Last Class

- Naming
  - Distributed naming
  - DNS
  - LDAP
- Clock synchronization

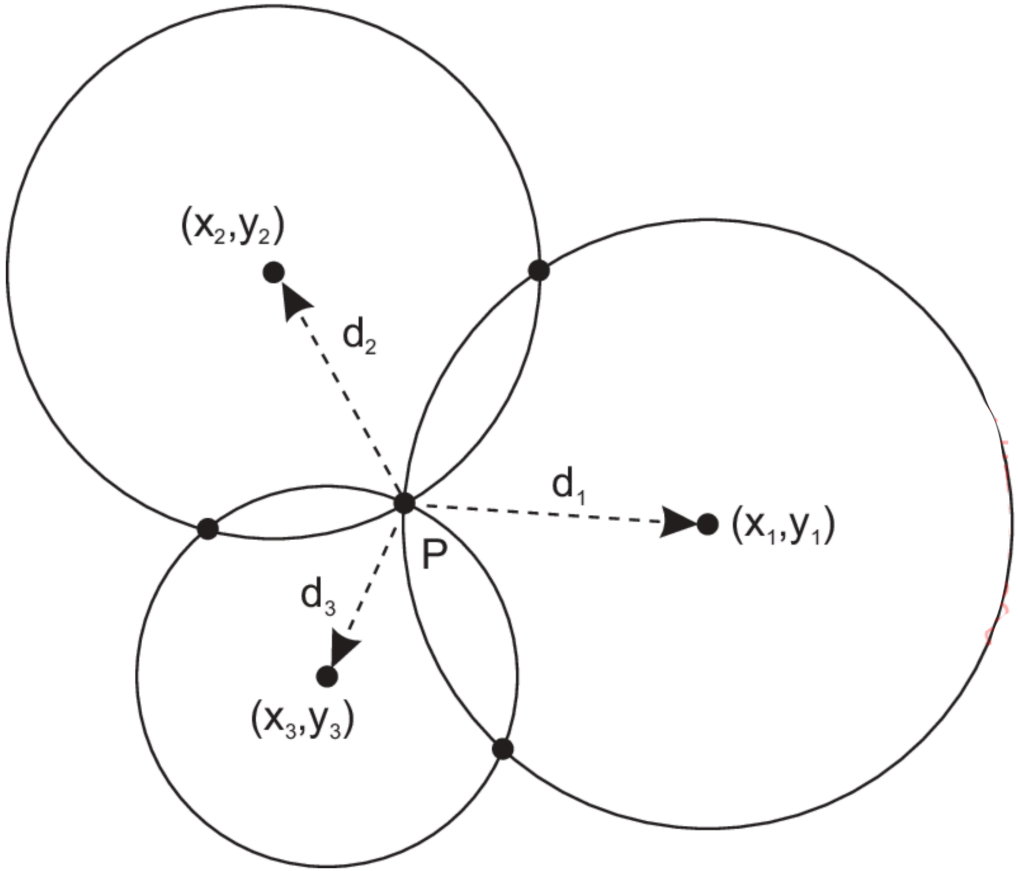
Today: Classical Problems in Distributed Systems

- Wireless clock synchronization, logical clocks

Next few classes:

- Leader election
- Mutual exclusion
- Distributed transactions
- Deadlock detection
- CAP Theorem

Global Positioning System



- Computing a position in a two-dimensional space.

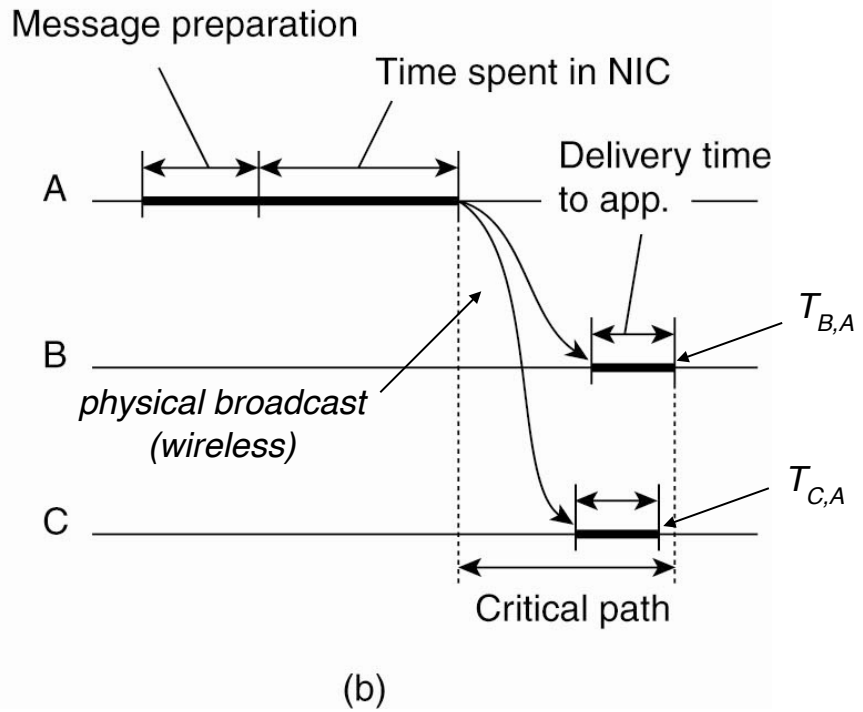
Global Positioning System

- Real world facts that complicate GPS
  - It takes a while before data on a satellite's position reaches the receiver.
  - The receiver's clock is generally not in synch with that of a satellite.

GPS Basics

- D_r – deviation of receiver from actual time
- Beacon from satellite i with timestamp T_i received at time T_{now}
  - Delay $D_i = (T_{now} - T_i) + D_r$
  - Distance $d_i = c\,(T_{now} - T_i)$
  - Also $d_i = \text{sqrt}[\,(x_i - x_r)^2 + (y_i - y_r)^2 + (z_i - z_r)^2\,]$
- Four unknowns, need 4 satellites

Clock Synchronization in Wireless Networks



- Reference broadcast sync (RBS): *receivers* synchronize with one another using RB server *A* based on the time of message delivery
  - Physical clock offset between two receivers *B* and *C* = $T_{B,A}$ - $T_{C,A}$
  - Local physical time at server does not matter. Server-side delays (message preparation, NIC time) are equal for both receivers.
  - The critical path length is approx the same for both receivers

Logical Clocks

- For many problems, internal consistency of clocks is important
  - Absolute time is less important
  - Use *logical* clocks
- Key idea:
  - Clock synchronization need not be absolute
  - If two machines do not interact, no need to synchronize them
  - More importantly, processes need to agree on the *order* in which events occur rather than the *time* at which they occurred

Event Ordering

- *Problem:* define a total ordering of all events that occur in a system
- Events in a single processor machine are totally ordered
- In a distributed system:
  - No global clock, local clocks may be unsynchronized
  - Can not order events on different machines using local times
- Key idea [Lamport]
  - Processes exchange messages
  - Message must be sent before received
  - Send/receive used to order events (and synchronize clocks)

Happened Before Relation

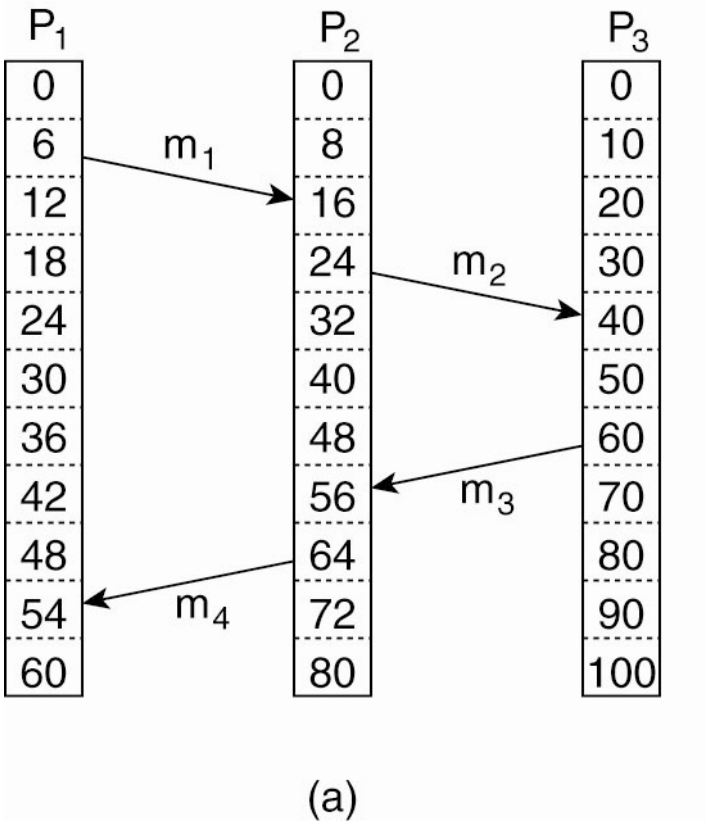
- If *A* and *B* are events in the same process and *A* executed before *B*, then *A* -> *B*

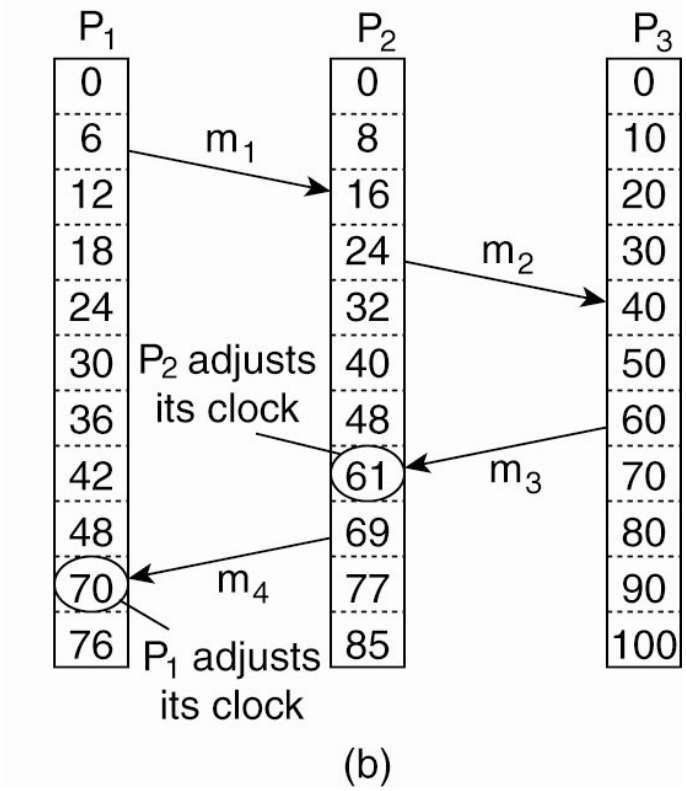
- If A represents sending of a message and B is the receipt of this message, then A -> B
- Relation is transitive:
  - A -> B and B -> C => A -> C
- Relation is undefined across processes that do not exchange messages
  - Partial ordering on events

Event Ordering Using *HB*

- Goal: define the notion of time of an event such that
  - If A-> B then C(A) < C(B)
  - If A and B are concurrent, then C(A) <, = or > C(B)
- Solution:
  - Each processor maintains a logical clock LC_i
  - Whenever an event occurs locally at i, $LC_i = LC_i + 1$
  - When *i* sends message to *j*, piggyback LC_i
  - When *j* receives message from *i*
    - If $LC_j < LC_i$ then $LC_j = LC_i + 1$ else do nothing
  - Claim: this algorithm meets the above goals

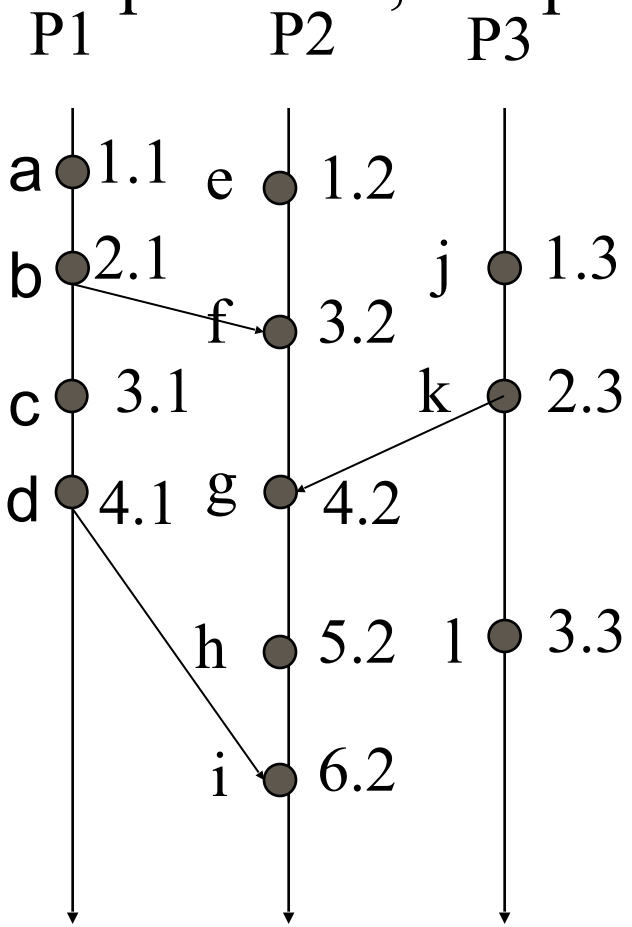
Lamport's Logical Clocks





Total Order

- Create total order by attaching process number to an event. If time stamps match, use process # to order



Example: Totally-Ordered Multicasting