Object-Oriented Databases

based on

Fundamentals of Database Systems
Elmasri and Navathe

Acknowledgement: Fariborz Farahmand

Minor corrections/modifications made by H. Hakimzadeh, 2005

Outline

- Overview of O-O Concepts
- O-O Identity, Object Structure and Type Constructors
- Encapsulation of Operations, Methods and Persistence
- Type Hierarchies and Inheritance
- Complex Objects
- Other O-O Concepts

Introduction

- Reasons for creation of Object Oriented Databases
  - Need for more complex applications
  - Need for additional data modeling features
  - Increased use of object-oriented programming languages

Public Domain and Commercial Systems

- **Research OODB systems:** Orion, Open-OODB at T.I., ODE at ATT Bell labs, Postgres - at UCB, etc.
- **Commercial OODB products:** Ontos, Gemstone, Objectivity, Objectstore, Versant, etc.
- For more see:
  - http://www.service-architecture.com/products/object-oriented_databases.html

The Main Claim

- OO databases try to maintain a direct correspondence between real-world and database objects so that objects do not lose their integrity and identity and can easily be identified and operated upon
- OODBs handle complex data types better
- OODBs handle complex operations better

Objects have two components:

- State (value)
- Behavior (operations)

OO vs. Traditional DBs

- **In OODBs,** objects may have an object structure of arbitrary complexity in order to contain all of the necessary information that describes the object.

- **In contrast, in traditional database systems,** information about a complex object is often scattered over many relations or records, leading to loss of direct correspondence between a real-world object and its database representation.

Instance Variables vs. Attributes

- The internal structure of an object in OOPLs includes the specification of **instance variables,** which hold the values that define the internal state of the object.

- Hence, an instance variable is similar to the concept of an attribute, except that instance variables may be encapsulated within the object and thus are not necessarily visible to external users.

Object-Oriented Concept

- Some OO models insist that all operations a user can apply to an object must be predefined. This forces a complete encapsulation of objects.

- To encourage encapsulation, an operation is defined in two parts:
  - **signature or interface** of the operation, specifies the operation name and arguments (or parameters).
  - **method or body**, specifies the implementation of the operation.

Encapsulation in OODBs

- Operations can be invoked by passing a **message** to an object, which includes the operation name and the parameters.
- The object then executes the method for that operation.
- Encapsulation permits modification of the internal structure of an object, as well as the implementation of its operations, without the need to disturb the external programs that invoke these operations

Versioning…

- Some OO systems provide capabilities for dealing with **multiple versions** of the same object (a feature that is essential in design and engineering applications).

- For example, an old version of an object that represents a tested and verified design should be retained until the new version is tested and verified:
  - very crucial for designs in manufacturing process control, architecture, software systems …

Overloading in OODBs

- Operator polymorphism: It refers to an operation's ability to be applied to different types of objects; This feature is also called operator overloading

Object Identity

- The OODB provides a **unique identity** to each independent object stored in the database.
- This unique identity is typically implemented via a unique system-generated **object identifier, or OID**.
- The main property required of an OID is that it be **immutable;** that is, the OID value of a particular object should not change over the life of the object. This preserves the identity of the real-world object being represented.

Object Structure

- Similar to Object Oriented Programming, OODB objects can be constructed from other object constructors.

- An object can be defined as a triple ***(i,c,v) .*** *Where*
  - *i is the* ***object identifier****,*
  - *c is the* ***type constructor****, and*
  - *v is the state or* ***value*** *of the object.*

Type Constructors

- The three most basic type constructors (types) are **atom, tuple,** and **set.**
- Other commonly used type constructors include **list, bag,** and **array.**

Example of an object

- The atom constructor is used to represent all basic atomic values, such as integers, real numbers, character strings, booleans, and any other basic data types that the system supports directly.

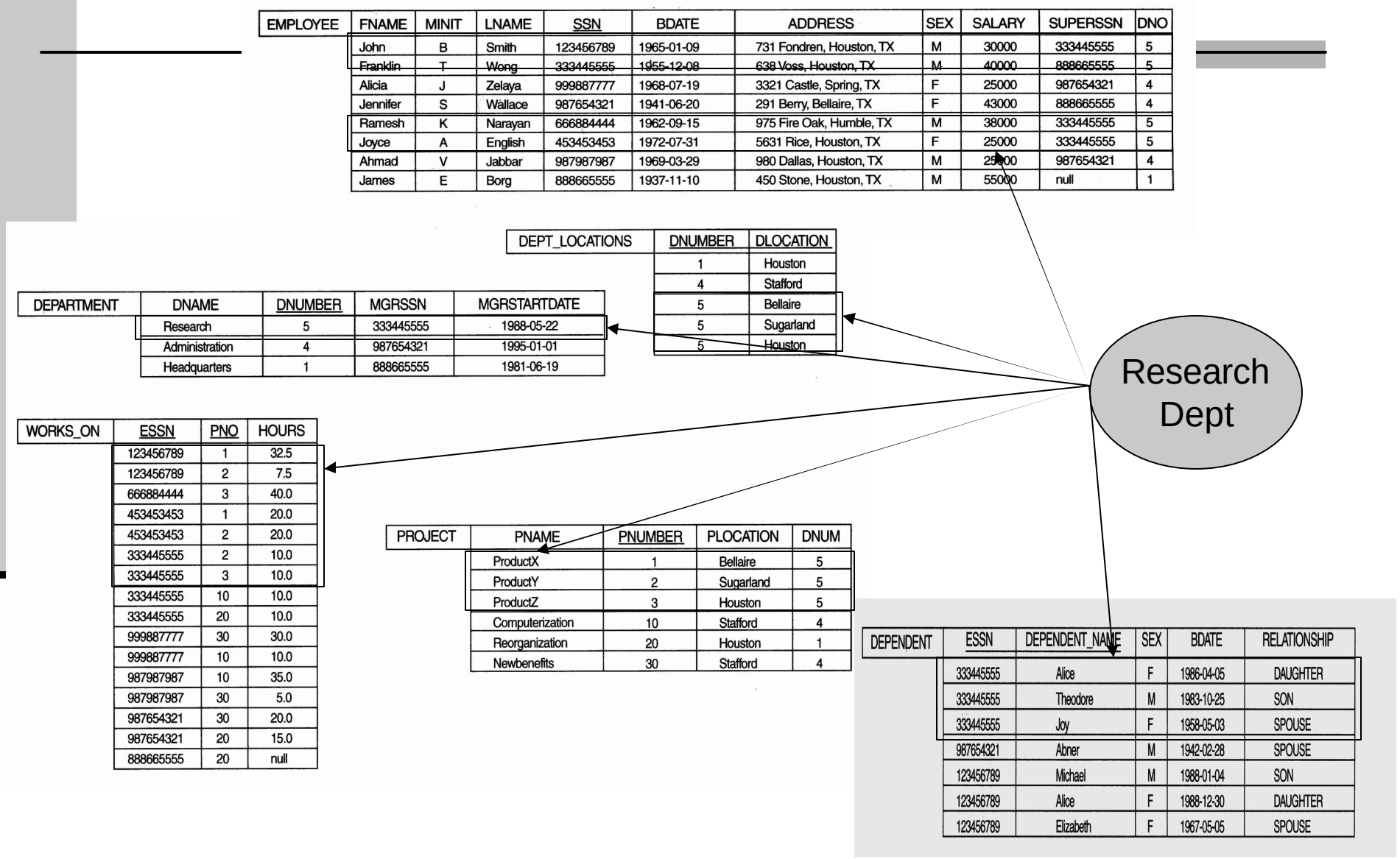
We use i_1, i_2, i_3, . . . to stand for unique system-generated object identifiers.

o_1 = (i_1, atom, 'Houston')

o_2 = (i_4, atom, 5)

o_3 = (i_7, set, {i_1, i_2, i_3})

- Recall the COMPANY schema, visualize the Research department and its components using the relational model.



EMPLOYEE

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-07-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	null	1

DEPT_LOCATIONS

DNUMBER	DLOCATION
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

DEPARTMENT

DNAME	DNUMBER	MGRSSN	MGRSTARTDATE
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

Research Dept

WORKS_ON

ESSN	PNO	HOURS
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	null

PROJECT

PNAME	PNUMBER	PLOCATION	DNUM
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

ESSN	DEPENDENT_NAME	SEX	BDATE	RELATIONSHIP
333445555	Alice	F	1986-04-05	DAUGHTER
333445555	Theodore	M	1983-10-25	SON
333445555	Joy	F	1958-05-03	SPOUSE
987654321	Abner	M	1942-02-28	SPOUSE
123456789	Michael	M	1988-01-04	SON
123456789	Alice	F	1988-12-30	DAUGHTER
123456789	Elizabeth	F	1967-05-05	SPOUSE

Example 1:

EMPLOYEE	FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
	John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
	Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
	Alicia	J	Zelaya	999887777	1968-07-19	3321 Castle, Spring, TX	F	25000	987654321	4
	Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
	Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
	Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
	Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
	James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	null	1

Example 1 (cont.):

DEPT_LOCATIONS	DNUMBER	DLOCATION
	1	Houston
	4	Stafford
	5	Bellaire
	5	Sugarland
	5	Houston

DEPARTMENT	DNAME	DNUMBER	MGRSSN	MGRSTARTDATE
	Research	5	333445555	1988-05-22
	Administration	4	987654321	1995-01-01
	Headquarters	1	888665555	1981-06-19

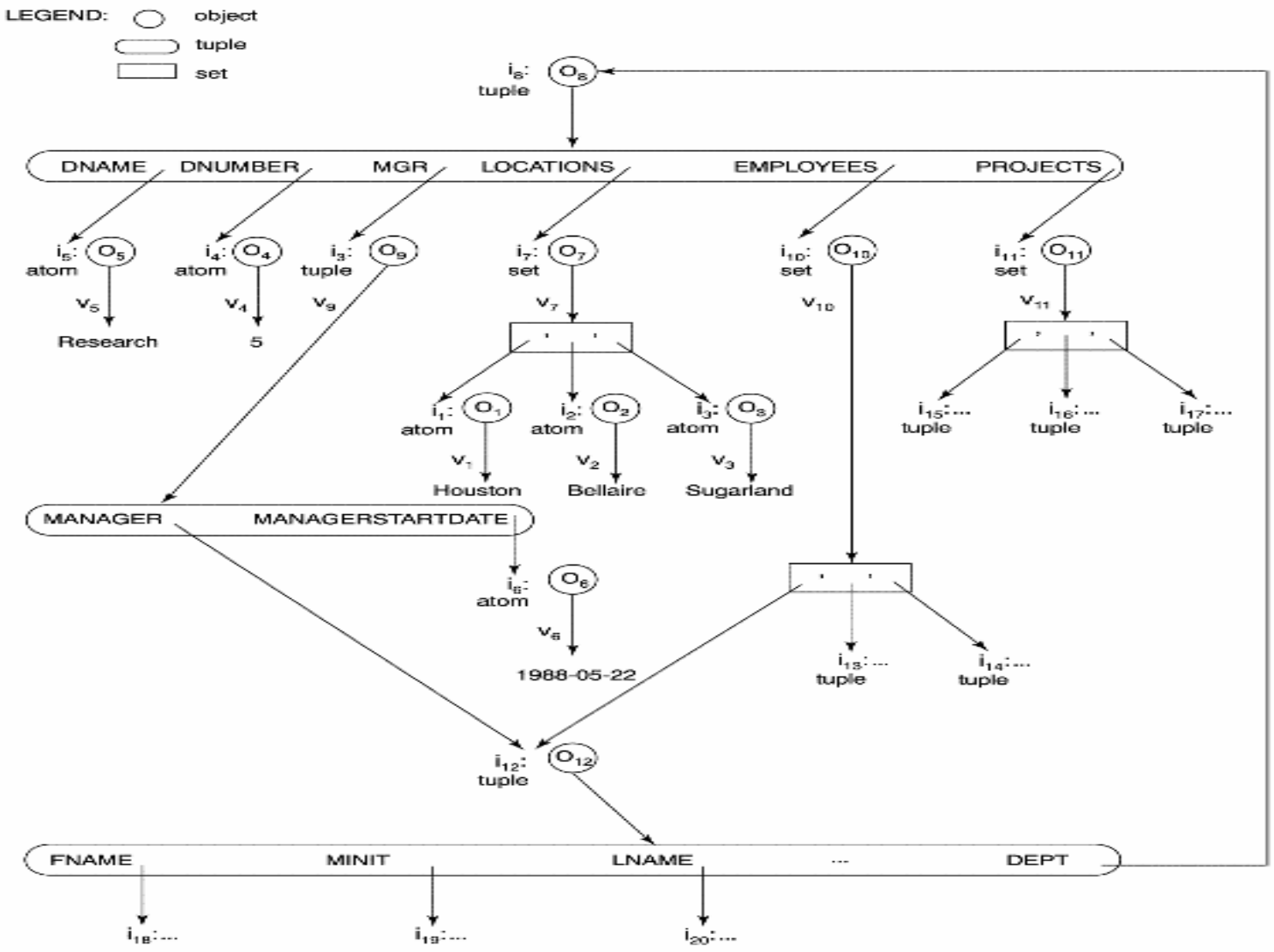
WORKS_ON	ESSN	PNO	HOURS
	123456789	1	32.5
	123456789	2	7.5
	666884444	3	40.0
	453453453	1	20.0
	453453453	2	20.0
	333445555	2	10.0
	333445555	3	10.0
	333445555	10	10.0
	333445555	20	10.0
	999887777	30	30.0
	999887777	10	10.0
	987987987	10	35.0
	987987987	30	5.0
	987654321	30	20.0
	987654321	20	15.0
	888665555	20	null

PROJECT	PNAME	PNUMBER	PLOCATION	DNUM
	ProductX	1	Bellaire	5
	ProductY	2	Sugarland	5
	ProductZ	3	Houston	5
	Computerization	10	Stafford	4
	Reorganization	20	Houston	1
	Newbenefits	30	Stafford	4

Example 1 (cont.)

DEPENDENT	ESSN	DEPENDENT_NAME	SEX	BDATE	RELATIONSHIP
	333445555	Alice	F	1986-04-05	DAUGHTER
	333445555	Theodore	M	1983-10-25	SON
	333445555	Joy	F	1958-05-03	SPOUSE
	987654321	Abner	M	1942-02-28	SPOUSE
	123456789	Michael	M	1988-01-04	SON
	123456789	Alice	F	1988-12-30	DAUGHTER
	123456789	Elizabeth	F	1967-05-05	SPOUSE

Now visualize the Research Department in Object Oriented Model



Object Identity, Object Structure, and Type Constructors

- **Example 1**

We use i_1, i_2, i_3, . . . to stand for unique system-generated object identifiers. Consider the following objects:

o_1 = (i_1, atom, ‘Houston’)
o_2 = (i_2, atom, ‘Bellaire’)
o_3 = (i_3, atom, ‘Sugarland’)
o_4 = (i_4, atom, 5)
o_5 = (i_5, atom, ‘Research’)
o_6 = (i_6, atom, ‘1988-05-22’)
o_7 = (i_7, set, {i_1, i_2, i_3})

Object Identity, Object Structure

Example 1 (cont.)

- The first six objects listed in this example represent atomic values.

- Object 7 is a set-valued object that represents the set of locations for department 5; the set refers to the atomic objects with values {‘Houston’, ‘Bellaire’, ‘Sugarland’}.

Object Identity

- **Example 1 (cont.)**

  o_8 = (i_8, tuple, <dname:i_5, dnumber:i_4, mgr:i_9, locations:i_7, employees:i_{10}, projects:i_{11}>)

  o_9 = (i_9, tuple, <manager:i_{12}, manager_start_date:i_6>)

  o_{10} = (i_{10}, set, {i_{12}, i_{13}, i_{14}})

  o_{11} = (i_{11}, set {i_{15}, i_{16}, i_{17}})

  o_{12} = (i_{12}, tuple, <fname:i_{18}, minit:i_{19}, lname:i_{20}, ssn:i_{21}, . . ., salary:i_{26}, supervi-sor:i_{27}, dept:i_8>)

  . . .

Object Identity, Object Structure

Example 1 (cont.)

- Object 8 is a tuple-valued object that represents department 5 itself, and has the attributes DNAME, DNUMBER, MGR, LOCATIONS, and so on.

Equal vs. Identical Objects

- **Two objects are said to have identical values**, if the graphs representing their values are identical in every respect, including the OIDs at every level.

- **Two object are said to have equal values** if their graph structure is the same, all the corresponding atomic values should also be the same (same states at the atomic level) but their OID do not have to be the same.

- Equal is a weaker notion.

Equal vs. Identical Objects

Example 2:

This example illustrates the difference between the two definitions for comparing object states for equality.

$o_1 = (i_1, \text{tuple}, \langle a_1{:}i_4, a_2{:}i_6 \rangle)$

$o_2 = (i_2, \text{tuple}, \langle a_1{:}i_5, a_2{:}i_6 \rangle)$

$o_3 = (i_3, \text{tuple}, \langle a_1{:}i_4, a_2{:}i_6 \rangle)$

$o_4 = (i_4, \text{atom}, 10)$

$o_5 = (i_5, \text{atom}, 10)$

$o_6 = (i_6, \text{atom}, 20)$

Equal Object States

Example 2 (cont.):

- In this example, The objects o_1 and o_2 have ***equal*** states, since their states at the atomic level are the same but the values are reached through distinct objects o_4 and o_5.

Identical Object States

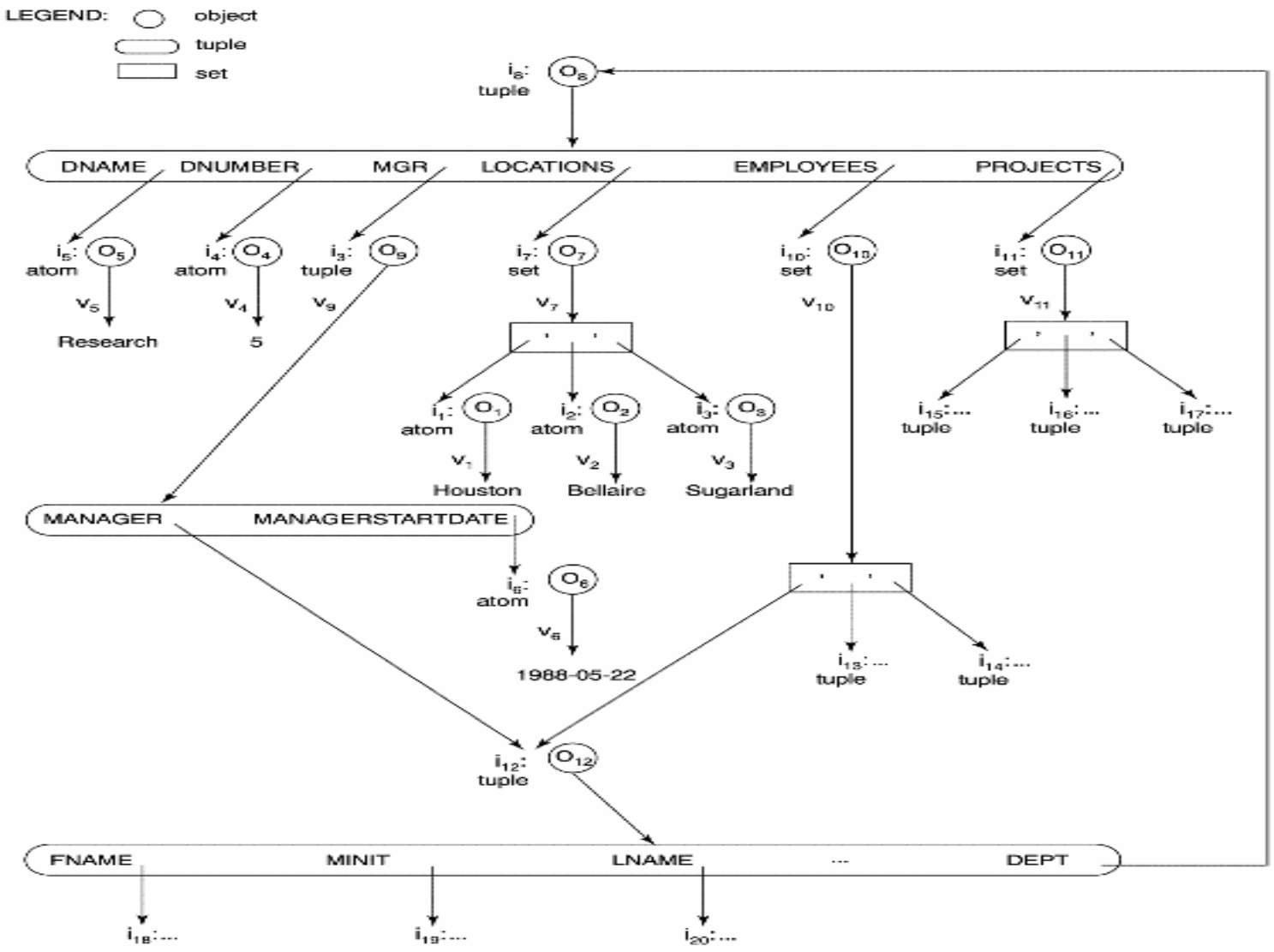
Example 2 (cont.):

- However, the states of objects o_1 and o_3 are ***identical,*** even though the objects themselves are not because they have distinct OIDs.

- Similarly, although the states of o_4 and o_5 are identical, the actual objects o_4 and o_5 are equal but not identical, because they have distinct OIDs.

Graphical Representation of a Complex Object

Research Department



Type Constructors

- **Specifying the object types Employee, date, and Department using type constructors**

```
define type Employee:
    tuple (     fname:          string;
                minit:          char;
                lname:          string;
                ssn:            string;
                birthdate:      Date;
                address:        string;
                sex:            char;
                salary:         float;
                supervisor:     Employee;
                dept:           Department;         );
define type Date
    tuple (     year:           integer;
                month:          integer;
                day:            integer;    );
define type Department
    tuple (     dname:          string;
                dnumber:        integer;
                mgr:            tuple (     manager:    Employee;
                                            startdate:  Date;       );
                locations:      set(string);
                employees:      set(Employee);
                projects        set(Project);   );
```

Encapsulation of Operations, Methods, and Persistence

Encapsulation

- One of the main characteristics of OO languages and systems
- related to the concepts of *abstract data types* and *information hiding* in programming languages

Encapsulation of Operations, Methods, and Persistence (cont.)

Specifying Object Behavior via Class Operations:

- The main idea is to define the **behavior** of a type of object based on the **operations** that can be externally applied to objects of that type.

Encapsulation of Operations, Methods, and Persistence (cont.)

Specifying Object Behavior via Class Operations (cont.):

- For database applications, the requirement that all objects be completely encapsulated is too stringent.
- One way of relaxing this requirement is to divide the structure of an object into visible and hidden attributes (instance variables).

Encapsulation of Operations, Methods, and Persistence (cont.)

Adding operations to definitions of Employee and Department:

```
define class DepartmentSet:
    type          set(Department);
    operations    add_dept(d: Department): boolean;
            (* adds a department to the DepartmentSet object *)
                  remove_dept(d: Department): boolean;
            (* removes a department from the DepartmentSet object *)
                  create_dept_set:        DepartmentSet;
                  destroy_dept_set:       boolean;
end DepartmentSet;

    ...

persistent name AllDepartments: DepartmentSet;
(* AllDepartments is a persistent named object of type DepartmentSet *)

    ...

d:= create_dept;
(* create a new Department object in the variable d *)

    ...

b:= AllDepartments.add_dept(d);
(* make d persistent by adding it to the persistent set AllDepartments *)

    ...
```

11.4 Type Hierarchies and Inheritance

- Type (class) Hierarchy

- A type in its simplest form can be defined by it's type name and a listing of the names of its visible/*public* functions.

  TYPE_NAME: function, function, . . . , function

  Example:

  PERSON: Name, Address, Birthdate, Age, SSN

Type Hierarchies and Inheritance (cont.)

- **Subtype:** when the designer or user must create a new type that is similar but not identical to an already defined type

- **Subtype:** inherits all the functions of the **predefined type (also known as the supertype)**

Type Hierarchies and Inheritance (cont.)

Example (1):

EMPLOYEE: Name, Address, Birthdate, Age, SSN, Salary, HireDate, Seniority

STUDENT: Name, Address, Birthdate, Age, SSN, Major, GPA

OR:

EMPLOYEE **subtype-of** PERSON: Salary, HireDate, Seniority

STUDENT **subtype-of** PERSON: Major, GPA

Type Hierarchies and Inheritance (cont.)

Example (2):

consider a type that describes objects in plane geometry, which may be defined as follows:

GEOMETRY_OBJECT type:

Shape, Area, ReferencePoint

Type Hierarchies and Inheritance (cont.)

- **Example (2) (cont.):**

  Now suppose that we want to define a number of subtypes for the GEOMETRY_OBJECT type, as follows:

  RECTANGLE **subtype-of** GEOMETRY_OBJECT: Width, Height

  TRIANGLE **subtype-of** GEOMETRY_OBJECT: Side1, Side2, Angle

  CIRCLE **subtype-of** GEOMETRY_OBJECT: Radius

Type Hierarchies and Inheritance (cont.)

- **Example (2) (cont.):**

  An alternative way of declaring these three subtypes is to specify the value of the Shape attribute as a condition that must be satisfied for objects of each subtype:

  RECTANGLE **subtype-of** GEOMETRY_OBJECT
  (Shape='rectangle'): Width, Height

  TRIANGLE **subtype-of** GEOMETRY_OBJECT
  (Shape='triangle'): Side1, Side2, Angle

  CIRCLE **subtype-of** GEOMETRY_OBJECT
  (Shape='circle'): Radius

Type Hierarchies and Inheritance (cont.)

- **Extents**: collections of objects of the same type.

Type Hierarchies and Inheritance (cont.)

- **Persistent Collection**: It holds a collection of objects that is stored permanently in the database and hence can be accessed and shared by multiple programs

Type Hierarchies and Inheritance (cont.)

- **Transient Collection**: It exists temporarily during the execution of a program but is not kept when the program terminates

11.5 Complex Objects

- **Unstructured complex object:**
  - Provided by a DBMS
  - Permits the storage and retrieval of large objects that are needed by the database application.
  - Examples: of such objects are ***bitmap images*** and ***long text strings*** (such as documents); they are also known as **binary large objects,** or **BLOBs** for short.

Complex Objects (cont.)

- **Structured complex object:**
  - Differs from an unstructured complex object in that the object's structure is defined by repeated application of the type constructors provided by the OODBMS. Hence, the object structure is defined and known to the OODBMS.

Other Objected-Oriented Concepts

- **Polymorphism (Operator Overloading):**

  This concept allows the same *operator name* or *symbol* to be bound to two or more different *implementations* of the operator, depending on the type of objects to which the operator is applied

11.6 Other Objected-Oriented Concepts (cont.)

- **Multiple Inheritance and Selective Inheritance**
  - Multiple inheritance in a type hierarchy occurs when a certain subtype T is a subtype of two (or more) types and hence inherits the functions (attributes and methods) of both supertypes.
  - Example, we may create a subtype ENGINEERING_MANAGER that is a subtype of both MANAGER and ENGINEER. This leads to the creation of a type lattice rather than a type hierarchy.

Other Objected-Oriented Concepts (cont.)

Versions and Configurations

- Many database applications that use OO systems require the existence of several versions of the same object

- There may be more than two versions of an object.

Other Objected-Oriented Concepts (cont.)

- **Configuration**: A configuration of the complex object is a collection consisting of one version of each module arranged in such a way that the module versions in the configuration are compatible and together form a valid version of the complex object.

Summary

- ***Object identity:*** Objects have unique identities that are independent of their attribute values.
- ***Type constructors****:* Complex object structures can be constructed by recursively applying a set of basic constructors, such as tuple, set, list, and bag.
- ***Encapsulation of operations****:* Both the object structure and the operations that can be applied to objects are included in the object class definitions.

Summary (cont.)

- ***Programming language compatibility****:* Both persistent and transient objects are handled uniformly. Objects are made persistent by being attached to a persistent collection.

- ***Type hierarchies and inheritance****:* Object types can be specified by using a type hierarchy, which allows the inheritance of both attributes and methods of previously defined types.

Summary (cont.)

- ***Extents****:* All persistent objects of a particular type can be stored in an extent. Extents corresponding to a type hierarchy have set/subset constraints enforced on them.

- ***Support for complex objects****:* Both structured and unstructured complex objects can be stored and manipulated.

- ***Polymorphism and operator overloading****:* Operations and method names can be overloaded to apply to different object types with different implementations.

Summary (cont.)

- ***Versioning****:* Some OO systems provide support for maintaining several versions of the same object.

Current Status

- OODB market not growing much per se
- O-O ideas are being used in a large number of applications, *without* explicitly using the OODB platform to store data.
- Growth: O-O tools for modeling and analysis, O-O Programming Languages like Java and C++
- Compromise Solution Proposed: Object Relational DB Management (Informix Universal Server, Oracle 9i, IBM's UDB, …)