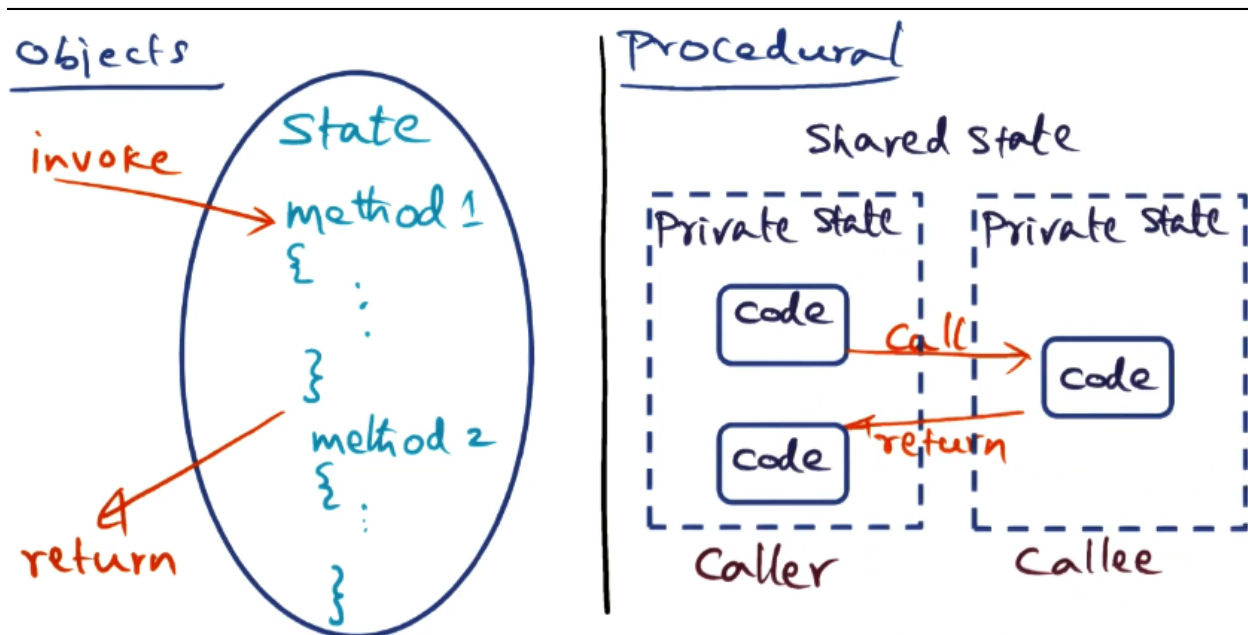


Distributed Objects and Middleware

Spring Operating System

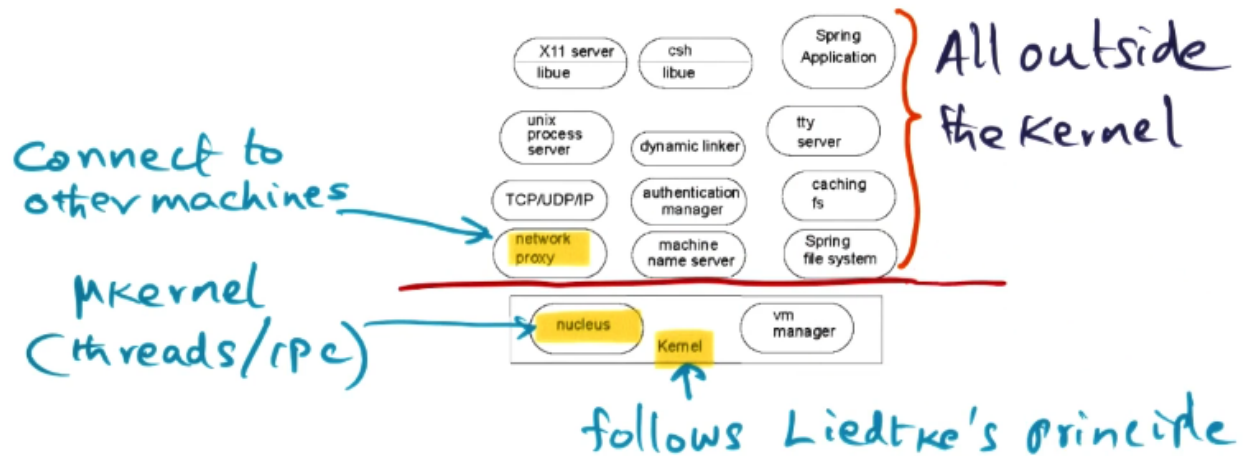
1. Spring Operating System Introduction
 - Spring became Sun's Solaris operating system
 - Created by Yousef Khalidi as a grad student at Georgia Tech
 - “Clouds” distributed operating system
2. How to Innovate OS
 - Brand new OS or better implementation of known OS?
 - Marketplace needs
 - Large complex server software
 - Legacy applications running on existing OS, so industry tends to innovate rather than reinvent
 - Intel inside: External interfaces stay the same, but there is innovation in the microarchitecture
 - Sun took this approach (“Unix inside”)
 - Externally looked like Unix, but provide new APIs for new features
 - Preserve everything good in standard OS, but also provide the capability for extensibility and flexibility
3. Object based vs Procedural Design
 - Procedural: Some shared state, some private state. Subsystems interact through procedure calls
 - State is all over the place (monolithic systems)
 - Object-based: Objects contain the state entirely within the object
 - Methods within the object to manipulate the state of the object
 - Strong interfaces that completely isolate discrete objects
 - Performance consideration: Border crossings across logical protection domains can be slow
 - Tornado also took an object-based approach to building OS kernels



Object-based vs Procedural Design

4. Spring Approach
 - Strong interfaces: Only thing exposed outside the subsystem is what services are provided by that subsystem, but not how
 - “How” can be changed at any given time
 - Leads to object-orientation

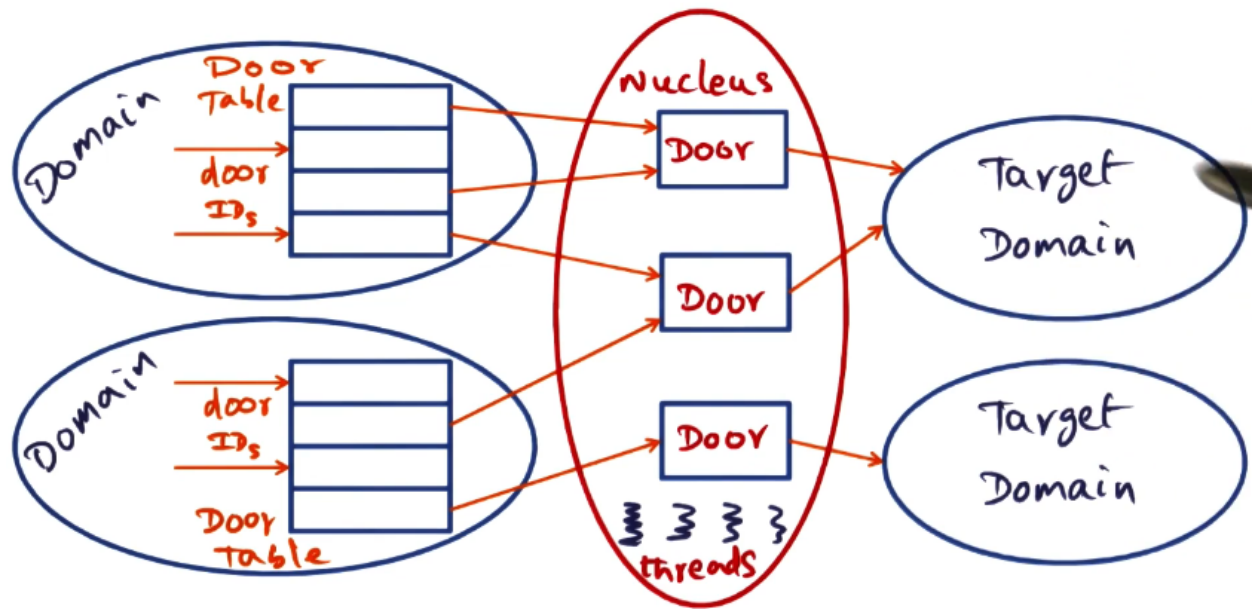
- Open, flexible, and extensible
- Don't want to be tied to a specific language
- Sun used Interface Definition Language (IDL) to expose interfaces
- Extensibility leads to microkernel-based approach
 - Nucleus provides abstractions of threads and IPC
 - Virtual memory manager provides memory management
 - Microkernel is composed of these two systems
 - Follows Liedtke's design principles
- Network proxy is interface that allows connection across machines



Spring Design

5. Nucleus Microkernel of Spring

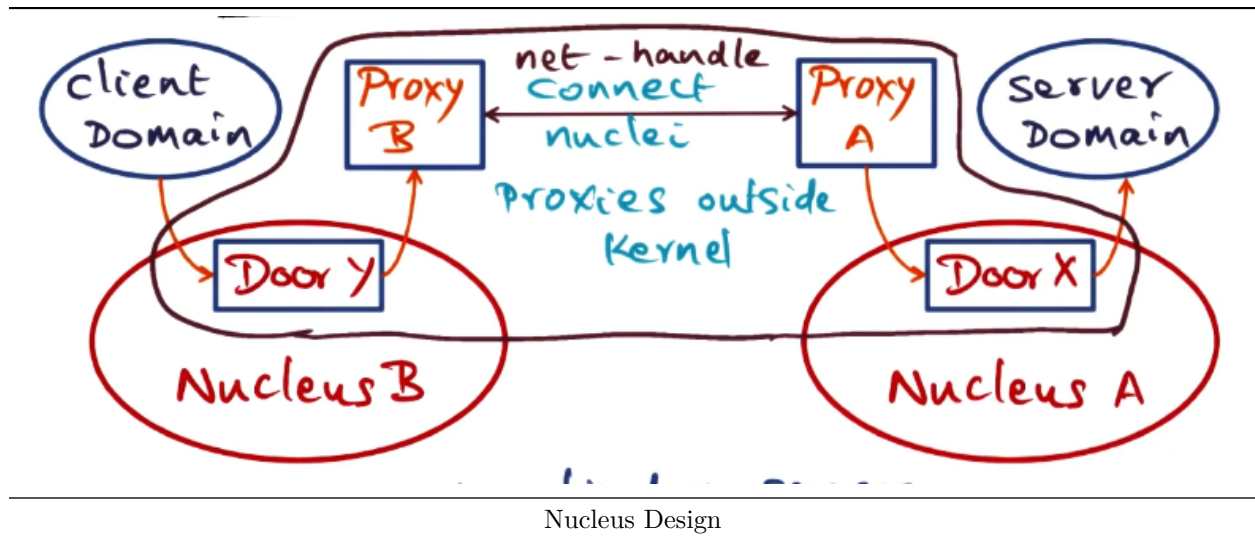
- “Microkernel” of Spring
- Only manages threads and IPC
- Domain: Similar to a Unix process (address space)
- Door: Provides access to a domain (entry points for target domain)
 - Analogy: Using `fopen` provides a file descriptor to access a file
 - Same idea, but for processes
- Nucleus is involved in every door call
 - Nucleus validates that the domain has the permission to use the door handle
 - Client thread is deactivated and the thread is allocated to the target domain so it can execute the invocation
 - Client thread is reactivated once the target domain is complete
 - This makes cross-domain calls very fast
- Kernel is composition of Nucleus plus memory manager



Nucleus Design

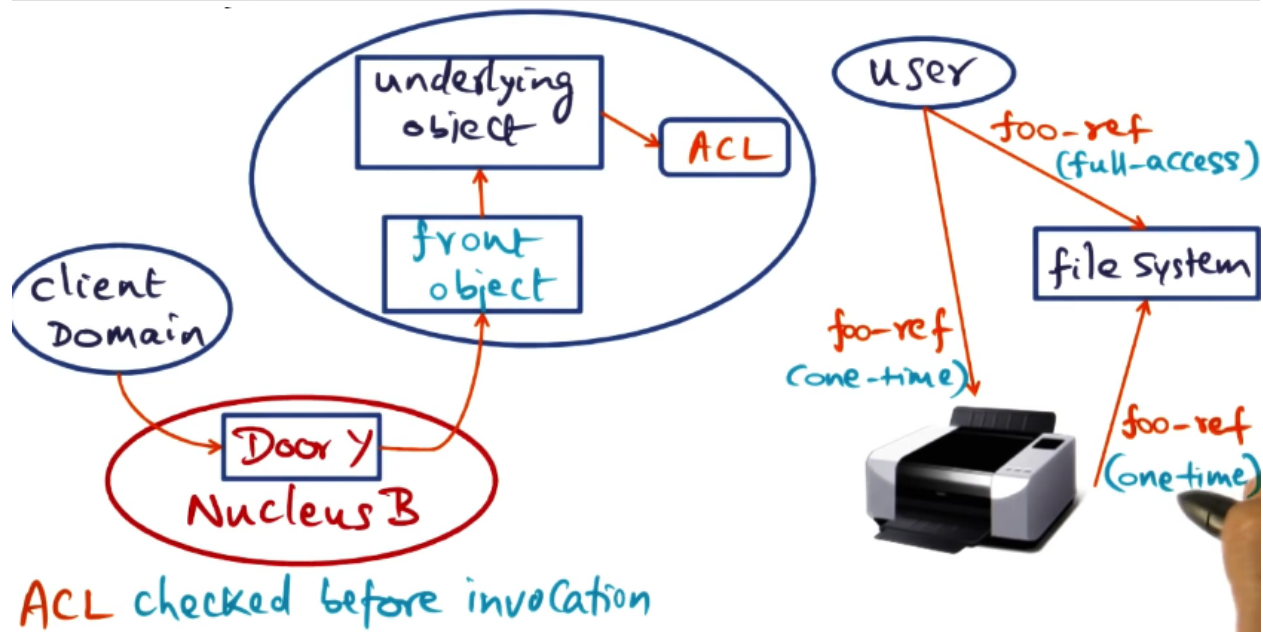
6. Object Invocation Across the Network

- Object invocation across the network is extended using network proxies
 - Proxies can potentially employ different protocols (LAN, WAN, etc)
- Key property of building network operating system
- Proxies are invisible to client and server (unaware if they are on the same machine or different machines)
- Steps for invoking an object across the network
 - Instantiate a proxy on the server node and establish a door for communication between proxy A and the server domain
 - Proxy A exports a network handle embedding door X to Nucleus A
 - Proxy B has door Y to connect to Nucleus B
 - Proxy B uses net handle to connect to nuclei
 - When the client wants to make an invocation on the server domain, it accesses proxy B which communicates with proxy A with door Y which communicates with the server domain through door X
- Proxies are outside the kernel



7. Secure Object Invocation

- Server object might need to provide different privilege levels for different clients
 - File server might provide different access protections
- Spring provides this through a “front object”
- Client domain can only access front object which arbitrates access to the underlying object through the access control list (ACL)
- Policies for accessing underlying object can be implemented through different instances of the front object
- Client can also limit access to its objects
 - Client has full access to file system
 - Client passes a one-time reference to the printing object, which passes it to the file system
- Network accesses are fast and secure due to this implementation
- External Unix interfaces are identical, but Spring has modified the underlying implementation using object technology



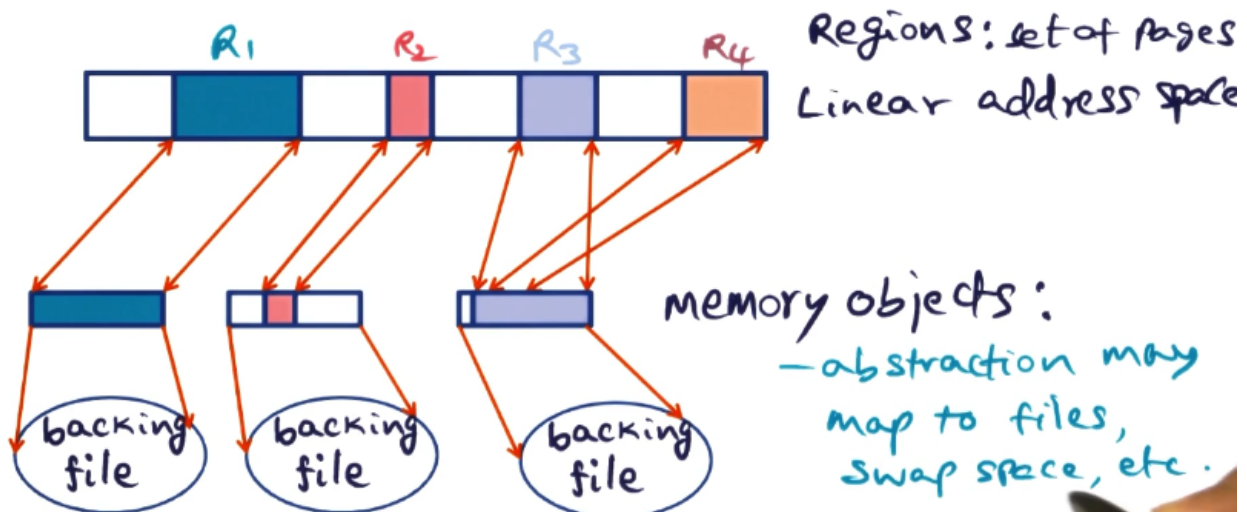
Secure Object Invocation

8. Abstractions

Feature	Nucleus	Liedtke
Threads	X	X
IPC	X	X
Add Space		X

9. Virtual Memory Management in Spring

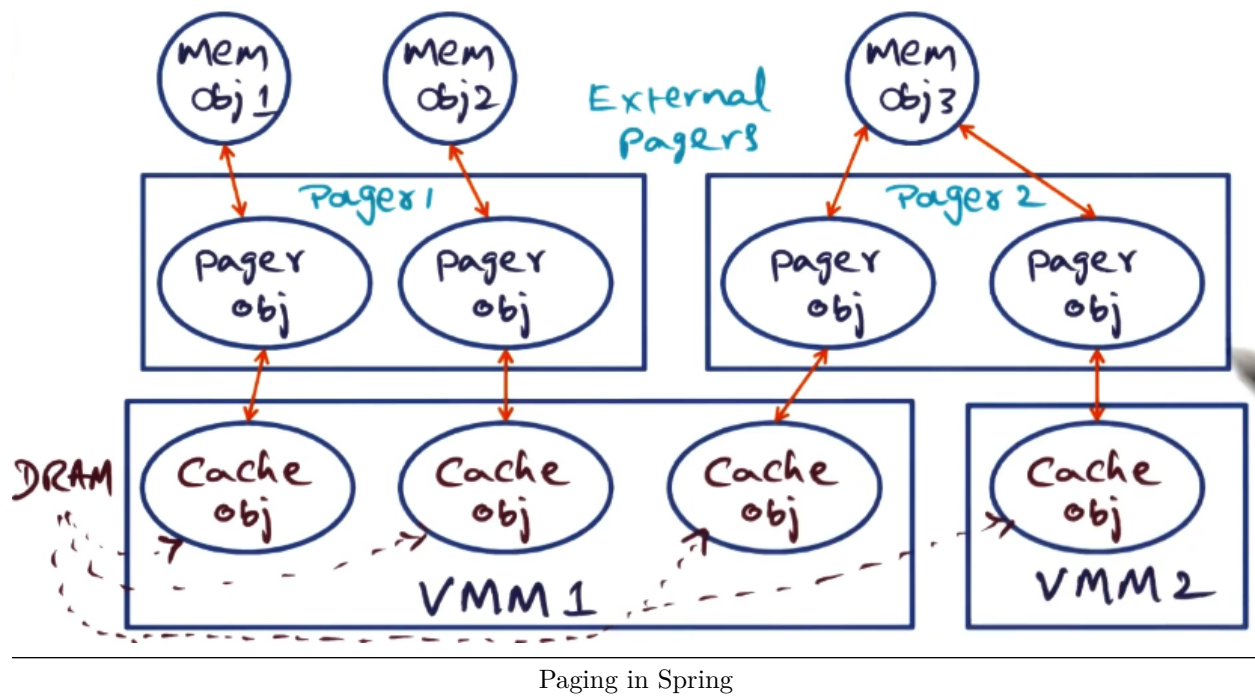
- Linear address space broken into regions (sets of pages)
- Memory objects: Abstraction that may map a portion of the address space to files, swap space, etc. (backing entities)
 - Multiple memory objects can map to the same backing entity



Memory Management in Spring

10. Memory Object Specific Paging

- Memory object: Virtual memory representation
- Pager object: Establishes connection between virtual memory and physical memory
 - Creates a cached objection representation in DRAM with the virtual memory management object
 - Address space manager can make any number of mappings
 - No single paging mechanism used for all memory objects
- Cache object: Physical memory representation
- If two VMMs have mappings to the same memory object, it is the responsibility of the pager object to coordinate the coherence of the cached objects, if this is needed
- In a single linear address space, you can have multiple pager objects managing different regions of the same address space

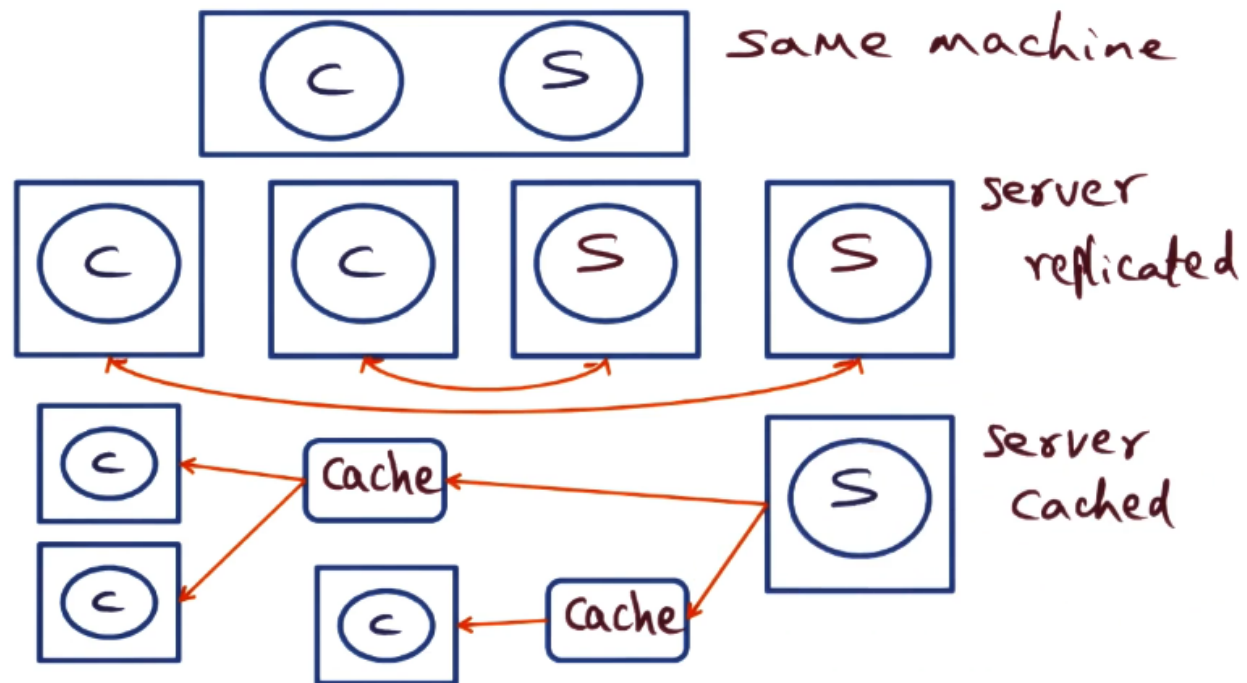


11. Spring System Summary

- Object oriented kernel
 - Nucleus: Threads and IPC
 - Microkernel: Nucleus and address space
 - Door and Doot table: Basis for cross-domain calls
 - Object invocation and cross machine calls
 - Virtual memory management: Address space object, memory object, external pages, cached objects
- Tornado also used object technology, but Tornado used clustered objects as a representation for implementing kernel services
- In Spring, object technology permeates the entire design. It's a structuring mechanism as opposed to only an optimization

12. Dynamic Client Server Relationship

- In Spring, clients and servers of the network OS can be on the same machine or different machines
 - The client/server interaction should be freed from their physical location
- Client requests are routed to different servers depending on physical proximity and current load
 - Similar to how Google works today
- Server can be cached or replicated



Dynamic Client Server Relationship in Spring

13. Subcontract

- Subcontracts make dynamic relationship between client and server possible
 - Analogous to offloading work to a third party
- Contract between client and server established through IDL
- Subcontract is the interface provided for realizing the IDL contract
- Client doesn't know or care if the server is a singleton, replicated
- Client side stub generation is simplified
- Subcontract responsible for details
- Can seamlessly add functionality to existing services using the subcontract interface
- Abstracts the underlying server implementation from the client

14. Subcontract Interface for Stubs

- Subcontract specifies if server is on same machine, different machine, different processor on the same machine, etc
- Three mechanisms
 - Marshaling/unmarshaling for arguments: Subcontract marshals appropriately based on where the server is, abstracted from client
 - On client
 - * Invoke: Calls function on server
 - On server
 - * Create: Makes new service
 - * Revoke: Remove a service
 - * Process: Ready to process invocation requests

15. Spring Operating System Conclusion

- Sun was still building Unix boxes, but had completely revolutionized the structure of the network operating system through object technology
- This formed the basis for Java RMI

Java RMI

1. Java History

- Invented by James Gosling at Sun
- Originally called Oak, later Java
- Originally invented for use on embedded devices in the early 1990s
- Intended for use with PDAs, then to set-top boxes, and then onto the Internet for powering E-commerce

2. Java Distributed Object Model

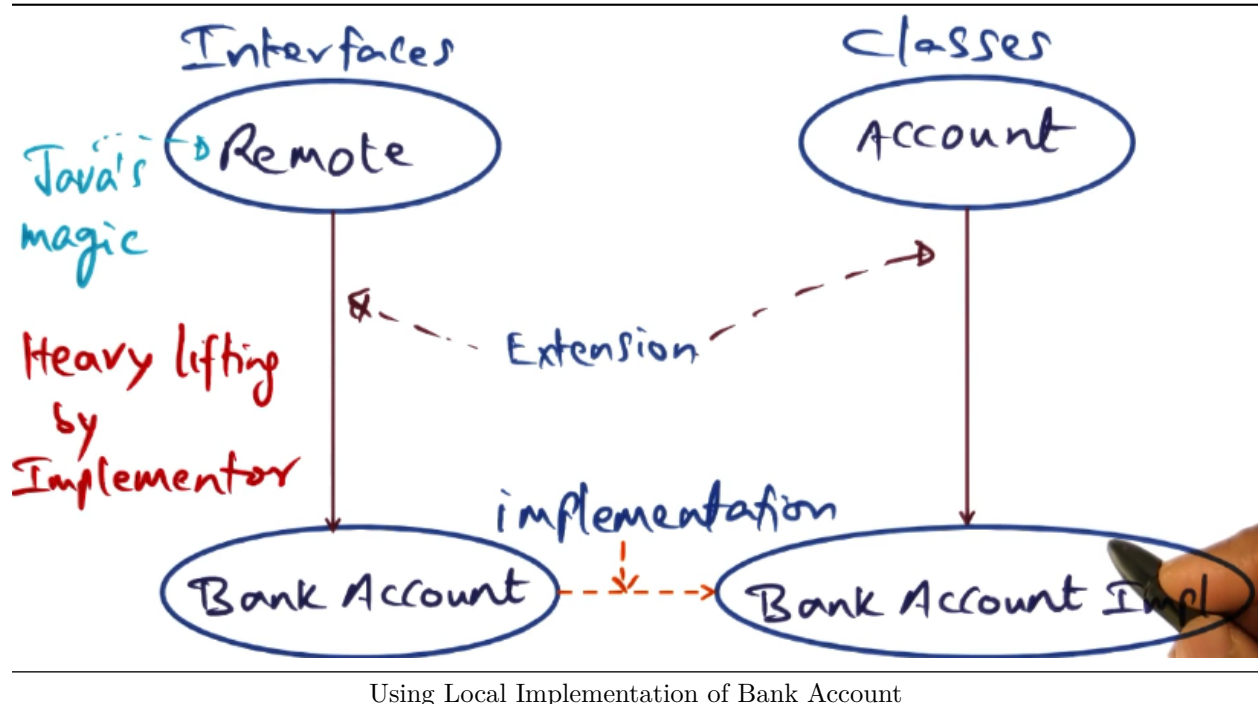
- Much of the heavy lifting involved in creating a client/server system with RPC (marshaling, unmarshaling, network interface) are subsumed by the Java distributed object runtime
- Remote object: Objects that are accessible from different address spaces
- Remote interface: Declarations for methods in a remote object
- Failure semantics: Clients deal with RMI exceptions
- Similarities/differences to local objects
 - Object references can be parameters
 - Parameters only passed as value/result
 - In Java, an object can be passed to a method by reference and modified. A remote object cannot be.

3. Bank Account Example

- Server API
 - Deposit
 - Withdraw
 - Balance
- How best to implement using Java?

4. Reuse of Local Implementation

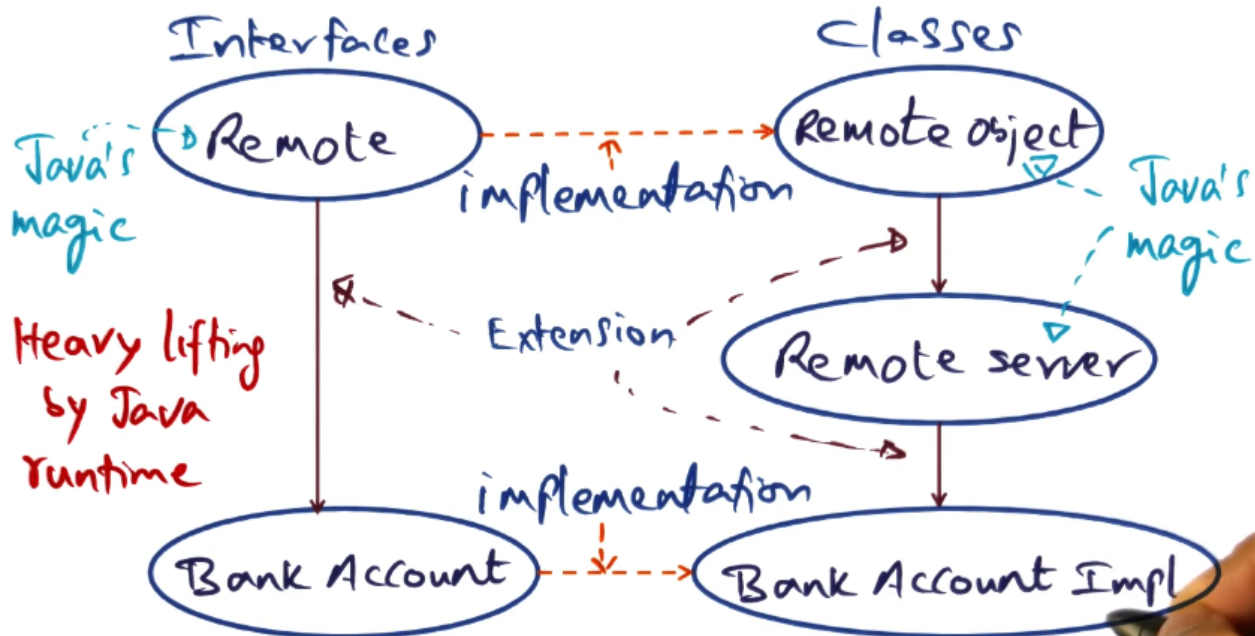
- Developer uses Java's remote interface to publish an existing local interface
- The developer has to do the heavy lifting



5. Reuse of Remote

- Developer writes the local object and uses the remote interface, identical to the local example

- The bank account implementation is derived from the Java built-in classes for remote object and remote server, not the Account superclass
- This results in the heavy lifting needed to make the bank account object visible to network clients is done by the Java runtime
- The remote implementation is preferable because the Java RMI is doing all of the heavy lifting



Using Remote Implementation of Bank Account

6. Java RMI at Work (Server)

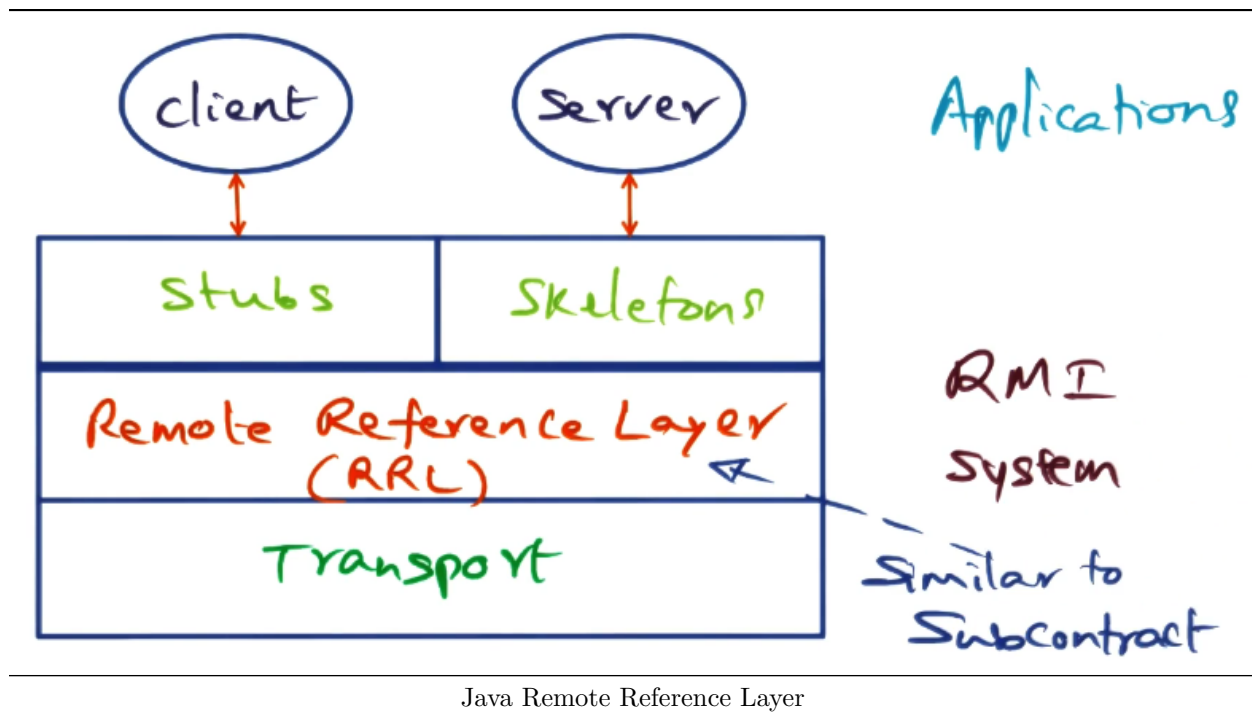
- `BankAccount acct = new BankAccountImpl();`
- `URL url = new URL("mywebaddress");`
- `Java.rmi.Naming.bind(url, acct);`

7. Java RMI at Work (Client)

- `BankAccount acct = Java.rmi.Naming.lookup(url);`
- `float balance;`
- `acct.deposit($);`
- `acct.withdraw($);`
- `balance = acct.balance();`
- Failures will throw remote exceptions back to the client
 - Client will not know where exactly the server failed

8. Java RMI Implementation (Remote Reference Layer)

- Remote Reference Layer (RRL):
- Client side stub initiates an RMI call using the RRL
 - All of the marshaling/unmarshaling is handled within the RRL
- Skeleton exists to unmarshal arguments from the client using the RRL
 - Makes the call up to the server, marshals the result, and goes through the RRL to send it
- There could be several instances of the server; the RRL handles where the server is, is it replicated, how it's handling requests, etc
- Derived from the subcontract mechanism from Spring



9. RMI Implementation Transport

- Provides the following connections:
 - Endpoint
 - Transport
 - Channel
 - Connection
- Endpoint: Protection domain (Java VM) with a table of remote objects it can access
 - Server/client can exist within this sandbox
- Connection Management
 - Setup, teardown, listen
 - Liveness monitoring
 - Choice of transport
- Transport: Listens on a channel
 - When an invocation arrives, the transport is responsible for identifying the dispatcher on the domain (endpoint)
- Channel: Medium over which communication occurs
 - TCP or UDP
- RRL decides the right transport to use depending on the location of the two endpoints

10. Java RMI Conclusion

- When the time is right, ideas from research form the basis for useful tools in industry

Enterprise Java Beans

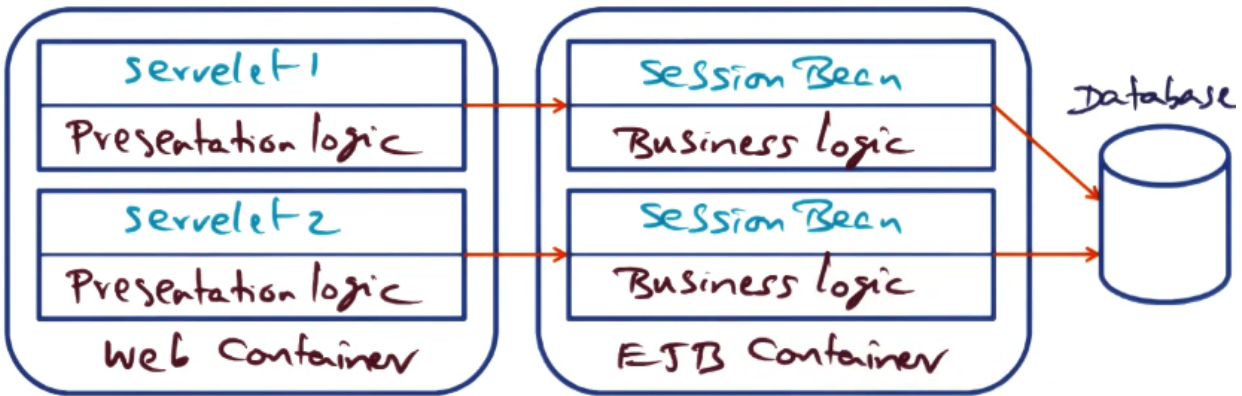
1. Enterprise Java Beans Introduction

- How do we structure the software for a large scale distributed service?
- Java bean: Many Java objects in a bundle that can be passed easily between applications

2. Inter Enterprise View

- Intra Enterprise View: Many services and servers interconnected
- Inter Enterprise View: Service may not be serviced by a single entity. Instead, many enterprises work together
 - Supply chain model

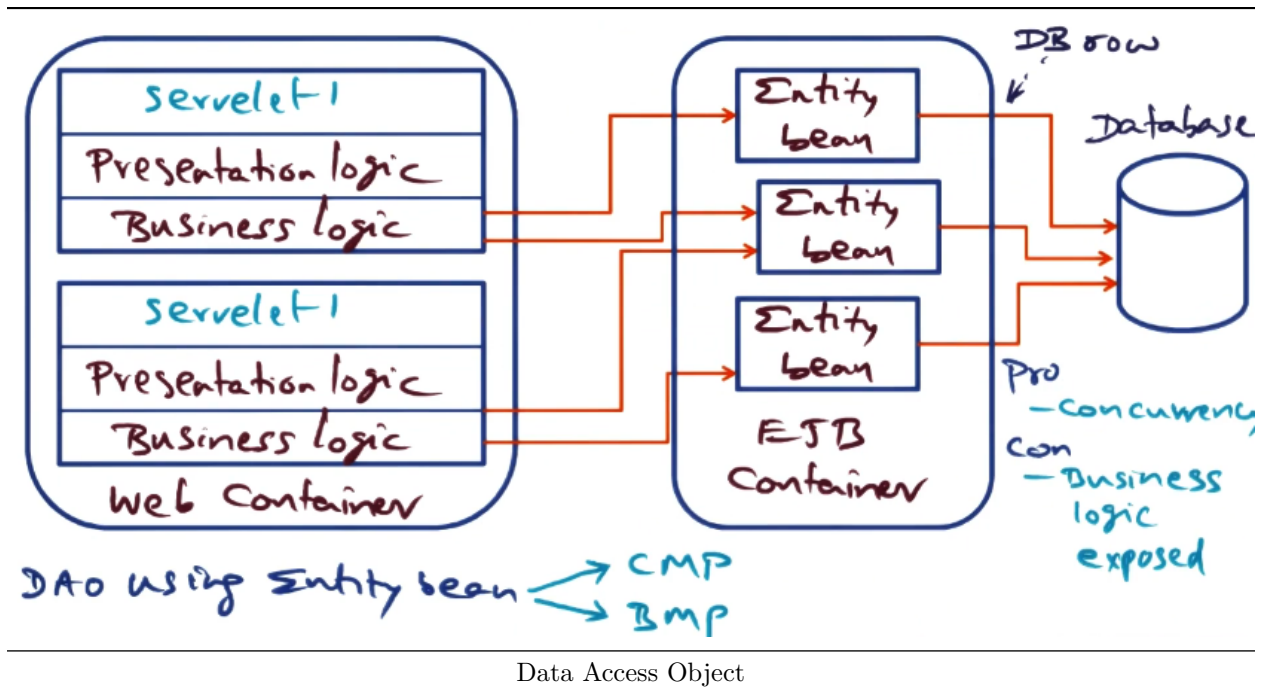
- Enterprise challenges
 - Interoperability
 - Interface compatibility
 - System evolution
 - Scalability
 - Reliability
- 3. Enterprise Java Beans Example
 - Giant scale services: Airline reservation, Gmail, web surfing as opposed to a local file service
 - There are many common features across services, such as a shopping cart
 - Don't want to reinvent the service every time
 - Object technology provides the ability to reuse components
- 4. N Tier Applications
 - Presentation layer: Paints the screen on browser
 - Application logic corresponding to what the service provides
 - Business logic corresponding to pricing, etc.
 - Database layer to access information for application and business logic
 - Issues in distributed programs:
 - Persistence
 - Transactions
 - Caching
 - Clustering
 - Security
 - Reduce network communication (latency), reduce security risks, improve concurrency in responding to individual requests
 - “Embarrassingly parallelizable” application
 - Desire to reuse components of application logic as much as possible
- 5. Structuring N Tier Applications
 - Using JEE framework for structuring applications
 - Container: Protection domain, typically in the form of a JVM
 - Client: Interact with browser on end client
 - Applet: Interact with browser on end client
 - Web: Presentation logic
 - EJB: Business logic (Enterprise Java Beans)
 - Bean: Unit of reuse (bundle of Java objects, such as shopping cart)
 - Entity: Row of a database (persistent objects with primary keys)
 - * Persistence can be built into the bean or container
 - Session: Associated with a client and temporal window (session)
 - * Could be stateful or stateless
 - Message: Useful for asynchronous behavior
 - Containers host beans
 - Fine-grained beans provides greater opportunity for increasing concurrency of requests
 - Makes business logic more complex
- 6. Design Alternative (Coarse-grain Session Beans)
 - One session bean responsible for the specific needs of the client it is servicing
 - Handles accesses to database
 - Multiple sessions within EJB depending on number of clients
 - Business logic is confined to the corporate network since it's within the EJB container, not the web container
 - Pros:
 - Minimal container services
 - Business logic not exposed
 - Cons:
 - App structure akin to monolithic kernel



Coarse-grain Session Beans

7. Design Alternative (Data Access Object)

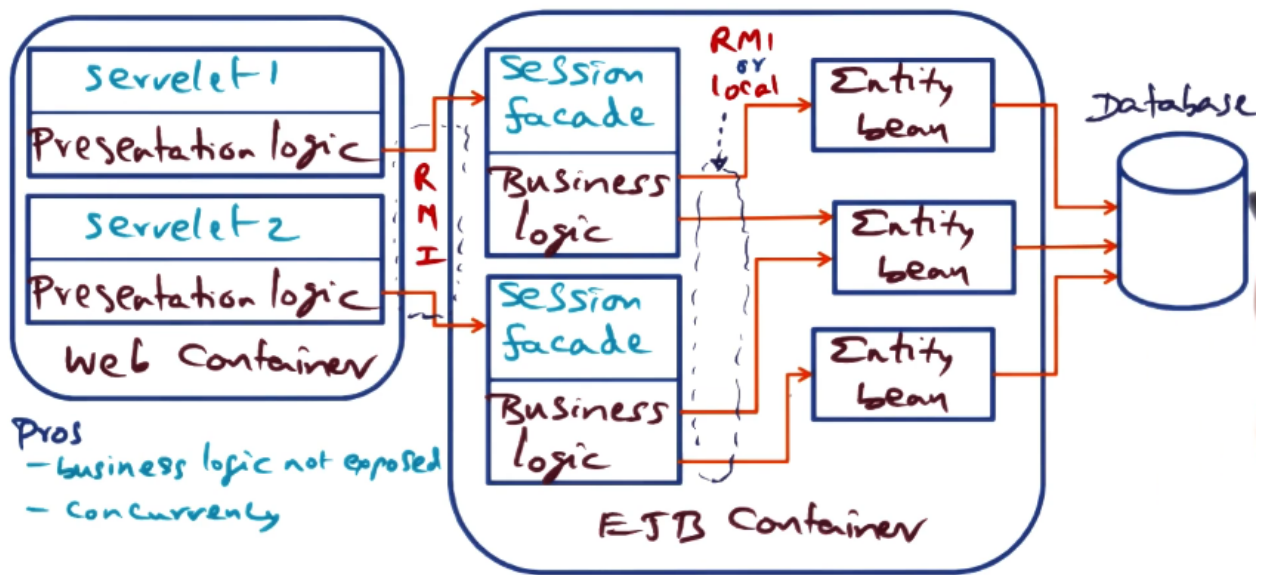
- Want parallelism in accessing database that coarse-grained doesn't provide
 - Database accesses are slow due to disk and network speeds
 - Reduce latency resulting from this slowness
- Presentation and business logic are contained within the client applet
- Database accesses are handled by entity beans in EJB container
 - Entity bean can be one or many rows of database
 - Reduces time for database access by exploiting concurrency
 - An entity bean could cluster requests and amortize database access if multiple requests are querying the same row
- Persistence is required for data access object
 - Bean managed persistence
 - Container managed persistence
- Pros
 - Concurrency
- Cons
 - Business logic exposed



Data Access Object

8. Design Alternative (Session Bean with Entity Bean)

- Use both session and entity beans to achieve parallelism without exposing the business logic
- Session facade: Worries about data access needs of associated business logic
- Can farm parallel requests to multiple entity beans
- Data access objects still use entity beans
- Web container uses RMI to communicate with the business logic
- Session facade communicates with entity bean with RMI or locally
 - Local communication eliminates the overhead of the network
- Pros
 - Business logic not exposed
 - Concurrency
- Cons
 - Additional network access, but can be mitigated by colocating session facade with entity bean



Session and Entity Bean

9. Enterprise Java Beans Conclusion

- EJB allows developers to develop business logic without being concerned with security and network interfaces