Mutual exclusion in computer networks concurrency control Ricar and Agrawala's Algorithm

- Ricart and Agrawala's Algorithm
  - 2*(N-1) messages
    - N is the number of nodes
    - node will request permission to all nodes
      - must wait until permission is granted
    - assumptions
      - no shared memory
      - transmission time varies
        - messages not received in order they are sent
      - nodes are assumed to operate correctly
      - nodes have unique *node-number*

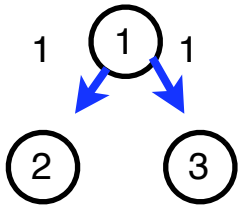
- Algorithm
  - node states
    - waiting for critical section
    - inside the critical section
    - idle
  - types of messages
    - `REQUEST`
      - request permission to enter critical section
    - `REPLY`
      - grant permission to enter a critical section

- each node has a `request-number`
  - initially set to zero
  - one higher than largest `request-number` seen on the network
  - `REQUEST` message is tagged with `request-number`

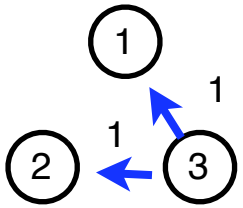
- algorithm
  - to enter critical section
    - send a `REQUEST` to all nodes
    - receive a `REPLY` from all nodes
  - node receiving `REQUEST`
    - idle
      - immediately `REPLY`
    - executing in critical section
      - delay `REPLY` until done with critical section

- waiting
  - compare request-numbers
    - if `REQUEST` has a lower `request-number`
      - `REPLY`
      - continue to wait
    - if `REQUEST` has a larger `request-number`
      - delay `REPLY`
      - continue to wait
    - if `REQUEST` has same `request-number`
      - lowest `node-number` has priority
  - node receiving `REQUEST` will always update `request-number`

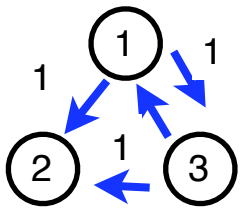
mutual exclusion-in networks



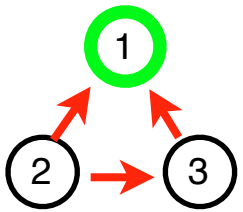
send REQUEST



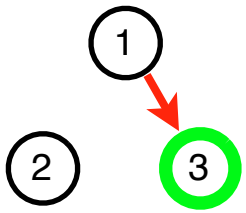
send REQUEST



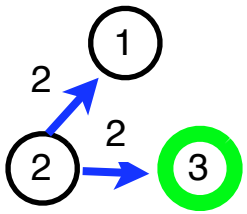
REQUEST arrives



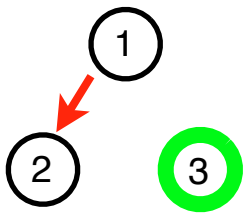
REPLY's



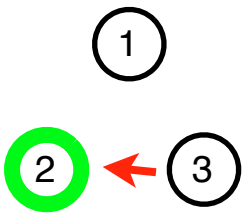
REPLY



send REQUEST



REPLY



REPLY

- implementation
  - three processes
  - run concurrently within each node
    - mutual exclusion
    - handle `REQUEST` messages
    - handle `REPLY` messages

- shared variables

```
#define  me                      /* my node number                    */
#define REPLY 0
int N;                           /* number of nodes                   */
int request_number;
int highest_reuest_number;       /* highest request number            */
int outstanding_reply;           /* # of outstanding replies          */
int request_CS;                  /* true when node requests critical
                                    section                           */
int reply_deferred[N];           /* reply_deferred[i] is true when
                                    node defers reply to node i       */
semaphore mutx;                  /* for mutual exclusion to shared
                                    variables                         */
semaphore wait_sem;              /* used to wait for all requests     */
```

- REQUEST message

```
/* k is the sequence number being requested  */
/* i is the node making the request  */

int defer_it   /* true if request must be deferred  */

if ( k > highest_request_number )
  highest_request_number = k;
P(mutex);
  defer_it =  (request_CS) &&
              ( ( k > request_number)  ||
                ( k == request_number && i > me ) );
V(mutex);

/* defer_it is true if we have priority  */
if ( defer_it)
   reply_deferred[i] = TRUE;
else
  send(REPLY, i);
```

- `REPLY` messages

```
outstanding_reply = outstanding_reply -- ;
V(wait_sem);
```

- mutual exclusion

```
P(mutex);
  request_CS = TRUE;
  request_number  = highest_request++;
V(mutex);
outstanding_reply = N-1;
for ( i=1; i<=N; i++ )
  send(request_number, i);
/* wait for replies  */
while ( outstanding_reply != 0 )
  P(wait_sem);
 CRITICAL SECTION;

request_CS = FALSE;
for ( i =1; i<=N; i++ ) {
  if ( reply_deferred[i] ) {
    reply_deferred[i] = FALSE;
    send (REPLY, i);
  }
}
```

- assertions
  - mutual exclusive is achieved
  - deadlock free
- node failure
  - time out mechanism
  - `are_you_there` message
- new nodes
  - broadcast to all nodes