The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle’s products remains at the sole discretion of Oracle.


Java™

MOVING JAVA FORWARD

ORACLE

EclipseLink: The Evolution of Java Persistence

Shaun Smith
shaun.smith@oracle.com /  @shaunMsmith

About Me

- From Toronto, Canada

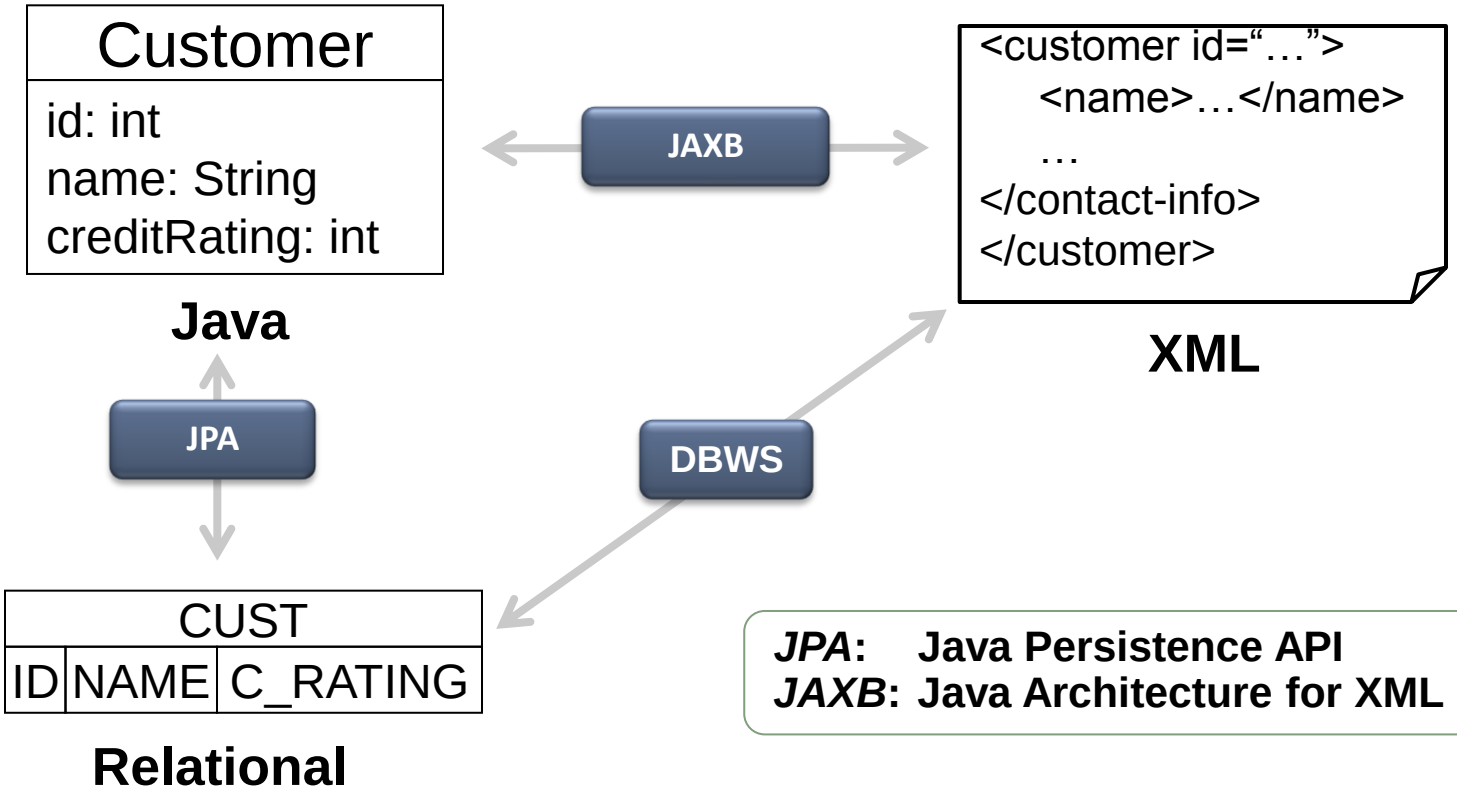
- Product Manager at Oracle for TopLink
- Object-Relational Mapping since ’96!
- Committer on various Eclipse projects including EclipseLink & Gemini
- Presented at many conferences including JavaOne, Devoxx, QCon, EclipseCon, & JAX

Agenda

- Introduction
- Evolutionary Pressures
- New EclipseLink Features
- Conclusion

JAVA PERSISTENCE

Java Persistence: The Problem Space

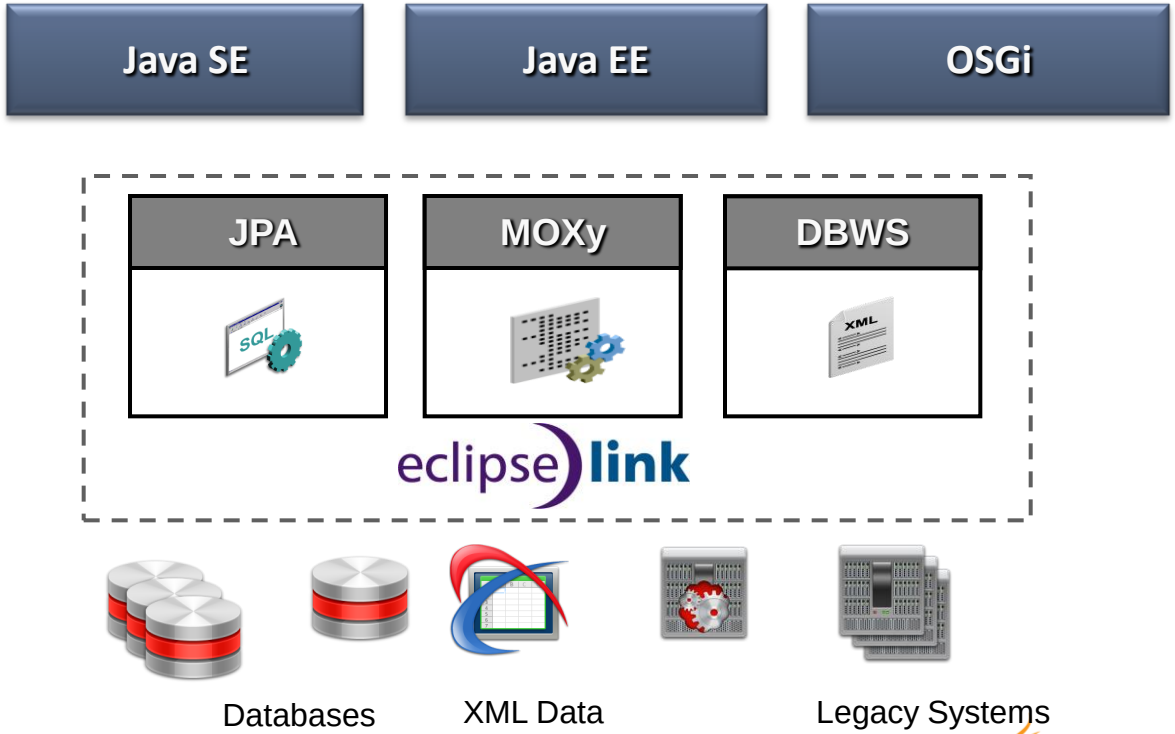


***JPA*: Java Persistence API**
***JAXB*: Java Architecture for XML Binding**

EclipseLink Project

- Object-Relational: Java Persistence API (JPA)
  - JPA 1.0 part of EJB 3.0 standard (JSR 220)
  - JPA 2.0 standardized in JSR 317
  - EclipseLink is *JPA 2.0 & 2.1 Reference Implementation*
- Object-XML: Java Architecture for XML Binding (JAXB)
  - JAXB 2.2 Certified Implementation
- Object-XML: Service Data Objects
  - SDO 2.1.1 standardized in JSR 235
  - EclipseLink is *SDO 2.1.1 Reference Implementation*

EclipseLink Project



EclipseLink: Distributions

- Eclipse.org
  - www.eclipse.org/eclipselink/downloads
  - http://download.eclipse.org/rt/eclipselink/updates
- Oracle
  - TopLink 11g & 12c
  - WebLogic Server 11g & 12c
- GlassFish v3
  - Replaces TopLink Essentials
  - JPA 2.0 Reference Implementation
- Spring Source
  - Spring Framework and Bundle Repository
- JOnAS
- JEUS TMaxSoft

EclipseLink History & Future

- EclipseLink 1.0 - July 2008
  - JPA 1.0, simple upgrade from TopLink Essentials (JPA 1.0 RI)
- EclipseLink 1.1 - March 2009
  - JPA 1.0 with some JPA 2.0 capabilities (1.1.2 in Eclipse Galileo)
- EclipseLink 2.0 - December 2009
  - JPA 2.0 reference Implementation
- EclipseLink 2.1 (Helios) – June 2010
- EclipseLink 2.3 (Indigo) – June 2011
- EclispeLink 2.4 (Juno) – June 2012

Software Evolution

- Computing architecture is constantly evolving: Mainframe, client/server, web/thin client, mobile/apps, ...
- Current technologies with increasing adoption include:
  - Cloud computing
  - HTML 5
  - NoSQL databases
- Java EE 7 is evolving to address many of these new requirements
- EclipseLink JPA and JAXB are also evolving!

New Features

- REST—client/server over HTTP with identified resources
- Dynamic Persistence—persistence for web (JavaScript) applications
- Multitenancy—support for multiple customers in single application/server/database
- Customization—customize application instances per customer

JPA-RS

EclipseLink JPA-RS

- Provides a service that exposes JPA mapped entities over REST via JAX-RS
- HTTP message body either XML or JSON
- Client
  - HTML 5 with JavaScript (primary focus)
  - JavaFX

What is REST?

- **REST – RE**presentational **S**tate **T**ransfer
- **Priniciples:**
  - Addressable resources (URI per resource)
  - Small set of well-defined methods (i.e. GET, PUT, POST, DELETE)
  - Representation-oriented
  - Communicate statelessly

What is JAX-RS?

- Java API for RESTful Services
  - Java EE specification (Jersey is reference implementation)
- **Principles**
  - Java EE framework for implementing RESTful services
  - Provides annotations to bind combination of URI and HTTP operation to Java methods.
- **Specifications**
  - JAX-RS 1.0 (JSR 311) – Released October 2008
  - JAX-RS 2.0 (JSR 339) – In Progress

JAX-RS with JPA Example – GET Invoice

```
public class InvoiceService {...


  public Invoice read(int id) {
       return null;
  }
...
```

JAX-RS with JPA Example – GET Invoice

```
@Stateless
public class InvoiceService {...



  public Invoice read(int id) {
       return entityManager.find(Invoice.class, id);
  }
...
```

JAX-RS with JPA Example – GET Invoice

```
@Path("/invoice")
@Stateless
public class InvoiceService {...


  public Invoice read(int id) {
       return entityManager.find(Invoice.class, id);
  }
...
```

JAX-RS with JPA Example – GET Invoice

```
@Path("/invoice")
@Stateless
public class InvoiceService {...

  @GET
  @Path("{id}")
  public Invoice read(@PathParam("id") int id) {
       return entityManager.find(Invoice.class, id);
  }
...
```

JAX-RS with JPA Example – GET Invoice

```
@Path("/invoice")
@Stateless
public class InvoiceService {...
  @GET
  @Path("{id}")
  @Produces({"application/xml", "application/json"})
  public Invoice read(@PathParam("id") int id) {
       return entityManager.find(Invoice.class, id);
  }
...
```

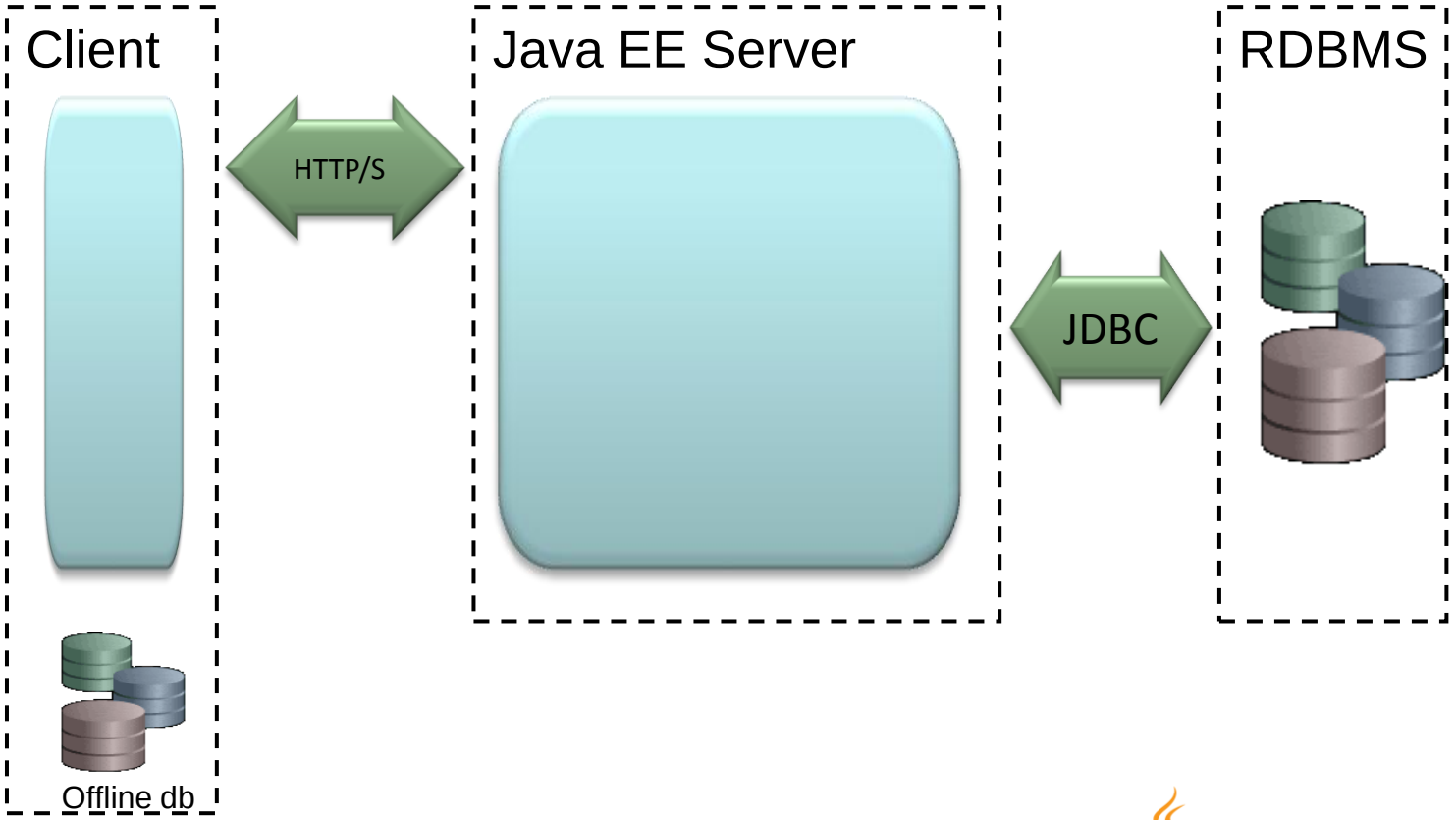
JAX-RS with JPA Example – GET Invoice

```
@Path("/invoice")
@Stateless
public class InvoiceService {...
  @GET
  @Path("{id}")
  @Produces({"application/xml", "application/json"})
  public Invoice read(@PathParam("id") int id) {
       return entityManager.find(Invoice.class, id);
  }
...
```

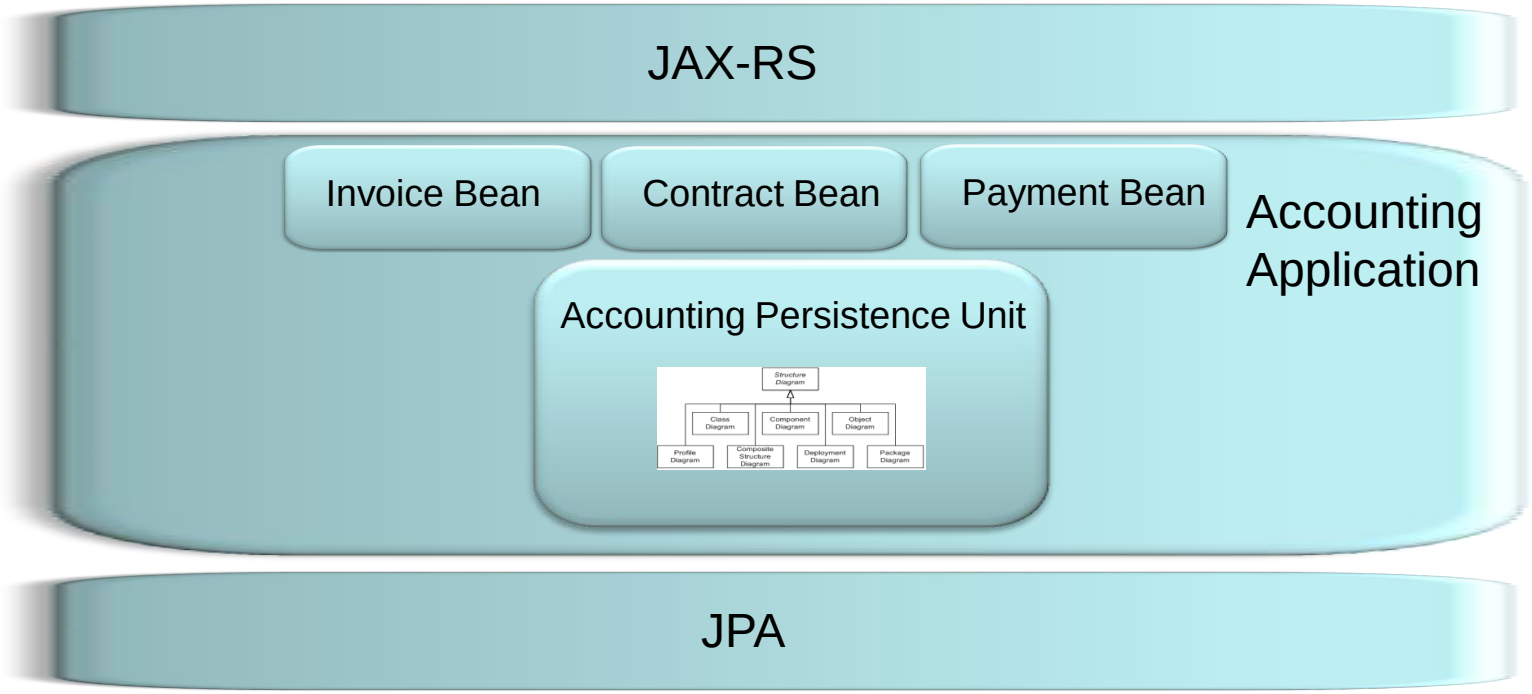
GET http://[*machine*]:[*port*]/[*web-context*]/invoice/4

Java™ ORACLE

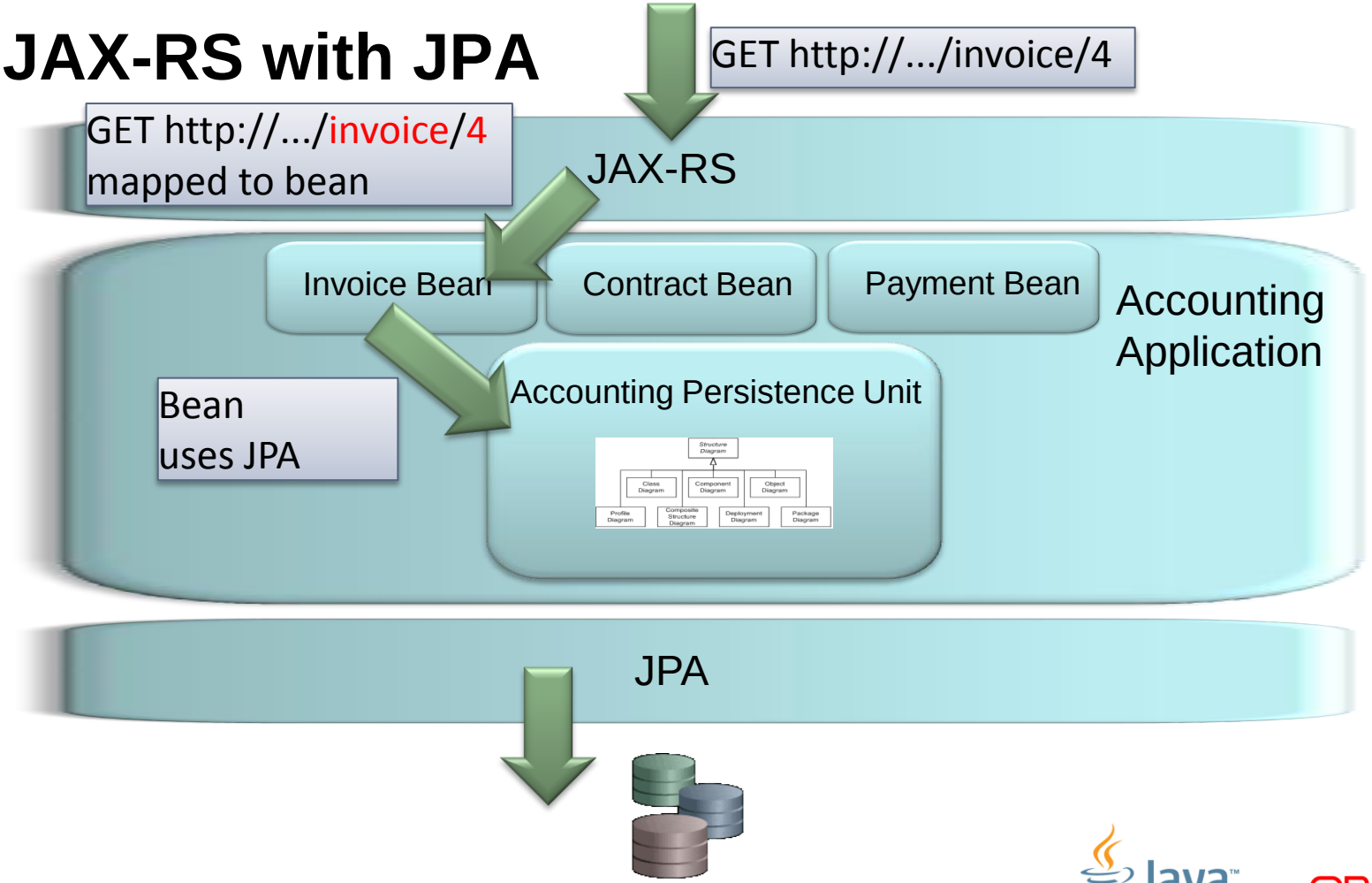
JAX-RS with JPA—High Level Architecture



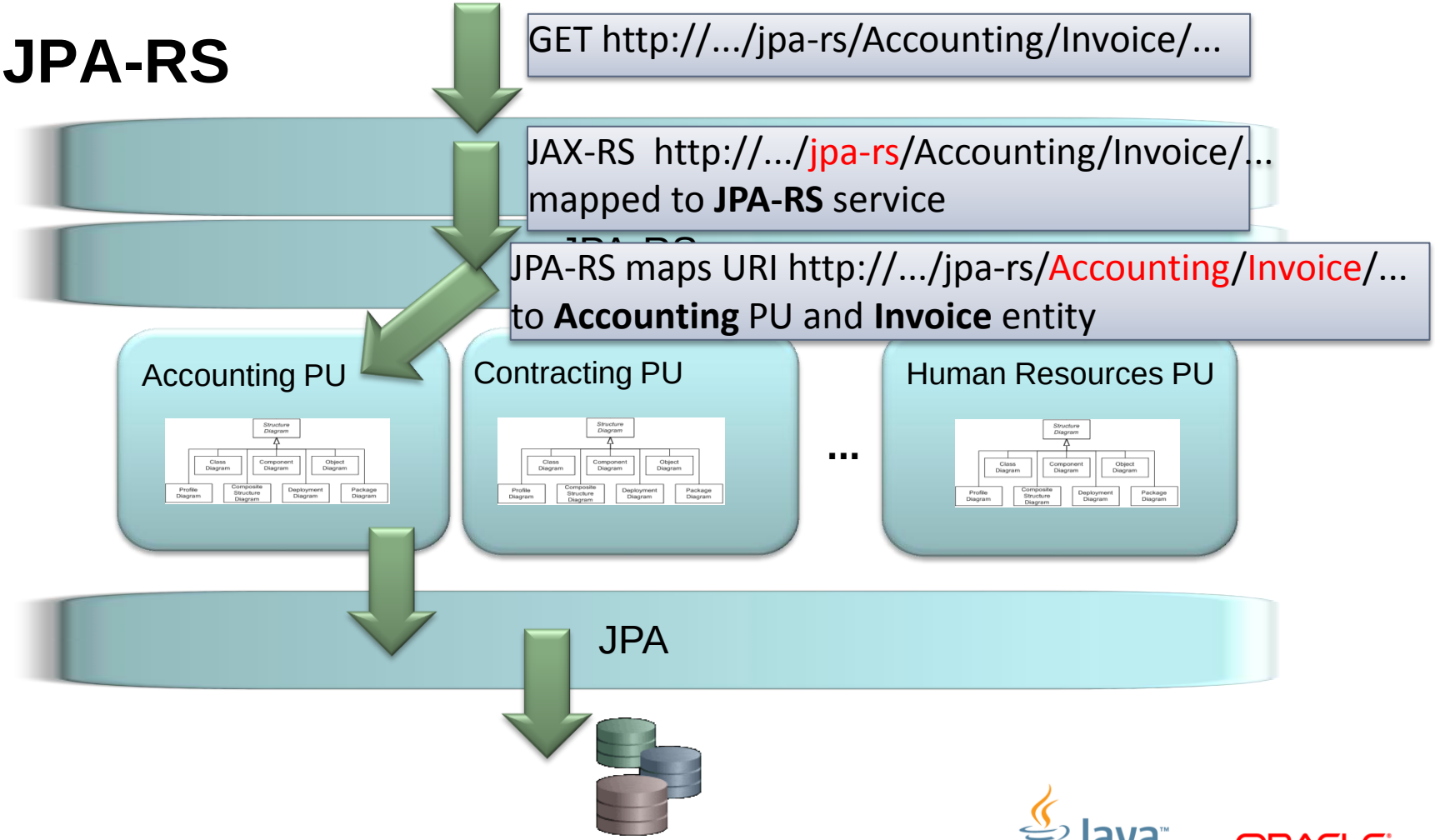
JAX-RS with JPA Example



JAX-RS with JPA



JPA-RS



JPA-RS Features

- Access relational data through REST
  - JSON or XML
- Provides REST operations for entities in persistence unit (GET, PUT, POST, DELETE)
- Supports invocation of named queries via HTTP
- Server Caching—EclipseLink clustered cache
- Client offline storage and sync
- Dynamic Persistence also supported
  - **Entities defined via metadata—no Java classes required**
  - Enables persistence services for HTML 5/JavaScript applications

JAX-RS DEMO

JPA-RS Related Technologies

- JPA-RS—Exposing JPA over RESTful HTTP Services
- Dynamic Provisioning—persistence units defined entirely with metadata—no Java classes.
- JSON Binding—Mapping Java classes/JPA entities to JSON
- REST Resource Mapping—defining mapping from Java model to REST resource model to control XML/JSON marshalling
- JAXB/JPA Fidelity—integration to permit marshalling/unmarshalling of JPA entities to XML (JSON)

DYNAMIC PROVISIONING

Dynamic Provisioning

- Persistence units defined entirely with metadata—no Java classes.
- Ideally suited to HTML 5 client applications
- Clients can dynamically define storage requirements for a set of classes (object types) and EclipseLink will instantiate a full JPA-RS CRUD service for those classes as well as JPQL query support.

DYNAMIC PROVISIONING DEMO

JSON BINDING

JSON Binding / EclipseLink “JSON-B”

- Provides Java/JSON binding similar to EclipseLink JAXB’s Java/XML binding.
- Marshall Java domain model to and from JSON
- Currently no Java standard—EclipseLink interprets JAXB XML bindings for JSON
- Content-type selectable by setting property on Marshaller/Unmarshaller

EclipseLink JSON-B Goals

- Offer the same flexibility as object-to-XML mappings
- Support both XML and JSON with one set of mappings
- No additional compile time dependencies over the JAXB APIs
- Be easy to use with JAX-RS (i.e., MessageBodyReader and MessageBodyWriter)

XML and JSON from JAXB Mappings

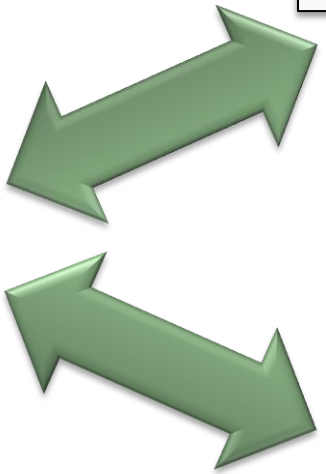
```java
@XmlRootElement(namespace="urn:example")
public class Foo {

    @XmlAttribute
    private int id;

    @XmlElement(namespace="urn:example")
    private String bar;

}
```

JAXB mapped Java



```xml
<foo xmlns="urn:examle" id="123">
   <bar>Hello World</bar>
</foo>
```

XML

```
{"foo" : {
    "id" : 123,
    "bar : "Hello World"
}}
```

JSON

Java™ ORACLE®

JSON-B DEMO

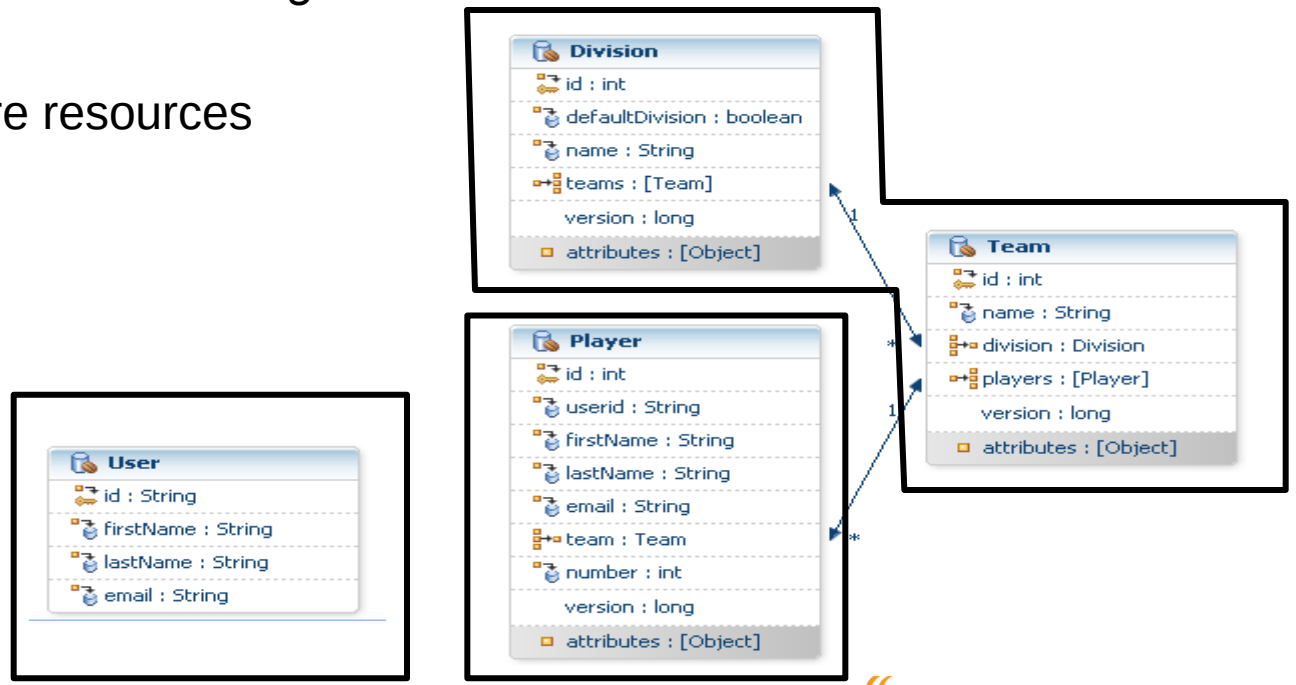
REST RESOURCE MAPPING

REST Resource Mapping

- REST requires URIs for identifiable resources
- Resources not 1:1 with classes
  - may be a graph of closely related objects
- Resources are connected via links
- Need a way to define Resource Model that can be leveraged by JAXB/JSON Binding

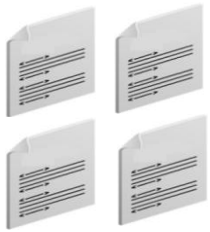
Resource Example

- Team and it's Divisions are a single resource
- User and Player are resources

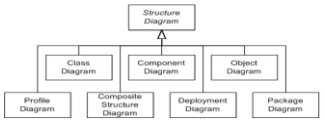


Resource Model

- Maps Java Object Model to REST Resources



Resource Model



Object Model

Java™ ORACLE®

EclipseLink Resource Model Status

- In development
- Resources (sub-graphs of domain graph) can be marshalled and unmarshalled (and reconnected)
- Links are being automatically generated
  - Currently requires use of JAXB annotations
- Future: simplify metadata declaration of resources

REST RESOURCE DEMO

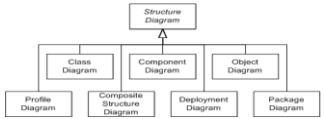
JPA/JAXB FIDELITY

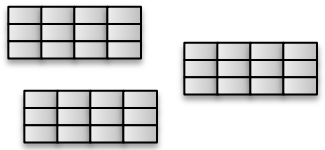
JAXB/JPA Fidelity

- JAXB and JPA specifications defined in relative isolation
- Have conflicting / differing semantics
- Enhancements required to permit marshalling/unmarshalling of JPA entities to/from XML (JSON)

Challenges – Mapping Java Objects (JPA Entities) to XML

```
<?xml version="1.0" ?>
<employee>
  <first>Mark</first>
  <last>Twain</last>
  <id>1</id>
</employee>
```

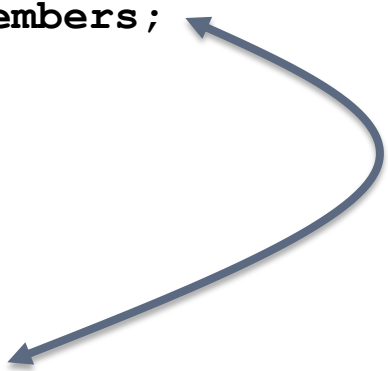




- Bidirectional/Cyclical Relationships
- Composite Keys/Embedded Key Classes
- Byte Code Weaving

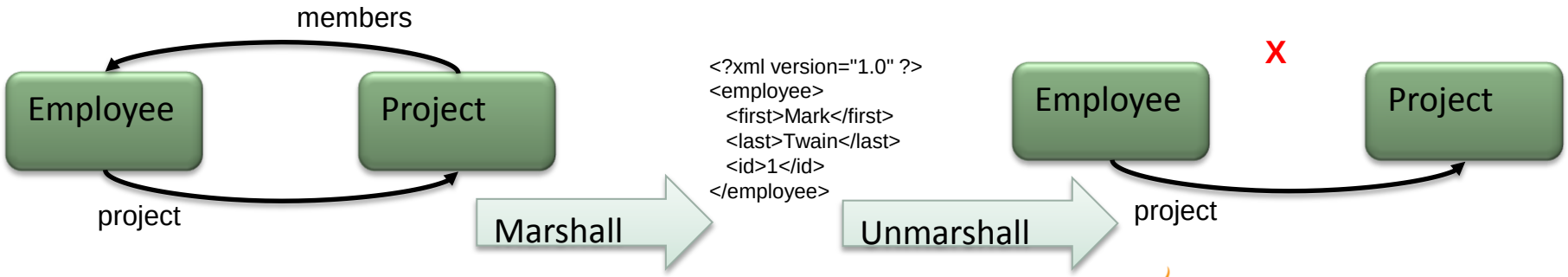
Bidirectional Relationship

```
@Entity
public class Project{
    ...
    @OneToMany(mappedBy="project")
    private List<Employee> members;
}
@Entity
public class Employee{
    ...
    @ManyToOne
    private Project project;
}
```



Bidirectional Relationships in JAXB

- JAXB specification does not support bidirectional relationships. One side must be marked **`@XmlTransient`**.
- But that loses the relationship!

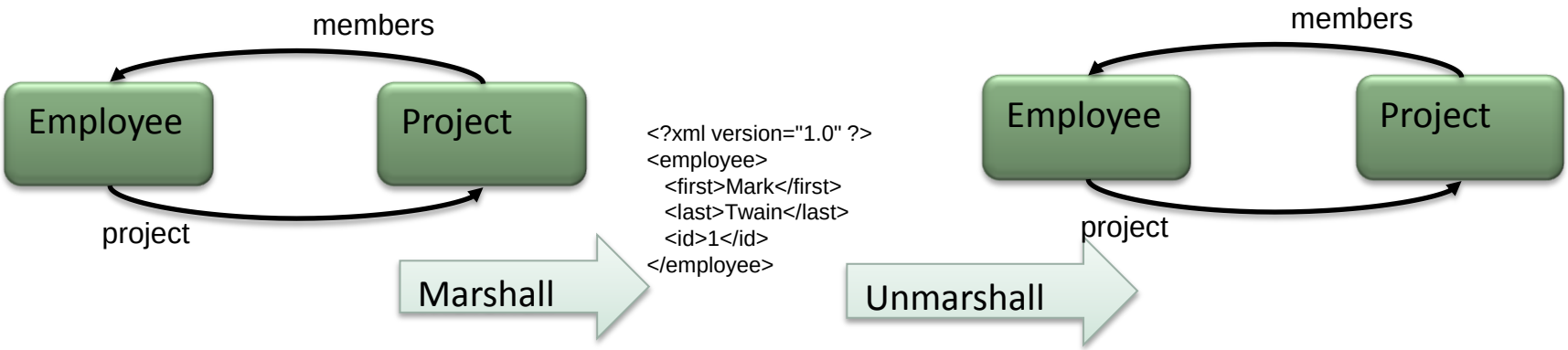


EclipseLink XmlInverseReference

```
@Entity
public class Project{
   ...
   @OneToMany(mappedBy="project")
   private List<Employee> members;
}
@Entity
public class Employee{
   ...
   @ManyToOne
   @XmlInverseReference(mappedBy="members")
   private Project project;
}
```

EclipseLink XmlInverseReference

- EclipseLink restores relationships on unmarshall!



JAXB/JPA FIDELITY DEMO

NOSQL PERSISTENCE

NoSQL Databases

- NoSQL (i.e., non-relational) database are increasingly popular
- No standards
- Differing APIs and feature sets
- Some offer query language/API—some not

EclipseLink NoSQL

- Support JPA access to NoSQL databases
  - Leverage non-relational database support for JCA (and JDBC when available)
- Define annotations and XML to identify NoSQL stored entities (e.g., @NoSQL)
- Support JPQL subset for each
  - Key principal: leverage what's available
- Initial support for MongoDB and Oracle NoSQL.
- Support mixing relational and non-relational data in single composite persistence unit

Example NoSQL Mapped Entities (not final)

```
@Entity
@NoSql(dataFormat=DataFormatType.MAPPED)
public class Order {
  @Id
  @Column(name="Id")
  public long id;
  public String orderedBy;
  @Field(name="address")
  public Address address;

  @OneToOne
  @JoinField(name="customerId",
      referencedFieldName="ID")
  public Customer customer;
  ...
}
```

```
@Embeddable
@NoSql(dataFormat=DataFormatType.MAPPED)
public class Address {
    @Field(name="addressee")
    public String addressee;
    public String street;
    ...
```

```
@Entity
@NoSql(dataFormat=DataFormatType.MAPPED)
public class Customer {
    @Id
    public String id;
    public String name;

    public String toString() {
        return "Customer(" + name + ")";
    }
}
```

MULTITENANCY

Multitenancy

- Multitenancy refers to a principle in software architecture where a single instance of the software runs on a server, serving multiple client organizations (tenants).
- Multitenancy is contrasted with a multi-instance architecture where separate software instances (or hardware systems) are set up for different client organizations.

- Wikipedia http://en.wikipedia.org/wiki/Multitenancy

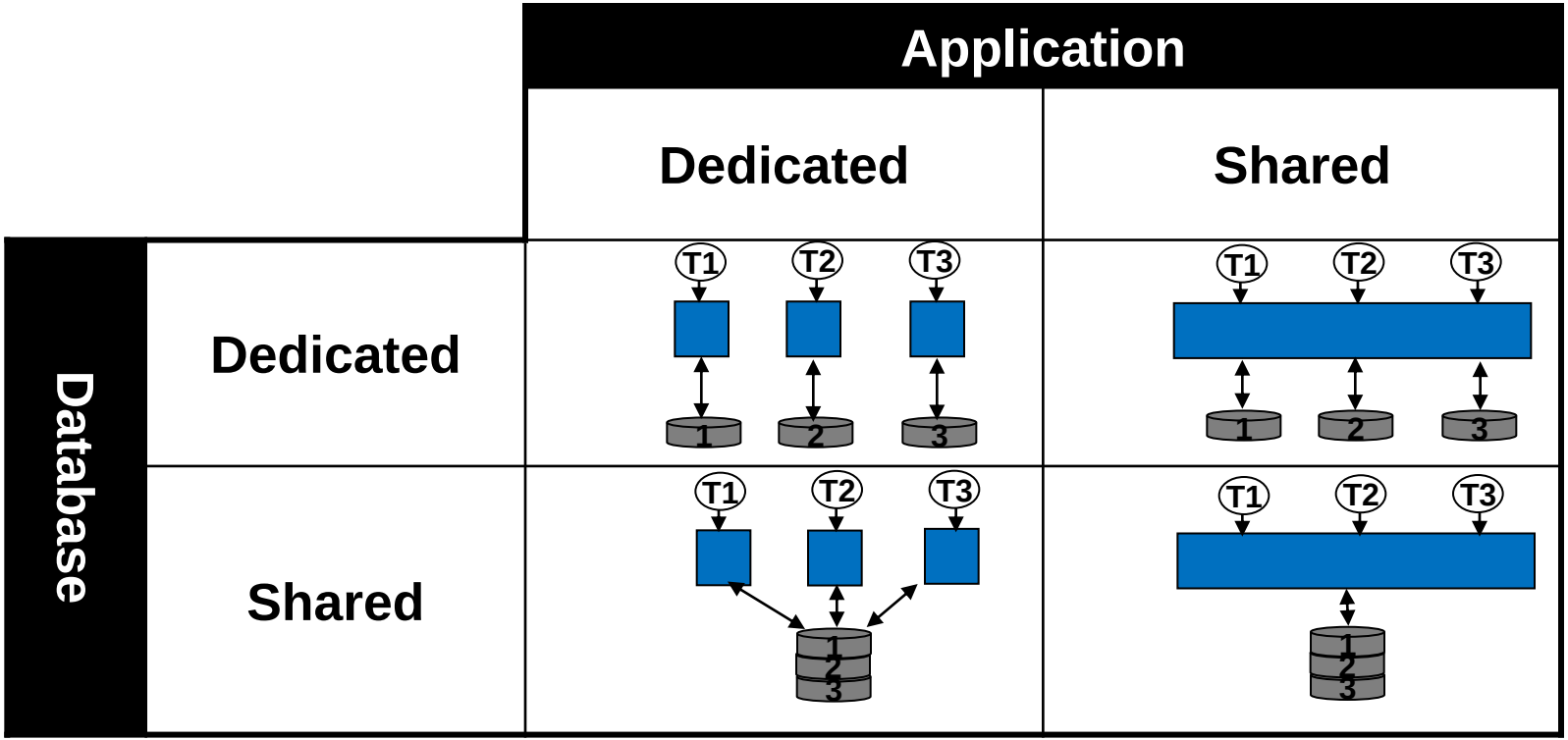
Application Development and the Cloud

- Today
  - Single Tenant or non-Tenant Applications
  - Dedicated application instance and database
- Future
  - Support multiple tenants
  - Support extensibility (custom fields per tenant)
  - Support various deployment architectures
    - Dedicated or shared application instances
    - Dedicated or shared databases

So Many Clouds

- Infrastructure - IaaS
  - E.g., Amazon Web Services
- Platform – PaaS
  - E.g., Oracle Public Cloud, Cloud Bees, Google App Engine
- Software – SaaS
  - E.g., Google Mail

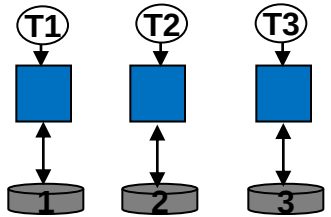
Multitenant Topologies



Note: Single application deployed to support various MT architectures

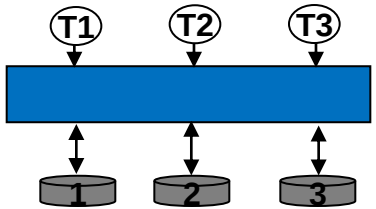
Multitenant: Dedicated Application Dedicated DB

- Dedicated application instance
  - Application instance per tenant
    - unique container or application class-loader
  - Caching supported
- Dedicated database
  - Unique tables (tablespace/schema/db) per tenant
  - Tenant specific data source required



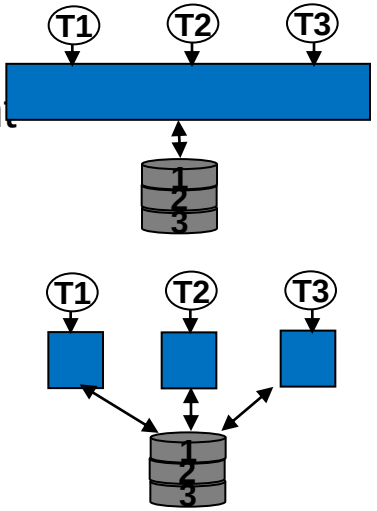
Multitenant: Shared App Dedicated DB

- Shared Application Instance
  - Application instances handle multiple tenants
  - Caching must isolate by tenant
- Dedicated Database
  - Common data source
    - Unique schema/tablespace per tenant
    - Common schema with table per tenant (partitioning)
    - Proxy Authentication
  - Data source per tenant



Shared Database

- **@Multitenant**
  - Application's persistence layer manages access
  - Row data includes tenant identifier values
  - Queries augmented to limit results based on current tenant
  - Database vendor independent
- **@Multitenant(VPD)**
  - Row data includes tenant identifier values
  - Database provides client limited view of database tables
    - Shared solution for all database clients
    - Native queries (SQL) supported



Multitenant Entity Strategies

- GOAL: support storage of entities from multiple tenants in a single shared schema
- @Multitenant Strategies
  - @Multitenant(SINGLE_TABLE) - default
  - @Multitenant(VPD)
    - SINGLE_TABLE + includeCriteria=false
    - SET_IDENTIFIER(property) & CLEAR_IDENTIFIER
    - DDL Gen of predicate function and ADD_POLICY
  - Future: @Multitenant(TABLE_PER_TENANT)

In the beginning…

- Application dedicated for single tenant
- All rows available to all queries

```
@Entity
public class Player {
```

PLAYER

ID	VERSION	F_NAME	L_NAME	LEAGUE
1	1	John	Doe	HTHL
2	3	Jane	Doe	OSL

DEMO—JPA SINGLE (NO) TENANCY

Multitenant: SINGLE_TABLE

- Simple configuration: Annotation or XML
- Flexible tenant identifier support
- EclipseLink augments generated SQL

```
@Entity
@Multitenant
@TenantDiscriminatorColumn(name="league-id", columnName="LEAGUE")
public class Player {
```

PLAYER

ID	VERSION	F_NAME	L_NAME	LEAGUE
1	1	John	Doe	HTHL
2	3	Jane	Doe	OSL

DEMO—SINGLE TABLE MULTITENANCY

Multitenant using Oracle VPD

- Leverage the Oracle Database

```
@Entity
@Multitenant(VPD)
@TenantDiscriminatorColumn(name="league-id", columnName="LEAGUE")
public class Player {
```

PLAYER

ID	VERSION	F_NAME	L_NAME	LEAGUE
1	1	John	Doe	HTHL
2	3	Jane	Doe	OSL

Multitenant: TENANT_PER_TABLE

- Planned Feature

```
@Entity
@Multitenant(TABLE_PER_TENANT)
public class Player {
```

HTHL.PLAYER

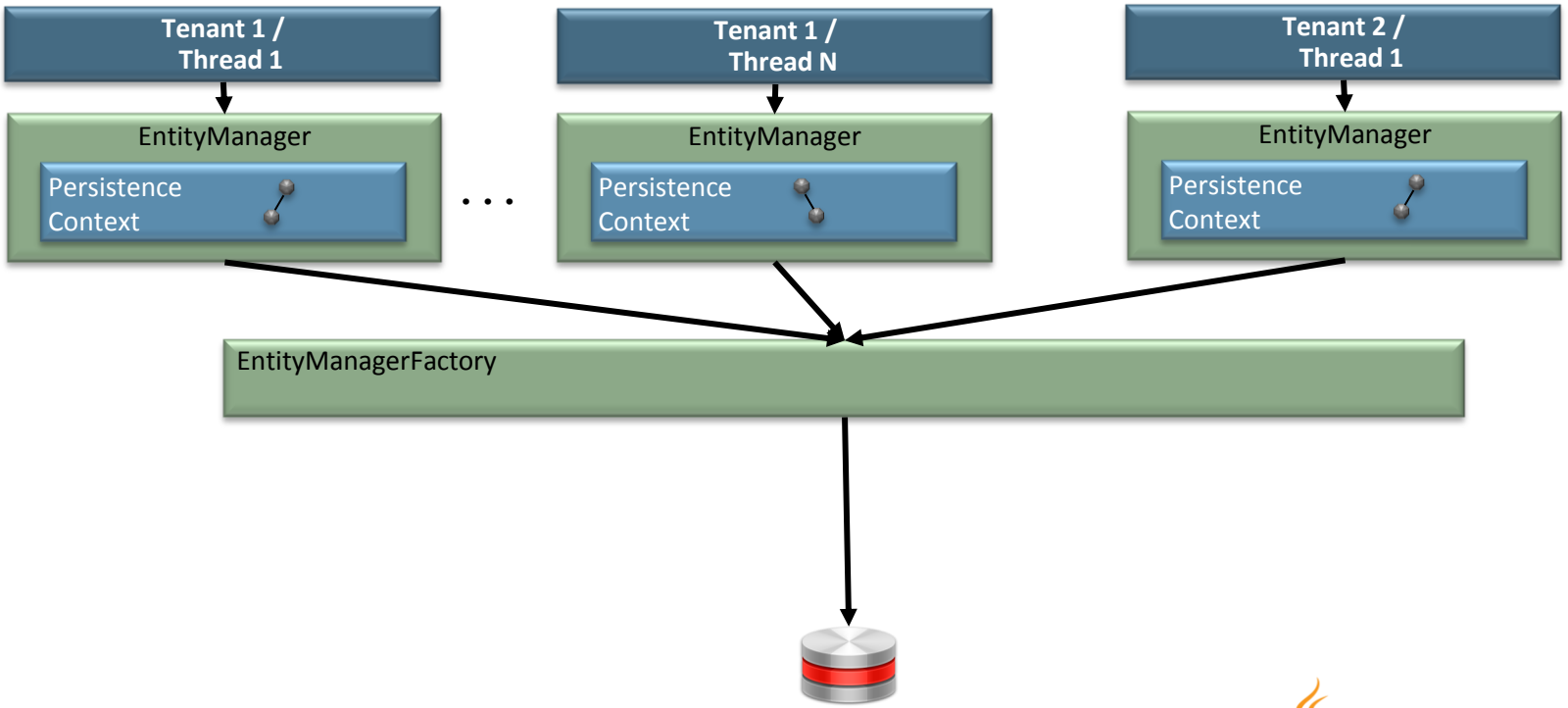
ID	VERSION	F_NAME	L_NAME
1	1	John	Doe

OSL.PLAYER

ID	VERSION	F_NAME	L_NAME
2	3	Jane	Doe

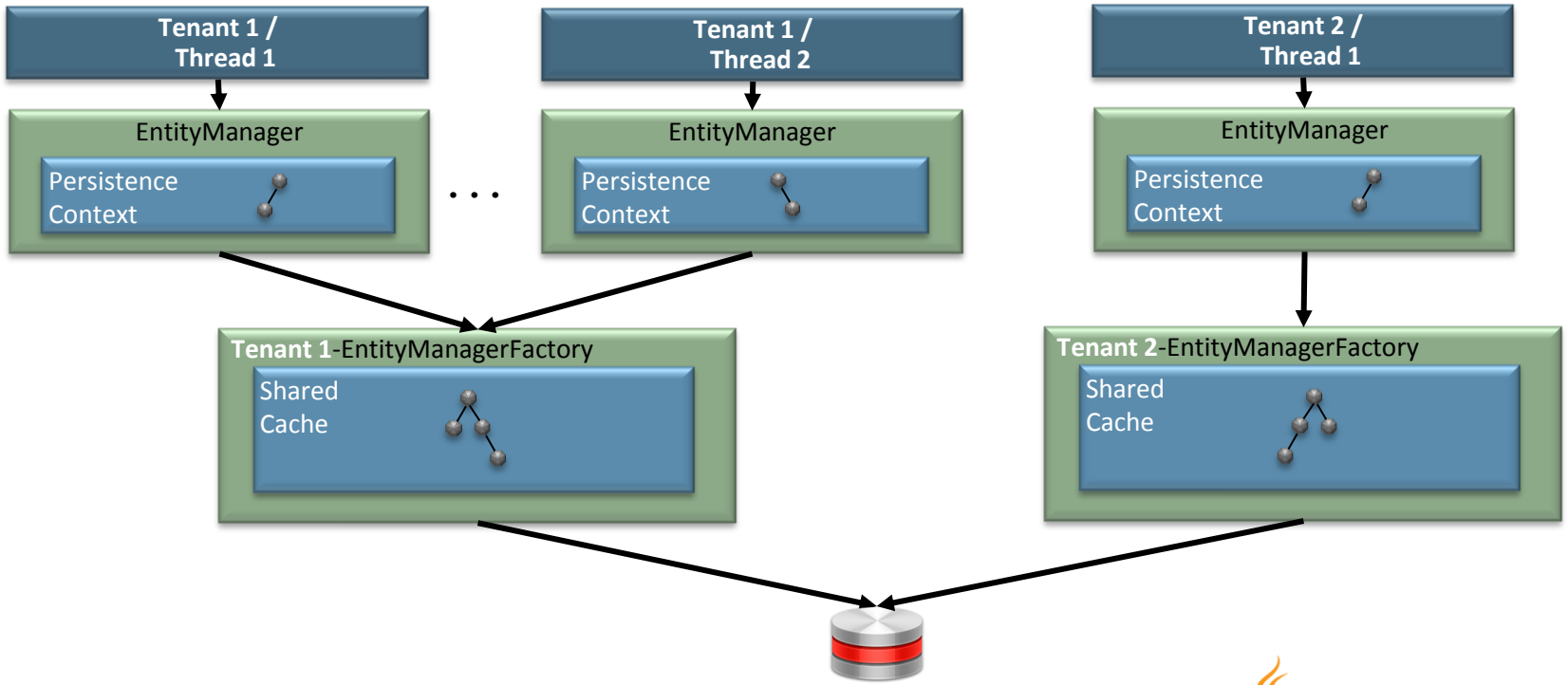
Caching & Multitenancy

- EntityManager/Tenant—Shared Cache Disabled



Caching & Multitenancy

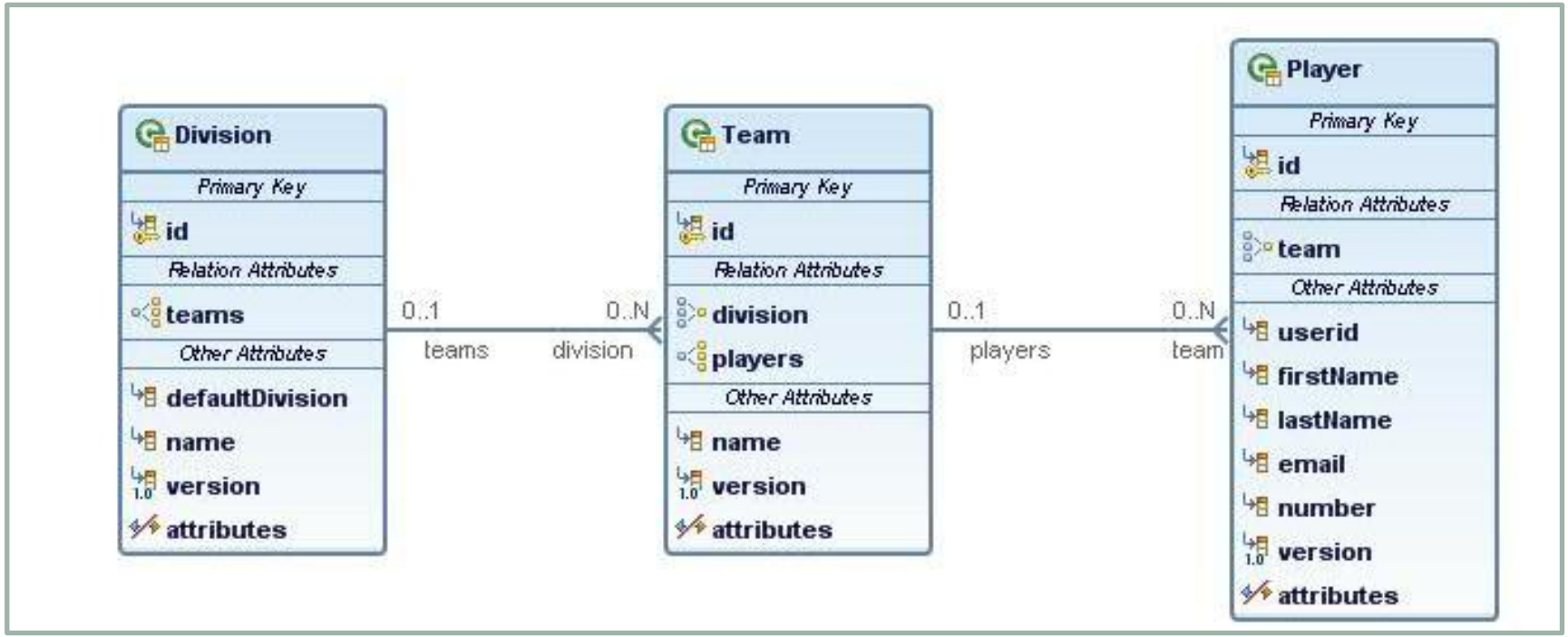
- EntityManagerFactory/Tenant—Shared Cache



MySports Demo

- Introduced in EclipseLink Indigo (2.3)
- Features
  - @Multitenant
    - EntityManagerFactory per tenant (shared cache enabled)
  - @VirtualAccessMethods (Extensions per Tenant)
  - External Metadata Sources
  - JSF, EJB, JPA
  - Admin: JSF + JAX-RS + JPA
- Wiki
  - http://wiki.eclipse.org/EclipseLink/Examples/MySports

MySports Demo Model

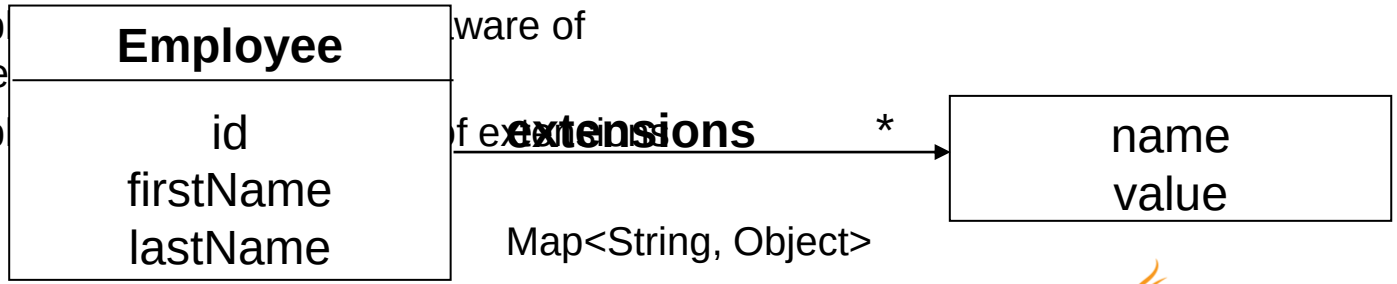


DEMO—MYSPORTS MULTITENANCY

DOMAIN MODEL EXTENSIONS

Domain Model Extensions

- Storage and querying of extended properties
  - Application developer enables extensions in entity
  - Schema created with extension columns/table(s)
  - Application Admin stores extension definitions
  - App ware of exte
  - App f extensions



Flex Extensions

```
@VirtualAccessMethods
public class Player{
…
@Transient
private Map<String, Object> attributes;

public <T> T get(String attributeName) {
        return (T) this.attributes.get(attributeName);
}
public Object set(String attributeName, Object value) {
        return this.attributes.put(attributeName, value);
}
```

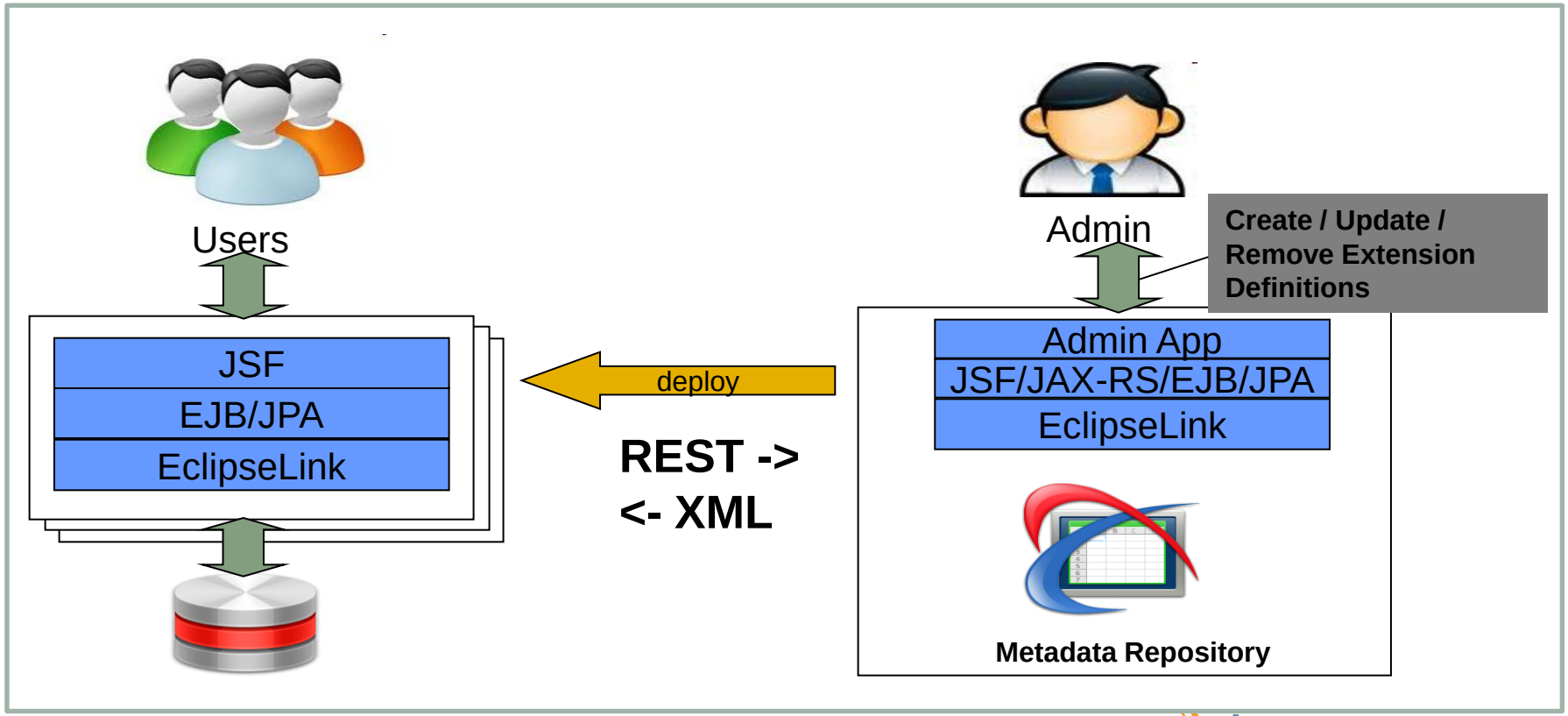
PLAYER

ID	F_NAME	L_NAME	FLEX_1	FLEX_2
1	John	Doe	‘R’	’22’
2	Jane	Smith	‘NONE’	

Virtual Access Mappings

```
<entity class="example.mysports.model.Player">
    <attributes>
        <basic name="penaltyMinutes" access="VIRTUAL"
               attribute-type="java.lang.Integer">
            <column name="flex_1"/>
        </basic>
        <basic name="position" access="VIRTUAL"
               attribute-type="java.lang.String">
            <column name="flex_2"/>
        </basic>
    </attributes>
</entity>
```

MySports Architecture



 |

DEMO—MYSPORTS EXTENSIBLE ENTITIES

Summary

- Java is evolving—and EclipseLink is evolving too!
  - JPA-RS
  - JSON Binding
  - REST Resource Mapping
  - Dynamic Provisioning
  - NoSQL
  - Multitenancy
  - Extensible entities
- EclipseLink is *the* center of innovation in Java persistence

Q & A

Provide Feedback, Get Involved!
User forums and lists at http://eclipse.org/eclipselink


Java™