Object-Oriented Databases
Commercial OODBMS: Versant

- Versant Object Database for Java
- Java Versant Interface (JVI)
- Versant Query Language (VQL)

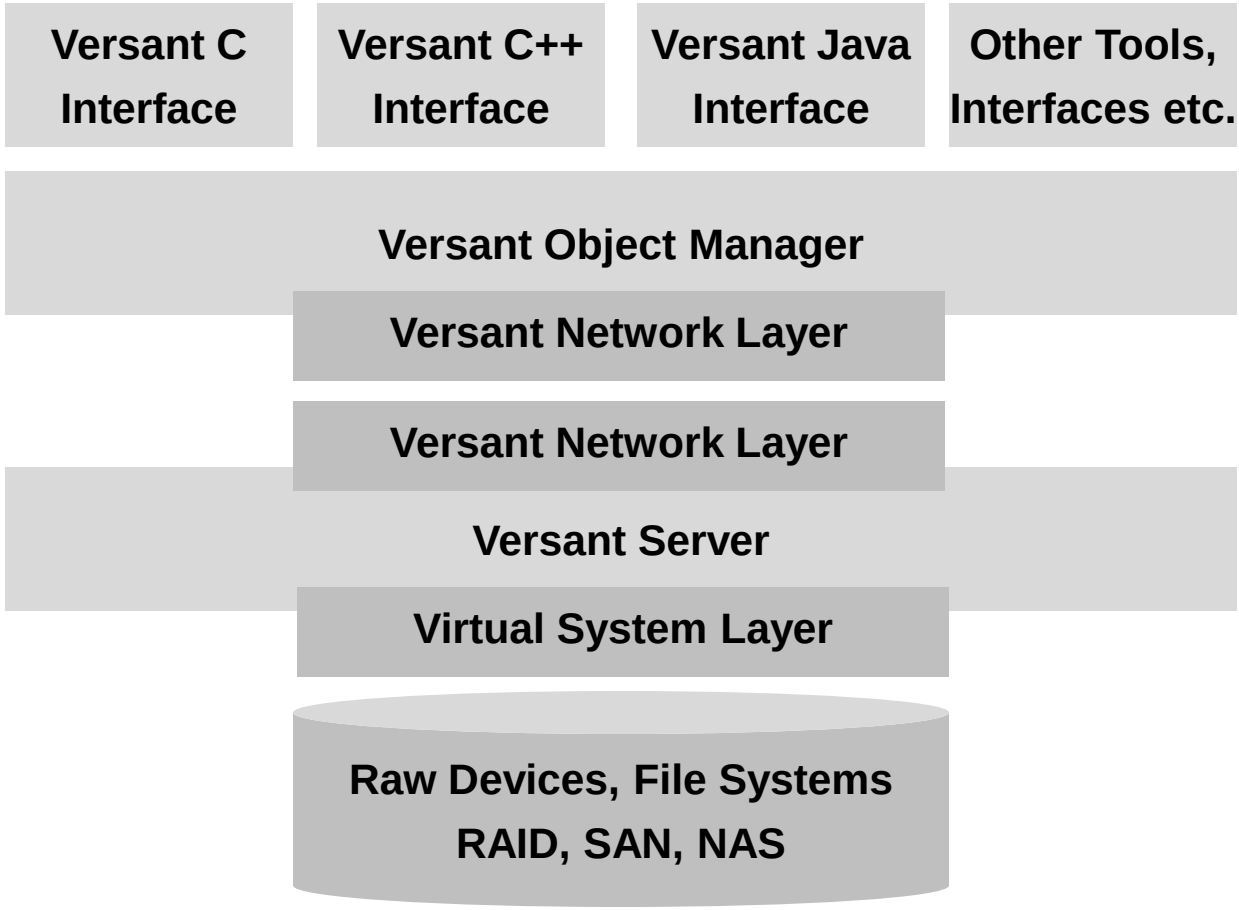
Versant

- 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)
  - market leader in object databases
  - current version 7.0.1
  - 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

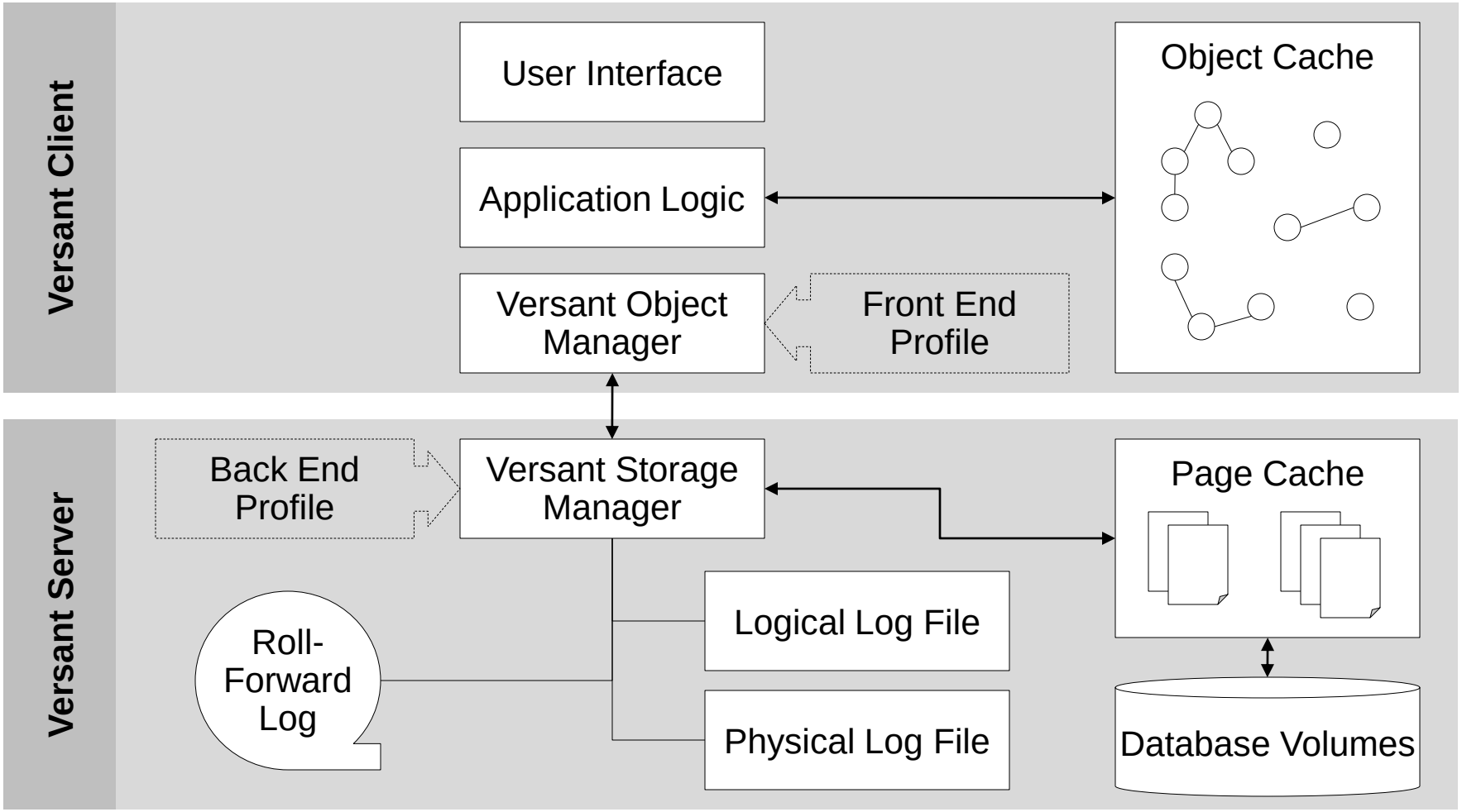
Worldwide Installations

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

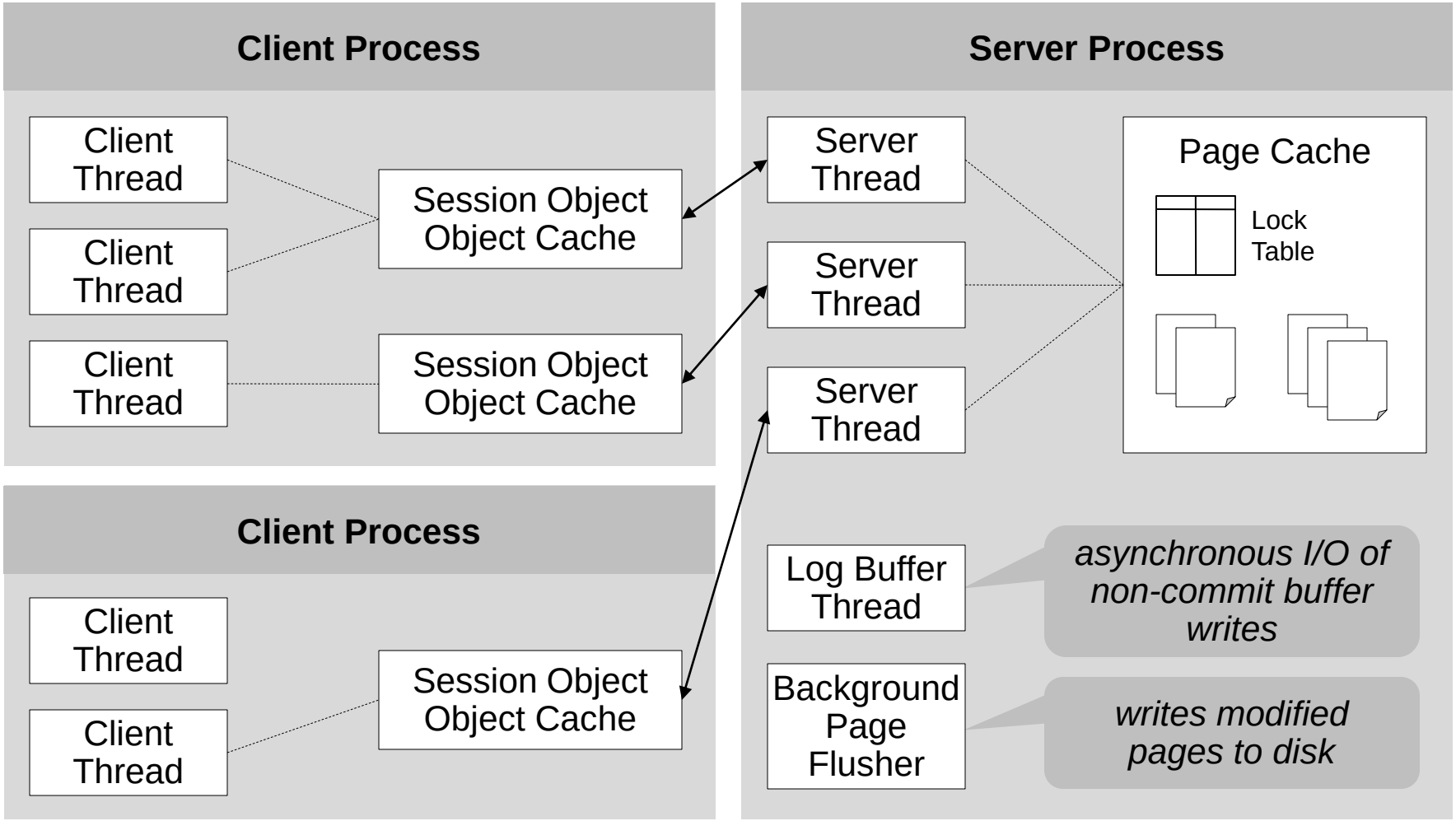
Versant Object Database Architecture



Versant Dual Cache Architecture



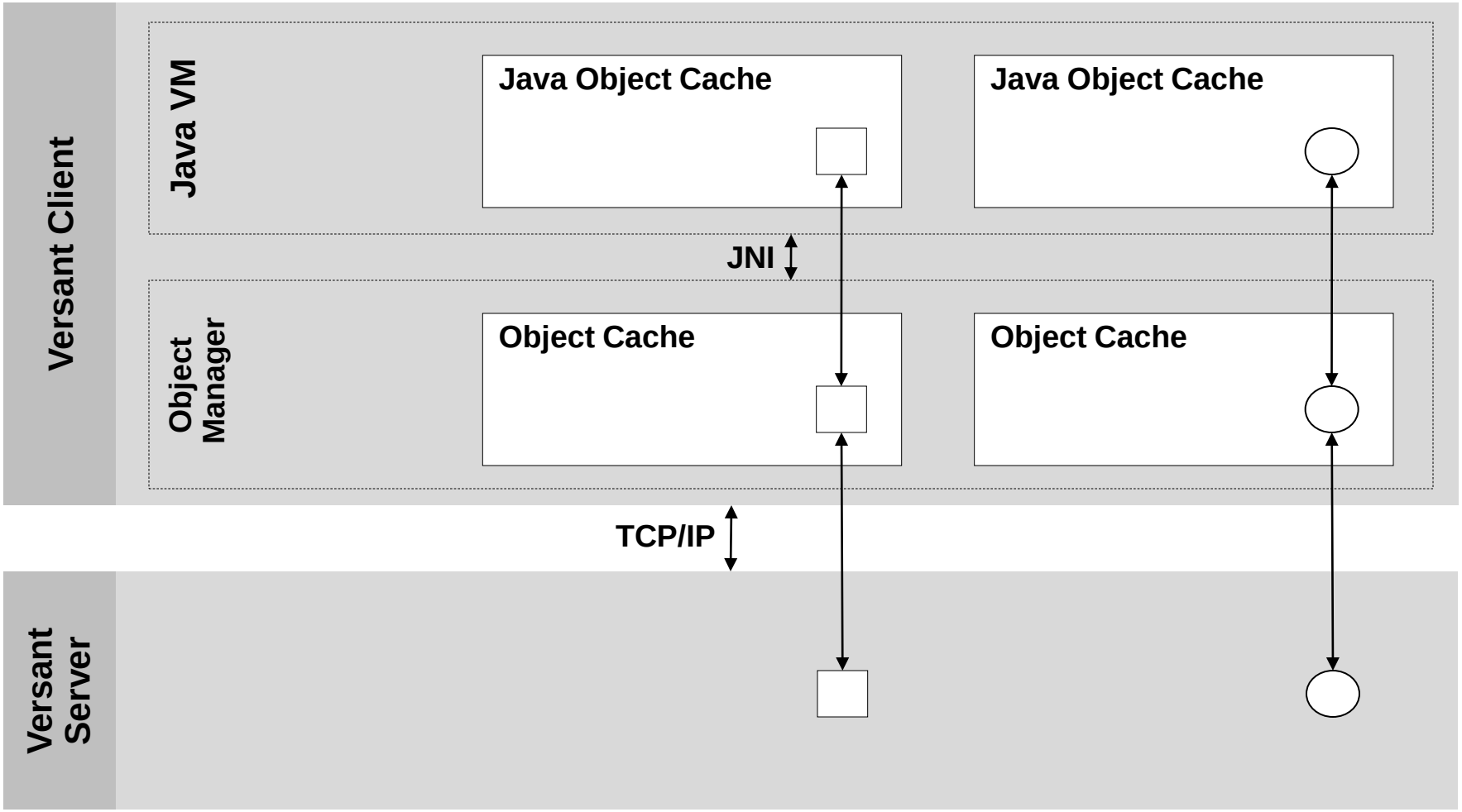
Versant Multi-Threaded Architecture



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 database and transaction model, ODMG collections
  - layered on top of transparent binding
  - packages **`com.versant.odmg`** and **`com.versant.odmg3`**

Application Development with Versant

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

Persistence and Navigation Model

- Versant provides persistence by reachability
- Database root can be used to persist the root of an object graph and assign it a name for retrieving it later
  - supported in both transparent and ODMG binding
  - intended to be applied to a relatively small number of objects
  - **`makeRoot()`** creates root and stores object
  - **`deleteRoot()`** removes root but does not delete object
  - **`findRoot()`** retrieves root object
- Transparent navigation
  - starts from identity, root object, class extent or query
  - navigation is used to access associated objects
  - Versant transparently locks and retrieves objects from database
  - works across database boundaries

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 to existing FCOs are always valid
  - fields marked as 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
  - object is automatically marked dirty when modified
- 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 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 and transparently modify attributes of an FCO
  - if a SCO of a FCO is modified, **`dirtyObject()`** must be called for the FCO that contains the SCO in order to save it
- 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
  - after **`commit()`** or **`rollback()`**, a new transaction is started automatically
  - **`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

Object Lifecycle

- Creation of persistent objects
  - Java objects are created in Java memory
  - internal database information per object in the Versant object cache
- Commit
  - object data written to database
  - “hollow” proxy Java objects retained in memory space
- Rollback
  - new database objects will be dropped
- Querying objects
  - query passed to database server
  - proxy object for every matching object in the result set
- Accessing objects
  - Versant transparently fetches object or de-serializes the object

Example

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

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

// create a new author assuming that the Author class is either "p" or "c"
Author moira = new Author("Moira C. Norrie");
for (Object object: publications) {
   Publication publication = (Publication) object;
   publication.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:** modification of SCO requires explicit dirty of owner FCO using method **`TransSession.dirtyObject()`**
  - the reason is that SCOs are serialised into owner FCO
  - if a SCO is contained in two FCOs, this will lead to two instances of the SCO in the Java memory after reloading the FCOs

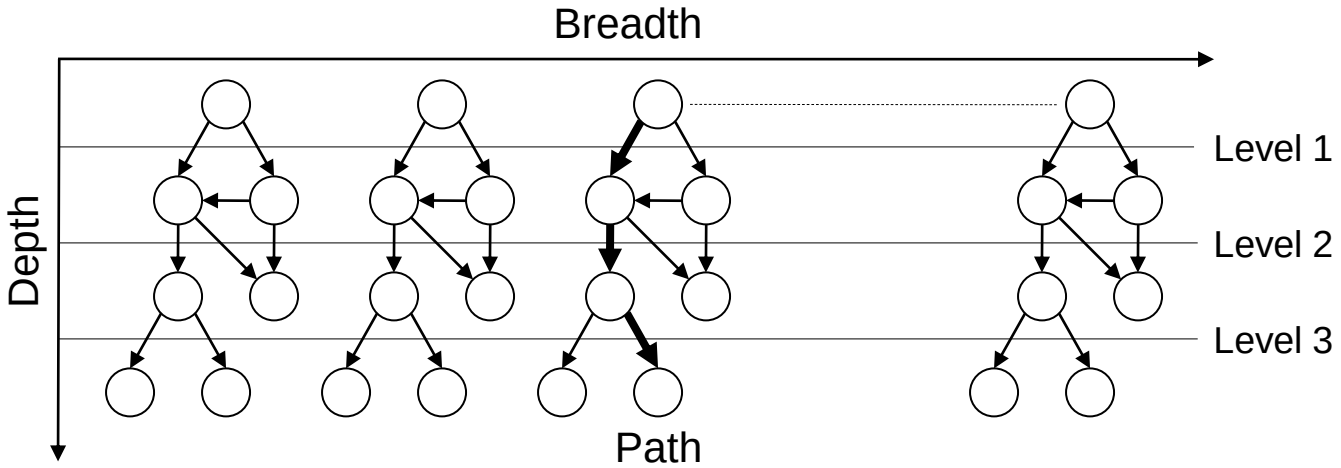
Deleting Objects

- 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 FCO will not serialise the SCO

JVI Client Cache Loader

- Versant uses a client-side object cache
  - contains query results and objects accessed through navigation
  - server tracks which objects 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, introduces 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 outside of 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

Versant Collections

- Storage of Java collections supported
  - **`Array`**, **`Vector`**, **`Hashtable`**, **`LinkedList`**
- First class object (FCO) collections
  - **`VVector`**, **`VHashtable`**
- Second class object (SCO) collections
  - **`DVector`**
- Scalable large collections
  - **`LargeVector`**
- ODMG collections

Scalable Large Collections

- Classes **`DVector`**, **`VVector`** and **`VHashtable`** as well as ODMG collections are implemented in the front-end
  - mapped to attributes of type variable-length storage (**`vstr`**)
  - performance issue with large number of objects in a collection
- Class **`com.versant.util.LargeVector`**
  - implements the standard interface of Vector
  - broken up into multiple nodes
  - only needed nodes are brought to front-end on element access
- Locking issues
  - more concurrency as not the whole object needs to be locked
  - potential for deadlocks
  - use locking protocol, e.g. update in ascending order only

ODMG Collections

- Follow the specification of the ODMG standard
  - implemented in Versant as thin layer over the transparent binding
  - collection classes also available from **`TransSession`**
- ODMG 2.0
  - JDK 1.1 style collections
  - Package **`com.versant.odmg`**
- ODMG 3.0
  - JDK 1.2 style collections
  - Package **`com.versant.odmg3`**
  - Extend the **`java.util.Collection`** interfaces
  - Versant recommends using this style of collections
- ODMG Collections are first class objects (FCO)

ODMG Collection Query Facilities

- ODMG collections provide additional query facilities
- **`VCollection`** implements **`java.util.Collection`** to add query capabilities over collections
  - **`boolean existsElement(String predicate)`**
  - **`DCollection query(String predicate)`**
  - **`Iterator select(String predicate)`**
  - **`Object selectElement(String predicate)`**
- Queries over ODMG collections
  - only objects in the collection are considered
  - predicate is the **`where`** part of a VQL Query
  - only persistent collections can be queried

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, 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.

Event Notification

- Propagation of events from database to registered clients
  - Java Beans event model
  - callback to event listener objects
- Event types
  - **class events:** create, modify or delete instance
  - **object events:** modify or delete object or group of objects
  - **transaction demarcation:** begin or end transaction
  - **user-defined events**
- Application programming interface
  - package **`com.versant.event`**
  - sub-interfaces of **`VersantEventListener`** for each event type
  - class **`EventClient`** provides client-side functionality

Event Channels

- Events communication based on event channels
  - abstraction for broadcasting an event notification
  - event listeners are registered via a channel
  - after creation, an application can “tune-in” to an event channel
- Global namespace for event channels
  - persistent across client applications
- Categories
  - **class-based:** class events for a specified set of classes
  - **object-based:** object events for a specified set of objects
  - **query-based:** class events for objects that match a specified query
- Channel management through **`EventClient`**
  - create new event channel using **`ChannelBuilder`**
  - access an existing event channel

Persistent Object Hooks

- Allow intervention at all stages of state transitions of a persistent object
  - compute transient attributes
  - build transient caches
  - perform housekeeping tasks to preserve referential integrity
- Hook methods
  - **`activate()`** and **`deactivate()`**
  - **`preRead(boolean act)`** and **`postRead(boolean act)`**
  - **`preWrite(boolean deact)`** and **`postWrite(boolean deact)`**
  - Boolean parameter indicates whether object has been activated/deactivated (**`true`**) or not (**`false`**)
  - **`vDelete()`** when object is deleted

Maintaining Referential Integrity

```
public class Author {

   private String name;
   private Date birthday;
   private Set<Publication> authoredBy;

   // delete hook that also removes publication of the deleted author
   void vDelete() {
      TransSession session = (TransSession)
            TransSession.sessionOfCurrentThread();
      for (Publication publication: this.authoredBy) {
         session.deleteObject(publication);
      }
   }

   ...

}
```

Schema Evolution

- Support for schema evolution based on application programming interface
- Fundamental binding
  - inserting, appending, dropping, and renaming attributes
  - adding and renaming classes

```
ClassHandle c = session.locateClass("Report");
c.renameClass("Form");
AttrString description = session.newAttrString("description");
AttrBuilder attribute = session.newAttrBuilder(description);
c.appendAttr(attribute);
```

- Transparent binding
  - method **`TransSession#setSchemaOption(int)`** to configure automatic schema evolution

Literature

- Versant Object Database
  - http://www.versant.com/

Next Week
Commercial OODBMS: ObjectStore

- ObjectStore PSE Pro for C++
- Virtual Memory Architecture
- Managing Object Data