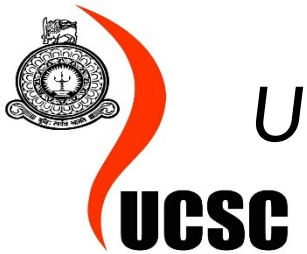
Advanced Java Application Development Using JavaEE



University of Colombo School of Computing

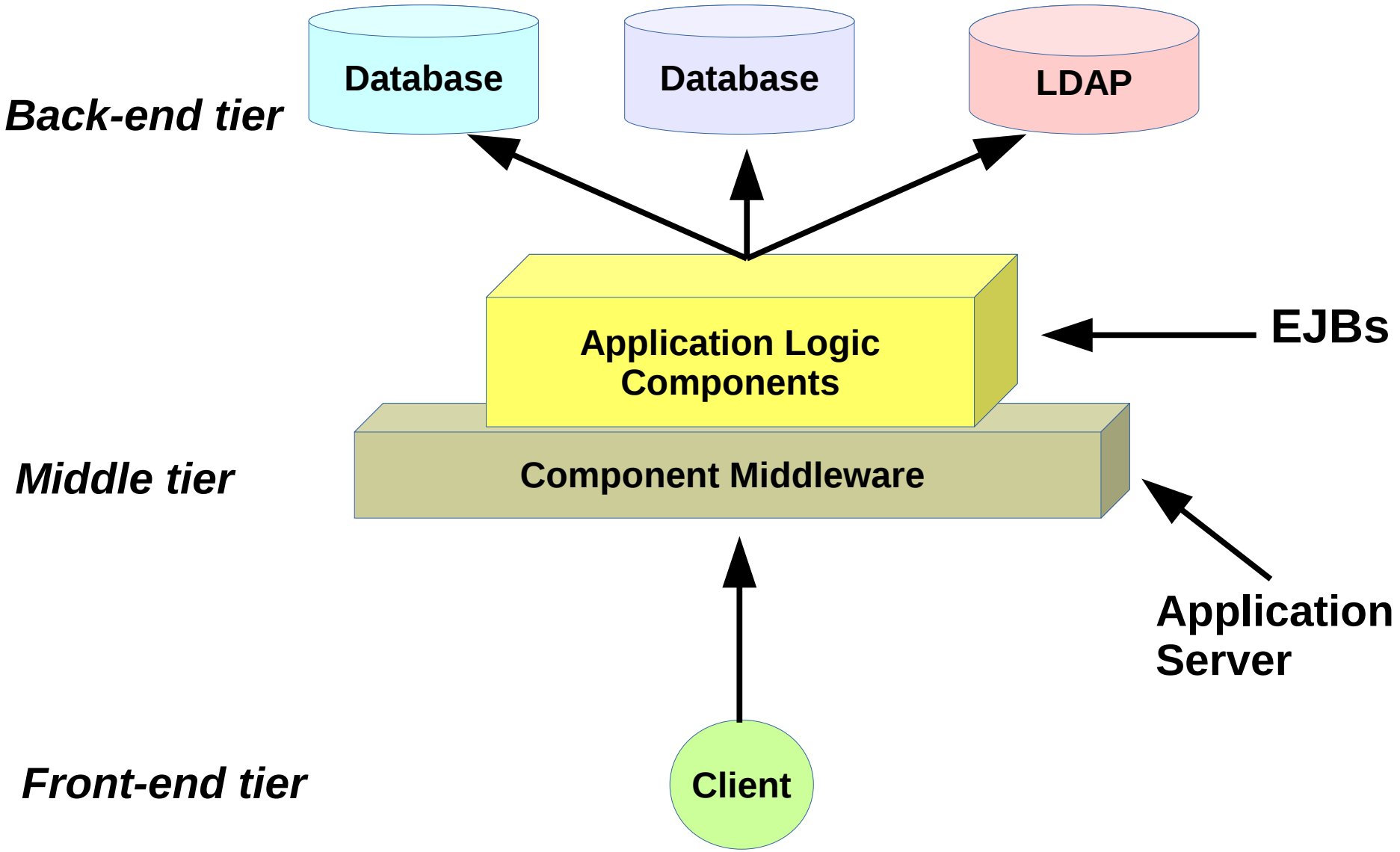
What is EJB 3 ?

- Enterprise JavaBeans (EJB) is an architecture for setting up program components written in the Java programming language
- EJB run in the server part of a computer network that uses the client/server model.
- Enterprise JavaBeans is built on the JavaBeans technology for distributing program components (which are called Beans) to clients in the network.
- Enterprise JavaBeans offers advantage of being able to control change at the server rather than having to update each individual computer with a client whenever a new program component is changed or added.
- EJB component have the advantage of being reusable in multiple applications. To deploy on EJB Bean or component, it must be a part of a specific application which is called container.

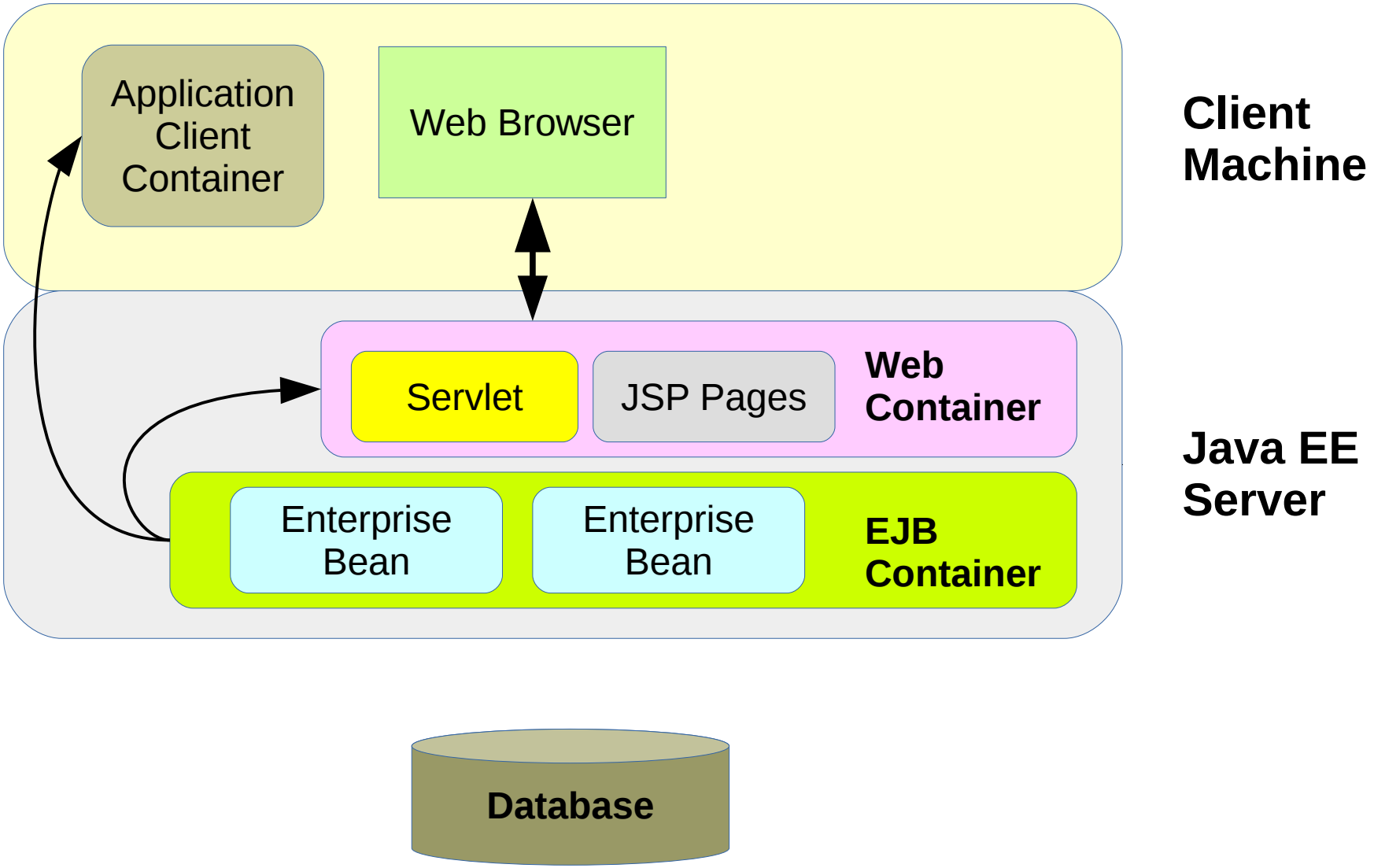
EJB's Targets

- Standard component architecture for building distributed business applications in Java.
- Interoperability between enterprise beans and other Java Platform Enterprise Edition Components, as well as non-Java applications.
- Compatible with other Java APIs and with COBRA (Common Object Library for Building Reliable Application) protocols.
- Follow the Write Once, Run Anywhere philosophy of Java an enterprise bean can be developed once and then deployed on multiple platforms without recompilation or source code modification.
- Define the “contracts” that enable tools from multiple vendors to develop and deploy components that can inter-operate at runtime.

EJB 3 Three Tiered Architecture



JEE 3 Three Tiered Architecture



EJBs as Components

- Enterprise Java Beans are components that provide middle-tier business logic.
- And interact heavily with the data layer of the application.
- EJB framework conforms to and at the same time induces 3-tier architecture for distributed applications

EJB as Component Model Framework

1. Programming model
2. Standardized interfaces
3. Runtime environment
4. Meta-data
5. Built-in component services (persistence, transactions, security, etc.)
6. Deployment facilities

EJB Specification

- EJB is an open specification – any vendor can develop a run-time environment that complies with the specification
- EJB code intended to be portable across brands (assuming uses only services defined by the spec, not additional vendor facilities)
- EJB specs have evolved:
  - originated with IBM 1997
  - Later adopted by Sun (1.0 1998, 1.1 1999)
  - Enhanced under Java community process (2.0 2001, 2.1 2003, 3.0 2006)
- EJB 3.0 is a major departure from earlier versions, but backwards compatible (old code works with 3.0 but not vice versa)

Enterprise Beans

- Body of code with fields and methods
- Encapsulates business data or business logic that operate on the enterprise's data
- Instances are created and managed at run time by a Container (application server)
- Client access is mediated by the bean instance's Container – isolates the bean from direct access by client applications (and other beans)

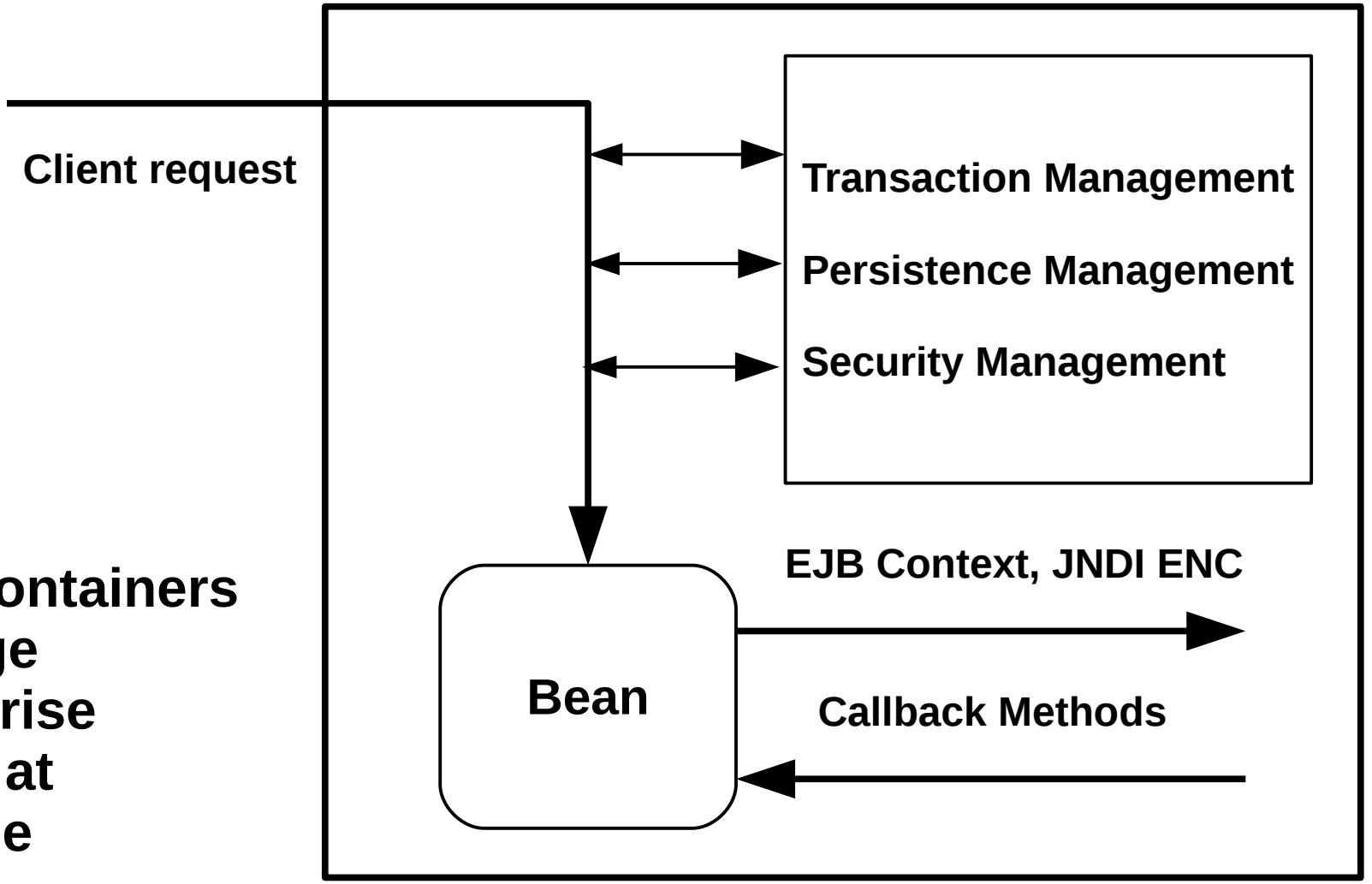
Enterprise Bean Portability

- If an enterprise bean uses only the services defined by the EJB specification (and not additional facilities peculiar to the vendor), the bean can be deployed in any compliant EJB Container
- Can be included in an assembled application without requiring source code changes or recompilation.
- Component services information, such as a transaction and security attributes, are separate from the enterprise bean class – this allows the services information to be managed by tools during application assembly and deployment.
- Can be customized at deployment time by editing the bean's environment entries and /or deployment descriptor.

EJB Container

- Manages every aspect of an enterprise bean at run-time and implements component services.
- When a client application invokes a method on an enterprise bean, the container first intercepts the invocation to ensure persistence, transactions and access control are applied properly to every operation a client performs on the bean.
- An enterprise bean cannot function outside of an EJB container.

EJB Container

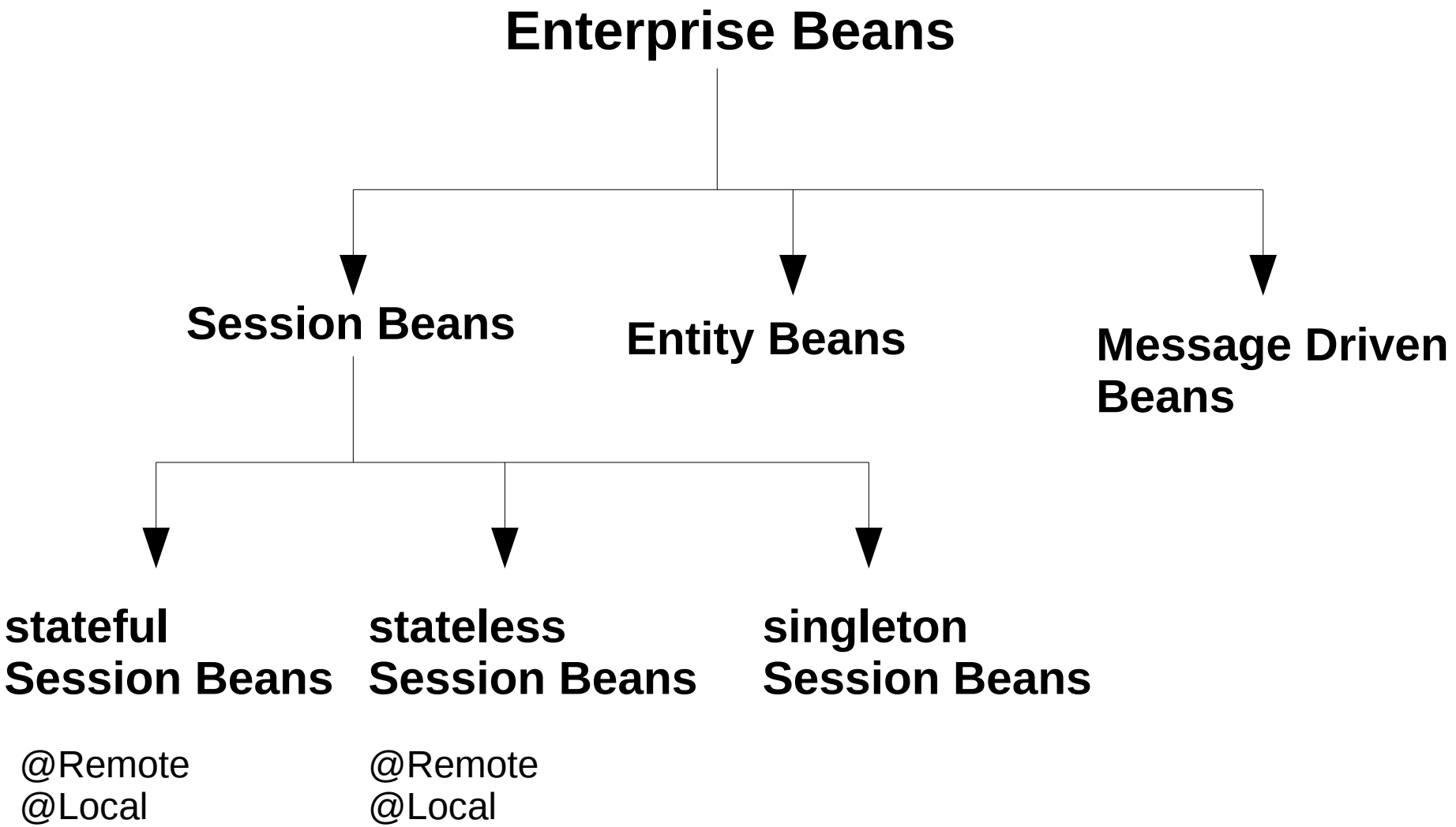


EJB Containers manage enterprise beans at runtime

Resource Management

- Containers manage many beans simultaneously
- To reduce memory consumption and processing, containers pool resources.
- When a bean is not being used, a container may replace it in a pool to be reused by another client.
- Or evict it from memory (passivate) and only bring it back (activate) when its needed.
- While its reference on the client remains intact.
- When the client invokes a method on the reference, the container re-incarnates the bean to service the request.

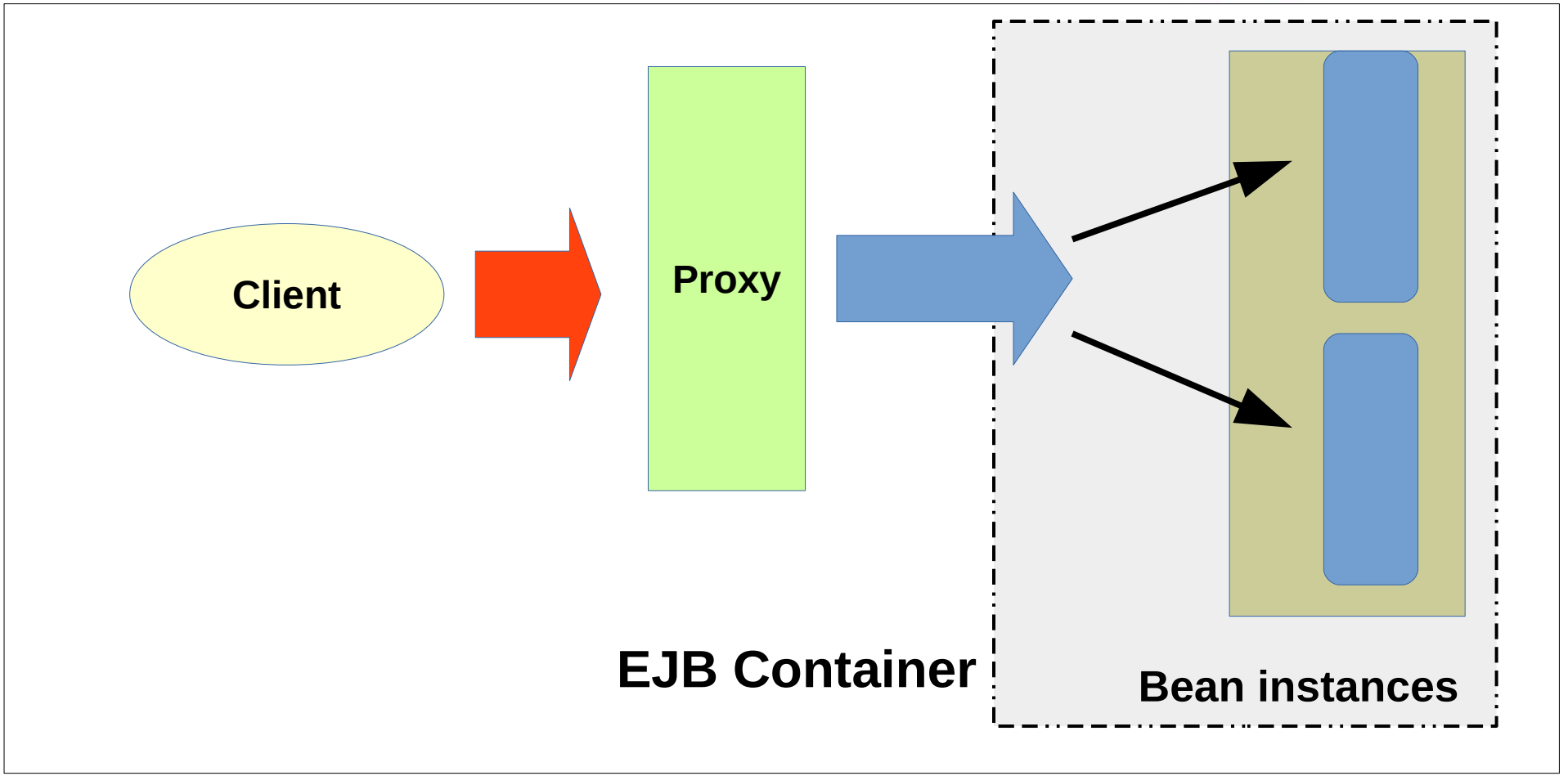
Types of Beans



Session Bean

- Frequently contain business methods
- Relevant EJB's instances create when the client access the bean via EJB container. This mean that the client does not access the EJB directly.
- This allows container to perform all sorts of magic before a request finally hit the target method.
- It's this separation that allows for the client to be completely unaware of the location of the server, concurrency policies, or queuing of requests to manage resources.
- As far as the client is concerned, it's operating directly upon an EJB. In truth client is invoking upon a proxy reference that will delegate the request along to the container and return the appropriate response.

Session Beans

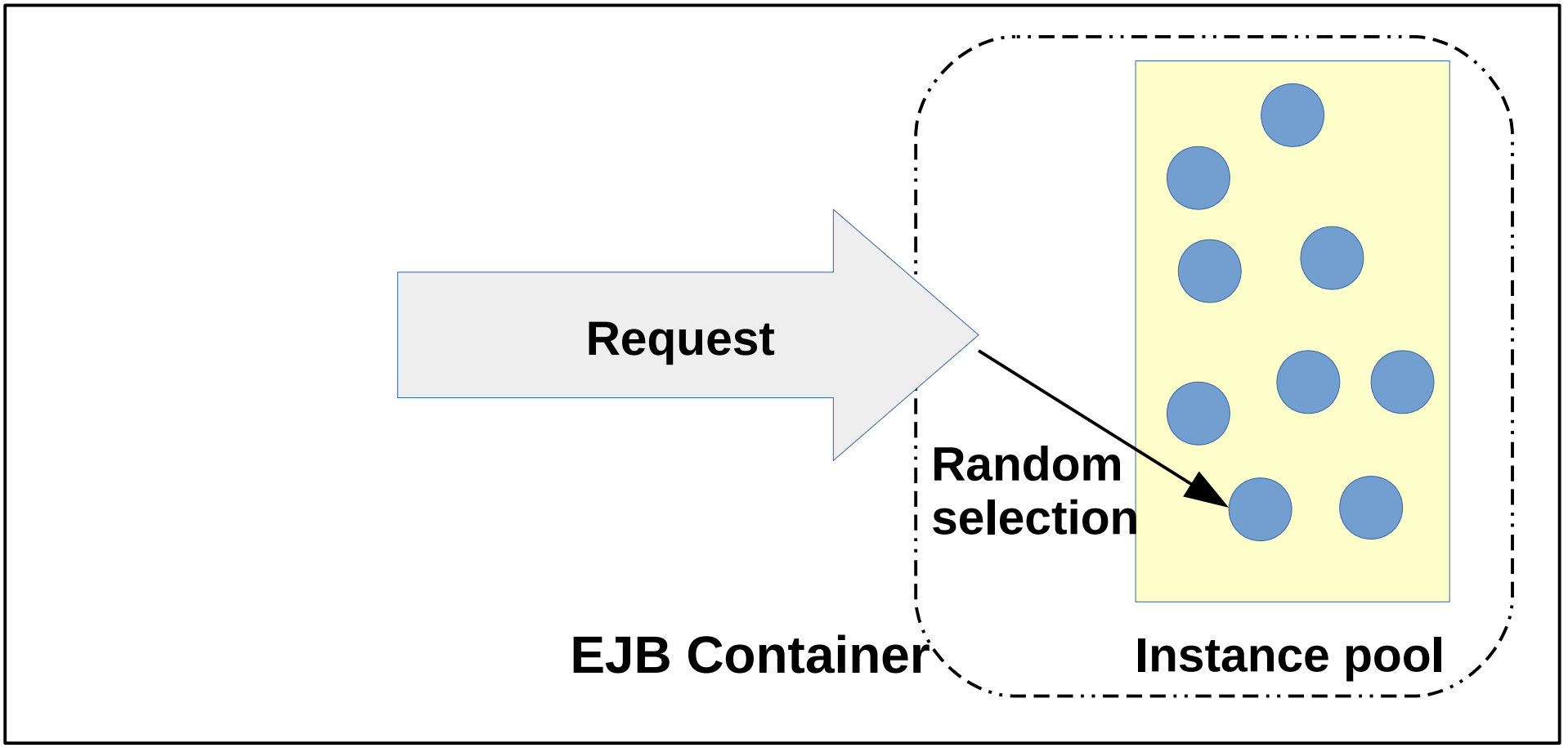


Ultimately, it's the bean instances created and managed by the Container that the service client request.

Stateless Session Beans (SLSB)

- Stateless Session Beans are useful for functions in which state does not need to be carried from invocation to invocation.
- A client cannot assume that subsequent requests will target any particular bean instance.
- The container will often create and destroy instances however it feels will be most efficient.
- How a container chooses the target instance is left to the vendor's discretion
- Because there's no rule linking an invocation to a particular target bean instance, these instances may be used interchangeably and shared by many clients.
- This allows the container to hold a much smaller number of objects in service, hence keeping memory footprint down.

Stateless Session Beans (SLSB)



An SLSB Instance Selector picking an instance at random

Stateless Session Beans (SLSB)

- Stateless session bean is designed for efficiency and simplicity
- In conversational state is the point of time that refer information exchanged and remembered within the context of series of requests between the client (caller) and EJB (service).

Illustration:

Peter: Hi!, my name is Peter.
Service: Hello, Peter
Peter: Do you know my name?
Service: Yes, your name is Peter

- Here, the state variable holding the name of the client is stored for the duration of the conversation. Stateless session beans are incapable of handling this exchange correctly.
- Each invocation upon a SLSB proxy operates independently from those both before and after. Really, its underlying bean instances may be swapped inter-changeably between requests.

Stateless Session Beans (SLSB)

- Bypassing conversational state doesn't mean that a stateless session bean can't have instance variables or maintain any kind of internal state.
- Nothing prevents you from keeping a variable that tracks the number of times a bean instance has been called or a variable that saves data for debugging.
- This state may never be visible to a client.
- The caller of SLSB can't assume that the same bean instance will service all of its request.
- Instance variable may have different values in different bean instances, so their state is likely to appear to change randomly as stateless session beans are swapped from one client to another.

Stateless Session Beans (SLSB)

Mike: Hi!, my name is Mike

SLSB: Hello, Mike

Mike: Do you know my name?

SLSB: Yes, your name is Jason

Mike: What? That's wrong, I'm going

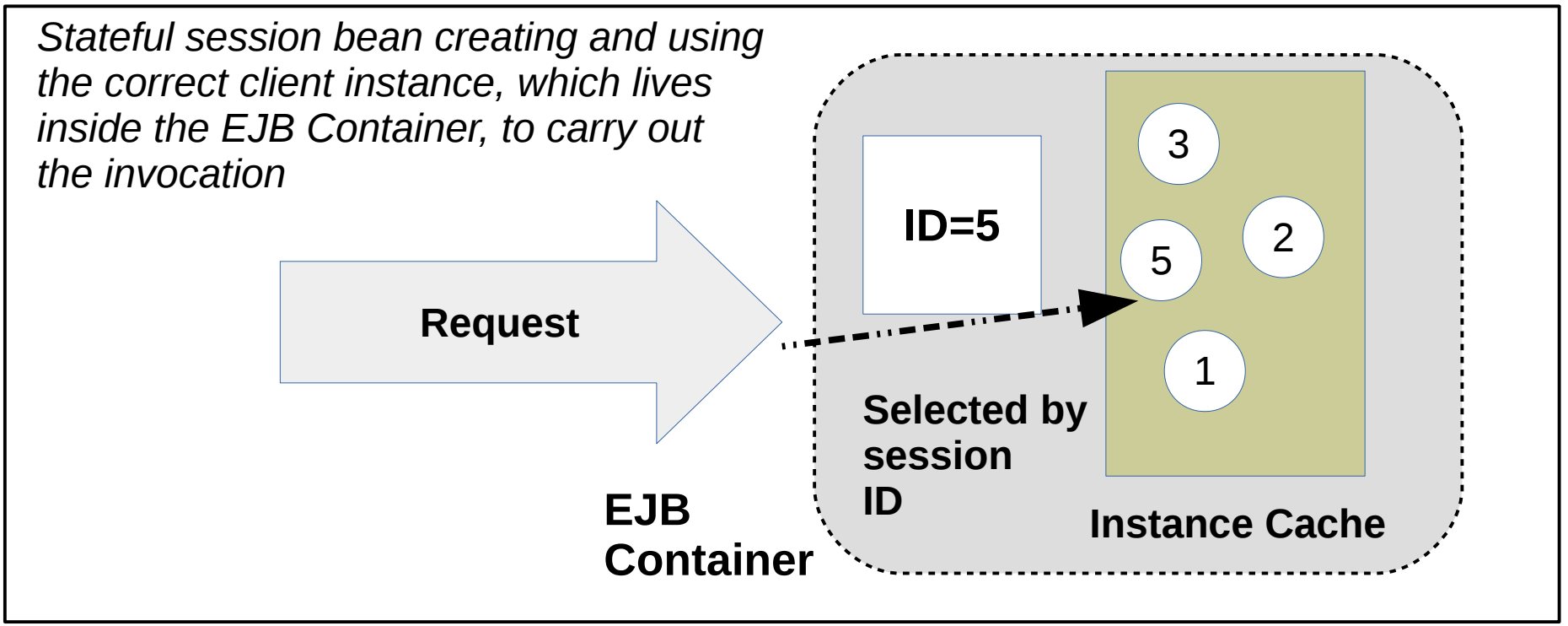
SLSB: OK, later, Dave

Stateful Session Beans (SFSB)

- Stateful session beans differ from SLSBs in that every request upon a given proxy reference is guaranteed to ultimately invoke upon the same bean instance.
- SFSB invocations share conversational state
- Each SFSB proxy object has an isolated session context so calls to one session will not affect another.
- Stateful sessions, and their corresponding bean instances are created sometime before the first invocation upon a proxy is made to its target instance.
- They live until the client invokes a method that the bean provider has marked on a remove event, or until the Container decides to evict the session (Usually due to some timeout though the space leaves this out-of-scope and up to the vendor)
- In order to minimize the number of stateful sessions carried around in memory, the Container may *passivate* a SFSB bean instances.

Stateful Session Beans (SFSB)

- During passivation the session's state is flushed to some persistent storage such that it may be removed from RAM. If the session is needed again before it's removed for good, the Container will activate it and bring the bean instance back into memory.



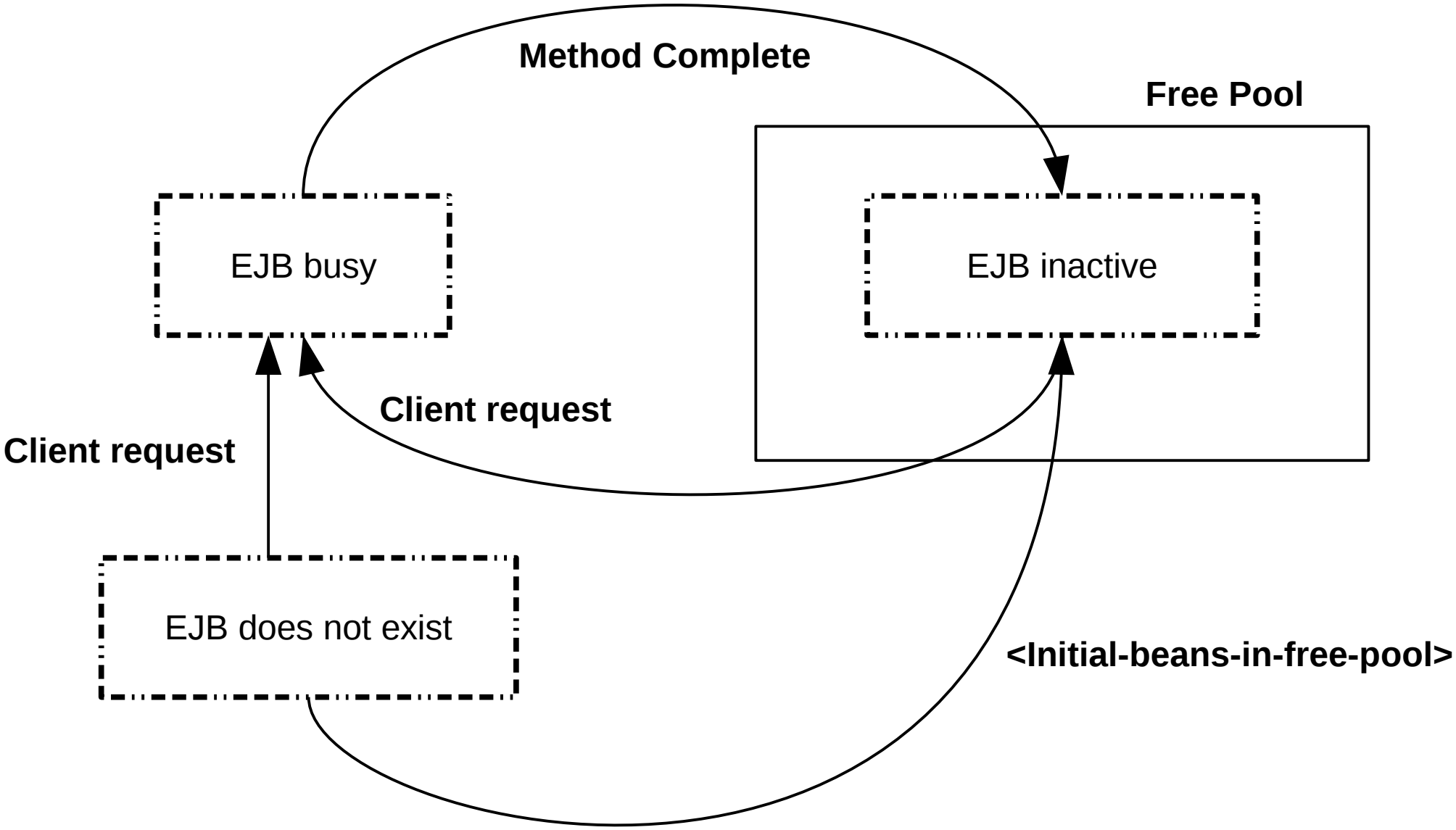
Comparison of Stateful Session Beans (SFSB) and Stateless Session Beans (SLSB)

Stateless Session Bean	Stateful Session Bean
Are pooled in memory to save the overhead of creating a bean every time one is needed. WebLogic server uses a bean instance when needed and put it back in the pool when the work is complete Stateless session bean provide faster performance than stateful beans.	Each client create a new instance of a bean, and eventually removes it. Instances may be passivated to disk if the cache fills up. An application issues ejbRemove() to remove the bean from the cache. Stateful session beans do not perform as well as stateless session beans
Have no identity and no client association, they are anonymous	Are bound to particular client instances. Each bean has an implicit identity. Each time a client interacts with a stateful session bean during session, it is the sam object
Do not persist. The bean has no state between calls	Persist. A stateful session bean's state is preserved for the duration of a session

Pooling for Stateless Session Beans [WebLogic Server]

- By default, no stateless session beans instances exist in WebLogic Server at start up time.
- As individual beans invoked, WebLogic server initializes new instances of the EJBs
- However, in a production environment, WebLogic Server can provide improved performance and throughput for stateless session bean, by maintaining a free pool of unbound stateless session bean instances that are not currently processing a method call. If an unbound instance is available to serve a request response time improves, because the request does not have to wait for an instance to be created. The free pool improves performance by reusing objects and skipping container callbacks when it can.
- Upon start up, WebLogic server automatically creates and populates the free pool with the quantity of instances specified in the bean's *initial-beans-in-free-pool* deployment element in the *weblogic-ejb-jar.xml* file. By default it has been set to 0.

Pooling for Stateless Session Beans [WebLogic Server]



Caching and Passivating Stateful Session Beans [WebLogic]

- WebLogic Server uses a cache of bean instances to improve the performance of stateful session bean.
- The cache stores active Bean instances in memory so that they are immediately available for client request. The cache contains Beans that are currently in use by a client and instances that were recently in use. Stateful Session Beans in cache are bound to a particular client

Stateful Session Beans Creation

- At start up no stateful session beans exist in WebLogic server
- Before a client begins accessing a stateful session bean,it creates a new bean instance to use during its session with the bean. When the session is over the instance is destroyed, while the session is in progress, the instance is cached in memory.

Caching and Passivating Stateful Session Beans [WebLogic]

Stateful Session Bean Passivation

- Passivation is the process by which WebLogic Server removes an EJB instances from cache while preserving its state on disk
- While passivated, beans are not in memory and are not immediately available for client request as they are when in the cache.

Controlling Passivation

- The rules that govern the passivation of stateful session beans vary, based on the value of the beans *cache-type* element.

LRU – least recently used – or eager passivation
NRU – not recently used – or lazy passivation

Caching and Passivating Stateful Session Beans [WebLogic]

Eager Passivation

Cache type to LRU

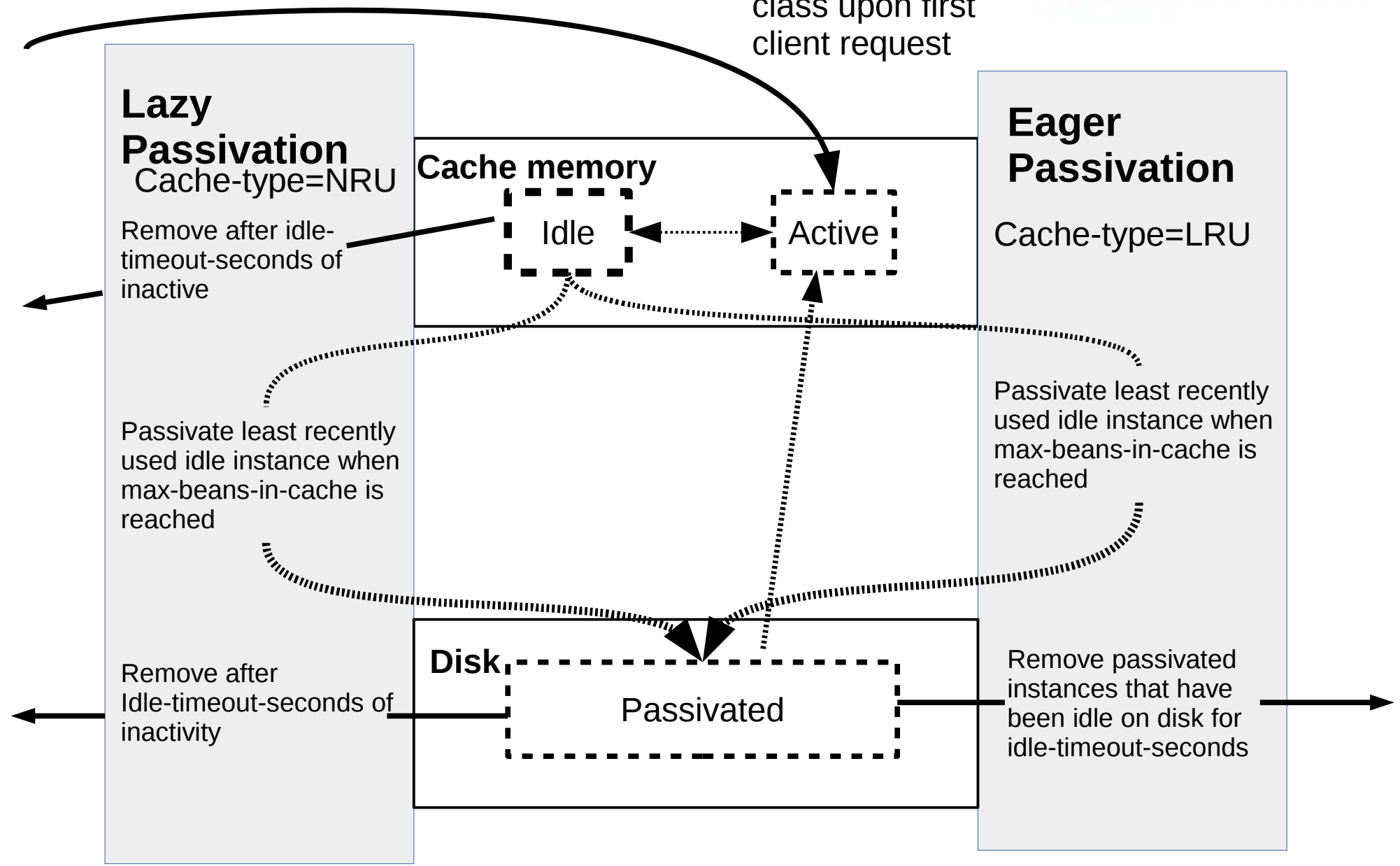
- As soon as an instance has been inactive for *idle-timeout-seconds* regardless of the value of *max-beans-in-cache*
- When *max-beans-in-cache* reached, even though *idle-timeout-seconds* has not expired.

Lazy Passivation

Cache type to NRU

- Removes a bean instance from cache when *idle-timeout-seconds* expires, and does not passivate into disk.
- Passivates instances to disk when *max-beans-in-cache* is reached, even though *idle-timeout-seconds* has not expired.

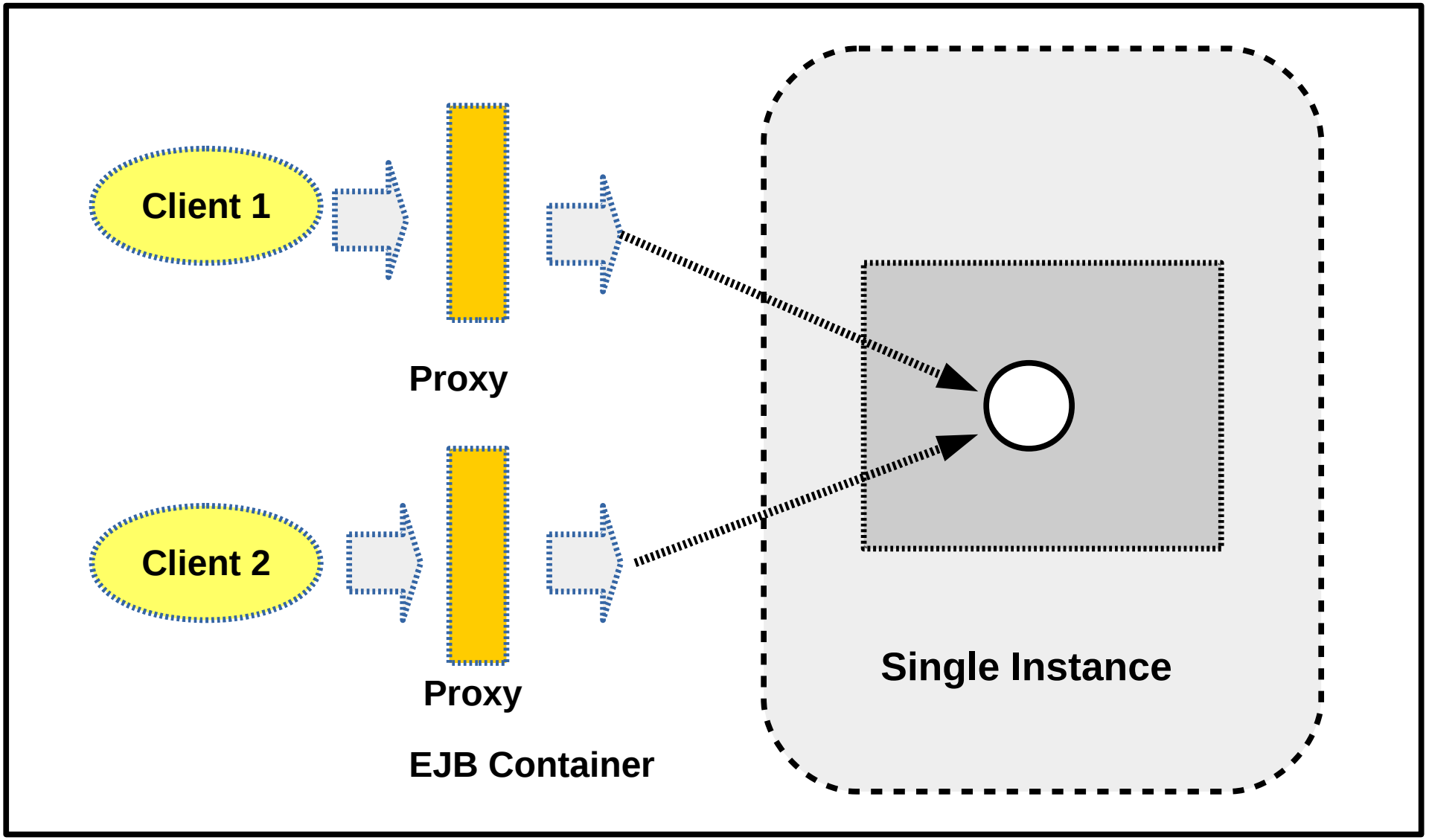
Stateful Session Bean Life Cycle



Singleton Bean

- Sometimes an application doesn't need more than one backing instance for business object
- Therefore EJB 3.1 introduces a new session component type, the Singleton Bean.
- Because, all requests upon a singleton are destined for the same bean instance, the Container doesn't have much work to do in choosing the target.
- The singleton session bean may be marked to eagerly load when an application is deployed, it may be leveraged to fire application life cycle events.

Singleton Bean



Singleton Bean

In this scenario all client requests are directed through the Container to a sole target instance. This paradigm closely resembles that of a pure POJO service approach and there are quite a few repercussions to consider

1) *The instance's state is shared by all requests.*
2) There may be any number of requests pouring through the instance's methods at any one time
3) Due to concurrent invocations, the EJB must be designed as thread safe.
4)Locking or synchronization done to ensure thread safety may result in blocking (waiting) for new requests. This may not show itself in single-user testing, but will decimate performance in a production environment if it doesn't design very carefully.

Singleton Bean

5) Memory footprint is the leanest of all session bean types. With only backing instance in play. Don't take up much RAM

6) Singleton bean is poised to be an increadibley efficient choice if applied correctly.

7) Used in the wrong circumstance, or with improper locking strategies, then it leads to disaster.

Session Bean Interfaces

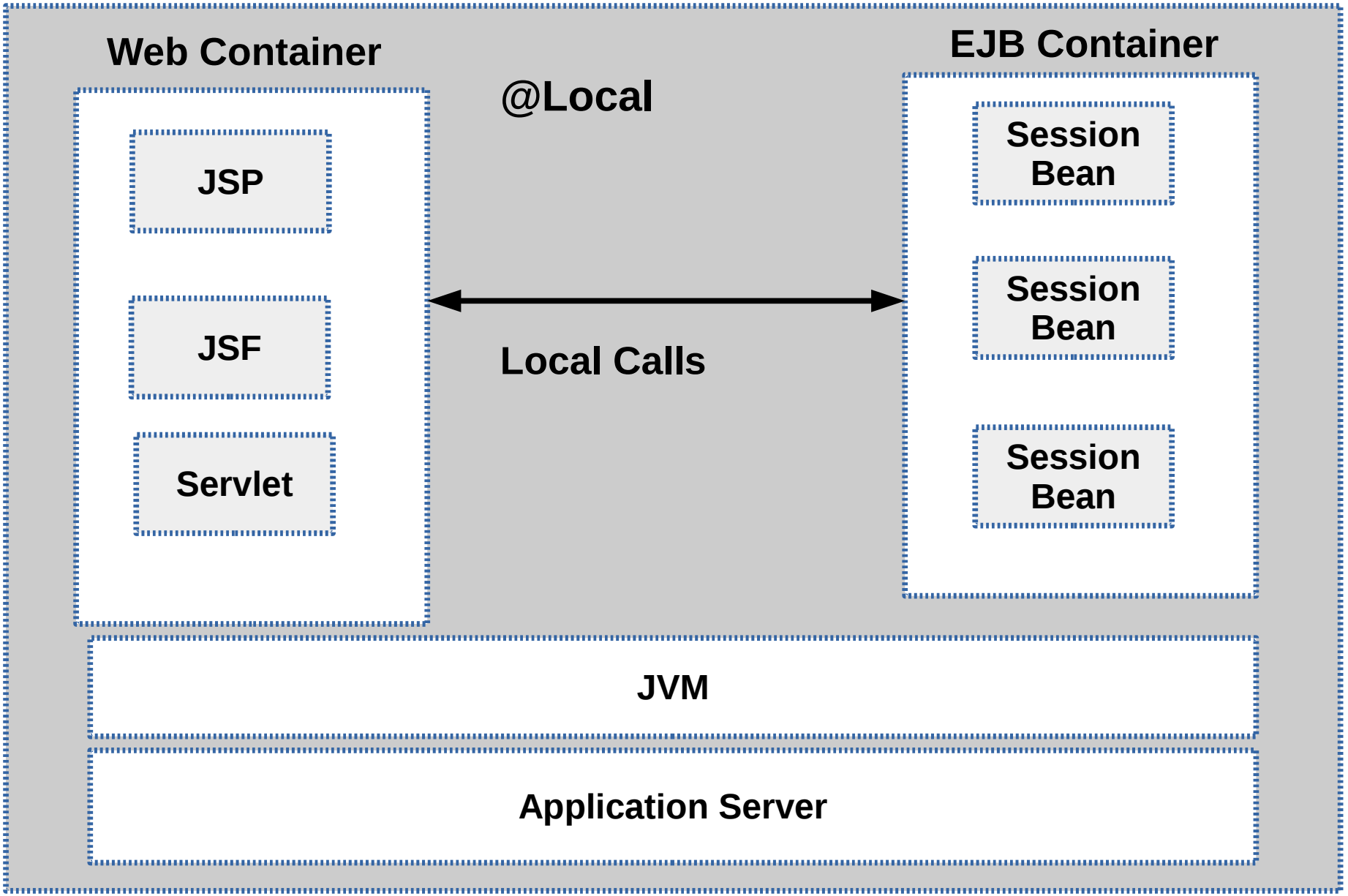
A session bean can have a business interface that is implemented by the bean class generated at design time by tools such as Jdeveloper, NetBeans, or Eclipse, or generated at deployment time by the EJB Container.

Business interfaces can use annotations.

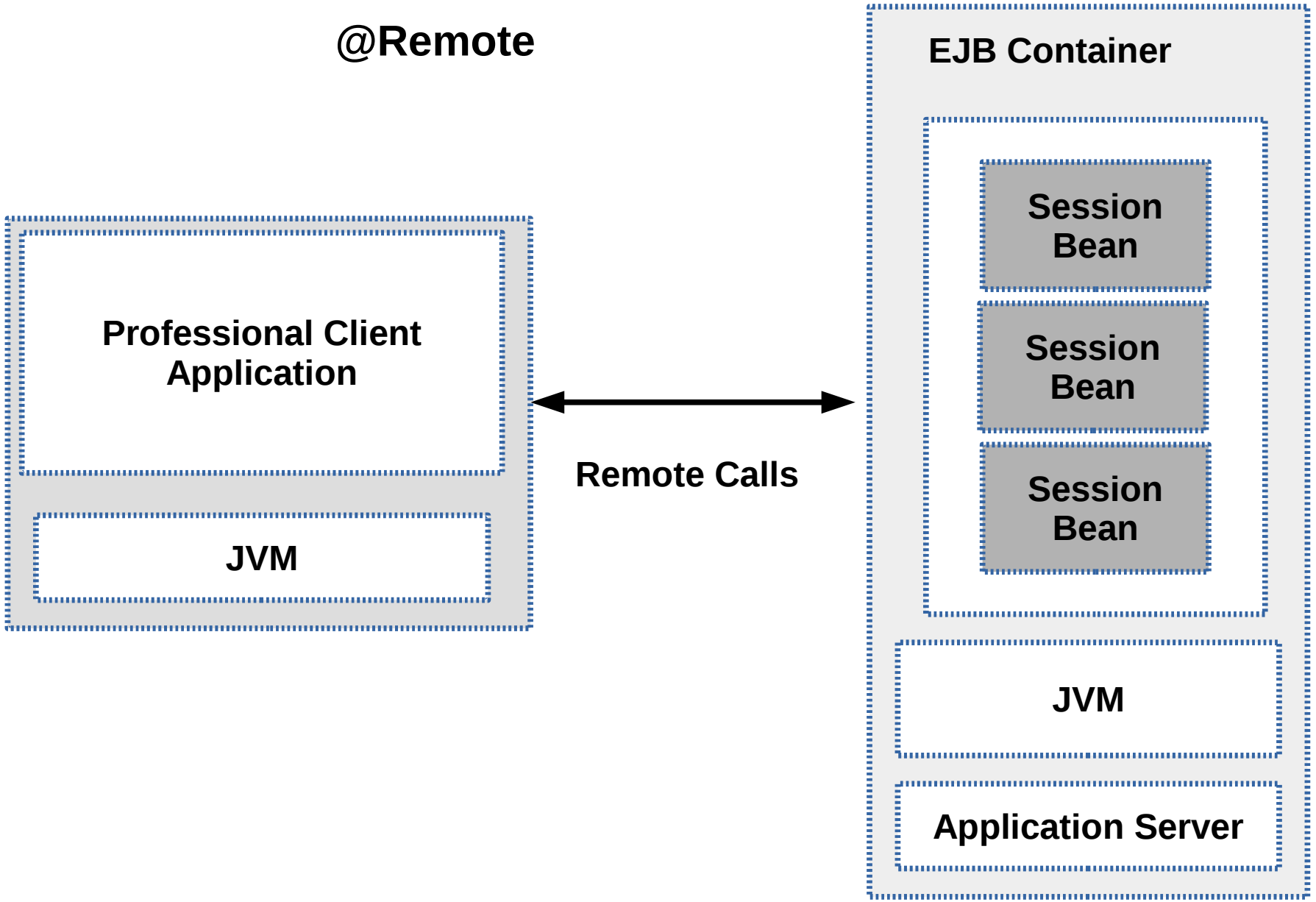
1. @Remote – remote business interface
2. @Local – local business iterface

According to designed architectural requirement whereby the client application (web application or rich client) has to run on a different JVM from on that is used to run the Session Bean in an EJB Container, then need to use remote interface. The separate JVM can be either same machine or different machine.

Session Bean Interfaces



Session Bean Interfaces



Message-Driven Bean (MDB)

- A message-driven bean is an enterprise bean that allows JEE applications to process messages asynchronously.
- This types of bean normally act as a JMS message listeners, which are similar to an event listeners, but receives JMS messages instead of events.
- The messages can be sent by any Java EE components (an application client, another enterprise bean, or a web component) or by a JMS application, or system that does not use Java EE technology. MDB can process JMS messages or other kind of messages.

Message-Driven Bean (MDB)

Difference Between MDB and Session Bean

- Main visible difference between MDB and SB is that client do not access MDB through interfaces.
- Unlike a SB a MDB has only bean class

In Several Respect a MDB Resembles a SLSB

- A MDB's instances retain no data or conversational state for a specific client.
- All instances of a MDB are equivalent, allowing the EJB Container to assign a messages to any MDB instance.
- The Container can pool these instances to allow streams of messages to be processed concurrently.
- A single MDB can process messages from multiple clients.

Message-Driven Bean (MDB)

Characteristics of MDB

- MDB execute upon receipt of a single client message.
- They are invoked asynchronously.
- They are relatively short-lived.
- MDB do not represent directly shared data in the database, but they can access and update this data.
- They can be transaction-aware.
- They are stateless.

Servers

What is Web Server ?

A Web server is a program that uses HTTP (Hypertext Transfer Protocol) to serve web documents to users, in response to their requests, which are forwarded by their computers. (HTTP clients).

Dedicated machines (computers) and relevant applications may be refered as Web Servers as well.

Ex. Apache

Servers

Where is the place Java applications developed for web purposes can be landed?

WildFly 8

Servlet Container
Web Server

EJB Container
Servlet Container
Web Container

WildFly 8

- WildFly 8 is based on modular architecture introduced in the previous release named Jboss AS 7
- It has improved on several key points including area of performance and management.
- The most important change for the developers is that this release implements the Java EE 7 standard completely.
- WildFly release has reduced the number of used ports. Now, it uses only two of them. One is (9990) for management JMX, and Web administration, and second one (8080) for standard services include HTTP, Web Sockets, remote JNDI and EJB invocation.

WildFly 8
Installation of Server

- Install JDK where WildFly will run
- Install WildFly
- Install the Eclipse development environment
- Install the Maven build management tool
- The Jboss WildFly application server can be downloaded from

  http://wildfly.org/download/
- After installation move to the WildFly Home/bin and directly run

  if windows – standalone.bat

  if Linux/Unix – standalone.sh

ucsc

Computing Services Centre