## Distributed Coordination

- What makes a system distributed?
- Time in a distributed system
- How do we determine the global state of a distributed system?
- Event ordering
- Mutual exclusion

## Distr. Systems: Fundamental Characteristics

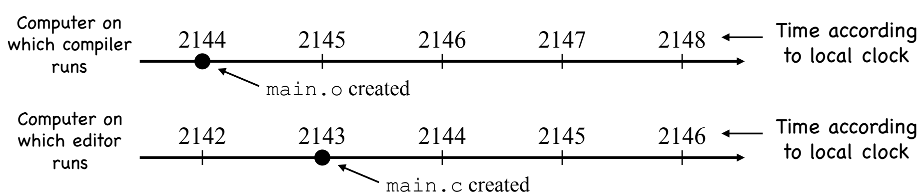
1. Multiple processors (*wlog*: assume one process per processor)
2. No shared memory
3. No common clock
4. Communication delays are not constant
5. Message ordering may not be maintained by the underlying communication infrastructure

# Effects of Lack of Common Clock

Example 1 : Distributed `make` utility (e.g. `pmake`)

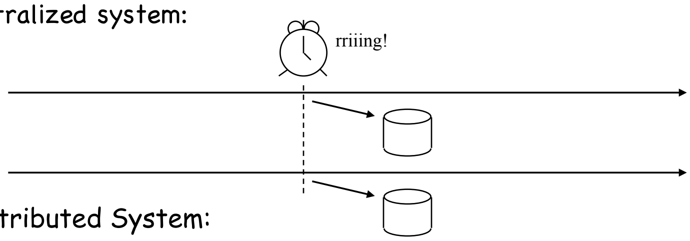
- `make` goes through all target files and determines (based on timestamps) which targets need to be "(re)compiled"
- Example:

```
main   : main.o
         cc -o main main.o
main.o : main.c
         cc -c main.c
```

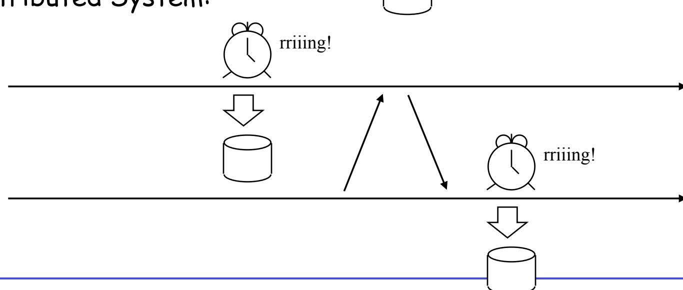



# Effects of Lack of Common Clock

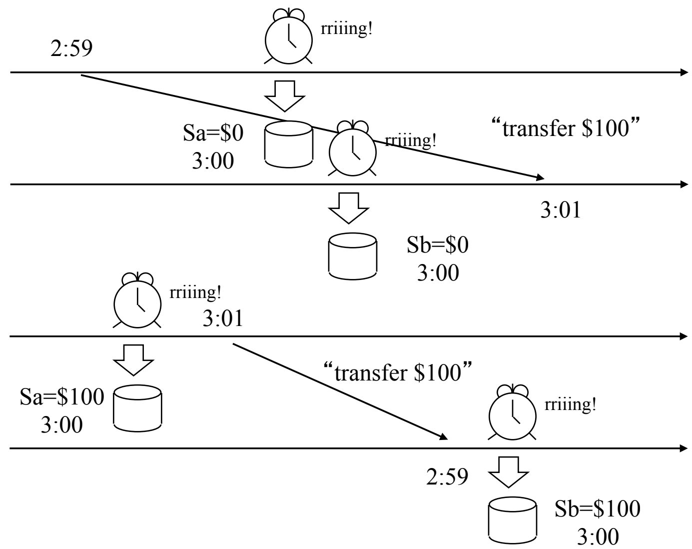
- Example 2 : Distributed Checkpointing
- "At 3pm everybody writes its state to stable storage."
- Centralized system:



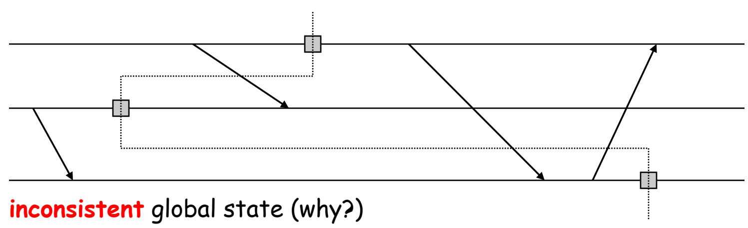

- Distributed System:



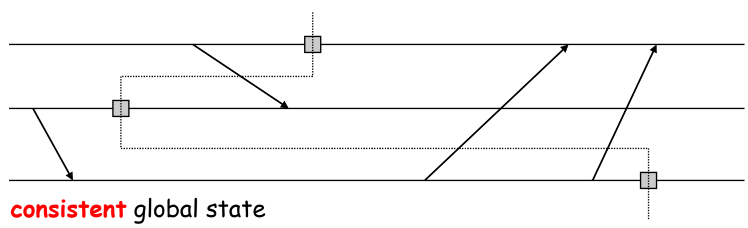
## Distributed Checkpointing (2)




## Consistent *vs.* Non-Consistent Global States



**inconsistent** global state (why?)



**consistent** global state

## Distributed Snapshot Algorithm (Chandy, Lamport)

- Process $P$ starts algorithm:
  - saves state $S_P$
  - sends out marker messages to all other processes
- Upon receipt of a marker message (from process $Q$), process $P$ proceeds as follows (atomically: no messages sent/received in the meantime):
  - 1. Saves local state $S_P$.
  - 2. Records state of incoming channel from $Q$ to $P$ as empty.
  - 3. Forward marker message on all outgoing channels.
- At any time after saving its state, when $P$ receives a marker from a process $R$:
  - Save state $SC_{RP}$ as sequence of messages received from $R$ since $P$ saved local state $S_P$ to when it received marker from $R$.

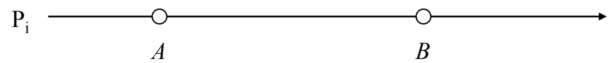
## Comments

- Any process can start algorithm.
- Even multiple processes can start it concurrently.
- Algorithm will terminate if message delivery time is finite.
- Algorithm is fully distributed.
- Once algorithm has terminated, consistent global state can be collected.
- Relies on ordered, reliable message delivery.

# Event Ordering

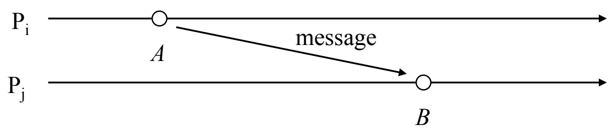
- Absence of central time means: no notion of *happened-when* (no total ordering of events)
- But can generate a *happened-before* notion (partial ordering of events)
- *Happened-Before* relation:

1.



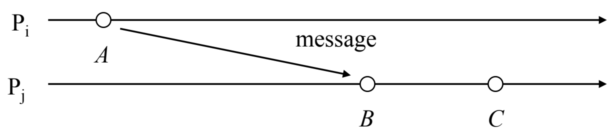

Event *A* happened-before Event *B*. (*A -> B)*

2.




Event *A* happened-before Event *B*. (*A -> B)*

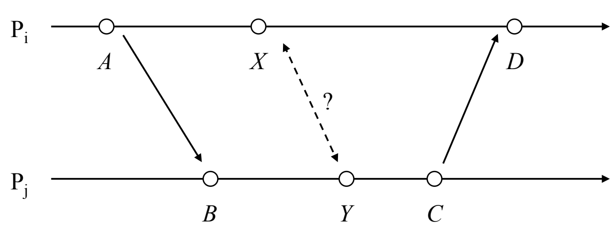
3.




Event *A* happened-before Event *C*. (*A -> C) (transitivity)*

# Concurrent Events

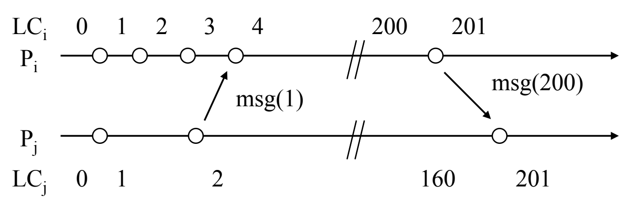
- What when no *happened-before* relation exists between two events?




Events *X* and *Y* are **concurrent**.

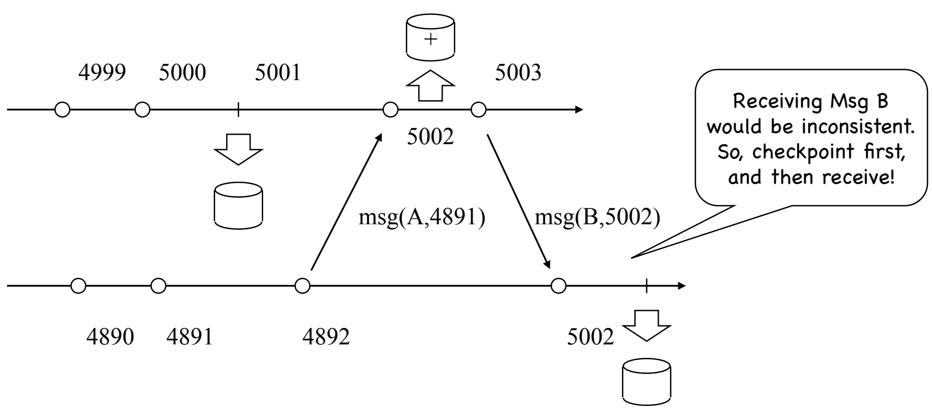
## Happened-Before Ordering: Implementation

- Define a Logical Clock $LC_i$ at each Process $P_i$.
- Used to timestamp each event:
  - Each event on $P_i$ is timestamped with current value of logical clock $LC_i$.
  - After each event, increment $LC_i$.
  - Timestamp each outgoing message at $P_i$ with value of $LC_i$.
  - When receiving a message with timestamp $t$ at process $P_j$, set $LC_j$ to $max(t, LC_j)+1$.




## Application to Distributed Checkpointing

"At logical-clock time 5000 write state to stable storage!"

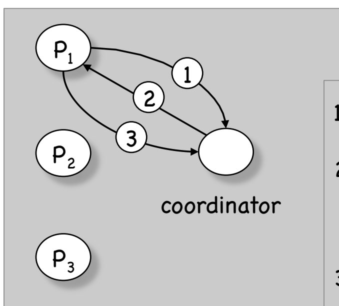


## Simple Example: Mutual Exclusion (*)

Recall: Mutual exclusion in shared-memory systems:

```
bool lock; /* init to FALSE */

while (TRUE) {

  while (TestAndSet(lock)) no_op;

  critical section;

  lock = FALSE;

  remainder section;

}
```

## Distributed Mutual Exclusion (D.M.E.): Centralized Approach (*)

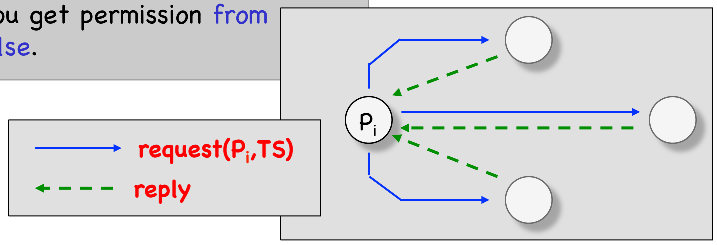



1. Send **request** message to coordinator to enter critical section (C.S.)
2. If C.S. is free, the coordinator sends a **reply** message. Otherwise it queues request and delays sending *reply* message until C.S. becomes free.
3. When leaving C.S., send a **release** message to inform coordinator.

Characteristics:
- ensures mutual exclusion
- service is fair
- small number of messages required
- fully dependent on coordinator

# D.M.E.: Fully Distributed Approach (*)

**Basic idea:** Before entering C.S., ask and wait until you get permission from everybody else.

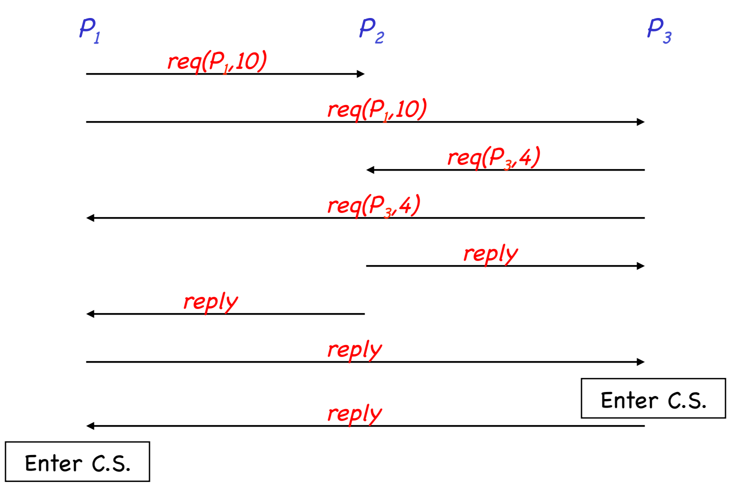



Upon receipt of a message **request($P_j$, $TS_j$)** at node $P_i$:

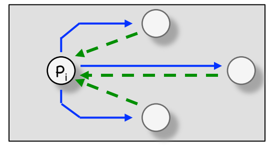
1. if $P_i$ does not want to enter C.S., immediately send a **reply** to $P_j$.
2. if $P_i$ is in C.S., defer **reply** to $P_j$.
3. if $P_i$ is trying to enter C.S., compare $TS_i$ with $TS_j$. If $TS_i > TS_j$ (i.e. "$P_j$ asked first"), send **reply** to $P_j$; otherwise defer **reply**.

# Fully Distributed Approach: Example (*)

*Scenario:* $P_1$ and $P_3$ want to enter C.S.



## D.M.E. Fully Distributed Approach (*)

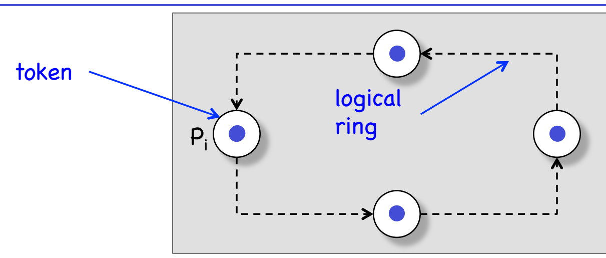



The Good:
- ensures mutual exclusion
- deadlock free
- starvation free
- number of messages per critical section: *2(n-1)*

The Bad:
- The processes need to know identity of all other processes involved ("join" & "leave" protocols needed)

The Ugly:
- One failed process brings the whole scheme down!

## D.M.E.: Token-Passing Approach (*)




- Token is passed from process to process (in logical ring)
- Only process owning a token can enter C.S.
- After leaving the C.S., token is forwarded

Characteristics:
- mutual exclusion guaranteed
- no starvation
- number of messages per C.S. varies

Problems:
- Process failure (new logical ring must be constructed)
- Loss of token (new token must be generated)

## Just for Fun: Recovering Lost Tokens (**)

Solution: use **two** tokens!
- When one token reaches $P_i$, the other token has been **lost** if
  the token has not met the other token since last visit
  **and**
  $P_i$ has not been visited by other token since last visit.

Algorithm:
- uses two tokens, called "ping" and "pong"
```
int nping = 1; /*invariant: nping+npong = 0 */
int npong = -1;
```
- each process keeps track of value of last token it has seen.
```
int m = 0; /* value of last token seen by Pi */
```

## "Ping-Pong" Algorithm (**)

upon arrival of ("ping", nping)
```
if (m == nping) {
  /* "pong" is lost!
      generate new one. */
  nping = nping + 1;
  pong = - nping;
}
else {
  m = nping;
}
```

when tokens meet
```
nping = nping + 1;
npong = npong - 1;
```

upon arrival of ("pong", npong)
```
if (m == npong) {
  /* "ping" is lost!
      generate new one. */
  npong = npong - 1;
  ping = - npong;
}
else {
  m = npong;
}
```

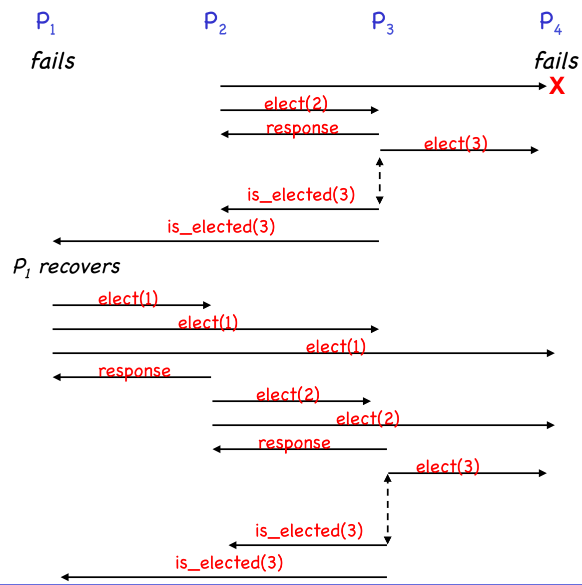
## Election Algorithms

- Many distributed algorithms rely on coordinator.
- Coordinator may fail. Then system must start a new coordinator
- Election algorithms determine where the new coordinator will be located.
- Remarks:
  - Each process has a priority number (*wlog* $P_i$ has priority $i$ )
  - Election algorithm picks active process with highest priority and informs all active processes about new coordinator.
  - Newly recovered process should be able to identify current coordinator.

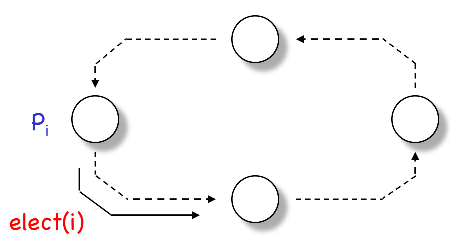
## Election: The Bully Algorithm (Garcia-Molina)

- Process $P_i$ times out during a request to coordinator; assumes that coordinator has failed.
- $P_i$ proceeds to elect itself as coordinator by sending *elect(i)* message to higher-priority processes.
  - If receives no response, considers itself elected and informs all lower-priority processes with a *is_elected(i)* message.
  - If receives reply, waits to hear who has been elected. If times out, assumes that something went wrong (processes failed), and restarts from scratch.
- At process $P_i$:
  - message *is_elected(j)* comes in ($j > i$): record information
  - message *elect(j)* comes in:
    - if ($i < j$) wait and see
    - if ($i > j$) send response to $P_j$ and start own election campaign.
- If process recovers from failure, starts new election campaign.

## Bully Algorithm: Example




## Election: Ring Algorithm




- Basic version:
  - Each process $P_i$ sends its own election message *elect(i)* around the ring.
  - All processes send their own number before passing on election messages of other processes.
  - When its own message returns, $P_i$ knows it has seen all the messages.
- How many messages are needed per election round?