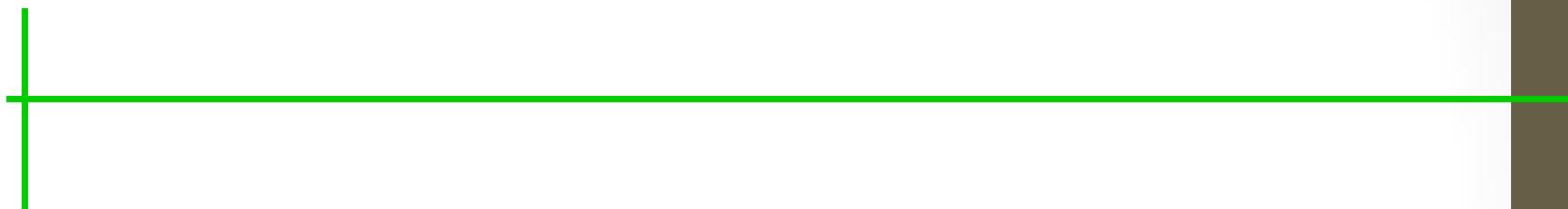


Lecture 1

Introduction to Distributed Systems



Presentation Outline

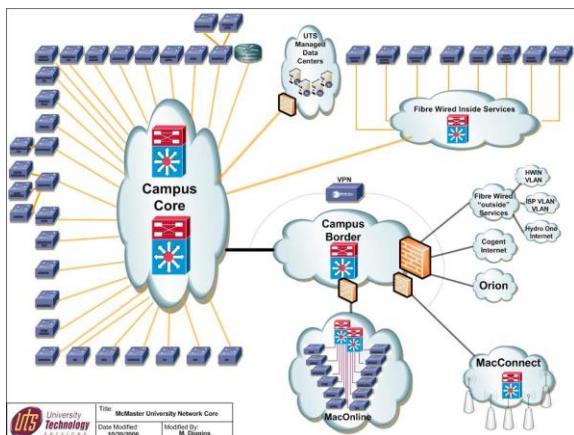
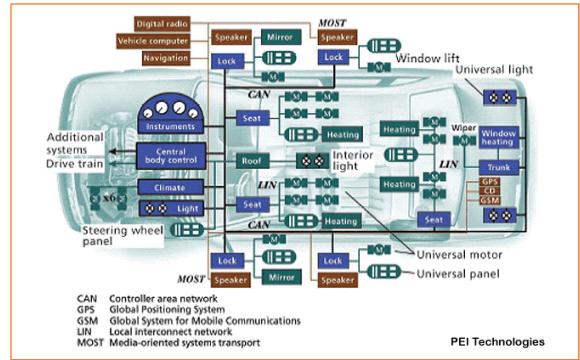
- Introduction
- Defining Distributed Systems
- Characteristics of Distributed Systems
- Example Distributed Systems
- Challenges of Distributed Systems
- Summary

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

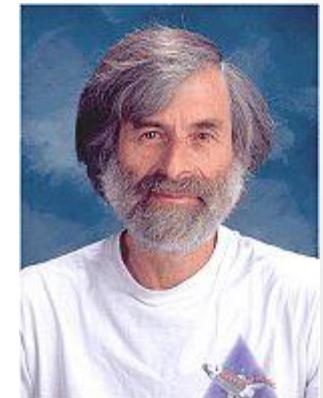


Defining Distributed Systems

- “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

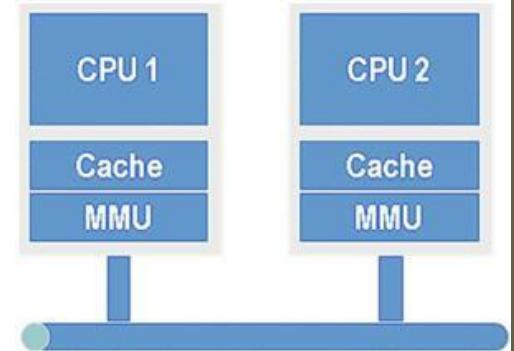
- Concurrency
 - Carry out tasks independently and parallelly
 - Tasks coordinate their actions by exchanging messages
- Communication via message passing
 - No shared memory
- Resource sharing
 - Printer, database, other services
- No global state
 - No single process can have knowledge of the current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

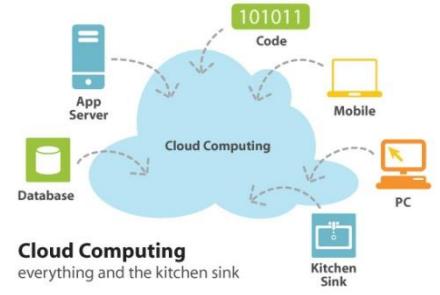


Differentiation with parallel systems is blurring

- 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

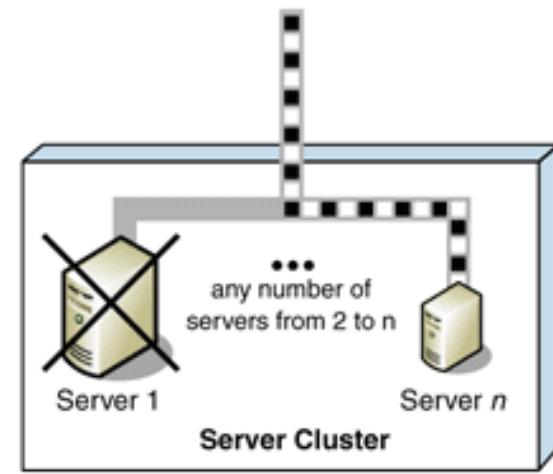
- Failure transparency
 - Tasks can be completed despite failures
 - E.g. message retransmission, failure of a Web server node should not bring down the website
- Replication transparency
 - Access to replicated resources as if there was just one. And provide enhanced reliability and performance without 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

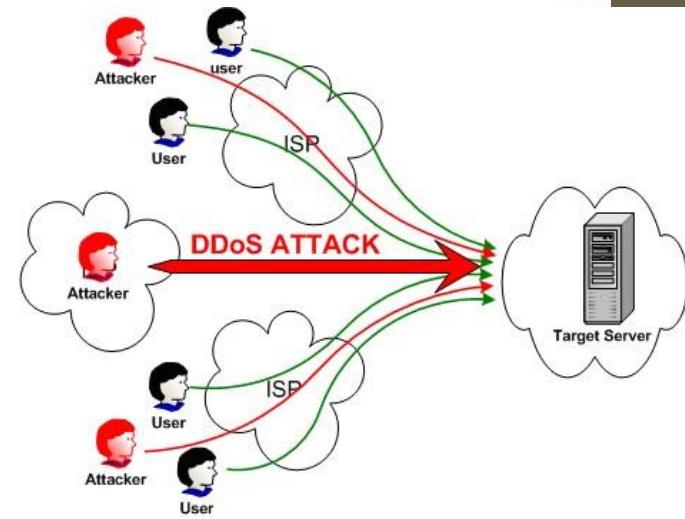
- Open message formats: e.g. XML
- Open communication protocols: e.g. HTTP, HTTPS
- Open spec/standard developers - communities:
 - ANSI, IETF, W3C, ISO, IEEE, OMG, Trade associations,...

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

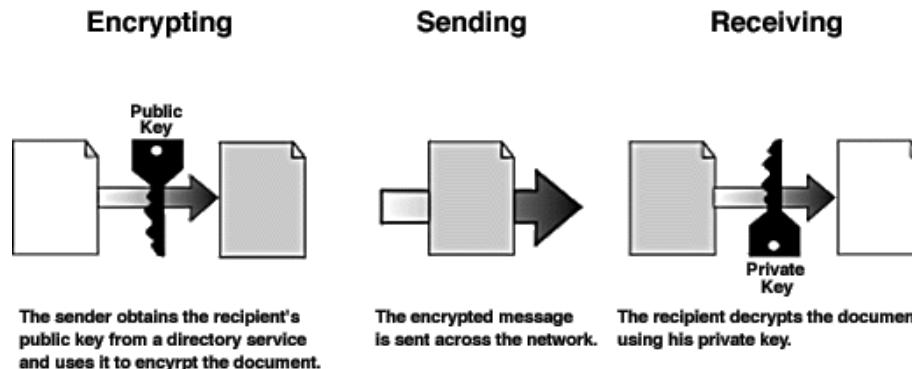
Security II

- 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
 - Purchas 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

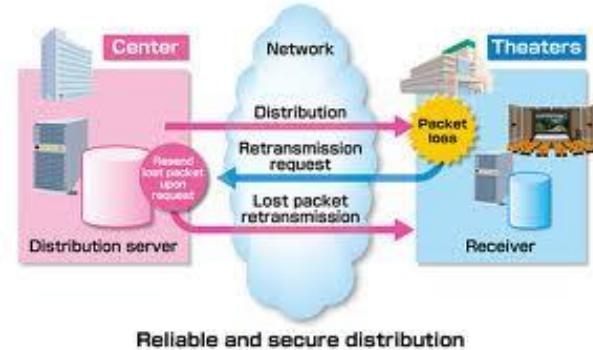
II

- 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

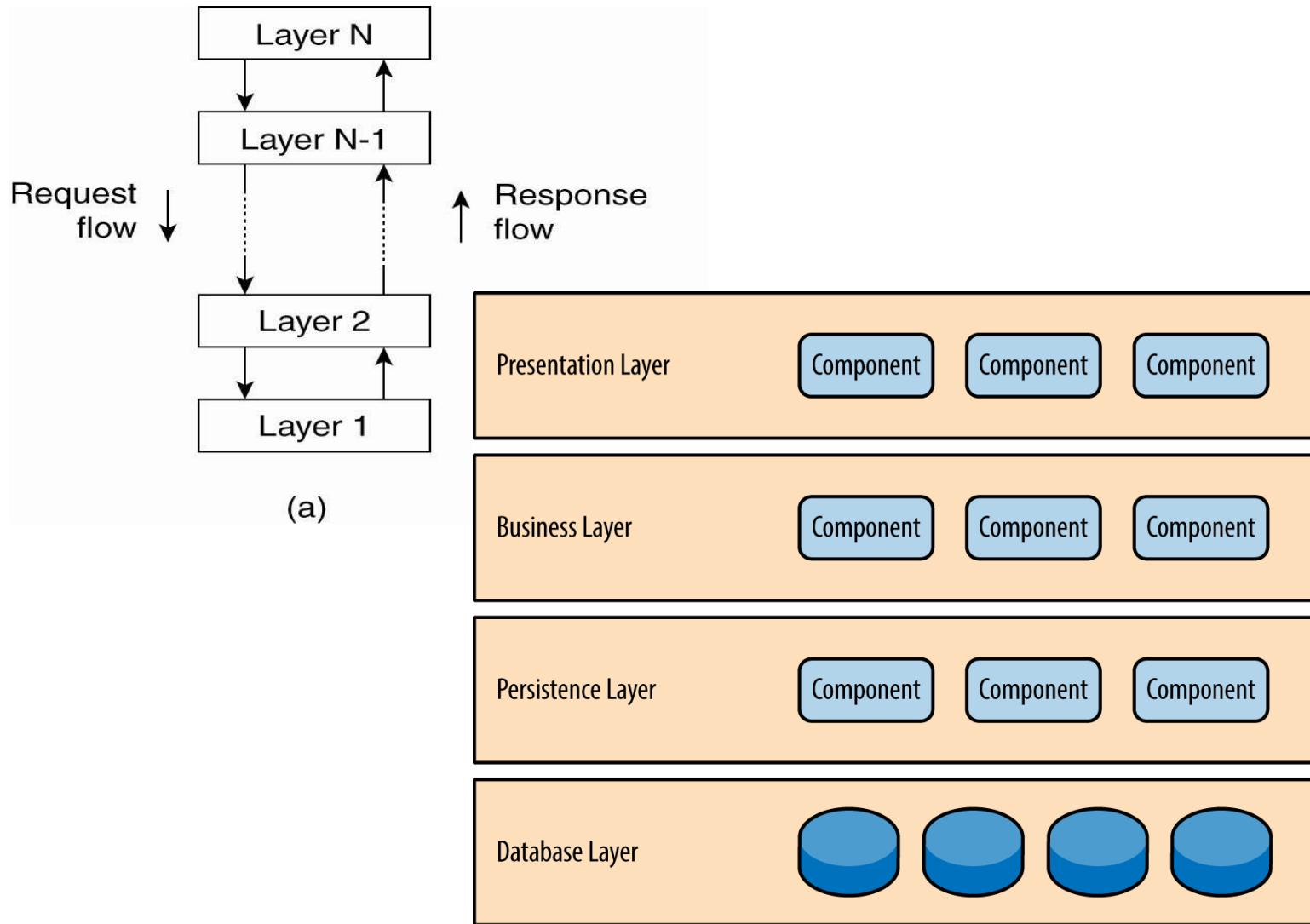
Architectural Styles

- 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

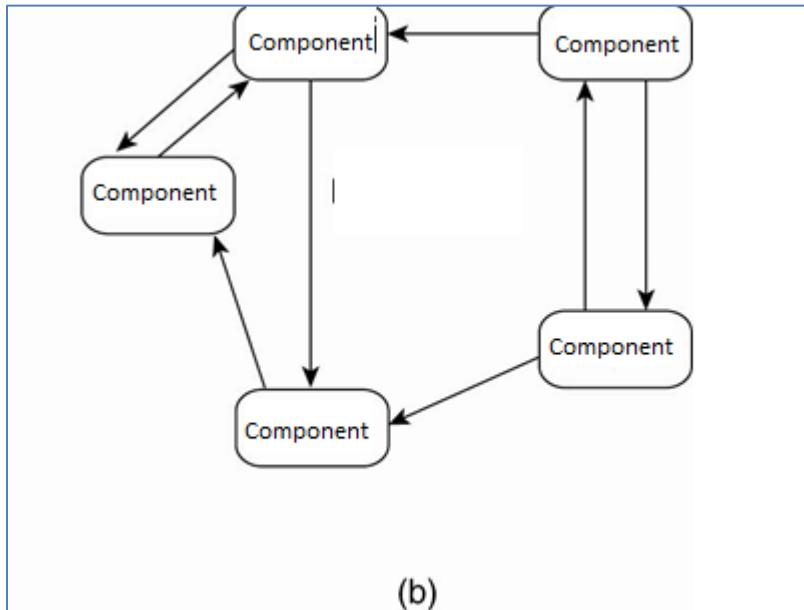
Layered Architecture

- 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

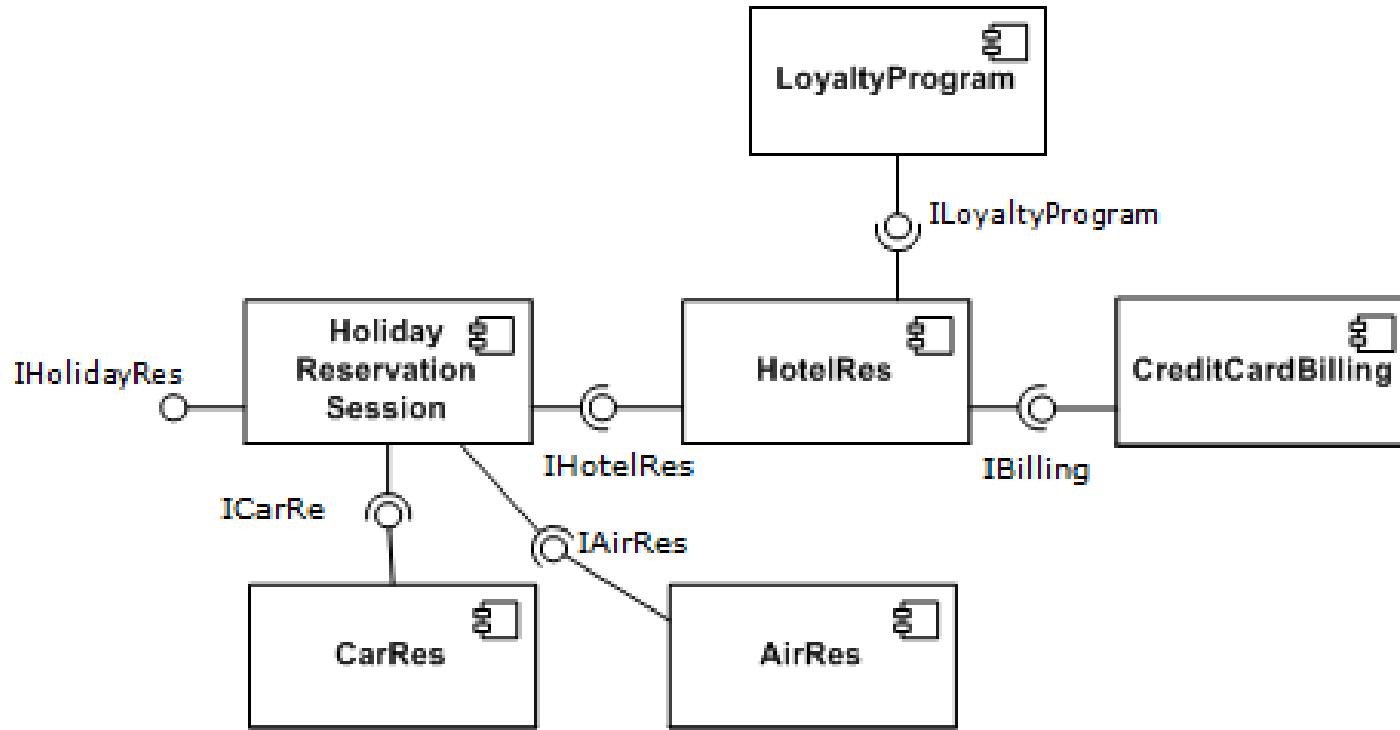


Component based architecture



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

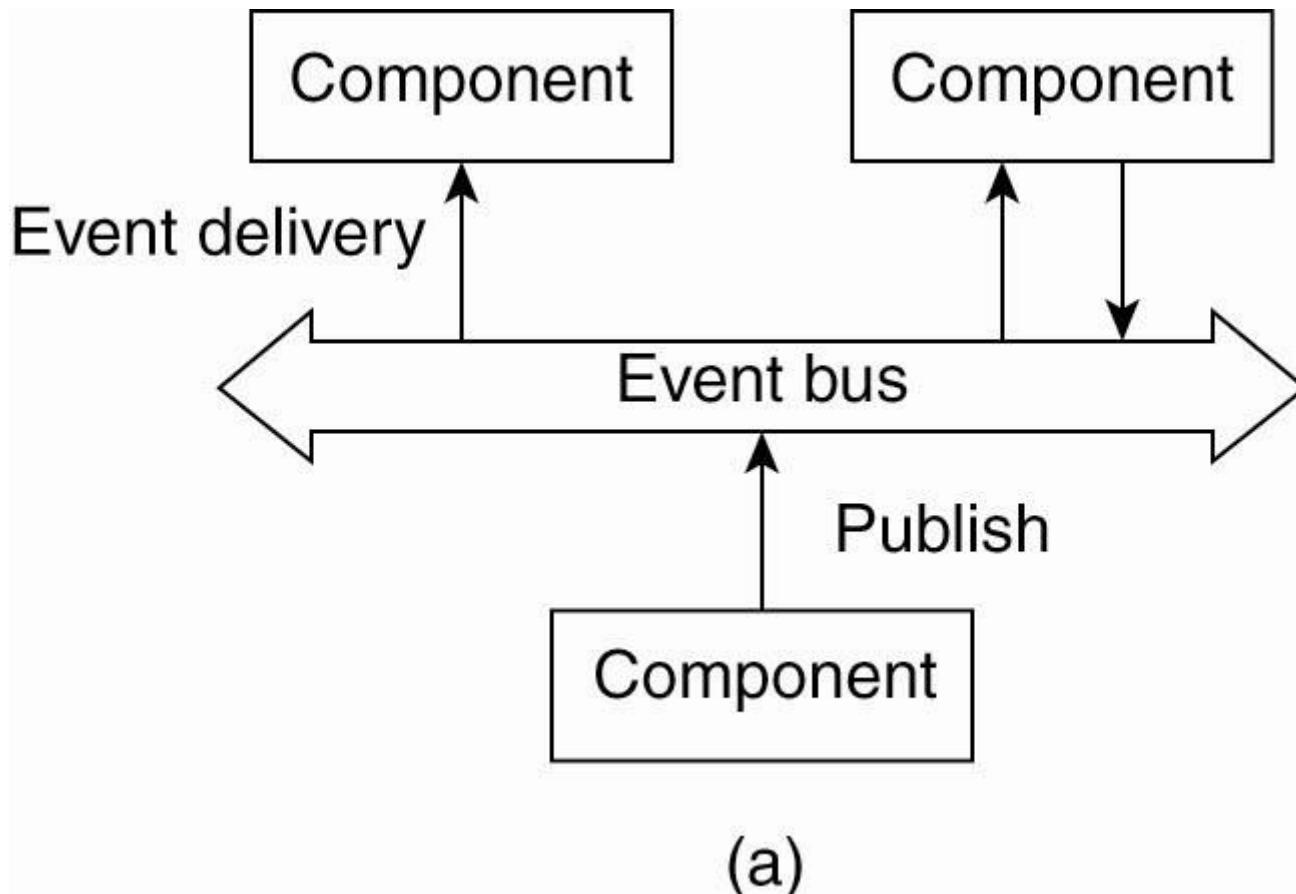
Component based architecture



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



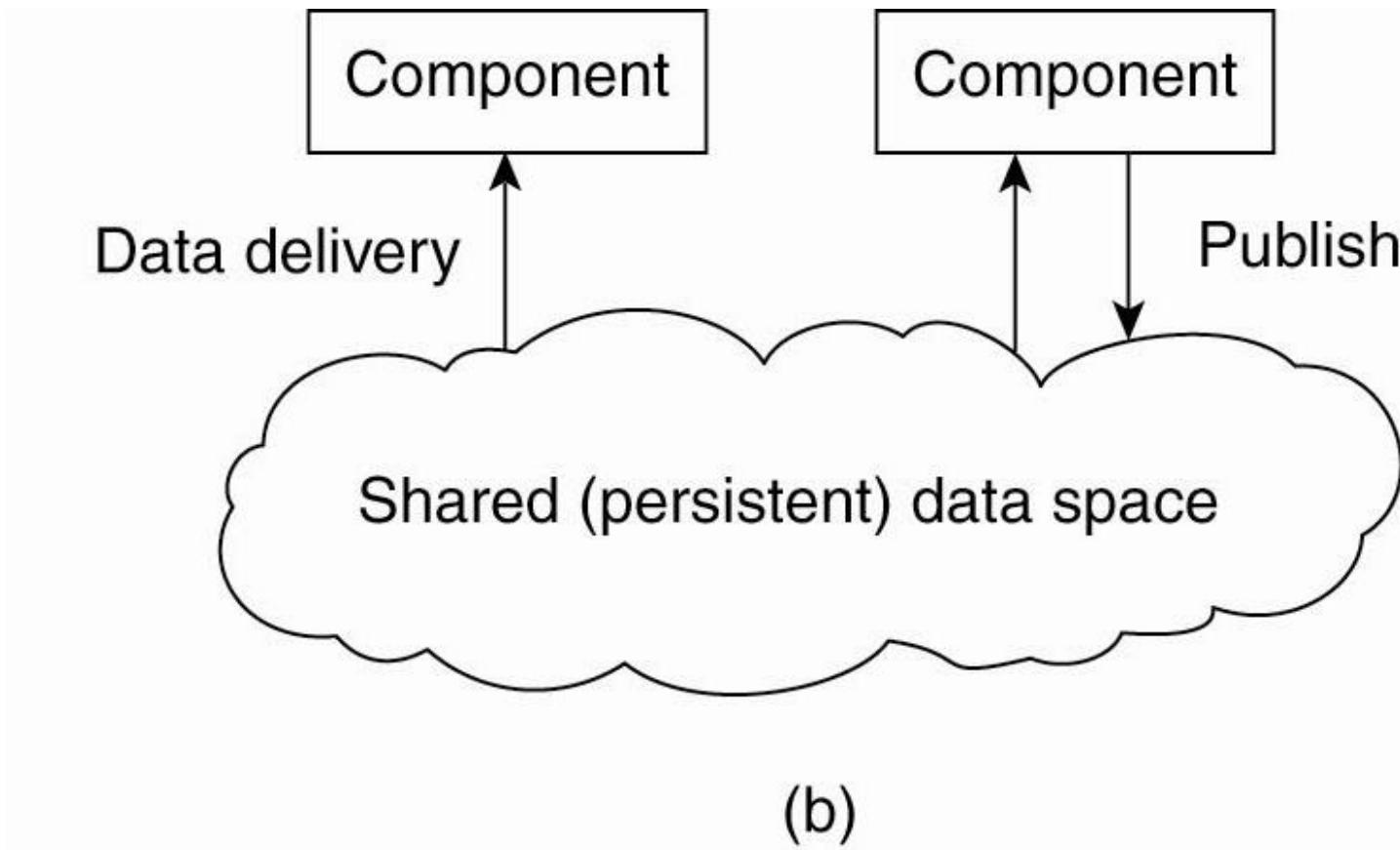
Event based Architectures

- 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 Architecture?

- 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

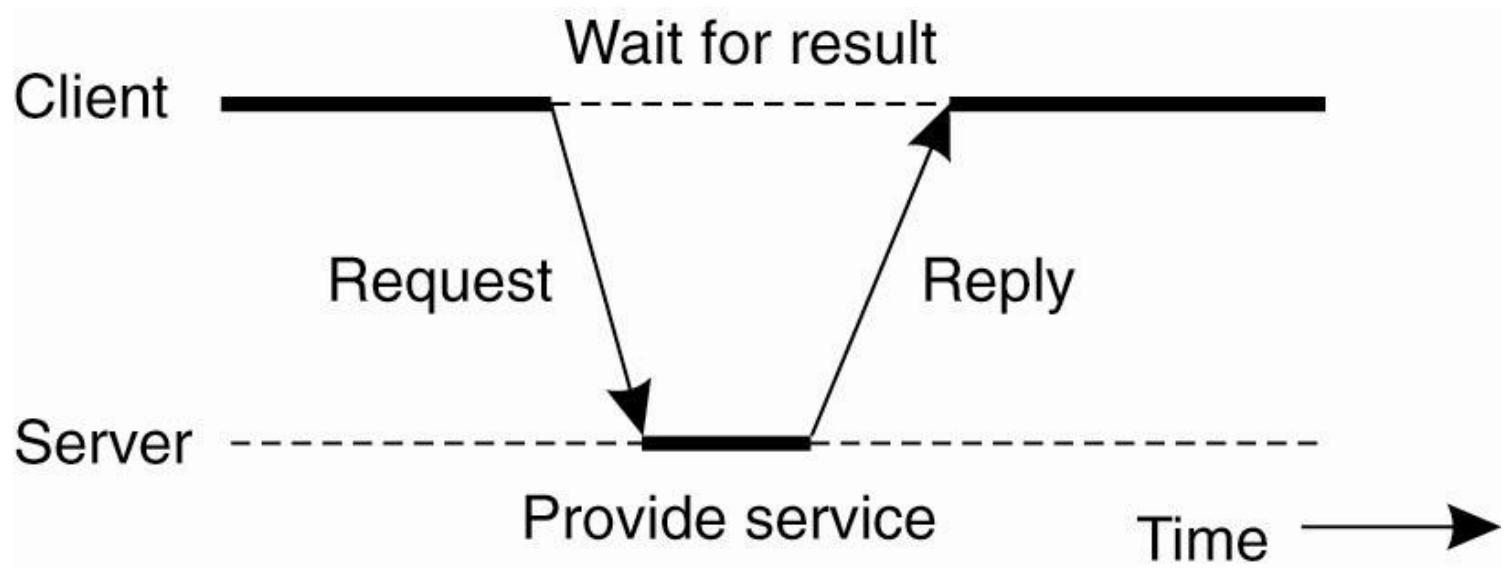


Figure 2-3. General interaction between a client and a server.

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 work load 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)

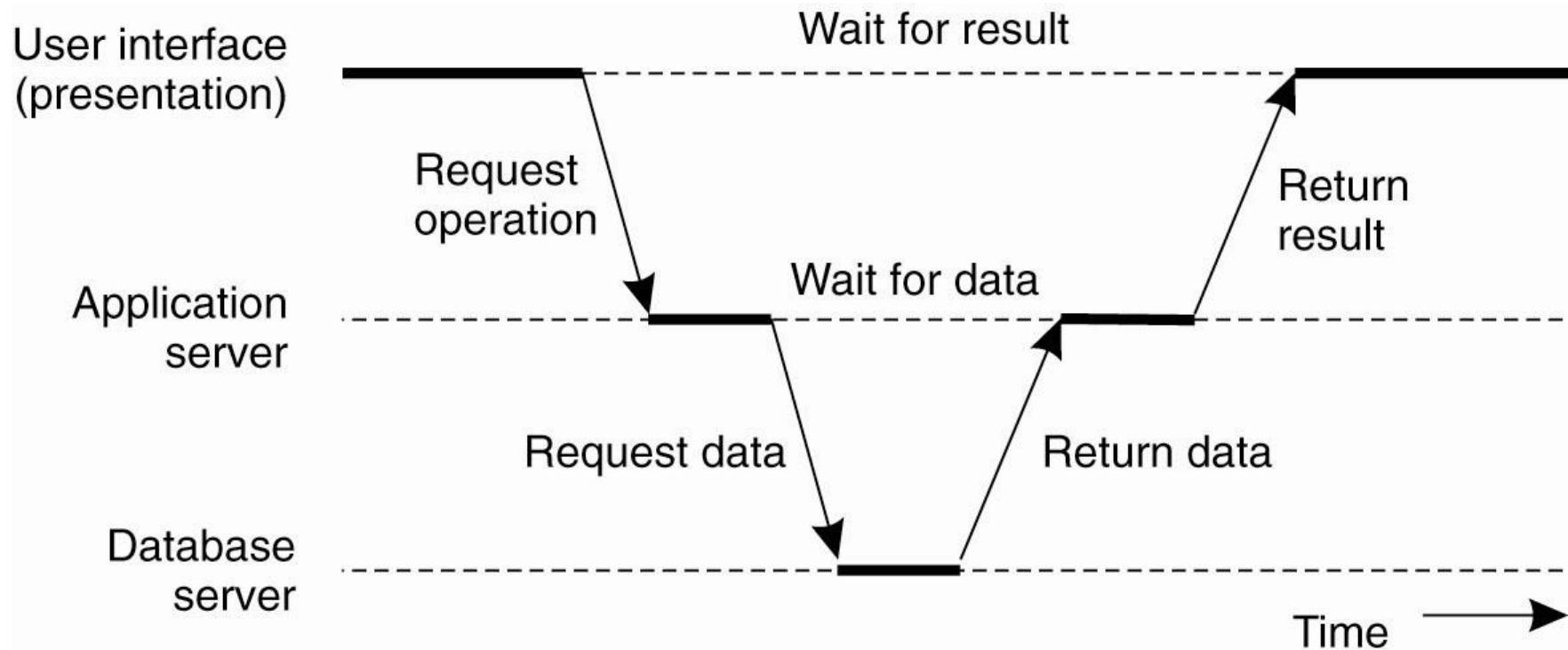


Figure 2-6. An example of a server acting as client.

Multitiered Architectures

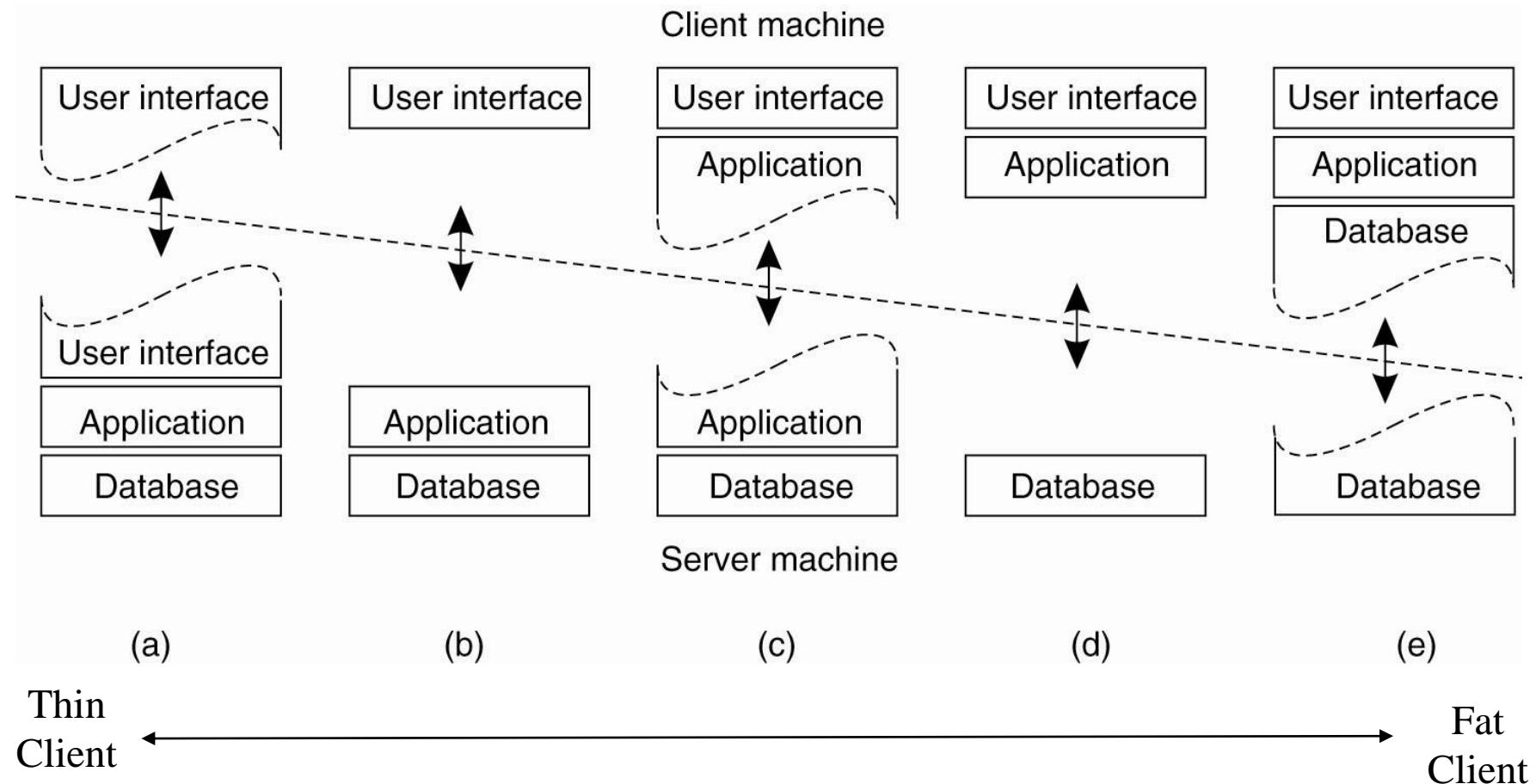


Figure 2-5. Alternative client-server organizations (a)–(e).

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

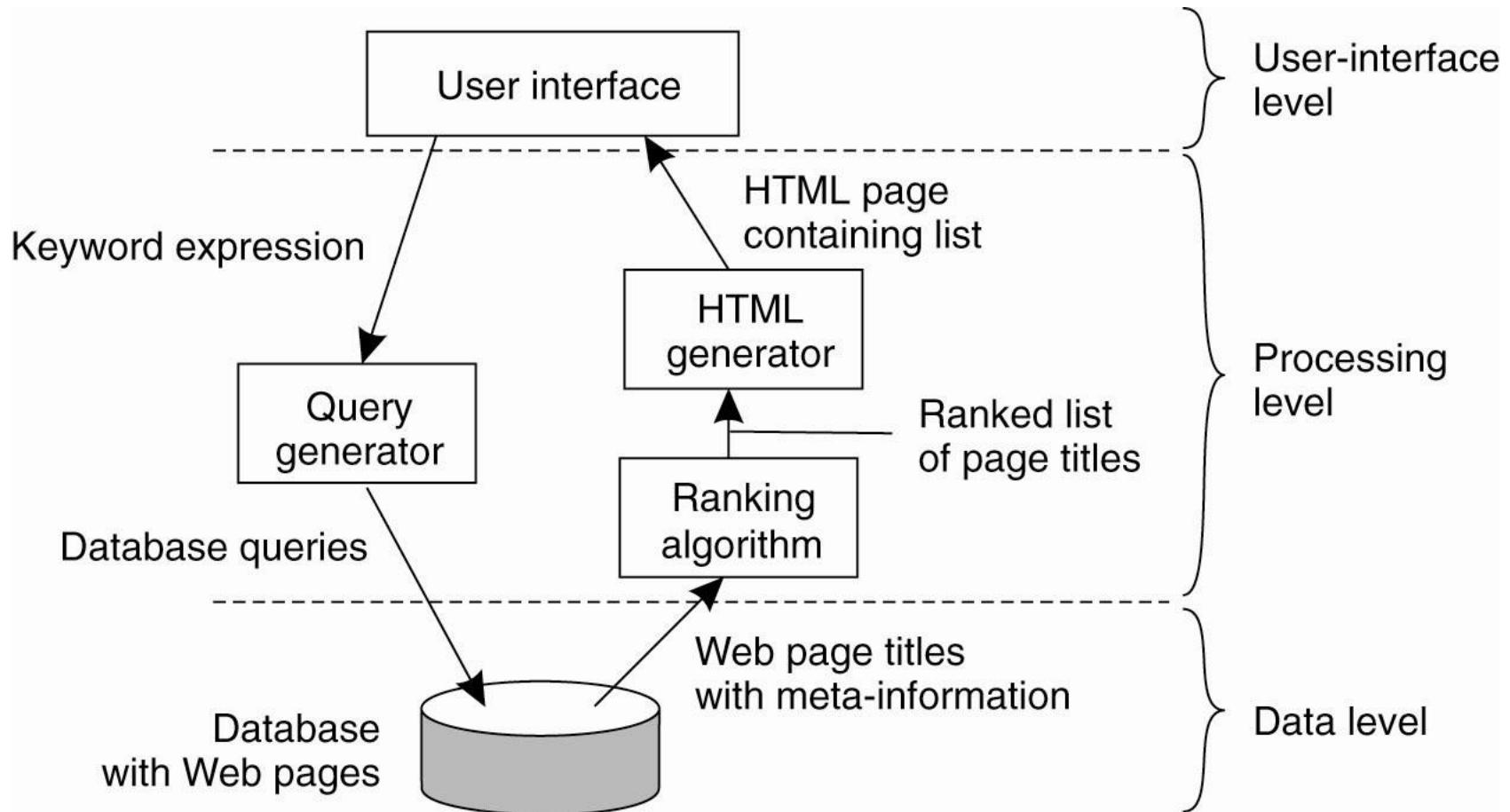
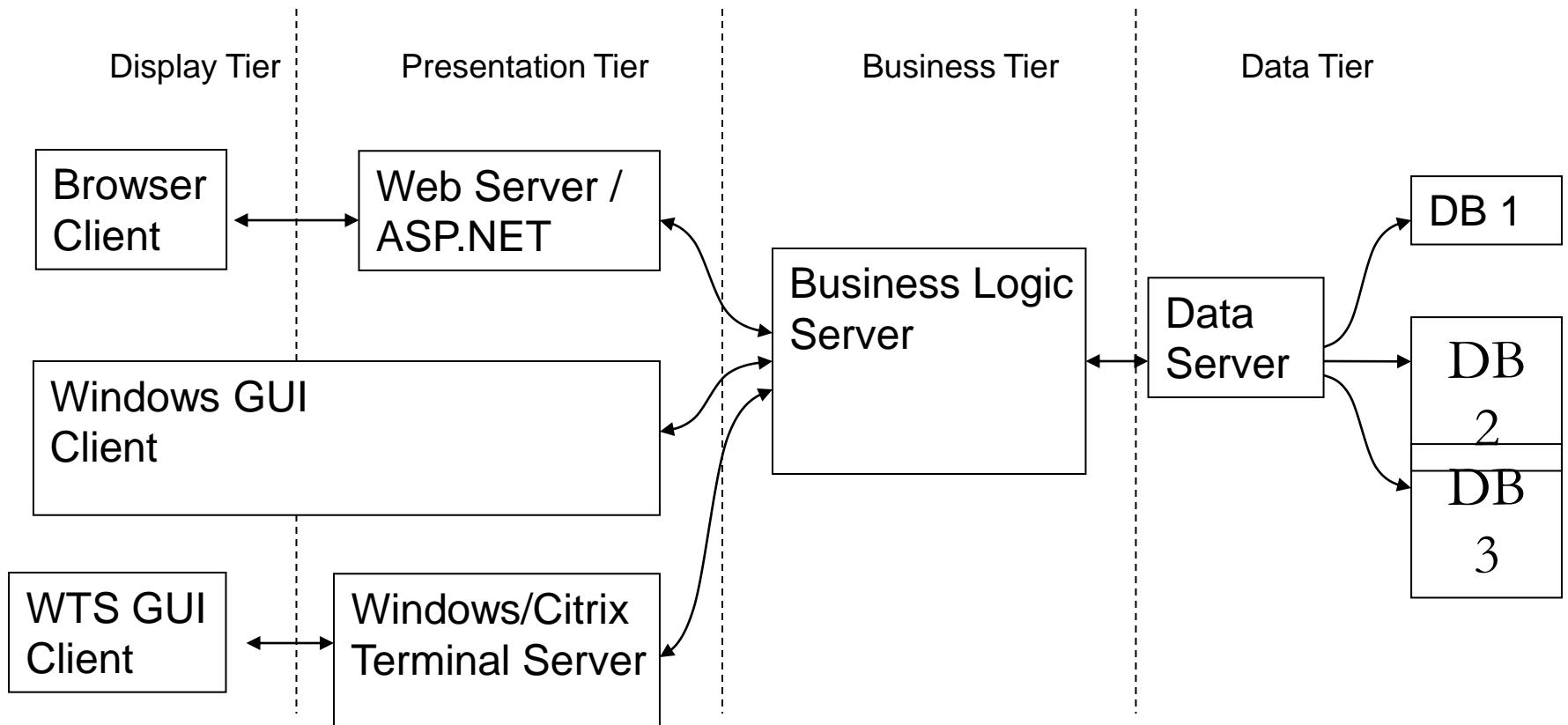


Figure 2-4. The simplified organization of an Internet search engine into three different layers.

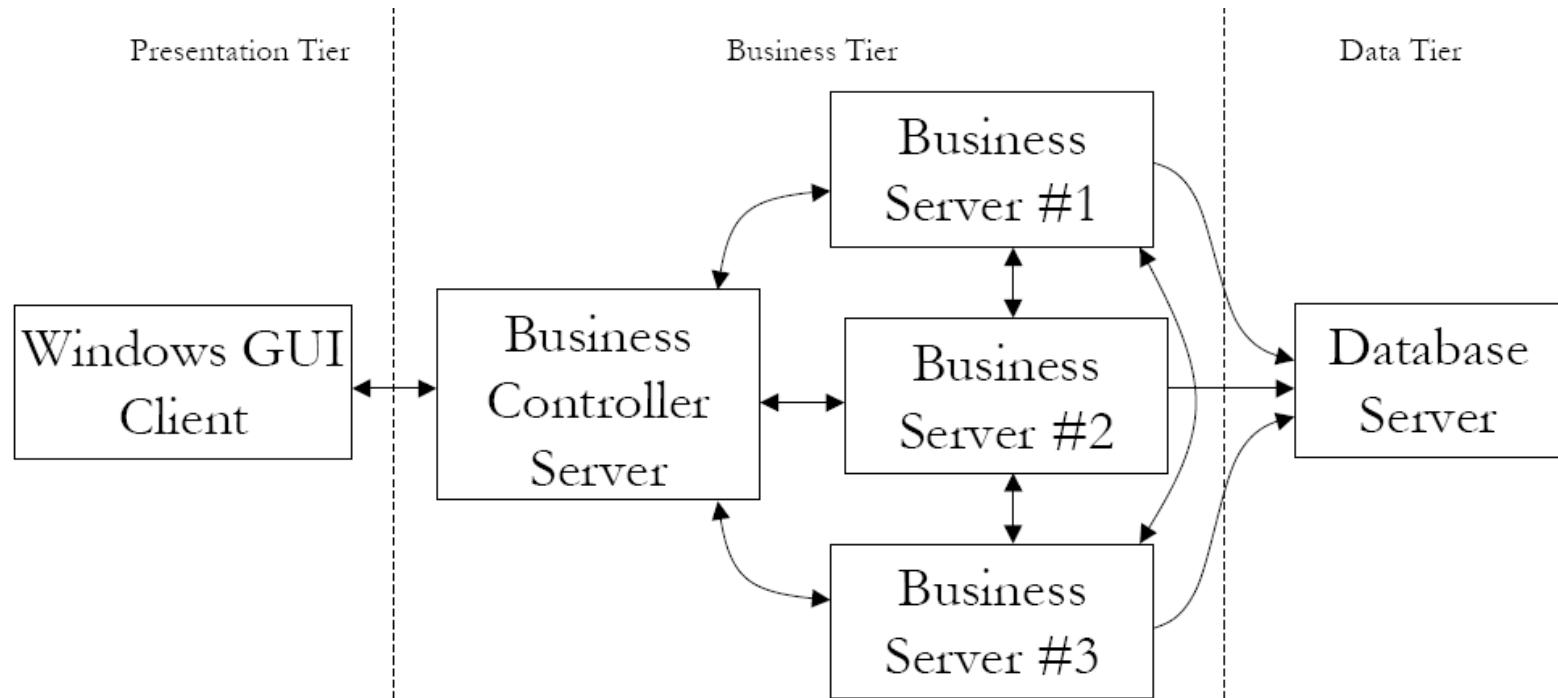
Flexible/Scalable Architecture



Distributing the Business Tier

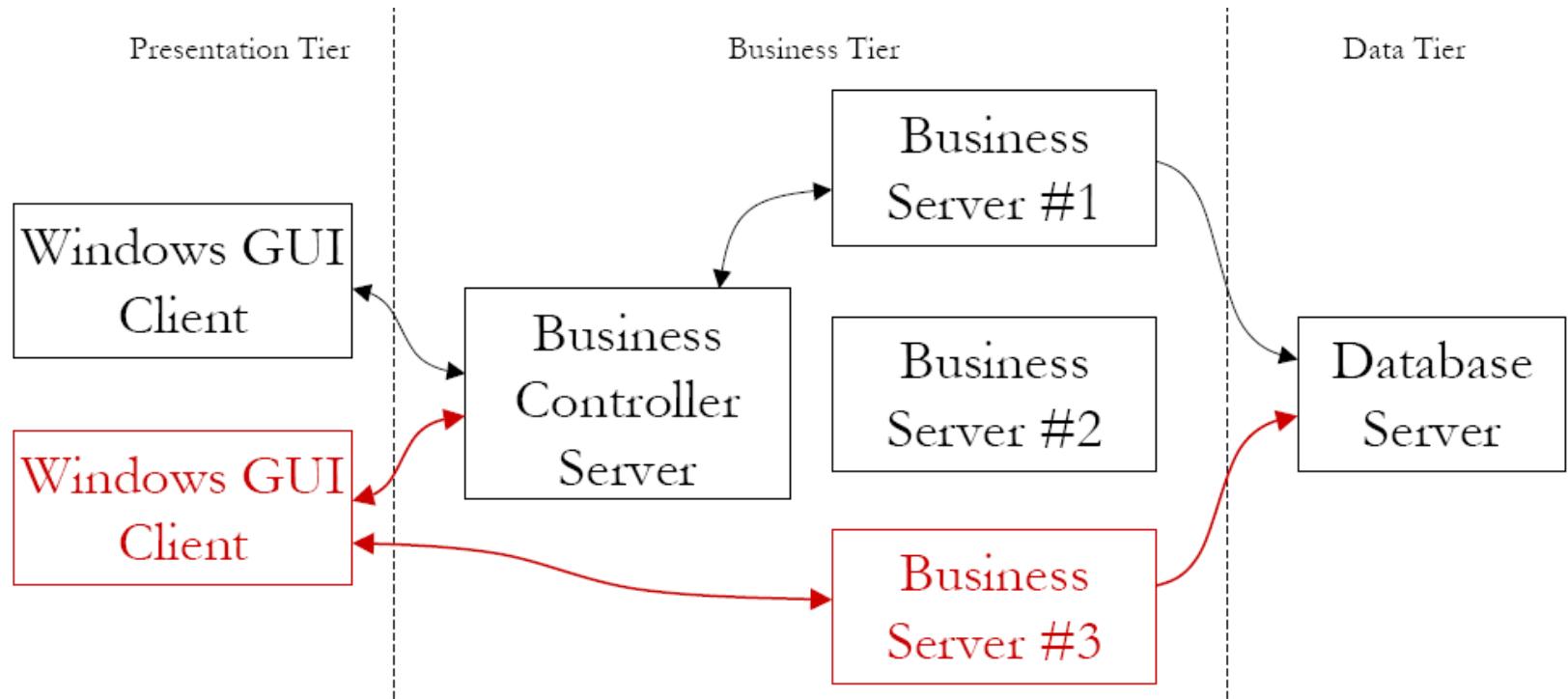
- Business tier could be distributed for various reasons:
 - Parallel computing - split one job among many servers
 - Load balancing - have a controller component redirect presentation clients to a least-utilized business tier server
 - Fault tolerance - robustness to system faults or data faults

Parallel Computing Architecture



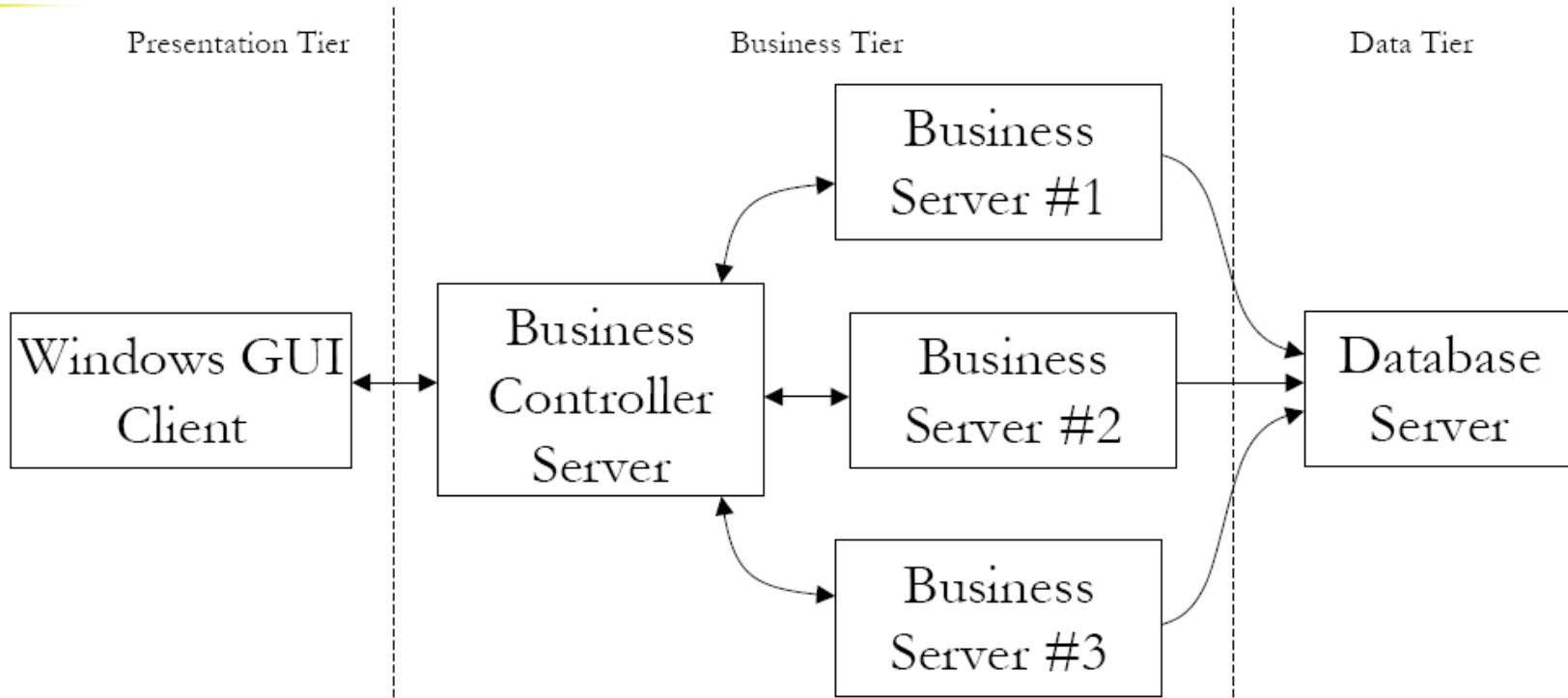
- Controller passes job to server farm, returns response
 - Business servers collaborate via data sharing

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 v Client/Server

- P2P computing allows end users to communicate without a dedicated server.
- 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 be more fault tolerant, more resistant to denial of service attacks because network content is distributed.
 - Individual hosts may be unreliable, but overall, the system should maintain a consistent level of service

P2P

- Structured – E.g. Chord
- Unstructured – E.g. BitTorrent

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.

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

Superpeers

- Maintain indexes to some or all nodes in the system
- Supports resource discovery
- Act as servers to regular peer nodes, peers to other superpeers
- Improve scalability by controlling floods
- Can also monitor state of network
- Example: Napster

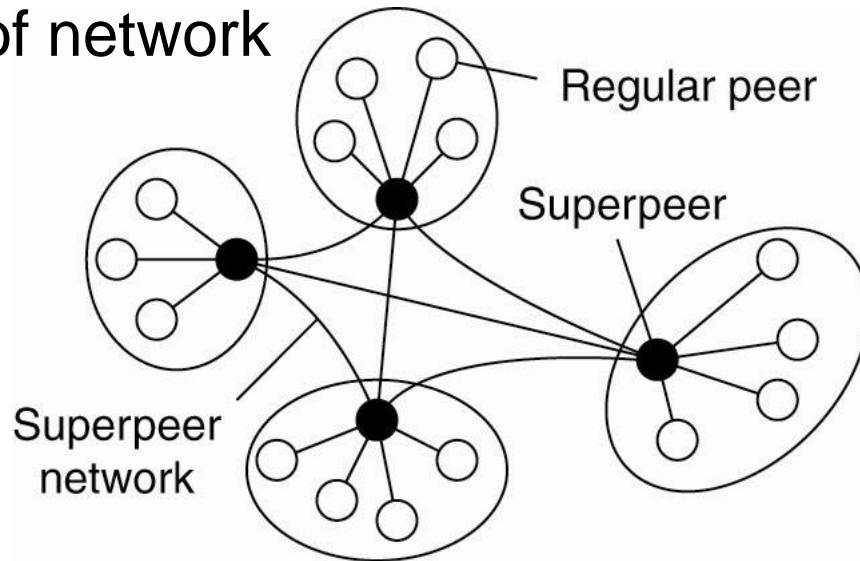


Figure 2-12.

Distributed Hash Tables

- A fully decentralized routing mechanism

[https://www.youtube.com/watch?v=-
UU_ugiPZ9k](https://www.youtube.com/watch?v=-UU_ugiPZ9k)

Possibilities with P2P?

- Currently mostly used for file sharing
- Blockchain uses P2P
- Online Social networks?
- Taxi applications?
- Etc etc

Centralized v Decentralized Architectures

- Traditional client-server architectures exhibit **vertical distribution**. Each tier serves a different purpose in the system.
 - *Logically* different components reside on different nodes
- **Horizontal distribution (P2P)**: 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

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 object 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

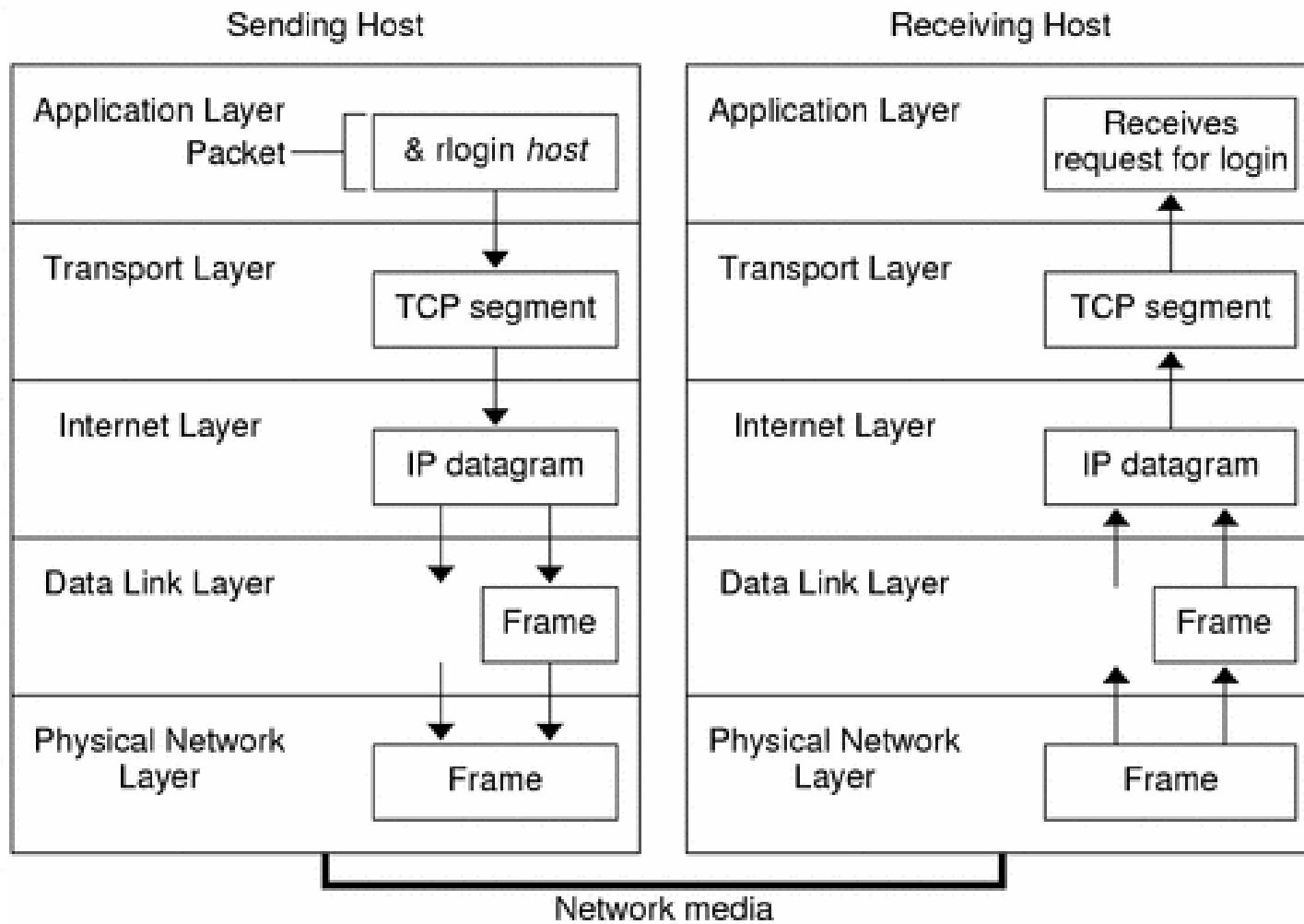
Lecture 3

Introduction to Socket Programming

What is a socket?

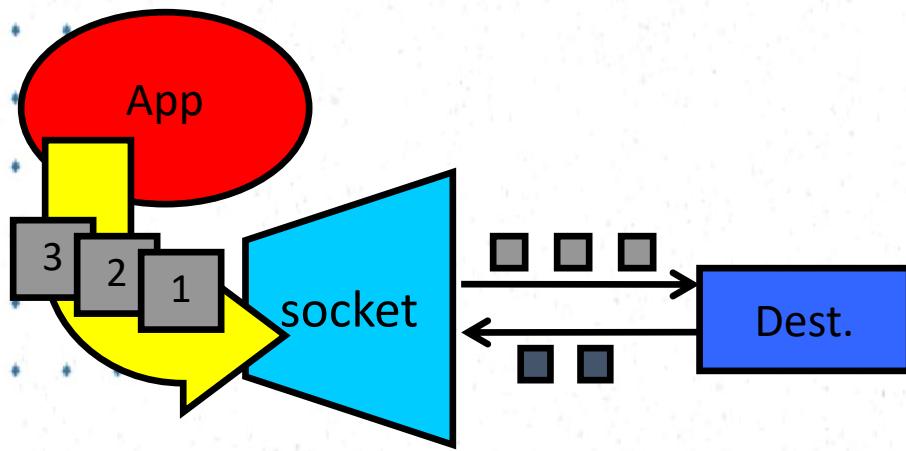
- Port vs. Socket
- An interface between application and network
 - The application creates a socket
 - The socket *type* dictates the style of communication
 - reliable vs. best effort
 - connection-oriented vs. connectionless
- Once configured the application can
 - pass data to the socket for network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

TCP/IP Stack

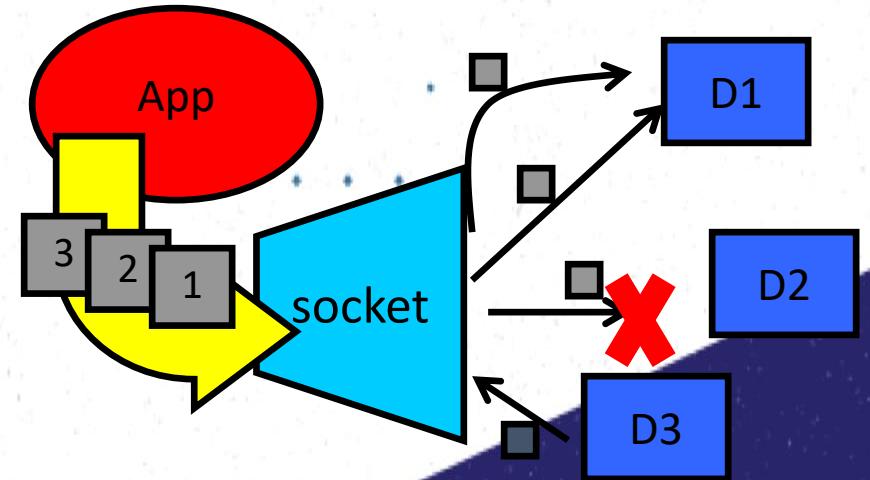


Two essential types of sockets

- 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

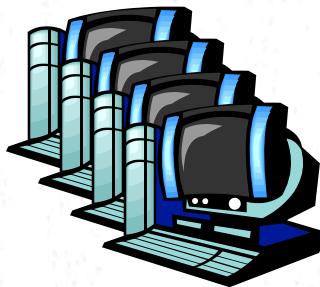
A Socket-eye view of the Internet



medellin.cs.columbia.edu
(128.59.21.14)



newworld.cs.umass.edu
(128.119.245.93)

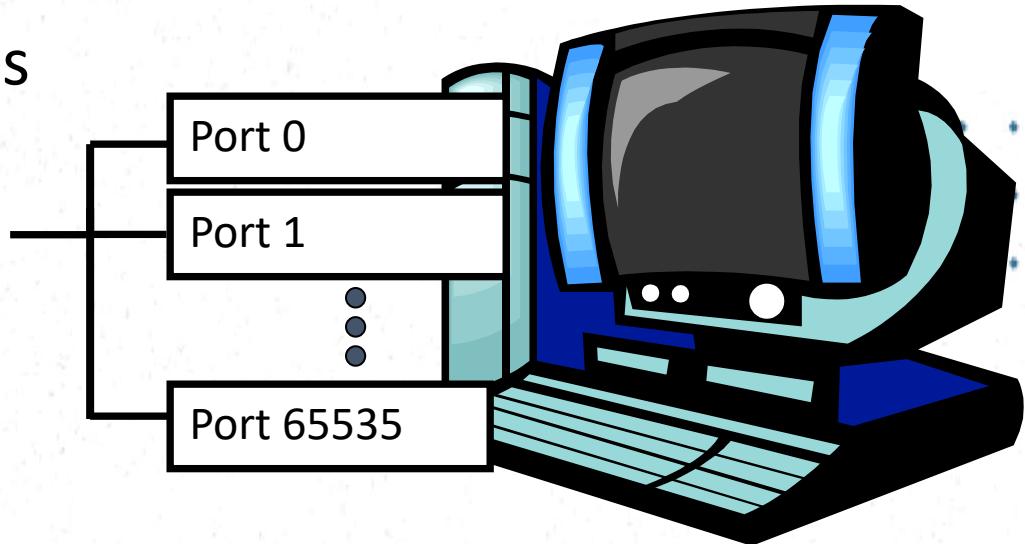


cluster.cs.columbia.edu
(128.59.21.14, 128.59.16.7,
128.59.16.5, 128.59.16.4)

- 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)

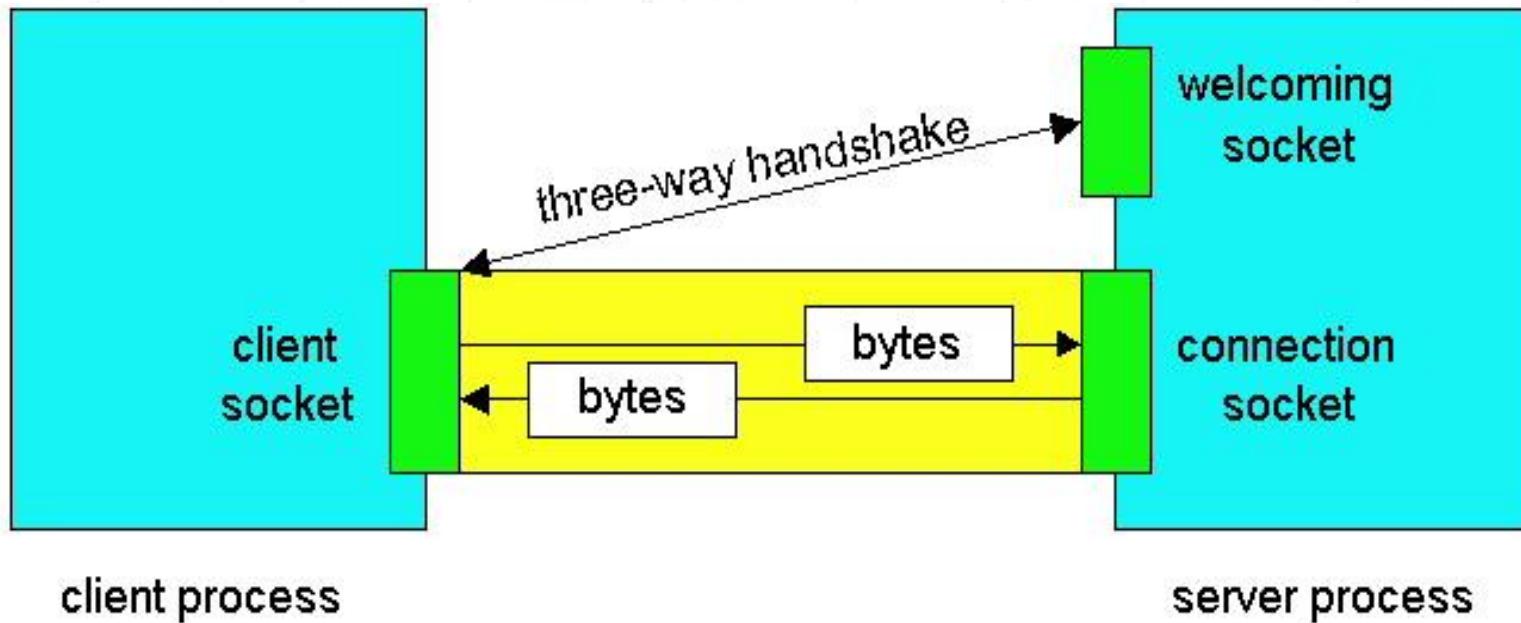


- A socket provides an interface to send data to/from the network through a port

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

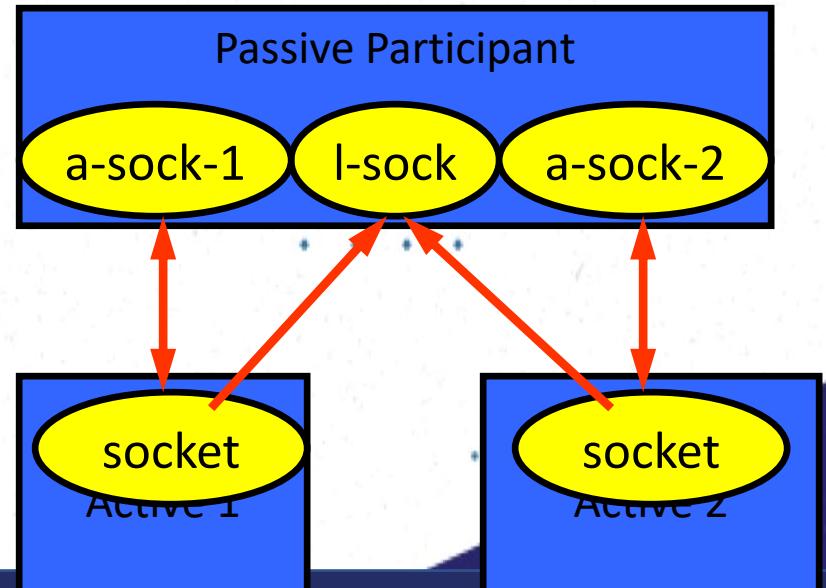


Client socket, welcoming socket (passive) and connection socket (active)

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);
```

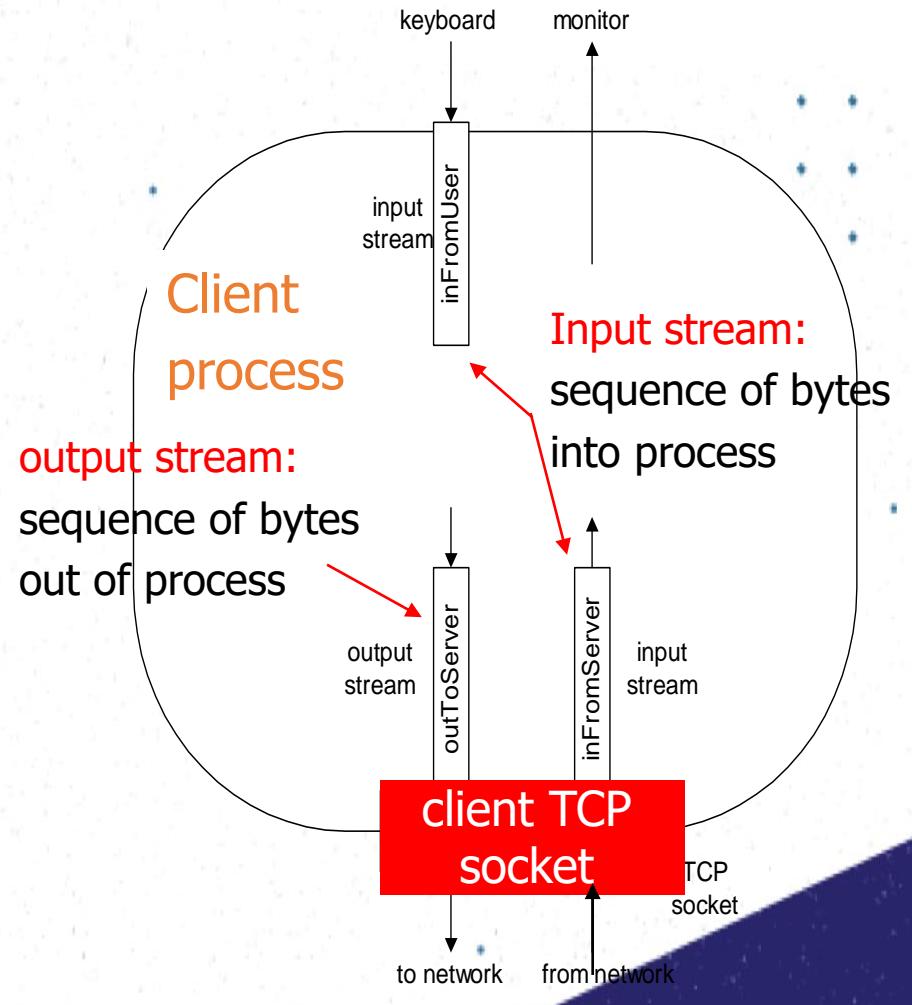
ServerSocket Methods

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

Socket programming with TCP

Example client-server app:

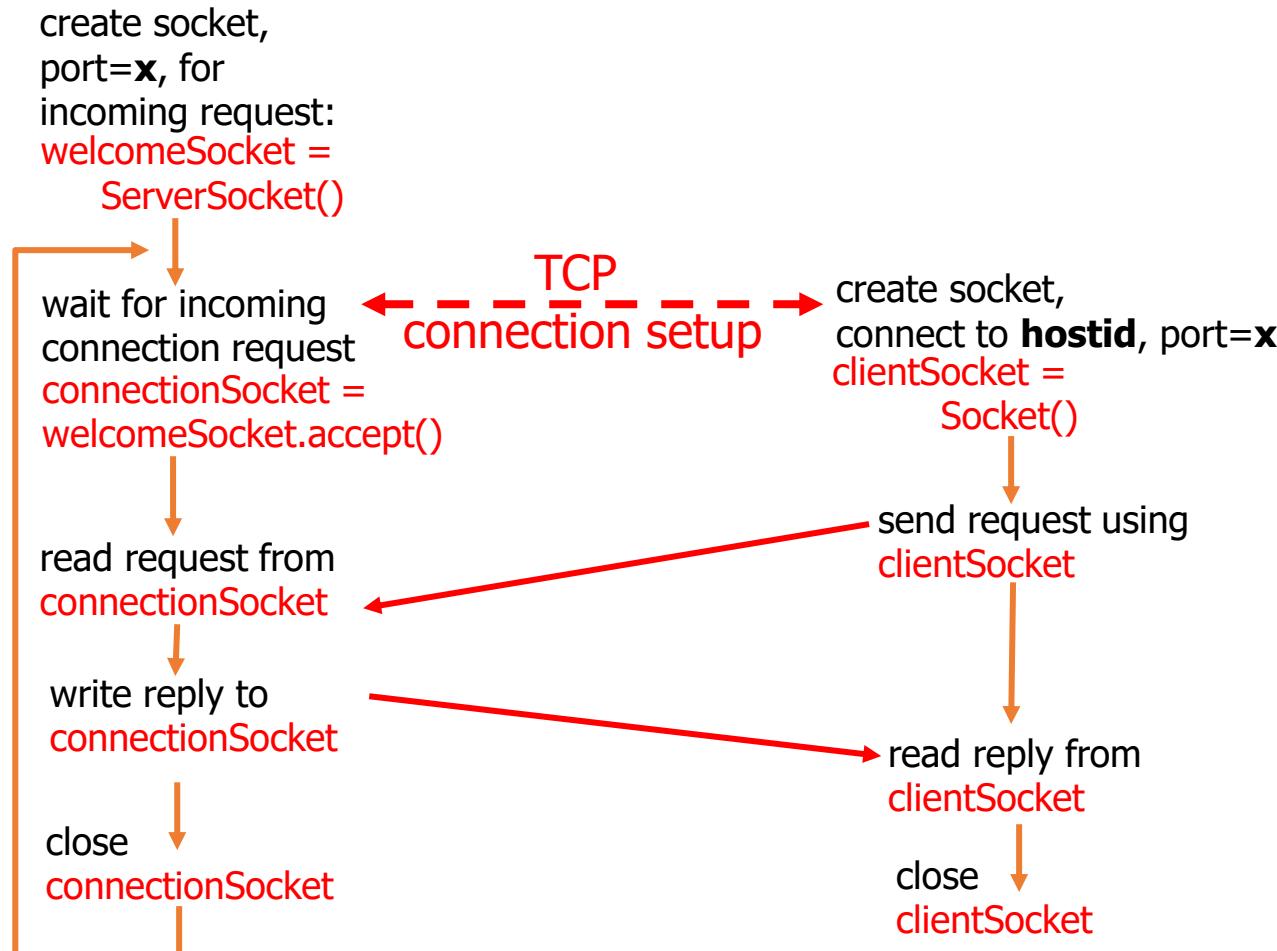
- client reads line from standard input (**inFromUser** stream), sends to server via socket (**outToServer** stream)
- server reads line from socket
- server converts line to uppercase, sends back to client
- client reads, prints modified line from socket (**inFromServer** stream)



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());
        BufferedReader inFromServer = 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()));
            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 host creates “packets” by attaching the IP destination address and port number to each batch of bytes.
 - The receiving process must unravel the 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);
```

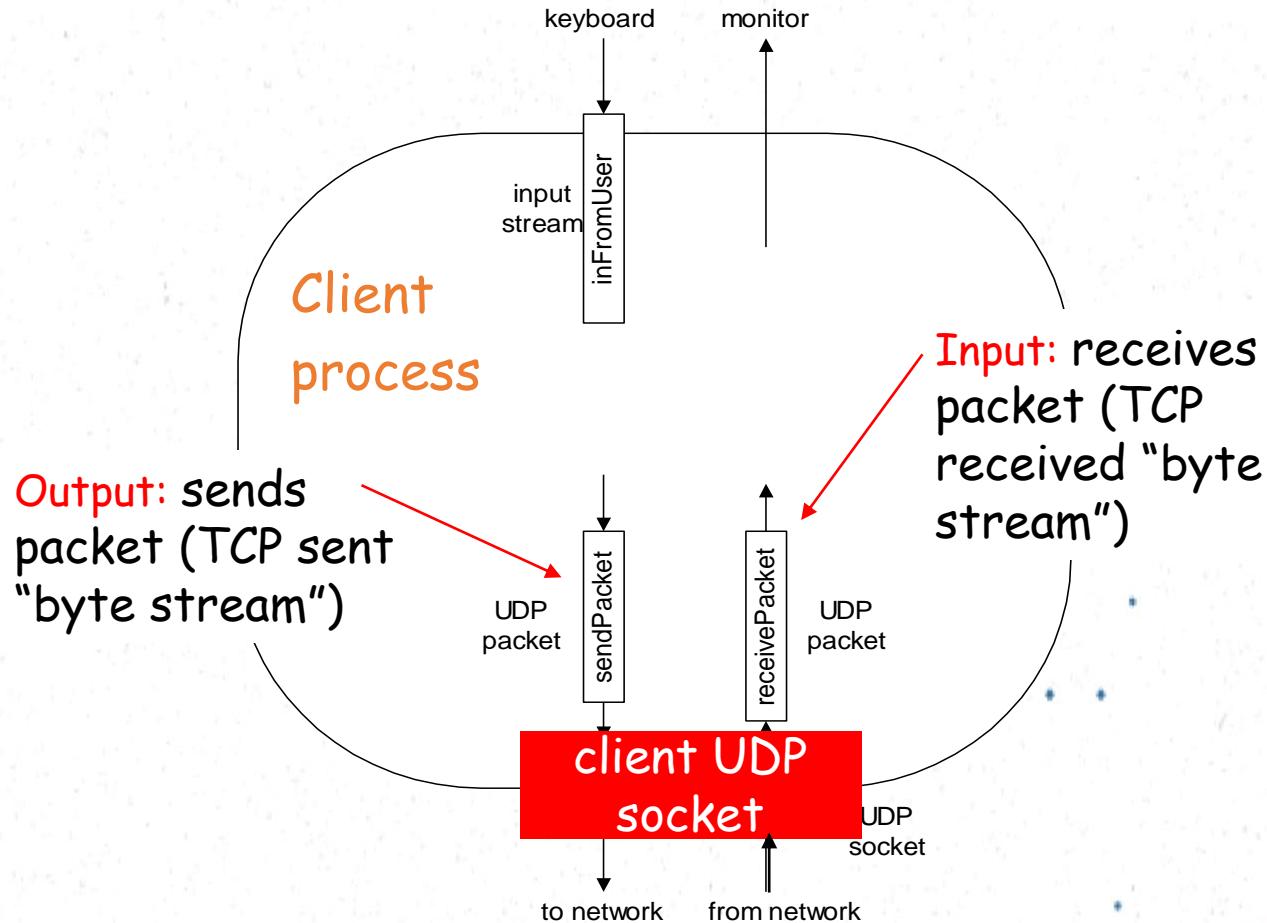
DatagramPacket methods

```
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();
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());
            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);

        }
    }
}
```

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 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)

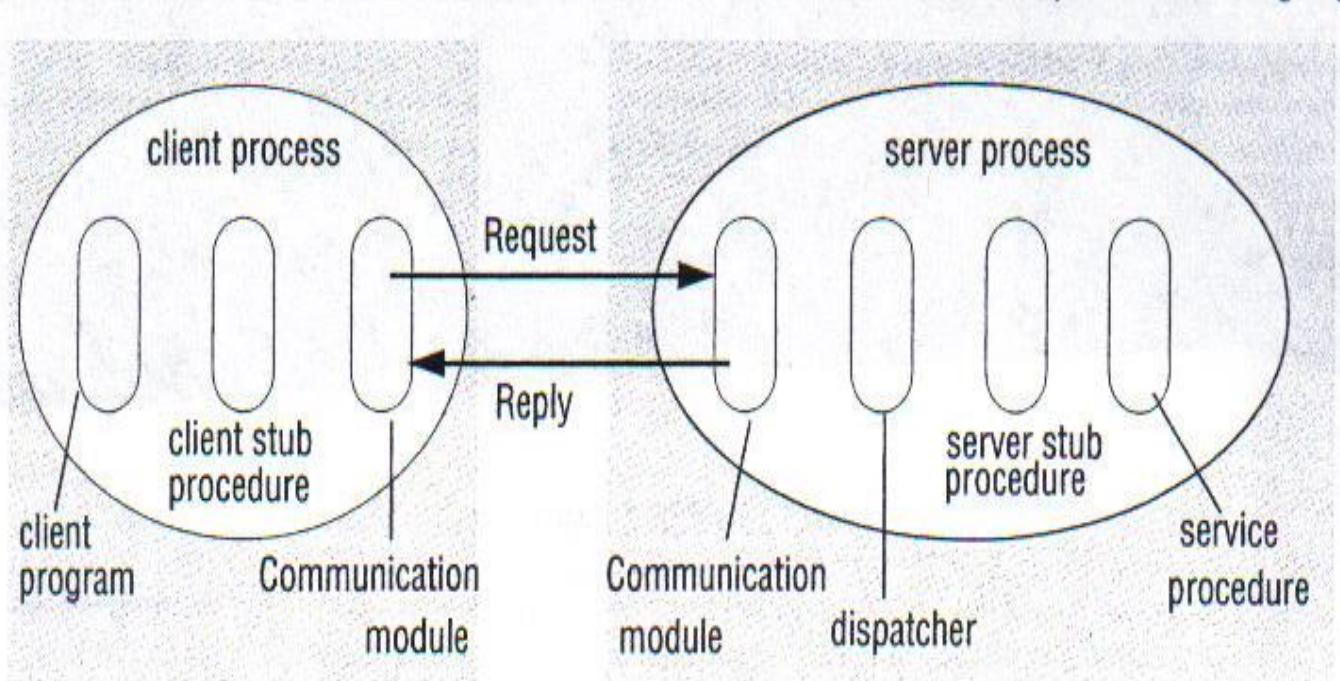


Figure 3. Role of client and server stub procedures in RPC in the context of a procedural language

Coulouris, Dollimore and Kindberg *Distributed Systems: Concepts & Design* Edn. 4, Pearson Education 2005

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

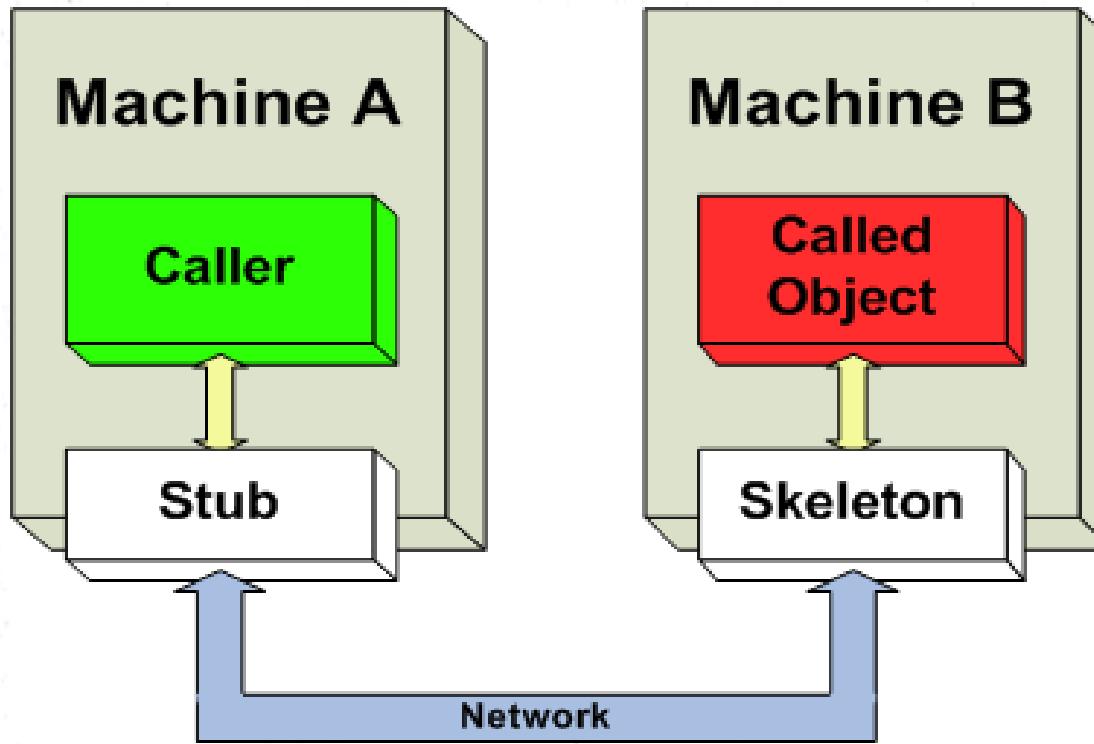


Fig: Distributed Object Technology

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

Let's divide into two perspectives:

- Layered Structure
- Working Principles

RMI Layered Structure

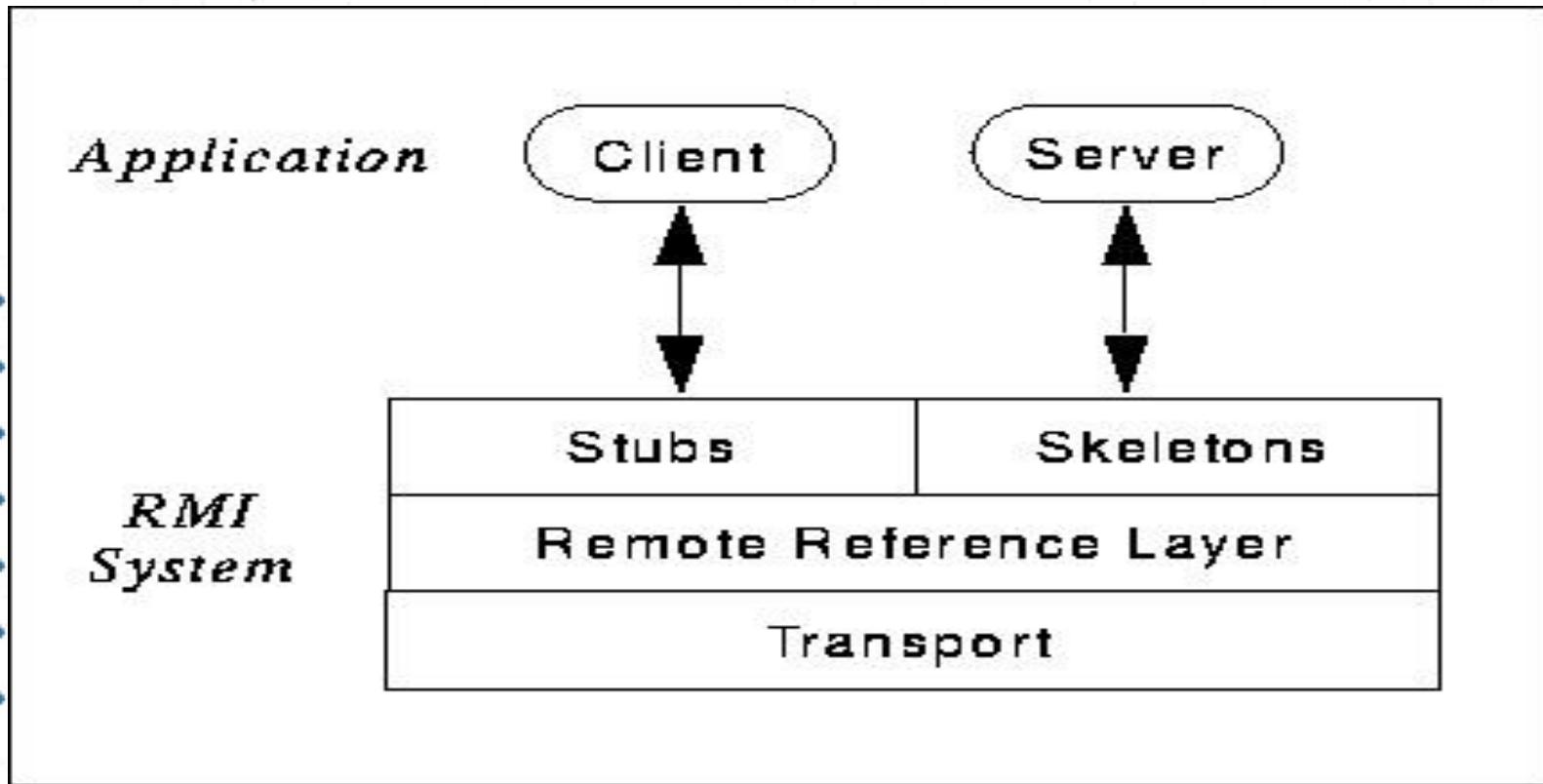


Fig: RMI Layered Structure

RMI Layered Structure

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

RMI Working Principles

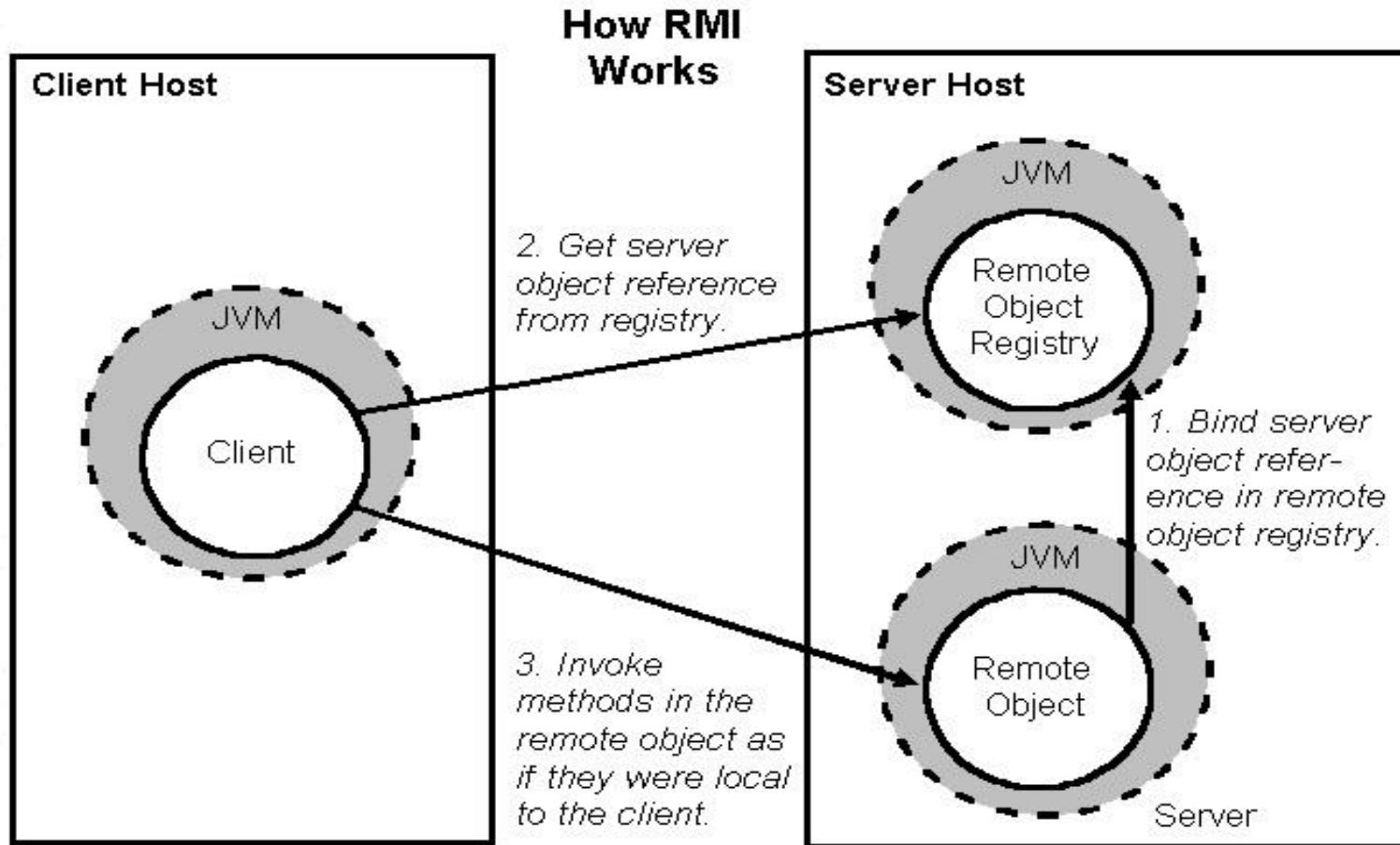
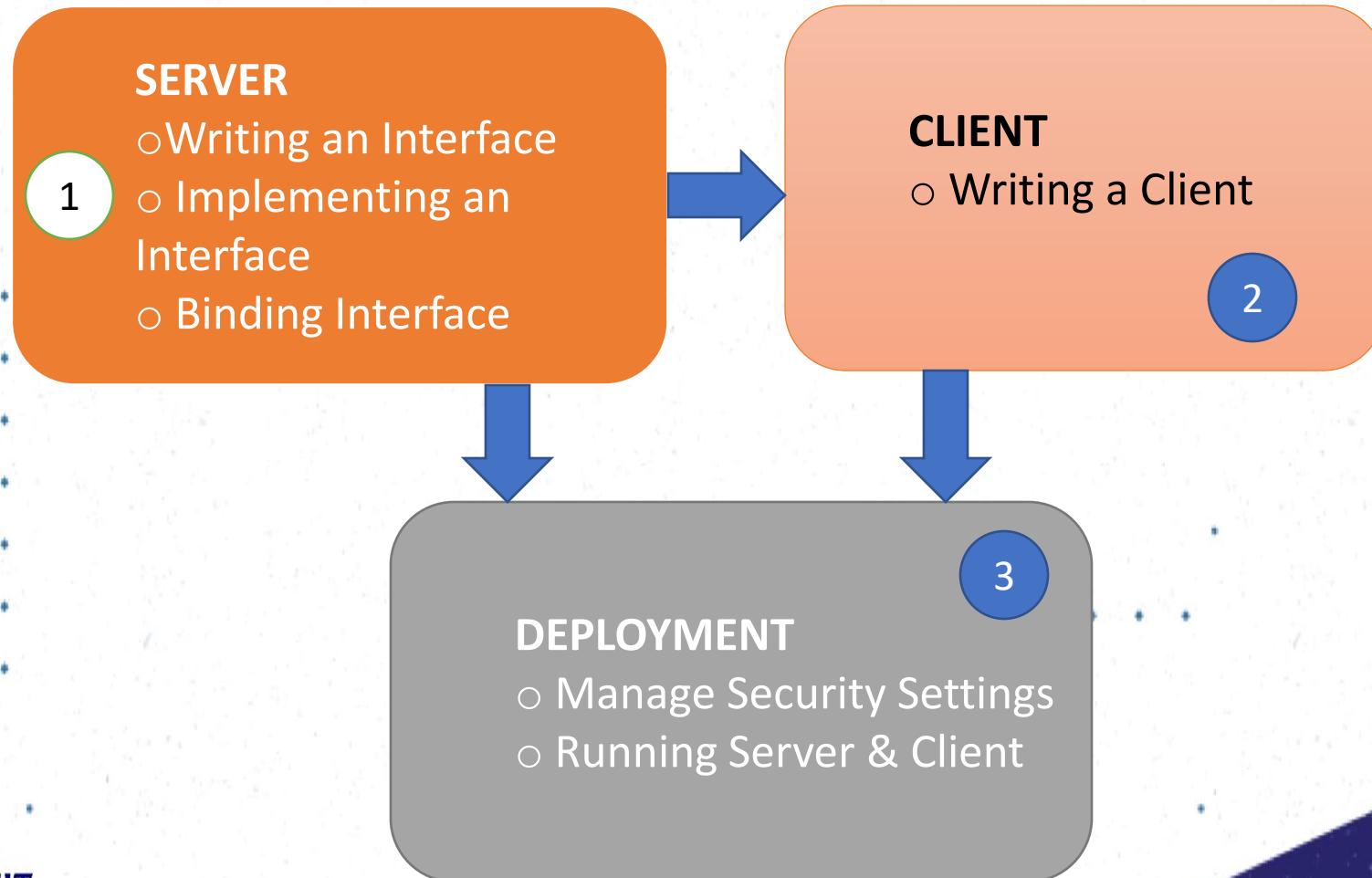


Fig: RMI Working principles

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;
```

- Square Operation

```
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 {

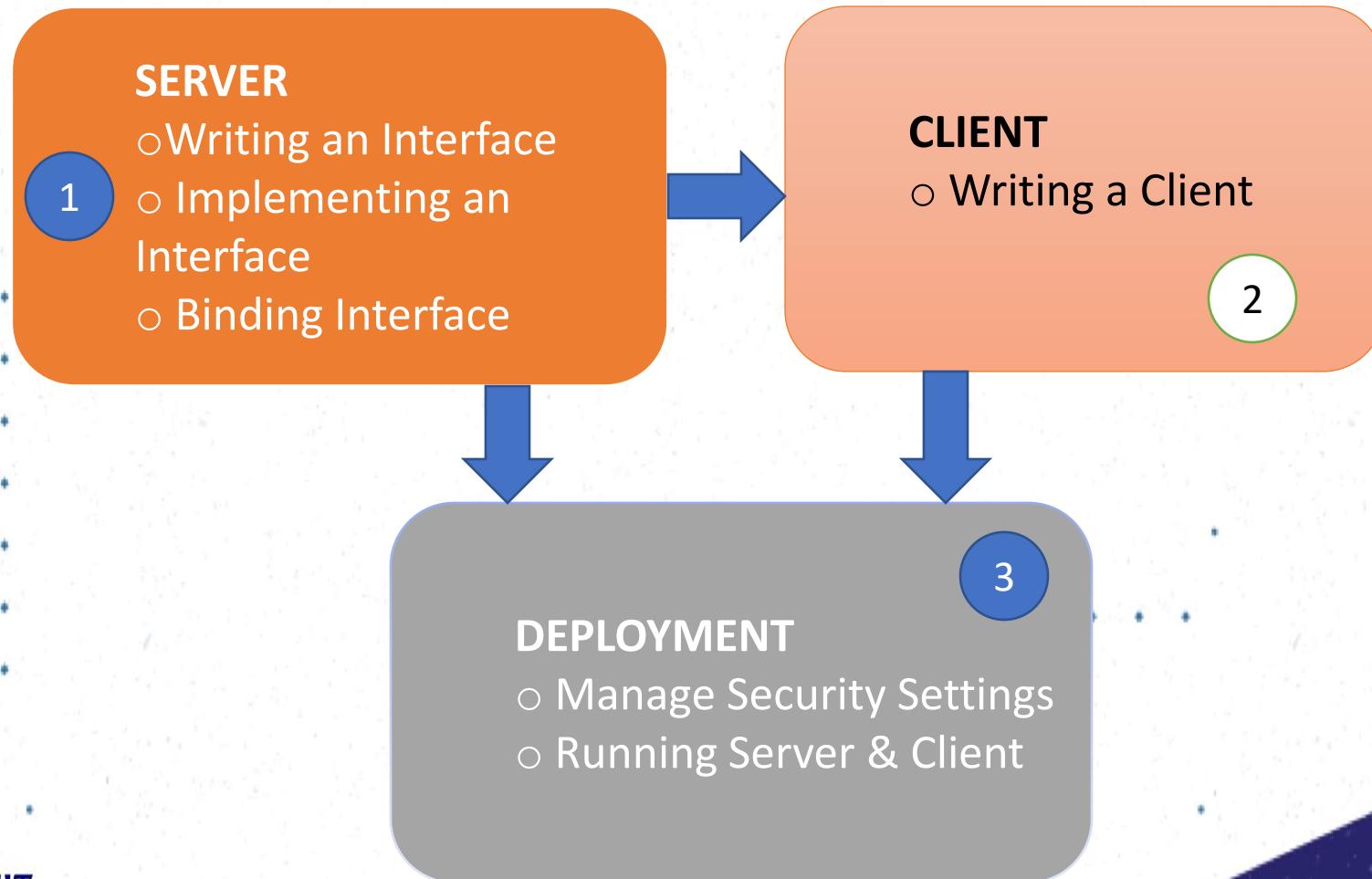
+
+
+
// MathServiceProvider implements all the methods of MathService interface
// service constructor
public MathServiceProvider() throws RemoteException {
    super();
}

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

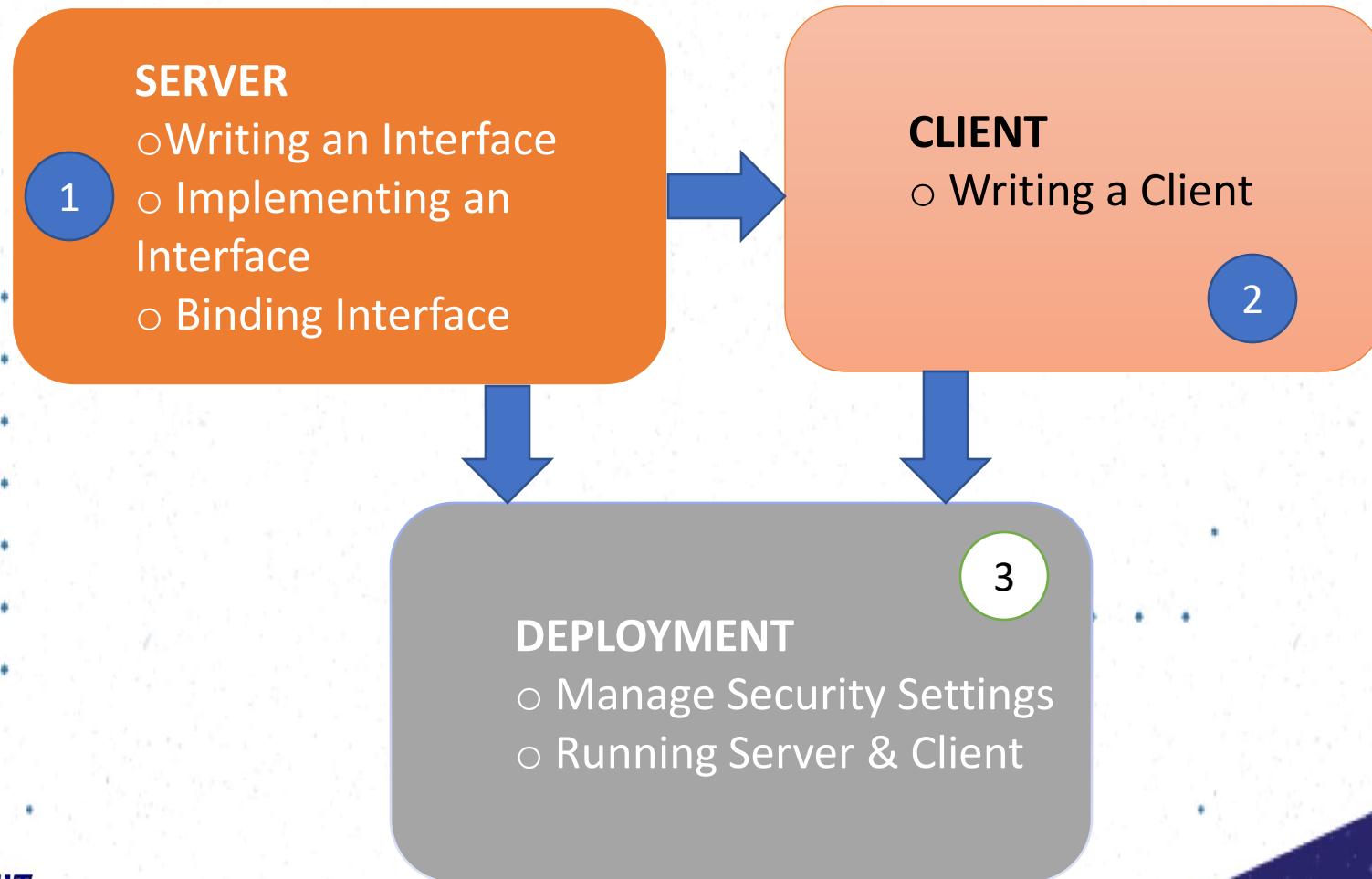
Fig: Client locating *MathService* service

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 Unix

```
$ 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 contains the following text:

```
Administrator: C:\Windows\System32\cmd.exe -java -Djava.security.policy=no.policy MathService...
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 Math
ServiceProvider
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)

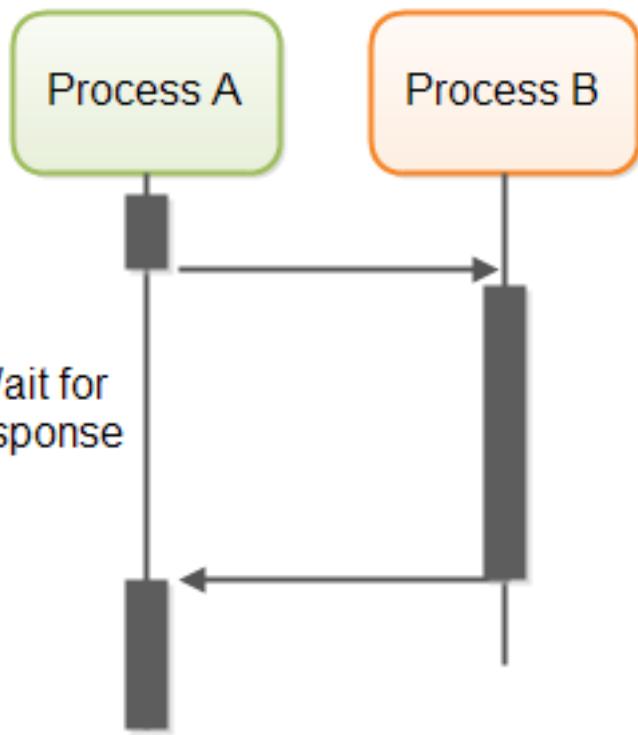
Lecture 5

Asynchronous Communication

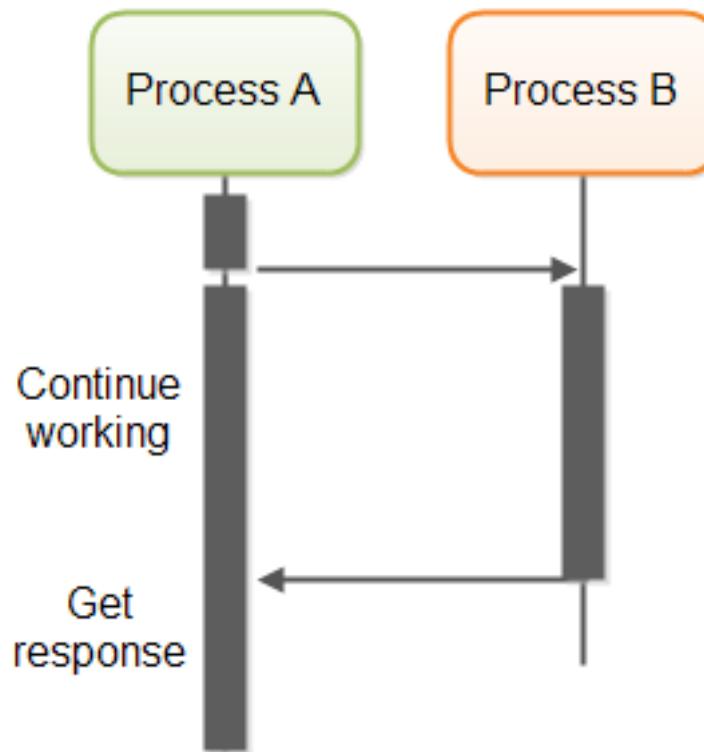
This Week

- 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 Call-back 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?

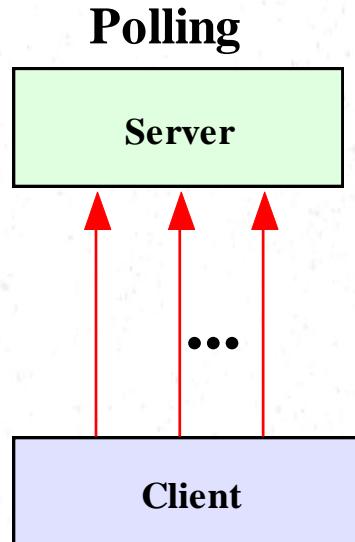
- Every RPC call is potentially long-running
 - Network/server failures are only detected after timeouts expire
 - Making every RPC call asynchronous increases code complexity, just on the *chance* a network failure occurs
- So use asynchronous calls only on functions that are expected to take a long 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

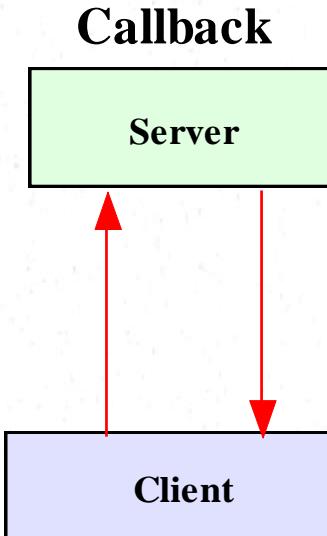
Polling vs. Callback

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 remote method call



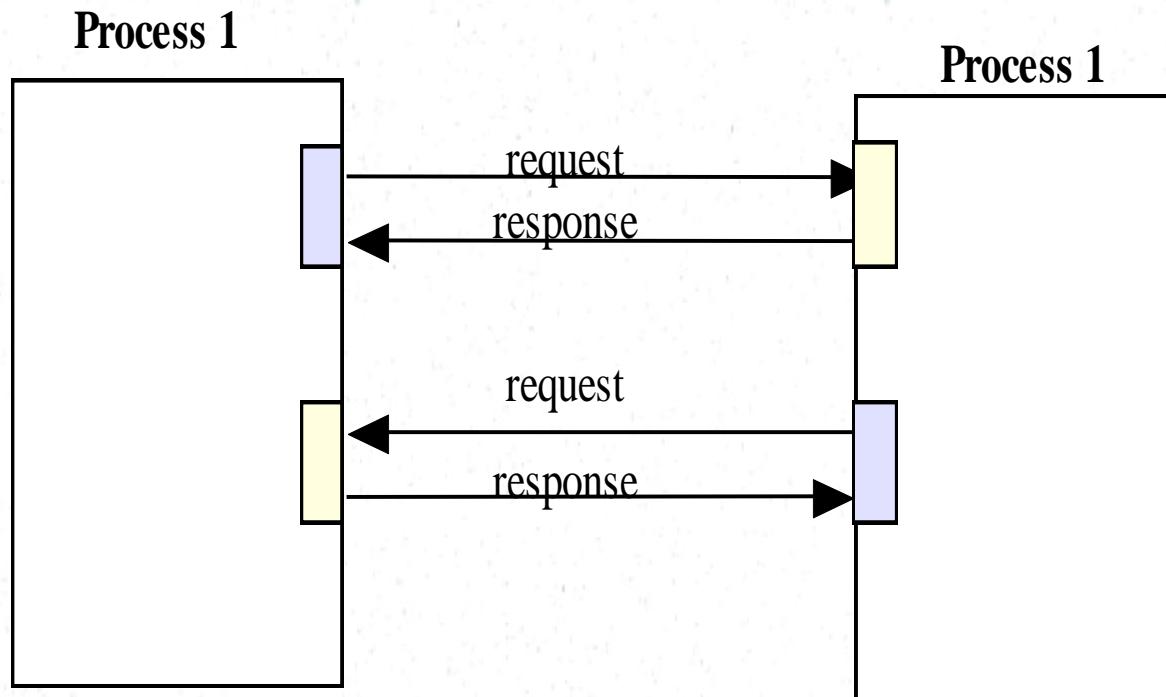
A client registers itself with the server, and wait until the server calls back.

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

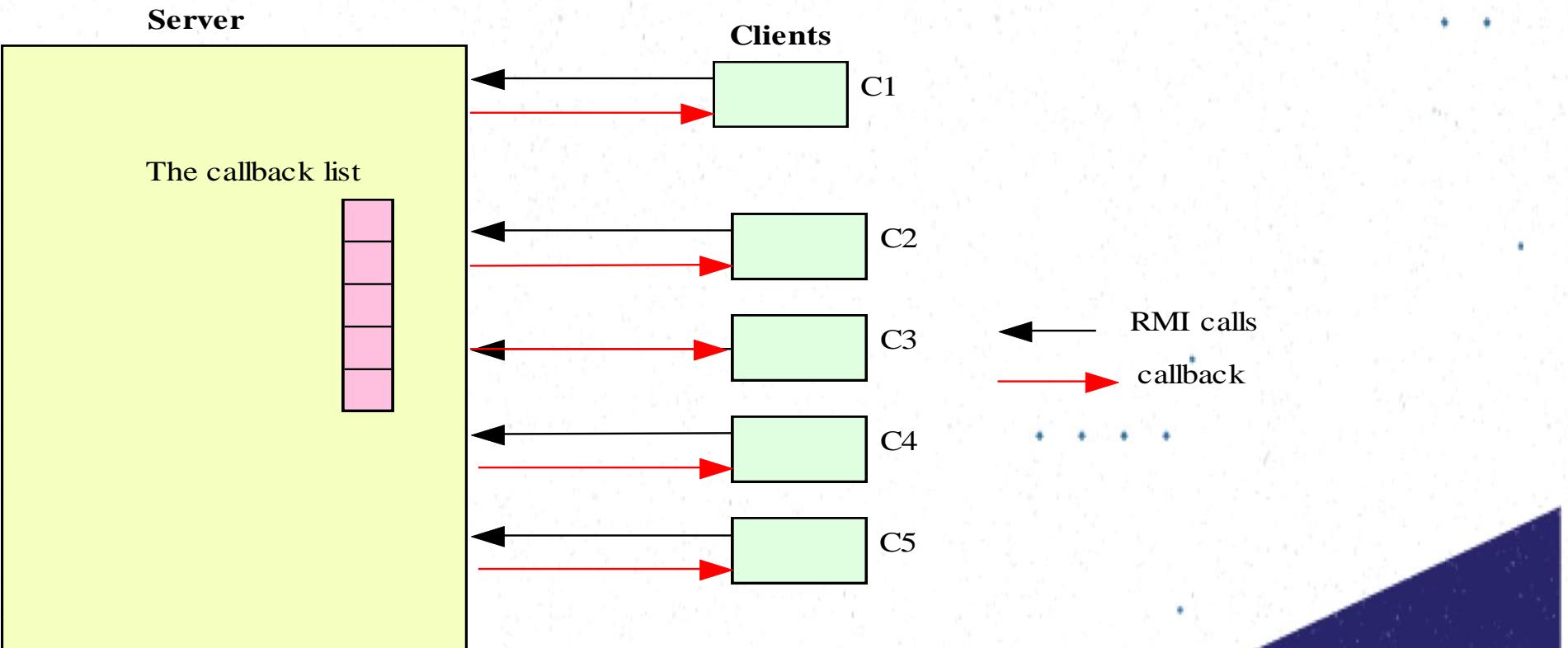
- Some applications require that both sides may initiate IPC.
- Using sockets, duplex communication can be achieved by using two sockets on either side.
- With connection-oriented sockets, each side acts as both a client and a server.



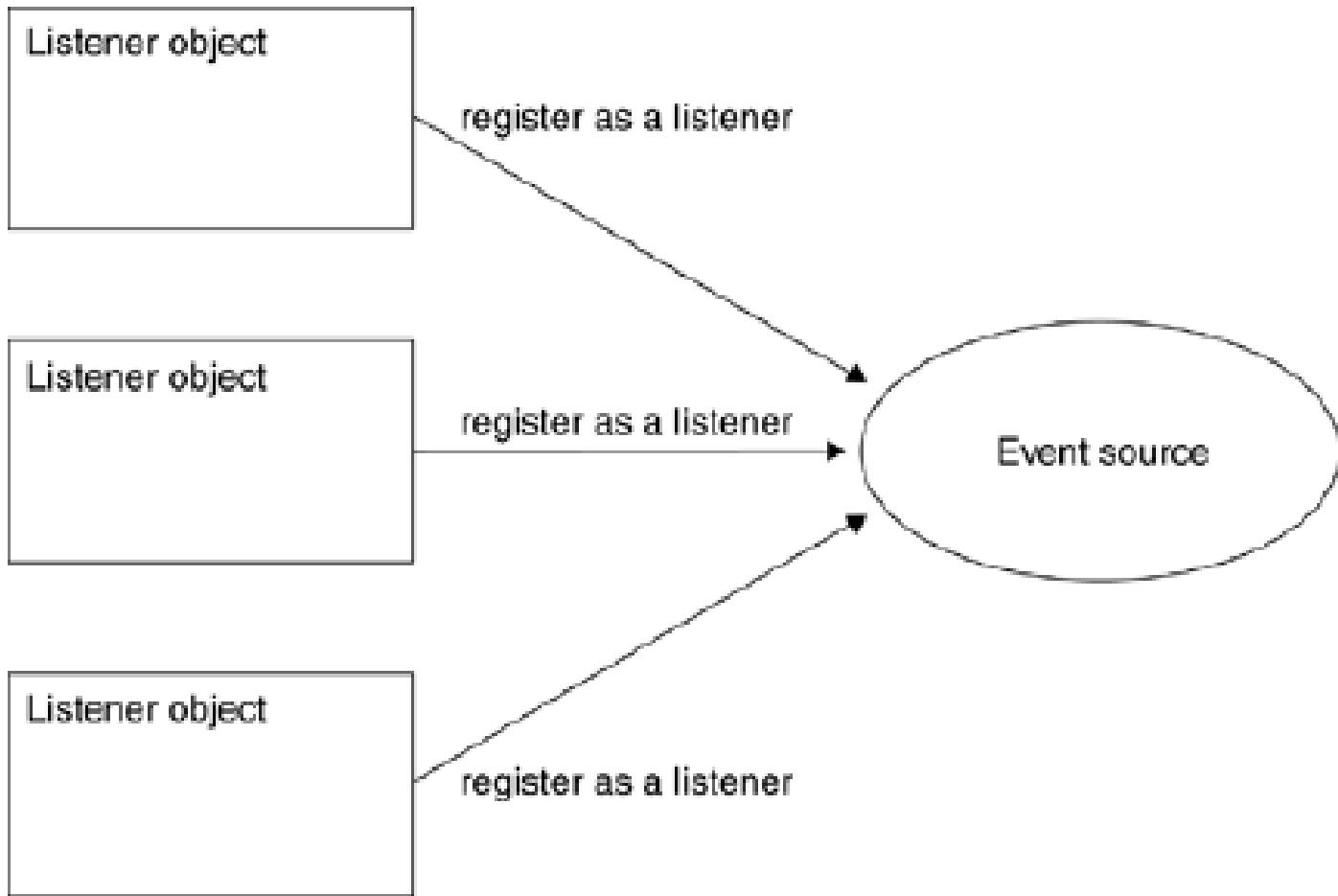
RMI Callbacks

RMI Callbacks

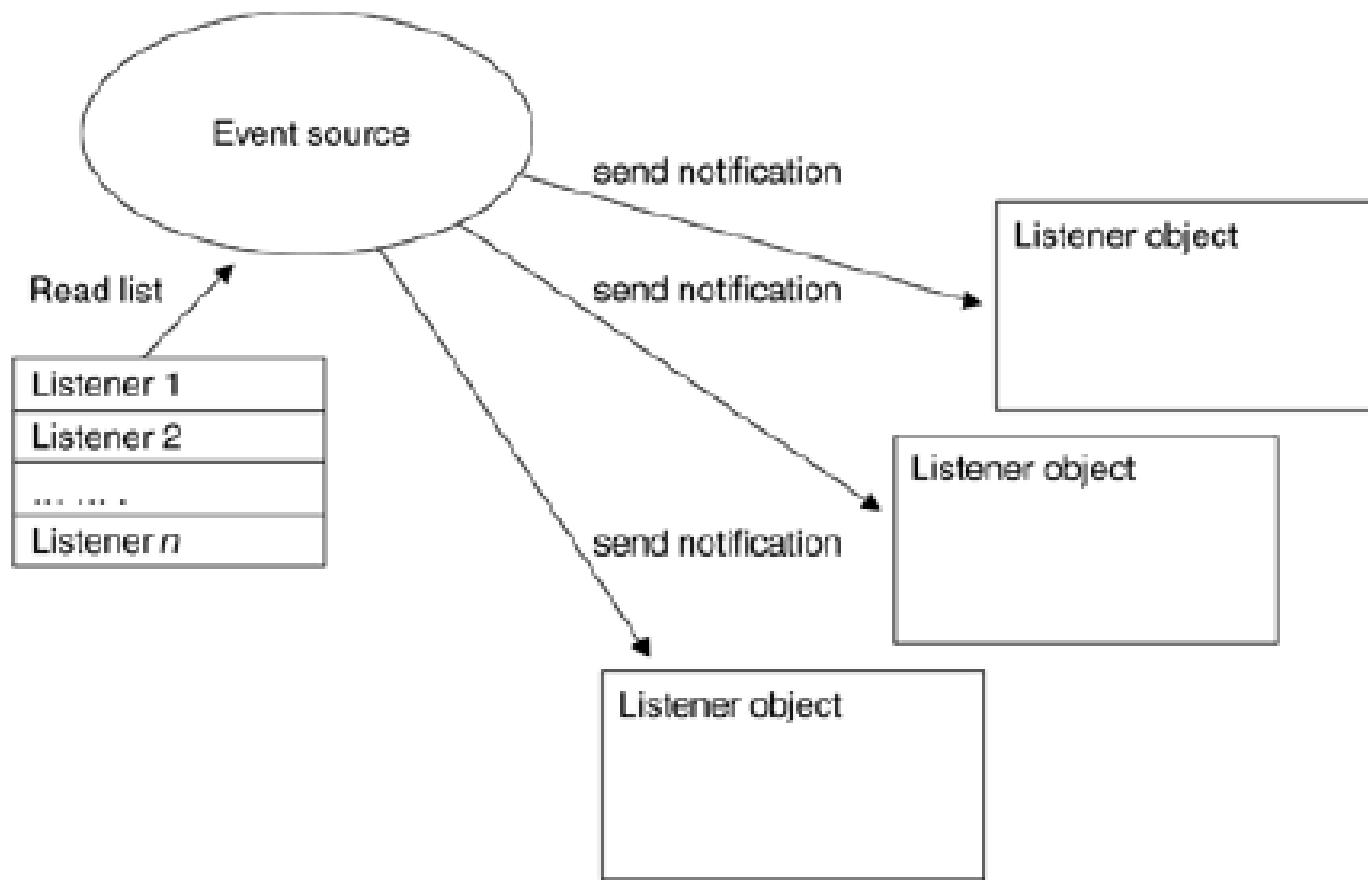
- A callback client registers itself with an RMI server.
- The server makes a callback to each registered client upon the occurrence of a certain event.



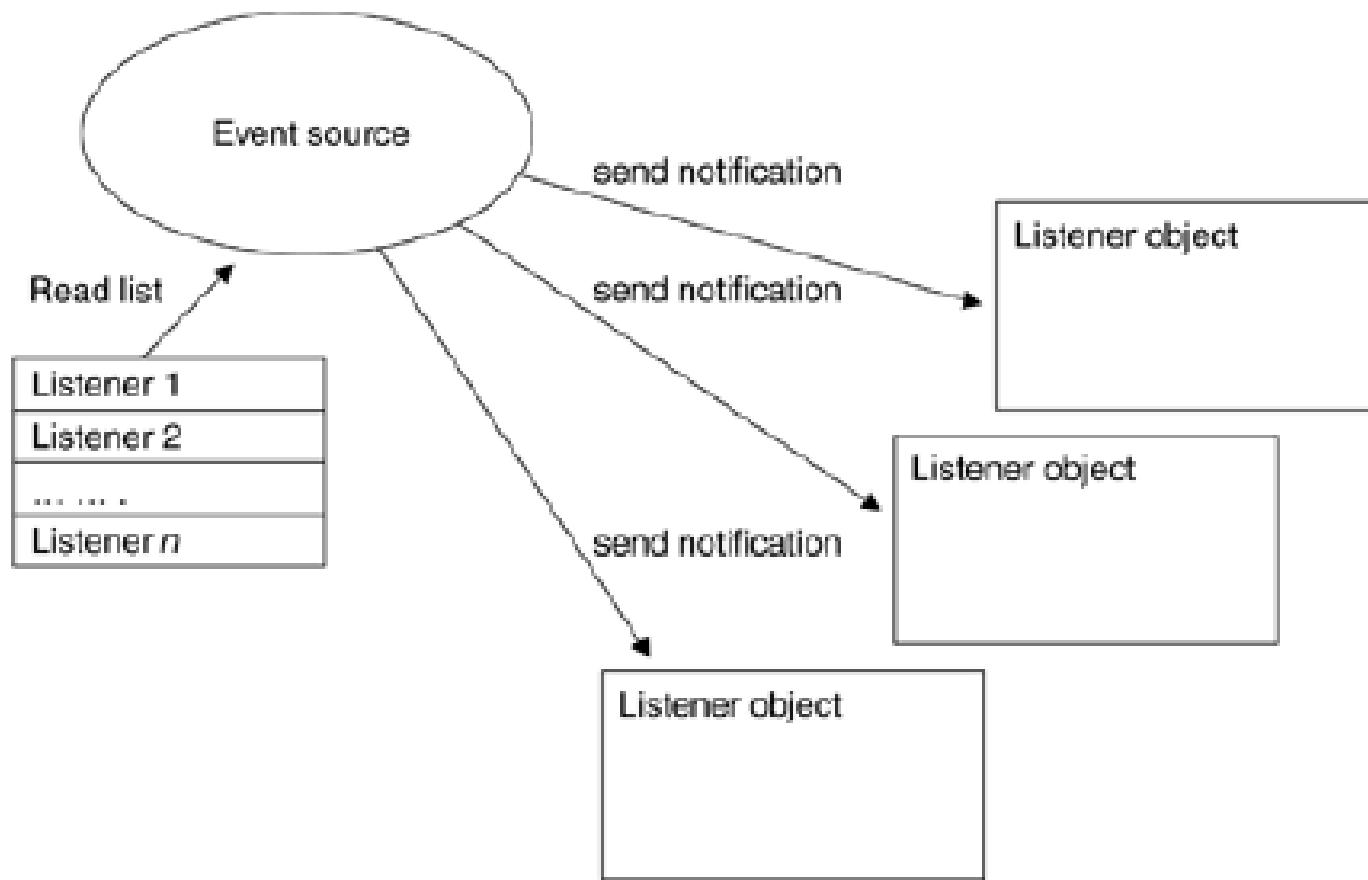
Multiple listeners



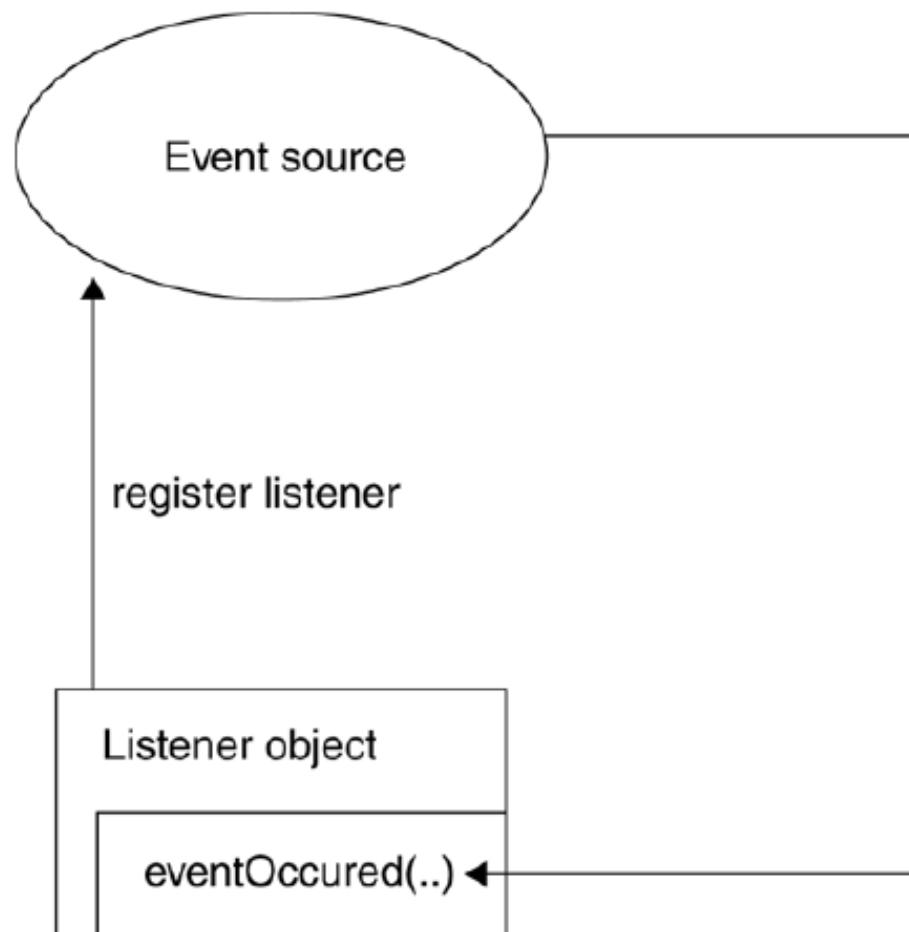
Callback notification to every registered listener



Callback notification to every registered listener



Callback implemented by invoking a method on a listening object



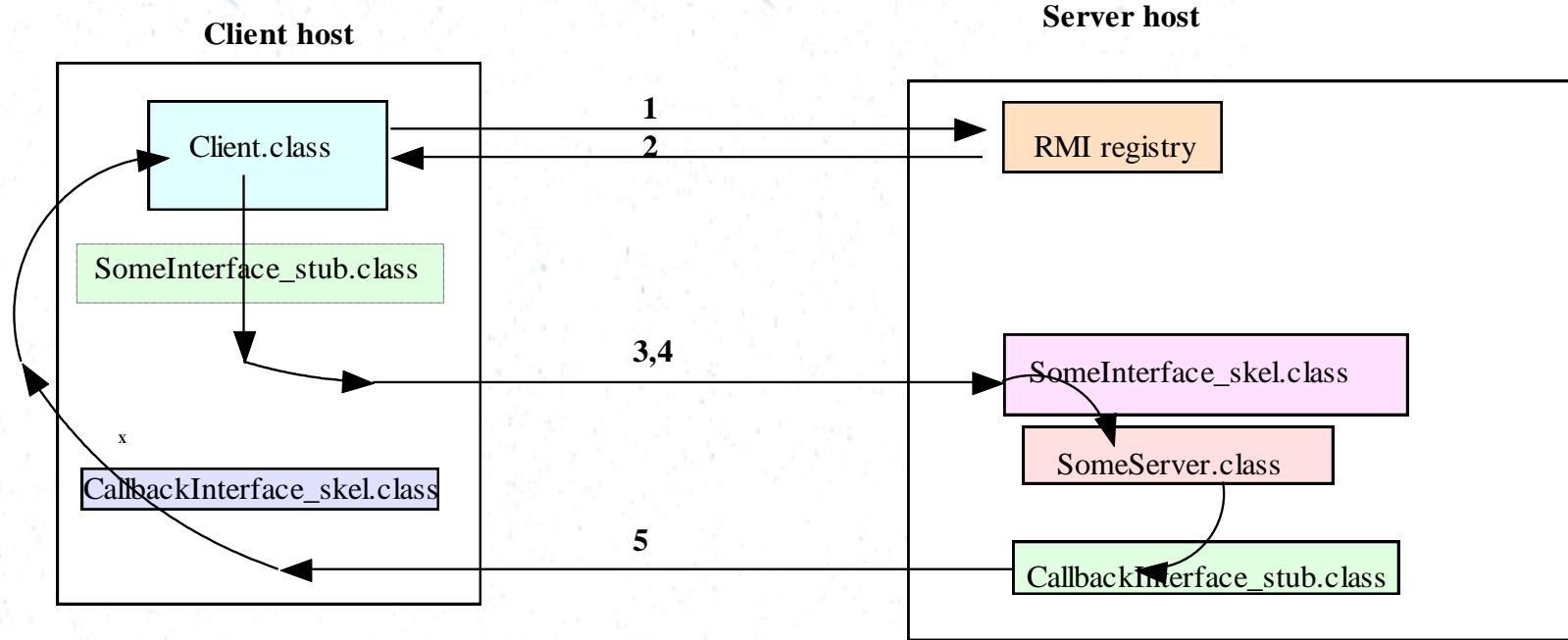
Client callback

- 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

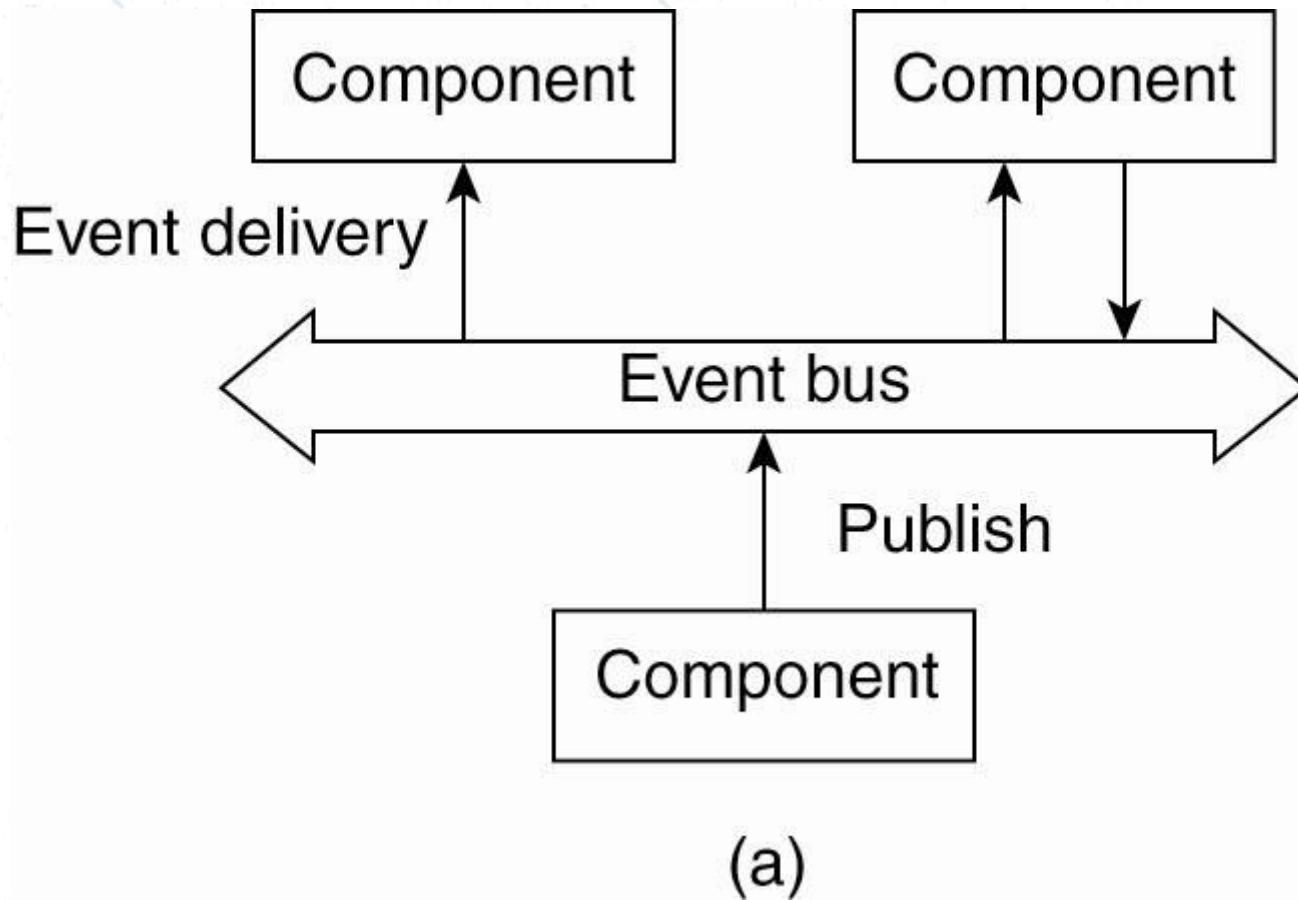
1. Compile the applications and generate stub/skeleton files for both
2. TemperatureSensorServer and TemperatureSensorMonitor.
3. Run the rmiregistry application.
4. Run the TemperatureSensorServer.
5. Run the TemperatureSensorMonitor.

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

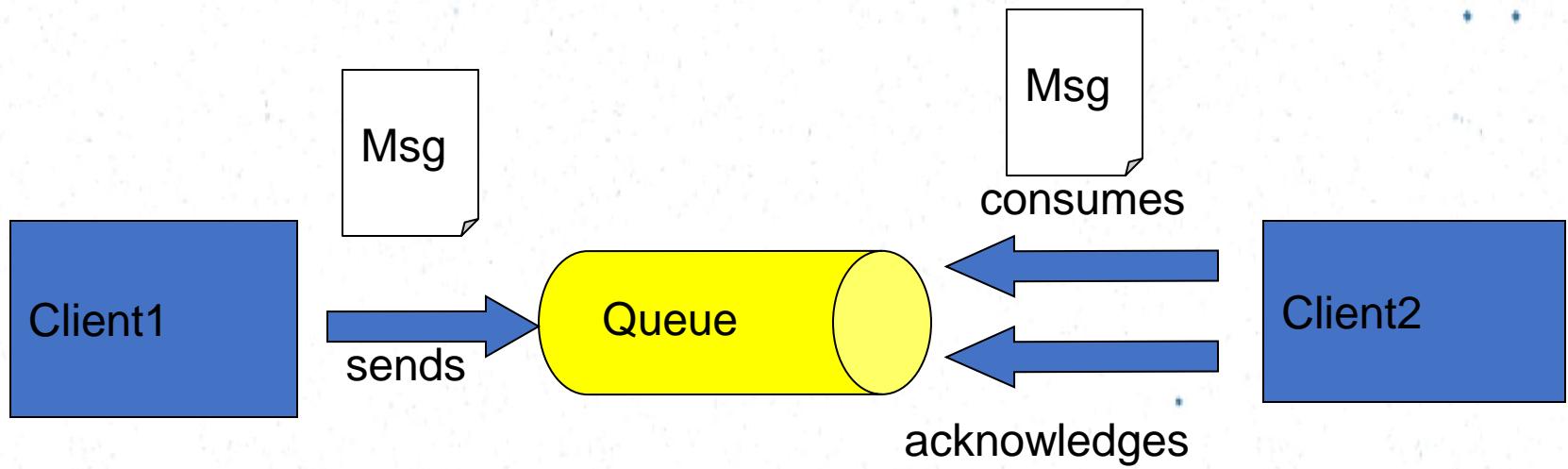
A JMS Application

- 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

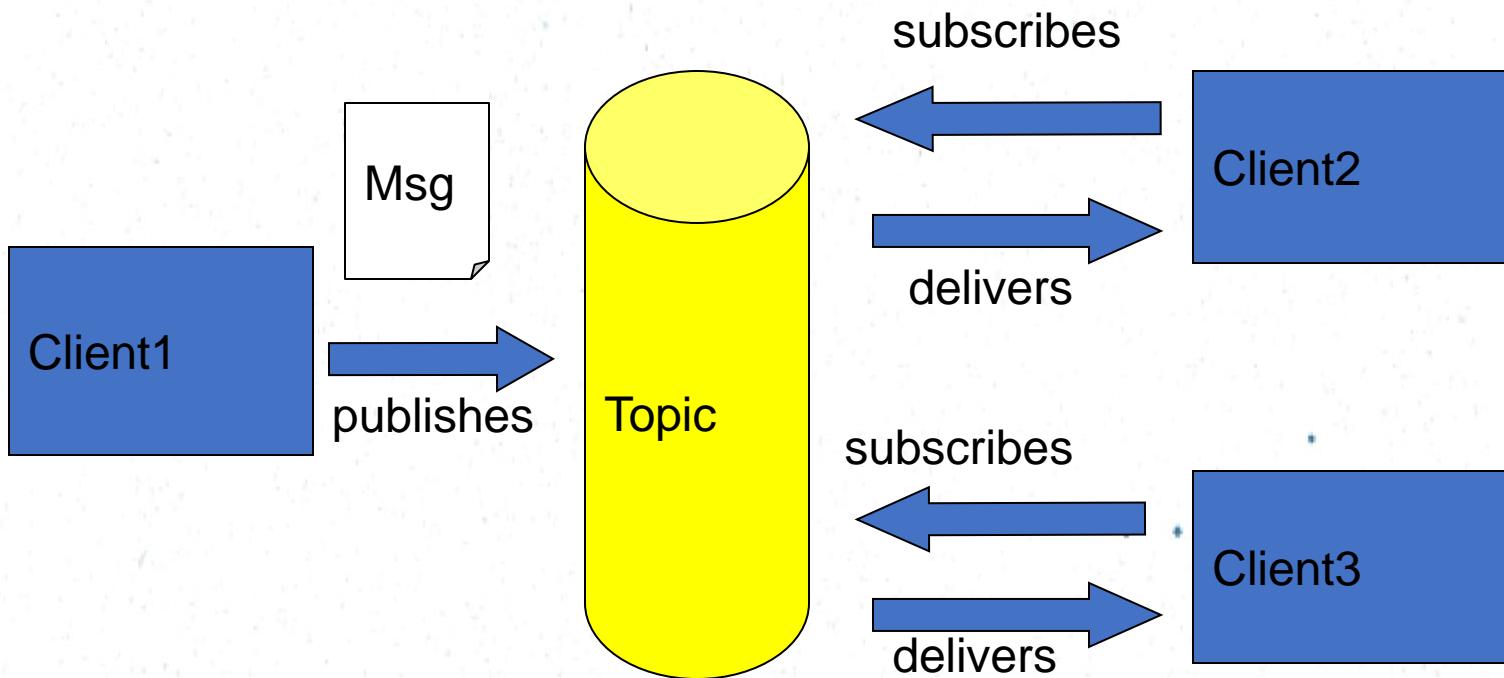
JMS Messaging Domains

- 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



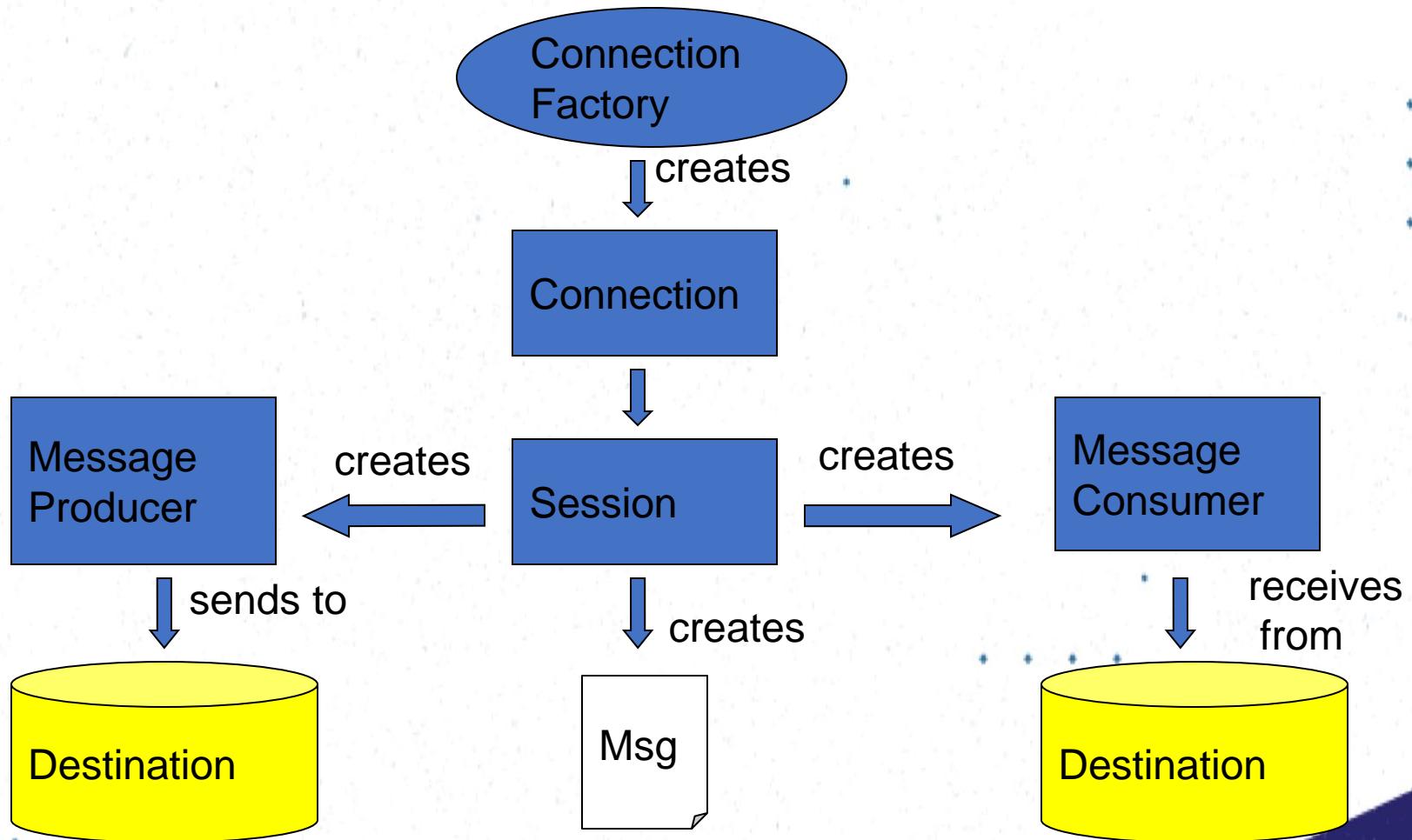
Publish/Subscribe Messaging



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 (JMSEException e) {  
        System.out.println("JMSEException 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>

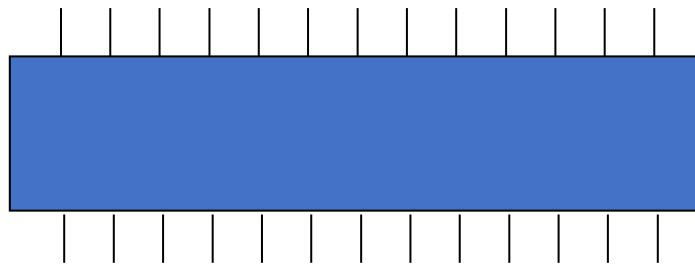
Summary

- 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

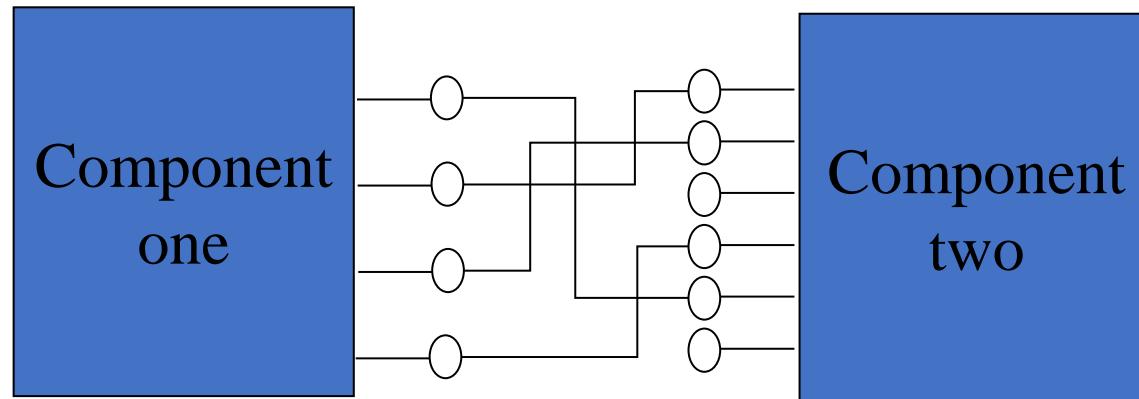
I/O Data bus

ACK

O Acknowledge: Accepted when low

Building complex systems

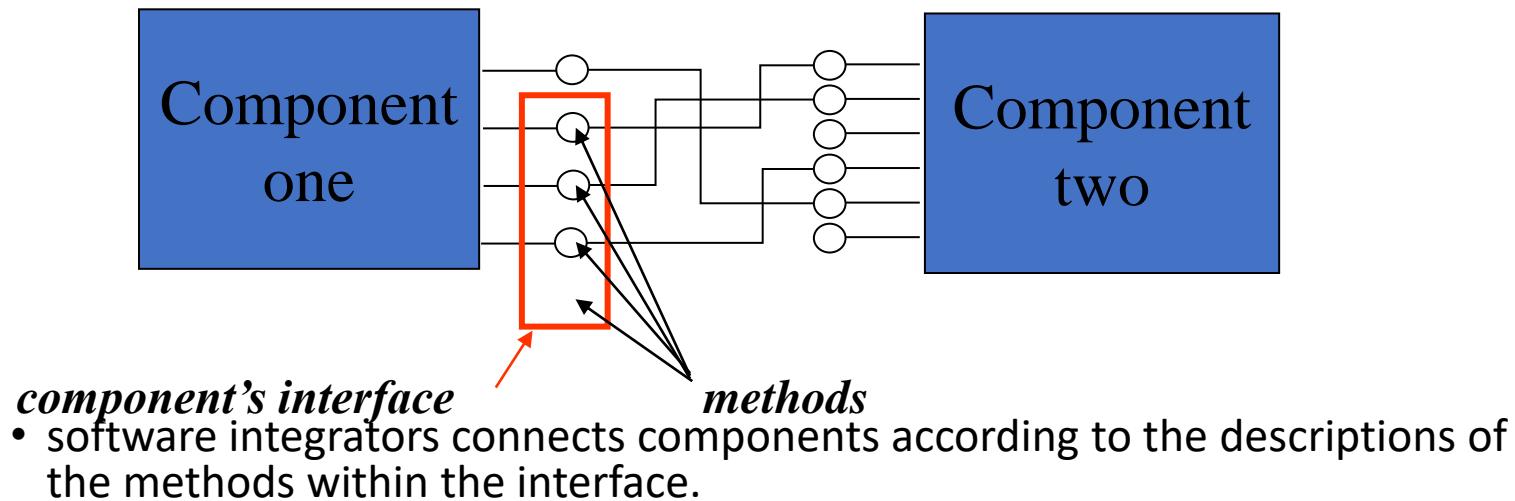
- Hardware system integrators connect pins of chips together according to their functions to build complex electronic devices such as computers.



- 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

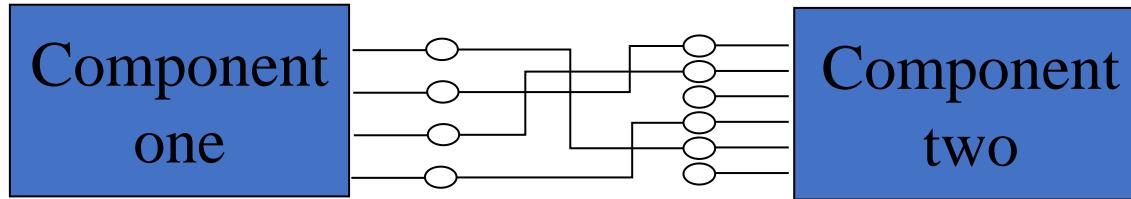
- The software component model takes a very similar approach:
 - the “pins” of software components are called interfaces
 - an interface is a set of methods that the component implements



Software 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

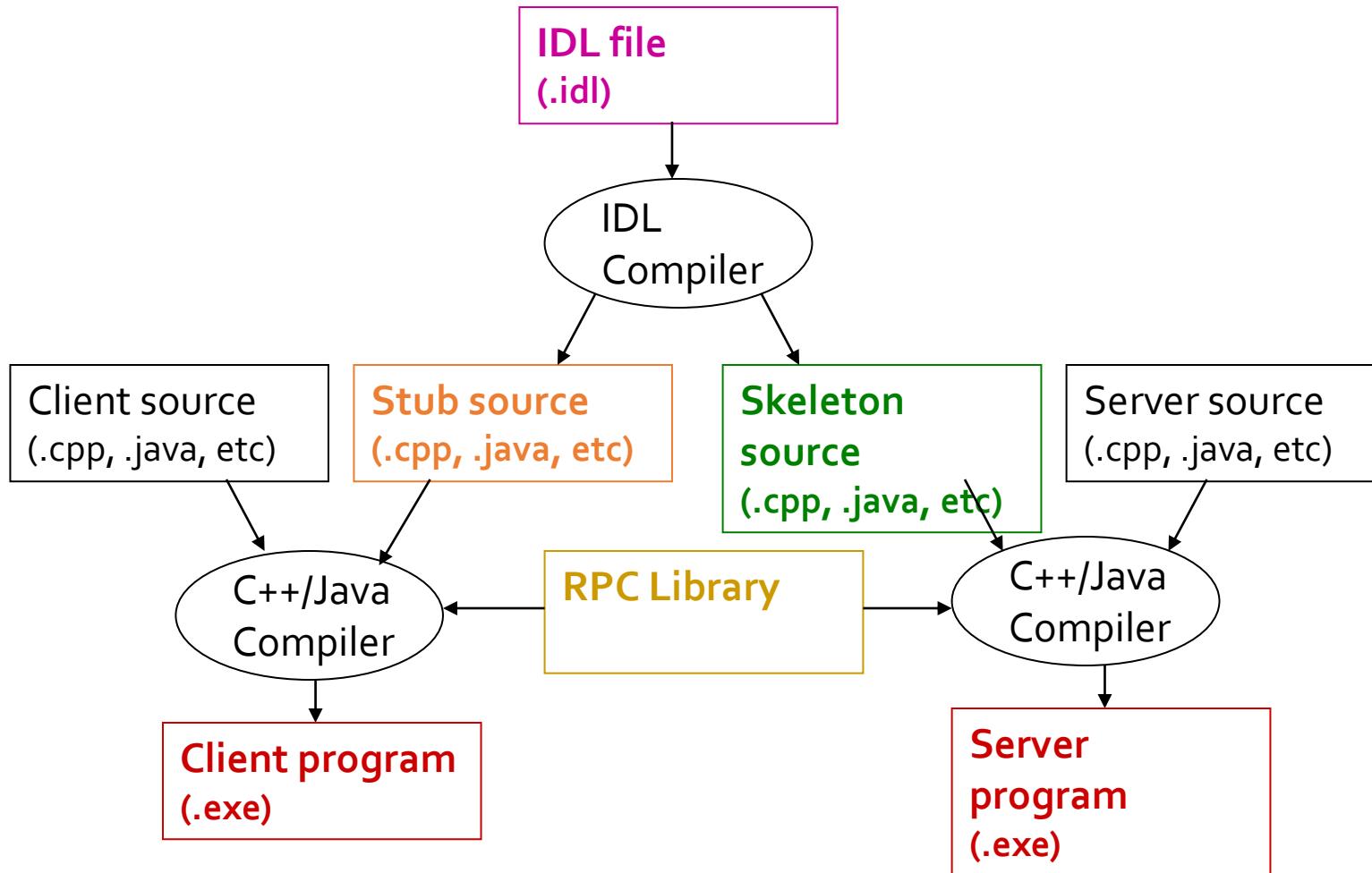
Examples of component frameworks

- 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



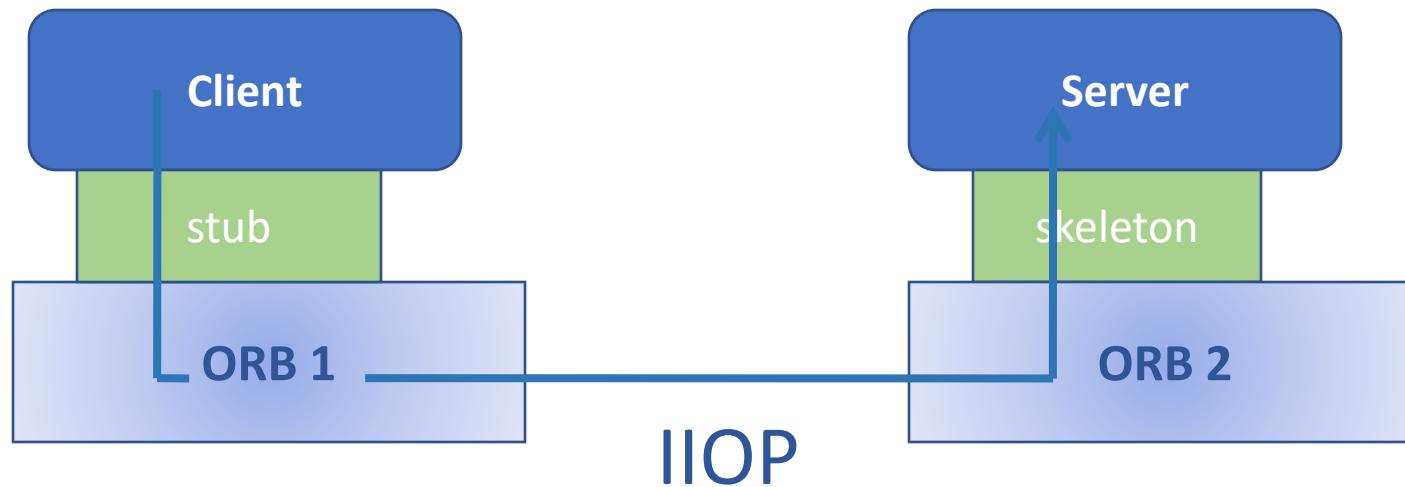
CORBA and O-O

- 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

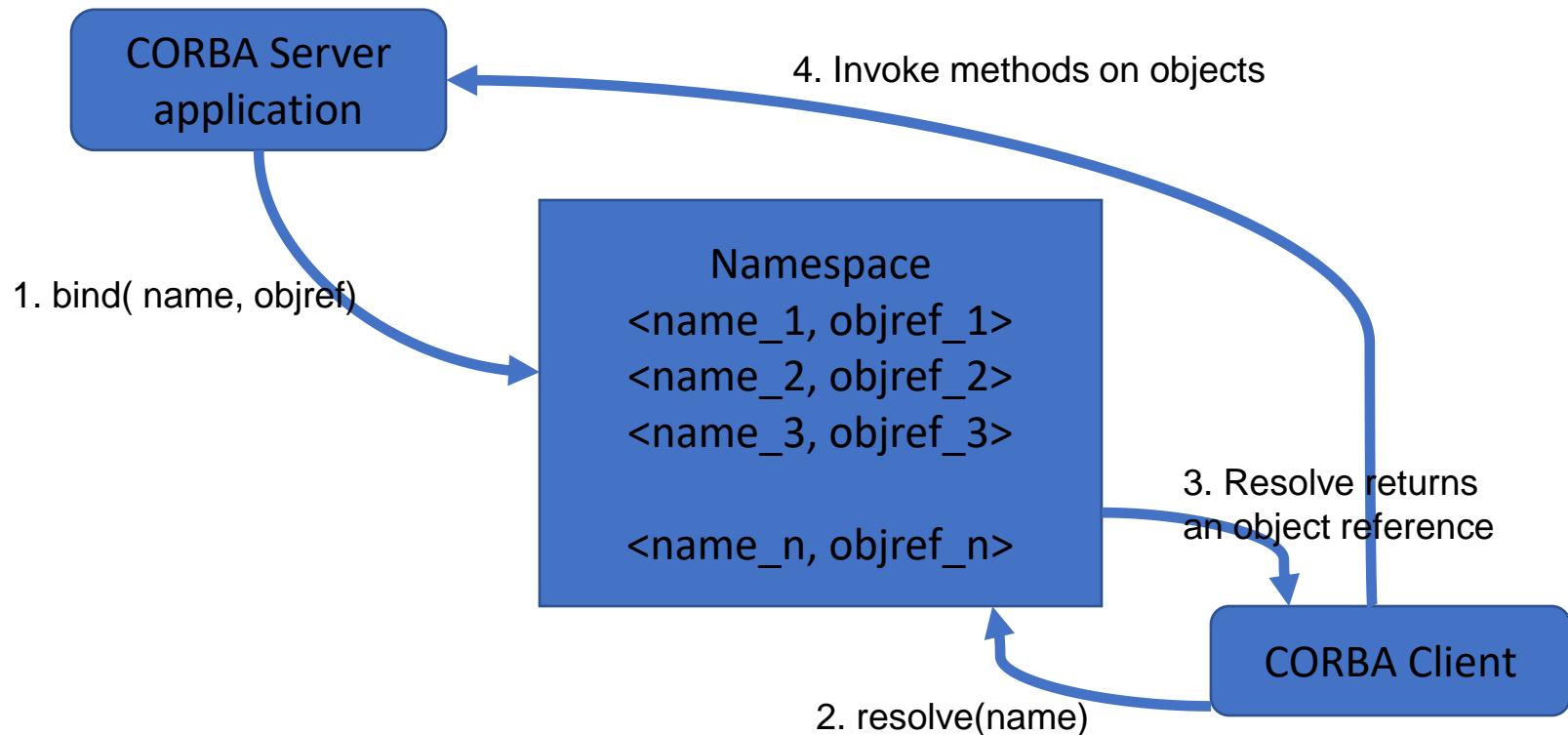
CORBA Interoperability



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

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

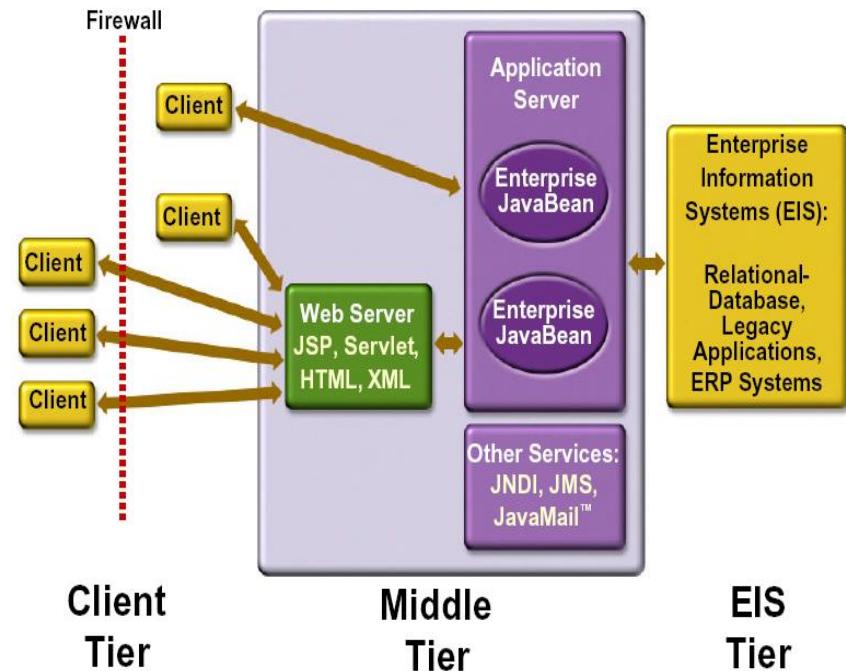
- Although conceptually quite elegant, CORBA never really caught on, for a couple of reasons:
 - The CORBA spec is huge, making it complex to use
 - Competing frameworks were developed around the same time that had better backing and/or a broader coder base

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

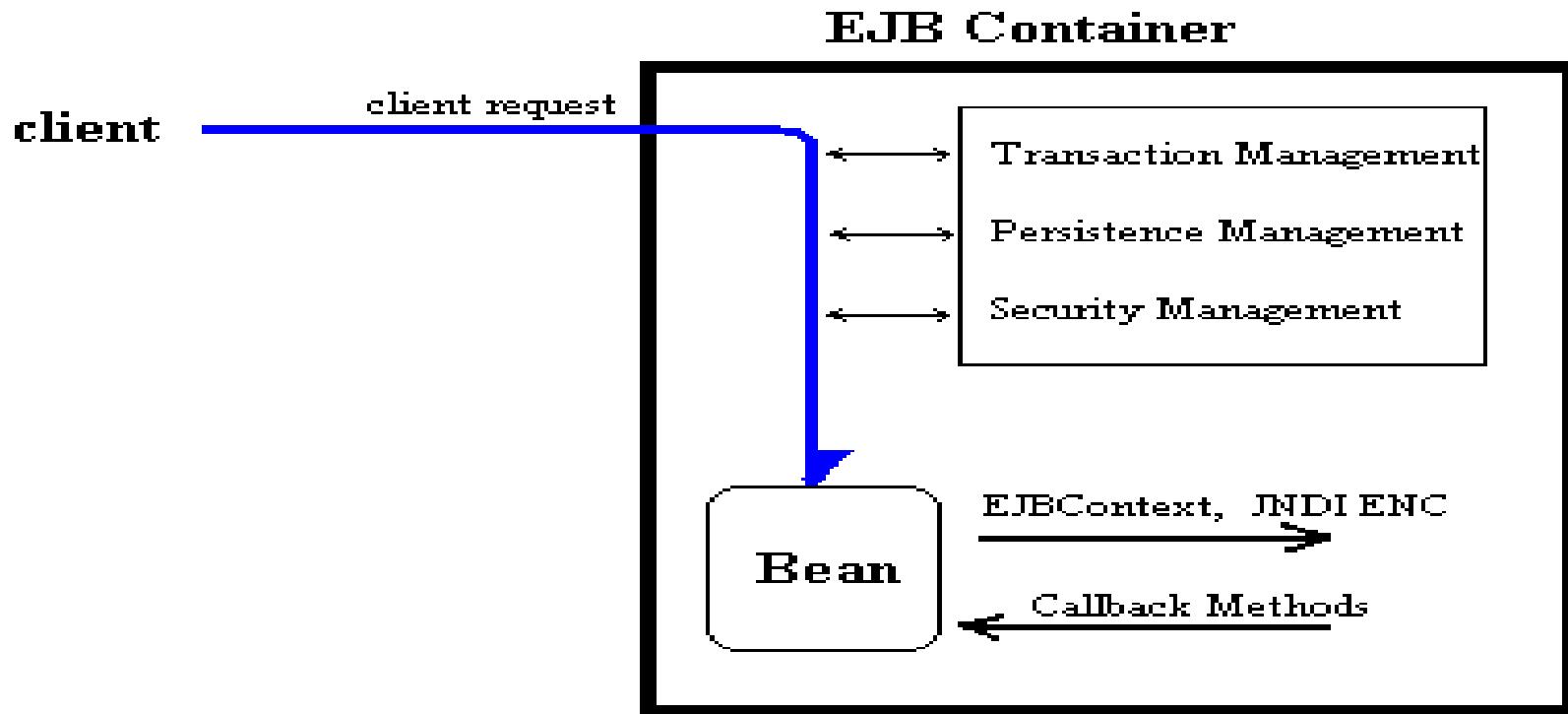
Enterprise Java Server (EJS) / Application Server

- 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)

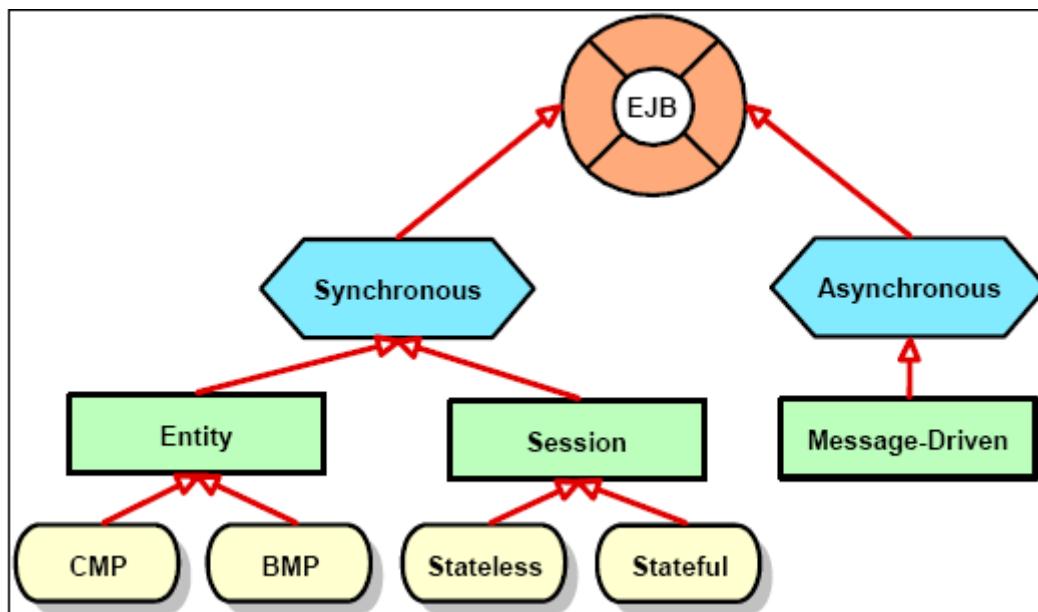


**EJB Containers manage
enterprise beans at runtime**

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)

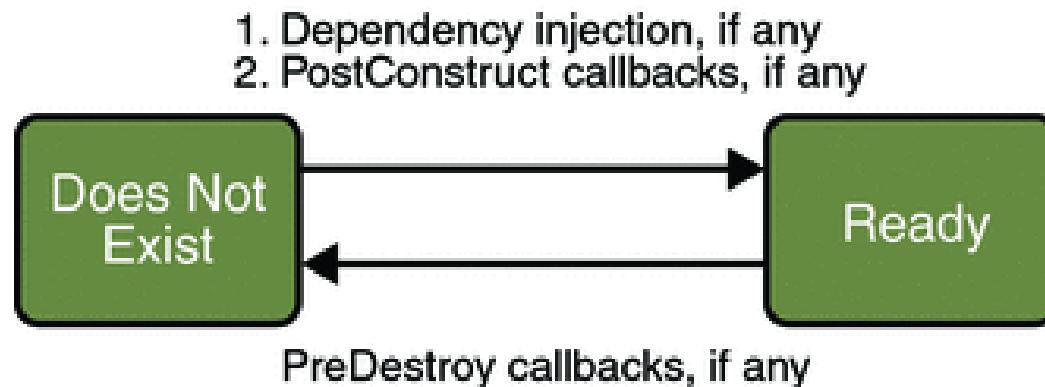
- The stateless bean with local interface:

```
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



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) :

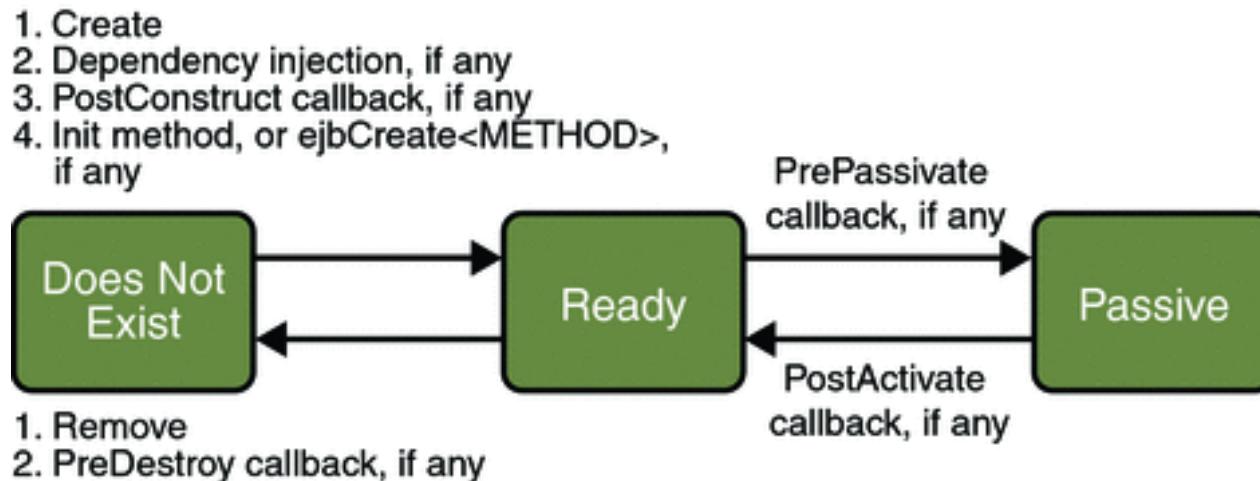
```
@Remote public interface ShoppingCart {  
    public void addItem(String item);  
    public void addItem(String item);  
    public Collection getItems();  
}
```

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

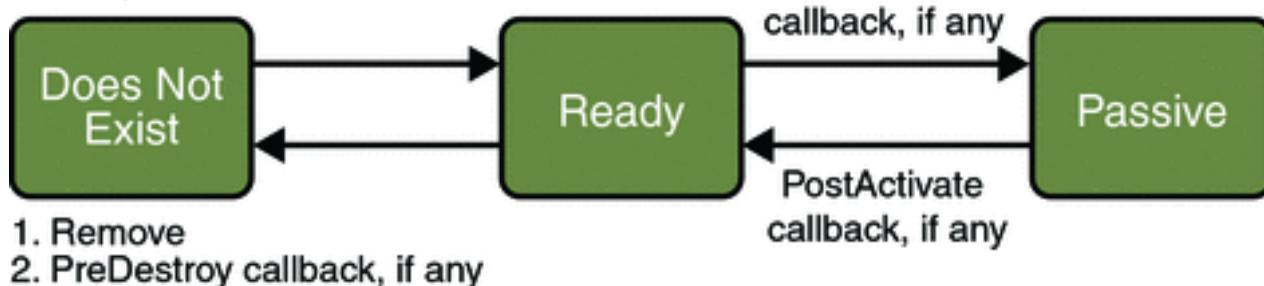
- Client initiates the lifecycle by obtaining a reference
- Container invokes the `@PostConstruct` and `@Init` methods, if any
- Now bean ready for client to invoke business methods



Lifecycle of a Stateful Session Bean

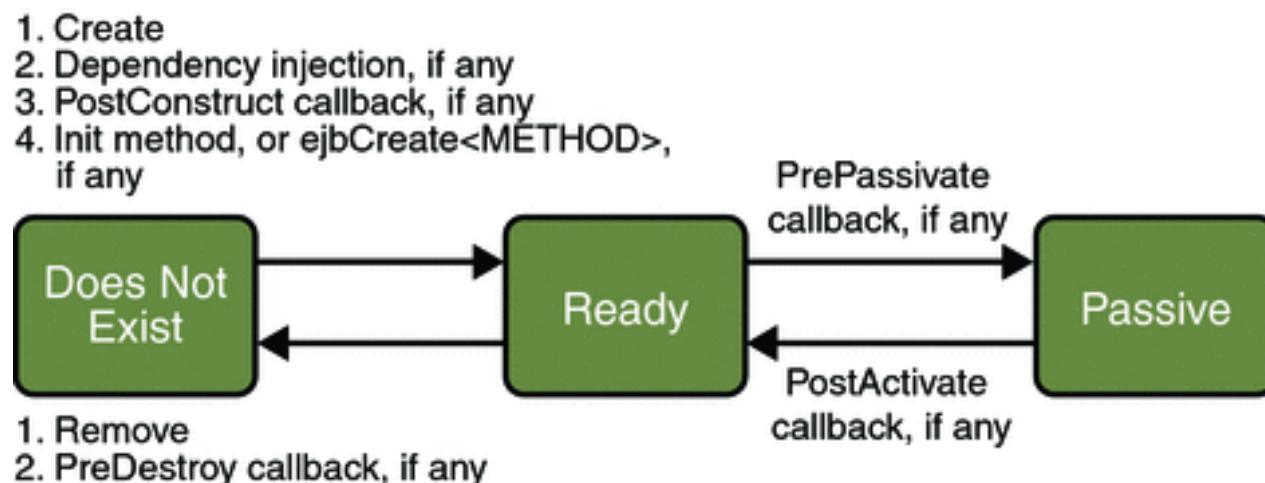
- While in ready state, container may passivate and invoke the `@PrePassivate` method, if any
- If a client then invokes a business method, the container invokes the `@PostActivate` method, if any, and it returns to ready stage

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



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)

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

EntityManager

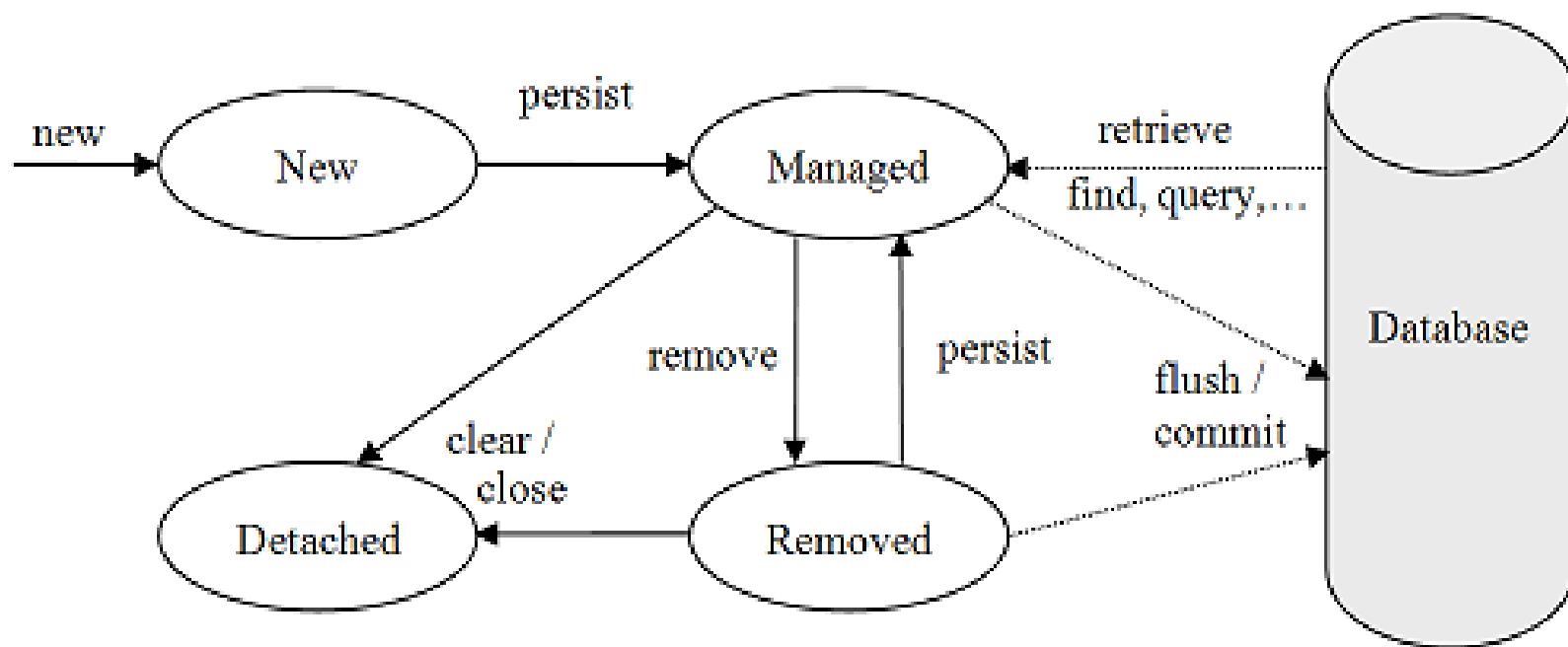
- EntityManager API is used to:
 - create and remove persistent entity instances
 - to find entities by their primary key identity, and to query over entities
- EntityManager supports EJBQL and (non-portable) native SQL

Entity Bean Lifecycle

Entity bean instance has four possible states:

- **New** entity bean instance has no persistent identity, and is not yet 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)

<u>Company</u>	<u>Product</u>
• IBM	WebSphere
• BEA Systems	BEA WebLogic
• Sun Microsystems	Sun Application Server
• Oracle	Oracle Application Server
• JBoss	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 ?

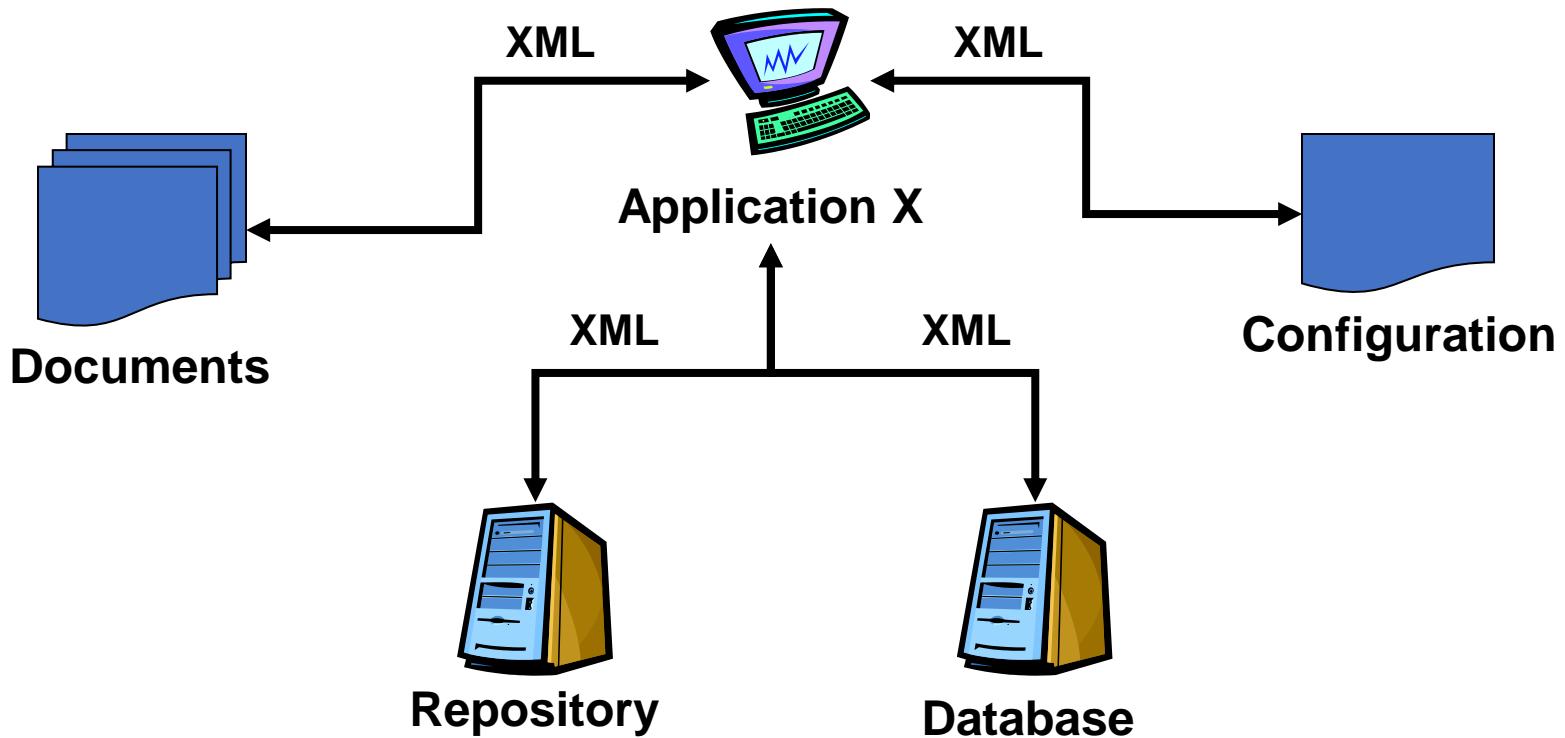
- In its simplest form, XML is a markup language for documents that contain structured data.
- The key tenet of XML is that it can be used to describe any data in a human-readable, structured form.
- Structured information can contain both content, as well as information that defines the content.
- XML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

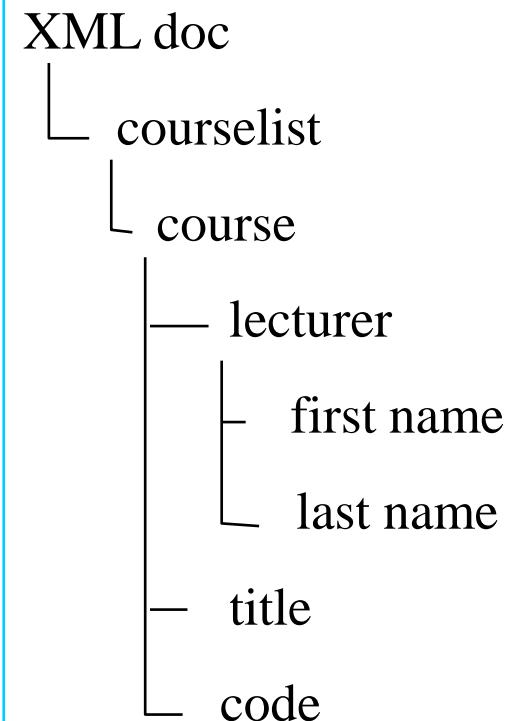
Universal Language

- 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

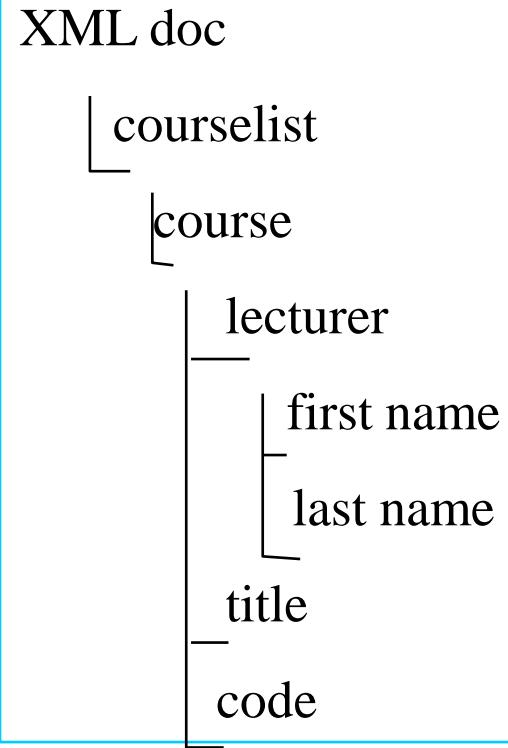
- Elements (or Tags): `<course></ course>`
 - Primary building blocks of an XML document
 - All tags must be closed and nested properly (as in XHTML)
 - Empty tags ok (e.g., `< course />`)
- Attributes: `< course level = "undergraduate">`
 - Additional information about an element
 - Attribute value 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 --&gt;
&lt;courselist xmlns="http://www.university.com" &gt;
&lt;course&gt;
  &lt;lecturer&gt;
    &lt;firstname&gt; H.T. &lt;/firstname&gt;
    &lt;lastname&gt; Shen &lt;/lastname&gt;
  &lt;/lecturer&gt;
  &lt;title&gt;SOA&lt;/title&gt;
  &lt;code&gt;INFS3204&lt;/code&gt;
&lt;/course&gt;
&lt;/courselist&gt;</pre>
```



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

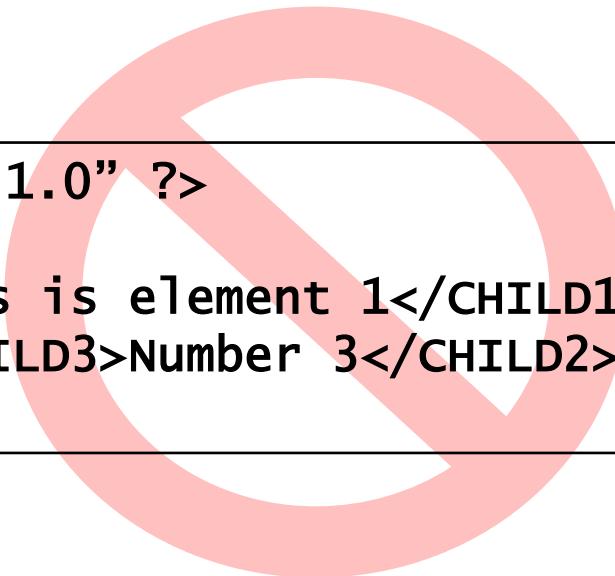
```
<courselist>
  <course>
    <lecturer>
      <firstname>H.T</firstname>

      <lastname>Shen</lastna
      me>
    </lecturer>
    <title>SOA</title>
    <code>INFS3204</code>
  </course>
</courselist>
```

- 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

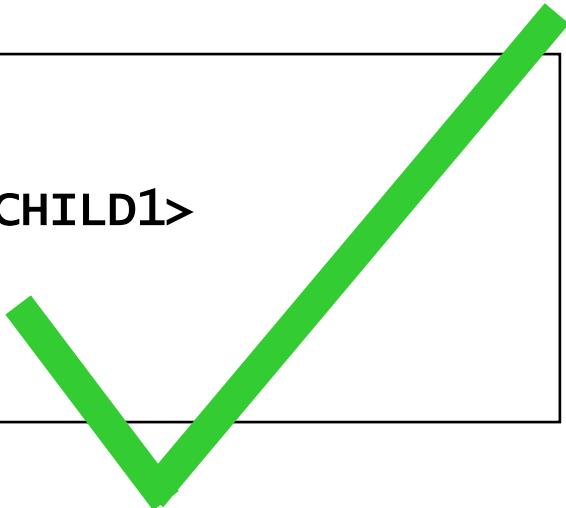
Well-formed XML 1?

```
<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>

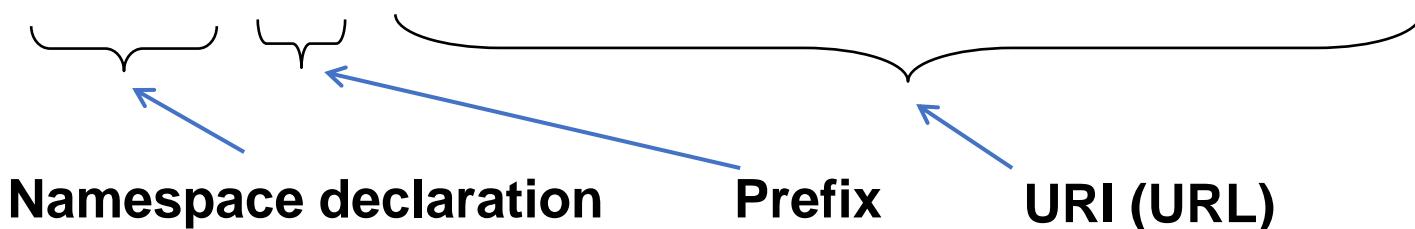
<f:table>
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>
```

In the example above, there will be no conflict because the two `<table>` elements have different names.

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/”
```

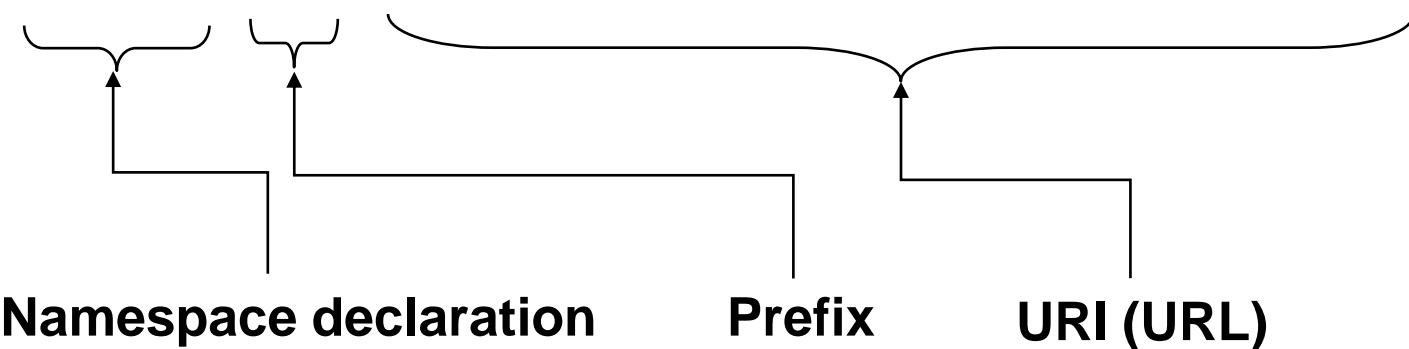


Namespaces: Declaration

```
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>
```

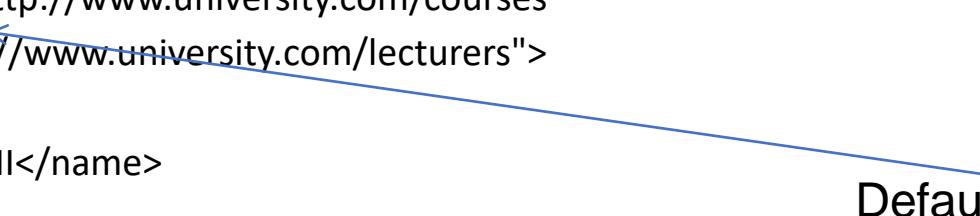
Another sample XML document

```
<?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:

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

Simple Elements in XSD

- The syntax for defining a simple element is:

```
<xsd: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:

```
<xss:element name="lastname" type="xs:string"/>
<xss:element name="age" type="xs:integer"/>
<xss:element name="dateborn" type="xs: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>
- <xssequence>
- <xselement name="Stocks"/>
- </xssequence>
- </xsccomplexType>
- </xselement>
- <xselement name="Stocks">
- <xsccomplexType>
- <xssequence>
- <xselement name="StockPurchase" minOccurs="0" maxOccurs="unbounded"/>
- </xssequence>
- </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&xslfile=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="zacz 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="zynergy 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>ZY SZ</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
Single, Universal Standard <ul style="list-style-type: none">▪ Web-friendly▪ Massive commercial support	Many binary formats, availability varies
Stored as a stream of Unicode, in a single file <ul style="list-style-type: none">▪ Organized as a tree▪ Cannot directly store 'native' numbers, must be encoded as unicode	Many organizations, <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]]  
  }  
}
```

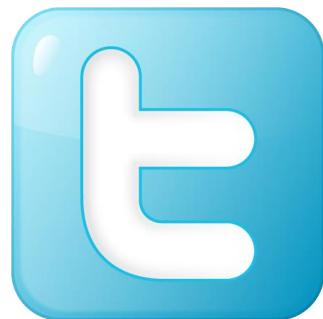
JSON Usage

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?

Lecture 8 : SOA and Web Services

This Week

- RMI, EJB and many other distributed computing frameworks suffer from many disadvantages.
- This week we'll look at Service Oriented Architecture which is an architecture proposed to solve many of these disadvantages
- Also we'll look at SOAP and REST as Web Service implementations

Issues with Traditional RPC

- The RPC frameworks we have discussed so far share a few common issues that tend to inter-relate
 - Tight coupling between client and server
 - Security problems:
 - Trust,
 - Firewalls
 - The Internet
 - Limited/non-existent interoperability between frameworks

Issue: Coupling

- Client and server in RPC are typically viewed as two parts in one (distributed) application
 - Stubs/Skeletons are generated from the same IDL file / interface
 - Marshalling/Serialization is technology dependent
 - Implicitly creates a coupling between client and server

Issue: Trust and Firewall Security

- Trust issues:
 - The server shares its information with the client
 - The client can compromise the server
- Firewall security: The RPC frameworks advocate assigning each server component with its own port
 - Follows good network protocol design - each different service has its own port (e.g. ftp = port 21, http = port 80)
 - Firewalls are then configured to block access to dangerous/risky ports to minimise the risk of attacks

Issue: Internet Security

- When the RPC is only internal to a corporate network, setup and security is less of an issue
 - Physical + login security cuts out most attack vectors
- But what if we must communicate over the Internet?
 - Need to open a ‘hole’ in the firewall at the *gateway*, one hole for each server component inside the network
 - Network administrators are *very* reluctant to do this!
 -

Issue: Interoperability

- Most RPC frameworks don't interoperate with other frameworks
 - Almost entirely down to incompatible communication protocols and message formats
 - E.g. IIOP for CORBA vs. MSRPC for DCOM,
 - Each protocol is tailored to the features of the framework it was designed for
 - Presently, RPC became well-understood enough to define stable protocol and message format standards

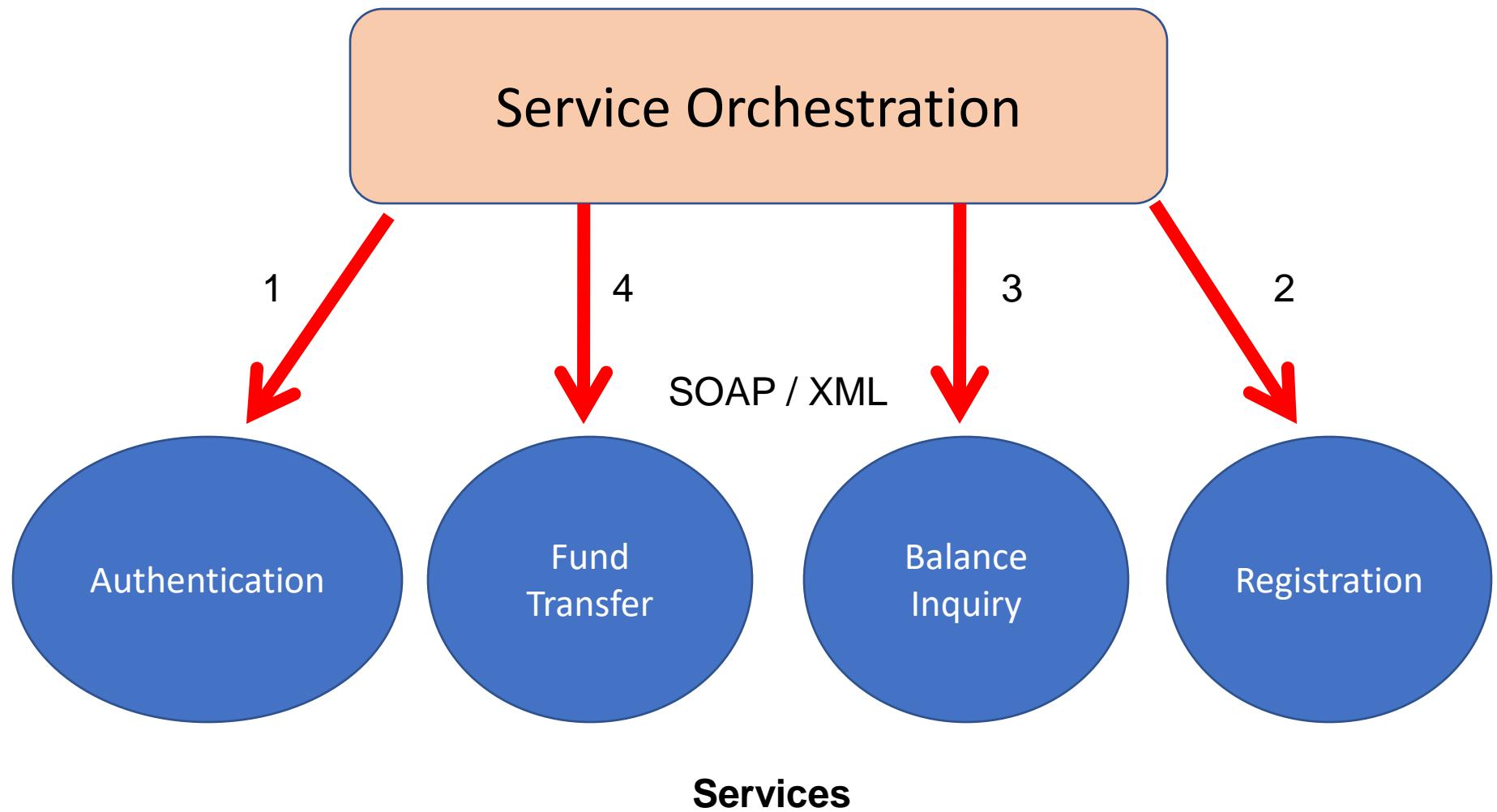
Software Architectures

- “The software architecture of a program or computing system is the structure or structures of the system, which include software components and the relationships among them.”
- In other words, software architecture describes the system’s components and the way they interact at a high level.
- Service-oriented architecture is a special kind of software architecture that has several unique characteristics

What is SOA?

- Service-oriented architecture (SOA) is an architectural style where existing or new functionalities are grouped into atomic services.
- SOA is commonly thought as an architecture that builds **loosely coupled, interoperable, Standard based** components called services.
- They typically implement functionalities most humans would recognize as a service
 - Filling out an online application for an account
 - Viewing an online bank statement
 - Placing an online book or airline ticket order.

What is SOA?

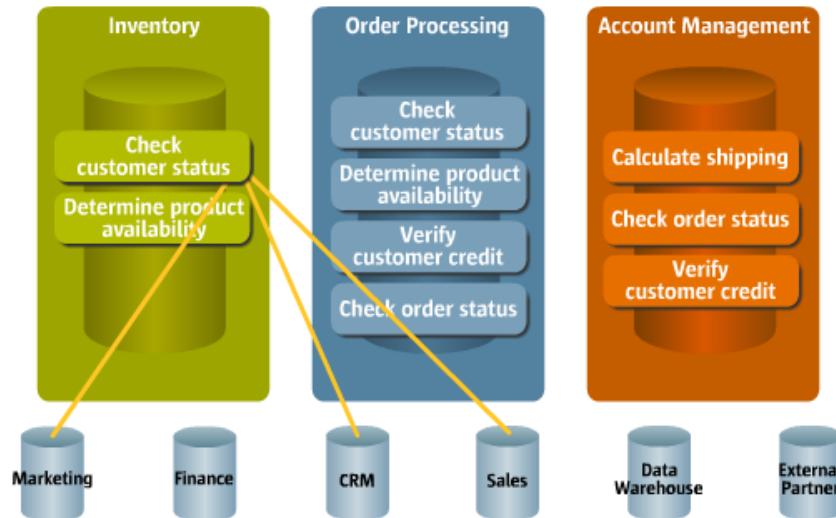


What is SOA?

- They have no calls to each other embedded in them.
- Instead of services embedding calls to each other in their source code, protocols are defined which describe how one or more services can talk to each other.
- This architecture then relies on a business process expert to link and sequence services, in a process known as **orchestration**, to meet a new or existing business system requirement.

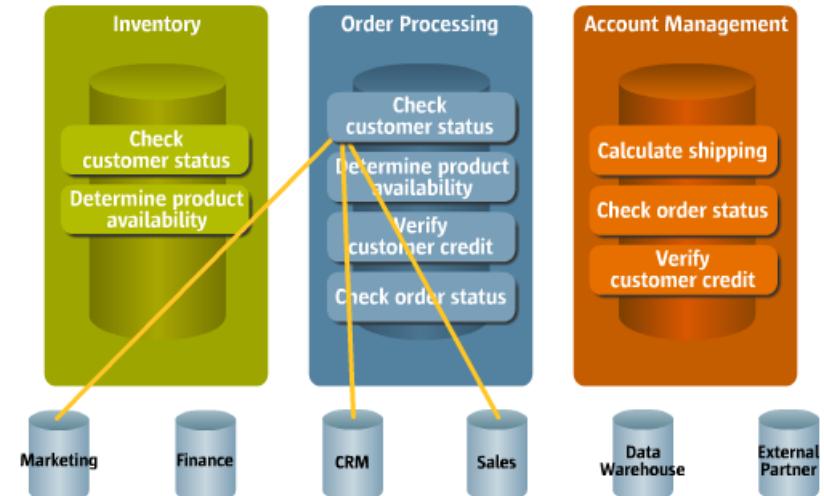
Traditional Distributed Systems

- Functions are duplicated



Order Processing also
needs checking
customer status

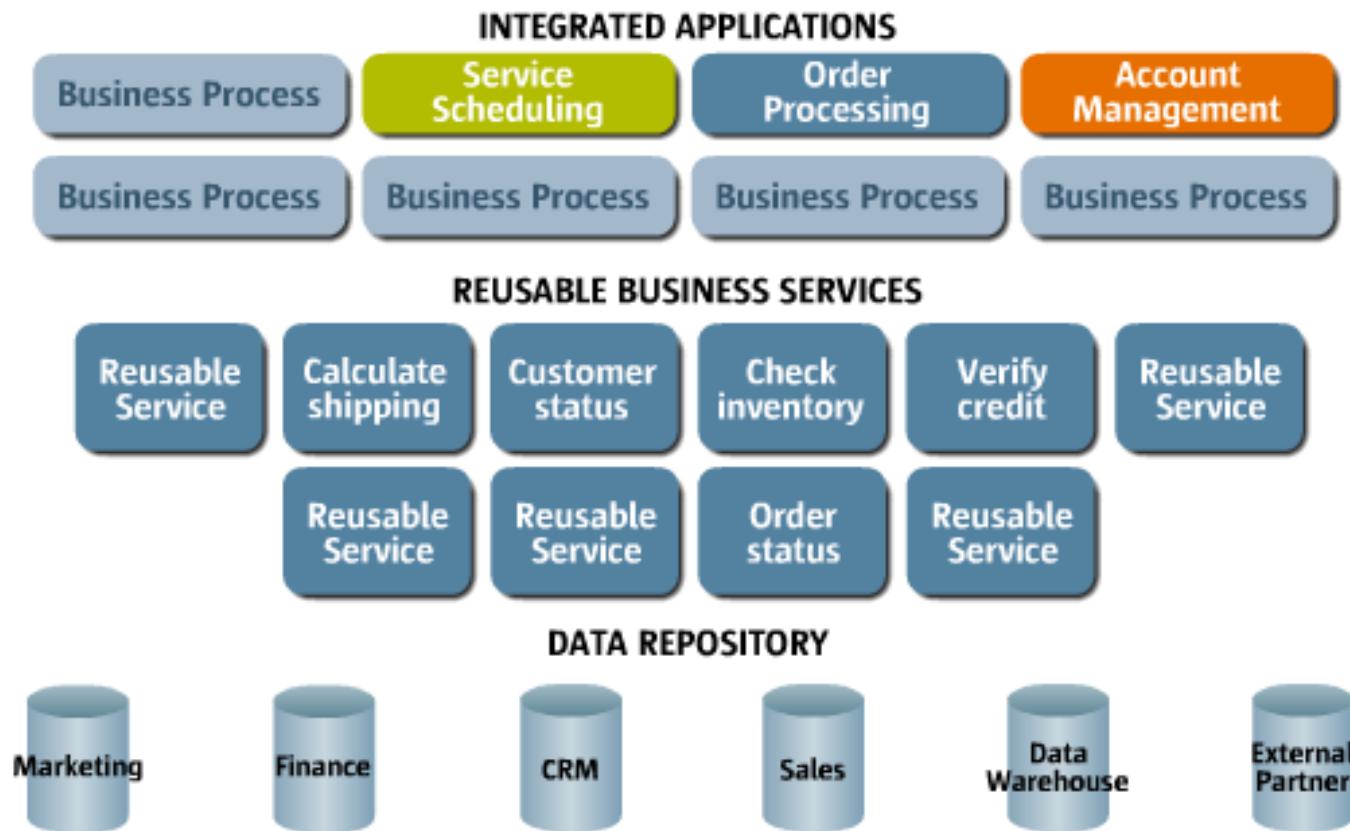
Inventory Processing
needs checking
customer status



Traditional Distributed Systems

- Low reusability
 - If you try to reuse lot of cross references between sub-systems.
- Adding a new function is difficult
 - Develop everything from the beginning
- Function inconsistency
 - Development of “Checking customer status” can be different from sub-systems to sub-system.

SOA Style

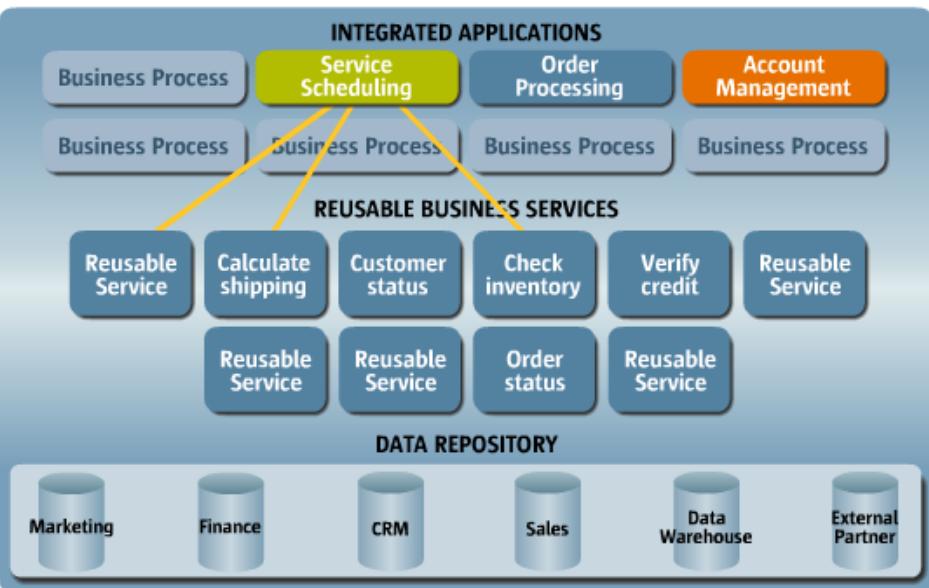


New ERP System is a reusable collection of services, that can be composed into Integrated Applications.

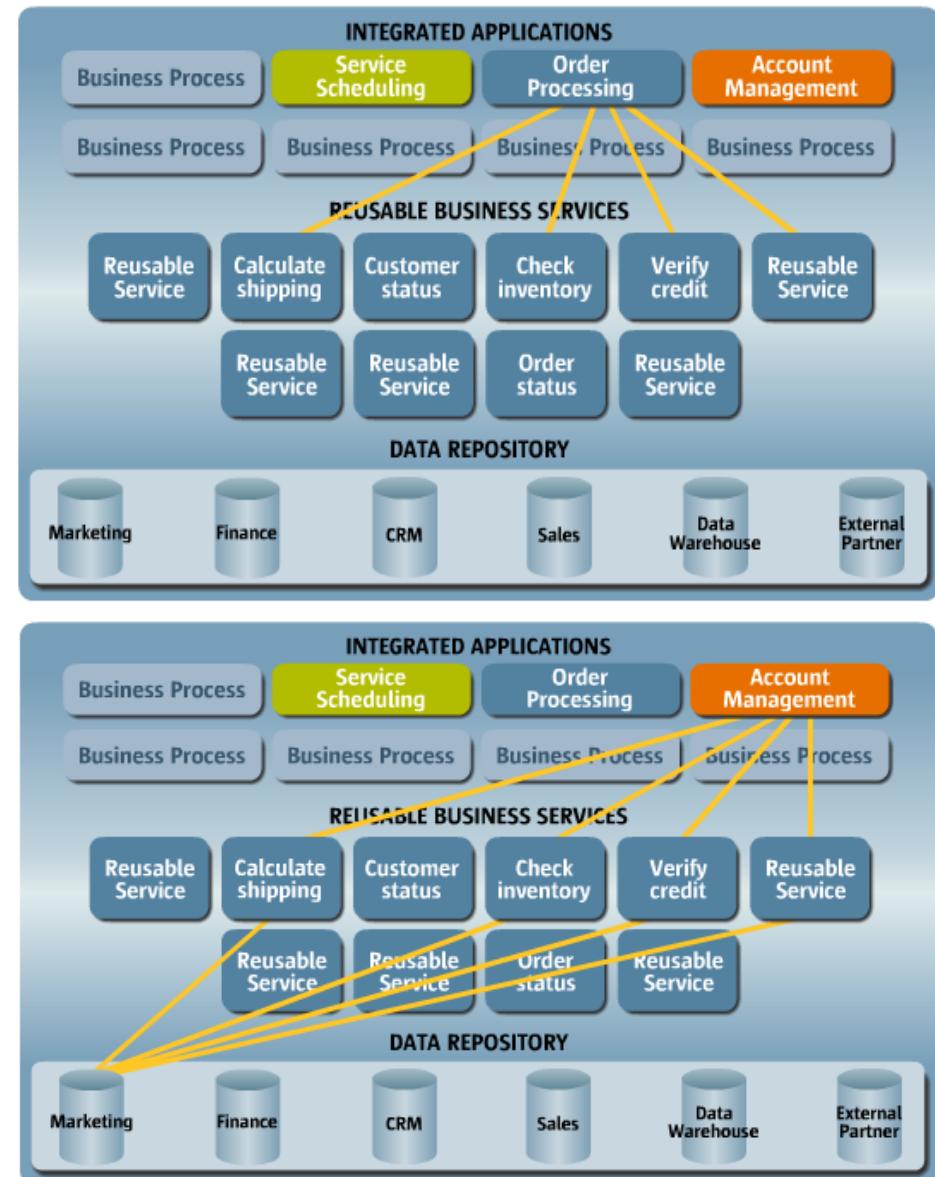
SOA Style

Order Processing Application

Service Scheduling Application



Management Report
Generation Application



SOA

- What distinguishes SOA from other architectures is loosely coupling.
- In loosely coupled systems the client of a service is essentially independent of the service.
- The way a client communicate with the service is not dependent of the service implementation.
- The client communicates with service according to a specified, well-defined interface.

What Happens in SOA?

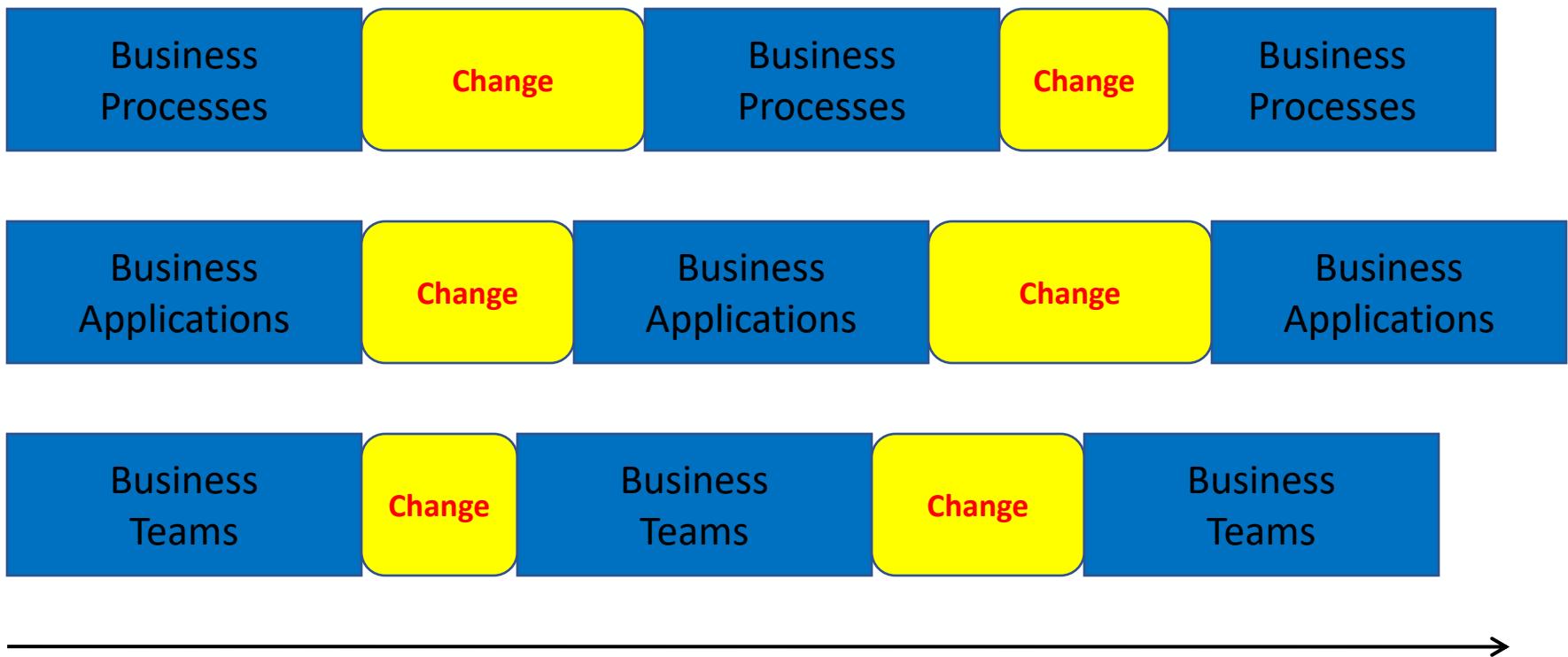
- Traditional
 - Services and service processing logic is mixed up in the code.
- SOA
 - Try to separate business process and services
 - Example : Data access layer detach the data management functionality out of application programs. Similarly we try to detach business process and the services.

Why SOA?

- Accommodate rapid changes to the IT landscape in relation to the changes in the Business environments.
- Promotes reuse of services across multiple business process automations.
- Simple dynamic interfacing of services.
- The services can be discovered and interfaces can be changed without major changes to applications.

Why SOA for Businesses

Manage CHANGE



Time

Characteristics of SOA

- **Loose Coupling** - client can discover server's supported protocols/formats and negotiate communication semantics
- **Reusable** – similar to object-oriented orientation
- **Autonomous** - runs independently of other systems
- **Stateless** - no ongoing commitment between client & service
- **Composable** - one service can contain another
- **Standards-based** - interoperability among SOA services
- **Contract-based** – i.e. uses interfaces
- **Fine-grained** - services should be small (higher cohesion)
- **Reusable, modular** - another way of saying ‘fine-grained’
- **Encapsulation** - information hiding
- **Heterogeneous** – technologies, platforms, applications, etc.
- **Location transparent**

Implementing SOA

- SOA can be implemented using many technologies:
 - Web services
 - RPC
 - CORBA
 - DCOM
 - SOAP
 - WCF (Windows Communication Foundations) Part of .NET framework.
 - REST (Web API)

Where to Use SOA?

- SOA is most useful for what it was designed for:
 - When crossing *platform* boundaries
 - When crossing *trust* boundaries
- Business logic that change frequently and highly reusable is more eligible for SOA.
 - E.g. Payment requests, Balance inquiries

Where *NOT* to Use SOA

- SOA isn't applicable everywhere. It's poor for:
 - **Non-distributed applications**
 - Applications with a **short deployed lifetime**
 - **Asynchronous communication** between servers
 - **Interactive GUI applications**
 - **A homogenous application environment**

A Web Service

- A service that is accessible over a web protocol
- Well defined interface - protocols define the interaction between the client and the server

Why Web Services?

- A Service accessible over a web URL!
 - Reusable functionality
 - Business to business integration
 - Information sharing
 - Business process automation
- Innovation - offer different services

Perform Web Service Invocation

What will you learn?

- Service Invocation

Hands-on

1 - Go to <http://openweathermap.org/>

2 - Read the documentation

3 - Signup and get a key

4 - Try to read weather by giving longitude and latitude

http://api.openweathermap.org/data/2.5/weather?lat=35&lon=139&APPID=your_key

5 - Try to read weather in Colombo

Web Services Everywhere!

- Grid computing
 - [SETI@home](#)
- Cloud computing
 - IaaS - AWS
 - SaaS APIs - Salesforce APIs, Netsuite APIs, PeopleHR API
- Google Maps APIs

Web Services & SOAP

SOAP

What does it stand for?

- Simple Object Access Protocol

What is it?

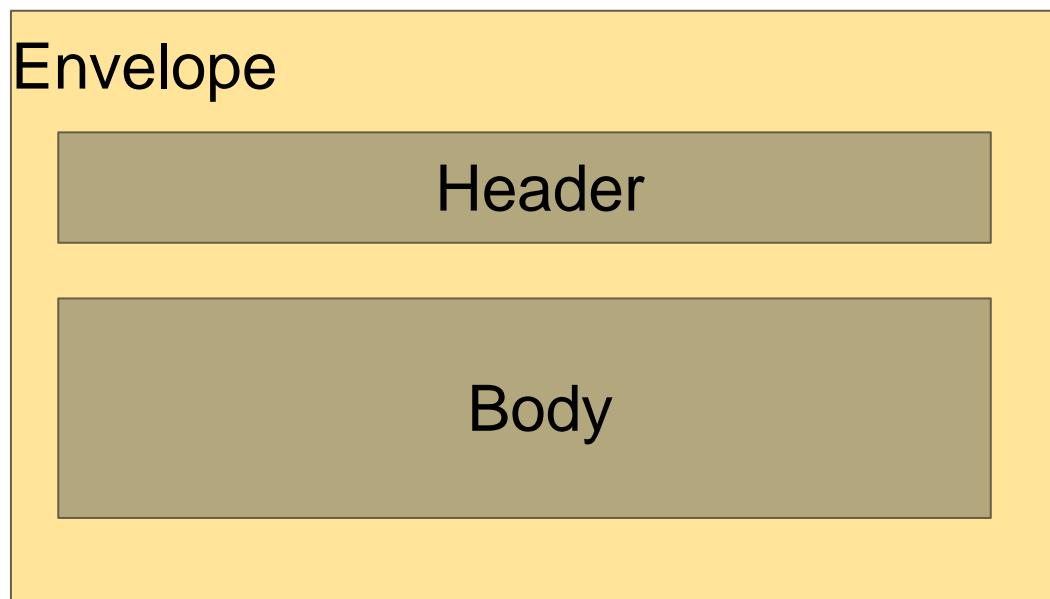
- Two versions - SOAP 1.1 and SOAP 1.2
- SOAP 1.2 became a W3C recommendation in 2003

Who/where/when?

- Initiated by IBM, Microsoft

SOAP Basics

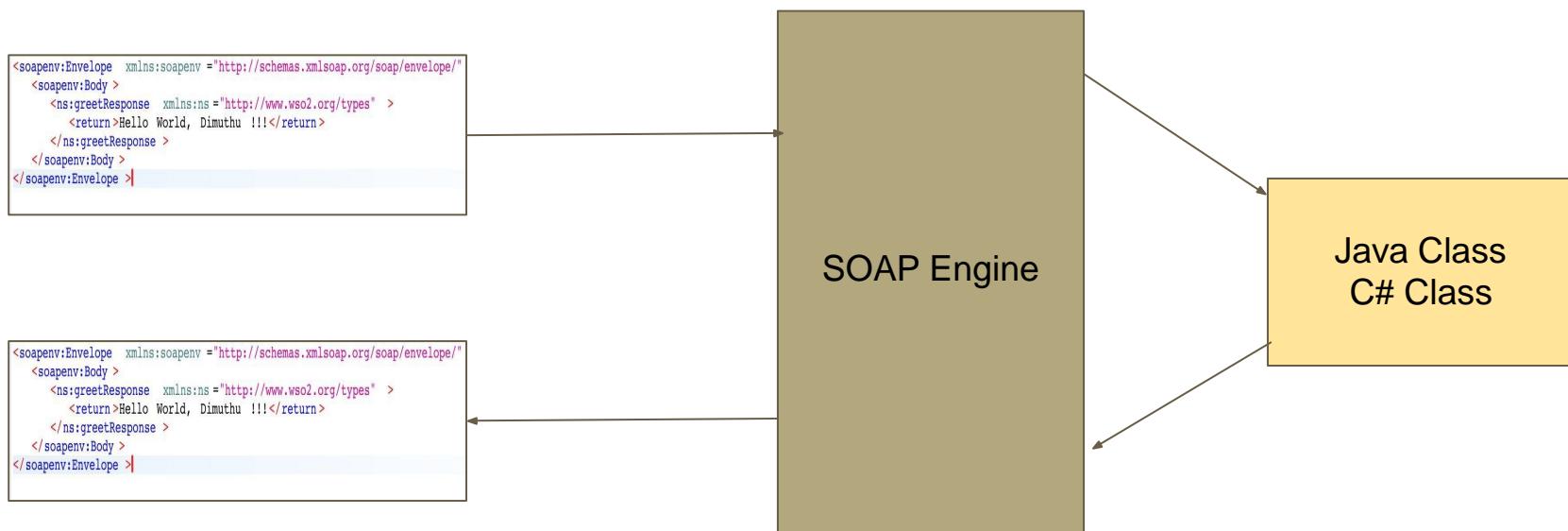
- Relies on XML and defines a message structure
- Can run on any protocol HTTP, SMTP



Sample SOAP Message

```
<soapenv:Envelope xmlns:soapenv = "http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <ns:greetResponse xmlns:ns = "http://www.wso2.org/types">
      <return>Hello World, Dimuthu !!!</return>
    </ns:greetResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP Engine



WSDL

- Web Service Description Language - WSDL1.1 & 2.0
- Describes a web service using XML
 - Uses XML Schema
 - Input message
 - Output message
 - Transports
 - Versions

SOAP and WSDL - A strong marriage

```
- <wsdl:definitions targetNamespace="http://www.wso2.org/types">
  <wsdl:documentation>HelloService</wsdl:documentation>
- <wsdl:types>
  - <xsschema attributeFormDefault="qualified" elementFormDefault="unqualified" targetNam
    - <xss:element name="greet">
      - <xss:complexType>
        - <xss:sequence>
          <xss:element minOccurs="0" name="name" nillable="true" type="xss:string"/>
        </xss:sequence>
      </xss:complexType>
    </xss:element>
  - <xss:element name="greetResponse">
    - <xss:complexType>
      - <xss:sequence>
        <xss:element minOccurs="0" name="return" nillable="true" type="xss:string"/>
      </xss:sequence>
    </xss:complexType>
  </xss:element>
</xss:schema>
</wsdl:types>
- <wsdl:message name="greetRequest">
  <wsdl:part name="parameters" element="ns:greet"/>
</wsdl:message>
- <wsdl:message name="greetResponse">
  <wsdl:part name="parameters" element="ns:greetResponse"/>
</wsdl:message>
```

Perform SOAP Service Invocation

What will you learn?

- Service Invocation
- Self contained functionality
- Service interface

WS* Specifications

- WSDL 2.0
- WS Security
- WS Addressing
- WS Policy
- WS Trust

REST

REST

What does it stand for?

- REpresentational State Transfer

What is it?

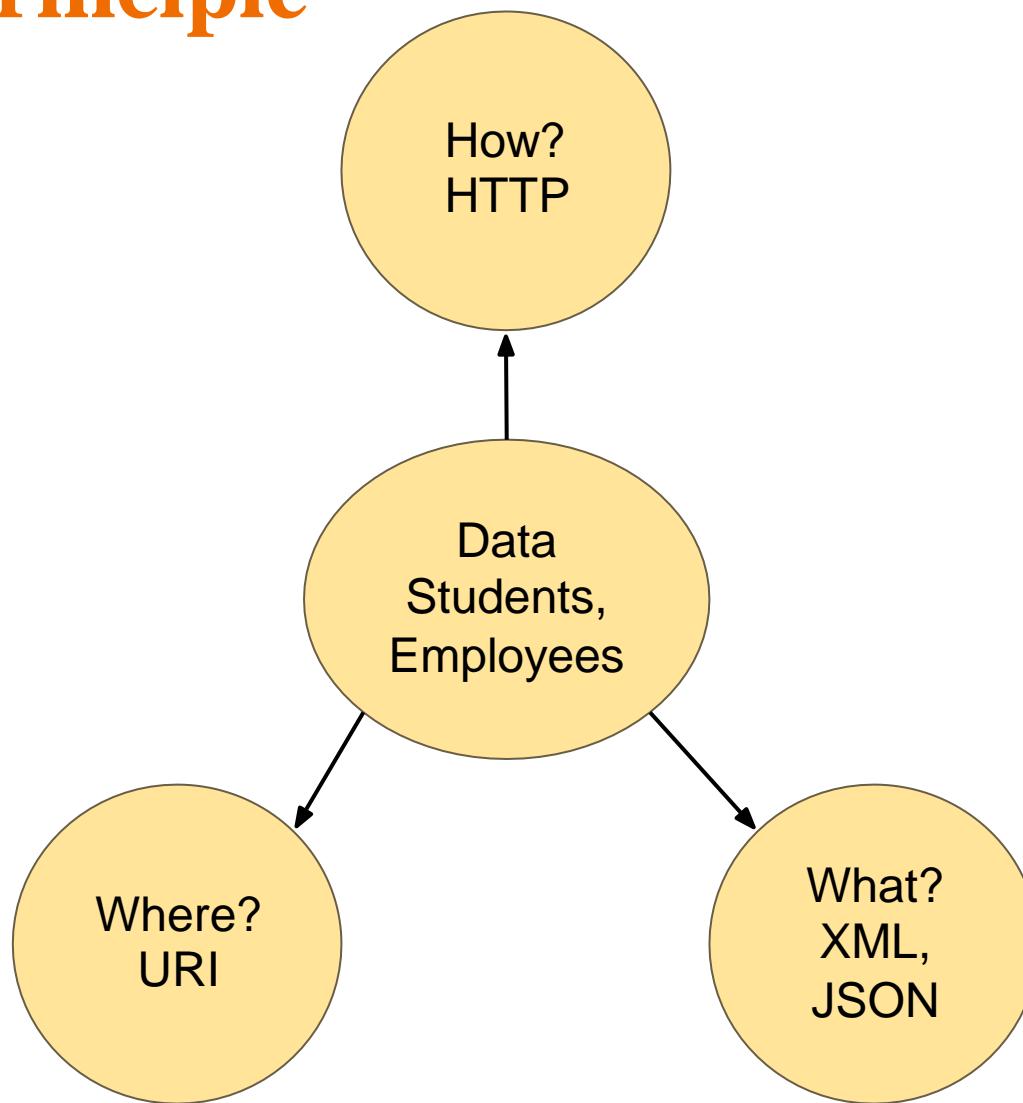
- Architectural pattern - not a standard

Who/where/when?

- Roy Fielding in 2001



REST Principle



Stateless

- No state stored on the server
- Every HTTP request executes in complete isolation
- Simpler to design and evolve
- Easier to scale

REST - Methods

- Defines the action taken with a URL
- Proper RESTful services expose all four

HTTP Method	Action	Example
POST	Create	http://wso2.com/general/dbusers/user/
GET	Read	http://wso2.com/general/dbusers/users http://wso2.com/general/dbusers/user/sam
PUT	Update or Create	http://wso2.com/general/dbusers/user/sam
DELETE	Delete	http://wso2.com/general/dbusers/user/sam

URIs - Addressability

- Name, address and version of resource
- Self-descriptive
- Unique URIs are exposed for every resource from RESTful system
 - URI per resource
- URIs are discoverable by clients

Data Representation

- Can be
 - XML, JSON, HTML
- Content negotiation based on HTTP headers
 - Accept or Content-Type
- Query parameters
 - GET /v1/employees/123?format=json
- URI extention
 - GET /v1/employees/123.xml

An Example REST

- In [http://ip-api.com/json/\[ip_address\]](http://ip-api.com/json/[ip_address]) service
 - Get the location of an Ip Address
 - Content type - Application/JSON
 - Send HTTP GET
-

REST Implementation

REST - Interface Description

- Swagger- Also known as OpenAPI specification
 - Interface description language for describing, producing, consuming and visualizing RESTful web services
- YAML based
- Allows both humans and machines to understand
- Goal - Update client and documentation at the same time as the server

Swagger

```
"paths": {  
    "/": {  
        "get": {  
            "operationId": "listVersionsv2",  
            "summary": "List API versions",  
            "produces": [  
                "application/json"  
            ],  
            "responses": {  
                "200": {  
                    "description": "200 300 response",  
                    "examples": {  
                        "application/json": "{\n                            \"versions\": [\n                                {\n                                    \"version\": \"1.0.0\",  
                                    \"status\": \"stable\"  
                                },  
                                {\n                                    \"version\": \"2.0.0\",  
                                    \"status\": \"beta\"  
                                },  
                                {\n                                    \"version\": \"3.0.0\",  
                                    \"status\": \"alpha\"  
                                }  
                            ]  
                        }  
                    }  
                }  
            }  
        }  
    }  
}
```

Swagger

The screenshot shows the Swagger UI interface running at `localhost:27262/swagger/ui/index#!/Values/Values_Get`. The top navigation bar includes the Swagger logo, a search bar with URL `http://localhost:27262/swagger/docs/v1`, an `api_key` input field, and an `Explore` button.

The main content area is titled **MyAPI**. It displays the **Account** and **Values** sections. The **Values** section is currently active, showing the **GET /api/Values** operation. The response class is defined as `[{"string"}]`. The response content type is set to `application/json`.

Below this, four other operations are listed: **POST /api/Values**, **DELETE /api/Values/{id}**, **GET /api/Values/{id}**, and **PUT /api/Values/{id}**.

At the bottom, a note indicates the base URL and API version: `[BASE URL: , API VERSION: V1]`.

Web APIs

Web APIs

What is it?

- Not a standard. Not an architecture pattern. Just a “term”.
- Concentrating on the **accessibility** of services.
 - Secured (access controlled), open and monitored services
- A business capability delivered over the Internet to internal/external consumers

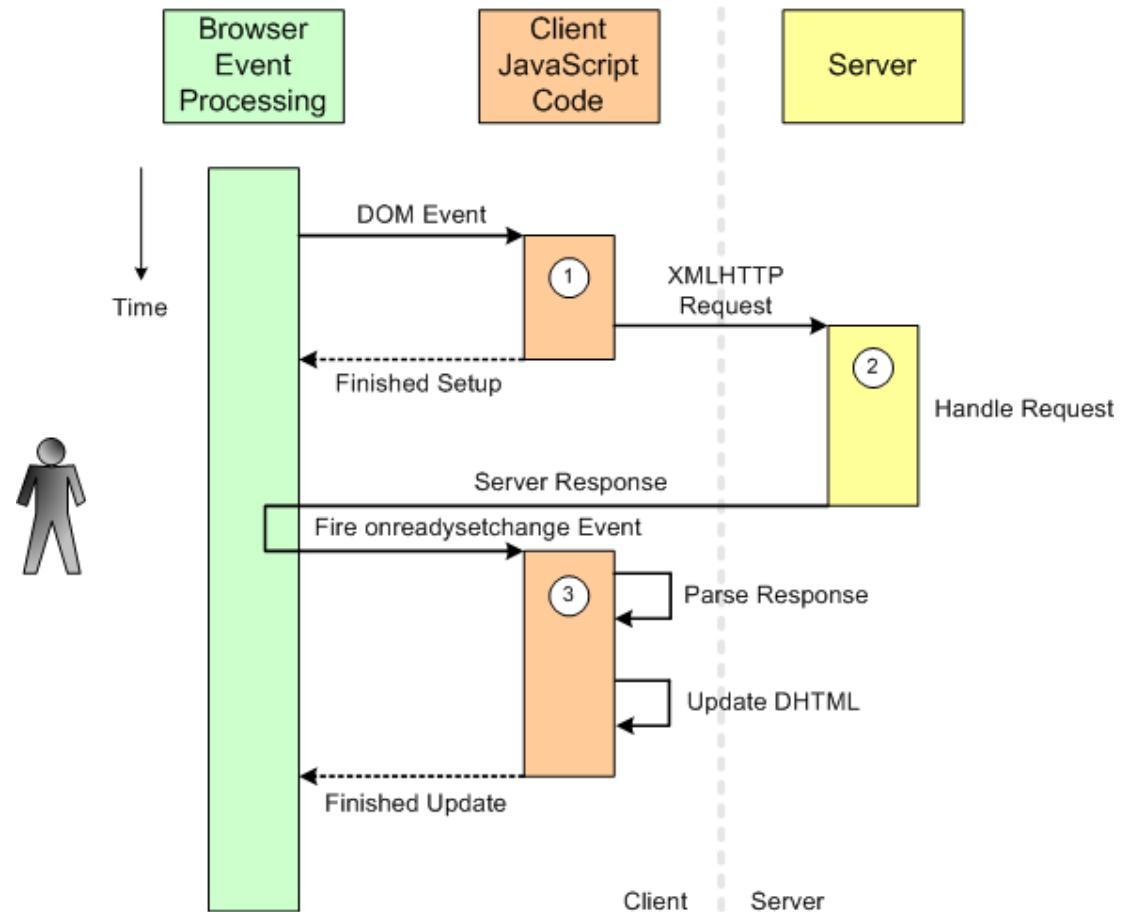
API = Service + Security + Documentation

Consuming REST Services - AJAX

Ajax

- Ajax stands for Asynchronous JavaScript And XML
 - Convergence of a few disparate technologies that together facilitate rich Web browser GUIs via client-side scripting
 - Term 'Ajax' was coined to describe their use together
 - JavaScript: scripting language (ie: interpreted on the fly at run-time) for client-side processing in Web browsers
 - Asynchronous: Built-in browser support for sending *arbitrary* messages *asynchronously* to a server via JavaScript
 - XML: General-purpose data document format; Web browsers have built-in XML parsers for rendering HTML

Ajax Sequence Diagram



XMLHttpRequest Class

- Methods of XMLHttpRequest object:
 - **open** - sets the URL for submitting (sending) the request to
 - **setRequestHeader** - Add/set headers, usually just Content-Type
 - **send** - accepts the text of the message contents and sends it
- Properties of XMLHttpRequest object:
 - **onreadystatechange** - pointer to completion callback function.
Called every time the readyState changes. Note: lower case!
 - **readyState** - Callback state (see previous slide)
 - **status** - Call success/failure (200=success, others are error codes)
 - **responseText** - Raw message text from server
 - **responseXML** - XML parser object attached to responseText

```

function AddRPCAsync_SOAP12(onCompletionFn) {
    req = null;

    if (window.XMLHttpRequest != undefined)
        req = new XMLHttpRequest();                                ← Firefox and compatible
    else
        req = new ActiveXObject("Microsoft.XMLHTTP");          ← Internet Explorer

    req.onreadystatechange = onCompletionFn;

    req.open("POST", "http://localhost/WebServices/Calculator.asmx", true); ← No .asxm/Add here!
    req.setRequestHeader("Content-Type", "application/soap+xml");      ← Set up header(s)

    req.send("<?xml version=\"1.0\" encoding=\"utf-8\"?> \
              <soap12:Envelope xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\" \
                xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\" \
                xmlns:soap12=\"http://www.w3.org/2003/05/soap-envelope\"> \
                  <soap12:Body> \
                    <Add xmlns=\"http://www.curtin.edu.au/SPD361/\"> \
                      <operand1>8</operand1> \
                      <operand2>4</operand2> \
                    </Add> \
                  </soap12:Body> \
                </soap12:Envelope>");
}

function AddRPC_SOAP_OnCompletion() {                                ← Same as SOAP 1.1
    if (req.readyState == 4) {
        if (req.status == 200) {
            var ndResult = req.responseXML.documentElement.getElementsByTagName("AddResult")[0];
            alert(ndResult.childNodes[0].nodeValue);           ← Access result (<AddResponse>) via DOM
        }
        else
            alert("Asynchronous call failed. ResponseText was:\n" + req.responseText);
    }
    req = null;
}

```

Ajax + Web Service Example (SOAP 1.2)

Calling a REST services with AJAX + JQUERY

```
$.ajax({  
    type: "GET",  
    dataType: "jsonp",  
    url: "http://localhost:8080/restws/json/product/get",  
    success: function(data){  
        alert(data);  
    }  
    error: function(data){  
        alert('error');  
    }  
});
```

AJAX in JQuery

- `$.get(url [, data] [, success(data,textStatus,jqXHR){})`

```
$.get( "ajax/test.html", function( data ) {
    $(".result").html( data );
    alert( "Load was performed." );
});
```

- `$.post(url [, data] [, success(data,textStatus,jqXHR){})`

```
$.post( "ajax/test.html", postdata, function( data ) {
    $(".result").html( data );
});
```

- `$.getJSON(url [, data] [, success(data,textStatus,jqXHR){})`

- Use an AJAX get request to get JSON data

REST and SOAP Implement SOA

Summary

- Service Oriented Architecture
- Web Services
- REST
- APIs & Microservices

Reuse

Many technologies for implementing a SOA.

None is perfect.

All achieve data sharing,
modularity, agility, reuse
and
innovation!

Questions?

Lecture 9 - Service Integration, Orchestration and Governance

Integration

Plumbing different software applications/services/systems and forming new software solutions is known as ‘Enterprise Integration’.

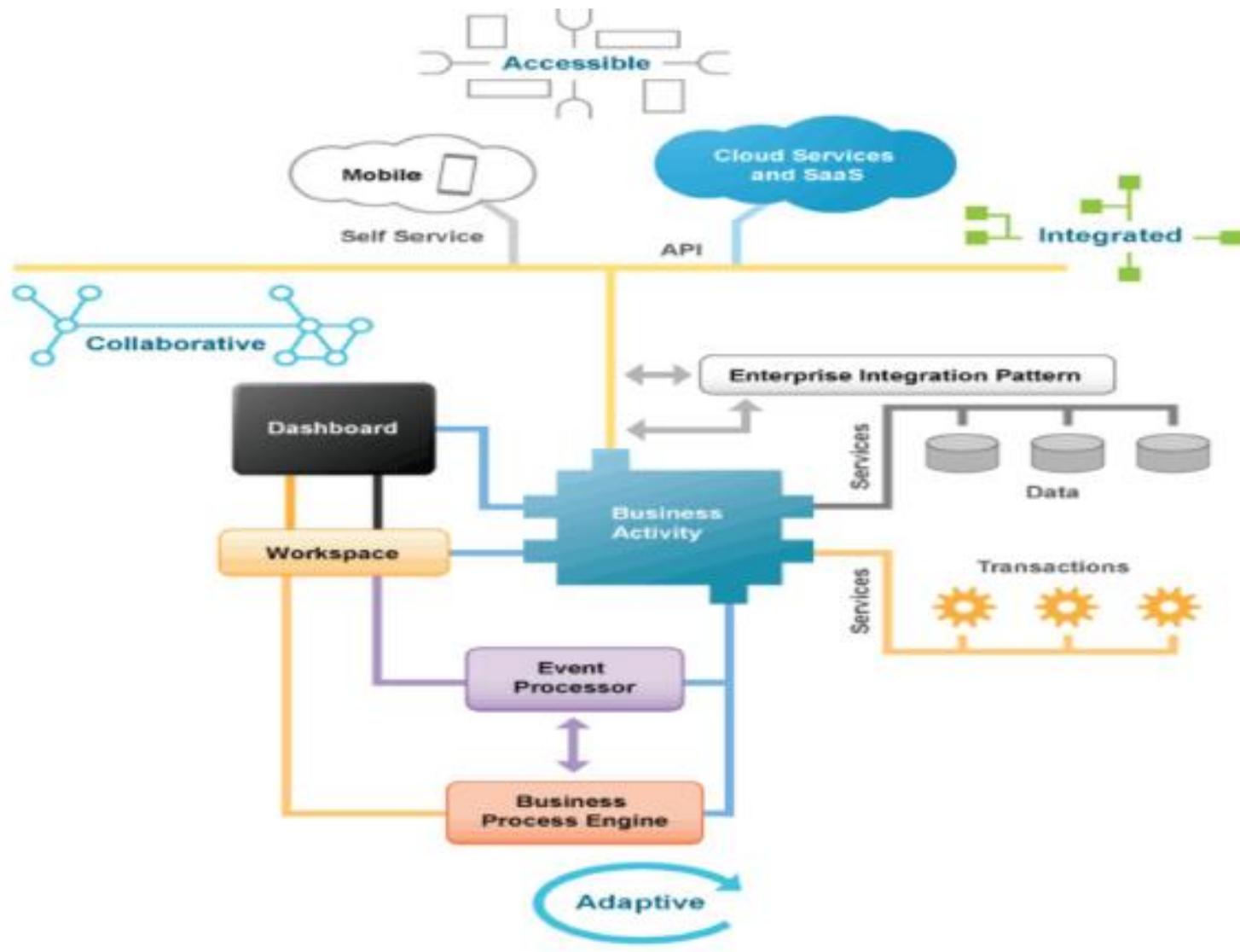
When SOA Integration is Used?

- Build new applications
- Expose a business function
- Reuse services to build new processes
- Business process automation
- Integrate data for analytics

Importance of Integration in SOA

- Enterprises heavily rely on the underlying software systems/services/applications.
- Disparate technologies and platforms
- No single solution or a vendor
- Diverse Business requirements

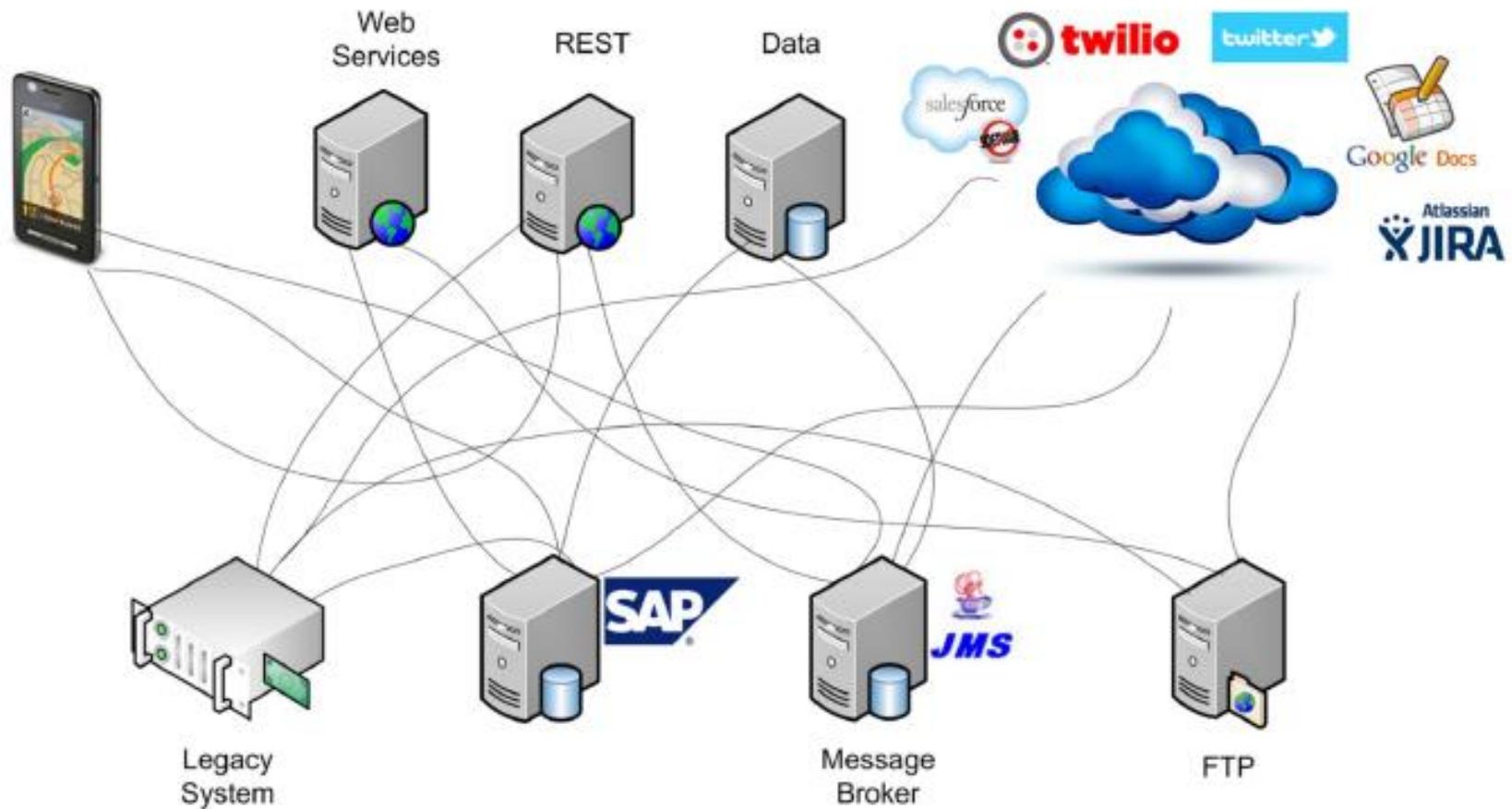
An Example



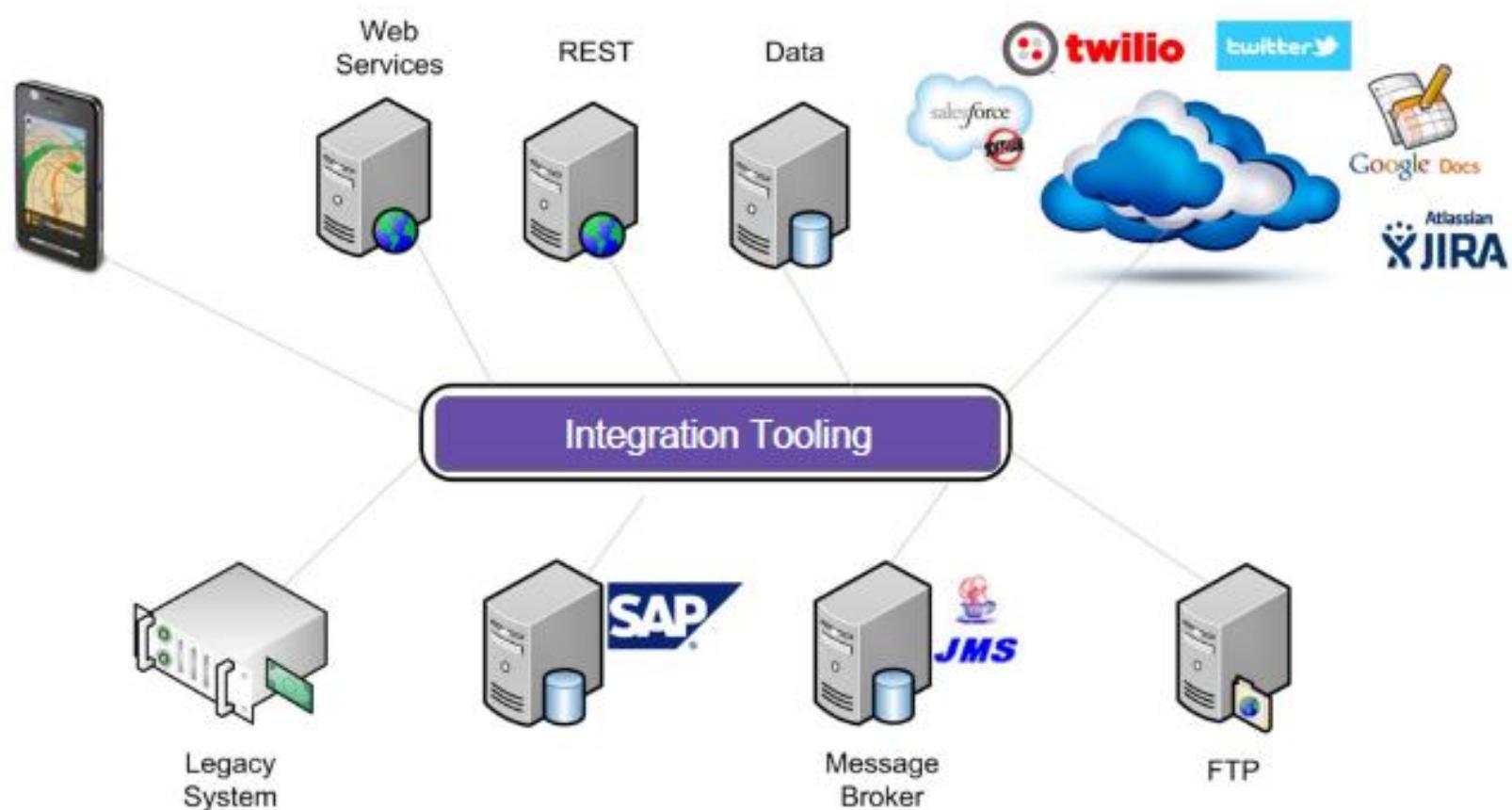
Challenges of Integration

- Heterogeneity: Disparate systems, protocols and standards
- Variety: Legacy systems, SOAP/REST services, Cloud APIs
- Disorganized: Spaghetti architecture, poorly managed
- Costly: Hardly scalable and maintainable
- Unquantifiable: Difficult to measure throughput & productivity

Unplanned Integration



Integration Tooling

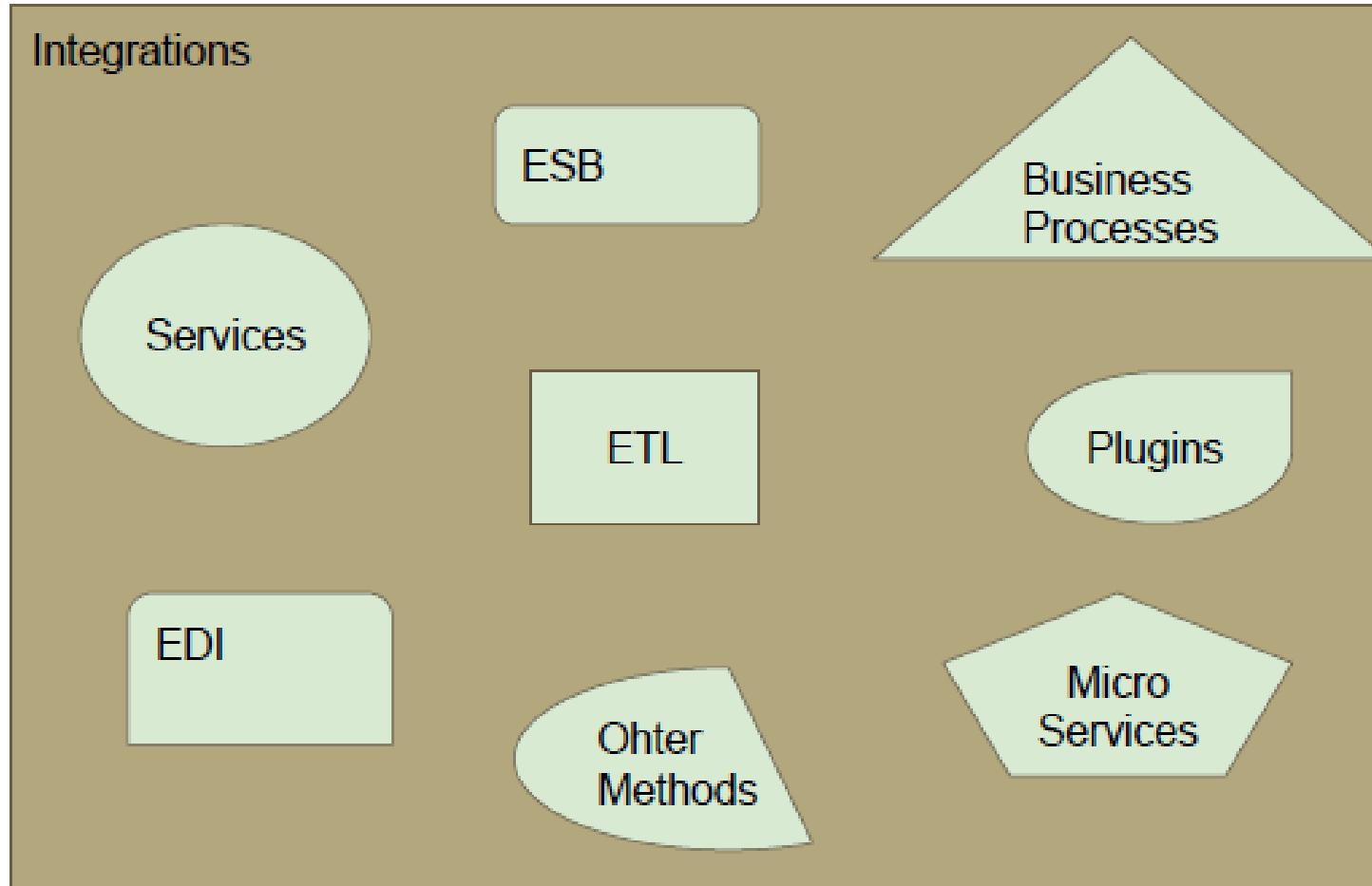


ESB

Implements a communication system between mutually interacting Software applications/services in a service-oriented architecture

- Routes
- Transforms
- Mediation

Integration Tooling - SOA and Non-SOA



ESB - Meets SOA Integration Challenges

- Transports: Support for web (HTTP), files (VFS), e-mail (POP, IMAP) and more..
- Formats/ Protocols: XML, JSON, CSV, EDI, SOAP, REST and more..
- Domain specific apps: Financial Services (FIX), Healthcare (HL7)..
- COTS: SAP, IBM WebSphere MQ, MSMQ and more..
- Cloud apps: Salesforce, Google Apps, Twitter, JIRA and more..
- Custom extensions: Handles proprietary/ non-standard integration cases

Hands On

Using an ESB for Integration

Enterprise Integration Patterns

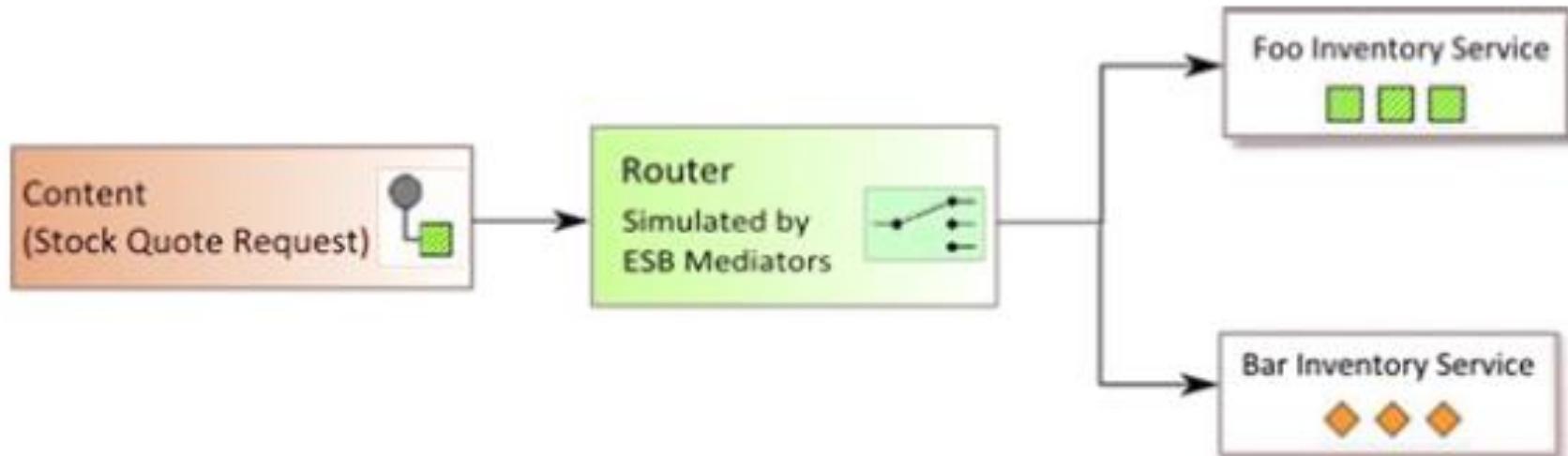
- A design Pattern - A pattern that keeps occurring
- Solution architects over the years found some patterns that kept recurring

<http://www.enterpriseintegrationpatterns.com/books1.html>

- Gregor Hohpe published a book - Enterprise Integration Patterns : Designing, Building, and Deploying Messaging Solutions
- Contains 65 integration patterns

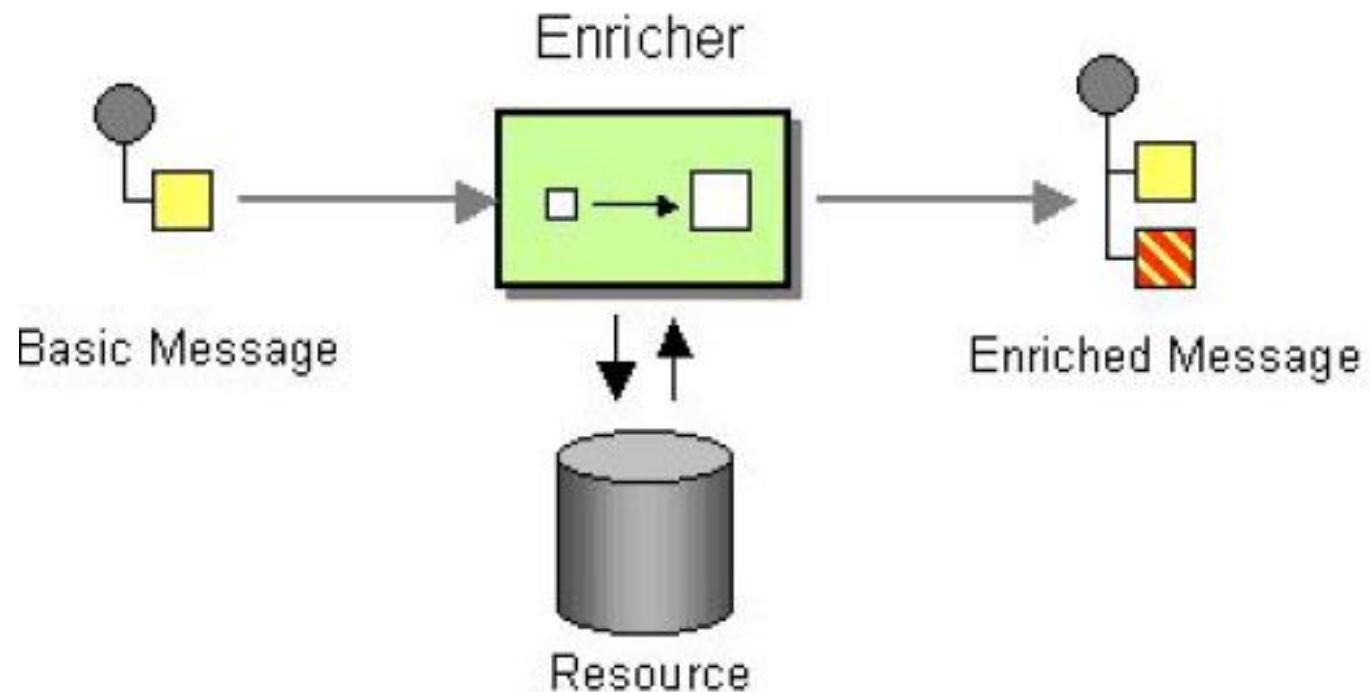
Enterprise Integration Patterns

Content Based Routing



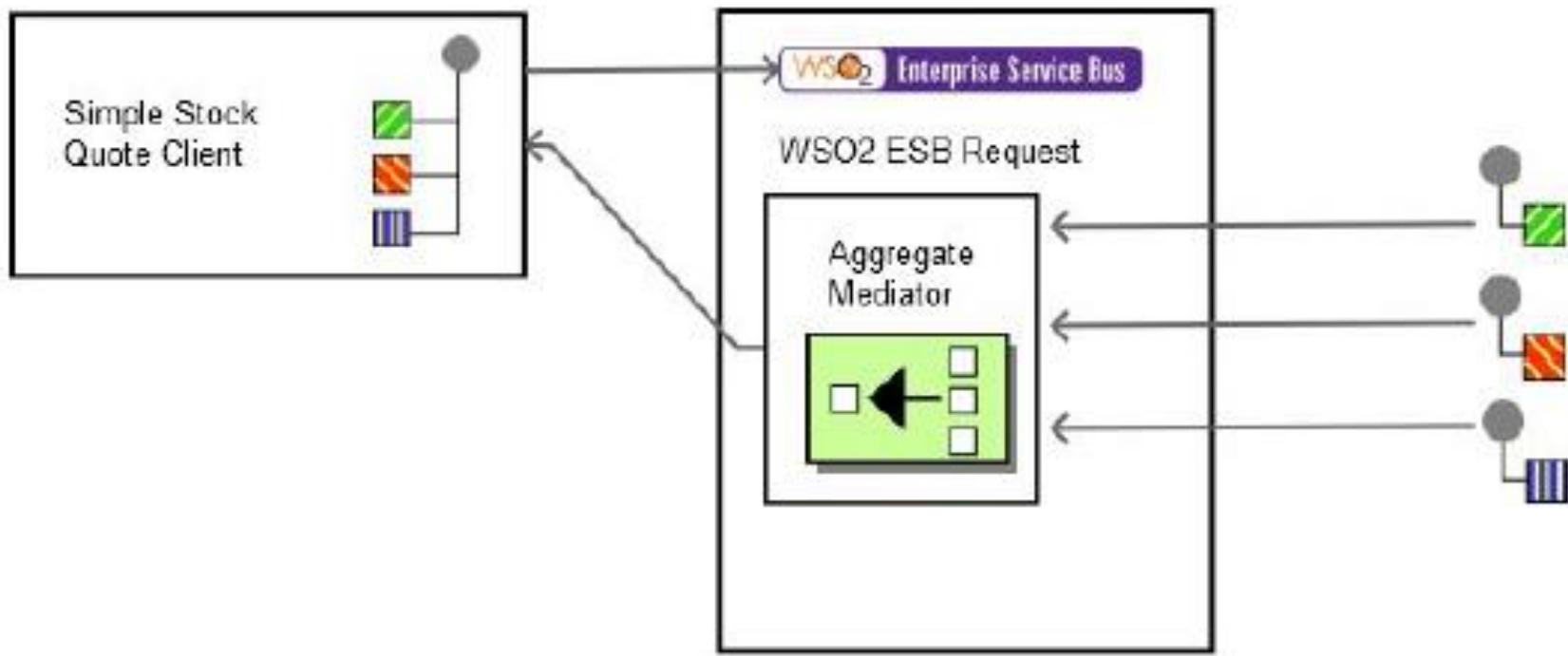
Enterprise Integration Patterns

Enricher



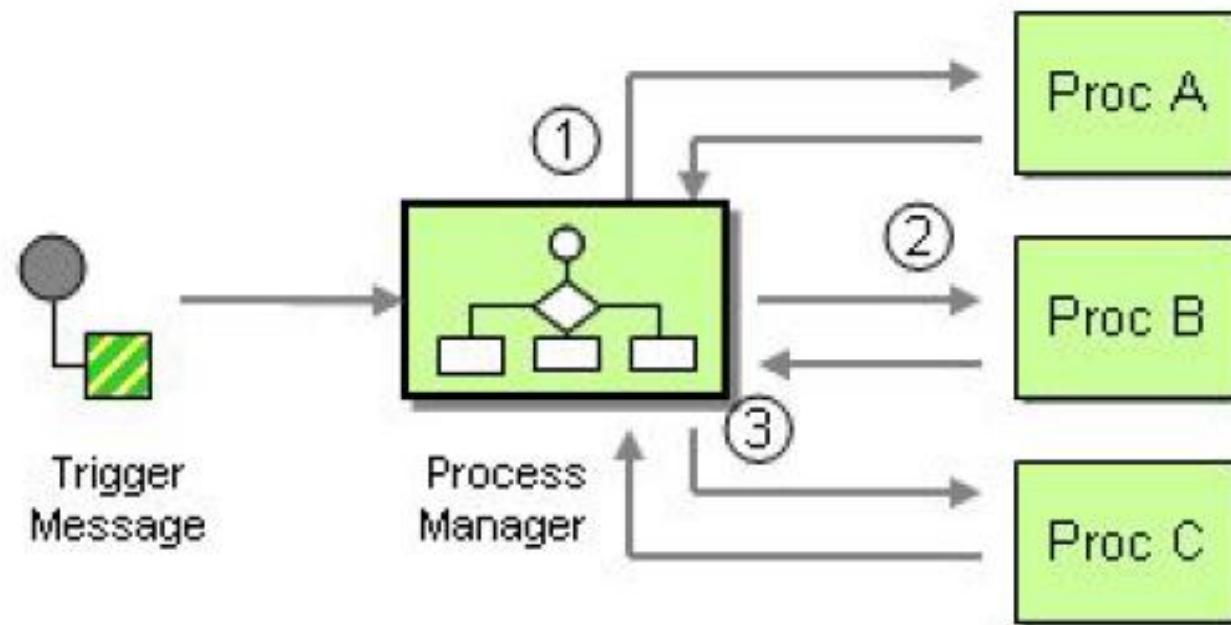
Enterprise Integration Patterns

Aggregator



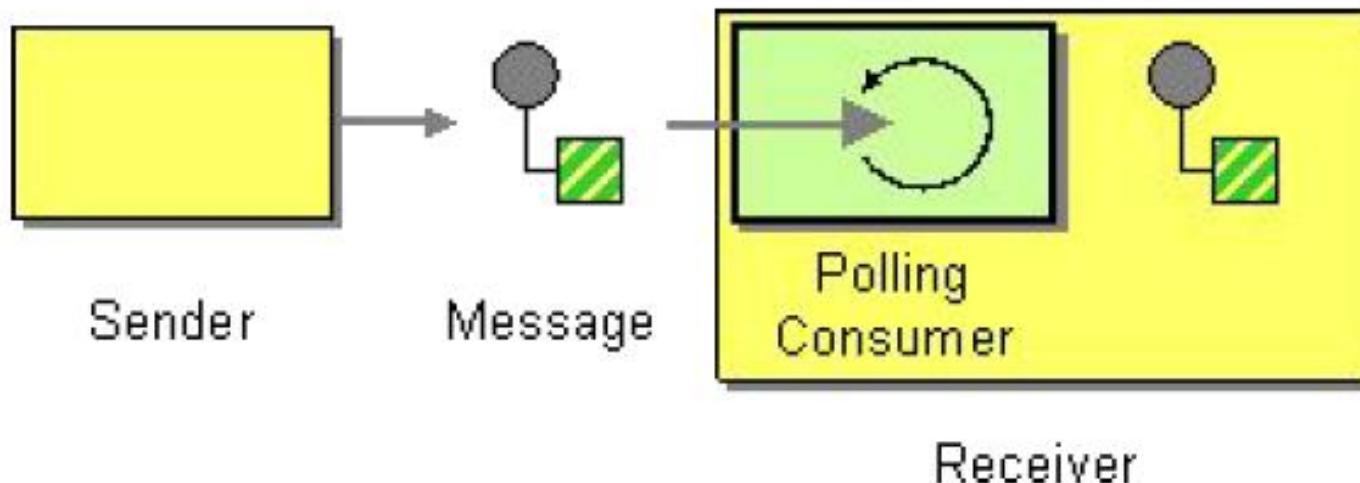
Enterprise Integration Patterns

Process Manager

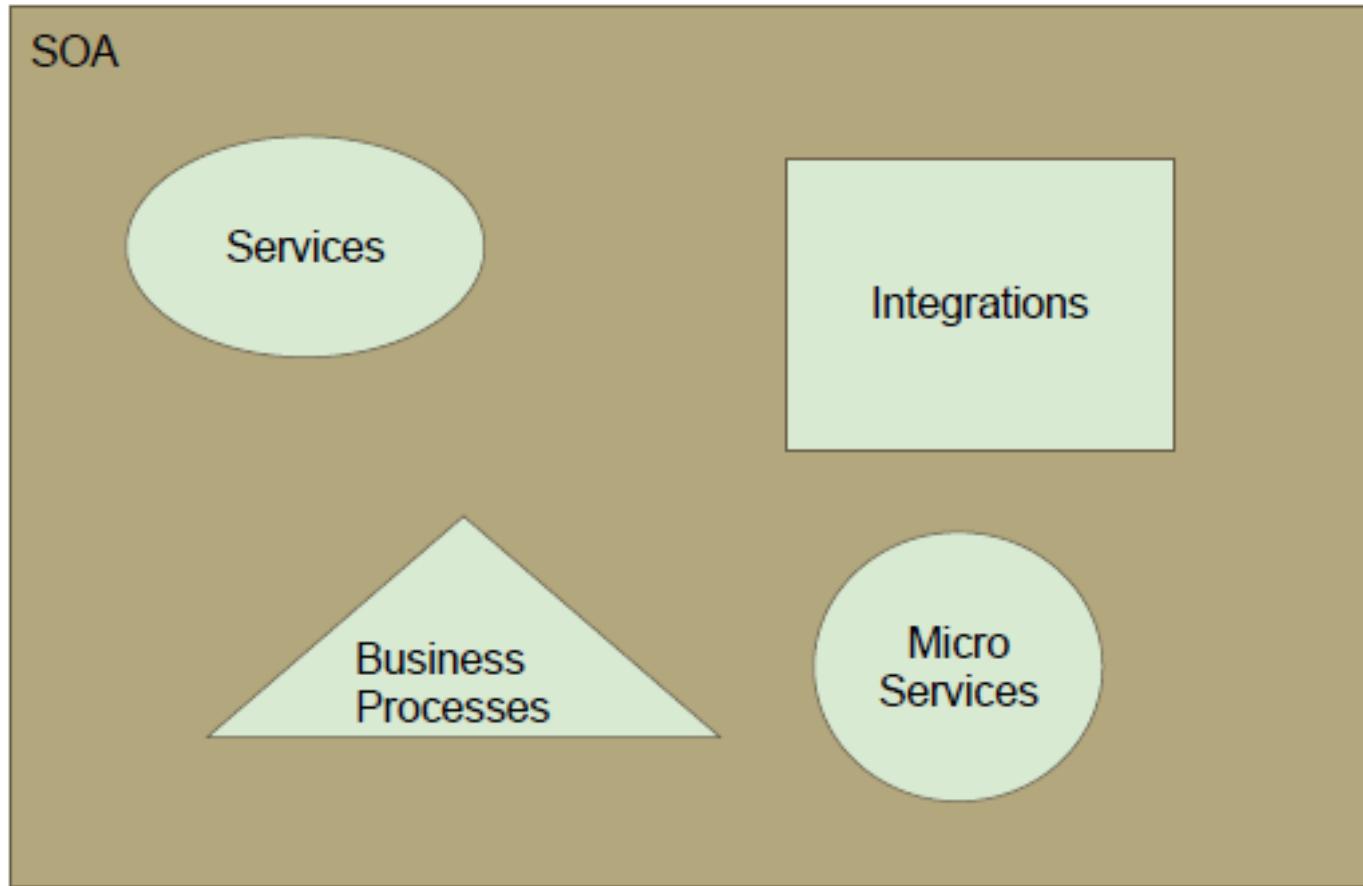


Enterprise Integration Patterns

Polling Consumer



SOA Space



Service Orchestration

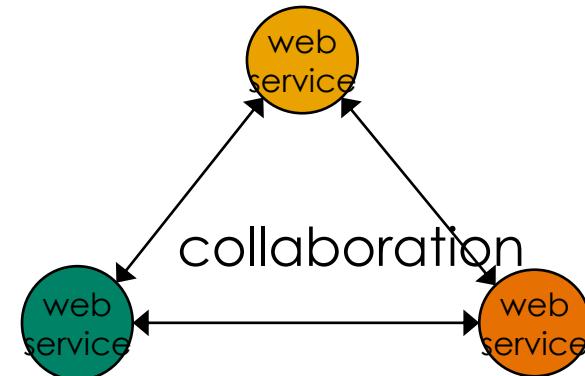
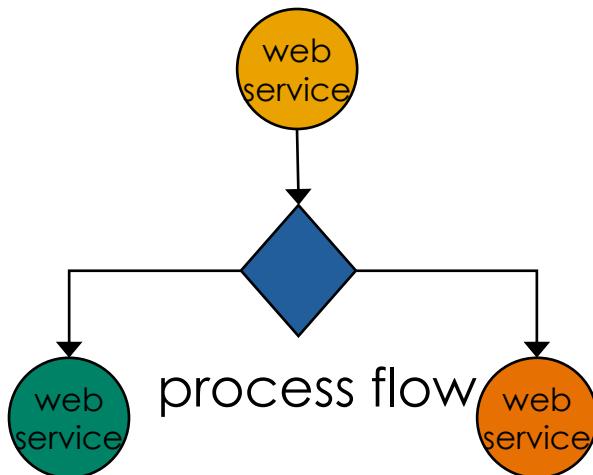


Service Orchestration

- Process Logic module is like a leader in an orchestration
- Services need to be linked and sequenced to form an application
 - This process is known as orchestration.
- Orchestration Models
 - Activity diagram
 - State charts
 - Petri Nets
 - Activity Hierarchy
 - Etc.

Orchestration vs. Choreography

- Orchestration
 - An executable business process describing a flow from the perspective and under control of a single endpoint
- Choreography
 - The observable public exchange of messages, rules of interaction and agreements between two or more business process endpoints



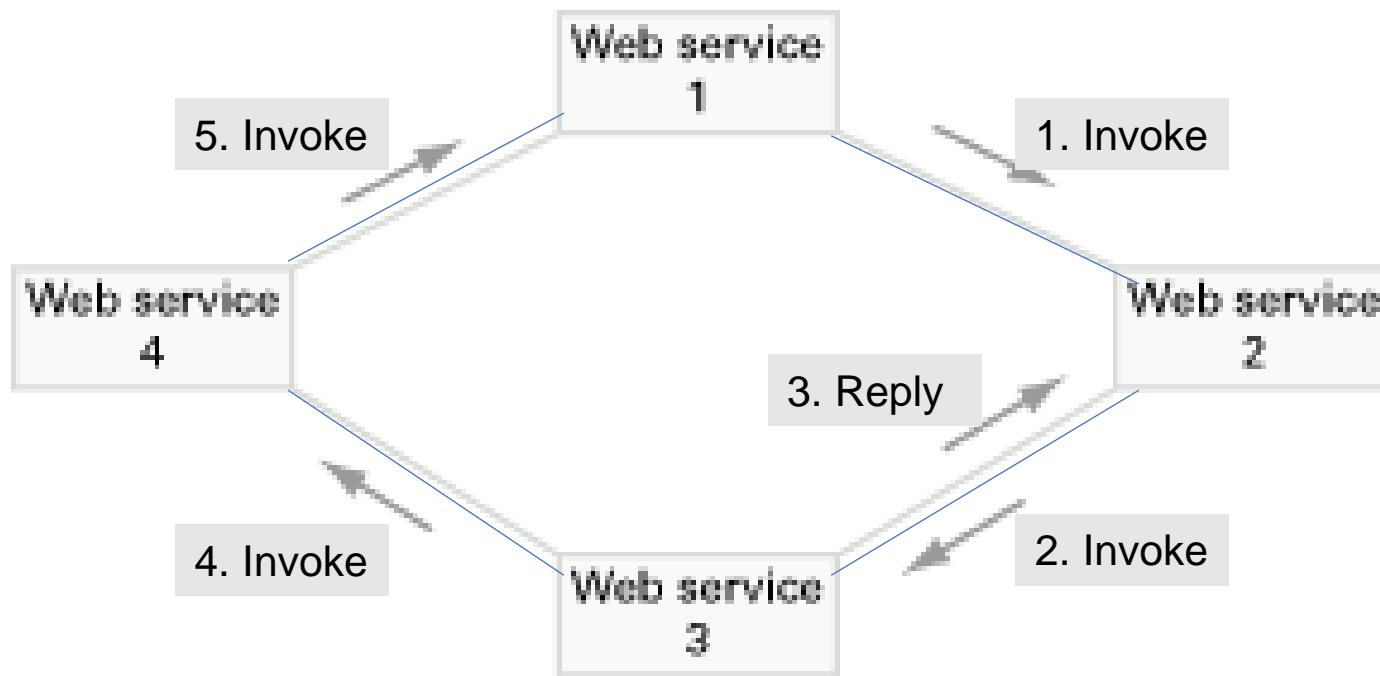
Service Choreography



Choreography

- Does not rely on a central coordinator
- Each Web service involved in the choreography knows exactly when to execute its operations and with whom to interact
- Collaborative effort focusing on the exchange of messages in public business processes
- All participants in the choreography need to be aware of the business process, operations to execute, messages to exchange, and the timing of message exchanges.

Choreography



Orchestration versus Choreography

- From the perspective of composing Web services to execute business processes, orchestration is a more flexible paradigm and has the following advantages over choreography:
 - The coordination of component processes is centrally managed by a known coordinator.
 - Web services can be incorporated without their being aware that they are taking part in a larger business process.
 - Alternative scenarios can be put in place in case faults occur.

Orchestration versus Choreography

- BPEL supports two different ways of describing business processes that support orchestration and choreography:
 - **Executable processes** allow you to specify the exact details of business processes. They follow the orchestration paradigm and can be executed by an orchestration engine.
 - **Abstract business protocols** allow specification of the public message exchange between parties only. They do not include the internal details of process flows and are not executable. They follow the choreography paradigm.

Business Processes & BPEL

- A **business process** is a collection of interrelated tasks, which are designed to deliver a particular result
- A business process can be decomposed into several sub-processes, which have their own attributes, but are aligned with the goal of the overall process
- The analysis of business processes typically includes the mapping of processes and sub-processes down to an activity level
- BPEL: Business Process Execution Language

BPEL

- Basically a tool to create programs using flow diagrams, whose building blocks are individual services
- Really meant for business analysts, not programmers
 - Facilitates orchestration without knowing how to code
 - Programmers would do it in a ‘proper’ programming language
- WSBPEL is a BPEL implementation for Web Services
 - BPEL could apply to other SOA approaches

BPEL

- BPEL is an XML-based language.
- It is an open standard – not proprietary
- BPEL scope includes:
 - Sequencing of process activities, especially Web Service interactions
 - Correlation of messages and process instances
 - Recovery behavior in case of failures and exceptional conditions
 - Bilateral Web Service based relationships between process roles

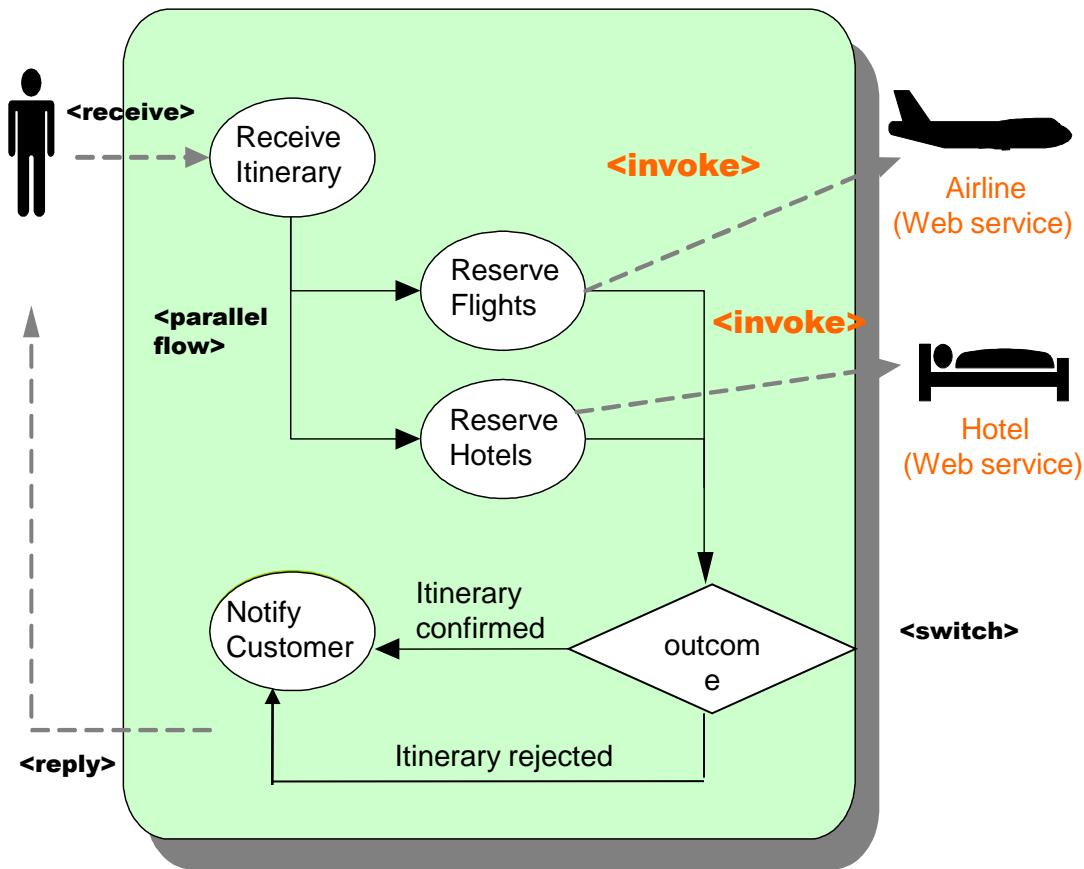
BPEL Activity

- A BPEL process consists of steps. Each step is called an activity.
- BPEL supports primitive and structural activities.
- Primitive activities represent basic constructs and are used for common tasks, such as those listed below:
 - Invoking Web services, using **<invoke>**
 - Waiting for the request, using **<receive>**
 - Manipulating data variables, using **<assign>**
 - Indicating faults and exceptions, using **<throw>**, etc.

BPEL Activity

- We can then combine these activities into more complex algorithms that specify the steps of a business process.
- To combine primitive activities, BPEL supports several structure activities.
- The most important are:
 - Sequence (**<sequence>**) for defining a set of activities that will be invoked in an ordered sequence
 - Flow (**<flow>**) for defining a set of activities that will be invoked in parallel
 - Case-switch construct (**<switch>**) for implementing branches
 - While (**<while>**) for defining loops, etc.

BPEL-Example



- <receive> and <reply> activities receive messages from and give feedback to customers
- <invoke> activities are used to trigger internal and/or external web services
- <parallel flow> activity allows tasks to be executed concurrently.
- <switch> activity allows conditional behaviours in business process.

A BPEL Process

```
001 <process name="purchaseOrderProcess"
002     targetNamespace="..."
003     xmlns="..."
004     xmlns:Ins="...">
...
044     <sequence>
045         <receive partnerLink="purchasing"
046             portType="Ins:purchaseOrderPT"
047             operation="sendPurchaseOrder"
048             variable="PO">
049     </receive>
050     <flow>
051         <links>
052             <link name="ship-to-invoice"/>
053             <link name="ship-to-scheduling"/>
054         </links>
055         <sequence>
056             <assign>
057                 <copy>
058                     <from variable="PO" part="customerInfo"/>
059                     <to variable="shippingRequest"
060                         part="customerInfo"/>
061                 </copy>
062             </assign>
063             <invoke partnerLink="shipping"
064                 portType="Ins:shippingPT"
065                 operation="requestShipping"
066                 inputVariable="shippingRequest"
067                 outputVariable="shippingInfo">
068                 <source linkName="ship-to-invoice"/>
069             </invoke>
070             <receive partnerLink="shipping"
071                 portType="Ins:shippingCallbackPT"
072                 operation="sendSchedule"
073                 variable="shippingSchedule">
074                 <source linkName="ship-to-scheduling"/>
075             </receive>
076         </sequence>
077     <sequence>
078         <invoke partnerLink="invoicing"
079             portType="Ins:computePricePT"
080             operation="initiatePriceCalculation"
081             inputVariable="PO">
082     </invoke>
083     <invoke partnerLink="invoicing"
084         portType="Ins:computePricePT"
085         operation="sendShippingPrice"
086         inputVariable="shippingInfo">
087         <target linkName="ship-to-invoice"/>
088     </invoke>
089     <receive partnerLink="invoicing"
090         portType="Ins:invoiceCallbackPT"
091         operation="sendInvoice"
092         variable="Invoice"/>
093     </sequence>
094     <sequence>
095         <invoke partnerLink="scheduling"
096             portType="Ins:schedulingPT"
097             operation="requestProductionScheduling"
098             inputVariable="PO">
099     </invoke>
100     <invoke partnerLink="scheduling"
101         portType="Ins:schedulingPT"
102         operation="sendShippingSchedule"
103         inputVariable="shippingSchedule">
104         <target linkName="ship-to-scheduling"/>
105     </invoke>
106     </sequence>
107     </flow>
108     <reply partnerLink="purchasing"
109         portType="Ins:purchaseOrderPT"
110         operation="sendPurchaseOrder"
111         variable="Invoice"/>
112     </sequence>
113 </process>
```

Structured Activities

```
001 <process name="purchaseOrderProcess"          077   <sequence>
002     targetNamespace="..."                   078     <invoke partnerLink="invoicing"
003     xmlns="..."                           079         portType="Ins:computePricePT"
004     xmlns:Ins="...">                      080         operation="initiatePriceCalculation"
...                                         081         inputVariable="PO">
044     <sequence>                            082     </invoke>
045         <receive partnerLink="purchasing"    083     <invoke partnerLink="invoicing"
046             portType="Ins:purchaseOrderPT"    084         portType="Ins:computePricePT"
047             operation="sendPurchaseOrder"    085         operation="sendShippingPrice"
048             variable="PO">                  086         inputVariable="shippingInfo">
049     </receive>                            087         <target linkName="ship-to-invoice"/>
050     <flow>                                088     </invoke>
051         <links>                            089     <receive partnerLink="invoicing"
052             <link name="ship-to-invoice"/>    090         portType="Ins:invoiceCallbackPT"
053             <link name="ship-to-scheduling"/> 091         operation="sendInvoice"
054         </links>                            092         variable="Invoice"/>
055     <sequence>                            093     </sequence>
056         <assign>                            094     <sequence>
057             <copy>                            095         <invoke partnerLink="scheduling"
058                 <from variable="PO" part="customerInfo"/> 096             portType="Ins:schedulingPT"
059                 <to variable="shippingRequest"    097             operation="requestProductionScheduling"
060                     part="customerInfo"/>      098             inputVariable="PO">
061             </copy>                            099     </invoke>
062         </assign>                            100     <invoke partnerLink="scheduling"
063         <invoke partnerLink="shipping"        101             portType="Ins:schedulingPT"
064             portType="Ins:shippingPT"          102             operation="sendShippingSchedule"
065             operation="requestShipping"       103             inputVariable="shippingSchedule">
066             inputVariable="shippingRequest"   104         <target linkName="ship-to-scheduling"/>
067             outputVariable="shippingInfo">    105     </invoke>
068         <source linkName="ship-to-invoice"/> 106     </sequence>
069     </invoke>                            107     </flow>
070     <receive partnerLink="shipping"        108     <reply partnerLink="purchasing"
071             portType="Ins:shippingCallbackPT" 109         portType="Ins:purchaseOrderPT"
072             operation="sendSchedule"        110         operation="sendPurchaseOrder"
073             variable="shippingSchedule">    111         variable="Invoice"/>
074         <source linkName="ship-to-scheduling"/> 112     </sequence>
075     </receive>                            113 </process>
076 </sequence>
```

Primitive Activities

```
001 <process name="purchaseOrderProcess"
002     targetNamespace="..."
003     xmlns="..."
004     xmlns:Ins="...">
...
044     <sequence>
045         <receive partnerLink="purchasing"
046             portType="Ins:purchaseOrderPT"
047             operation="sendPurchaseOrder"
048             variable="PO">
049     </receive>
050     <flow>
051         <links>
052             <link name="ship-to-invoice"/>
053             <link name="ship-to-scheduling"/>
054         </links>
055         <sequence>
056             <assign>
057                 <copy>
058                     <from variable="PO" part="customerInfo"/>
059                     <to variable="shippingRequest"
060                         part="customerInfo"/>
061                 </copy>
062             </assign>
063             <invoke partnerLink="shipping"
064                 portType="Ins:shippingPT"
065                 operation="requestShipping"
066                 inputVariable="shippingRequest"
067                 outputVariable="shippingInfo">
068                 <source linkName="ship-to-invoice"/>
069             </invoke>
070             <receive partnerLink="shipping"
071                 portType="Ins:shippingCallbackPT"
072                 operation="sendSchedule"
073                 variable="shippingSchedule">
074                 <source linkName="ship-to-scheduling"/>
075             </receive>
076         </sequence>
077     <sequence>
078         <invoke partnerLink="invoicing"
079             portType="Ins:computePricePT"
080             operation="initiatePriceCalculation"
081             inputVariable="PO">
082     </invoke>
083         <invoke partnerLink="invoicing"
084             portType="Ins:computePricePT"
085             operation="sendShippingPrice"
086             inputVariable="shippingInfo">
087             <target linkName="ship-to-invoice"/>
088         </invoke>
089         <receive partnerLink="invoicing"
090             portType="Ins:invoiceCallbackPT"
091             operation="sendInvoice"
092             variable="Invoice"/>
093     </sequence>
094     <sequence>
095         <invoke partnerLink="scheduling"
096             portType="Ins:schedulingPT"
097             operation="requestProductionScheduling"
098             inputVariable="PO">
099     </invoke>
100     <invoke partnerLink="scheduling"
101             portType="Ins:schedulingPT"
102             operation="sendShippingSchedule"
103             inputVariable="shippingSchedule">
104             <target linkName="ship-to-scheduling"/>
105     </invoke>
106     </sequence>
107     </flow>
108     <reply partnerLink="purchasing"
109             portType="Ins:purchaseOrderPT"
110             operation="sendPurchaseOrder"
111             variable="Invoice"/>
112     </sequence>
113 </process>
```

Data Flow

```
001 <process name="purchaseOrderProcess"
002     targetNamespace="..."
003     xmlns="..."
004     xmlns:Ins="...">
...
044     <sequence>
045         <receive partnerLink="purchasing"
046             portType="Ins:purchaseOrderPT"
047             operation="sendPurchaseOrder"
048             variable="PO">
049     </receive>
050     <flow>
051         <links>
052             <link name="ship-to-invoice"/>
053             <link name="ship-to-scheduling"/>
054         </links>
055         <sequence>
056             <assign>
057                 <copy>
058                     <from variable="PO" part="customerInfo"/>
059                     <to variable="shippingRequest"
060                         part="customerInfo"/>
061                 </copy>
062             </assign>
063             <invoke partnerLink="shipping"
064                 portType="Ins:shippingPT"
065                 operation="requestShipping"
066                 inputVariable="shippingRequest"
067                 outputVariable="shippingInfo">
068                 <source linkName="ship-to-invoice"/>
069             </invoke>
070             <receive partnerLink="shipping"
071                 portType="Ins:shippingCallbackPT"
072                 operation="sendSchedule"
073                 variable="shippingSchedule">
074                 <source linkName="ship-to-scheduling"/>
075             </receive>
076         </sequence>
077     <sequence>
078         <invoke partnerLink="invoicing"
079             portType="Ins:computePricePT"
080             operation="initiatePriceCalculation"
081             inputVariable="PO">
082     </invoke>
083     <invoke partnerLink="invoicing"
084         portType="Ins:computePricePT"
085         operation="sendShippingPrice"
086         inputVariable="shippingInfo">
087         <target linkName="ship-to-invoice"/>
088     </invoke>
089     <receive partnerLink="invoicing"
090         portType="Ins:invoiceCallbackPT"
091         operation="sendInvoice"
092         variable="Invoice"/>
093     </sequence>
094     <sequence>
095         <invoke partnerLink="scheduling"
096             portType="Ins:schedulingPT"
097             operation="requestProductionScheduling"
098             inputVariable="PO">
099     </invoke>
100     <invoke partnerLink="scheduling"
101         portType="Ins:schedulingPT"
102         operation="sendShippingSchedule"
103         inputVariable="shippingSchedule">
104         <target linkName="ship-to-scheduling"/>
105     </invoke>
106     </sequence>
107     </flow>
108     <reply partnerLink="purchasing"
109         portType="Ins:purchaseOrderPT"
110         operation="sendPurchaseOrder"
111         variable="Invoice"/>
112     </sequence>
113 </process>
```

Partner Links

```
001 <process name="purchaseOrderProcess"
002     targetNamespace="..."
003     xmlns="..."
004     xmlns:Ins="...">
...
044     <sequence>
045         <receive partnerLink="purchasing"
046             portType="Ins:purchaseOrderPT"
047             operation="sendPurchaseOrder"
048             variable="PO">
049     </receive>
050     <flow>
051         <links>
052             <link name="ship-to-invoice"/>
053             <link name="ship-to-scheduling"/>
054         </links>
055         <sequence>
056             <assign>
057                 <copy>
058                     <from variable="PO" part="customerInfo"/>
059                     <to variable="shippingRequest"
060                         part="customerInfo"/>
061                 </copy>
062             </assign>
063             <invoke partnerLink="shipping"
064                 portType="Ins:shippingPT"
065                 operation="requestShipping"
066                 inputVariable="shippingRequest"
067                 outputVariable="shippingInfo">
068                 <source linkName="ship-to-invoice"/>
069             </invoke>
070             <receive partnerLink="shipping"
071                 portType="Ins:shippingCallbackPT"
072                 operation="sendSchedule"
073                 variable="shippingSchedule">
074                 <source linkName="ship-to-scheduling"/>
075             </receive>
076         </sequence>
077     <sequence>
078         <invoke partnerLink="invoicing"
079             portType="Ins:computePricePT"
080             operation="initiatePriceCalculation"
081             inputVariable="PO">
082     </invoke>
083     <invoke partnerLink="invoicing"
084         portType="Ins:computePricePT"
085         operation="sendShippingPrice"
086         inputVariable="shippingInfo">
087         <target linkName="ship-to-invoice"/>
088     </invoke>
089     <receive partnerLink="invoicing"
090         portType="Ins:invoiceCallbackPT"
091         operation="sendInvoice"
092         variable="Invoice"/>
093     </sequence>
094     <sequence>
095         <invoke partnerLink="scheduling"
096             portType="Ins:schedulingPT"
097             operation="requestProductionScheduling"
098             inputVariable="PO">
099     </invoke>
100     <invoke partnerLink="scheduling"
101         portType="Ins:schedulingPT"
102         operation="sendShippingSchedule"
103         inputVariable="shippingSchedule">
104         <target linkName="ship-to-scheduling"/>
105     </invoke>
106     </sequence>
107     </flow>
108     <reply partnerLink="purchasing"
109         portType="Ins:purchaseOrderPT"
110         operation="sendPurchaseOrder"
111         variable="Invoice"/>
112     </sequence>
113 </process>
```

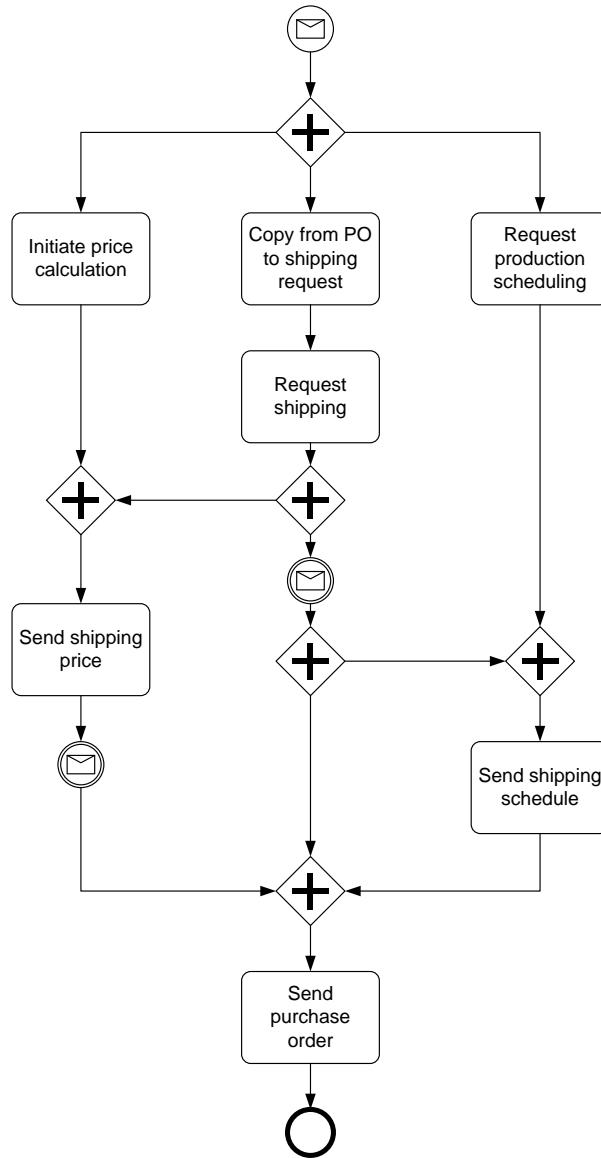
BPMN (Business Process Markup and Notation)

The primary goal of **BPMN** is to provide a notation that is readily understandable by all business users

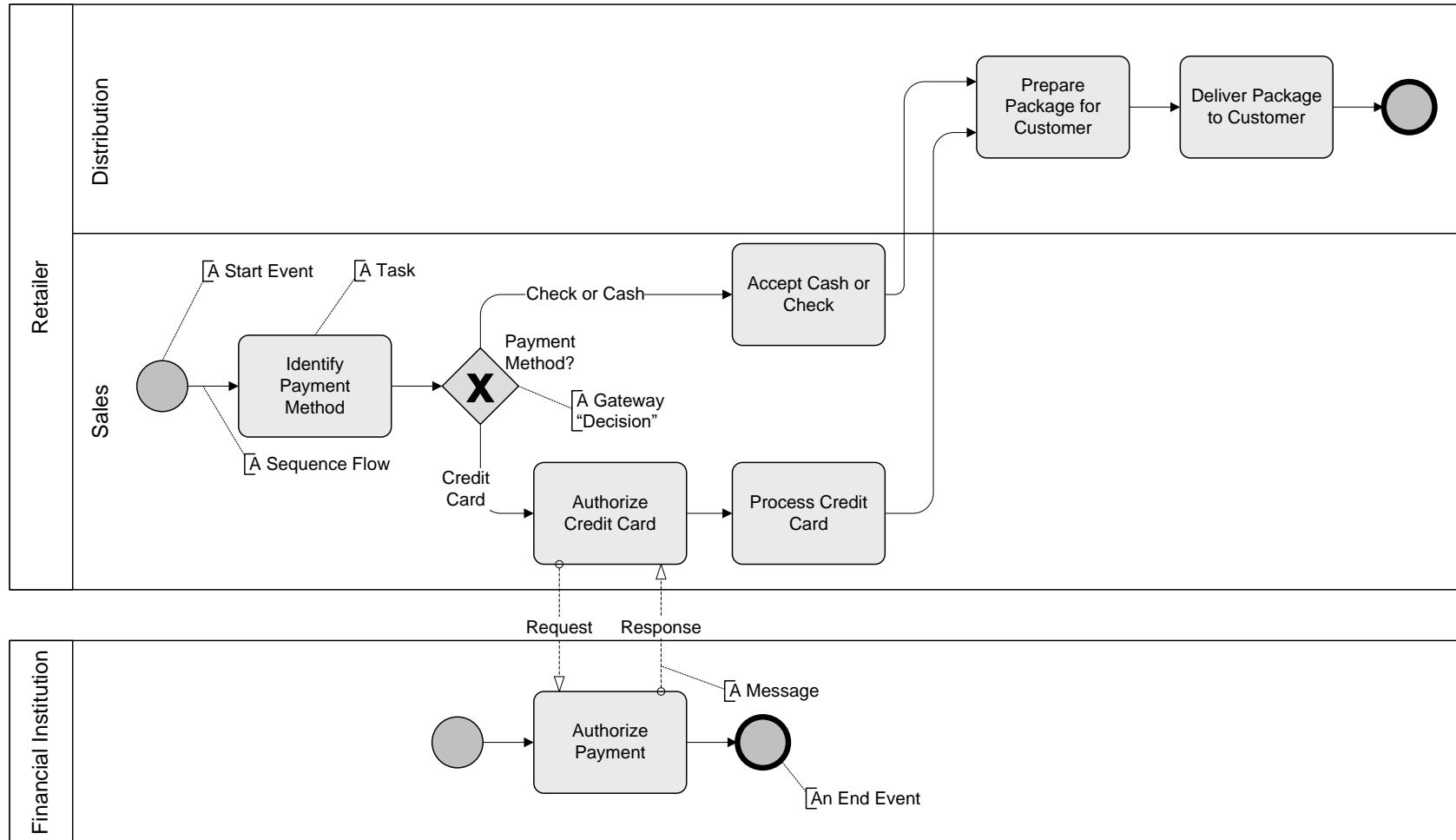
BPMN creates a **standardized bridge** for the gap between the business process design and process implementation.

Another goal, but no less important, is to ensure that XML languages designed for the execution of business processes, such as **BPEL4WS** (Business Process Execution Language for Web Services), can be visualized with a business-oriented notation.

The BPEL Process in BPMN



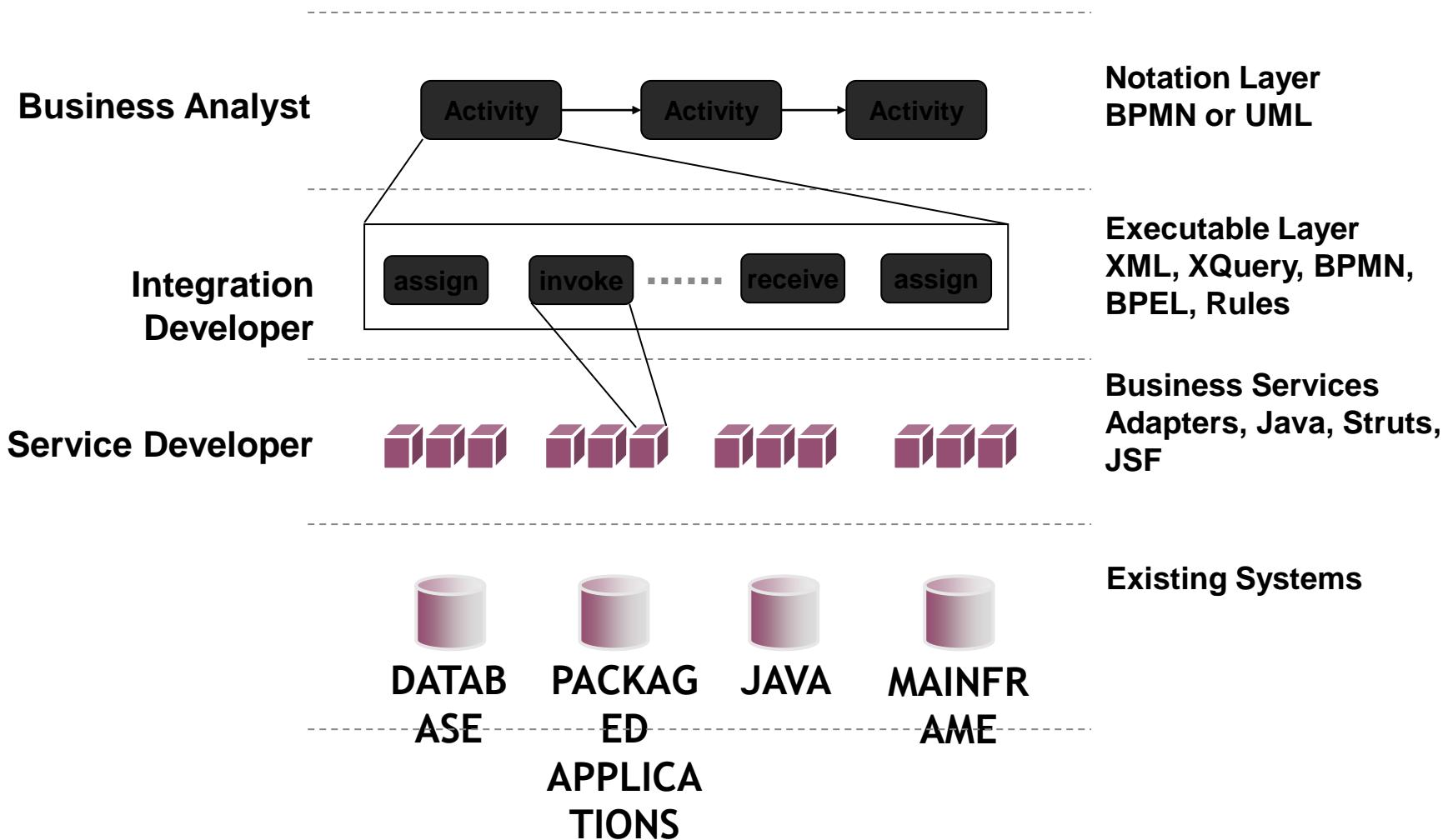
BPMN explained



BPMN Vs BPEL

- BPMN is richer than BPEL
- Transformation from BPEL to BPMN less a problem
- From BPMN to BPEL
 - Loss of information
 - Loss of design considerations
- Potential Solutions
 - Restriction to a subset of BPMN
 - Extension of BPEL

The Top Down Perspective



Securing Web Services

Security Fundamentals

- Authentication
- Authorization
- Integrity
- Confidentiality
- Availability
- Non-Repudiation

Security a Web Service

- WS-Security (SOAP services only)
- SSL with HTTP BasicAuth/HTTP Digest Auth
- SSL with Username Token (OAuth)
- SSL with IDToken (OpenID)

WS Security

- WS Security 1.0 on 2004 and 1.1 on 2006
- Based on
 - PKI
 - X509 Certificates
 - XML Security - XML Encryption and Signature
- Related Standards
 - WS Secure Conversation
 - WS Trust

WS Security

- Username Token
- Message level encryption
- Message level signature
- Message level encryption and signature

WS Security Sepcification Family

- A comprehensive specification
- Provides message level security
- Industry is no longer using it

TLS for HTTP → HTTPS

- TLS - Transport Layer Security
 - Version 1, 1.1, 1.2 and 1.3
- Predecessor of TLS is SSL
- Adds an additional encryption layer over HTTP
- Transport layer provides
 - Transport level confidentiality
 - Transport level integrity

HTTP BasicAuth

- HTTP Header with Username & Password
 - Authorization : Basic <Base64 encoded Username:Password>
- Base64 is not encryption. What is it?
 - Difference between
 - Encoding
 - Encryption
 - Hashing

Hands-on

Create a BasicAuth Header

Demo

HTTP Basic Auth Sample

TLS with BasicAuth

- Provides Authenticity
- Possible to do Authorization
- Transport layer confidentiality and integrity

Http Digest Auth

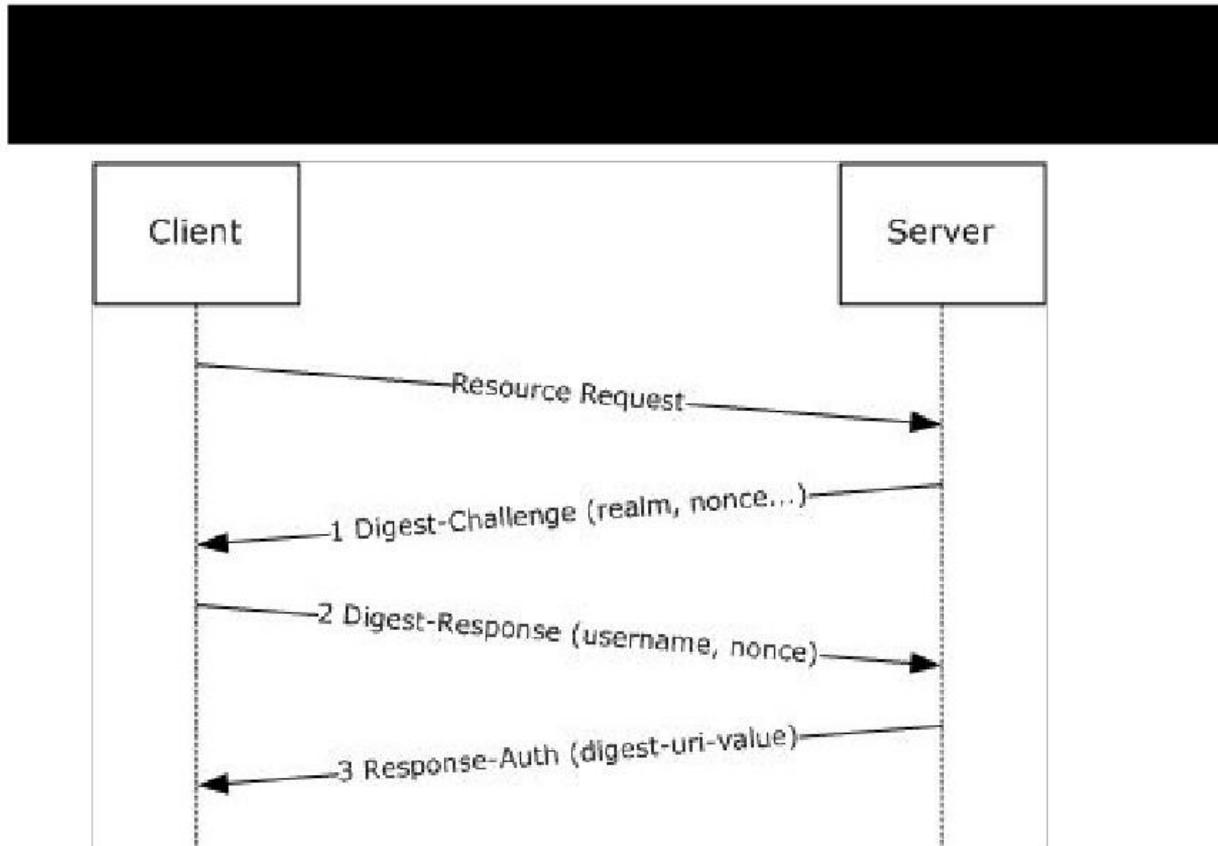
More on Hashing

- Irreversible. Result is called digest.
 - $d = \text{hash}(m_1)$
- Deterministic
- Small change → Drastic result in the digest
- Second image resistance. Given $d=\text{hash}(m_1)$, it is hard to find m_2 such that $d=\text{hash}(m_2)$
- Collision resistance
 - $\text{hash}(m_1) = \text{hash}(m_2)$

Cryptographic Nonce

- A number that is used only once
- Random Number
- Adding a Nonce to a message before hashing makes the Digest attacks hard

HTTP DigestAuth



Http DigestAuth

- *STEP 1* : a client sends a request to a server
- *STEP 2* : the server responds with a special code (called a **nonce** i.e. **number used only once**), another string representing the ‘realm’ and asks the client to authenticate
- *STEP 3* : the client responds with the hashed value of this nonce and the username, password and realm
- *STEP 4* : the server responds with the requested information if the client hash matches their own hash of the nonce, username, password and realm, or an error if not

TLS with HTTP DigestAuth

- Secure than BasicAuth
- Provides Authenticity
- Possible to do Authorization
- Transport layer confidentiality and integrity

OAuth for Security REST Services

- OAuth1.0 & 2.0
 - Open standard for Authorization
- OAuth 2.0 Framework and Bearer Token Usage RFC published on 2012
- OAuth2.0 - Most commonly used standard for security REST Services
 - Facebook Graph API, Google APIs
- Different implementations: e.g. WSO2 Identity Server,

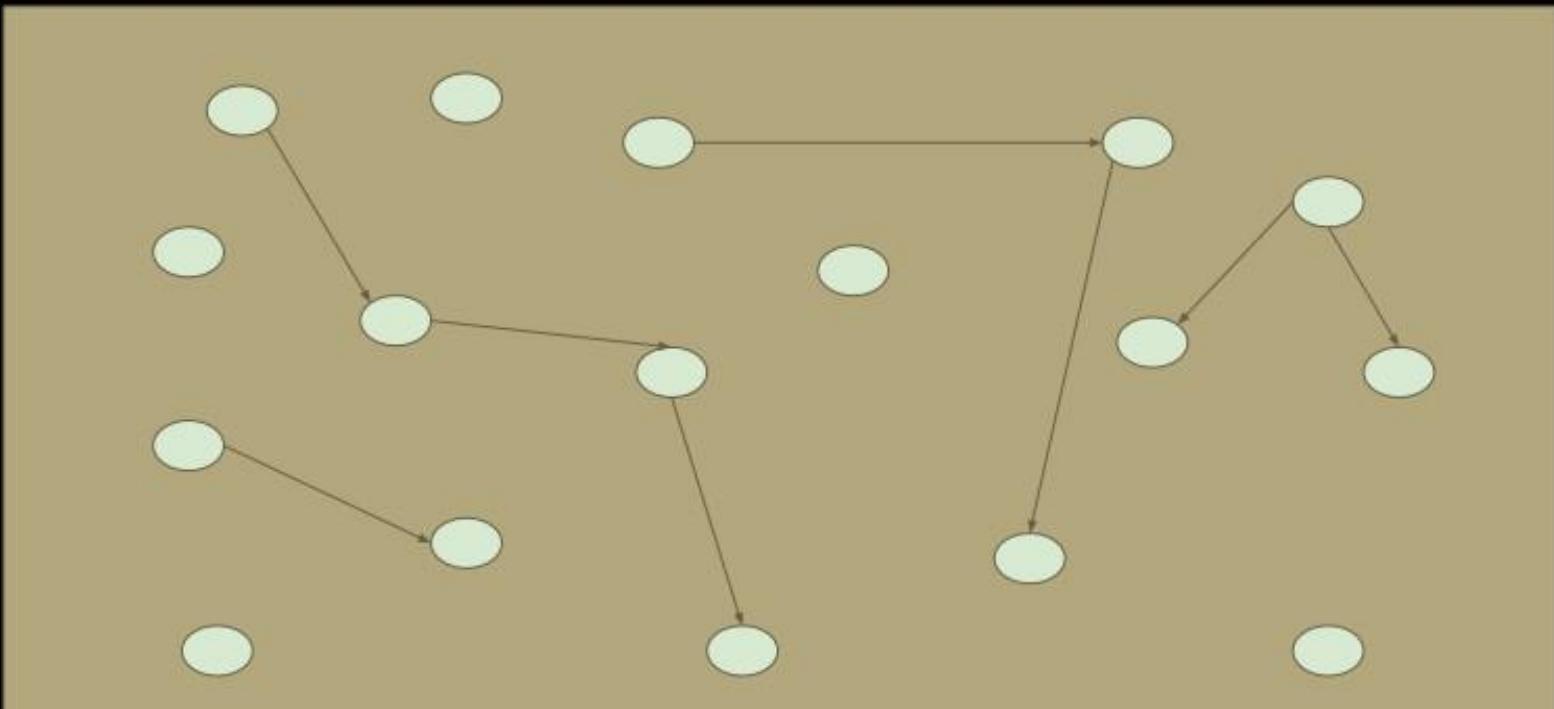
SOA Governance



SOA Governance

- The “course of action” taken to ensure that an effective decision-making process is in place
- A set of processes, responsibilities and tools, which reinforces good behavior and help avoid bad behaviors
- Ensure defined processes and responsibilities are followed through the implementation of proper measurement technique
- It’s all about control!

SOA



Why SOA Governance

- Essential part in making SOA successful
 - Need to reuse services
 - Need to make developing application, automated processes easy
 - Need to manage services

Why SOA Governance

How to promote reuse services

- List services and their descriptions
- Show the technical/business owners of a service
- Analyze inter relationship of service
- Validate the service - Check technical and business rules

Why SOA Governance

Manage Services

- Version the services
- Provide security policies at runtime
- Provide secure access to services
- Track the QoS
- Maintain lifecycle management

When is Governance Required?

- Architecture Governance
- Design-time Governance
- Run-time Governance
- Organizational Governance

How to implement SOA Governance

Service Registry is the central piece of SOA governance

- Service Catalog
- Service Description
- Service Consumption
- Service inter-dependencies
- Service Discovery
- Service Lifecycle
- Service Policies

Service Description

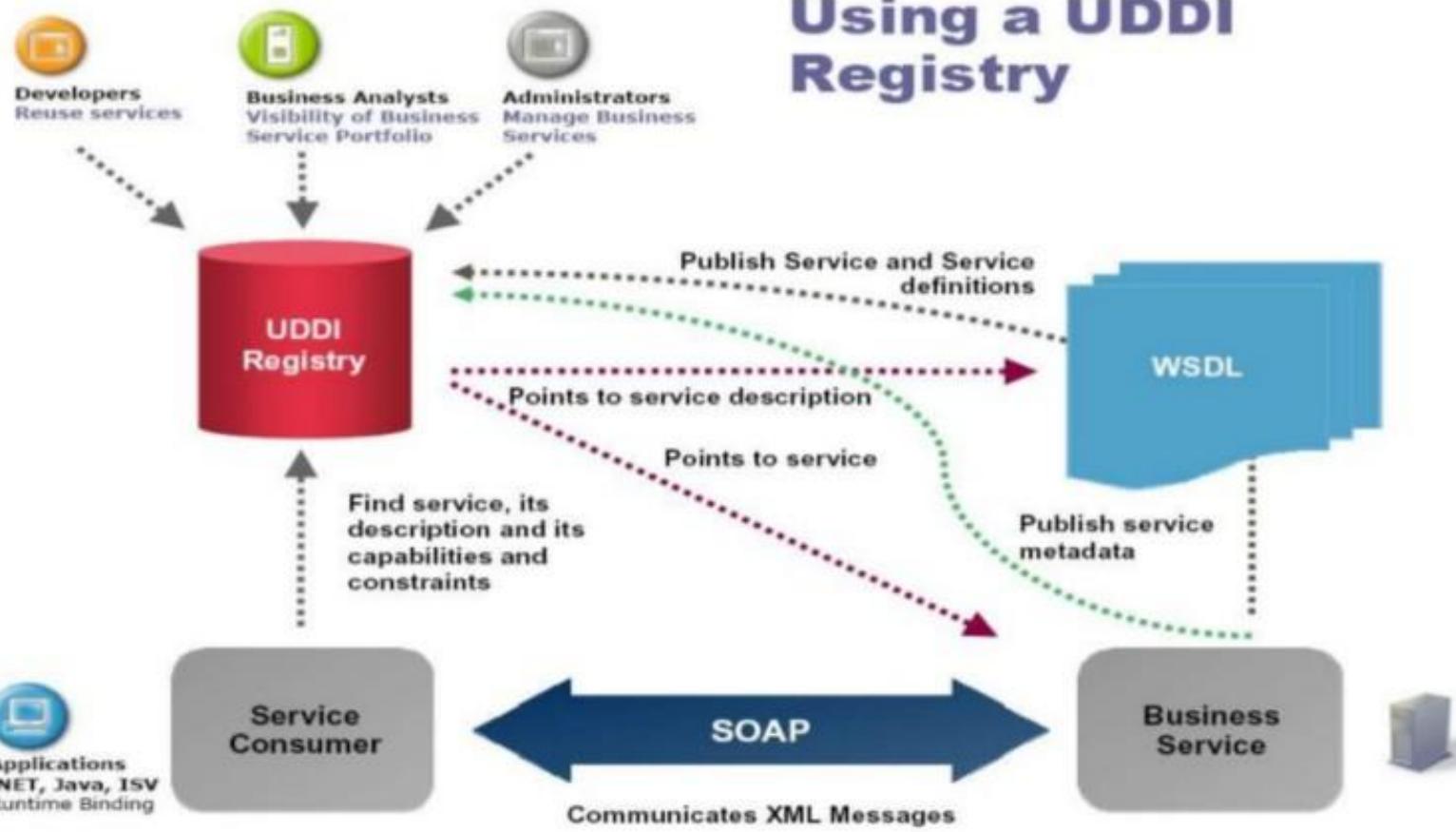
Description	Amazon E-Commerce service exposing catalog and commerce function
Version	1.0.00
Owner	Provider Manager
Status	Open
Creation Date	Jan 08 2007 07:50:17 PM GMT
Expiration Date	Jan 08 2010 07:50:17 PM GMT
Business Name	Amazon E-Commerce
Business Description	Amazon E-Commerce service (ECS) exposes Amazon's product data and e-commerce functionality.
Consumer Classes Supported	Internal/External
Service Usage Status	Active
Lifecycle Status	Production
Production Release Date	Jan 10 2007 02:15:00 PM GMT
WSDL URL	Amazon E Commerce.wsdl

UDDI

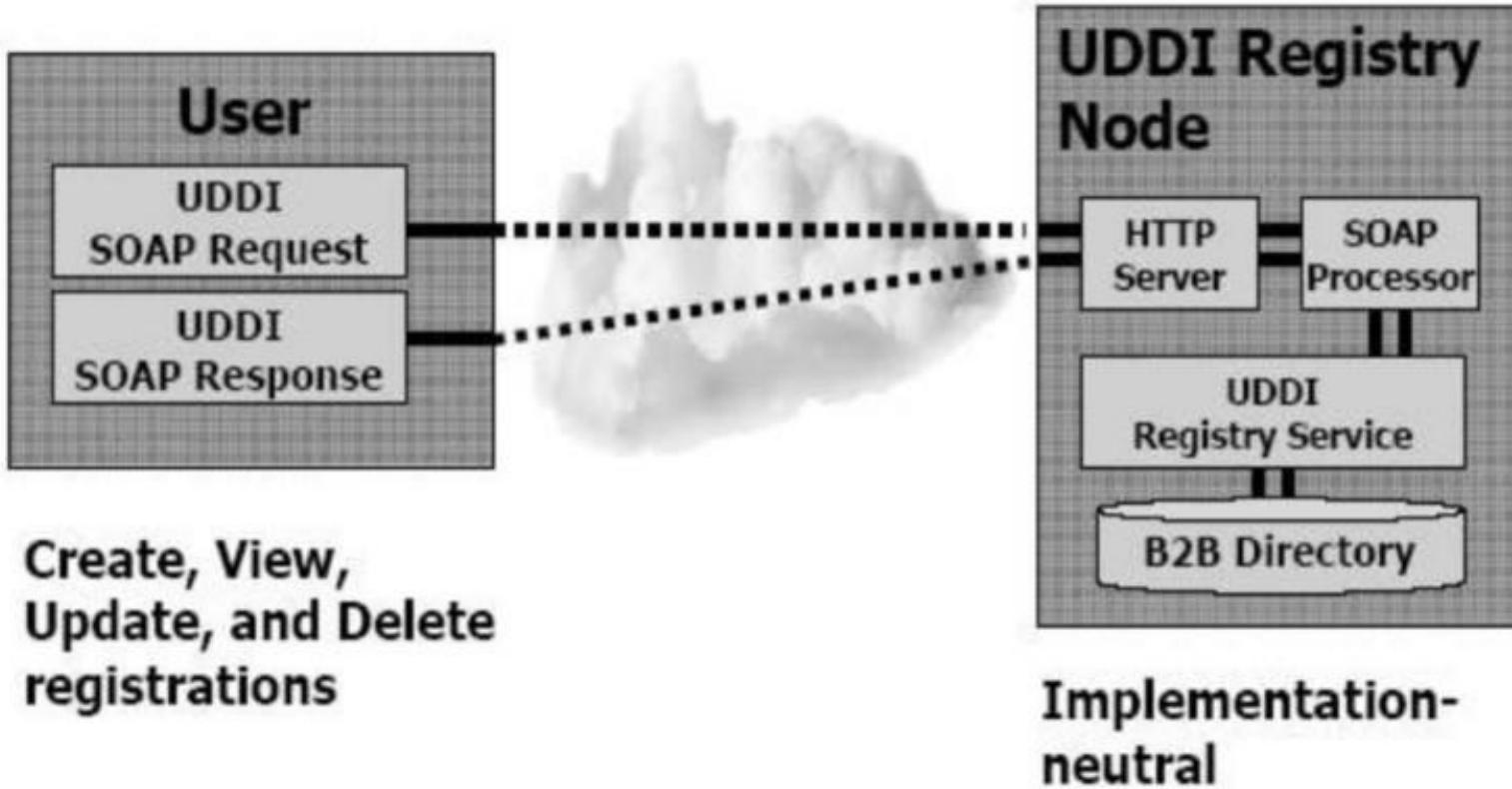
- A registry standard - Universal Description, Discovery and Integration
- SOAP 1.1 based standard since 2000
 - Describing services
 - Publishing services
 - Discovering services
- Four components
 - Registry
 - Data, meta-data
 - UDDI specification
 - API for publishing, managing and discovering services

UDDI

Using a UDDI Registry



UDDI



UDDI - Low Adoption Rate

- Tightly coupled to SOAP/WSDL
- No homogeneous standard in different business
- Look up services at runtime by clients
 - not a common usecase
- Original spec didn't have human friendly formats
- Hard to use (long document required)
(Governance Interoperability Framework)

Current Registry Implementations

- Infravio
- TIBCO Active Matrix
- BEA
- Oracle
- Sun
- Systinet
- WSO2

Modern Governance Products

- Human friendly service catalog
- Has REST interface - No standard
- Can publish services in different formats
- Integrates with developer tooling
- Innovation enabler, not a restrictor
- Notifications to watchers, workflows
- Lifecycle management

Summary

- Orchestration/Choreography to manage the business process
- Service integration to connect services
- Enterprise Service Bus
- Securing web services
 - WS-Security
 - HTTP + BasicAuth/HTTP+ DigestAuth
 - OAuth/OpenID
- SOA Governance

Introduction to Cloud Computing



Lakmal Warusawithana

Vise President, Apache Stratos

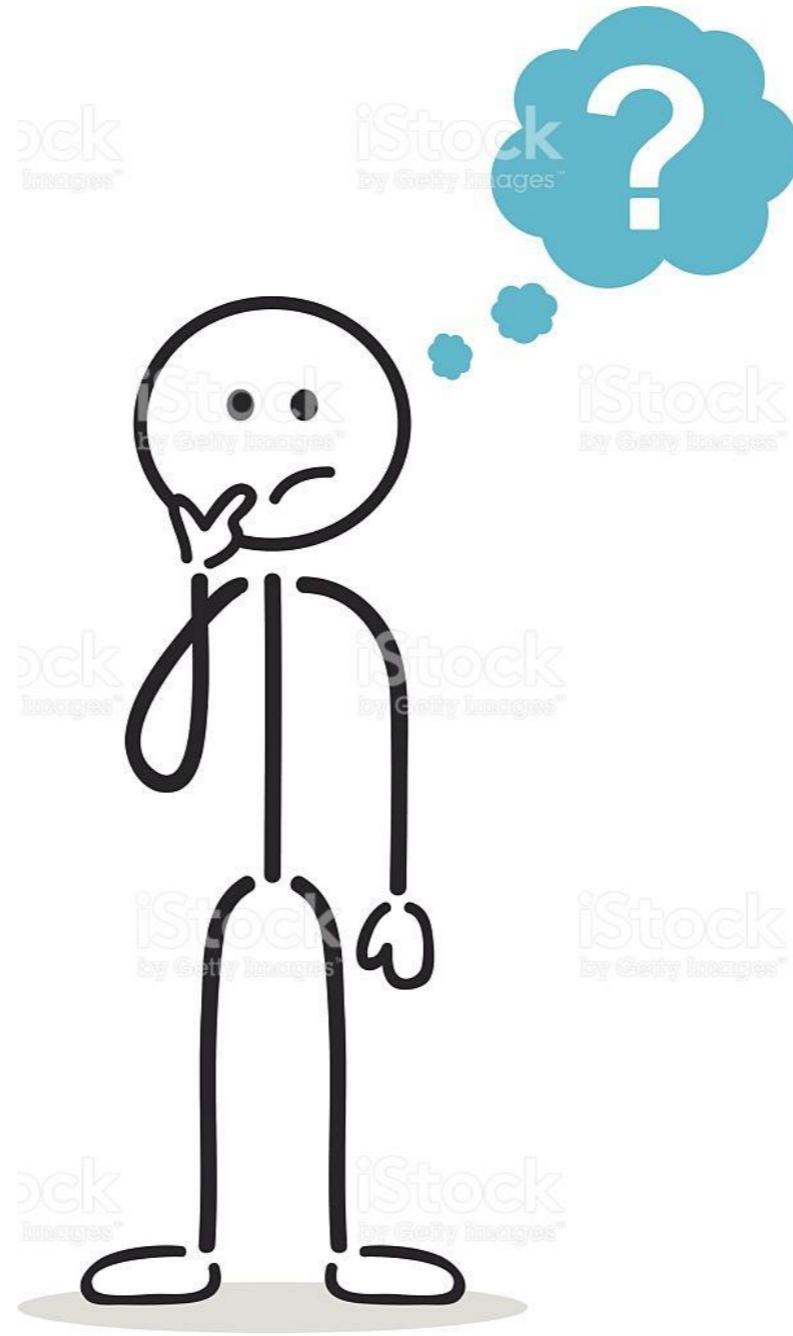
Director - Cloud Architecture, WSO2 Inc

lakmal@apache.org / lakmal@wso2.com

Twitter : lakwarus

2016 - University of Sabaragamuwa

What is Cloud Computing?



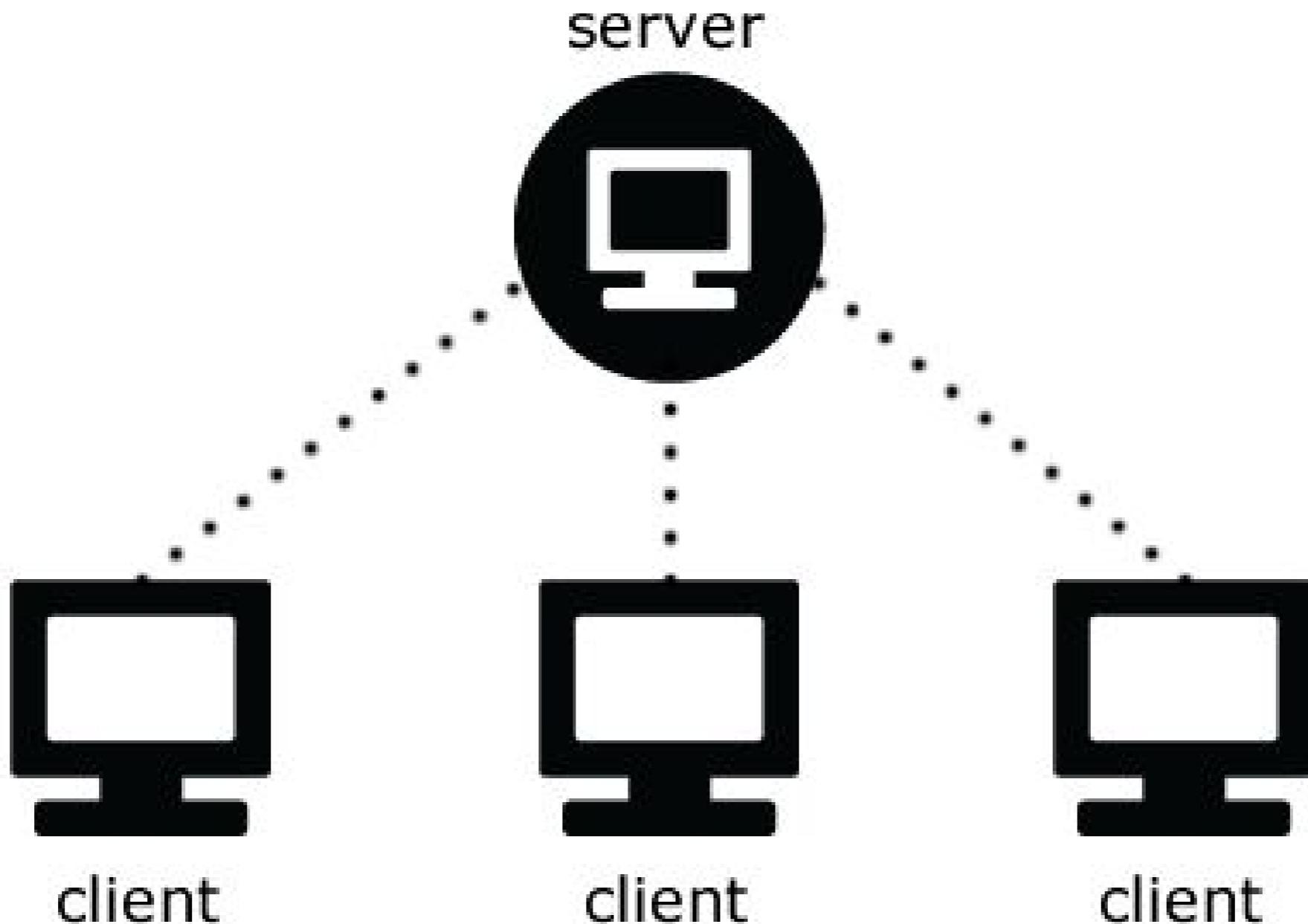
The Evolution of Software – Standalone



The Evolution of Software – Standalone

- Software installed on a local machine
 - e.g. most PC software that need installation
- Local access to software
- Problems - HR System on a PC
 - limited availability
 - limited reliability
 - limited capacity
 - high maintenance cost
 - e.g. software updates
 - high scaling cost

Software Evolution - Client-Server



Software Evolution - Client-Server

- Software installed on a (remote) machine (server)
- Client(s) access(es) server via network
 - communicate using a protocol
- Problems solved:
 - increase availability
 - if you have the proper client, you can use (hopefully)
- Problems remaining:
 - availability (still limited by server capacity)
 - reliability (depends on server)
 - need redundant server(s) & storage as back-up

Software Evolution - Web Applications



Software Evolution - Web Applications

- Still client-server applications
- Use web technologies (e.g. HTTP) to:
 - access application
 - connect components
- Problems solved:
 - thin client
 - limited processing on the client side
 - most processing are done on the server
 - use the “standardised” web interface and protocol
 - further increase availability & accessibility
- Problems remaining:
 - inherits client-server problems

Software Evolution - SuperComputing



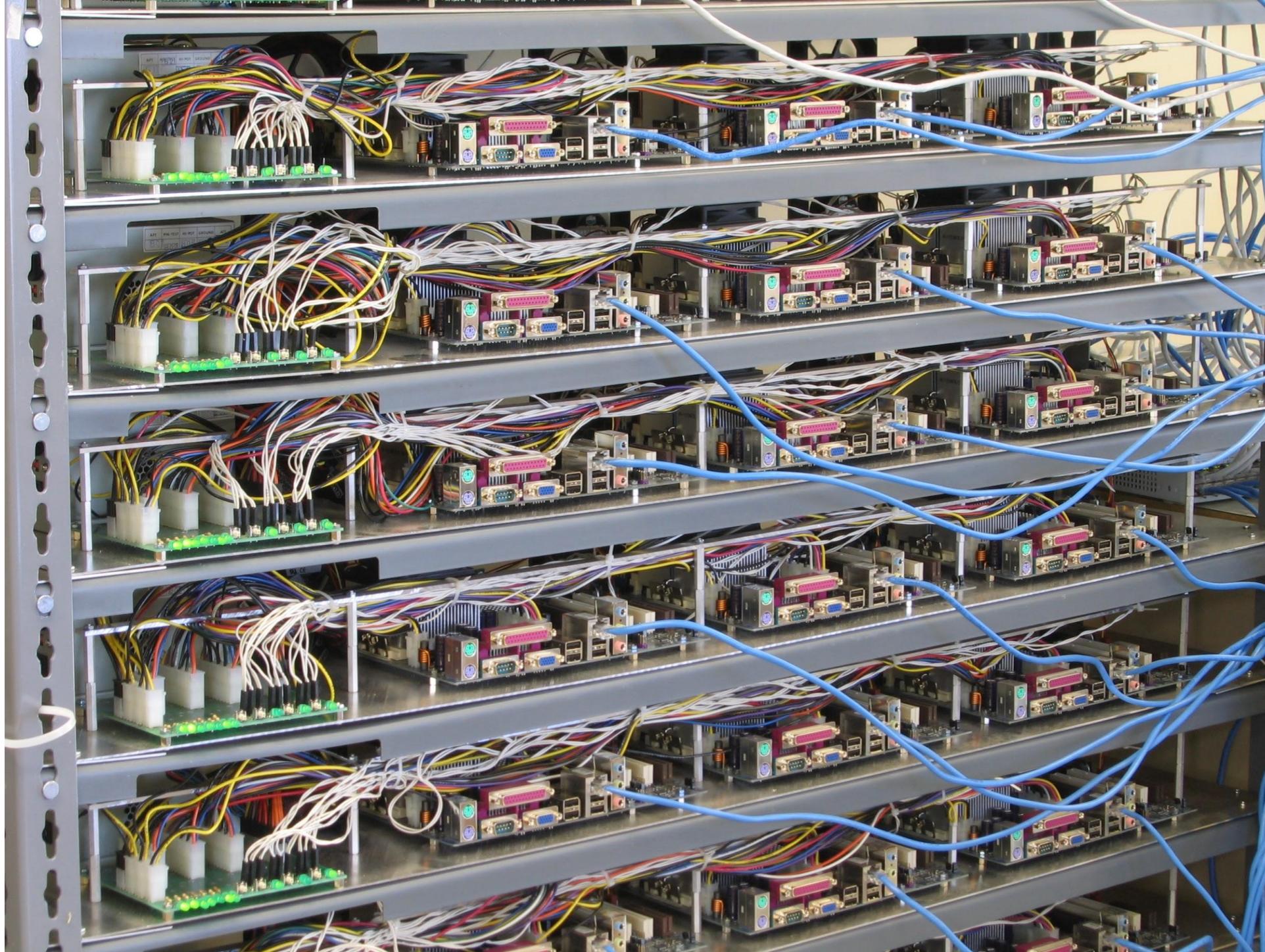
Software Evolution - SuperComputing

<http://www.top500.org/>

The Top 500 list

Rank	Site	System	Cores	Rmax [TFlop/s]	Rpeak [TFlop/s]	Power (kW)
1	National Supercomputing Center in Wuxi China	Sunway TaihuLight - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway NRCPC	10,649,600	93,014.6	125,435.9	15,371
2	National Super Computer Center in Guangzhou China	Tianhe-2 (MilkyWay-2) - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P NUDT	3,120,000	33,862.7	54,902.4	17,808
3	DOE/SC/Oak Ridge National Laboratory United States	Titan - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc.	560,640	17,590.0	27,112.5	8,209
4	DOE/NNSA/LLNL United States	Sequoia - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM	1,572,864	17,173.2	20,132.7	7,890
5	RIKEN Advanced Institute for Computational Science (AICS) Japan	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect Fujitsu	705,024	10,510.0	11,280.4	12,660
6	DOE/SC/Argonne National Laboratory United States	Mira - BlueGene/Q, Power BQC 16C 1.60GHz, Custom IBM	786,432	8,586.6	10,066.3	3,945

Cluster Computing

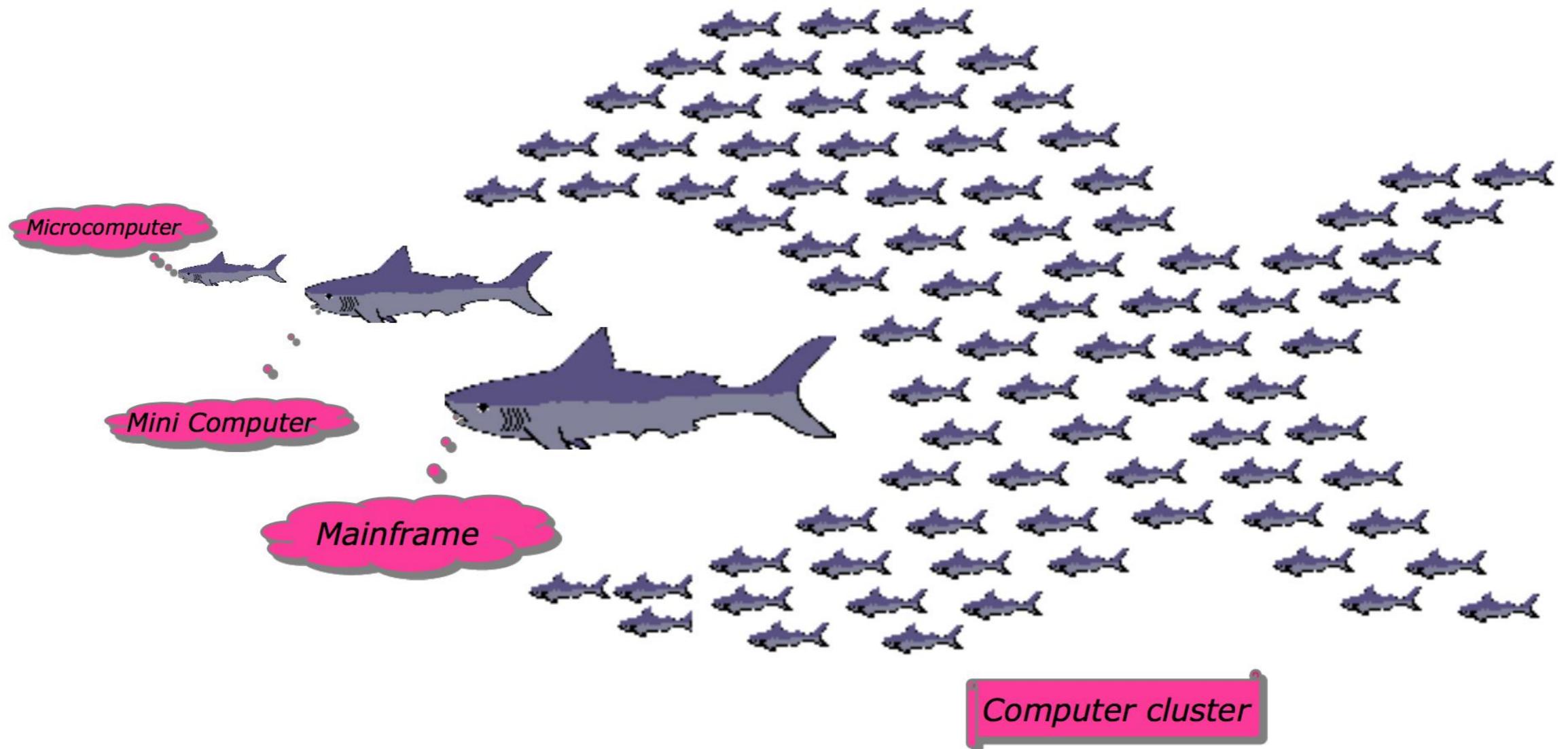


Cluster Computing

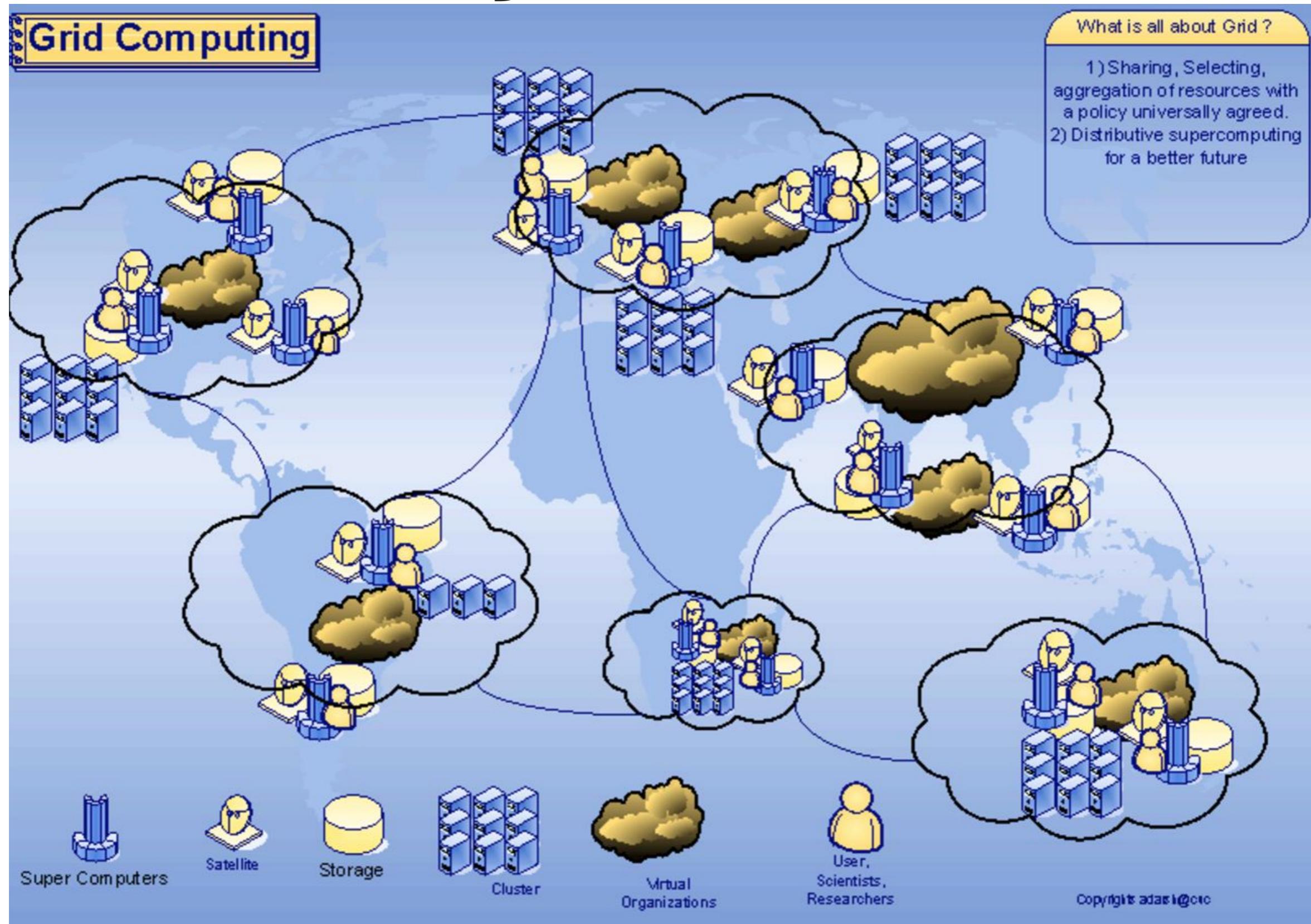
- Clusters are linked computer systems that can co-operate to perform computations and deliver services:
 - Often function / appear as single server.
 - Typically linked over LAN.
 - Offers scalability over single-server.
 - Used for high-availability (redundant), load balancing, shared compute

Cluster Computing

the story of small fish/big fish



Grid Computing

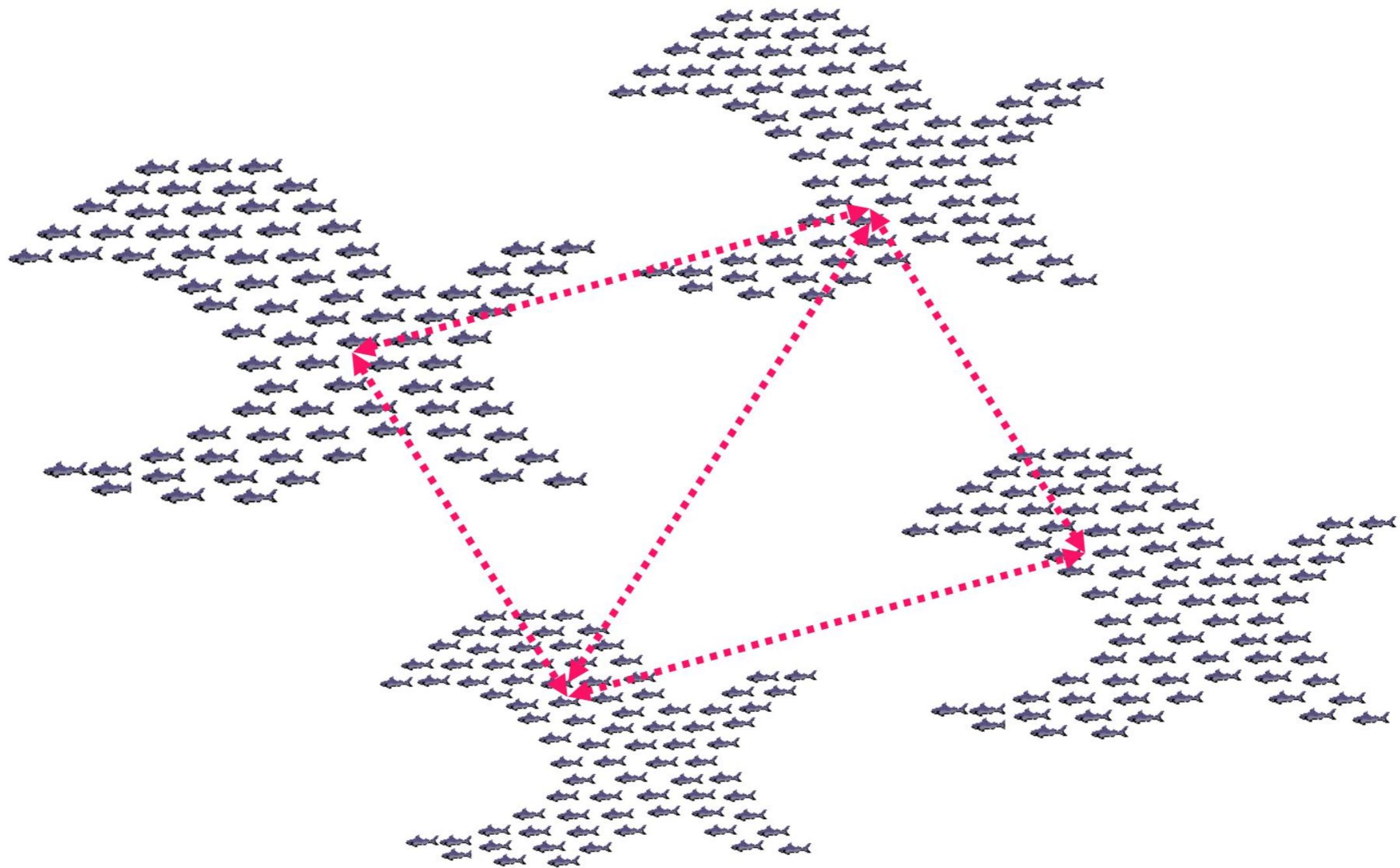


Grid Computing

- Grids are autonomous and dynamic distributed resources contributed by multiple organisations.
- Solve complex problems with large computational power
- Can offer computing, network, sensor and storage resources.
- Resources are loosely coupled, heterogeneous, geographically dispersed.
- Used in diverse fields: climate modelling, drug design and protein analysis to solve “Grand Challenges”

Grid Computing

Service Grids



Grid Computing - Use case

- **BOINC** - Berkeley Open Infrastructure for Network Computing - Enables Grid Computing.

Application of BOINC

- **Folding@home** - Simulates protein folding, drug design and molecular dynamics
- **Einstein@home** - Searches for signals from rotating neutron stars in the data generated from telescopes
- **SETI@home** - Analyze radio signals, searching for signs of extraterrestrial intelligence

What is Cloud Computing?

An analogy: think of electricity services...

You simply plug into a vast electrical grid managed by experts to get a low cost, reliable power supply – available to you with much greater efficiency than you could generate on your own.

Power is a utility service - available to you on-demand and you pay only for what you use.



What is Cloud Computing?

Cloud Computing is also a utility service - giving you access to technology resources managed by experts and available on-demand.



You simply access these services over the internet, with no up-front costs and you pay only for the resources you use.

Cloud Computing....

- Cloud computing is a type of Internet-based computing that provides shared computer processing resources and data to computers and other devices on demand.
- It is a model for enabling, on-demand access to a shared pool of configurable computing resources (e.g., computer networks, servers, storage, applications and services), which can be rapidly provisioned and released with minimal management effort.

Wikipedia

Cloud Computing....

- “Cloud computing has the following characteristics:
 - (1) The illusion of infinite computing resources...
 - (2) The elimination of an up-front commitment by Cloud users...
 - (3). The ability to pay for use...as needed...”
- UC Berkeley RADLabs

Cloud Computing....

- "... a pay-per-use model for enabling available, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."

**- National Institute of Standards and Technology
(NIST)**

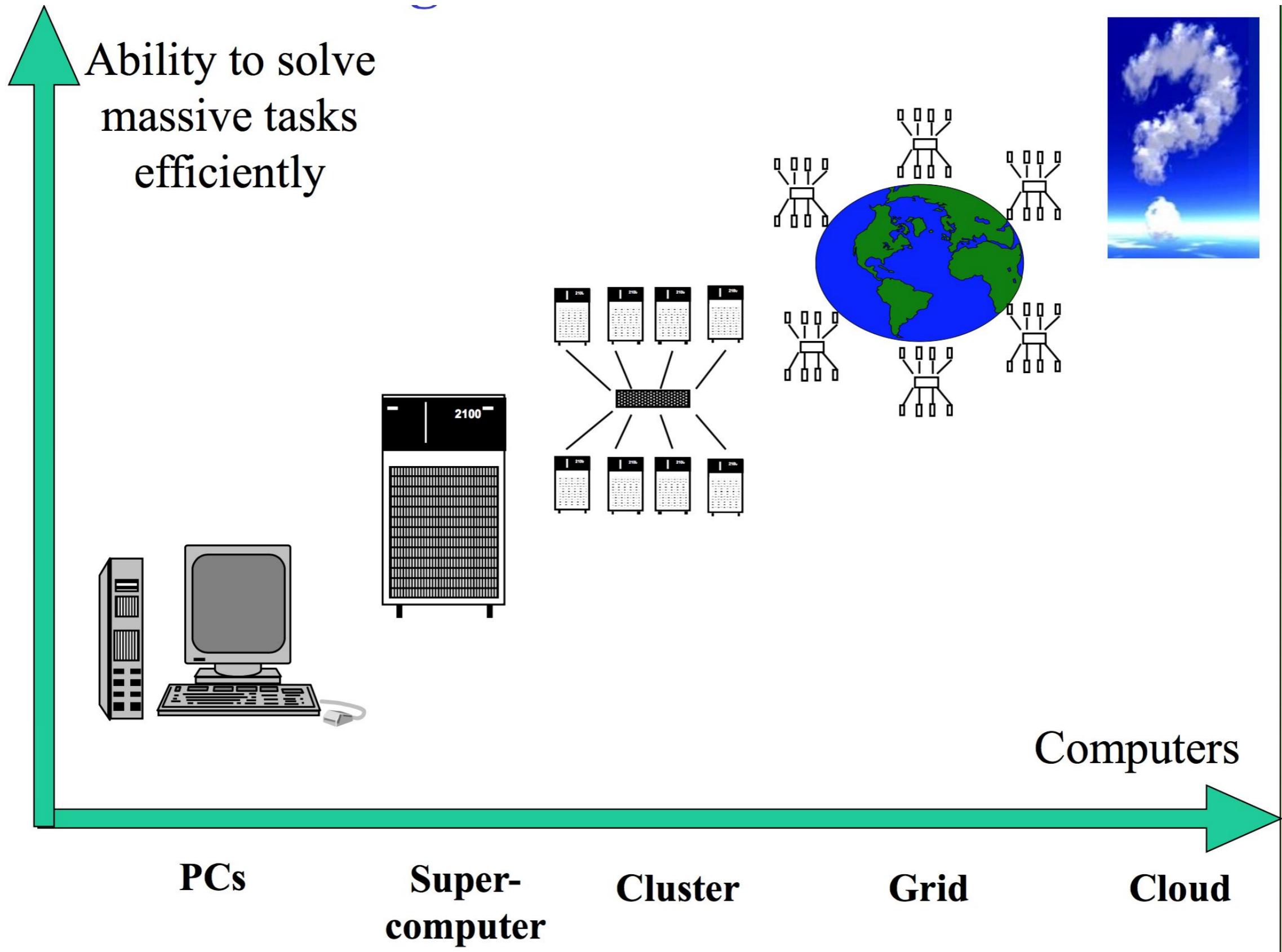
Cloud Computing....

- Cloud computing, often referred to as simply “the cloud,” is the delivery of on-demand computing resources—everything from applications to data centers—over the Internet on a pay-for-use basis.
- **IBM**

Common ground - Cloud Computing..

- Pay as you go. Pay for what you use
- Self-service interface
- Elastic capacity - scale up/down on demand.
- Resources are abstracted / virtualized and endless.
- Faster time to market. Focus on your business

Progress to Cloud Computing



Enabling Technologies

- Virtual Machines / Containers
- Virtualized Storage
- Load Balancing
- AutoScaling
- Virtualized Network
- Monitoring
- Metering
- Multi Tenancy
- Self-healing

Virtualization

Virtual Machines - Run multiple instances on the same machine. Shared machines

Virtual Network - Provides a dedicated network segments in one LAN. Shared network

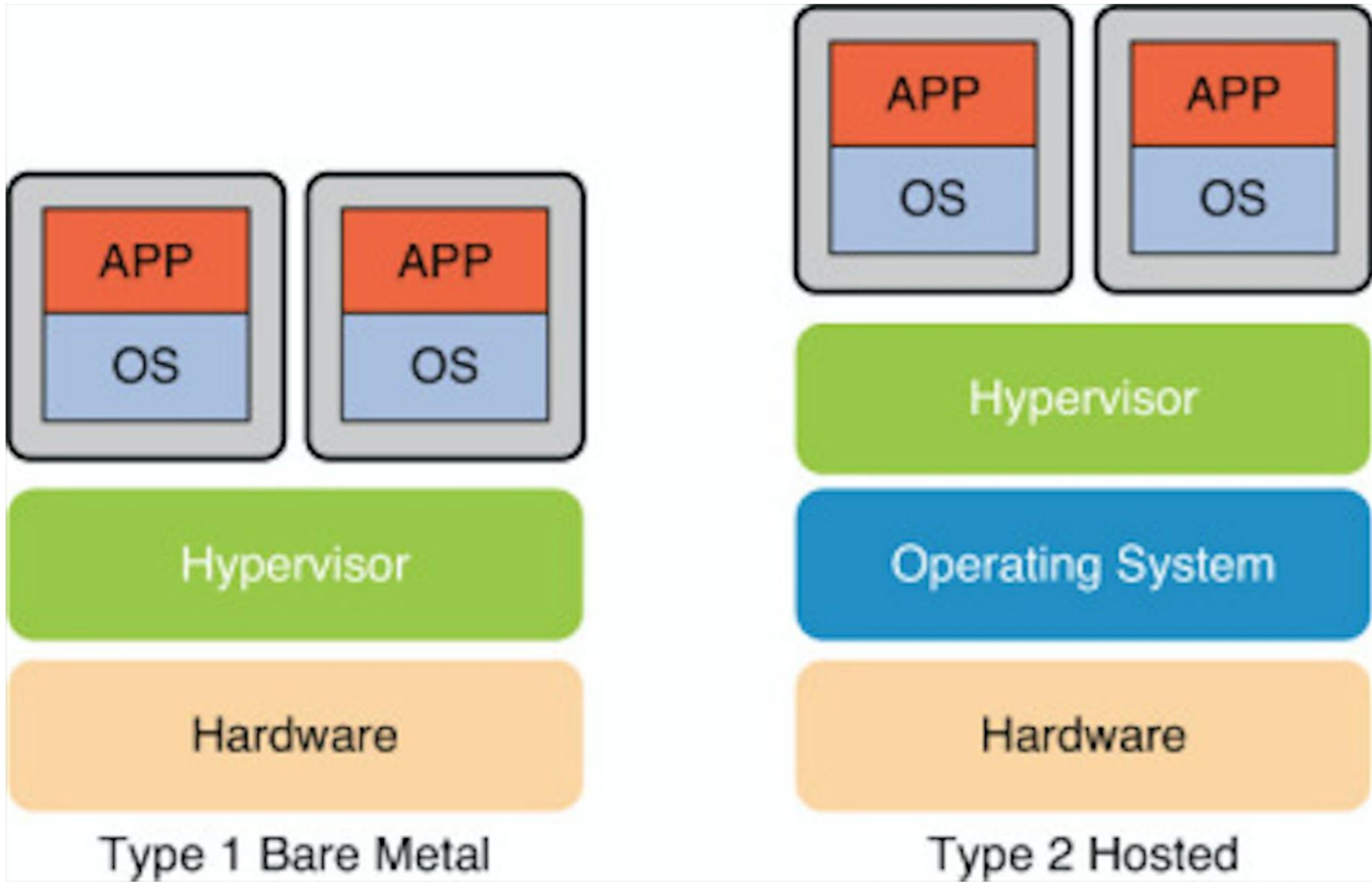
Virtual Storage - Provides a logical space for you to store data. Actual physical location (probably shared) of the data is handled by the underlying system.

Shared, shut-down, restored, re-allocated on-demand!

Enabling Technologies

- Virtual Machines / Containers - Cloud resources
- Virtualized Storage - Cloud resources
- Load Balancing - Cloud resources
- AutoScaling - Cloud resources / Elasticity
- Virtualized Network - Cloud resources
- Monitoring - Pay-as-you go model
- Metering - Pay-as-you go model
- Multi Tenancy - Cloud resources
- Self-healing - Cloud resources

Virtualization



Hypervisors

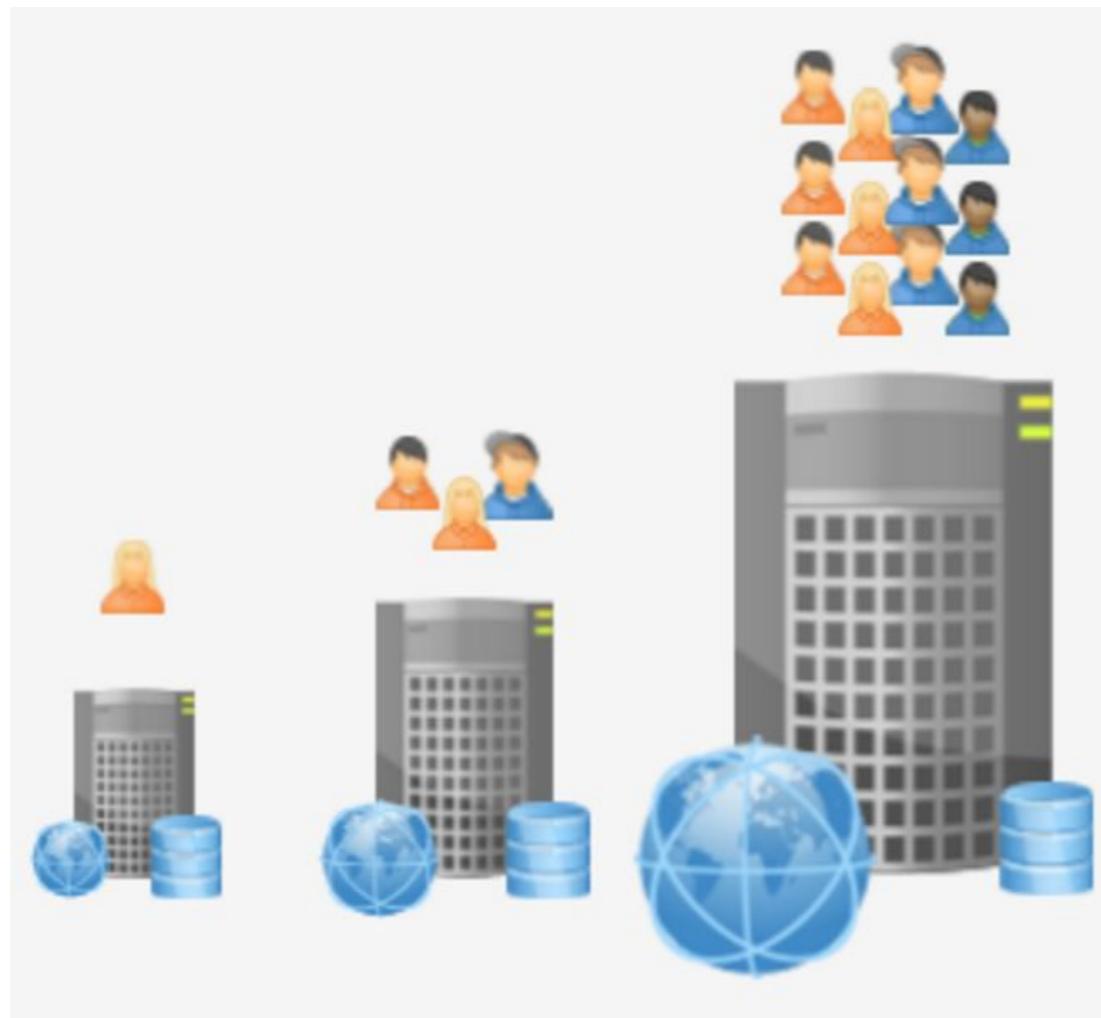
- Type 1, which is considered a bare-metal hypervisor and runs directly on top of hardware. The Type 1 hypervisor is often referred to as a hardware virtualization engine.
- Type 2, which operates as an application on top of an existing operating system.

The need to Scale

- Demand to services fluctuates
 - fluctuation occurs in different time scales
 - year-long cycle/pattern: Christmas shopping, summer travelling
 - fluctuation within 90 min: happy hour credit card deal
- Fixed resource provisioning results in under/over-provisioning
 - **under-provisioning:** slow system response, unable to serve clients
 - **over-provisioning:** waste of resources

Vertical Scaling

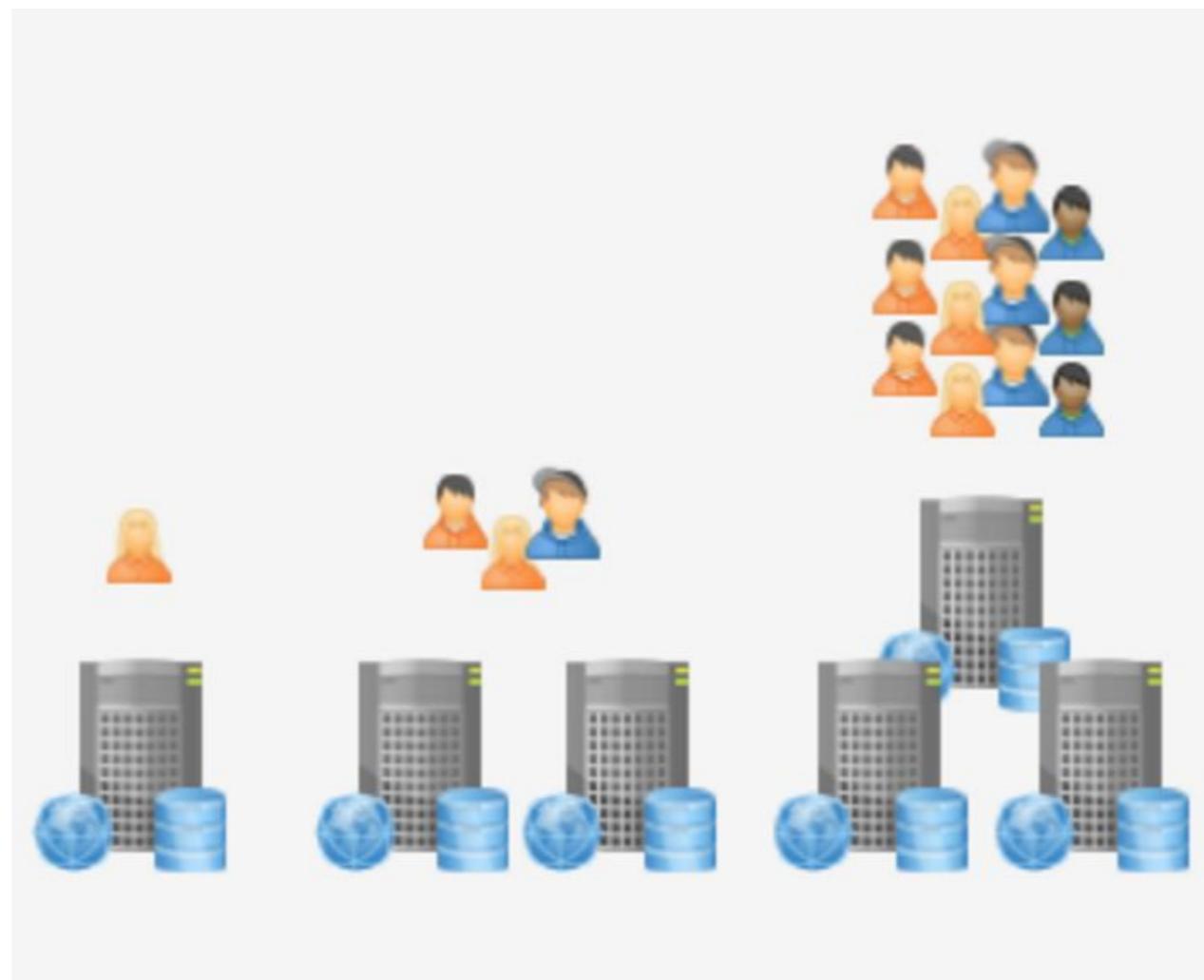
- Vertical scaling means that you **scale** by adding more power (CPU, RAM) to an existing machine
- **Scale up** : more powerful server
- **Scale down** : less powerful server



Auto Scaling

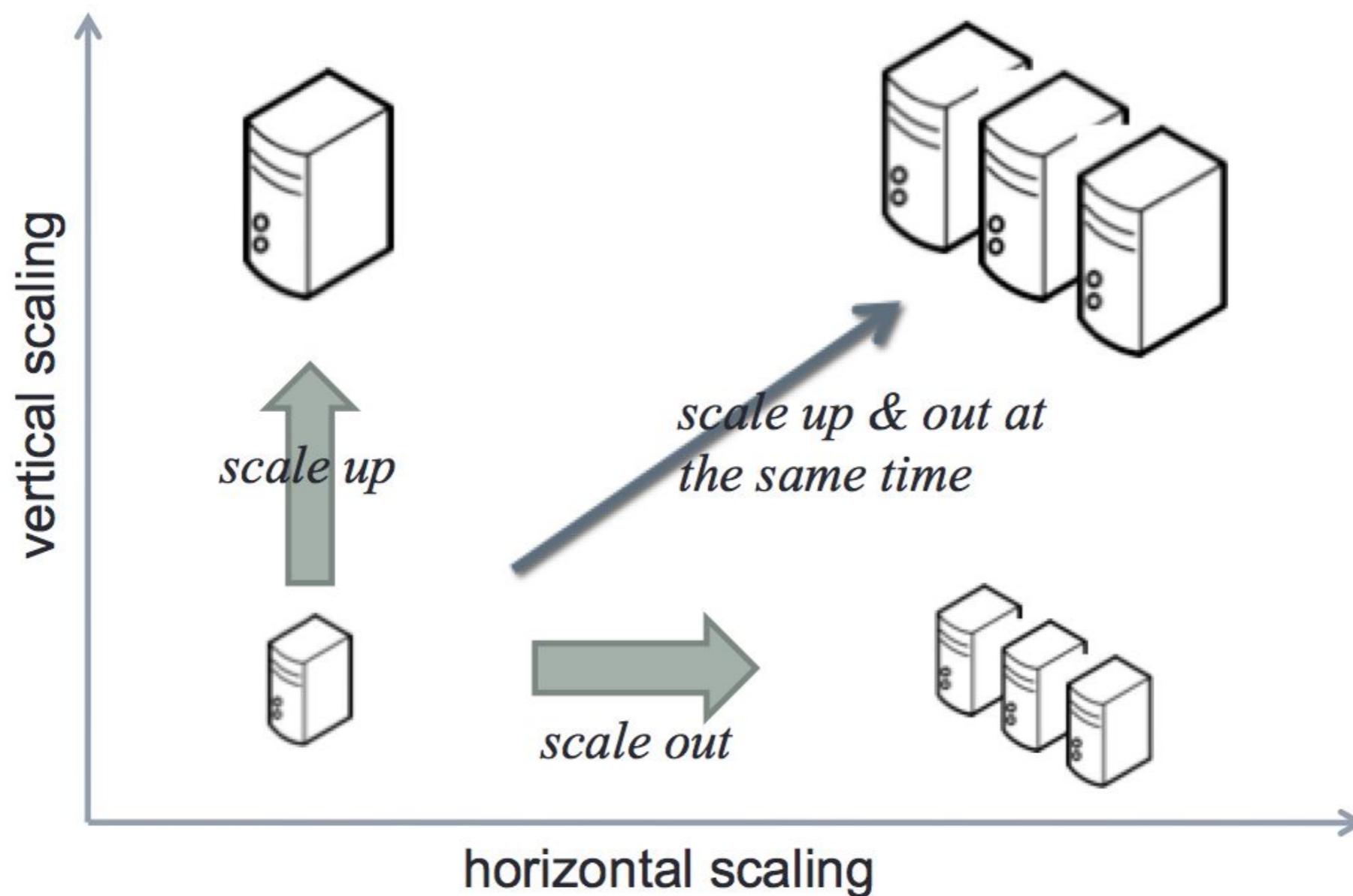
Horizontal Scaling

- **Horizontal scaling** means that you **scale** by adding more machines into your pool of resources
- **Scale out** : more instances
- **Scale in** : few instances

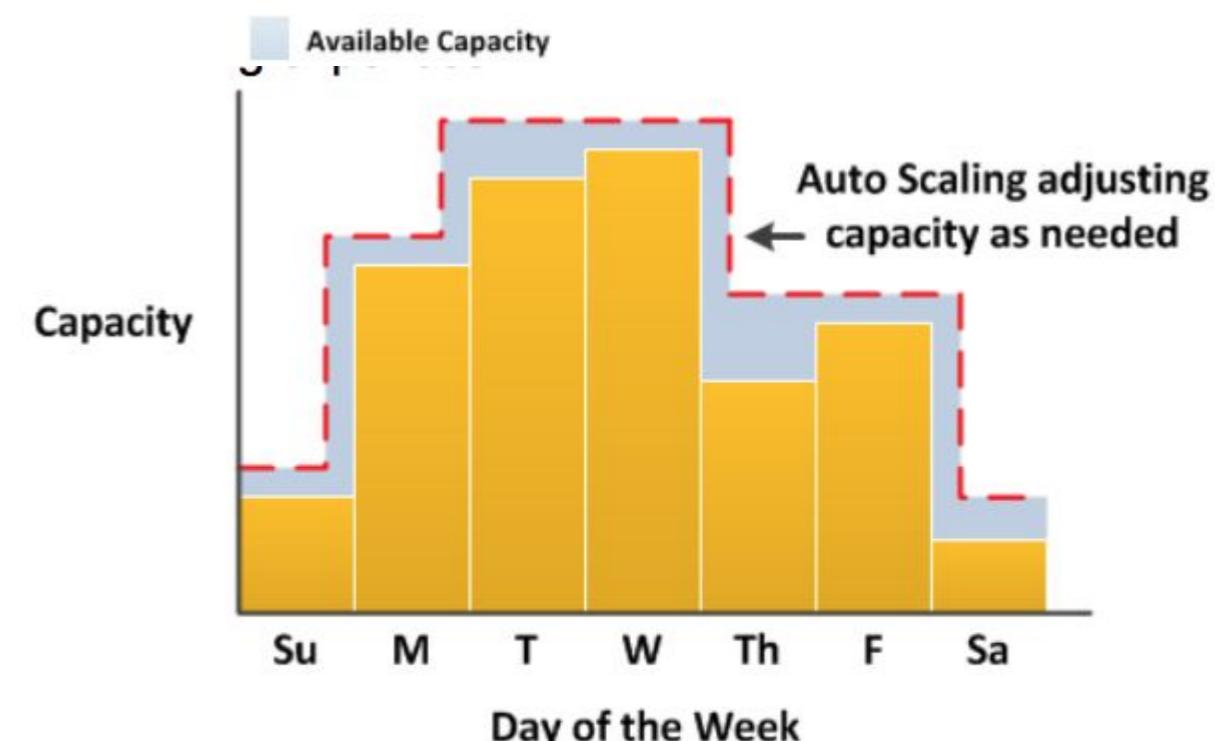
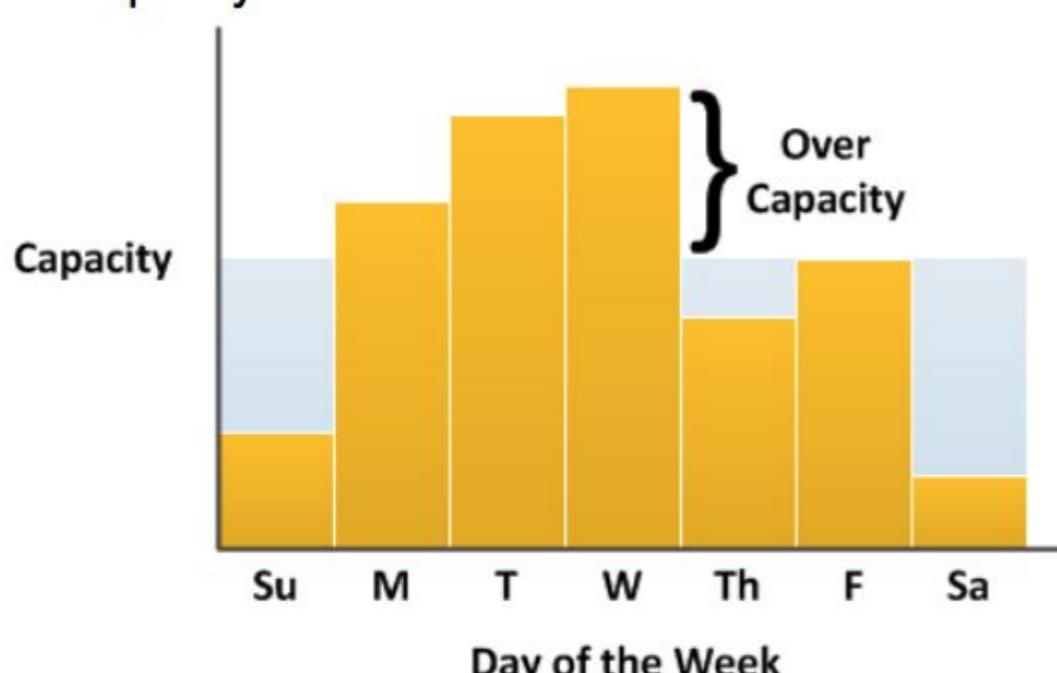
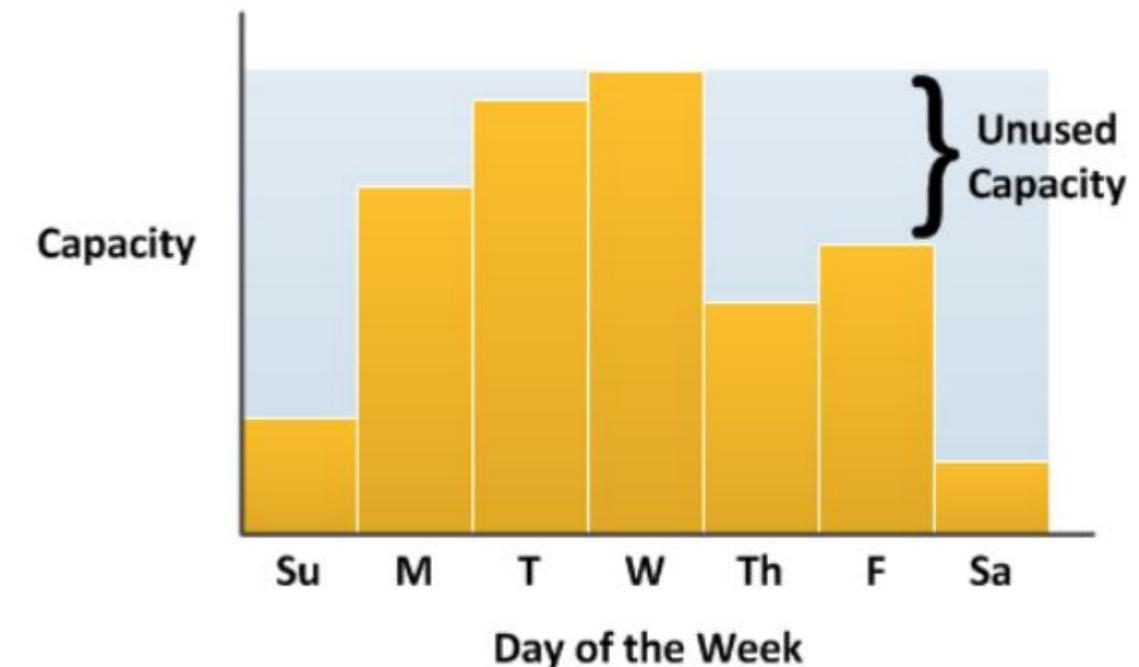
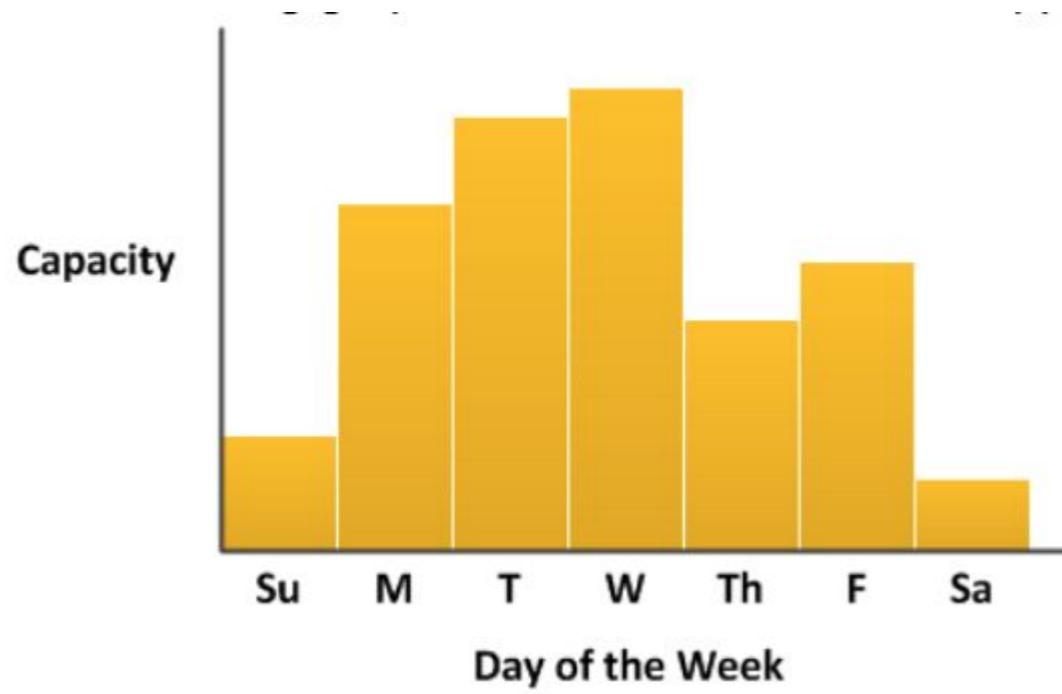


Auto Scaling

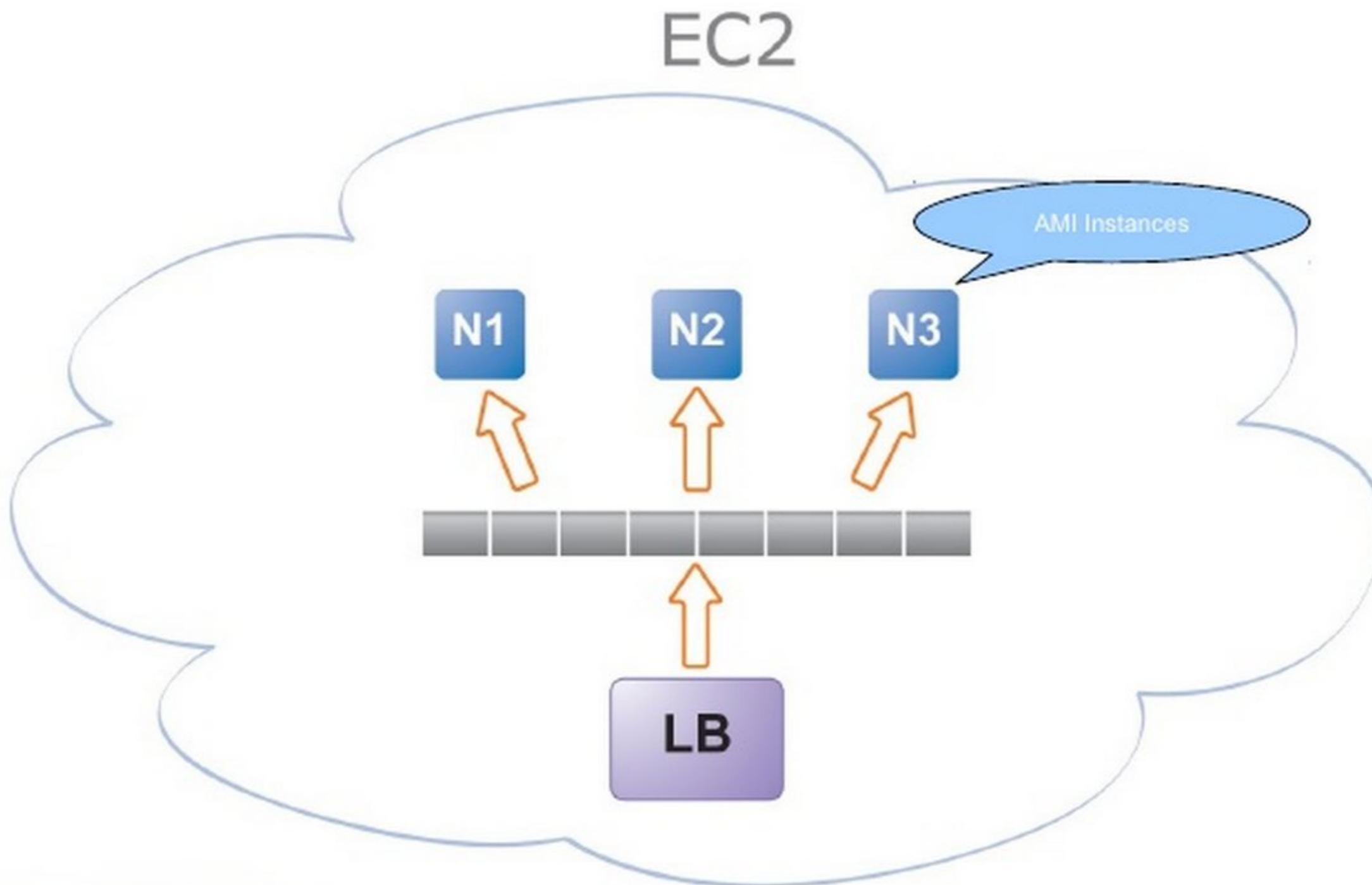
Combine scaling



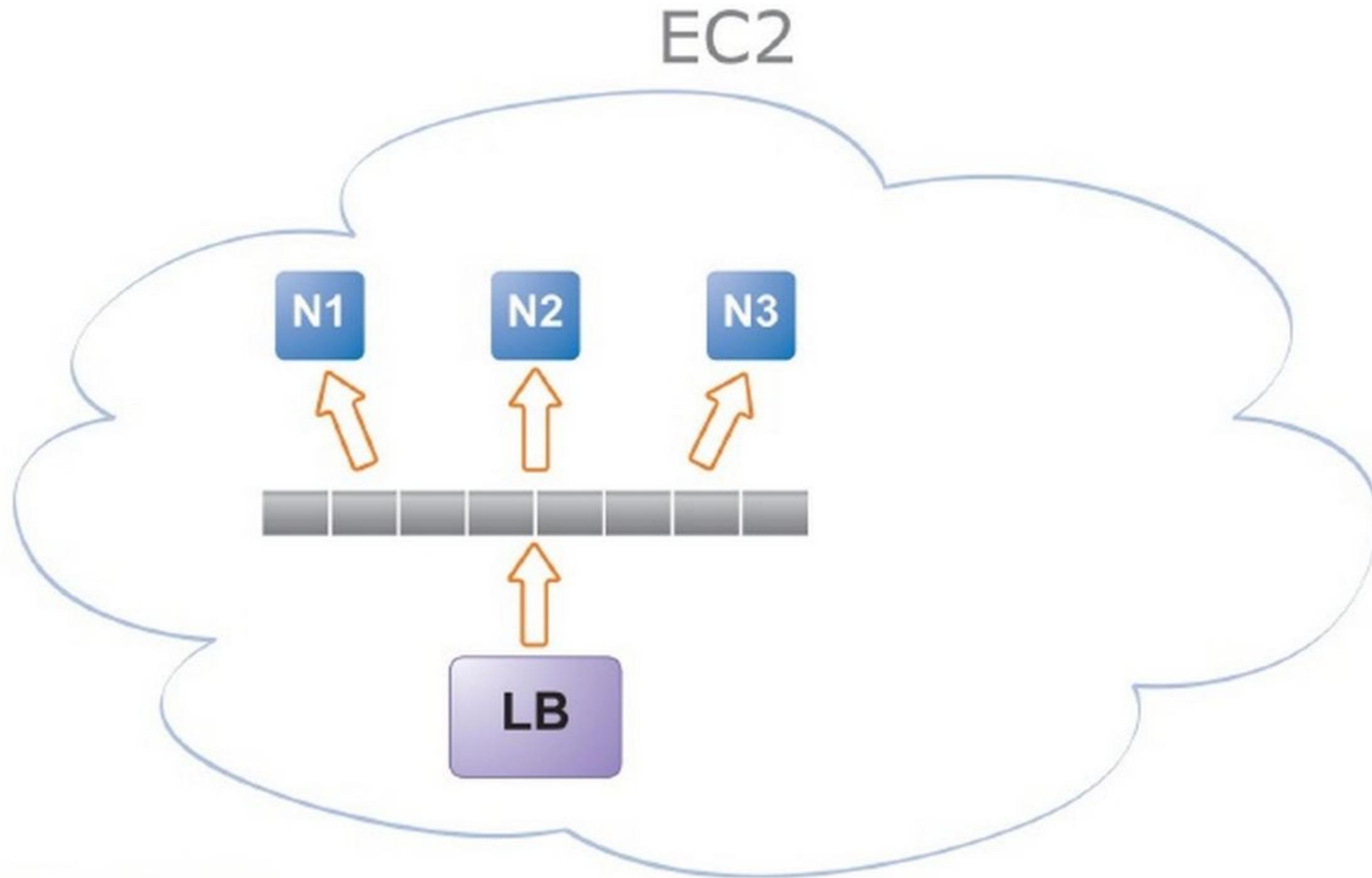
Handling variable Demands



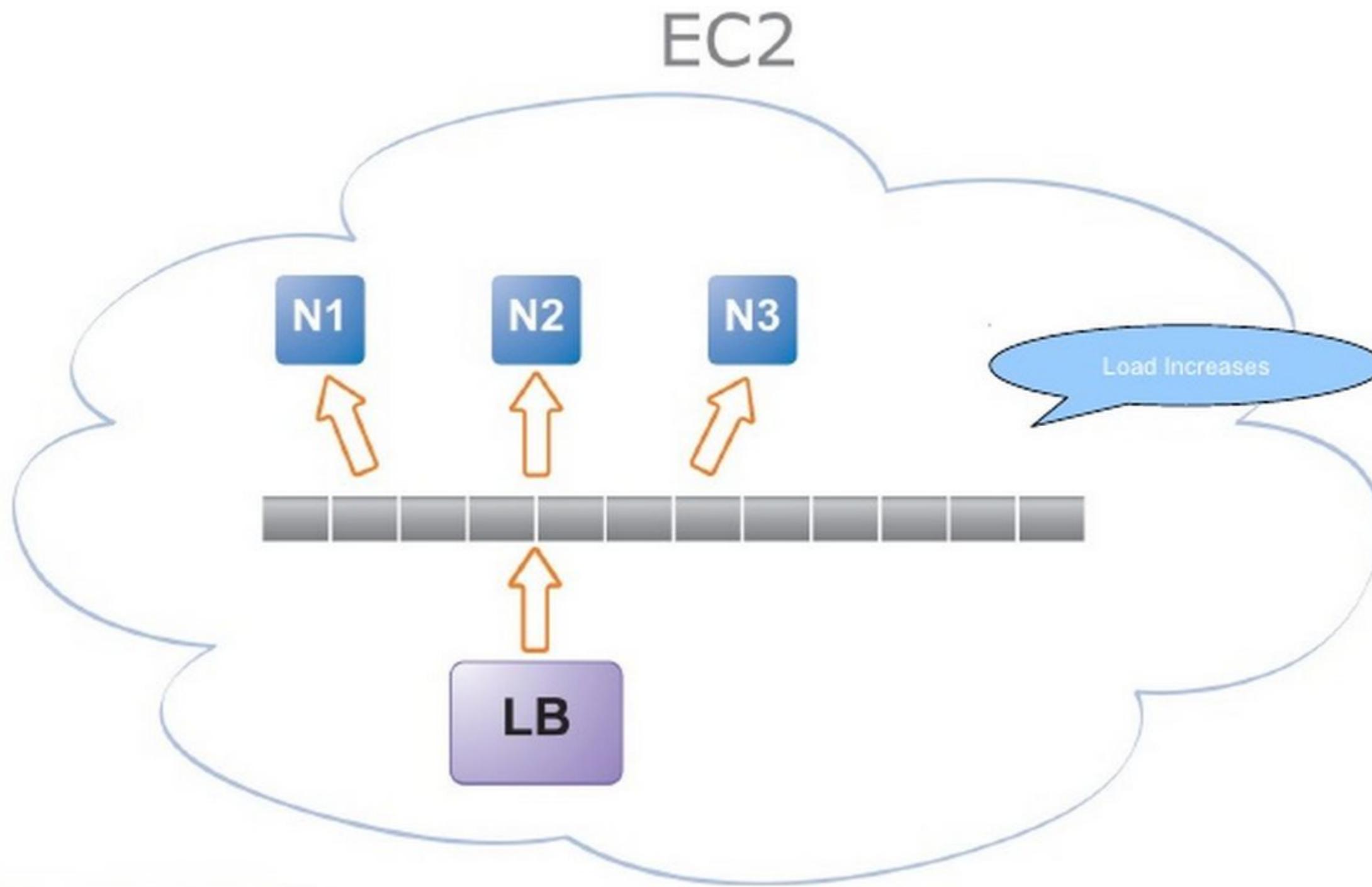
Load Balancing / Autoscaling



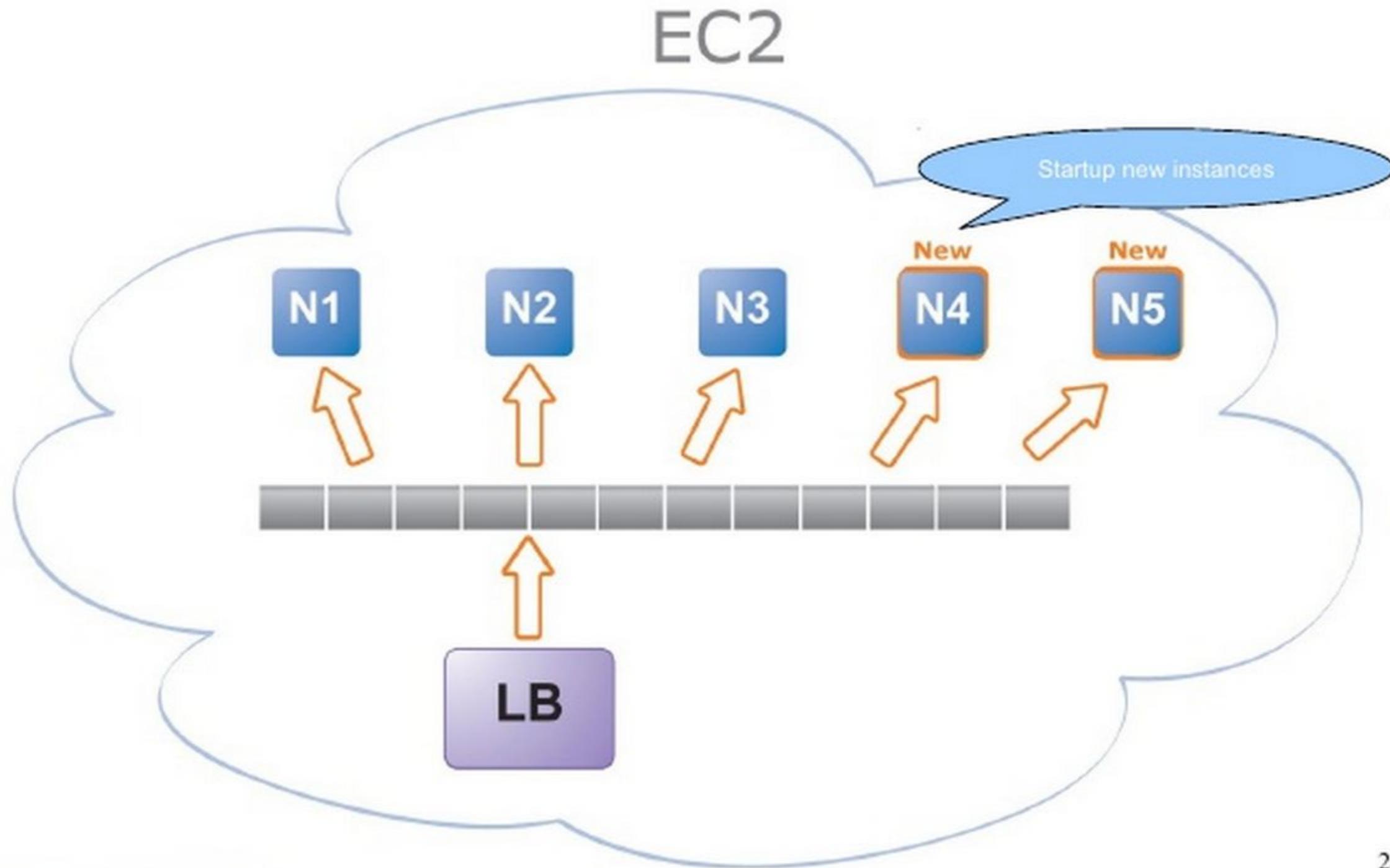
Load Balancing / Autoscaling



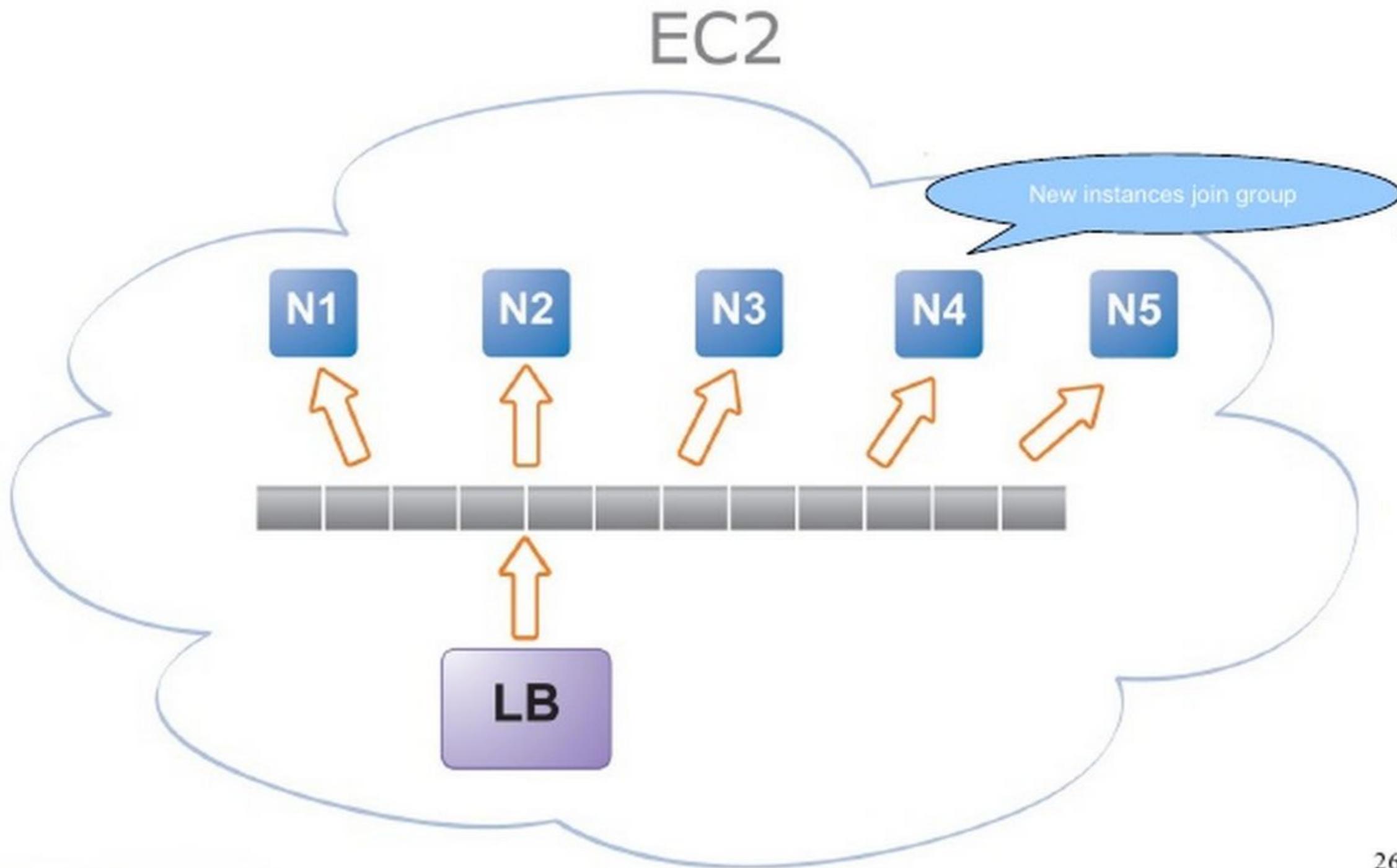
Load Balancing / Autoscaling



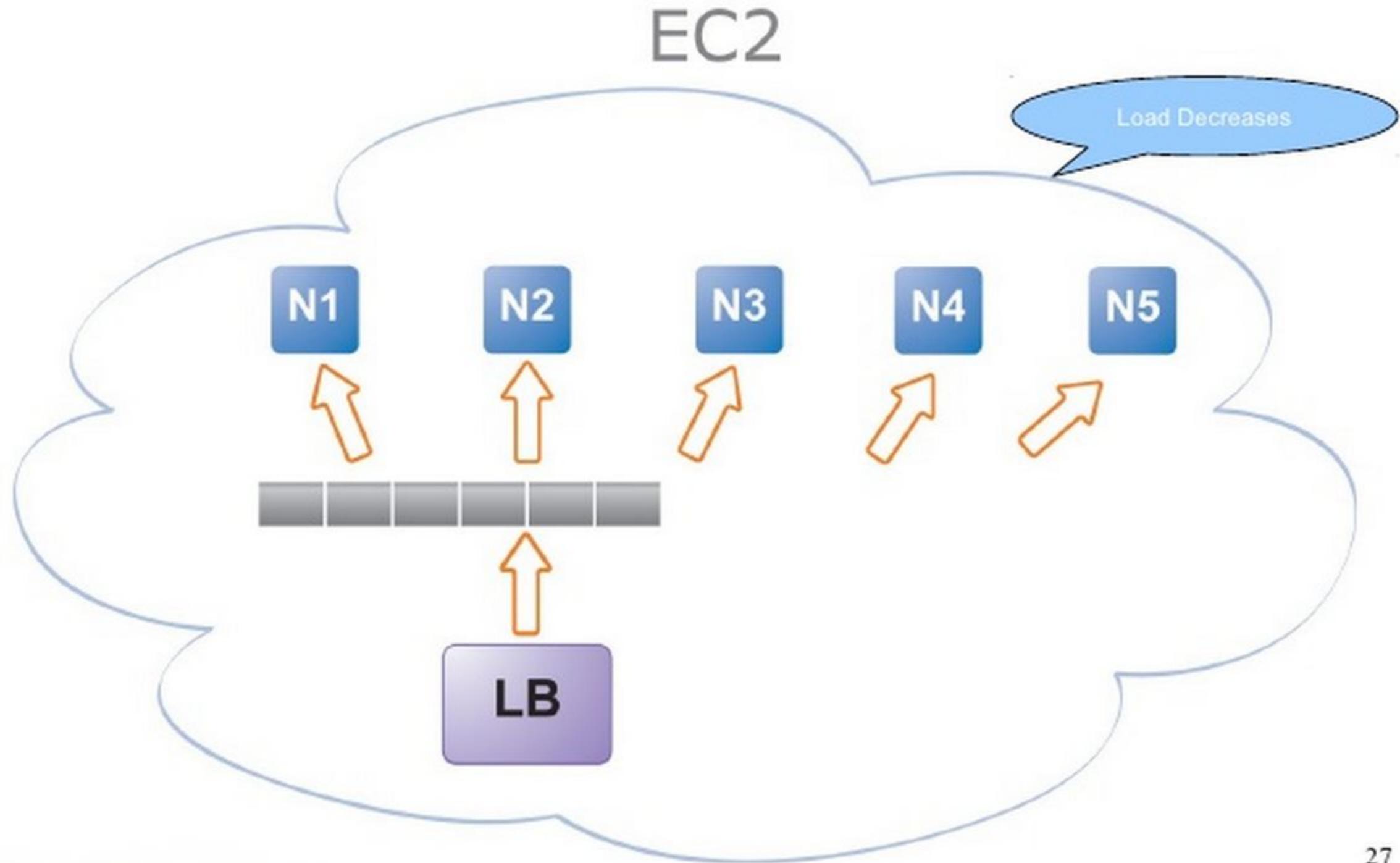
Load Balancing / Autoscaling



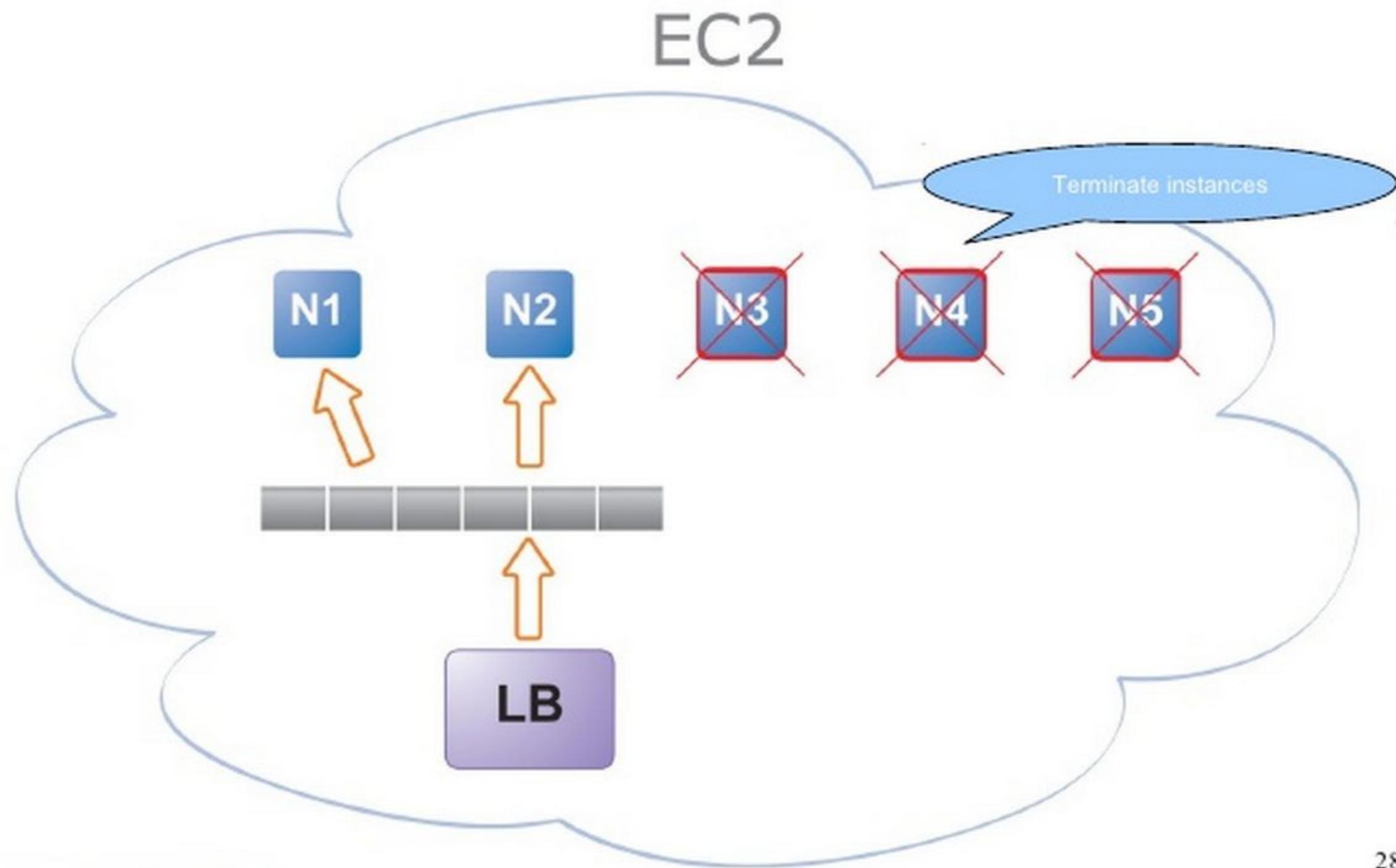
Load Balancing / Autoscaling



Load Balancing / Autoscaling

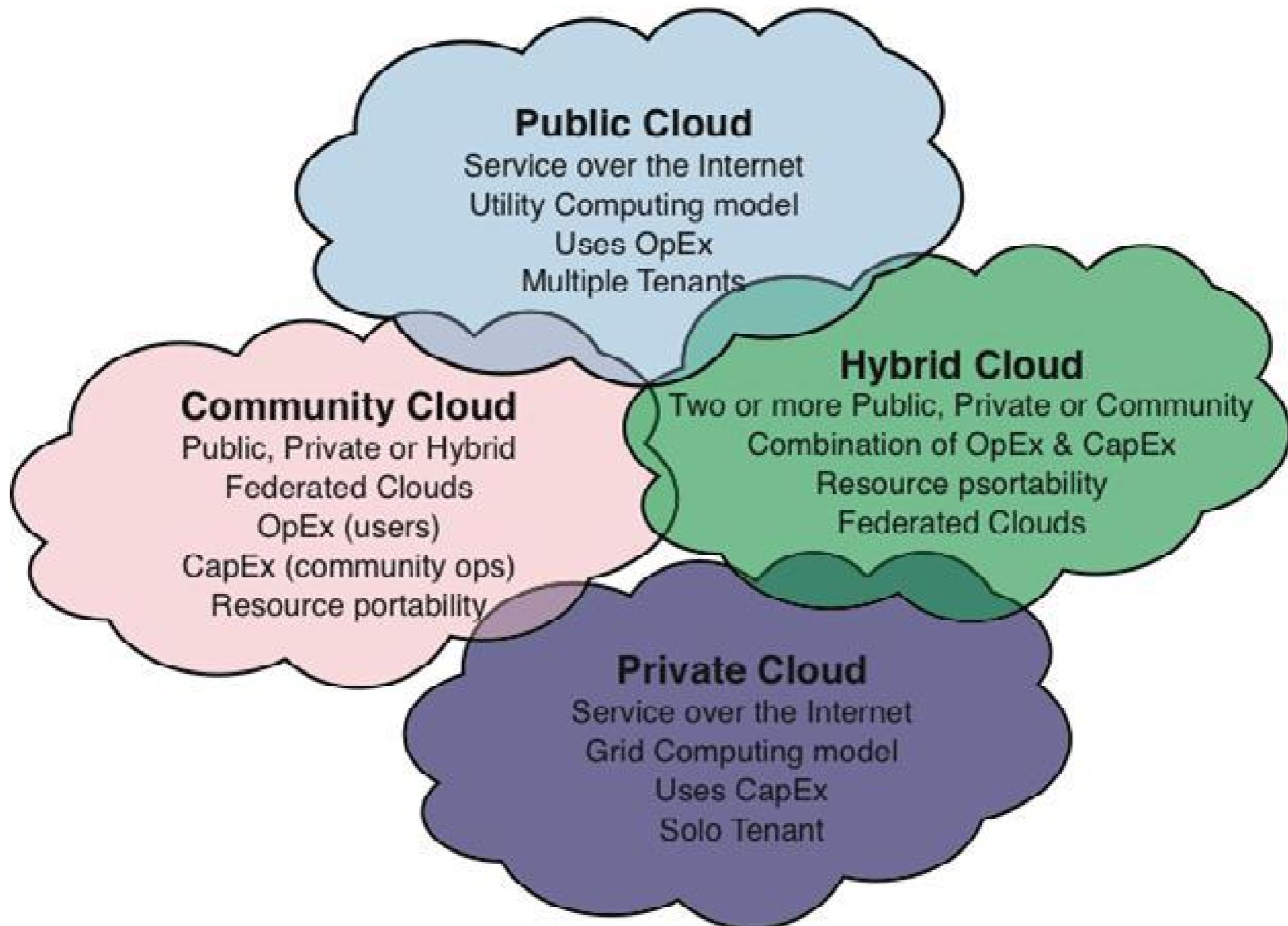


Load Balancing / Autoscaling



Demo

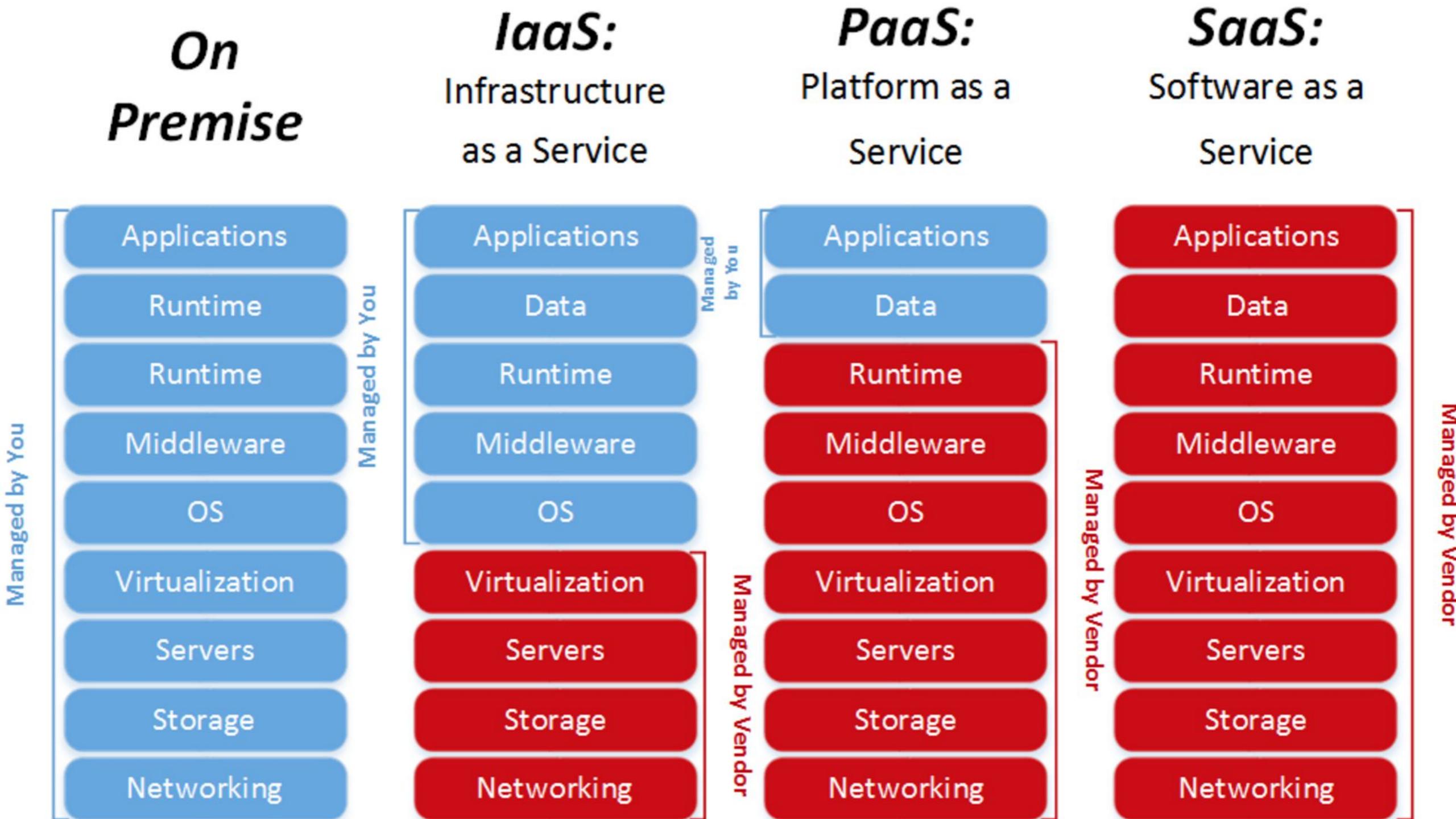
Type of Clouds



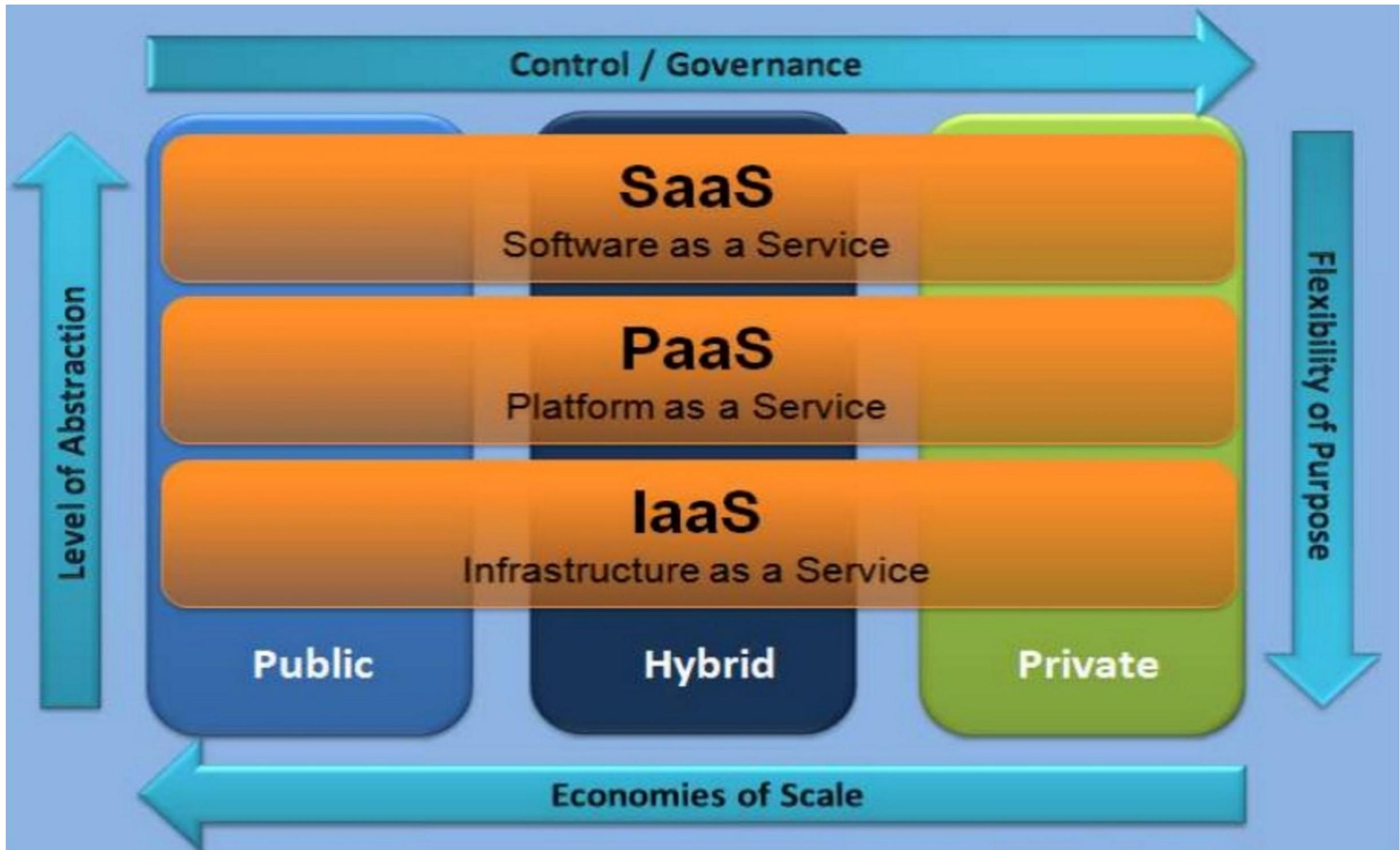
IaaS, PaaS and SaaS

- IaaS - Infrastructure as a Service. Such as diskspace, machine power, networking from the cloud
- PaaS - Platform as a Service. Platform such as tomcat from the Cloud managed
- SaaS - Software as a Service. Software from the Cloud

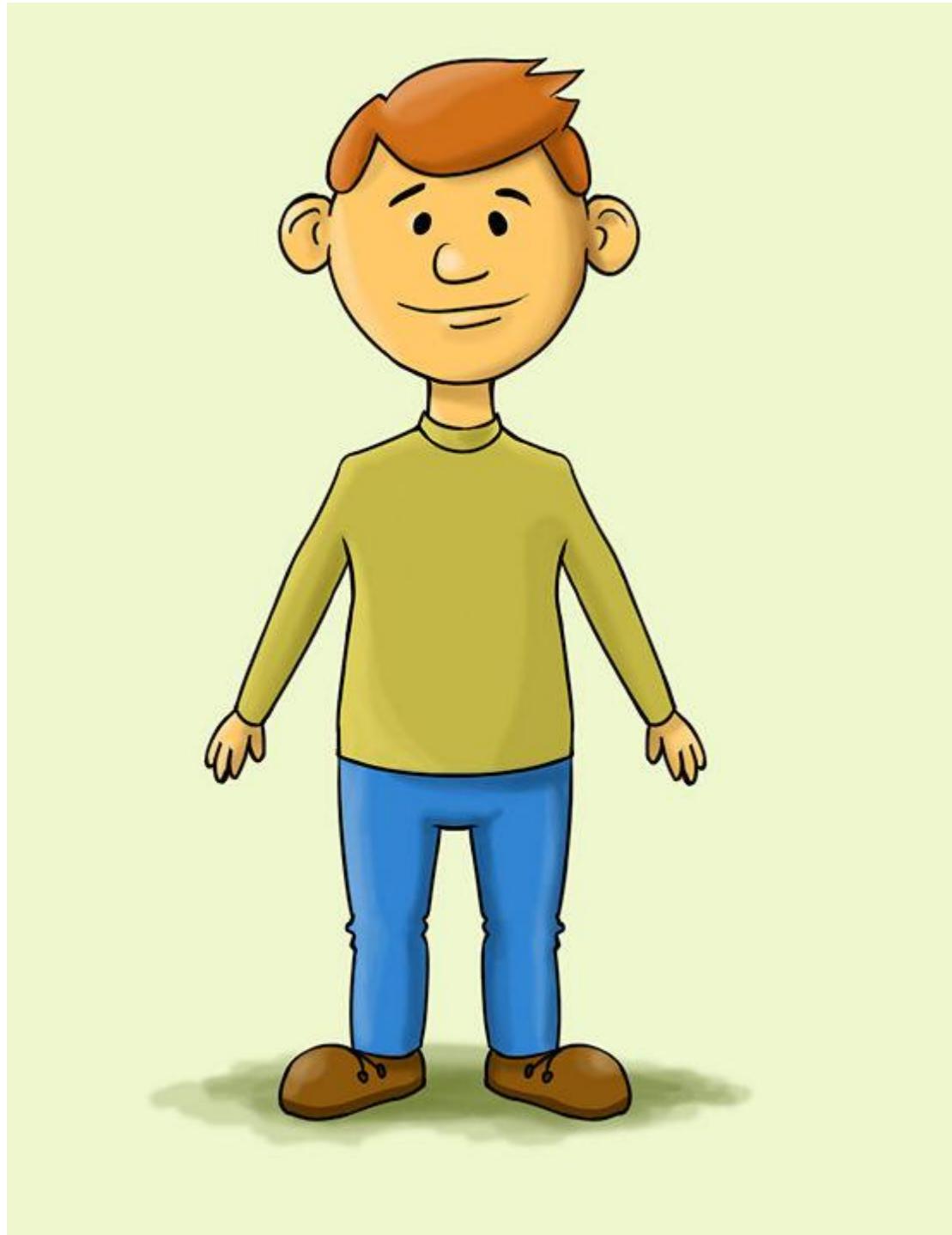
Service Types



Service Model



Please meet Joe...



- My Name is Joe
- I am a software Architect
- I love cloud computing
- I love to take challengers
- I want to find a job

Vacancy - Cloud Architect

A **cloud architect** is an IT professional who is responsible for overseeing a company's **cloud computing strategy**. This includes cloud adoption plans, cloud application design, and cloud management and monitoring. Cloud architects oversee application architecture and deployment in cloud environments -- including **public cloud, private cloud and hybrid cloud**. Additionally, cloud architects act as consultants to their organization and need to stay up-to-date on the latest trends and issues.

Qualifications for the position should include a strong understanding of cloud computing technology and infrastructure as well as experience designing and migrating applications to the cloud. Cloud architects should have experience in a consultant role, as they need to build relationships with customers and team members.

Meeting @ First day

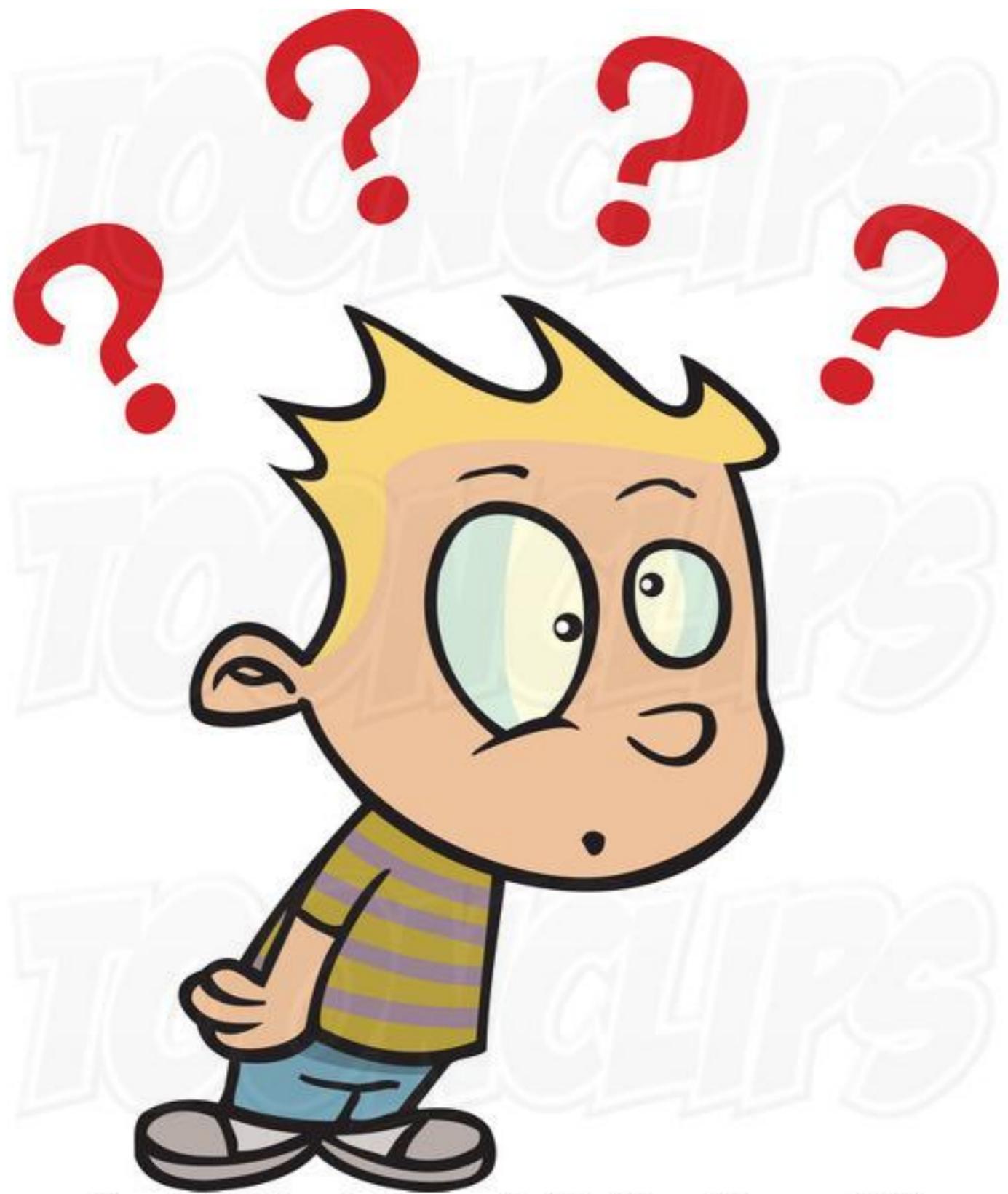


Meeting notes

- They are selling Michael Jackson's music albums
- They have a web App
- Written in PHP
- They have bunch of developers
- They are experts in their business domain
- They want to go to market quickly. (ASAP)
- After launching within couple of weeks they are expecting million of users come to website
- They have an Operation team
- Hired me for Operation team
- My team members
 - Only me :O



My Feeling :)



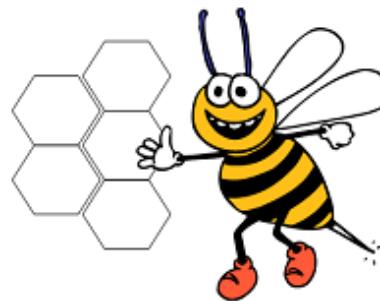
I start thinking.....



- I have learn Cloud Computing
- I trust myself

Lecture 11 - Microservices

'Microservices'?



Buzzword!



hype.

'Monolithic' Architecture

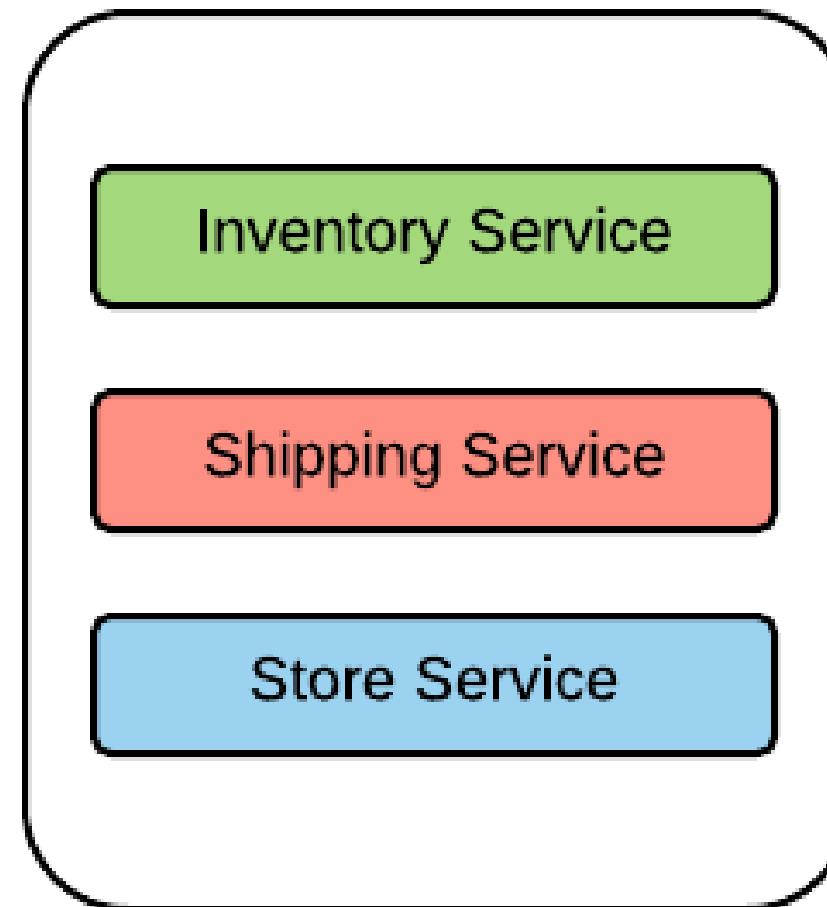


Monolithic Architecture

- All functionalities are implemented/deployed into a single software application.
 - Enterprise software applications - ERPs, CRMs etc.
 - SOA/web services: 'coarse-grained' services, broad scope, mammoth services with several dozens of operations and complex message formats

Monolithic Architecture

- Use case : Online *Retail software application* which comprises of multiple business functionalities.



Monolithic Architecture

- Developed and deployed as a single unit.
- Overwhelmingly complex; which leads to nightmares in maintaining, upgrading and adding new features.
- Redeploy the entire application, in order to update a part of it.
- Scaling : scaled as a single application and difficult to scale with conflicting resource requirements
- Reliability - One unstable service can bring the whole application down.
- Hard to innovate, practice agile development and delivery methodologies

'Microservices' Architecture

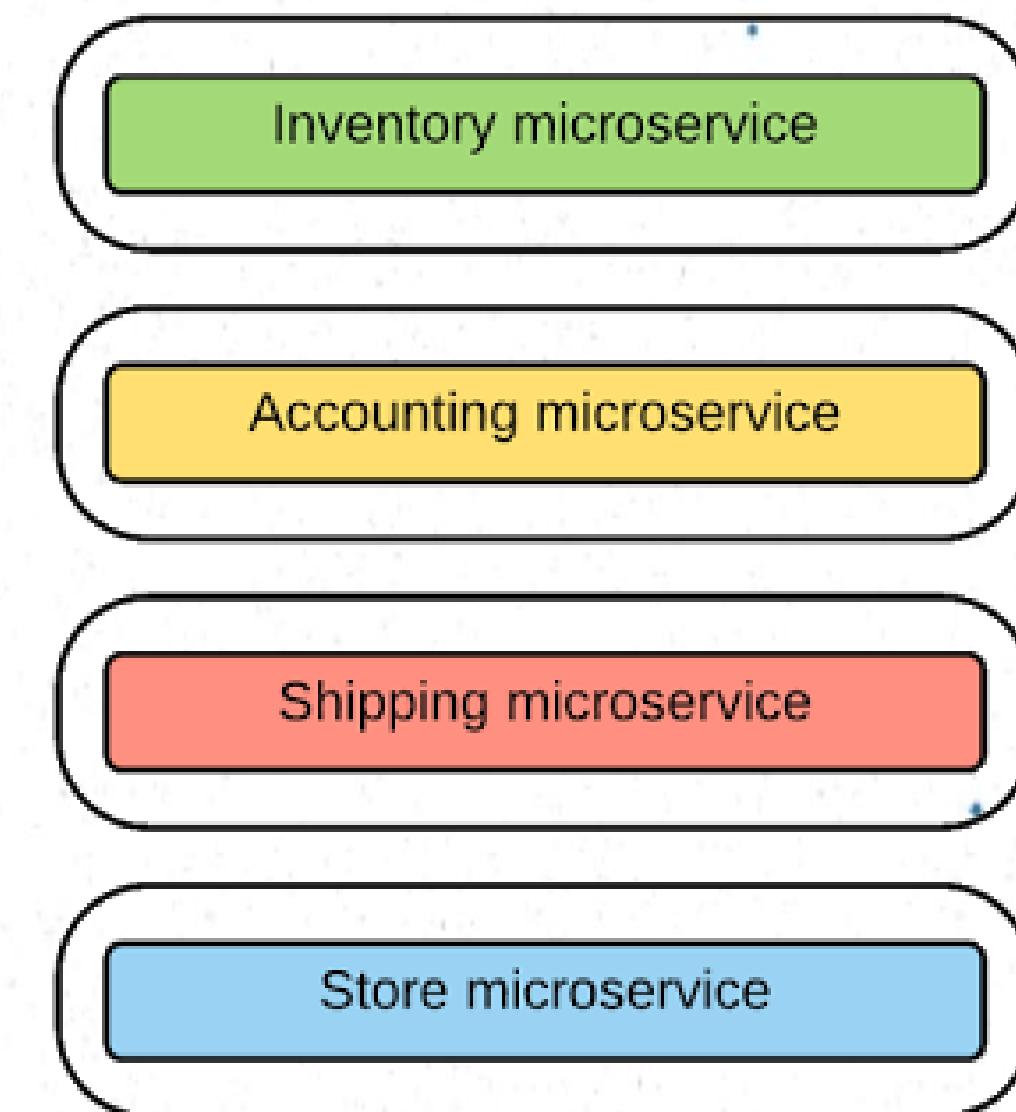


Microservices Architecture

- *The foundation of microservices architecture(MSA) is about developing a single application as a **suite of fine-grained and independent services** that are running in its own process, developed and deployed independently*
- Its just more than segregating the services in a monolith.

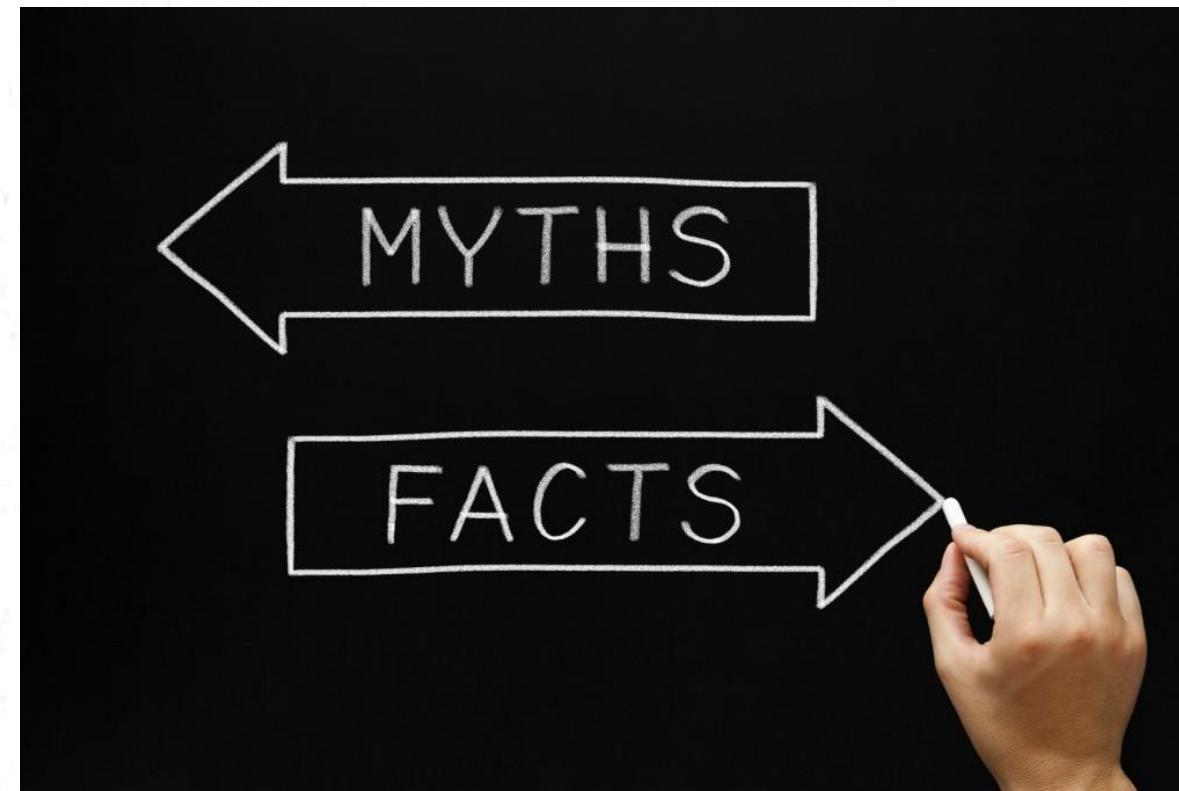
Microservices Architecture

- **Use case :** Online retail application can be implemented with a suite of microservices



Designing Microservices : Size, scope and capabilities

- **Common Misconceptions**
 - Lines of Code
 - Team size
 - 'Micro' is a bit misleading term
 - Use web services and rebranding them as microservices



Designing Microservices : Size, scope and capabilities

- Single Responsibility Principle(SRP): Having a limited and a focused business scope.
- Find the service boundaries and align them with the business capabilities .
- Make sure the microservices design ensures the agile/independent development and deployment of the service.
- Focus on scope of the microservice, but not about making the the service smaller- righted sized services
- Unlike service in web services, a given microservice should have a very few operations/functionalities and simple message format.
- Start with relatively broad service boundaries to begin with, refactoring to smaller ones (based on business requirements) as time goes on.

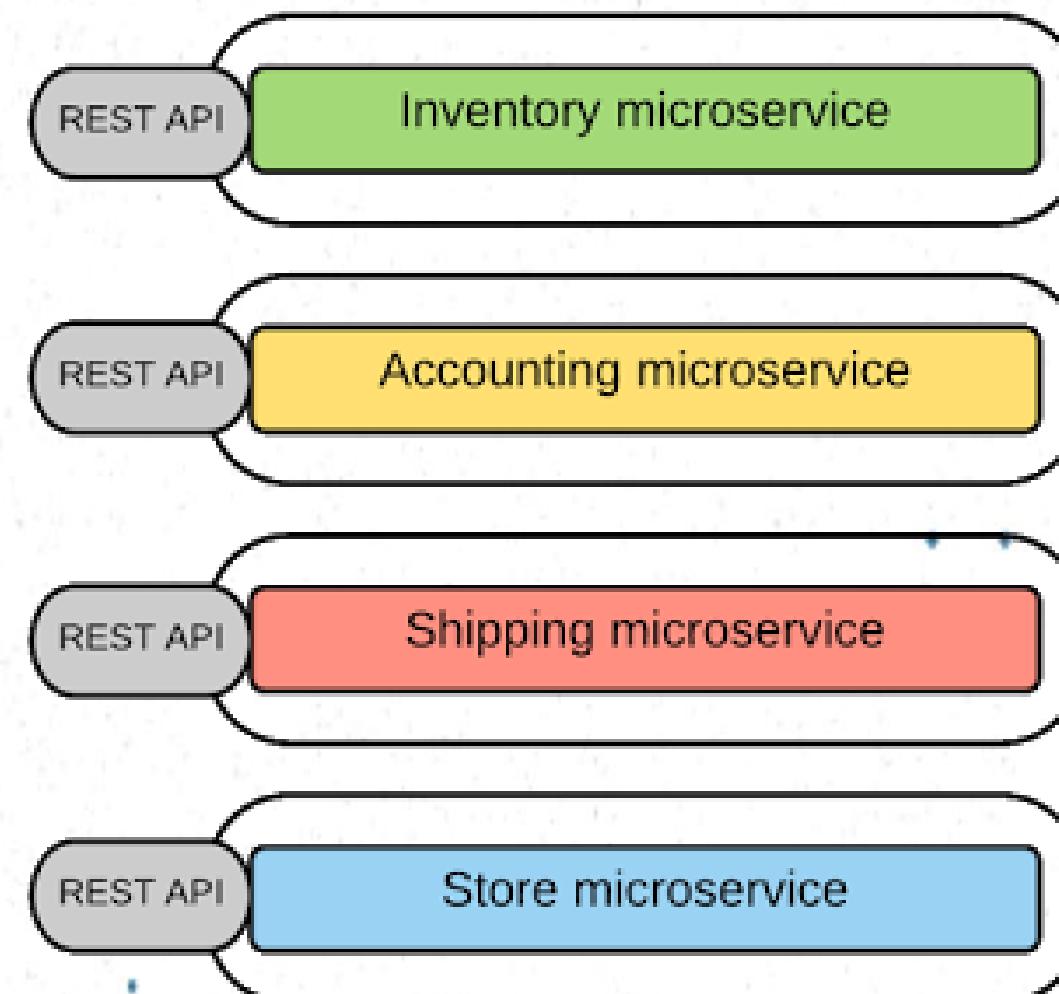
Messaging in Microservices

- In **Monolithic architecture**:
 - Function calls or language-level method calls
 - SOA/web services : SOAP and WS* with HTTP, JMS etc.
 - Webservices with several dozens of operations and complex message schemas
- In **Microservice architecture**:
 - Simple and lightweight messaging mechanism.

Messaging in Microservices

- **Synchronous Messaging**

- *Client expects a timely response from the service and waits till it gets it.*
- REST, Thrift



Messaging in Microservices

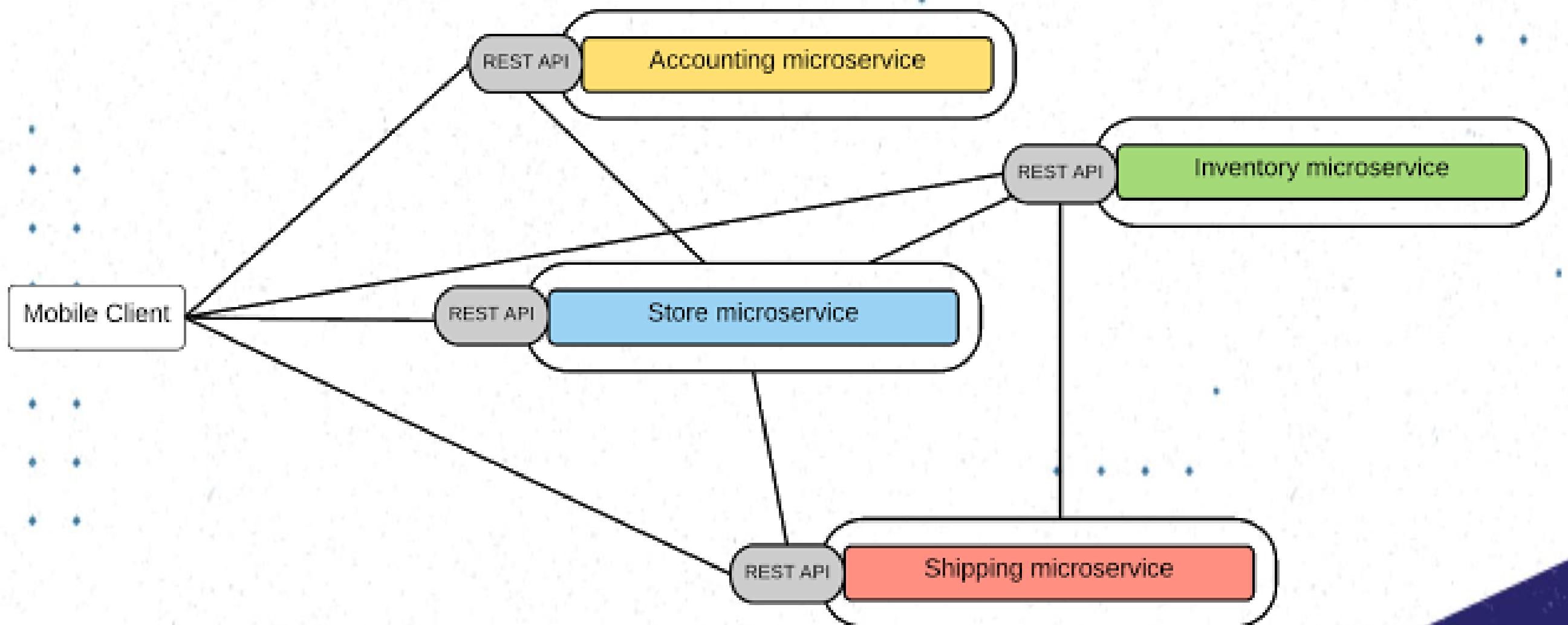
- **Asynchronous Messaging**
 - Client doesn't expect a response immediately, or not accept a response at all
 - AMQP, STOMP, MQTT
- **Message Formats**
 - JSON, XML, Thrift, ProtoBuf, Avro
- **Service Contracts**
 - Defining the service interfaces - Swagger, RAML, Thrift IDL

Integrating Microservices (Inter-service Communication)

- Required to have the communication structures between different microservices.
- SOA/web services used ESB.
- Microservices promotes to eliminate the central message bus/ESB and move the 'smart-ness' or business logic to the services and client(known as 'Smart Endpoints').
- Connect services through 'dumb' pipes.
-

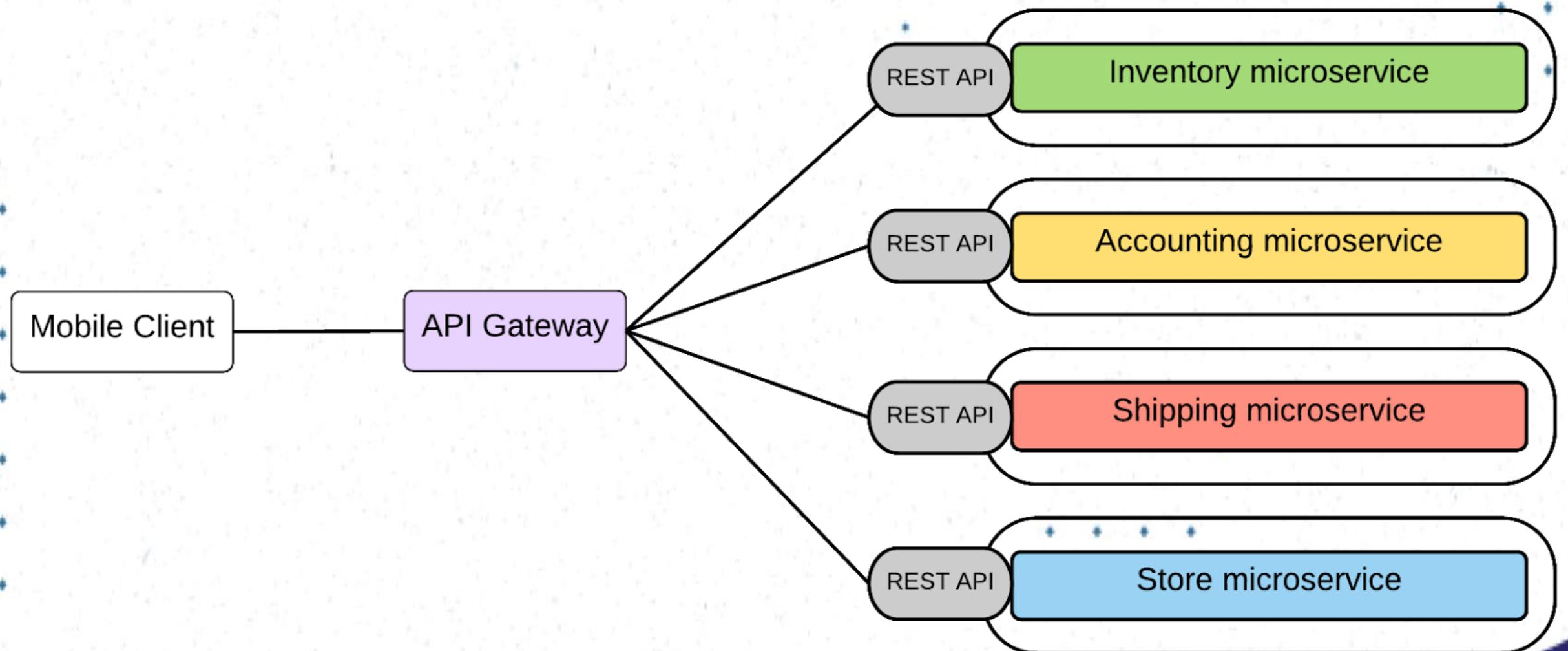
Integrating Microservices (Inter-service Communication)

- Point-to-point style - Invoking services directly



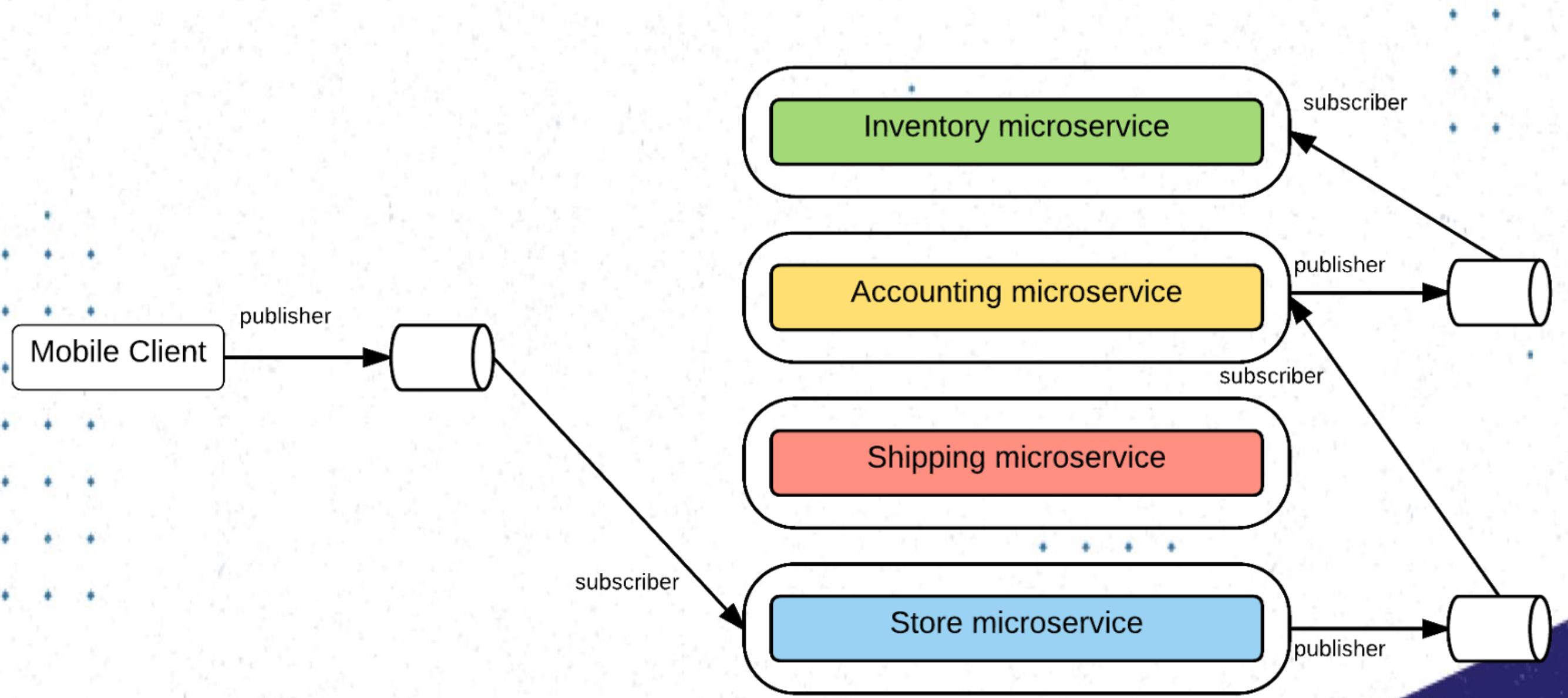
Integrating Microservices (Inter-service Communication)

- API-Gateway style



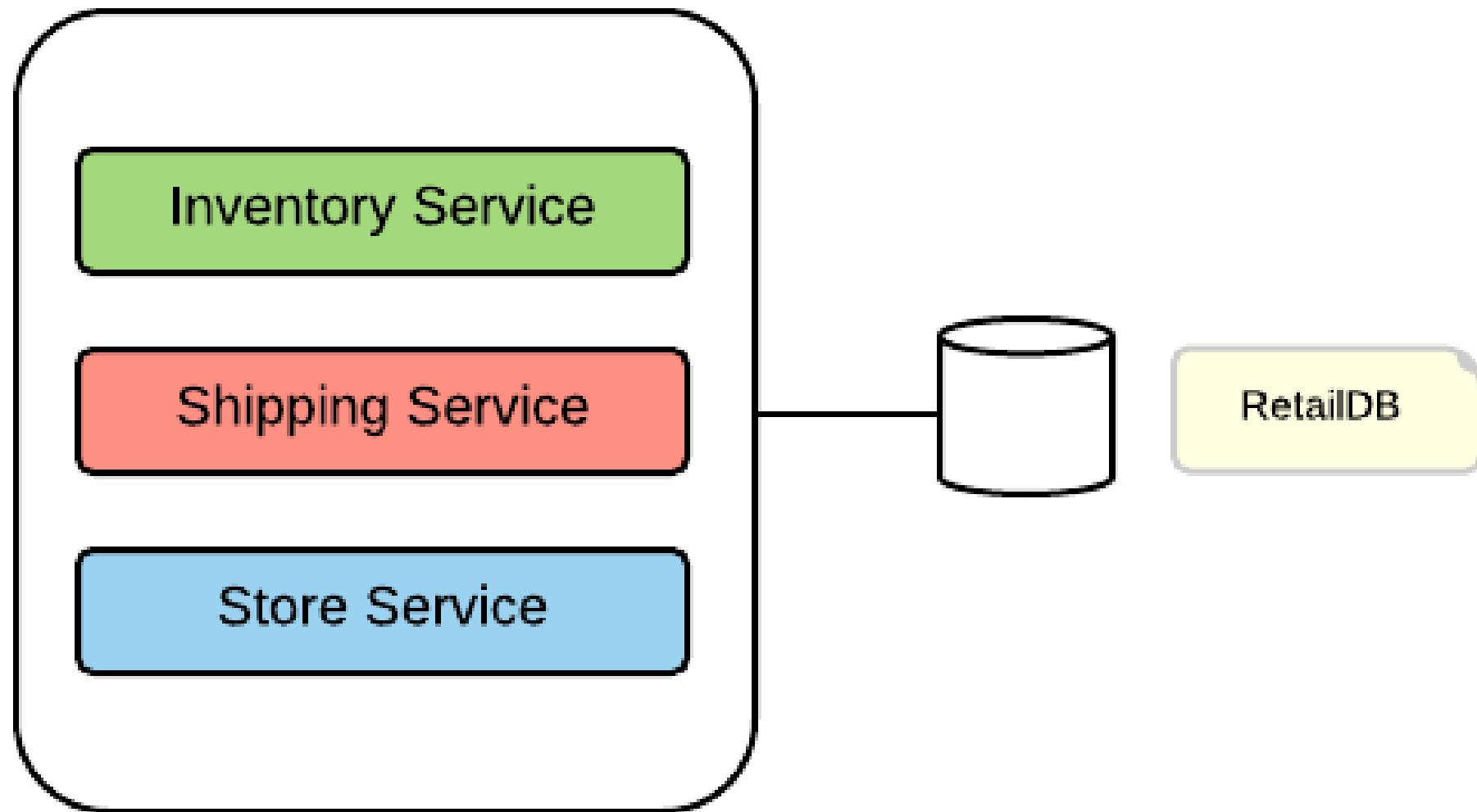
Integrating Microservices (Inter-service Communication)

- Message Broker style



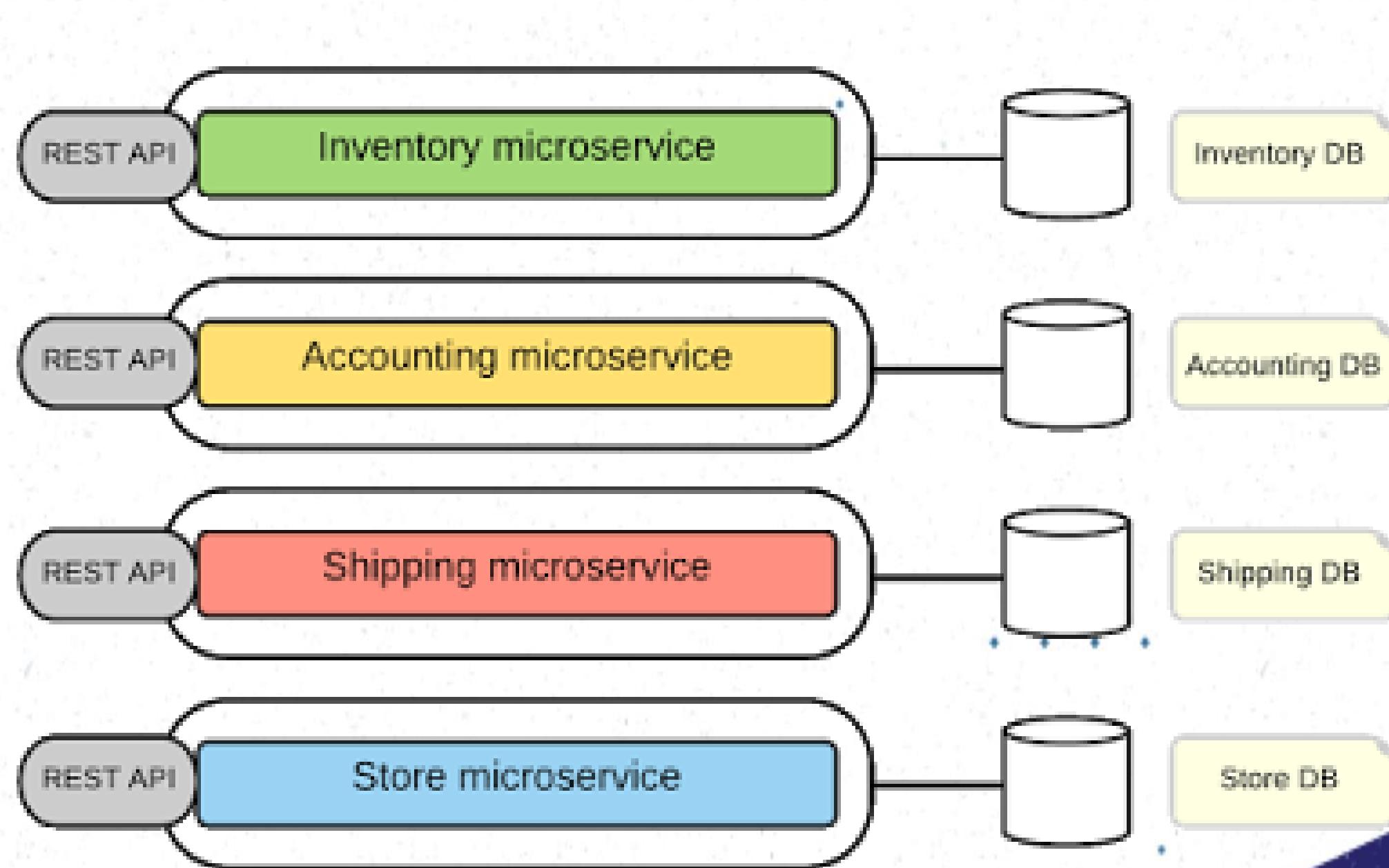
Data Management

- Monolithic applications use a centralized database



Data Management

- Decentralized Data management with Microservices



Data Management

- Decentralized Data management with Microservices
 - Each microservice can have a **private database** to persist the data that requires to implement the business functionality offered from it.
 - A given microservice can **only access the dedicated private database** but not the databases of other microservices.
 - In some business scenarios, you might have to update several database for a single transaction. In such scenarios, **the databases of other microservices** should be updated through its service API only
 - ▪ ▪ ▪
 - ▪ ▪ ▪

Decentralized Governance

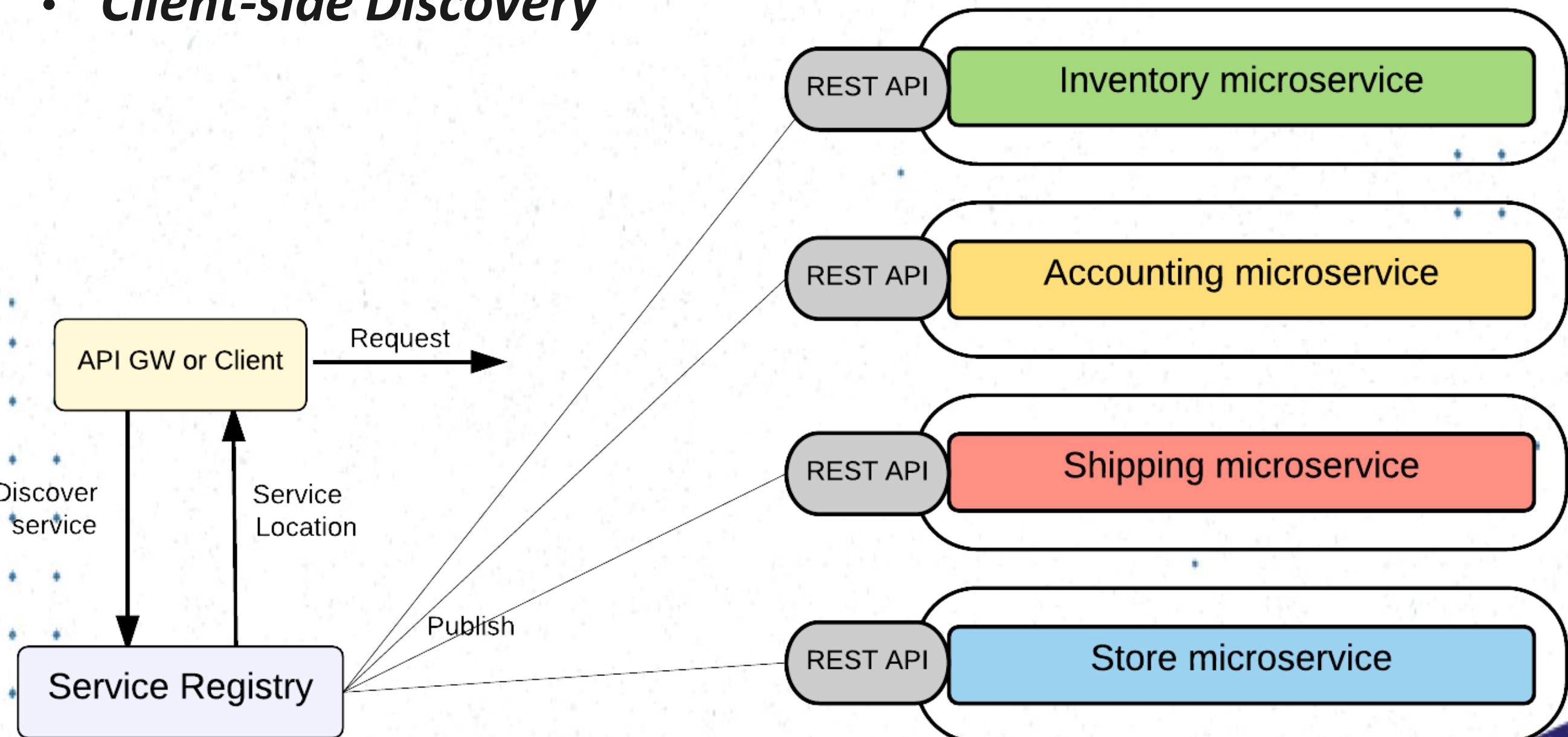
- **Governance** - establishing and enforcing how people and solutions work together to achieve organizational objectives
 - Design and runtime governance.
- In Microservices Architecture:
 - No centralized design-time governance.
 - Make their own decisions about its design and implementation.
 - Foster the sharing of common/reusable services.
 - Run-time governance aspects such as SLAs, throttling, monitoring, common security requirements and service discovery may be implemented at API-GW level.

Service Registry and Service Discovery

- ***Service Registry*** - Holds the microservices instances and their locations
 - ***Service Discovery*** - find the available microservices and their location

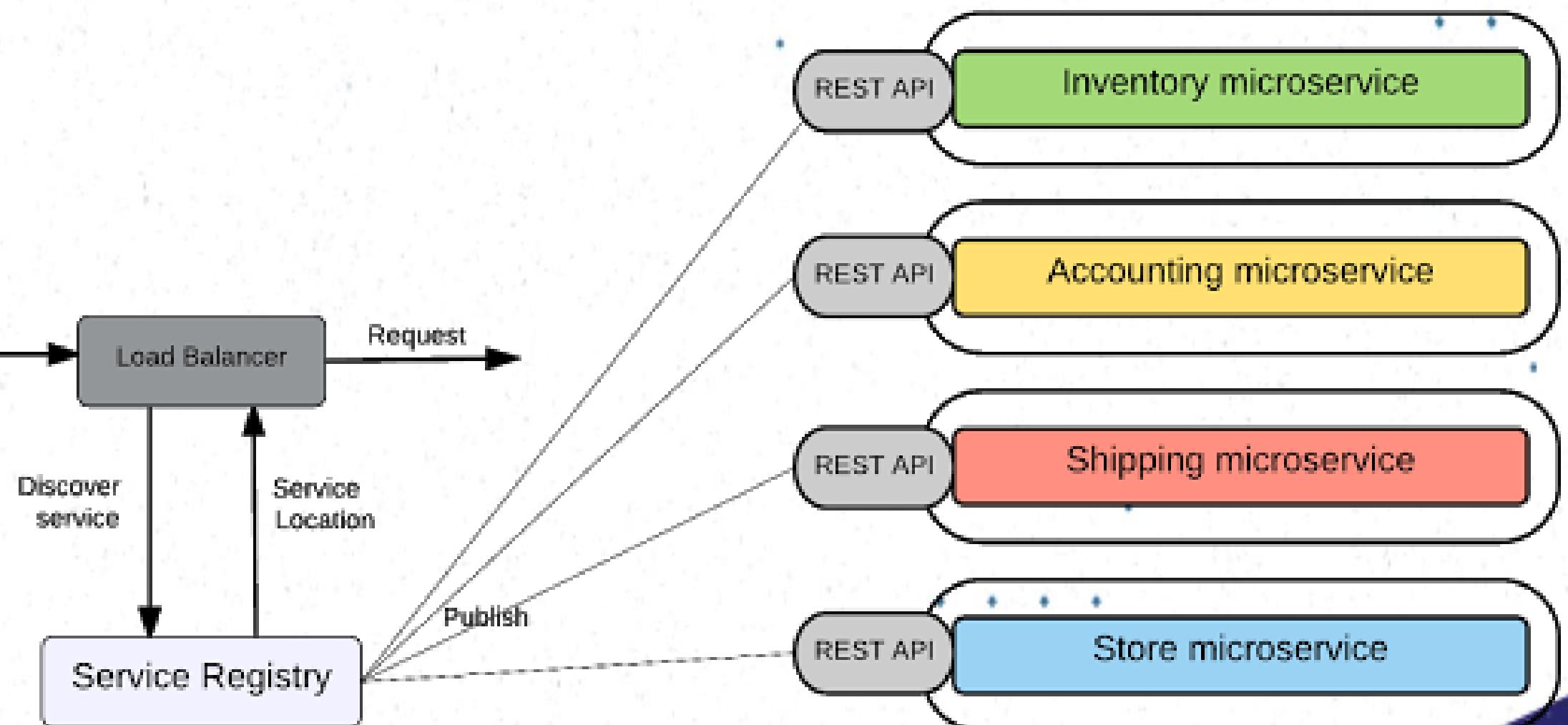
Service Discovery

- *Client-side Discovery*



Service Discovery

- *Server-side Discovery*

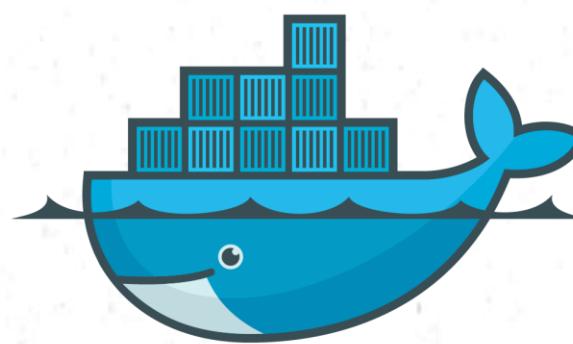


Microservice Deployment

- Ability to deploy/un-deploy independently of other microservices.
 - Must be able to scale at each microservices level.
 - Building and deploying microservices quickly.
 - Failure in one microservice must not affect any of the other services.

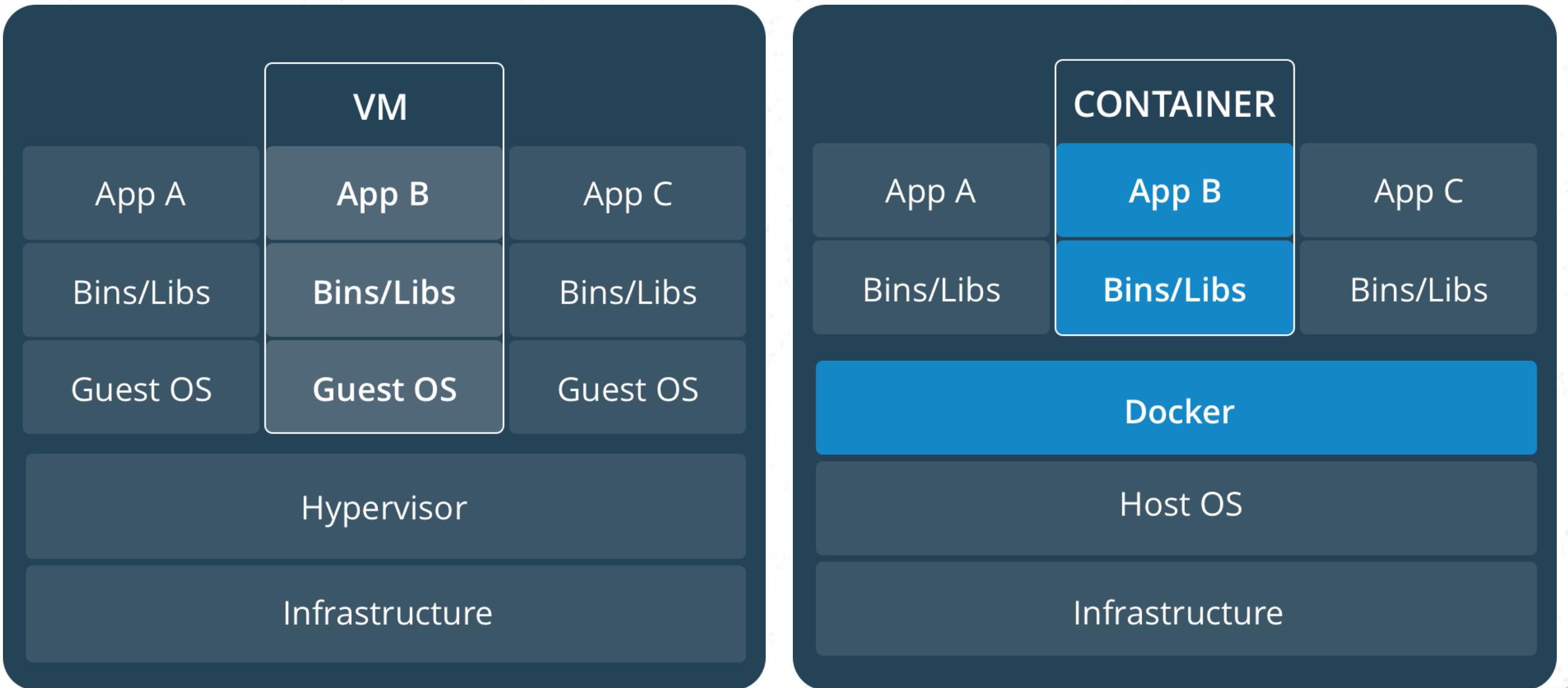
Microservice Deployment

- Docker
 - *Docker is becoming an extremely popular way of packaging and deploying services.*
 - Package the microservice as a (Docker) container image.
 - Deploy each service instance as a container.
 - Scaling is done based on changing the number of container instances.
 - Building, deploying and starting microservice will be much faster as we are using docker containers



docker

Virtual Machines Vs. Docker



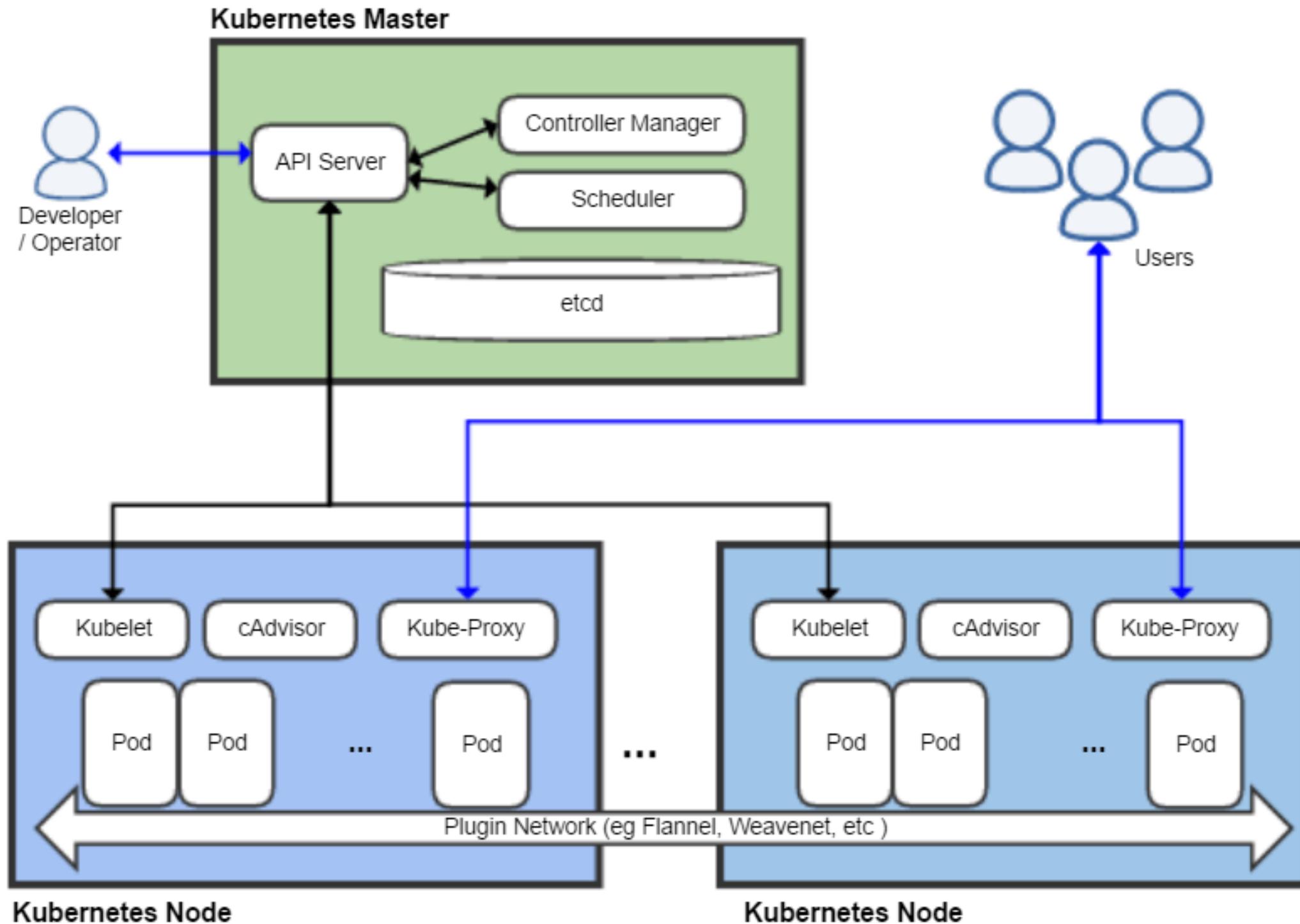
Microservice Deployment

- Kubernetes
 - Extending Docker's capabilities by allowing to manage a cluster of Linux containers as a single system, managing and running Docker containers across multiple hosts, offering co-location of containers, service discovery and replication control.



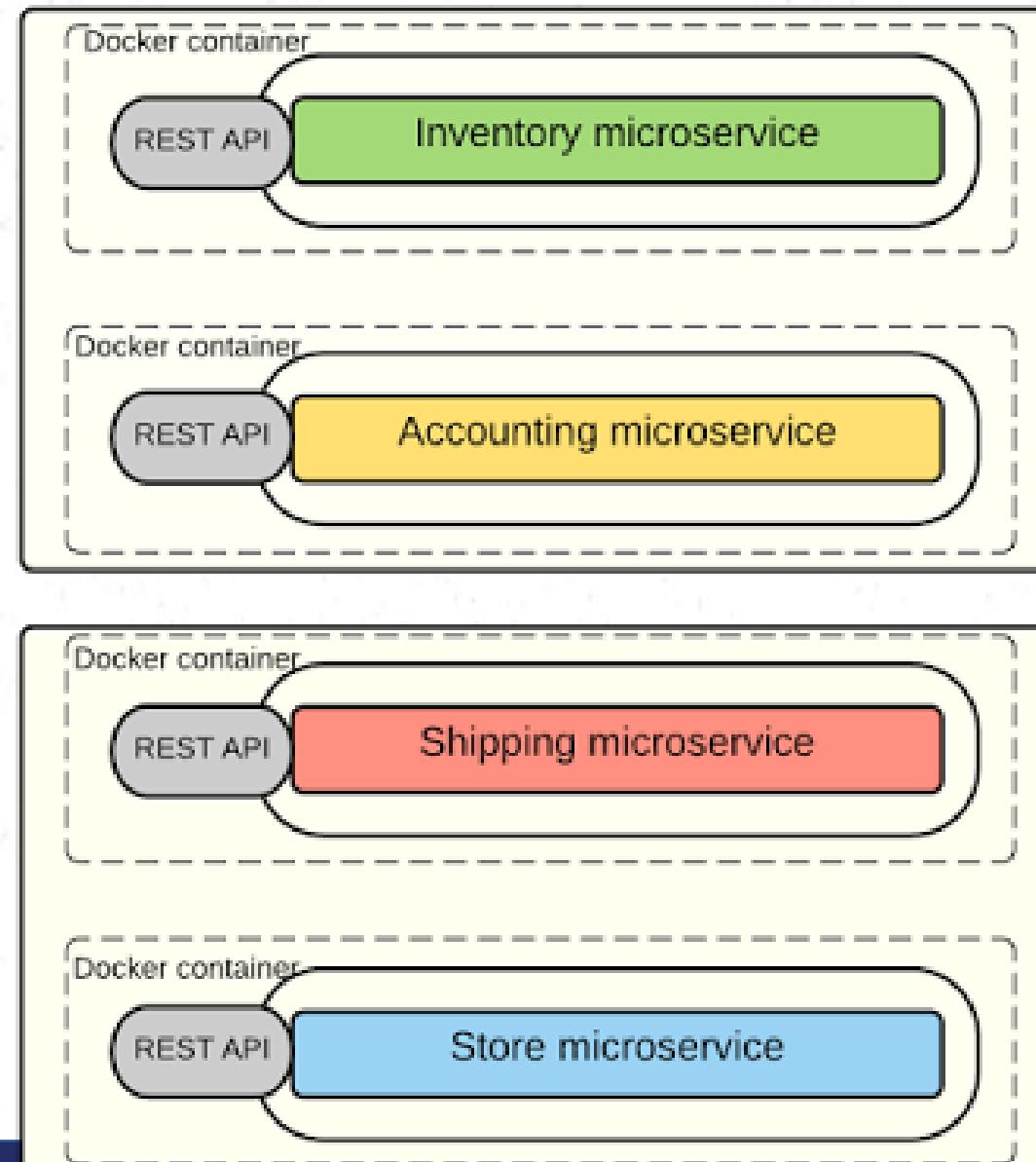
kubernetes

Kubernetes



Microservice Deployment

- Use case : The microservices of Online *Retail software application* with *can be deployed and scaled with Docker and Kubernetes*.



Security with Microservice

- Security in Monolithic applications
 - Its about '*who is the caller*', '*what can the caller do*' and '*how do we propagate that information*'.
 - Often implemented at a common security component which is at the beginning of the request handling chain and that component populates the required information with the use of an underlying user repository
- Security in Microservices
 - a security component implemented at each microservice's level that uses a central user repository/store.
 - Leverage the widely used API-Security standards such as OAuth2 and OpenID Connect

Security with Microservice

- OAuth 2.0
 - The client authenticates with authorization server and get an opaque token which is known as 'Access token'. Access token has zero information about the user/client.
 - It only has a reference to the user information that can only be retrieved by the Authorization server. Hence this is known as a '**by-reference token**' and it is safe to use this token even in the public network/internet.



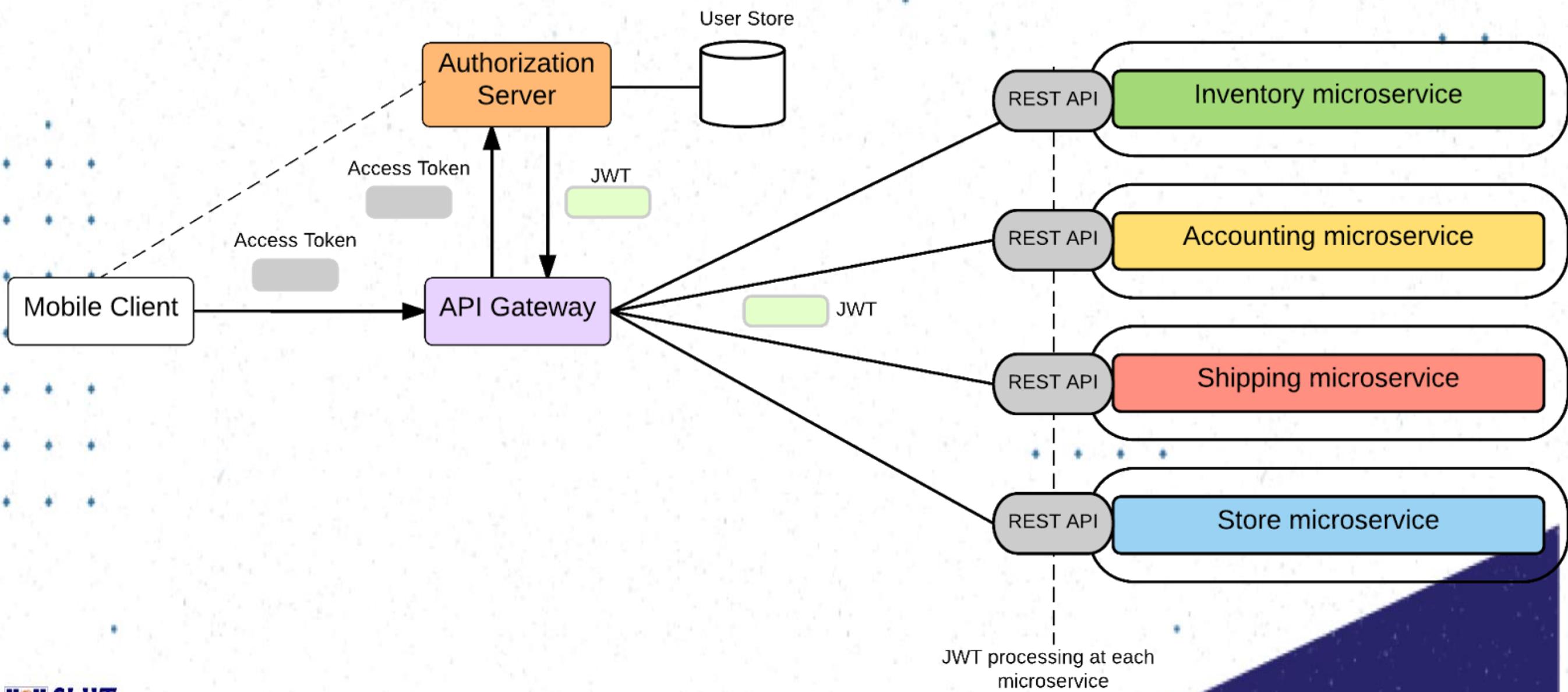
Security with Microservice

- OpenID Connect
 - OpenID Connect behaves similar to OAuth but in addition to the Access token, the authorization server issues an ID token which contains information about the user.
 - Implement with a JWT (JSON Web Token) and that is signed by authorization server. So, this ensures the trust between the authorization server and the client.
 - JWT token is therefore known as a '**By-value token**' as it contains the information of the user and obviously it is not safe to use it outside the internal network.



Security with Microservice

- Microservice security with OAuth2 and OpenID Connect



Transactions

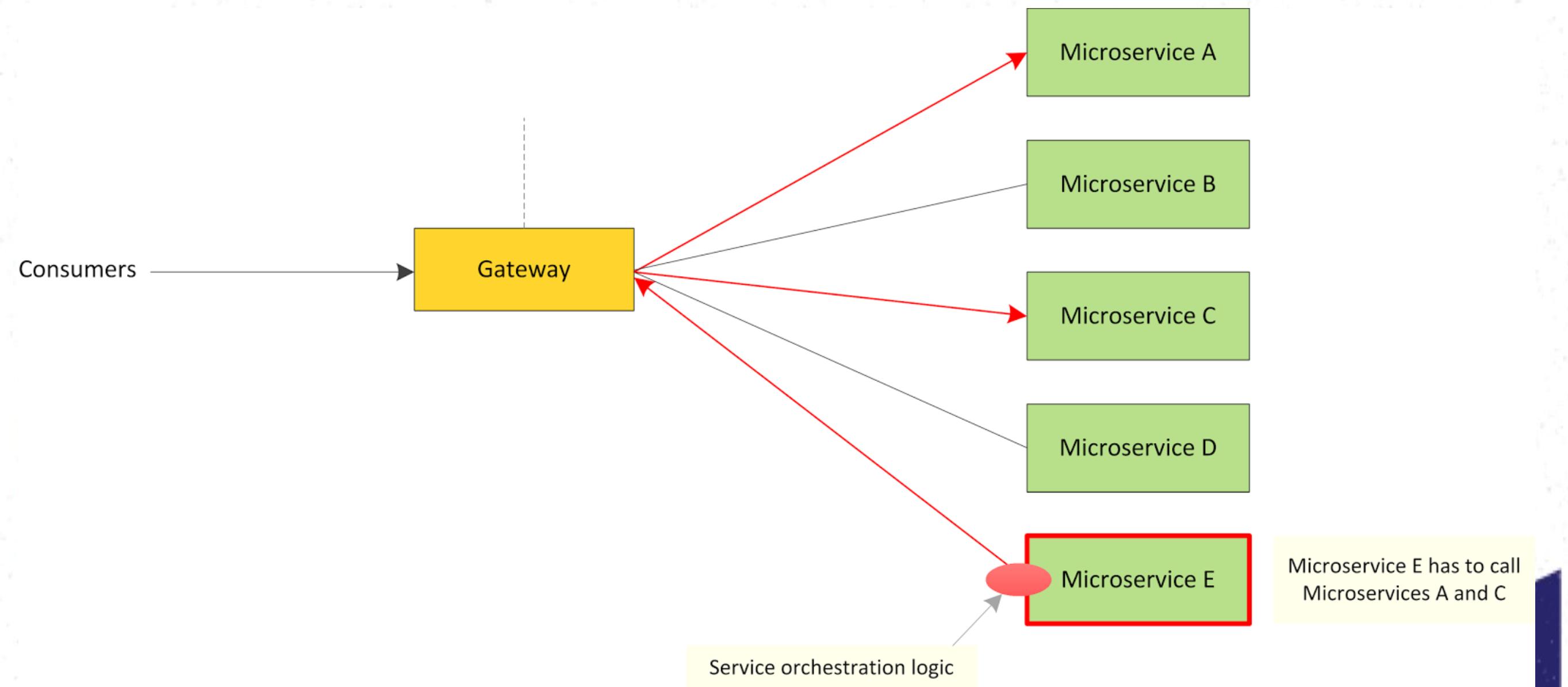
- Supporting distributed transactions across multiple microservices – Too complex.
 - Microservice architecture itself encourages the transaction-less coordination between services.
 - Mandatory transaction requirements can be fulfilled with 'compensating operations'

Design for Failures

- Increases the possibility of having failures at each service level
- Unavailable or unresponsive microservice should not bring the whole system down
- Microservices should be fault tolerant, be able to recover when that is possible and the client has to handle it gracefully.
- Error handling patterns
 - Circuit Breaker
 - Timeout
 - Bulkhead

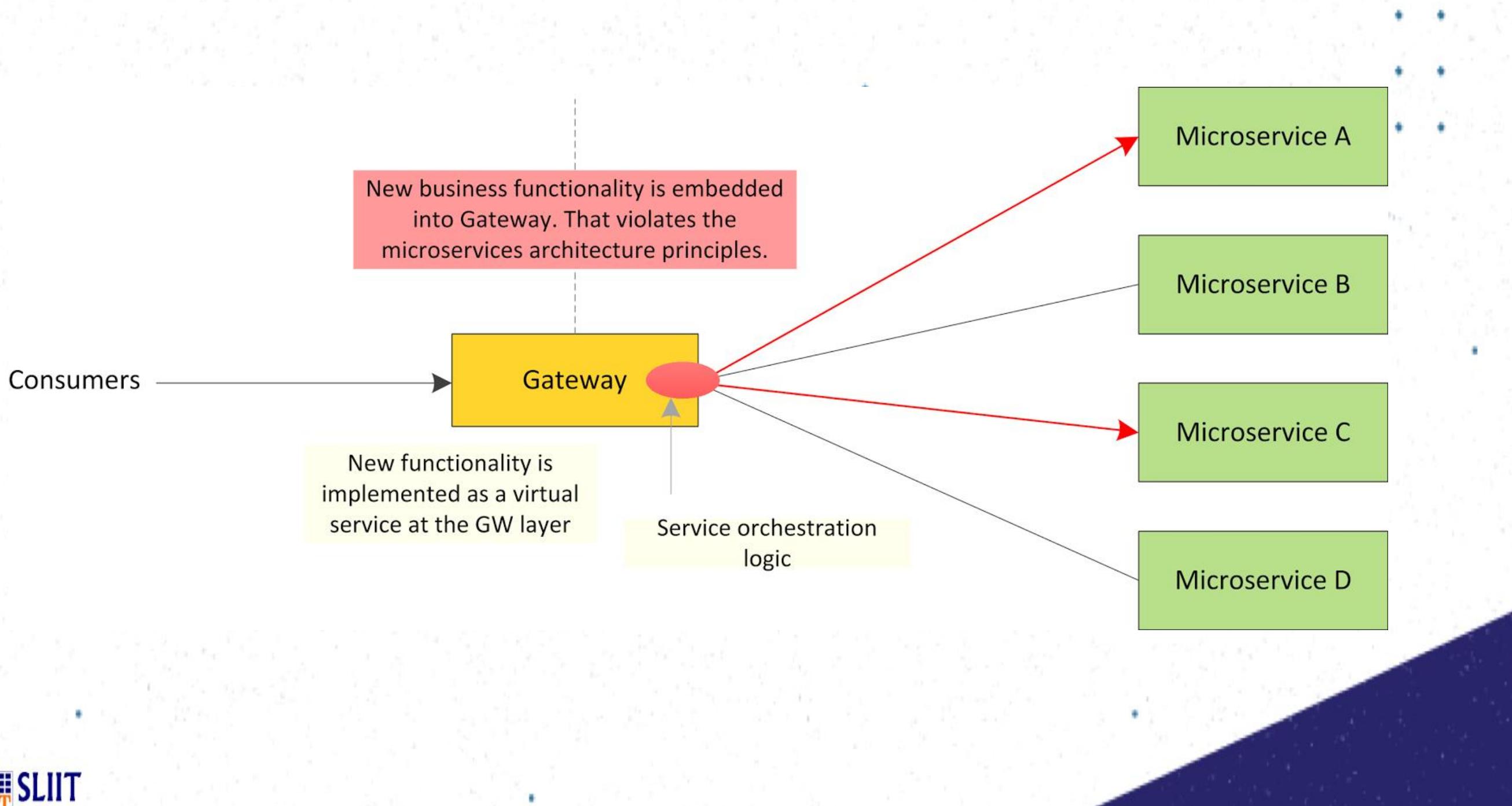
Orchestrating Microservices

- Orchestration at Microservices Layer



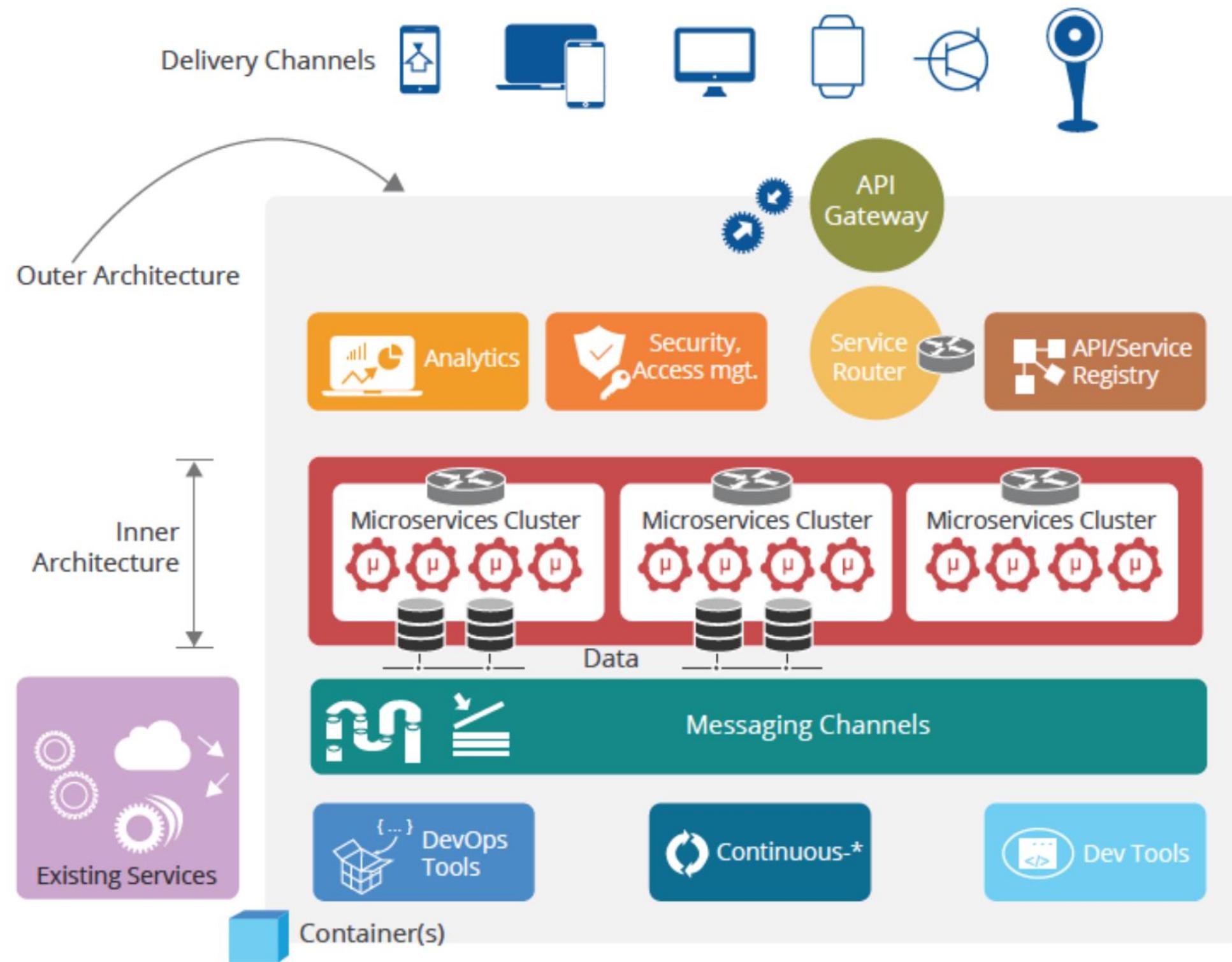
Orchestrating Microservices

- Orchestration at the Gateway Layer



Microservices in Modern Enterprises

- Inner and Outer Architecture



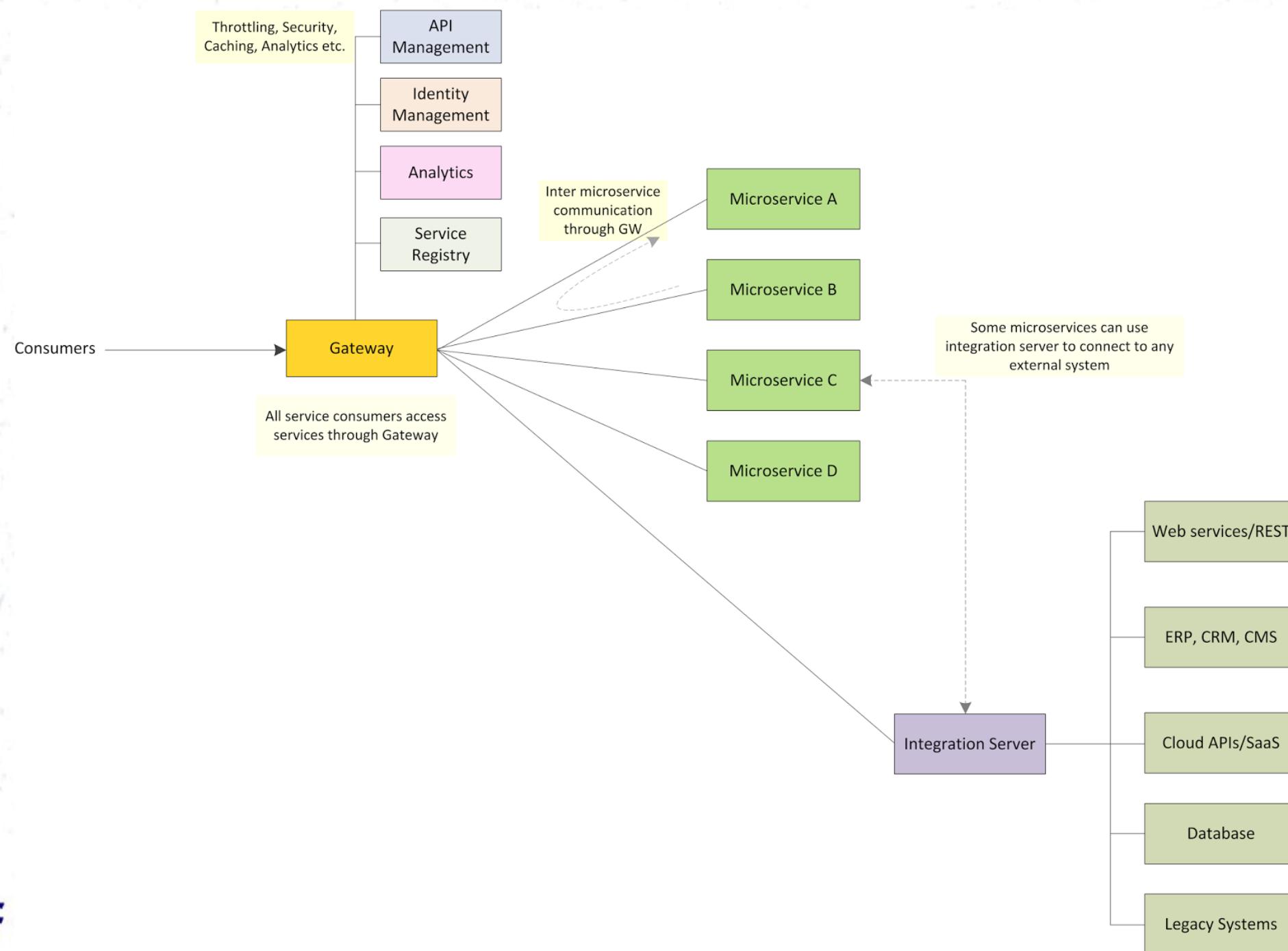
Microservices in Modern Enterprises

- **Inner and Outer Architecture**

- ***Inner Architecture*** : The pure microservices components which is less complex are categorized under 'Inner Architecture'
- ***Outer Architecture*** : This delivers the platform capabilities that are required to build a solution around the microservices that we build.

Microservices in Modern Enterprises

- Modern Enterprise Architecture with Microservices, Enterprise Integration and API Management



Microservices – Conclusion

- Microservices is not a panacea : It won't solve all your enterprise IT needs
- ‘SOA done right’?
- Most enterprises won't be able to convert their entire enterprise IT systems to microservices.
- Enterprise Integration never goes away.
- Microservices are exposed as APIs.
- Interaction between microservices should be support via a lightweight orchestration engine/Gateway or inside another microservice.
-

References

- <http://kasunpanorama.blogspot.com/2015/11/microservices-in-practice.html>
 - <http://kasunpanorama.blogspot.com/2016/02/microservices-enterprise-integration.html>
 - <http://microservices.io/patterns/>
 - <http://martinfowler.com/microservices/>

Fault Tolerance

Topics

- ▶ What is fault tolerance?
- ▶ Why we need fault tolerant systems?
- ▶ Existing fault tolerant techniques/models
(and where they are applied)?
- ▶ Challenges in developing fault tolerant systems.
- ▶ Future of fault tolerance.

Software Services

- ▶ People use software services and applications everyday.
- ▶ They will continue to use them only if these services/applications are

DEPENDABLE.

Dependability

- ▶ **Specified Service** : How the behavior of the service should be
- ▶ **Derived Service** : Actual behavior of the service
- ▶ If the dependability goes down, Derived service will start to deviate from the Specified Service

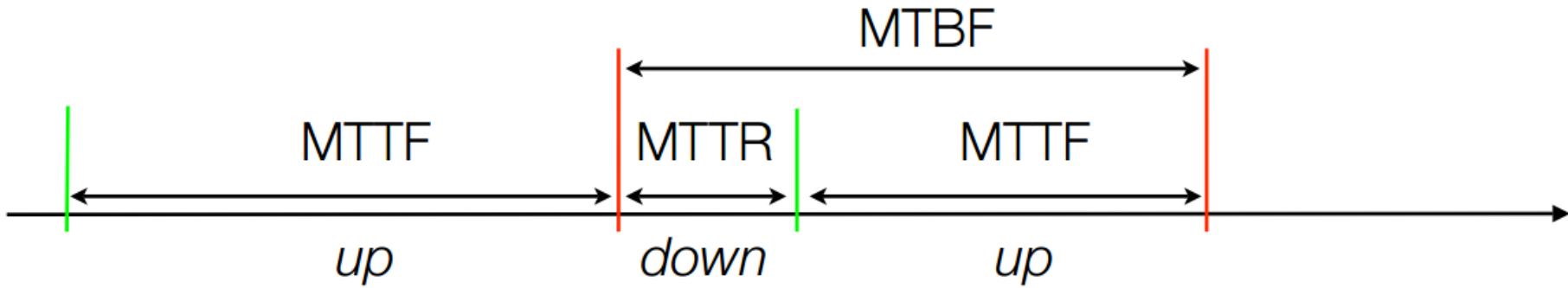
Dependability

- ▶ *Availability* – the system is ready to be used immediately.
- ▶ *Reliability* – the system can run continuously without failure.
- ▶ *Safety* – if a system fails, nothing catastrophic will happen.
- ▶ *Maintainability* – when a system fails, it can be repaired easily and quickly (and, sometimes, without its users noticing the failure).

Reliability and Availability

- ▶ **Mean time to failure (MTTF)** – Average time it takes for the system to fail
- ▶ **Mean time to recover (MTTR)** – Average time it takes to recover
- ▶ **Mean time between failures (MTBF)** – Average time between failures

Reliability and Availability



$$A = \frac{\text{Uptime}}{\text{Uptime} + \text{Downtime}} = \frac{MTTF}{MTTF + MTTR}$$

Reliability and Availability

Availability	Downtime per year	Downtime per week
90.0 % (1 nine)	36.5 days	16.8 hours
99.0 % (2 nines)	3.65 days	1.68 hours
99.9 % (3 nines)	8.76 hours	10.1 min
99.99 % (4 nines)	52.6 min	1.01 min
99.999 % (5 nines)	5.26 min	6.05 s
99.9999 % (6 nines)	31.5 s	0.605 s
99.99999 % (7 nines)	0.3 s	6 ms

Faults, Errors and Failures

- ▶ A fault is the cause of an error. It is associated with a defect.
- ▶ An error is part of the system state that may cause a subsequent failure: a failure occurs when an error reaches the service interface and alters the service.
- ▶ A system failure is an event that occurs when the derived service deviates from specified service.
- ▶ A fault originally causes an error within the state of one (or more) components, but system failure will not occur as long as the error does not reach the service interface of the system.

What is fault tolerance?

- ▶ The approach of fault-tolerance expect faults to be present during system operation, but employs design techniques which insure the continued correct execution of the computing process

Why do we need fault tolerant systems?

- ▶ Average costs per hour of downtime (Gartner 1998)
 - Brokerage operations in finance: \$6.5 million
 - Credit card authorization: \$2.6 million
 - Home catalog sales: \$90.000
 - Airline reservation: \$89.500
- ▶ 22-hour service outage of eBay in June 1999
 - Interruption of around 2.3 million auctions
 - 9.2% stock value drop

Why we need fault tolerant systems?



For the Fortune 1000, the average total cost of unplanned application downtime per year is

\$1.25 billion - \$2.5 billion



The average cost of a critical application failure per hour is

\$500,000 - \$1 million

The average hourly cost of an infrastructure failure is

\$100,000 per hour



Stats from
2015

Fault Classification

- ▶ By cause
 - Hardware Faults
 - Design Faults
 - Operation Faults
 - Environment Faults

- ▶ By duration
 - Permanent Faults
 - Intermittent Faults
 - Transient Faults

Failure Classification

Type of failure	Description
Crash failure	A server halts, but is working correctly until it halts
Omission failure <i>Receive omission</i> <i>Send omission</i>	A server fails to respond to incoming requests A server fails to receive incoming messages A server fails to send messages
Timing failure	A server's response lies outside the specified time interval
Response failure <i>Value failure</i> <i>State transition failure</i>	The server's response is incorrect The value of the response is wrong The server deviates from the correct flow of control
Arbitrary failure (Byzantine failure)	A server may produce arbitrary responses at arbitrary times

Fault Tolerance Strategies

- ▶ Fault tolerance in computer system is achieved through redundancy in hardware, software, information, and/or time.
Such redundancy can be implemented in static, dynamic, or hybrid configurations.
- ▶ Fault tolerance can be achieved by many techniques:
 - **Fault masking** is any process that prevents faults in a system from introducing errors. Example: Error correcting memories and majority voting.
 - **Reconfiguration** is the process of eliminating faulty component from a system and restoring the system to some operational state.

Reconfiguration Approach

- ▶ **Fault detection** is the process of recognizing that a fault has occurred. Fault detection is often required before any recovery procedure can be initiated.
- ▶ **Fault location** is the process of determining where a fault has occurred so that an appropriate recovery can be initiated.
- ▶ **Fault containment** is the process of isolating a fault and preventing the effects of that fault from propagating throughout the system.
- ▶ **Fault recovery** is the process of regaining operational status via reconfiguration even in the presence of faults.

The Concept of Redundancy

- ▶ *Redundancy* is simply the addition of information, resources, or time beyond what is needed for normal system operation.
- ▶ **Software redundancy** is the addition of extra software, beyond what is needed to perform a given function, to detect and possibly tolerate faults.
- ▶ **Hardware redundancy** is the addition of extra hardware, usually for the purpose either detecting or tolerating faults.
- ▶ **Information redundancy** is the addition of extra information beyond that required to implement a given function; for example, error detection codes.

The Concept of Redundancy

- ▶ Time redundancy uses additional time to perform the functions of a system such that fault detection and often fault tolerance can be achieved. *Transient faults* are tolerated by this.
- ▶ The use of redundancy can provide additional capabilities within a system. But, redundancy can have very important impact on a system's performance, size, weight and power consumption.

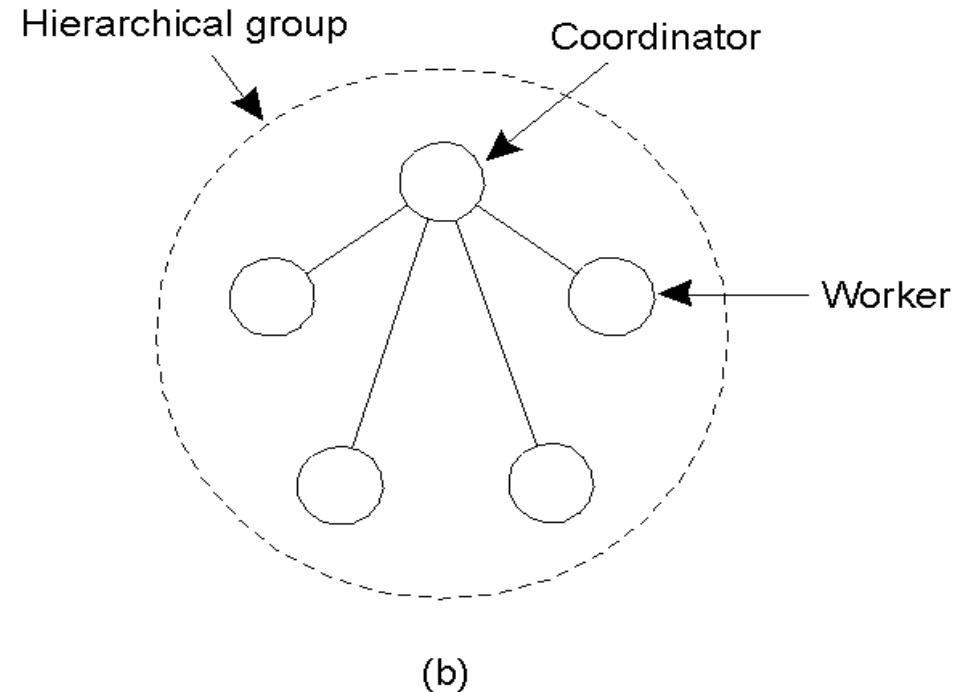
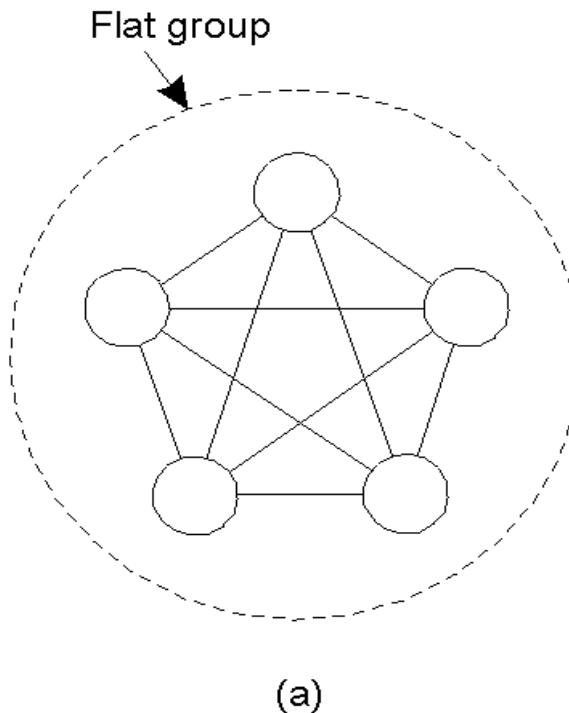


Software Redundancy

Process Resilience

- ▶ Mask process failures by replication
- ▶ Organize processes into groups, a message sent to a group is delivered to all members
- ▶ If a member fails, another should fill in

Flat Groups versus Hierarchical Groups



- a) Communication in a flat group.
- b) Communication in a simple hierarchical group

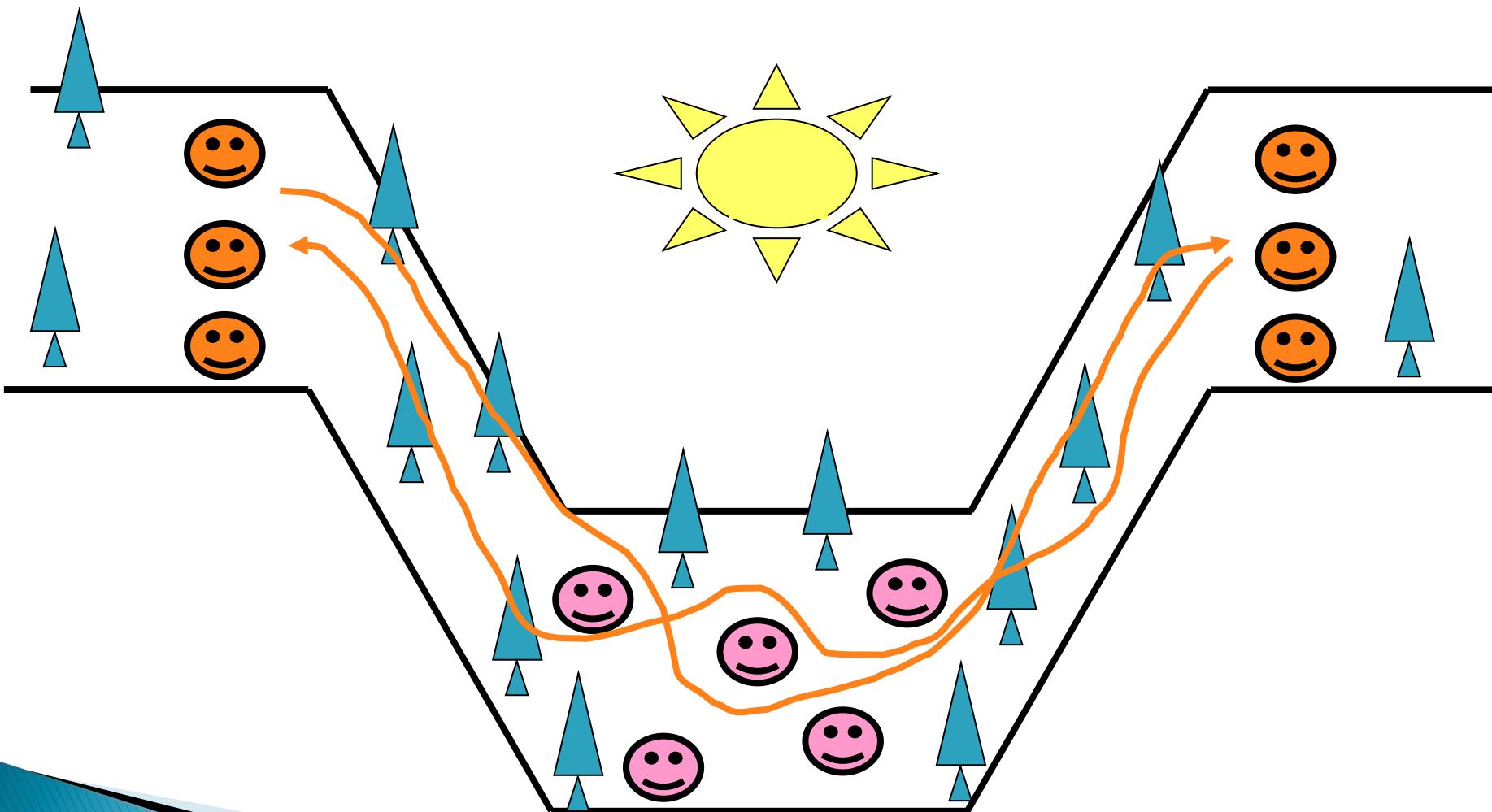
Process Replication

- ▶ Replicate a process and group replicas in one group
- ▶ A system is k fault-tolerant if it can survive and function even if it has k faulty processes
 - For crash failures (a faulty process halts, but is working correctly until it halts)
 - $k+1$ replicas
 - For Byzantine failures (a faulty process may produce arbitrary responses at arbitrary times)
 - $2k+1$ replicas

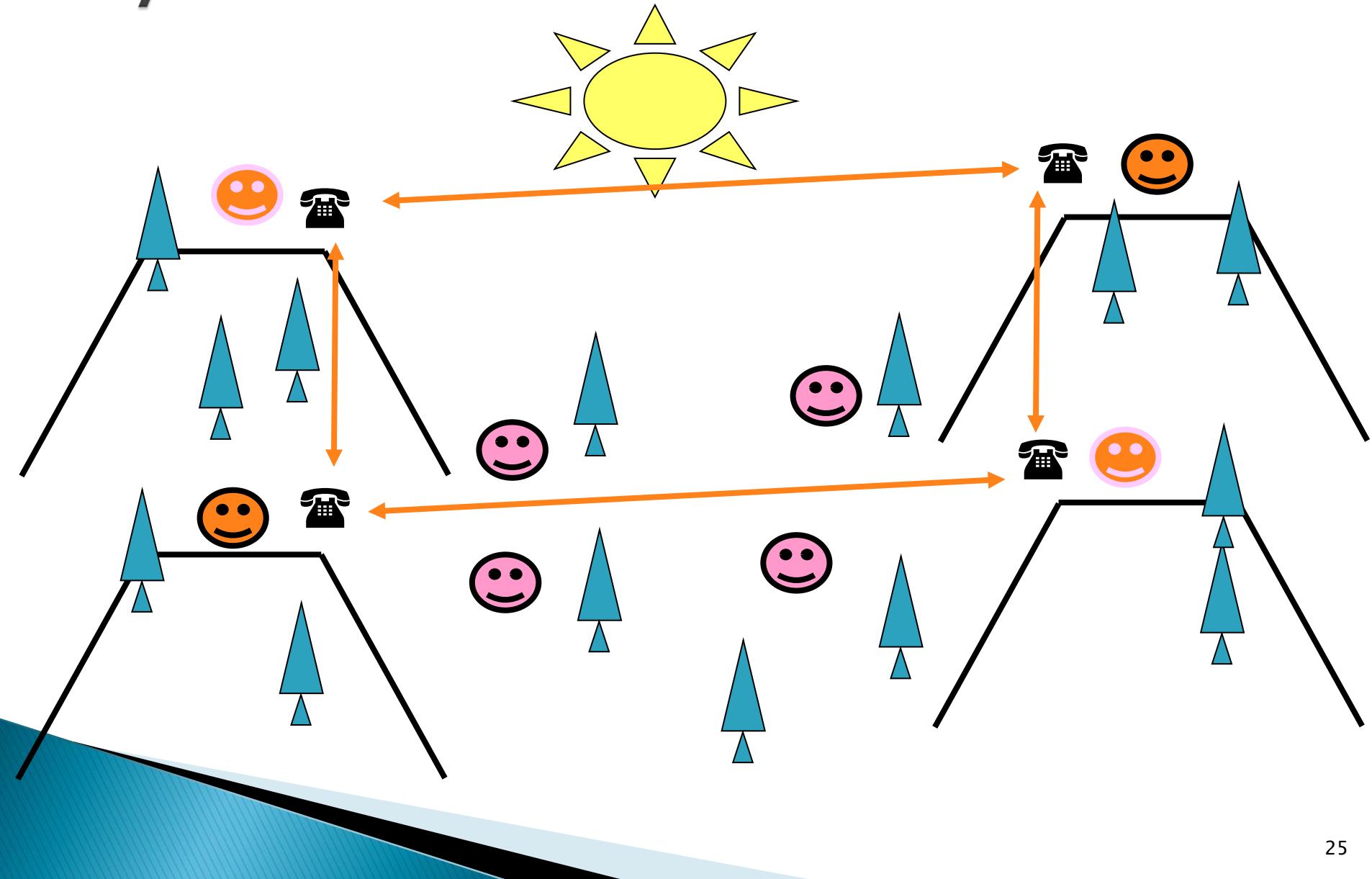
Agreement

- ▶ Need agreement in DS:
 - Leader, commit, synchronize
- ▶ **Distributed Agreement algorithm:** all non-faulty processes achieve consensus in a finite number of steps
- ▶ Perfect processes, faulty channels: two-army
- ▶ Faulty processes, perfect channels: Byzantine generals

Two-Army Problem



Byzantine Generals Problem



Distributed COMMIT

- ▶ General Goal:
 - *We want an operation to be performed by all group members, or none at all.*
- ▶ There are three types of “commit protocol”: single-phase, two-phase and three-phase commit.

Commit Protocols

► One-Phase Commit Protocol:

- An elected co-ordinator tells all the other processes to perform the operation in question.
- But, what if a process cannot perform the operation? There's no way to tell the coordinator! Whoops ...

► The solutions:

- The *Two-Phase* and *Three-Phase Commit Protocols*.

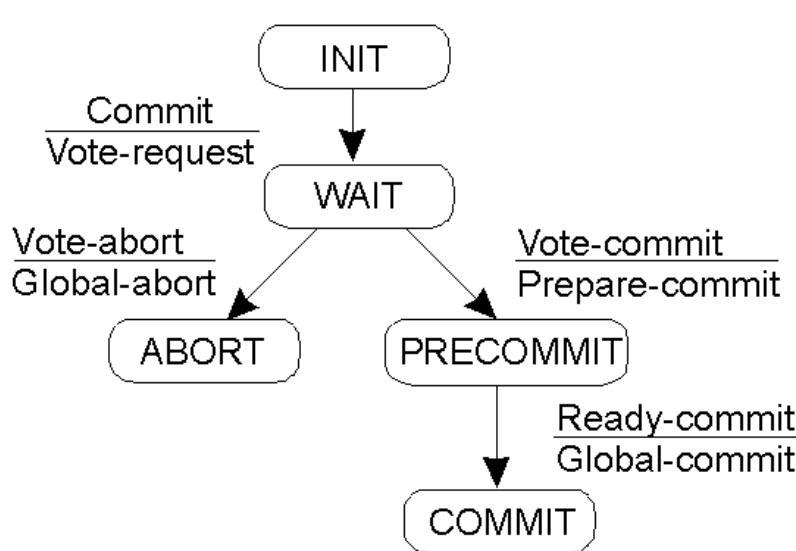
The Two-Phase Commit Protocol

- ▶ First developed in 1978!!!
- ▶ *Summarized: GET READY, OK, GO AHEAD.*
 1. The coordinator sends a ***VOTE_REQUEST*** message to all group members.
 2. The group member returns ***VOTE_COMMIT*** if it can commit locally, otherwise ***VOTE_ABORT***.
 3. All votes are collected by the coordinator. A ***GLOBAL_COMMIT*** is sent if all the group members voted to commit. If one group member voted to abort, a ***GLOBAL_ABORT*** is sent.
 4. The group members then **COMMIT** or **ABORT** based on the last message received from the coordinator.

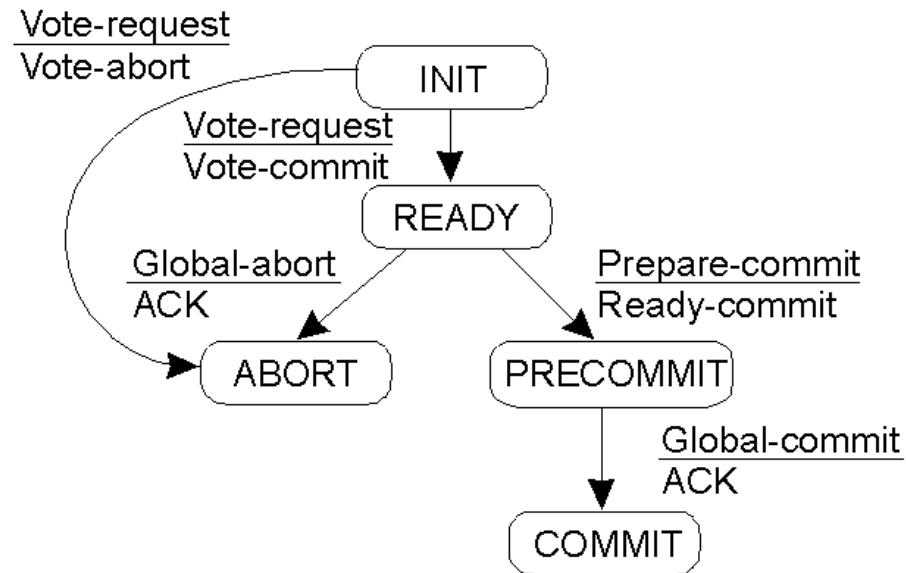
Big Problem with Two-Phase Commit

- ▶ It can lead to both the coordinator and the group members **blocking**, which may lead to the dreaded *deadlock*.
- ▶ If the coordinator crashes, the group members may not be able to *reach a final decision*, and they may, therefore, block until the coordinator *recovers* ...
- ▶ Two-Phase Commit is known as a **blocking-commit protocol** for this reason.
- ▶ The solution? *The Three-Phase Commit Protocol*.

Three-Phase Commit



(a)



(b)

- a) Finite state machine for the coordinator.
- b) Finite state machine for a group member.
- c) **Main point:** although 3PC is generally regarded as *better* than 2PC, it is *not applied often in practice*, as the conditions under which 2PC blocks rarely occur.

Software Redundancy – to Detect Hardware Faults

- ▶ **Consistency checks** use a priori knowledge about the characteristics of the information to verify the correctness of that information.
Example: Range checks, overflow and underflow checks.
- ▶ **Capability checks** are performed to verify that a system possesses the capability expected.
Examples: Memory test – a processor can simply write specific patterns to certain memory locations and read those locations to verify that the data was stored and retrieved properly.

Software Redundancy – to Detect Hardware Faults

- ▶ **ALU tests:** Periodically, a processor can execute specific instructions on specific data and compare the results to known results stored in ROM.
- ▶ **Testing of communication** among processors, in a multiprocessor, is achieved by periodically sending specific messages from one processor to another or writing into a specific location of a shared memory.

Software Redundancy – to Detect Hardware Faults.

- All modern day microprocessors use instruction retry
- Any transient fault that causes an exception such as parity violation is retried
- Very cost effective and is now a standard technique

Software Redundancy – to Detect Software Faults

- ▶ There are two popular approaches: **N-Version Programming** (NVP) and **Recovery Blocks** (RB).
- ▶ NVP is a forward recovery scheme – it masks faults.
- ▶ RB is a backward error recovery scheme.
- ▶ In NVP, multiple versions of the same task is executed concurrently, whereas in RB scheme, the versions of a task are executed serially.
- ▶ NVP relies on *voting*.
- ▶ RB relies on *acceptance test*.

Single Version Fault Tolerance: Software Rejuvenation

- **Example:** Rebooting a PC
- As a process executes
 - it acquires memory and file-locks without properly releasing them
 - memory space tends to become increasingly fragmented
- The process can become faulty and stop executing
- To head this off, proactively halt the process, clean up its internal state, and then restart it
- Rejuvenation can be **time-based** or **prediction-based**
- Time-Based Rejuvenation – periodically
- Rejuvenation period – balance benefits against cost



HARDWARE REDUNDANCY

Hardware Redundancy

- ▶ **Static techniques** use the concept of fault masking. These techniques are designed to achieve fault tolerance without requiring any action on the part of the system. Relies on voting mechanisms.
(also called passive redundancy or fault-masking)
- ▶ **Dynamic techniques** achieve fault tolerance by detecting the existence of faults and performing some action to remove the faulty hardware from the system. That is, active techniques use fault detection, fault location, and fault recovery in an attempt to achieve fault tolerance.
(also called active redundancy)

Hardware Redundancy (Cont'd)

- ▶ **Hybrid techniques** combine the attractive features of both the passive and active approaches.
 - Fault masking is used in hybrid systems to prevent erroneous results from being generated.
 - Fault detection, location, and recovery are also used to improve fault tolerance by removing faulty hardware and replacing it with spares.

Future of Fault Tolerance

- ▶ Apache Zookeeper
 - ZooKeeper Atomic Broadcasts
- ▶ ETCD
- ▶ Consul
 - Gossip Protocol
- ▶ Doozer

Lecture 13 – Clock Synchronization in Distributed Systems

Concept of Time in Distributed Systems

Time in DS

- Time is an interesting and Important issue in DS
- Algorithms that depend upon clock synchronization have been developed for several problems.
- Due to loose synchrony, the notion of physical time is problematic in DS
 - There is no absolute physical “global time” in DS

Clocks

- How time is really measured?
 - Earlier: Solar day, solar second, mean solar second
- Solar day: time between two consecutive transits of the sun
- Solar second: 1/86400 of a solar day
- Mean solar day: average length of a solar day
- Problem: solar day gets longer because of slowdown of earth rotation due to friction (300 million years ago there were 400 days per year)

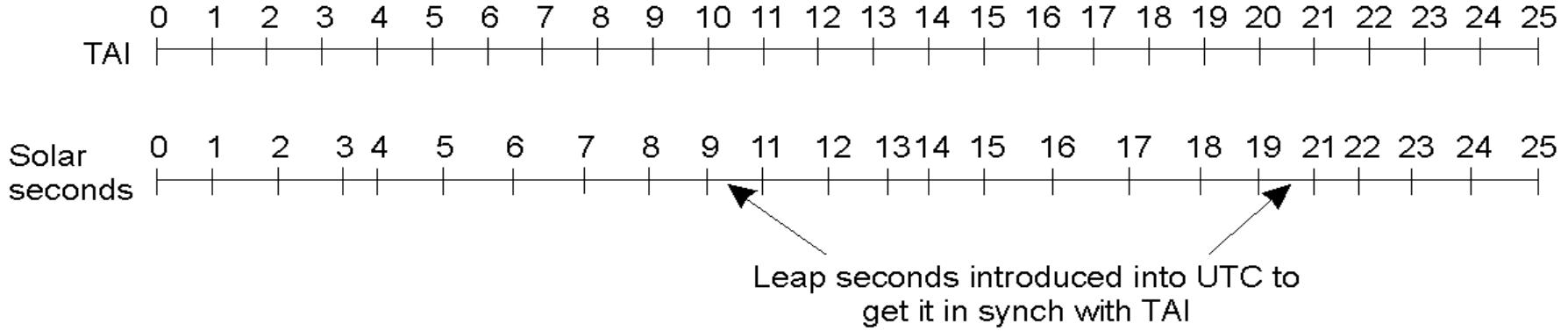
Physical Clocks

- International Atomic Time (TAI): number of ticks of Cesium 133 atom since 1/1/58 (atomic second)
- Atom clock: one second defined as (since 1967) 9,192,631,770 transitions of the atom Cesium 133

Physical Clocks

- Because of slowdown of earth, leap seconds have to be introduced
- Correction of TAI is called Universal Coordinated Time (UTC): 30 leap seconds introduced so far
- Network Time Protocol (NTP) can synchronize globally with an accuracy of up to 50 msec

Physical Clocks



- TAI seconds are of constant length, unlike solar seconds. Leap seconds are introduced when necessary to keep in phase with the sun.

Why synchronization?

You want to catch the 5pm train at the Fort railway station, but your watch is off by 15 minutes

What if your watch is Late by 15 minutes?

What if your watch is Fast by 15 minutes?

Synchronization is required for

Correctness

Fairness

Why synchronization?

- make recompiles if *foo.c* is newer than *foo.o*
- Scenario
 - make on machine *A* to build *foo.o*
 - Test on machine *B*; find and fix a bug in *foo.c*
 - Re-run make on machine *B*
 - *Nothing happens!*
- Why?

Why synchronization?

Airline reservation system

Server A receives a client request to purchase last ticket on flight ABC 123.

Server A timestamps purchase using local clock **9h:15m:32.45s**, and logs it. Replies ok to client.

That was the last seat. Server A sends message to Server B saying “flight full.”

B enters “Flight ABC 123 full” + local clock value (which reads **9h:10m:10.11s**) into its log.

Server C queries A's and B's logs. Is confused that a client purchased a ticket after the flight became full.

May execute incorrect or unfair actions.

Basics – Processes and Events

An Asynchronous Distributed System (DS) consists of a number of *processes*.

Each process has a *state* (values of variables).

Each process takes *actions* to change its state, which may be an *instruction* or a communication action (*send*, *receive*).

An *event* is the occurrence of an action.

Each process has a local clock – events *within* a process can be assigned *timestamps*, and thus ordered linearly.

But – in a DS, we also need to know the time order of events across different processes.

- ⌚ Clocks across processes are not synchronized in an asynchronous DS
 - unlike in a multiprocessor/multicore system

Physical Clocks & Synchronization

In a DS, each process has its own clock.

Clock Skew versus Drift

- Clock **Skew** = Relative Difference in clock *values* of two processes
- Clock **Drift** = Relative Difference in clock *frequencies (rates)* of two processes

A non-zero clock drift causes skew to increase (eventually).

Maximum Drift Rate (**MDR**) of a clock

Absolute MDR is defined relative to Coordinated Universal Time (UTC).

UTC is the “correct” time at any point of time.

- MDR of a process depends on the environment.

Synchronizing Physical Clocks

$C_i(t)$: the reading of the software clock at process i when the real time is t .

External synchronization: For a synchronization bound $D > 0$, and for source S of UTC time,

$$|S(t) - C_i(t)| < D, \quad \text{for } i=1,2,\dots,N \text{ and for all real times } t.$$

Clocks C_i are externally accurate to within the bound D .

In external synchronization, clock is synchronized with an authoritative external source of time

Synchronizing Physical Clocks

Internal synchronization: For a synchronization bound $D > 0$,

$$|C_i(t) - C_j(t)| < D$$

for $i, j = 1, 2, \dots, N$ and for all real times t .

Clocks C_i are internally accurate within the bound D .

In internal synchronization clocks are synchronized with one another with a known degree of accuracy

External synchronization with $D \Rightarrow$ Internal synchronization with $2D$

Internal synchronization with $D \Rightarrow$ External synchronization with ??

Clock synchronization

- UTC signals are synchronized and broadcast regularly from land-based radio stations and satellites covering many parts of the world
 - E.g. in the US the radio station WWV broadcasts time signals on several short-wave frequencies
 - Satellite sources include Geo-stationary Operational Environmental Satellites (GOES) and the GPS

Clock synchronization

- Radio waves travel at near the speed of light. The propagation delay can be accounted for if the exact speed and the distance from the source are known
- Unfortunately, the propagation speed varies with atmospheric conditions – leading to inaccuracy
- Accuracy of a received signal is a function of both the accuracy of the source and its distance from the source through the atmosphere

Clock Synchronization Algorithms

- The constant ρ is specified by the manufacturer and is known as the maximum drift rate.
- If two clocks are drifting from the Universal Coordinated Time (UTC) in opposite direction, at a time Δt after they are synchronized, they maybe as much as $2 * \rho * \Delta t$ apart.
- If the operating system designer want to guarantee that no two clocks ever differ by more than δ , clocks must be synchronized at least every $\delta / 2 \rho$ seconds.

Clock synchronization

- Remember the definition of synchronous distributed system?
 - Known bounds for message delay, clock drift rate and execution time.
 - Clock synchronization is easy in this case
- In practice most DS are asynchronous.
 - Cristian's Algorithm
 - The Berkeley Algorithm

Clock synchronization in a synchronous system

- Consider internal synch between two procs in a synch DS
- P sends time t on its local clock to Q in a msg m
- In principle, Q could set its clock to the time $t + T_{trans}$, where T_{trans} is the time taken to transmit m between them
- The two processes would then agree (internal synch)

Clock synchronization in a synchronous system

- Unfortunately, T_{trans} is subject to variation and is unknown
 - All processes are competing for resources with P and Q and other messages are competing with m for the network
 - But there is always a minimum transmission time min that would be obtained if no other processes executed and no other network traffic existed
 - min can be measured or conservatively estimated

Clock synchronization in a synchronous system

- In synch system, by definition, there is also an upper bound \max on the time taken to transmit any message
- Let the uncertainty in the msg transmission time be u , so that $u = (\max - \min)$
 - If Q sets its clock to be $(t + \min)$, then clock skew may be as much as u (since the message may in fact have taken time \max to arrive).
 - If Q sets it to $(t + \max)$, the skew may again be as large as u .
 - If, however, Q sets it clock to $(t + (\max - \min)/2)$, then the skew is at most $u/2$.
 - In general, for a synch system, the optimum bound that can be achieved on clock skew when synchronizing N clocks is $u(1-1/N)$

Cristian's clock synchronization algorithm

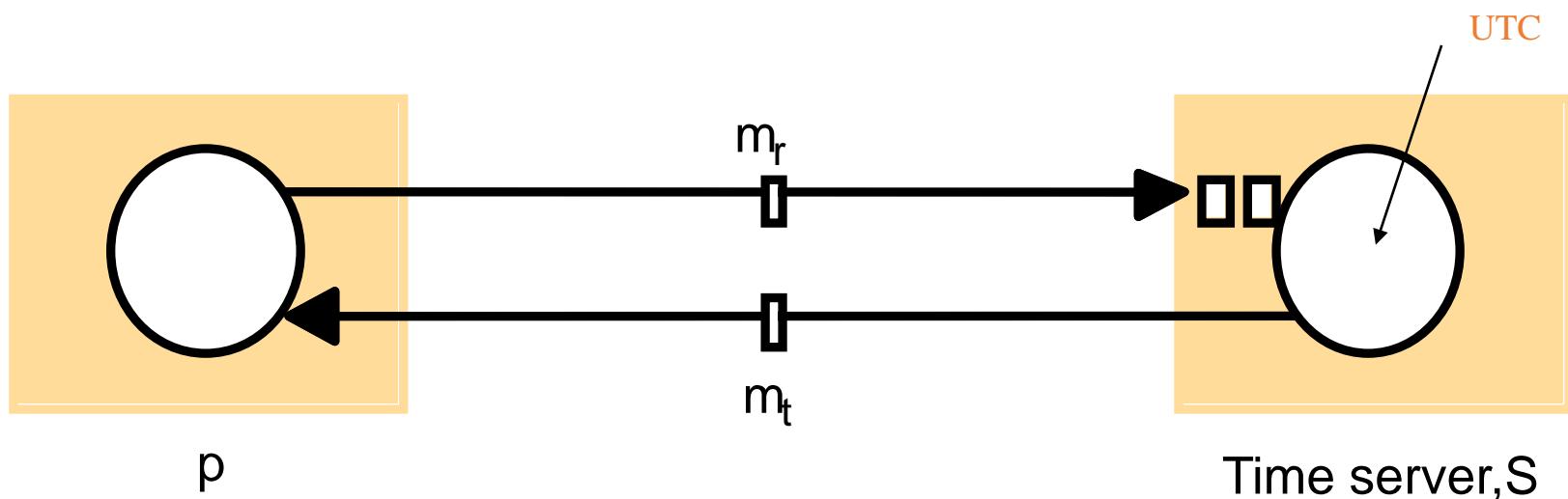
- Asynchronous system
 - Achieves synchronization only if the observed RTT between the client and server is sufficiently short compared with the required accuracy.

Cristian's clock synchronization algorithm

- Introduced by [Flaviu Cristian](#) in 1989
- Observations:
 - RTT between processes are reasonably short in practice, yet theoretically unbounded
 - Practical estimate possible if RTT is sufficiently short in comparison to required accuracy

Cristian's clock synchronization algorithm

- Periodically ($< \delta/2\rho$) each machine sends a request message m_r and the server sends the time in m_t .
- The roundtrip message transmission delay time is computed, say this is T_{round} .
- The algorithm sets $t + T_{round}/2$ as the new time. (naïve i.e. the elapsed time is split equally before & after S placed time in m_t)



Cristian's Algorithm (Accuracy)

- Assumption
 - Request & reply via same network
 - The value of minimum transmission time min is known or conservatively estimated.

Cristian's Algorithm (Accuracy)

- The earliest point at which S could have placed the time in m_t was *min* after p dispatched m_r .
- The latest point at which it could have done this was *min* before m_t arrived at p.
- The time by S's clock when the reply message arrives is therefore in the range $[t + min, t + T_{round} - min]$
- The width of this range is $T_{round} - 2min$, so the accuracy is $+/- (T_{round} / 2 - min)$

Cristian's Algorithm (Discussion)

- Variability can be dealt with making several requests & taking the min value of RTT to give the accurate estimate
- Cristian suggested making a series of measurements, and throw away measurements which exceed some threshold value. He assumes that these discarded entries are victims of network congestion. These measurements are then averaged to give a better number for network delay and got added to the current time.
- Cristian stated that the message that came back the fastest can be taken to be the most accurate since it presumably encountered the least traffic.

Cristian's Algorithm (Discussion)

- Only suitable for deterministic LAN environment of Intranet
- Problem of failure of single time server S
 - Redundancy through a group of servers (use multicasting and use only the first reply obtained)
 - How to decide if replies vary? (byzantine agreement)
- Imposter providing false clock readings
 - Authentication

The Berkeley Algorithm (internal synchronization)

- Gusella and Zatti at the University of California, Berkeley in 1989.
- A coordinator (time server): *master*
- Just the opposite approach of Cristian's algorithm
 - Periodically the *master* polls the time of each client (slave) whose clocks are to be synchronized.
 - Based on the answer (by observing the RTT as in Cristian's algorithm), it computes the average (including its own clock value) and broadcasts the new time.
- This method is suitable for a system in which no machine has a WWV receiver getting the UTC.
- The coordinator's time must be set manually by the operator periodically.

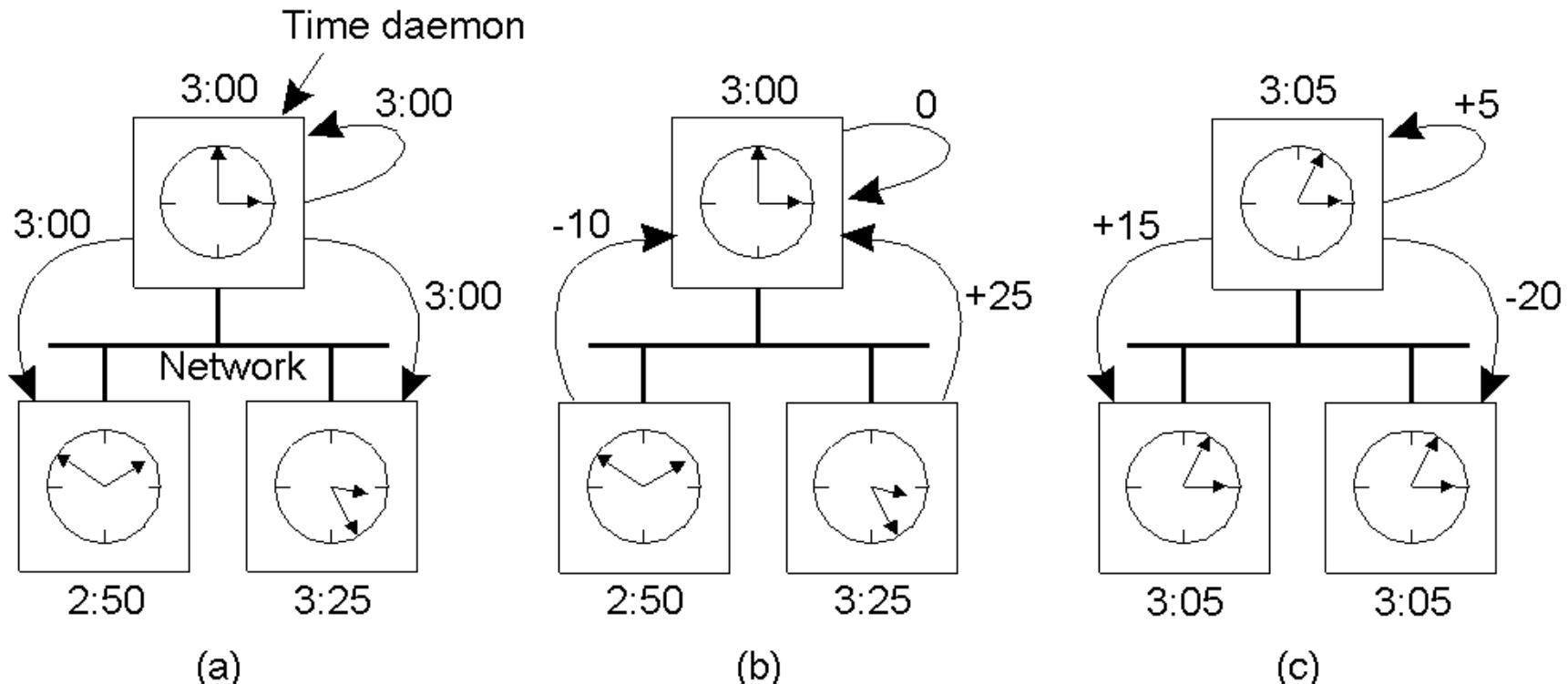
The Berkeley Algorithm

- The balance of probabilities is that the average cancels out the individual clock's tendencies to run fast or slow
- The accuracy depends upon a nominal maximum RTT between the master and the slaves
- The master eliminates any occasional readings associated with larger times than this maximum

The Berkeley Algorithm

- Instead of sending the updated current time back to the comps – which will introduce further uncertainty due to message transmission time – the master send the amount by which each individual slave's clock requires adjustment (+ or -)
- The algorithm eliminates readings from faulty clocks (since these could have significant adverse effects if an ordinary average was taken) – a subset of clock is chosen that do not differ by more than a specified amount and then the average is taken.

The Berkeley Algorithm



- The time daemon asks all the other machines for their clock values
- The machines answer
- The time daemon tells everyone how to adjust their clock

Averaging Algorithms

- Both Cristian's and Berkeley's methods are highly centralized, with the usual disadvantages - single point of failure, congestion around the server, ... etc.
- One class of decentralized clock synchronization algorithms works by dividing time into fixed-length re-synchronization intervals.
- The i^{th} interval starts at $T_0 + iR$ and runs until $T_0 + (i+1)R$, where T_0 is an agreed upon moment in the past, and R is a system parameter.

Averaging Algorithms

- At the beginning of each interval, every machine broadcasts the current time according to its clock.
- After a machine broadcasts its time, it starts a local timer to collect all other broadcasts that arrive during some interval S.
- When all broadcasts arrive, an *algorithm* is run to compute a new time.

Averaging Algorithms

- Some algorithms:
 - average out the time.
 - discard the m highest and m lowest and average the rest -- this is to prevent up to m faulty clocks sending out nonsense
 - correct each message by adding to it an estimate of propagation time from the source. This estimate can be made from the known topology of the network, or by timing how long it takes for probe message to be echoed.

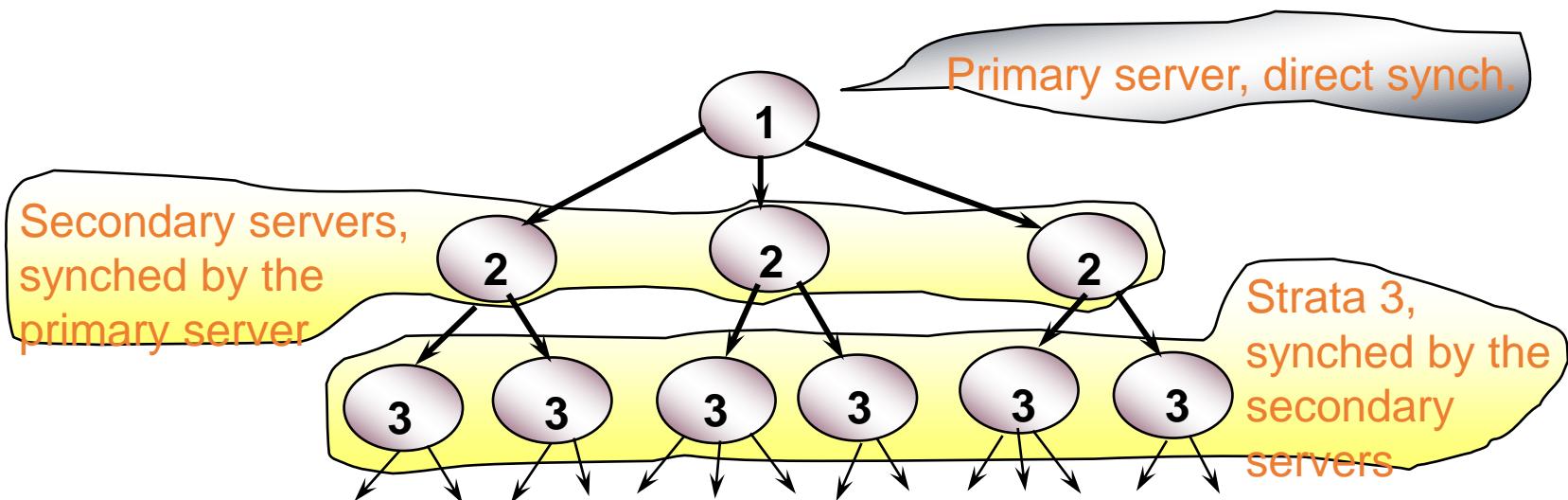
The Network Time Protocol (NTP)

- Cristian's and Berkeley algorithms → synch within intranet
- NTP – defines an architecture for a time service and a protocol to distribute time information over the Internet
 - Provides a service enabling clients across the Internet to be synchronized accurately to UTC
 - Provides a reliable service that can survive lengthy losses of connectivity
 - Enables client to resynchronize sufficiently frequently to offset clock drifts
 - Provides protection against interference with the time service

The Network Time Protocol (NTP)

Uses a network of time servers to synchronize all processes on a network.

Time servers are connected by a synchronization subnet tree. The root is in touch with UTC. Each node synchronizes its children nodes.



Messages exchanged between a pair of NTP peers

- Three modes of synchronization
- Multicast
 - Intended for use in high speed LAN
 - One or more servers periodically multicasts the time to servers running in the other computers connected to LAN
 - Servers set their clocks assuming a small delay
 - Can achieve relatively low accuracies – considered sufficient for many purposes

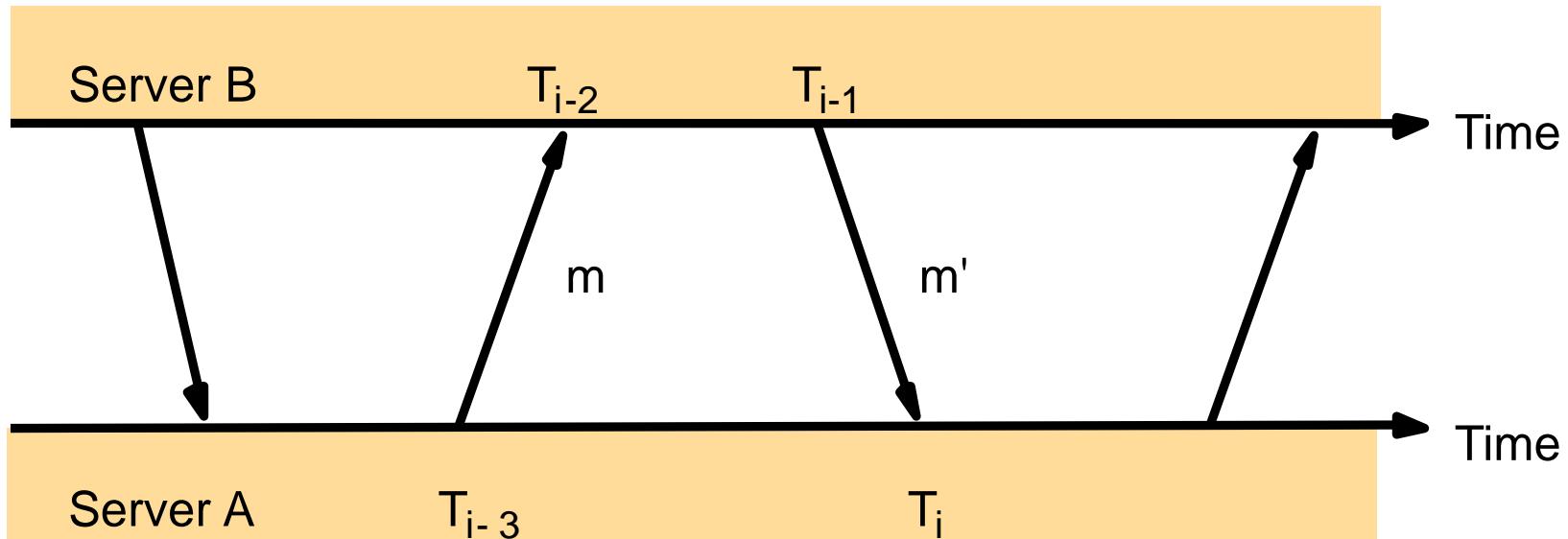
Messages exchanged between a pair of NTP peers

- Procedure-call mode
 - Similar to the operation of the Cristian Algorithm
 - One servers accepts requests from other servers, which it processes by replying with its timestamp
 - Suitable where higher accuracies are required than can be achieved with multicast or where multicast is not supported in hardware
 - E.g. file servers on the same or other LAN could contact a local server in procedure-call mode

Messages exchanged between a pair of NTP peers

- Symmetric mode
 - Intended for use by the servers that supply time information in LANs and by the higher levels (lower strata) of the synchronization subnet, where the highest accuracies to be achieved
 - A pair of servers operating in symmetric mode exchange messages bearing timing information

Messages Exchanged Between a Pair of NTP Peers (“Connected Servers”)



Each message bears timestamps of recent message events: the local time when the previous NTP message was sent and received, and the local time when the current message was transmitted.

Compensating for clock drift

- Compare time T_s provided by the time server to time T_c at computer C
 - If $T_s > T_c$ (e.g. 9:07 am vs 9:05 am), could advance C's time to T_s
 - May miss some clock ticks, probably OK
 - If $T_s < T_c$ (e.g. 9:07 am vs 9:10 am), cannot rollback C's time to T_s
 - Many applications assume that time always advances

Compensating for clock drift

- The solution is not to set C's clock back – but can cause C's clock to run slowly until it resynchronizes with the time server
- This can be achieved in SW, w/o changing the rate at which the HW clock ticks (an operation which is not always supported by HW clocks)

Is it enough to synchronize physical clocks?

- Value received from UTC receiver is only accurate to within 0.1–10 milliseconds
- At best, we can synchronize clocks to within 10–30 milliseconds of each other
- We have to synchronize frequently, to avoid local clock drift

Is it enough to synchronize physical clocks?

- In 10 ms, a 100 MIPS machine can execute 1 million instructions
 - Accurate enough as time-of-day
 - Not sufficiently accurate to determine the relative order of events on different computers in a distributed system

Logical Clocks

- Lamport (1978) showed that clock synchronization is possible and presented an algorithm for it. He also pointed out that clock synchronization need not be absolute.
- If two processes do not interact, it is not necessary that their clocks be synchronized because the lack of synchronization would not be observable and thus could not cause problems.

Logical Clocks (contd.)

- What usually matters is not that all processes agree on exactly what time it is, but rather, that they **agree on the order in which events occur.**
- Therefore, it is the internal consistency of the clocks that matters, not whether they are particularly close to the real time.
- It is conventional to speak of the clocks as **logical clocks.**

Logical Clocks (contd.)

- On the other hand, when clocks are required to be the same, and also must not deviate from the real time by more than a certain amount, these clocks are called **physical clocks**.

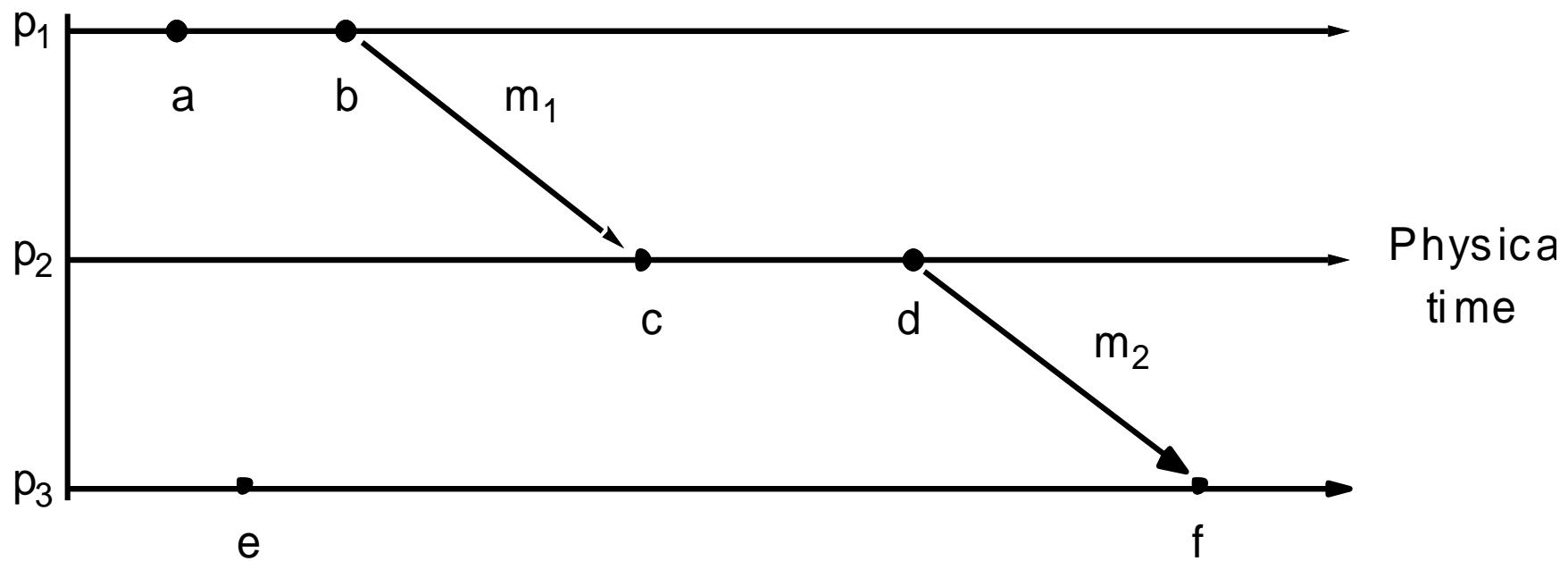
Logical Clocks

- ❖ Is it always necessary to give *absolute* time to events?
- ❖ Suppose we can assign *relative* time to events, in a way that does not violate their *causality*
 - ❖ Well, that would work – we humans run our lives without looking at our watches for everything we do
 - ❖ First proposed by Leslie Lamport in the 70's
 - ❖ Define a logical relation *Happens-Before* (\rightarrow) among events:
 1. On the same process: $a \rightarrow b$, if $time(a) < time(b)$
 2. If p1 sends m to p2: $send(m) \rightarrow receive(m)$
 3. (Transitivity) If $a \rightarrow b$ and $b \rightarrow c$ then $a \rightarrow c$

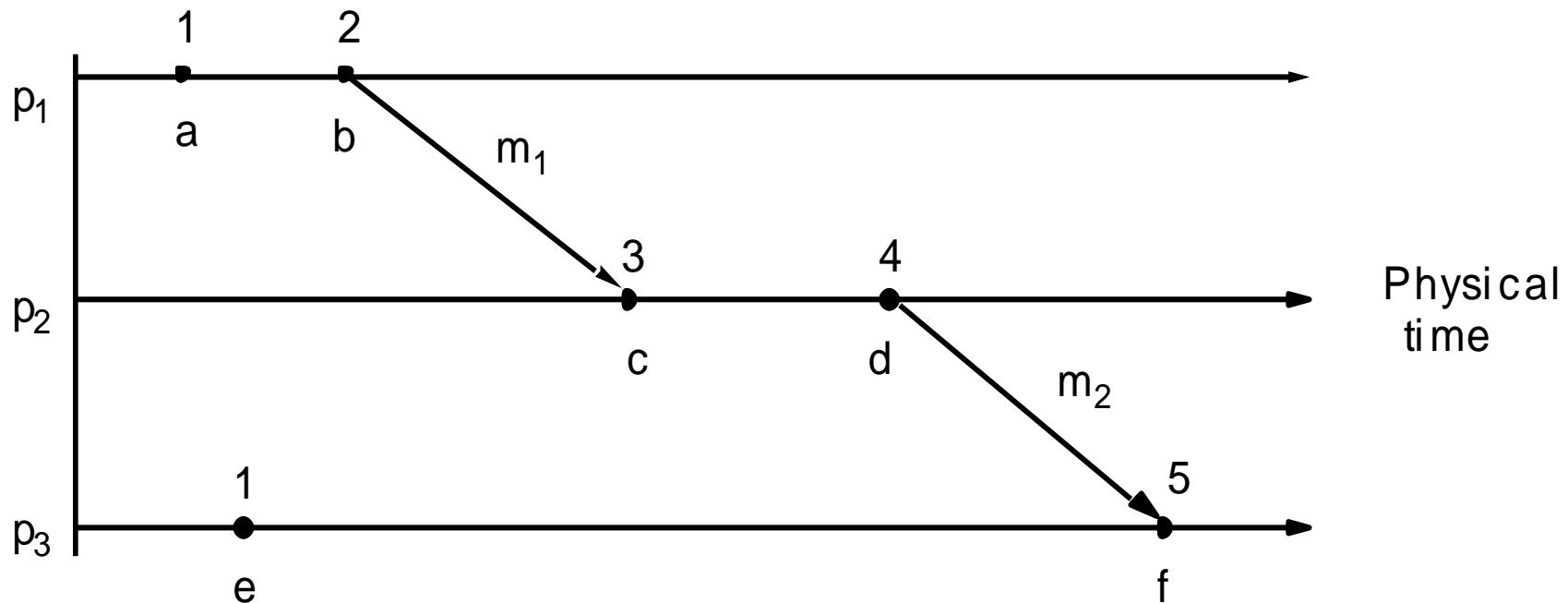
Logical Clocks

- ❖ Lamport Algorithm assigns **logical timestamps to events**:
 - All processes use a counter (clock) with initial value of zero
 - A process increments its counter when a **send** or an **instruction** happens at it. The counter is assigned to the event as its timestamp.
 - A **send (message)** event carries its timestamp
 - For a **receive (message)** event the counter is updated by
$$\max(\text{local clock}, \text{message timestamp}) + 1$$

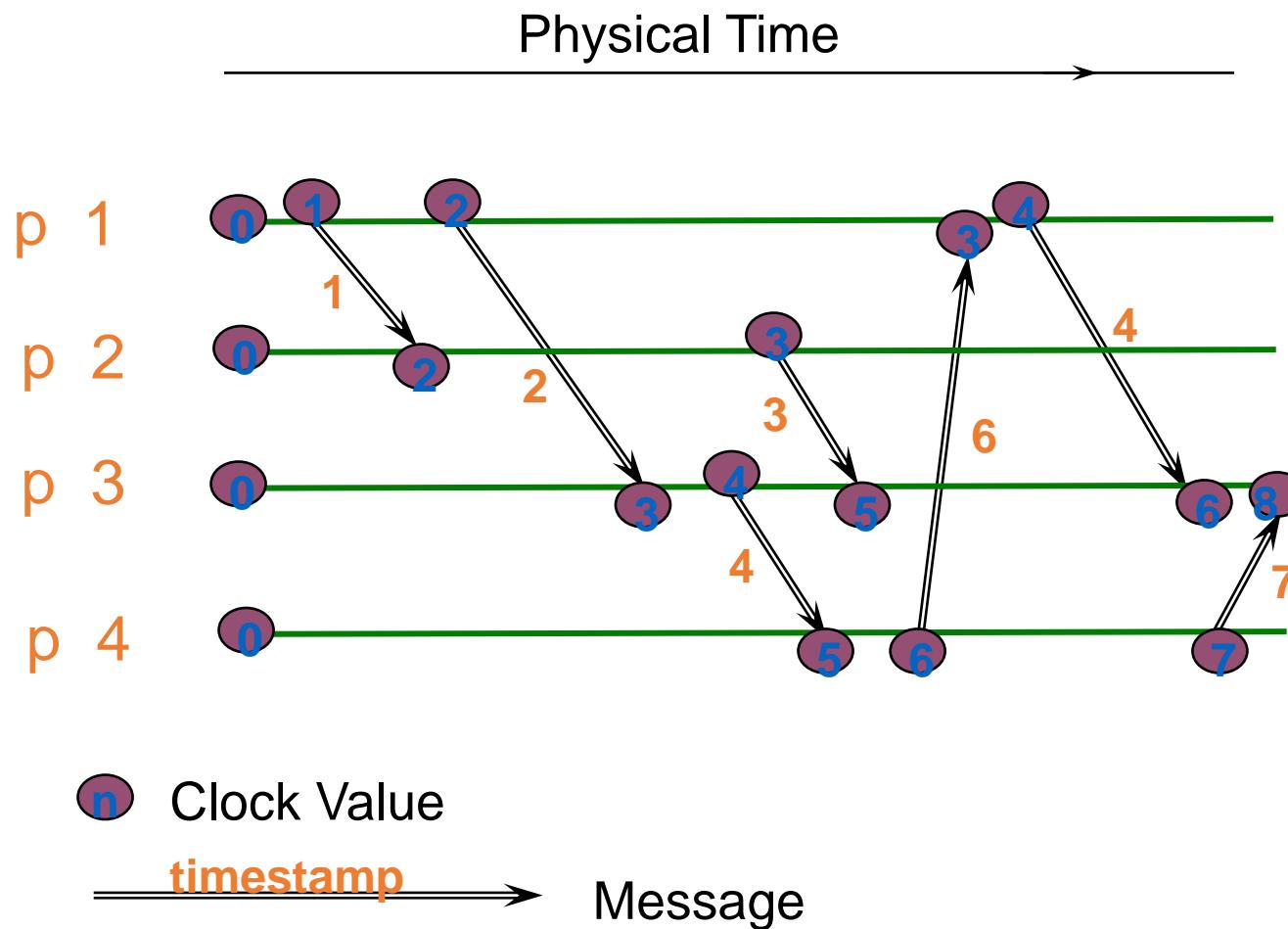
Events Occurring at Three Processes



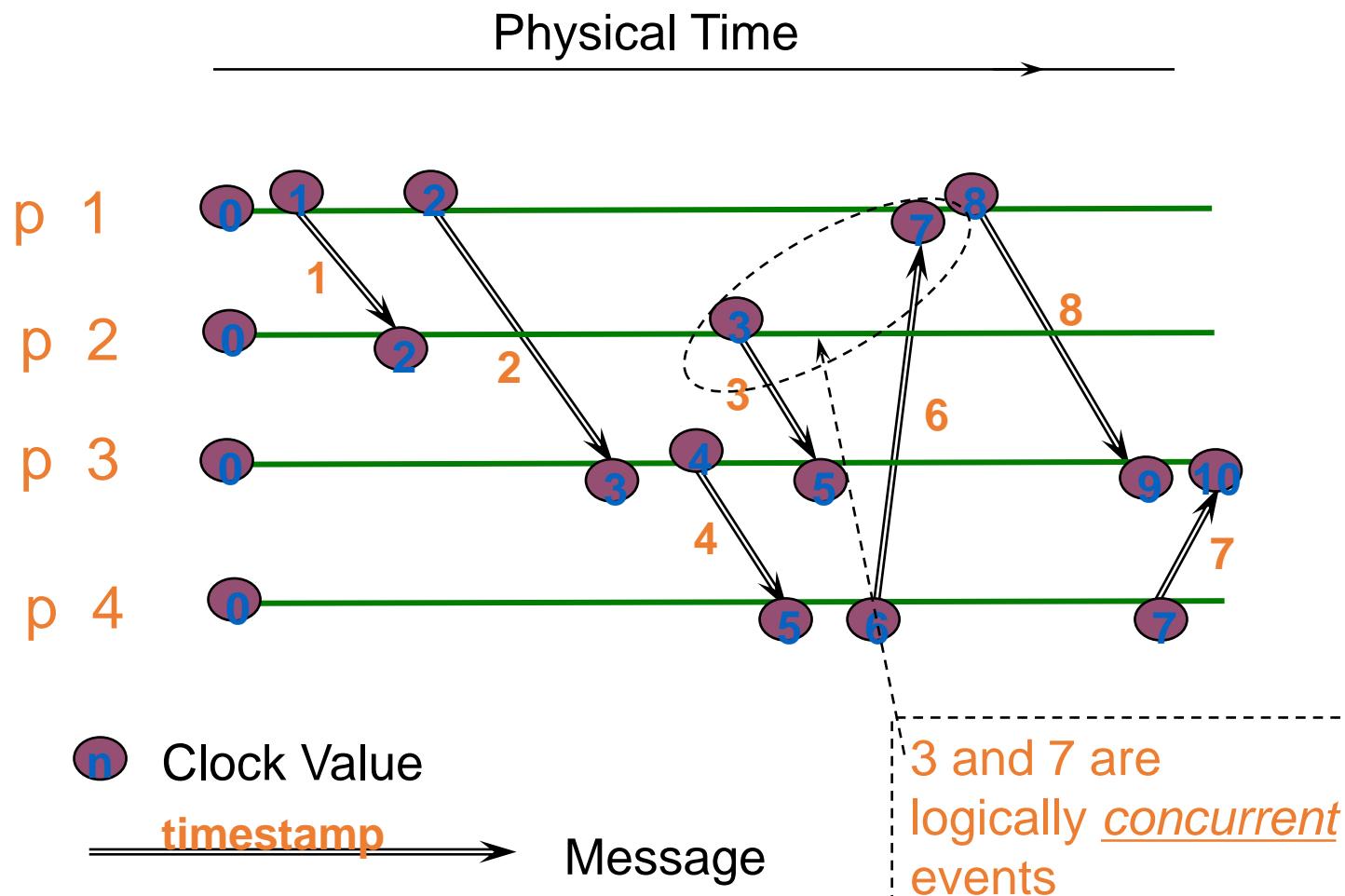
Lamport Timestamps



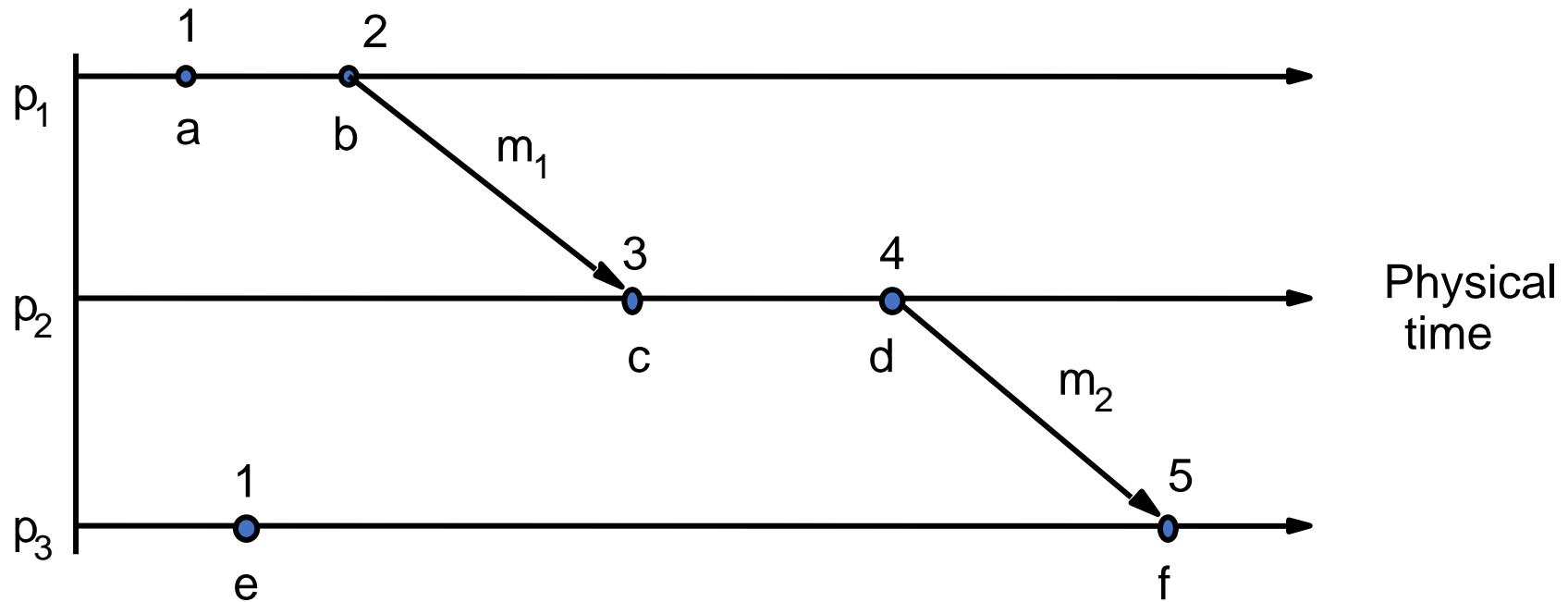
Find the Mistake: Lamport Logical Time



Corrected Example: Lamport Logical Time



Events occurring at three processes



Limitations of Lamport Clocks:

It represents a *partial order*, if $a \rightarrow b$ it implies that $L(a) < L(b)$ but the converse is not true, i.e., if $L(a) < L(b)$ it doesn't imply $a \rightarrow b$. From the fig. $L(b) > L(e)$ but in fact, $b \parallel e$.

Vector Logical Clocks

- ❖ With Lamport Logical Timestamp

$e \rightarrow f \Rightarrow \text{timestamp}(e) < \text{timestamp}(f)$, but
 $\text{timestamp}(e) < \text{timestamp}(f) \Rightarrow \{e \rightarrow f\} \text{ OR } \{e \text{ and } f \text{ concurrent}\}$

- ❖ Vector Logical time addresses this issue:

- ❑ N processes. Each uses a vector of counters (logical clocks), initially all zero. i^{th} element is the clock value for process i .
 - ❑ Each process i increments the i^{th} element of its vector upon an **instruction** or **send** event. Vector value is timestamp of the event.
 - ❑ A **send(message)** event carries its vector timestamp (counter vector)
 - ❑ For a **receive(message)** event,

$$V_{\text{receiver}}[j] = \begin{cases} \text{Max}(V_{\text{receiver}}[j], V_{\text{message}}[j]), & \text{if } j \text{ is not self} \\ V_{\text{receiver}}[j] + 1 & \text{otherwise} \end{cases}$$

Vector Clocks (Mattern '89 & Fidge '91)

- To overcome the problems of Lamport Clock → Vector clocks
 - A *vector clock* for a system of N processes is an array of N integers. P_i keeps its VC V_i , to timestamp local events.
 - Vector clock rules:

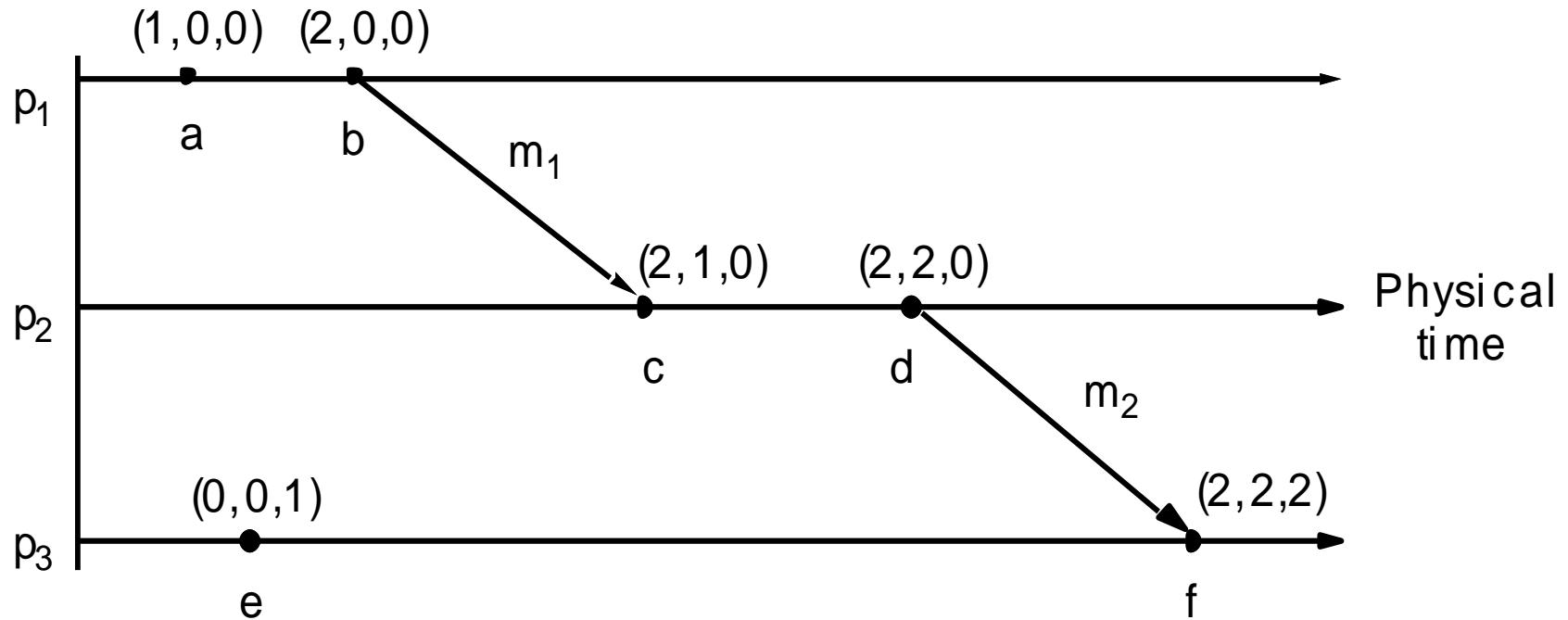
VC1: Initially for each process p_i , $V_i[j] = 0$,
for $j = 1, 2, \dots, N$

VC2: Just before time stamping an event, p_i sets
 $V_i[i] = V_i[i] + 1$.

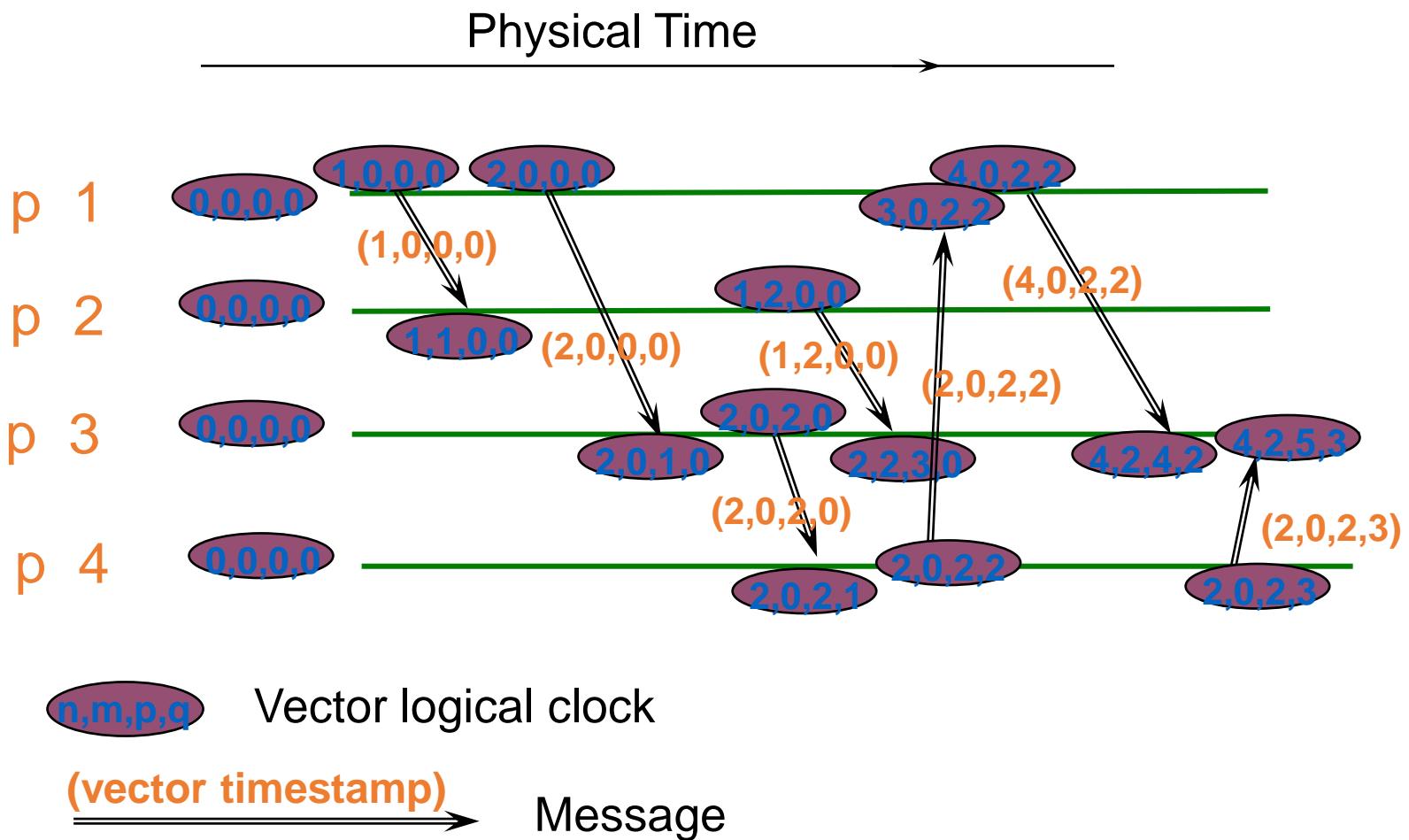
VC3: p_i sends $t = V_i$ in every message it sends

VC4: When p_i receives t in a message, it sets
 $V_i[j] = \max(V_i[j], t[j])$, *for* $j = 1, 2, \dots, N$
(the max is computed component wise)

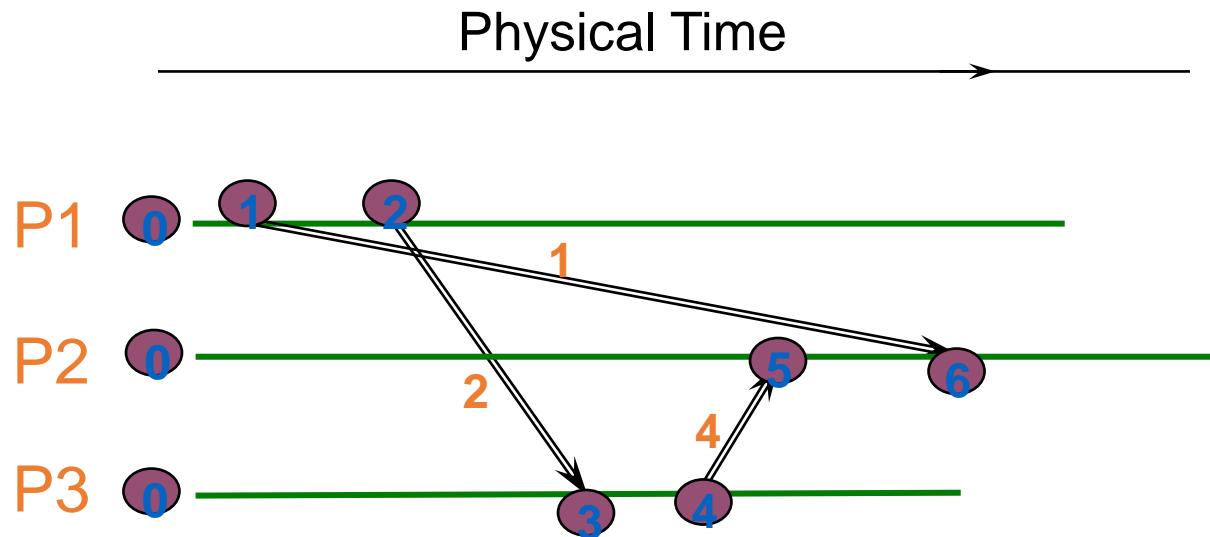
Vector Timestamps



Example: Vector Timestamps

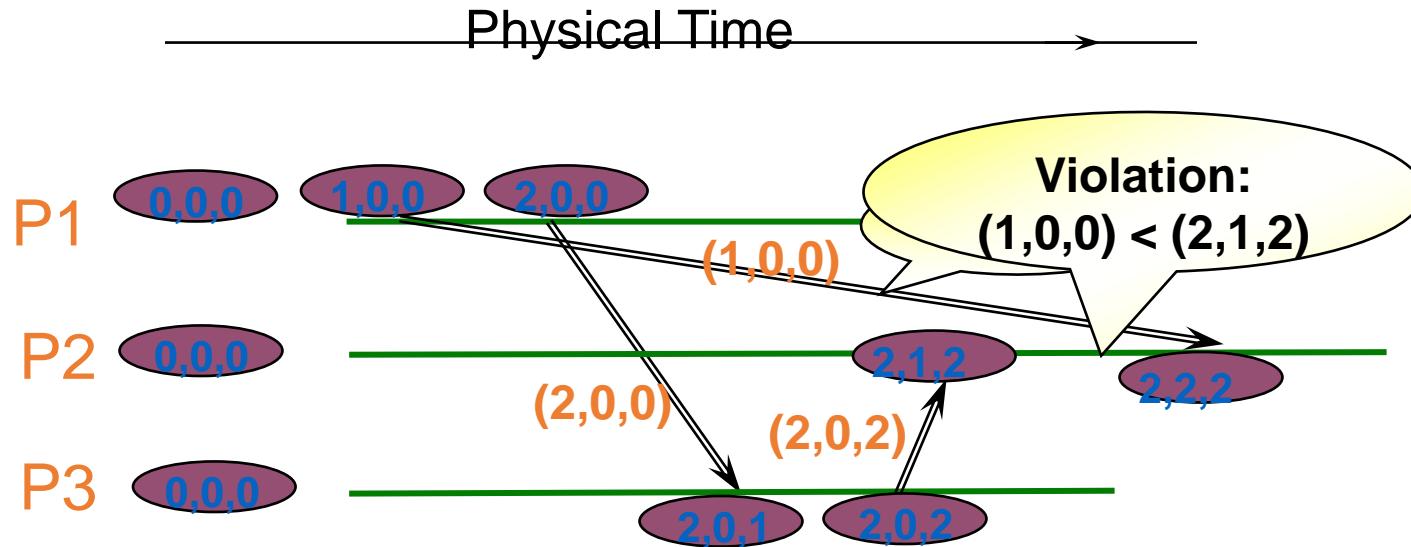


Side Issue: Causality Violation



- Causality violation occurs when order of messages causes an action based on information that another host has not yet received.
- In designing a distributed system, potential for causality violation is important to notice

Detecting Causality Violation



- Potential causality violation can be detected by vector timestamps.
- If the vector timestamp of a message is less than the local vector timestamp, on arrival, there is a potential causality violation.

Summary

Time synchronization important for distributed systems

Physical clock Synchronization

Cristian's algorithm

Berkeley algorithm

NTP

Logical Clock Synchronization - Relative order of events enough for practical purposes

Lamport's logical clocks

Vector clocks