Lecture#6 –Communication in DS

S. M. Ahsan Kazmi

Outline

- Communication in DS
  - Introduction
  - Types of communication
  - Communication Models
  - Remote procedure calls

Recap

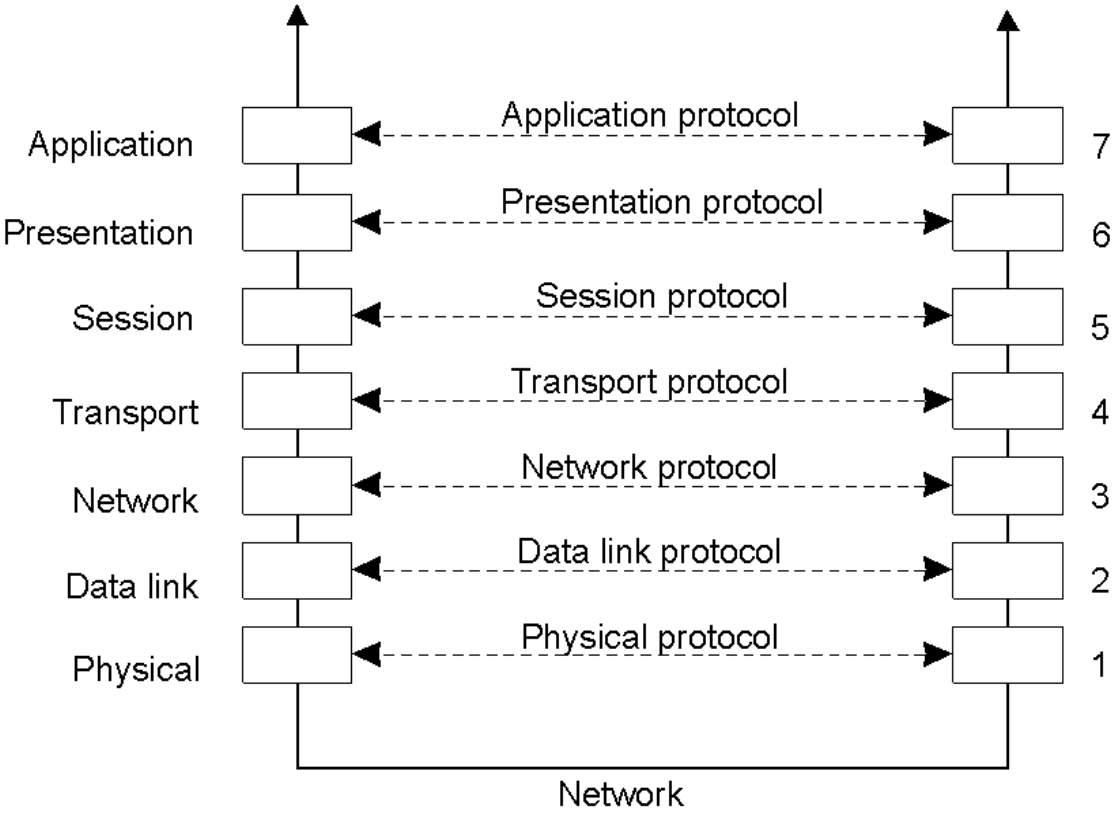
- OS-Level Virtualization
  - Comparison with different types of virtualization
  - Containers Building Blocks
  - Containers Orchestration
- Memory Technologies and Hierarchies
  - Virtualization- Memory Basics and Challenges
  - Memory reclamation approaches
- Data Center and Cloud
  - Inside a data center
  - Data center economics
  - Challenges
  - Service and Deployment models

Communication Between Processes

- Unstructured communication
  - Use shared memory or shared data structures
- Structured communication
  - Use explicit messages (IPCs)
- Distributed Systems: both need low-level communication support

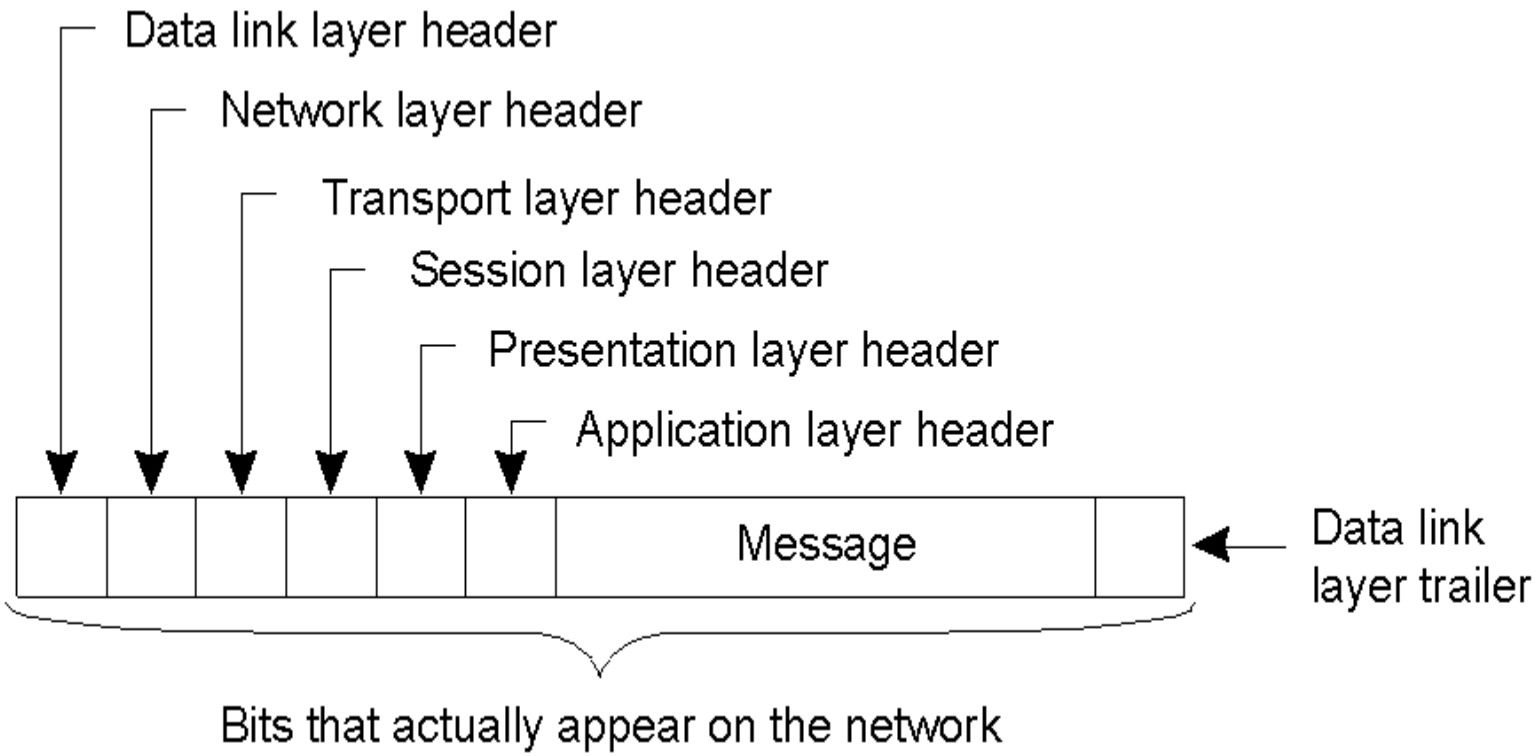
Layered Protocols

- **OSI**: Open Systems Interconnection
- Developed by the International Organization for Standardization (ISO)
- Provides a generic framework to discuss the layers and functionalities of communication protocols.



Layers, interfaces, and protocols in the OSI model.

OSI Model (con.t)



A typical message as it appears on the network.

OSI Protocol Layers

- **Physical layer**
  - Deals with the transmission of bits
  - Physical interface between data transmission device
  - (e.g. computer) and transmission medium or network
  - **Concerned with:**
    - Characteristics of transmission medium, Signal levels, Data rates

- **Data link layer:**
  - Deals with detecting and correcting bit transmission errors
  - Bits are group into frames
  - Checksums are used for integrity

OSI Protocol Layers (con.t)

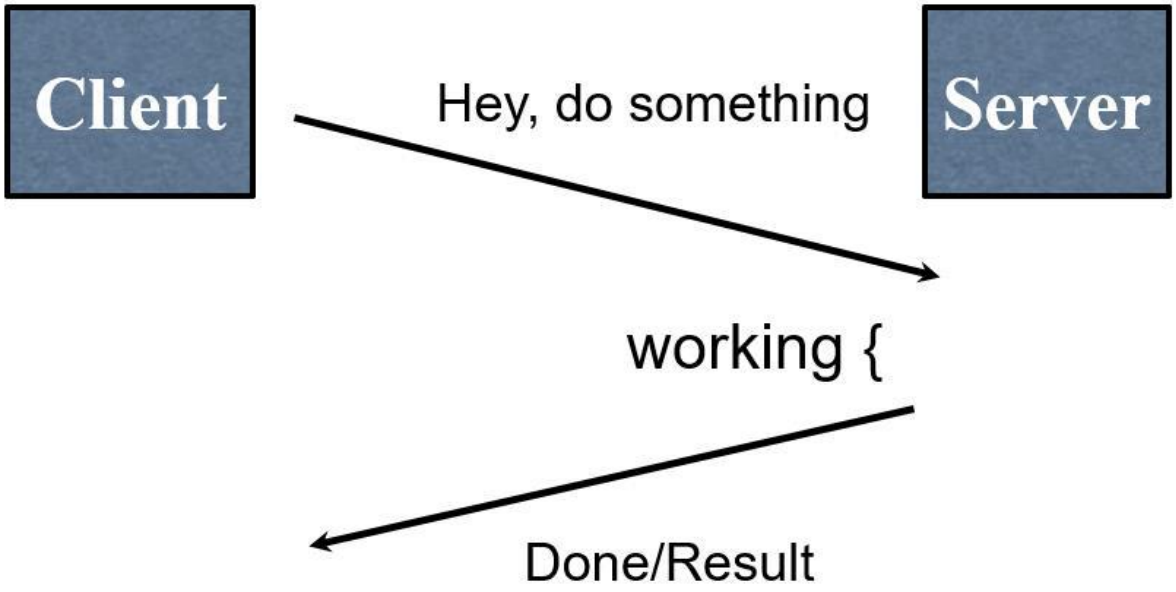
- **Network layer:**
  - Performs multi-hop routing across multiple networks
  - Implemented in end systems and routers

- **Transport layer:**
  - Packing of data
  - Reliable delivery of data (breaks message into pieces small enough, assign each one a sequence number and then send them)
  - Ordering of delivery
  - **Examples:**
    - TCP (connection-oriented)
    - UDP (connectionless)

OSI Protocol Layers (con.t)

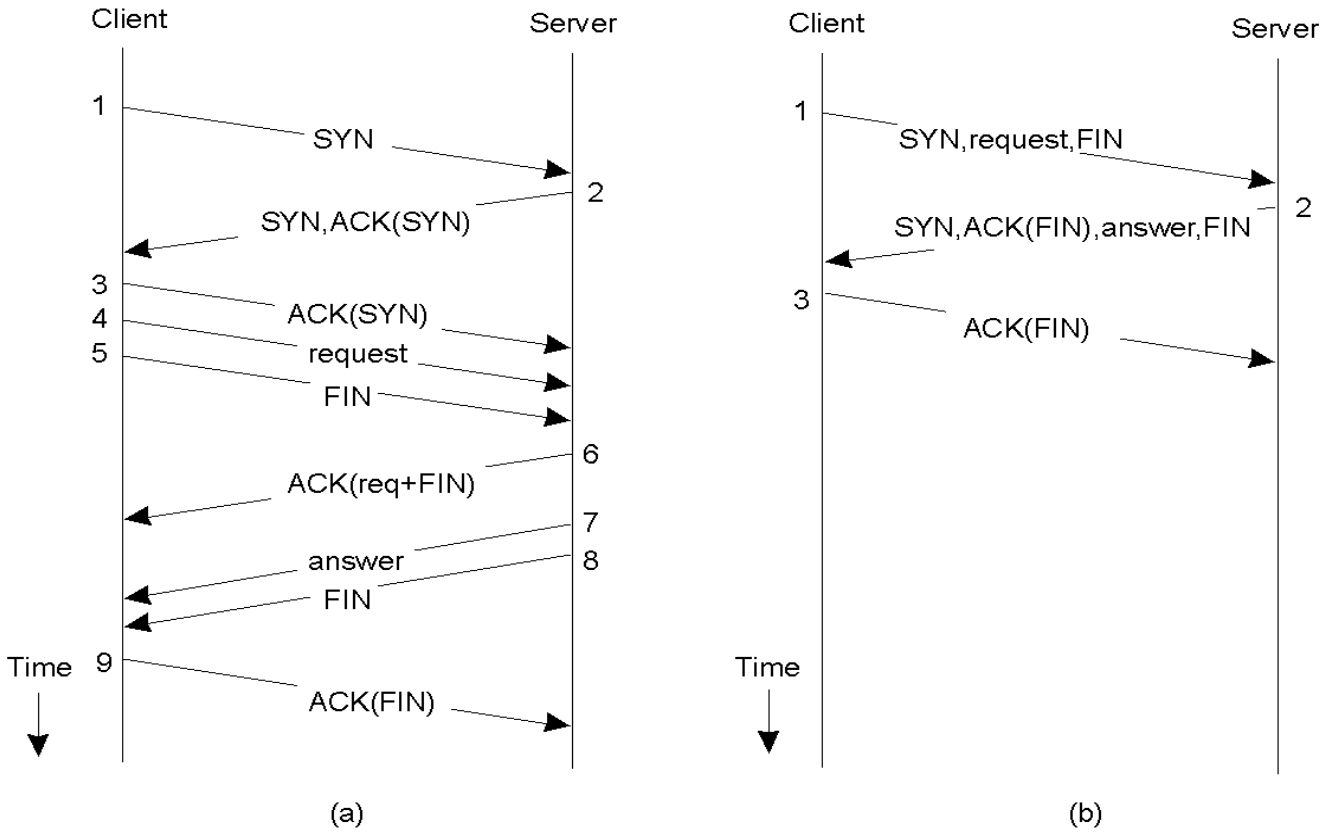
- **Session layer**
  - Provide dialog control to keep track of which party is talking and it provides synchronization facilities
  -
- **Presentation layer**
  - Deals with non-uniform data representation and with compression and encryption
- **Application layer**
  - Support for user applications
  - e.g. HTTP, SMPT, FTP

Typical Communication Pattern



OSI Protocol Layers (con.t)

Client-Server TCP protocol



(a) Normal operation of TCP. (b) Transactional TCP.

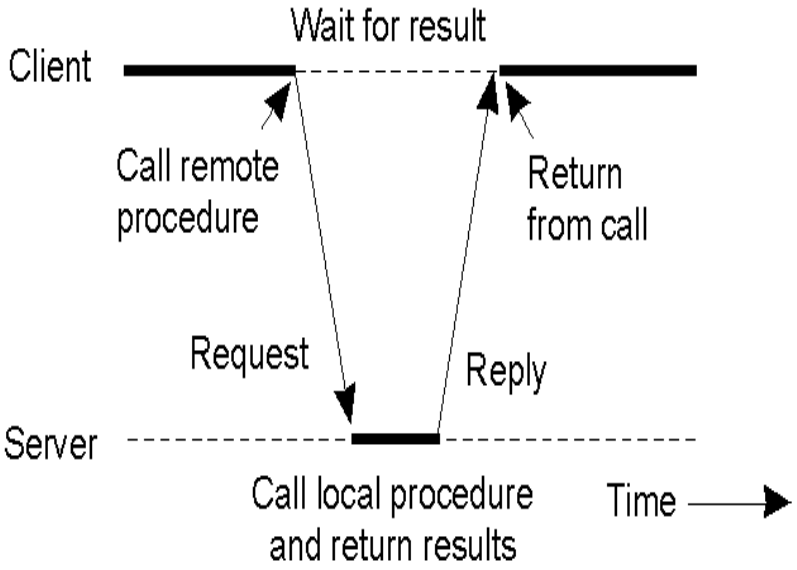
Communication Models: Push or Pull ?

- Client-pull architecture
  - Clients pull data from servers (by sending requests)
  - Example: HTTP

- Server-push architecture
  - Servers push data to client
  - Example: video streaming

Remote Procedure call

- **Basic idea:** To execute a procedure at a remote site and ship the results back.
- **Goal**: To make this operation as distribution transparent as possible (i.e., the remote procedure call should look like a local one to the calling procedure).



RPC Goals

- A remote procedure call makes a call to a remote service look like a local call
  - RPC makes transparent whether the server is local or remote
  - RPC makes the architecture of remote machines transparent

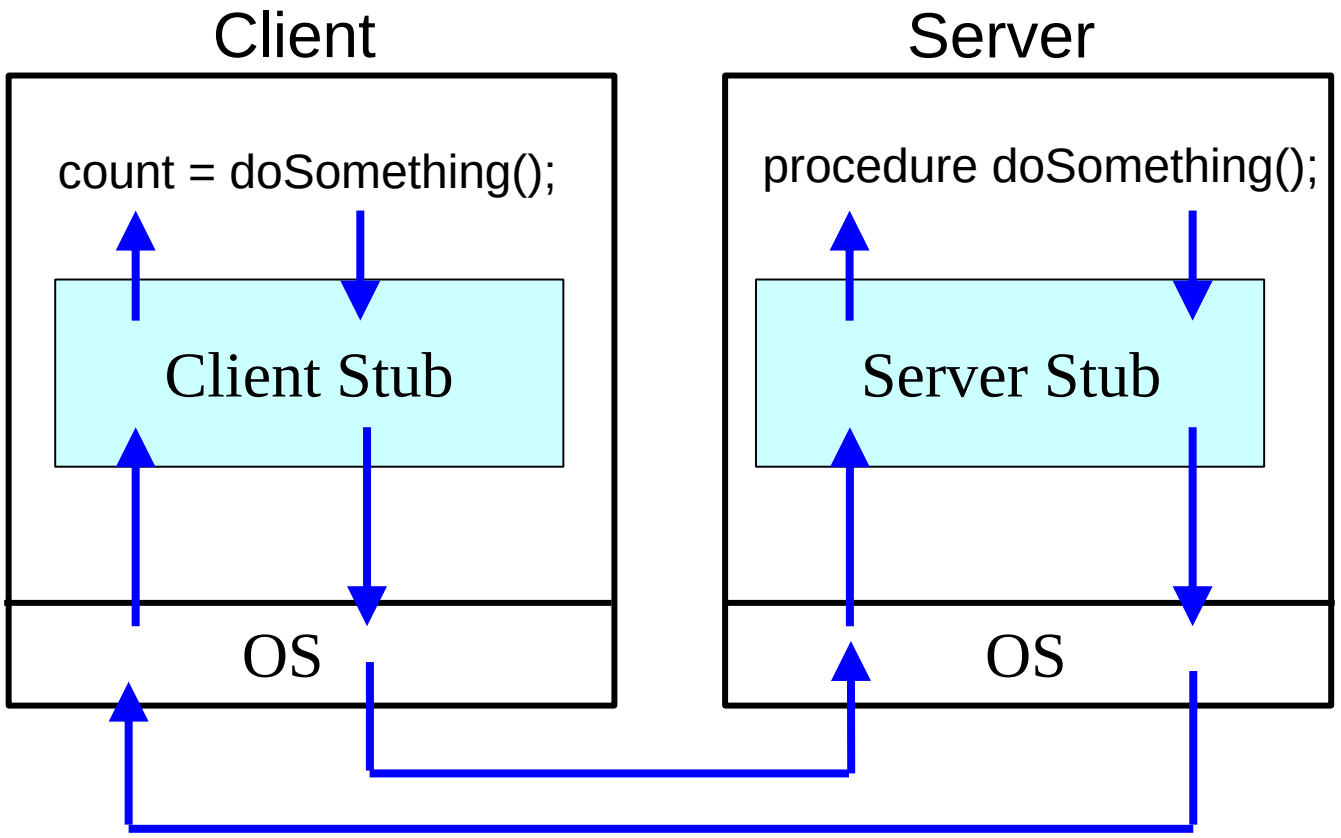
- Benefits
  - Ease of programming
  - Hide complexity
  - Automates task of implementing distributed computation
  - Familiar model for programmers (just make a function call)

Parameter Passing

- Local procedure parameter passing
  - Call-by-value
  - Call-by-reference: arrays, complex data structures
- Remote procedure calls simulate this through:
  - Stubs
  - Marshalling
- Related issue: global variables are not allowed in RPCs

Client and Server Stubs

Stubs are additional functions that are added to the main functions in order to support for RPC



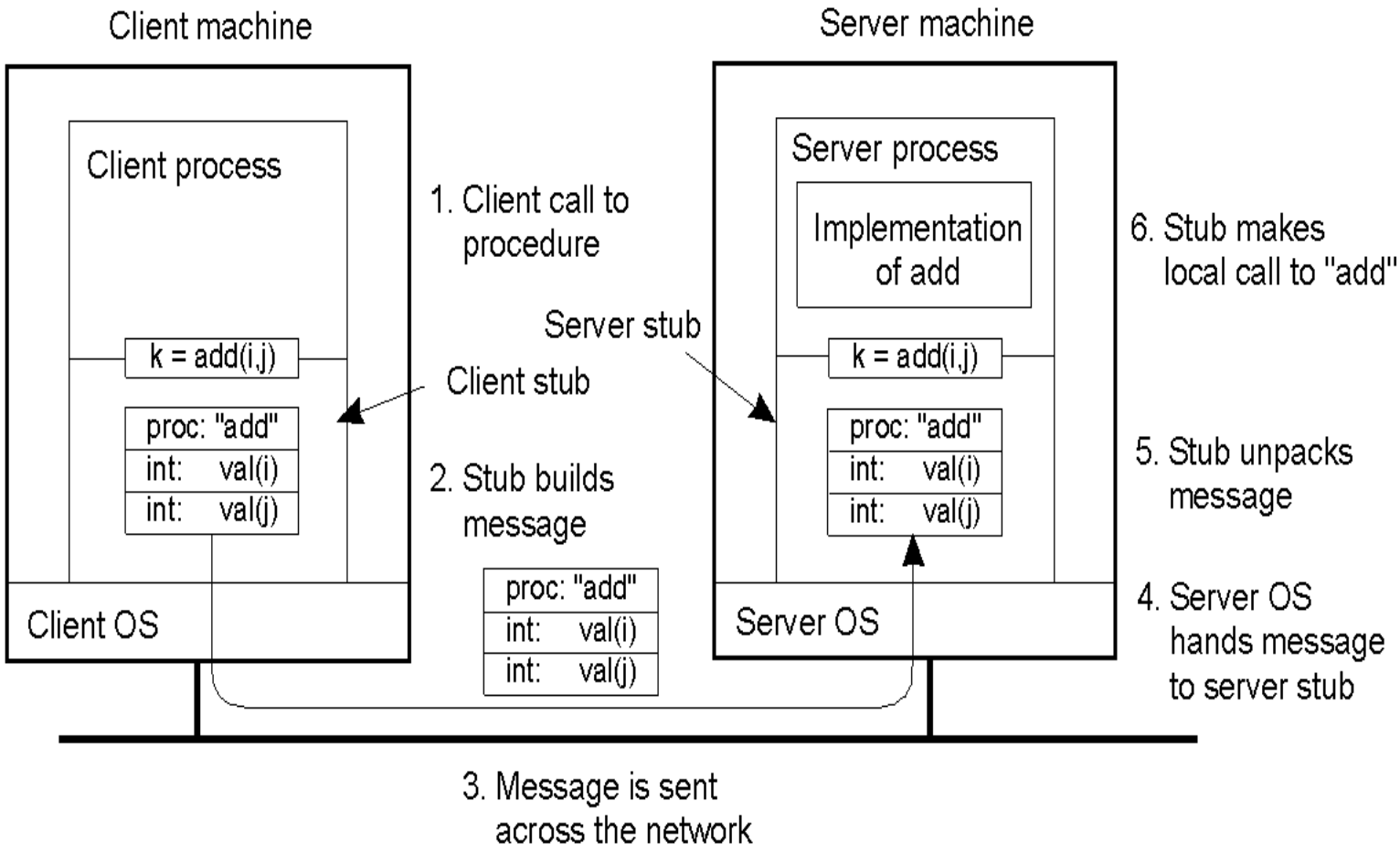
Client and Server Stubs

- Client makes procedure call (just like a local procedure call) to the client stub
- Server is written as a standard procedure
- Stubs take care of packaging arguments and sending messages
- Packaging parameters are called marshaling
- Stub compiler generates stub automatically from specs in an Interface Definition Language (IDL)
  - Simplifies programmer task

Steps of a Remote Procedure Call

1. Client procedure calls client stub in a normal way
2. Client stub builds message, calls local OS
3. Client's OS sends a message to the remote OS
4. Remote OS gives a message to the server stub
5. Server stub unpacks parameters, calls server
6. Server does work, returns result to the stub
7. Server stub packs it in a message, calls local OS
8. Server's OS sends a message to the client's OS
9. Client's OS gives a message to the client stub
10. Stub unpacks result, returns to client

Example



Steps involved in doing remote computation through RPC

RPC Challenges

- But it's not always simple
  - Calling and called procedures run on different machines, with different address spaces
  - And perhaps different environments or operating systems
  - Must convert to a local representation of data
  - Machines and networks can fail

Marshalling

- Problem: different machines have different data formats
  - Intel: little endian, SPARC: big endian
- Solution: use a standard representation
  - Example: external data representation (XDR)
- Problem: how do we pass pointers?
  - If it points to a well-defined data structure, pass a copy and the server stub passes a pointer to the local copy
- What about data structures containing pointers?
  - Prohibit
  - Chase pointers over the network
- Marshalling: transform parameters/results into a byte stream

Binding

- Problem: how does a client locate a server?
  - Use Bindings

- Server
  - Export server interface during initialization
  - Send name, version no, unique identifier, and handle (address) to binder

- Client
  - First RPC: send message to the binder to import server interface
  - Binder: check to see if a server has exported the interface
  - Return handle and unique identifier to client

Binding: Comments

- Exporting and importing incur overheads
- Binder can be a bottleneck
  - Use multiple binders
- Binder can do load balancing

RPC Implementation Failure Semantics

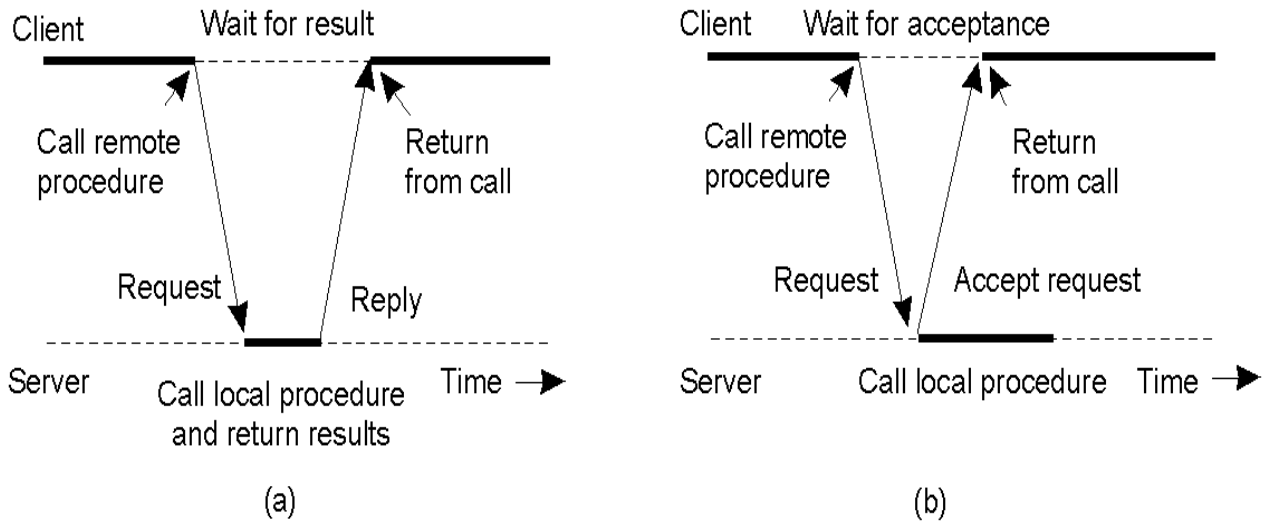
- Client unable to locate server: return error
- Lost request messages: simple timeout mechanisms
- Lost replies: timeout mechanisms
  - Make operation idempotent
  - Use sequence numbers, mark retransmissions
- Server failures: did the failure occur before or after an operation?
  - At most once
  - No guarantee

Failure Semantics

- Client failure: what happens to the server computation?
  - Referred to as an orphan
  - Extermination: log at client stub and explicitly kill orphans
    - Overhead of maintaining disk logs
  - Reincarnation: Divide time into epochs between failures and delete computations from old epochs
  - Expiration: give each RPC a fixed quantum T; explicitly request extensions
  - Periodic checks with clients during long computations

Asynchronous RPC

- Avoids blocking of the client process.
- Allows the client to proceed without getting the result of the call.



a) The interconnection between client and server in a traditional RPC

b) The interaction using asynchronous RPC

References

- Above the Clouds: A Berkeley View of Cloud Computing, https://www2.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf

- Andrew Birrell and Bruce Nelson, Implementing RPCs, ACM Transactions on Computer Systems, Vol. 2, No. 1, Pages 39-59, February 1984.

- B. Bershad, T. Anderson, E. Lazowska, and H. Levy, Lightweight Remote Procedure Call, Proceedings of the 12th ACM Symposium on Operating Systems Principles, Operating Systems Review, Vol. 23, No. 5, Pages 12-113, December 1989

- Distributed Systems: Principles and Paradigms by Tanenbaum and van Steen, Chapter 4.1~4.2.