1DT066
DISTRIBUTED INFORMATION SYSTEM

Time, Coordination and Agreement

OUTLINE

- Time
  - Physical time
  - Logical time
- Coordination and agreement
- Multicast communication
- Summary

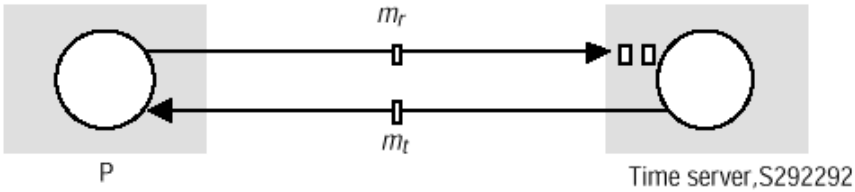
1 Time

- The notation of time
- External synchronization
- Internal synchronization
- Physical clocks and their synchronization
- Logical time and logical clocks

1.1 Synchronizing Physical Clocks

- Computer each contains its physical clock.
- Physical clock is limited by its resolution - the period between updates of the clock register.
- Clock drift often happens to physical clocks.
- To compensate for clock drifts, computers are synchronized to a time service, e.g., UTC - Coordinated universal time.
- Several other algorithms for synchronization.

1.1 Cristian's Clock Synchronization



- A process **P** can record the total round-trip time $\mathbf{T}_{round}$ taken to send the request $\mathbf{m}_r$ and receive the reply $\mathbf{m}_t$.
- A simple estimate of the time to which **P** should set its clock is $\mathbf{t} + \mathbf{T}_{round}/\mathbf{2}$.

1.1 The Berkeley Algorithm

- A coordinator computer is chosen to act as the *master*. Master periodically polls to *slaves* whose clocks are to be synchronized.
- The master estimates their local clock times by observing the round-trip times, and it averages the values obtained.
- The master takes a fault-tolerant average.
- Should the master fail, then another can be elected to take over.

1.1 The Network Time Protocol

- NTP distributes time information to provide:
  - a service to synchronize clients in Internet
  - a reliable service that survives loss of connection
  - a frequent resynchronization for client's clock drift
  - protection against interference with time server
- NTP service is provided by various servers:
  - Primary servers, secondary servers, and servers of other levels (called *strata*).
- Synchronization subnet: the servers which are connected in a logical hierarchy.

1.2 Logical Time and Logical Clocks

- The order of the events
  - two events occurred in the order they appear in a process.
  - event of sending occurred before event of receiving.
- happened-before relation, denoted by ->

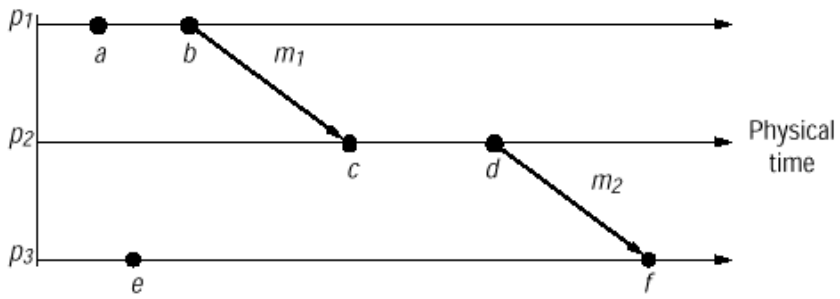
HB1: If some process p: $x \rightarrow_p y$, then $x \rightarrow y$.

HB2: For any message m, $send(m) \rightarrow rcv(m)$,

HB3: If x, y and z are events such that $x \rightarrow y$ and $y \rightarrow z$, then $x \rightarrow z$.

1.2 Logical Timestamps Example

- Events occurring at three processes



1.2 Lamport Logical Timestamps

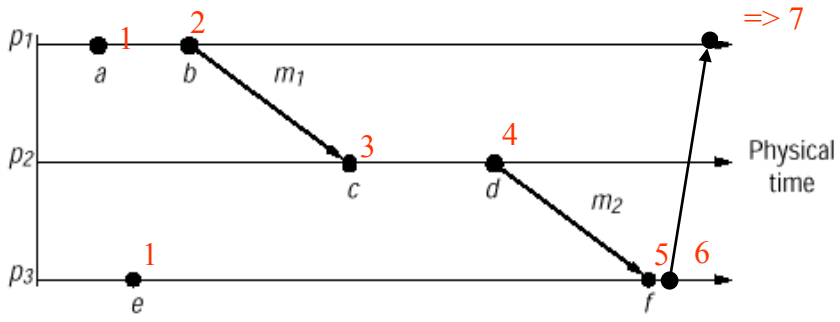
- Logical clock - a monotonically increasing software counter.
- C_p: logical clock for process p; $C_p(a)$: timestamp of event a at p; $C(b)$: timestamp of event b
- *LC1*: event issued at process p: $C_p := C_p + 1$

  LC2: a) p sends message m *to* q with value $t = C_p$

  b) $C_q := \max(C_q, t)$ and applies *LC1* to $rcv(m)$.
- If $a \rightarrow b$ then $C(a) < C(b)$, but not visa versa!
- Total order logical clock and vector clock.

1.2 Lamport Timestamps Example

- Events occurring at three processes

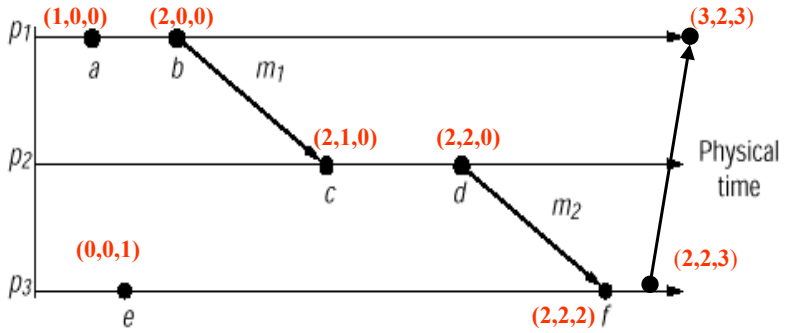


1.3 Vector Clocks

- Vector clock
  - A vector clock of N processes is an array of N integers
  - Each process keeps its own vector clock V_i, which it uses to timestamp a local event
- **VC1:** Initially, $V_i[j] = 0$, for $i, j = 1, 2..., N$.
- **VC2:** Just before p_i timestamps an event, it sets $V_i[i] := V_i[i] + 1$.
- **VC3:** p_i includes the value $t = V_i$ in every message it sends.
- **VC4:** When p_i receives a timestamp t in a message, it sets $V_i[j] := max(V_i[j], t[j])$, for $j = 1, 2..., N$. Taking the component-wise maximum of two vector timestamps in this way is known as a merge operation.

1.3 Vector Clocks Example

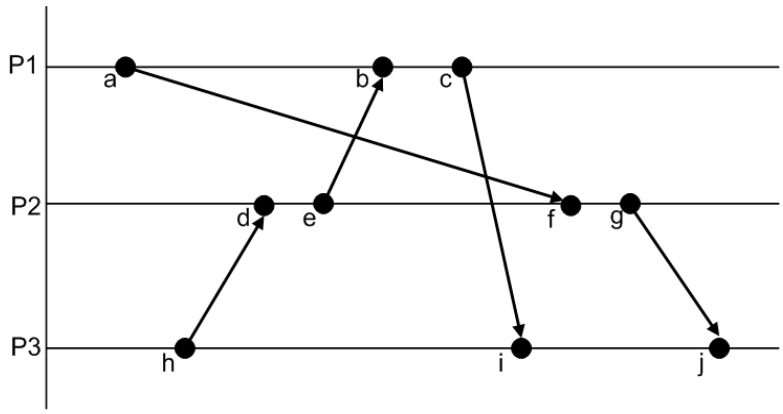
- Events occurring at three processes



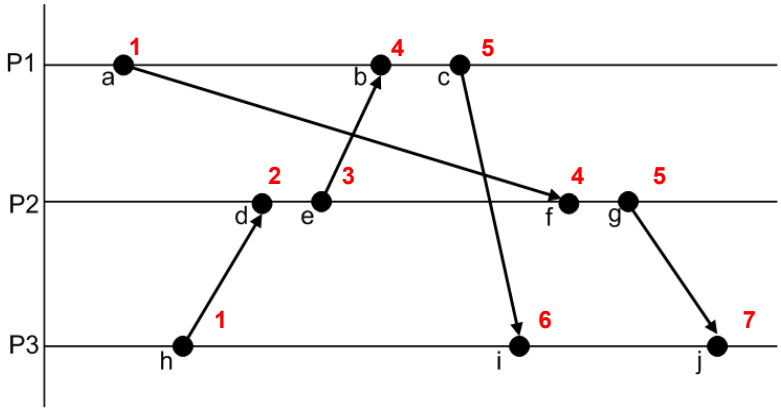
1.4 Comparison

- In Lamport's clock, C(e)<C(e') does not imply e ->e'; while in Vector timestamp, V(e)<V(e') implies e ->e'.
- Vector timestamps take up an amount of storage and message payload that is proportional to N, the number of process; while Lamport's clock does not.

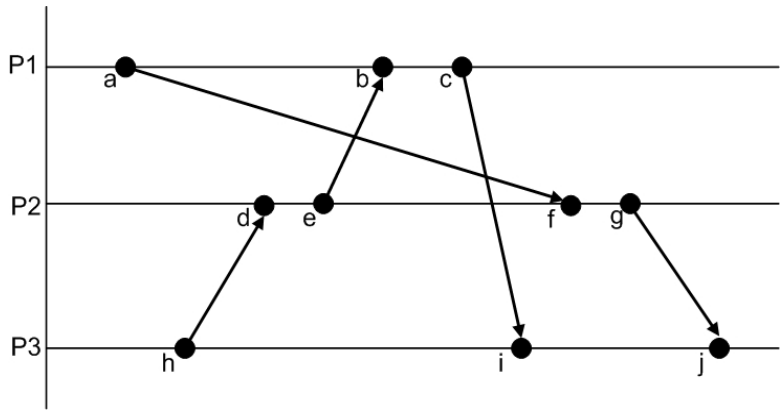
1.5 Lamport Timestamps Exercise



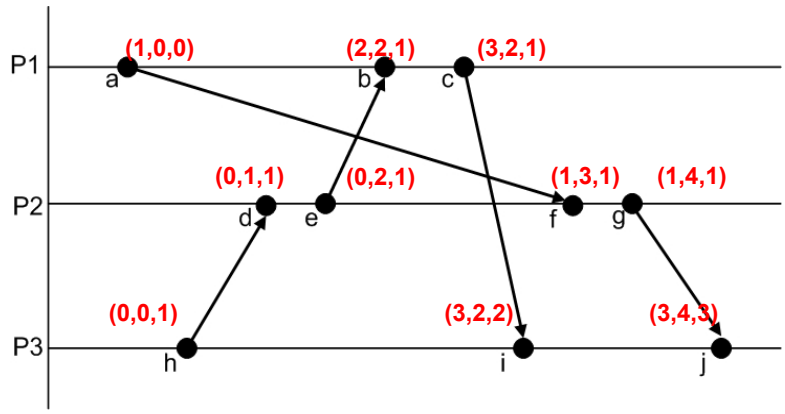
1.5 Lamport Timestamps Answer



1.5 Vector Clocks Exercise



1.5 Vector Clocks Answer



2 Coordination

- Distributed processes need to coordinate their activities.
- Distributed mutual exclusion is required for safety, liveness, and ordering properties.
- Election algorithms: methods for choosing a unique process for a particular role.

2.1 Distributed Mutual Exclusion

- The basic requirements for mutual exclusion:
  - ME1 (safety): At most one process may execute in the critical section (CS) at a time.
  - ME2 (liveness): A process requesting entry to the CS is eventually granted.
  - ME3 (ordering): Entry to the CS should be granted in happened-before order.
- The central server algorithm.
- A ring-based algorithm.
- A distributed algorithm using logical clocks.

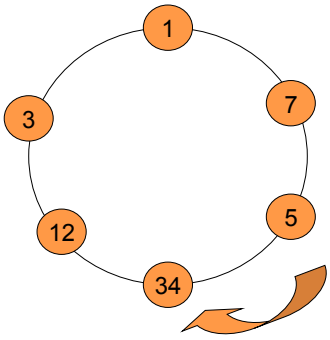
2.2 Elections

- An election is a procedure carried out to choose a process from a group.
- A ring-based election algorithm.
- The bully algorithm.

2.2.1 Ring-Based Election Algorithm

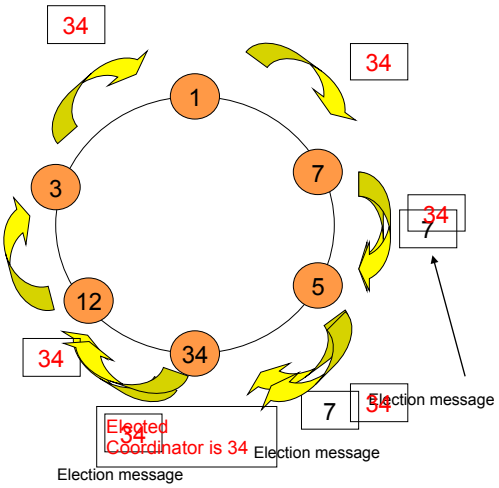
- Each process P(i) has a communication channel to the next process P(i+1) mod N.
- Messages are sent clockwise.
- The goal is to elect a single process called the coordinator, which is the process with the largest identifier.

2.2.1 Ring-Based Algorithm



Process number	status
1	Non-participant
3	Non-participant
5	Non-participant
7	Non-participant
12	Non-participant
34	Non-participant

2.2.1 Ring-Based Algorithm



Process number	status
1	Non-participant
3	Non-participant
5	Non-participant
7	Non-participant
12	Non-participant
34	Non-participant

- Every process can begin an election
- A process begins an election by marking itself as a participant, and sends an election message to its neighbor by placing its identifier
- Suppose process 7 now begins the election

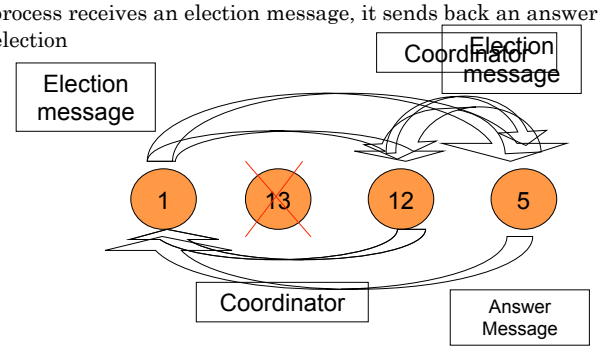
2.2.2 Bully Algorithm

- The processes themselves are synchronous. I.e. they use timeouts to detect a process failure.
- Unlike the ring-based algorithm in which processes only know their neighbors, bully algorithm allows processes to know those processes with a higher identifier.
- There are three types of message:
  - Election
  - Answer
  - Coordinator

Coordinator is now 13, because it has the highest identifier

2.2.2 Bully Algorithm

- The election begins when a process notices that the coordinator is failed.
- Several processes may discover this concurrently
- A process which detects the failure will send an election message to those with a higher identifier
- When a process receives an election message, it sends back an answer message and begins another election

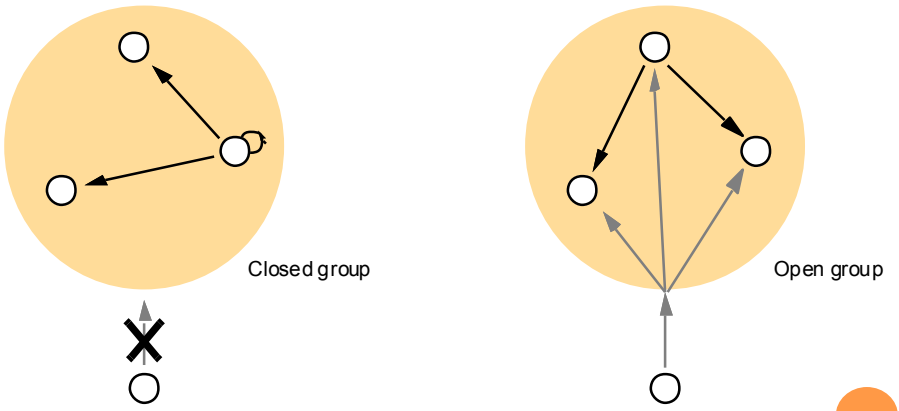


Process 12 will know that it is the highest identifier now as all its higher identifier process (i.e. process 13) have failed, this process will then send back the coordinator message to all its lower identifier process.

3 Multicast Communication

- Group (multicast) communication requires coordination and agreement.
- One multicast operation is much better than multiple send operation in terms of *efficiency* and *delivery guarantees*.
- Basic multicast: guarantees a correct process will eventually deliver the message.
- Reliable multicast: requires that all correct processes in the group must receive a message if any of them does.

3.1 Open and Closed Groups



3 Bulletin Board Example

Bulletin board: os.interesting		
Item	From	Subject
23	A.Hanlon	Mach
24	G.Joseph	Microkernels
25	A.Hanlon	Re: Microkernels
26	T.L'Heureux	RPC performance
27	M.Walker	Re: Mach
end		

Bulletin board: os.interesting		
Item	From	Subject
20	G.Joseph	Microkernels
21	A.Hanlon	Mach
22	A.Sahiner	Re: RPC performance
23	M.Walker	Re: Mach
24	T.L'Heureux	RPC performance
25	A.Hanlon	Re: Microkernels
end		

3.2 Consistency and Request Ordering

- Criteria: correctness vs. expenses.
- Total, causal, and FIFO ordering requirements.
- Implementing request ordering.
- Implementing total ordering.
- Implementing causal ordering with vector timestamps.

3.2.1 Total, FIFO, Causal Ordering

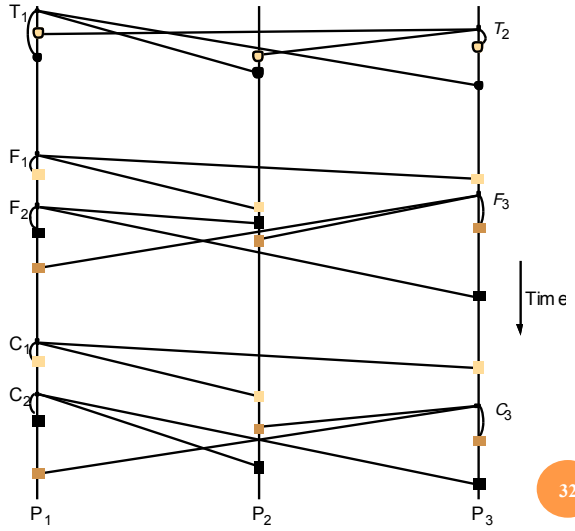
- Let m_1 and m_2 be messages delivered to the group.
- Total ordering: Either m_1 is delivered before m_2 or m_2 is delivered before m_1, at all processes.
- Causal ordering: If m_1 happened-before m_2 then m_1 is delivered before m_2 at all processes.
- FIFO ordering: If m_1 is issued before m_2 then m_1 is delivered before m_2 at all processes.

3.2.1 Ordering of Multicast Messages

Totally ordered messages
T_1 ,T_2 and F_1,

FIFO-related messages
F_1 and F_2 ; C_1 and C_2

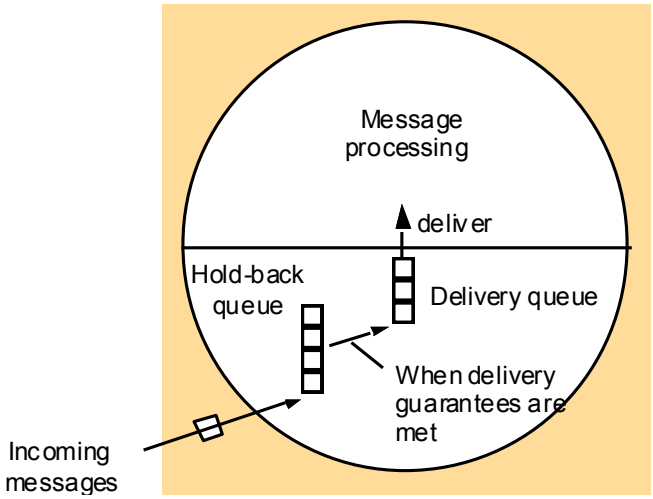
Causally related messages
C_1 and C_3 (assuming C_3 is a reply to C_1 at P_3)



3.2.2 Implementing Message Ordering

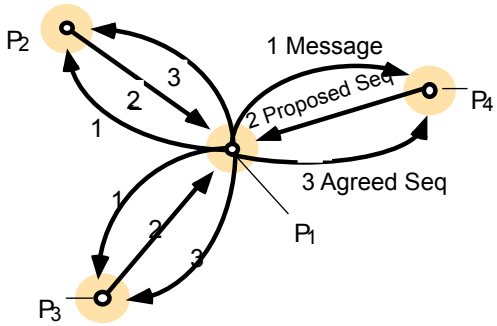
- Hold-back: A received message is not delivered until ordering constraints can be met.
- Stable message: all prior messages processed.
- Hold-back queue vs. delivery queue.
- Safety property: no message will be delivered out of order by being prematurely transferred.
- Liveness property: no message should wait on the hold-back queue forever.

3.2.2 The Hold-Back Queue



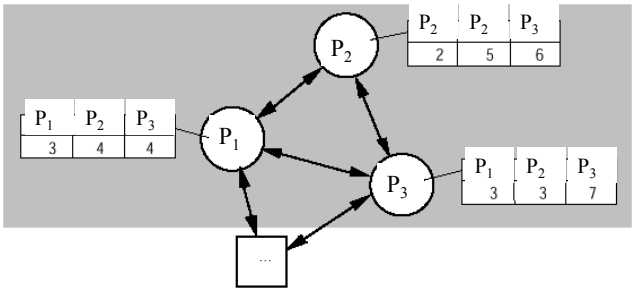
3.2.3 Implementing Total Ordering

- Basic approach: assign totally ordered sequence to messages.
  - Sequencer
  - Distributed agreement in assigning sequence.



3.2.4 Implementing Causal Ordering

- Vector timestamp: a list of counts of update events, one for each of the processes.
- Merging vector timestamps: choose the largest values from the two vectors, component-wise.



3.3 Multicast Communication

- Sample multicasting operation

```
void Multicast (in orderType order, in
groupId group, in msg m, in int
nReplies, out msgSeq replies) raises
(…);
```

- Order types:
  - unordered
  - total ordering
  - causal ordering
  - sync-ordering

4 Summary

- Timing issues
  - Synchronizing physical clocks.
  - Logical time and logical clocks.
- Distributed coordination and mutual exclusions.
- Coordination.
- Elections
  - Ring-based algorithm
  - Bully algorithm
- Multicast communication.
- Read textbook Chapters 14 and 15.