Introducing ObJectRelationalBridge (OJB)

Atlanta Java Users Group

Chuck Cavaness

April 15, 2003

Speaker Introductions

- Senior Technologist (S1 Corporation)
- Co-Author of Special Edition EJB 2.0 and Special Edition Java 2
- Author of Jakarta Struts from O'Reilly and Struts Pocket Reference
- Various Articles for O'Reilly, Linux Magazine, JavaWorld, etc.

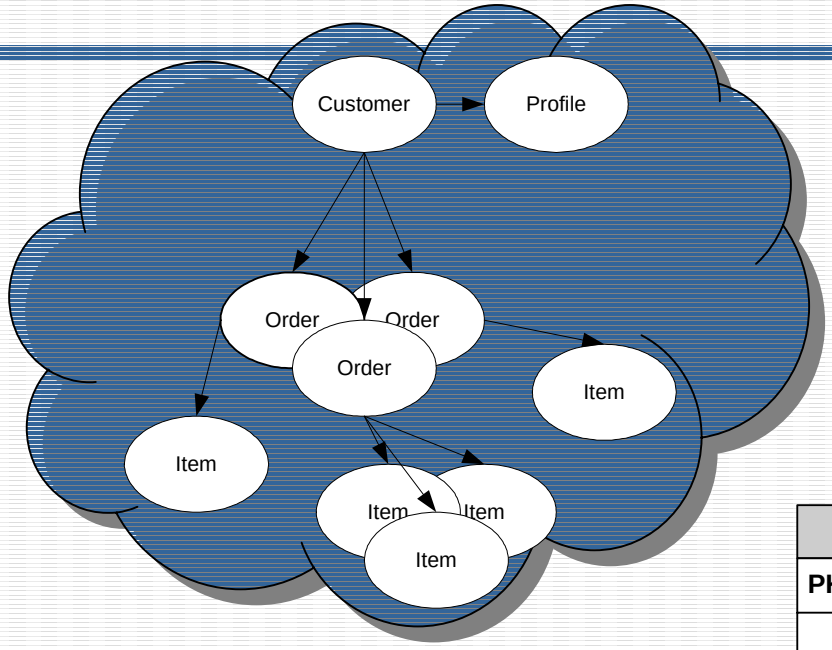
Goals for Presentation

- Make a case for using ORM frameworks like OJB
- Introduce OJB
- Discuss Build/Buy/Borrow Persistence Decisions

Assumptions about Audience

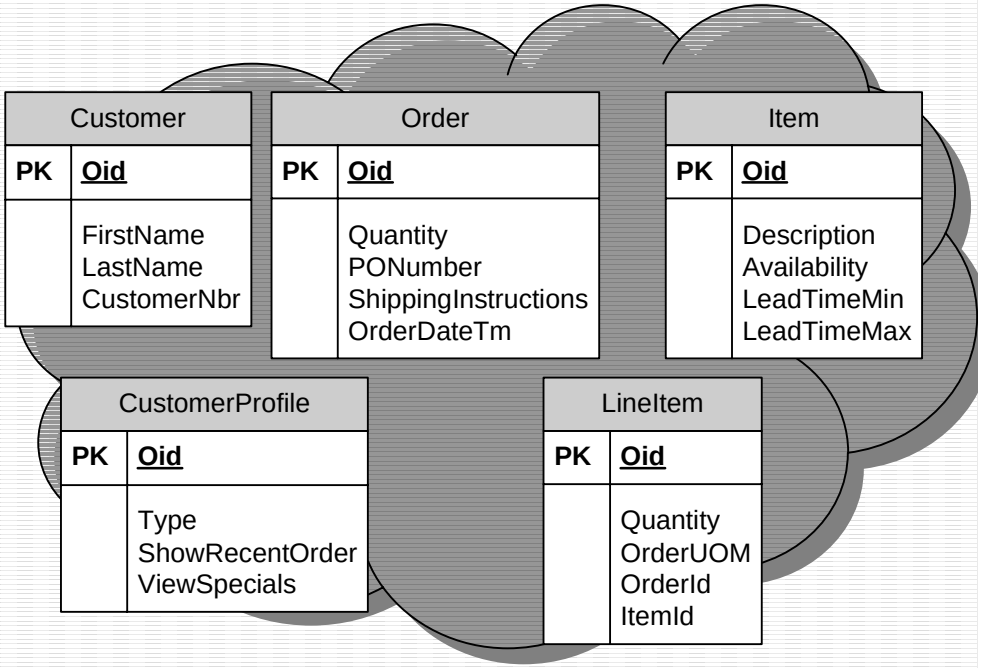
- Knowledge of Java and JDBC
- Knowledge of Relational Databases
- General Knowledge of Object-to-Relational Mapping Techniques

The Impedance Mismatch



The Logical World

The Physical World



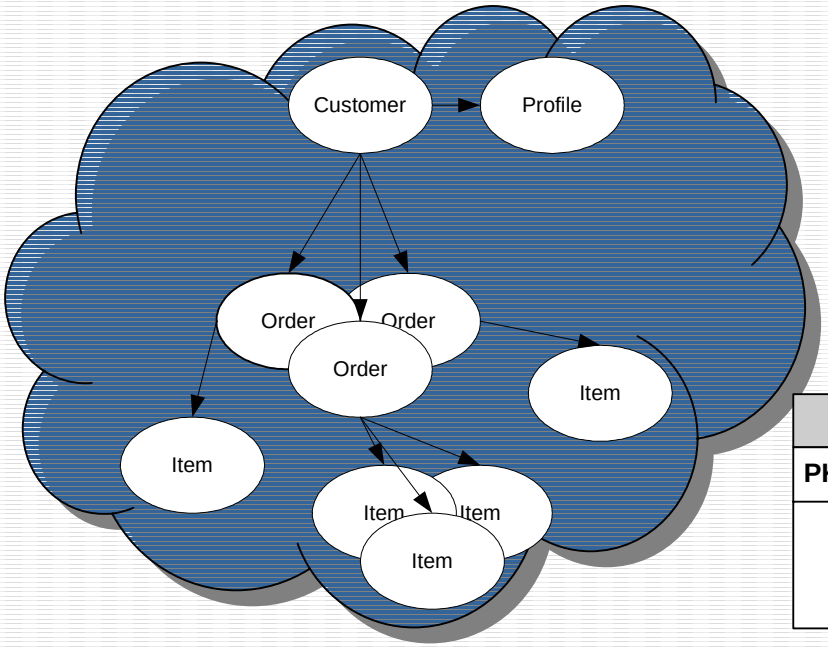
Object/Relational Differences

- Technical Differences
- Cultural Differences

Object/Relational Technical Differences

- Object-oriented paradigm based on Engineering Principles
- Relational paradigm based on Mathematical Principles
- Objects are traversed through relationships
- Relational paradigm joins data from tables

Deceptive Similarities, Subtle Differences *(Scott Amber)*



The Logical World

The Physical World

Customer	
PK	**Oid**
	FirstName LastName CustomerNbr

Order	
PK	**Oid**
	Quantity PONumber ShippingInstructions OrderDateTm

Item	
PK	**Oid**
	Description Availability LeadTimeMin LeadTimeMax

CustomerProfile	
PK	**Oid**
	Type ShowRecentOrder ViewSpecials

LineItem	
PK	**Oid**
	Quantity OrderUOM OrderId ItemId

Object/Relational Cultural Differences

- Object people and data people look at problems differently
- Scott Ambler calls this the "Object-Data Divide"

Consequences of the "Object-Data Divide"

- Software is not delivered on time and within budget
- The technical mismatch between Object Model and data model is worsened
- Data models often fail to be good drivers for the object model
- Frustration within the organization causes higher turnovers.

Solving the Impedance Mismatch

- Technical mismatch can be overcome by educating yourself and team on both technologies
- Decouple object world from database access as much as possible
- Cultural mismatch is harder to solve, but solveable

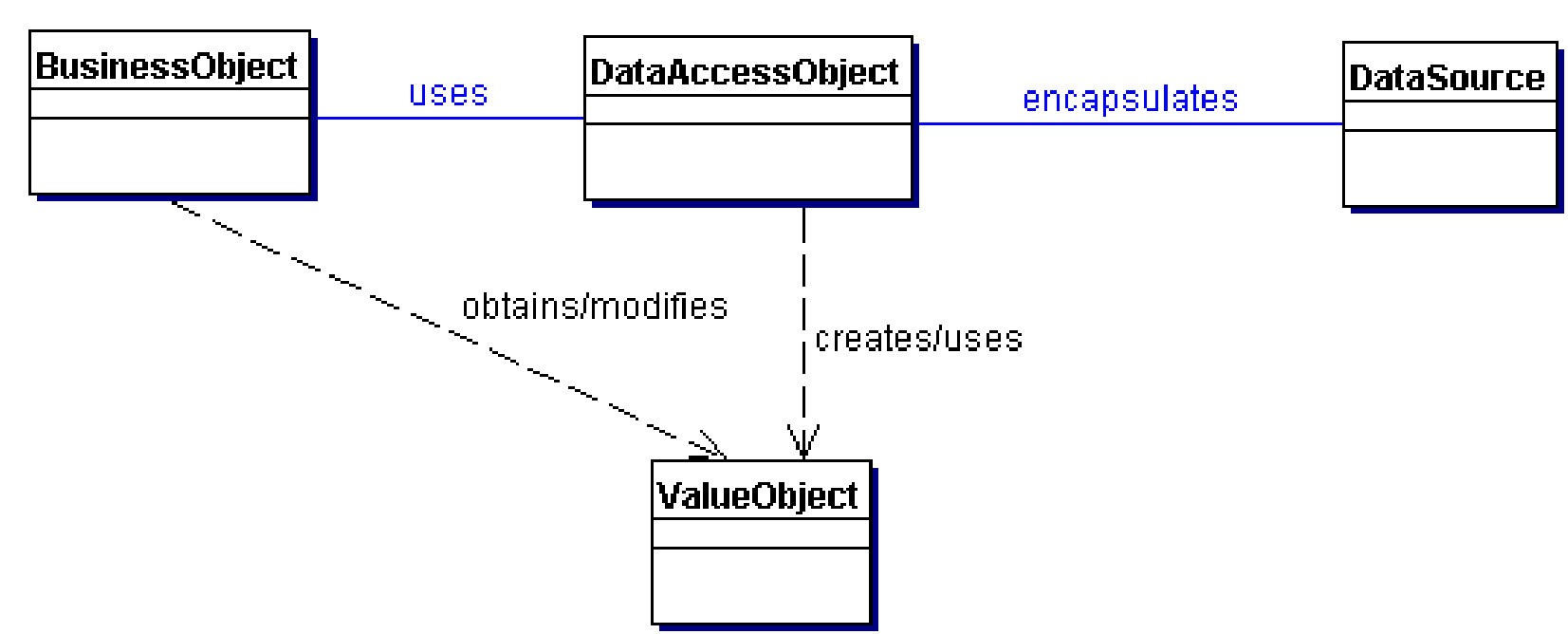
Data Access Strategies

- Brute Force
- Roll your own Approach
- Persistence Frameworks
- JDO
- J2EE Container-Managed Persistence

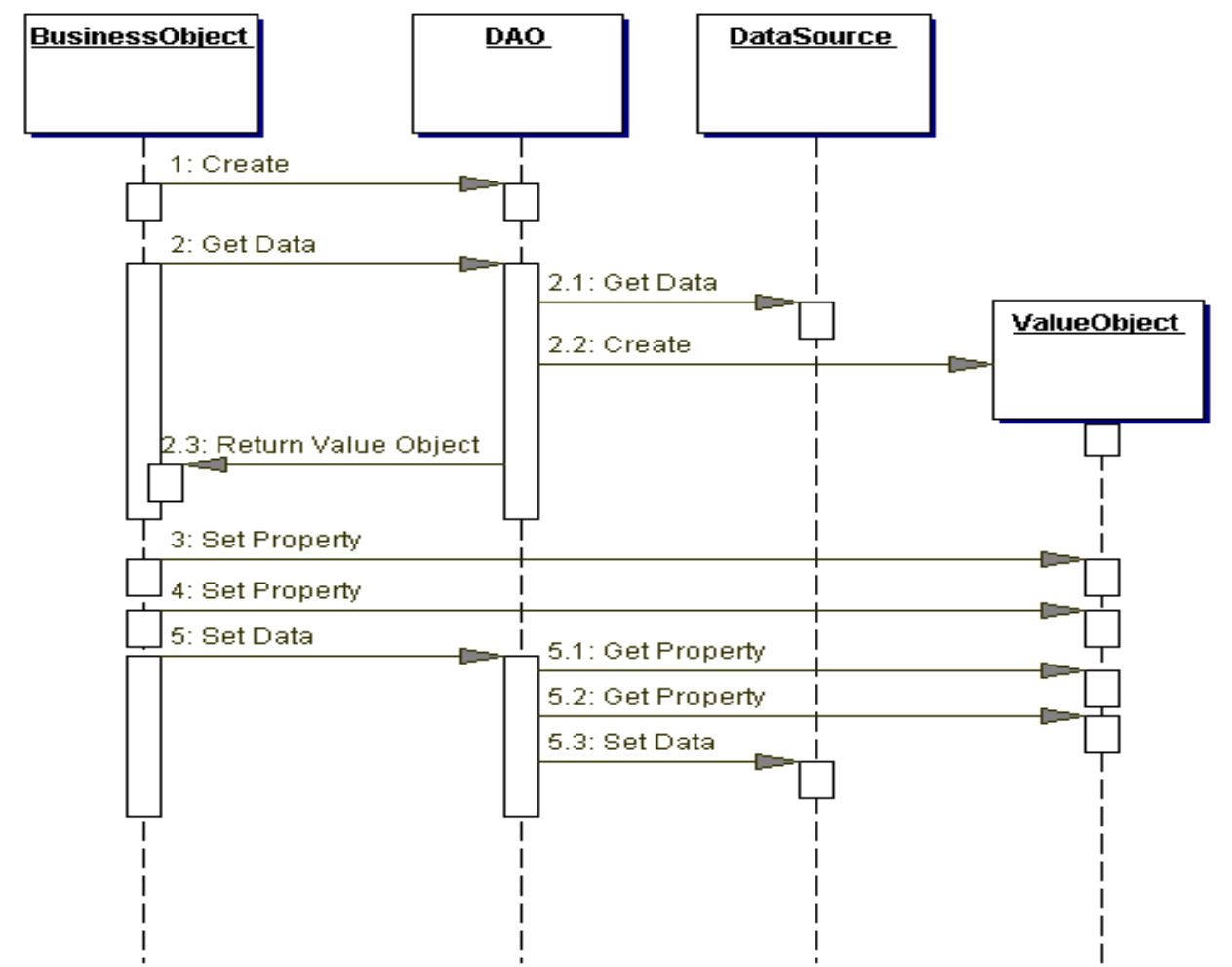
Strategey #1: Brute Force

```
public class Auction extends PersistentEntity {
  private int oid;
  private String name;
  private String description;
  private static String SELECT_SQL = "SELECT oid,name,description FROM Auction WHERE oid = ?"
  public void load() throws SQLException {
    Connection conn = getConnection();
    try{
      PreparedStatement pStmt = conn.prepareStatement(SELECT_SQL);
      pStmt.setInt(1, getOid() );
      ResultSet results = pStmt.executeQuery();
      setOid( results.getInt(1) );
      setName( results.getString(2) );
      setDescription( results.getString(3) );
    }catch( SQLException ex ){
      logger.error( “SQLException occurred”, ex );
      throw ex;
    }finally{
      if (conn != null && !conn.isClosed() ){
        conn.close();
      }
    }
  }
```

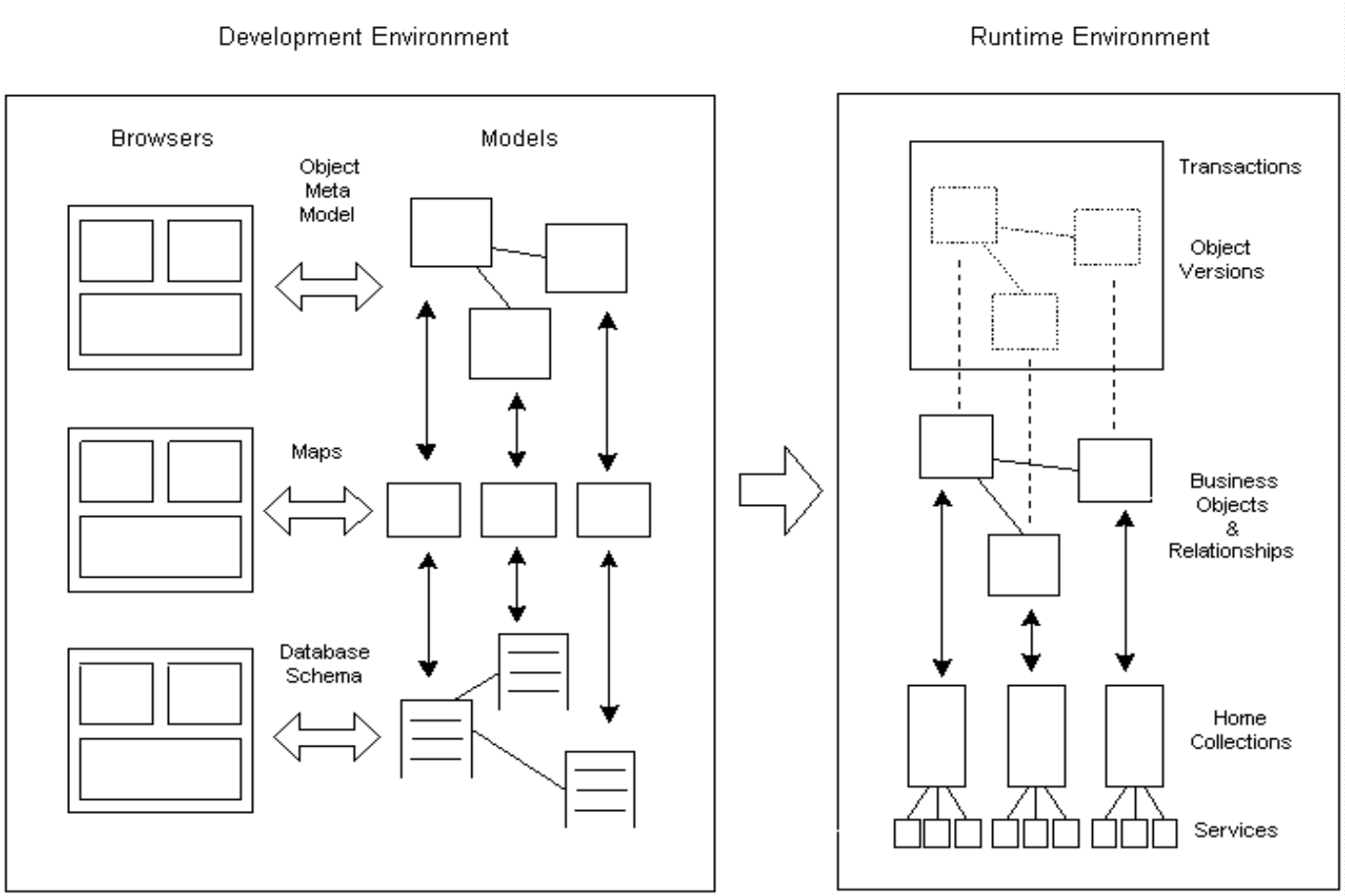
Strategy #2: DAO



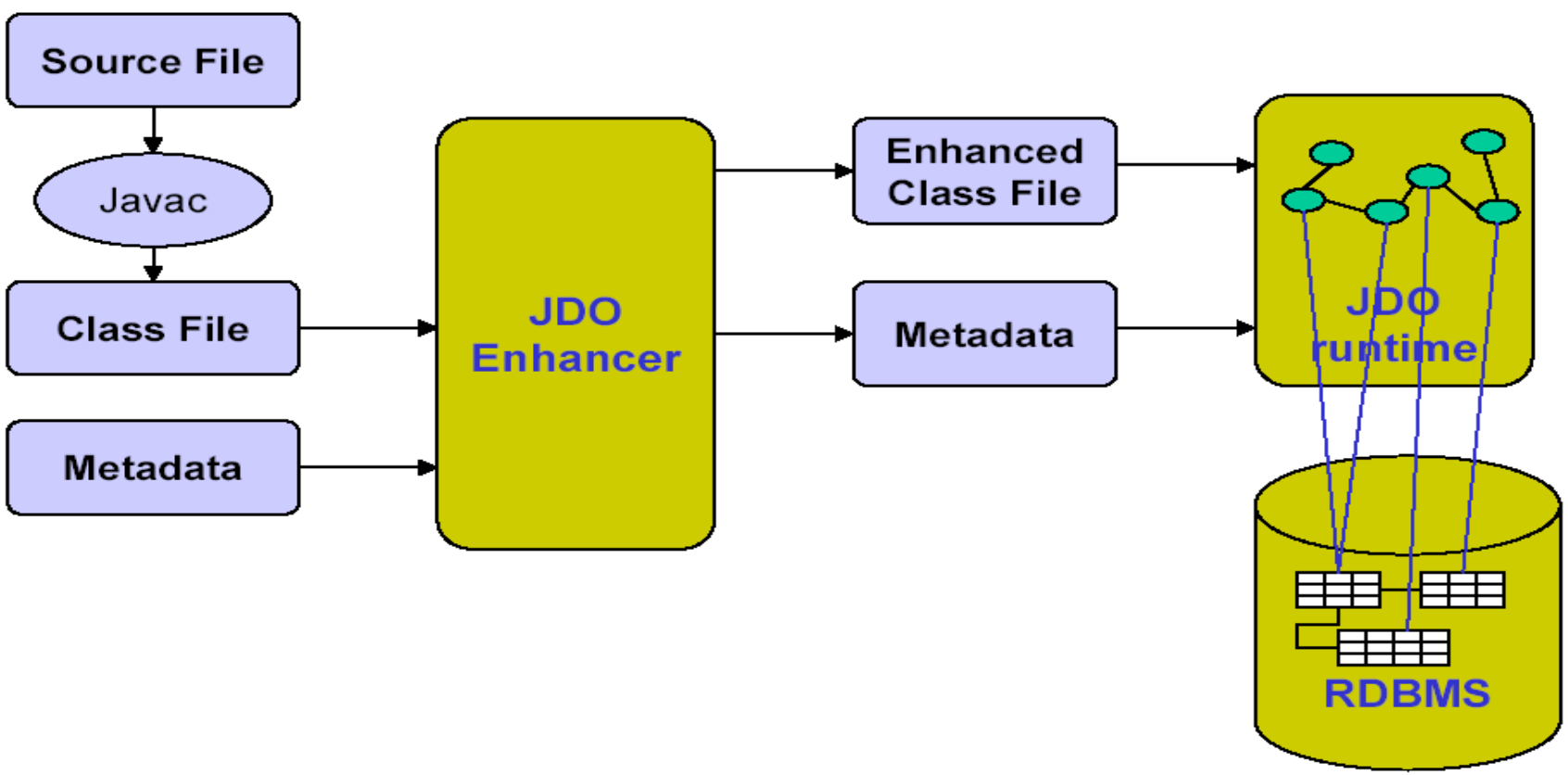
DAO Sequence Diagram



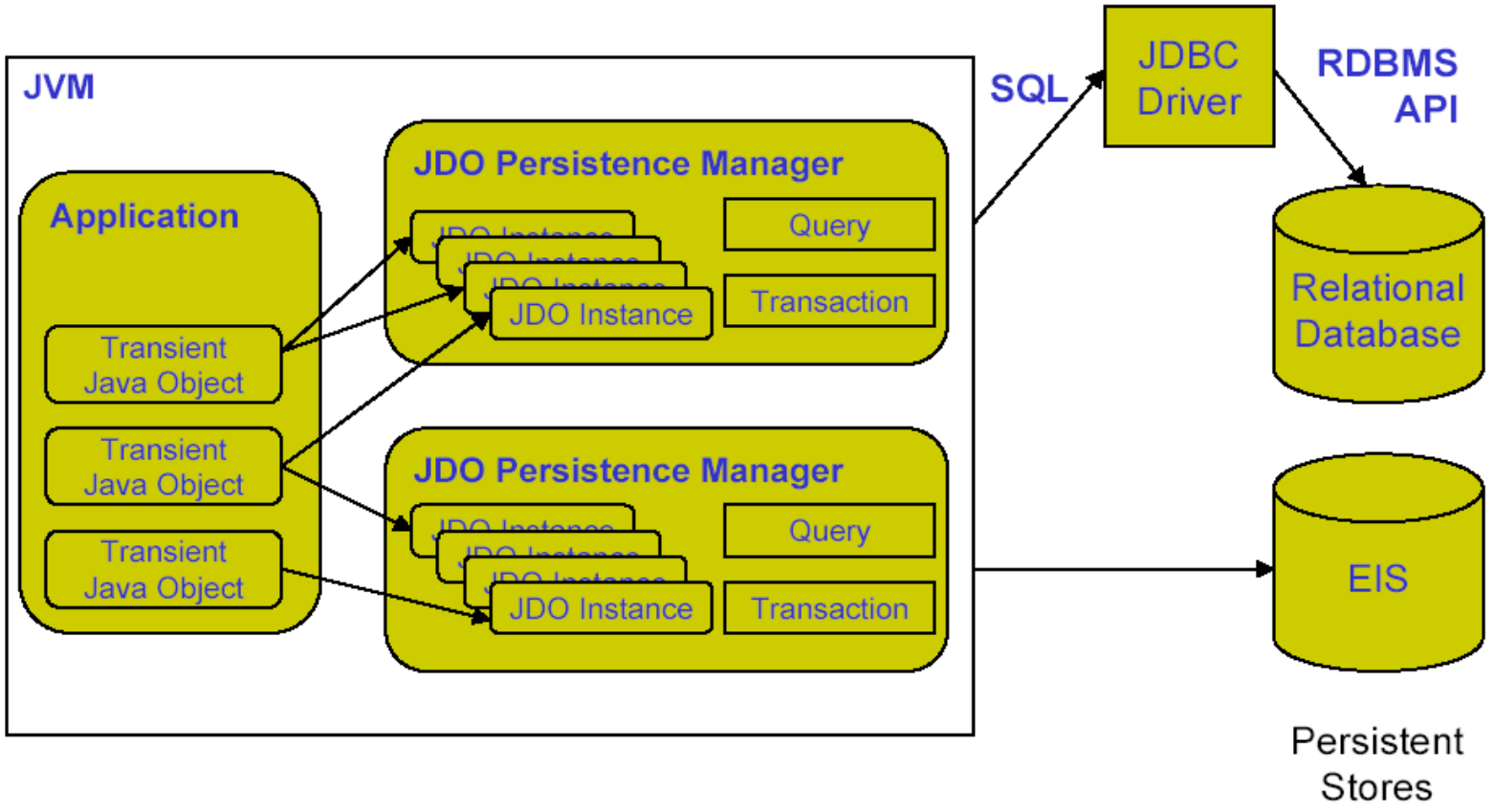
Strategy #3: Persistence Framework



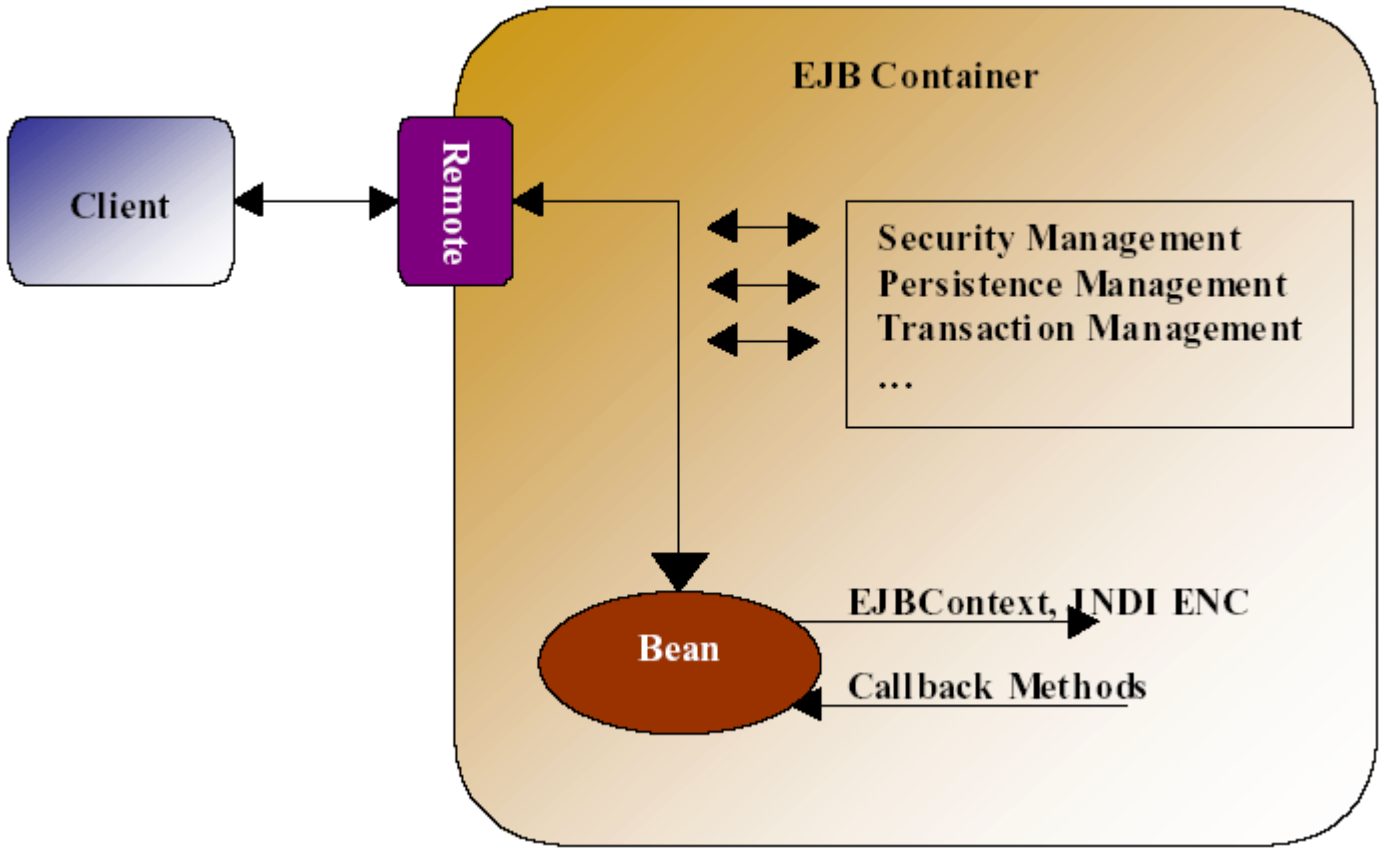
Strategy #4: JDO



JDO Non-Managed Runtime



Strategy #5: EJB CMP



Choosing an Access Strategy

- Which strategy is right for an application depends on several factors

Using Brute Force Approach

Positives:

- Very simple approach
- Can develop code quickly
- Support very bad data models

Negatives:

- Couples application to database
- Developers need to know SQL
- Possibly limits reuse

Using DAO Approach

Positives:

- Data access code encapsulated into a set of classes
- Business classes not coupled to database

Negatives:

- Often platform specific
- Still need to know SQL
- Still a certain amount of coupling

Using Persistence Frameworks

Positives:

- All developers don’t need to know schema or SQL
- If built by experts, probably optimized

Negatives:

- Often proprietary
- If poorly built, probably slower performance

Using EJB CMP

Positives:

- Container independent
- Built by vendors who know persistence

Negatives:

- Not robust for very complex data models
- Have to use entity beans

Using Java Data Objects

Positives:

- Standardized Approach
- Transparent Persistence
- Many vendors support JDO

Negatives:

- Maturity?
- Touches byte code
- JDOQL is not liked by all

Need for Transparent Persistence

- Abstracting out persistence details
- Ability to switch transactional data stores
- Ability to switch persistence approaches

Some Persistence Requirements

- Minimal Intrusion
- Simplicity (not simplistic)
- Support for Transactions
- Support for Managed and Unmanaged environments
- Support for caching, queries, pk generation and mapping tools
- Cohesive API for CRUD

Introducing Object Relational Bridge (OJB)

- Created by Thomas Mahler
- Originally released to the public in October 2000 on SourceForge
- 6,000 downloads per month and about 600 posts to the user mailing list per month
- Now part of Apache DB Project
- Release Candidate 2 (RC2)

OJB Major Influencers

- Craig Larmans “Applying UML and Patterns”
- The Siemens Guys “Pattern-Oriented Software Architecture”
- Scott Amblers classes papers on O/R Mapping techniques
- The “Crossing Chasms” paper from Brown et. Al.
- The GOF Design Patterns

What is OJB?

*A **mapping framework** that allows transparent persistence for **Java Objects** against **relational databases**.*

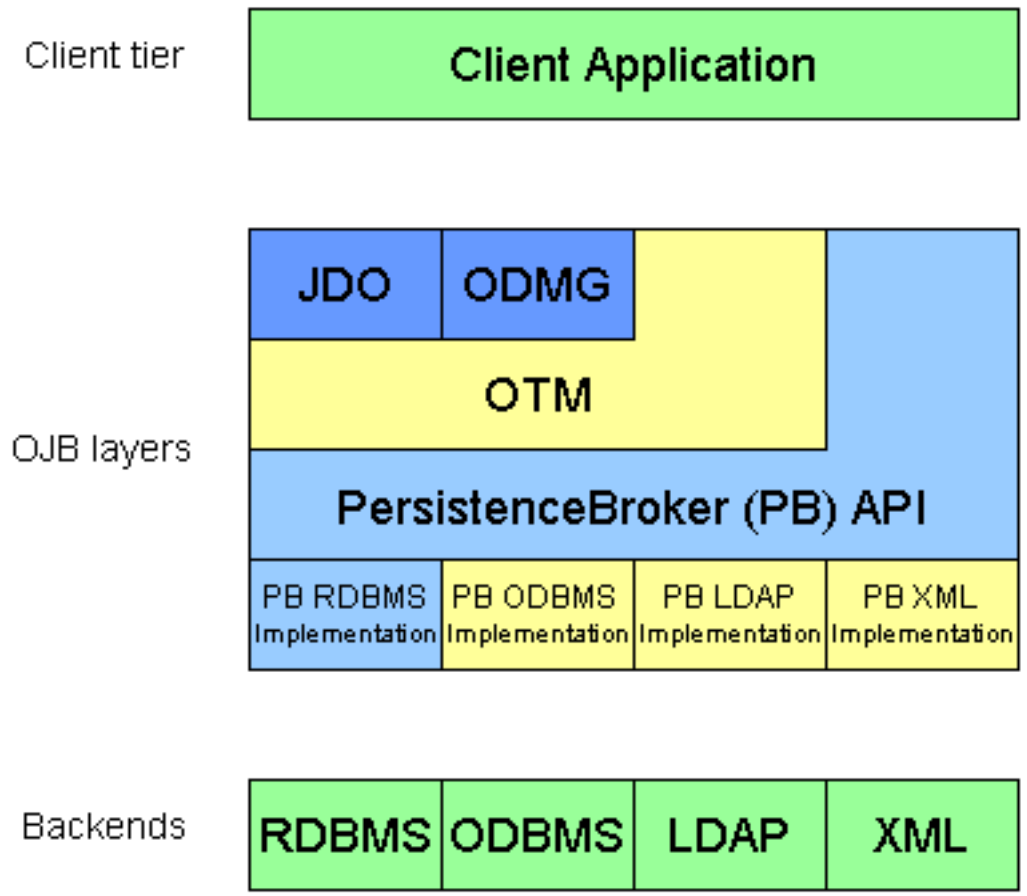
“Some” of the OJB Features

- Transparent persistence
- Persistence by reachability
- Mapping support for 1:1, 1:n and m:n
- Configurable collection queries to control loading of relationships
- Mappings are defined in an XML Repository
- Support for Lazy Materialization
- Support for Caching
- Support for Sequence-Managing
- Support for prefetched relationships
- Support for Pessimistic and Optimistic Locking
- JTA / JCA integration
- Cache synchronization for distributed caches
- 100 %: pure Java, Open Source, Apache License

Support for Multiple APIs

- **PersistenceBroker API**
- A **ODMG 3.0** compliant API
- A Plugin for **JDO**
- An Object Transaction Manager (OTM) layer that contains all features that JDO and ODMG have in common (work in progress)

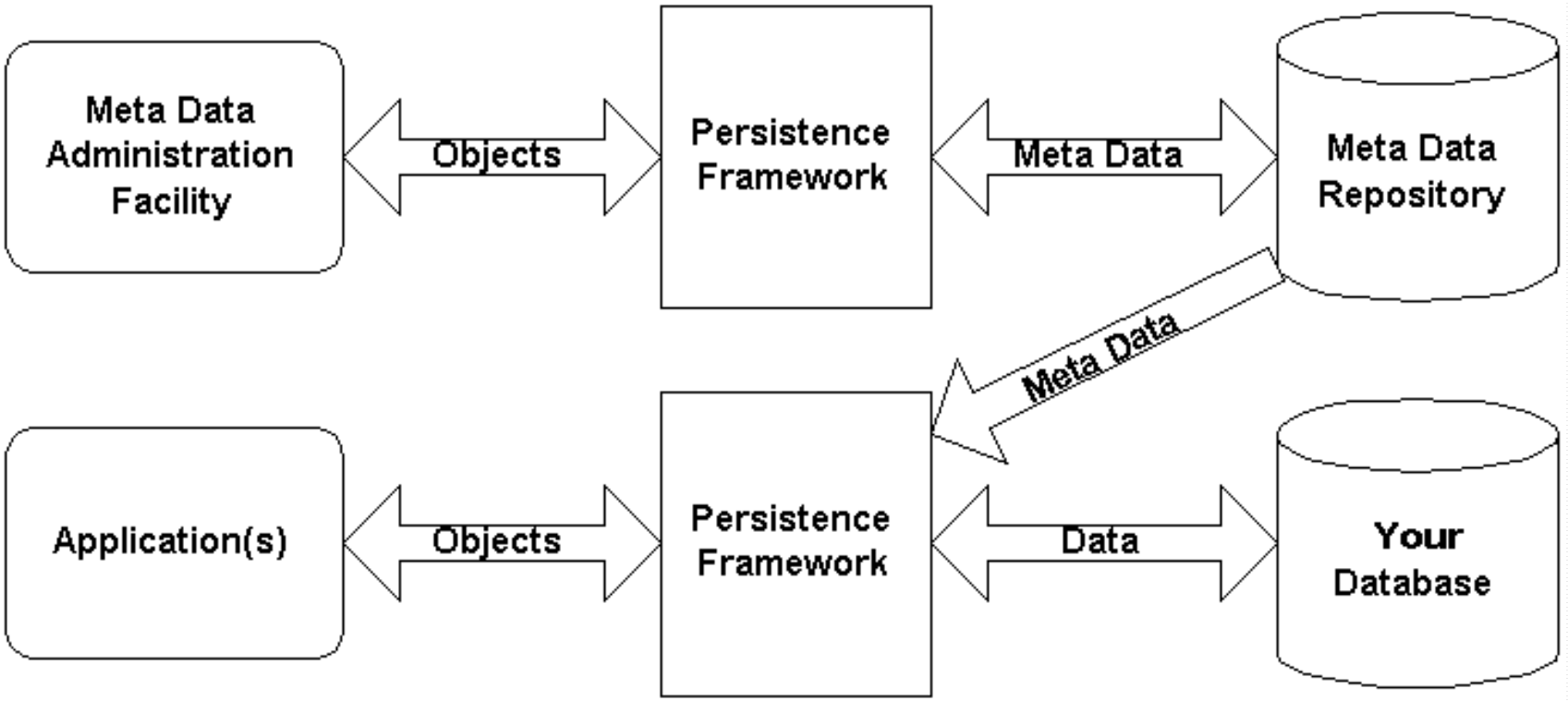
The Layers of OJB



Note:

Layers in Yellow are not implemented yet

The Big Picture



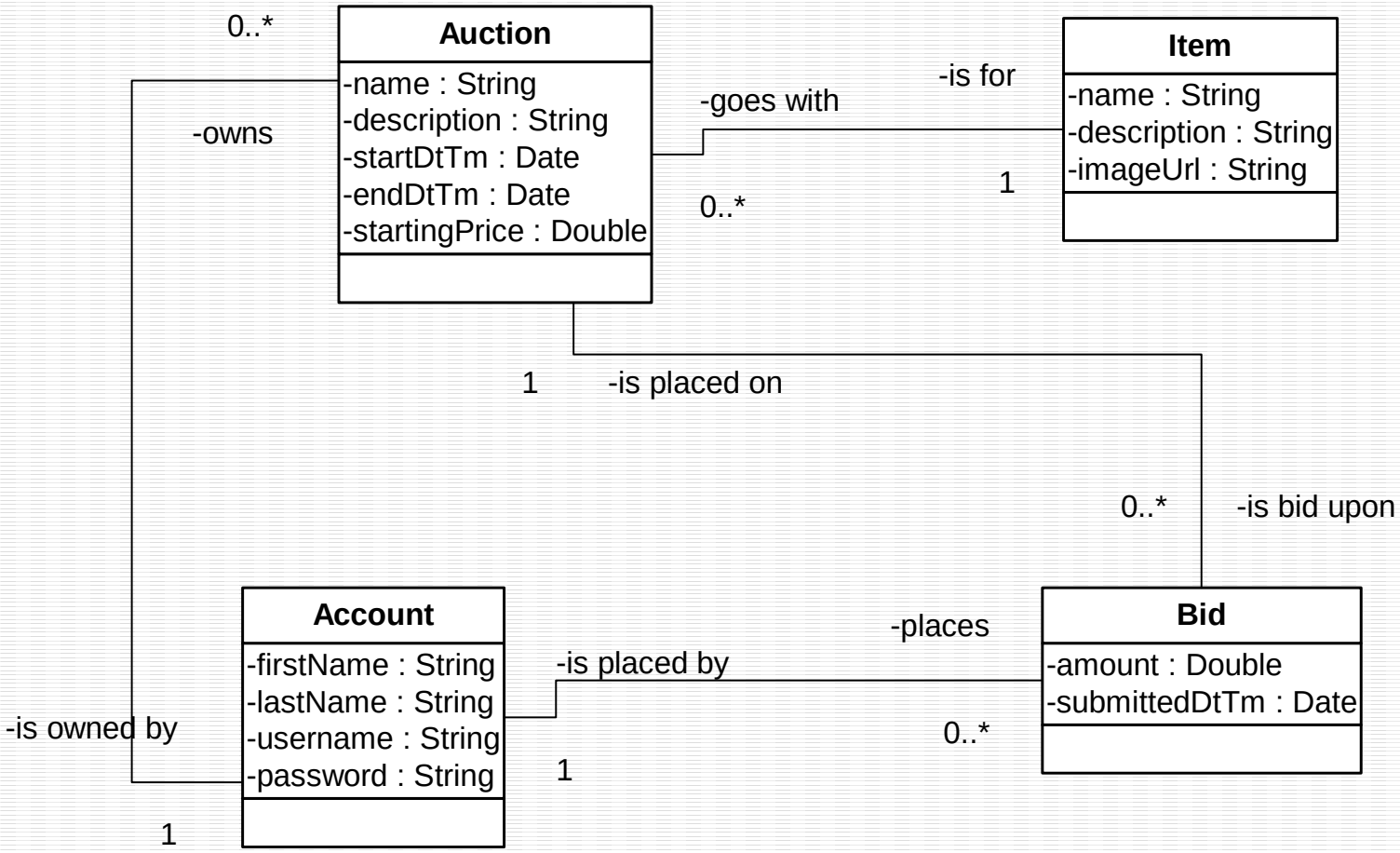
Supported DB Platforms

- HyperSonic (HSQLDB)
- SQL Server
- MySql
- DB2
- Oracle
- MS Access
- Postgresql
- Sybase
- SapDB
- Informix

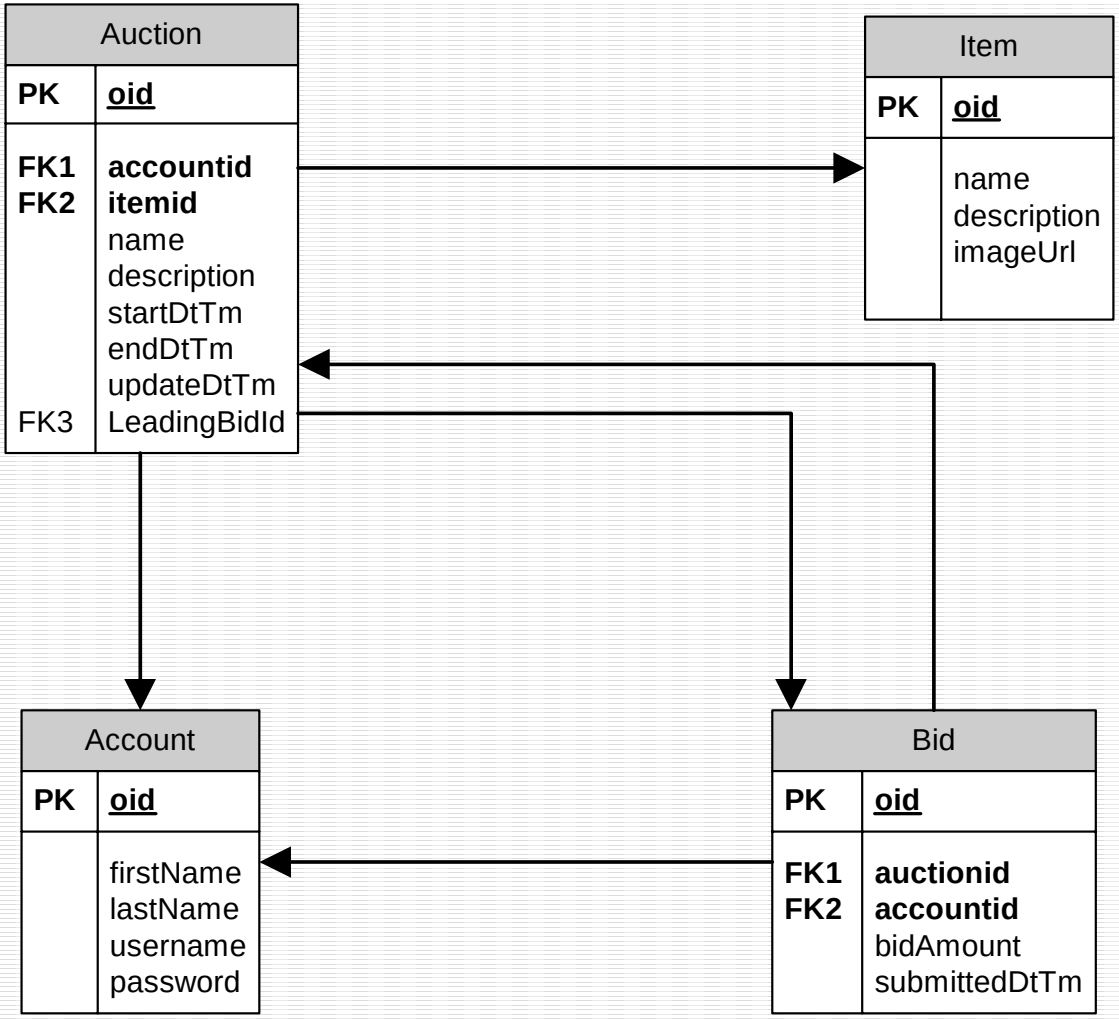
Auction Example

- We'll use the Auction domain model for our discussion

Auction Logical Model



Auction Physical Model



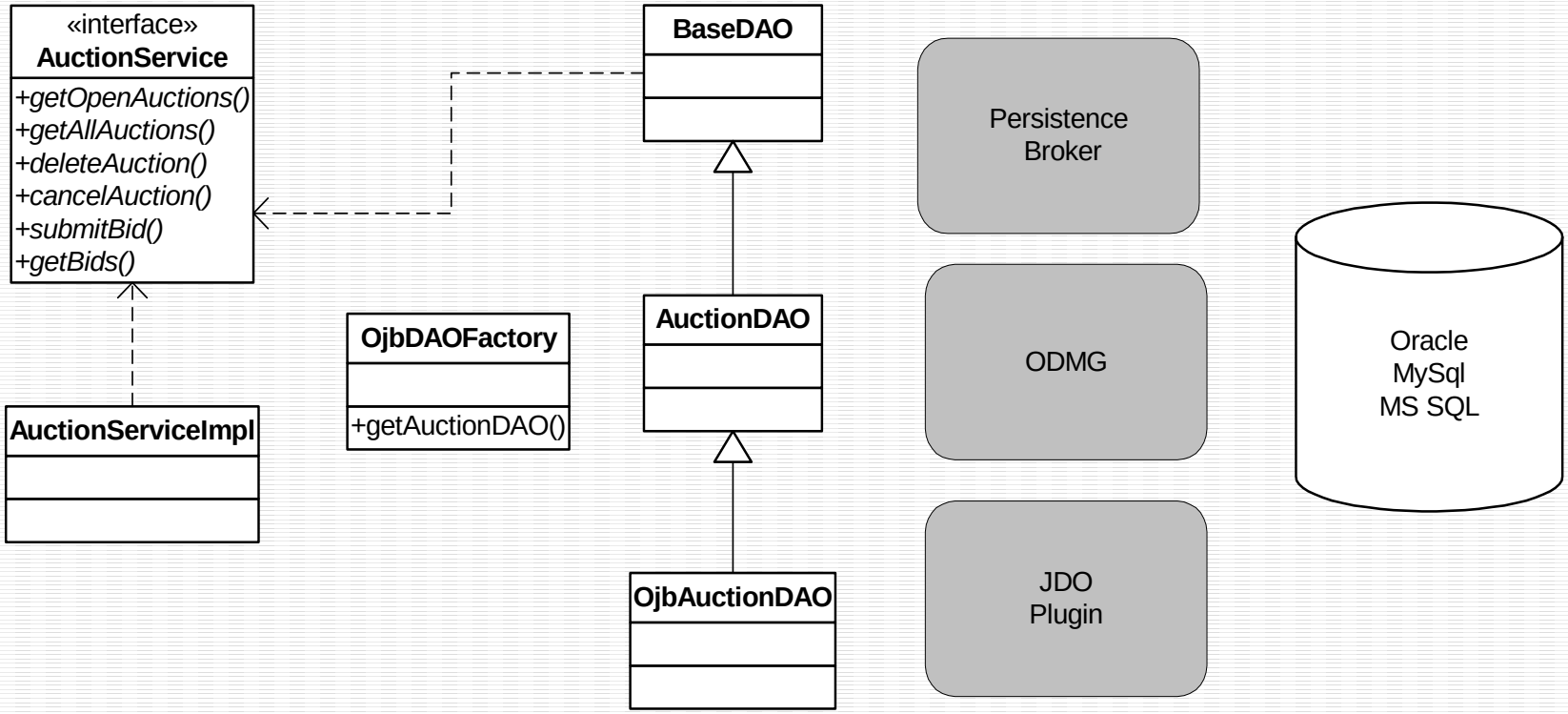
Auction CRUD Functionality

- Create a new Auction
- Retrieve a list open of Auctions
- Submit a bid for an open Auction
- Cancel an Auction

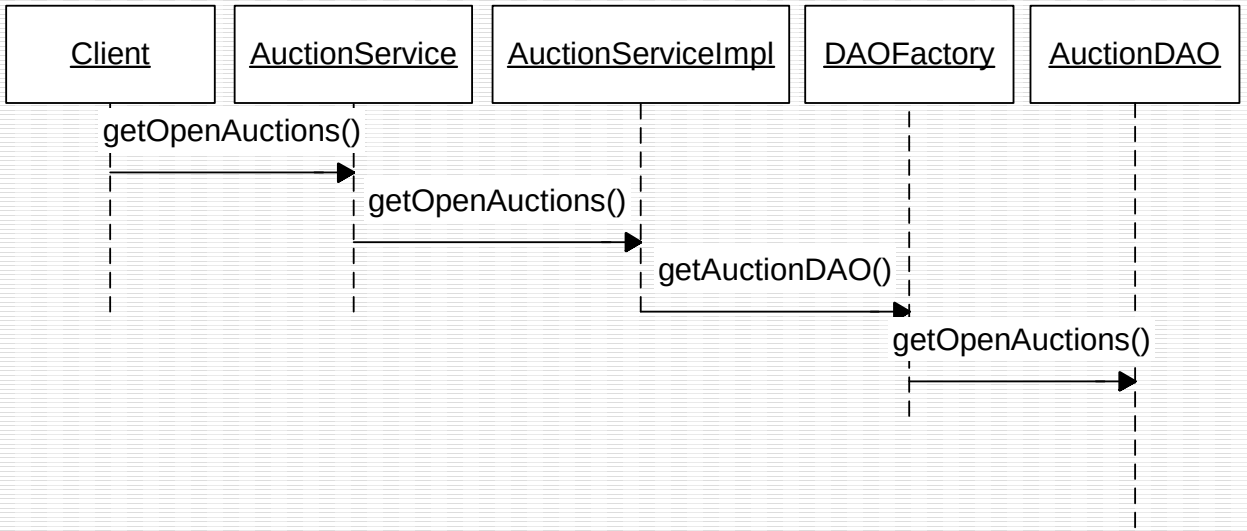
Advanced Auction Functionality

- Paging through a list of Auctions
- Demonstrate Indirection
- Utilize Report Queries
- Write custom SQL

Auction Diagrams



Auction Sequence Diagram



The Mapping Repository

```
<class-descriptor class="com.cavaness.entity.Auction" table="AUCTION">
  <field-descriptor
      name="oid"
      column="OID"
      jdbc-type="INTEGER"
      primarykey="true"
      autoincrement="true" />
…
 <collection-descriptor
      name="bids"
      proxy="true"
      element-class-ref="com.cavaness.entity.Bid"
      auto-delete="true">
      <inverse-foreignkey field-ref="auctionId"/>
    </collection-descriptor>

    <reference-descriptor name="owner" class-ref="com.cavaness.entity.Account">
      <foreignkey field-ref="ownerId"/>
    </reference-descriptor>

    <reference-descriptor name="item" class-ref="com.cavaness.entity.Item">
      <foreignkey field-ref="itemId"/>
    </reference-descriptor>
  </class-descriptor>
```

Generating the Mappings

- Forward Engineering
- Reverse Engineering
- Mapping Proper (Typical Case)

OJB Field Descriptors

- Property Name
- Column Name
- Table Name
- JDBC-Type
- Primary Key
- Nullable
- Indexed
- Auto Increment
- Locking
- Access (readonly, read/write, etc.)
- Conversion
- Length
- Precision
- scale

OJB Properties File

- Let’s look at the file itself!

OJB PersistenceBroker API

The PersistenceBroker provides a minimal API for transparent persistence:

- O/R mapping
- storing (insert, update) of objects to RDBMS
- deleting of objects from RDBMS
- Retrieval of objects with a simple query interface from RDBMS

PB Create Example

```
public Auction createAuction(int ownerId, int itemId, AuctionView view) throws CreateAuctionException {
        Auction newAuction = new Auction();
        newAuction.setName(view.getName());
        newAuction.setDescription(view.getDescription());
        newAuction.setEndDtTm(view.getEndDtTm());
        newAuction.setStartDtTm(view.getStartDtTm());
        newAuction.setStartingPrice(view.getStartingPrice());
        newAuction.setStatus("OPEN");
        newAuction.setItemId(new Integer(itemId));
        newAuction.setOwnerId(new Integer(ownerId));

        PersistenceBroker broker = getPersistenceBroker();
        try {
            broker.beginTransaction();
            broker.store(newAuction);
            broker.commitTransaction();
        } catch ( Exception ex ) {
            broker.abortTransaction();
            throw new CreateAuctionException();
        } finally {
            broker.close();
        }
        return newAuction;
    }
```

PB Retrieve Example

```
public List getOpenAuctions() {
   PersistenceBroker broker = getPersistenceBroker();

   try {
     // Only want "OPEN" auctions that haven't been deleted
     Criteria criteria = new Criteria();
     criteria.addEqualTo("status", AuctionStatus.OPEN.getStatus() );
     criteria.addNotEqualTo("isDeleted", "T");

     // Create a new query with criteria
     QueryByCriteria query = QueryFactory.newQuery(Auction.class, criteria);
     query.addOrderByAscending("name");

     // Execute the query
     Collection openAuctions = broker.getCollectionByQuery(query);
     return new ArrayList(openAuctions);
   }finally {
     broker.close();
   }
 }
```

PB Update Example

```
public void cancelAuction(int auctionId) throws AuctionNotFoundException {
   PersistenceBroker broker = getPersistenceBroker();
   try {
     broker.beginTransaction();
     Auction auction = getAuctionById(broker, auctionId);
     auction.setStatus(AuctionStatus.CANCELED.getStatus());
     broker.store(auction);
     broker.commitTransaction();
   }finally {
     broker.close();
   }
 }

 public List getAllBidsForAuction(int auctionId) throws  AuctionNotFoundException {
   PersistenceBroker broker = getPersistenceBroker();
   try {
     Auction auction = getAuctionById(broker, auctionId);
     return auction.getBids();
   }finally {
     broker.close();
   }
 }
```

PB Delete Example

```
public void deleteAuction(int auctionId) throws AuctionNotFoundException {
       PersistenceBroker broker = getPersistenceBroker();
       Auction auction = null;

       try {
           auction = getAuctionById(broker, auctionId);

           broker.beginTransaction();
           auction.setIsDeleted("T");
           broker.store(auction);
           broker.commitTransaction();
       } finally {
           broker.close();
       }
   }
```

PB Support for Pagination

```
public List getOpenAuctions(int size, int startIndex, String orderByField, boolean ascending) {
      PersistenceBroker broker = getPersistenceBroker();
      Criteria criteria = new Criteria();
      criteria.addEqualTo("status", "OPEN");
      criteria.addNotEqualTo("isDeleted", "T");

      QueryByCriteria query = QueryFactory.newQuery(Auction.class, criteria);
      query.setStartAtIndex(startIndex);

      if ( ascending ) {
          query.addOrderByAscending(orderByField);
      } else {
          query.addOrderByDescending(orderByField);
      }

      query.setStartAtIndex(startIndex);
      query.setEndAtIndex(startIndex + size);

      try {
          return new ArrayList(broker.getCollectionByQuery(query));
      } finally {
          broker.close();
      }
  }
```

Persistence by Reachability

```
public void submitBid(int auctionId, int ownerId, BidView bidView) throws
   AuctionNotFoundException, AccountNotFoundException {

       PersistenceBroker broker = getPersistenceBroker();
       try {
           Auction auction = getAuctionById(broker, auctionId);
           if ( auction.isOpen() ) {
               Bid newBid = new Bid();
               newBid.setBidAmount(bidView.getAmount());
               newBid.setAuctionId(auction.getOid());
               newBid.setSubmittedDtTm(new Timestamp(System.currentTimeMillis()));

               Account account = getAccountById(broker, ownerId);
               newBid.setOwner(account);
               broker.beginTransaction();
               auction.submitBid(newBid);
               broker.store(newBid);
               broker.store(auction);
               broker.commitTransaction();
           }
       } finally {
           broker.close();
       }
   }
```

Using DTOs with OJB

```
public List getAllBidsForAuction( int auctionId ) throws AuctionNotFoundException{
   // Get a collection of Bid business objects for an Auction
   List bids = getAuctionDAO().getAllBidsForAuction(auctionId);

   //Create a BidView DTO for each Bid business object
   List views = new ArrayList();
   Iterator iter = bids.iterator();
   while( iter.hasNext() ){
     Bid bid = (Bid) iter.next();
     views.add(createBidView(bid));
   }
   // Return the Bid DTOs to the caller
   return views;
 }
```

PB Query Framework

- Query for Whole Objects
- Use a ReportQuery for row data

The parts of a Query

- The class of the objects to be retrieved
- A list of criteria
- Additional ORDER BY and GROUP BY

Four Query Factory Methods

- Create criteria to compare a field to a value: ie. addEqualTo("firstname", "tom");
- Create criteria to compare a field to another field: ie. addEqualToField("firstname", "other_field");
- Create criteria to check null value: ie. addIsNull("firstname");
- Create a raw sql criteria: ie: addSql("REVERSE(name) like 're%'");

Other Query Functionality

- In/not
- And and or’s
- Ordering and Grouping
- Joins
- Raw SQL

Comparing Fields to a Value

- addEqualTo
- addLike
- addGreaterOrEqualThan
- addGreaterThan
- addLike
- addBetween , this methods has two value parameters
- addIn , this method uses a Collection as value parameter
- and of course there negative forms

PB Query Example

```
public List getAllAuctions() {
       PersistenceBroker broker = getPersistenceBroker();

       try {
           // Create a criteria to filter out auctions which don't have
           // a status of "OPEN".
           Criteria criteria = new Criteria();
           criteria.addEqualTo("status", "OPEN");
           criteria.addNotEqualTo("isDeleted", "T");

           // Create a new query with criteria
           QueryByCriteria query = QueryFactory.newQuery(Auction.class, criteria);
           query.addOrderByAscending("name");

           // Execute the query
           Collection openAuctions = broker.getCollectionByQuery(query);
           return new ArrayList(openAuctions);
       } finally {
          broker.close();
       }
   }
```

Using ReportQueries

```
Criteria crit = new Criteria();
Collection results = new Vector();
ReportQueryByCriteria q = QueryFactory.newReportQuery( Auction.class, crit);

// define the 'columns' of the report
q.setColumns(new String[] {
  “name",
  “max(bids.bidAmount)",
  “min(bids. bidAmount)" });
q.addGroupBy(“name");
Iterator iter = broker.getReportQueryIteratorByQuery(q);
```

Instance Callbacks

- Positive: Allows persistent objects to better interact with the persistence framework
- Negative: Persistent classes must implement OJB specific interface

The PersistenceBrokerAware Interface

```
public interface PersistenceBrokerAware {
  public void beforeStore() throws PersistenceBrokerException;
  public void afterStore() throws PersistenceBrokerException;
  public void beforeDelete() throws PersistenceBrokerException;
  public void afterDelete() throws PersistenceBrokerException;
  public void afterLookup() throws PersistenceBrokerException;
}
```

Field Conversions

- OJB allows you to use pre-defined or your own customized Field Conversions
- Example: Converting int db field into a boolean
- OJB comes with a set of pre-defined conversion routines

Using Proxy Objects

- Allows for lazy loading (aka. Lazy Materialization)
- Supports Single Object Proxies or Collection Proxies
- Supports Java 1.3 Dynamic Proxies

Why choose PB API?

- Want maximum performance and flexibility
- Not bound to a particular API (like JDO)
- Willing to implement some higher-level persistence functionality on your own

OJB ODMG API

- OJB Provides a ODMG 3.0 Implementation
- The ODMG is a consortium of vendors and interested parties that work on specifications for object database and object-relational mapping products
- OJB's ODMG implementation is built on top of the PersistenceBroker (for now)

Features of OJB's ODMG Implementation

- Real Object Transactions
- Complete OQL Query API
- Locking Mechanism for Concurrent Threads, including isolation levels

Internal OJB Tables

- **OJB_HL_SEQ** - High/low sequence manager.
  OJB_LOCKENTRY - Used to store Object locks if the LockManager is run in distributed mode. Not needed in singlevm mode.
- **OJB_NRM** - The "Named Roots Map". ODMG allows to bind persistent objects to an user defined name. If bind() and lookup() are not used in client apps, this table is not needed.
- **OJB_DLIST** - Used for the ODMG persistent DList collections.
- **OJB_DLIST_ENTRIES** - Stores the entries of DLists (a wrapper to objects stored in the Dlist.
- **OJB_DSET** - Used to store ODMG persistent DSET collections.
- **OJB_DSET_ENTRIES** - Stores the entries of DSets.
- **OJB_DMAP** - Used to store the ODMG persistent DMap tables.
- **OJB_DMAP_ENTRIES** - The table containing the DMap entries.

ODMG Create Example

```
public Auction createAuction(int ownerId, int itemId, AuctionView view) throws
     CreateAuctionException, AccountNotFoundException, ItemNotFoundException {

   Auction auction = new Auction();
   auction.setName( view.getName() );
   auction.setDescription( view.getDescription() );
   auction.setStartDtTm( view.getStartDtTm() );
   auction.setEndDtTm( view.getEndDtTm() );
   auction.setStartingPrice( view.getStartingPrice() );
   auction.setStatus( AuctionStatus.OPEN.getStatus() );

     Account owner = getAccountById( ownerId );
     auction.setOwner( owner );
     Item item = getItemById( itemId );
     auction.setItem( item );

     Transaction tx = getNewTransaction();
     tx.begin();
     tx.lock( auction, Transaction.WRITE );
     tx.commit();

     // Return the newly created Auction so the client can have the OID
     return auction;
 }
```

ODMG Retrieve Example

```
public List getOpenAuctions() {
   OQLQuery query = createQuery();
   List allAuctions = null;

   try {
     String className = Auction.class.getName();
     StringBuffer buf = new StringBuffer();
     buf.append("select openAuctions from ");
     buf.append(className);
     buf.append(" where status = $1");
     buf.append(" and DELETED = $2");

     query.create(buf.toString());
     query.bind(AuctionStatus.OPEN.getStatus());
     query.bind("");

     allAuctions = (DList) query.execute();
   }catch (Exception ex) {
     logger.error( "Exception thrown in getOpenAuctions()", ex );
   }
   return allAuctions;
 }
```

ODMG Update Example

Optimistic Locking Example

```
public void cancelAuction(int auctionId) throws AuctionNotFoundException {
   Transaction tx = getOdmg().newTransaction();
   try{
     tx.begin();
     Auction auction = getAuctionById(auctionId);
     auction.setStatus(AuctionStatus.CANCELED.getStatus());
     tx.commit();
   }catch( OptimisticLockException ex ){
     logger.error( "OptimisticLockException", ex );
     throw ex;
   }
 }
```

Pessimistic Locking Example

```
public void cancelAuction(int auctionId) throws AuctionNotFoundException {
   Transaction tx = getOdmg().newTransaction();
   try{
     tx.begin();
     Auction auction = getAuctionById(auctionId);
     tx.lock( auction, Transaction.WRITE );
     auction.setStatus(AuctionStatus.CANCELED.getStatus());
     tx.commit();
   }catch( org.odmg.LockNotGrantedException ex ){
     logger.error( "LockNotGrantedException", ex );
     throw ex;
   }
 }
```

ODMG Delete Example

```
public void deleteAuction(int auctionId) throws AuctionNotFoundException {
   Transaction tx = getNewTransaction();

   try{
     tx.begin();
     Auction auction = getAuctionById(auctionId);
     tx.lock(auction, Transaction.WRITE);
     auction.setStatus(AuctionStatus.CANCELED.getStatus());
     tx.commit();
   }catch( org.odmg.LockNotGrantedException ex ){
     logger.error( "LockNotGrantedException", ex );
     throw ex;
   }
 }
```

Why choose the ODMG API?

- Slightly more robust than PB
- Based on a standard
- Provides OQL

What is Java Data Objects(JDO)?

JDO is a standard that allows for transparent persistence for Java applications.

JDO Facts

- First version, toolkit and reference implementation released April 2002
- Many vendor and a few open source implementations available

OJB's JDO API

- OJB doesn't provide its own implementation of JDO yet
- Currently, a plugin to the JDO reference implementation is provided
- The plugin is called OjbStore

Proposed Benefits of JDO

- Portability
- Database Independence
- Ease of Use
- Increased Performance
- Integration with EJB

OJB's JDO Implementation

- Don't use it yet!

OJB Extensibility

- OJB has many extension points
- OJB.properties and repository.xml are access points for customization

Support for Caching

- Supports several types of Caching implementations
- Objects are cached on load or store
- Default cache maintains object unqiueness
- Supports CacheFilters to filter out Classes or Packages
- Plugin your own or other third-party cache implementations

Sequence Manager Support

- Support several SequenceManager implementations
- Can use native sequencing on supported platforms (Oracle, PostgreSQL,etc)
- Can also add your own implementation

The LockingManager

- OJB supports Optimistic and Pessimistic Locking

Debugging and Troubleshooting

- Logging controlled through Ojb.properties file
- P6Spy can be used easily with OJB

Deploying OJB

1. Add the OJB jar to classpath.
2. Configure and add the `ojb.properties` and `repository.xml` files.
3. Add the dependant JARs (Jakarta Commons, log4j, etc).
4. Install the JDBC Driver.

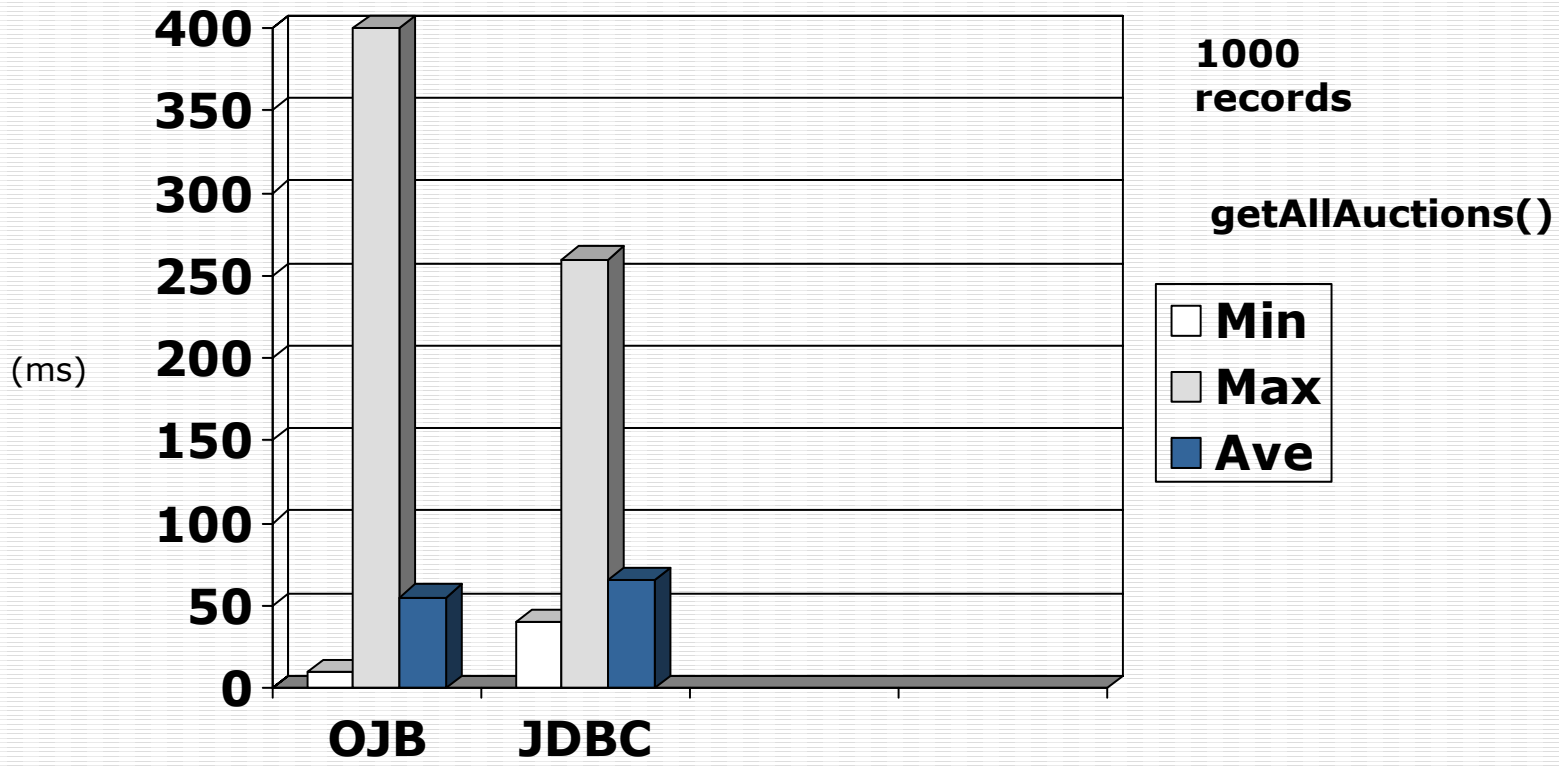
But what about Performance?

- O/R Tools are designed to hide ugly details
- Price of using O/R **MAY** be performance
- Software Architects need to determine on a case-by-case basis

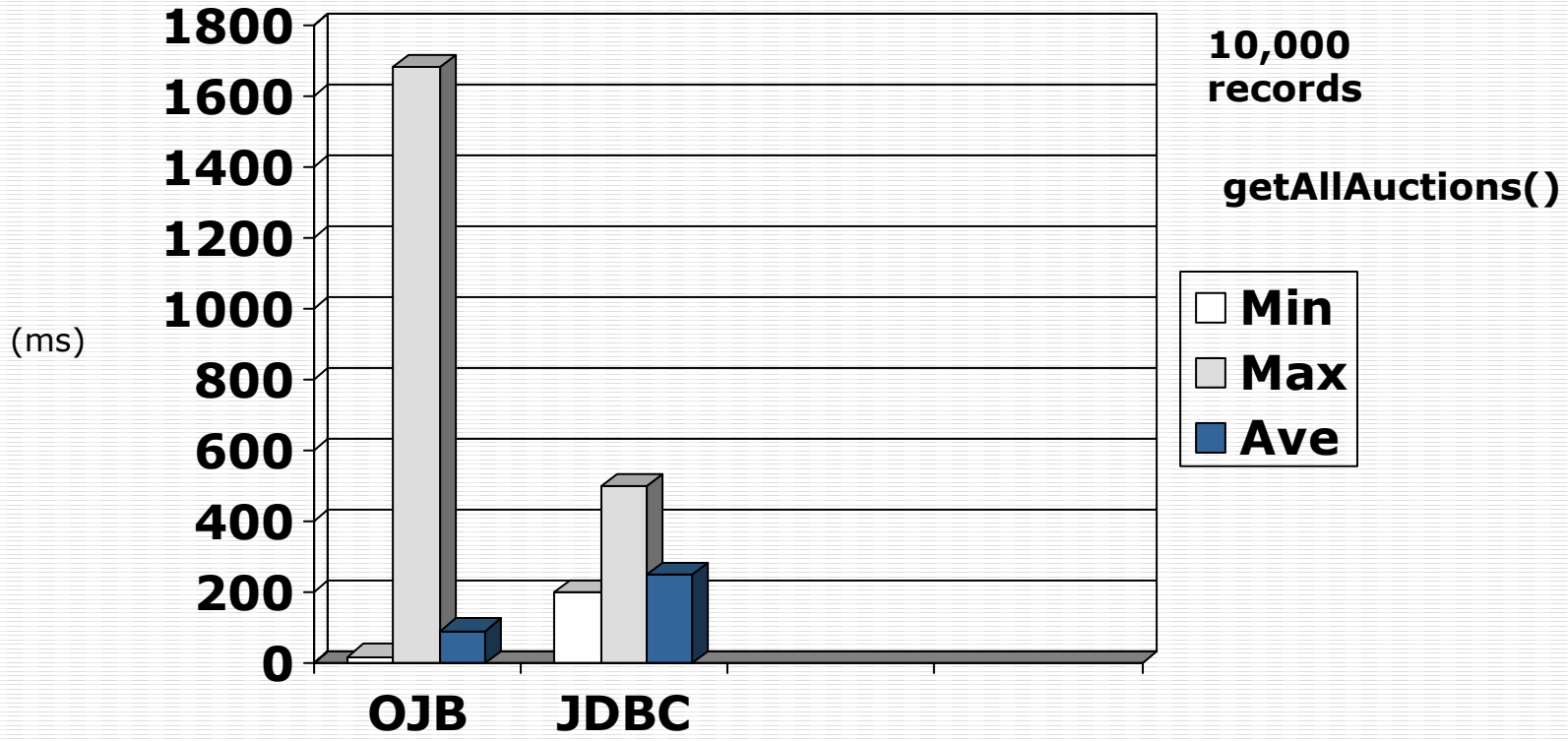
Unscientific test on Auction Ex.

- 1000/10,000 Auction Records
- Calling getOpenAuctions()
- Calling method 100 X and Averaging
- Comparing JDBC & OJB

Performance Results



Performance Results



Performance isn’t Everything

- Usability of the supported API's
- Flexibility of the framework
- Scalability of the framework
- Community support
- Other Benefits…

Performance Test Suite

- OJB Comes with a Performance Test Suite
- Allows you to compare performance against native JDBC
- Supports various RDBMS
- Test are integrated into OJB’s build scripts

OJB Competitors

- Hibernate on Sourceforge (http://sourceforge.net/projects/hibernate)
- Toplink from Oracle (http://otn.oracle.com/products/ias/toplink)
- Cocobase from Thought Inc. (http://www.cocobase.com)
- Castor Project (http://www.castor.org/)
- And there are others…

Resources and Further Reading

- OJB Project Site (http://db.apache.org/ojb)
- OJB Mailing Lists (http://db.apache.org/ojb/mail-lists.html)
- Scott Ambler's Online Writings (http://www.ambysoft.com/onlineWritings.html)

The Future of OJB

- Full JDO Compliant Interface
- Graphical Mapping Tool
- Support for EJB 2.0 Query Language???

References

- Scott Ambler (http://www.ambysoft.com)

The End…

Q & A