Object-Oriented Database Systems

- Motivation **(OODBS)**
  - Relational Models not adequate for the need of
    - CAD (Computer-Aided Design)
    - CASE (Computer-Aided Software Engineering)
    - Geographic Databases
    - Multimedia Databases

⇨

- **Richer data types needed** (images, audio, video, geographical data, text)
- Need to model complex objects (design for engineering of car in CAD, newtal documents)
- Need for better generalizational inheritance
- Long-duration transactions (problem of waits/locks)
- Temporal evolution of data (version control)

Object-Oriented DBS Concepts

Objects – Real World Entities ← Like entities in an ER diagram

Encapsulate state and behavior

State: set of attribute values

Behavior: Set of Methods ← Operations that action objects may be uniquely specialized

↑ Includes methods for creation and destruction of objects

⇒ Objects offer **encapsulation** of both attributes and specialized operations/methods

OODBS Concepts (Continued)

CLASS — group of objects sharing the same attributes and methods

e.g. employee …

department …

INSTANCE — Individual, uniquely identified objects with attribute values

e.g. employee25 ('John Smith', M, 39, …)

(analogous to entity-scheme == class

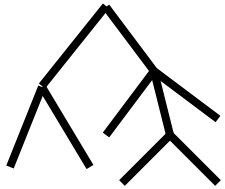
entity tuple values == instance

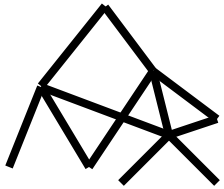
Class	:	Object(instance)	(OO model)
Table	:	Tuple	(Rational model)
Entity Set	:	Entity	(ER model)

CLASS INHERITANCE

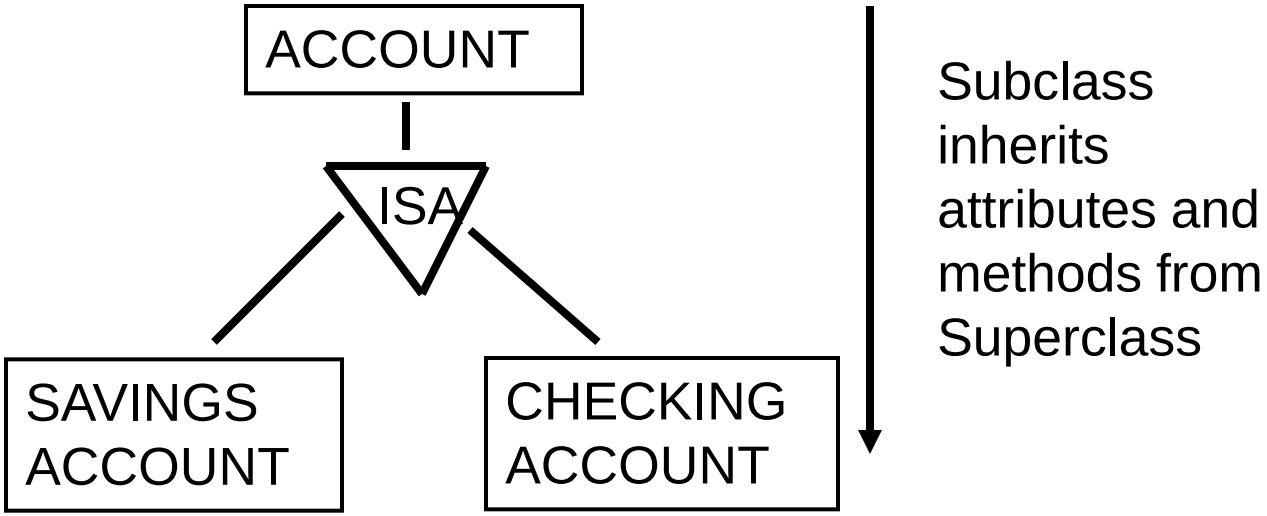
Classes organized in a

TYPE HIERARCITY (Tree or Directed Acyclic Graph)

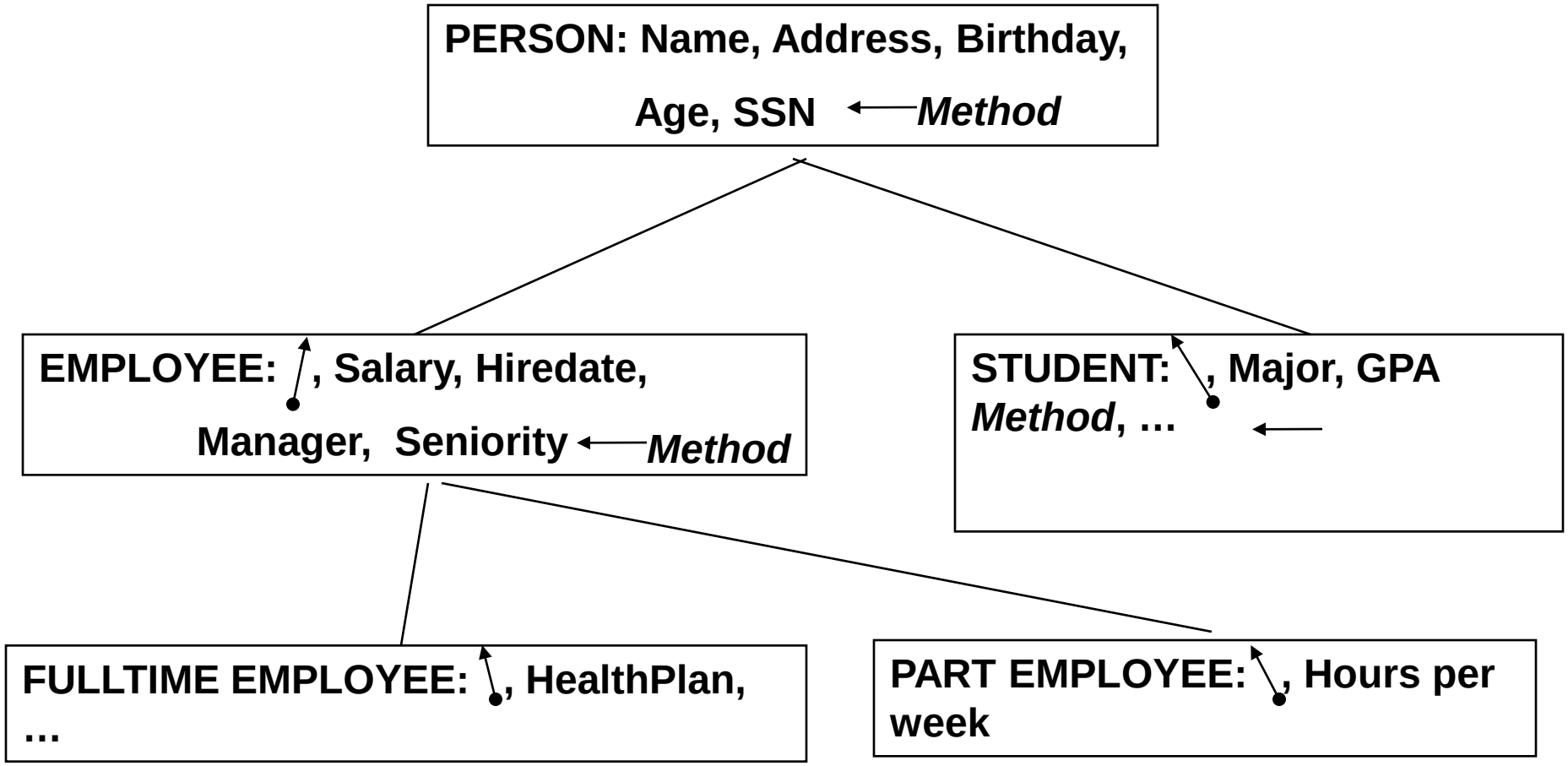




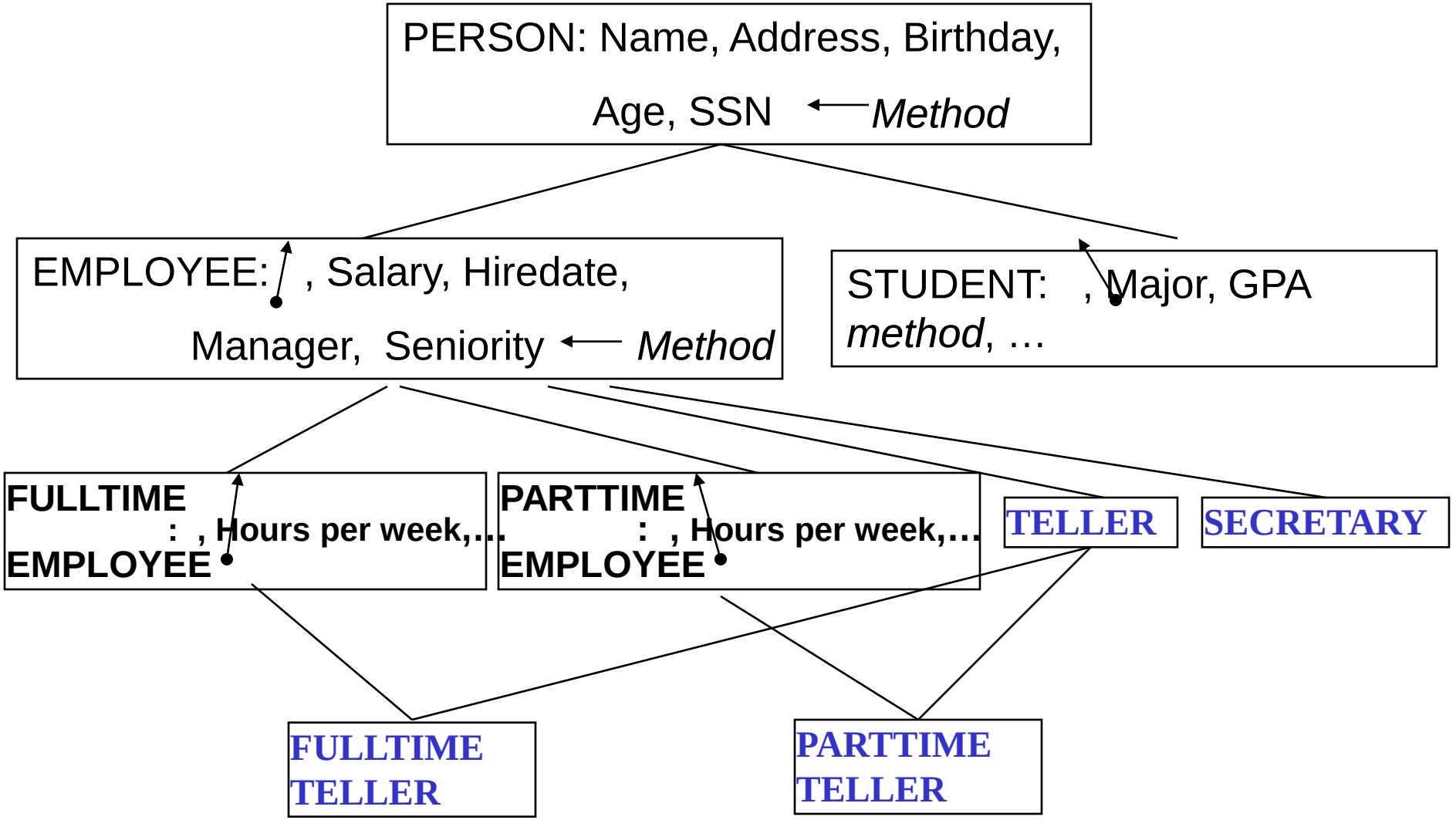
Like ER Diagram



CLASS INHERITANCE



CLASS INHERITANCE



Problem of Multiple Inheritance

- **Inherit Attributes, methods from:**
  - Both/ALL superclasses
  - Only dominant superclass (all cases or only where conflict)
  - Ad hoc

Which is most appropriate for the employee case?

- **SELECTIVE INHERITANCE**

  EXCEPT (attribute/method)

  Listed in subclass to indicate attributes not to be inherited

OBJECT INDENTITY

Remains invariant once object is created

→ Unique permanent object ID number

⇒ Unlike Relational Model where

Object ID is derived exclusively from data values

(Tuples with the same values ≡ same)

Advanced Concepts

- **Polymorphism (Operator overloading)**

  Geometric database with multiple methods for computing 'area' dependent on implementation

  (late binding of method code)

- **VERSION CONTROL**

  (e.g. Attribute values change over time,

  Maintain version graph to capture multiple states)

  (like software version control)

SAMPLE OODBS

- O2 (O2 Technology) -- 1991
- ObjectStore (Object Design Inc.) -- 1991

Data Definition in O2

- **Atomic data types (char, int, …)**
- **type constructors (tuple, list, set, unique set)**
- **Class definitions**
  - attributes (have type)
  - methods (have defining case)
  - inherit <supertype> ← multiple inheritance of all superclass attributes
- **Objects (member of a class)**

Persistent or transient ← temporary

↑ remain after database shutdown

OODBS QUERY LANGUAGES

- **Convert/Export O2 DBs into persistent**

  object store for use with C++

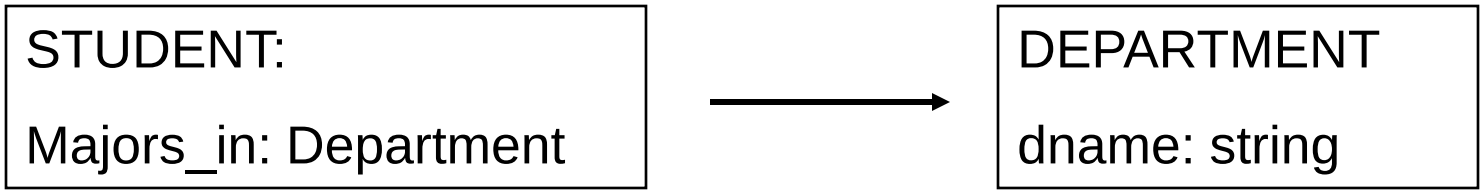
- **O2SQL Language**

Select tuple (fname: s.name.firstname,
lname: s.name.lastname)
from S in Student
where S.majors_in.dname = “Computer Science”

Select firstname, lastname
from Student, majors_in
where majors_in.dname = “Computer Science”
And majors_in.student_id = student.student_id *-- Join Condition*

DOT NOTATION ≈ Join

S:



S.majors_in.dname = “Computer Science”

≈

Nested dot resolution → pointer following

(like network DBS)

Takes place of Join.

Representing Relations in OODBS

Relations are poorly captured in OODBS.

One of their greatest weaknesses

BINARY RELATIONSHIPS

```
Class student inherit person
.......
        majors in: department,
.......
end
```

```
Class department
.......
        majors: set(student),
.......
end
```

Major_in relation captured by “pointers” from one object to another.

Problems with Relation Representation

- **Information duplicated:** (space issue)

  Majors: set(student)
  Duplicates info contained in each *Majors_in:department* listing.

- **Consistency not enforced by DBS**

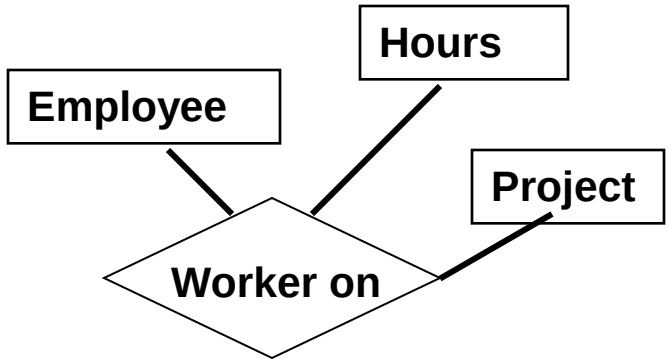
  - Programmer writing change-major method responsible for ensuring that changes to *student: major_in* also reflected/updated in *department: majors: set(student)*

  - This consistency **NOT guaranteed** so different methods of extracting a list of majors may yield different results.

  Jones: *majors_in='Compute science'* but
  Jones not in *'ComputerScience.majors'* set

Problems with Relational Representation (cont)

- **Relationships can't have attributes in pointer representation**

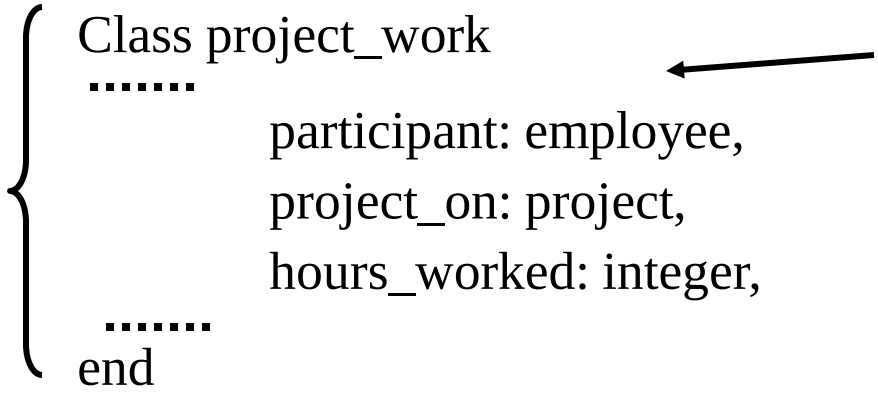


Class employee
.....
works_on: **set(project)**
.....
end

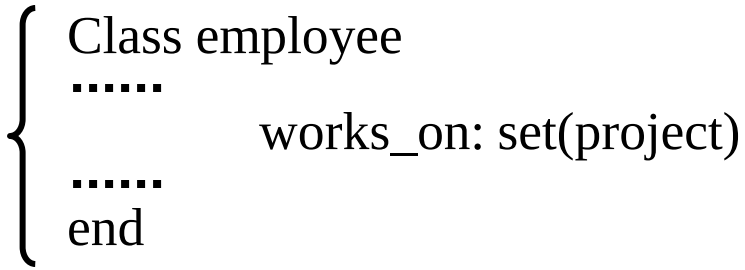
Class project
.....
members: set(employee),
.....
end

How to represent hours worked by each employee on each project is problematic.

Problems with representing relations

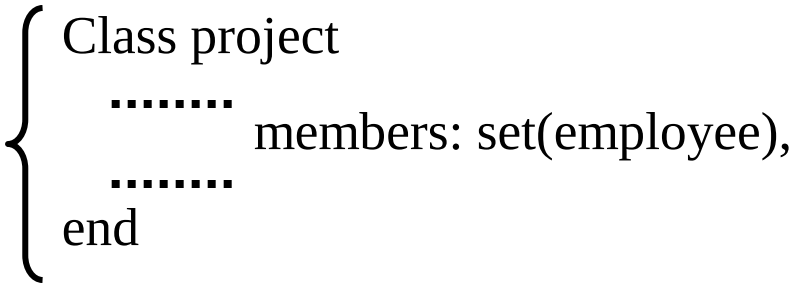


Can simulate a relation on DB relation turned objects, but not in spirit of OODBs.



Each employee/project pairing is its own object.

Probably also set lists in employee and project.



No explicit database support or consistency enforcement -left to programmer.

Advantages / Disadvantages of OODB

Advantages	Disadvantages
•Class inheritance •Encapsulation of attributes/methods •Extensible/flexible definition of complex data types and methods(support for complex objects) •Much greater power given to the programmer to add or change databases semantics.	•**Handling of relationships** ➢Cumbersome ➢Data duplicated ➢Consistency not enforced ▪**Table based representation is often more** ➢Natural ➢Intuitive ➢Efficient ▪**May give too much power to programmer** ▪**Integrity/consistency poorly enforced** ➢More restrictive relational mode semantics makes integrity correctness enforcement easier.