Object-Oriented Databases
Commercial OODBMS: Objectivity/DB

- Objectivity/DB for .NET
- Logical Storage Model: Federated Databases
- Language Integrated Queries (LINQ)

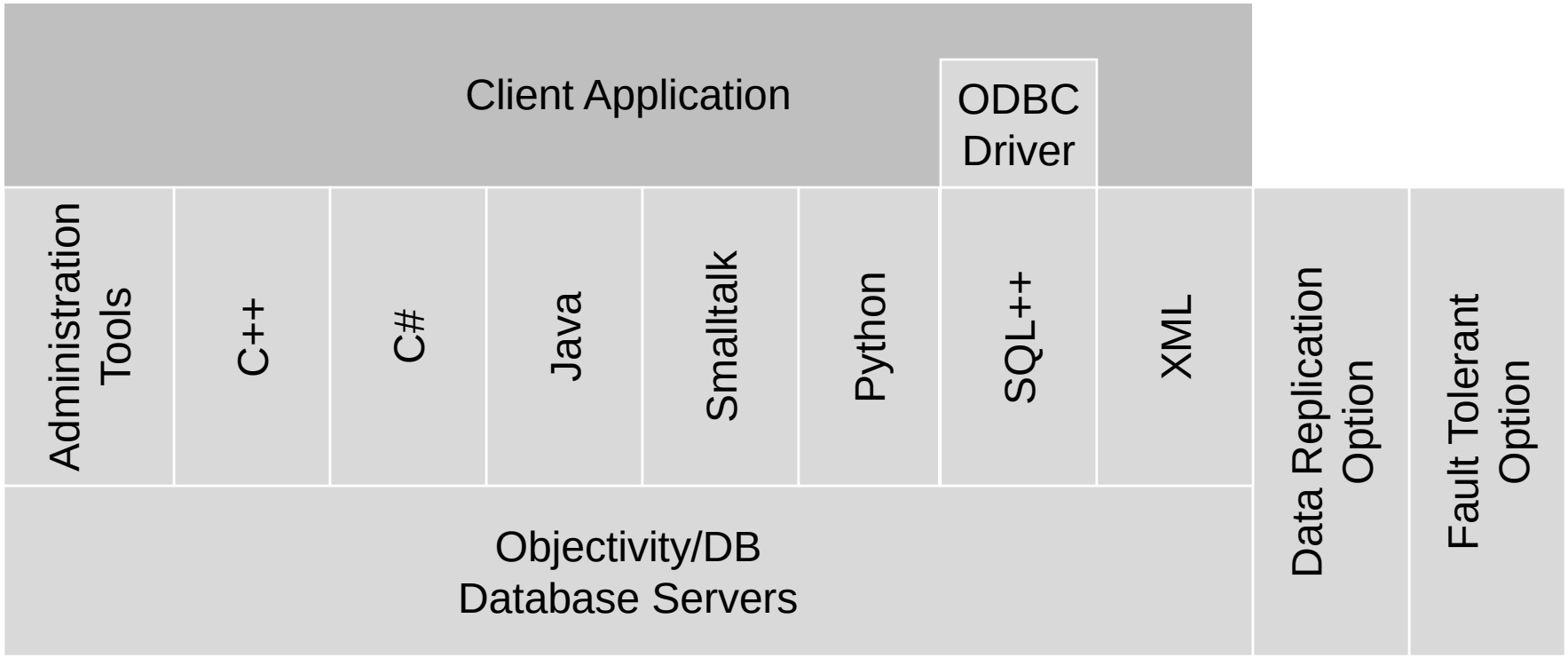
Objectivity/DB

- Object-oriented database management system
  - developed since 1993 by Objectivity, Inc. (founded 1988)
  - version 9.4 released in February 2008
  - version 10.0 to be released in early 2010
- 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 EC^2 and other cloud computing platforms

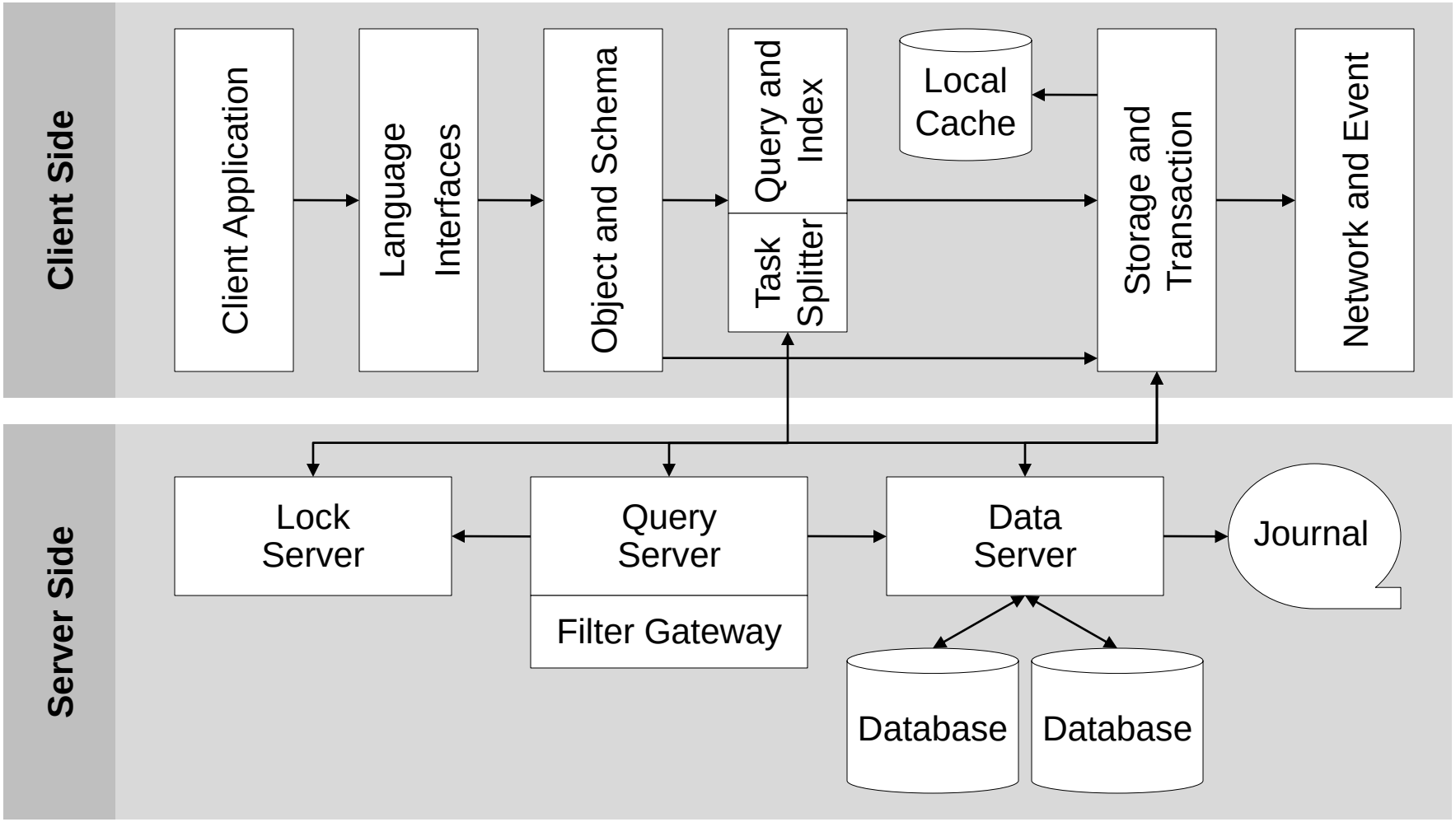
Typical Customers

- 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

Objectivity Product Family



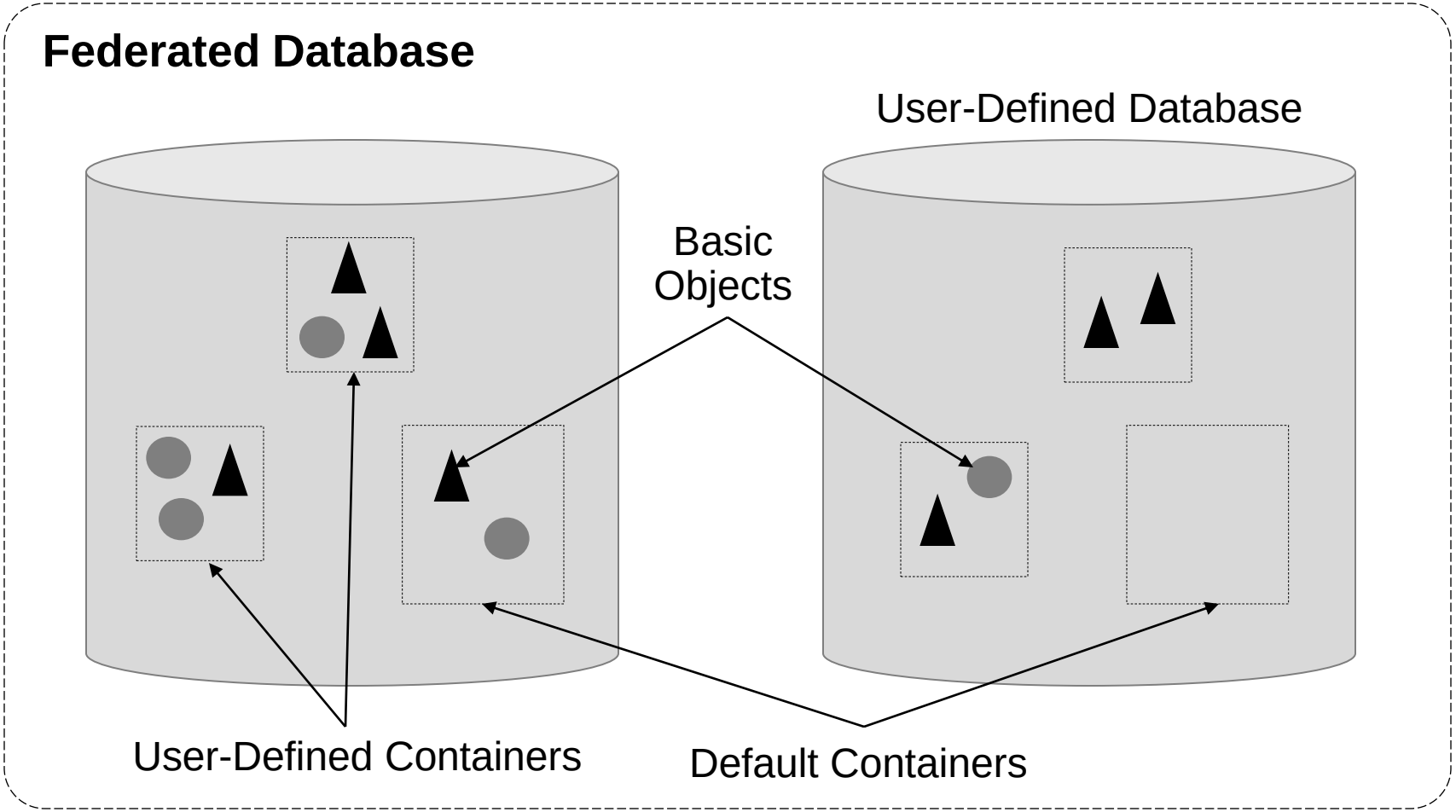
Architecture Overview



Architecture

- Scalability and availability
  - simple and distributed servers process
  - data, query, and lock servers
- Customisable parallel query engine (PQE)
  - task splitter aims queries at specific databases and containers
  - filters can run complex qualification methods
  - gateways can access other databases or search engines
  - replaceable components for smarter optimization
- Performance advantages
  - efficient storage and navigation of relationships
  - clustering and multi-dimensional indexing
  - scalable collections
  - client-side caching

Logical Storage Model

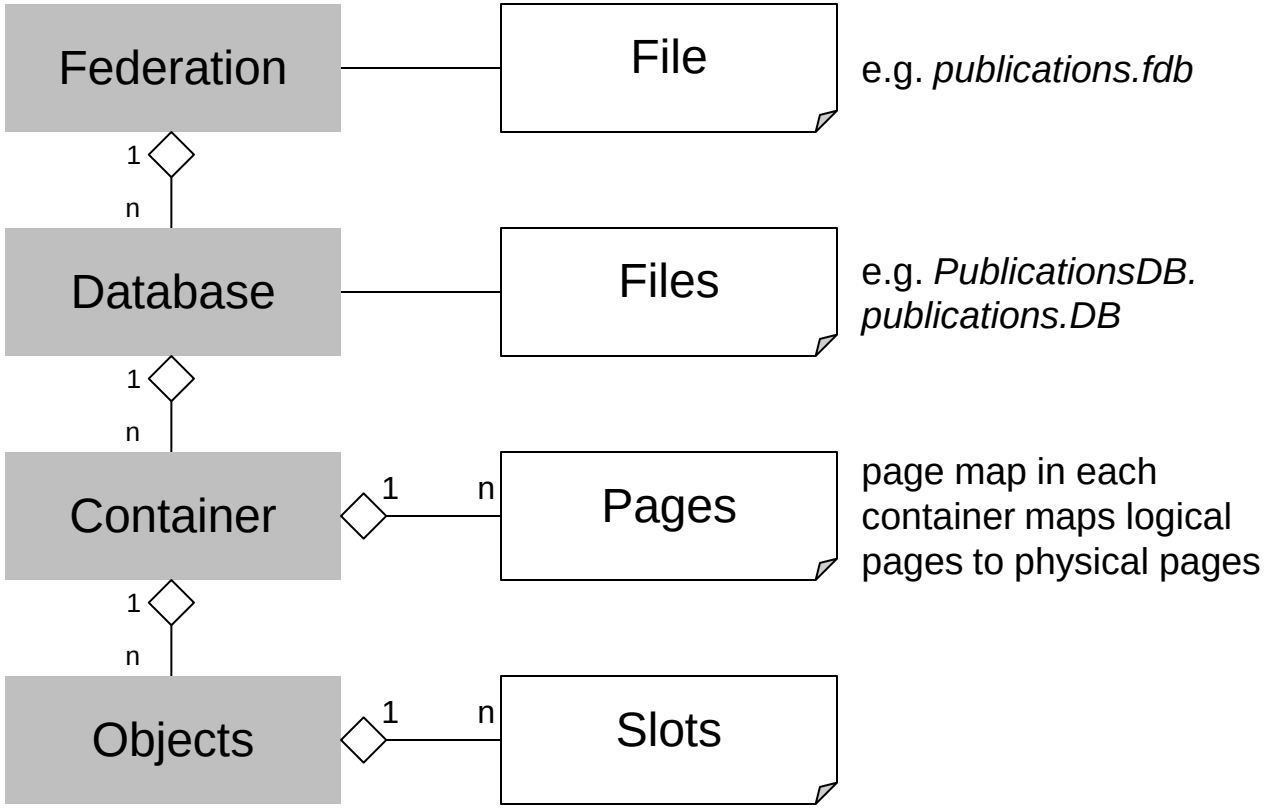


Physical Components of a Federation

Logical

- Federation
- Schema
- Database Catalog
- Container Catalog
- Page Map

Physical



Databases

- Databases are files
  - contain objects in containers
  - default name is *name*.*federation*.*DB*
  - up to 65530 databases per federation
  - databases hold up to 65530 containers
- Can be moved or copied to any disk or machine
  - databases can be close to clients using them for best performance
  - databases can be distributed to maximize parallelism
- Can be as large as the operating system will allow
- Can be marked read-only or taken off-line

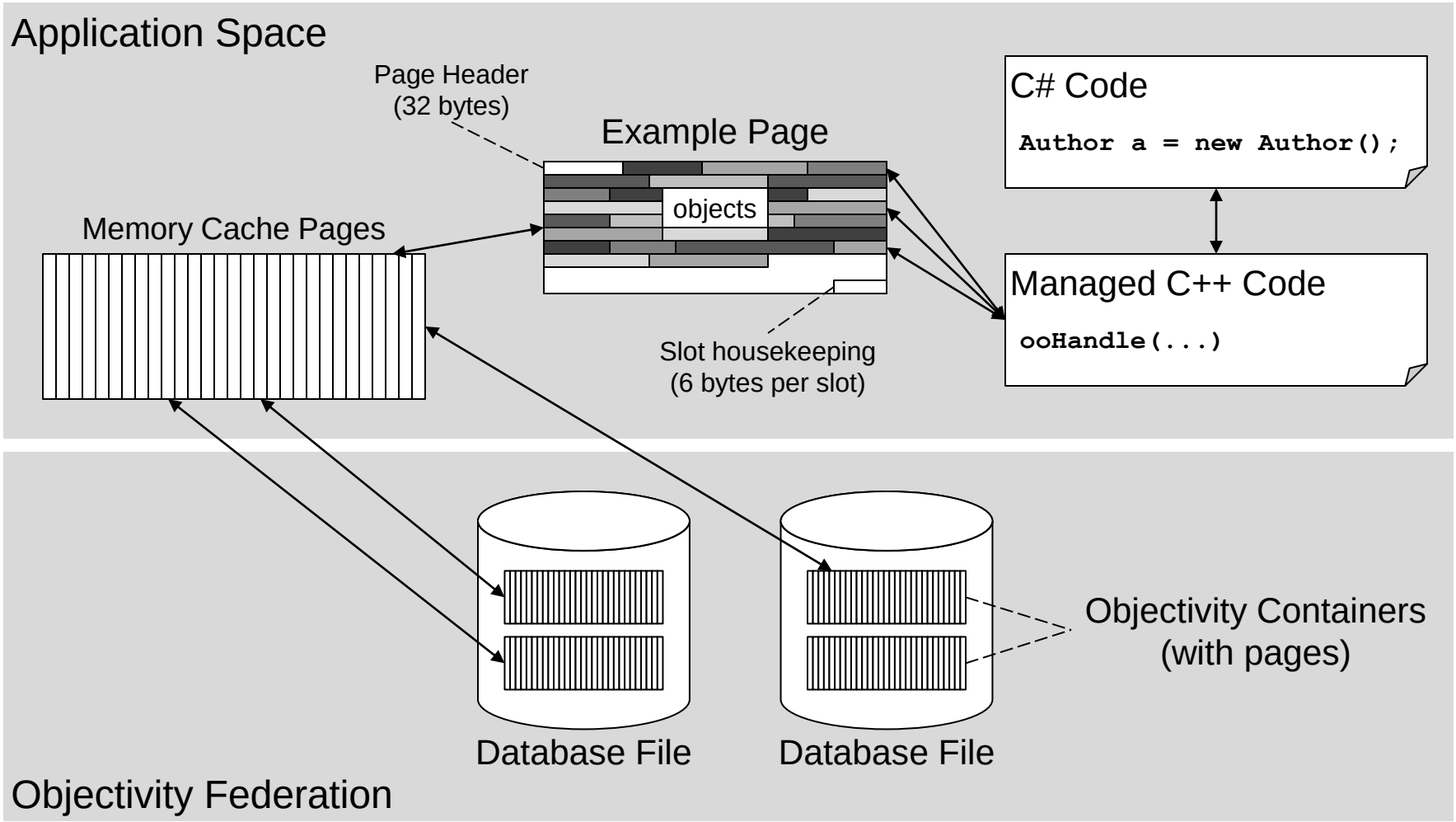
Containers

- Containers are collections of objects in a database
  - contained objects can be of any size or type
  - have a logical size in pages which can grow up to 65530 pages
- Very useful for logical partitions
  - by owner *Moira's books and articles*
  - by attribute *publications from 1990 to 2000*
  - by time *authors entered yesterday*
  - by edition e.g. a document with chapters, paragraphs, sentences, images etc.
- Unit of locking
  - one writer
  - multiple readers

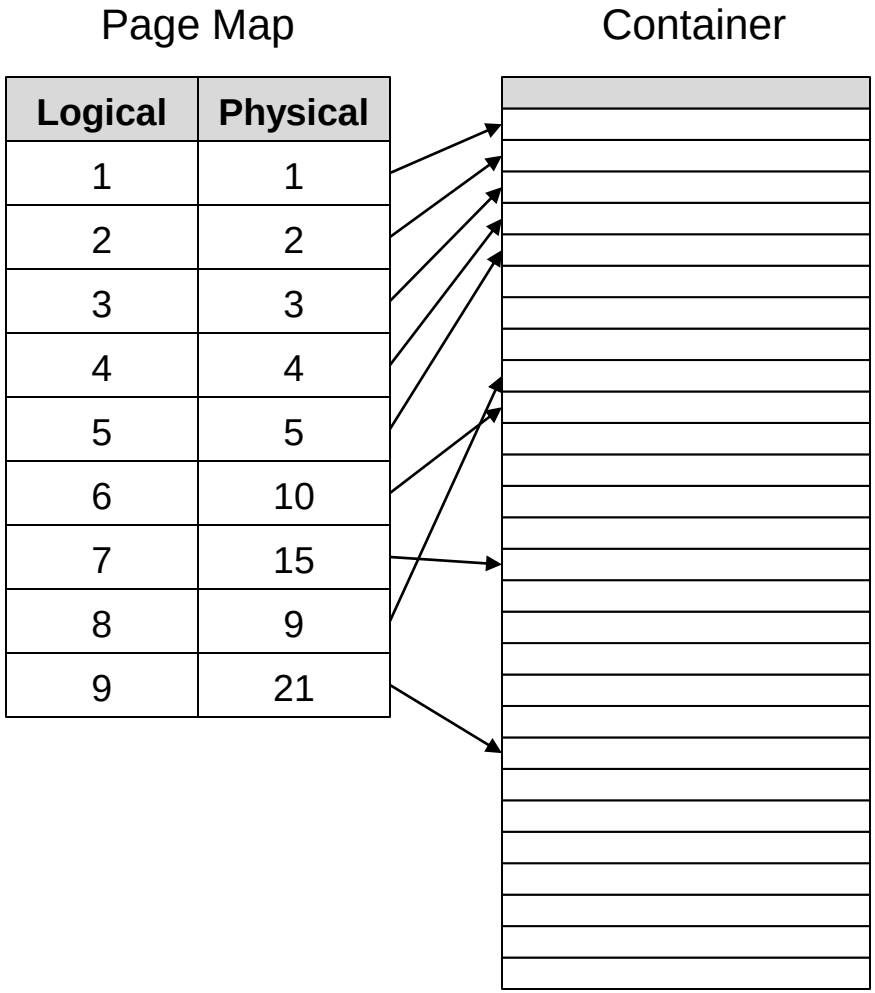
Pages

- Both logical and physical
  - logical page is permanent part of the object id (OID)
  - physical page is where in the file the page is put
- Default size per federation
  - Objectivity allows different page sizes for databases
  - default of 8K bytes, can be as large as 64K
  - each page contains 12 bytes of housekeeping
  - each object has an overhead of 14 bytes: 6 for the slot, 4 for the method pointer and 4 for the dynamic relationship slot
  - “large” objects or VArrays can occupy multiple physical pages
- Unit of transfer from disk or page server to cache

Pages


Application Space
Page Header
(32 bytes)
Example Page
objects
C# Code
Author a = new Author();
Memory Cache Pages
Managed C++ Code
ooHandle(...)
Slot housekeeping
(6 bytes per slot)
Objectivity Containers
(with pages)
Database File
Database File
Objectivity Federation

Logical and Physical Pages



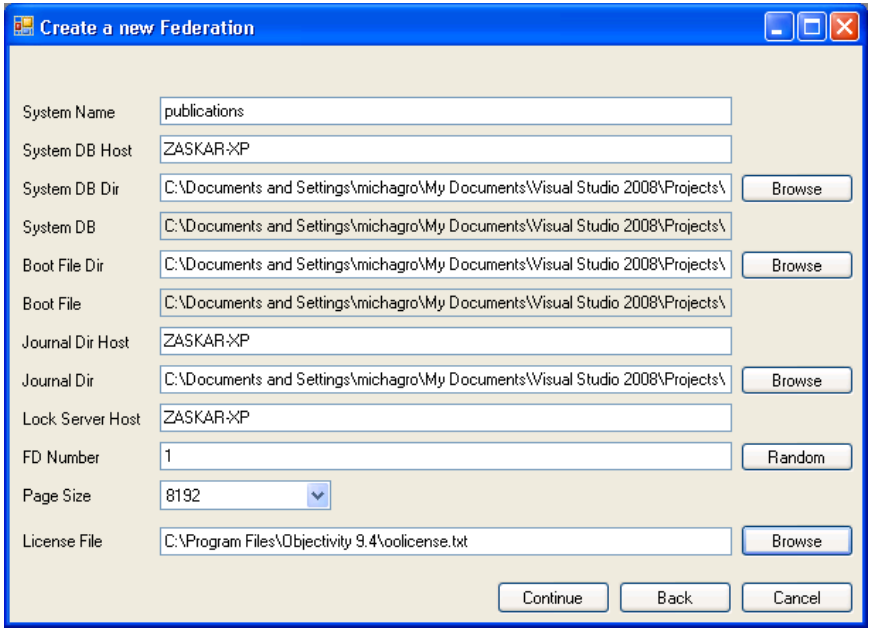
- When a page is modified
  - a new (defragmented) page is written to container, but the old page is kept
  - a journal file is written containing current page map
- When a transaction commits
  - the page map is updated and written to disk
  - old page is returned to “free page list”
  - journal file is truncated
  - lock is removed on the lock server
- If a transaction aborts
  - new page is returned to “free page list”
  - lock is removed from lock server

C# Development Process

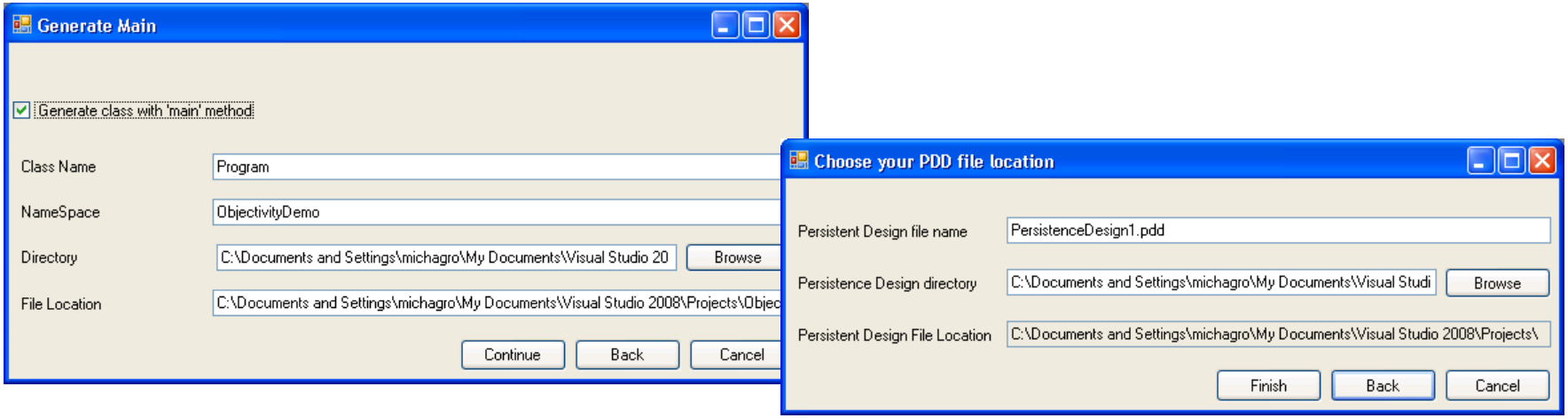
- Prerequisites
  - install Microsoft Visual Studio 2008
  - install Objectivity/DB for .NET
  - copy license file to the Objectivity/DB installation directory
  - check that lock server runs (run **`oocheckls`** from command prompt)
- Start Visual Studio 2008
- Create a new C# project
  - select “Empty Project” from “Templates”
- Add the Objy PDD Wizard to the project
  - right-click project in “Solution Explorer”
  - select “Add” ▸ “New Item...”
  - select “Objy PDD Wizard” from “Templates”

C# Development Process

- Create a new federation
- Set “System Name”
- “FD Number” identifies federation
  - can be left at default 1
  - needs to be randomised if the lock server serves multiple federations
- “Page Size”
  - default is 8K bytes
  - for reasons of performance default size matches disk page size of many modern operation systems
- Choose “License File”



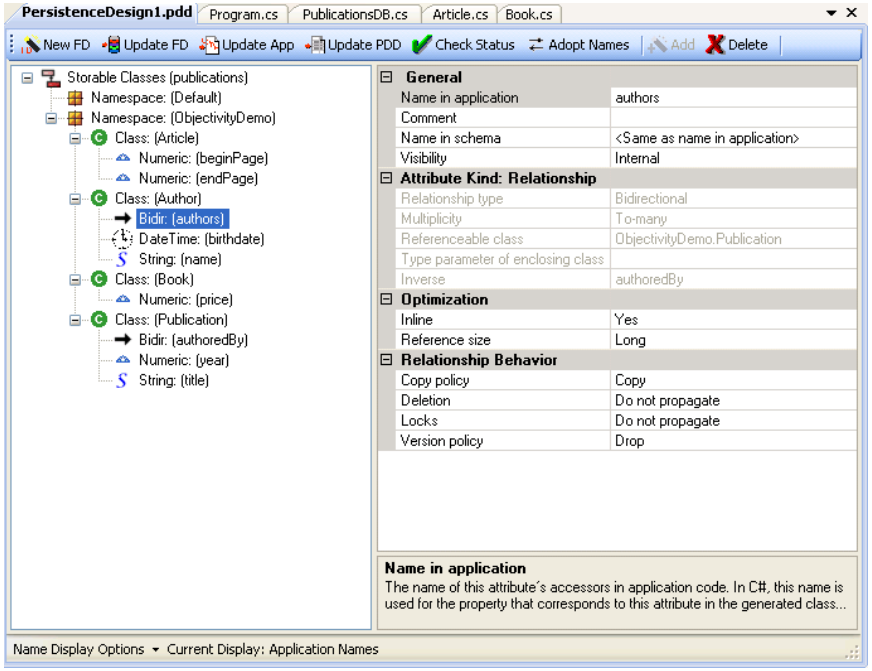
C# Development Process



- Generate main
  - set “Class Name” and “NameSpace”
- Choose your PDD file location
  - accept default values
- Wizard generates C# main and persistence design files
- Wizard adds Objectivity/DB assembly to the project

Persistence Designer

- Schema is defined using the persistence designer
- Schema is updated in federation by clicking “Update FD”
- Domain classes and application code is generated by clicking “Update App”
  - for each class a pair of files is generated, e.g. **`Author.cs`** and **`AuthorPD.cs`**
  - **`AuthorPD.cs`** should not be modified
  - **`Author.cs`** can be used to add getter and setters as well as derived methods if required



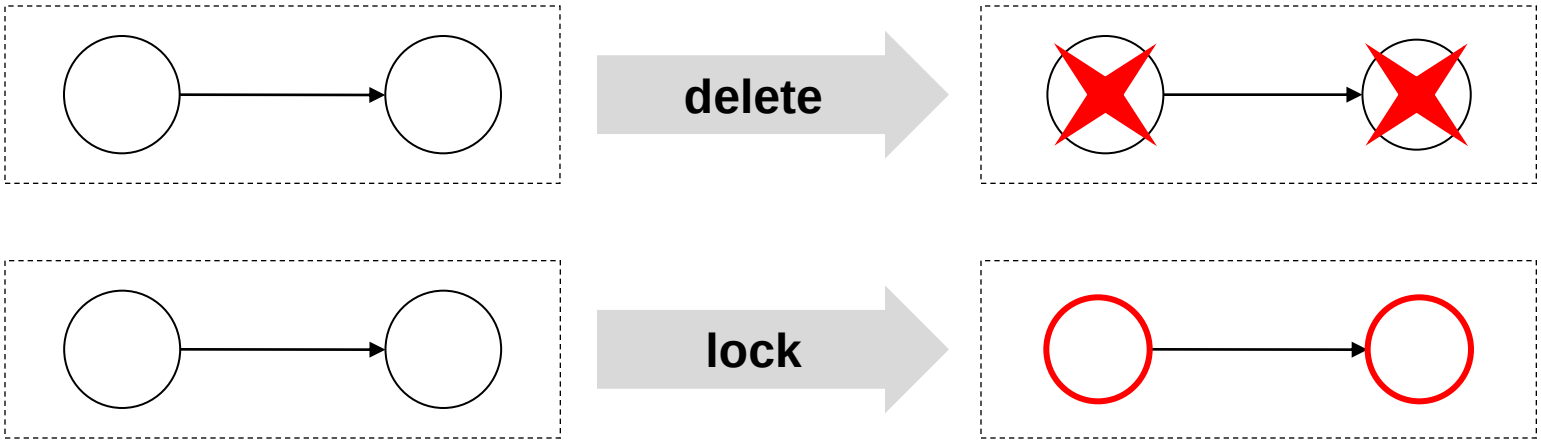
Persistent Object Model

- Language independent with persistence by inheritance
- Base types
  - numeric: sbyte, short, int, long, byte, ushort, uint, ulong, float, double
  - string: ASCII 8-bit, UTF8, UTF16
  - boolean
  - date/time
- Complex types
  - embedded: stored as part of the parent object
  - reference: parent object stores object identifier of referenced object
  - enumeration
  - fixed and variable-length arrays
- Relationships
- Collections

Relationships

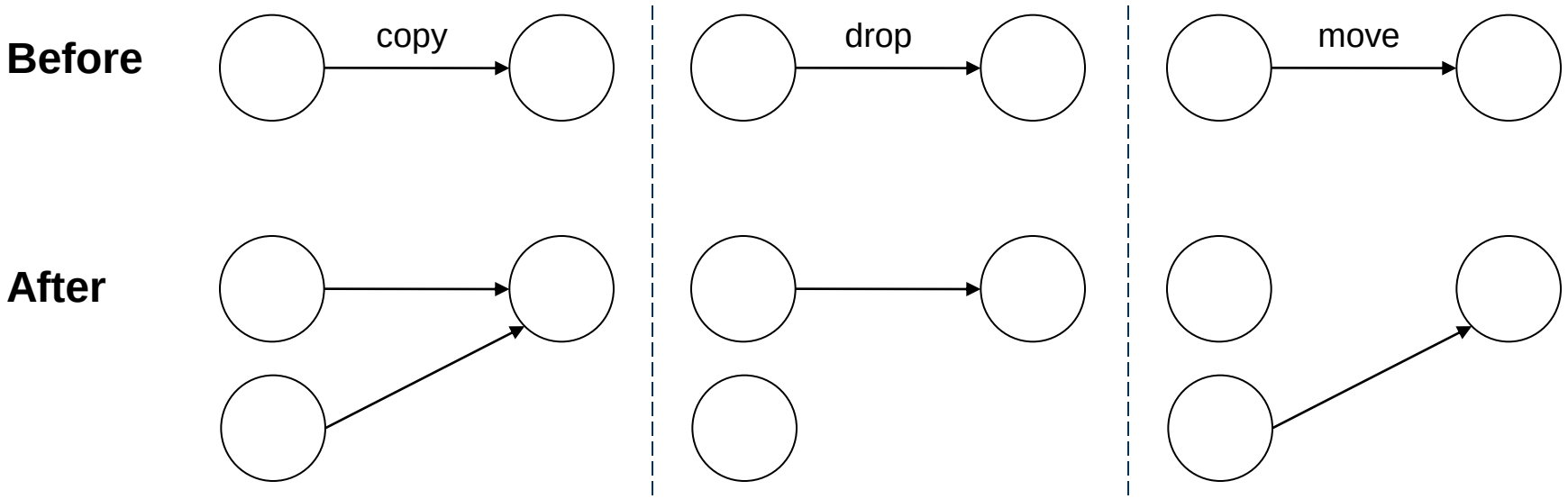
- Relationships between objects declared within classes
  - unary and binary relationships
  - to-one and to-many relationships
- Storage and management of relationships
  - non-inline: stored in object's system default association array
  - inline: stored in a dedicated attribute per relationship, as 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

Domain Classes

- 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 GetStringValue(nameField); }
  set { SetStringValue(nameField, value); }
 }

 ...

}
```

Connecting to Objectivity/DB

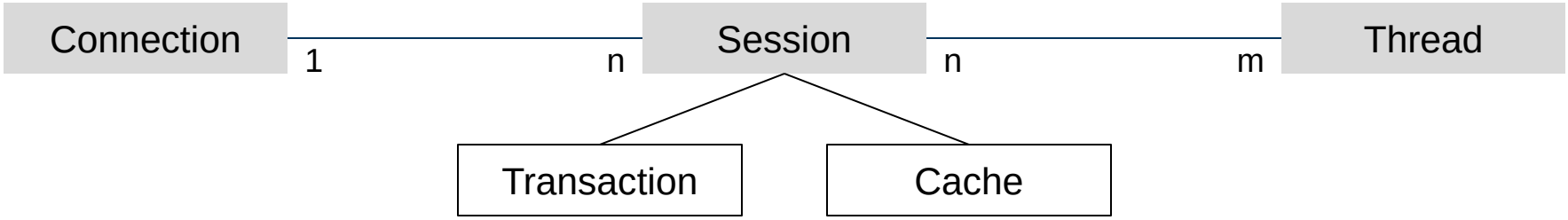
```
// Initialize Objectivity/DB
Objy.Startup();
// Establish a connection to the federation
Connection connection = Objy.GetConnection("publications.boot", true);


// Application code that interacts with the database


// Shutdown Objectivity/DB
Objy.Shutdown();
```

- Static functions for startup, shutdown and connection
  - initialisation is needed before calling any other database function
  - shutdown only possible after all database functions
  - federation can be auto-recovered when establishing connection
- Note that this API has been redesigned in Version 10.0

Connections, Sessions and Threads



- One connection to federation per application process
- Sessions
  - manage resources, i.e. cache and transaction state
  - one or many per thread, one shared by many threads
- Cache
  - remains intact through commits and checkpoints
  - is flushed if transaction is aborted
  - if updated object exceed the cache size, overflow pages are pre-written to disk

Interacting with Objectivity/DB

```
Objy.Startup();
Connection connection = Objy.GetConnection("publications.boot", true);

// Create a session
Session session = connection.CreateSession("main");
// Start a transaction
session.BeginTransaction(OpenMode.Update);
// Get the federation
Federation fd = session.Federation;
// Lookup or create the publications database
Database db;
if (fd.HasDatabase("PublicationsDB"))
{
   db = fd.LookUpDatabase("PublicationsDB");
}
else
{
   db = new Database(fd, "PublicationsDB");
}

...

session.CommitTransaction();

Objy.Shutdown();
```

Working with Containers and Objects

```
Container authors;
if (db.HasContainer("Authors"))
{
   authors = fd.LookUpContainer("Authors");
}
else
{
   authors = new Container(db, "Authors");
}

// Create new objects
Author michael = new Author(authors, "Michael Grossniklaus", new DateTime(1976, 6, 22));
Author moira = new Author(michael, "Moira Norrie");
Publication icoodb = new Book(db, "Proceedings of ICOODB 2009", 2009);

// Update objects
moira.Name = "Moira C. Norrie";
icoodb.AddAuthor(moira);
icoodb.AddAuthor(michael);

// Access objects
Console.WriteLine("{0} is {1} years old.", michael.Name, michael.GetAge());

// Delete object
icoodb.Delete();
```

Persistent Collections

- Built-in persistent collections provide sets, lists and maps
  - automatically persistent when created
  - conform to the **`System.Collections.Generic`** interface
- Type of persistent collections
  - ordered vs. unordered collections
  - scalable vs. non-scalable collections

	Scalable	Non-Scalable
Ordered	**`TreeListX<T>`** **`TreeMapX<K,V>`** **`TreeSetX<T>`**	
Unordered	**`HashMapX<T>`** **`HashSetX<T>`**	**`Map<T>`**

Iterators

- An iterator is a transient object that provides access to persistent objects of a given class that meet certain criteria
  - scope: collection, container, database or federation
  - criteria: PQL predicate given as a string
- Not an efficient method for looking up objects
  - predicates are evaluated on client, unless index is available
  - can be improved with indexes and scoping
- Result is not built entirely before it is returned
  - clients can start process result stream instead of getting blocked
  - sorting of result is not possible
- Each iterator class is a subclass of the corresponding object handle class

Iterator Example

```
// Print all books in the database
Iterator<Book> iBooks = new Iterator<Book>(db);
Book book;
while ((book = iBbooks.Next()) != null)
{
   Console.WriteLine("Book '{0}' was published in {1}", book.Name, book.Year);
}


// Print all authors that have a name starting with M
Iterator<Author> iAuthors = new Iterator<Author>(authors, "name =~ \"M.*\"");
Author author;
while ((author = iAuthors.Next()) != null)
{
   Console.WriteLine("{0} has published {1} publications", author.Name,
      author.GetPublicationCount());
}
```

Scope Names

```
// Assigning a scope name
HashSetX<Book> books = new HashSetX<Book>(db);
db.NameObject(books, "books");

// Looking up a scope name
HashSetX<Book> allBooks =
   (HashSetX<Book>)(HashSetX<IReferenceableObject>) db.LookUpObject("books");

// Unassigning a scope name
db.UnnameObject(allBooks);
```

- Scope names generalise the concept of database roots
- Objects can be assigned a unique name within each level of the storage hierarchy (scope)
  - container
  - database
  - federation

Retrieving Objects

- Scope names
- Creating and following links
- Individual and group lookup of persistent objects
  - through keys and iterators
- Parallel query
  - Parallel Query Engine (Objectivity/PQE)
  - divides the query scope among a number of query servers
- Content-based filtering
  - predicate-query language supporting primitive types and strings
  - used for predicate scans in group lookups
  - used when following a to-many relationship
  - used to find destination objects in parallel queries

LINQ Overview

- Language Integrated Queries
  - 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);
```

Literature

- Objectivity/DB
  - http://www.objectivity.com/

Next Week
The OM Data Model

- Multiple Inheritance, Instantiation and Classification
- Collections and Associations
- Cardinality, Classification and Evolution Constraints