CLOUD COMPUTING CONCEPTS

with Indranil Gupta (Indy)

MUTUAL EXCLUSION

Lecture B

DISTRIBUTED MUTUAL EXCLUSION

System Model

- Before solving any problem, specify its System Model:
  - Each pair of processes is connected by reliable channels (such as TCP).
  - Messages are eventually delivered to recipient, and in FIFO (First In First Out) order.
  - Processes do not fail.
    - Fault-tolerant variants exist in literature.

Central Solution

- Elect a central master (or leader)
  - Use one of our election algorithms!
- Master keeps
  - A **queue** of waiting requests from processes who wish to access the CS
  - A special **token** which allows its holder to access CS
- Actions of any process in group:
  - enter()
    - Send a request to master
    - Wait for token from master
  - exit()
    - Send back token to master

Central Solution

- Master Actions:
  - On receiving a request from process P*i*

    if (master has token)

        Send token to P*i*

    else

        Add P*i* to queue
  - On receiving a token from process P*i*

    if (queue is not empty)

        Dequeue head of queue (say P*j*), send that process the token

    else

        Retain token

Analysis of Central Algorithm

- Safety – at most one process in CS
  - Exactly one token
- Liveness – every request for CS granted eventually
  - With N processes in system, queue has at most N processes
  - If each process exits CS eventually and no failures, liveness guaranteed
- FIFO Ordering is guaranteed, in order of requests received at master

Analyzing Performance

Efficient mutual exclusion algorithms use fewer messages, and make processes wait for shorter durations to access resources. Three metrics:

- ***Bandwidth***: the total number of messages sent in each *enter* and *exit* operation.
- ***Client delay***: the delay incurred by a process at each enter and exit operation (when *no* other process is in, or waiting)

  (We will prefer mostly the enter operation.)
- ***Synchronization delay***: the time interval between one process exiting the critical section and the next process entering it (when there is *only one* process waiting)

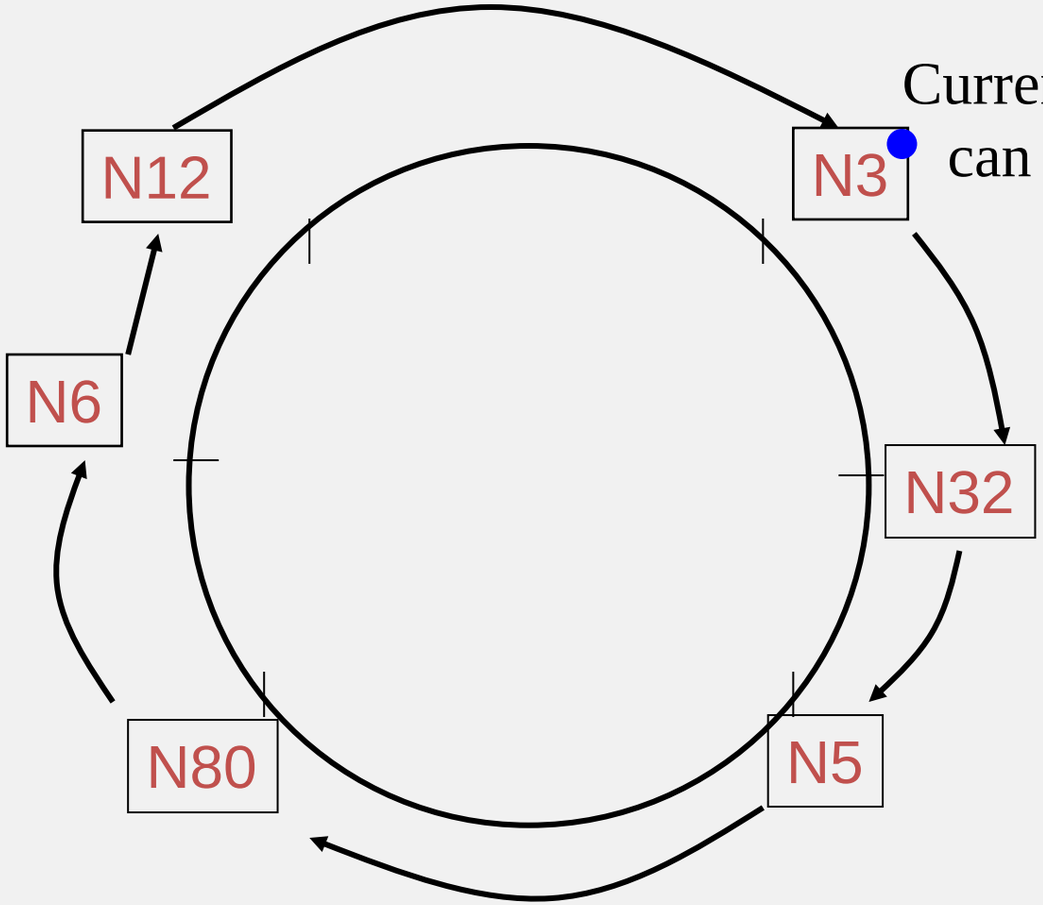
Analysis of Central Algorithm

- ***Bandwidth***: the total number of messages sent in each *enter* and *exit* operation.
  - 2 messages for enter
  - 1 message for exit
- ***Client delay***: the delay incurred by a process at each enter and exit operation (when *no* other process is in, or waiting)
  - 2 message latencies (request + grant)
- ***Synchronization delay***: the time interval between one process exiting the critical section and the next process entering it (when there is *only one* process waiting)
  - 2 message latencies (release + grant)

But…

- The master is the performance bottleneck and SPoF (single point of failure)

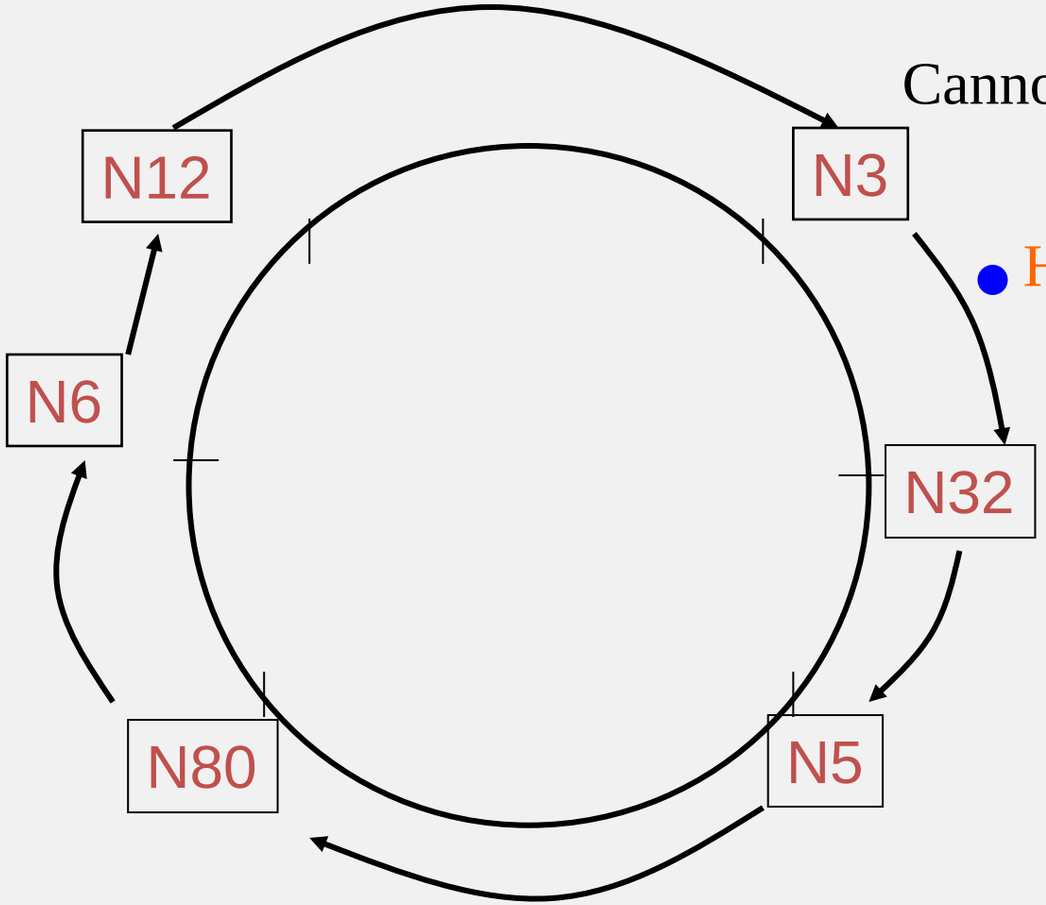
Ring-based Mutual Exclusion



Currently holds token, can access CS

Token: ●

Ring-based Mutual Exclusion

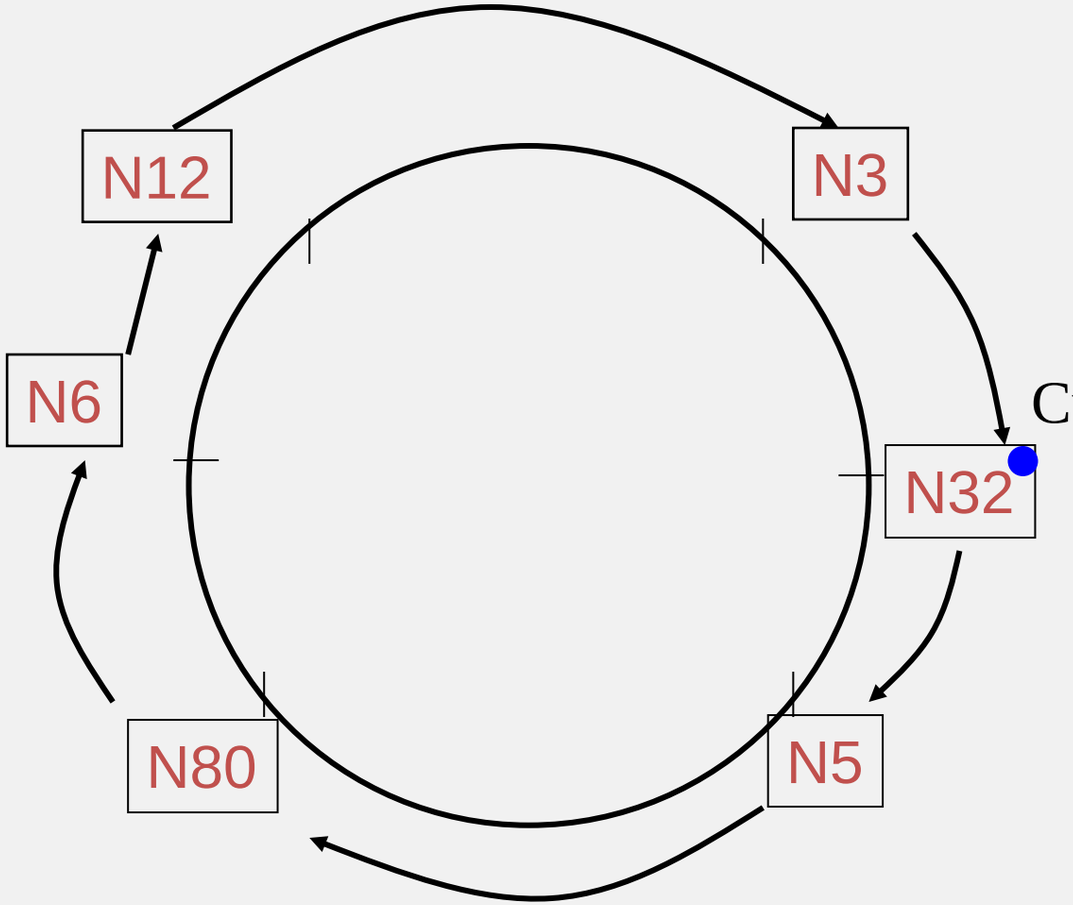


Cannot access CS anymore

● Here's the token!

Token: ●

Ring-based Mutual Exclusion



Currently holds token,
can access CS

Token: ●

Ring-based Mutual Exclusion

- *N* Processes organized in a virtual ring
- Each process can send message to its successor in ring
- Exactly 1 token
- enter()
  - Wait until you get token
- exit() // already have token
  - Pass on token to ring successor
- If receive token, and not currently in enter(), just pass on token to ring successor

Analysis of Ring-based Mutual Exclusion

- Safety
  - Exactly one token
- Liveness
  - Token eventually loops around ring and reaches requesting process (no failures)
- Bandwidth
  - Per enter(), 1 message by requesting process but N messages throughout system
  - 1 message sent per exit()

Analysis of Ring-Based Mutual Exclusion (2)

- Client delay: 0 to *N* message transmissions after entering enter()
  - Best case: already have token
  - Worst case: just sent token to neighbor
- Synchronization delay between one process' exit() from the CS and the next process' enter():
  - Between 1 and (*N-1*) message transmissions.
  - Best case: process in enter() is successor of process in exit()
  - Worst case: process in enter() is predecessor of process in exit()

Next

- Client/Synchronization delay to access CS still O(N) in Ring-Based approach.
- Can we lower this?