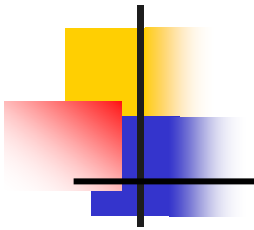
# Using RUBiS to Compare JDO with JDBC and Entity Beans

David Jordan

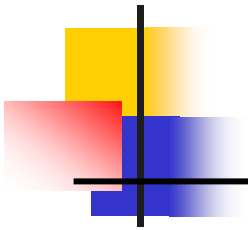
Object Identity, Inc.

www.objectidentity.com



# Presentation Overview

- What is RUBiS?
  - Data model
  - Architectures
- The JDO implementation of RUBiS
- Examine source of a session bean
  - Illustrate use of JDO
  - Compare with JDBC and Entity Beans
- Comparing the technologies



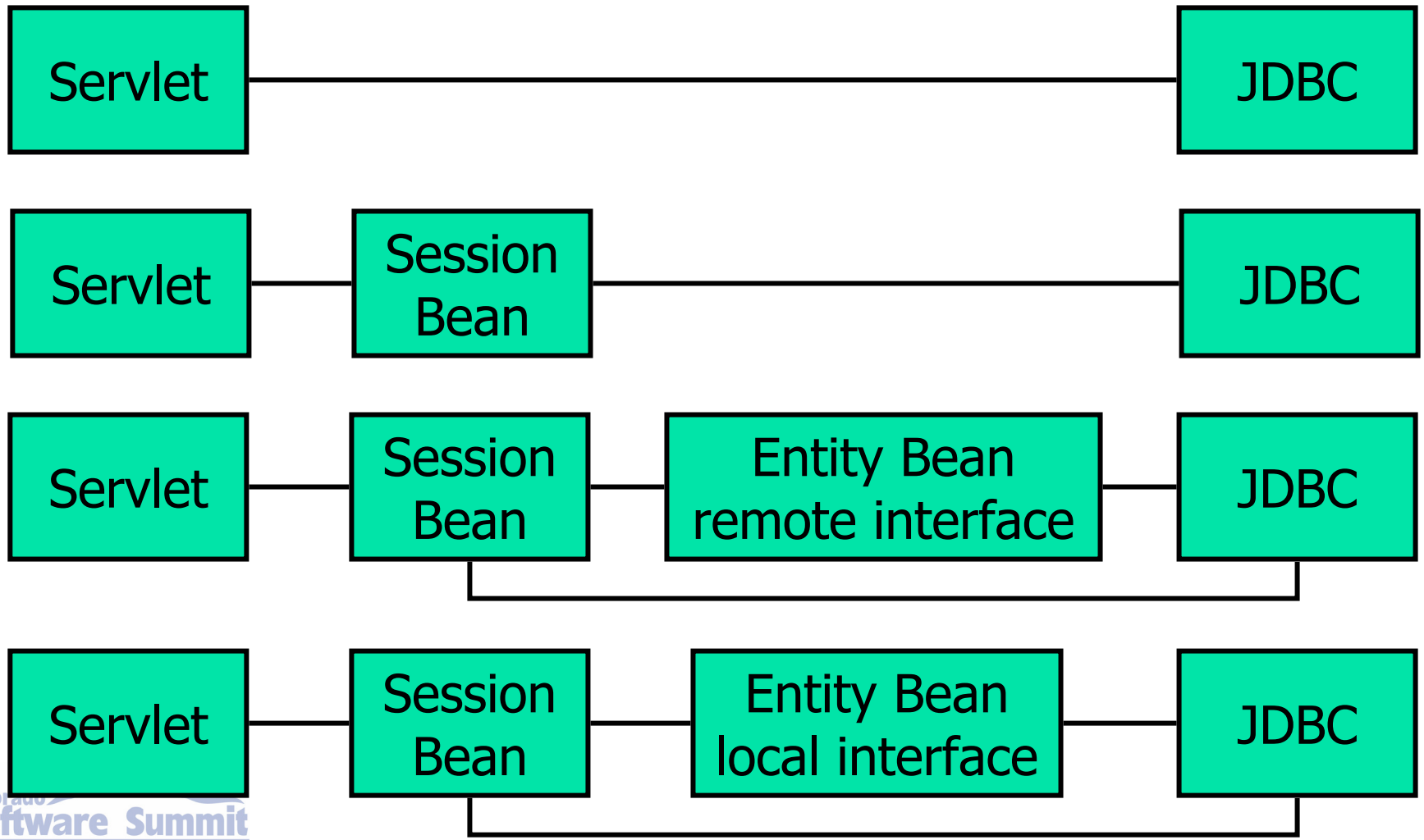
# What Is RUBiS?

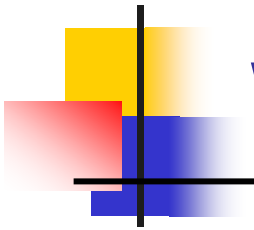
- Benchmark first developed at Rice University
- Application emulates auction site like Ebay
- Goal is to evaluate
  - Application design patterns
  - Performance scalability of application servers
- Application functionality implemented using several enterprise architectures
- Now managed by Inria (www.inria.fr)
- Web site: (rubis.objectweb.org)

# No Performance Results

- Inria's DB server capabilities inadequate
  - Due to our need to use InnoDB tables
  - Database server was very CPU bound
- Performance of EJB and JDBC were nearly identical due to database server bottleneck
- Inria decided useful results required more capable (multiprocessor) database server
- Inria and vendors unwilling to pay for this
- Project was cancelled before its completion

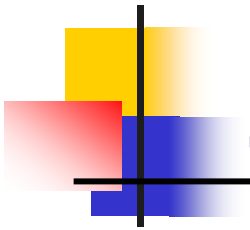
# Some Current Architectures





# Why Do RUBiS with JDO?

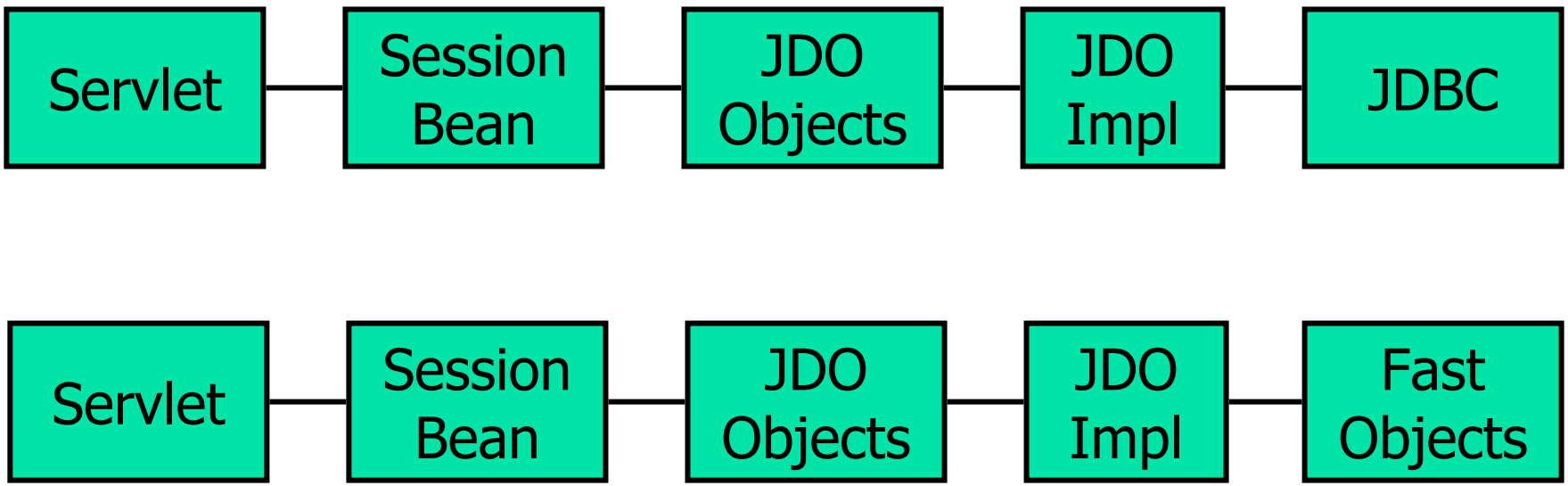
- JDO community wanted a benchmark to compare JDO with JDBC and EJB
- RUBiS provides a comparison of various J2EE architectures
- RUBiS has gained respect/recognition
- JDBC and EJB work already done
  - Less work to do
  - Cannot claim biases in implementation

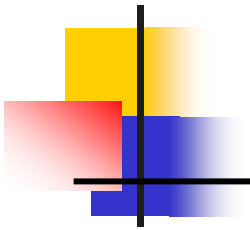


# JDO Vendors Participating

- FastObjects, t7 (object database)
- Hemisphere Technologies, JDO Genie
- LIBeLIS, LiDO
- ObjectFrontier, Frontier Suite for JDO
- Object Industries, JRelay

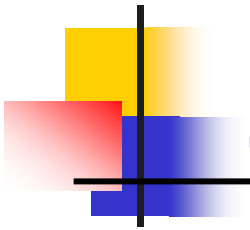
# JDO RUBiS Architectures





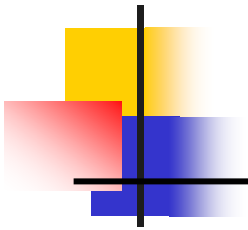
# RUBiS Database&Transactions

- Supported MySQL and Postgres databases
  - Different SQL code required for each
- MyISAM table type used for MySQL
  - Non-transaction safe (no transaction overhead)
  - Faster, uses less memory and disk space
  - Cannot combine operations in atomic operation
    - Can get partial update of a transaction if crash occurs
    - Cannot rollback a complete set of operations, *etc.*



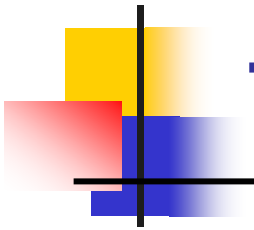
# JDO Database/Transactions

- JDO implementations require
  - Transaction safe facilities
  - Apply multiple operations atomically
  - Failure does not result in partial results in DB
- MySQL InnoDB provides transaction support
  - JDO vendors require InnoDB for MySQL support
- Locking granularity
  - MySQL InnoDB uses row-level locking
  - FastObjects uses object-level locking



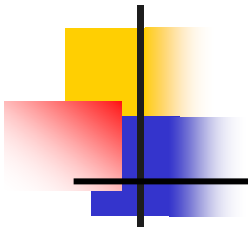
# JDO Transaction Types

- Pessimistic (required feature in JDO)
  - Locks acquired when first accessed and held
  - Reads involve "select for update"
- Optimistic (optional feature)
  - Optional feature in JDO
  - Concurrency conflicts checked at end of tx
  - Not supported by FastObjects at project start
- Nontransactional Read/Write (optional)
  - Not widely supported



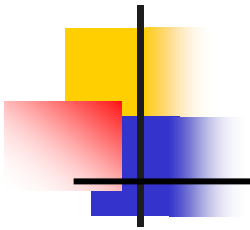
# Transaction Decision(s)

- Initially chose pessimistic transactions
  - Had significant lock conflicts and deadlocks
  - FastObjects initially did not support optimistic
- Migrated to optimistic transactions
  - FastObjects has recently added support
  - Provides a closer level of locking equality with the other RUBiS implementations



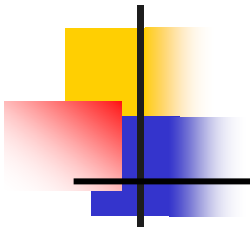
# Transaction granularity

- Other RUBiS architectures were run using InnoDB tables
  - But each business method performed multiple operations that were not atomically applied
  - Still had lack of atomicity
  - Smaller lock durations
- JDO nontransactional read/write would have most closely matched current RUBiS access
  - But not widely supported yet
  - Resulting robustness/integrity is poor



# How JDBC, EJB, & JDO Differ

- Data model
  - JDBC: relational model, no objects
  - EJB: distributed component model
  - JDO: Java classes
- Query language
  - JDBC: SQL
  - EJB: SQL in EJB 1.x, EJBQL in EJB 2.x
  - JDO: JDOQL, uses Java model and operators

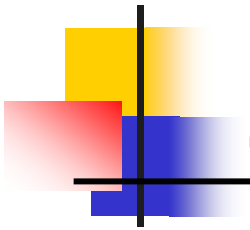


# Form and Access of Data

- JDBC
  - Application copies column values to variables
  - No cache management, redundant copies
- EJB
  - Interface handles to potentially remote objects
  - Cache and location managed by EJB server
- JDO
  - Application has direct access to Java objects
  - Cache management:  one copy per transaction

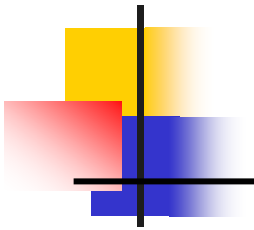
# Entity Bean Bad Practices

- *Core J2EE Patterns: Best Practices and Design Strategies*, by Alur, Crupi, Malks
- Map object model directly to entity beans
- Map relational model to entity bean
- Using entity beans as fine-grained objects
- General issue: Entity Beans should be
  - Coarse-grained components with high-level operations, efficiently invoked remotely
  - Many do not design entity beans properly
    - Cause of many entity bean project failures



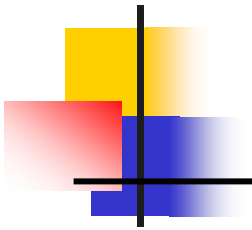
# JDO's Primary Objective

- JDO is designed to directly map fine-grained Java object models to underlying datastores
- Persistence, transactions, queries, cache management
- JDO designed to map directly to relational and object databases
  - Insulates application from underlying datastore
- JDO can be used with Session Beans in an EJB environment



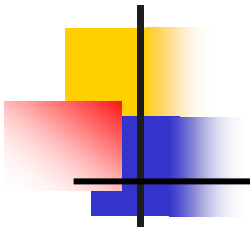
# RUBiS Schema and Row Count

- Users: 1,000,848
- Items: 33,721
- Old_items: 500,000
- Categories: 20
- Regions: 62
- Bids: 330,000 (avg. of 10 per item)
- Buy_now: 1882
- Comments: 533,426



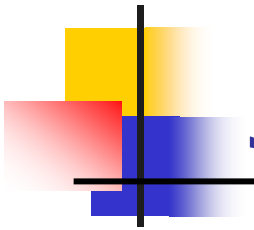
# RUBiS' Original Users Table

```
CREATE TABLE users (
   id              INTEGER UNSIGNED NOT NULL UNIQUE
                           AUTO_INCREMENT,
   firstname       VARCHAR(20),
   lastname        VARCHAR(20),
   nickname        VARCHAR(20) NOT NULL UNIQUE,
   password        VARCHAR(20) NOT NULL,
   email           VARCHAR(50) NOT NULL,
   rating          INTEGER,
   balance         FLOAT,
   creation_date DATETIME,
   region          INTEGER UNSIGNED NOT NULL,
   PRIMARY KEY(id)
);
```



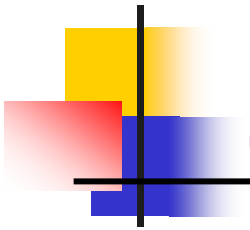
# Some Changes for JDO

- UNSIGNED INTEGER and FLOAT
  - One JDO vendor did not support the mapping of Java int and float to these unsigned SQL types
- AUTO_INCREMENT
  - Used for generating unique ids at database level
  - Not supported by JDO vendors
- Many JDO implementations use a versioning column to support optimistic transactions



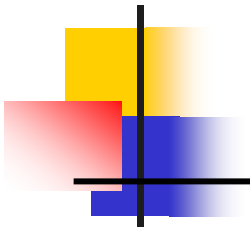
# JDO Users Table

```
CREATE TABLE users (
    id              INTEGER NOT NULL UNIQUE,
    firstname       VARCHAR(20),
    lastname        VARCHAR(20),
    nickname        VARCHAR(20) NOT NULL UNIQUE,
    password        VARCHAR(20) NOT NULL,
    email           VARCHAR(50) NOT NULL,
    rating          INTEGER,
    balance         FLOAT,
    creation_date DATETIME,
    region          INTEGER NOT NULL,
    version         INTEGER,
    PRIMARY KEY(id)
) TYPE = InnoDB;
```



# JDO Identity

- Datastore identity
  - JDO impl. or database provides unique identity
- Application identity
  - Application provides unique identity based on value of one or more fields
- Initially decided to use datastore identity
  - Supported by all vendors
  - FastObjects does not support application identity

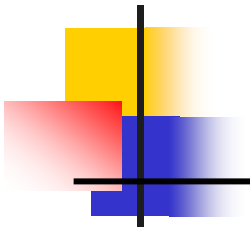


# Identity and RUBiS

- RUBiS application code makes heavy use of application identity values
  - Foreign key values returned to client tier by one session bean used in subsequent session bean
- Could not leverage use of datastore identity
  - Also resulted in redundant identity columns where the key column still used for lookup
- Design decision change
  - Use application identity for relational impl.
  - Use datastore identity in FastObjects

# JDO RUBiS class: User

```
public class User {
  private int     id;
  private String  firstName;
  private String  lastName;
  private String  nickName;
  private String  password;
  private String  email;
  private int     rating;
  private float   balance;
  private Date    creationDate;
  private Region  region;
  private HashSet bids;              // Bid.user is this User
  private HashSet commentsToUser;  // Comment  (Comment.to)
. . .
```



# Use of inverse

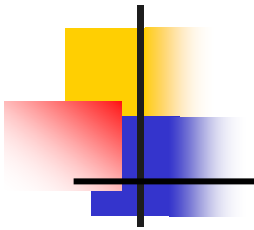
- Foreign key represents a pseudo-collection
  - Bid.user references a user
  - Multiple bids can reference the same user
  - For given user, there is set of associated bids
- inverse allows use of collection in User class
- JDO implementation maintains other side of relationship when one side is modified
- inverse not part of JDO, but often necessary to use a collection with legacy schema

# JDO Metadata: User class

```
<class name="User" identity-type="application"
                   objectid-class="edu.rice.rubis.User$Id">
  <extension vendor-name="X" key="table" value="users"/>
  <extension vendor-name="X" key="optimistic-locking">
    <extension vendor-name="X" key="column" value="version"/>
  </extension>
  <field name="id" primary-key="true">
    <extension vendor-name="X" key="column">
      <extension vendor-name="X" key="name" value="id"/>
    </extension>
  </field>
  <field name="firstName">
    <extension vendor-name="X" key="column">
     <extension vendor-name="X" key="name" value="firstname"/>
    </extension>
  </field>
```

# User class metadata *(Continued)*

```
<field name="region">
  <extension vendor-name="X" key="column">
    <extension vendor-name="X" key="name" value="region"/>
  </extension>
</field>
<field name="bids">
  <collection element-type="edu.rice.rubis.Bid">
    <extension vendor-name="X" key="inverse" value="user"/>
  </collection>
</field>
<field name="commentsToUser">
  <collection element-type="edu.rice.rubis.Comment">
    <extension vendor-name="X" key="inverse" value="to"/>
  </collection>
</field>
</class>
```

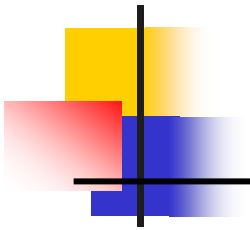


# Items and Old_items tables

- Tables represent items
  - Have identical form with same columns
  - Physically partitioned to provide fast access to current items
- Three tables have item_id foreign key
  - bids and comments reference items or old_items
  - buy_now references items
- Current RUBiS code used item_id to lookup row in either items or old_items tables

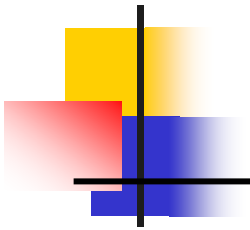
# Map Foreign Keys to References

- Foreign key usually maps to Java reference
- Item and OldItem classes used to represent items and old_items tables
- item_id column referenced two tables
- No means of having a Java reference to more than one table
- Could have managed as a column value
  - But makes model more relational, less Java



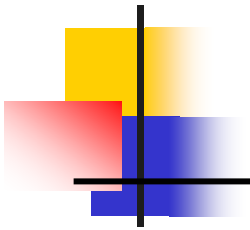
# References *vs.* Foreign Keys

- RUBiS made extensive use of primary and foreign keys
- In SQL, foreign key field can be set without establishing object rep. in memory
- In Java, this is represented by a reference to the object
- `PM.getObjectById(`*`objectid`*`, false)` can be used to construct the instance in memory without loading its state from the database



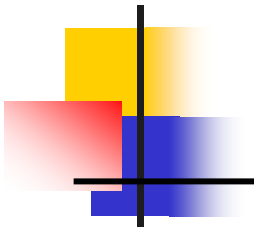
# Collections

- Application functionality benefited from having a collection of Bid references in Item and OldItem classes
- Could not use a collection with an inverse (Item.bids collection) unless field at other end was a reference to an Item (Bid.item)
- Added old_item_id column to reference old_items table
  - One of item_id, old_item_id columns is null



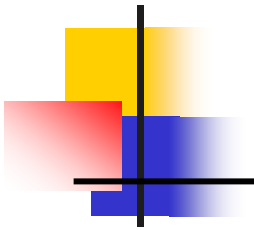
# User Representation in EJB

- User.java
  - Interface defines get/set methods for attributes
  - Not necessary in JDO
- UserBean.java
  - Corresponds to JDO User class
  - EJB requires get/set methods for each attribute
- UserPK.java
  - Corresponds to JDO application identity class
  - Equivalent methods/functionality



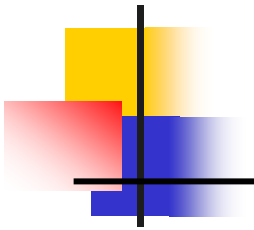
# User Representation in EJB

- UserHome.java
  - Defines create and query methods
  - We place equivalent methods in JDO User class
- ejb-jar.xml
  - Describes entity beans, not necessary in JDO
- jaws.xml
  - Mapping of bean attributes to columns
    - Similar to JDO metadata
  - Primitive expressions for finder methods



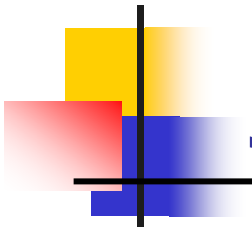
# Relationship Representation

- Foreign keys in relational schema and in entity bean represented as integers
- Application must use integer value to call a method findByPrimaryKey()
- More involved lookups use QueryBean class with JDBC
- Many aspects of relational data model permeate throughout EJB code
  - Does not look like a Java object model



# Establish datastore connection

- `PersistenceManagerFactory` serves as interface to datastore connection
- JDO implementation and configuration defined *via:*
  - Property file
  - XML service file
- Similar to DataSource, as used in JDBC/EJB



# JDO PMF Resource

```
<ejb-jar>
 <enterprise-beans>
  <session>
    <ejb-name>SB_ViewUserInfo</ejb-name>
    <home>edu.rice.rubis.beans.SB_ViewUserInfoHome</home>
    <remote>edu.rice.rubis.beans.SB_ViewUserInfo</remote>
    <ejb-class>edu.rice.rubis.beans.SB_ViewUserInfoBean
    </ejb-class>
    <session-type>Stateless</session-type>
    <transaction-type>Container</transaction-type>
    <resource-ref>
      <res-ref-name>jdo/RubisPMF</res-ref-name>
      <res-type>javax.jdo.PersistenceManagerFactory
      </res-type>
      <res-auth>Container</res-auth>
    </resource-ref>
  </session>
```

# Vendor PMF Service file

```
<!- Placed in Jboss  server/default/deploy directory -->

<?xml version="1.0" encoding="UTF-8"?>
<service>
  <mbean code="vendor.jdo.mbean.PMFService"
         name="jboss:service=vendor-rubis">
    <classpath codebase="lib" archives="vendorjdo.jar"/>
    <depends>jboss:service=TransactionManager</depends>
    <attribute name="JNDIName">java:/RubisPMF</attribute>
    <attribute name="ConfigFileName">jdo.properties
    </attribute>
  </mbean>
</service>
```

# Set Session Bean Context

```
protected PersistenceManagerFactory pmf = null;

public void setSessionContext(SessionContext sessionContext)
            throws RemoteException {
  if( pmf == null ){
    try {
      Context initialContext = new InitialContext();
      pmf = (PersistenceManagerFactory)
         initialContext.lookup("java:comp/env/jdo/RubisPMF");
    } catch(Exception ex){
      throw new RemoteException("Cannot get PMF: " +
                                ex.getMessage());
    }
  }
}
```

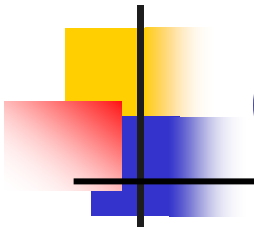
# Equivalent Code in EJB/JDBC

```
protected DataSource dataSource = null;

public void setSessionContext(SessionContext sessionContext)
   throws RemoteException {
  if (dataSource == null) {
    try {
        initialContext = new InitialContext();
        dataSource = (DataSource)
           initialContext.lookup("java:comp/env/jdbc/rubis");
    } catch (Exception e) {
      throw new RemoteException(
                    "Cannot get JNDI InitialContext");
    }
  }
}
```

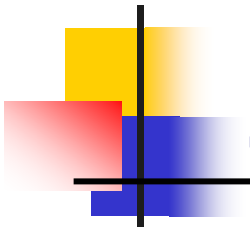
# Portability of Configuration Info

- Approach and naming scheme used varies among JDO vendors
- Establishing proper configuration required some iteration with vendor of file contents
- Alternative method is initialization of PersistenceManagerFactory *via* property file



# Queries to Access instances

- To access instances of a specific class
  - Defined static methods in class to execute a query
  - Created queries once and compiled them
- This localized commonly used queries
- More complex queries were placed in Session Bean classes



# JDO: User.initializeQueries()

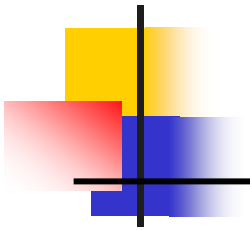
```
public static synchronized void
                  initializeQueries(PersistenceManager pm) {
  if(queryGetUserByNickname != null) return;
  Extent userExtent = pm.getExtent(User.class, true);

  query = pm.newQuery(userExtent, "id == uid");
  query.declareParameters("int uid");
  query.compile();
  queryGetUserByUid = query;

  query = pm.newQuery(userExtent, "nickName == nickname");
  query.declareParameters("String nickname");
  query.compile();
  queryGetUserByNickname = query;
}
```

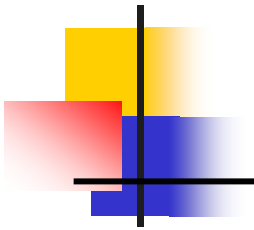
# JDO: User.getUser()

```
public static User getUser(PersistenceManager pm, int uid) {
  User user = null;
  if( JDORubisFlags.APPLICATION_IDENTITY ){
    try {
      user = (User) pm.getObjectById(new User.Id(uid), true);
    } catch(JDOException e) { }
  } else {
    if( queryGetUserByUid == null ) initializeQueries(pm);
    Query query = pm.newQuery(queryGetUserByUid);
    query.setCandidates(pm.getExtent(User.class, true));
    Collection result = (Collection) query.execute(
                                          new Integer(uid));
    Iterator iter = result.iterator();
    user = iter.hasNext() ? (User) iter.next() : null;
    query.close(result);
  }
  return user;
}
```



# Examination of Session Bean

- ViewUserInfoBean is a Session Bean that views information about a user
- Uses Container Managed Transactions
- Other beans use Bean Managed Transactions
  - To allow multiple transactions per bean method
- Will examine this bean in JDO, JDBC, EJB
- Two main methods in bean
  - getUserInfo()
  - setSessionContext()

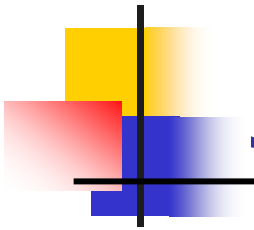


# JDO: getUserInfo()

```
public String getUserInfo(Integer userId)
                          throws RemoteException {
  try {
    pm = pmf.getPersistenceManager();
    User user = User.getUser(pm, userId.intValue());
    if( user == null ) return "User does not exist!<br>";
    StringBuffer html = new StringBuffer();
    html.append(user.getHTMLGeneralUserInformation());
    Set comments = user.getCommentsToUser();
    Iterator iter = comments.iterator();
    if( !iter.hasNext() ){
      html.append(
      "<h3>There is no comment yet for this user.</h3><br>");
      return html.toString();
    }
```

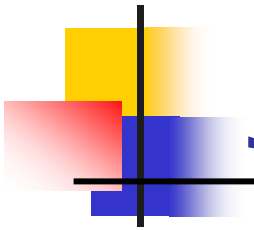
# JDO: getUserInfo() *(Continued)*

```
html.append(
    "<br><hr><br><h3>Comments for this user</h3><br>");
html.append(printCommentHeader());
 while( iter.hasNext() ){
  Comment comment = (Comment) iter.next();
  String commentString = comment.getComment();
  dateStr =
        TimeManagement.dateToString(comment.getDate());
  User from = comment.getFromUser();
  String authorName = from.getNickName();
  html.append(printComment(authorName, dateStr,
                  commentString, from.getId()));
}
html.append(printCommentFooter());
return html.toString();
```



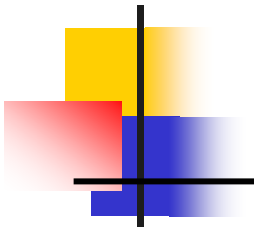
# JDO: getUserInfo() *(Continued)*

```
  } catch(JDOException e){
    throw new RemoteException(
           "JDO exception thrown getting user information: "
            + e.getMessage());
  } finally {  pm.close();  }
}
```



# JDBC: getUserInfo()

```
public String getUserInfo(Integer userId)
                    throws RemoteException {
  StringBuffer html = new StringBuffer();
  Connection conn = null;
  PreparedStatement stmt = null;
  ResultSet rs = null;
  try {
    conn = dataSource.getConnection();
    stmt = conn.prepareStatement(
                    "SELECT * FROM users WHERE id=?");
    stmt.setInt(1, userId.intValue());
    rs = stmt.executeQuery();
    stmt.close();
```

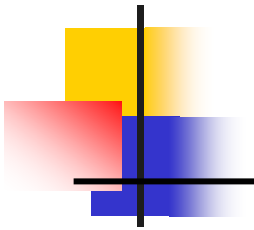


# JDBC: getUserInfo() *(Continued)*

```
} catch (SQLException e) {
    try {
      if (stmt != null) stmt.close();
      if (conn != null) conn.close();
    } catch (Exception ignore) { }
    throw new RemoteException("Failed to get user info : "
                                + e );
}
```

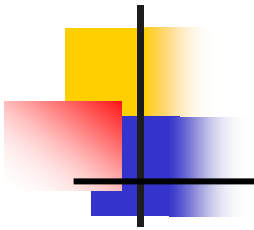
# JDBC: getUserInfo() *(Continued)*

```
try {
  if (rs.first()) {
    String firstname = rs.getString("firstname");
    String lastname = rs.getString("lastname");
    String nickname = rs.getString("nickname");
    String email = rs.getString("email");
    String date = rs.getString("creation_date");
    int rating = rs.getInt("rating");
    html.append(
      getHTMLGeneralUserInformation(firstname, lastname, )
    );
    html.append(getComments(userId, conn));
  } else
    html.append("This user does not exist!<br>");
  conn.close();
```



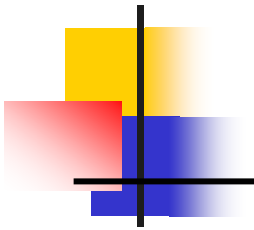
# JDBC: getUserInfo() *(Continued)*

```
  } catch (Exception e) {
    try {
      if (stmt != null) stmt.close();
      if (conn != null) conn.close();
    } catch (Exception ignore) { }
    throw new RemoteException(
            "Cannot get user information (got exception: "
             + e + ")<br>");
  }
  return html.toString();
}
```



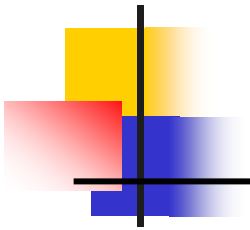
# JDBC: getUserInfo() *(Continued)*

```
public String getComments(Integer userId, Connection conn)
                    throws RemoteException {
  StringBuffer html;
  PreparedStatement stmt = null; ResultSet rs = null;
  String comment=null, date=null; int authorId;
  try {
    stmt = conn.prepareStatement(
                "SELECT * FROM comments WHERE to_user_id=?");
    stmt.setInt(1, userId.intValue());
    rs = stmt.executeQuery();
    stmt.close();
  }
```



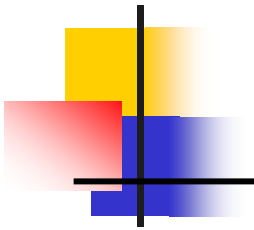
# JDBC: getUserInfo() *(Continued)*

```
catch (SQLException e) {
  try {
    if (stmt != null) stmt.close();
    if (conn != null) conn.close();
  } catch (Exception ignore) { }
  throw new RemoteException(
          "Failed to get categories list " + e);
}
```



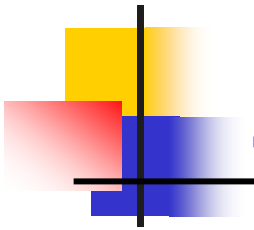
# JDBC: getUserInfo() *(Continued)*

```
try {
  if (!rs.first())
    html = new StringBuffer(
    "<h3>There is no comment yet for this user.</h3><br>");
  else {
    html = new StringBuffer(
        "<br><hr><br><h3>Comments for this user</h3><br>");
    html.append(printCommentHeader());
    try {
      stmt = conn.prepareStatement(
                "SELECT nickname FROM users WHERE id=?");
```



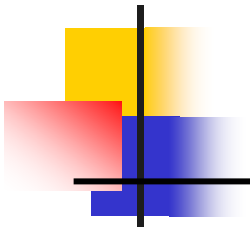
# JDBC: getUserInfo() *(Continued)*

```
do {
  comment = rs.getString("comment");
  date = rs.getString("date");
  authorId = rs.getInt("from_user_id");
  String authorName = "none";
  ResultSet authorRS = null;
  stmt.setInt(1, authorId);
  authorRS = stmt.executeQuery();
  if (authorRS.first())
    authorName = authorRS.getString("nickname");
  html.append(printComment(authorName, date,
                           comment, authorId));
} while (rs.next());
```



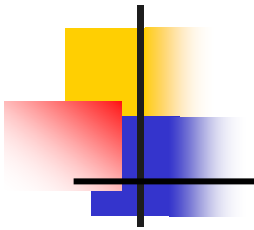
# JDBC: getUserInfo() *(Continued)*

```
    } catch (Exception e) {
      try {
        if (stmt != null) stmt.close();
        if (conn != null) conn.close();
      } catch (Exception ignore) { }
      throw new RemoteException(
        "This author does not exist(got exception: "
         + e +")<br>");
    }
    html.append(printCommentFooter());
  }
  if (stmt != null) stmt.close();
```



# JDBC: getUserInfo() *(Continued)*

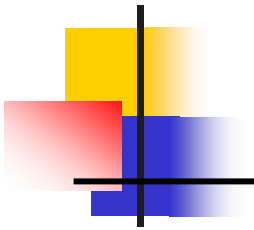
```
  } catch (Exception e) {
    try {
      if (stmt != null) stmt.close();
      if (conn != null) conn.close();
    } catch (Exception ignore) { }
    throw new RemoteException(
            "Exception getting comment list: " + e +"<br>");
  }
  return html.toString();
}
```



# EJB: getUserInfo()

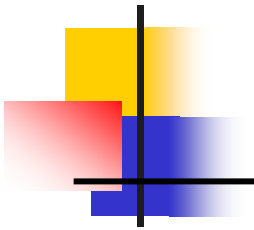
```
public String getUserInfo(Integer userId)
                    throws RemoteException {
  StringBuffer html = new StringBuffer();
  UserHome     uHome = null; User user = null;

// Try to find the user corresponding to the userId
  try {
    uHome = (UserHome)PortableRemoteObject.narrow(
            initialContext.lookup("java:comp/env/ejb/User"),
                                  UserHome.class);
  } catch (Exception e) {
    throw new RemoteException("Cannot lookup User: "
                              + e + "<br>");
  }
```



# EJB: getUserInfo() *(Continued)*

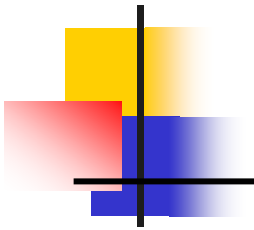
```
  try {
    user = uHome.findByPrimaryKey(new UserPK(userId));
    html.append(user.getHTMLGeneralUserInformation());
    html.append(getComments(uHome, userId));
  } catch (Exception e) {
    throw new RemoteException(
              "Cannot get user information (got exception: "
              + e +")<br>");
  }
  return html.toString();
}
```



# EJB: getUserInfo() *(Continued)*

```
public String getComments(UserHome userHome,
                          Integer userId) throws RemoteException {
  Collection   list;  StringBuffer html;
  CommentHome  cHome = null;   Comment comment = null;
  User         user = null;

  // Try to find the comments corresponding for this user
  try {
    cHome = (CommentHome)PortableRemoteObject.narrow(
        initialContext.lookup("java:comp/env/ejb/Comment"),
        CommentHome.class);
  } catch (Exception e) {
    throw new RemoteException("Cannot lookup Comment: " +
                              e +"<br>");
  }
```



# EJB: getUserInfo() *(Continued)*

```
utx = sessionContext.getUserTransaction();
try {
  utx.begin();
  list = cHome.findByToUser(userId);
  if (list.isEmpty())
    html = new StringBuffer("<h3>There is no comment" +
                            " yet for this user.</h3><br>");
  else {
    html = new StringBuffer(
      "<br><hr><br><h3>Comments for this user</h3><br>");
    html.append(printCommentHeader());
    Iterator it = list.iterator();
```

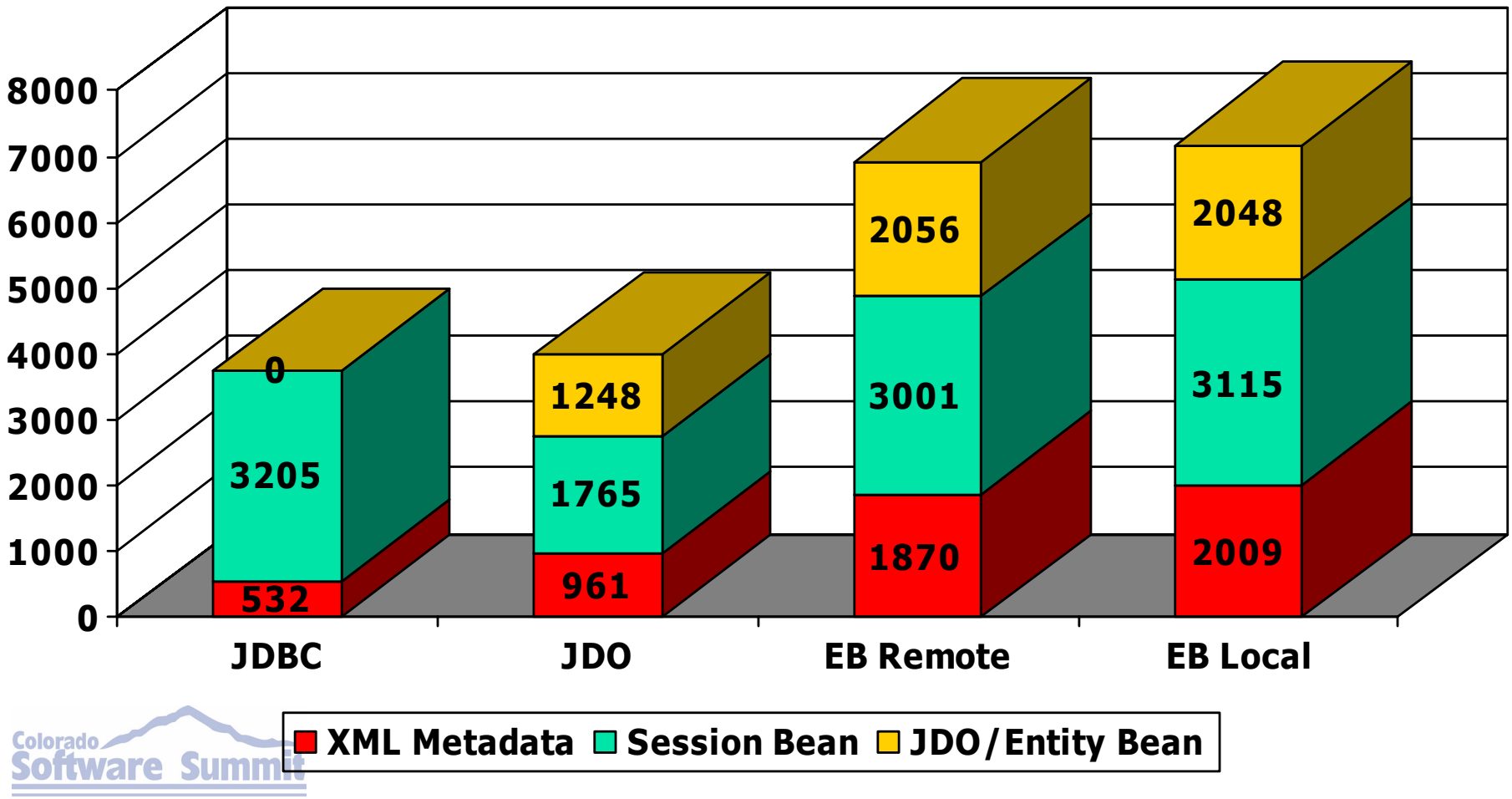
# EJB: getUserInfo() *(Continued)*

```
while (it.hasNext()) {
  comment = (Comment)it.next();
  String userName;
  try {
    user = userHome.findByPrimaryKey(
                new UserPK(comment.getFromUserId()));
    userName = user.getNickName();
  } catch (Exception e) {
    throw new RemoteException("This author does not"+
                  " exist, exception: " +e+")<br>");
  }
  html.append(printComment(userName, comment));
}
html.append(printCommentFooter());
}
utx.commit();
```

# EJB: getUserInfo() *(Continued)*

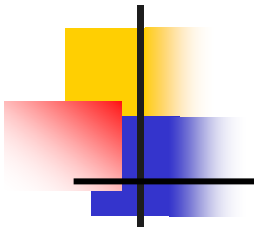
```
    } catch (Exception e) {
      try {
        utx.rollback();
        throw new RemoteException(
                  "Exception getting comment list: " +
                   e + "<br>");
      } catch (Exception se) {
        throw new RemoteException(
                        "Transaction rollback failed: " +
                         e +"<br>");
      }
    }
    return html.toString();
  }
```

# Non-Comment Source Lines



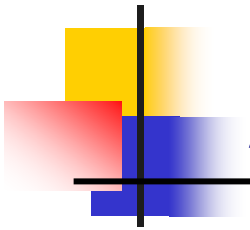
# RUBiS/JDO Appropriateness

- Data model was very simple
  - Only a few cases where collections made sense for navigating 1-many relationships
  - No inheritance
- Transaction characteristics
  - Did not have repeated access to same object
    - JDO's object caching provided no benefits, just upfront overhead
  - Data just passed to servlets
  - No object manipulation



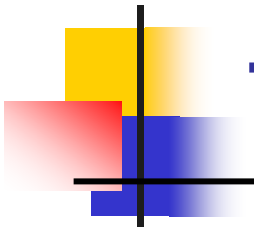
# RUBiS Reqmt Not in JDO

- GUI provided next, previous buttons to page through multiple pages of items
- RUBiS benchmark used SQL LIMIT operator to access page of data for the GUI
- JDO lacks a corresponding operator
- Required changes in Session Beans and servlets to save first and last ids displayed
- These ids were used in JDO as query constraints for processing subsequent page



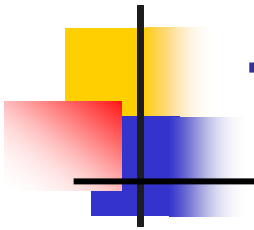
# Aspects of Other Architectures

- EJB implementation used JDBC/SQL for requests other than simple bean lookup
  - Did not use EJB model to navigate relationships
- Very little SQL join processing in benchmark
- Most queries were index lookup in one table
  - Relationships were traversed by application using foreign key and calling findByPrimaryKey()



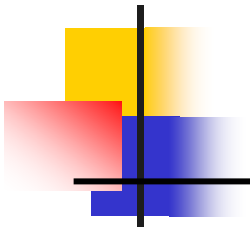
# Termination of JDO RUBiS

- Benchmarking performed at Inria's lab
- 1Ghz, 1Gb memory Linux database server
- EJB, JDBC, JDO results did not show conclusive performance differences
  - Dramatic difference between InnoDB/MyISAM
  - Database server was severely CPU bound
- Inria decided a larger, multiprocessor machine was necessary to get viable performance comparisons



# Termination of JDO RUBiS

- Current hardware needed for other uses
- Neither Inria nor JDO vendors had financial resources or interests to acquire Inria a sufficiently capable database server
- JDO RUBiS effort dropped, lack of resources
- RUBiS benchmark main focus was measuring application server throughput, it was not designed to deal with database bottlenecks, concurrency conflicts, *etc.*



# In Summary

- We examined differences between JDO, JDBC, and Entity Beans with respect to source code for RUBiS benchmark
- Unfortunately, a lack of funds and resources prevented completion of the benchmark
- Characteristics of original RUBiS application would not have taken advantage of many of JDO's capabilities