

INTERSHOP
Creating the Digital Economy™

Designing High Performance EJB Client-Server Applications (1)





Ulrike Müller
Chief Software Architect

INTERSHOP ENFINITY™



Agenda

- **Design Issues (1st half, Ulrike Müller)**
  - Implication of App Server on the design
  - Usage of different EJB types
  - Design patterns for the use of EJBs
  - Example
- **Implementation Issues (2nd half, Thomas Wieger)**
  - Performance Implementation tactics
  - Performance and the development cycle
  - Benchmarking & Tools
  - Performance & Deployment



- **INTERSHOP uses J2EE application server based on different 3rd party components**
  - Sun Java Web Server (Web Tier)
  - Persistence PowerTier (EJB Tier)
  - Visibroker (CORBA)
- **INTERSHOP has more than three years of experience in building a complex EJB system**
- **Presentation is not specific to INTERSHOP system**
- **Presentation is general and can easily used with different app servers and different systems**
- **Focus on Web based applications**



Introduction

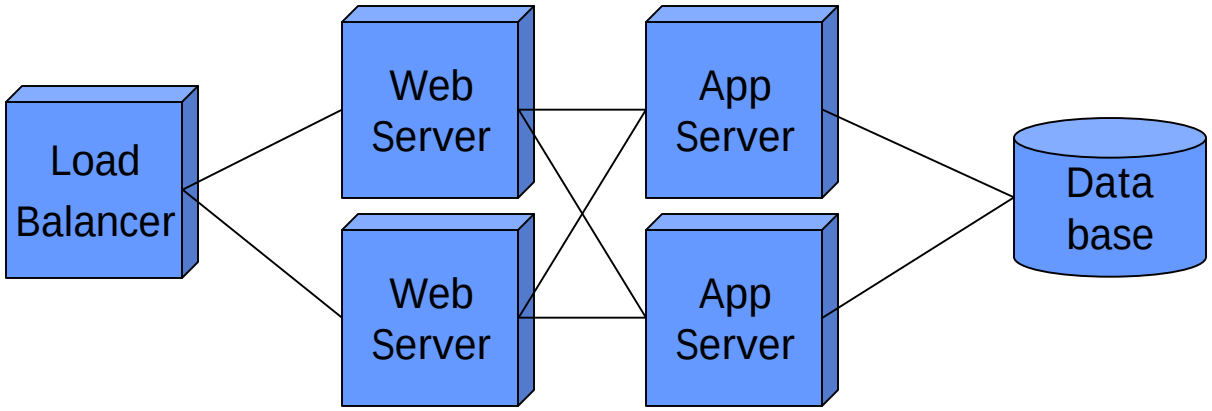


Todays Requirements

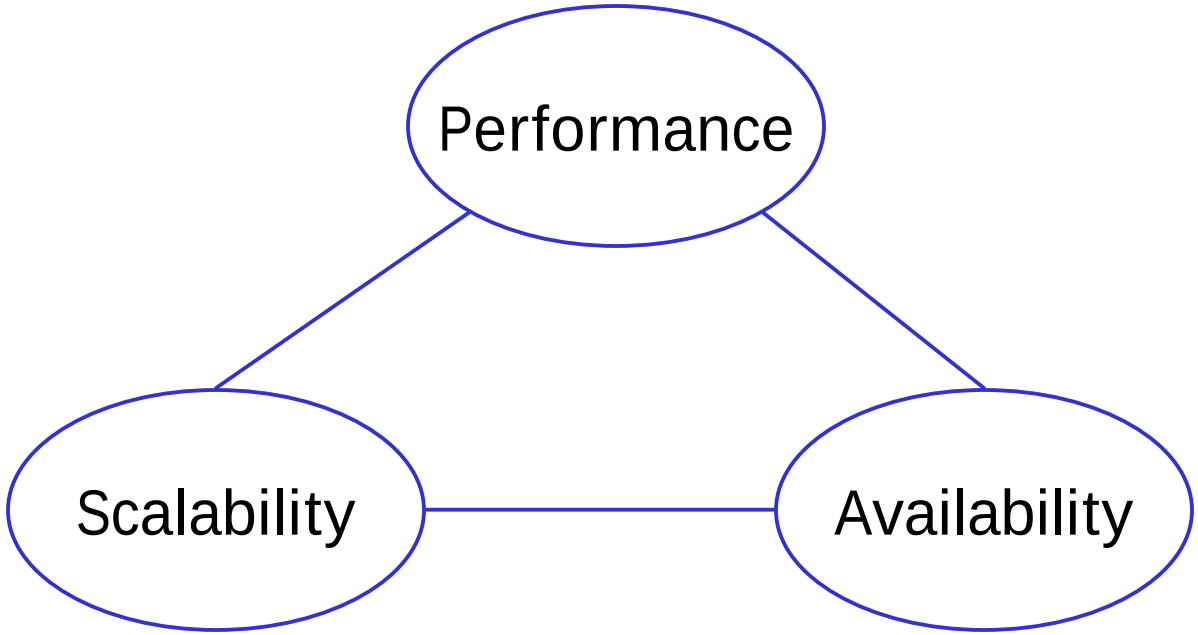
- **Page views: 15 Mio / day**
- **Data volume: 35 Gbyte / day**
- **Orders: 60.000 / day**
- **Number of products: 2 Mio.**
- **Number of customers: 2 Mio.**
- **Number of concurrent session: 50.000**

Typical Deployment Environment

- **Multiple tiers**
  - Load balancer
  - Multiple web servers
  - Multiple application servers
  - Clustered database server

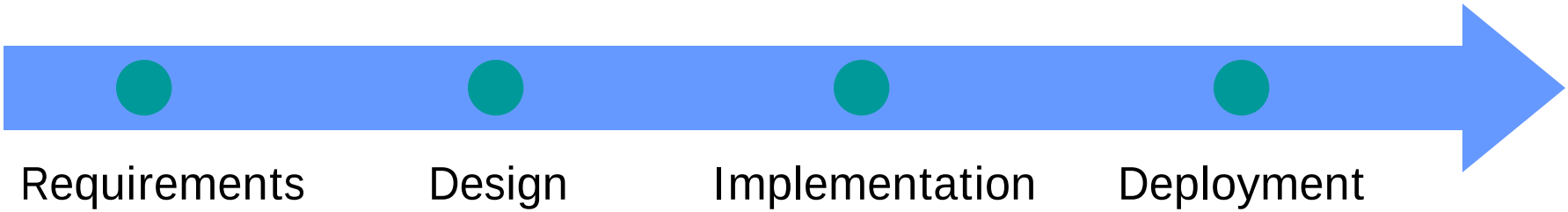


Requirements for Architecture



- **All three requirements essential for client-server system**
- **Focus on performance and scalability**

Performance in all development phases

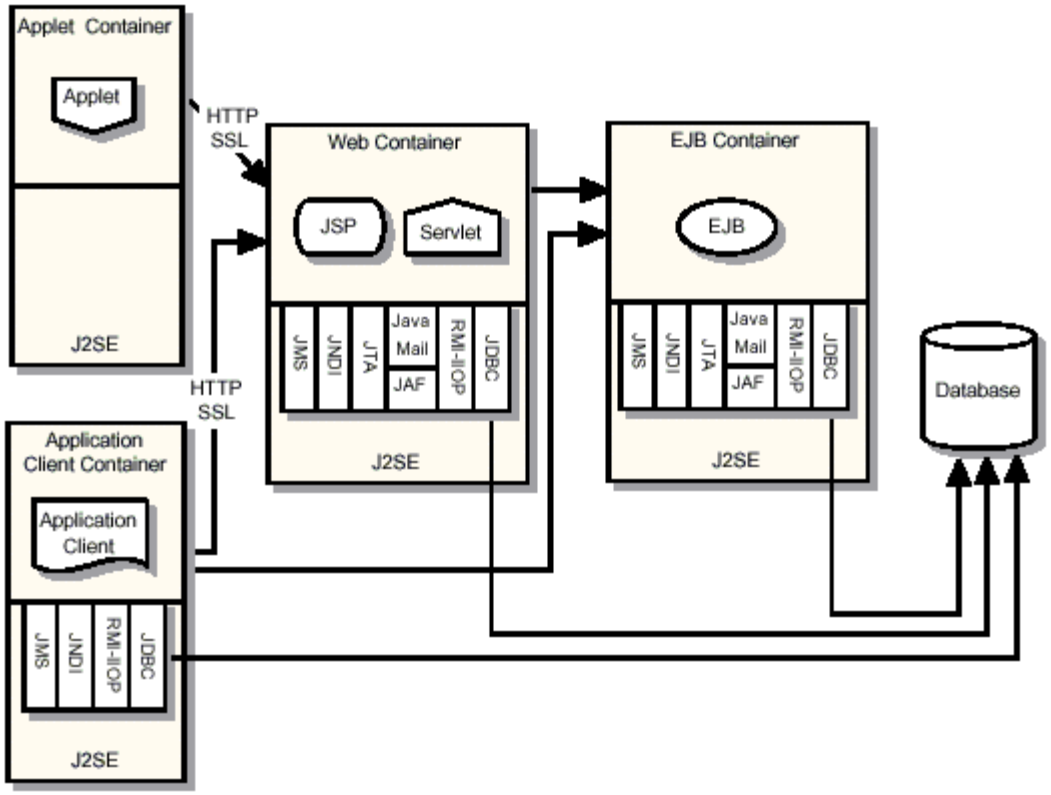


- **Performance goal has impact on all phases**



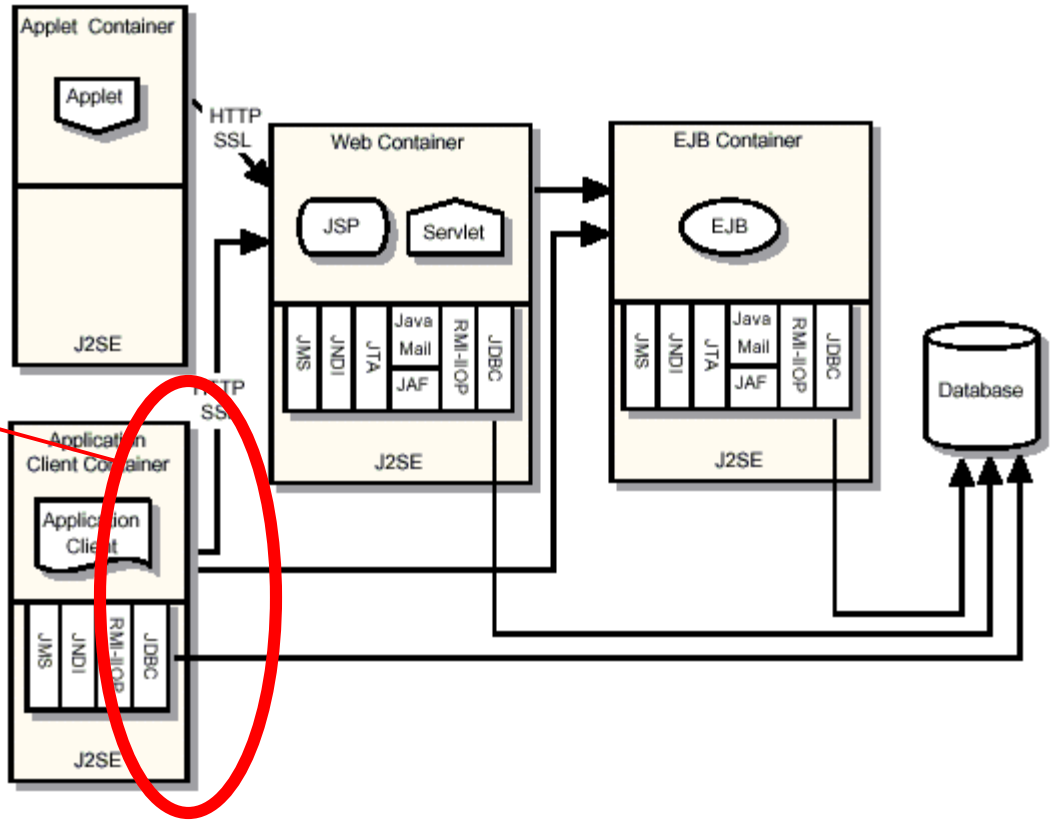
Performance Strategy

J2EE Architecture



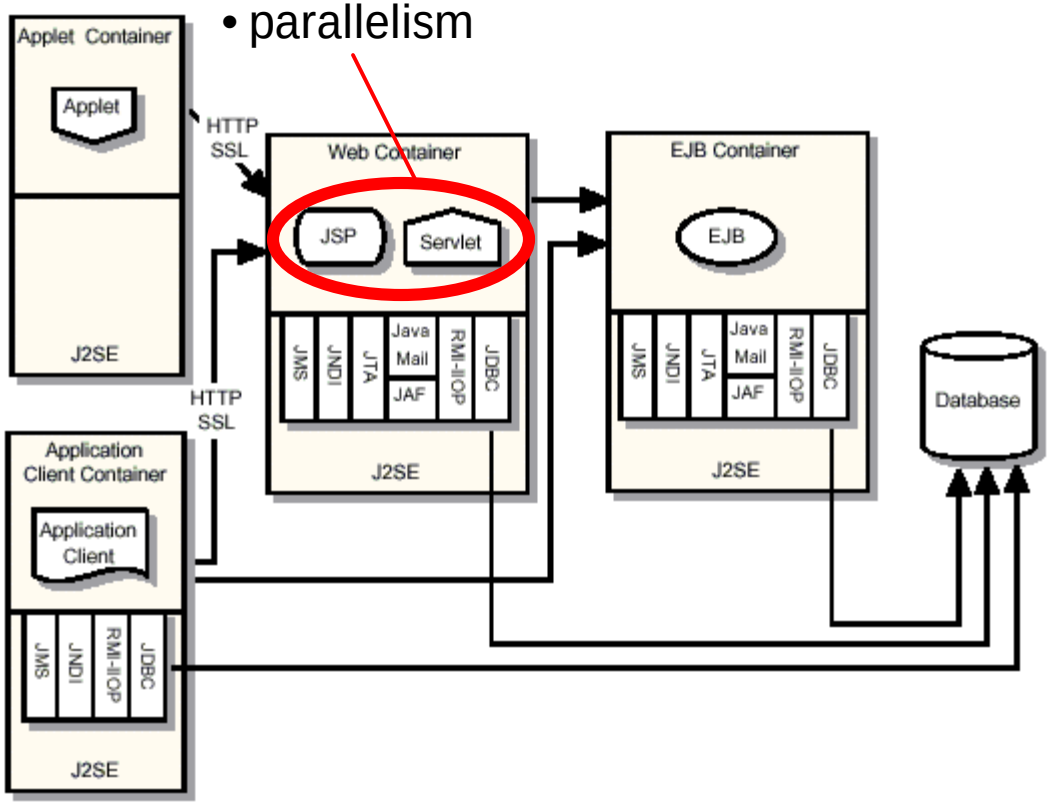
J2EE Architecture

Remote communication from thick client

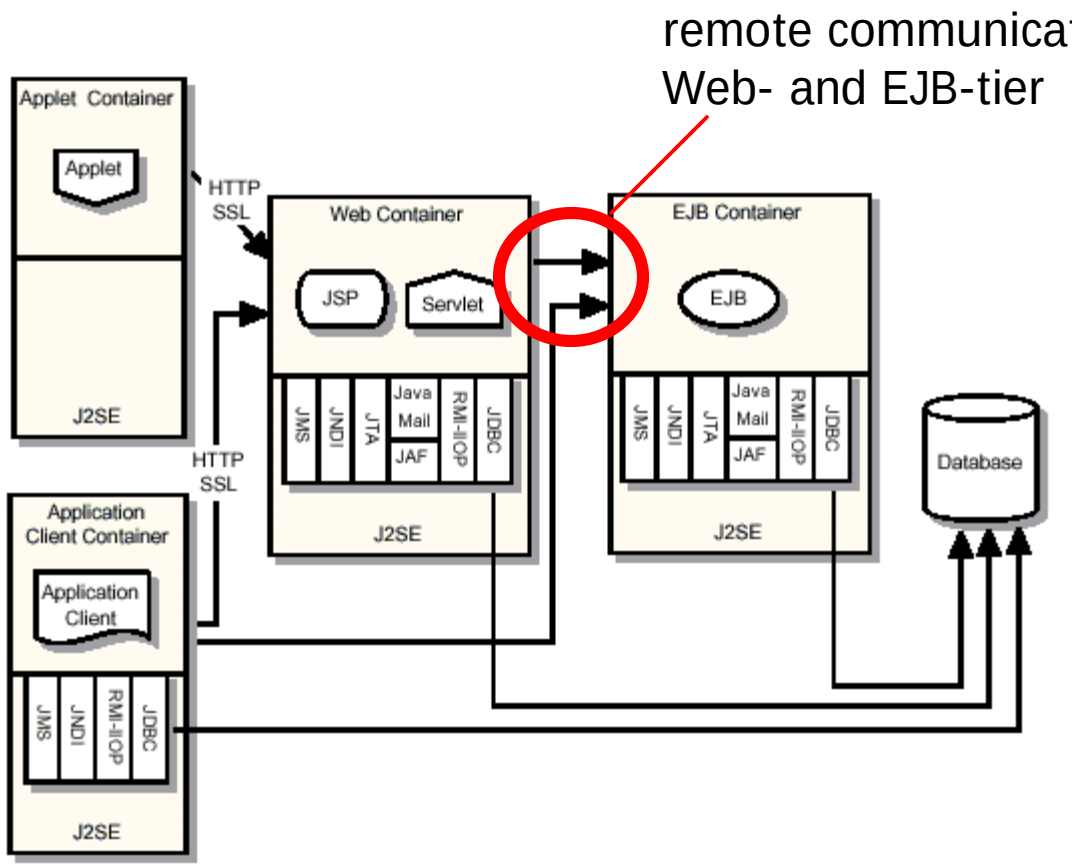


J2EE Architecture

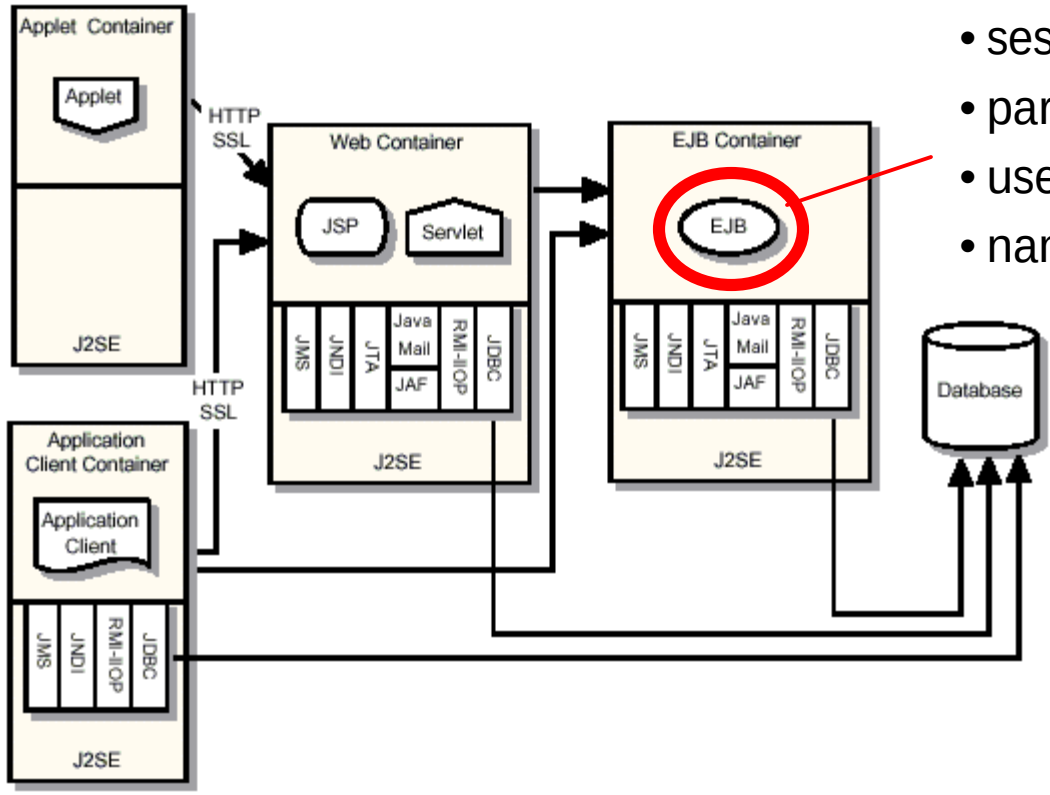
- session fail-over
- parallelism



J2EE Architecture

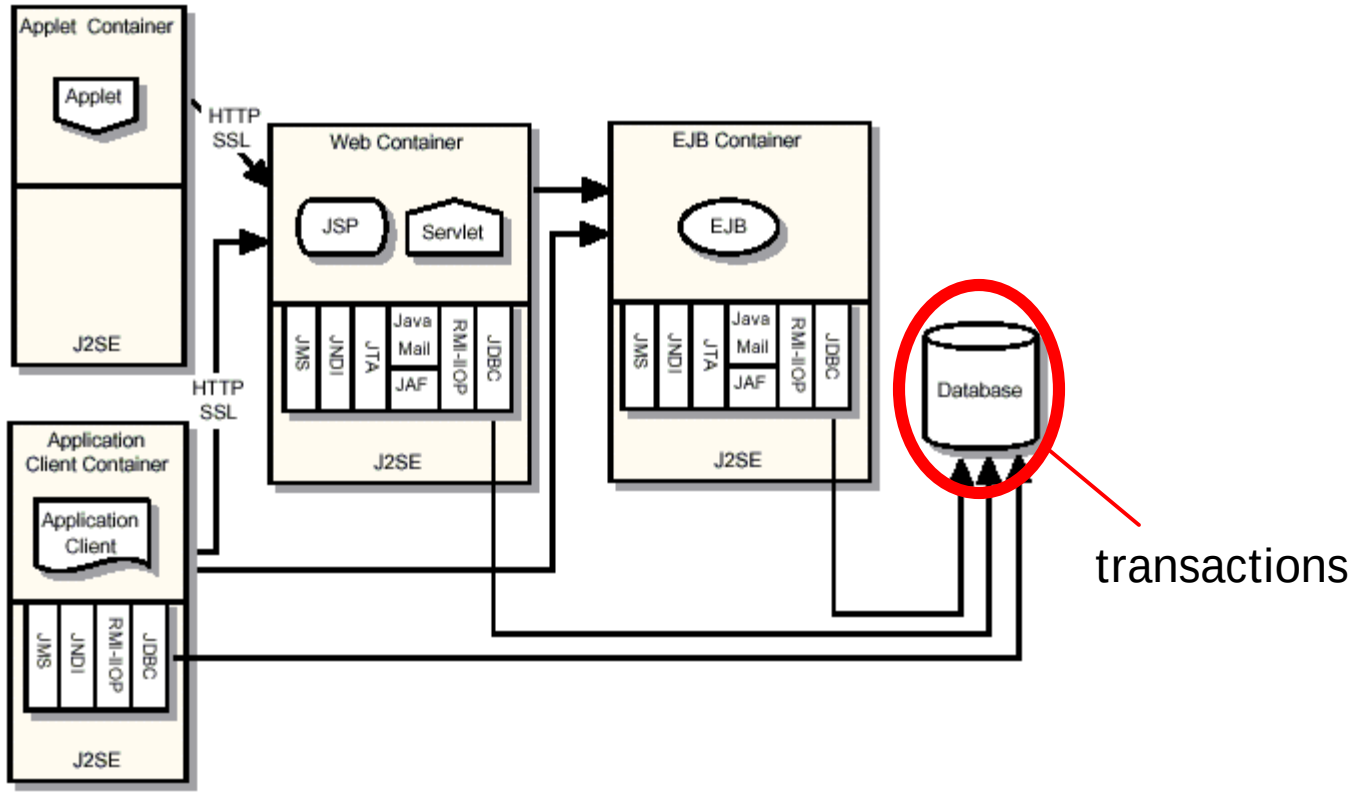


J2EE Architecture



- session fail-over
- parallelism
- use of Entity Beans
- naming server

J2EE Architecture





Overall Strategy

- **Handle critical resources efficient**
  - Network
  - Database transactions
- **Improve parallelism**
  - Parallelism important for multi-processor system
  - More focus on parallelism than on speed of single request
- **Distributed environment and clustering**
  - Changing an application to run on a distributed environment at a later time is hard to achieve



Technical Considerations

App Server Architecture

- Single vs. multiple Java VMs
- Session handling
- Remote communication
- Optimized local Bean-Bean call
- Implementation of naming service
- Support for CMP Entity Beans
- Features for persistent mapping of CMP Entity Beans

App Server Architecture

- **VMs**
  - Web-Tier and EJB-Tier in same VMs
    - Supported by most App Servers
    - Communication between tiers no performance problem
  - Web-Tier and EJB-Tier in different VMs
    - Supported as deployment option in same app servers
    - Communication between tiers get relevant

- **Note: INTERSHOP enfinity uses one VM for both tiers**

App Server Architecture

- **Session Handling**
  - Support for session fail-over in clusters environment
    - Session must be written to backing storage
    - Can be a performance problem
    - Different technologies for backing storage (file system, database)
  - HTTPSession
    - Session distribution in clustered environment supported by most App Servers
    - Session stickiness in load balanced scenario
  - Stateful Session EJB
    - Storing state and distribution on clustered environment supported by some App Servers

Remote Communication

- **Remote call very slow compared to local call**
- **Reduce number of necessary remote calls**
- **Tests have shown, that number of calls are more relevant than the size of data**
- **Result: Prefer to transfer more data per call vs. more small calls**

- **Optimized local Bean-Bean calls**

- **Note: Main design goal in INTERSHOP enfinity**



Naming Service

- **Naming Service is an essential concept within an EJB app server**
- **One naming service in the network might cause significant network traffic**
- **Replicated naming service requires slow backend storage**
- **Cache possible, because content of naming service changes rarely in running server**
- **What to do?**
  - Check your app server, how naming service works
  - Cache lookup within method and across methods

Database Transactions Optimization

- **Reduce number of database operations**
  - Caching
- **Transactions should be short**
  - Database resources are locked only for a short time
- **Reduce number of operations in transaction**
- **Reduce number of database communications**
  - Buffer database changes up to commit point

- **Note: INTERSHOP enfinity tuned according this concepts**

Database Locking Strategy

- **Pessimistic Locking**
  - Lock may decrease transaction performance
  - Risk of dead locks
- **Optimistic Locking**
  - Better transaction performance
  - Better locking strategy for read caching
  - May cause complete rollback of transaction, while pessimistic locking just causes wait
  - OR-tools usually offer transparent optimistic locking
- **Pessimistic locking can be used sometimes to sync servers in cluster**



Usage of EJBs

Stateless Session Bean

- **Preferred EJB type to implement business logic**
- **No need to replicate state**
- **Easy fail-over and clustering**
- **App server can pool Stateless Session Beans very efficient**
- **Some app server offer reentrant Stateless Session Beans**

- **Note: this is the reason why most app server vendors don t support Stateful Session Bean clustering**



Stateful Session Beans

- **Pooling more complex for App Server**
  - Reserve bean as long as client has reference
    ? huge number of beans to serve huge number of clients
    ... or ...
  - Passivate and activate state
    ? a lot of overhead
  - For clustering and fail-over state must be serialized to some backing storage
    ? performance critical

Stateful Session Bean

- **Very few use cases**
  - Only necessary to store client state on server
  - Thick client can usually store its state in its own memory
  - Thin (HTML) client needs server side store basically for communication state (Which is the next page?, Where am I in the page flow?)
  - HTTP session better suited for storing thin client state
  - Examples: store login related information on the server
  - Examples: „context information“ in order to avoid repeated transfer of big parameters in method calls

Stateful Session Beans

- **Alternative:**
  - If state is Web tier related store it in the Servlet session
  - Check whether bean state can be store in database and move state into entity bean
  - Entity bean can be handled more efficient by app server because no special cluster and fail-over support is necessary (DB is backing storage)



Entity Beans

- **BMP Entity Beans**
  - More programming effort
  - Very open part of the specification
  - A lot of design patterns can be implemented with BMP Entity Beans
  - BMP Entity Beans can be used to plug-in OR-mapping tools and frameworks
- **CMP Entity Beans**
  - Entity Beans with integrated automated persistence
  - Small effort for persistence
  - Limited control about persistence mechanisms



CMP Entity Beans

- **CMP Entity Bean combines following aspects**
  - Automated persistence
  - Transaction control
  - Integration into distribution protocol
  - Access control
- **Distribution protocol necessary?**
- **Access control necessary?**

- **Conclusion: Very heavy weight**
- **Note: CORBA POS has no distribution**

CMP Entity Beans

- **Automated persistence**
  - EJB 1.1 specification very weak (no relationship support, no dependent objects)
  - EJB 2.0 detailed enough to be base for OR-Mapping
- **Application server implementation**
  - Most app servers have very weak implementations for CMP Entity Beans
  - Most implementation just map table rows into objects
  - Sophisticated OR-Mapping products for integration available (TopLink, Cocobase)
- **Note: INTERSHOP enfinity based on app server Persistence PowerTier with contains OR-Mapping**

How much should EJBs be used?

- **What to do if logic is not exposed remote? Use EJB or plain Java objects?**
- **General answer: Use EJB as much as possible.**
  - Common programming model
  - Move a lot of tasks like pooling or naming server to app server

    ... But ...
- **Check overhead of local calls**
  - Local Bean-Bean calls goes trough RMI layer
  - Spec says, local call must follow RMI semantic
  - App server may provide optimization, so that no significant overhead exists



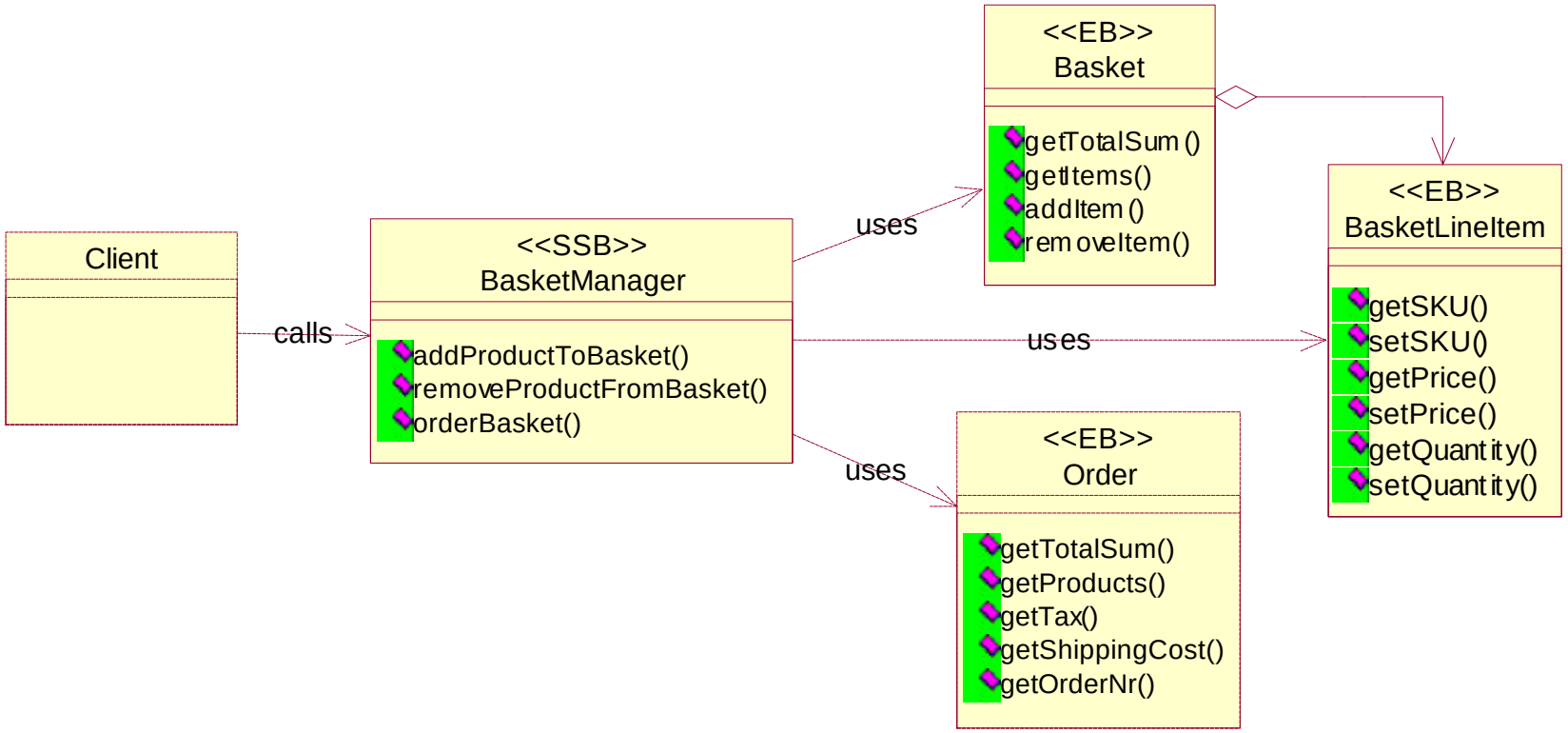
Patterns and Design Strategies

Facade Pattern

- **Session Bean as remote interface**
- **Session Bean wraps Entity Beans**
- **Session Bean contains**
  - Business logic
  - Object-by-Value mechanisms
- **Stateless Session Bean preferred bean type**

- **Note: Used in INTERSHOP enfinity for communication between thick client and server**

Facade Pattern

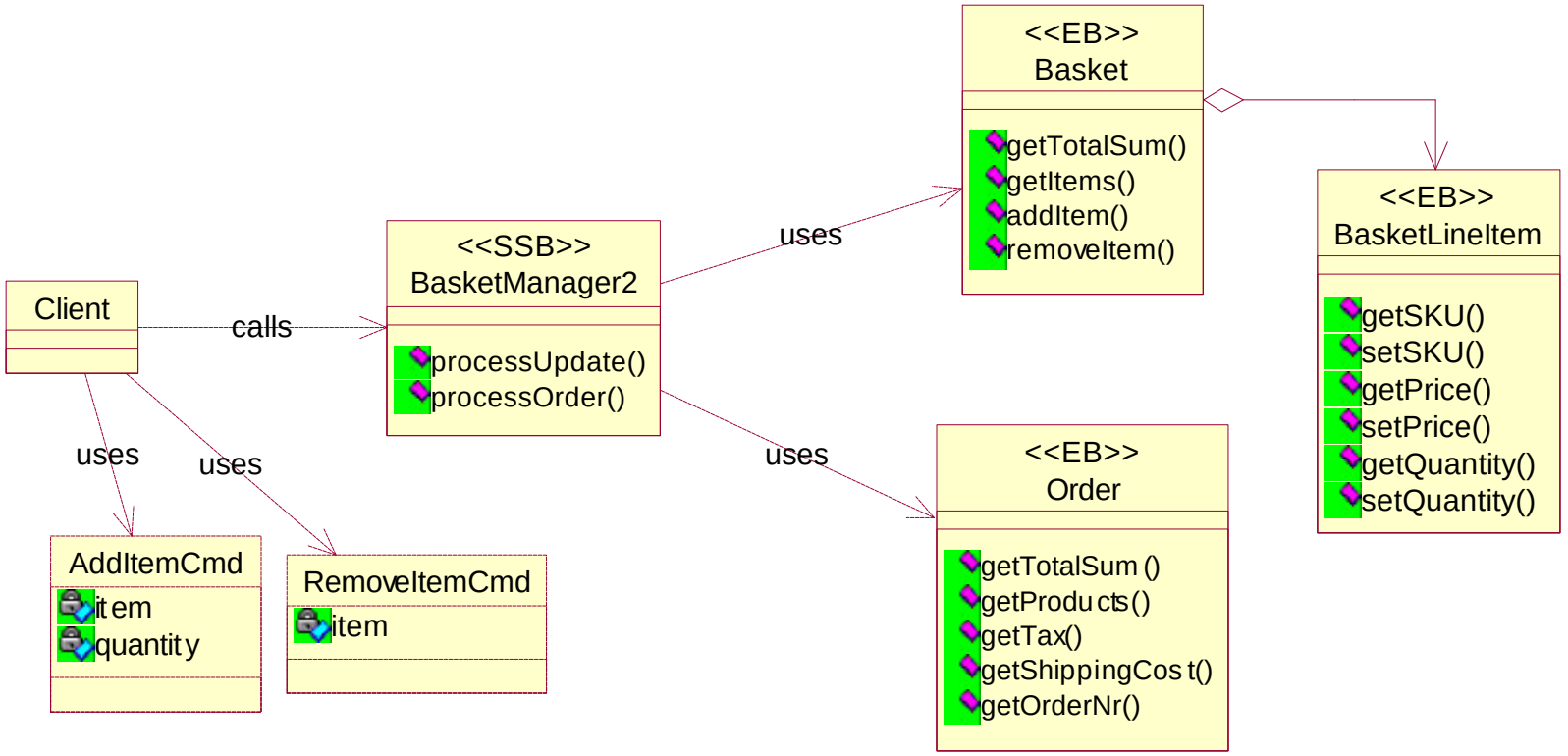




Command Pattern

- **Use command to send business logic action to the server**
- **No fine grained communication**
- **Efficient use of remote communication**
- **Facade can be used as server side pattern to receive command objects**

Command Pattern



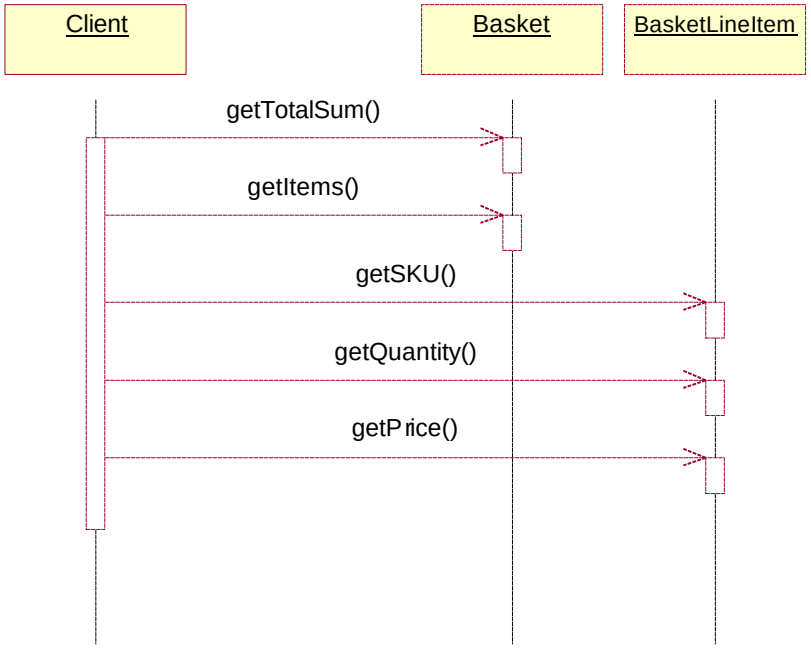
Object-by-Value Pattern

- Transfer of state from server side object
- No single remote method calls for each object
- For complex object the serialization effort can get significant
- Some IDE tools provide support for state objects (e.g. VisualAge)
- Supported with RMI as remote protocol
- Standardized with CORBA 2.3 as remote protocol

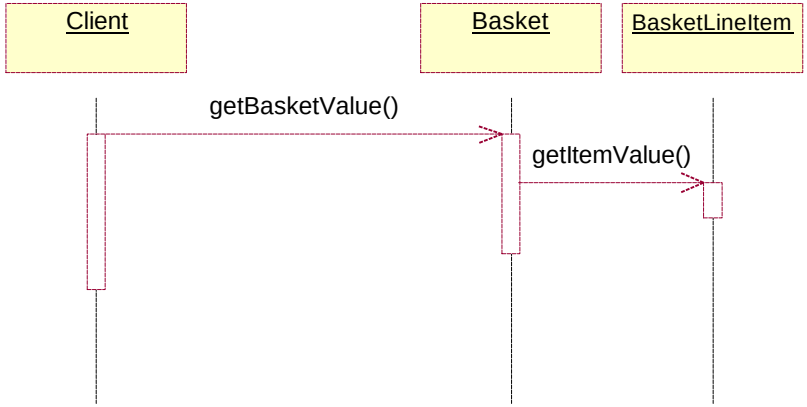
- Note: Used in INTERSHOP enfinity for the communication between thick client and server

Object-by-Value Pattern

Direct access to Enitity Beans



Access through State Objects



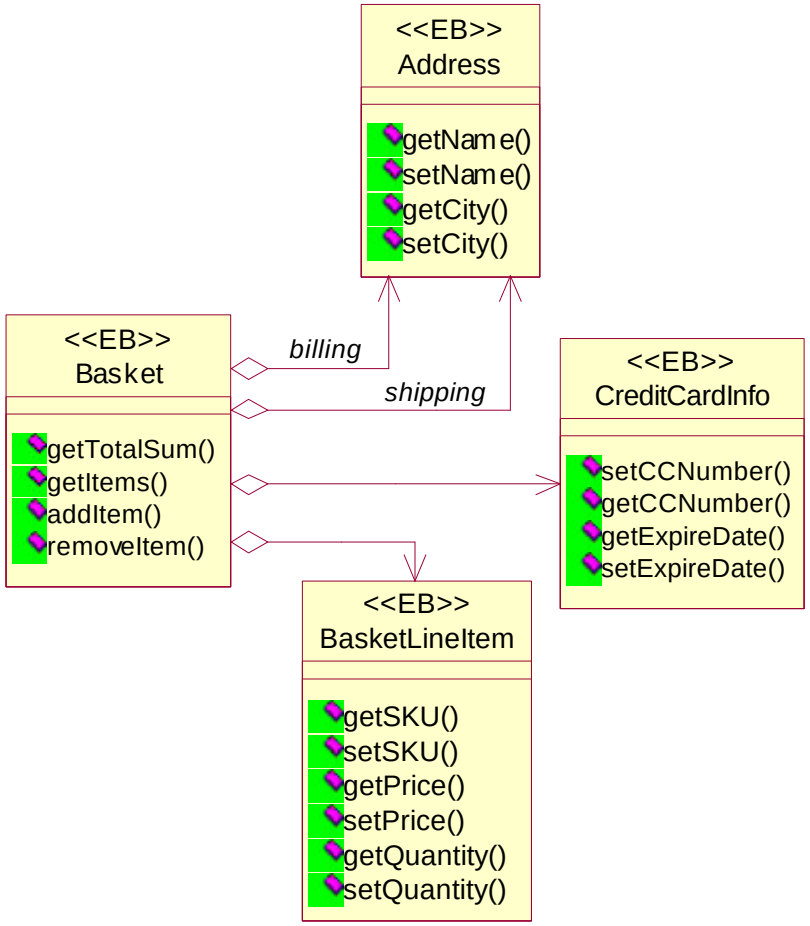


CMP Bean Patterns

- **Pattern: Fine grained CMP Entity Beans**
  - Every persistent object is implemented as Entity Bean
  - A lot of heavy weight business objects
  - A lot of server internal Bean-Bean communication

- **What to do?**
  - Check overhead of local communication in app server
  - Check overhead for initialization in distributed environment
  - Check overhead of empty security check

CMP Bean Patterns



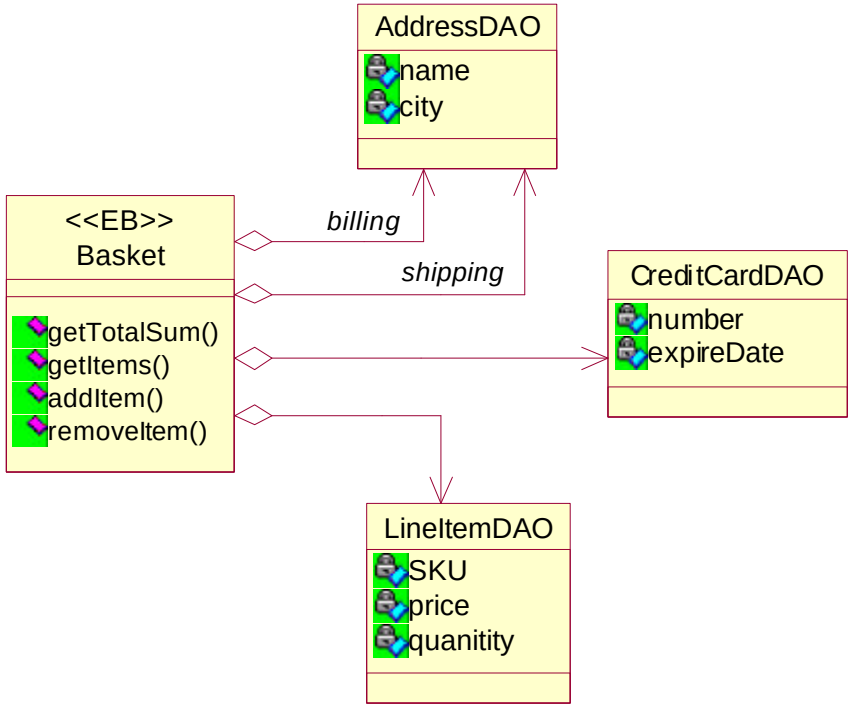
- All depending objects are fill EJBs



CMP Bean Patterns

- **Pattern: Coarse grained CMP Entity Beans**
  - Public interface of Entity Bean represents coarse and very abstracted interface
  - Result is no overhead for distributed protocol and security check for details of persistent state
  - Example: OrderBean but no bean for line items
  - Not possible to implement with CMP and EJB 1.1
  - Possible to implement with CMP and EJB 2.0
  - Alternative: Use OR-Mapping product and BMP Beans

CMP Bean Patterns



- only Basket is a Enity Bean
- dependent objects are modell like described in J2EE blueprint



CMP Bean Patterns

- **Advice:**
  - Use pattern of fine grained CMP Entity Beans
  - With complex business objects use 3rd party OR-Mapping product
  - Get familiar with EJB 2.0
  - Design application for easy migration to EJB 2.0
  - Migrate to EJB 2.0 as soon as possible
  - For critical business objects use BMP

- **Note: INTERSHOP enfinity uses fine grained appraoch**



Caching Patterns

- **Caching is the most efficient way to reduce database operations**
- **Main memory is cheap**
- **Caching strategy depends on data usage**
  - Small number of referential data
  - Huge number of referential data
  - Transactional data



Caching Patterns

- **Small referential data**
  - Data, which changes very rarely
  - Small amount of data, which can be cached completely
  - Easy caching, preloading of all data (whole table)
  - Query should happen in memory
  - Easy use of hash table for lookup
  - Example: Currency table

- **Note: In INTERSHOP enfinity small tables are preloaded at server start**



Caching Patterns

- **Big amount of referential data**
  - Data, which changes very rarely
  - Only selective data can be cached
  - Use dynamic caching e.g. Based on Last-Recently-Used algorithms
  - Query must happen in database
  - Primary key lookup can happen in memory, e.g. With hash table
  - Examples: product data

- **Note: INTERSHOP enfinity uses LRU algorithms**



Caching Patterns

- **Changes to referential data**
  - Changes to referential data may happen rarely
  - Complicate in clustered environment
  - Due to requirement of availability the possibility for changes must be included in design
  - Option 1: Flush cache in case of changes and reload data
  - Option 2: Communicate changes between servers



Caching Patterns

- **Transactional data**
  - Very hard to cache
  - Read cache possible in combination with optimistic locking
  - Caching possible as long as data resides in one VM
    - Support for session stickiness
    - No session migration
    - With session migration risk of stale data and failed transactions due to optimistic locking



Caching Patterns

- **Caching and OR-mapping tools**
  - Most OR-mapping tools provide some caching support, some tools even provide cache synchronization between clustered servers
  - Don t rely on transparent caching
  - Design must explicitly include caching concepts
  - Some architectural components are necessary to handle cache mechanisms



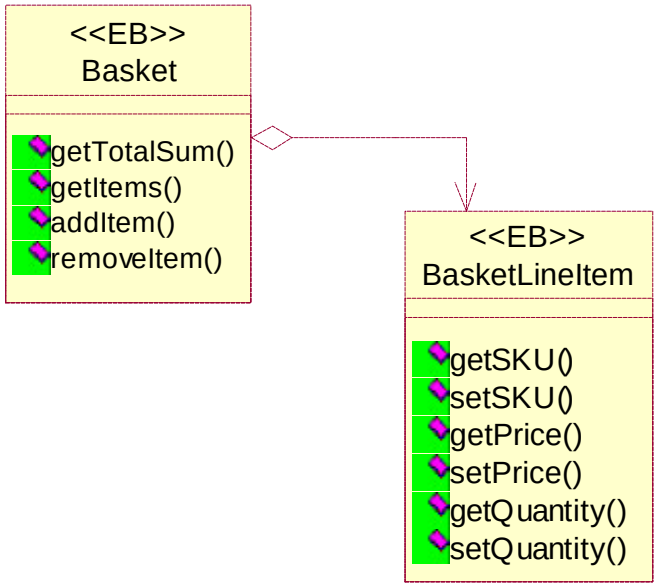
Example

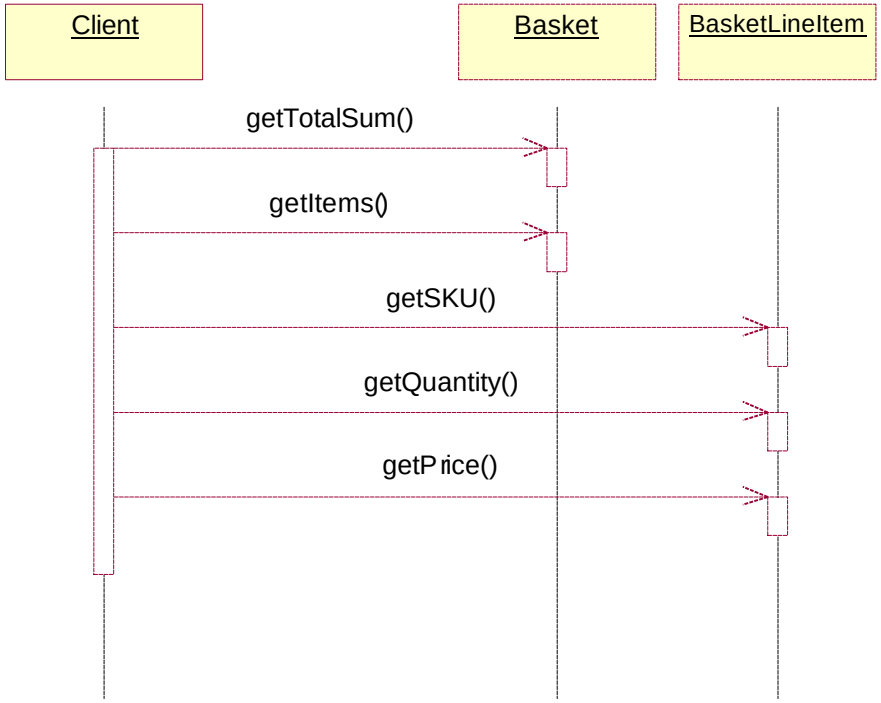
Example: Handling of Basket

- **Operations on basket**
  - View basket
  - Update basket (add items, remove items, change quantity, change attributes)
  - Order basket
- **Example with different patterns**
  - Entity Bean directly exposed in remote protocol
  - Pure Object-By-Value usage
  - Object-By-Value plus command pattern

Example: Entity Bean directly exposed

- **Implementation**
  - Entity Bean for basket directly exposed



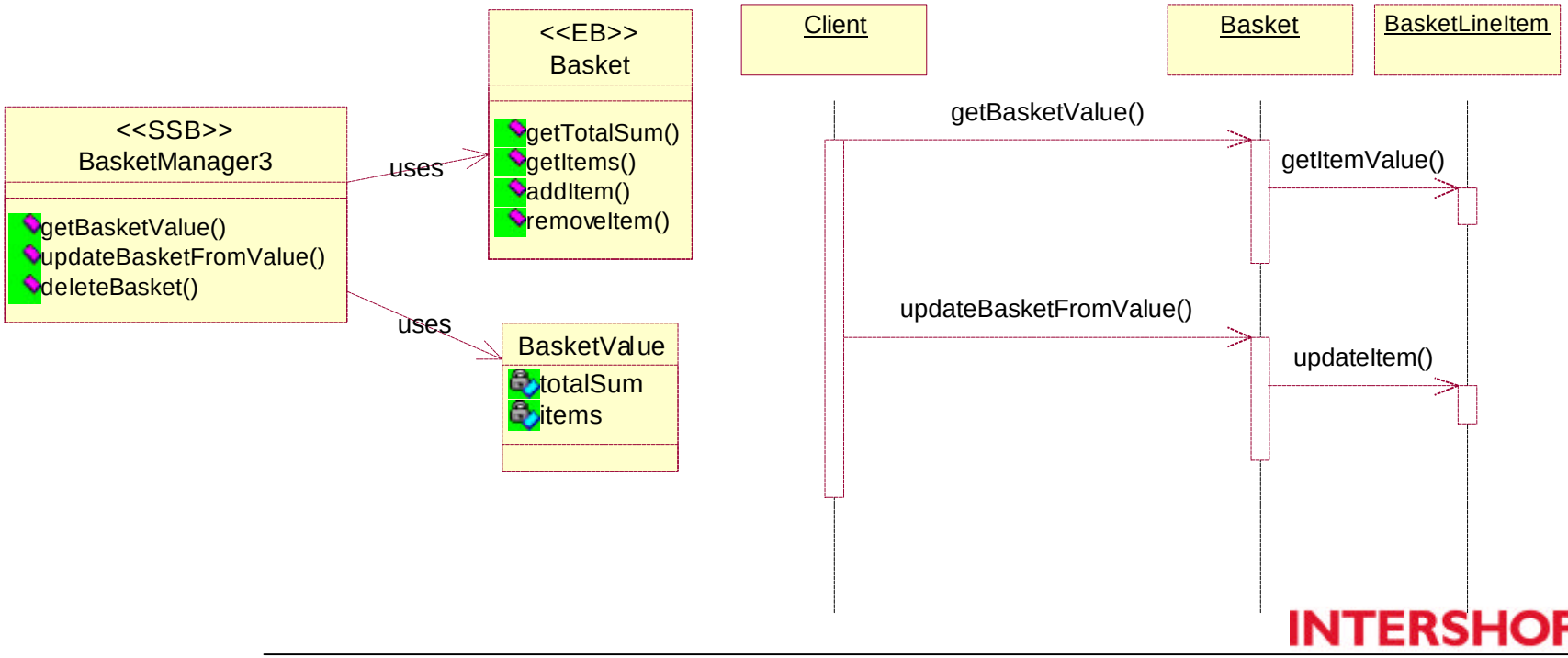


Example: Entity Bean directly exposed

- **Advantage**
  - Efficient for small changes like changing quantity
- **Disadvantage**
  - Many small remote calls (get-, set- method for each attribute)
  - No business logic control about changes
  - May cause a lot of database operation (app server stores EJB content after each method call in DB)

Examples: Object-By-Value

- **Implementation**
  - One method for reading the whole state of the basket
  - One method for updating the whole state of the basket
  - Comparable to simple CRUD (create-update-delete) concept



Examples: Object-By-Value

- **Advantage**
  - Efficient remote communication for reading
  - Efficient remote communication for big changes
  - Database operations can be „bundled“ together
- **Disadvantage**
  - Big database transactions (all values must be copied back into database)
  - Complex logic to find update delta to achieve smaller transactions
  - No business logic control about changes
  - Whole state must be read from database and send to client even so only a small part is of interest

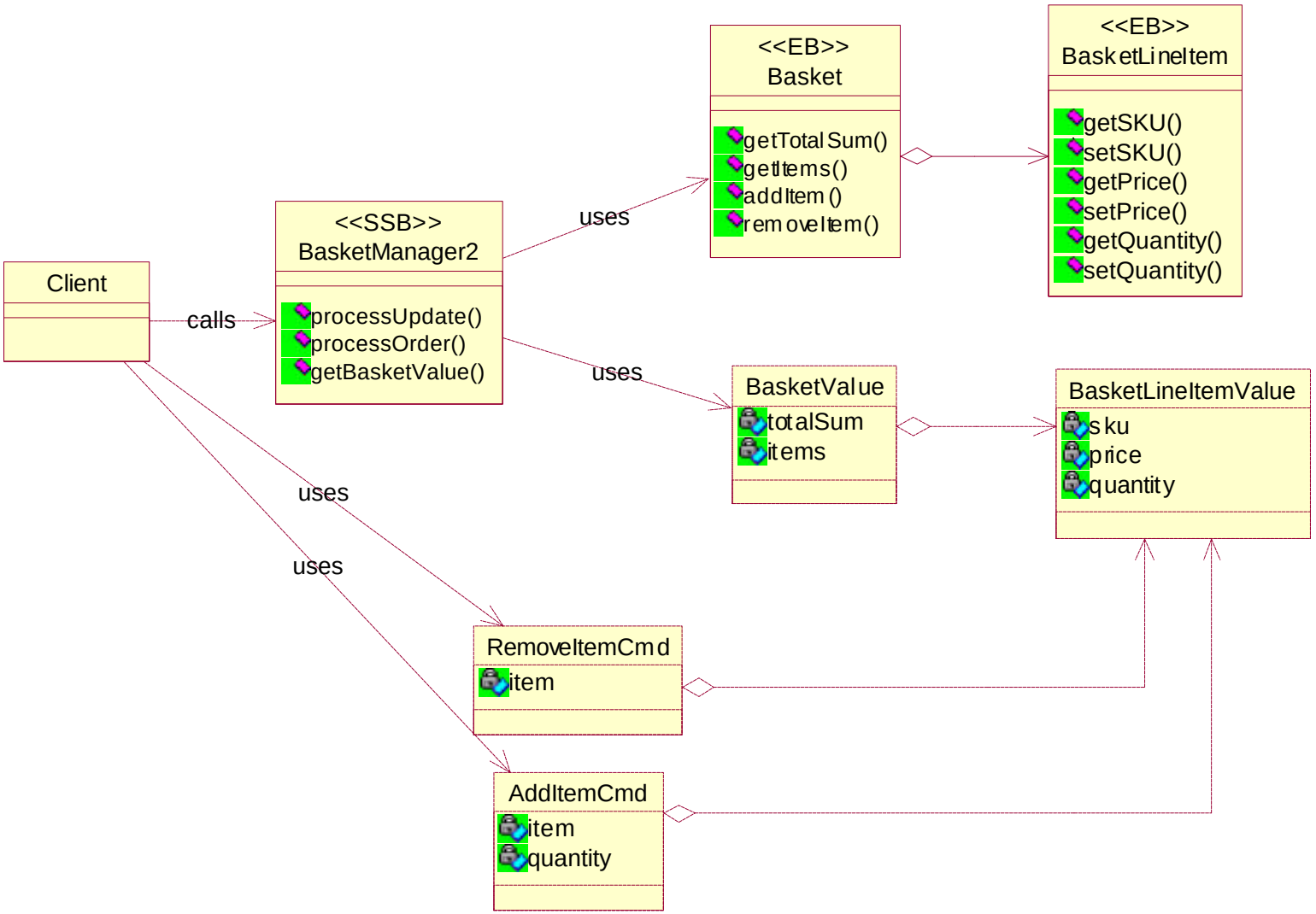
Examples: Object-By-Value

- **Possible Improvements**
  - Different states for one business object, to retrieve only certain aspects (e.g. state for single line item in basket)
  - Separate update mechanisms for certain aspects (e.g. Update of single line item in basket)

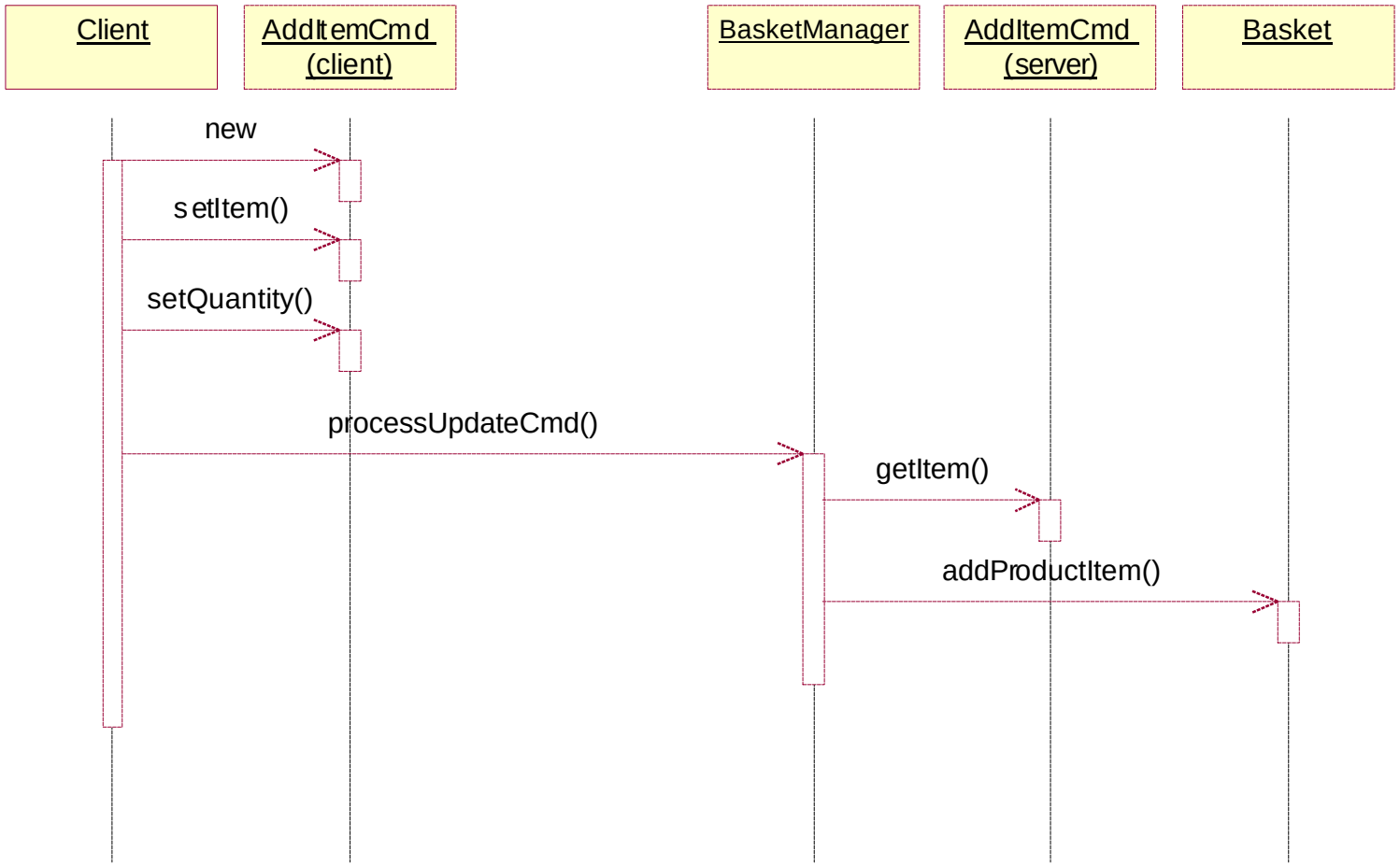
Example: Facade + ObV + Command

- **Implementation**
  - Methods for reading state of basket with ObV efficiently
  - Commands for updating certain aspects
    - Command for updating quantity
    - Command with list of items to add
    - Command with list of items to remove
    - Small change ? single command
    - Big changes ? multiple commands in one remote communication
  - Variation
    - Don t use commands, use method with multiple arguments

Example: Facade + ObV + Command



Example: Facade + ObV + Command



Example: Facade + ObV + Command

- **Advantage**
  - ObV helps on reading object state efficiently
  - Commands for update help
  - Business logic can control changes
  - Changes specified well enough, so that database transactions can be made small
  - All database operations can be „bundled“ together
- **Disadvantage**
  - More effort necessary to design interfaces, no straight forward implementation possible



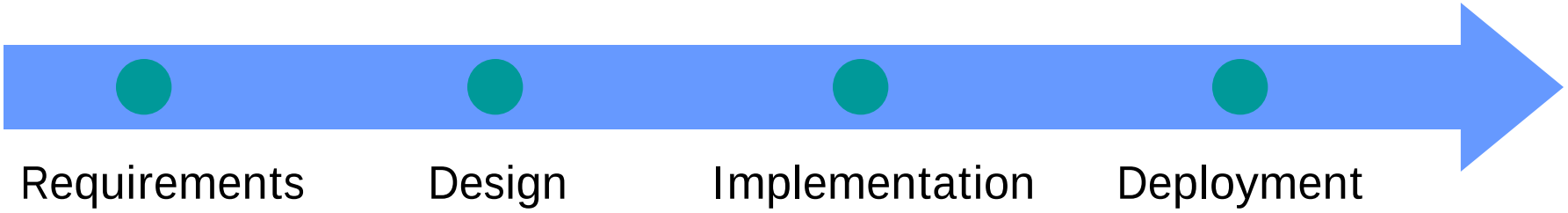
Advice

- **Preferences should be on hiding entity beans**
- **A lot of small business objects and simple transactions**
  - **go with pure ObV**
- **Complex business objects and more complex transactions**
  - **go with ObV plus command pattern**



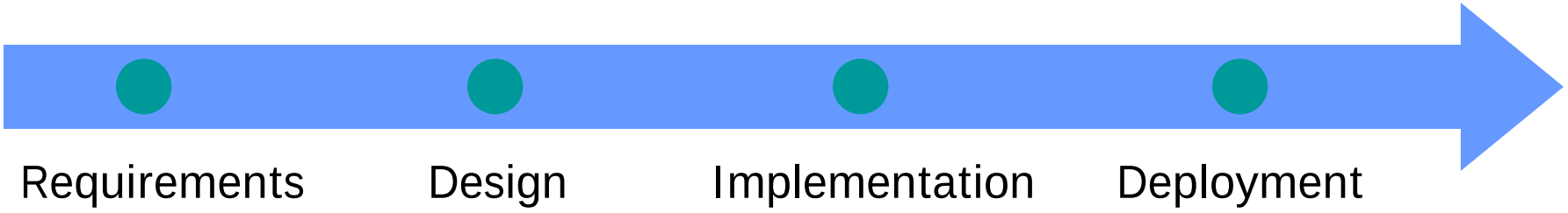
Wrap Up

Performance in all development phases



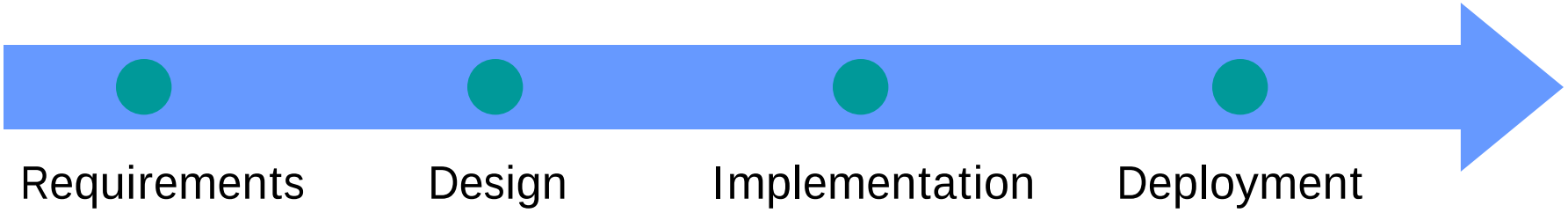
- **Requirements**
  - Collect information about app server architecture
  - Analyze general structure of application
    - more database reading than transaction
    - data volume
    - expected number of requests

Performance in all development phases



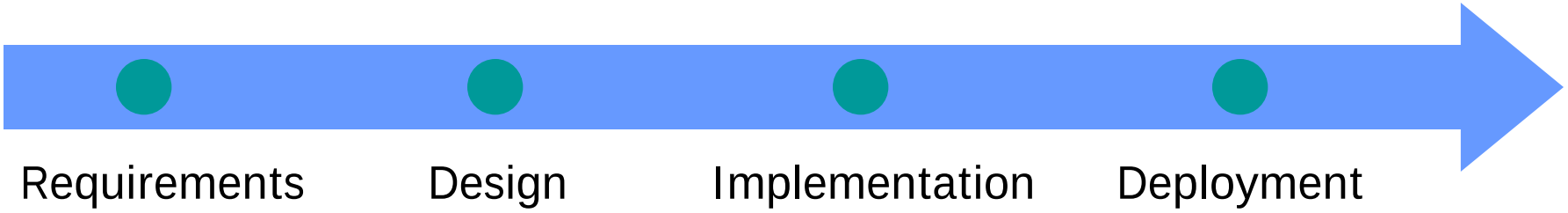
- **Design**
  - Optimize design with respect to remote communication
  - Optimize design with respect to database transactions
  - Design patterns for these aspects

Performance in all development phases



- **Implementation**
  - Optimize on parallelism
    - Multi-threading
    - Multi-processor
  - Resource efficient implementation

Performance in all development phases



- **Deployment**
  - Operating system tuning
    - Tune resources managed by OS (e.g. socket handling)
  - Database tuning



Advanced Concepts

- **JTA impact**
  - Distributed transaction vs. Local transaction
  - EJB 2.0 addresses this issue
- **JMS used in design**
  - Use JMS to queue tasks for later processing
  - e.g. don t process order immediately, push into queue
- **In-Memory-Databases**
  - With 64-Bit architecture feasible
  - Significantly faster than disk based RDMS
  - Use as cache, for short time persistent data and processing buffer



Links

- **http://www.theserverside.com**
  - News and articles about J2EE app servers
  - Discussion about design patterns and implementation concepts
- **http://www.c2.com/cgi/wiki?EjbRoadmap**
  - Several EJB design patterns
  - Other articles around EJB

INTERSHOP
Creating the Digital Economy™

Q&A






INTERSHOP ENFINITY™