# Distributed mutual exclusion

CS 451

August 13, 2003

Prof. R.K. Joshi
Dept of CSE
IIT Bombay

# Machine Characteristics

- N nodes
- Completely connected
- Message passing
- Each processor has concurrent activities

# Fully distributed algorithm?

- How do they reach a uniform decision on mutual exclusion?
- Each of them may want to enter critical section at any point of time

# Attempt I

- Initially machine 0 ....

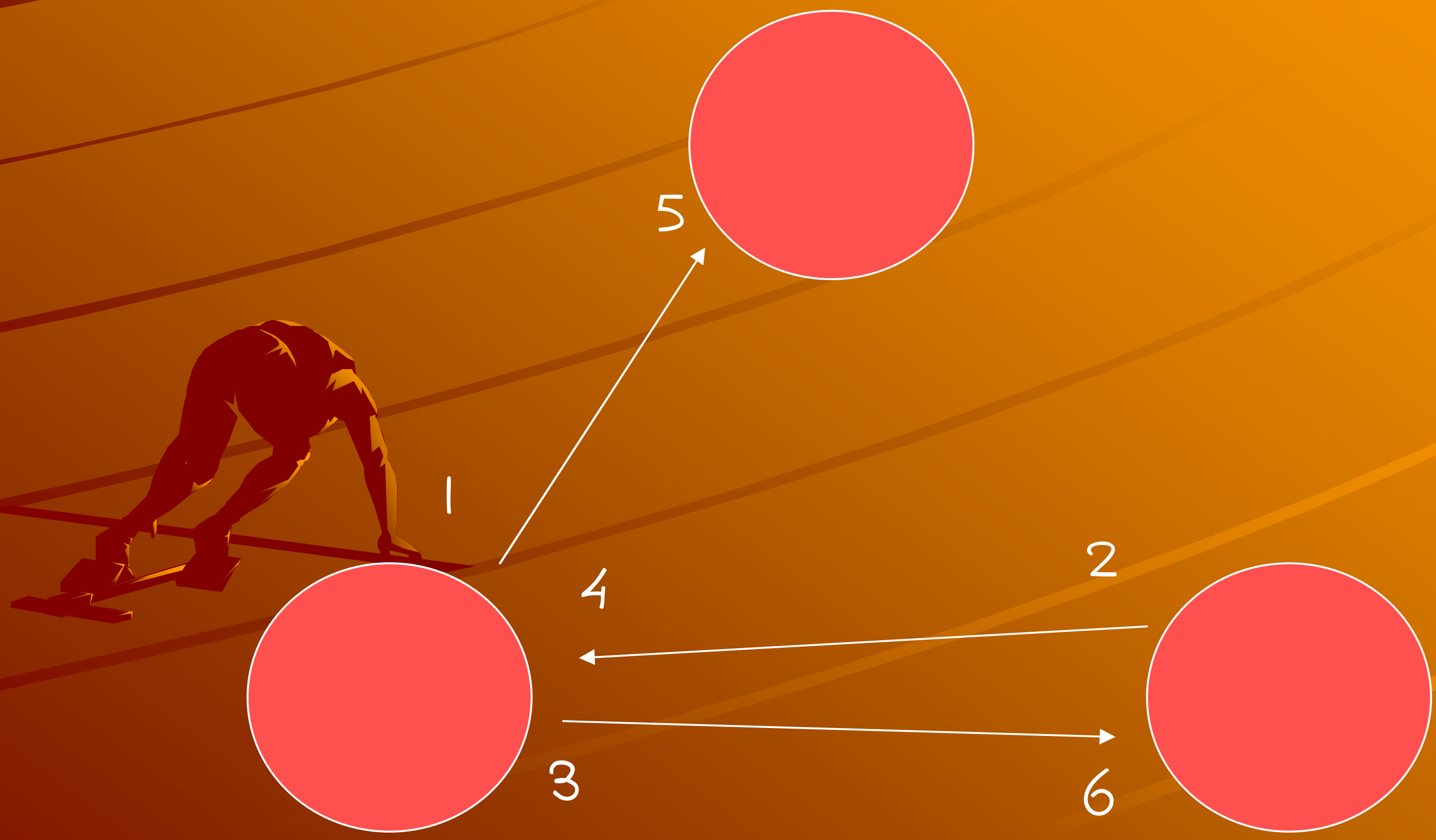
  Non progressive !

- Okay, if machinbe 0 is not interested, machine 1 ..

  How do you find out who is not interested?!

Evolve a symmetric protocol

# Try 3 processes



# Attempt 3

ECS code

1. Send REQ to all
2. Wait for OK from all
3. CS
4. Send OK to all processors in pending requests queue

Thread that receives REQ

1. Send OK to sender if you haven't sent a REQ
2. Else send OK if sender id < your id
3. Else enqueue sender into pending requests

# Attempt 3a

Init: s0

**ECS code**

1\. s1

1\. Send REQ to all s2

2\. Wait for GRANT from all

3\. s3

4\. CS

5\. Send GRANT to all

6\. s0

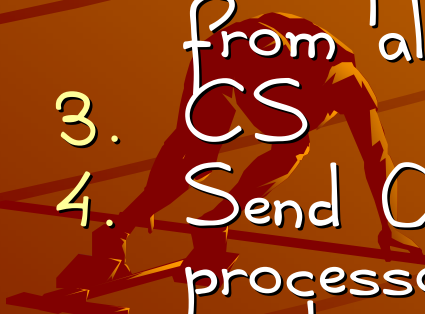
**Thread that receives REQ**

1. If (s3) then nothing else if (s1) or (s2)
    1. If (senderid < myid) Send GRANT else nothing
2. Else if (s0) send GRANT

# Attempt 4

## ECS code

1. Send REQ to all
2. Wait for OK from all
3. CS
4. Send OK to all processors in pending requests queue



## Thread that receives REQ

1. Send OK to sender if you haven't sent a REQ
2. Else send OK if sender id < your id AND OK from sender is not received till this point
3. Else enqueue sender into pending requests

# A simple algorithm

**ECS code**

1. Send REQ to all
2. Wait for GRANT from all
3. If REJECT is recd., abort
4. CS
5. Nullify your REQ

**Thread that receives REQ**

1. Send GRANT if you have not sent a REQ
2. Else send REJECT

Starvation!
Livelock!

*progressive*
*correct*
*deadlock-free*

Switch to Lecture on Logical Clocks and then continue with the next slide

# Using Logical clocks (TSs)

ECS code

1. Send REQ to all with TSi
2. Wait for GRANT from all
3. If REJECT is recd., abort
4. CS
5. Nullify your REQ

Thread that receives REQ

1. Send GRANT if you have not sent a REQ
2. Else send REJECT

# Lamport's Algorithm

Requesting CS

1. Send REQ (TSi, i) to all
2. Place the request on its own request queue

Enter CS when

1. A message with timestamp larger that (TSi, i) is received from all other sites
2. Pi's request is on top of the request queue

Request queue is ordered by timestamps

Thread that receives REQ

1. Send a time stamped REPLY
2. Place the request on its own request queue

On CS exit

1. Send DONE message
2. Remove your request from your request queue

On receiving DONE

1. Remove that request from your request queue

# Ricart and Agrawala

Requesting CS

1. Send a timestamped REQUEST to all sites
2. Enter CS after REPLY from all sites is received

Releasing CS

1. Send REPLY to all deferred requests

On receiving REQUEST from $S_j$

1. Send REPLY if $S_i$ has not made a REQUEST else
2. If REQUEST is made by $S_i$ and $TS(R_i) > TS(R_j)$ send REPLY
3. Else reply is deferred

# Performance

- Lamport's Algorithm
  - 3(N-1) messages per CS invocation
    - N-1 requests
    - N-1 replies
    - N-1 releases
- Ricart and Agrawala
  - 2(N-1) messages per CS
    - N-1 requests
    - N-1 replies

# Readings

- Lamport: Time, clocks and ordering of events in distributed systems, CACM July 1978
- Ricart and Agrawala, An optimal algorithm for mutual exclusion in computer networks, CACM Jan 1981