Parallel and Distributed Computing
CS3006 (BDS-6A)
Lecture 12

Instructor: Dr. Syed Mohammad Irteza

Assistant Professor, Department of Computer Science, FAST

09 March, 2023

Previous Lecture

- OpenMP `locks`
- Clauses: `private`, `lastprivate`, `firstprivate`, `if`, `default`
- OpenMP Scheduling: `guided`, `runtime`
- OpenMP Environment Variables:
  - `OMP_NESTED, OMP_DYNAMIC, OMP_NUM_THREADS, OMP_SCHEDULE`
- Parallel Pi
  - Using the Monte Carlo method (randomized)

Computing Pi using the Monte Carlo method

(Parallel construct [parallel_pi.c])

```c
#pragma omp parallel shared(niter) private(i, x, y, z, chunk_size, seed) reduction(+:count) {

  num_threads = omp_get_num_threads();
  chunk_size = niter / num_threads;
  seed = omp_get_thread_num();
  #pragma omp master
  { printf("chunk_size=%ld\n",chunk_size); }

  count=0;
  for (i=0; i < chunk_size; i++) {
        //get random points
        x = (double) rand_r(&seed) / (double) RAND_MAX;
        y = (double) rand_r(&seed) / (double) RAND_MAX;
        z = ((x-0.5)*(x-0.5))+((y-0.5)*(y-0.5));
        //check to see if point is in unit circle
        if (z<0.25) {
              ++count;
        }
  }
}
pi = ((double) count / (double) niter) * 4.0;
```

```
total number of allocated cores are:16
chunk_size=6250000
parallel Pi: 3.141515
Parallel time: 0.9560 seconds
Seq_Pi: 3.141745
 Sequential time: 13.3521 seconds
speedup: 13.9669
```

Total points= 10 millions

```
total number of allocated cores are:16
chunk_size=62500000
parallel Pi: 3.141598
Parallel time: 8.5668 seconds
Seq_Pi: 3.141576
 Sequential time: 132.0383 seconds
speedup: 15.4128
```

Total points= 100 millions

More Detailed Discussion

- Full Example Online: http://www.umsl.edu/~siegelj/cs4790/openmp/pimonti_omp.c.HTML
- Further Reading (optional):
  - https://1drv.ms/p/s!Apc0G8okxWJ12jlUANaQsYO-JVdx?e=VixgYX (just slide 1-9)
  - https://passlab.github.io/CSCE569/notes/lecture04-07_OpenMP.pdf
  - https://www3.nd.edu/~zxu2/acms60212-40212/Lec-12-OpenMP.pdf

Classic computing

- Advantages:
  - Ease of programming (shared memory)
  - Robustness
  - Good chance of optimization
- Disadvantages:
  - data protection from illegal operations
  - low performance
    - single CPU, sequential computation
    - mitigated with multi-CPU systems and concurrent programming (thread, processes)
  - requires physical access to the system (for usage)
    - mitigated with modem / network connections

Distributed System

- A distributed System is:

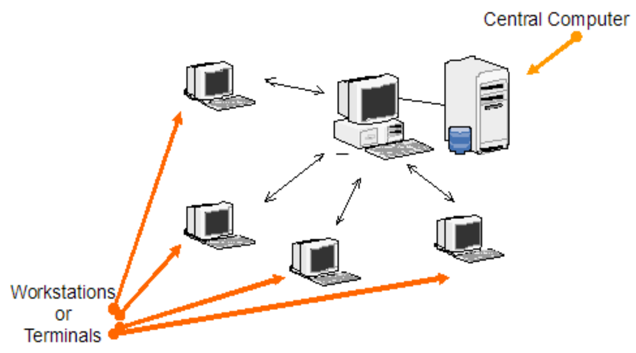
A collection of independent computers that appears to its users as a single coherent system

Distributed Systems

- Networks of computers are everywhere!
  - Mobile phone networks
  - Campus networks
  - In-car networks
  - On board networks in planes and trains
- *"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]

Reasons for 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 (e.g., Coles, Woolworths)

Reasons for Distributed Systems

- Power imbalance and load variation:
  - Distribute computational load among different computers.

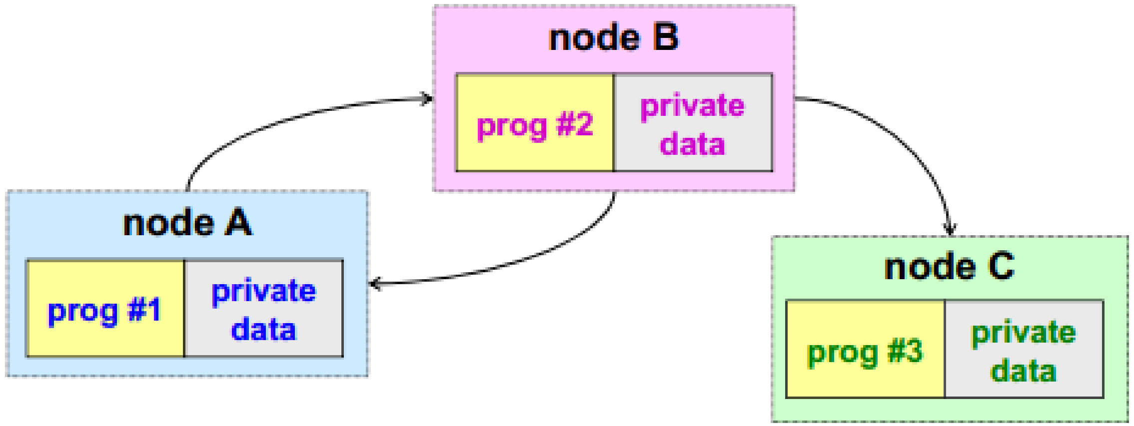
- Reliability:
  - Long term preservation and data backup (replication) at different locations.

- Economies:
  - Sharing a printer by many users and reduce the cost of ownership.
  - Building a supercomputer out of a network of computers.

Distributed systems

- “A distributed system is a system designed to *support* the development of applications and services which can *exploit* a physical architecture consisting of multiple *autonomous* processing elements that *do not share primary memory* but cooperate by sending *asynchronous* messages over a communications network”
    - [Blair and Stefani, 1998]

Distributed Systems

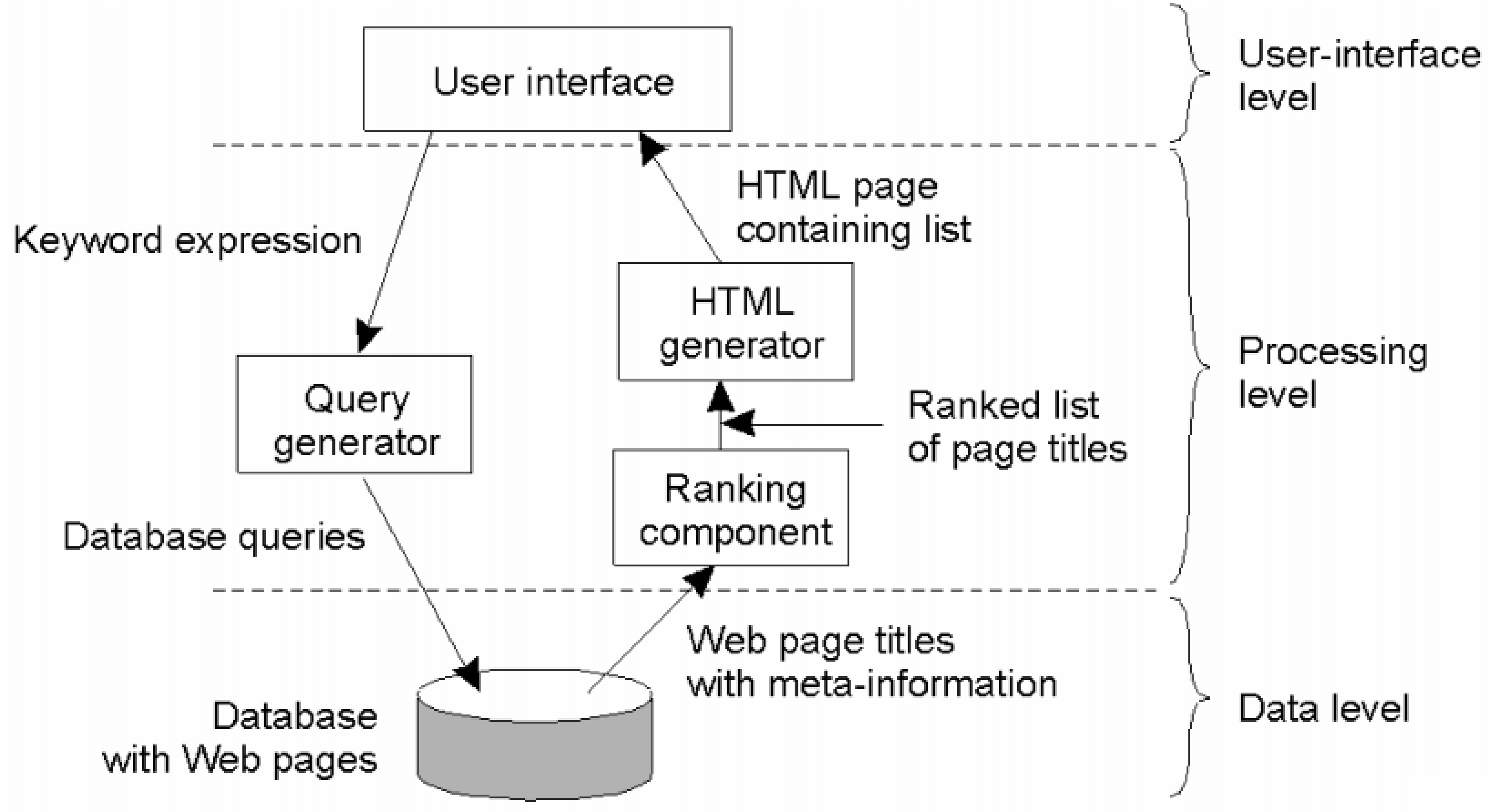


- Each system has its local data (private)
- Multiple address spaces
- Concurrent computation over different CPUs
- multiple computation flows

Examples of distributed systems

- Internet: network of networks
  - Global access to everyone
  - Huge size
  - No single authority
- Intranets
  - Protected access
  - Single authority
- E-learning
- Online gaming
- Mobile ad-hoc networks

Search engine



- http://www.ece.rutgers.edu/~parashar

Advantages of Distributed systems

- High performance
  - Several CPUs
- Good scalability
  - Increasing the number of CPUs is easier then increasing the performance of a single CPU
- Data protected from illegal operations
  - Disjoint memory spaces, accessible only by their respective programs
- Network access
  - user physical presence not required

Trends in distributed systems

- Pervasive network technology
  - modern Internet is a vast interconnected collection of computer networks of many different types
  - range of types increasing all the time
  - devices can be connected at any time and at any place

- Mobile and ubiquitous computing
  - integration of small and portable computing devices into distributed systems
  - Laptops, PDAs, smart watches, smart embedded devices in appliances
  - location-aware or context-aware computing

Trends in distributed systems

- presence of computers everywhere only becomes useful when they can communicate with one another
- it is more desirable for users to control their washing machine or their entertainment system from their phone or a ‘universal remote control’ device in the home

- Spontaneous interoperation
  - associations between devices are routinely created and destroyed
  - Challenges: should be fast and convenient

Trends in distributed systems

- Distributed computing as a utility
  - resources are provided by service suppliers and effectively rented rather than owned by the end user
  - Cloud Computing
- Focus on resource sharing
  - we routinely share hardware resources such as printers, files, and resources with more specific functionality such as search engines
  - Client/Server computing
  - Peer to Peer computing

Forms of Distributed Computing

Cluster Computing

"A computer cluster is a group of loosely coupled computers that work together closely so that in many respects it can be viewed as though it were a single computer. Clusters are commonly connected through fast local area networks."

Forms of Distributed Computing

Grid Computing

- “Grid is an infrastructure that involves the integrated and collaborative use of Computers, networks, databases and scientific instruments owned and managed by multiple organizations”

- Grid Computing is Biased towards the solution of Scientific problems

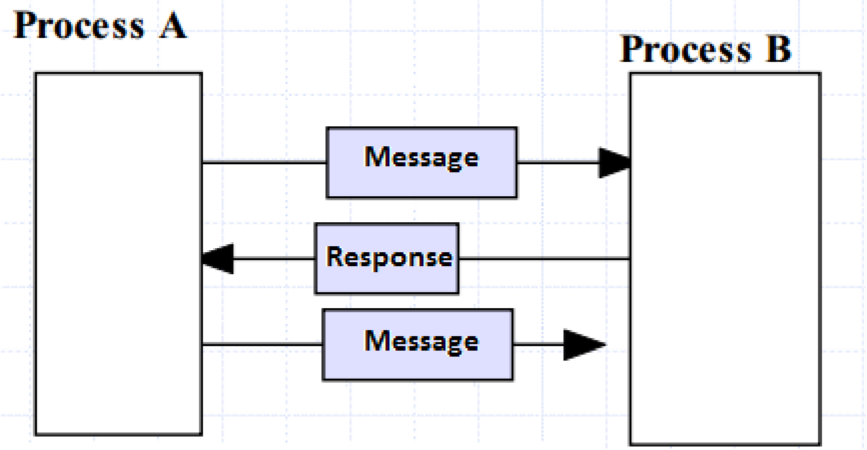
Forms of Distributed Computing

Cloud Computing

A cloud is defined as a set of Internet-based application, storage and computing services sufficient to support most users' needs, thus enabling them to largely or totally dispense with local data storage and application software

Message passing paradigm

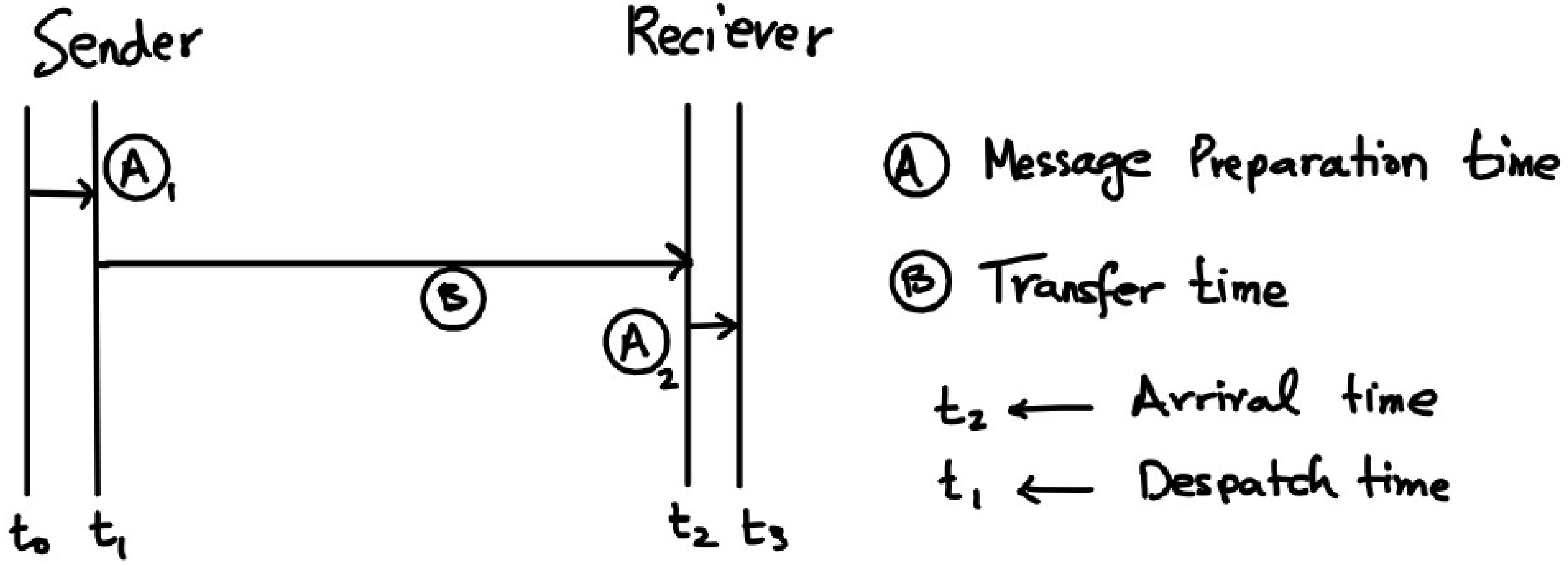
- Most fundamental paradigm for distributed applications



- 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*
- In turn, the *reply* may trigger a further *request*, which leads to a subsequent *reply*, and so forth

Communication Model

- Message preparation & Message transfer time



- **Arrival latency = transfer time = $t_2 - t_1$**

Parameters for measuring arrival latency

Bandwidth : Data Volume (r) sent / unit time

e.g 400 mbits / second $\Rightarrow$ 1 mbit transfer = $\frac{1s}{400}$

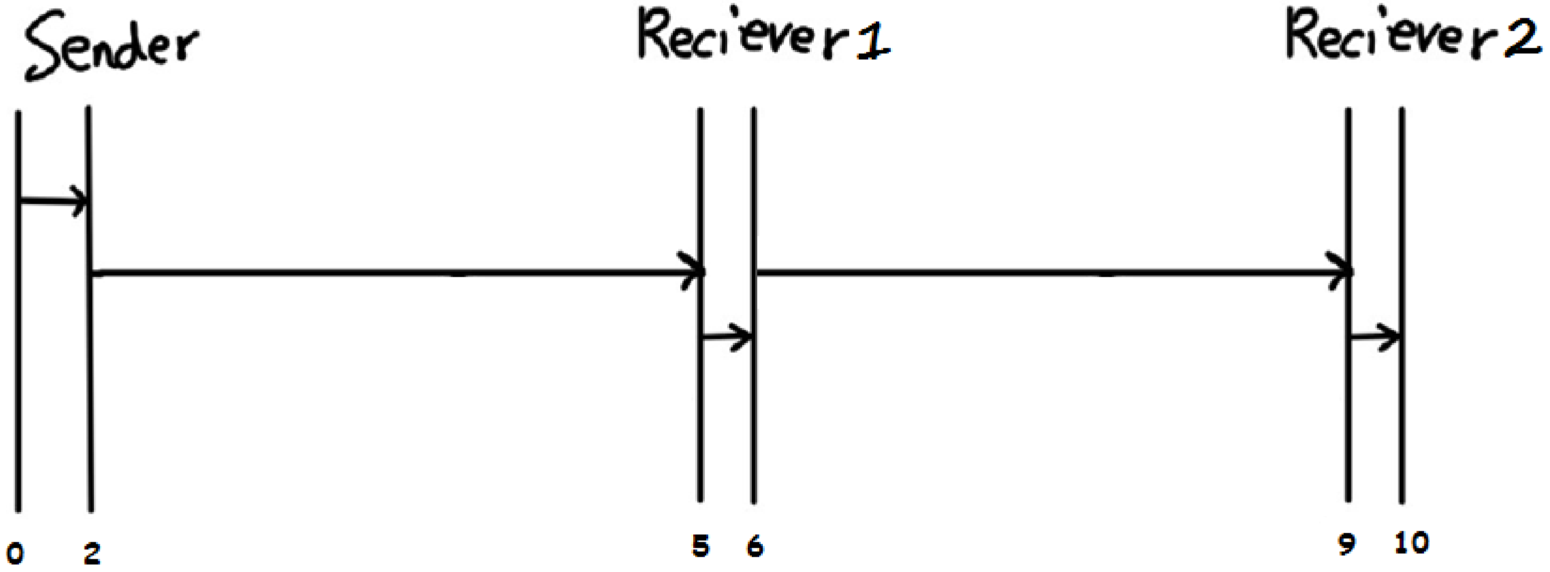
1 unit transfer = 1 / r

- T_w = 1 unit transfer time = 1/r
- T_s = preparation time = $t_1 - t_0$
- T_h = Arrival latency = $t_2 - t_1$
- T_r = Receiver handling time = $t_3 - t_2$
- Size of message = m
- No. of hops/nodes, the message is to be sent across = n

- Total time for sending a message of size m across n hops:

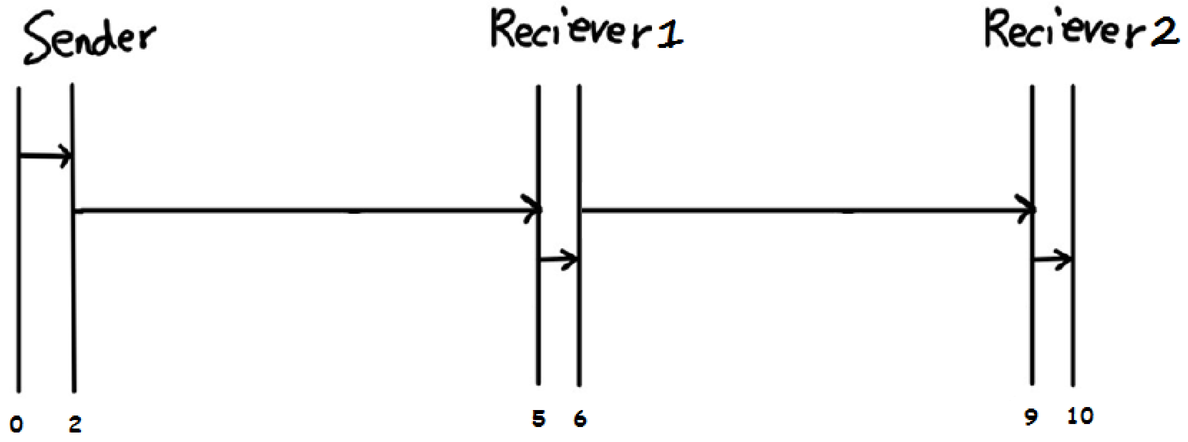
$$t = t_s + (t_w\, m + t_h + t_r)\, n$$

- Example:



Example: $t = t_s + (t_w\, m + t_h + t_r)\, n$

- Bandwidth = 2 Mbps (megabits per second)
- m (message size) = 100 megabits
- n = 2

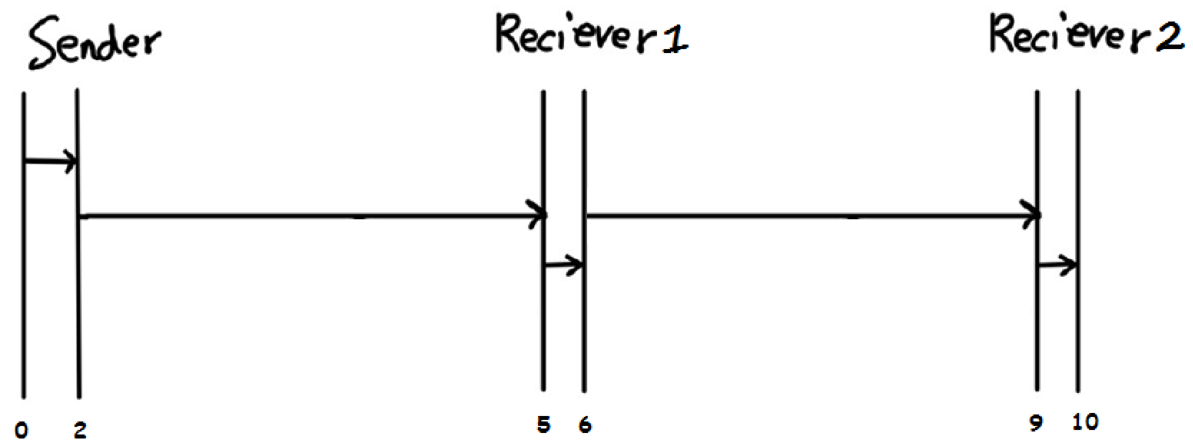


Example: $t = t_s + (t_w\, m + t_h + t_r)\, n$

- Bandwidth = 2 Mbps (megabits per second)
- m (message size) = 100 megabits
- n = 2

T_w = 1 unit transfer time = ½ (in terms of Mbits)
$T_s = t_1 - t_0 = 2 - 0 = 2$
$T_h = t_2 - t_1 = 5 - 2 = 3$
$T_r = t_3 - t_2 = 6 - 5 = 1$

$t = t_s + (t_w\, m + t_h + t_r)\, n$
$= 2 + (½ * 100 + 3 + 1) * 2 = 110s$



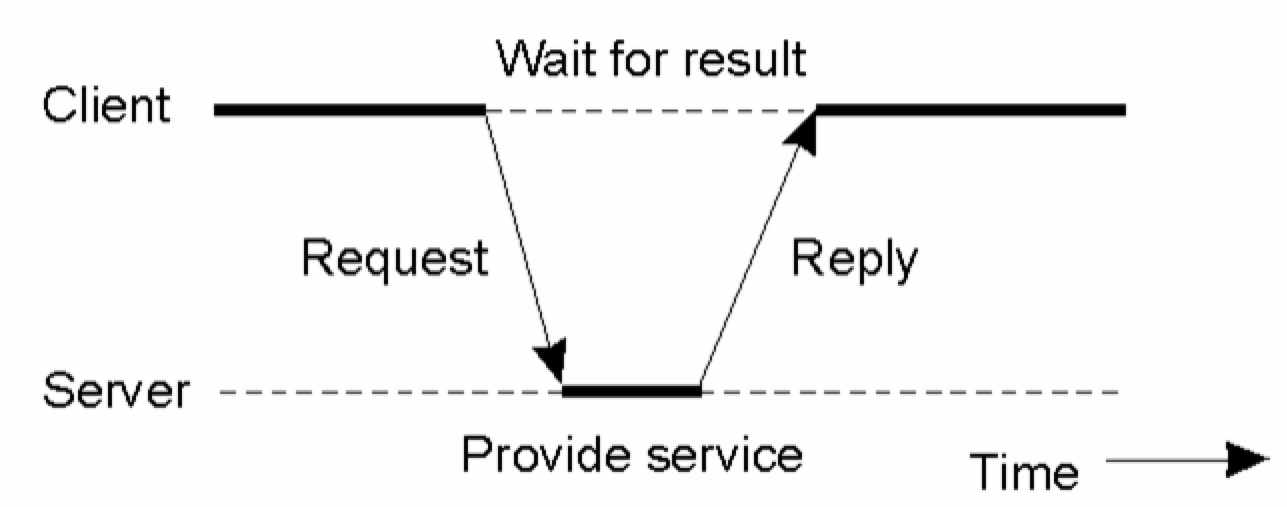
Architectures

- Client-Server (C/S) architecture
  - Asymmetric architecture
  - Server position is determined *a priori*

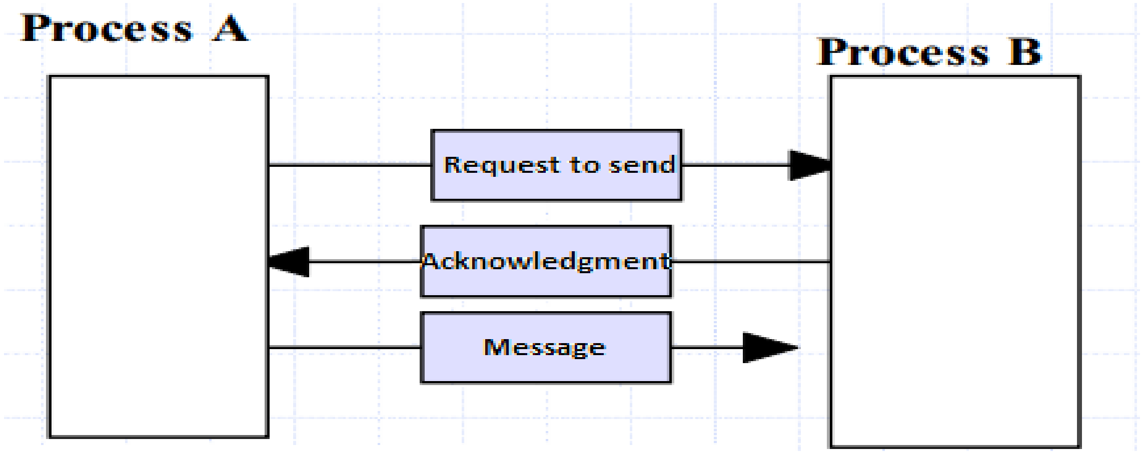
- Peer-to-Peer (P2P) architecture
  - Symmetric architecture
  - Every node can play both the client and server roles (simultaneously or at different times)

Client/Server paradigm

- Most *widely used paradigm* for network applications
- *Server* plays the role of a *service provider* which waits passively for the arrival of request
- *Client* issues specific *requests* to the server and waits for its response



Synchronous message passing

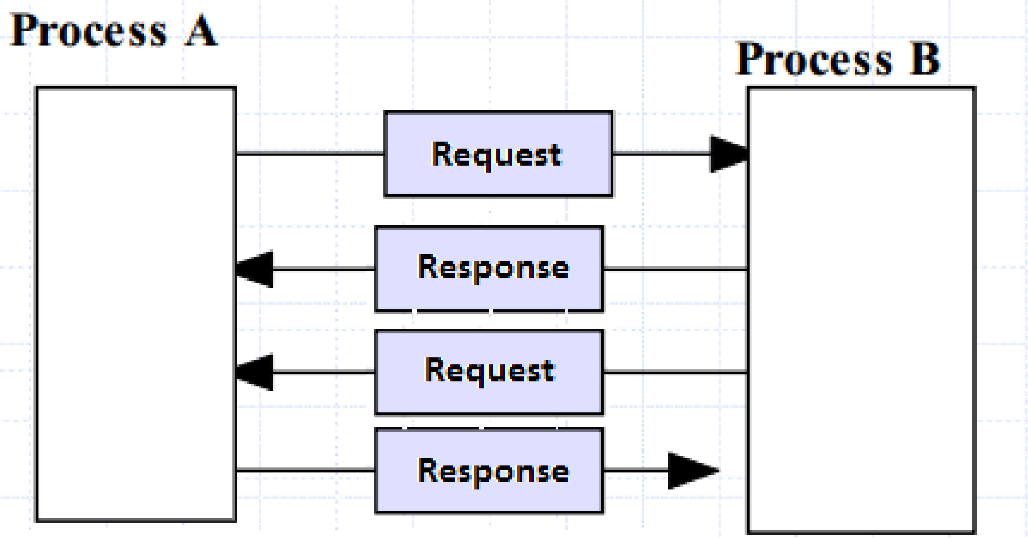


- Routines *return* when message transfer *completed*
- Send routine *waits* until complete message can be accepted by the receiving process before *sending* the message
- Receive routine *waits* until the message it is expecting *arrives*

Asynchronous Message Passing

- Routines *do not wait* for actions to complete before returning
- Local *storage/message queue* is required to hold messages
- Advantage: *Faster*, since it allows processes to move forward
- Disadvantage: *Less reliable* and must be used with care
- Message buffer needed between source and destination to hold messages
- Send routine *wait* in case the buffer space is exhausted

Peer to Peer system architecture



- In the peer-to-peer paradigm, the participating 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*

Peer to Peer system architecture

- Computer resources and services are *directly exchanged* between computer systems
- Resources and services include the *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
- The peer-to-peer paradigm is more appropriate for applications such as *instant messaging*, *peer-to-peer file transfers*, *video conferencing*, and *collaborative work*

P2P systems: some history

- [1.3 P2P Systems - Rui Pan's Blog (gitbook.io)]
  - Napster
  - Gnutella
  - BitTorrent

Distributed Systems: Issues

- *Programming Complexity*
  - How the various programs communicate together?
  - Which data format on the various network nodes?
  - Need to define (application) protocols
  - Operations synchronization may lead to delay and slowing down
- *Scarce Robustness*
  - Higher chance of errors/faults
- *Hard to Optimize*
  - Lack of a global view

Quiz 02 (10 minutes) -- BDS 6A

- What are the possible ways to set the number of threads within an OpenMP program to 4? (3 marks)
- What is the impact of setting the environment variable OMP_NESTED to TRUE? (3 marks)
- Draw a possible mapping of iterations to threads, assuming that we are using guided, with C=2, and total iterations are 18 and there are 3 threads. (6 marks)