Distributed Algorithms

Partha Sarathi Mandal

Department of Mathematics

IIT Guwahati

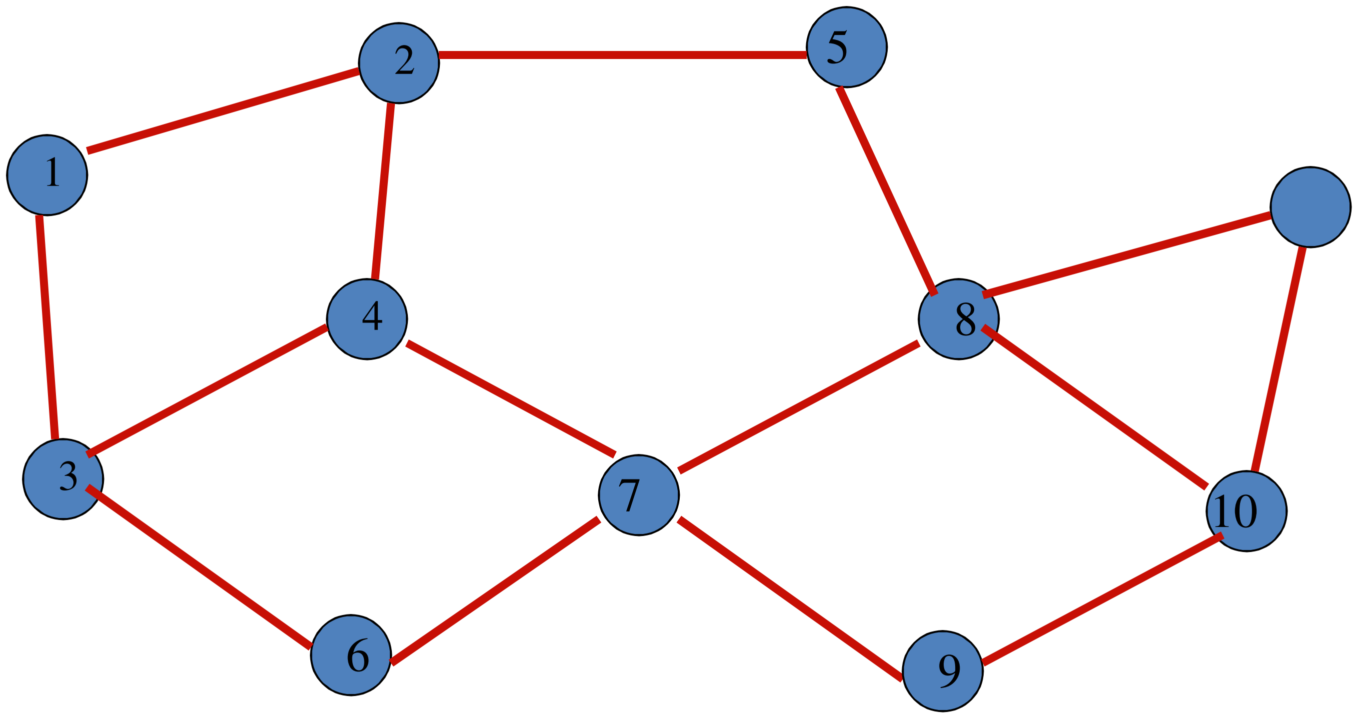
- Thanks to **Dr. Sukumar Ghosh** for the slides

Distributed Algorithms

- Distributed algorithms for various graph theoretic problems have numerous applications in distributed computing system.

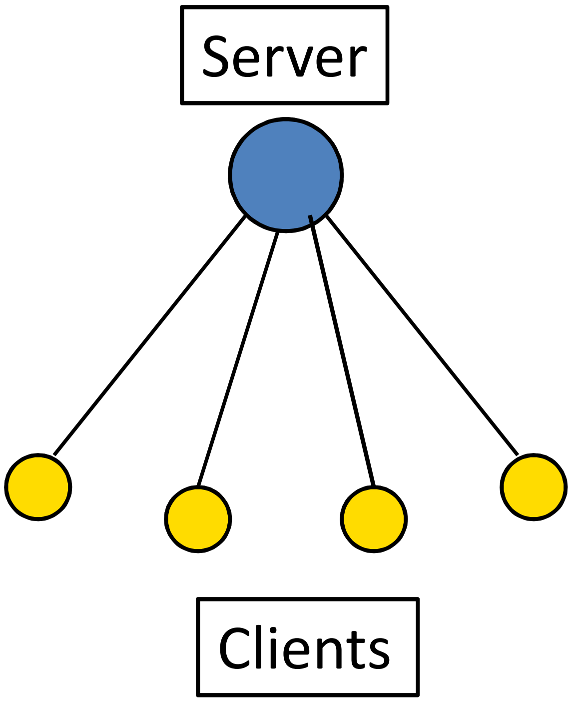
- What is a distributed computing system ?
  - The topology of a distributed system is represented by a graph where the nodes represent processes, and the links represent communication channels.

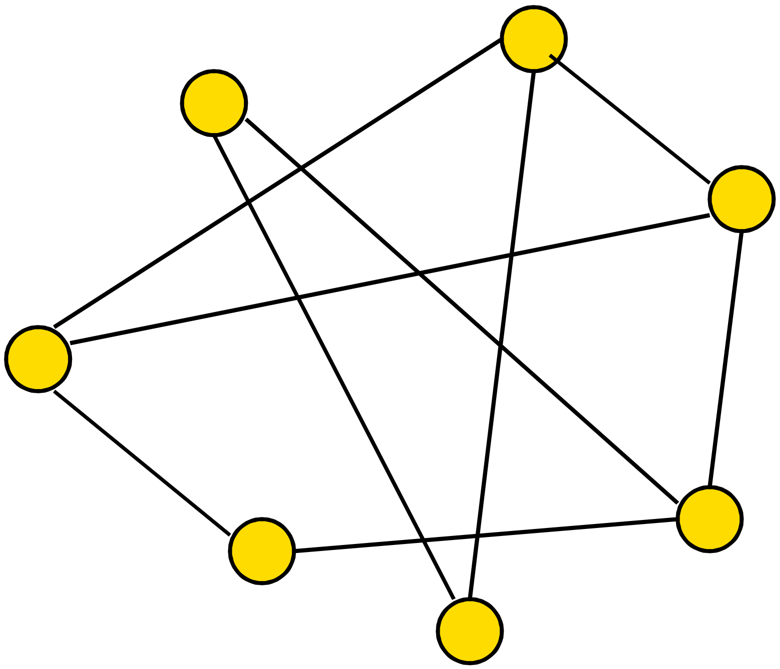
Topology of a DS is represented by a graph



A **network** of processes. The **nodes** are processes, and the **edges** are communication channels.

Examples





Client-server model

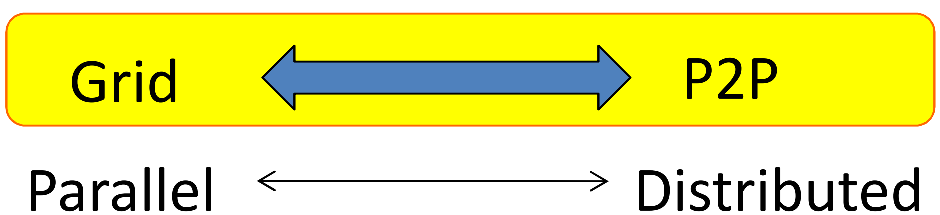
Server is the coordinator

Peer-to-peer model

No unique coordinator

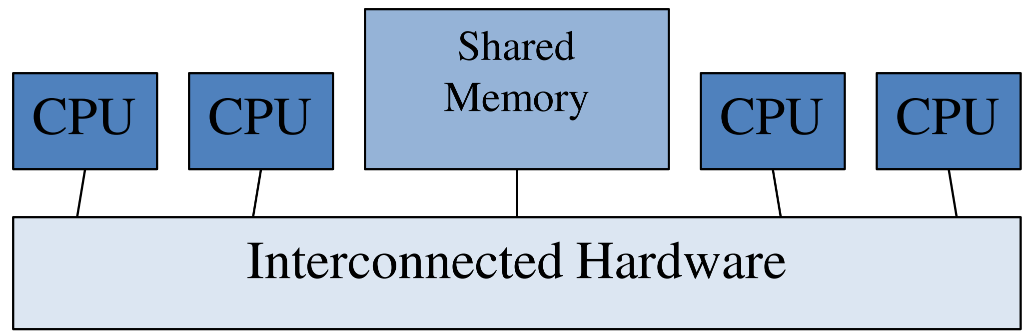
Parallel vs Distributed

- In both parallel and distributed systems, the events are ***partially ordered***.
- The distinction between parallel and distributed is not always very clear.
- In **parallel** systems, the primarily issues are
  - speed-up and increased data handling capability.
- In **distributed** systems the primary issues are
  - fault-tolerance, synchronization, scalability, etc.

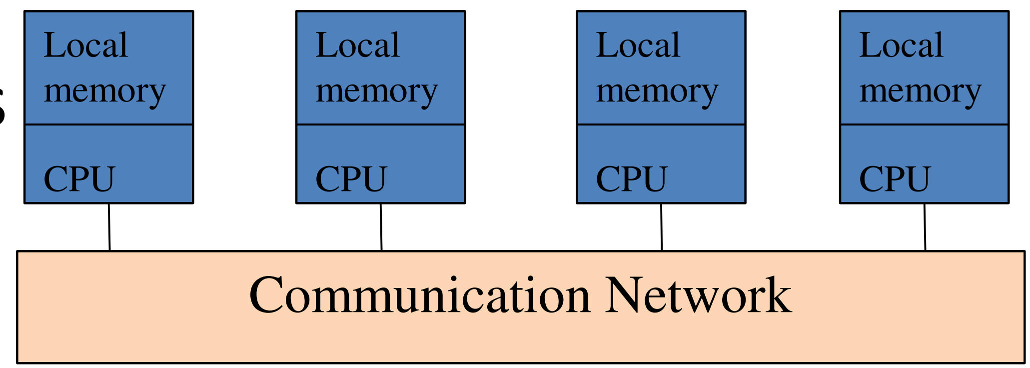


Parallel vs Distributed
(Multiple processors)

- Tightly coupled systems
  – Parallel processing systems.



- Loosely coupled systems
  – Distributed computing systems.



Examples

Large networks are very commonplace these days. Think of the world wide web.

Other examples are:

- Social Networks
- Sensor networks
- BitTorrent for downloading video / audio
- Skype for making free audio and video communication
- Computational grids
- Network of mobile robots

Why distributed systems

- Geographic distribution of processes
- Resource sharing (example: P2P networks, grids)
- Computation speed up (as in a grid)
- Fault tolerance
- etc.

Important services

- Internet banking
- Web search
- Net meeting
- Distance education
- Video distribution
- Internet auction
- Google earth
- Skype
- Publish subscribe

Important issues

- Knowledge is local
- Clocks are not synchronized
- No shared address space
- Topology and routing : everything is dynamic
- **Scalability: all solutions do not scale well**
- Processes and links fail: **Fault tolerance**

Some common subproblems

- Leader election
- Mutual exclusion
- Time synchronization
- Global state collection
- Replica management
- Consensus
- Coloring
- Self-stabilization

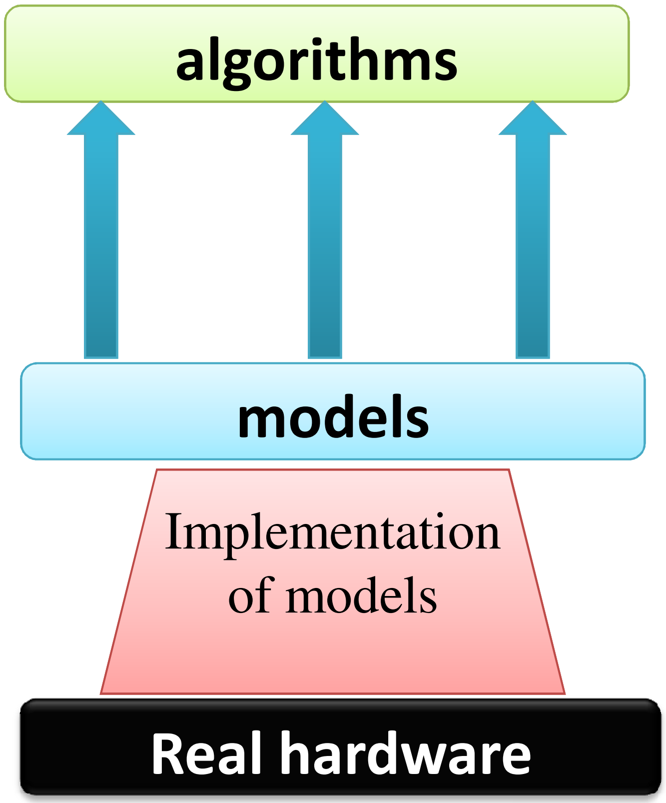
Models

- We will motive about distributed systems using models. There are many dimensions of variability in distributed systems.
- Examples:
  - types of processors
  - inter-process communication mechanisms
  - timing assumptions
  - failure classes
  - security features, etc

Models

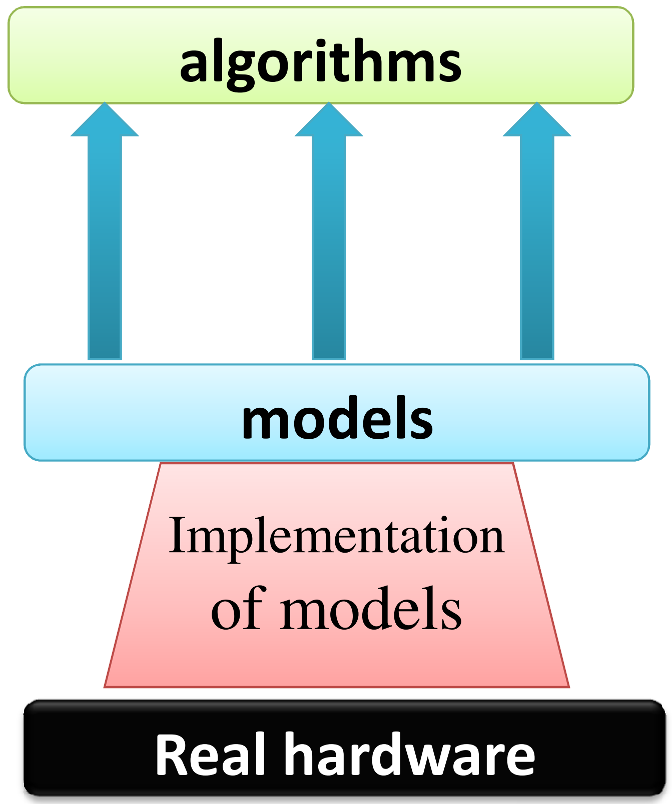
- Models are simple abstractions that help to overcome the variability -- abstractions that preserve the essential features, but hide the implementation details and simplify writing distributed algorithms for problem solving.

- Optical or radio communication ?
- PC or Mac ?
- Are clocks perfectly synchronized ?



Understanding Models

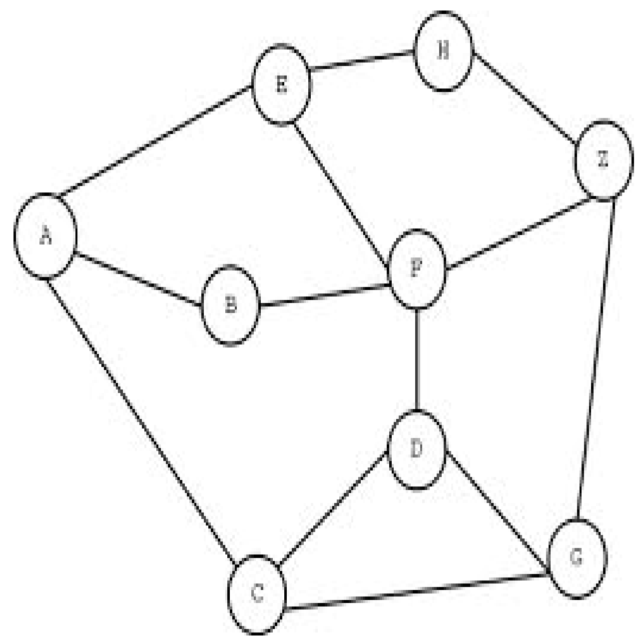
- How models help



Modeling Communication

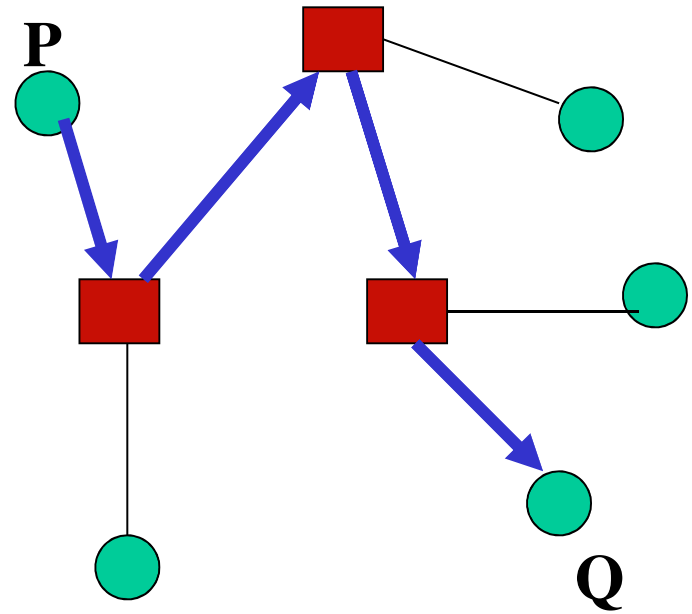
- System topology is a graph G = (V, E), where V = set of nodes (sequential processes) E = set of edges (links or channels, bi/unidirectional).

  Four types of actions by a process:
  - internal action
  - input action
  - communication action
  - output action

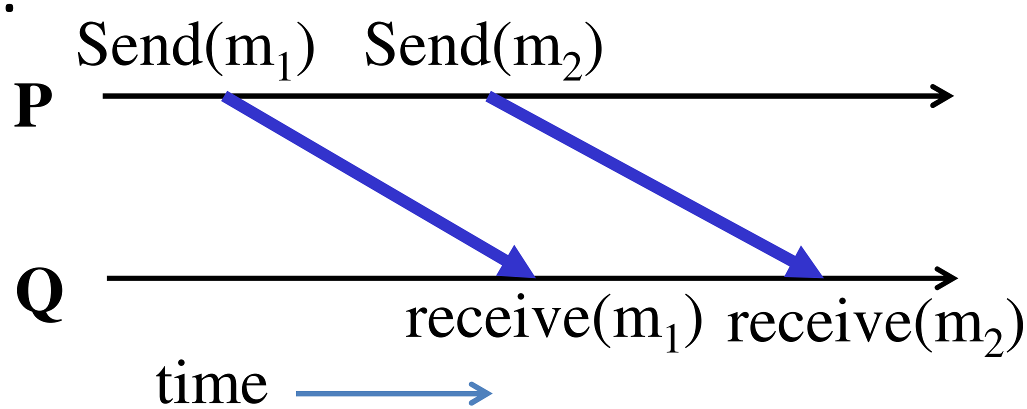


Example: A Message Passing Model

A Reliable FIFO Channel

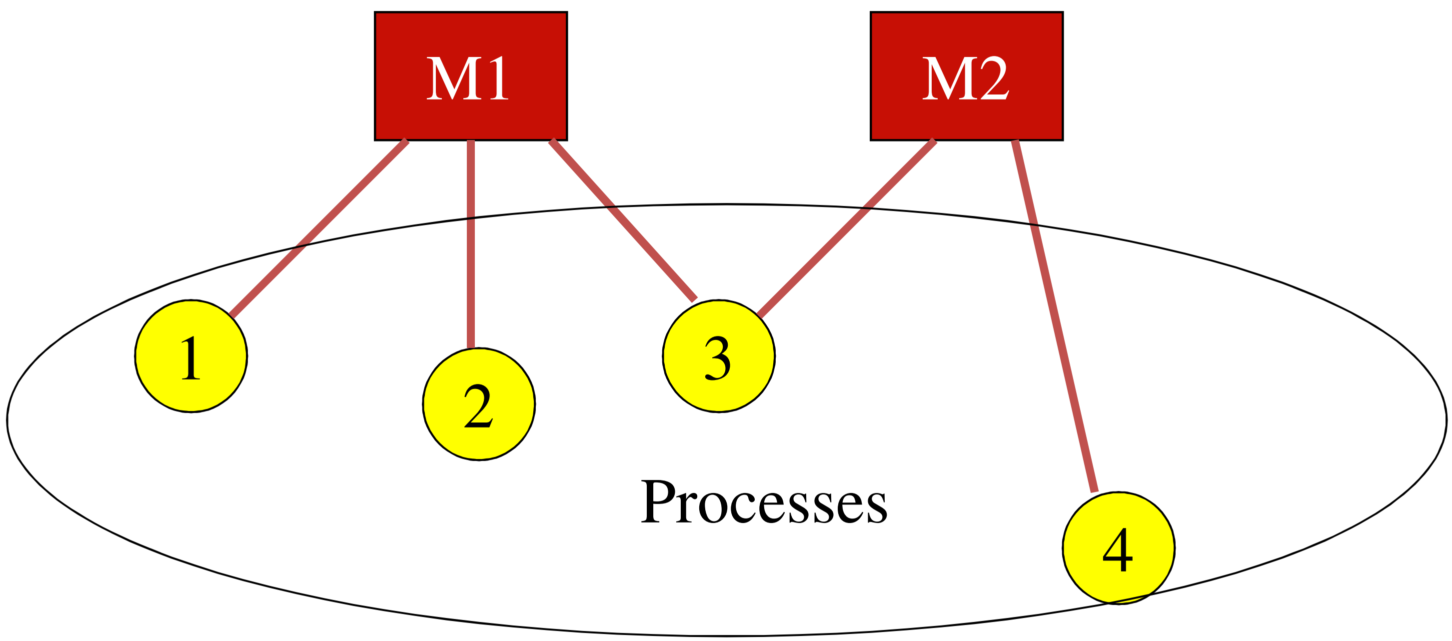


- **Axiom 1**. Message m sent ⇔ message m received
- **Axiom 2**. Message propagation delay is arbitrary but finite.
- **Axiom 3**. m_1 sent before m_2 ⇔ m_1 received before m_2.



Example: Shared memory model

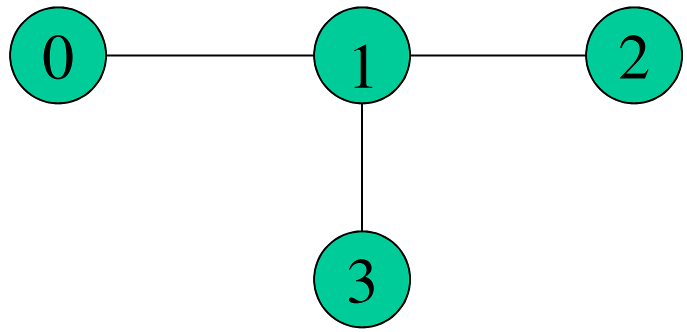
- Address spaces of processes overlap



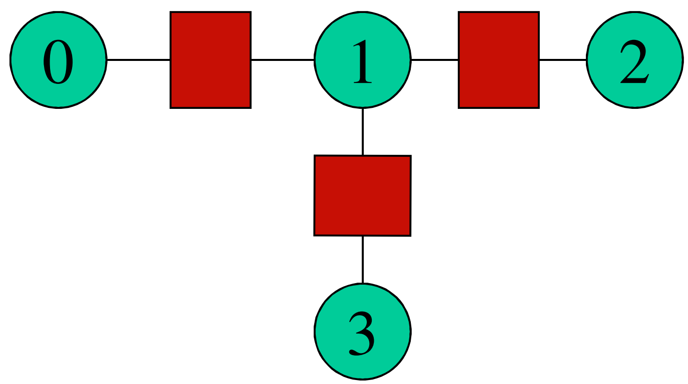
- Concurrent operations on a shared variable are serialized

Variations of shared memory models

- State reading model
  - Each process can read the states of its neighbors

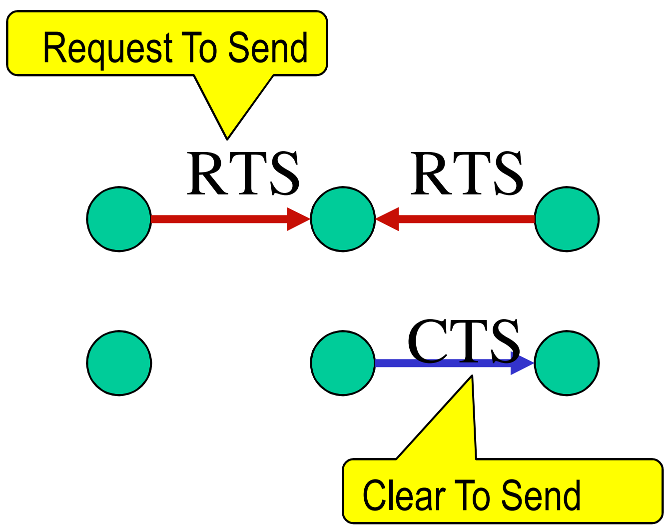


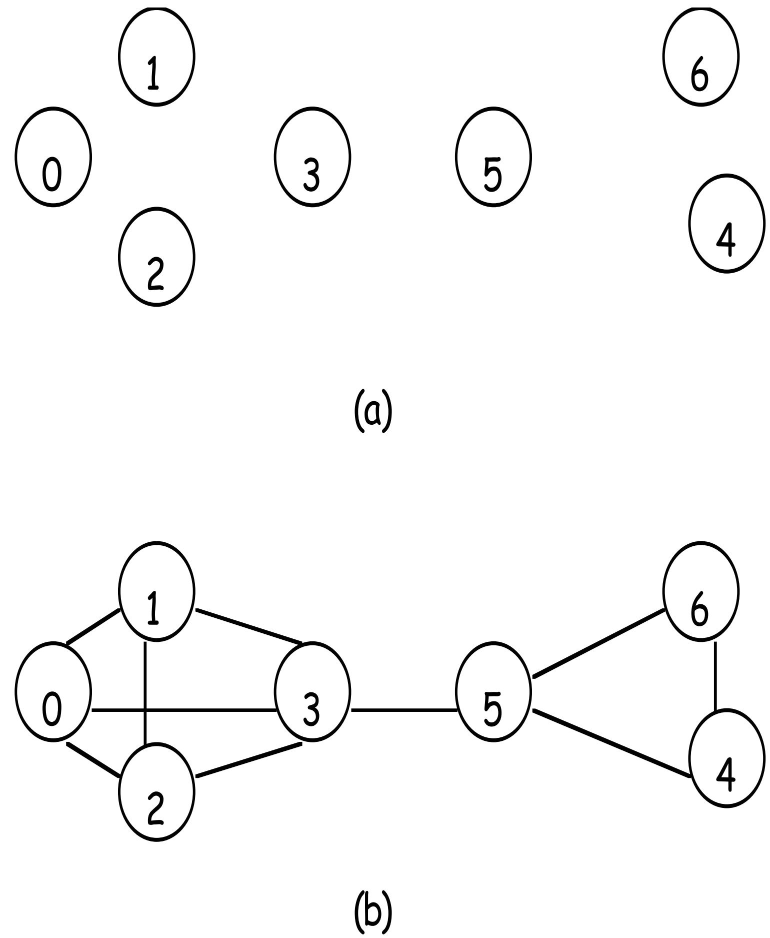
- Link register model
  - Each process can read from and write to adjacent registers. The entire local state is not shared.



Modeling wireless networks

- Communication via broadcast
- Limited range
- Dynamic topology
- Collision of broadcasts
  (handled by CSMA/CA)





Synchrony vs. Asynchrony

Synchronous clocks	Physical clocks are synchronized
Synchronous processes	Lock-step synchrony
Synchronous channels	Bounded delay
Synchronous message-order	First-in first-out channels
Synchronous communication	Communication via *handshaking*

Send & receive can be
blocking or **non-blocking**

Postal communication is
asynchronous

Telephone communication
is synchronous

Synchronous
communication or not?

(1) Remote Procedure Call,

(2) Email

Weak vs. Strong Models

- One object (or operation) of a **strong model** = More than one simpler objects (or simpler operations) of a **weaker model**.
- Often, ***weaker models*** are synonymous with ***fewer restrictions***.
- *One can add layers (**additional restrictions**) to create a **stronger model** from weaker one.*

<u>Examples</u>

High level language is *stronger* than **assembly language**.

Asynchronous is *weaker than* **synchronous** (communication).

Bounded delay is *stronger* than **unbounded delay** (channel)

Model transformation

Stronger models

- simplify reasoning, but
- needs extra work to implement

*"Can **model X** be implemented using **model Y?"** is an interesting question in computer science.*

Weaker models

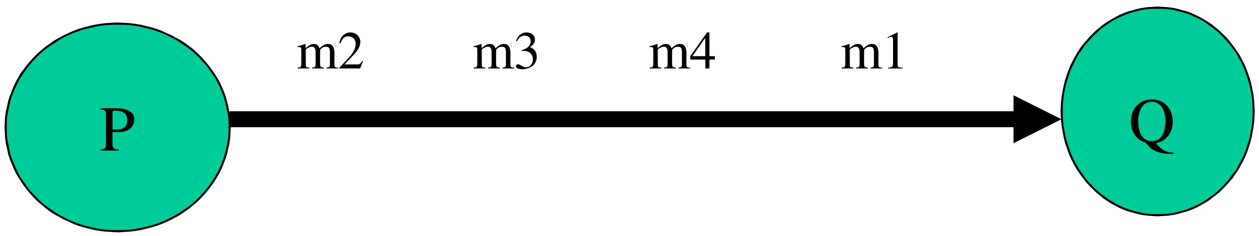
- are easier to implement.
- Have a closer relationship with the real world

Sample exercises

1. ***Non-FIFO*** to ***FIFO*** *channel*
2. ***Message passing*** to ***shared memory***
3. ***Non-atomic broadcast*** to ***atomic broadcast***

Non-FIFO to FIFO channel

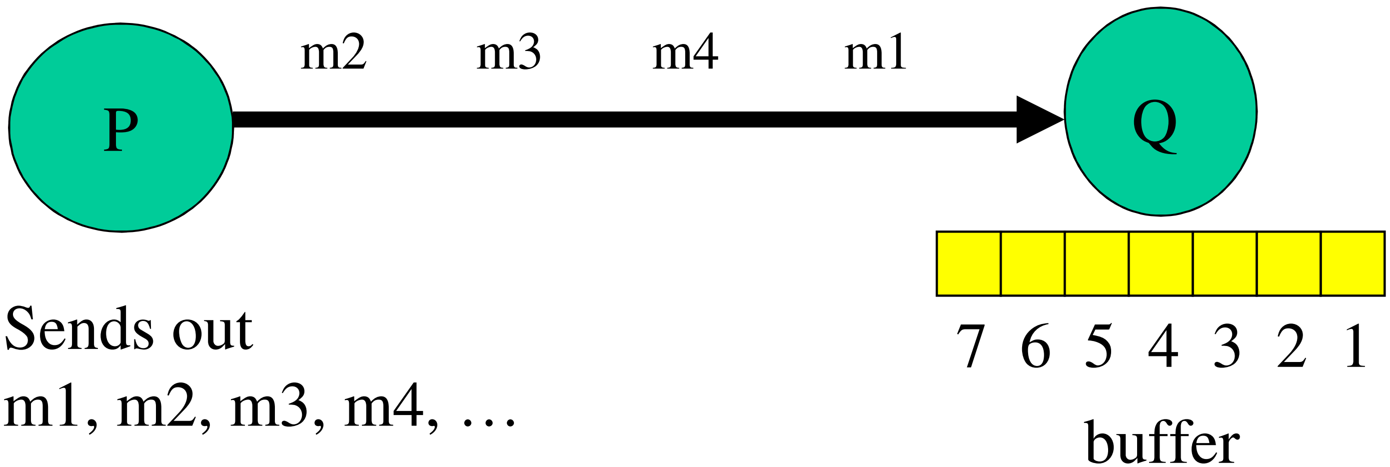
FIFO = First-In-First-Out



Sends out
m1, m2, m3, m4, …

Non-FIFO to FIFO channel

FIFO = First-In-First-Out



Sends out
m1, m2, m3, m4, …

Non-FIFO to FIFO channel

{Sender process P}

var i : **integer** {initially 0}

repeat
    send m[i],i to Q;
    i := i+1
forever

{Receiver process Q}

var k : **integer** {initially 0}
    buffer: **buffer**[0..∞] of msg
    {initially ∀k: buffer [k] = empty

repeat {STORE}
    receive m[i],i from P;
    store m[i] into buffer[i];
    {DELIVER}
    while buffer[k] ≠ empty **do**
    begin
        deliver content of buffer [k];
        buffer [k] := empty; k := k+1;
    end
forever

Needs **unbounded buffer &**
unbounded sequence no
THIS IS BAD

Observations

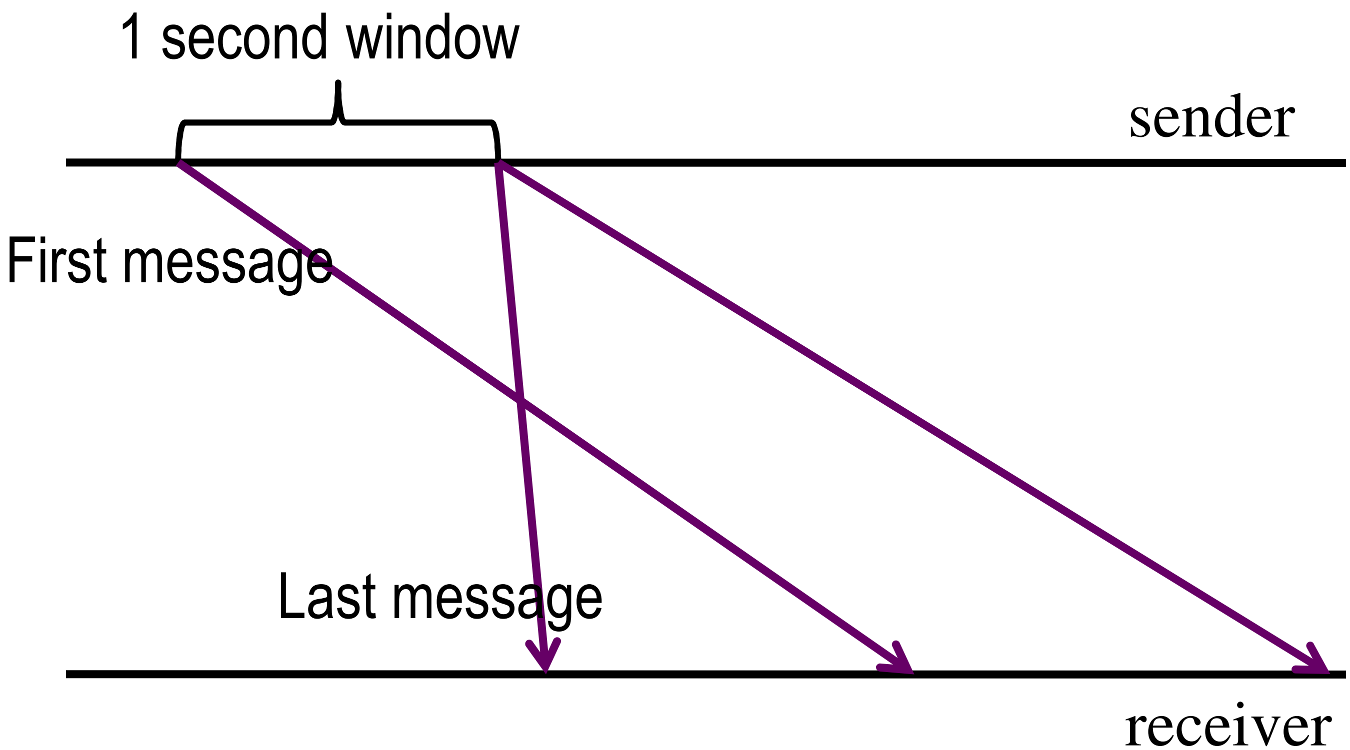
- Now solve the same problem on a model where
  a) The propagation delay has a known upper bound of **T**.
  b) The messages are sent out @ **r** per unit time.
  c) The messages are received at a rate *faster than **r***.

The buffer requirement drops to r.T.

(Lesson) Stronger model helps.

Question. *Can we solve the problem using **bounded** buffer space if the propagation delay is arbitrarily large?*

Example

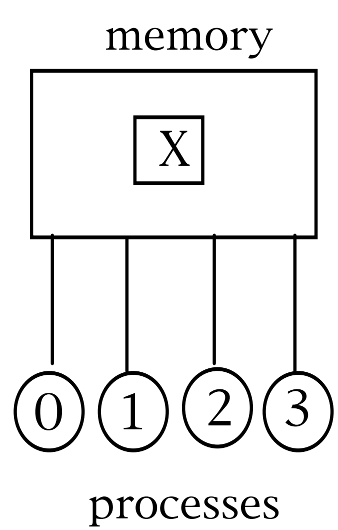

1 second window
sender
First message
Last message
receiver

Implementing Shared memory using Message passing

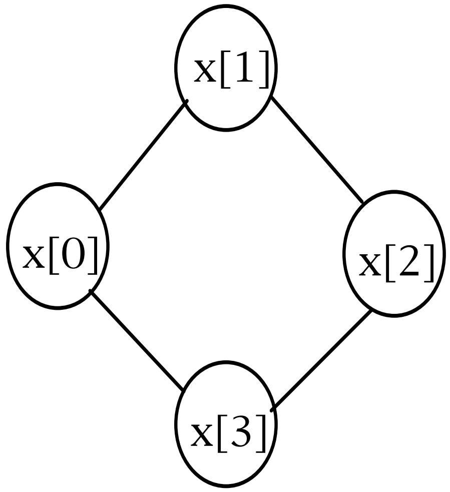
{Read **X** by process **i**}**:** read **x[i]**

{Write **X:= v** by process **i**}

- **x[i] := v**;
- **Atomically broadcast v** to every other process **j (j ≠ i);**
- After receiving broadcast, process **j (j ≠ i)** sets **x[j]** to **v**.



(a)



(b)

*Understand the significance of **atomic operations**. It is not trivial, but is very important in distributed systems.*

Implementation **Atomically broadcast** is far from trivial

Atomic = all or nothing

Non-atomic to atomic broadcast

Atomic broadcast = either <u>everybody</u> or <u>nobody</u> receives

{process i is the sender}
for j = 1 to N-1 (j ≠ i) **send** message **m** to neighbor [j]
(Easy!)

Now include **crash failure** as a part of our model.
What if the sender crashes at the middle ?

How to implement atomic broadcast in presence of crash?

Complexity Measures

Common measures

Space complexity

How much space is needed per process to run an algorithm? (measured in terms of N, the size of the network)

Time complexity

What is the max. time (number of steps) needed to complete the execution of the algorithm?

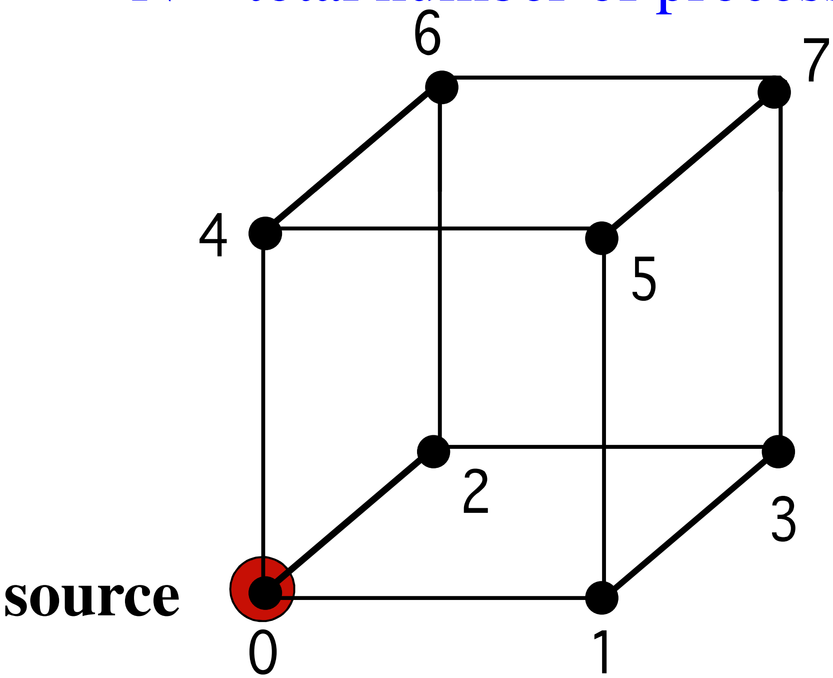
Message complexity

How many message are needed to complete the execution of the algorithm?

An example

Consider broadcasting in an n-cube (here n=3)

N = total number of processes = $2^n = 8$



Each process $j > 0$ has a variable $x[j]$, whose initial value is arbitrary

Broadcasting using messages

{**Process 0**} m.value := x[0];

send **m** to all neighbors

{**Process i > 0**}

repeat

receive m {m contains the value**}**;

if m is received for the *first time*

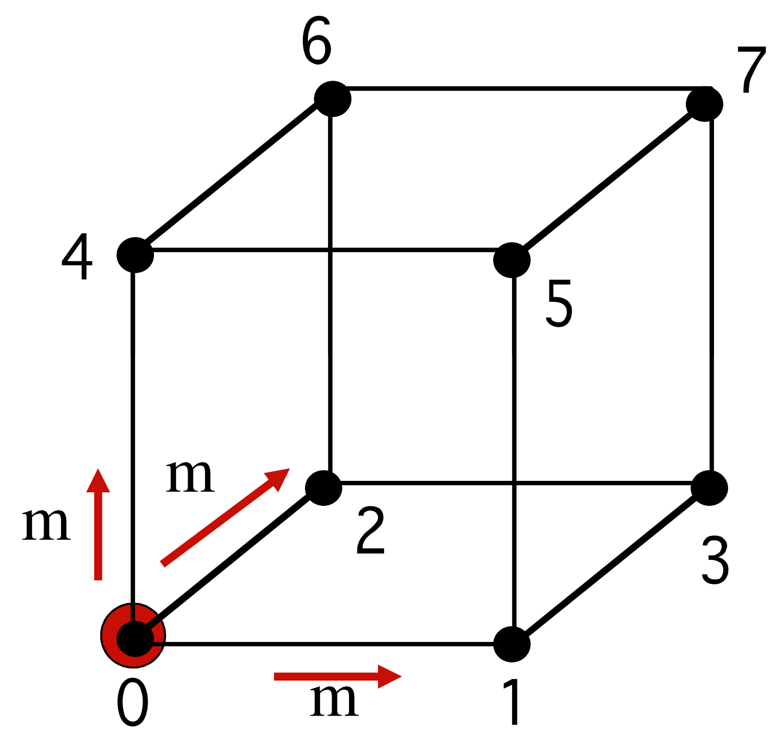
then x[i] := m.value;

send **x[i]** to each neighbor **j > i**

else discard **m**

end if

Forever



What is the

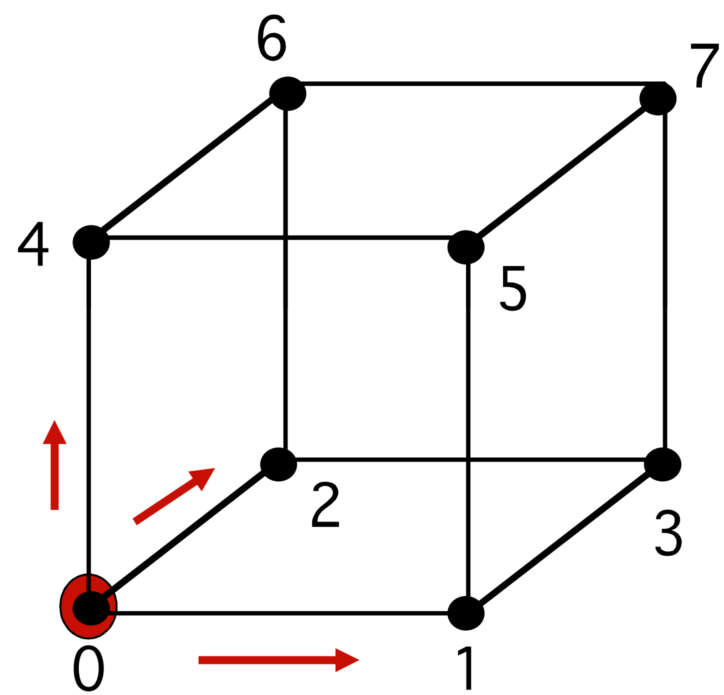
(1) Message & time complexities → $1/2\ (N \log_2 N)$ & $\log_2 N$

(2) space complexity per process? → $\log_2 N$

Broadcasting using shared memory

{Process 0} x[0] := v

{Process i > 0}
repeat
    if $\exists$ a neighbor j < i : x[i] $\neq$ x[j]
        then x[i] := x[j] (PULL DATA)
        {this is a step}
        else skip
    end if
forever



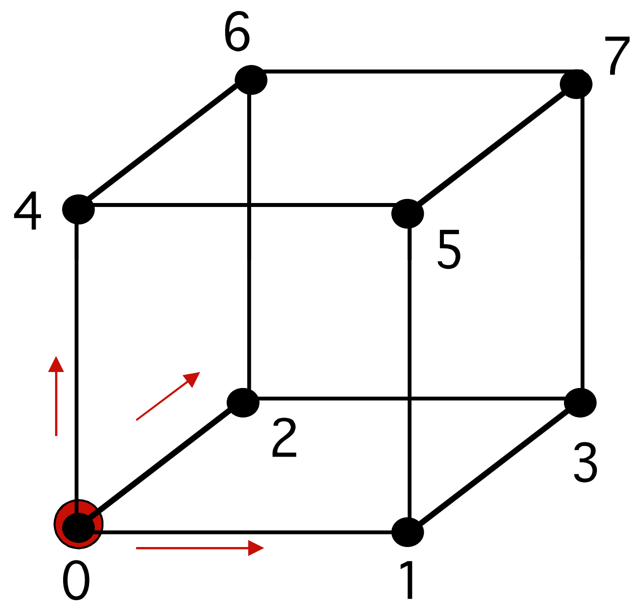
What is the time complexity?
(i.e. how many steps are needed?) → Arbitrarily large!

Broadcasting using shared memory

Now, use "**large atomicity**", where in one step, a process j reads the states of ALL neighbors with smaller id, and updates x[j] only when these are equal, but different from x[j].

What is the time complexity ?

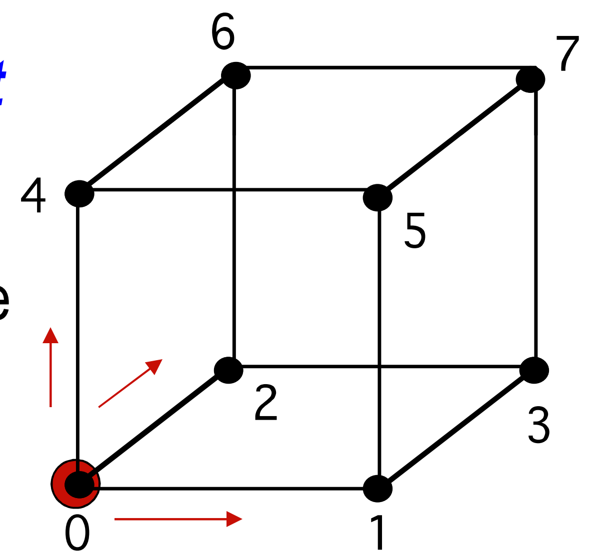
How many steps are needed ?



Time complexity in rounds

Rounds are truly defined for **synchronous systems**. An **asynchronous round** consists of a number of steps **where every process** **(including the slowest one)** **takes** ***at least one step*****.**

How many rounds will you need to complete the broadcast using the large atomicity model ?



An easier concept is that of synchronous processes executing their steps in lock-step synchrony

Graph Algorithms vs Distributed Algorithms

Graph Algorithms

- Why graph algorithms?
- Many problems in DS can be modeled as graph problems. Note that
  - The *topology* of a distributed system is a **graph**
  - *Routing table* computation uses the **shortest path algorithm**
  - *Efficient broadcasting* uses a **spanning tree**
  - *Maxflow* algorithm determines the **maximum flow** between a pair of nodes in a graph, etc.
  - *Reuse of frequencies* **in wireless networks** (“no“ interference) uses **Vertex coloring**