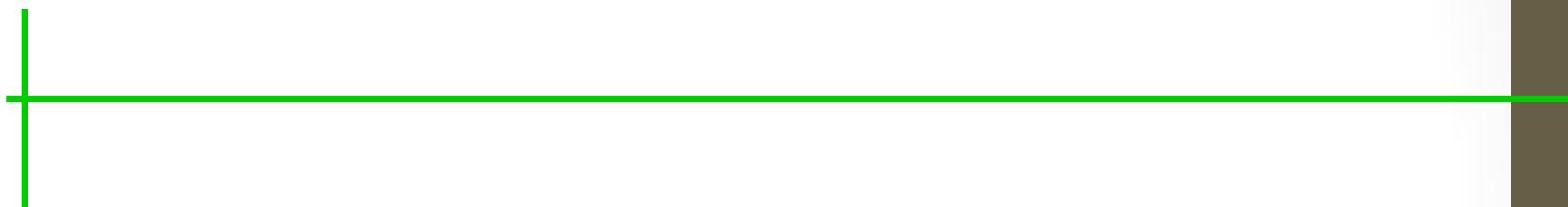


Lecture 1

Introduction to Distributed Systems



Presentation Outline

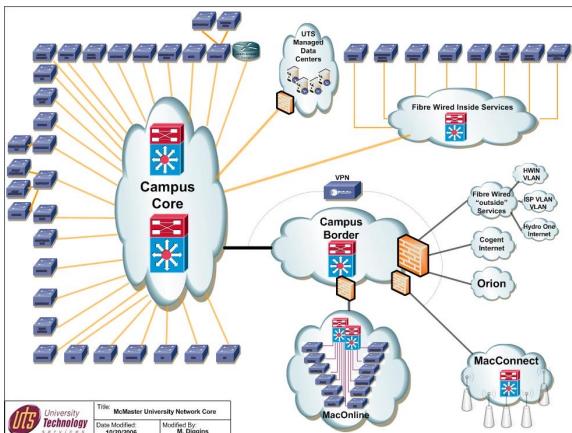
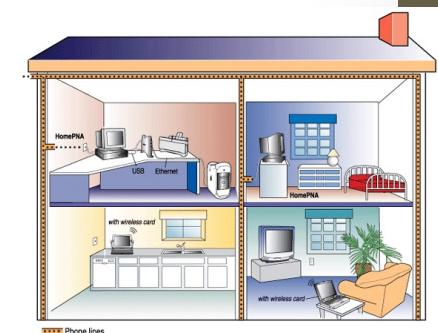
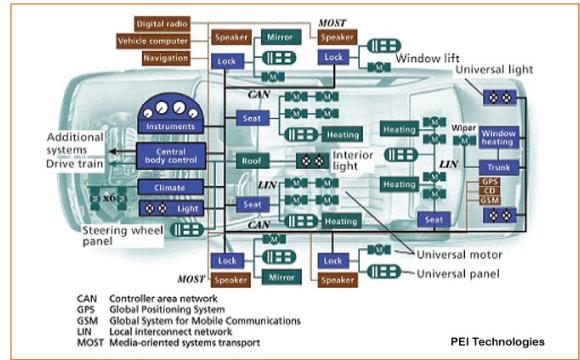
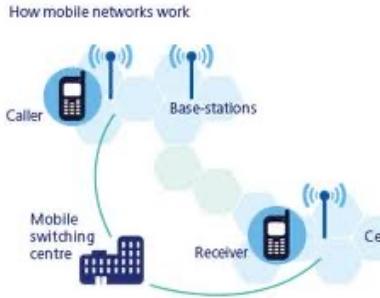
- Introduction
- Defining Distributed Systems
- Characteristics of Distributed Systems
- Example Distributed Systems
- Challenges of Distributed Systems
- Summary

Aims of this module

- Introduce the features of Distributed Systems that impact system designers and implementers
- Introduce the main concepts and techniques that have been developed to help in the tasks of designing and implementing Distributed Systems

Introduction

- Networks of computers are everywhere!
 - Mobile phone networks
 - Corporate networks
 - Factory networks
 - Campus networks
 - Home networks
 - In-car networks
 - On board networks in planes and trains

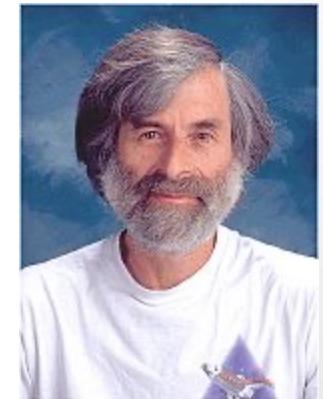


Defining Distributed Systems

- “A system in which hardware or software components located at **networked** computers communicate and coordinate their actions only by **message passing**.” [Coulouris]
- “A distributed system is a collection of **independent** computers **that appear** to the users of the system as a single computer.” [Tanenbaum]

Leslie Lamport's Definition

- *"A distributed system is one on which I **cannot** get any work done because some machine I have never heard of has crashed."*
 - Leslie Lamport – a famous researcher on timing, message ordering, and clock synchronization in distributed systems.



Networks vs. Distributed Systems

- Networks: A media for interconnecting local and wide area computers and exchange messages based on protocols. Network entities are visible and they are explicitly addressed (IP address).
- Distributed System: existence of multiple autonomous computers is transparent
- However,
 - many problems (e.g., openness, reliability) in common, but at different levels.
 - Networks focuses on packets, routing, etc., whereas distributed systems focus on applications.
 - Every distributed system relies on services provided by a computer network.

Distributed Systems

Computer Networks

Reasons for having Distributed Systems

- Functional Separation:
 - Existence of computers with different capabilities and purposes:
 - Clients and Servers
 - Data collection and data processing
- Inherent distribution:
 - Information:
 - Different information is created and maintained by different people (e.g., Web pages)
 - People
 - Computer supported collaborative work (virtual teams, engineering, virtual surgery)
 - Retail store and inventory systems for supermarket chains)

Reasons for having Distributed Systems

- Power imbalance and load variation:
 - Distribute computational load among different computers.
- Reliability:
 - Long term preservation and data backup (replication) at different locations.
- Economies:
 - Sharing resources to reduce costs and maximize utilization (e.g. network printer)
 - Building a supercomputer out of a network of computers.

Characteristics of Distributed Systems

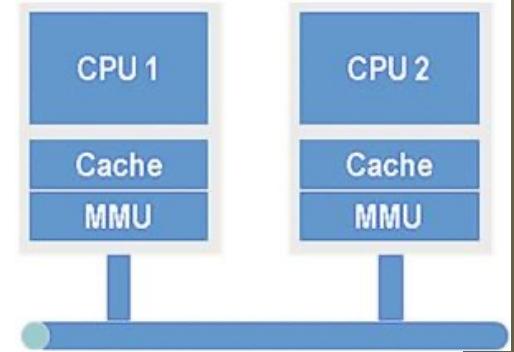
- Concurrency
 - Carry out tasks independently and parallelly
 - Tasks coordinate their actions by exchanging messages
- Communication via message passing
 - No shared memory
- Resource sharing
 - Printer, database, other services
- No global state
 - No single process can have knowledge of the current global state of the system

Characteristics of Distributed Systems

- Heterogeneity – Different devices operating together
- Independent and distributed failures
- No global clock
 - Only limited precision for processes to synchronize their clocks

Differentiation with parallel systems

- Multiprocessor/Multicore systems
 - Shared memory
 - Bus-based interconnection network
 - E.g. SMPs (symmetric multiprocessors) with two or more CPUs, GPUs
- Multicomputer systems / Clusters
 - No shared memory
 - Homogeneous in hard- and software
 - Massively Parallel Processors (MPP)
 - Tightly coupled high-speed network
 - PC/Workstation clusters
 - High-speed networks/switches-based connection.

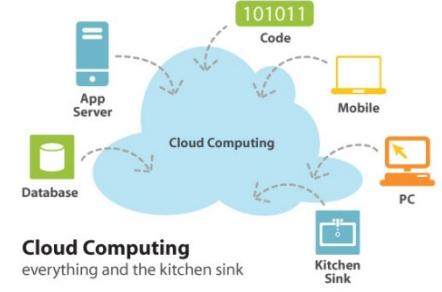


Differentiation with parallel systems is blurring

- Extensibility of clusters leads to heterogeneity
 - Adding additional nodes as requirements grow
 - Leading to the rapid convergence of various concepts of parallel and distributed systems

Examples of Distributed Systems

- They (DS) are based on familiar and widely used computer networks:
 - Internet
 - Intranets, and
 - Wireless networks
- Example DS:
 - Web (and many of its applications like Facebook)
 - Data Centers and Clouds
 - Mobile applications
 - Wide area storage systems
 - Banking Systems



Challenges with Distributed Systems

- Heterogeneity
 - Heterogeneous components must be able to interoperate
- Distribution transparency
 - Distribution should be hidden from the user as much as possible
- Fault tolerance
 - Failure of a component (partial failure) should not result in failure of the whole system
- Scalability
 - System should work efficiently with an increasing number of users
 - System performance should increase with inclusion of additional resources

Challenges with Distributed Systems

- Concurrency
 - Shared access to resources must be possible
- Openness
 - Interfaces should be publicly available to ease inclusion of new components
- Security
 - The system should only be used in the way intended

Heterogeneity

- Heterogeneous components must be able to interoperate across different:
 - Operating systems
 - Hardware architectures
 - Communication architectures
 - Programming languages
 - Software interfaces
 - Security measures
 - Information representation



Distribution Transparency

- To hide from the user and the application programmer the separation/distribution of components, so that the system is perceived as a whole rather than a collection of independent components.
- ISO Reference Model for Open Distributed Processing (ODP) identifies the following forms of transparencies:
 - **Access transparency**
 - Access to local or remote resources is identical
 - E.g. Network File System / **Dropbox**
 - **Location transparency**
 - Access without knowledge of location
 - E.g. separation of domain name from machine address.



Distribution Transparency II

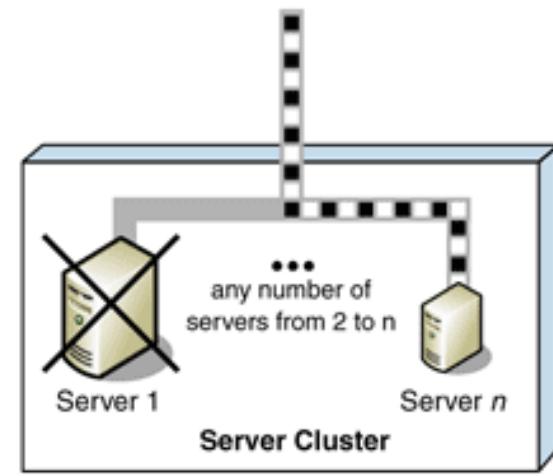
- Failure transparency
 - Tasks can be completed despite failures
 - E.g. message retransmission, failure of a Web server node should not bring down the website
- Replication transparency
 - Access to replicated resources as if there was just one. And provide enhanced reliability and performance without knowledge of the replicas by users or application programmers.
- Migration (mobility/relocation) transparency
 - Allow the movement of resources and clients within a system without affecting the operation of users or applications.
 - E.g. switching from one name server to another at runtime; migration of an agent/process from one node to another.

Distribution Transparency III

- **Concurrency transparency**
 - A process should not notice that there are others sharing the same resources
- **Performance transparency:**
 - Allows the system to be reconfigured to improve performance as loads vary
 - E.g., dynamic addition/deletion of components, switching from linear structures to hierarchical structures when the number of users increases
- **Scaling transparency:**
 - Allows the system and applications to expand in scale without changes in the system structure or the application algorithms.
- **Application level transparencies:**
 - Persistence transparency
 - Masks the deactivation and reactivation of an object
 - Transaction transparency
 - Hides the coordination required to satisfy the transactional properties of operations

Fault Tolerance

- Failure: an offered service no longer complies with its specification
- Fault: cause of a failure (e.g. crash of a component)
- Fault tolerance: no failure despite faults



Fault Tolerance Mechanisms

- Fault detection
 - Checksums, heartbeat, ...
- Fault masking
 - Retransmission of corrupted messages, redundancy, ...
- Fault toleration
 - Exception handling, timeouts,...
- Fault recovery
 - Rollback mechanisms,...

Scalability

- System should work efficiently at many different scales, ranging from a small Intranet to the Internet
- Remains effective when there is a significant increase in the number of resources and the number of users
- Challenges of designing scalable distributed systems:
 - Cost of physical resources
 - Performance Loss
 - Preventing software resources running out:
 - Numbers used to represent Internet addresses (32 bit->64bit)
 - Y2K-like problems
 - Avoiding performance bottlenecks:
 - Use of decentralized algorithms (centralized DNS to decentralized)

Concurrency

- Provide and manage concurrent access to shared resources:
 - Fair scheduling
 - Preserve dependencies (e.g. distributed transactions)
 - Avoid deadlocks
 - Preserve integrity of the system

Java Concurrency



Openness and Interoperability

- Open system:
"... a system that implements sufficient **open specifications** for interfaces, services, and supporting formats to enable properly engineered applications software to be ported across a wide range of systems with minimal changes, to interoperate with other applications on local and remote systems, and to interact with users in a style which facilitates user portability" (Guide to the POSIX Open Systems Environment, IEEE POSIX 1003.0)

Openness and Interoperability

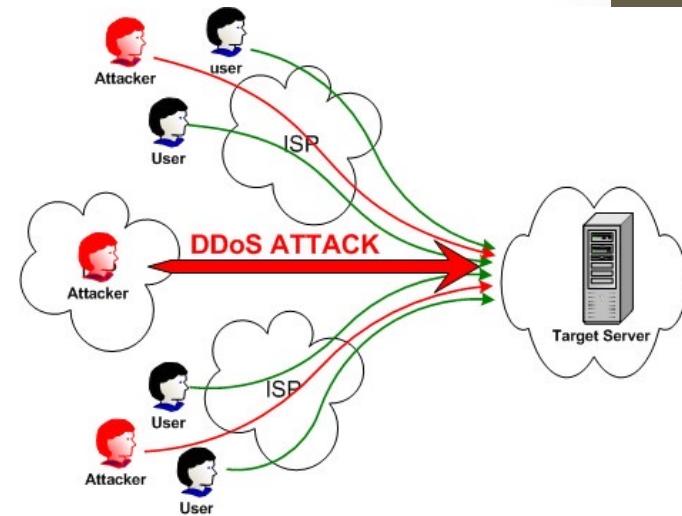
- Open message formats: e.g. XML
- Open communication protocols: e.g. HTTP, HTTPS
- Open spec/standard developers - communities:
 - ANSI, IETF, W3C, ISO, IEEE, OMG, Trade associations,...

Security I

- Resources are accessible to authorized users and used in the way they are intended
- Confidentiality
 - Protection against disclosure to unauthorized individual information
 - E.g. ACLs (access control lists) to provide authorized access to information
- Integrity
 - Protection against alteration or corruption
 - E.g. changing the account number or amount value in a money order

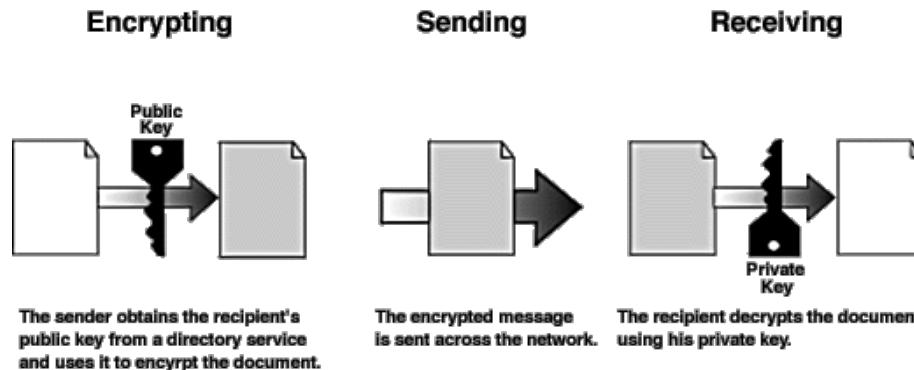
Security II

- Availability
 - Protection against interference targeting access to the resources.
 - E.g. denial of service (DoS, DDoS) attacks
- Non-repudiation
 - Proof of sending / receiving an information
 - E.g. digital signature



Security Mechanisms

- Encryption
 - E.g. Blowfish, RSA
- Authentication
 - E.g. password, public key authentication
- Authorization
 - E.g. access control lists



Business Example and Challenges

- Web/Mobile app to search and purchase online courses
 - Customers can connect their computer to your server (locally hosted or cloud hosted):
 - Browse your courses
 - Purchase courses
 - ...

Business Example – Challenges

I

- What if
 - Your customers use different devices? (Dell laptop, Android device ...)
 - Your customers use different OSs? (Android, IoS, Ubuntu...)
 - a different way of representing data? (Text, Binary,...)
 - **Heterogeneity**
- Or
 - You want to move your business and computers to China (because of the lower costs)?
 - Your client moves to a different country(more likely)?
 - **Distribution transparency**

Business Example – Challenges

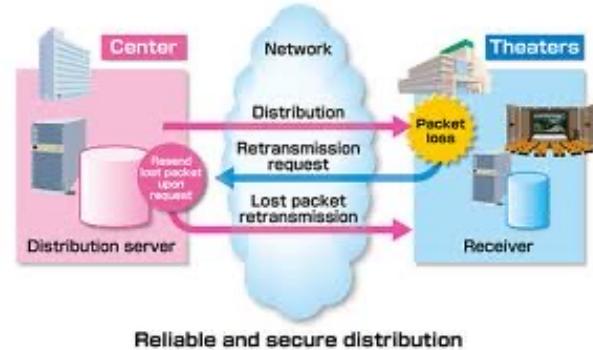
II

- What if
 - Two customers want to order the same item at the same time?
 - **Concurrency**
- Or
 - The database with your inventory information crashes?
 - Your customer's computer crashes in the middle of an order?
 - **Fault tolerance**

Business Example – Challenges

III

- What if
 - Someone tries to break into your system to alter data?
 - ... sniffs for information?
 - ... someone says they have enrolled to the course but they haven't?
 - **Security**
- Or
 - You are so successful that millions of people are using your app at the same time.
 - **Scalability**



Business Example – Challenges

IV

- When building the system...
 - Do you want to write the whole software on your own (network, database,...)?
 - What about updates, new technologies?
 - Adding a web client later on?
 - Will your system need to communicate with existing systems (e.g. payment gateways, SMS servers)
 - **Reuse and Openness (Standards)**



Impact of Distributed Systems

- New business models (e.g. Uber, Airbnb)
- Global financial markets
- Global labor markets
- E-government (decentralized administration)
- Ecommerce
- Driving force behind globalization
- Social/Cultural impact
- Media getting decentralized

Summary

- Distributed Systems are everywhere
- The Internet enables users throughout the world to access its services wherever they are located
- Resource sharing is the main motivating factor for constructing distributed systems
- Construction of DS produces many challenges:
 - Heterogeneity, Openness, Security, Scalability, Failure handling, Concurrency, and Transparency
- Distributed systems enable globalization:
 - Community (Virtual teams, organizations, social networks)
 - Science (e-Science)
 - Business (e-Business)

Lecture 2 - Distributed System Architectures

Definitions

- **Software Architectures** – describe the organization and interaction of software components; focuses on logical organization of software (component interaction, etc.)
- **System Architectures** - describe the placement of software components on physical machines

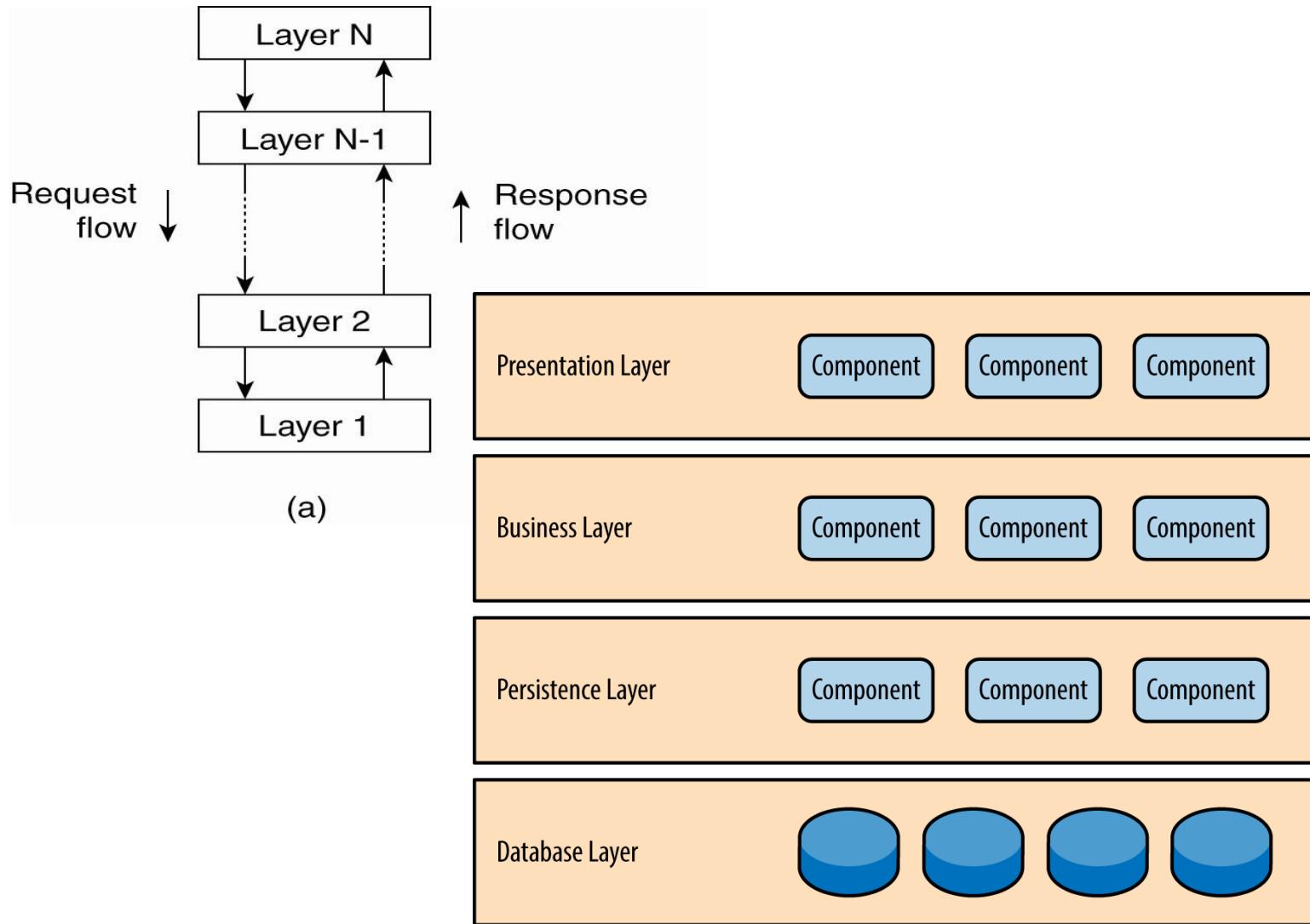
Architectural Styles

- An **architectural style** describes a particular way to configure a collection of components and connectors.
 - **Component** - a module with well-defined interfaces; reusable, replaceable
 - **Connector** – communication link between modules
- Architectures suitable for distributed systems:
 - Layered architectures*
 - Component-based architectures*
 - Data-centered architectures
 - Event-based architectures

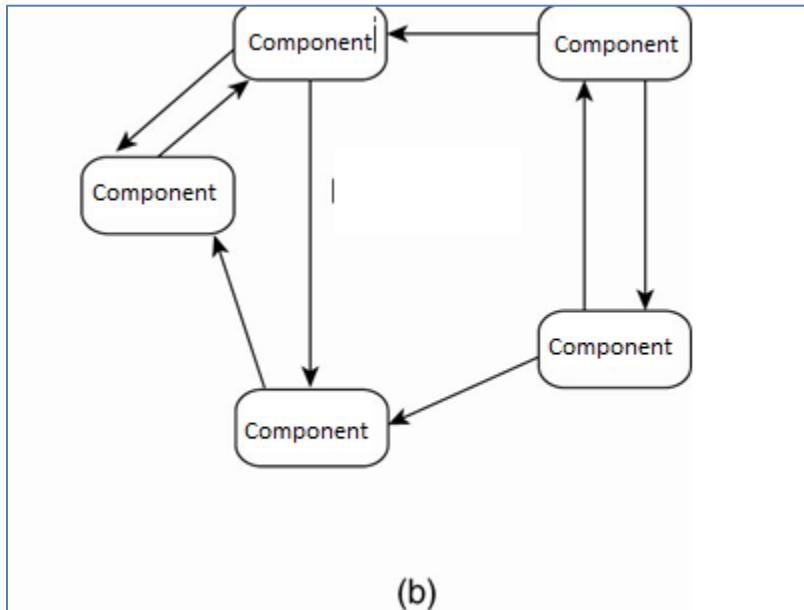
Layered Architecture

- Each layer/tier is allocated a specific responsibility of the system
- Each layer can only interact with the neighboring layers (important for integrity and security)
- Each layer may contain layers of its own (e.g. Presentation layer could contain two layers: client layer and client presentation layer)

Layered Architecture

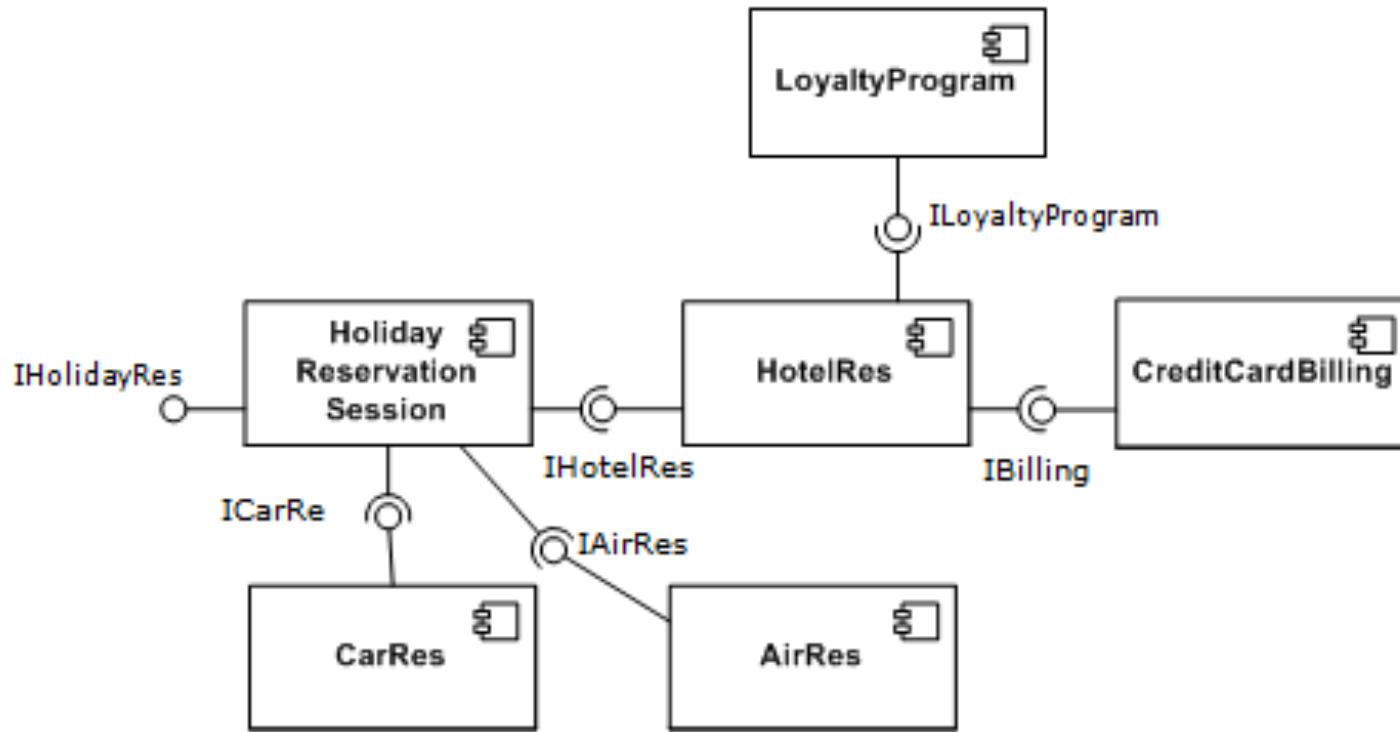


Component based architecture



- Consists of components and connectors
- Component based is less structured

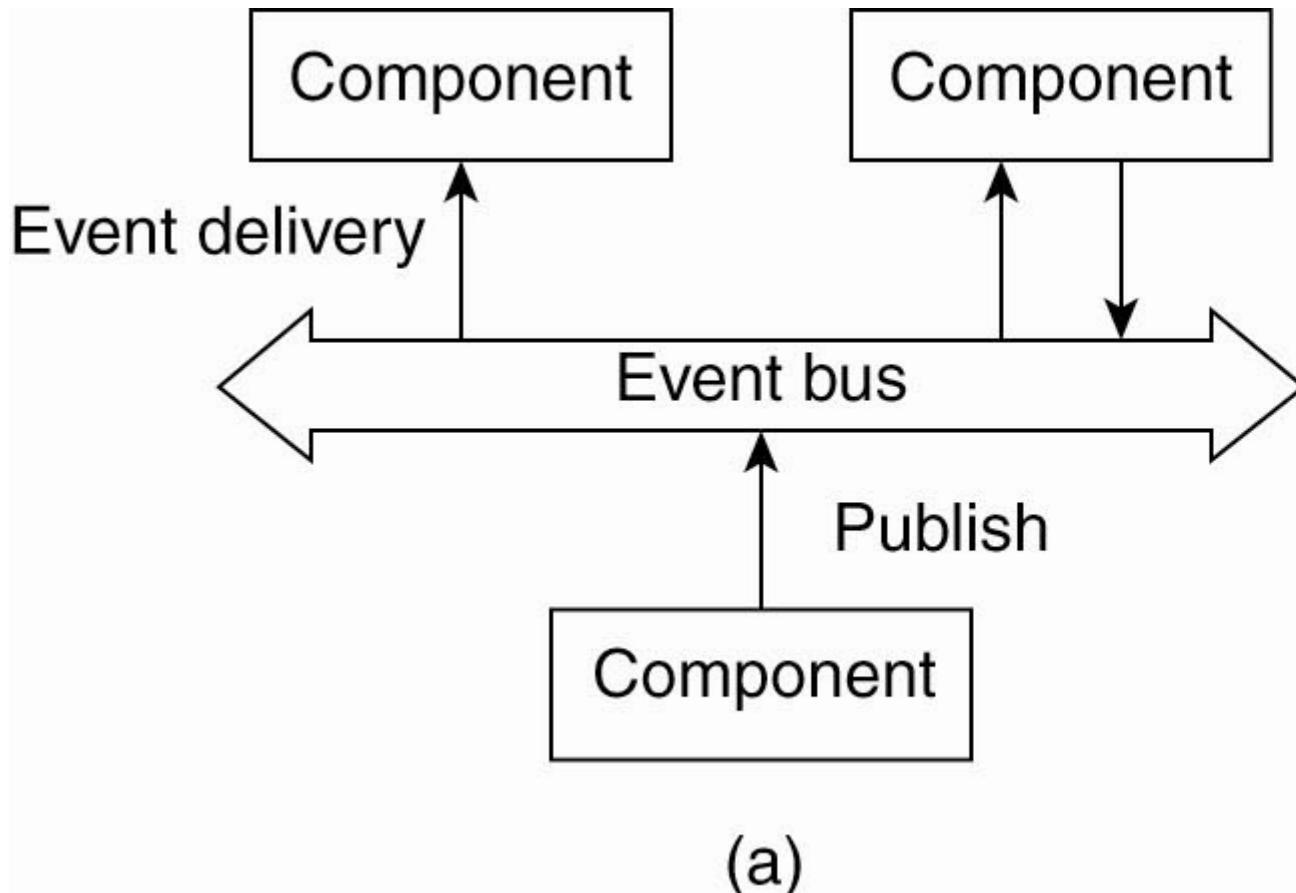
Component based architecture



Data-Centered Architectures

- Main purpose: data access and update
- Processes interact by reading and modifying data in some shared repository (active or passive)
 - Traditional data base (passive): responds to requests
 - Blackboard system (active): clients solve problems collaboratively; system updates clients when information changes.

Event based Architectures



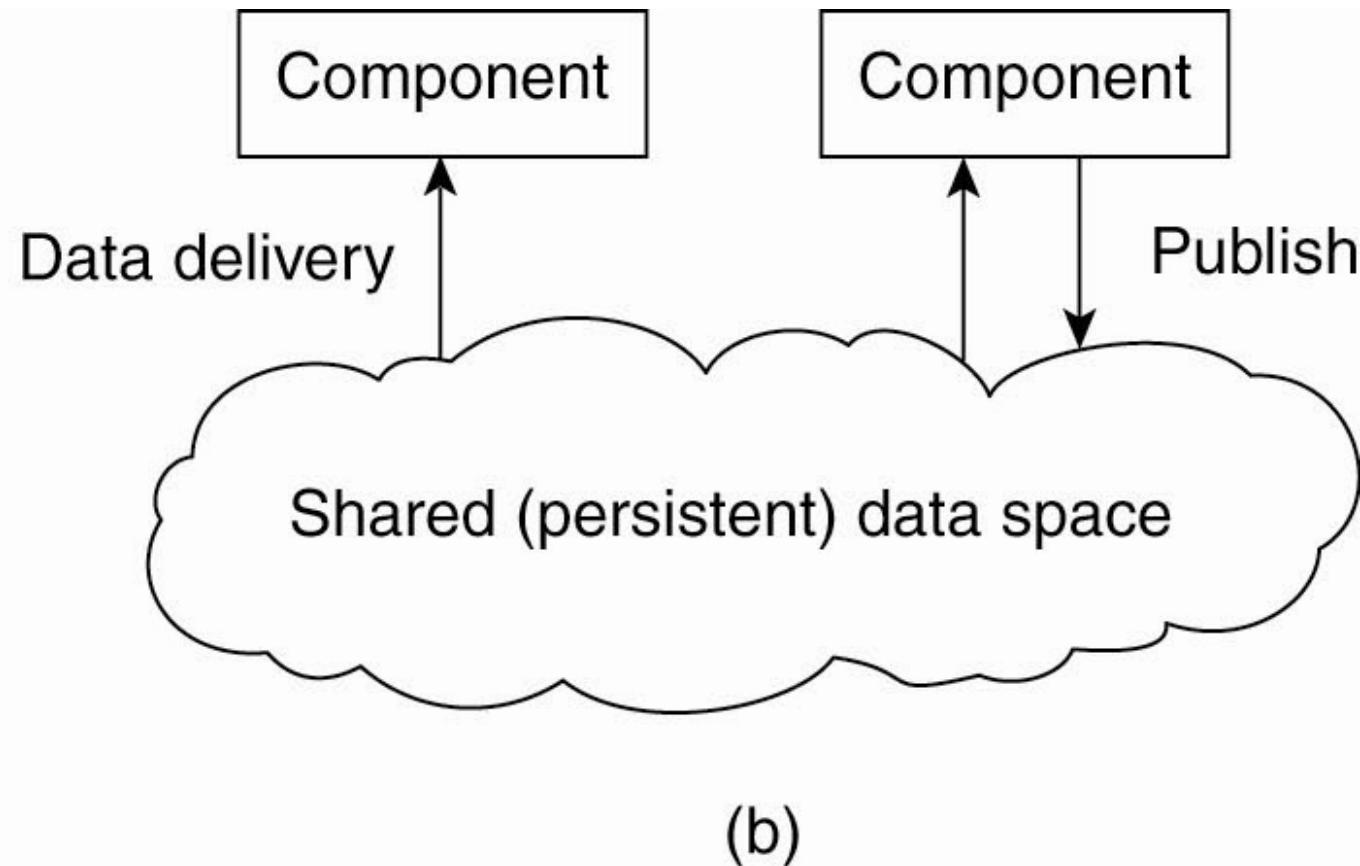
Event based Architectures

- Communication via event propagation, in dist. systems seen often in Publish/ Subscribe; e.g., register interest in market info; get email updates
- Decouples sender & receiver; asynchronous communication
- Event-based architecture supports several communication styles:
 - Publish-subscribe
 - Broadcast
 - Point-to-point

Shared Data-Space Architecture

- Multiple Architectures can be combined in the same system architecture
 - E.g. A component in the object based architecture may have a layered architecture
- Shared Data-Space Architecture combines the Data-centric architecture and Event based architecture

Shared Data-Space Architecture



- E.g., shared distributed file systems or Web-based distributed systems
- Processes communicate asynchronously

Which Software Architectural style?

- An online forum to share travel information among users
- A remote monitoring system that monitors the health of an elderly person.
- A music file sharing system among a group of users.
- A mobile taxi app

Distribution Transparency

- An important characteristic of software architectures in distributed systems is that they are designed to support distribution transparency.
- Transparency involves trade-offs
- Different distributed applications require different solutions/architectures
 - There is no “silver bullet” – no one-size-fits-all system.
(Compare NOW, Seti@home, Condor)

System Architectures

System Architectures for Distributed Systems

- **Centralized:** traditional client-server structure
 - Vertical (or hierarchical) organization of communication and control paths (as in layered software architectures)
 - Logical separation of functions into client (requesting process) and server (responder)
- **Decentralized:** peer-to-peer
 - Horizontal rather than hierarchical comm. and control
 - Communication paths are less structured; symmetric functionality
- **Hybrid:** combine elements of C/S and P2P
 - Edge-server systems
 - Collaborative distributed systems.
- Classification of a system as centralized or decentralized refers to communication and control organization, primarily.

Traditional Client-Server

- Processes are divided into two groups (clients and servers).
- Synchronous communication: request-reply protocol
- In LANs, often implemented with a connectionless protocol (unreliable)
- In WANs, communication is typically connection-oriented TCP/IP (reliable)
 - High likelihood of communication failures

C/S Architectures

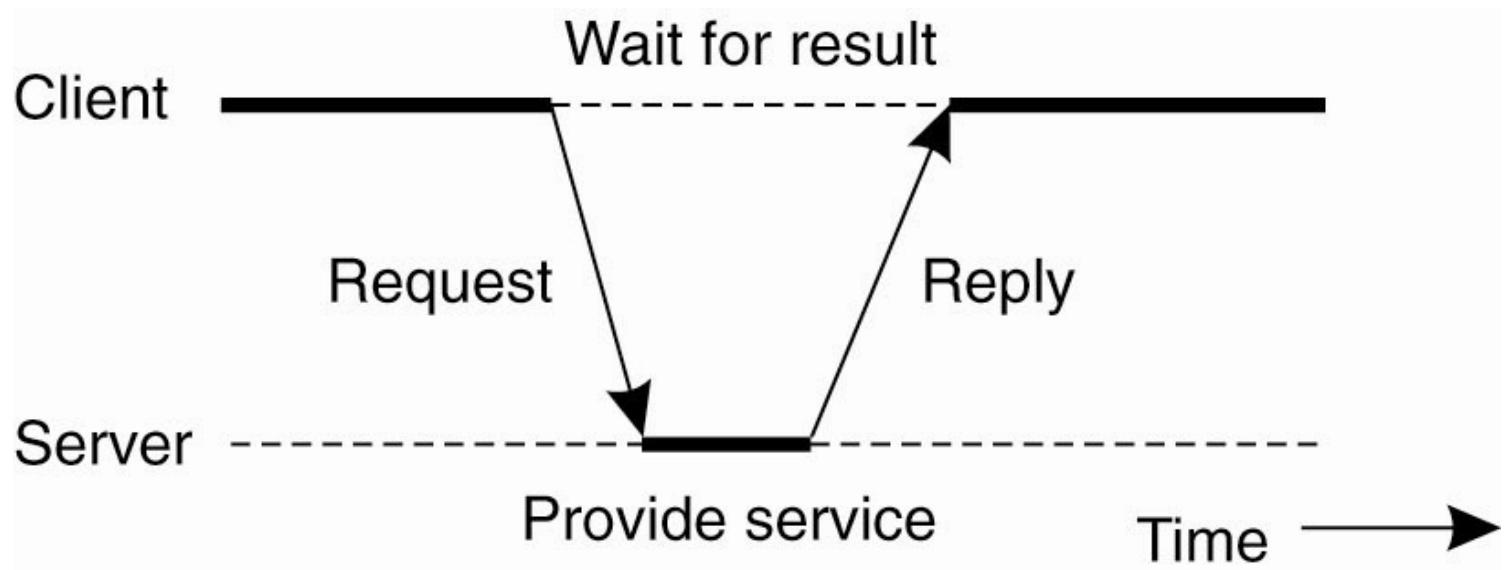


Figure 2-3. General interaction between a client and a server.

Two-tiered C/S Architectures

- Server provides processing and data management; client provides simple graphical display (**thin-client**)
 - Perceived performance loss at client
 - Easier to manage, more reliable, client machines don't need to be so large and powerful
- At the other extreme, all application processing and some data resides at the client (**fat-client** approach)
 - Pro: reduces workload at server; more scalable
 - Con: harder to manage by system admin, less secure

Three-tiered Architectures

- In some applications servers may also need to be clients, leading to a three-level architecture
 - Distributed transaction processing
 - Web servers that interact with database servers
- Distribute functionality across three levels of machines instead of two.

Multitiered Architectures (3 Tier Architecture)

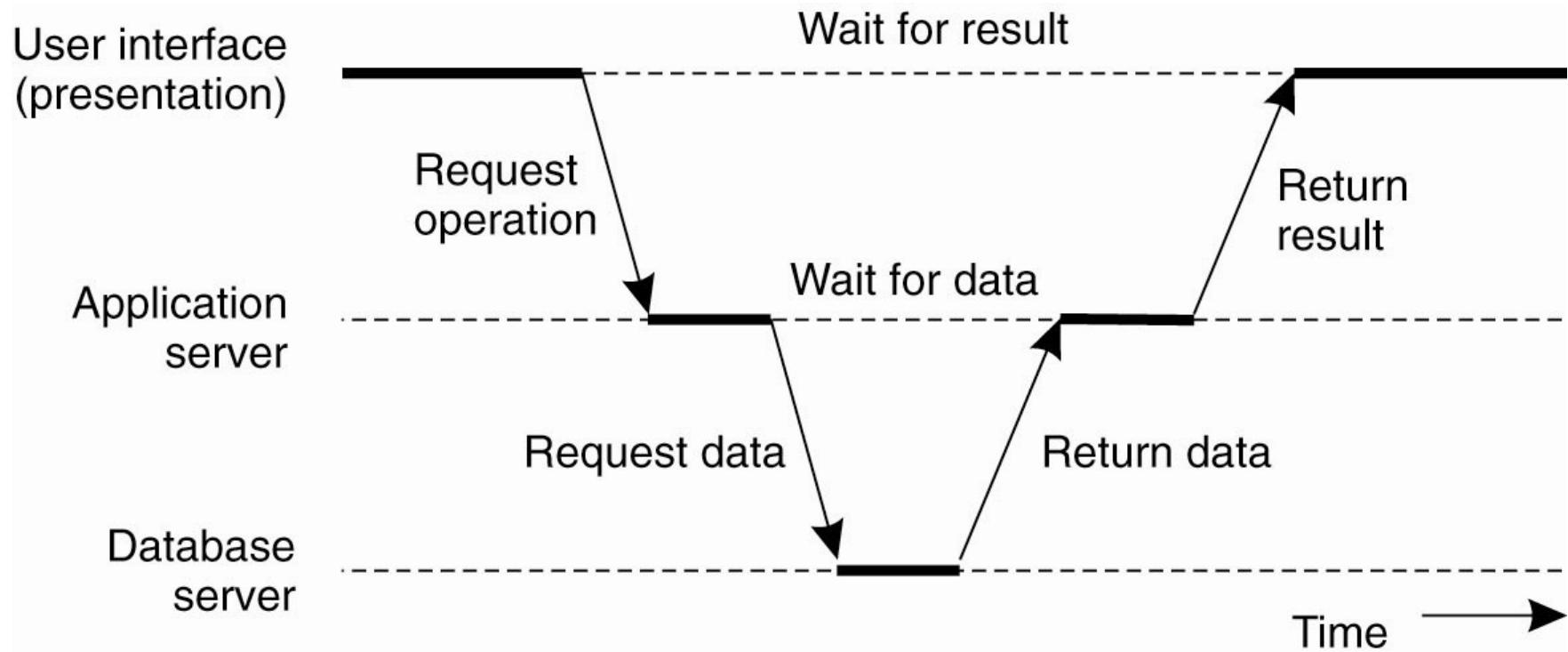


Figure 2-6. An example of a server acting as client.

Multitiered Architectures

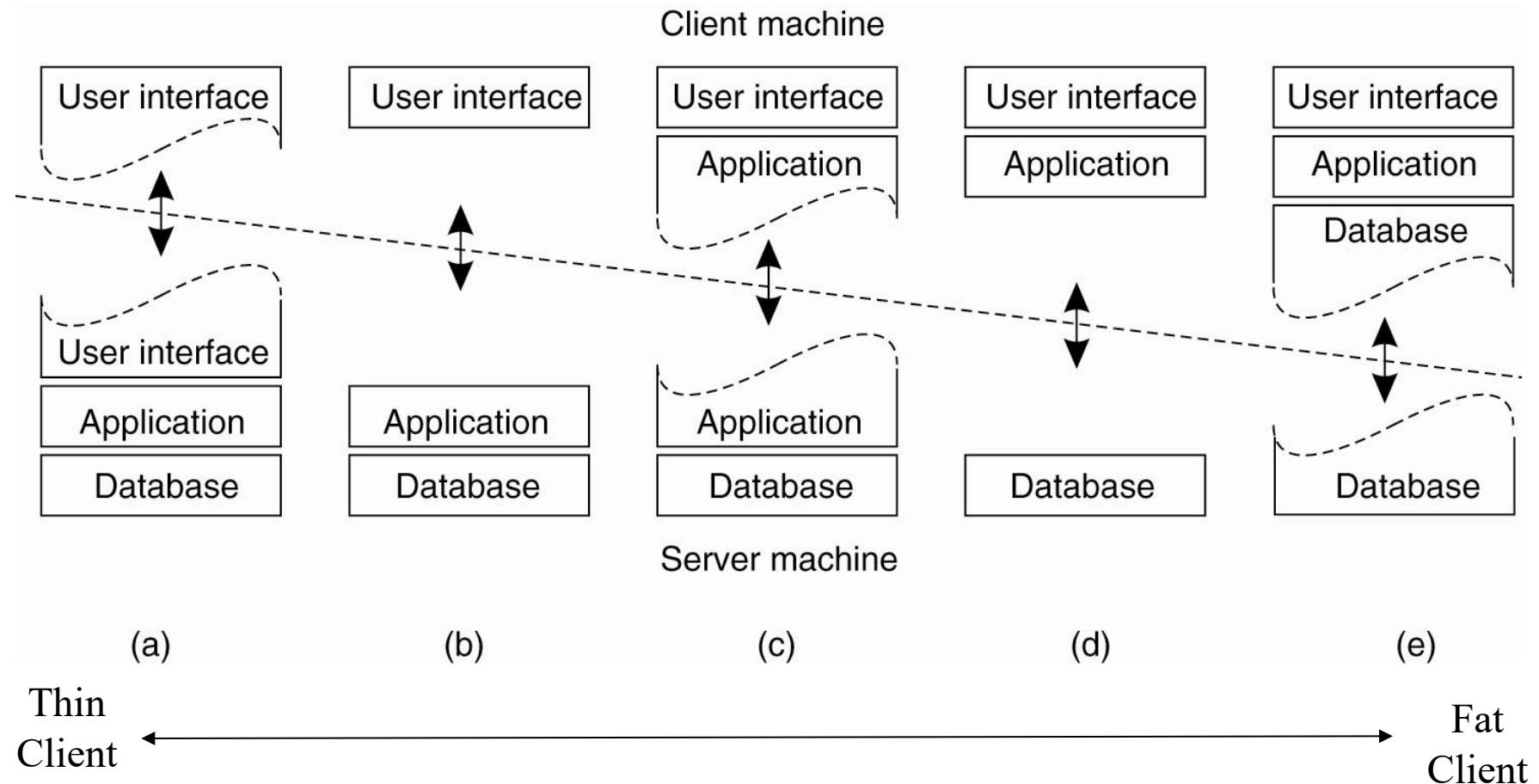


Figure 2-5. Alternative client-server organizations (a)–(e).

Layered (software) Architecture for Client-Server Systems

- **User-interface level:** GUI's (usually) for interacting with end users
- **Processing level:** data processing applications
 - the core functionality
- **Data level:** interacts with data base or file system
 - Data usually is persistent; exists even if no client is accessing it
 - File or database system

Examples

- Web search engine
 - Interface: type in a keyword string
 - Processing level: processes to generate DB queries, rank replies, format response
 - Data level: database of web pages
- Stock broker's decision support system
 - Interface: likely more complex than simple search
 - Processing: programs to analyze data; rely on statistics, AI perhaps, may require large simulations
 - Data level: DB of financial information
- Cloud based “office suites”
 - Interface: access to various documents, data,
 - Processing: word processing, database queries, spreadsheets,...
 - Data : file systems and/or databases

Application Layering

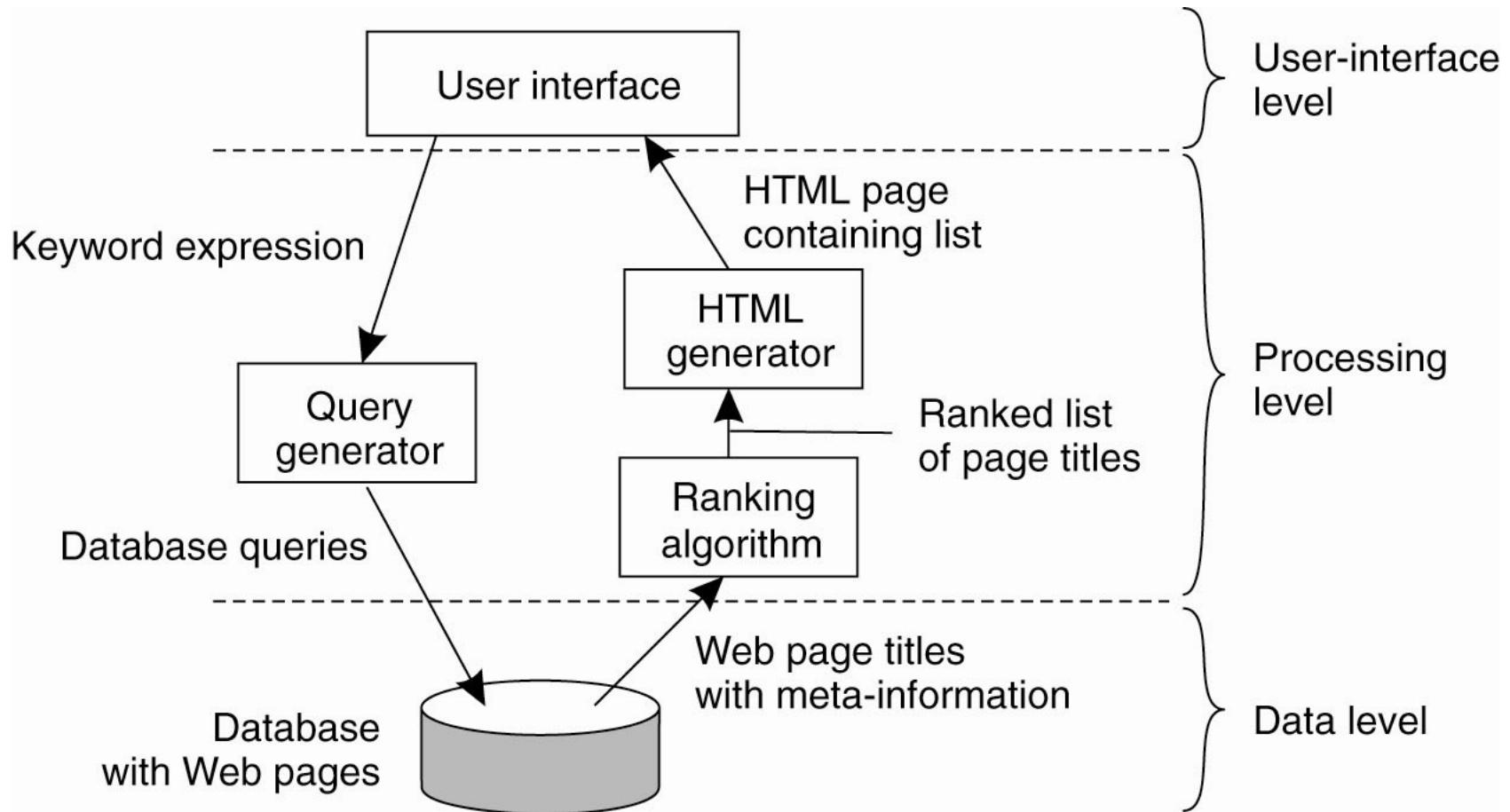
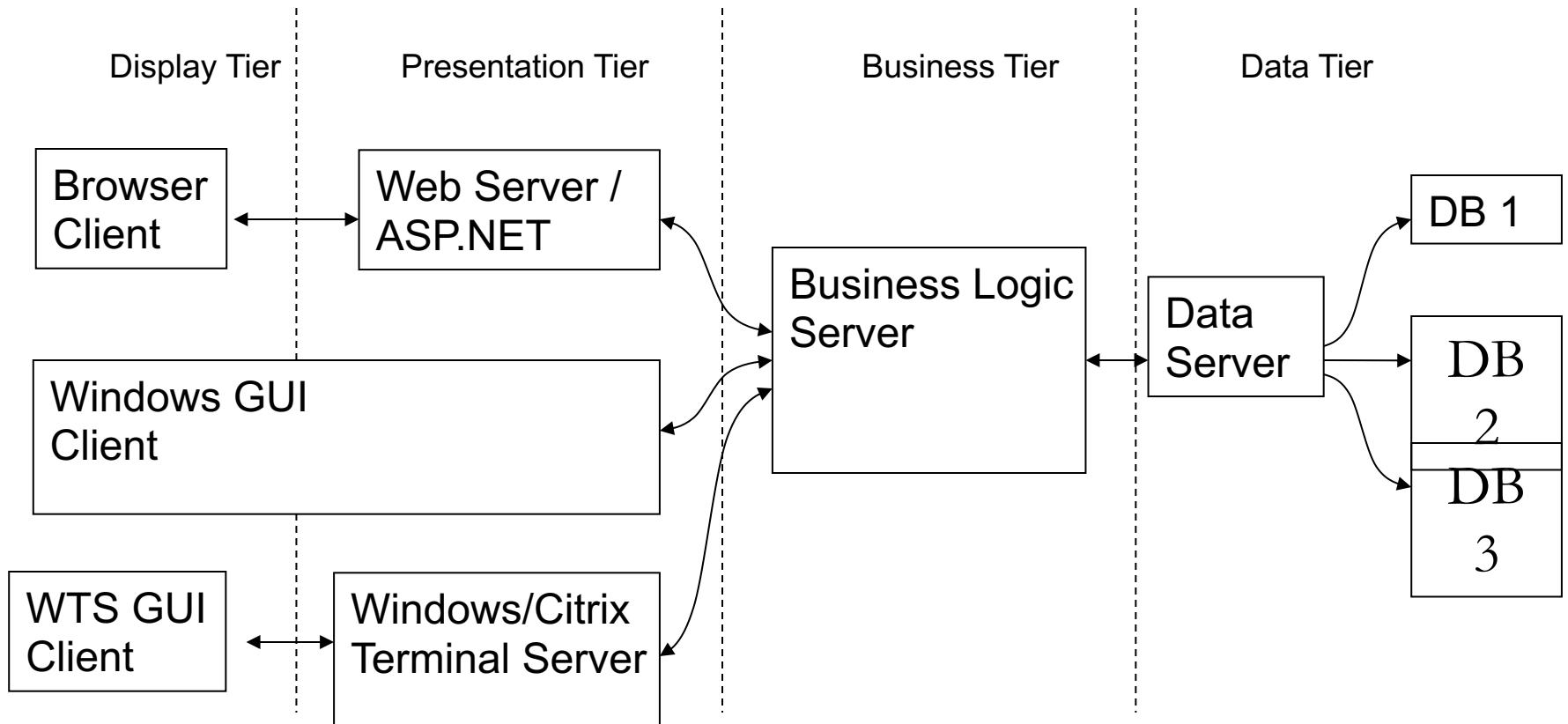


Figure 2-4. The simplified organization of an Internet search engine into three different layers.

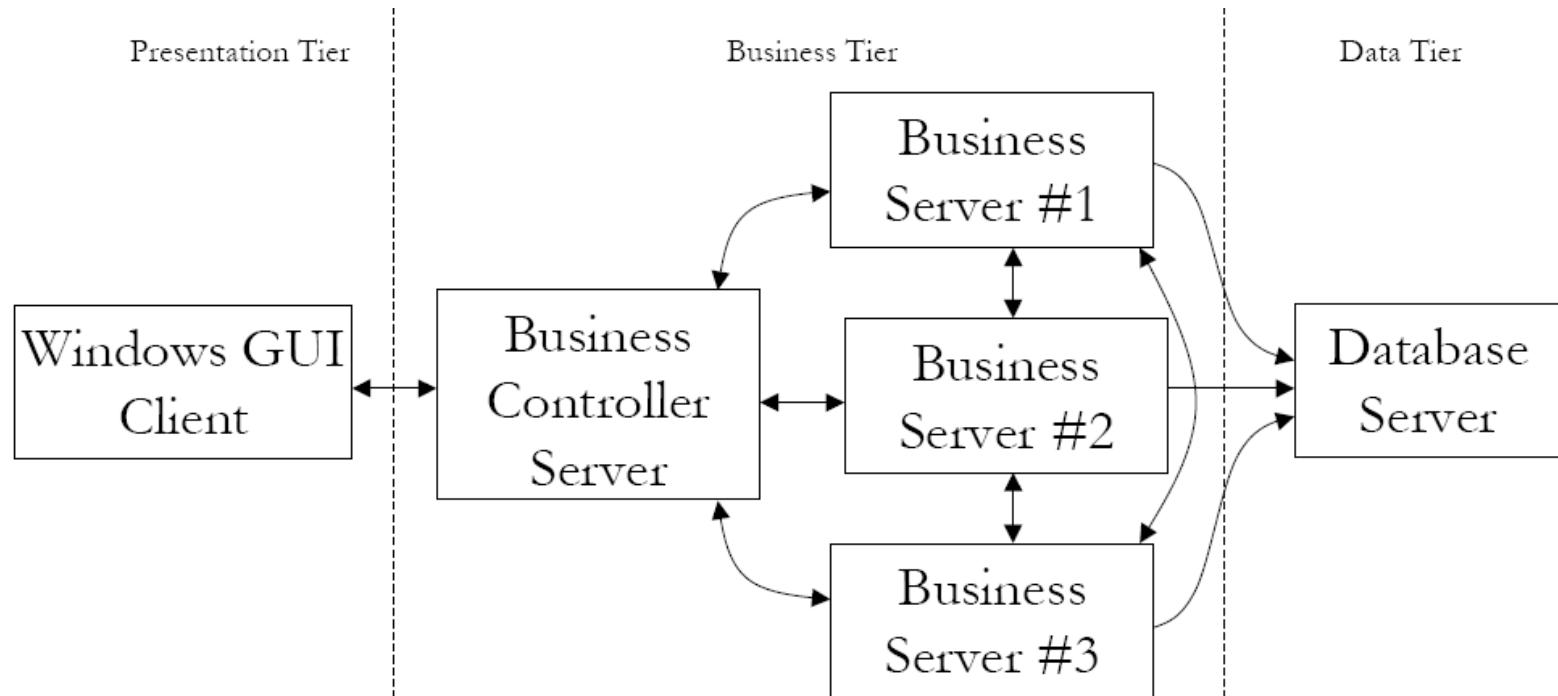
Flexible/Scalable Architecture



Distributing the Business Tier

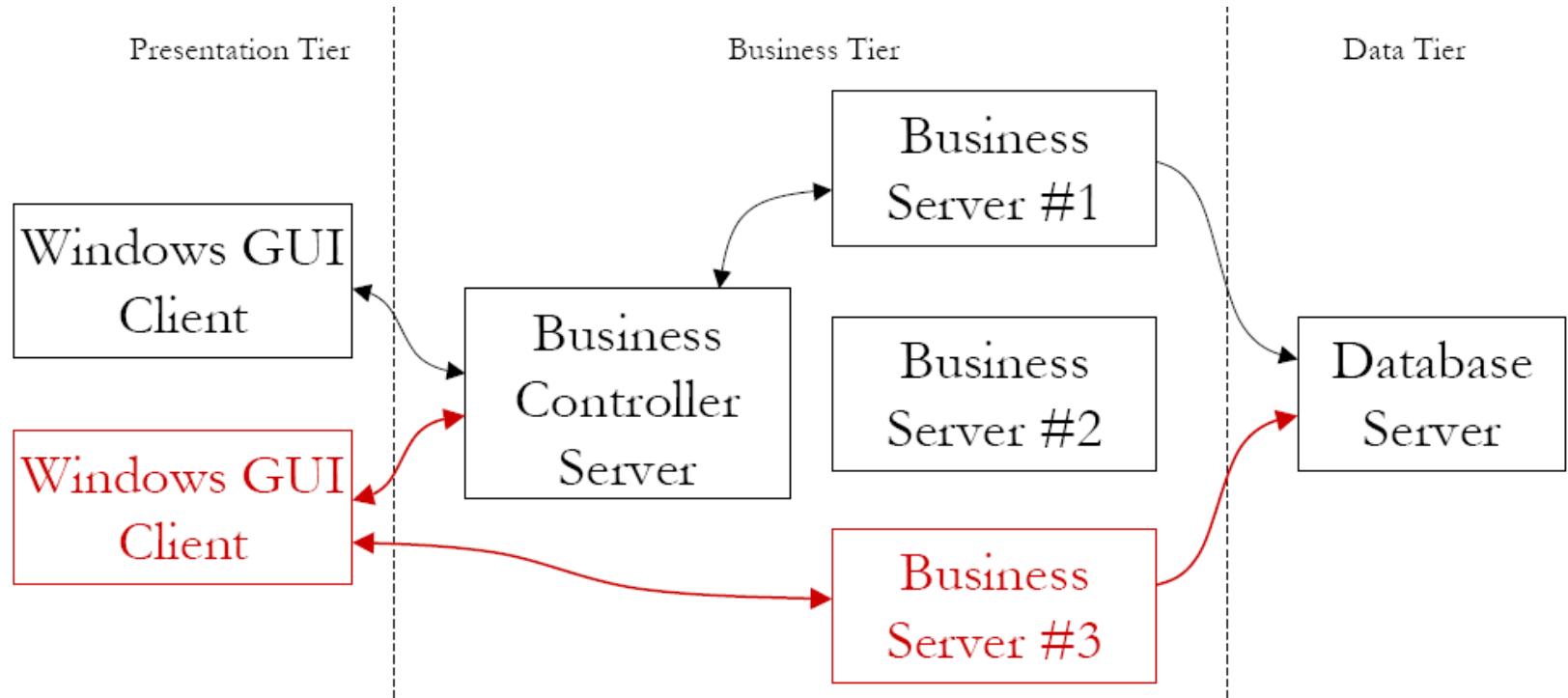
- Business tier could be distributed for various reasons:
 - Parallel computing - split one job among many servers
 - Load balancing - have a controller component redirect presentation clients to a least-utilized business tier server
 - Fault tolerance - robustness to system faults or data faults

Parallel Computing Architecture



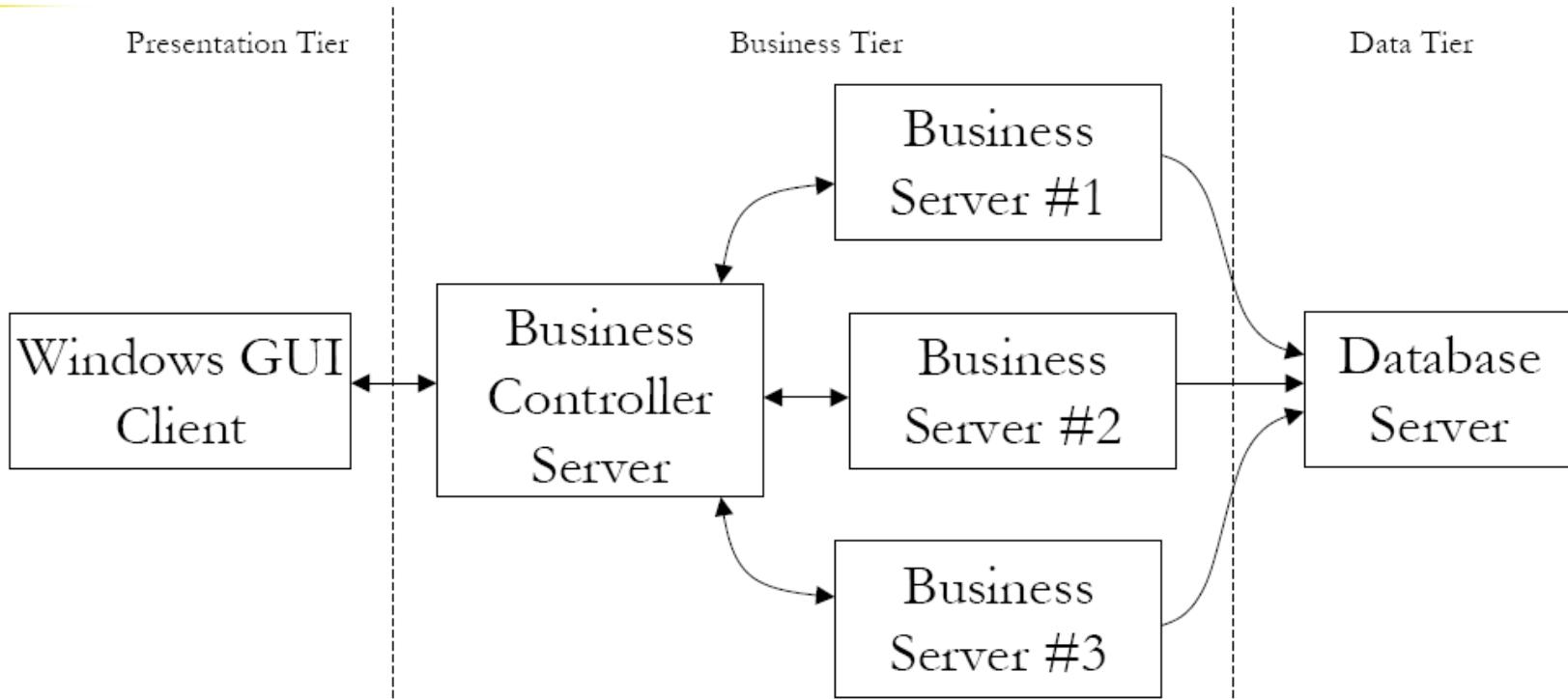
- Controller passes job to server farm, returns response
 - Business servers collaborate via data sharing

Load Balancing Architecture



- Controller passes job to one server, returns response
- **Alternative:** Controller tells client which server to use

Fault Tolerant Architecture



- Data faults: Controller asks all servers to do the same job
 - Then compares results to detect faults
- System faults: Controller uses any server that is up₃₀

Whether to tier or not?

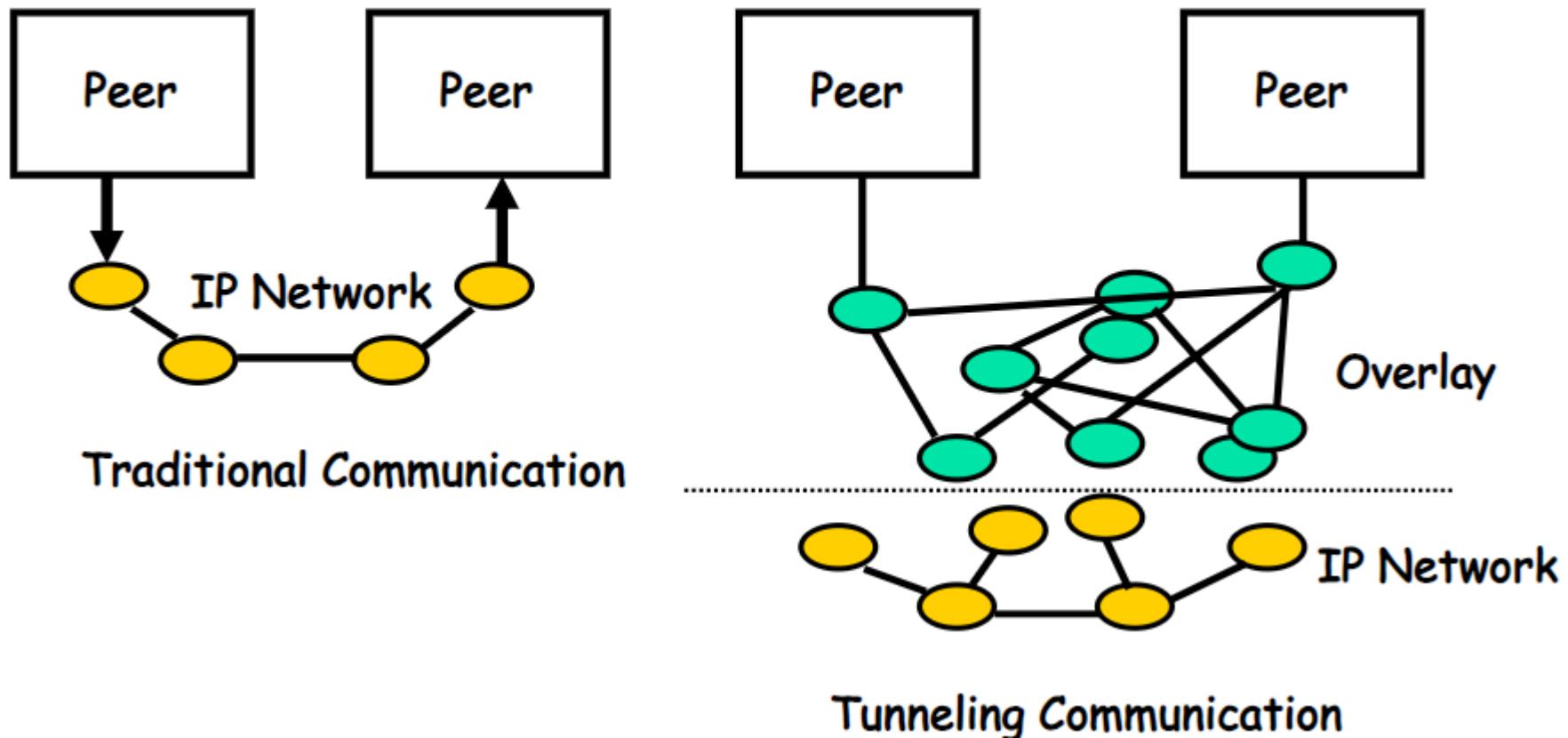
- Advantages
 - Can reuse code (high coherence, low coupling)
 - Scalable
 - Integrity
 - Fault tolerance

- Disadvantages
 - Increases communication overhead
 - Errors/Losses in message transmission
 - Can pose security risks (e.g. packet sniffing)
 - Increased Complexity

Peer-to-Peer

- Nodes act as both client and server; interaction is symmetric (e.g. Pastry, Chord)
- Each node acts as a server for part of the total system data
- **Overlay networks** connect nodes in the P2P system
 - Nodes in the overlay use their own addressing system for storing and retrieving data in the system
 - Nodes can route requests to locations that may not be known by the requester.

P2P Overlay networks



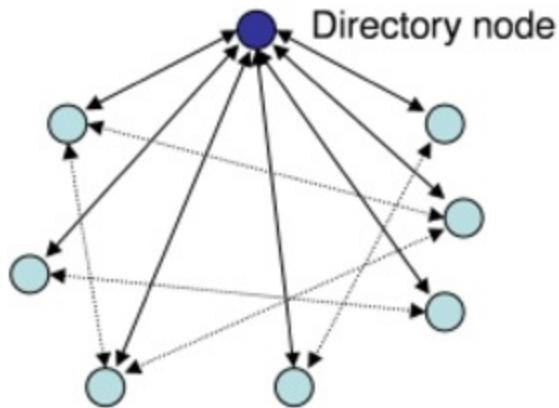
P2P v Client/Server

- P2P computing allows end users to communicate without a dedicated server.
- P2P overcomes the issue of Single point of failure in C/S
- Communication is still usually synchronous (blocking)
- There is less likelihood of performance bottlenecks since communication is more distributed.
 - Data distribution leads to workload distribution.
- Resource discovery is more difficult than in centralized client-server computing & look-up/retrieval is slower
- P2P can have freeloading/free-riding issue (particularly in resource sharing)
- P2P can be more fault tolerant, more resistant to denial of service (DoS) attacks because network content is distributed.
 - Individual hosts may be unreliable, but overall, the system should maintain a consistent level of service

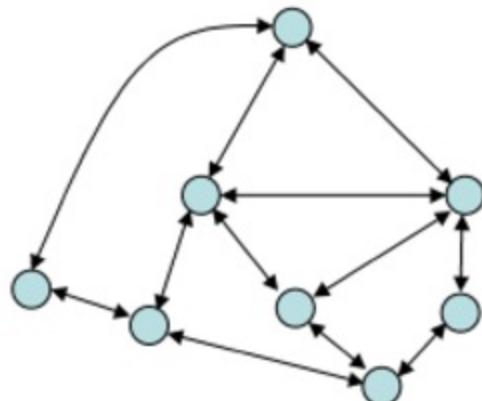
P2P architectures

- Structured – E.g. Chord, Pastry, uses Distributed Hash Tables (DHT), has Circular structures
- Unstructured – E.g. BitTorrent, Napster, has unstructured complex structures, both pure P2P and Hybrid are unstructured

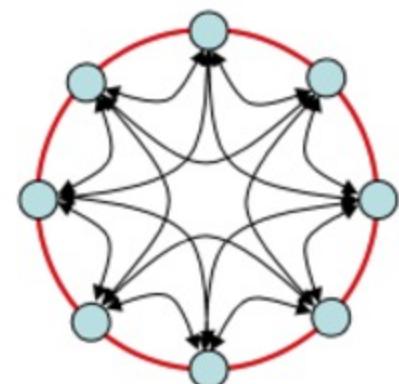
Structured vs Unstructured P2P



Hybrid (Napster)



Unstructured overlay



Structured overlay

Hybrid Architectures

- Combine client-server and P2P architectures
 - Edge-server systems; e.g. ISPs, which act as servers to their clients, but cooperate with other edge servers to host shared content
 - Collaborative distributed systems; e.g., BitTorrent, which supports parallel downloading and uploading of chunks of a file. First, interact with C/S system, then operate in decentralized manner.

Superpeers

- Maintain indexes to some or all nodes in the system
- Supports resource discovery
- Act as servers to regular peer nodes, peers to other super-peers
- Improve scalability by controlling floods
- Can also monitor state of network
- Example: Napster
- Edge-Server computing
(e.g. ISP)

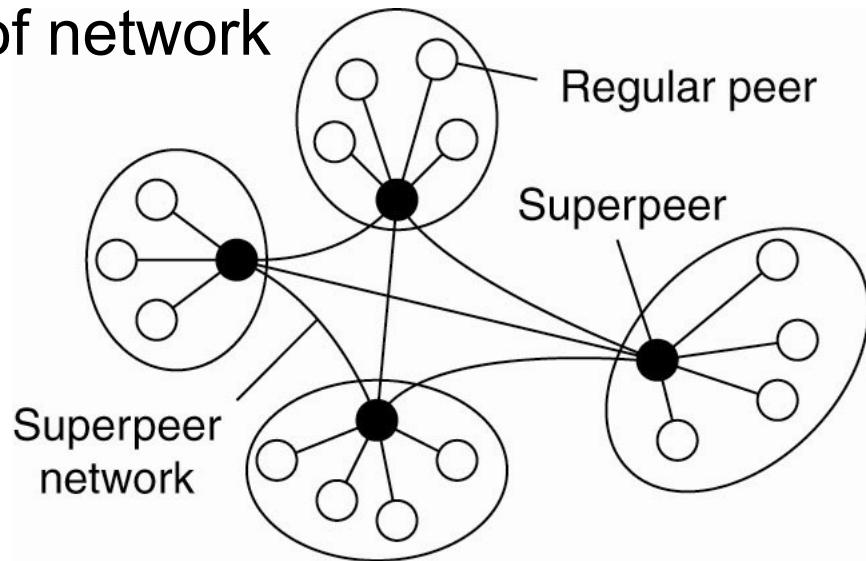


Figure 2-12.

Hybrid (System Architecture) example

- BitTorrent

<https://www.youtube.com/watch?v=6PWUCFmOQwQ>

Distributed Hash Tables

- A fully decentralized routing mechanism
(unlike TCP/IP routing)

Distributed Hash Table (DHT)

Simple database with **(key, value)** pairs:

- key: human name; value: social security #

Key	Value
John Washington	132-54-3570
Diana Louise Jones	761-55-3791
Xiaoming Liu	385-41-0902
Rakesh Gopal	441-89-1956
Linda Cohen	217-66-5609
.....
Lisa Kobayashi	177-23-0199

- key: movie title; value: IP address

Hash Table

- More convenient to store and search on numerical representation of key
- key = hash(original key)

Original Key	Key	Value
John Washington	8962458	132-54-3570
Diana Louise Jones	7800356	761-55-3791
Xiaoming Liu	1567109	385-41-0902
Rakesh Gopal	2360012	441-89-1956
Linda Cohen	5430938	217-66-5609
.....
Lisa Kobayashi	9290124	177-23-0199

Distributed Hash Table (DHT)

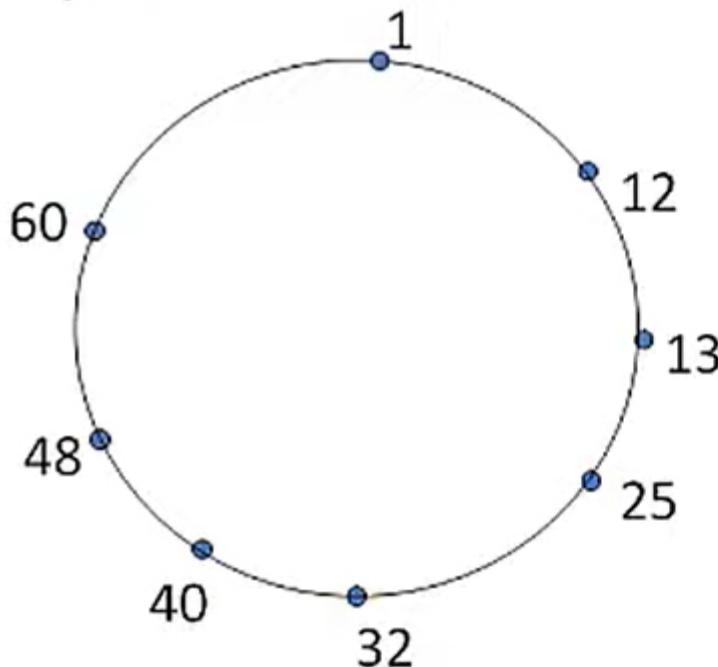
- Distribute (key, value) pairs over millions of peers
 - pairs are evenly distributed over peers
- Any peer can **query** database with a key
 - database returns value for the key
 - To resolve query, small number of messages exchanged among peers
- Each peer only knows about a small number of other peers
- Robust to peers coming and going (churn)

Assign key-value pairs to peers

- rule: assign key-value pair to the peer that has the *closest* ID.
- convention: closest is the *immediate successor* of the key.
- e.g., ID space $\{0,1,2,3,\dots,63\}$
- suppose 8 peers: 1,12,13,25,32,40,48,60
 - If key = 51, then assigned to peer 60
 - If key = 60, then assigned to peer 60
 - If key = 61, then assigned to peer 1

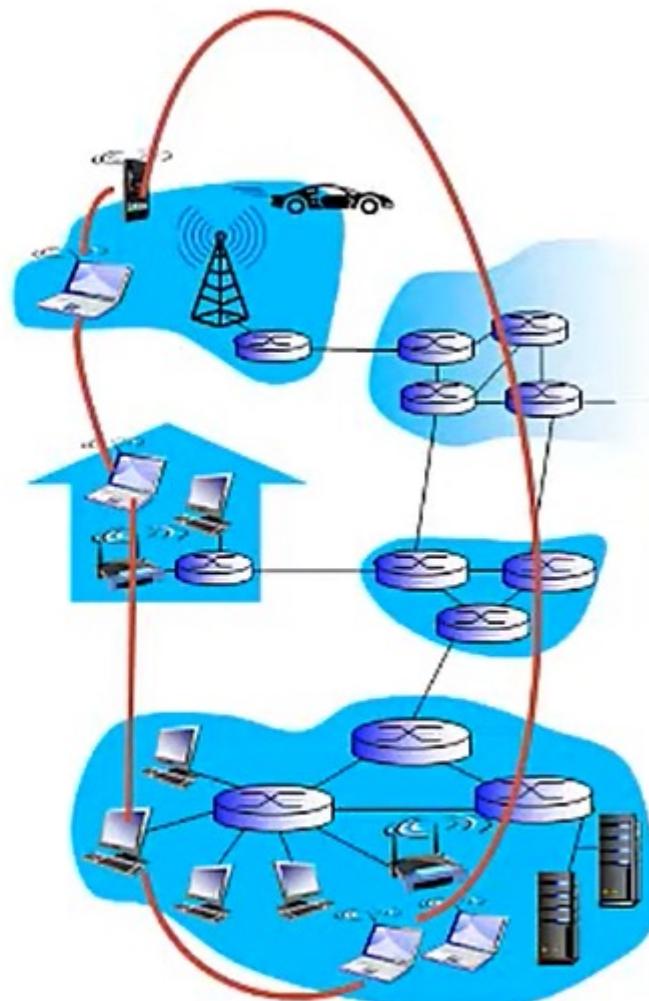
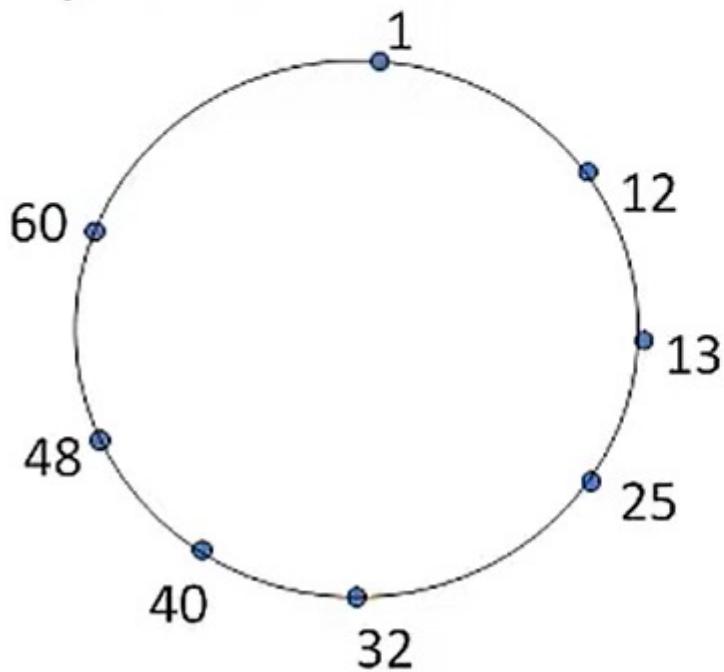
Circular DHT

- each peer *only* aware of immediate successor and predecessor.



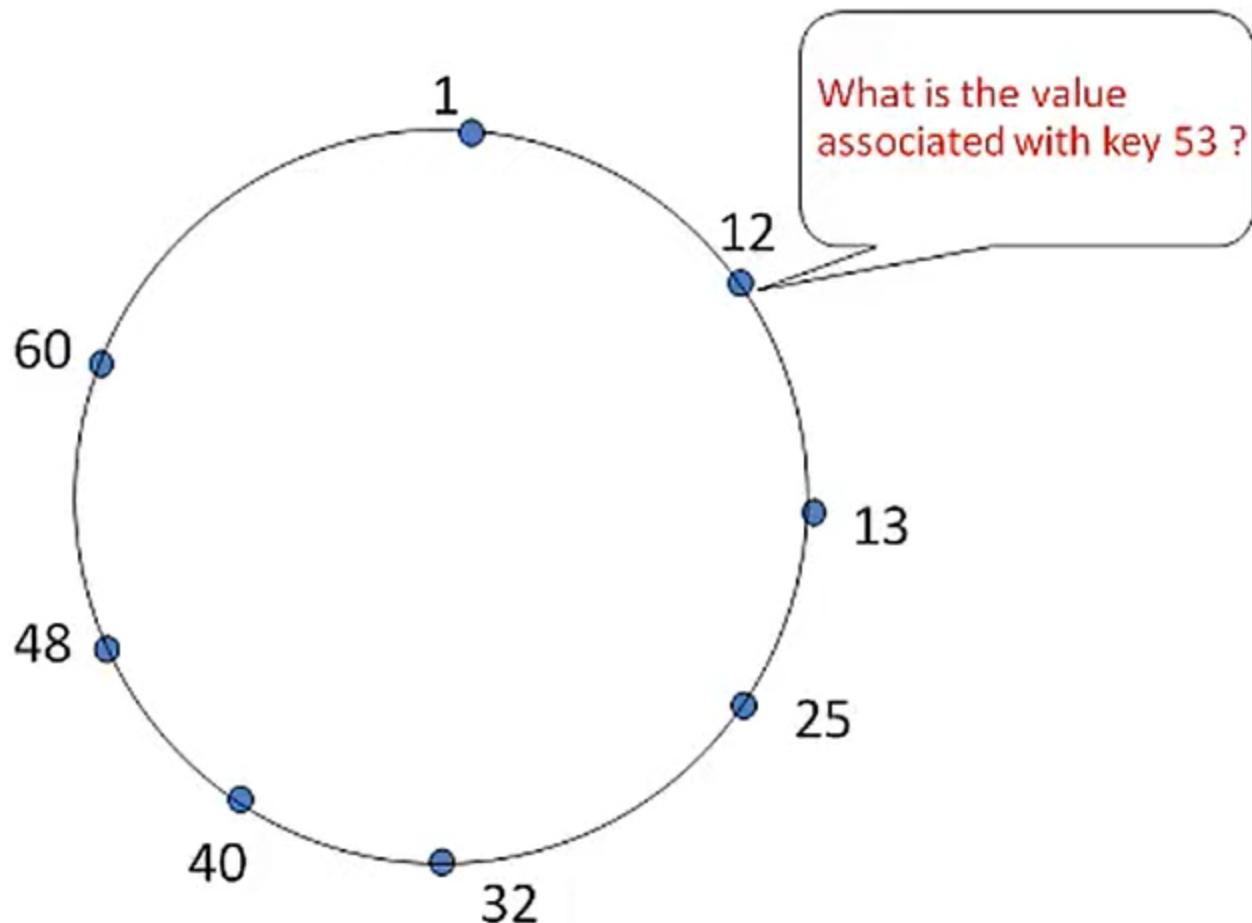
Circular DHT

- each peer *only* aware of immediate successor and predecessor.

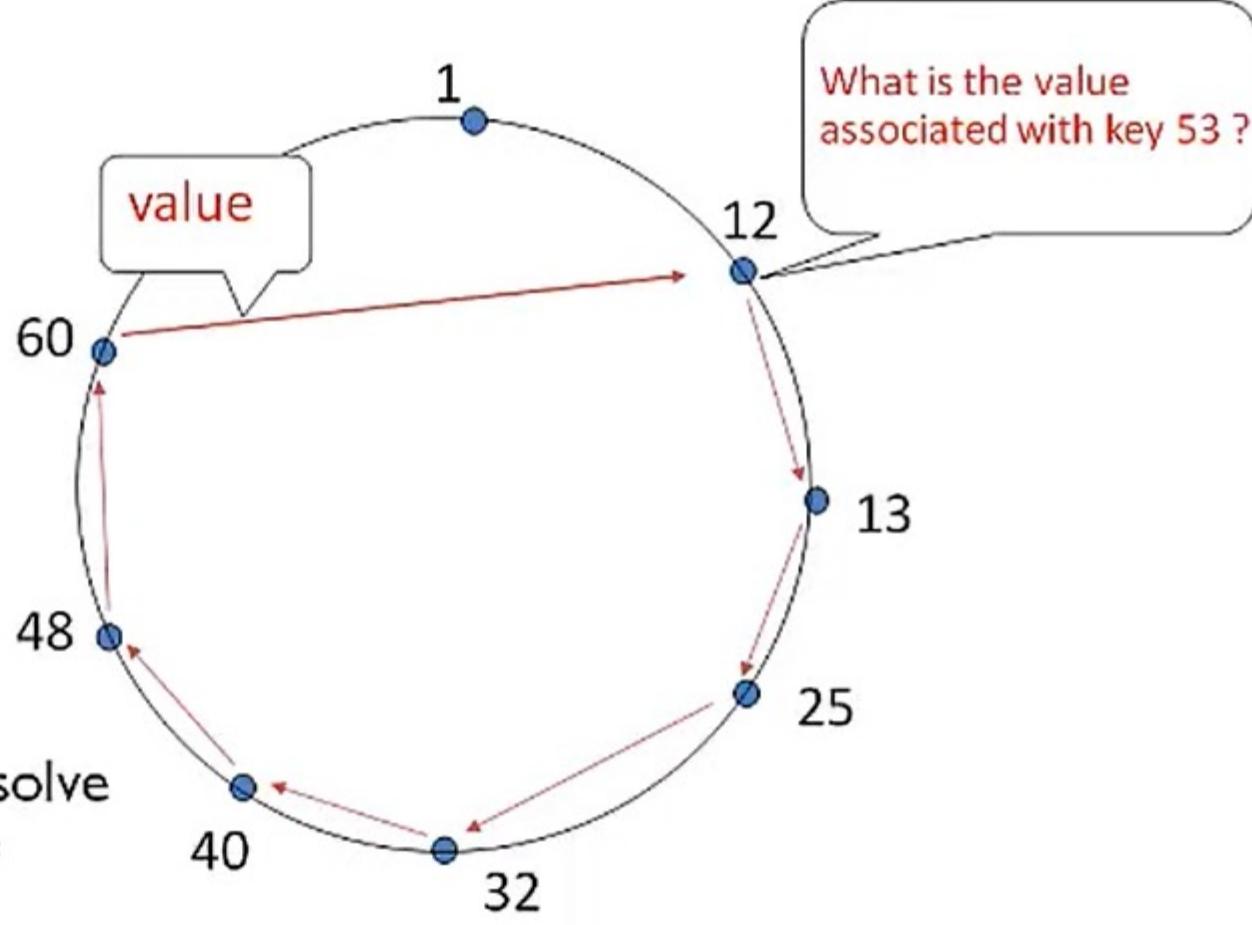


“overlay network”

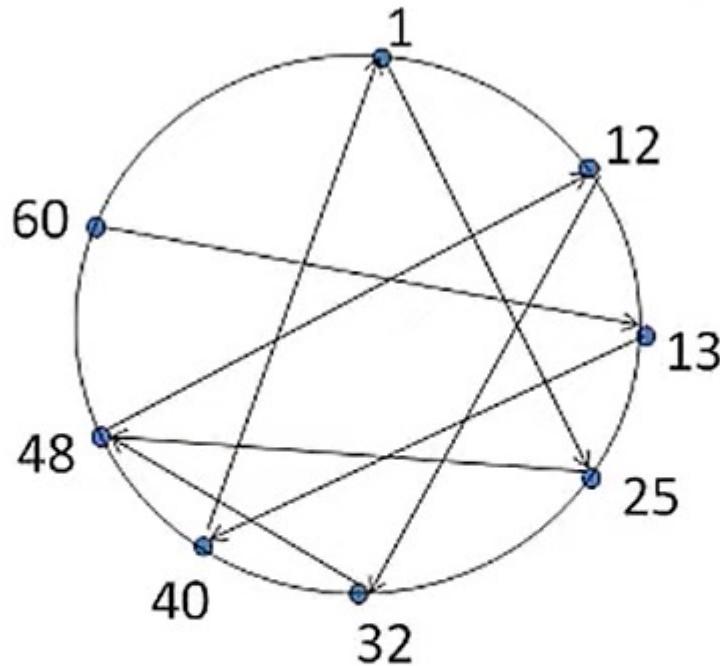
Resolving a query



Resolving a query

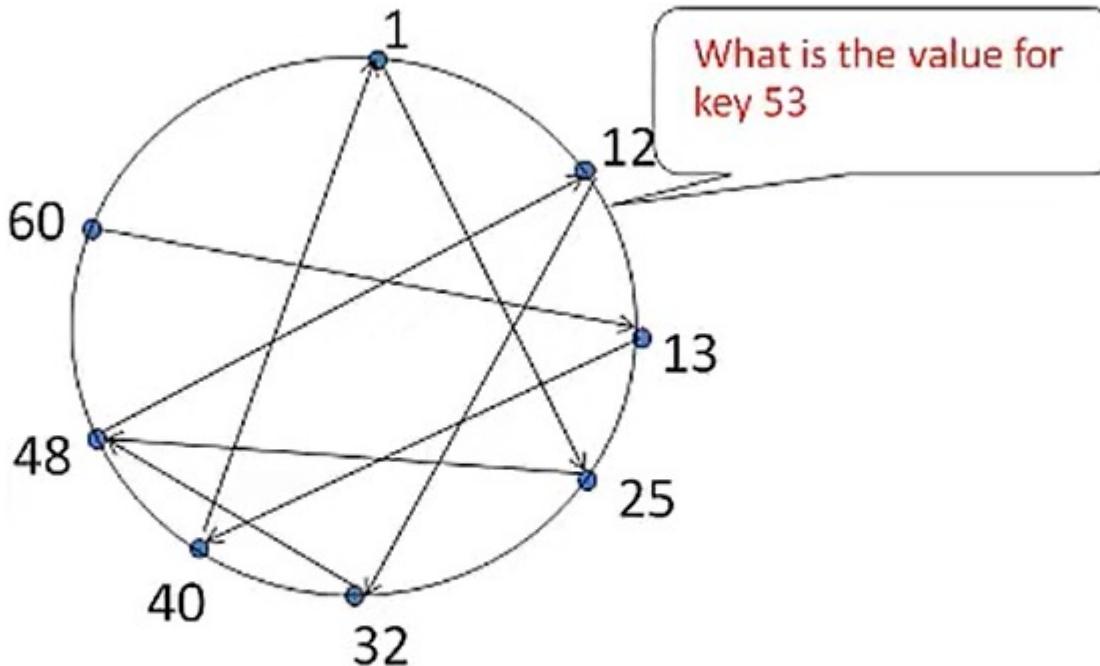


Circular DHT with shortcuts



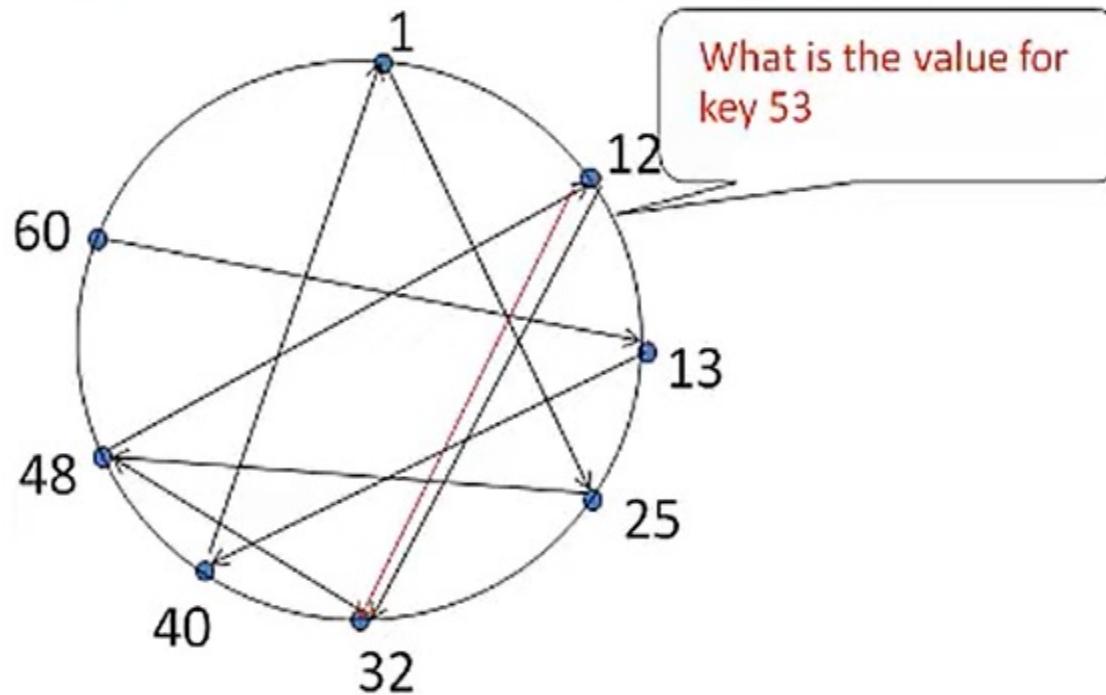
- each peer keeps track of IP addresses of predecessor, successor, short cuts.

Circular DHT with shortcuts



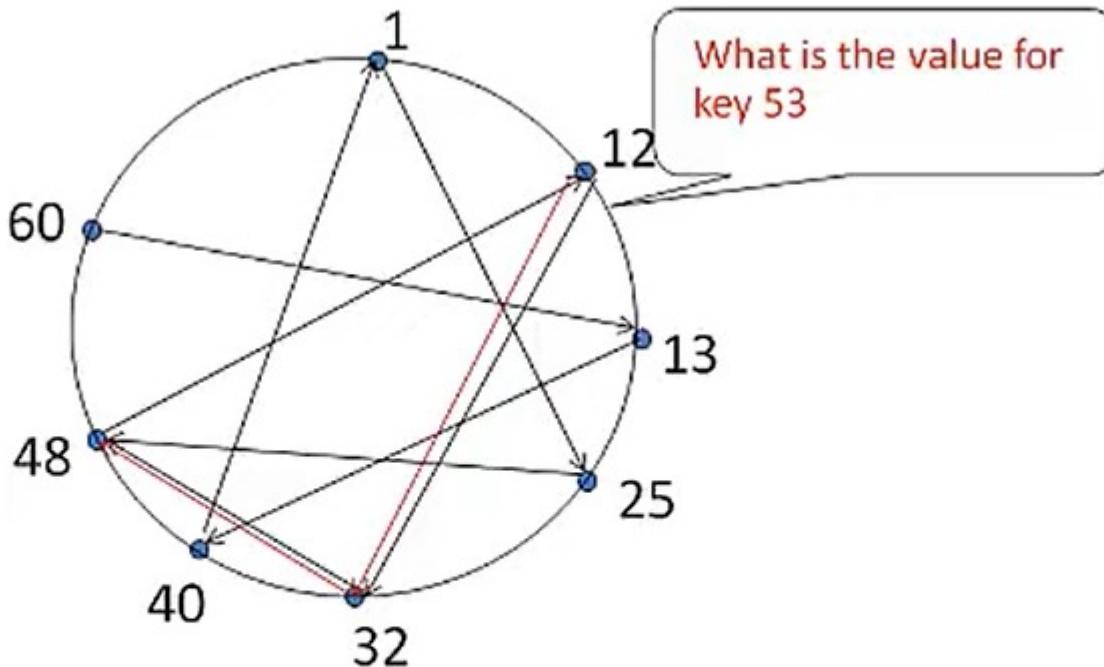
- each peer keeps track of IP addresses of predecessor, successor, short cuts.

Circular DHT with shortcuts



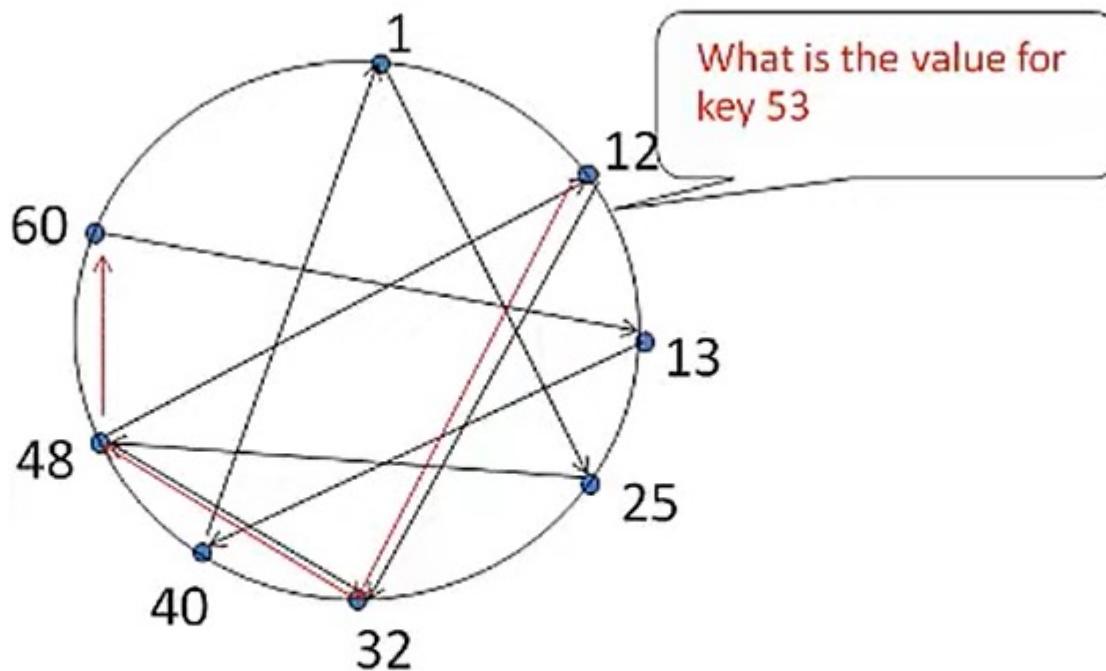
- each peer keeps track of IP addresses of predecessor, successor, short cuts.

Circular DHT with shortcuts



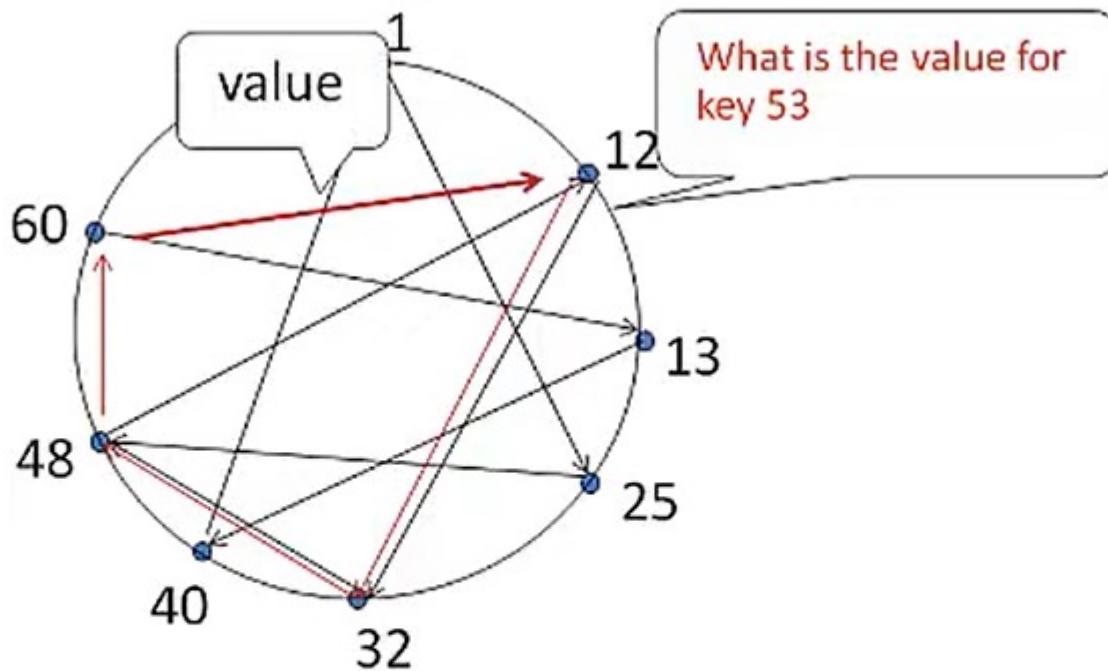
- each peer keeps track of IP addresses of predecessor, successor, short cuts.

Circular DHT with shortcuts



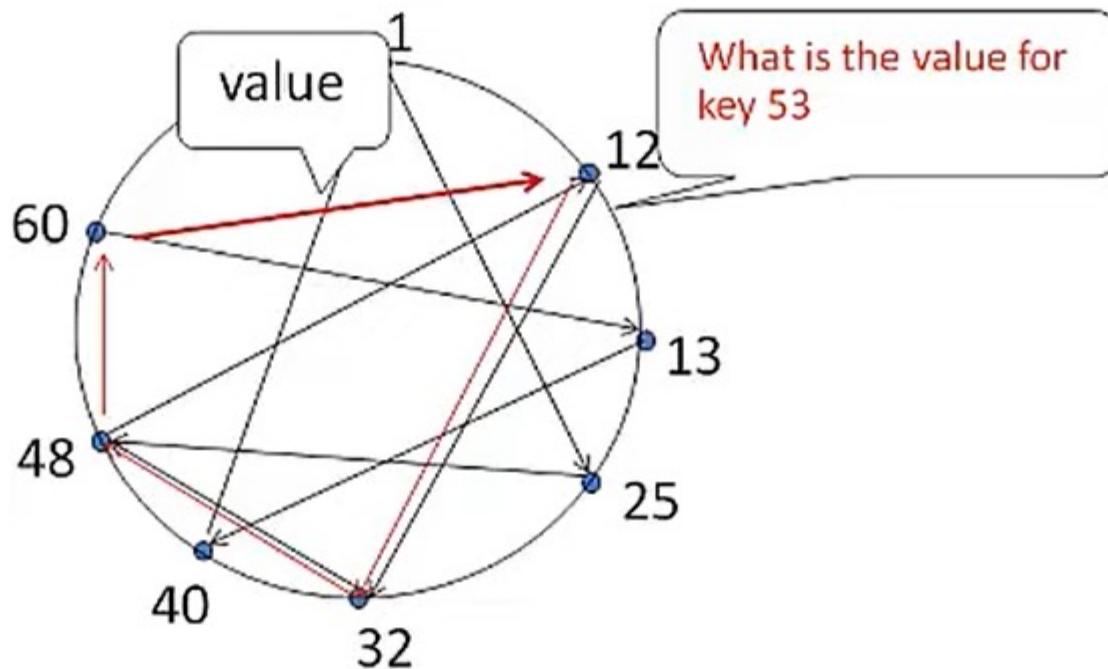
- each peer keeps track of IP addresses of predecessor, successor, short cuts.

Circular DHT with shortcuts



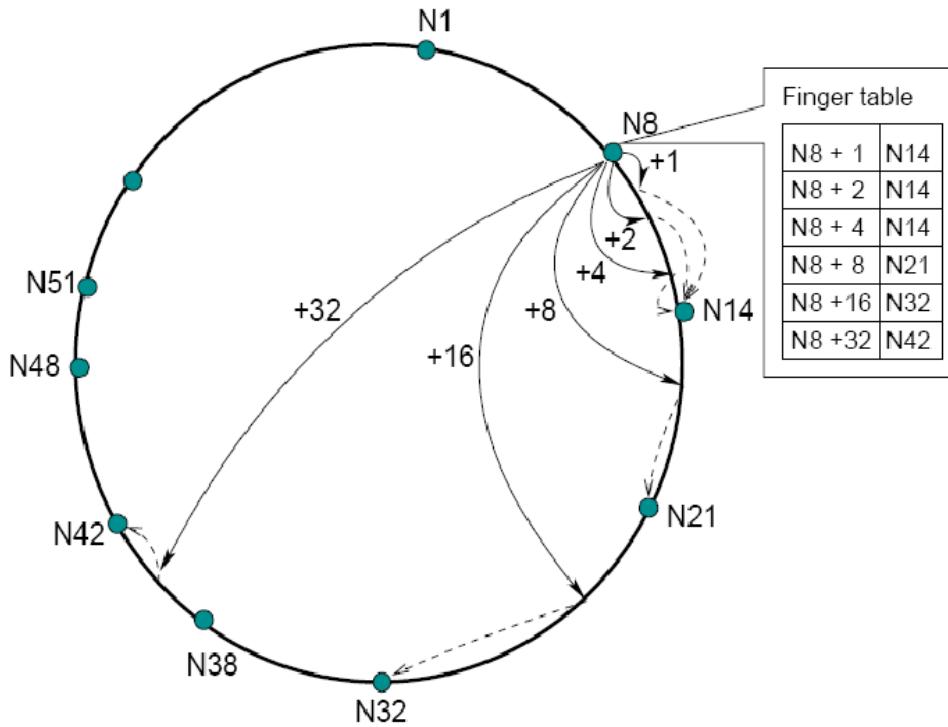
- each peer keeps track of IP addresses of predecessor, successor, short cuts.

Circular DHT with shortcuts



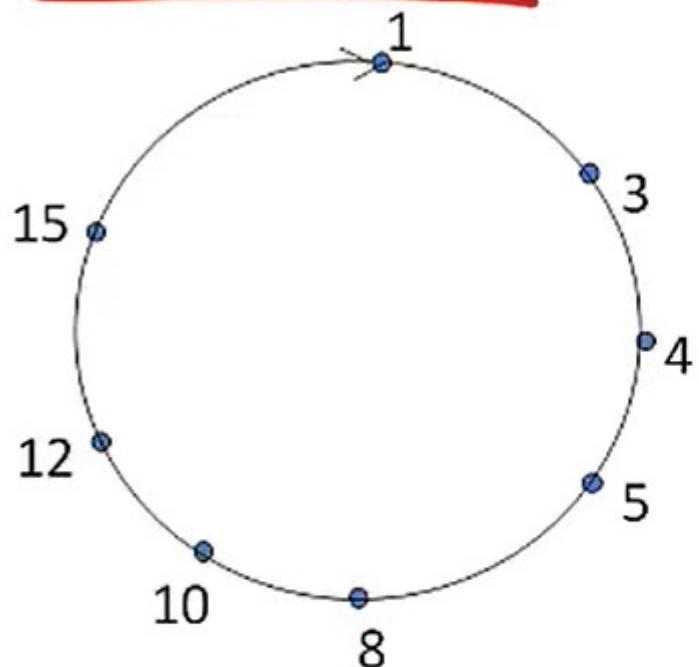
- each peer keeps track of IP addresses of predecessor, successor, short cuts.
- reduced from 6 to 3 messages.

Chord - Finger table



- <https://pdos.csail.mit.edu/papers/ton:chord/paper-ton.pdf>
- Reduces time complexity of lookup to $O(\log n)$

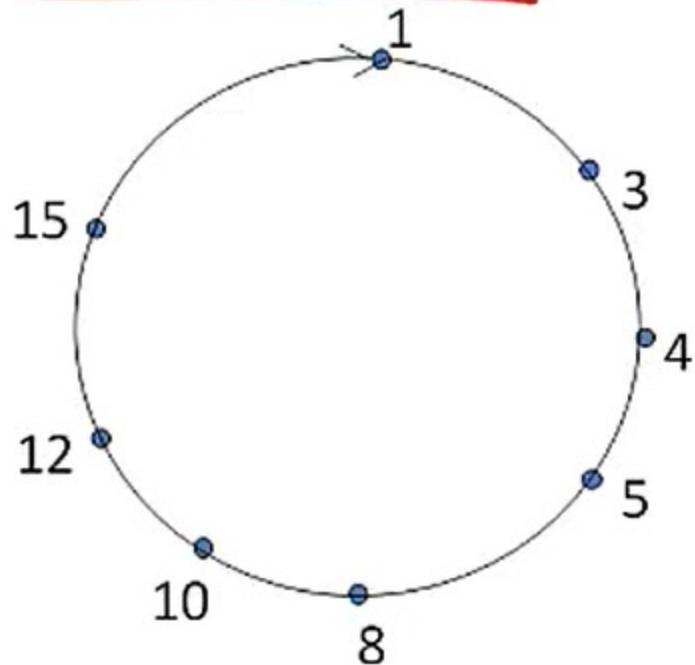
Peer churn



handling peer churn:

- ❖ peers may come and go (churn)

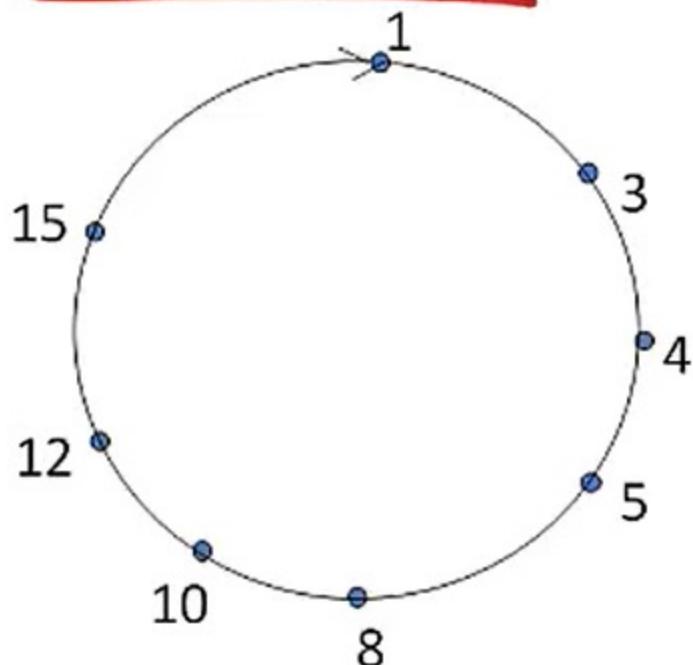
Peer churn



handling peer churn:

- ❖ peers may come and go (churn)
- ❖ each peer knows address of its two successors

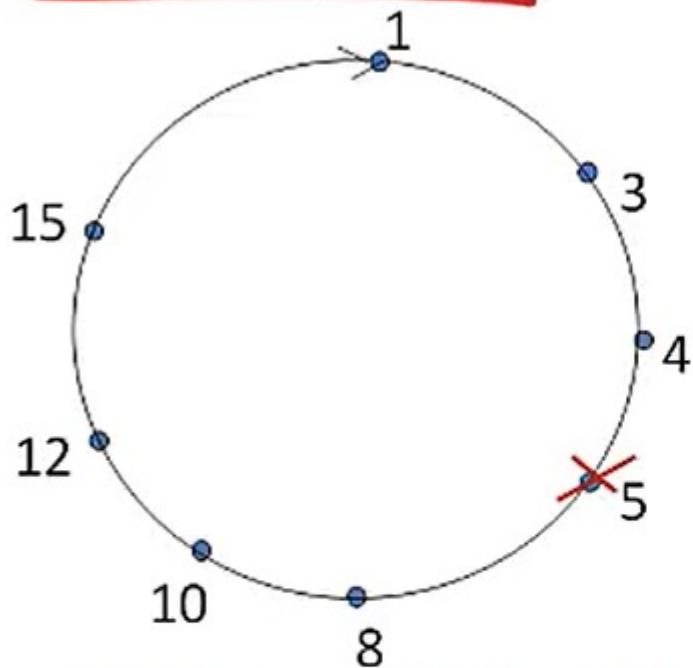
Peer churn



handling peer churn:

- ❖ peers may come and go (churn)
- ❖ each peer knows address of its two successors
- ❖ each peer periodically pings its two successors to check aliveness

Peer churn

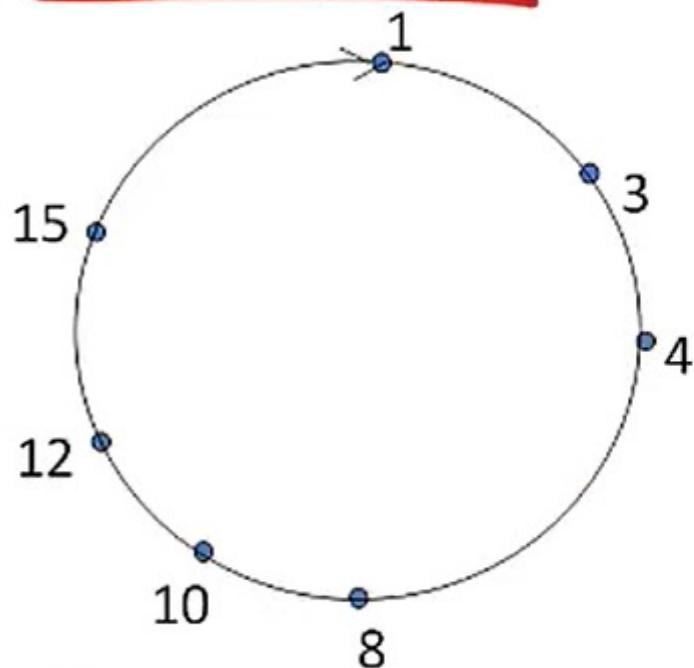


example: peer 5 abruptly leaves

handling peer churn:

- ❖ peers may come and go (churn)
- ❖ each peer knows address of its two successors
- ❖ each peer periodically pings its two successors to check aliveness
- ❖ if immediate successor leaves, choose next successor as new immediate successor

Peer churn



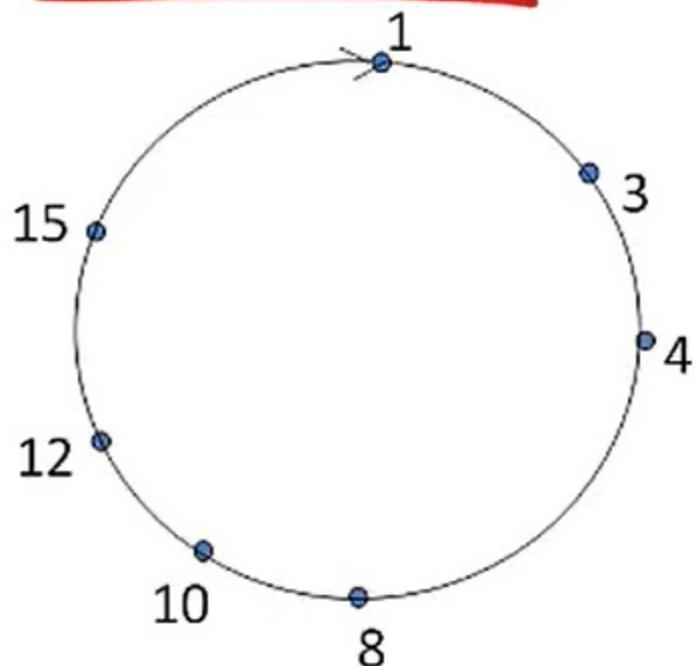
handling peer churn:

- ❖ peers may come and go (churn)
- ❖ each peer knows address of its two successors
- ❖ each peer periodically pings its two successors to check aliveness
- ❖ if immediate successor leaves, choose next successor as new immediate successor

example: peer 5 abruptly leaves

- peer 4 detects peer 5's departure; makes 8 its immediate successor

Peer churn



handling peer churn:

- ❖ peers may come and go (churn)
- ❖ each peer knows address of its two successors
- ❖ each peer periodically pings its two successors to check aliveness
- ❖ if immediate successor leaves, choose next successor as new immediate successor

example: peer 5 abruptly leaves

- peer 4 detects peer 5's departure; makes 8 its immediate successor
- 4 asks 8 who its immediate successor is; makes 8's immediate successor its second successor.

Peer Churn

- What happens to the data (values) that were stored in peer 5 in the previous example?
- Have to duplicate data in multiple peers (maybe the successors)

Member join

- Have to update the ids when new members join the network
- P2P is generally more complex than C/S due to peer join, peer churn and distributed routing

Possibilities with P2P?

- Currently mostly used for file sharing
- Blockchain uses P2P
- Online Social networks?
- Taxi applications?
- Etc etc
- <https://blockonomi.com/youtube-alternative/>
- <https://dailycoin.com/best-decentralized-social-media-platforms-top-10-alternatives-to-consider/>

Vertical vs Horizontal Distribution

- Traditional client-server architectures exhibit **vertical distribution across tiers**. Each tier serves a different purpose in the system.
 - *Logically* different components reside on different nodes
- **Horizontal distribution**: each node has roughly the same processing capabilities and stores/manages part of the total system data.
 - Better load balancing, more resistant to denial-of-service attacks, harder to manage than C/S
 - Communication & control is not hierarchical; all about equal
 - P2P/Hybrid are also examples of horizontal distribution

Volunteer computing

- Seti@Home
- BOINC (<https://boinc.berkeley.edu/>)
- https://en.wikipedia.org/wiki/Volunteer_computing

Architecture versus Middleware

- Where does middleware fit into an architecture?
- Middleware: the software layer between user applications and distributed platforms.
- Purpose: to provide distribution transparency
 - Applications can access programs running on remote nodes without understanding the remote environment

Architecture versus Middleware

- Middleware may also have an architecture
 - e.g., CORBA has an component-based style.
- Use of a specific architectural style can make it easier to develop applications, but it may also lead to a less flexible system.
- Possible solution: develop middleware that can be customized as needed for different applications with different architectures.

Summary

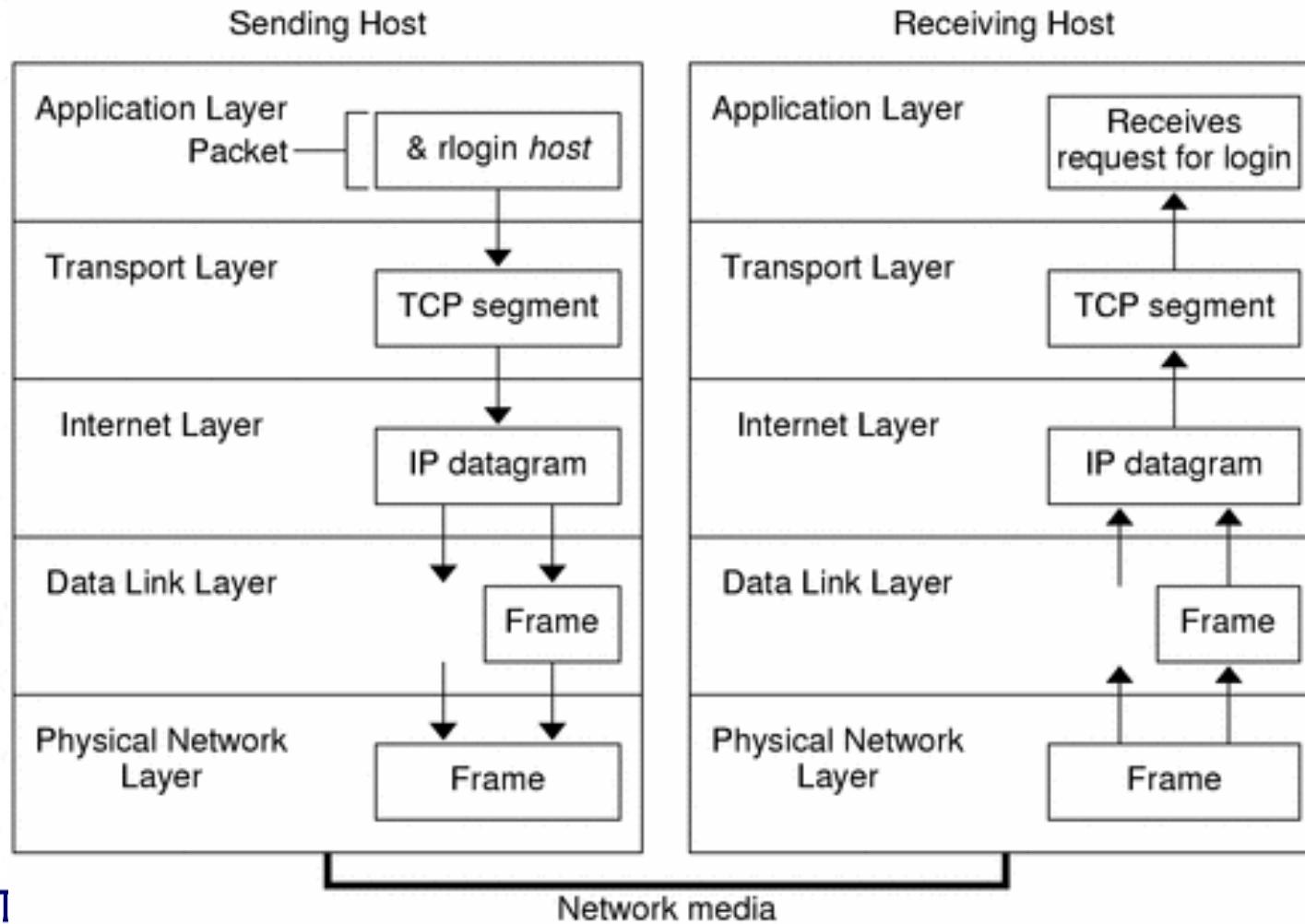
- Software Architecture Vs System Architecture
- Different Software Architectural styles – Layered, Component based, event driven, data centered...
- Can have combinations of these styles
- Different System Architectures – Client Server, Peer to Peer, Hybrid
- P2P – structured/unstructured, Distributed Hash Tables (DHT)

Lecture 3 : Introduction to Socket Programming

What is a socket?

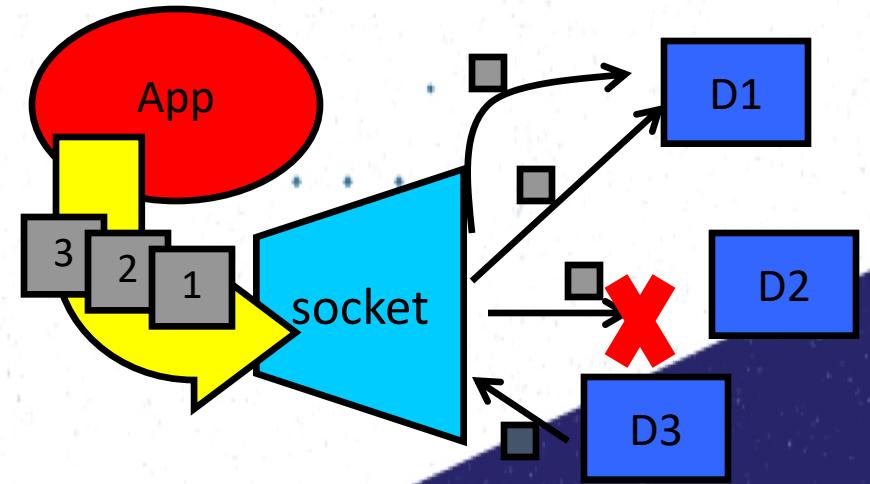
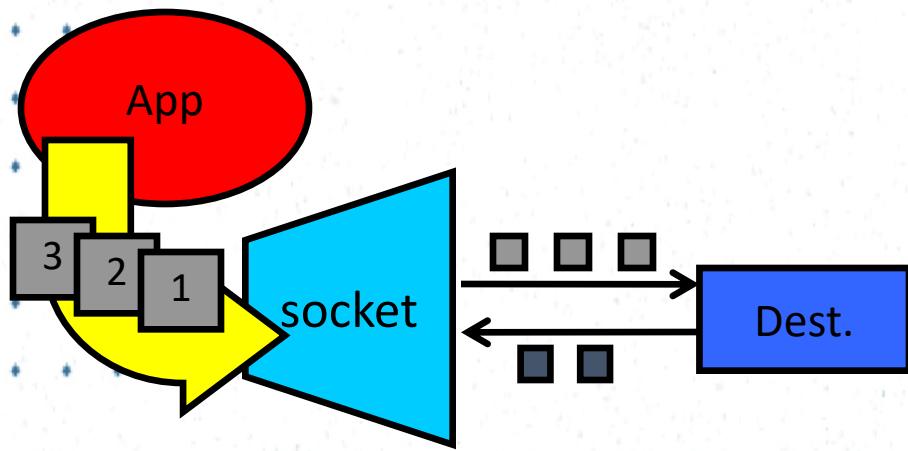
- Port vs. Socket
- An interface between application and network
 - The application creates a socket
 - The socket *type* dictates the style of communication
 - reliable vs. best effort
 - connection-oriented vs. connectionless
- Once configured the application can
 - pass data to the socket for network transmission
 - receive data from the socket (transmitted through the network by some other host)

TCP/IP Stack



Two essential types of sockets

- TCP Socket
 - reliable delivery
 - in-order guaranteed
 - connection-oriented
 - bidirectional
- UDP Socket
 - unreliable delivery
 - no order guarantees
 - no notion of “connection” – app indicates dest. for each packet
 - can send or receive



Applications

- TCP (Transmission control protocol)
 - Point to point chat applications, File transfer (FTP), Email (SMTP)
 - Used when there's a requirement for guaranteed delivery
- UDP (User datagram protocol)
 - Streaming, Multicast/Broadcast
 - Useful when the speed of more important than the assurance of delivery

A Socket-eye view of the Internet



medellin.cs.columbia.edu
(128.59.21.14)



newworld.cs.umass.edu
(128.119.245.93)

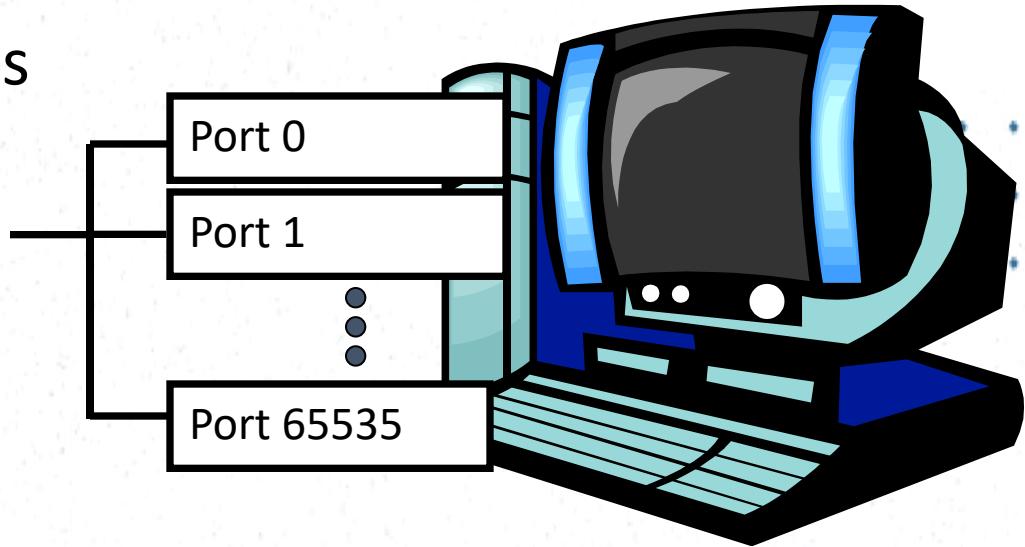


cluster.cs.columbia.edu
(128.59.21.14, 128.59.16.7,
128.59.16.5, 128.59.16.4)

- Each host machine has an IP address
- When a packet arrives at a host

Ports

- Each host has 65,536 ports
- Some ports are *reserved for specific apps*
 - 20,21: FTP
 - 23: Telnet
 - 80: HTTP
 - see RFC 1700 (about 2000 ports are reserved)

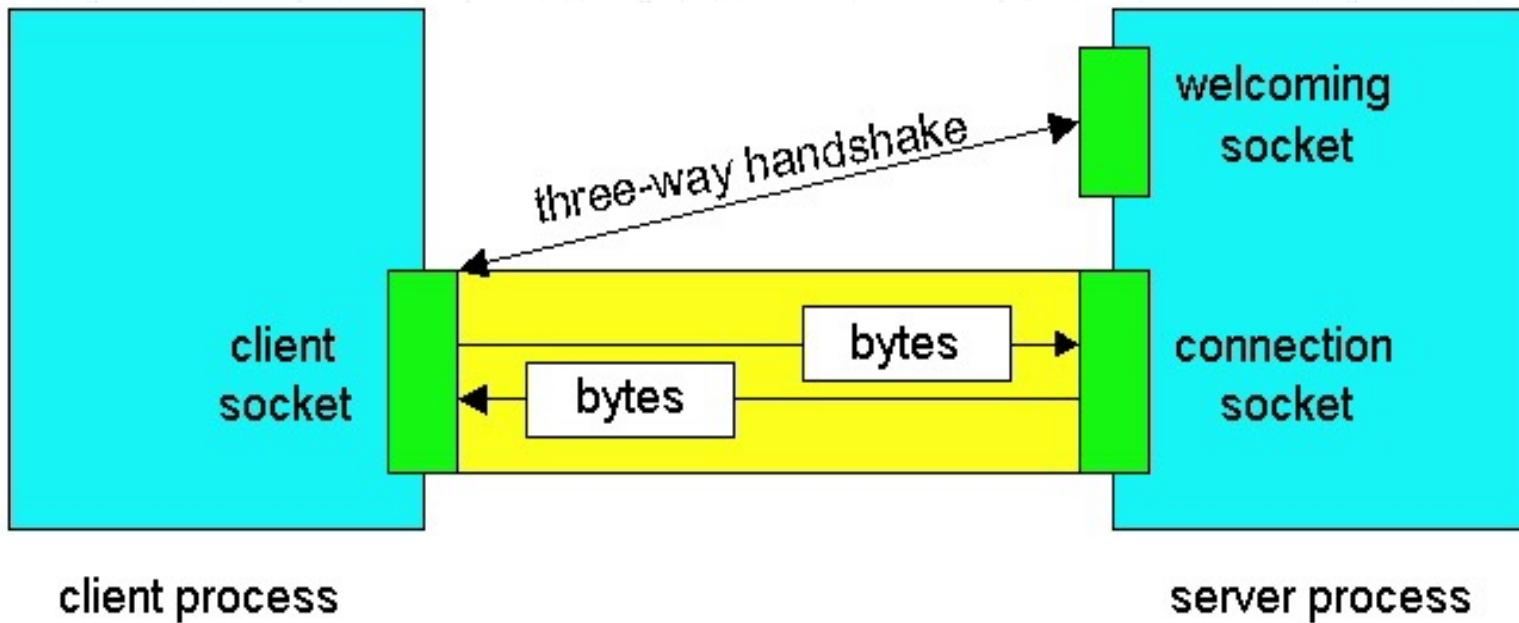


A socket provides an interface to send data to/from the network through a port

Addresses, Ports and Sockets

- In TCP, only one application (process) can listen to a port
- In UDP Multiple applications (processes) may listen to incoming messages on a single port
- Like apartments and mailboxes
 - You are the application
 - Your apartment building address is the address
 - Your mailbox is the port
 - The post-office is the network
 - Each family (process) of the apartment complex (computer) communicates with some same mailbox (port)

TCP Sockets

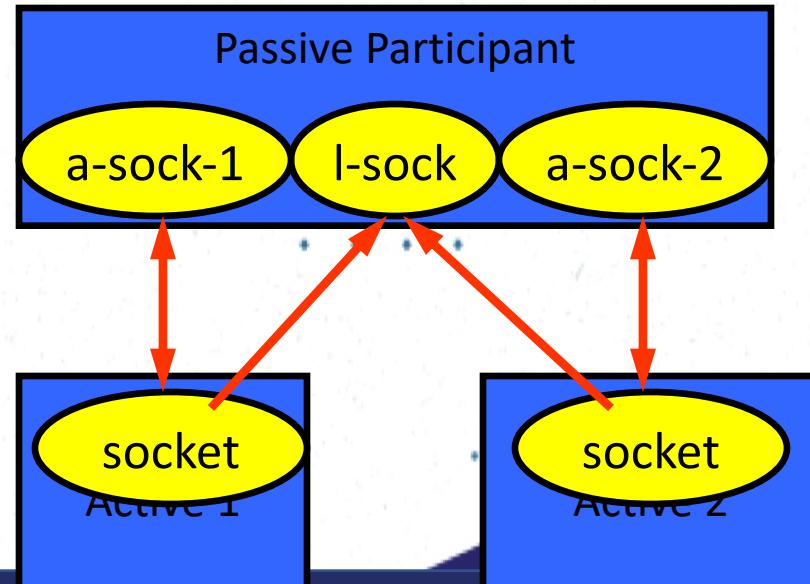


Client socket, welcoming socket (passive) and connection socket (active)

Connection setup

- Passive participant
 - step 1: **listen** (for incoming requests)
 - step 3: **accept** (a request)
 - step 4: data transfer
- The accepted connection is on a new socket
- The old socket continues to listen for other active participants

- Active participant
 - step 2: request & establish **connection**
 - step 4: data transfer



Dealing with blocking

- Calls to sockets can be blocking (no other client may be able to connect to the server)
- Can be resolved using multi-threaded programming
 - Start a new thread for every incoming connection

Java Sockets Programming

- The package `java.net` provides support for sockets programming (and more).
 - Typically you import everything defined in this package with:

```
import java.net.*;
```

Classes

InetAddress

Socket

ServerSocket

DatagramSocket

DatagramPacket

InetAddress class

- Static methods you can use to create new InetAddress objects.
 - `getByName(String host)`
 - `getAllByName(String host)`
 - `getLocalHost()`

```
InetAddress x = InetAddress.getByName(  
    "cse.unr.edu");
```

❖ Throws `UnknownHostException`

```
try {  
  
    InetAddress a = InetAddress.getByName(hostname);  
  
    System.out.println(hostname + ":" +  
        a.getHostAddress());  
  
} catch (UnknownHostException e) {  
  
    System.out.println("No address found for " +  
        hostname);  
  
}
```

Socket class

- Corresponds to active TCP sockets only!
 - client sockets
 - socket returned by accept();
- Passive sockets are supported by a different class:
 - ServerSocket
- UDP sockets are supported by
 - DatagramSocket

JAVA TCP Sockets

- `java.net.Socket`
 - Implements client sockets (also called just “sockets”).
 - An endpoint for communication between two machines.
 - Uses input/output streams to pass messages
 -
- `java.net.ServerSocket`
 - Implements server sockets.
 - Waits for requests to come in over the network.
 - Accepts the client connection requests
 - Performs some operation based on each request

Socket Constructors

- Constructor creates a TCP connection to a named TCP server.
 - There are a number of constructors:

```
Socket(InetAddress server, int port);  
...  
Socket(InetAddress server, int port,  
       InetAddress local, int localport);  
...  
Socket(String hostname, int port);  
...
```

Socket Methods

```
void close();  
InetAddress getInetAddress();  
InetAddress getLocalAddress();  
InputStream getInputStream();  
OutputStream getOutputStream();  
...
```

- Lots more (setting/getting socket options, partial close, etc.)

Socket I/O

- Socket I/O is based on the Java I/O support
 - in the package `java.io`
- `InputStream` and `OutputStream` are abstract classes
 - common operations defined for all kinds of `InputStreams`, `OutputStreams`...

InputStream Basics

```
// reads some number of bytes and  
// puts in buffer array b  
int read(byte[] b);  
  
// reads up to len bytes  
int read(byte[] b, int off, int len);
```

Both methods can throw **IOException**.

Both return -1 on EOF.

OutputStream Basics

```
// writes b.length bytes  
void write(byte[] b);  
  
// writes len bytes starting  
// at offset off  
void write(byte[] b, int off, int len);  
.....
```

Both methods can throw **IOException**.

ServerSocket Class (TCP Passive Socket)

- Constructors:

```
ServerSocket(int port);
```

```
ServerSocket(int port, int backlog);
```

```
ServerSocket(int port, int backlog,  
            InetAddress bindAddr);
```

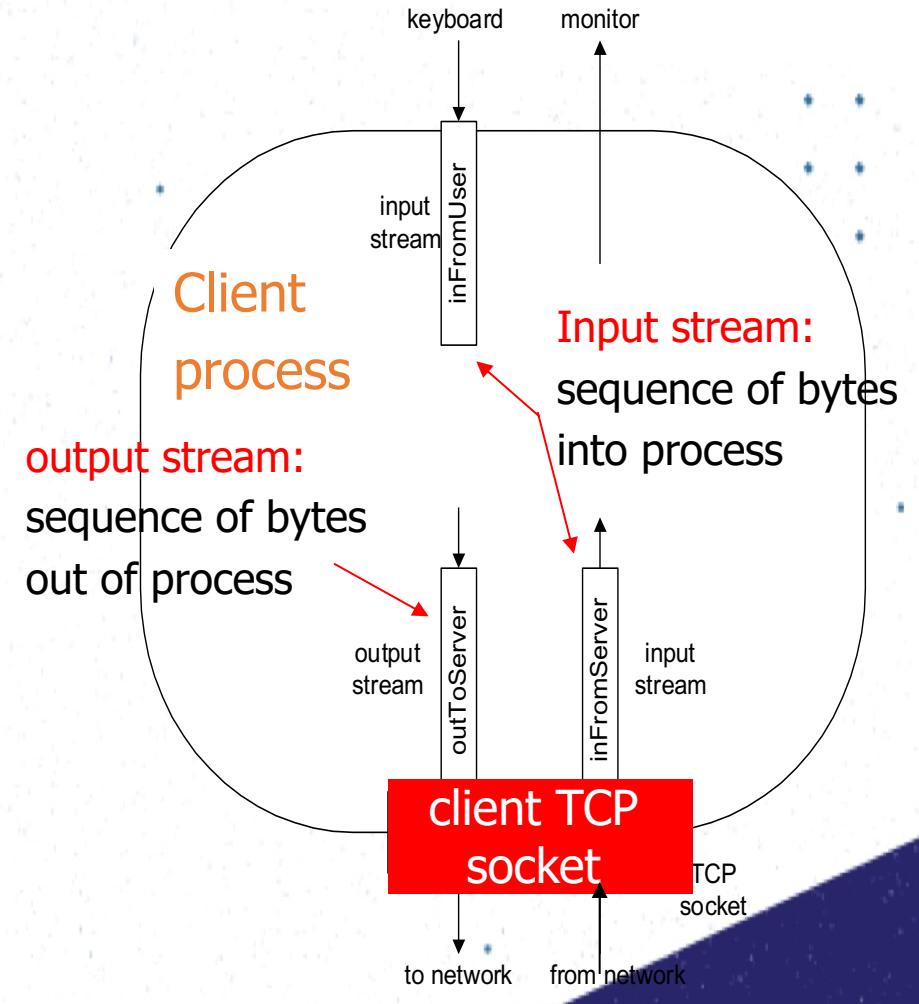
ServerSocket Methods

```
Socket accept();  
void close();  
InetAddress getInetAddress();  
int getLocalPort();  
throw IOException, SecurityException
```

Socket programming with TCP

Example client-server app:

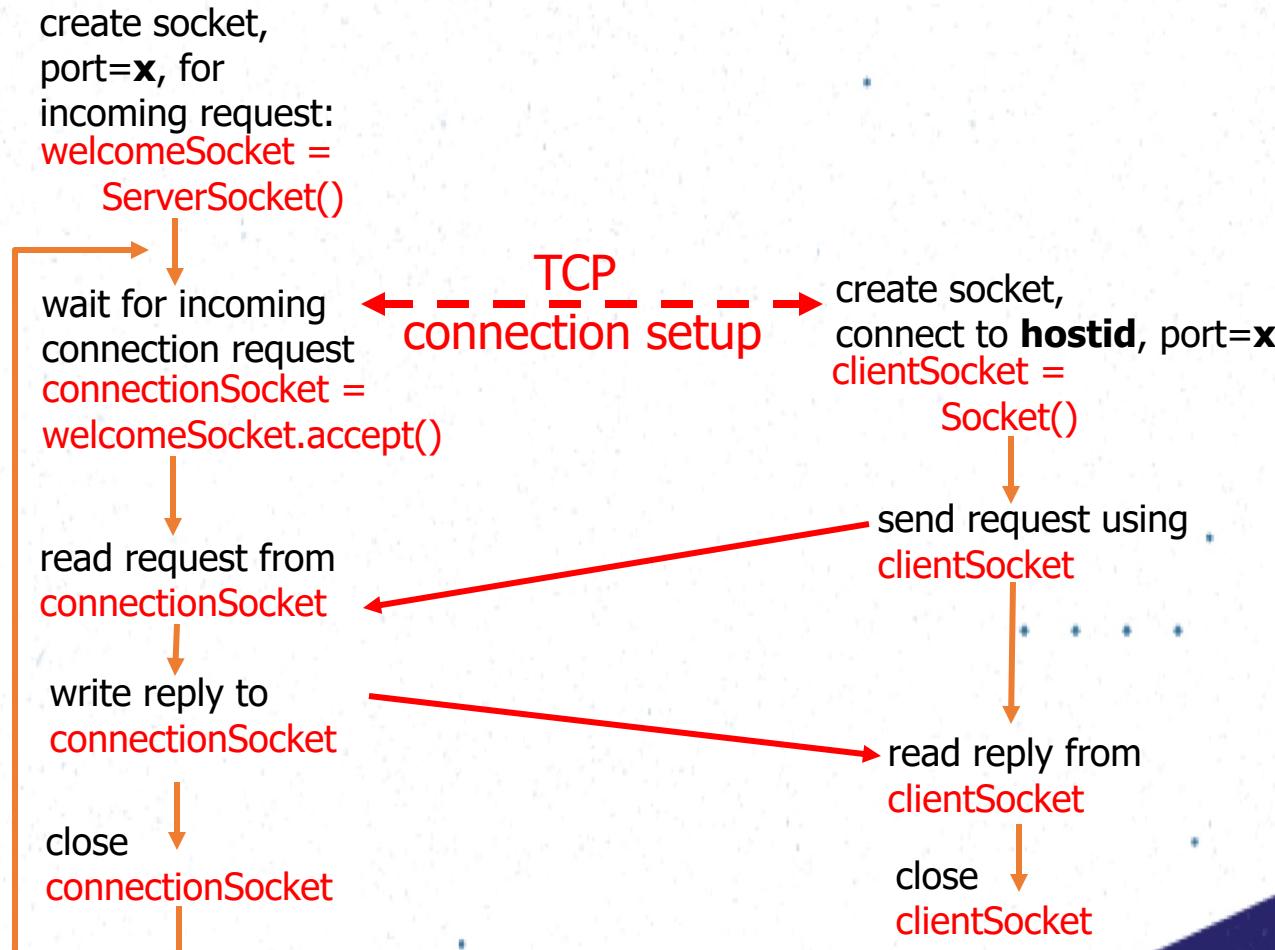
- client reads line from standard input (**inFromUser** stream), sends to server via socket (**outToServer** stream)
- server reads line from socket
- server converts line to uppercase, sends back to client
- client reads, prints modified line from socket (**inFromServer** stream)



Client/server socket interaction: TCP

Server (running on **hostid**)

Client



TCPClient.java

```
import java.io.*;
import java.net.*;

class TCPClient {
    public static void main(String argv[]) throws Exception
    {
        String sentence;
        String modifiedSentence;

        BufferedReader inFromUser =
            new BufferedReader(new InputStreamReader(System.in));

        Socket clientSocket = new Socket("hostname", 6789);

        DataOutputStream outToServer =
            new DataOutputStream(clientSocket.getOutputStream());
    }
}
```

TCPClient.java

```
BufferedReader inFromUser =  
    new BufferedReader(new  
InputStreamReader(clientSocket.getInputStream()));  
  
sentence = inFromUser.readLine();  
  
    . . .  
  
outToServer.writeBytes(sentence + '\n');  
  
    . . .  
  
modifiedSentence = inFromServer.readLine();  
  
    . . .  
  
System.out.println("FROM SERVER: " + modifiedSentence);  
  
    . . .  
  
clientSocket.close();  
  
}  
}
```

TCPServer.java

```
import java.io.*;
import java.net.*;
class TCPServer {
    public static void main(String argv[]) throws Exception
    {
        String clientSentence;
        String capitalizedSentence;
        . . .
ServerSocket welcomeSocket = new ServerSocket(6789);
        while(true) {
            Socket connectionSocket = welcomeSocket.accept();
            BufferedReader inFromClient = new BufferedReader(new
                InputStreamReader(connectionSocket.getInputStream()));
        }
    }
}
```

TCPServer.java

```
    DataOutputStream outToClient =  
        new DataOutputStream(connectionSocket.getOutputStream());
```

```
    clientSentence = inFromClient.readLine();
```

```
    capitalizedSentence = clientSentence.toUpperCase() + '\n';
```

```
    outToClient.writeBytes(capitalizedSentence);
```

```
}
```

```
}
```

UDP Sockets

- DatagramSocket class
- DatagramPacket class needed to specify the payload
 - incoming or outgoing

Socket Programming with UDP

- UDP
 - Connectionless and unreliable service.
 - There isn't an initial handshaking phase.
 - Doesn't have a pipe.
 - Transmitted data may be received out of order, or lost
- Socket Programming with UDP
 - No need for a welcoming socket.
 - No streams are attached to the sockets.
 - the sending hosts creates “packets” by attaching the IP destination address and port number to each batch of bytes.
 - The receiving process must unravel to received packet to obtain the packet's information bytes.

JAVA UDP Sockets

- In Package `java.net`
 - `java.net.DatagramSocket`
 - A socket for sending and receiving datagram packets.
 - Constructor and Methods
 - `DatagramSocket(int port)`: Constructs a datagram socket and binds it to the specified port on the local host machine.
 - `void receive(DatagramPacket p)`
 - `void send(DatagramPacket p)`
 - `void close()`

DatagramSocket Constructors

`DatagramSocket() ;`

`DatagramSocket(int port) ;`

`DatagramSocket(int port, InetAddress a) ;`

All can throw SocketException or SecurityException

Datagram Methods

```
void connect(InetAddress, int port);  
.  
.  
.  
void close();  
.  
.  
.  
void receive(DatagramPacket p);  
.  
.  
.  
void send(DatagramPacket p);  
.
```

Lots more!

Datagram Packet

- Contain the payload
 - a byte array
- Can also be used to specify the destination address
 - when not using connected mode UDP

DatagramPacket Constructors

For receiving:

```
DatagramPacket( byte[] buf, int len);
```

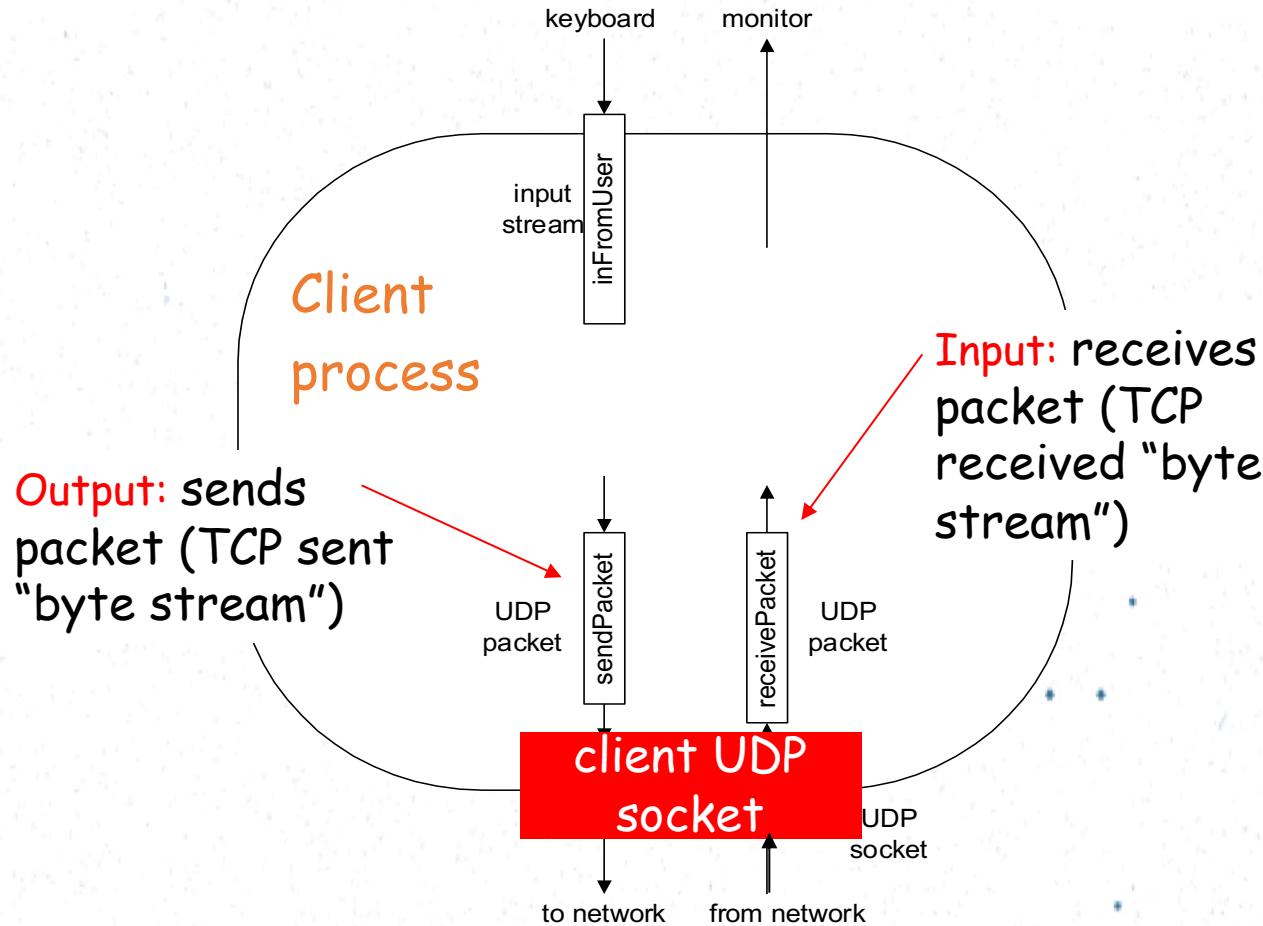
For sending:

```
DatagramPacket( byte[] buf, int len  
InetAddress a, int port);
```

DatagramPacket methods

```
byte[] getData();  
void setData(byte[] buf);  
void setAddress(InetAddress a);  
void setPort(int port);  
  
InetAddress getAddress();  
int getPort();  
...
```

Example: Java client (UDP)



Client/server socket interaction: UDP

Server (running on **hostid**)

create socket,
port=x, for
incoming request:
`serverSocket =
DatagramSocket()`

read request from
`serverSocket`

write reply to
`serverSocket`
specifying client
host address,
port number

Client

create socket,
`clientSocket =
DatagramSocket()`

Create, address (**hostid, port=x**,
send datagram request
using `clientSocket`

read reply from
`clientSocket`

close
`clientSocket`

UDPClient.java

```
import java.io.*;
import java.net.*;

class UDPClient {
    public static void main(String args[]) throws Exception
    {

        BufferedReader inFromUser =
            new BufferedReader(new InputStreamReader(System.in));

DatagramSocket clientSocket = new DatagramSocket();

InetAddress IPAddress =
InetAddress.getByName("hostname");

        byte[] sendData = new byte[1024];
        byte[] receiveData = new byte[1024];

        String sentence = inFromUser.readLine();

        sendData = sentence.getBytes();
    }
}
```

UDPClient.java

```
DatagramPacket sendPacket =  
    new DatagramPacket(sendData, sendData.length,  
IPAddress, 9876);  
  
clientSocket.send(sendPacket);  
  
DatagramPacket receivePacket =  
    new DatagramPacket(receiveData, receiveData.length);  
  
clientSocket.receive(receivePacket);  
  
String modifiedSentence =  
    new String(receivePacket.getData());  
  
System.out.println("FROM SERVER:" + modifiedSentence);  
  
clientSocket.close();  
  
}  
}
```

UDPServer.java

```
import java.io.*;
import java.net.*;

class UDPServer {
    public static void main(String args[]) throws Exception
    {
        DatagramSocket serverSocket = new
        DatagramSocket(9876);

        byte[] receiveData = new byte[1024];
        byte[] sendData = new byte[1024];

        while(true)
        {
            DatagramPacket receivePacket =
                new DatagramPacket(receiveData, receiveData.length);

            serverSocket.receive(receivePacket);

            String sentence = new String(receivePacket.getData());
    }
}
```

UDPServer.java

```
InetAddress IPAddress = receivePacket.getAddress();  
int port = receivePacket.getPort();  
String capitalizedSentence = sentence.toUpperCase();  
sendData = capitalizedSentence.getBytes();  
DatagramPacket sendPacket =  
    new DatagramPacket(sendData, sendData.length, IPAddress, port);  
serverSocket.send(sendPacket);  
.  
.  
.  
}  
}  
}
```

TCP vs HTTP

- <https://networkinterview.com/http-vs-tcp-know-the-difference/>

WebSockets

- <https://www.wallarm.com/what/a-simple-explanation-of-what-a-websocket-is>

Summary

- Socket programming is the most fundamental form of Client-Server distributed computing available for app. developers
- Can be used to develop client-server distributed applications (e.g. Messaging applications)
- However, most real-world distributed systems use more high level distributed computing technologies (E.g. Web services, EJBs)
- Yet the underlying communication mechanism of these high level Dist. Computing frameworks is socket communication

Lecture 2 - Concurrency and Multithreading

Outline

- Appreciate the (increasing) importance of parallel programming
- Understand fundamental concepts:
 - Parallelism, threads, multi-threading, concurrency, locks, etc.
- See some basics of this is done in Java
- See some common uses:
 - Divide and conquer, e.g. mergesort
 - Worker threads in Swing

Background

- An area of rapid change!
 - 1990s: parallel computers were \$\$\$\$
 - Now: 4 core machines are commodity
- Variations between languages
- Evolving frameworks, models, etc.
 - E.g. Java's getting Fork/Join since Java 1.7
 - MAP/REDUCE

(Multi)Process vs (Multi)Thread

- Assume a computer has one CPU
- Can only execute one statement at a time
 - Thus one program at a time
- Process: an operating-system level “unit of execution”
- Multi-processing
 - Op. Sys. “time-slices” between processes
 - Computer appears to do more than one program (or background process) at a time

Multicore vs Multithreading

- Modern CPUs have multiple cores that can execute multiple processes at the same time
- Multiple threads can run even in a single core CPU
- Multithreading can utilize the multiple cores
- Still, it is the job of the operating system to schedule and run the parallel tasks

Advantages of Multithreading

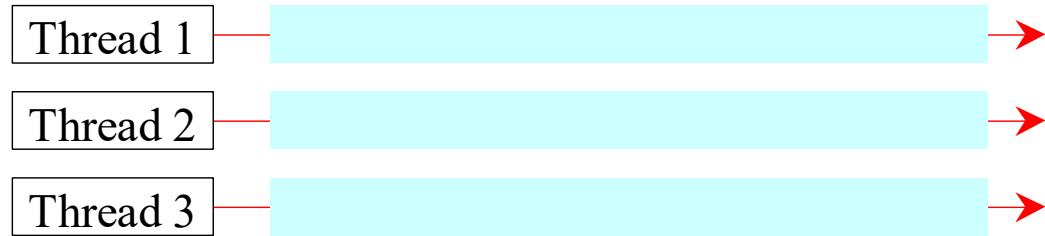
- Reactive systems – constantly monitoring
- More responsive to user input – GUI application can interrupt a time-consuming task
- Server can handle multiple clients simultaneously
- Can take advantage of parallel processing

Multithreading

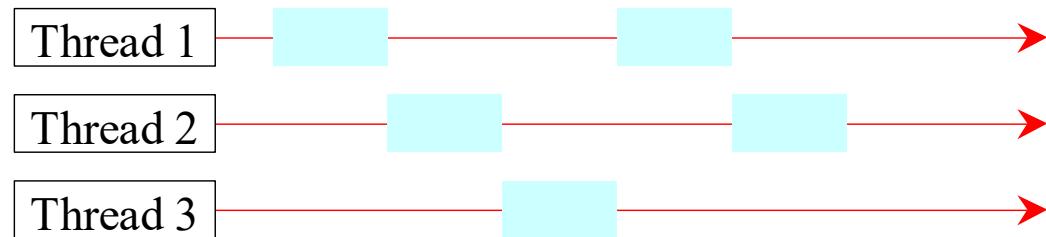
- Different processes do not share memory space.
- A thread can execute concurrently with other threads within a single process.
- All threads managed by the JVM share memory space and can communicate with each other.

Threads Concept

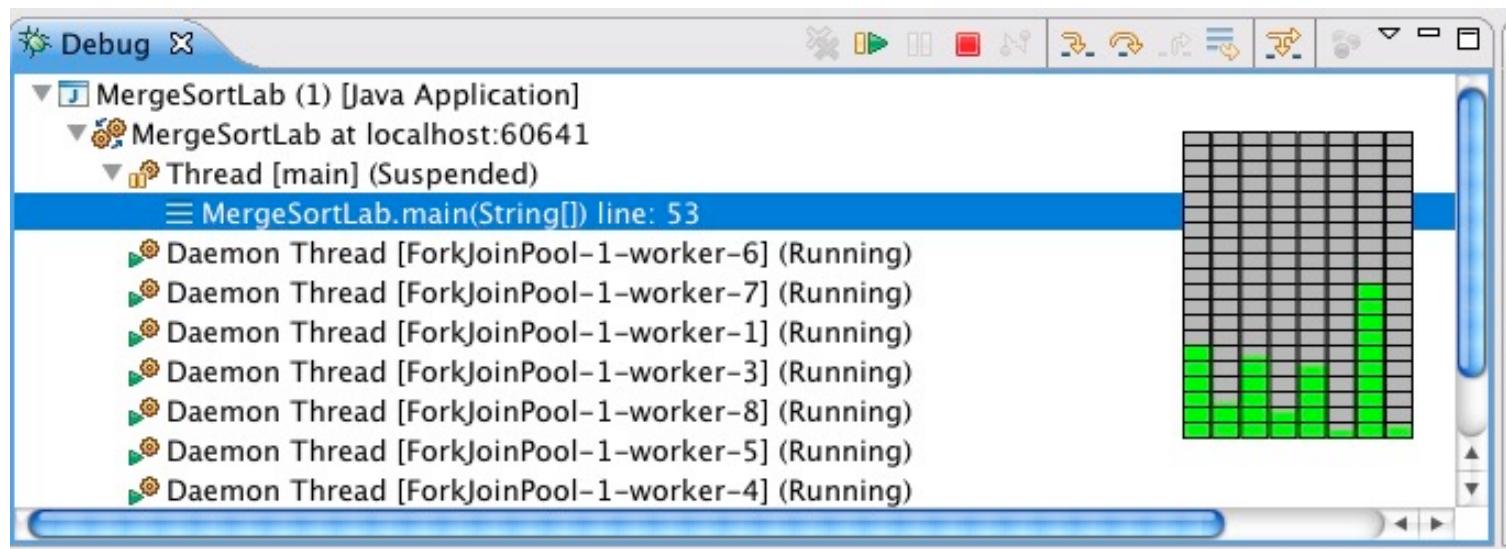
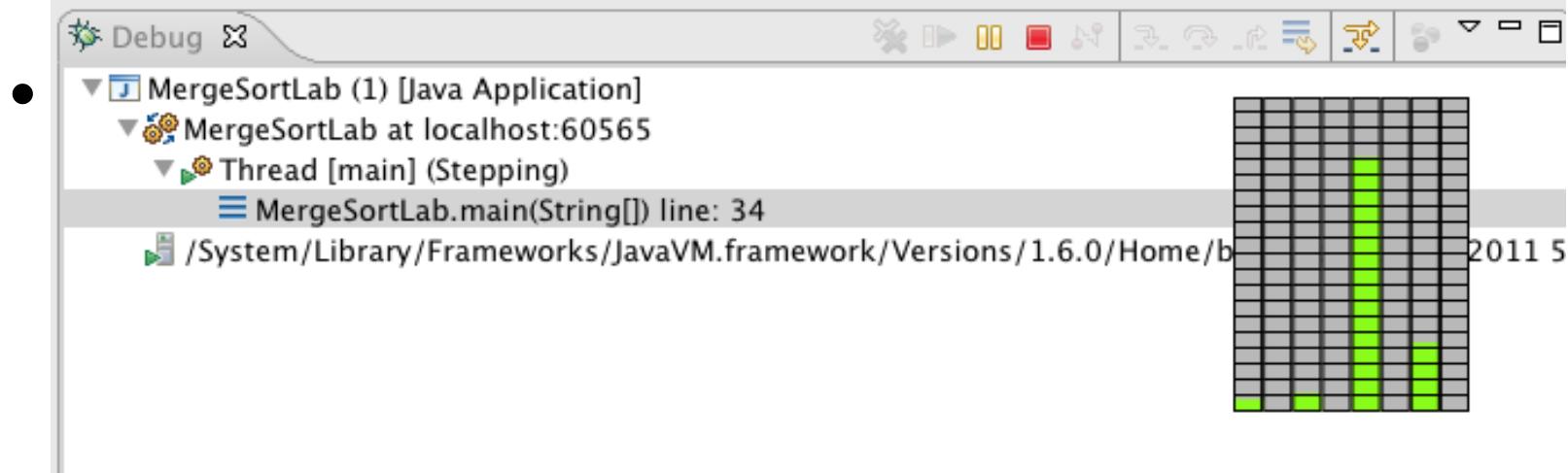
Multiple threads on multiple CPUs



Multiple threads sharing a single CPU



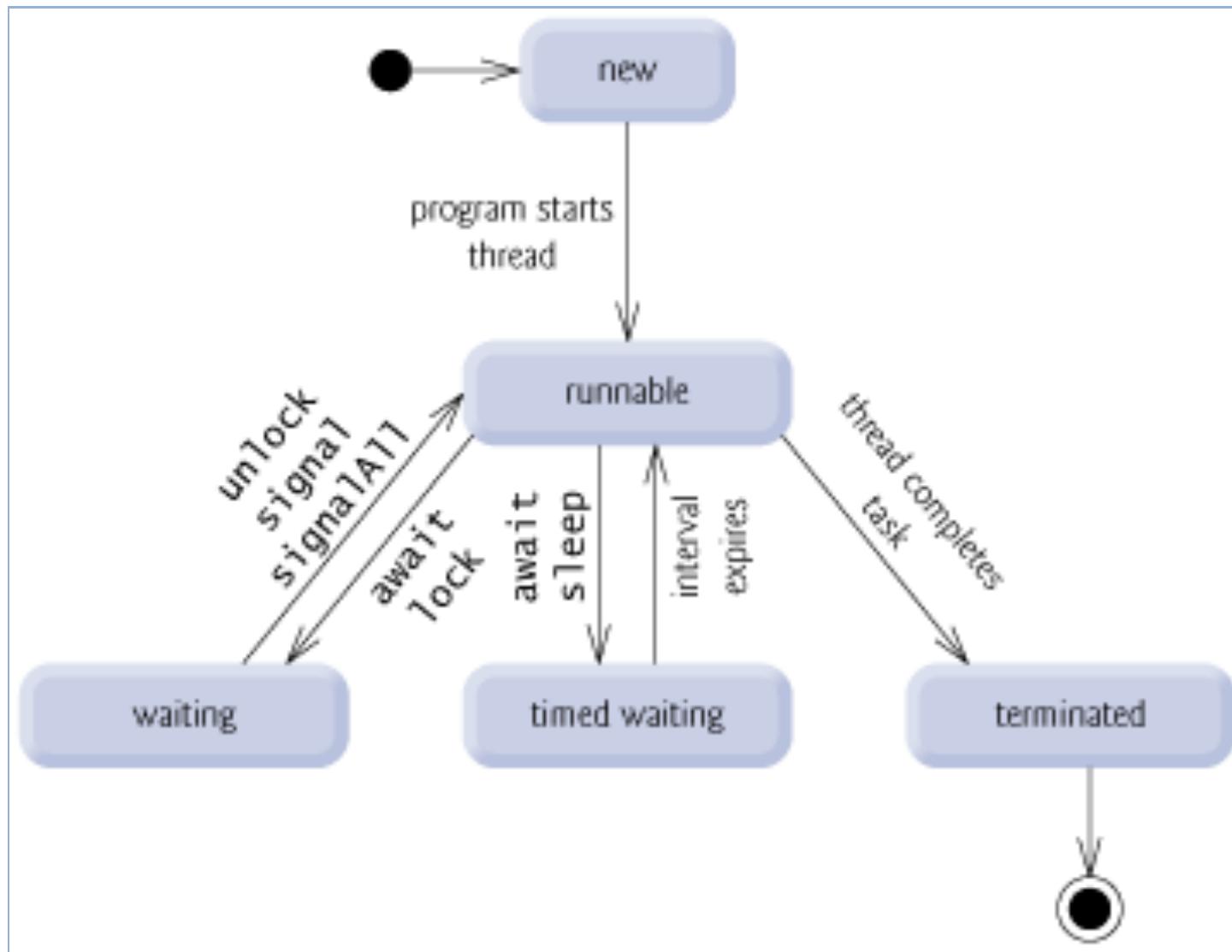
Screenshots: For single- and multi-threaded Mergesort: Threads in Eclipse Debug window, and Mac's CPU usage display



Tasks and Threads

- **Thread**: “a thread of execution”
 - “Smaller”, “lighter” than a **process**
 - smallest unit of processing that can be scheduled by an operating system
 - Has its own run-time call stack, copies of the CPU’s registers, its own program counter, etc.
 - Process has its own memory address space, but threads share one address space
- A single program can be multi-threaded
 - Time-slicing done just like in multiprocessing
 - Repeat: the threads share the same memory

Thread States



Task

- A task is an abstraction of a series of steps
 - Might be done in a separate thread
 - Parallelizable
- In Java, there are a number of classes / interfaces that basically correspond to this
 - Example (details soon): Runnable
 - work done by method run()

Java: Statements → Tasks

- Consecutive lines of code:

```
Foo tmp = f1;  
f1 = f2;  
f2 = tmp;
```

- A method:

```
swap(f1, f2);
```

- A “task” object:

```
SwapTask task1= new SwapTask(f1, f2);  
task1.run();
```

Why a task object?

- Actions, functions vs. objects. What's the difference?

Why a task object?

- Actions, functions vs. objects. What's the difference?
- Objects:
 - Are persistent. Can be stored.
 - Can be created and then used later.
 - Can be attached to other things. Put in Collections.
 - Contain state.
- Functions:
 - Called, return (not permanent)

Java Library Classes for Concurrency

- For task-like things:
 - Runnable, Callable
 - SwingWorker, RecursiveAction, etc.
- Thread class
- Managing tasks and threads
 - Executor, ExecutorService
 - ForkJoinPool
- In Swing
 - The Event-Dispatch Thread
 - SwingUtilities.invokeLater()

Java Thread Classes and Methods

- Java has some “primitives” for creating and using threads
 - Most sources teach these, but in practice they’re hard to use well
 - Now, better frameworks and libraries make using them directly less important.
- But let’s take a quick look

Java's Thread Class

- Class Thread: its method run() does its business when that thread is run
- But you never call run(). Instead, you call start() which lets Java start it and call run()
- To use Thread class directly (not recommended now):
 - define a subclass of Thread and override run() – not recommended!
 - Create a task as a Runnable, link it with a Thread, and then call start() on the Thread.
 - The Thread will run the Runnable's run() method.

Creating a Task and Thread

- Again, the first of the two “old” ways
- Get a thread object, then call start() on that object
 - Makes it available to be run
 - When it’s time to run it, Thread’s run() is called
- So, create a thread using inheritance
 - Write class that extends Thread, e.g. MyThread
 - Define your own run()
 - Create a MyThread object and call start() on it
- Not good design!

Runnables and Thread

- Use the “task abstraction” and create a class that implements Runnable interface
 - Define the run() method to do the work you want
- Now, two ways to make your task run in a separate thread
 - First way:
 - Create a Thread object and pass a Runnable to the constructor
 - As before, call start() on the Thread object
 - Second way: hand your Runnable to a “thread manager” object

Creating Tasks and Threads

`java.lang.Runnable`

`TaskClass`

```
// Custom task class
public class TaskClass implements Runnable {
    ...
    public TaskClass(...) {
        ...
    }

    // Implement the run method in Runnable
    public void run() {
        // Tell system how to run custom thread
        ...
    }
    ...
}
```

```
// Client class
public class Client {
    ...
    public void someMethod() {
        ...
        // Create an instance of TaskClass
        TaskClass task = new TaskClass(...);

        // Create a thread
        Thread thread = new Thread(task);

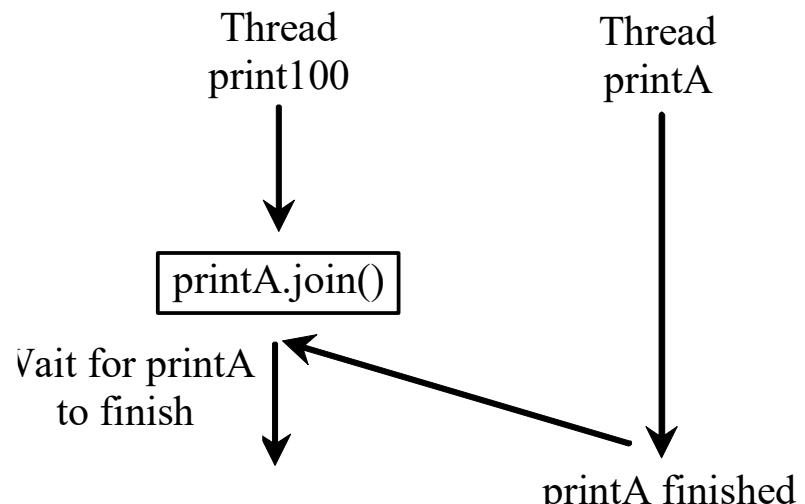
        // Start a thread
        thread.start();
        ...
    }
    ...
}
```

Join

- The **Thread** class defines various primitive methods you could not implement on your own
 - For example: **start**, which calls **run** in a new thread
- The **join()** method is one such method, essential for coordination in this kind of computation
 - Caller blocks until/unless the receiver is done executing (meaning its **run** returns)
 - E.g. in method `foo()` running in “main” thread, we call:
`myThread.start(); myThread.join();`
 - Then this code waits (“blocks”) until `myThread’s run()` completes
- This style of parallel programming is often called “fork/join”
 - Warning: we’ll soon see a library called “fork/join” which simplifies things. In that, you never call `join()`

Join

```
public void run() {  
    Thread thread4 = new Thread(  
        new PrintChar('c', 40));  
    thread4.start();  
    try {  
        for (int i = 1; i <= lastNum; i++) {  
            System.out.print(" " + i);  
            if (i == 50) thread4.join();  
        }  
    }  
    catch (InterruptedException ex) {  
    }  
,
```



Threading in Swing

- Threading matters a lot in Swing GUIs
 - You know: main's thread ends “early”
 - `JFrame.setVisible(true)` starts the “GUI thread”
- Swing methods run in a separate thread called the **Event-Dispatching Thread (EDT)**
 - Why? GUIs need to be responsive quickly
 - Important for good user interaction
- But: slow tasks can block the EDT
 - Makes GUI seem to hang
 - Doesn't allow parallel things to happen

Thread Rules in Swing

- All operations that update GUI components must happen in the EDT
 - These components are not thread-safe (later)
 - `SwingUtilities.invokeLater(Runnable r)` is a method that runs a task in the EDT when appropriate
- But execute slow tasks in separate *worker threads*
- To make common tasks easier, use a `SwingWorker` task

SwingWorker

- A class designed to be extended to define a task for a worker thread
 - Override method **doInBackground()**
This is like run() – it's what you want to do
 - Override method **done()**
This method is for updating the GUI afterwards
 - It will be run in the EDT
- Note you can get interim results too

Java ForkJoin Framework

- Designed to support a common need
 - Recursive divide and conquer code
 - Look for small problems, solve without parallelism
 - For larger problems
 - Define a task for each subproblem
 - Library provides
 - a Thread manager, called a ForkJoinPool
 - Methods to send your subtask objects to the pool to be run, and your call waits until their done
 - The pool handles the multithreading well

The ForkJoinPool

- The “thread manager”
 - Used when calls are made to RecursiveTask’s methods fork(), invokeAll(), etc.
 - When created, knows how many processors are available
 - Pretty sophisticated
 - “Steals” time from threads that have nothing to do

Overview of How To

- Create a ForkJoinPool “thread-manager” object
- Create a task object that extends RecursiveTask
 - Create a task-object for entire problem and call invoke(task) on your ForkJoinPool
- Your task class’ compute() is like Thread.run()
 - It has the code to do the divide and conquer
 - First, it must check if small problem – don’t use parallelism, solve without it
 - Then, divide and create >1 new task-objects. Run them:
 - Either with invokeAll(task1, task2, ...). Waits for all to complete.
 - Or calling fork() on first, then compute() on second, then join()

Same Ideas as Thread But...

To use the ForkJoin Framework:

- A little standard set-up code (e.g., create a **ForkJoinPool**)

Don't subclass **Thread**

Do subclass **RecursiveAction<V>**

Don't override **run**

Do override **compute**

Don't call **start**

Do call **invoke**, **invokeAll**, **fork**

Don't just call **join**
or

Do call **join** which returns answer

Do call **invokeAll** on multiple tasks

Mergesort Example

- Top-level call. Create “main” task and submit

```
public static void mergeSortFJRecur(Comparable[] list, int first,  
        int last) {  
    if (last - first < RECURSE_THRESHOLD) {  
        MergeSort.insertionSort(list, first, last);  
        return;  
    }  
    Comparable[] tmpList = new Comparable[list.length];  
    threadPool.invoke(new SortTask(list, tmpList, first, last));  
}
```

Mergesort's Task-Object Nested Class

```
static class SortTask extends RecursiveAction {  
    Comparable[] list;  
    Comparable[] tmpList;  
    int first, last;  
    public SortTask(Comparable[] a, Comparable[] tmp,  
        int lo, int hi) {  
        this.list = a;    this.tmpList = tmp;  
        this.first = lo; this.last = hi;  
    }  
    // continued next slide
```

compute() Does Task Recursion

```
protected void compute() { // in SortTask, continued from previous slide
    if (last - first < RECURSE_THRESHOLD)
        MergeSort.insertionSort(list, first, last);
    else {
        int mid = (first + last) / 2;
        // the two recursive calls are replaced by a call to
        invokeAll
        SortTask task1 = new SortTask(list, tmpList, first, mid);
        SortTask task2 = new SortTask(list, tmpList, mid+1, last);
        invokeAll(task1, task2);
        MergeSort.merge(list, first, mid, last);
    }
}
```

Nice to Have a Thread “Manager”

- If your code is responsible for creating a bunch of tasks, linking them with Threads, and starting them all, then you have much to worry about:
 - What if you start too many threads? Can you manage the number of running threads?
 - Enough processors?
 - Can you shutdown all the threads?
 - If one fails, can you restart it?

Executors

- An Executor is an object that manages running tasks
 - Submit a Runnable to be run with Executor's execute() method
 - So, instead of creating a Thread for your Runnable and calling start() on that, do this:
 - Get an Executor object, say called exec
 - Create a Runnable, say called myTask
 - Submit for running: exec.execute(myTask)

How to Get an Executor

- Use static methods in Executors library.
- Fixed “thread pool”: at most N threads running at one time

```
Executor exec =
```

```
    Executors.newFixedThreadPool(MAX_THREADS);
```

- Unlimited number of threads

```
Executor exec =
```

```
    Executors.newCachedThreadPool();
```

Summary So Far

- Create a class that implements a Runnable to be your “task object”
 - Or if ForkJoin framework, extend RecursiveTask
- Create your task objects
- Create an Executor
 - Or a ForkJoinPool
- Submit each task-object to the Executor which starts it up in a separate thread

Concurrency and Synchronization

- **Concurrency:**

Multiple-threads/Processes accessing shared data

- **Synchronization:**

Methods to manage and control concurrent access to shared data by multiple-threads

Possible Bugs in Multithreaded Code

- Possible bug #1

```
i=1; x=10; x = i + x; // x could be 12 here
```

- Possible bug #2

```
if ( ! myList.contains(x) )  
    myList.add(x); // x could be in list twice
```

- Why could these cause unexpected results?

How $1 + 10$ might be 12

- Thread 1 executes:

(x is 10, i is 1)

- Get i (1) into register 1
- Get x (10) into its register 2
(other thread has CPU)
- Add registers
- Store result (11) into x

(x is now 11)

(other thread has CPU)

(other thread has CPU)

- Do next line of code
(x changes to 12 even though no code in this thread has touched x)

- Thread 2 executes:

(x is 10, i is 1)

(other thread has CPU)

(other thread has CPU)

- Get i (1) into its register 1
(other thread has CPU)
- Get x (11) into is register 2
(other thread has CPU)
- Add registers
- Store result (12) into x

(x is now 12)

Synchronization

- Understand the issue with concurrent access to shared data?
 - Data could be a counter (int) or a data structure (e.g. a Map or List or Set)
- A **race condition**: Two threads will access something. They “compete” causing a problem
- A **critical section**: a block of code that can only be safely executed by one thread at a time
- A lock: an object that is “held” by one thread at a time, then “released”

Synchronized Methods

- Common situation: all the code in a method is a critical section
 - I.e. only one thread at a time should execute that method
 - E.g. a getter or setter or mutator, or something that changes shared state info (e.g. a Map of important data)
- Java makes it easy: add `synchronized` keyword to method signature. E.g.

```
public synchronized void update(...) {
```

```
public class Counter {  
    private int counter;  
    public synchronized void increment() {  
        counter++;  
    }  
  
    public int read() {  
        return counter;  
    }  
}
```

Synchronization using Locks

- Any object can serve as a lock
 - Separate object: `Object myLock = new Object();`
 - Current instance: the `this` object
- Enclose lines of code in a *synchronized* block

```
synchronized(myLock) {
    // code here
}
```
- More than one thread could try to execute this code, but one acquires the lock and the others “block” or wait until the first thread releases the lock
- More fine grained than method synchronization (more prone to errors)

Summary So Far

- Concurrent access to shared data
 - Can lead to serious, hard-to-find problems
 - E.g. race conditions
- The concept of a lock
- Synchronized blocks of code or methods
 - One thread at a time
 - While first thread is executing it, others block

Some Java Solutions

- There are some synchronized collections
 - Classes like AtomicInteger
 - Stores an int
 - Has methods to operate on it in a thread-safe manner
- int getAndAdd(int delta) instead of i=i+1

Volatile keyword

- Compilers cache variable values from the memory in the registers for faster performance
- This can sometimes lead to errors in execution in a multithreaded program
- Volatile keyword ensures that each variable update is visible to all threads
- However, it does not ensure atomicity of operations (which has to be handled separately)

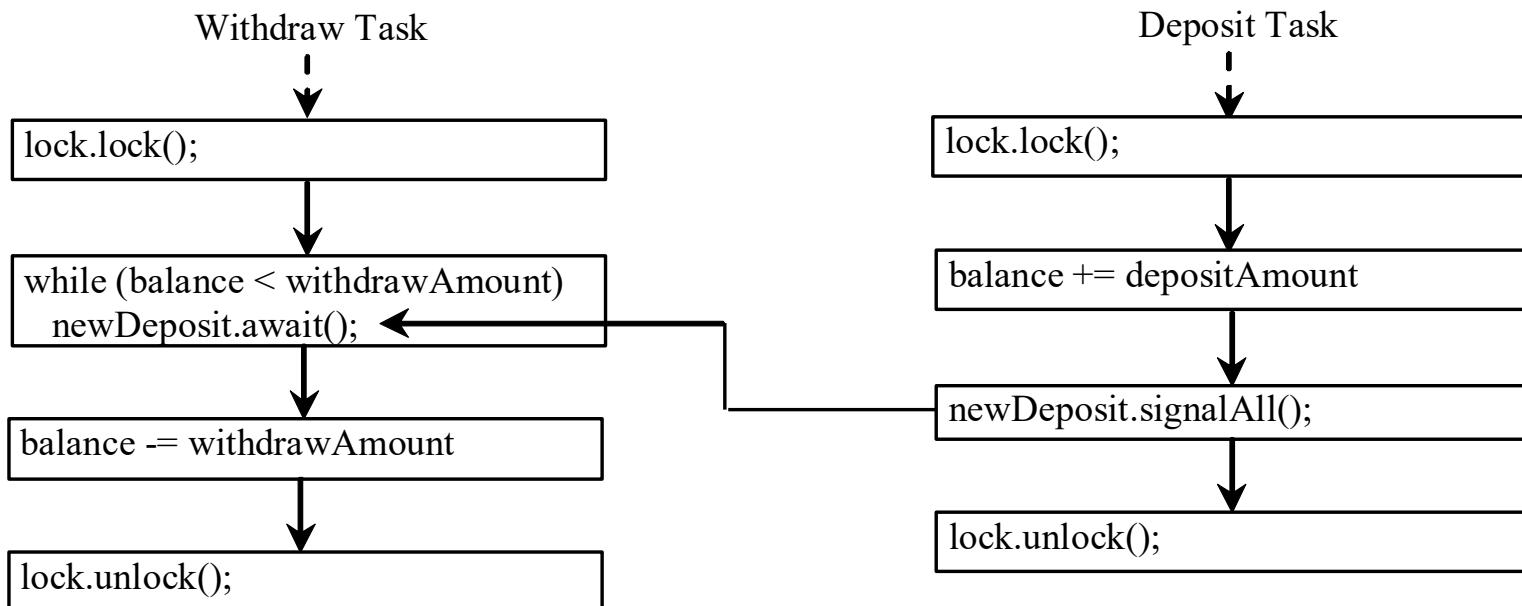
Cooperation/Communication Among Threads

- Conditions can be used for communication among threads.
- A thread can specify what to do under a certain condition.
- newCondition() method of Lock object.
- Condition methods:
 - await() current thread waits until the condition is signaled
 - signal() wakes up a waiting thread
 - signalAll() wakes all waiting threads

«interface»	
<i>java.util.concurrent.Condition</i>	
+ <i>await()</i> : void	Causes the current thread to wait until the condition is signaled.
+ <i>signal()</i> : void	Wakes up one waiting thread.
+ <i>signalAll()</i> : Condition	Wakes up all waiting threads.

Cooperation Among Threads

- Lock with a condition to synchronize operations: newDeposit
 - If the balance is less than the amount to be withdrawn, the withdraw task will wait for the newDeposit condition.
 - When the deposit task adds money to the account, the task signals the waiting withdraw task to try again.
-
- Interaction between the two tasks:



Monitor objects

- A *monitor* is an object with mutual exclusion and synchronization capabilities.
- Only one thread can execute a method at a time in the monitor.
- A thread enters the monitor by acquiring a lock (synchronized keyword on method / block) on the monitor and exits by releasing the lock.
- A thread can wait in a monitor if the condition is not right for it to continue executing in the monitor.
- *Any object can be a monitor.* An object becomes a monitor once a thread locks it.

`wait()`, `notify()`, and `notifyAll()`

- Use the `wait()`, `notify()`, and `notifyAll()` methods to facilitate communication among threads.
- The `wait()`, `notify()`, and `notifyAll()` methods must be called in a synchronized method or a synchronized block on the calling object of these methods. Otherwise, an `IllegalMonitorStateException` would occur.
- The `wait()` method lets the thread wait until some condition occurs. When it occurs, you can use the `notify()` or `notifyAll()` methods to notify the waiting threads to resume normal execution. The `notifyAll()` method wakes up all waiting threads, while `notify()` picks up only one thread from a waiting queue.

Example: Using Monitor

Task 1

```
synchronized (anObject) {  
    try {  
        // Wait for the condition to become true  
        while (!condition)  
            anObject.wait();  
  
        // Do something when condition is true  
    }  
    catch (InterruptedException ex) {  
        ex.printStackTrace();  
    }  
}
```

Task 2

```
synchronized (anObject) {  
    // When condition becomes true  
    anObject.notify(); or anObject.notifyAll();  
    ...  
}
```

Interrupting threads

- `Interrupt()`

If a thread is currently in the Ready or Running state, its interrupted flag is set; if a thread is currently blocked, it is awakened and enters the Ready state, and an `java.io.InterruptedIOException` is thrown.

- The `isInterrupt()` method tests whether the thread is interrupted.
- Can be used to gracefully stop a waiting thread

```
class TestInterruptingThread1 extends Thread{
    public void run(){
        try{
            Thread.sleep(1000);
            System.out.println("task");
        }catch(InterruptedException e){
            throw new RuntimeException("Thread interrupted..."+e);
        }
    }

    public static void main(String args[]){
        TestInterruptingThread1 t1=new TestInterruptingThread1();
        t1.start();
        try{
            t1.interrupt();
        }catch(Exception e){System.out.println("Exception handled "+e);}
    }
}
```

The Static sleep(milliseconds) Method

- The sleep(long mills) method puts the thread to sleep for the specified time in milliseconds

```
public void run() {  
    for (int i = 1; i <= lastNum; i++) {  
        System.out.print(" " + i);  
        try {  
            if (i >= 50) Thread.sleep(1);  
        }  
        catch (InterruptedException ex) {  
        }  
    }  
}
```

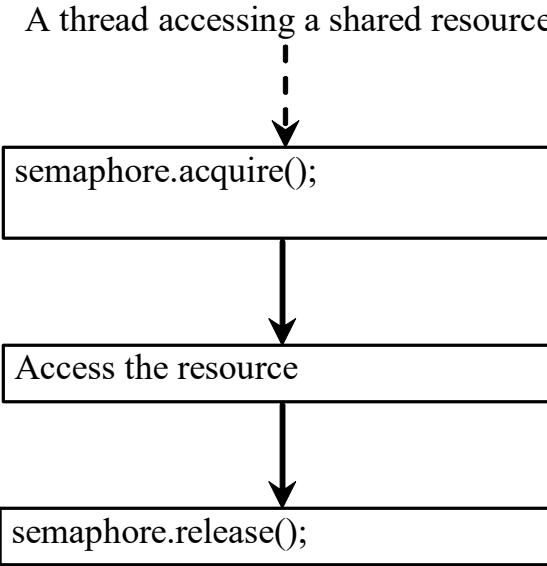
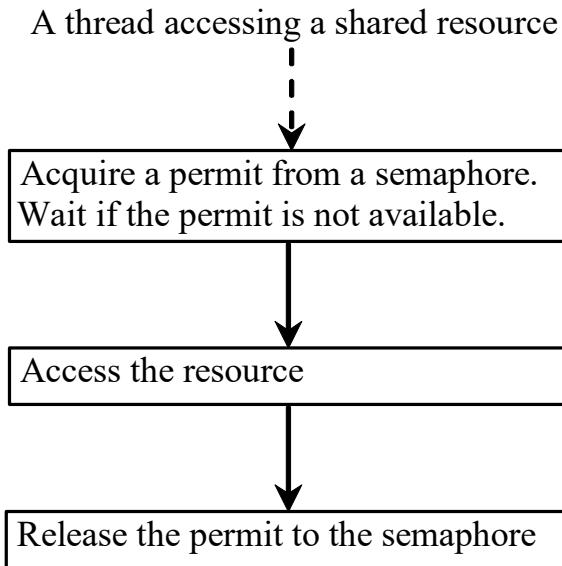
- Every time a number (≥ 50) is printed, the print100 thread is put to sleep for 1 millisecond.

More Advanced Synchronization

- A semaphore object
 - Allows simultaneous access by N threads
 - If $N==1$, then this is known as a mutex (mutual exclusion)
 - Java has a class Semaphore

Semaphores

- Semaphores can be used to restrict the number of threads that access a shared resource.
- There is no notion of ‘ownership’ in Semaphores (unlike ‘locks’)



Creating Semaphores

- To create a semaphore, you have to specify the number of permits with an optional fairness policy
- Once a permit is acquired, the total number of available permits in a semaphore is reduced by 1.
- Once a permit is released, the total number of available permits in a semaphore is increased by 1.

java.util.concurrent.Semaphore
+Semaphore(numberOfPermits: int)
+Semaphore(numberOfPermits: int, fair: boolean)
+acquire(): void
+release(): void

Creates a semaphore with the specified number of permits. The fairness policy is false.

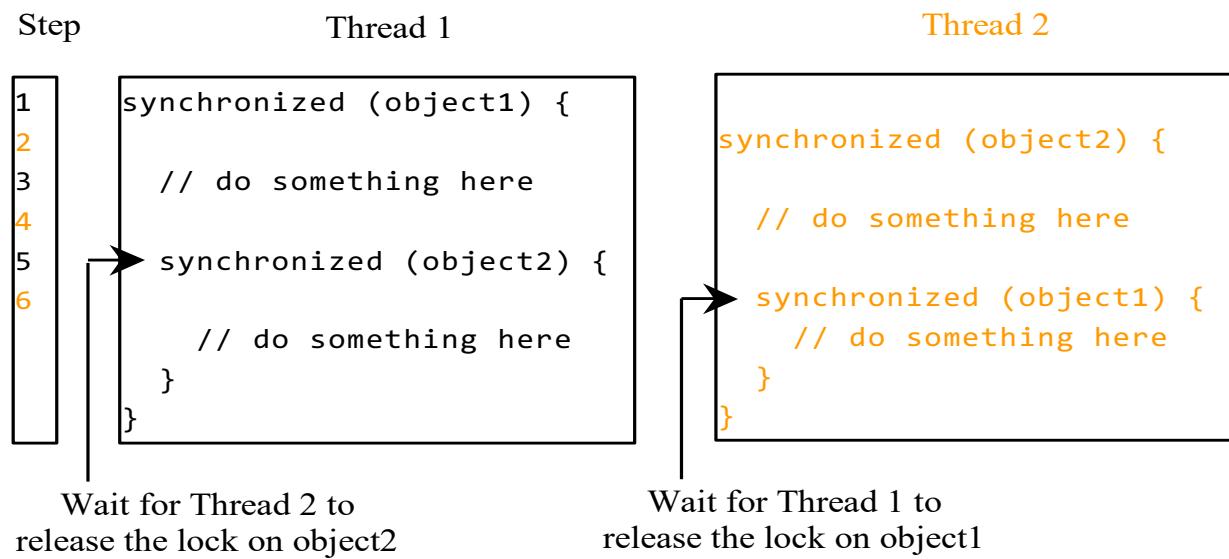
Creates a semaphore with the specified number of permits and the fairness policy.

Acquires a permit from this semaphore. If no permit is available, the thread is blocked until one is available.

Releases a permit back to the semaphore.

Deadlock

- Sometimes two or more threads need to acquire the locks on several shared objects.
- This could cause *deadlock*, in which each thread has the lock on one of the objects and is waiting for the lock on the other object.
- In the figure below, the two threads wait for each other to release the in order to get a lock, and neither can continue to run.



Preventing Deadlocks

- Deadlock can be easily avoided by resource ordering.
- With this technique, assign an order on all the objects whose locks must be acquired and ensure that the locks are acquired in that order.
- How does this prevent deadlock in the previous example?

Summary

- Volatile variables are used to avoid synchronization issues due to caching of variables
- Monitors can be used for inter-thread communication
- Interrupts are used to gracefully stop threads
- Sleep() operation is used to suspend a thread for a specified time
- Semaphores control the access to a shared object
- Ordering of locks can avoid deadlocks

Lecture 4 – RPC/RMI

Topics

- Introduction to Distributed Objects and Remote Invocation
- Remote Procedure Call
- Java RMI

Two main ways to do DC (apart from socket programming)

- Remote Method Invocation (RMI)
 - ❖ Local object invokes methods of an object residing on a remote computer
 - ❖ Invocation as if it was a local method call
- Event-based Distributed Programming
 - ❖ Objects receive asynchronous notifications of events happening on remote computers/processes

Remote Procedure Call (RPC)

- Objects that can receive remote method invocations are called remote objects and they implement a remote interface.
- Programming models for distributed applications are:
 - Remote Procedure Call (RPC)
 - ❖ Client calls a procedure implemented and executing on a remote computer
 - ❖ Call as if it was a local procedure

RPC Interfaces

■ Interfaces for RPC

- An explicit interface is defined for each module.
- An Interface hides all implementation details.
- Accesses the variables in a module can only occur through methods specified in interface.

Remote Procedure Call (RPC)

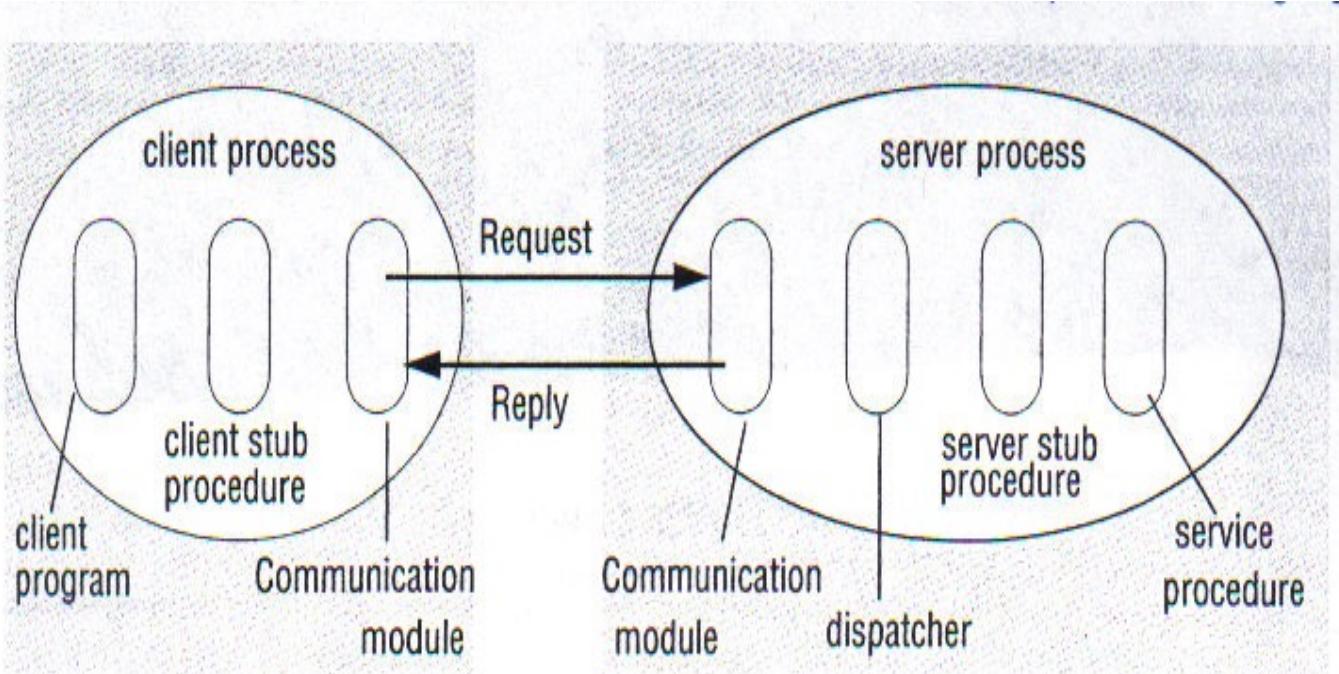


Figure 3. Role of client and server stub procedures in RPC in the context of a procedural language

SUN ONC RPC

- RPC only addresses procedure calls.
- RPC is not concerned with objects and object references.
- A client that accesses a server includes one **stub procedure** for each procedure in the service interface.
- A client stub procedure is similar to a proxy method of RMI (discussed later).
- A server stub procedure is similar to a skeleton method of RMI (discussed later).

Strength and Weaknesses of RPC

- RPC (or even RMI) is not well suited for adhoc query processing. (e.g. SQL queries)
- It is not suited for transaction processing without special modification.
- A separate special mode of querying is proposed –
- Remote Data Access (RDA).
- RDA is specially suited for DBMS.
- In a general client_server environment both RPC and RDA are needed.



RMI

Java Remote Method Invocation

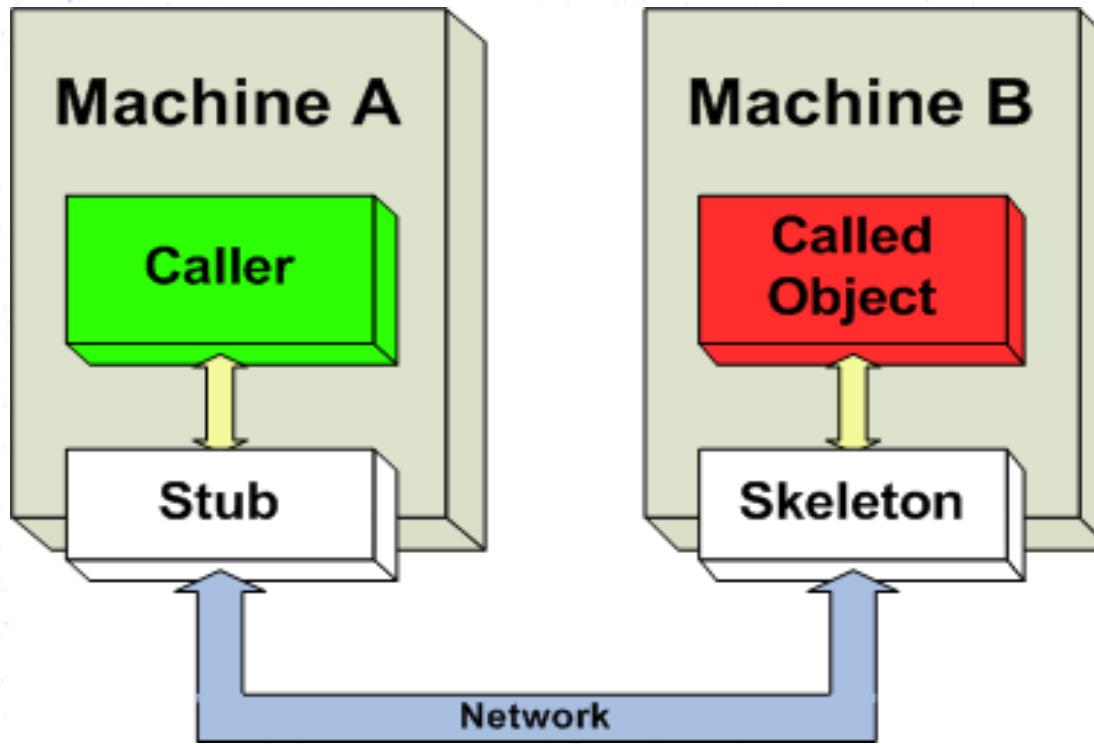


Fig: Distributed Object Technology

Java Remote Method Invocation

- RMI Server, client, interface, stubs, skeletons
- RMI Registry
- Objects + RPC = RMI
- Method Invocation between different JVMs
- Java RMI API
- JRMP (Java Remote Method Protocol)
- Java object serialization
- Parameter Marshalling

RMI System Architecture

Lets divide into two perspectives:

- Layered Structure
- Working Principles

RMI Layered Structure

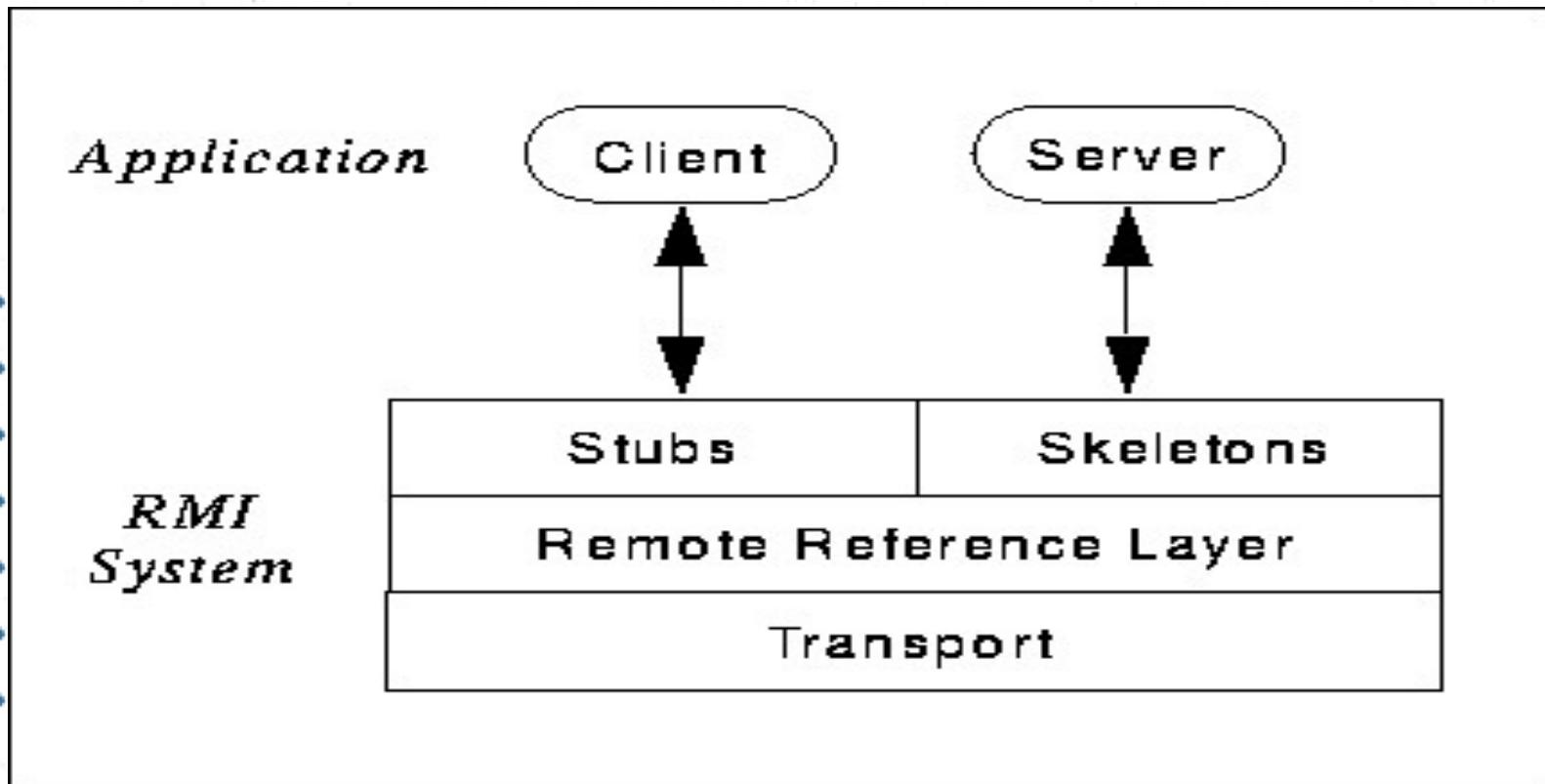


Fig: RMI Layered Structure

RMI Layered Structure

- Application layer: Server, Client
- Interface: Client stub, Server skeleton
- Remote Reference layer: RMI registry
- Transport layer: TCP/IP

RMI Working Principles

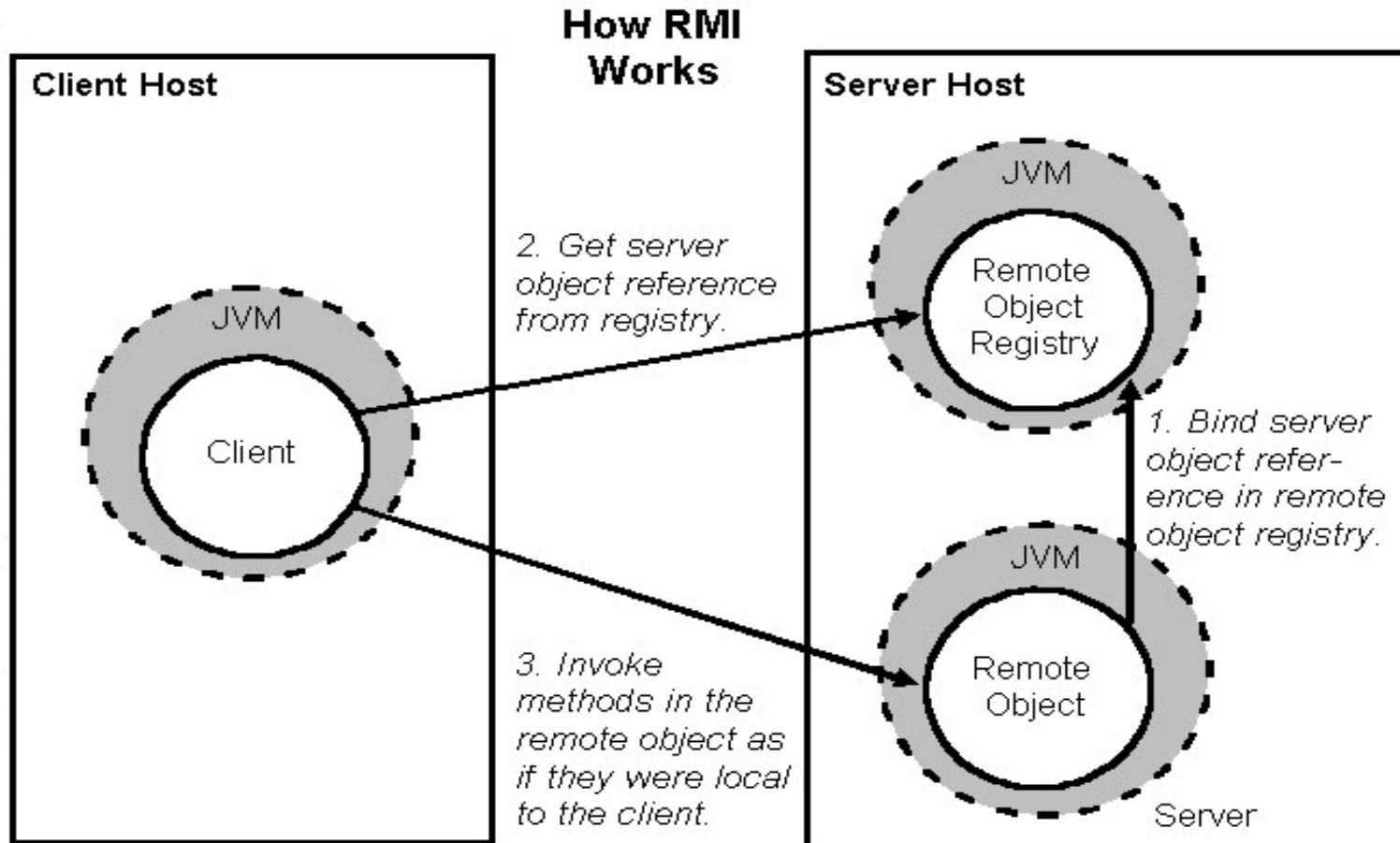
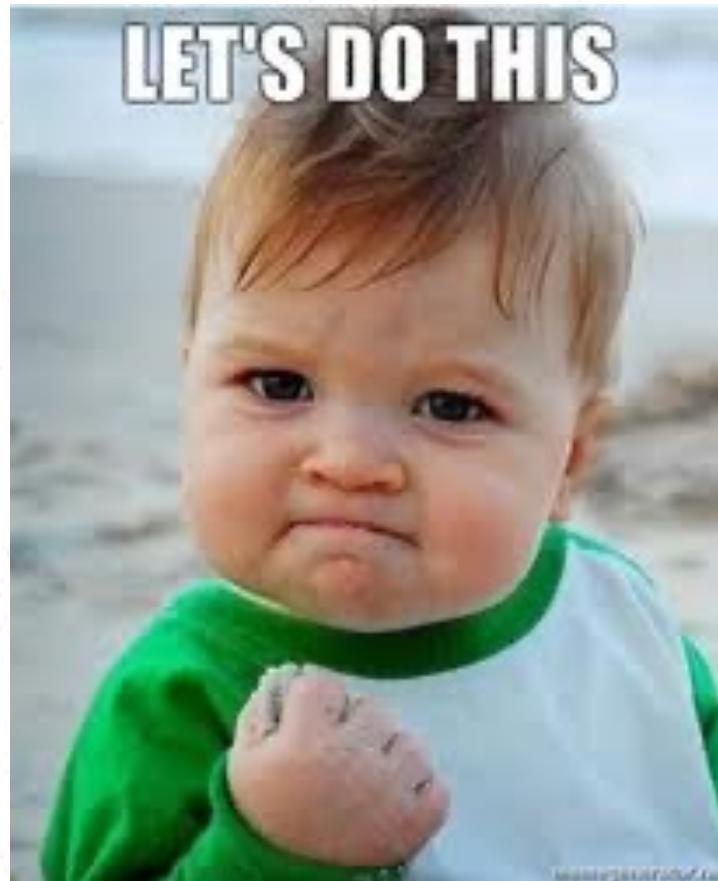
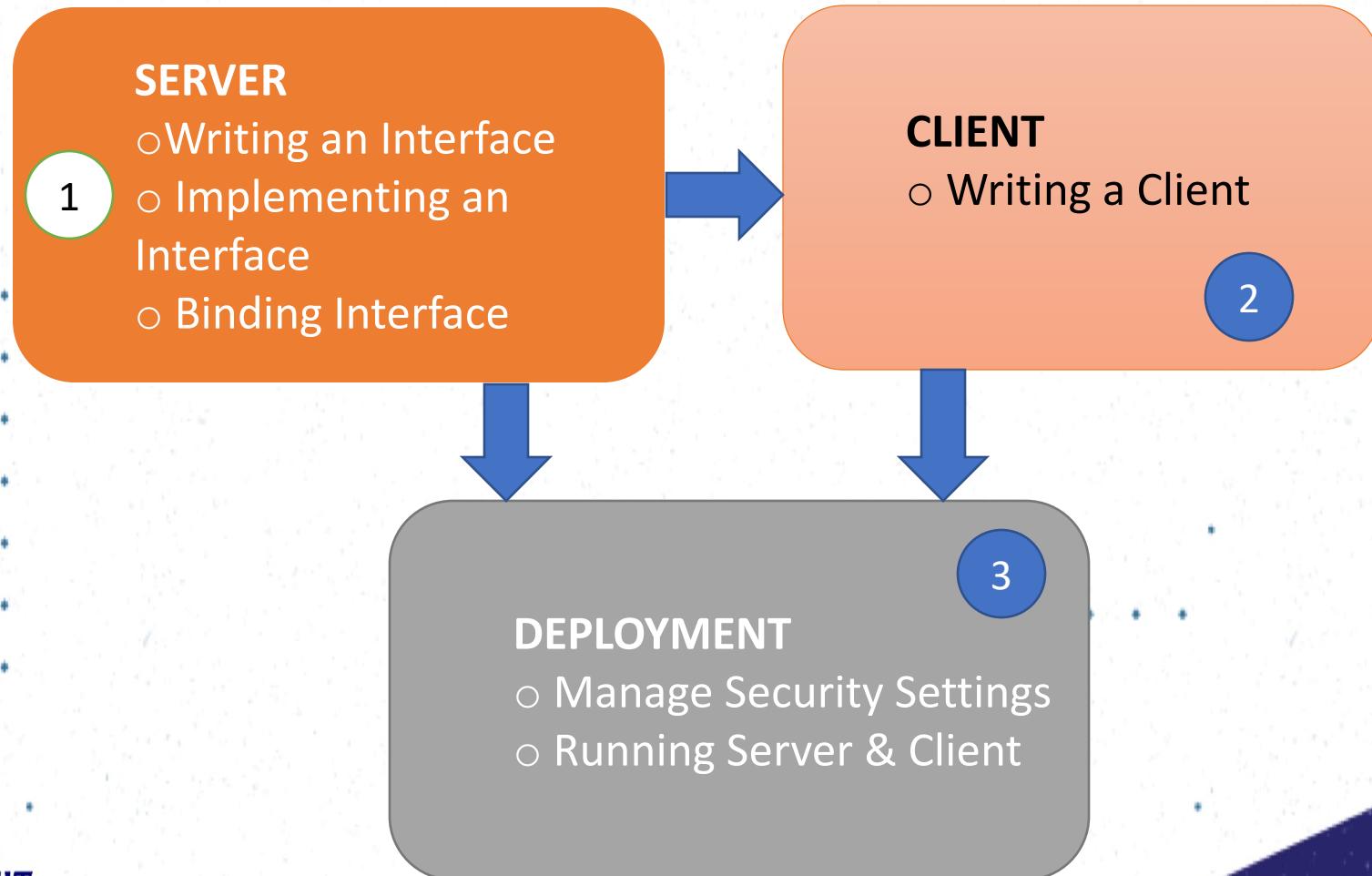


Fig: RMI Working principles

Ready to Develop One?



A Simple RMI Application



Service Interface: An Agreement Between Server & Client

- Factorial Operation

```
public long factorial(int number) throws  
    RemoteException;
```

- Check Prime Operation

```
public boolean checkPrime(int number) throws  
    RemoteException;
```

```
public BigInteger square(int number) throws  
    RemoteException;
```

Server Application: Writing a Service Interface

```
//interface between RMI client and server

import java.math.BigInteger;
import java.rmi.*;

public interface MathService extends Remote {
    // every method associated with RemoteException
    // calculates factorial of a number
    public long factorial(int number) throws
        RemoteException;
    // check if a number is prime or not
    public boolean checkPrime(int number) throws
        RemoteException;
    //calculate the square of a number and returns
    // BigInteger
    public BigInteger square(int number) throws
        RemoteException;
}
```

Server Application: Implementing the Service Interface

```
//MathService Server or Provider

import java.awt.font.NumericShaper;
import java.math.BigInteger;
import java.rmi.*;
import java.rmi.registry.LocateRegistry;
import java.rmi.server.UnicastRemoteObject;

public class MathServiceProvider extends UnicastRemoteObject implements
    MathService {

    // MathServiceProvider implements all the methods of MathService interface
    // service constructor
    public MathServiceProvider() throws RemoteException {
        super();
    }

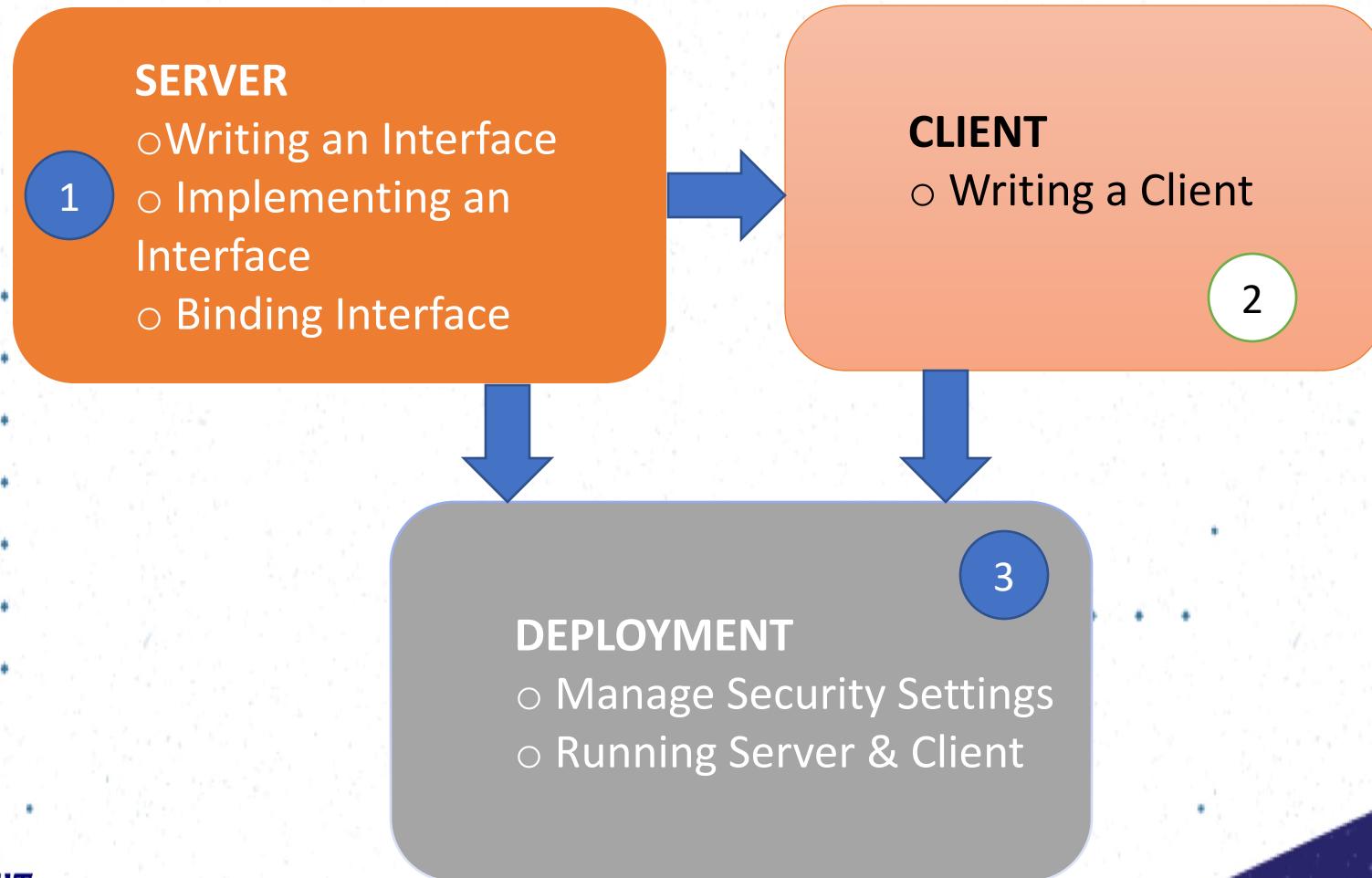
    // implementation of factorial
    public long factorial(int number) {
        // returning factorial
        if (number == 1)
            return 1;
        return number * factorial(number - 1);
    }
}
```

Server Application: Instantiating & Binding the Service

```
public static void main(String args[]) {
    try {
        // setting RMI security manager
        if (System.getSecurityManager() == null) {
            System.setSecurityManager(new
                RMISecurityManager());
        }

        * * *
        // creating server instance
        MathServiceProvider provider = new
            MathServiceProvider();
        // binding the service with the registry
        LocateRegistry.getRegistry().bind(
            "MathService", provider);
        System.out.println("Service is bound to RMI
            registry");
    } catch (Exception exc) {
        // showing exception
        System.out.println("Cant bind the service:
            " + exc.getMessage());
        exc.printStackTrace();
    }
}
```

A Simple RMI Application



Client Application: Service Lookup

```
// Assign security manager
if (System.getSecurityManager() == null) {
    System.setSecurityManager(new
        RMISecurityManager());
}

// Accessing RMI registry for MathService
String hostName = "localhost"; //this can
    be any host
MathService service = (MathService) Naming.
    lookup("//" + hostName
    + "/MathService");
    * * * *
```

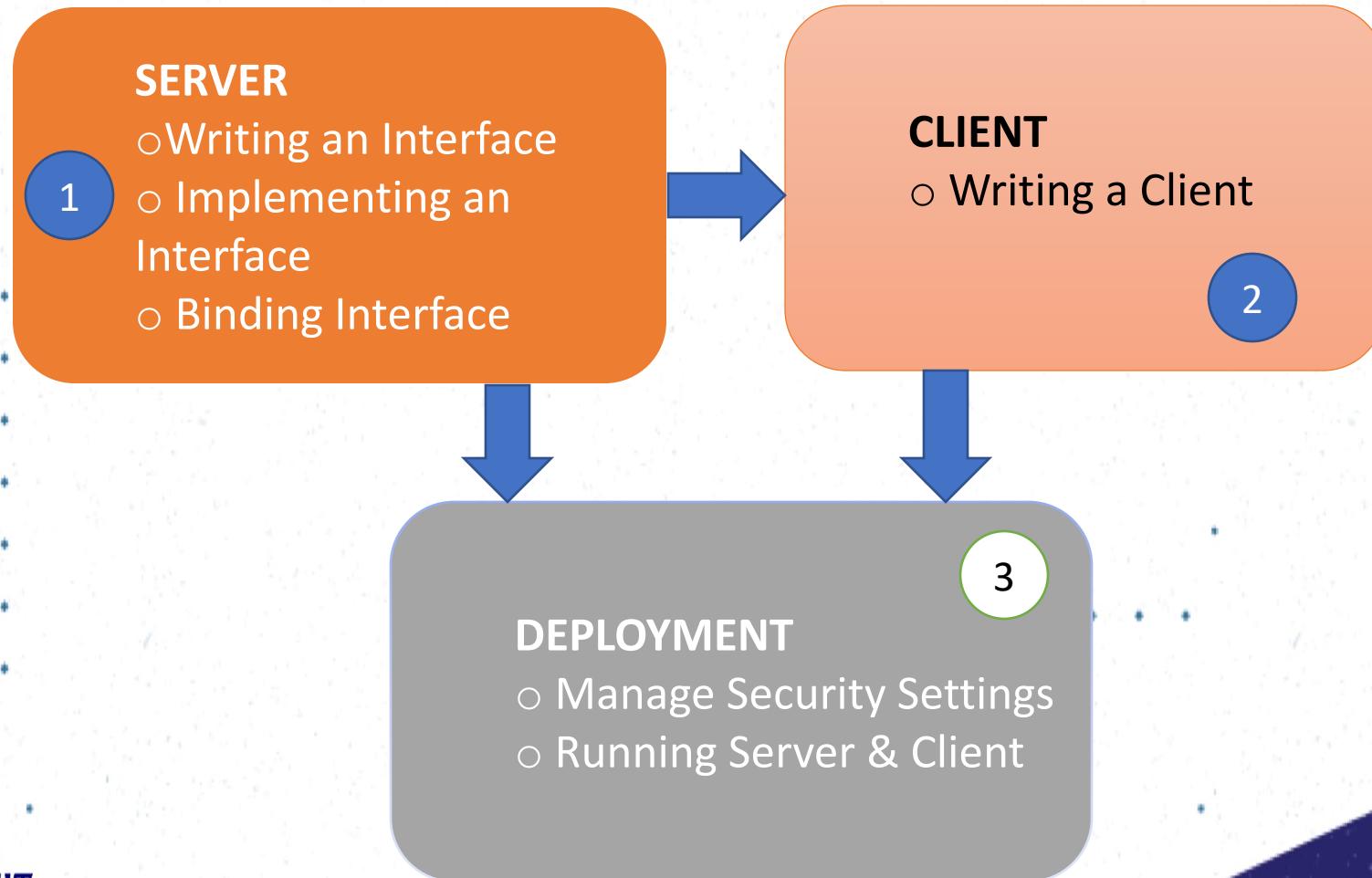
Fig: Client locating *MathService* service

Client Application: Accessing Service

```
// Call to factorial method  
System.out.println("The factorial of " + number +  
    "=" + service.factorial(number));  
  
// Call to checkPrime method  
boolean isprime=service.checkPrime(number);  
  
//Call to square method  
BigInteger squareObj=service.square(number);
```

Fig: Client accessing *MathService* service

A Simple RMI Application



Server Deployment: Start RMI Registry

- To start RMI registry on windows

```
$ start rmiregistry
```

- To start RMI registry on Linux

```
$ rmiregistry &
```

Server Deployment: Compile the Server

- Compile both MathService interface and MathServiceProvider class

```
$ javac MathService.java MathServiceProvider.  
      java
```

Security Deployment: Create Security Policy file (Both Client & Server)

- Create a security policy file called *no.policy* with the following content and add it to CLASSPATH
- This step implies for both server and client

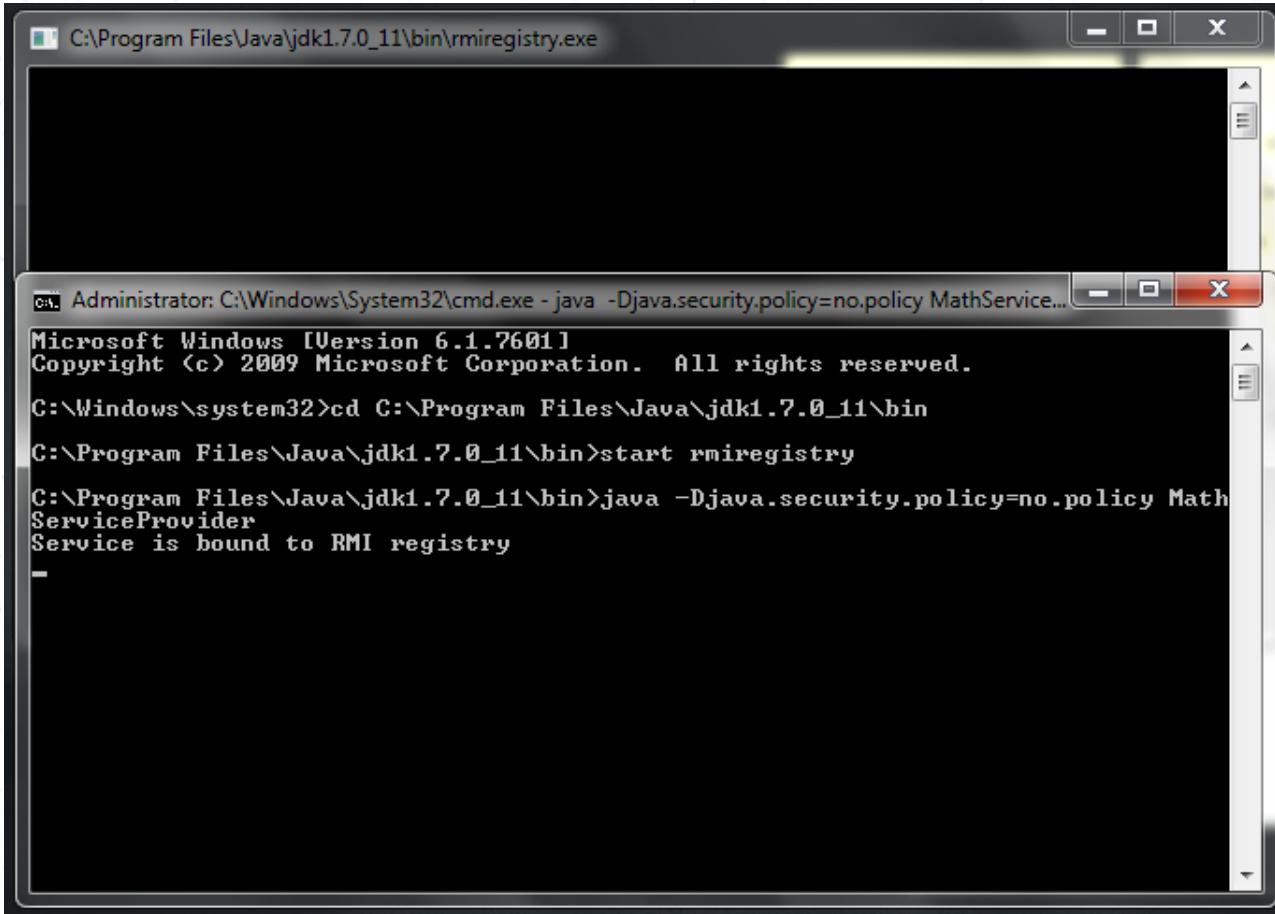
```
grant {  
    permission java.security.AllPermission;  
};
```

Start the Server

- Execute the command to run server

```
$ java -Djava.security.policy=no.policy  
MathServiceProvider
```

Server Running



The screenshot shows a Windows command prompt window titled "C:\Program Files\Java\jdk1.7.0_11\bin\rmiregistry.exe". The window title bar also includes the path "C:\Program Files\Java\jdk1.7.0_11\bin\rmiregistry.exe" and the text "Administrator: C:\Windows\System32\cmd.exe - java -Djava.security.policy=no.policy MathService...". The command line shows the following steps:

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Windows\system32>cd C:\Program Files\Java\jdk1.7.0_11\bin
C:\Program Files\Java\jdk1.7.0_11\bin>start rmiregistry
C:\Program Files\Java\jdk1.7.0_11\bin>java -Djava.security.policy=no.policy MathServiceProvider
Service is bound to RMI registry
```

Start the Client

- Execute the command to run client

```
$ java -Djava.security.policy=no.policy  
MathServiceClient localhost
```

Client Interface

```
C:\Windows\system32\cmd.exe
C:\Program Files\Java\jdk1.7.0_11\bin>java -Djava.security.policy=no.policy Math
ServiceClient localhost
Enter your visiting card information
Enter your first name:
Masud
Enter your last name:
Rahman
Enter your roll number:
11130260
Enter your mobile number
01913213887
First name:Masud
Is the card signed? true
Enter your number to get factorial
10
The factorial of 10=3628800
Enter your number to check prime
13
13 is a prime number
Enter your number to get square
24
The square of 24 is =576
C:\Program Files\Java\jdk1.7.0_11\bin>
```

Java RMI notes

- Java Object Serialization
- Parameter Marshalling & Unmarshalling
- Object Activation
 - Singleton
 - Per client
 - Per call

Strength Of Java RMI

- *Object Oriented:* Can pass complex object rather than only primitive types
- *Mobile Behavior:* Change of roles between client and server easily
- *Design Patterns:* Encourages OO design patterns as objects are transferred
- *Safe & Secure:* The security settings of Java framework used
- *Easy to Write /Easy to Use:* Requires very little coding to access service

Strengths Of Java RMI

- *Connects to Legacy Systems:* JNI & JDBC facilitate access.
- *Write Once, Run Anywhere:* 100% portable, run on any machine having JVM
- *Distributed Garbage Collection:* Same principle like memory garbage collection
- *Parallel Computing:* Through multi-threading RMI server can serve numerous clients
- *Interoperable between different Java versions:* Available from JDK 1.1, can communicate between all versions of JDKs

Weaknesses of Java RMI

- *Tied to Java System*: Purely Java-centric technology, does not have good support for legacy system written in C, C++, Ada etc.
- *Performance Issue* : Only good for large-grain computation
- *Security Restrictions & Complexities*: Threats during downloading objects from server, malicious client request, added security complexity in policy file.

.NET Remoting

- .NET equivalent of Java RMI
- Uses Windows registry as the registry service
- Java RMI supports more communication protocols and .NET remoting
- .NET remoting creates proxies at runtime similar to RMI

Conclusion

- RMI/RPC makes it easier to build distributed systems with programmers not having to worry about network comm. (sockets, etc)
- .NET Remoting
- No interoperability (Java only due to binary messages used)



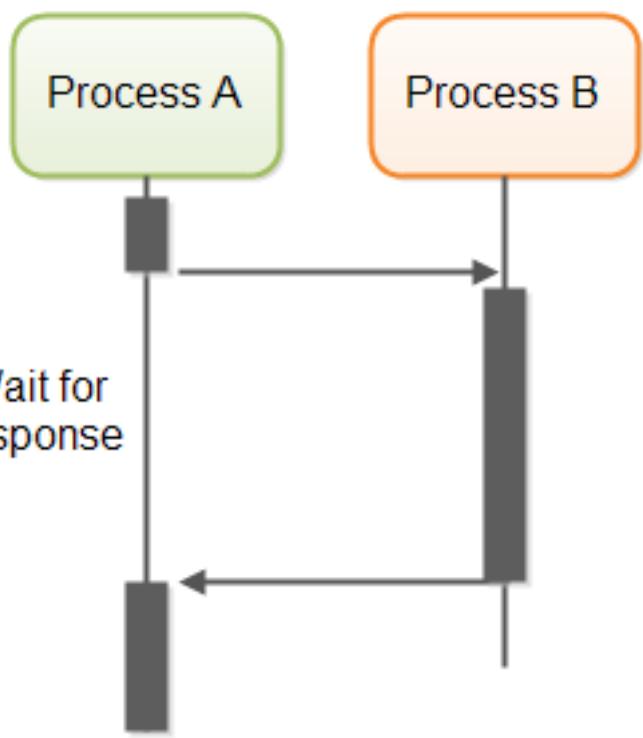
Lecture 5

Asynchronous Communication

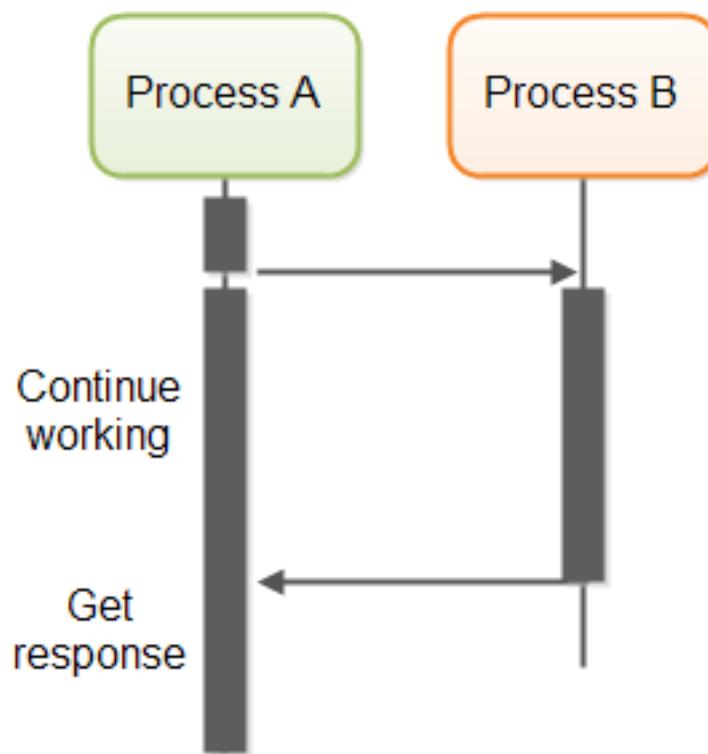
This Week

- The interactivity of an application that invokes slow methods at the remote end.
- We will see how this issue can be addressed using asynchronous calls.
- We will look at two main approaches of making asynchronous calls
 - Remote Callback functions
 - Asynchronous Messaging

Synchronous



Asynchronous



Blocking Calls and Distributed Computing

- When a function is called, the caller typically must wait (block) until the called function completes & returns
- In a distributed computing system, blocking for a remote call can easily be a waste of resources
 - Especially if it is a call that could take a while
 - i.e. Client waits while server performs a long job
 - Not utilising resources properly/efficiently there!

Synchronous vs. Asynchronous

- Synchronous invocation = blocking call
 - Serial processing
 - Control is passed to called function
 - Caller cannot continue until called function returns
- Asynchronous invocation = non-blocking call
 - Parallel processing (or at least one way to implement it)
 - Control is returned immediately to the caller
 - Called function carries on in the background
 - At some later time, caller retrieves function's return value

Local asynchronous Calls

- Reasons for using asynchronous calls
 - Maintain GUI responsiveness
 - Utilise resources of caller more efficiently
(e.g. continue doing other work during a long-running call)
 - Java Swing event dispatching

Distributed Asynchronous calls

- In the client server model, the server is passive: the IPC is initiated by the client;
- Some applications require the server to initiate communication upon certain events.
 - monitoring
 - games
 - auctioning
 - voting/polling
 - chat-room
 - message/bulletin board
 - groupware

When to use Asynchronous Calls?

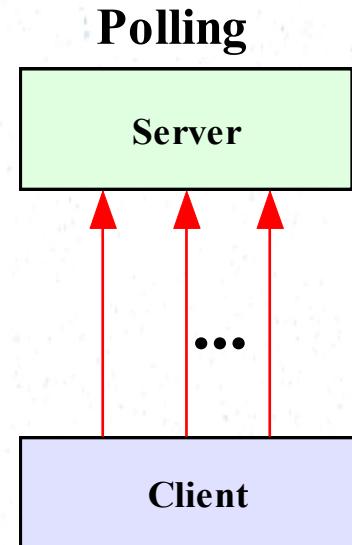
- Every RPC call is potentially long-running
 - Network/server failures are only detected after timeouts expire
 - Making every RPC call asynchronous increases code complexity, just on the *chance* a network failure occurs
- So use asynchronous calls only on functions that are expected to take a long time
 - Heavy processing tasks, intensive disk I/O tasks, etc.
 - For GUI clients, responsiveness is also a key issue

Remote asynchronous communication

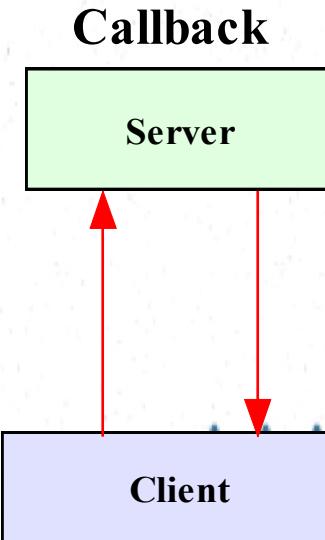
- Remote Callback functions
 - Messaging (e.g. JMS, Microsoft Messaging Queuing)
 - Both Java and .NET supports callback functions

Polling vs. Callback

In the absence of callback, a client will have to poll a passive server repeatedly if it needs to be notified that an event has occurred at the server end.



A client issues a request to the server repeatedly until the desired response is obtained.



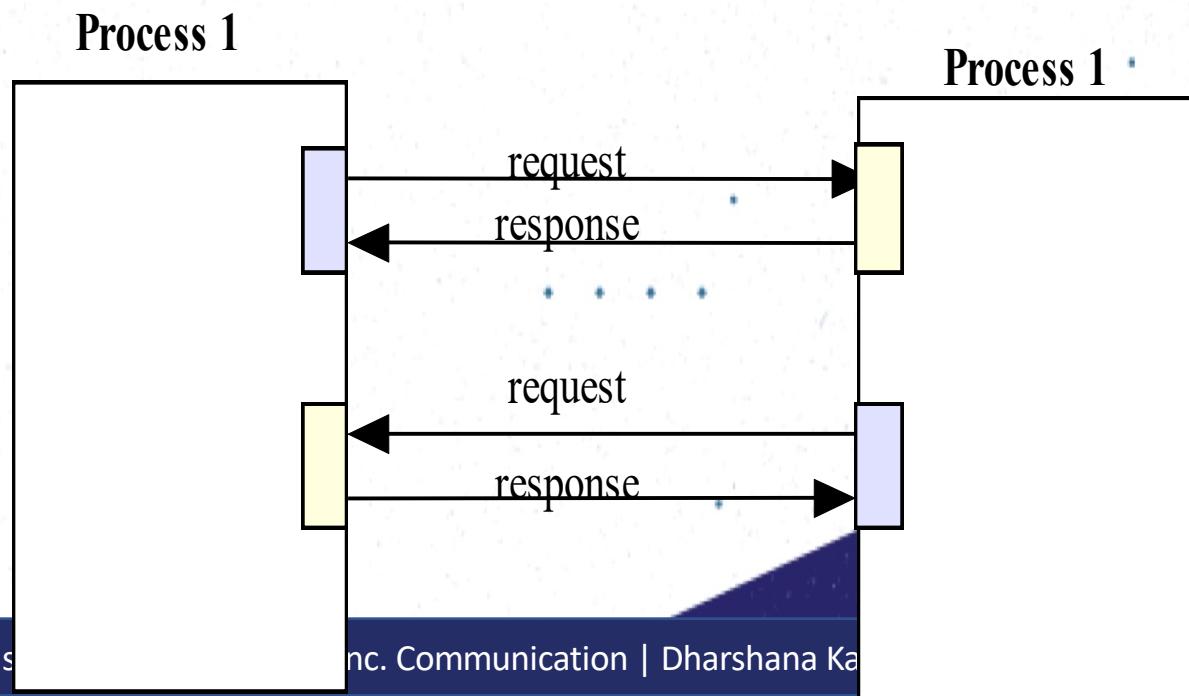
A client registers itself with the server, and wait until the server calls back.

Polling vs. Callback

- Blocking is like making a call and waiting for the other party to respond (if the other party is busy with some other call)
- Polling is like repeatedly making a telephone call and check whether the other party is available.
- Callback is like making a call and leaving a message to other party to call back with certain information

Two-way communications

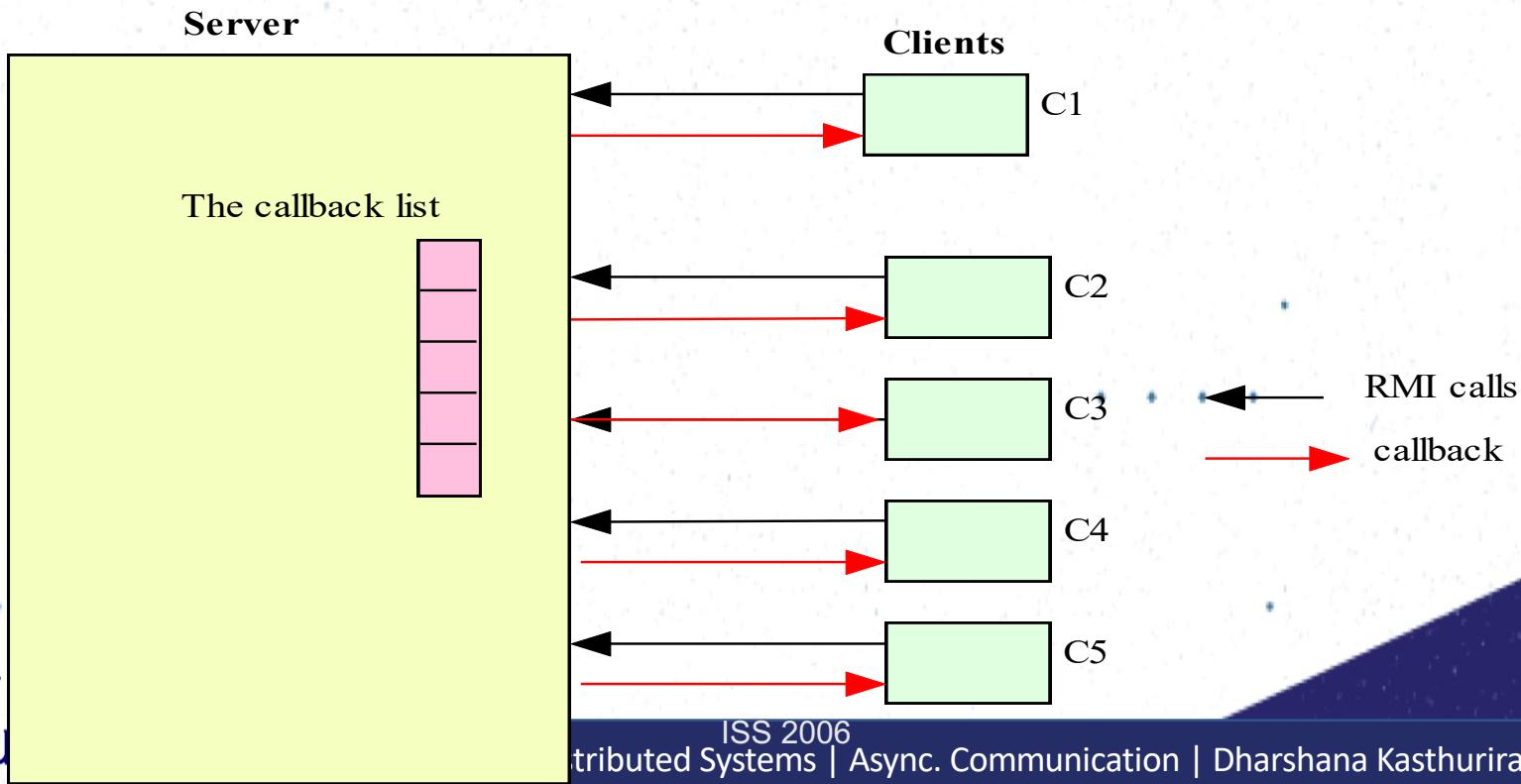
- Some applications require that both sides may initiate IPC.
- Using sockets, duplex communication can be achieved by using two sockets on either side.
- With connection-oriented sockets, each side acts as both a client and a server.



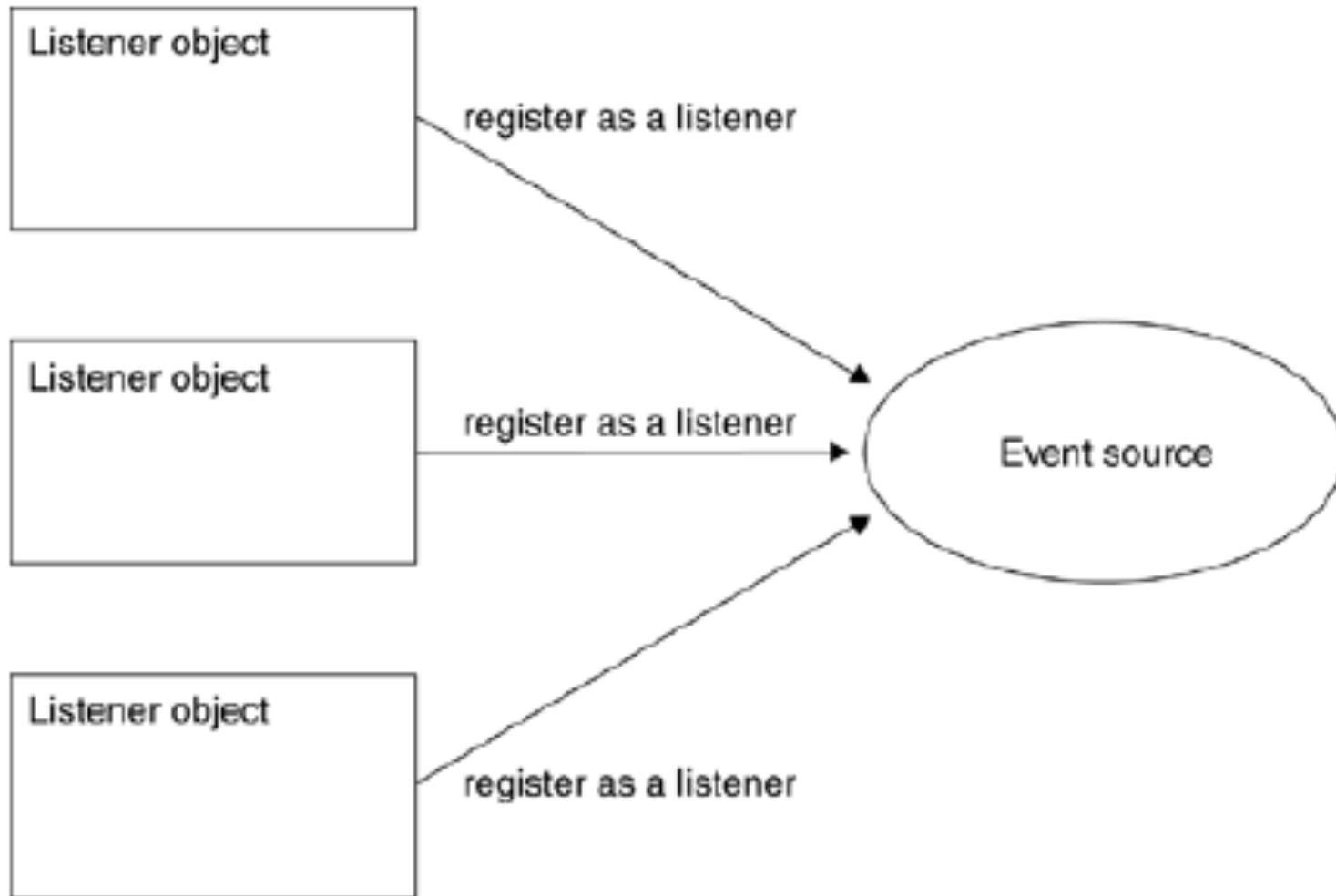
RMI Callbacks

RMI Callbacks

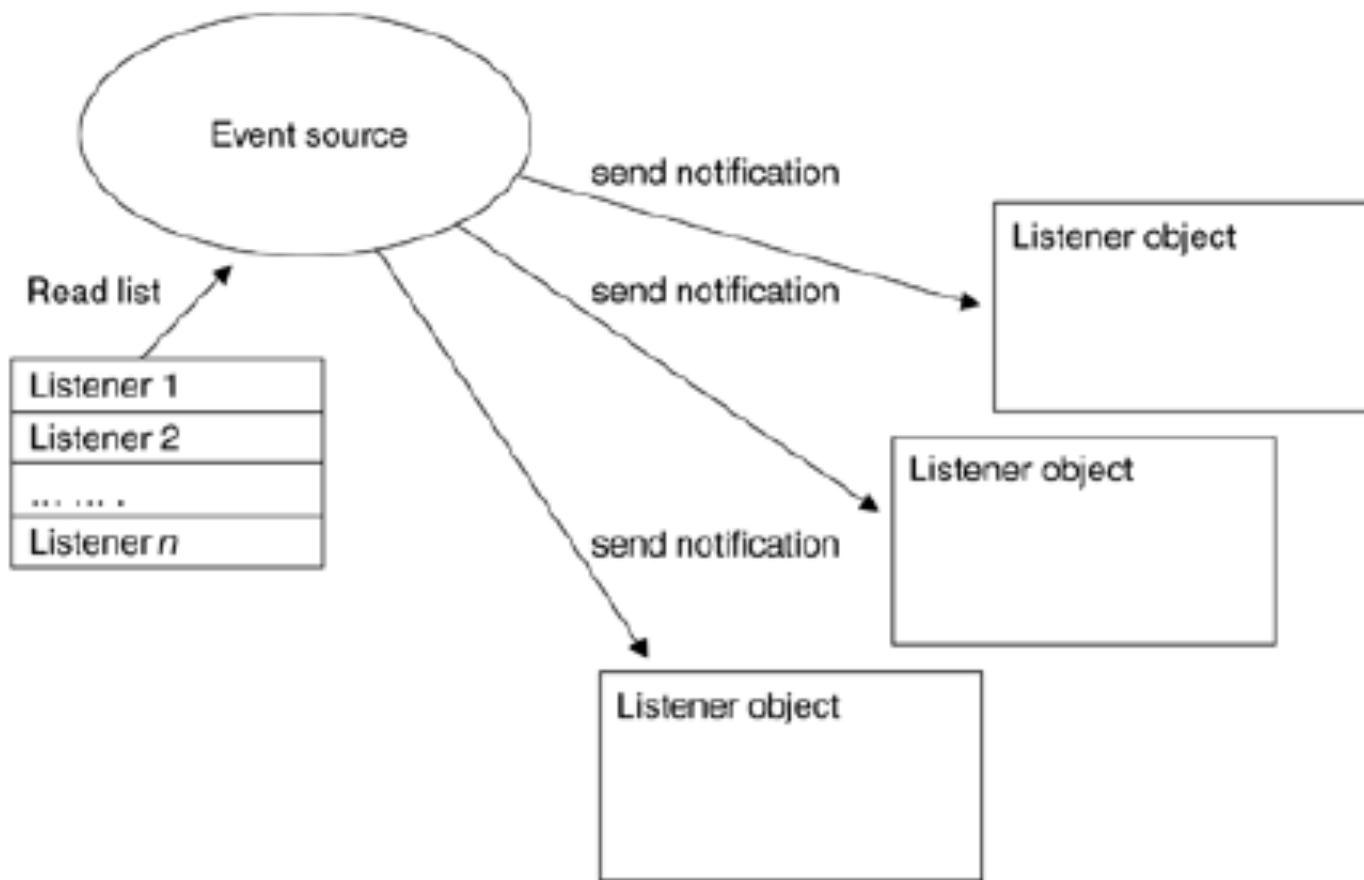
- A callback client registers itself with an RMI server.
- The server makes a callback to each registered client upon the occurrence of a certain event.



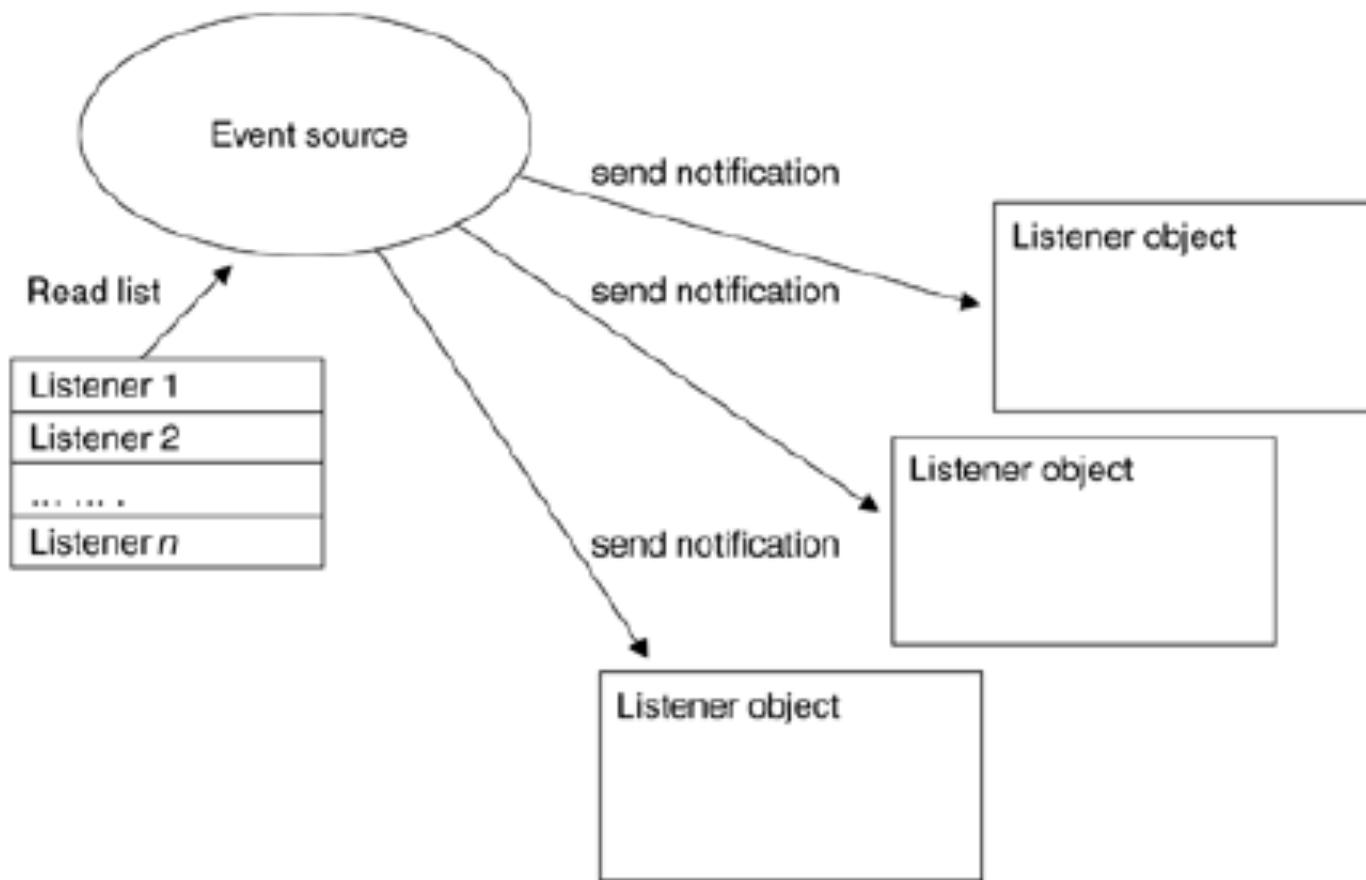
Multiple listeners



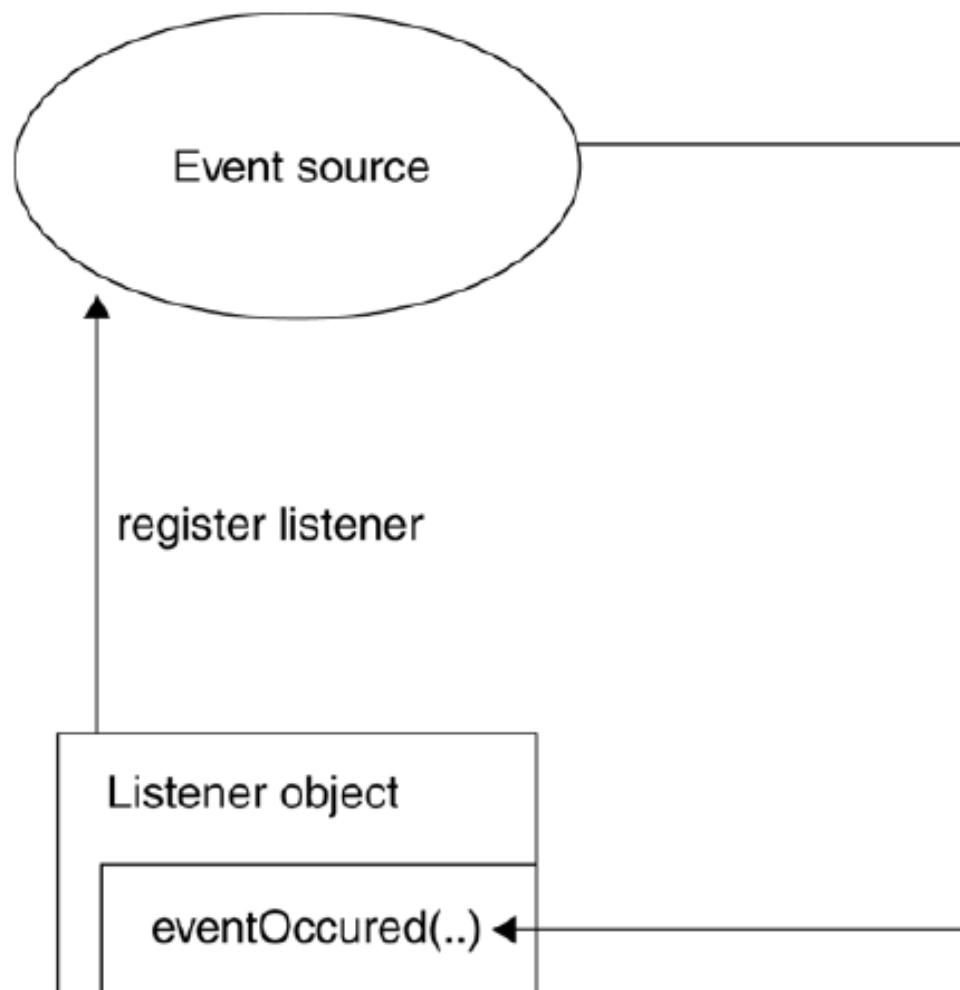
Callback notification to every registered listener



Callback notification to every registered listener



Callback implemented by invoking a method on a listening object



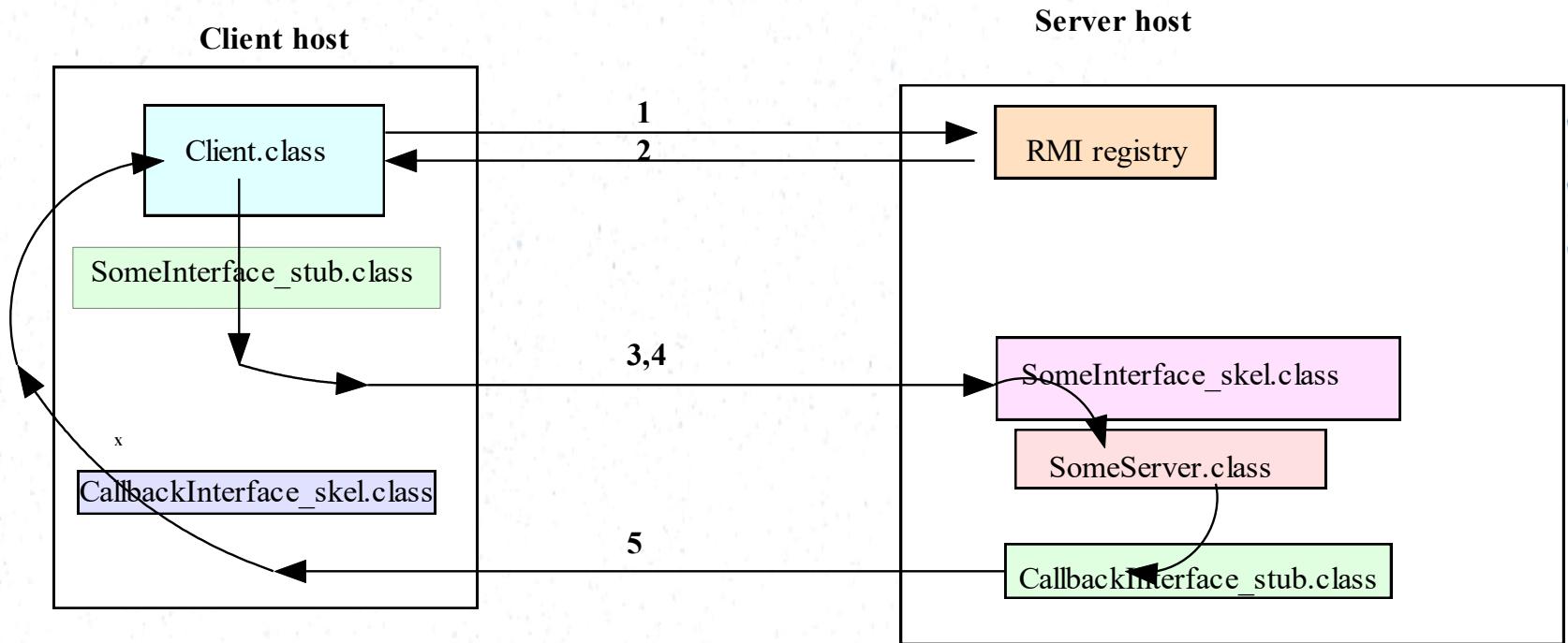
Client callback

- To provide client callback, the client-side software
 - supplies a remote interface,
 - instantiate an object which implements the interface,
 - passes a reference to the object to the server via a remote method call to the server.

Client callback

- The remote server:
 - collects these client references in a data structure.
 - when the awaited event occurs, the remote server invokes the callback method (defined in the client remote interface) to pass data to the client.
- Two sets of stub-skeletons are needed: one for the server remote interface, the other one for the client remote interface.

Callback Client-Server Interactions



1. Client looks up the interface object in the RMI registry on the server host.
2. The RMI Registry returns a remote reference to the interface object.
3. Via the server stub, the client process invokes a remote method to register itself for callback, passing a remote reference to itself to the server. The server saves the reference in its callback list.
4. Via the server stub, the client process interacts with the skeleton of the interface object to access the methods in the interface object.
5. When the anticipated event takes place, the server makes a callback to each registered client via the callback interface stub on the server side and the callback interface skeleton on the client side.

RMI Callback Example

- Temperature monitoring system
- Server will sense the temperature of the environment
- The Server will notify the client listeners of the changes in temperature
- Polling is not efficient thus we can use callback as a means asynchronous notifications
- In addition to the normal RMI classes/interfaces there will be a client interface defined as well (so that the server can ‘call back’)

RMI Callback Example

- Server Interface (same as in blocking RMI)

```
interface TemperatureSensor extends  
java.rmi.Remote  
{  
    public double getTemperature() throws  
        java.rmi.RemoteException;  
    public void addTemperatureListener  
        (TemperatureListener listener )  
        throws java.rmi.RemoteException;  
    public void removeTemperatureListener  
        (TemperatureListener listener )  
        throws java.rmi.RemoteException;
```

RMI Callback Example

- Listener Interface (Client side)

```
interface TemperatureListener extends  
java.rmi.Remote  
{  
    public void temperatureChanged(double  
temperature)  
        throws java.rmi.RemoteException;  
}
```

- Defines the callback method

RMI Callback Example

- Server Implementation

```
public class TemperatureSensorServer extends  
UnicastRemoteObject implements TemperatureSensor,  
Runnable{  
  
    public void addTemperatureListener ( TemperatureListener  
    listener ) throws java.rmi.RemoteException{  
        list.add (listener);  
    }  
    public void run(){  
        for (;;) {  
            if(checkTempChanged()){  
                // Notify registered listeners  
                notifyListeners();  
            }  
        }  
    }  
}
```

RMI Callback Example

```
private void notifyListeners(){
for (Enumeration e = list.elements(); e.hasMoreElements(); ){
    TemperatureListener listener = (TemperatureListener)
        e.nextElement();
    listener.temperatureChanged (temp);
    list.remove( listener );
}
}

public static void main(String args[]){
    TemperatureSensorServer sensor = new
    TemperatureSensorServer();
    String registration = "rmi://" + registry
    +"/TemperatureSensor";
    Naming.rebind( registration, sensor );
    Thread thread = new Thread (sensor);
    thread.start();
}
```

RMI Callback Example

Client implementation

```
public class TemperatureMonitor extends UnicastRemoteObject
    implements TemperatureListener{

    public static void main(String args[]){
        Remote remoteService = Naming.lookup ( registration );
        TemperatureSensor sensor = (TemperatureSensor)remoteService;
        double reading = sensor.getTemperature();
        System.out.println ("Original temp : " + reading);
        TemperatureMonitor monitor = new TemperatureMonitor();
        sensor.addTemperatureListener(monitor);
    }

    public void temperatureChanged(double temperature)
        throws java.rmi.RemoteException
    {
        System.out.println ("Temperature change event : " +
            temperature);
    }
}
```

Running the example

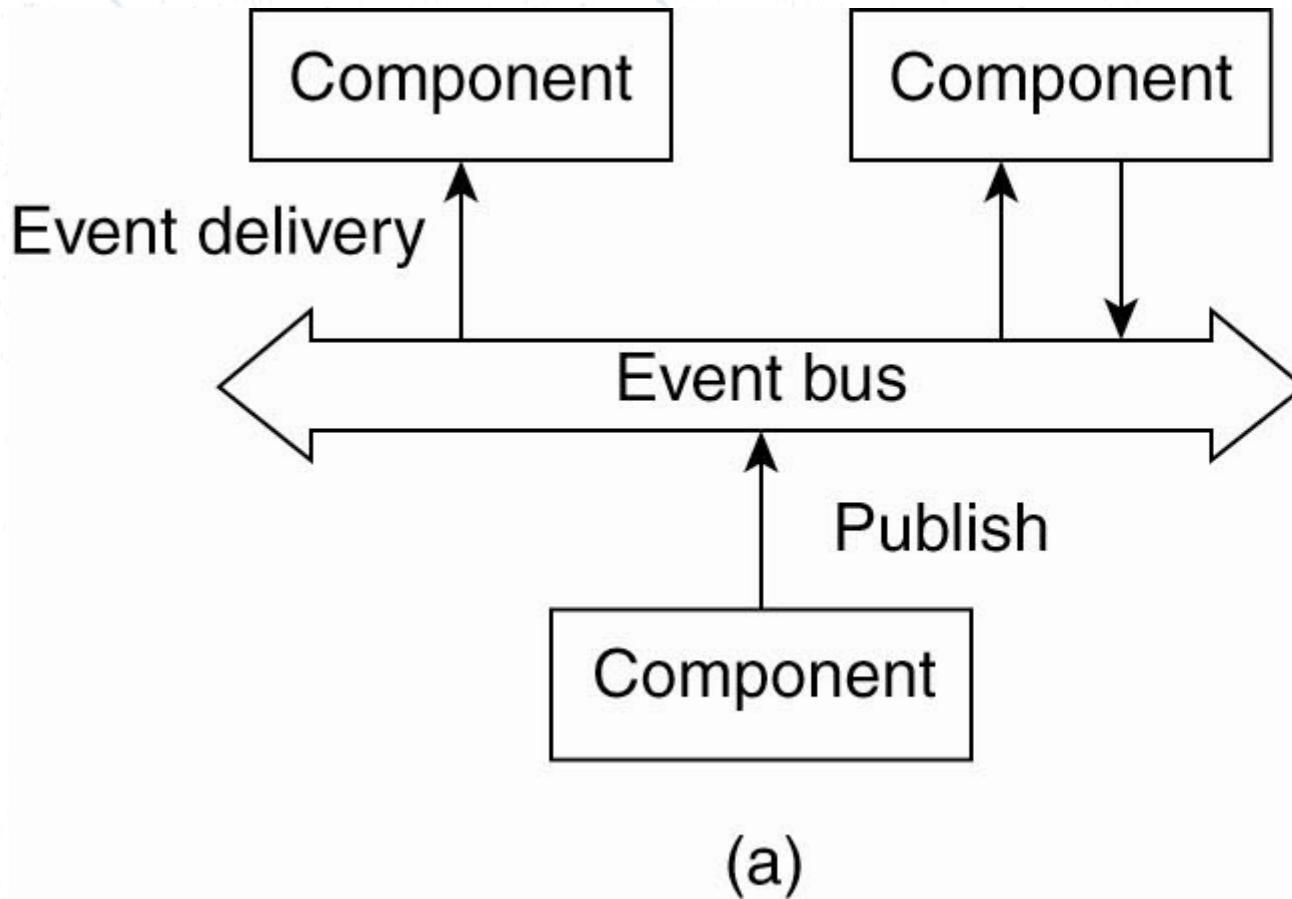
1. Compile the applications and generate stub/skeleton files for both
2. TemperatureSensorServer and TemperatureSensorMonitor.
3. Run the rmiregistry application.
4. Run the TemperatureSensorServer.
5. Run the TemperatureSensorMonitor.

Asynchronous Callback functions and thread safety

- Callback functions use threads in the background
- Main thread does the remote call and then a worker thread calls the callback function
- Main thread is running at the same time
- Have to handle thread safety issues manually

Asynchronous Messaging services

Event based Architectures



Java Message Service (JMS)

- A **specification** that describes a common way for Java programs to create, send, receive and read distributed enterprise messages
- *loosely coupled* communication
- *Asynchronous* messaging
- *Reliable* delivery
 - A message is guaranteed to be delivered once and only once.
- Outside the specification
 - Security services
 - Management services

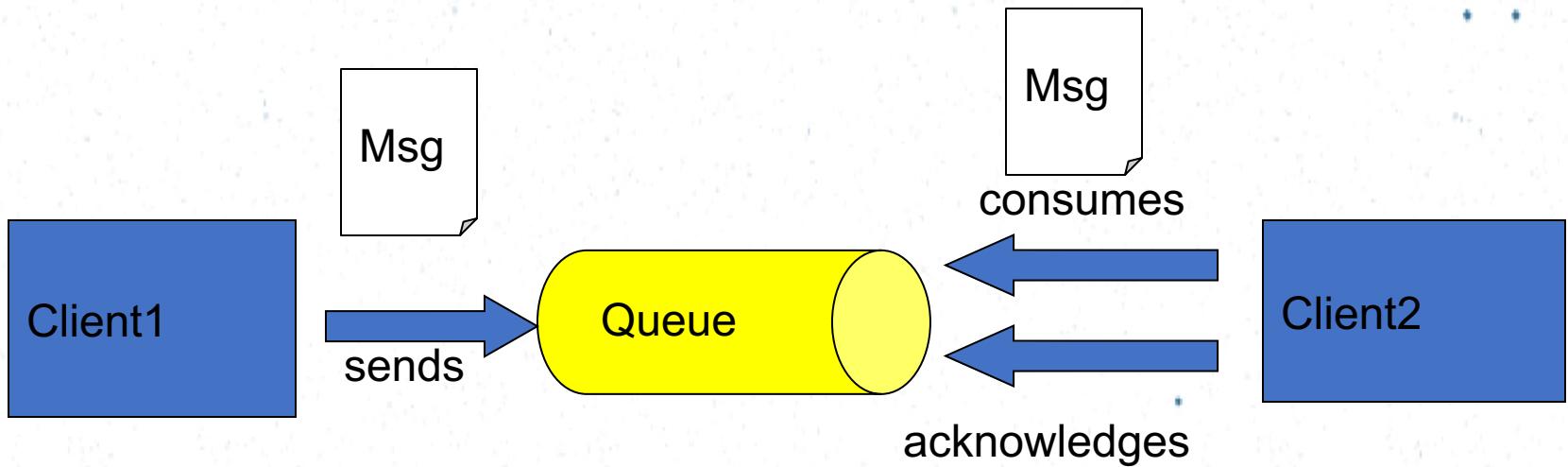
A JMS Application

- JMS Clients
 - Java programs that send/receive messages
- Messages
- Administered Objects
 - preconfigured JMS objects created by an admin for the use of clients
 - ConnectionFactory, Destination (queue or topic)
- JMS Provider
 - messaging system that implements JMS and administrative functionality

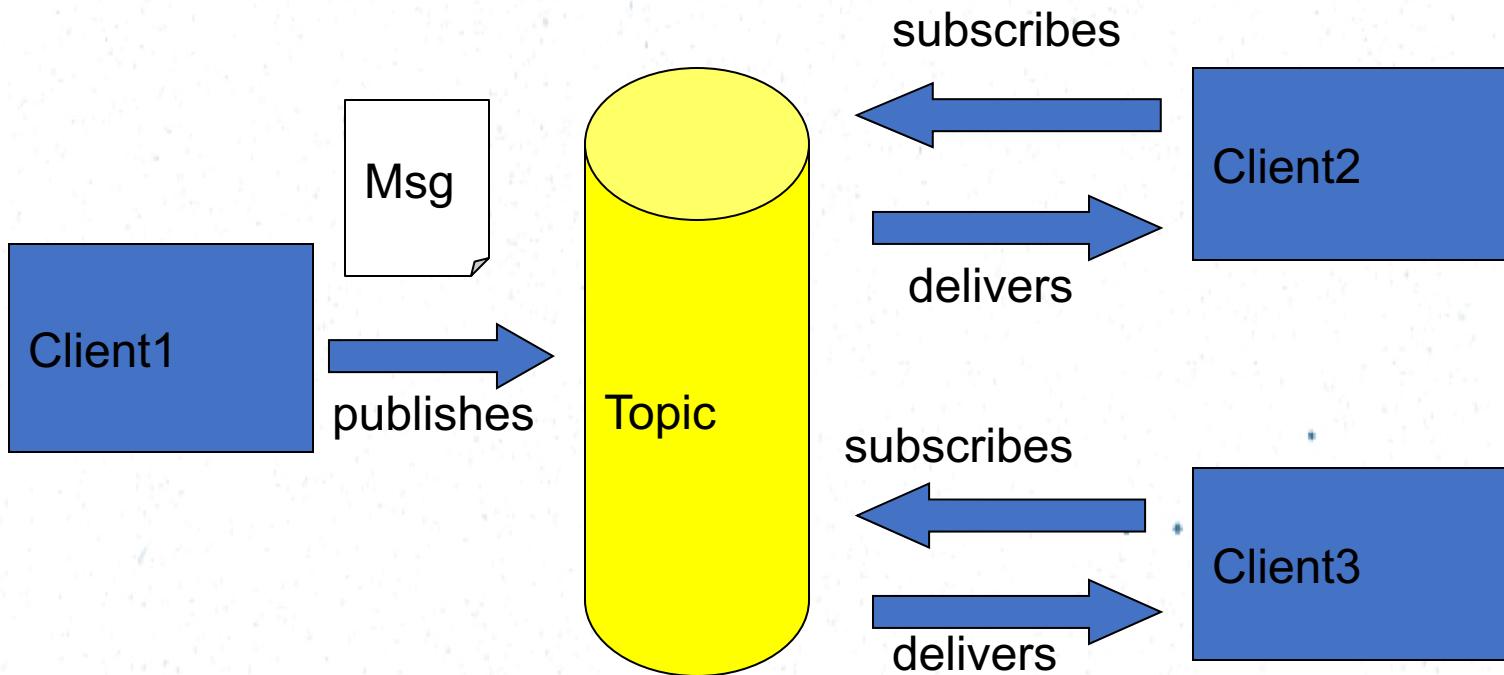
JMS Messaging Domains

- Point-to-Point (PTP)
 - built around the concept of message queues
 - each message has only one consumer
- Publish-Subscribe systems
 - uses a “topic” to send and receive messages
 - each message has multiple consumers

Point-to-Point Messaging



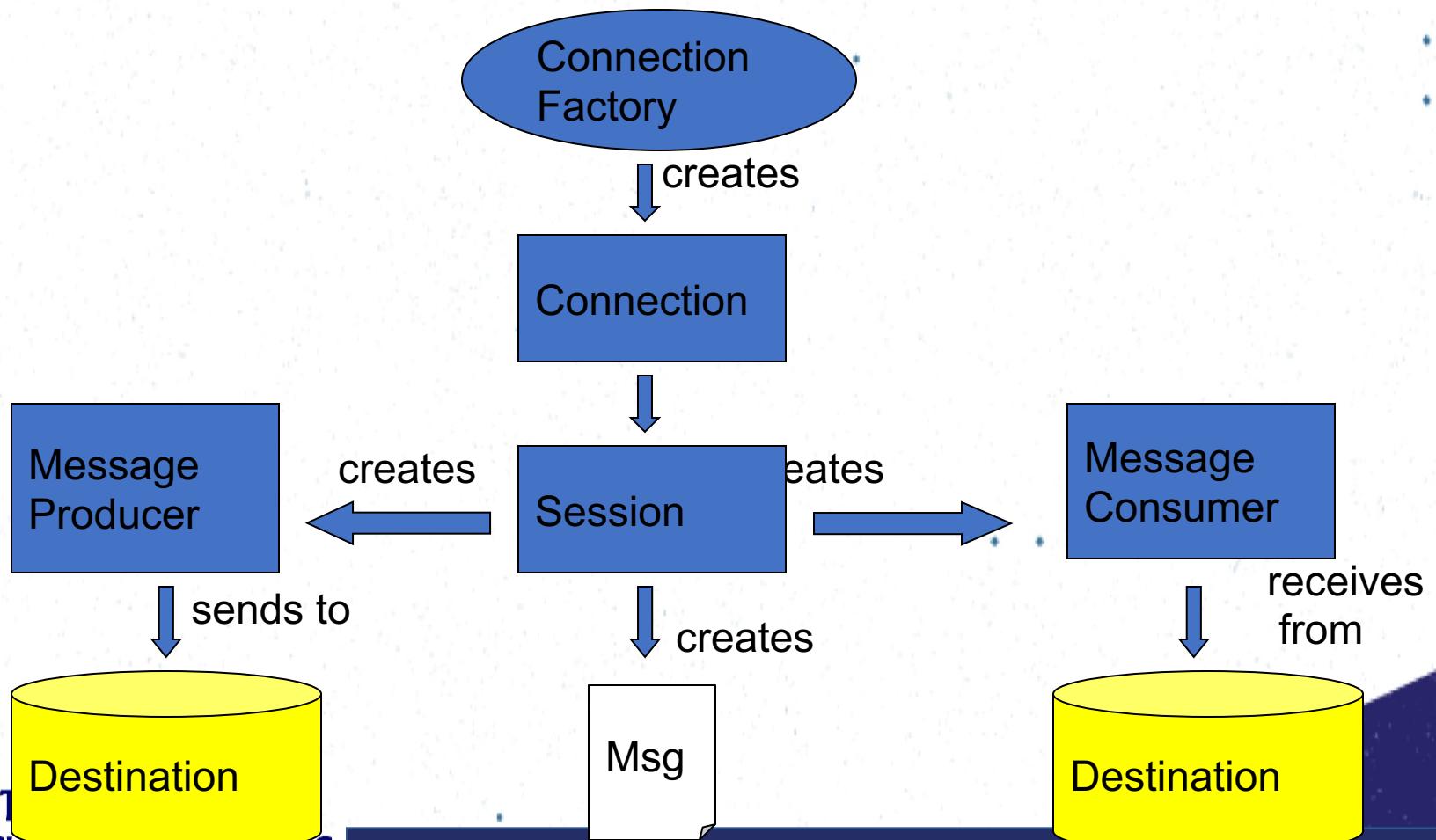
Publish/Subscribe Messaging



Message Consumptions

- Synchronously
 - A subscriber or a receiver explicitly fetches the message from the destination by calling the receive method.
 - The receive method can *block* until a message arrives or can time out if a message does not arrive within a specified time limit.
- Asynchronously
 - A client can register a *message listener* with a consumer.
 - Whenever a message arrives at the destination, the JMS provider delivers the message by calling the listener's `onMessage()` method.

JMS API Programming Model



JMS Client Example

- Setting up a connection and creating a session

```
InitialContext jndiContext=new InitialContext();
//look up for the connection factory
ConnectionFactory cf=jndiContext.lookup(connectionfactoryname);
//create a connection
Connection connection=cf.createConnection();
//create a session
Session session=connection.createSession(false,Session.AUTO_ACKNOWLEDGE);
//create a destination object
Destination dest1=(Queue)
    jndiContext.lookup("/jms/myQueue"); //for PointToPoint
Destination dest2=(Topic)jndiContext.lookup("/jms/myTopic");
//for publish-subscribe
```

Producer Sample

- Setup connection and create a session
- Creating producer

```
MessageProducer producer=session.createProducer(dest1);
```

- Send a message

```
Message m=session.createTextMessage();
```

```
m.setText("just another message");
```

```
producer.send(m);
```

- Closing the connection

```
connection.close();
```

Consumer Sample (Synchronous)

- Setup connection and create a session
- Creating consumer

```
MessageConsumer consumer=session.createConsumer(dest1);
```

- Start receiving messages

```
connection.start();
```

```
Message m=consumer.receive();
```

Consumer Sample (Asynchronous)

- Setup the connection, create a session
- Create consumer
- Registering the listener
 - `MessageListener listener=new myListener();`
 - `consumer.setMessageListener(listener);`
- `myListener` should have `onMessage()`

```
public void onMessage(Message msg){  
    //read the message and do computation  
}
```

Listener Example

```
public void onMessage(Message message) {  
    TextMessage msg = null;  
    try {  
        if (message instanceof TextMessage) {  
            msg = (TextMessage) message;  
            System.out.println("Reading message: " + msg.getText());  
        } else {  
            System.out.println("Message of wrong type: " +  
                message.getClass().getName());  
        }  
    } catch (JMSException e) {  
        System.out.println("JMSException in onMessage(): " + e.toString());  
    } catch (Throwable t) {  
        System.out.println("Exception in onMessage(): " + t.getMessage());  
    }  
}
```

JMS Messages

- Message Header
 - used for identifying and routing messages
 - contains vendor-specified values, but could also contain application-specific data
 - typically name/value pairs
- Message Properties (optional)
- Message Body(optional)
 - contains the data
 - five different message body types in the JMS specification

JMS Message Types

Message Type	Contains	Some Methods
TextMessage	String	getText,setText
MapMessage	set of name/value pairs	setString,setDouble,setLong,getDouble(getString)
BytesMessage	stream of uninterpreted bytes	writeBytes,readBytes
StreamMessage	stream of primitive values	writeString,writeDouble,writeLong,readString
ObjectMessage	serialize object	setObject,getObject

More JMS Features

- Durable subscription
 - by default a subscriber gets only messages published on a topic while a subscriber is alive
 - durable subscription retains messages until they are received by a subscriber or expire
- Request/Reply
 - by creating temporary queues and topics
 - Session.createTemporaryQueue()
 - producer=session.createProducer(msg.getJMSReplyTo());
reply= session.createTextMessage("reply");
reply.setJMSCorrelationID(msg.getJMSMessageID());
producer.send(reply);

More JMS Features

- Transacted sessions
 - session=connection.createSession(true,0)
 - combination of queue and topic operation in one transaction is allowed
 - void onMessage(Message m) {
 try { Message m2=processOrder(m);
 publisher.publish(m2); session.commit();
 } catch(Exception e) { session.rollback(); }
}

More JMS Features

- Persistent/nonpersistent delivery
 - `producer.setDeliveryMethod(DeliveryMode.NON_PERSISTENT);`
 - `producer.send(msg, DeliveryMode.NON_PERSISTENT, 3, 1000);`
- Message selectors
 - SQL-like syntax for accessing header:
`subscriber = session.createSubscriber(topic, "priority > 6 AND type = 'alert'");`
 - Point to point: selector determines single recipient
 - Pub-sub: acts as filter

JMS Providers

- SunONE Message Queue (SUN)
- MQ JMS (IBM)
- WebLogic JMS (BEA)
- JMSCourier (Codemesh)
- Apache ActiveMQ

JMS API in a JEE Application

- Since the J2EE1.3 , the JMS API has been an integral part of the platform
- JEE components can use the JMS API to send messages that can be consumed asynchronously by a specialized Enterprise Java Bean
 - message-driven bean

Microsoft Messaging Queue

- .NET Equivalent of JMS

<https://msdn.microsoft.com/en-us/library/ms731089.aspx>

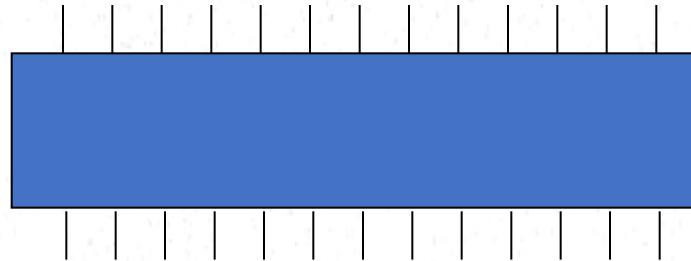
Summary

- Asynchronous Communication helps to make non blocking calls among distributed components
- It maximizes the performance and response time of distributed Systems
- Callback functions and Messaging Services are two common ways of implementing asynchronous communication
- Some calls may have to be synchronous (blocking) if further processing cannot be done without the information in server response

Lecture 6 - Distributed Component frameworks

What is a component?

- Very intuitive, but often vaguely defined
- A semiconductor chip is a component



- The chip vendor publishes manuals that tell developers the functionality of each pin

A0-A16

I/O Address bus

D0-D8

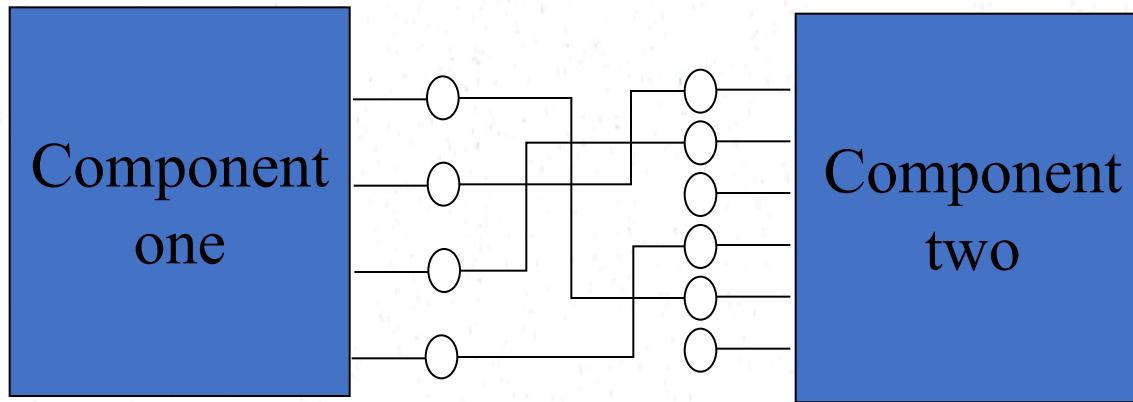
I/O Data bus

ACK

O Acknowledge: Accepted when low

Building complex systems

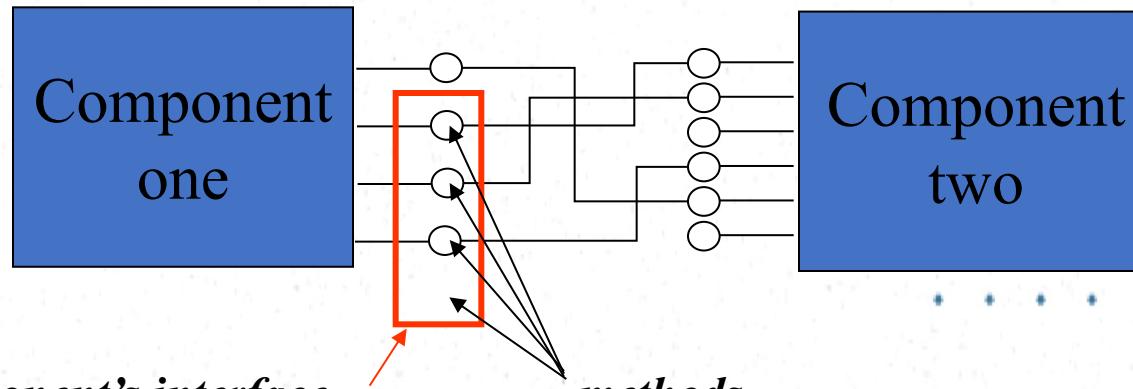
- Hardware system integrators connect pins of chips together according to their functions to build complex electronic devices such as computers.



- Pins functionality defines behavior of the chip.
- Pins functionality is standardized to match functionality of the board (and take advantage of common services).

Software components

- The software component model takes a very similar approach:
 - the “pins” of software components are called interfaces
 - an interface is a set of methods that the component implements

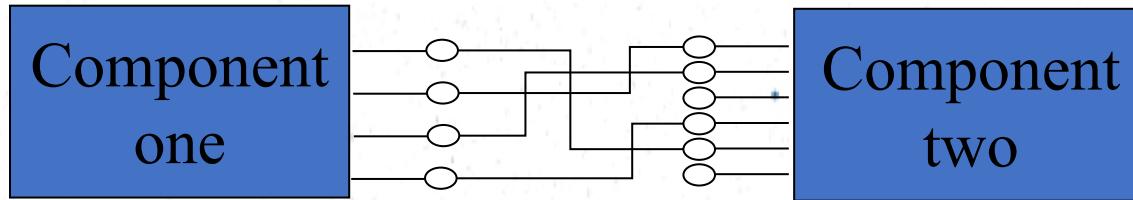


- software integrators connects components according to the descriptions of the methods within the interface.

Software components (2)

- Component technology is still young, and there isn't even agreement on the definition of its most important element - the component.
- ***"a component is a software module that publishes or registers its interfaces"***, a definition by P. Harmon, Component Development Strategy newsletter (1998).
- Note: a component does not have to be an object!
An object can be implemented in any language as long as all the methods of its interface are implemented.
- Objects – Design level
- Components – Architectural level

Component wiring



- The traditional programming model is caller-driven (application calls methods of a component's interface, information is pulled from the callee as needed). Component never calls back.
- In the component programming model, connecting components may call each other (connection-oriented programming).
- The components can interact with each other through **event notification** (a component pushes information to another component as some event arises). This enables wiring components at runtime.

Pragmatic definition

- Software Components:
 - predictable behavior: implement a *common* interface
 - embedded in a framework that provides common services (events, persistence, security, transactions,...)
 - developer implements only business logic

Distributed Component Models

- Hide implementation details
- Bring power of a container
- Focus on business logic
- Enable development of third-party interoperable components

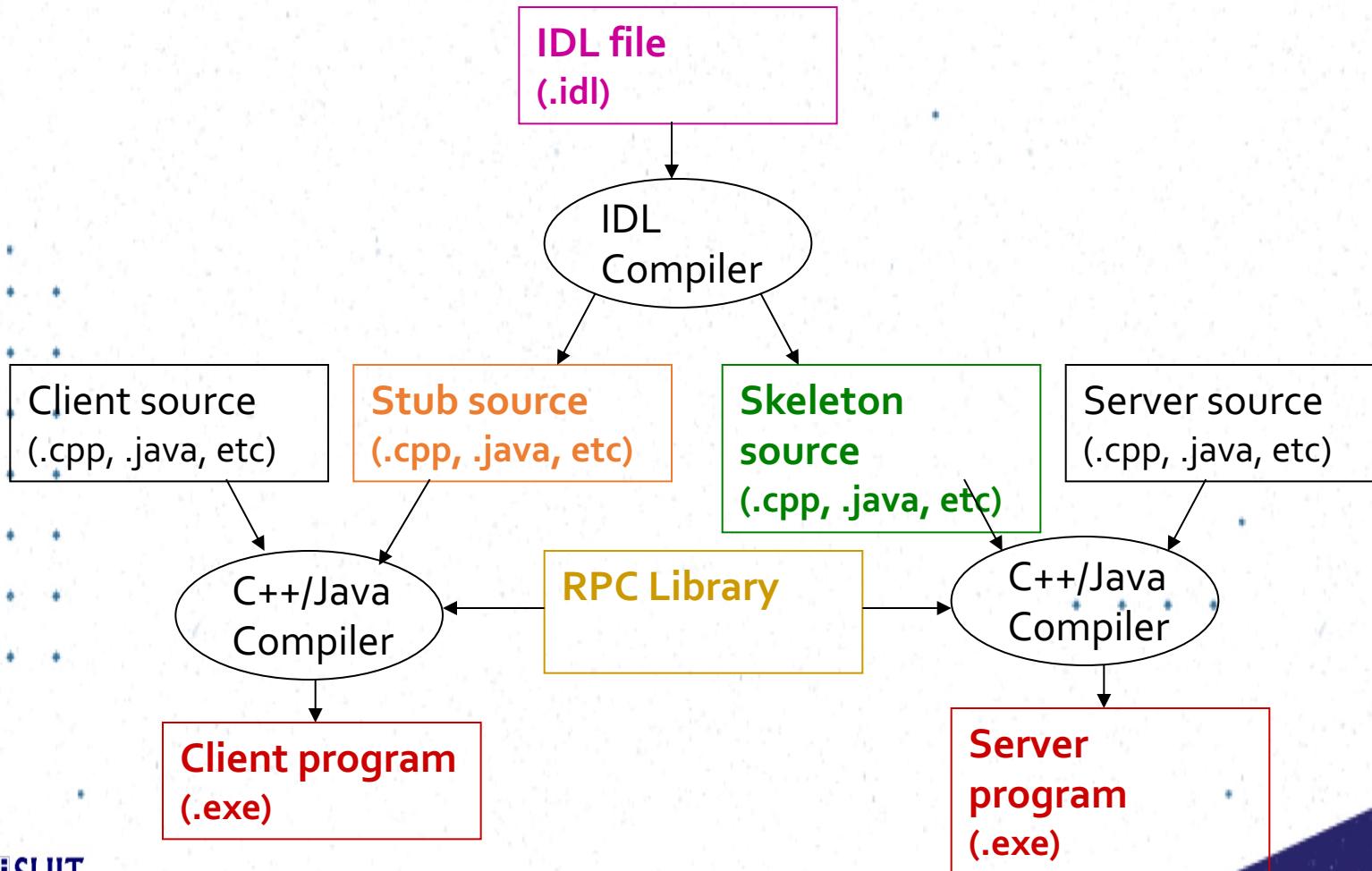
Examples of component frameworks

- CORBA (Component Object Request Broker Architecture)
- Java/Java EE - EJB (Enterprise Java Beans)
- Spring Framework
- Microsoft/.NET – DCOM, WCF Services

CORBA

- CORBA is the acronym for Common Object Request Broker Architecture
- CORBA grew out of academic efforts to build a distributed computing framework around RPC
 - Dubbed ‘middleware’ since it aims to transparently connect systems running on different platforms
- CORBA specification administered by the Object Management Group (OMG)
 - Now up to CORBA 3
- Implementations of the CORBA spec are referred to as Object Request Brokers (ORBs)
 - An ORB is built for a particular language (e.g. C++, Java)

CORBA Code Generation



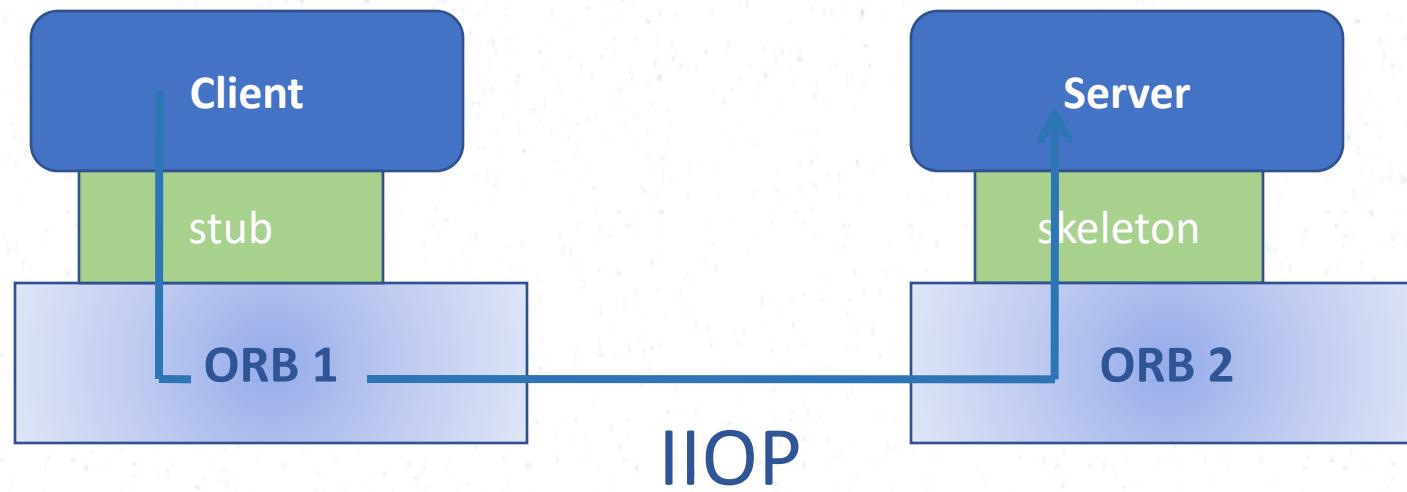
CORBA and O-O

- Object-orientation gave the opportunity to hide the internal details of RPC such as marshaling
 - Interfaces are implemented as C++/Java/etc objects that inherit from an IDL-generated C++/Java/etc class
 - Uses polymorphism to direct incoming call to your object
 - An object that implements an interface is roughly equivalent to the concept of a component
 - CORBA never really mentioned ‘component’ until the CORBA Component Model (CCM) that added things like support for authentication, persistence and transactions

CORBA Interoperability

- ORB's use IIOP to communicate with each other.
- Using the standard protocol IIOP, a CORBA-based program from any vendor, on almost any computer, operating system, programming language, and network, can interoperate with a CORBA-based program from the same or another vendor, on almost any other computer, operating system, programming language, and network.
 - IIOP (Internet Inter-ORB Operability Protocol); Defines:
 - Message types and binary message format
 - Data format - Binary format for data types (e.g. int, double, string)
 - Object reference format - identifies the object and its host server

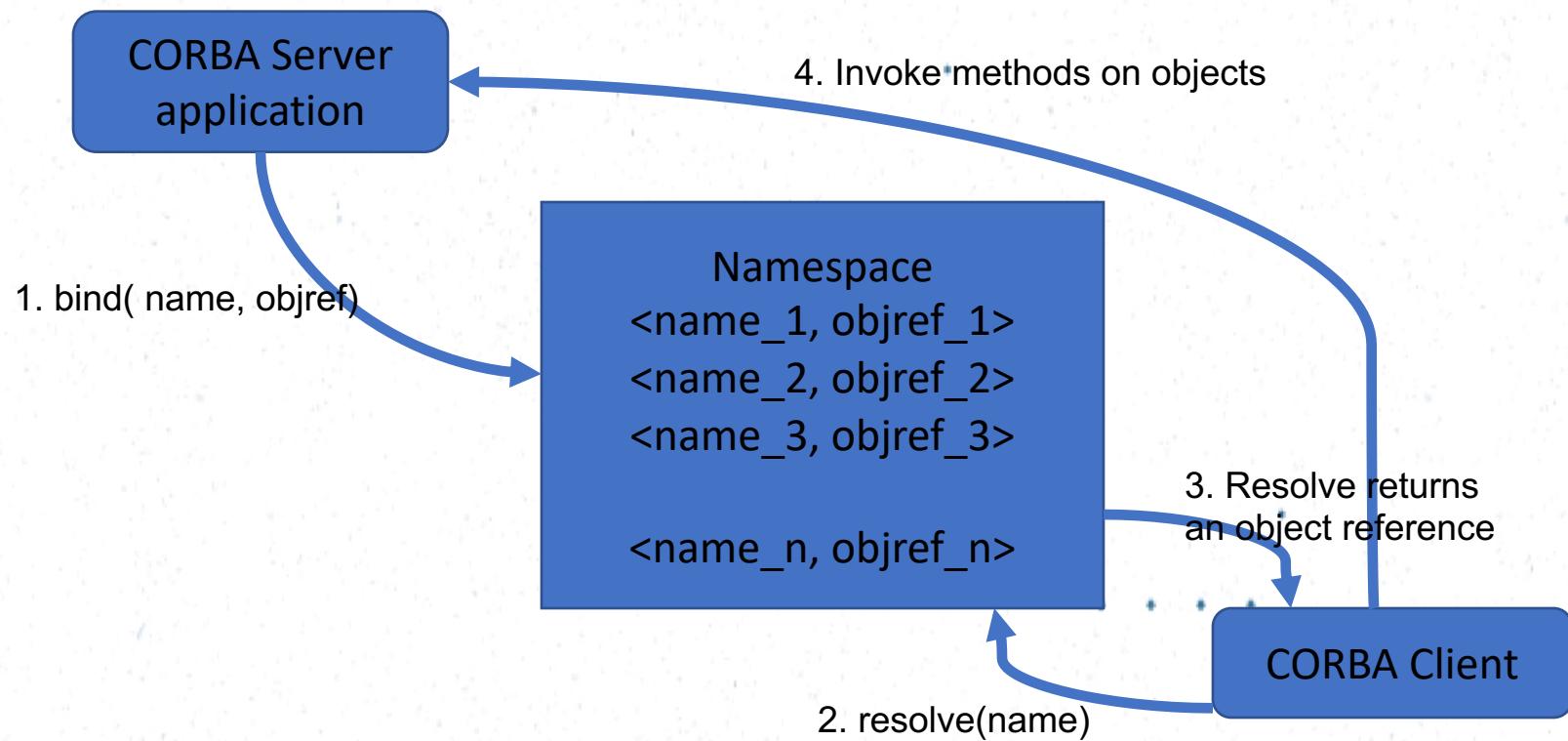
CORBA Interoperability



CORBA Naming Service

- CORBA Naming Service allows you to associate abstract names with CORBA objects and allows clients to find those objects by looking up the corresponding name.
- A naming service is a central server that knows where other servers can be found, like an index
 - Allows locating an object based on a user-friendly name, avoiding the need to hard-code object-server allocations
 - Means that the server machine, an object is hosted on, can be changed without breaking the distributed application

CORBA Naming Service



IDL Example: CORBA IDL

```
module Utilities
{
    // Basic example interface
    interface Calculator
    {
        int Add(in int operand1, in int operand2);
        double Add(in double operand1, in double operand2);
    }

    // Interface inheritance
    interface ScientificCalculator : Calculator
    {
        // Returning values by the parameter list
        void SolveQuadratic(in float a, in float b, in float c, out float
x1,
                           * out float x2);

        // Example with a string
        double EvaluateExpression(in string expr);
    }
}
```

CORBA Example (Java) - Server

```
public class CalculatorImpl
    extends CalculatorImplBase           ← xyzImplBase is generated by IDL compiler
{
    public CalculatorImpl() {
        super();                      ← Let xyzImplBase do important initialisation work
    }
    public int Add(int operand1, int operand2) {      ← Actual interface function
        return operand1 + operand2;
    }
}

import org.omg.*;

public class CalcServer {
    public static void main(String[] args) {
        CORBA.ORB orb = CORBA.ORB.init(args, null);   ← Initialise Java's ORB
        CalculatorImpl calc;
        calc = new CalculatorImpl();
        orb.connect(calc);                            ← Tell ORB about Calculator object
        sObjRef = orb.object_to_string(calc);
        System.out.println("Object ref: " + sObjRef);  ← Print stringified obj ref for use by clients
        System.out.println("Press Enter to exit");       (using a name server is better, but more complex)
        System.in.readln();                          ← Wait for client requests
    }
}
```

CORBA Example (Java) - Client

```
import org.omg.*;  
  
public class CalcClient  
{  
    public static void Main(String[] args) {  
        CORBA.ORB orb = CORBA.ORB.init(args, null); ← Initialise Java's ORB  
  
        String sServerRef = args[0]; ← Assume user has reference for server  
                                (better to use name server, but more complex)  
  
        CORBA.Object temp = orb.string_to_object(sServerRef); ← Create  
Calculator                         on remote server, return ref  
  
        Calculator calc = CalculatorHelper.narrow(temp); ← Narrow (downcast)  
                                         Object to Calculator  
  
        System.out.println("1 + 2 = " + calc.Add(1, 2));  
    }  
}
```

CORBA Notes

- Each vendor will have slightly different ways of declaring and creating CORBA objects
 - The basic organisation of the code will be similar, just the specifics of ORB initialisation and where to get classes
- Note that a CORBA server object is *first* created by the server host process, *then* registered with the ORB
 - Means that the server object is always running awaiting new incoming client connections
 - Registration (via orb.connect) must only happen once for each object

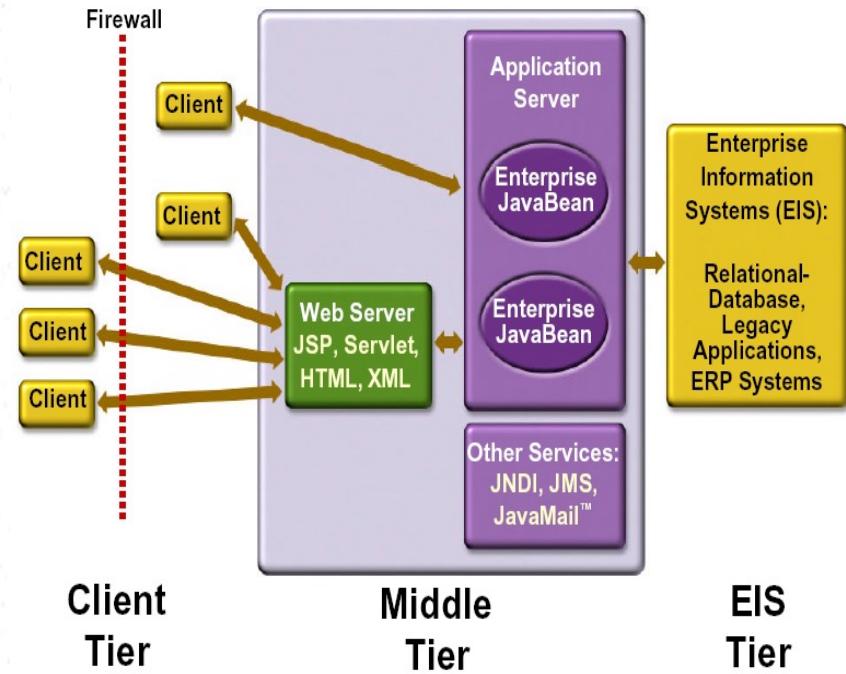
CORBA Today

- Although conceptually quite elegant, CORBA never really caught on, for a couple of reasons:
 - The CORBA spec is huge, making it complex to use
 - Competing frameworks were developed around the same time that had better backing and/or a broader coder base
- Although CORBA was not widely adopted, some concepts that it focused on such as interoperability and common standards were relevant for Web Services as well.

JEE/Enterprise Java Beans

Java EE Architecture

- Three/Four tiered applications that run in this way extend the standard two-tiered client and server model by placing a multithreaded application server between the client application and back-end storage



Java EE Containers

- The application server maintains control and provides services through an interface or framework known as a *container*
- Server-side containers:
 - The server itself, which provides the Java EE runtime environment and the other two containers
 - An **EJB container** to manage EJB components
 - A **Web container** to manage servlets and JSP pages

Java EE Components

- As said earlier, Java EE applications are made up of components
- A *Java EE component* is a self-contained functional software unit that is assembled into a Java EE application with its related classes and files and that communicates with other components
- Client components run on the client machine, which correlate to the client containers
- Web components -servlets and JSP pages
- EJB Components

Packaging Applications and Components

- Under Java EE, applications and components reside in Java Archive (JAR) files
- These JARs are named with different extensions to denote their purpose, and the terminology is important

Various File types

- Enterprise Archive (EAR) files represent the application, and contain all other server-side component archives that comprise the application
- Web components reside in Web Archive (WAR) files
- Client interface files and EJB components reside in JAR files

EJB Components

- EJB components are server-side, modular, and reusable, comprising specific units of functionality
- They are similar to the Java classes we create every day, but are subject to special restrictions and must provide specific interfaces for container and client use and access
- We should consider using EJB components for applications that require scalability, transactional processing, or availability to multiple client types

EJB Components- Major Types

- **Session beans (verbs of a system)**
 - These may be either *stateful* or *stateless* and are primarily used to encapsulate business logic, carry out tasks on behalf of a client, and act as controllers or managers for other beans
- **Entity beans (nouns of a system)**
 - Entity beans represent persistent objects or business concepts that exist beyond a specific application's lifetime; they are typically stored in a relational database
- **Message Driven Beans:**
 - Asynchronous communication with MOM
 - Conduit for non-Java EE resources to access Session and Entity Beans via JCA Resource adapters

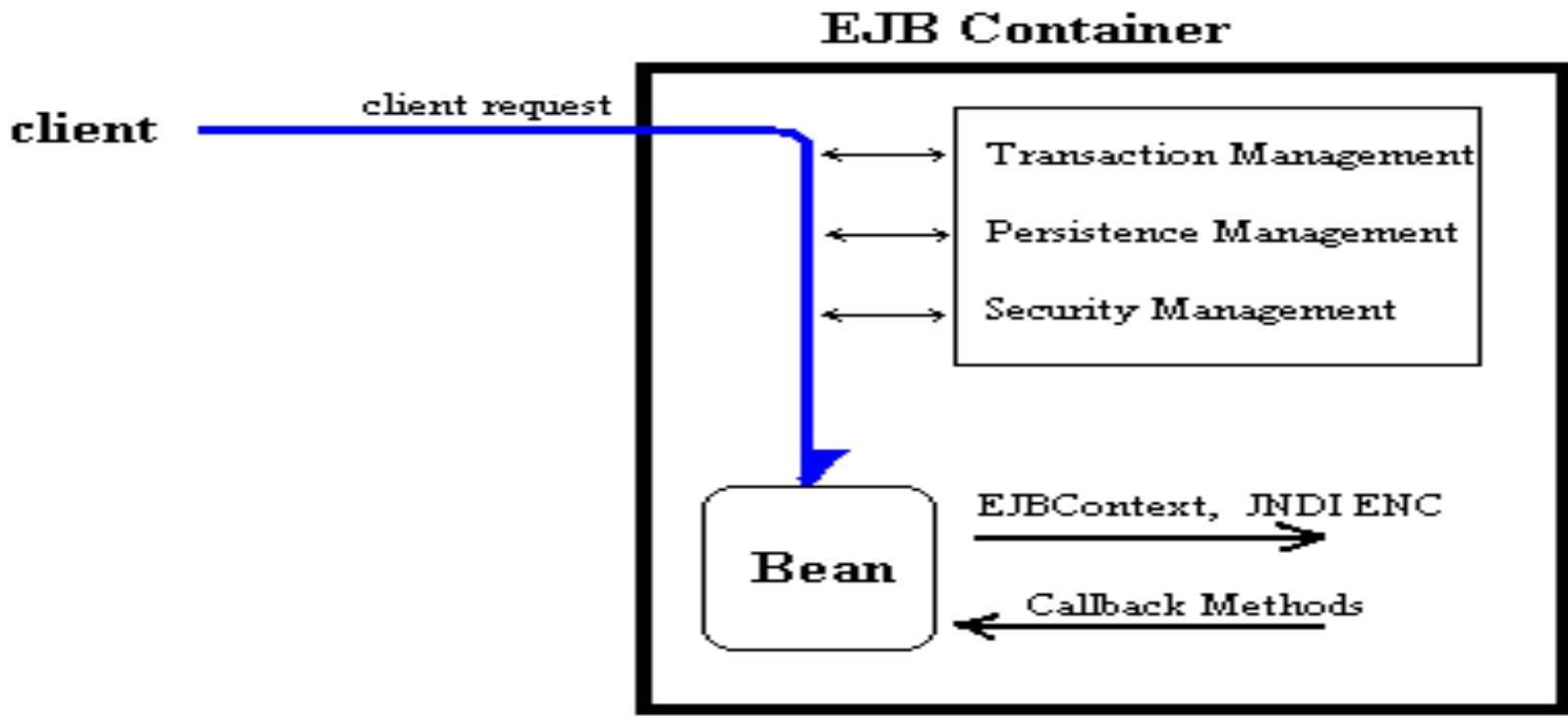
Enterprise Java Server (EJS) / Application Server

- Part of an application server that hosts EJB containers
- EJBs do not interact directly with the EJB server
- EJB specification outlines eight services that must be provided by an EJB server:
 - Naming
 - Transaction
 - Security
 - Persistence
 - Concurrency
 - Life cycle
 - Messaging
 - Timer

EJB Container

- Functions as a runtime environment for EJB components beans
- Containers are transparent to the client in that there is no client API to manipulate the container
- Container provides EJB instance life cycle management and EJB instance identification.
- Manages the connections to the enterprise information systems (EISs)

EJB Container(cont'd)

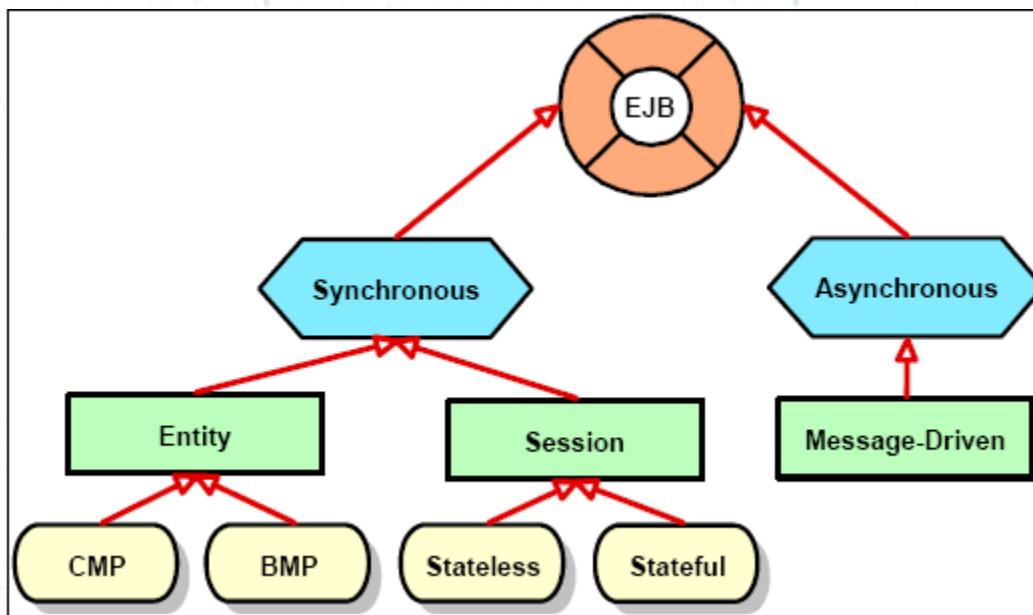


**EJB Containers manage
enterprise beans at runtime**

EJB Client

- Finds EJBs via JNDI.
- Invokes methods on EJBs.

EJB components



EJB Interfaces - Local and Remote

- Local Interface
 - Used for invoking EJBs within the same JVM (process)
 - **@Local** annotation marks an interface local
 - Parameters passed by reference
- Remote Interface
 - Used for invoking EJBs across JVMs (processes)
 - **@Remote** annotation marks an interface remote
 - Parameters passed by value (serialization/de- serialization)

Note: An EJB can implement both interfaces if needed.

Business Interface

- Defines business methods
- Session beans and message-driven beans require a business interface, optional for entity beans.
- Business interface do not extend local or remote component interface (unlike EJB2.x)
- Business Interfaces are POJIs (Plain Old Java Interfaces)

Business Interface - examples

- Shopping cart that maintains state

```
public interface ShoppingStatefulCart {  
    void startShopping(String customerId);  
    void addProduct(String productId);  
    float getTotal();  
}
```

- Shopping cart that does not maintain state

```
public interface ShoppingStatelessCart {  
    String startShopping(String customerId); //  
    return cartId  
    void addProduct(String cartId, String productId);  
    float getTotal(String cartId);  
}
```

Stateless Session EJB (SLSB)

- Does not maintain any conversational state with client
- Instances are pooled to service multiple clients
- **@Stateless** annotation marks a bean stateless.
- Lifecycle event callbacks supported for stateless session beans (optional)
 - **@PostConstruct** occurs before the first business method invocation on the bean
 - **@PreDestroy** occurs at the time the bean instance is destroyed

Stateless Session EJB example (1/2)

- The business interface:

```
public interface HelloSessionEJB3Interface{  
    public String sayHello();  
}  
* * *  
* * *  
* * *  
* * *  
* * *  
* * *  
* * *  
* * *
```

Stateless Session EJB example (2/2)

- The stateless bean with local interface:

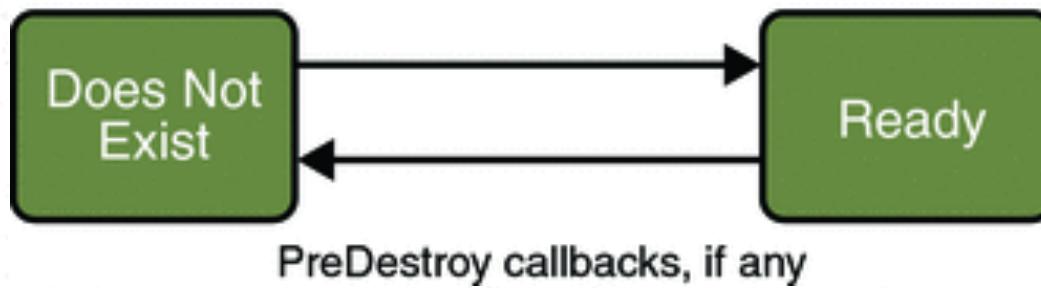
```
import javax.ejb.*;
import javax.annotation.*;
@Local({HelloSessionEJB3Interface.class})
@Stateless public class HelloSessionEJB3 implements

    HelloSessionEJB3Interface{
public String sayHello(){
    return "Hello from Stateless bean";
}
@PreDestroy void restInPeace() {
    System.out.println("I am about to die now");
}
```

Lifecycle of a Stateless Session Bean

- A client initiates the life cycle by obtaining a reference
- The container invokes the `@PostConstruct` method, if any
- The bean is now ready to have its business methods invoked by clients

- - 1. Dependency injection, if any
 - 2. PostConstruct callbacks, if any



Stateful Session EJB (SFSB)

- Maintains conversational state with client
- Each instance is bound to specific client session
- Support callbacks for the lifecycle events listed on the next slide

Stateful Session EJB – example (1/2)

- Define remote business interface (remote can be marked in bean class also) :

```
@Remote public interface ShoppingCart {  
    public void addItem(String item);  
    public void addItem(String item);  
    public Collection getItems();  
}
```

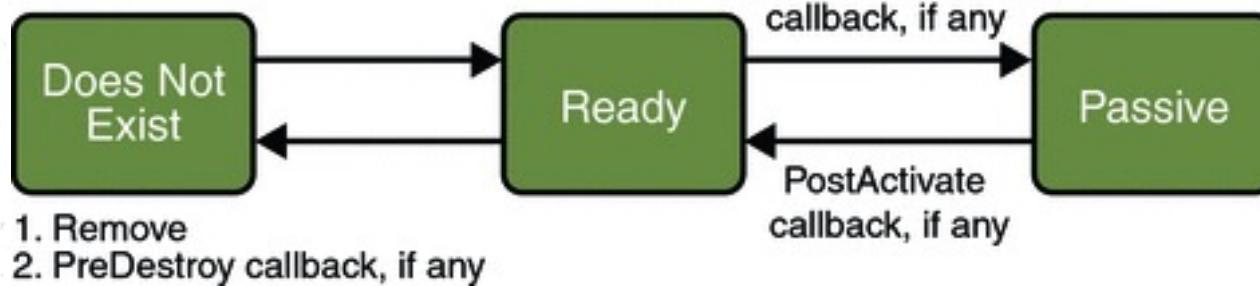
Stateful Session EJB – example (2/2)

```
@Stateful public class CartBean implements  
 ShoppingCart {  
     private ArrayList items;  
     @PostConstruct public void initArray() {  
         items = new ArrayList();  
     }  
     public void addItem(String item) {  
         items.add(item);  
     }  
     public void removeItem(String item) {  
         items.remove(item);  
     }  
     public Collection getItems() {  
         return items;  
     }  
     @Remove void logoff() {items=null;}  
 }
```

Lifecycle of a Stateful Session Bean

- Client initiates the lifecycle by obtaining a reference
- Container invokes the @PostConstruct and @Init methods, if any
- Now bean ready for client to invoke business methods

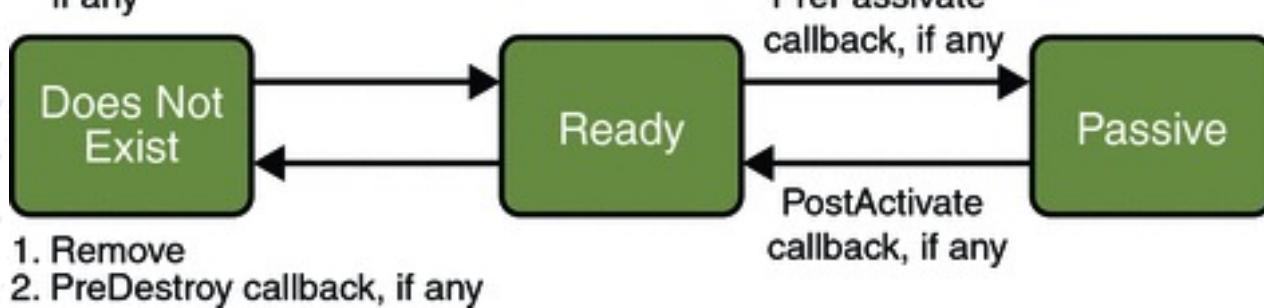
- - 1. Create
 - 2. Dependency injection, if any
 - 3. PostConstruct callback, if any
 - 4. Init method, or ejbCreate<METHOD>, if any



Lifecycle of a Stateful Session Bean

- While in ready state, container may passivate and invoke the `@PrePassivate` method, if any
- If a client then invokes a business method, the container invokes the `@PostActivate` method, if any, and it returns to ready stage

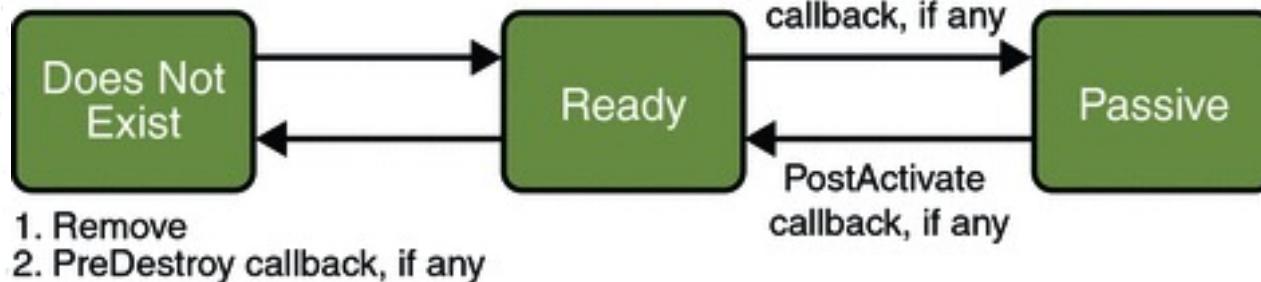
- - 1. Create
 - 2. Dependency injection, if any
 - 3. PostConstruct callback, if any
 - 4. Init method, or `ejbCreate<METHOD>`, if any



Lifecycle of a Stateful Session Bean

- At the end of the life cycle, the client invokes a method annotated @Remove
- The container calls the @PreDestroy method, if any

- • •
- 1. Create
- 2. Dependency injection, if any
- 3. PostConstruct callback, if any
- 4. Init method, or ejbCreate<METHOD>, if any



Entity EJB (1)

- **It is permanent.** Standard Java objects come into existence when they are created in a program. When the program terminates, the object is lost. But an entity bean stays around until it is deleted. In practice, entity beans need to be backed up by some kind of permanent storage, typically a database.
- **It is identified by a primary key.** Entity Beans must have a primary key. The primary key is unique -- each entity bean is uniquely identified by its primary key. For example, an "employee" entity bean may have Social Security numbers as primary keys.
- Note: Session beans do not have a primary key.

Entity Bean Class

- `@Entity` annotation marks a class as Entity EJB
- Persistent state of an entity bean is represented by non-public instance variables
- For single-valued persistent properties, these method signatures are:
`<Type> getProperty()`
`void setProperty(<Type> t)`
- Must be a non-final concrete class
- Must have public or protected no-argument constructor
- No methods of the entity bean class may be final
- If entity bean must be passed by value (through a remote interface) it must implement `Serializable` interface

Entity EJB

- CMP (Container Managed Persistence)
 - Container maintains persistence transparently using JDBC calls
- BMP (Bean Managed Persistence)
 - Programmer provides persistence logic
 - Used to connect to non-JDBC data sources like LDAP, mainframe etc.
 - Useful for executing stored procedures that return result sets

Entity EJB – example (1)

```
@Entity // mark as Entity Bean
public class Customer implements Serializable {
    private Long id;
    private String name;
    private Collection<Order> orders = new
        HashSet();
    @Id(generate=SEQUENCE) // primary key
    public Long getId() {
        return id;
    }
    public void setId(Long id) { ...
        this.id = id;
    }
    ...
}
```

Entity EJB – example (2)

```
@OneToMany // relationship between  
Customer and Orders  
public Collection<Order> getOrders() {  
    return orders;  
}  
public void setOrders(Collection<Order>  
    orders) {  
    this.orders = orders;  
}  
}
```

EntityManager

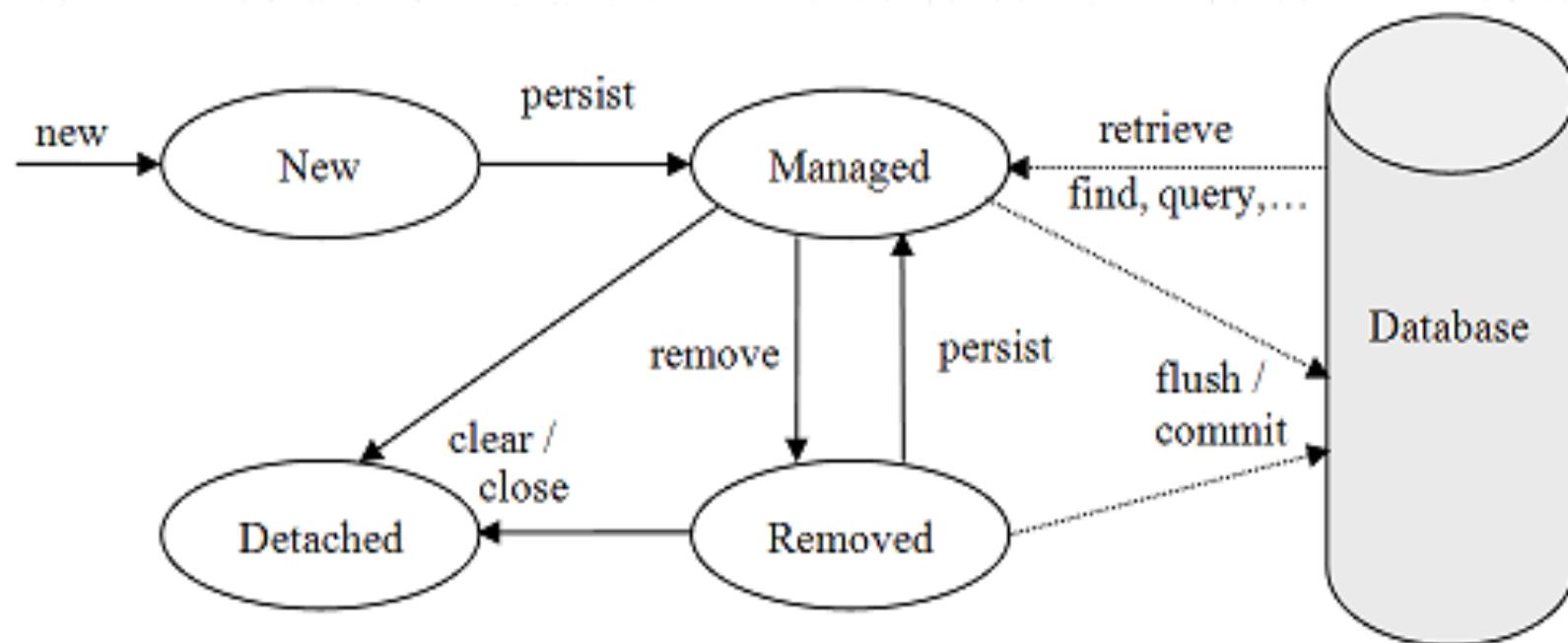
- EntityManager API is used to:
 - create and remove persistent entity instances
 - to find entities by their primary key identity, and to query over entities
- EntityManager supports EJBQL and (non-portable) native SQL

Entity Bean Lifecycle

Entity bean instance has four possible states:

- **New** entity bean instance has no persistent identity, and is not associated with a persistence context.
- **Managed** entity bean instance is an instance with a persistent identity that is currently associated with a persistence context.
- **Detached** entity bean instance is an instance with a persistent identity that is not (or no longer) associated with a persistence context.
- **Removed** entity bean instance is an instance with a persistent identity, associated with a persistence context, scheduled for removal from the database.

Entity Bean Lifecycle



Example of Use of EntityManager API

```
@Stateless public class OrderEntry {  
    @Inject EntityManager em;  
    public void enterOrder(int custID, Order  
    newOrder) {  
        Customer cust = (Customer)em.find("Customer",  
                                         custID);  
        cust.getOrders().add(newOrder);  
        newOrder.setCustomer(cust);  
    }  
}
```

Message Driven EJB

- Invoked by asynchronously by messages
- Cannot be invoked with local or remote interfaces
- **@MessageDriven** annotation within class marks the Bean message driven
- Stateless
- Transaction aware

Message Driven EJB example

```
import javax.jms.Message;
import javax.jms.MessageListener;
import javax.ejb.MessageDriven;
@MessageDriven
public class MessageDrivenEJBBean implements
    MessageListener {
    ...
    public void onMessage(Message message) {
        if(message instanceof MyMessageType1)
            doSomething(); // business method 1
        if(message instanceof MyMessageType2)
            doSomethingElse(); // business method 2
    ...
}
    ...
}
```

EJB Query Language (EJBQL)

- EJBQL : RDBMS vendor independent query syntax
- Query API supports both static queries (i.e., named queries) and dynamic queries.
- Since EJB3.0, supports HAVING, GROUP BY, LEFT/RIGHT JOIN etc.

EJBQL - examples

- Define named query:

```
@NamedQuery(  
    name="findAllCustomersWithName",  
    queryString="SELECT c FROM Customer c WHERE c.name  
    LIKE :custName")
```

...
...

- Use named query:

```
@Inject public EntityManager em;  
//..  
List customers =  
    em.createNamedQuery("findAllCustomersWithName")  
        .setParameter("custName", "Smith")  
        .getResultList();
```

Deploying EJBs

- EJB 3.0 annotations replace EJB 2.0 deployment descriptors in almost all cases
- Values can be specified using annotations in the bean class itself
- Deployment descriptor *may be used* to override the values from annotations

Some EJB Servers (Application Servers)

Company

- IBM
- BEA Systems
- Sun Microsystems
- Oracle
- JBoss

Product

- WebSphere
- BEA WebLogic
- Sun Application Server
- Oracle Application Server
- JBoss

Advantages of EJB

- Simplifies the development of middleware components that are secure, transactional, scalable & portable.
- Simplifies the process to focus mainly on business logic rather than application development.
- Overall increase in developer productivity
- Reduces the time to market for mission critical applications

Summary

- Component based development focuses in grouping cohesive functions into reusable units called components
- Components interact with other components and clients through interfaces
- Distributed computing makes extensive use of component based development as it allows the different functionalities to be distributed over different systems.
- Different component development technologies are there by different vendors (Java EE, Spring Framework, .NET WCF services, etc.)

Lecture 7 – Open Message Formats

This Week

- In previous weeks we studied about different component and RPC frameworks.
 - This week we will look at XML and JSON which are used in network transfer of objects via serialization.
 - Many of current RPC methods use these message formats widely in transferring data.

XML

What is XML?

- eXtensible Markup Language
- Marked-up text is text that is structured by marking it with textual tags to describe the meaning of the text contents
- A markup language only defines the set of valid tags and their valid structure, not what to do with them

ASCII Data

- 10, Nimal, 56
- Data can be stored like above in ASCII format so it can be retrieved in any platform (including mainframes etc).
- But we don't know what 10, 56 mean.

XML Data

- XML is self describing and also platform neutral

```
<Person>
  <Empno> 10 </Empno>
  <Name> Nimal </Name>
  <Age> 56 </Age>
</Person>
```

XML vs HTML

- HTML (HyperText Markup Language) defines the valid tags and structure for Web pages
 - Fairly inconsistent standard; eg: tags don't always need to be closed with </tag>, eg: <p> (paragraph)
- XML can be used to mark up text.
- A language used to define other languages (Meta Language)
 - XML itself has no tags
 - XML tags are not predefined. You must "invent" your own tags (new language)
- HTML is about ***displaying*** information, while XML is about ***describing*** information. (the main difference)

Why XML ?

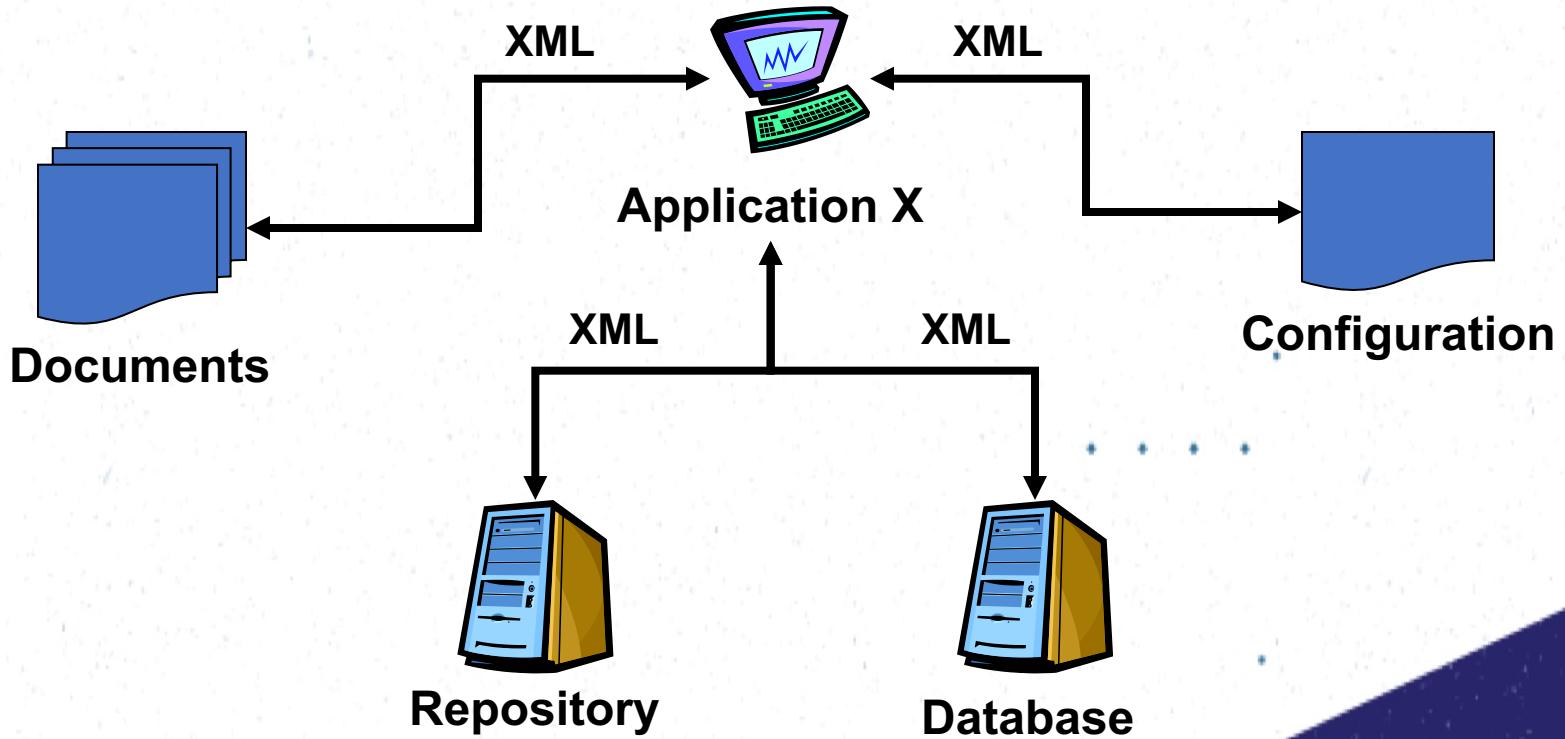
- In its simplest form, XML is a markup language for documents that contain structured data.
- The key tenet of XML is that it can be used to describe any data in a human-readable, structured form.
- Structured information can contain both content, as well as information that defines the content.
- XML is a cross-platform, software and hardware independent tool for transmitting information/data.

Where to Use XML

- XML is everywhere.
- XML is ideal for:
 - e-commerce (in particular B2B), content management, Web Services, distributed computing, peer-to-peer (P2P) networking and the Semantic Web...
 - However, it's huge in space
 - 3-20 times larger comparing to binary format
- XML will be as important to the future of the Web as HTML has been to the foundation of the Web and that XML will be the most common tool for all data manipulation and data transmission.

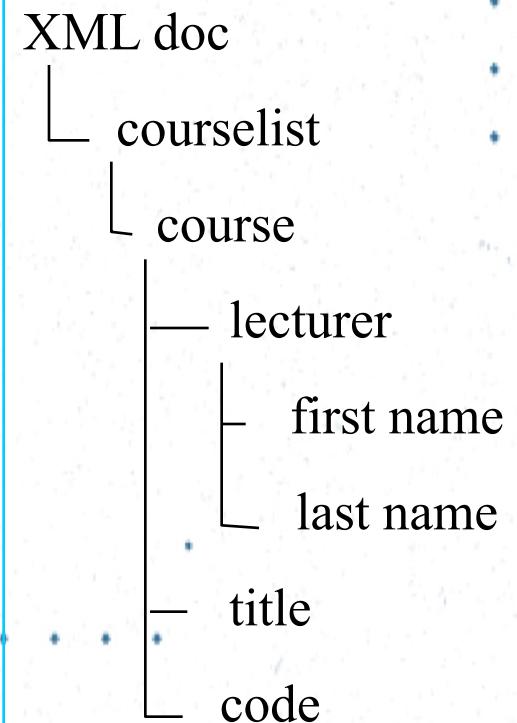
Universal Language

- XML is a “use everywhere” data specification



A Sample XML Document

```
<?xml version = "1.0" encoding="ISO-8859-1"?>
<!-- my first XML doc -->
<courselist xmlns="http://www.university.com" >
<course>
<lecturer>
  <firstname> H.T. </firstname>
  <lastname> Shen </lastname>
</lecturer>
<title>SOA</title>
<code>INFS3204</code>
</course>
</courselist>
```



Benefits of XML

- Open W3C standard
- Representation of data across heterogeneous environments
 - Cross platform
 - Allows for high degree of interoperability
- Strict rules
 - Syntax
 - Structure
 - Case sensitive

XML Syntax

- An XML document are composed of
 - An XML declaration: <?xml version = "1.0" encoding="ISO-8859-1"?>
 - Optional, but should be used to identify the XML version, namespace, and encoding schema to which the doc conforms
 - XML markup types
 - **1.Elements (and attributes)**
 - 2.Entity references
 - 3.Comments: <!-- what ever you want to say -->
 - 4.Processing instructions: <?instruction options?>
 - **Content**

Elements and Attributes

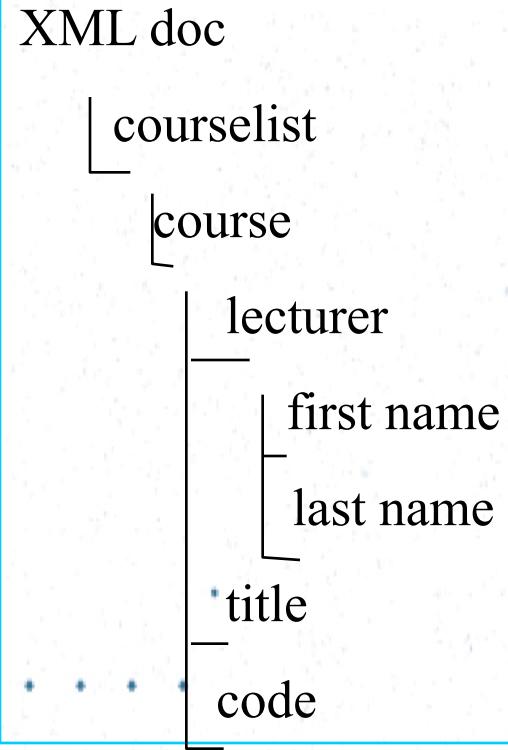
- Elements (or Tags): <course></ course>
 - Primary building blocks of an XML document
 - All tags must be closed and nested properly (as in XHTML)
 - Empty tags ok (e.g., < course />)
- Attributes: < course level = "undergraduate">
 - Additional information about an element
 - Attribute value can be required, optional or fixed, and can have a default value
 - An attribute can often be replaced by nested elements
 - All values must be quoted (even for numbers, as in XHTML)

Elements and Attributes

- Elements and attributes can be named in almost any way you like
 - Should be descriptive and not confusing (human-readable!)
 - No length limit, case sensitive and ok to use the underscore (_)
 - No names are reserved for XML (namespaces can solve naming conflicts)
 - Must not start with a number or punctuation character or XML or Xml
 - Cannot contain spaces, the colon (:), space, greater-than (>) , or less-than (<)
 - Avoid using the hyphen (-) and period (.)

Element Relationship

```
<?xml version = "1.0" encoding="ISO-8859-1"?>
<!-- my first XML doc -->
<courselist xmlns="http://www.university.com" >
  <course>
    <lecturer>
      <firstname> H.T. </firstname>
      <lastname> Shen </lastname>
    </lecturer>
    <title>SOA</title>
    <code>INFS3204</code>
  </course>
</courselist>
```



courselist is the **root element**. *lecturer*, *title* and *code* are **child elements** of *course*. *lecturer*, *title* and *code* are **siblings** (or **sister elements**) because they have the same parent.

XML Content

- The text within the elements
 - So the document is structured!
- XML content can consist of any data
 - As long as the content does not confuse with valid XML metadata
 - instructions (use entity references for special characters!)
- Every XML doc must have one and only one root element

```
<courselist>
  <course>
    <lecturer>
      <firstname>H.T</firstname>
      <lastname>Shen</lastname>
    </lecturer>
    <title>SOA</title>
    <code>INFS3204</code>
  </course>
</courselist>
```

Well-formed XML 1?

```
<xml? version="1.0" ?>
<PARENT>
    <CHILD1>This is element 1</CHILD1>
    <CHILD2><CHILD3>Number 3</CHILD2></CHILD3>
</PARENT>
```

Well-formed XML 2?

```
<xml? version="1.0" ?>
<PARENT>
    <CHILD1>This is element 1</CHILD1>
    <CHILD2/>
    <CHILD3></CHILD3>
</PARENT>
```

XML Namespaces

- XML tags are user defined
- Therefore it's possible that we can use the same element name to define two different things
- This will lead to a naming conflict.
- Also real world applications may have to use several XML based languages defined by various parties
- These several languages may have same element names

```
<Employee>
  <name>John Smith</name>
  <department>Admin</department>
  <salary>40000</salary>
  <dependents>
    <dependent>
      <name> Mark</name>
      <age>12</age>
    </dependent>
  </dependents>
</Employee>
```

XML Namespaces

- Eg:
 - Think about a online furniture store selling furniture items from multiple vendors
 - Many vendors will have same `<table>` element
 - To resolve this kind of conflicts namespaces are used.

```
<table>
  <tr>
    <td>Apples</td>
    <td>Bananas</td>
  </tr>
</table>
```

```
<table>
  <name>African Coffee Table</name>
  <width>80</width>
  <length>120</length>
</table>
```

XML Namespaces

- Name conflicts in XML can easily be avoided using a name prefix.

```
<h:table>
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>

<f:table>
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>
```

In the example
<table>

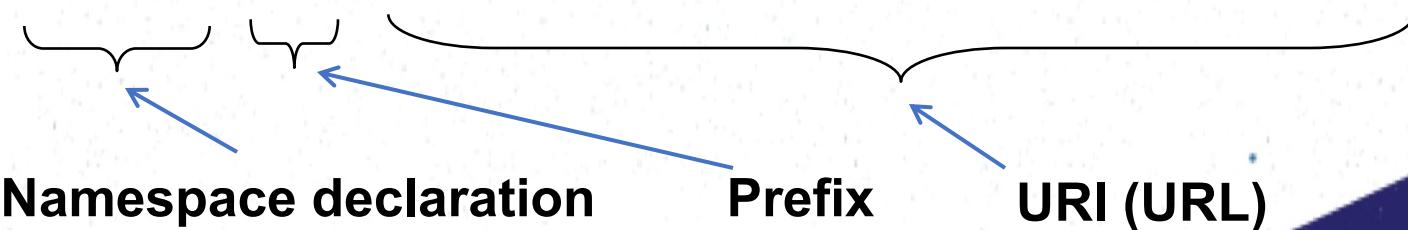
elements have different names.

/O

XML Namespaces

- When using prefixes in XML, a **namespace** for the prefix must be defined
- The namespace is defined by the **xmlns attribute** in the start tag of an element.
- The namespace declaration has the following syntax.
`xmlns:prefix="URI".`
- A namespace is identified by a URI

```
xmlns: bk = "http://www.example.com/bookinfo/"
```

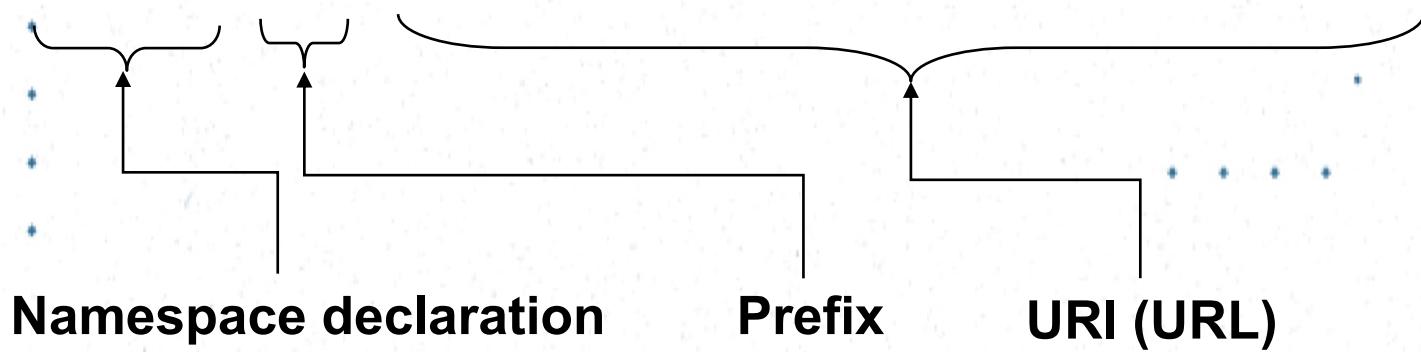


Namespaces: Declaration

```
xmlns: bk = "http://www.example.com/bookinfo/"
```

```
xmlns: bk = "urn:mybookstuff.org:bookinfo"
```

```
xmlns: bk = "http://www.example.com/bookinfo/"
```



Multiple XML Namespaces

I want to use my own furniture definitions (as the default), and two more name spaces (one is HTML and another is IKEA's furniture definitions)

```
<furniture xmlns = “http://www.itee.uq.edu.au/~zxf/furniture”  
           xmlns:ikea = “http://www.ikea.com/names”  
           xmlns:html=“http://www.w3.org/TR_REC-html40”>
```

- **<furniture> element has 3 namespaces**
 - The first one is the default one in this example
 - No naming conflicts anymore
 - <table>, <ikea:table>, <html:table>
 - All child elements of <furniture> inherit the 3 namespaces (unless overwritten)

Default and scope

- An XML namespace declared without a prefix becomes the default namespace for all sub-elements
- All elements without a prefix will belong to the default namespace
- Unqualified elements belong to the inner-most default namespace.
 - BOOK, TITLE, and AUTHOR belong to the default book namespace
 - PUBLISHER and NAME belong to the default publisher namespace

```
<BOOK xmlns="www.bookstuff.org/bookinfo">
    <TITLE>All About XML</TITLE>
    <AUTHOR>Joe Developer</AUTHOR>
    <PUBLISHER xmlns="urn:publishers:publinfo">
        <NAME>Microsoft Press</NAME>
    </PUBLISHER>
</BOOK>
```

Another sample XML document

```
<?xml version="1.0" encoding="utf-8"?>
<courselist xmlns:c="http://www.university.com/courses"
             xmlns:l="http://www.university.com/lecturers">
    <c:course id="001">
        <c:name>SPDII</c:name>
        <l:lecturers>
            <l:lecturer>
                <l:fname>Sheron</l:fname>
                <l:lname>Dinushka</l:lname>
            </l:lecturer>
            <l:lecturer>
                <l:name>Nishani</l:name>
                <l:lname>Ranpatabandi</l:lname>
            </l:lecturer>
        </l:lecturers>
    </c:course>
</courselist>
```

Two namespaces

another way...

```
<?xml version="1.0" encoding="utf-8"?>
<courselist xmlns="http://www.university.com/courses"
    xmlns:l="http://www.university.com/lecturers">
    *
    * <course id="001">
    *     <name>SPDII</name>
    *     <l:lecturers>
    *         *
    *         *     <l:lecturer>
    *             <l:fname>Sheron</l:fname>
    *             <l:lname>Dinushka</l:lname>
    *             </l:lecturer>
    *             <l:lecturer>
    *                 <l:name>Nishani</l:name>
    *                 <l:lname>Ranpatabandi</l:lname>
    *             </l:lecturer>
    *         </l:lecturers>
    *     </course>
    </courselist>
```

Default namespace

XML Schema (XSD)

- XML Schema Definition (XSD) language
 - Description of the structure of an XML document
 - XSD is itself an XML file following a fixed standard
 - Replaces the less-flexible DTD (Document Type Definition)
 - XSD used by parsers to check that an associated XML file is **valid** (as opposed to merely **well-formed**)
 - Well-formed: General XML syntax rules are followed
 - eg: Tags have end-tags, nesting is correct
 - Valid: Document follows XSD's semantics
 - eg: tags/attributes used are all defined in XSD; structure is OK

What does XSD define?

- An XML Schema:
 - defines elements that can appear in a document
 - defines attributes that can appear in a document
 - defines which elements are child elements
 - defines the order of child elements
 - defines the number of child elements
 - defines whether an element is empty or can include text
 - defines data types for elements and attributes.
 - defines default and fixed values for elements and attributes

Lets look at an example (Note.xml)

```
<?xml version="1.0"?>  
  
<note  
    xmlns="http://www.w3schools.com/note"  
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
    xsi:schemaLocation="http://www.w3schools.com/note.xsd">  
  
    <to>Tove</to>  
    <from>Jani</from>  
    <heading>Reminder</heading>  
    <body>Don't forget me this weekend!</body>  
    </note>
```

XSD for Note.xml

```
<?xml version="1.0"?>

<xs:schema xmlns:xs= "http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com/note"
xmlns="http://www.w3schools.com"
elementFormDefault= "qualified">

    <xs:element name="note">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="to"    type="xs:string"/>
                <xs:element name="from"   type="xs:string"/>
                <xs:element name="heading" type="xs:string"/>
                <xs:element name="body"   type="xs:string"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

<schema> element

- The <schema> element is the root element of every XML Schema.
- The <schema> element may contain some attributes.

➤ **xmlns:xs= <http://www.w3.org/2001/XMLSchema>**

- Indicates that the elements and data types that come from the "http://www.w3.org/2001/XMLSchema" namespace should be prefixed with xs:

<schema> element

➤ **targetNamespace="http://www.w3schools.com/note"**

- Indicates that the elements defined by this schema (note, to, from, heading, body.) belong to the target namespace.

➤ **xmlns=http://www.w3schools.com**

- Indicates the default namespace

➤ **elementFormDefault="qualified"**

- Indicates that elements used by the XML instance document which were declared in this schema must be namespace qualified.

Referencing a XSD in a XML doc

```
> xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance  
> xsi:schemaLocation="http://www.w3schools.com note.xsd">
```

- XML documents can be linked with their schemas using the schemaLocation attribute.
- Because the schemaLocation attribute itself is in the http://www.w3.org/2001/XMLSchema-instance namespace, it is necessary to declare this namespace and map it to a prefix, usually xsi.

Complex Types in XSD

- **What is a Complex Element?**

- A complex element is an XML element that contains other elements and/or attributes.

- **Ex:** <employee>

```
<firstname>John</firstname>
<lastname>Smith</lastname>
</employee>
```

- The "employee" element can be declared directly by naming the element, like this:

```
<x:element name="employee">
  <x:complexType>
    <x:sequence>
      <x:element name="firstname" type="xs:string"/>
      <x:element name="lastname" type="xs:string"/>
    </x:sequence>
  </x:complexType>
</x:element>
```

Simple Elements in XSD

- The syntax for defining a simple element is:

```
<xss:element name="xxx" type="yyy"/>
```

- Where xxx is the name of the element and yyy is the data type of the element.
- XML Schema has a lot of built-in data types. The most common types are:
 - xs:string
 - xs:decimal
 - xs:integer
 - xs:boolean
 - xs:date
 - xs:time

Simple Elements in XSD

- Ex

```
<lastname>Smith</lastname>
<age>36</age>
<dateborn>1970-03-27</dateborn>
```

- Corresponding simple element definitions:

```
<xsd:element name="lastname"
  type="xsd:string"/>
<xsd:element name="age"
  type="xsd:integer"/>
<xsd:element name="dateborn"
  type="xsd:date"/>
```

Attributes in XSD

- The syntax for defining an attribute is:

```
<xs:attribute name="xxx" type="yyy"/>
```

- Where xxx is the name of the attribute and yyy specifies the data type of the attribute.
- Simple elements can't have attributes!

- Ex

```
<lastname lang="EN">Smith</lastname>
```

- corresponding attribute definition:

```
<xs:attribute name="lang" type="xs:string"/>
```

Stocks Example: XML for XSD

```
<?xml version="1.0"?>
<?xmlstylesheet type="text/css" href="StockPortfolio.css"?>
<StockPortfolio xmlns:xs="http://www.w3.org/2001/XMLSchema"
                  xmlns:xsi="http://www.w3.org/2001/XMLSchema-Instance"
                  xsi:schemaLocation="http://www.cs.curtin.edu.au/spd361
StockPortfolio.xsd">
  <Stocks>
    <StockPurchase>
      <Ticker>GOOG</Ticker>
      <PurchasePrice>330.06</PurchasePrice>
      <NumPurchased>30</NumPurchased>
    </StockPurchase>

    <StockPurchase>
      <Ticker>MSFT</Ticker>
      <PurchasePrice>17.21</PurchasePrice>
      <NumPurchased>580</NumPurchased>
    </StockPurchase>
  </Stocks>
</StockPortfolio >
```

Stocks Example: XSD

```
• <?xml version="1.0" encoding="UTF-8"?>
• <xsschema xmlns:xs="http://www.w3.org/2001/XMLSchema">
•   <xselement name="StockPortfolio">
    <xsccomplexType>
      <xsssequence>
        <xselement name="Stocks"/>
      </xsssequence>
    </xsccomplexType>
  </xselement>
  <xselement name="Stocks">
    <xsccomplexType>
      <xsssequence>
        <xselement name="StockPurchase" minOccurs="0"
          maxOccurs="unbounded"/>
      </xsssequence>
    </xsccomplexType>
  </xselement>
•
```

- <xs:element name="StockPurchase">
- <xs:complexType>
- <xs:sequence>
- <xs:element name="Ticker"/>
- <xs:element name="PurchasePrice"/>
- <xs:element name="NumPurchased"/>
- </xs:sequence>
- </xs:complexType>
- </xs:element>
- <xs:element name="Ticker" type="xs:string"/>
- <xs:element name="NumPurchased" type="xs:int"/>
- <xs:element name="PurchasePrice" type="xs:double"/>
- </xs:schema>

Exercise

- Write the XSD for following XML

```
<?xml version = "1.0" encoding="ISO-8859-1"?>
<courselist>
  <course>
    <lecturer>
      <firstname> H.T. </firstname>
      <lastname> Shen </lastname>
    </lecturer>
    <title>SOA</title>
    <code>INFS3204</code>
  </course>
</courselist>
```

Automating XSD Creation

- The XSD schema example was written by hand
 - Error prone
 - Time consuming
- Tools exist that will take classes and generate appropriate XSD schemas automatically
 - .NET: xsd.exe
 - Java: JAXB (Java Architecture for XML Binding) xjc.exe
 - There are others

Displaying XML

- XSLT (extensible stylesheet language transformations) is used to format XML documents
- XML contains ‘content’
- XSLT contains ‘formatting info’
- XML + XSLT → HTML

XSLT Example (cdcatalog.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog>
  <cd>
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <country>USA</country>
    <company>Columbia</company>
    <price>10.90</price>
    <year>1985</year>
  </cd>
  .
  .
</catalog>
```

XSL Stylesheet (cdcatalog.xsl)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
    <body>
      <h2>My CD Collection</h2>
      <table border="1">
        <tr bgcolor="#9acd32">
          <th>Title</th>
          <th>Artist</th>
        </tr>
        <xsl:for-each select="catalog/cd">
          <tr>
            <td><xsl:value-of select="title"/></td>
            <td><xsl:value-of select="artist"/></td>
          </tr>
        </xsl:for-each>
      </table>
    </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

Link the XSL stylesheet to XML

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="cdcatalog.xsl"?>
<catalog>
  <cd>
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <country>USA</country>
    <company>Columbia</company>
    <price>10.90</price>
    <year>1985</year>
  </cd>
  .
  .
  .
</catalog>
```

XSLT Example

My CD Collection

Title	Artist
Empire Burlesque	Bob Dylan
Hide your heart	Bonnie Tyler
Greatest Hits	Dolly Parton
Still got the blues	Gary Moore
Eros	Eros Ramazzotti
One night only	Bee Gees
Sylvias Mother	Dr.Hook
Maggie May	Rod Stewart
Romanza	Andrea Bocelli

XSLT Online Editor (w3schools)

- <https://www.w3schools.com/xml/tryslt.asp?xmlfile=cdcatalog&xsltfile=cdcatalog>

XSLT Contd.

- XSL can be used to transform one XML to another
- XML to csv, xls....



```
<?xml version="1.0"?>
<investments>
  <item type="stock" exch="nyse"    symbol="ZCXM" company="zacx corp"
        price="28.875"/>
  <item type="stock" exch="nasdaq"   symbol="ZFFX" company="zaffymat inc"
        price="92.250"/>
  <item type="stock" exch="nasdaq"   symbol="ZYSZ" company="zymergy inc"
        price="20.313"/>
</investments>
```

XSLT Contd.

```
<?xml version="1.0"?>
<portfolio>
    <stock exchange="nyse">
        <name>zacx corp</name>
        <symbol>ZCXM</symbol>
        <price>28.875</price>
    </stock>
    <stock exchange="nasdaq">
        <name>zaffymat inc</name>
        <symbol>ZFFX</symbol>
        <price>92.250</price>
    </stock>
    <stock exchange="nasdaq">
        <name>zysmerry inc</name>
        <symbol>ZYSZ</symbol>
        <price>20.313</price>
    </stock>
</portfolio>
```

XSLT Contd.

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="xml" indent="yes"/>
<xsl:template match="/">
  <portfolio>
    <xsl:for-each select="investments/item[@type='stock']">
      <stock>
        <xsl:attribute name="exchange">
          <xsl:value-of select="@exch"/>
        </xsl:attribute>
        <name><xsl:value-of select="@company"/></name>
        <symbol><xsl:value-of select="@symbol"/></symbol>
        <price><xsl:value-of select="@price"/></price>
      </stock>
    </xsl:for-each>
  </portfolio>
</xsl:template>

</xsl:stylesheet>
```

Data Formats

- A “binary” format maps data concepts to the native entities of the computer system, most of the time to bytes.
- A “text” format maps the data concepts to character strings, which must be translated to and from computer entities.
 - For example, to perform arithmetic, the string “-7.0” must be converted to an appropriate representation.
 - One advantage of text representations is that they can (to a degree) be read by a human.
 - XML has emerged as the most popular text format.

XML vs Binary

XML	Binary
'Human readable'	Requires software interpretation
<ul style="list-style-type: none">Single, Universal StandardWeb-friendlyMassive commercial support	Many binary formats, availability varies
<ul style="list-style-type: none">Stored as a stream of Unicode, in a single fileOrganized as a treeCannot directly store 'native' numbers, must be encoded as unicode	<p>Many organizations,</p> <ul style="list-style-type: none">store numbers in 'native' formats which are typically compact and require no translation in order to perform transfer

XML vs Binary

- XML advantages over Binary
 - XML is self describing and also platform neutral
 -
 -
 -
 -
- Binary advantages over XML
 - Size: Smaller data size, minimal overhead size (ie: no tags)
 - Speed: No need to convert between text and numeric data
 - Simpler: Parsing XML is a complicated exercise
 - And XML can be *too* flexible: multiple ways to do same thing
 -
 -
 -
 -

XML and Binary Data Transfer

- Formats (encoding) for network protocols and data transfer are traditionally binary
 - eg: JRMP (Java)
- XML has been used to define a format for data transmission entirely in text using XML tags
 - eg: SOAP: basis of Web services

Serialization

- Serialization is the problem of converting/encoding an object into a single, transportable stream
 - ...including any aggregated (contained) objects
 - Used for saving to disk (persistency), network transfer, etc
- Deserialization is the process of (re)creating an object from a serialized string
- Binary serialization
 - Common, but suffers from aforementioned problems
- XML serialization
 - Convert object to a single textual XML description

JSON

Overview

- What is JSON?
- Comparisons with XML
- Syntax
- Data Types
- Usage



JSON is...

- A lightweight text based data-interchange format
- Completely language independent
- Based on a subset of the JavaScript Programming Language
- Easy to understand, manipulate and generate

JSON is NOT...

- Overly Complex
- A “document” format
- A markup language
- A programming language



Why use JSON?



- Straightforward syntax
- Easy to create and manipulate
- Can be natively parsed in JavaScript using **eval()**
- Supported by all major JavaScript frameworks
- Supported by most backend technologies

JSON vs. XML



Much Like XML

- Plain text formats
- “Self-describing” (human readable)
- Hierarchical (Values can contain lists of objects or values)



Not Like XML

- Lighter and faster than XML
- JSON uses typed objects. All XML values are type-less strings and must be parsed at runtime.
- Less syntax, no semantics
- Properties are immediately accessible to JavaScript code

Knocks against JSON

- Lack of namespaces
- No inherit validation (XML has DTD and Schemas, but there is JSONlint for checking syntax)
- Not extensible
- It's basically just *not* XML

Syntax

JSON Object Syntax

- Unordered sets of name/value pairs
- Begins with { (left brace)
- Ends with } (right brace)
- Each name is followed by : (colon)
- Name/value pairs are separated by , (comma)

JSON Example

```
var employeeData = {  
    "employee_id": 1234567,  
    "name": "Jeff Fox",  
    "hire_date": "1/1/2013",  
    "location": "Norwalk, CT",  
    "consultant": false  
};
```

Arrays in JSON

- An ordered collection of values
- Begins with **[** (left bracket)
- Ends with **]** (right bracket)
- Name/value pairs are separated by **,** (comma)

JSON Array Example

```
var employeeData = {  
    "employee_id": 1236937,  
    "name": "Jeff Fox",  
    "hire_date": "1/1/2013",  
    "location": "Norwalk, CT",  
    "consultant": false,  
    "random_nums": [ 24, 65, 12, 94 ]  
};
```

Data Types

Data Types: Strings

- Sequence of 0 or more Unicode characters
- Wrapped in "double quotes"
- Backslash escapement

Data Types: Numbers

- Integer
- Real
- Scientific
- No octal or hex
- No NaN or Infinity – Use **null** instead.

Data Types: Booleans & Null

- Booleans: true or false
- Null: A value that specifies nothing or no value.

Data Types: Objects & Arrays

- Objects: Unordered key/value pairs wrapped in { }
- Arrays: Ordered key/value pairs wrapped in []
- **Can nest objects (supports complex objects)**

Nested Objects

```
{  
    "stuff": {  
        "onetype": [  
            {"id":1,"name":"John Doe"},  
            {"id":2,"name":"Don Joeh"}  
        ],  
        "othertype": {"id":2,"company":"ACME"}  
    },  
    "otherstuff": {  
        "thing": [[1,42],[2,2]]  
    }  
}
```

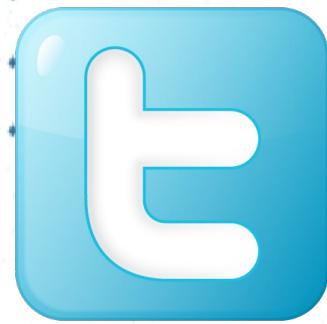
JSON Usage

How & When to use JSON

- Transfer data to and from a server
- Perform asynchronous data calls without requiring a page refresh
- Working with data stores
- Compile and save form or user data for local storage

Where is JSON used today?

- Anywhere and everywhere!



And many,
many more!

Other open data/message formats

- YAML (yet another markup language)
- <https://blog.stackpath.com/yaml/>
- <https://octopus.com/blog/state-of-config-file-formats>
- Often used to define the interfaces of REST apis (will discuss later)
- Defining component/service interfaces is one key usage of open data formats (e.g. XML, YAML)

Summary

- **Binary** data formats (e.g. RMI object-streams) are more **efficient** in sending and receiving messages
- However, **Textual** data formats like XML and JSON are **Open** and **Standardized**, so they facilitate **Interoperability**.
- **XML** has rich set of features such as **namespaces** and **Schemas**.
- **JSON** is more **lightweight** and **efficient**, yet lack Schemas and namespaces
 - YAML is another example for an open data format
 - One key usage of Open data formats is to define service interfaces (e.g. XML, YAML)

Questions?