Lecture #2.

A Model of Distributed Computations

Distributed System Model:

- a set of processes connected by a communication network.

System Model:

- provides the facility to exchange information among processes.
- finite but unpredictable communication delay.
- no common global memory and communicate solely by passing messages.
- no physical global clock.
- messages may be delivered out-of-order, may be lost, garbled, or duplicated
- processors may fail
- communication links may go down.

⇒ System can be modeled as a graph
 vertices: processes.
 edges: links

A distributed Program.

is composed of a set of n asynchronous processes labelled

$$P_1, P_2, \ldots, P_n$$

WLOG: assume each P_i on a different processor.

C_{ij} : communication channel between P_i & P_j

m_{ij} : denotes a message sent from P_i to P_j.

Global state :

$$\underbrace{\{\text{states of } P_i\}}_{\text{local memory}} \cup \underbrace{\{\text{states of } C_{ij}\}}_{\text{messages in transit.}}$$

A model of distributed executions.

- the execution of a process consists of a <u>sequential</u> execution of its actions.

- actions are <u>atomic</u> and are of three types:
  1. Internal events
  2. message send events
  3. message receive events.

event e_i^x : x^{th} event at p_i.

msg m: send(m)
recv(m)

<u>Events</u> cause:

state transition of processes
state transition of channels
$\Rightarrow$ affect the global state.

Internal event: only affect the process.

send event: affects the sender and the channel.

receive event: affects the receiver and the channel.

events @ p_i: linear sequence.

$$e_i^1,\ e_2^i,\ \ldots,\ e_i^x,\ e_i^{x+1},\ \ldots$$

$$H_i = (h_i, \rightarrow_i)$$

ordering relation

$$e_i^x \rightarrow e_i^{x+1}.$$

$\rightarrow_i$: causal ordering or dependence.

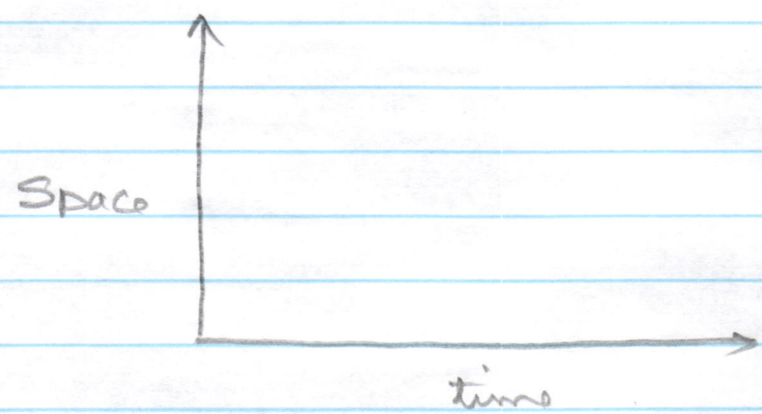
send and receive events capture the flow of information between processes

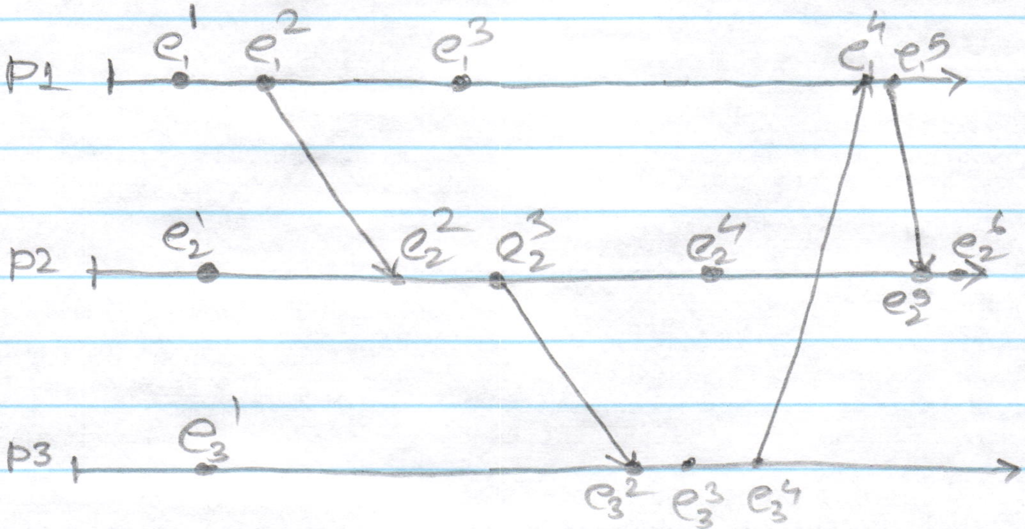
$\rightarrow_{msg}$: causal dependency due to messages.

For every m :

$$send(m) \longrightarrow_{msg} recv(m).$$

A natural way to visualize the evolution of distributed computation is to use space-time diagrams.





Causal Precedence Relation:

$$H = \bigcup_i h_i$$

Define a binary relation

$$(H, \rightarrow)$$

$\forall e_i^x, \forall e_j^y \in H \quad e_i^x \rightarrow e_j^y$

iff:

1. $e_i^x \rightarrow_i e_i^y$ (i.e., $i = j \wedge x < y$), or

2. $e_i^x \longrightarrow_{msg} e_j^y$, or

3. $\exists e_k^z$ s.t. $e_i^x \rightarrow e_k^z \wedge e_k^z \longrightarrow e_j^y$.

Called the happens-before relation.

Concurrent events:

e_i and e_j are concurrent $e_i \parallel e_j$

iff

$e_i \not\rightarrow e_j \wedge e_j \not\rightarrow e_i$

For any two events e_i + e_j:

either

$e_i \rightarrow e_j$

or

$e_j \rightarrow e_i$

or $e_i \parallel e_j$

Models of Communication NWs.

FIFO : channel is a queue

non-FIFO : channel is a set

Causal Order: channels satisfy happen-before

m_{ij} + m_{kj}

if $send(m_{ij}) \rightarrow send(m_{kj})$

then $recv(m_{ij}) \rightarrow recv(m_{kj})$.

Causally related messages destined to the same destination are delivered in an order consistent with the causality.

$CO \subset FIFO \subset NON\text{-}FIFO$.

CO considerably simplifies the design of distributed algorithms.

<u>e.g.</u> updates to replica

Global State of a distributed system.

$$\{\text{state of } p_i\} \cup \{\text{state of } C_i\}.$$

$\downarrow$

LS_i^x : state of p_i after e_i^x. (initial state: LS_i^0).

$$SC_{ij}^{x,y} = \{ m_{ij} \mid send(m_{ij}) < LS_i^x \wedge recv(m_{ij}) \not< LS_j^y \}$$

denotes all messages that P_i sent up to event e_i^x and P_j has not received until e_i^y.

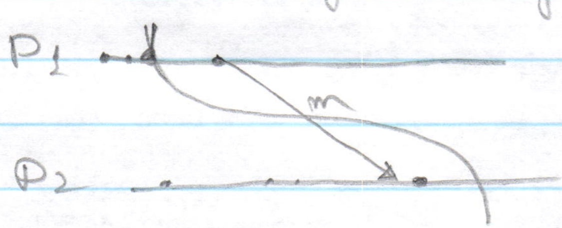
$$GS = \left\{ \bigcup_i LS_i^{x_i} , \bigcup_{j,k} SC_{jk}^{y_j, z_k} \right\}$$

Meaningful $\underline{GS}$: ~~all~~ states of all components must be recorded at a single instant.

$\Downarrow$

This is generally not possible.

So what can go wrong? Construct an example



$\Rightarrow$ P2 says m is recvd but P_1 has no memory of it being sent.

$\Rightarrow$ when a snapshot is taken

$\Downarrow$

not all snapshots be consistent.

GS is consistent iff

$$\forall m_{ij} : send(m_{ij}) \leq LS_i^{x_i} \Rightarrow m_{ij} \notin SC_{ij}^{x_i, y_j} \wedge recv(m_{ij}) \not\leq LS_j^{y_j}$$

$\boxed{\text{CUTS}}$