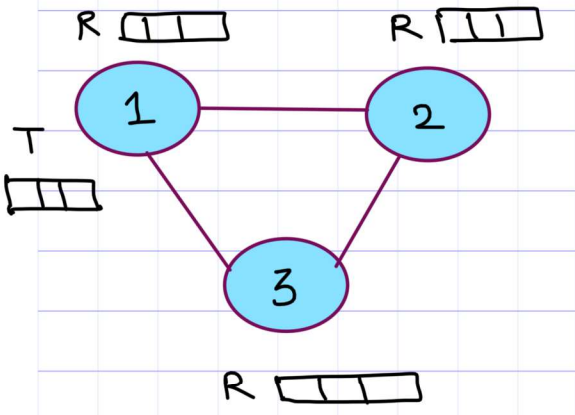
Token-Based Algorithm.

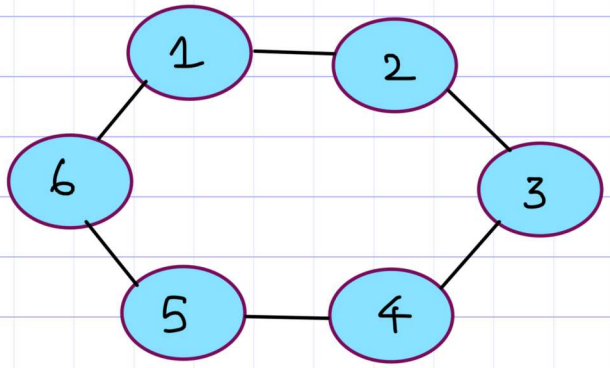


Data structures

1. request arrays @ each node. (R_i)
2. token array (only one)

Phases – two.

1. request
2. exec CS & pass token to deserving process.



naively passing token will take N-1 passes in worst case.

Suzuki-Kasami Algorithm.

Request

R [ ] n

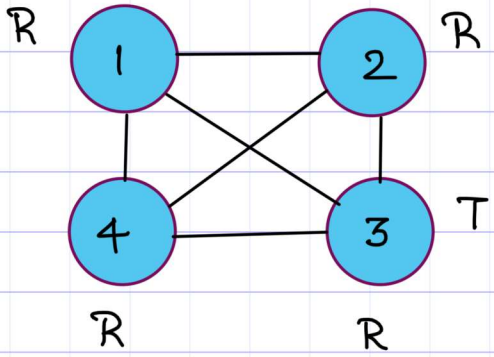
i – sender
j – receiver.

1) Increment seq. no.
2) $R_i[i]$ = seq. no.
3) Broadcast req(seq. no., Pid)

Receive

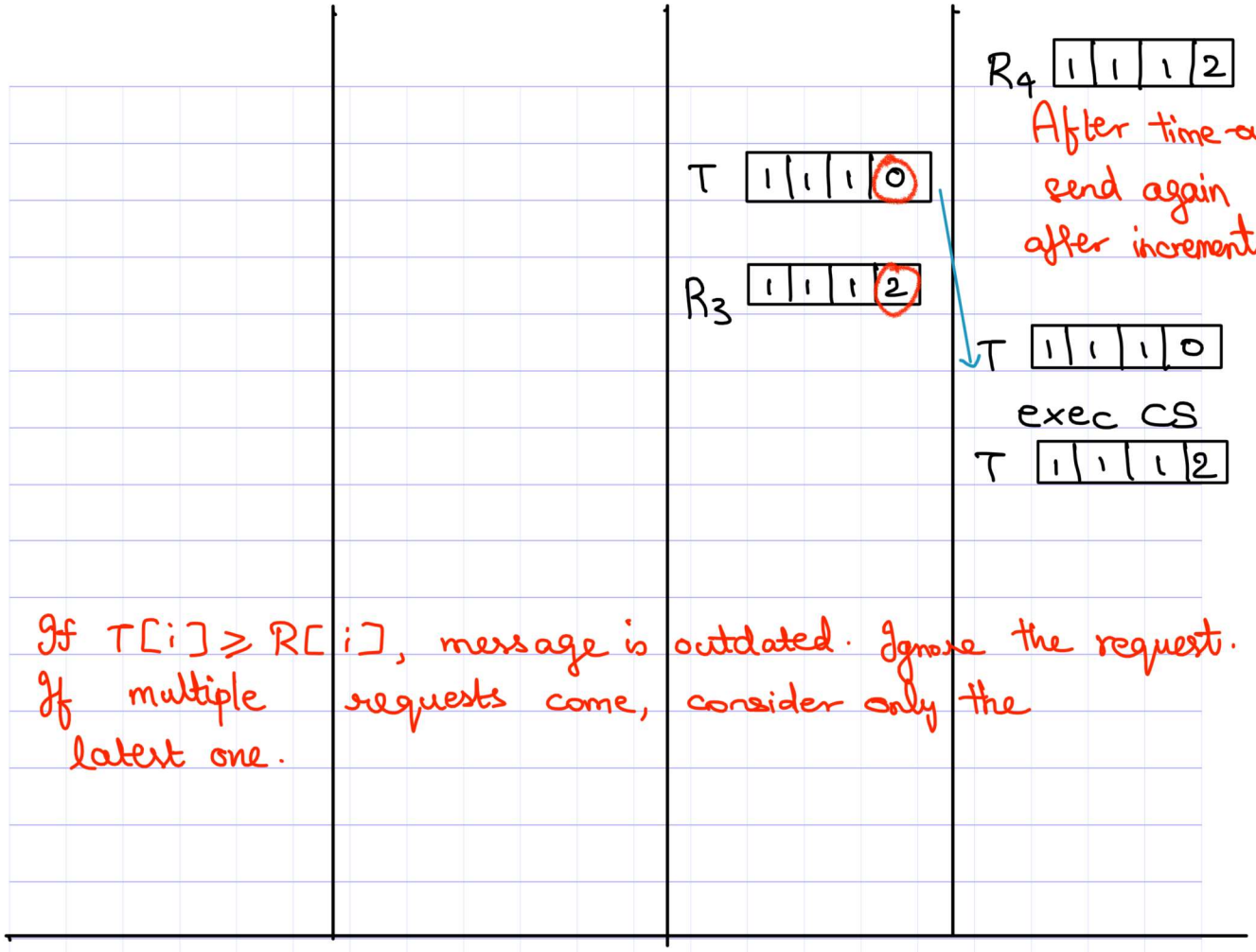
1) $R_j[i] = \max(\text{seq.no.}, R_j[i])$ — all receivers
2) If $R_j[i] - 1 == T[i]$ } only receiver w/ the token.
   then, send token to i

Suzuki - Kasami Algorithm



Req_i (seq. no., p. id.)

P_1	P_2	P_3	P_4
R_1 1 0 0 0	R_2 0 0 0 0	R_3 0 0 0 0	R_4 0 0 0 0
$Req_1(1, 1)$ (broadcasted)	rec (1, 1)	rec (1, 1)	rec (1, 1)
	R_2 1 0 0 0	R_3 1 0 0 0	R_4 1 0 0 0
		T 0 0 0 0	
T 0 0 0 0			
exec CS			
T 1 0 0 0			
	Assume P_2 & P_4	P_3 requests	for T parallely.
	R_2 1 1 0 0	R_3 1 0 1 0	
R_1 1 1 1 0			R_4 1 1 1 0
Both P_2 & P_3	are deserving.	Tie is resolved	from Left to Right
	T 1 0 0 0		
	exec CS		
	T 1 1 0 0		
		T 1 1 0 0	
		exec CS	
		T 1 1 1 0	
			R_4 1 1 1 1
			Message not delivered to others.



If $T[i] \geq R[i]$, message is outdated. Ignore the request.
If multiple requests come, consider only the latest one.

Synchronization Delay : 0 / T → round-trip delay.
↓
if the token is currently w/ the requestor.

Message Complexity : N per CS execution.
(N : no. of processes)