Distributed Systems

"autonomous" — should be able to perform some tasks independently

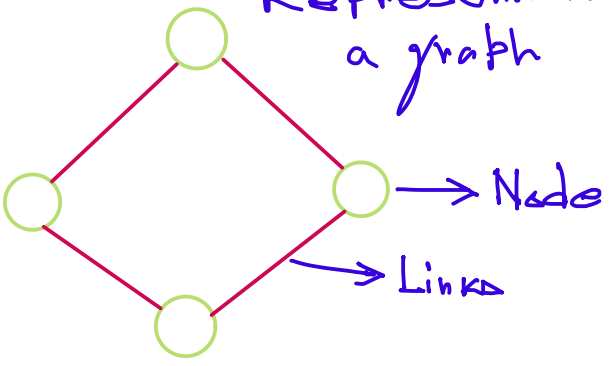
"communicate" — so, your PC Lab is not a dist system

"some task" — achieve something "global"

SIMD

Flynn's Classification

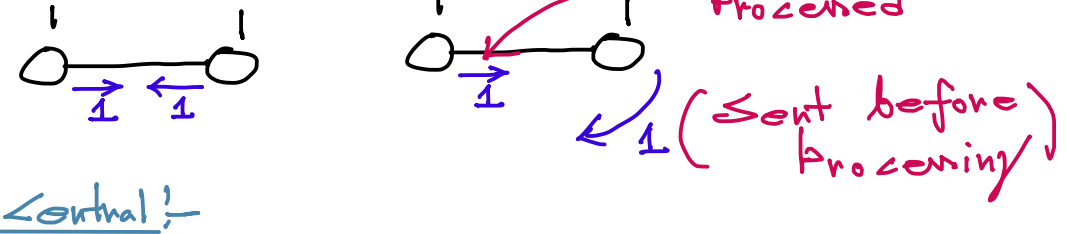
Parallel
Concurrent
Distributed



Fault-tolerant

Replicated

Atomicity Issues!-



Central!-

1) How to elect leader?
2) How everyone knows who is the leader?
   (Each non-leader node)
3) Leader failure?
4) Structure of the network (How to reach the leader?)

1. Dist Algorithms
2. Fault
3. Replication
4. "Other"

ISIS

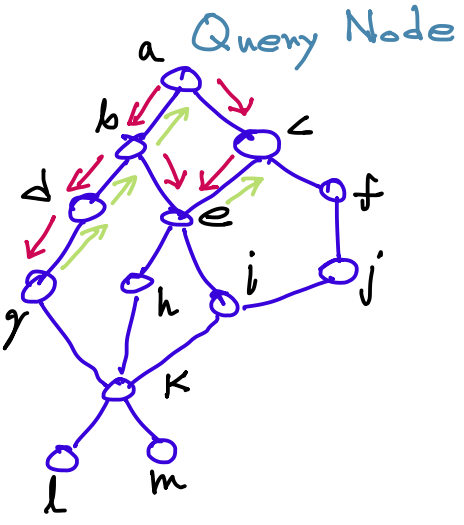
Every algo has some assumptions of the system

Asynchronous:-
Makes no assumption on timing anywhere in the system.
— No assumption on max msg delay
— No assumption on processing delay

Synchronous:- Makes some assumption on timing
Typically
— Assumes max msg delay
— Assumes processing delay

$2 \times d \times \alpha + \beta$
↑
diameter

→ Choose as parent the first node you get the query msg from



1) On receiving a msg
   a) First! Send <CHILD to parent and forward to else
   b) Else! Send REJ

Q) How do we write a distributed algorithm?

Main Idea:- Identify events, write event handlers

So: ⓐ On initiation (if needed)

ⓘ Identify all messages you will use, including the content

For each Process:-

ⓐ Write the send rule for each msg
(When will a msg of that type be sent?)

ⓑ Write a receive rule for each msg
(What does the process do on receiving a msg)

Add other types as needed:-

1) On a timeout
2) On a fault
3) etc

Dijkstra's guarded Commands:-

```
do
    guard 1 ⟶ action 1
    (Condition)   (What is done when the condition is true)
    guard 2 ⟶ action 2
od
```

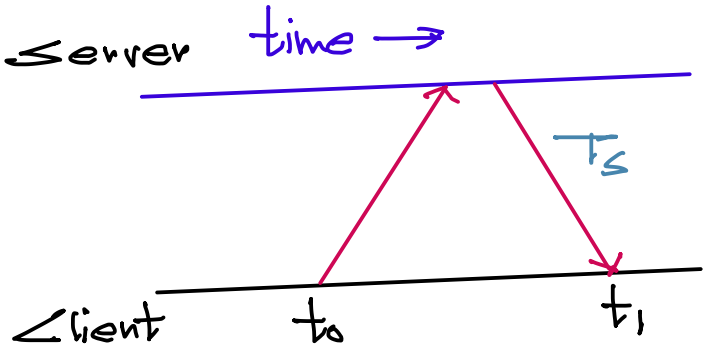
Note:- Both these ways but no constraint on the order of things done if (i) More than one event occurs at the same time (More than one guard true) and (ii) an event happens in the middle of handling of another event.

Q) Is the real world sync or async?

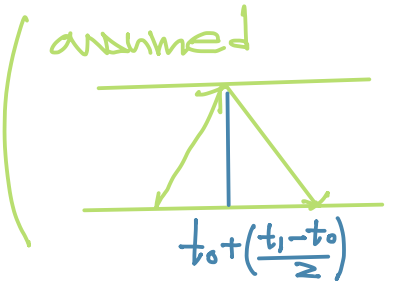
⇒ Async → But in practice, we make it sync with timeouts.

Time in DisSys

Cristian's Algo:-



Set $T_c = T_s + \left(\frac{t_1 - t_0}{2}\right)$

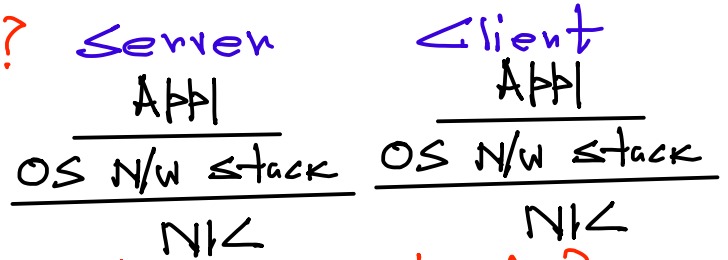


Max Error Possible? $\left|\frac{(t_1 - t_0)}{2}\right|$

Q) Source of Error?

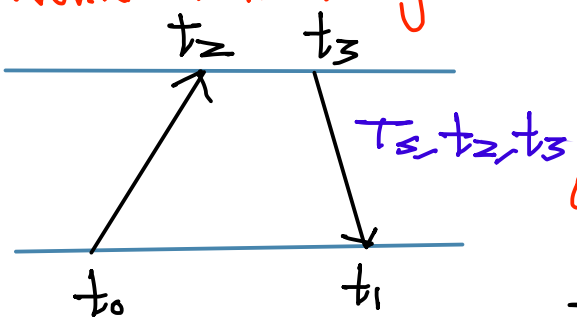
① Network

② Processing



Q) Would it help if we also know t_2, t_3?

helps!!

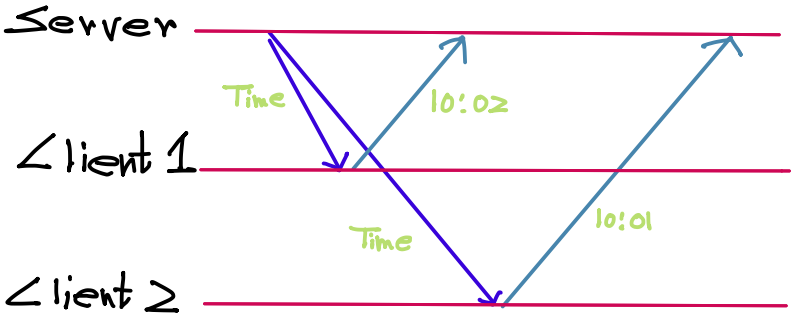


Can remove processing time

Q) But where does the server timestamp t_2, t_3?

→ If at app layer, not helpful

→ So, Need to timestamp t_2, t_3 as close to N/W as possible (Use NIC Timestamping)



Server sends

Client 1 −0.04

Client 2 +0.04

— Server adjusts for message delay (same as Christian's)

— Suppose after message delay

Client 1 10:23

Client 2 10:15

10:19 (say)

Client Average

Problem in advancing clock immediately (in some case):

May miss events scheduled to happen in the gap jumped.

NTP

pool.ntp.org

NTP pool

① Use hierarchy to scale

② Ask more than one time server.

— Compute may delay offset etc for each, choose a subset (clock filtering algorithm)

Good estimation of delay

—from the subset, select another subset to eliminate potentially bad clocks (Clock selection algorithm)

good clocks

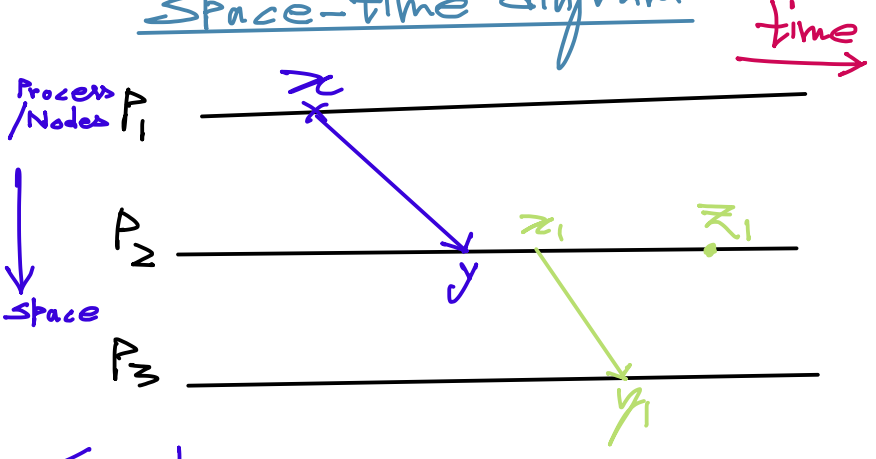
—Choose one find one based on ??

But not a total order

for eg, can't say $y_1 \to z_1$ or $z_1 \to y_1$

y_1 & z_1 are called "Concurrent" Events

Space-time diagram



$x \to y$
$x \to z_1$
$x \to x_1$
$x_1 \to y_1$
$y_1 \to z_2$
$y \to z_1$
$x \to z_1$

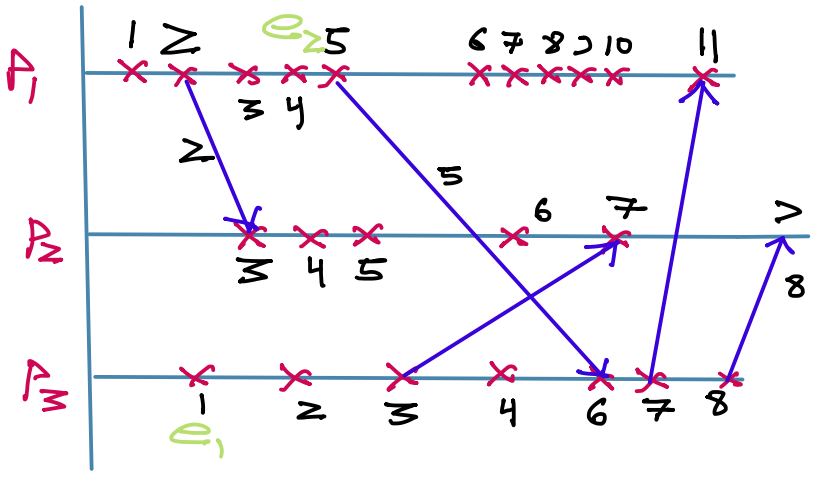
Events

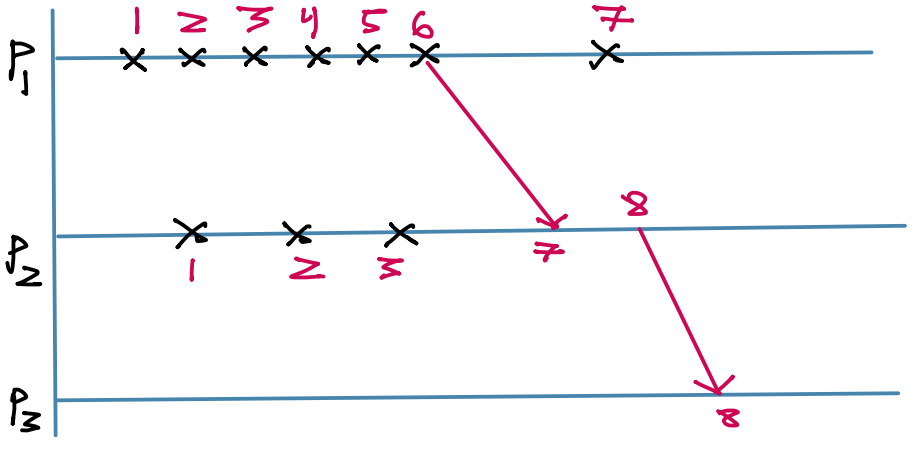
Send (Send Event) of a msg

Receive (Receive Event) of a msg

Anything else (local events)

X-events





$e_1 \not\to e_2$ $e_2 \not\to e_1$ $\Rightarrow e_1 \| e_2$

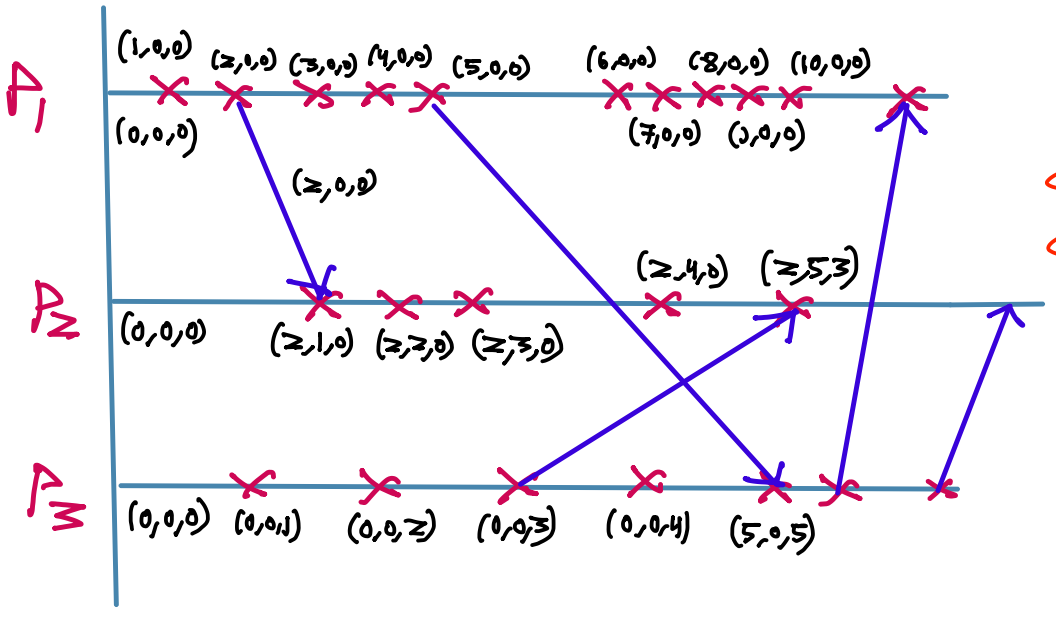
$\Rightarrow$ To make a total order,

timestamp of e_1 = $(1, P_3)$

timestamp of e_2 = $(4, P_1)$

Vector Clock:-

$C_i[k]$ = The no of events at P_k (Process k) that i knows of.



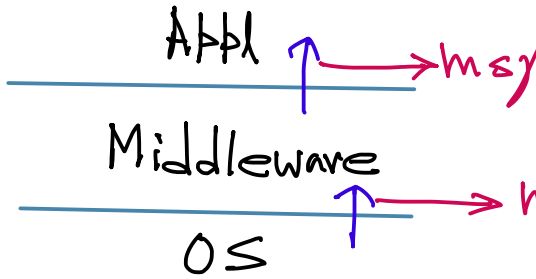
(P_1, P_2, P_3)

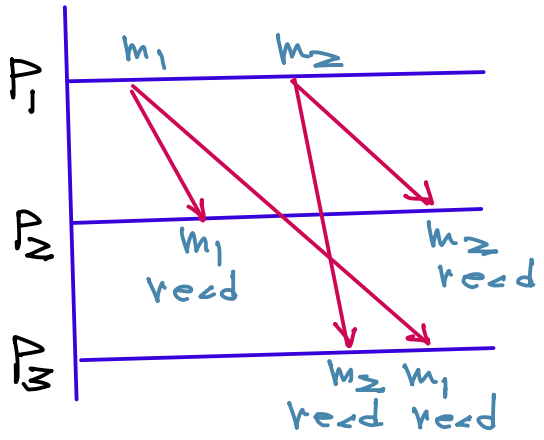
$e_1 = (3,0,0)$ $t_{e_1} \le t_{e_2}$

$e_2 = (3,0,5)$ $t_{e_1} < t_{e_2}$

$e_3 = (0,0,3)$

$e_1 < e_3$ & $e_3 < e_1$ $\Rightarrow e_1 \| e_3$





Atomic Order $\equiv$ Total Order

Will not be delivered to until m_1 is rec'd

FIFO, Causal, Atomic — Base

Can have Atomic FIFO, Causal Atomic

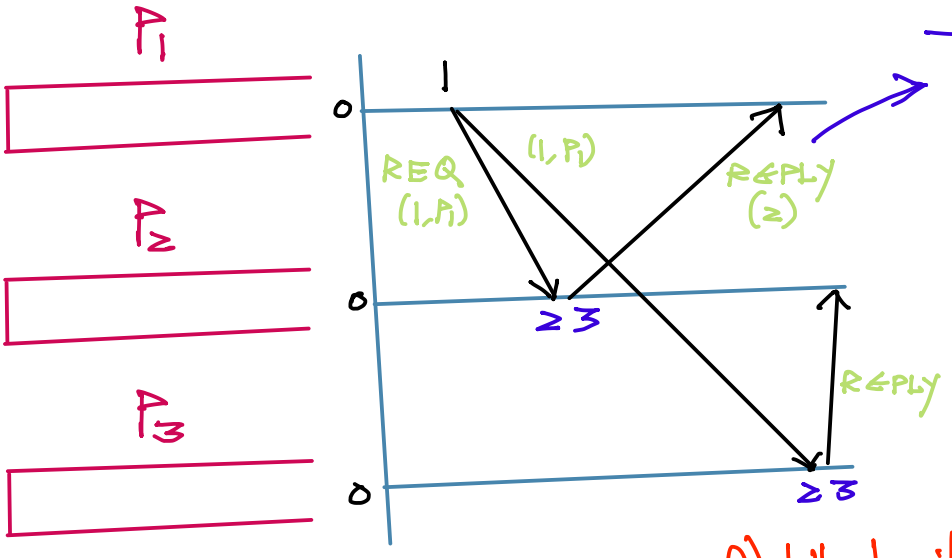
Q) What "event" are important for the "Clock"?

⇒ Depends on the application

Q) Show lamport clock does not work for this broadcast

Metrics for evaluating a dist^{n} mutex algo

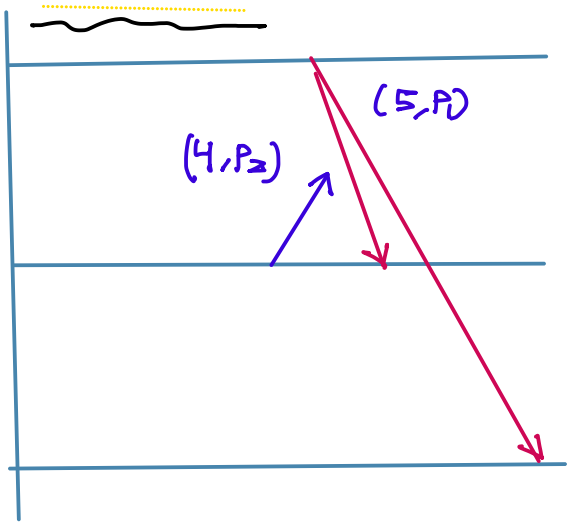
1. # of msgs per CS entry req served
2. # Synchronization delay (Gap in time between one process leaving CS & the next waiting process entering CS) → Sync system (Otherwise no Guarantee)
3. Response Time

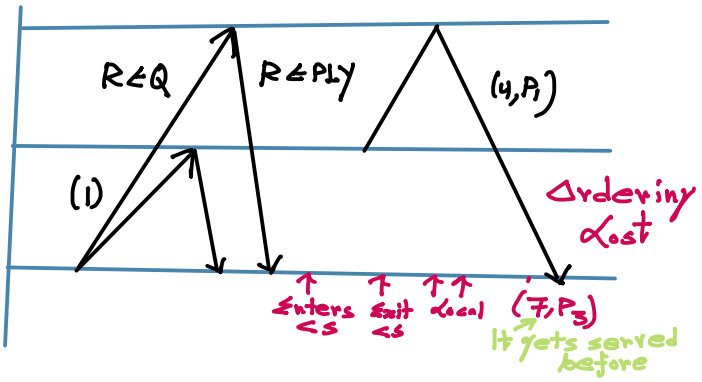


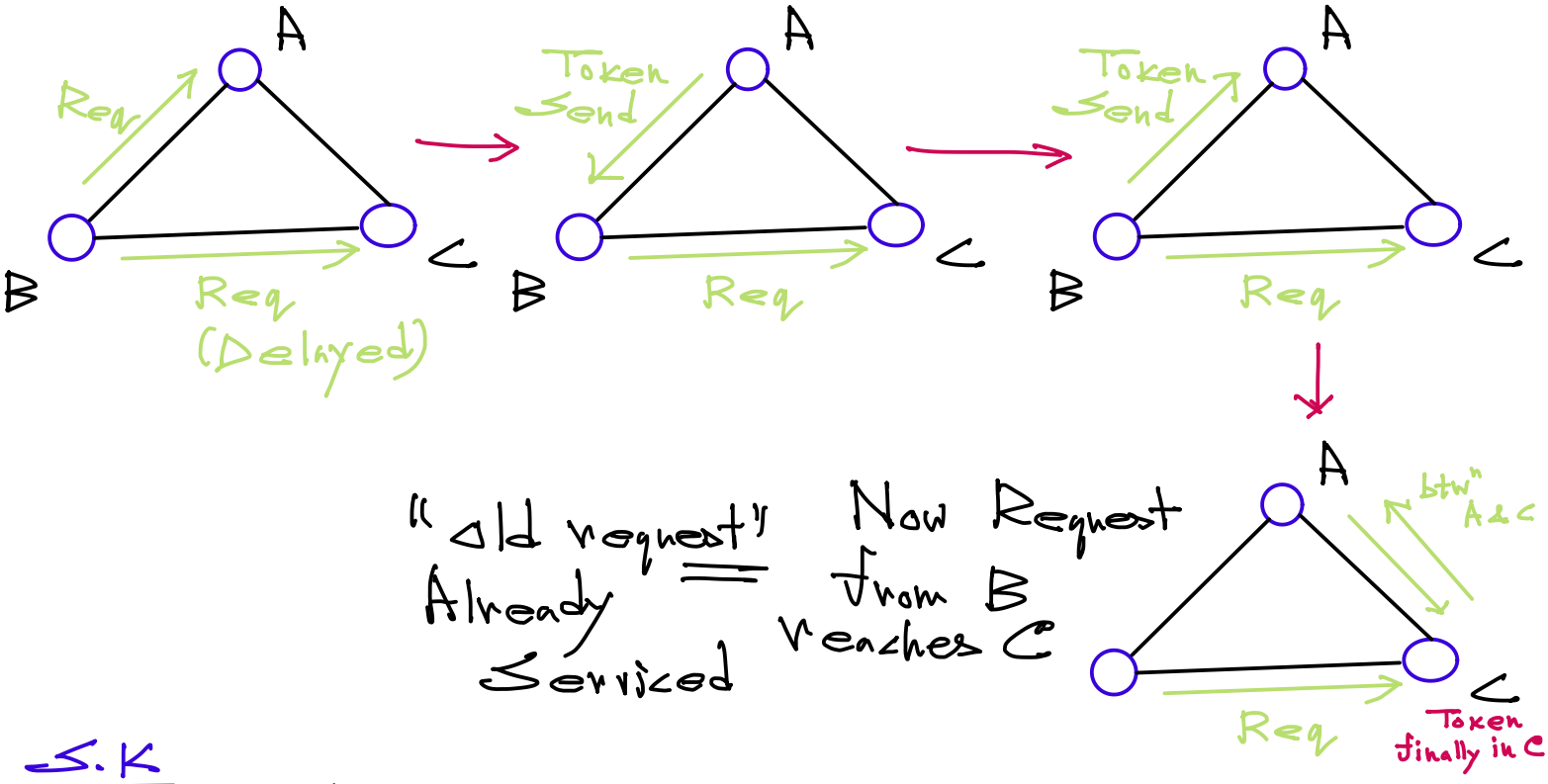
The REPLY ensures that all older messages were delivered (FIFO)

Q) What if I just hold the token if I will be making consecutive requests?

→ Adaptive Raicainal - Carvalho







S.K

2 types of messages

① Request

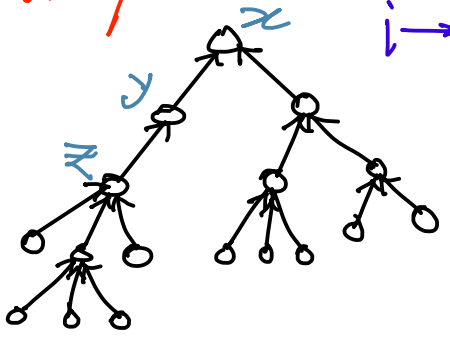
② Token

Token Idle :- Node not using token and no pending request in queue

Assume, one node can make the request only at a time.

Q) Can you build a token-based mutex algo along this line?

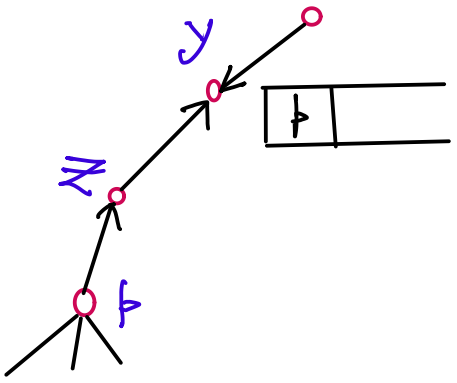
$i \to j \equiv$ to_token(i) = j



Naive algo :- Keep token always at root
- Ask root to send token if needed
- Return to root once down

Better :- ① Make actual root floating (but keep the def^n (root = node with token))

② Let each node "take charge" for all nodes in the subtree rooted at it (Keep queue, Get token etc.)



At the same time token reaches y, y makes a request R

Possibility 1:-

Token is handled before R
- token sent towards P
- To token(y) = Z
- But R in Q
- Send a Req for R to Z

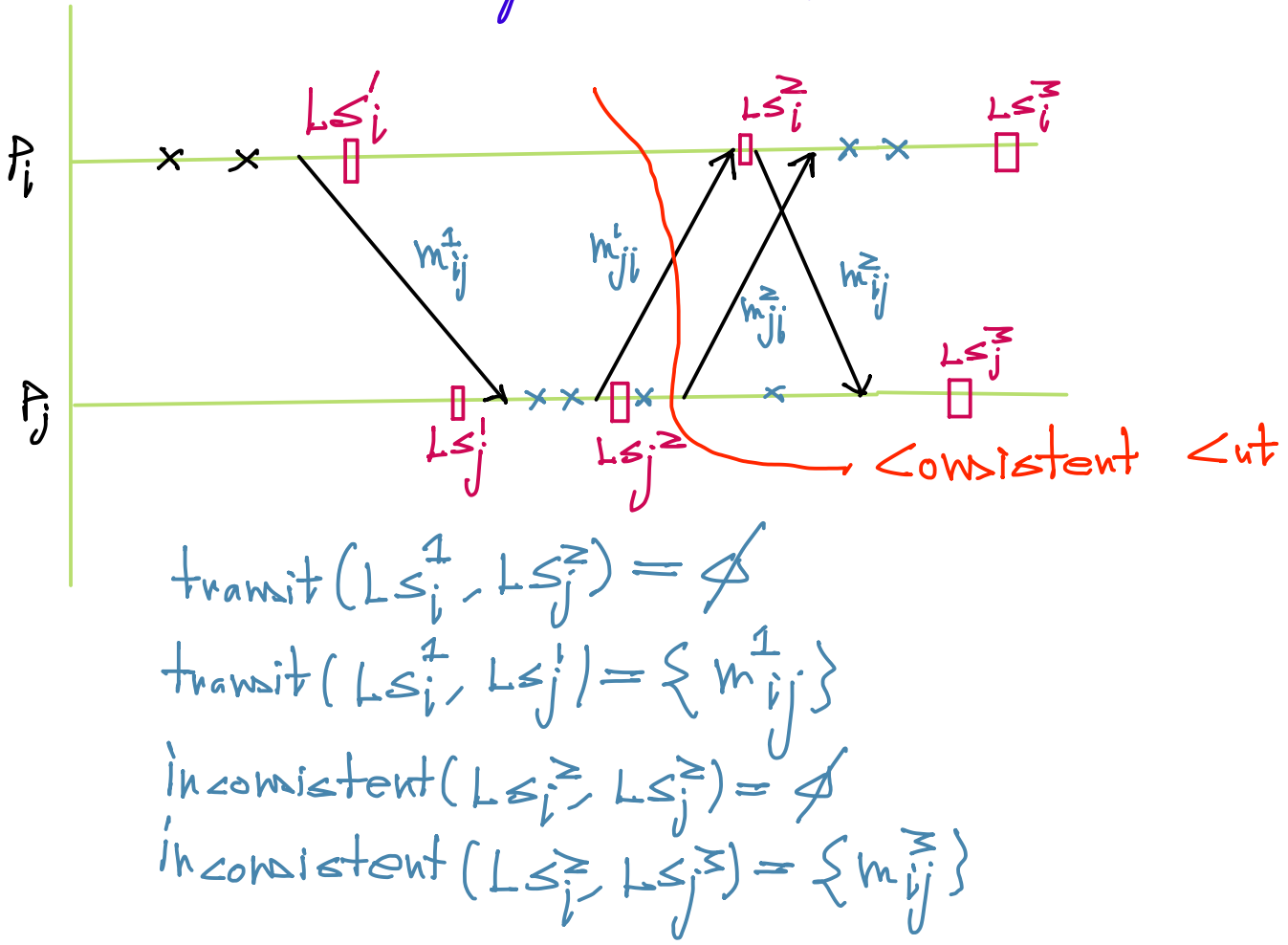
Possibility 2:-

R is handled before token

Q) Ensure all sort of atomicity

Stable property = a property (predicate) that once true, stay true until some external intervention happens.

For token ring, stable property: Exactly one token. But may keep on going



$transit(LS_i^1, LS_j^2) = \phi$

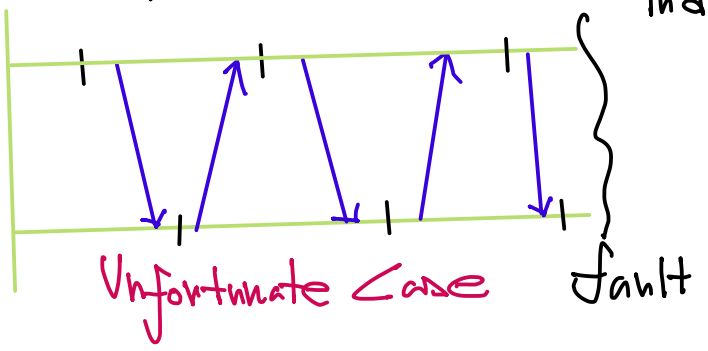
$transit(LS_i^1, LS_j^1) = \{m_{ij}^1\}$

$inconsistent(LS_i^2, LS_j^2) = \phi$

$inconsistent(LS_i^2, LS_j^3) = \{m_{ij}^2\}$

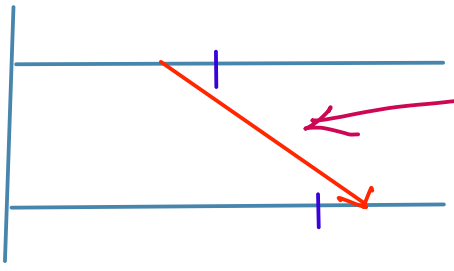
Q) Is consistent cut useful?

⇒ "Replay" from the cut.



Independent / Async Checkpointing

Domino Effect



Do we need to replay it?

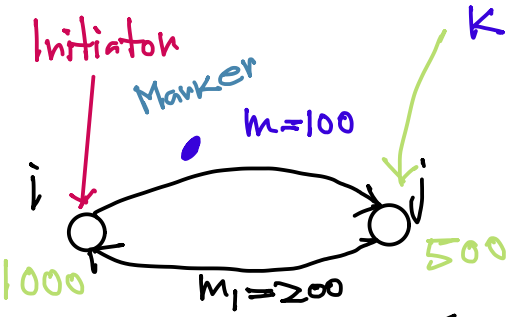
⇒ Depends on application

Q) Is consistent cut any good or should we always take strongly consistent cut?

Depends,

- if comm^n mode is unreliable communication, consistent cut is fine as the msgs in transit can get lost any way, application should take care of it.
- if model is reliable comm^n, consistent is still ok but msg in transit has to be replayed first if recovery done from the cut.

C.L!- Keeps channel states explicitly



Send m 200

taken snapshot,

sends marker M

S_i=100

gets m_1, $C_{ji}=m_1$

Send m_1 300

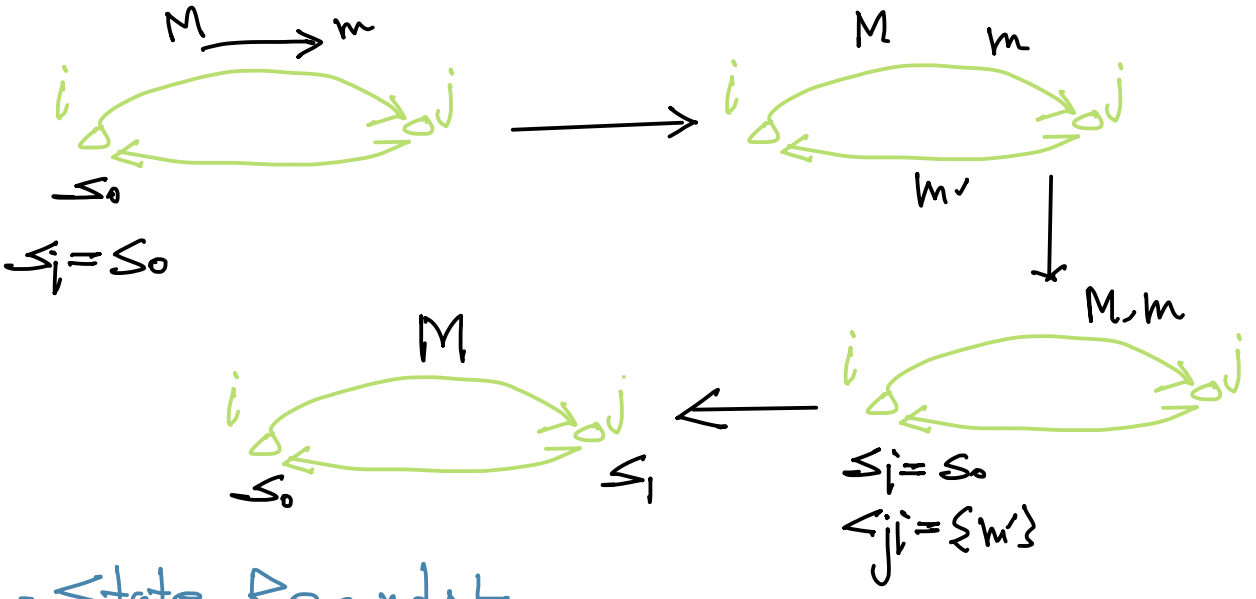
gets m 400

gets M S_4=400

$C_{ij}=\phi$

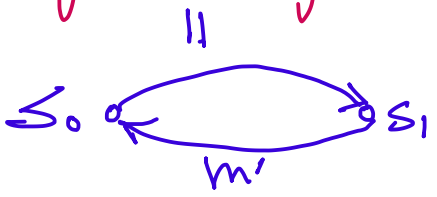
$\langle S_i, S_j, C_{ij}, C_{ji} \rangle$

$\langle 300, 400, \phi, m_1(200) \rangle$



State Records:

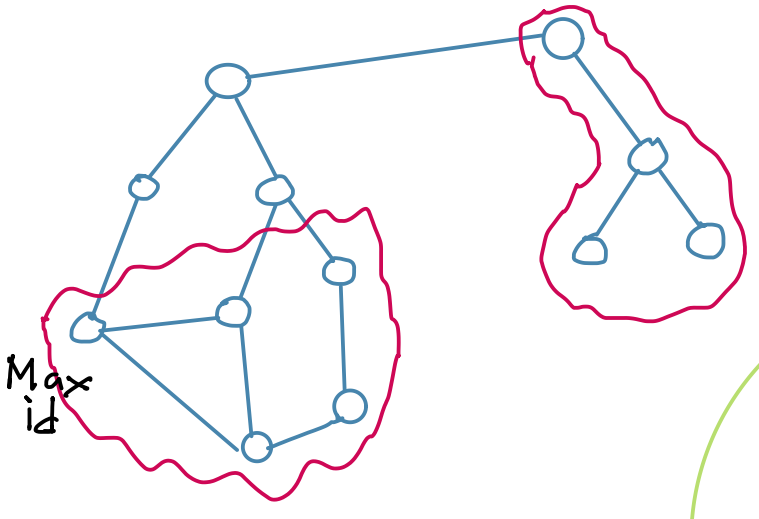
$s_i = s_0$, $s_j = s_1$, $\langle ij \rangle = \phi$, $\langle ji \rangle = m'$



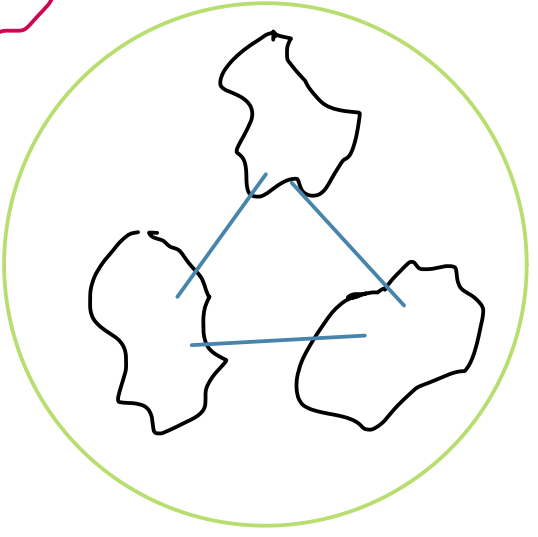
Q) What if no FIFO?

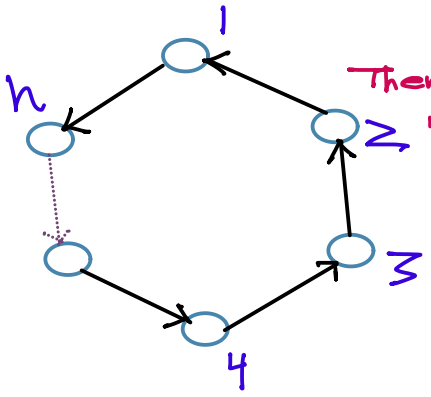
— Piggyback a flag to distinguish msg sent before & after taking state (The same work done by Marker in C.L)

⇒ How do I know how many messages?



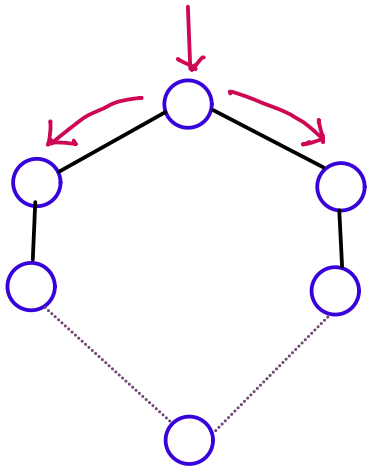
Parent
↓
Id of the root





1 starts first – eaten up by n — 1
Then 2 " next — " — 2
2 " 3 " " " — 3
⋮

$O(n^2)$



Phase No	Max no of nodes left	Max # of msgs sent per alive node
0	$n/2^0$	$2.2.2^0$
1	$n/2$	$2.2.2^1$
⋮	⋮	⋮
$K = O(\log n)$		$2.2.2^K$

$2.2.2^K$: Send and Reply; both side; link length; Per Node

Total = $O(n \log n)$

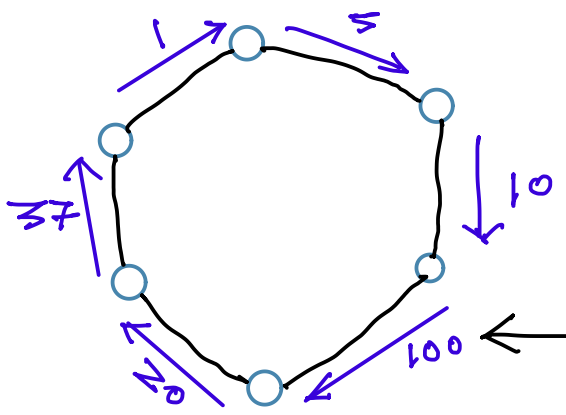
Q) Study Unidirectional case
Q) Can we do better than $O(n \log n)$? (Not comparison based)

Round in a synchronous system!

Can be defined in several (slightly diff ways)

Here's how We will use!-

Round = Max time for processing (α)
+ Max time for sending a msg (if needed to send)
(& the msg reading the receiver) (β)



Add delays in the node
⇒ delay ↓ for id ↑
Q How much proportional?

Should go faster than others ←

Peterson
Bully algorithm

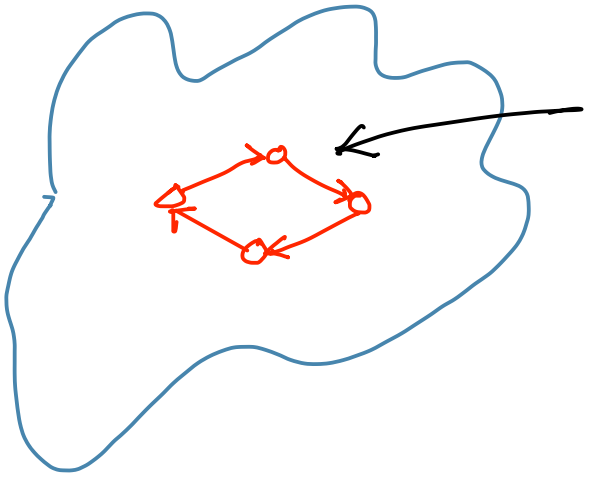
⇒ Round difference can be at most d
(if not started together)
(Exam: Assume Start together)

Reusable Resource:-
- Resource can be re-used
  - eg. A lock, A semaphore

Consumable Resource:-
- Resource that can not be reused
  - eg. A message

AND deadlock:- Process wait for a set of resources and all of them must be acquired before the process can start.

OR deadlock:- Any one can wake up the process.

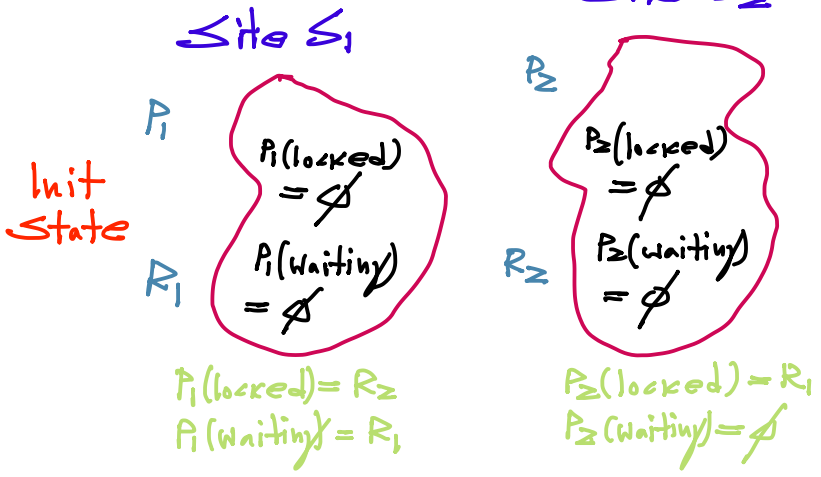


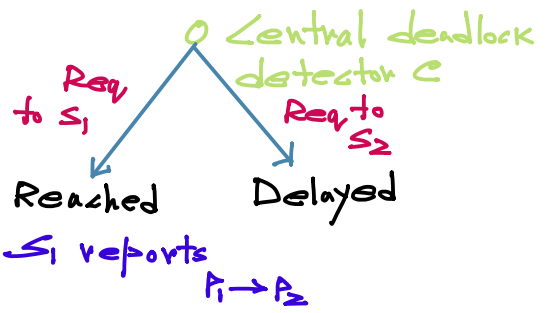
No need for whole snapshot

AND deadlock:-
- Cycle in WFS ≡ deadlock

OR deadlock:-
- Knot in WFG ≡ deadlock

Ho-Ramamoorthy Two-Phase algo





Now, P_1, P_2 finishes
So back to init state
! More resource requests by P_1, P_2

P_1(locked) = R_1
P_1(waiting) = ϕ

P_2(locked) = R_2
P_2(waiting) = R_1

Now, T_1's request reaches S_2

S_2 report $P_2 \to P_1$

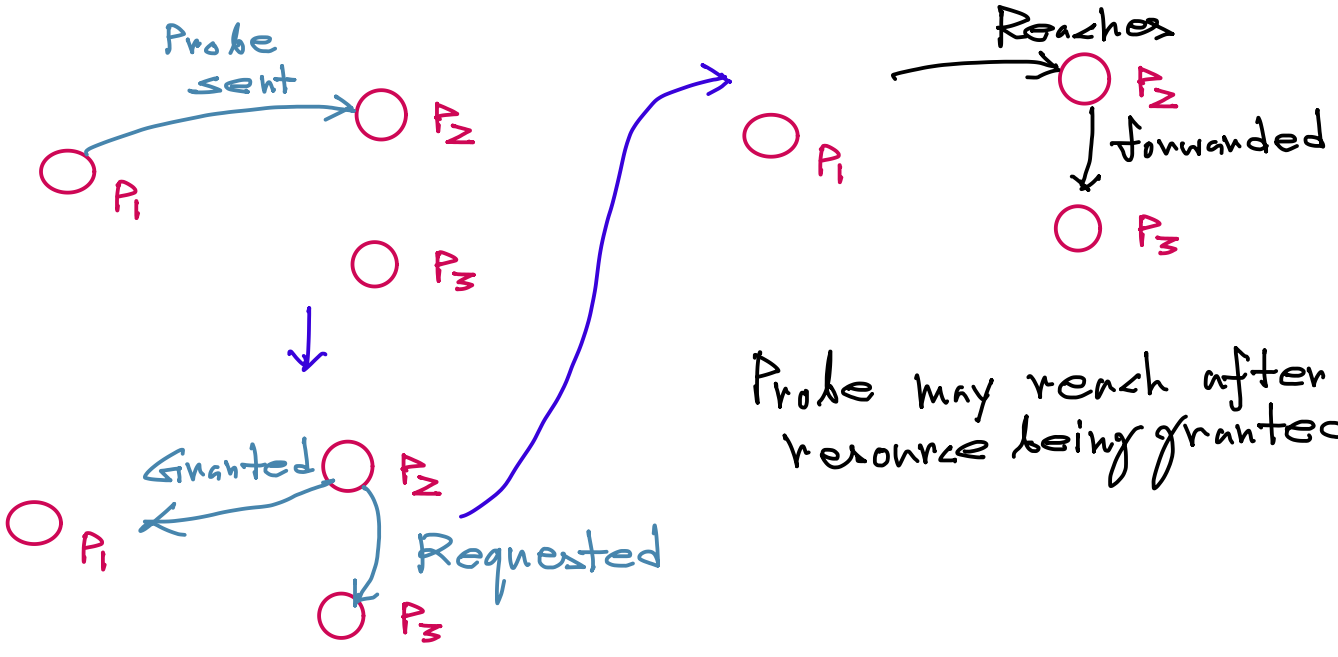
Says deadlock = Phantom Deadlock

Q) Do one phase

⇒ Keep two tables
1) What resource every process locked and waiting for
2) What process has locked a request and who are waiting for it.

Idea:- One table will report that no deadlock through inconsistency

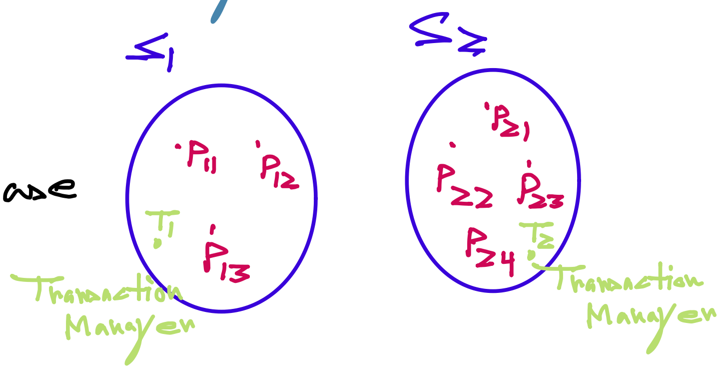
Problem with simple probe for WFG

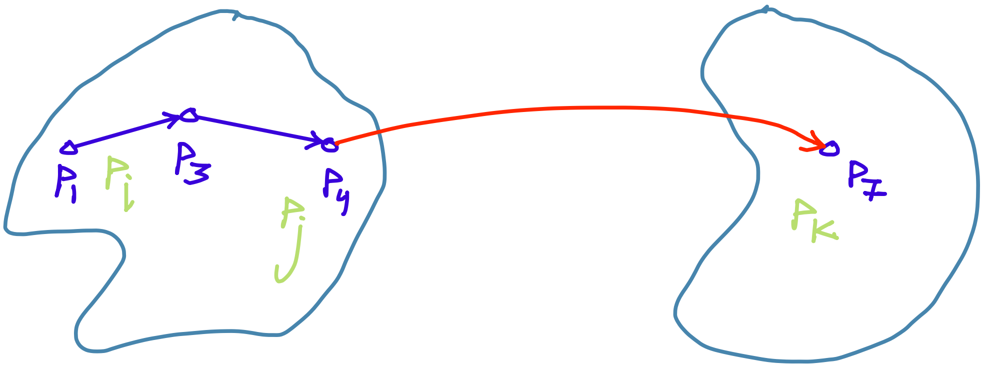


Probe may reach after the resource being granted

Background

Think locals
Part of a
dist^r database
(DDS)





XTC Algo:-

v (_, w, _, u, _ _)

NUM = { _ W _ _ }

u

L (_ _ _ _ _)

↑

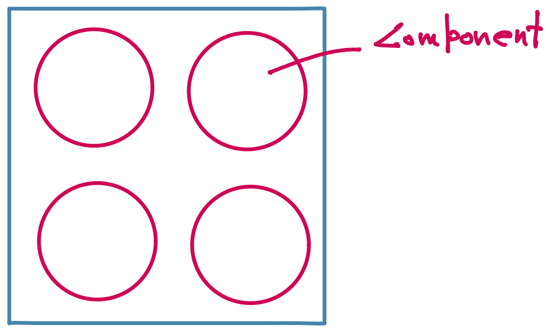
v

Q) V & u both add each other (Even for arbitrary ordering) or don't

Fault – Some component has failed

Error – Manifestation of fault, may or may not cause the system to fail

Failure – System fails



System

Reliable ⇒ Available

Available ⇏ Reliable

Stable Storage

– Storage that is available even after the crash of the component.

Building Blocks

1. Stable storage
   - storage that persists even when a system crashes
   - Allows state to be accesible even after crash

Q) How do you make your h/w more reliable

1. Servers — CPU, memory, disk, Power supply, fan
2. H/W — Switches, cables
3. UPS
4. AC

How do you make a Server reliable?

→ CPU — Nothing specific
→ Memory — ECC RAM
→ Disk — RAID
→ Power supply /Fan — dual redundant

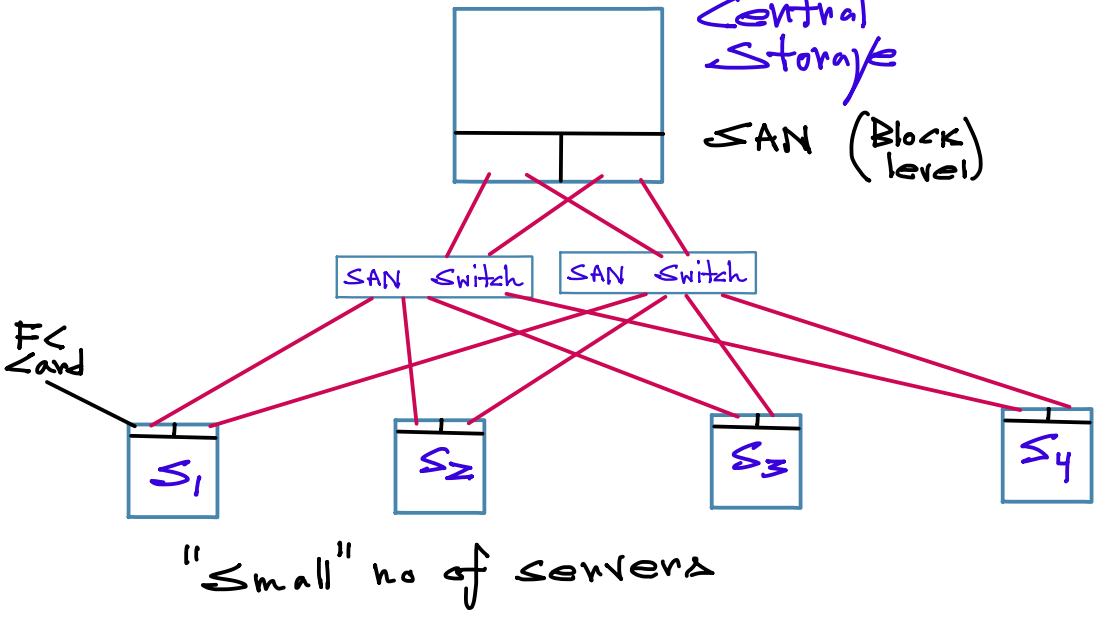
N/W?

Switch — redundant power supply/fan
extra ports
redundant control cards for chesis based switch

Q) How do you make storage reliable?

→ RAID "hot swap" disks

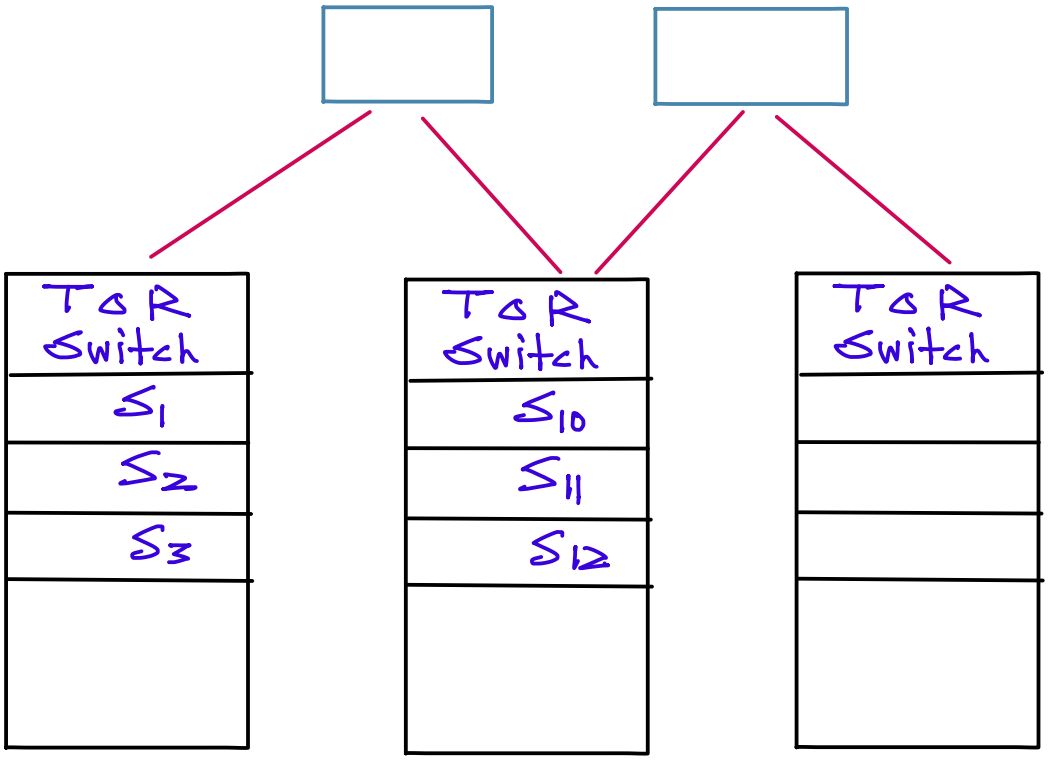
Still if the box fails as a whole (say powersupply failed)
→ Storage is not accesible.
So, take it out of the box.



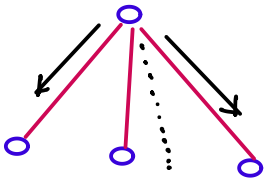
"small" no of servers

Data Center

TOR = Top of Rack



Byz Gen



	I/P	O/P
B Agree	one Value	one Value
Consensus	many Values	one Value
IC	many Values	many (vector) Value

Q) If I know one, can I do the other?

Third (Implied) Condition
in all three version

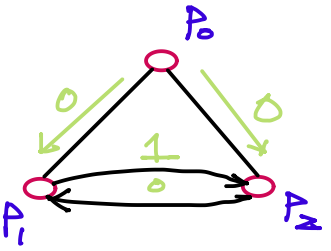
- All non faulty process eventually decide a value

Agreement under Byzantine Fault!

Why assume only 2 values 0 & 1

Impossible with 3 process & one fault. $n=3, m=1$

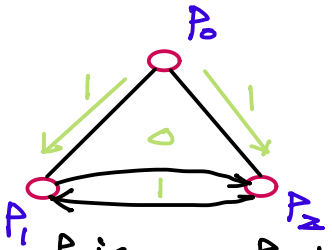
Case 1:-



P_1 is faulty

P_2 must decide on 0

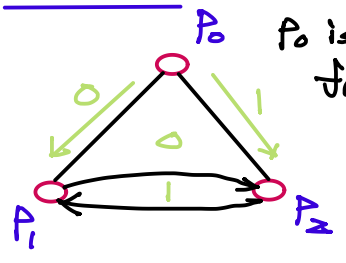
Case-2:-



P_1 is faulty

P_2 must decide 1

Case-3:-

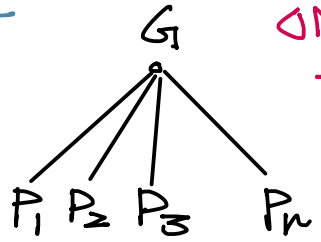


P_0 is faulty

P_1 must decide 0 (from case 1 decision of P_2)

P_2 must decide 1 (from case 2)

L-S-P!-



OM(m) – G sends value to all

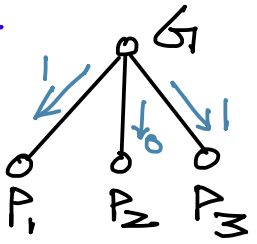
Say value received by P_i from G is V_i

Each P_i starts a OM(m-1) with P_i as general & V_i as the value sent by it. P_i^x = P_i's decision on what G sent to P_x.

Each process maintains a vector

So, OM(m-1) for P_i is for all other n-2 nodes to decide on what value is sent to P_i in first round.

n=4, m=1!-



	1	2	3
P_1	1	0	1
P_2	1	0	1
P_3	1	0	1

OM(1) → by G

OM(0) → by P_1

→ sends 1 to P_2 & P_3

OM(1) → by P_2

→ Sends 0 to P_1 & P_3

OM(2) → by P_3

→ Sends 1 to P_1 & P_2

x in $OM(x)$	No of invocation	No of msgs/ invocation
n	1	$n-1$
$n-1$	$n-1$	$n-2$
$n-2$	$(n-1)(n-2)$	$n-3$
⋮	1	1
△	1	.

$$\text{Total} = O(n^m)$$

Q) $n=7, m=2$

Q) Given Byzantine Agreement soln, can you do consensus?

⟹ Each process has a value
" " starts agreement pr with its value

At the end, each process has a vector
Where i-th component = Value P_i started Byz agreement with

(So you solved IC)
↓ Take majority
Consensus Solved

Q) Other way round?

I/p: one general with a value, $n-1$ lieutenant.

General send to all lieutenants.
lieutenants $(n-1)$ solve consensus to agree on value.

Notes about $OM(m)$ algo:-

For the $n=3, m=1$ impossibility
note that the problem case was if the general is faulty.

If general is not (known), problem is trivial.
But the problem is that you do not know whether the general is faulty or not.

OM(m) – started by 1 node

↓ causes

(n-1) OM(m-1) to start.

We reasoned if OM(m-1) is so what they are supposed to, OM(m) will work.

Case 1:- General is faulty

So rest of the nodes,

$n_1 = n-1$ nodes, at most $m_1 = m-1$ faulty nodes.

$n_1 > 3m_1$ holds ⟶ So OK

Case 2:- if general is non-faulty, all m faults can be in (n-1) node

So, $n_1 > 3m$ may not hold.

So, is OM(m-1) supposed to work?

Works since general has sent the same value to "most" of the process

Assignment 2:-

To be submitted hand work on paper

Work out 4-5-P if $n=7, m=2$

When (1) General is faulty (sends 0 to P_1, P_2, P_3 & 1 to P_4, P_5, P_6)

(2) General is non-faulty (Choose P_1, P_2 as faulty)

Crash tolerant agreement:-

Suppose at end, there are 2 process P_i & P_j which decides on 2 diff minimum. Say V_i and V_j

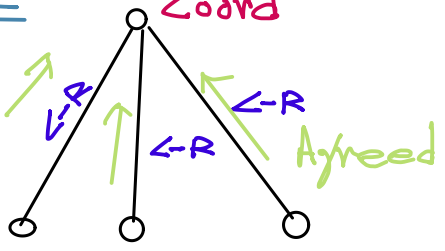
& say $V_i < V_j$

So, P_j does not have V_j in its set. V_j must have been "hidden" for P_j

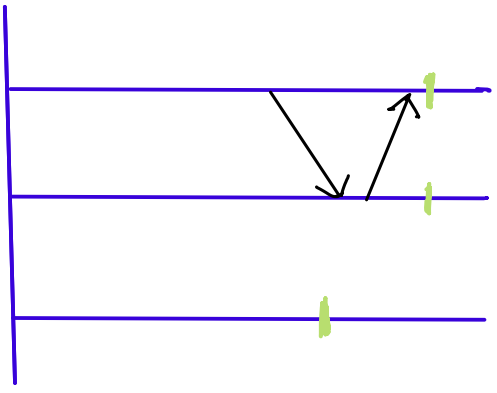
Can happen only if at any round, whoever knows about V_i, fails before sending to P_j (1 failure per round)
Can be hidden for a maximum of m rounds
Still has a failure free round at end of which P_j will know V_i

Adv:— You can check convergence after every round.
⟹ No new process failed means convergence

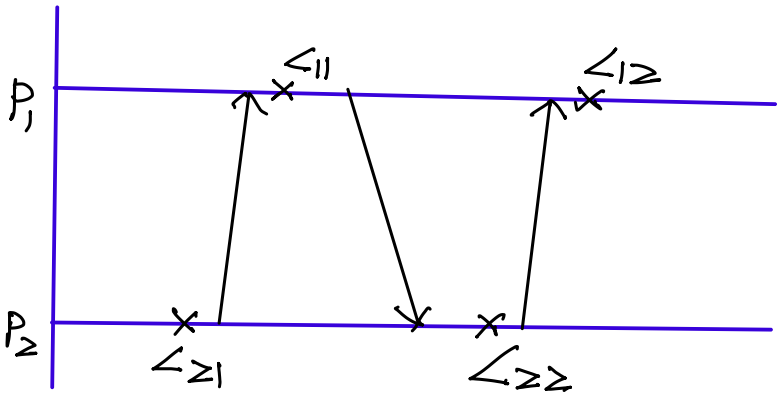
2PC



Commit ⟹ Abort



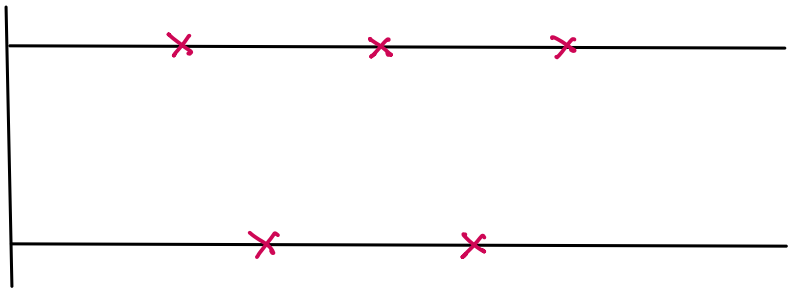
Async / Uncoord Checkpoint



(C_{12}, C_{22})
→ Not Consistent
Need to coordinate which checkpoints to take from each process to get a consistent recovery line.

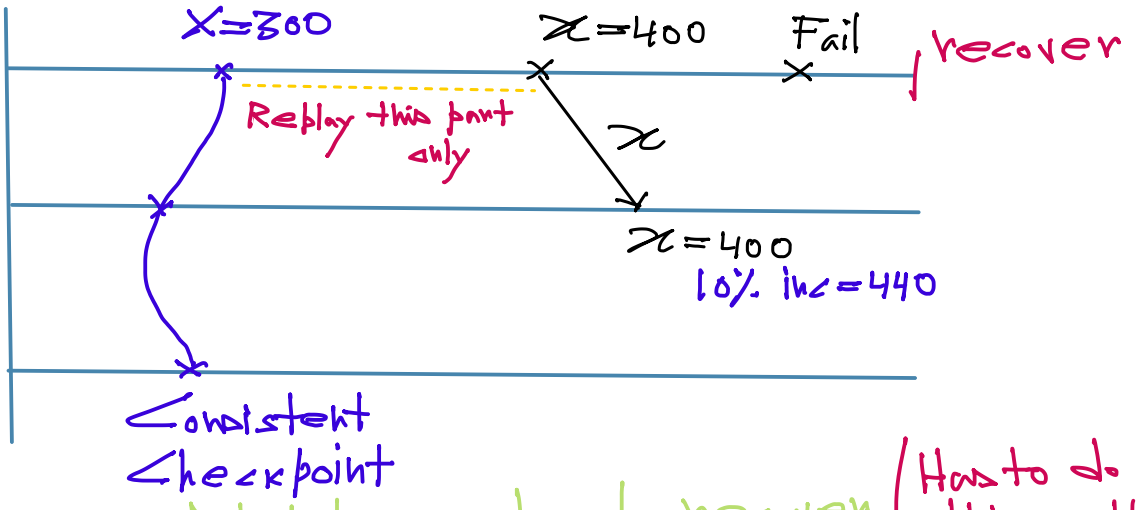
Domino Effect

Communication Induced

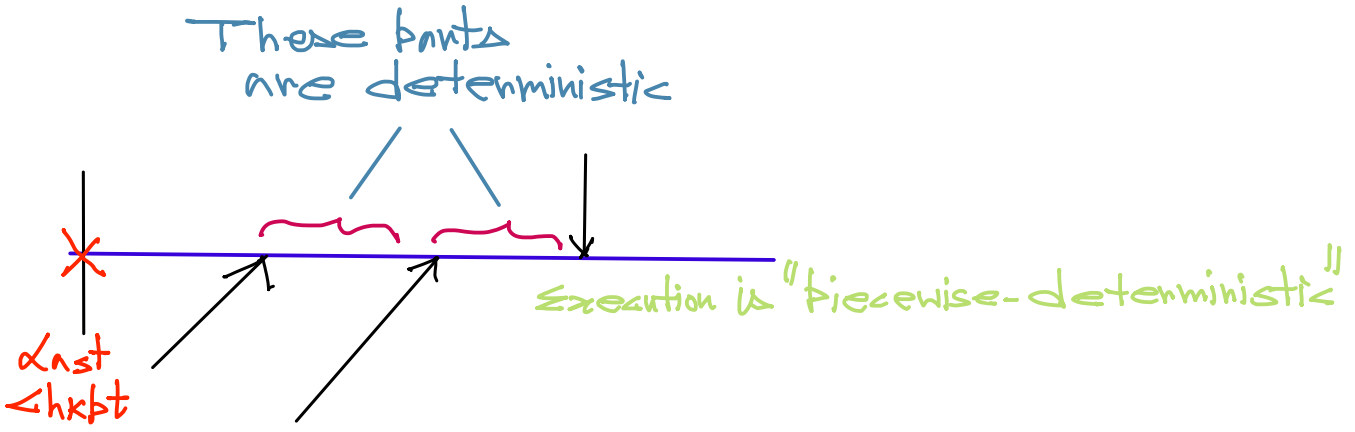


No communication so far
So could have done Uncoord checkpoints

None of Async, Sync, Comm-Induced handle the output commit problem



① only the failed process has to recover (Has to do lot of things although)



Koo-Teug:-

Checkpointing Protocol:-

Idea:-

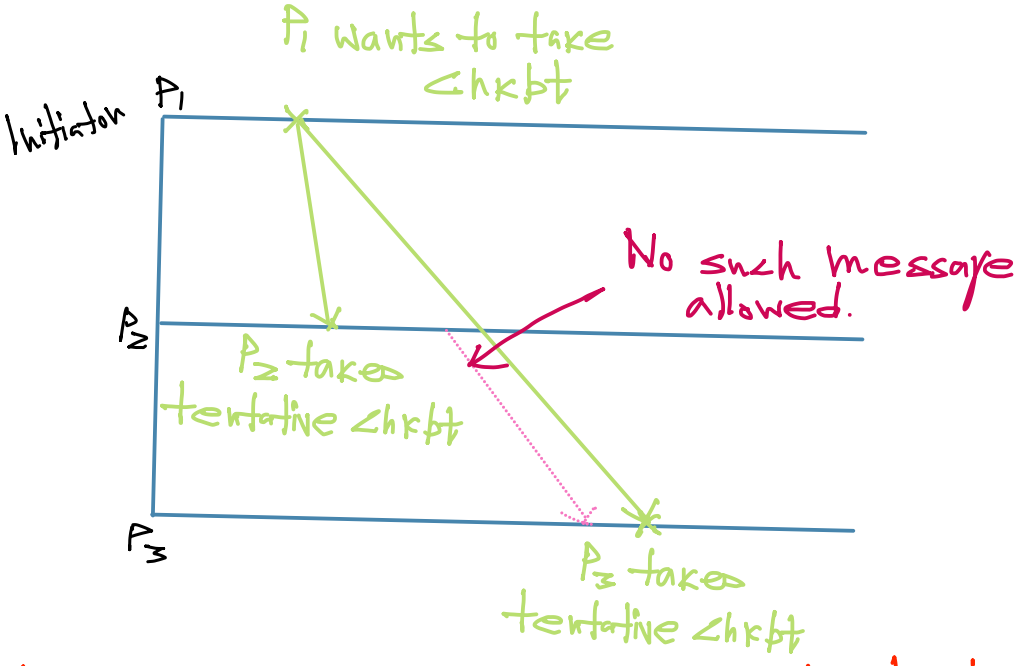
1. One process initiates
   - takes "tentative" checkpoint
   - asks other process to take checkpoint

2. Any other process, takes a "tentative" checkpoint on receiving the request & answers "yes", else answers "no"

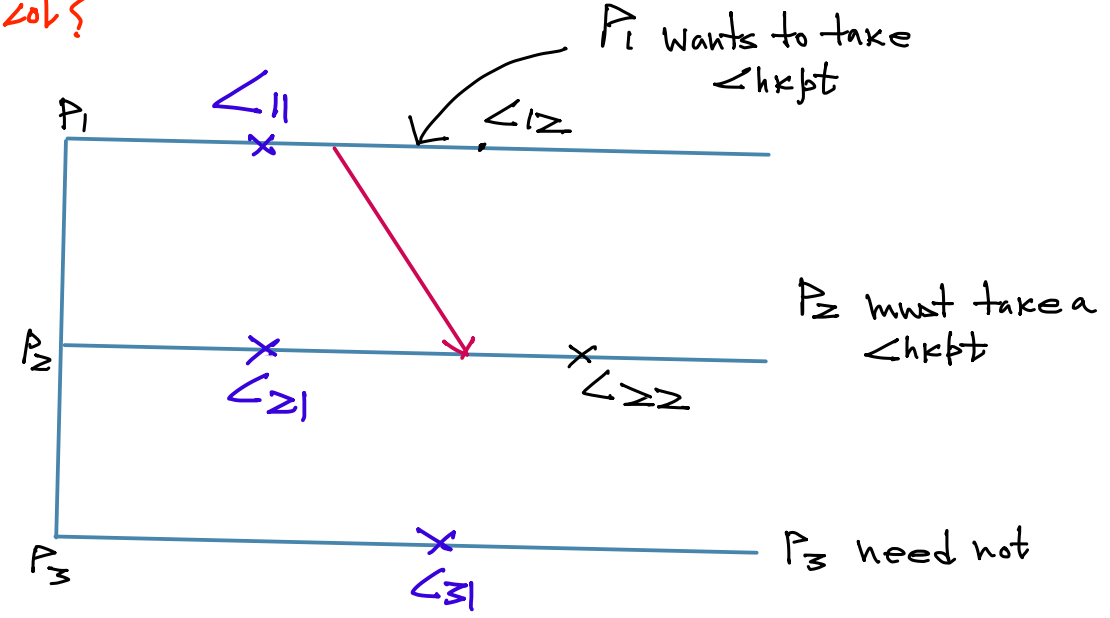
3. Coordinator / Initiator after getting all responses
   - If any one "no", asks everyone to discard the checkpoint
   - Else makes its own checkpoint "permanent", asks others to do so.

4. If asked, other makes it permanent.

Between the time a chkpt is tentative & permanent, do not send any other messages.

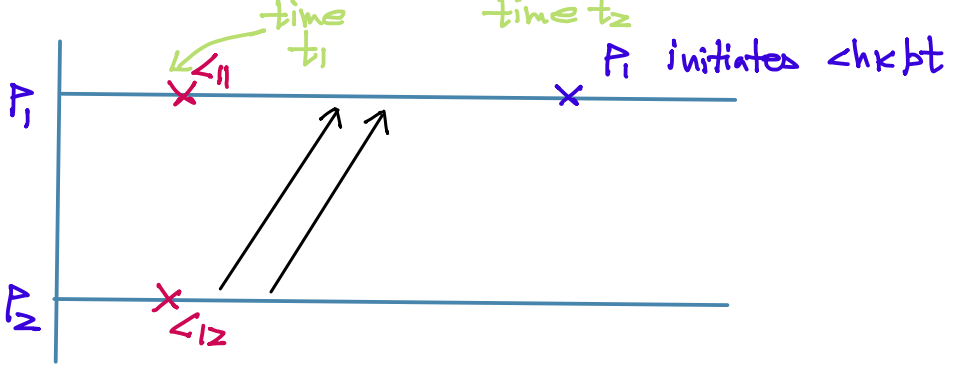


Q) Do all processes need to be part of checkpointing protocol?



(C_{12}, C_{22}, C_{31}) = Consistent

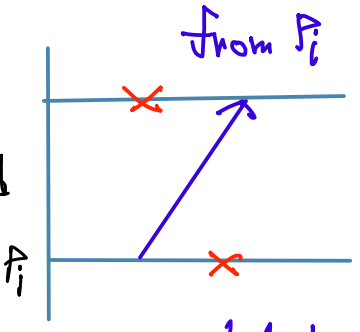
So all process need not be involved. So, who all?



Note, between t_1 & t_2,

① If P_i has not recd any msg from another process P_j, then P_i need not take chkpt

② If P_i has recd a msg from P_j, but P_i has already included in its last chkpt, P_i need not take another chkpt.



⟹ Each appl msg has a distinct, monotonically increasing label

Each process keeps the following data structures:—

$last\_label_x[y]$ = label of last msg received by x from y after x's checkpoint

$first\_label\_sent_x[y]$ = label of first msg sent by x to y after its last checkpoint

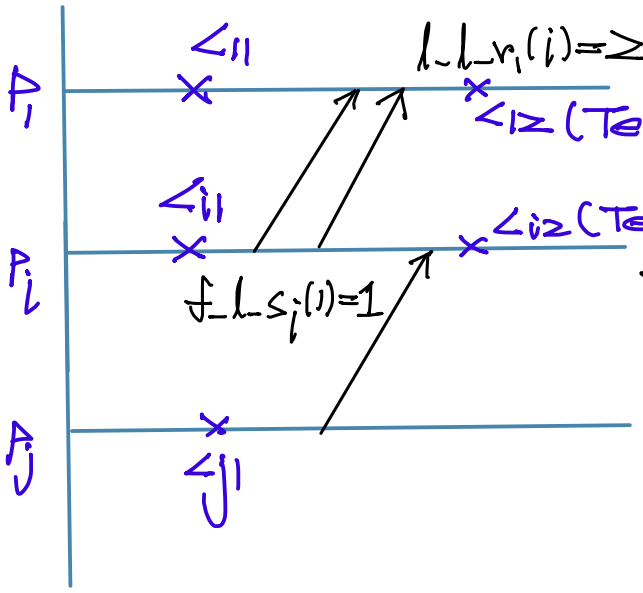
So, P_i = initiator
= sends only to process P_j such that

$$last\_label(j) = NULL$$

On receiving a req from P_i,
P_j does this.

P_j takes a chkpt if $\underline{last\_label_j[i]} \geq first\_label_j[i]$

sent in req msg from P_i to P_j



So, P_j must chkpt

→ But then by same logic P_k must also take chkpt.
Even though the initiator has not sent a request

So, how P_j starts as an initiator on behalf of P_i

A non-communicating node will lose lost data.
So, it will itself initiate one

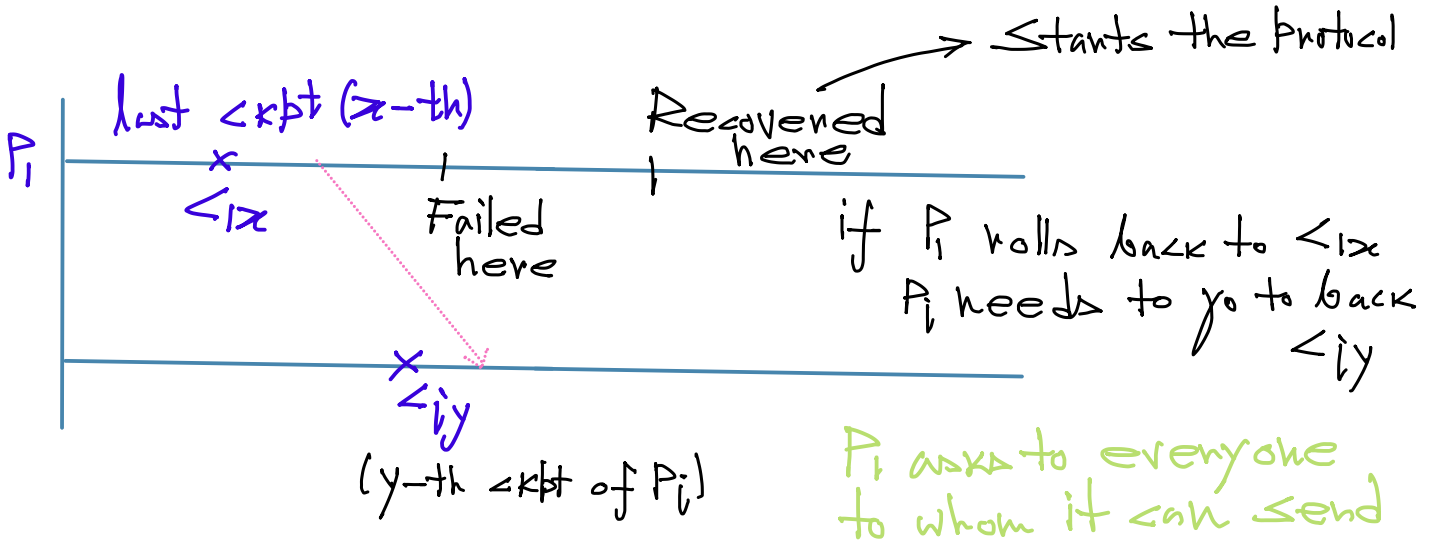
Rollback Recovery

Idea!
- One initiator
- Everyone rolls back to its last checkpoint

⇒ Consistent but everyone loses somewhere

Q) Do everyone need to rollback?

P_i = initator
= failed & now trying to recover



P_i asks to everyone to whom it can send message.
(Also send the id of last msg sent before ckpt)

P_j will decide if to roll back

But