Paradigms in Distributed Computing

Grado en Ingeniería Informática

Pablo García Sánchez

Departamento de Ingeniería Informática
Universidad de Cádiz
Pablo García Sánchez
Based on the work by Guadalupe Ortiz, Mei Ling-Liu, Marteen Van Steem and A. Tanenbaum

Curso 2017 – 2018

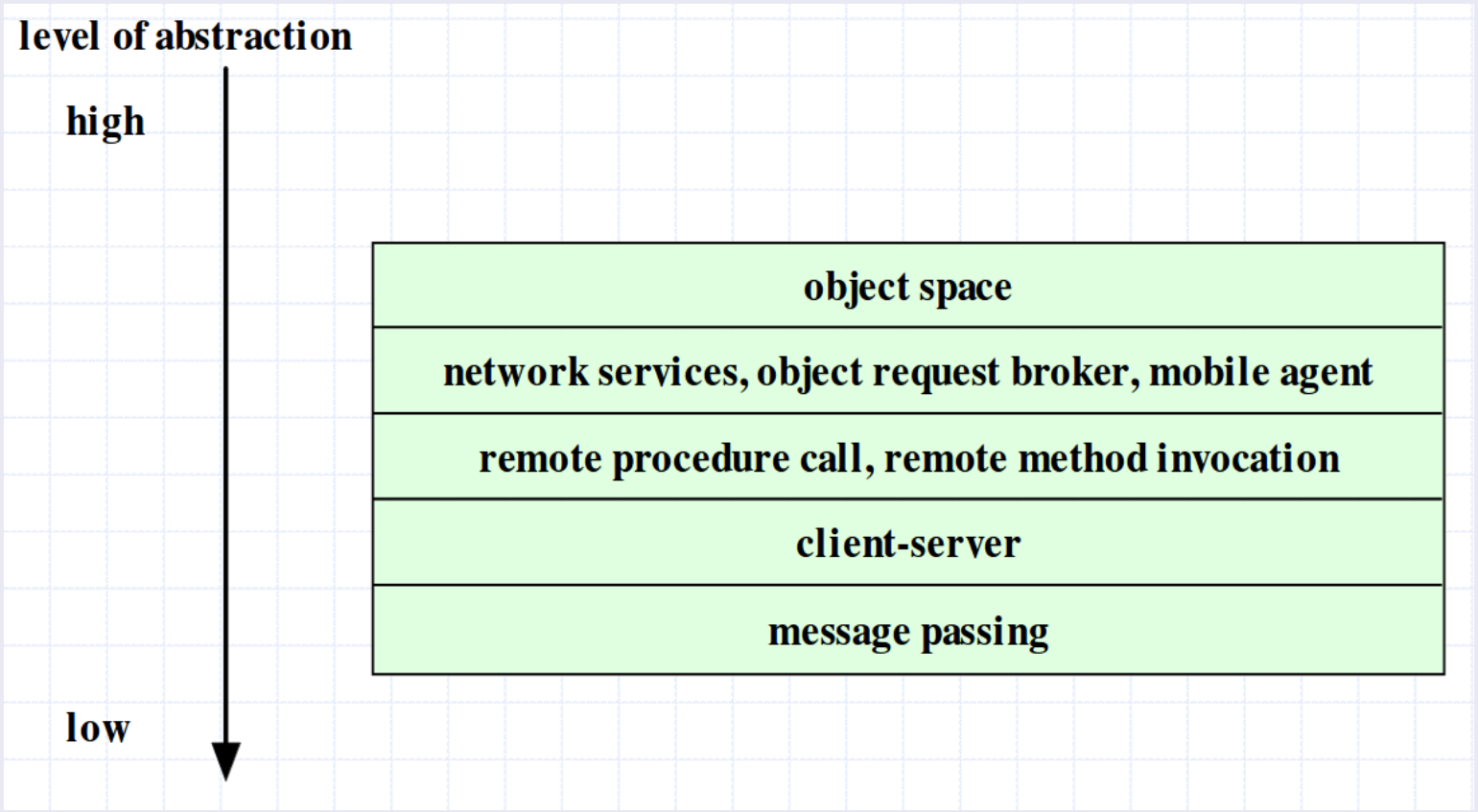
Index

Section 1 | Introduction

Paradigm

- Definition of paradigm: a pattern, example or model.
- Characteristics of distributed applications vs. conventional applications (one machine)
  - Interprocess Communication
  - Event Synchronization
- Abstraction in engineering is realized with the provision of tools or facilities which allow software to be built without the developer having to be cognizant of some of the underlying complexities.

Distributed Application Paradigms

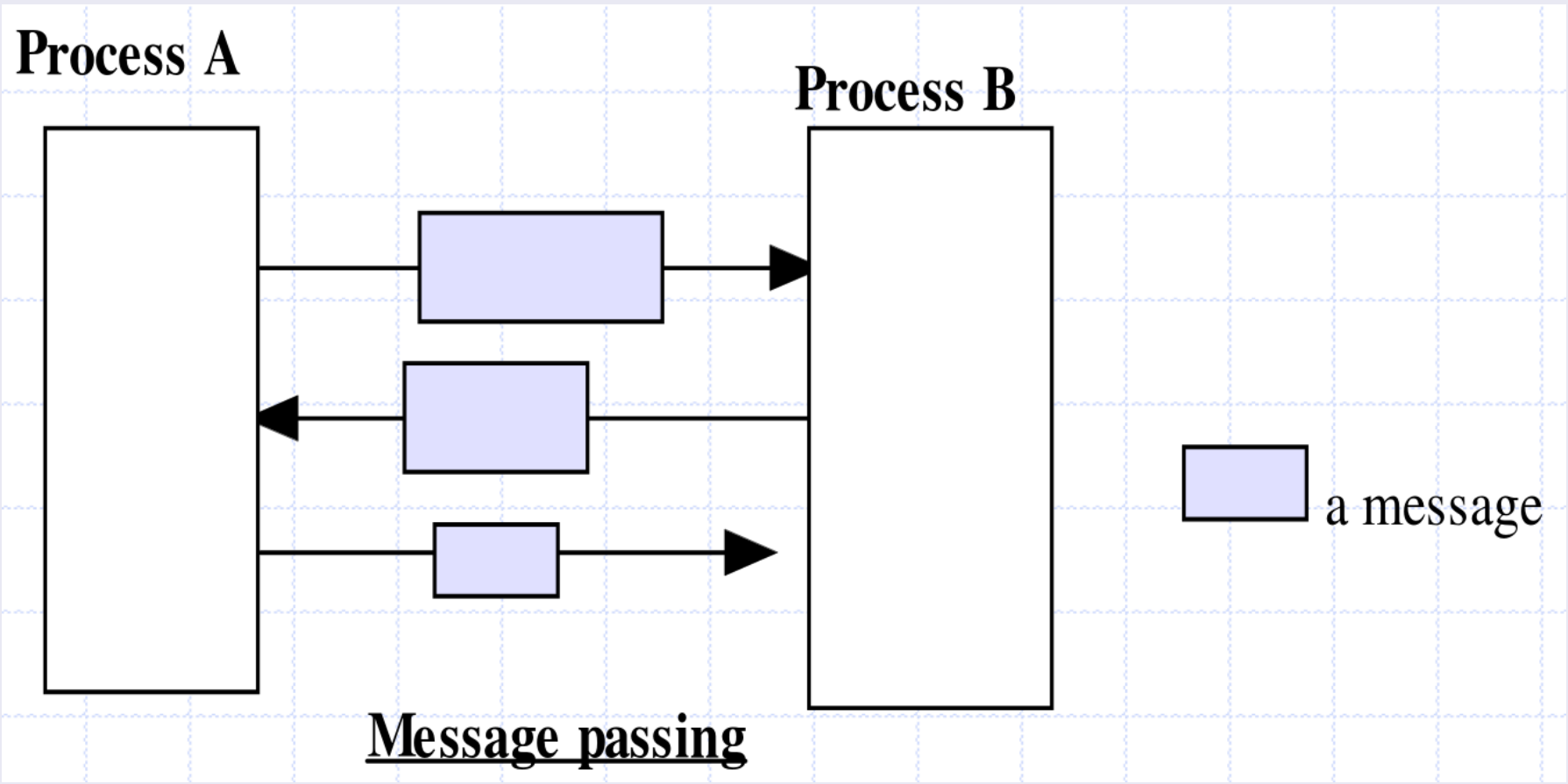


Section 2 | Message Passing Paradigm

Message Passing Paradigm

- The most fundamental paradigm for distributed applications.
- A process sends a message representing a request.
- The message is delivered to a receiver, which processes the request, and sends a message in response.
- The reply may trigger a further request, which leads to a subsequent reply, and so forth.

Message Passing



Message Passing Paradigm

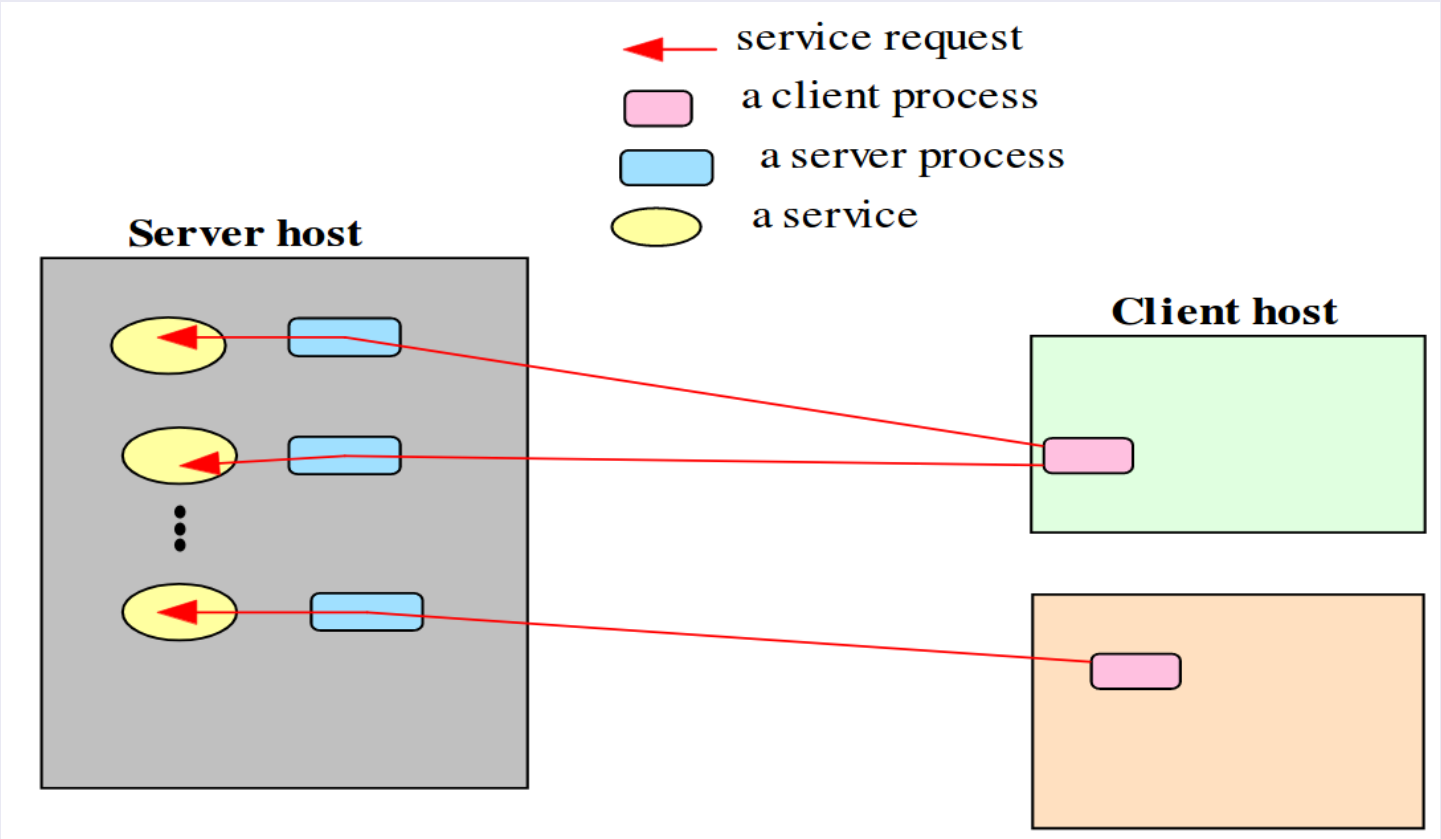
- Basic operations required to support the basic message passing paradigm are send, and receive.
- For connection-oriented communication, the operations connect and disconnect are also required.
- With the abstraction provided by this model, the interconnected processes perform input and output to each other, in a manner similar to file I/O. The I/O operations encapsulate the detail of network communication at the operating-system level.
- The socket application programming interface is based on this paradigm, as we know :)

Section 3 | Client-Server paradigm

Introduction to Client-Server

- The most prevalent model for distributed computing protocols.
- It is the basis of all distributed computing paradigms at a higher level of abstraction.
- It is service-oriented, and employs a request-response protocol.

Client-Server



Client-Server

- A server process, running on a server host, provides access to a service.
- A client process, running on a client host, accesses the service via the server process.
- The interaction of the process proceeds according to a protocol.

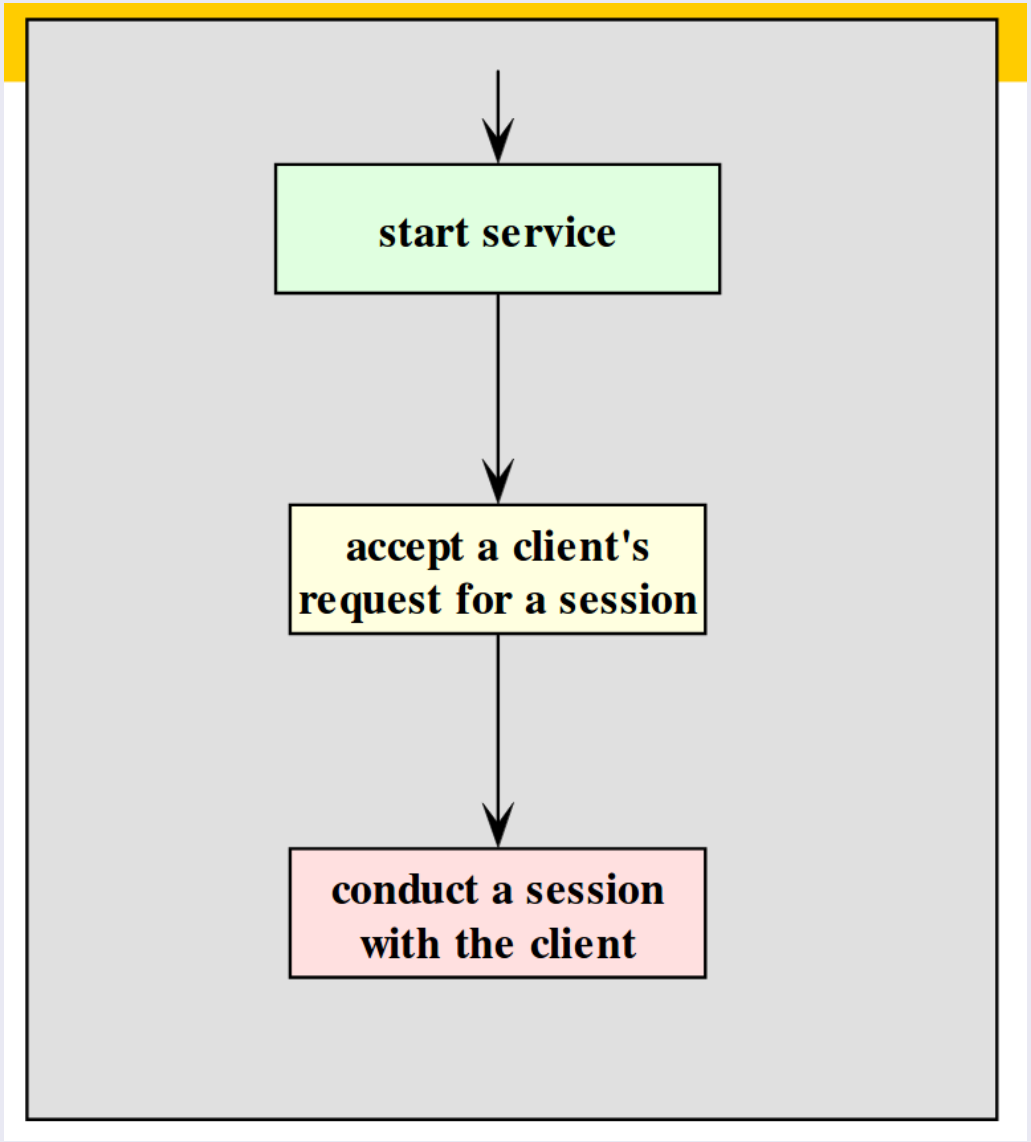
Client-Server: applications and services

- An application based on the client-server paradigm is a client-server application.
- On the Internet, many services are Client-server applications. These services are often known by the protocol that the application implements.
- Well known Internet services include HTTP, FTP, DNS, finger, gopher, etc.
- User applications may also be built using the client-server paradigm.

Client-Server: Session

- Session: the interaction between the server and one client.
- The service managed by a server may be accessed by multiple clients who desire the service, sometimes concurrently.
- Each client, when serviced by the server, engages in a separate session with the server, during which it conducts a dialog with the server until the client has obtained the service it required

Client-Server



Client-Server: Protocols

- A protocol is needed to specify the rules that must be observed by the client and the server during the conduction of a service.
- Such rules include specifications on matters such as
  - how the service is to be located
  - the sequence of interprocess communication
  - the representation and interpretation of data exchanged with each IPC
- On the Internet, such protocols are specified in the RFCs

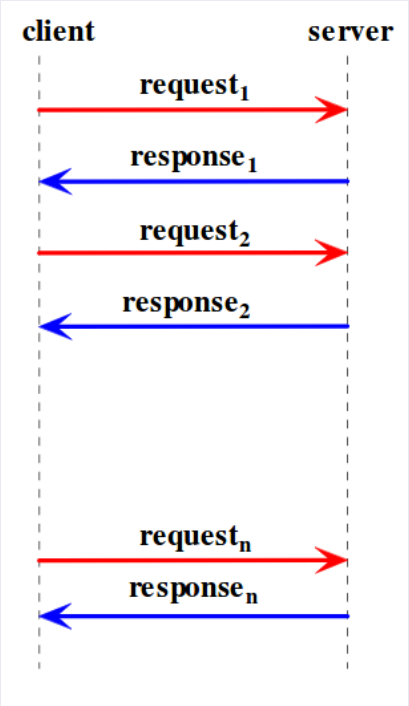
Client-Server: Locating a Service

- A mechanism must be available to allow a client process to locate a server for a given service.
- For example, in Internet services: address of the server process/hostname + protocol port number assigned.
- Well-known services such as FTP, HTTP, or telnet are assigned to a default port number reserved.
- Higher level of abstraction: service identified using a logical name registered with a registry.
- Logical name mapped to the physical location.
- If mapped at runtime (when a client process is run), then it is possible for the service's location to be dynamic, or moveable

Client-Server: Implementing a service

- Implementations needs to adhere to the specification for the protocol, including how the dialogs of each session should proceed. The specification defines:
  - which side (client or server) should speak first
  - the syntax and semantic of each request and response
  - the action expected of each side upon receiving a particular request or response

Client-Server: synchronization



Session IPC examples

Daytime service [RFC867]

- Client: Hello, <client address> here. May I have a timestamp please.
- Server: Here it is: (time stamp follows)

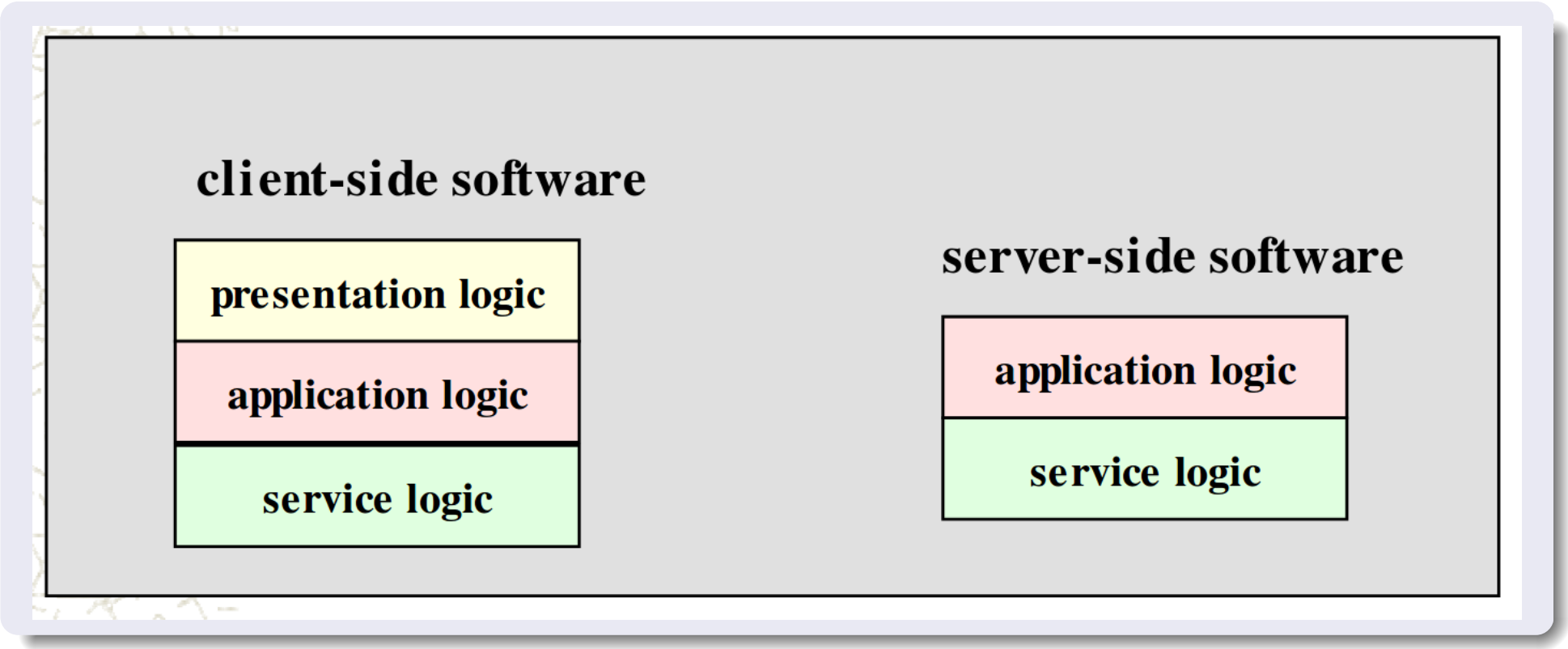
HTTP [RFC2616]

- Client: Hello, <client address> here.
- Server: Okay. I am a web server and speaks protocol HTTP1.0
- Client: Great, please get me the web page index.html at the root of your document tree
- Server: Ok, here is the web page (content follows)

Client-Server: Protocol Data Representation

- Part of the specification of a protocol is the synt ax and semantics of each request and response.
- Depends on the nature and the needs of the protocol.
- Representing data using text (character strings) is common, as it facilitates data marshalling and allows the data to be readable by human.
- Most well known Internet protocols are client- server, request-response, and text-based.

Client-Server: software engineering



Client server: advantages of separating the layers of logic

- It allows each module to be developed by people wi th special skills to focus on a module for which they have exp ertise.
  - UX software engineers may concentrate on developing the modules for the presentation logic
  - Other developers specialized in application logic and the service logic may focus on developing the other modules
- The separation allows modifications to be made to the logic at the presentation without requiring changes to be ma de at the lower layers.
- For example, the user interface can be changed from text-mode to graphical without requiring any change be made to the application logic or the service logic.

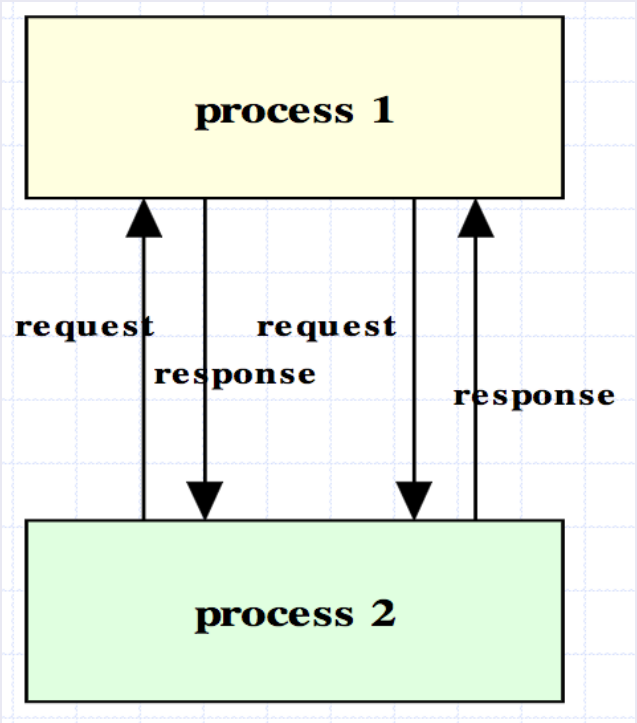
Section 4 | Peer-to-peer

The P2P System architecture

- An architecture where computer resources and services are direct exchanged between computer systems.
- Exchange of information, processing cycles, cache storage, and disk storage for files..
- Computers that have traditionally been used solely as clients communicate directly among themselves and can act as both clients and servers, assuming whatever role is most efficient for the network.

P2P roles

- The participant processes play equal roles, with equivalent capabilities and responsibilities (hence the term “peer”).
- Each participant may issue a request to another participant and receive a response



P2P usage and examples

- Client-server paradigm is an ideal model for a centralized network service,
- But Peer-to-peer paradigm is more appropriate for applications such as instant messaging, peer-to-peer file transfers, video conferencing, and collaborative work.
- It is also possible for an application to be based on both the client-server model and the peer-to-peer model
- Examples: Napster, GNutella, BitTorrent...

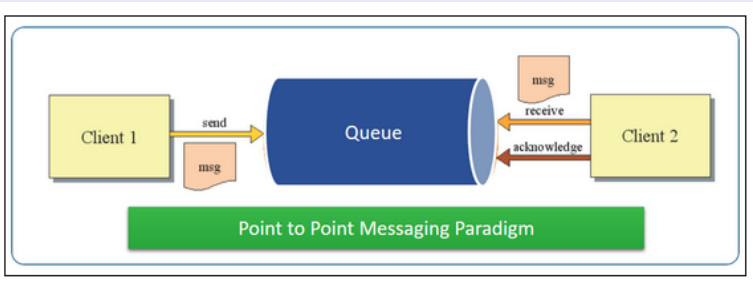
Section 5 | Message System Paradigm

Message System Paradigm

- It is an elaboration of the basic message-passing paradigm.
- A message system serves as an intermediary among separate, independent processes.
- The message system acts as a switch for messages, through which processes exchange messages asynchronously, in a decoupled manner.
- A sender deposits a message with the message system, which forwards it to a message queue associated with each receiver.
- Once a me ssage is sent, the sender is free to move on to other tasks.
- Two subtypes:
  - Point-to-point
  - Publish/Subscribe

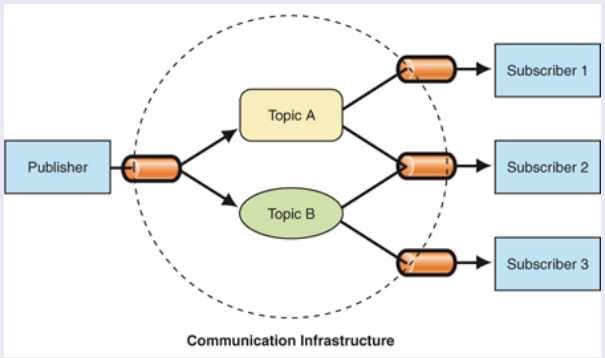
Point to point

- A message system forwards a message from the sender to the receiver's message queue.
- Unlike the basic message passing model, the middleware provides:
  - A message depository , that allows the sending and the receiving to be decoupled.
  - Additional abstraction for asynchronous operati ons. Only using message-passing: a developer will have to make use of threads or child processes.
- Via the middleware, a sender deposits a message in the message queue of the receiving process.
- A receiving process extracts the messages from its message queue, and handles each one accordingly.



Publish/Subscribe Message model

- Each message is associated with a specific topic or event.
- Applications interested in the occurrence of a specific event may subscribe to messages for that event.
- When the awaited event occurs, the process publishes a message announcing the event or topic.
- The middleware message system distributes the message to all its subscribers.
- The publish/subscribe message model offers a powerful abstraction for multicasting or group communication.



Section 6 | Remote Procedure Call

Remote Procedure Call

- A paradigm which allows distributed software to be programmed in a manner similar to conventional applications which run on a single processor.
- A remote procedure call involves tw o independent processes, which may reside on separate machines.
  - A process, A, wishing to make a request to another process, B, issues a procedure call to B, passing with the call a list of argument values.
  - As in the case of local procedure calls, a remote procedure call triggers a predefined action in a procedure provided by process B.
  - At the completion of the procedure, process B returns a value to process A.
- Tool: *rpcgen*

Section 7 | Distributed Objects

Distributed objects paradigms

- Applications access objects distributed over a network.
- Objects provide methods, through the invocation of which an application obtains access to services.

Object-oriented paradigms

- Remote method invocation (RMI): object in a remote host
- Network services: using a service directory (example: Jini)
- Object request broker: CORBA
- Object spaces: publish/subscribe objects in a shared space

References

- Distributed Systems. Marteen Van Steen and Andrew Tanenbaum (2017).
- Mei Ling-Liu. Distributed Computing Algorithms.
- Classroom notes by Guadalupe Ortiz