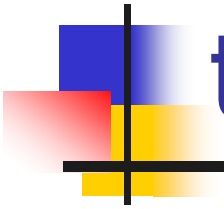
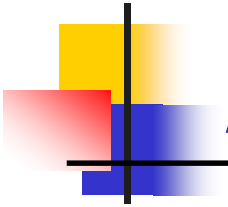


Standards-based Persistence through Web Services

Michael Keith

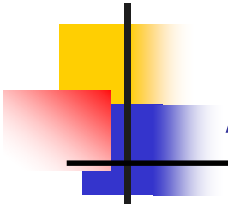
Oracle Corp.

mkeith@oracle.com



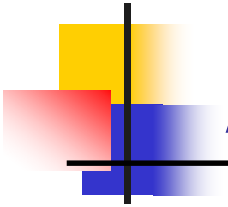
About Me

- 14+ years experience in OO persistence
- Worked on many projects, including one of the earliest EJB (1.0) CMP implementations
- Technical Architect for OracleAS TopLink and OracleAS EJB Container in OC4J
- Was on JSR 243 (JDO 2.0) expert group
- Currently on JSR 220 (EJB 3.0) expert group



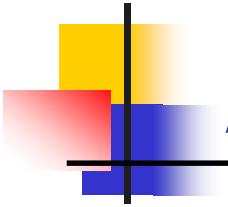
Audience Poll

1. How many have used pure JDBC?
2. How many people have used EJB entities?
3. How many have used JDO?
4. How many have used all three?



Agenda

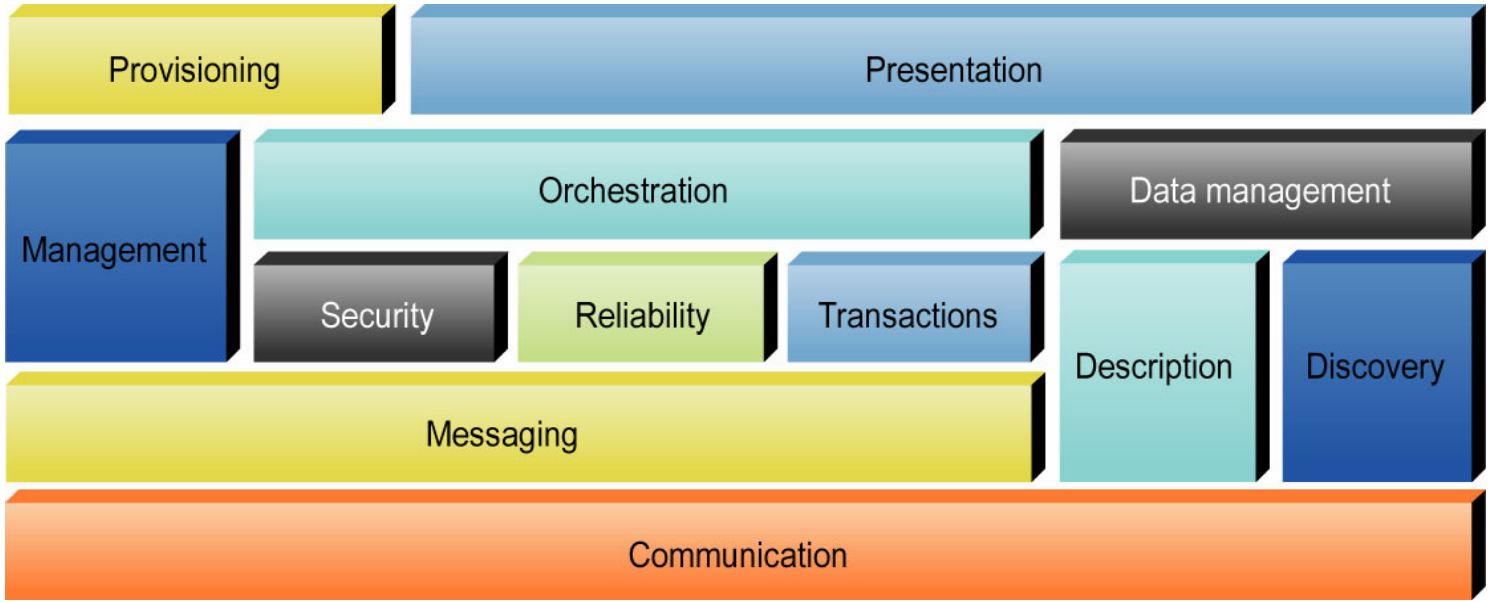
- The Chaos
- Focus on Persistence
- JDBC
- JDO
- EJB
- Summary



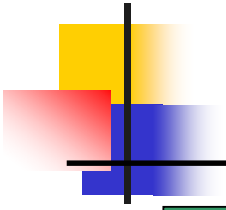
Agenda

- The Chaos
- Focus on Persistence
- JDBC
- JDO
- EJB
- Summary

The Chaos

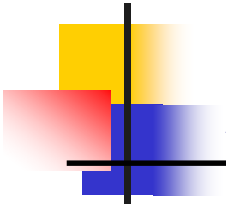


Web Services Framework functional layers



The Chaotic Details

Functional Layer	Specification
Data Management	XPath, XSLT
Description	XMLSchema, WSDL
Discovery	UDDI
Messaging	SOAP
Orchestration	BPEL4WS, WS-Choreography
Presentation	WSRP
Provisioning	SPML
Reliability	WS-Reliability ?, WS-RM ?
Security	WS-Security
Transactions	WS-CAF?, WS-Coordination?



Absent Chaos

Where is the persistence layer?

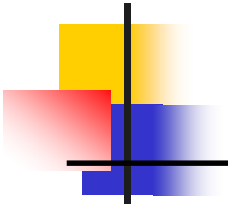
- The theory is that you can just plug one in

What should we use?

- Depends on our requirements

Who can we trust?

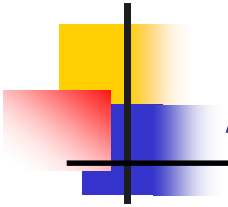
- Ourselves. We need to inform ourselves about the solutions then we can make our own decisions



Persistence Problem

Problem Statement:

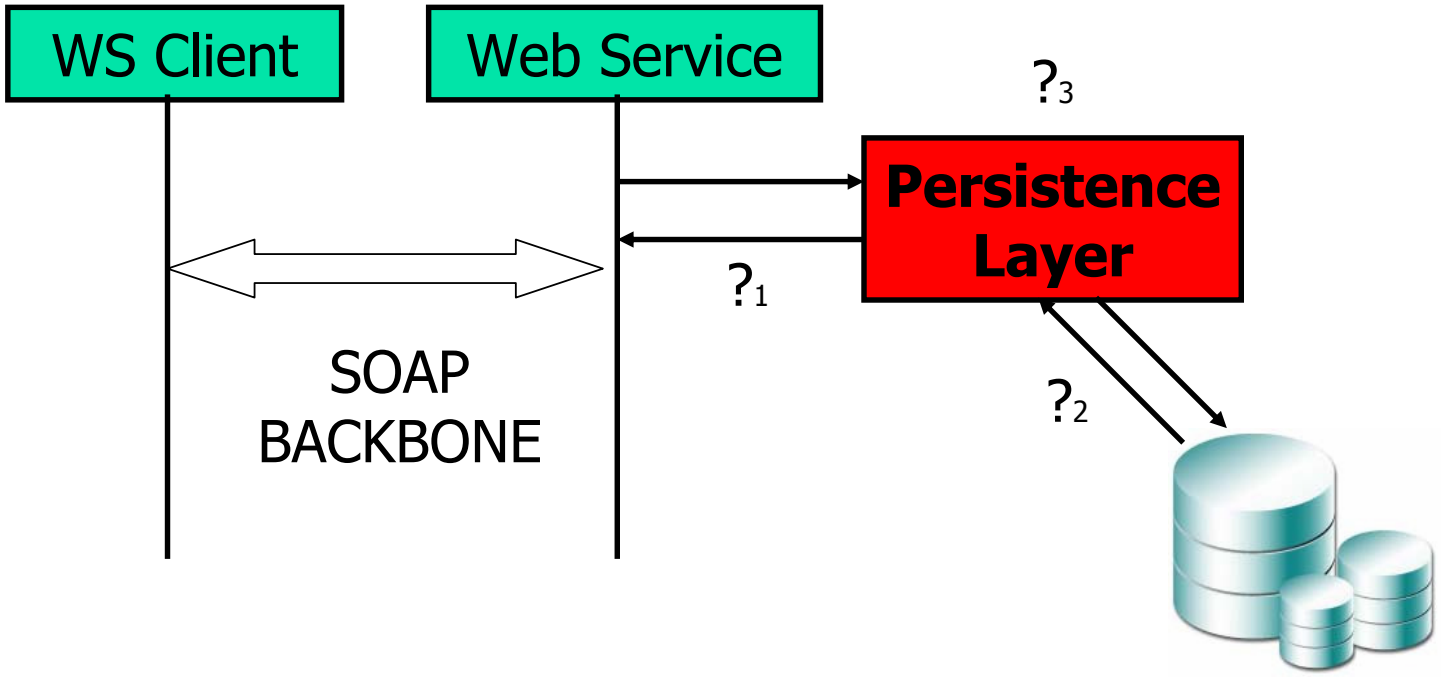
On server-side receipt of a JAX-RPC message, how do we persist and manage the relevant user state and service metadata in a standards-based and portable way?



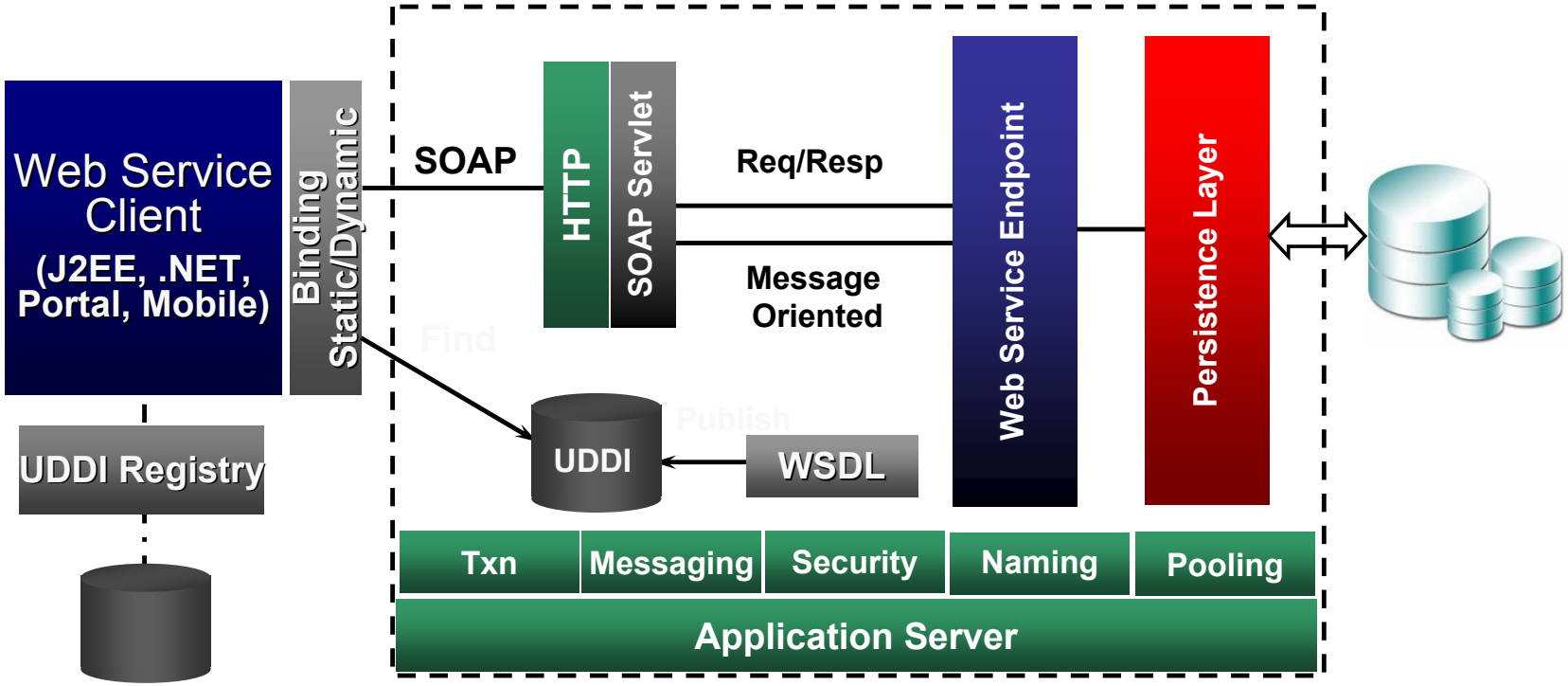
Agenda

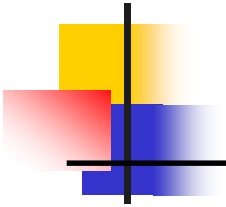
- The Chaos
- Focus on Persistence
- JDBC
- JDO
- EJB
- Summary

High Level View



Detailed View

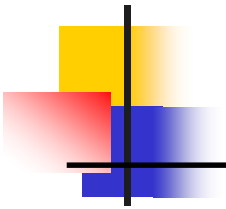




Persistence Requirements

1. Standards

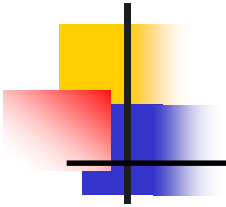
- Well-defined technology
- Potential for vendor migration
- Controlled evolution and direction
- Big vendor support
- Test suites (compatibility/quality standards)
- Integration with other standards



Persistence Requirements

2. Features

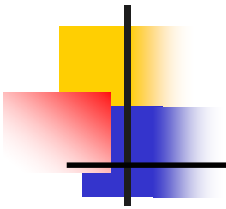
- Need all of the basics (*e.g.* queries, locking, caching, txn support (2PC?), *etc.*)
- Feature parameterization/customization
- Ability to map to different data store types
- Supports data-driven model for legacy data
- Tools for operating on development artifacts



Persistence Requirements

3. Performance

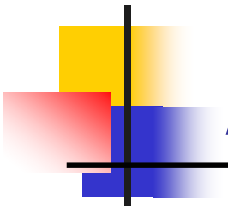
- Support large or small backend data stores equally well
- Optimizations/caching for data store type
- Scales well in the face of high request concurrency
- Optimizes for clustered/multi-tier architectures



Persistence Requirements

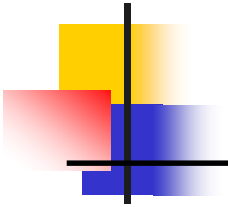
4. Pluggability

- Seamless integration into Web Service
- Integrates on back end into other types of data (EIS, ERP, *etc.*)
- Provides plugs for messaging to other layers
- Integrates into tools, IDEs, and other utilities
- Integration is not domination!



Anti-Requirements

- Custom code layer – maintenance hell
- Code Intrusion – don't want persistence code scattered throughout the domain layer
- Development Intrusion – don't want the technology to dictate the development process
- Architectural Intrusion – don't want the persistence layer to affect how other layers are designed



Persistence Choices

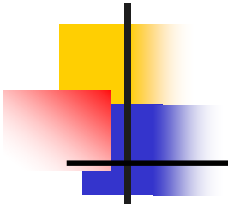
Standards:

- Java Database Connectivity (JDBC)
- Java Data Objects (JDO)
- Enterprise JavaBeans (EJB)

Others:

1. Open source – Hibernate, Apache OJB, Cayenne
2. Commercial – TopLink

Interestingly these are the most popular!

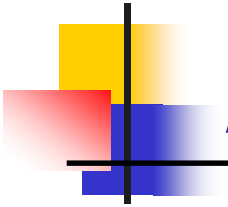


Persistence Choices

Not Covered here:

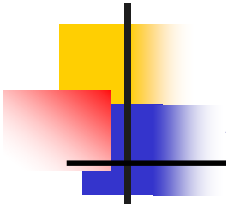
- Staying in XML (not converting to objects)
- Mapping objects to XML and persisting the XML

  (See *Donald Smith – "Mapping Java Objects to XML and Relational Databases"*
- Other custom transforms to persistent stores (*e.g.* directly to J2C)
- Hosts of custom approaches to persistence



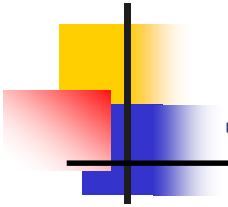
Aspects of Persistence

- Persistence model, distinguishing features
- Querying, query language
- Updating, writing
- Transactions
- Caching
- Locking
- Configuration



Agenda

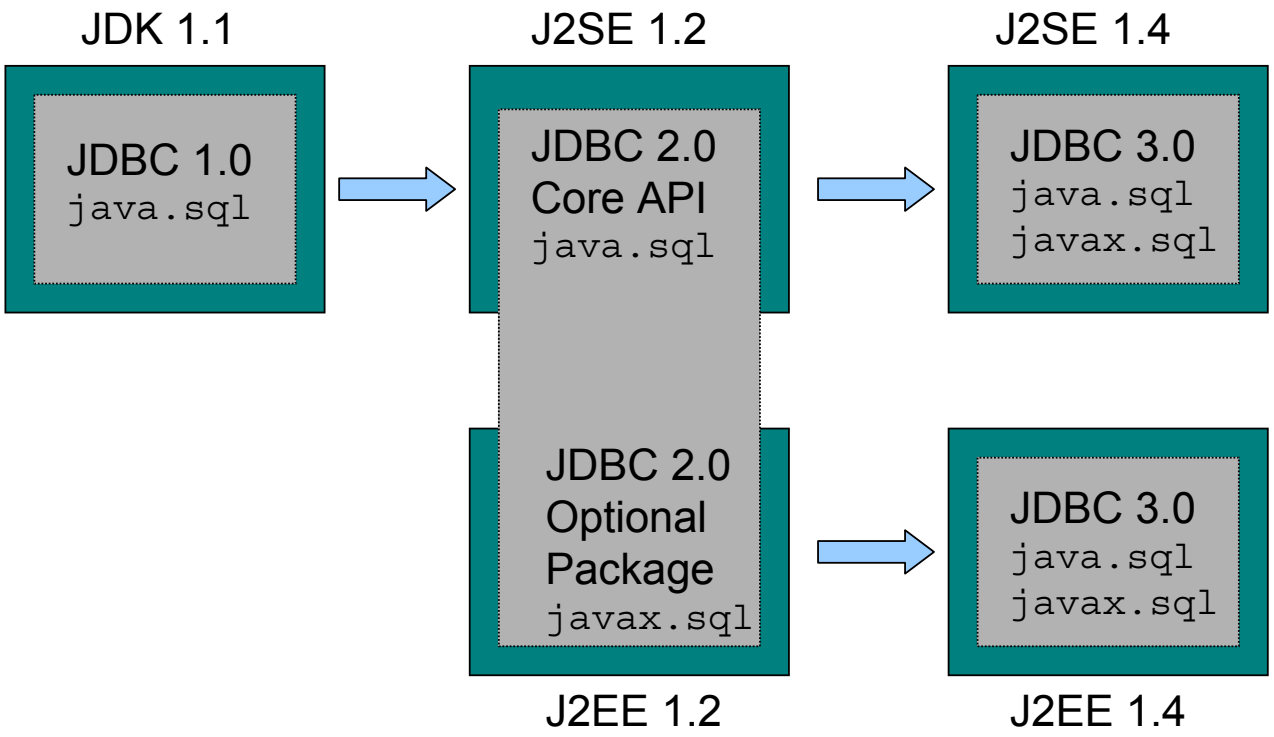
- The Chaos
- Focus on Persistence
- JDBC
- JDO
- EJB
- Summary

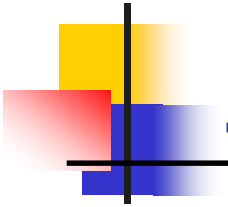


JDBC – What Is It?

- Java DataBase Connectivity
- Current version is JDBC 3.0
- Java standard for accessing databases
- CLI like ODBC, but for Java?
- Part of the Java 2 SDK (J2SE) + J2EE
- Requires a JDBC driver implementation for a given database
- Not used by James Gosling!

Some JDBC History

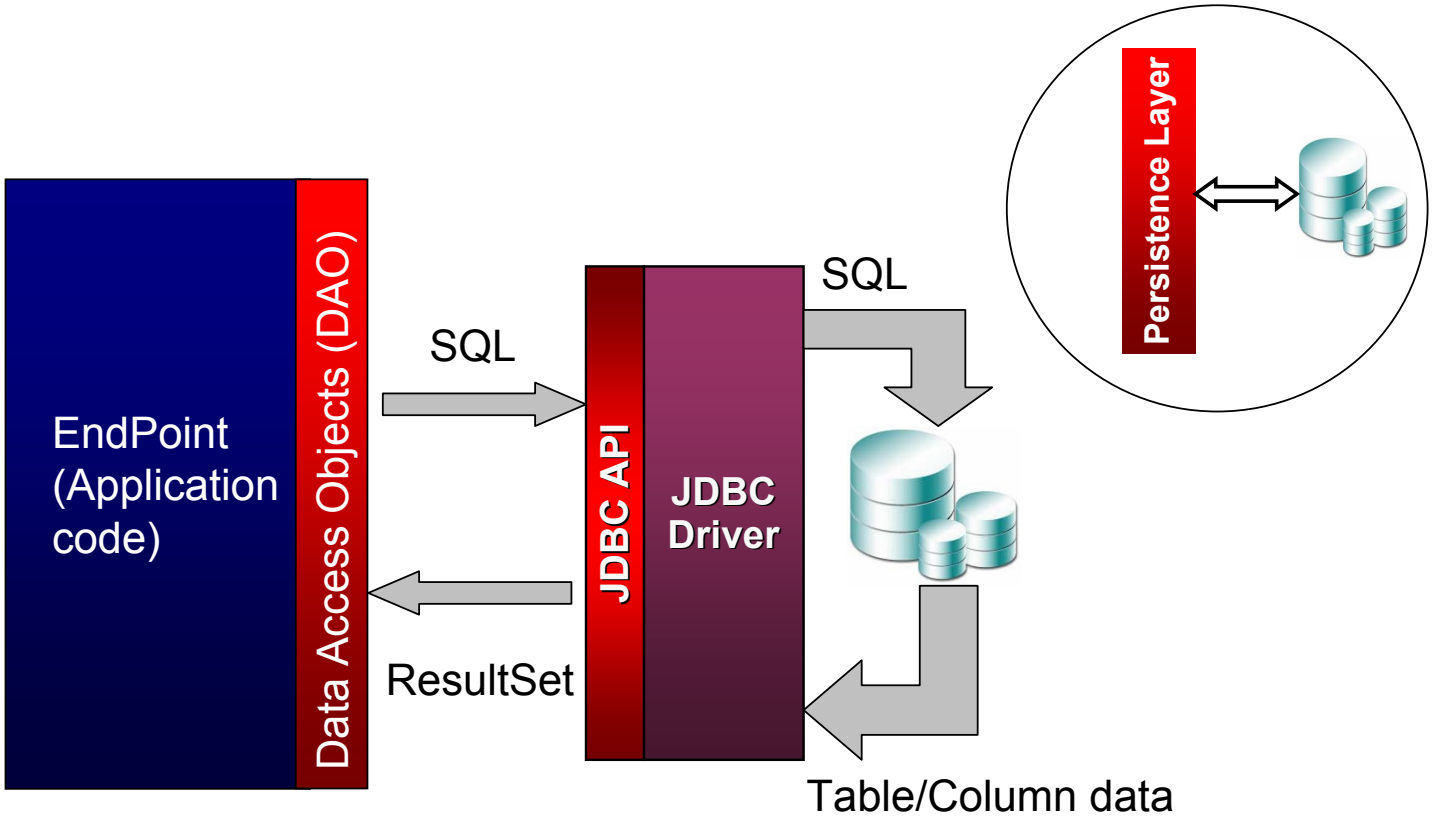




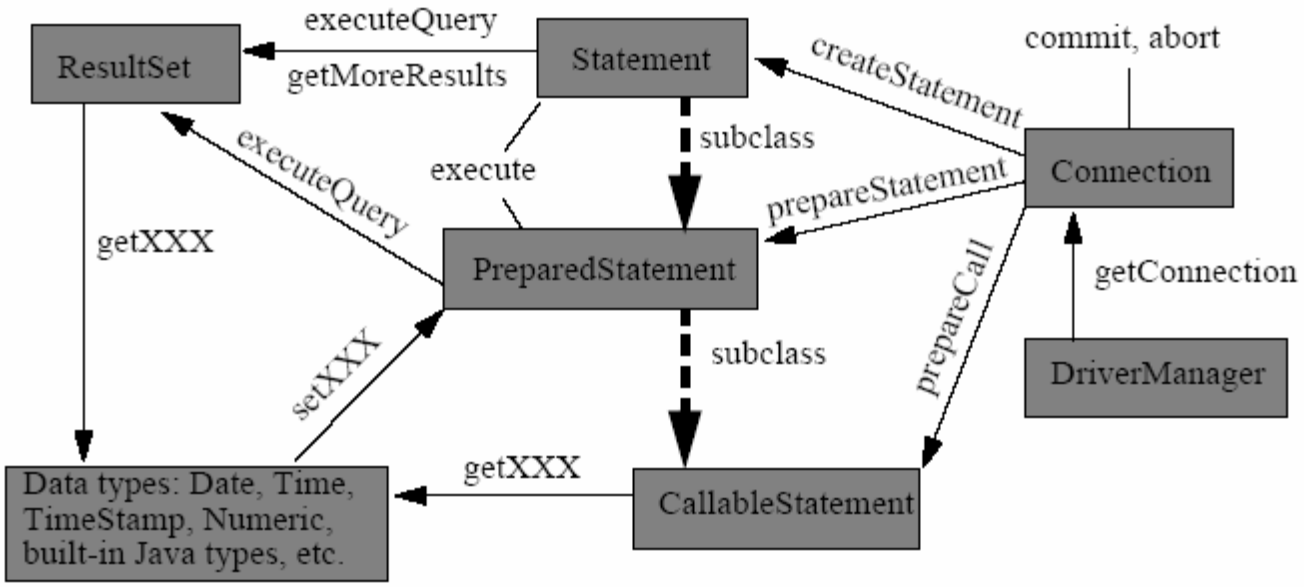
JDBC

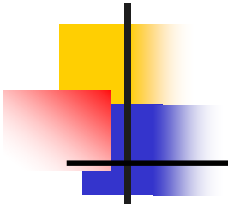
- Use Java API to operate directly on the database
- No object-table mapping component
- No object conversion layers
- Outgoing data in SQL statement form
- Incoming data as tabular result sets
- Not object-oriented

JDBC Architecture View



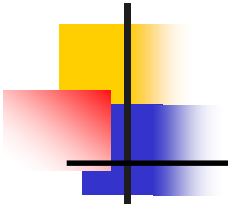
Basic JDBC Model





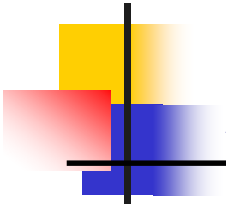
Reading/Query Language

- Set query criteria on statement
- Native SQL for database
- Execution method aligns with query type
- Scroll through ResultSet data
- Must convert the raw data into object form
- Can set result set size limit



Reading/Query Language

```
public Vector<Employee> getEmployeesNamed(String empName)
                                        throws SQLException {
     PreparedStatement stmt = conn.prepareStatement(
        "SELECT emp_no, emp_name, salary " +
        "FROM employees WHERE emp_name LIKE ?");
     stmt.setString(1, empName);
     ResultSet rs = stmt.executeQuery();

     Vector<Employee> emps = new Vector<Employee>();
     while (rs.next()) {
        Employee emp = new Employee();
        emp.setId(rs.getLong(1));
        emp.setName(rs.getString(2));
        emp.setSalary(rs.getInt(3));
        emps.add(emp);
     }
     stmt.close();
     return emps;
  }
```



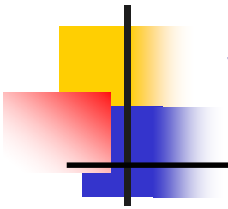
Anti-Practice

- When connections are closed the associated statements and result sets get automatically closed
- When finalization runs on statements they are closed and reclaimed (along with their result sets)

Some people rely upon this finalization or reclamation by the parent resource. Don't do it!

Don't rely on the system to clean up

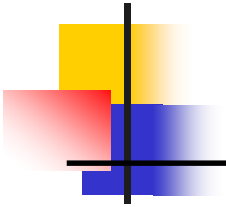
The current resource may run out before the reclaiming part actually ends up happening.



Writing

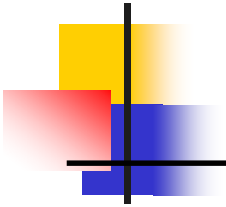
Same as reading, except:

- Set update criteria on statement
- Must strip the data out of the object
- Use executeUpdate method
- Result is integer update count



Writing

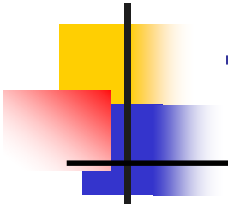
```
public void updateEmployee(Employee emp))
                                    throws SQLException {
    PreparedStatement stmt = conn.prepareStatement(
      "UPDATE emp SET emp_name = ?, salary = ? " +
      "WHERE emp.emp_no = ?");
    stmt.setString(1, emp.getName());
    stmt.setInt(2, emp.getSalary());
    stmt.setLong(3, emp.getId());
    int count = stmt.executeUpdate();

    if (count == 0) {
      createEmployee(emp);
    }
}
```



Transactions

- Simple window on database transaction
- Single transaction per connection
- autocommit flag on connection
  - Defaults to starting a transaction for every statement and committing it afterwards

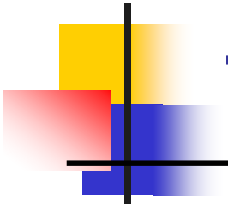
```
conn.setAutoCommit(false);
ResultSet rs = conn
   .createStatement("SELECT * FROM EMP")
   .executeQuery();
…
conn.commit();
```



Transactions

- Savepoints (in 3.0)

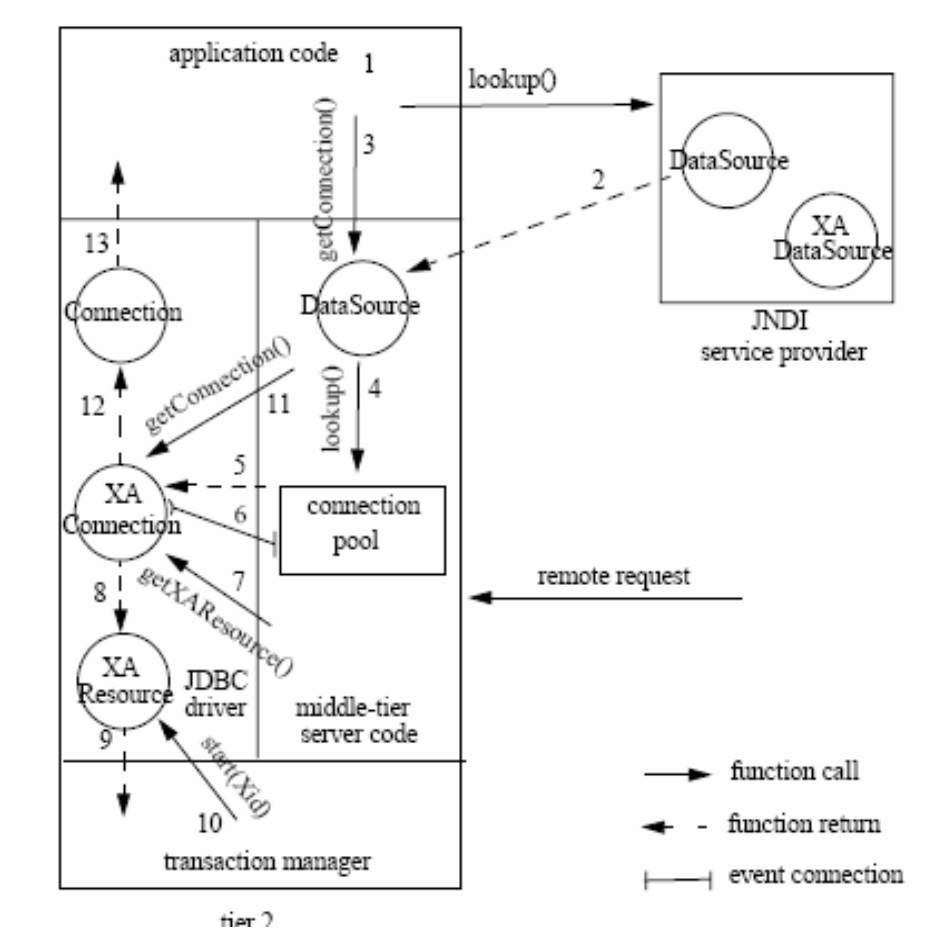
```
conn.setAutoCommit(false);
Statement stmt = conn.createStatement();
int rows = stmt.executeUpdate(
   "INSERT INTO EMP (EMP_NAME) VALUES " +
   "('JOHN SMITH')");
Savepoint sp1 = conn.setSavepoint("SP_1");
rows = stmt.executeUpdate(
   "INSERT INTO EMP (EMP_NAME) VALUES " +
   "('MARY SMITH')");
conn.rollback(sp1);
…
conn.commit();
```

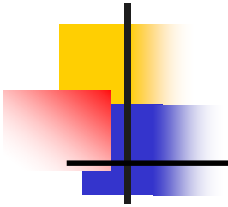


Transactions

- Support for XA distributed transactions
- XADataSource, XAConnection used by the server
- Transaction manager must do all of the XA protocol handling by obtaining an XAResource from the XAConnection
- Layer of XA artifacts is completely transparent to the user
- Not all drivers support it

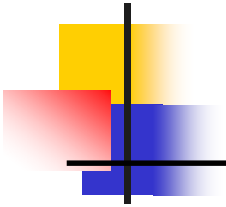
Transactions





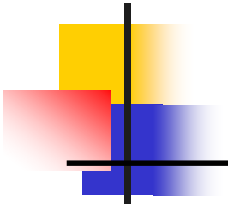
Caching

- Some connection caching (pooling)
- Some statement caching – PreparedStatements offer DB precompiled statement caching
- No caching of ResultSet data!



Locking

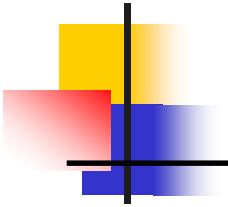
- Transaction Isolation
  - 5 levels:
    - NONE
    - READ_UNCOMMITTED
    - READ_COMMITTED
    - REPEATABLE_READ
    - SERIALIZABLE
  - Not all levels are necessarily supported by the DB



Locking

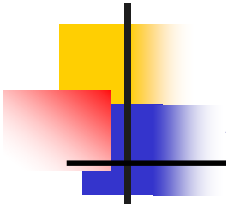
Implementation-defined as to when the isolation level may be set (usually not in the middle of a transaction)

```
conn.setAutoCommit(false);
conn.setTransactionIsolation(
        TRANSACTION_READ_COMMITTED);
```



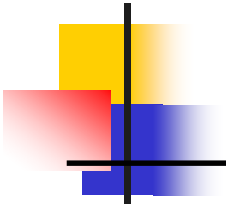
Configuration

- No special configuration required
- Put the driver on the classpath
- Lookup the DataSource in JNDI (or use DriverManager in two-tier archs)



Agenda

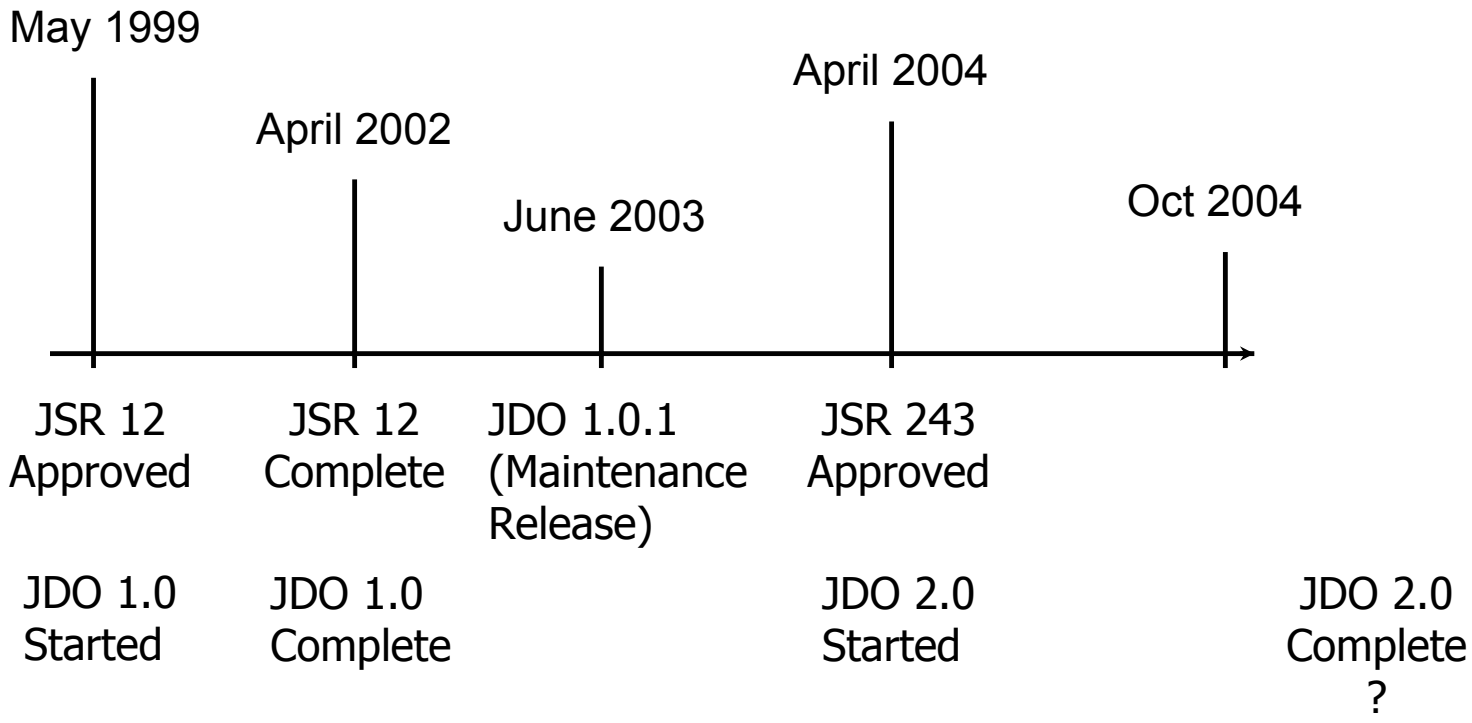
- The Chaos
- Focus on Persistence
- JDBC
- JDO
- EJB
- Summary



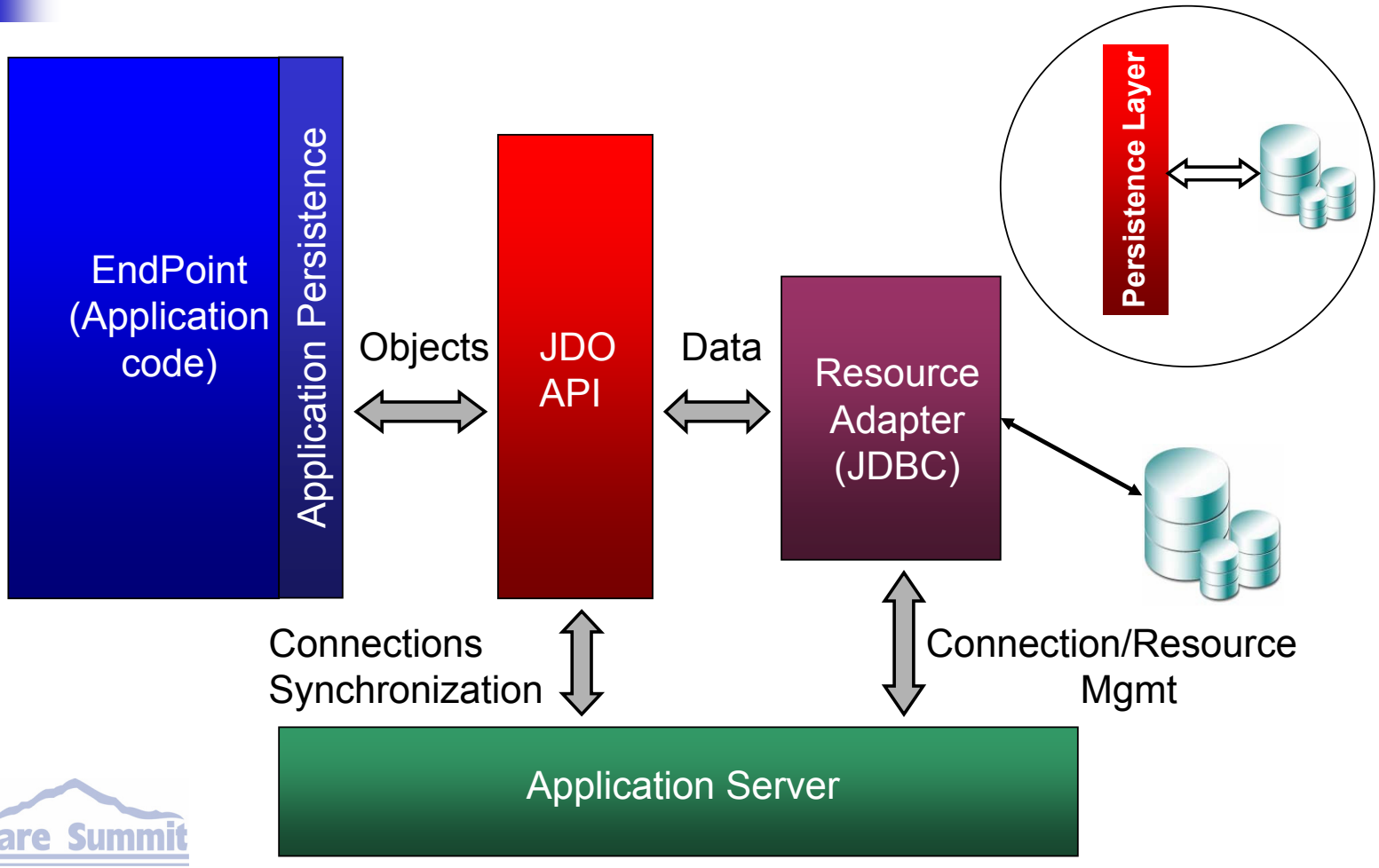
JDO – What Is It?

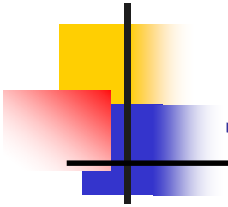
- Java Data Objects 1.1 (2.0 in progress)
- Object persistence layer originally over ODBMS, in process of expansion to RDBMS
- Optional package to J2SE
- Core set of features, plus some specified optional ones
- Persistence-by-reachability
- Byte-code enhancement

Some JDO History



JDO Architecture View



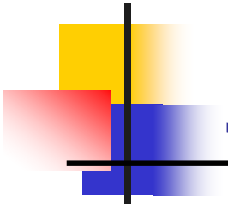


JDO API

Spec is a combination of application API and vendor SPI

Application API:

- **JDOHelper** – state querying, bootstrap methods
- **PersistenceManagerFactory** – factory used to configure PMs that get created
- **PersistenceManager** – main API class
- **Query** – composed of target class, extent, filter
- **Extent** – candidate collection of queryable objects
- **Transaction** – normally maps to data source txn
- **InstanceCallbacks** – can implement if interested

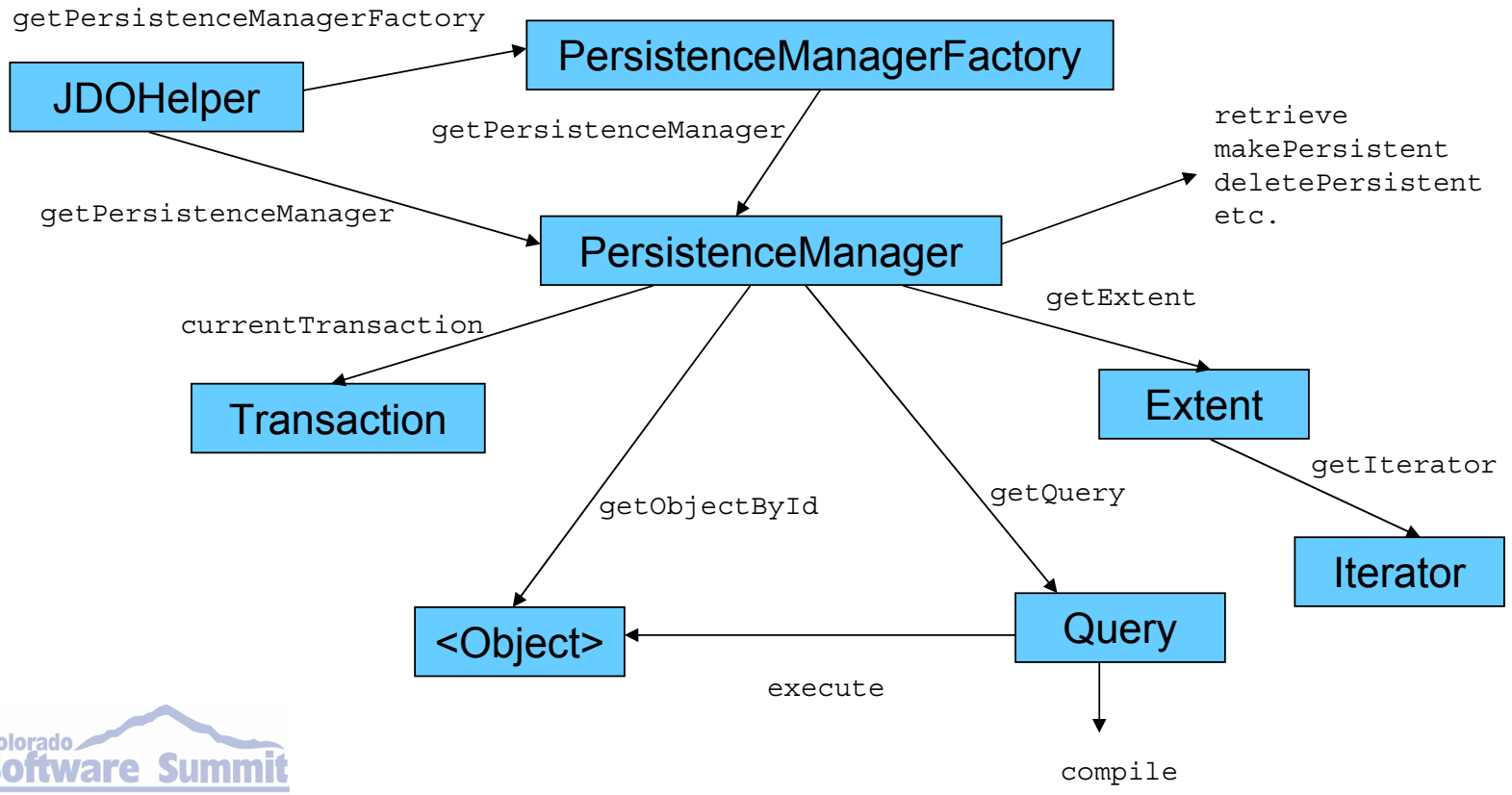


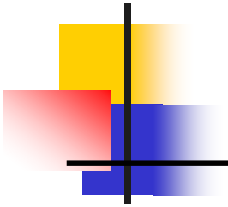
JDO SPI

Vendor SPI:

- **PersistenceCapable** – vendors enhance every persistent object to implement this
- **StateManager** – vendors must use this class to store the life cycle state of the object
- **JDOHelperImpl** – utility methods for vendors
- **RegisterClassListener** – notified when a new class gets registered with the helper
- **RegisterClassEvent** – event that gets passed to the listener and gives the metadata of the class being registered

Basic JDO Model





Enhancement

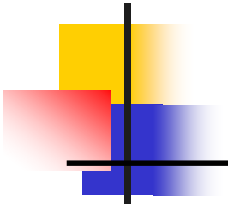
Byte code enhancement contract:

1. All persistent objects must implement the PersistenceCapable interface

```
public Employee { … }
```

In byte code becomes:

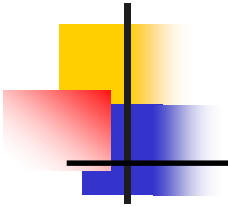
```
public Employee
      implements PersistenceCapable {
 …
}
```



Enhancement

2. Enhancer must modify byte codes and add PC methods prior to loading the class

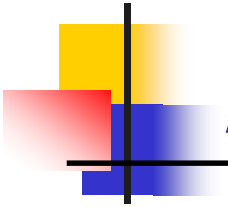
```
interface PersistenceCapable {

        PersistenceManager jdoGetPersistenceManager();
        void jdoMakeDirty(String fieldName);
        boolean jdoIsDirty();
        Object jdoNewInstance(Statemanager sm);
        boolean jdoReplaceField(int fieldNum);
              …
}
```



Enhancement

3. Enhanced persistent classes must be binary compatible between vendors

Enhanced by \ Runtime Env	**Ref Impl**	**Vendor A**	**Vendor B**
Ref Enhancer	Ref Contract	Ref Contract	Ref Contract
Vendor A	Ref Contract	Vendor A Contract	Ref Contract
Vendor B	Ref Contract	Ref Contract	Vendor B Contract



Anti-Practice

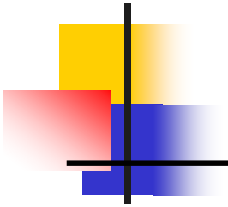
- PersistenceCapable interface is available from the object, so developer could do:

```
if (((PersistenceCapable)emp).jdoIsDeleted()) {
   throw new ObjectDeletedException();
}
```

- Likewise, users are allowed to implement the PC interface as part of the domain object definition. Don't do it!

Don't reference PersistenceCapable in code

PC is an impl interface that may disappear at any time.

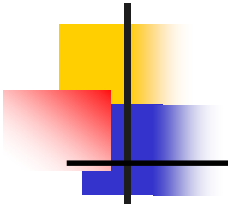


Identity

3 types of persistent identity:

- Application identity – primary key, defined in the application domain object
- Datastore identity – associated with the object in a way that is transparent to the application
- Nondurable identity – only associated with the object when in main memory (***a*** != ***a*** if ***a*** is evicted and reloaded later!)

Vendors must implement either one (or both) of Application or Datastore identity



Life Cycle State

Persistent object may be in any one of 10 states:

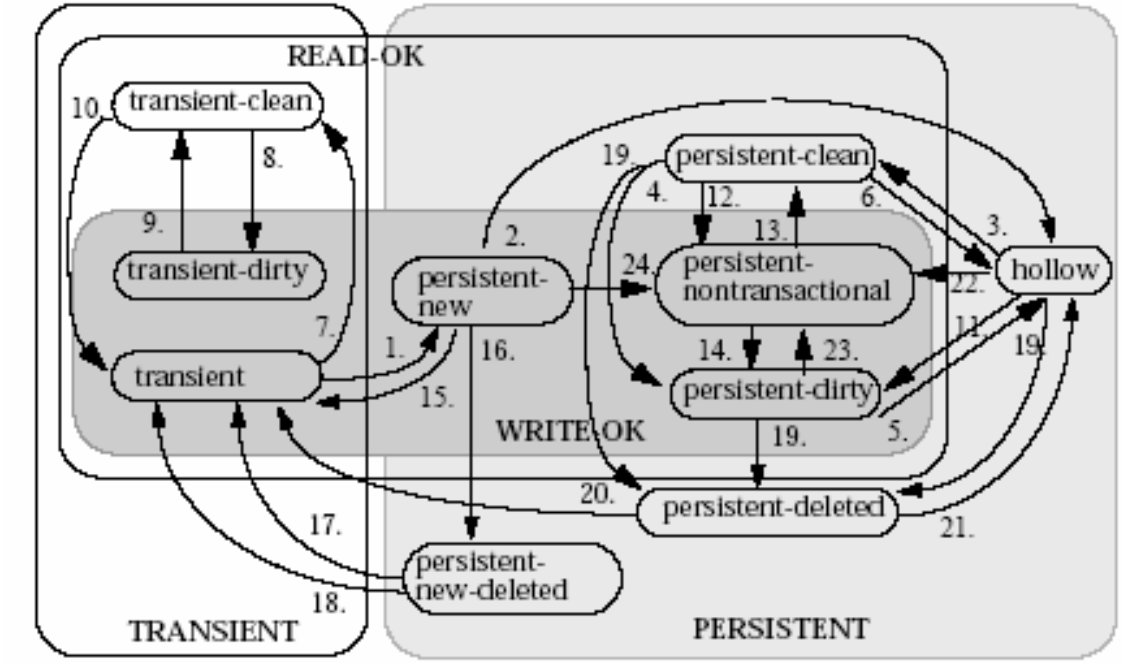
- Transient
- Persistent-New
- Persistent-Dirty
- Hollow
- Persistent-Clean
- Persistent-Deleted
- Persistent-New-Deleted

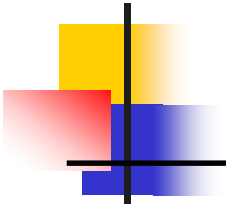
(Optional)

- NonTransactional
- Persistent-NonTransactional
- Transient-Transactional

State Transitions

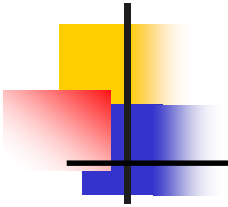
State transitions are not so simple:





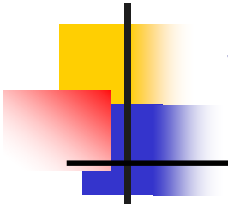
Reading/Query Language

- Create a Query object and execute it
- JDOQL query language is "Java-like"
- Specify in the query:
  - Target class – object type and identifier scope
  - Candidate object set – Collection or Extent
  - Filter – criteria as a JDOQL string
- Additionally define:
  - Import statements – to expand identifier scope
  - Parameters – to pass in argument data
  - Variables – can represent instance in a Collection



Reading/Query Language

```
Query q = pm.newQuery(Department.class);
String filter =
    "emps.contains(emp) & emp.salary > sal";
q.setFilter(filter);
q.declareImports("import acme.Employee");
q.declareVariables("Employee emp");
q.declareParameters("Integer sal");
Collection deps = (Collection)
    q.execute(new Integer(30000));
```



Writing

Creating:

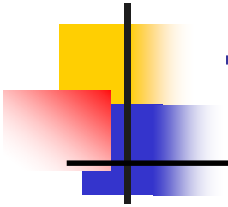
- Pass the object into pm.makePersistent()
- Relate the object to another already-managed persistent object

Updating:

- Modify the managed object in the context of a transaction and commit the transaction

Deleting:

- Pass the object into pm.deletePersistent()



Transactions

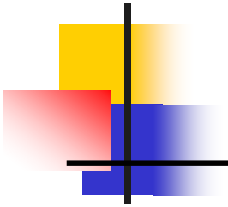
- Single transaction / PersistenceManager
- Accessed from PM

```
Transaction txn = pm.currentTransaction();
```

- Demarcate using calls on the transaction:

```
txn.begin();
…
txn.commit(); // or txn.rollback();
```

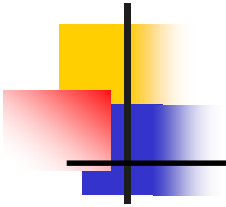
- Uses synchronization when in managed environment (no explicit begin/commit)



Caching

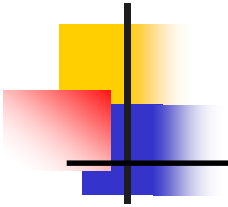
- Each PersistenceManager has its own cache
- Single unique object instance in each cache
- Cache management operations, such as evict, refresh, *etc*.
- Optional properties to control object field data across transactions (not required by implementation)

  example: RetainValues=true



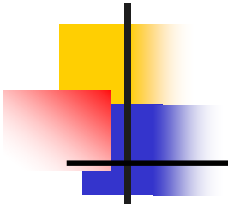
Locking

- Optional feature for optimistic locking
  - Set on Transaction.setOptimistic(true), or OptimisticTransaction
  - Objects are not transactional unless they are modified or marked as transactional
  - Verification done by vendor (in vendor-specific way) at commit-time
- No specific transaction isolation support is included in the specification



Configuration

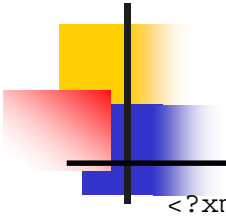
- Metadata for all persistent classes must be declared in XML file(s)
- Used at enhancement-time and normally at runtime
- Configuration .jdo files may be included at:
  - Class level – <package>/<class>.jdo
  - Package level – <package>/…/package.jdo
  - Application level – [META-INF/]package.jdo



Configuration

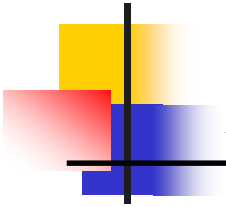
Search order used for org.acme.Employee:

1. META-INF/package.jdo
2. WEB-INF/package.jdo
3. package.jdo
4. org/package.jdo
5. org/acme/package.jdo
6. org/acme/Employee.jdo



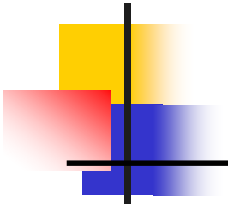
Configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jdo SYSTEM "jdo.dtd">
<jdo>
    <package name="org.acme">
        <class name="Employee" identity-type="application" objectidclass="EmployeePK">
            <field name="id" primary-key="true"/>
            <field name="name"/>
            <field name="salary"/>
            <field name="dept">
                <extension vendor-name="MyJDO" key="inverse" value="emps"/>
            </field>
        </class>
        <class name="Department" identity-type="application" objectidclass="DeptKey">
            <field name="id" primary-key="true"/>
            <field name="name"/>
            <field name="emps">
                <collection element-type="Employee">
                    <extension vendor-name="MyJDO" key="element-inverse" value="dept"/>
                </collection>
            </field>
        </class>
    </package>
</jdo>
```



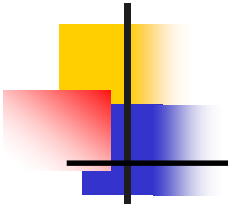
Agenda

- The Chaos
- Focus on Persistence
- JDBC
- JDO
- EJB
- Summary



EJB – What Is It?

- Enterprise JavaBeans 2.1 (3.0 in progress)
- Entities are core persistence component in J2EE layer
- Can be Bean-Managed (BMP) or Container-Managed (CMP) entity beans
- Clients access the object through either a local or remote interface (or both)
- Bean instances stay on the server (are shielded from clients)



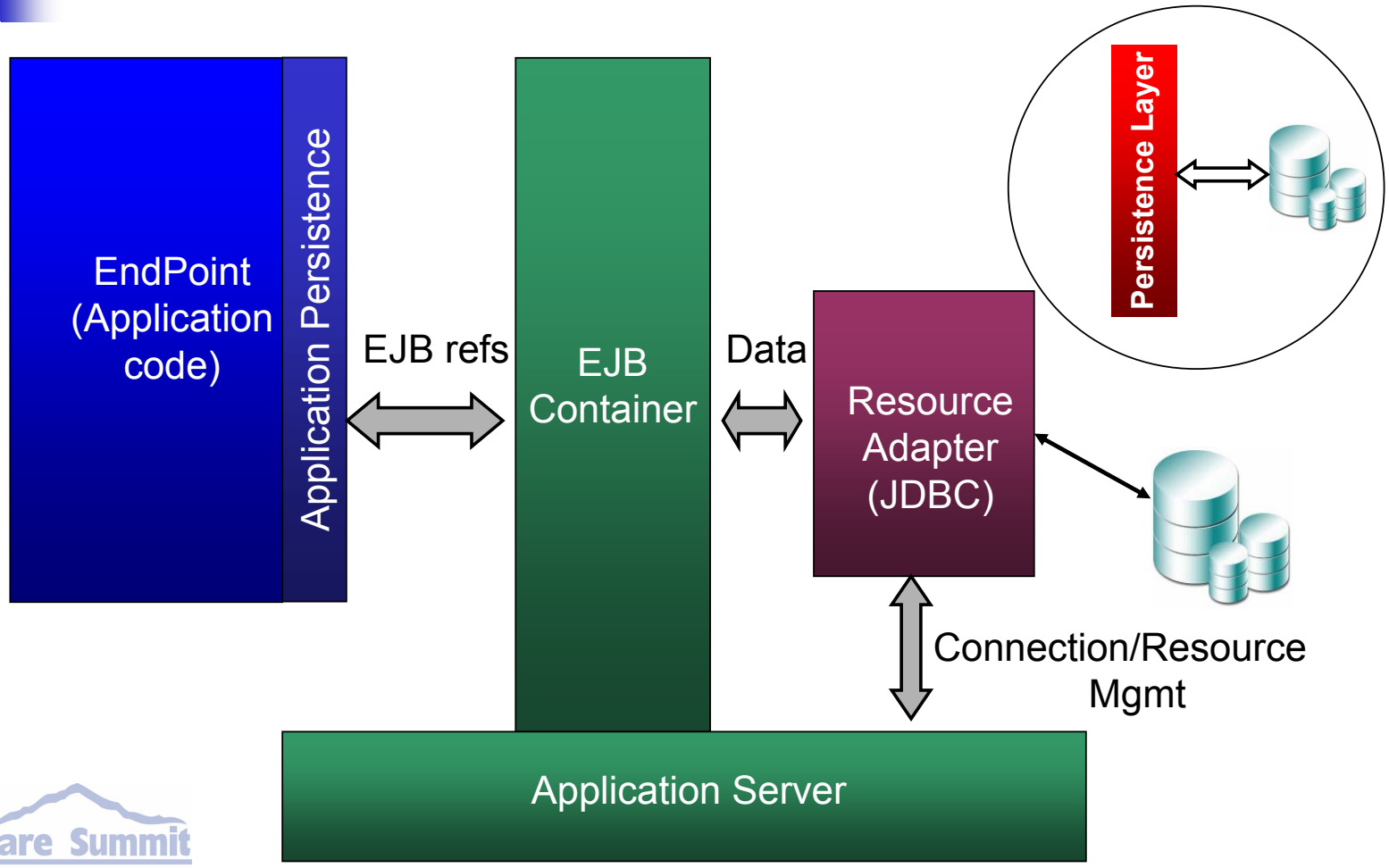
EJB History

For history of EJB see:

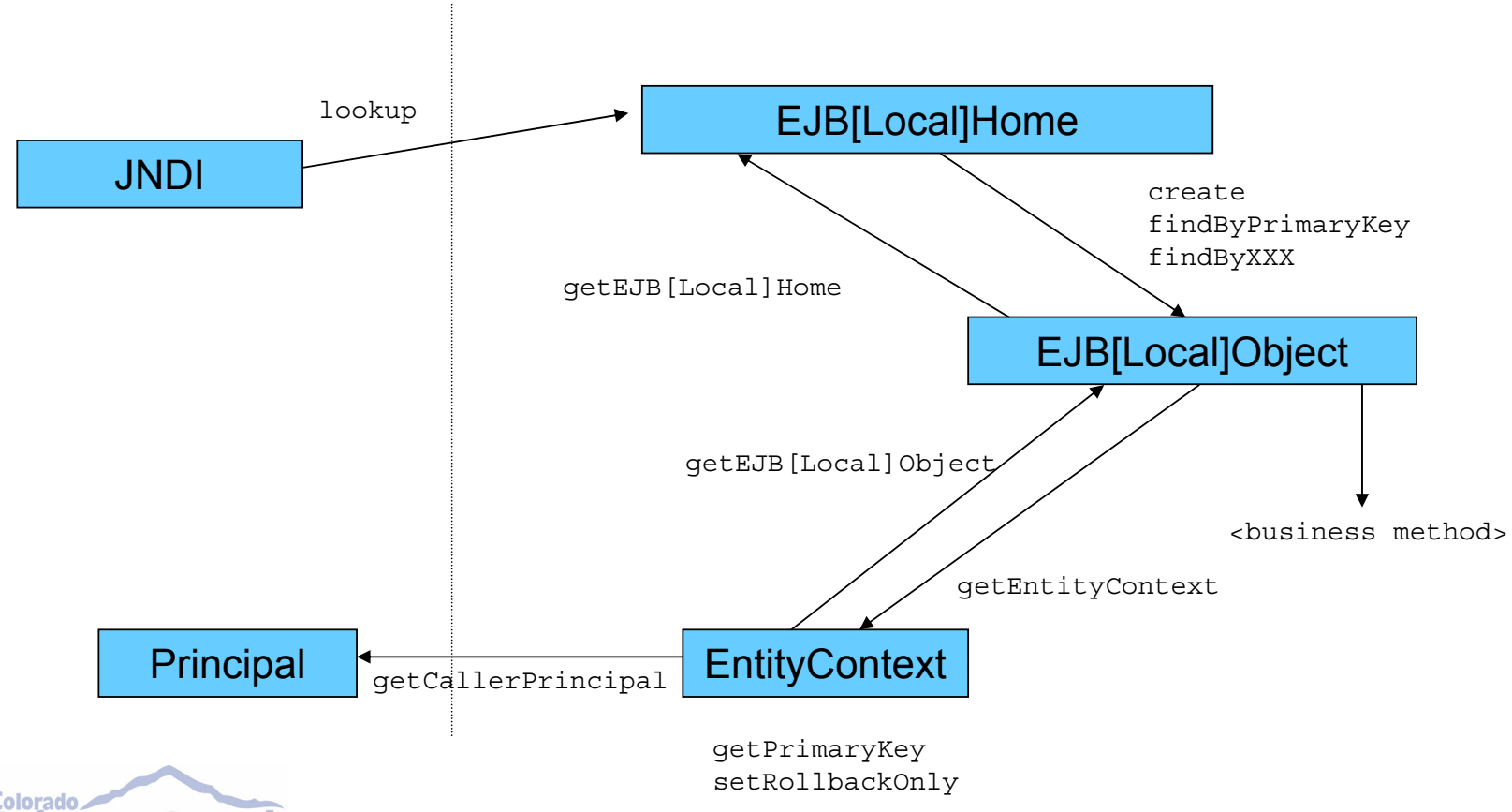
"Evolution of the EJB Entity"

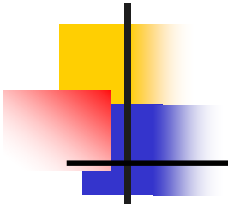
Michael Keith, Oracle Corp.

EJB Architecture View



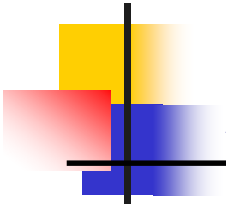
Basic EJB Model





EJB Model

- Component model provides solid separation between interface and implementation
- Separate pre-compile step is needed for the EJB Container to generate deployable JAR containing generated classes:
  - Bean implementation subclasses
  - [Local]Home classes
  - EJB[Local]Object implementations



Anti-Practice

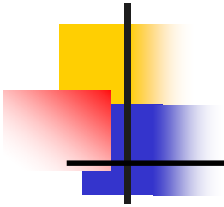
- Early EJB entities were all remote-enabled so people used them from remote clients

  Don't use remote entities

  Remote entities will not even be around in 3.0

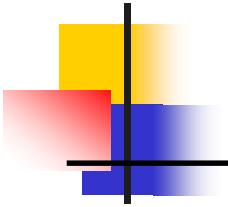
- Session beans were not used as often as they are now, so business logic got included in the entity classes

  Don't pile business logic into entities



Reading/Query Language

- Queries are defined statically in XML metadata
- EJB QL query language is "SQL-like"
- Look up the "home" object for the target target class and invoke the appropriate finder method on it
- ejbSelect queries are special queries that can be executed from within the bean object and can do data projection
- Can also return other types of objects from ejbSelect calls

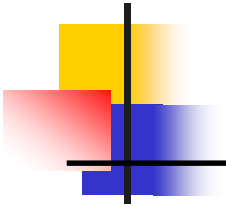


Reading/Query Language

```
Context ctx = new InitialContext();
DepartmentHome deptHome = (DepartmentHome)
    ctx.lookup("java:comp/env/ejb/DepartmentHome");
Collection depts = deptHome.findLargeDepts(empCount);
```

In XML descriptor:

```
<ejb-ql>
  <![CDATA[ SELECT OBJECT(d) FROM Department AS d,
        IN (d.emps) e WHERE e.salary > ?1 >]]
</ejb-ql>
```



Reading/Query Language

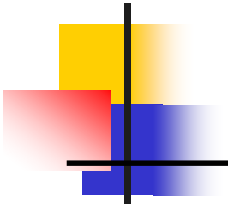
```
int bigShot = empHome.getLargestSalary();
```

getLargestSalary() method on EmployeeHome interface:

```
public int getLargestSalary();
```

ejbHomeGetLargestSalary() method on EmployeeBean:

```
public int ejbHomeGetLargestSalary() {
    return ejbSelectGetLargestSalary();
};
```



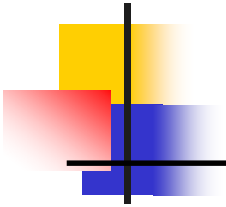
Reading/Query Language

ejbSelectGetLargestSalary() method on EmployeeBean:

```
// Impl gets generated by the EJB container
public int ejbSelectGetLargestSalary();
```

In XML descriptor:

```
<ejb-ql>
  ![CDATA[ SELECT MAX e.salary FROM Employee e ]]
</ejb-ql>
```



Writing

Creating:

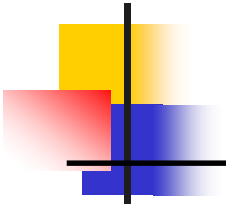
- Must call a home.create() factory method, passing in the primary key for the new object

Updating:

- Modify the entity in the context of a transaction and commit the transaction

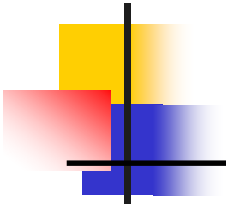
Deleting:

- Call bean.remove()
- Can use home.remove(pk) if bean has not already been invoked upon
- Parent deleted and relationship set to cascade-delete



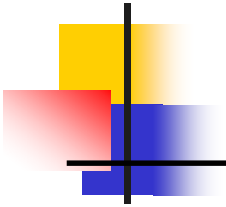
Transactions

- Integral with JTA TransactionManager
- Business method transaction attributes are declaratively stated at deployment-time
  - NotSupported
  - Required
  - Supports
  - RequiresNew
  - Mandatory
  - Never
- May envelope entity method invocation, or may piggy-back on transaction already in progress
- Accessed through EntityContext, or JNDI



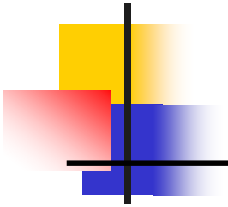
Caching

- Bean instance cache fairly well-specified
- Life cycle of cached instances is revealed to the application through callback methods on the instances
- Different caching options w.r.t. the state in the bean are described by the specification
  - Option C – Must take a new bean instance and set its PK and context for every transaction
  - Option B – Can leave the PK and context in the instance across transactions, but bean state must be reloaded at the beginning of every transaction
  - Option A – Can trust that the bean state is always synchronized with the data store, no need to reload the data on every transaction boundary



Locking

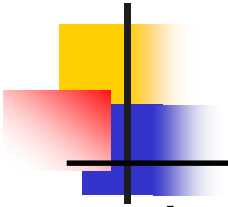
- Beans can be declared as being reentrant or non-reentrant
- Reentrancy implies that a bean that may already have a method invocation in progress may be invoked from the same or a different thread within the same transaction
- No specific transaction isolation support is included in the specification



Configuration

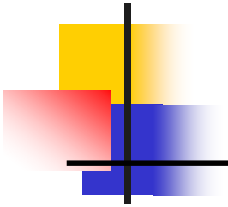
- Metadata for all entities must be declared in XML file(s)
  - META-INF/ejb-jar.xml
  - Application server-dependent XML file(s)
- Used at EJB compile/build-time and normally at deployment-time
- Specifies everything about the entities that the EJB Container needs:

  Ex.
  - The names of the home and component interfaces
  - The relationship structure and multiplicity
  - The queries and search criteria for the queries



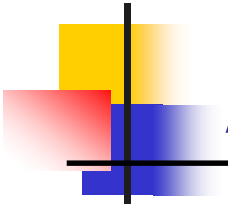
Configuration

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ejb-jar xmlns=http://java.sun.com/xml/ns/j2ee
    xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/ejb-jar_2_1.xsd" version="2.1">
<enterprise-beans>
    <entity>
        <ejb-name>Employee</ejb-name>
        <local-home>org.acme.EmployeeHome</local-home>
        <local> org.acme.Employee</local>
        <ejb-class>org.acme.EmployeeBean</ejb-class>
        <persistence-type>Container</persistence-type>
        <prim-key-class> org.acme.EmployeePK</prim-key-class>
        <reentrant>False</reentrant>
        <cmp-version>2.x</cmp-version>
        <abstract-schema-name>Emp</abstract-schema-name>
        <cmp-field>
            <field-name>id</field-name>
        </cmp-field>
        <cmp-field>
            <field-name>name</field-name>
        </cmp-field>
        <cmp-field>
            <field-name>salary</field-name>
        </cmp-field>
```



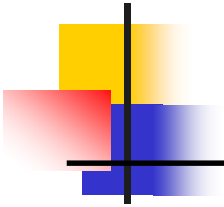
Configuration

```
            <query>
                <query-method>
                    <method-name>ejbSelectGetLargestSalary</method-name>
                    <method-params/>
                </query-method>
                <ejb-ql>SELECT MAX e.salary FROM Emp e</ejb-ql>
            </query>
        </entity>
         . . .
    <enterprise-beans>
    <relationships>
       . . .
    </relationships>
    <assembly-descriptor>
        <container-transaction>
            . . .
        </container-transaction>
    </assembly-descriptor>
    </ejb-jar>
```



Agenda

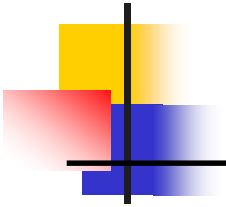
- The Chaos
- Focus on Persistence
- JDBC
- JDO
- EJB
- Summary



Summary

JDBC:

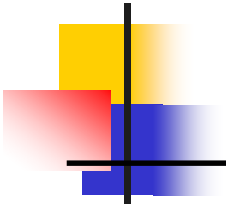
- "Low-level" standard that is here to stay (even without James Gosling's blessing!)
- Has maximum customization potential, superior performance tuning control and virtually no limits to the SQL that can be executed
- Connection artifacts must be managed by the application
- Requires writing lots of O-R mapping/translation code
- Much harder to maintain and doesn't scale to large numbers of objects and tables
- Hard-coded SQL makes it very difficult to port to a different database



Summary

JDO:

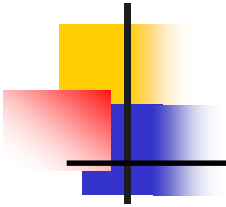
- Can be used outside of an app server (not coupled to J2EE)
- Offers most of the needed API and query features
- Includes comprehensive test suite
- Binary compatibility provides vendor interop (in theory) but introduces unnecessary complexity
- Not part of J2EE specification
- No major vendor implementations
- No standardized O-R mapping layer (is provided by vendors) or facility to detach and re-attach objects



Summary

EJB:

- Mature and ubiquitous specification
- Integration with other specs (integral part of J2EE)
- Integrated support for security, threading, transactions
- All major vendors have implementations
- Coarse-grained objects and overhead can lead to performance degradation
- Complicated XML deployment descriptors hard to construct without tools
- No standardized O-R mapping layer (is provided by vendors) or facility to detach and re-attach objects



Summary

- Despite their benefits, standards are not the most popular technology right now
- Standards don't currently provide enough of the features that applications require
- Persistence standards do not make good on their portability promise
- Given the time and the correct mandate the standards will catch up
- 2005 should the year of blue-collar persistence!