Introduction

- Outline
  - Object-identifier, object structure
  - Encapsulation
  - Type and class hierarchy
  - Structured and unstructured objects
  - Polymorphism and operator overloading
  - Multiple inheritance

Introduction

- OO has roots in Programming languages
- SIMULA (simulation language) has the concept of classes - late 60s
- SMALL TALK is the first pure OO language
- Hybrid languages incorporate OO concepts into an already exisiting programming language
  - Example: C++

Introduction

- Programming objects only exist during the program execution
- Database objects need to exist permanently (persistent objects)
- Concepts such as encapsulation, inheritance, identity and evolving relationships must be applied in the context of OODBMS
- OODBMS must also support transactions, concurrency, recovery

Object Identity

- Each object in the DB has a unique identity (OID)
  - even though the value of an object changes, its identity must not change (the OID is immutable)
  - if an object is deleted its OID must not be assigned to any other object
- These two properties imply that an object identifier must not depend on an attribute
- Some systems use the physical address of the object in storage as an OID
- Relational database tables have a primary key
  - value can change
  - could have same *object* - two tables - different primary keys

Object Structure

- In OODB, the value of a complex object can be constructed from other objects
- Each object can be viewed as a **triple**
  - (i, c, v)
  - where i is the unique object identifier (**OID**)
  - c is the constructor or an indication of how the object value is constructed (operator)
  - v is the value of the object (state)
- Basic constructors are atom, tuple, set
  - Others: list, array

Object Structure

- The value v can be interpreted on the basis of the constructor c
- Example:
  - if c = atom then v = atomic value

    $o_1 = (i_1, \text{atom}, \text{Houston})$

    $o_2 = (i_2, \text{atom}, \text{Bellaire})$

    $o_3 = (i_3, \text{atom}, \text{Sugarland})$

    $o_4 = (i_4, \text{atom}, 5)$

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

    $o_6 = (i_6, \text{atom}, \text{22-May-78})$

The value 'Houston'

Object Structure

- Example:
  - if $c = \text{tuple}$ then $v = < a_1{:}i_1 .. a_n{:}i_n >$

A department tuple

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

$o_9 = (i_9, \text{tuple}, < \text{manager}{:}i_{12}, \text{managerstartdate}{:}\ i_6>)$

Object Structure

- Example:
  - if c = set then v = $\{i_1, i_2, i_3\}$

    $o_7 = (i_7, set, \{ i_1, i_2, i_3 \})$

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

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

A set of employees

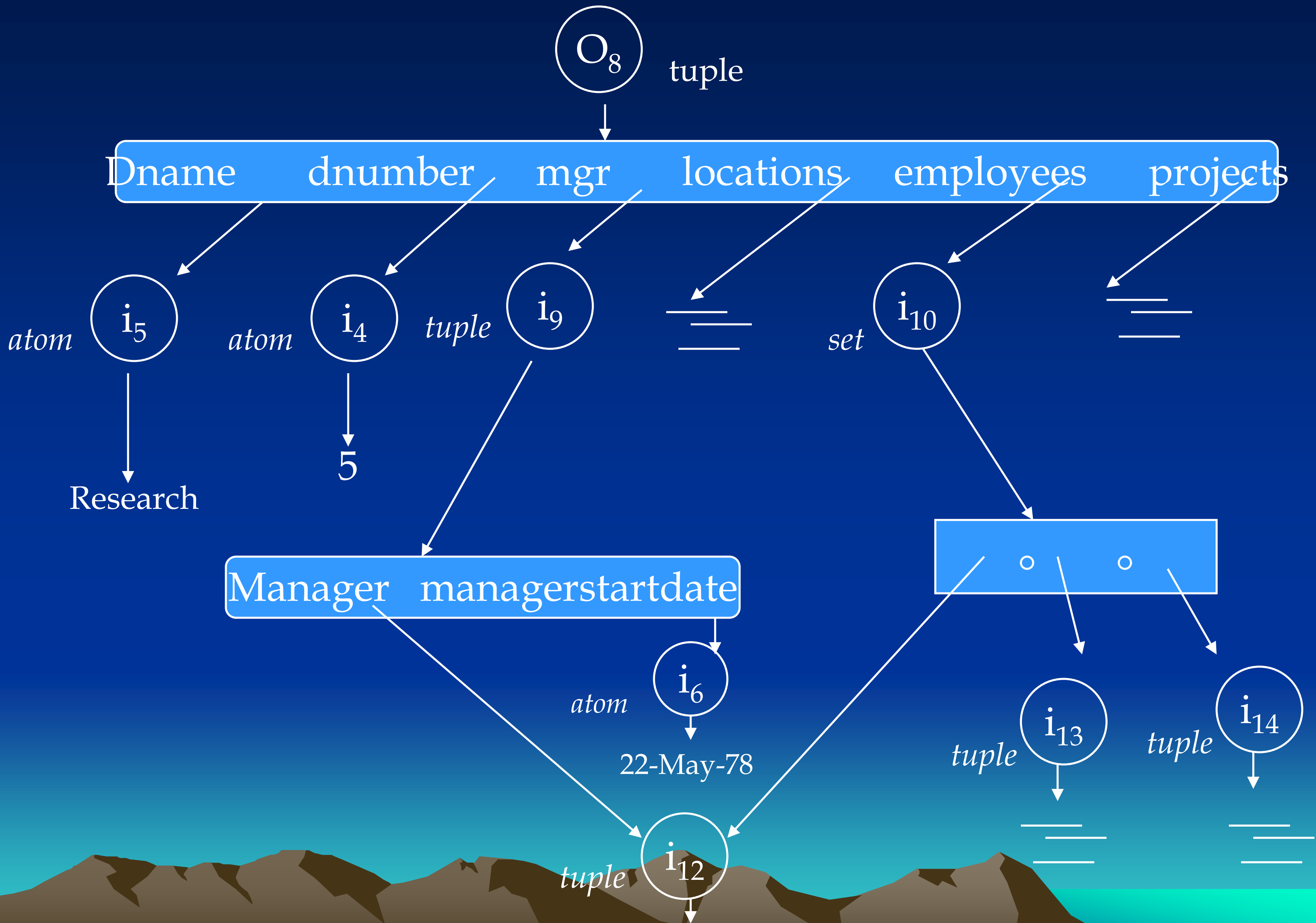
Type Constructors

```
Define type employee
tuple(
        name            string
        ssn             string
        birthdate       date
        work_in         department   )

Define type department
tuple(
        dname           string
        dnumber                     integer
        mgr             tuple (     manager          employee
                                    startdate:       date)
        locations       set (string)
        employees       set (employee )
        projects        set (project ) )
```

We will have references to other objects

Type Constructors



Encapsulation

- Encapsulation - Structure of an object is not visible to the external world
  - all operations on an object are predefined
  - some operations may be used to create, destroy, modify the values or retrieve the values of an object
  - External users only have access to the **interface** of the object (signature) which defines the names and types of all parameters to each operation
  - **Methods** specify the implementation of operations
  - A method is invoked (call) by sending a **message**

Encapsulation

- In a RDBMS, the structure of an object is visible to all users
  - That is, a relation and attributes are visible
  - All database operations (selection, insertion, deletion..) are applicable to any relation (all object types)
- In an OODBMS, one can divide a structure into visible and hidden parts
  - The hidden attributes are completely encapsulated and accessed only through pre-defined operations

Encapsulation/Persistence

- It is customary for an OODBMS to be closely coupled with an OO programming language
- An OO programming language is used to specify the method implementations
  - O2 uses O2C … O2C is C adapted for objects
  - ObjectStore uses C++

- In an OODBMS, not all objects are *persistent*; some are *transient*
- In an EER or relational model, all objects are persistent

Type and Class Hierarchies

- Types are different from classes even though they lead to the same structures
- A type has a name, a set of attributes (instance variables) and operations (methods)
- A new type can be defined based on other predefined types leading to a Type hierarchy
- An object can belong to a type
- Type definitions do not generate objects of their own
  - Example: Person: Name, Address, Birthdate, Age
  - Employee *subtype of* Person: Salary, HireDate

Class Hierarchies

- Class is a collection of objects meaningful to some application
- In most OODBs, a class is a collection of objects belonging to the same type
- A class is defined by its name and the collection of objects included in the class
- We can also define subclasses and superclasses creating a class hierarchy
- In OODBs the concept of a type and class are the same. Hence, the hierarchies are the same.
- Each class then has a particular type and holds a collection of persistent objects of that type

Complex Objects

- Motivation for the development of OO systems is to represent complex objects
- Two types of complex objects:
  - Unstructured
  - Structured

Complex Objects

- Unstructured Complex Object
  - The structure of these objects is not known to the DBMS
  - Only the application programs can interpret the objects
  - Ex: Bitmap images - BLOB (Binary large objects)
  - These objects require a large amount of storage and not a part of the standard type definitions

Complex Objects

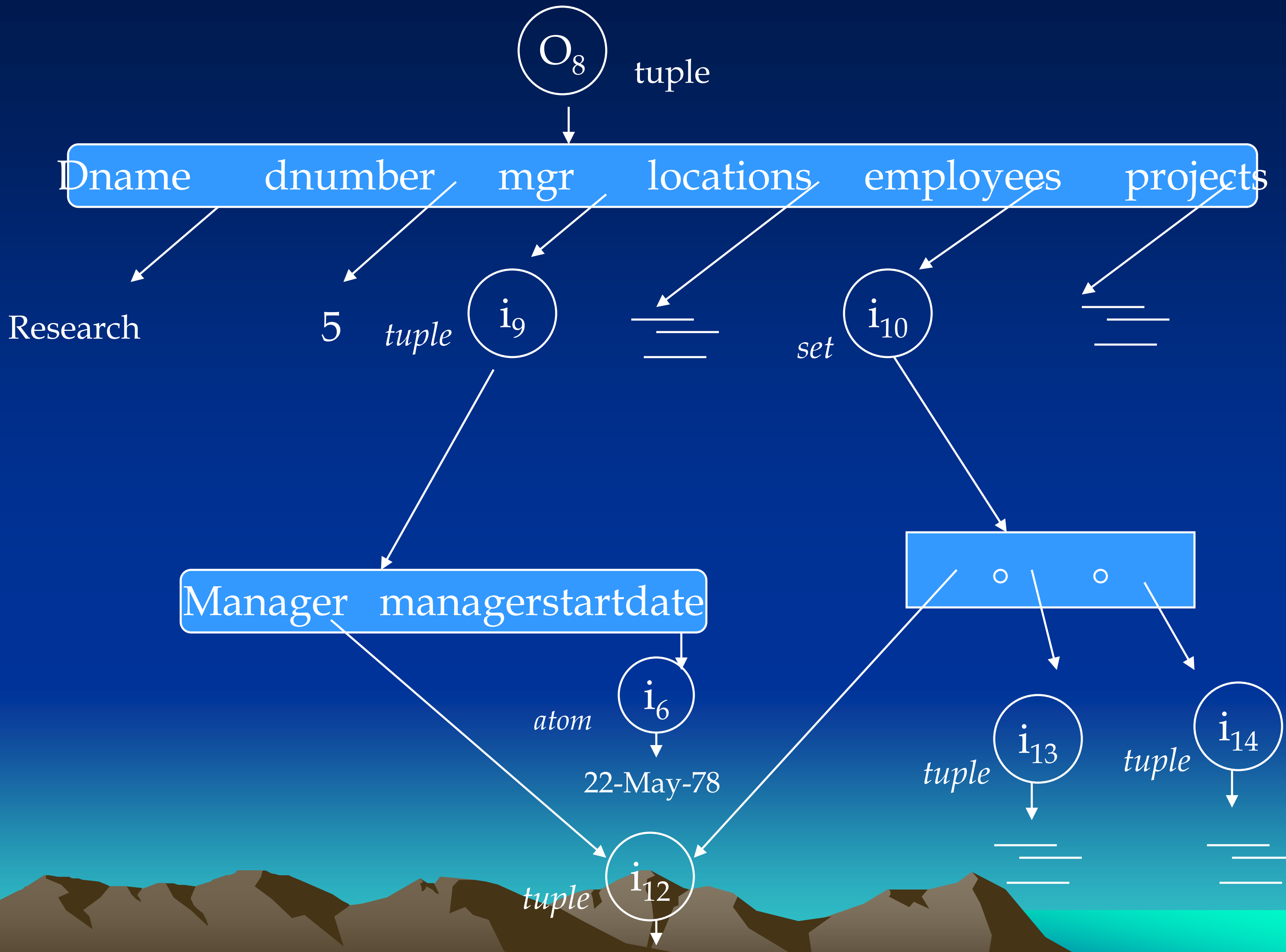
- ☞ Unstructured Complex Object
  - DBMS may retrieve only a portion of the object
  - DBMS may use caching and buffering to prefetch portions of the object
  - The DBMS does not have the capability to directly process selection conditions based on values of these objects unless the application programs provide the code
  - In OODBMS, this is done by defining an *Abstract* data type with operations for selection, comparison, etc
  - These feature allows the OODBM to have an *extensible* type system
    - That is, new types can be created and hence libraries of new types

Complex Objects

- Structured Complex Object
  - The object structure is defined and known to the DBMS
  - Object Structure is defined using type constructors (set, atom, tuple)
  - Two types of reference semantics exist between a complex object and its component:
    - Ownership
    - Reference

Complex Objects

- Ownership Semantics
  - Subobjects are encapsulated within a complex object are considered a part of the complex object
- Reference Semantics
  - Components of a complex object are themselves independent objects, but at the same time may be considered a part of the complex object



O8
tuple
Dname dnumber mgr locations employees projects
Research
5
tuple
i9
set
i10
Manager managerstartdate
atom
i6
22-May-78
tuple
i13
tuple
i14
tuple
i12

Complex Objects

- The ownership semantics leads to an *is-part-of* or *is-component-of* relationship
  - are employees part of the department?

- The is-part-of relationship (or ownership semantics) means that the encapsulated objects *can be accessed by the methods of that object* and deleted if the object is deleted

Complex Objects

- The reference semantics leads to an *is-associated-with* relationship
  - are the employees associated with the department?

- An OODBMS should provide the storage options for clustering the component objects together in order to increase efficiency
- The mechanism of building objects from complex object structures is called object assembly

Other OO concepts

- ☞ Polymorphism
  - An operator can be applied to different types of objects
  - When an operator has distinct implementations then we have operator **overloading**

  - Example: + when applied to integers implies integer addition
  - + when applied to sets implies set union

Other OO concepts

- Polymorphism
- Example:

  Geometry_Object: Shape, Area, CenterPoint

  - 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

  Area is a method that would be different for each sub-Type

Other OO concepts

☞ Strongly Typed systems:

Method selection is done at compile time (early binding)

☞ Weakly Typed systems:

Method selection is done at run time (late binding).

Lisp and Small Talk are examples late-binding

Other OO concepts

- Multiple Inheritance:
  - allowed in O2
  - leads to a lattice
  - One problem: if a subtype inherits two distinct methods with the same name from two different supertypes
  - A solution: check for ambiguity when the subtype is created and let the user choose the function
  - Another solution: use some system default
  - A third solution: disallow multiple inheritance if ambiguity occurs

Other OO concepts

- Selective Inheritance:
  - When a subtype inherits only a few methods
  - This mechanism is not usually provided by OODBMS

Other OO concepts

- Versions:
  - Ability to maintain several versions of an object
  - Commonly found in many software engineering and concurrent engineering environments
  - Merging and reconciliation of various versions is left to the application program
  - Some systems maintain a version graph
- Configuration:
  - A configuration is a collection compatible versions of modules of a software system (a version per module)