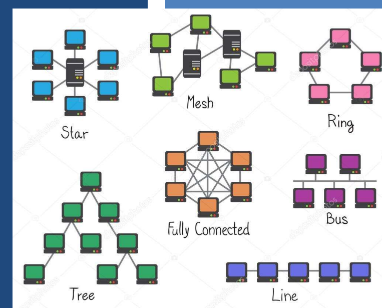
TCSS 558: APPLIED DISTRIBUTED COMPUTING



Chapter 6 - Coordination

Wes J. Lloyd
School of Engineering and Technology
University of Washington - Tacoma

1

OBJECTIVES

- Assignment 2 - questions
- Feedback from 2/27
- Chapter 6: Coordination
- Chapter 6.2: Logical Clocks
  Vector Clocks
- Class Activity – Total Ordered Multicasting
- Chapter 6.3: Distributed Mutual Exclusion

2

MATERIAL / PACE

- Please classify your perspective on material covered in today's class:
- 1-mostly review, 5-equal new/review, 10-mostly new
- <u>Average – 7.6</u>

- Please rate the pace of today's class:
- 1-slow, 5-just right, 10-fast
- <u>Average – 6.1</u>

FEEDBACK FROM 2/27

- ***<u>With regards to locking a key/value pair, do you mean that I have to prevent other nodes from sending in a "del" command to the transaction leader server node when the key/value pair is "locked"?</u>***
  - For "del" command, leader sends ddel1 to every node
  - ***<u>Can the leader ensure other nodes don't send a "ddel1" request?</u>***

- When a leader starts a transaction they start by locking the key, and sending "dput1" or "ddel1" to all known nodes.

- If the leader receives a "ddel1" or "dput1" for the same key while it is locked, they reject it (ABORT). Sending ABORT to a leader cancels the second transaction across the distributed system.

- The original transaction continues to be processed.
- If every node acknowledges the "dput1/ddel1", the leader proceeds to "dput2" or "ddel2" to commit data changes to every node

FEEDBACK - 2

- ***How does locking work in the two-phase commit protocol?***
- The first phase establishes locks the key/value pair at every node
- Every node sends the transaction leader an ACK (acknowledgement message)
- If even just one node sends an ABORT, the leader will send dputabort (ddelabort) to all nodes to cancel the transaction
- The first phase causes the key/value pair to become globally locked across the distributed system once complete
- During the second phase, the transaction is committed (data changes are written) at every node.

FEEDBACK - 3

- One weakness in our protocol for assignment #2 is that we don't support aborting the transaction in the second phase.
- When the leader sends dput2 to every node, we assume that every node will successfully make the commit.
- ***How could the two-phase commit protocol be modified to abort a transaction that fails during dput2?***

SHORT-HAND-CODES FOR MEMBERSHIP TRACKING APPROACHES

- Include readme.txt or doc file with instructions in submission
- Must document membership tracking method
- **S-1:** Static file membership tracking only = 0 pts
- **T-1:** TCP membership tracking only = +5 pts (*should be dynamic once servers point to membership server*)
- **U-1:** UDP membership tracking only = +10 pts (*automatically discovers nodes with no configuration*)
- **S+T-2:** Static file + TCP membership tracking = +15 pts (*Static file is not reread to refresh membership during operation*)
- **S+U-2:** Static file + UDP membership tracking = +15 pts (*Static file is not reread to refresh membership during operation*)
- **SD+U-2:** Static file + UDP membership tracking = +20 pts (*Static file is periodically reread to refresh membership during operation*)
- **T+U-2:** TCP + UDP membership tracking = 20 pts (*both dynamic*)

March 3, 2020	TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma	L16.7

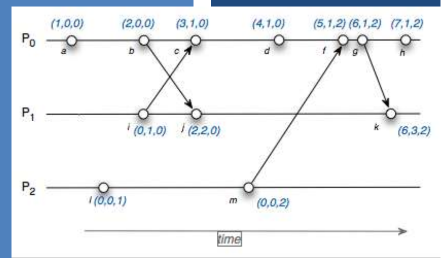
7

CHAPTER 6 - COORDINATION

- 6.1 Clock Synchronization
  - Physical clocks
  - Clock synchronization algorithms
- 6.2 Logical clocks
  - Lamport clocks
  - Vector clocks
- 6.3 Mutual exclusion
- 6.4 Election algorithms
- 6.6 Distributed event matching (*light*)
- 6.7 Gossip-based coordination (*light*)

March 3, 2020	TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma	L16.8

8



CH. 6.2: LOGICAL CLOCKS

L16.9

9

LOGICAL CLOCKS

- In distributed systems, synchronizing to actual time may not be required…
- It may be sufficient for every node to simply agree on a current time (e.g. logical)
- ***Logical clocks*** provide a mechanism for capturing chronological and ***causal*** relationships in a distributed system
- Think ***counters . . .***
- Leslie Lamport [1978] seminal paper showed that absolute clock synchronization often is not required
- Processes simply need to agree on the order in which events occur

March 3, 2020	TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma	L16.10

10

LOGICAL CLOCKS - 2

- **Happens-before relation**
- A→B: **Event A**, happens before **event B**…
- All processes must agree that **event A** occurs first
- Then afterward, **event B**
- Actual time not important. . .

- If **event A** is the event of proc P1 sending a msg to a proc P2, and **event B** is the event of proc P2 receiving the msg, then A→B is also true. . .
- The assumption here is that message delivery takes time
- Happens before is a ***transitive relation***:
- A→B, B→C, therefore A→C

March 3, 2020	TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma	L16.11

11

LOGICAL CLOCKS – 3

- If two events, say event X and event Y do not exchange messages, not even via third parties, then the sequence of X→Y vs. Y→X **can not be determined!!**

- Within the system, these events appear **concurrent**

- **Concurrent:** nothing can be said about when the events happened, or which event occurred first

- Clock time, C, must always go forward (increasing), never backward (decreasing)

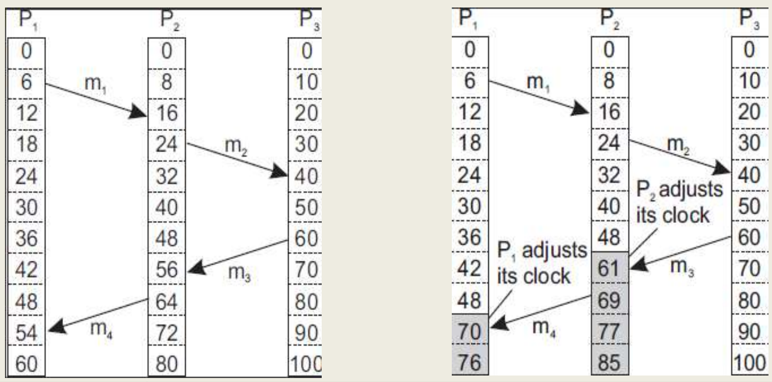
- Corrections to time can be made by adding a positive value, but never by subtracting one

March 3, 2020	TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma	L16.12

12

LOGICAL CLOCKS - 4

- Three processes each with local clocks
- ***Lamport's algorithm*** corrects process clock values
- Always propagate the most recent known value of logical time

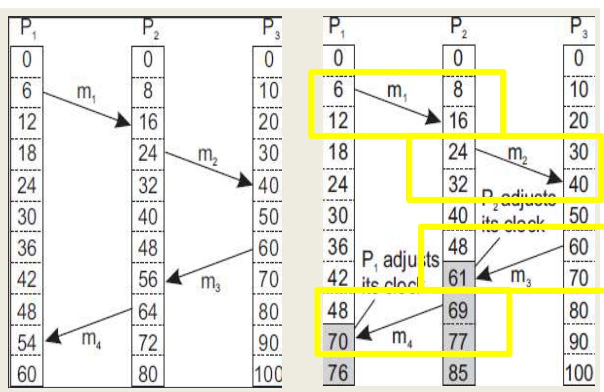


13

LOGICAL CLOCKS

- **Events:**

6: P1 send m1 to P2
16: P2 receives m1
24: P2 sends m2 to P3
40: P3 receives m2
60: P3 sends m3 to P2
56: P2 receives m3
56: P2 clock reset=61
64: P2 sends m4 to P1
54: P1 receives m4
70: P1 clock reset=70



14

LAMPORT LOGICAL CLOCKS - IMPLEMENTATION

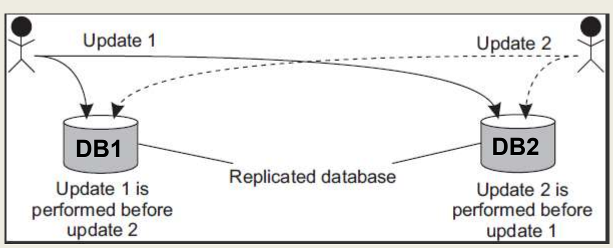
- Negative values not possible
- When a message is received, and the local clock is before the timestamp when then message was sent, the local clock is updated to message_sent_time + 1

1. Clock is incremented before an event: sending a message, receiving a message, some other internal event
   Pi increments Ci: Ci ← Ci + 1
2. When Pi send msg m to Pj, m's timestamp is set to Ci
3. When Pj receives msg m, Pj adjusts its local clock
   Cj ← max{Cj, timestamp(m)}
4. Ties broken by considering Proc ID: i<j; <40,i> < <40,j>
   Both Lamport clocks are = 40
   The winner has a higher alphanumeric Process ID
   J (winner) is greater than i, alphabetically

15

TOTAL-ORDERED MULTICASTING

- Consider concurrent updates to a replicated database
- Communication latency between DB1 and DB2 is 250ms



- <u>Initial Account balance</u>: $1,000
- <u>Update #1</u>: Deposit $100
- <u>Update #2</u>: Add 1% Interest
- Total Ordered Multicasting needed

16

TOTAL-ORDERED MULTICASTING EXAMPLE

- Two messages (m_1, m_2) must be distributed, to two processes (p_1, p_2)
- We assume messages have correct lamport clock timestamps
- m_1(10, p_1, add $100)
- m_2(12, p_2, add 1% interest)
- Each process maintains a queue of messages
- Arriving messages are placed into queues ordered by the Lamport clock timestamp
- In each queue, each message must be acknowledged by every process in the system before operations can be applied to the local database

March 3, 2020	TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma	L16.17

17

TOTAL-ORDERED MULTICASTING EXAMPLE

- Two messages (m_1, m_2) must be distributed, to two processes (p_1, p_2)
- We assume messages have correct lamport clock timestamps
- m_1(10, p_1, add $100)

Key point:

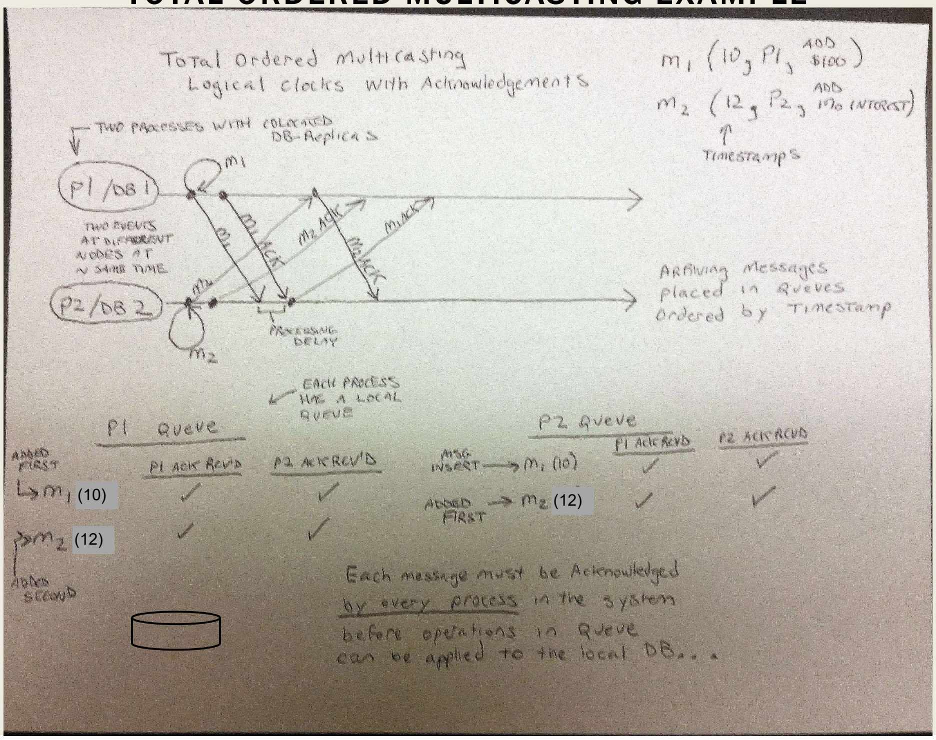
Multicast messages are also received by the sender (*itself*)

- Arriving messages are placed into queues ordered by the Lamport clock timestamp
- In each queue, each message must be acknowledged by every process in the system before operations can be applied to the local database

March 3, 2020	TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma	L16.18

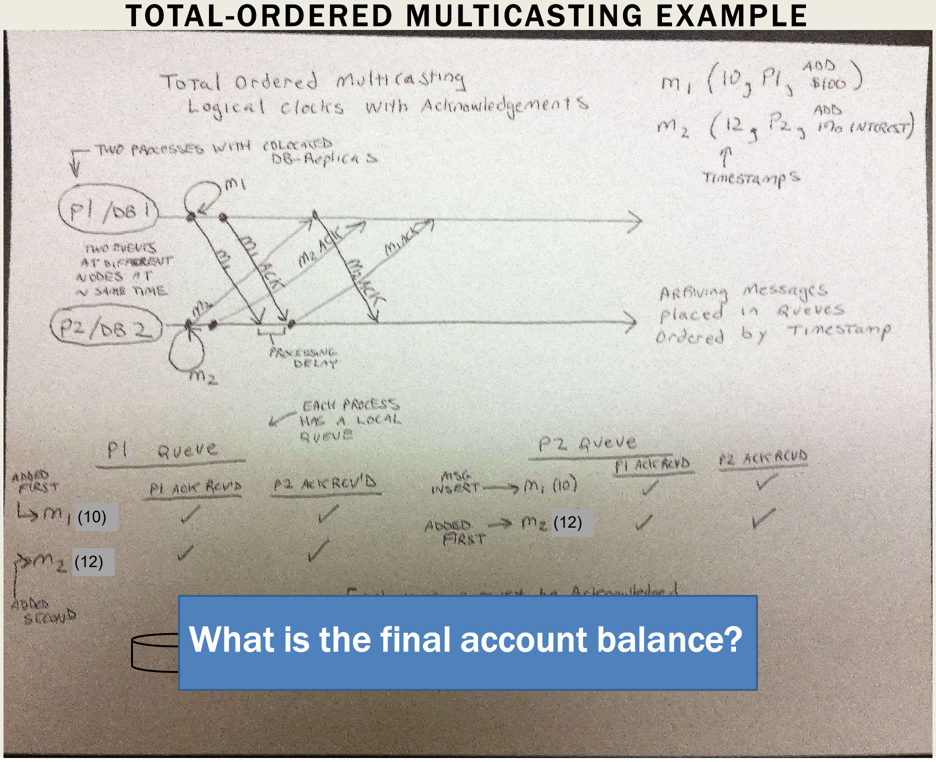
18

TOTAL-ORDERED MULTICASTING EXAMPLE



19

TOTAL-ORDERED MULTICASTING EXAMPLE



20

TOTAL-ORDERED MULTICASTING - 2

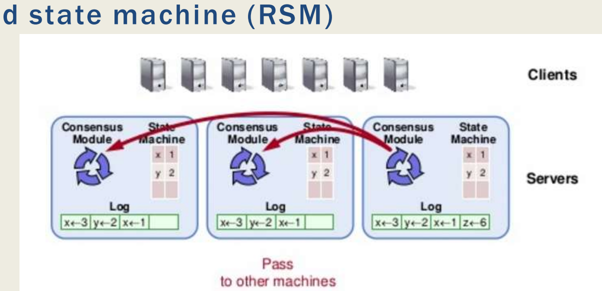
- Each message timestamped with local logical clock of sender
- **Multicast messages are also received by the sender (*itself*)**
- Assumptions:
  - Messages from same sender received in order they were sent
  - No messages are lost
- When messages arrive they are placed in local queue ordered by timestamp
- Receiver ***multicasts*** acknowledgement of message receipt to other processes
  - Time stamp of message receipt is lower the acknowledgement
- This process ***replicates*** queues across sites
- Messages delivered to application (database) only when message at the head of the queue has been acknowledged by ***every*** process in the system

March 3, 2020	TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma	L16.21

21

TOTAL-ORDERED MULTICASTING - 3

- Can be used to implement replicated state machines (RSMs)
- Concept is to replicate event queues at each node
- (1) ***Using logical clocks*** and (2) ***exchanging acknowledgement messages***, allows for events to be ***"totally"*** ordered in replicated event queues
- Events can be applied **"in order"** to each (distributed) replicated state machine (RSM)



March 3, 2020	TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma	L16.22

22

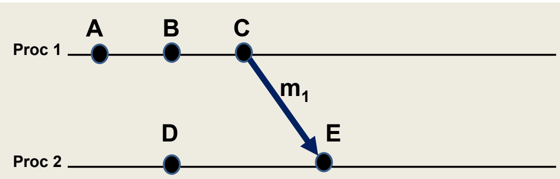
VECTOR CLOCKS

- Lamport clocks don't help to determine causal ordering of messages
- Vector clocks capture causal histories and can be used as an alternative
- But what is causality? …

March 3, 2020 | TCSS558: Applied Distributed Computing [Winter 2020]
School of Engineering and Technology, University of Washington - Tacoma | L16.23

23

WHAT IS CAUSALITY?



- Having a causal relationship between two events (A and E) indicates that event E results from the occurrence of event A.
- When one event results from another, there is a causal relationship between the two events.
- This is also referred to as cause and effect.

March 3, 2020 | TCSS558: Applied Distributed Computing [Winter 2020]
School of Engineering and Technology, University of Washington - Tacoma | L16.24

24

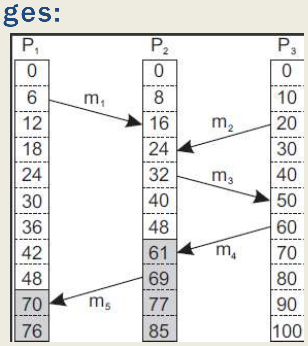
CAUSALITY - 2

- **Disclaimer:**
- Without knowing actual information contained in messages, it is not possible to state with certainty that there is a causal relationship or perhaps a conflict
- Lamport/Vector clocks can help us suggest possible causality
- But we never know for sure…

25

CAUSALITY - 3

- Consider the messages:



- P2 receives m1, and subsequently sends m3
- **Causality:** Sending m3 ***may*** depend on what's contained in m1
- P2 receives m2, receiving m2 is ***not*** related to receiving m1
- ***Is sending m3 causally dependent on receiving m2?***

26

VECTOR CLOCKS

- Vector clocks help keep track of <u>causal history</u>
- If two local events happened at process P, then the causal history H(p2) of event p2 is {p1,p2}

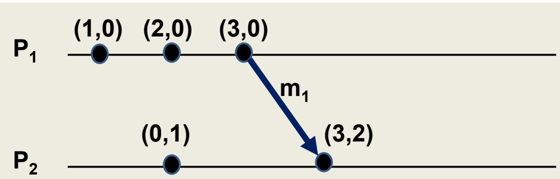
- P sends messages to Q (*event p3*)
- Q previously performed event q1
- Q records arrival of message as q2
- Causal histories merged at Q H(q2)= {p1,p2,p3,q1,q2}

- Fortunately, can simply store history of last event, as a vector clock → H(q2) = (3,2)
- Each entry corresponds to the last event at the process

March 3, 2020	TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma	L16.27

27

VECTOR CLOCKS - 2



- Each process maintains a vector clock which
  - Captures number of events at the local process (e.g. logical clock)
  - Captures number of events at all other processes
- Causality is captured by:
  - For each event at Pi, the vector clock (VC_i) is incremented
  - The msg is timestamped with VC_i; and sending the msg is recorded as a new event at P_i
  - P_j adjusts its VC_j choosing the <u>max</u> of: the message timestamp –or- the local vector clock (VC_j)

March 3, 2020	TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma	L16.28

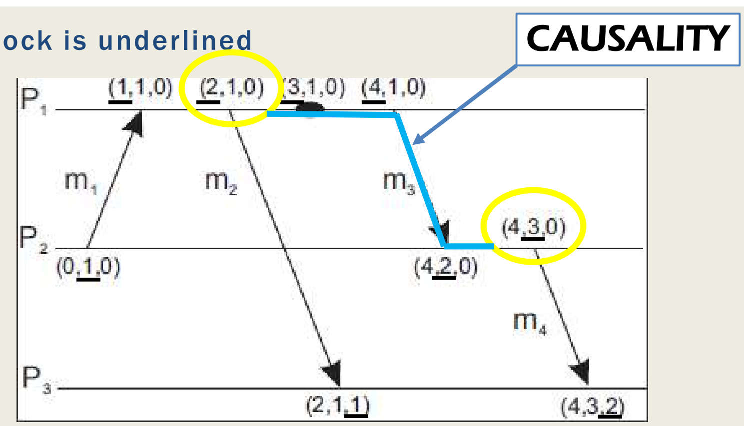
28

VECTOR CLOCKS - 3

- Pj knows the # of events at Pi based on the timestamps of the received message
- Pj learns how many events have occurred at other processes based on timestamps in the vector
- These events *"may be causally dependent"*
- **In other words:** they may have been necessary for the message(s) to be sent…

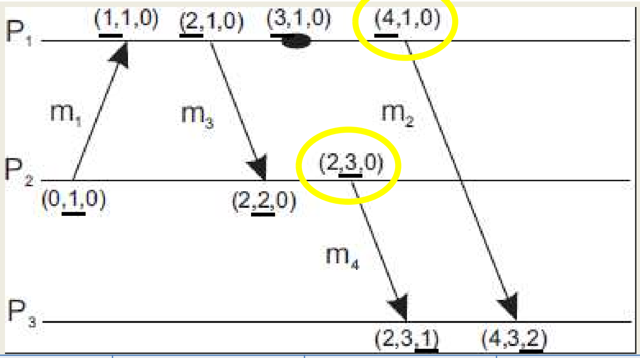
VECTOR CLOCKS EXAMPLE

- Local clock is underlined



ts (m_2)	ts(m_4)	ts(m_2)<ts(m_4)	ts(m_2)>ts(m_4)	Conclusion
(2,1,0)	(4,3,0)	Yes	No	m2 may causally precede m4

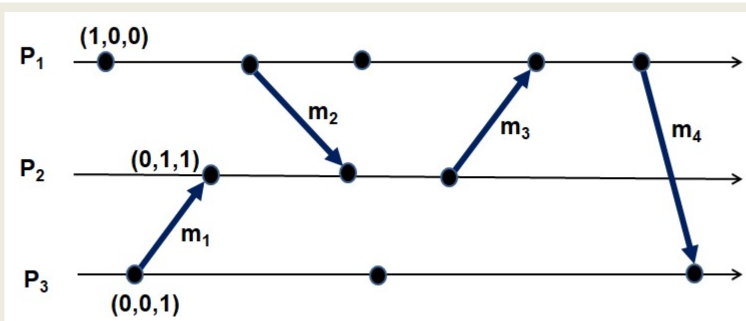
VECTOR CLOCKS EXAMPLE - 2



ts (m_2)	ts(m_4)	ts(m_2)<ts(m_4)	ts(m_2)>ts(m_4)	Conclusion
(4,1,0)	(2,3,0)	No	No	m2 and m4 may conflict

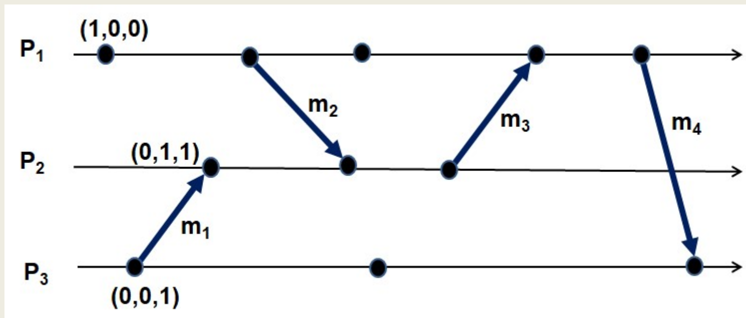
- P3 can't determine if m4 may be causally dependent on m2
- Is m4 causally dependent on m3 ?

VECTOR CLOCKS EXAMPLE - 3



- Provide a vector clock label for unlabeled events

VECTOR CLOCKS EXAMPLE - 4

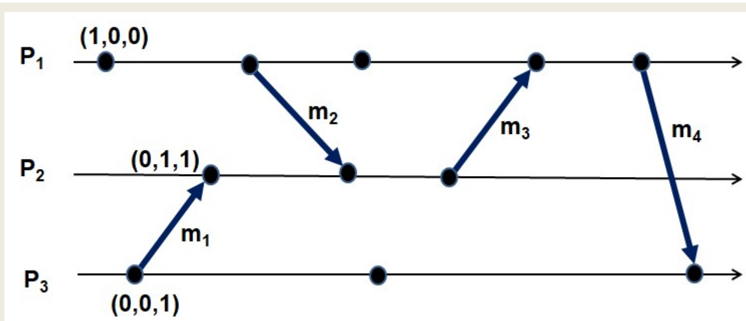


- TRUE/FALSE:
- The sending of message m_3 is causally dependent on the sending of message m_1.
- The sending of message m_2 is causally dependent on the sending of message m_1.

March 3, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L16.33

33

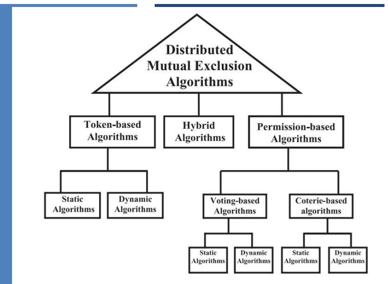
VECTOR CLOCKS EXAMPLE - 5



- TRUE/FALSE:
- P_1 (1,0,0) and P_3 (0,0,1) may be concurrent events.
- P_2 (0,1,1) and P_3 (0,0,1) may be concurrent events.
- P_1 (1,0,0) and P_2 (0,1,1) may be concurrent events.

March 3, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L16.34

34



CH. 6.3: DISTRIBUTED MUTUAL EXCLUSION

L16.35

35

DISTRIBUTED MUTUAL EXCLUSION ALGORITHMS

- Coordinating access among distributed processes to a shared resource requires **Distributed Mutual Exclusion**
- **Algorithms in 6.3**
- Token-ring algorithm
- ***Permission-based algorithms:***
- Centralized algorithm
- Distributed algorithm (Ricart and Agrawala)
- Decentralized voting algorithm (Lin et al.)

March 3, 2020	TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma	L16.36

36

TOKEN-BASED ALGORITHMS

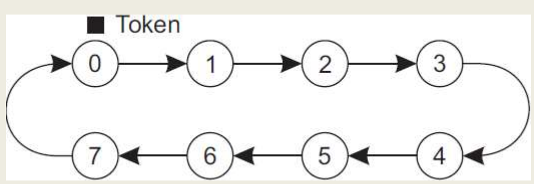
- Mutual exclusion by passing a "token" between nodes
- Nodes often organized in ring
- Only one token, holder has access to shared resource
- Avoids starvation: *everyone gets a chance to obtain lock*
- Avoids deadlock: easy to avoid

March 3, 2020	TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma	L16.37

37

TOKEN-RING ALGORITHM

- Construct overlay network
- Establish logical ring among nodes



- Single token circulated around the nodes of the network
- Node having token can access shared resource
- If no node accesses resource, token is constantly circulated around ring

March 3, 2020	TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma	L16.38

38

TOKEN-RING CHALLENGES

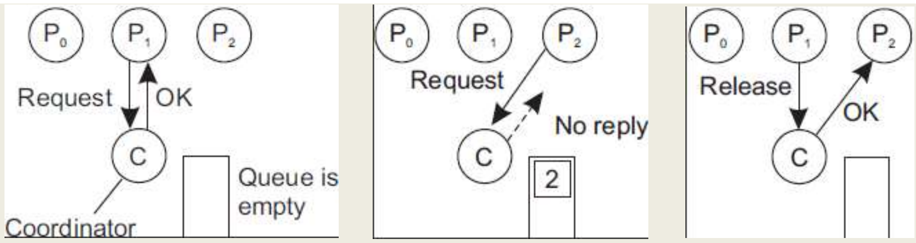
1. If token is lost, token must be regenerated
   - **Problem**: may accidentally circulate multiple tokens
2. Hard to determine if token is lost
   - What is the difference between token being lost and a node holding the token (***lock***) for a long time?
3. When node crashes, circular network route is broken
   - Dead nodes can be detected by adding a receipt message for when the token passes from node-to-node
   - When no receipt is received, node assumed dead
   - Dead process can be "jumped" in the ring

DISTRIBUTED MUTUAL EXCLUSION ALGORITHMS - 3

- **Permission-based algorithms**
- Processes must require permission from other processes before first acquiring access to the resource
  - CONTRAST: Token-ring did not ask nodes for permission
- **Centralized algorithm**
- Elect a single leader node to coordinate access to shared resource(s)
- Manage mutual exclusion on a distributed system similar to how it mutual exclusion is managed for a single system
- Nodes must all interact with leader to obtain ***"the lock"***

CENTRALIZED MUTUAL EXCLUSION

Permission granted from coordinator ∨ No response from coordinator



P_1 executes **P_2 blocks** **P_1 finishes; P_2 executes**

- When resource not available, coordinator can block the requesting process, or respond with a reject message
- P2 must ***poll*** the coordinator if it responds with reject otherwise can wait if simply blocked
- Requests granted permission fairly using FIFO queue
- Just three messages: (request, grant (OK), release)

CENTRALIZED MUTUAL EXCLUSION - 2

- **Issues**
- Coordinator is a single point of failure
- Processes can't distinguish dead coordinator from ***"blocking"*** when resource is unavailable
  - No difference between CRASH and Block (*for a long time*)
- Large systems, coordinator becomes performance bottleneck
  - Scalability: Performance does not scale

- **Benefits**
- Simplicity:
  Easy to implement compared to distributed alternatives

DISTRIBUTED ALGORITHM

- Ricart and Agrawala [1981], use total ordering of all events
  - Leverages Lamport logical clocks
- Package up resource request message (AKA Lock Request)
- Send to all nodes
- Include:
  - Name of resource
  - Process number
  - Current (logical) time
- Assume messages are sent reliably
  - No messages are lost

43

DISTRIBUTED ALGORITHM - 2

- <u>When each node receives a request message they will:</u>

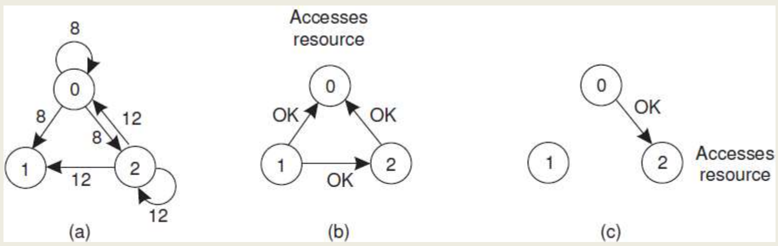
1. Say OK (*if the node doesn't need the resource*)
2. Make <u>no reply</u>, queue request (*node is using the resource)*
3. ***If node is also waiting to access the resource:*** perform a timestamp comparison -
   1. Send OK if requester has lower logical clock value
   2. Make <u>no reply</u> if requester has higher logical clock value

- Nodes sit back and wait for all nodes to grant permission
- Requirement: every node must know the entire membership list of the distributed system

44

DISTRIBUTED ALGORITHM - 3

- Node 0 and Node 2 simultaneously request access to **resource**
- Node 0's time stamp is lower (8) than Node 2 (12)
- Node 1 and Node 2 grant Node 0 access
- Node 1 is not interested in the resource, it OKs both requests



- **In case of conflict, lowest timestamp wins!**
  - Node 2 rejects its own request (1@) in favor of node 0 (8)

March 3, 2020	TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma	L16.45

45

CHALLENGES WITH DISTRIBUTED ALGORITHM

- **Problem:** Algorithm has N points of failure !
- Where N = Number of Nodes in the system

- **No Reply Problem:** When node is accessing the resource, it does not respond
  - Lack of response can be confused with **failure**
  - ***Possible Solution:*** When node receives request for resource it is accessing, always send a reply either granting or denying permission (ACK)
  - Enables requester to determine when nodes have died

March 3, 2020	TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma	L16.46

46

CHALLENGES WITH DISTRIBUTED ALGORITHM - 2

- **Problem**: Multicast communication required –or- each node must maintain full group membership
  - Track nodes entering, leaving, crashing…
- **Problem**: Every process is involved in reaching an agreement to grant access to a shared resource
  - This approach ***may not scale*** on resource-constrained systems
- **Solution**: Can relax total agreement requirement and proceed when a **simple majority** of nodes grant permission
  - *Presumably any one node locking the resource prevents agreement*
  - *If one node gets majority of acknowledges no other can*
  - *Requires every node to know size of system (# of nodes)*
- Distributed algorithm for mutual exclusion works best for:
  - Small groups of processes
  - When memberships rarely change

DECENTRALIZED ALGORITHM

- Lin et al. [2004], decentralized voting algorithm
- Resource is replicated N times
- Each replica has its own coordinator …(N coordinators)
- Accessing resource requires majority vote:
  total votes (m) > N/2 coordinators
- **Assumption #1:** When coordinator does not give permission to access a resource (because it is busy) it will inform the requester

DECENTRALIZED ALGORITHM - 2

- **Assumption #2:** When a coordinator crashes, it recovers quickly, but will have forgotten votes before the crash.
- Approach assumes coordinators reset arbitrarily at any time
- **Risk**: on crash, coordinator forgets it previously granted permission to the shared resource, and on recovery it errantly grants permission again
- **The Hope**: if coordinator crashes, *upon recovery*, ***the node granted access to the resource has already finished before the restored coordinator grants access again . . .***

March 3, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L16.49

49

DECENTRALIZED ALGORITHM - 3

- With 99.167% coordinator availability (30 sec downtime/hour) chance of violating correctness is so low it can be neglected in comparison to other types of failure
- Leverages fact that a new node must obtain a majority vote to access resource, ***which requires time***

N	m	p	Violation
8	5	3 sec/hour	$< 10^{-15}$
8	6	3 sec/hour	$< 10^{-18}$
16	9	3 sec/hour	$< 10^{-27}$
16	12	3 sec/hour	$< 10^{-36}$
32	17	3 sec/hour	$< 10^{-52}$
32	24	3 sec/hour	$< 10^{-73}$

N	m	p	Violation
8	5	30 sec/hour	$< 10^{-10}$
8	6	30 sec/hour	$< 10^{-11}$
16	9	30 sec/hour	$< 10^{-18}$
16	12	30 sec/hour	$< 10^{-24}$
32	17	30 sec/hour	$< 10^{-35}$
32	24	30 sec/hour	$< 10^{-49}$

N = number of resource replicas, m = required “majority” vote
p=seconds per hour coordinator is offline

March 3, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L16.50

50

DECENTRALIZED ALGORITHM - 4

- **Back-off Polling Approach for *permission-denied*:**
- If permission to access a resource is denied via majority vote, process can poll to gain access again with a ***random*** delay (***known as back-off***)
- Node waits for a random amount, retries…
- If too many nodes compete to gain access to a resource, majority vote can lead to low resource utilization
  - ***No one can achieve majority vote to obtain access to the shared resource***
  - *Mimics elections where with too many candidates, where no one candidate can get >50% of the total vote*
- Problem Solution detailed in [Lin et al. 2014]

DISTRIBUTED MUTUAL EXCLUSION ALGORITHMS REVIEW

- Which algorithm offers the best scalability to support distributed mutual exclusion in a large distributed system?
- (A) Token-ring algorithm
- (B) Centralized algorithm
- (C) Distributed algorithm
- (D) Decentralized voting algorithm

DISTRIBUTED MUTUAL EXCLUSION ALGORITHMS REVIEW - 2

- Which algorithm(s) involve blocking when a resource is not available?
- (A) Token-ring algorithm
- (B) Centralized algorithm
- (C) Distributed algorithm
- (D) Decentralized voting algorithm

March 3, 2020	TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma	L16.53

53

DISTRIBUTED MUTUAL EXCLUSION ALGORITHMS REVIEW - 3

- Which algorithm(s) involve arriving at a consensus to determine whether a node should be granted access to a resource?
- (A) Token-ring algorithm
- (B) Centralized algorithm
- (C) Distributed algorithm
- (D) Decentralized voting algorithm

March 3, 2020	TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma	L16.54

54

DISTRIBUTED MUTUAL EXCLUSION ALGORITHMS REVIEW - 4

- Which algorithm(s) have N points of failure, where N = Number of Nodes in the system?
- (A) Token-ring algorithm
- (B) Centralized algorithm
- (C) Distributed algorithm
- (D) Decentralized voting algorithm

55

QUESTIONS

56