DS

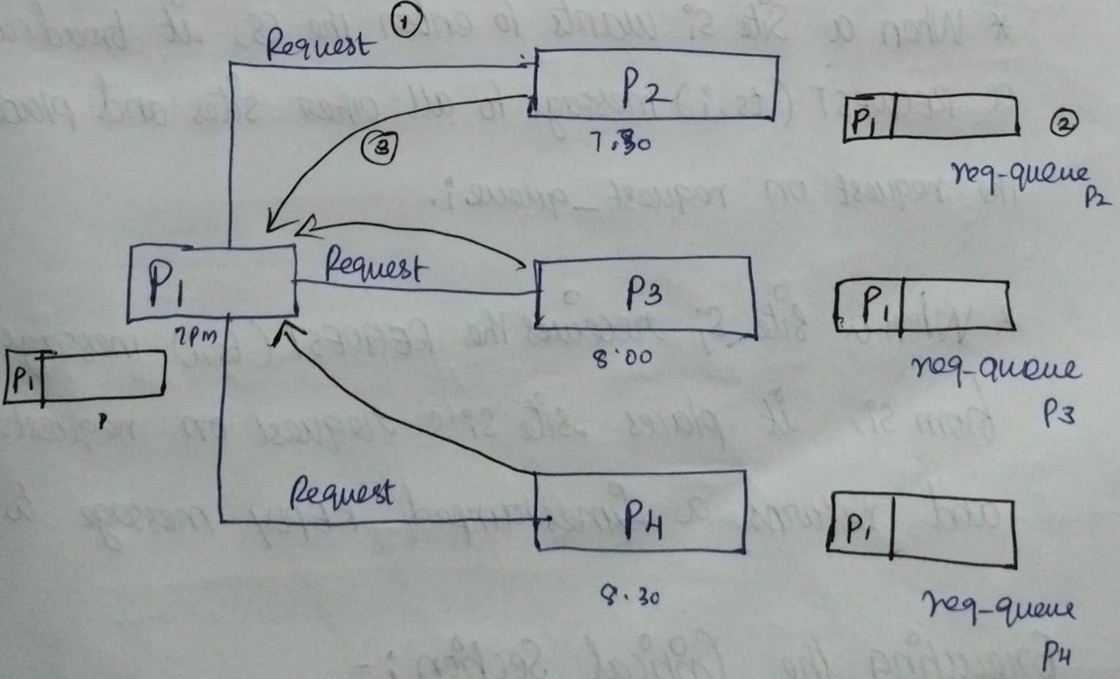
UNIT-3

I LAMPORT'S AHGORITHM FOR MUTUAL EXCLUSION:

* Lamport Algorithm is a method to used to ensure mutal exclusion in distributed systems.

* Mutual exclusion means that only one process can access a shared resource at a time to avoid conflicts or error.

* The algorithm is fair in the sense that a request for CS are executed in the order of their timestamp and time is determined by logical clocks.



CS enter

① All OK.

② All OK timestamp > req-timestamp.

③ Own queue => first.

CS Exit

① Remove own Queue entry.

② RELEASE msg to all.

③ Pj remove process Pi from its queue

ALGORITHM :-

Requesting the Critical Section :-

* When a Site S_i wants to enter the CS, it broadcasts a REQUEST (ts_i, i) message to all other sites and places the request on request_queue$_i$.

* When a site S_j receives the REQUEST (ts_i, i) message from S_i, it places site S_i's request on request_queue$_j$ and returns a timestamped REPLY message to S_i.

Executing the Critical Section :-

Site S_i enters the CS when the following two conditions hold :

L1: S_i has received a message with timestamp larger than (ts_i, i) from all other sites.

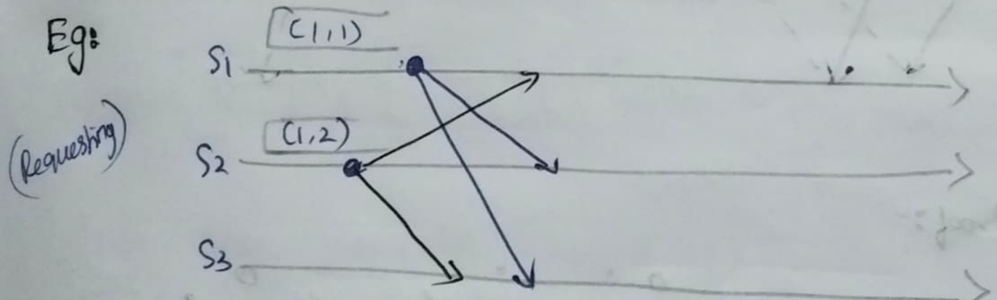
L2: S_i's request is at the top of request_queue.

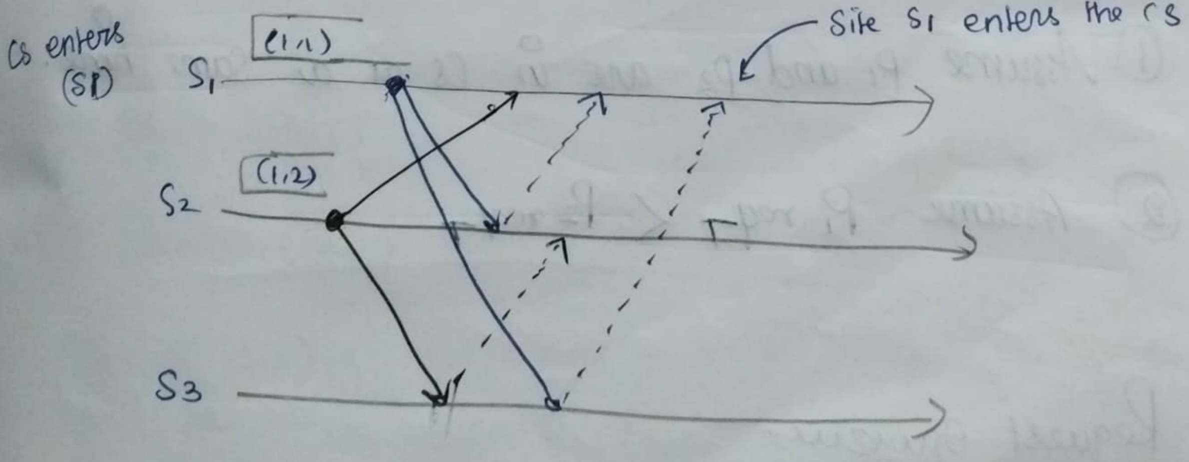
Releasing the Critical Section:-

* Site S_i, upon exiting the CS, removes its request from the top of its request queue and broadcasts a timestamped RELEASE message to all other sites.

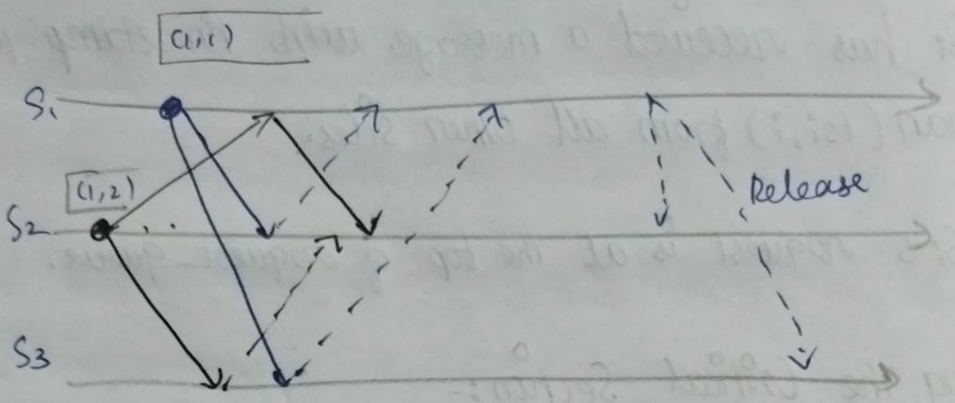
* When a site S_j receives a RELEAS message from site S_i, it removes S_i's request from its request queue.

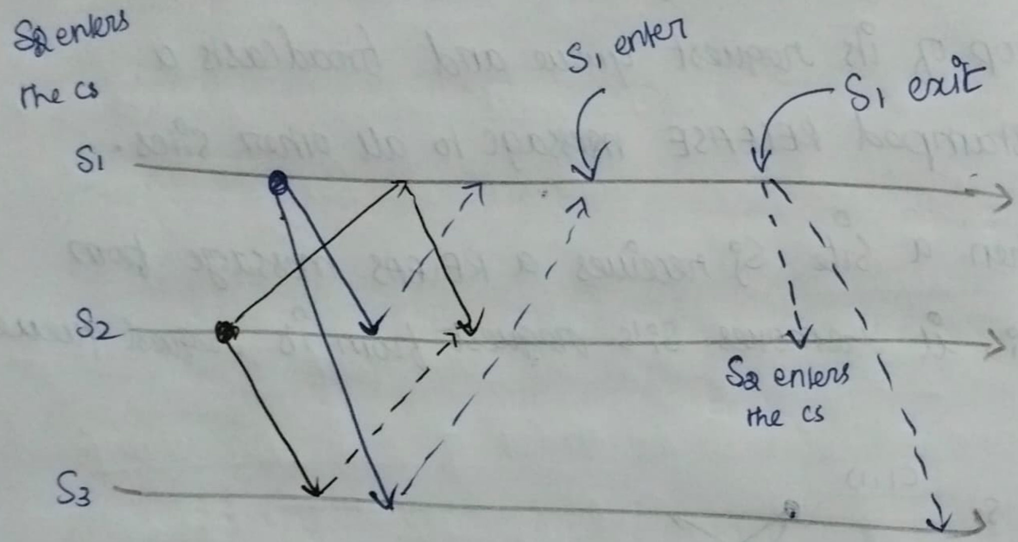
Eg:





Release.





Theorem 8 Proof:-

1. Lamports algorithm achieves mutual exclusion:

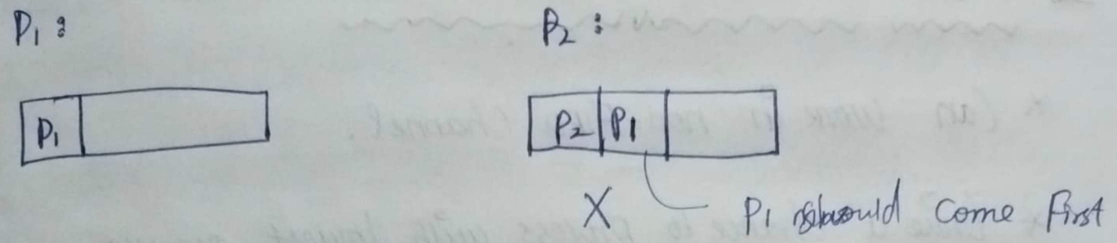
Proof by Contradiction.

① Assume P1 and P2 are in CS at a same time.

② Assume $P_1 \, req_T < P_2 \, req_T$

Request Queue.

↓



2. Lamport's algorithm is fair :
└ req executed in order of arrival

① Assume : CS(S_j)

② $S_{i_T} < S_{j_T}$

Req queue :

S_j	S_i	
8pm	7:30 pm	

L_1 → ack + time > request (S_j)

Performance :

* Request (n-1)

* Reply (n-1)

* Release (n-1)

Tt = 3(n-1)

2] RICART AGRAWALA'S ALGORITHM:

- Can work in non-FIFO channel.
- Give a chance to process with lowest request time stamp.
- Two types of MSG.

When P_1 wants to enter the cs?

1. Send out REQUEST MSG to all
2. The process P_j reply by following conditions:

   Case1: P_2 not in cs or it didn't sent REQUEST to enter cs, then send REPLY.

   Case 2: P_2 wants to enter by P_1's msg (request) time is lesser, REPLY.

   Case 3: Defer sending the reply and make $RD_2[3]=1$.

3. Exit: send all differed replies.

ALGORITHM:

Requesting the Critical Section:-

(a) When a site S_i wants to enter the CS, it broadcasts a timestamped REQUEST message to all other sites.

(b) When site S_j receives a REQUEST message from site S_i, it sends a REPLY message to site S_i if site S_j is neither requesting nor executing the CS, or if the S_j is requesting and S_i's request's timestamp is smaller than site S_j's own request's timestamp. Otherwise, the reply is deferred and S_j sets $RD_j[i] = 1$.

Executing the Critical Section :-

Site S_i enters the CS after it has received a REPLY message from every site it sent a REQUEST message to.

Releasing the Critical Section :-

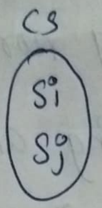
When site S_i exits the CS, it sends all the deferred ~~REPLY~~ REPLY messages : if $RD_i[j] = 1$, then sends a REPLY message to S_j and sets $RD_i[j]=0$.

Theorem & proof :

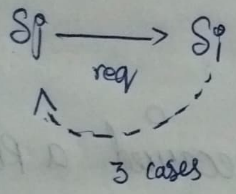
1. Algorithm achieves mutual Exclusion :-

Proof by Contradiction.

① Assume s_i and s_j in CS.



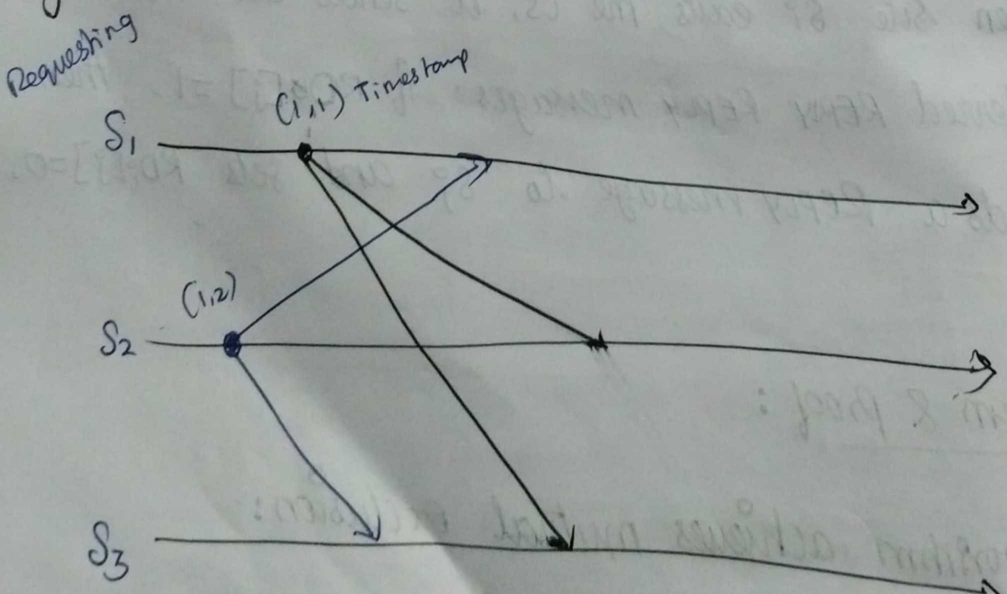
② Assume $S_{i_T} < S_{j_T}$.



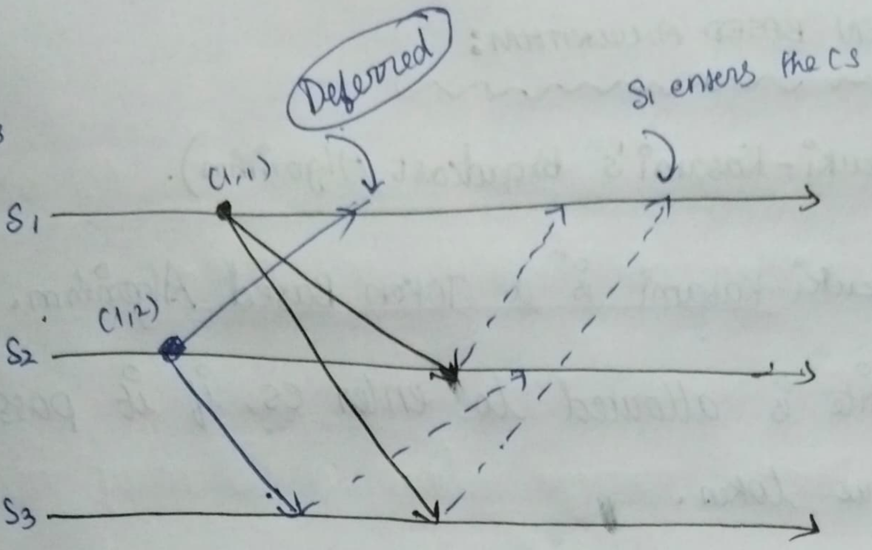
(2nd Case)

Sj not execute in CS.

Eg:



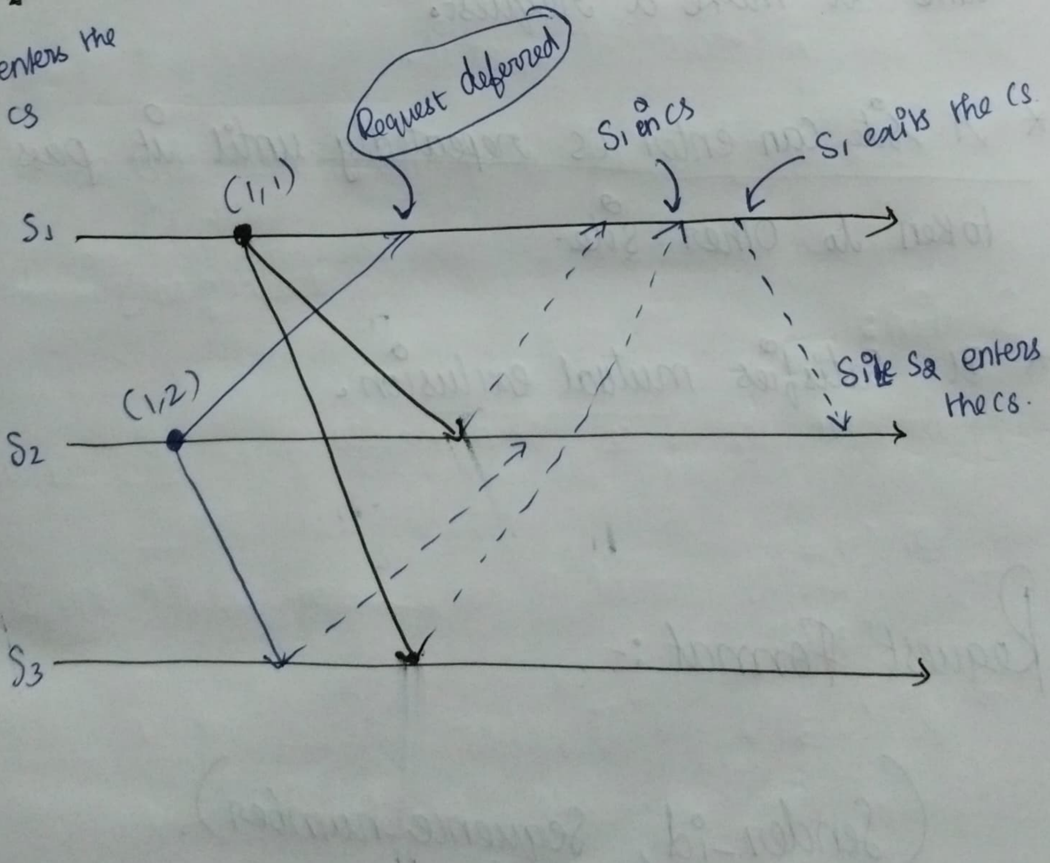
Site S1 enters the cs



S1 exit from cs and Reply to all

&

S2 enters the cs



3 TOKEN BASED ALGORITHM:

(Suzuki-Kasami's Broadcast Algorithm).

* Suzuki-kasami is a Token Based Algorithm.

* A site is allowed to enter CS, if it possesses unique token.

* Token Based algorithm uses sequence numbers.

* Request for entering critical section - Seq. no.

* Sequence number is incremented by the site every time it make a request.

* A site can enter CS repeatedly until it pass the token to other site.

* It satisfies mutual exclusion.

Request format:-

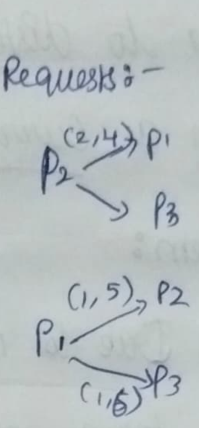
(Sender_id, sequence_number).

* Each Process maintains its own Request array.

Eg:

	P1	P2	P3
P1 :		4	
P2 :	5		
P3 :	6	4	1

Requests :-



* Holder of token maintains aditionally two:

(i) Array Last

(ii) Queue.

Eg: P3 is a holder of token:

then,

	P1	P2	P3
P3 : RN	6	4	
LAST	5	3	

→ Critical Section Completed no.

→ Based on this two array can add the requests to Queue.

Queue □ → FIFO

Next endha process ku token Send pannanumo, andha value va Store panni vaippom.

Design issues

1. How to distinguish an outdated REQUEST message from a current REQUEST message.

Problem:

* Due to message delays, a site may receive a token. request msg after the corresponding req. has been statisfied.

* If a site can not determine that it is outdated, it may dispatch the token to the requested site which does not need it.

* It may lead to performance degradation.
  ⇓
  (washing msg) more.

Solution:

Consider P_3 is a token holder.

Eg: P_1 (1,4)

P_2 (1,5)

P_1 (1,3)

at P_1 ∴ 3 < (4 already given)

3pm < 4pm

(1st)

	P1	P2	P3
P3: RN	0	0	0

(2nd)

	P1	P2	P3
RN	4	0	0

max(0,4) = 4

(3rd)

	P1	P2	P3
RN	4	5	0

max(0,5) = 5

(4th)

	P1	P2	P3
RN	4	5	0

max(4, 3) = 3
↳ outdated.
∴ (4 > 3)

Q. How to determine which site an outstanding request for CS?

↓

Endha process ku high demand irukko CS ah access panna.

Eg: P3 token:

	P1	P2	P3
P3: RN	3	5	
LN	2	2	

Queue

P1	P4	P2	P5

- FIFO

P1 → Outstanding request (cs access)

Edho top la irukko andha site ku dhaan token kudukum.

$LN[i] + 1 = RN[i]$ $\Rightarrow$ $\overset{2+1}{LN[i] + 1} = \overset{3}{RN[i]}$

ALGORITHM :

Requesting CS :-

1. If requesting site S_i does not have token, it increments seq. no $RN_i[i]$ and sends REQUEST (i, S_n) msg to all other Sites.

2. when site S_2 recieves this msg, it sets $RN_2[i]$ to max $(RN_2[i], S_n)$.

   If S_j has idle token, it sends the token to S_i if

   $$RN_2[i] = LN[i] + 1.$$

Executing CS :-

3. Site S_1 executes CS after it has received token.

Releasing CS :-

4. After finishing the execution of CS, Site S_1 does:

→ Sets $LN[i]$ element of token array equal to $RN_1[i]$.

→ For every site S_j whose id is not in token queue, it appends its id to token queue if

$$RN_i[j] = LN[j] + 1$$

-> If token queue is non-empty, si deletes the top site id from queue and sends token to it.

-> After executing CS, it gives priority to other sites with outstanding requests for CS.

4 CHANDY MISRA HAAS FOR AND MODEL :

* It is one of the best deadlock detection algorithms for distributed systems.

* It is a probe based algorithm.

Probe Message :-

Each probe msg contains the following information:

1. The id of the process that initiates the probe message.

2. The id of the process that sends the particular probe msg.

3. The id of the process that should receive the probe message.

ALGORITHM :-

1. If a process makes a request for a resource which times out, then the process generates a probe message and sends it to each process holding one or more its requested resources.

2. When a process receives a probe message, it checks to see if it is also waiting for resources. If not, it is currently using the resource and will eventually finish and release the resource.

3. If it is also waiting for resources, it passes on the probe message to all the process that holds the resources it requested.

4. The process first modifies the probe message, changing the sender and receiver ids.

5. If the initiator process receives a probe message, then there is a cycle in the system.

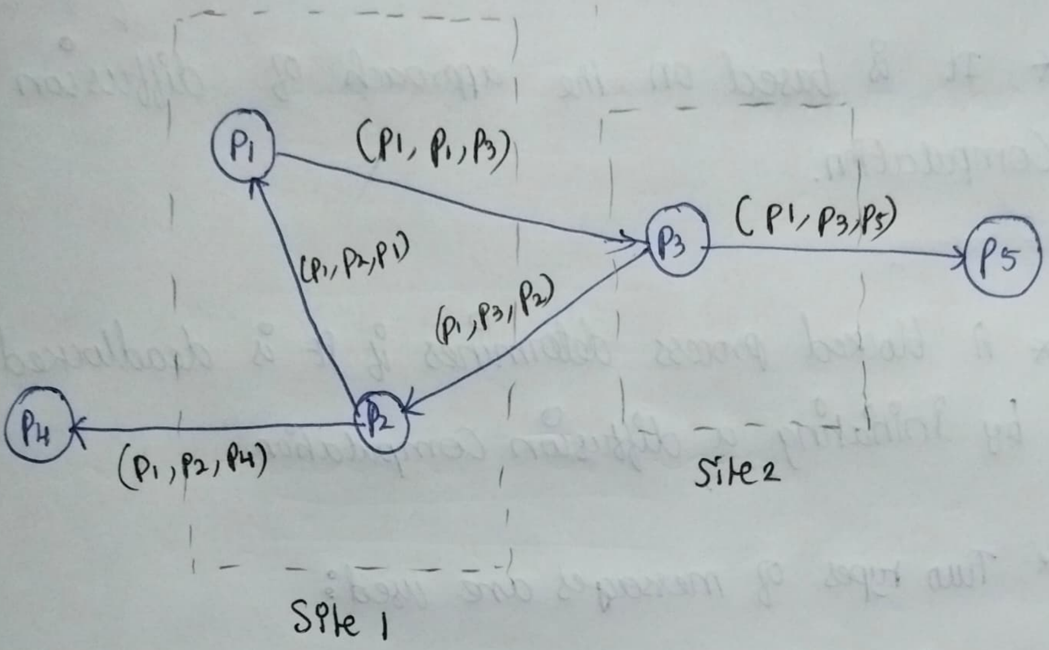
6. Thus deadlock is detected

Example :- (i, j, k)

initiator

sender

receiver



* In this case the process p1 initiates the probe message, so that all the probe message have p1 as the initiator.

* When the probe msg is received by p3, it modifies the sender and receiver ids and passes it to two more resources P_2 and P_5.

* The prob message eventually returned to the initiator process p1 by p2, thus there is a cycle in the system.

* Thus deadlock is detected.

5 CHANDY MISRA HAAS FOR OR MODEL :

Spreading

* It is based on the approach of diffusion Computation.

task executing

* A blocked process determines if it is deadlocked by initiating a diffusion computation.

* Two types of messages are used:

(i) query (i, j, k)

(ii) reply (i, j, k)

ALGORITHM :-

1. A blocked Process P_i initiates the deadlock detection by sending query message to all the Processes in its dependent set.

2. If an active process receives a query or reply, it discards it.

3. When a blocked Process P_K receives a query (i, j, k):

(i) If this is the first query message (engaging query) from P_i for deadlock detection.

-> It forwards the query to all the processes in its dependent set and sets a local variable $num_k(i)$ to the number of query messages sent.

-> When it receives a reply message for the query, it decrements $num_k(i)$.

-> It sends the reply message to the engaging query only when after it has received a reply for all the query messages it sent.

(ii) If this is not the engaging query:

-> It returns a reply message immediately to it, given P_k has been continuously blocked since it received the last query message from P_i.

4. The initiator process detects a deadlock when it receives reply message to all the query messages it sent.