J2EE Persistence Options: JDO, Hibernate and EJB 3.0

Sridhar Reddy

Sridhar.Reddy@sun.com

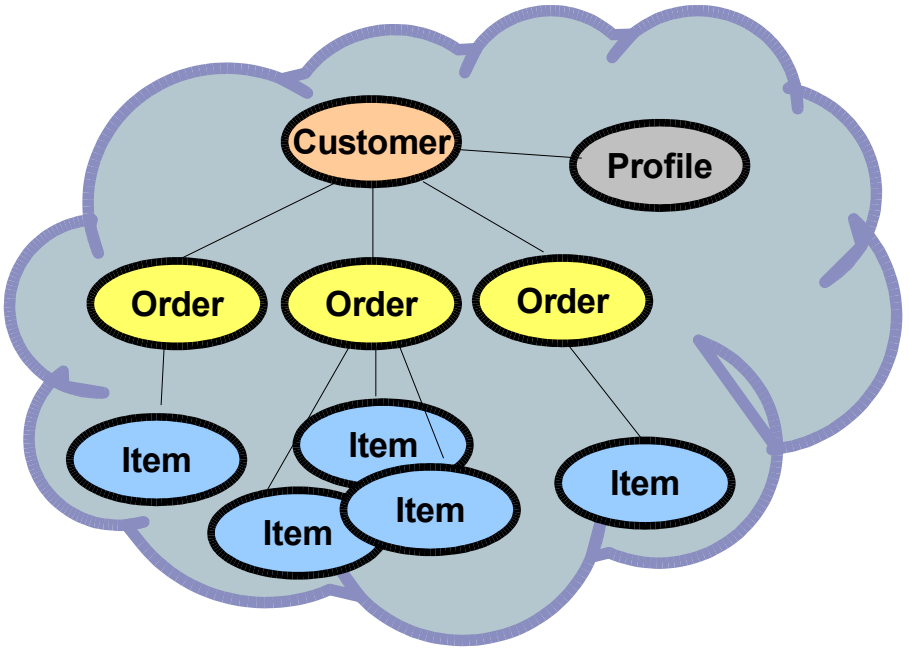
Sun™ Tech Days

The Landscape

Persistence In A Nut Shell

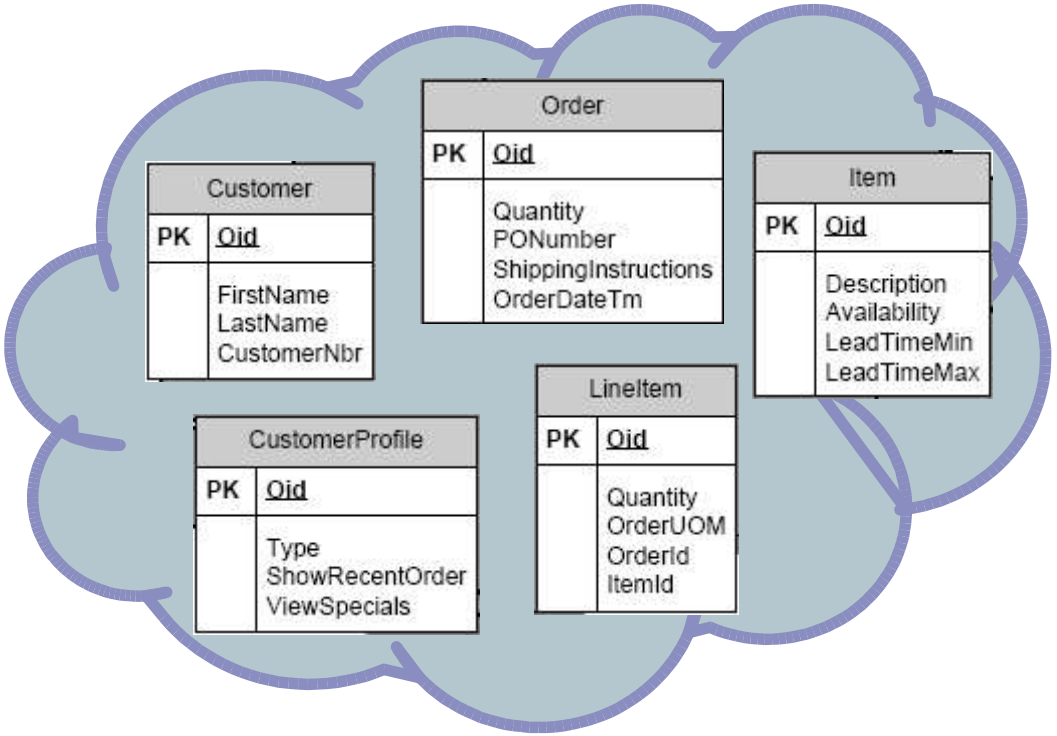
- Persistence
  - Saving your (persistent) application data
  - Mapping your component/object model to the persistence store (typically referred to as O/R mapping)
- Data consistency and concurrent access
- Transactional semantics
- Managing your persistent state is non-trivial and complex

Object-Relational Impedance Mismatch



The Logical World

The Physical World



Persistence Techniques

- Entity EJB
  - Bean-Managed Persistence (BMP)
  - Container-Managed Persistence (CMP)
- JDBC
  - Relational databases (RDBMS)
  - Object databases (ODBMS)
- Java Data Objects (JDO)
- Hibernate
- ObjectRelationalBridge (OJB)

Approaches to Object Persistence

Functional Approach

- Better suited for coarse-grained persistent business objects
- Class author or tool implement a standard set of functions to persistent-enable their domain classes
- Two models programmatically reflect the same domain
- Ex: Entity Bean persistence framework

Approaches to Object Persistence

Orthogonal Language Transparent Approach

- Suited for both coarse-grained and fine-grained persistent domain objects
- Persistent objects in the code are just like any other objects
- Application design decoupled and independent of the underlying persistence infrastructure
- Ex: JDO persistence framework, Hibernate

Can The Two Co-exist?

- YES!
- Transparent persistence with Java Data Objects (JDO), Hibernate, and others are *not* replacements for Enterprise JavaBeans (EJB) architecture
- They *complement* it!

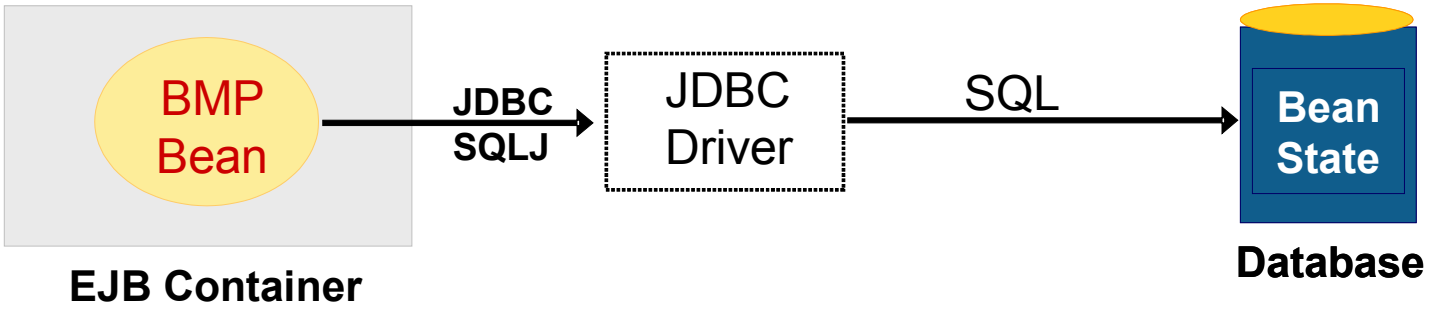
Push your development further

Container-Managed Persistence (CMP)

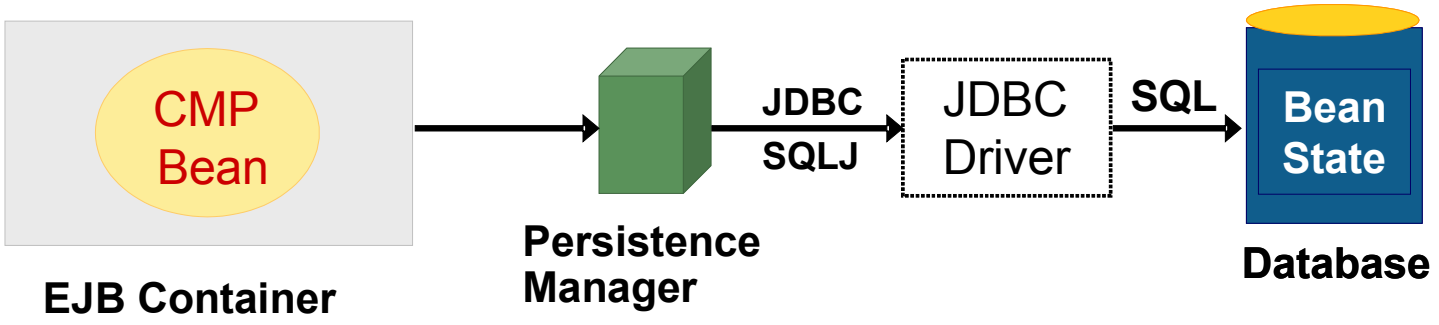
CMP 2.0

- Rich modeling capability with relationships
  - Referential integrity, Cardinality, Cascading delete
- Container manages the relationships not you!
- Freedom from maintaining interactions with the data store
- EJB Query Language (EJB-QL)
- Truly portable code!

BMP vs. CMP



1) Bean provider manages State and Data consistency
2) Bean provider handles relationships and OR Mapping



1) Container manages State and Data consistency
2) Container/PM provides concurrency, relationships and OR Mapping

Role of the Persistence Mgr

- O/R mapping
- Managing the persistence state
- Relationships management
- Concrete bean sub-class generation
- QoS (e.g., Data caching)

Roles of the Container

- Basic “wrapper” code generation
- Making the info from ejb-jar.xml available to the PM during deployment
- Life cycle management
- Making the Transaction Manager available to the PM

CMP 2.0

- Use CMP 2.0 whenever possible!
- Performs better than BMP
- Improved portability, performance over CMP 1.0
- Easier to develop and deploy than BMP
- Produces portable code over multiple databases
- If you have to build BMP entity bean, subclass CMP 2.0 bean
  - Easy migration to CMP later on

Sun microsystems

Push your development further

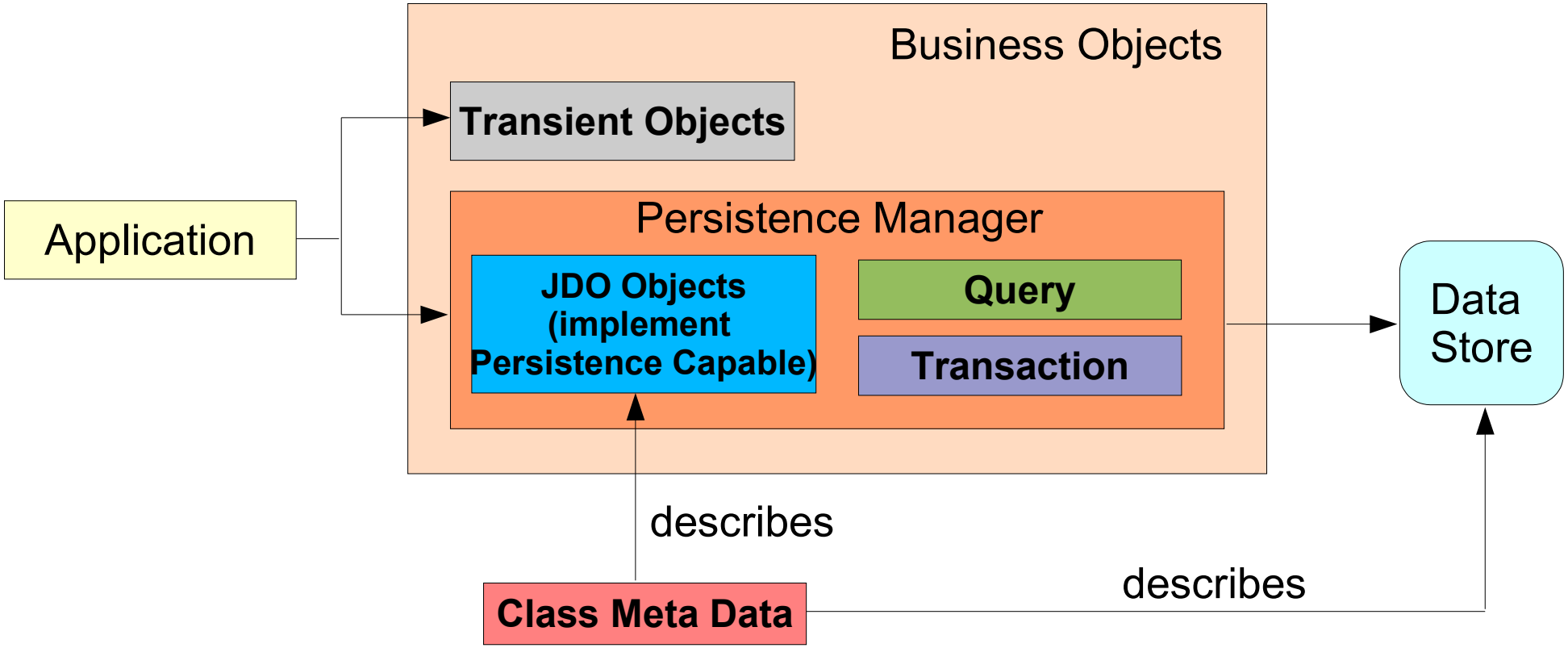
Java Data Objects (JDO)

JDO

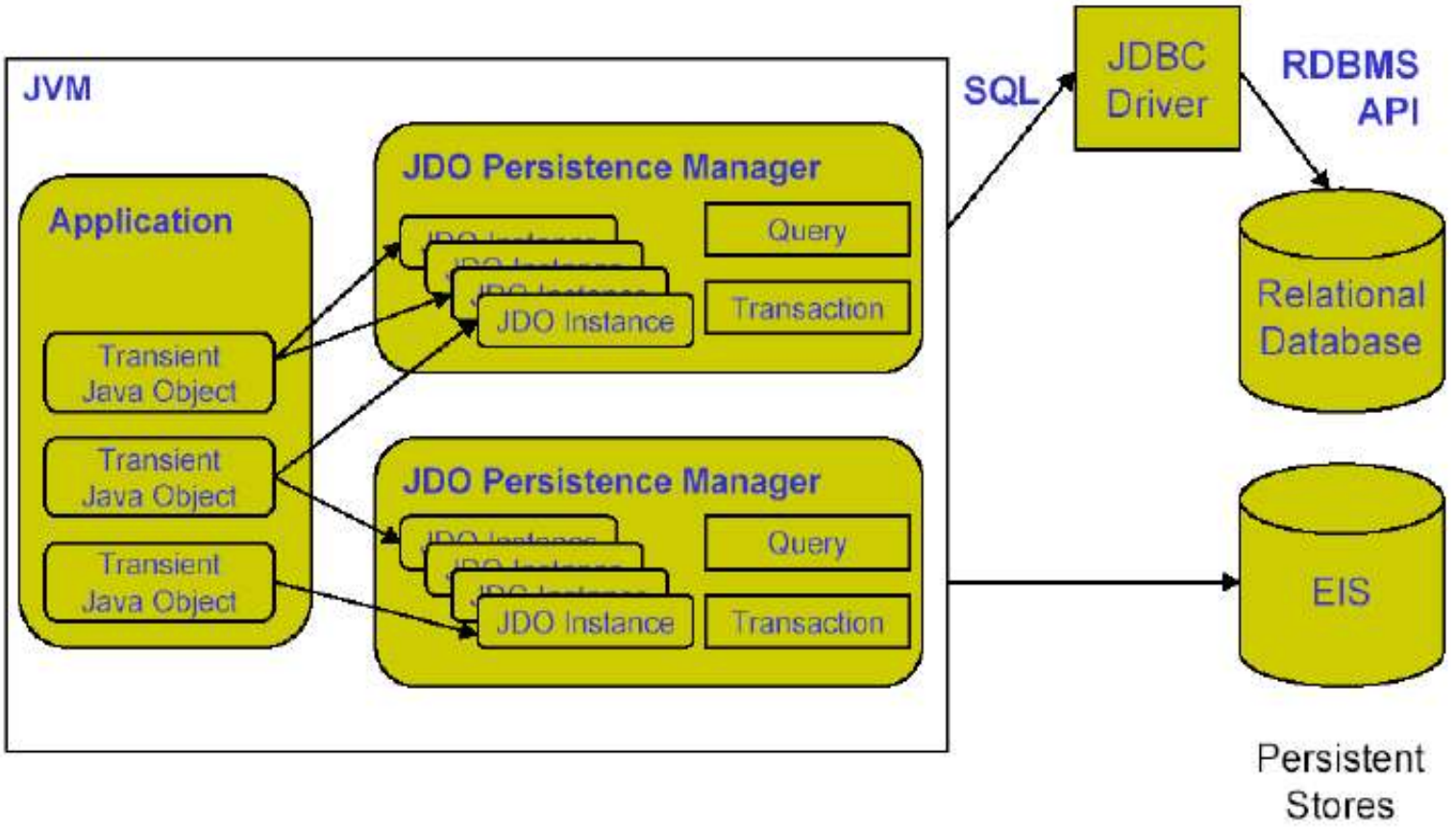
- Standard for generic/transparent Java object persistence
  - Provides developers with a Java-centric and object view of persistence and data store access
- Designed to allow pluggable vendor “drivers” for accessing any database/data store
- Connector Architecture used to specify the contract between JDO Vendor and Application Server for instance, connection, and transaction management

JDO Architecture

- Persistence By Reachability
  - Any object loaded directly or indirectly (by reference) from a JDO loaded object is automatically persisted if the enclosing transaction commits



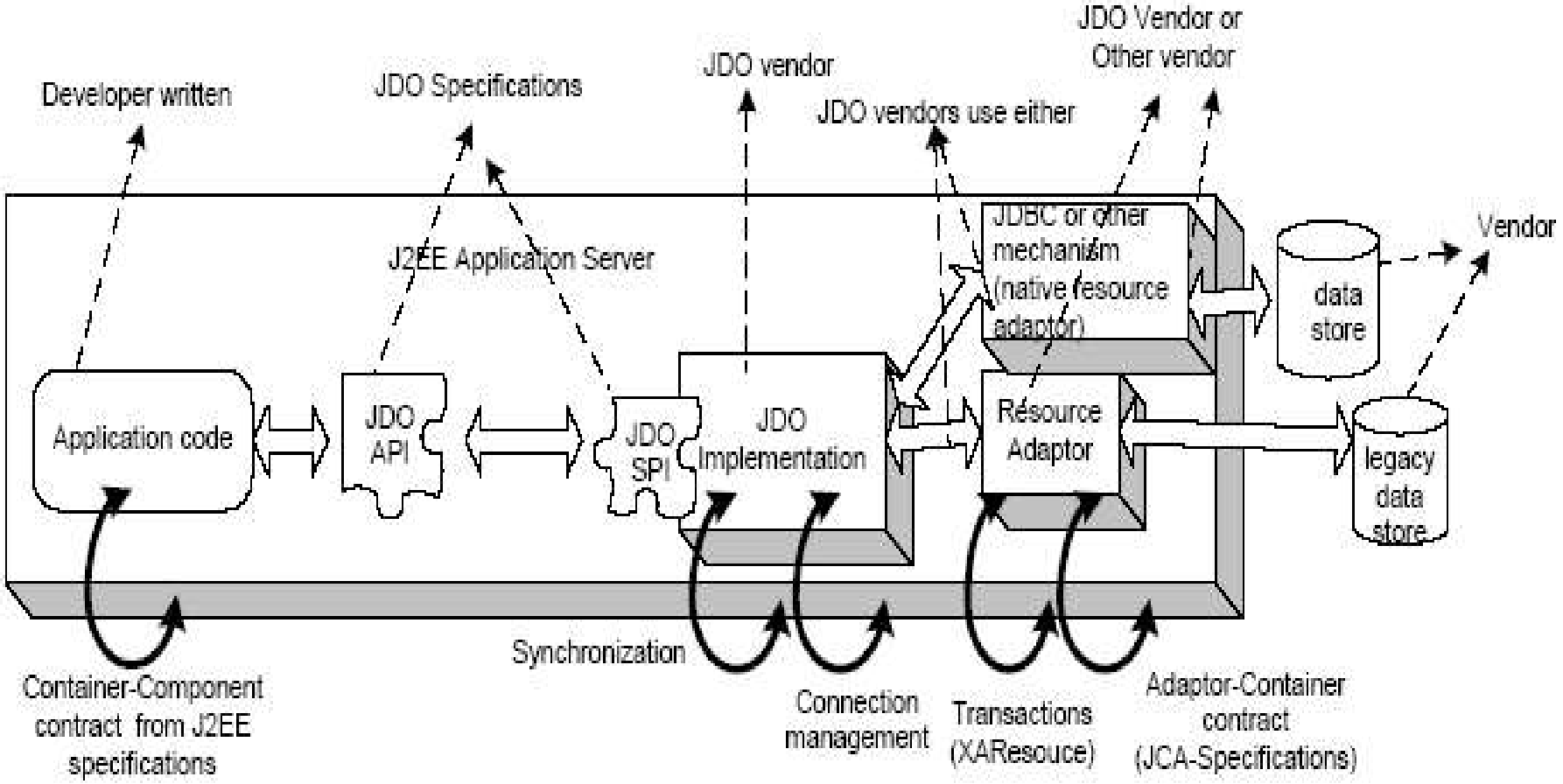
JDO Non-Managed Runtime



Managed Environment

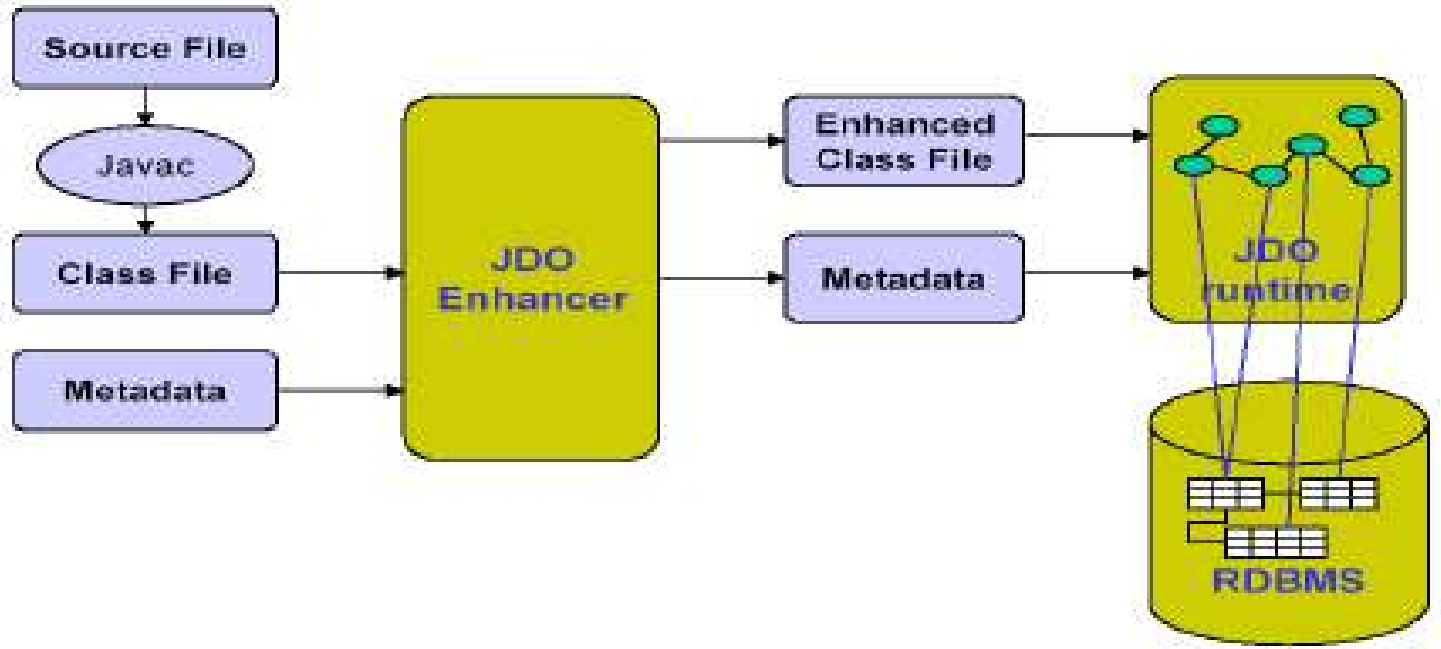
J2EE-based, multi-tier

- Lifetime of PM, pooling, and caching limited to transaction scope



Byte Code Enhancement

- Most JDO vendors use the bytecode modification for the following reasons:
  - Avoid potentially messy source code modification
  - Allow persistence to be hidden from the programmer. The programmer is database unaware



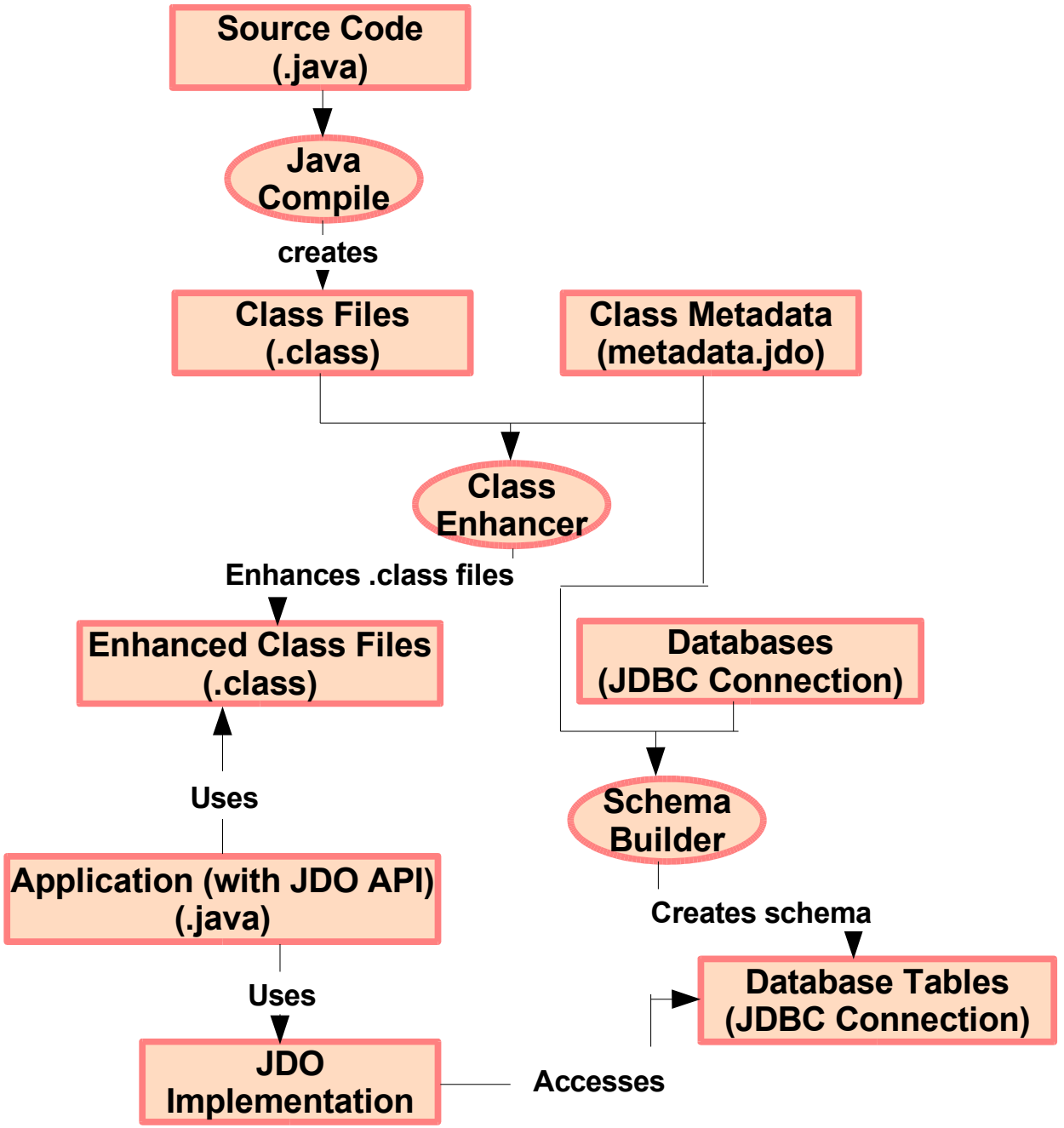
JDO Interfaces and Classes

- Use PersistenceManagerFactory to get a PersistenceManager
  - PersistenceManager embodies a database connection
- Use a PersistenceManager to create a Transaction or a Query
- Use a Transaction to control transaction boundaries
- Use a Query to find objects

JDO Interfaces and Classes (cont)

- Enhanced classes implicitly implement PersistenceCapable
- PersistenceCapable classes can implement InstanceCallbacks

JDO Deployment Process



JDO API For Persistence

```
Public static void main(String[] args) {
PersistenceManagerFactory pmf =
   JDOHelper.getPersistenceManagerFactory
(System.getProperties());
PersistenceManager pm =
   pmf.getPersistenceManager();
Transaction tx = pm.currentTransaction();
tx.begin();
   Author author = new Author("Mr. Author");
   Book book = new Book("Java Book",
                        "0-11-570731-7");
   author.addBook(book);

// do some other work with books, publishers etc
   pm.makePersistent(author);

tx.commit();
pm.close();
}
```

Hibernate

Hibernate (old)

- Persistence for JavaBeans
- Explicit Save/Update for each object
  - Changed objects must be enlisted for update so that the next transaction commit will effectuate the changes in the data store
- Support for very fine-grained, richly typed object models
- Support for detached persistent objects
- Hibernate Query Language (HQL)

Hibernate

- Persistence for POJOs (JavaBeans)
- Flexible and intutive mapping
- Explicit Save/Update for each object
  - Changed objects must be enlisted for update so that the next transaction commit will effectuate the changes in the data store
- Support for very fine-grained, richly typed object models
- Support for detached persistent objects
- Hibernate Query Language (HQL)

Hibernate

- Persistence for POJOs (JavaBeans)
- Flexible and intutive mapping
- Explicit Save/Update for each object
  - Changed objects must be enlisted for update so that the next transaction commit will effectuate the changes in the data store
- Support for very fine-grained, richly typed object models
- Support for detached persistent objects
- Hibernate Query Language (HQL)

Detached Object Support

- For applications using servlets + session beans
  - Row “select” not needed for updating
- DTO's not necessary
- You may serialize objects to the web tier, then serialize them back to the EJB tier in the next request
- Hibernate lets you selectively reassociate a subgraph
  - Performance advantage

Sun microsystems

Push your development further

Complementing J2EE and JDO/Hibernate

JDO In Transaction-Managed Environment

- Because JDO is *only* concerned with persistence, it is best used within protective boundaries of a J2EE application server

JDO Access Scheme

```
// factory via JNDI
PersistenceManagerFactory pmf = ...;

// Create a session-scoped persistence manager
PersistenceManager sess =
      pmf.getPersistenceManager();

// Get object
Person p = (Person)
      sess.getObjectById(new PersonPK("100170"));

// Set/Get properties as for any POJO
p.setName("Peter Jensen");
int year = p.getYearOfBirth();
```

Hibernate In Transaction-Managed Environment

Hibernate Access Scheme

```
// factory via JNDI
SessionFactory sessionFactory = ...;

// Create a session-scoped persistence manager
Session sess =
      sessionFactory.getSession();

// Get object
Person p = (Person)
      sess.loadClass(Person.class, "100170");

// Set/Get properties as for any POJO
p.setName("Peter Jensen");
int year = p.getYearOfBirth();

sess.saveOrUpdate(p);
```

JDO and EJB

- JDO works well with Session and Message-Driven Beans
  - Works out-of-the-box
  - Bean explicitly controls transactions or uses CMT
- JDO can be used with BMP Entity Beans
- JDO can be used by containers for CMP

JDO and EJB

- Usage with Session, Message-Driven Bean
  - Fascade pattern
- Bean methods represent coarse-grained business processes
  - Business logic uses JDO to represent the data model
- Bean programming can still leverage standard OO concepts
- JDO object can be used in Data Transfer Object Pattern

JDO and Stateless Session Beans


```
public class ExampleCMTBeanWithJDO implements SessionBean{
private SessionContext ejbCtx;
private PersistenceManagerFactory jdoPMF;

public void setSessionContext(SessionContext sessionCtx) throws
EJBException{
ejbCtx = sessionCtx;
InitialContext ctx = new InitialContext();
Object o = ctx.lookup("java:comp/env/jdo/bookstorePMF");
jdoPMF = (PersistenceManagerFactory)
PortableRemoteObject.narrow
                (o,PersistenceManagerFactory.class);...}

  /* business method */

  public void doSomething(int arg){
  PersistenceManager pm = jdoPMF.getPersistenceManager();

  // Do something using JDO now...

pm.close();

}                                                          }
```

JDO and EJB

- JDO can be used as a BMP strategy
  - Sun Java System App Server uses JDO for CMP
- Leverages J2CA
- Layered Architecture
  - Use JDO objects directly
  - Use same objects within EJB to take advantage of other J2EE container services
- JDO/BMP approach – cost effective
- Entity Bean and JDO usage similiarities
  - Not distributed, non-remote, not access-controlled

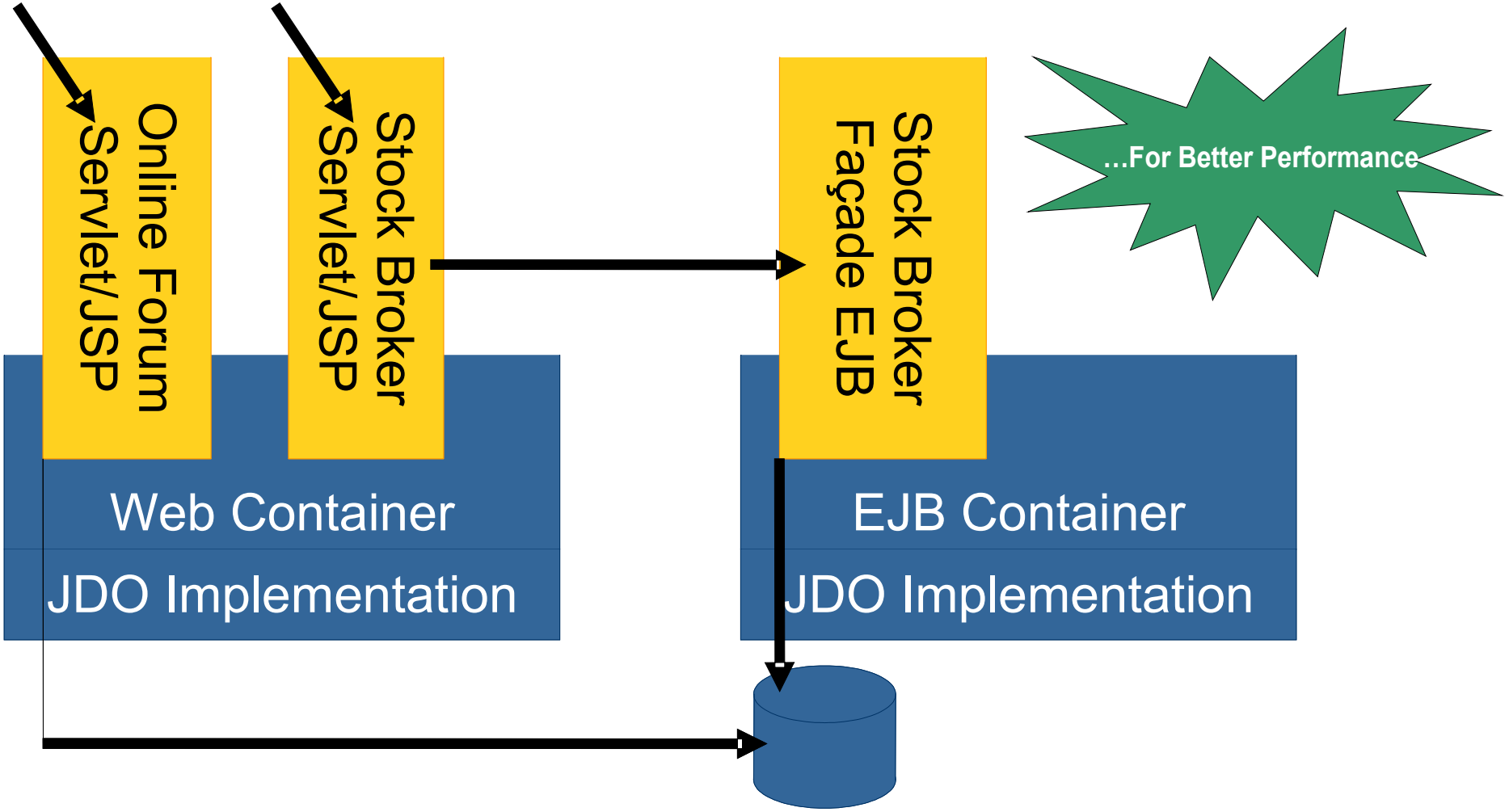
JDO and EJB

- Entity Beans Wrapping JDO classes
  - Use a BMP delegate strategy
  - Allows the JDO classes to be remotely accessible directly, rather than through a session facade

JDO and J2CA

- Java Connector Architecture
  - Mandated as plug-in for non-JDBC data access
- Common Client Interface
  - Standard APIs for obtaining a connection
    - `javax.resource.cci.ConnectionFactory`
    - `javax.resource.cci.Connection`
  - `PersistenceManagerFactory -> ConnectionFactory`
  - `PersistenceManager -> Connection`
- JDO PMF is bound to JNDI by a J2CA adaptor

Access From Application Clients Directly



- Techniques for concurrency modifications
  - JDO – Store PersistenceManager in HTTP Session
  - Hibernate – Store Data Objects in HTTP Session

JDO and Servlets

```
public class JDOServlet extends HttpServlet{

PersistenceManagerFactory pmf;
public void init(ServletConfig config);
InitialContext ctx = new InitialContext();
pmf = (PersistenceManagerFactory)
ctx.lookup("java:comp/env/PMF");}

protected void service(HttpServletRequest request,
                       HttpServletResponse response){

String authorname = request.getParameter("authorname");
PersistenceManager pm = pmf.getPersistenceManager();
Transaction tx = pm.currentTransaction();
tx.begin();

Author author = new Author("authorname");pm.makePersistent
(author);

tx.commit();

}
```

Sun microsystems

Push your development further

Choosing Your Strategy

JDBC

Sun™ Tech Days

Sun microsystems

Application Suitability

- Moderate development cycle
  - No intermediate step needed
- Dependence on SQL to handle computational logic
- Ideal for RDBMS-centric apps
  - Harness RDBMS computational power
  - Tight coupling – domain object models and database schema

JDBC

Sun™ Tech Days

Sun microsystems

Application Suitability

- Lack of client caching
- *Moderately* portable code

JDBC

Sun™ Tech Days

Sun microsystems

Object Query

- Uses SQL
  - Refers directly to the data store schema
- Query sent *directly* to the data store as String arguments
- No statement error detection at compile time
- Excel at Aggregational Queries
- Computational load on the RDBMS

JDBC

Sun™ Tech Days

Sun microsystems

Object Query

- Driver explicitly fetches records from the data source for Navigational queries
- Ad-hoc results possible
- Query results returned as ResultSet
  - *High* overhead for marshalling query results

CMP

Application Suitability

- Application server has rich feature set
  - Load balancing, transactions, security, messaging, etc
- *High* performance optimization
- Fetch-on-demand
- *Highly* portable code
  - BMP less so
- Development can be non-linear

CMP

Object Query

- Declarative query using abstract finder methods in bean interface
- Deployment descriptor defines how the finder is realized in EJB-QL
- Compiler translates EJB-QL query to syntax of target data store
- Compiler inserts execution statements into generated concrete bean class

CMP

Sun™ Tech Days

Sun microsystems

Object Query

- Ideal for Navigation Query
  - Driver implicitly fetches instances from the data source
- Domain object model decoupled from schema and data store
  - Cannot assume specific query capabilities of the data store
- Ad-hoc queries not possible

CMP

Object Query

- Query results returned as domain objects
  - *Low* overhead for marshalling query results
- Provides advanced declarative transaction semantics
- EJB-QL Definition in Server-side descriptors
- No Access to Statement Generation
- Simplicity in Database Mapping

Application Suitability

- Application code relatively simple
  - JDO driver handles automatic persistence, mapping, and identification in transactions
- Ideal for object-centric applications
- Ideal when data store is primarily navigational
- Object methods handle bulk of data store computational logic

JDO

Sun™ Tech Days

Sun microsystems

Application Suitability

- Ideal when working with multiple types of data stores
- Ideal for navigation access over a graph of interconnected objects
  - Persistence-by-reachability
- *Moderate* performance optimization
  - Client cache managed by JDO implementation
- *Highly* portable code

JDO

Object Query

- Programmatic approach using Java-like syntax
- Query represented as instance of javax.jdo.Query object
- Attributes refer only to elements in the Java application space
- JDO driver translates query into syntax of target data store when execute( ) method invoked on Query object

JDO

Object Query

- JDO-QL used
- Computational load on the client
- Ideal for Navigation Query
  - Driver implicitly fetches instances from the data source
- Domain object model decoupled from schema and data store
  - Cannot assume specific query capabilities of the data store

JDO

Object Query

- Ad-hoc Queries possible
- Query results returned as domain objects
  - *Low* overhead for marshalling query results
- Persistence-by-reachability enable strong transaction state management

EJB 3.0

EJB 3.0 Goals for CMP

- Simplify Programming Model
  - Reduce number of programming artifacts
  - POJO/JavaBeans like other EJB 3.0 beans
  - Eliminate deployment descriptor from developer's view
- Make instances usable outside the container
  - Facilitate testability
  - Remove need for data transfer objects (DTOs)

EJB 3.0 Goals for CMP

- Support for lightweight domain modeling
  - Inheritance and polymorphism
  - Object/Relational mapping metadata
- More complete query capabilities

EJB 2.0 Persistent Model

- Enables light-weight implementation
  - Entities typically accessed through local interfaces
  - Transactions typically started in session bean or Web tier
  - Methods are often “unchecked”
- Provides a standardized SQL-like query language integrated with entity model EJB-QL
- Usage has supplanted that of BMP
  - Held back by need for more EJB-QL
  - High-quality, high-performance implementations well-established in industry

EJB Limitations

- Lack of sufficient modeling capabilities
  - No polymorphism
  - No support for implementation inheritance
  - Lacks O/R mapping specification
- Query language still missing some important features
  - Projection, Subqueries, Outer joins, Dynamic queries
  - Provision for direct SQL queries

POJO Entity Beans

- Concrete classes (no longer abstract)
- No required bean interfaces
- Support new()
- Usable outside the EJB container
  - As detached entities
  - For testing of business logic
- getter/setter methods
  - Can contain logic (for validation, transformation, etc.)

POJO Entity Beans

- Collection interfaces for relationships
- Entities are not remotable
- No required callback interfaces
- Many points of control
  - Over lifecycle
    - CASCADE capabilities (CREATE, REMOVE, ALL, ...)
  - Scope of persistence context
  - Fetch/faulting behavior
    - FETCH JOINS, O/R mapping metadata
  - Isolation semantics

EJB 3.0 Entity Bean Example

```
@Entity public class Customer {
  private Long id;
  private String name;
  private Address address;
  private HashSet orders = new HashSet();

  @Id(generate=AUTO) public Long getID() {
    return id;
  }

  protected void setID (Long id) {
    this.id = id;
  }

  @OneToMany(cascade=ALL)
  public Set<Order> getOrders() {
    return orders;
  }

  public void setOrders(Set<Order> orders) {
    this.orders = orders;
  }}
```

EJB 3.0 Client Example

```
@Stateless public class OrderEntryBean {
  private EntityManager em;

  @Inject void setEntityManager(EntityManager em){
    this.em = em;
  }

  public void enterOrder(int custID, Order newOrder){
    Customer c = em.find("Customer", custID);
    c.getOrders().add(newOrder);
    newOrder.setCustomer(c);
    }

  // other business methods
  }
```

EJB QL Enhancements

- Bulk update and delete operations
- Projection list (SELECT clause)
- Group by, Having
- Subqueries (correlated and not)
- Additional SQL functions
  - UPPER, LOWER, TRIM, CURRENT_DATE, ...
- Dynamic queries
- Polymorphic queries
- Criteria queries probably in 3.1

Persistence Context and Metadata Access Updates

- Outer Fetch Joins
  - Very useful for explicitly controlling data prefetch
  - Minimize database roundtrips
  - Programmer knows data access patterns
- Relationship fetching
  - FetchType, EAGER, LAZY
- Optimistic locking support
  - @Version, @Timestamp

Inheritance Mapping Example

```
@Entity
@Table(name="CUST")
@Inheritance(strategy=SINGLE_TABLE,
             discriminatorType=STRING,
             discriminatorValue="CUST")
public class Customer {...}

@Entity
@Inheritance(discriminatorValue="VCUST")
public class ValuedCustomer extends Customer{...}
```

Extended Persistence Context

- Persistence context typically corresponds to a single JTA transaction
- Extended persistence context can span multiple JTA transactions
  - Important use case: “Application transactions”
  - Preserves state across longer-lived client interactions (especially from Web tier)
  - Stateful session beans a natural fit for maintaining extended persistence context
  - Optimistic transactions

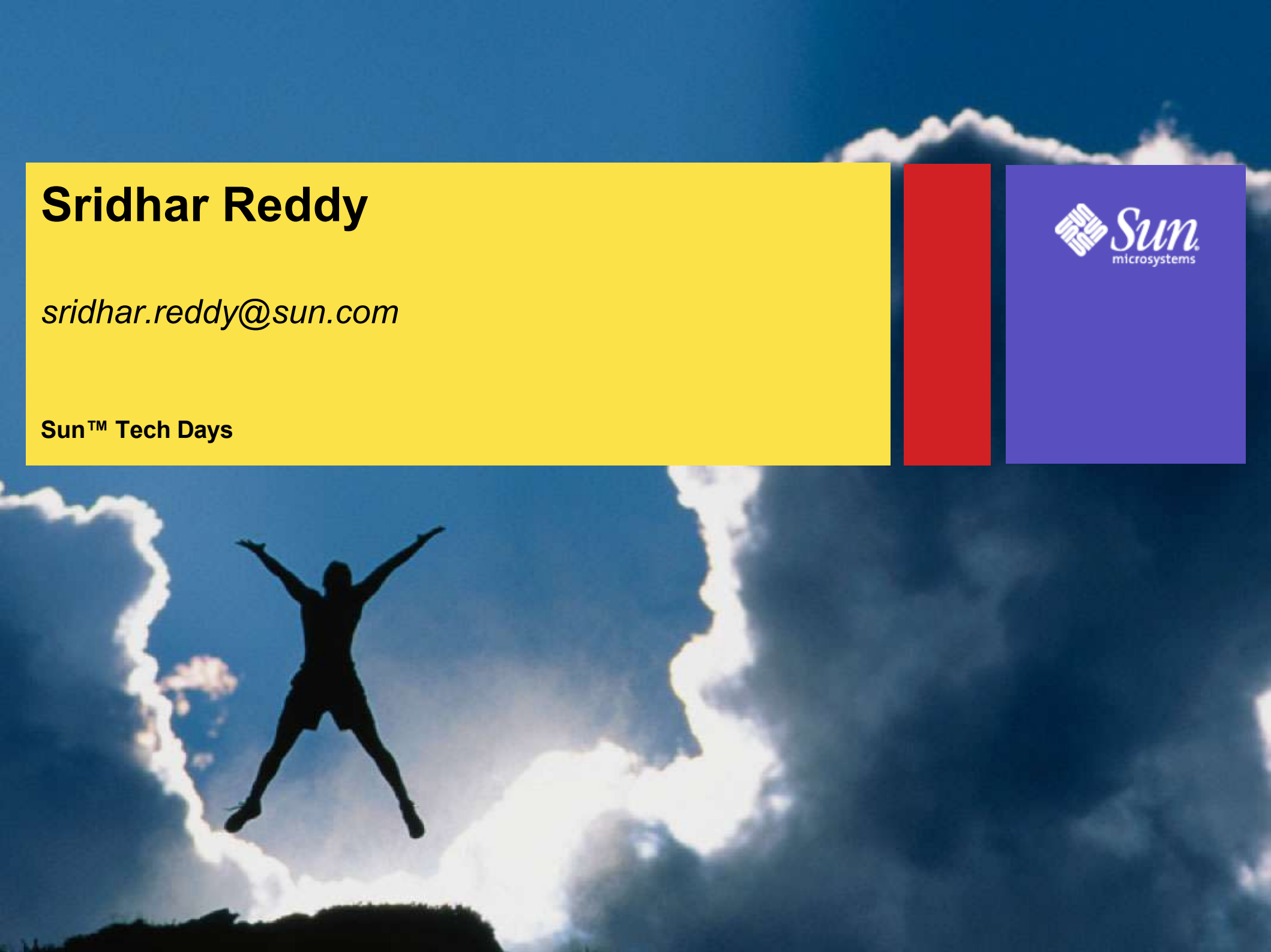
Resources

- J2EE
  - http://java.sun.com/j2ee/index.jsp
- JDO
  - http://access1.sun.com/jdo/
  - http://www.jdocentral.com
- Hibernate
  - http://www.hibernate.org
- OJB
  - http://db.apache.org/ojb/

Q&A

Sun microsystems

Push your development further



Sridhar Reddy

sridhar.reddy@sun.com

Sun™ Tech Days

Sun microsystems

Push your development further

Demo

CMP Development on Sun Java Studio Enterprise