# CS4231
# Parallel and Distributed Algorithms

Lecture 6

Instructor: YU Haifeng

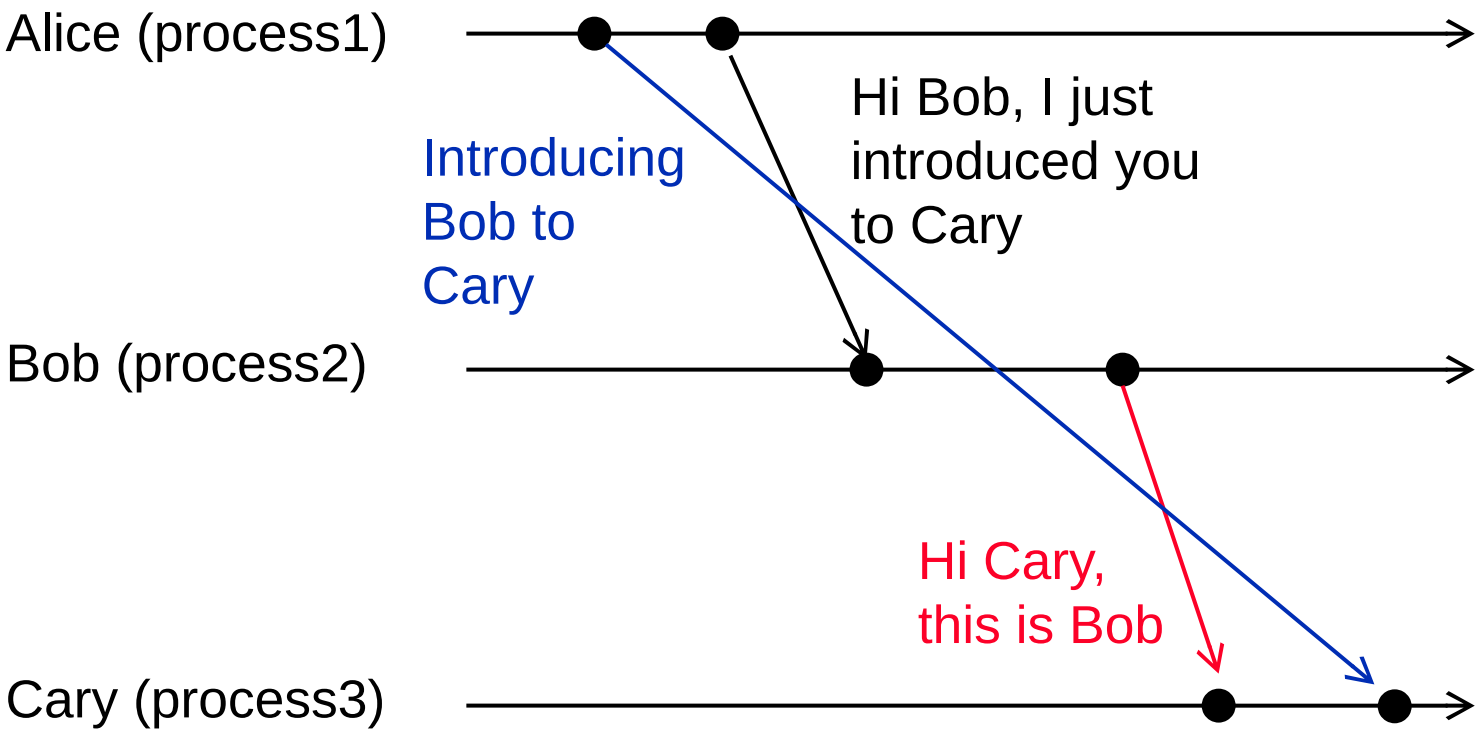
# Review of Last Lecture

- “Global Snapshot”
- What is our goal?
- Formalizing consistent global snapshot
- Protocol for capturing a consistent global snapshot

# Today's Roadmap

- "Message Ordering"
- FIFO ordering
  - Already discussed last lecture
- Causal ordering and its application
- Total ordering and its application

# A Internet Chat Room Example

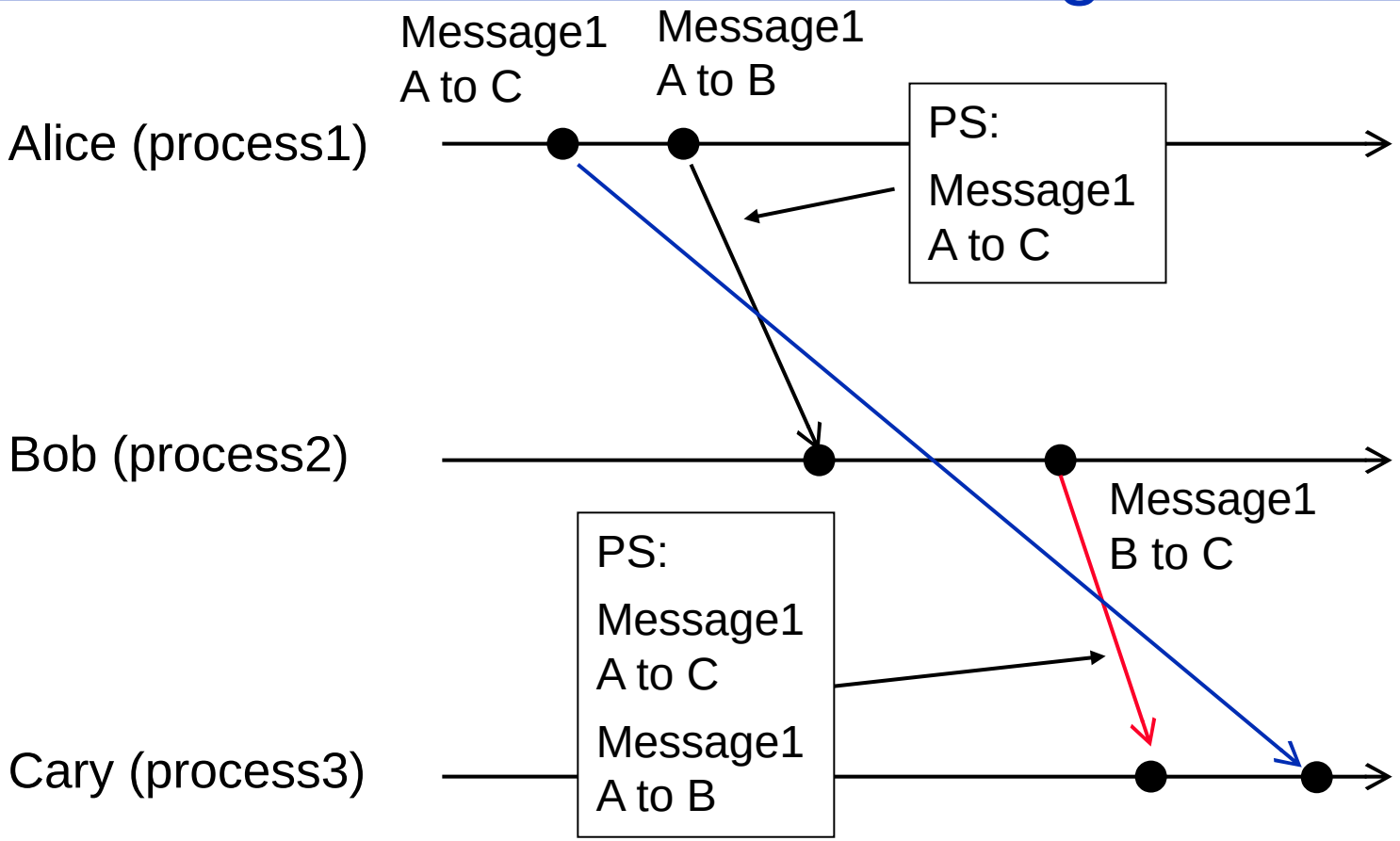



The notion of "causal order".

# Formalizing The Notion of Causal Order

- If a “send” event s1 caused a “send” event s2
  - Causal order requires the corresponding receive event r1 be before r2 (r1 and r2 on the same process)
  - But how do we know whether s1 caused s2?
- If a “send” event s1 happened before a “send” event s2
  - Then s1 may have caused s2
  - Let’s be pessimistic and be safe -- Assume s1 indeed caused s2
  - Can we avoid being pessimistic?
- Causal order: If s1 happened before s2, and r1 and r2 are on the same process, then r1 must be before r2

# How to Ensure Causal Ordering – Intuition




How do you think of this protocol? Is it correct? Any issues?

Quick Poll https://pollev.com/haifengyu229

# How to Ensure Causal Ordering – Protocol

- Each process maintains $n$ by $n$ matrix $M$
  - This is not the matrix clock!
  - $M[i, j]$: # of messages sent from $i$ to $j$, as known by process $i$
- If process $i$ send a message to process $j$
  - On process $i$: $M[i, j]$++;
  - Piggyback $M$ on the message

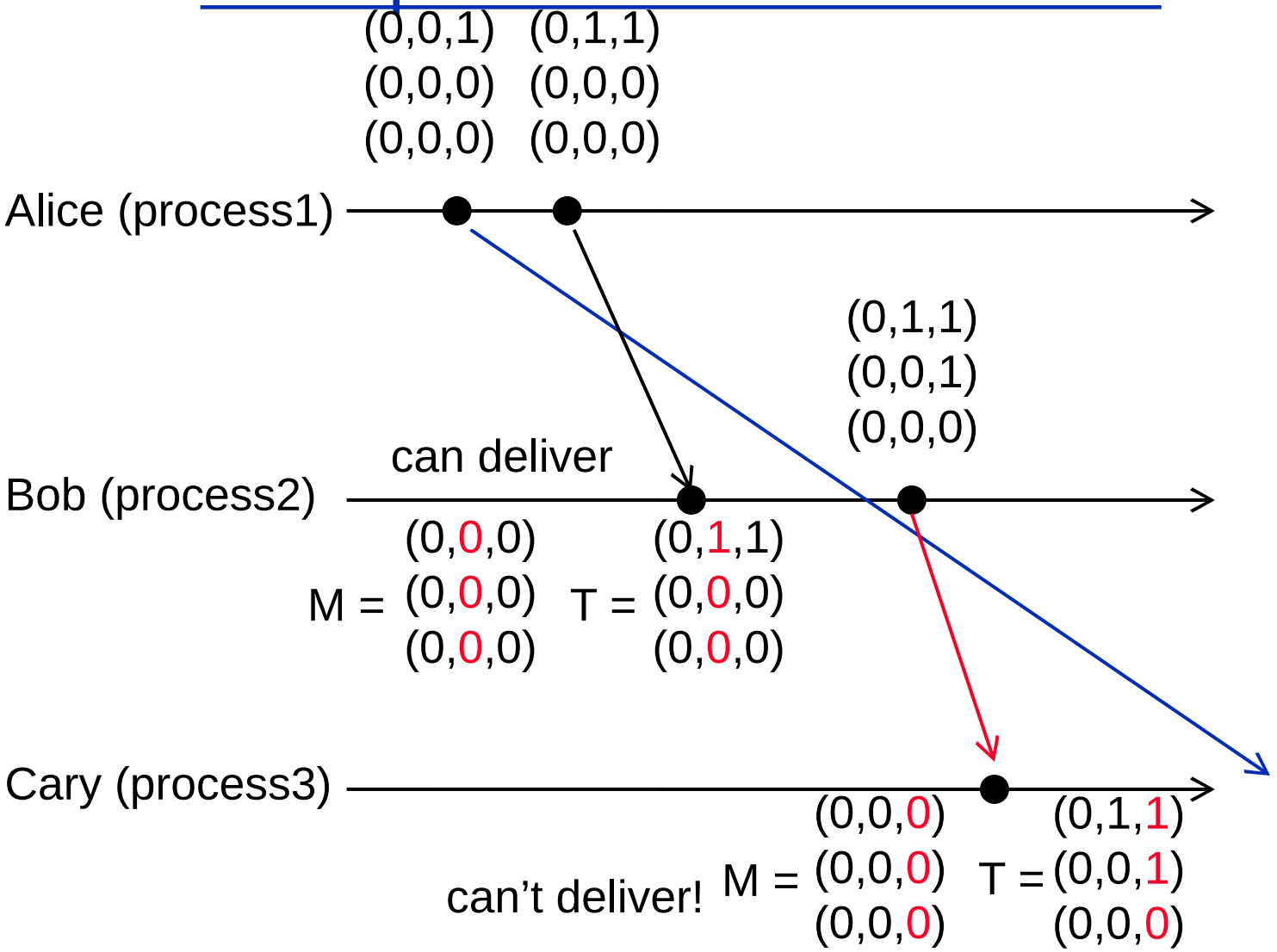
# How to Ensure Causal Ordering – Protocol

- Upon process $j$ receiving a message from process $i$ with matrix $T$ piggybacked
  - Let $M$ be the local matrix on process $j$
- Deliver the message and set $M$ = pairwise-max($M$, $T$), if

$$\begin{cases} T[k, j] <= M[k, j] & \text{for all } k \neq i \\ T[i, j] = M[i, j] + 1 \end{cases}$$

  - Intuitively, $M[i, j]$ on process j takes consecutive values
- Otherwise delay message
- Intuition: *The matrix summarizes the information in the message log in the previous protocol on Slide 6*

# Example Run for the Protocol



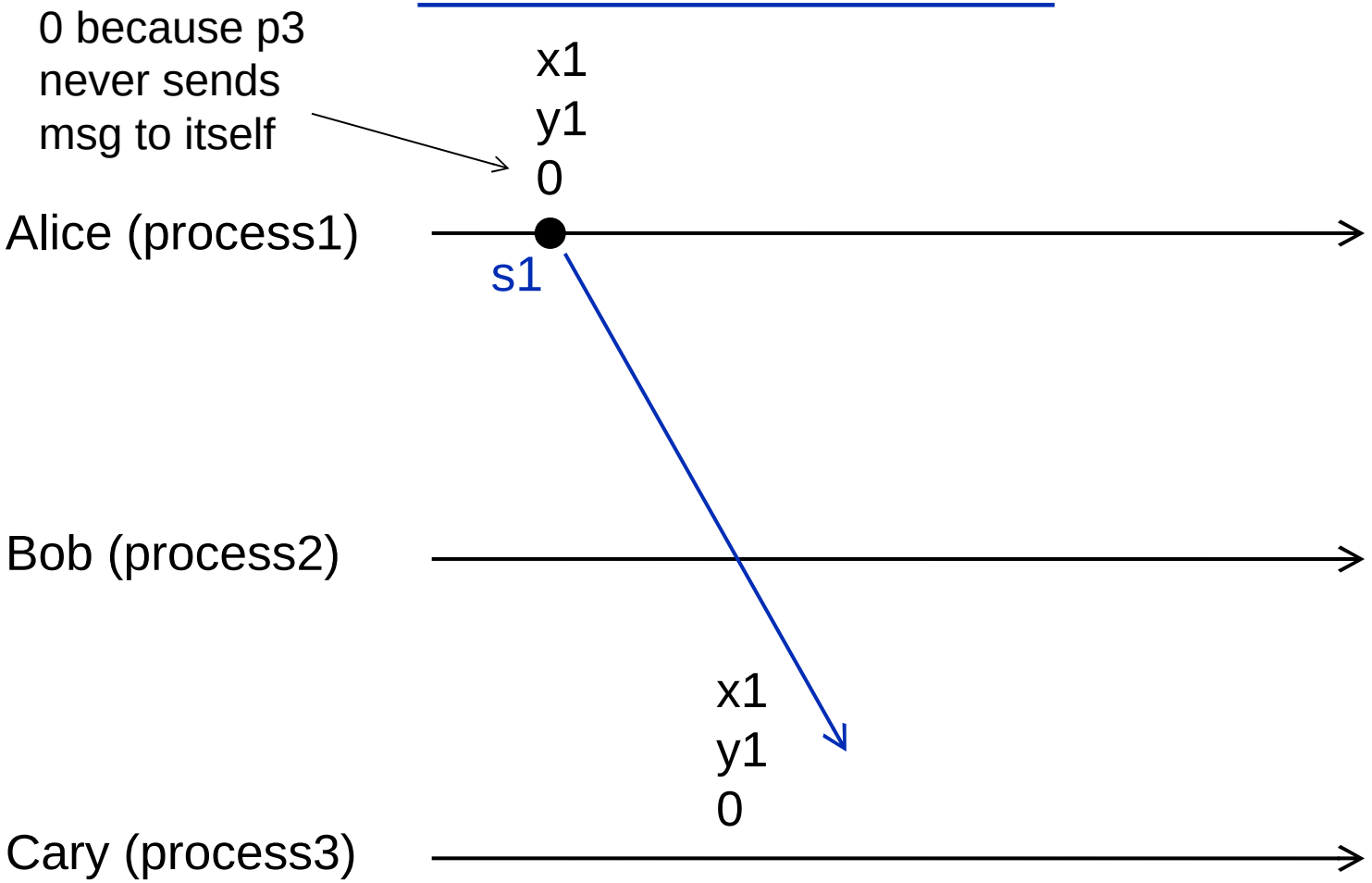
# Correctness Proof of the Protocol

- s1, s2, r1, r2, where s1 happened before s2
  - Need to prove: both r1 and r2 will eventually be delivered – so either r1 before r2 or r2 before r1
  - Need to prove: r2 is not before r1 (we prove this first)

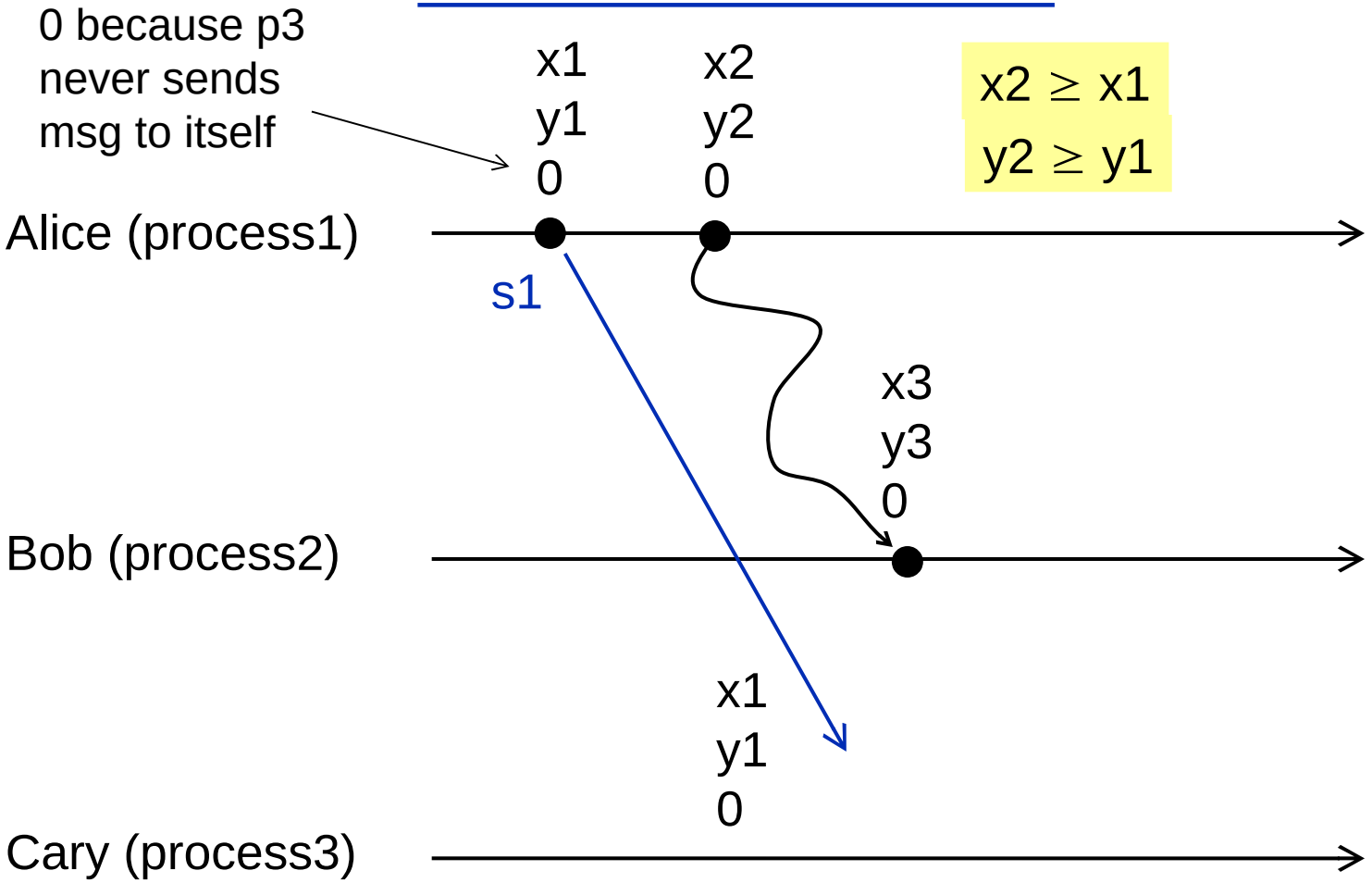
- Consider the case where s1 and s2 are on different processes
  - If they are on the same process, it is easier – an exercise for you
- Assume s1 on process1, s2 on process2, r1 and r2 on process3
- Focus on the top 3 elements of the 3rd column of the matrices

(0,0,0)
(0,0,0)
(0,0,0)

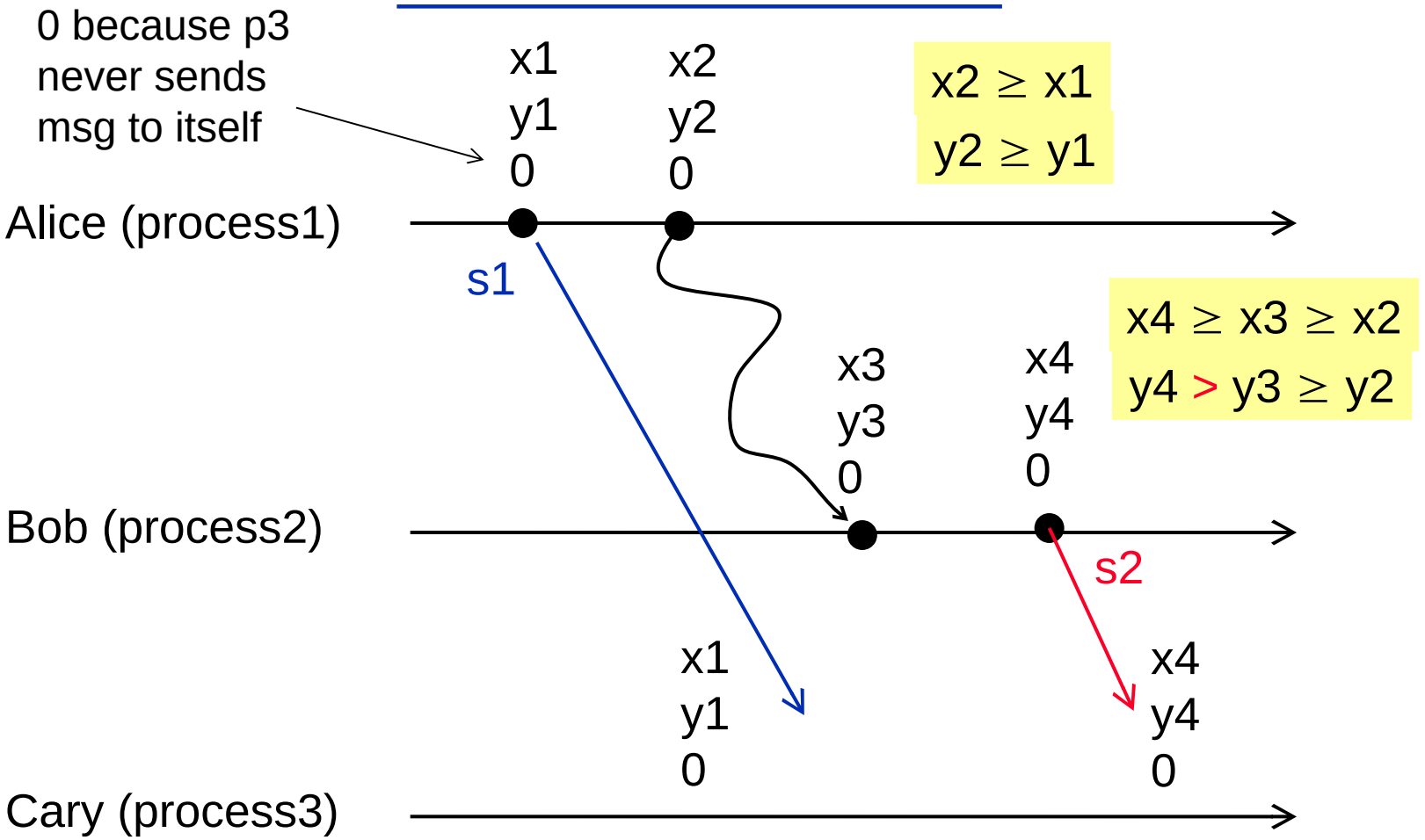
# Correctness Proof



# Correctness Proof



# Correctness Proof




conclusion so far: x4 ≥ x1 and y4 > y1

# Correctness Proof (continued)

- Prove by contradiction: Assume Red delivered before Blue
  - After delivering Red: 3rd column of matrix M on process3 will be

$$M = \begin{matrix} \geq x4 \geq x1 \\ \geq y4 > y1 \\ 0 \end{matrix}$$

  - M never decreases: Blue can never be delivered any more

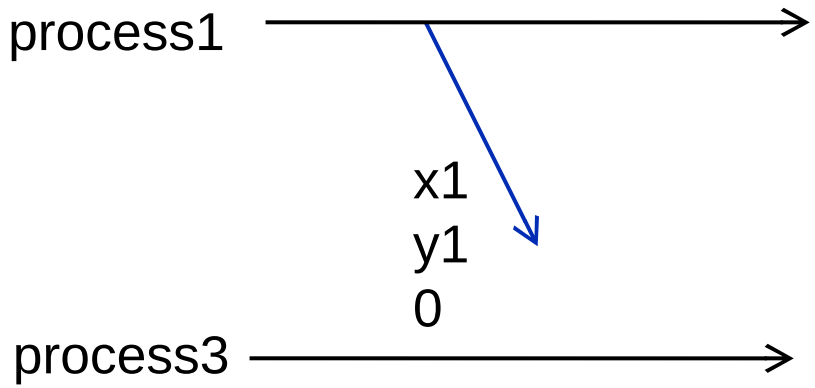
To deliver, needs to at least have

$\underline{M[1, 3] = x1 - 1}$

$M[2, 3] \geq y1$

$M[3, 3] \geq 0$

The underlined condition can never be met



# Need More Correctness Proof

- What we proved so far:
  - If s1 happened before s2, then r2 will not be before r1
  - But we don't know if r1 and r2 will be delivered at all

- We will now prove that
  - At any given time, there must be one message that can be delivered to the process – induction will take care of the other messages

# More Correctness Proof

- Consider a given receiver process $j$ and its corresponding column in the matrix M

x_1
…
x_i
…
x_n

- Consider the non-empty set of all undelivered messages
  - Consider the senders of these messages
  - For each sender $i$ in the set, the exists an undelivered message whose matrix T has the column

…
x_i + 1
…

call such undelivered message successor messages

# More Correctness Proof

- Consider the set of successor messages
  - This set must be non-empty
  - This set has at most one message from any given sender
  - This set has at most $n$ messages total

- Consider all the corresponding "send" events of the successor messages
  - There must be a send event $s$ that does not have any other send events that happened before $s$ – why?
  - There can be multiple such $s$ events
  - Call the corresponding message top successor message

# More Correctness Proof

- Claim: Any top successor message can be delivered
  - W.l.o.g, assume the message's sender is process1

M =
x_1
x_2
…
x_n

The matrix column on the receiver (process *j*)

T =
x_1 + 1
y_2
…
y_n

The matrix column in the message

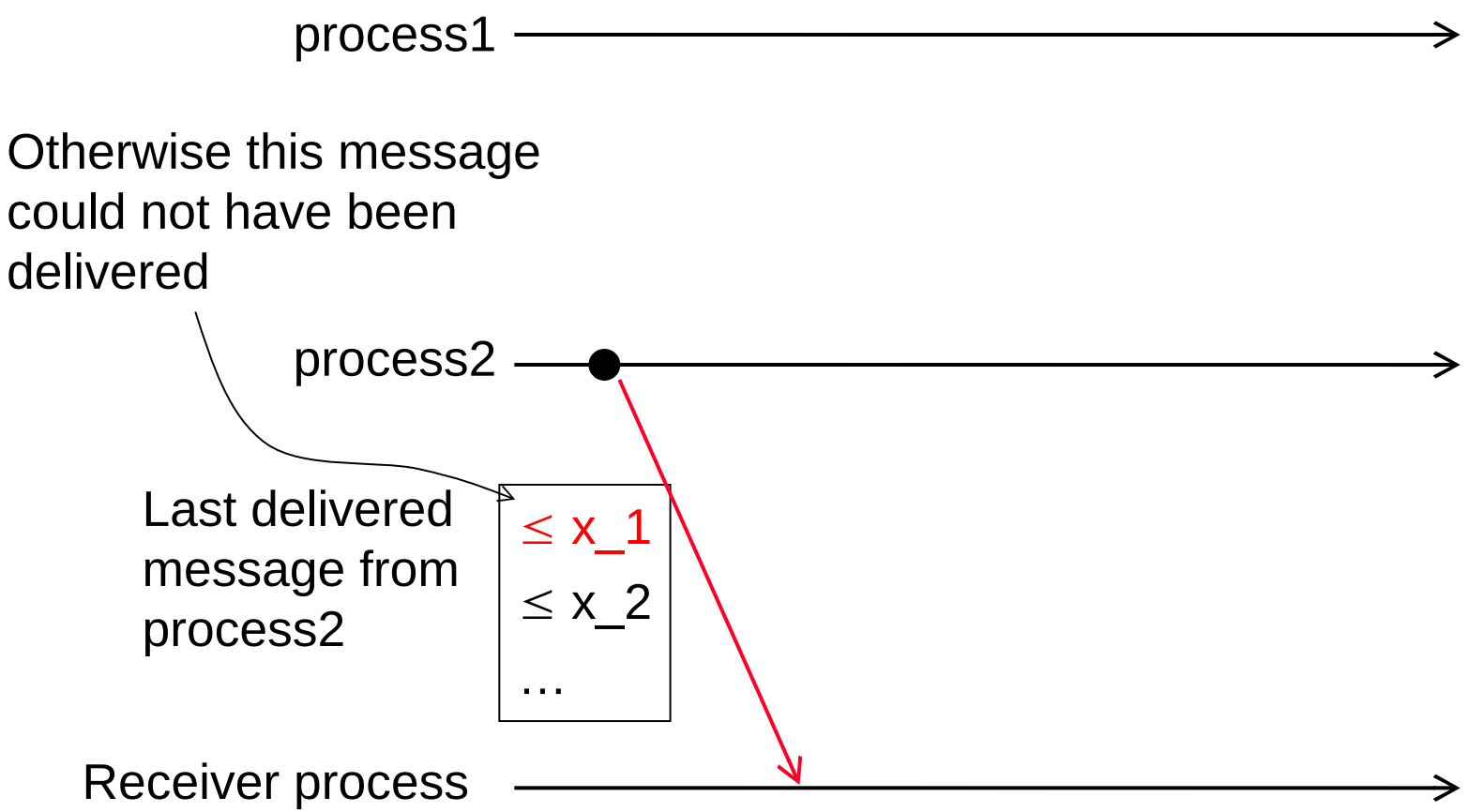
need to show:

$x_i \geq y_i$ for $2 \leq i \leq n$

We prove the case for i = 2. Other cases are the same.

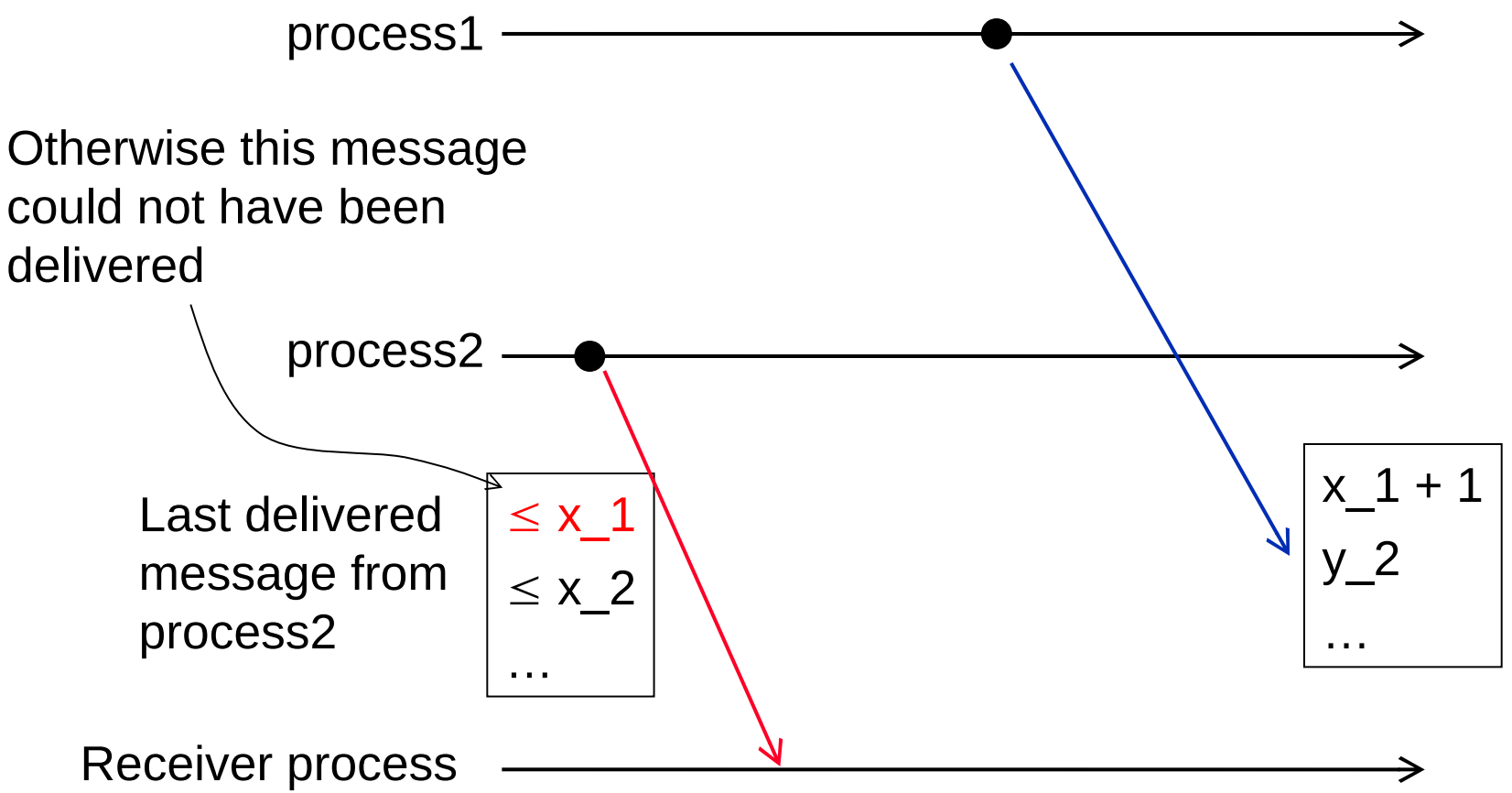
# More Correctness Proof

- Prove by contradiction: Suppose x_2 < y_2. Note that this entry corresponds to the # messages sent from process 2 to the receiver process



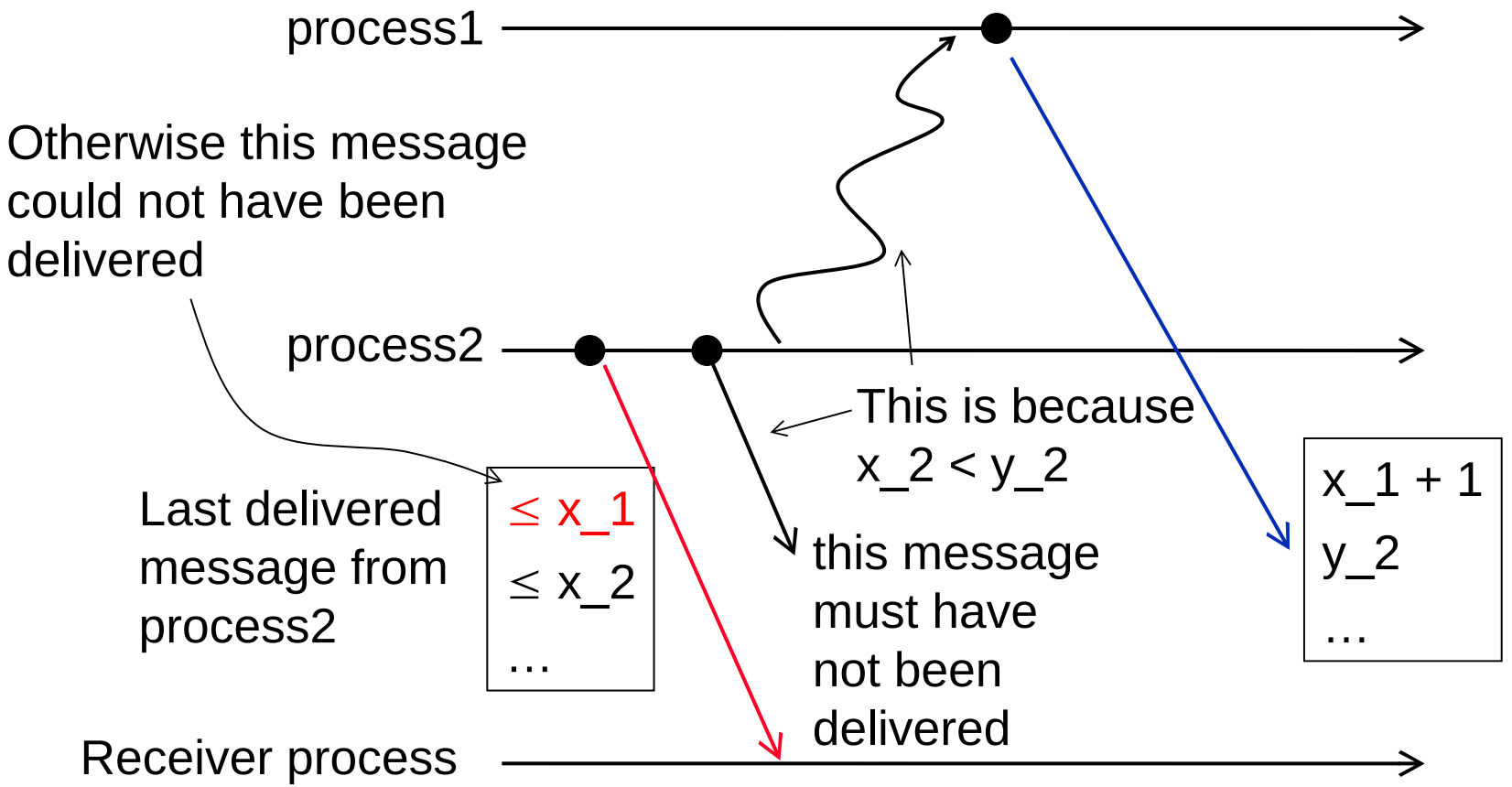
# More Correctness Proof

- Prove by contradiction: Suppose x_2 < y_2. Note that this entry corresponds to the # messages sent from process 2 to the receiver process



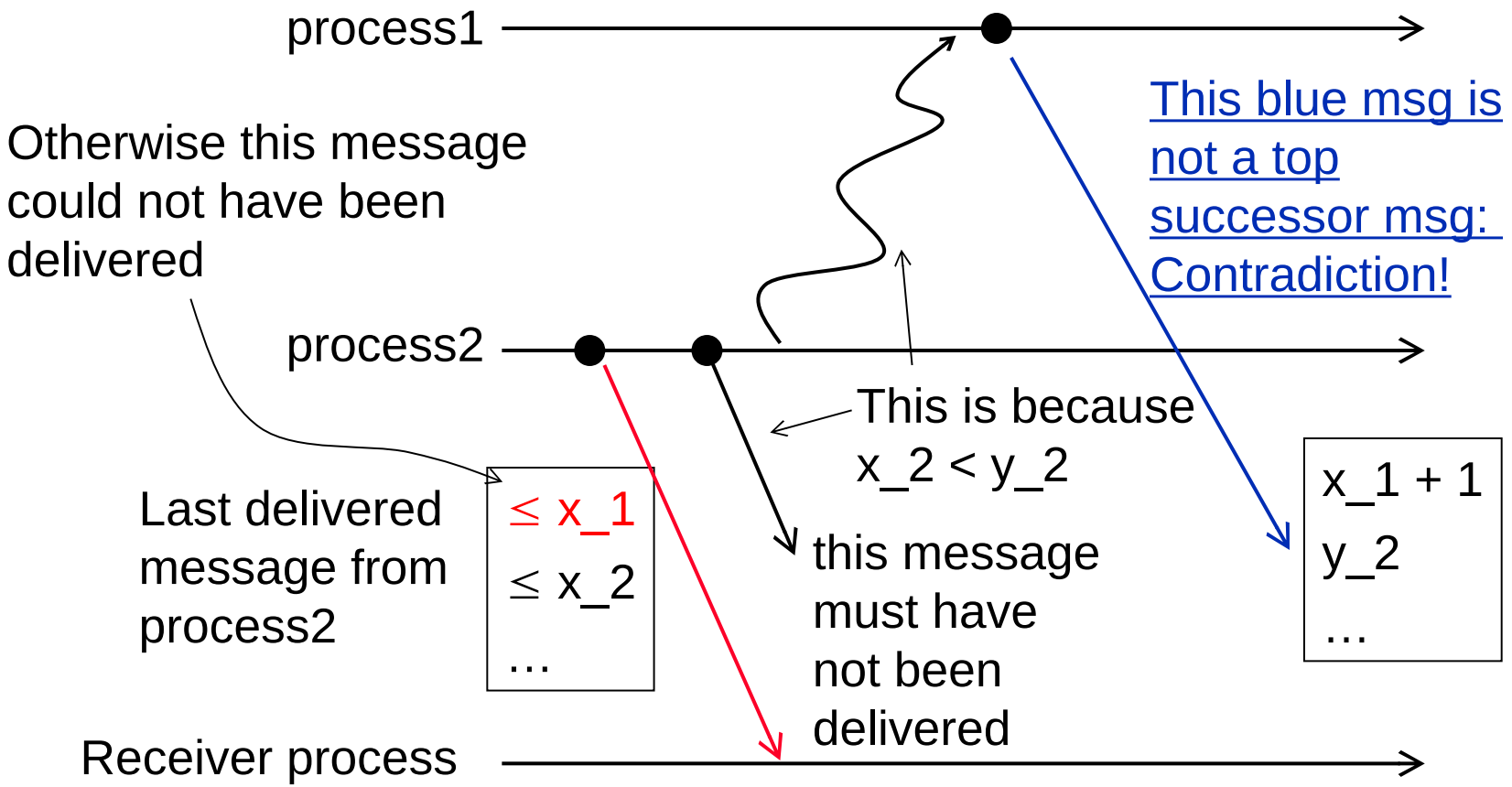
# More Correctness Proof

- Prove by contradiction: Suppose x_2 < y_2. Note that this entry corresponds to the # messages sent from process 2 to the receiver process



# More Correctness Proof

- Prove by contradiction: Suppose x_2 < y_2. Note that this entry corresponds to the # messages sent from process 2 to the receiver process



# Summary of Correctness Proof

- All messages will eventually be delivered
- The delivery order satisfies causal order

# Causal Ordering of Broadcast Messages

- Broadcast: Every message is sent to all people (including the sender itself)
  - Modeled as n point-to-point messages
- Application: Internet chat room
- For same reason as before, we may need to ensure causal order among messages
  - Each process uses the previous protocol

# Total Ordering of Broadcast Messages

- All messages delivered to all processes in exactly the same order
  - Also called atomic broadcast
- Total ordering only applies to broadcast messages

- Now assume that all messages in the system are broadcast messages (i.e., every message is sent to all nodes)
  - What is the relation between total order and causal order?

Quick Poll https://pollev.com/haifengyu229

# Application: Internet Chat Room

- We want to assign numbers to people have said
  - The number have to be consistent across all users

[0] Alice: Welcome!
[1] Bob: Hello, Alice
[2] Alice: Let's try to prove P=NP
[3] Bob: OK, where do we start?
….
….
….
[1712] Bob: I am confused, what were we trying to prove?
[1713] Alice: Please refer to message [2]

# Application: Distributed Ledger / Blockchain

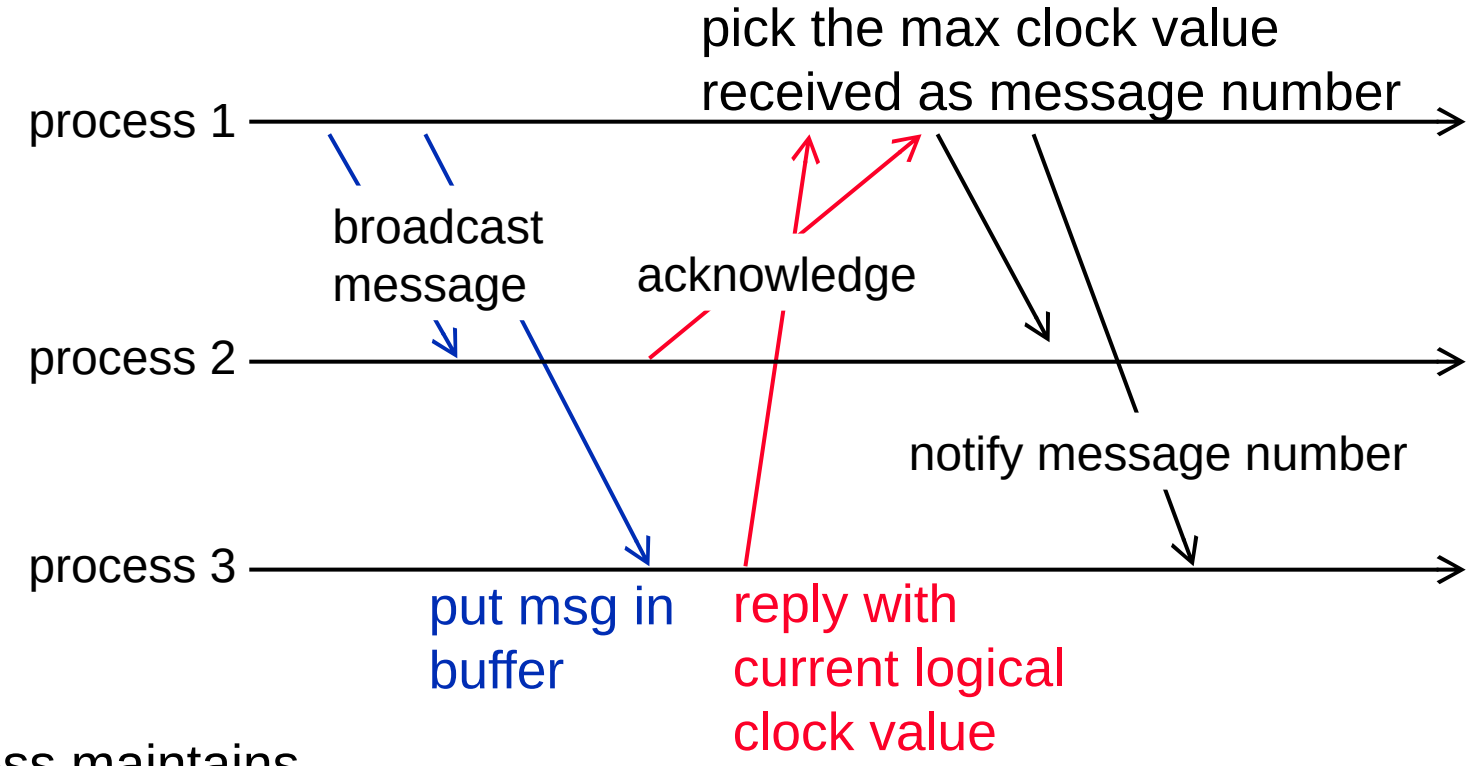
- Each node maintains a ledger of transactions
  - Alternatively, each node maintains a list of blocks, where each block contains a list of transactions
- Each new transaction is broadcast to all nodes
- We want all the nodes to have the same ordering for all the transactions
  - This ensures consistency across the ledgers on all nodes
- Our protocol next are assuming no failures…

# Using a Coordinator for Total Order Broadcast

- A special process is assigned as the coordinator
- To broadcast a message
  - Send a message to the coordinator
  - Coordinator assigns a sequence number to the message
  - Coordinator forward the message to all processes with the sequence number
  - Messages delivered according to sequence number order

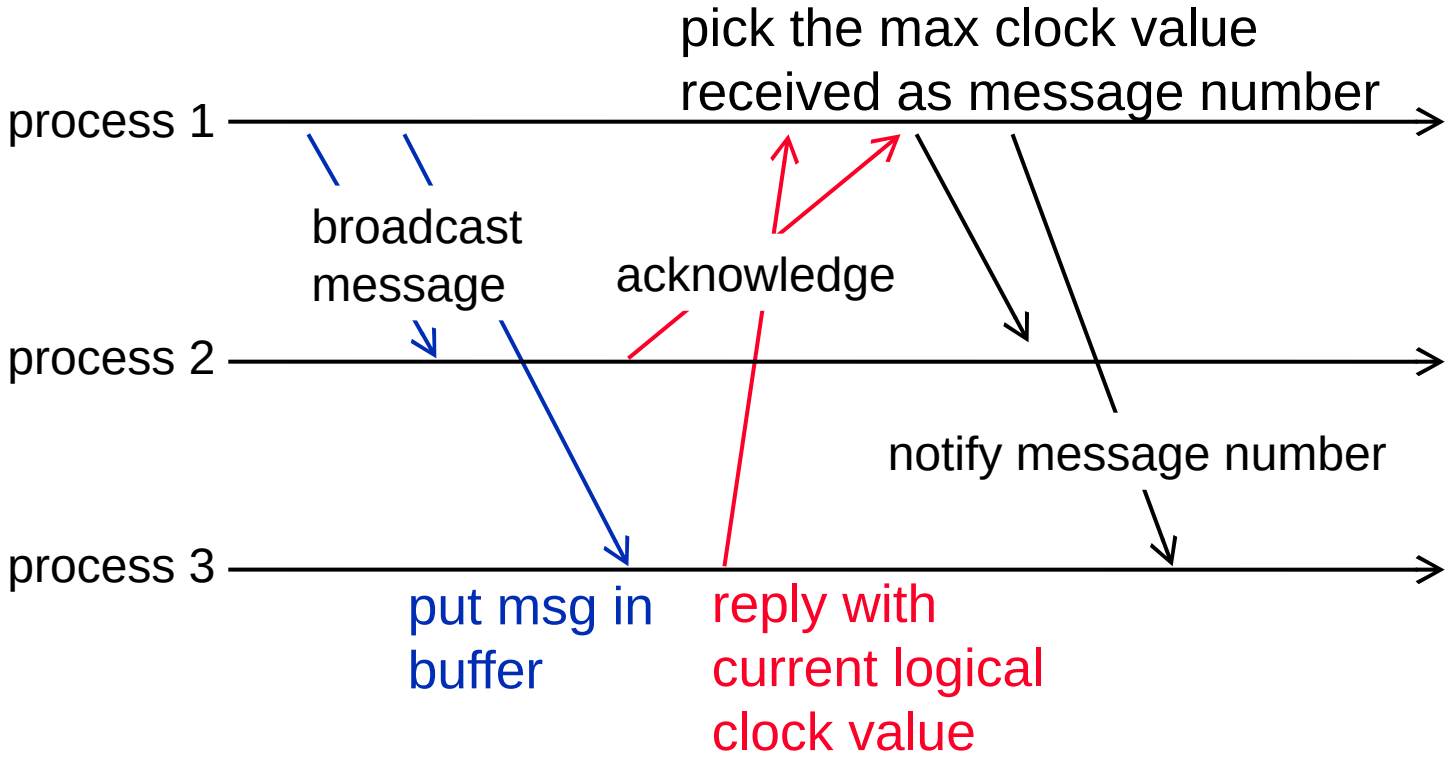
- Problem:
  - Coordinator has too much control

# Skeen's Algorithm for Total Order Broadcast



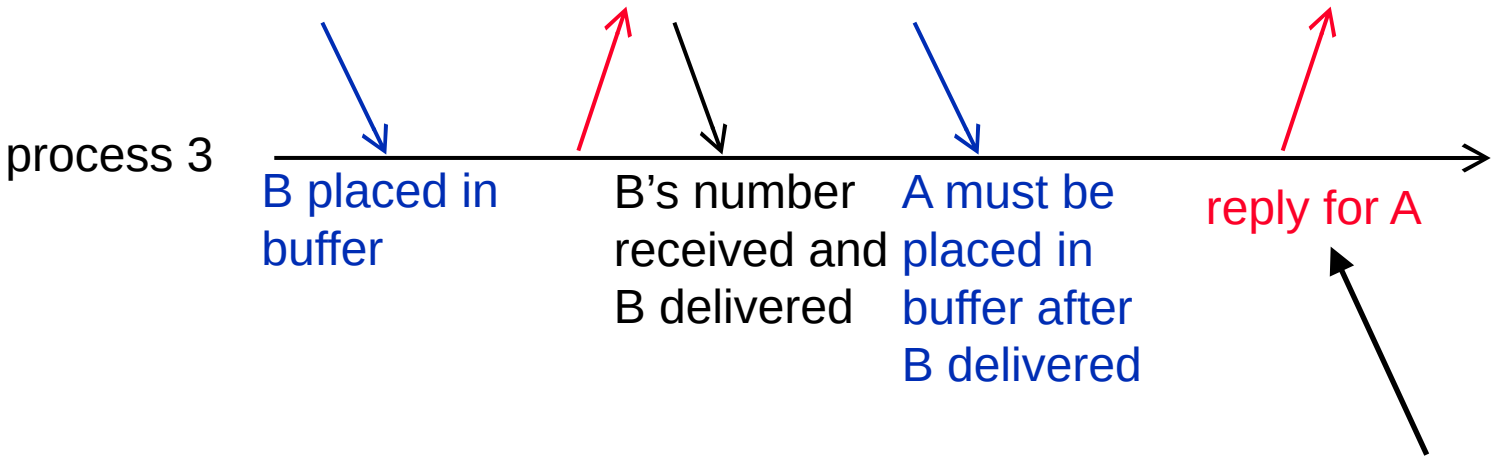

- Each process maintains
  - Logical clock and a message buffer for undelivered messages
- A message in the buffer is delivered / removed if
  - All messages in the buffer have been assigned numbers
  - This message has the smallest number

# Correctness Proof for Skeen's Algorithm




- Claim: All messages will be assigned message numbers
- Claim: All messages will be delivered
- Claim: If message A has a number smaller than B, then B is delivered after A – Prove by contradiction on next slide (trivial?)

# Correctness Proof for Skeen's Algorithm -- Continued




key: Process 3's logical clock now must be larger than B's number

- Suppose A is delivered on process 3 after B.
- Then A must have been placed in buffer after B was delivered
- A must have a number larger than B – Contradiction.

# Summary

- “Message Ordering”
- FIFO ordering for point-to-point messages
  - Already discussed last lecture
- Causal ordering for point-to-point messages
  - Applications
  - Protocol to ensure causal ordering
- Causal ordering for broadcast messages
  - Protocol
- Total ordering for broadcast messages
  - Application
  - Skeen’s algorithm

# Homework Assignment (on this and next few slides)

- Show the relationship between conditions (C1), (C2), and (C3) on message delivery of a system.

  (C1): $s1 \rightarrow s2 \Rightarrow \neg(r2 \rightarrow r1)$

  (C2): $s1 < s2 \Rightarrow \neg(r2 \rightarrow r1)$

  (C1): $s1 \rightarrow s2 \Rightarrow \neg(r2 < r1)$

  Here s1 and s2 are sends of any two messages, and r1 and r2 are the corresponding receives. Note that a computation satisfies a delivery condition if and only if the condition is true for all pairs of messages. Here "s1 $\rightarrow$ s2" means that s1 happened-before s2, and "s1 $<$ s2" means that s1 and s2 are on the same process and s1 is before s2.

  For each pair (x, y) of the three conditions (i.e., C1 & C2, C1 & C3, C2 & C3)

  - Prove x implies y and y implies x, OR
  - Prove x implies y, and a counter example that satisfies y but not x, OR
  - Prove y implies x, and a counter example that satisfies x but not y, OR
  - A counter example that satisfies y but not x, and a second counter example that satisfies x but not y

- For the protocol on Slide 23, does the resulting total order always satisfy causal order? Why?

- Bring your completed homework to class next week.