Service-Oriented Architecture

SOA and Web services

Lecture 01

Henry Novianus Palit
hnpalit@petra.ac.id

SOA *(1)*

- (Software) Services:
  - Self-contained modules that can be described, published, located, orchestrated, and programmed over standard middleware platforms
  - They encapsulate logic within a distinct context, specific to a business task, a business entity, or some other logical grouping
  - The size and scope of the service logic can vary; further, service logic can encompass the logic provided by other services
- Service-Oriented Computing:

  An emerging computing paradigm that utilizes services as the constructs to support the development of rapid, low cost composition of distributed applications

SOA *(2)*

- Service-Oriented Architecture:
  - A model in which automation logic is decomposed into smaller, distinct units of logic
  - A logical way of designing a software system to provide services to either end user applications or to other services distributed in a network, via published and discoverable interfaces
  - As a design philosophy, SOA is independent of any specific technology (i.e., implementation-agnostic), e.g., Web services or J2EE
  - It is positioned as the next phase in the evolution of business automation

SOA *(3)*

- Key aspects of service-oriented computing:
  - *Loose coupling* → minimize dependencies
  - *Service contract* → adhere to a communication agreement
  - *Autonomy* → have control over the logic they encapsulate
  - *Abstraction* → hide logic from the outside world
  - *Reusability* → divide logic into smaller services to promote reuse
  - *Composability* → collections of services can be coordinated and assembled to form composite services
  - *Statelessness* → minimize retaining information specific to an activity
  - *Discoverability* → can be found and assessed via available discovery mechanisms

Web Service *(1)*

- A self-describing, self-contained software module available via a network, such as the Internet, which completes tasks, solves problems or conducts transactions on behalf of a user or application
- A platform-independent, loosely coupled, self-contained, programmable Web-enabled application that can be described, published, discovered, composed, and configured using XML artifacts (open standards) for the purpose of developing distributed interoperable applications
- It constitute a distributed computer infrastructure made up of many different interacting application modules trying to communicate over private or public networks to form virtually a single logical system

Web Service *(2)*

- Closer examination of the terms:
  - *Loosely coupled* software modules → the service interface needs to be neutral and independent of the underlying platform, OS, and the programming language the service is implemented in
  - *Discrete* functionality → the module describes what it does, how to invoke its functionality, and what result to expect in return
  - *Programmable* access → it can be called by and exchange data with other software modules and applications
  - Be *dynamically found and included* in applications → it can be assembled to serve a particular function, solve a specific problem, or deliver a particular solution to a customer
  - Be *described* in a standard description language → WSDL
  - Be *distributed* over the Internet → it uses existing, ubiquitous transport Internet protocols like HTTP

Web Service *(3)*

- Its efforts focus on reusing existing applications (including legacy code) to facilitate integration with other applications, often motivated by the desire for new forms of sharing of services across lines of business or between business partners
- Its long term goal is to enable distributed applications that can be dynamically assembled according to changing business needs, and customized based on device, network, and user access, while enabling wide utilization of any given piece of business logic wherever it is needed
- Examples of Web service:
  - A self-contained **business task**, e.g., funds withdrawal / deposit service
  - A fully fledged **business process**, e.g., automated bill inquiry handling
  - An **application**, e.g., life insurance application, demand forecasts & stock replenishment
  - A **service-enabled resource**, e.g., access to a particular back end DB containing patient medical records

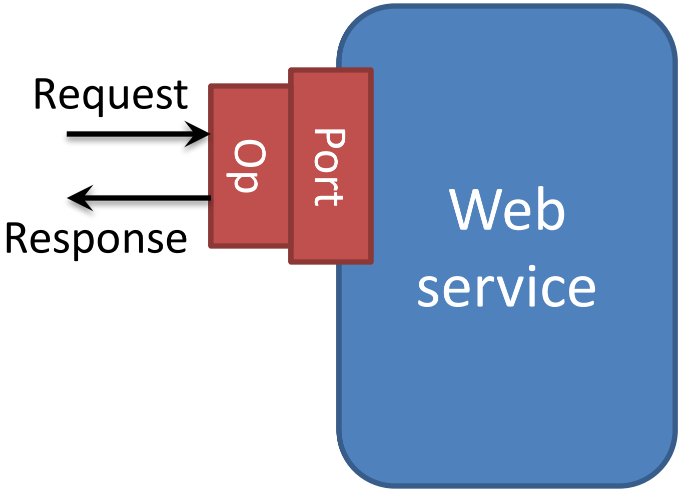
Misconceptions about SOA

- An application that uses Web services is service-oriented
- SOA is just a marketing term used to re-brand Web services
- SOA is just a marketing term used to re-brand distributed computing with Web services
- SOA simplifies distributed computing
- An application with Web services that uses WS-* extensions is service-oriented
- If you understand Web services you won’t have a problem building SOA
- Once you go SOA, everything becomes interoperable

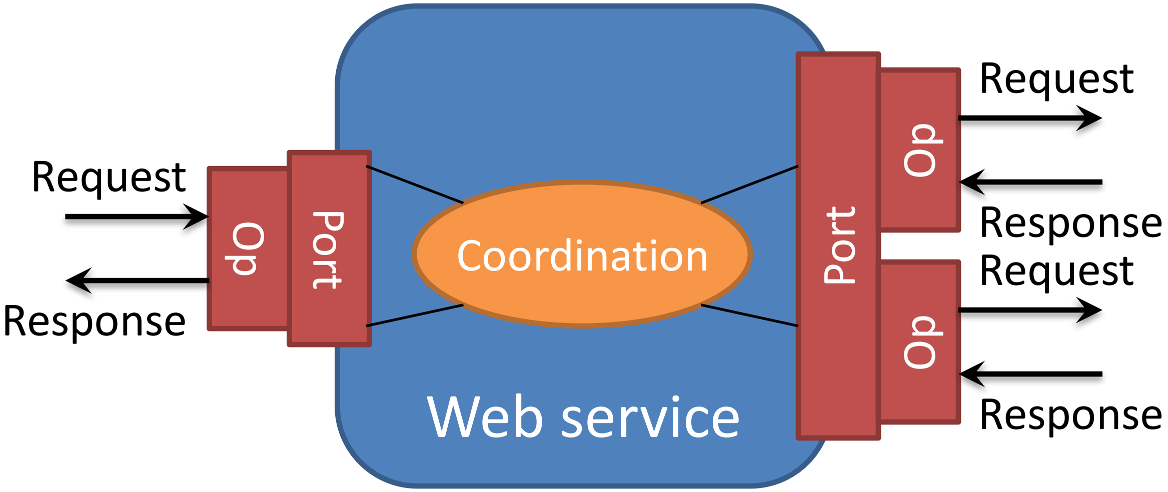
Benefits of SOA

- Improved integration (and intrinsic interoperability)
- Inherent reuse → lower the cost and effort
- Streamlined architectures and solutions → reduce processing overhead and skill-set requirements
- Leveraging the legacy investment
- Establishing standardized XML data representation
- Focused investment on communications infrastructure
- “Best-of-breed” alternatives → vendor neutral
- Organizational agility → reduce cost and effort to respond and adapt to business / technology change

Types of Web service *(1)*



Simple or informational service



Complex service

Types of Web service *(2)*

- Subcategories of informational services:
  - Pure **content services** → e.g., weather report info, simple financial info, stock quote info, news items, etc.
  - Simple **trading services** → provide a seamless aggregation of info across disparate existing systems and information sources so that the requester can make informed decision, e.g., logistic services
  - Simple **syndication services** → value-added info Web services that purport to plug into commerce sites of various types, e.g., reservation services on a travel site or rate quote services on an insurance site
- Informational services are *atomic* in nature (i.e., performing a complete unit of work that leaves its underlying data stores in a consistent state) and *not transactional* in nature (although their back-end realizations may be)

Types of Web service *(3)*

- Complex (composite) services typically <u>involve the assembly and invocation of many pre-existing services</u>, possibly found in diverse enterprises, <u>to complete a multi-step business interaction that requires coordination</u>
- For example, a supply chain application which involves order taking, stocking orders, sourcing, inventory control, financials, and logistics
- Subcategories of complex services:
  - Complex services that compose **programmatic Web services** → e.g., an inventory management process that involves an inventory checking service
  - Complex services that compose **interactive Web services** → expose a multi-step Web app behavior and typically deliver the app directly to a browser and eventually to a human user for interaction

Web Service Properties *(1)*

- Functional vs. non-functional descriptions
  - Functional → operational characteristics that define the overall behavior of the service (e.g., how the service is invoked, the location where it is invoked, etc.)
  - Non-functional → service quality attributes such as service metering & cost, performance metrics (e.g., response time or accuracy), security attributes, authorization, authentication, (transactional) integrity, reliability, scalability, and availability
- Stateless vs. stateful
  - Stateless → maintain no context or state, e.g., an informational weather report service
  - Stateful → require their context to be preserved from one invocation to the next, e.g., typical business processes

Web Service Properties *(2)*

- Tight vs. loose coupling
  - Tight coupling (synchronous interaction) → need to know how their partner applications behave, how their partner requires to be communicated with, where the location of their partner
  - Loose coupling (asynchronous or event-driven interaction) → connect and interact more freely, need not know or care how their partner applications behave or are implemented

	Tight coupling	Loose coupling
Method of communication	Synchronous	Asynchronous
Messaging style	RPC-style	Document-style
Message path	Hard coded	Routed
Underlying platform	Homogeneous	Heterogeneous
Binding protocol	Static	Dynamic – late binding
Overall objective	Re-use	Re-use, flexibility, broad applicability

Service-Oriented Architecture

Distributed Computing Infrastructure

Lecture 01

Henry Novianus Palit

hnpalit@petra.ac.id

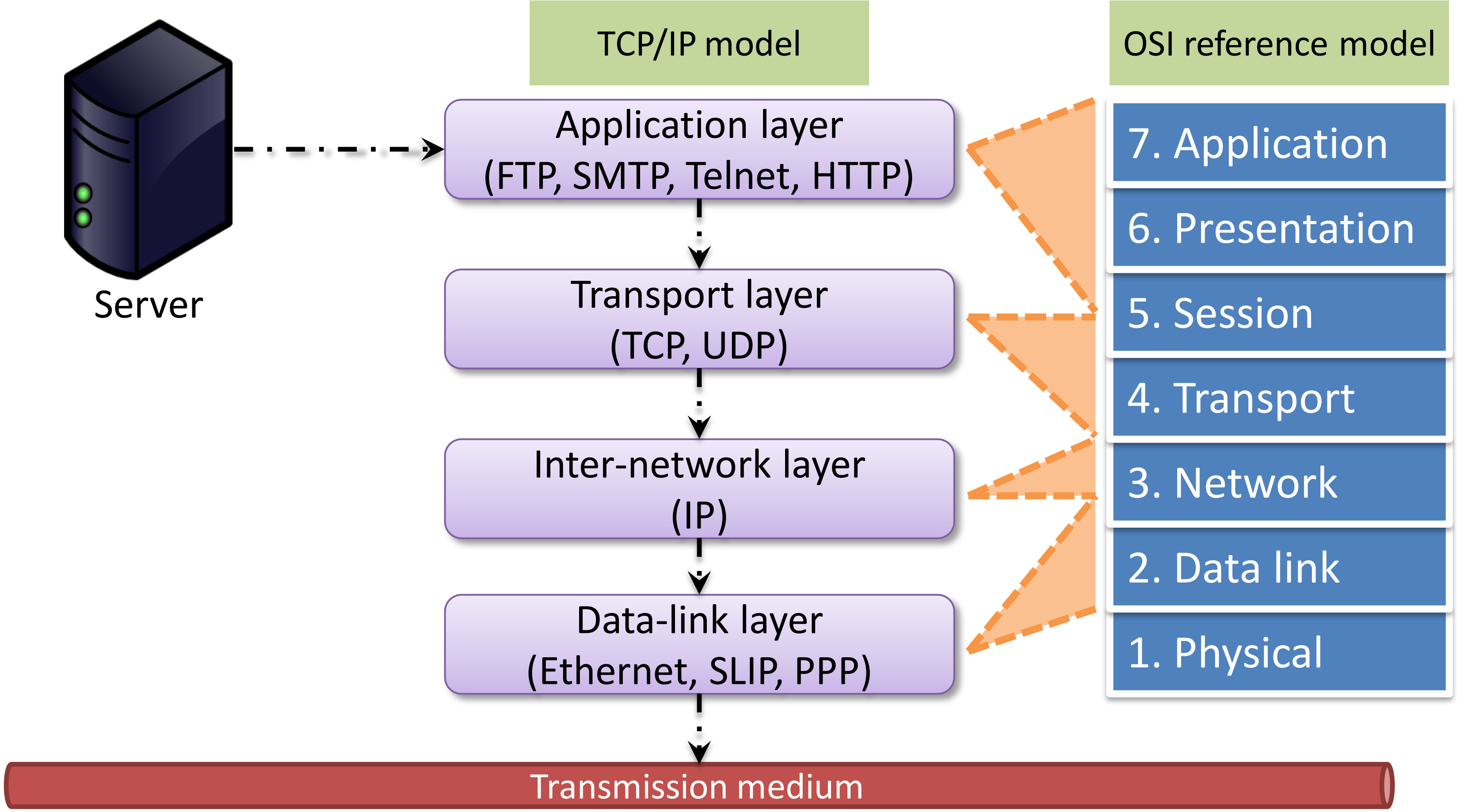
Distributed Systems

- A collection of (probably heterogeneous) networked computers, which communicate & coordinate their actions by passing messages
- It has numerous operational components (computational elements, such as servers or applications), which are
  - Autonomous → have full control over their parts at all times
  - Heterogeneous (typically) → may be written in different programming languages and operate under different OS & hardware platforms
- Characteristics:
  - Sharing of resources
  - Executing applications (often, multi-threaded) concurrently
  - Requiring inter-process communication mechanisms to manage interaction between processes on different machines
  - Can fail independently in many ways (e.g., computer failure or crashed application)

Support of Internet Protocols

- The most prominent of the Internet protocols is TCP/IP
  - Internet Protocol (IP)
    - The basic protocol of the Internet, which enables the unreliable delivery of individual packets from one host to another
    - It makes no guarantees as to whether the packet will be successfully delivered, how long it will take, or if multiple packets will arrive in the order they were sent
  - Transport Control Protocol (TCP)
    - It complements IP by providing for connectivity and reliable delivery of streams of data from one host to another across networks
- To identify a host within an interconnected network, each host is assigned an address, called an *IP address*, which consists of a network identifier and a host identifier
- To facilitate client-server communication, each application (e.g., FTP or telnet) is assigned a unique address, called a *port*

TCP/IP Stack


TCP/IP model
OSI reference model
Server
Application layer
(FTP, SMTP, Telnet, HTTP)
Transport layer
(TCP, UDP)
Inter-network layer
(IP)
Data-link layer
(Ethernet, SLIP, PPP)
7. Application
6. Presentation
5. Session
4. Transport
3. Network
2. Data link
1. Physical
Transmission medium

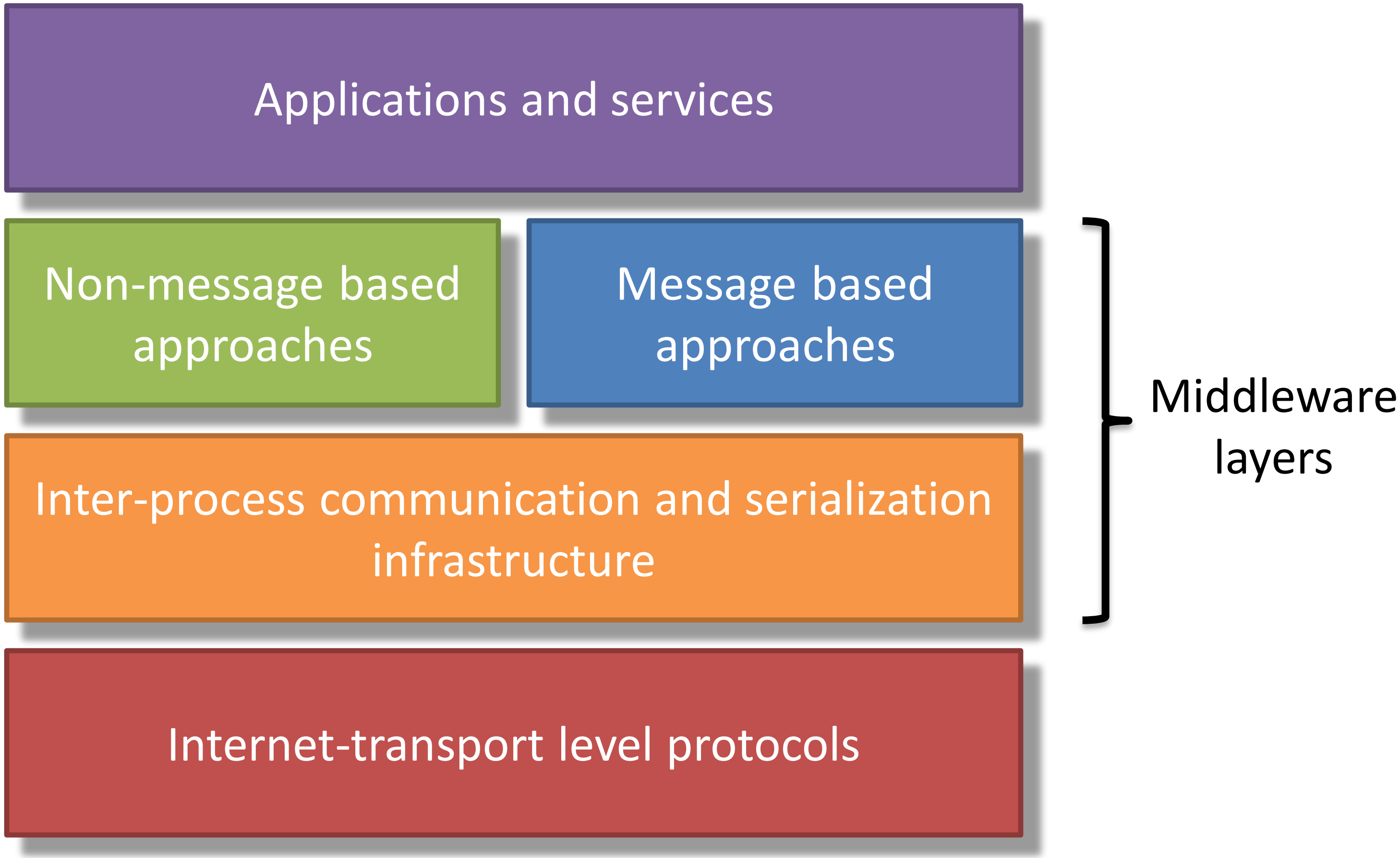
Support of Middleware *(1)*

- Middleware is a layer of enabling software services that allow application elements to interoperate across network links, despite differences in underlying communication protocols, system architectures, operating systems, databases, and other application services
  - It is designed to help manage the complexity and heterogeneity inherent in distributed systems by building a bridge between different systems, thereby enabling communication and transfer of data
  - Its role is to ease the task of designing, programming, and managing distributed applications by providing a simple, consistent, integrated distributed programming environment
  - It is essentially a distributed platform that lives above the OS and abstracts over the complexity and heterogeneity of the underlying distributed environment

Support of Middleware *(2)*

- Middleware services provide APIs (application programming interfaces) to allow an application to
  - locate applications transparently across the network
  - shield software developers from low level, tedious, and error prone platform details, e.g., socket level network programming
  - provide a consistent set of higher level network oriented abstractions that are much closer to application requirements to simplify the development of distributed systems
  - leverage previous developments and reuse them
  - provide a wide array of services such as reliability, availability, authentication, and security that are necessary for applications to operate effectively in a distributed environment
  - scale up in capacity without losing function

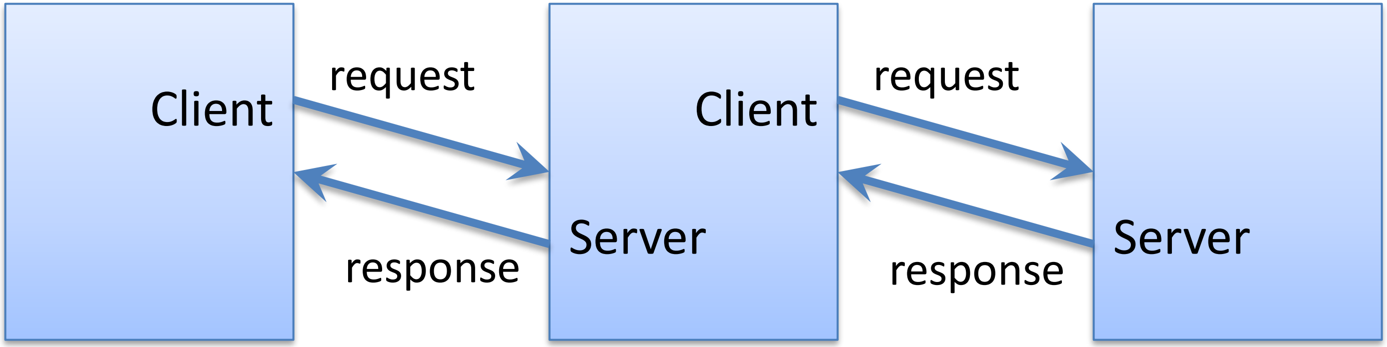
Middleware Layers


Applications and services
Non-message based approaches
Message based approaches
Inter-process communication and serialization infrastructure
Internet-transport level protocols
Middleware layers

Client-Server Architecture *(1)*

- A computational architecture in which processing and storage tasks are divided between two classes of network members – clients and servers
- It is a widely applied form of distributed processing and one of the common solutions to the conundrum of handling the need for both centralized data control & widespread data accessibility
- It involves client processes (service consumers) requesting service from server processes (service providers)
- Servers may, in turn, be clients of other servers, e.g., a Web server is often a client of a local file server (or database server) that manages the files (storage structure) in which Web pages are stored

Client-Server Architecture *(2)*



- The same device may function as both client and server, e.g., a Web server can function as both client and server when local browser sessions are run there
- It is the most prevalent structure for Internet applications such as the Web, e-mail, file transfer, Telnet applications, newsgroups, etc.
- *Thin client* → the architecture in which the client neither download nor execute the app code on the client's computer

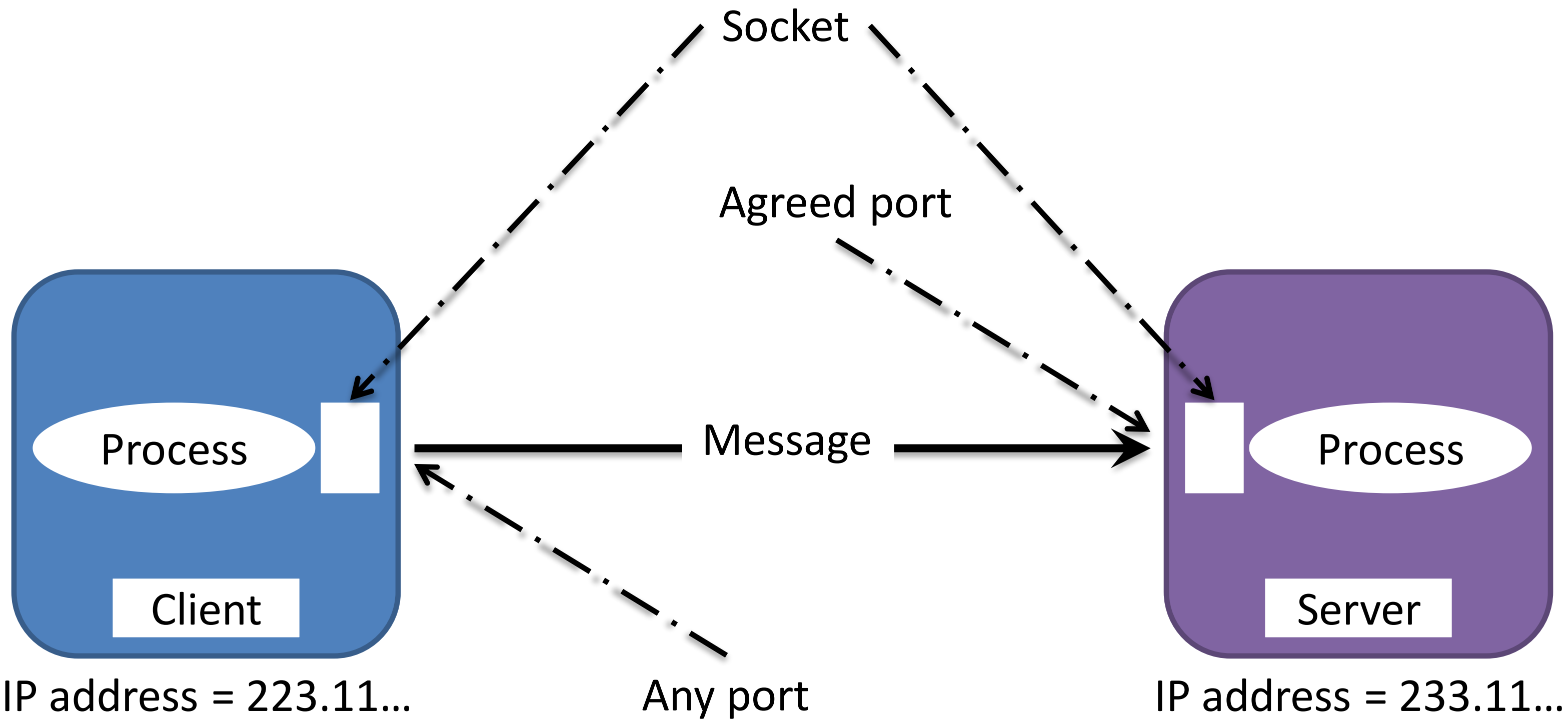
Inter-Process Communication *(1)*

- Distributed systems and applications communicate by exchanging messages
- A message typically comprises three basic elements:
  - Message header → used by both the messaging system and the application developer to provide info about message characteristics (e.g., destination, type, expiration time, etc.)
  - Message properties → contain a set of application-defined name-value pairs; these properties are essentially parts of the message body that get promoted to a special section of the message so that clients can apply filtering or specialized routing to the message
  - Message body → carries the actual payload of the message; the common formats are plain text, a raw stream of bytes, or a special XML message type
- Distributed applications have application layer protocols that define the format and orders of the messages exchanged between processes, as well as the actions taken on the transmission or receipt of a message

Inter-Process Communication *(2)*

- *Marshalling* (*serialization* in Java and XML terms) is the process of taking an object or any other form of structured data item and breaking it up so that it can be transmitted as a stream of bytes over a communication network
- *Unmarshalling* (*deserialization* in Java and XML terms) is the process of converting the assembled stream of bytes on arrival to produce an equivalent object or form of structured data at the destination point
- For inter-process communication, servers usually publicize their port numbers (to receive messages) for use by clients
- A process sends messages into, and receives messages from, the network through its socket; a socket could be thought of as the entry point to the process and it must be bound to a local port & refer to the IP address of the host

Ports and Sockets



Inter-Process Communication *(3)*

- Messaging modes:
  - *Synchronous* or time dependent
    - Both the sending and the receiving applications must be ready to communicate with each other at all times
    - Execution flow at the client's side (i.e., the sending application) is interrupted to execute the call
    - E.g., remote procedure calls (RPC)
  - *Asynchronous* or time independent
    - Both the sending and the receiving applications do not have to be active at the same time for processing to occur
    - The sending application employs a *send and forget* approach that allows it to continue to execute after it sends the message
    - E.g., store & forward, publish/subscribe

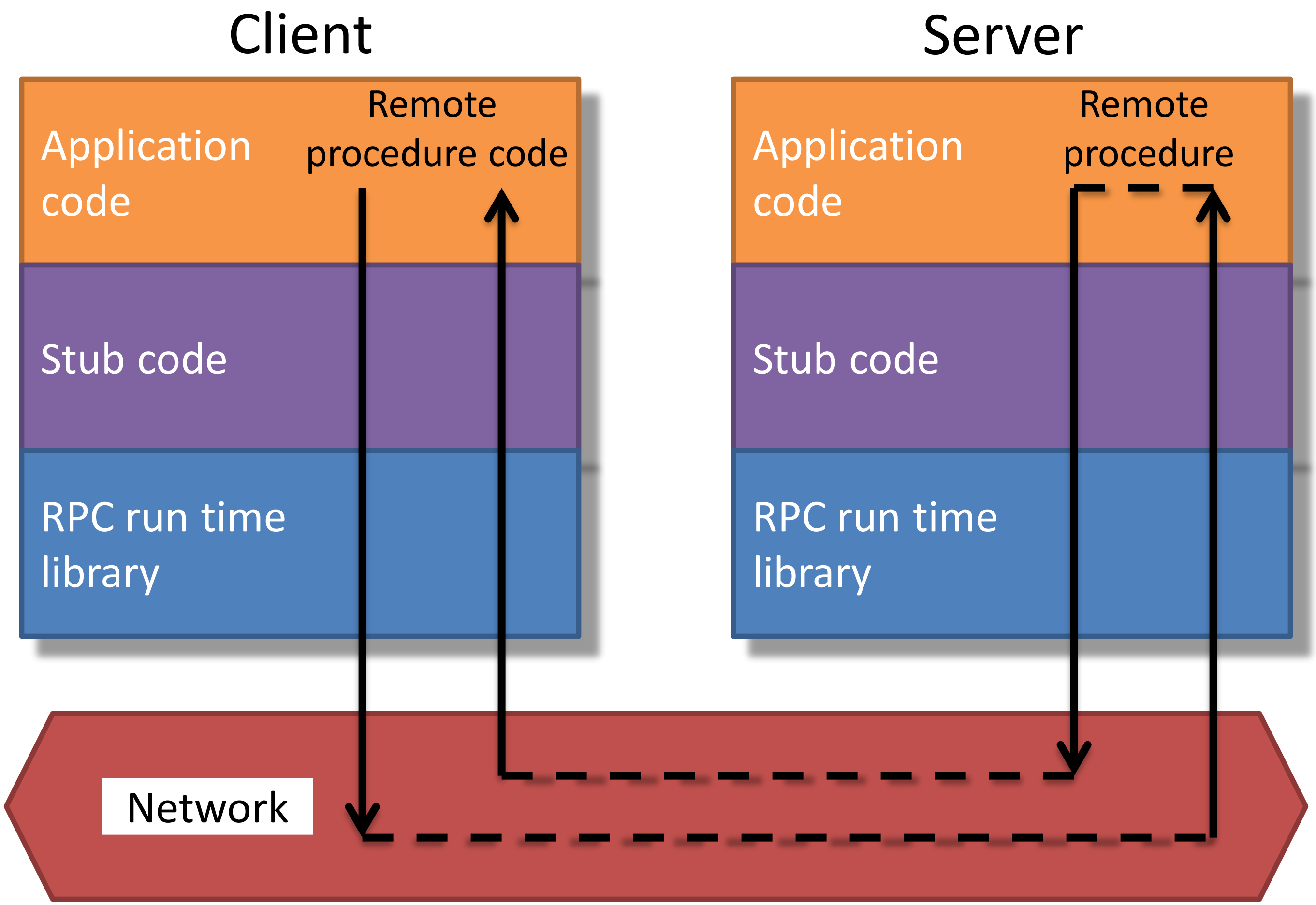
Remote Procedure Calls / RPC *(1)*

- The RPC programming style mimics the serial thread of execution that a normal non-distributed application would use
- It is the simplest way to implement client-server applications because it keeps the details of network communications out of the application code
- In RPC programming style, an object and its methods are *remoted* such that the invocation of the method can happen across a network separation
- In client application code, an RPC looks like a local procedure call, because it is actually a call to a local proxy known as a *client stub* (i.e., a surrogate code that supports RPCs); the role is to marshall the procedure identifier and arguments into a request message, sent to the server via a comm module

Remote Procedure Calls / RPC *(2)*

- The client stub communicates with a *server stub* using the RPC run time library (i.e., a set of procedures that support all RPC applications); the server stub will unmarshall the arguments in the request message, call the corresponding service procedure, and marshall the return results for the reply message
- The server stub communicates its output to the client stub, again by using the RPC run time library
- In an RPC environment, <u>each application needs to know the intimate details of the interface of every other application</u>, i.e., the number of methods and the details of each method signature it exposes
- RPCs work well for smaller, simple applications where <u>communication is primarily point-to-point</u> and <u>do not scale well to large, mission critical applications</u>

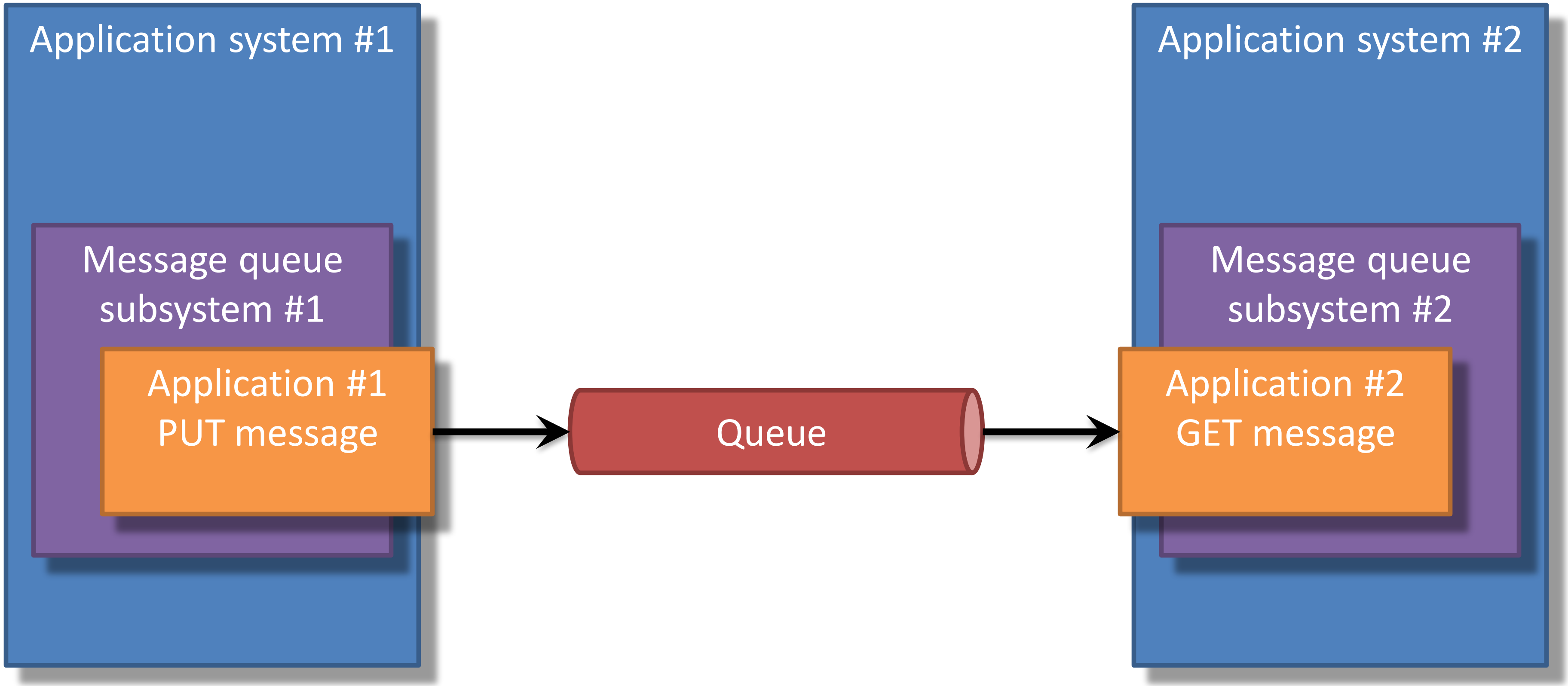
Remote Procedure Calls / RPC *(3)*



Store and Forward *(1)*

- In a store and forward queuing mechanism, messages are placed on a virtual channel called a *message queue* by a sending application and are retrieved by the receiving application as needed
- The message queue is independent of both the sender and receiver applications and acts as a buffer between the communicating applications
- Message queuing provides a highly reliable, although not always timely, means of ensuring that application operations are completed
- Store and forward queuing mechanism is typical of a many-to-one messaging paradigm
- A *message acknowledgment* is a mechanism that allows the messaging system to monitor the progress of a message so that it knows when the message was successfully produced and consumed

Store and Forward *(2)*


Application system #1
Message queue subsystem #1
Application #1 PUT message
Queue
Application system #2
Message queue subsystem #2
Application #2 GET message

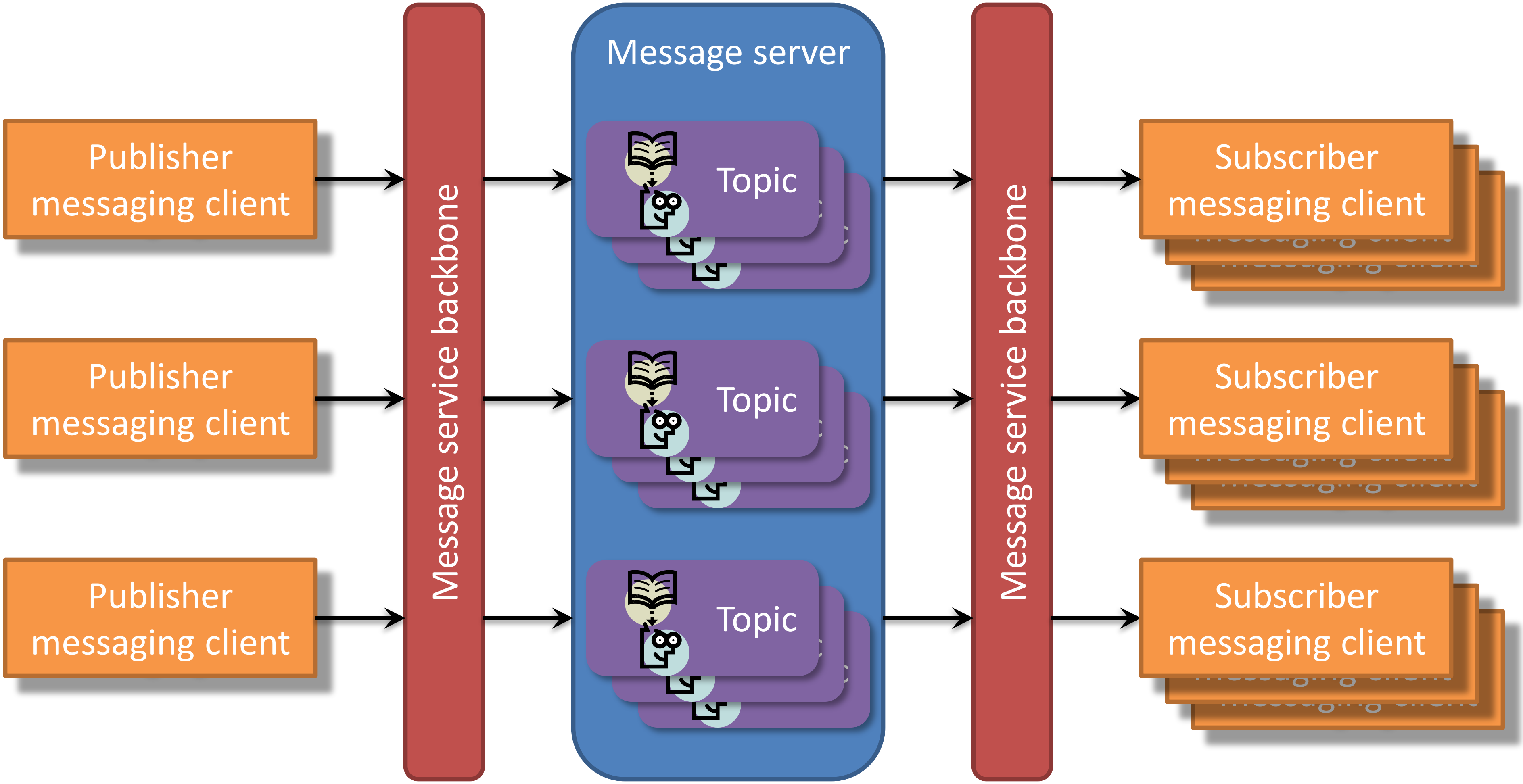
Publish/Subscribe *(1)*

- In this type of asynchronous communication, the application that produces information publishes it and all other applications that need this type of information subscribe to it
- Each application in this scheme may have a dual role – it may act as a publisher or subscriber of different types of information
- It is a slightly more scalable form of messaging, compared to the store and forward mechanism
- The publish/subscribe semantics work as follows:
  - Publishers publish messages to specific topics
  - A message server keeps track of all the messages and all currently active & inactive subscribers; the message server also handles authorization & authentication
  - As soon as messages are published on a specific topic, they are distributed to all of its subscribers; inactive subscribers can retrieve the messages if they come up within a specified time

Publish/Subscribe *(2)*

- The message server takes the responsibility for delivering the published messages to the subscribing applications
- Every message has an expiration time that specifies the maximum amount of time that it can live from the time of its publication
- All subscribers have a *message event listener* that takes delivery of the message from the topic and delivers it to the messaging client application for further processing
- Subscribers can filter the messages they receive by qualifying their subscription with a *message selector*, which evaluates message headers and properties with the provided filter

Publish/Subscribe *(3)*


Publisher messaging client
Publisher messaging client
Publisher messaging client
Message service backbone
Message server
Topic
Topic
Topic
Message service backbone
Subscriber messaging client
Subscriber messaging client
Subscriber messaging client

Event-Driven Processing *(1)*

- Systems often must react to events representing changes in the environment, information of interest, or process status – the familiar one-to-one request/reply interaction is inadequate for such systems
- The asynchrony, heterogeneity, and inherent loose coupling that characterize modern applications in distributed systems promote event interaction as a natural design abstraction for a growing class of software systems
- An *event notification service* complements other general purpose middleware services by offering a many-to-many communication and integration facility
- Two kinds of clients in an event notification scheme: *objects of interest* (i.e., producers of notifications) and *interested parties* (i.e., consumers of notifications); a client can act as both, an object of interest and an interested party

Event-Driven Processing *(2)*

- An event notification service typically realizes the publish/subscribe asynchronous messaging scheme
- Clients may direct the event notification service to perform a *selection process*
  - Apply a *filter* such that it will deliver only notifications that contain certain specified data values
  - Look for *patterns* of multiple events such that it will deliver only sets of notifications associated with the pattern of event occurrences; e.g., a customer might be interested in receiving price change notifications for a certain product if specific suppliers of the product introduce a change in the price of this product simultaneously
- To achieve scalability in a wide area network, the event notification service must be implemented as a distributed network of servers; the service is responsible to route each notification through the network of servers to all subscribers and to keep tract of the identity of the subscriber

Next Lecture: Reading Assignments

- Learn about XML and XML Schema from any source, e.g., books or websites
- If you cannot find any, you may start with w3schools online tutorial
  - http://www.w3schools.com/xml/default.asp
  - http://www.w3schools.com/schema/default.asp