LECTURE -3

LOGICAL CLOCK SYNCHRONIZATION

Prof. D. S. Yadav
Department of Computer Science
IET Lucknow

EVENT STRUCTURES

- A process can be viewed as consisting of a sequence of events, where an event is an atomic transition of the local state which happens in no time
- Process Actions can be modeled using the 3 types of events
  - Send
  - Receive
  - Internal (change of state)

CAUSAL RELATIONS

- **Distributed application results in a set of distributed events**
  - Induces a partial order → causal precedence relation

- **Knowledge of this causal precedence relation is useful in reasoning about and analyzing the properties of distributed computations**
  - Liveness and fairness in mutual exclusion
  - Consistency in replicated databases
  - Distributed debugging, Checkpointing

AN EVENT FRAMEWORK FOR LOGICAL CLOCKS

- **Events are related**
  - Events occurring at a particular process are totally ordered by their local sequence of occurrence.
  - Each receive event has a corresponding send event
  - Future can not influence the past (causality relation)
  - Event structures represent distributed computation (in an abstract way)
- An event structure is a pair *(E,<)*, where *E* is a set of events and < is a irreflexive partial order on *E*, called the causality relation

EVENT ORDERING

- **Lamport defined the “happens before” (<) relation**

- If a and b are events in the same process, and a occurs before b, then a<b.
- If a is the event of a message being sent by one process and b is the event of the message being received by another process, then a < b.
- If X <Y and Y<Z then X < Z.

If a < b then time (a) < time (b)

CAUSAL ORDERING

- **“Happens Before” also called causal ordering**
- **Possible to draw a causality relation between TWO events if**
  - They happen in the same process
  - There is a chain of messages between them
- **“Happens Before” notion is not straightforward in distributed systems**
  - No guarantees of synchronized clocks
  - Communication latency

LOGICAL CLOCKS

- **Used to determine causality in distributed systems**
- **Time is represented by non-negative integers**

- **A logical Clock C is some abstract mechanism which assigns to any event e∈E the value C(e) of some time domain T such that certain conditions are met**

$C: E \rightarrow T$ :: T is a partially ordered set

Such that, $e<e' \Rightarrow C(e)<C(e')$ holds

- **Consequences of the clock condition [Morgan 85]:**

Rule 1: If an event e occurs before event e' at some single process, then event e is assigned a logical time earlier than the logical time assigned to event e'

Rule 2 : For any message sent from one process to another, the logical time of the send event is always earlier than the logical time of the receive event

IMPLEMENTING LOGICAL CLOCKS

- **Requires**
  - Data structures local to every process to represent logical time and
  - a protocol to update the data structures to ensure the consistency condition.

- **Each process Pi maintains data structures that allow it the following two capabilities:**
  - A local logical clock, denoted by LCi , that helps process Pi measure its own progress.
  - A logical global clock, denoted by GCi , that is a representation of process Pi ’s local view of the logical global time. Typically, LCi is a part of GCi

IMPLEMENTING LOGICAL CLOCKS (CONTD..)

The protocol ensures that a process's logical clock, and thus its view of the global time, is managed consistently.

The protocol consists of the following two rules:

R1: This rule governs how the local logical clock is updated by a process when it executes an event.

R2: This rule governs how a process updates its global logical clock to update its view of the global time and global progress.

Types of Logical Clocks

- **Systems of logical clocks differ in their representation of logical time and also in the protocol to update the logical clocks.**
- **Three types of logical clocks**
  - Scalar
  - Vector
  - Matrix

HAPPENED BEFORE RELATION (→)

- Happened Before Relation (→) captures causal dependencies among various events in distributed systems.
- Internal Events / Message Events.
- In a same process, A →B indicate event A happened before Event B. (A and B are internal events)
- If A is event of sending a message M from process P to process Q, and B is event of receipt of message M at process Q then A →B.
- Happened Before Relation (→) is transitive ;

  A →B & B →C y A →C
- Causally Related Events : Event A causally affects event B if A →B.
- Concurrent Events : Two distinct events are said to be concurrent (A‖B) iff,

  A 2 B & B 2 A.

SCALAR LOGICAL CLOCKS - LAMPORT

- Proposed by Leslie Lamport in 1978 as an attempt to totally order events in a distributed system.
- Time domain is the set of non-negative integers.
- The logical local clock of a process Pi and its local view of the global time are squashed into one integer variable Ci .
- Monotonically increasing counter
  - No relation with real clock
- Each process keeps its own logical clock used to timestamp events

CONDITIONS TO BE SATISFIED BY SYSTEM OF LOGICAL CLOCKS

- **For any two distinct events A & B :**

If $A \rightarrow B$ **then** $C(A) < C(B)$

Happened Before Relation ($\rightarrow$) can be realized by using logical clocks if following conditions are satisfied :

[C1] : For any two events **A** & **B** occurred in a process $\mathbf{P_i}$ then :

$$\mathbf{C_i\,[A] < C_i\,[B]}$$

where $\mathbf{C_i}$ is local clock at $\mathbf{P_i}$

[C1] : If **A** is event of sending a message **M** from process $\mathbf{P_i}$ to process $\mathbf{P_j}$, and **B** is event of receipt of message **M** at process $\mathbf{P_j}$ then :

$$\mathbf{C_i\,[A] < C_j\,[B]}$$

where $\mathbf{C_i}$ is local clock at $\mathbf{P_i}$ and $\mathbf{C_j}$ is local clock at $\mathbf{P_j}$

IMPLEMENTATION RULES OF LAMPORT'S LOGICAL CLOCKS : PROTOCOL

- **To guarantee the clock condition, local clocks must obey a simple protocol:**
  - When executing an internal event or a send event at process P_i the clock C_i ticks
    - $\mathbf{C_i = C_i + d}$ (d>0)
  - When P_i sends a message m to Process P_j , it piggybacks a logical timestamp t which equals the time of the send event : $\mathbf{t(m) = C_i}$
  - When executing a receive event at P_j where a message with timestamp *t* is received, the clock is advanced

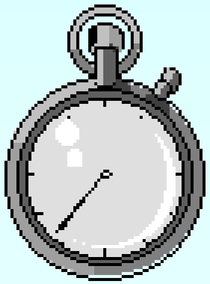
    $\mathbf{C_j = max(C_j,}\boldsymbol{t}\mathbf{) + d}$ (d>0)

⇒ **The above results in a partial ordering of events.**

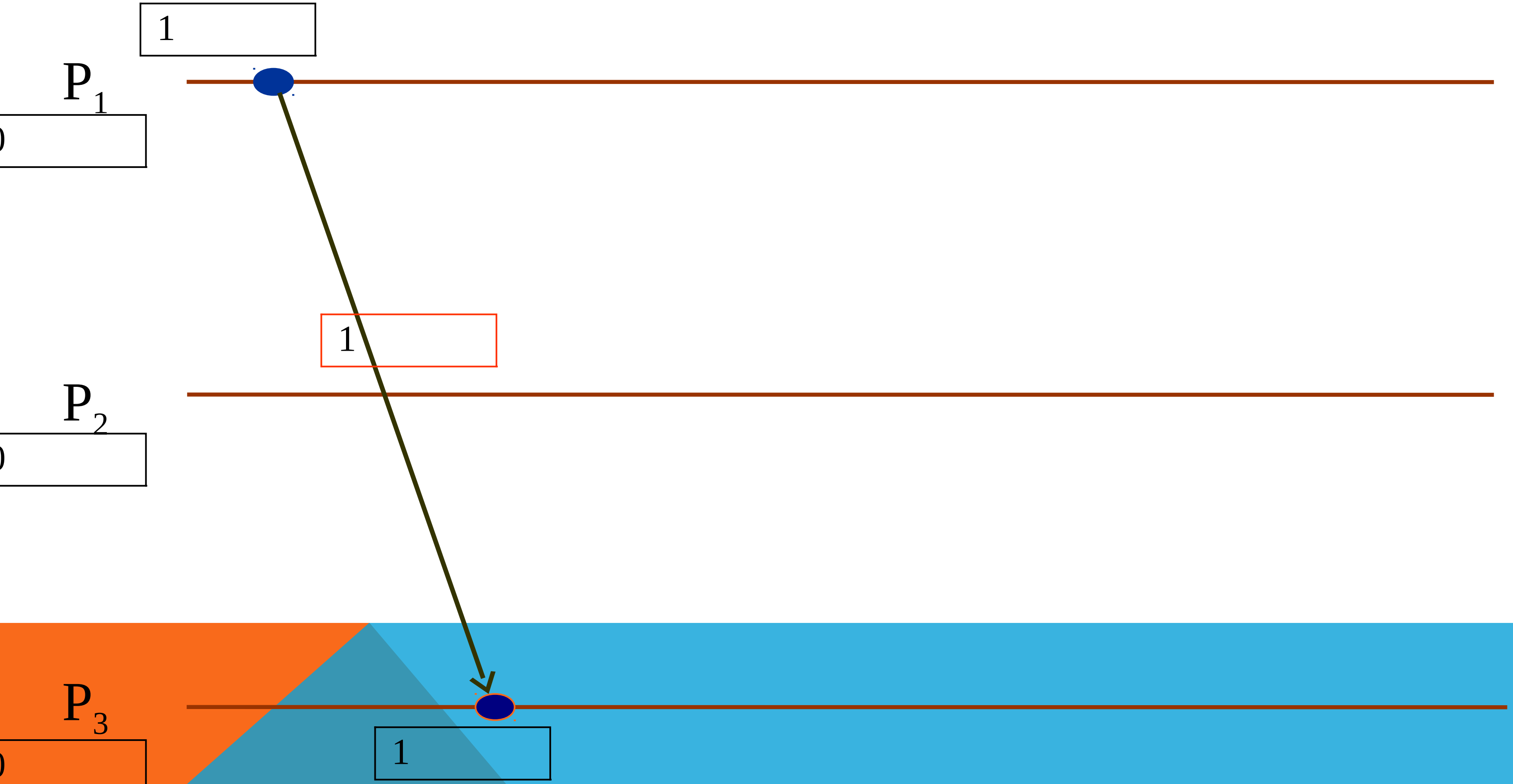
Lamport Logical Clock

P1	P2	P3
0	0	0
4	5	3
8	10	6
12	15	9
16	20	12
20	25	15
24	30	18
28	35	21
32	40	24

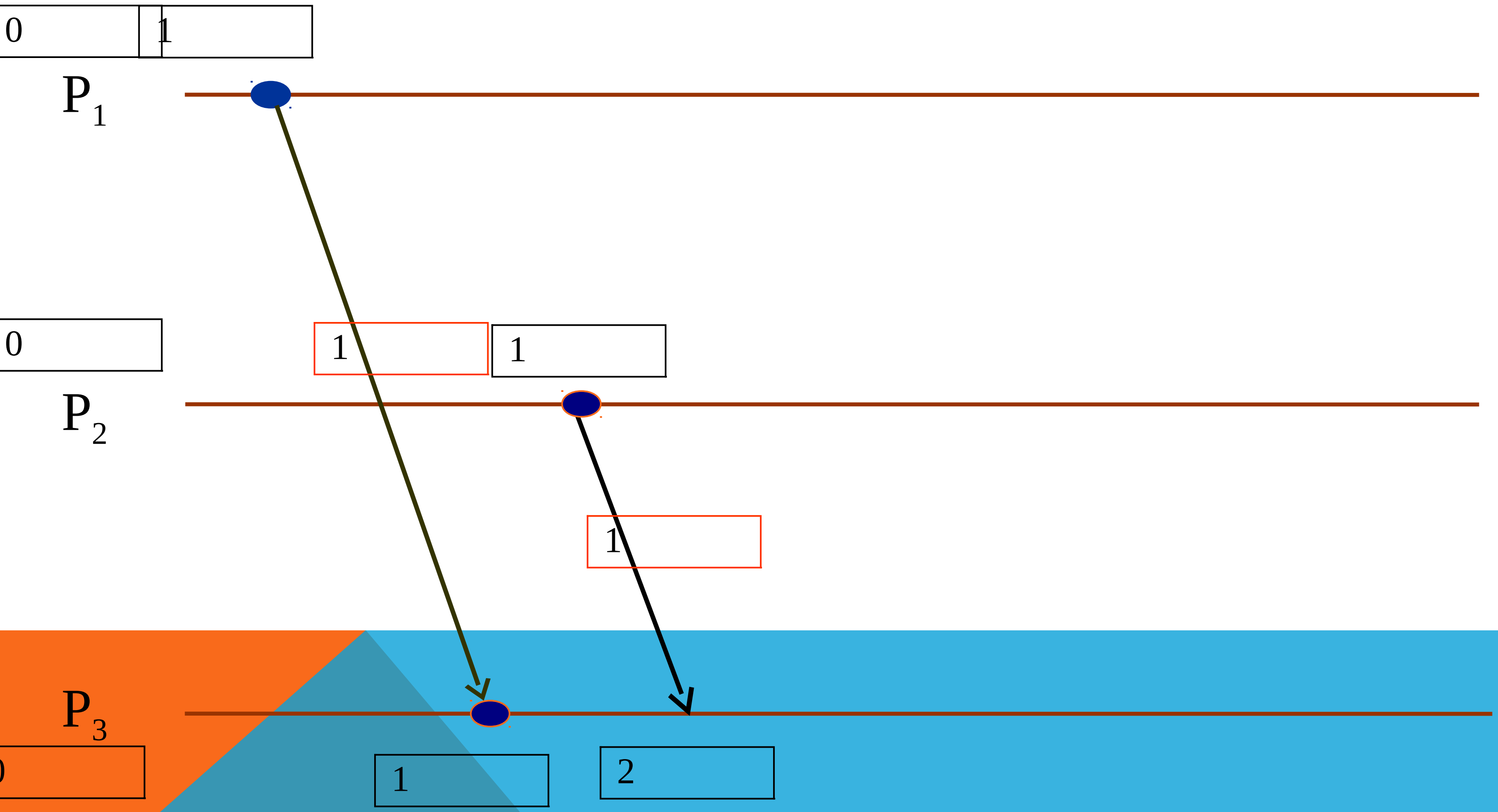
P1	P2	P3
0	0	0
4	5	3
8	10	6
12	15	**11**
16	20	14
20	25	17
24	30	20
28	35	23
36	40	26



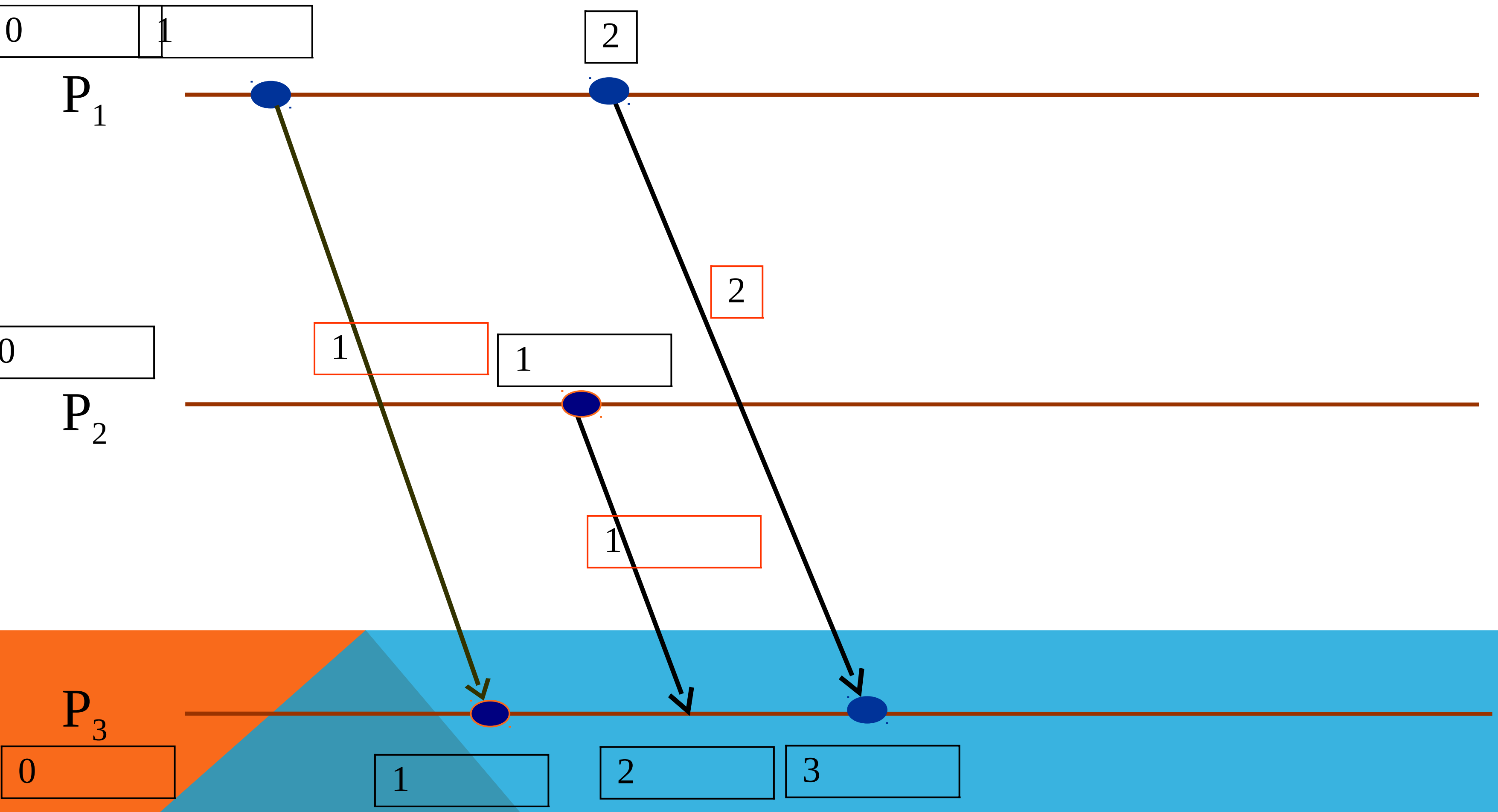
LAMPORT'S CLOCKS
UNICAST SYSTEM



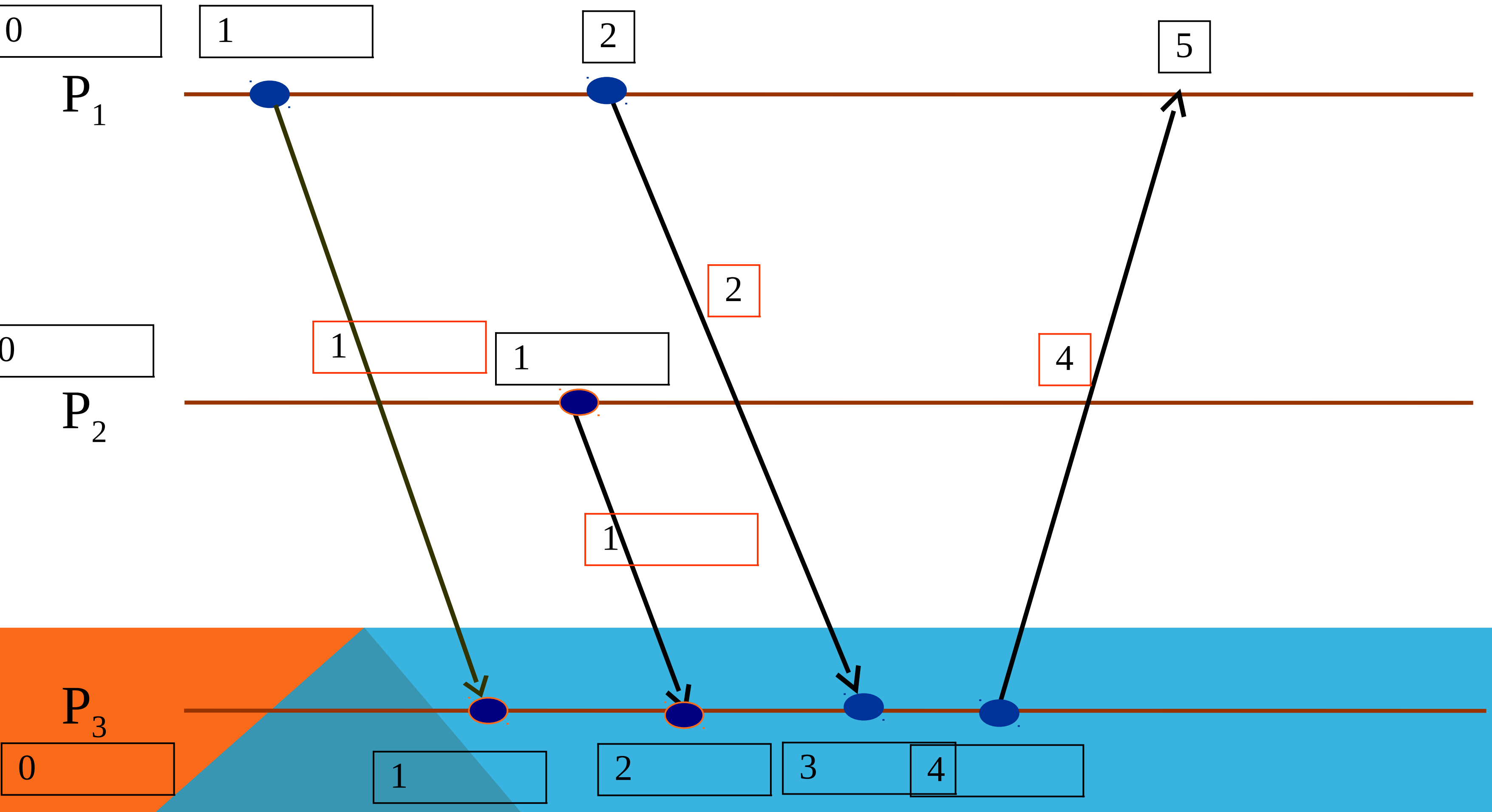
LAMPORT'S CLOCKS
UNICAST SYSTEM



LAMPORT'S CLOCKS
UNICAST SYSTEM

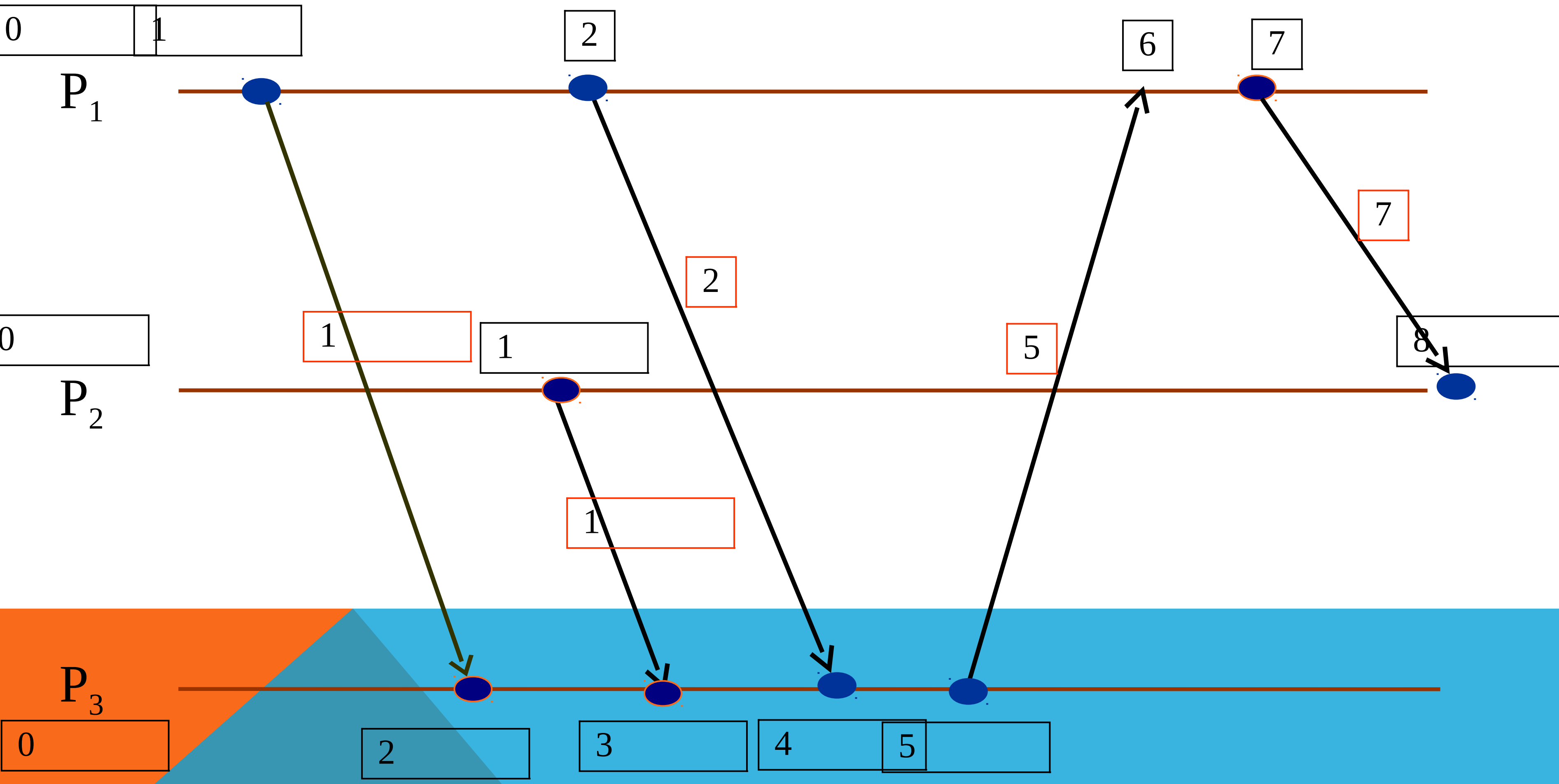


LAMPORT'S CLOCKS
UNICAST SYSTEM



LAMPORT'S CLOCKS

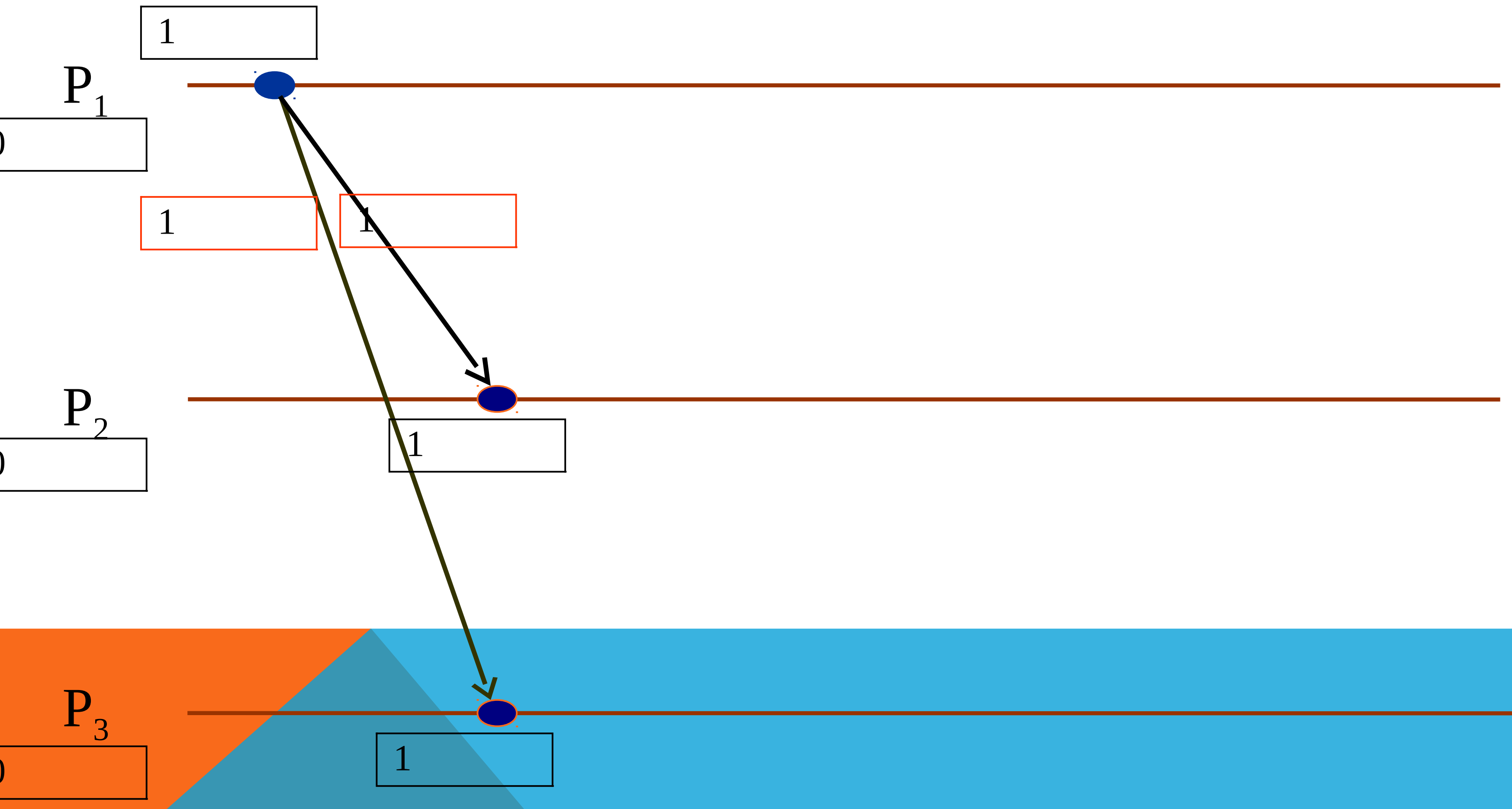
UNICAST SYSTEM



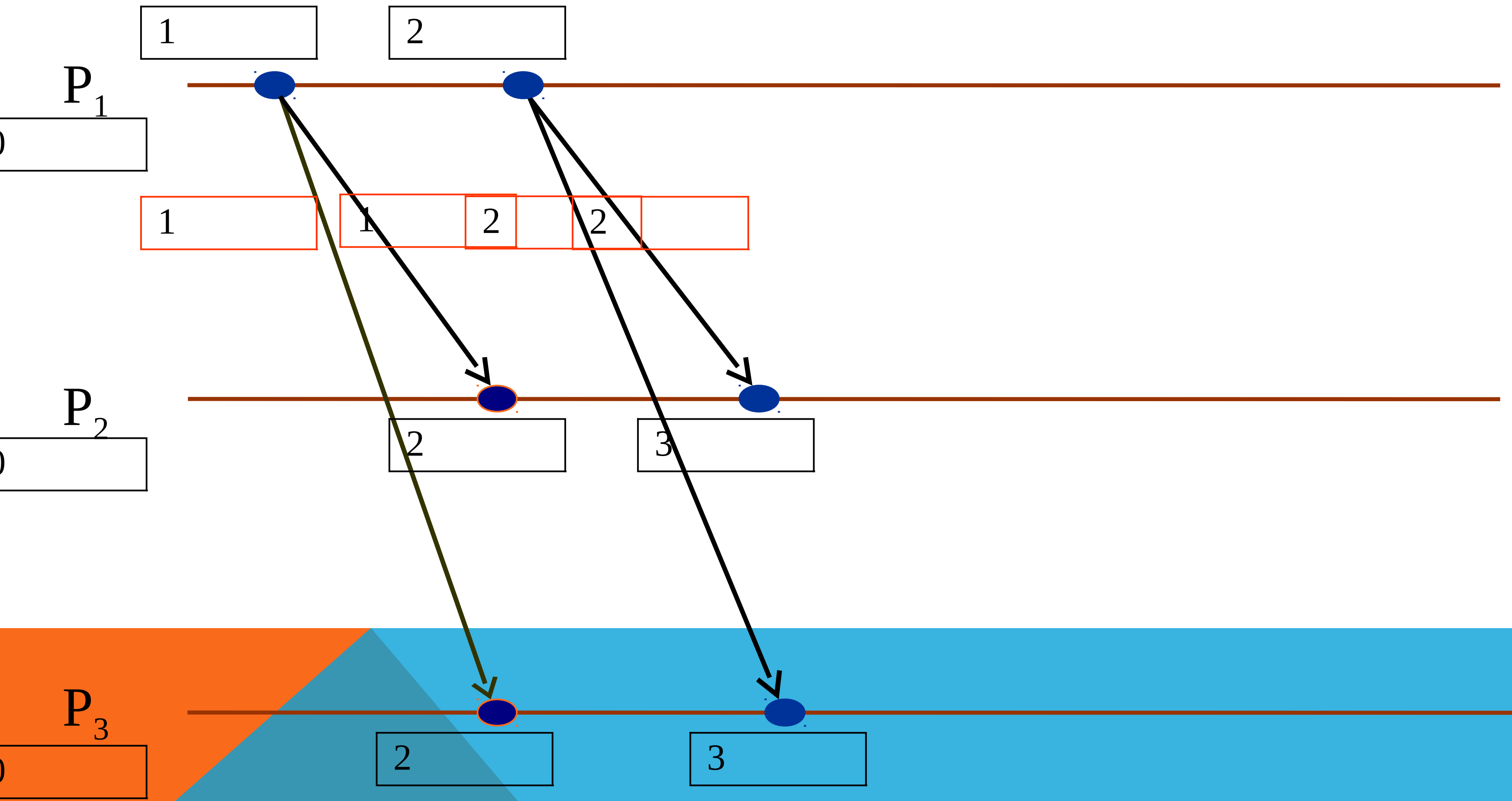
How Lamport's Clock work for Broadcast ?

APPLYING LAMPORT'S CLOCKS

TO BROADCAST SYSTEM

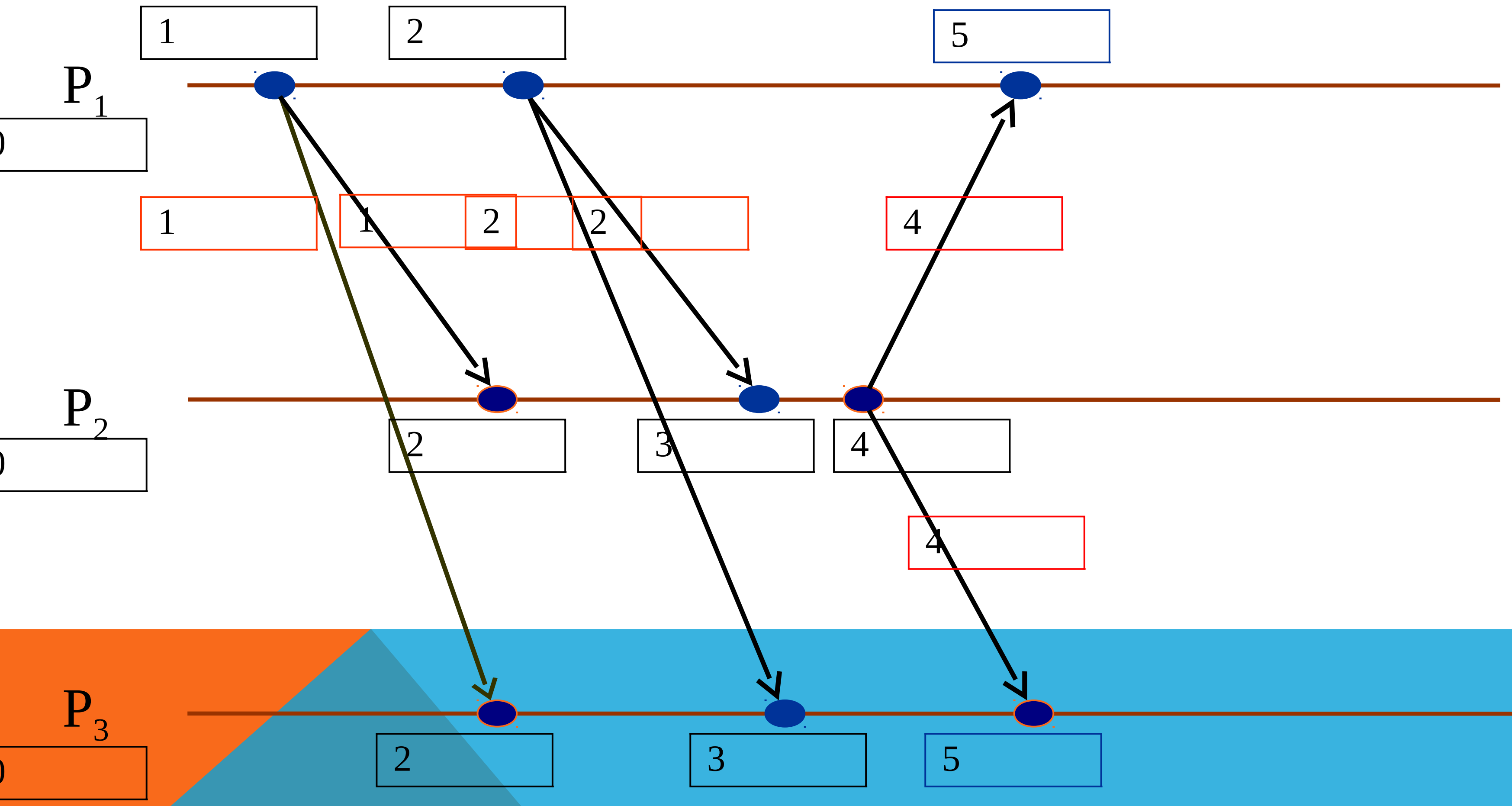


CLOCKS
TO BROADCAST SYSTEM



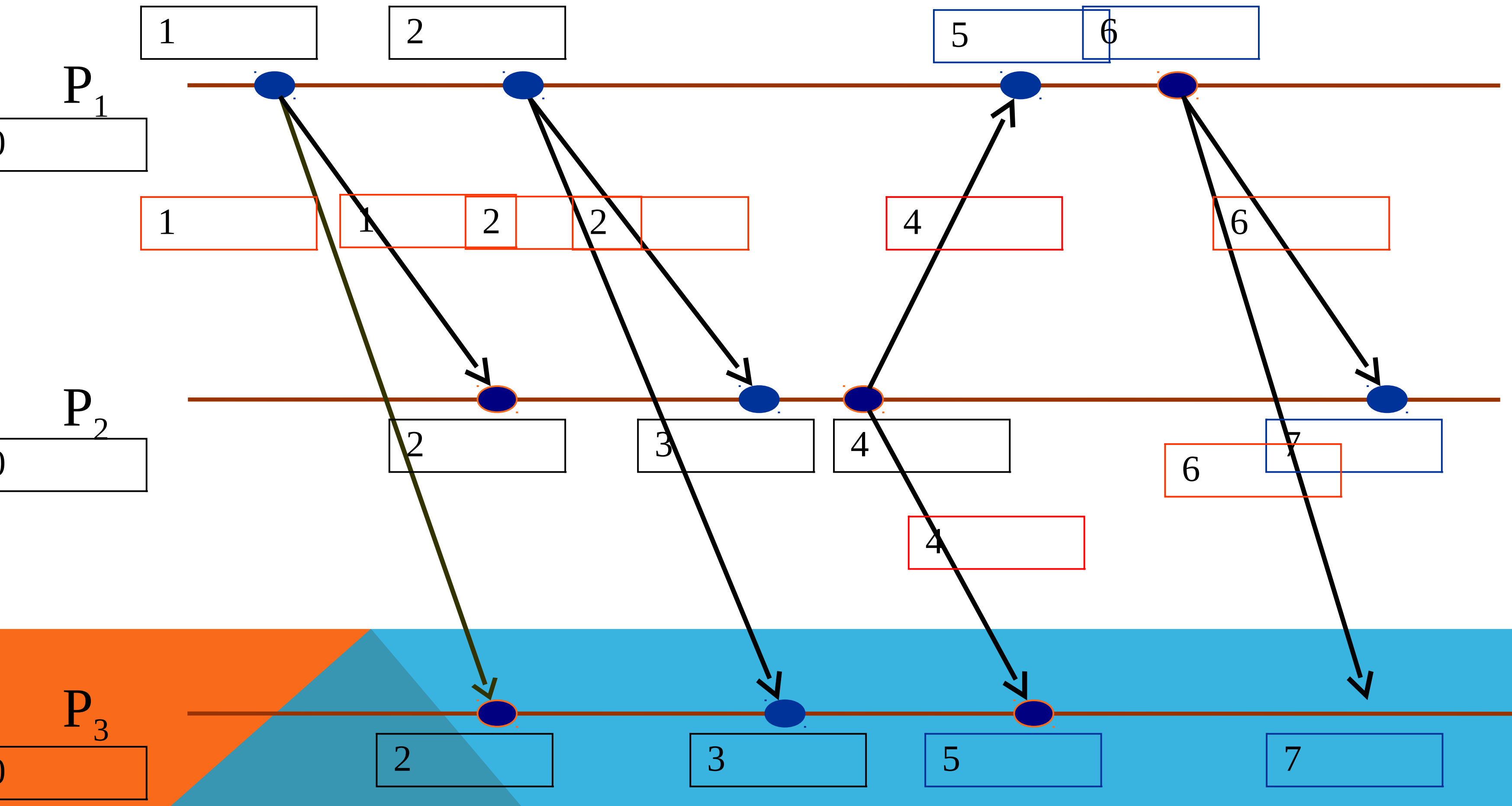
CLOCKS

TO BROADCAST SYSTEM



CLOCKS

TO BROADCAST SYSTEM



OBTAINING TOTAL ORDERING

- **Extending partial order to total order**

TIME	PROCESS ID

- **Global timestamps:**
  - (Ta, Pa) where Ta is the local timestamp and Pa is the process id.
    - **(Ta,Pa) < (Tb,Pb) iff**
    - **(Ta < Tb) or ((Ta = Tb) and (Pa < Pb))**
  - Total order is consistent with partial order.

PROPERTIES OF SCALAR CLOCKS

- **Event counting;**

  - If the increment value d is always 1, the scalar time has the following interesting property:

    if event e has a timestamp h, then h-1 represents the minimum logical duration, counted in units of events, required before producing the event e;

  - We call it the height of the event e.

  - In other words, h-1 events have been produced sequentially before the event e regardless of the processes that produced these events.

LIMITATION OF SCALAR CLOCKS

- No Strong Consistency
- The system of scalar clocks is not strongly consistent; that is, for two events ei and ej , ;LG

  $C(ei) < C(ej) \Rightarrow ei < ej$.

- Reason: In scalar clocks, logical local clock and logical global clock of a process are squashed into one, resulting in the loss of causal dependency information among events at different processes.

INDEPENDENCE

- **Two events e,e' are mutually independent (i.e. e||e') if ;**
- **~(e<e') ˄ ~(e'<e)**
  - Two events are independent if they have the same timestamp
  - Events which are causally independent may get the same or different timestamps
- **By looking at the timestamps of events it is not possible to assert that some event *could not* influence some other event**
  - If C(e)<C(e') then ~(e<e') *however*, it *is not possible* to decide whether e<e' or e||e'

    C is an order *homomorphism* which preserves < but it does not preserves negations (i.e. obliterates a lot of structure by mapping E into a linear order)
  - An *isomorphism* mapping E onto T is required

Assignment 2

Last Date of Submission : 12 Sept 2019
Date of presentation : will be announced later

END OF LECTURE 3