Object-Oriented Databases
Commercial OODBMS: Part 2

- Versant Object Database for Java
- OODBMS Architectures, Revisited and Defended

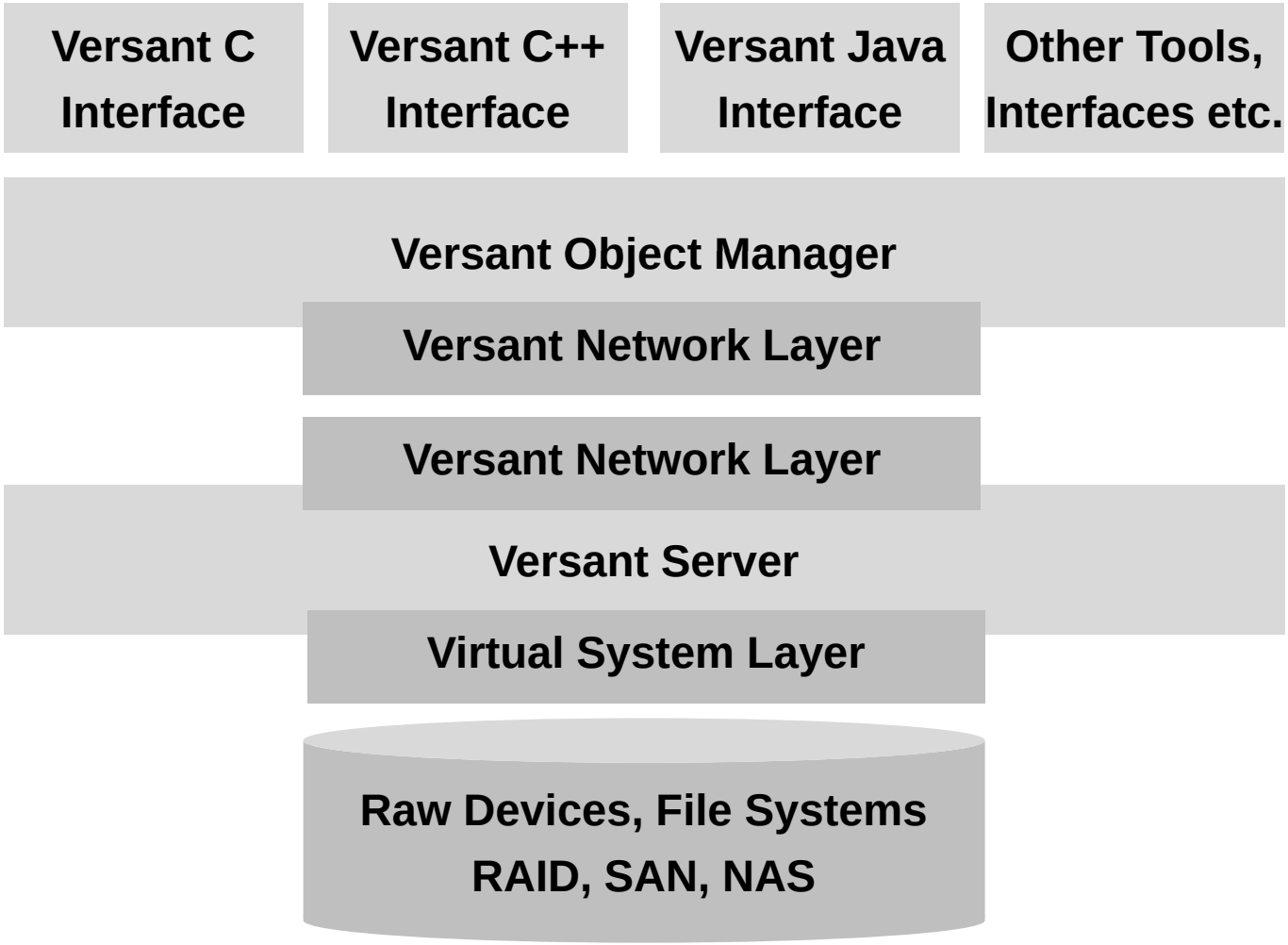
Versant

- Company founded in 1988
- Object Database Management Systems
  - highly scalable and distributed object-oriented architecture
  - patented caching algorithm
- Versant Object Database (C, C++ and Java)
  - market leader in object databases
  - current version 7.0.1.3
  - available for many platforms
  - high availability option and tools
- Versant FastObjects .NET (Microsoft .NET Framework 2.0)
  - taken over from the merger with Poet in 2004
  - current version 10.0
  - 5.5 MB memory footprint

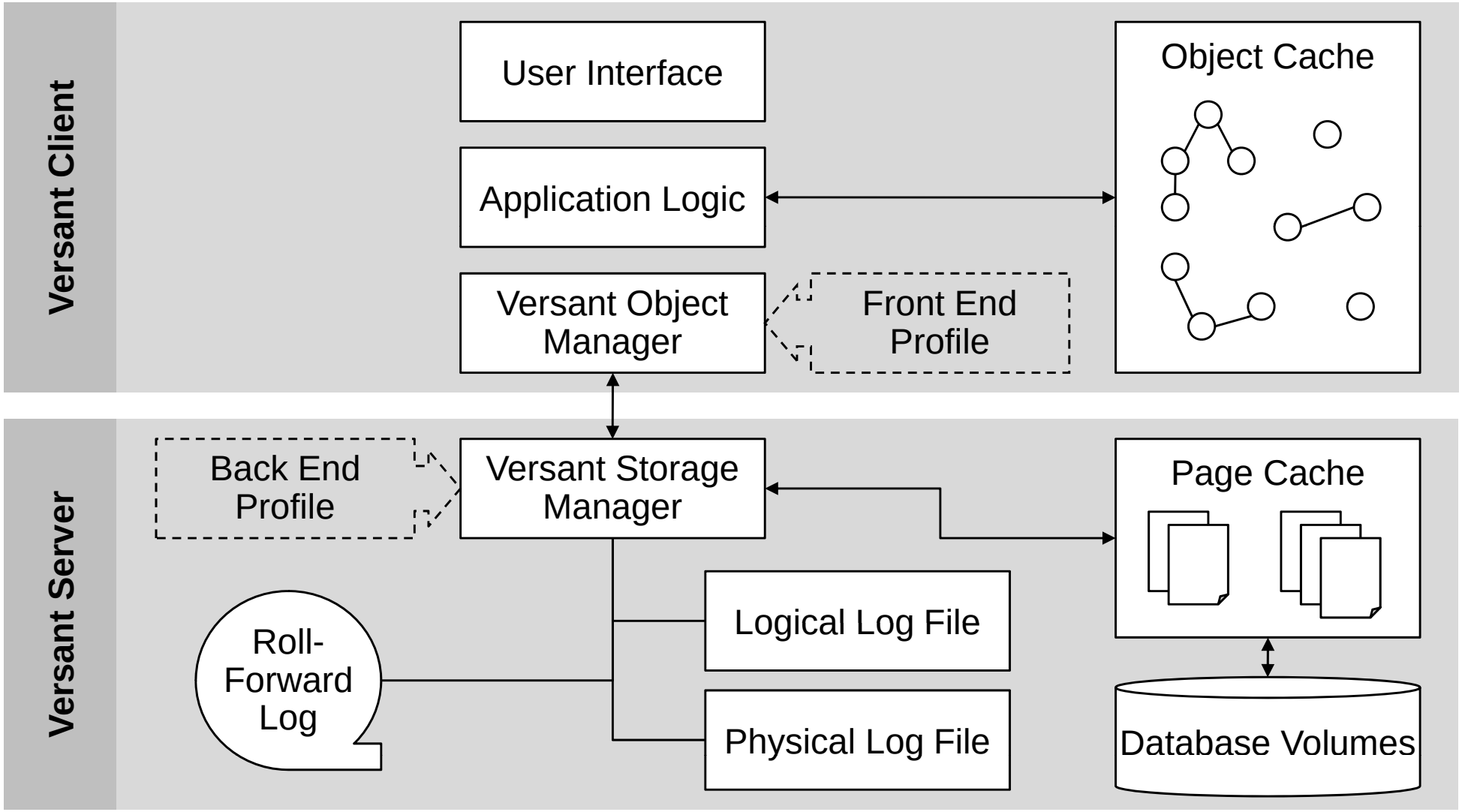
Worldwide Installations

- Telecommunications
  - Alcatel-Lucent, AT&T, Ericsson, Siemens, Nortel, France Telecom, Verizon, Samsung, Keymile, NEC
- Defense
  - BAE Systems, ESA, 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

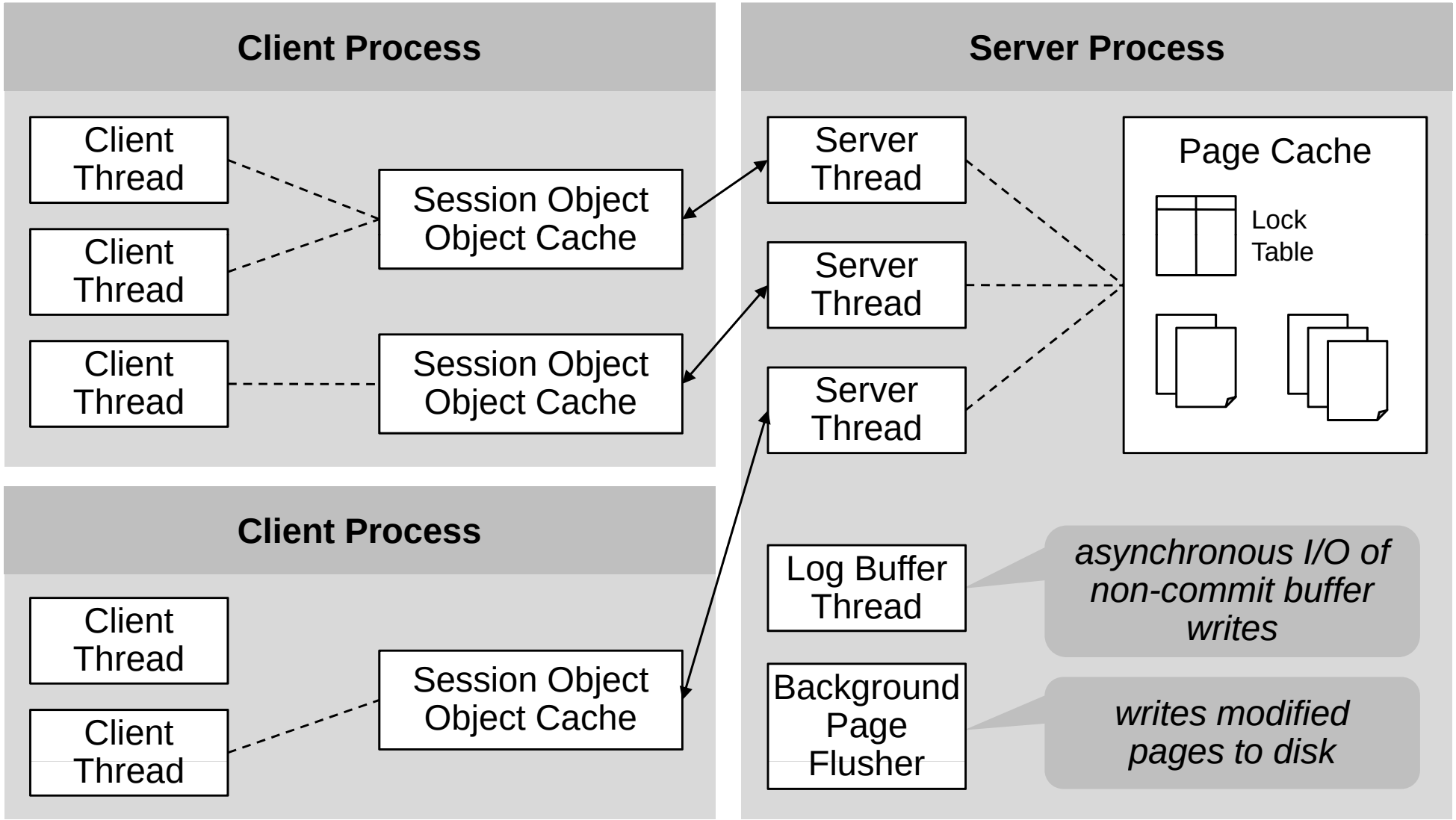
Versant Object Database Architecture



Versant Dual Cache Architecture



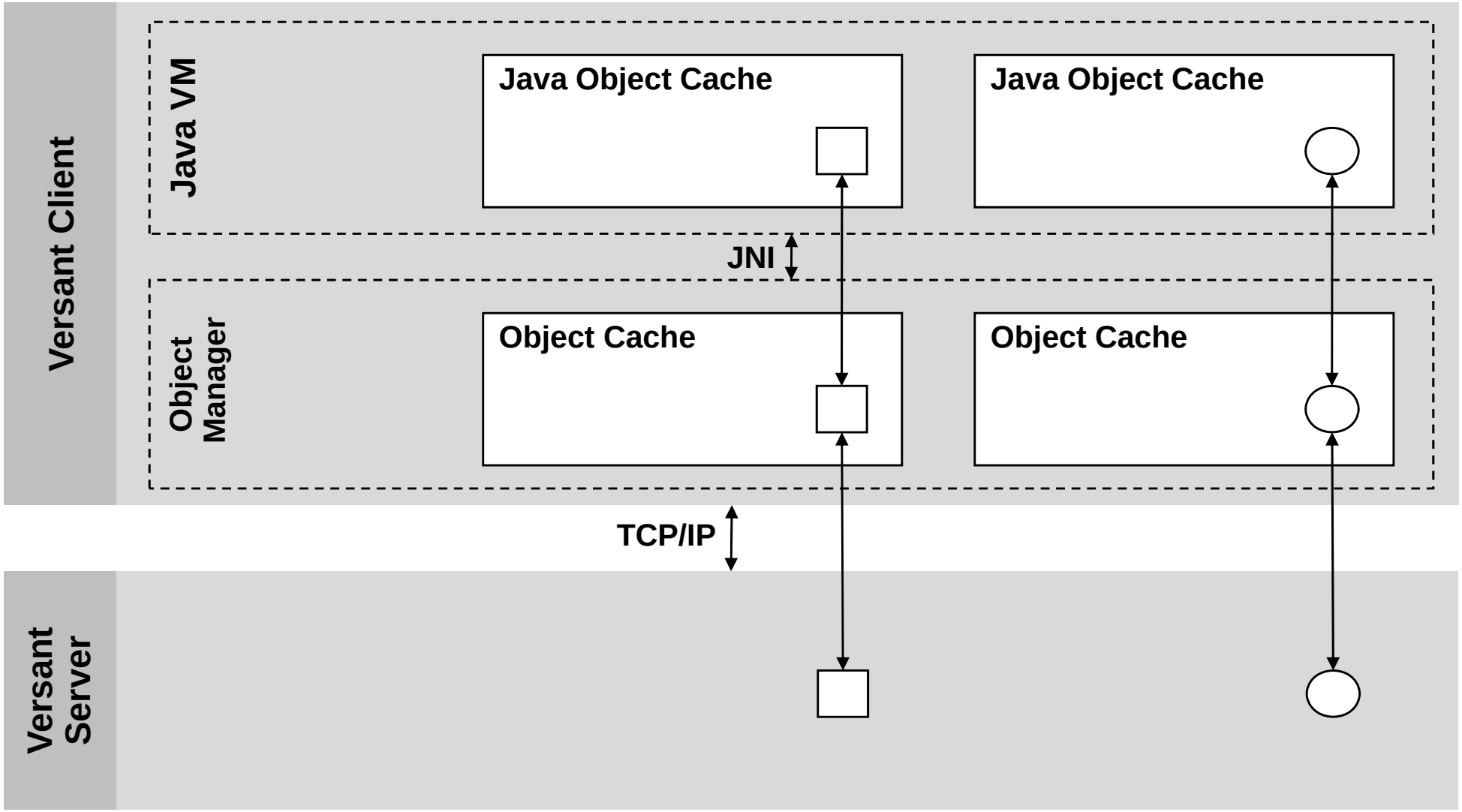
Versant Multi-Threaded Architecture



Java Versant Interface (JVI)

- Provide easy-to-use storage of persistent Java objects
  - pure Java syntax and semantics
  - instances of nearly all classes can be stored and accessed
  - works seamlessly with the Java garbage collector
  - multiple threads can work in shared or independent transactions
- Client-server architecture
  - provide access to the Versant object database
  - client libraries cache objects for faster access and navigation
  - database queries are executed on the server
- Support for Java Development Kit
  - Version 6.0.5 supports JDK 1.3
  - Version 7.0.1 supports JDK 1.4 and 1.5

JVI Architecture



JVI Layers

- Fundamental Layer
  - database-centric
  - objects manipulated indirectly through handles
  - package **`com.versant.fund`**
- Transparent Layer
  - language-centric
  - layered on top of fundamental binding
  - package **`com.versant.trans`**
- ODMG Layer
  - language-centric
  - ODMG 2.0 database and transaction model, ODMG collections
  - layered on top of transparent binding
  - package **`com.versant.odmg`**

Application Development with Versant

- Develop Java classes
  - make code “persistence aware”
  - sessions, transactions and concurrency
- Compile Java classes to generate byte-code
  - persistence behaviour inherited from base class **`com.versant.trans.Persistent`**
- Create configuration file for enhancer program
  - specify the persistence category for each Java class
- Run enhancer to make byte-code changes
- Create database
- Run application

First and Second Class Objects

- First Class Objects (FCO)
  - can be saved and retrieved independently as standalone objects.
  - have Logical Object Identifiers (LOID)
  - can be the subject of queries
  - changes to existing instances are saved automatically
  - references are always valid
  - fields marked with transient are not saved in the database
- Second Class Objects (SCO)
  - can be saved only as part of an FCO
  - cannot be the subject of queries
  - if a SCO does not have a corresponding Versant attribute type it is stored as serialized Java byte stream
  - fields marked with transient are not saved in the database

Persistence Categories (FCO)

- Persistent always (p)
  - becomes persistent at object instantiation itself
- Persistence capable (c)
  - new instances are initially transient, but may become persistent
  - **`makeRoot()`**, **`makePersistent()`** or persistence by reachability
  - object is automatically marked dirty when modified
- Superclass of a “p” or “c” class as must also be “p” or “c”
  - unless the superclass is **`Object`**
  - note that this rule is recursive

Persistence Categories (SCO)

- Transparent dirty owner (d)
  - changes to object automatically mark its owner object as dirty
  - used for serialized collections
- Persistence aware (a)
  - can directly modify attributes of an FCO
  - will call automatically mark that FCO as dirty upon any changes
- Not persistent (n)
  - no byte code enhancement.
  - cannot directly access the fields of a persistent object
  - access to such fields will throw an **`IllegalAccessError`**

Connecting to a Database

- Applications perform database operations in sessions
  - access to databases, methods, data types and persistent objects
  - must be closed before application terminates
  - one or more sessions can be open at the same time
- In each JVI layer, a session implementation exists
- Client session elements
  - object cache
  - cached object descriptor table
- Server session elements
  - associated with each connected database is a page cache for recently accessed pages
  - server page cache is in shared memory of the machine containing the connected database

Transaction Model

- Upon starting a session, Versant is always in a transaction
  - a new transaction is started automatically after **`commit()`** or **`rollback()`**
  - **`endSession()`** commits the last transaction
- Transactions have the following characteristics
  - atomic, consistent, independent, durable
  - coordinated: objects are locked for coordination with other users
  - distributed: two-phase commit for working with multiple databases
  - ever-present: application code is always in a transaction
- Committing units of work
  - **`commit()`**: releases locks and flushes cache
  - **`checkpointCommit()`**: retains locks and retains cached objects
  - **`commitAndRetain()`**: releases locks and retains cached objects

Example

```
// use the transparent layer
TransSession session = new TransSession("publications");

// find a previously defined root
Set< ? > pubs = (Set< ? >) session.findRoot("pubs");

// create a new author assuming that the Author class is either “p” or “c”
Author moira = new Author("Moira C. Norrie");
for (Object pub: pubs) {
   Publication p = (Publication) pub;
   p.addAuthor(moira);
}

// commit the changes
session.commit();

// end the session
session.endSession();
```

Updating Objects

- First Class Objects
  - changes to first class objects are automatically applied to the database upon commit
  - database objects are modified transparently
  - values of basic types are copied to database
- Second Class Objects
  - **Transparent Dirty Owner:** changes to objects are automatically applied to the database upon commit
  - **Persistent Aware** and **Not Persistent Aware:** modification of objects requires explicit dirty of owner object using method **`TransSession.dirtyObject()`**
  - The reason for this is that second class objects are serialised into the first class object that contains them

Deleting Objects

- First Class Objects
  - delete explicitly with **`TransSession.deleteObject()`** and **`TransSession.groupDeleteObjects()`**
  - these methods refer to database objects, Java instances will be garbage collected by the JVM
  - JVM calls **`finalize()`** upon garbage collection, not deletion
- Second Class Objects
  - deleted implicitly by setting reference to null
  - memory will be garbage collected by the JVM
  - upon commit, the containing first class object will not serialise the second class object

Versant Query Language (VQL)

- 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 and executed on the database server
- Queries can be parameterised
  - parameter starts with a $ followed by characters, digits or underscore
  - 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

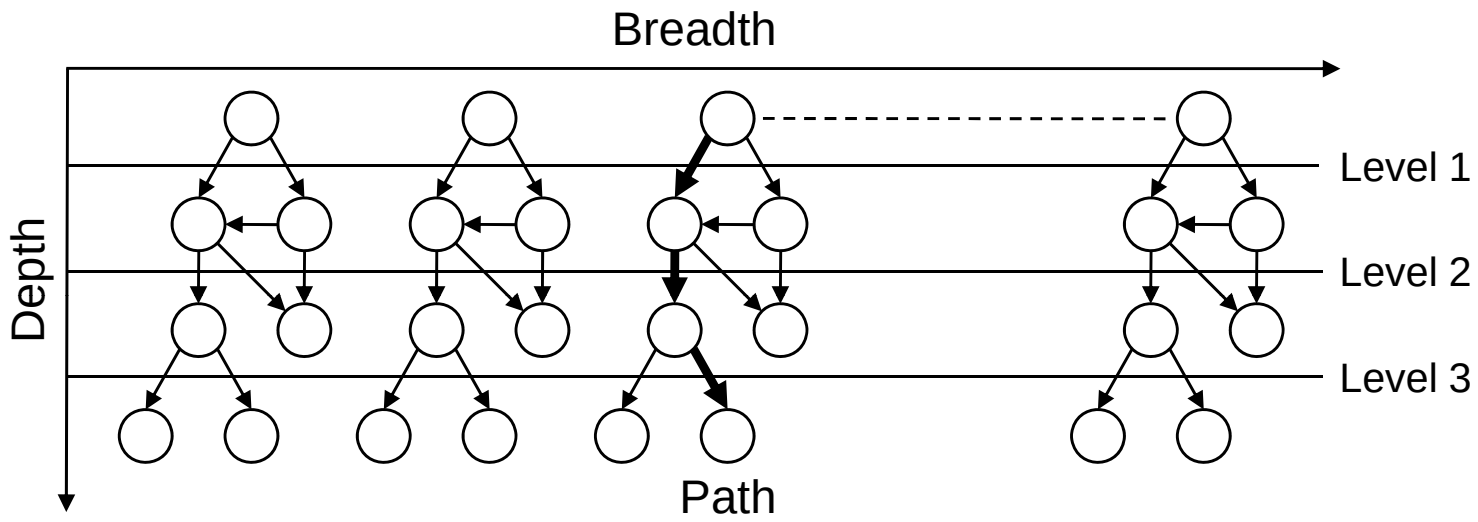
```
// find all publications by Moira C. Norrie and Michael Grossniklaus
String queryString =
   "select selfoid " +
   "from Publication " +
   "where Publication::authors subset_of $authors";

// 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.

JVI Client Cache Loader

- Versant uses a client-side object cache
  - contains query results and objects access through navigation
  - server tracks which object are cached by the clients
- Automatic object loading through closure
  - given a starting point, closure is defined as the identification and retrieval of related objects relative to the starting point
  - each time an object is dereferenced, the object manager decides if closure is required and will then locate and load the related object
- The JVI Client Cache Loader API can be used to control how and when objects are loaded
  - each dereference consists of network RPC, object lookup and I/O
  - efficiency can be improved by loading multiple objects at once
  - however, introduced vendor-specific code into domain classes

Breadth, Depth and Path Loading



- Client closure helper classes provide two simple API calls
  - **`groupReadObjects()`** and **`getClosure()`** in class **`Loader`**
- Load policies provide control out application code
  - policies to control the loading of object specified in XML file
  - XML “compiled” by Versant PolicyMaker utility
  - **`load()`** in class **`Loader`** loads objects based on the specified policy

OODBMS 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

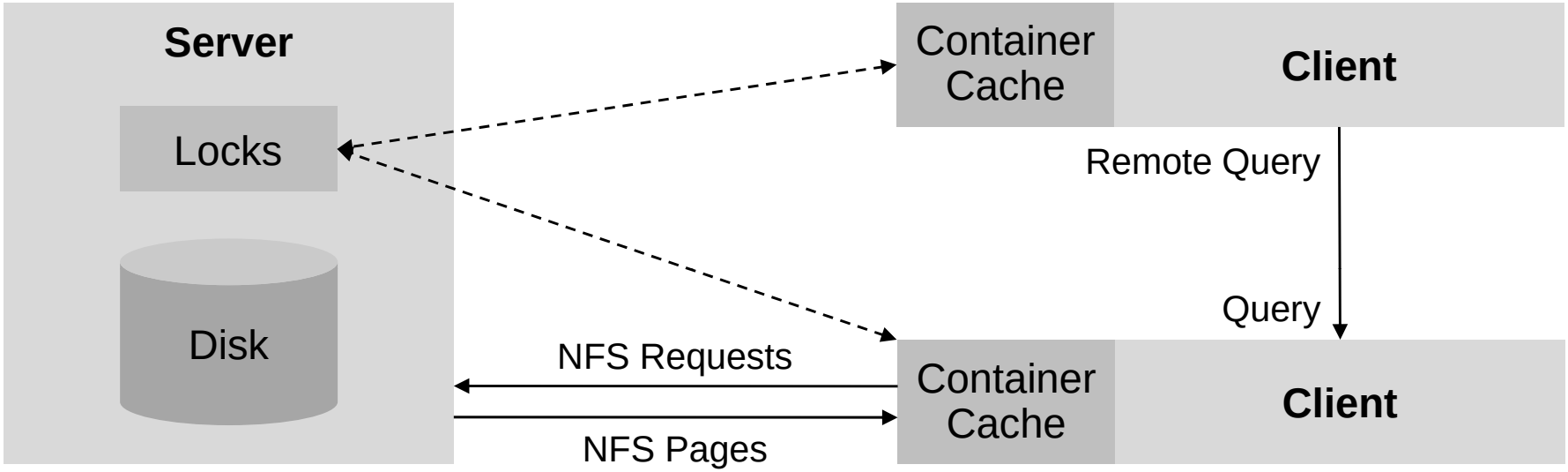
The Major Factors

- 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

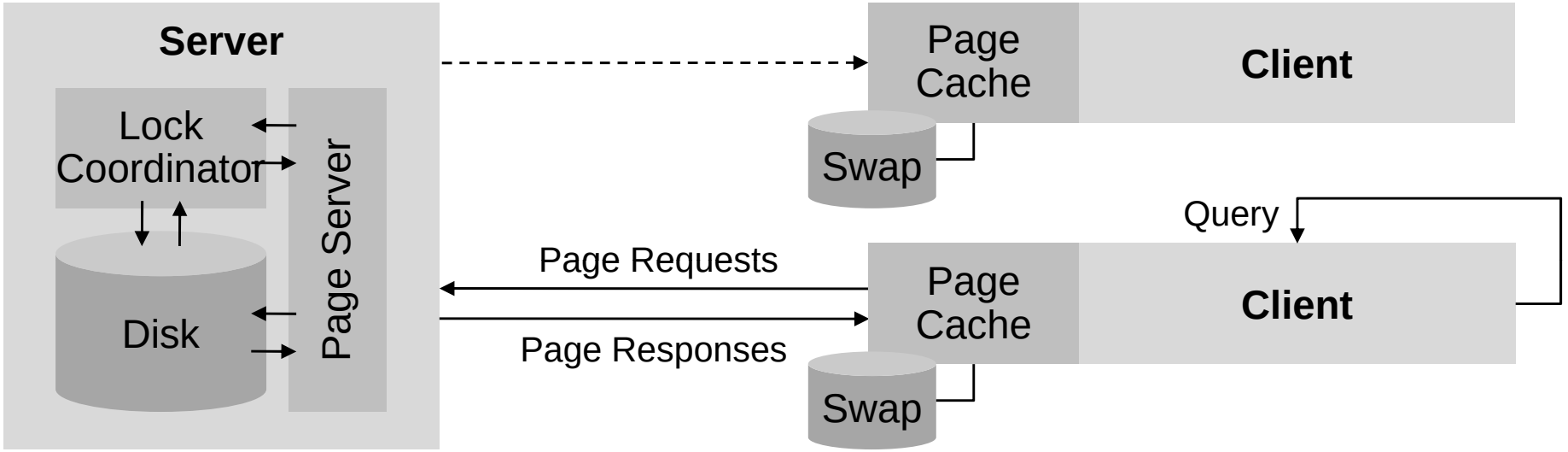
- Core architecture determines the following aspects
  - caching
  - query processing
  - transaction management
  - object life cycle management, i.e. tracking of new, dirty and deleted
- Three architectural variants are popular in OODBMS
  - container-based
  - page-based
  - object-based
- Name of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

Container-Based Architecture



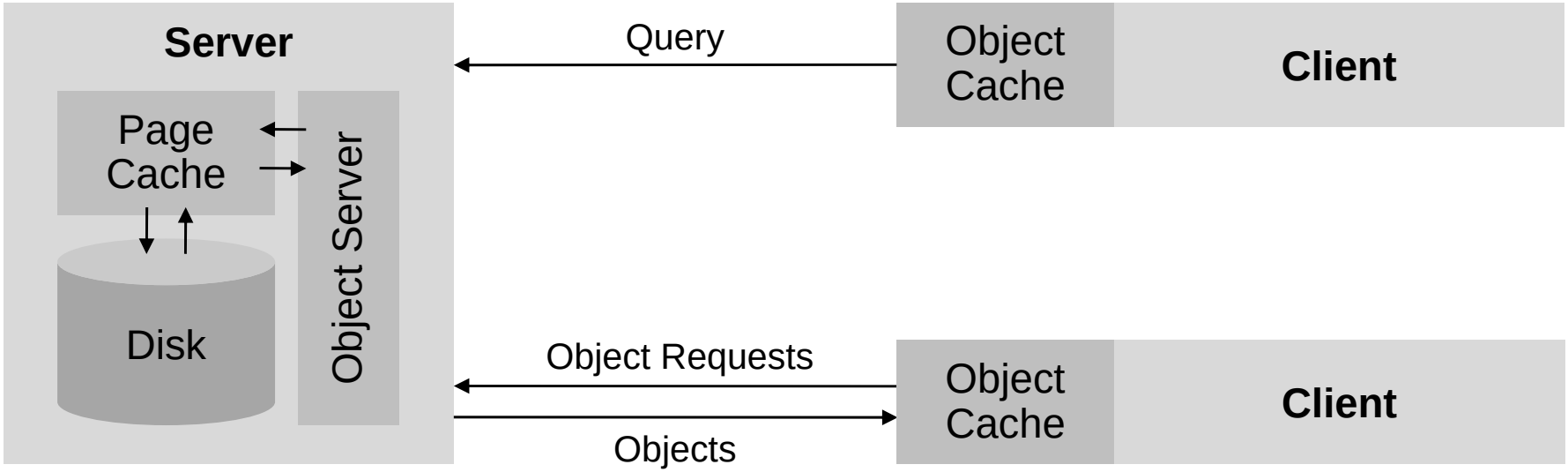
- Ships disk segments (containers) across the network
- Client libraries implement database functionality
  - container caching, query processing, object life cycle management and transactions
- All objects must reside inside a container
- Container model is layered over application domain model

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

Concurrency Model

- The three core architectures have differing concurrency models to provide transaction isolation
  - container concurrency
  - page concurrency
  - object concurrency
- All implementations of these models
  - are tightly coupled with network characteristics
  - can cache locks locally at the client across transaction boundaries
  - are likely to require lock coordination and cache consistency operations for updates
- In container and page-based systems, object placement and locking are tightly coupled
- In object-based systems, these issues are orthogonal

Container Concurrency

- Separate lock server coordinates concurrent access to data in same container
- Locking algorithm
  - clients requests lock before caching a container
  - locks are generally released at transaction boundaries
  - write request will establish a queue if there already read requests
  - subsequent read and write requests inserted in queue
  - after write request has been filled, other requests are filled
  - queue disappears if no further write requests
  - clients caching an updated container must refresh it before read
- As containers hold many objects, possibility of false waits and deadlocks

Page Concurrency

- Page server coordinates concurrent access
  - tracks which applications are accessing each page
  - grants permission to lock pages locally at the client
- Locking algorithm
  - client request lock before caching a page
  - write request causes server to use lock callbacks
  - clients either release lock and give up permission to cache page, or request blocks on each objecting client for a specified timeout
  - clients that have released lock will refresh page on next access
- As locks and permissions are taken out at the page level, false waits and deadlocks can occur

Object Concurrency

- Server maintains queues at object level to control concurrency of access to the same object
- Locking algorithm
  - clients request lock before caching object locally
  - locks are generally released at transaction boundaries
  - write request will establish queue, if read locks already exist
  - all subsequent requests are inserted into queue
  - after write request has been filled, other requests are filled
  - queue disappears if no further write requests
  - clients caching an updated object must refresh it before read
- Since locking is done at object level, no false waits or deadlocks can occur

Network Model

- Core architectures imply three different network models
  - container network model
  - page network model
  - object network model
- In a distributed system, different network models affect
  - bandwidth utilisation
  - performance
- Network model is closely tied to locking model
  - information transfer occurs upon lock, if not already cached locally
  - lowest granularity level of network transfer is equal to lowest granularity level of locking

Network Models

- Container network model
  - object request translated to container request
  - entire container transferred, even if not all objects are accessed
  - locks are place on the whole container
- Page network model
  - object request translated to page request
  - entire page transferred, even if not all objects are accessed
  - locks are placed on the whole page
- Object network model
  - unit of transfer is an object or collection of objects
  - object or collection request may include a depth request
  - locks are placed on one object or all objects within a collection

Query Implementation

- OODBMS focus on supporting seamless navigation between related objects using language constructs
  - native support of navigational access is a key advantage of OODBMS over the RDBMS complex join concept
  - relationships are a static part of the system rather than runtime computed
- Querying data in OODBMS consists of two steps
  - access first level objects of a use case
  - use navigation to access related objects
- Query implementation impacts on
  - where does the query execution take place
  - flexibility of what can be queried
  - indexing capabilities

Container Query

- Query processing at the client
  - client may be another remote client
  - client is, however, a process separate from the NFS page server
  - “opposite” architecture compared to a relational database
- Query processing
  - all objects involved in the query must be identified by the database or container they reside in (which also includes potential indexes) and loaded into the client process for query execution
  - query returns the containers holding the result objects
- From network and locking perspective, result may contain objects that did not actually satisfy the query predicate

Page Query

- Query processing at the client
- Query processing
  - all pages containing objects involved in the query are loaded
  - query returns references to objects that satisfy the query predicate
  - pages containing these objects have, implicitly, already been loaded across the network and translated into the client memory
- Indexed query processing
  - all index pages relevant to the query are loaded
  - query returns references to objects that satisfy the query predicate
  - pages containing these objects may have to be loaded across the network and translated into the client memory
- From network and locking perspective, result may contain objects that did not actually satisfy the query predicate

Object Query

- Query execution engine runs within database server
  - any object is reachable via query, even if it has no relationships
  - indexes for object attributes maintained on server
- Query processing
  - query statement sent to server from client
  - query executed on server using optimiser and indexes
  - result set of objects satisfying the query predicate returned to client
- Only query statement and objects satisfying the query are transferred across the network

Identity Management

- OODBMS use object identity for establish uniqueness and implementing relationships
- Identity management impacts on
  - long-term operational behaviour
  - flexibility and data scalability
- Physical identity
  - unique identifier is dependent on the physical location
  - mutable, reusable, immobile and rigid
  - dereferencing very fast
- Logical identity
  - unique identifier is independent of the physical location
  - immutable, never reused, mobile and flexible
  - logical references needs to be translated into physical reference

OODBMS 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

Literature

- Versant Object Database
  - http://www.versant.com/
- Robert Greene: OODBMS Architectures
  - http://www.odbms.org/experts.html#article9
- Adrian Marriott: OODBMS Architectures Revisited
  - http://www.odbms.org/experts.html#article11
- Robert Greene: OODBMS Architectures Defended
  - http://www.odbms.org/experts.html#article12

Next Week
Storage and Indexing

- Type Hierarchy Indexing
- Aggregation Path Indexing
- Collection Operations