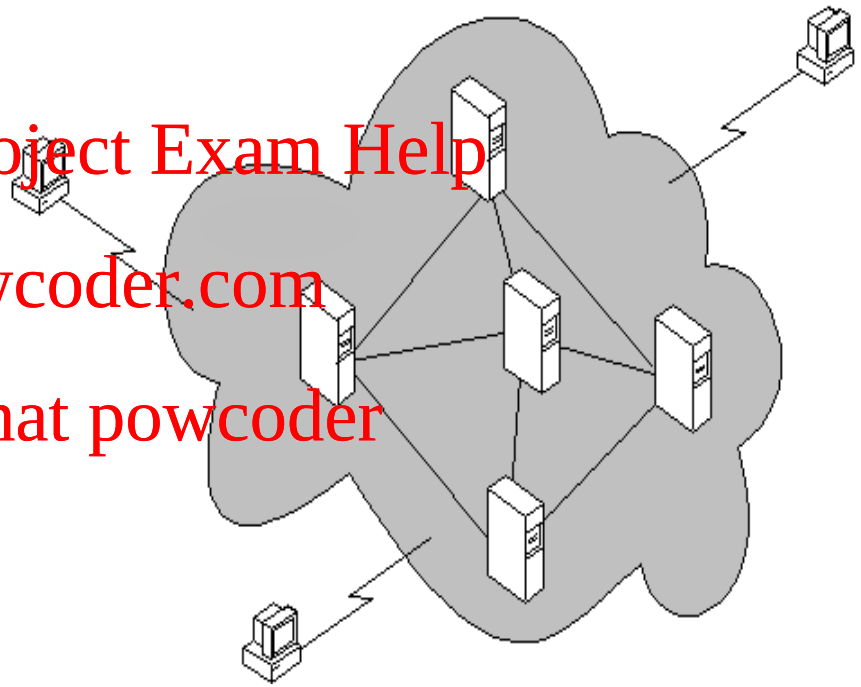
Distributed Systems Foundations

Lecture 1

Main Characteristics of Distributed Systems

- Independent processors, sites, processes
- Message passing
- No shared memory
- No shared clock
- Independent failure modes



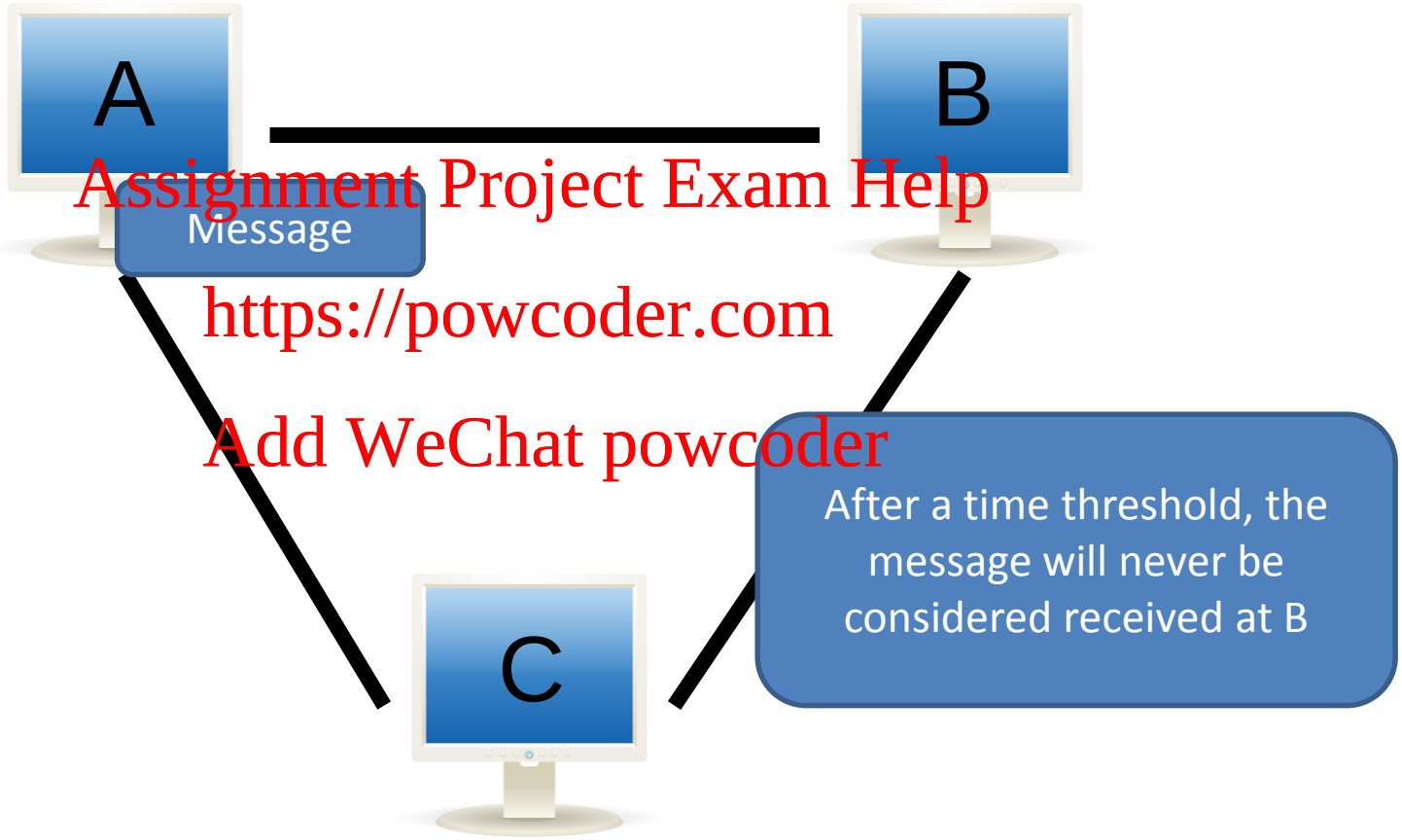
Distributed System Models

- Synchronous System: Known bounds on times for message transmission, processing , bounds on local clock drifts, etc.
  - Can use timeouts.
- Asynchronous System: No known bounds on times for message transmission, processing, bounds on local clock drifts, etc.
  - More realistic, practical, but no timeout.

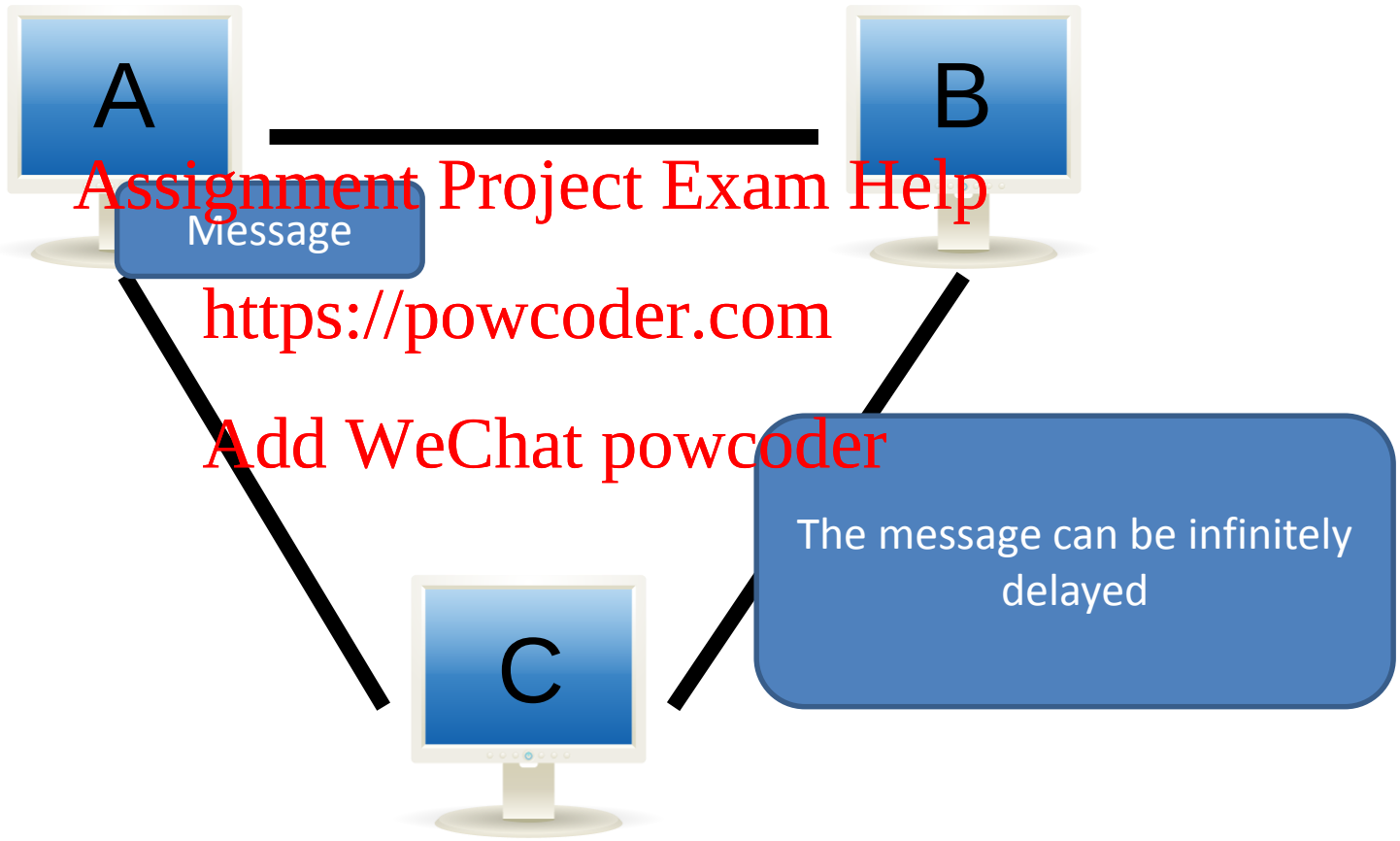
Distributed System Models

- Synchronous System: Known bounds on times for message transmission, processing , bounds on local clock drifts, etc.
  - Can use timeouts.
- Asynchronous System: No known bounds on times for message transmission, processing, bounds on local clock drifts, etc.
  - More realistic, practical, but no timeout.

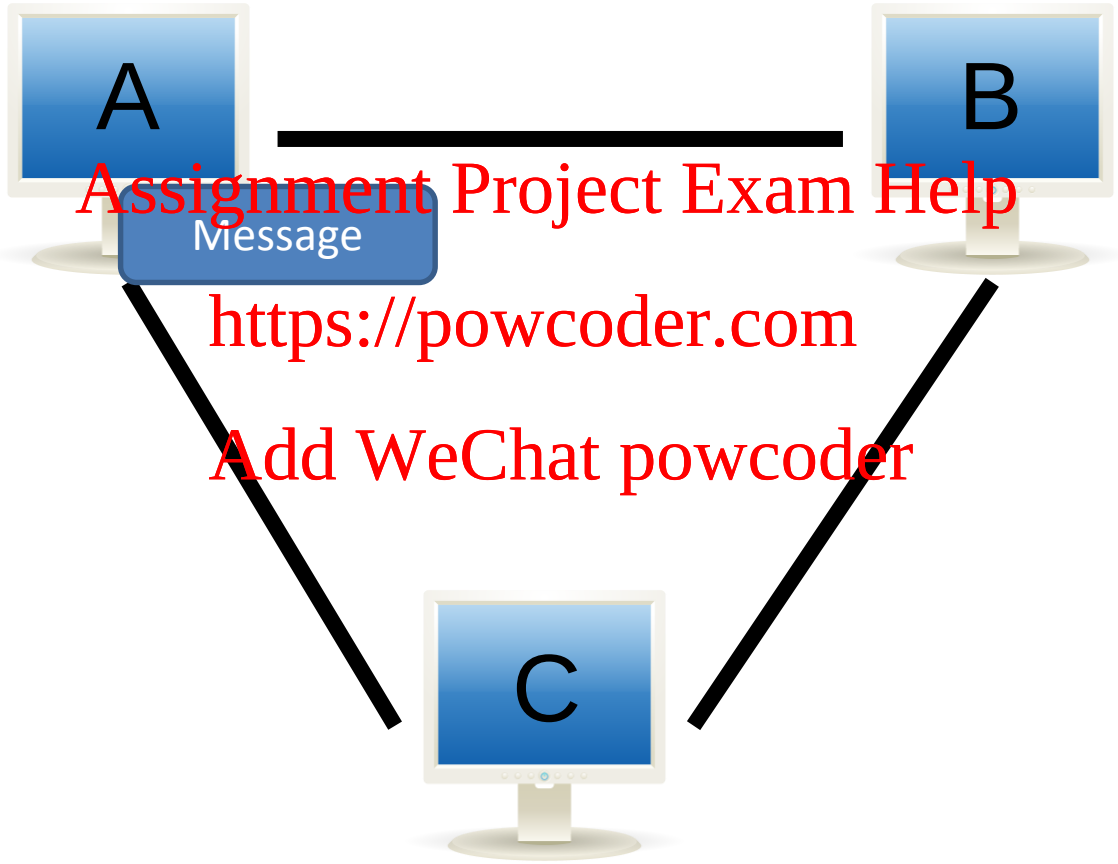
Synchronous model example



Asynchronous model example



Data link layer in a distributed system



Goal: Send a message from A to B

Data Link Layer Example

- Assumptions:
  - Infinite buffer
  - No message loss
- Solution?

Data Link Layer Example

- Assumptions:
  - Finite buffer
  - No message loss

- Solution:
  - Synchronous System?
  - Asynchronous System?

Data Link Layer Example

- Assumptions:
  - Finite buffer
  - Message loss
- Solution?

Data Link Layer Lessons

What are the assumptions?

CAUSALITY AND TIME

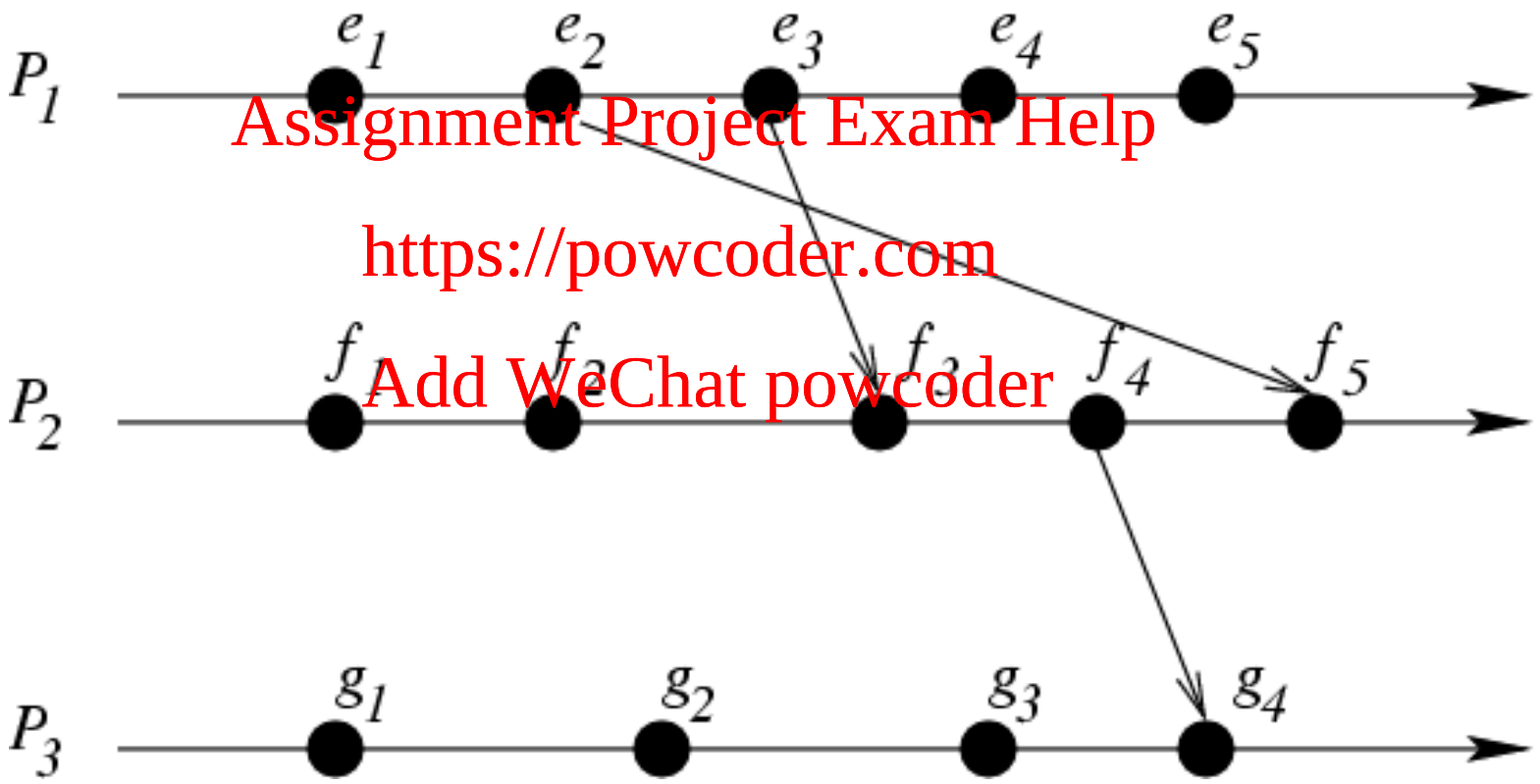
What is a Distributed System?

- A simple model of a distributed system proposed by Lamport in a landmark 1978 paper:
- "Time, Clocks and the Ordering of Events in a Distributed System" Communications of the ACM

What is a Distributed System?

- A set of processes that communicate using message passing.
- A process is a sequence of events
- 3 kinds of events:
  - Local events
  - Send events
  - Receive events
- Local events on a process for a total order.

Example of a Distributed System



Why do we care about "Time" in a distributed system?

- May need to know the time of day at which some event happens on a specific computer
  - **external clock synchronization**
- May need to know the relative order two events that happened on different computers
  - **internal clock synchronization**

Physical Clocks in Distributed Systems

- Does this work?
  - Synchronize all the clocks to some known high degree of accuracy
  - Measure time relative to local clock to determine order between events

- Well, there are some problems…
  - It's difficult to synchronize the clocks

  - Crystal-based clocks tend to drift over time-count time at different rates, and diverge from each other

    - Physical variations in the crystals, temperature variations, etc.
    - Drift is small, but adds up over time
    - For quartz crystal time, typical drift rate is about one second every 10^6 seconds=11.6days
    - Best atomic clocks drift one second in 10^{13} seconds = 300,000 years

Logical Clocks

- Idea — abandon idea of physical time

- For many purposes, it is sufficient to know the **order** in which events occurred

- Lamport (1978) — introduce logical *time*, to provide consistent event ordering

ORDERING EVENTS

- Event ordering linked with **causality**:
  - Saying that event *a* happened before event *b* is same as saying that event *a* could have affected the outcome of event *b*
  - If events *a* and *b* happen on processes that do not exchange any data, their exact ordering is not important

Happens Before or Causal Order on Events

- Event *e happens before (causally precedes)* event *f*, denoted $e \rightarrow f$ if:
  1. The same process executes *e* before *f* ; or
  2. *e* is send(*m*) and *f* is receive(*m*); or
  3. Exists *h* so that $e \rightarrow h$ and $h \rightarrow f$
- We define *concurrent*, $e \,||\, f$, as:

$$\neg(e \rightarrow f \ \vee f \rightarrow e)$$

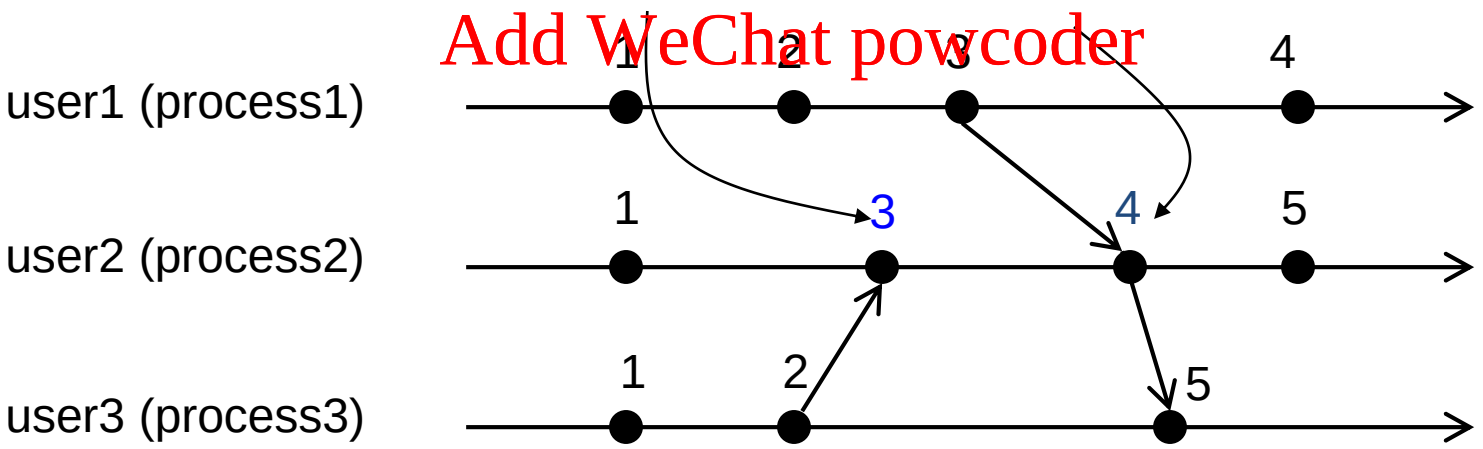
Lamport Logical Clocks

- Assign "clock" value to each event such that
  - if $a \rightarrow b$ then **clock(a) < clock(b)**
- Assign each process a "clock" "counter".
  - Clock is incremented between any two events in the same process
  - Each message carries the sender's clock value
- When a message arrives set local clock to:
  - max(local value, message timestamp) + 1

– This clock forms a partial order.

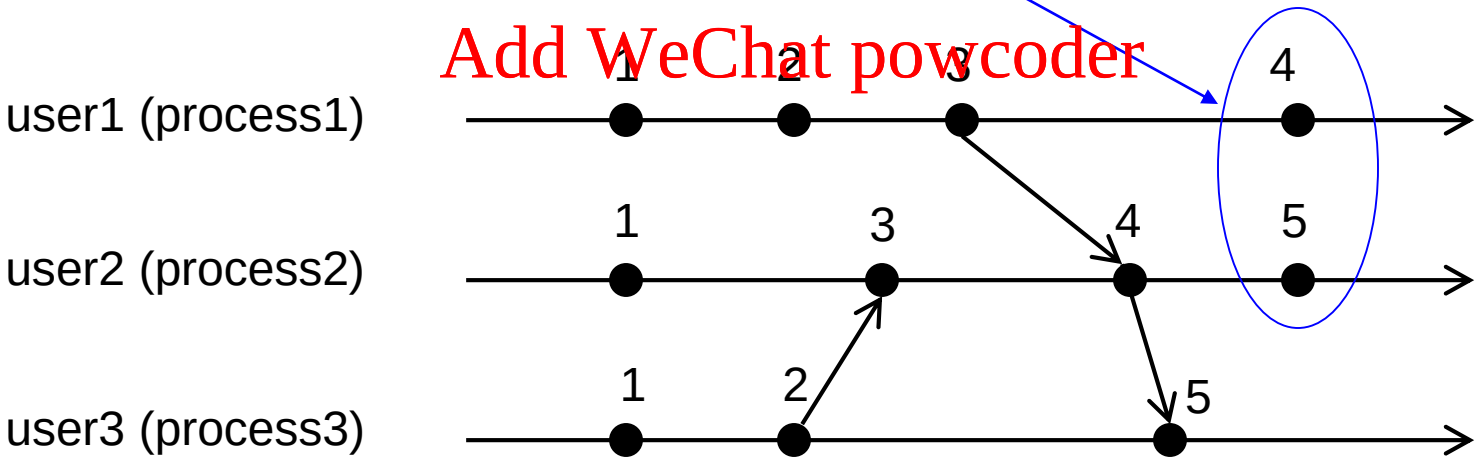
Logical Clocks

- Each event has a single integer as its logical clock
  - Each process has a local counter C
  - Increment C between two events
  - At each "send", logical clock value V is attached.
  - At each "receive", C = max(C, V) + 1

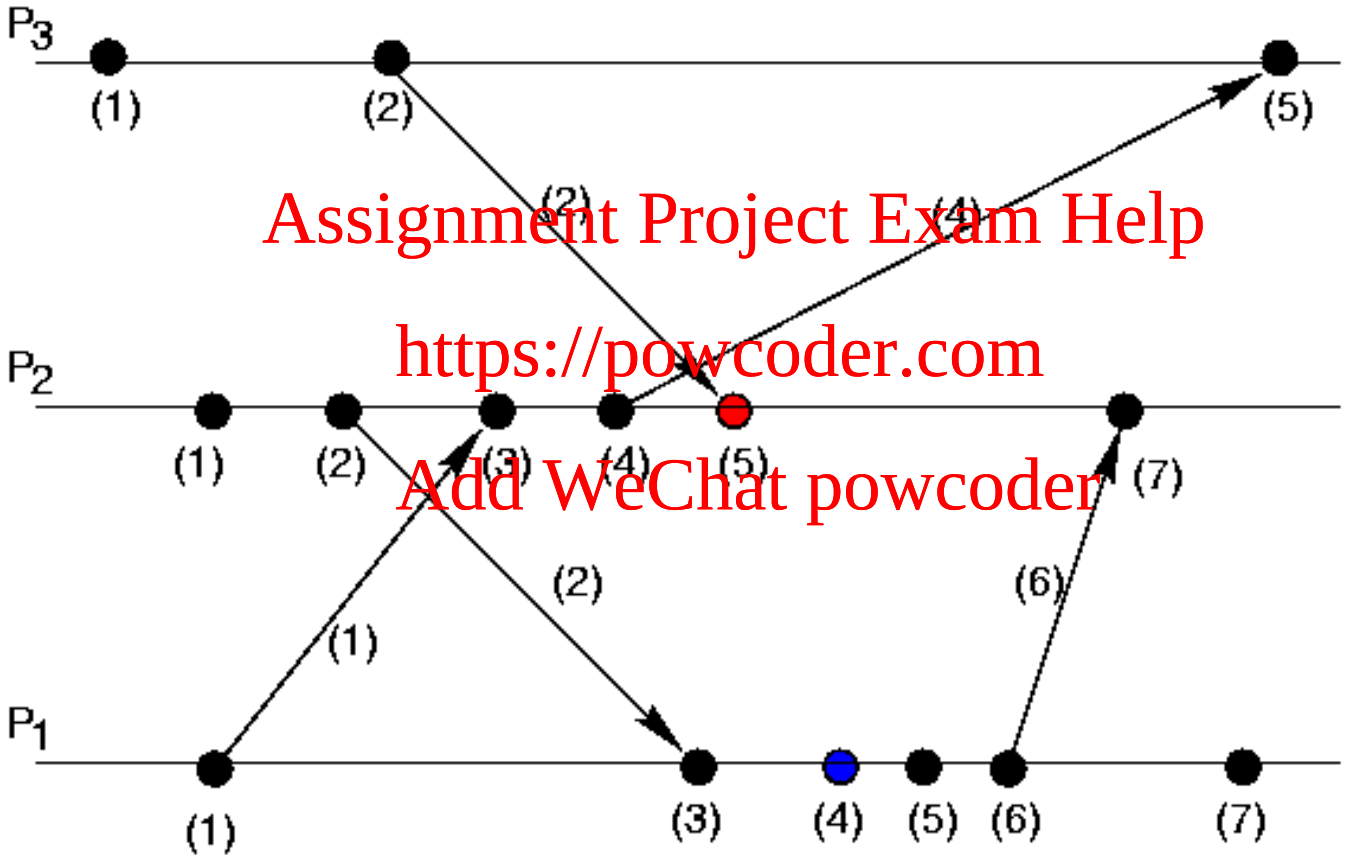


Logical Clock Properties

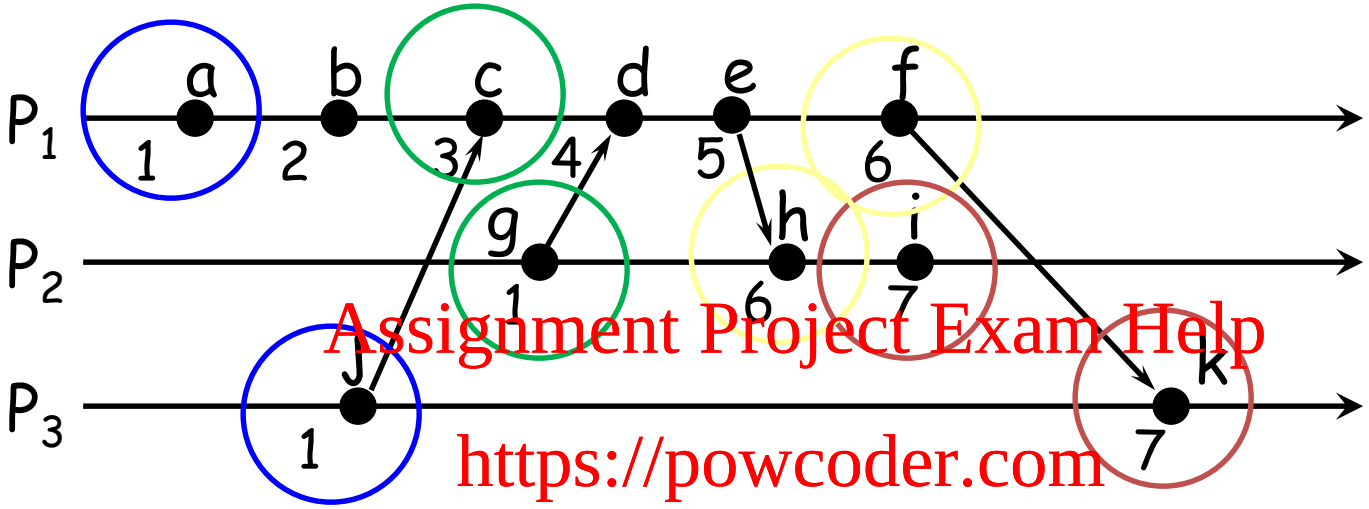
- Theorem:
  - Event s happens before t $\Rightarrow$ logical clock value of s is smaller than the logical clock value of t.
- The reverse may not be true.



Example of a Logical Clock



Problem: Identical timestamps



a→b, b→c, … : local events sequenced

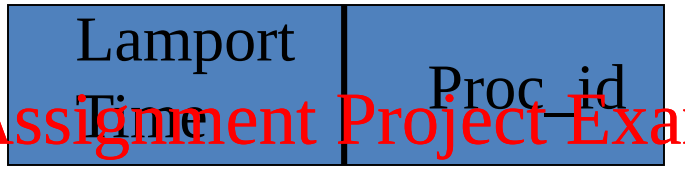
j→c, g→d , f→k, … : Lamport imposes a *send*→*receive* relationship

Concurrent events (e.g., a & j) *may* have the same timestamp

… or not (e.g., c & g)

Total Order Lamport Clock

- To extend the partial order to total order: use process ids to break ties



- Global timestamps:
  - (Ta, Pa): Ta is local Lamport logical timestamp
    Pa is process id.
  - (Ta,Pa) < (Tb,Pb) iff
    - (Ta < Tb) or ((Ta = Tb) and (Pa < Pb))
- Total order is consistent with partial order.
- Total order useful in many applications.

Unique (totally ordered) timestamps