Time, Clocks and Ordering for Distributed Systems

Tuochao Chen
Tongyan wang

Time synchronization is important for computer systems

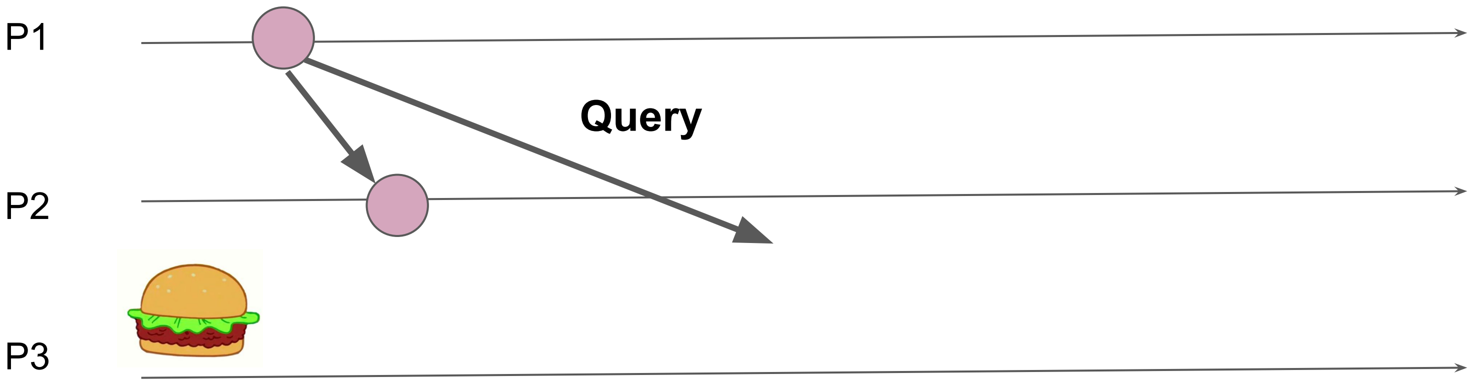
(Correctness) Event ordering and causality

(Fairness)Time-stamping events

Time synchronization is important for computer systems

In distributed system, a global and shared physics clock is usually unavailable

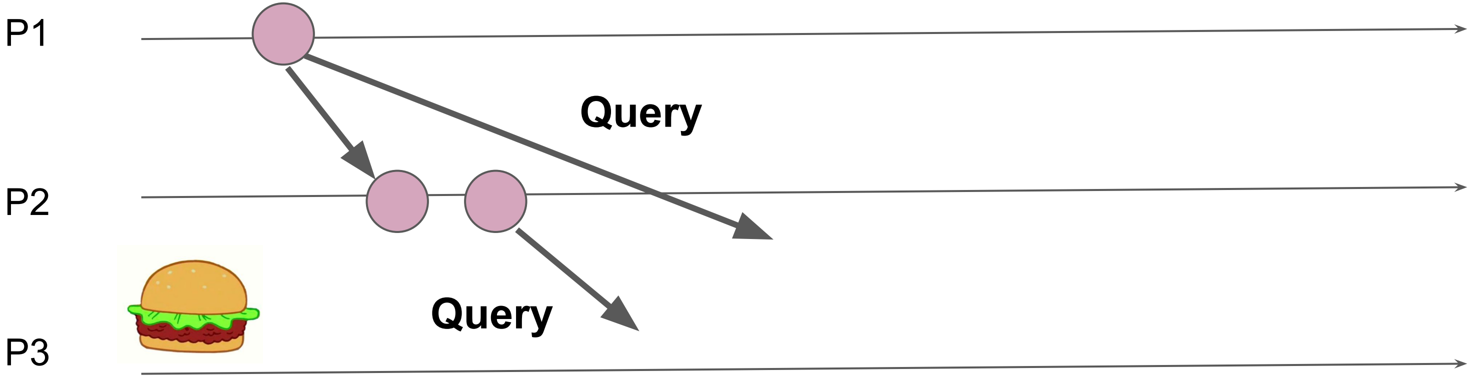
Mutual exclusion service:



Time synchronization is important for computer systems

In distributed system, a global and shared physics clock is usually unavailable

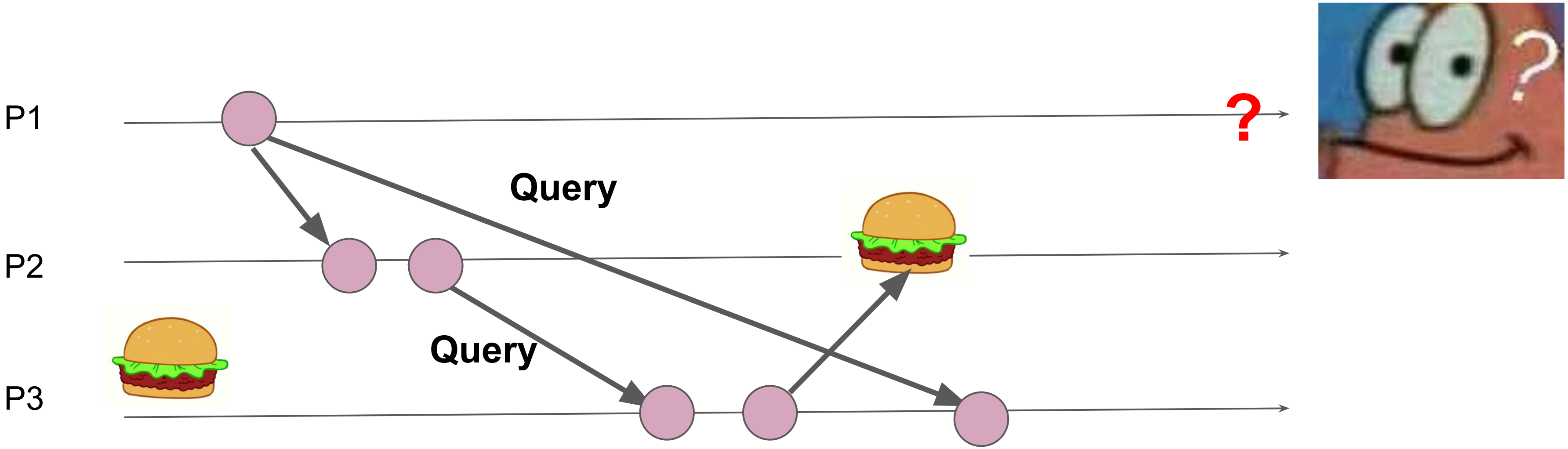
Mutual exclusion service:



Time synchronization is important for computer systems

In distributed system, a global and shared physics clock is usually unavailable

Mutual exclusion service:



Definition of Ordering: “Happen Before” and Concurrency.

Ci() is a function that represents a clock and assign a number to an event in process i.

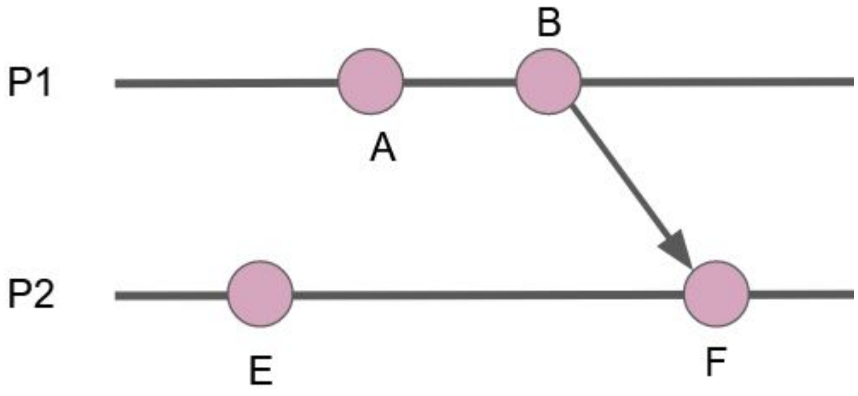
Happens Before: (a -> b)

C1. If a and b are events in process P_i, and a comes before b, then Ci(a) < Ci(b).

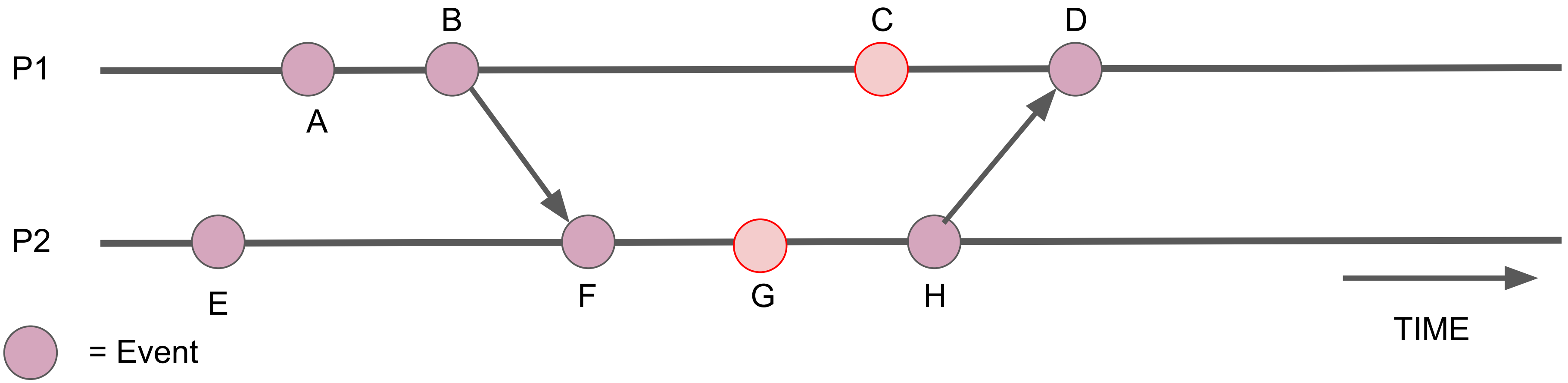
C2. If a is the sending of a message by process Pi and b is the receipt of that message by process Pj, then Ci(a) < Cj(b).

Concurrency: (a not -> b and b not -> a)

the order of any two events can not be determined is concurrent events.



Example for Concurrent Events and A “Happens Before” B



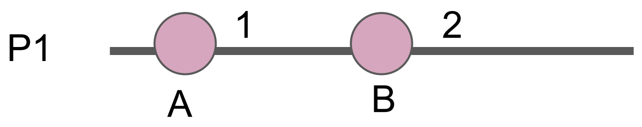
P: A process on an individual machine

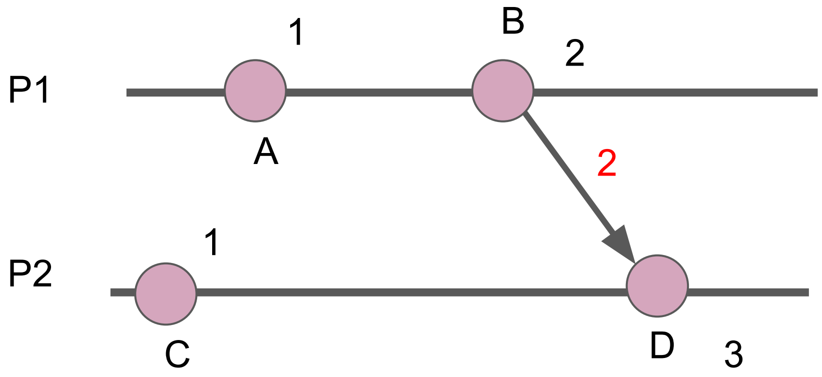
Introduction of Logical Clocks and Two Implementation Rules

Logical Clocks refer to implementing a protocol on all machines within your distributed system, so that the machines are able to maintain consistent ordering of events within some virtual timespan.

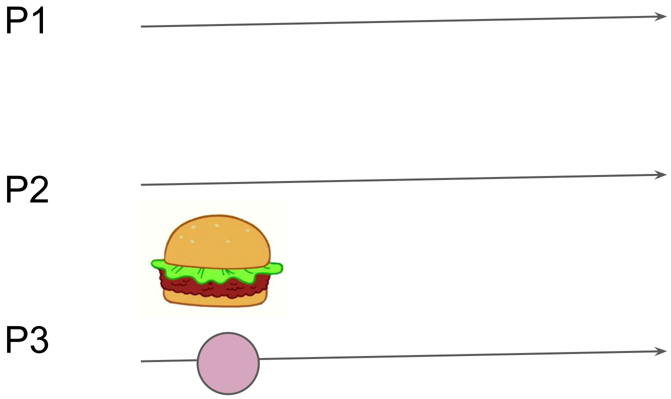
- Rule 1: A process updates its own clock when an event occurs.
  local_lock = local_clock + 1

- Rule 2: A process updates its own clock when it receives a message from another process.
  local_clock = max(local_clock, received_clock)
  local_clock = local_clock + 1





Solve Mutual exclusion service using logic clock:



1. To request the resource, process P_i sends the message T_m:P_i *requests resource* to every other process, and puts that message on its request queue, where T_m is the timestamp of the message.

2. When process P_j receives the message T_m:P_i *requests resource*, it places it on its request queue and sends a (timestamped) acknowledgment message to P_i.[5]

3. To release the resource, process P_i removes any T_m:P_i *requests resource* message from its request queue and sends a (timestamped) P_i *releases resource* message to every other process.

4. When process P_j receives a P_i *releases resource* message, it removes any T_m:P_i *requests resource* message from its request queue.

5. Process P_i is granted the resource when the following two conditions are satisfied: (i) There is a T_m:P_i *requests resource* message in its request queue which is ordered before any other request in its queue by the relation $\Rightarrow$. (To define the relation "$\Rightarrow$" for messages, we identify a message with the event of sending it.) (ii) P_i has received a message from every other process timestamped later than T_m.[6]

Solve Mutual exclusion service using logic clock:

3. To release the resource, process P_i removes any T_m:P_i *requests resource* message from its request queue and sends a (timestamped) P_i *releases resource* message to every other process.

4. When process P_j receives a P_i *releases resource* message, it removes any T_m:P_i *requests resource* message from its request queue.

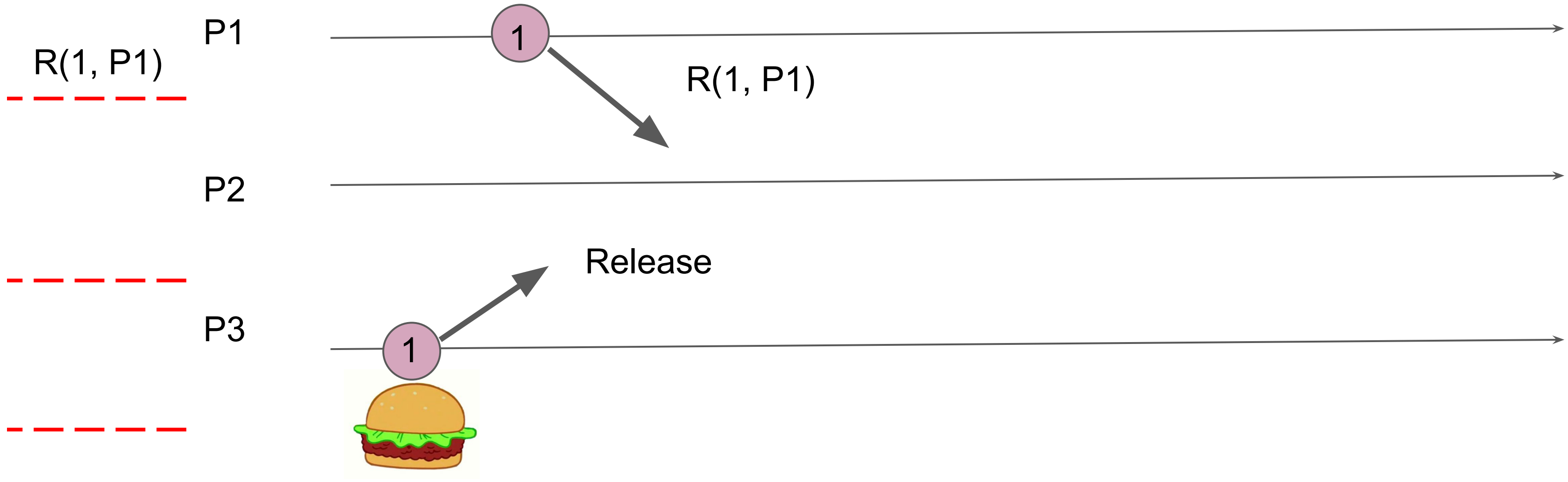
Stack

Solve Mutual exclusion service using logic clock:

P1 send request

1. To request the resource, process P_i sends the message T_m:P_i *requests resource* to every other process, and puts that message on its request queue, where T_m is the timestamp of the message.

Stack



Solve Mutual exclusion service using logic clock:

P2, P3 recv request from P1

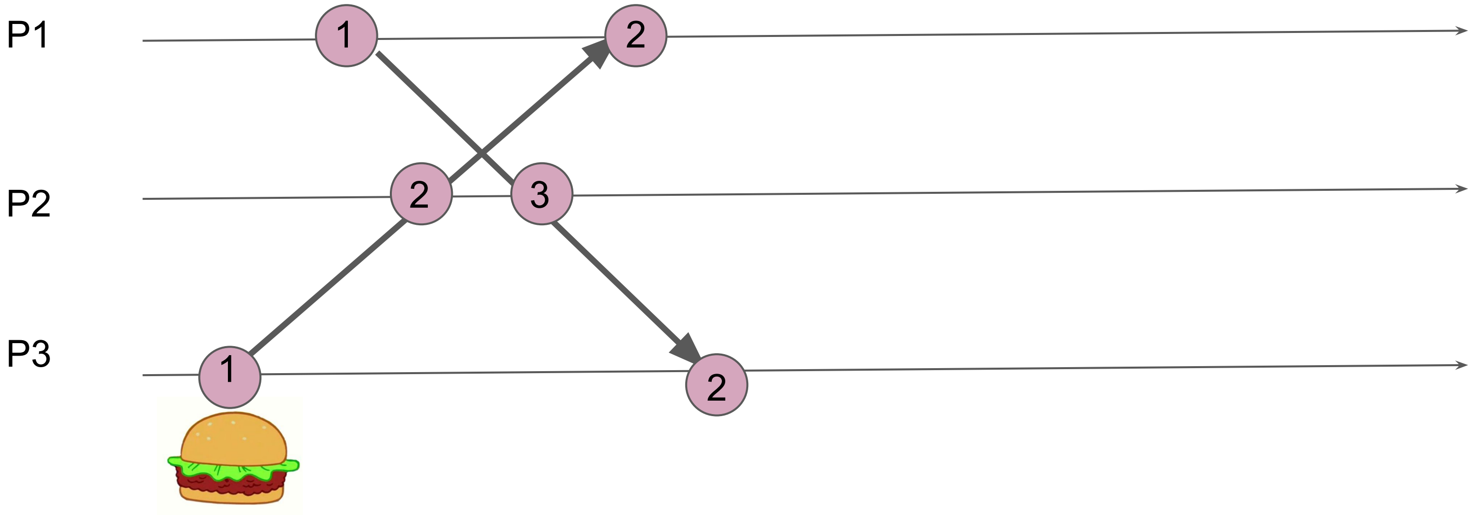
1. To request the resource, process P_i sends the message T_m:P_i *requests resource* to every other process, and puts that message on its request queue, where T_m is the timestamp of the message.

Stack

R(1, P1)

R(1, P1)

R(1, P1)

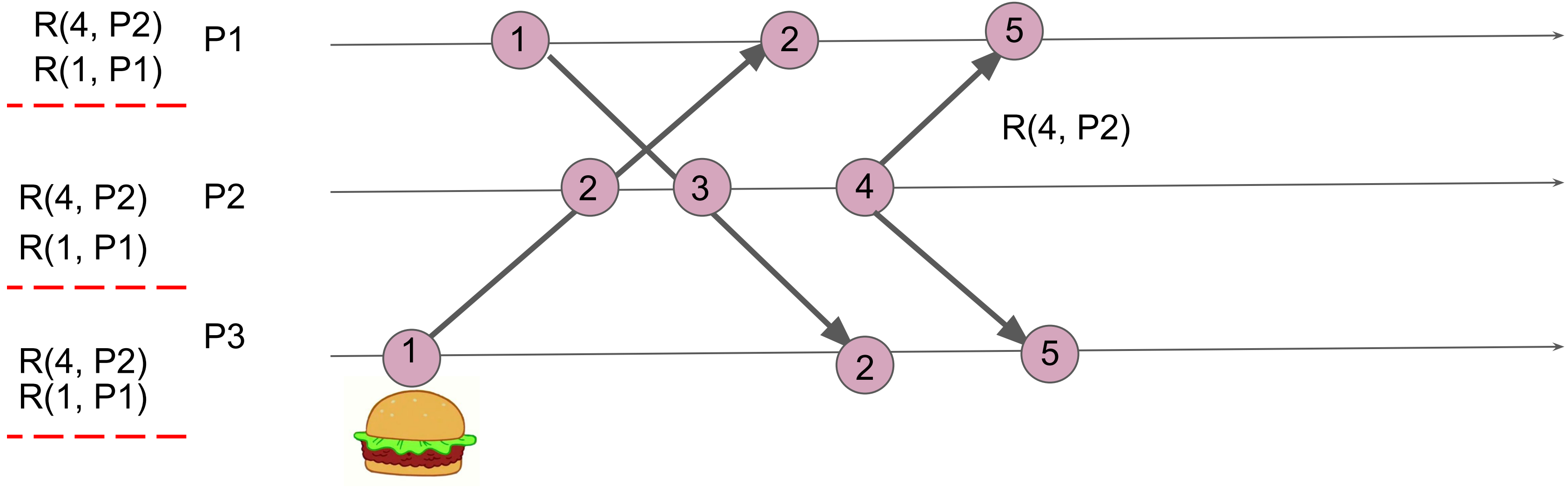


Solve Mutual exclusion service using logic clock:

P2 send request

> 1. To request the resource, process P_i sends the message T_m:P_i *requests resource* to every other process, and puts that message on its request queue, where T_m is the timestamp of the message.

Stack



Solve Mutual exclusion service using logic clock:

P3 send ack back to P1 and P2

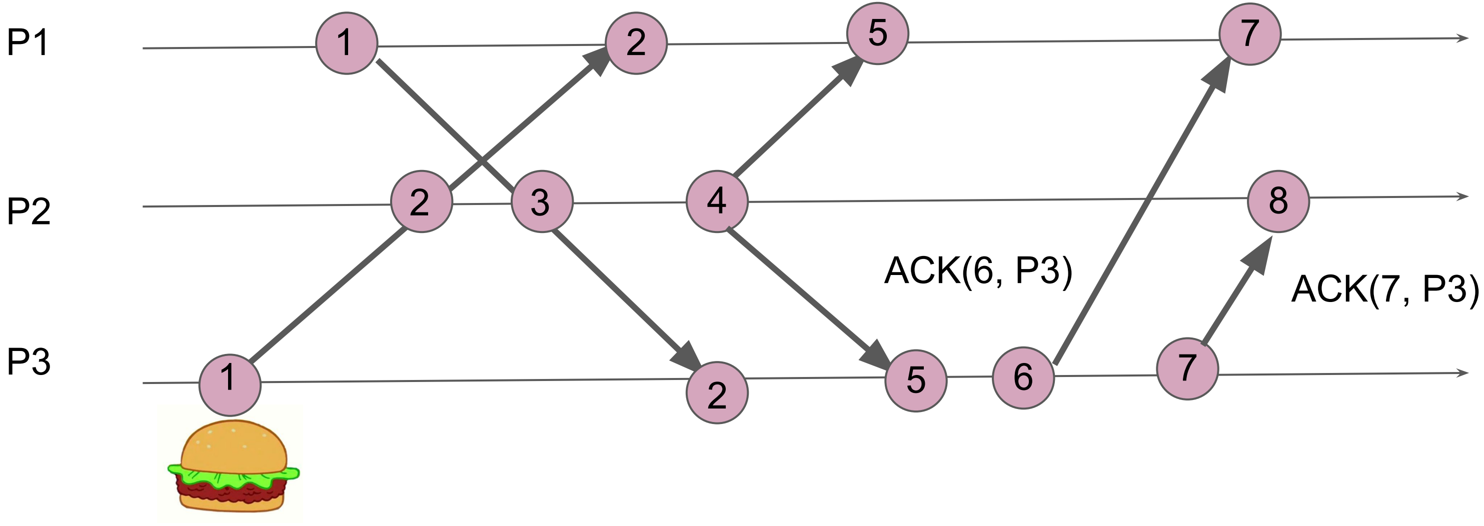
2. When process P_j receives the message T_m:P_i *requests resource*, it places it on its request queue and sends a (timestamped) acknowledgment message to P_i.[5]

Stack

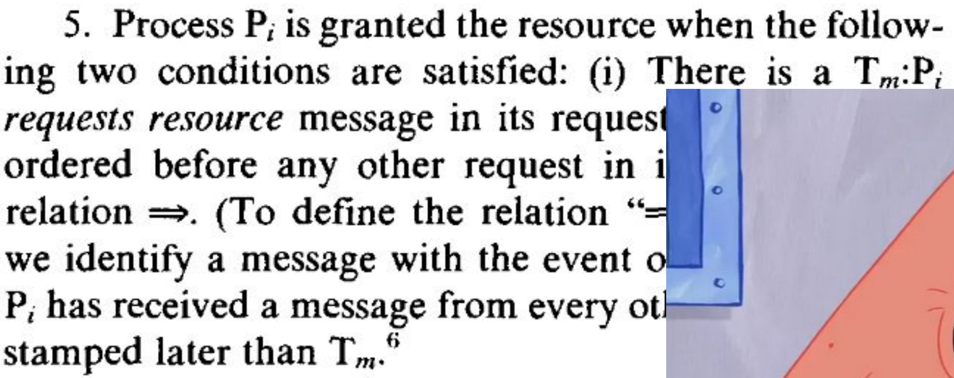
A(6, P3)
R(4, P2)
R(1, P1)

A(7, P3)
R(4, P2)
R(1, P1)

R(4, P2)
R(1, P1)



Solve Mutual exclusion service using logic clock:

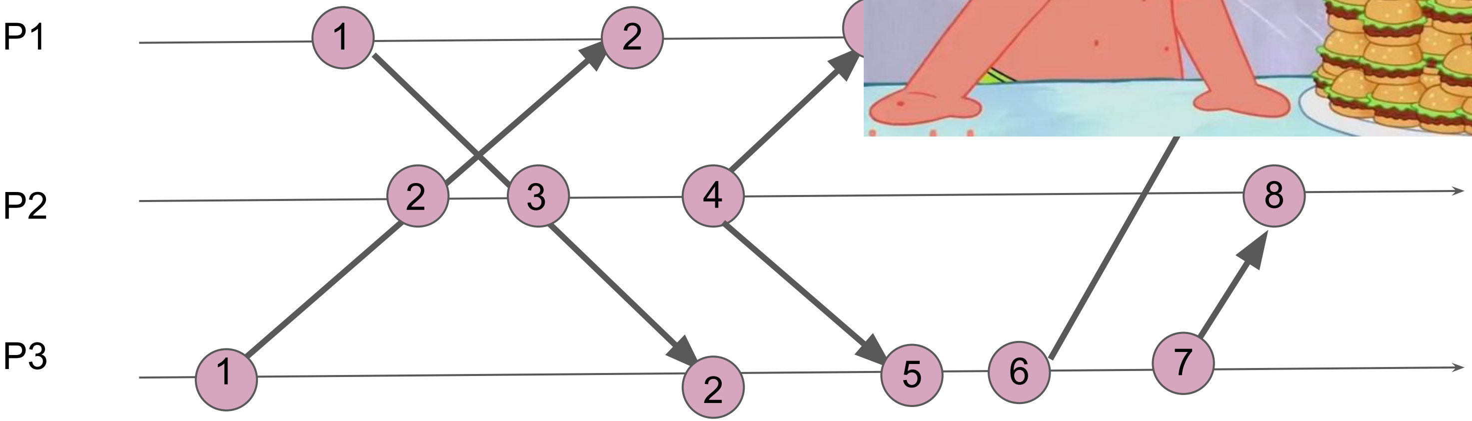

5. Process P_i is granted the resource when the follow-
ing two conditions are satisfied: (i) There is a T_m:P_i
requests resource message in its request
ordered before any other request in i
relation ⇒. (To define the relation "⇒
we identify a message with the event o
P_i has received a message from every ot
stamped later than T_m.[6]

Stack

A(6, P2)
R(4, P2)
R(1, P1)

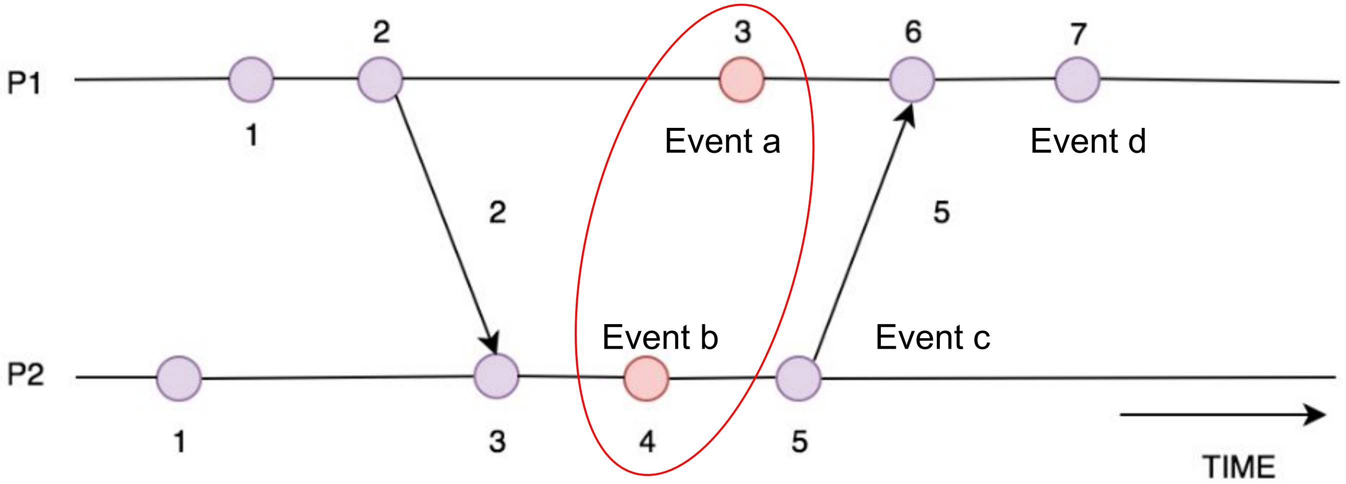
A(7, P3)
R(4, P2)
R(1, P1)

R(4, P2)
R(1, P1)



Drawbacks of Logic Clock I

1. Lamport clocks cannot tell if events was concurrent and their causality relationship.



c -> d ⟶ C1(d) > C2(c)

C2(b) = 4 > 3 = C1(a) ✗⟶ a -> b

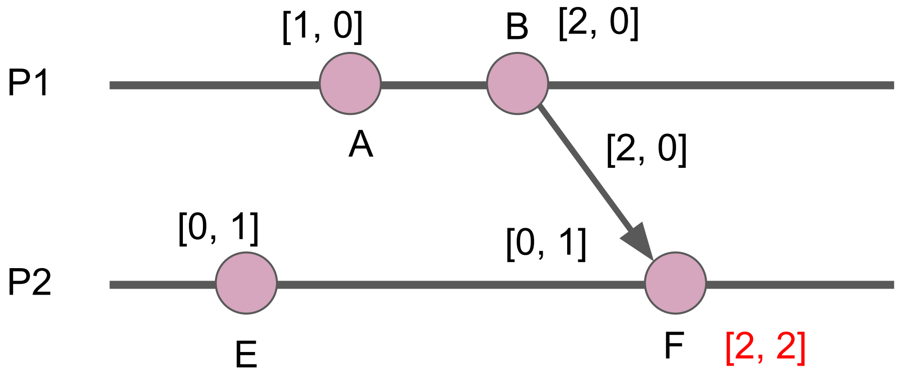
a,b is concurrent !

Solution to the Concurrency and Causality Issue — vector clock

Vector Clocks expand upon Scalar Time to provide a causally consistent view of the world.

- Rule 1:
  local_vector[i] = local_vector[i] + 1

- Rule 2:
  for k = 1 to N: V_i[k] = max(local_vector[k], sent_vector[k])
  local_vector[i] = local_vector[i] + 1



Max([0, 1], [2, 0]) = [2, 1]

[2, 1] + [0, 1] = [2, 2]

Relationship between vector clock and “happen before”

We define Vi(a) > Vj(b): $\forall$ k, Vi(a)[k] $\geqslant$ Vj(b)[k] and $\exists$ k, Vi(a)[k] > Vj(b)[k]

[2, 2, 0] > [1, 0, 0]

[1, 5, 0] $\not>$ [1, 5, 0]

[2, 4, 5] $\not>$ [1, 5, 0]

If Vi(a) > Vj(b): b->a

Elif Vi(a) < Vj(b): a->b

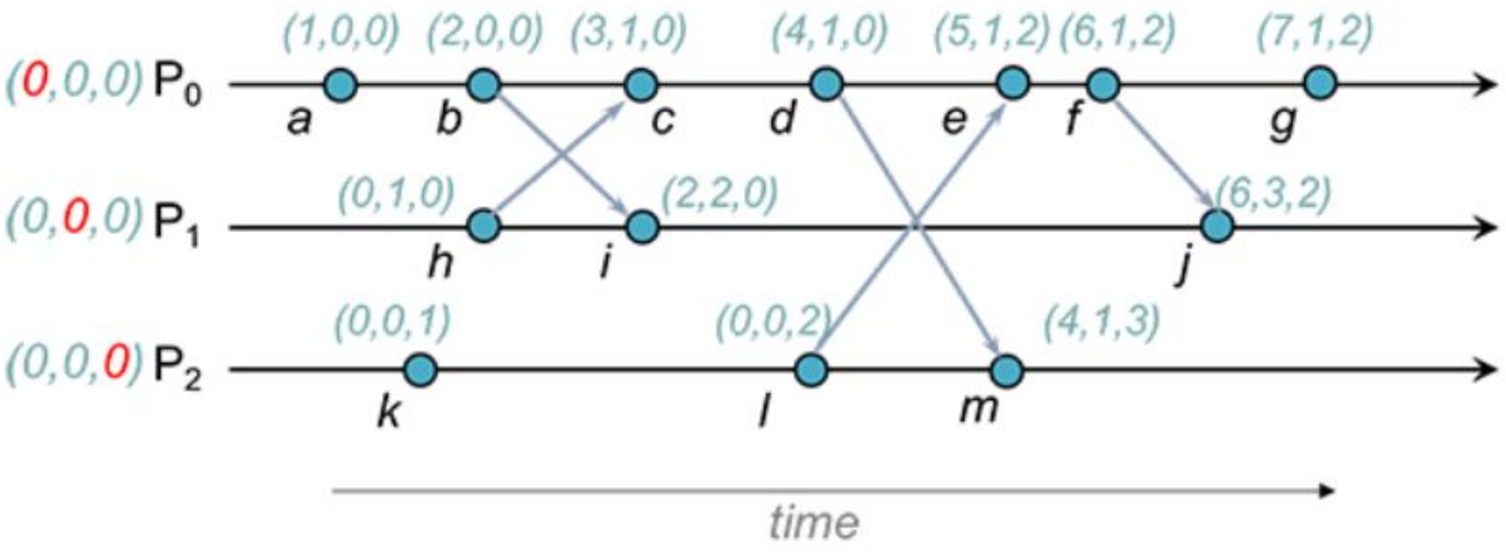
Else: a, b is concurrent

Solve causality and concurrency

a [1, 0, 0] and l [0, 0, 2] :   a,l is concurrent

k [1, 0, 0] and g [7, 1, 2] :   k->g

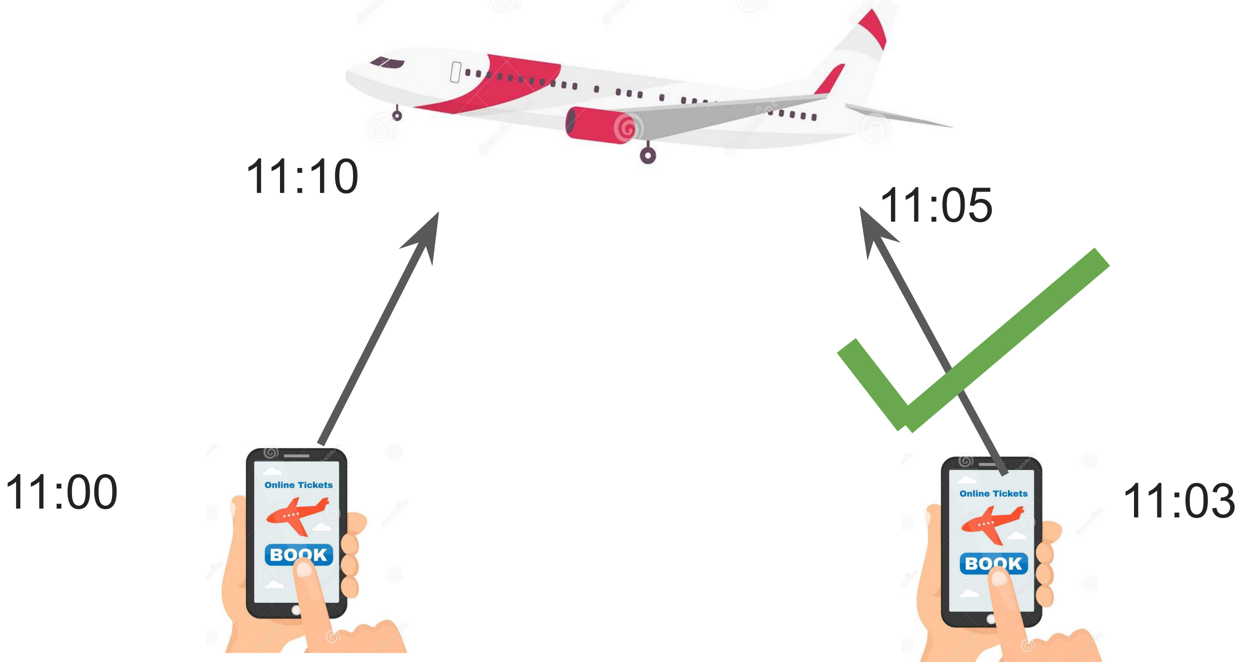
i [2, 2, 0] and e[5, 1, 2] :   i,e is concurrent



Drawbacks of Logic Clock II

“anomalous behavior”:

Logic clock is based on the interaction between each distributed node. It will fail when precedence relationship is based on information external to the system.

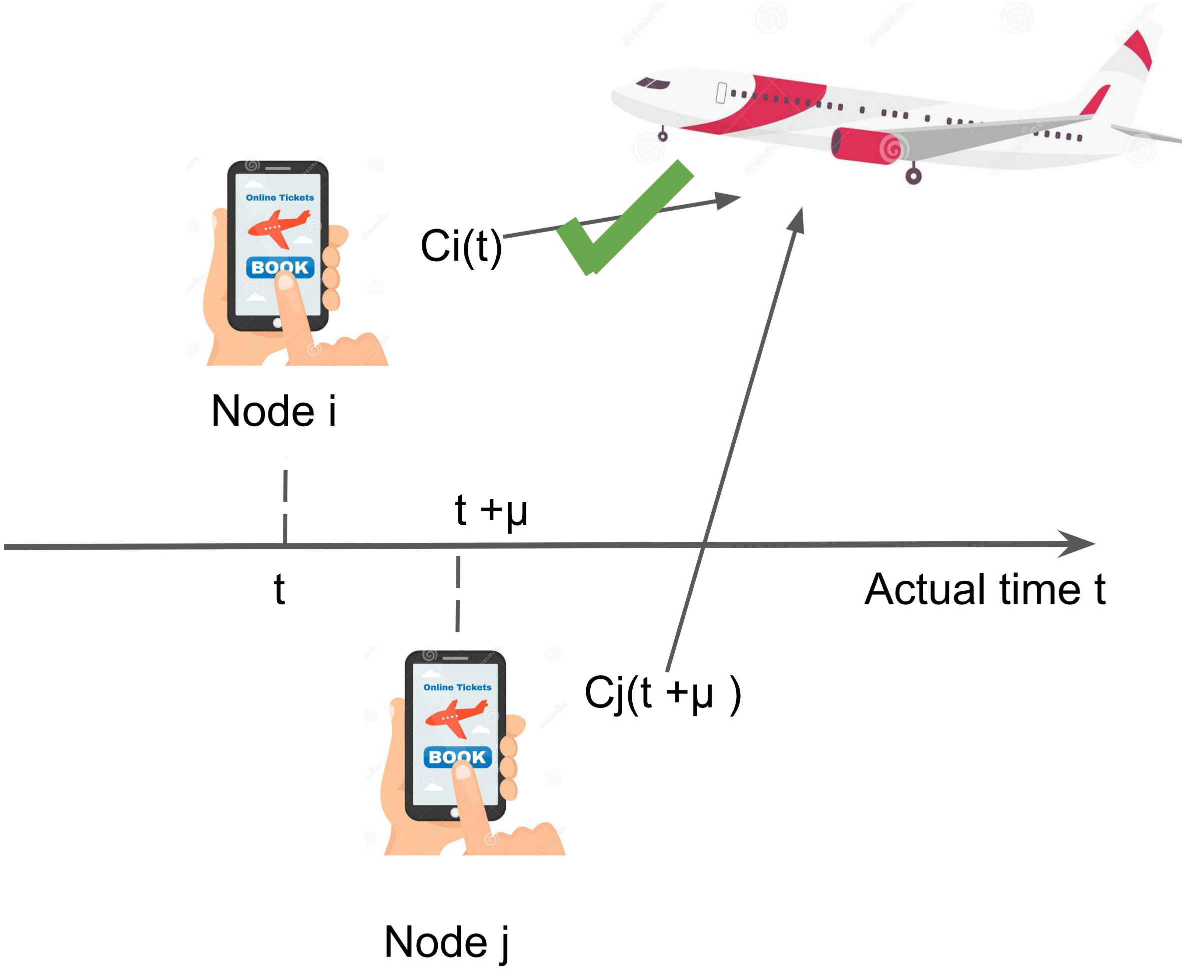


Physical Clocks distributed system:

Requirement:

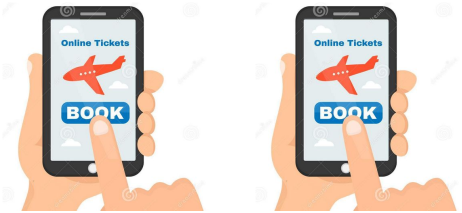
Cj(t+μ) > Ci(t)

μ is a type of “tolerance”

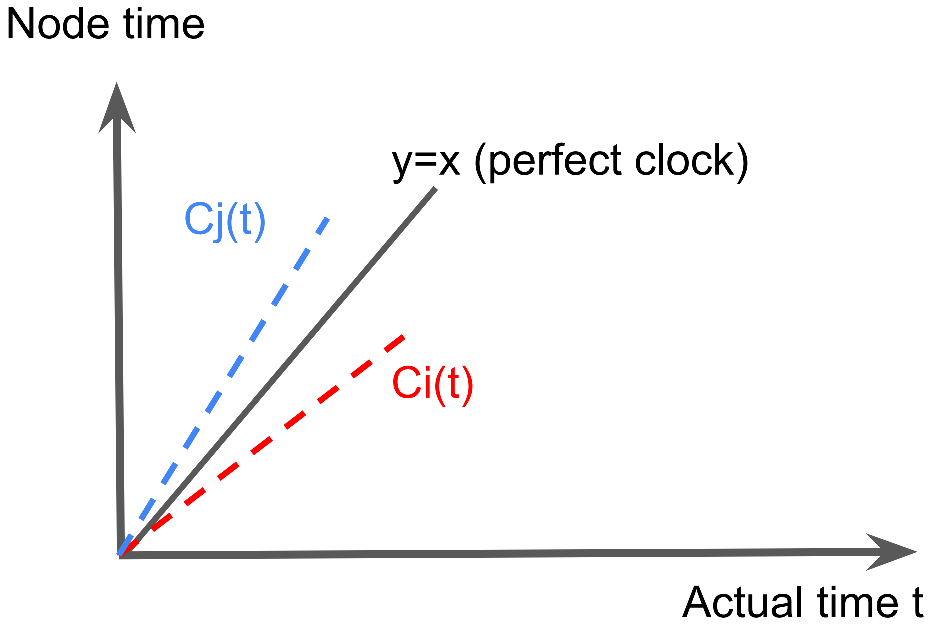


To achieve the requirement

Clock drifting: $dC_i(t)/dt \neq 1$



Node i   Node j



Clock condition 1:

$$|dC_i(t)/dt - 1| < \kappa << 1$$

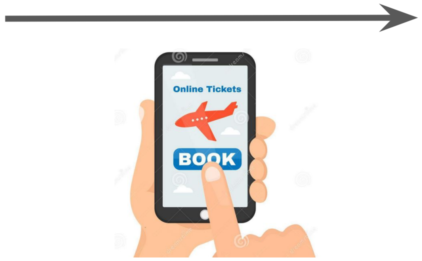
Clock condition 2:

$$|C_i(t) - C_j(t)| < \varepsilon$$

To ensure clock condition 2:

Rule 1: No messages received:

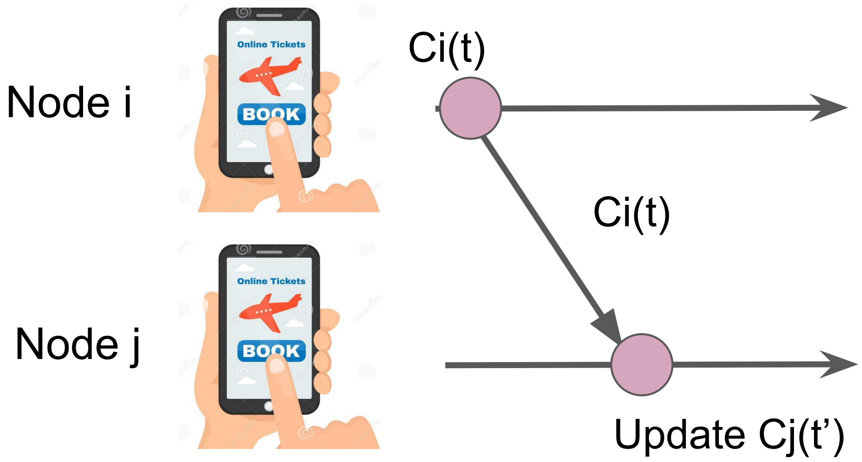
Ci(t) keep increasing as dCi(t)/ dt



Node i

Rule 2: Node i sends message at t with local time Ci(t)

Node j receives message at t’, and update
Cj(t’) = max{Cj(t’-0), Ci(t) + μm}
(μm is min transmission delay)



Discussion Questions

- What are the drawbacks of the Vector Clock compared to Lamport Clock?

  Ans: it requires additional memory to store the vectors and extra bandwidth to send the vector.

Discussion Questions

- What are some use cases for logical clocks? How are they different from the given use cases for physical clocks? Give specific examples of how it is helpful.

  Ans: The system only care about the order of node interaction, other than the order based external information.

  A. Mutual exclusion service such resource allocation, Data read/write scheduling

  B. State machine transition system

  C. Some e-commercial system: Amazon's Dynamo

Discussion Questions

- Assume we had perfect physical clocks. Describe some examples of systems that could take advantage of this and how it is helpful.

  Ans: A perfect physical clock would help to solve the anomalous behaviours as mentioned in the paper where ordering is determined based on information external to the system. Also, it is needed in a system where passage of time matters.

  A. An airline reservation system (With a physical clock, exact time that customers initially sent their reservation requests.

  B. A GPS system would be more accurate if equipped with a perfect physical clock since exact time difference is needed to compute accurate speed, and fewer satellites are needed in return.