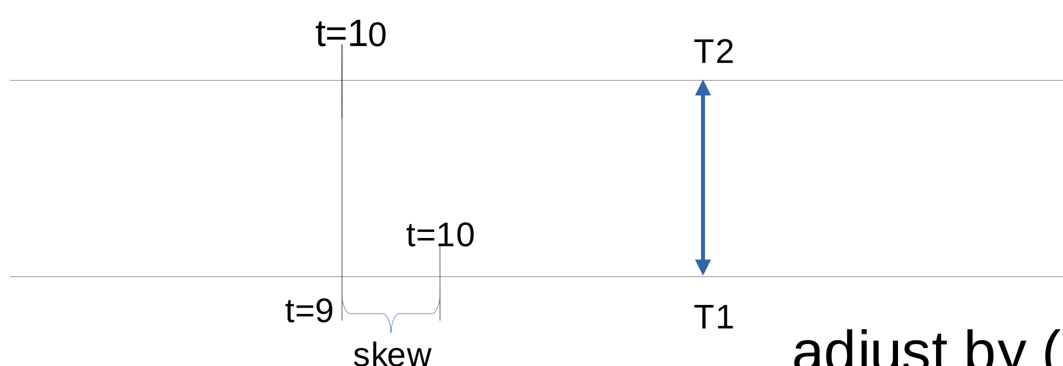
Sistemas Distribuídos

José Orlando Pereira

Departamento de Informática
Universidade do Minho

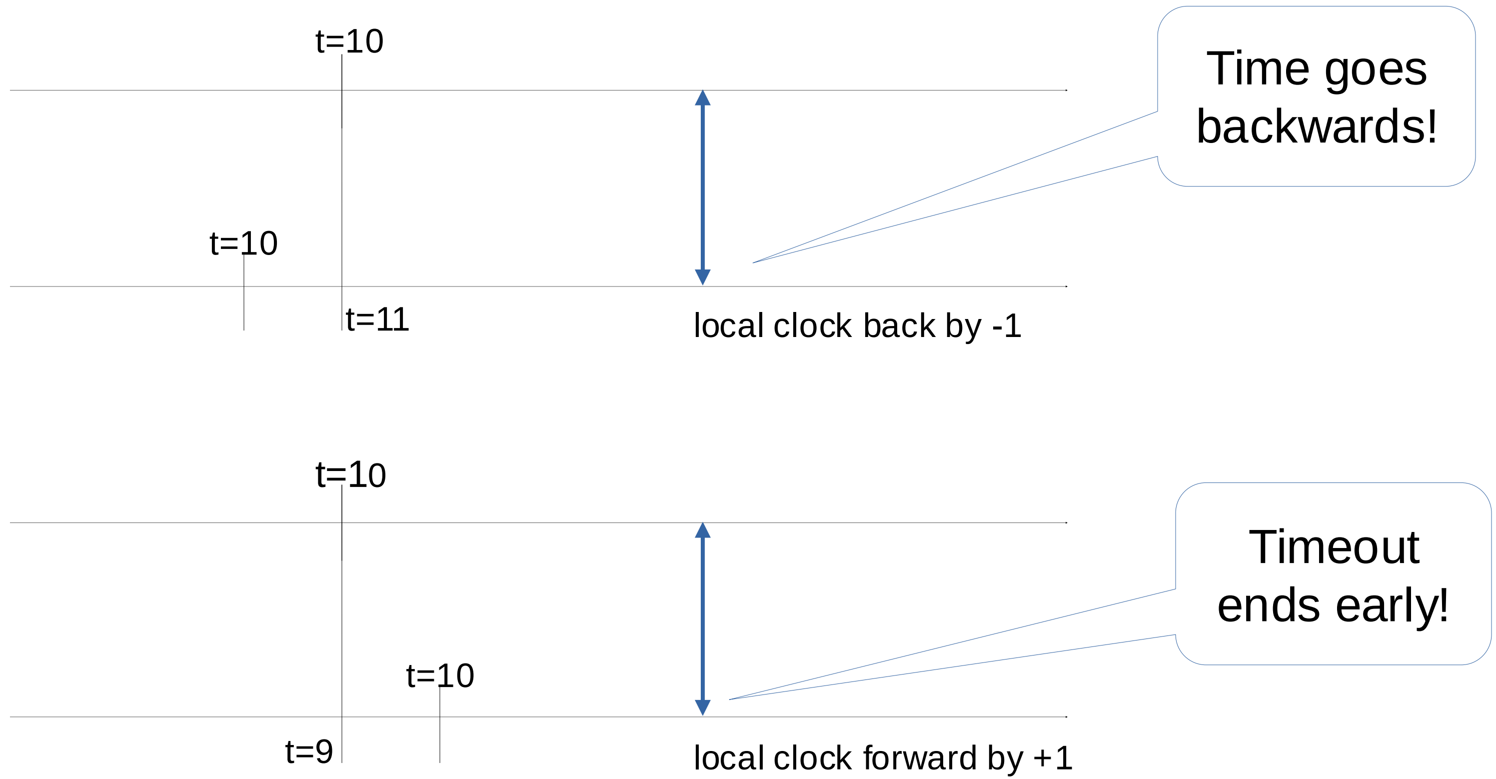
Clock synchronization

- Hardware clocks are not perfect and drift over time
- Clock skew can be a problem with:
  - shared files
  - certain algorithms…
- Ideally:



adjust by (T2-T1)

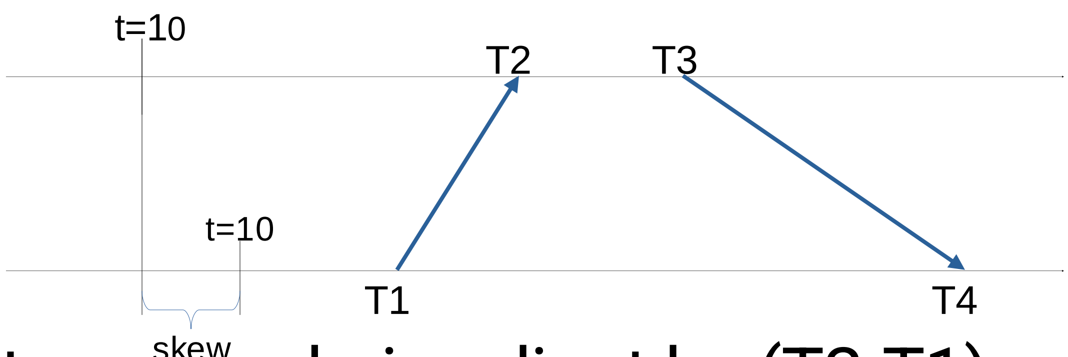
Clock synchronization



- The clock must be <u>adjusted in small increments</u>, over a longer period of time by making it faster or slower

Clock synchronization

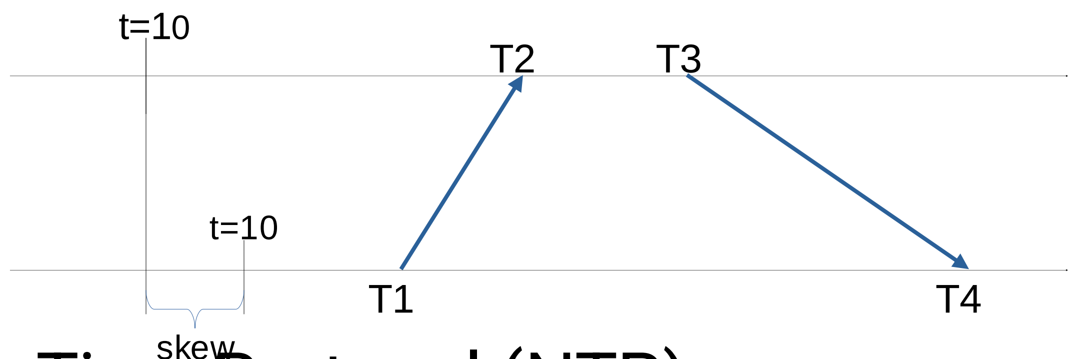
- In practice, there are unpredictable
  - transmission delays
  - processing delays



- The best we can do is adjust by (T2-T1) – (estimated message delay)

Clock synchronization

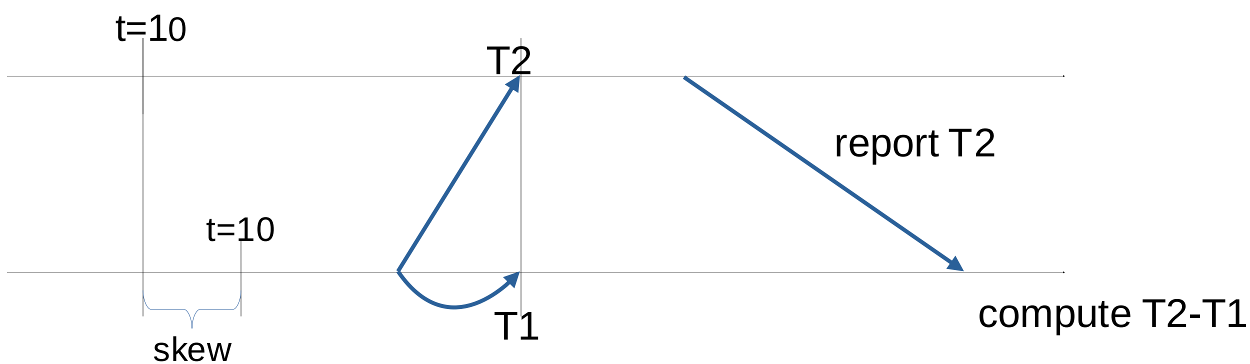
- What is the message delay?



- Network Time Protocol (NTP):
  - assume delays are the same = ((T2-T1)+(T4-T3))/2
  - repeat several times and pick the smallest delay

Clock synchronization

- Reference Base Synchronization (RBS):
  - assume true broadcast medium (aprox. zero skew)

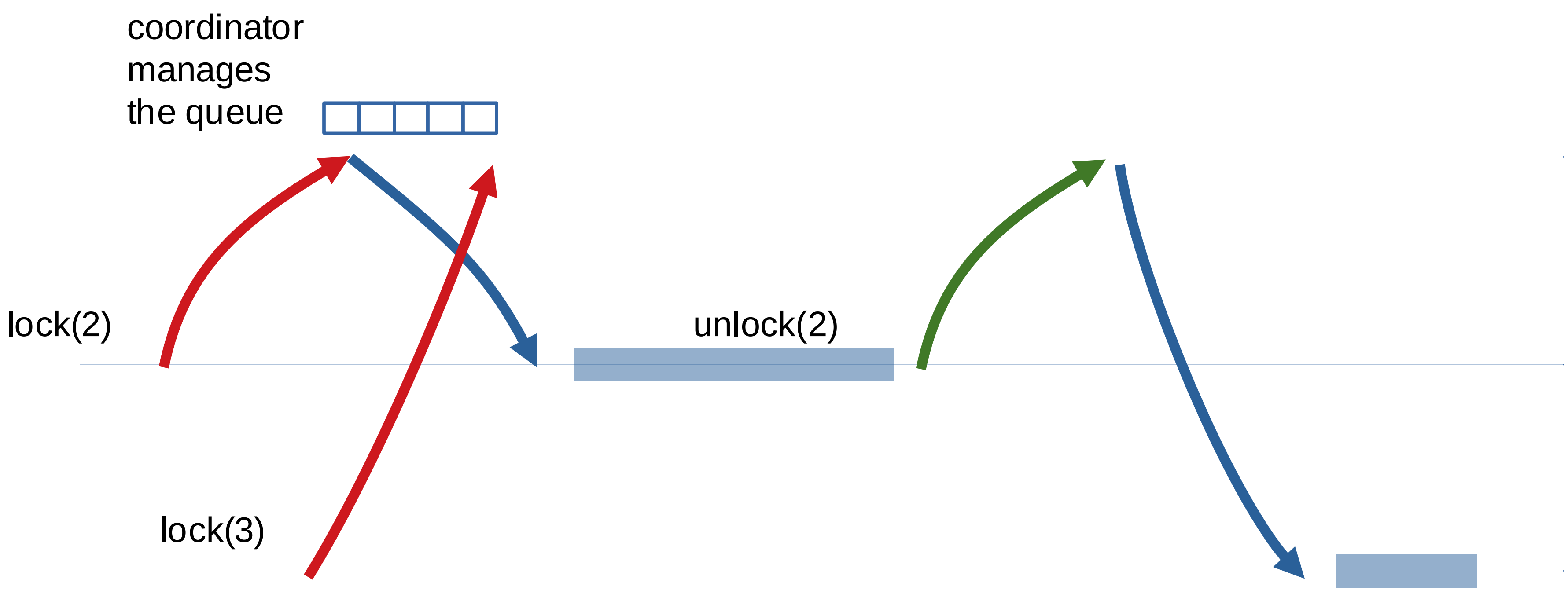


Mutual exclusion

- Solution in a distributed system?
- Recall the definition of mutual exclusion:
  - No two threads in the critical section
  - No deadlock / no starvation
- Comparison criteria:
  - Number of message hops to entering critical section
  - Load balancing

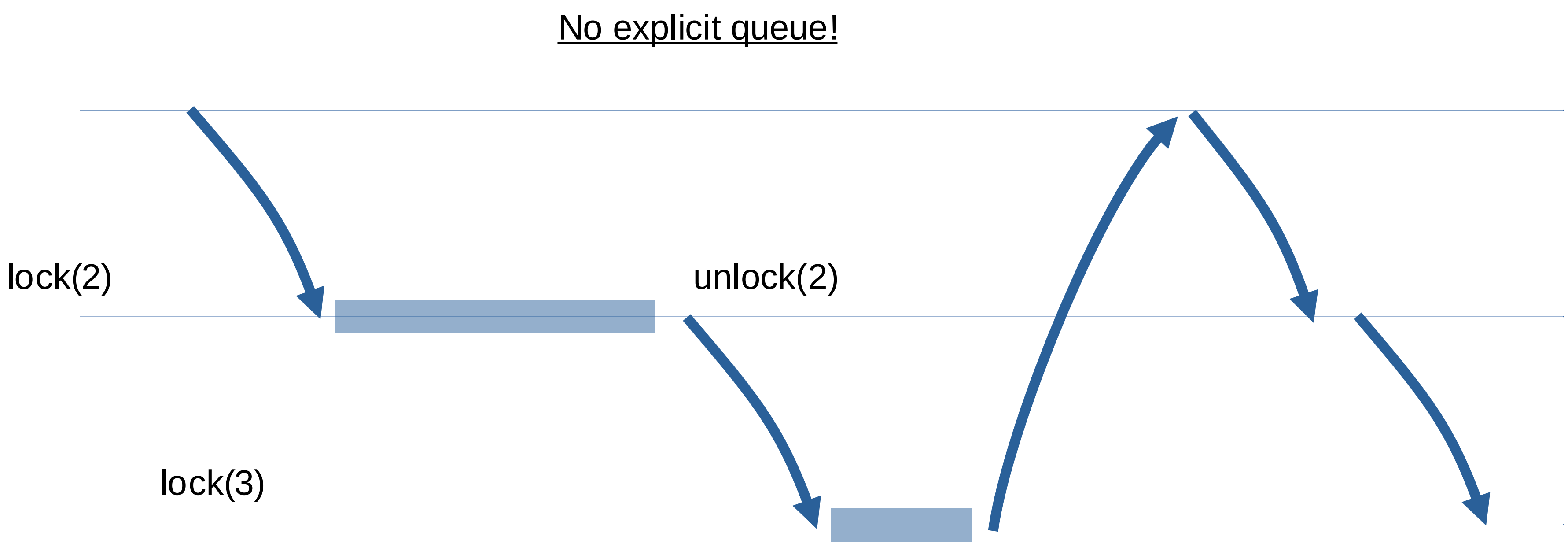
Mutual exclusion

- Centralized queue kept by a coordinator:
  - 1 round-trip to enter / asymmetric load



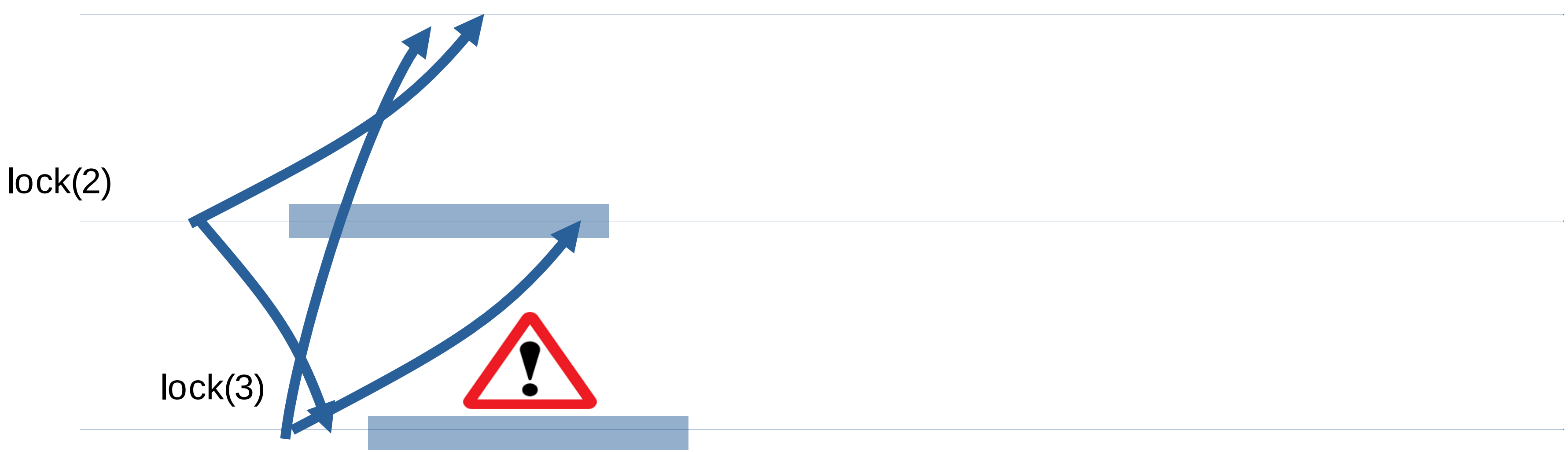
Mutual exclusion

- Exchange a token in a ring:
  - n/2 hops to entering / symmetric load



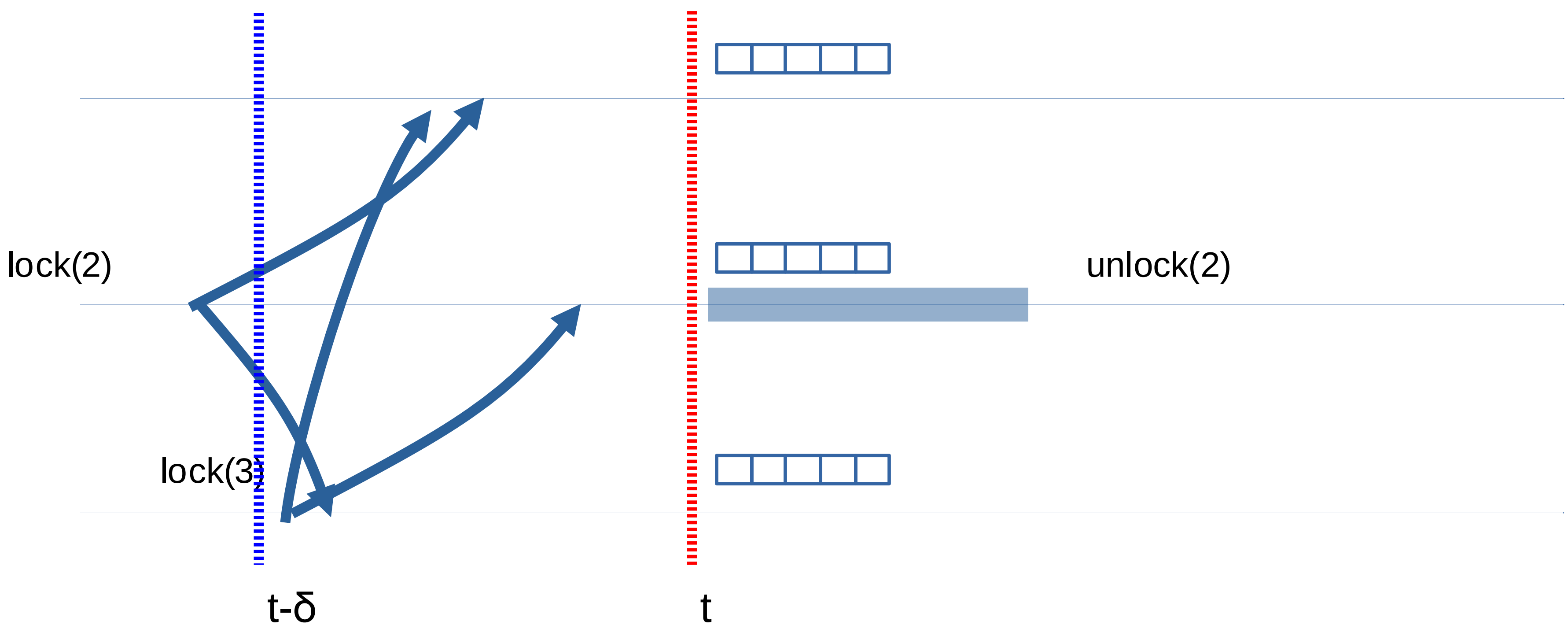
Mutual exclusion

- A distributed algorithm is hard to achieve:
  - As concurrent lock requests are received by different destinations in different orders, safety is not ensured



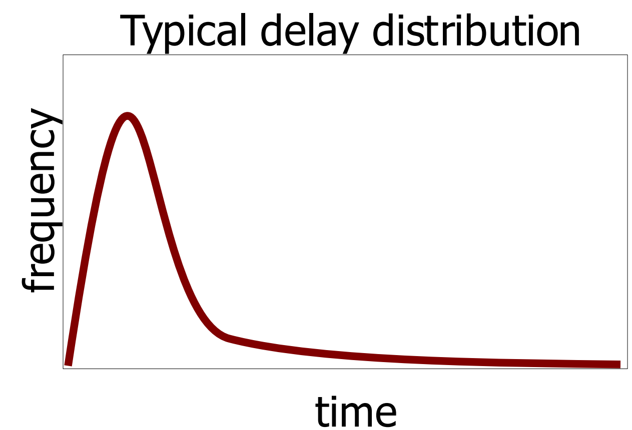
Mutual exclusion

- Taking advantage of synchronized clocks:
  - Assume δ > (transmission delay + skew)
  - Consider only messages up to t-δ, order by timestamp



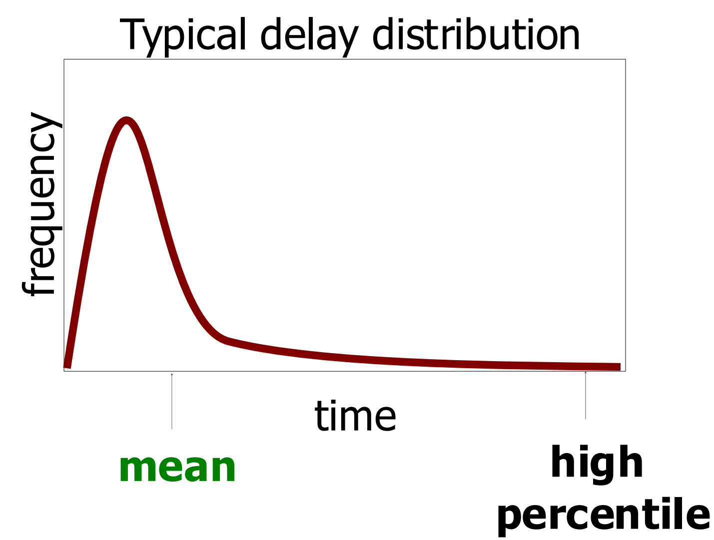
Timeouts

- Used to assess status of remote processes
- Tight timeouts are dangerous:
  - E.g., proportional to mean delay
  - Means low coverage
- Large timeouts are not useful:
  - E.g., proportional to high percentile
  - Taking advantage of time causes a very large performance penalty



Using real time

- Solutions that do not use time are more robust:
  - In wide area networks
  - With performance perturbations
- Solutions that do not use time might have better performance:
  - Run time proportional to mean delay
  - Even if more message exchanges are necessary

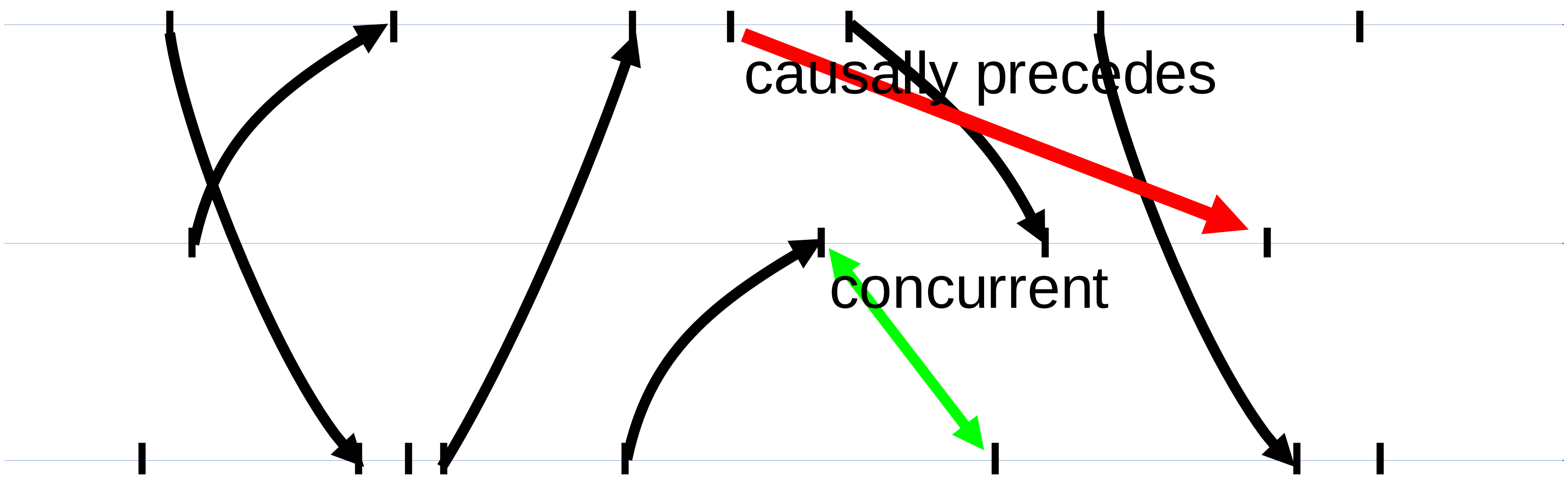


Asynchronous system model

- Assume no global time reference
- Assume no bounds on:
  - clock drift
  - processing time
  - message passing time
- Can we still solve important problems?

Time and causality

- What is special about time that makes it useful for distributed algorithms?

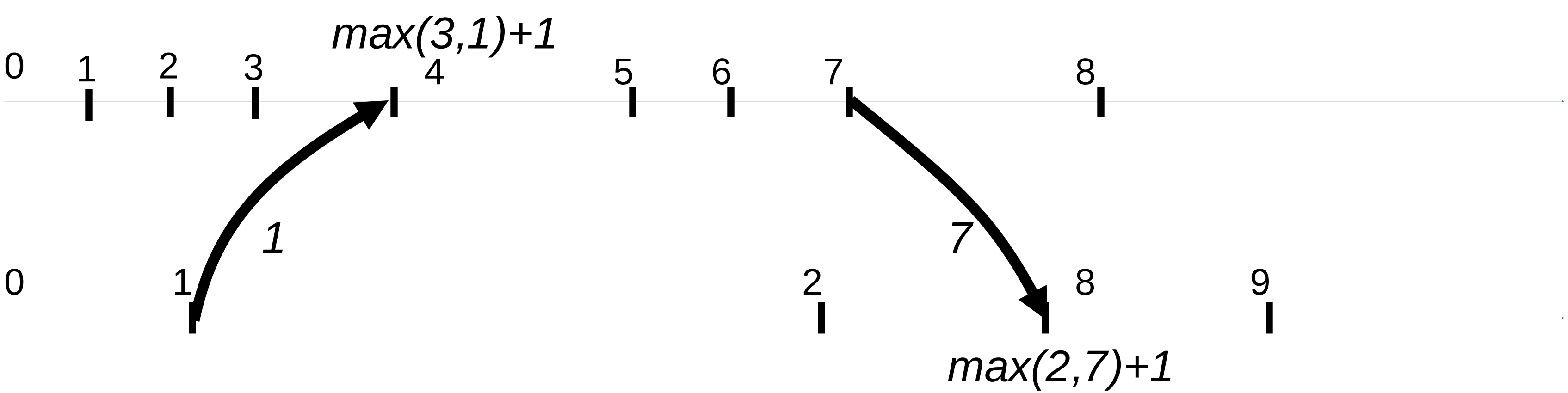


Time and causality

- *Clock(i)* the time at which i happened
- If *i precedes j* then *Clock(i)<Clock(j)*
- For some event j:
  - When we are sure that there is no unknown i such that Clock(i)<Clock(j)
  - Then there is no i such that *i precedes j*
- **Can we build a logical clock with the same property?**

Lamport's logical clocks

- Local events: increment counter
- Send events: increment and then tag with counter
- Receive events: update local counter to maximum and then increment

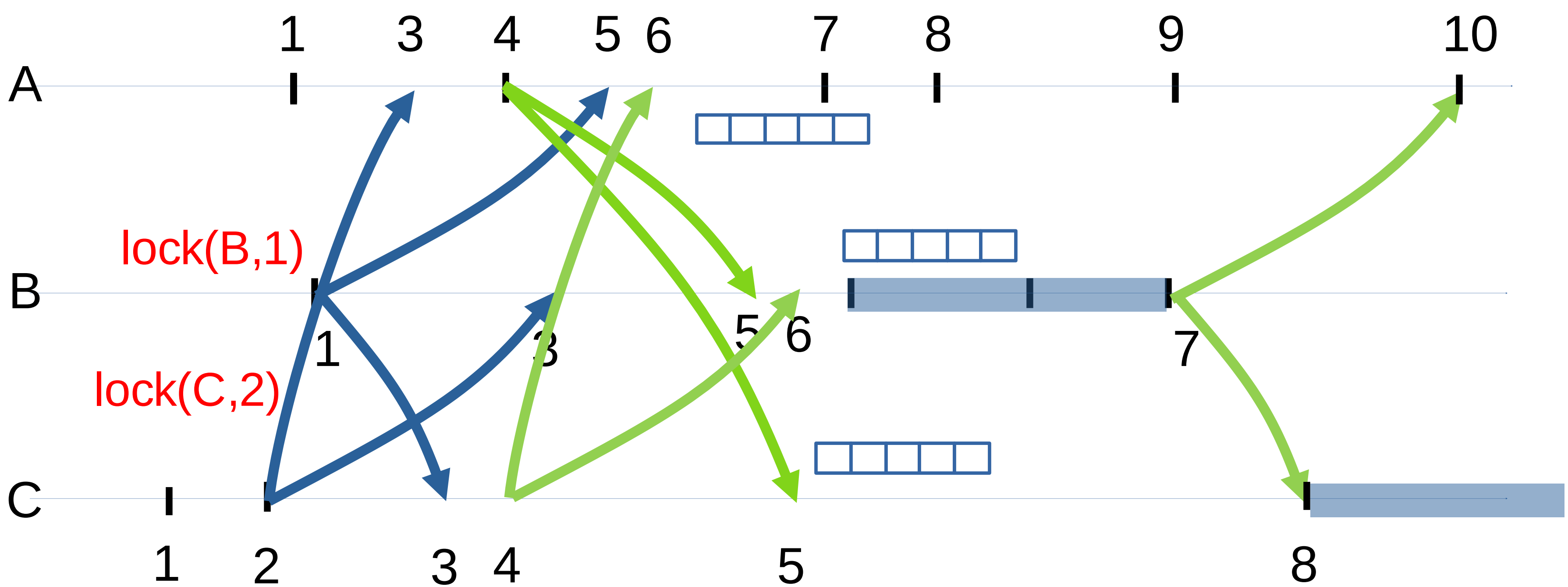


Mutual exclusion

- Algorithm sketch:
  - Start by assuming that processes are continually exchanging messages over FIFO channels...
  - ri[j] latest timestamp from j at i
  - Consider requests with t <= min(ri[j], for all j)
    - (akin to t-δ!)
  - Order by timestamp, break ties by process id
- (The complete version is the Ricart-Agrawala distributed mutex algorithm)

Mutual exclusion

- 1 hop to enter / symmetric load



Conclusion

- The same approach used for the waiting queue in the mutex can be used for other deterministic applications
  - Replicated State Machine (RSM)
- Logical time is widely applicable in distributed systems to solve many problems