# CS4231
# Parallel and Distributed Algorithms

Lecture 4

Instructor: Haifeng YU

# Review of Last Lecture

- What is consistency? Why do we care about it?
- Sequential consistency
- Linearizability
  - Linearizability is a local property
- Consistency models for registers

# Today's Roadmap

- Chapter 7 "Models and Clocks"
  - Goal: Define "time" in a distributed system
- Logical clock
- Vector clock
- Matrix clock

# Assumptions

- Process can perform three kinds of atomic actions/events
  - Local computation
  - Send a single message to a single process
  - Receive a single message from a single process
  - No atomic broadcast

- Communication model
  - Point-to-point
  - Error-free, infinite buffer
  - Potentially out of order

# Motivation

- Many protocols need to impose an ordering among events
  - Event A: You mom deposit some money into your bank account as your birthday gift
  - Event B: You use your ATM card to use the money to buy some stuff
  - Your bank needs to properly order the two events
- Physical clocks:
  - Seems to completely solve the problem
  - But what about theory of relativity?
  - Even without theory of relativity – efficiency problems
- How accurate is sufficient?
  - Without out-of-band communication: Message propagation delay
  - With out-of-band communication: distance/speed of light
  - In other words, some time it has to be “quite” accurate

# Software “Clocks”

- Software “clocks” can incur much lower overhead than maintaining (sufficiently accurate) physical clocks

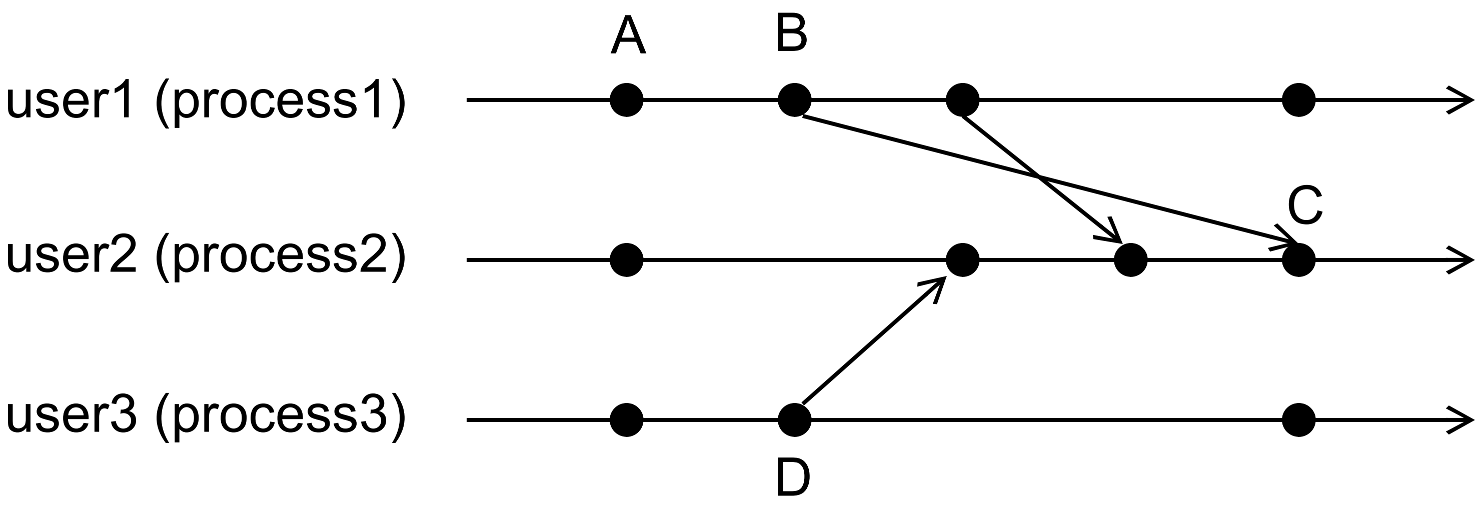
- Allows a protocol to infer ordering among events

- Goal of software “clocks”: Capture event ordering that are visible to users who do not have physical clocks
  - But what orderings are visible to users without physical clocks?

# Visible Ordering to Users without Physical Clocks

- $A \rightarrow B$ (process order)
- $B \rightarrow C$ (send-receive order)
- $A \rightarrow C$ (transitivity)

- A ? D
- B ? D
- C ? D

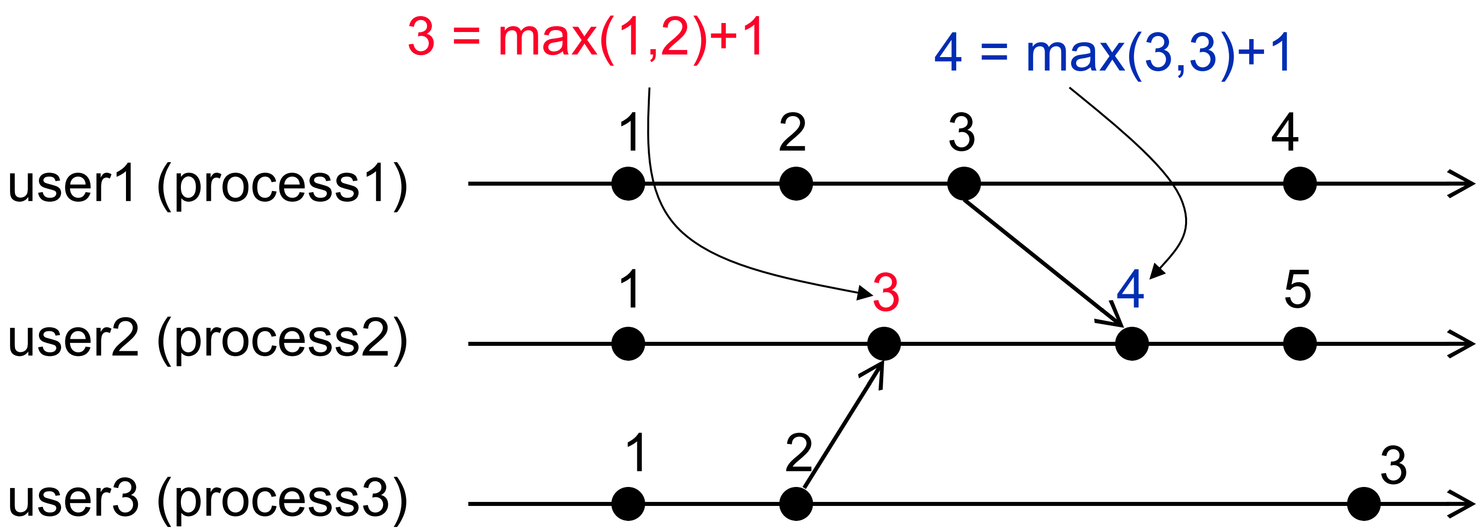


# “Happened-Before” Relation

- “Happened-before” relation captures the ordering that is visible to users when there is no physical clock
  - A partial order among events
  - Process order, send-receive order, transitivity
- First introduced by Lamport – Considered to be the first fundamental result in distributed computing
- Goal of software “clock” is to capture the above “happened-before” relation

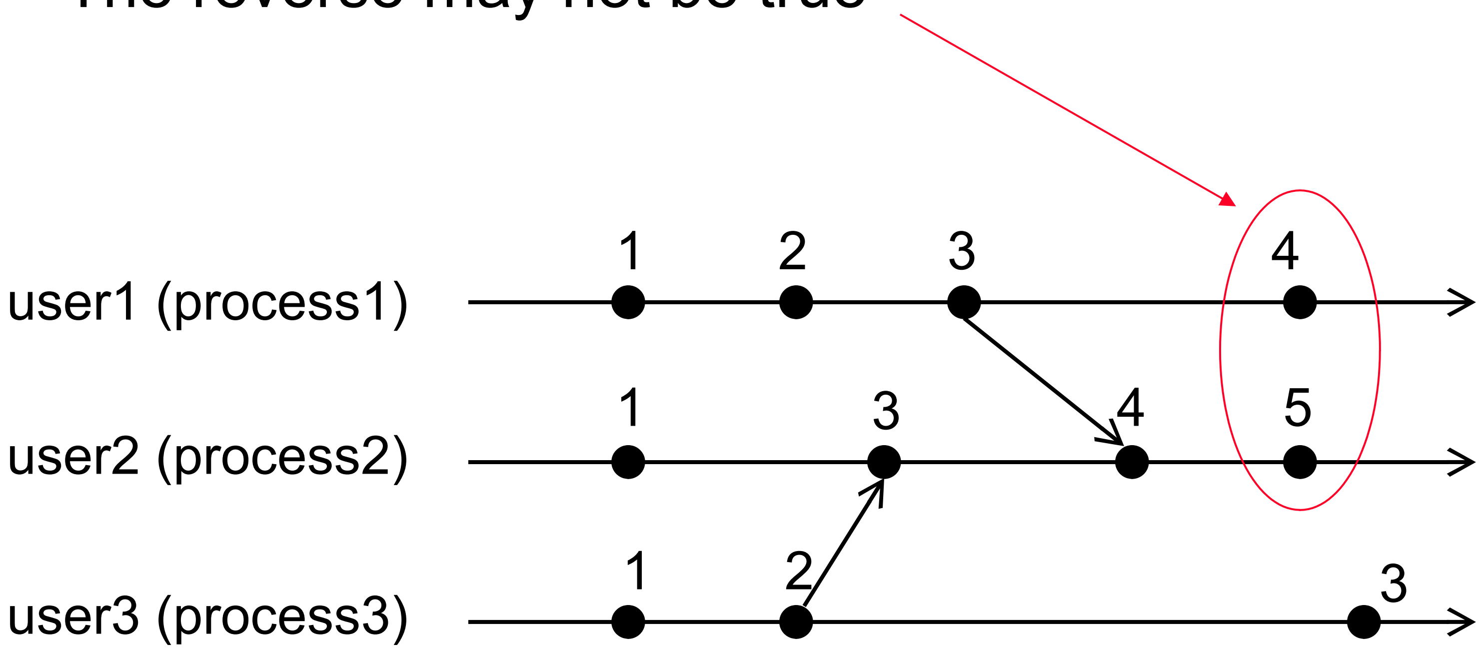
# Software "Clock" 1: Logical Clocks

- Each event has a single integer as its logical clock value
  - Each process has a local counter C
  - Increment C at each "local computation" and "send" event
  - When sending a message, logical clock value V is attached to the message. At each "receive" event, C = max(C,  V) + 1



# Logical Clock Properties

- Theorem:
  - Event s happens before t $\Rightarrow$ the logical clock value of s is smaller than the logical clock value of t.
- The reverse may not be true



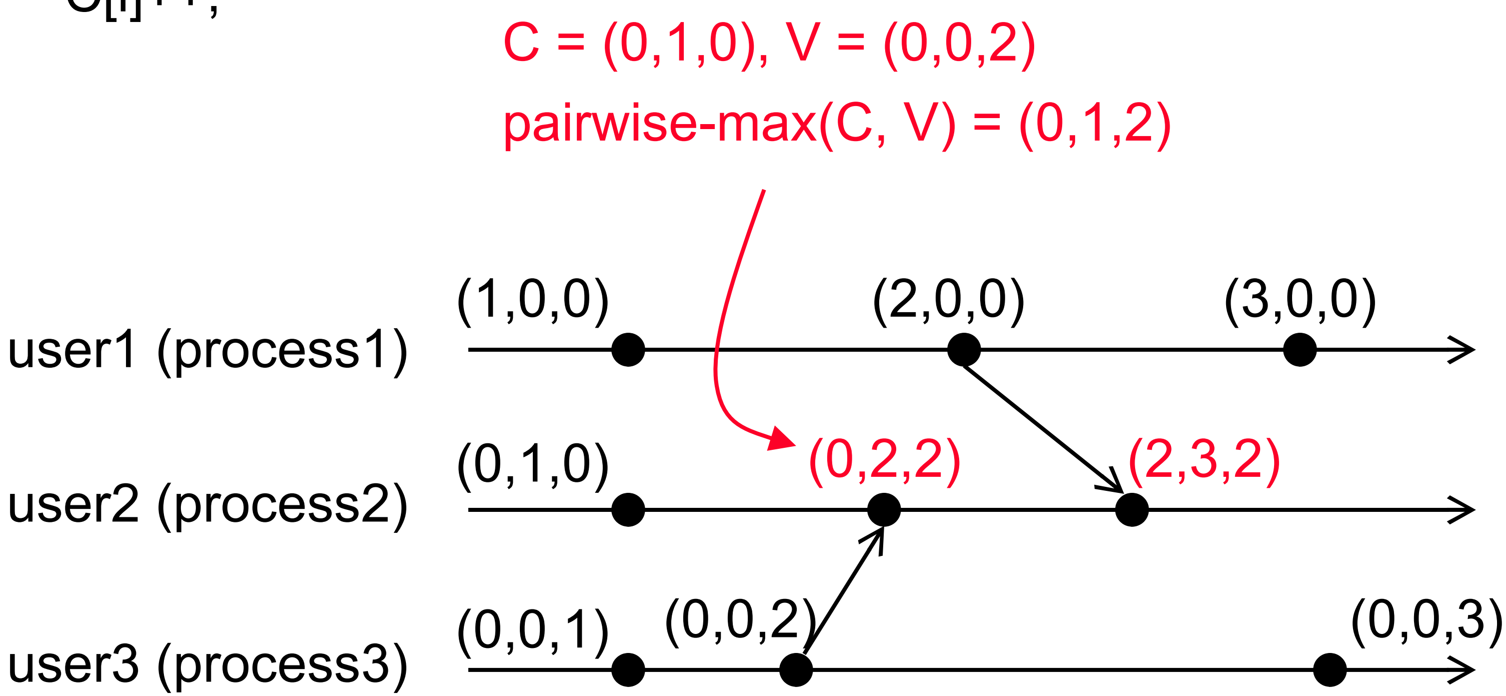
# Software "Clock" 2: Vector Clocks

- Logical clock:
  - Event s happens before event t $\Rightarrow$ the logical clock value of s is smaller than the logical clock value of t.
- Vector clock:
  - Event s happens before event t $\Leftrightarrow$ the vector clock value of s is "smaller" than the vector clock value of t.
- Each event has a vector of *n* integers as its vector clock value
  - v1 = v2 if all n fields same
  - v1 $\leq$ v2 if every field in v1 is less than or equal to the corresponding field in v2
  - v1 < v2 if v1 $\leq$ v2 and v1 $\neq$ v2

Relation "<" here is not a total order

# Vector Clock Protocol

- Each process i has a local vector C
- Increment C[i] at each “local computation” and “send” event
- When sending a message, vector clock value V is attached to the message. At each “receive” event, C = pairwise-max(C,  V); C[i]++;



# Vector Clock Properties

- Event s happened before t ⇒

  vector clock value of s < vector clock value of t.
- Prove by enumeration all possible cases.
- If s happened before t due to process order…
- If s happened before t due to send-receive order…
- If s happened before t due to transitivity, then there must be a chain of evens such that

  s happened before x1,

  x1 happened before x2,

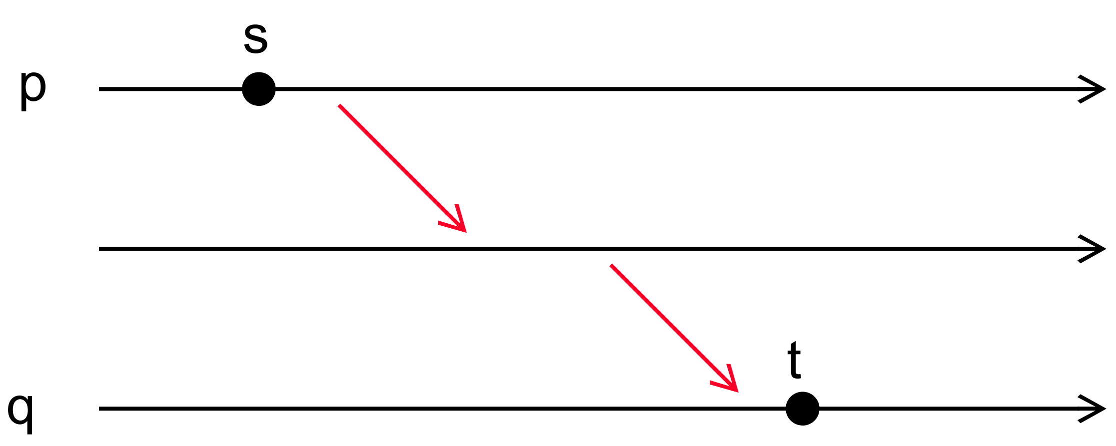
  …

  x_n happened before t
- Then the vector clock value of s < vector clock value of x1 < … < vector clock value of x_n < vector clock value of t.
- Smaller than relation among vector clock values is transitive as well…

# Vector Clock Properties

- Vector clock value of s < vector clock value of t $\Rightarrow$

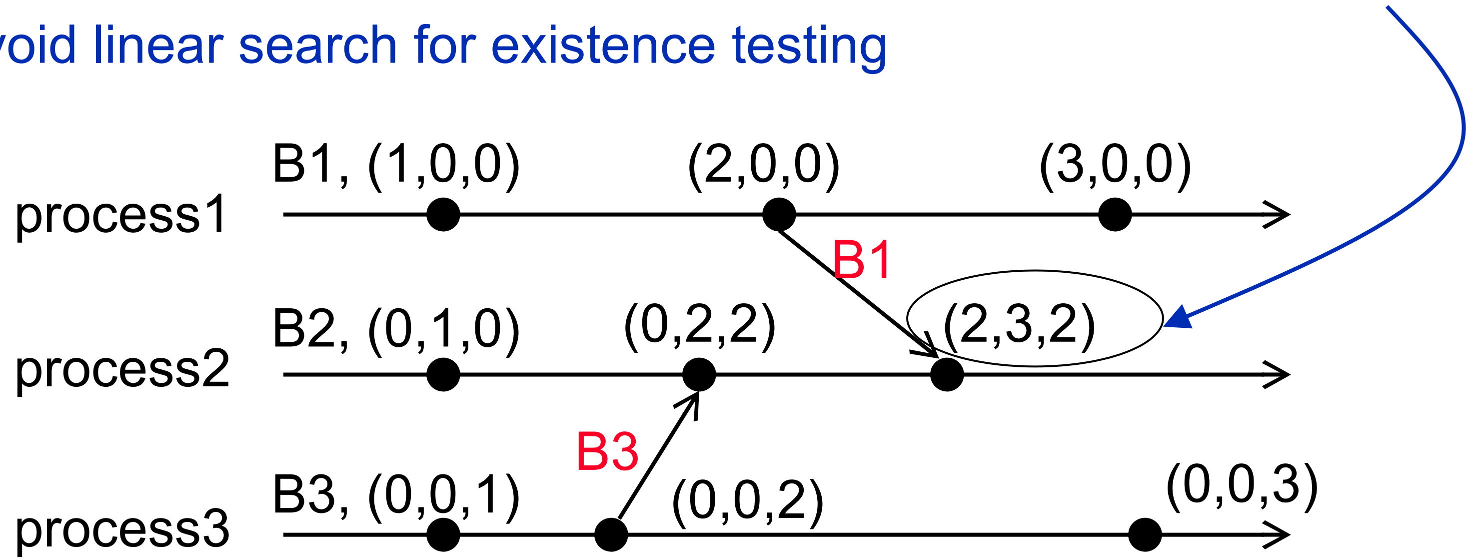
  s happened before t
- Prove by separately considering two cases.
- If s and t on same process, done.
- If s is on p and t is on q, let VS be s's vector clock and VT be t's.
- VS < VT $\Rightarrow$ VS[p] ≤ VT[p] $\Rightarrow$ Must be a sequence of message from p to q after s and before t.



# Example Application of Vector Clock

- In Bitcoin, each process has a block
  - Want all processes to know all blocks
- Each block has a vector clock value

p2 have seen all blocks whose vector clock is smaller than (2,3,2):
p2 can avoid linear search for existence testing



# Software "Clock" 3: Matrix Clocks

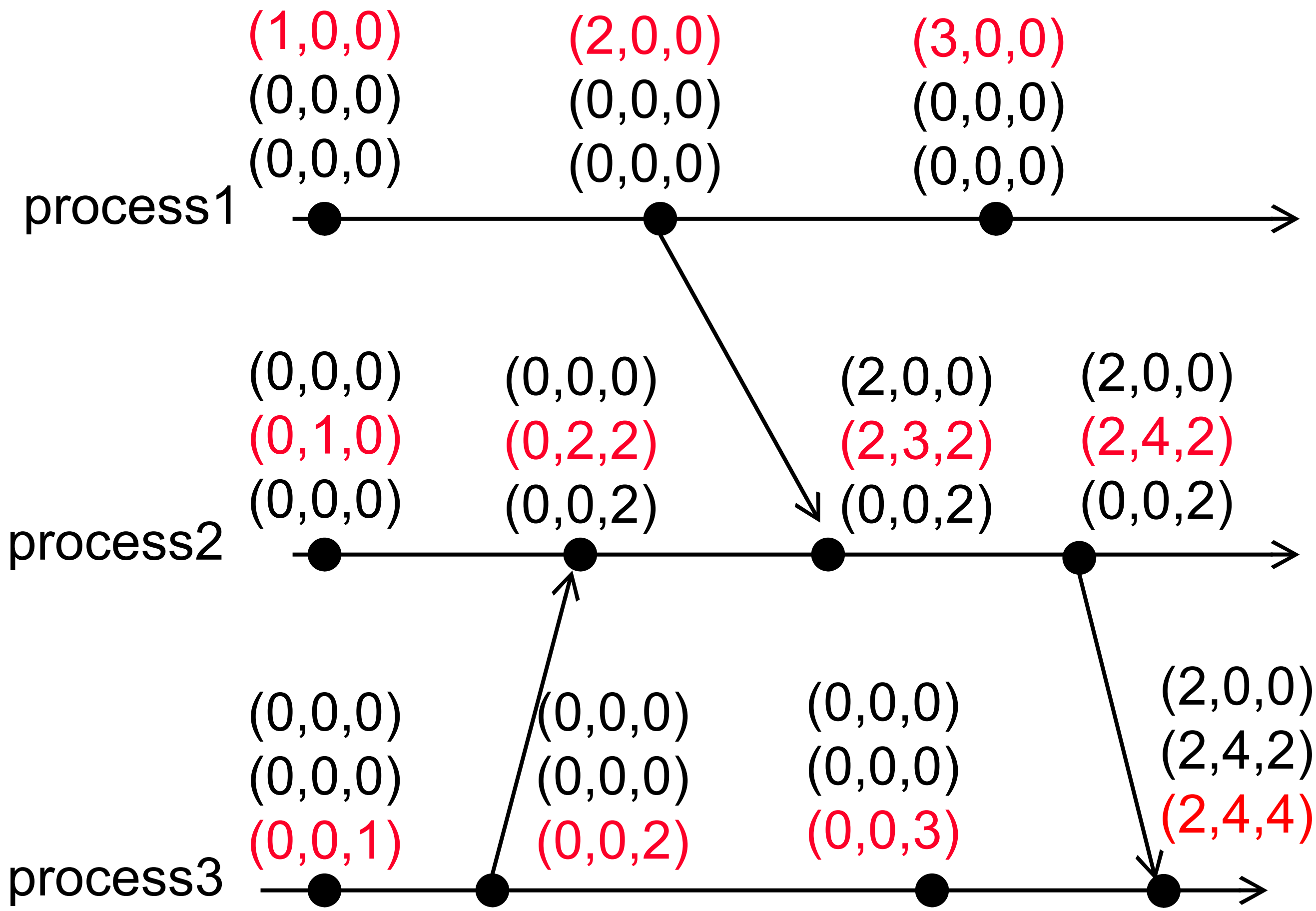
- Motivation
  - My vector clock describe what I "see"
  - In some applications, I also want to know what other people see
- Matrix clock:
  - Each event has n vector clocks, one for each process
  - The *i* th vector on process *i* is called process *i* 's principle vector
  - Principle vector is the same as vector clock before
  - Non-principle vectors are just piggybacked on messages to update "knowledge"

# Matrix Clock Protocol

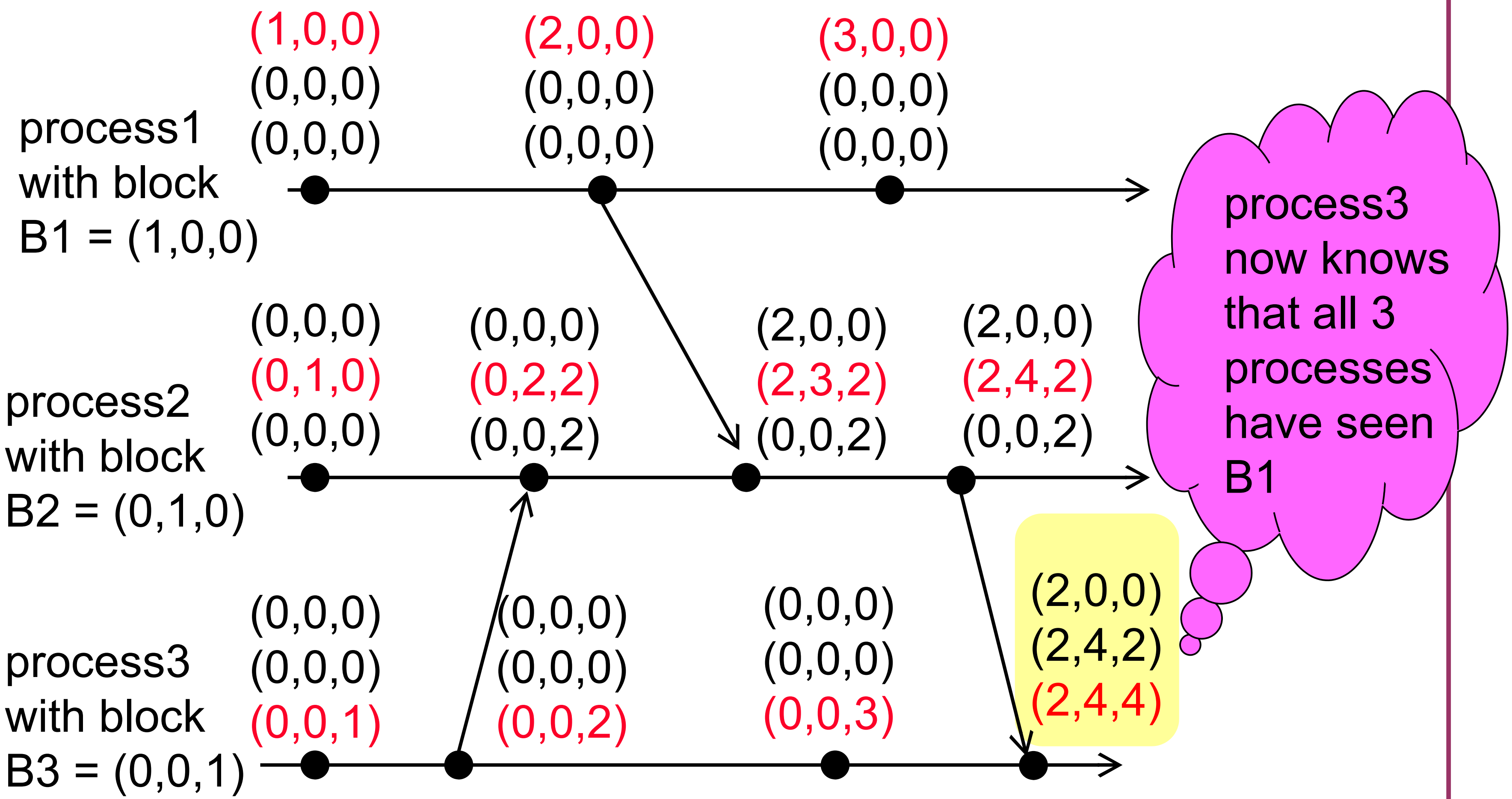
- For principle vector C on process i
  - Increment C[i] at each "local computation" and "send" event
  - When sending a message, all n vectors are attached to the message
  - At each "receive" event, let V be the principle vector of the sender. C = pairwise-max(C, V); C[i]++;

- For non-principle vector C on process i, suppose it corresponds to process j
  - At each "receive" event, let V be the vector corresponding to process j as in the received message. C = pairwise-max(C, V)

# Matrix Clock Example



# Application of Matrix Clock



# A Snack for Mind

- Vector clock tells me what I know
  - One-dimensional data structure
- Matrix clock tells me what I know about what other people know
  - Two-dimensional data structure
- ?? tells me what I know about what other people know about what other people know
  - ??-dimensional data structure

# Summary

- Chapter 7 “Models and Clocks”
  - Goal: Define “time” in a distributed system
- Logical clock
  - “happened before” $\Rightarrow$ smaller clock value
- Vector clock
  - “happened before” $\Leftrightarrow$ smaller clock value
- Matrix clock
  - Gives a process knowledge about what other processes know

# Homework Assignment

- Page 126
  - Problem 7.2 – give a counter-example

    Show that "concurrent with" is not a transitive relation
  - Problem 7.4

    We discussed a method by which we can totally order all events within a system. If two events have the same logical time, we broke the tie using process identifiers. This scheme always favor processes with smaller identifiers. Suggest a scheme that does not have this disadvantage. (Hint: Use th evalue of the logical clock in determining the priority.)
  - Problem 7.7 – Prove that the solution satisfies the property of logical clocks

    Assume that you have implemented the vector clock algorithm. However, some application needs Lamport's logical clock. Write a function *convert* that takes as input a vector timestamp and outputs a logical clock timestamp.
- Read Chapter 9