Time, Clocks, and the Ordering of Events in a Distributed System

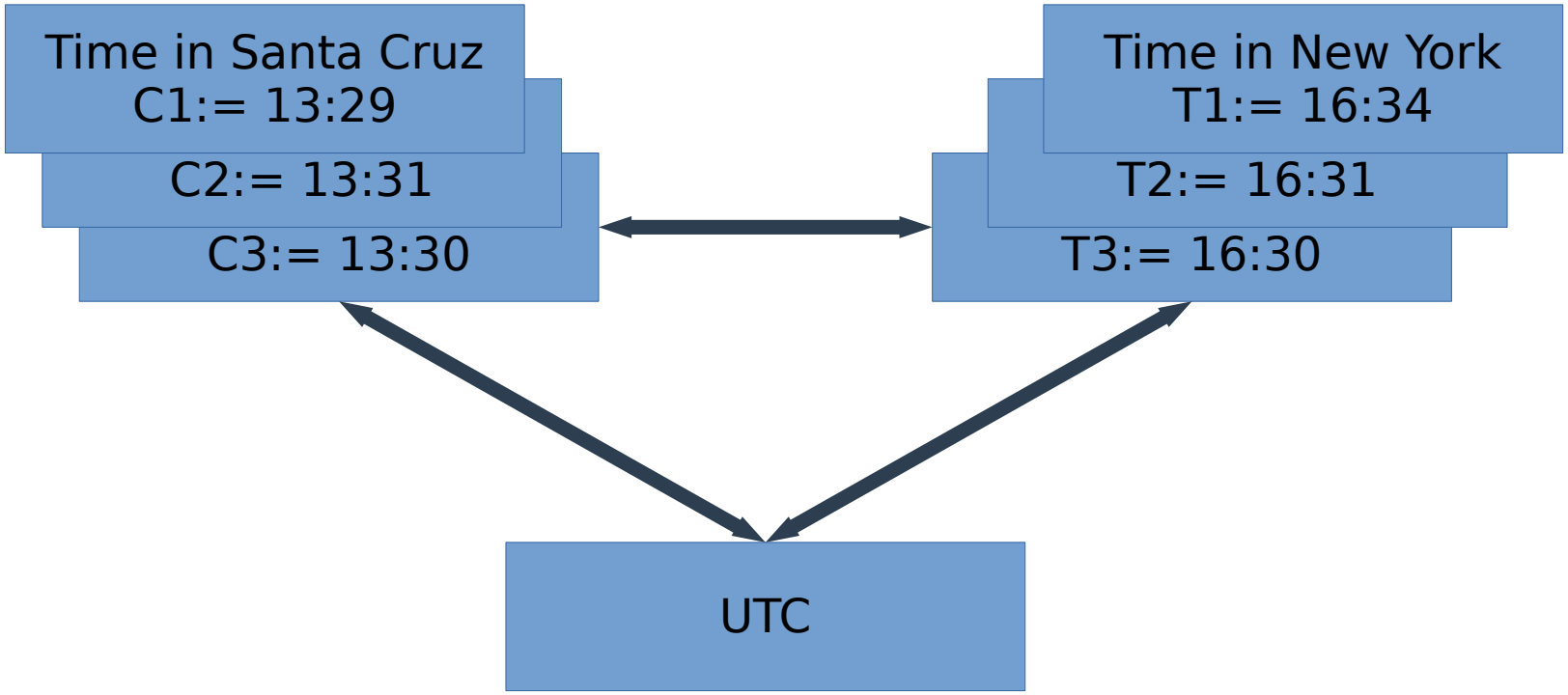
Author: Leslie Lamport
Presenter: Abhishek A. Singh

The problem

- **How do we order events in a distributed system?**

- **What are the problems with physical clocks?**

Physical clocks: The problems



- **Synchronization and coordination**
- **Skew (affects all measuring instruments)**

Proposed solution

- **Abandon clock (physical)!**

Ideas proposed in the paper

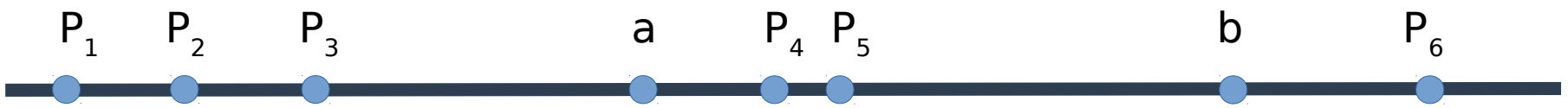
- **Partial ordering of events**
  - (→ operator)
- **Logical clock**
- **Total ordering based on the logical clock**

Partial ordering

- **Better understood from the perspective of “Logical Implication”**
- **Let A and B be two events, then**
  - A → B : A “happens before” B
  - If A→ B, and if B is true, A cannot be false.
- **It does not mean A and B are causally linked.**
  - It is possible but don’t bet on it.

What is a single process?

- **Single threaded process***
- **Events in a process form a sequence**
- **‘a’ occurs before ‘b’ in this sequence if ‘a’ happens before ‘b’**



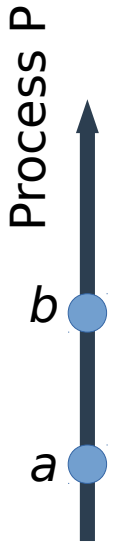
* Not explicitly stated in the paper

Definition of “→” relation

- **The “→” relation on a set of events of a system is the *smallest relation* satisfying three conditions (next).**
- **What is the meaning of ‘smallest relation’ here?**

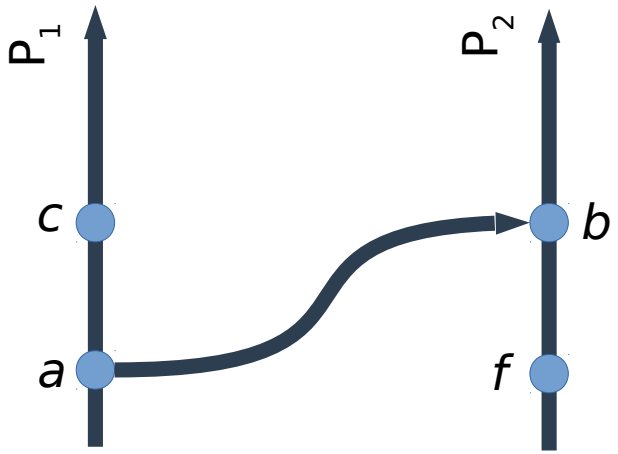
Condition #1

- **Let *a* and *b* be two events**
- **If *isSameProcess(a,b)* == true AND *a* comes before *b*, then *a* → *b***



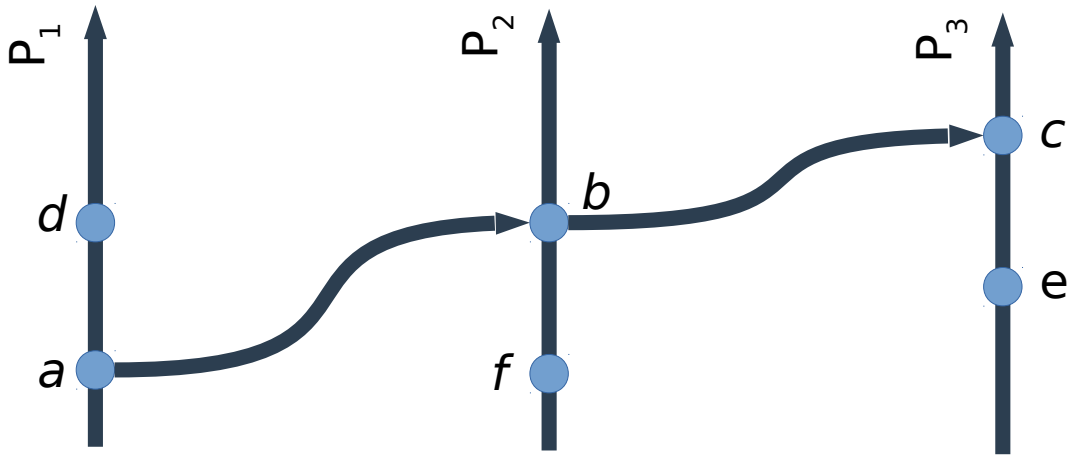
Condition #2

- **If**
  - *a* is the sending of a message by P_1
- **And**
  - *b* is the receipt of the same message by P_2
- **Then, *a* → *b***



Condition #3

- **If**
  - $a \rightarrow b$
- **And**
  - $b \rightarrow c$
- **Then, $a \rightarrow c$**

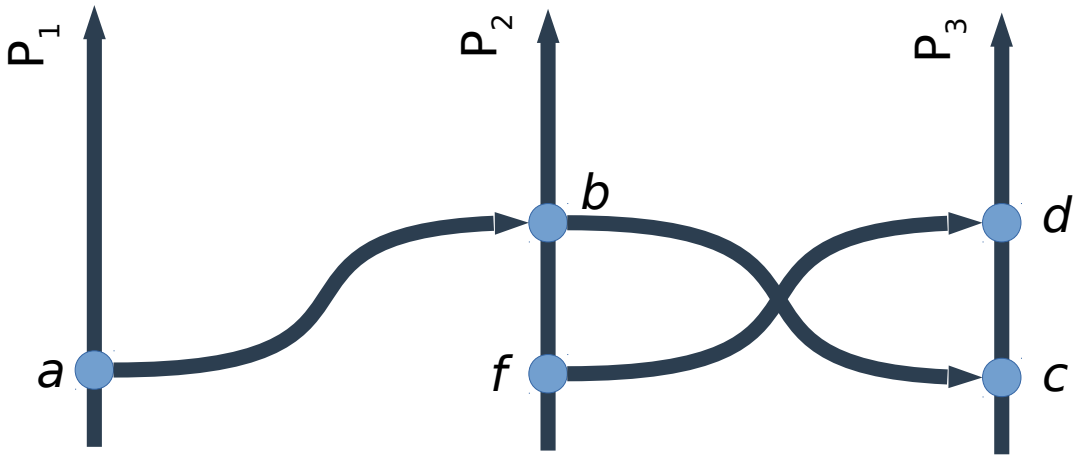


Concurrency condition

“Two distinct events are said to be *concurrent* if $a \not\rightarrow b$ and $b \not\rightarrow a$”

Question

- **What’s going on here?**



Bonus Question

a→b, b→c, c→d, f→d, f→b

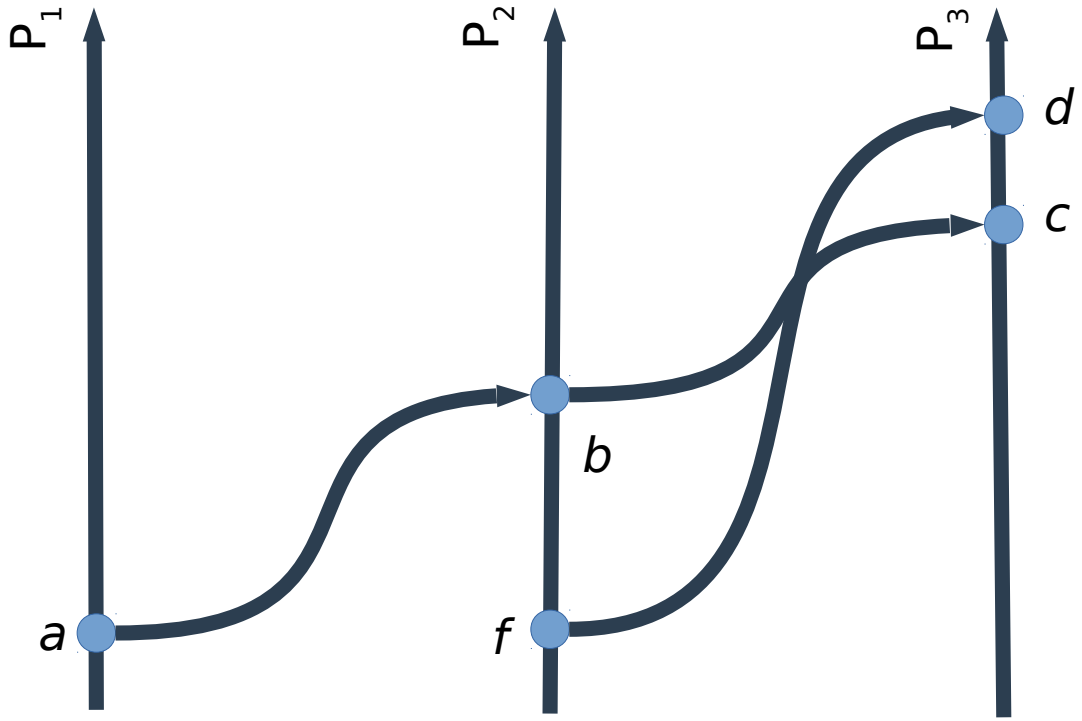
a→c

a→d

f→b

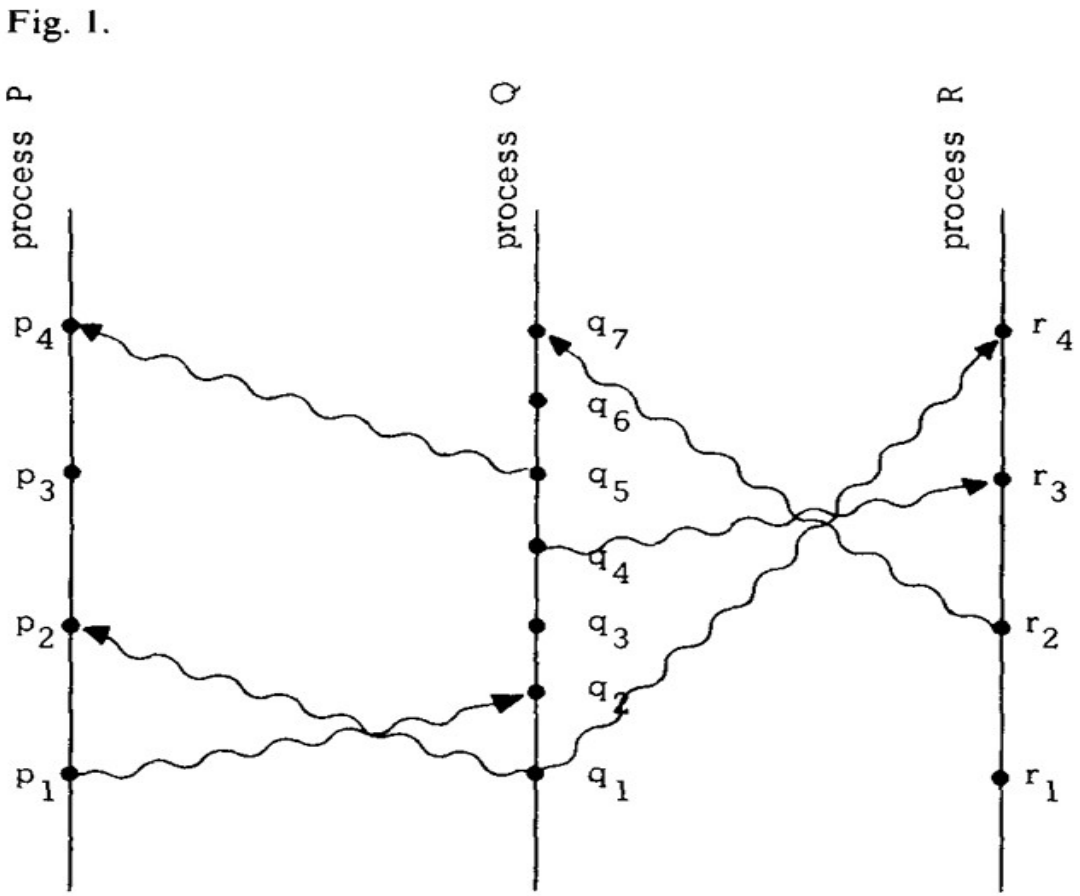
f→d

f→c



Identify the relationship

- p_1 ? q_2 ? r_4
- p_1 ? q_1 ? r_1
- p_3 ? q_3
- p_4 ? q_6
- q_7 ? r_3
- q_3 ? r_3



The BIG idea

- **“We should be able to determine if a system performed correctly by knowing only those events which *did* occur, without knowing which events *could* have occurred.”**

Logical clock

- **TLDR; Assign a number to every event in the collection of processes**
- **Define a Clock function C_i which assigns a number $C_i(a)$ to every event a in process P_i**

Clock condition

For any event *a* and *b*,

If

$a \rightarrow b$

Then,

$C(a) < C(b)$

- **If *a* happened before *b* then *a* should be assigned a smaller number than *b***
- **Converse is not true:**
  - If *a* is assigned a smaller number than *b* that does not mean a happened before *b*

Conditions of satisfaction

Clock condition holds if,

[Condition #1]

- ***a* and *b* are events in process P_i AND *a* comes before *b*, then $C_i(a) < C_i(b)$**

[Condition #2]

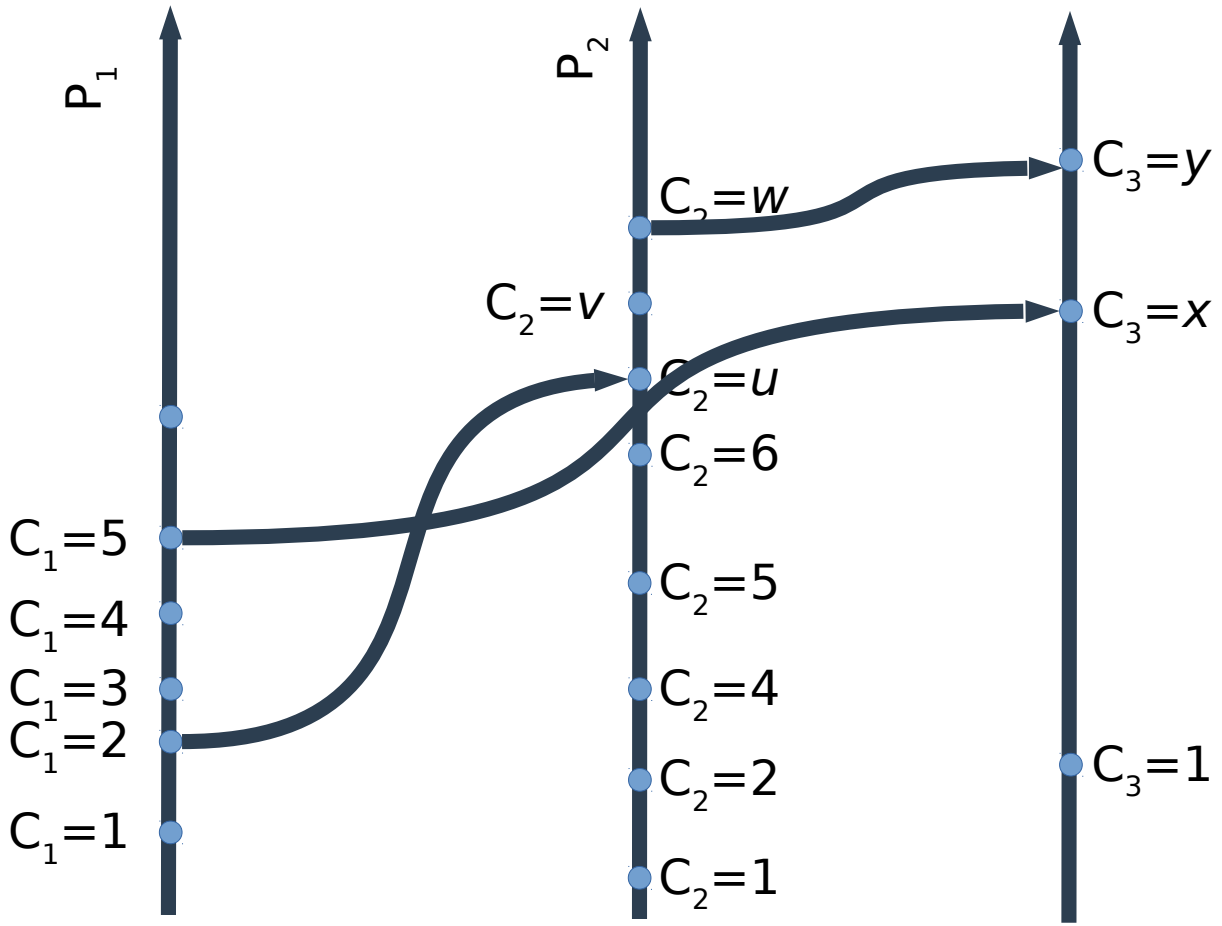
- ***a* denotes sending of a message by P_i and *b* denotes receipt of a message by P_j, then $C_i(a) < C_j(b)$**

Implementation Rules

- **To meet condition #1**
  - Each process P_i increments C_i between any two successive events
- **To meet condition #2**
  - Messages exchanged contain Timestamps
    - T_m, time at which message was sent
      - $T_m = C_i(a)$
    - When P_j receives the message,
      - $C_j = max(C_j, T_m) + 1$

Exercise

- **What are the values of *u,v,w,x,y*?**



Total order

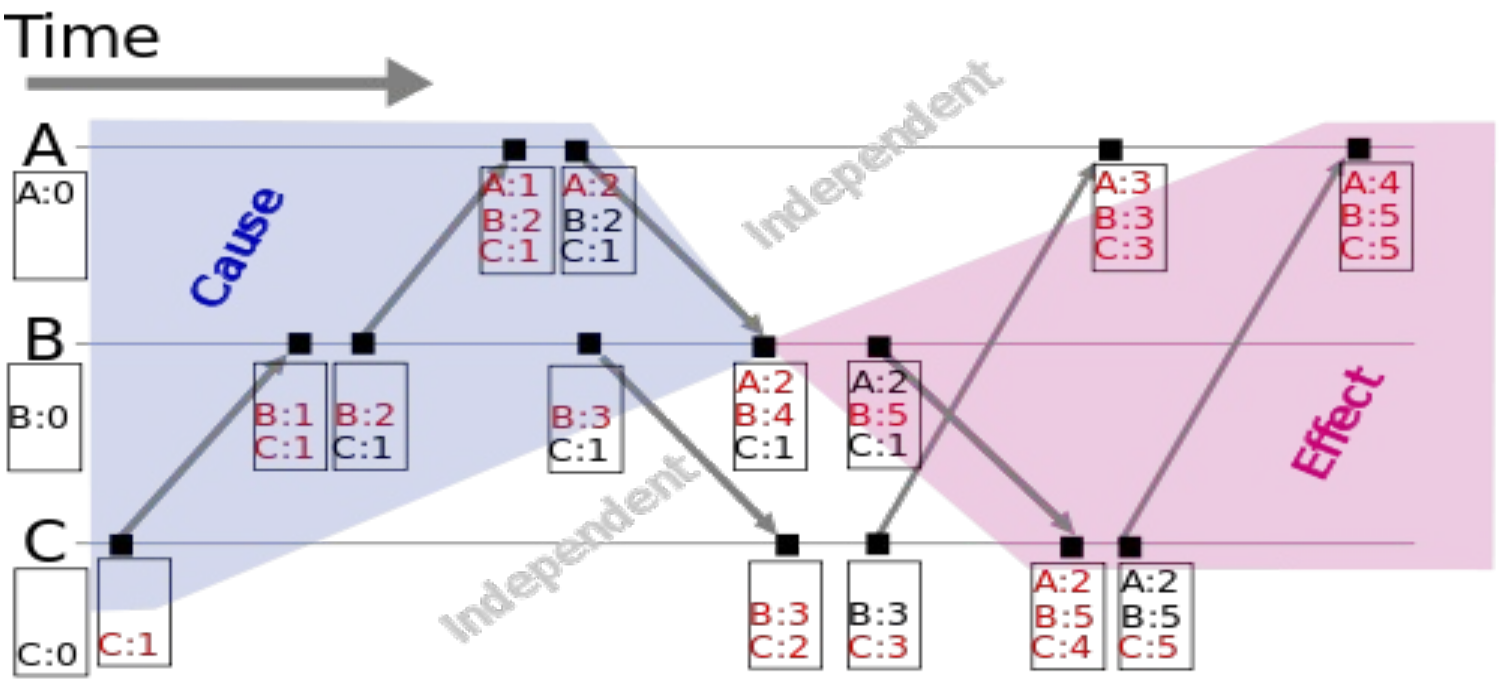
- **Defined by the “=>” relation**
- **A total order on the set of all system events**
  - Order the events by the time at which they occur.
  - What if two events have the same time?
    - Break ties using some property of the processes to enforce total order

Definition

- **Let events be**
  - a in P_i and b in P_j
- **Define: $a => b$**
  - IF
    - $C_i(a) < C_j(b)$
  - *OR*
    - $C_i(a) = C_j(b)$ **AND** $P_i \triangleleft P_j$
      - Where '$\triangleleft$' is some arbitrary total ordering of the processes

So what exactly is the difference?

Vector Clocks



$$VC(x) < VC(y) \iff \forall z[VC(x)_z \leq VC(y)_z] \wedge \exists z'[VC(x)_{z'} < VC(y)_{z'}]$$

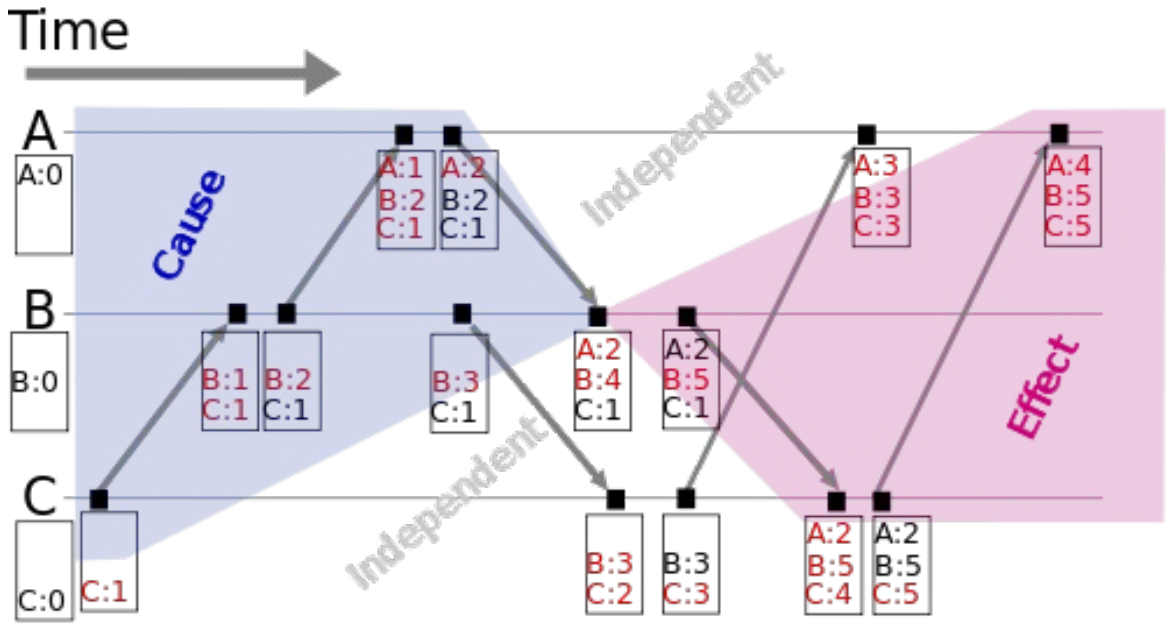
Properties of Vector Clocks:

1. if VC(a) < VC(b), then a → b | Additionally, if a→b, then a *causally* preceeds b
2. if VC(a) < VC(b), then
   ¬(VC(b) < VC(a))
3. if a→b and b→c, then a→c

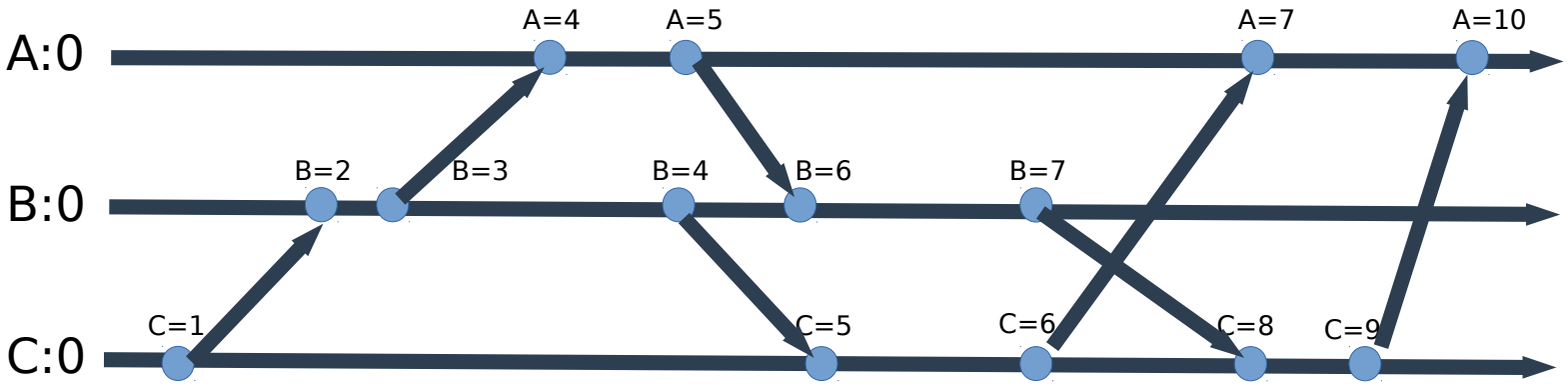
Source: Wikipedia (https://en.wikipedia.org/wiki/Vector_clock)

Comparisons with Vector Clock

Vector clock diagram
(Stolen from Wikipedia)



Equivalent Lamport Clock Space-time Diagram



What are the differences?

Did I say “Abandon clock!”?

Sorry, spoke too soon. :-P

Anomalous Behavior

- **Total ordering messes with real Time.**
- **Strong Clock Condition uses physical clocks**

- **Can Physical clocks be eliminated in distributed system?**
  - Do vector clocks help resolve the anomalous behavior described in the paper?
- **Other questions?**