Synchronizing Physical Clock Logical Clock & Vector Clock

Y. V. Lokeswari

ASP / CSE

Overview

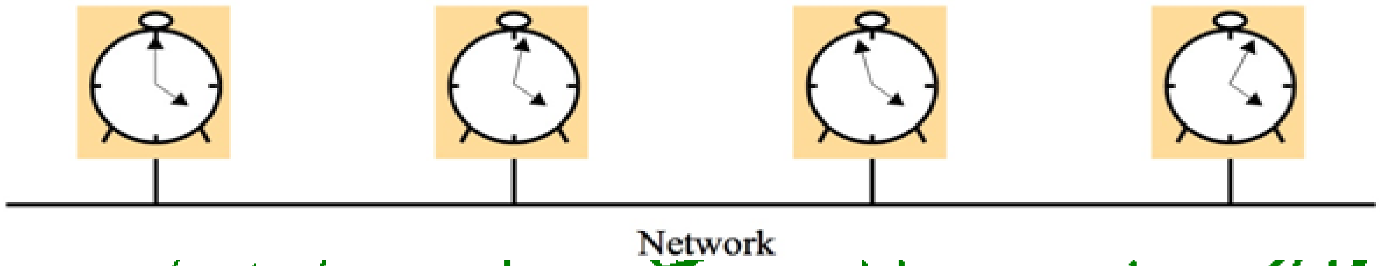
- Physical Clocks
- Synchronizing Physical Clock (Algorithms)
- Problems with Physical Clock
- Lamport's Logical Clock
- Problems with Logical Clock
- Vector Clock
- Drawbacks of Vector Clocks
- Applications of Vector Clocks

Clock Synchronization

- Temporal ordering of events produced by concurrent processes
- Synchronization between senders and receivers of messages
- Serialization of concurrent access for shared objects
- **Physical Clock**: It is the internal clock present in a computer. (Time of a day)
- **Logical Clock**: keeps track of event ordering among related (causal) events.

Clock Synchronization

- Getting two systems to agree on time
  - Two clocks hardly ever agree
  - **Quartz oscillators oscillate at slightly different frequencies**
- **Clock Drift :** Clocks tick at different rates. Create ever-widening gap in perceived time
  - Clock drift in ordinary clocks based on quartz crystal is 10^-6 seconds.
  - This creates a difference of 1 sec for every 11.6 days (1,000,000 sec)
  - Clock drift of high precision clock is 10^-7 to 10^-8
- **Clock Skew :** Difference between two clocks at one point in time.



Clock Synchronization

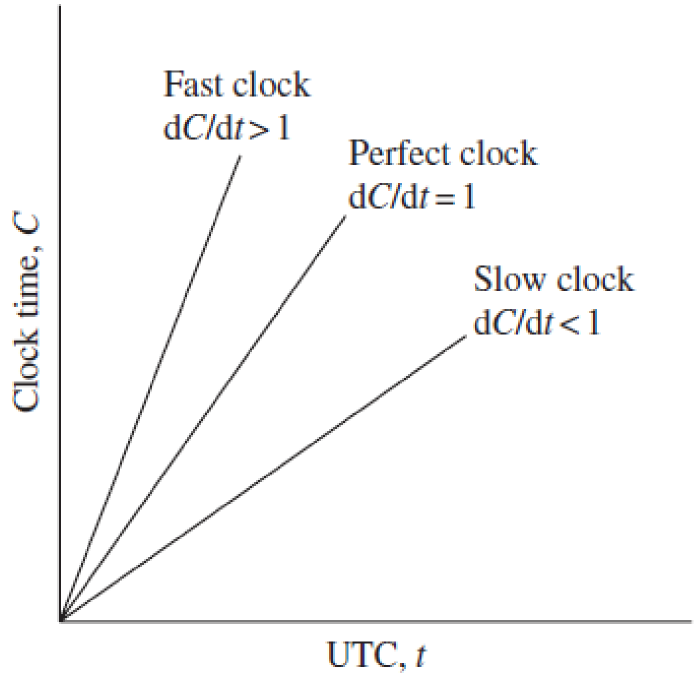
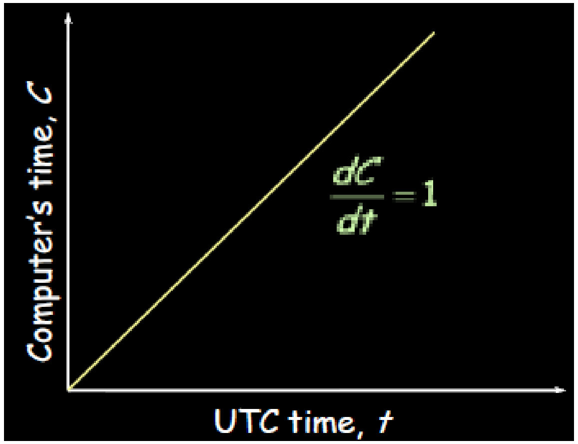


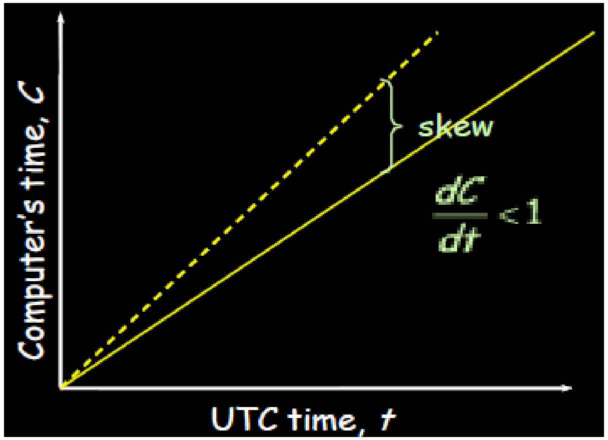
Figure 3.8 The behavior of fast, slow, and perfect clocks with respect to UTC.

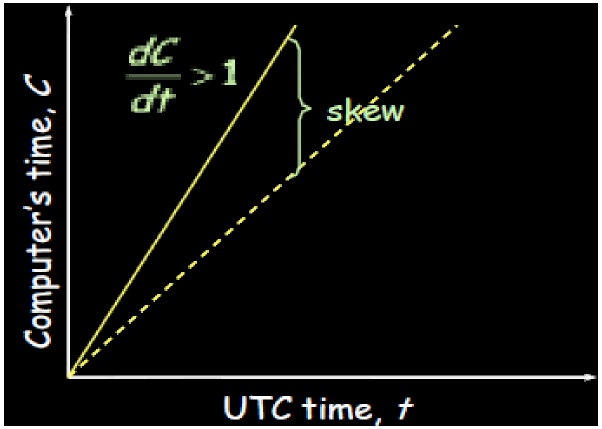
Clock Synchronization

- Dealing with Clock Drift: Go for gradual clock correction

UTC : Universal Coordinated Time







If slow:

Make clock run faster until it Synchronizes

If fast:

Make clock run slower until it Synchronizes

Clock Synchronization

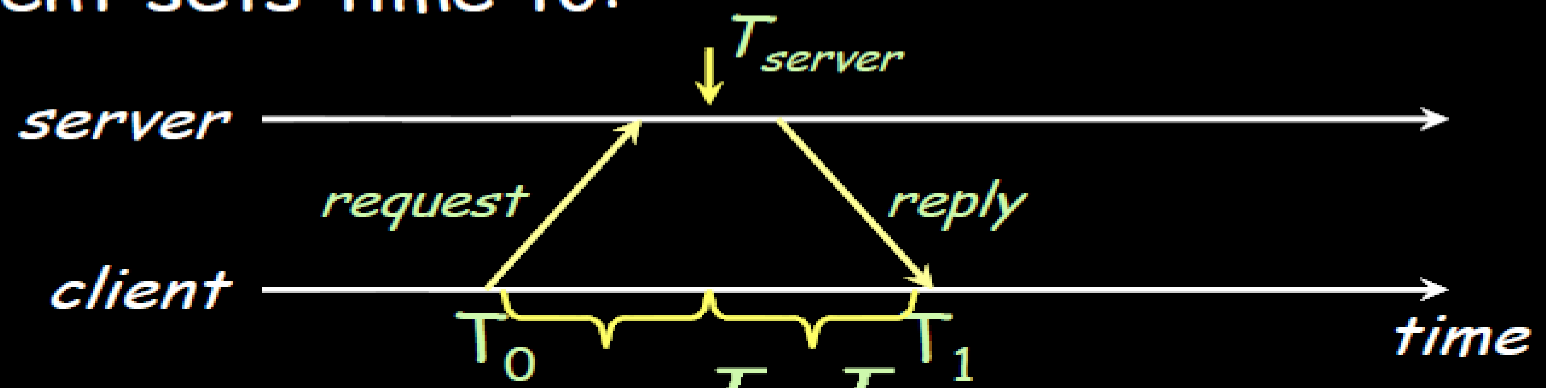
- **External Synchronization** : Clock synchronizes to correct time from external timing elements like Radio / Satellite time.
- **Internal Synchronization** : Clock synchronizes to correct time by getting timings from other computers.
- 3 - Algorithms for Synchronizing Physical Clock
  1. **Cristian's Algorithm**
  2. **Berkeley Algorithm**
  3. **Network Time Protocol (NTP)**

Clock Synchronization

1. Cristian's Algorithm

(T1-T0)/2 is round-trip time

Client sets time to:



$$\frac{T_1 - T_0}{2}$$ = estimated overhead in each direction

$$T_{new} = T_{server} + \frac{T_1 - T_0}{2}$$

Clock Synchronization

1. Cristian's Algorithm – Example

- Send request at 5:08:15.100 (T0)
- Receive response at 5:08:15.900 (T1)
- Response contains 5:09:25.300 (Tserver)
- Elapsed time is T1 -T0

  5:08:15.900 - 5:08:15.100 = 800 msec
- Best guess: timestamp was generated

  400 msec ago (800/2)
- Set time to Tserver+ elapsed time

  5:09:25.300 + 400 = 5:09.25.700

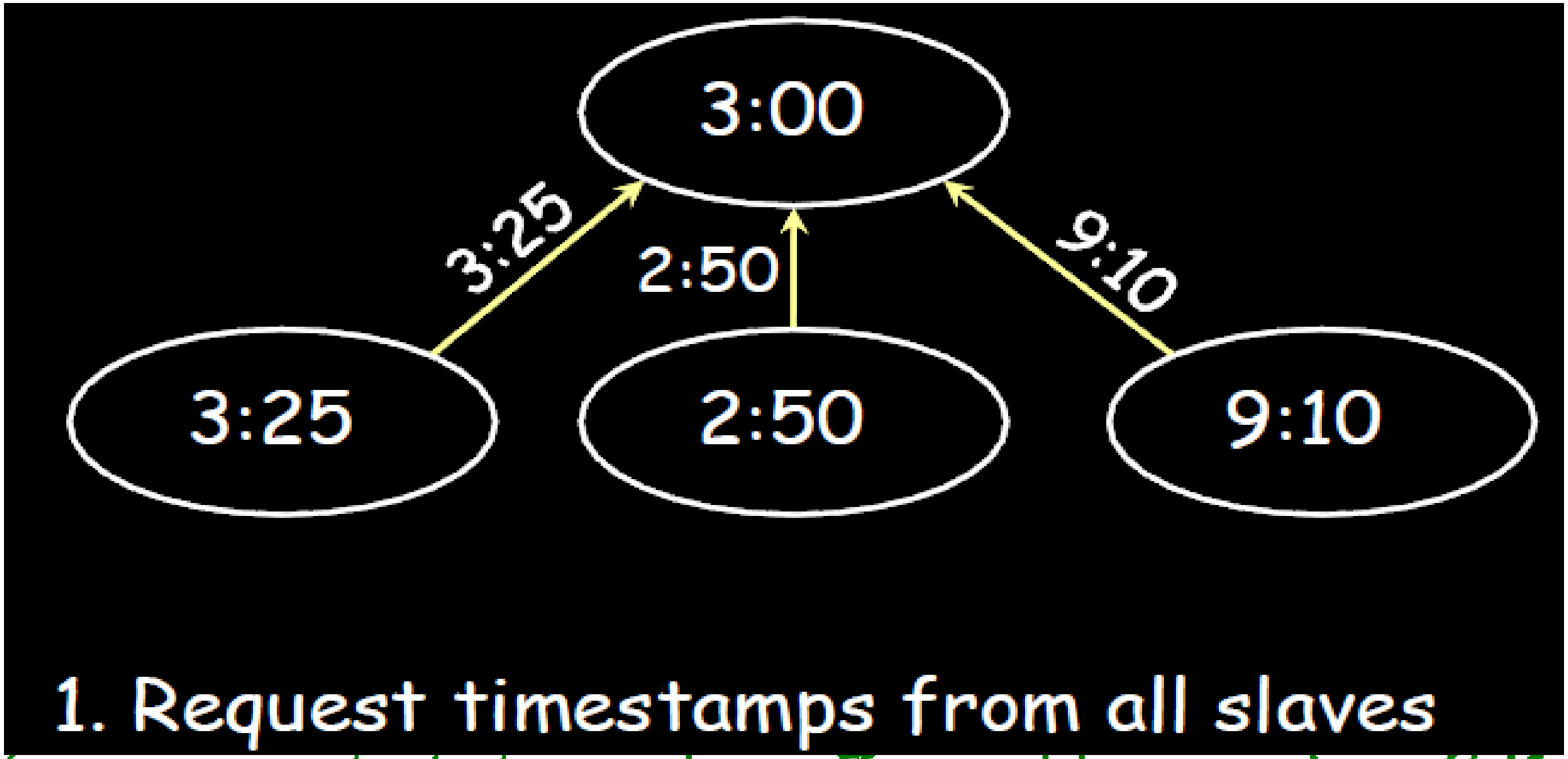
Clock Synchronization

2. Berkeley Algorithm

- Machines run **time dæmon** Process that implements protocol
- One machine is elected (or designated) as the server (**master)** Others are **slaves**
- Master polls each machine periodically and ask each machine for time
- Can use Cristian's algorithm to compensate for network latency
- When results are received, master computes average of times - Including master's time
- **Hope**: Average cancels out individual clock's tendencies to run fast or slow (clock drift)
- Send offset by which each clock needs adjustment to each slave.
- Algorithm has provisions for ignoring readings from clocks whose skew is too great
- Compute a **fault-tolerant average**
- If master fails - any slave can take over

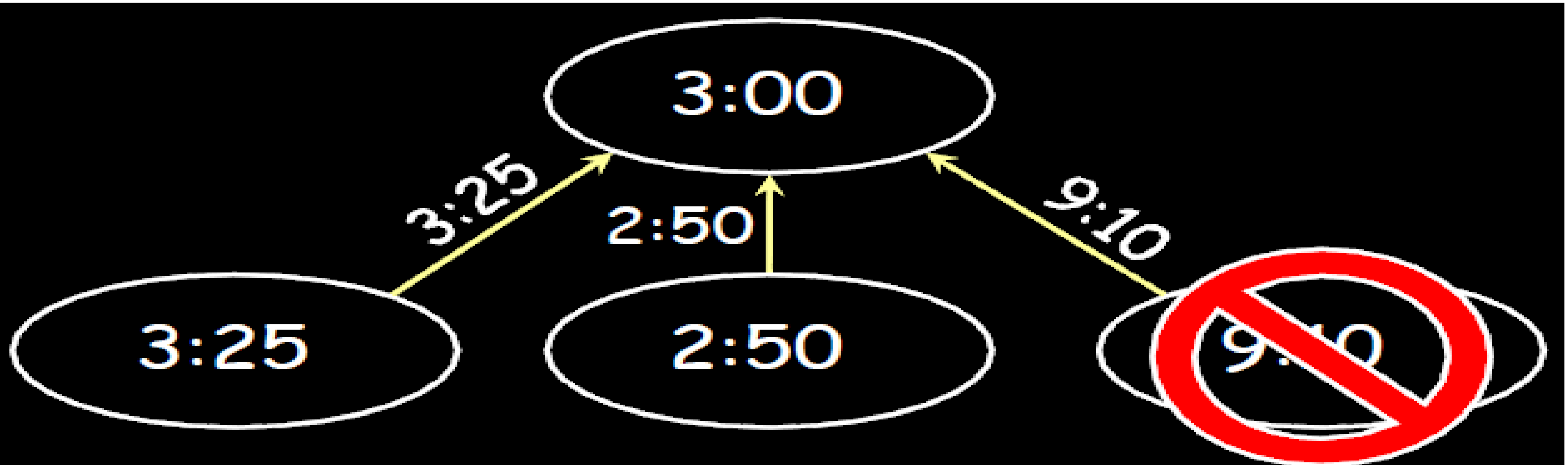
Clock Synchronization

2. Berkeley Algorithm Example



Clock Synchronization

2. Berkeley Algorithm Example

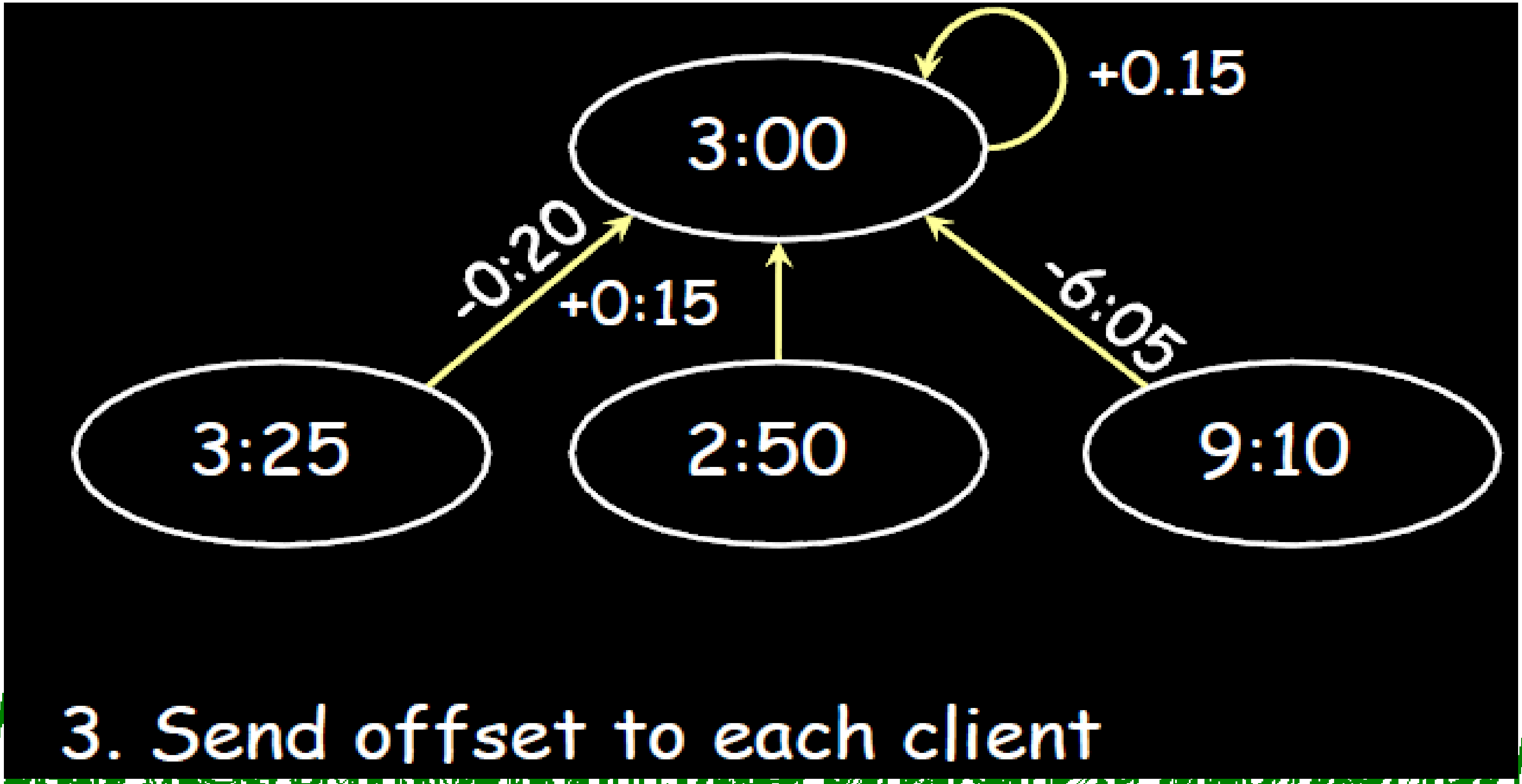


2. Compute fault-tolerant average:

$$\frac{3{:}25 + 2{:}50 + 3{:}00}{3} = 3{:}05$$

Clock Synchronization

2. Berkeley Algorithm Example



Clock Synchronization

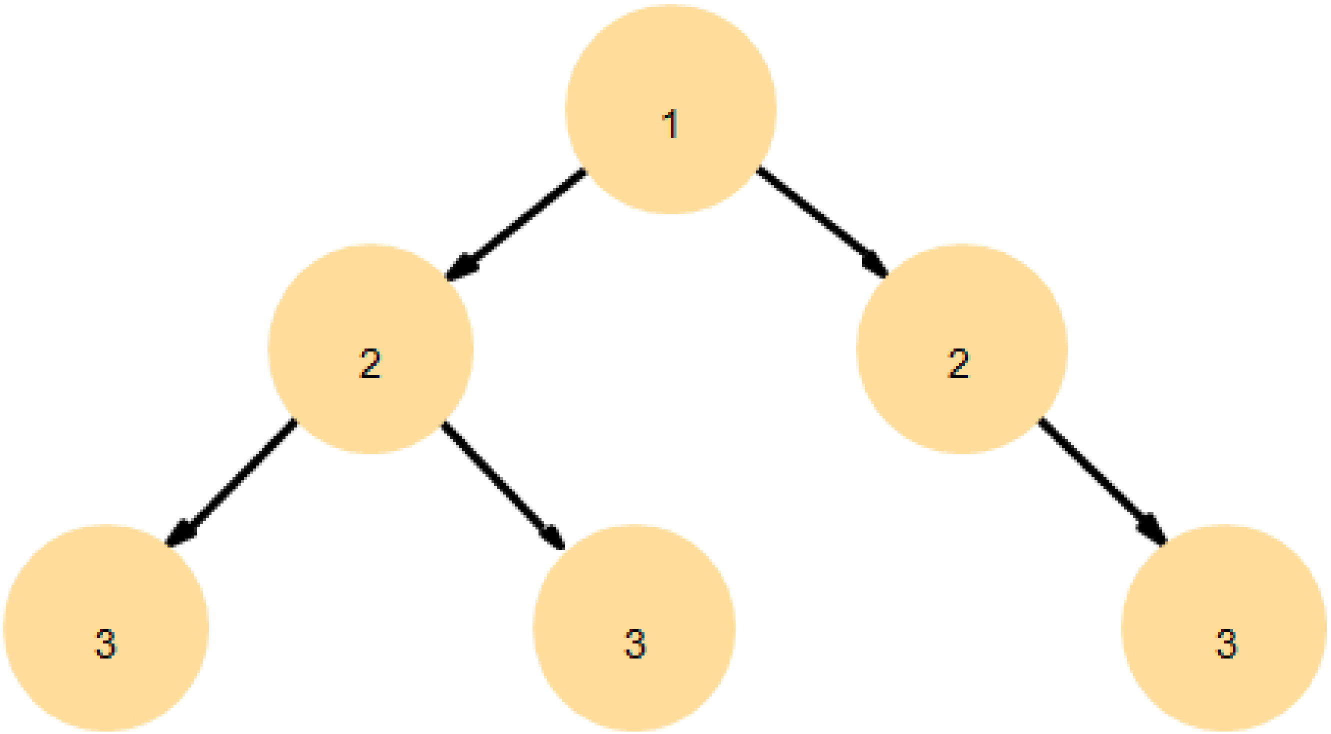
3. Network Time Protocol

- Enable clients across Internet to be accurately synchronized to UTC despite message delays
- Primary servers are connected directly to a time source such as a radio clock receiving UTC; secondary servers are synchronized with primary servers.
- The servers are connected in a logical hierarchy called a **synchronization subnet** whose levels are called **strata**.
- Primary servers occupy stratum 1: they are at the root.
- Stratum 2 servers are secondary servers that are synchronized directly with the primary servers;
- Stratum 3 servers are synchronized with stratum 2 servers, and so on.

Clock Synchronization

3. Network Time Protocol

Arrows denote synchronization control, numbers denote strata.



Clock Synchronization

Network Time Protocol

Let a = $T_1 - T_3$ and b = $T_2 - T_4$.

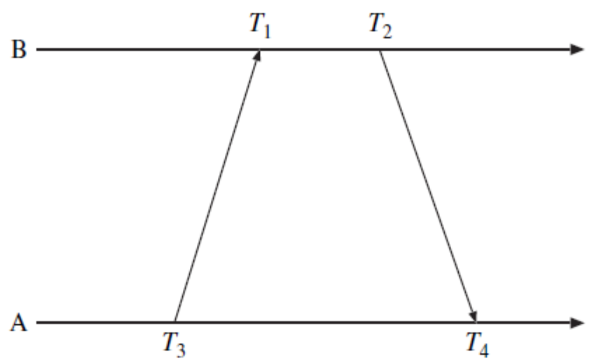


Figure 3.9 Offset and delay estimation [15].

$$\theta = \frac{a+b}{2}, \qquad \delta = a - b.$$

the offset O_i can be estimated as

$$O_i = (T_{i-2} - T_{i-3} + T_{i-1} - T_i)/2.$$

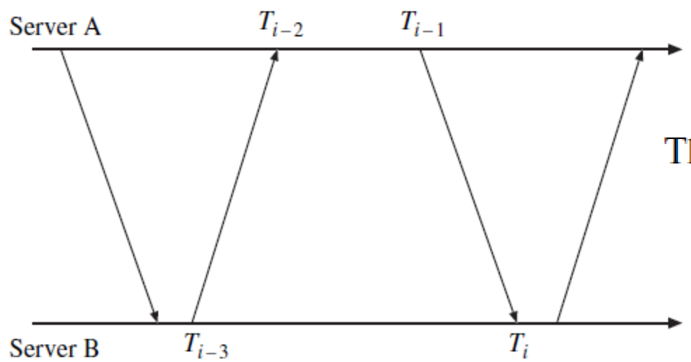


Figure 3.10 Timing diagram for the two servers [15].

The round-trip delay is estimated as

$$D_i = (T_i - T_{i-3}) - (T_{i-1} - T_{i-2}).$$

Clock Synchronization

3. Network Time Protocol (Synchronization Modes)

- **Multicast mode**
  - Every node sends its time to all the other nodes in the LAN
  - For high speed LANs
  - Lower accuracy but efficient
- **Procedure call mode**
  - Similar to Cristian's algorithm
- **Symmetric mode**
  - Intended for master servers
  - Pair of servers exchange messages and retain data to improve synchronization over time

All messages delivered unreliably with UDP

Problems with Physical Clocks

- **Clock Drift :** Clocks tick at different rates. Create ever-widening gap in perceived time
- **Clock Skew :** Difference between two clocks at one point in time.
- For quartz crystal clocks, typical drift rate is about one second every 106 seconds =11.6 days
- Best atomic clocks have drift rate of one second in 1013 seconds = 300,000 years.
- **Quartz clock run at rate of 1.5 microseconds slower for every 35 days.**

Problems with Physical Clocks

8:01:24

8:01:48

Skew = +84 seconds
+84 seconds/35 days
Drift = +2.4 sec/day

Oct 23, 2006
8:00:00

Skew = +108 seconds
+108 seconds/35 days
Drift = +3.1 sec/day

Logical Clocks

Need for Logical Clock

- For many purposes, it is sufficient to know the order in which events occurred.
- Lamport (1978) — introduce logical (*virtual) time,* synchronize *logical clocks.*
- An event may be an instruction execution, may be a function execution, etc.
- Events include message send / receive

Within a single process, or between two processes on the same computer

- The order in which two **events** occur **can** be determined using the **physical clock**

Between two different computers in a distributed system

- The order in which two events occur **cannot** be determined **using local physical clocks**, since those clocks cannot be synchronized perfectly

Happened Before Relation

- Lamport defined the happened before relation (denoted as "→"), which describes a **causal ordering of events:**

1. if a and b are events in the same process, and a occurred before b, then a → b
2. if a is the event of sending a message m in one process, and b is the event of receiving that message m in another process, then a → b
3. if a → b, and b → c, then a → c (i.e., the relation "→" is transitive

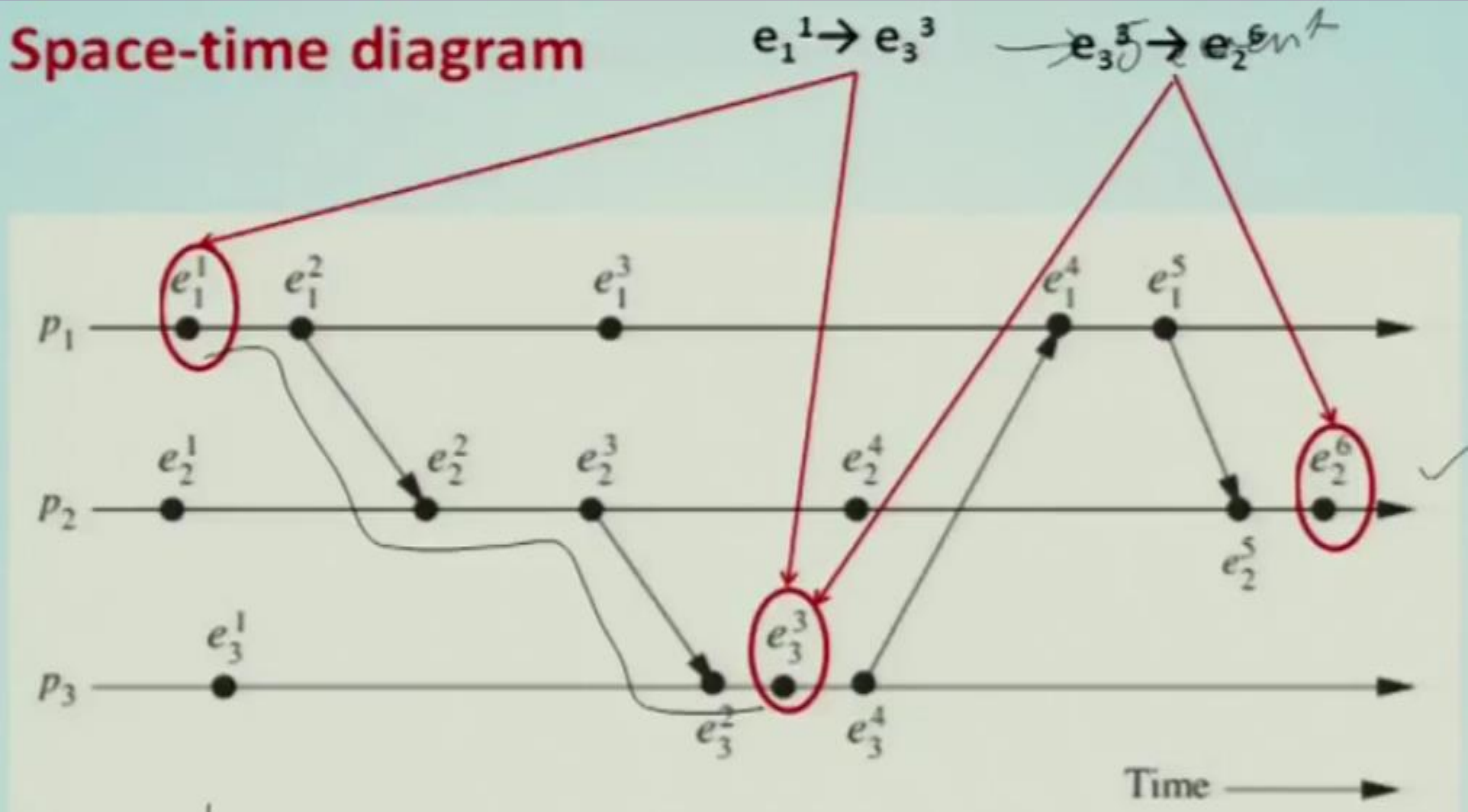
Causality

- Past events influence future events
- This influence among causally related events (those that can be ordered by "→") is referred to **causally affects.**
- If a → b, event **a** causally affects event **b**

Concurrent events

- Two distinct events **a** and **b** are said to be concurrent (denoted "a || b"), if neither a→ b nor b → a
- In other words, concurrent events do not causally affect each other
- For any two events a and b in a system, either: a → b or b → a or a || b

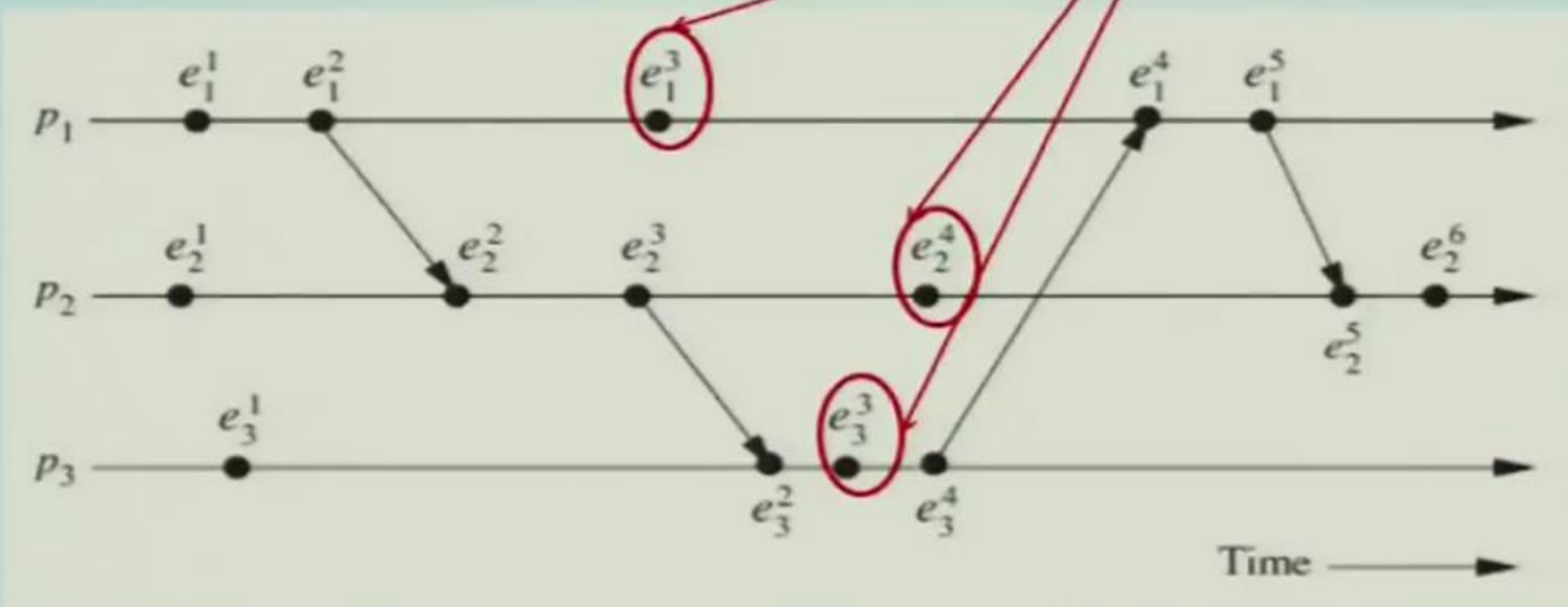
Causal Events



Concurrent Events

Space-time diagram

events in the set $\{e_1^3, e_2^4, e_3^3\}$ are logically concurrent.



Lamport's Logical Clock

- To implement " → " in a distributed system, Lamport (1978) introduced the concept of logical clocks, which captures " → " numerically
- Each process Pi has a logical clock Ci
- Clock Ci can assign a value Ci (a) to any event **a** in process Pi
- The value Ci (a) is called the timestamp of event **a** in process Pi
- The value C(a) is called the timestamp of event **a** in whatever process it occurred.
- The timestamps have no relation to physical time, which leads to the term logical clock.
- The logical clocks assign monotonically increasing timestamps, and can be implemented by simple counters

Lamport's Logical Clock

- **Clock condition: if a $\rightarrow$ b, then $C(a)<C(b)$**
- If event a happens before event b, then the clock value (timestamp) of a should be less than the clock value of b
- Note that we can **not say: if $C(a)<C(b)$, then a $\rightarrow$ b**
- **Correctness conditions (must be satisfied by the logical clocks to meet the clock condition above):**
- [C1] For any two events a and b in the same process Pi ,

  if a happens before b, then $Ci\ (a) < Ci\ (b)$
- [C2] If event a is the event of sending a message m in process Pi , and event b is the event of receiving that same message m in a different process Pk, then $Ci\ (a) < Ck\ (b)$

Lamport's Logical Clock

- **[IR1]** Clock Ci must be incremented between any two successive events in process Pi

  Ci := Ci + d , (d>0) (usually d=1)

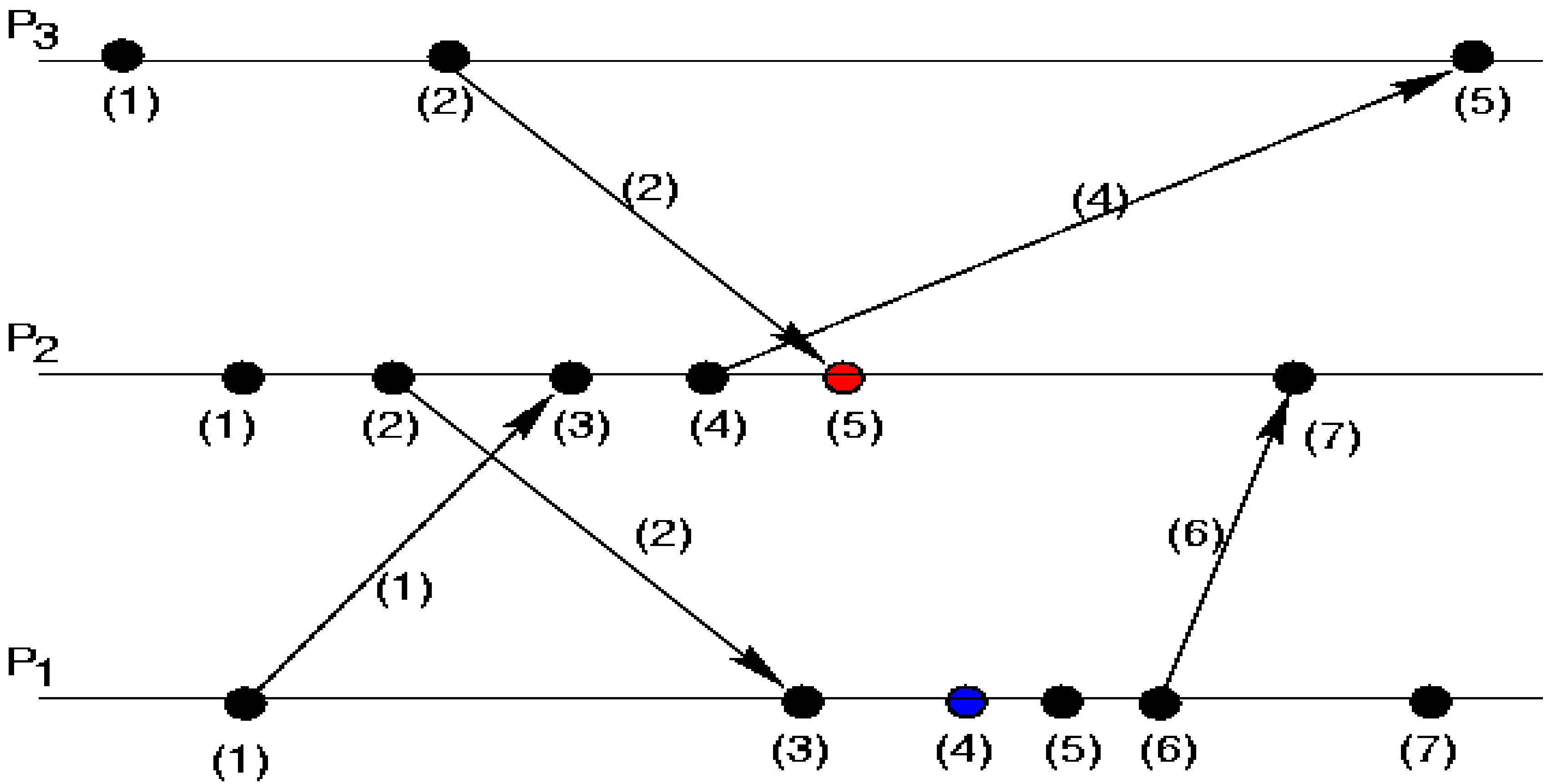
- **[IR2]** If event a is the event of sending a message m in process Pi , then message m is assigned a timestamp Cmsg =Ci (a) When that same message m is received by a different process Pk, Ck is set to a value greater than or equal to its present value, and greater than Cmsg.

  Ck := max(Ck , Cmsg + d) , (d>0) (usually d=1)

Lamport's Logical Clock

IR 1: Ci := Ci + d , (d>0), (usually d=1)

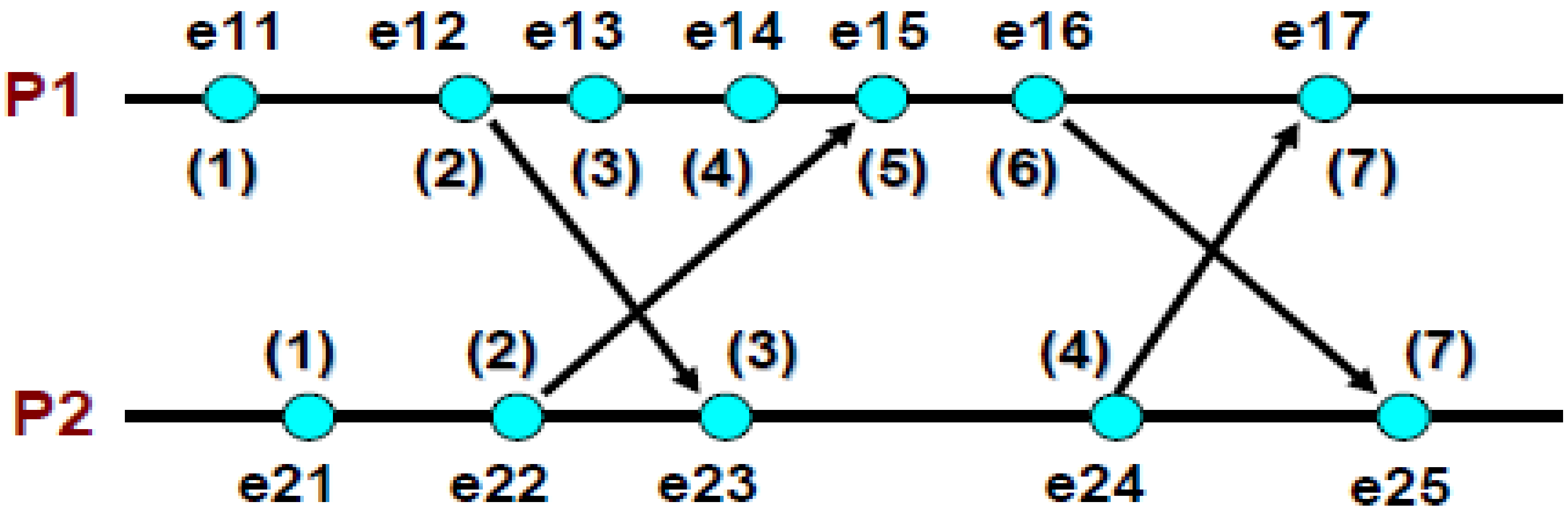
IR2: Ck := max(Ck , Cmsg + d) , (d>0) (Usually d=1)



Lamport's Logical Clock

IR 1: Ci := Ci + d , (d>0) , (usually d=1)

IR2: Ck := max(Ck , Cmsg + d) , (d>0) (Usually d=1)



Lamport's Logical Clock

- A **total order of events ("=>") can be obtained as follows:**
- If a is any event in process Pi , and b is any event in process Pk , then **a => b** if and only if either:
- **Ci (a) < Ck (b) or**
- **Ci (a) = Ck (b) and Pi << Pk (if scalar timestamps of a and b are equal, ordering of events need to done through some mechanism).**

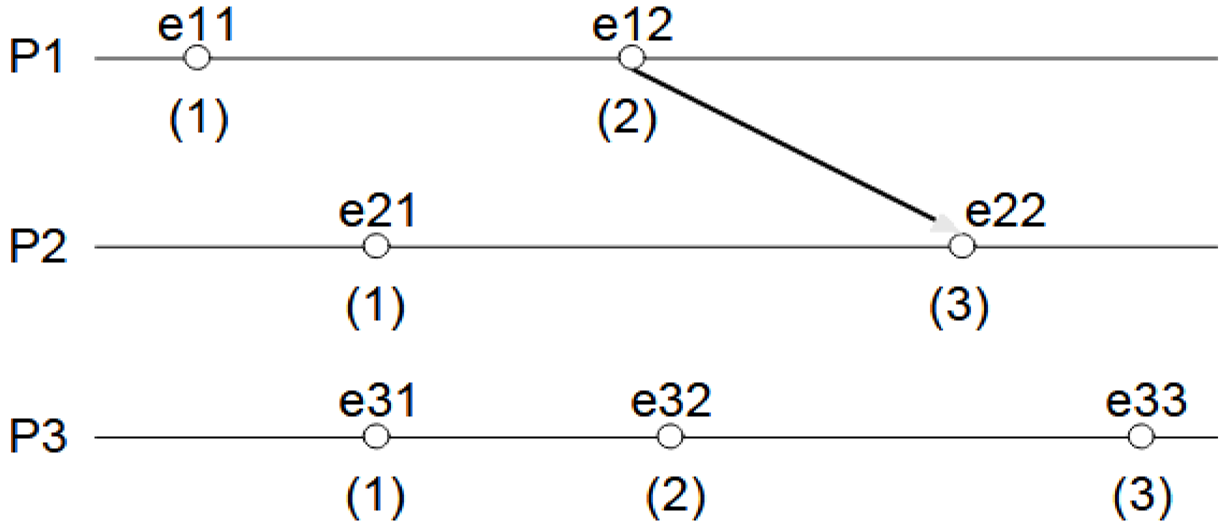
where "<<" denotes a relation that totally orders the processes. Process identifier is used to break ties.

Lower the process identifier in ranking higher the priority.

Limitations of Lamport's Logical Clock

- With Lamport's logical clocks,
- if a $\rightarrow$b, then $C(a) < C(b)$
- The following is **not necessarily true if events a and b occur in** different processes:
- if $C(a) < C(b)$, then a $\rightarrow$ b **is NOT TRUE, if a and b events occur in two different processes.**
- **Scalar time is not strongly consistent.**

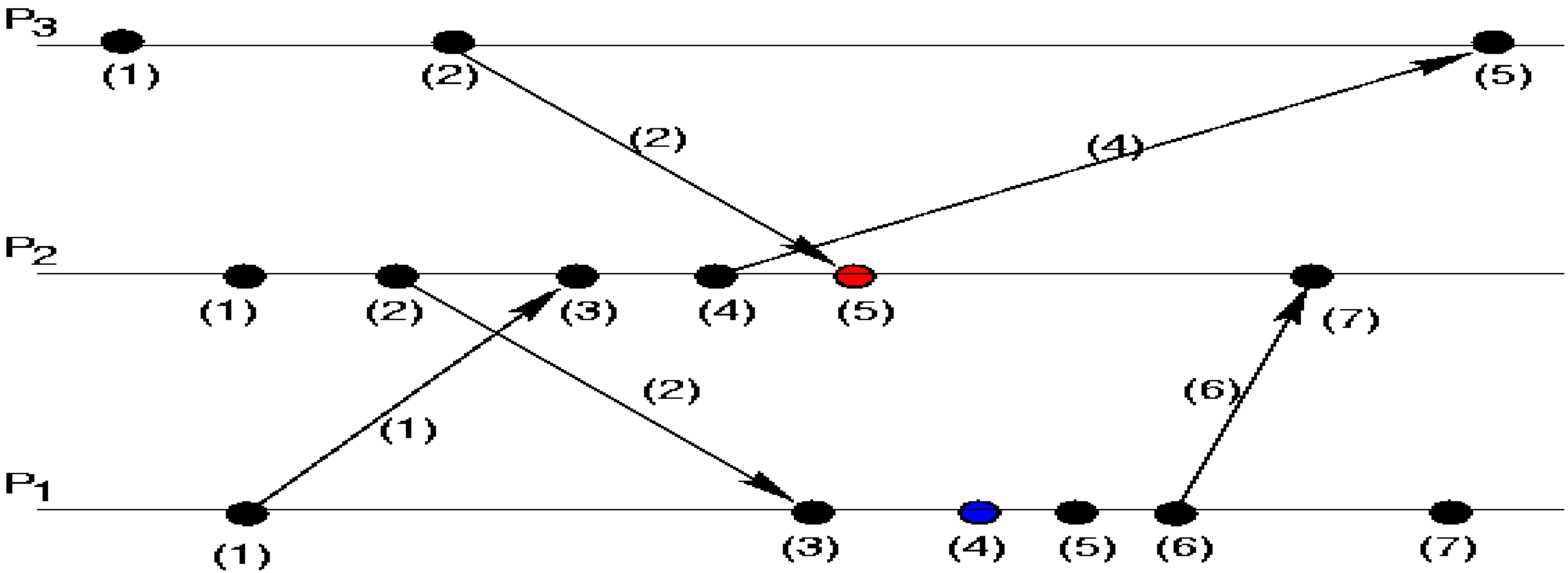
Limitations of Lamport's Logical Clock



- With Lamport's logical clocks, if $a \rightarrow b$, then $C(a) < C(b)$
  - The following is **not** necessarily true if events *a* and *b* occur in different processes: if $C(a) < C(b)$, then $a \rightarrow b$
  - $C(e11) < C(e22)$, and $e11 \rightarrow e22$ is true
  - $C(e11) < C(e32)$, but $e11 \rightarrow e32$ is false
- Cannot determine whether two events are causally related from timestamps

Limitations of Lamport's Logical Clock

There is drastic increase in clock values due to events that occur in different processes and that causally affects the other events. This is not captured in logical clocks.



Vector Clocks

Vector Clock

- Maintain a vector of values for every event that happens in all processes.
- Update happens for group of values in every event.

Vector Clock

- [IR1] Clock Ci must be incremented between any two successive events in process Pi :

  $C_i[i] := C_i[i] + d$, (d>0, usually d=1)

- [IR2] If event a is the event of sending a message m in process Pi , then message m is assigned a vector timestamp tm =Ci (a) When that same message m is received by a different process Pk , Ck is updated as follows:

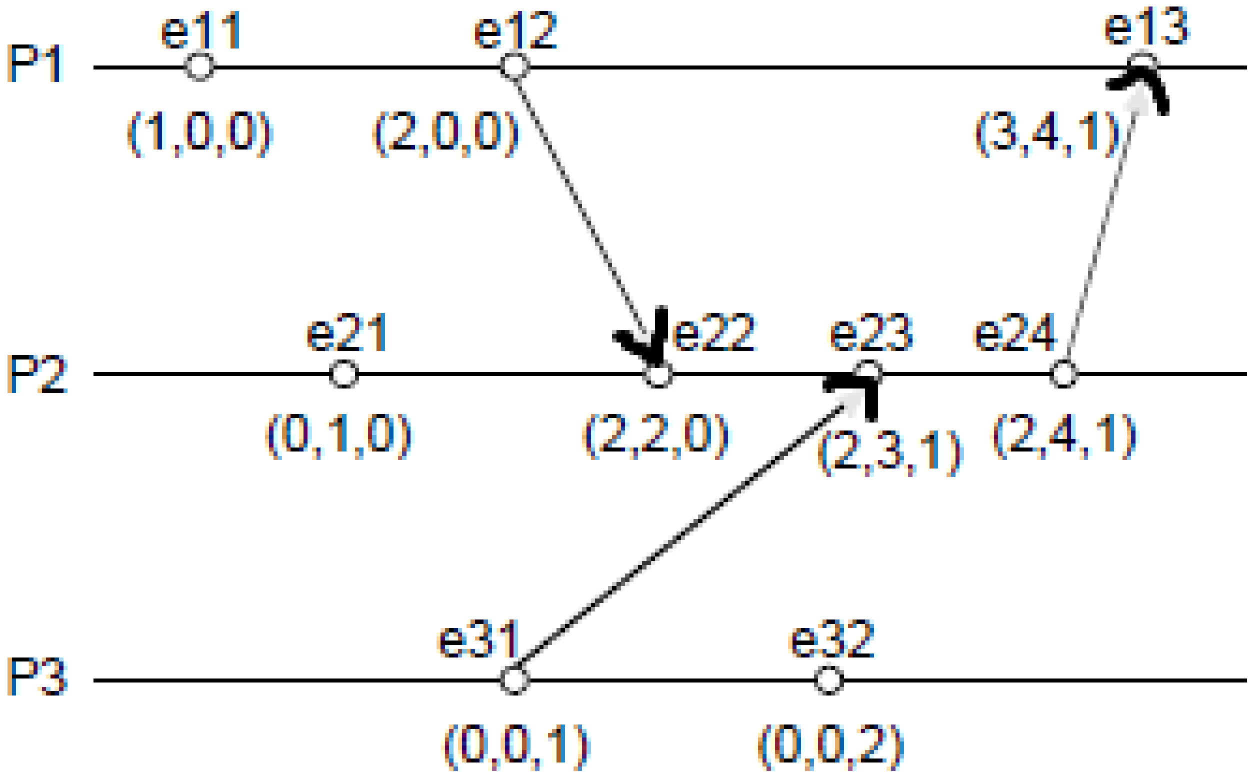
  $\forall p,\ C_k[p] := \max(C_k[p], tm[p] + d)$, (usually d=0 unless needed to model network delay)

- It can be shown that **$\forall i, \forall k : C_i[i] >= C_k[i]$**
- Rules for comparing timestamps can also be established so that
- if ta < tb , then a → b/ Solves the problem with Lamport's clocks

Vector Clock

IR1: Ci [i]:= Ci [i] + d . (d=1)

IR2: ∀ p, Ck [p]:= max(Ck [p], tm [p] + d) . (d=0)



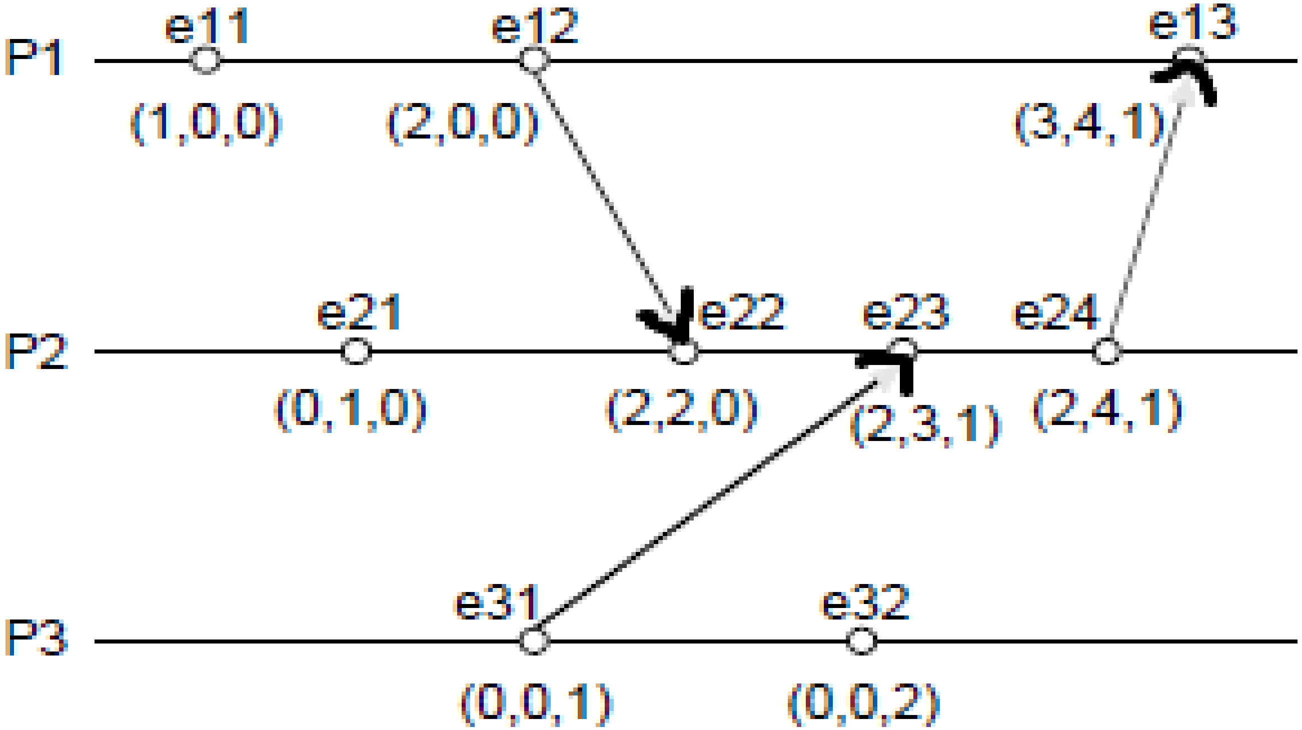
"e*nn*" is event; "(*n*,*n*,*n*)" is clock value

Vector Clock – Solving Logical Clock Problem

if t(e31) < t(e23) then e31 → e23 is true.

if t(e12) < t(e24) then e12 → e24 is also true.

Concurrent events e32 || e24; e32 ↛ e24 and e24 ↛ e32



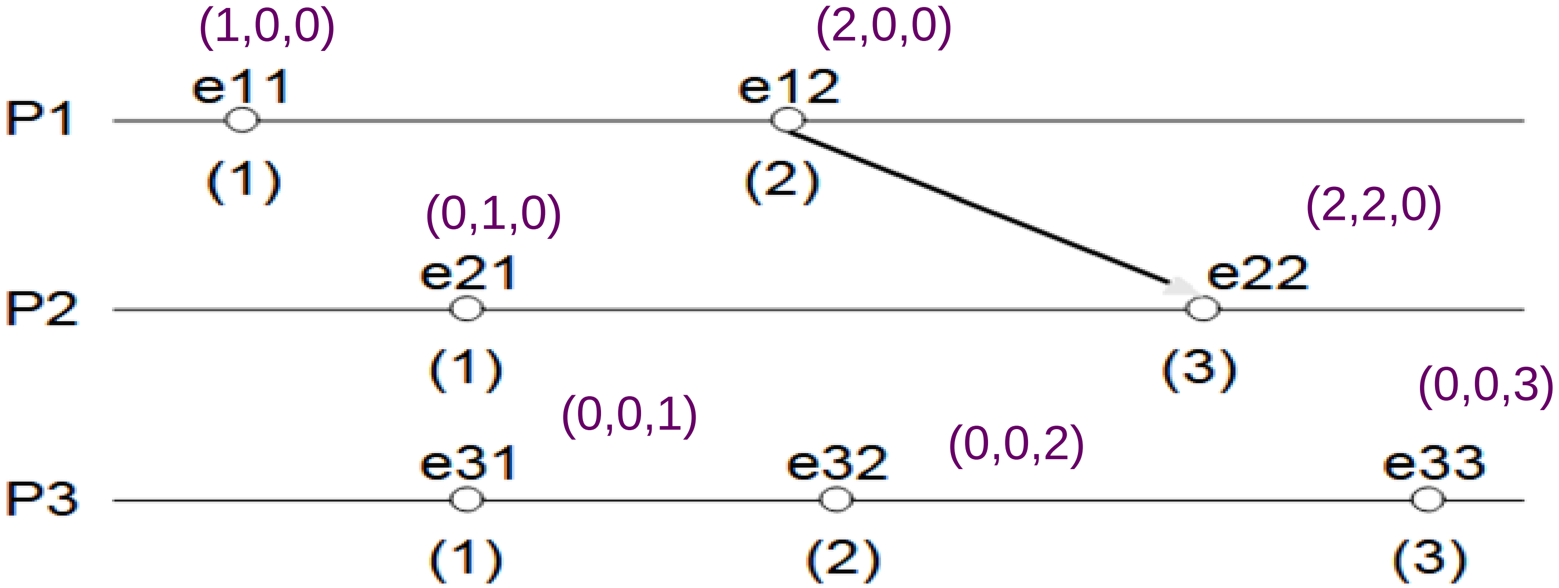
"enn" is event; "(n,n,n)" is clock value

Vector Clock – Solving Logical Clock Problem

if C(e11) < C(e22) then e11 → e22 is true.

if C(e11) < C(e32) then e11 → e32 is also true.

Vector time is strongly consistent.



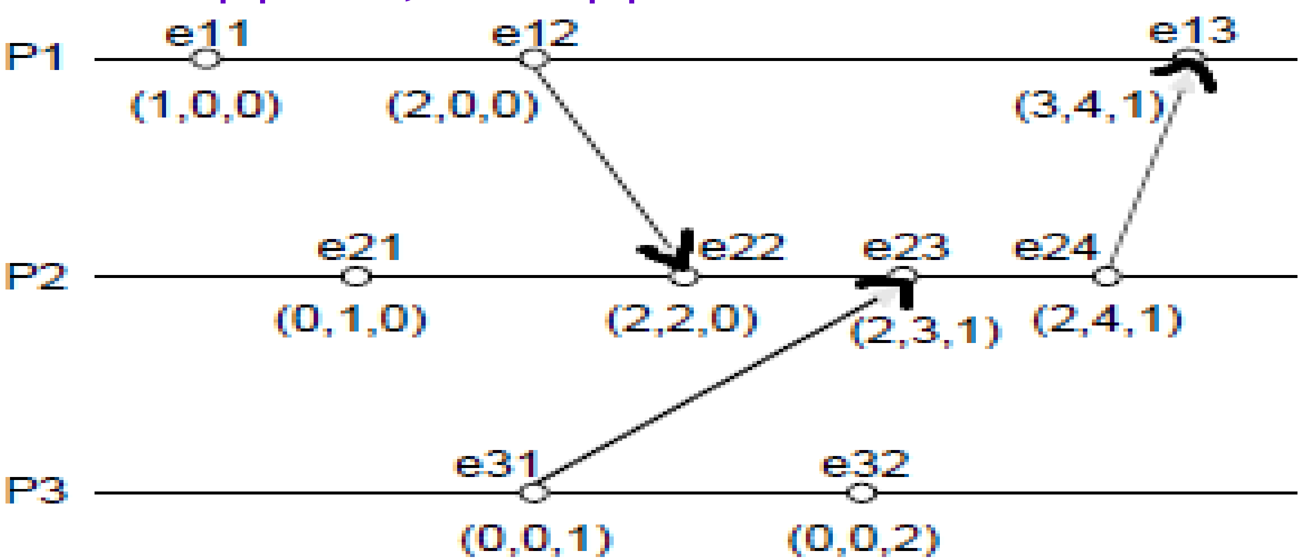
Vector Clock – Concurrent Events

Concurrent events e32 ||e24; e32 $\not\rightarrow$ e24 and e24 $\not\rightarrow$ e32

C(e32) = (0,0,2) and C(e24) = (2,4,1)

C(e32) is neither less nor grater than C(e24)

e11 || e21, e11||e31, e11||e32, e21 ||e32, e12 ||e32, e22 || e32, e32 ||e13.



"e*nn*" is event; "(*n*,*n*,*n*)" is clock value

Vector Clock

Example of Vector Clock

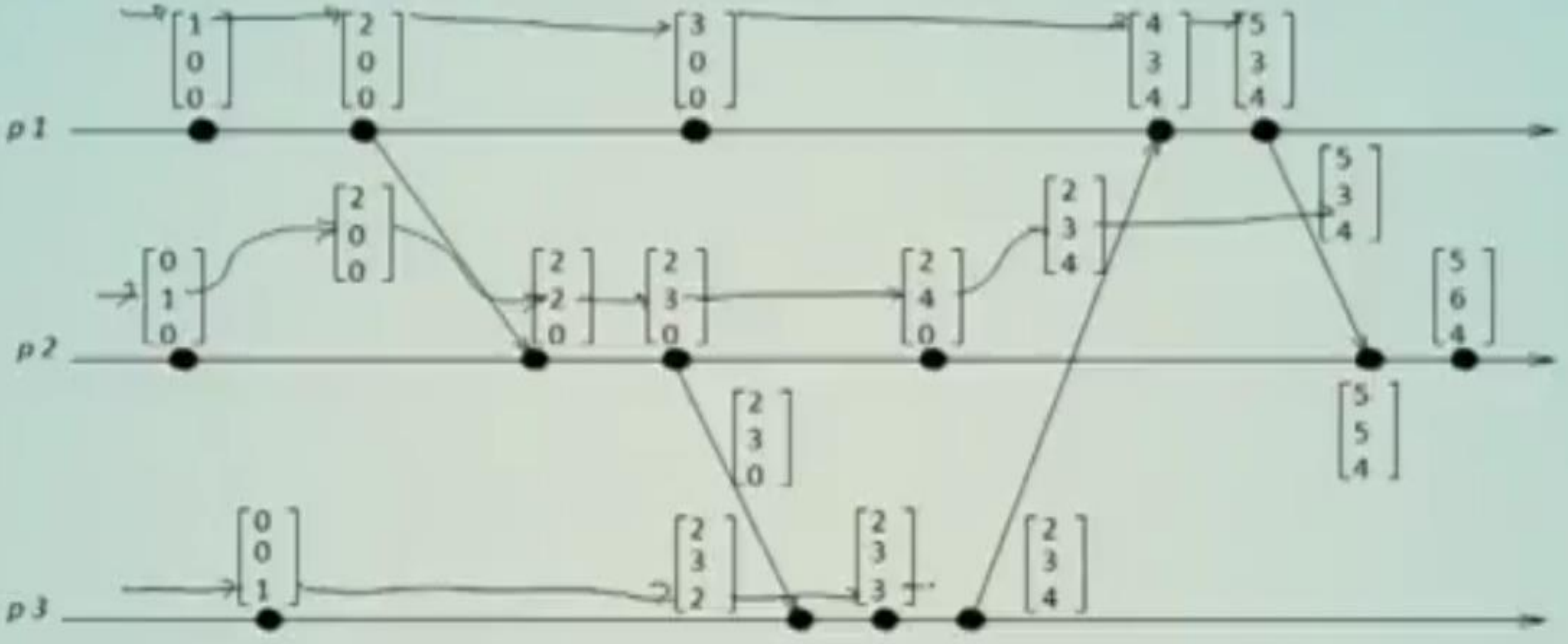


Figure 4.3: Evolution of vector time.

Drawbacks of Vector Clock

- Need to maintain memory units for tracking all events in all processes as a vector.
- The total numbers of processes may not be know in advance. Number of processes may be created or terminated at any time.
- Unnecessary wastage of maximum allocation of memory units.

Applications of Vector Clock

- **Distributed debugging**.
- Implementations of **causal ordering** communication and **causal distributed shared memory**.
- Establishment of **global breakpoints**.
- Determining the **consistency of checkpoints** in optimistic recovery.

Summary

- Physical Clocks
- Synchronizing Physical Clock (Algorithms)
- Problems with Physical Clock
- Lamport's Logical Clock
- Problems with Logical Clock
- Vector Clock
- Drawbacks of Vector Clocks
- Applications of Vector Clocks

References

1. Kshemkalyani, Ajay D., and Mukesh Singhal. Distributed computing: principles, algorithms, and systems. Cambridge University Press, 2011
2. Mukesh Singhal & N.G. Shivaratri, Advanced Concepts in Operating Systems
3. George Coulouris, Jean Dollimore and Tim Kindberg, "Distributed Systems Concepts and Design", Fifth Edition, Pearson Education, 2012

Thank You