COS3043
System Fundamentals

Lecture 7

Topics

1.	**Abstractions** **1.1 Hardware Resources** **1.2 OS Functionality** **1.3 Managing the CPU and Memory**
2.	**OS Structure** **2.1 SPIN Approach** **2.2 Exokernel Approach** **2.3 L3/L4 Micro-Kernel Approach**
3.	**Virtualization** **3.1 Intro to Virtualization** **3.2 Memory Virtualization** **3.3 CPU and Device Virtualization**
4.	**Parallelism** **4.1 Shared Memory Machines** **4.2 Synchronization** **4.3 Communication** **4.4 Scheduling**
5.	**Distributed Systems** **5.1 Definitions** **5.2 Lamport Clocks** **5.3 Latency Limit**
6.	**Distributed Object Technology** **6.1 Spring Operating System** **6.2 Java RMI** **6.3 Enterprise Java Beans**
7.	**Design and Implementation of Distributed Services** **7.1 Global Memory System** **7.2 Distributed Shared Memory** **7.3 Distributed File System**
8.	**System Recovery** **8.1 Lightweight Recoverable Virtual Memory** **8.2 Rio Vista** **8.3 Quicksilver**
9.	**Internet Scale Computing** **9.1 GiantScale Services** **9.2 Content Delivery Networks** **9.3 MapReduce**
10.	**Real-Time and Multimedia** **10.1 Persistent Temporal Streams**

List of Discussion

- Spring Operating System
- Java Remote Method Invocation (RMI)
- Enterprise Java Beans (EJB)

Spring Operating System

How to Innovate OS?

Brand new OS?

Or

Better implementation of known OS?

How to Innovate OS?

Brand new OS?

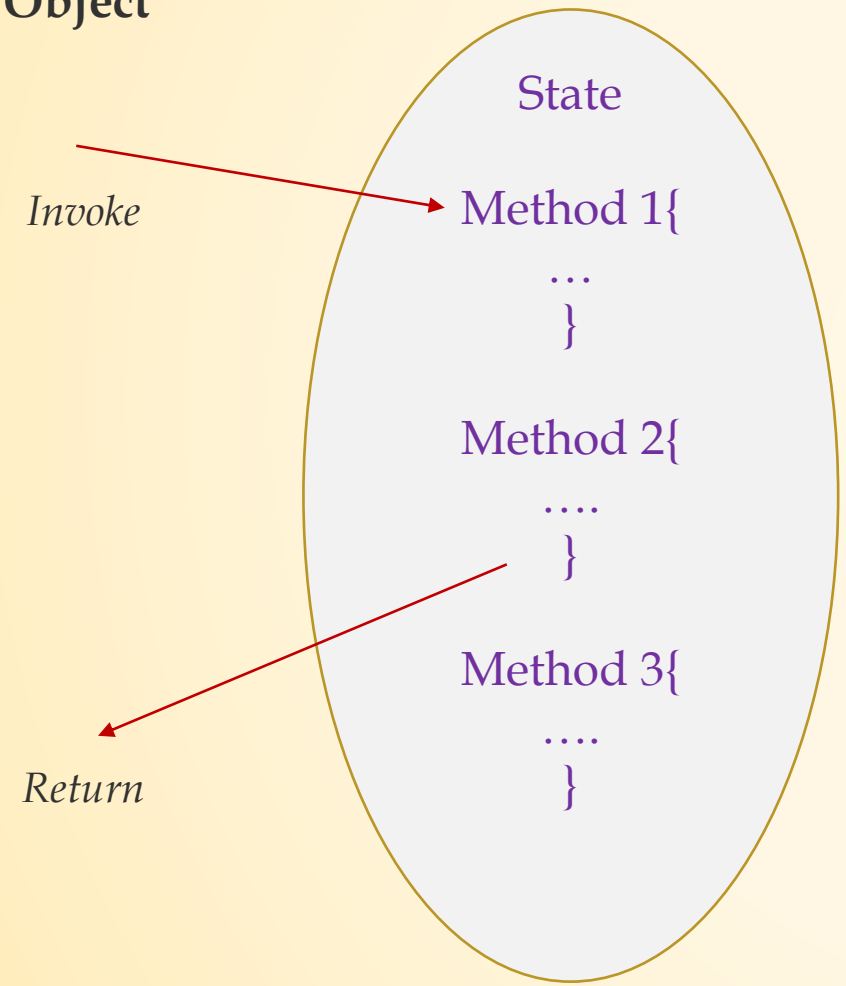
Or

Better implementation of known OS?

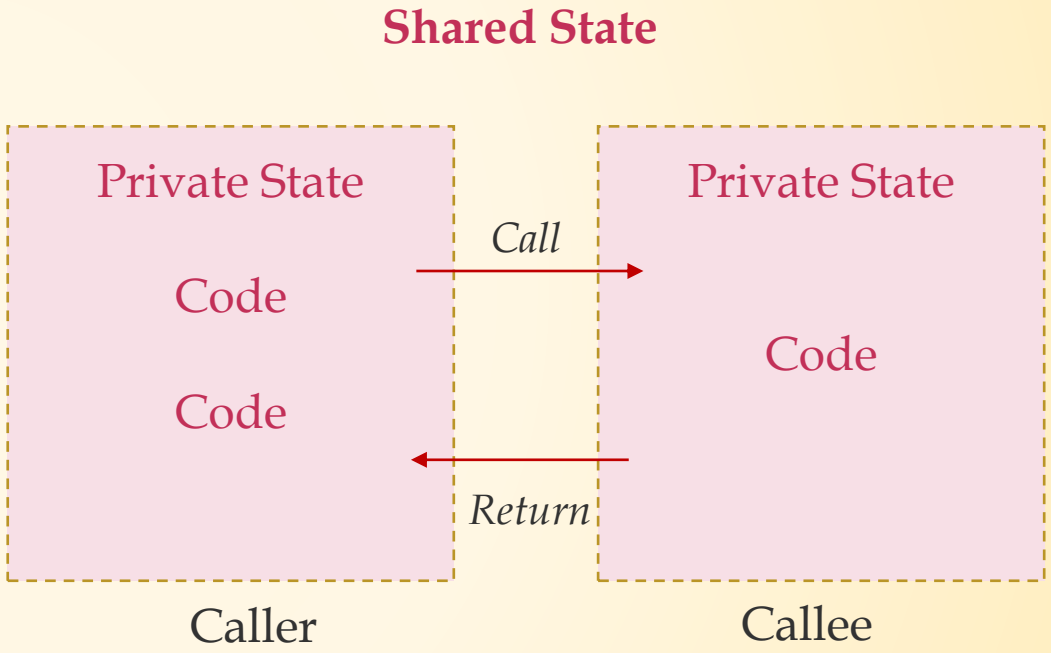
- Market place driven:
  - Large complex server software
  - Existing and ongoing software system
- Sun Microsystem => External interface remains ‘Unix’, but inside the box, innovated as and when needed.
- From the research, it seems that object based structure is the best method to achieve good OS structure design.

Object Based Vs. Procedural Design

Object

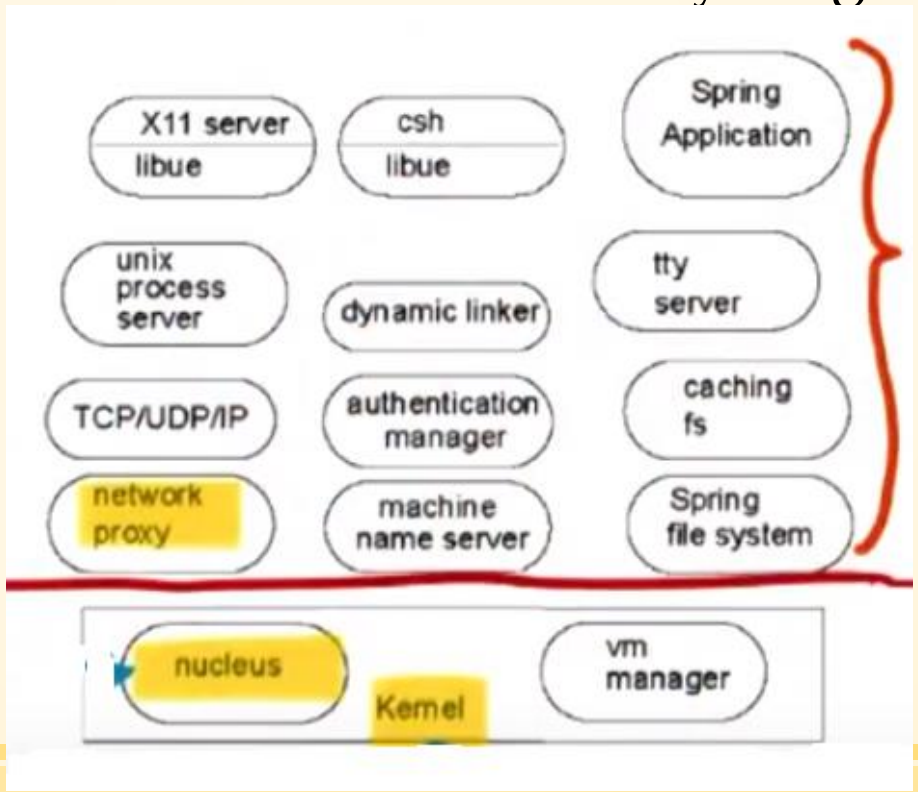


Procedural



Spring Approach

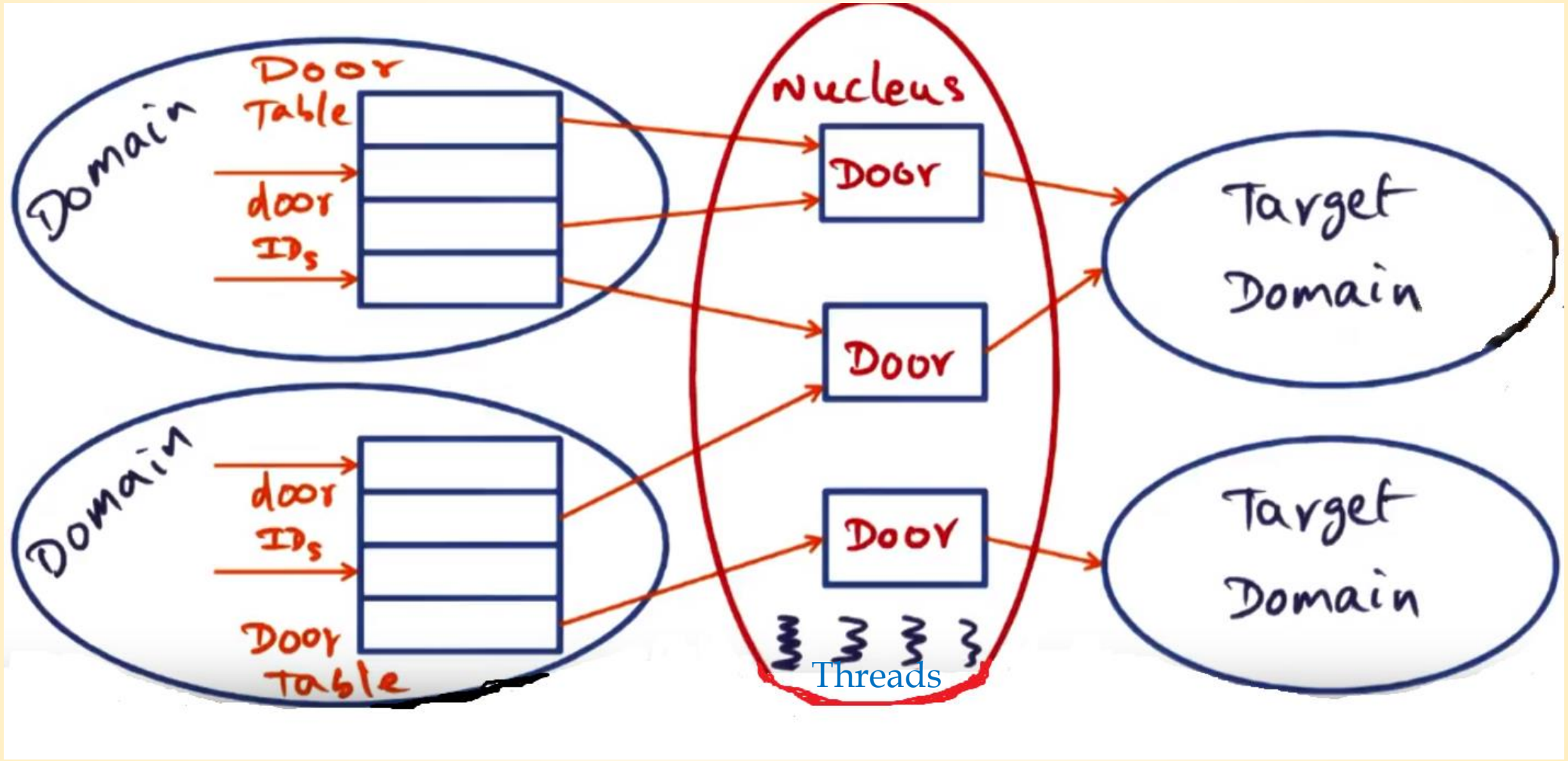
- Strong Interfaces
  - ➢Provide services to external via sub systems => object oriented.
- Open, flexible and extensible
  - ➢Open=> subs systems can be written in any languages, not tied to one.



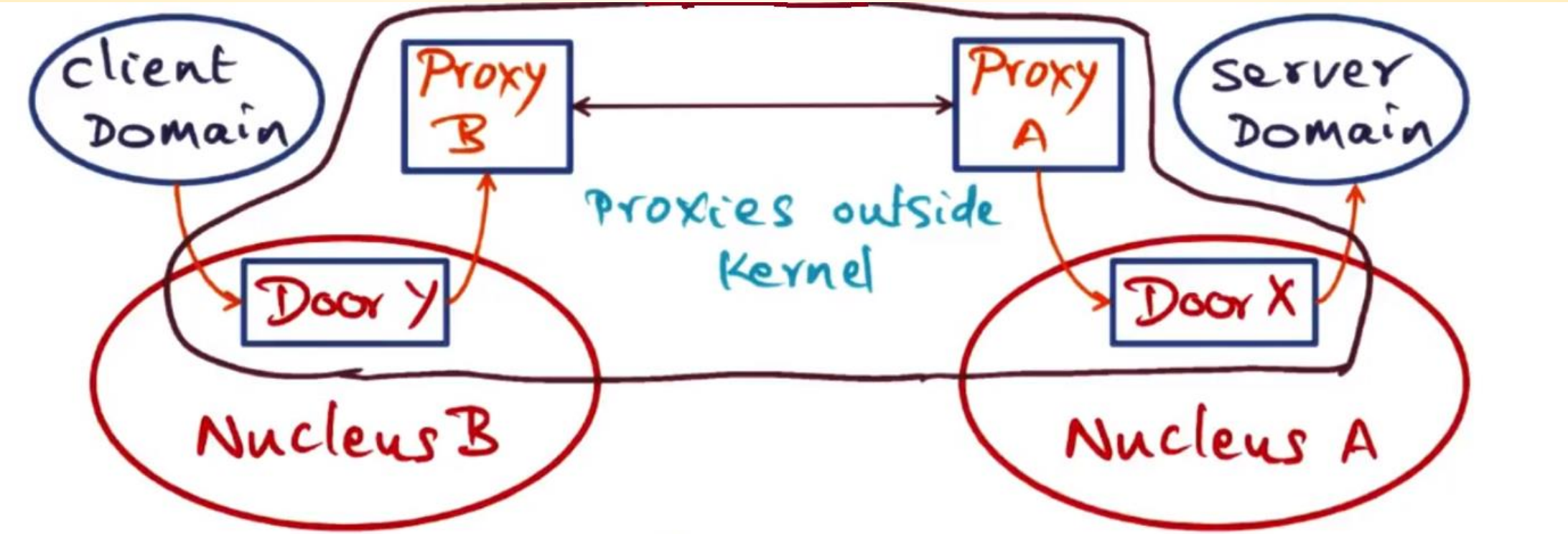
All outside the kernel

Microkernel (threads/IPC)

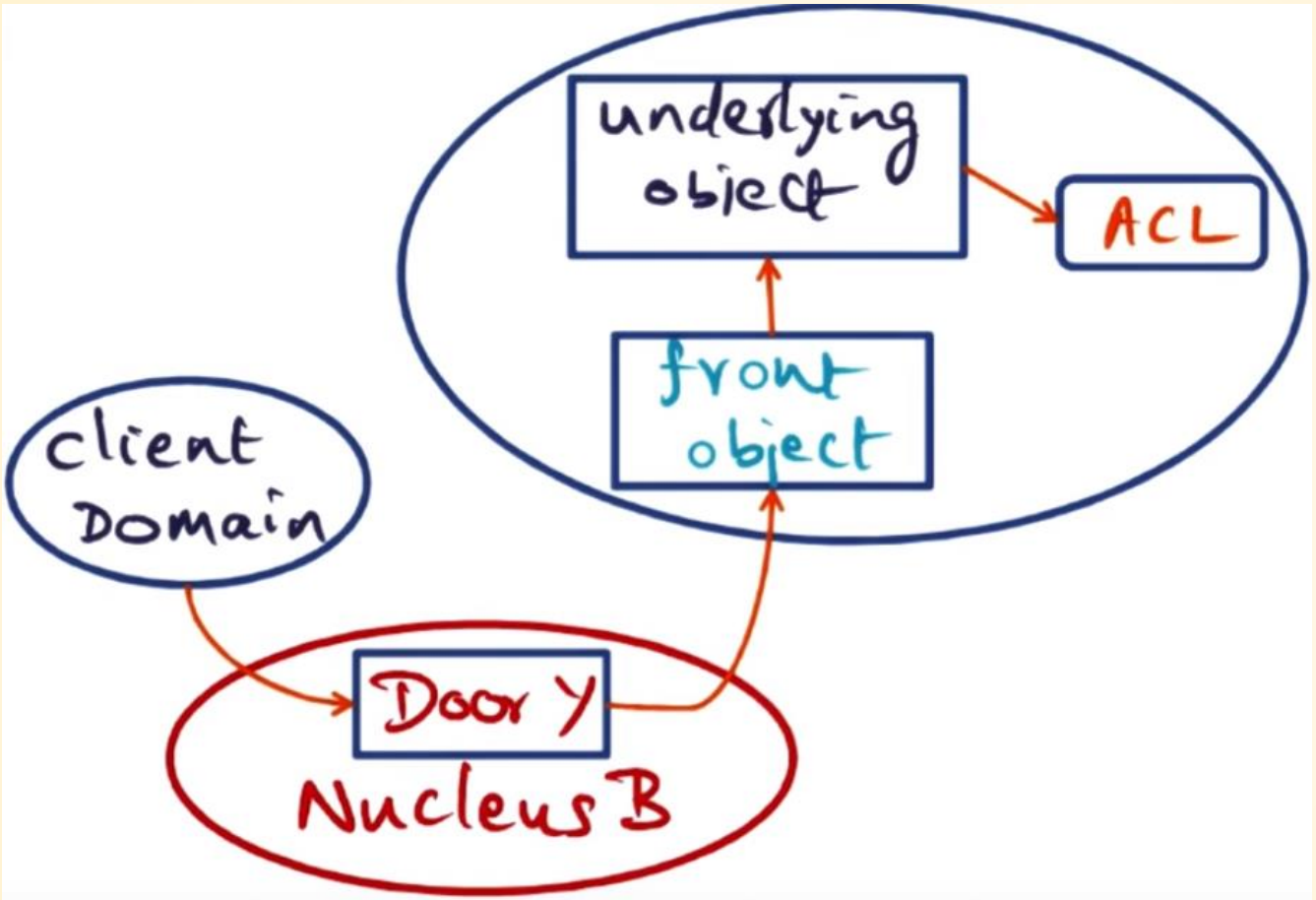
Nucleus – "Microkernel" of Spring



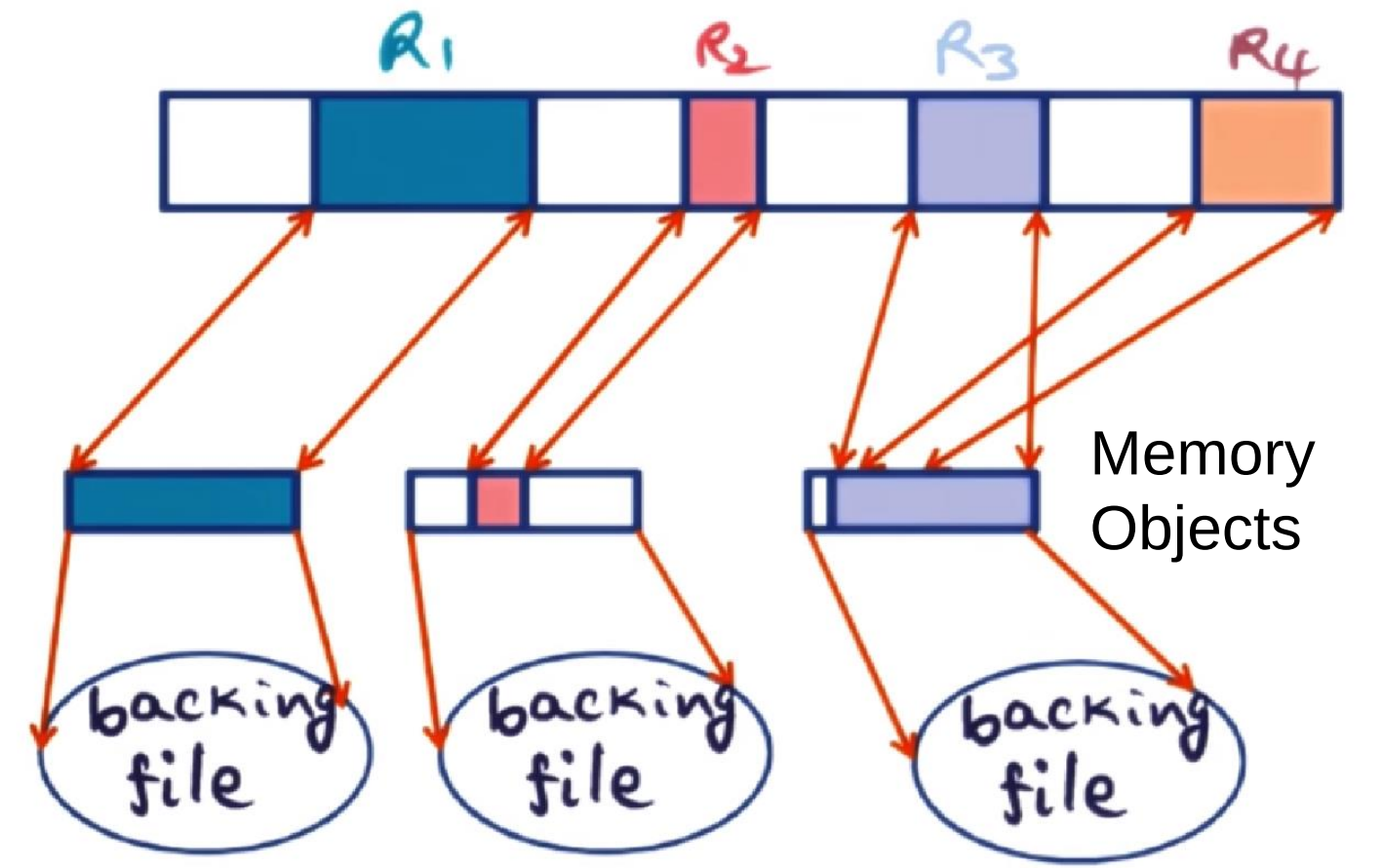
Object Invocation Across Network



Secure Object Invocation

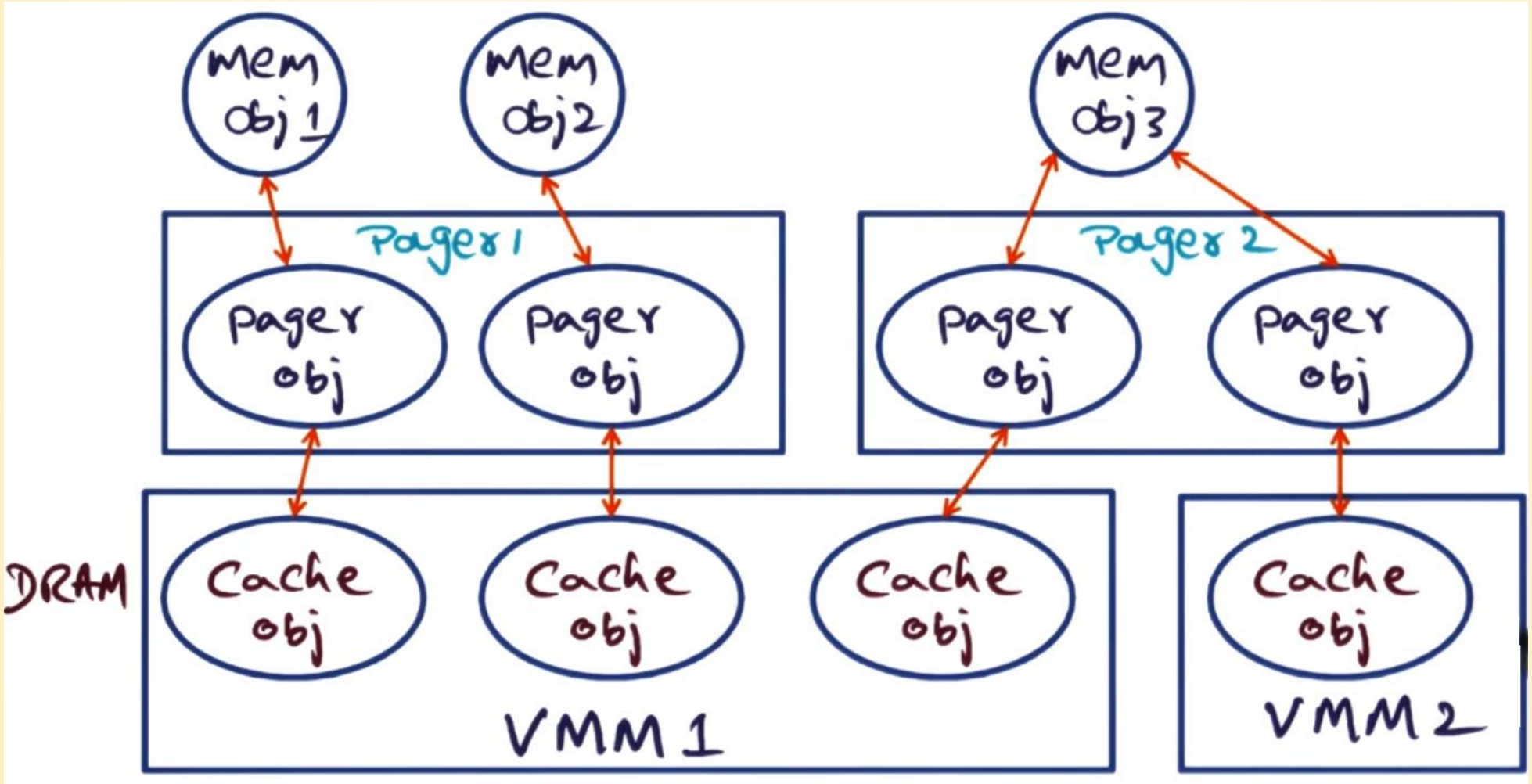


Virtual Memory Management in Spring



- Regions: set of pages (in Linear address space)
- Memory objects: abstractions may map to files, swap space, etc.

Memory Object Specific Paging



Spring System Summary

Object Oriented Kernel

- Nucleus => Threads + IPC
- Microkernel => Nucleus + Address Space
- Door + Door Table => Basis for cross domain calls
- Object invocation and cross machines calls
- Virtual Memory Management => address space objects, memory objects, external pages, cached objects

Java Remote Method Invocation (RMI)

Java History

- Invented by James Gosling at Sun Microsystems
- Initially called Oak, later changed to Java
- Originally intension is for the use of PDA as embedded software.
- Then eventually moved to Internet powering e-commerce.

Java Distributed Object Model

- Remote Object
  - Object that are accessible from different address spaces
- Remote Interface
  - Declaration for methods in a remote object
- Failure Semantics
  - Clients deal with RMI exceptions
- Similarity/Difference to Local Objects
  - Similarity: Object references can be parameters
  - Difference: Parameters only as value/result (for remote).

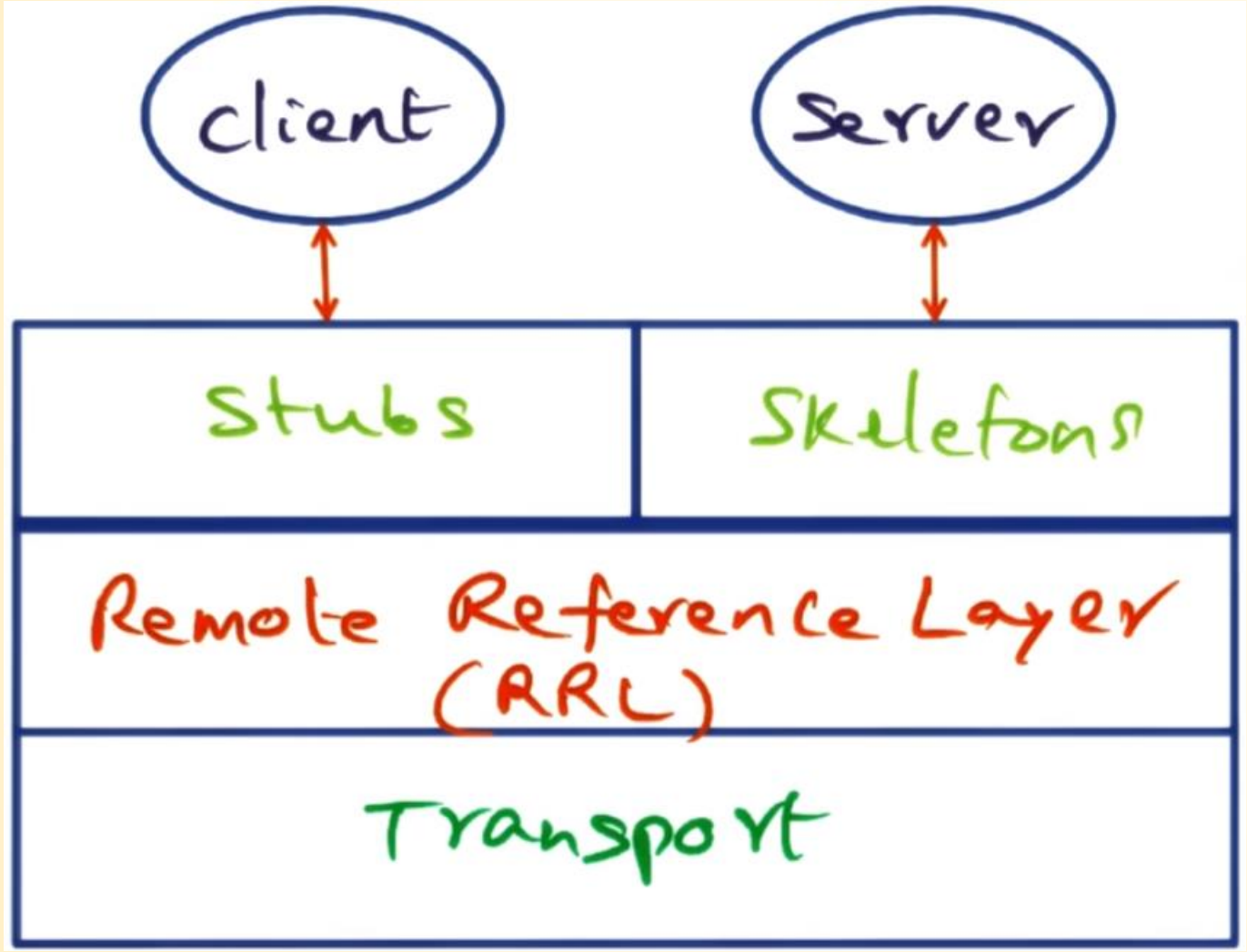
Java RMI at Work (Server)

```
BankAccount acct = new BankAcctImp();

URL url = new URL (“MyWebAddress”);

Java.rmi.Naming.bind = (url, acct);
```

Java RMI at Work (Client)

```
BankAccount acct = Java.rmi.Naming.lookup(url) ;


float balance;
acct.deposit ($$);
acct.withdraw ($$);
balance = acct.balance ();
```

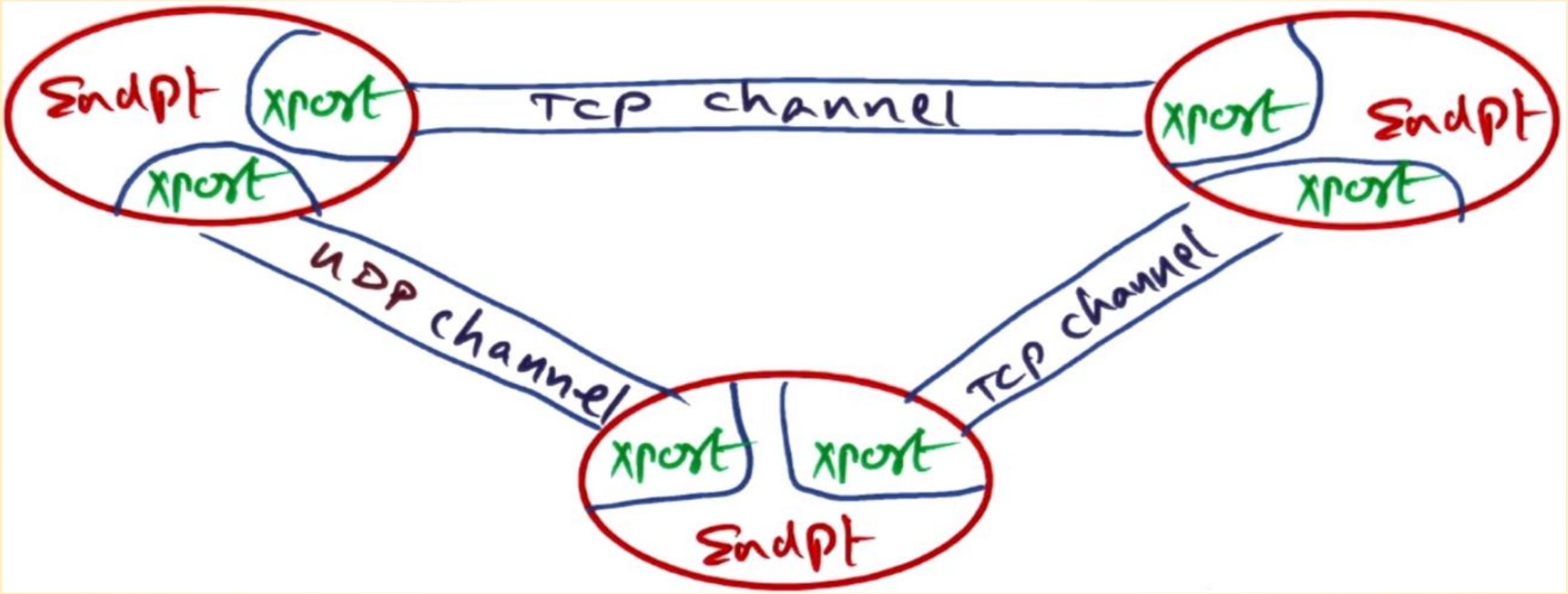
RMI Implementation - RRL



Applications

RMI System

RMI Implementation - Transport



Endpoint:
- Protection domain
- Table of remote objects

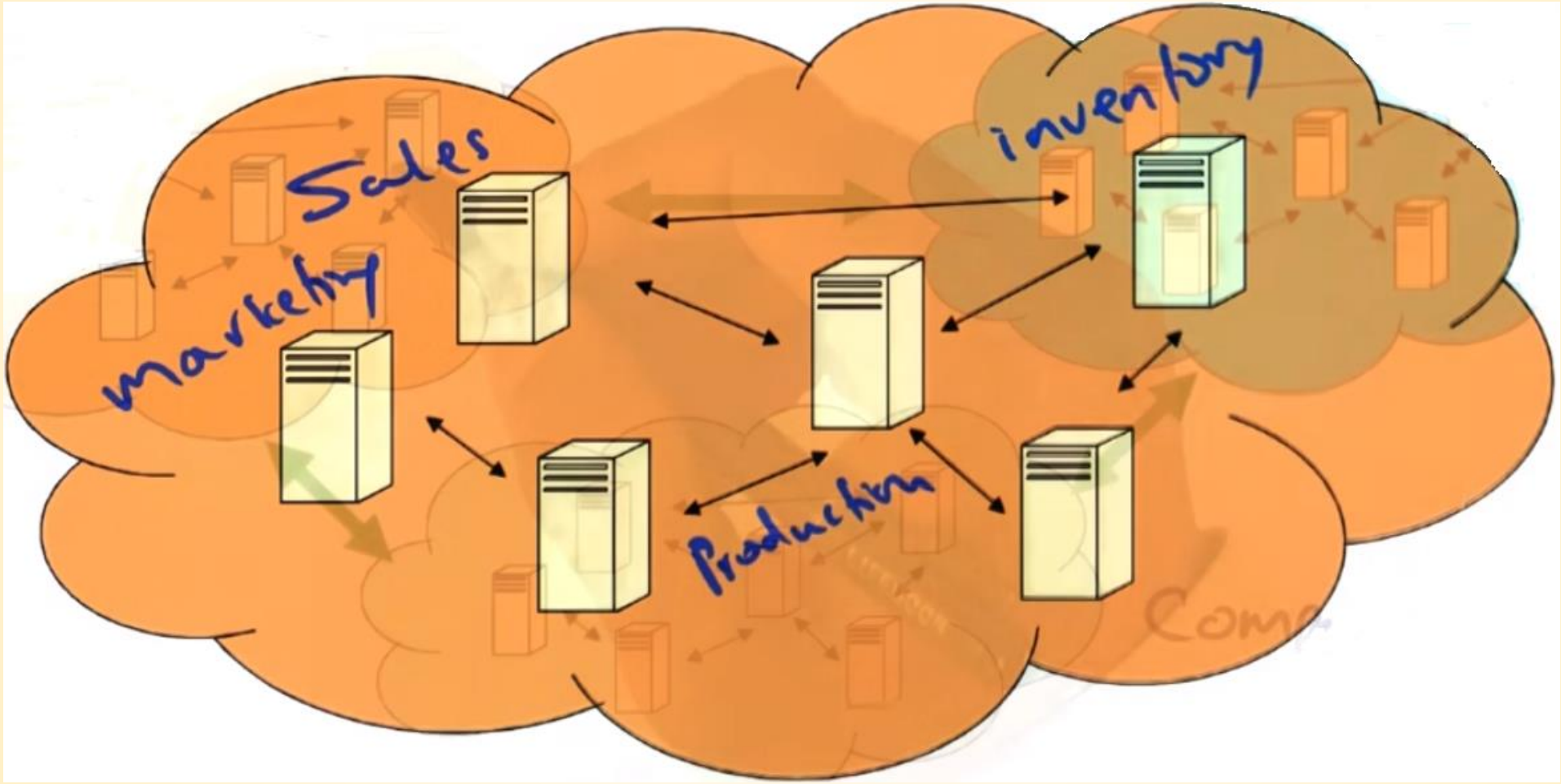
Connection Management:
- Setup, teardown, listen & establishing connections
- Liveness monitoring
- Choice of transport (xport)

Enterprise Java Beans (EJB)

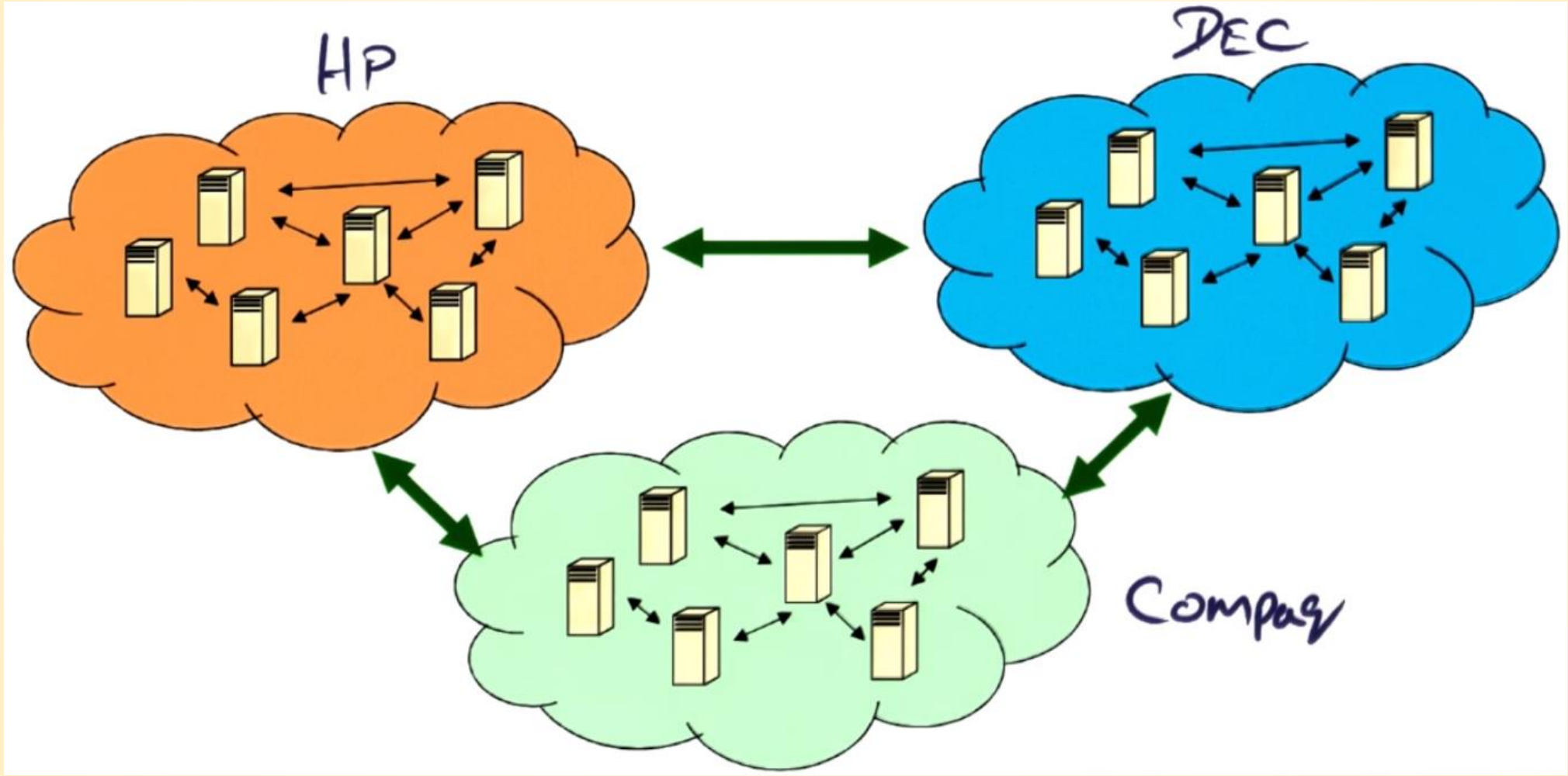
Enterprise Java Beans (EJB)

- Written in the Java programming language, an enterprise java bean is a server-side component that encapsulates the business logic of an application.
- The business logic is the code that fulfills the purpose of the application. In an inventory control application, for example, the enterprise beans might implement the business logic in methods called checkInventoryLevel and orderProduct.
- By invoking these methods, clients can access the inventory services provided by the application.

Intra Enterprise View



Inter Enterprise View



Example

Singapore

Singapore

Agoda

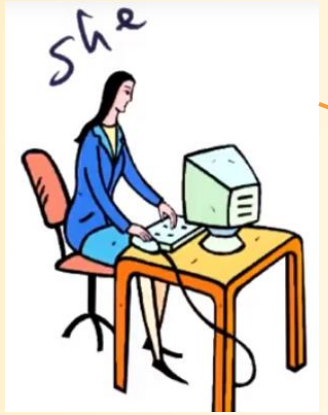
Shangri-La

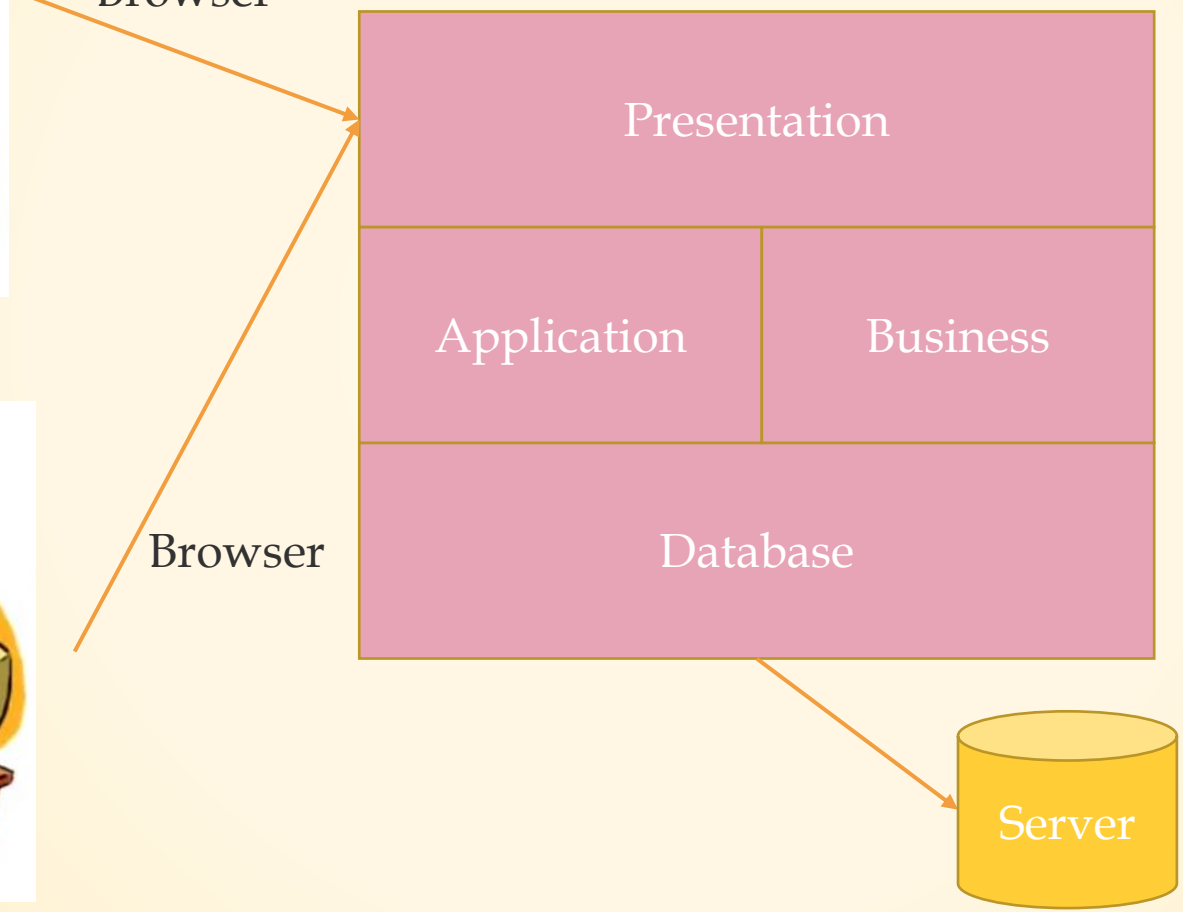
Sheraton

Westin

Others

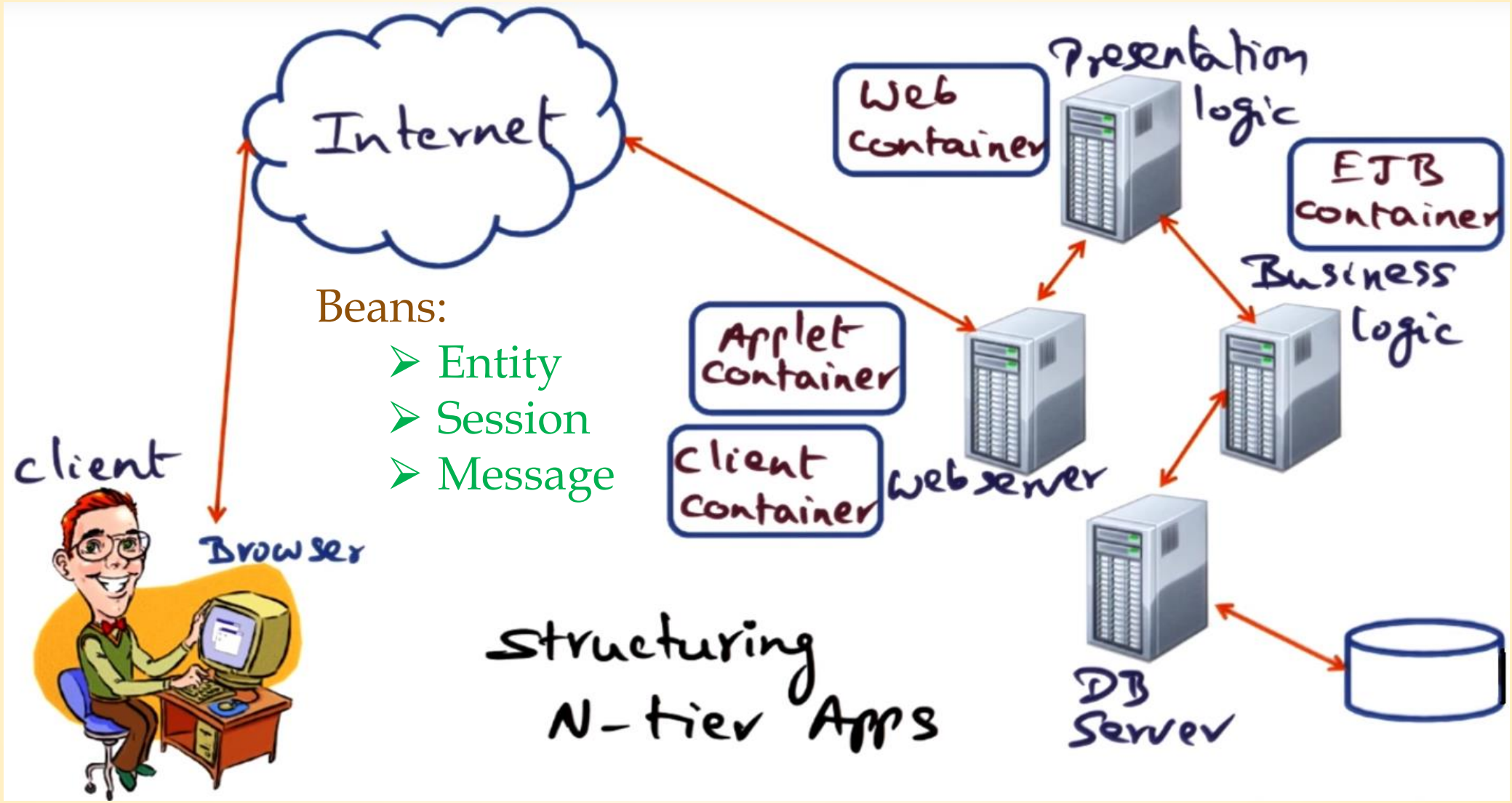
N-Tier Applications





- Persistence
- Transactions
- Caching
- Clustering
- Security

Structuring N-Tier Application



Benefits of Enterprise Java Beans

- For several reasons, enterprise beans simplify the development of large, distributed applications. First, because the EJB container provides system-level services to enterprise beans, the bean developer can concentrate on solving business problems. The EJB container, rather than the bean developer, is responsible for system-level services such as transaction management and security authorization.
- Second, because the beans rather than the clients contain the application's business logic, the client developer can focus on the presentation of the client. The client developer does not have to code the routines that implement business rules or access databases. As a result, the clients are thinner, a benefit that is particularly important for clients that run on small devices.
- Third, because enterprise beans are portable components, the application assembler can build new applications from existing beans. These applications can run on any compliant Java EE server provided that they use the standard APIs.

When to Use Enterprise Beans

- You should consider using enterprise beans if your application has any of the following requirements:
- The application must be scalable. To accommodate a growing number of users, you may need to distribute an application's components across multiple machines. Not only can the enterprise beans of an application run on different machines, but also their location will remain transparent to the clients.
- Transactions must ensure data integrity. Enterprise beans support transactions, the mechanisms that manage the concurrent access of shared objects.
- The application will have a variety of clients. With only a few lines of code, remote clients can easily locate enterprise beans. These clients can be thin, various, and numerous.