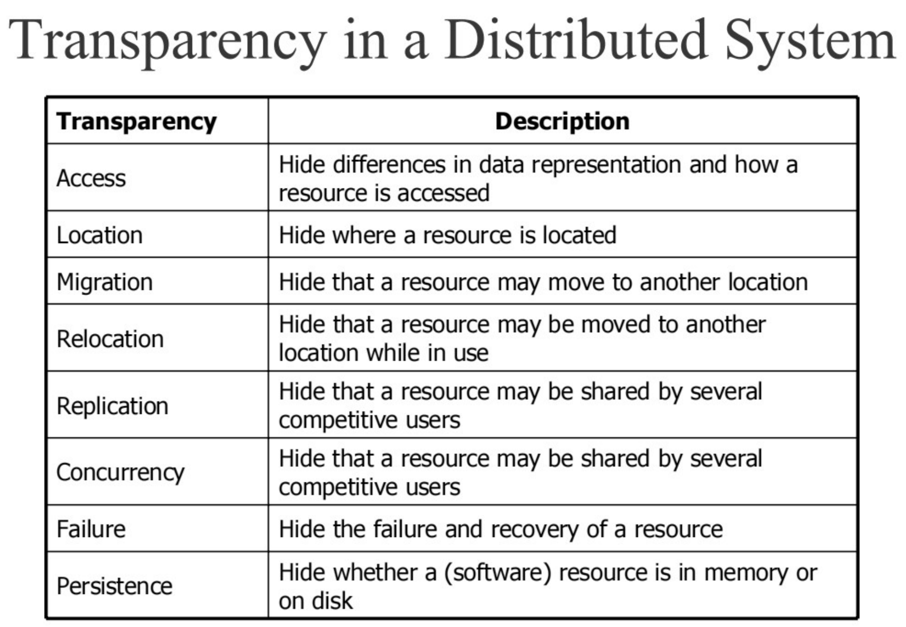
Question 1

1) a) A distributed operating system is a system which manages a collection of independent computers and makes them appear to the user of the system as a single computer.



Transparency in a Distributed System

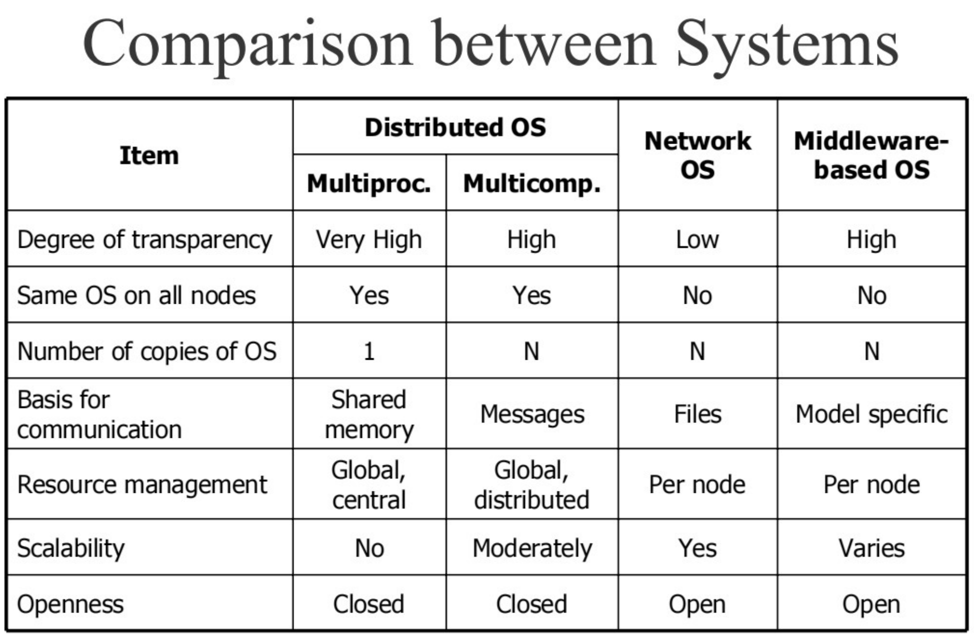
Transparency	Description
Access	Hide differences in data representation and how a resource is accessed
Location	Hide where a resource is located
Migration	Hide that a resource may move to another location
Relocation	Hide that a resource may be moved to another location while in use
Replication	Hide that a resource may be shared by several competitive users
Concurrency	Hide that a resource may be shared by several competitive users
Failure	Hide the failure and recovery of a resource
Persistence	Hide whether a (software) resource is in memory or on disk

2)

.

2)a) software that acts as a bridge between an operating system or database and applications, especially on a network.

b)

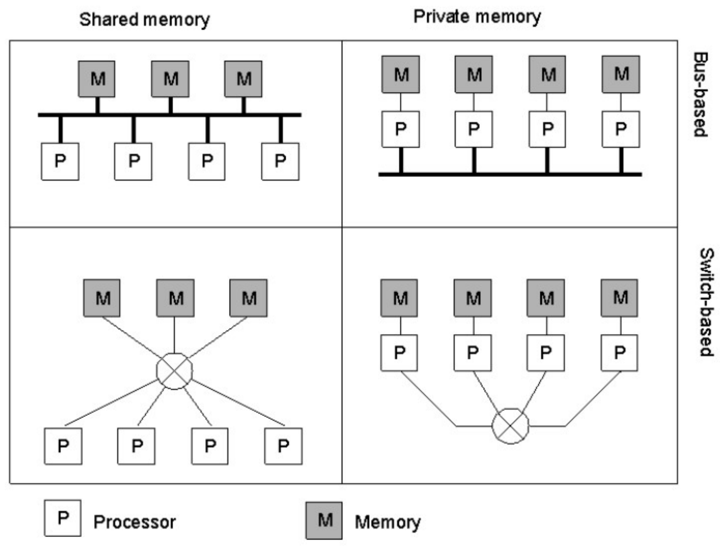


Comparison between Systems

Item	Distributed OS		Network OS	Middleware-based OS
	Multiproc.	Multicomp.		
Degree of transparency	Very High	High	Low	High
Same OS on all nodes	Yes	Yes	No	No
Number of copies of OS	1	N	N	N
Basis for communication	Shared memory	Messages	Files	Model specific
Resource management	Global, central	Global, distributed	Per node	Per node
Scalability	No	Moderately	Yes	Varies
Openness	Closed	Closed	Open	Open

c)

Hardware Concepts



Question 2

1) Message-Oriented Communication

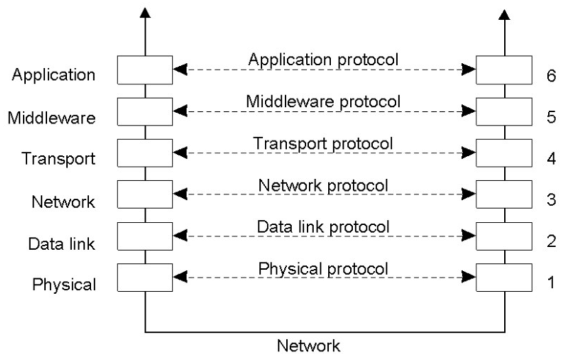
• RPC/RMI are also known as request/response.

• If we assume that there are no responses, we move into message-oriented communication.

• MOC is typically more decoupled than RPC/RMI.

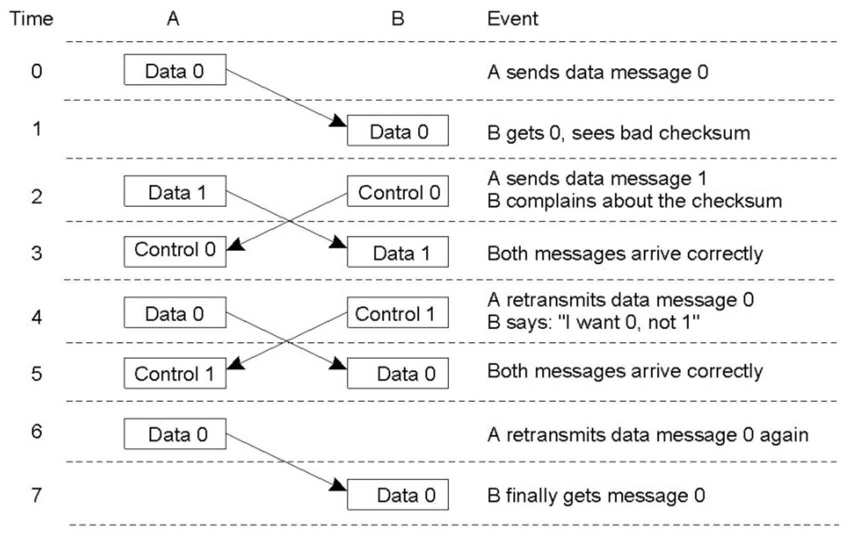
2)

Middleware Protocols



3)

Data Link Layer



Time	A	B	Event
0	Data 0		A sends data message 0
1		Data 0	B gets 0, sees bad checksum
2	Data 1	Control 0	A sends data message 1 B complains about the checksum
3	Control 0	Data 1	Both messages arrive correctly
4	Data 0	Control 1	A retransmits data message 0 B says: "I want 0, not 1"
5	Control 1	Data 0	Both messages arrive correctly
6	Data 0		A retransmits data message 0 again
7		Data 0	B finally gets message 0

4)

Clock synchronization is necessary for the ordering of events and to preserve the state of resources. As per algorithms, we can say that for clock synchronization there is need to consider propagation time of messages among each node in both types of algorithms centralized and distributed.

5)

A logical clock is quite different from a physical clock in that there is no central notion of time, and the clock is just a counter that increments based on events in the system

6)

When an event occurs

Question 3

1)

Berkeley Algorithm – Internal Synch

Internal: synchronize without access to an

external resource

$|C_i(t) – C_j(t)| < D$

Periodically,

S: send C(S) to each client P

P: calculate $\alpha P = C(P) – C(S)$

send α P to S

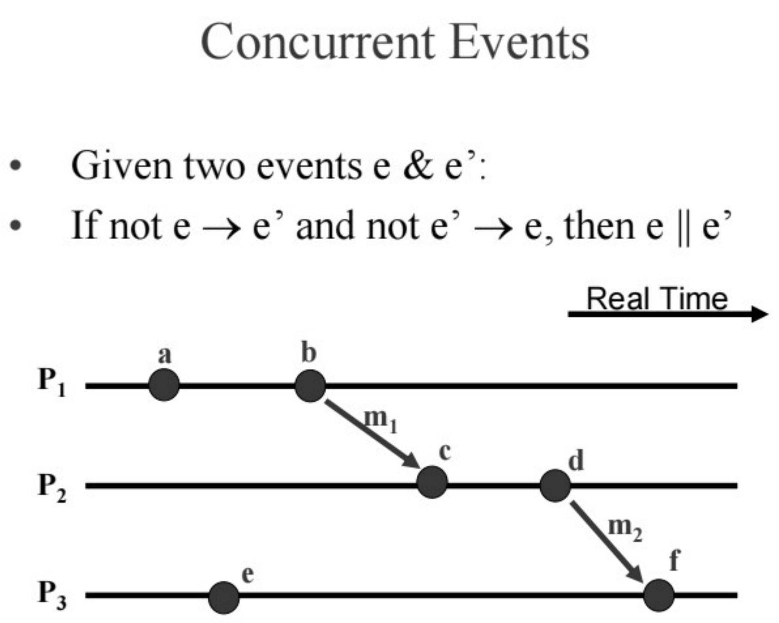
S: receive all α P's

compute an average Δ of α P, including α S

send Δ to all clients P

P: apply Δ to C(P)

2)b)



Concurrent Events

- Given two events e & e':
- If not e → e' and not e' → e, then e || e'

c) Thus two timestamps or counters may be the same within a distributed system, but in applying the logical clocks algorithm events that occur will always maintain at least a strict partial ordering.

d)1)

not a → b' and not a' → b, and

If LC(b) < LC(c), then it is

not necessarily true that b→ c

Hence a<c is not necessarily true.

2)

not a → b' and not a' → b,

If LC(b) < LC(c), then it is

not necessarily true that b→ c

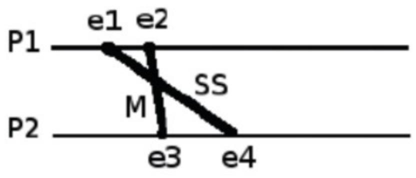
Hence it is true that a//c

4)

See 2(b)

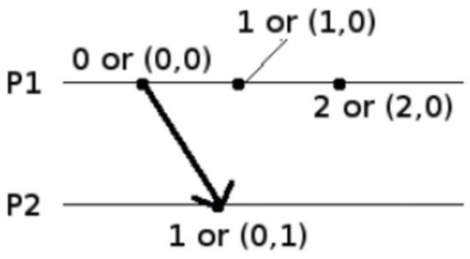
Question 4

1) The step of the proof that falls apart is: "If ej occurred before pj recorded its state, then ei must have occurred before pi recorded its state". Consider the Figure with P1 sending generic message M and snapshot marker message SS. If the FIFO channel assumption is violated, then it is possible that a message receive event (e3) is in the saved state of P2, even though the corresponding message send event (e2) is not in the saved state of P1. The resulting cut will contain the event e3 but exclude an event that happens-before it, thus is not consistent



2)

Consider the following example of 2 processes (Figure 6). The Lamport timestamp 2 in P1 is clearly larger than Lamport timestamp 1 in P2, but the corresponding events are not in happen-before relation. On the other hand with vector timestamps (2,0) and (0,1), corresponding to those two events, we can clearly conclude that these two events are not in happen-before relation (since VT1[P1] > VT2[P1] and VT1[P2] < VT2[P2]).



3)

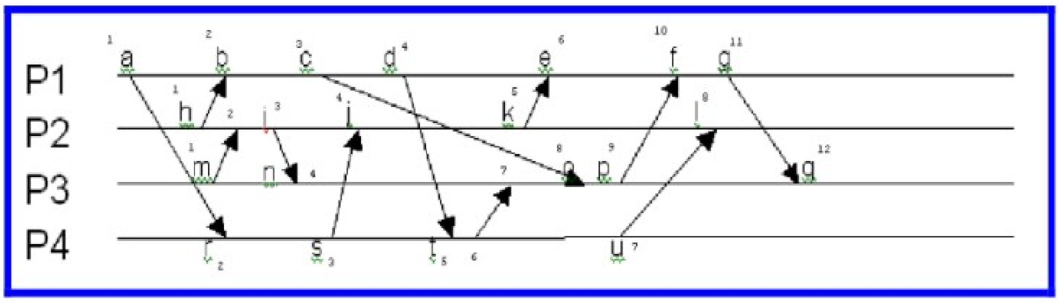
If the channel is symmetric, then t = t′, and o = oi+ (t′ – t)/2 = oi

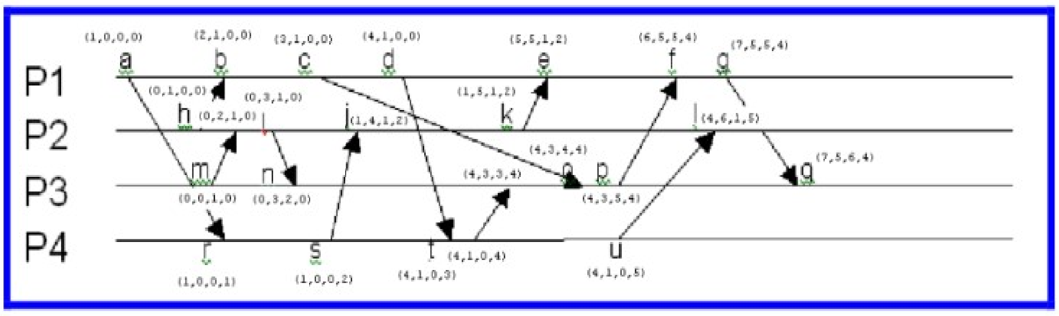
That is, the actual offset (clock skew)

is perfectly estimated (i.e., the error (t′-t)/2 = 0)

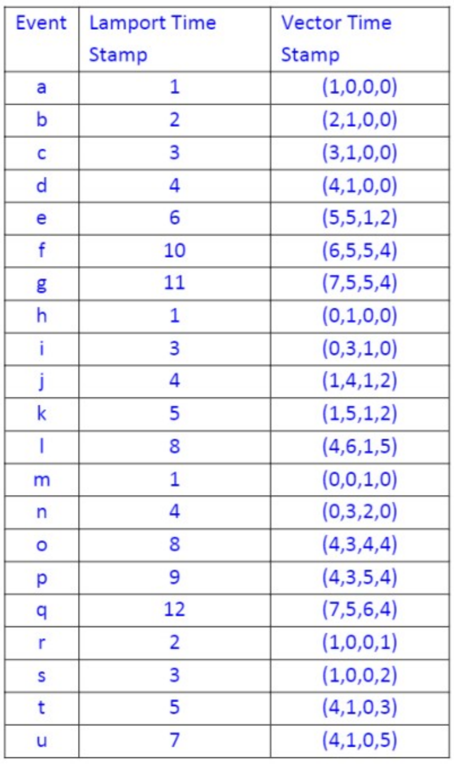
Question 5

1)





2)



Event	Lamport Time Stamp	Vector Time Stamp
a	1	(1,0,0,0)
b	2	(2,1,0,0)
c	3	(3,1,0,0)
d	4	(4,1,0,0)
e	6	(5,5,1,2)
f	10	(6,5,5,4)
g	11	(7,5,5,4)
h	1	(0,1,0,0)
i	3	(0,3,1,0)
j	4	(1,4,1,2)
k	5	(1,5,1,2)
l	8	(4,6,1,5)
m	1	(0,0,1,0)
n	4	(0,3,2,0)
o	8	(4,3,4,4)
p	9	(4,3,5,4)
q	12	(7,5,6,4)
r	2	(1,0,0,1)
s	3	(1,0,0,2)
t	5	(4,1,0,3)
u	7	(4,1,0,5)

3)

Yes, there is a potential causality violation. Specifically from point C to point O, the message passed holds (3,1,0,0) as the vector timestamp but the local vector timestamp on message arrival at point O is (4,3,3,4). Because the vector timestamp passed in the message is less than the local vector timestamp on arrival there is potential for a causal violation.

Question 6

a)

No, the run is not a linearization of events. One of the reason is that the event e24 (effect) is before e33 event (cause) which violated the happen-before relation. Linearization is a sequence of events that is consistent with happen-before relation. e33 e24 is in a happen-before relation, and so if it were linearization run, e24 would have been before e33. Another reason might be that the linearization does not have a complete set of all events in the global history of Figure 2 b)

No. For a cut to be consistent, if ei is in the cut and ej happens before ei, then eJ must be in the cut. In this example, e24 (receive event) is inthe cut, but the corresponding send event (e33) that happens before it is not in the cut.

c)

See Figure 4 for two additional consistent cuts. Cuts Y and Z are consistent cuts. The frontier for Cut Y is <e11, e22, e31>,The frontier for Cut Z is <e12, e23, e32>

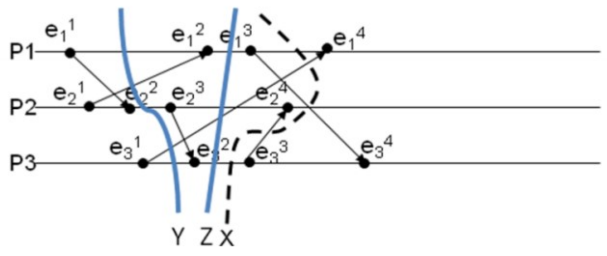


Figure 4: *Answer to Problem 4c – two additional consistent cuts Y and Z.*

d)

Events that happened before event e24: e11, e21, e22, e23, e31, e32, e33. Events that are concurrent with event e24: e12, e13, e14, e34.

Question 7

a)

RTT: Reply –Request = 610-540 = 70 1/100sec.

Adjusted local time: Server + RTT/2 = 375 + 35 = 410 1/100sec.

Drift: Adjusted local time – local time = 410 – 610 = -200 1/100sec.= -2 sec

b)

B's block is running 2 seconds too fast.

c)

To adjust B’s clock we cannot just set the time back 2 seconds or we lose our monotonicity condition (The condition that the clock always moves only forward), which could result in events appearing as if they occurred in the future and a variety of issues in software relating to this problem. For B to adjust its clock safely, it should decrease the rate at which updates are given to its clock (at least in software) thus ”slowing down” the time until it is caught up with the server’s time.

2)

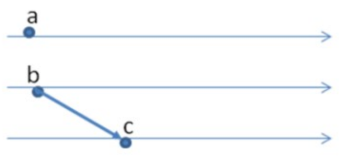
(a) Space-time physics orders two events in space-time according to the communication of the event by light leaving it. If event A's light reaches the event B's position before B's event time, then A precedes B. If B's light reaches A's position before A's event time, then B precedes A. If neither of these occur, which is not uncommon as light has a maximum speed, then A and B are unordered, or simultaneous. In distributed computing, the analog to light traveling from event A to event B is a message sent by A's thread after event A and received by B's thread before event B. If there is no communication between events, there is no basis for ordering them, and they are simultaneous.

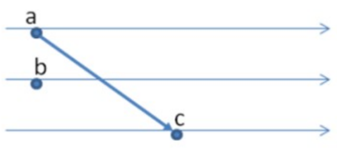
(b) On a multi-threaded single processors threads are (mostly) context switched at arbitrary times relative to their internal events/operations. An event A in one thread, that happens at cycle N1 but does not precede an explicit communication received before an event B in another thread, at cycle N2 > N1, cannot order itself, A, before B because in a different execution (scheduling) A's thread may be context switched at a different time, delayed and observe N1 > N2. Events that may be reordered merely by a different set of clock interrupts and scheduler decisions should be understood to be simultaneous.

(c) Logical clocks assign a time to each event in multiple threads so that any two events that should be understood to be ordered by communication will have logical clock values in the appropriate order, and all simultaneous events are given an order (not necessarily unique), consistent with the logical clocks of communicating events, so that all events have a possible total ordering. Programmers can the reason within this total ordering that any events that they wish to have ordered in some other ordering will require changes in the communications (and/or thread local code) to enforce. Programmers can also tag threads and events with an on-the-fly logical clock value on which to base order-sensitive decisions (eg. FIFO scheduling).

Question 8

1) a) No, consider counter example in Figure, a and b are concurrent, b -->c; however a is also concurrent with c.
b) No, consider counter example in Figure 5b: bis concurrent with both a and c, however, a-->c.





2) (a) From a set of trials, the fastest round trip (probe-respond) message delay can be used to estimate the skew in machine clocks, because the receiver time at reception of the probe should be the send time at the sender plus the message transit time. By using the fastest of many round trips to estimate the message transit time, we can estimate the difference in the sender and receiver clocks. For example, 1/2 the fastest round trip time, minus the turn around at the receiver, can be used to estimate the message transit time, and define the timestamp that the receiver should have had at message receive.

(b) Any communication channel not using the logical clock tagging algorthim could defeat logical clocks. A thread printing to a screen, causing a user to call a friend on the phone, who inputs to a different computer an event as a result of the call, is actually communication between threads, but won't have the appropriate tagging

UNLESS the logical clock tagging is real physical time (with speed of light communication speeds) so that the phone call could not communicate information faster than the physical clock advances.

(c) NTP is about close to synchronized clocks. It does not eliminate all covert channels because it cannot guarantee that the machines' clocks are so closely synchronized that no message might arrive fast enough for the clocks to be mis-ordered. In particular, there is no guarantee that the fastest sample of a set of message round trip trials is in fact the minimum transit time, so it may be possible for a particularly fast message to be received "before" it was sent, allowing a covert channel.

Question 9

a) Send (M1, 1)

Send (M2, 3)

Send (M3, 5)

b) A=10,

B=9,

C=5.

(Remember, receiving and sending are different events!)

c) This is a relatively ordered system

d)   If B received the message before sending the message to C, then C's clock would read 4. Otherwise, C's clock would read 2.

e) Because the communication over the cell phones was not timestamped using the lamport clock protocol. Therefore, the protocol did not know that the message from A to B had to have happened before the message from B to C.

f) It’s more likely to be 5. Most of the time, the communication latency between two computers at CMU is likely to be in the few milliseconds range, which is much faster than the two humans can communicate commands in words. Therefore, message M1 is likely to reach machine B well before the user instructs it to send a message to C.

Question 10

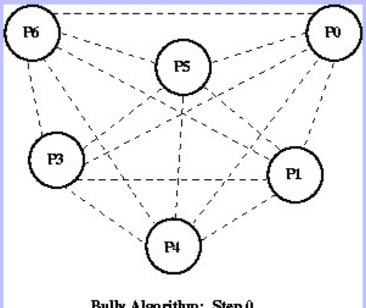
1)

We start with 6 processes,

all directly connected to each other.

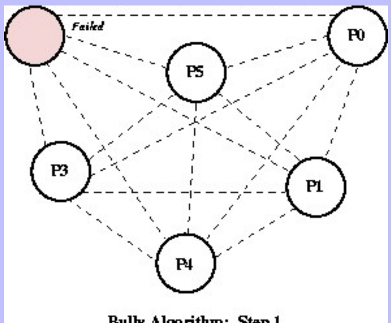
Process 6 is the leader,

as it has the highest number.



Bully Algorithm: Step 0

Process 6 fails.

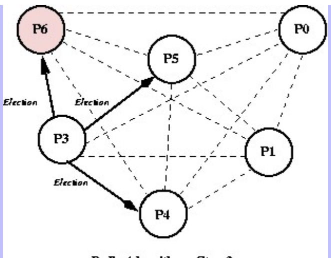


Bully Algorithm: Step 1

Process 3 notices that Process 6 does not respond

So it starts an election, notifying those processes

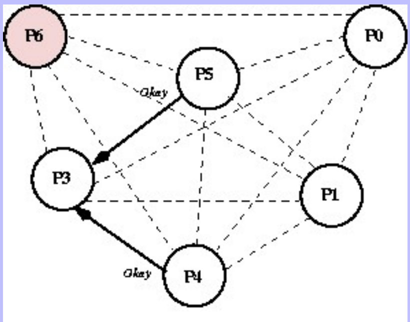
with ids greater than 3.



Bully Algorithm: Step 2

Both Process 4 and Process 5 respond,

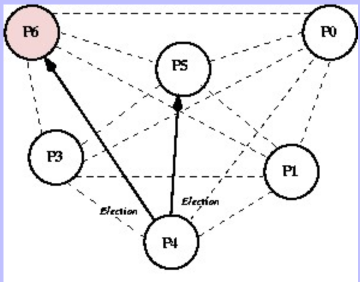
telling Process 3 that they'll take over from here.



Bully Algorithm: Step 3

Process 4 sends election messages

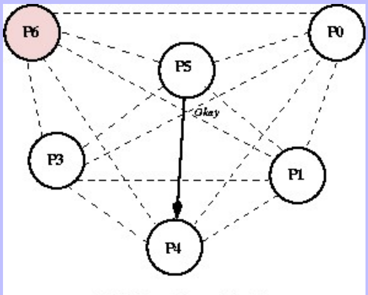
to both Process 5 and Process 6.



Bully Algorithm: Step 4

Only Process 5 answers

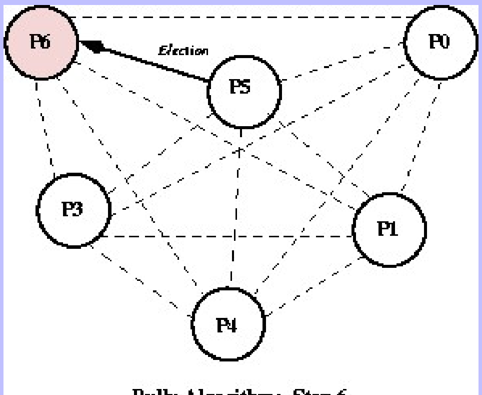
and takes over the election.



Bully Algorithm: Step 5

Process 5 sends out only one election message

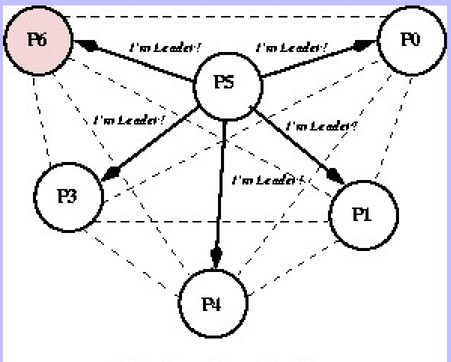
to Process 6.



Bully Algorithm: Step 6

When Process 6 does not respond

Process 5 declares itself the winner.



Bully Algorithm: Step 7

2)

A Token Ring Election Algorithm

by Chang & Roberts

Each process has a unique id.

Each process knows its successor in the ring.

When a process notices the leader is down,

it sends an election message to its successor.

If the successor is down,
    the originating process sends the message
        to the next process in the logical ring.

Each process that receives an election message,
    passes it on to the next process in the ring.

Each sender appends its own id to the message.

The election message eventually returns
    to the originating process
        and contains its id.

At this point, the election message is changed
    to a coordinator *(new leader)* message
        and sent around ring.

The process with the highest id
    in the circulated election message
        becomes the new leader.

When the coordinator message comes back
    to the originating process, it is deleted.