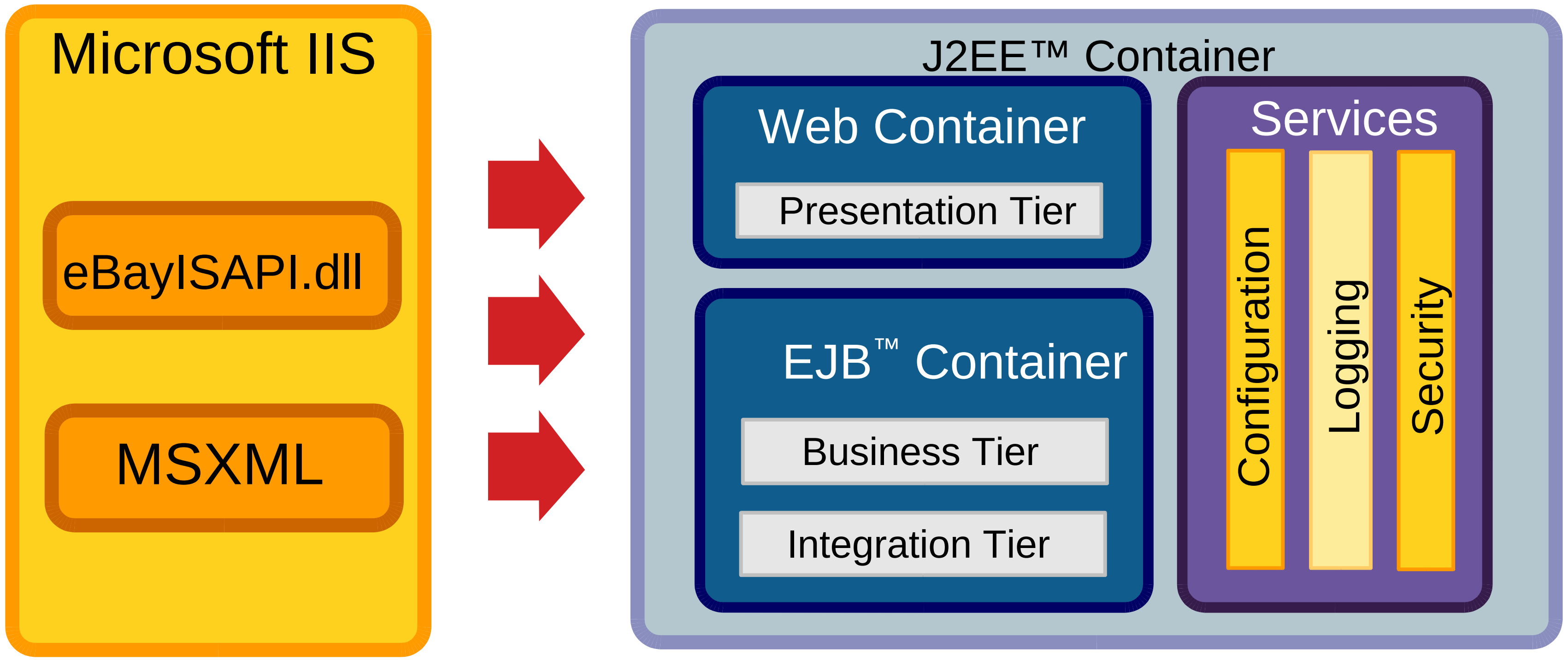
J2EE Best Practices using Real-life Examples.

Carol McDonald
Technology Evangelist
carol.mcdonald@sun.com

Sun™ Tech Days

eBay Architecture: How to Go…

From there… … to here?



- Monolithic
- Proprietary

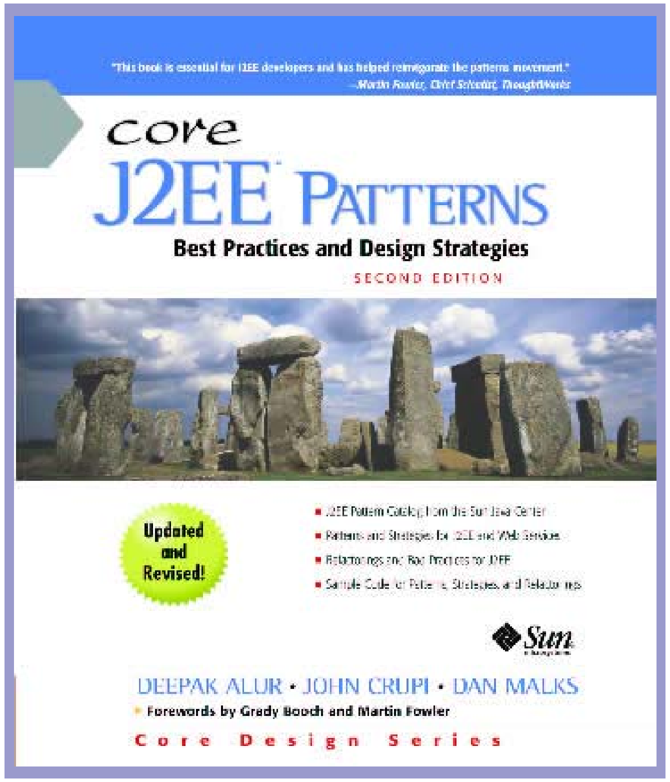
- Layered
- Loosely coupled
- Modular
- Standards based

What Is Needed…

- Learning the technology is not enough
- Industry best practices
- Proven solutions
- How to avoid bad practices?
- Experience and expertise

Core J2EE™ Patterns

- Platform Patterns for J2EE™ architecture
- Based on our experience
- Collection of best practices for J2EE™ platform
- J2EE™ architecture refactorings and bad practices
- Create a common vocabulary
- Proven solutions
  - Reuse design
  - Robust and scalable architecture
  - Improve quality
  - Flexibility and maintainability



"This book is essential for J2EE developers and has helped reinvigorate the patterns movement."
—Martin Fowler, Chief Scientist, ThoughtWorks

core J2EE PATTERNS
Best Practices and Design Strategies
SECOND EDITION

Updated and Revised!

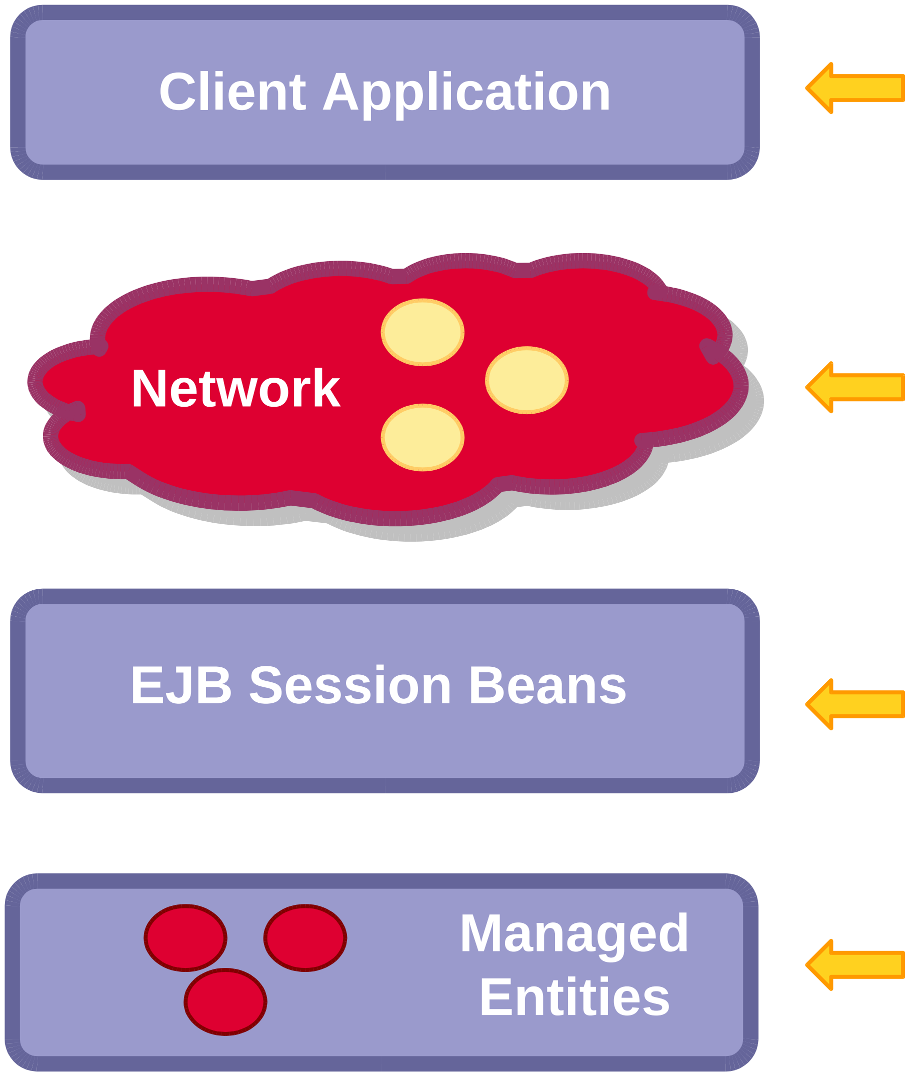
- J2EE Pattern Catalog from the Sun Java Center
- Patterns and Strategies for J2EE and Web Services
- Refactorings and Bad Practices for J2EE
- Sample Code for Patterns, Strategies, and Refactorings

DEEPAK ALUR • JOHN CRUPI • DAN MALKS
Forewords by Grady Booch and Martin Fowler
Core Design Series

Push your development further

Some Web Tier Best Practices and Patterns

Layering



Presentation Layer

Data Transfer Objects
i.e. Managed Entity Values

Business Logic Layer
Service Layer

Domain Layer
Managed Entities

Rationale for Layers

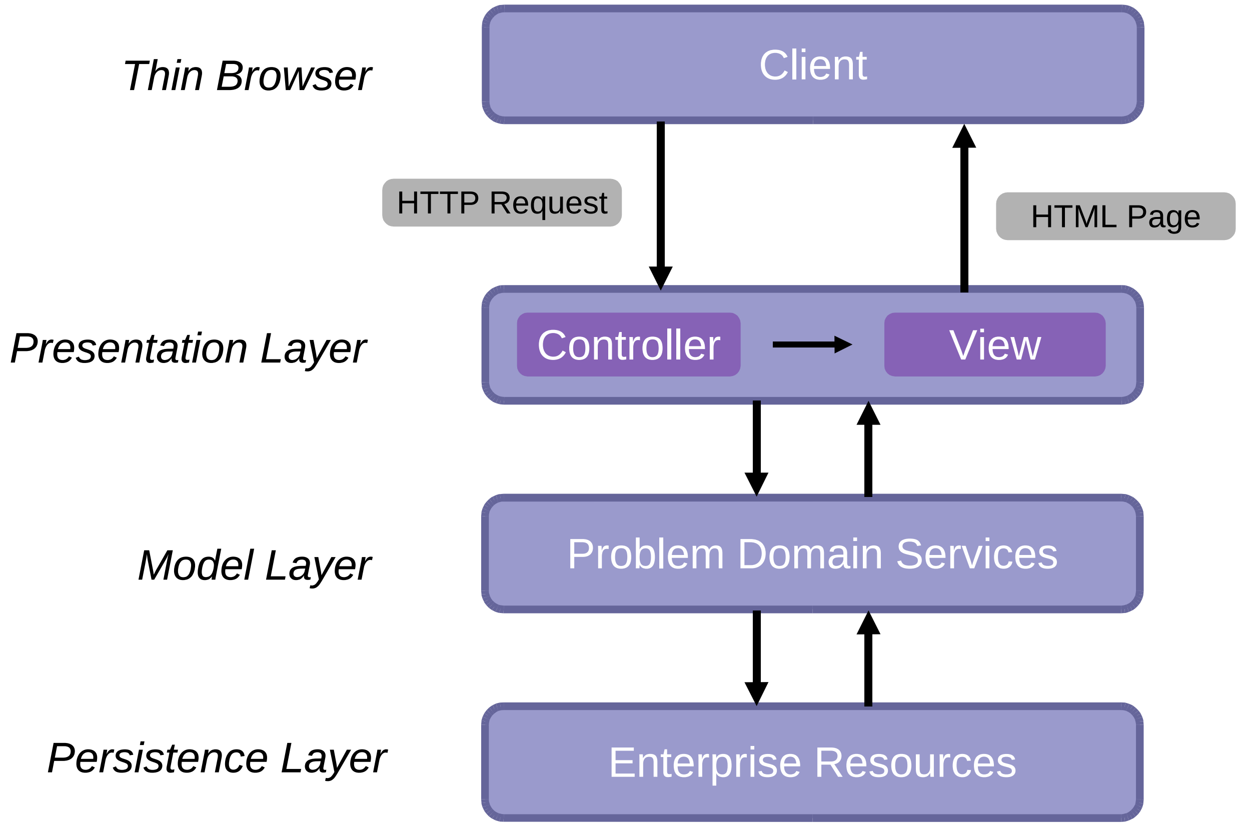
- More flexible, extensible:
  - Presentation Layers
    - Frequent changes, designed to be flexible
  - Business Services
    - Implement “use-cases”
    - Public interface, resource access
    - Should remain valid for changes in presentation and data store
  - Data Layers
    - Fewer changes, designed to optimize data access

Layered View (cont.)

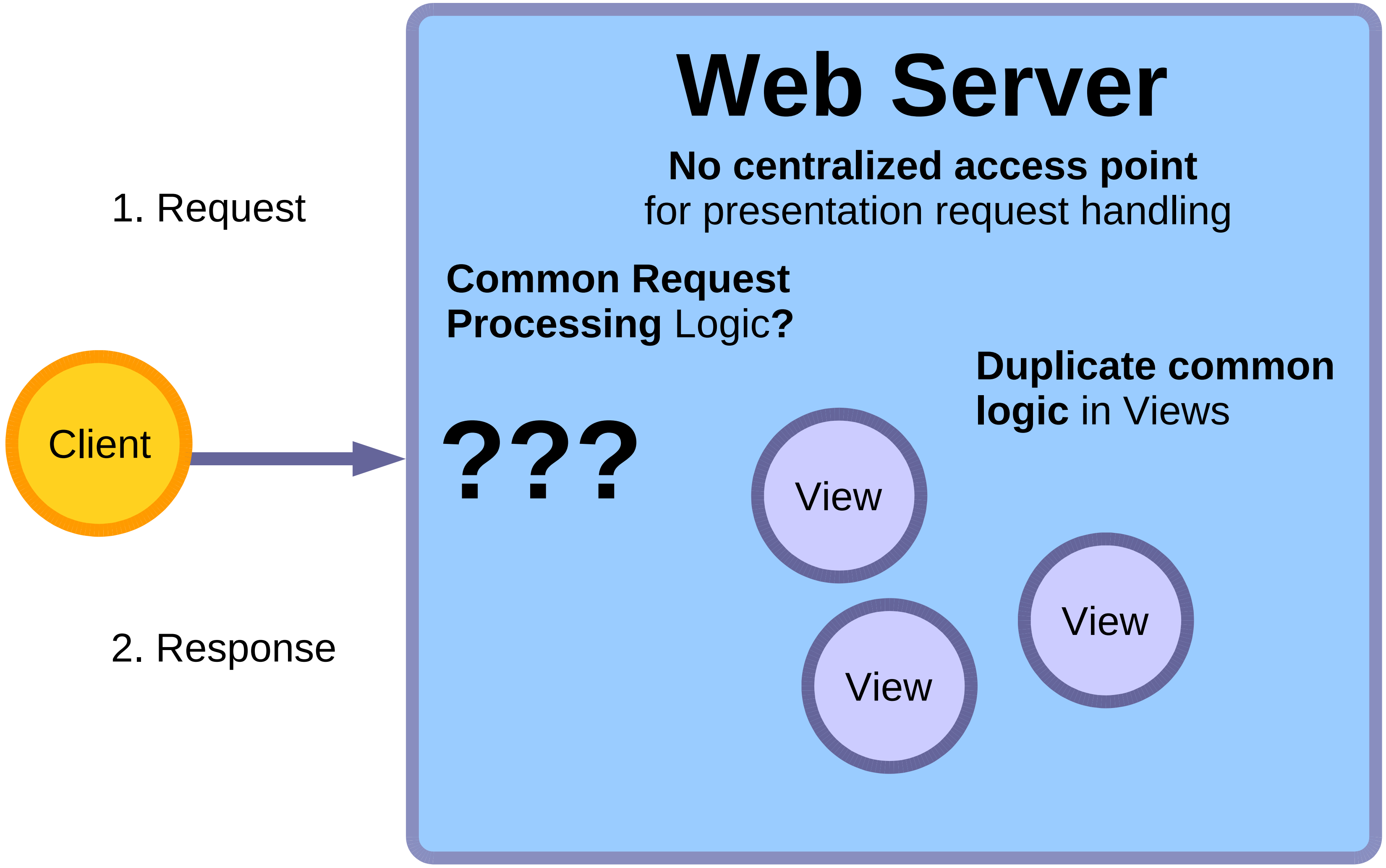
- Layered View helps to:
  - Clearly divide responsibilities
  - Decouple business logic completely
  - Provide a common “place” for pre-processing and post-processing of requests and responses (like logging, translations, transformations, etc.)
  - Expose a web service interface to existing business logic

Model View Controller

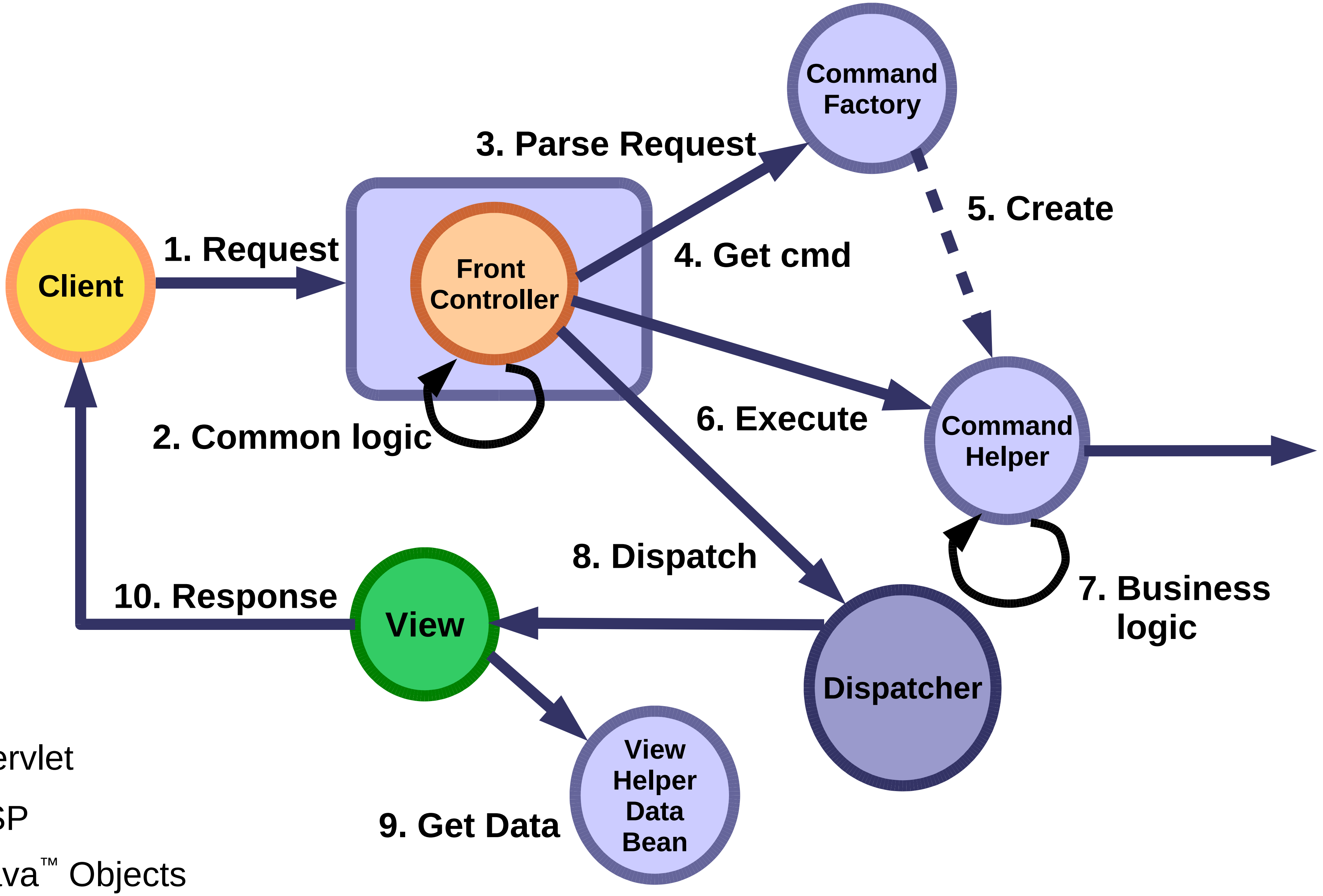
- **Model** = **business data and rules**
- **View** = view of **model**, **presents the user interface**
- **Controller mediates** their **interactions**



Web Tier: Problem


1. Request
2. Response
Client
Web Server
No centralized access point
for presentation request handling
Common Request Processing Logic?
???
Duplicate common logic in Views
View
View
View

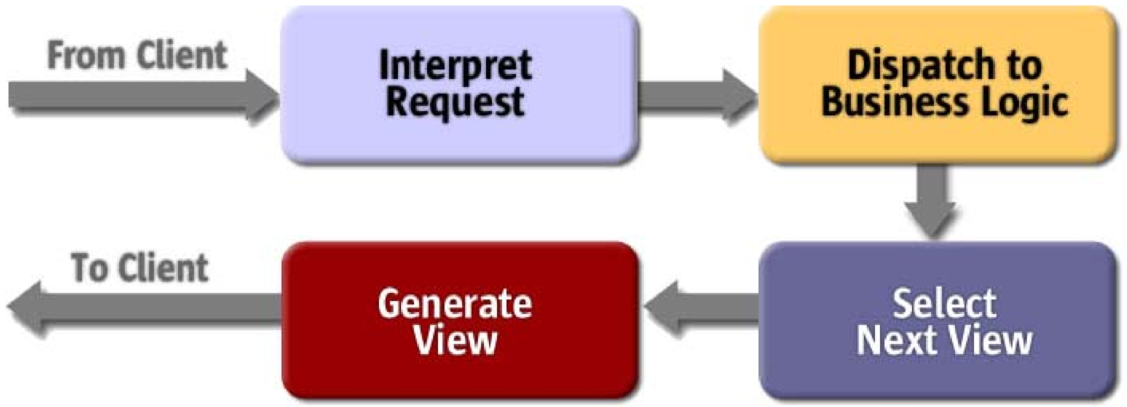
Service to Worker



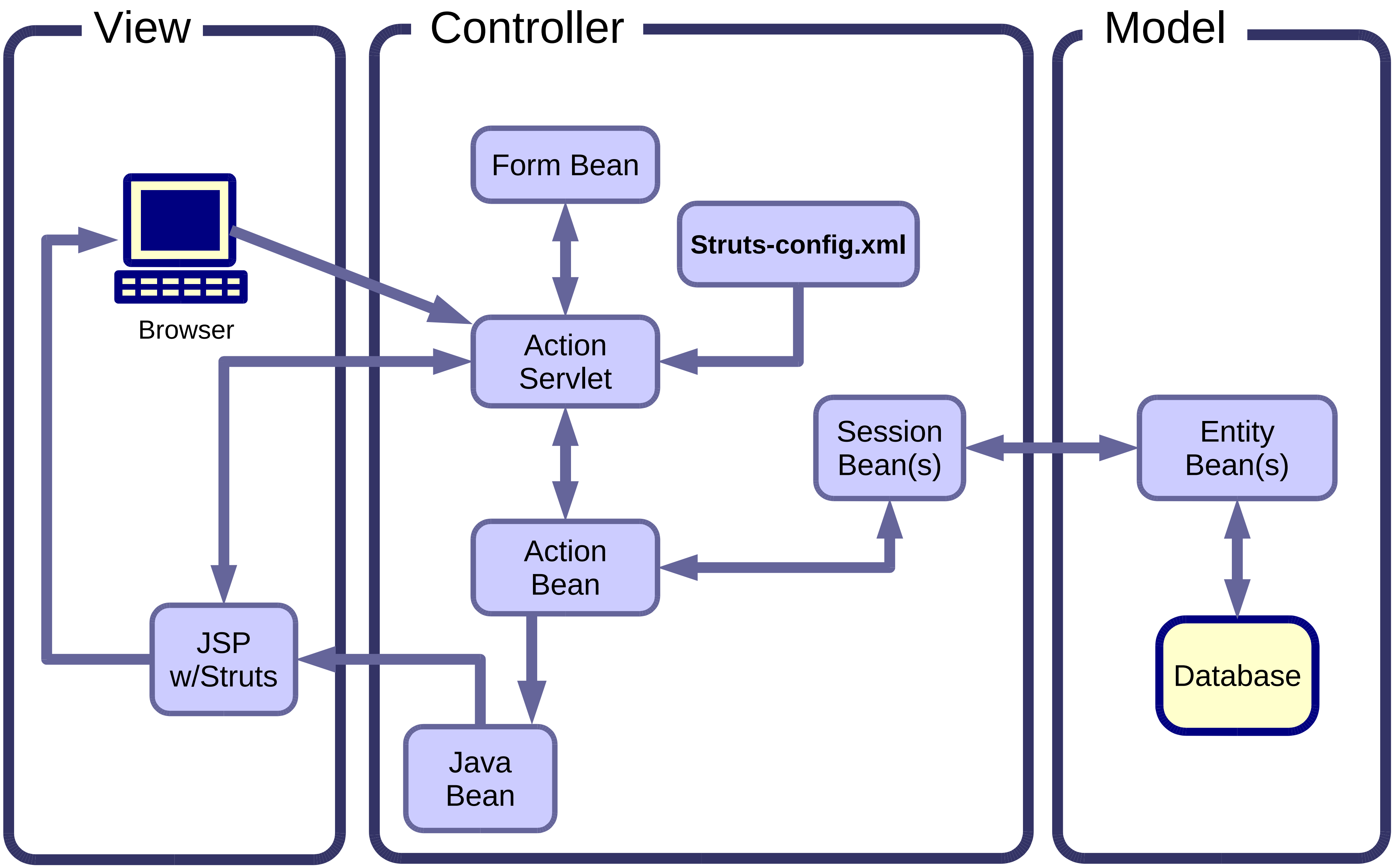
🟠 Servlet
🟢 JSP
🟣 Java™ Objects

Use an MVC Framework

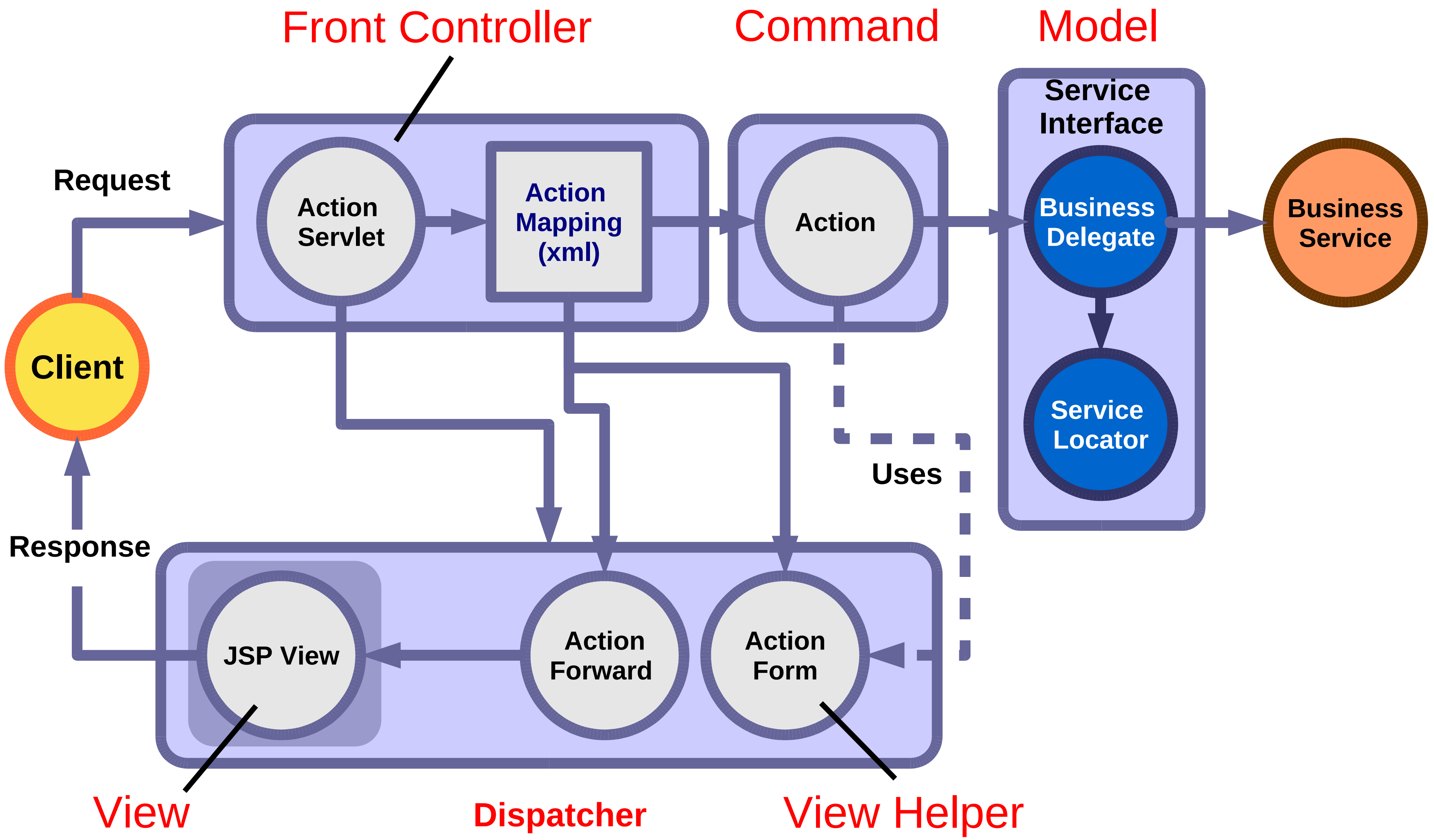
- Faster, easier development of
  - Request processing
  - Response generation
- Use Struts, JSF, Sun ONE ... application framework



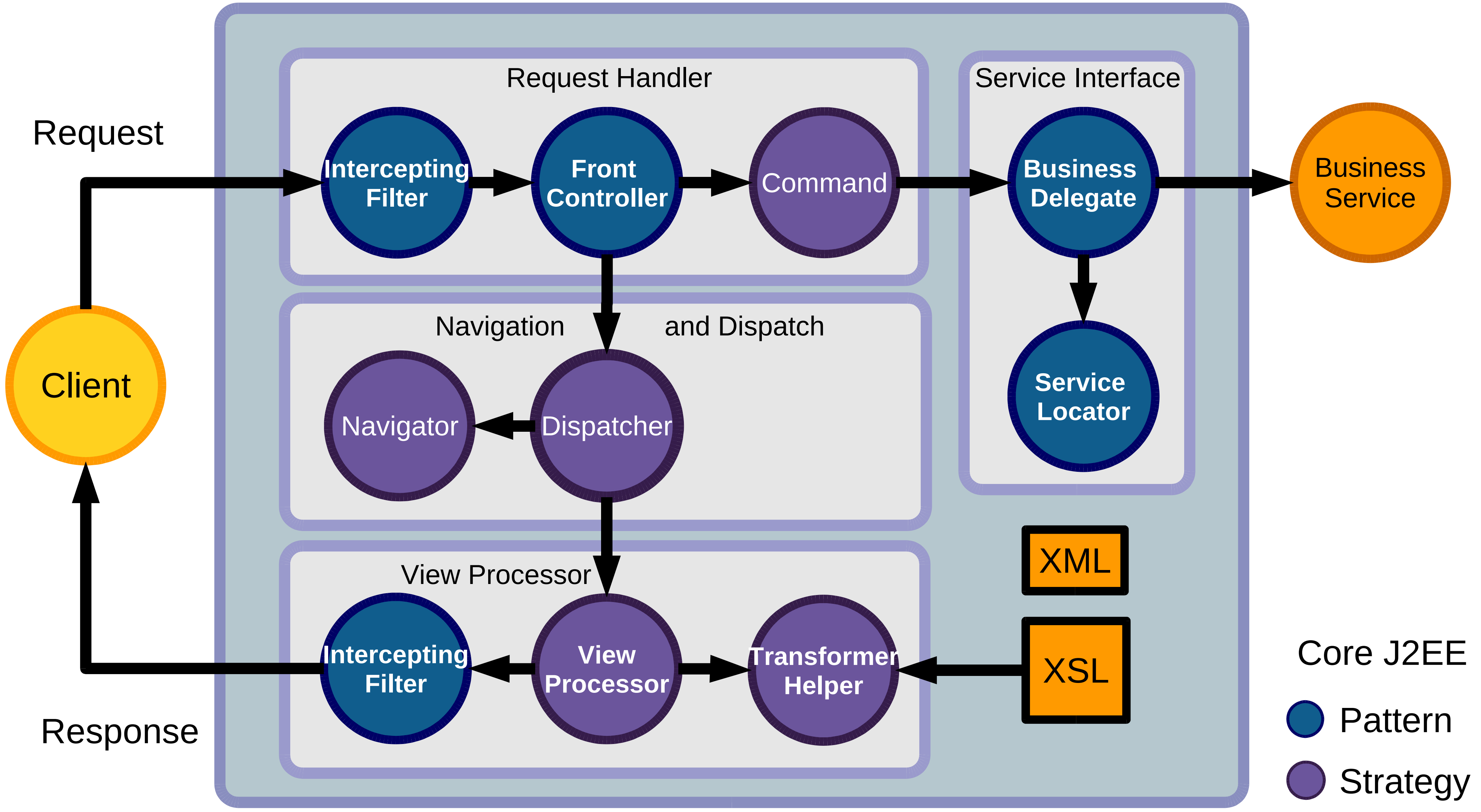
Struts Framework



Struts and Core J2EE Patterns



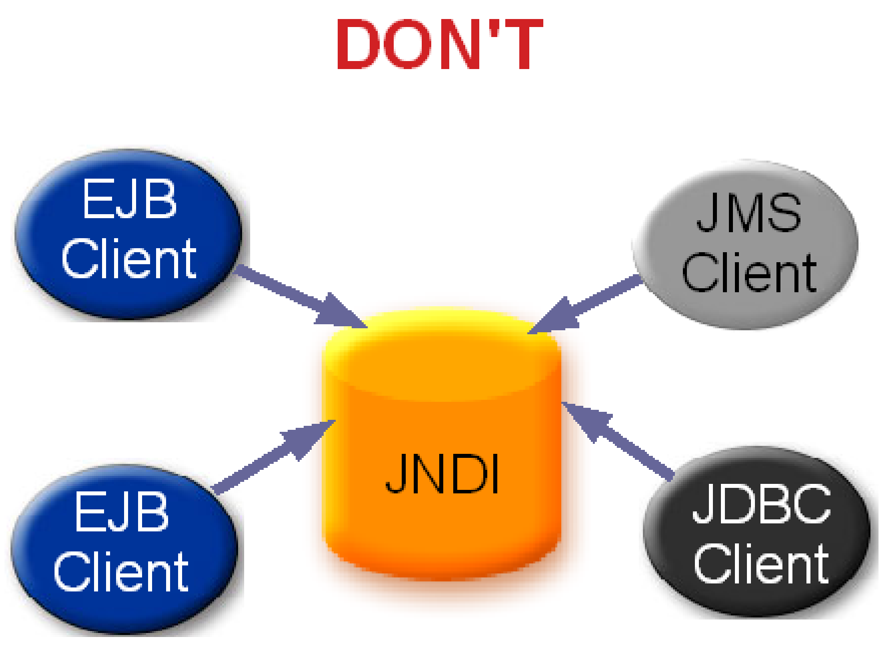
Ebay.com: Presentation Tier

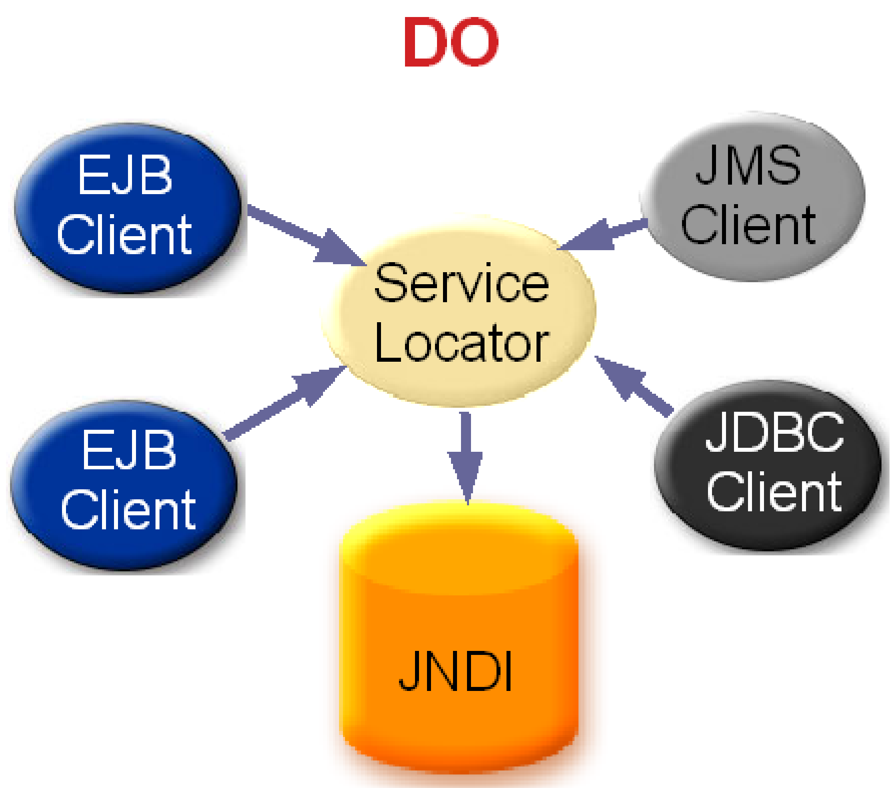


Cache for Performance

- Some things can be cached and shared (singleton):
  - InitialContext object
  - Anything retrieved from JNDI, EJB Home interfaces
  - Repeated lookups can be expensive! Use Service Locator Pattern
- Some things are cached by individual clients (session):
  - Cache search results when you are only displaying a few at a time (non-transactional data)
  - JDBC™ `CachedRowSet`
  - Value List Pattern

Service Locator Pattern



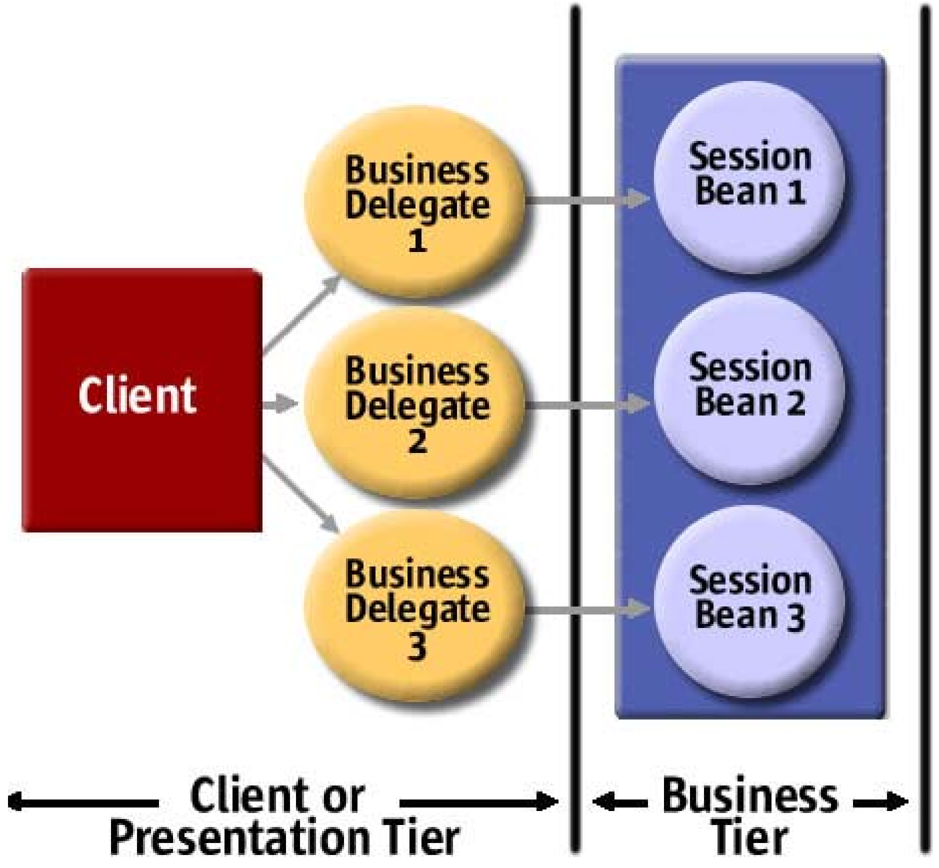


Repeated lookups can be expensive!

Use the Service Locator Pattern to
Cache references obtained by JNDI lookups
(ejb references, datasources, jms)

Business Delegate

- Client independent from ejb tier
- Ejb details hidden
- Mock objects can be used to test client w/o ejb tier



Tips for Servlets

- Servlets are Multithreaded
  - Avoid use of shared modified class variables
  - Synchronize only small blocks in code
- Remove servlet sessions when they are no longer needed:
  - In logoff call `session.invalidate()`
- On client side, don’t access your EJBs directly, use service locator with business delegate

Servlet Tips

- DO use high performance J2EE design patterns
  - MVC, Service Locator
- DON’T hold resources – explicitly free them
- DON’T intensively process immutable objects.

Tips for JSPs

- Avoid scriptlets in JSP
  - Use JSTL (JSP 2.0 Standard Tag Libraries):
  - `<c:forEach var="item" values="${cart}">`
- Pass data to JSP in Servlet request not session
- If JSP does not use the HTTP session
  - Use `<%page session="false"%>` to prevent HTTP Sessions from being automatically created

Push your development further

Some EJB Tier Best Practices and Patterns

Do you need EJBs?

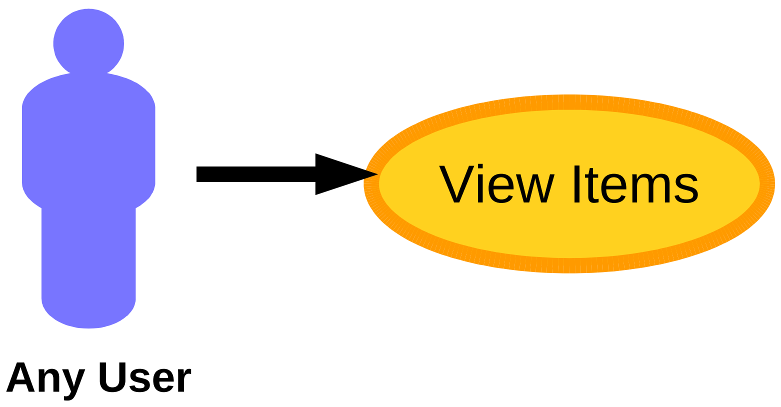
- Do you need declarative transactional support?
- Do you need to distribute your business logic?
- Do you need JMS, JAX-RPC, RMI?
- Do you need to support multiple client types?
- Do you need method-level security on your objects?
- Do you need a standards-based architecture?

Architecting Applications

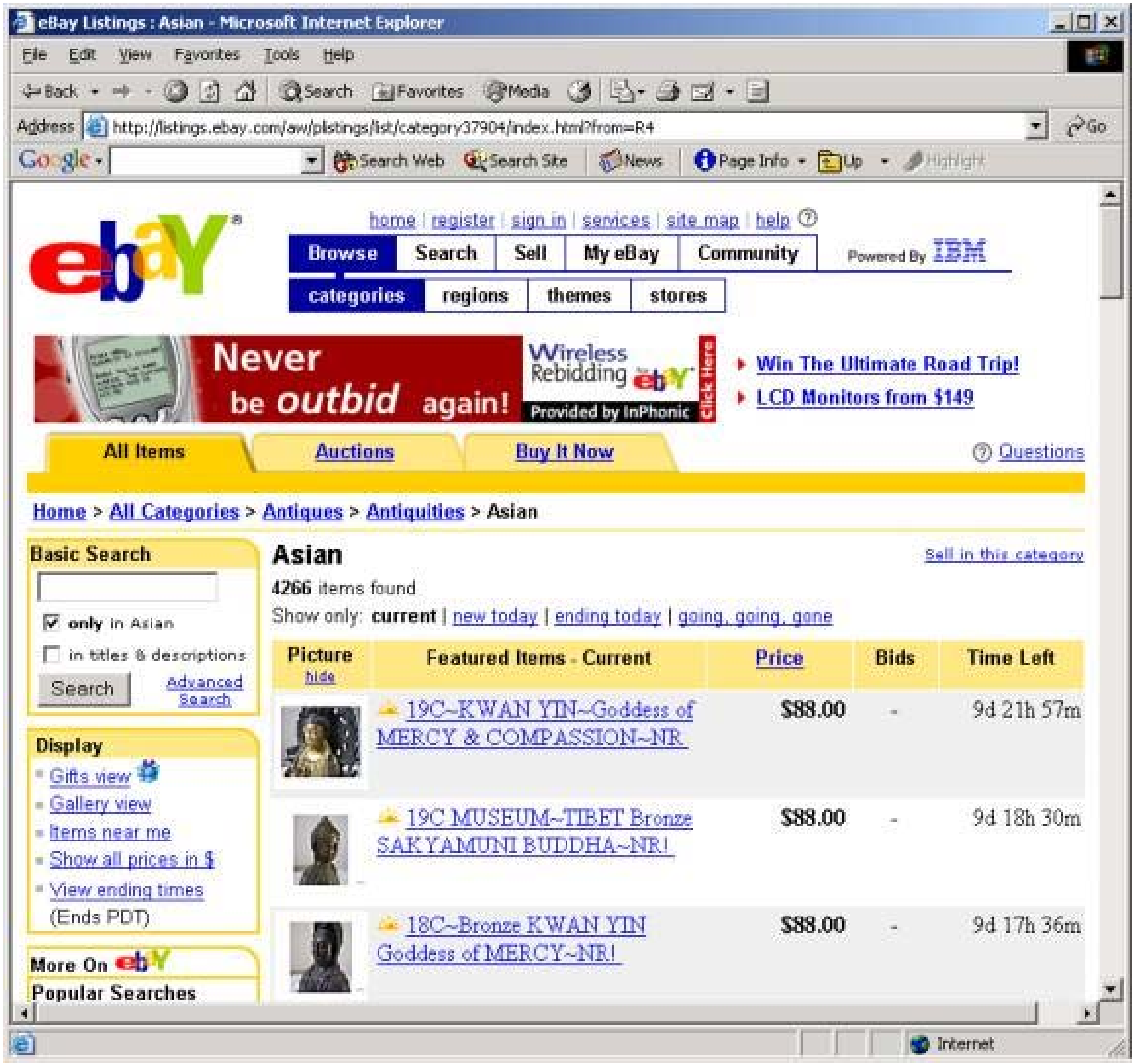
- Functional requirements captured via use-cases
- Use-cases implemented using MVC & Command design patterns
- Implement Business Logic as a Service

Use case realization

View Items



Any user can view any items available for bidding or sale


eBay Listings : Asian - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://listings.ebay.com/aw/plistings/list/category37904/index.html?from=R4
Go
Search Web Search Site News Page Info Up Highlight
home | register | sign in | services | site map | help
Browse Search Sell My eBay Community
Powered By IBM
categories regions themes stores
Never be outbid again!
Wireless Rebidding
Provided by InPhonic
Click Here
Win The Ultimate Road Trip!
LCD Monitors from $149
All Items Auctions Buy It Now
Questions
Basic Search
only in Asian
in titles & descriptions
Search
Advanced Search
Display
Gifts view
Gallery view
Items near me
Show all prices in $
View ending times
(Ends PDT)
More On eBay
Popular Searches
Internet

Home > All Categories > Antiques > Antiquities > Asian

Asian

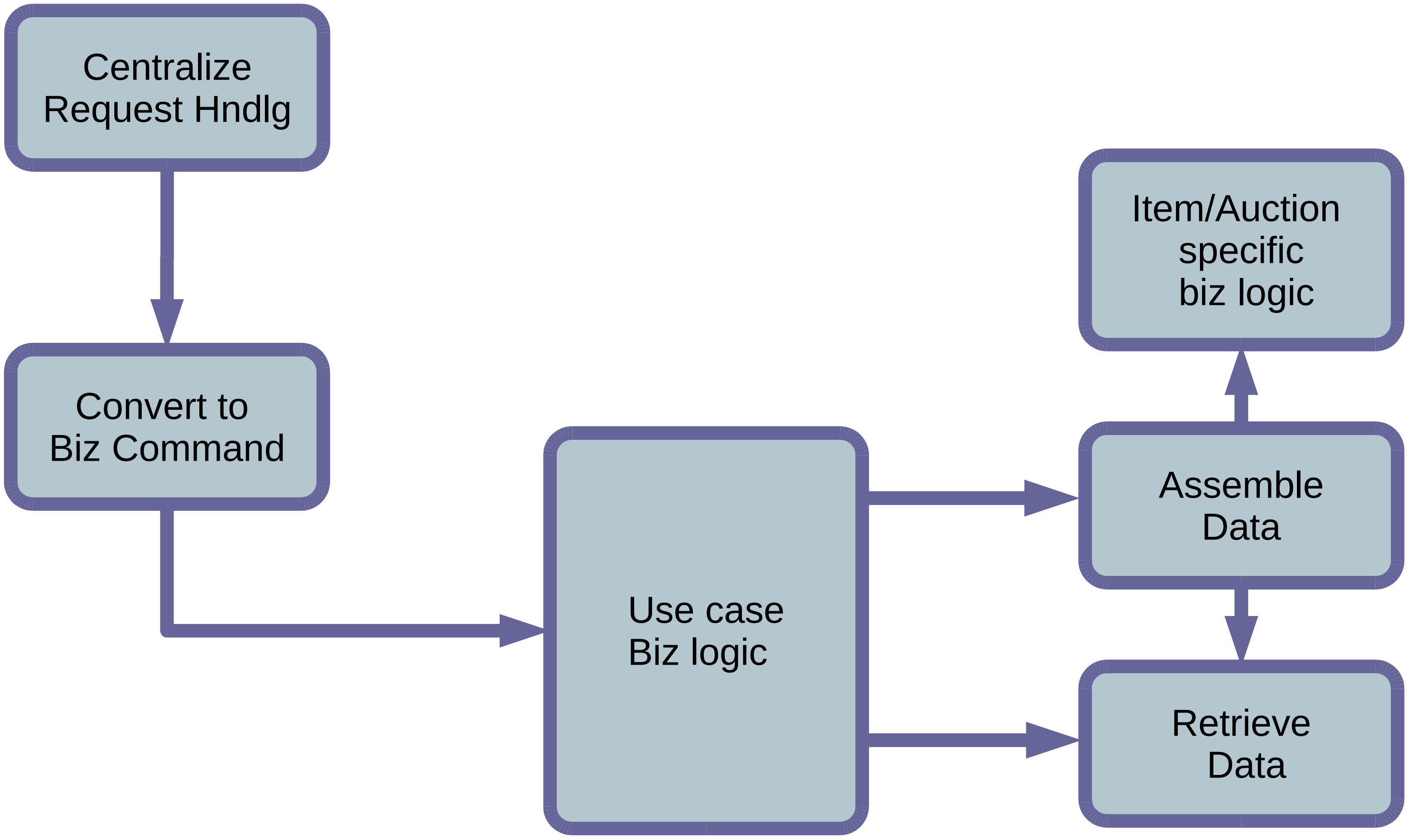
4266 items found

Sell in this category

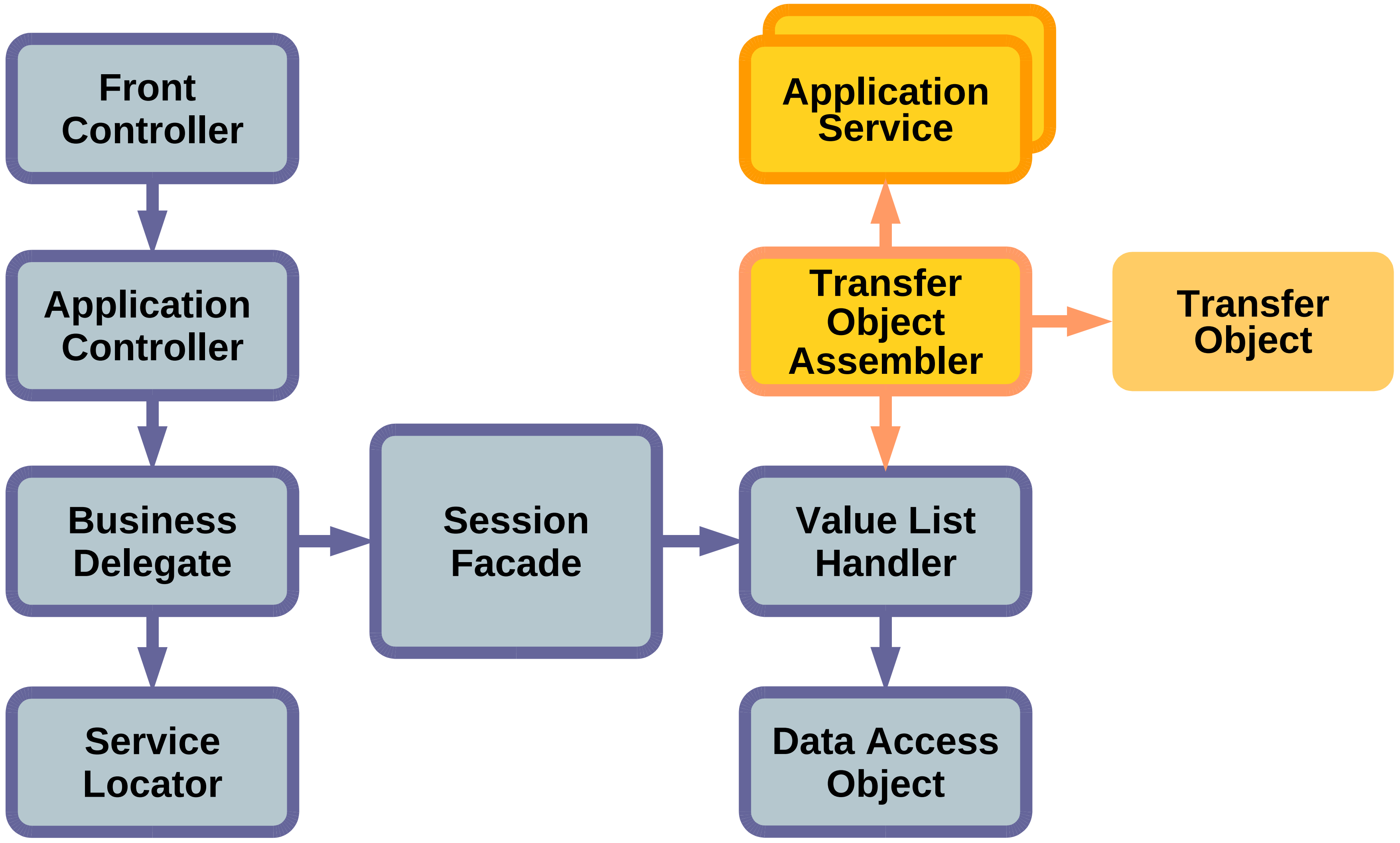
Show only: current | new today | ending today | going, going, gone

Picture hide	Featured Items - Current	Price	Bids	Time Left
	19C~KWAN YIN~Goddess of MERCY & COMPASSION~NR	$88.00	-	9d 21h 57m
	19C MUSEUM~TIBET Bronze SAKYAMUNI BUDDHA~NR!	$88.00	-	9d 18h 30m
	18C~Bronze KWAN YIN Goddess of MERCY~NR!	$88.00	-	9d 17h 36m

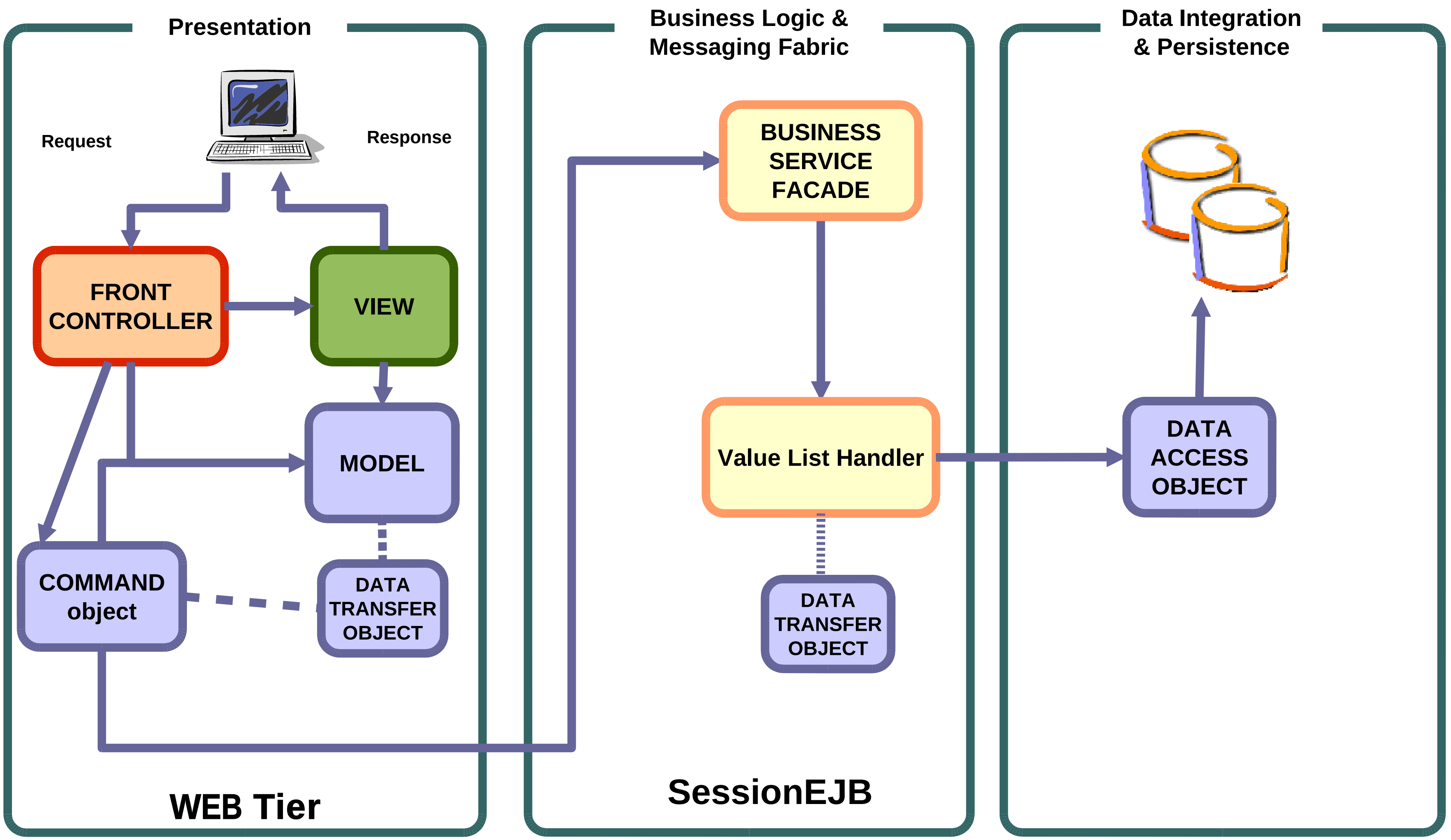
View Items Design Requirements



View Items Design

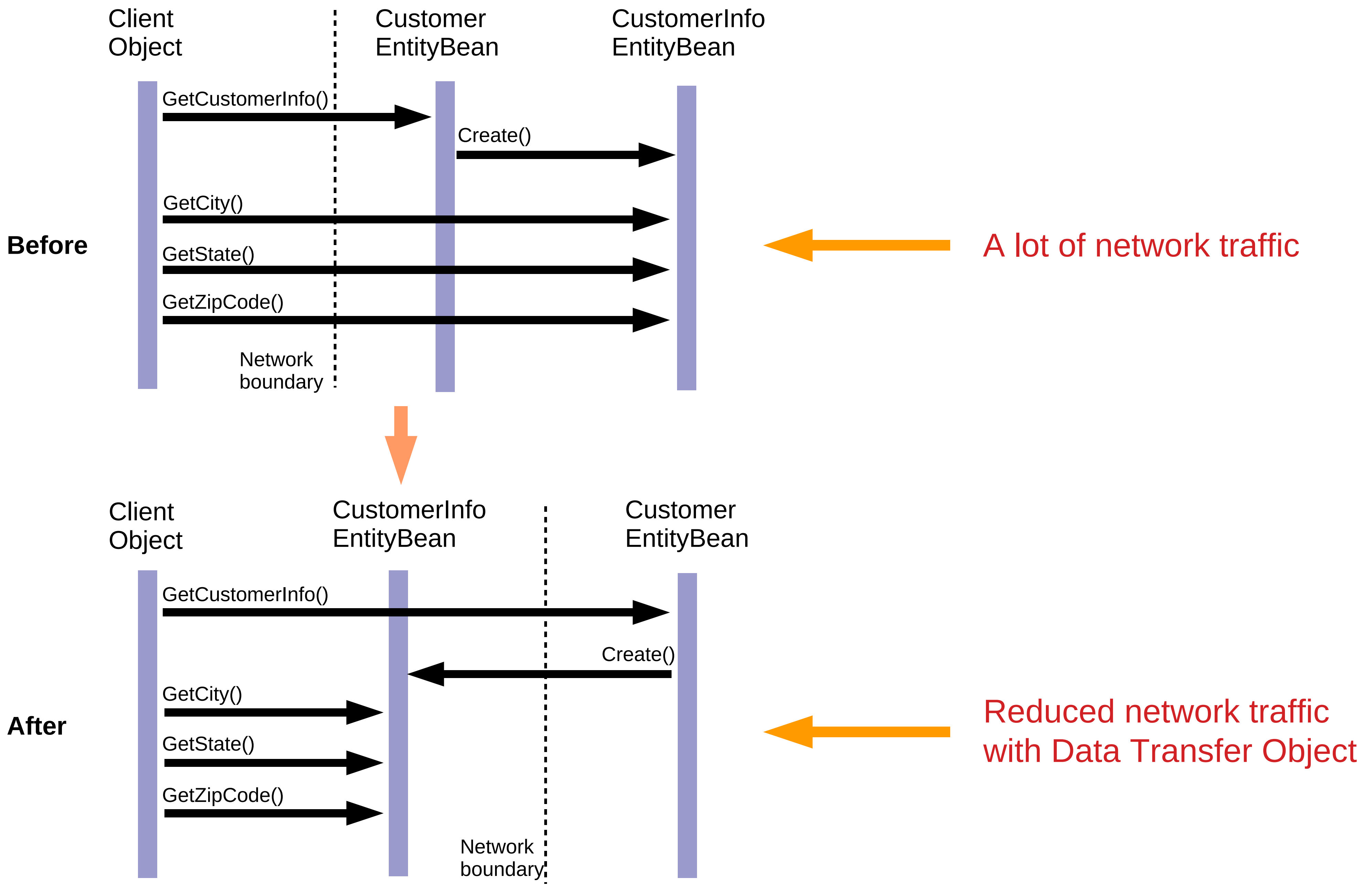


Core Application Design Patterns



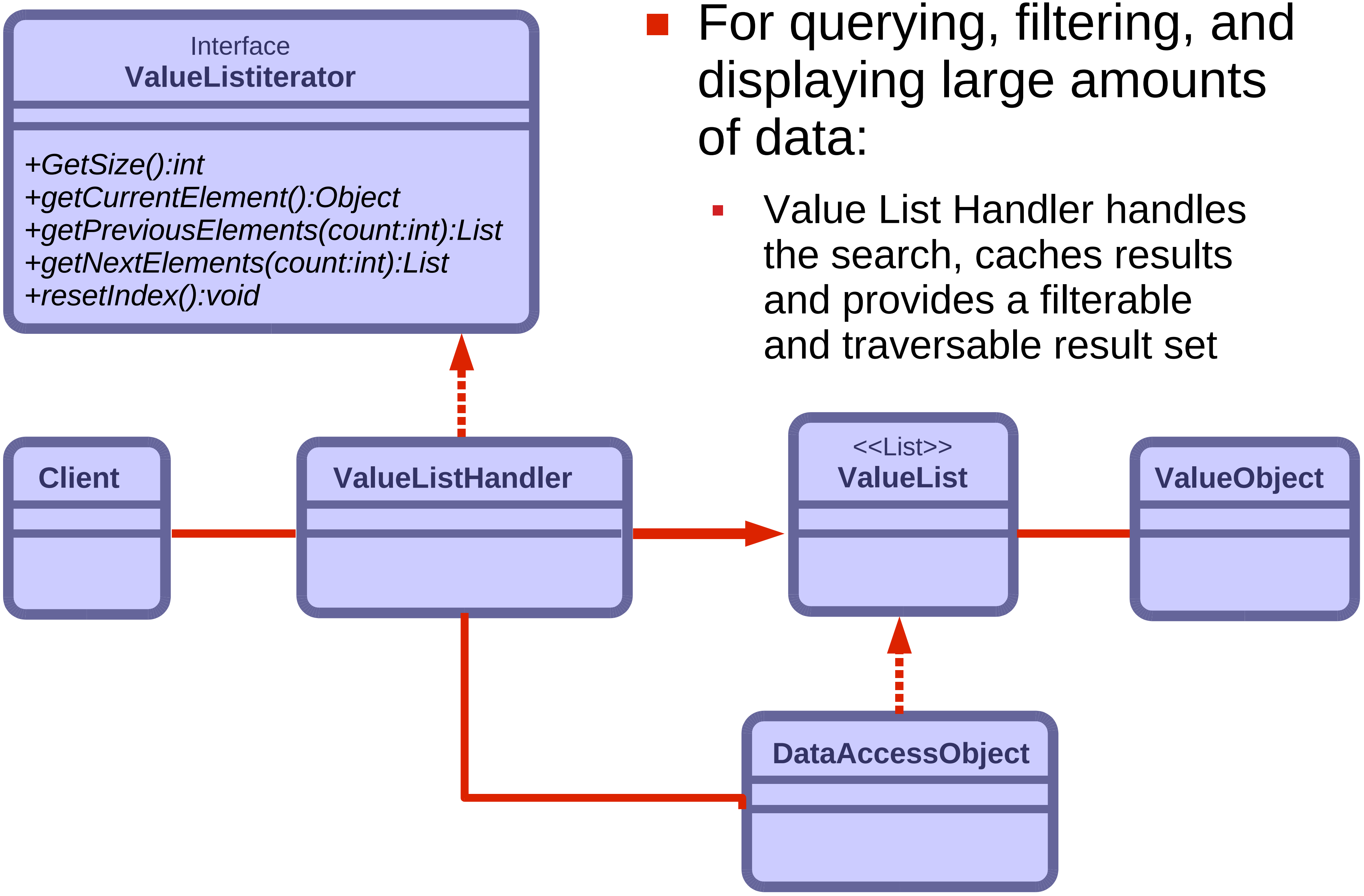
🟠 Servlet	🟣 Java™ Objects
🟢 JSP	🟡 Session EJB

Data Transfer Object

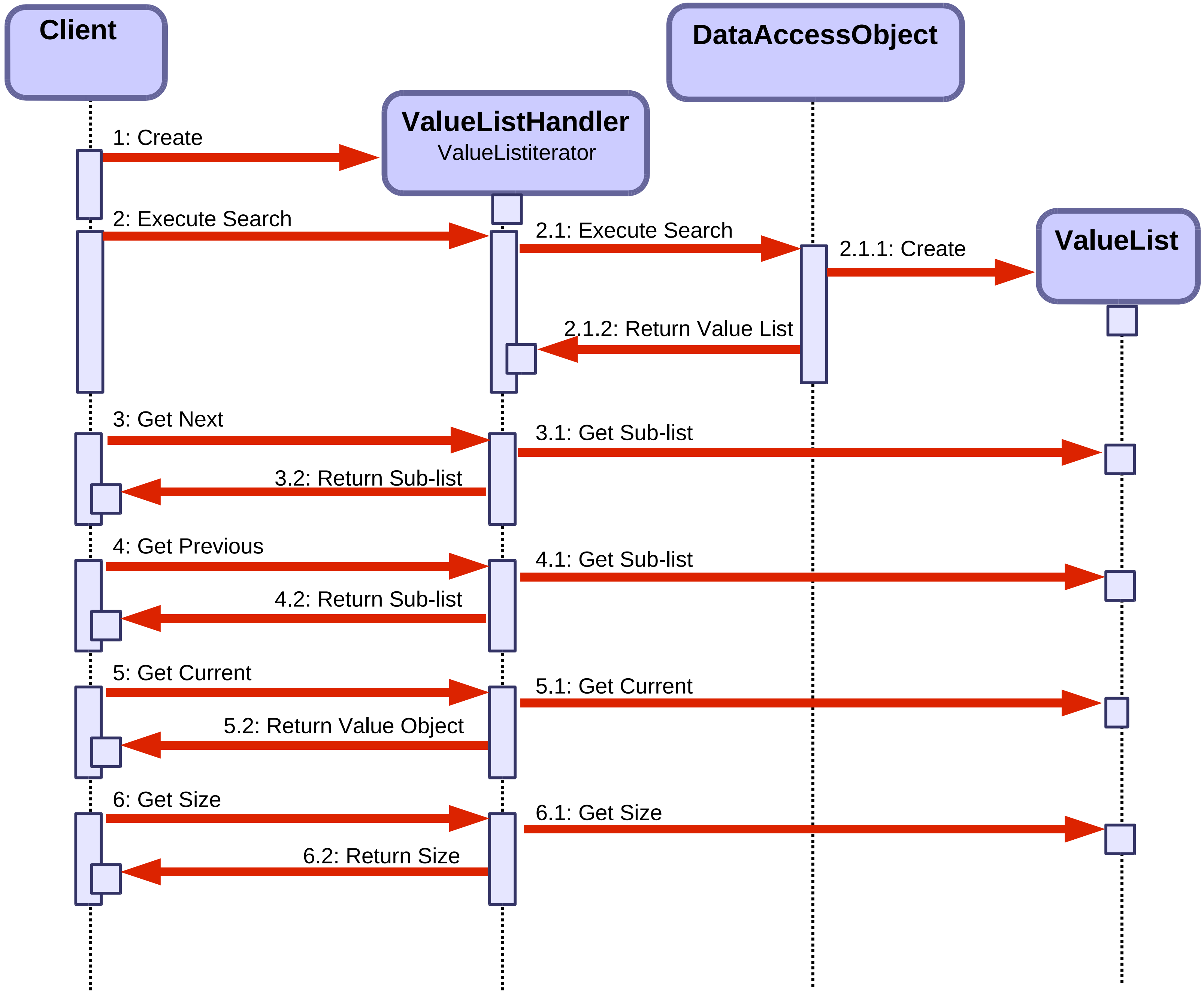


Value List Handler

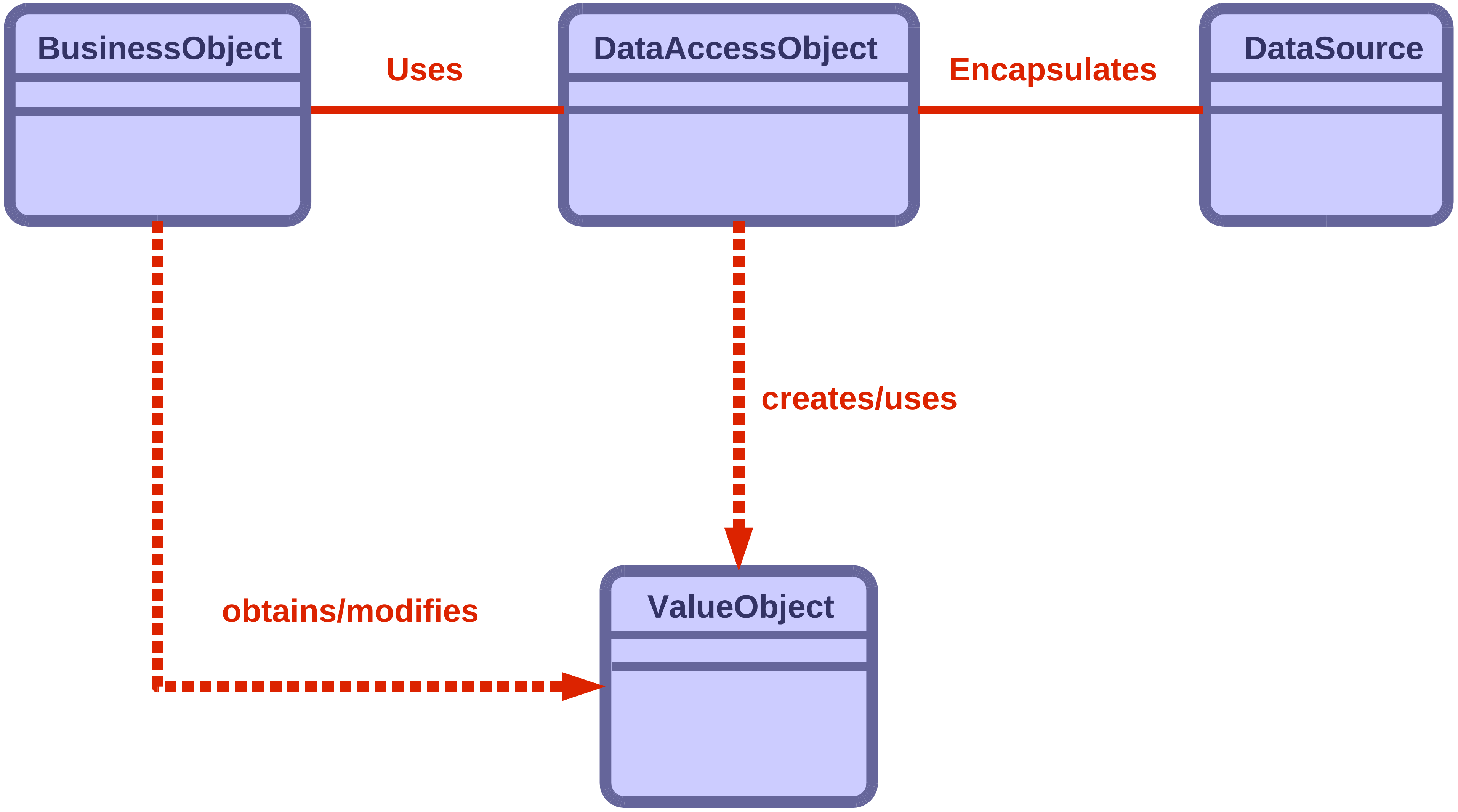
- For querying, filtering, and displaying large amounts of data:
  - Value List Handler handles the search, caches results and provides a filterable and traversable result set



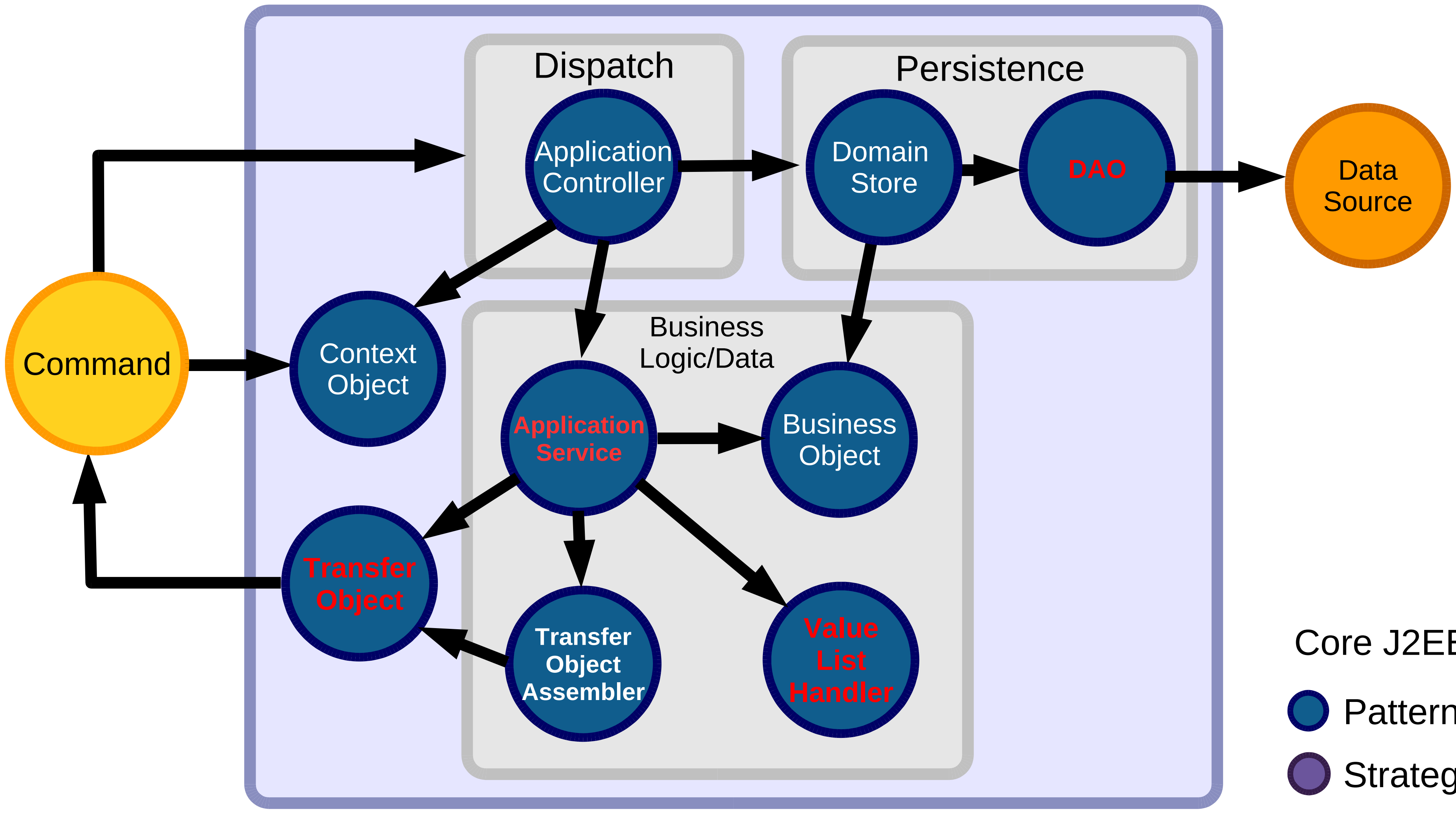
Sequence Diagram



Data Access Object



eBay.com: Business Tier



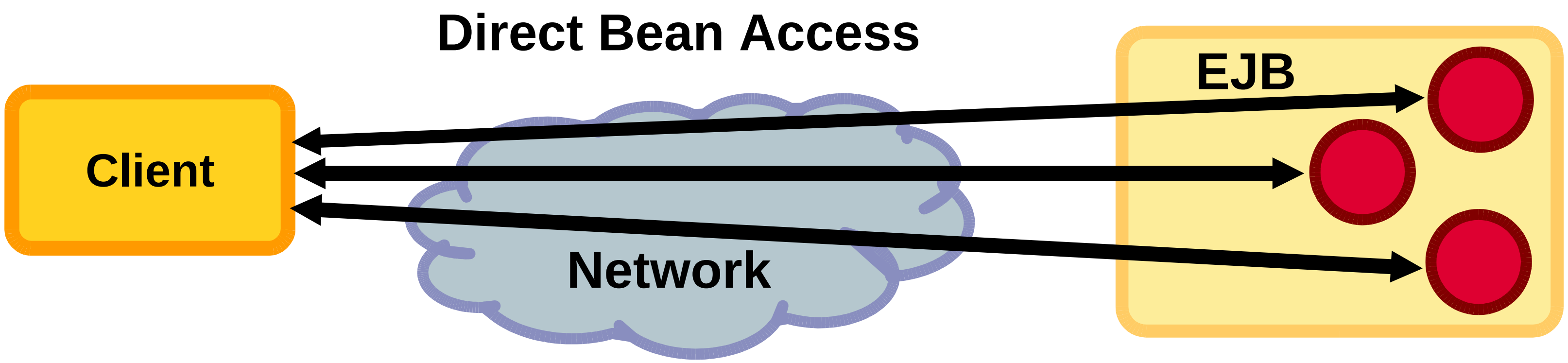
Core J2EE

● Pattern

● Strategy

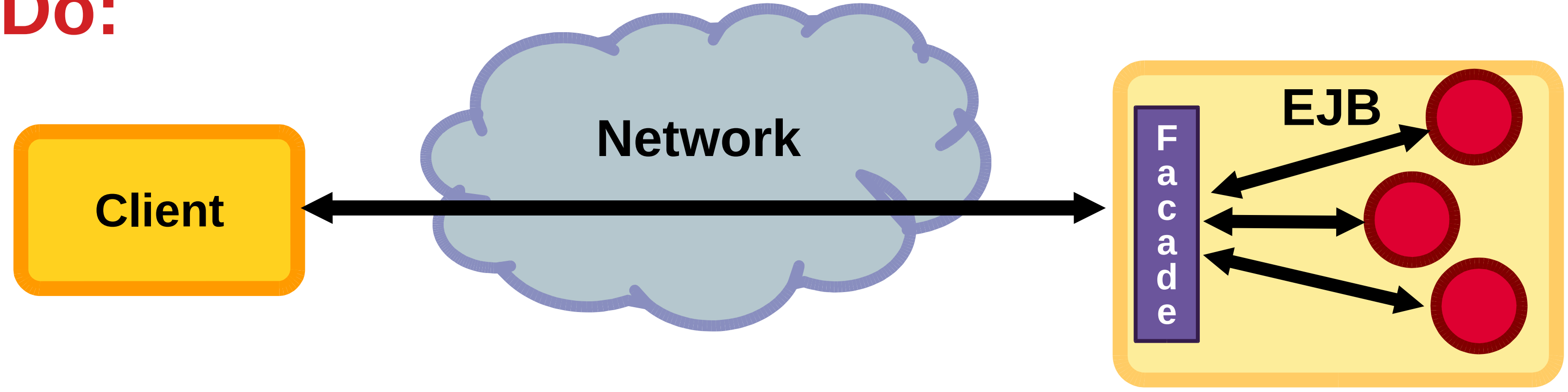
Session Facade pattern

Don't:

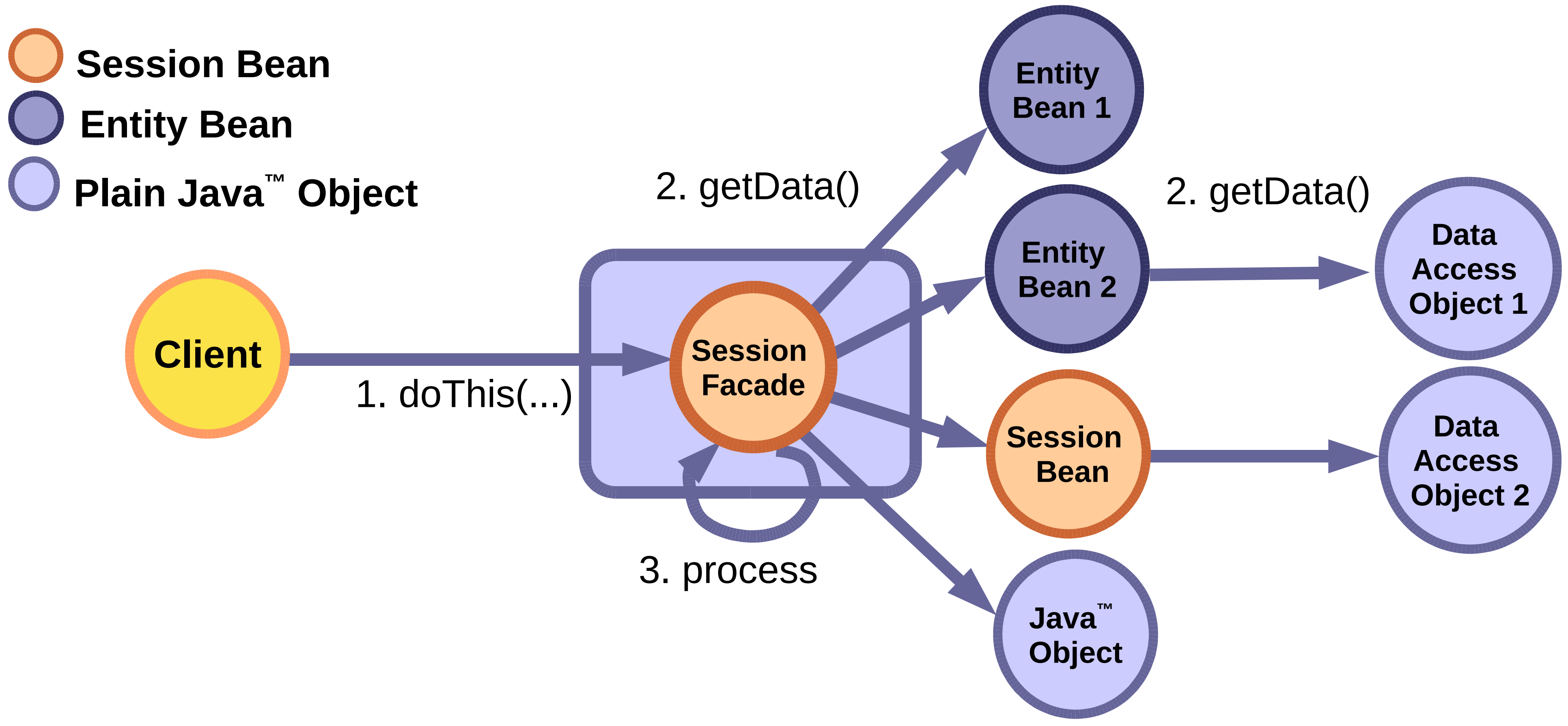


Direct Entity Bean access results in excessive network overhead and multiple transactions

Do:

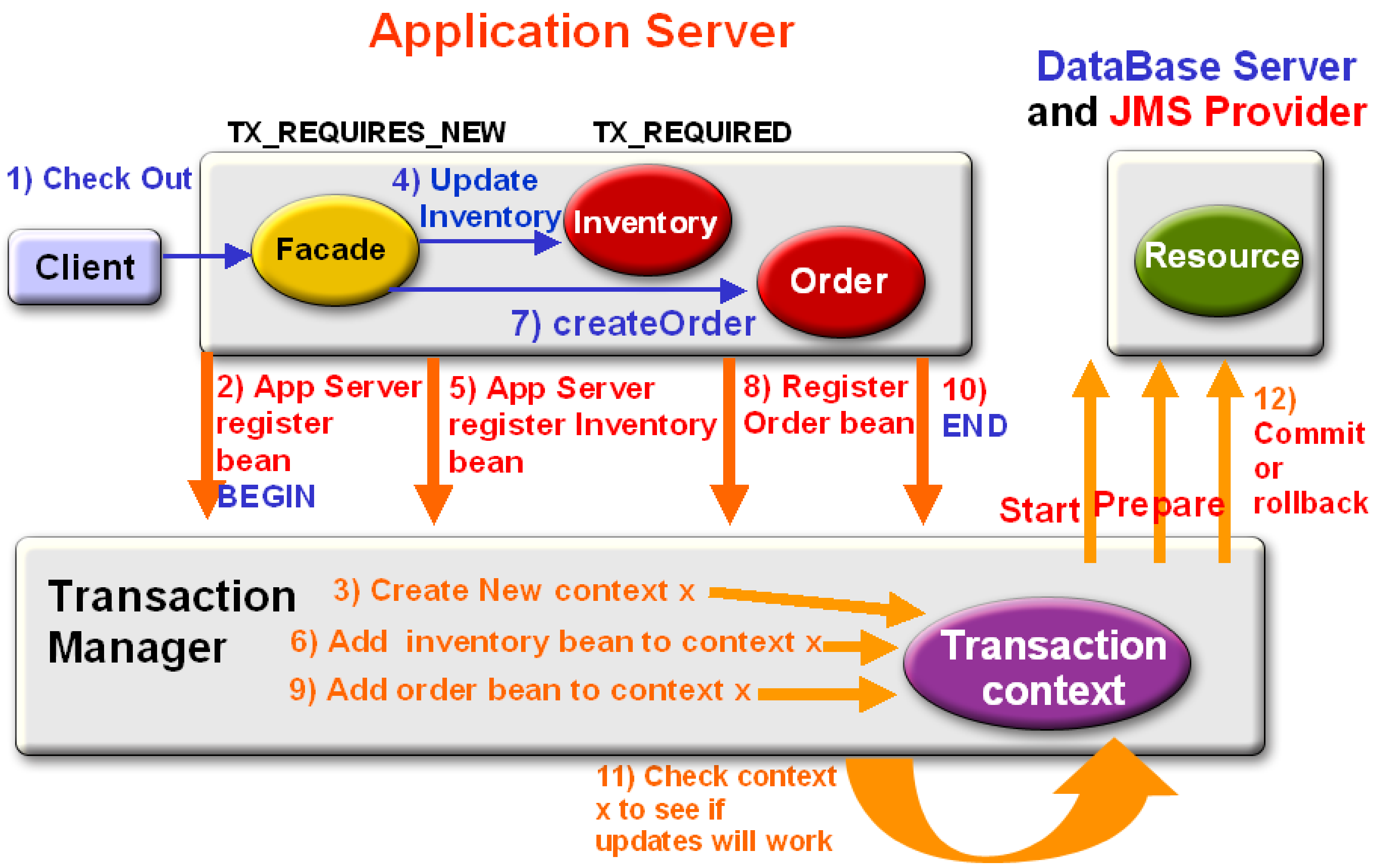


Session Facade pattern



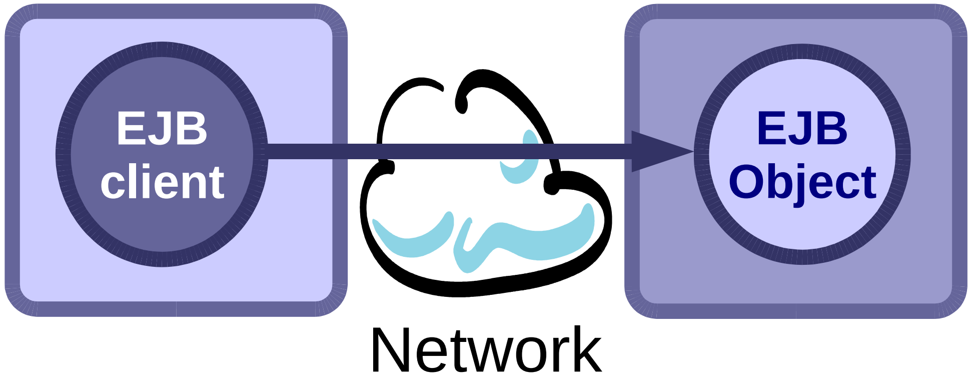
- Use the Session Facade:
  - To reduce network overhead
  - To group related updates into transactions

Container Managed Transactions



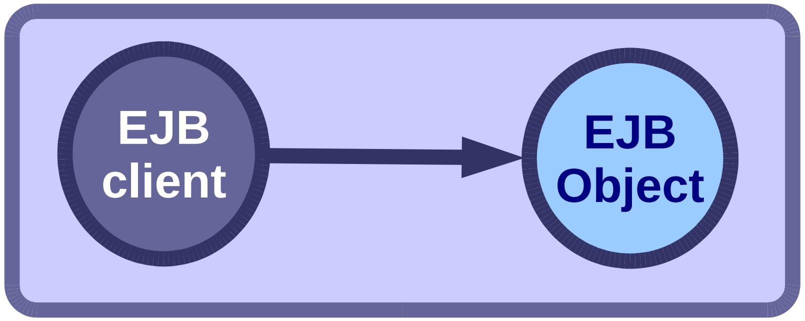
Local vs. Remote Interfaces

Remote Interfaces



Pass by Value:
Serialize/deserialize
method parameters

Local Interfaces



Pass by Reference:
Better performance

Session EJB tips

- Do use session beans as a service facade
  - With local interfaces: Faster than remote
  - Container managed transactions: calls to entity beans within one transaction!
- DO design fine-grained components as Java™ classes
- DO minimize the number of service calls with a large-grained component design

Session EJB tips

- Stateless session beans give best performance, watch use of stateful
- Remove stateful session beans to reduce unnecessary passivation
  - Bean.remove()
- Limit size of objects stored in session beans (performance for passivation)

Service Tier DON'T s

- DO NOT store large amount of data in Stateful SessionEJB
- DO NOT access entity EJBs remotely
- DO NOT implement fine-grained components as remote EJBs

Design Patterns Summary

- **Value Object**
  - Exchanges data J2EE tiers
- **Service Locator**
  - Holds results of JNDI lookups
- **Value List Handler**
  - Handles larger result sets from database queries
- **Business Delegate**
  - Simplifies the coupling between the J2EE Web & EJB tiers
- **Session Facade**
  - Provide business logic functionality
- **Data Access Object**
  - Isolation layer for interfaces to a systems resources
- **Etc.**
  - www.sun.com/developer/blueprints/.

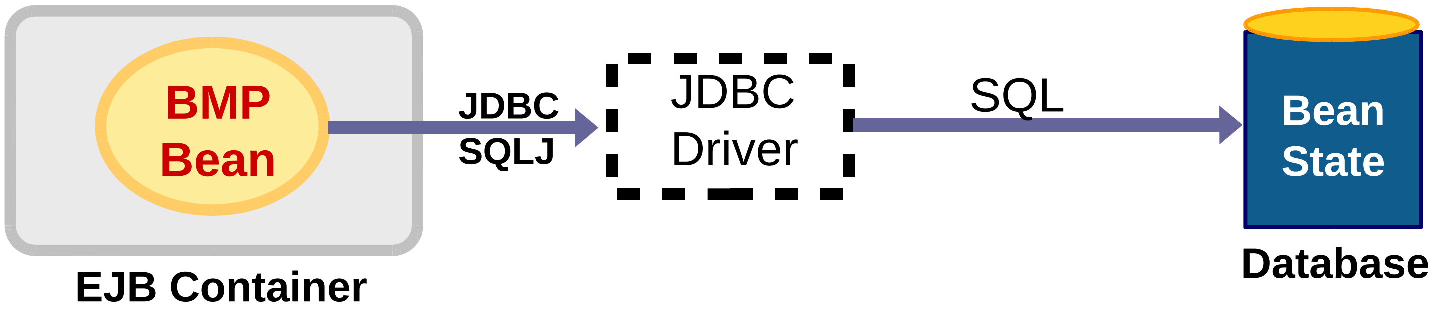
Push your development further

Persistence Options

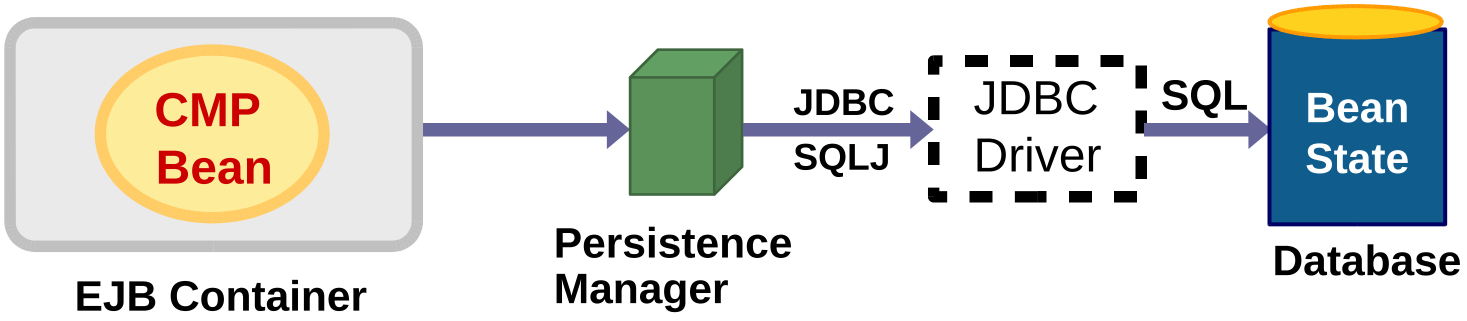
Data Access in J2EE

- Formal object-oriented model
  - EJB container managed persistence
    - Great choice for representing existing databases
  - JDO
- Tabular access
  - Plain JDBC
  - JDBC RowSets
    - Good choice for ease of development
- Non-relational, non-object data access
  - Use J2EE Connectors

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

Use BMP or DAO for

- Dealing with legacy database and/or other persistence store
- Previously written complex application
- Connector driven data stores, EIS
- Efficient execution of complicated queries
  - Bulk updates
  - Complex joins
  - Aggregates

Entity Bean Tips

- Do not use EJB entity beans for batch loading or queries that return large result sets. Use Data Access Objets encapsulating JDBC
- Use CMP rather than BMP entity bean when possible
- Do not call EJB entity bean get & set methods from client
  - Wrap with session beans to provide course grain access and single transaction context

JDBC Tips

- Select a certified, high performance type 2 JDBC driver
- Tune connection pool size
- Close resources as soon as you’re done with them (in finally)
  - E.g. Statements, Connections, ResultSets…
- Use JDBC’s PreparedStatement instead of Statement when possible
- Turn off Auto-Commit
  - Group updates into a transaction

JDBC Tips

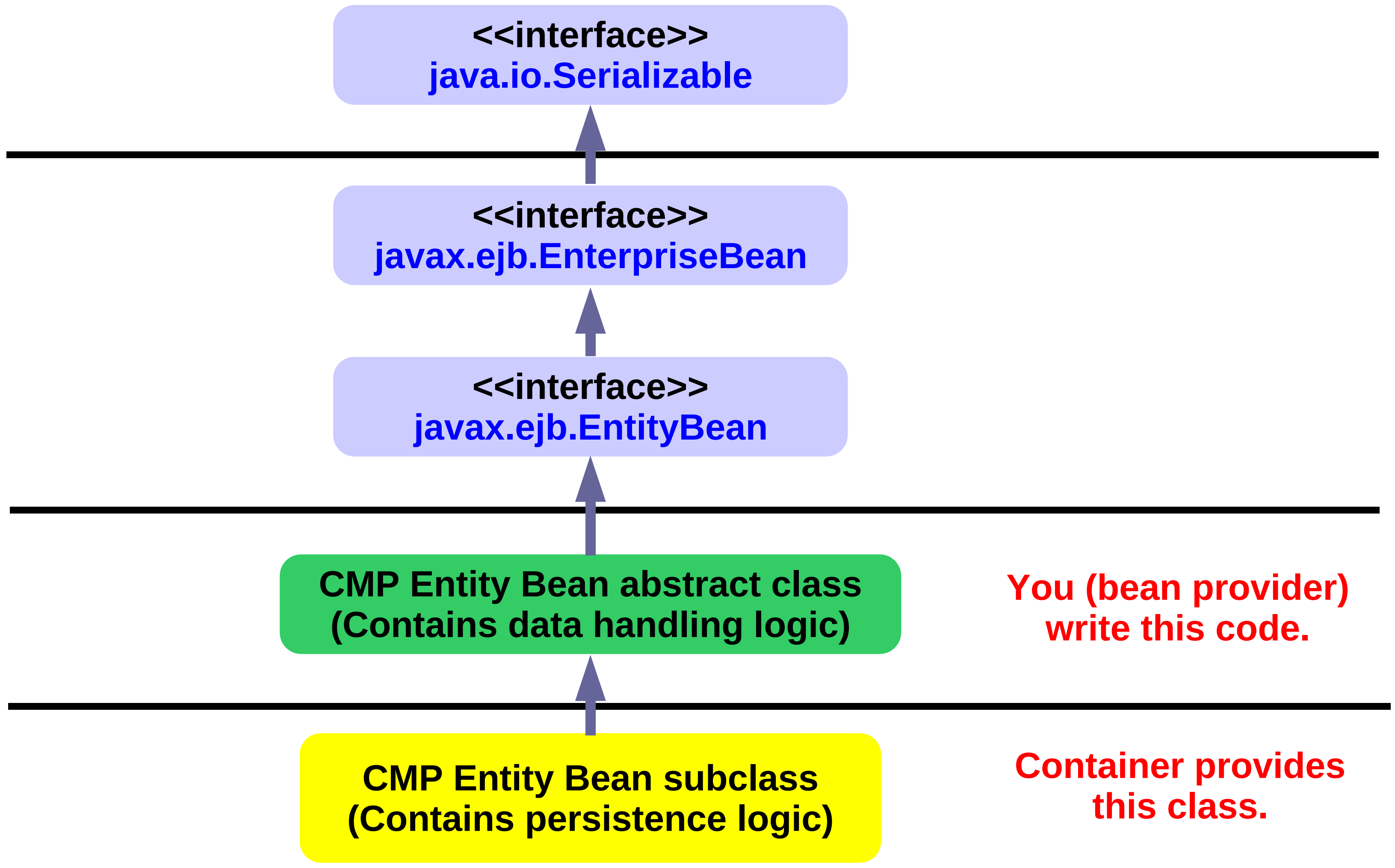
- JDBC Supports a number of high performance optimization techniques
- Typically you can optimize JDBC performance by:
  - Using CachedRowSets
  - Using setDefaultRowPrefetch
  - Using statically bound column types
  - Using update batching
  - Using statement caching

Database Performance

- Common performance bottleneck
- Typical problems:
  - Inefficient queries - sending SQL data that asks the database to do more work than necessary
  - Excessive querying - efficient queries called too frequently
  - Large Data Sets - processing large sets of data in ResultSets

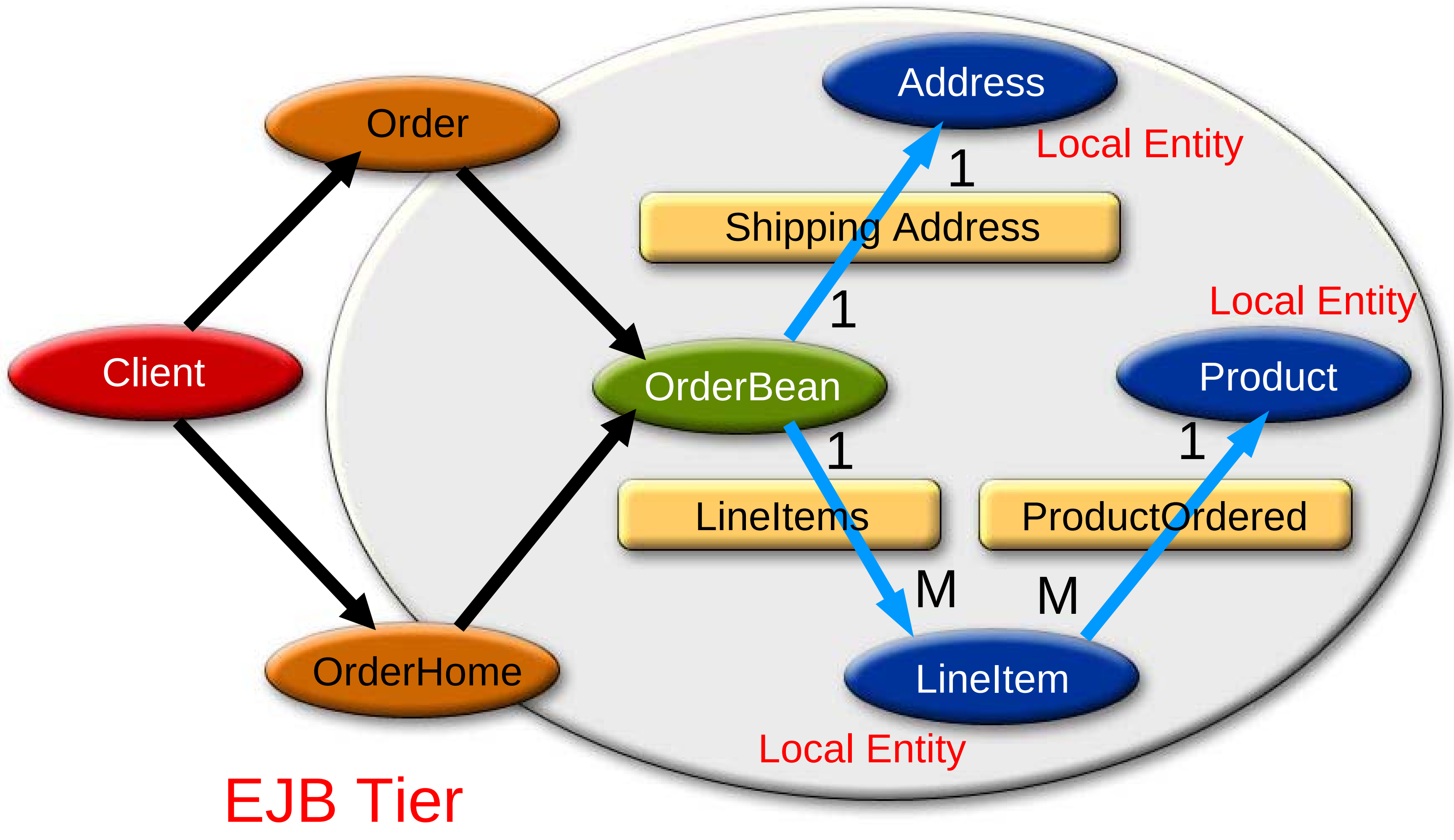
CMP 2.0 Entity Beans

Entity Bean is now an abstract class, container extends it



<<interface>>
java.io.Serializable
<<interface>>
javax.ejb.EnterpriseBean
<<interface>>
javax.ejb.EntityBean
CMP Entity Bean abstract class
(Contains data handling logic)
You (bean provider) write this code.
CMP Entity Bean subclass
(Contains persistence logic)
Container provides this class.

Container Managed Relationships



Accessors for CMP and CMR

```
public abstract class OrderBean implements EntityBean {

   private EntityContext context;

   //access methods for cmp fields

   public abstract String getOrderID(); //primary key
   public abstract void setOrderID(String id);
   . . .
   //access methods for cmr fields

   public abstract Collection getLineItems();
   public abstract void setLineItems(Collection lineItems);
```

CMP 2.0 Relationship Handling

- In CMP 2.0, you declare fields and relationships in deployment descriptor
  - Container generates all the necessary code

```
<ejb-jar>
<enterprise-beans>
     ... define your enterprise beans ...
     <cmp-field> elements represent container-managed
  persistent fields
</enterprise-beans>
<relationships>
     ... define EJB relationships ...
</relationships>
```

EJB™ QL Example

Find orders for a specific product:

```
SELECT OBJECT(o)
FROM Orders o, IN (o.lineItems) l
WHERE l.product.name = ?1
```

- XML:

```
<query>
  <query-method>
    <method-name>findByProduct</method-name>
    <method-intf>LocalHome</method-intf>
  </query-method>
  <ejb-ql>SELECT OBJECT(o) FROM Order o,IN (o.lineItems) l
       WHERE I.product.name= ?1</ejbql>
</query>
```

Advantages of CMP 2.0 for developer

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

Push your development further

CMP

Advantages of CMP 2.0 for Container

- Optimization is possible because persistent fields are only accessible via get and set methods
  - Lazy loading
  - Dirty checking
  - Optimistic locking
- Optimization is possible in query operation
  - Because Query is defined in deployment descriptor via EJB QL

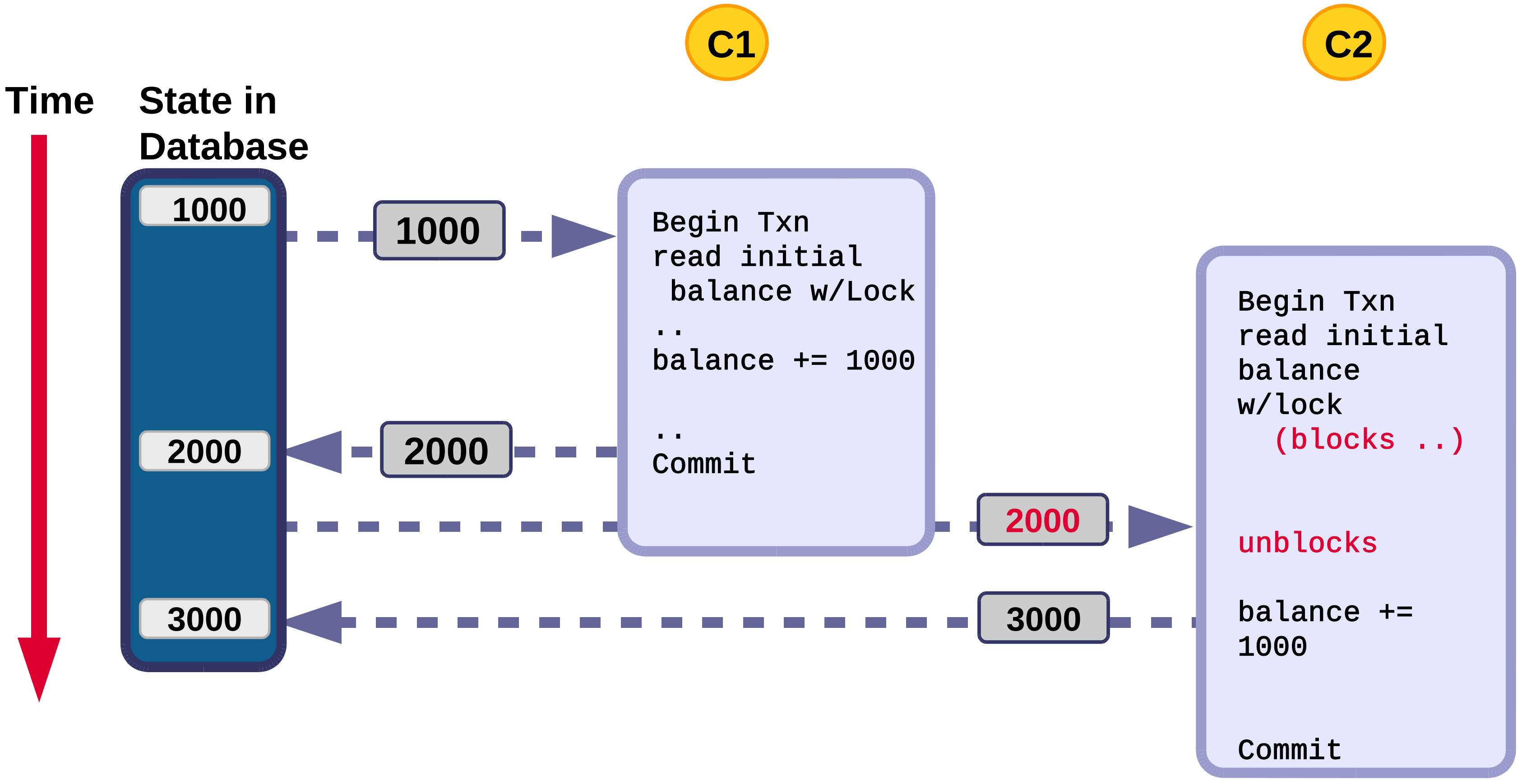
CMP Optimizations

- Aggressive Loading
  - Loading fields relationships and fields of children in the same query
- Lazy Loading
  - Deferring loading of any data until it is accessed
- Dirty Writes
  - Only update data which has been changed in the database

CMP: Standard Vs. Vendor Specific Features

- Standard features
  - Declarative specification of:
    - Persistent attributes, abstract schema, relationships, queries for finder/select methods(via EJBQL), transactional attributes
- Vendor specific
  - O/R mapping, concurrency and consistency semantics, caching semantics, performance and usability

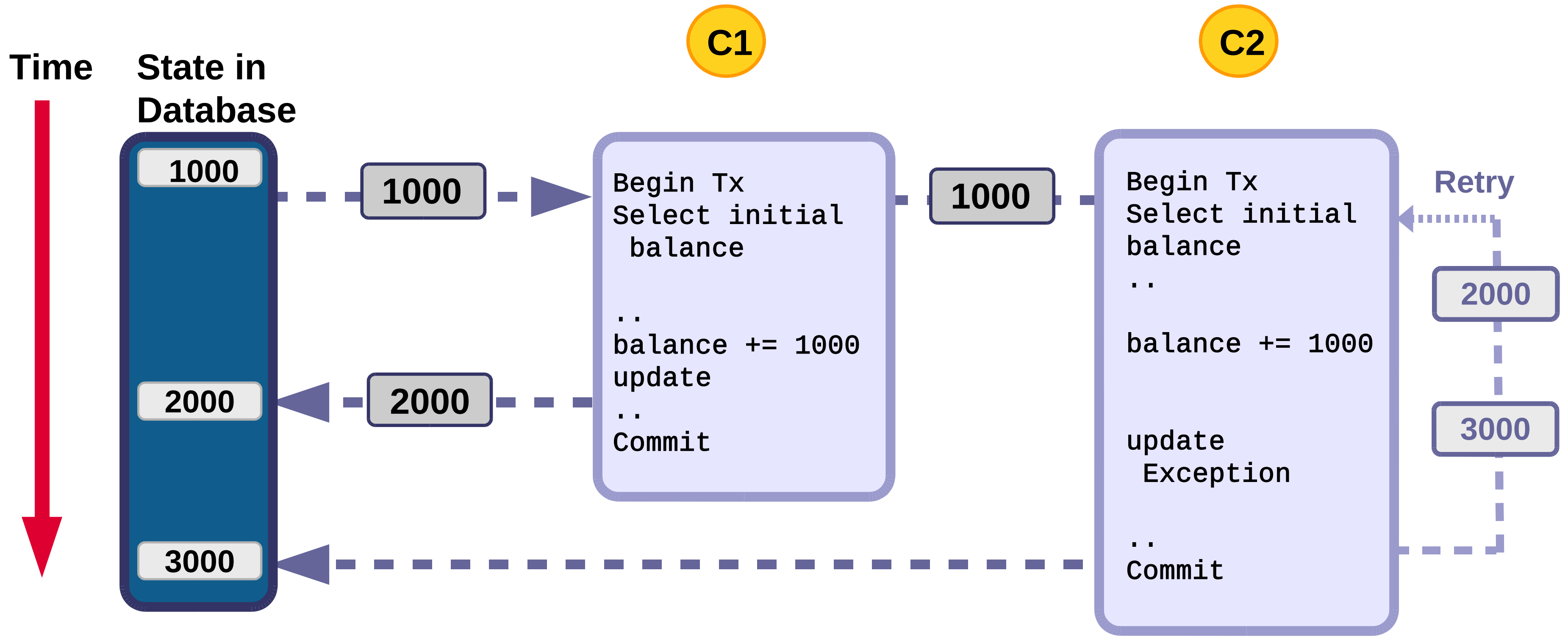
Pessimistic Locking



The row is locked for the duration of the transaction

Optimistic Locking With “ Update-conflict” Detection

Concurrent Access to same Bean (same Primary Key)



```
Begin Tx
Select initial
 balance

..
balance += 1000
update
..
Commit
```

```
Begin Tx
Select initial
balance
..

balance += 1000


update
 Exception

..
Commit
```

Concurrent Access to same Bean (same Primary Key)
Lock obtained at update time, possible conflict detection with version id column

Usage Guidelines

- Pessimistic
  - Serialized access
  - Recommended for beans where conflicts are bound to happen
  - Scalability depends on DB and App server locking granularity
- Optimistic
  - Concurrent access
  - Large-scale deployment
  - Requires collision and exception handling
  - Choices for conflict detection

Data and Caching

- Static data
  - Keep a local copy, hang on to it in memory, don't worry about being stale
- Near static data
  - Keep a local copy, hang on to it in memory, lazily check for updates
- Dynamic data
  - Work on local copy, cache carefully, use optimistic locking
- Hot data
  - Pessimistic locking

Database Isolation Modes

- Read Uncommitted
  - Dirty reads, non-repeatable reads and phantom reads can occur
- Read Committed
  - Dirty reads are prevented; non-repeatable reads and phantom reads can occur
- Repeatable Read
  - Dirty reads and non-repeatable reads are prevented; phantom reads can occur
- Serializable
  - Dirty reads, non-repeatable reads and phantom reads are prevented

Entity Bean Caching

- Commit Option A
  - At the end of the transaction, the instance stays ready and the instance state is valid
- Commit Option B
  - At the end of the transaction, the instance stays ready but the instance state is NOT valid
- Commit Option C
  - At the end of the transaction, neither the instance nor its state is valid
- Best Option: Check your app server

Transaction Do's

- Do use READ_COMMITTED with Optimistic locking as much as possible
- Do use Optimistic locking only and SERIALIZABLE if required.

Transaction Don'ts

- Don't use SERIALIZABLE if it can be avoided
- Don't use pessimistic locks if it can be avoided

EJB Summary

- EJB Container Services – use appropriately for:
  - Distributed transaction management
  - Robust security service
  - Resource management (threads, sockets, database connections)
  - Container persistence
  - Remote accessibility
  - Dispatch and life-cycle management.
- Use EJB 2.0 local interfaces for performance improvements
  - When running in same JVM.

J2EE Performance Tips

Push your development further

Tune App Server

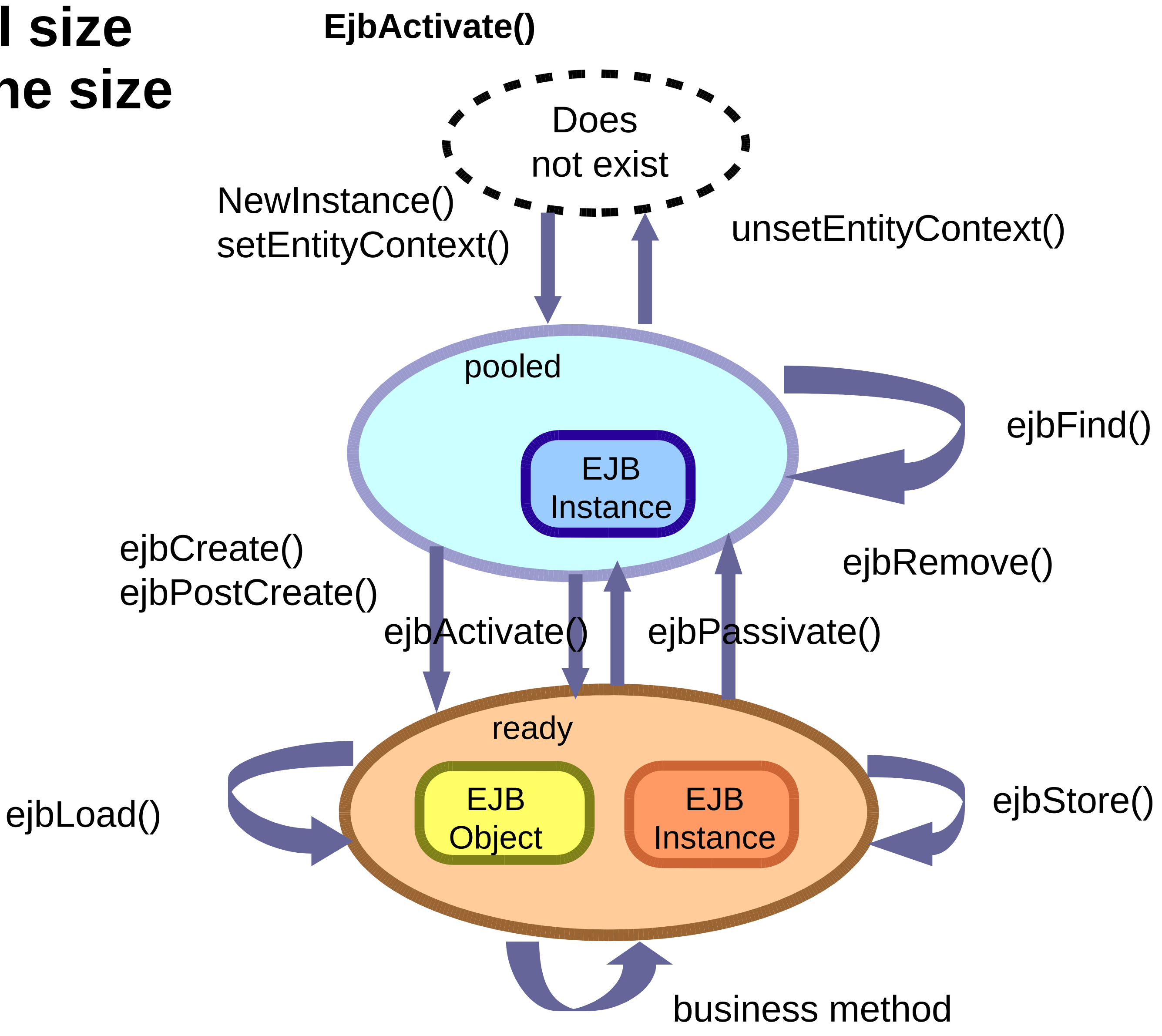
- Execute threads that requests run on
- JDBC connection pools
- JDBC prepared statement cache size
- Correct settings depend on application

Tune Key Container Parameters

- Session timeouts
- Stateful session bean and entity bean cache
  - Cache = EJB instances with state
- Stateless session and entity bean pools
  - Pool = EJB instances with no assigned state
- Transaction isolation level

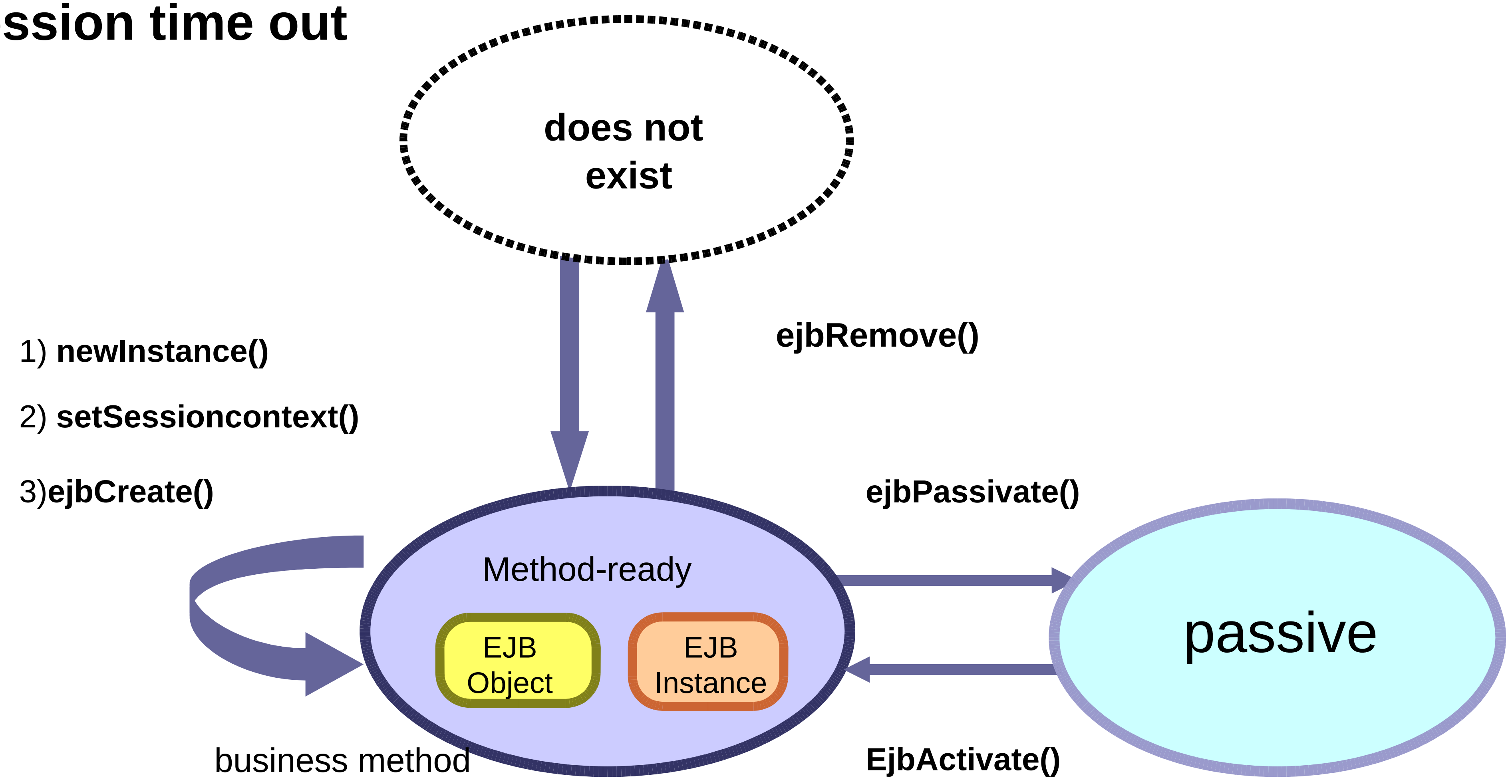
Entity Bean

Bean pool size
Bean cache size



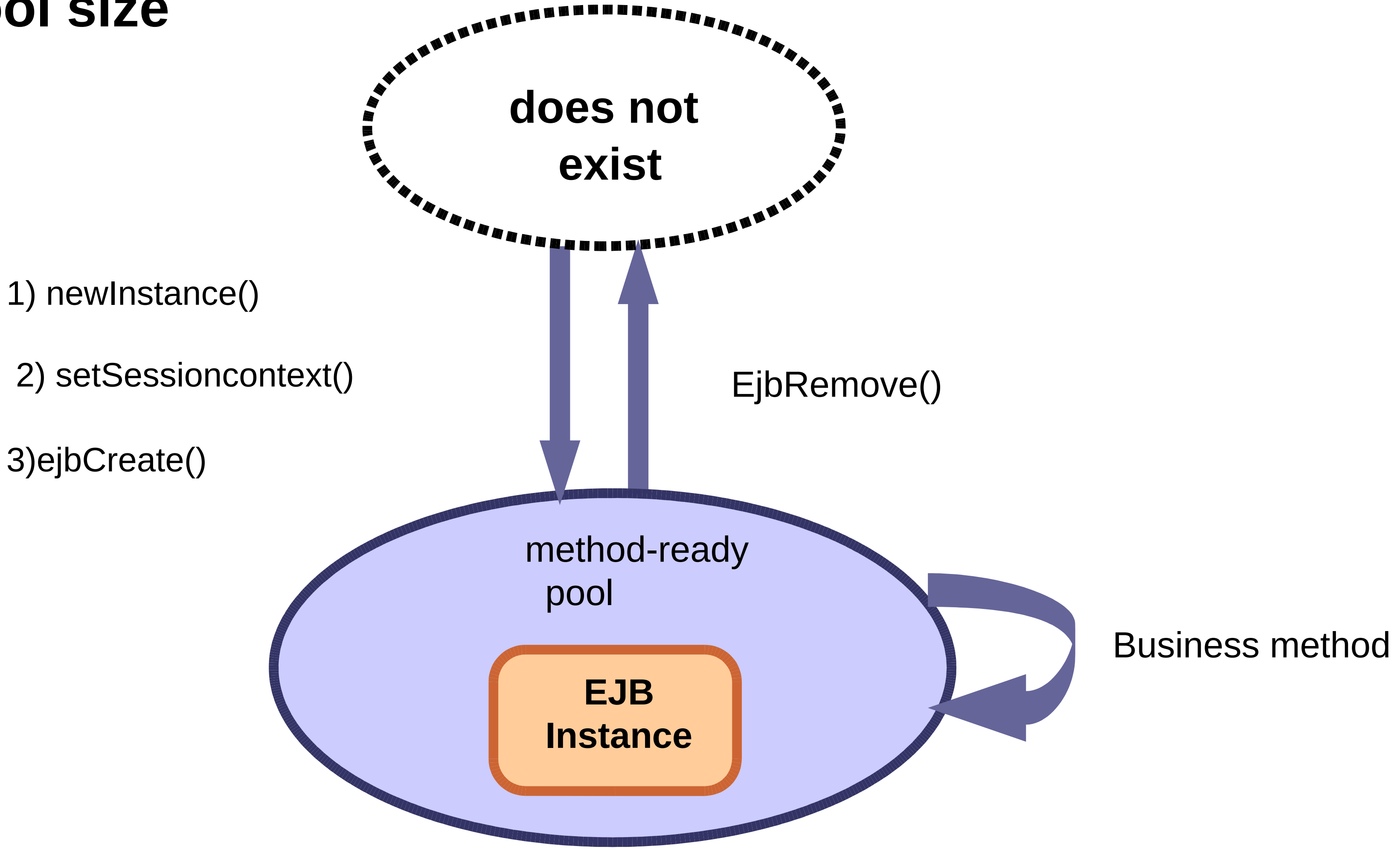
Stateful Session Bean

Bean cache size
Session time out



Stateless Session Bean and Message Driven Bean

Bean pool size



Manage Expensive Resources

- Cache "EJB homes"
- Cache data sources
- Minimize use of HTTP sessions
- Release database connections
- Remove unused stateful session beans
- Use local Interfaces

Design Patterns can Significantly Help Performance

- Session Facades
- Service Locator
- Value List Handler
- Data Transfer Object

Push your development further

Performance Testing

Performance Tips Summary

- Tips for better performance
  - Tune app server and infrastructure
  - Leverage proven design patterns
    - Design coarse grain EJB interfaces: Value Object
  - Reduce JNDI look-ups: service locator
  - Use session bean wrappers: session facade
- Database access
  - Use JDBC for:
    - Batch loading: session bean or message bean
    - Large result sets: value list handler
  - Use CMP rather than BMP Entity Beans
  - Use right isolation level and database transactional control (locking)

JMS, Messaging, Web Services J2EE Best Practices

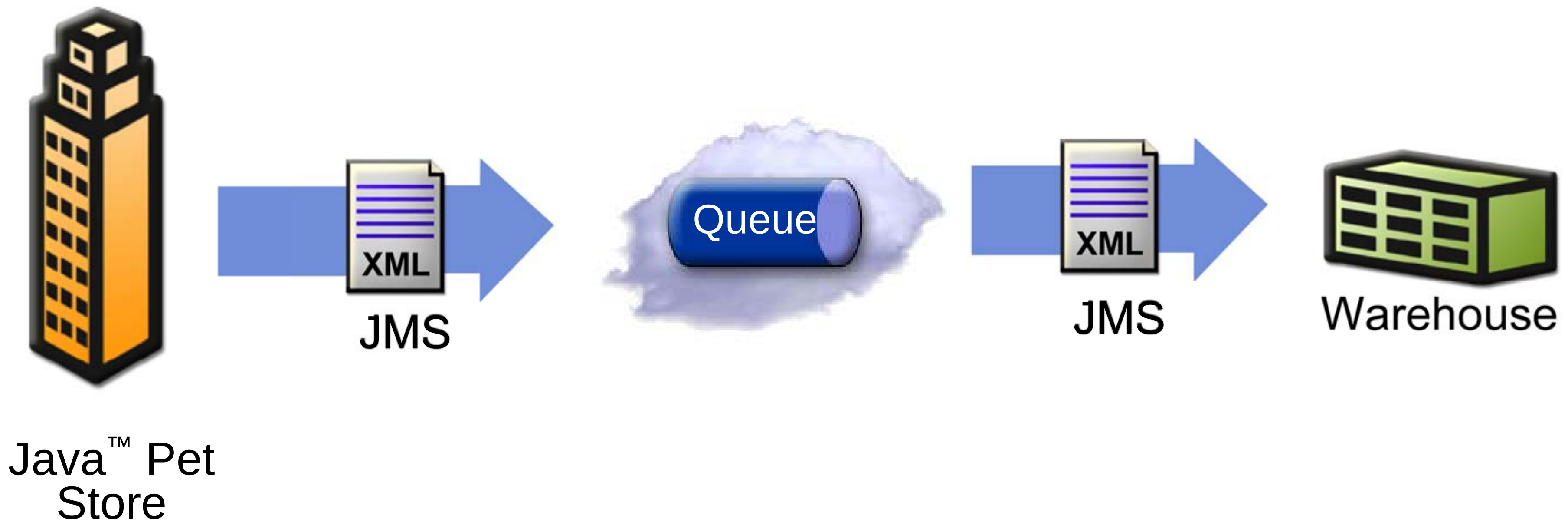
Push your development further

JMS

- Use JMS for **loosely coupled** applications that need **reliable, scalable** document oriented message exchange
- Including Message-Driven Beans and JMS in your J2EE™ application can greatly increase performance, robustness, and efficiency

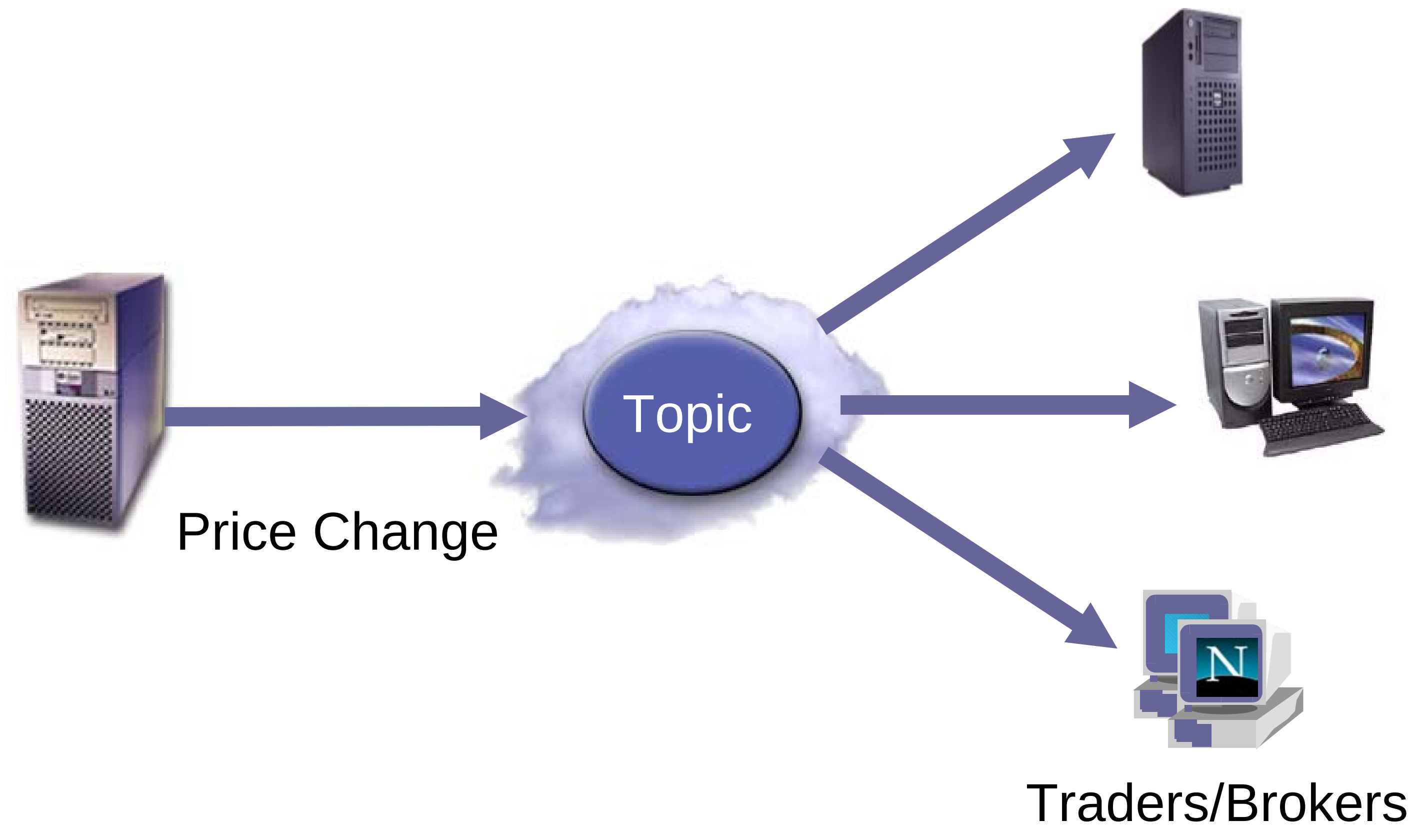
Point-to-Point

Example: Order and Fulfillment



Publish and Subscribe

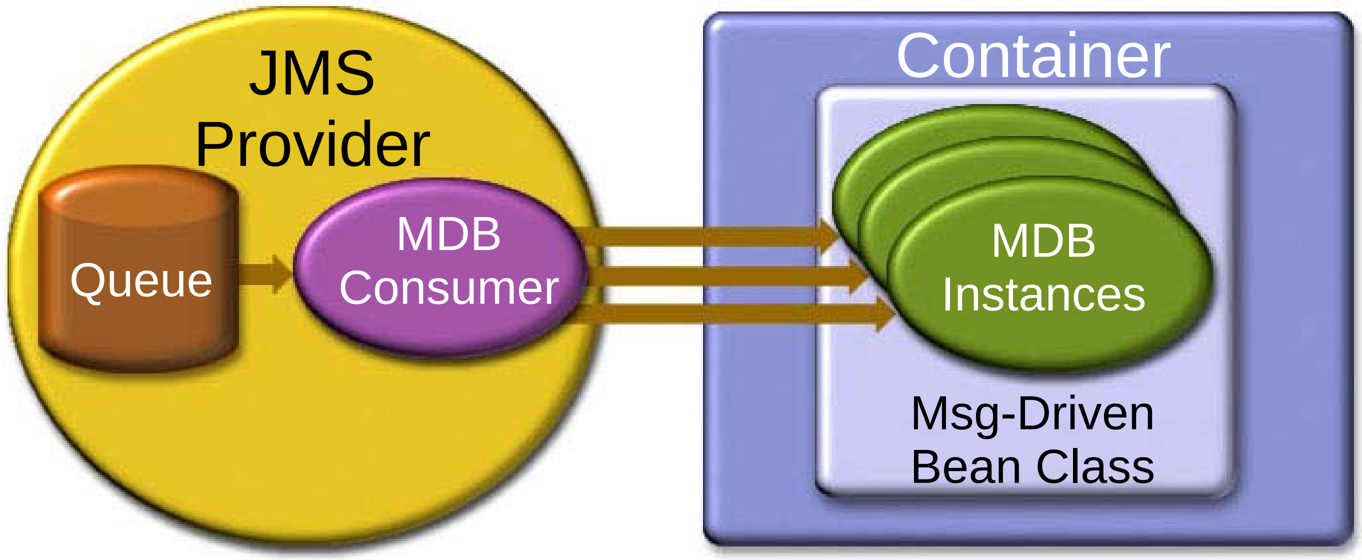
Example: Stock Price Changes



Message-Driven Bean

Concurrent Asynchronous Processing

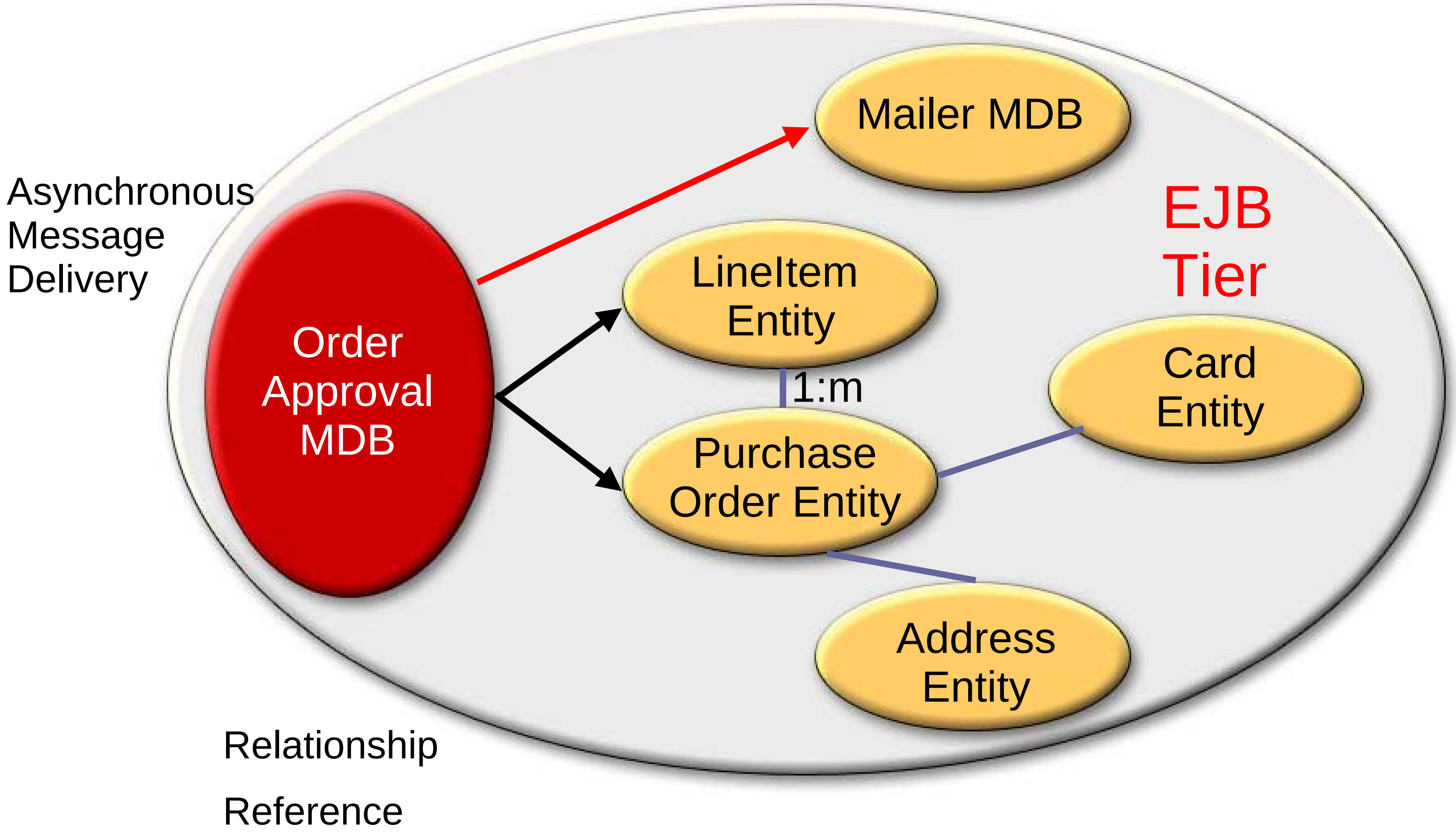
- High Scalability, High Throughput
  - MDB instances are pooled by the container
  - Allow for asynchronous concurrent message consumption from the same destination



Use JMS for

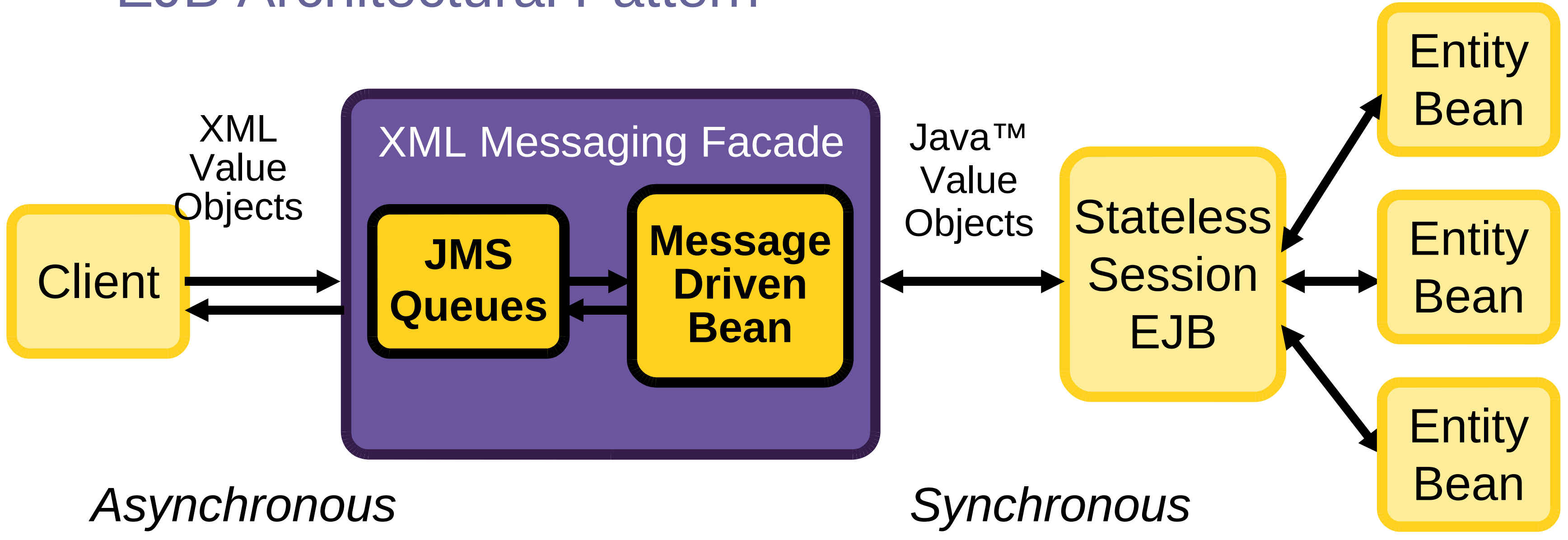
- Asynchronous Interaction
- Concurrent processing
- Broadcasting events (messages)
- Reliable messaging
- Messaging with Transaction Support
- Scalability
- Loose Coupling
- Batch processing

MDB Facade Pattern



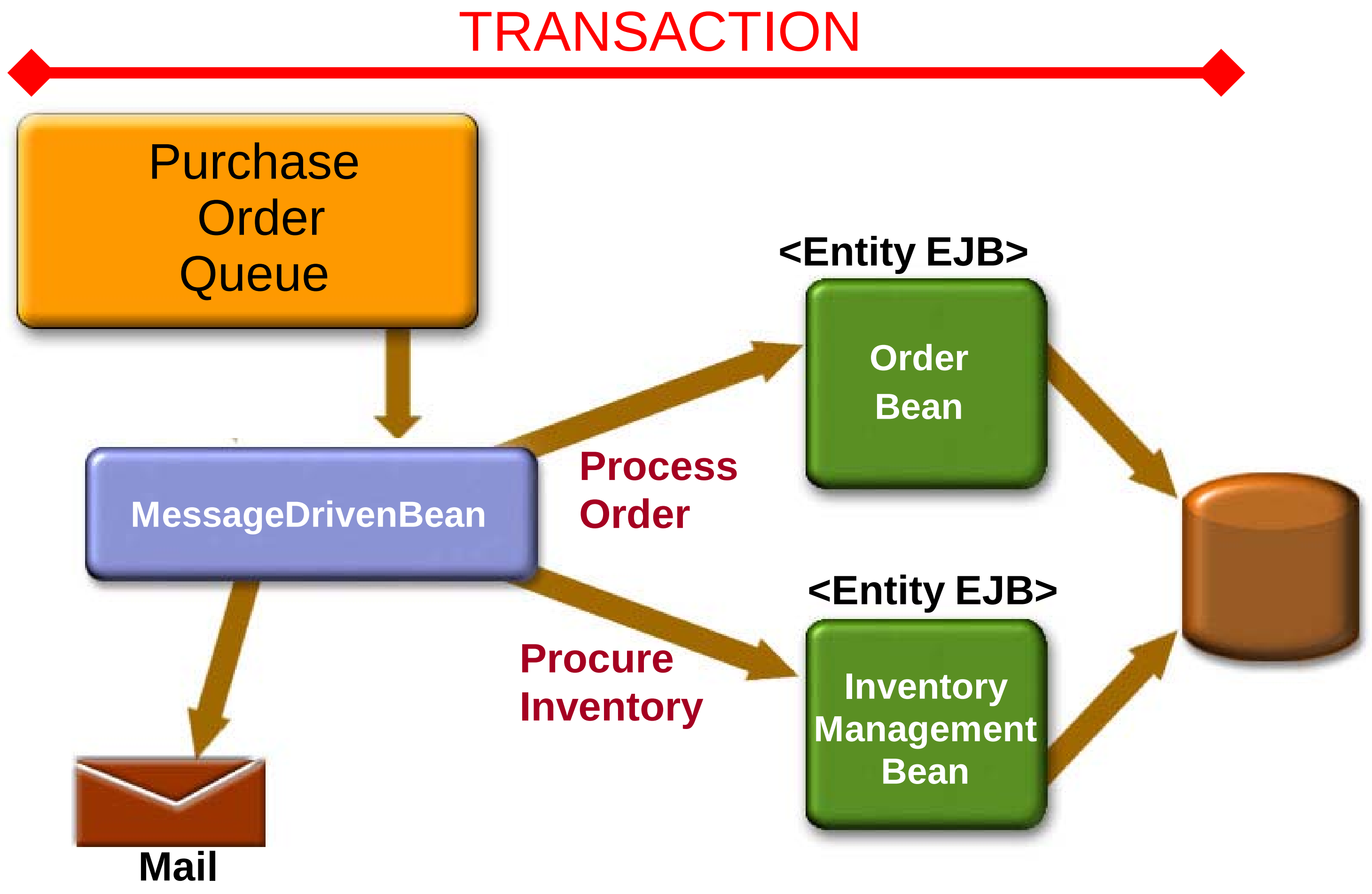
XML Message Facade

EJB Architectural Pattern



Example implementation of XML interaction model on top of Java™ interaction model

MDBs Container Managed Transactions

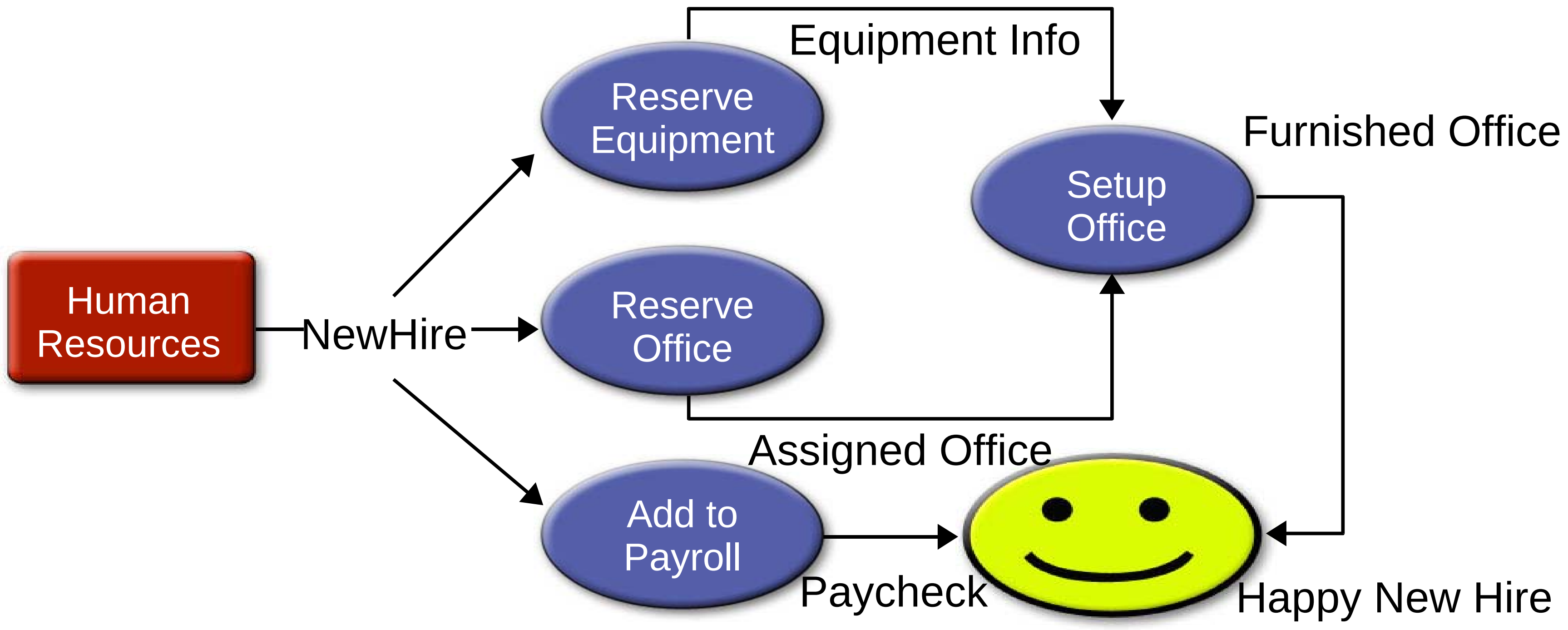


Use JMS for Event-Driven Interactions

- Designing for event-driven interactions:
  - Not interface-driven
  - Loose coupling among participants
  - Asynchronous communication model
  - Reliable communication or many to many communication model
- Serves as an asynchronous facade to a subsystem or an application

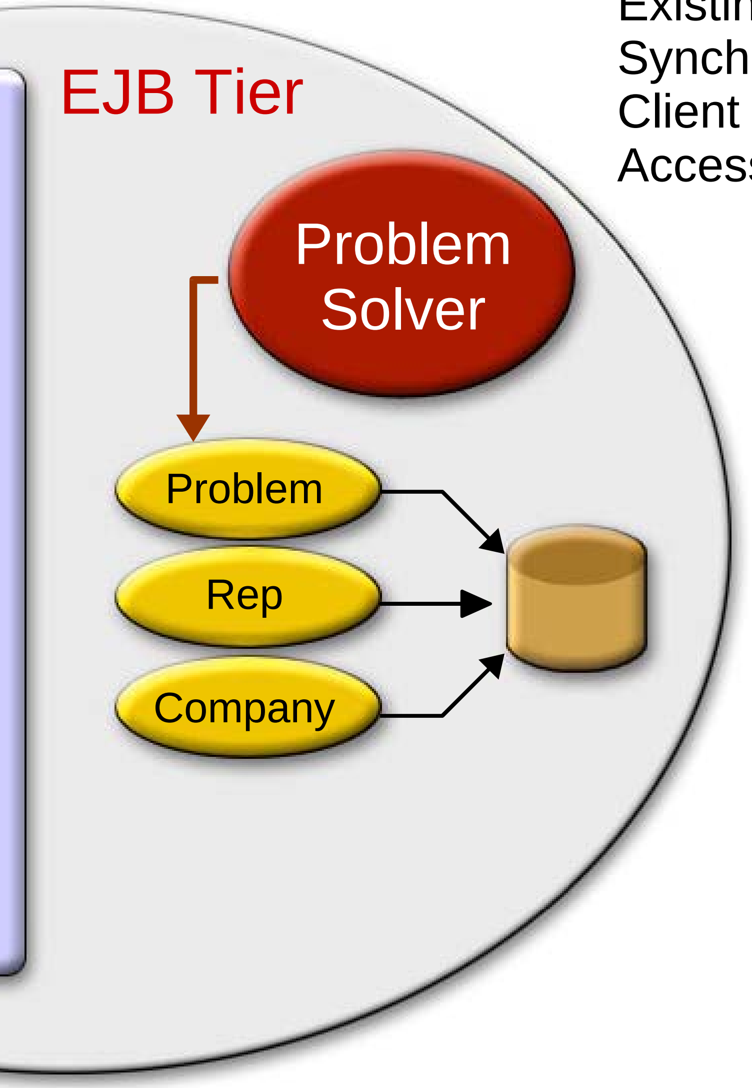
MDB Join Pattern

- Join pattern
  - MDB collects different messages
  - Stores to same set of entity beans



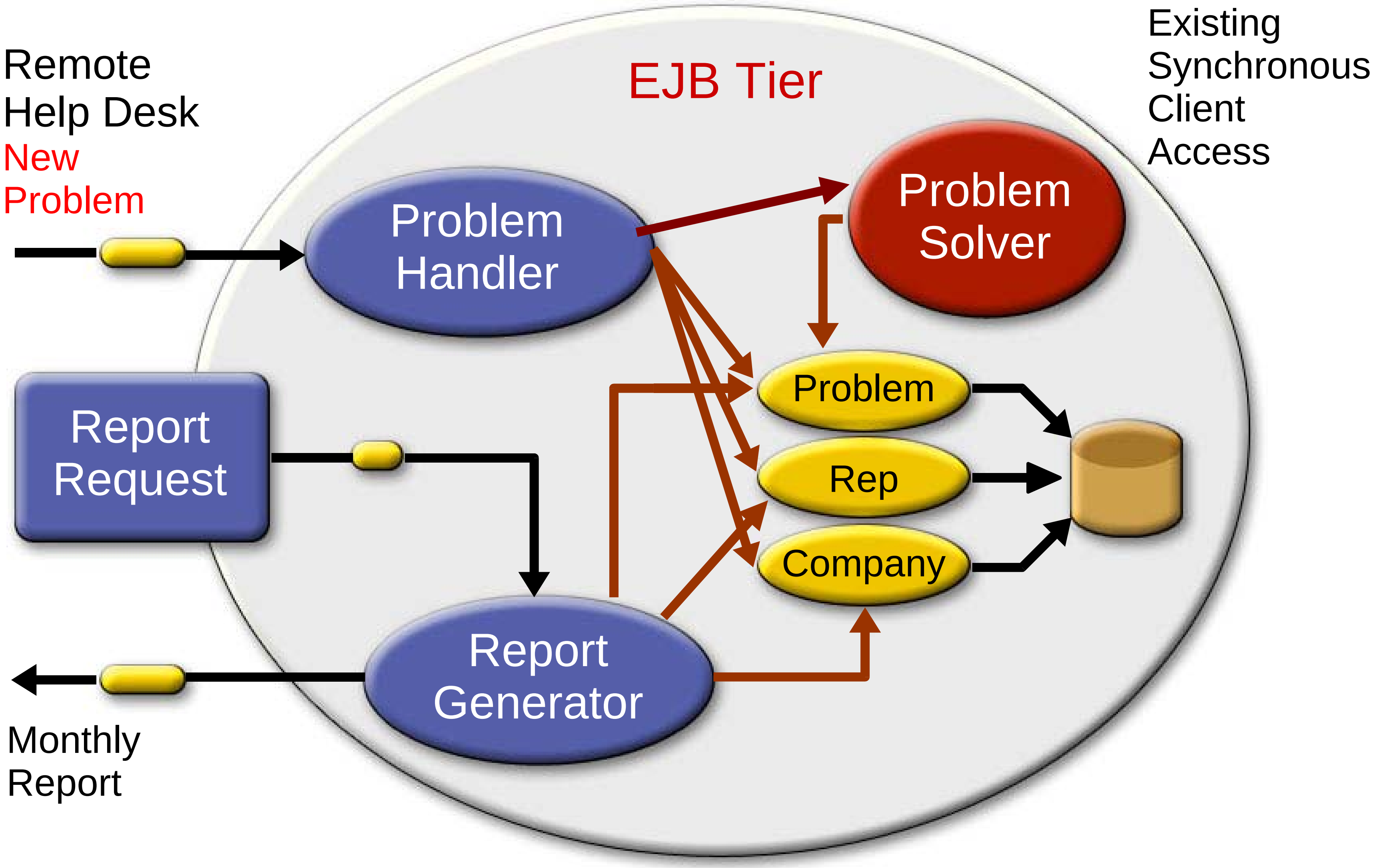
Example Problem Tracking

- A Session EJB provides the business logic for the Customers or help desk to synchronously enter Problem data, or search for similar problems
- Entity EJBs provide the business logic for accessing/updating the domain Data objects



Existing Synchronous Client Access

Extension Solution Internal View



JMS Do's & Don'ts

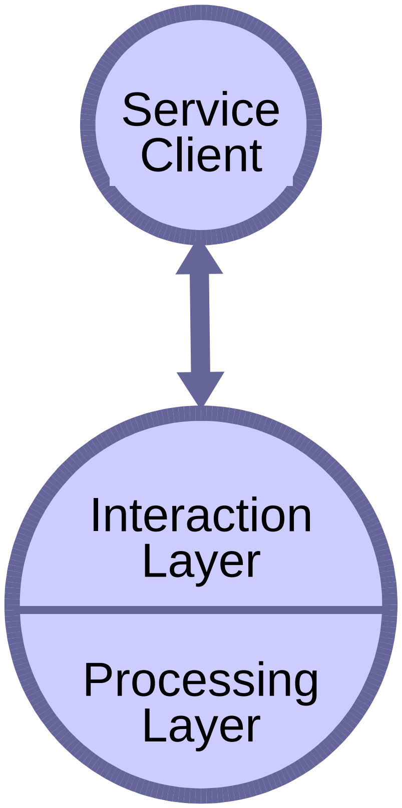
- Do watch message size
- Do only use XML when necessary (lower performance)
- Do only use reliable messaging when necessary (lower performance)

Tips for XML messaging

- Inside your Application don't overuse XML
  - Parsing and processing can cause a performance hit
  - Use XML for messages between systems…
  - Within a system just use data objects
- Convert the data from XML into a Java™ class as soon as possible
  - Can use JAXB for this
- Use XML mainly for transmission

Key Web Service Guidelines

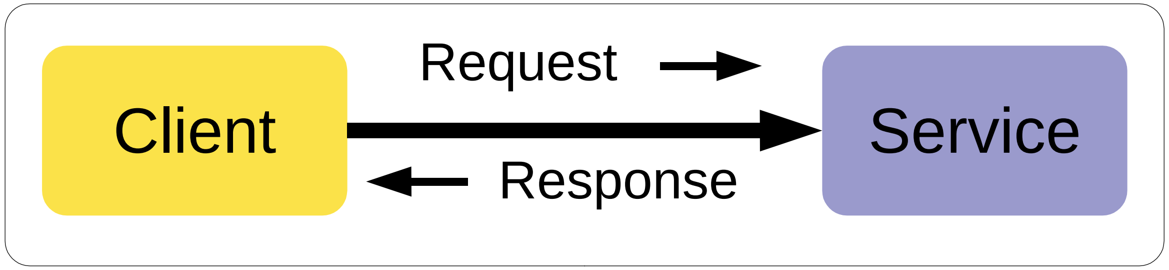
- Structure application in two layers
  - Interaction layer and processing layer
- Interaction layer
  - Interface to clients
  - Receive requests and perform required translations and transformations
  - Delegate request to processing layer for processing
  - Respond to clients



Layered View

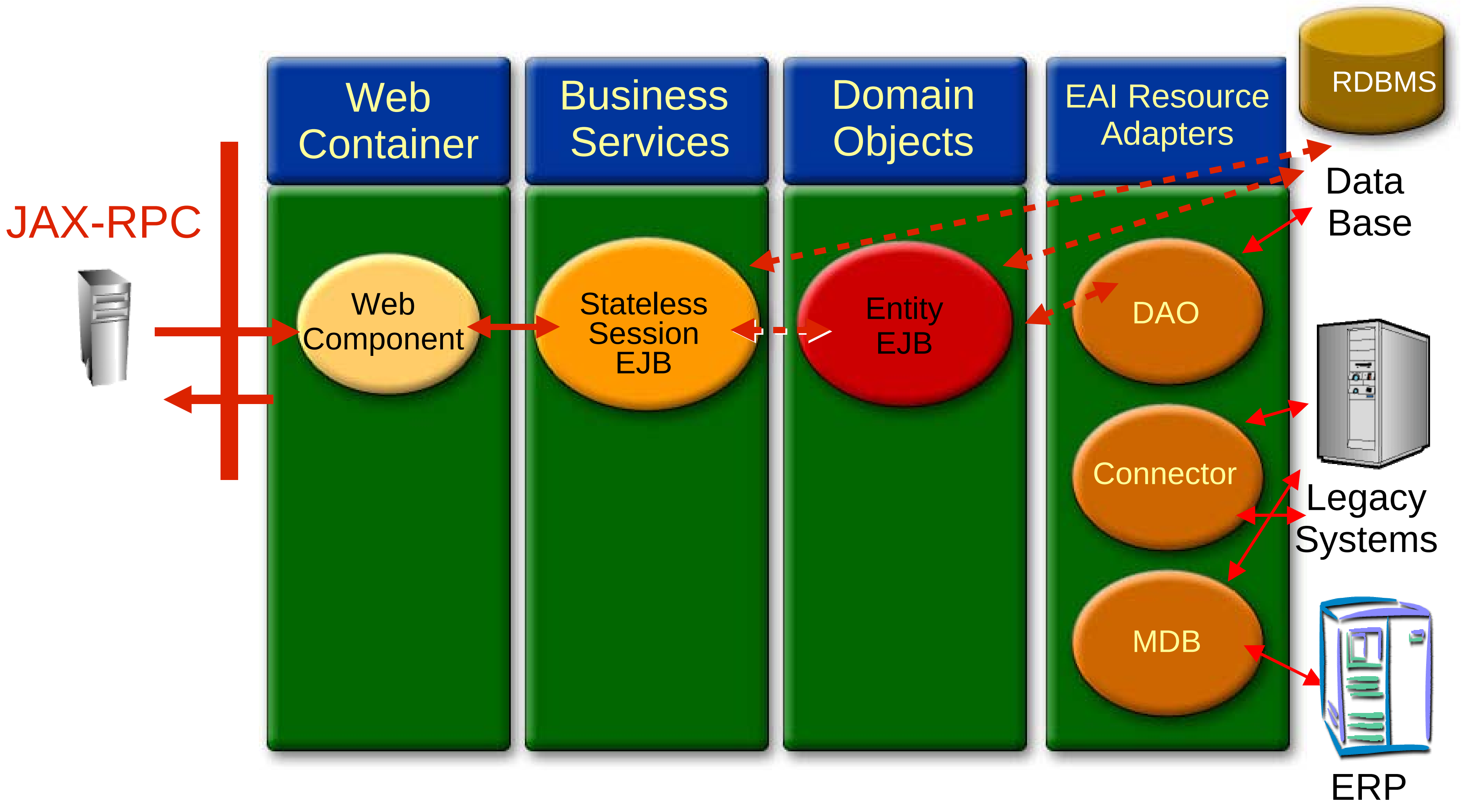
- Processing layer
  - Process request
  - Apply business logic
  - Integrate with EIS
  - Interact with peers
- Layered view helps to:
  - Clearly divide responsibilities
  - Decouple business logic completely
  - Expose a web service interface to existing business logic

RPC – Simple Client/Service

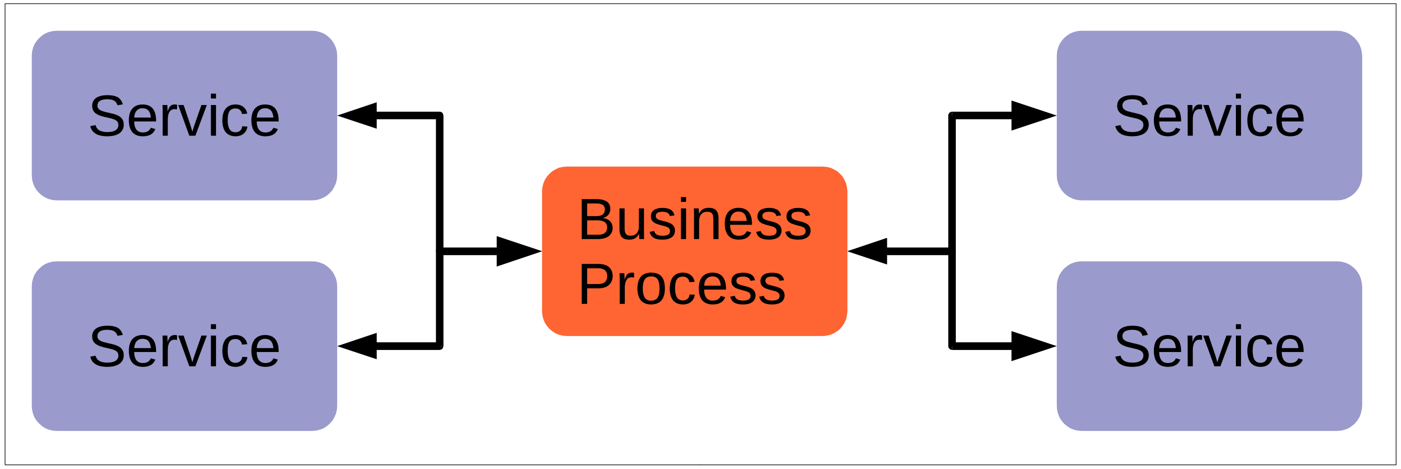


- Common approaches:
  - SOAP using JAX-RPC (WSDL)
- Features:
  - Stateless and conversation-less
- Industry examples:
  - Amazon, Fedex, eBay, credit check, get Weather, get Stock price

RPC Style Web Services

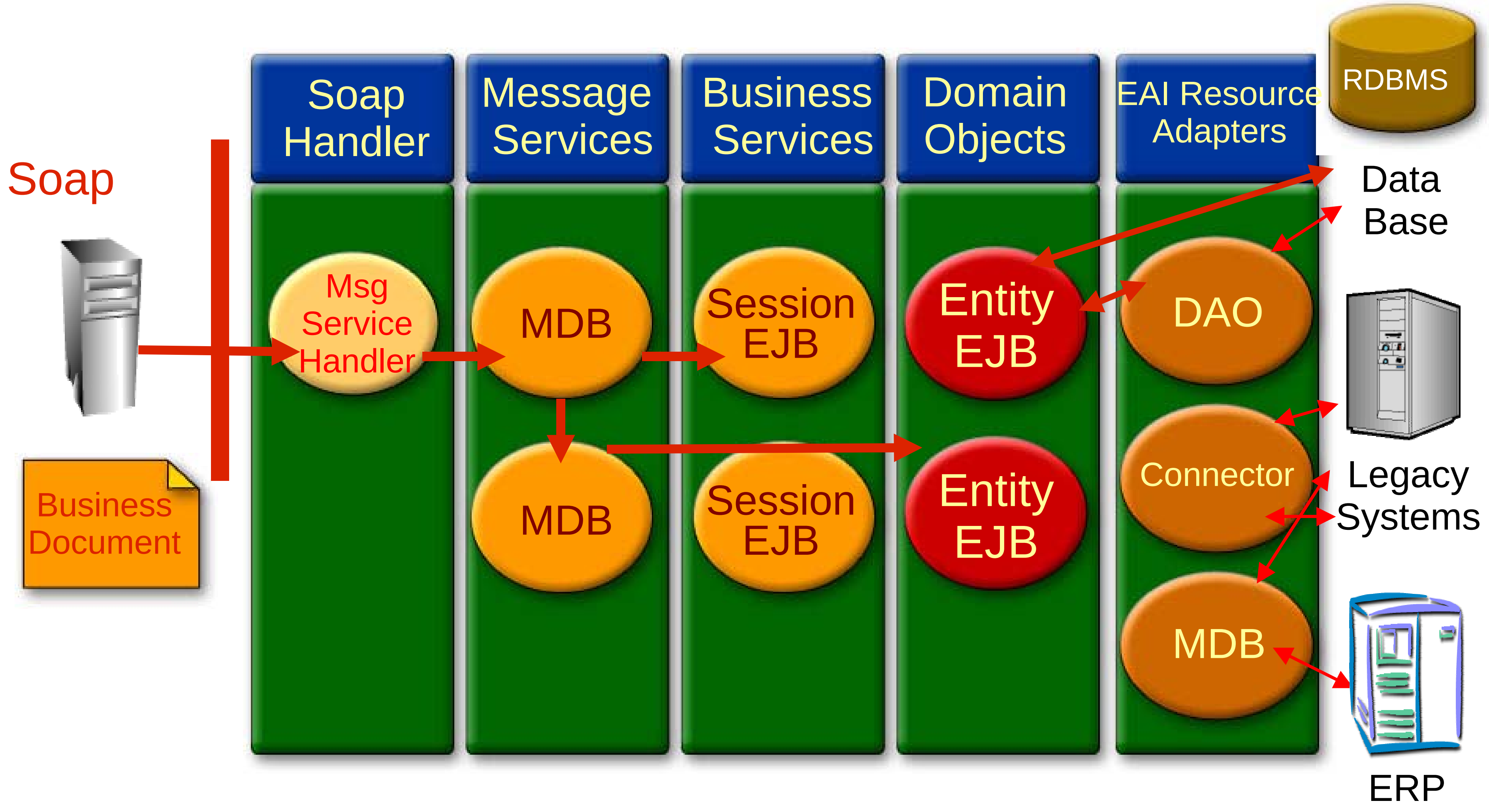


Collaborative B2B Web Services

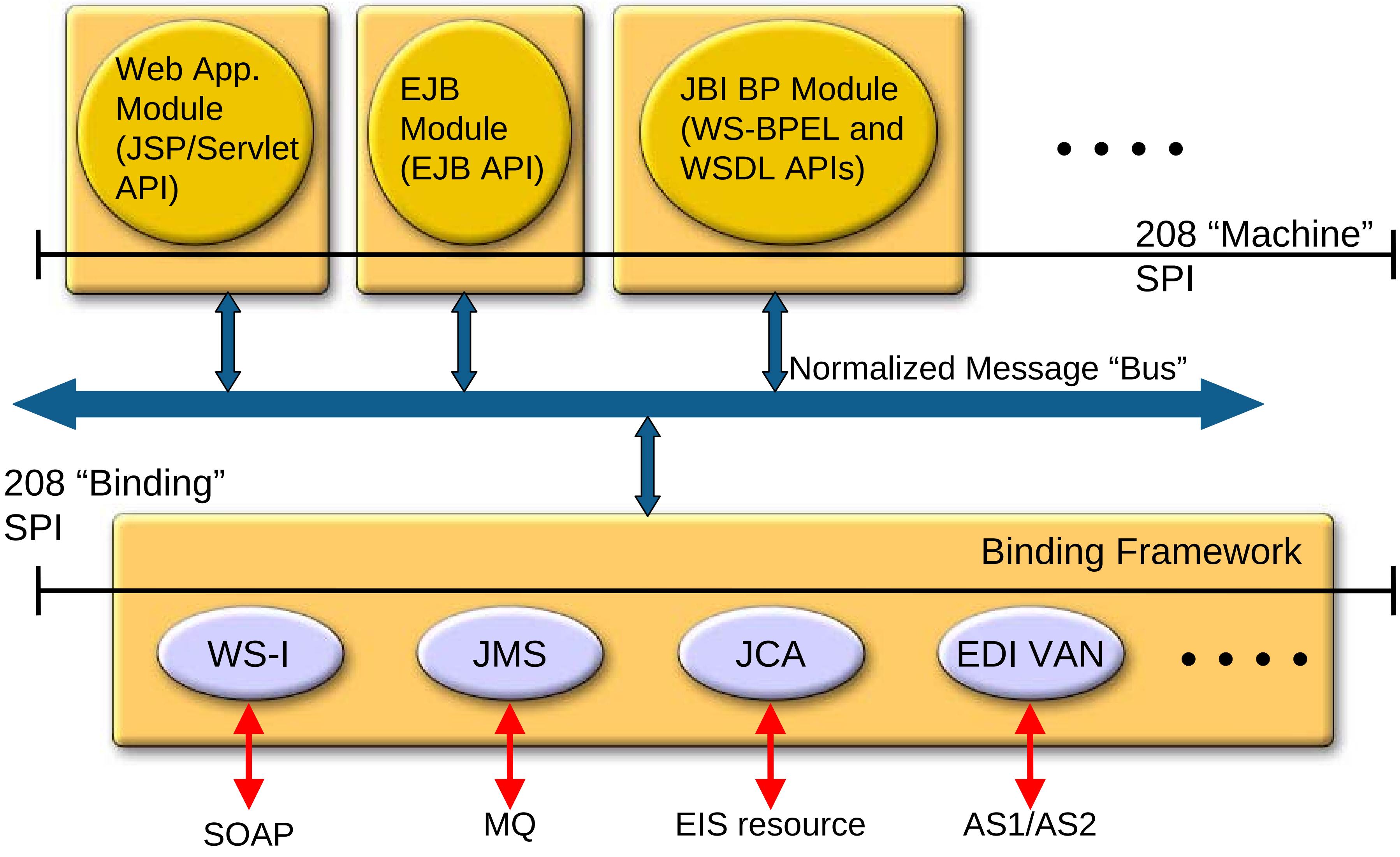


- Features:
  - Orchestration, Choreography, Collaboration
  - Business process defines Message sequence
  - Asynchronous document exchange
- Industry examples:
  - OTA, Sabre, GM

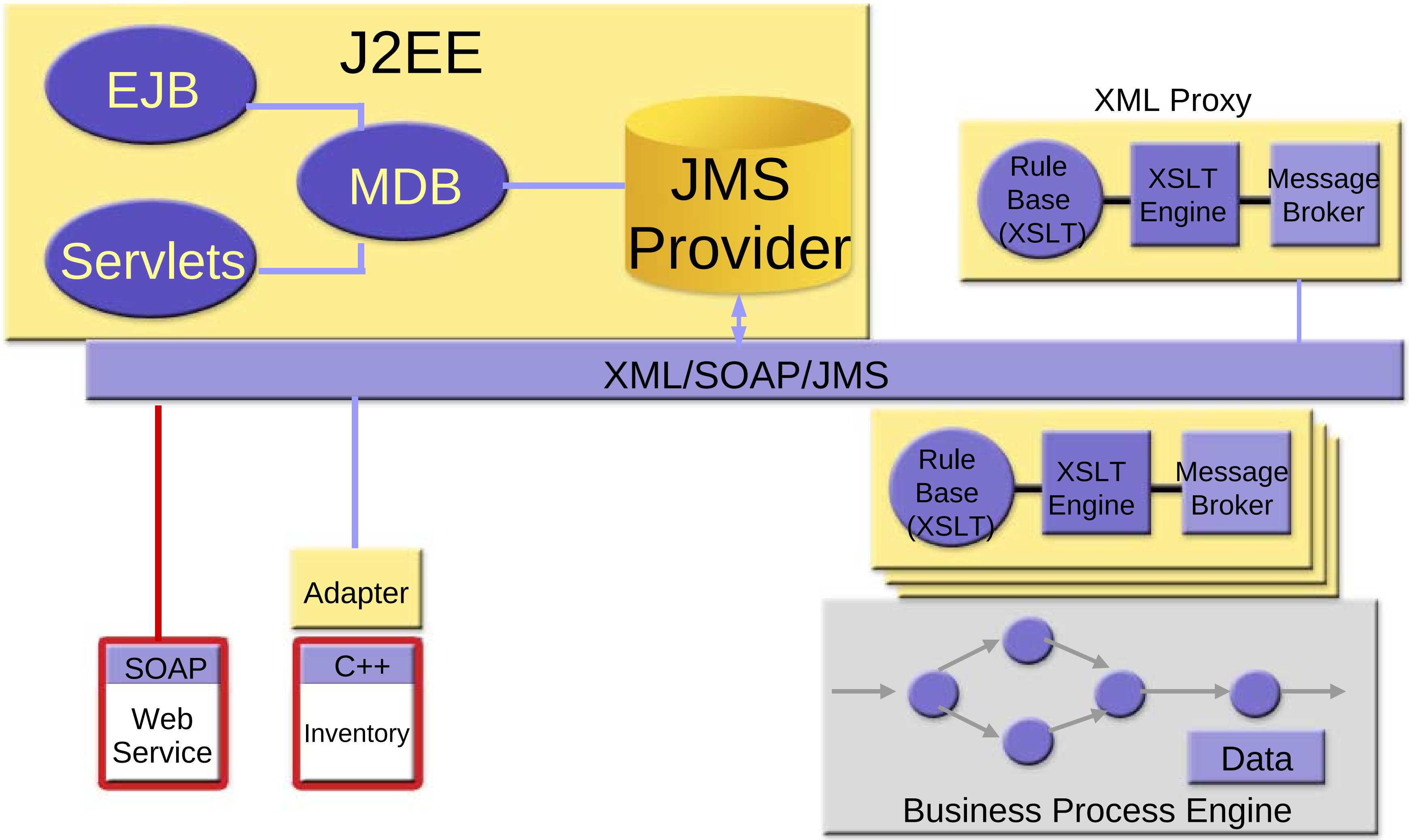
Document Style Web Services



JBI and J2EE


Web App. Module (JSP/Servlet API)
EJB Module (EJB API)
JBI BP Module (WS-BPEL and WSDL APIs)
• • • •
208 "Machine" SPI
Normalized Message "Bus"
208 "Binding" SPI
Binding Framework
WS-I
JMS
JCA
EDI VAN
• • • •
SOAP
MQ
EIS resource
AS1/AS2

Sun ONE BPM, Messaging & J2EE



Carol McDonald
Technology Evangelist

carol.mcdonald@sun.com

Sun™ Tech Days