Mutual Exclusion With Shared Memory

Lamport's Distributed Mutex Algorithm

- **Requesting process**

1. Push request in own queue (ordered by time stamps)
2. Broadcast request to all processes.
3. Wait for replies from all other processes.
4. If own request is at the head of queue and all replies have been received, enter critical section.
    1. **A response may have a lower timestamp if an earlier process sent a request. In which case we wait.**
    2. **At this point: everyone knows of your request, AND you are aware of any earlier requests**
5. Upon exiting the critical section, remove its request from the queue and send a release message to every process.

- **Other processes**

1. **After receiving a request, pushing the request in its own request queue (ordered by time stamps) and reply with a time stamp.**
    1. **Previous class: Reply after request is head of queue.**
    2. **Alternatively: If waiting for reply from j, wait till j replies to you**
2. After receiving release message, remove the corresponding request from queue.

Lamport's Bakery Algorithm

- Shared memory
- Non-atomic operations!
- To enter the critical section, each process acquires a ticket **number**, higher than current max known ticket number ("The doorway")
- Enter the critical section IFF the ticket number is higher than all others
- Lexicographic ordering to break ties (similar to Lamport clock.pid)
  - (Number[i], i) > (Number[k] > k)
  - If numbers are equal, then move to pid
- Each process writes to its own index in a shared array

```
Choosing: array [1..N] of bool = {false};  // N processes
Number: array [1..N] of integer = {0};
lock(integer i) {
        Choosing[i] = true;
M:      Number[i] = 1 + max(Number[1], ..., Number[N]);  //non-atomic
        Choosing[i] = false;
        for (k = 1; k <= N; k++) {
                // Wait until thread j receives its number:
                L2: while (Choosing[k]);
                // Wait until all threads with smaller numbers or with the same
                // number, but with higher priority, finish their work:
                L3: while ((Number[k] != 0) && ((Number[k], k) < (Number[i], i))) ; } }


unlock(integer i) {
        Number[i] = 0; }
```

Bakery algorithm can be converted to distributed locking

- Leslie Lamport. CACM. September 2022 (Recent!)