Distributed Systems

04r. Pre-Exam 1 Review

David Domingo

Rutgers University

Fall 2018

Assignment 2 Review

Question 1

You have the following timestamps:

Operation	timestamp
Client sends request:	8:22:10.300
Client receives response:	8:22:10.350
Server receives request:	8:10:00.600
Server sends response:	8:10:00.610

Note that the client's time is ahead of the server's. We expect a negative offset

In the case of a client synchronizing with the server, A refers to the client and B refers to the server in the NTP RFC. Using NTP, what is the new time (add the offset, theta, to the client receives response time)?

Question 1

Operation	timestamp
Client sends request:	8:22:10.300
Client receives response:	8:22:10.350
Server receives request:	8:10:00.600
Server sends response:	8:10:00.610

T_1 = 8:22:10.300
T_2 = 8:10:00.600
T_3 = 8:10:00.610
T_4 = 8:22:10.350

Formula from RFC5905, page 28:

$$\text{theta} = T(B) - T(A) = 1/2 * [(T_2 - T_1) + (T_3 - T_4)]$$

theta = 1/2 * [(8:10:00.600 - 8:22:10.300) + (8:10:00.610 - 8:22:10.350)]

theta = 1/2 * [(-0:12:09.700) + (-0:12:09.740)]

theta = 1/2 * [- 0:24:19.440] = - 0:12:09.720

theta is the offset: add the offset to our current time (T_4)

time = T_4 + theta = 8:22:10.350 - 0:12:09.720 = **8:10:00.630**

Question 2

How does Lamport define concurrent events?
(Just the high-level definition, not using timestamps.)

Page 559 – right column:

> Another way of viewing the definition is to say that $a \rightarrow b$ means that it is possible for event a to causally affect event b. **Two events are concurrent if neither can causally affect the other.** For example, events p_3 and q_3

Two events, *a* & *b*, are causal and $a \rightarrow b$ (*a* happened before *b*) **if**

a took place before *b* on the same system

a is the event of sending a message & *b* is the event of receiving it

or there is a transitive relationship such that

$$a \rightarrow q_0,\ q_0 \rightarrow q_1,\ \ldots,\ q_{n-1} \rightarrow q_n,\ q_n \rightarrow b$$

Question 3

From the *Why Vector Clocks are Easy* paper, how can you tell if one vector clock is a descendent of another vector clock?

In order for vector clock ***B*** to be considered a descendant of vector clock ***A***, each marker in clock ***A*** must have a corresponding marker in clock ***B*** that has a revision number greater than or equal to the marker in ***A***.

In the paper, saying *B is a descendant of A* is the same as saying that *B is causally dependent on A*: there is a causal relationship.

A:	Alice:1,	Ben:1,	Cathy:1,	Dave:2	
	≥	≥	≥	≥	*B* is a descendant of *A*
B:	Alice:3,	Ben:4,	Cathy:1,	Dave:2	

Question 3

A: Alice:1, Ben:1, Cathy:1, Dave:2, Emily:3

≥ ≥ ≥ ≥

B: Alice:3, Ben:4, Cathy:1, Dave:2

B is NOT a descendant of *A*
Emily is missing from *B*
(Emily is implicitly 0 in *B*)

A: Alice:1, Ben:1, Cathy:1, Dave:2

≥ ≥ ≥ $<$

B: Alice:3, Ben:4, Cathy:1, Dave:1

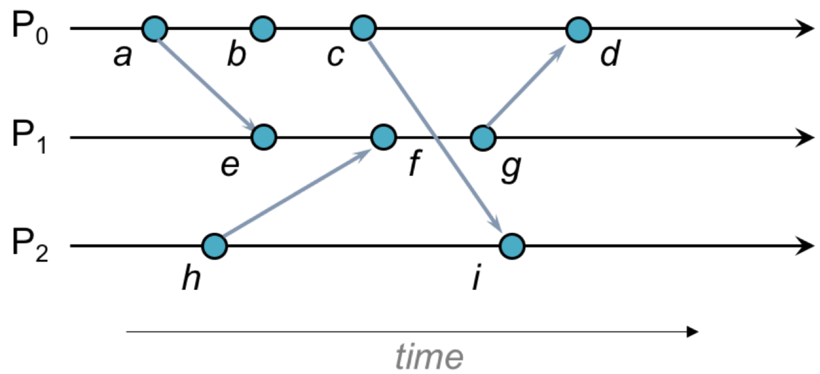
B is NOT a descendant of *A*
Dave:2 is NOT ≥ Dave:1

Note: A vector of clock values (e.g., {2, 4, 8, 3}) instead of {Process ID, clock} tuples makes sense only if the group membership is known ahead of time and everyone's place in the group is uniquely identified.

In real life, it is likely that new processes may join.

Question 4

The diagram below shows nine events (*a, b, … i*) on three processes.

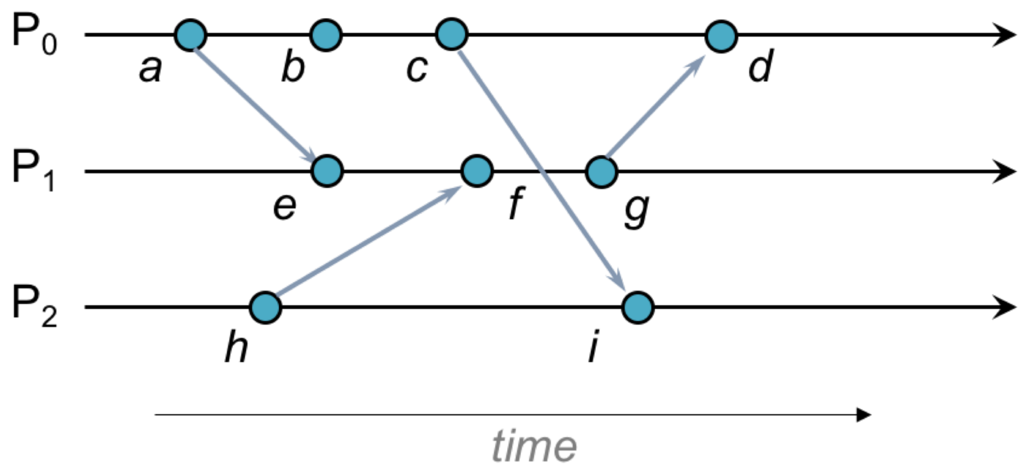


Assign Lamport timestamps to each event.

The event clock on each process is initialized to 0 at the beginning and incremented prior to timestamping each event.

For instance, the clock on P_0 starts at 0 and event a gets assigned a Lamport timestamp of 1 for event *a*.

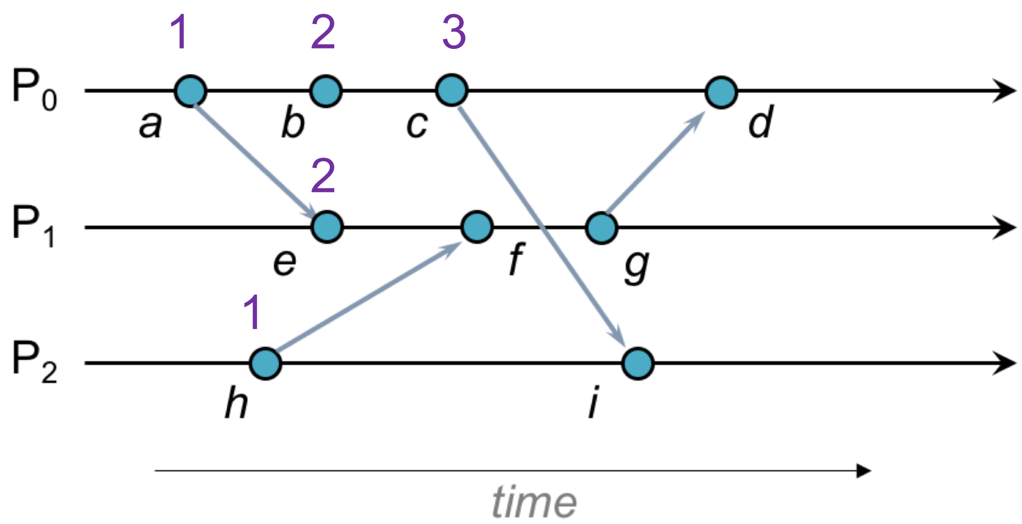
Question 4



Lamport timestamping rules:

1. Each system maintains a counter, initialized to 0.
2. Before you associate a timestamp with an event, you increment the counter.
3. If the event is that of receiving a message then you still pre-increment the counter but then compare the received timestamp with the one you were planning to associate with the event.
   If the received timestamp is ≤ to the local one, then set the event timestamp (and your counter) to the received timestamp + 1.

Question 4

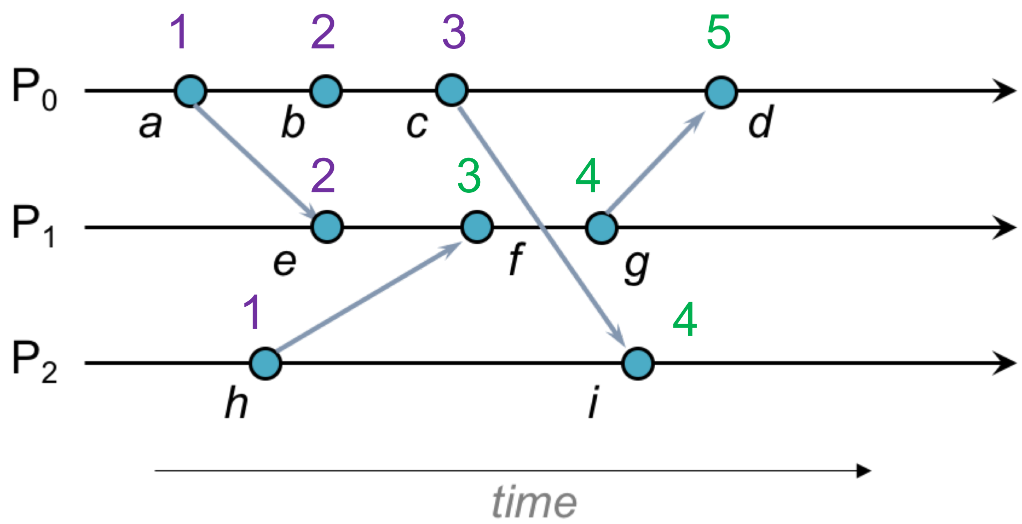


Event a is 1 : P_0's counter was initialized to 0. 0+1 = 1
Event b is 2 : P_0's counter was previously 1. 1+1 = 2
Event c is 3 : P_0's counter was previously 2. 2+1 = 3

Event e is the receipt of a message.
If it was a regular event, it would get a value of 1
– but the received message is 1, so we set it to *received_timestamp +1 = 2*
P_1's counter is now 2.

Event h is 1 : P_2's counter was initialized to 0. 0+1 = 1

Question 4



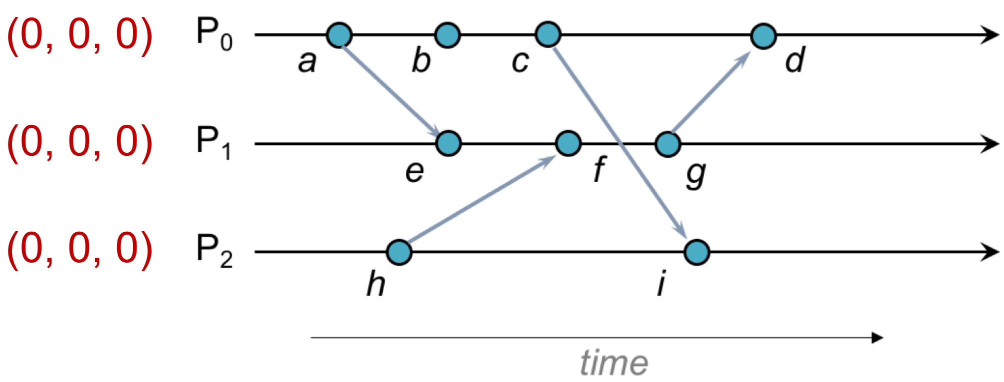
Event f is the receipt of a message from h, so it contains a value of 1.
If it was a regular event, it would get a value of 2+1 = 3.
Since the clock value in the received message is < 3, event f stays with 3.

Event i is the receipt of a message from event c (3).
i would normally get 1+1=2 but 3 ≥ 2, so we set i's value to 3+1 = 4.

Event g is 4 – it is the the next event after f, so 3+1 = 4

Event f is the receipt of a message from g (4). 4 > 3, so set d's value to 4+1 = 5

Question 5



Using the same set of events as in the previous question, assign **vector timestamps** to each event.

The event clock vector at each process is initialized to all zeros at the beginning and a process increments its position in the vector prior to timestamping each event. Process positions in the vector are (P0, P1, P2).

Question 5

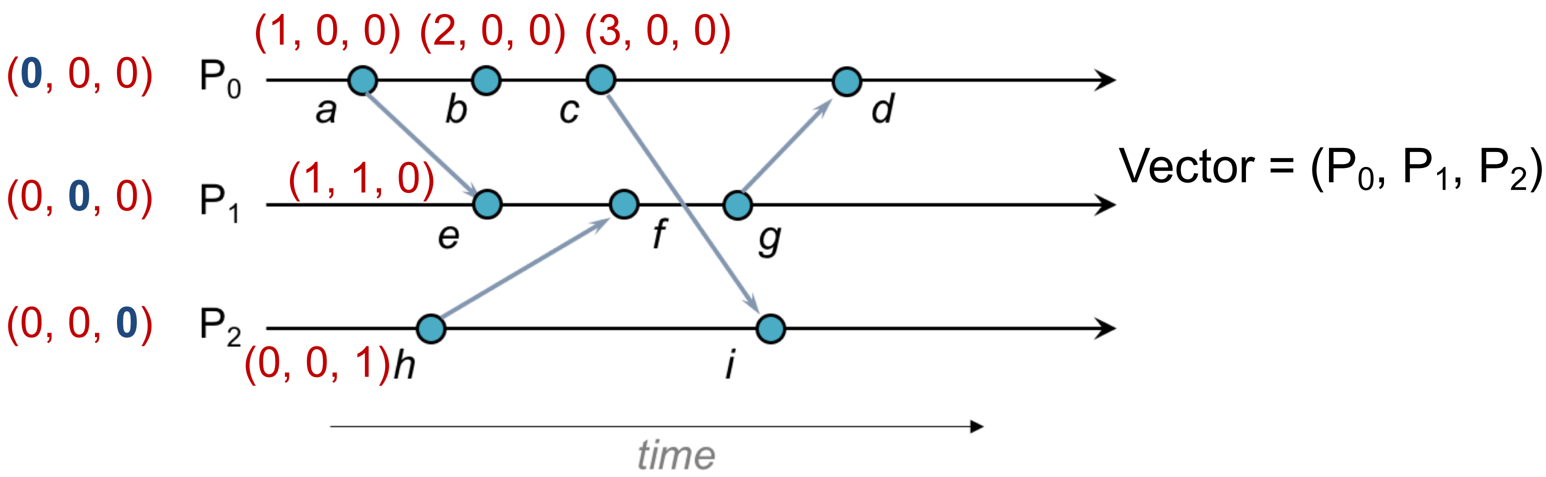
- Rules for generating vector timestamps
- Each process keeps a vector of the latest timestamps of all processes it has received
  - Example { 0, 2, 3, 1 } to represent timestamps for { P_0, P_1, P_2, P_3 }
  - In practice, if we don't know all group members, we will store & send:

    { (P_0, 0), (P_1, 2), (P_2, 3), (P3, $_1$) }

  For purposes of analysis, it's the same – just a matter of implementation

- Before timestamping an event, increment **only** your process' counter in the vector
  - Example: P_2 would change its vector from { 0, 2, **3**, 1 } to { 0, 2, **4**, 1 } .

Question 5



a = (1, 0, 0) – P_0 only increments P_0's element before timestamping an event
b = (2, 0, 0) – — *same* —
c = (3, 0, 0) – — *same* —
h = (0, 0, 1) – P_2 only increments P_2's element before timestamping an event

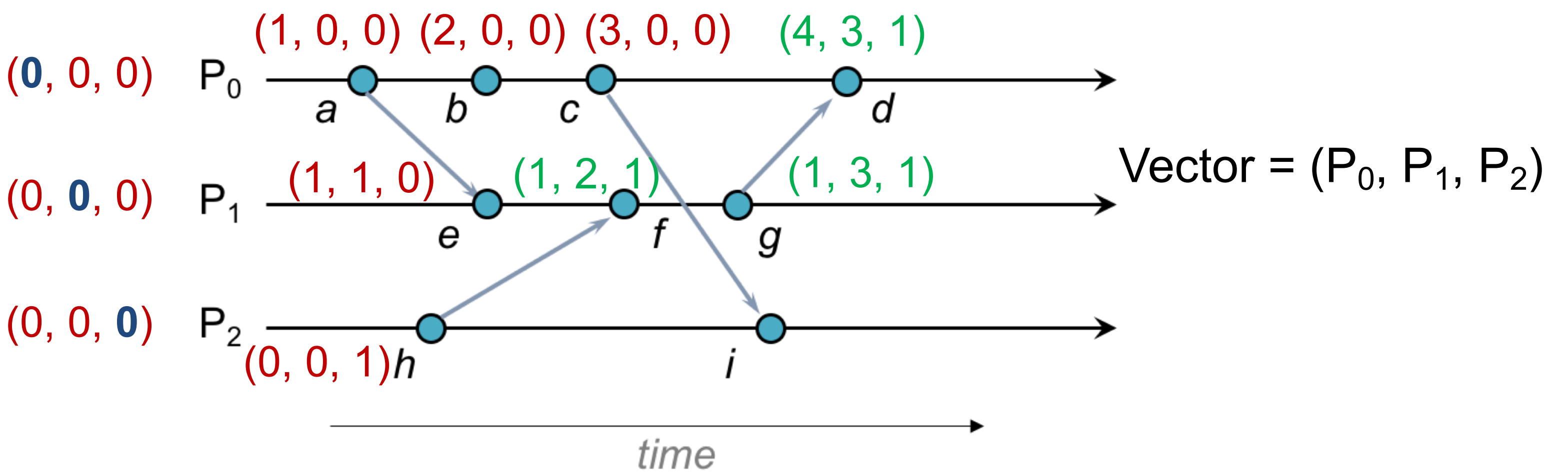
e would get a vector timestamp = (0, 1, 0) if it was a local event
Since it's the receipt of (1, 0, 0):

1. Compare the vectors element by element
2. Pick the greater value of each pair of elements for the new vector

(1, 0, 0) : (0, 1, 0) ⇒ (1, 1, 0)

e = (1, 1, 0)

Question 5



f – P_1 only increments P_1's element before timestamping an event
e was (1, 1, 0), so f would be (1, 1+1, 0) = (1, 2, 0)
but it's the receipt of (0, 0, 1), so compare the elements of (1, 2, 0) and (0, 0, 1):

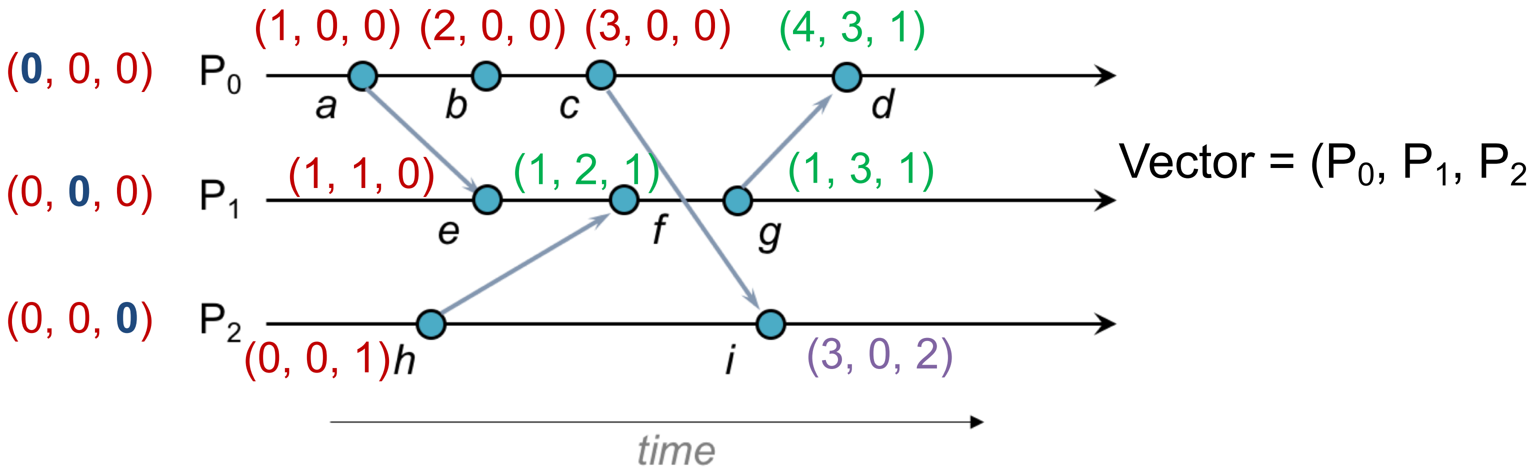
(1, 2, 0) : (0, 0, 1) ⇒ **(1, 2, 1)**

g : (1, 2+1, 0) = **(1, 3, 1)**

d would normally be (3+1, 0, 0) = (4, 0, 0)
but it's the receipt of (1, 3, 1), so compare the elements of (1, 3, 1) and (4, 0, 0):

(1, 3, 1) : (4, 0, 0) ⇒ **(4, 3, 1)**

Question 5



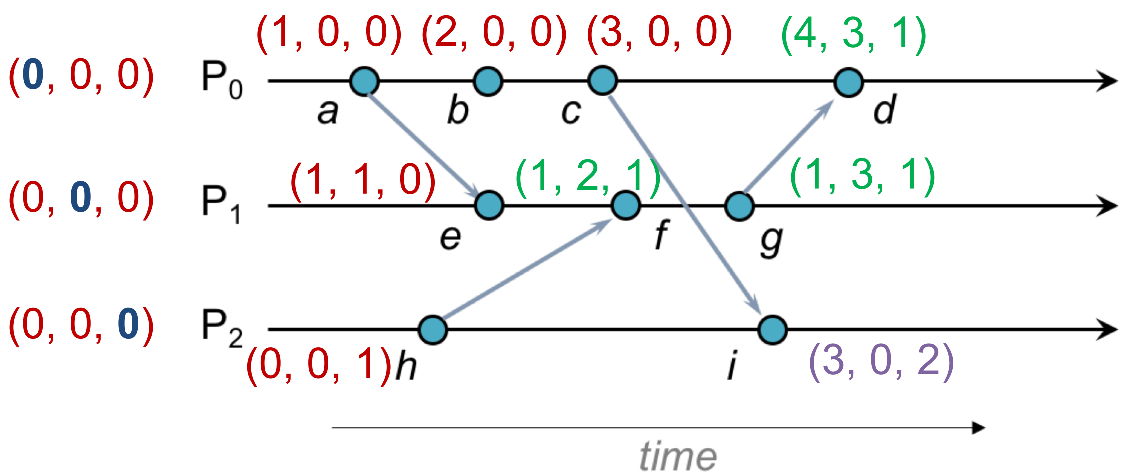
i would normally be (0, 0, 1+1) = (0, 0, 2)

but it's the receipt of (3, 0, 0), so compare the elements of (0, 0, 2) and (3, 0, 0):

(0, 0, 2) : (3, 0, 0) ⇒ **(3, 0, 2)**

Question 6

Which events are concurrent with event b?



Do an element-by-element comparison of b=(2,0,0) with other vectors

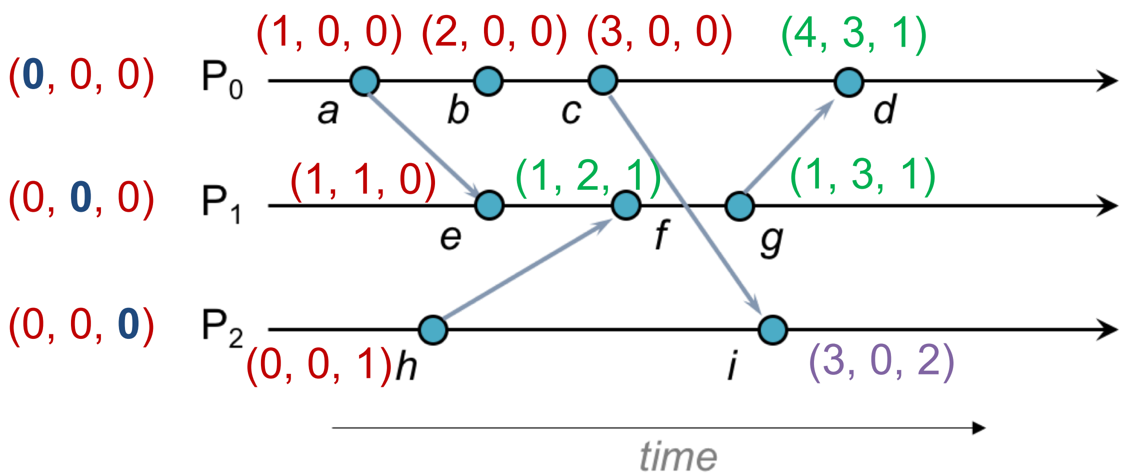
If each corresponding element of b is ≥ the other vector then
b happened before the other vector.

If each corresponding element of b is ≤ the other vector then
the other vector happened before b.

Find vectors where neither of these apply

Question 6

Which events are concurrent with event b?



Example: (4, 3, 1) ≥ (2, 0, 0). Therefore, **b → d**, so d is NOT concurrent with b.

Concurrent with b:

e: (1, 1, 0) ≰ (2, 0, 0) and (1, 1, 0) ≱ (2, 0, 0)
f: (1, 2, 1) ≰ (2, 0, 0) and (1, 2, 1) ≱ (2, 0, 0)
g: (1, 3, 1) ≰ (2, 0, 0) and (1, 3, 1) ≱ (2, 0, 0)
h: (0, 0, 1) ≰ (2, 0, 0) and (0, 0, 1) ≱ (2, 0, 0)

Selected questions from past exams

Fall 2017 Question 1

What problem can arise with a system that exhibits fail-restart behavior?

Stale state: the system has an outdated view of the world when it starts up.

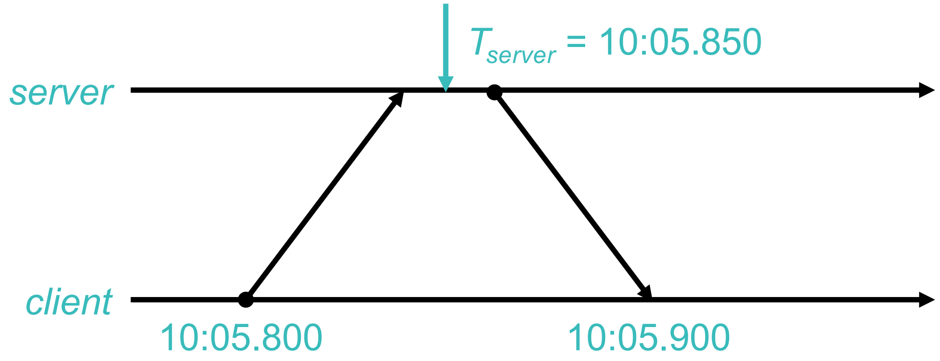
Not: data gets lost or missed messages
– that is true for fail-stop behavior as well

Fall 2017 Question 2

At 10:05.800, a client sends the server a request for the time. The server response arrives at 10:05.900 containing a time stamp of 10:05.850. Using **Cristian's algorithm**, to what value does the client set its clock?

$$\begin{aligned} \text{offset} &= (T_{\text{received}} - T_{\text{sent}}) \div 2 \\ &= (10{:}05.900 - 10{:}05.800) \div 2 \\ &= 0.1 \div 2 = 0.05 \end{aligned}$$

New time = T_{server} + offset = 10:05.850 + 0.05 = **10:05.9**



Fall 2016 Question 1

Why does it not make sense to use TCP (Transmission Control Protocol) for the Network Time Protocol (NTP)?

TCP offers reliable delivery **but** via *retransmission*.

TCP also may delay the transmission of data.

These factors may lead to jitter – variations in the delay, which will make the assumption that the timestamp is generated in the middle invalid

Bad answers:

- *TCP has longer latency*
- *TCP has high overhead*

Fall 2016 Question 2

What is a benefit of lease-based garbage collection over reference count based garbage collection?

It's not fault tolerant.

If a client process dies or exits without properly decrementing reference counts, the object would not get deleted.

Fall 2016 Question 3

(a) Explain the role of an interface definition language in remote procedure calls

Describes the programming interface for remote (functions, data types, parameters, return values) so that stub functions can be generated.

Bad answer: creates stubs

(b) Explain the purpose of marshaling in remote procedure calls.

Convert a list of parameters into a sequence of bytes (a serialized format).

Fall 2015 Question 1

Why did the use of reference counting for remote objects prove to be impractical? Explain.

It's not fault tolerant.

If a client process dies or exits without properly decrementing reference counts, the object would not get deleted.

Bad answers:

- *Requires more network usage (or extra unnecessary requests issued by client)*
  - *That may be true only in some cases (e.g., a lot of object referencing activity on the client) but it does not make the solution impractical*
- *Problems with lost messages*
  - *That could be a problem but is a problem with any protocol, including leasing. You need to use reliable messaging (e.g., acknowledgements & retransmissions).*

Fall 2015 Question 2

(a) What is the advantage of vector clocks over Lamport clocks?

Vector clocks allow you to tell whether a set of events are causally related or concurrent by comparing their timestamps.

(b) What is a disadvantage?

1. Vector timestamps use more space because you have a vector (one element for each process) rather than one integer.
2. Comparing them takes more time since you need to do an element-by-element comparison.

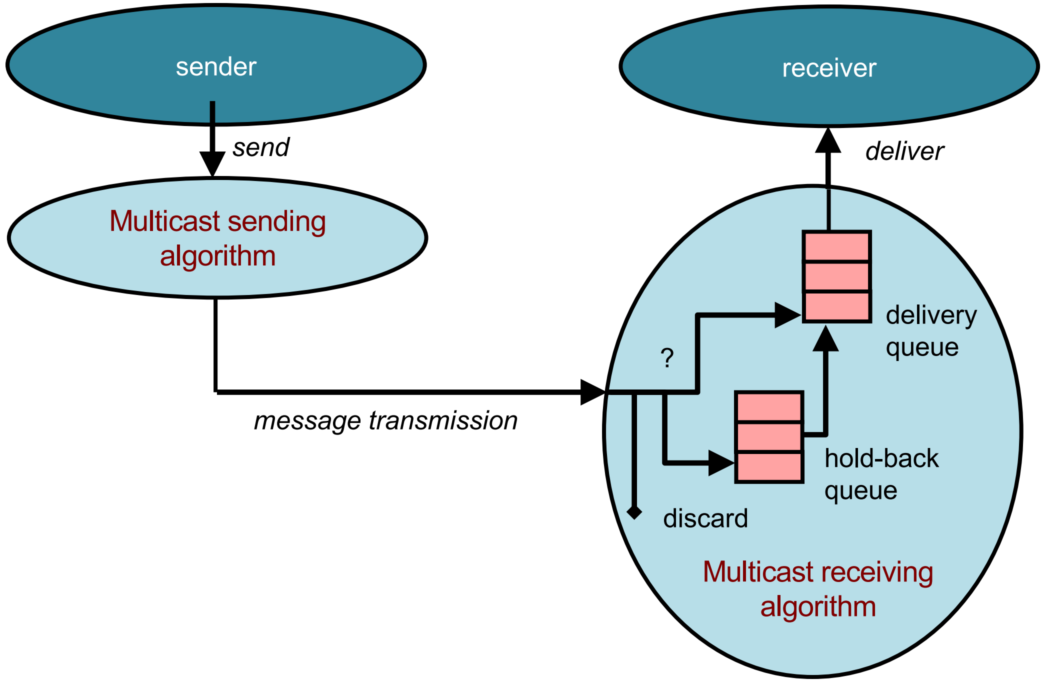
Bad answer: "more expensive", "slower"
Answers such as these are too vague to show that you understand the material.

Fall 2017: Question 4

Explain the distinction between receiving and delivering a message.

Receiving = message arrives the computer

Delivering = message is presented to the application



Fall 2017 Question 7

A network *partition* refers to:

(a) A protected segment of the network for administrative tasks.

(b) Each local area network within the Internet.

(c) A type of fault where the network fragments into two or more disconnected sub-networks.

(d) A file system that is shared among multiple systems on a network.

Fall 2017 Question 10

Piggybacked acknowledgements:
(a) Prevent feedback implosion.
(b) Incorporate an acknowledgement within a response message.
(c) Optimize network use by sending one acknowledgement for multiple messages.
(d) Are a way for the sender to acknowledge receipt of an acknowledgement.

(a) Feedback implosion
- Send a multicast message out and get replies from all group members
- Piggybacked ACKs can help a bit with feedback implosion but this doesn't answer the question.

(c) Sending one ACK for multiple messages
- This is a cumulative acknowledgement

(d) Protocols generally do not acknowledge receipt of ACKs

Fall 2016 Question 10

IP is designed to be implemented over:

a) Unreliable connectionless networks.
b) Reliable connectionless networks.
c) Unreliable connection-oriented networks.
d) Reliable connection-oriented networks.

Fall 2016 Question 11

Port numbers are used in:

a) IP.
b) UDP only.
c) UDP & TCP.
d) TCP only.

Port numbers are a transport-layer construct to identify socket endpoints.

The network layer (IP) is only responsible for getting packets to the computer, so it has no need for port numbers.

Fall 2016 Question 12

TCP cannot provide:

a) Reliable delivery.
b) In-order delivery.
c) Constant latency.
d) Congestion control.

- Reliable delivery = retransmit lost or damaged data
- In-order delivery = each segment contains a sequence number
- Congestion control = reduce transmission rate (window size) if packet loss is detected

TCP cannot control how long it takes to deliver a packet.

Fall 2016 Question 15

A key advantage of multi-canonical marshaling is that it:

a) Enables a set of data to be sent to multiple servers simultaneously.
b) Allows clients and servers to have different processor architectures.
c) Reduces the overall amount of data conversion that needs to be performed.
d) Allows clients to communicate directly with servers without routing messages through a proxy.

- Ideally, neither client nor server will have to convert data to a local format.

Fall 2017 Question 12

An RPC server *skeleton* (stub):

(a) Receives requests from clients and calls the local function on the server.

(b) Is an automatically-generated template for writing server functions.

(c) Is used to discover remote procedures that reside on the server.

(d) Is called when the server-side function cannot be found.

(b) It's automatically generated server code but its purpose is not to be a template

(c) A name server is used for this (e.g., *portmap* on Linux)

(d) There's no "default" service that is called if the real service cannot be found

Fall 2016 Question 17

A surrogate process in Microsoft's COM+:

a) Runs on the client and loads client-side stub objects.
b) Runs on the client and receives requests if the server cannot be reached.
c) Runs on the server and starts RPC services at boot time.
d) Runs on the server and loads objects based on client requests.

Fall 2016 Question 18

In a group of two computers, a client's local clock reads 6:27:10. Using the Berkeley clock synchronization algorithm, to what value does the client set its time if the server's clock reads 6:28:30? Ignore message transit times.

a) 6:27:50
b) 6:28:30
c) 6:29:10
d) 6:29:50

In the Berkeley algorithm, there is no concept of a server that has the "true time"

- Server = master; client = slave
- Berkeley synchronization averages out all time values – we only have two:

(6:27:10 + 6:28:30) / 2 = 6: 27: (10 + 90)÷2 = 6:27:(100÷2) = 6:27:50

Fall 2016 Question 19

An NTP synchronization subnet is:

a) A high-speed network that is dedicated to clock synchronization.
b) The set of servers that offers clock synchronization services.
c) Reserved capacity dedicated to clock synchronization in an existing network.
d) Any network over which an NTP server continuously sends time broadcasts.

Fall 2017 Question 21

Events *x, y, z* have Lamport timestamps of 3, 3, 5, respectively. They may or may not have occurred on different processes. What can you definitively say definitively about these events?

(a) *x* and *y* are concurrent.
(b) Both *x* and *y* happened before *z*.
(c) Both (a) and (b).
(d) Neither (a) nor (b).

By looking at Lamport timestamps, we cannot tell the ordering:

- If L(a) < L(b), we don't know that a → b

However, if two events are causal (a → b) then L(a) < L(b)

- Two causal events will ***never*** have the same timestamps

Fall 2016 Question 21

Atomic multicast differs from reliable multicast because atomic multicast

a) Is much faster since it uses the network hardware to ensure reliability.
b) Only requires partial ordering.
c) Does not need to deliver messages reliably.
d) Accounts for system failures.

Fall 2017 Question 24

Protocol Independent Multicast is used to:

(a) Route IP multicast packets within the Internet.
(b) Support multiple forms of multicast beyond IP multicast.
(c) Provide sender-selectable levels of reliability in multicast streams.
(d) Provide sender-selectable levels of reliability and message ordering in multicast streams.

(b): PIM just handles IP multicast

(c), (d): IP multicast does not offer varying levels of reliability

The end