

Object Databases Tutorial

Michael Grossniklaus
Institute for Information Systems, ETH Zurich, Switzerland



About

- **Host**
 - Michael Grossniklaus (grossniklaus@inf.ethz.ch)
 - ETH Zurich (2001-2007) – CS PhD in Group of Moira Norrie
 - Politecnico di Milano (2008-2010) – Group of Stefano Ceri
 - Portland State University (2010-2012) – Group of David Maier
 - Experience in object databases, query processing, context-aware data management and Web engineering
- **Tutorial**
 - based on ETH Zurich’s lecture “Object-Oriented Databases” by Michael Grossniklaus and Moira C. Norrie → <http://www.globis.ethz.ch/education/oodb>
 - offered from 1995 to 2003 and from 2007 to now
 - 14 lectures of 90 minutes, 446 slides (<http://www.odbms.org>)

Agenda

09:00–10:30 Part 1: Introduction to Object Databases

- What is an object database?
- Object models: classes, inheritance, relationships, and collections
- Persistence models
- Query languages and facilities

10:30–11:00 Coffee Break

11:00–12:00 Part 2: Object Databases in Action

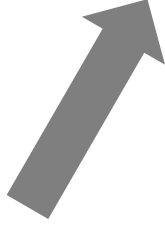
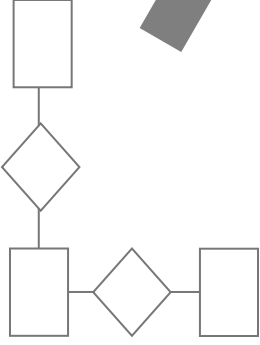
- Application examples: db4o, Objectivity/DB, ObjectStore, Versant
- Object database standards
- Object database architectures

PART 1

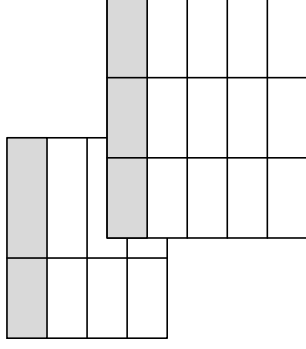
INTRODUCTION TO OBJECT DATABASES

Database Design

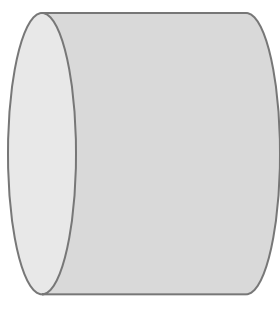
Conceptual Design



Implementation Design



Physical Design



Database Management Systems

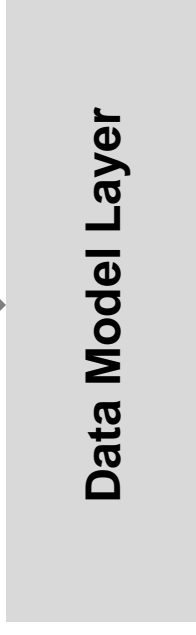
Conceptual modelling
Data access and
representation



E/R
SQL, JDBC, ODBC



Data and operational
semantics



Relational Model

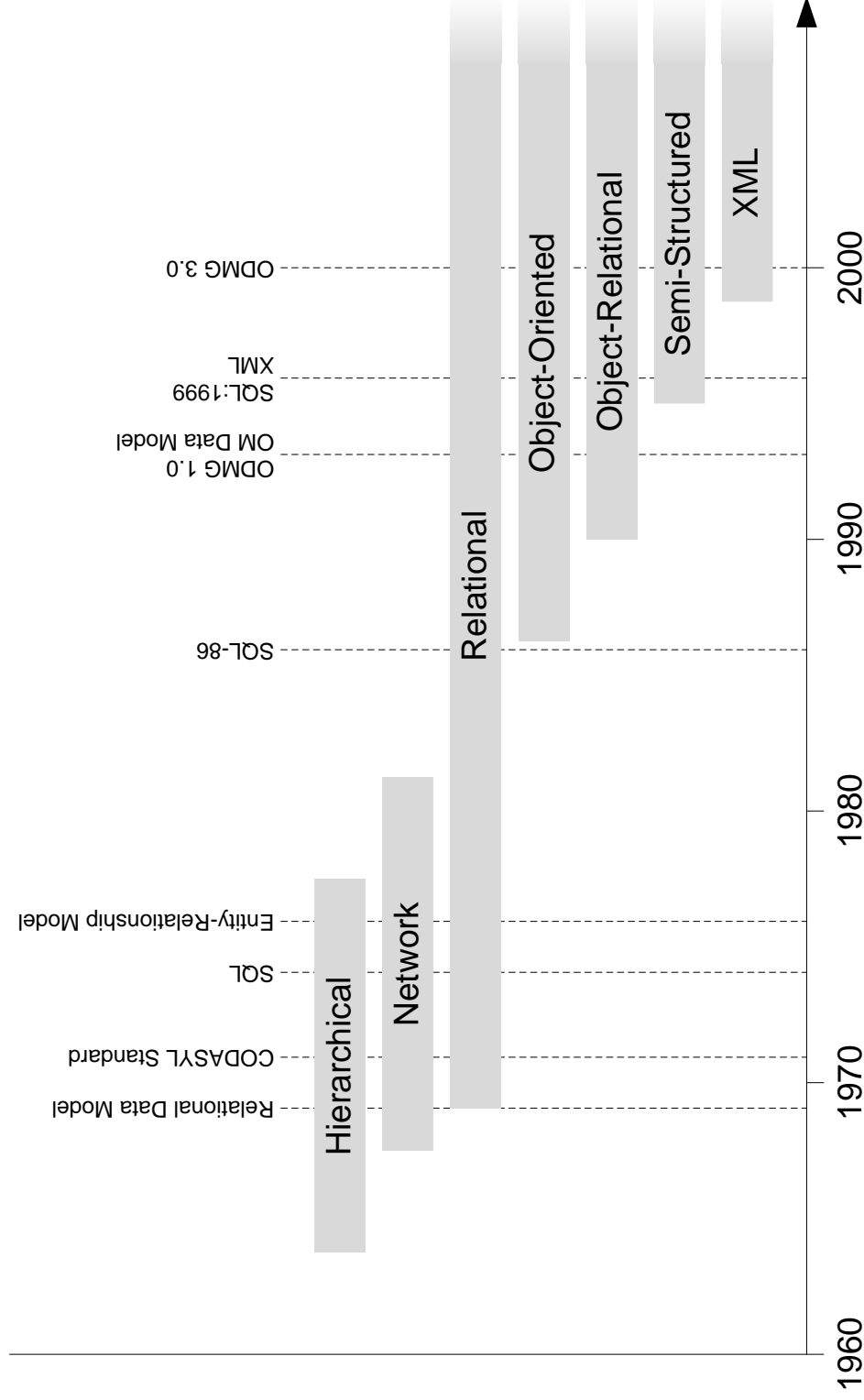


Persistence
ACID
Distribution



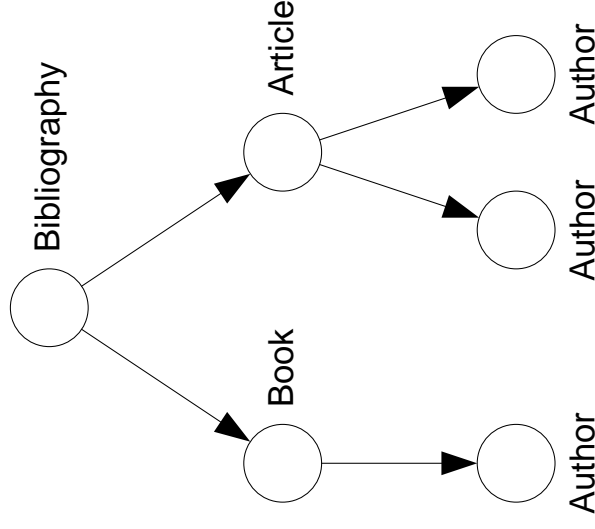
RDBMS

Evolution and History



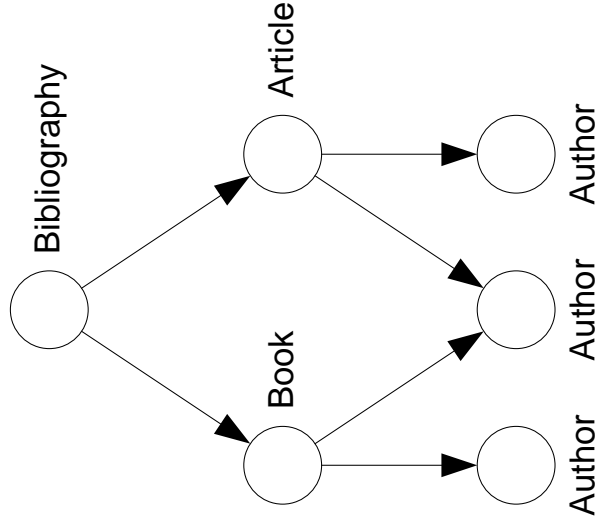
Hierarchical Databases

- Data organised in a tree
 - a parent can have many children
 - a child can have only one parent
- Records described by entity types
- 1:N (one-to-many) relationships
- Query by path navigation
- Examples
 - File system
 - LDAP
 - Windows Registry and Active Directory
 - XML documents and XQuery



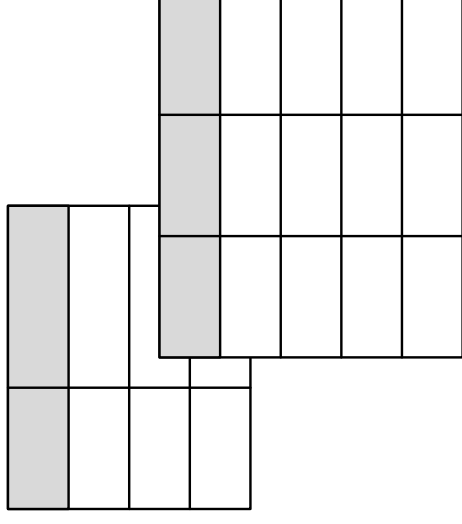
Network Databases

- Data organised in graph (lattice)
 - a parent can have many children
 - a child can have many parents
- Bachmann diagrams
- Record types define properties
- Set types defined relationships
 - parent-child, (double) linked list, ...
- Query by graph navigation
- Examples
 - CODASYL



Relational Databases

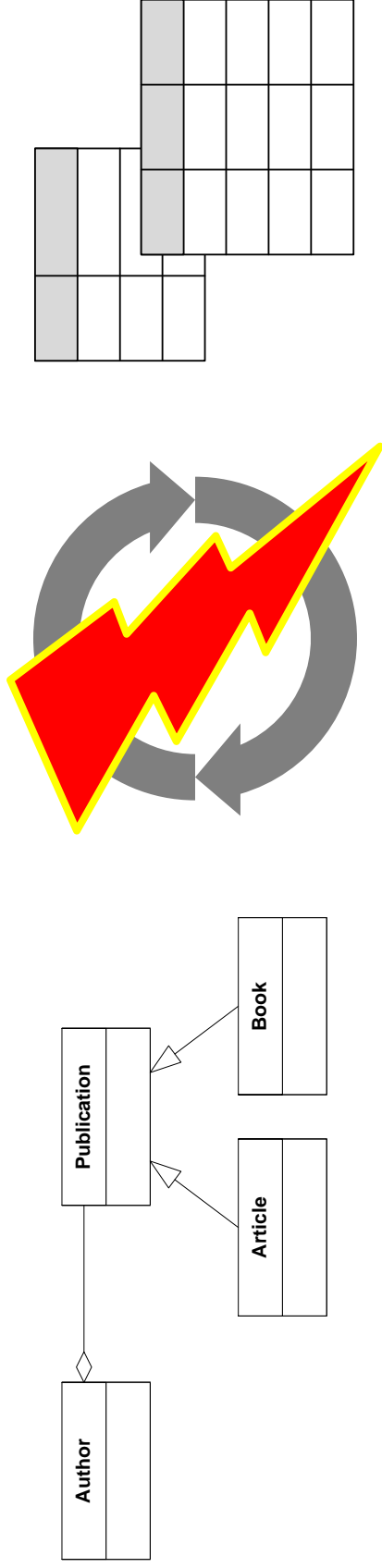
- Data organised as tuples in relations
- Link between data tuples
 - primary and foreign keys
- Relational algebra
 - project, select, join
- Relational normal forms
- Declarative language
 - data definition, consistency, manipulation and querying
- Examples
 - Oracle 11g, Microsoft SQL Server, IBM DB2
 - PostgreSQL, MySQL



Relational Databases

- Relational model is very simple
 - only basic concepts → references need to be simulated
 - restricted type system → no user-defined types
- Lack of semantic modelling
 - complex data, versioning, roles
- Little support for data and schema evolution
- Object-relational impedance mismatch

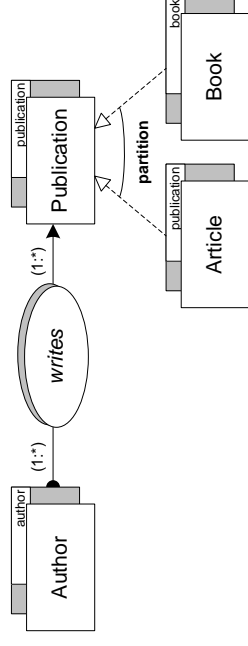
Object-Relational Impedance Mismatch



- Object-oriented application development and relational data management results in clash of two incompatible models
- Code to map between models is considerable overhead, costly and hard to maintain

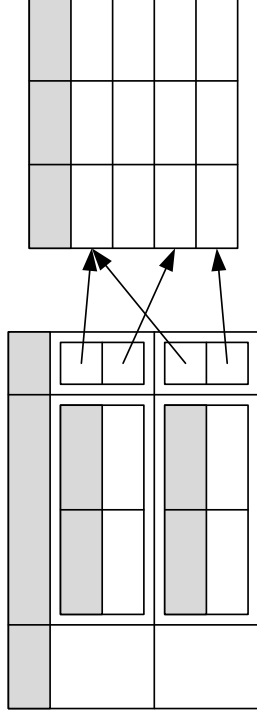
Object Databases

- Data represented as objects
 - object identity
 - attributes and methods
 - references, relationships, associations
- Extensible type hierarchy
 - user-defined types, abstract data types
 - single or multiple inheritance
 - overloading, overriding, late binding
- Declarative language for ad hoc purposes
- Binding for object-oriented programming language

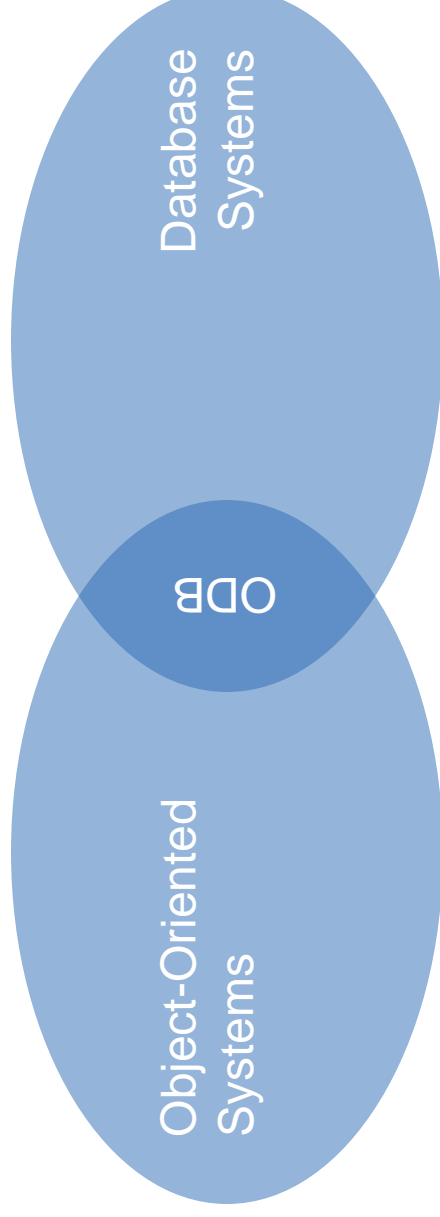


Object-Relational Databases

- Relational model extended
 - nested relations
 - references
 - sets
 - row types, abstract types
 - functions
- Declarative language extended
 - computationally complete
- Fundamental impedance mismatch remains
- Commingling of models



Object Databases



- Avoid object-relational impedance mismatch
- Provide a uniform data model
- Combine features and properties of
 - object-oriented systems and languages
 - database management systems

Defining Object Databases

- Diverse focus of object-oriented database systems
 - making object-oriented programming languages persistent
 - managing and storing object data
- Many attempts to define object-oriented databases
- The object-oriented database manifesto
 - 13 mandatory features
 - 5 optional characteristics
 - 4 open choices
- Manifesto aftermath
 - several refutations from the relational camp
 - several important properties not addressed

The Object-Oriented Database Manifesto

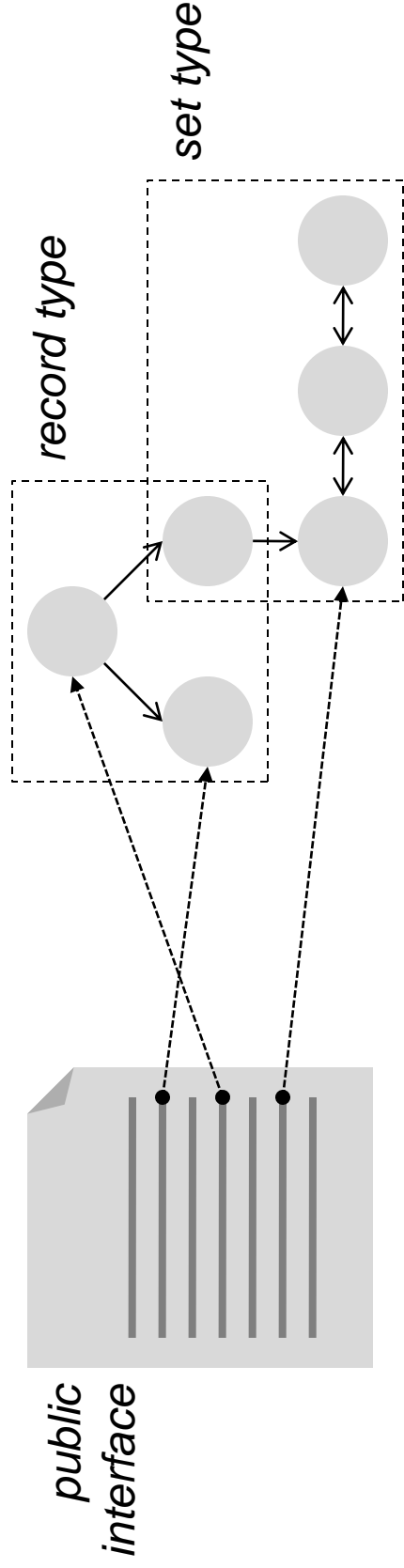
-
1. Complex objects
2. Object identity
3. Encapsulation
4. Types and classes
5. Type and class hierarchies
6. Overriding, overloading and late binding
7. Computational completeness
8. Extensibility
9. Persistence
10. Efficiency
11. Concurrency
12. Reliability
13. Declarative query language
- Object-oriented systems
- Database management systems

Objects

- **Complex objects**
 - complex object formed from simpler ones by constructors
 - record, set, bag, list and array complex object constructors
 - constructor orthogonality
- **Object identity and equality**
 - every object has unique and immutable object identifier (OID)
 - sharing of objects through references
 - two objects are identical if they have the same OID
 - two objects are equal if they have the same state
 - shallow and deep equality

Objects

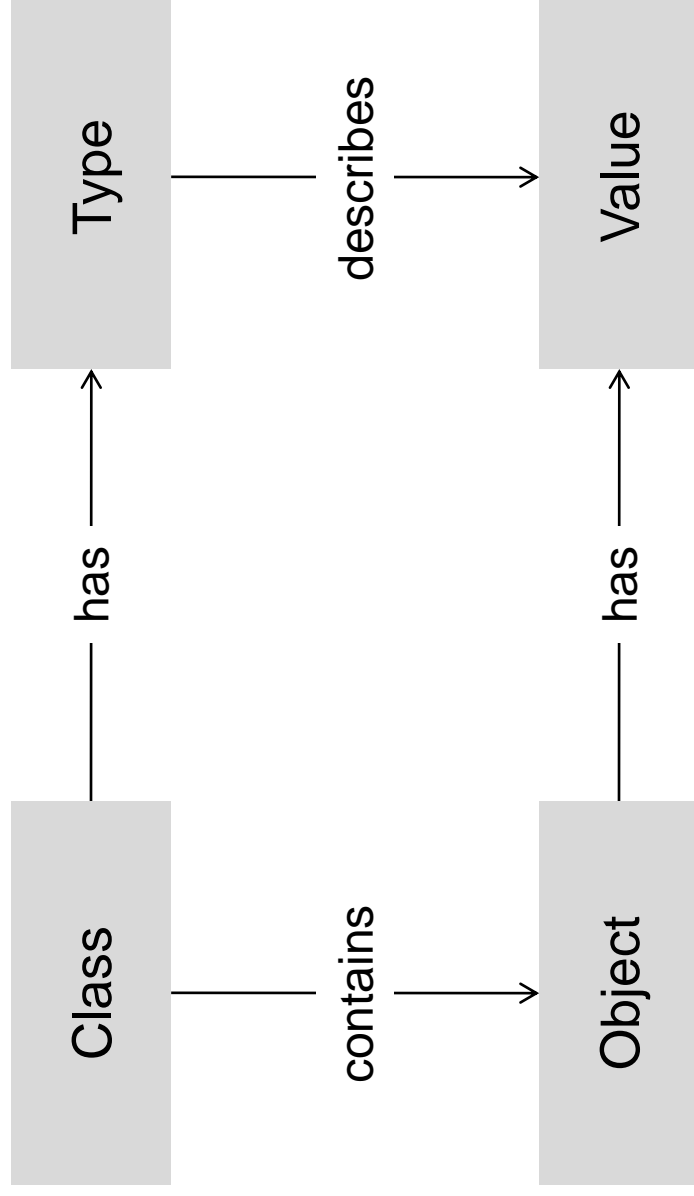
- Encapsulation
 - object consists of interface and implementation
 - interface defines signatures of public methods
 - implementation includes object data and methods
 - object state is only modified through public methods
 - object data structure may be exposed for declarative queries



Types and Classes

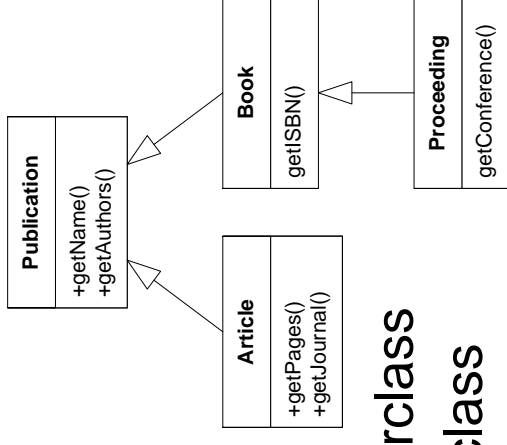
- **Data types**
 - definition of object properties
 - static part describes object structure
 - dynamic part describes object behaviour
 - separation of interface and implementation
 - used to check correctness of programs at compile time
- **Object classes**
 - container for objects of the same type
 - objects can be added and removed
 - used to create and manipulate objects at run time

Types and Classes



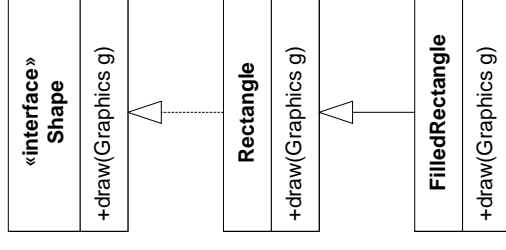
Generalisation Hierarchies

- **Advantages**
 - powerful modelling tool
 - guarantee semantic complexity
 - reuse of specification and implementation
- **Inheritance**
 - objects of subclass belong automatically to superclass
 - attributes and methods are inherited from superclass
 - subclass can introduce new attributes and methods
- **Migration between classes**
 - move objects between hierarchy levels
 - object specialisation (↓) and generalisation (↑)
 - class instance versus class member



Overriding, Overloading and Late Binding

- **Method overriding**
 - method is redefined in subtype
 - guarantees specialisation of methods
 - preserves uniform method interface
- **Method overloading**
 - effect caused by method overriding
 - various version of a method can exist in parallel
- **Late binding**
 - appropriate version of overloaded method selected at run time
 - also known as virtual method dispatching



Computational Completeness and Extensibility

- Computational completeness
 - requirement for the method implementation language
 - any computable function can be expressed
 - can be realised through connection with existing language
- Extensibility
 - database has a set of predefined types
 - developers can define new types according to requirements
 - no usage distinction between system and user types

Persistence

- Data has to outlive the program execution
 - implicit and explicit persistence
- Transparent persistence
 - persistence is orthogonal to type
 - any object may be persistent or transient
 - no need to write code for persistence
- Persistence strategies
 - **by inheritance**: persistence capabilities inherited from pre-defined persistent class
 - **by instantiation**: objects made persistent and get persistence capabilities upon instantiation
 - **by reachability**: objects made persistent if reachable from other persistent object

Efficiency, Concurrency Control and Recovery

- **Secondary storage management**
 - index management, data clustering and buffering
 - access path selection and query optimisation
- **Concurrency**
 - management of multiple users concurrently interacting
 - atomicity, consistency, isolation and durability
 - serialisability of operations
- **Reliability**
 - resiliency to user, software and hardware failures
 - transactions can be committed or aborted
 - restore previous coherent state of data
 - redoing and undoing of transactions
 - logging of operations

Declarative Query Language

- **High-level language**
 - express non-trivial queries concisely
 - text-based or graphical interface
 - declarative
- **Efficient execution**
 - possibility for query optimisation
- **Application independent**
 - work on any possible database
 - no need for additional methods on user-defined types

Optional Characteristics and Open Choices

- **Optional characteristics**
 - multiple inheritance
 - type checking and inference
 - distribution
 - design transactions, long transactions, nested transactions
 - versions
- **Open choices**
 - programming paradigm
 - representation system
 - type system
 - uniformity

Beyond the Manifesto

- Database administration utilities
- View definition and derived data
- Object roles
 - objects have roles in addition to types
 - roles can be gained and lost dynamically
- Database evolution
 - schema and data has to evolve gracefully over time
- Constraints
 - integrity, semantic and evolution constraints
 - definition, management and enforcement of constraints

PART 2 OBJECT DATABASES IN ACTION

Real-Life Object Databases

- Objectivity
 - Objectivity/DB for .NET (C#)
- Progress
 - ObjectStore PSE Pro for C++
- Versant
 - db4o for Java
 - Versant Object Database for Java
- ...and how they handle
 - persistence strategies
 - object model (i.e. relationships and collections)
 - queries

Company Profile: Objectivity

- Object-oriented database management system
 - developed since 1993 by Objectivity, Inc. (founded 1988)
 - current version 10.0
- Database core implemented in C++
- Front-end language support
 - C++, C#, Java, Smalltalk, Python, SQL++, and XML
- Platform Support
 - Windows, Linux, Solaris, HP-UX, IBM RS/6000, Altix
 - both 32 bit and 64 bit platforms are supported
- Cloud computing option available
 - based on the Amazon AWS EC² and other cloud computing platforms

Typical Customers: Objectivity/DB

- Telecommunications
 - Nortel, Ericsson
- Financial services
 - AWD, Cuna Mutual Group
- Medical systems
 - Dräger Medical, LMS Medical
- Process management
 - Emerson, Metso Automation, Nec, Siemens
- Security and defense
 - Northrop Grumman, Raytheon
- Energy
 - Furgo Jason, Schlumberger
- Information technology
 - WebLOQ, Ciena
- Research
 - NASA, CERN, SLAC, Los Alamos National Laboratory, Fermilab



Company Profile: Progress

- Both Java and C++ environments supported
- ObjectStore Personal Storage Edition (PSE) Pro
 - lightweight object database
 - large, single-user databases
 - small memory footprint (~500kB)
 - multithreaded
 - embedded systems, mobile computing and desktop applications
- ObjectStore Enterprise
 - high-performance, distributed, multi-user database
 - distributed, persistent, transactional object caching
 - clustering, online backup, replication, high availability
- Migration of applications to from PSE to Enterprise is easy

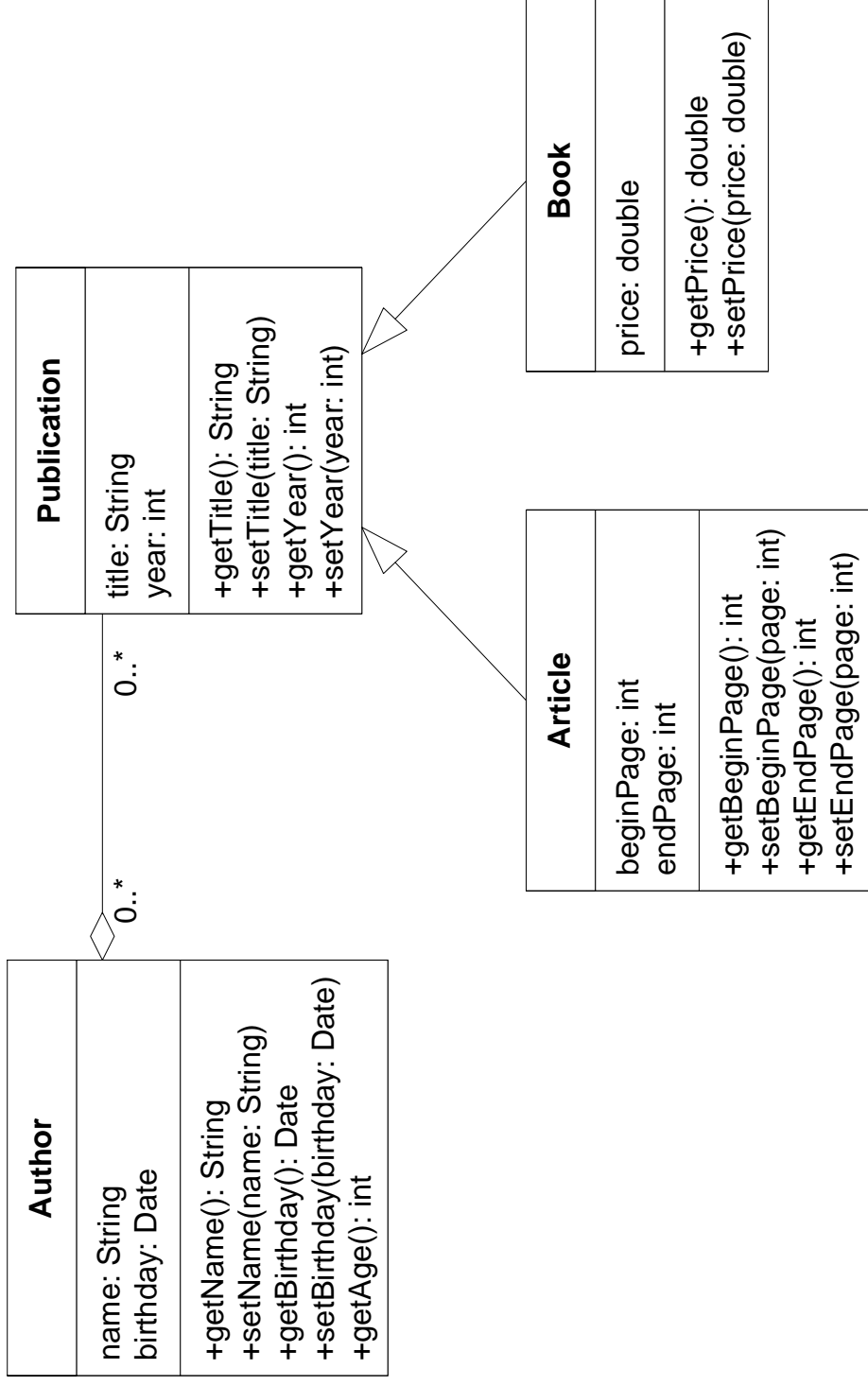
Company Profile: Versant

- Company founded in 1988
- Object Database Management Systems
 - highly scalable and distributed object-oriented architecture
 - patented caching algorithm
- Versant Object Database (C, C++, Java and .NET)
 - current version 8.0
 - available for many platforms
 - high availability option and tools
- db4o (Java and .NET)
 - current version 7.12
 - open source native object database for Java and .NET
 - small memory foot-print (single 2Mb library)

Typical Customers: Versant Object Database

- **Telecommunications**
 - Alcatel-Lucent, AT&T, Ericsson, Siemens, Nortel, France Telecom, Verizon, Samsung, Keymile, NEC
- **Defense**
 - BAE Systems, Lockheed Martin, FGM, Qinetiq, Raytheon, Northrup Grumman, Thales
- **Financial services**
 - BNP/Paribas, JP Morgan, AMEX, ING Barings, LCH Clearnet
- **Transportation**
 - British Airways, Sabre Group, Air France, GE Transportation, Qantas, Amadeus
- **Other**
 - Biomerieux, Factiva, EDS, Quantel, Oracle, Ovid, ESA

Example Class Hierarchy



Persistence Strategies

- Persistence by inheritance
 - persistence capabilities inherited from pre-defined persistent class
 - Versant (C++), Objectivity/DB (C++)
- Persistence by instantiation
 - objects made persistent and get persistence capabilities upon instantiation
 - ObjectStore (C++)
- Persistence by reachability
 - objects made persistent if reachable from other persistent object
 - O₂ (C++/Java), ObjectStore (Java), Versant (Java/Smalltalk), Objectivity/DB (Java/Smalltalk), db4o (Java/.NET), ODMG

Persistence by Inheritance (Objectivity/DB)

- Objectivity/DB uses .NET partial classes to separate application code and persistence support
- Persistence by inheritance
 - both partial classes inherit from **ReferenceableObject**
- Persistent support class
 - defines schema class and attributes
 - provides functionality to create and dispose objects
 - properties for attributes defined by the schema
 - maintains a proxy cache for each relationship
 - implements helper and utility functionality
- Application class contains user-defined code

Generated Domain Classes

```
// Author.cs
using System;
using Objectivity.Db;

public partial class Author : ReferenceableObject
{
    public Author(IStorable near,
        string name, DateTime birthdate) :
        base(near, schemaClass)
    {
        this.name = name;
        this.birthdate = birthdate;
    }

    public string Name
    {
        get { return this.name; }
        set { this.name = value; }
    }
    ...
}
```

```
// AuthorPD.cs
using System;
using Objectivity.Db;
using Objectivity.Db.Internal;
public partial class Author : ReferenceableObject
{
    private static SchemaClass schemaClass =
        new SchemaClass(
            "ObjectivityDemo.Author", 1000000);

    private static SchemaAttribute nameField =
        new SchemaAttribute(schemaClass,
            "name", AttributeKinds.String);

    private string name
    {
        get { return GetStringAttribute(nameField); }
        set { SetStringValue(nameField, value); }
    }
    ...
}
```


Persistence by Instantiation (ObjectStore)

- ObjectStore uses persistence by instantiation in C++
- Overloaded persistent **new** operator takes three arguments
 - allocation of the new object
 - type spec of the new object
 - optionally, how many objects are to be allocated
- Several options for object allocation
 - transiently on the heap
 - database
 - segment
 - cluster
 - next to another object

Note: *given the virtual memory architecture it is helpful to co-locate objects which are used together to achieve high performance designs and implementations*

- Persistence is orthogonal to the type of an object and one codebase can be used for transient and persistent objects

Creating Persistent Objects

- Objects in the database are created with the overloaded persistent **new** operator
- creating a single persistent object

```
os_database *db = os_database::open("publications.db", 0, 1);
Author *scheel = new(db, os_ts<Author>::get()) Author("Matthias Geel");
db->close();
```

- creating a persistent array of objects

```
void Author::setName(const char* name)
{
    delete [] _name;
    _name = 0;
    if (name) {
        int length = static_cast<int>(strlen(name)) + 1;
        _name = new(os_cluster::of(this), // allocate in the same cluster
                   os_typespec::get_char(), // get char type spec
                   length) char[length]; // create an array of size length

        strcpy_s(_name, length, name);
        _name[length] = 0;
    }
}
```

Persistence by Reachability (db4o)

```
// create a publication
Publication article = new Publication("Concepts for Content Management");

// create authors
Author michael = new Author("Michael Grossniklaus");
Author moira = new Author("Maira C. Norrie");

// assign authors to publication
article.addAuthor(michael);
article.addAuthor(moira);

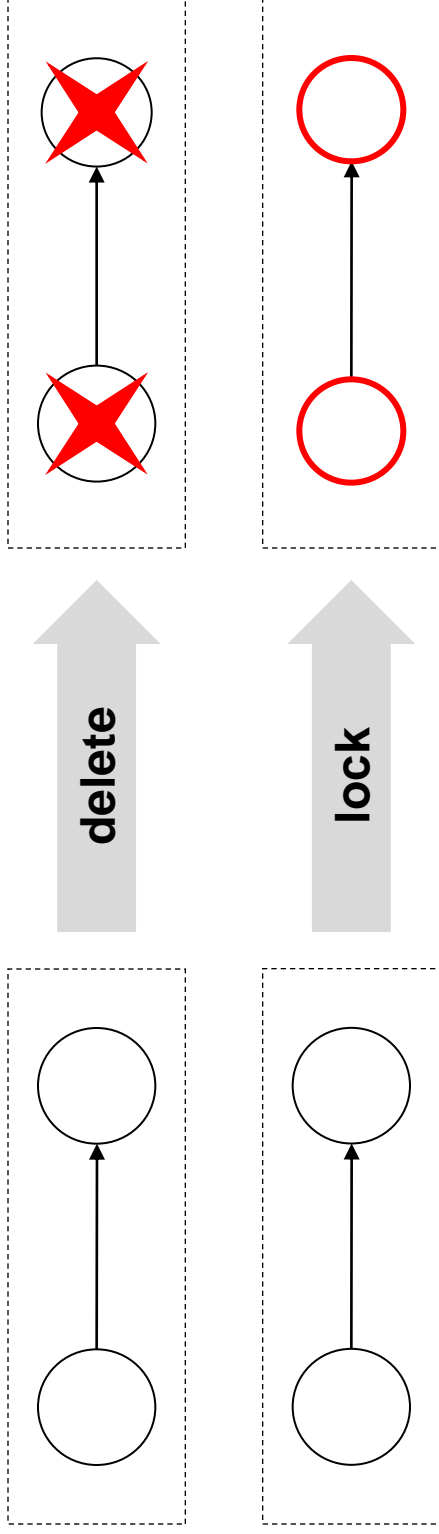
// store complex object
ObjectContainer db = Db4oEmbedded.openFile("test.db");
db.store(article);
```

- Store objects with method store of ObjectContainer
- Stores objects of arbitrary complexity
- Persistence by reachability

Relationships (Objectivity/DB)

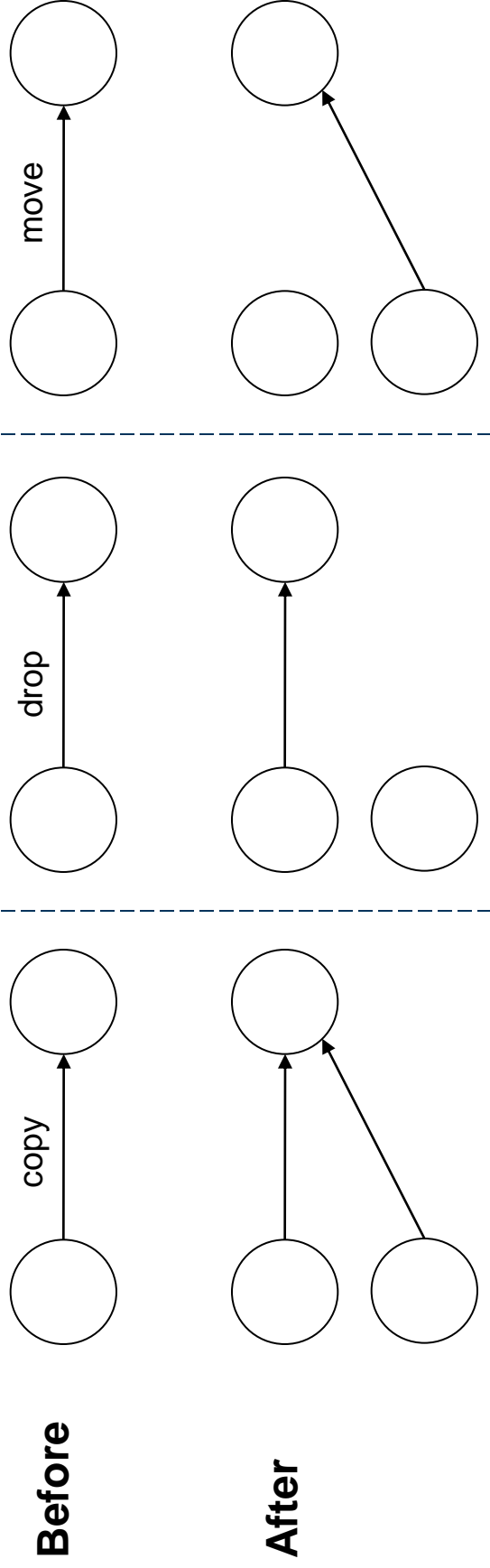
- Relationships between objects declared within classes
 - unary and binary relationships
 - to-one and to-many relationships
- Storage and management of relationships
 - non-inline: stored in object's system default association array
 - inline: stored in a dedicated attribute per relationship, as a reference to a single object (to-one) or to a variable-length array (to-many)
 - binary associations are represented as separate construct internally
- Consistency of relationships
 - referential integrity is maintained by the system
 - inverse relationship of a binary relationship is updated automatically when objects are added or removed
 - objects are removed from all relationships when deleted

Deletion and Lock Propagation



- Relationship can have semantics
- Deletion propagation
 - if an object is deleted all associated objects are also deleted
- Lock propagation
 - if an object is locked all associated objects are also locked

Copy and Versioning Behaviour



- Policies define what happens to object relationships when a copy or a version of an object is made
 - copy: old and new object are associated with the same objects
 - drop: only old object is associated with other objects
 - move: only new object is associated with other objects

Collections and Relationships (ObjectStore)

```
// Forward declaration
class Publication;

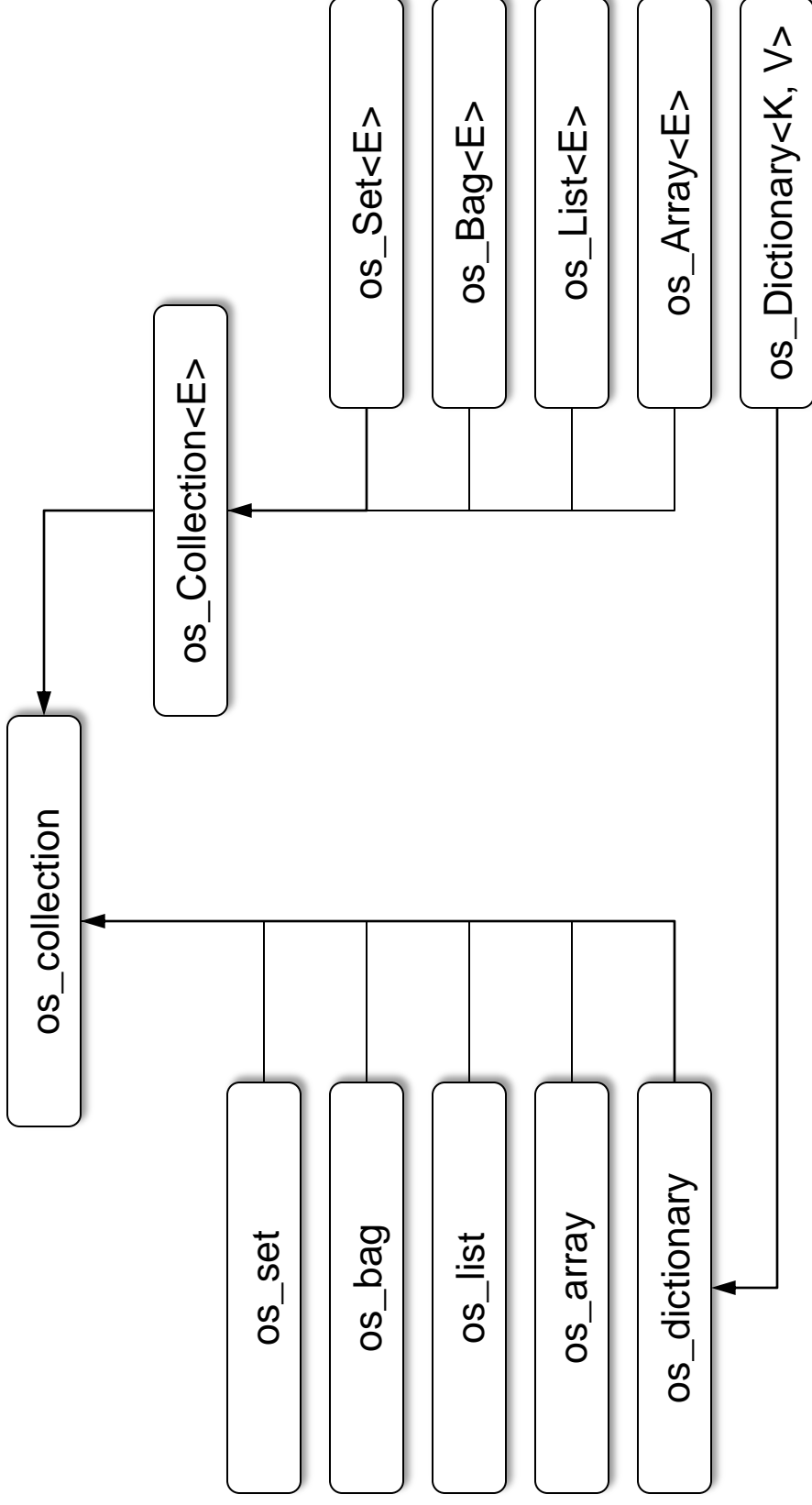
class Author
{
private:
    ...
    friend class Publication;
    os_Set<Publication*>* _authors;
public:
    ...
    void addPublication(Publication *p);
    void removePublication(Publication *p);
};
```

```
// Forward declaration
class Author;

class Publication
{
private:
    ...
    friend class Author;
    os_List<Author*>* _authoredBy;
public:
    ...
    void addAuthor(Author *a);
    void removeAuthor(Author *a);
};
```

- Relationships between classes modelled as collections
- ObjectStore collection facility
 - a library of non-templated and templated collection types
 - traversal, manipulation, and retrieval functionality
 - represented by class `os_collection`

Collection Hierarchy



Collections Example

- Creating a collection

```
Author::Author(const char *name)
{
    ...
    _authors = new(os_cluster::of(this),
                  os_Set<Publication*>::get_os_typespec()) os_Set<Publication*>();
}
```

- Accessing and manipulating a collection

```
void Author::addPublication(const Publication *p)
{
    _authors->insert((Publication*) p);
    os_List<Author*> *authoredBy = p->_authoredBy;
    authoredBy->insert(this);
}
```

- Deleting a collection

```
Author::~~Author(void)
{
    ...
    delete _authors;
    _authors = 0;
}
```

Cursors over Collections

```
const os_Set<Publication*>* Author::getPublications() const
{
    os_Set<Publication*> *result = new(
        os_database::get_transient_database(),
        os_Set<Publication*>::get_os_typespec()) os_Set<Publication*>();
    os_Cursor<Publication*> c(*_authors);
    for (Publication *publication = c.first(); c.more(); publication = c.next()) {
        result->insert(publication);
    }
    return result;
}
```

- Cursors are used to navigate and manipulate collections
 - represented by class `os_Cursor`
 - `first()` positions the cursor at the first element
 - `next()` moves the cursor to the next element
 - `more()` returns true if the cursor points to an element
- Cursor can be reused by rebinding it to another collection

Queries (db4o)

- db4o supports three query languages
- Query by Example
 - simple method based on prototype objects
 - selects exact matches only
- Native Queries
 - expressed in application programming language
 - type safe
 - transformed to SODA and optimised
- Simple Object Data Access (SODA)
 - query API based on the notion of a query graph
 - methods for descending graph and applying constraints

Query by Example

```
ObjectContainer db = Db4oEmbedded.openFile("test.db");

// get author "Moirra C. Norrie"
Author proto = new Author("Moirra C. Norrie");
ObjectSet<Author> authors = db.queryByExample(proto);
for (Author author: authors) {
    System.out.println(author.getName());
}

// get all publications
ObjectSet<Publication> publications = db.query(Publication.class);
for (Publication publication: publications) {
    System.out.println(publication.getTitle());
}
```

Native Queries

```
ObjectContainer db = Db4oEmbedded.openFile("test.db");

// find all publications after 1995
ObjectSet<Publication> publications = db.query(

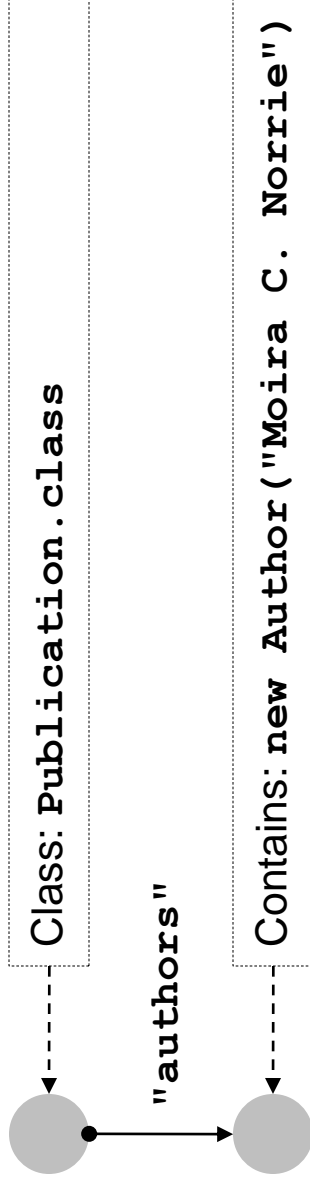
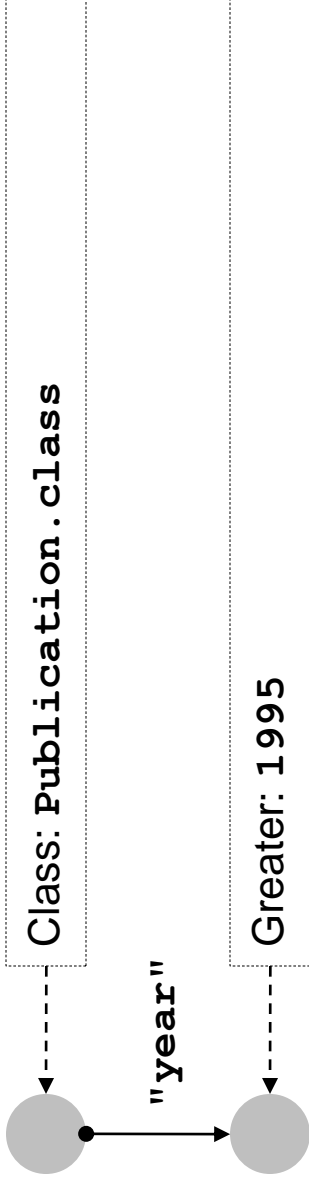
    new Predicate<Publication>() {
        public boolean match(Publication publication) {
            return publication.getYear() > 1995;
        }
    }
);
for (Publication publication: publications) {
    System.out.println(publication.getTitle());
}
```

SODA Queries

- Expressed using **Query** objects
 - **descend** adds or traverses a node in the query tree
 - **constrain** adds a constraint to a node in the query tree
 - **sortBy** sorts the result set
 - **orderAscending** and **orderDescending**
 - **execute** executes the query
- **Interface Constraint**
 - **greater** and **smaller** comparison modes
 - **identity**, **equal** and **like** evaluation modes
 - **and**, **or** and **not** operators
 - **startsWith** and **endsWith** string comparisons
 - **contains** to test collection membership

SODA Queries

- Find all publications published after 1995
- Find all publications of author "Moira C. Norrie"



SODA Queries

```
ObjectContainer db = Db4oEmbedded.openFile("test.db");

// find all publications after 1995
Query query = db.query();
query.constrain(Publication.class);
query.descend("year").constrain(Integer.valueOf(1995)).greater();
ObjectSet<Publication> publications = query.execute();
for (Publication publication : publications) {
    System.out.println(publication.getTitle());
}

// find all publications of author "Moirra C. Norrie"
Query query = db.query();
query.constrain(Publication.class);
Author proto = new Author("Moirra C. Norrie");
query.descend("authors").constrain(proto).contains();
ObjectSet<Publication> publications = query.execute();
for (Publication publication : publications) {
    System.out.println(publication.getTitle());
}
```


Queries (Versant)

- VQL 6
 - VQL 6 queries are a subset of OQL as specified by ODMG 2.0
 - no sorting, no extensions for new capabilities, limited API
 - as of Versant 7.0, VQL 6 queries are deprecated
- VQL 7
 - support for complex expressions
 - support for server-side sorting
 - improved indexing capabilities
- VQL queries are specified as a query string that is compiled, optimised and executed on the database server
- Queries can be parameterised
 - parameter starts with \$ followed by characters, digits or underscores
 - parameters are bound to values using the `bind()` method

VQL 7 Example

```
// create a new publication, assuming the Publication class is "p"  
Publication pub = new Publication("Web 2.0 Survey");  
  
// find authors Stefania Leone and Moira C. Norrie  
String queryString = "select selfoid from Author where name = $name";  
Query query = new Query(session, queryString);  
query.bind("name", "Stefania Leone");  
QueryResult result = query.execute();  
Object author = result.next();  
if (author != null) {  
    pub.addAuthor((Autor) author);  
}  
  
query.bind("name", "Moira C. Norrie");  
result = query.execute();  
author = result.next();  
if (author != null) {  
    pub.addAuthor((Author) author);  
}
```

VQL 7 Example

```
// find all publications by Moira C. Norrie and Michael Grossniklaus
String queryString =
    "select selfoid " +
    "from Publication " +
    "where Publication::authors subset_of $authors";

// precompile the query on the server
Query query = new Query(session, queryString);

// bind query to set of already existing author objects
query.bind("authors", new Object[] { moira, michael });
QueryResult result = query.execute();

// print out the names of the publications
for (Object pub = result.next; pub != null; ) {
    Publication p = (Publication) pub;
    System.out.println(p.getTitle());
}
```

Currently, collections can neither contain strings nor be parameters. Hence, this example is not possible.

Queries (Objectivity/DB)

- Language Integrated Queries (LINQ)
 - part of Microsoft's .NET framework (`System.Linq`)
 - introduced in Version 3.5
 - adds native query capabilities to .NET languages
- Standard query operators defined by class **Enumerable**
- Language extensions are translated into method calls by the .NET compiler
- LINQ providers
 - LINQ to Objects: querying of in-memory collections
 - LINQ to SQL: used to query Microsoft SQL Server databases
 - LINQ to XML: queries XML documents based on **XElement**
 - LINQ to DataSet: query databases using ADO.NET
 - many other providers, e.g. LINQ to db4o

LINQ Example

- Find all authors that are younger than 35 years of age and that have authored a publication prior to the year 2000

```
ICollection<Publication> publications = from p in this.db.OfType<Publication>()  
    where p.Year < 2000  
    select p;
```

```
ICollection<Author> result =  
    from a in this.db.OfType<Author>()  
    where a.GetAge() < 35 &&  
           publications.Intersect(a.GetPublications()).Count() > 0  
    select a;
```

- Code after translation by the .NET compiler

```
ICollection<Publication> publications =  
    this.db.Cast<Publication>().Where(p => p.Year < 2000);
```

```
ICollection<Author> result =  
    this.db.Cast<Author>().Where(a => a.GetAge() < 35 &&  
    publications.Intersect(a.GetPublications()).Count() > 0);
```

Object Data Management Group (ODMG)

- ODMG formed very early in development of OODBMS
- Informal standards body involving all major vendors
 - initiated in 1991 by Rick Cattell of SunSoft
 - initially ODMG comprised five people from OODBMS vendors
- Promote portability and interoperability across products
- Not developing a standard OODBMS product
 - products will vary in terms of languages, tools, interfaces, performance, etc.
 - products may be tailored to application domains, e.g. version management for Computer-Aided Software Engineering (CASE)

ODMG Standard

- Object Model
- Object Definition Language (ODL)
- Object Query Language (OQL)
- Language bindings
 - C++ Binding
 - Smalltalk Binding
 - Java Binding

ODMG Object Model

- Based on the OMG object model
- Basic modelling primitives
 - object unique identifier
 - literal no identifier
- Object state defined by the values carried for a set of properties, i.e. attributes or relationships
- Object behaviour defined by the set of operations that can be executed
- Objects and literals are categorised by their type which defines common properties and common behaviour

Object Query Language (OQL)

- Based on ODMG Object Model and SQL-92

```
select list of values  
from list of collections and typical members  
where condition
```

- collections can be extents or expressions that evaluate to collection
- names can be used to denote a typical member of a collection
- Path expressions to navigate complex objects
 - `person.spouse.address.street.name`
- Not computationally complete
 - rather simple to use query language
- No explicit update operators
 - but can invoke update operations on objects

Object Database Architectures

- RDBMS architectures are very similar
 - server centric, index-based, relational algebra execution engine
 - performance and scalability numbers vary by small percentages
- OODBMS architectures vary considerably and exhibit wildly different characteristics
 - performance and scalability numbers may vary by orders of magnitude
- OODBMS architectures and their impact on expectations of early adopters can be seen as **one** of the factors for the, initially, limited success of OODBMS
- It is important to consider application characteristics and understand which OODBMS architecture is best suited

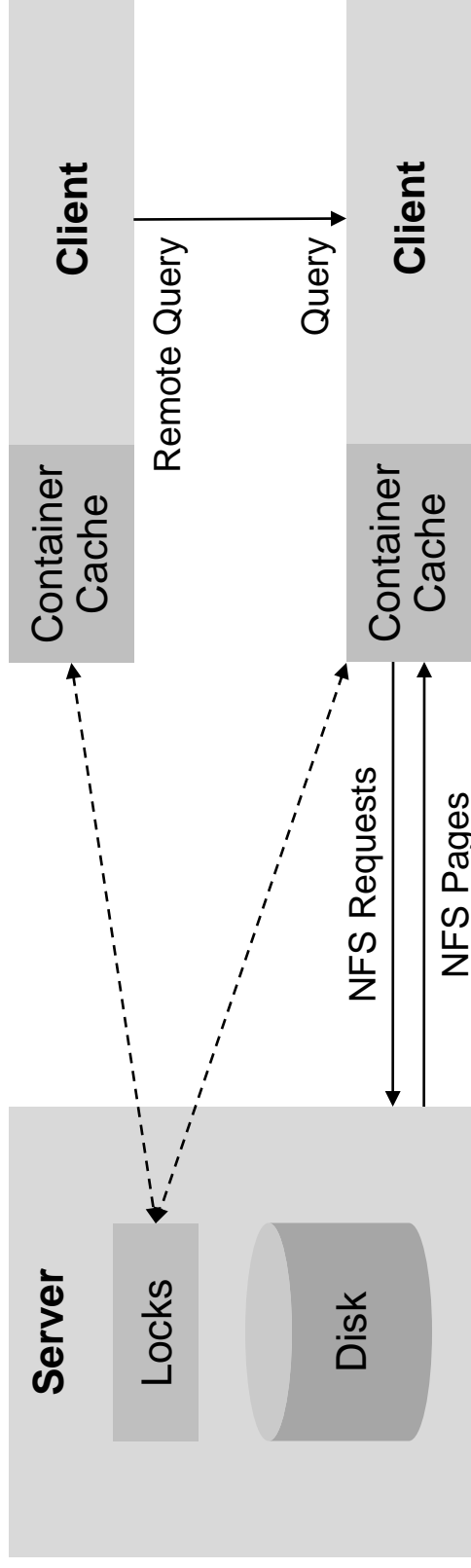
The Major Factors

- Key distinguishing implementation differences lead to vastly different runtime characteristics
- Primary areas impacting on performance include
 - core architecture
 - concurrency model
 - network model
 - query implementation
 - identity management
- Other feature functionality may also have an impact depending on application characteristics

Core Architecture

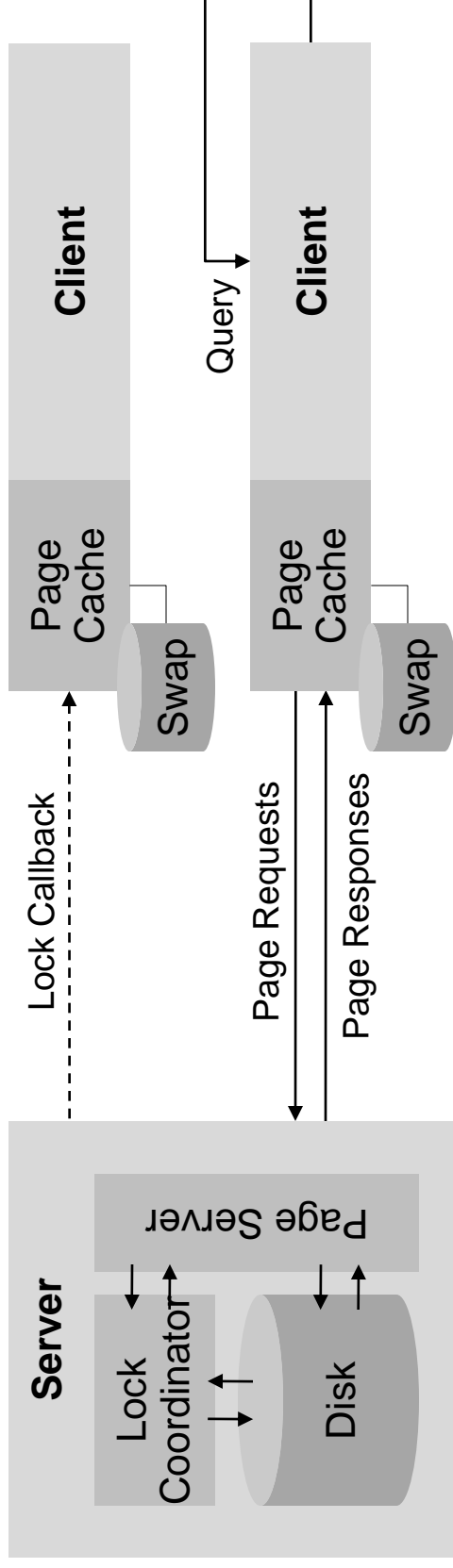
- Core architecture determines the following aspects
 - caching
 - query processing
 - transaction management
 - object life cycle management, i.e. tracking of new, dirty and deleted
- Three architectural variants are popular in OODBMS
 - container-based
 - page-based
 - object-based
- Name of the architecture reflects both
 - unit of transfer in network calls
 - lowest level of locking granularity

Container-Based Architecture



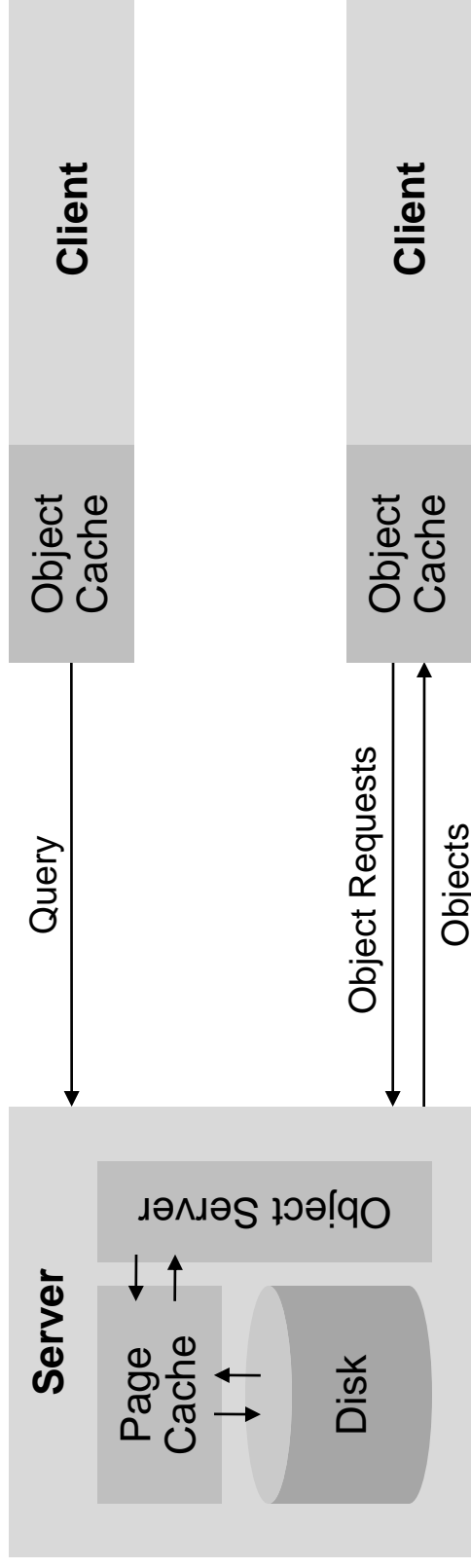
- Ships disk segments (containers) across the network
- Client libraries implement database functionality
 - container caching, query processing, object life cycle management and transactions
- All objects must reside inside a container
- Container model is layered over application domain model

Page-Based Architecture



- Ships pages of disk across the network
 - pages get address translated into virtual memory of an application
- Client libraries implement database functionality
 - container caching, query processing, object life cycle management and transactions
- Typically, object placement strategies are implemented

Object-Based Architecture



- Ships objects across the network
- Caching and behaviour in both client and server
 - **server**: page cache, indexes, locks, queries and transactions
 - **client libraries**: object caching, local locking and object life cycles
- No object placement strategies have to be implemented

Object Database Architectures Revealed

- **Objectivity/DB**
 - container-based architecture
 - physical identity
- **ObjectStore Enterprise**
 - page-based architecture (queries can be executed on the server)
 - physical identity
- **Versant Object Database**
 - object-based architecture
 - logical identity
- **db4o**
 - object-based architecture
 - physical identity

Review

- **Object database**
 - object-oriented programming language plus database
- **Integration avoids “impedance mismatch”**
- **Powerful model**
 - relationships, collections, inheritance
- **Large performance advantage**
 - working set can be cached in-process
 - intense use of inheritance in application model
 - database is reference intensive

Literature and Links

- M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, and S. Zdonik: **The Object-Oriented Database System Manifesto**, In: *Building an Object-Oriented Database System*, Morgan Kaufmann 1992
- db4o Open-Source Object Database
 - <http://www.db4o.com/>
- Objectivity/DB
 - <http://www.objectivity.com/>
- ObjectStore PSE Pro
 - <http://www.progress.com/>
- Versant Object Database
 - <http://www.versant.com/>

Object Databases Tutorial

Questions and Discussion

