Hệ thống phân tán

KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

Các thuật toán đảm bảo truy cập tuần tự

- Không sử dụng token:
  - A site/process có thể vào miền găng khi một điều kiện nào đó thỏa.
  - Thuật toán đảm bảo điều kiện chỉ thỏa cho chỉ một site/process.
- Sử dụng token:
  - Một token duy nhất đc chia sẻ giữa các sites/processes.
  - Site giữ token đc phép vào miền găng.
  - Phải xử lý tình huống mất token,
    site giữ token bị tắt, khả năng nhiều token, v.v...

Mô hình hệ thống

- Giả thiết có nhiều site muốn vào miền găng, các yêu cầu đc đưa vào hàng đợi.
- Site States: Requesting CS, executing CS, idle.
- Requesting CS: blocked until granted access, cannot make additional requests for CS.
- Executing CS: using the CS.
- Idle: action is outside the site. In token-based approaches, *idle* site can have the token.

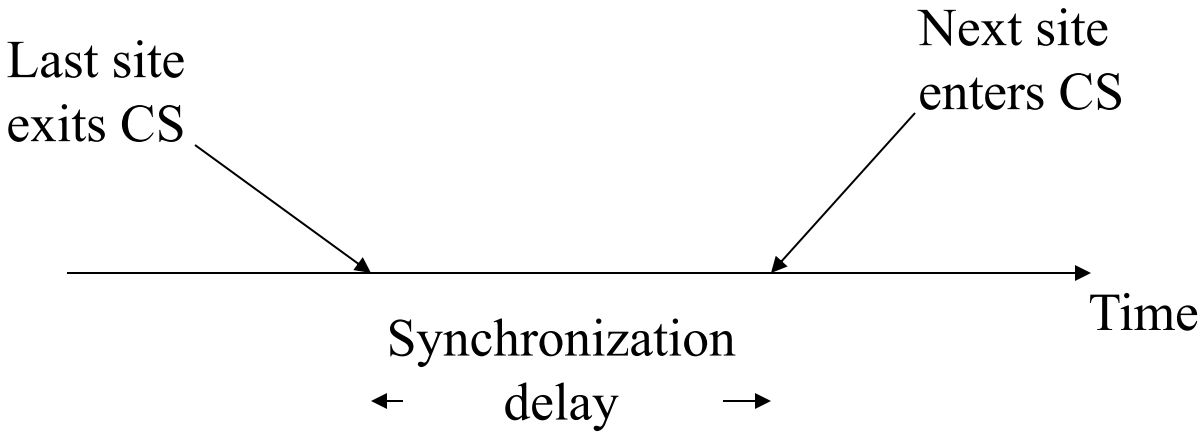
Truy cập luân phiên: Yêu cầu

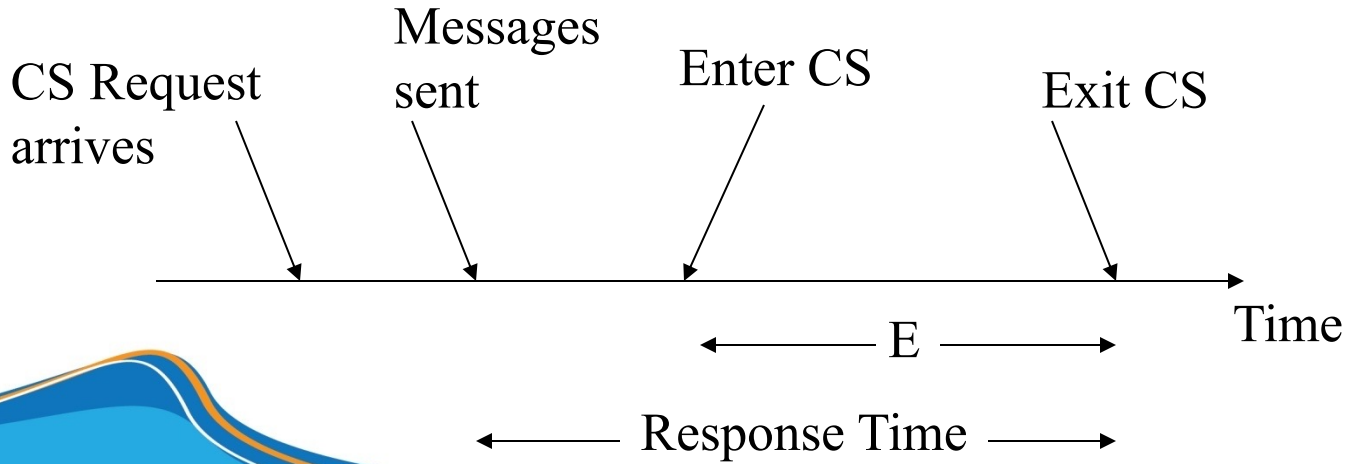
- Không bị deadlock: two or more sites should not endlessly wait on conditions/messages that never become true/arrive.
- Không bị starvation: No indefinite waiting.
- Công bằng: Order of execution of CS follows the order of the requests for CS. (equal priority).
- Chịu lỗi: recognize “faults”, reorganize, continue. (e.g., loss of token).

Hiệu suất

- Số thông điệp trao đổi để vào miền găng: should be minimized.
- Synchronization delay,: tgian từ khi một site rời miền găng và site kế tiếp vào miền găng.
- Response time: khoảng tgian từ khi gửi yêu cầu vào miền găng đến khi rời khỏi miền găng.
- System throughput, : số lượng site thực hiện xong miền găng trong một đơn vị tgian.
- If *sd* is synchronization delay, E trung bình tgian thực thi trong miền găng: system throughput = 1 / (*sd* + E).

Hiệu suất


Last site
exits CS
Next site
enters CS
Time
Synchronization
delay


CS Request
arrives
Messages
sent
Enter CS
Exit CS
Time
E
Response Time

Performance ...

- Low and High Load:
  - Low load: No more than one request at a given point in time.
  - High load: Always a pending mutual exclusion request at a site.
- Best and Worst Case:
  - Best Case (low loads): Round-trip message delay + Execution time. 2T + E.
  - Worst case (high loads).
- Message traffic: low at low loads, high at high loads.
- Average performance: when load conditions fluctuate widely.

Giải pháp đơn giản

- Control site: cấp quyền vào miền găng.
- A site sends REQUEST message to control site.
- Controller cấp quyền từng site một.
- Synchronization delay: 2T -> site rời miền găng gửi thông báo đến controller, controller gửi quyền đến site mới.
- System throughput: 1/(2T + E). If synchronization delay giảm còn T, throughput tăng gấp đôi.
- Controller bị bottleneck, có thể dẫn đến tắt nghẽn.

Thuật toán không sử dụng token

- Notations:
  - Si: Site I
  - Ri: Request set, containing the ids of all Sis from which permission must be received before accessing CS.
  - Non-token based approaches use time stamps to order requests for CS.
  - Smaller time stamps get priority over larger ones.
- Lamport's Algorithm
  - Ri = {S1, S2, …, Sn}, i.e., all sites.
  - Request queue: maintained at each Si. Ordered by time stamps.
  - Assumption: message delivered in FIFO.

Thuật toán Lamport

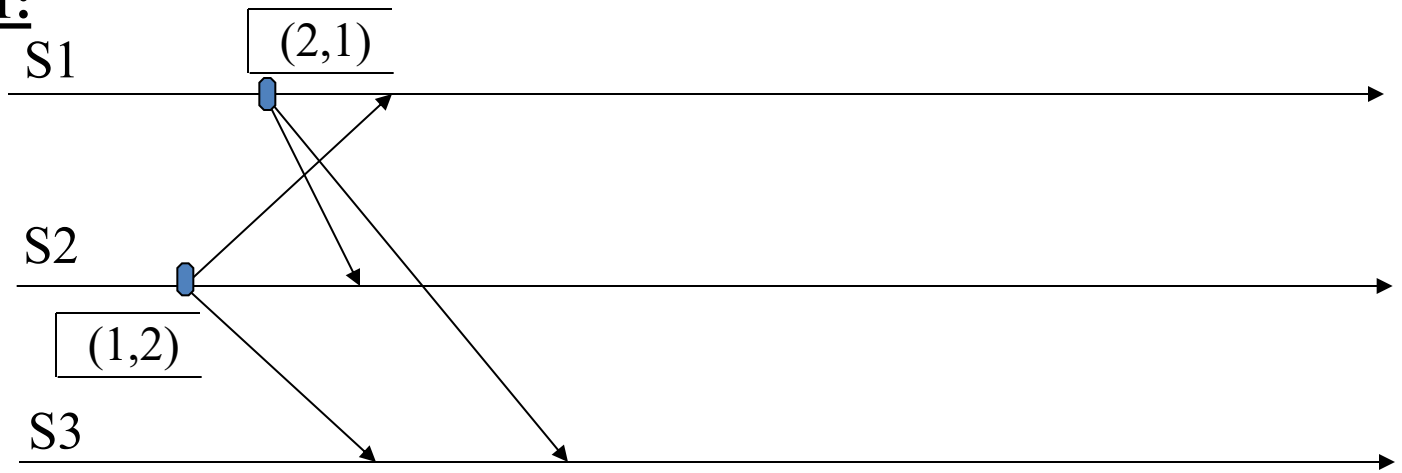
- Requesting CS:
  - Send REQUEST(tsi, i). (tsi,i): Request time stamp. Place REQUEST in *request_queuei*.
  - On receiving the message; sj sends time-stamped REPLY message to si. Si's request placed in *request_queuej.*
- Executing CS:
  - Si has received a message with time stamp larger than (tsi,i) from all other sites.
  - Si's request is the top most one in *request_queuei*.
- Releasing CS:
  - Exiting CS: send a time stamped RELEASE message to all sites in its request set.
  - Receiving RELEASE message: Sj removes Si's request from its queue.

Thuật toán Lamport

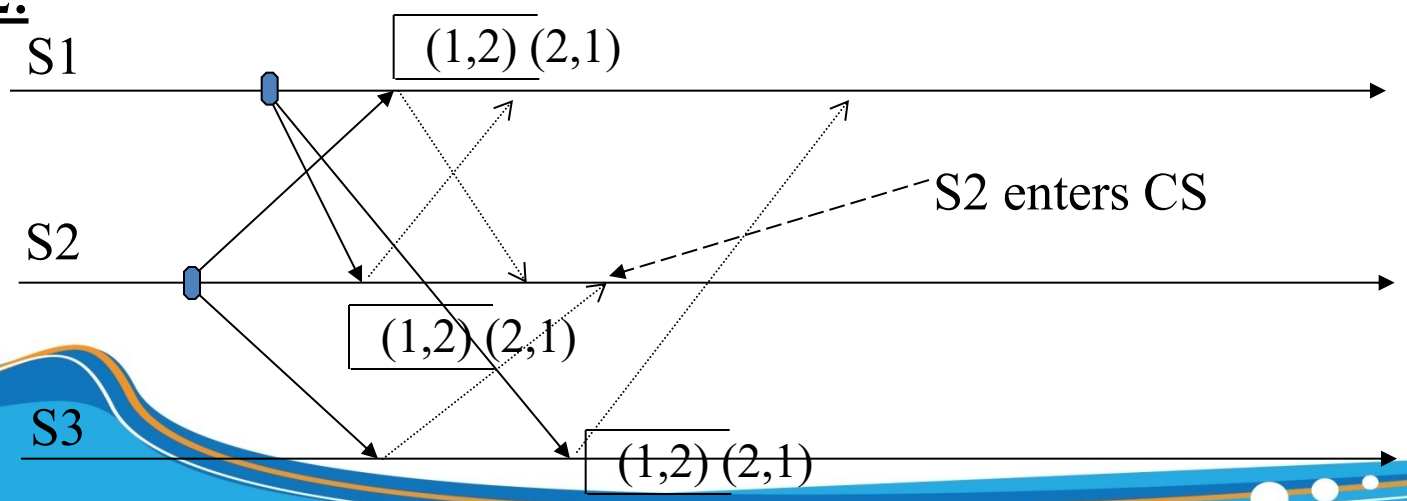
- Performance.
  - 3(N-1) messages per CS invocation. (N - 1) REQUEST, (N - 1) REPLY, (N - 1) RELEASE messages.
  - Synchronization delay: T
- Optimization
  - Giữ lại reply msg. (e.g.,) Sj nhận REQUEST tử Si sau khi nó gửi REQUEST message với time stamp lớn hơn của Si's. KHÔNG gửi REPLY message.
  - Messages giảm còn khoảng 2(N-1) tới 3(N-1).

Ví dụ thuật toán Lamport

Step 1:

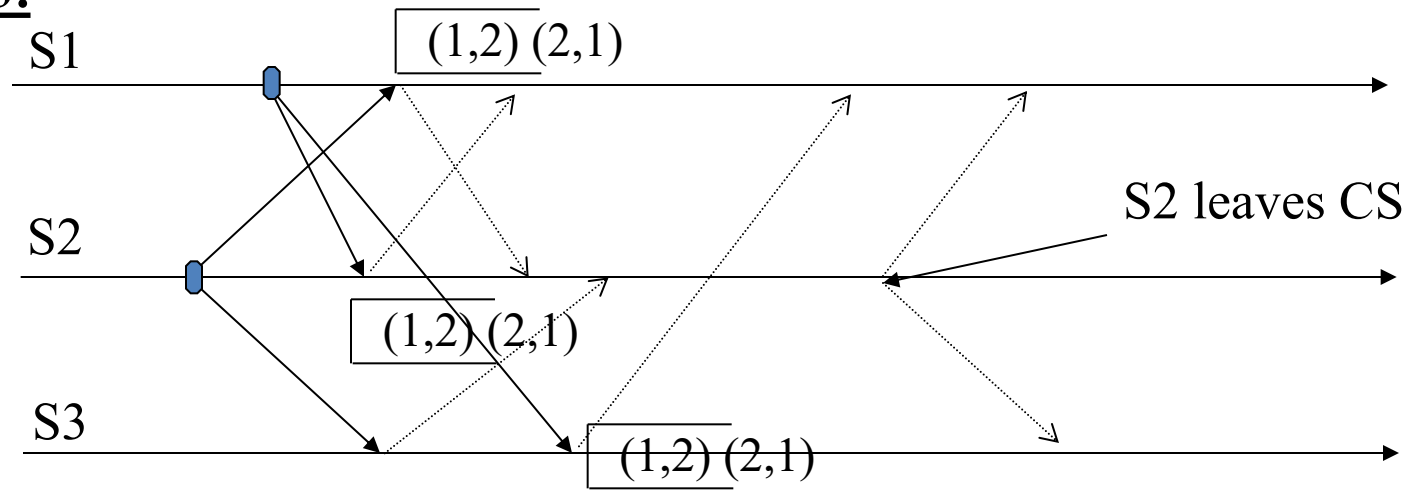


Step 2:

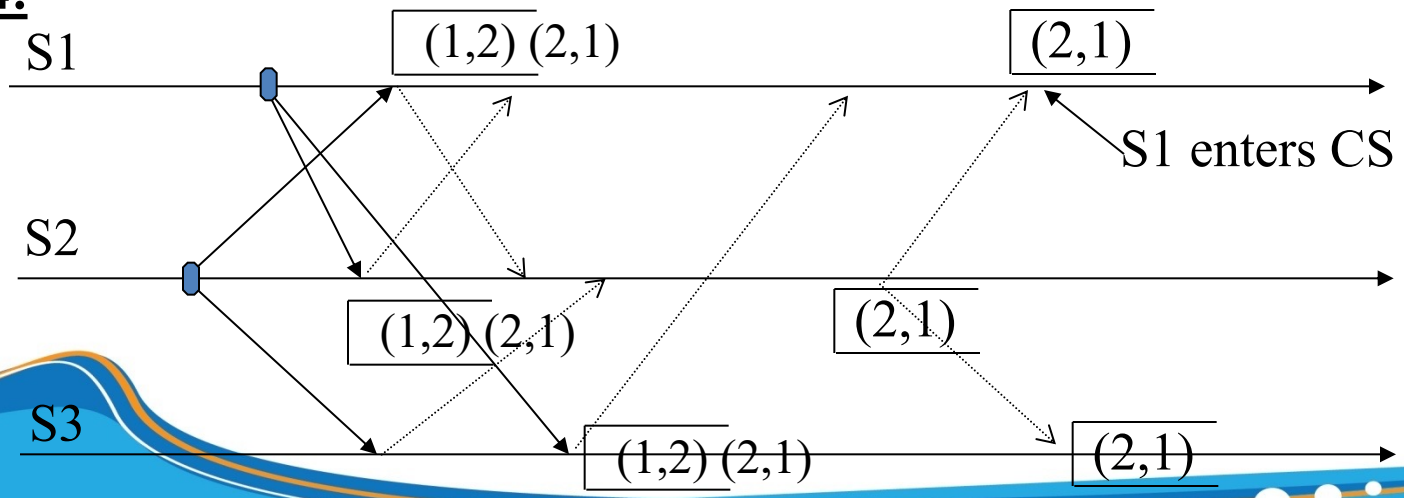


Ví dụ Lamport's:

Step 3:



Step 4:



Thuật toán Ricart-Agrawala

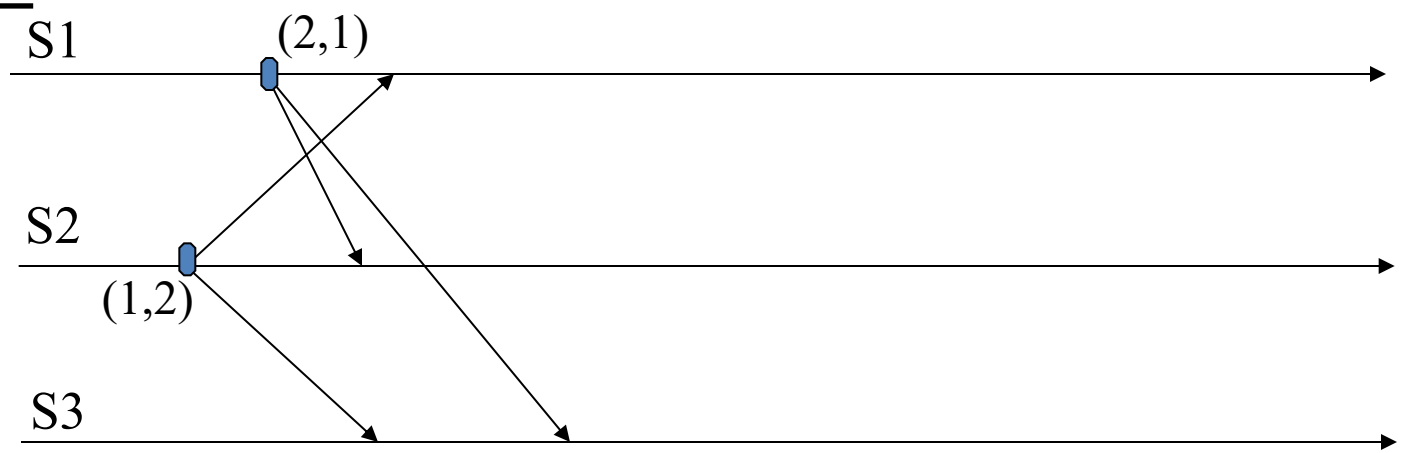
- Requesting critical section
  - Si sends time stamped REQUEST message
  - Sj sends REPLY to Si, if
    - Sj is not requesting nor executing CS
    - If Sj is requesting CS and Si's time stamp is smaller than its own request.
    - Request is deferred otherwise.
- Executing CS: after it has received REPLY from all sites in its request set.
- Releasing CS: Send REPLY to all deferred requests. i.e., a site's REPLY messages are blocked only by sites with smaller time stamps

Ricart-Agrawala: Performance

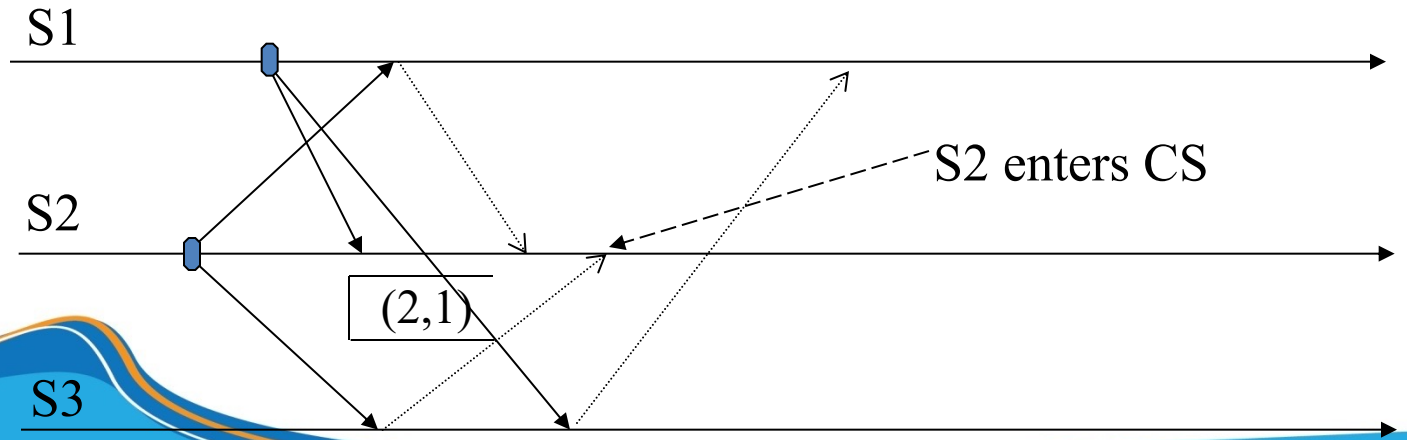
- Performance:
  - 2(N-1) messages per CS execution. (N-1) REQUEST + (N-1) REPLY.
  - Synchronization delay: T.
- Optimization:
  - Khi Si nhận REPLY message từ Sj -> có quyền vào miền găng cho đến khi
    - Sj gửi REQUEST message và Si gửi REPLY message.
    - Truy cập miền găng nhiều lần
    - A site requests permission from dynamically varying set of sites: 0 to 2(N-1) messages.

Ví dụ Ricart-Agrawala

Step 1:

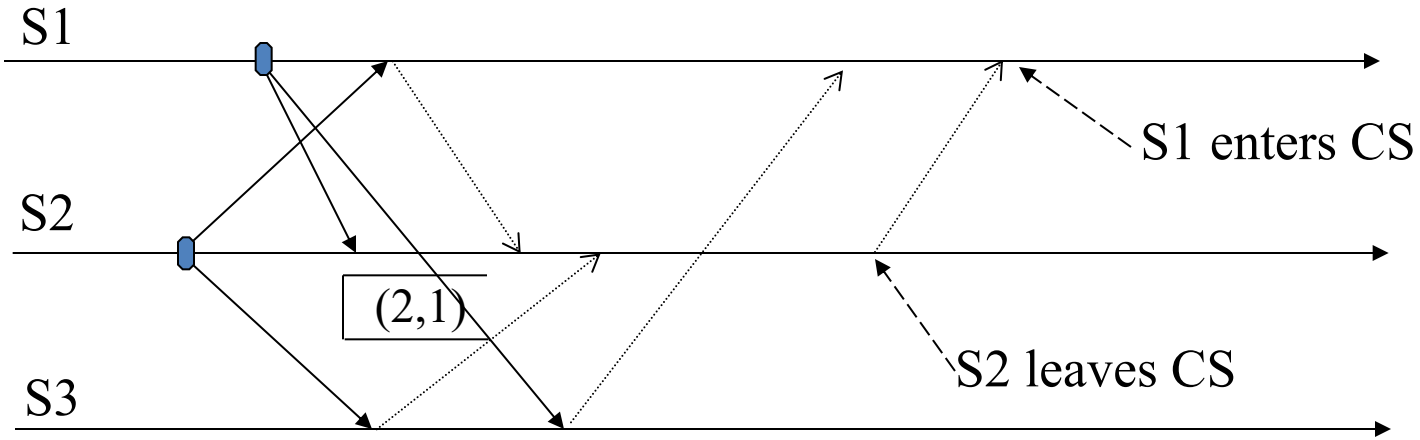


Step 2:



Ví dụ Ricart-Agrawala

Step 3:



Thuật toán Maekawa

- A site requests permission only from a subset of sites.
- Request set of sites si & sj: Ri, Rj such that Ri and Rj will have atleast one common site (Sk). Sk mediates conflicts between Ri and Rj.
- A site can send only one REPLY message at a time, i.e., a site can send a REPLY message only after receiving a RELEASE message for the previous REPLY message.
- Request Sets Rules:
  - Sets Ri and Rj have atleast one common site.
  - Si is always in Ri.
  - Cardinality of Ri, i.e., the number of sites in Ri is K.
  - Any site Si is in K number of Ris. N = K(K - 1) + 1 -> K = square root of N.

Thuật toán Maekawa

- Requesting CS
  - Si sends REQUEST(i) to sites in Ri.
  - Sj sends REPLY to Si if
    - Sj has NOT sent a REPLY message to any site after it received the last RELEASE message.
    - Otherwise, queue up Si's request.
- Executing CS: after getting REPLY from all sites in Ri.
- Releasing CS
  - send RELEASE(i) to all sites in Ri
  - Any Sj after receiving RELEASE message, send REPLY message to the next request in queue.
  - If queue empty, update status indicating receipt of RELEASE.

Các tập con

- Example k = 2; (N = 3).
  - R1 = {1, 2}; R3 = {1, 3}; R2 = {2, 3}
- Example k = 3; N = 7.
  - R1 = {1, 2, 3}; R4 = {1, 4, 5}; R6 = {1, 6, 7};
  - R2 = {2, 4, 6}; R5 = {2, 5, 7}; R7 = {3, 4, 7};
  - R3 = {3, 5, 6}
- Algorithm in Maekawa's paper (uploaded in Lecture Notes web page).

Thuật toán Maekawa

- Performance
  - Synchronization delay: 2T
  - Messages: 3 times square root of N (one each for REQUEST, REPLY, RELEASE messages)
- Deadlocks
  - Message deliveries are not ordered.
  - Assume Si, Sj, Sk concurrently request CS
  - Ri intersection Rj = {Sij}, Rj Rk = {Sjk}, Rk Ri = {Ski}
  - Possible that:
    - Sij is locked by Si (forcing Sj to wait at Sij)
    - Sjk by Sj (forcing Sk to wait at Sjk)
    - Ski by Sk (forcing Si to wait at Ski)
    - -> deadlocks among Si, Sj, and Sk.

Xử lý Deadlocks

- Si *yields* to a request if that has a smaller time stamp.
- A site suspects a deadlock when it is locked by a request with a higher time stamp (lower priority).
- Deadlock handling messages:
  - FAILED: from Si to Sj -> Si has granted permission to higher priority request.
  - INQUIRE: from Si to Sj -> Si would like to know Sj has succeeded in locking all sites in Sj's request set.
  - YIELD: from Si to Sj -> Si is returning permission to Sj so that Sj can yield to a higher priority request.

Xử lý Deadlocks

- REQUEST(tsi,i) to Sj:
  - Sj is locked by Sk -> Sj sends FAILED to Si, if Si's request has higher time stamp.
  - Otherwise, Sj sends INQUIRE(j) to Sk.
- INQUIRE(j) to Sk:
  - Sk sends a YIELD (k) to Sj, if Sk has received a FAILED message from a site in Sk's set. (or) if Sk sent a YIELD and has not received a new REPLY.
- YIELD(k) to Sj:
  - Sj assumes it has been released by Sk, places Sk's request in its queue appropriately, sends a REPLY(j) to the top request in its queue.
- Sites may exchange these messages even if there is no real deadlock. Maximum number of messages per CS request: 5 times square root of N.

Thuật toán dựa trên token

- Unique token circulates among the participating sites.
- A site can enter CS if it has the token.
- Token-based approaches use sequence numbers instead of time stamps.
  - Request for a token contains a sequence number.
  - Sequence number of sites advance independently.
- Correctness issue is trivial since only one token is present -> only one site can enter CS.
- Deadlock and starvation issues to be addressed.

Thuật toán Suzuki-Kasami

- If a site without a token needs to enter a CS, broadcast a REQUEST for token message to all other sites.
- Token: (a) Queue of request sites (b) Array LN[1..N], the sequence number of the most recent execution by a site j.
- Token holder sends token to requestor, if it is not inside CS. Otherwise, sends after exiting CS.
- Token holder can make multiple CS accesses.
- Design issues:
  - Distinguishing outdated REQUEST messages.
    - Format: REQUEST(j,n) -> jth site making nth request.
    - Each site has RNi[1..N] -> RNi[j] is the largest sequence number of request from j.
  - Determining which site has an outstanding token request.
    - If LN[j] = RNi[j] - 1, then Sj has an outstanding request.

Thuật toán Suzuki-Kasami

- Passing the token
  - After finishing CS
  - (assuming Si has token), LN[i] := RNi[i]
  - Token consists of Q and LN. Q is a queue of requesting sites.
  - Token holder checks if RNi[j] = LN[j] + 1. If so, place j in Q.
  - Send token to the site at head of Q.
- Performance
  - 0 to N messages per CS invocation.
  - Synchronization delay is 0 (if the token holder repeats CS) or T.

Ví dụ Suzuki-Kasami

Step 1: S1 has token, S3 is in queue

Site	Seq. Vector RN	Token Vect. LN	Token Queue
S1	10, 15, 9	10, 15, 8	3
S2	10, 16, 9		
S3	10, 15, 9		

Step 2: S3 gets token, S2 in queue

Site	Seq. Vector RN	Token Vect. LN	Token Queue
S1	10, 16, 9		
S2	10, 16, 9		
S3	10, 16, 9	10, 15, 9	2

Step 3: S2 gets token, queue empty

Site	Seq. Vector RN	Token Vect. LN	Token Queue
S1	10, 16, 9		
S2	10, 16, 9	10, 16, 9	<empty>
S3	10, 16, 9		

Thuật toán Singhal's Heuristic

- Instead of broadcast: each site maintains information on other sites, guess the sites likely to have the token.
- Data Structures:
  - Si maintains SVi[1..M] and SNi[1..M] for storing information on other sites: state and highest sequence number.
  - Token contains 2 arrays: TSV[1..M] and TSN[1..M].
  - States of a site
    - R : requesting CS
    - E : executing CS
    - H : Holding token, idle
    - N : None of the above
  - Initialization:
    - SVi[j] := N, for j = M .. i; SVi[j] := R, for j = i-1 .. 1; SNi[j] := 0, j = 1..N. S1 (Site 1) is in state H.
    - Token: TSV[j] := N & TSN[j] := 0, j = 1 .. N.

Thuật toán Singhal's Heuristic …

- Requesting CS
  - If Si has no token and requests CS:
    - SVi[i] := R. SNi[i] := SNi[i] + 1.
    - Send REQUEST(i,sn) to sites Sj for which SVi[j] = R. (sn: sequence number, updated value of SNi[i]).
  - Receiving REQUEST(i,sn): if sn <= SNj[i], ignore. Otherwise, update SNj[i] and do:
    - SVj[j] = N -> SVj[i] := R.
    - SVj[j] = R -> If SVj[i] != R, set it to R & send REQUEST(j,SNj[j]) to Si. Else do nothing.
    - SVj[j] = E -> SVj[i] := R.
    - SVj[j] = H -> SVj[i] := R, TSV[i] := R, TSN[i] := sn, SVj[j] = N. Send token to Si.
- Executing CS: after getting token. Set SVi[i] := E.

Singhal's Heuristic …

- Releasing CS
  - SVi[i] := N, TSV[i] := N. Then, do:
    - For other Sj: if (SNi[j] > TSN[j]), then {TSV[j] := SVi[j]; TSN[j] := SNi[j]}
    - else {SVi[j] := TSV[j]; SNi[j] := TSN[j]}
  - If SVi[j] = N, for all j, then set SVi[i] := H. Else send token to a site Sj provided SVi[j] = R.
- Fairness of algorithm will depend on choice of Si, since no queue is maintained in token.
- Arbitration rules to ensure fairness used.
- Performance
  - Low to moderate loads: average of N/2 messages.
  - High loads: N messages (all sites request CS).

  Synchronization delay: T.

Ví dụ Singhal

	1	2	3	4	n
Sn	R	R	R	R	N
S4	R	R	R	N	N
S3	R	R	N	N	N
S2	R	N	N	N	N
S1	H	N	N	N	N

(a) Initial Pattern

Each row in the matrix has increasing number of Rs.

	1	2	3	4	n
Sn	R	R	R	R	N
S4	R	R	R	N	N
S2	R	N	R	N	N
S1	N	N	R	N	N
S3	N	N	H	N	N

(b) Pattern after S3 gets the token from S1.

Stair case is pattern can be identified by noting that S1 has 1 R and S2 has 2 Rs and so on. Order of occurrence of R in a row does not matter.

Ví dụ Singhal

- Assume there are 3 sites in the system. Initially:
  Site 1: SV1[1] = H, SV1[2] = N, SV1[3] = N. SN1[1], SN1[2], SN1[3] are 0.
  Site 2: SV2[1] = R, SV2[2] = N, SV2[3] = N. SNs are 0.
  Site 3: SV3[1] = R, SV3[2] = R, SV3[3] = N. SNs are 0.
  Token: TSVs are N. TSNs are 0.
- Assume site 2 is requesting token.
  S2 sets SV2[2] = R, SN2[2] = 1.
  S2 sends REQUEST(2,1) to S1 (since only S1 is set to R in SV[2])
- S1 receives the REQUEST. Accepts the REQUEST since SN1[2] is smaller than the message sequence number.
  Since SV1[1] is H: SV1[2] = R, TSV[2] = R, TSN[2] = 1, SV1[1] = N.
  Send token to S2
- S2 receives the token. SV2[2] = E. After exiting the CS, SV2[2] = TSV[2] = N. Updates SN, SV, TSN, TSV. Since nobody is REQUESTing, SV2[2] = H.
- Assume S3 makes a REQUEST now. It will be sent to both S1 and S2. Only S2 responds since only SV2[2] is H (SV1[1] is N now).

Thuật toán Raymond

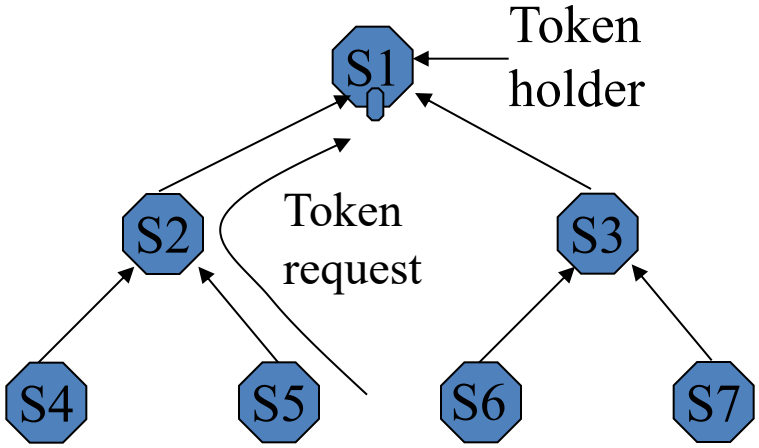
- Sites are arranged in a logical directed tree. Root: token holder. Edges: directed towards root.
- Every site has a variable *holder* that points to an immediate neighbor node, on the directed path towards root. (Root's holder point to itself).
- Requesting CS
  - If Si does not hold token and request CS, sends REQUEST *upwards* provided its *request_q* is empty. It then adds its request to *request_q*.
  - Site on path to root receiving REQUEST -> propagate it up, if its *request_q* is empty. Add request to *request_q*.
  - Root on receiving REQUEST -> send token to the site that forwarded the message. Set *holder* to that forwarding site.
  - Any Si receiving token -> delete top entry from *request_q*, send token to that site, set *holder* to point to it. If request_q is non-empty now, send REQUEST message to the *holder* site.

Thuật toán Raymond…

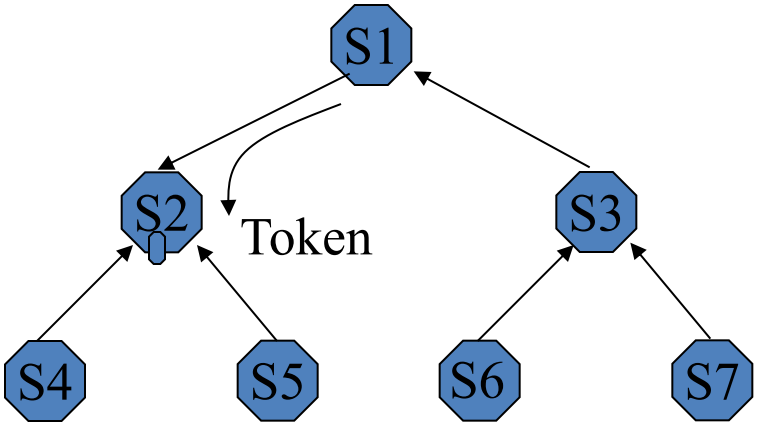
- Executing CS: getting token with the site at the top of *request_q*. Delete top of *request_q*, enter CS.
- Releasing CS
  - If *request_q* is non-empty, delete top entry from q, send token to that site, set *holder* to that site.
  - If *request_q* is non-empty now, send REQUEST message to the *holder* site.
- Performance
  - Average messages: O(log N) as average distance between 2 nodes in the tree is O(log N).
  - Synchronization delay: (T log N) / 2, as average distance between 2 sites to successively execute CS is (log N) / 2.
  - Greedy approach: Intermediate site getting the token may enter CS instead of forwarding it down. Affects fairness, may cause starvation.

Ví dụ Raymond

Step 1:

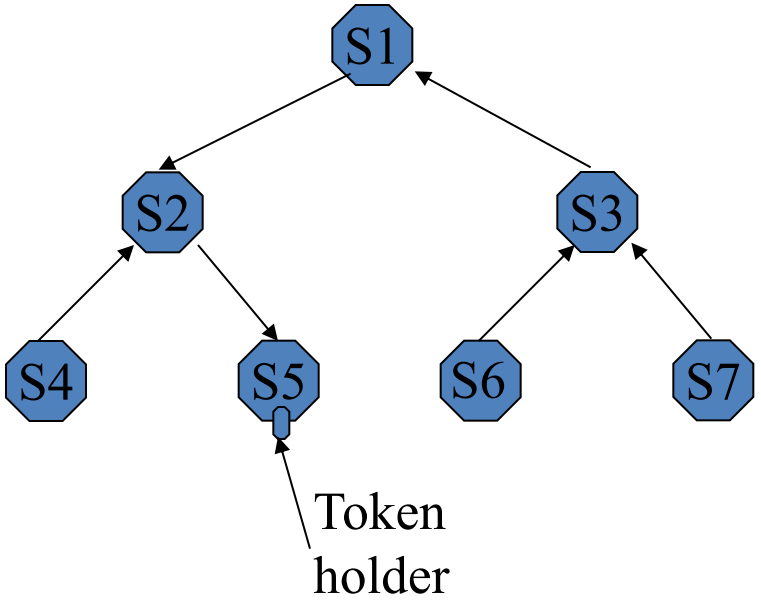


Step 2:



Ví dụ Raymond…

Step 3:



So sánh

Non-Token	**Resp. Time(ll)**	**Sync. Delay**	**Messages(ll)**	**Messages(hl)**
Lamport	2T+E	T	3(N-1)	3(N-1)
Ricart-Agrawala	2T+E	T	2(N-1)	2(N-1)
Maekawa	2T+E	2T	3*sq.rt(N)	5*sq.rt(N)
Token	**Resp. Time(ll)**	**Sync. Delay**	**Messages(ll)**	**Messages(hl)**
Suzuki-Kasami	2T+E	T	N	N
Singhal	2T+E	T	N/2	N
Raymond	T(log N)+E	Tlog(N)/2	log(N)	4