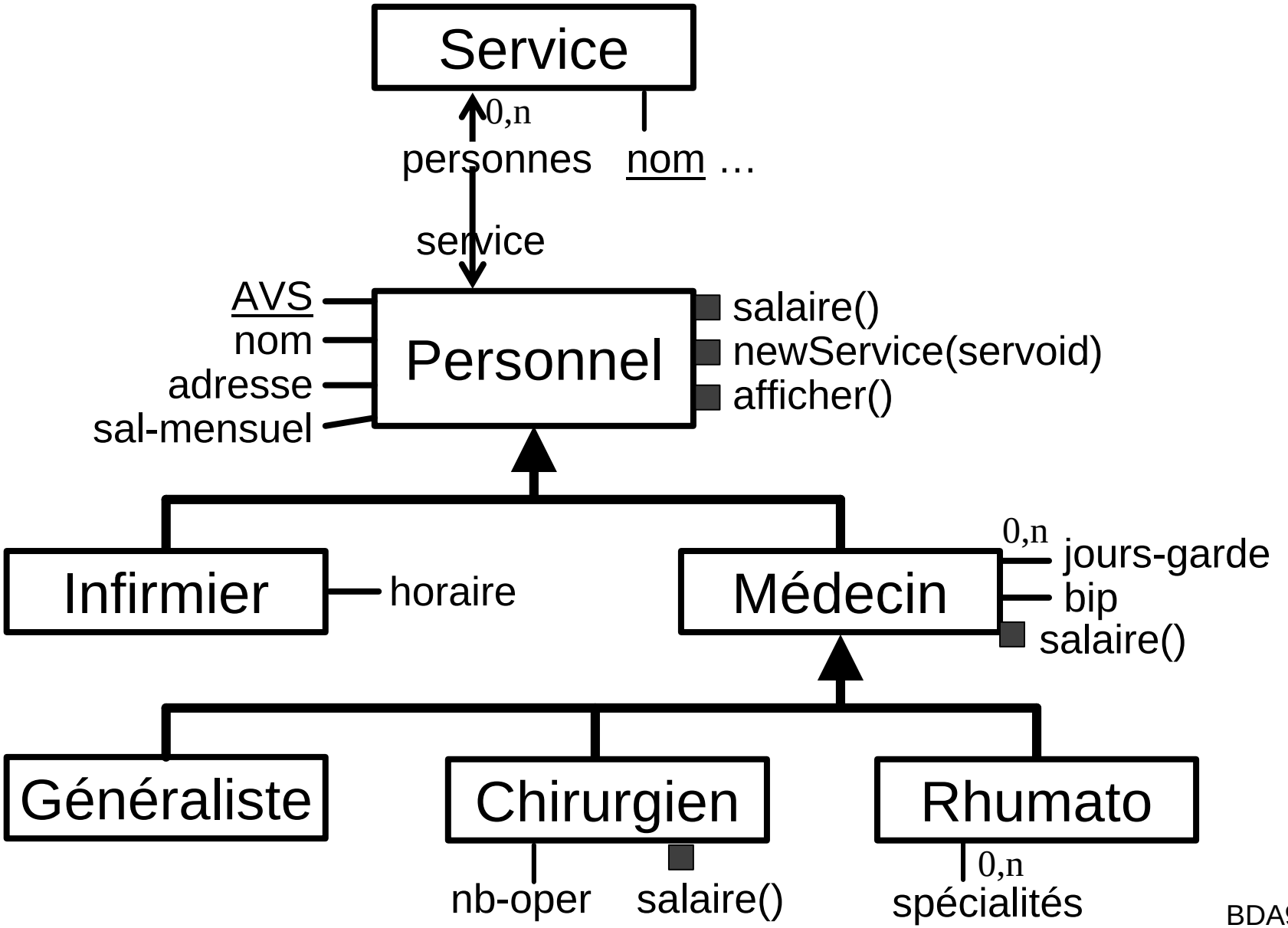
  les objets de la classe (ou les valeurs du type de données)

Méthode

- Signature de la méthode
  - nom de la méthode
  - type du résultat (si existe)
  - paramètres (si existent) : nom et type pour chacun
- Code de la méthode
  - instructions d'un LP OO
  - instructions du SGBD OO
    - requêtes

      SELECT ... FROM … WHERE …
    - mises à jour d'objets
  - appels de méthodes sur d'autres objets

Personnel d'un hôpital avec méthodes



Encapsulation

- Objectif : cacher l'implémentation des classes pour
  - faciliter la réutilisation des classes : il suffit d'en connaître l'interface
  - permettre l'évolution de l’implémentation des classes : si elle change, l’application doit seulement être re-compilée
- Principe : depuis l'extérieur de l'objet seules les signatures de ses méthodes sont visibles
- Implantation cachée
  - structure des objets
  - code des méthodes

Exemple d'encapsulation

CLASS Personnel

- Interface visible

  INT salaire()

  VOID newService(servoid : Service)

  VOID afficher()

  signatures des méthodes

- Implémentation invisible

  ATTRIBUTE AVS : STRING ;

  ATTRIBUTE nom : STRING ;

  ATTRIBUTE adresse : STRING ;

  ATTRIBUTE sal_mensuel : INT ;

  RELATIONSHIP service : Service
  INVERSE Service.personnes

  structure des données

Exemple d'encapsulation (2)

Implémentation invisible (suite) : code des méthodes

salaire ()

```
{ return sal_ mensuel }
```

newService (servoid: Service)

```
{ self.service := servoid }
```

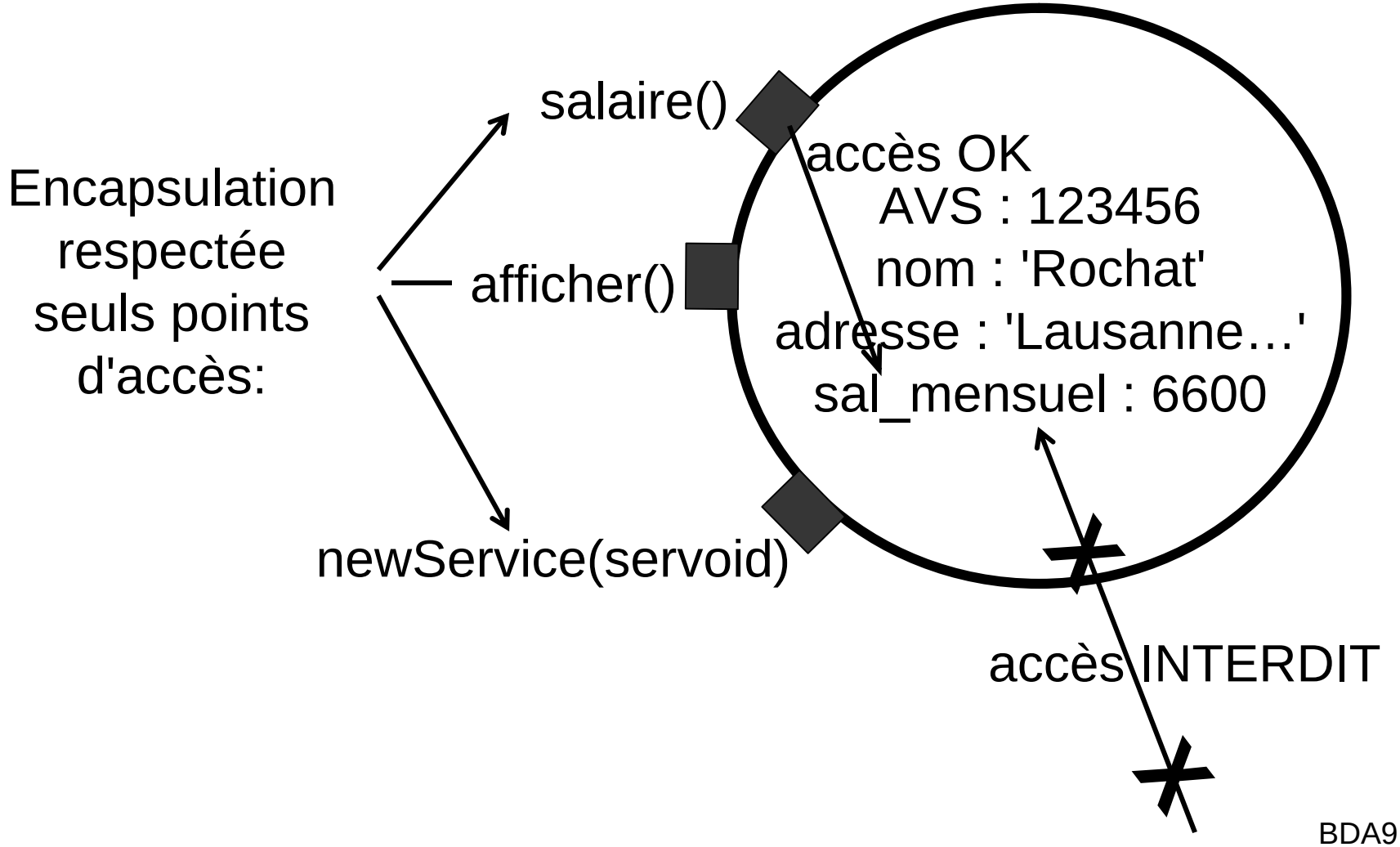
afficher ()

```
{
  PRINT('AVS:', self.AVS) ;  PRINT('nom:', self.nom) ;
  PRINT('adresse:', self.adresse) ;
  PRINT('salaire mensuel:', self.sal_mensuel) ;
}
```

Encapsulation : seul l'objet lui-même (c-à-d les instructions de ses méthodes) peut accéder à ses attributs

Exemple d'encapsulation (3)

- Un objet de la classe Personnel



Impact sur l'interface utilisateur

- Interface procédurale :
  LP + messages d'appel des méthodes
  navigationnelle : en suivant les liens de composition et en balayant les collections
- LMD déclaratif (exemple OQL) :
  - L'encapsulation est contraire au principe sous-jacent des BD classiques : accès libre de tous à toutes les données
  - Si les requêtes ne sont pas réutilisées, l'encapsulation est inutile
  - => encapsulation ± stricte, par exemple :
    - depuis LP OO : encapsulation
    - depuis requêtes : pas d'encapsulation

Redéfinition des méthodes

- Objectif : adapter le code à la sous-classe
- Signature inchangée
- Exemple
  - **Personnel** salaire() = self**.**sal_mensuel
  - **Médecin** salaire() = self**.**sal_mensuel +
    (self**.**jours_garde * PrimeJG)
  - **Chirurgien** salaire() = self**.**sal_mensuel +
    (self**.**jours_garde * PrimeJG) +
    (self**.**nb_oper * PrimeOp)
- Sans redéfinition => méthodes de noms différents
  - **Personnel** salaire()
  - **Médecin** medSalaire()
  - **Chirurgien** chirurSalaire()

Salaire mensuel de tout le personnel

- Sans redéfinition

```
SELECT p.salaire()
   FROM p IN lespersonnes
   WHERE NOT (p IN lesmédecins)

SELECT p.medSalaire()
   FROM p IN lesmédecins
   WHERE NOT (p IN leschirurgiens)

SELECT p.chirurSalaire()
   FROM p IN leschirurgiens
```

- Avec redéfinition : même nom de méthode, codes différents

```
SELECT p.salaire()
   FROM p IN lespersonnes
```

Edition de liens

- SELECT p.salaire()
  FROM p IN lespersonnes
- La méthode salaire() est redéfinie dans plusieurs sous-classes
- Quelle méthode salaire() exécuter ?
- Solution 1 : celle de la classe déclarée
  - choix statique à la compilation
  - Exemple => même formule de calcul du salaire pour tous (= sal_mensuel)

Solution 2 : liaison dynamique

- Choisir la méthode de la classe la plus spécialisée contenant l'objet
  - choix lors de l'exécution seulement
  - "liaison dynamique"
  - Exemple : formule de calcul du salaire particulière à la sous-classe de chaque personne
- => Instanciation unique des objets pour éviter toute ambiguïté
- Il faut décrire dans le schéma toutes les intersections de classes possibles : Chirurgien-Rhumato, etc

Redéfinition / Surcharge

- La liaison dynamique n'est pas toujours souhaitable
  Cela dépend des programmes d'application
- Certains SGBD OO proposent différents types de re-déclaration des propriétés :
  - redéfinition avec liaison dynamique
    - le résultat doit être compatible avec celui de la sur-classe
  - surcharge sans liaison dynamique
    - le résultat peut être quelconque

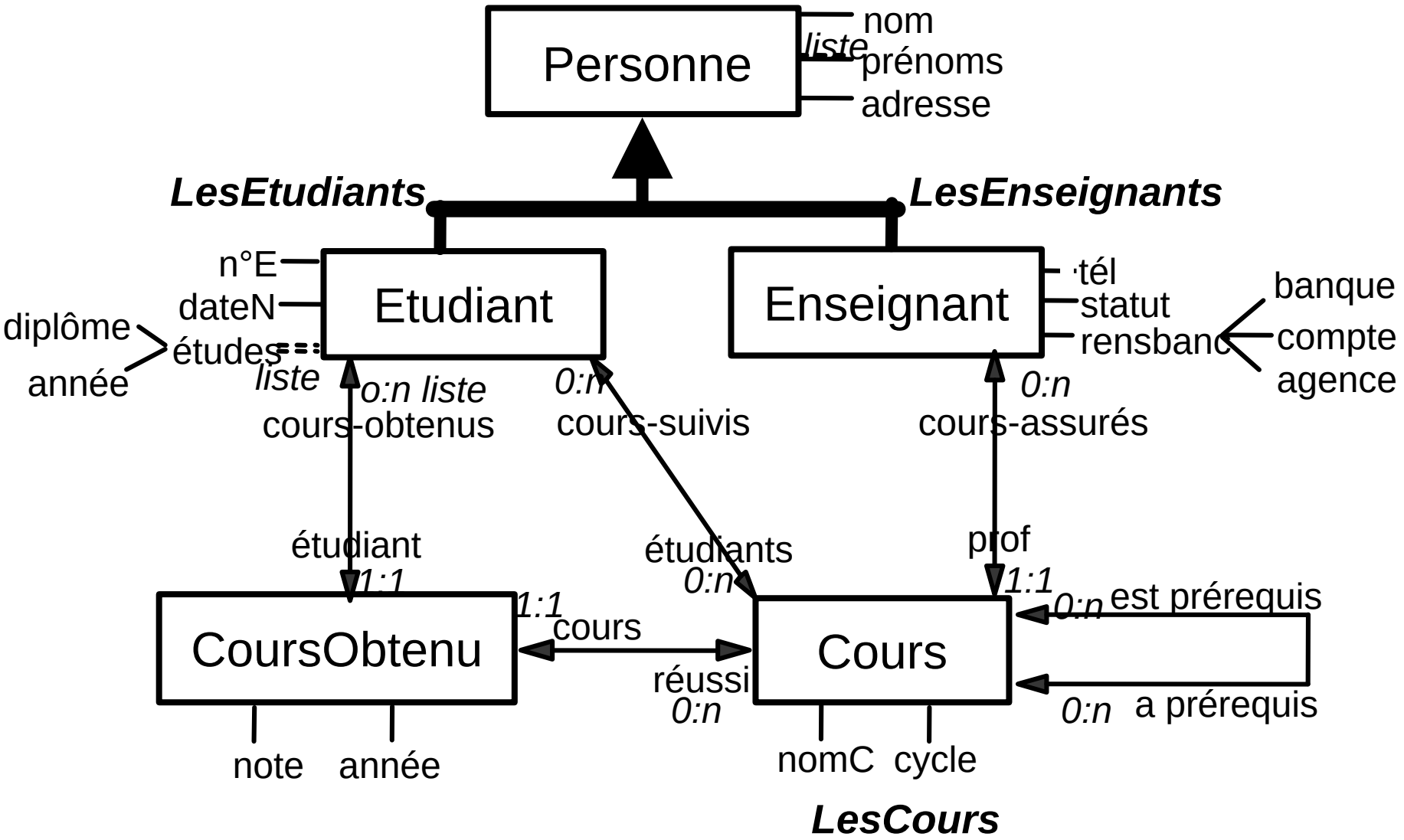
Redéfinition / Surcharge (2)

- Exemple :
  - **Personnel** salaire() = self**.**sal_mensuel (1)
  - **Médecin** salaire() = self**.**sal_mensuel + (2)
    (self**.**jours_garde * PrimeJG)
- Un médecin : Muller d'AVS 12345
- SELECT p.salaire() FROM p IN lespersonnes WHERE AVS=12345
  - salaire() redéfini dans Médecin => calcul (2)
  - salaire() surchargé dans Médecin => calcul (1)
- SELECT p.salaire() FROM p IN lesmédecins WHERE AVS=12345
  - => calcul (2)

Bibliothèques de classes (ou types)

- Collections
  - insert_element(e)
  - remove_element(e)
  - …
- LIST
  - insert_first_element(e)
  - retrieve_element_at(position) –> element
  - …
- Types géographiques (Point, Ligne, Polygone)
  - inside(g) –> BOOLEAN
  - adjacent(g) –> BOOLEAN
  - distance(g) –> FLOAT
  - …
- Les SGBDO offrent des bibliothèques ± complètes

FormaPerm en OO



FormaPerm - remarques

- L'étude des requêtes a montré que tous les liens de composition sont utilisés dans les deux sens
  - Exemple : Enseignant.cours_assurés—prof.Cours
  - Quel est le professeur de tel cours ?
    - Cours.prof —> Enseignant
  - Quels cours donne tel professeur ?
    - Enseignant.cours_assurés —> Cours

- NB Faute de place, les méthodes n'ont pas été représentées sur le diagramme

FormaPerm (1)

```
CLASS Personne
    { ATTRIBUTE nom : STRING ;
    ATTRIBUTE prénoms : LIST STRING ;
    ATTRIBUTE adresse : Tadresse ;
    VOID afficher() ;
    VOID nouvelle_adresse(nvadr : Tadresse) }

TYPEDEF Tadresse  STRUCT
    { ATTRIBUTE rue : STRING ;
    ATTRIBUTE numéro : STRING ;
    ATTRIBUTE ville : STRING ;
    ATTRIBUTE NPA : STRING }
```

FormaPerm (2)

```
CLASS Etudiant : Personne
    EXTEND LesEtudiants
    KEY n°E
    { ATTRIBUTE n°E : INT ;
    ATTRIBUTE dateN : DATE ;
    ATTRIBUTE études : LIST STRUCT Etude
        { année : INT ;
          diplôme : STRING } ;
    RELATIONSHIP cours-obtenus : LIST CoursObtenu INVERSE
       CoursObtenu.étudiant ;
    RELATIONSHIP cours-suivis : SET Cours INVERSE Cours.étudiants ;
    VOID afficher() ;
    VOID inscrire ( nvcours : Cours ) ;
    VOID aobtenu ( nvcours : Cours , note : FLOAT , année : INT ) ;
    INT age()  }
```

FormaPerm (3)

CLASS Cours

EXTEND LesCours

KEY nomC

{ ATTRIBUTE nomC : STRING ;

ATTRIBUTE cycle : INT ;

RELATIONSHIP prof : Enseignant INVERSE Enseignant.cours-assurés ;

RELATIONSHIP étudiants : SET Etudiant INVERSE Etudiant.cours-suivis ;

RELATIONSHIP a-prérequis : SET Cours INVERSE Cours.est-prérequis ;

RELATIONSHIP est-prérequis : SET Cours INVERSE Cours.a-prérequis ;

RELATIONSHIP réussi : SET CoursObtenu INVERSE CoursObtenu.cours ;

VOID afficher() ;

INT nb-inscrits() }

FormaPerm (4)

CLASS CoursObtenu

{ ATTRIBUTE année : INT ;

ATTRIBUTE note : FLOAT ;

RELATIONSHIP cours : Cours INVERSE Cours.réussi;

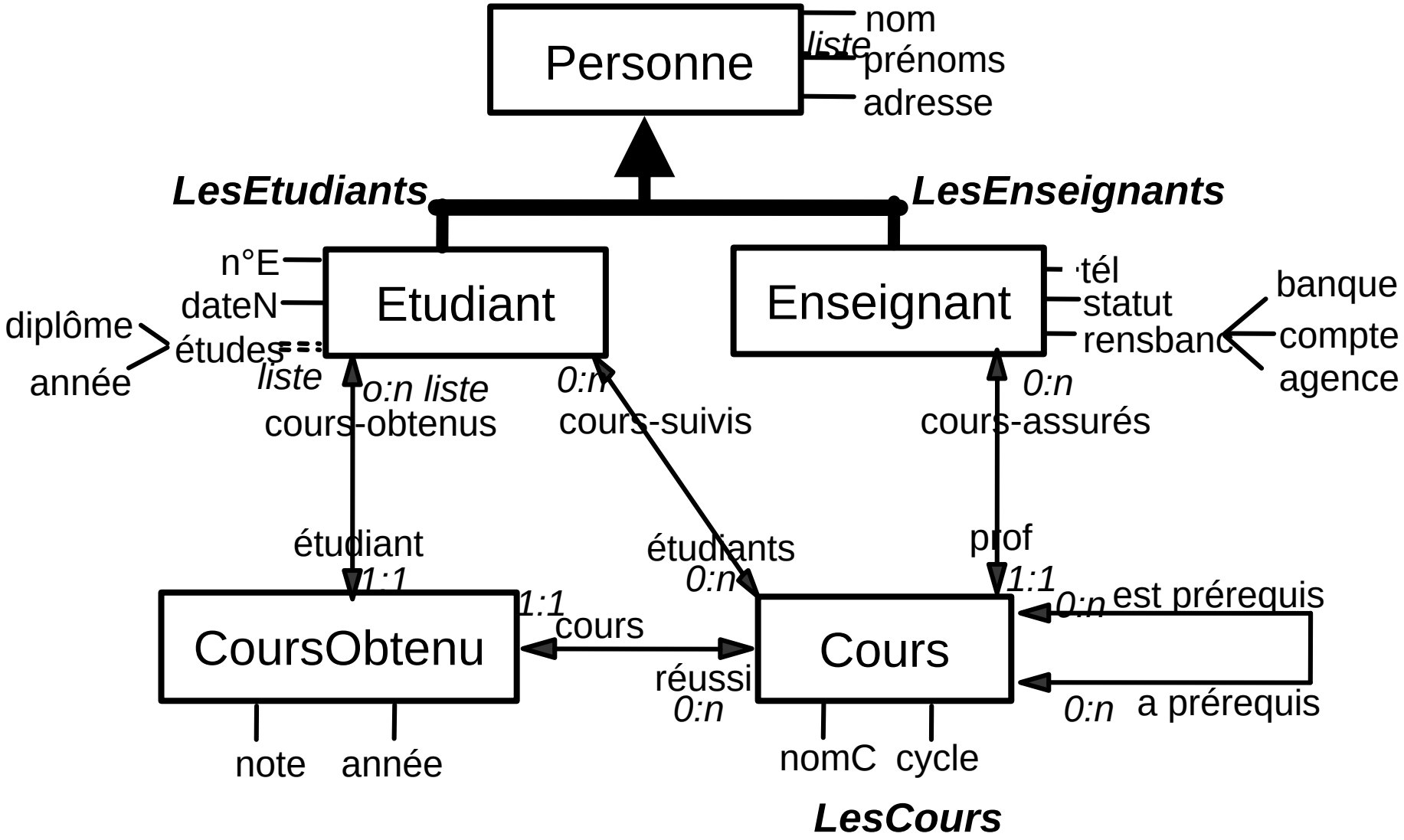
RELATIONSHIP étudiant : Etudiant INVERSE
Etudiant.cours-obtenus }

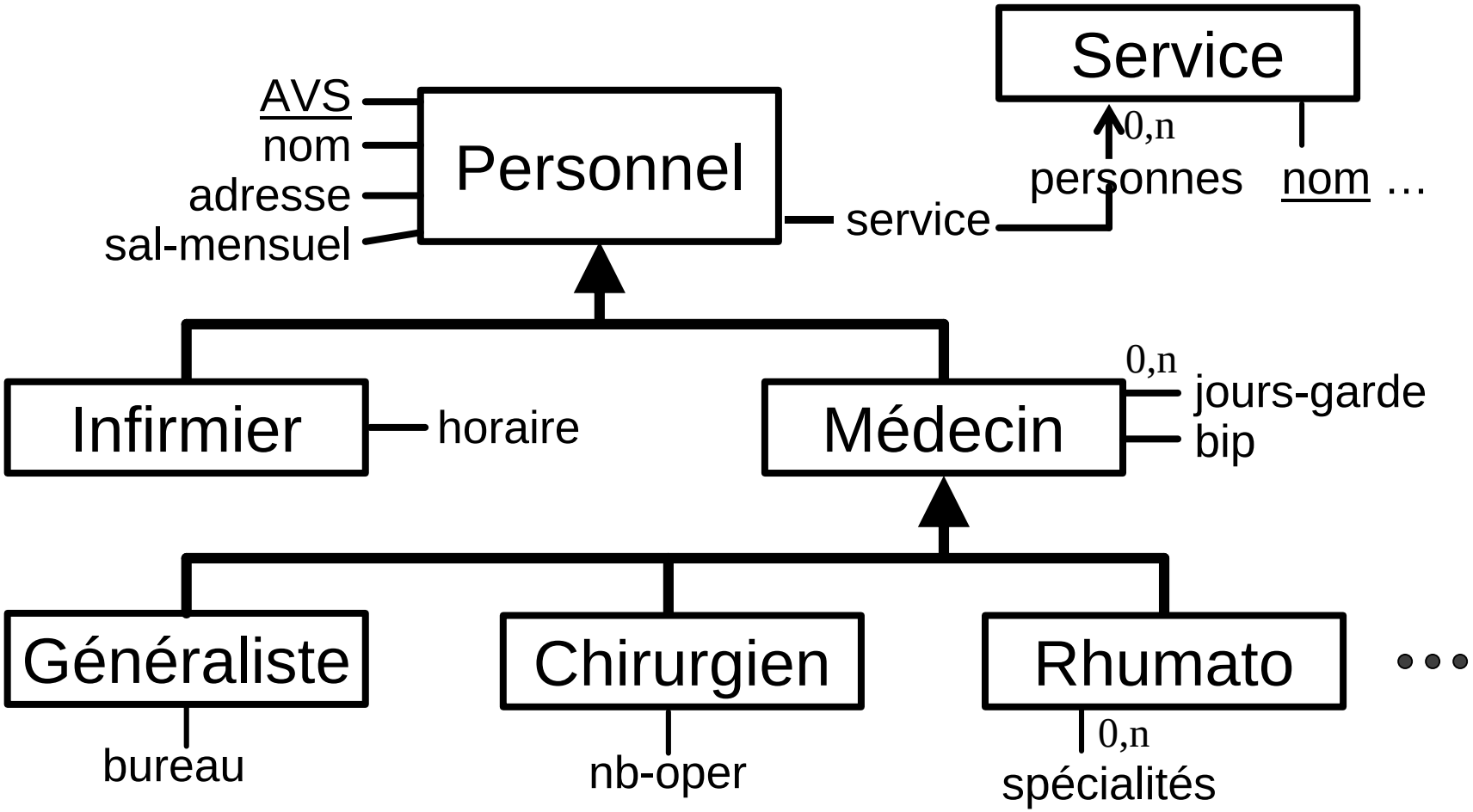
FormaPerm (5)

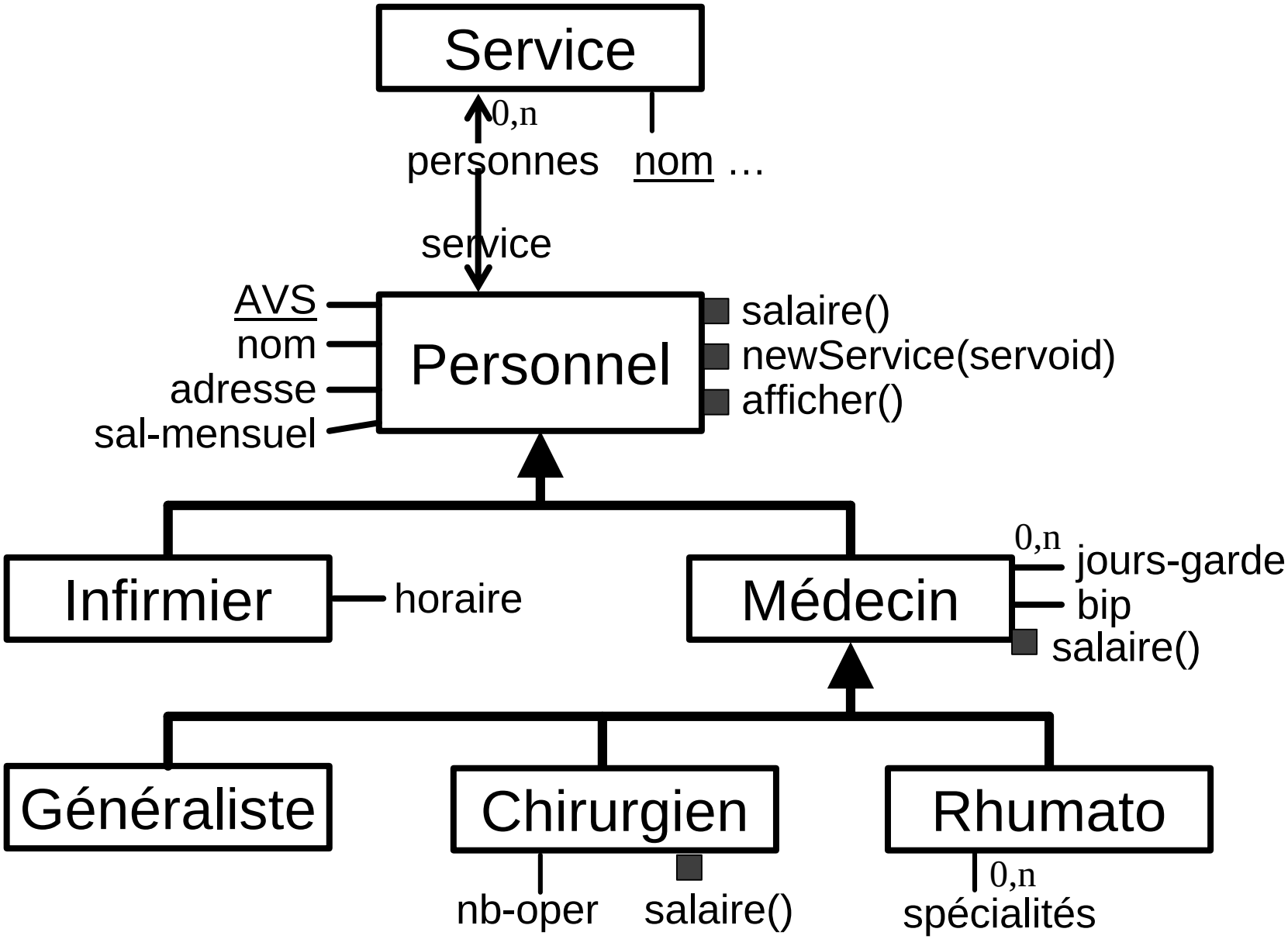
```
CLASS Enseignant : Personne
    EXTENT LesEnseignants
    { ATTRIBUTE tél : INT ;
     ATTRIBUTE statut : ENUM ( "prof", "assist" ) ;
     ATTRIBUTE rens.banc : STRUCT RensBq
         { banque : STRING ;
          agence : STRING ;
          compte : INT } ;
    RELATIONSHIP cours-assurés : SET Cours INVERSE
       Cours.prof ;
    VOID afficher() ;
    VOID assure (nvcours : Cours) ;
    VOID nassureplus  (oldcours : Cours) }
```

CONCLUSION

- Objectifs atteints
  - meilleure représentation du monde réel
  - réutilisation
  - efficacité pour les applications nouvelles
- MAIS
  - Les SGBDO ne sont pas adaptés à tout type d'application !
  - Méthodologies de conception incomplètes
    - normalisation de la structure, conception des méthodes
  - Compétition entre standards
  - Absence de théorie, formalisation
  - Vues
  - Evolution du schéma
  - Versions, temps
  - Migration difficile des SGBD classiques aux SGBDO


Personne
nom
liste
prénoms
adresse
LesEtudiants
LesEnseignants
Etudiant
Enseignant
n°E
dateN
diplôme
études
année
liste
tél
statut
rensbanc
banque
compte
agence
0:n liste
cours-obtenus
0:n
cours-suivis
0:n
cours-assurés
étudiant
1:1
étudiants
0:n
prof
1:1
1:1
cours
0:n
est prérequis
CoursObtenu
Cours
réussi
0:n
0:n
a prérequis
note
année
nomC
cycle
LesCours


Service
AVS
nom
adresse
sal-mensuel
Personnel
0,n
personnes
nom …
service
Infirmier
horaire
0,n
Médecin
jours-garde
bip
Généraliste
bureau
Chirurgien
nb-oper
Rhumato
0,n
spécialités


Service
0,n
personnes
nom …
service
AVS
nom
adresse
sal-mensuel
Personnel
salaire()
newService(servoid)
afficher()
Infirmier
horaire
0,n
Médecin
jours-garde
bip
salaire()
Généraliste
Chirurgien
nb-oper
salaire()
Rhumato
0,n
spécialités