CLOUD COMPUTING CONCEPTS

with Indranil Gupta (Indy)

MUTUAL EXCLUSION

Lecture C

RICART-AGRAWALA'S ALGORITHM

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.

Ricart-Agrawala's Algorithm

- Classical algorithm from 1981
- Invented by Glenn Ricart (NIH) and Ashok Agrawala (U. Maryland)

- No token
- Uses the notion of causality and multicast
- Has lower waiting time to enter CS than Ring-Based approach

Key Idea: Ricart-Agrawala Algorithm

- enter() at process P*i*
    - multicast a request to all processes
        - Request: <T, P*i*>, where T = current Lamport timestamp at P*i*
    - Wait until *all* other processes have responded positively to request
- Requests are granted in order of causality
- P*i* in request <T, P*i*> is used to break ties (since Lamport timestamps are not unique for concurrent events)

Messages in RA Algorithm

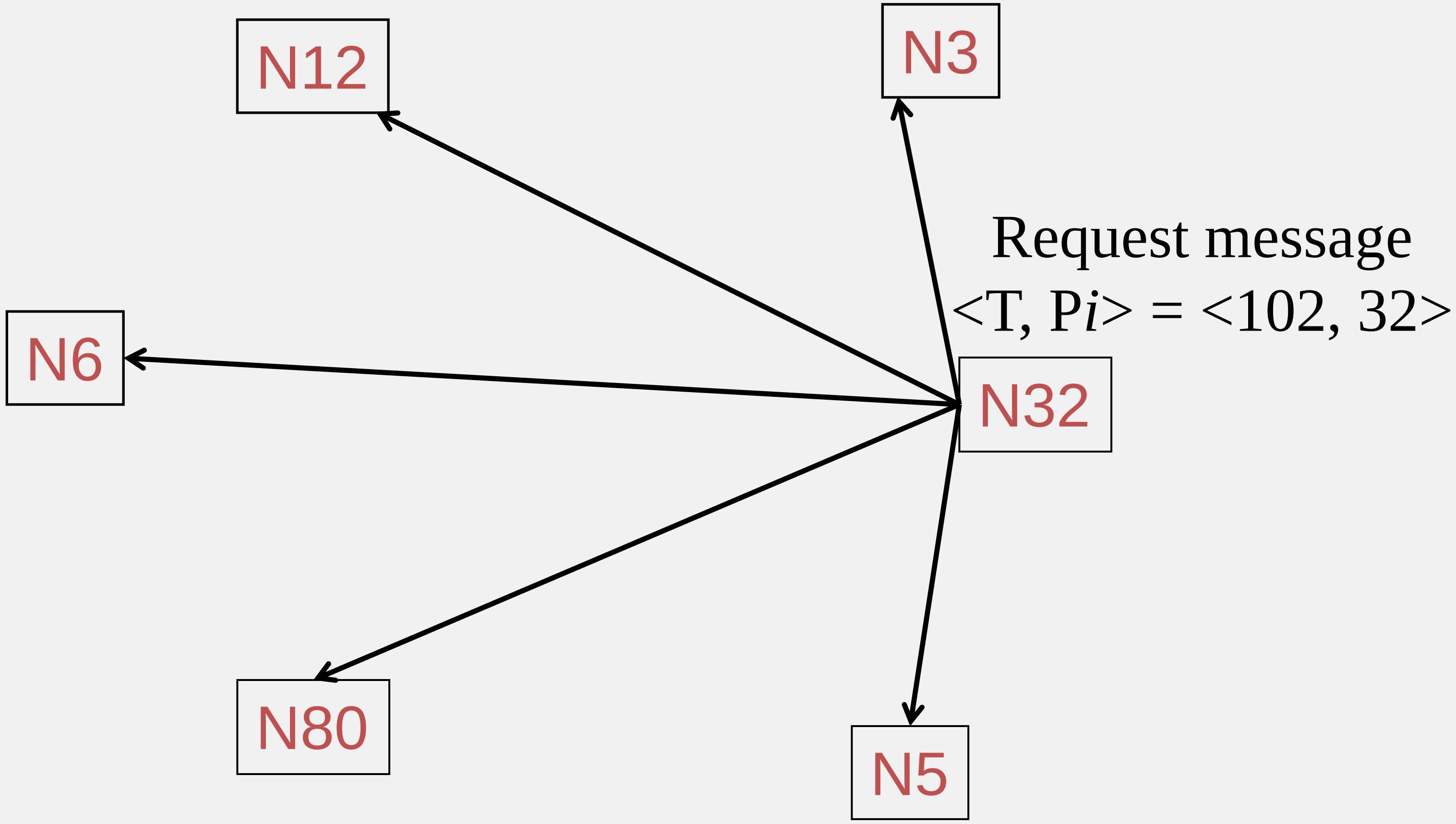
- enter() at process Pi
  - set state to Wanted
  - multicast "Request" <Ti, Pi> to all processes, where Ti = current Lamport timestamp at Pi
  - wait until ***all*** processes send back "Reply"
  - change state to Held and enter the CS
- On receipt of a Request <Tj, Pj> at Pi ($i \neq j$):
  - **if** (state = Held) or (state = Wanted & (Ti, i) < (Tj, j))

    // lexicographic ordering in (Tj, Pj)

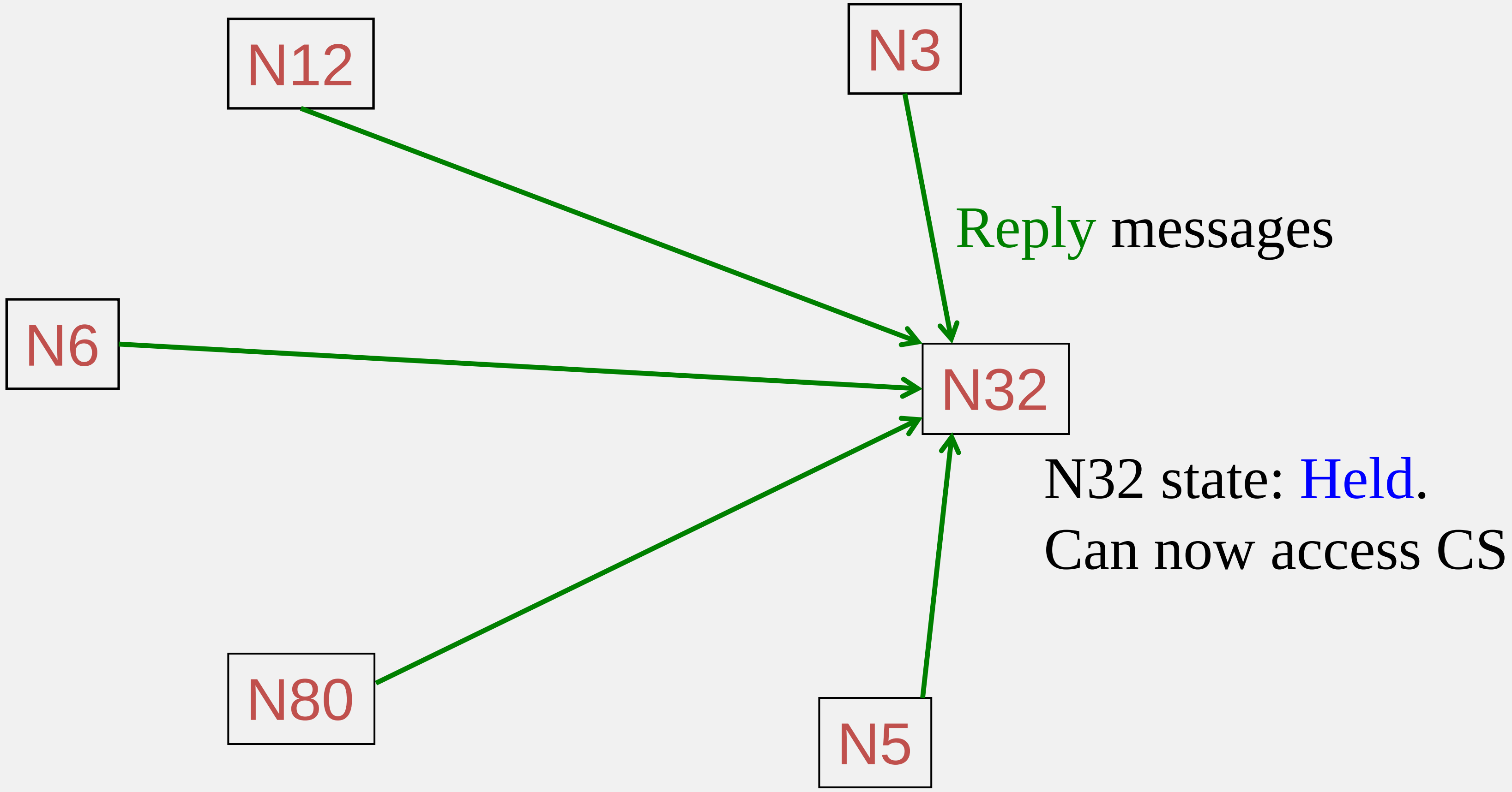
    add request to local queue (of waiting requests)

    else send "Reply" to Pj
- exit() at process Pi
  - change state to Released and "Reply" to *all* queued requests.

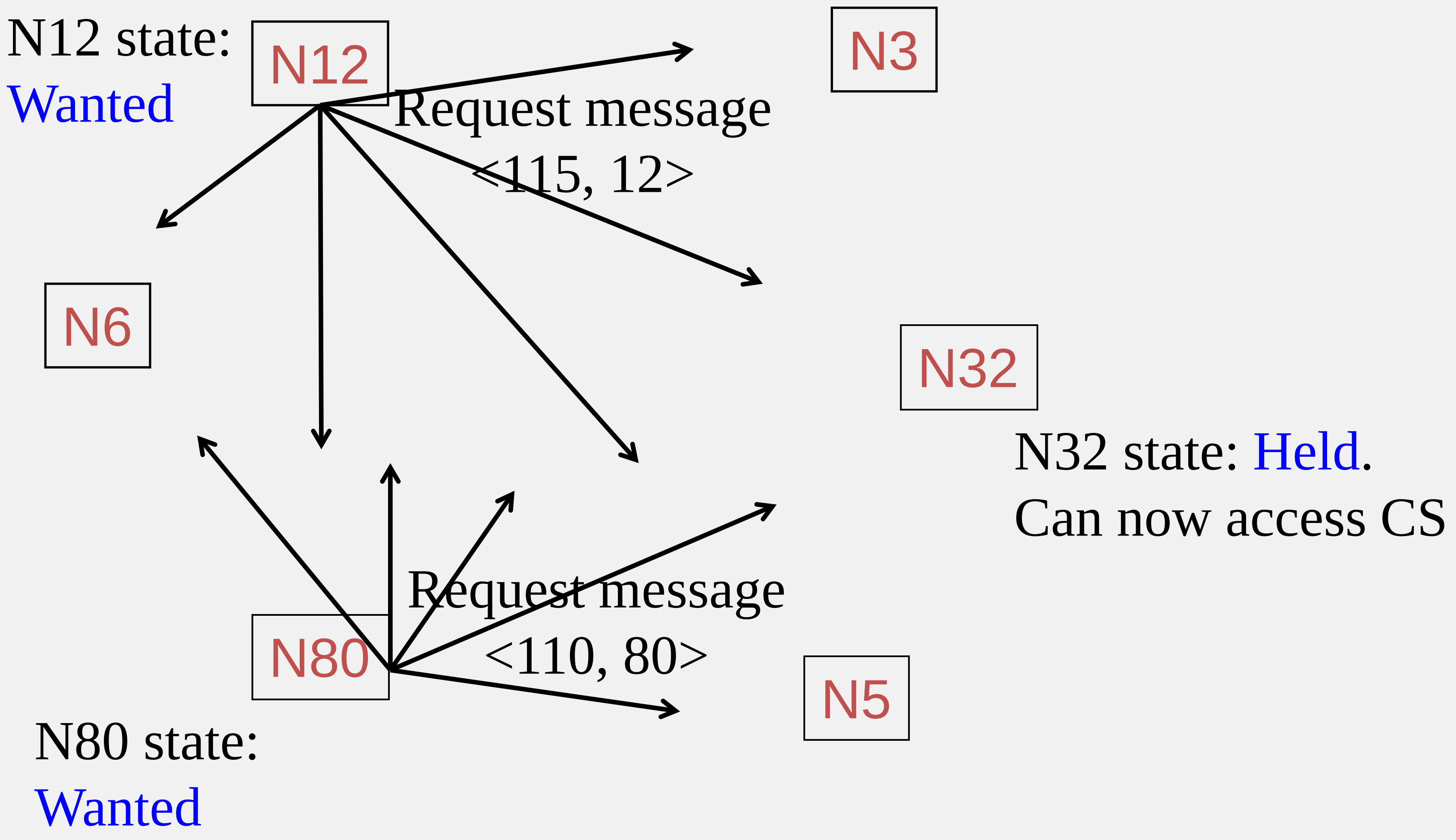
Example: Ricart-Agrawala Algorithm



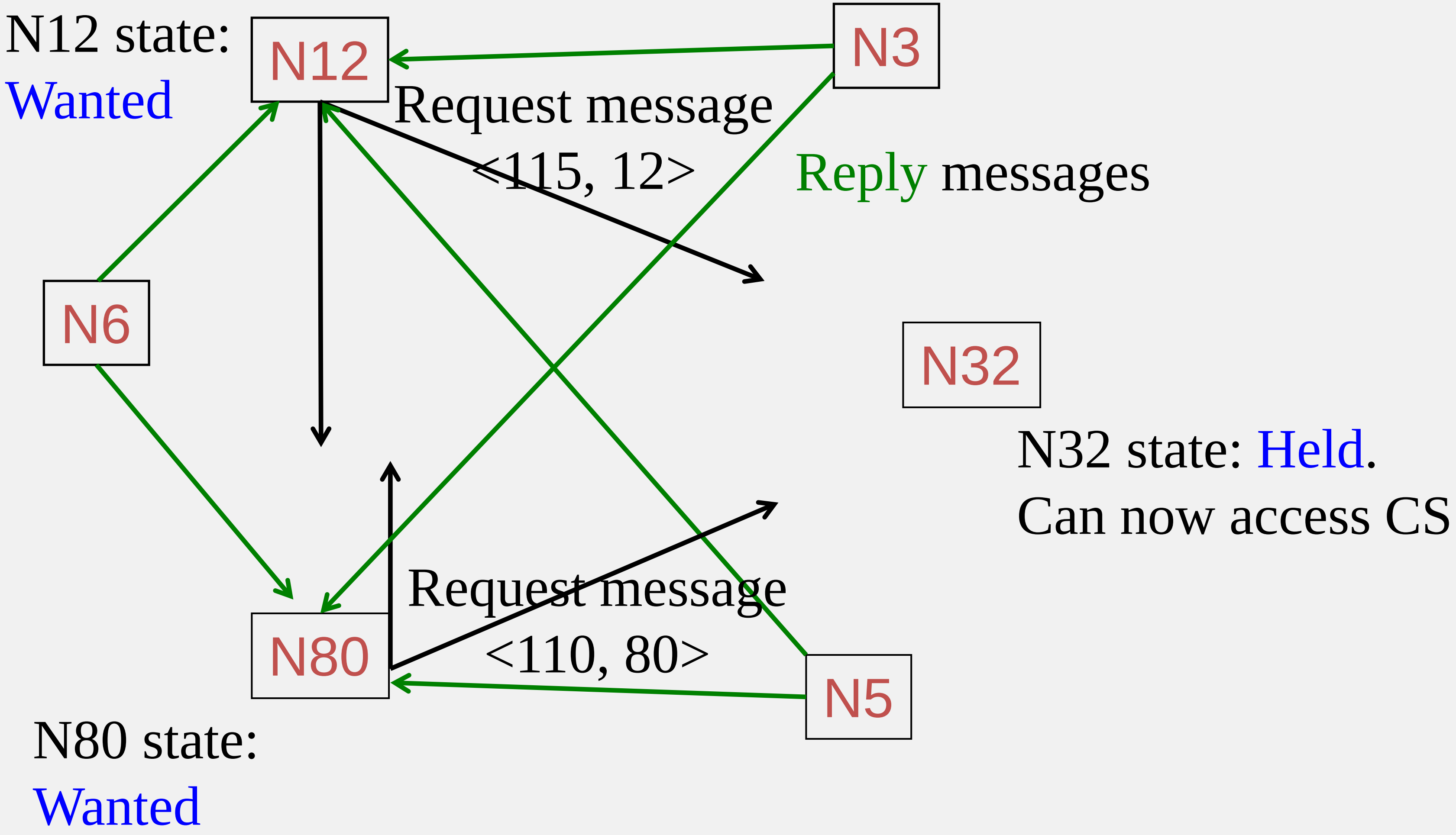
Example: Ricart-Agrawala Algorithm



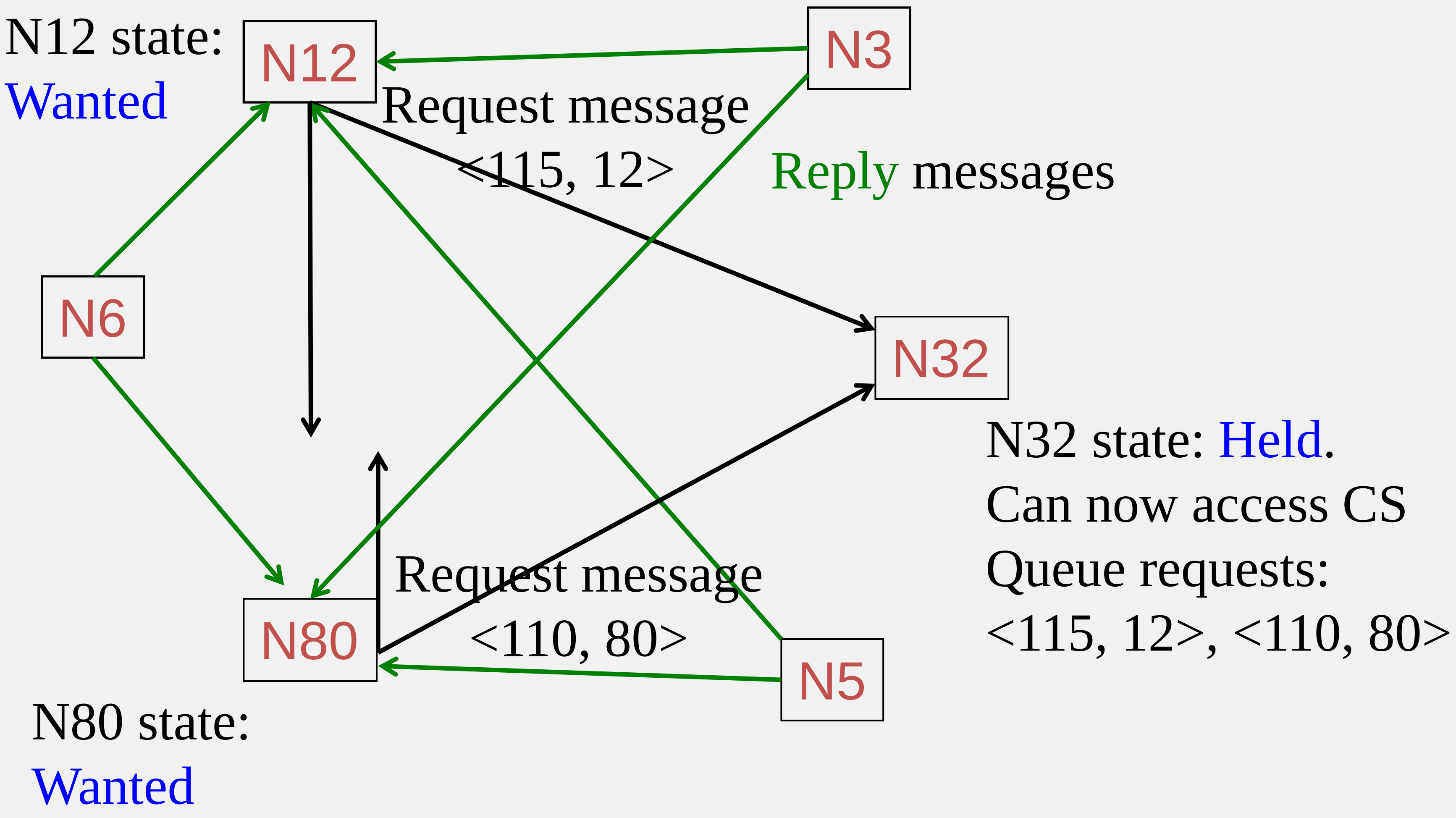
Example: Ricart-Agrawala Algorithm



Example: Ricart-Agrawala Algorithm

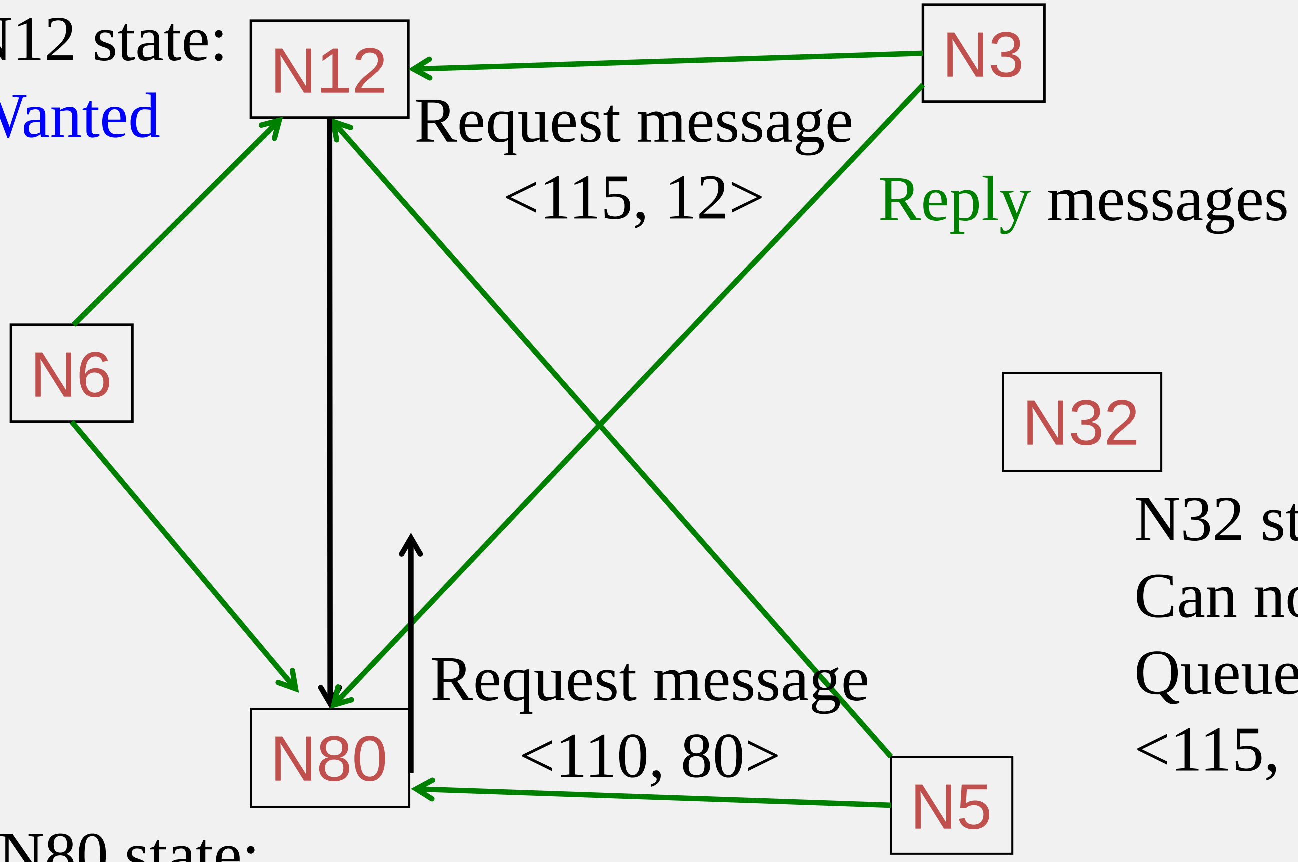


Example: Ricart-Agrawala Algorithm



Example: Ricart-Agrawala Algorithm

N12 state:
Wanted

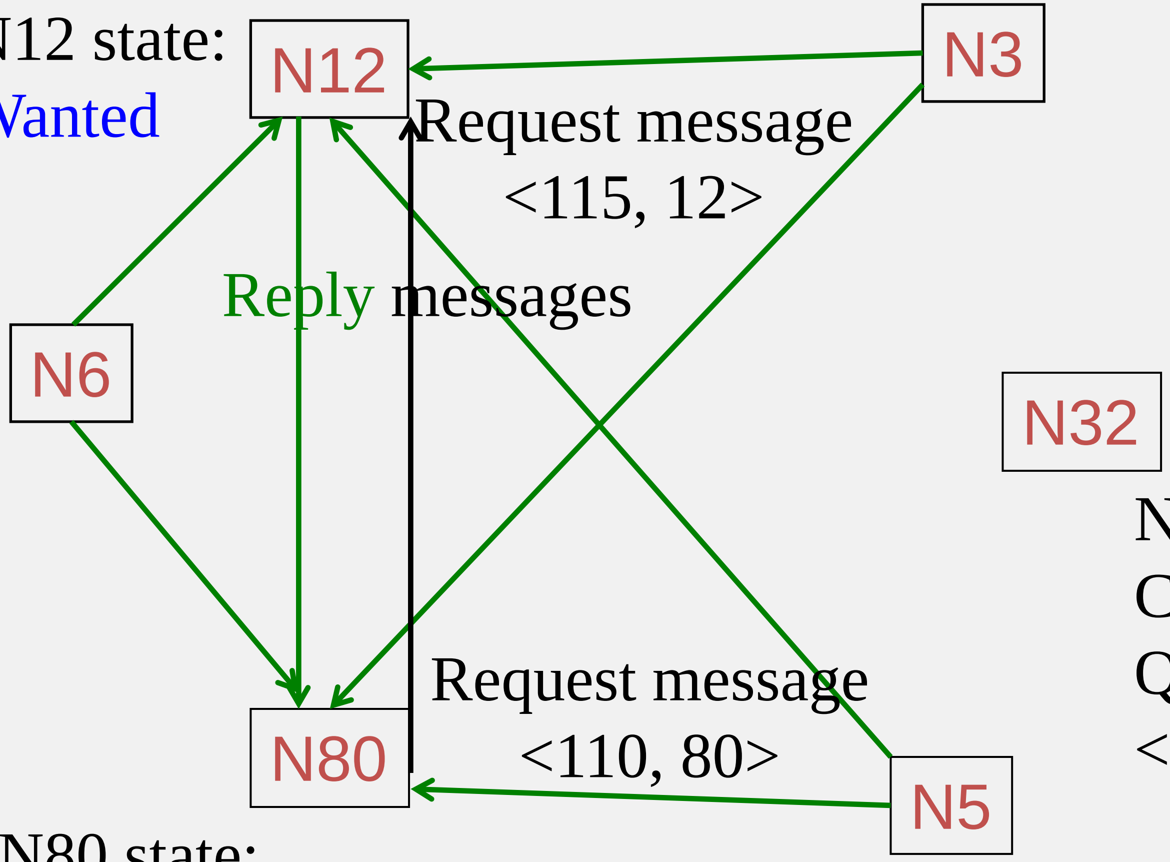


N32 state: Held.
Can now access CS
Queue requests:
<115, 12>, <110, 80>

N80 state:
Wanted
Queue requests: <115, 12> (since > (110, 80))

Example: Ricart-Agrawala Algorithm

N12 state:
Wanted

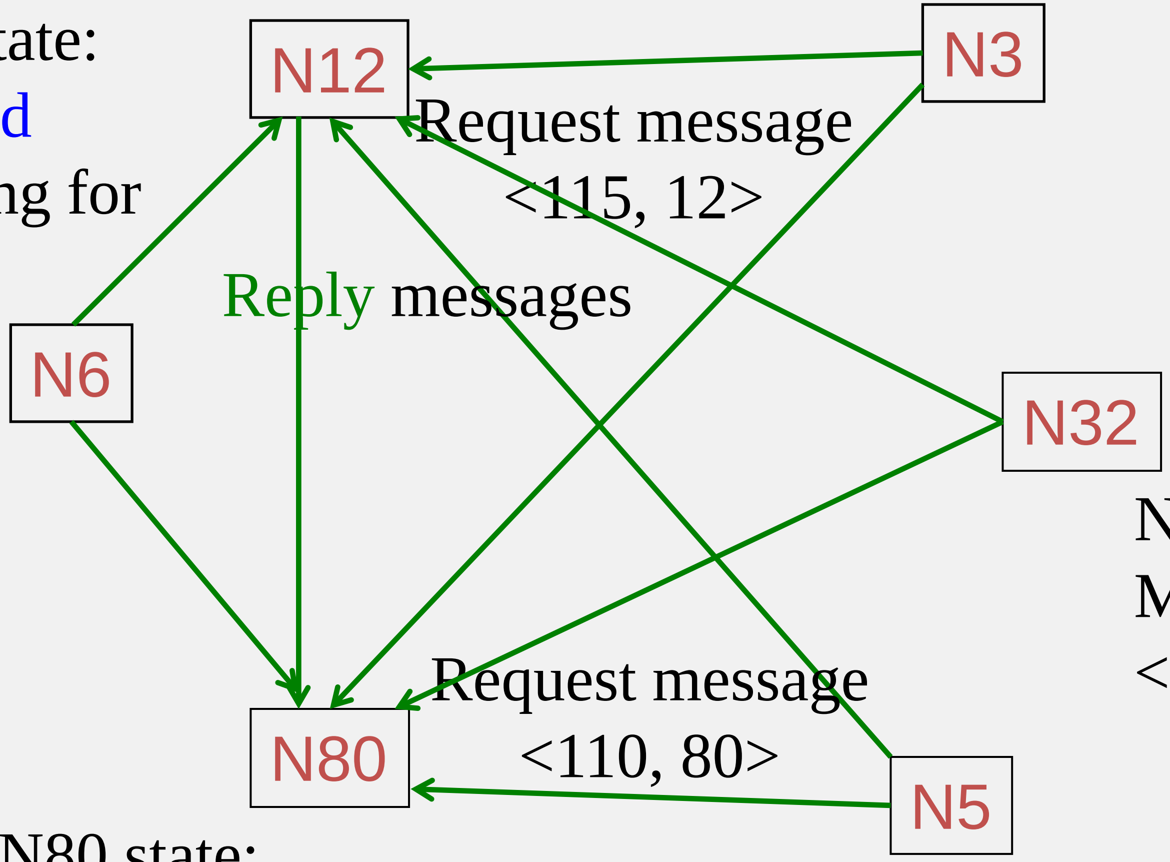


N32 state: Held.
Can now access CS
Queue requests:
<115, 12>, <110, 80>

N80 state:
Wanted
Queue requests: <115, 12>

Example: Ricart-Agrawala Algorithm

N12 state:
Wanted
(waiting for N80's reply)



N32 state: Released.
Multicast Reply to
<115, 12>, <110, 80>

N80 state:
Held. Can now access CS.
Queue requests: <115, 12>

Analysis: Ricart-Agrawala's Algorithm

- Safety
  - Two processes P*i* and P*j* cannot both have access to CS
    - If they did, then both would have sent Reply to each other
    - Thus, (T*i*, *i*) < (T*j*, *j*) and (T*j*, *j*) < (T*i*, *i*), which are together not possible
    - What if (T*i*, *i*) < (T*j*, *j*) and P*i* replied to P*j*'s request before it created its own request?
      - Then it seems like both P*i* and P*j* would approve each others' requests
      - But then, causality and Lamport timestamps at P*i* implies that T*i* > T*j* , which is a contradiction
      - So this situation cannot arise

Analysis: Ricart-Agrawala's Algorithm (2)

- Liveness
  - Worst-case: wait for all other (*N-1*) processes to send Reply
- Ordering
  - Requests with lower Lamport timestamps are granted earlier

Performance: Ricart-Agrawala's Algorithm

- Bandwidth: 2*(*N-1*) messages per enter() operation
  - *N-1* unicasts for the multicast request + *N-1* replies
  - *N* messages if the underlying network supports multicast
  - *N-1* unicast messages per exit operation
    - 1 multicast if the underlying network supports multicast
- Client delay: one round-trip time
- Synchronization delay: one message transmission time

Ok, but …

- Compared to Ring-Based approach, in Ricart-Agrawala approach
  - Client/synchronization delay has now gone down to O(1)
  - But bandwidth has gone up to O(N)
- Can we get *both* down?