





**MOVING JAVA  
FORWARD**

**ORACLE®**

Java Persistence API on the Grid



**Latin America 2011**

December 6–8, 2011

**Tokyo 2012**

April 4–6, 2012

# JavaOne Bookstore

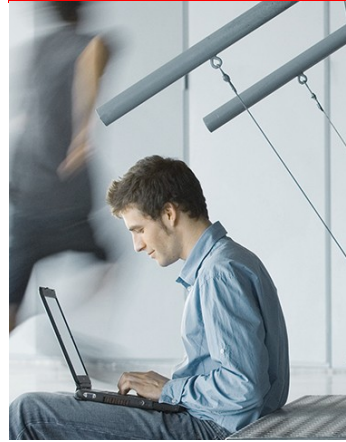
- **Visit the Store @ the Zone for a fabulous selection of books on many of the conference topics and more!**
- **Store located at Hilton Hotel, Ballroom Level**
- **All Books at 20% Discount**

**DigitalGuru**  
Technical Bookshop



# Agenda

- Introduction
- JPA and Data Grids
- Integration Architectures
- Implementations
- Challenges
- Conclusion



# Java Persistence API

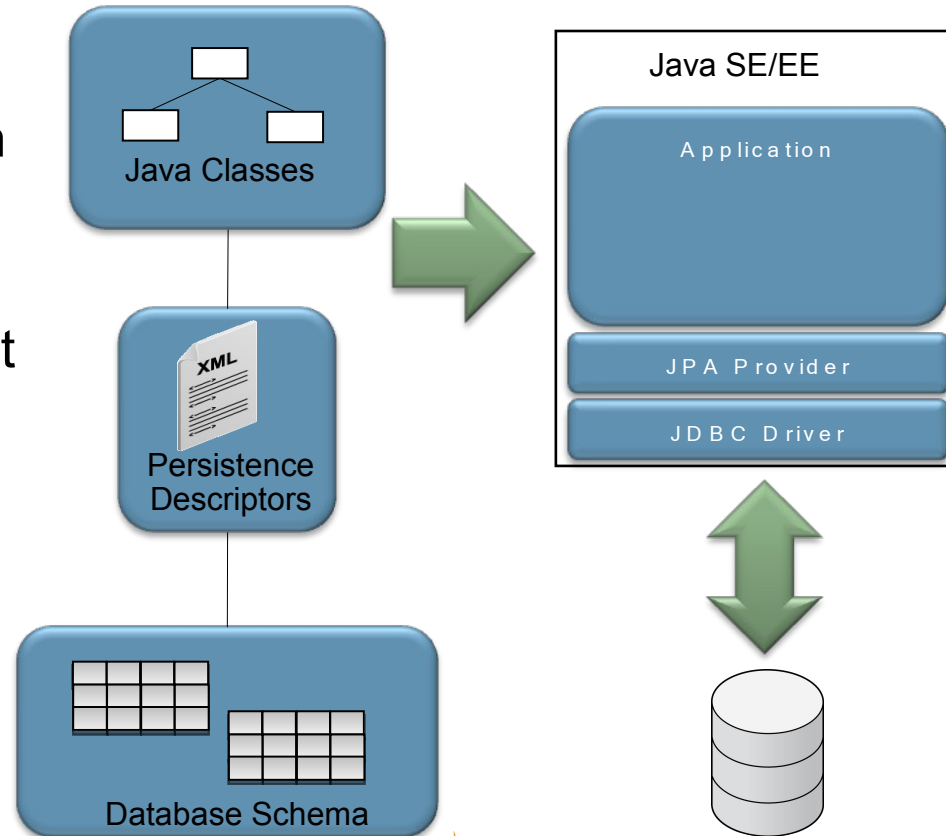
## Introduction

- A standardization of industry practices for Java POJO Object Relational Persistence
- Suitable for use in different modes
  - Standalone in Java SE environment
  - Hosted within a Java EE Container
- Merging of expertise from persistence vendors and communities including: TopLink, Hibernate, JDO, EJB vendors and individuals

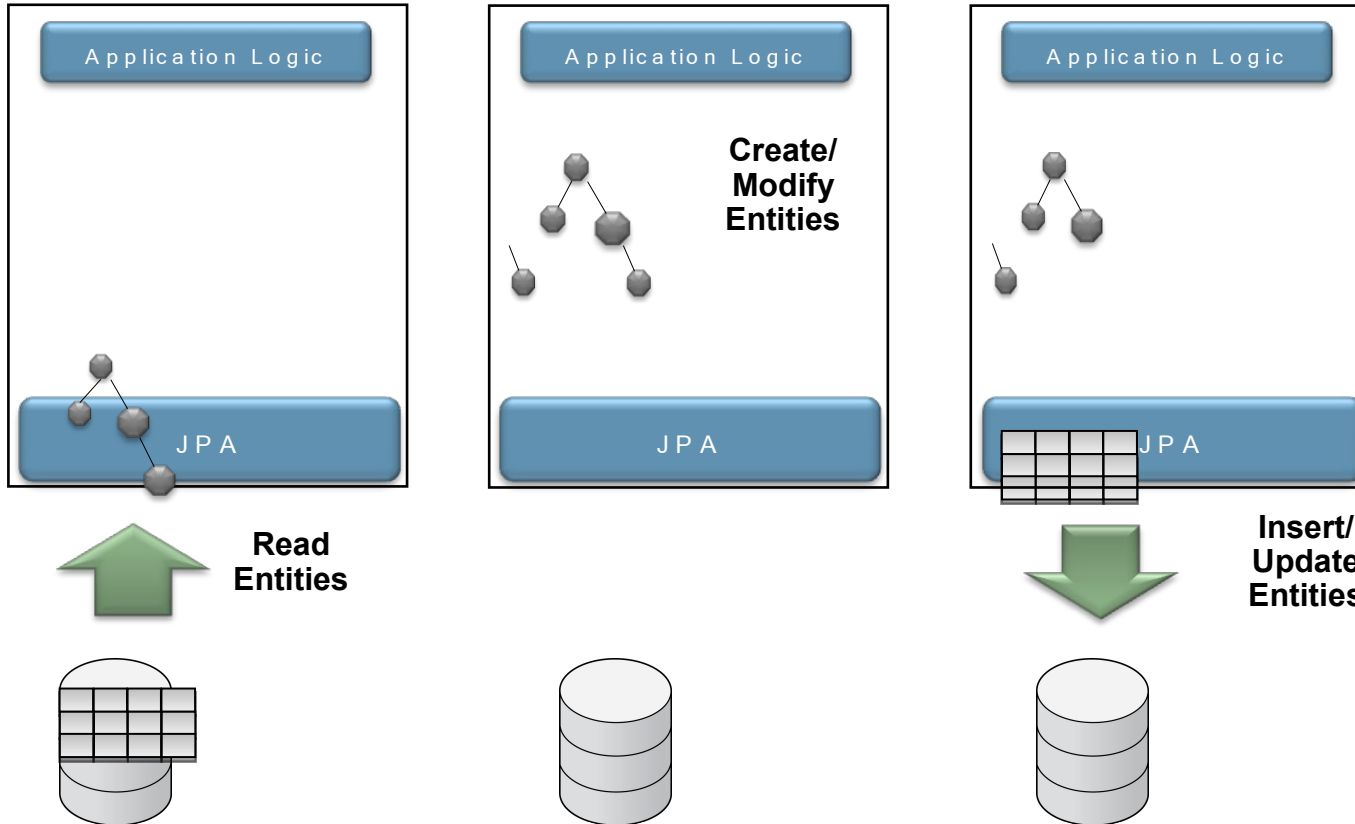
# Java Persistence API

Defines...

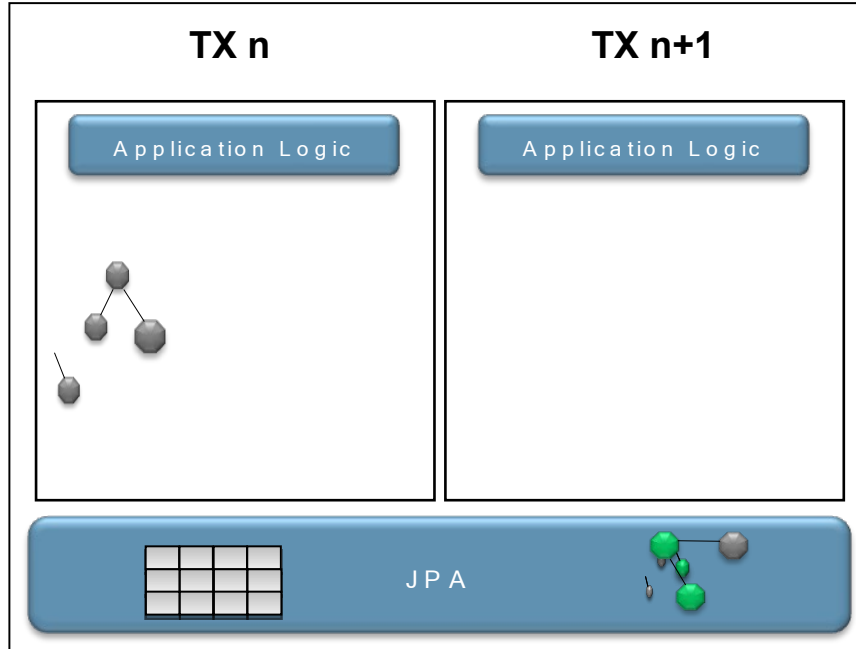
- How Java objects are stored in relational database
- A programmer API for reading, writing, and querying persistent Java objects (“Entities”)
- A full featured query language in JP QL
- A container contract that supports plugging any JPA runtime in to any compliant Java EE container



# Mechanics of a JPA Application



# JPA with Cache



Cache hits avoid object build cost

# Data Grids

## Overview

- Software that aggregates the storage and processing power of a set of servers
- Provide APIs that shields developers from complexity of distributed computing
- Typically provide ad hoc query support
- Typically provide in place processing

# JPA on the Grid

## Motivation

- Java developers are moving to the cloud and need the ability to scale out applications
- Data grids offer a way to scale out applications that rely on large amounts of data
- Integrating JPA with data grid technology promises a way to scale out applications that don't bottleneck on the database

# Read Intensive Application Example

## Bank Transaction Validation

- Process incoming transactions, e.g.: deposit, withdrawal
- Validate account, owner, sufficient funds, etc.
  - Requires querying database for many associated objects
- Implementing with JPA straightforward:
  - Map all relevant domain classes
  - Write validation logic
  - Write required JPQL to retrieve associated objects
- Validation may load large amounts of data—but it works!



# Read Intensive Application Example

## Improving Performance with Cache

- Performance bottleneck is querying database for associated objects required for validation
- Placing validation data in cache will improve performance
  - But cache size is limited by heap space
  - Random data access pattern can mean no cache hits

# Read Intensive Application Example

## Improving Performance with Data Grid

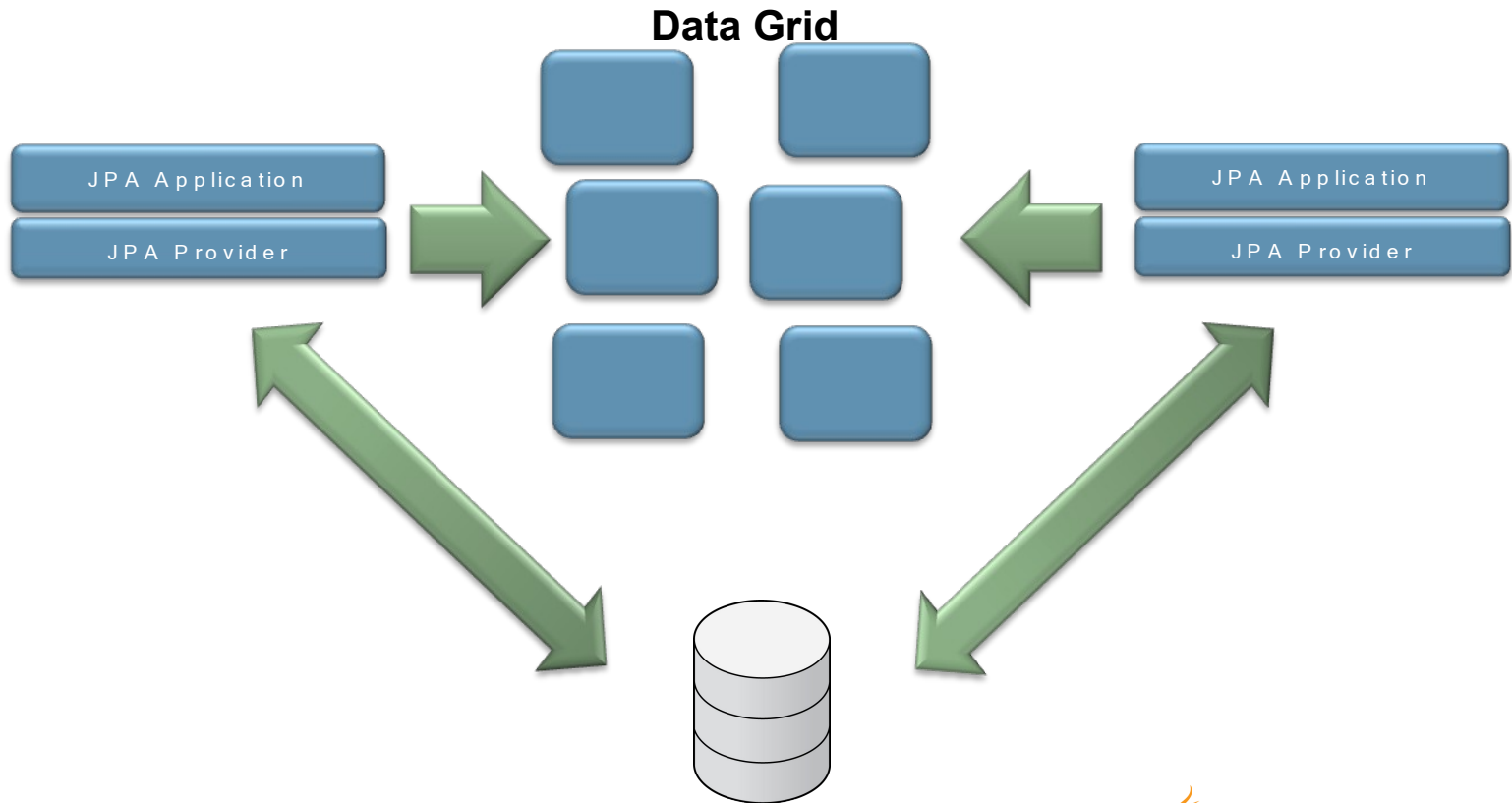
- By aggregating cluster heap shared cache size can be very large—big enough for all validation data
- Pre-loading data grid with validation data will ensure cache hit

# INTEGRATION ARCHITECTURES

# Integration Architectures

- Object Cache
- System of Record

# Object Cache



# Object Cache

## Bank Transaction Validation Example

- Consider a simple validation method in our banking system:

```
public boolean isValidAccount(Request request) {
    Account account = entityManager
        .find(Account.class, request.getAccountId());
    if (account == null) {
        return false;
    } else {
        return account.isValid();
    }
}
```

# Object Cache

## Non PK Queries

- Primary key finds are easily directed to the data grid but what about JP QL queries?

```
public Customer getTxCustomer(Request request) ... {
    Customer customer = entityManager
        .createQuery(
            "select c from Customer c " +
            "where c.masterAccountId = :id",
            Customer.class)
        .setParameter("id", request.getMasterAccountId())
        .getSingleResult();
    return customer;
}
```



# Object Cache

## Non PK Queries

- To query the data grid for an object that matches an arbitrary JPQL criterion requires:
  - The data grid to provide a query framework
  - JPA/data grid integration that can translate from JP QL into the framework

# Object Cache

## Query Parallelization

- With objects stored across data grid cluster members parallel query processing is possible
- For the same data set, parallel query execution time should improve as the number of members in the data grid increases.
- Consider processing a query on 100,000 objects on one node vs. that same query on 10,000 objects on 10 nodes

# Object Cache

## Query Results

- When querying the grid results will be limited to what's loaded.
- There are three ways to deal with this problem:
  - Ensure the grid is fully populated so results are complete
  - Direct queries to the database when grid contents are incomplete
    - Could be achieved via a query hint, e.g. `force-db-query=true`
  - Query the grid and evaluate whether results are sufficient
    - “sufficient” is application specific

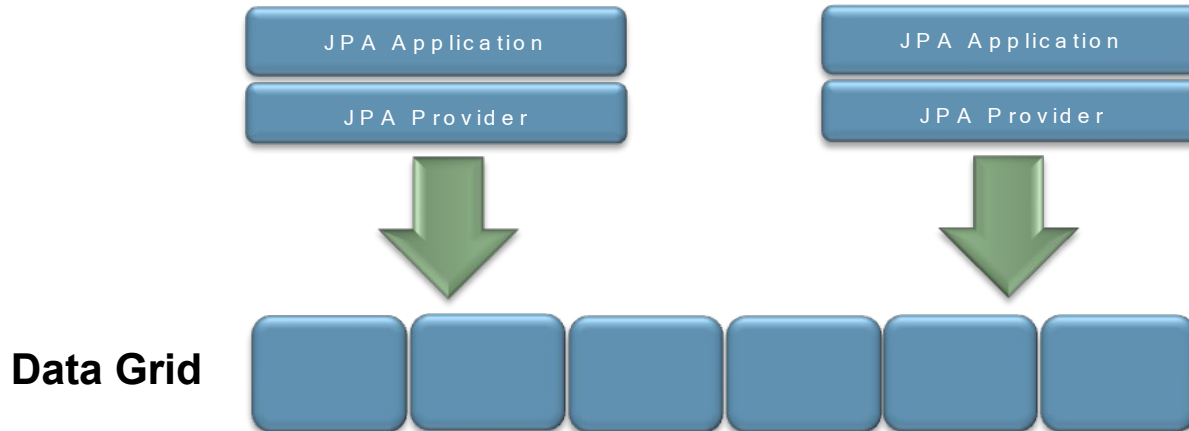


# Object Cache

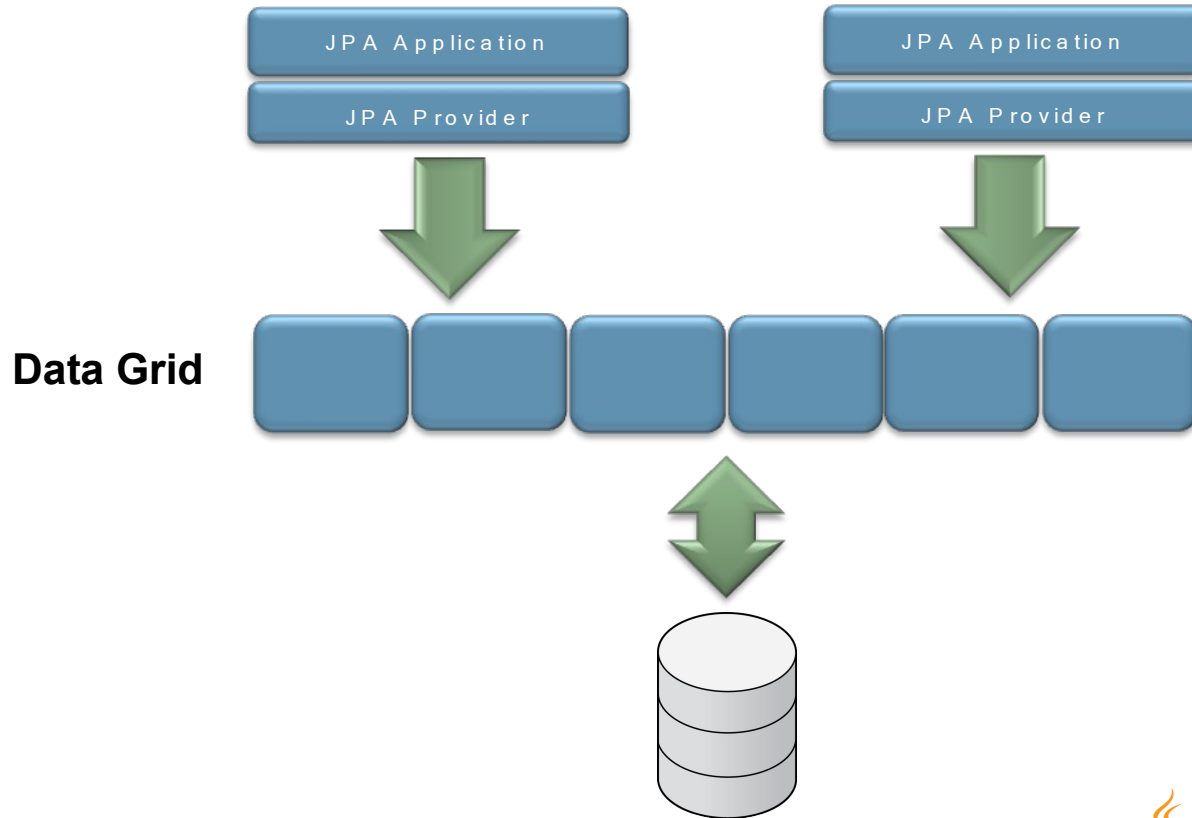
## Updates

- Beyond queries, JPA also supports creation, modification, and deletion of objects
- When using the grid as an object cache new/change/deletions must be propagated to the grid
  - Upon a successful database commit

# System of Record



# System of Record with Backing Database



# Integration Architectures

## Object Cache vs. System of Record

- Difference boils down to whether database is updated before grid or just the grid
  - True for insert, update, and delete
- If JPA/data grid integration implementation supports fine grained configuration then it should be possible to mix “database entities” with “grid entities” in the same application

# Object Storage Format

## Serialization vs. Tuples

- Storing JPA entities in a data grid requires either serialization or conversion to a serializable record/tuple
- Use of tuples in the grid incurs JPA meta data access and object construction cost on each (cache hit) retrieval

# Challenges

## Cache Staleness

- Shared grid cache ensure all clients see each other's changes
- Third party updates remain an issue—as in any application employing caching
- Standard JPA cache staleness techniques still apply
  - Eviction policies
  - Refresh policies
  - Database change notification (e.g., Oracle GoldenGate)

# Challenges

## Querying

- JPQL is the standard object-oriented query language suitable for translation to SQL
- Most data grid products provide proprietary query frameworks
- Ability to translate JPQL to a given data grid query framework will depend on the expressiveness of that framework

# Challenges

## Relationships/Partitioning

- Entity relationships typically encoded by JPA integration layer
- Issues include:
  - Types of relationships supported: OneToOne, OneToMany, Embedded, etc.
  - Location of related objects—same partition or different
  - Impact of location on query ability

# Implementations

## IBM WebSphere eXtreme Scale

- Focus on transaction processing and caching—not scaling JPA
- Provides proprietary JPA-like API
  - JPQL-like Object Grid Query Language (OGQL)
- Supports both Object Cache and System of Record architectures

# Implementations

## Hibernate/Various

- Focus on object cache architecture
- Pluggable 3rd party cache interface
  - Oracle Coherence, Ehcache, Infinispan implementations
- PK access only—grid cannot be queried with JPQL
- JPQL queries evaluated on database only
- Entities converted to tuples and serialized

# Implementations

## Hibernate OGM/Infinispan

- Hibernate fork focused on JPA for NoSQL databases
  - Currently targeting Infinispan
  - System of Record architecture
- JPQL translation to Lucene queries—not parallelized
  - Lucene indexes maintained and replicated on all grid members
- Persistence unit level configuration
- Entities and relationships converted to tuples in grid
- Significantly restricted JPA feature set

# Implementations

## Oracle TopLink Grid/Oracle Coherence

- Focus on supporting scale out of JPA applications
  - Object Cache Architecture
  - System of Record Architecture
- JPQL execution on database or grid
  - JPQL translation to Coherence filter framework—parallel query
- Entity level configuration: cached/not cached/grid cached, database entity/grid entity
- Serialized object format (high performance POF support)

# Conclusions

- Data grids offer a way to scale JPA applications
  - Quick access to in-memory objects
  - Large in-memory cache capacity through heap aggregation
  - Query processing in grid can reduce database load
- Support for JPA on the Grid features varies greatly
  - Need to evaluate whether specific product meets requirements
- Available products are young (some still in development) but provide functionality that can be leveraged today

# Q&A

