

CS 60002: Distributed Systems

T4: More on Clocks!

Department of Computer Science
and Engineering



INDIAN INSTITUTE OF TECHNOLOGY
KHARAGPUR



Sandip Chakraborty
sandipc@cse.iitkgp.ac.in

What We Have Learnt So Far ...

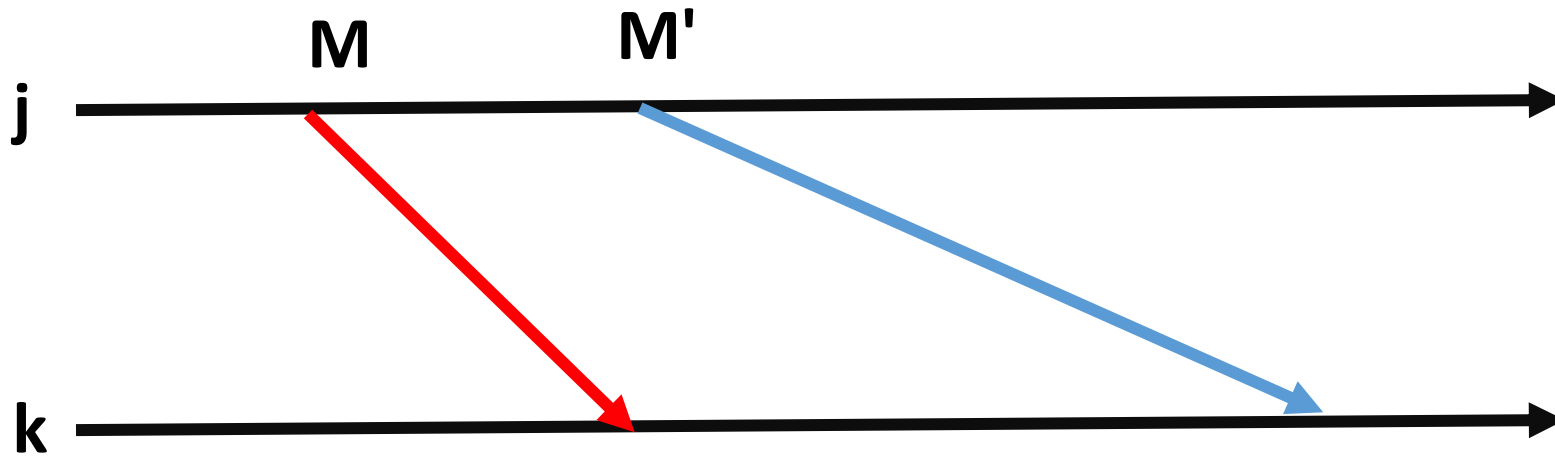
- Common and Distributed Knowledge
- Global States
- Logical Clocks
- Distributed Checkpoint
- Consistency and Consistent Cuts

Message Delivery - FIFO

- For a process k
 - $\text{Send}(k, M) \Rightarrow$ Process k sends a message M
 - $\text{Receive}(k, M) \Rightarrow$ Process k receives a message M
- Consider process j sends messages M and M' to process k , and $\text{Send}(k, M) \rightarrow \text{Send}(k, M')$
- **FIFO delivery** ensures, $\text{Receive}(k, M) \rightarrow \text{Receive}(k, M')$

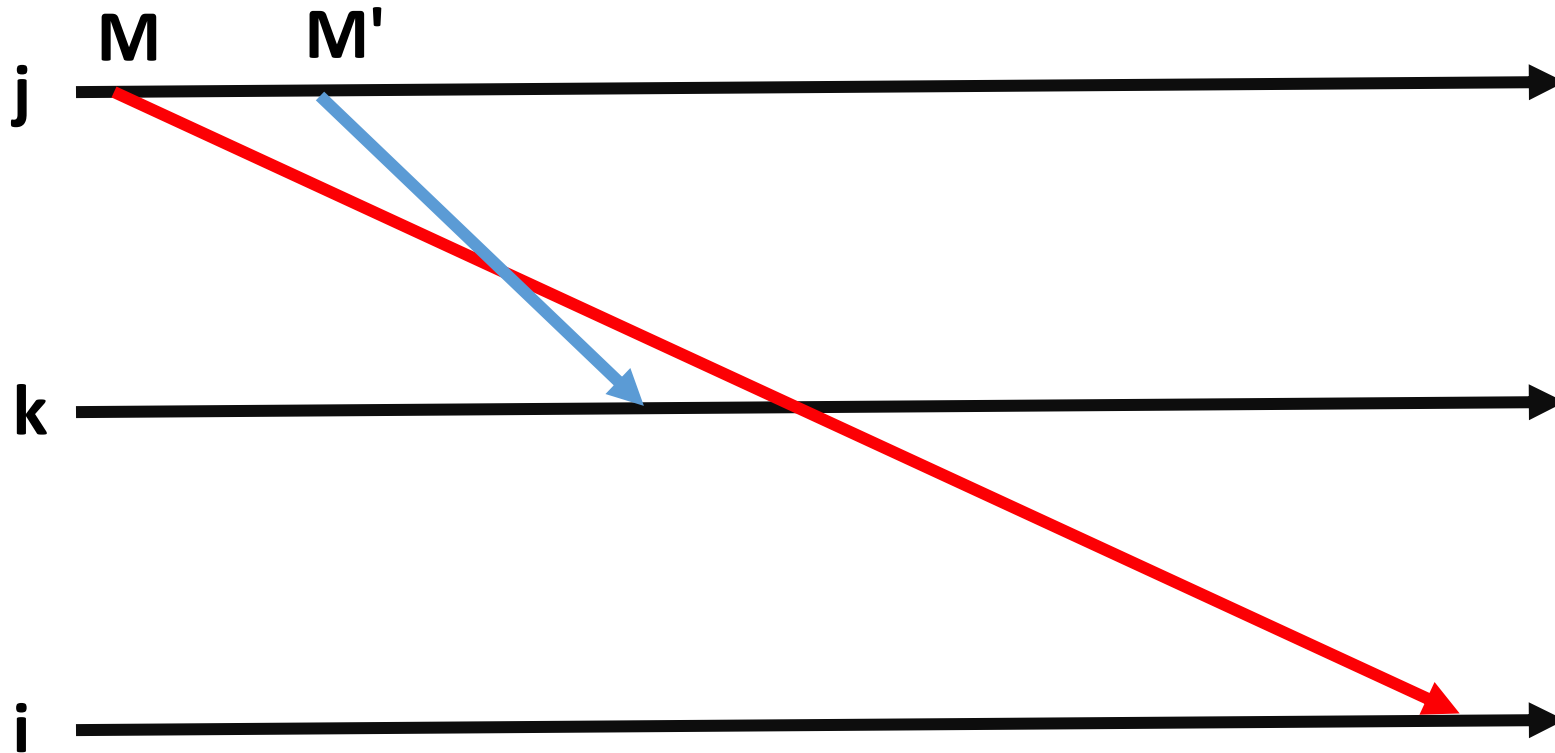
Message Delivery - FIFO

- **FIFO delivery** ensures, $\text{Receive}(k, M) \Rightarrow \text{Receive}(k, M')$



Message Delivery - FIFO

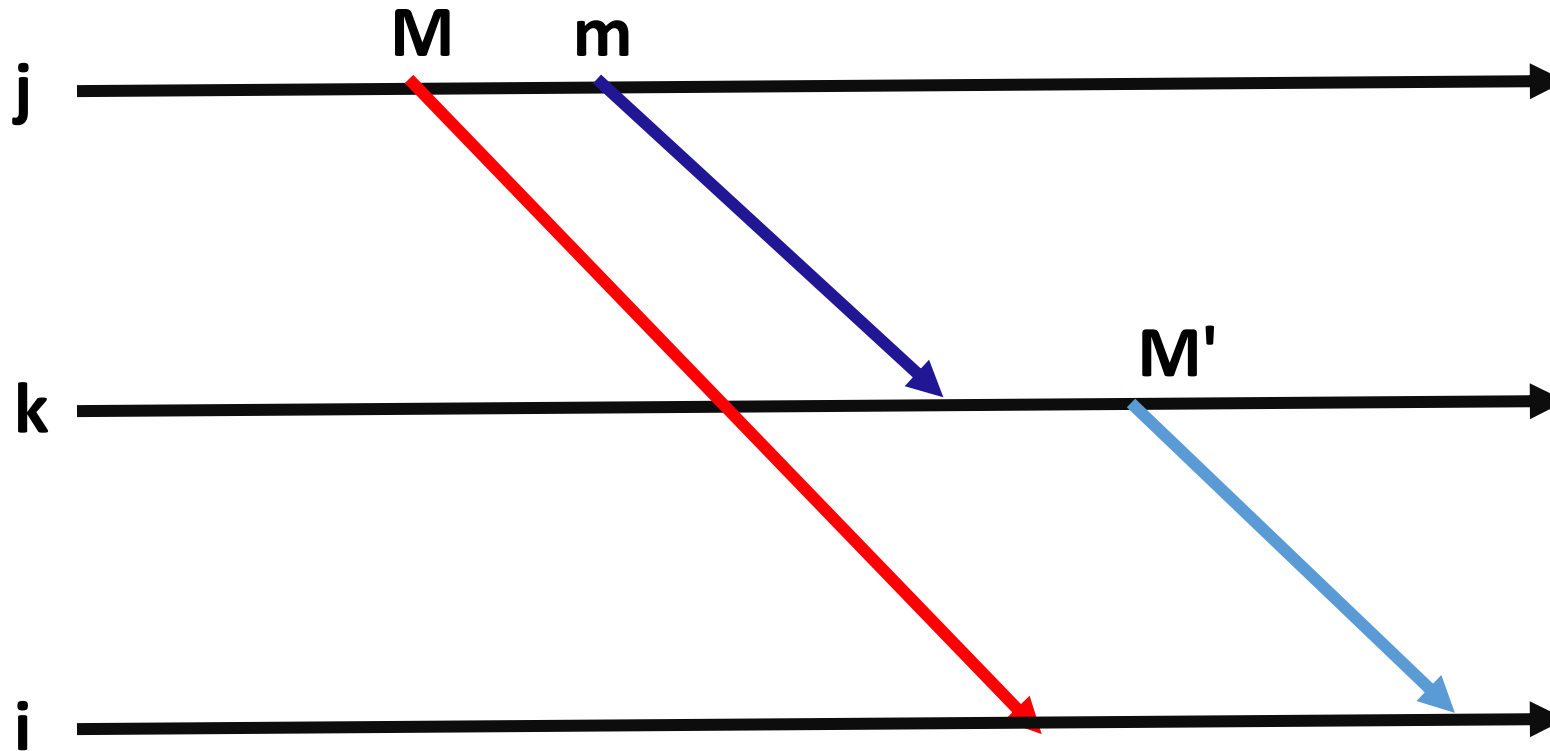
- Can you characterize FIFO across three processes?



Message Delivery - Causal

- For a process k
 - $\text{Send}(k, M) \Rightarrow$ Process k sends a message M
 - $\text{Receive}(k, M) \Rightarrow$ Process k receives a message M
- Consider process i sends message M and process j sends message M' to process k , and
 $\text{Send}(i, M) \rightarrow \text{Send}(j, M')$
- **Causal delivery** ensures, $\text{Receive}(k, M) \rightarrow \text{Receive}(k, M')$
 - The messages are causally related, even if they are sent by different processes

Causal Delivery

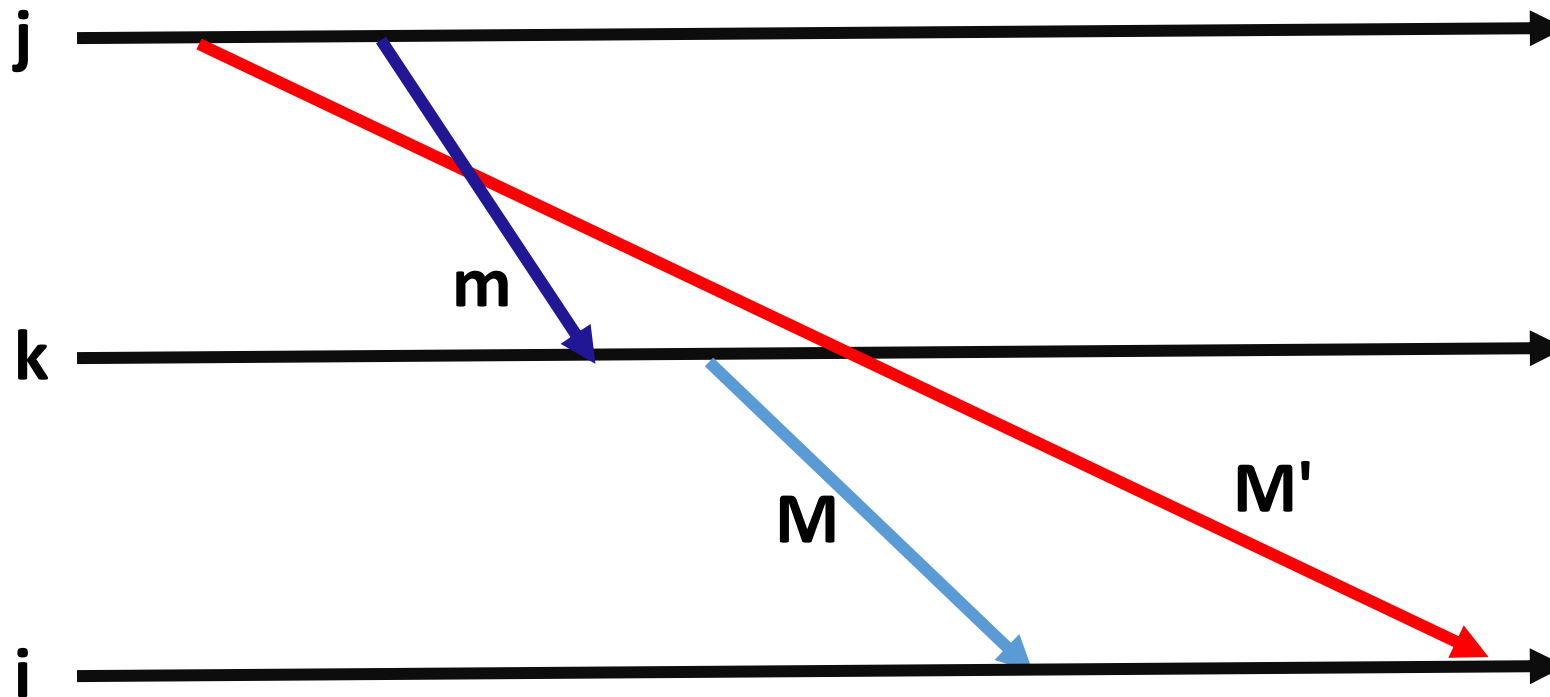


Causal Delivery

- Does FIFO between each pair of processes ensure Causal?

Causal Delivery

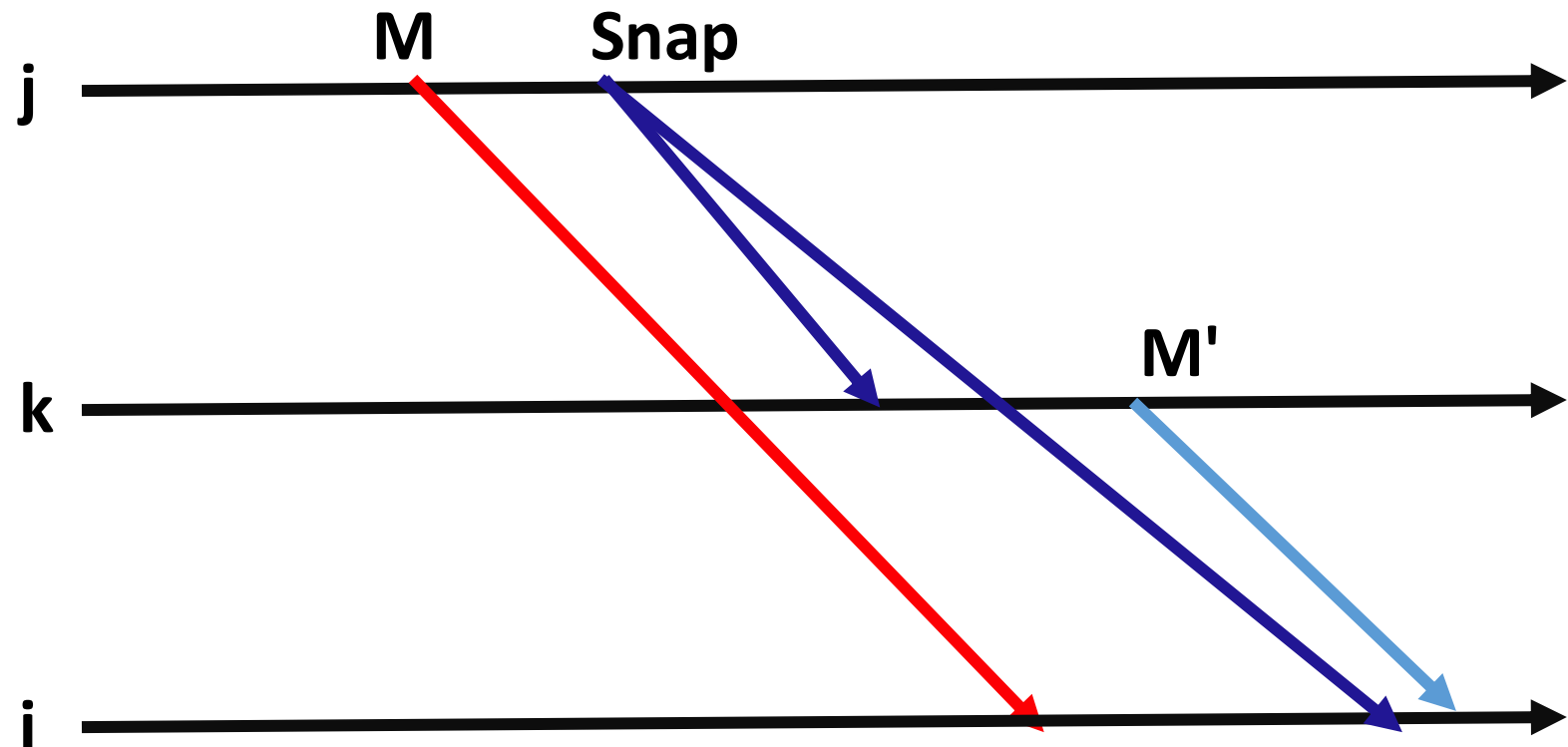
- Does FIFO between each pair of processes ensure Causal?



FIFO between (*j*, *k*), (*k*, *i*), (*j*, *i*) but not Causal

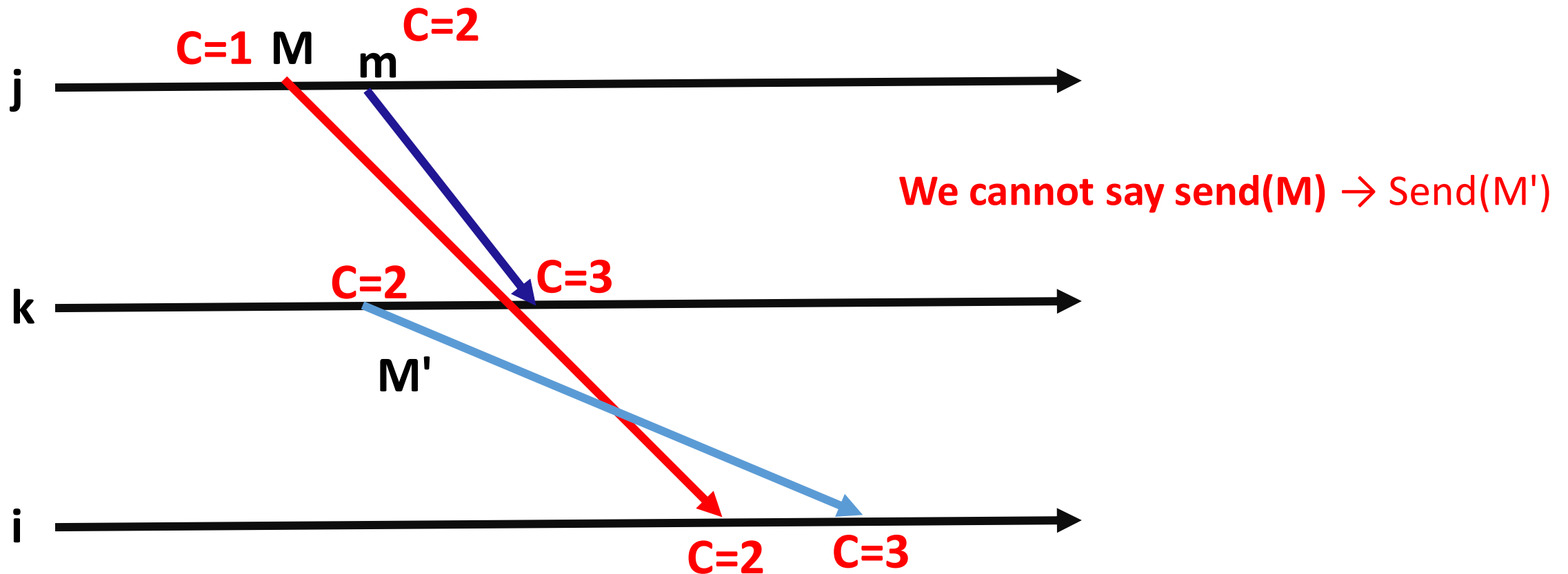
Causal Delivery and Consistency

- If a process uses a delivery rule satisfying causal delivery, then all of its observations will be consistent.
 - The process always receives messages in the causal order in which they have been sent → not possible that the receipt of the message is logged but the sent has not been logged



How Do We Ensure Causal Delivery

- Lamport's clock does not tell anything about the ordering of the events based on their clock – **is its major limitation**

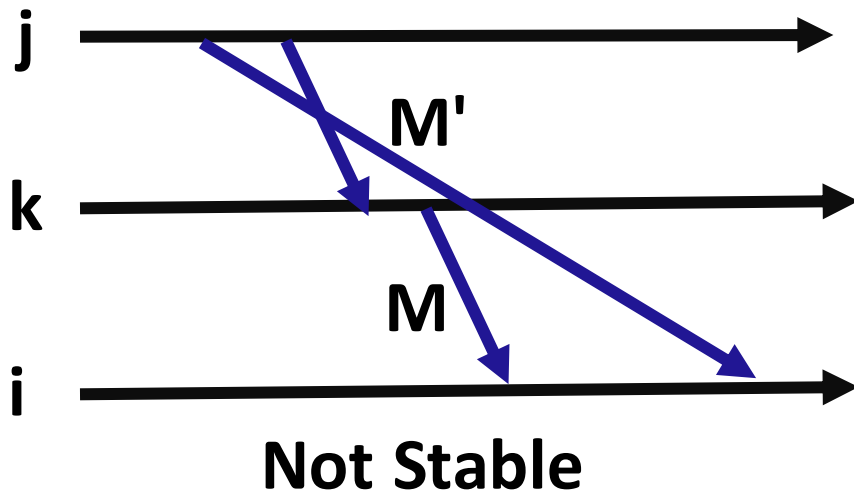


Stable Messages

- Let a message M be received at a process p with timestamp $TS(M)$. Let logical clock be used to generate the timestamp.
- Message M is called to be stable, if process p receives no other message M' after the receipt of p such that $TS(M') < TS(M)$

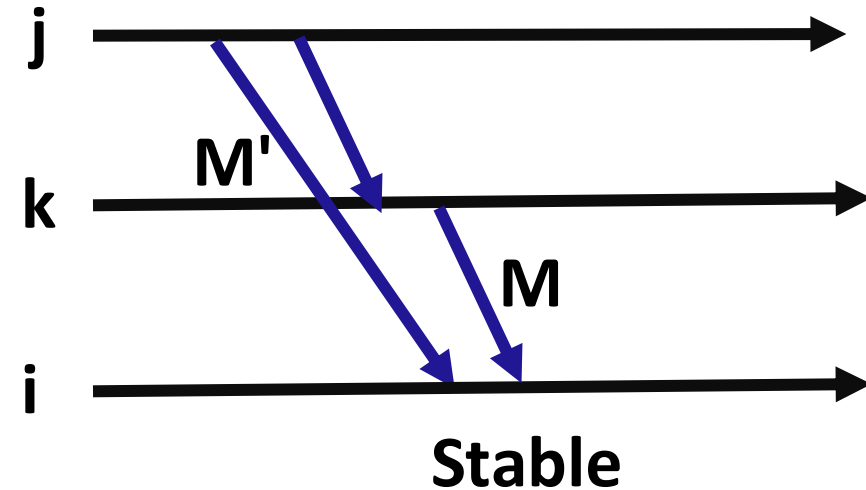
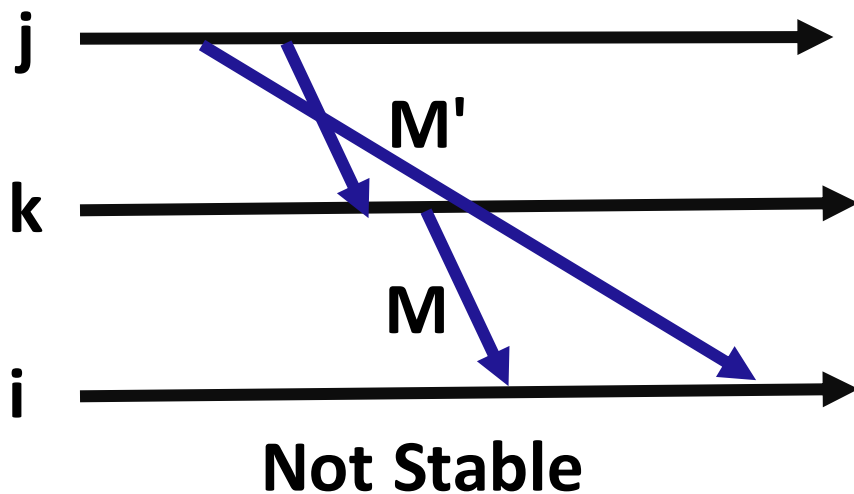
Stable Messages

- Let a message M be received at a process p with timestamp $TS(M)$. Let logical clock be used to generate the timestamp.
- Message M is called to be stable, if process p receives no other message M' after the receipt of p such that $TS(M') < TS(M)$



Stable Messages

- Let a message M be received at a process p with timestamp $TS(M)$. Let logical clock be used to generate the timestamp.
- Message M is called to be stable, if process p receives no other message M' after the receipt of p such that $TS(M') < TS(M)$

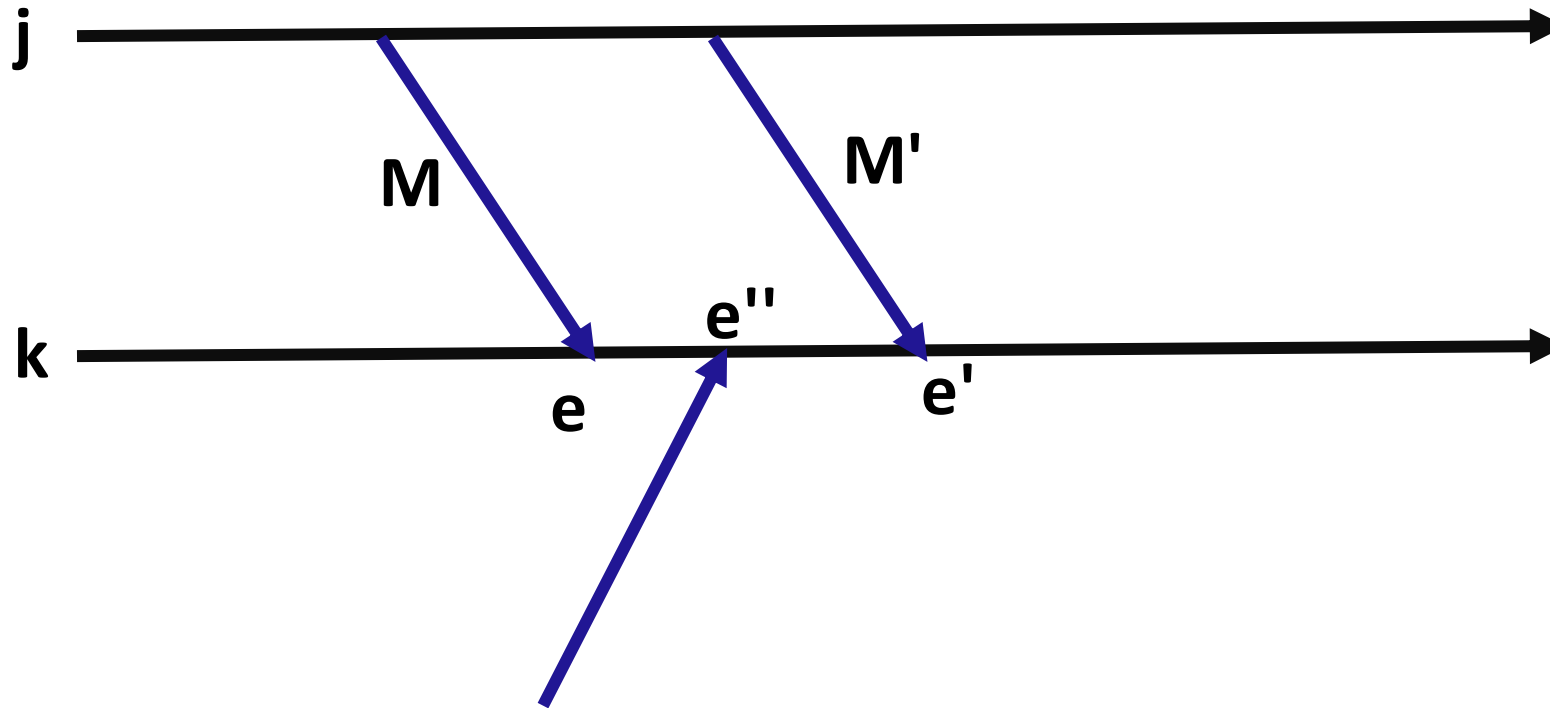


Stable Messages

- Let a message M be received at a process p with timestamp $TS(M)$. Let logical clock be used to generate the timestamp.
- Message M is called to be stable, if process p receives no other message M' after the receipt of p such that $TS(M') < TS(M)$
- **Causal delivery ensures that the message is stable.**

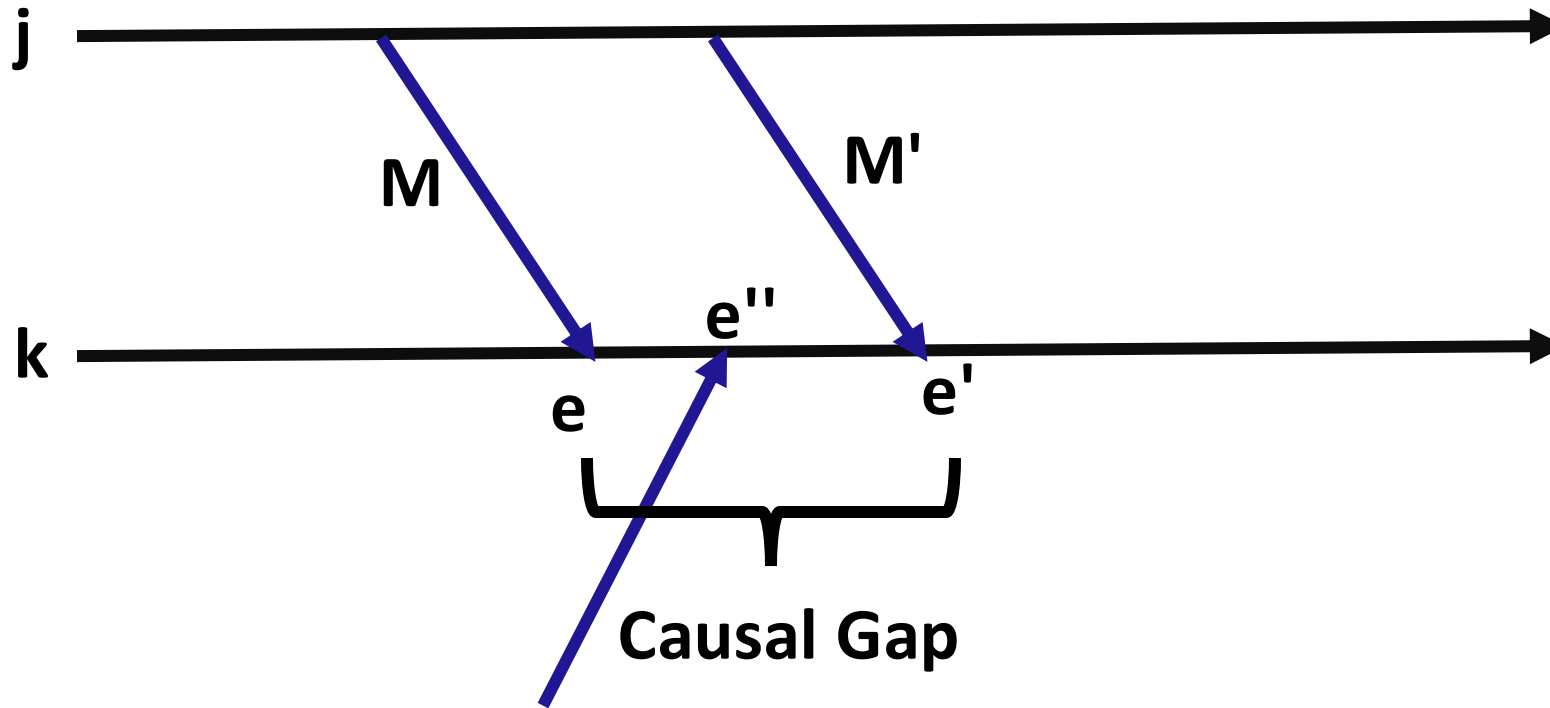
Gap Detection – Ensuring Causal Delivery

- Given two events e and e' along with their logical clock values $LC(e)$ and $LC(e')$ where $LC(e) < LC(e')$, determine whether some other event e'' exists such that $LC(e) < LC(e'') < LC(e')$



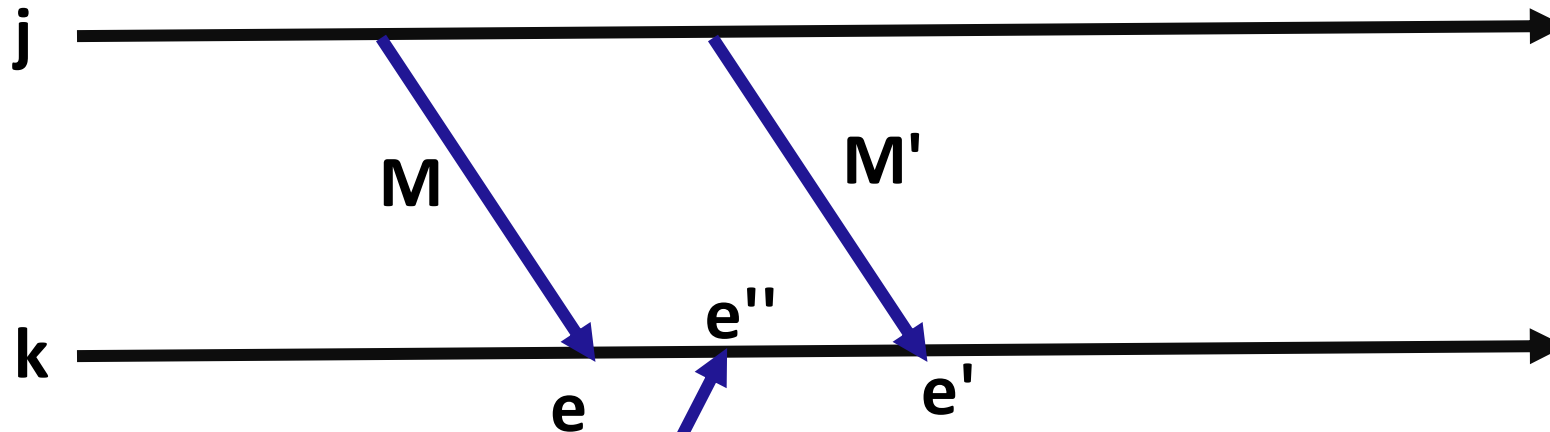
Gap Detection – Ensuring Causal Delivery

- Given two events e and e' along with their logical clock values $LC(e)$ and $LC(e')$ where $LC(e) < LC(e')$, determine whether some other event e'' exists such that $LC(e) < LC(e'') < LC(e')$



Gap Detection – Ensuring Causal Delivery

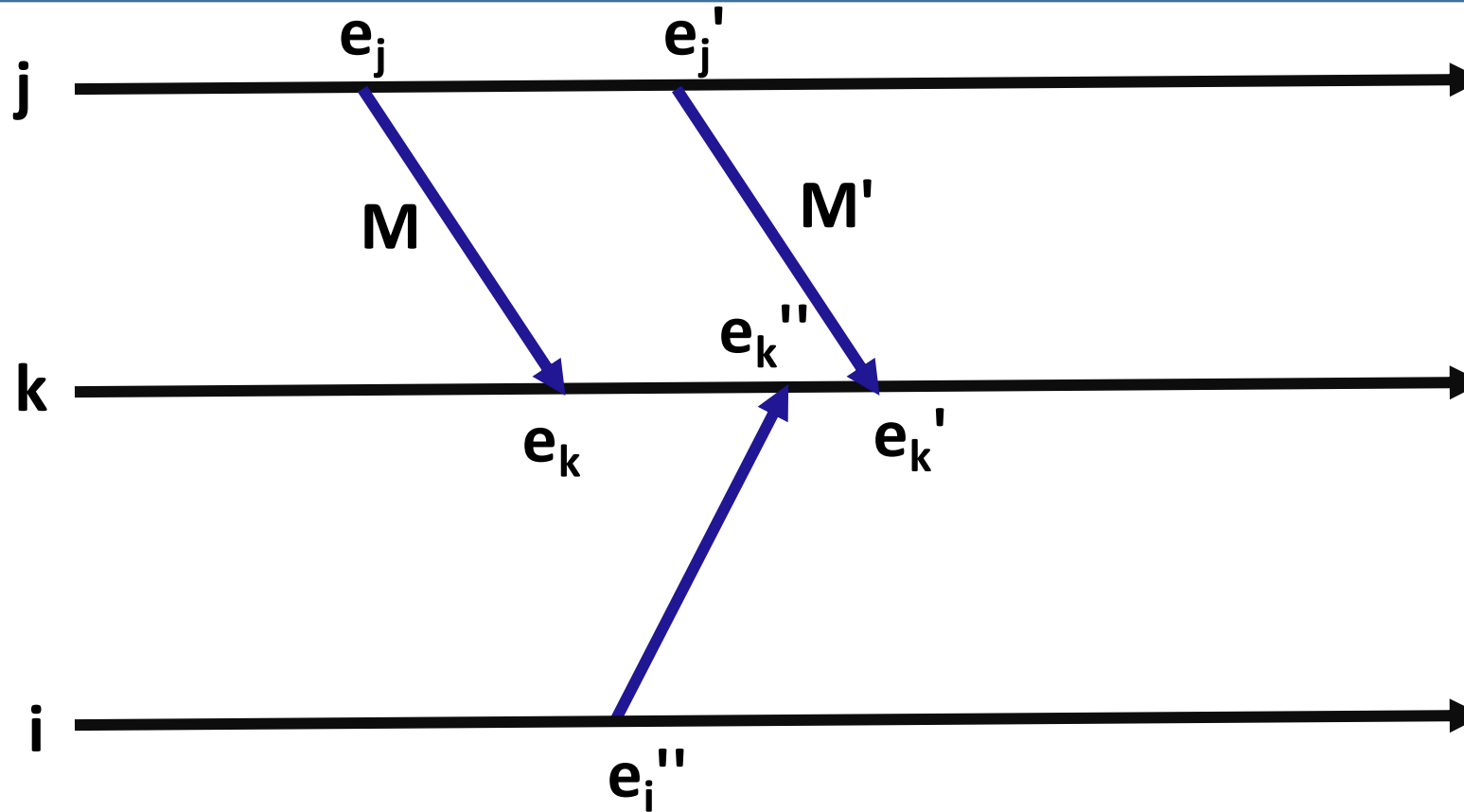
- Given two events e and e' along with their logical clock values $LC(e)$ and $LC(e')$ where $LC(e) < LC(e')$, determine whether some other event e'' exists such that $LC(e) < LC(e'') < LC(e')$



We need additional information apart from Lamport Clock to detect gaps

Ex. FIFO delivery of two consecutive messages between a pair of processes

The Causal Dilemma with Lamport's Clock



- With Lamport's clock, we really do not know whether $e_i'' \rightarrow e_j'$
 - Whether to log e_k'' or not, depends upon this causal relationship.

Requirement for Causal Delivery

- Given events e and e' that are causally related and their clock values, does there exist some other events e'' such that $e \rightarrow e'' \rightarrow e'$?
 - e'' falls in the causal gap between e and e'
- Lamport clock ensures, $e \rightarrow e' \Rightarrow LC(e) < LC(e')$
- We need a true clock (TC), $e \rightarrow e' \Leftrightarrow TC(e) < TC(e')$
- Method for generating true clocks:
 - Causal history (a brute force approach)
 - Vector clocks

Vector Clocks

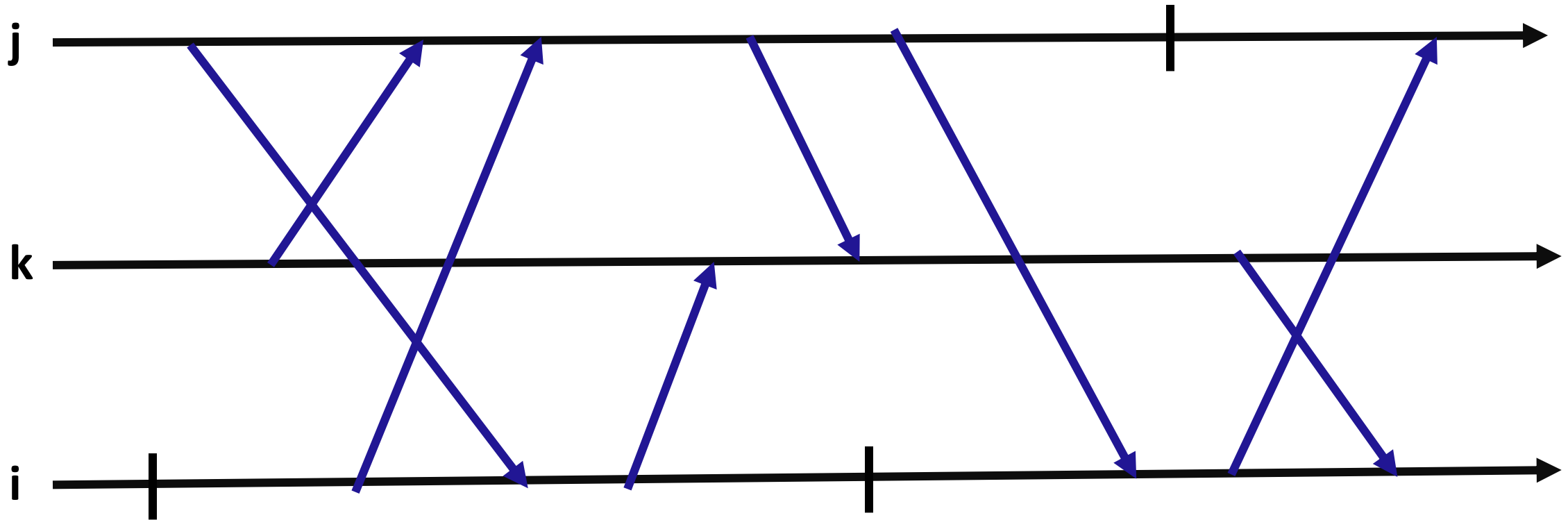
- Discovered independently by many researchers in many different contexts !
- Each process p_i maintains a local vector VC of natural numbers where $VC(e_i)$ denotes the vector clock value of p_i when it executes event e_i
- Each process initializes VC to contain all zeros; all messages contain a timestamp $TS(m)$

Reinhard Schwarz and Friedemann Mattern. **Detecting causal relationships in distributed computations: In search of the Holy Grail**. Technical Report SFB124-15/92, Department of Computer Science, University of Kaiserslautern, Kaiserslautern, Germany, December 1992

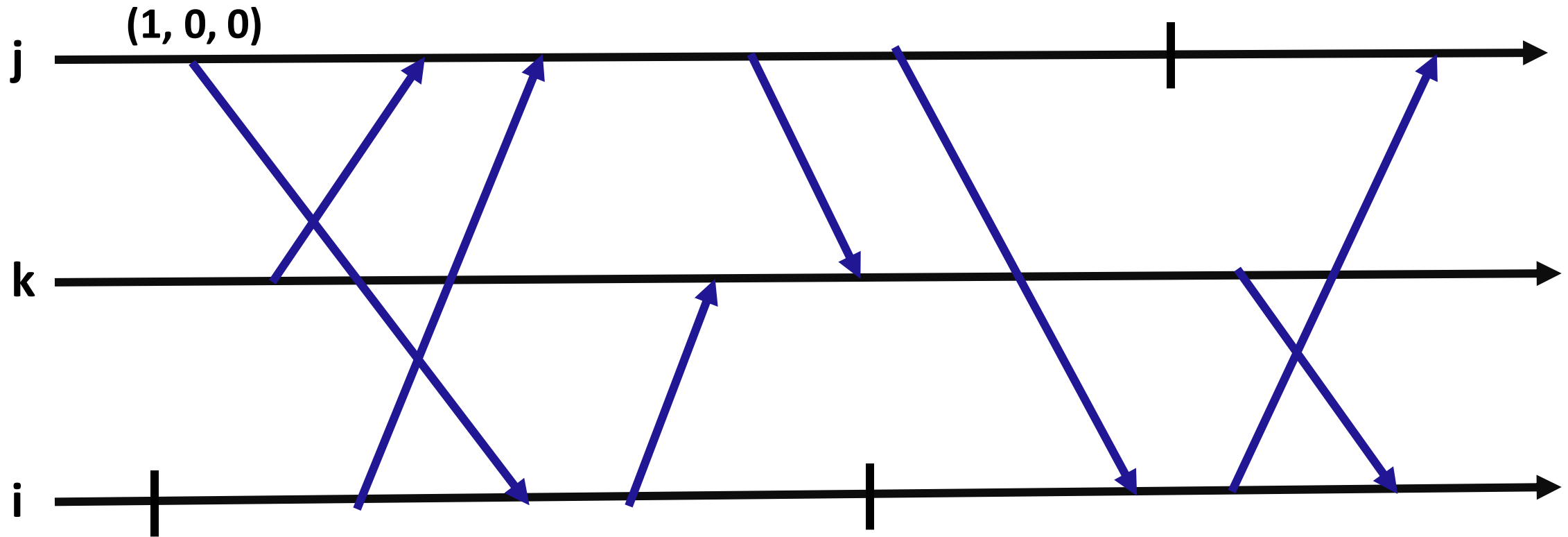
Vector Clocks – Update Rules

- $VC(e_i)[i] := VC[i] + 1$ if e_i is an internal or a send event
 - Internal or send events simply increment the local component of the vector clock
- $VC(e_i) := \max\{VC, TS(m)\}$ if $e_i = \text{receive}(m)$
 $VC(e_i)[i] := VC[i] + 1$
 - For a receive event, update the vector clock with the greater of local clock and the received timestamp, and then increment it

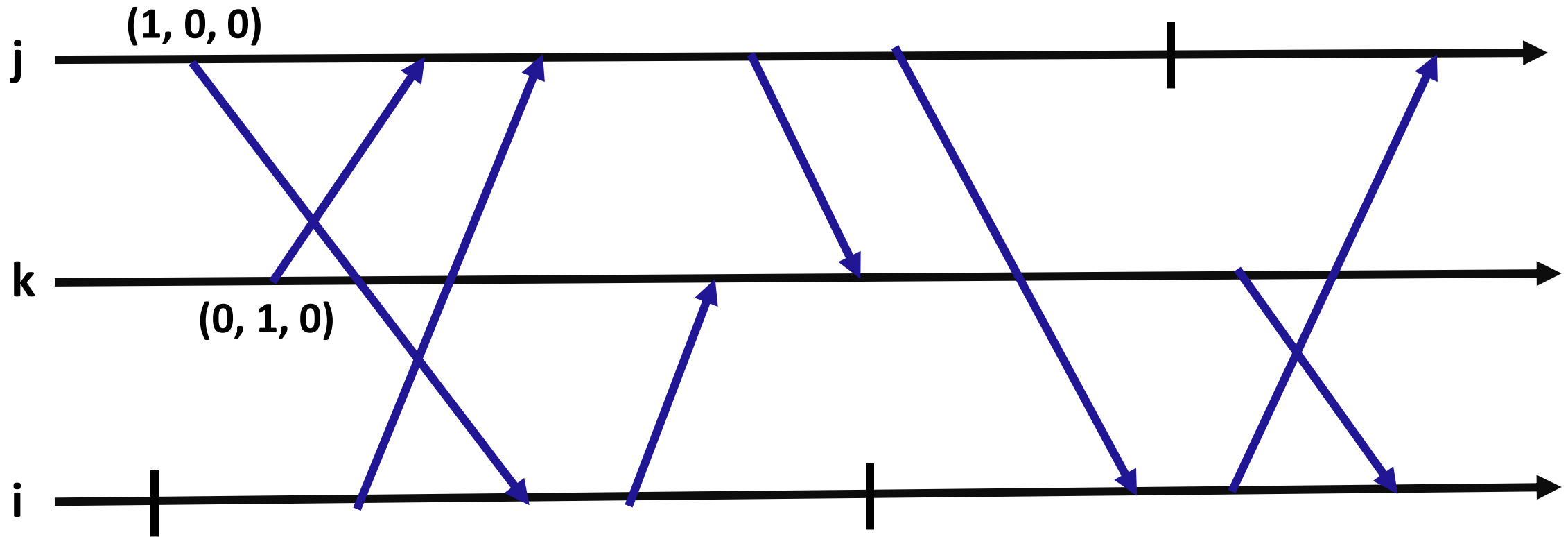
Vector Clock -- Visualization



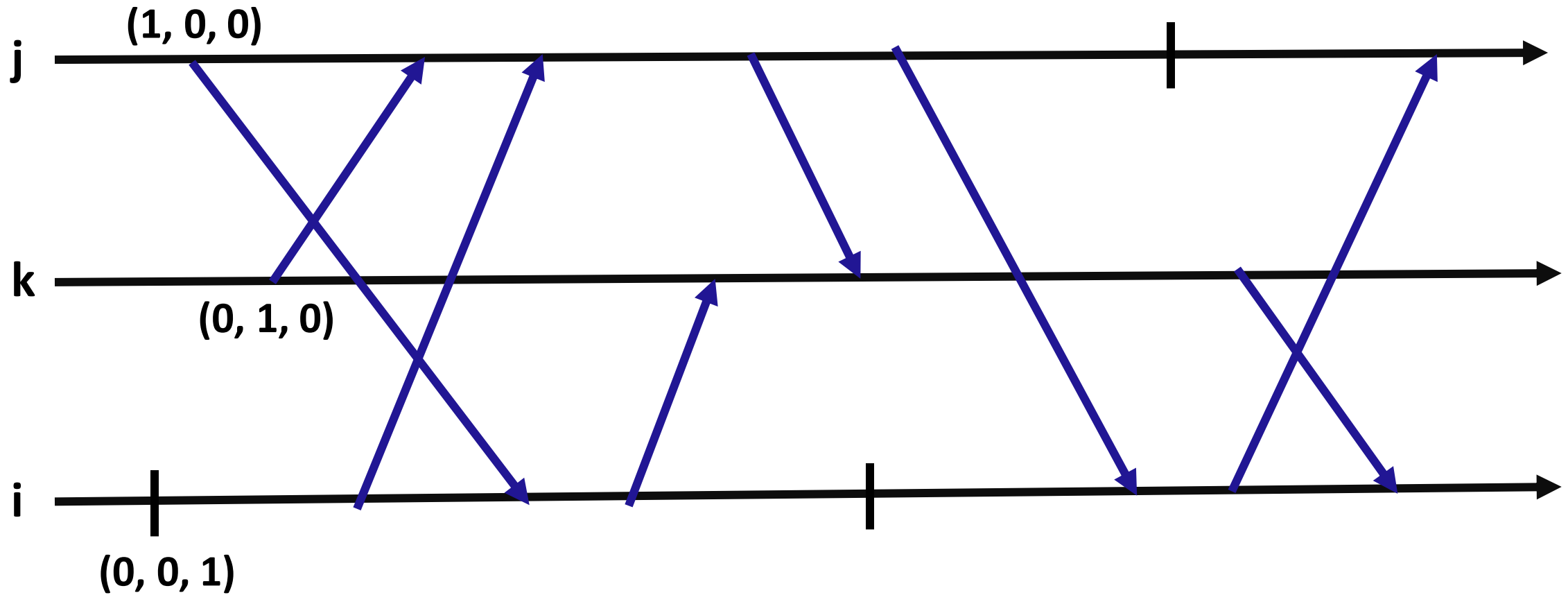
Vector Clock -- Visualization



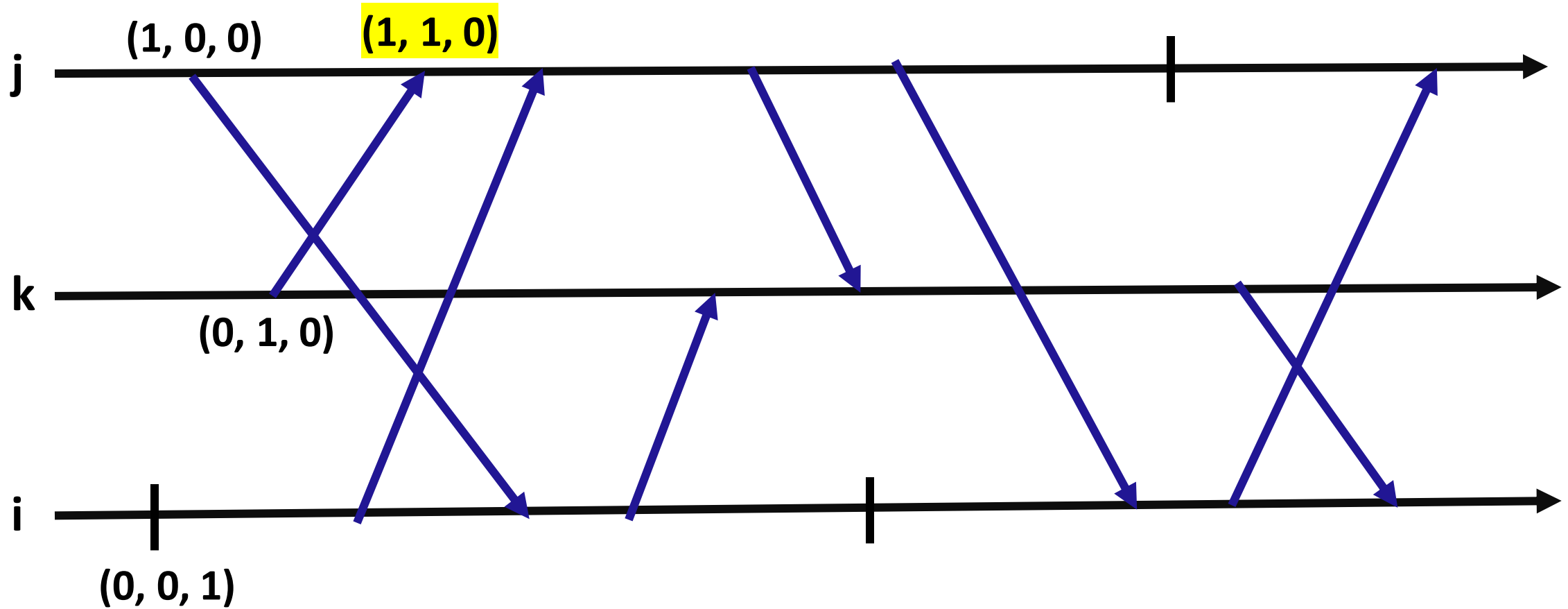
Vector Clock -- Visualization



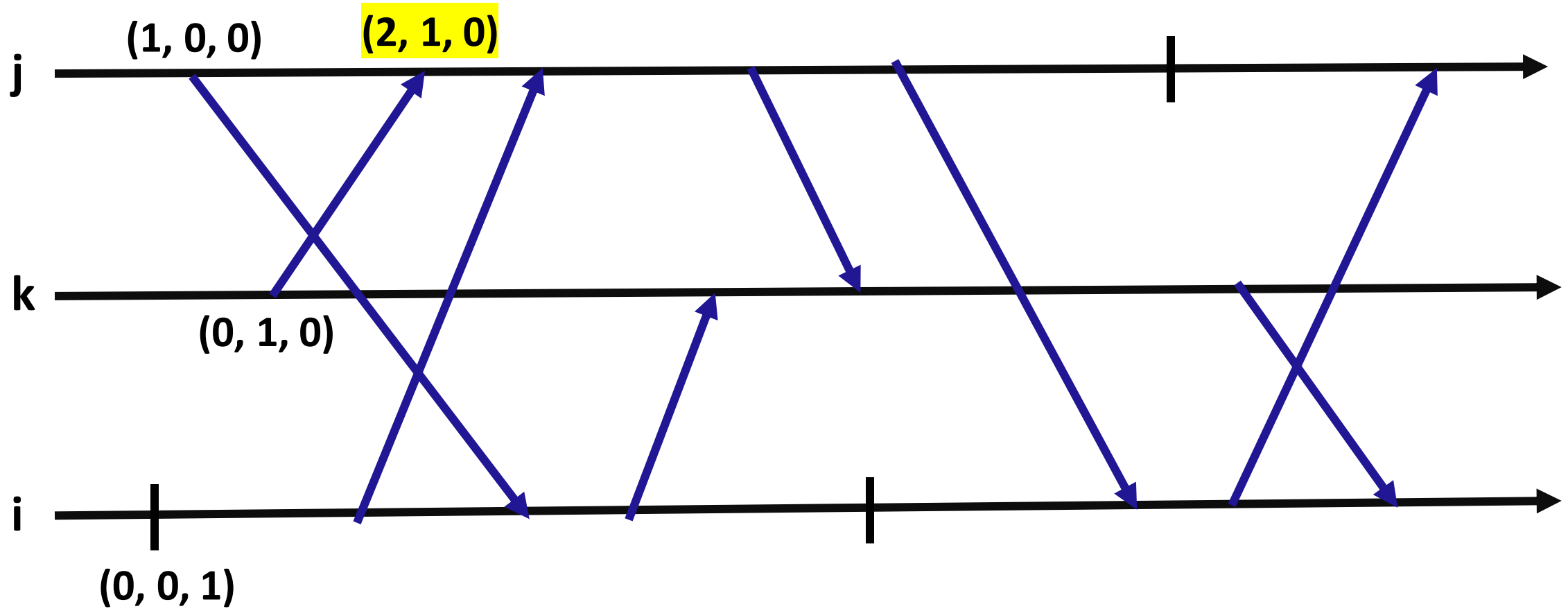
Vector Clock -- Visualization



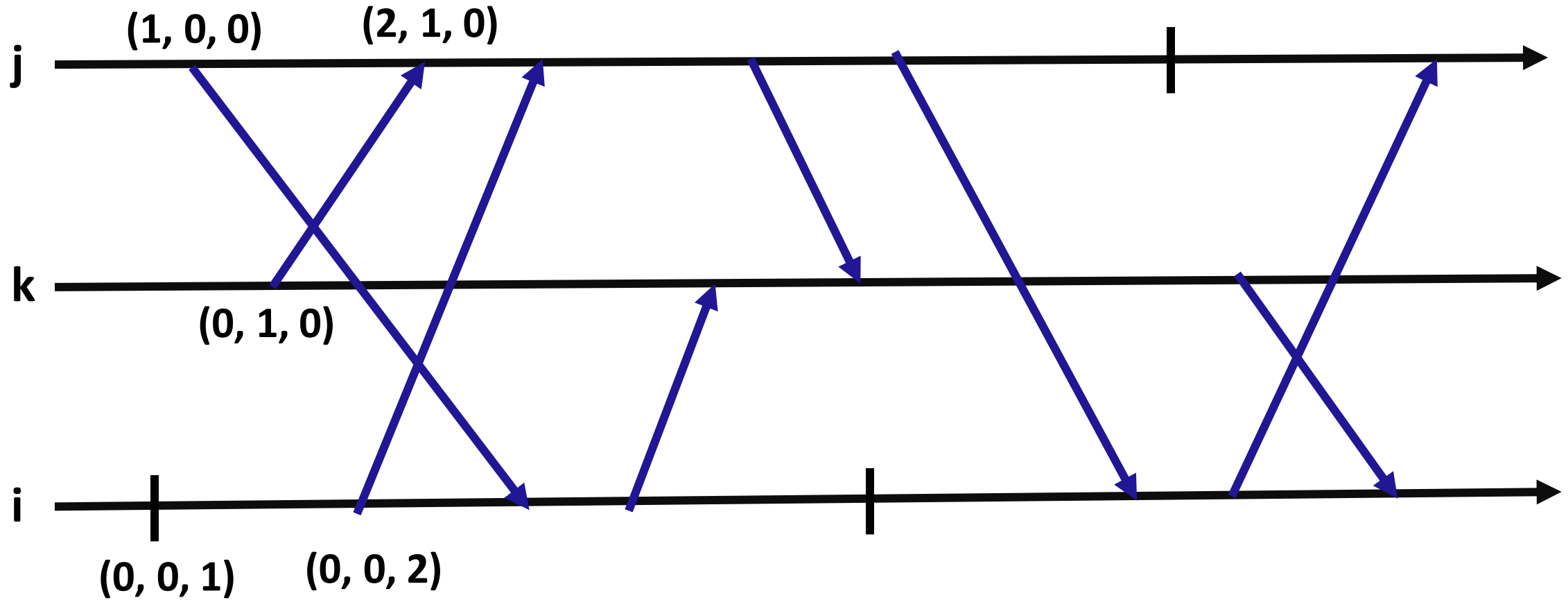
Vector Clock -- Visualization



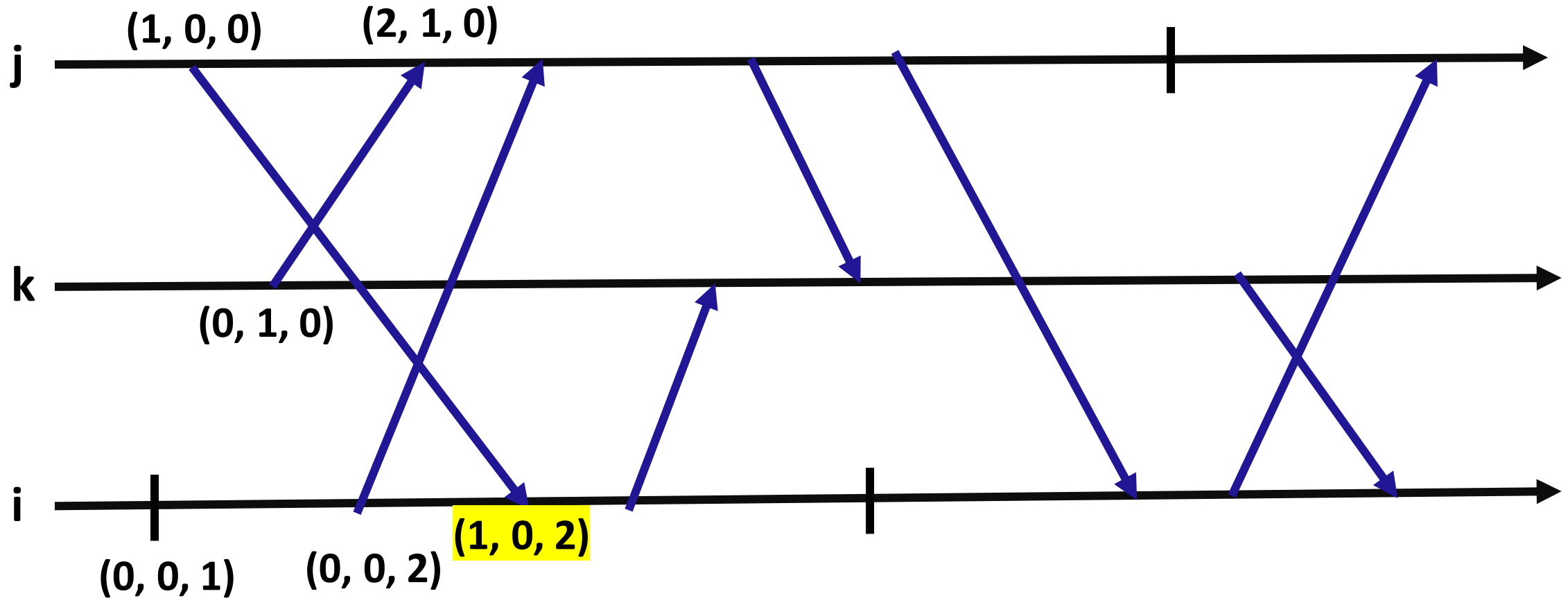
Vector Clock -- Visualization



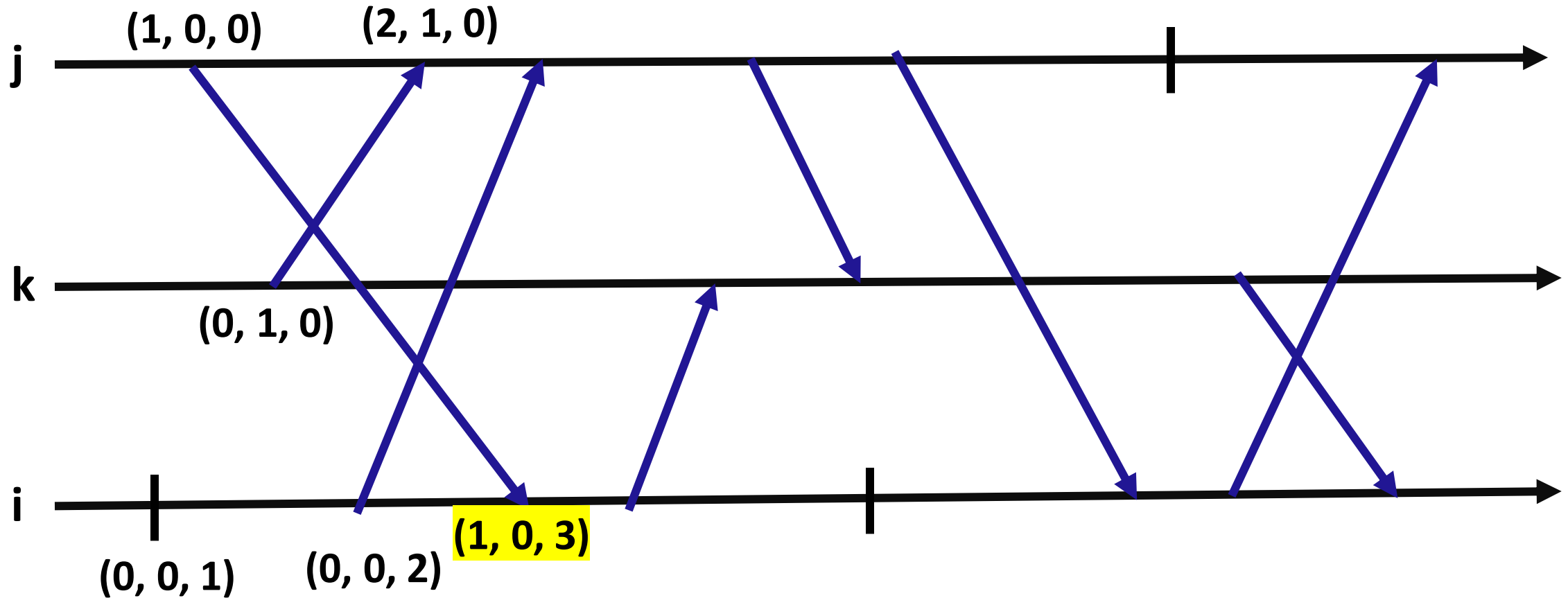
Vector Clock -- Visualization



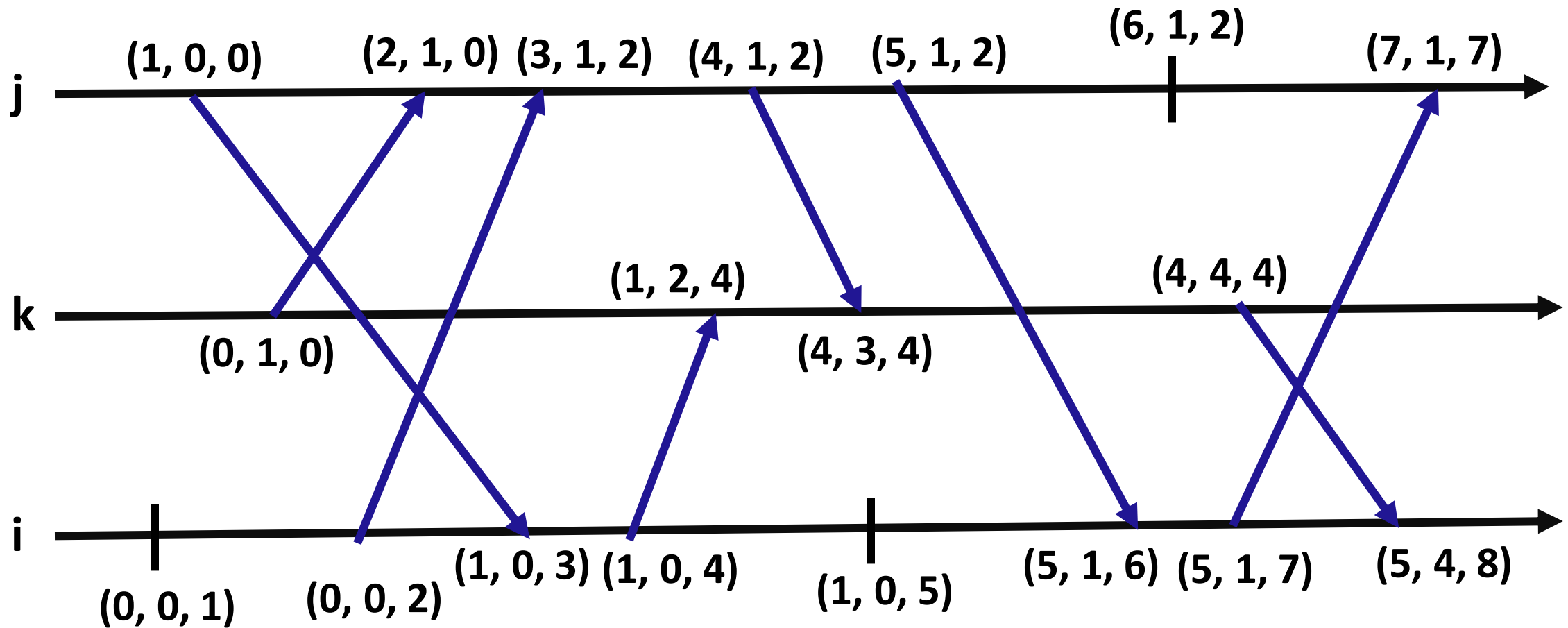
Vector Clock -- Visualization



Vector Clock -- Visualization



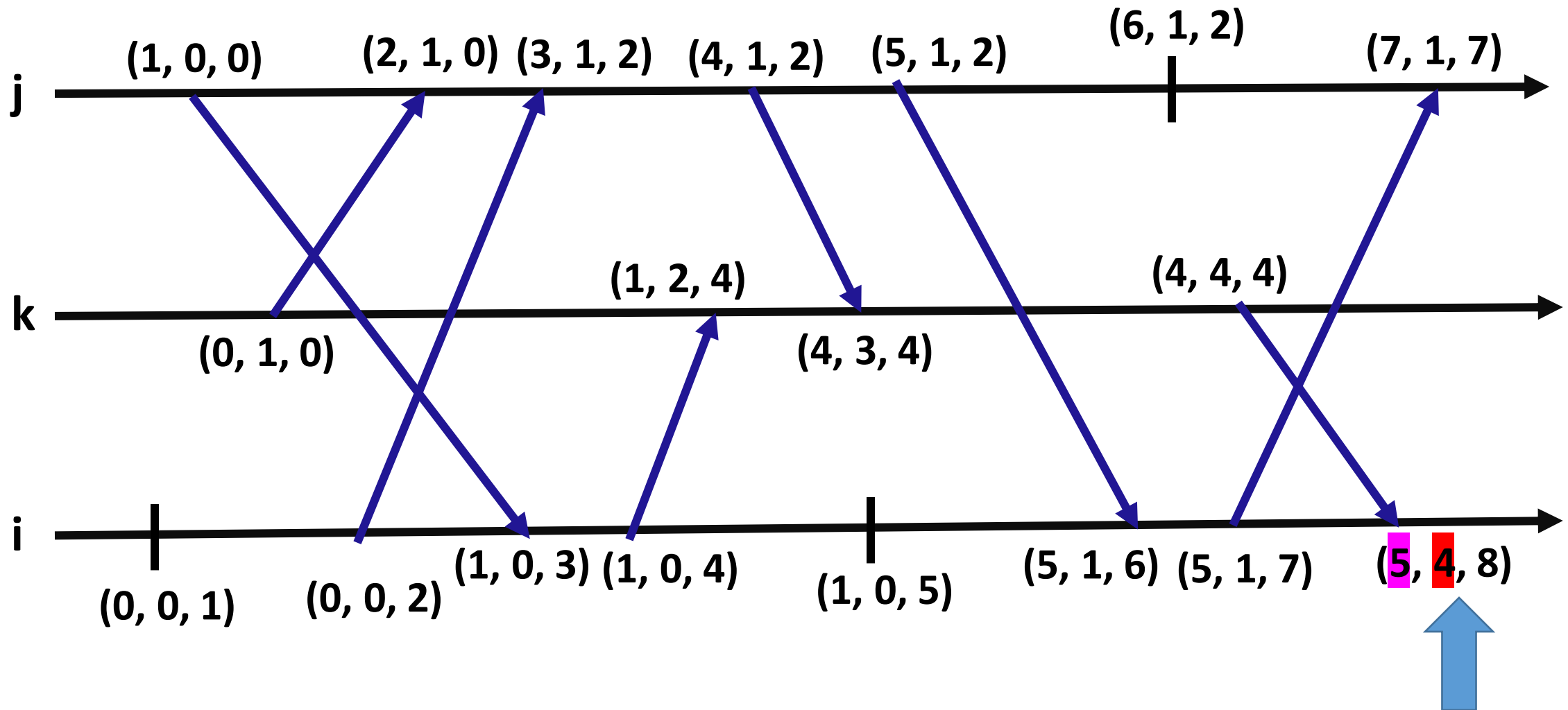
Vector Clock -- Visualization



Vector Clocks – Update Rules

- $VC(e_i)[i] := VC[i] + 1$ if e_i is an internal or a send event
 - Internal or send events simply increment the local component of the vector clock
- $VC(e_i) := \max\{VC, TS(m)\}$ if $e_i = \text{receive}(m)$
 $VC(e_i)[i] := VC[i] + 1$
 - For a receive event, update the vector clock with the greater of local clock and the received timestamp, and then increment it
- **For all $j \neq i$,**
 $VC(e_i)[j] = \text{number of events of } p_j \text{ that causally precedes event } e_i \text{ of } p_i$

Vector Clock and Causality



There are 5 events from process P_j and 4 events from process P_k that causally precedes this event

Strong (True) Clock Condition

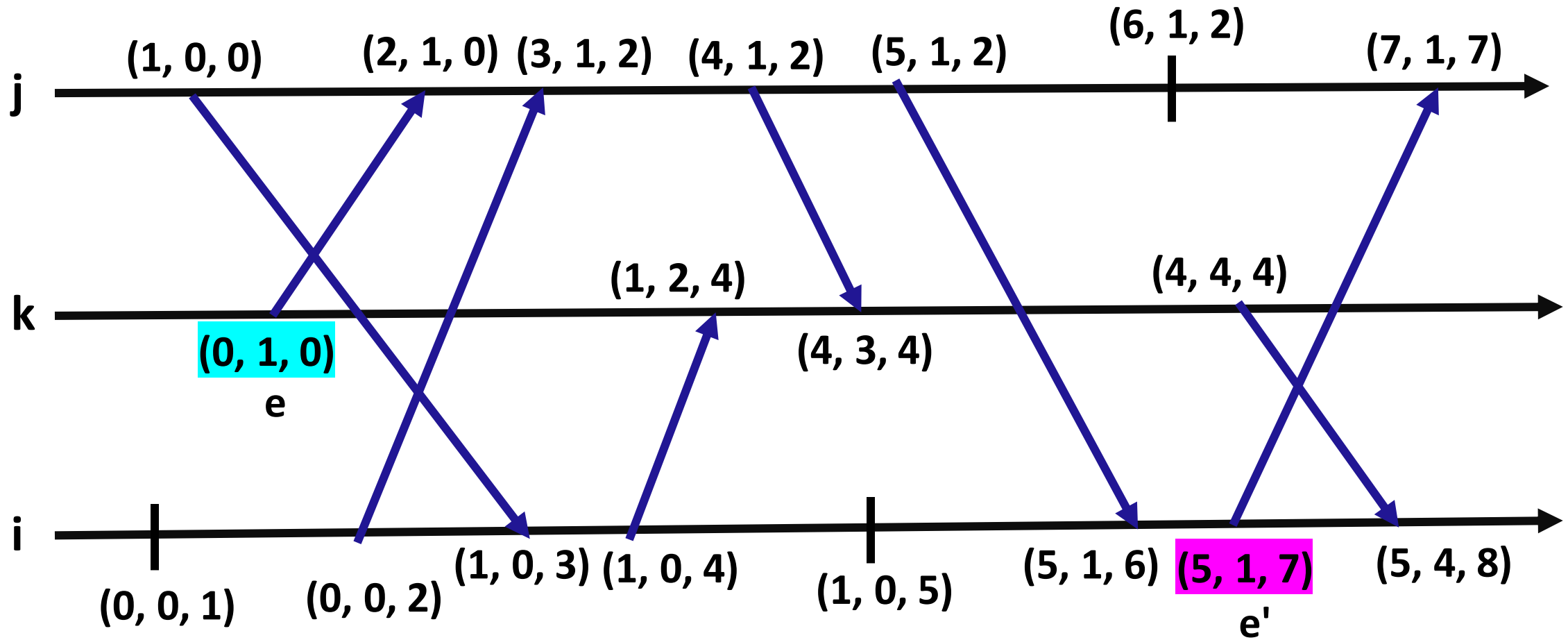
- Given two n-dimensional vector V and V'

$$V < V' \Rightarrow (V \neq V') \wedge (\forall k: 1 \leq k \leq n: V[k] \leq V'[k])$$

- Vector clock supports strong (true) clock condition:

$$e \rightarrow e' \Leftrightarrow VC(e) < VC(e')$$

Strong (True) Clock Condition

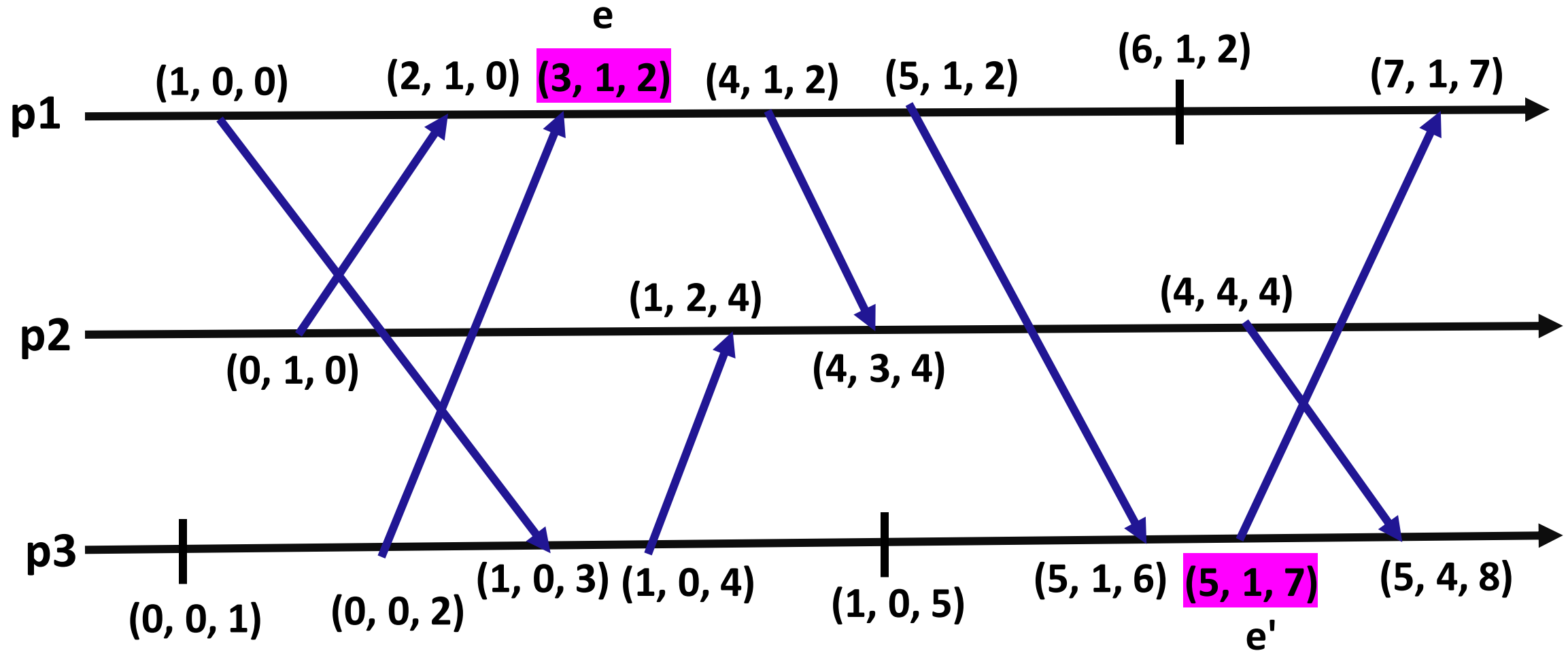


$e \rightarrow e'$

Properties of Vector Clocks

- **Simplified Strong Clock Condition:** Given event e_i of process p_i and event e_j of process p_j , where $i \neq j$,
$$e_i \rightarrow e_j \Leftrightarrow VC(e_i)[i] \leq VC(e_j)[i]$$

Properties of Vector Clock

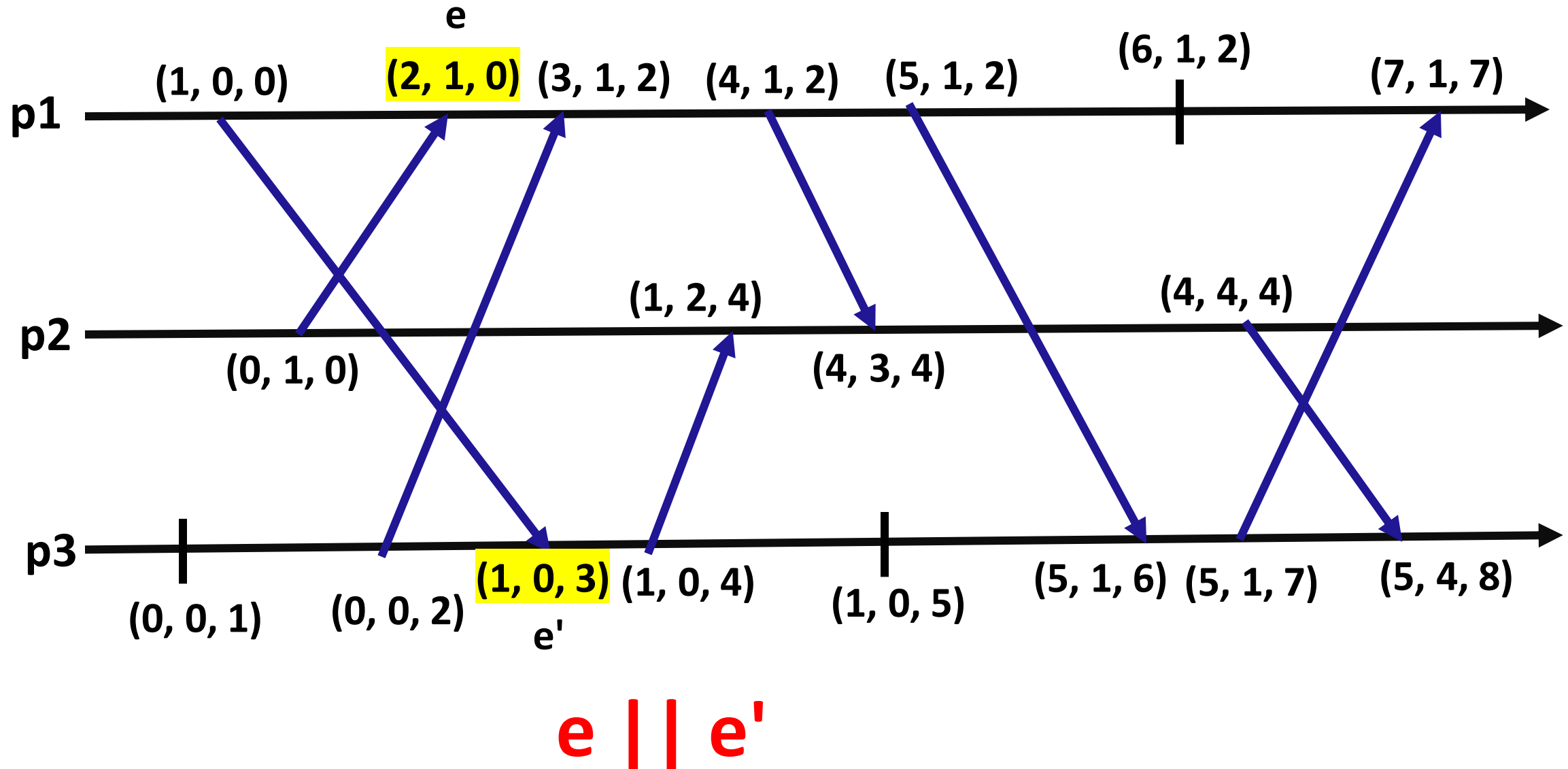


$e \rightarrow e'$

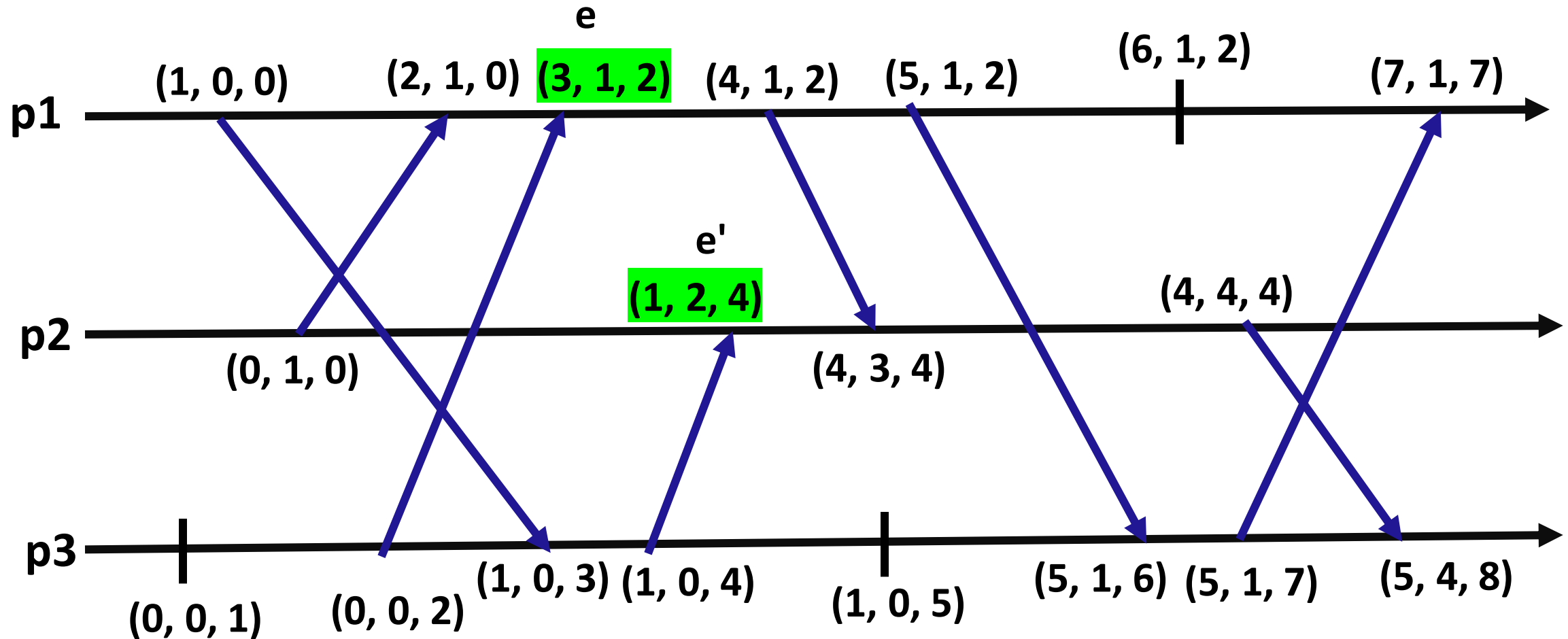
Properties of Vector Clocks

- **Simplified Strong Clock Condition:** Given event e_i of process p_i and event e_j of process p_j , where $i \neq j$,
$$e_i \rightarrow e_j \Leftrightarrow VC(e_i)[i] \leq VC(e_j)[i]$$
- **Concurrent events:** Given event e_i of process p_i and event e_j of process p_j , where $i \neq j$,
$$e_i \parallel e_j \Leftrightarrow (VC(e_i)[i] > VC(e_j)[i]) \text{ AND } (VC(e_j)[j] > VC(e_i)[j])$$

Properties of Vector Clock

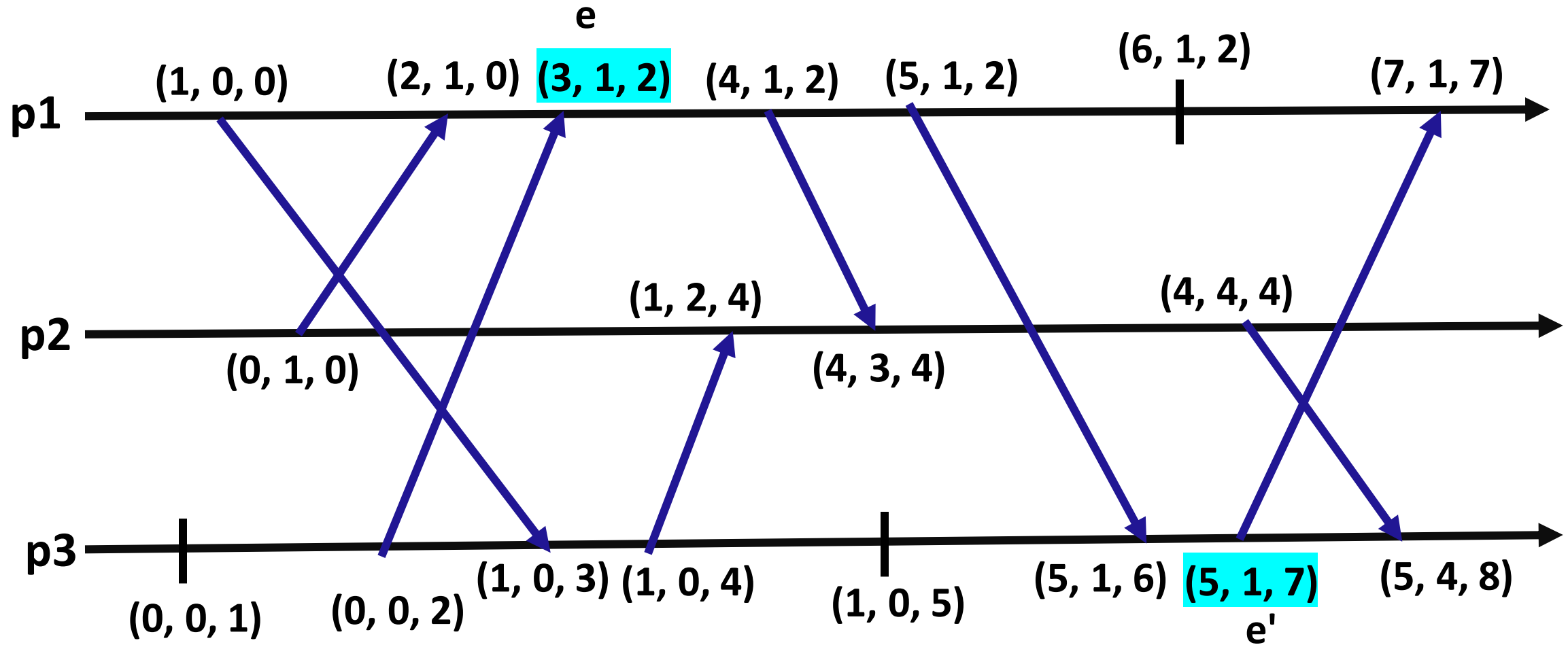


Properties of Vector Clock



What is the relationship between e and e' ?

Properties of Vector Clock



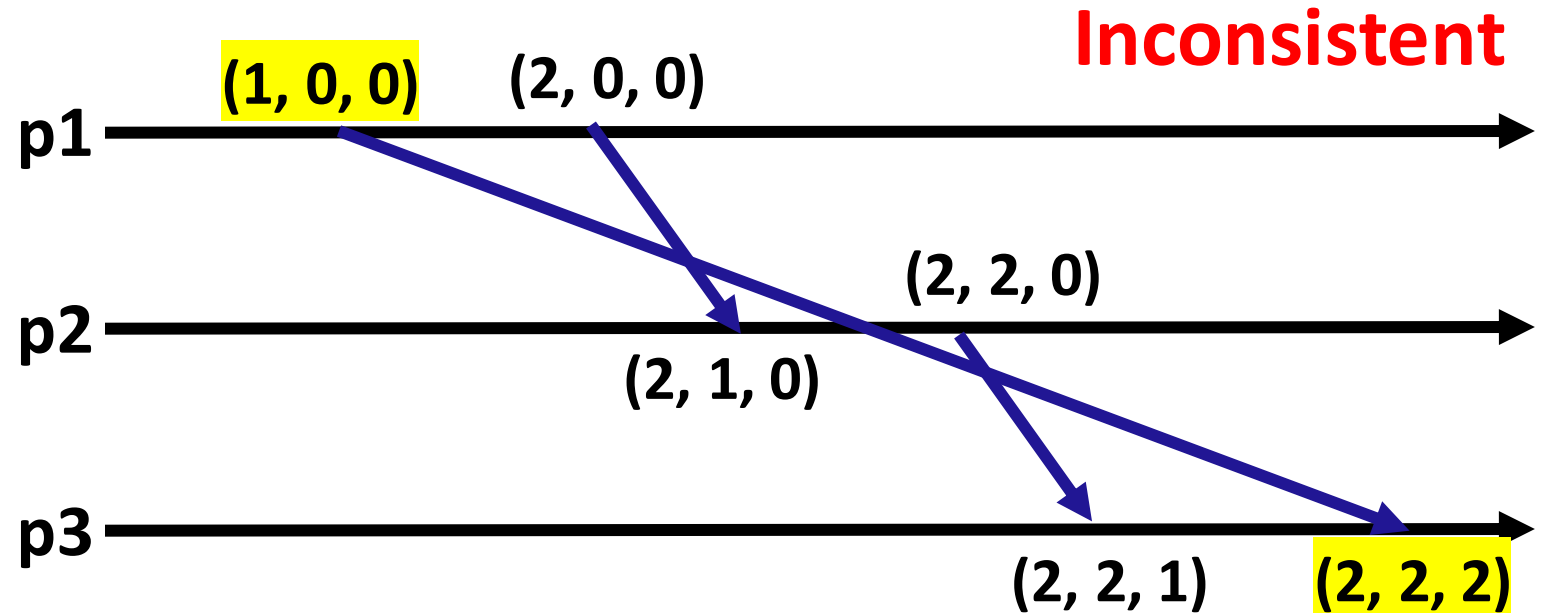
What is the relationship between e and e' ?

Pairwise Inconsistent Events

- Event e_i or process p_i is **pairwise inconsistent** with event e_j of process p_j , where $i \neq j$ if and only if

$$(VC(e_i)[i] < VC(e_j)[i]) \text{ OR } (VC(e_j)[j] < VC(e_i)[j])$$

Pairwise :=
{send(M), Receive(M)}

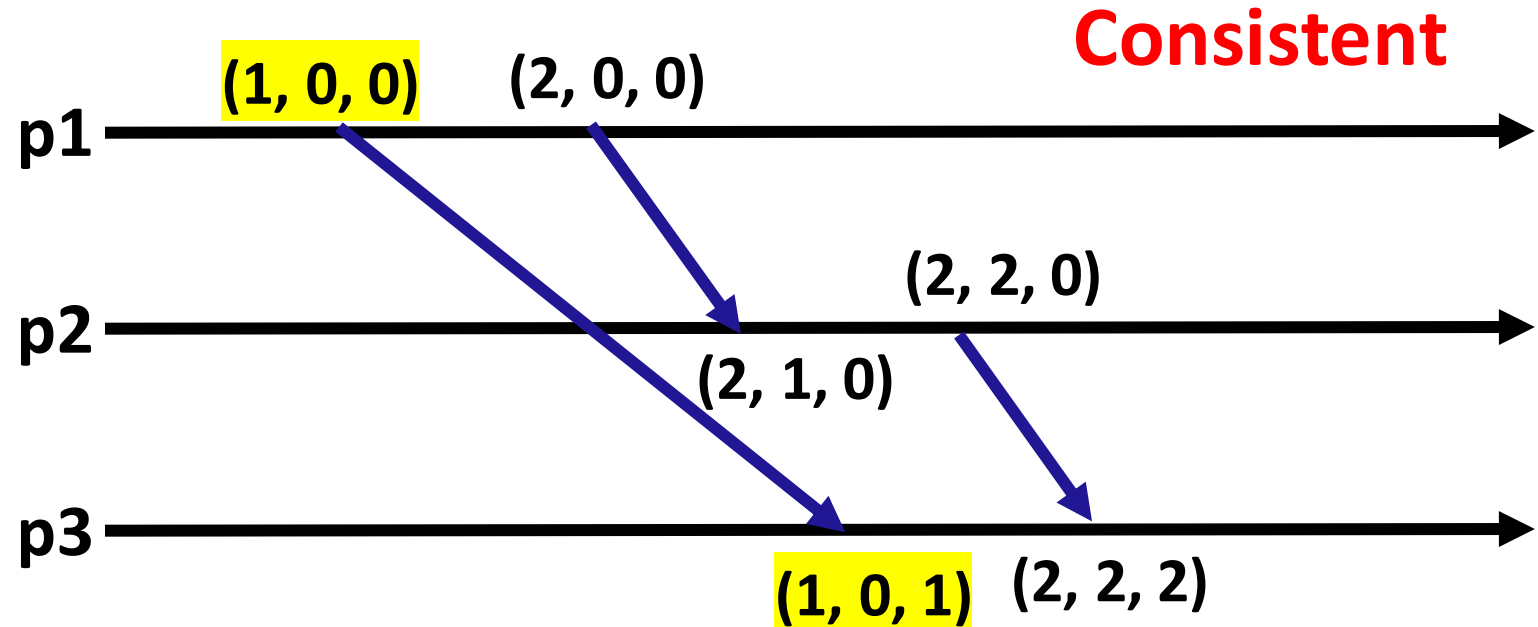


Pairwise Inconsistent Events

- Event e_i or process p_i is **pairwise inconsistent** with event e_j of process p_j , where $i \neq j$ if and only if

$$(VC(e_i)[i] < VC(e_j)[i]) \text{ OR } (VC(e_j)[j] < VC(e_i)[j])$$

Pairwise :=
{send(M), Receive(M)}



Causal Delivery using Vector Clock

- Each application process contains a middleware that delivers the message to the application based on the causal rule.
- Each message contains a timestamp, i.e., vector clock of the corresponding Send event.
- The middleware delivers the message M, if
 - No other message from the sender causally preceeds M
 - Let M' be the last delivered message. There is no other message M'' from a third process, such that $\text{Send}(M') \rightarrow \text{Send}(M'') \rightarrow \text{Send}(M)$

Further Reads

- Kshemkalyani, Ajay D., Michel Raynal, and Mukesh Singhal. "**An introduction to snapshot algorithms in distributed computing.**" *Distributed systems engineering* 2.4 (1995): 224.
 - Summarizes snapshot algorithms under FIFO, non-FIFO, and causal channels

