THE UNIVERSITY of EDINBURGH

Distributed Systems
Fall 2025

Yuvraj Patel

Scan to check-in

Disclaimer: Slides prepared using multiple sources (UW-Madison – Remzi, Andrea, Mike; Cambridge – Martin Kleepman; Distributed Systems book by MVS/ AST); University of Edinburgh – Yuvraj Patel

Today’s Agenda

Communication

- Fundamentals
- Remote Procedure Calls

Computation vs. Communication

Processes/Threads/VMs/Nodes perform computation

They alone cannot comprise the Distributed System

The interaction between the computational components make any system a distributed system

- Like human beings and society

Some methodology needed to let the computational components interact

Communication

Communication not a prerogative for distributed systems only

- Single node/process can communicate using function calls, IPC, etc.

Communication paradigms describe and classify a set of methods by which computational nodes can interact and exchange data

Communication involves many problems/issues

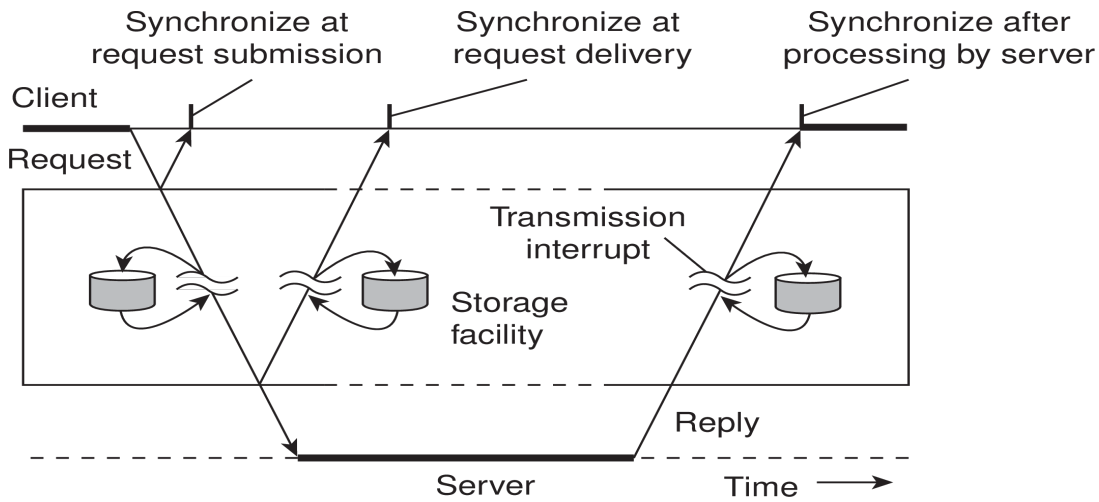
- Physical transmission to application level
- Need to standardize to make things easy

Types of Communication

Persistent vs Transient

Persistent → Message sent is stored by the middleware until it is delivered to the receiver; Example – Email server

Transient → Message sent is stored by the middleware only as long as both the receiver and sender executing; Example – RPC

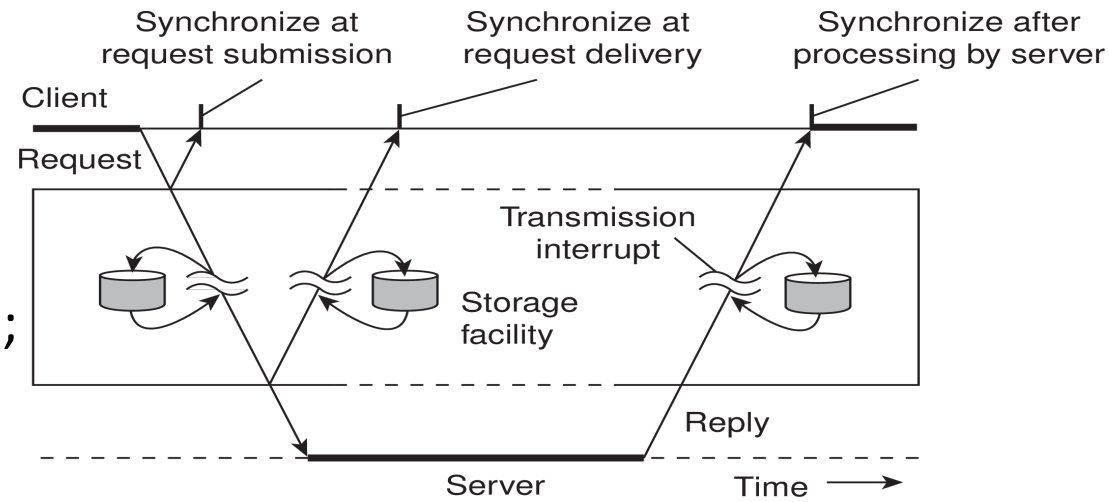


Types of Communication (contd…)

Asynchronous vs. Synchronous

Asynchronous → Sender keeps on executing after sending a message

Synchronous → Sender blocks execution after sending a message and waits for response; 3 levels of responses

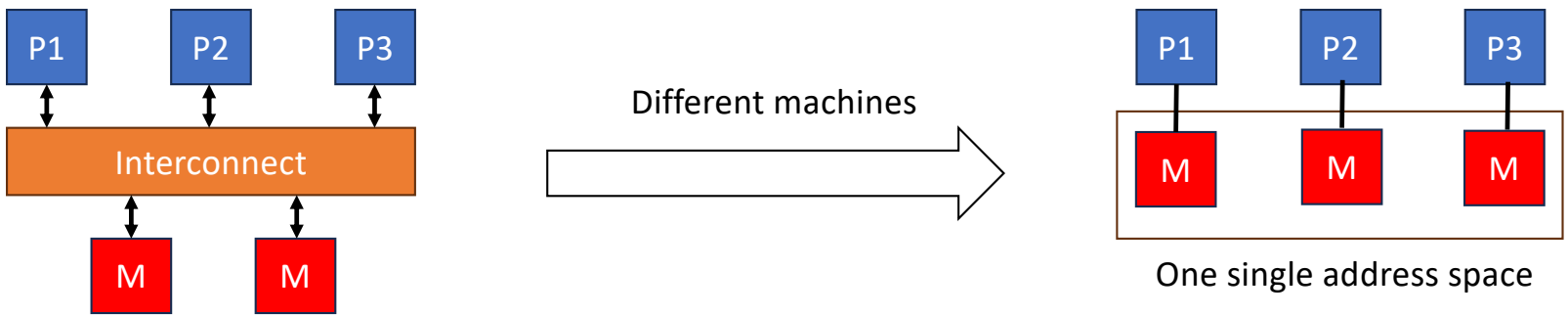


Classification of Communication Paradigms

Three categories

- Same address space – Global Variables, Procedure calls
- Different address spaces (Within a computer) – Files, Shared Memory, Signals
- Different address space (Multiple computers) – Shared Memory, Message Passing – RPC, sockets

Distributed Shared Memory



Message Passing

Assume no explicit sharing of data elements in the address space of computational components

Essence of message passing is copying

- Implementation may avoid copying wherever possible

Problem-solving with messages – more active involvement by participants

Send and Receive two main primitives

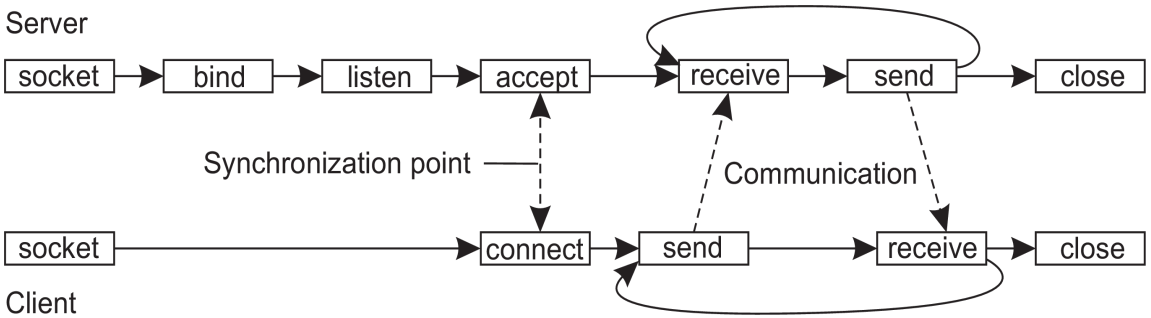
Client<—>Server interaction

Socket Programming

Socket – Software structure that serves as endpoint for sending and receiving data across the network

Several APIs to interact with sockets

Operation	Description
socket	Create a new communication end point
bind	Attach a local address to a socket
listen	Tell OS what is the maximum number of pending connection requests should be
accept	Block caller until a connection request arrives
connect	Actively attempt to establish a connection
send	Send some data over the connection
receive	Receive some data over the connection
close	Release the connection



Socket Code in Python

Server

```
from socket  import *

class Server:
  def run(self):
    s  = socket(AF_INET, SOCK_STREAM)
    s.bind((HOST,  PORT))
    s.listen(1)
    (conn, addr) = s.accept()  # returns new socket and addr. client
    while  True:                # forever
      data  = conn.recv(1024)   # receive data from client
      if not data: break        # stop if client stopped
      conn.send(data+b"*")      # return sent data plus an "*"
    conn.close()                # close the connection
```

Client

```
class Client:
  def run(self):
    s  = socket(AF_INET, SOCK_STREAM)
    s.connect((HOST, PORT)) # connect to server (block until accepted)
    s.send(b"Hello, world") # send same data
    data = s.recv(1024)     # receive the response
    print(data)             # print what you received
    s.send(b"")             # tell the server to close
    s.close()               # close the connection
```

Remote Procedure Call

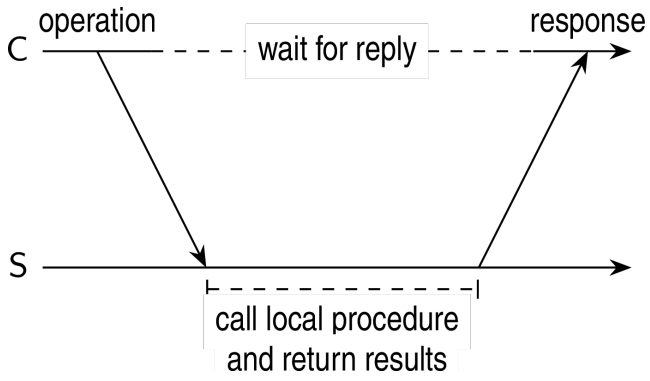
Allow remote services to be called as procedures

- Transparency with respect to location, implementation, language, etc.

Goal is to make distributed computing look like centralized computing

Basic Idea

- Programs can call procedures on other machines
- When process A calls a procedure foo() on machine B, A is suspended
- Execution of foo() takes place on machine B
- After execution of foo(), the result is sent back to A, which resumes execution



Procedure Call to Remote Procedure Call

Procedure Call to Remote Procedure Call

Procedure Call to Remote Procedure Call

Procedure Call to Remote Procedure Call

RPC – Challenges

Separate callee and caller address space

- How to transfer data?
- Need for a common reference space

Machines may be different

- Parameters and results must be passed and handled correctly

Thousands of procedures exported by servers

- How does client locate a server?

Client and server might fail independently

- How to handle failures?

Parameter Passing

Marshalling/Packing – Parameters passed into a message to be transmitted

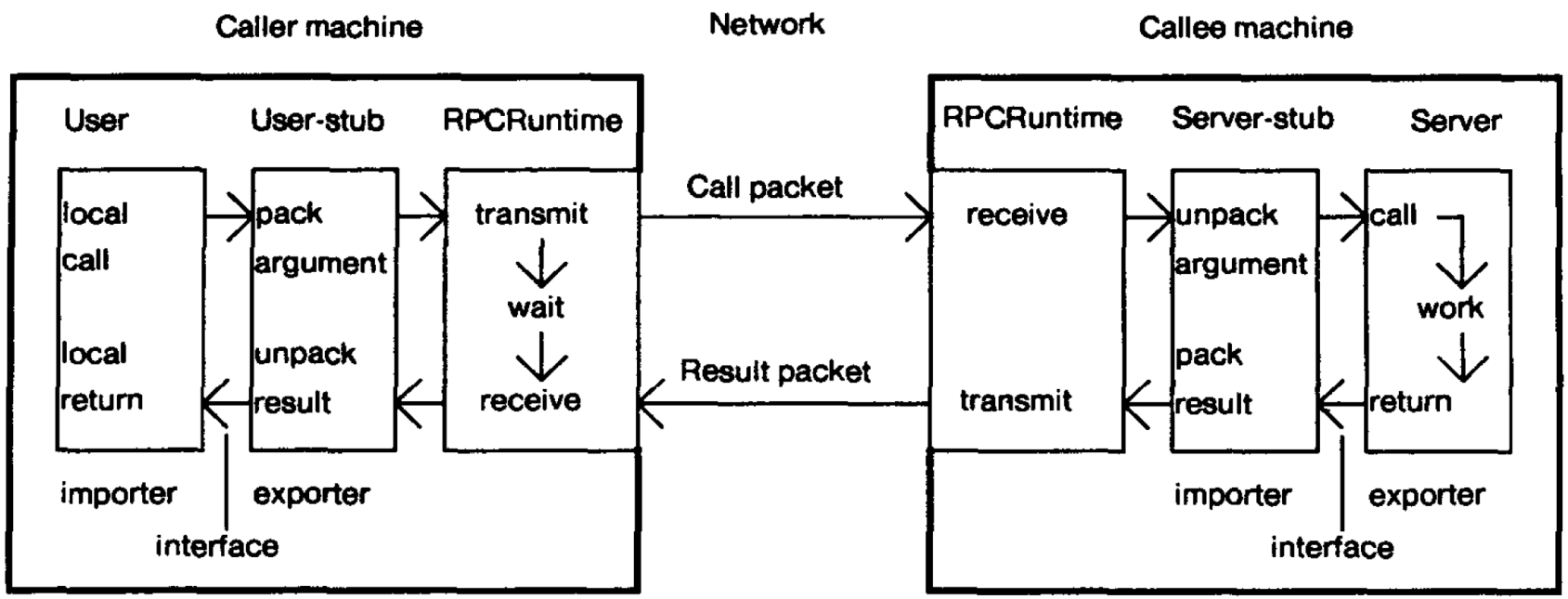
Both parameters and results must be marshalled

Two types of parameters

- Value – directly encoded into the message
- Reference – Can lead to incorrect results (or crash); Solutions???

Client and Server stub takes care of marshalling

Parameter Passing (contd…)



Data Representation

Different micro-architecture and OS

- Size of data-type differs – size of long in 32-bit vs. 64-bit machines
- Format in which data is stored – Little-endian vs. Big-endian

Client and server must agree on how simple data is represented in the message

- Rely on Interface Definition Language (IDL) for the specification
- Stub compiler generates stub automatically from the specification

Binding

Binder

- Use bindings to let clients locate a server

Server

- Export server interfaces during initialization
- Send name, version number, unique identifier, handle to a binder

Client

- Send message to binder to import server interface
- Binder will check to see if a server has exported valid interface
- Return handle and unique identified to client

Binding may incur overhead

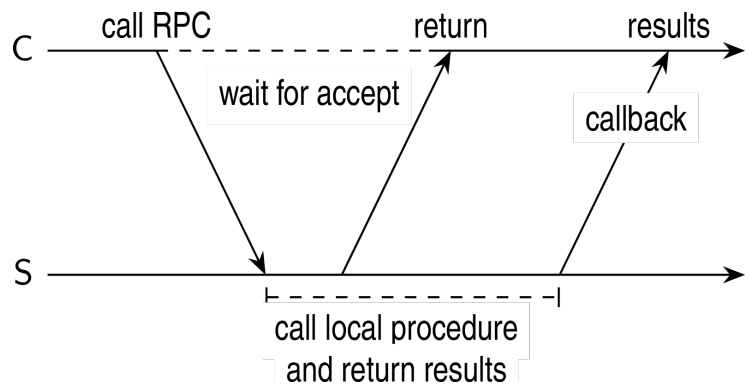
- Multiple binders – Replicate binding information; More availability; Load Balancing

Asynchronous RPC

Request-reply behavior may not be needed

- Blocking may waste resources

Asynchronous behavior – Client continue without waiting for an answer from the server



Demo

Next Class…

Failure Handling

Next Class…