SGG 4653
Advance Database System

Introduction and Preliminaries of: Object-oriented DBMS

Introduction to Object DBMS

Outline

- Advanced database applications.
- Unsuitability of RDBMS for advanced database applications.
- Object-oriented concepts.
- Problems of storing objects in relational database.
- The next generation of database systems.

Advanced Database Applications

- Computer-Aided Design (CAD).
- Computer-Aided Manufacturing (CAM).
- Computer-Aided Software Engineering (CASE).
- Network Management Systems.
- Office Information Systems (OIS) and Multimedia Systems.
- Digital Publishing.
- Geographic Information Systems (GIS).
- Interactive and Dynamic Web sites.
- Other applications with complex and interrelated objects and procedural data.

Computer-Aided Design (CAD)

- Stores data relating to mechanical and electrical design, for example, buildings, airplanes, and integrated circuit chips.
- Designs of this type have some common characteristics:
  - Data has many types, each with a small number of instances.
  - Designs may be very large.
  - Design is not static but evolves through time.
  - Updates are far-reaching.
  - Involves version control and configuration management.
  - Cooperative engineering.

Advanced Database Applications

- Computer-Aided Manufacturing (CAM)
  - Stores similar data to CAD, plus data about discrete production.
- Computer-Aided Software Engineering (CASE)
  - Stores data about stages of software development lifecycle.

Network Management Systems

- Coordinate delivery of communication services across a computer network.
- Perform such tasks as network path management, problem management, and network planning.
- Systems handle complex data and require real-time performance and continuous operation.
- To route connections, diagnose problems, and balance loadings, systems have to be able to move through this complex graph in real-time.

Office Information Systems (OIS) and Multimedia Systems

- Stores data relating to computer control of information in a business, including electronic mail, documents, invoices, and so on.
- Modern systems now handle free-form text, photographs, diagrams, audio and video sequences.
- Documents may have specific structure, perhaps described using mark-up language such as SGML, HTML, or XML.

Digital Publishing

- Becoming possible to store books, journals, papers, and articles electronically and deliver them over high-speed networks to consumers.
- As with OIS, digital publishing is being extended to handle multimedia documents consisting of text, audio, image, and video data and animation.
- Amount of information available to be put online is in the order of petabytes (10^{15} bytes), making them largest databases DBMS has ever had to manage.

Geographic Information Systems (GIS)

- GIS database stores spatial and temporal information, such as that used in land management and underwater exploration.
- Much of data is derived from survey and satellite photographs, and tends to be very large.
- Searches may involve identifying features based, for example, on shape, color, or texture, using advanced pattern-recognition techniques.

Interactive and Dynamic Web Sites

- Consider web site with online catalog for selling clothes. Web site maintains a set of preferences for previous visitors to the site and allows a visitor to:
  - obtain 3D rendering of any item based on color, size, fabric, etc.;
  - modify rendering to account for movement, illumination, backdrop, occasion, etc.;
  - select accessories to go with the outfit, from items presented in a sidebar;
- Need to handle multimedia content and to interactively modify display based on user preferences and user selections. Also have added complexity of providing 3D rendering.

Weaknesses of RDBMS

- Poor Representation of “Real World” Entities
  - Normalization leads to relations that do not correspond to entities in “real world”.
- Semantic Overloading
  - Relational model has only one construct for representing data and data relationships: the relation.
  - Relational model is *semantically overloaded*.

Weaknesses of RDBMS

- Poor Support for Integrity and Enterprise Constraints
- Homogeneous Data Structure
  - Relational model assumes both horizontal and vertical homogeneity.
  - Many RDBMS now allow *Binary Large Objects* (*BLOBs*).

Weaknesses of RDBMS

- Limited Operations
  - RDBMS only have a fixed set of operations which cannot be extended.

- Difficulty Handling Recursive Queries
  - Extremely difficult to produce recursive queries.
  - Extension proposed to relational algebra to handle this type of query is unary transitive (recursive) closure operation.
    - Relation augmented with all tuples successfully deduced by transitivity, e.g. : if (a,b) and (b,c) are tuples of R, than tuples (a,c) is also added to result.

Example - Recursive Query

staffNo	managerstaffNo
S005	S004
S004	S003
S003	S002
S002	S001
S001	NULL

(a) Staff

staffNo	managerstaffNo
S005	S004
S004	S003
S003	S002
S002	S001
S001	NULL
S005	S003
S005	S002
S005	S001
S004	S002
S004	S001
S003	S001

(b) Transitive Closure of Staff

Weaknesses of RDBMS

- Impedance Mismatch
  - Most Data Manipulation Language (DML) lack of *computational completeness*.
  - To overcome this, SQL can be embedded in a high-level 3GL.
  - This produces an impedance mismatch - mixing different programming paradigms.
  - Estimated that as much as 30% of programming effort and code space is expended on this type of conversion.

Weaknesses of RDBMS

- Other Problems with RDBMS
  - Transactions are generally short-lived and concurrency control protocols not suited for long-lived transactions.
  - Schema changes are difficult.
  - RDBMS are poor at navigational access.

Object-Oriented Concepts

- Abstraction, encapsulation, information hiding.
- Objects and attributes.
- Object identity.
- Methods and messages.
- Classes, subclasses, super classes, and inheritance.
- Overloading.
- Polymorphism and dynamic binding.

Abstraction

- Process of identifying essential aspects of an entity and ignoring unimportant properties.
- Concentrate on what an object is and what it does, before deciding how to implement it.

Encapsulation and Information Hiding

- Encapsulation
  - Object contains both data structure and set of operations used to manipulate it.
- Information Hiding
  - Separate external aspects of an object from its internal details, which are hidden from outside.
- Allows internal details of an object to be changed without affecting applications that use it, provided external details remain same.
- Provides data independence.

Object

Uniquely identifiable entity that contains both the attributes that describe the state of a real-world object and the actions associated with it.

- Definition very similar to definition of an entity, however, object encapsulates both state and behavior; an entity only models state.

Attributes

Contain current state of an object.

- Attributes can be classified as simple or complex.
- Simple attribute can be a primitive type such as integer, string, etc., which takes on literal values.
- Complex attribute can contain collections and/or references.
- Reference attribute represents relationship.
- An object that contains one or more complex attributes is called a complex object.

Object Identity

Object identifier (OID) assigned to object when it is created that is:

- System-generated.
- Unique to that object.
- Invariant.
- Independent of the values of its attributes (that is, its state).
- Invisible to the user (ideally).

Object Identity - Implementation

- In RDBMS, object identity is value-based: primary key is used to provide uniqueness.
- Primary keys do not provide type of object identity required in OO systems:
  - key only unique within a relation, not across entire system;
  - key generally chosen from attributes of relation, making it dependent on object state.

Object Identity - Implementation

- Programming languages use variable names and pointers/virtual memory addresses, which also compromise object identity.
- In C/C++, OID is physical address in process memory space, which is too small - scalability requires that OIDs be valid across storage volumes, possibly across different computers.
- Further, when object is deleted, memory is reused, which may cause problems.

Advantages of OIDs

- They are efficient.
- They are fast.
- They cannot be modified by the user.
- They are independent of content.

Methods and Messages

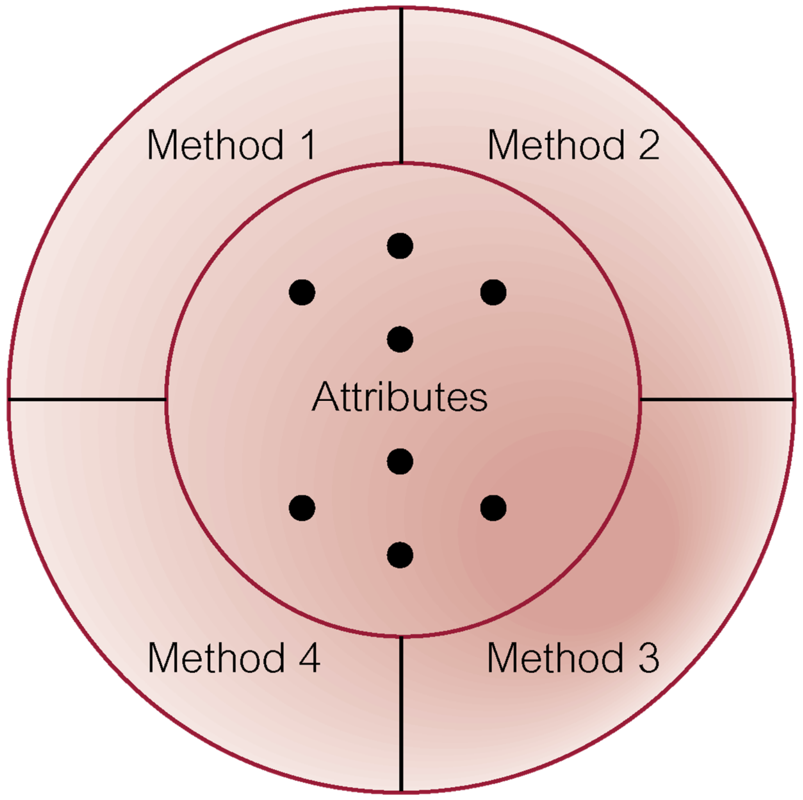
Method

- Defines behavior of an object, as a set of encapsulated functions.

Message

- Request from one object to another asking second object to execute one of its methods.

Object Showing Attributes and Methods



Example of a Method

```
**method** void updateSalary(float increment)
{
      salary = salary + increment;
}
```

Class

Blueprint for defining a set of similar objects.

- Objects in a class are called *instances*.
- Class is also an object with own *class attributes* and *class methods*.

Subclasses, Superclasses, and Inheritance

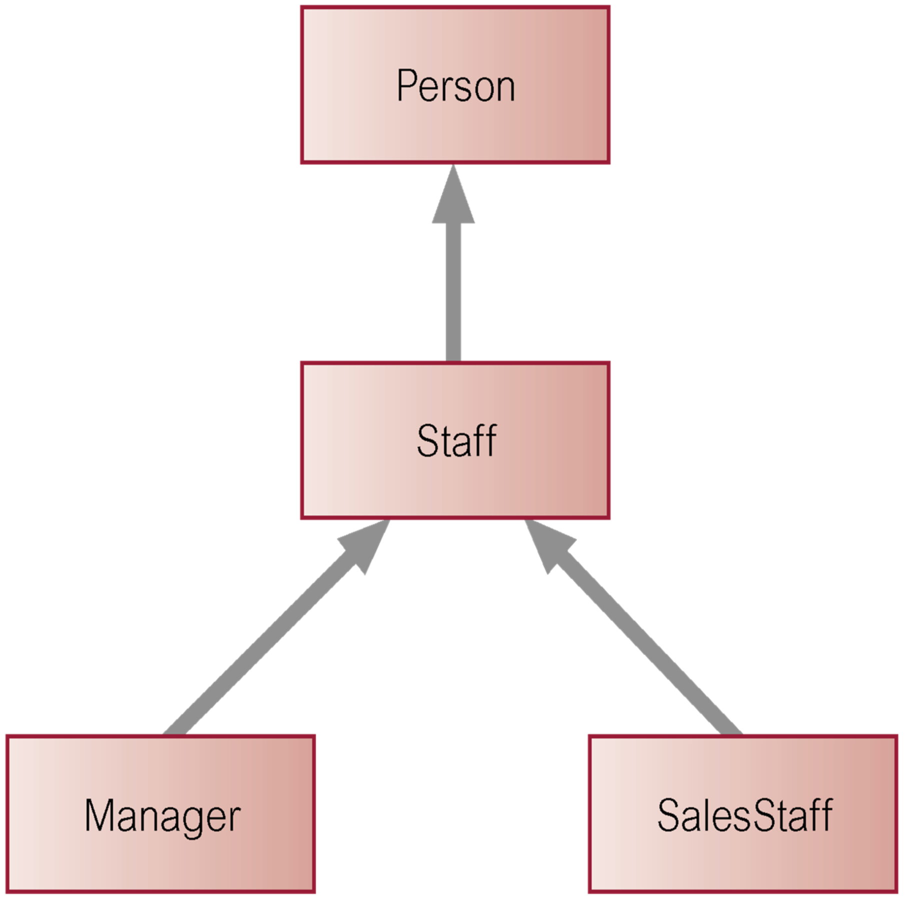
Inheritance allows one class of objects to be defined as a special case of a more general class.

- Special cases are *subclasses* and more general cases are *superclasses*.
- Process of forming a superclass is *generalization*; forming a subclass is *specialization*.
- Subclass inherits all properties of its superclass and can define its own unique properties.
- Subclass can redefine inherited methods.

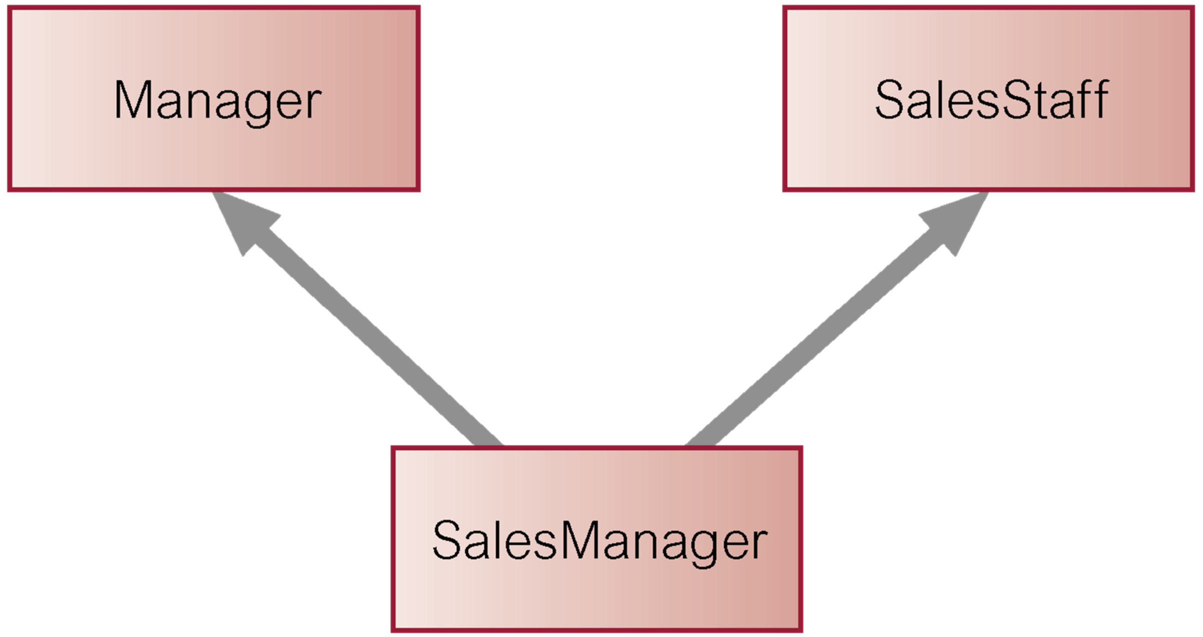
Subclasses, Superclasses, and Inheritance

- All instances of subclass are also instances of superclass.
- *Principle of substitutability* states that instance of subclass can be used whenever method/construct expects instance of superclass.
- Relationship between subclass and superclass known as A KIND OF (AKO) relationship.
- Four types of inheritance: single, multiple, repeated, and selective.

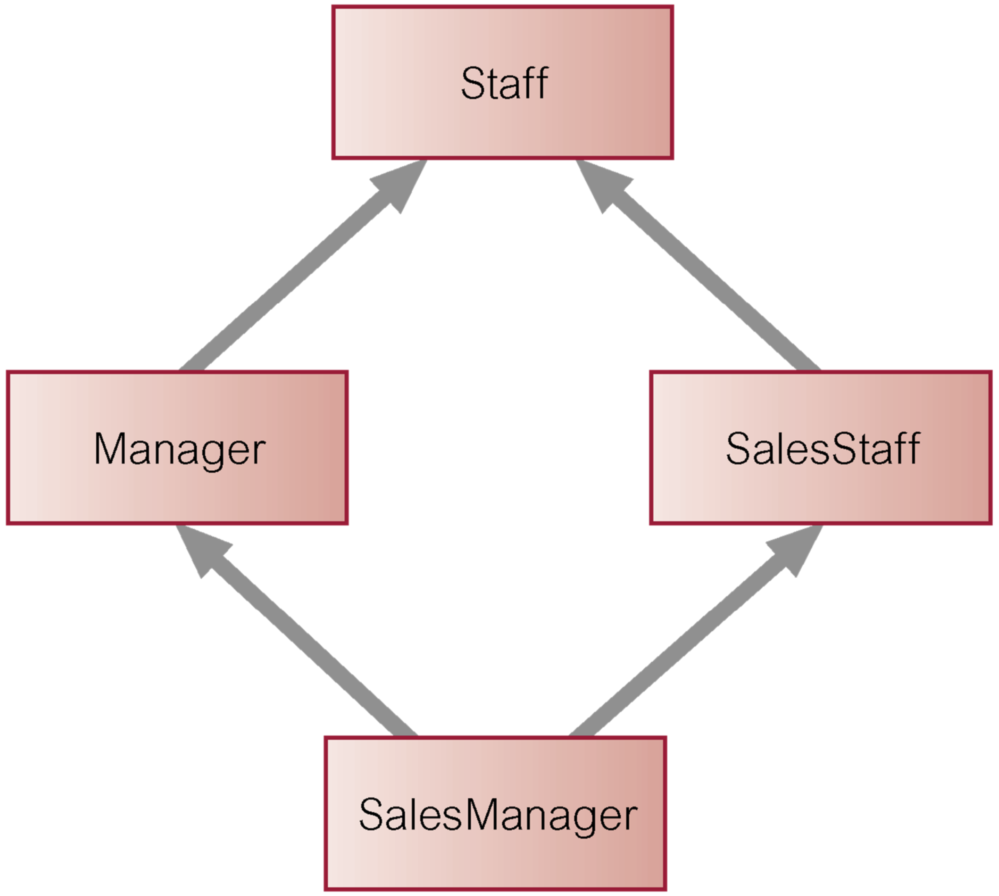
Single Inheritance



Multiple Inheritance



Repeated Inheritance



Overriding, Overloading, and Polymorphism

Overriding

- Process of redefining a property within a subclass.

Overloading

- Allows name of a method to be reused within a class or across classes.

Polymorphism

- Means ‘*many forms*’. Three types: operation, inclusion, and parametric.

Example of Overriding

- Might define method in Staff class to increment salary based on commission:

```
method void giveCommission(float branchProfit) {
     salary = salary + 0.02 * branchProfit; }
```

- May wish to perform different calculation for commission in Manager subclass:

```
method void giveCommission(float branchProfit) {
     salary = salary + 0.05 * branchProfit; }
```

Dynamic Binding

- Runtime process of selecting appropriate method based on an object’s type.
- With list consisting of an arbitrary number of objects from the Staff hierarchy, we can write:
  - list[i]. print
- and runtime system will determine which print() method to invoke depending on the object’s (sub)type.

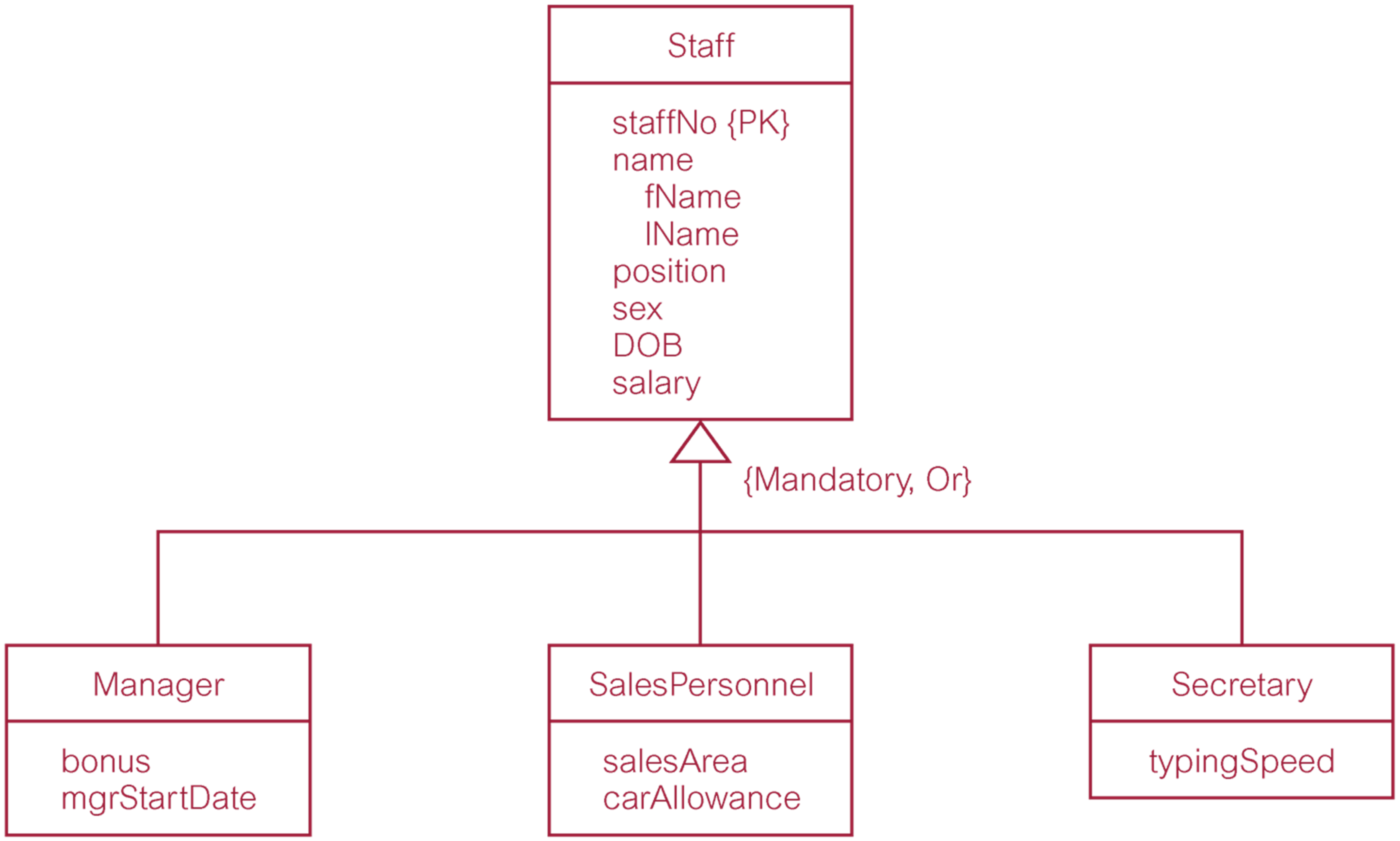
Complex Objects

- An object that consists of subobjects but is viewed as a single object.
- Objects participate in a A-PART-OF (APO) relationship.
- Contained object can be encapsulated within complex object, accessed by complex object’s methods.
- Or have its own independent existence, and only an OID is stored in complex object.

Storing Objects in Relational Databases

- One approach to achieving persistence with an OOPL is to use an RDBMS as the underlying storage engine.
- Requires mapping class instances (i.e. objects) to one or more tuples distributed over one or more relations.
- To handle class hierarchy, have two basics tasks to perform:
  - design relations to represent class hierarchy;
  - design how objects will be accessed.
    - Write code to decompose objects into tuples & store decomposed objects in relations
    - Write code to read tuples from relation & reconstruct objects

Storing Objects in Relational Databases



Mapping Classes to Relations

- Number of strategies for mapping classes to relations, although each results in a loss of semantic information.

(1) Map each class or subclass to a relation:

Staff (staffNo, fName, lName, position, sex, DOB, salary)

Manager (staffNo, bonus, mgrStartDate)

SalesPersonnel (staffNo, salesArea, carAllowance)

Secretary (staffNo, typingSpeed)

Mapping Classes to Relations

(2) Map each subclass to a relation

Manager (staffNo, fName, lName, position, sex, DOB, salary, bonus, mgrStartDate)

SalesPersonnel (staffNo, fName, lName, position, sex, DOB, salary, salesArea, carAllowance)

Secretary (staffNo, fName, lName, position, sex, DOB, salary, typingSpeed)

(3) Map the hierarchy to a single relation

Staff (staffNo, fName, lName, position, sex, DOB, salary, bonus, mgrStartDate, salesArea, carAllowance, typingSpeed, typeFlag)

Sample implementation for Manager in 1st case

```
EXEC SQL INCLUDE sqlca;
EXEC SQL BEGIN DECLARE SECTION;
class Manager : public Staff {
public:
            Manager (int, char *, char *, char *, char, int, int, int,
            float, float, int, int, int);
            ~Manager();
            void print();
Private:
            float bonus;
            Date mgrStartDate; }
EXEC SQL END DECLARE SECTION;
```

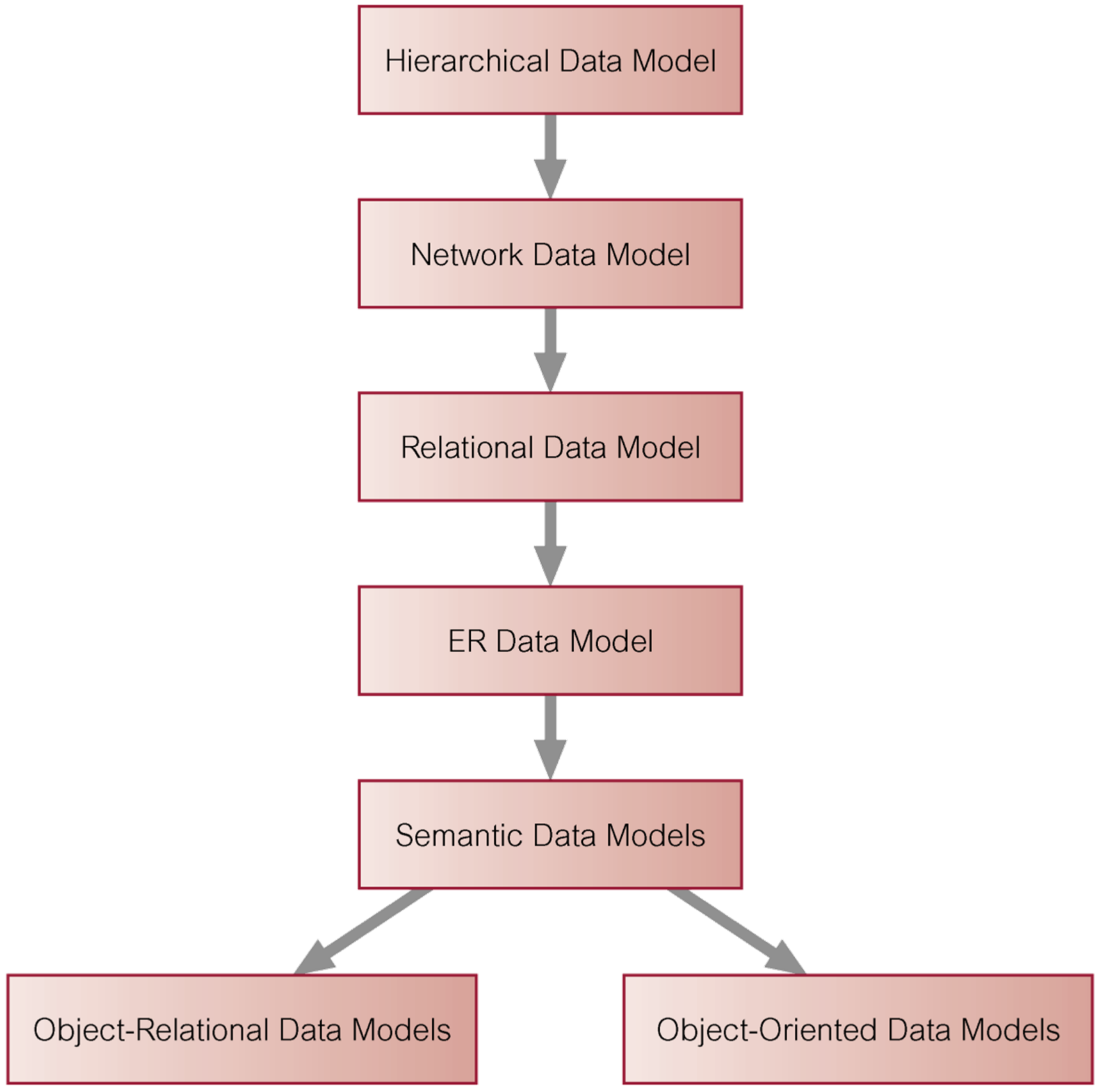
Sample implementation for Manager in 2nd case

```
EXEC SQL INCLUDE sqlca;
EXEC SQL BEGIN DECLARE SECTION;
class Manager {
public:
            Manager (int, char *, char *, char *, char, int, int, int,
            float, float, int, int, int);
            ~Manager();
            void print();
Private:
            int staffNo;
            char fName[15];
            char lName[15];
            char position[10];
            char sex;
            Date DOB;
            float salary;
            float bonus;
            Date mgrStartDate; }
EXEC SQL END DECLARE SECTION;
```

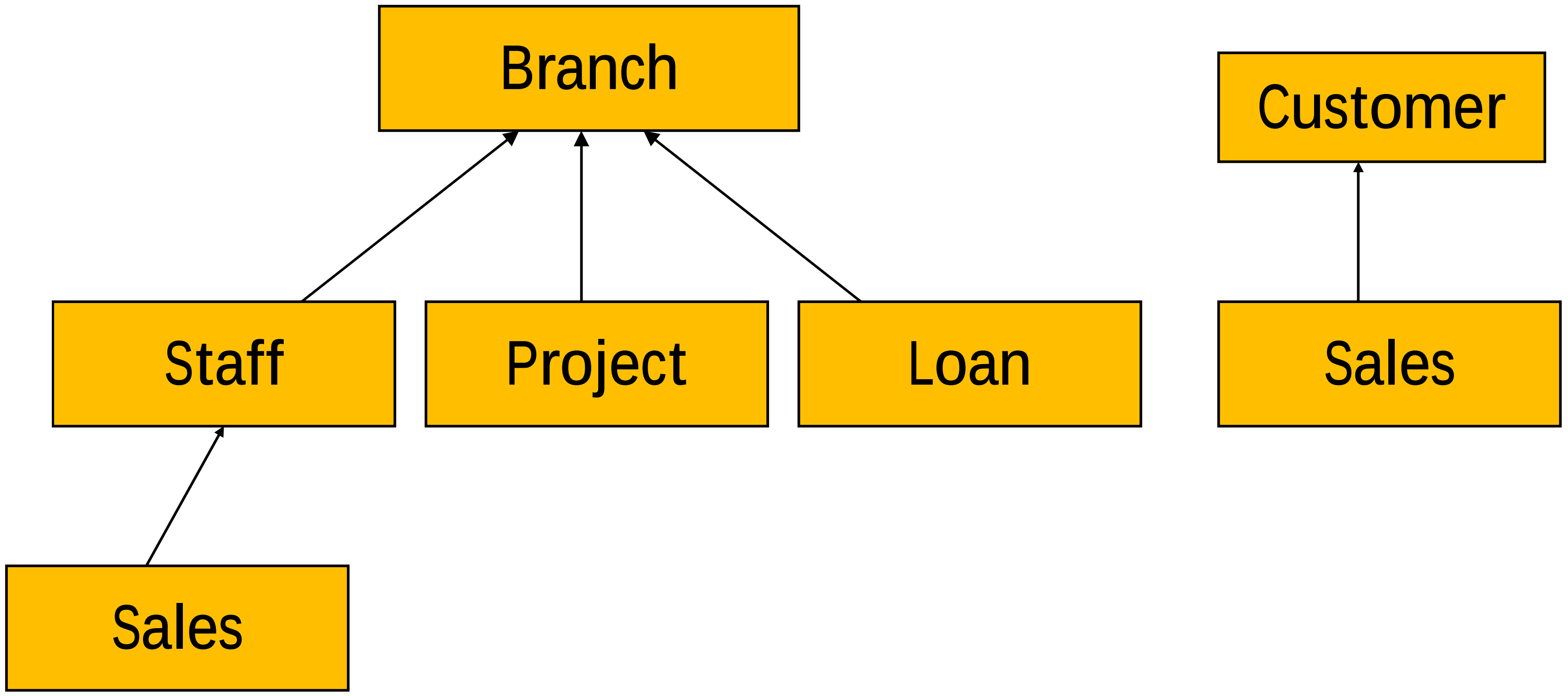
Next Generation Database Systems

- First Generation DBMS: Network and Hierarchical
  - Required complex programs for even simple queries.
  - Minimal data independence.
  - No widely accepted theoretical foundation.
- Second Generation DBMS: Relational DBMS
  - Helped overcome these problems.
- Third Generation DBMS: OODBMS and ORDBMS.

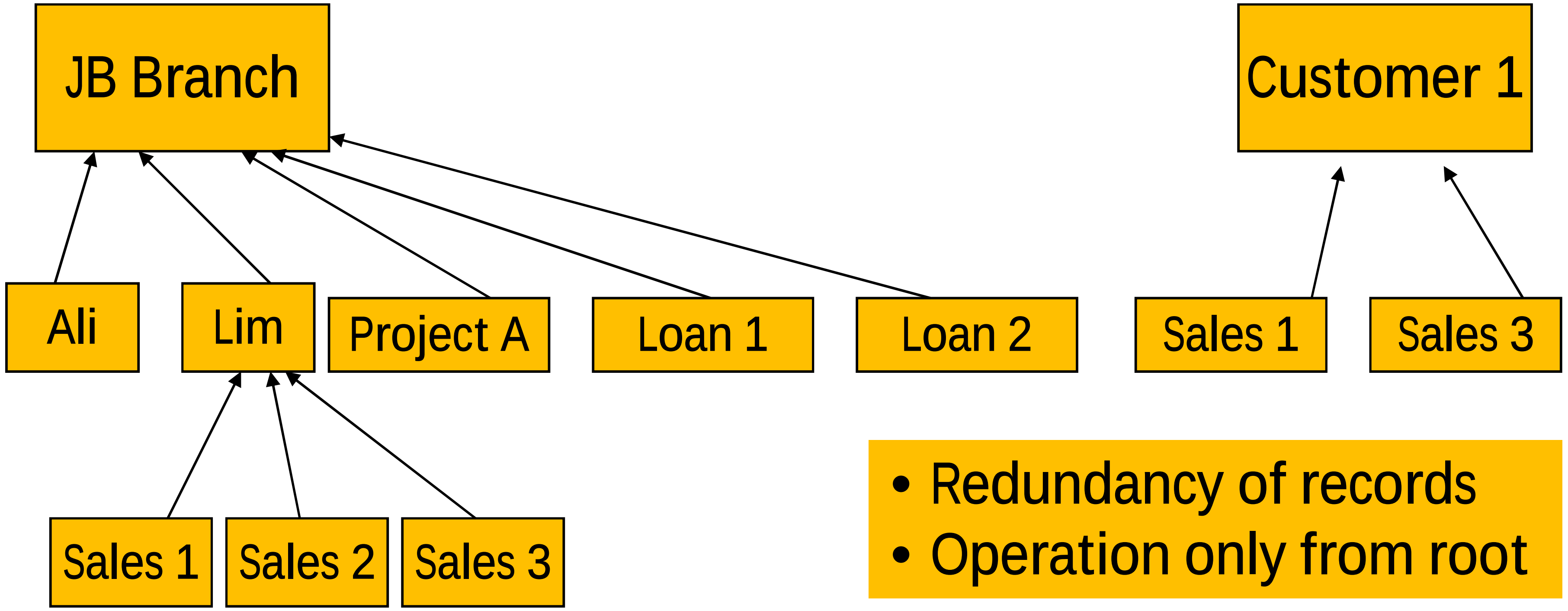
History of Data Models



E.g. Hierarchical Model



E.g. Hierarchical Model



- Redundancy of records
- Operation only from root

Object-Oriented DBMS: Concepts and Design

Object-Oriented Data Model

- No one agreed object data model. One definition:
- Object-Oriented Data Model (OODM)
  - Data model that captures semantics of objects supported in object-oriented programming.
- Object-Oriented Database (OODB)
  - Persistent and sharable collection of objects defined by an ODM.
- Object-Oriented DBMS (OODBMS)
  - Manager of an ODB.

Object-Oriented Data Model

- Zdonik and Maier present a threshold model that an OODBMS must, at a minimum, satisfy:
  - It must provide database functionality.
  - It must support object identity.
  - It must provide encapsulation.
  - It must support objects with complex state.

Object-Oriented Data Model

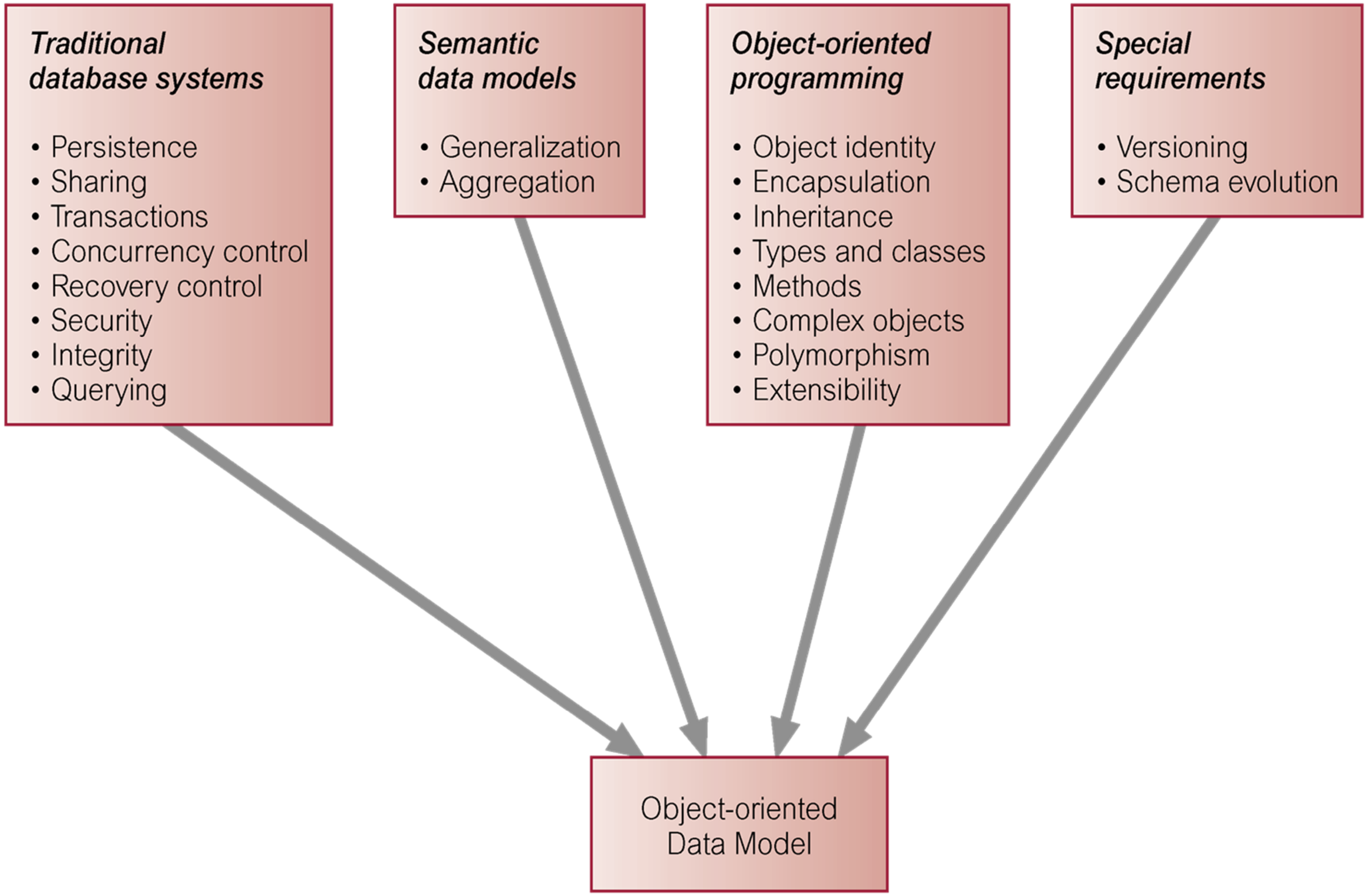
- Khoshafian and Abnous define OODBMS as:
  - OO = ADTs + Inheritance + Object identity
  - OODBMS = OO + Database capabilities.

- Parsaye *et al*. gives:
  - High-level query language with query optimization.
  - Support for persistence, atomic transactions: concurrency and recovery control.
  - Support for complex object storage, indexes, and access methods.
  - OODBMS = OO system + (1), (2), and (3).

Commercial OODBMSs

- GemStone from Gemstone Systems Inc.,
- Itasca from Ibex Knowledge Systems SA,
- Objectivity/DB from Objectivity Inc.,
- ObjectStore from eXcelon Corp.,
- Ontos from Ontos Inc.,
- Poet from Poet Software Corp.,
- Jasmine from Computer Associates/Fujitsu,
- Versant from Versant Object Technology.

Origins of the Object-Oriented Data Model



Persistent Programming Languages (PPLs)

- Language that provides users with ability to (transparently) preserve data across successive executions of a program, and even allows such data to be used by many different programs.

- In contrast, database programming language (e.g. SQL) differs by its incorporation of features beyond persistence, such as transaction management, concurrency control, and recovery.

Persistent Programming Languages (PPLs)

- PPLs eliminate impedance mismatch by extending programming language with database capabilities.
  - In PPL, language's type system provides data model, containing rich structuring mechanisms.

- In some PPLs procedures are 'first class' objects and are treated like any other object in language.
  - Procedures are assignable, may be result of expressions, other procedures or blocks, and may be elements of constructor types.
  - Procedures can be used to implement ADTs.

Persistent Programming Languages (PPLs)

- PPL also maintains same data representation in memory as in persistent store.
  - Overcomes difficulty and overhead of mapping between the two representations.
- Addition of (transparent) persistence into a PPL is important enhancement to IDE, and integration of two paradigms provides more functionality and semantics.

Alternative Strategies for Developing an OODBMS

- Extend existing object-oriented programming language.
  - GemStone extended Smalltalk.
- Provide extensible OODBMS library.
  - Approach taken by Ontos, Versant, and ObjectStore.
- Embed OODB language constructs in a conventional host language.
  - Approach taken by O_2,which has extensions for C.

Alternative Strategies for Developing an OODBMS

- Extend existing database language with object-oriented capabilities.
  - Approach being pursued by RDBMS and OODBMS vendors.
  - Ontos and Versant provide a version of OSQL.
- Develop a novel database data model/language.

Single-Level vs. Two-Level Storage Model

- Traditional programming languages lack built-in support for many database features.
- Increasing number of applications now require functionality from both database systems and programming languages.
- Such applications need to store and retrieve large amounts of shared, structured data.

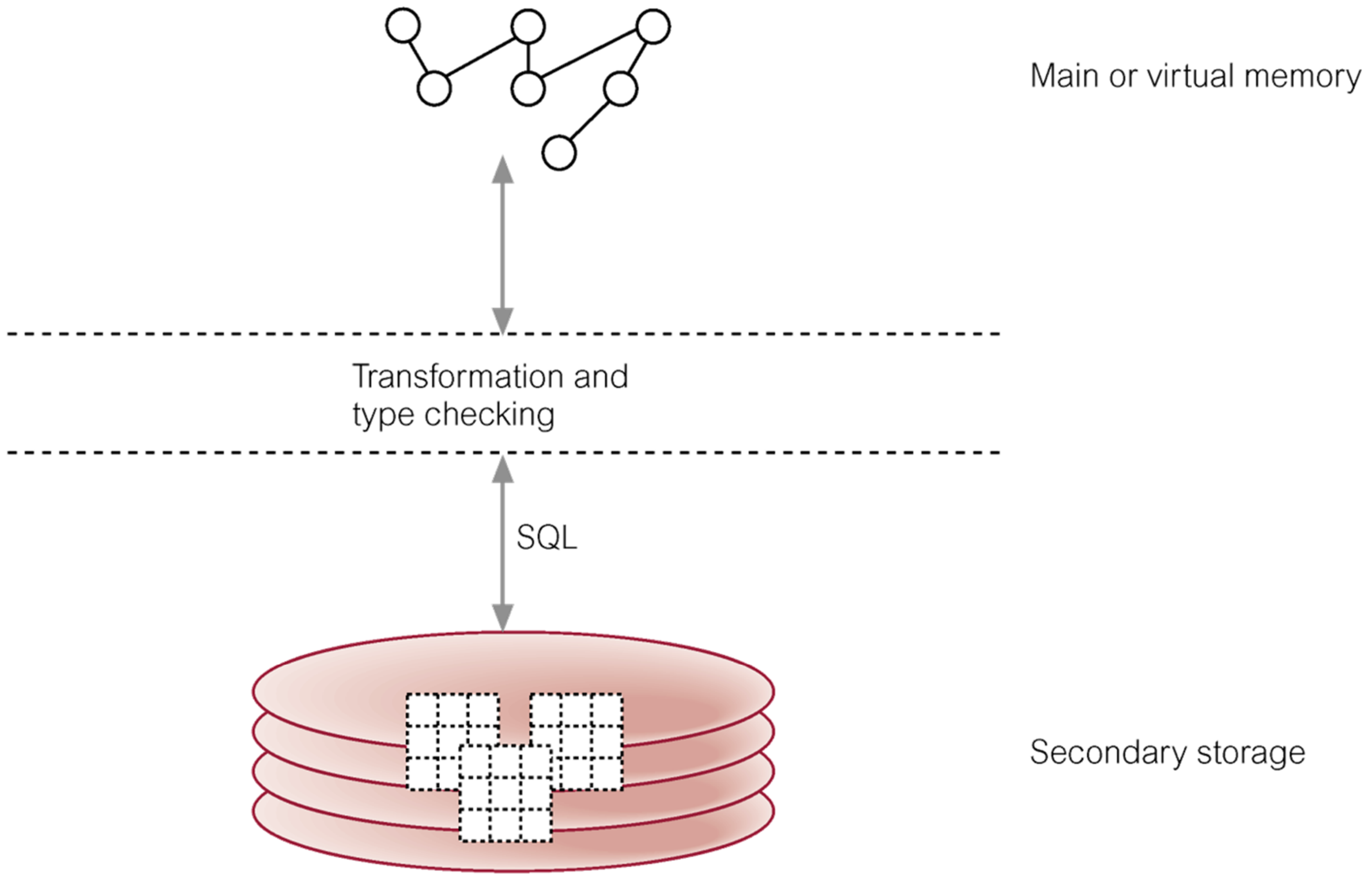
Single-Level vs. Two-Level Storage Model

- With a traditional DBMS, programmer has to:
  - Decide when to read and update objects.
  - Write code to translate between application's object model and the data model of the DBMS.
  - Perform additional type-checking when object is read back from database, to guarantee object will conform to its original type.

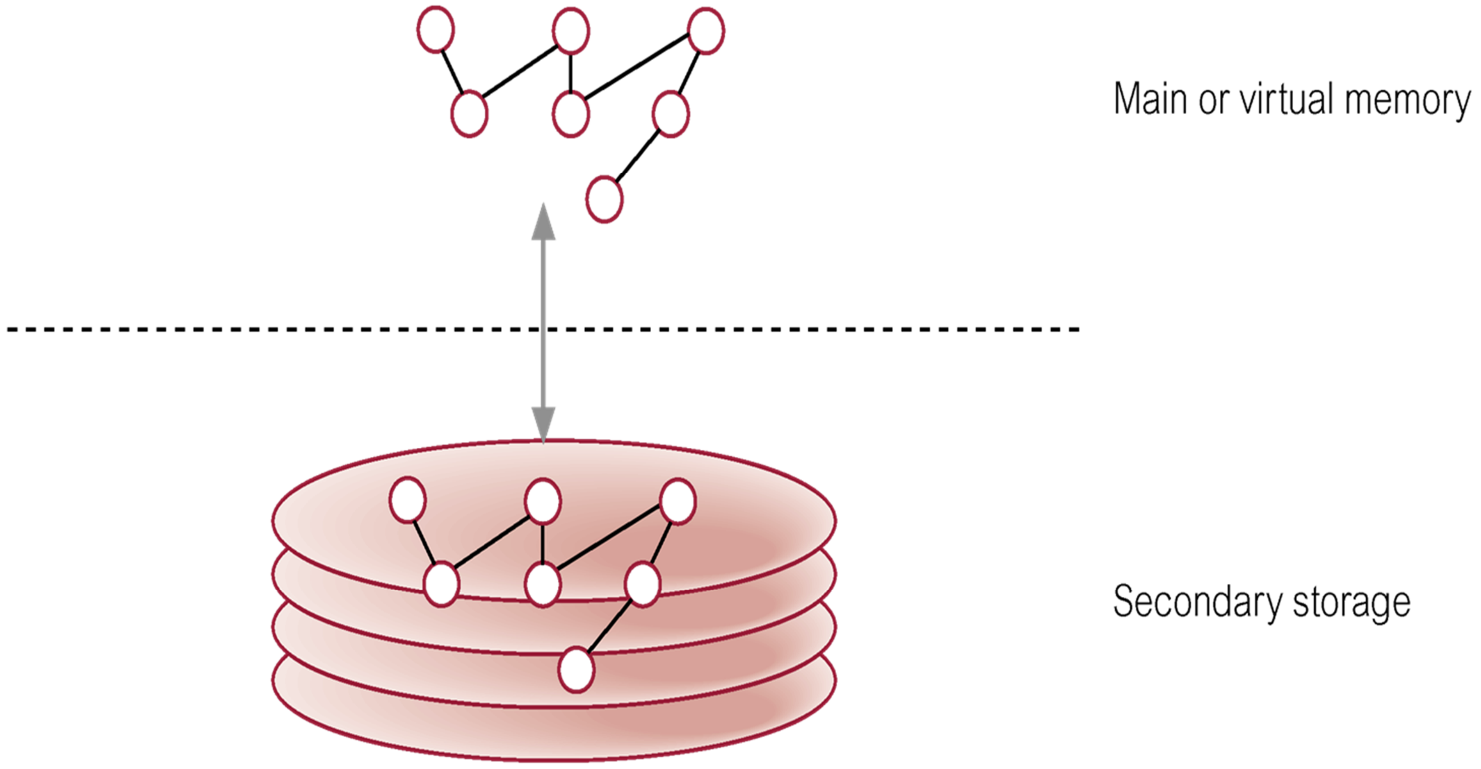
Single-Level vs. Two-Level Storage Model

- Difficulties occur because conventional DBMSs have two-level storage model: storage model in memory, and database storage model on disk.
- In contrast, OODBMS gives illusion of single-level storage model, with similar representation in both memory and in database stored on disk.
  - Requires clever management of representation of objects in memory and on disk (called "pointer swizzling").

Two-Level Storage Model for RDBMS



Single-Level Storage Model for OODBMS



Object Referencing

- Need to distinguish between resident and non-resident objects.
- Most techniques variations of edge marking or node marking.
- Edge marking marks every object pointer with a tag bit:
  - if bit set, reference is to memory pointer;
  - else, still pointing to OID and needs to be swizzled when object it refers to is faulted into.

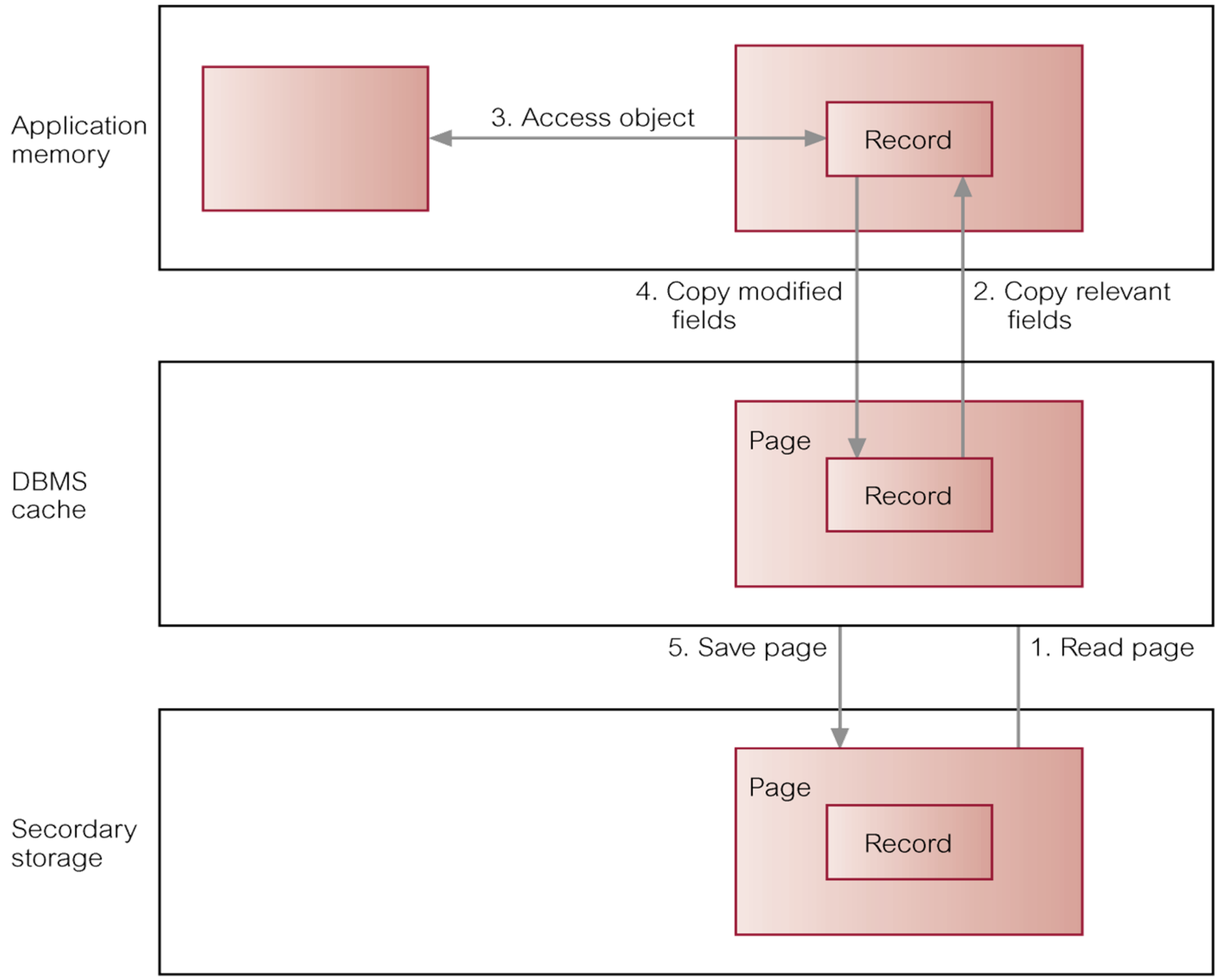
Object Referencing

- Node marking requires that all object references are immediately converted to virtual memory pointers when object is faulted into memory.
- First approach is software-based technique but second can be implemented using software or hardware-based techniques.

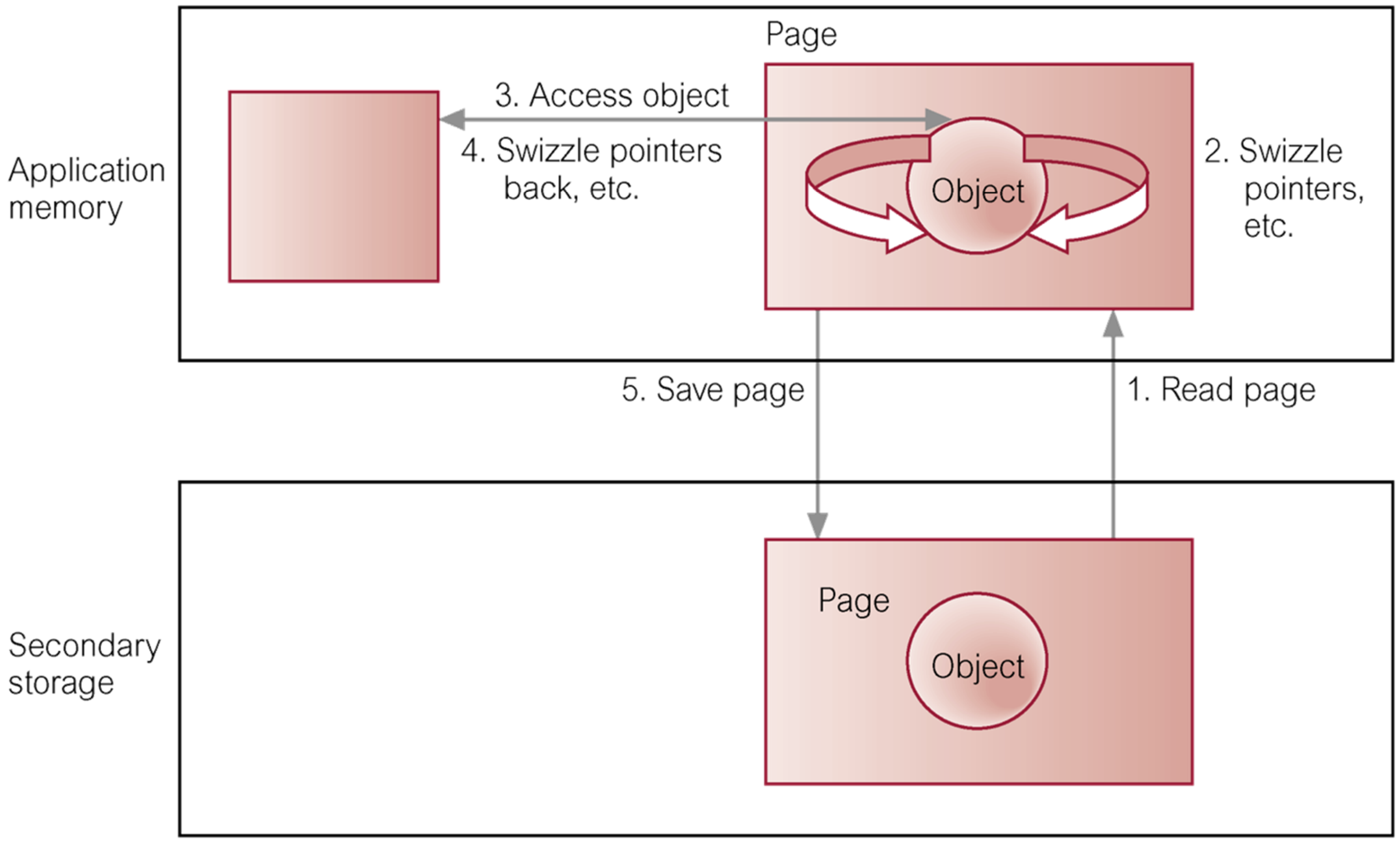
Hardware-Based Schemes

- Use virtual memory access protection violations to detect accesses of non-resident objects.
- Use standard virtual memory hardware to trigger transfer of persistent data from disk to memory.
- Once page has been faulted in, objects are accessed via normal virtual memory pointers and no further object residency checking is required.
- Avoids overhead of residency checks incurred by software approaches.

Accessing an Object with a RDBMS



Accessing an Object with an OODBMS



Persistent Schemes

- Consider three persistent schemes:
  - Checkpointing.
  - Serialization.
  - Explicit Paging.
- Note, persistence can also be applied to (object) code and to the program execution state.

Checkpointing

- Copy all or part of program’s address space to secondary storage.
- If complete address space saved, program can restart from checkpoint.
- In other cases, only program’s heap saved.
- Two main drawbacks:
  - Can only be used by program that created it.
  - May contain large amount of data that is of no use in subsequent executions.

Serialization

- Copy closure of a data structure to disk.
- Write on a data value may involve traversal of graph of objects reachable from the value, and writing of flattened version of structure to disk.
- Reading back flattened data structure produces new copy of original data structure.
- Sometimes called serialization, pickling, or in a distributed computing context, marshaling.

Serialization

- Two inherent problems:
  - Does not preserve object identity.
  - Not incremental, so saving small changes to a large data structure is not efficient.

Explicit Paging

- Explicitly ‘page’ objects between application heap and persistent store.
- Usually requires conversion of object pointers from disk-based scheme to memory-based scheme.
- Two common methods for creating/updating persistent objects:
  - Reachability-based.
  - Allocation-based.

Orthogonal Persistence

- Three fundamental principles:
  - Persistence independence.
  - Data type orthogonality.
  - Transitive persistence (originally referred to as ‘persistence identification’ but ODMG term ‘transitive persistence’ used here).

Persistence Independence

- Persistence of object independent of how program manipulates that object.
- Conversely, code fragment independent of persistence of data it manipulates.
- Should be possible to call function with its parameters sometimes objects with long term persistence and sometimes only transient.
- Programmer does not need to control movement of data between long-term and short-term storage.

Data Type Orthogonality

- All data objects should be allowed full range of persistence irrespective of their type.
- No special cases where object is not allowed to be long-lived or is not allowed to be transient.
- In some PPLs, persistence is quality attributable to only subset of language data types.

Transitive Persistence

- Choice of how to identify and provide persistent objects at language level is independent of the choice of data types in the language.
- Technique that is now widely used for identification is reachability-based.

Orthogonal Persistence - Advantages

- Improved programmer productivity from simpler semantics.
- Improved maintenance.
- Consistent protection mechanisms over whole environment.
- Support for incremental evolution.
- Automatic referential integrity.

Orthogonal Persistence - Disadvantages

- Some runtime expense in a system where every pointer reference might be addressing persistent object.
  - System required to test if object must be loaded in from disk-resident database.
- Although orthogonal persistence promotes transparency, system with support for sharing among concurrent processes cannot be fully transparent.

OODBMS Issues

- Long duration transaction
- Versions
- Schema evolution

Transaction Support

Transaction

Action, or series of actions, carried out by user or application, which accesses or changes contents of database.

- Logical unit of work on the database.
- Application program is series of transactions with non-database processing in between.
- Transforms database from one consistent state to another, although consistency may be violated during transaction.

Example Transaction

```
read(staffNo = x, salary)
salary = salary * 1.1
write(staffNo = x, new_salary)
```

```
delete(staffNo = x)
for all PropertyForRent records, pno
begin
    read(propertyNo = pno, staffNo)
    if (staffNo = x) then
    begin
        staffNo = newStaffNo
        write(propertyNo = pno, staffNo)
    end
end
```

Transaction Support

- Can have one of two outcomes:
  - Success: transaction *commits* and database reaches a new consistent state.
  - Failure: transaction *aborts*, and database must be restored to consistent state before it started.
  - Such a transaction is *rolled back* or *undone*.
- Committed transaction cannot be aborted.
- Aborted transaction that is rolled back can be restarted later.

Properties of Transactions

- Four basic *(ACID)* properties of a transaction are:

Atomicity	'All or nothing' property.
Consistency	Must transform database from one consistent state to another.
Isolation	Partial effects of incomplete transactions should not be visible to other transactions.
Durability	Effects of a committed transaction are permanent and must not be lost because of later failure.

OODBMS Transaction

- Transactions of application with complex objects can be much longer than those of business transactions
- Unit of concurrency control & recovery normally an object
- Unacceptable when long transaction aborted owing to a lock conflict

Timestamping

Timestamp

A unique identifier created by DBMS that indicates relative starting time of a transaction.

§ Can be generated by using system clock at time transaction started, or by incrementing a logical counter every time a new transaction starts.

Timestamping

- Read/write proceeds only if *last update on that data item* was carried out by an older transaction.
- Otherwise, transaction requesting read/write is restarted and given a new timestamp.
- Also timestamps for data items:
  - read-timestamp - timestamp of last transaction to read item;
  - write-timestamp - timestamp of last transaction to write item.

Multiversion Timestamp Ordering

- Versioning of data can be used to increase concurrency.
- Basic timestamp ordering protocol assumes only one version of data item exists, and so only one transaction can access data item at a time.
- Can allow multiple transactions to read and write different versions of same data item, and ensure each transaction sees consistent set of versions for all data items it accesses.

Nested Transaction Model

- Transaction viewed as hierarchy of subtransactions.
- Top-level transaction can have number of child transactions.
- Each child can also have nested transactions.
- In Moss's proposal, only leaf-level subtransactions allowed to perform database operations.
- Transactions have to commit from bottom upwards.
- However, transaction abort at one level does not have to affect transaction in progress at higher level.

Nested Transaction Model

- Parent allowed to perform its own recovery:
  - Retry subtransaction.
  - Ignore failure, in which case subtransaction non-vital.
  - Run contingency subtransaction.
  - Abort.
- Updates of committed subtransactions at intermediate levels are visible only within scope of their immediate parents.

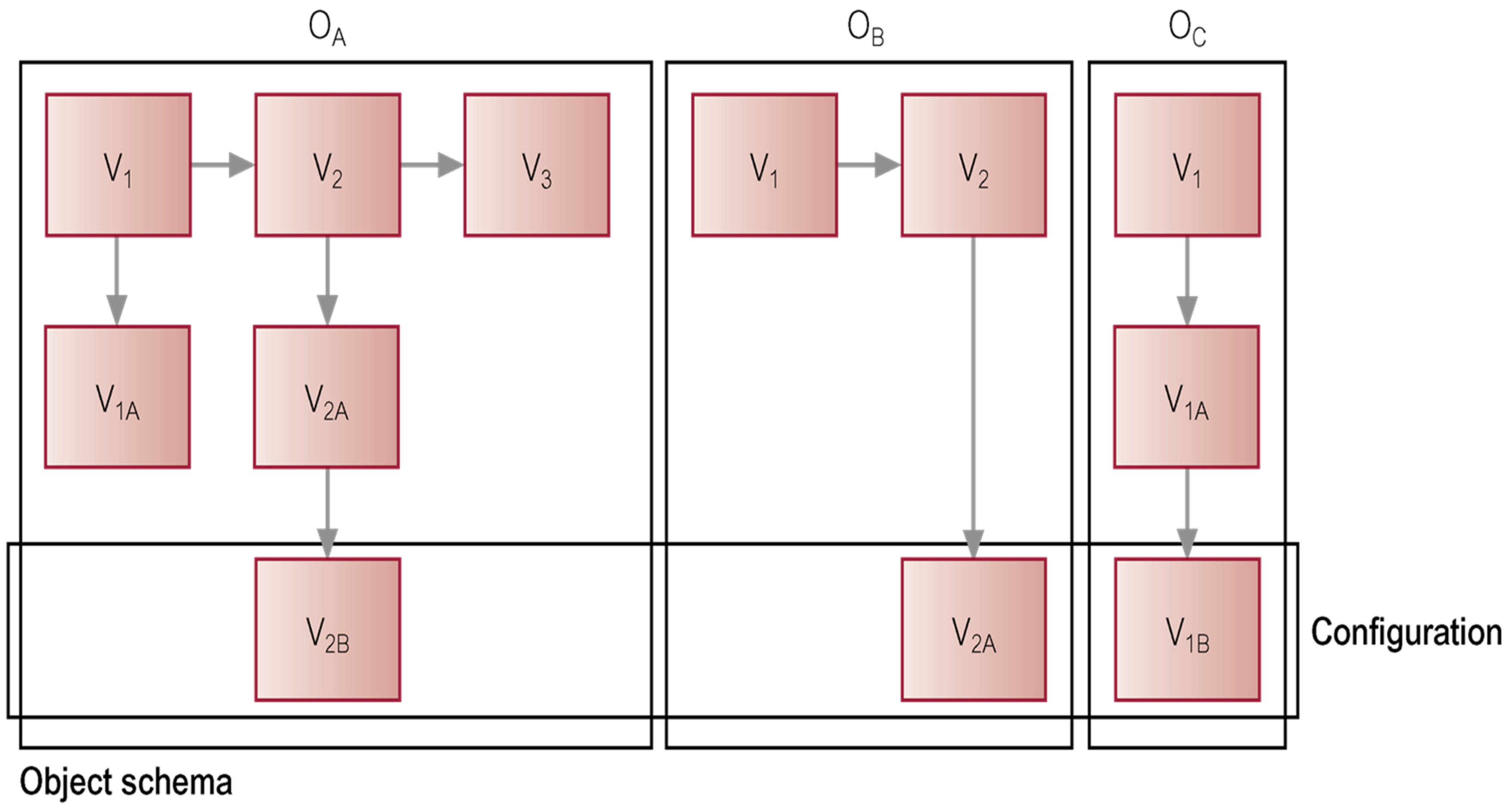
Nested Transaction Model

- Further, commit of subtransaction is conditionally subject to commit or abort of its superiors.
- Using this model, top-level transactions conform to traditional ACID properties of flat transaction.

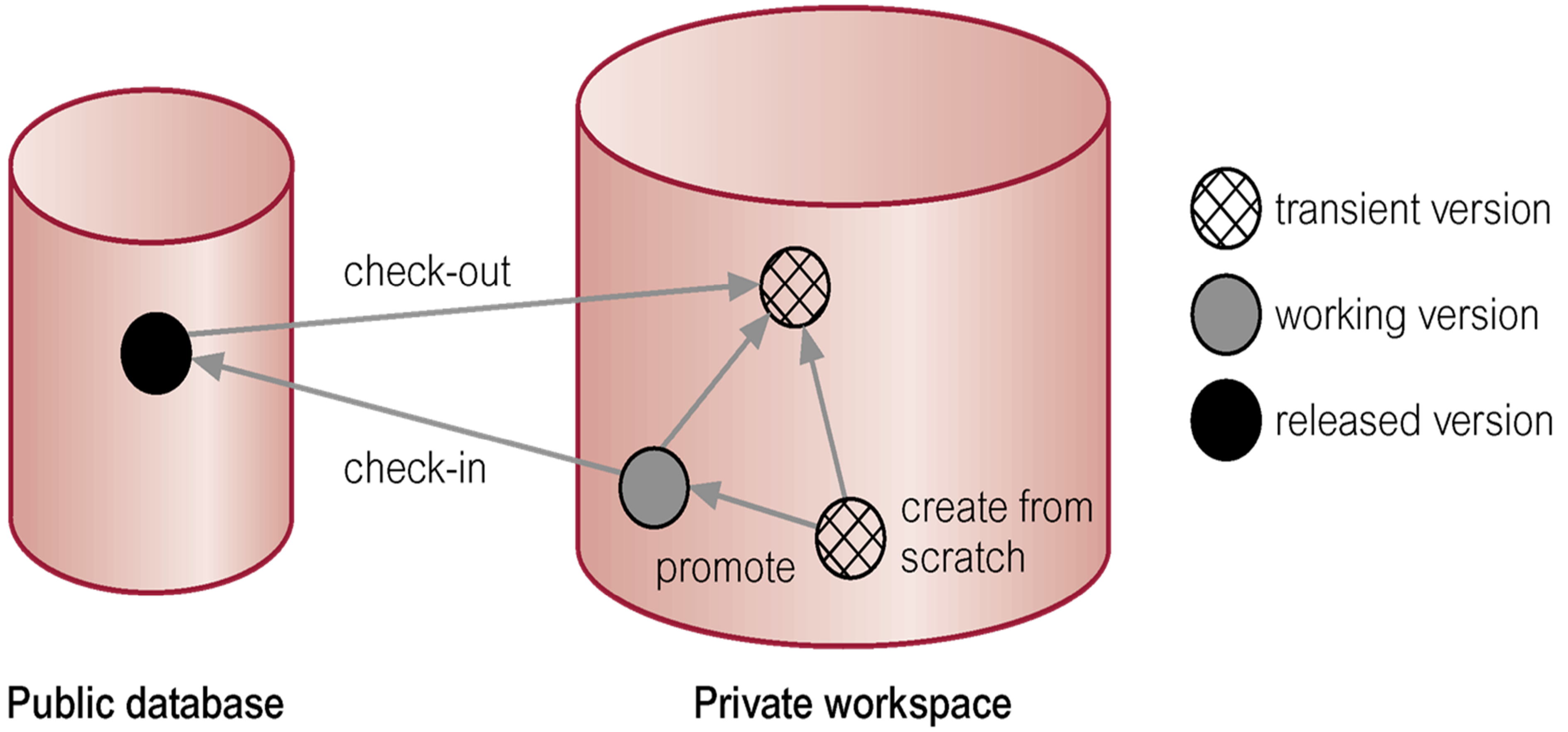
Versions

- Allows changes to properties of objects to be managed so that object references always point to correct object version.
- Itasca identifies 3 types of versions:
  - Transient Versions.
  - Working Versions.
  - Released Versions.

Versions and Configurations



Versions and Configurations



Schema Evolution

- Some applications require considerable flexibility in dynamically defining and modifying database schema.
- Typical schema changes:

(1) Changes to class definition:

(a) Modifying Attributes.

(b) Modifying Methods.

Schema Evolution

(2) Changes to inheritance hierarchy:

- (a) Making a class S superclass of a class C.
- (b) Removing S from list of superclasses of C.
- (c) Modifying order of superclasses of C.

(3) Changes to set of classes, such as creating and deleting classes and modifying class names.

§ Changes must not leave schema inconsistent.

Schema Consistency

1. Resolution of conflicts caused by multiple inheritance and redefinition of attributes and methods in a subclass.

1.1 Rule of precedence of subclasses over superclasses.

1.2 Rule of precedence between superclasses of a different origin.

1.3 Rule of precedence between superclasses of the same origin.

Schema Consistency

2. Propagation of modifications to subclasses.

2.1 Rule for propagation of modifications.

2.2 Rule for propagation of modifications in the event of conflicts.

2.3 Rule for modification of domains.

Schema Consistency

3. Aggregation and deletion of inheritance relationships between classes and creation and removal of classes.

3.1 Rule for inserting superclasses.

3.2 Rule for removing superclasses.

3.3 Rule for inserting a class into a schema.

3.4 Rule for removing a class from a schema.

Client-Server Architecture

- Three basic architectures:
  - Object Server.
  - Page Server.
  - Database Server.

Object Server

- Distribute processing between the two components.
- Typically, client is responsible for transaction management and interfacing to programming language.
- Server responsible for other DBMS functions.
- Best for cooperative, object-to-object processing in an open, distributed environment.

Page and Database Server

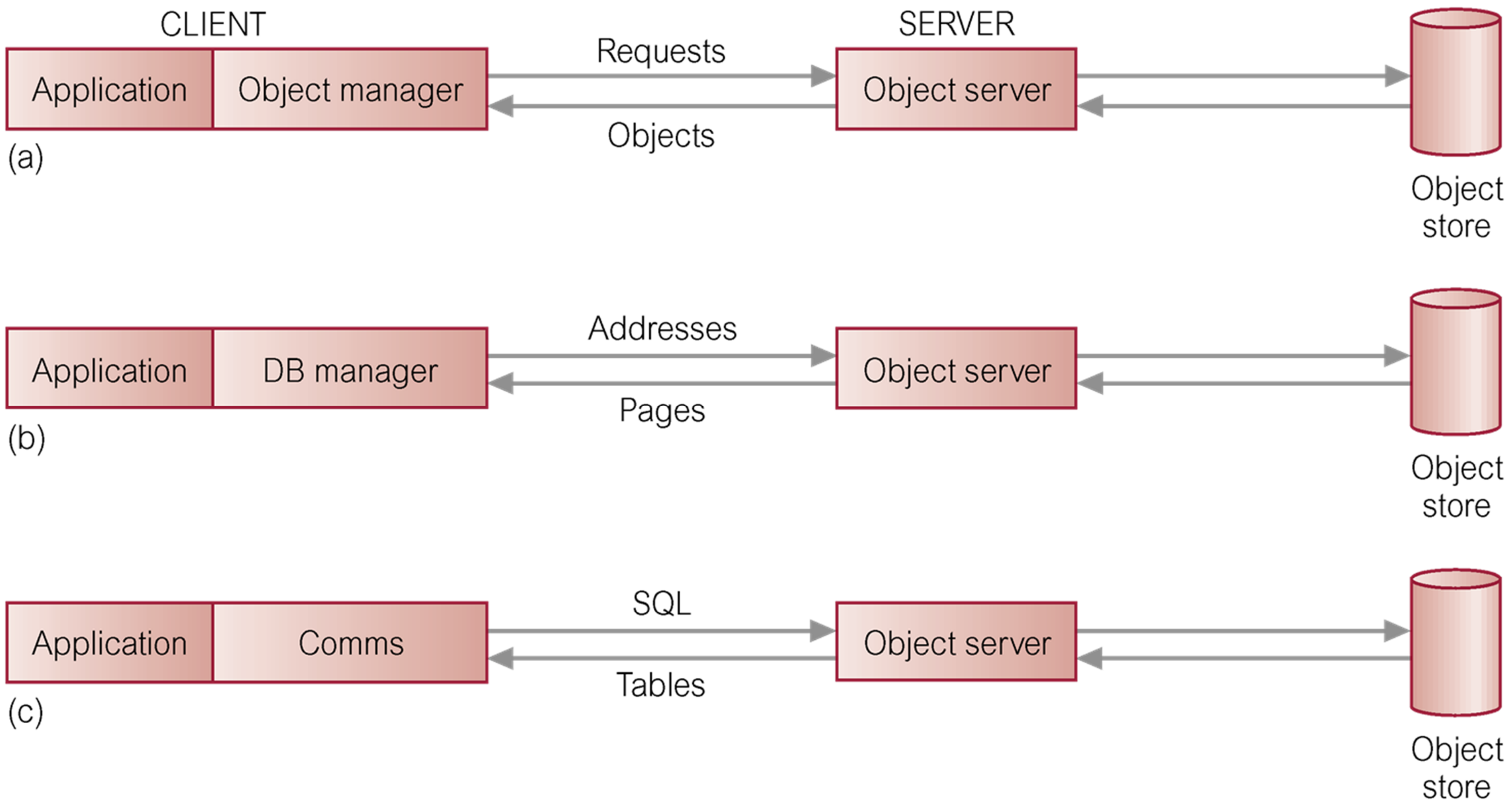
Page Server

- Most database processing is performed by client.
- Server responsible for secondary storage and providing pages at client's request.

Database Server

- Most database processing performed by server.
- Client simply passes requests to server, receives results and passes them to application.
- Approach taken by many RDBMSs.

Client-Server Architecture



Architecture - Storing and Executing Methods

- Two approaches:
  - Store methods in external files.
  - Store methods in database.
- Benefits of latter approach:
  - Eliminates redundant code.
  - Simplifies modifications.

Architecture - Storing and Executing Methods

  - Methods are more secure.
  - Methods can be shared concurrently.
  - Improved integrity.

- Obviously, more difficult to implement.

Object Operations Version 1 (OO1) Benchmark

- Intended as generic measure of OODBMS performance. Designed to reproduce operations common in advanced engineering applications, such as finding all parts connected to a random part, all parts connected to one of those parts, and so on, to a depth of seven levels.

- About 1990, benchmark was run on GemStone, Ontos, ObjectStore, Objectivity/DB, and Versant, and INGRES and Sybase. Results showed an average 30-fold performance improvement for OODBMSs over RDBMSs.

OO7 Benchmark

- More comprehensive set of tests and a more complex database based on parts hierarchy.
- Designed for detailed comparisons of OODBMS products.
- Simulates CAD/CAM environment and tests system performance in area of object-to-object navigation over cached data, disk-resident data, and both sparse and dense traversals.
- Also tests indexed and non-indexed updates of objects, repeated updates, and the creation and deletion of objects.

OODBMS Manifesto

- Complex objects must be supported.
- Object identity must be supported.
- Encapsulation must be supported.
- Types or Classes must be supported.
- Types or Classes must be able to inherit from their ancestors.
- Dynamic binding must be supported.
- The DML must be computationally complete.

OODBMS Manifesto

- The set of data types must be extensible.
- Data persistence must be provided.
- The DBMS must be capable of managing very large databases.
- The DBMS must support concurrent users.
- DBMS must be able to recover from hardware/software failures.
- DBMS must provide a simple way of querying data.

OODBMS Manifesto

- The manifesto proposes the following optional features:
  - Multiple inheritance, type checking and type inferencing, distribution across a network, design transactions and versions.
- No direct mention of support for security, integrity, views or even a declarative query language.

Advantages of OODBMSs

- Enriched Modeling Capabilities.
- Extensibility.
- Removal of Impedance Mismatch.
- More Expressive Query Language.
- Support for Schema Evolution.
- Support for Long Duration Transactions.
- Applicability to Advanced Database Applications.
- Improved Performance.

Disadvantages of OODBMSs

- Lack of Universal Data Model.
- Lack of Experience.
- Lack of Standards.
- Query Optimization compromises Encapsulation.
- Object Level Locking may impact Performance.
- Complexity.
- Lack of Support for Views.
- Lack of Support for Security.

Object-Oriented Database Design

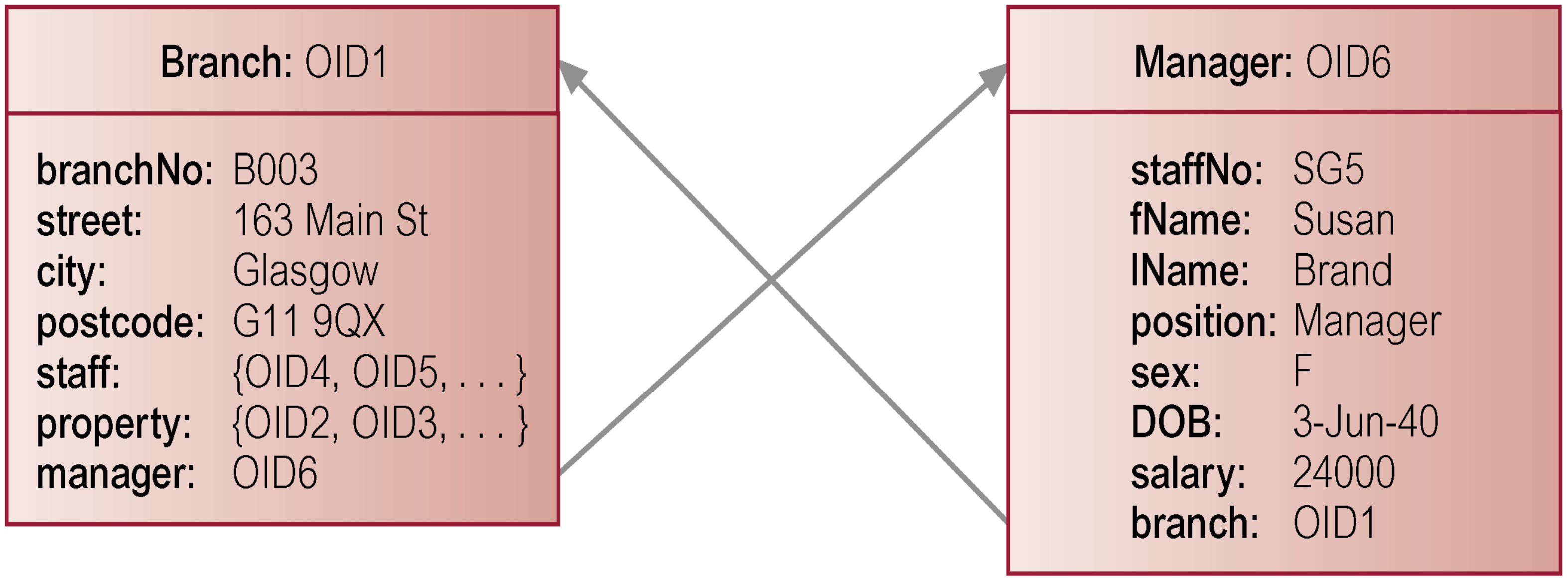
OODM	CDM	Difference
Object	Entity	Object includes behavior
Attribute	Attribute	None
Association	Relationship	Associations are the same but inheritance in OODM includes both state and behavior
Message		No corresponding concept in CDM
Class	Entity type/Supertype	None
Instance	Entity	None
Encapsulation		No corresponding concept in CDM

Relationships

- Relationships represented using reference attributes, typically implemented using OIDs.
- Consider how to represent following binary relationships according to their cardinality:
  - 1:1
  - 1:*
  - *:*

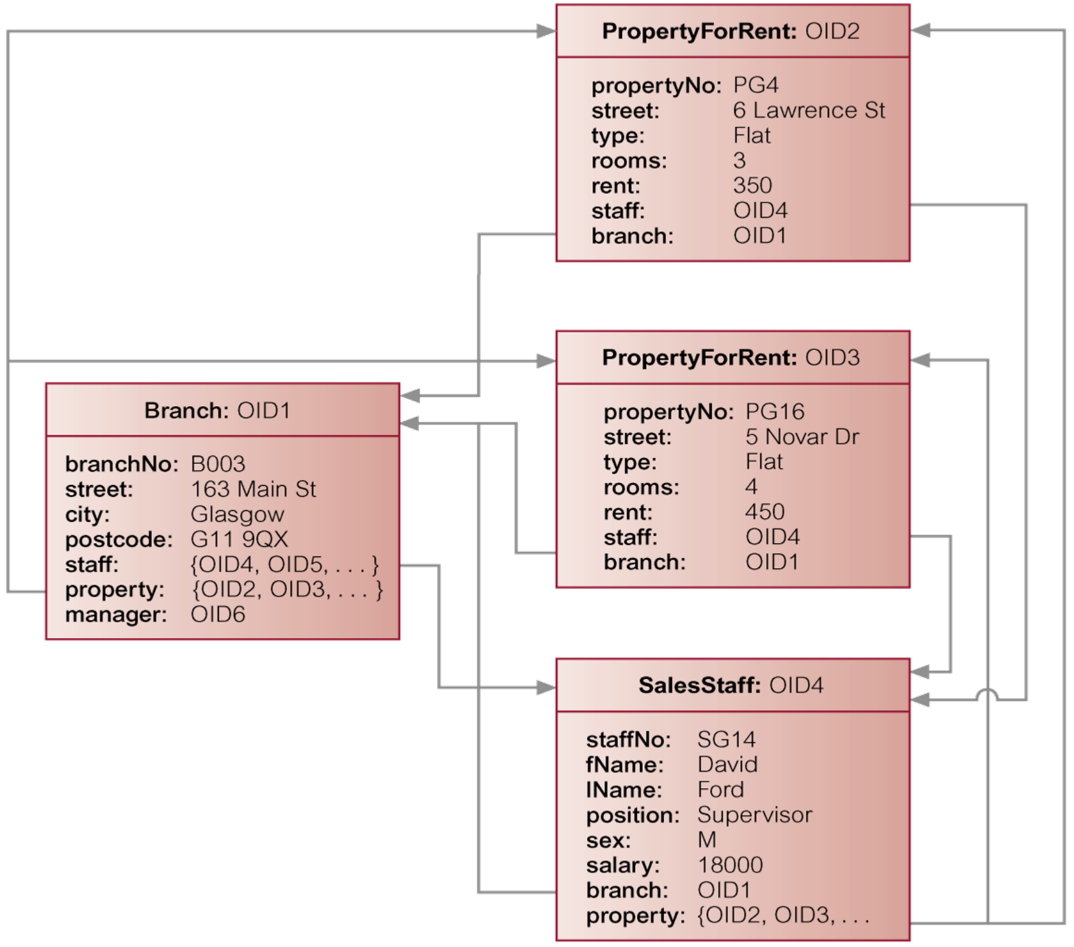
1:1 Relationship Between Objects A and B

- Add reference attribute to A and, to maintain referential integrity, reference attribute to B.



1:* Relationship Between Objects A and B

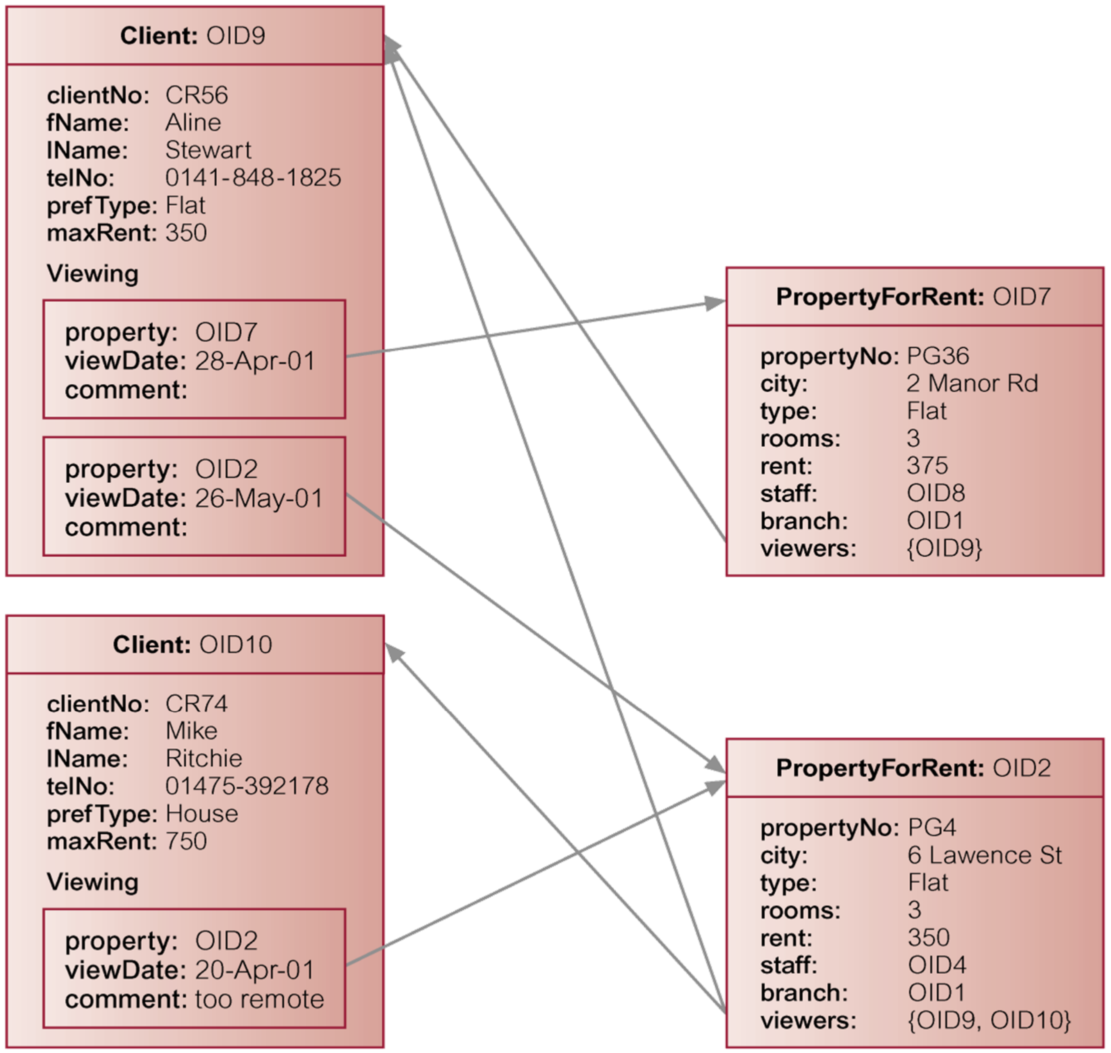
- Add reference attribute to B and attribute containing set of references to A.



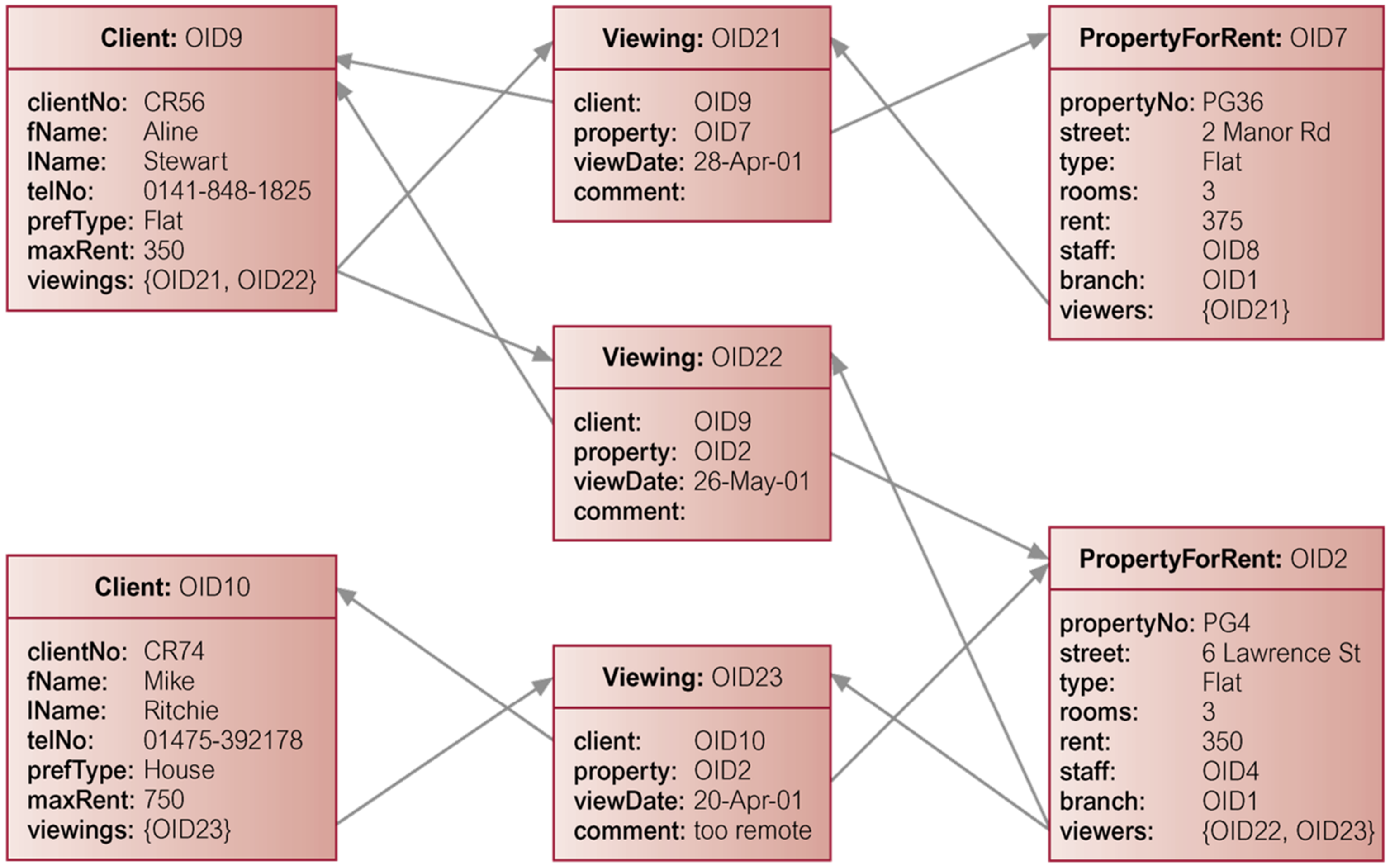
: Relationship Between Objects A and B

- Add attribute containing set of references to each object.
- For relational database design, would decompose *:N into two 1:* relationships linked by intermediate entity. Can also represent this model in an ODBMS.

: Relationships



Alternative Design for *:* Relationships



Referential Integrity

Several techniques to handle referential integrity:

- Do not allow user to explicitly delete objects.
  - System is responsible for “garbage collection”.

- Allow user to delete objects when they are no longer required.
  - System may detect invalid references automatically and set reference to NULL or disallow the deletion.

Referential Integrity

- Allow user to modify and delete objects and relationships when they are no longer required.
  - System automatically maintains the integrity of objects.
  - Inverse attributes can be used to maintain referential integrity.

Behavioral Design

- EER approach must be supported with technique that identifies behavior of each class.
- Involves identifying:
  - public methods: visible to all users
  - private methods: internal to class.
- Three types of methods:
  - constructors and destructors
  - access
  - transform.

Behavioral Design - Methods

- Constructor - creates new instance of class.
- Destructor - deletes class instance no longer required.
- Access - returns value of one or more attributes (Get).
- Transform - changes state of class instance (Put).

Identifying Methods

- Several methodologies for identifying methods, typically combine following approaches:
  - Identify classes and determine methods that may be usefully provided for each class.
  - Decompose application in top-down fashion and determine methods required to provide required functionality.