Distributed Time

Marquette University

Physical Time

- System clock
  - Precisely machined quartz crystal
  - Counter and Holding register
  - Each oscillation of quartz decrements counter
  - When counter gets to zero, generate interrupt
    - Counter is reloaded from holding register
    - Time is incremented

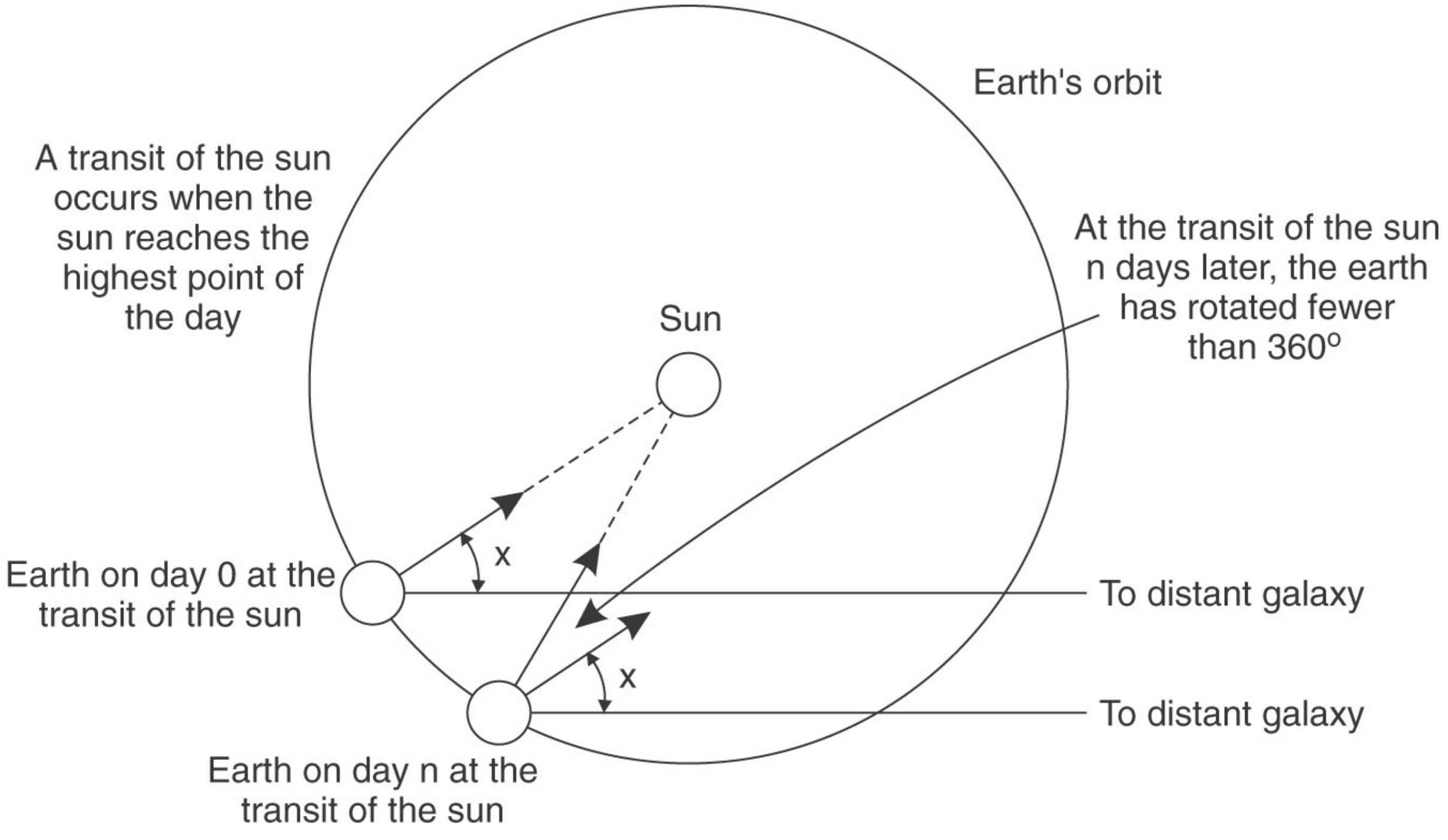
Physical Time

- On a single system
  - Absolute time does not really matter
  - Important are relative times
  - Example:
    - Make will recompile *.c files if their modified time is later than the corresponding *.o file

Physical Time

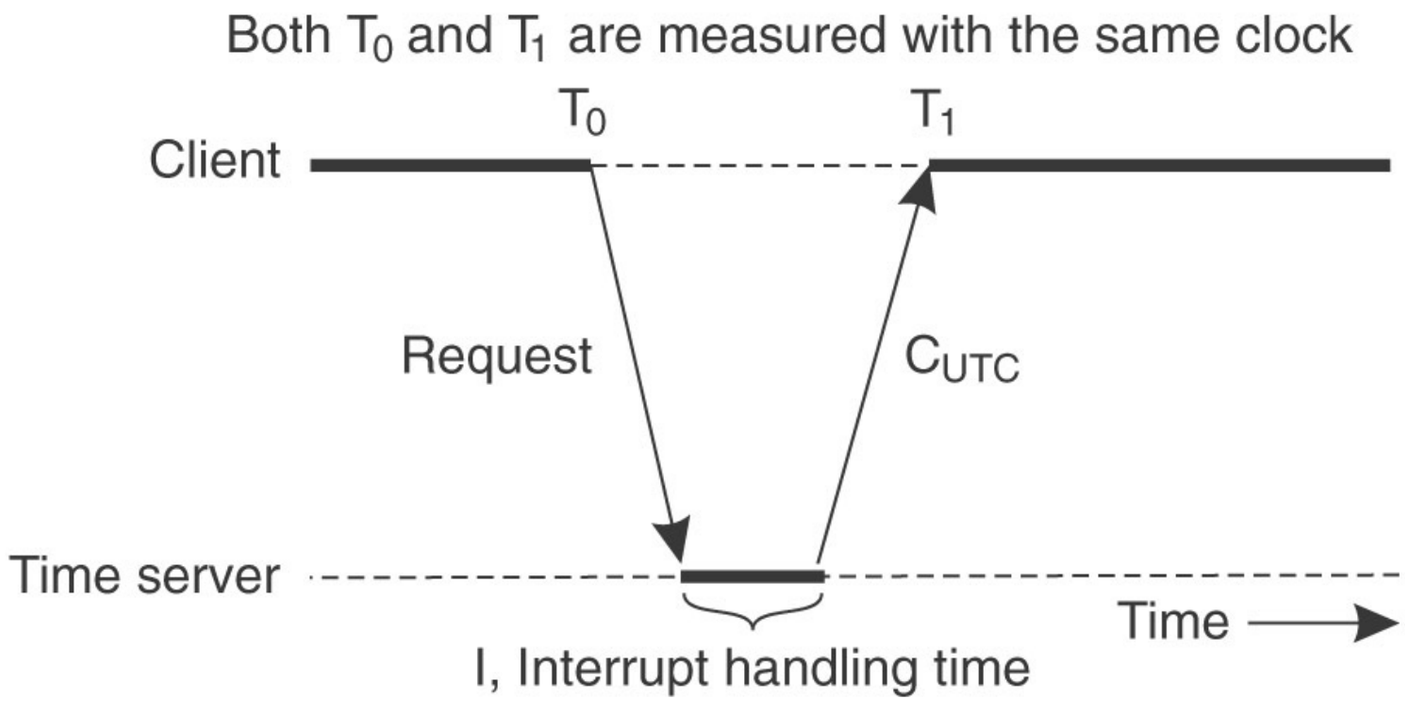
- Multiple CPU with their own clock
- Distributed systems
  - Need to deal with clock skew
- Is there a single notion of time?
  - Astronomical time
  - Atom clock time TAI
    - Leap seconds, UTC

Physical Time



Physical Time

- Clock synchronization algorithms
  - Cristian's algorithm
    - Ask time server for time
    - Determine time



Physical Time

- To determine most likely time given a value send by a time server:
  - Repeat several times
    - Record request send and answer received times (in local time)
  - Record answer received
  - Eliminate outliers
  - Calculate delta
  - Average delta
  - Adjust local time

$$\Delta = t_{\text{time server}} - (t_{\text{received}} + t_{\text{sent}})/2$$

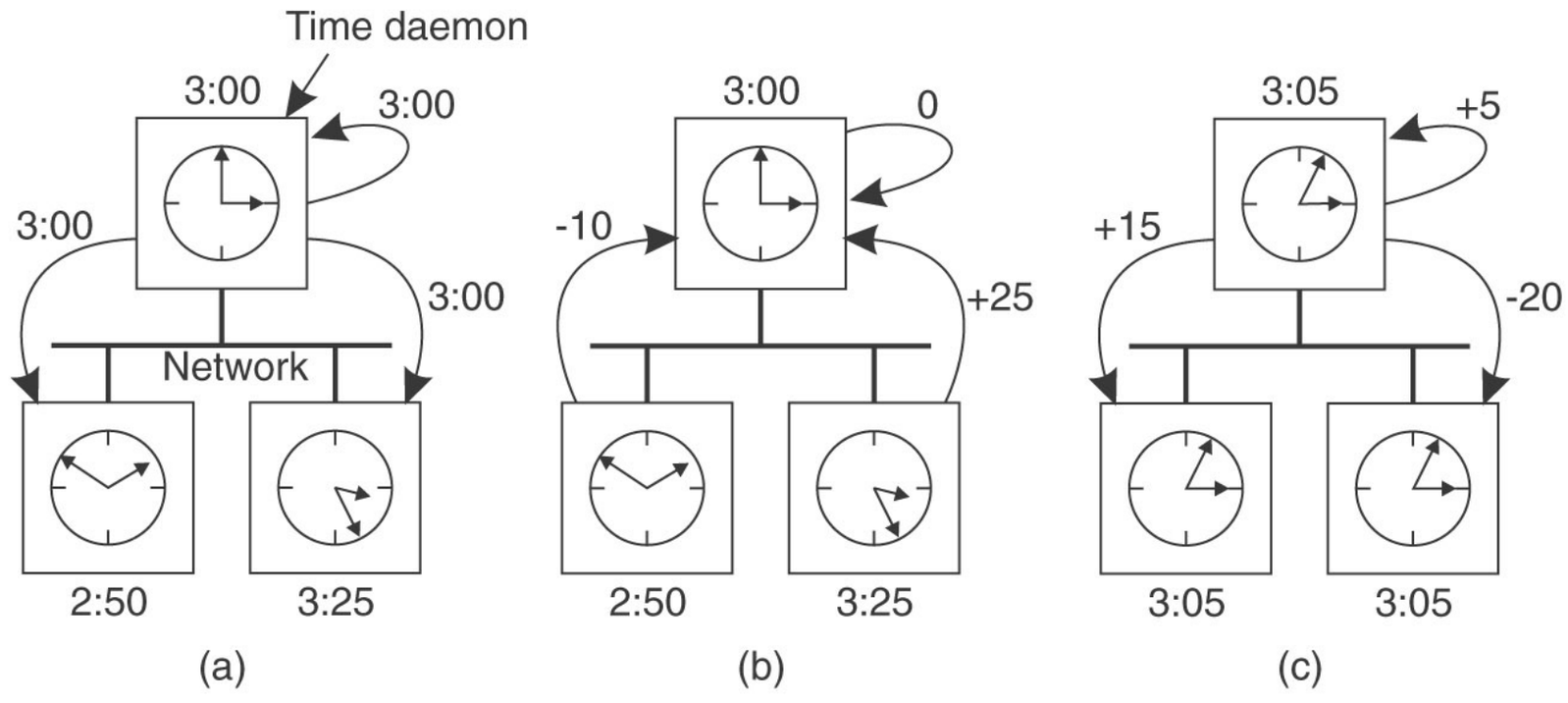
Physical Time

- Group quiz
  - Calculate clock adjustment

sent	received	value
300	350	410
450	495	555
600	750	620
800	845	915
1000	1055	1135
1200	1400	1590

Physical Time

- Berkeley Algorithm
  - Time daemon polls all machines asking for their time
  - Calculates average time
  - Tells all other machines how to adjust



Logical Clocks

- Absolute time is rarely used
  - Exceptions: time outs
- For distributed protocols:
  - Need Logical Time
  -

Logical Clocks

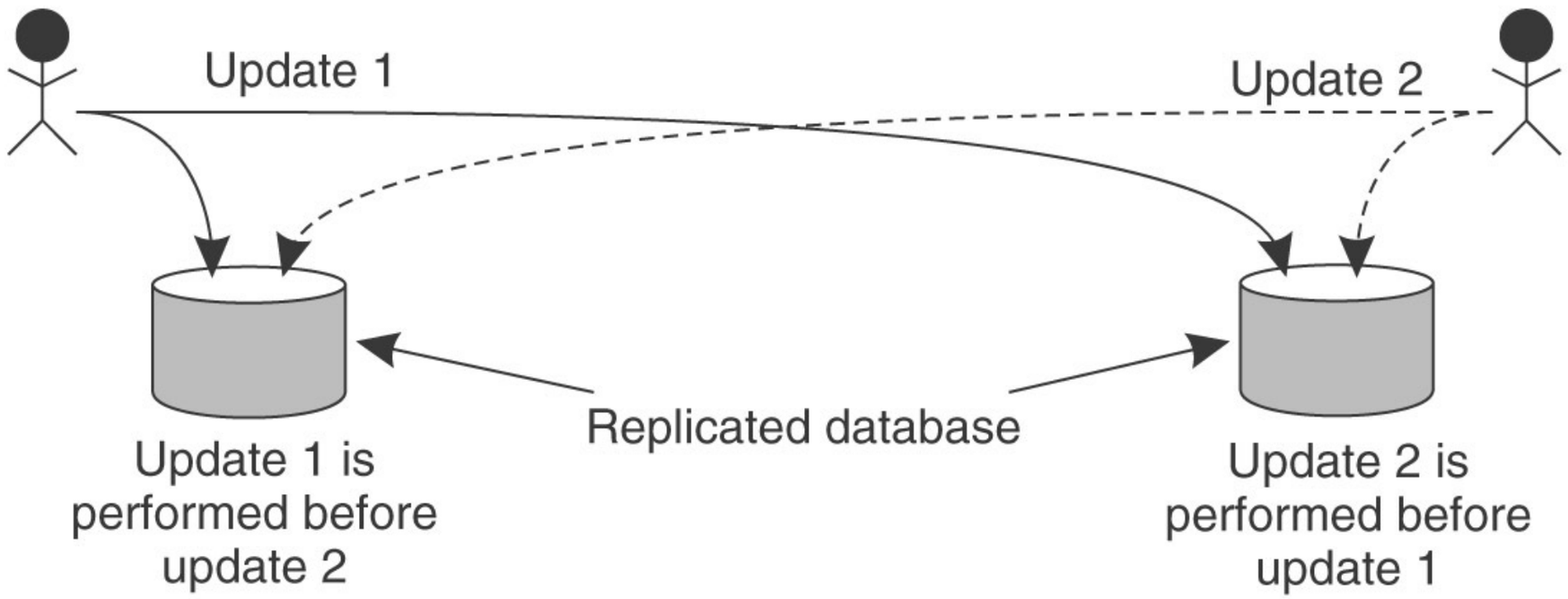
- Lamport time stamps
  - “Happens before” relationship between events:
    - a <* b
  - Axioms:
    - a, b events in the same process, and a happens before b, then a<*b
    - If
      - a — message being sent
      - b — message being received
    - then
      - a <* b

Logical Clocks

- Each system maintains its own logical time
  - Each local event advances the logical time by (at least one tick)
  - Each message is time-stamped
    - When a message is received, set local time to
      - MAX(local_time + 1, time_stamp + 1)
- Properties
  - Local events have different times

Logical Clocks

- Replica management
  - All replicas need to see the same sequence of updates



Logical Clocks

- **Totally Ordered Multicast**
  - Multicast in which all messages are delivered in the same order to receivers
    - Group of processes multicasting to each other
    - Each message is time-stamped
    - Messages are received in order sent
    - Messages are sent to everyone, including the sender
    - Messages are put in local queues ordered by timestamp
    - Receiver multicast acknowledgments to the other processes
  - Lamport clock algorithm assures that all messages have different timestamps
  - All processes eventually have the same messages in their queue

Logical Clocks

- Individual quiz:
  - Assume C1 and C2 send out update messages at the same time to replica servers R1 and R2

Vector Timestamps

- Lamport timestamps do not capture causality
- Vector timestamps better capture causality

$$V_i = [V_i[1], V_i[1], \ldots V_i[n]]$$

  - Number of events that have occurred at process Pi:

$$V_i[i]$$

    - Pi knows that *k* events have occurred at Pj if

$$V_i[j] = k$$

Vector Timestamps

- Process Pi increments Vi[i] whenever something local happens
- Process Pj receives message with Vj
  - Sets Vj[k] = MAX(Vi[k], Vj[k])

Vector Timestamps

- Bulletin board with several threads
  - Local events are sending of messages
  - Maintainer of the bulletin board only posts messages if it knows that all casually anterior messages have been received