Checkpointing and Recovery in Distributed Systems

Advanced Operating Systems

Neeraj Mittal

March 14, 2013

Outline

- Overview
- Main Issues
- Checkpointing and Recovery Protocols
  - Koo and Toueg's Protocol
  - Juang and Venkatesan's Protocol

Outline

- Overview
- Main Issues
- Checkpointing and Recovery Protocols
  - Koo and Toueg's Protocol
  - Juang and Venkatesan's Protocol

Overview

- System components often fail in real-world for many different reasons
  - Power outage
  - Operating system crash
  - Memory leakage
- Significant problem for long running applications
  - Likelihood that some component fails while the application is running is quite high

Tolerating Failures

- Assumption: Failed component eventually recovers and the application process is restarted
- Naïve Solution: Restart the application from the beginning whenever a failure occurs
  - All work done by the application before the failure is lost and has to be redone
  - No guarantee that the application will ever complete successfully!

Tolerating Failures (Contd.)

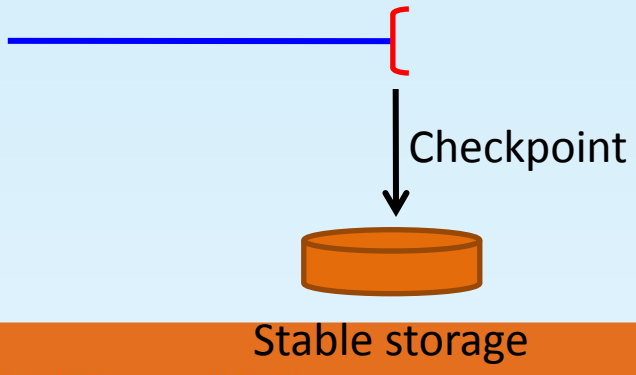
- Regularly save process' current state in reliable storage (e.g., hard disk)
  - Process' state is referred to as checkpoint
    - A checkpoint stores values of all the relevant program variables and registers
  - Reliable storage is referred to as stable storage
    - Assumed to be unaffected by failures

Tolerating Failures (Contd.)

- Regularly save process' current state in reliable storage (e.g., hard disk)
  - Process' state is referred to as checkpoint
    - A checkpoint stores values of all the relevant program variables and registers
  - Reliable storage is referred to as stable storage
    - Assumed to be unaffected by failures

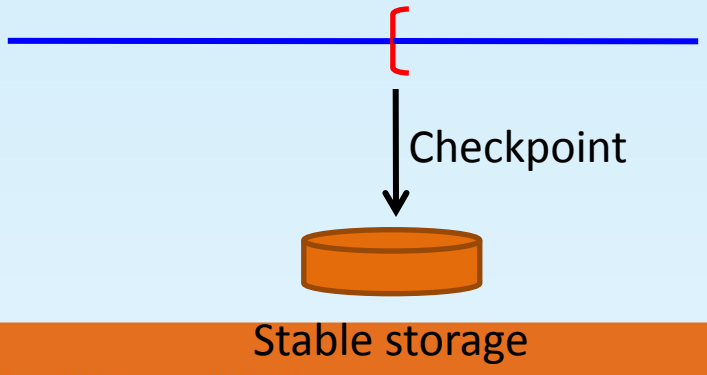
Tolerating Failures (Contd.)

- Regularly save process' current state in reliable storage (e.g., hard disk)
  - Process' state is referred to as checkpoint
    - A checkpoint stores values of all the relevant program variables and registers
  - Reliable storage is referred to as stable storage
    - Assumed to be unaffected by failures



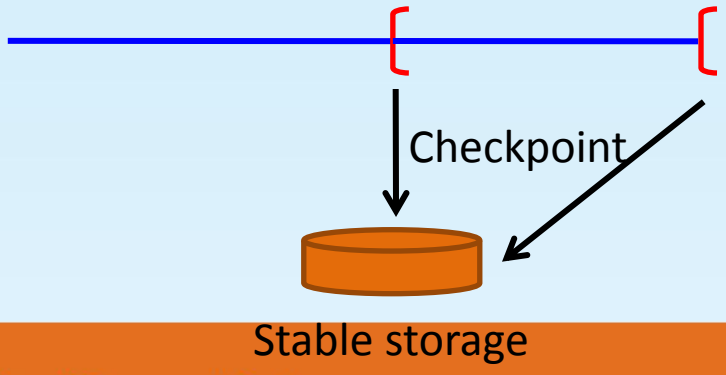
Tolerating Failures (Contd.)

- Regularly save process' current state in reliable storage (e.g., hard disk)
  - Process' state is referred to as checkpoint
    - A checkpoint stores values of all the relevant program variables and registers
  - Reliable storage is referred to as stable storage
    - Assumed to be unaffected by failures



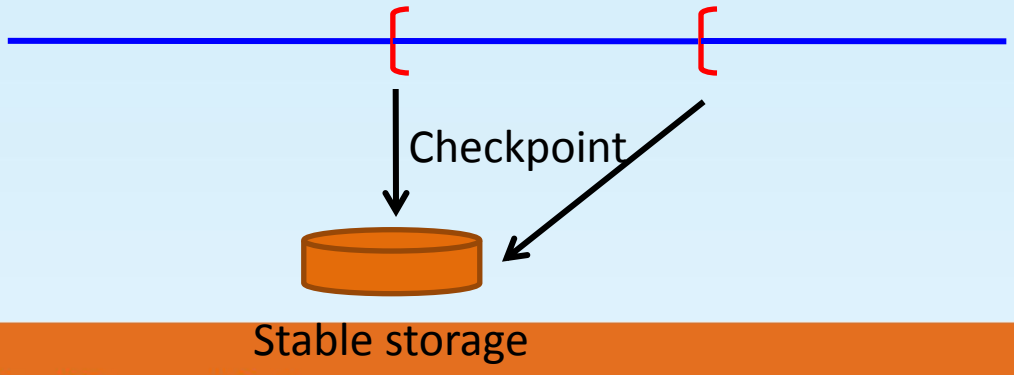
Tolerating Failures (Contd.)

- Regularly save process' current state in reliable storage (e.g., hard disk)
  - Process' state is referred to as checkpoint
    - A checkpoint stores values of all the relevant program variables and registers
  - Reliable storage is referred to as stable storage
    - Assumed to be unaffected by failures



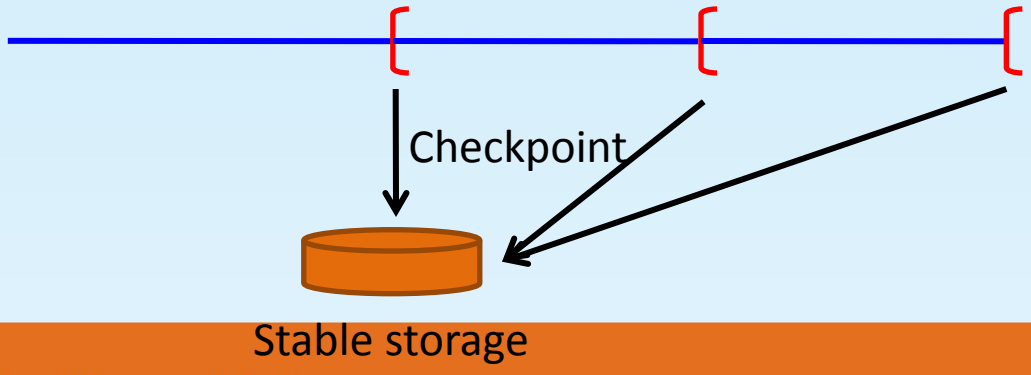
Tolerating Failures (Contd.)

- Regularly save process' current state in reliable storage (e.g., hard disk)
  - Process' state is referred to as checkpoint
    - A checkpoint stores values of all the relevant program variables and registers
  - Reliable storage is referred to as stable storage
    - Assumed to be unaffected by failures



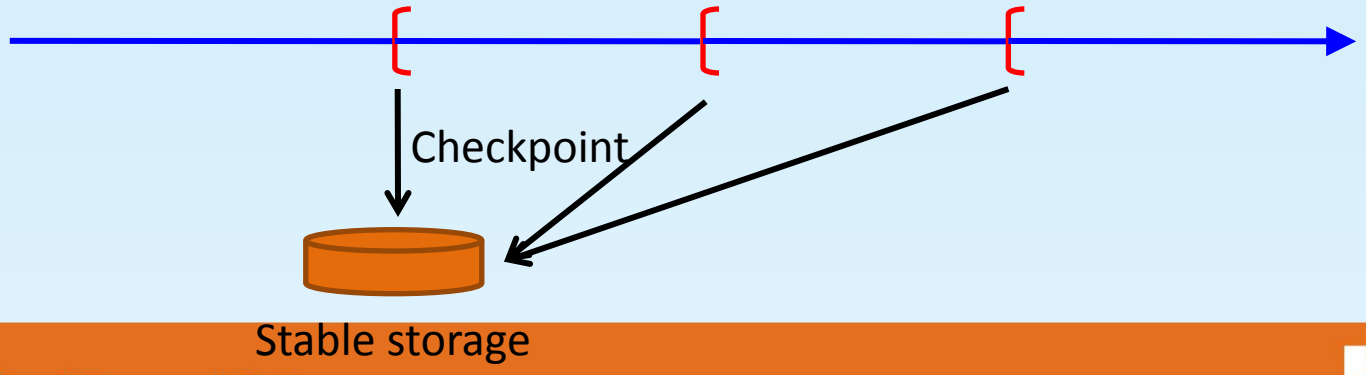
Tolerating Failures (Contd.)

- Regularly save process' current state in reliable storage (e.g., hard disk)
  - Process' state is referred to as checkpoint
    - A checkpoint stores values of all the relevant program variables and registers
  - Reliable storage is referred to as stable storage
    - Assumed to be unaffected by failures



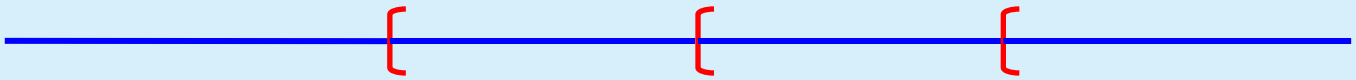
Tolerating Failures (Contd.)

- Regularly save process' current state in reliable storage (e.g., hard disk)
  - Process' state is referred to as checkpoint
    - A checkpoint stores values of all the relevant program variables and registers
  - Reliable storage is referred to as stable storage
    - Assumed to be unaffected by failures



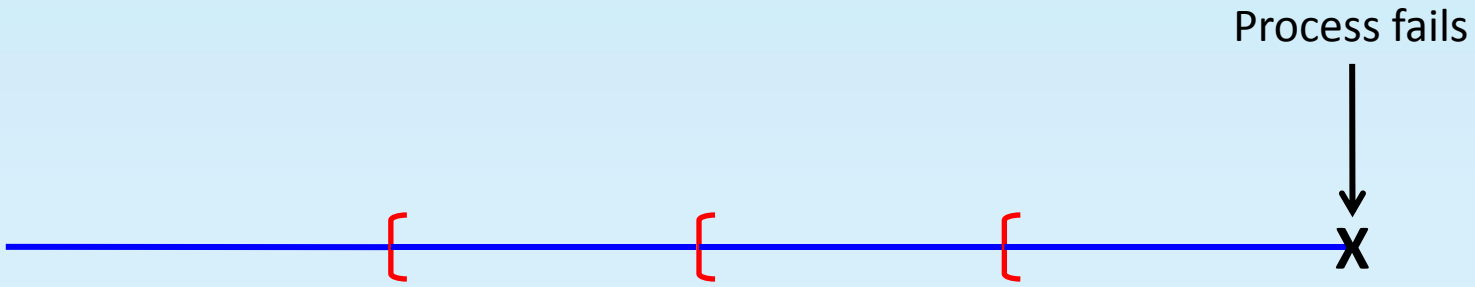
Tolerating Failures (Contd.)

- When a process fails and recovers, resume the execution from a saved checkpoint
  - Work done since the checkpoint until the failure point is lost



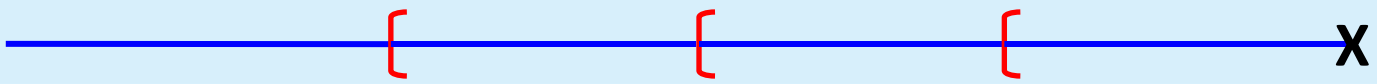
Tolerating Failures (Contd.)

- When a process fails and recovers, resume the execution from a saved checkpoint
  - Work done since the checkpoint until the failure point is lost



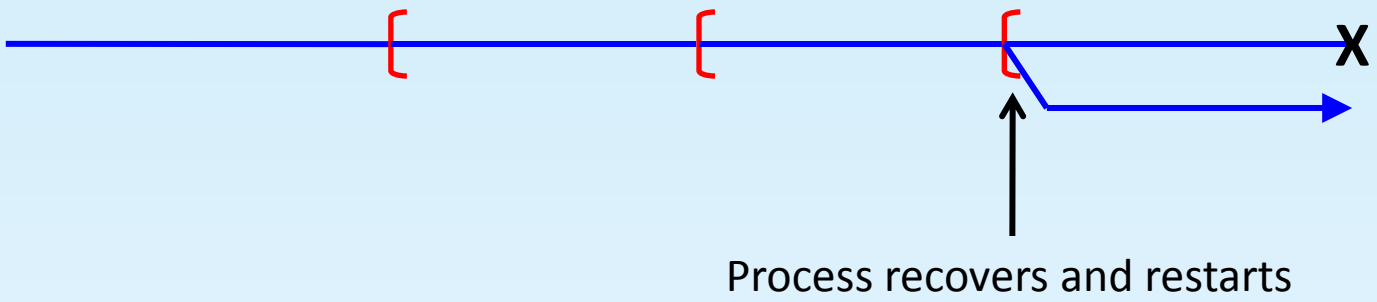
Tolerating Failures (Contd.)

- When a process fails and recovers, resume the execution from a saved checkpoint
  - Work done since the checkpoint until the failure point is lost



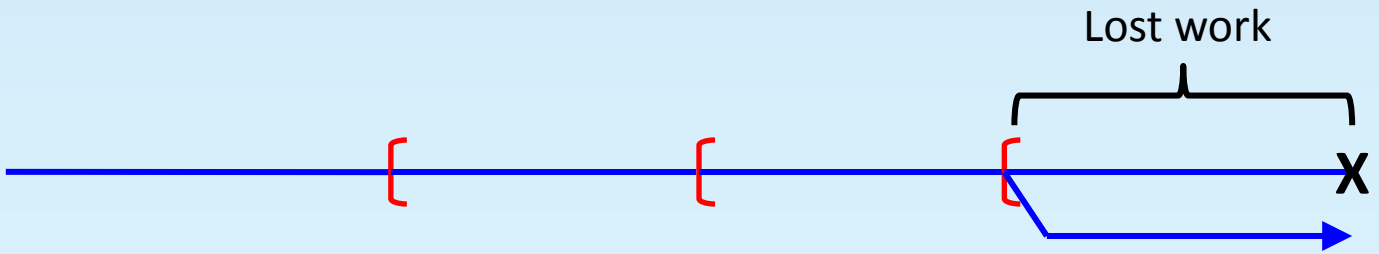
Tolerating Failures (Contd.)

- When a process fails and recovers, resume the execution from a saved checkpoint
  - Work done since the checkpoint until the failure point is lost



Tolerating Failures (Contd.)

- When a process fails and recovers, resume the execution from a saved checkpoint
  - Work done since the checkpoint until the failure point is lost



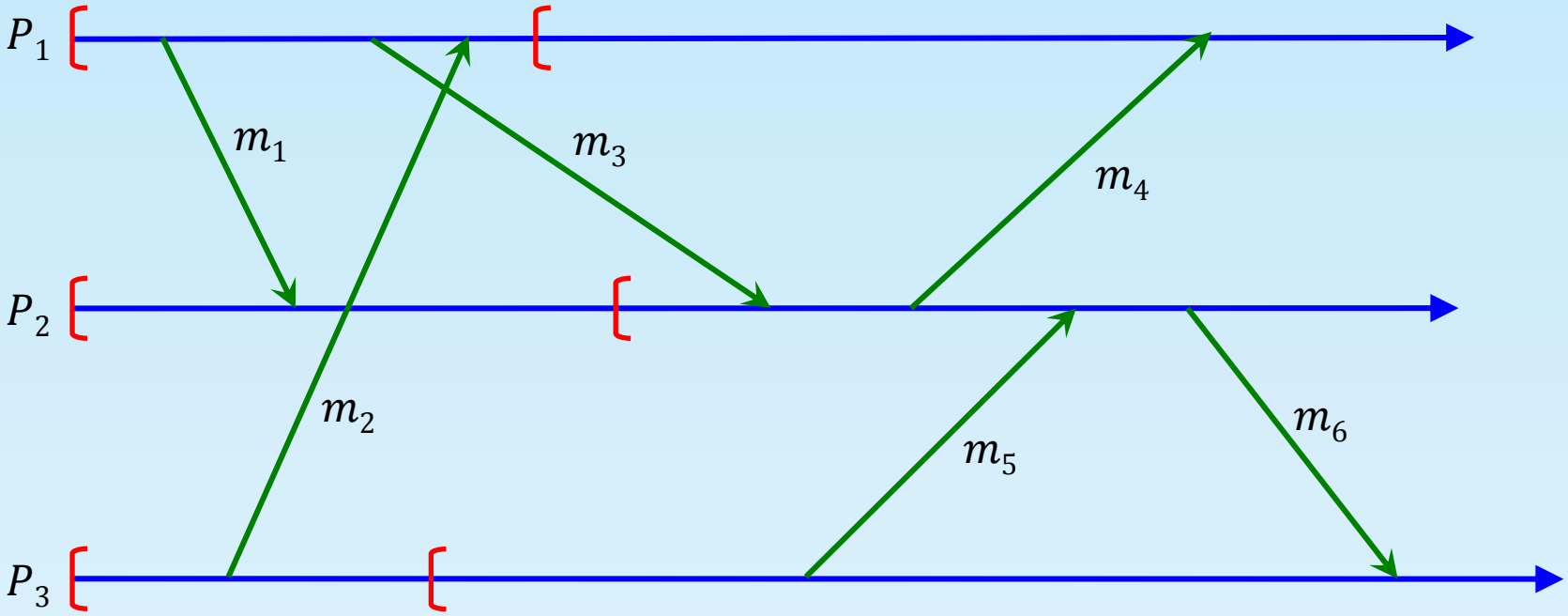
Outline

- Overview
- Main Issues
- Checkpointing and Recovery Protocols
  - Koo and Toueg's Protocol
  - Juang and Venkatesan's Protocol

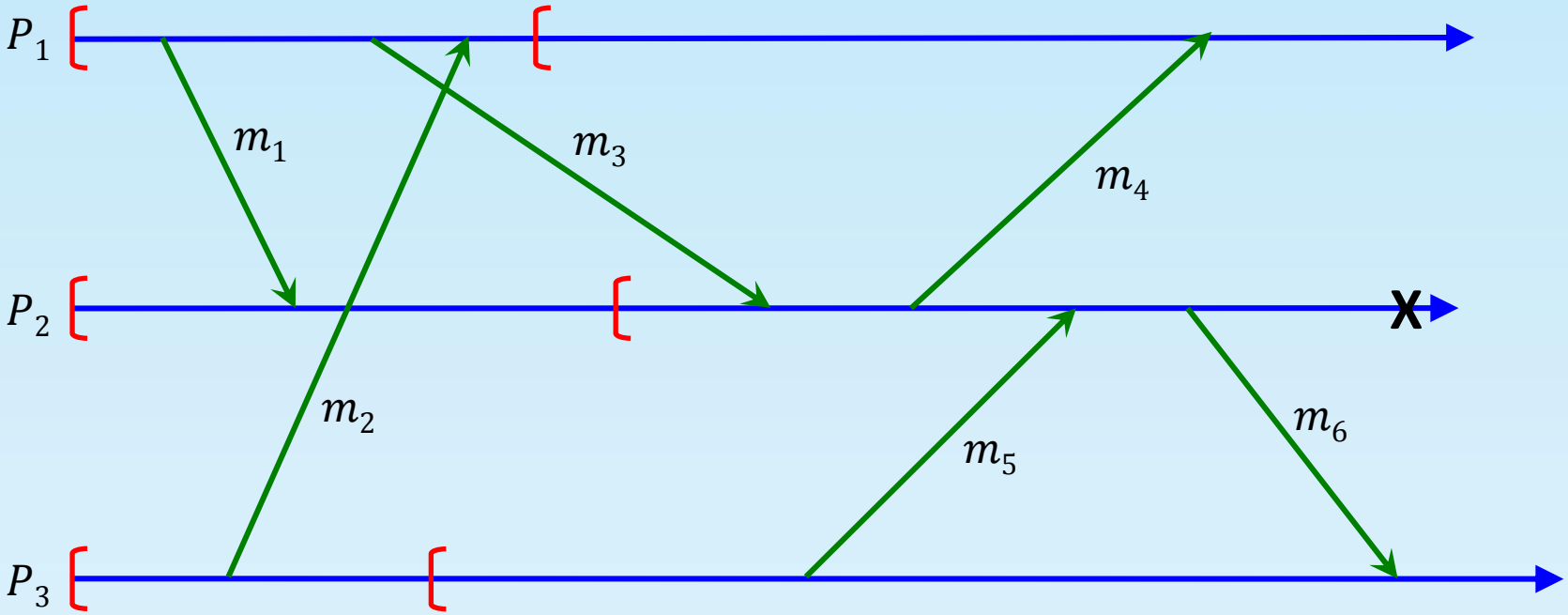
Issues in Distributed Systems

- Two main issues:
  - Lost messages: messages whose receive event has been lost
  - Orphan messages: messages whose send event has been lost

Lost and Orphan Messages: An Illustration

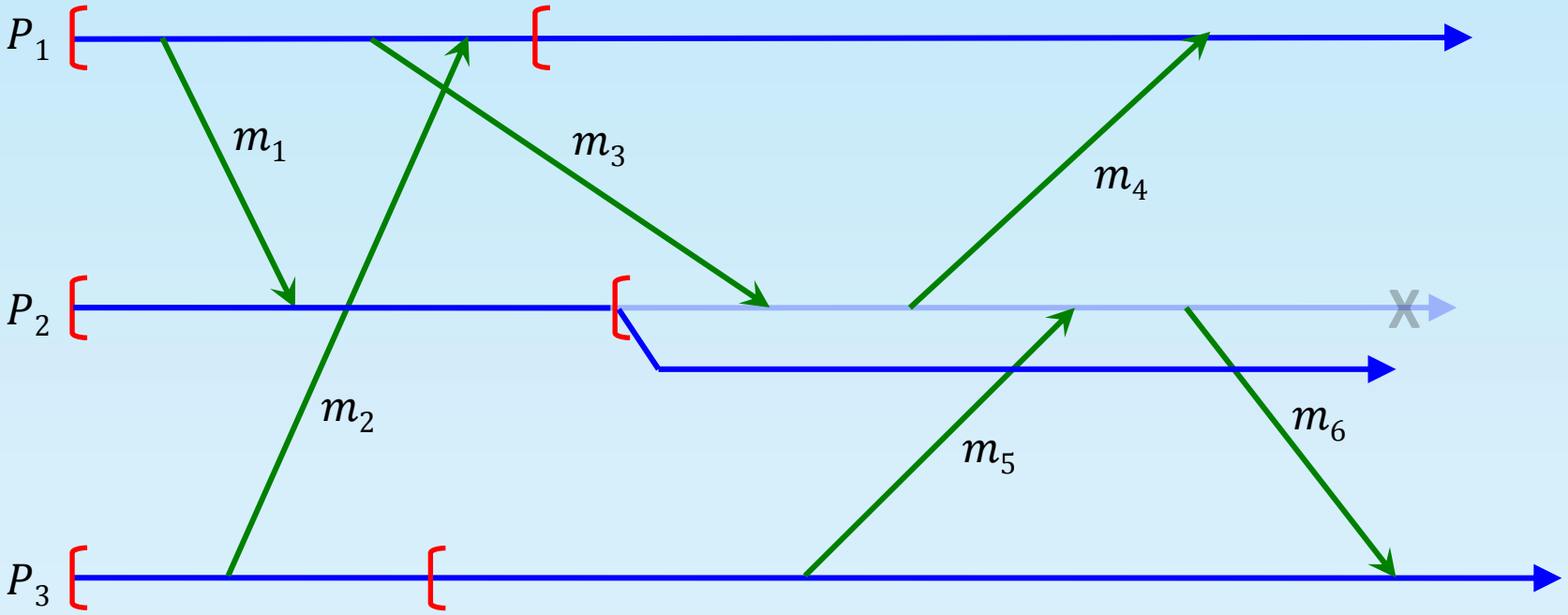


Lost and Orphan Messages: An Illustration



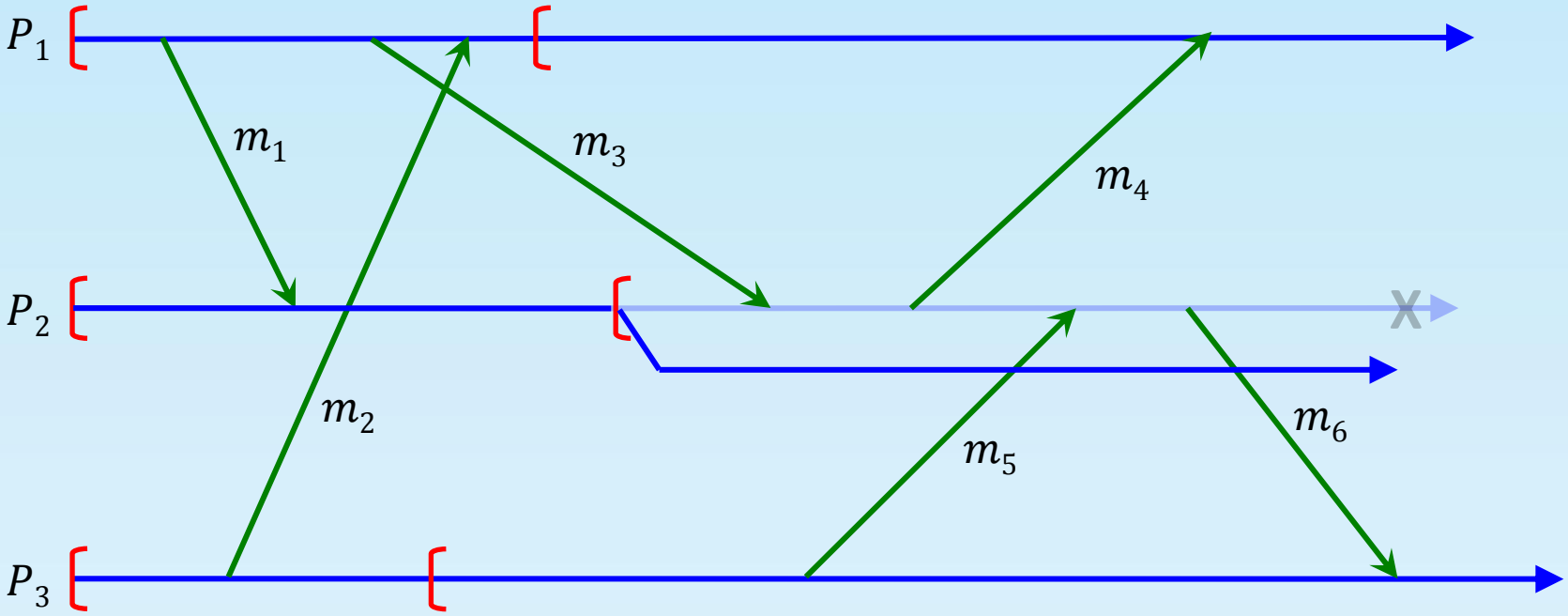
P_2 fails

Lost and Orphan Messages: An Illustration



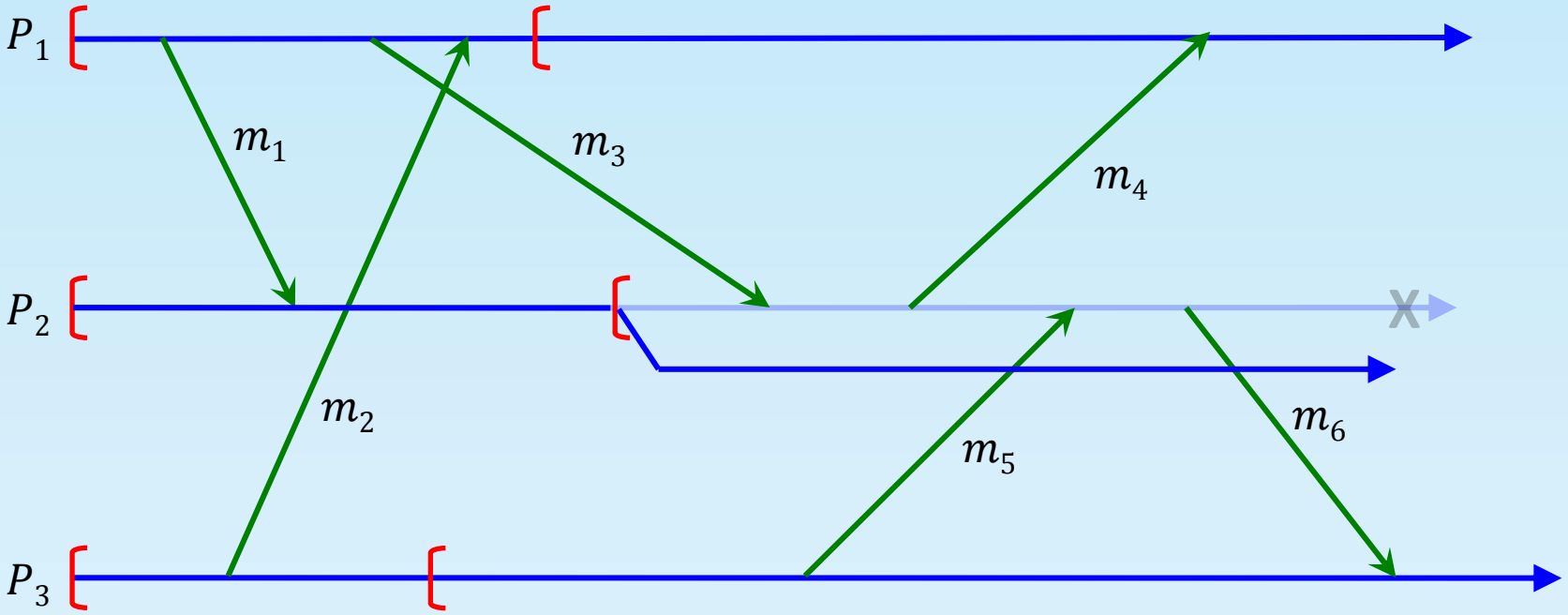
P_2 recovers and restarts

Lost and Orphan Messages: An Illustration



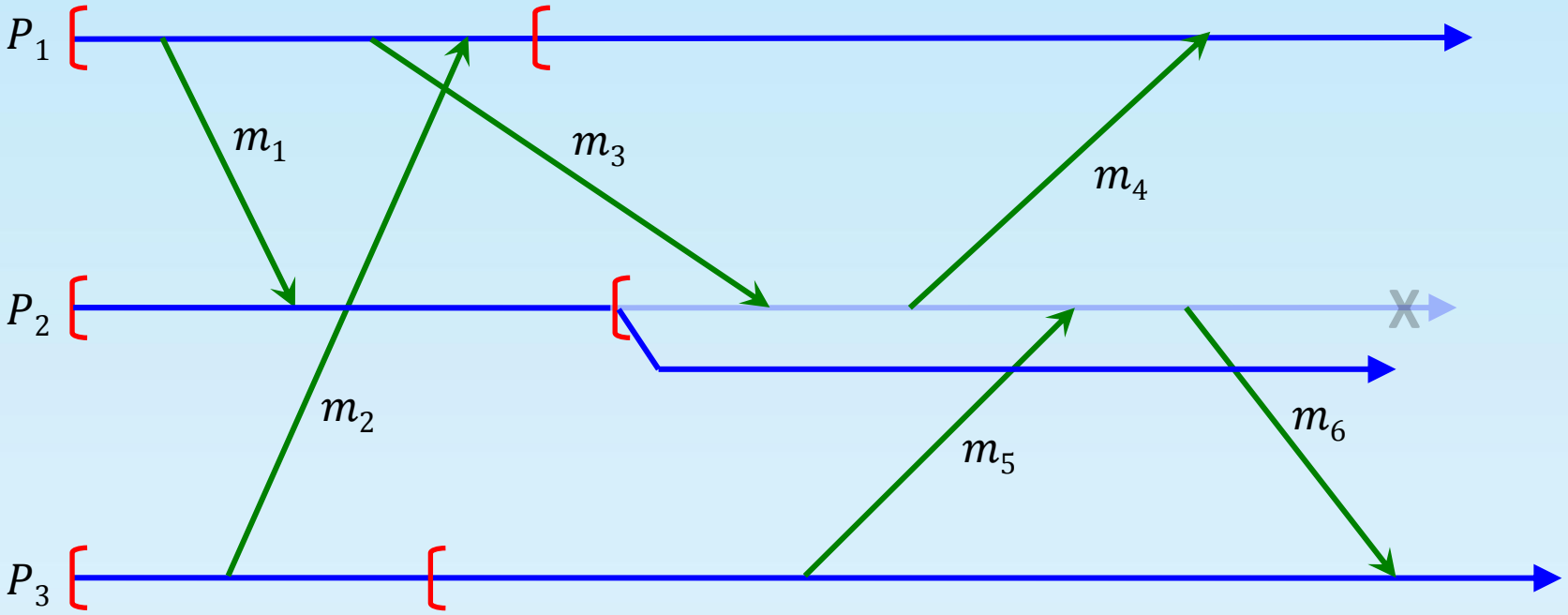
m_3 and m_5 become lost messages

Lost and Orphan Messages: An Illustration



m_4 and m_6 become orphan messages

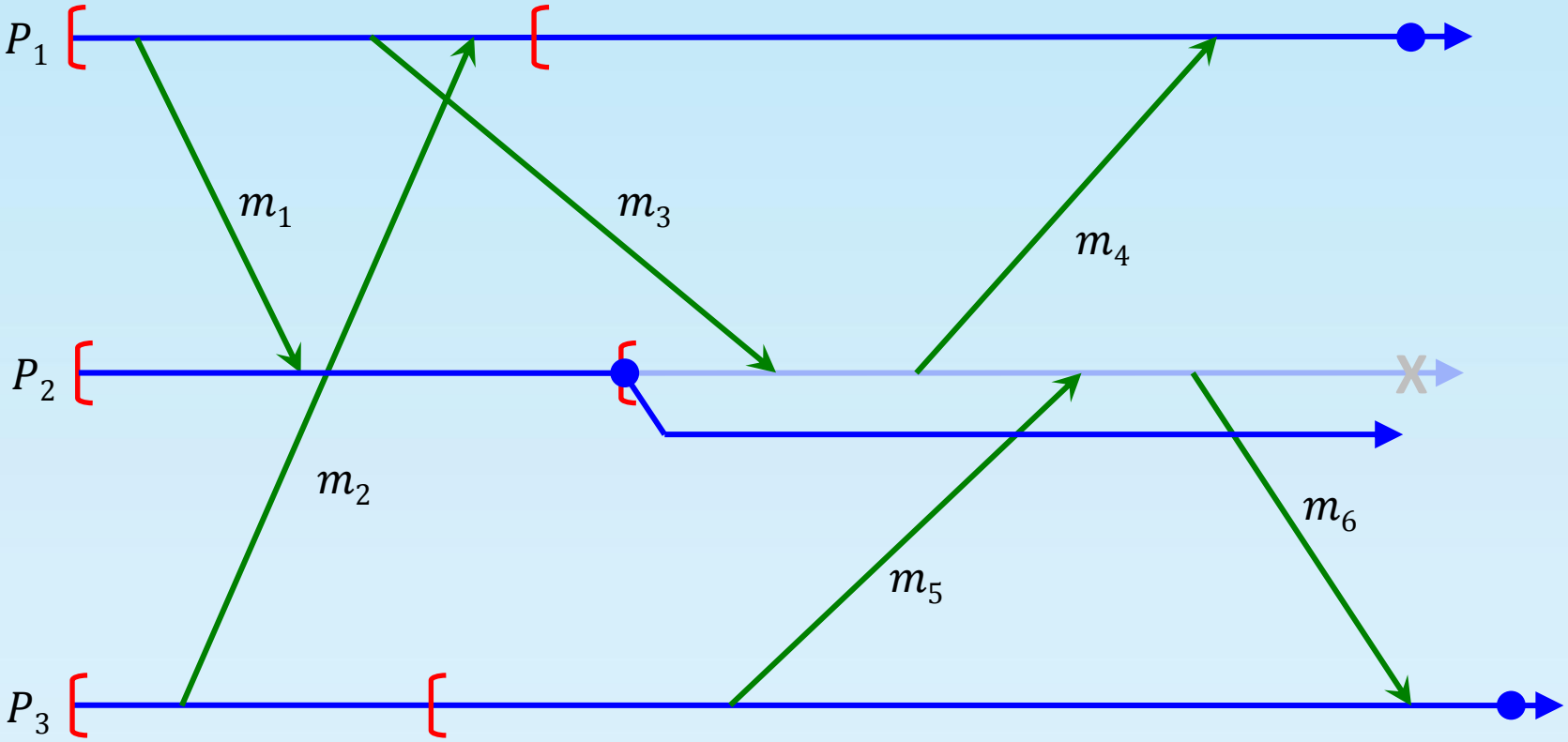
Lost and Orphan Messages: An Illustration



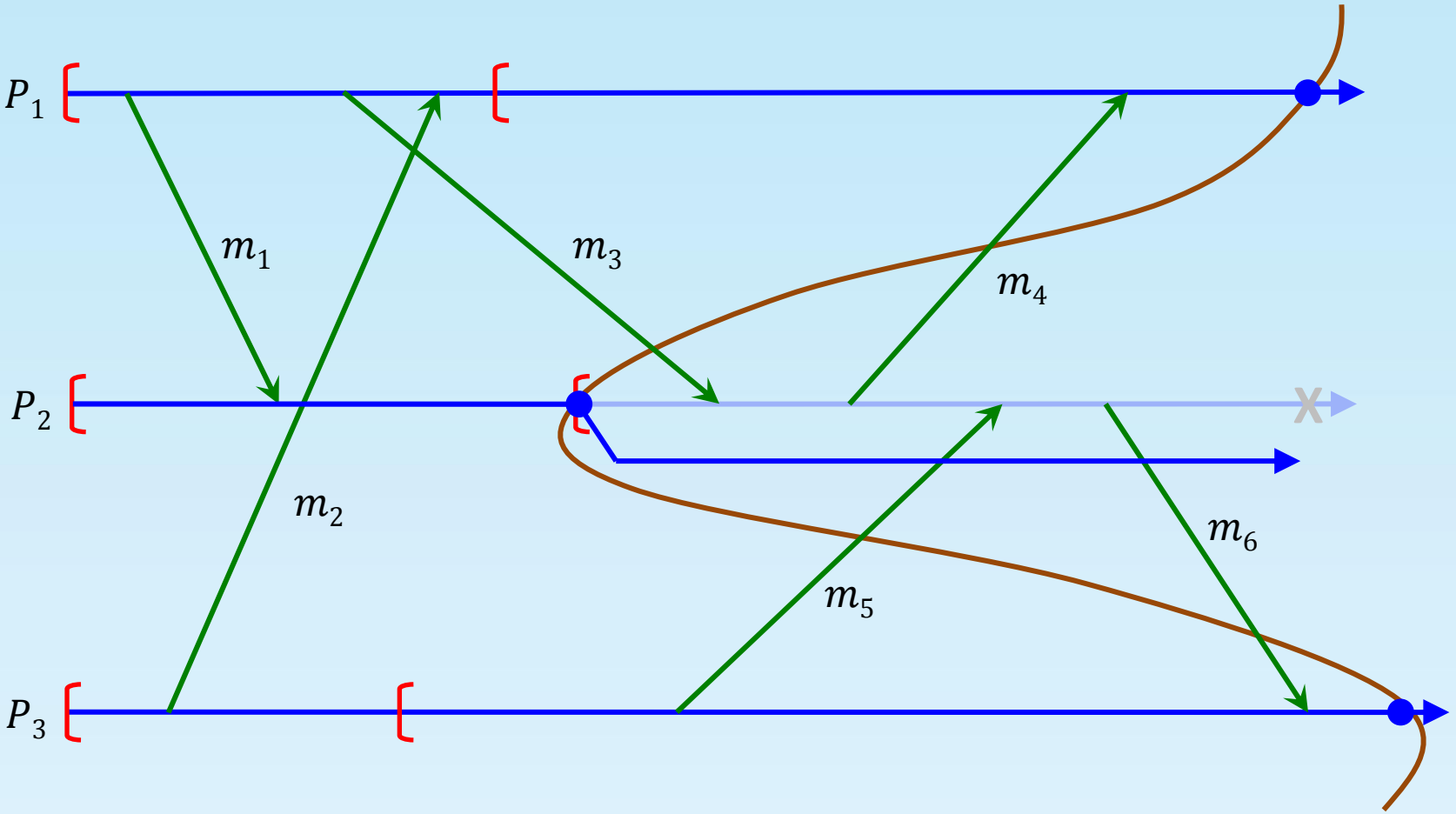
Lost and Orphan Messages

- Lost messages can be handled using retransmissions
  - At application layer
  - Requires message logging
- Orphan messages require more complex solution
  - System state depends on messages that no longer exist (and may never be generated)
  - Other processes need to be rolled back to eliminate any trace of orphan messages

Eliminating Orphan Messages: An Illustration

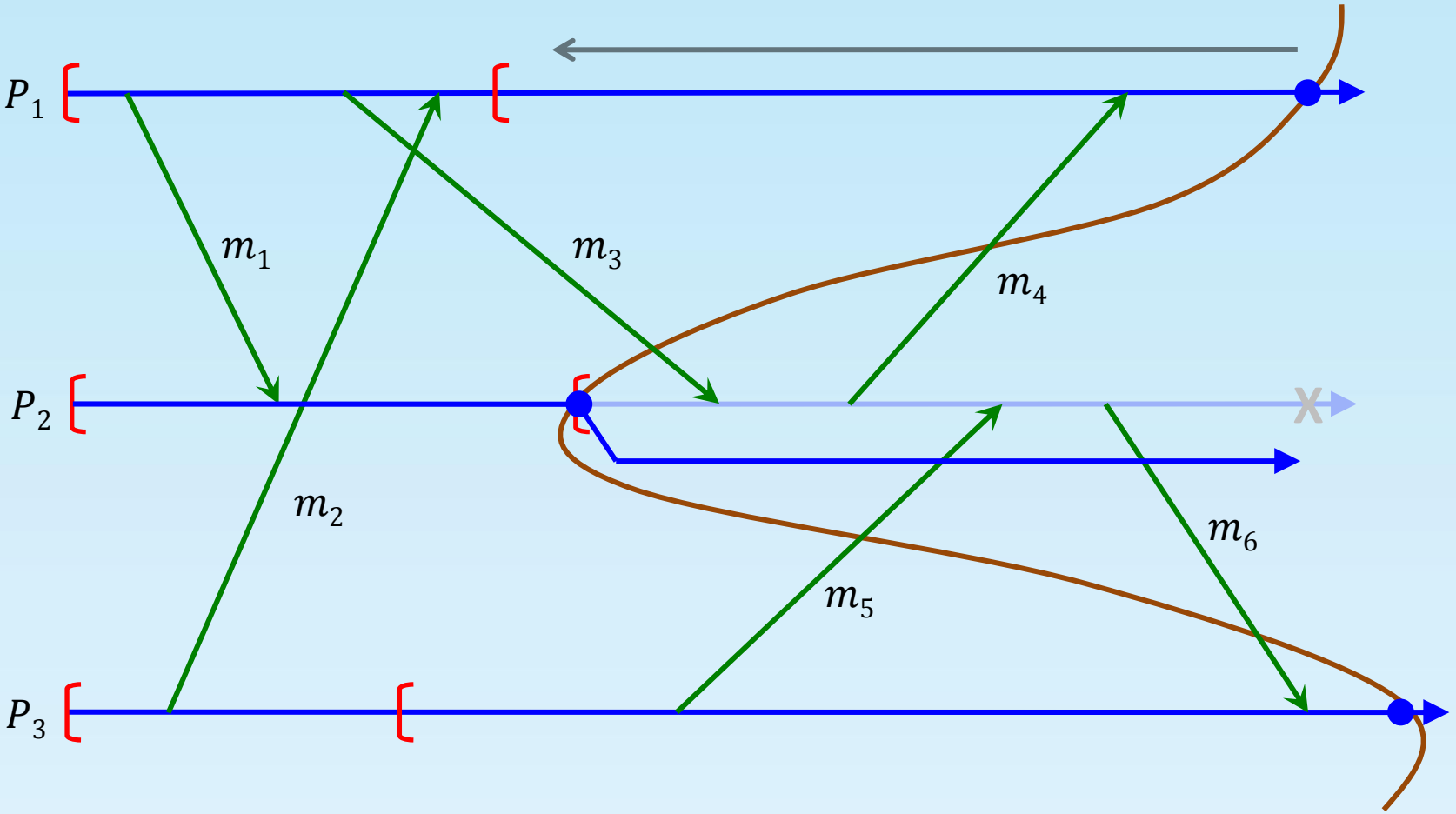


Eliminating Orphan Messages: An Illustration



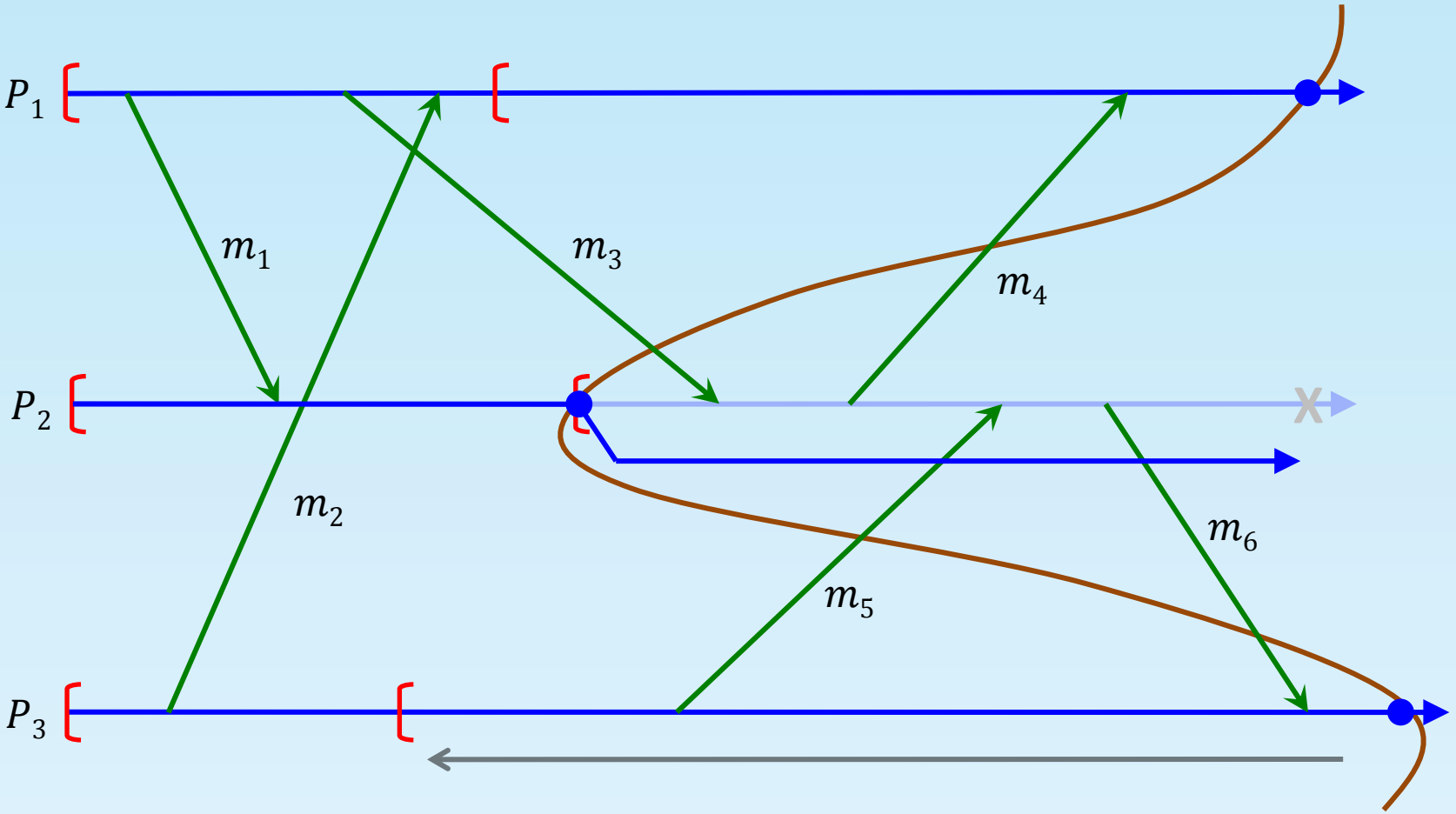
System state when P_2 restarts

Eliminating Orphan Messages: An Illustration



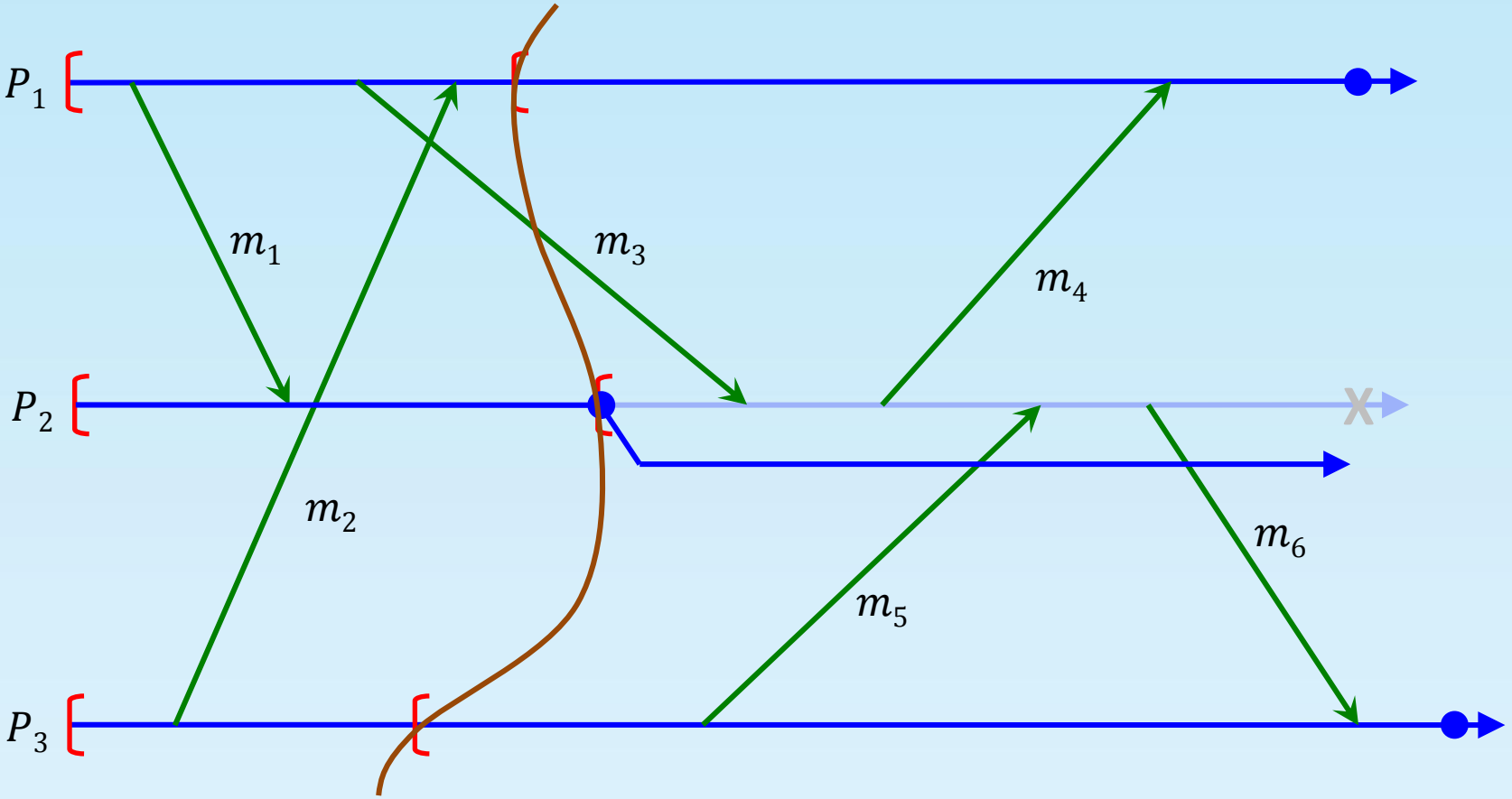
To eliminate trace of m_4, P_1 has to rollback

Eliminating Orphan Messages: An Illustration



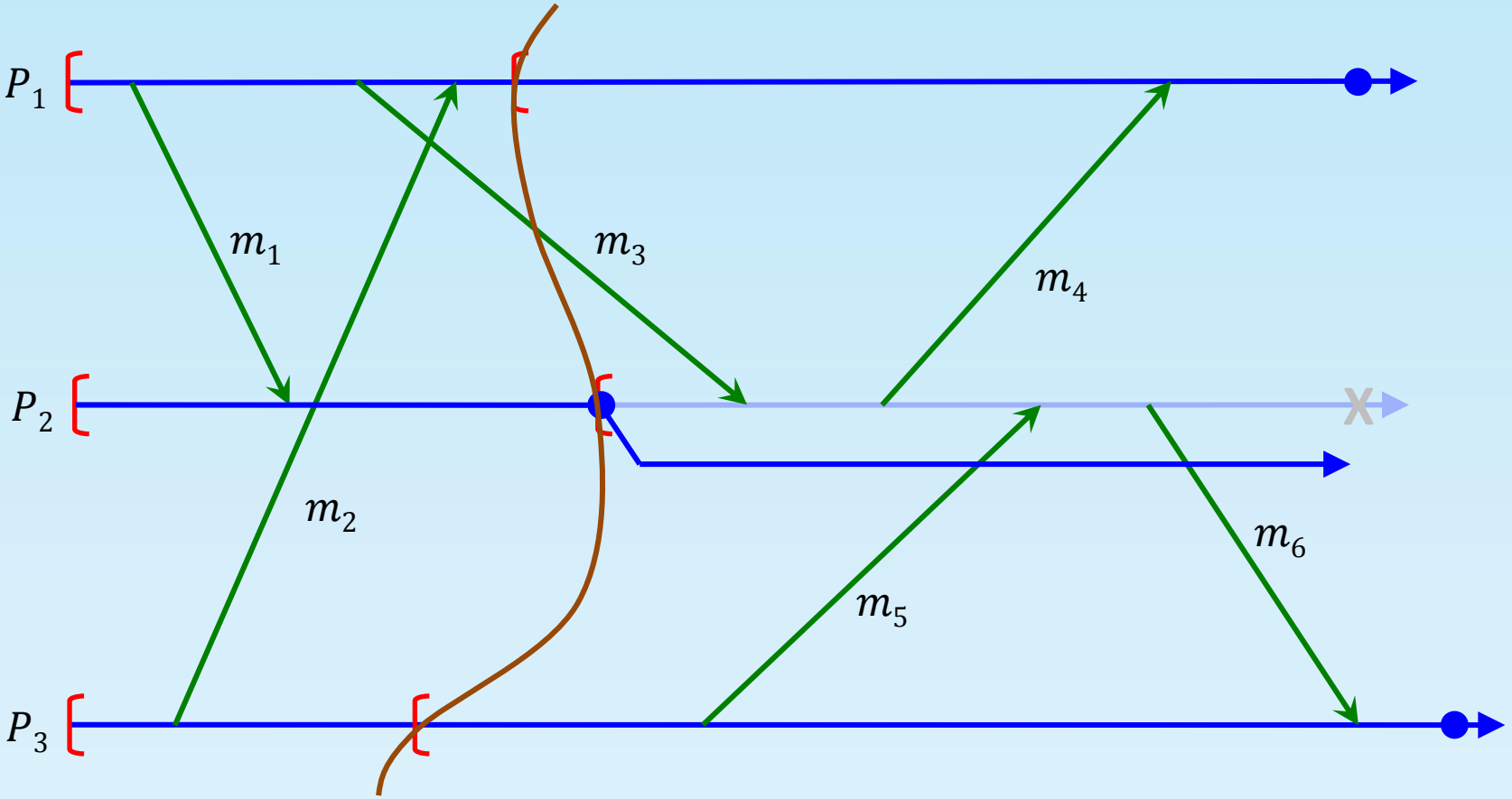
To eliminate trace of m_6, P_3 has to rollback

Eliminating Orphan Messages: An Illustration



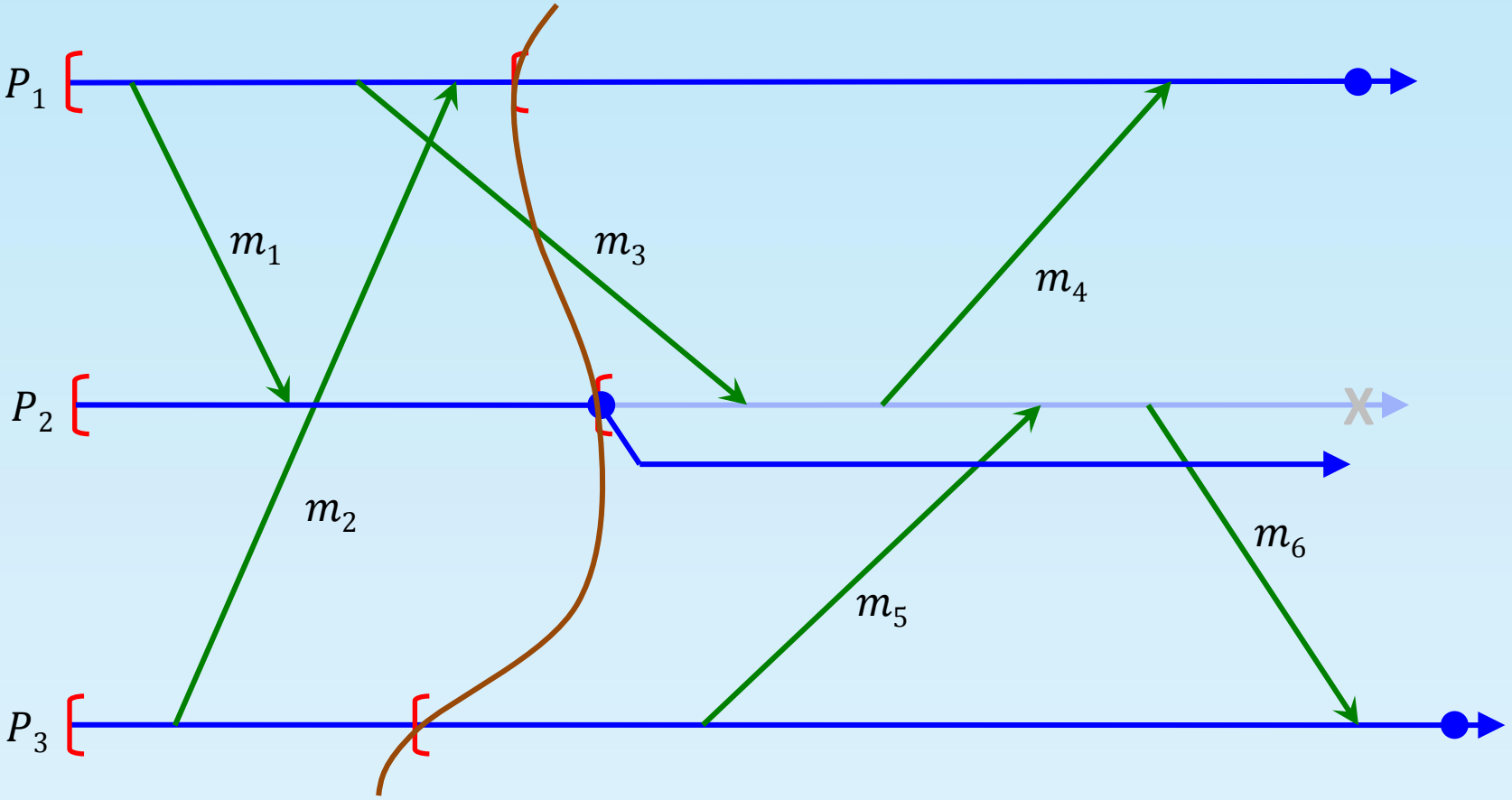
System state with all orphan messages eliminated

Eliminating Orphan Messages: An Illustration



Message m_3 is a lost message with respect to the recovery line

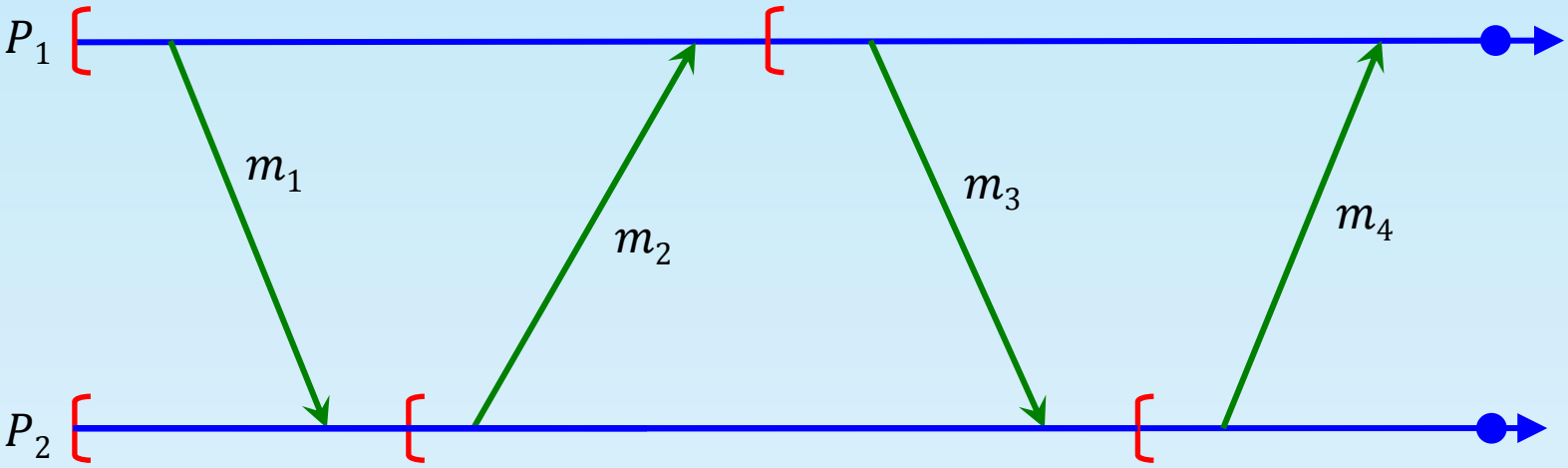
Eliminating Orphan Messages: An Illustration



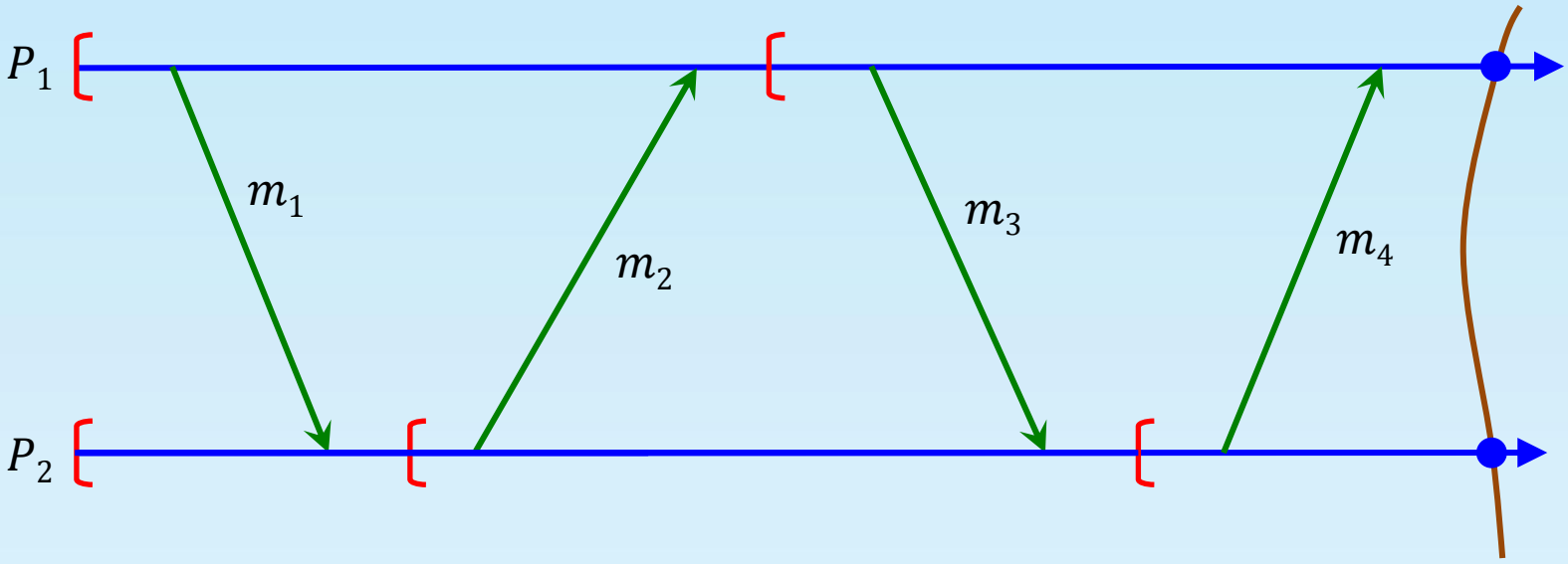
Domino Effect

- Rollback of processes that did not fail may cause other messages to become orphan
- May cause domino effect
  - Processes may repeatedly roll back each other such that the system is rolled back all the way to the initial state

Domino Effect: An Illustration

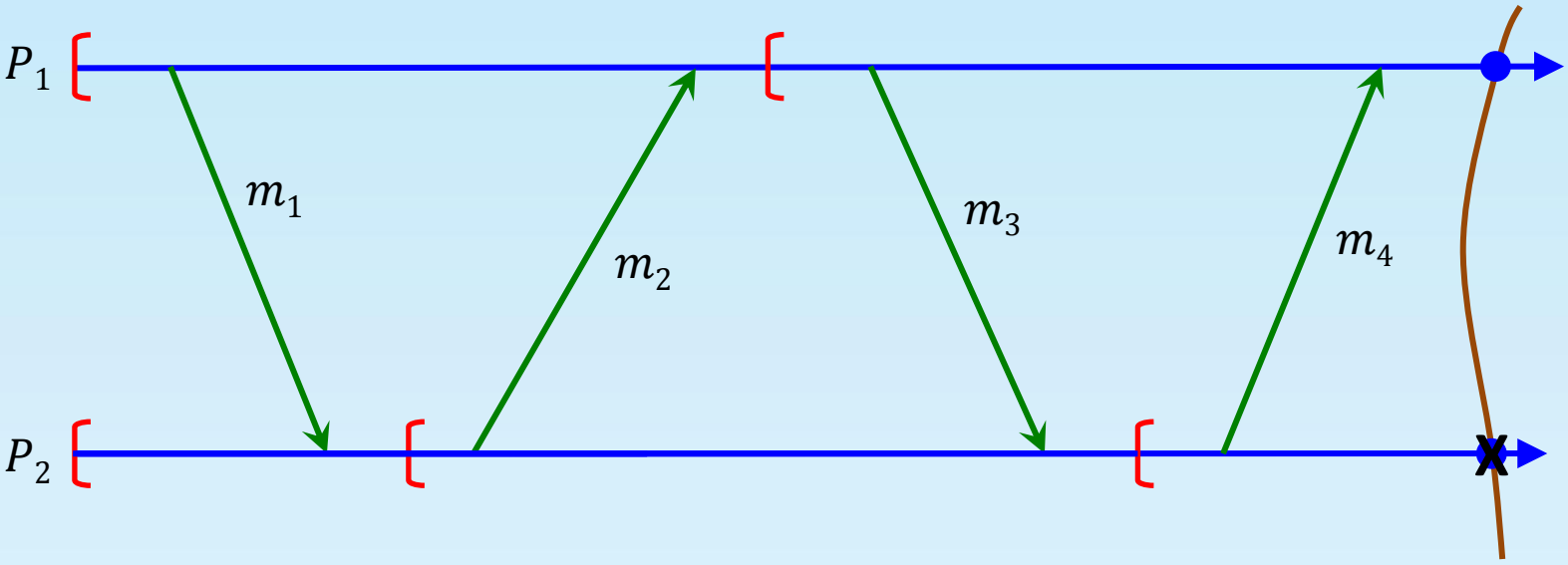


Domino Effect: An Illustration



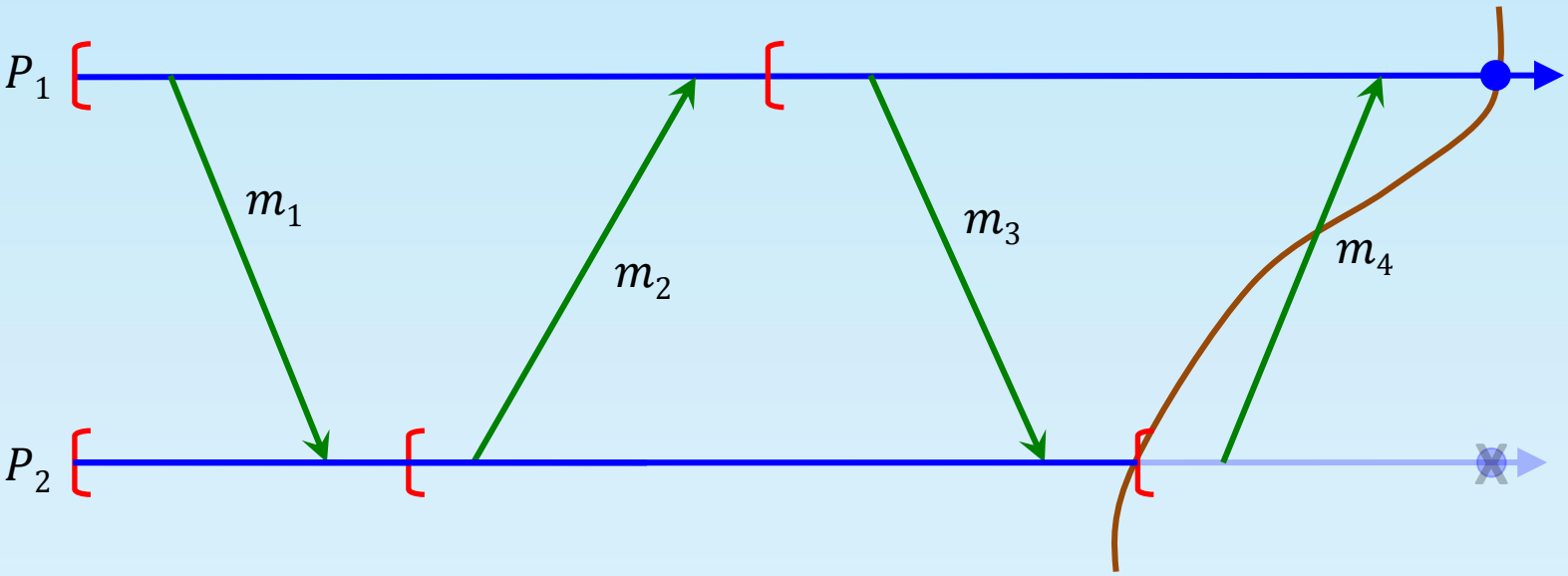
System state before failure

Domino Effect: An Illustration



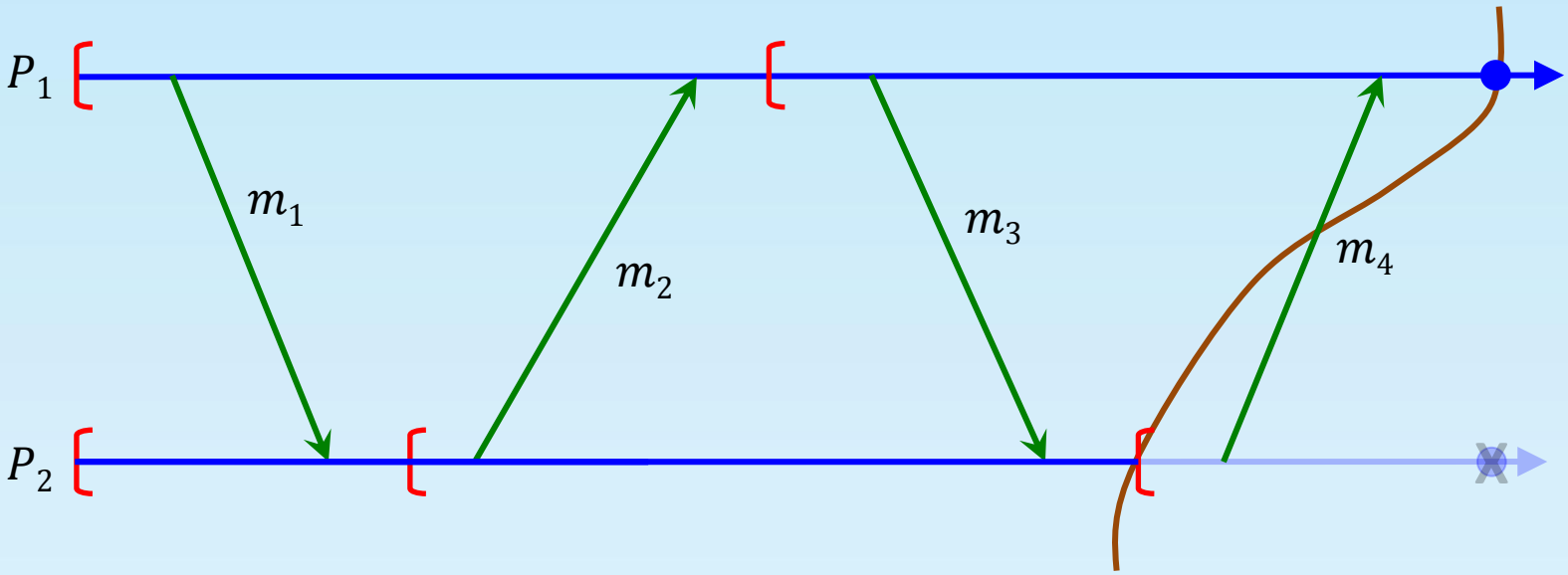
P_2 fails

Domino Effect: An Illustration



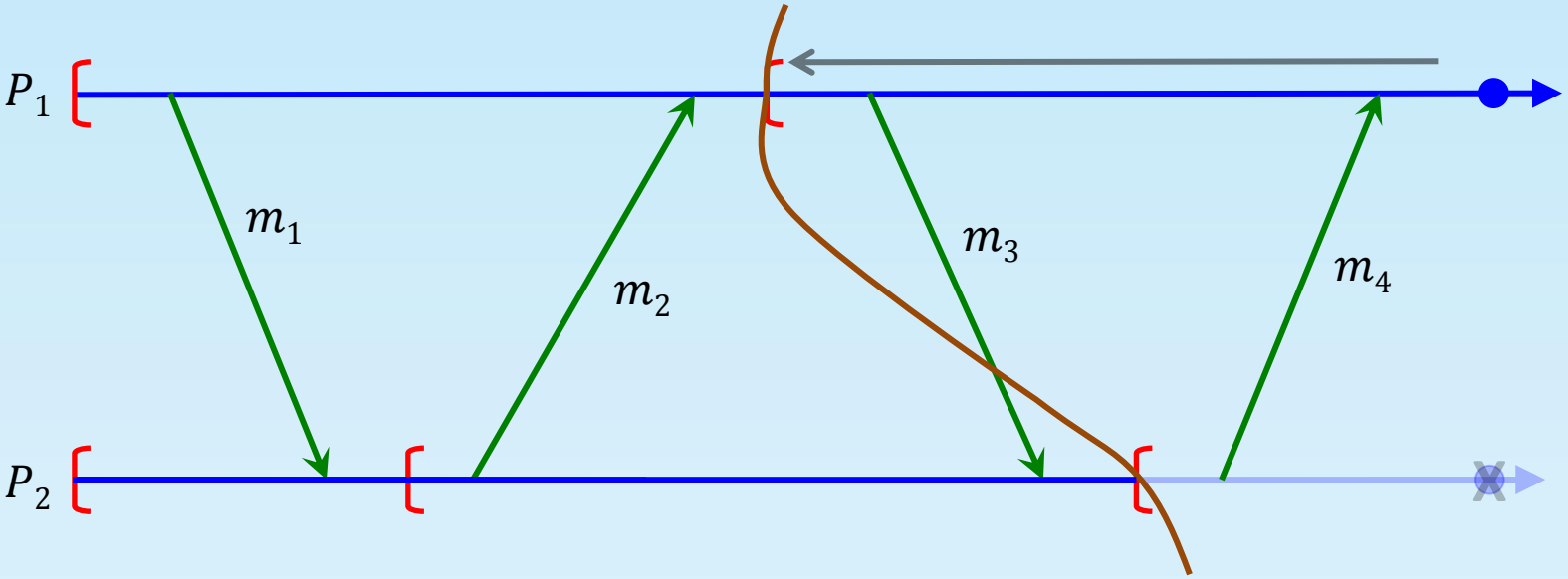
System state just after P_2 restarts

Domino Effect: An Illustration



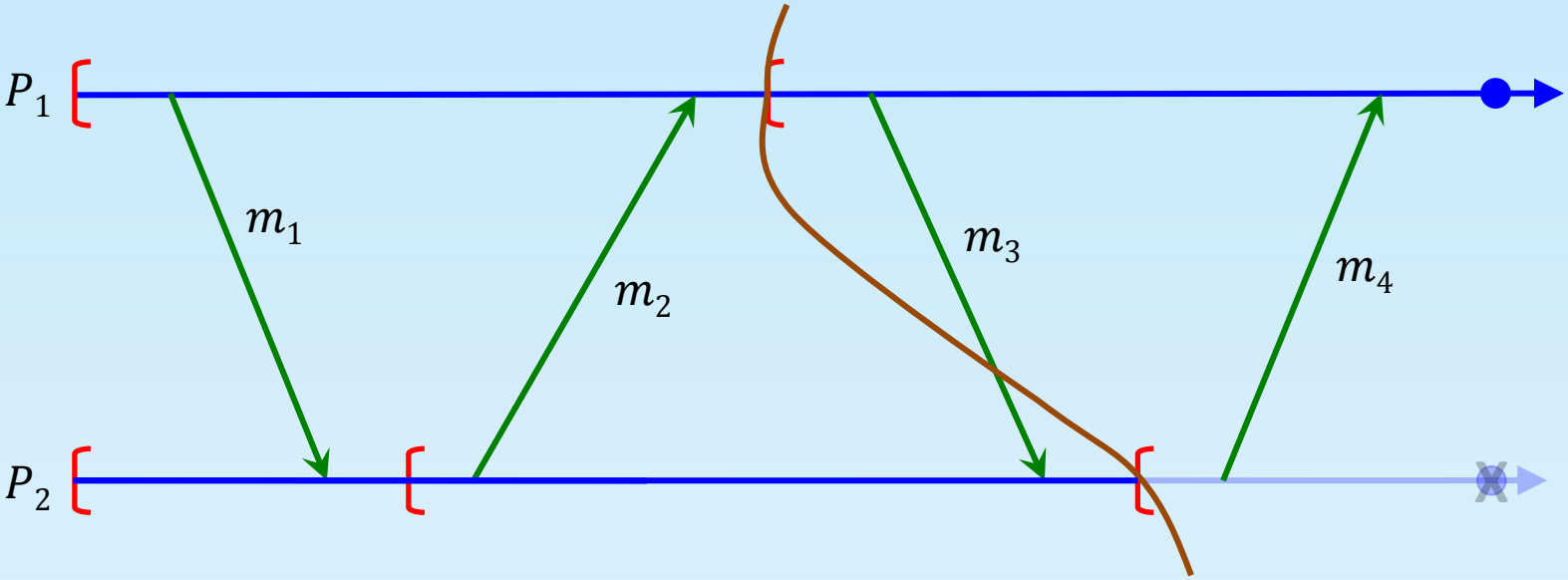
m_4 becomes an orphan message

Domino Effect: An Illustration



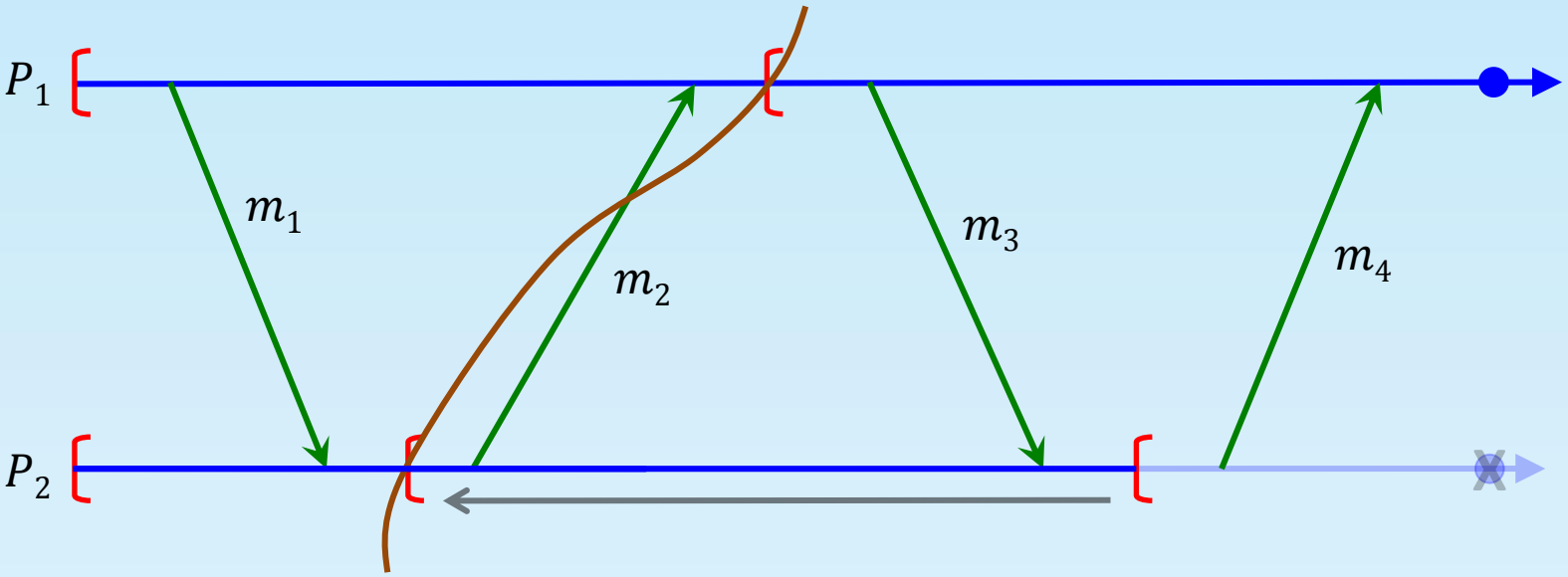
P_1 has to roll back to its previous checkpoint

Domino Effect: An Illustration



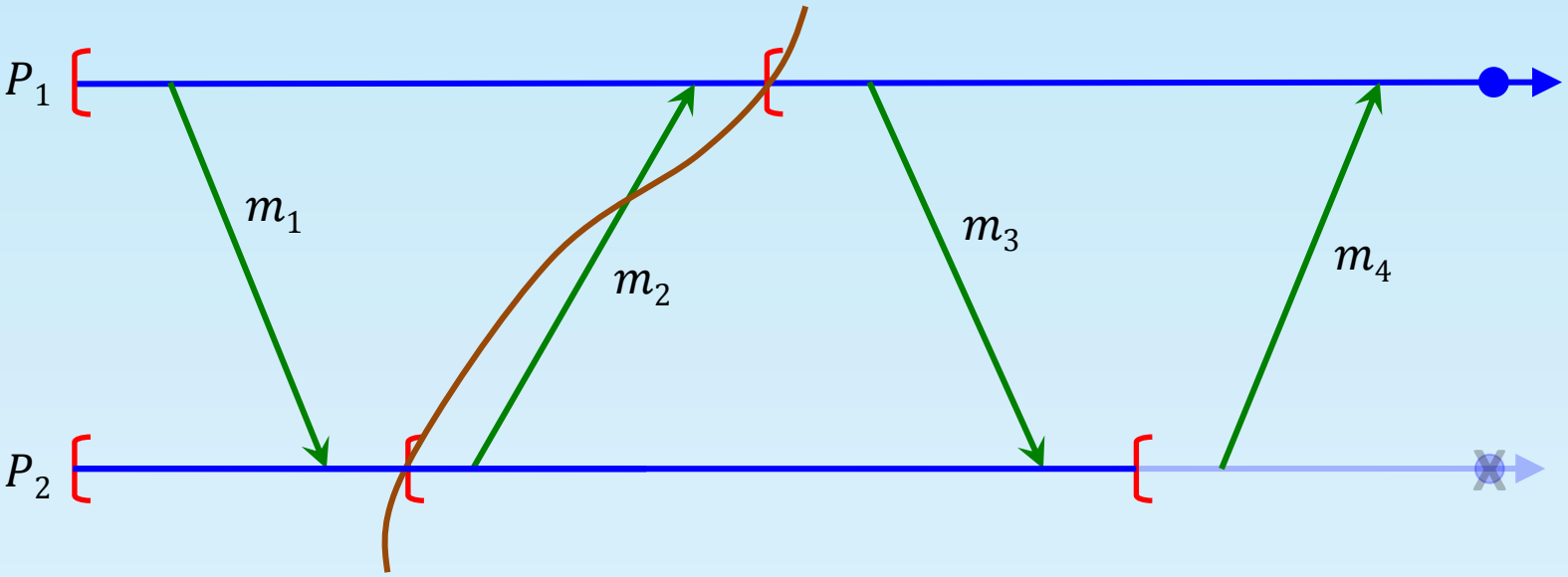
Now m_3 becomes an orphan message

Domino Effect: An Illustration



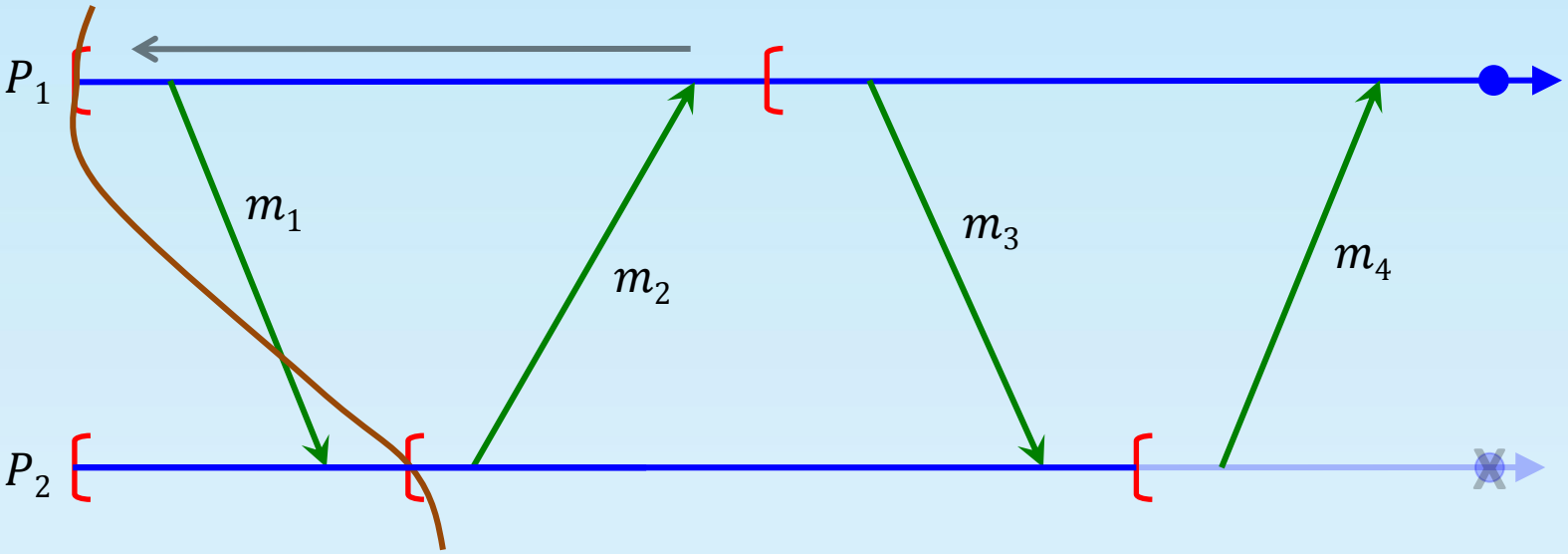
P_2 has to roll back to its previous checkpoint

Domino Effect: An Illustration



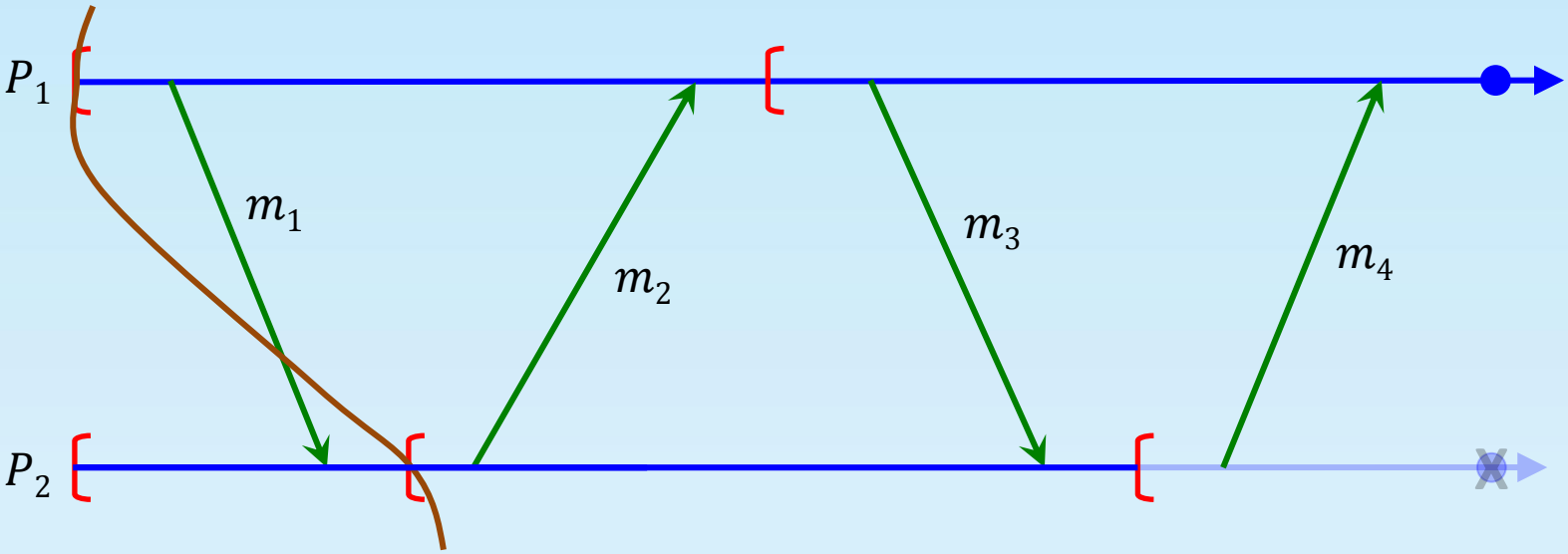
Now m_2 becomes an orphan message

Domino Effect: An Illustration



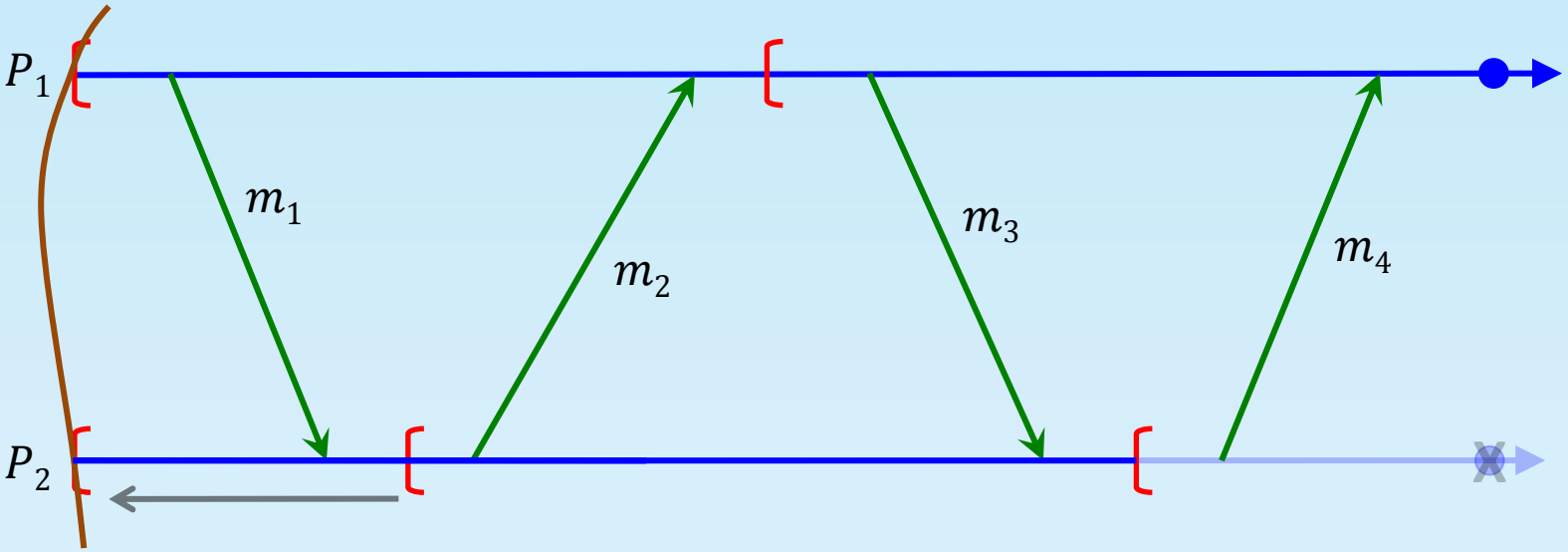
P_1 has to roll back to its previous checkpoint

Domino Effect: An Illustration



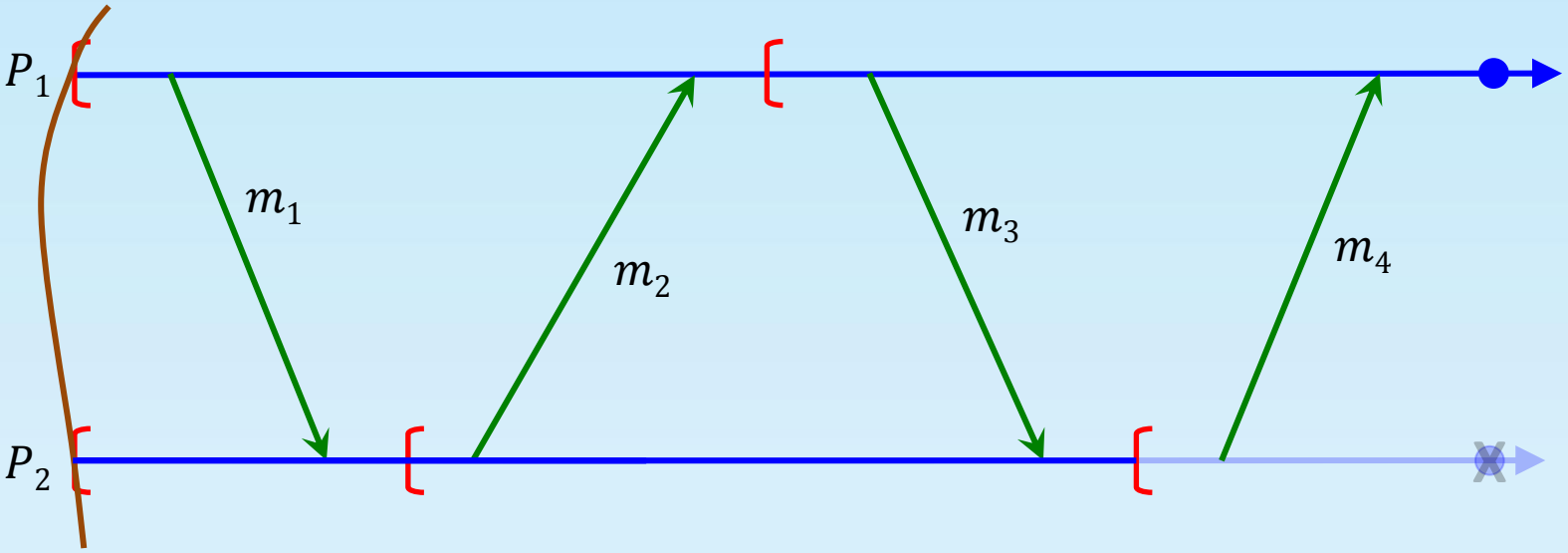
Now m_1 becomes an orphan message

Domino Effect: An Illustration



P_2 has to roll back to its previous checkpoint

Domino Effect: An Illustration



System has rolled back all the way to the beginning

Outline

- Overview
- Main Issues
- Checkpointing and Recovery Protocols
  - Koo and Toueg's Protocol
  - Juang and Venkatesan's Protocol

Checkpointing and Recovery Protocols

- Different protocols provide different trade off between checkpointing overhead and recovery overhead
- Koo and Toueg's Protocol
  - Checkpointing overhead: high
  - Recovery overhead: medium
- Juang and Venkatesan's Protocol
  - Checkpointing overhead: low
  - Recovery overhead: high

Assumptions

- All channels are FIFO
- All channels are bidirectional
- All channels are reliable
- Communication topology need not be a complete graph

Outline

- Overview
- Main Issues
- Checkpointing and Recovery Protocols
  - Koo and Toueg's Protocol
  - Juang and Venkatesan's Protocol

Koo and Toueg's Protocol

- Ensures that the last checkpoints of any processes are concurrent

  As a result:

  - No process has to roll back beyond its last checkpoint
  - Checkpointing by one process may cause other processes to take a checkpoint as well
  - Recovery involves rolling back processes to their last checkpoints

Koo and Toueg's Protocol (Contd.)

- At any given time, multiple instances of checkpointing and recovery protocols may be in progress:
  - A process participates in at most instance (checkpointing or recovery) at any given time
    - It will refuse to participate in other instances, thereby causing them to abort
    - Aborted instances are restarted later by their initiators

Koo and Toueg's Checkpointing Protocol

- Consists of two phases
  - First Phase:
    - Processes take tentative checkpoints if they can
      - A process after taking a tentative checkpoint cannot send any application messages until the second phase completes
  - Second Phase:
    - If all required processes are able to take checkpoints in the first phase, then tentative checkpoints are made permanent
    - Otherwise, tentative checkpoints are discarded

Koo and Toueg's Checkpointing Protocol (Contd.)

- Minimizes the number of processes that take checkpoints
  - Each process assigns labels with monotonically increasing value to its messages
  - $\perp$ is a special label:
    - It is smaller than any other label value
  - Each process maintains two vectors with one entry for each of its neighbors:
    - $last\_label\_rcvd$
    - $first\_label\_sent$

Koo and Toueg's Checkpointing Protocol: Details

- Consider processes P_i and P_j that are neighbors
- Definition of $last\_label\_rcvdi[j]$:
  - Let m be the last message that P_i has received from P_j since its last permanent/tentative checkpoint

$$last\_label\_rcvdi[j] \triangleq \begin{cases} \text{label of } m, & m \text{ exists} \\ \perp, & m \text{ does not exist} \end{cases}$$

Koo and Toueg's Checkpointing Protocol: Details (Contd.)

- Definition of $first\_label\_senti[j]$:
  - Let m be the first message that P_i has sent to P_j since its last permanent/tentative checkpoint

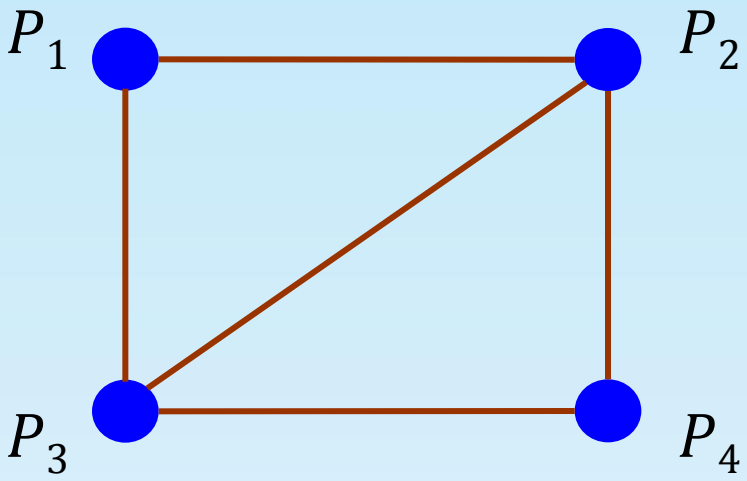
$$first\_label\_sent_i[j] \triangleq \begin{cases} \text{label of } m, & m \text{ exists} \\ \perp, & m \text{ does not exist} \end{cases}$$

Koo and Toueg's Checkpointing Protocol: Details (Contd.)

- Assume that P_i has taken a (tentative) checkpoint:
  - P_j does not need to take a checkpoint if $last\_label\_rcvdi[j] = \perp$
  - Otherwise, P_i requests P_j to take a checkpoint and sends $last\_label\_rcvdi[j]$ to P_j
  - P_j takes a (tentative) checkpoint if:

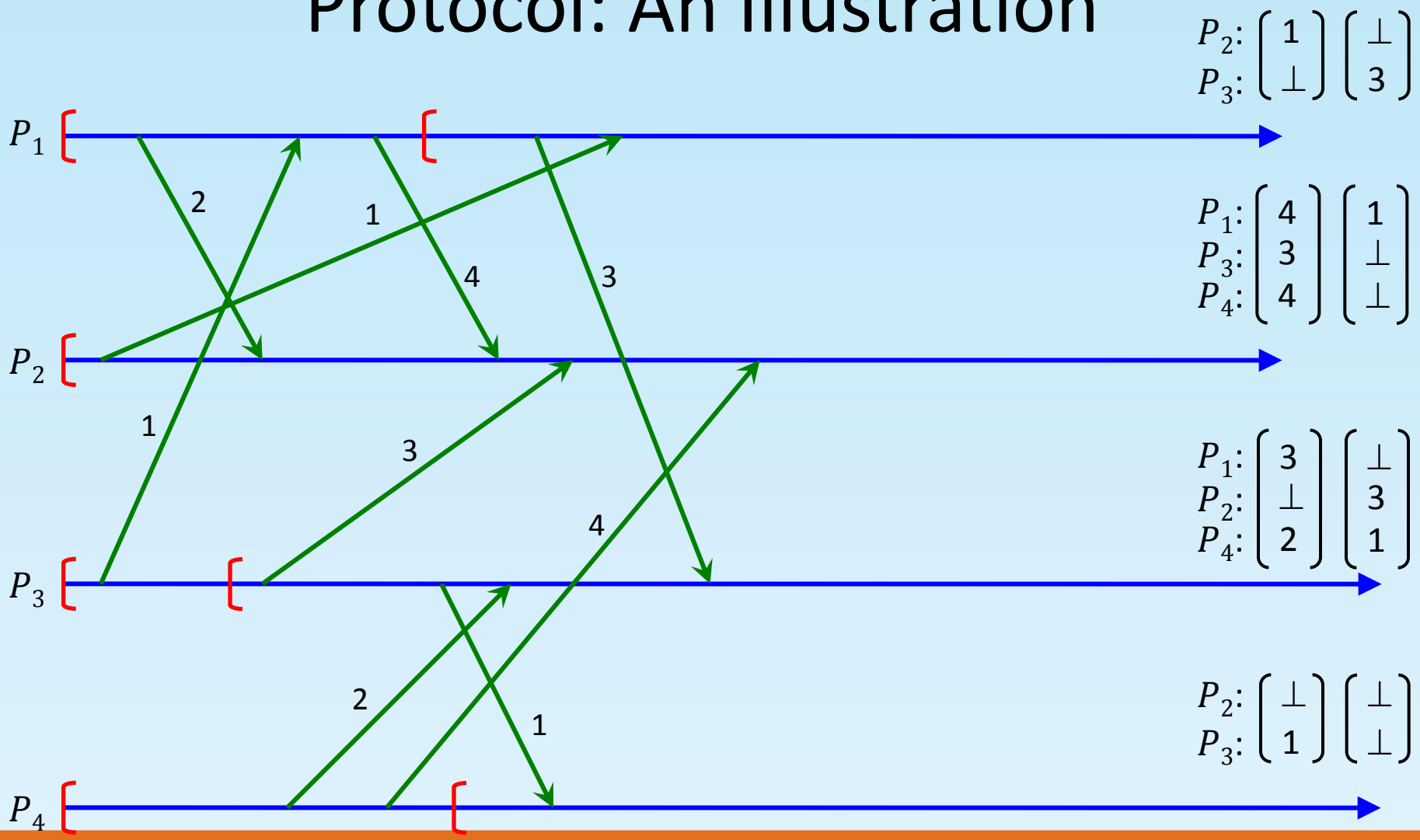
    $last\_label\_rcvdi[j] \geq first\_label\_sentj[i] > \perp$

Koo and Toueg's Checkpointing Protocol: An Illustration

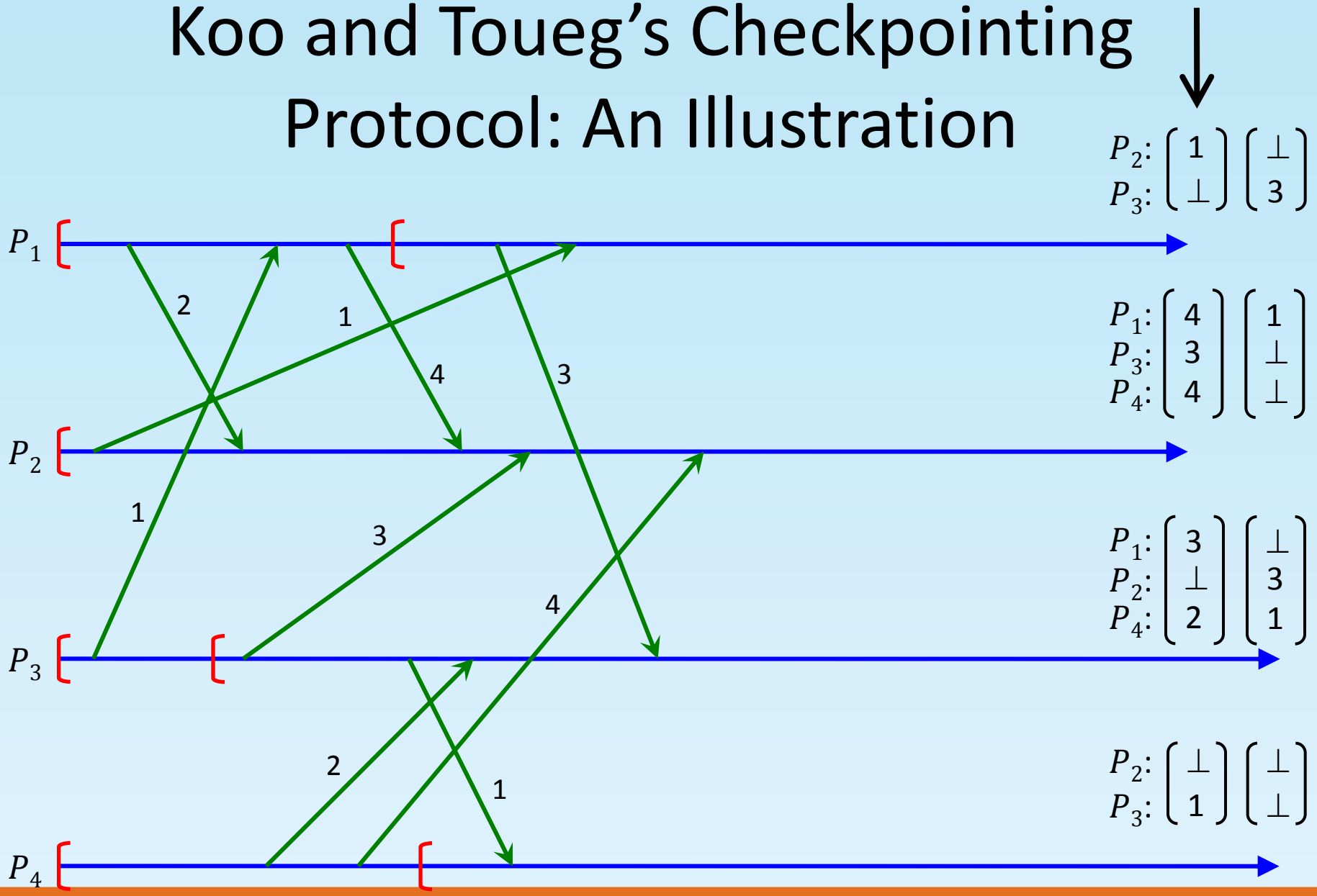


Communication topology (used in all illustrations)

Koo and Toueg's Checkpointing Protocol: An Illustration

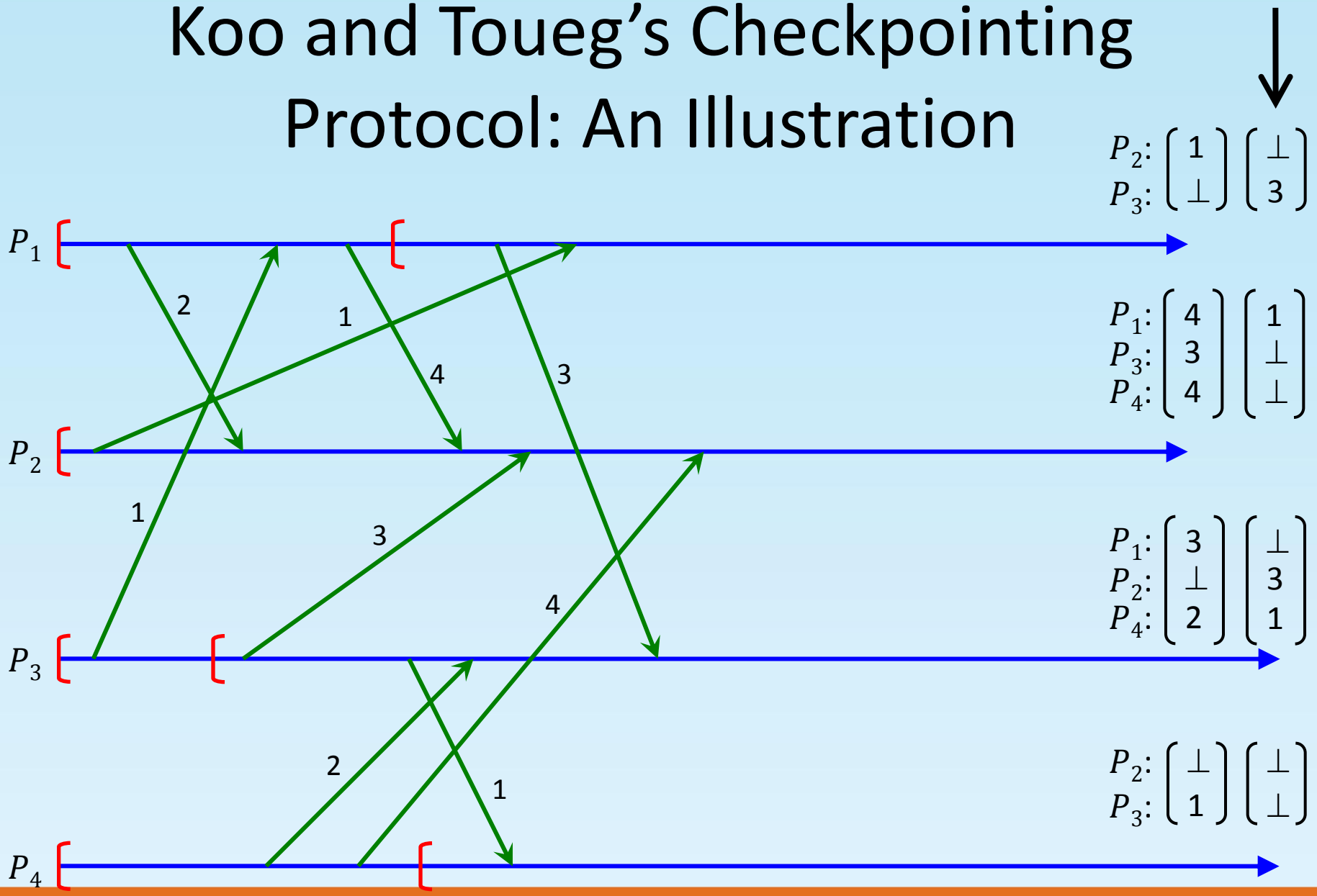


Koo and Toueg's Checkpointing Protocol: An Illustration



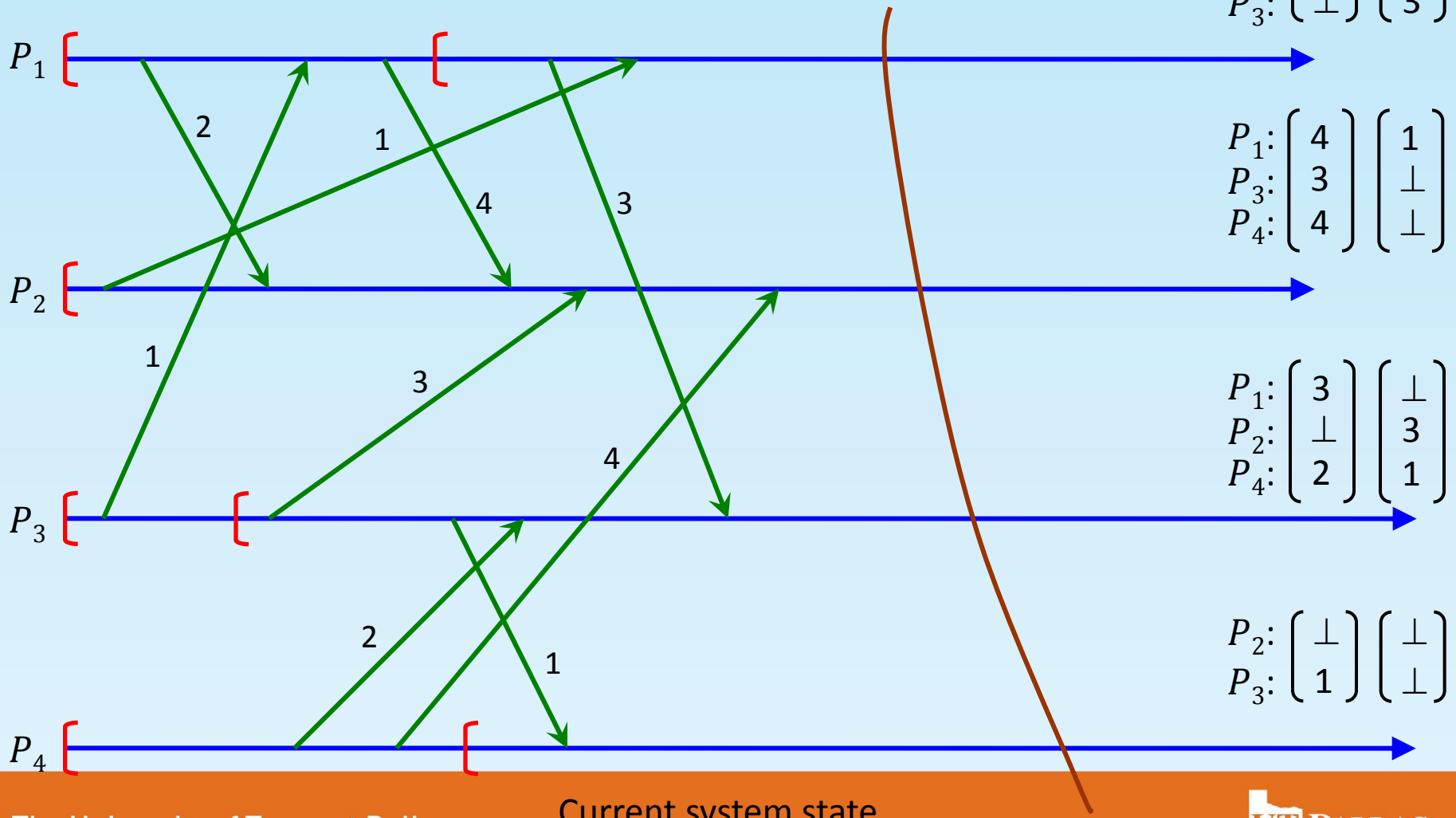
First vector: $last\_label\_rcvd$

Koo and Toueg's Checkpointing Protocol: An Illustration

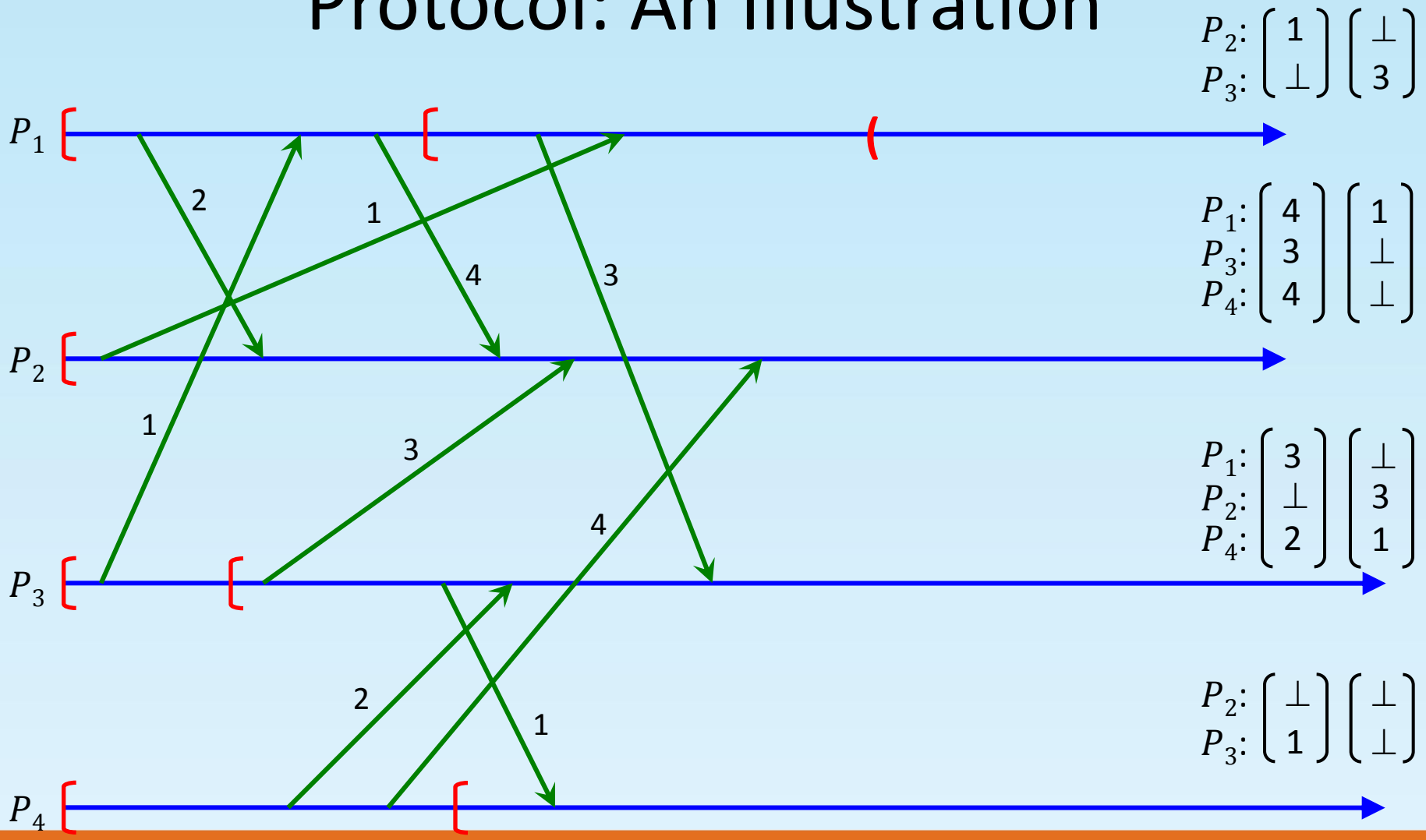


Second vector: $first\_label\_sent$

Koo and Toueg's Checkpointing Protocol: An Illustration

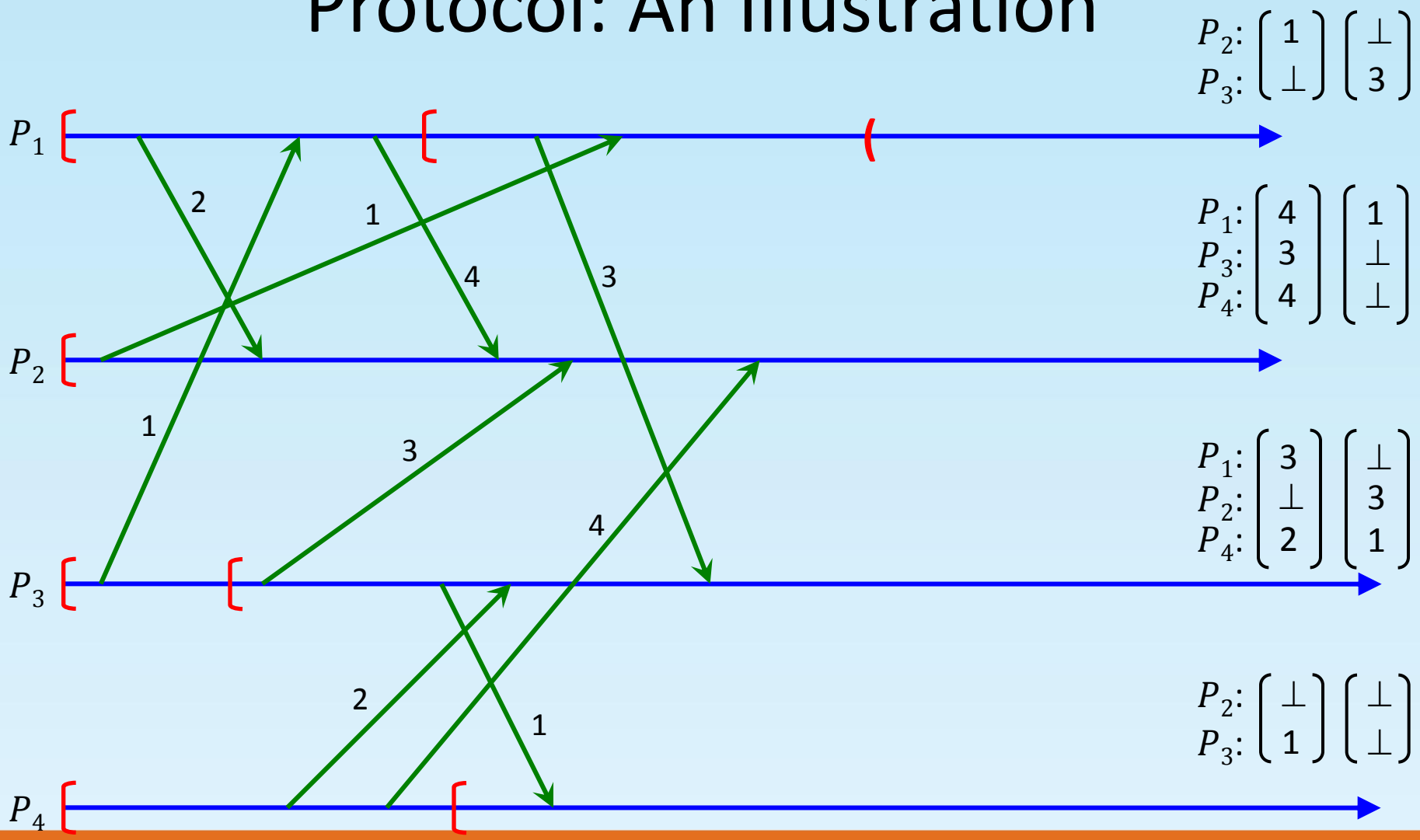


Koo and Toueg's Checkpointing Protocol: An Illustration



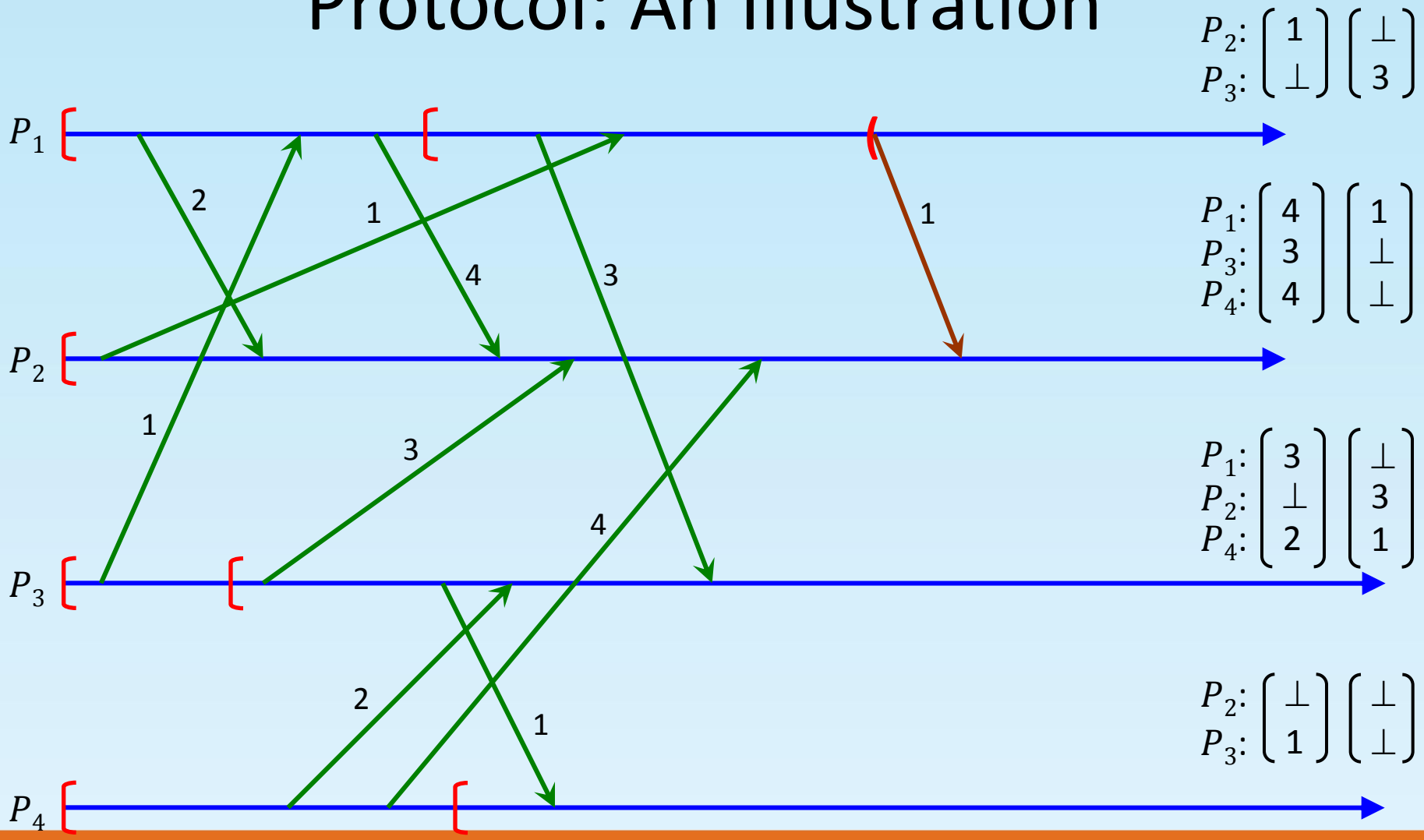
P_1 initiates checkpointing protocol by taking a tentative checkpoint

Koo and Toueg's Checkpointing Protocol: An Illustration



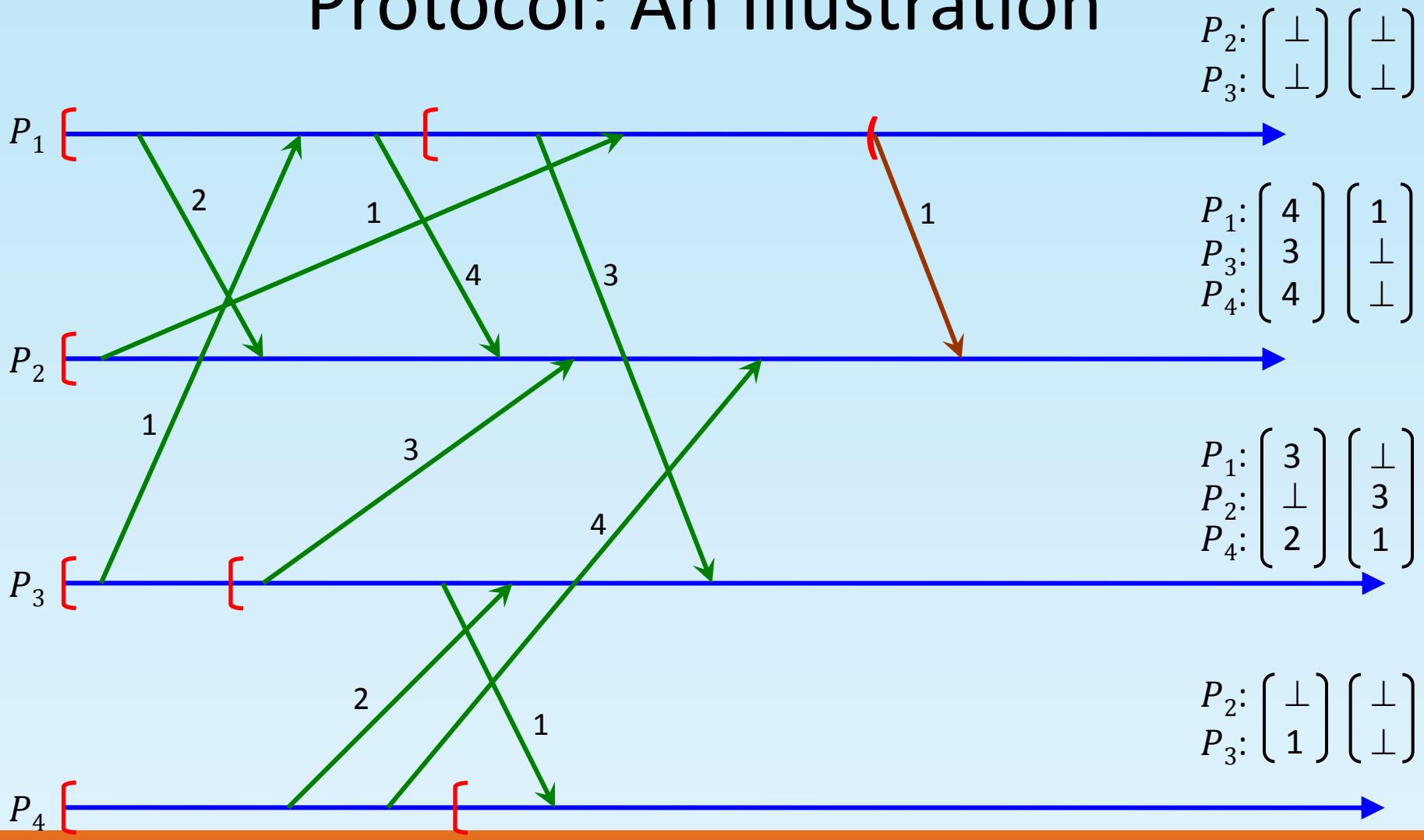
Checkpoint cohort set of P_1 is given by $\{P_2\}$

Koo and Toueg's Checkpointing Protocol: An Illustration



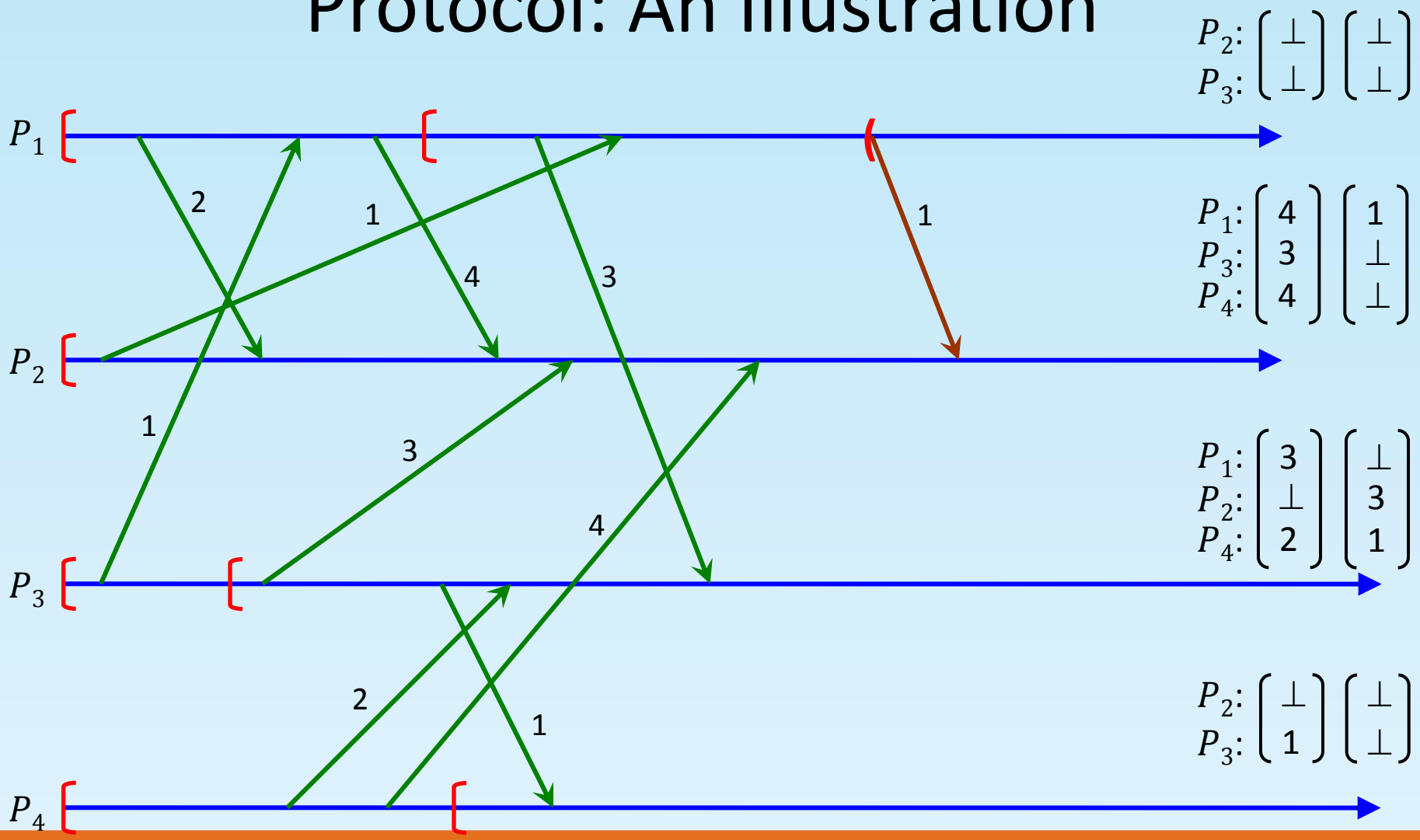
P_1 sends the value of $last\_label\_rcvd_1[2] = 1$ to P_2

Koo and Toueg's Checkpointing Protocol: An Illustration



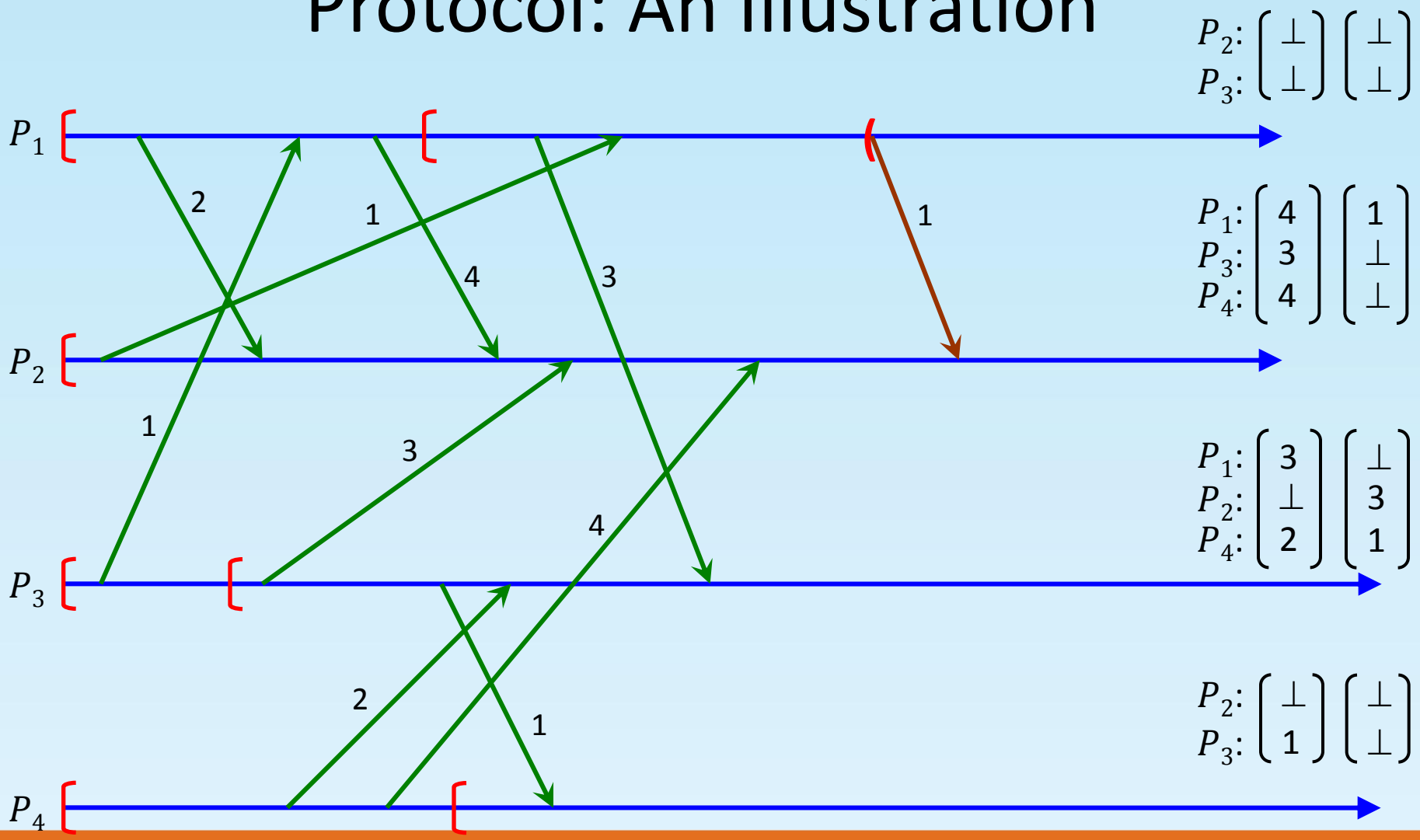
P_1 resets all entries in $last\_label\_rcvd_1$ and $first\_label\_sent_1$ vectors

Koo and Toueg's Checkpointing Protocol: An Illustration



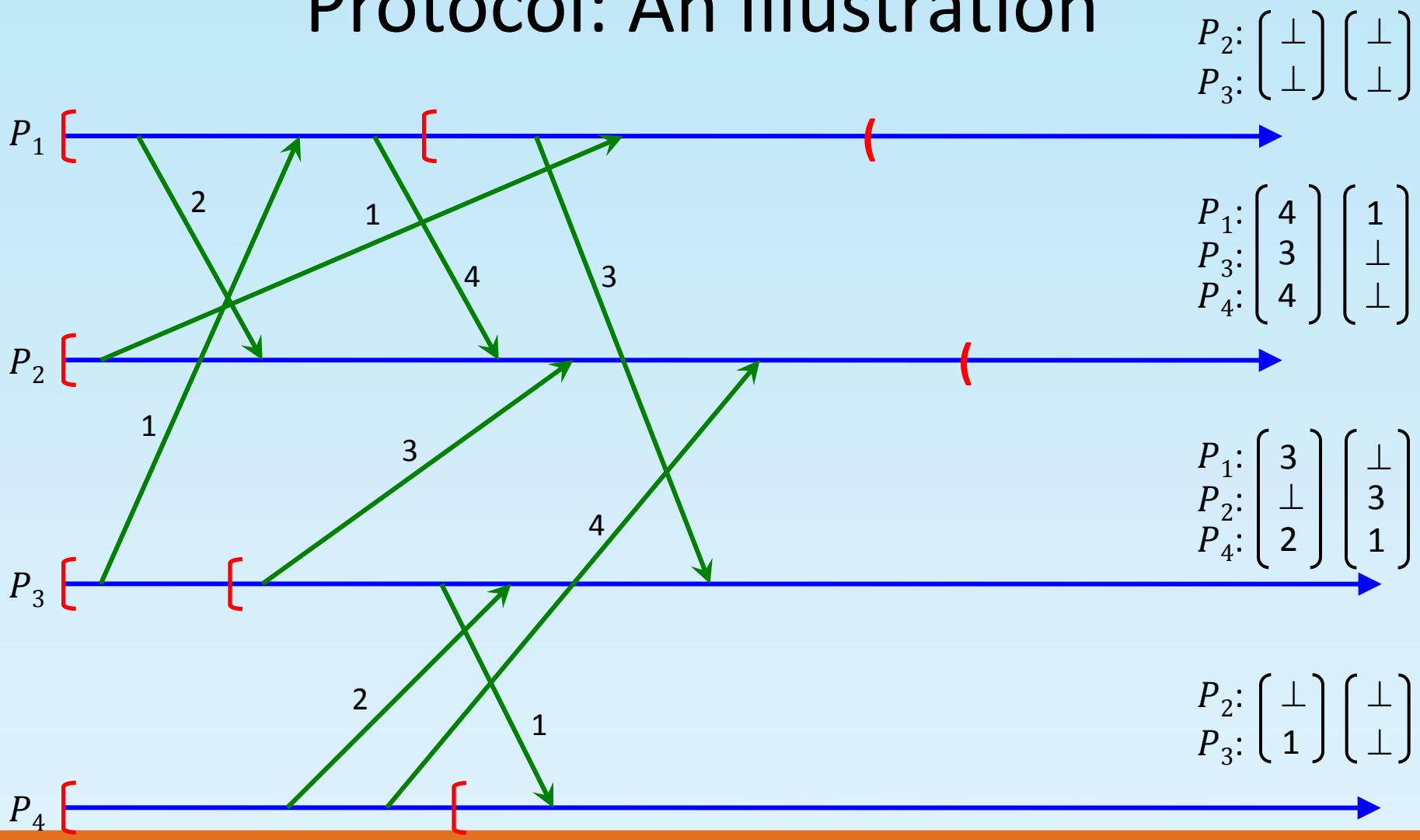
P_2 receives the message from P_1 and processes it

Koo and Toueg's Checkpointing Protocol: An Illustration



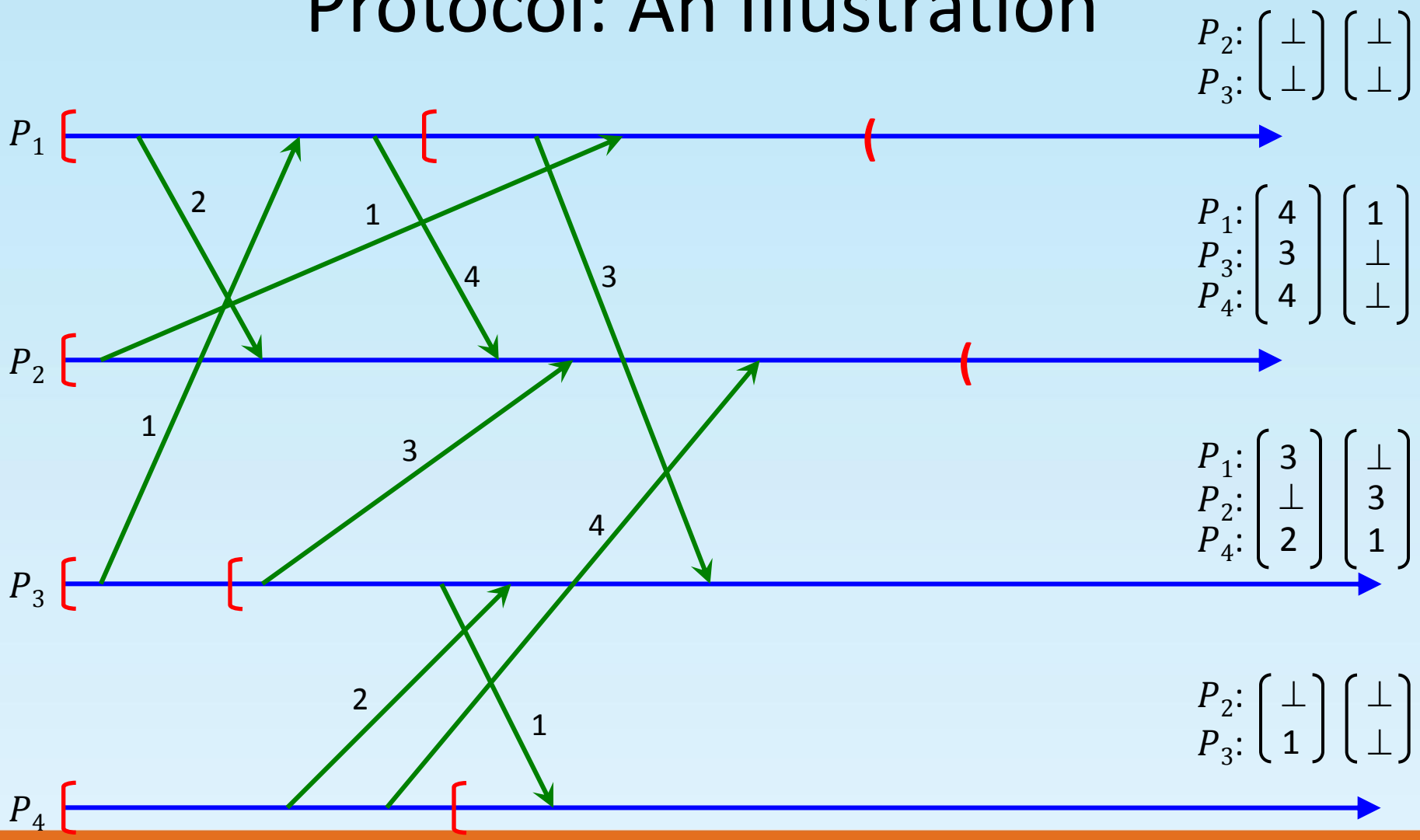
P_2 tests whether $last\_label\_rcvd_1[2] \geq first\_label\_sent_2[1] > \perp$ holds

Koo and Toueg's Checkpointing Protocol: An Illustration



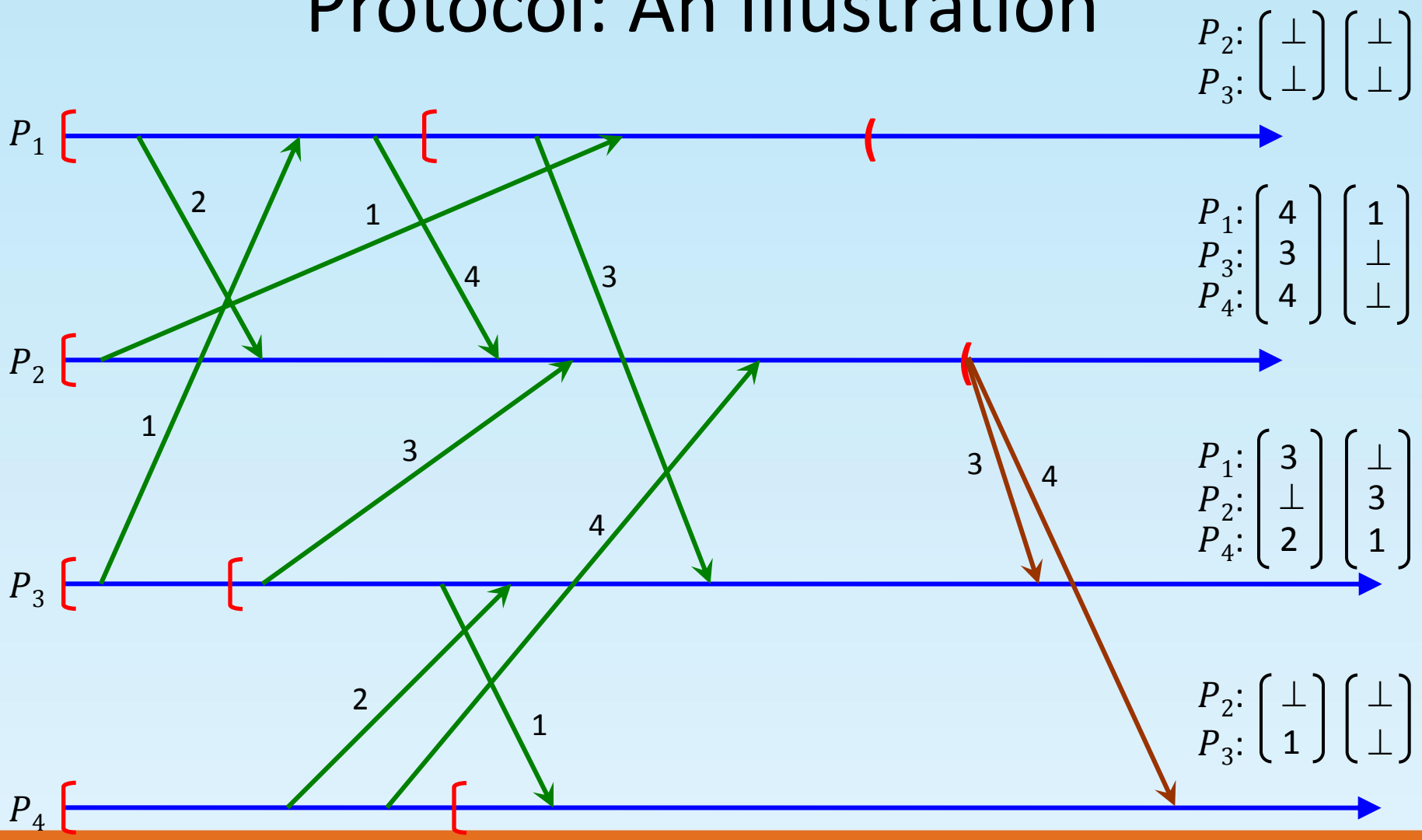
The test evaluates to true and P_2 takes a tentative checkpoint

Koo and Toueg's Checkpointing Protocol: An Illustration



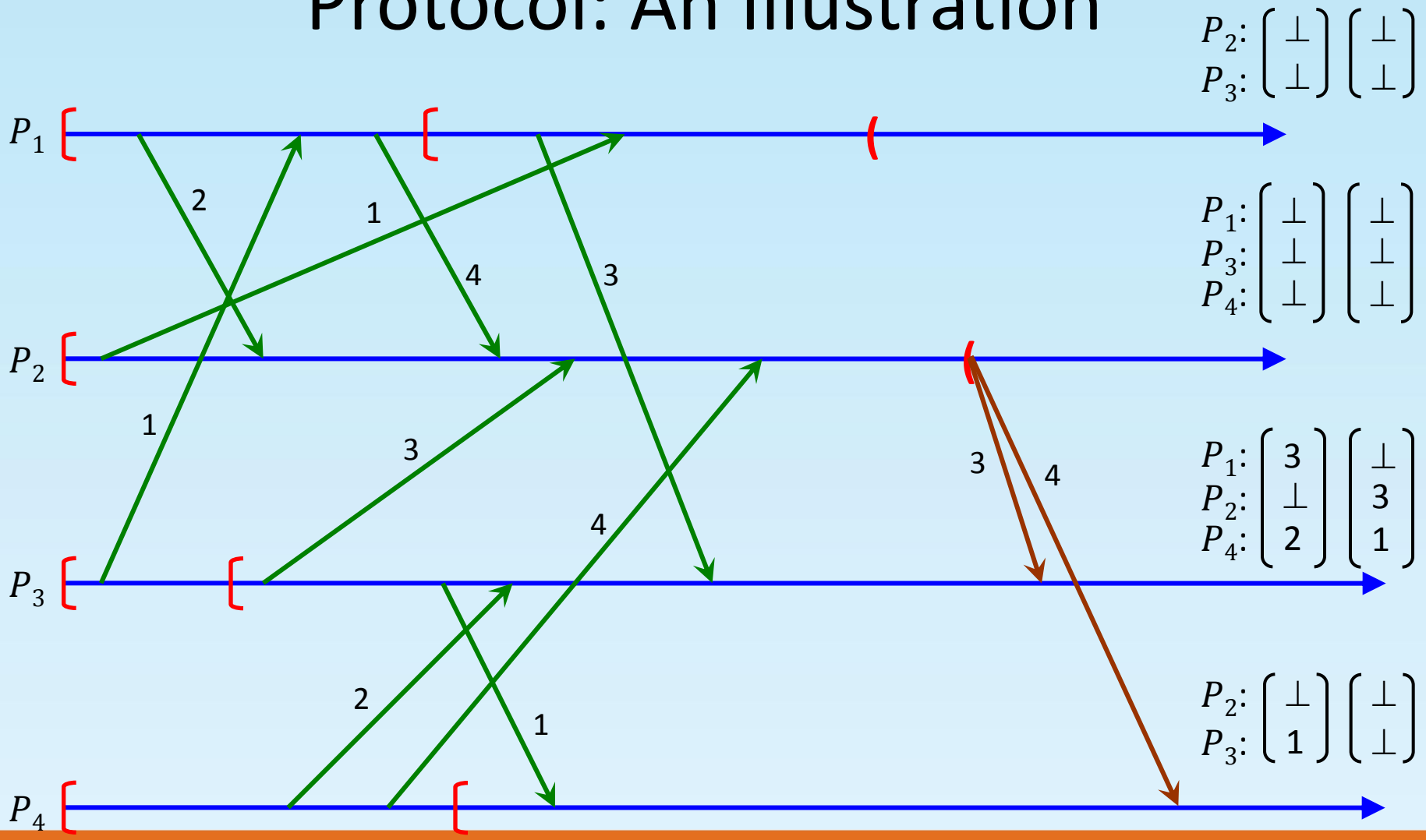
Checkpoint cohort set of P_2 is given by $\{P_1, P_3, P_4\}$

Koo and Toueg's Checkpointing Protocol: An Illustration



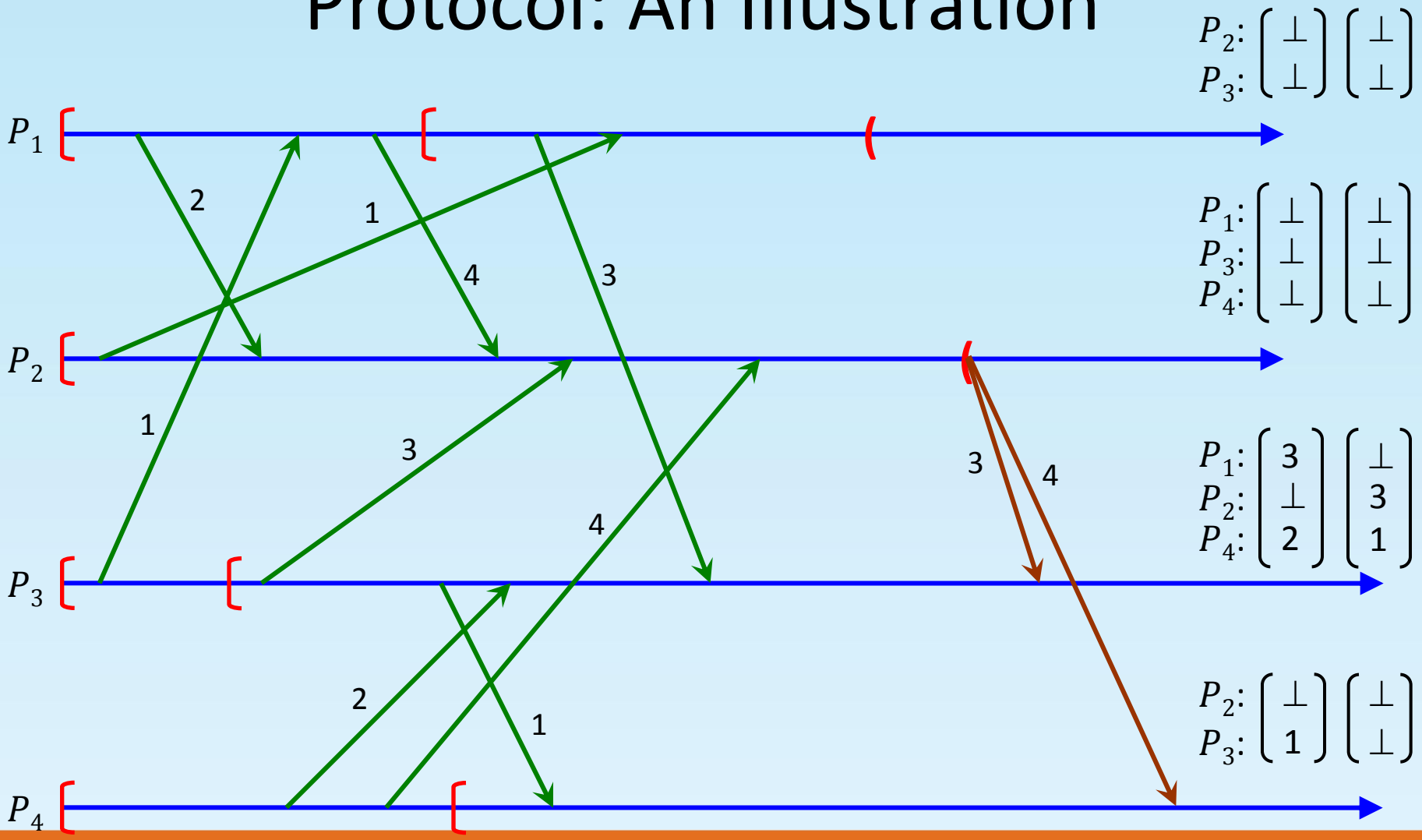
P_2 sends the value of $last\_label\_rcvd_2[3] = 3$ to P_3 and $last\_label\_rcvd_2[4] = 4$ to P_4

Koo and Toueg's Checkpointing Protocol: An Illustration



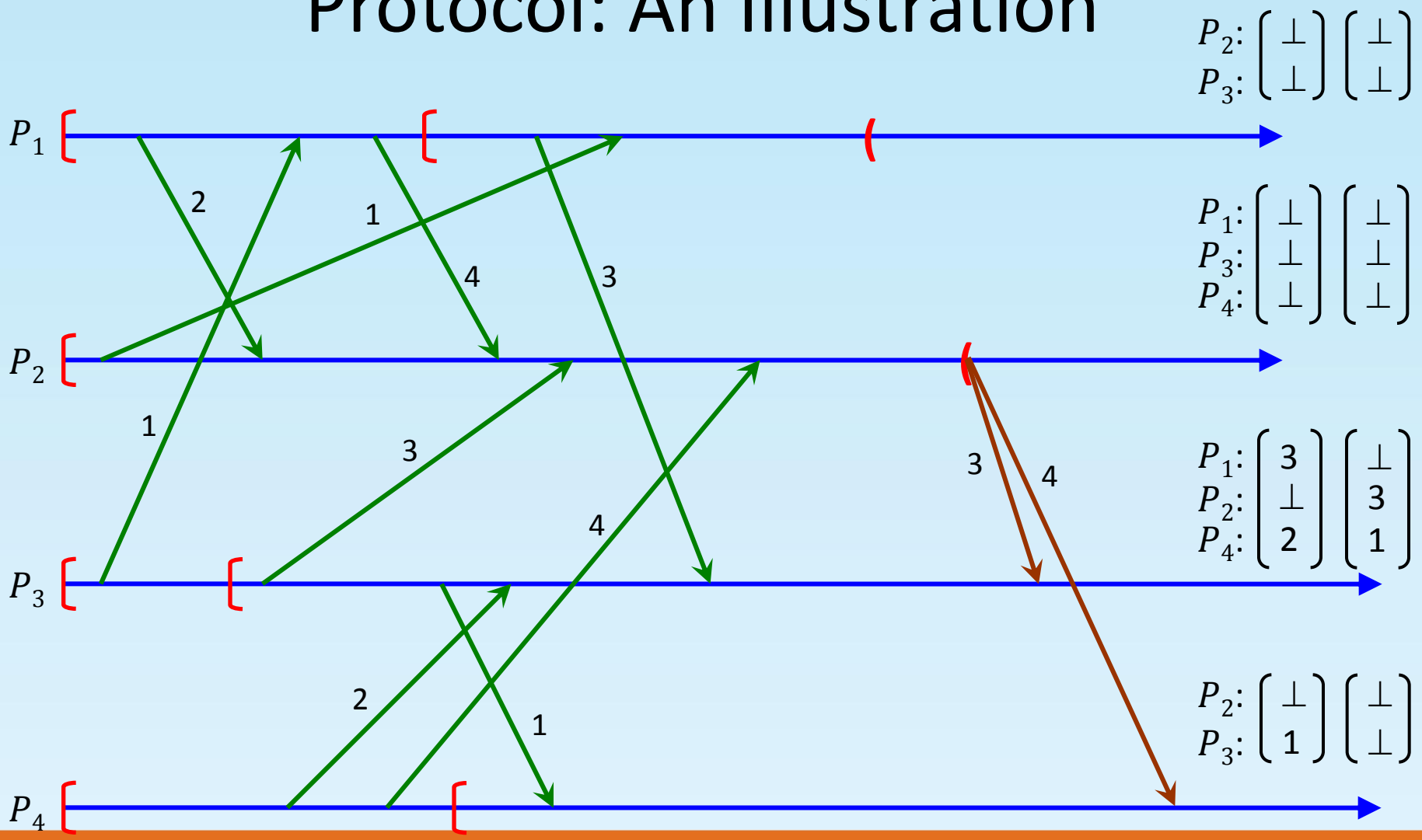
P_2 resets all entries in $last\_label\_rcvd_2$ and $first\_label\_sent_2$ vectors

Koo and Toueg's Checkpointing Protocol: An Illustration



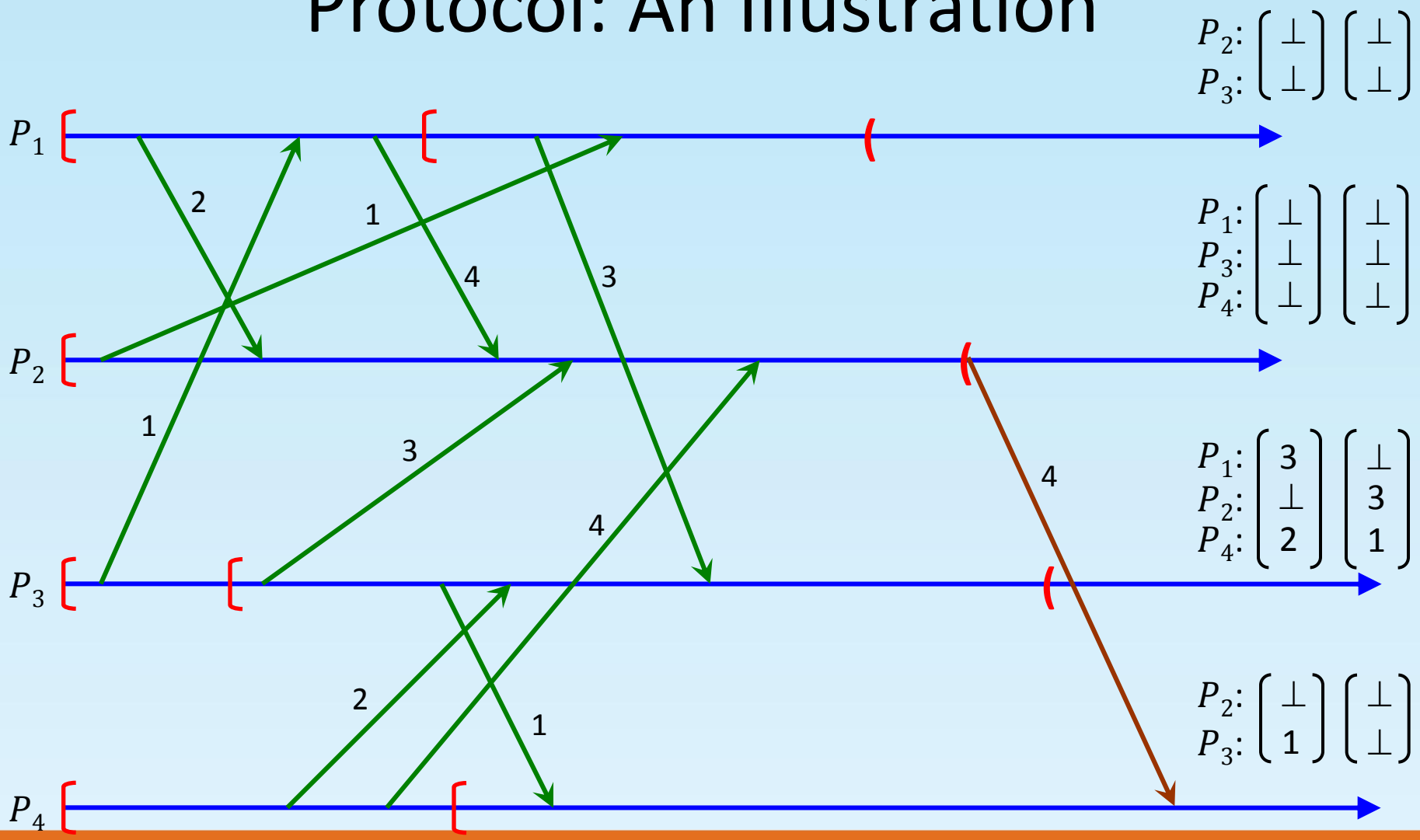
P_3 receives the message from P_2 and processes it

Koo and Toueg's Checkpointing Protocol: An Illustration



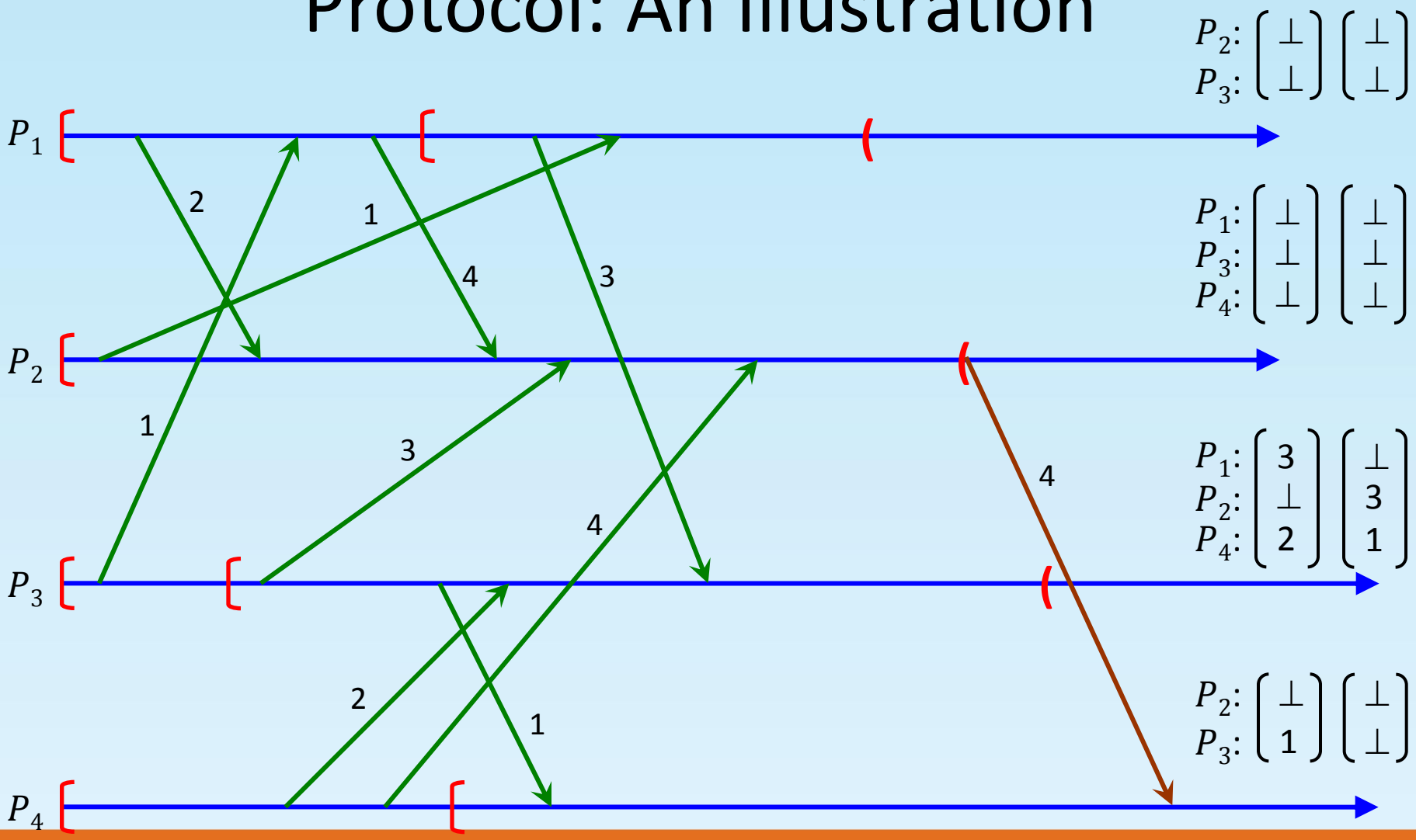
P_3 tests whether $last\_label\_rcvd_2[3] \geq first\_label\_sent_3[2] > \perp$ holds

Koo and Toueg's Checkpointing Protocol: An Illustration



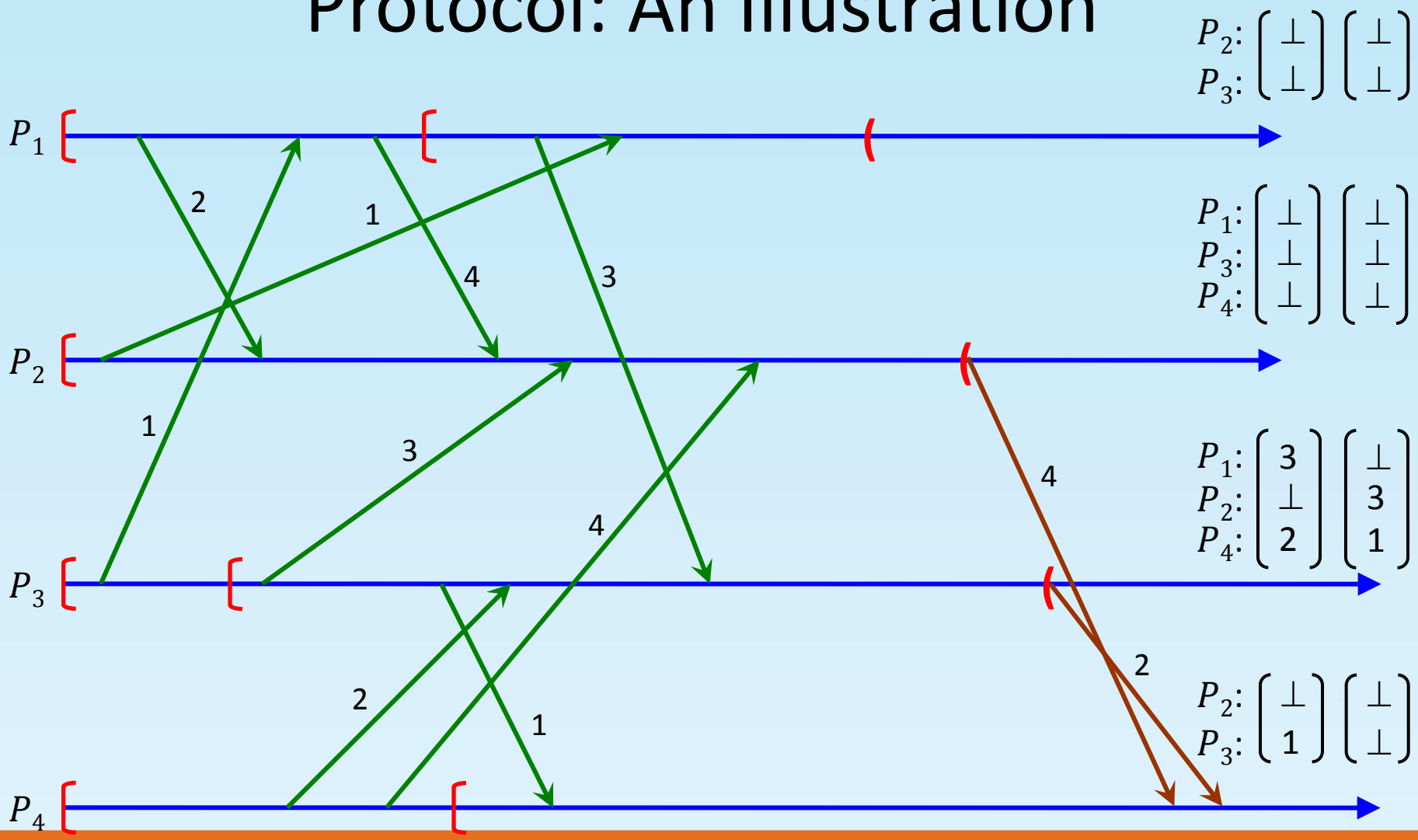
The test evaluates to true and P_3 takes a tentative checkpoint

Koo and Toueg's Checkpointing Protocol: An Illustration



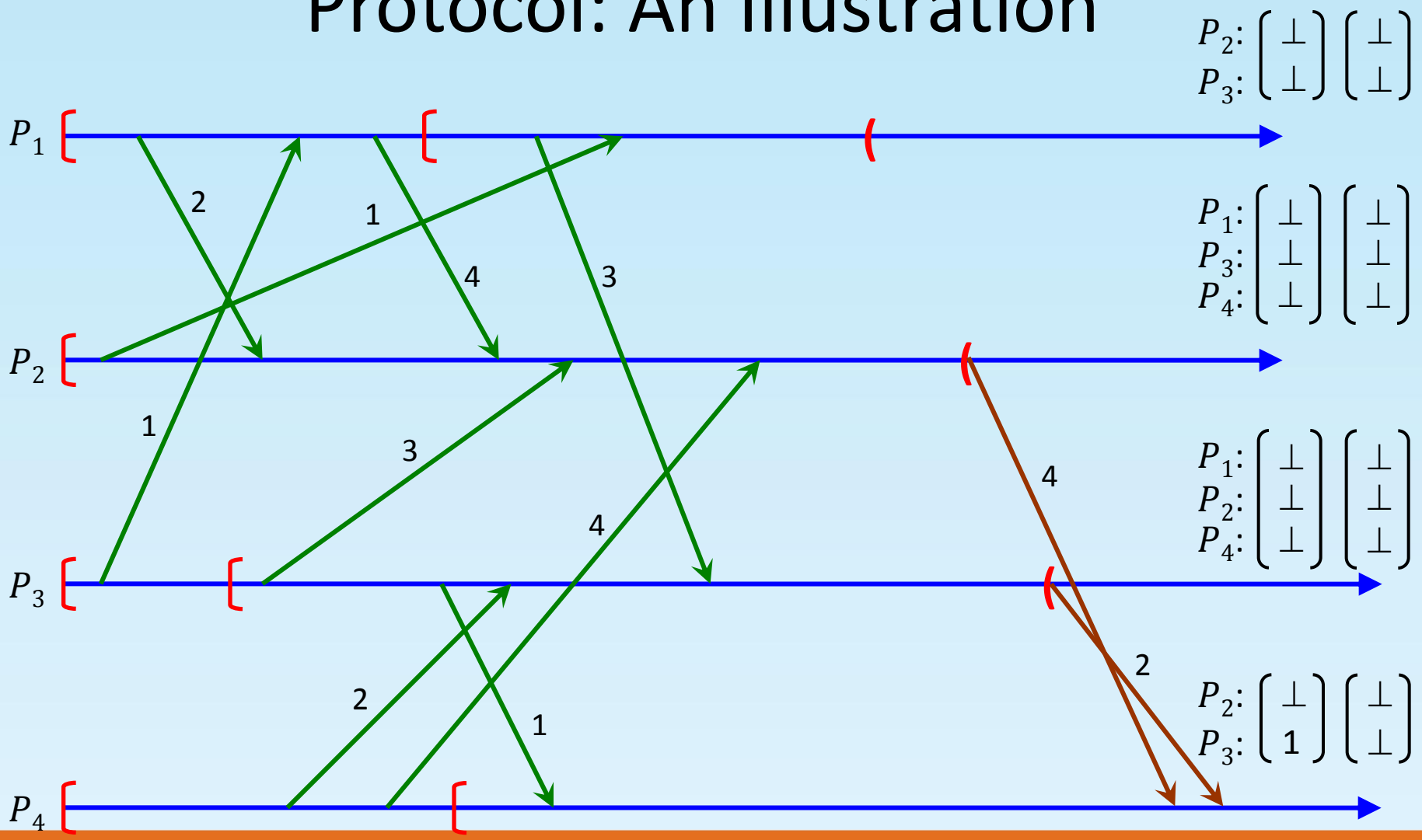
Checkpoint cohort set of P_3 is given by $\{P_1, P_4\}$

Koo and Toueg's Checkpointing Protocol: An Illustration



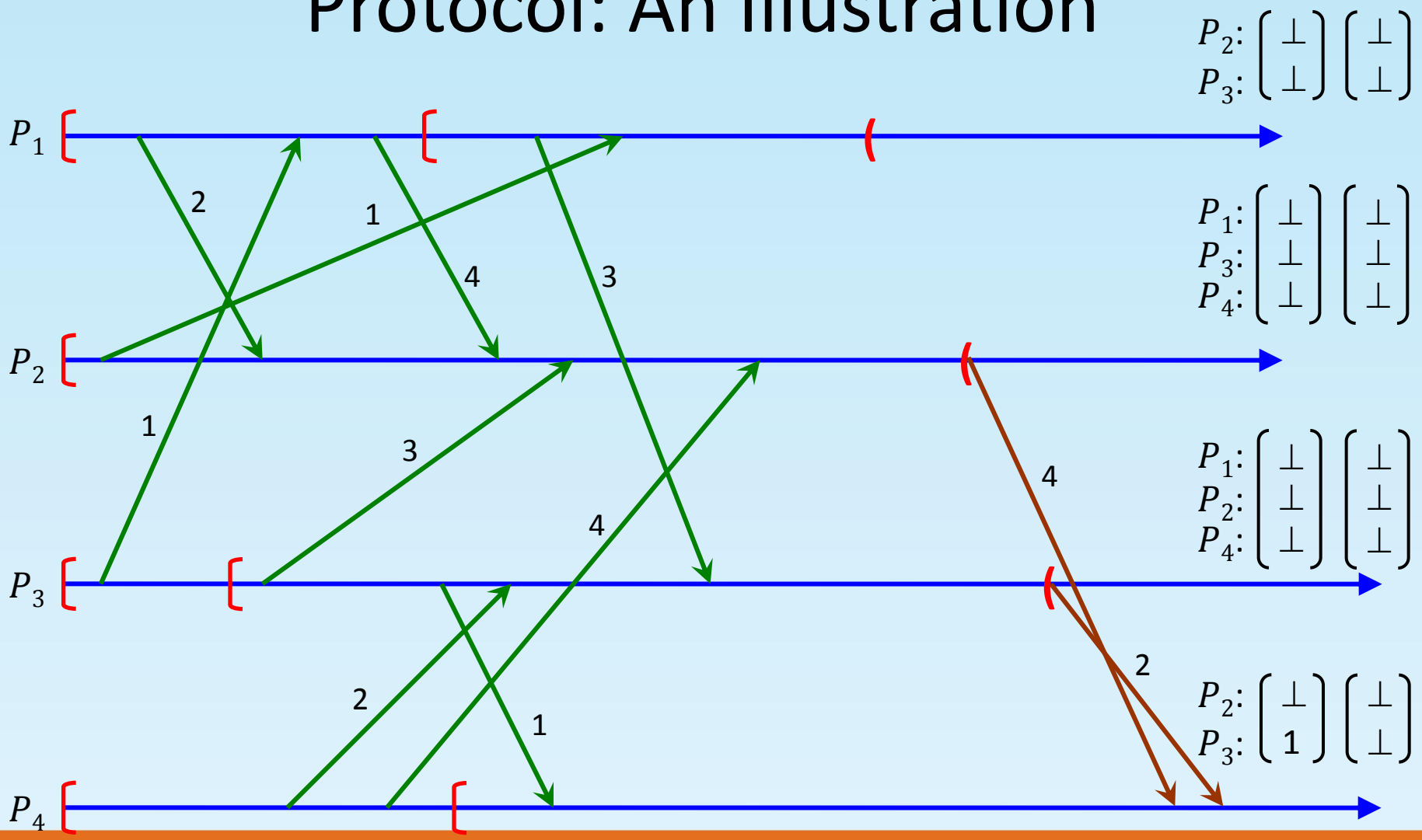
P_3 sends the value of $last\_label\_rcvd_3[4] = 2$ to P_4

Koo and Toueg's Checkpointing Protocol: An Illustration



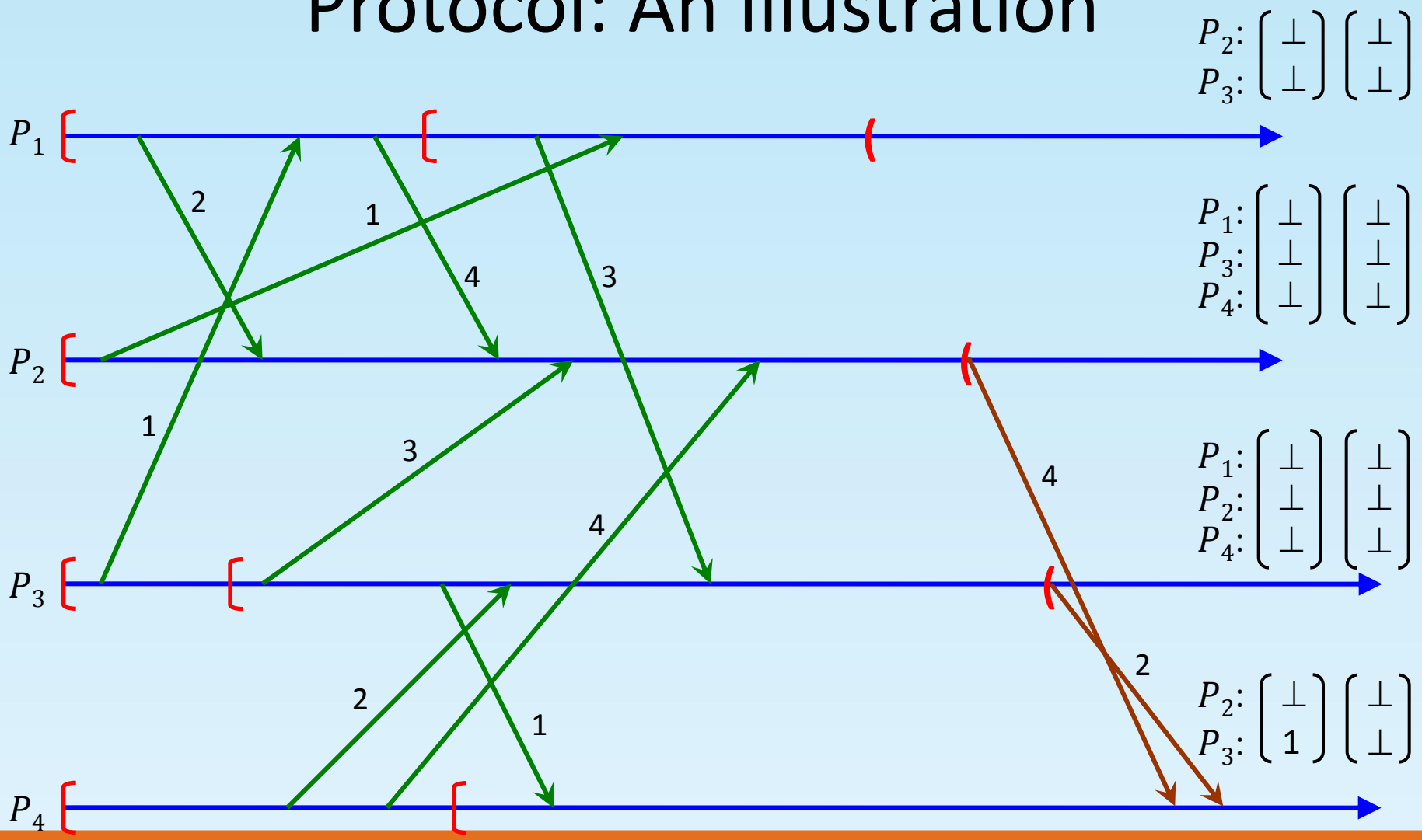
P_3 resets all entries in $last\_label\_rcvd_3$ and $first\_label\_sent_3$ vectors.

Koo and Toueg's Checkpointing Protocol: An Illustration



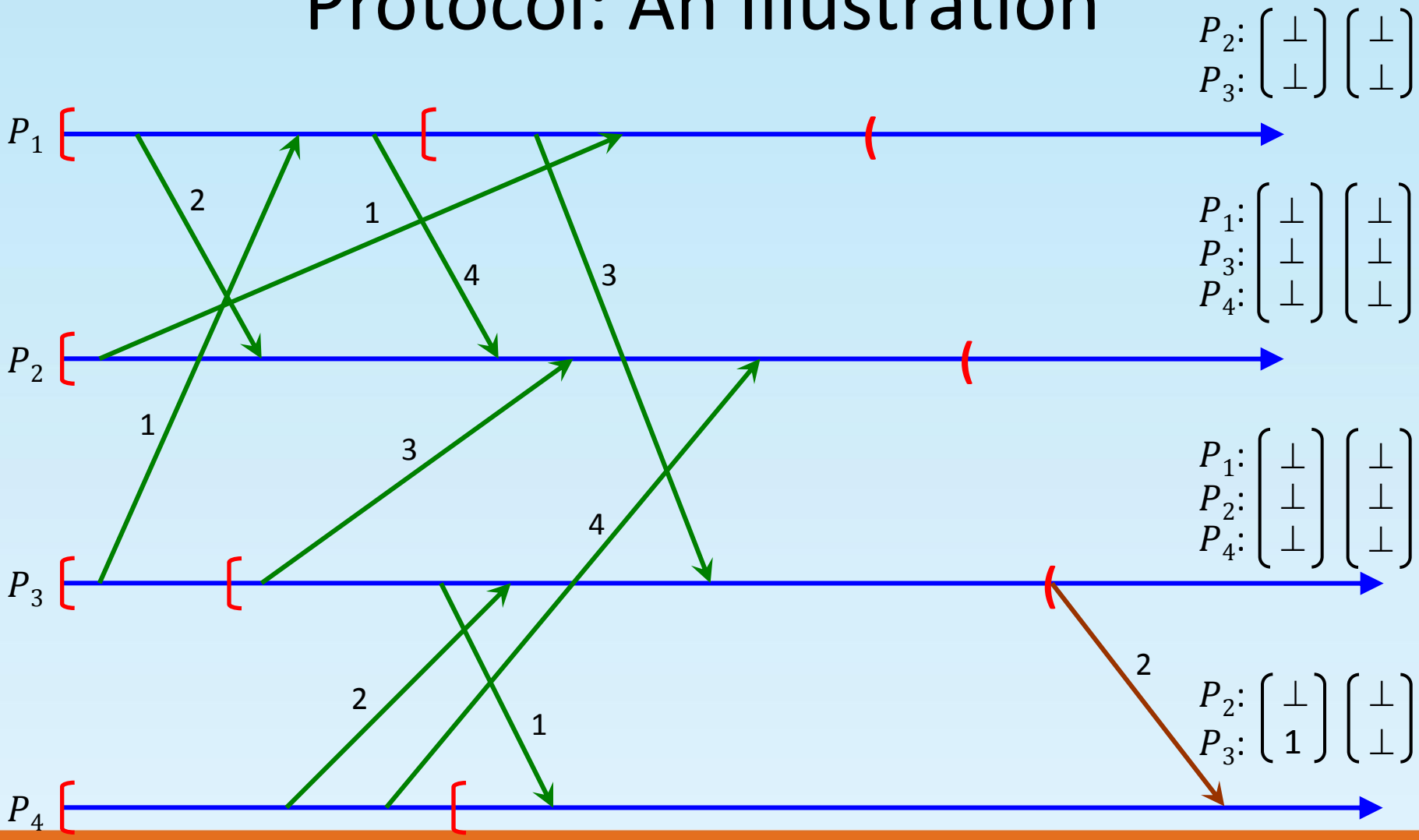
P_4 receives the message from P_2 and processes it

Koo and Toueg's Checkpointing Protocol: An Illustration



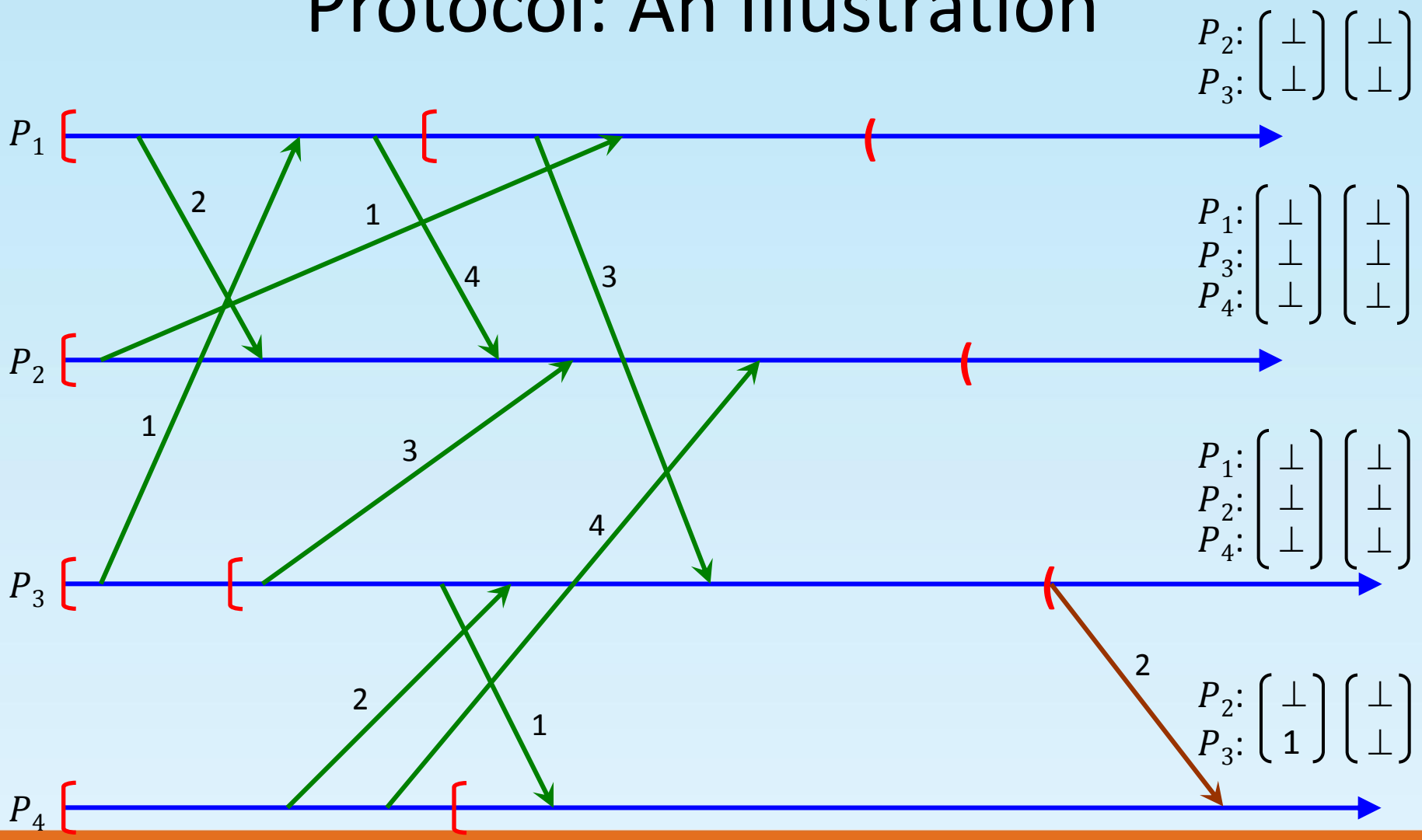
P_4 tests whether $last\_label\_rcvd_2[4] \geq first\_label\_sent_4[2] > \perp$ holds

Koo and Toueg's Checkpointing Protocol: An Illustration



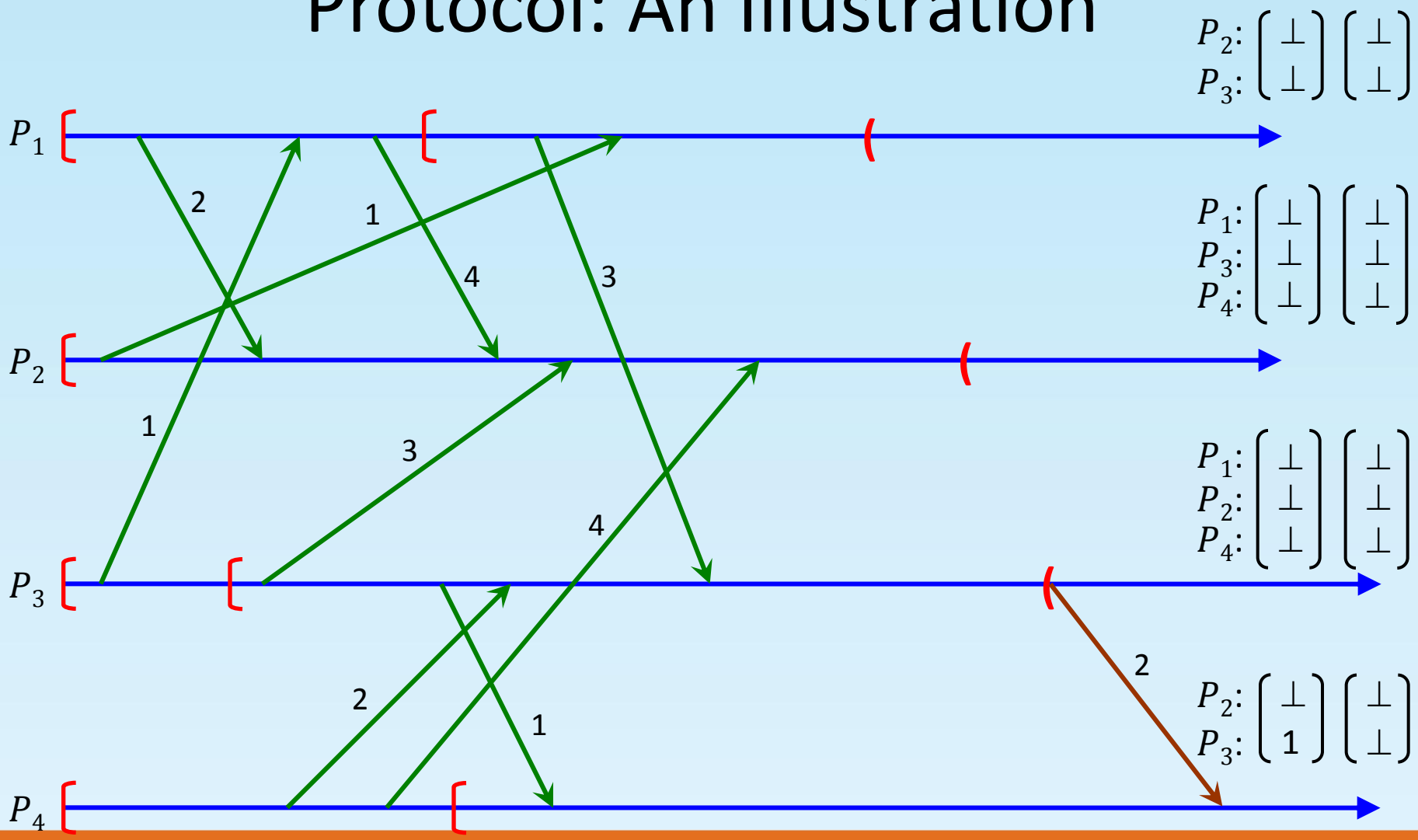
The test fails and P_4 does not take any checkpoint

Koo and Toueg's Checkpointing Protocol: An Illustration



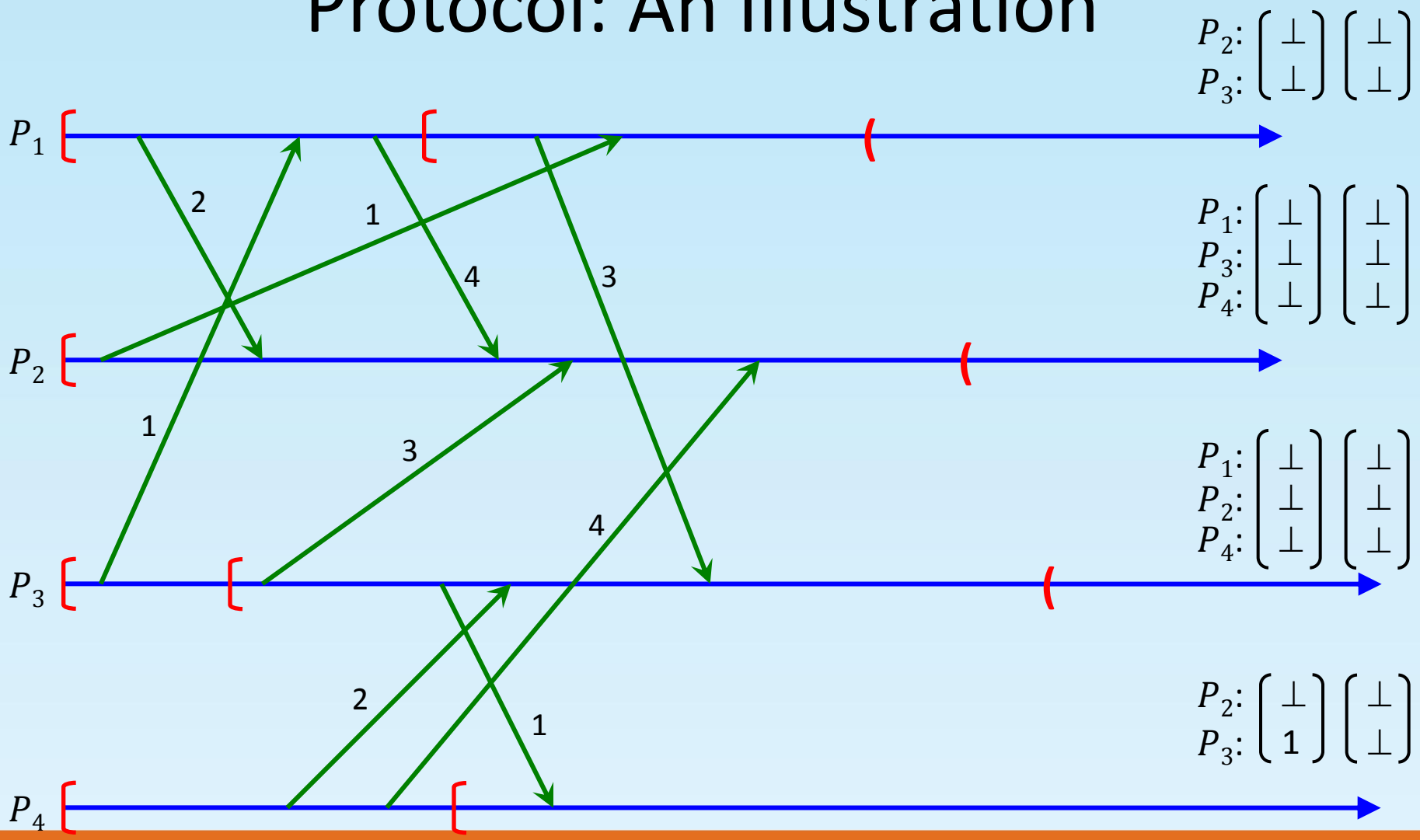
P_4 receives the message from P_3 and processes it

Koo and Toueg's Checkpointing Protocol: An Illustration



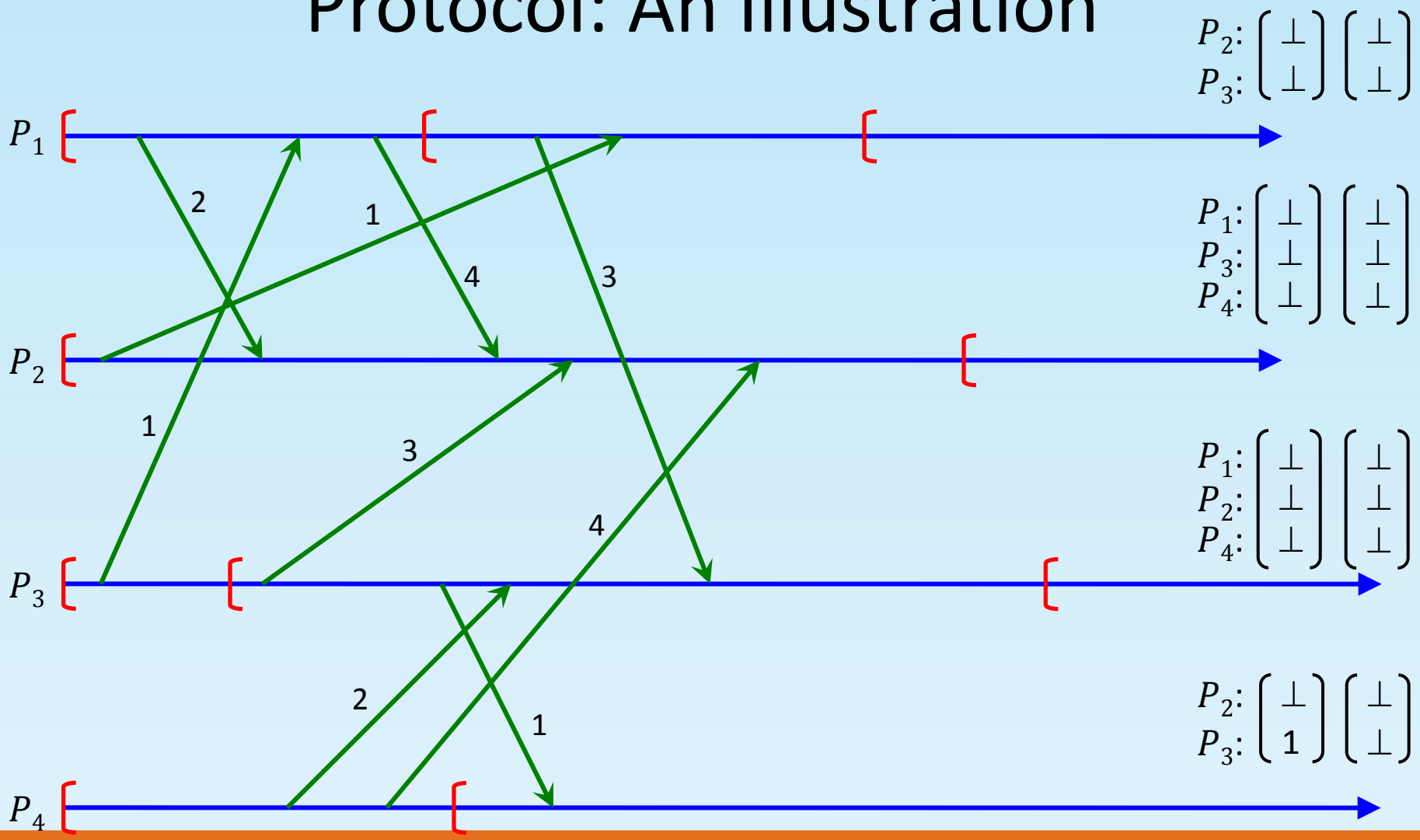
P_4 tests whether $last\_label\_rcvd_3[4] \geq first\_label\_sent_4[3] > \perp$ holds

Koo and Toueg's Checkpointing Protocol: An Illustration



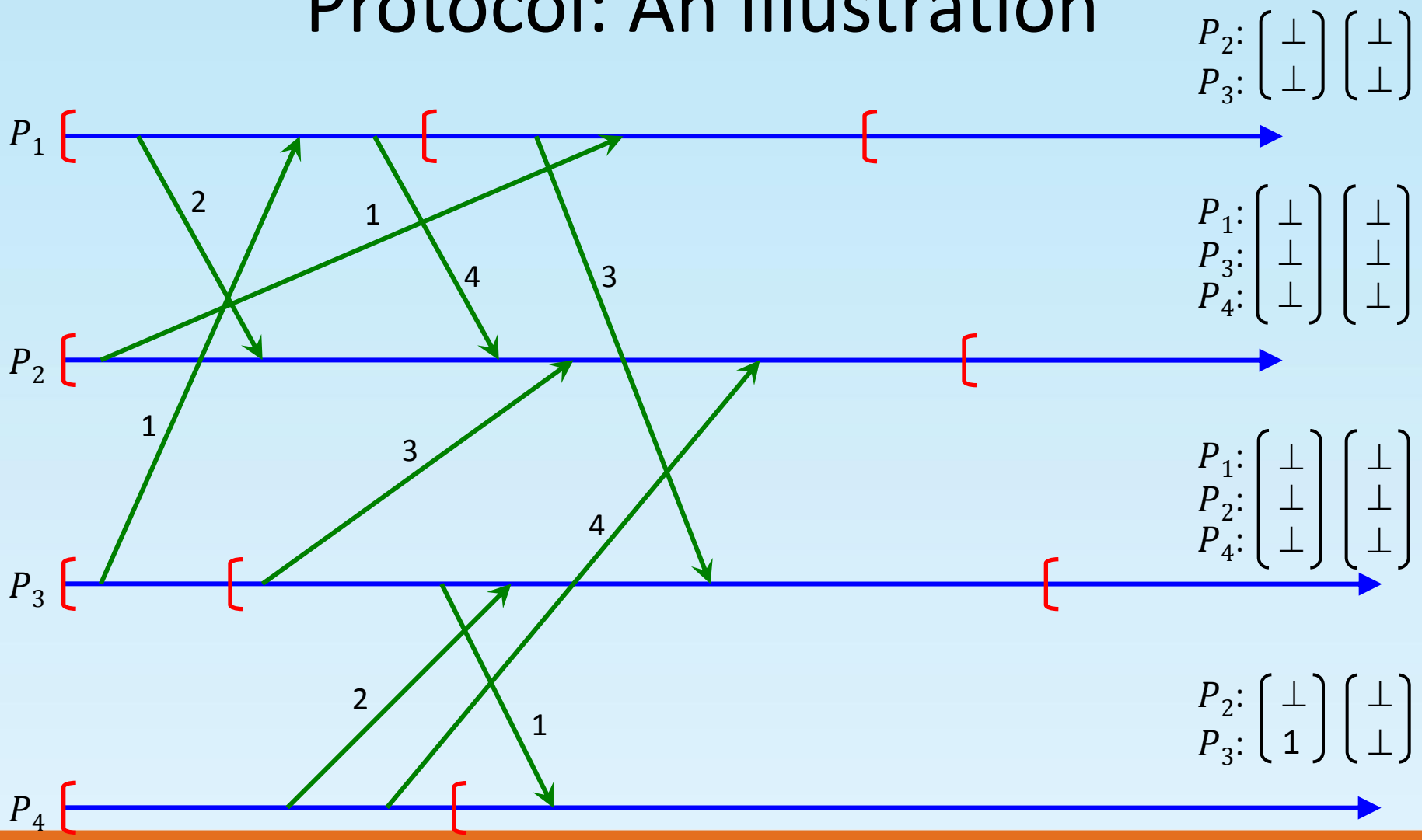
The test fails and P_4 does not take any checkpoint

Koo and Toueg's Checkpointing Protocol: An Illustration



All tentative checkpoints are made permanent and the checkpoint protocol terminates

Koo and Toueg's Checkpointing Protocol: An Illustration



Koo and Toueg's Recovery Protocol

- Only permanent checkpoints are used in recovery
- Consists of two phases
  - First Phase:
    - Processes agree to roll back if they can
      - A process after agreeing to roll back stops its execution until the second phase completes
  - Second Phase:
    - If all required processes agree to roll back in the first phase, then they restart their execution from the last checkpoint
    - Otherwise, processes resume their execution from their current point

Koo and Toueg's Recovery Protocol: Details

- Consider processes P_i and P_j that are neighbors
- Definition of $last\_label\_senti[j]$:
  - Let m be the last message that P_i sent to P_j before its last permanent checkpoint

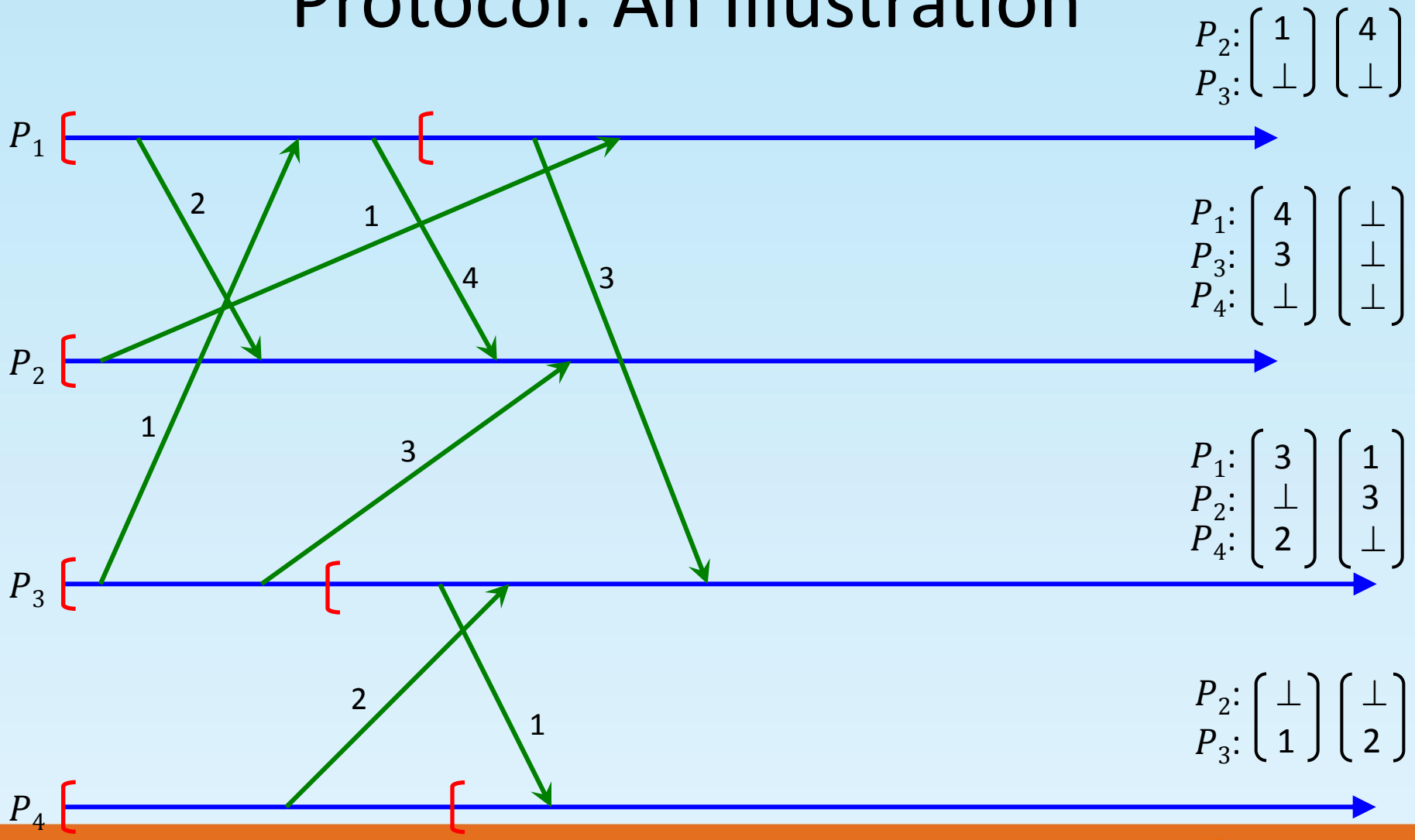
$$last\_label\_senti[j] \triangleq \begin{cases} \text{label of } m, & m \text{ exists} \\ \perp, & m \text{ does not exist} \end{cases}$$

Koo and Toueg's Recovery Protocol: Details (Contd.)

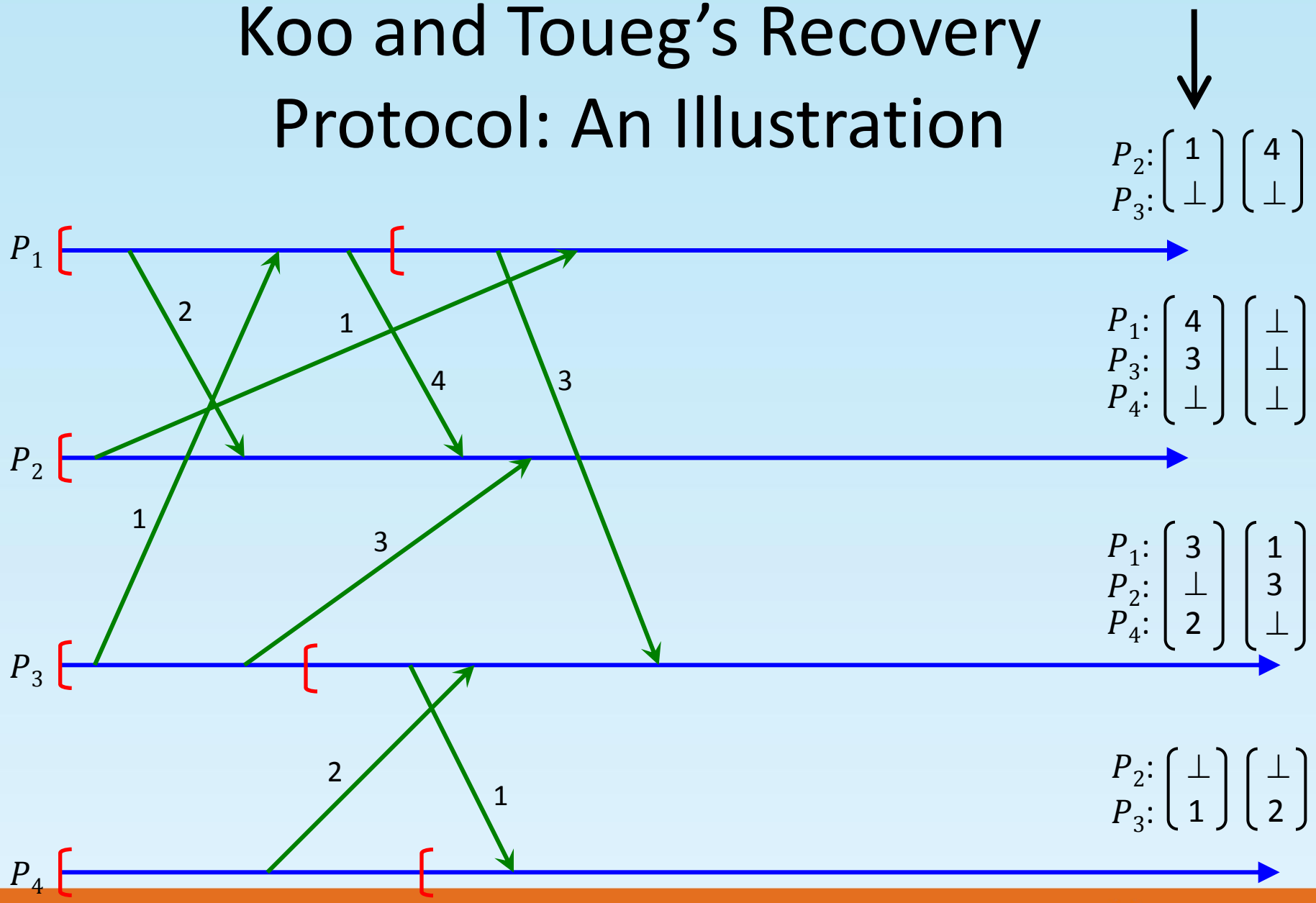
- Assume that P_i has agreed to roll back:
  - P_i requests P_j to roll back sends $last\_label\_senti[j]$ to P_j
  - P_j agrees to roll back if:

$$last\_label\_rcvdj[i] \; > \; last\_label\_senti[j]$$

Koo and Toueg's Recovery Protocol: An Illustration

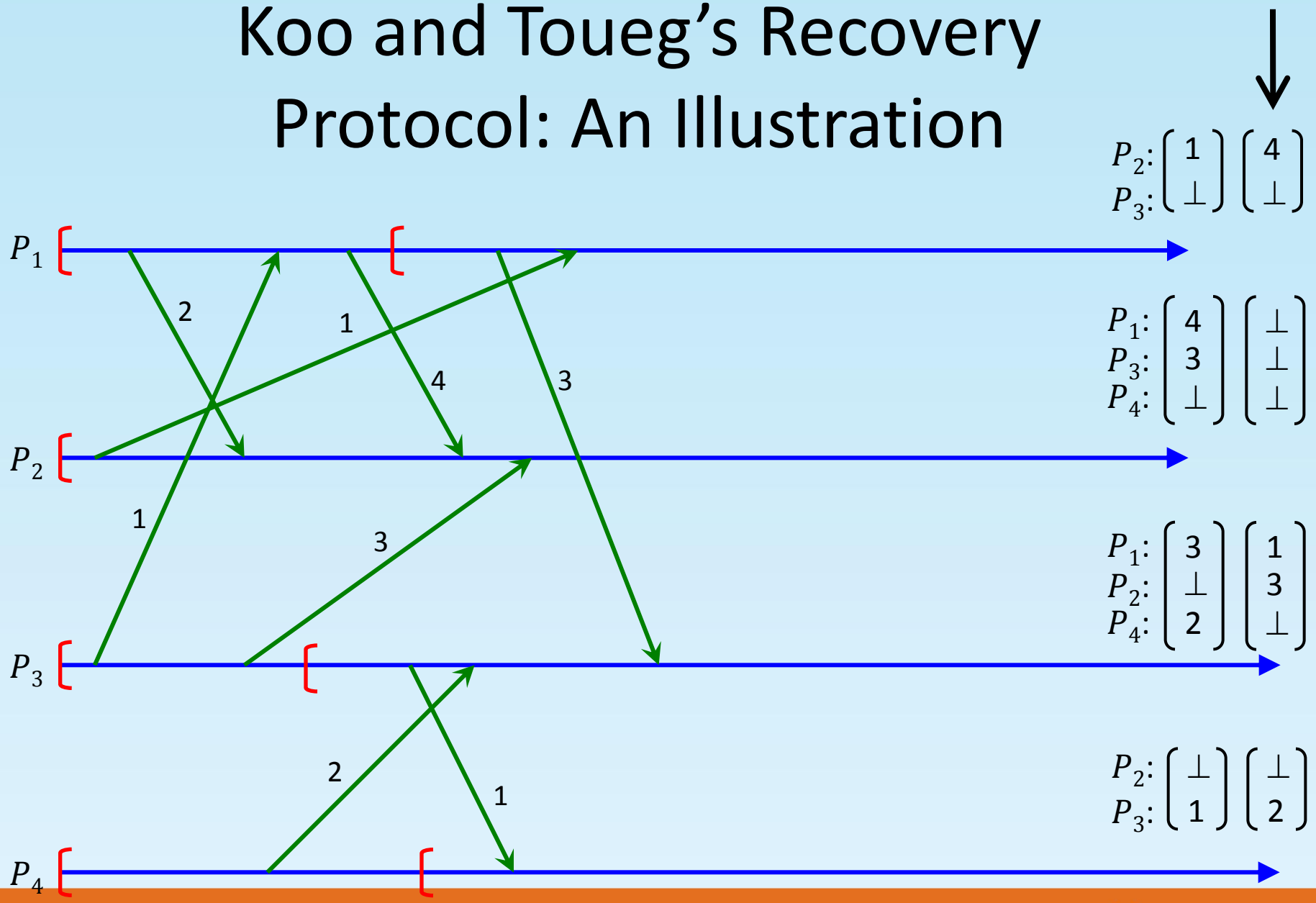


Koo and Toueg's Recovery Protocol: An Illustration



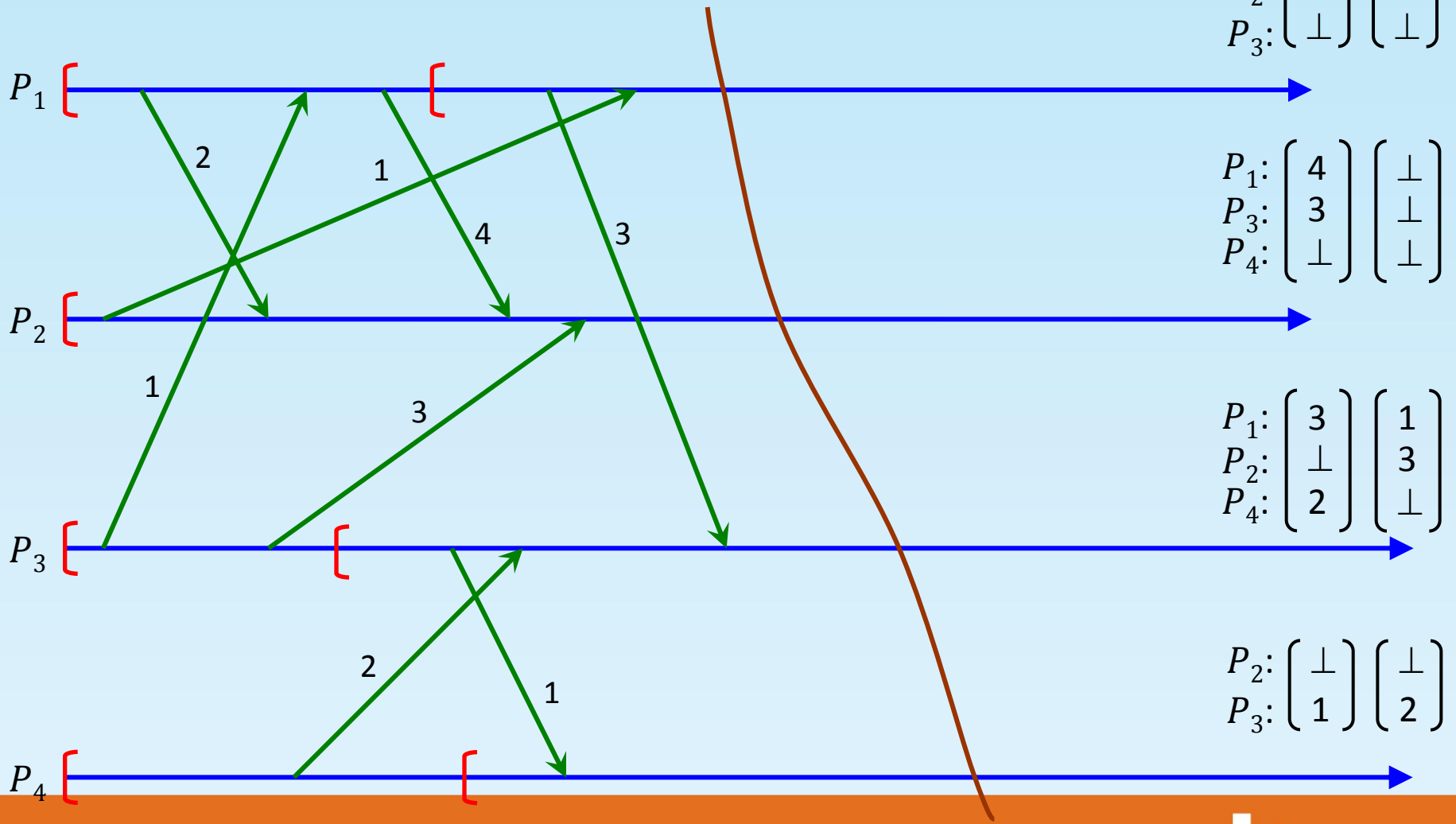
First vector: $last\_label\_rcvd$

Koo and Toueg's Recovery Protocol: An Illustration



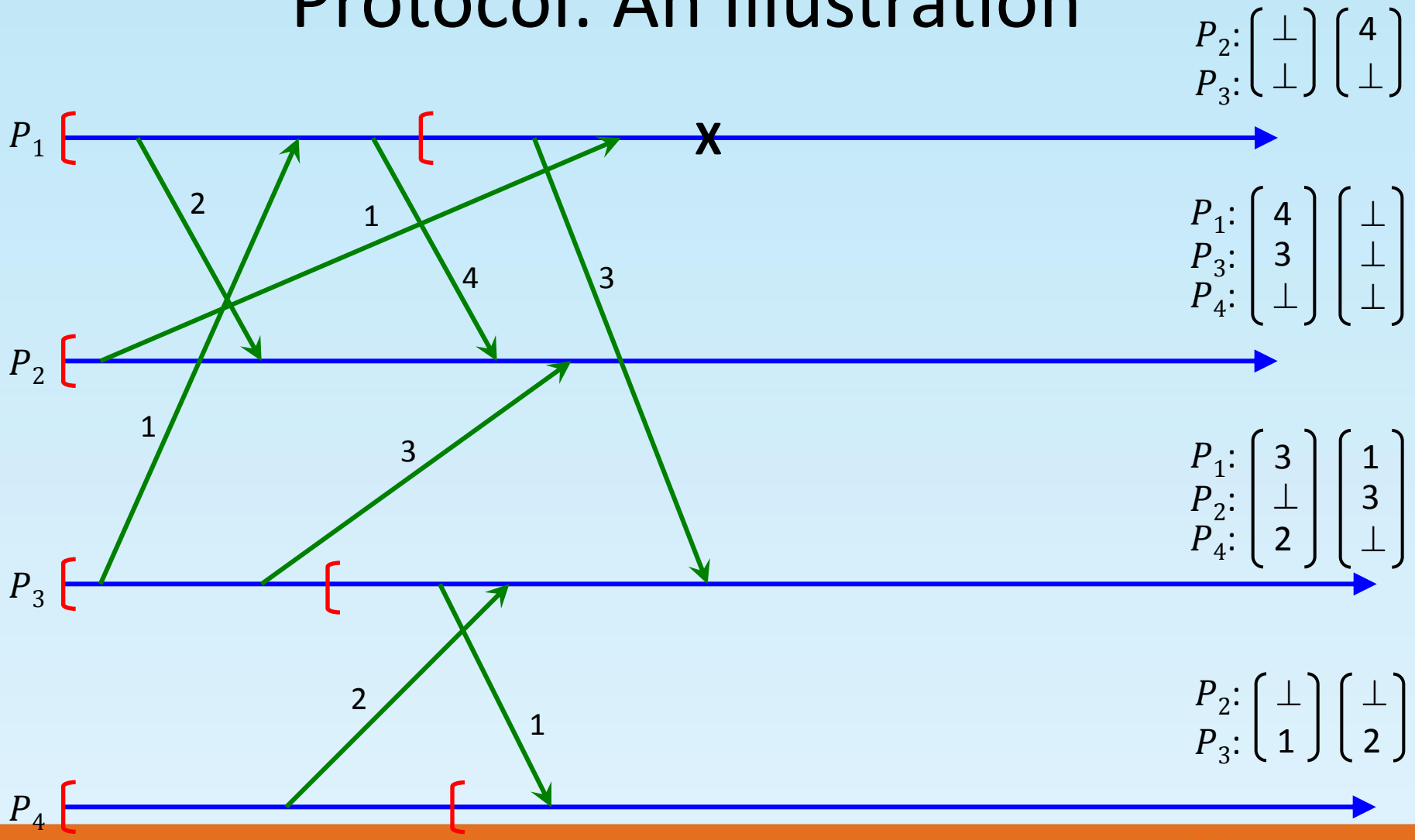
Second vector: *last_label_sent*

Koo and Toueg's Recovery Protocol: An Illustration



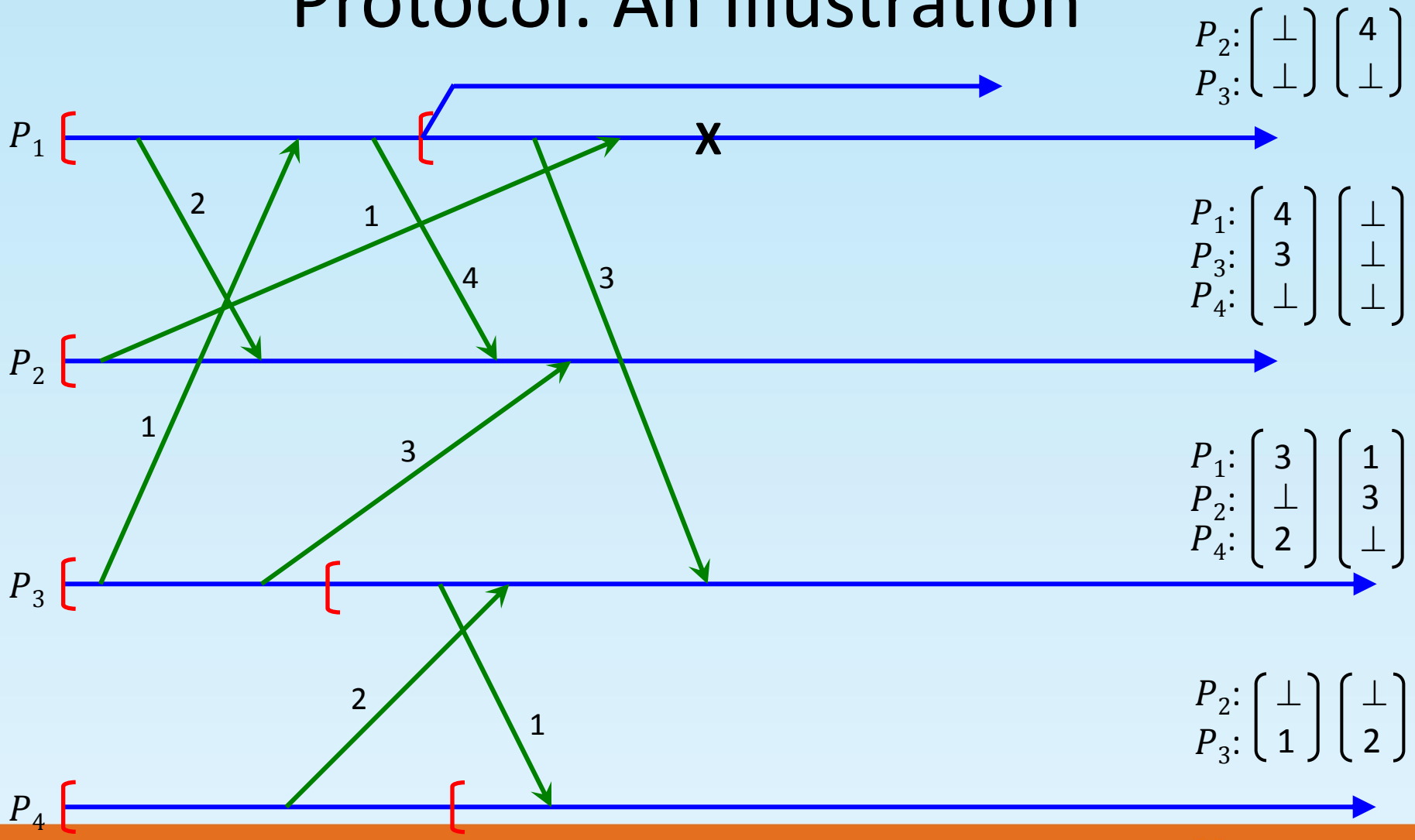
Current system state

Koo and Toueg's Recovery Protocol: An Illustration



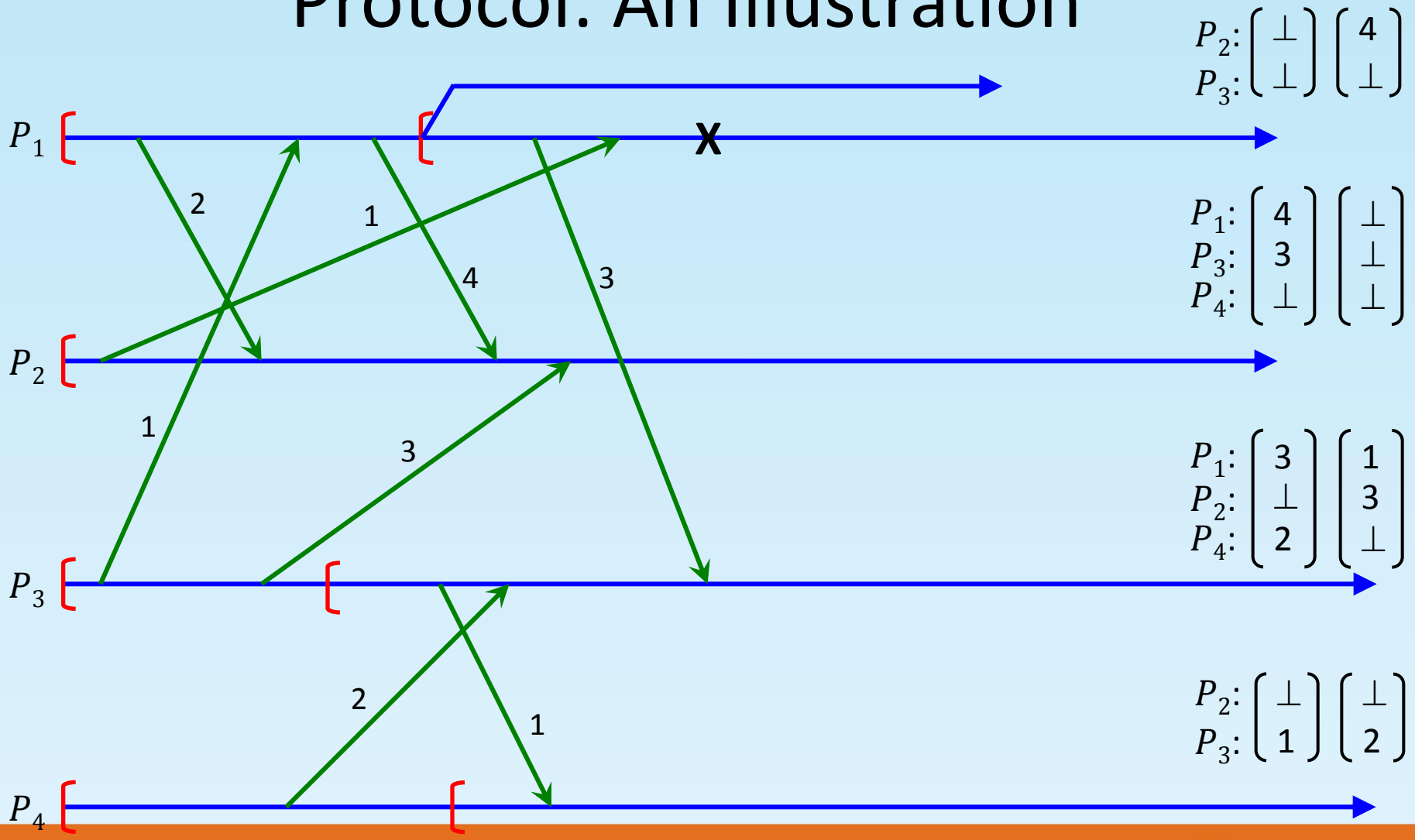
P_1 crashes

Koo and Toueg's Recovery Protocol: An Illustration



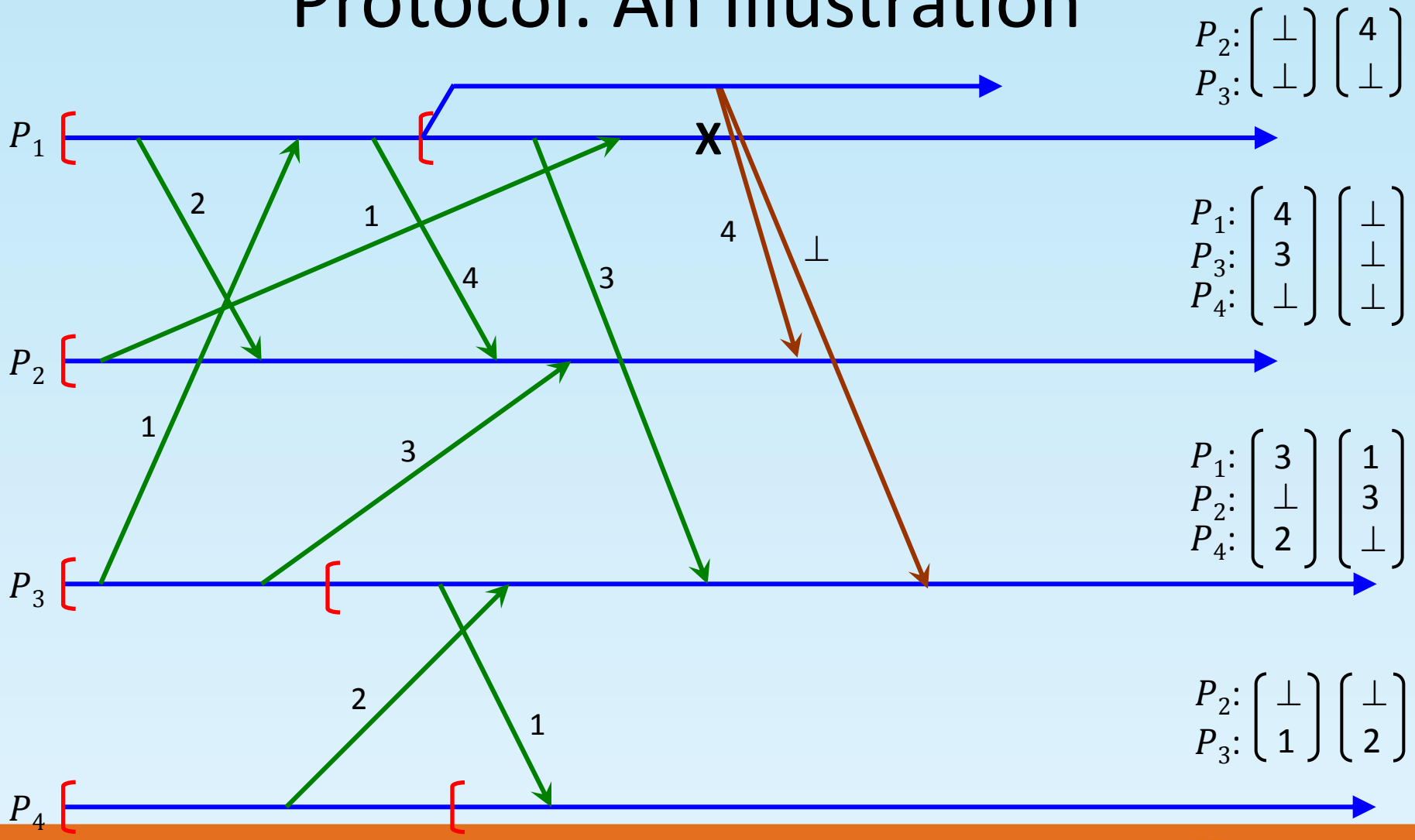
P_1 restarts from the last (permanent) checkpoint

Koo and Toueg's Recovery Protocol: An Illustration



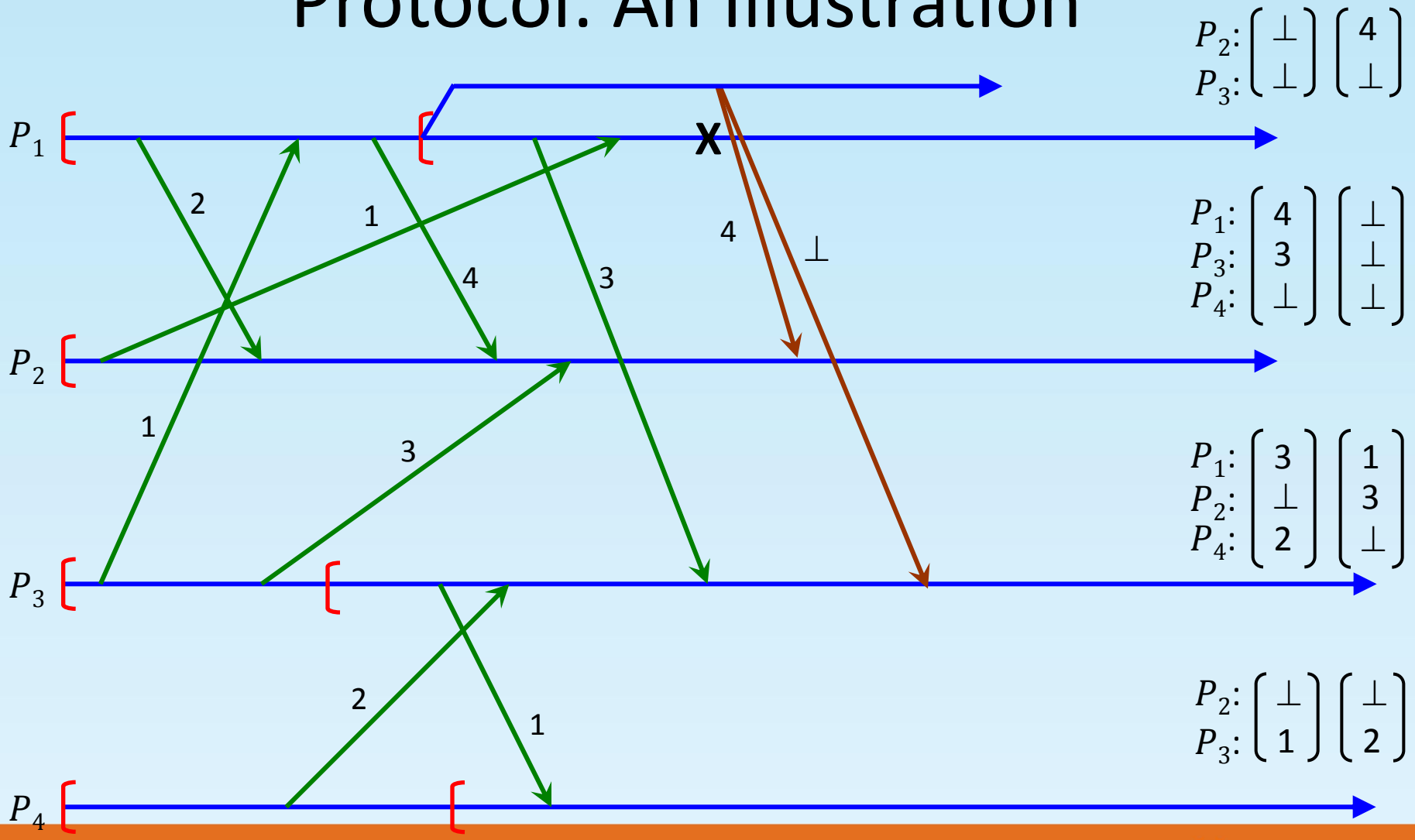
P_1 sends the value of $last\_label\_sent$ entries to its neighbors

Koo and Toueg's Recovery Protocol: An Illustration



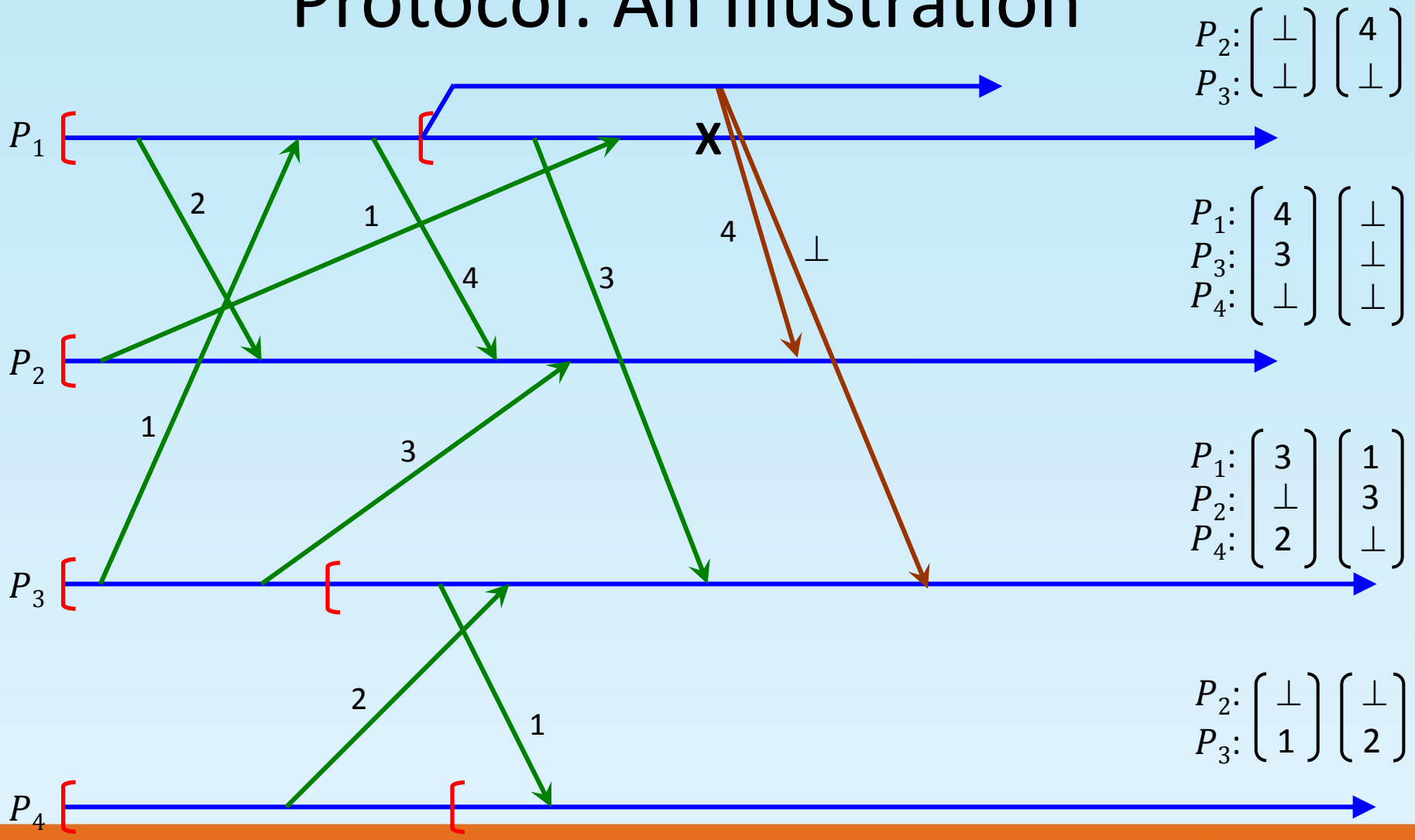
Specifically, P_1 sends 4 to P_2 and $\perp$ to P_3

Koo and Toueg's Recovery Protocol: An Illustration



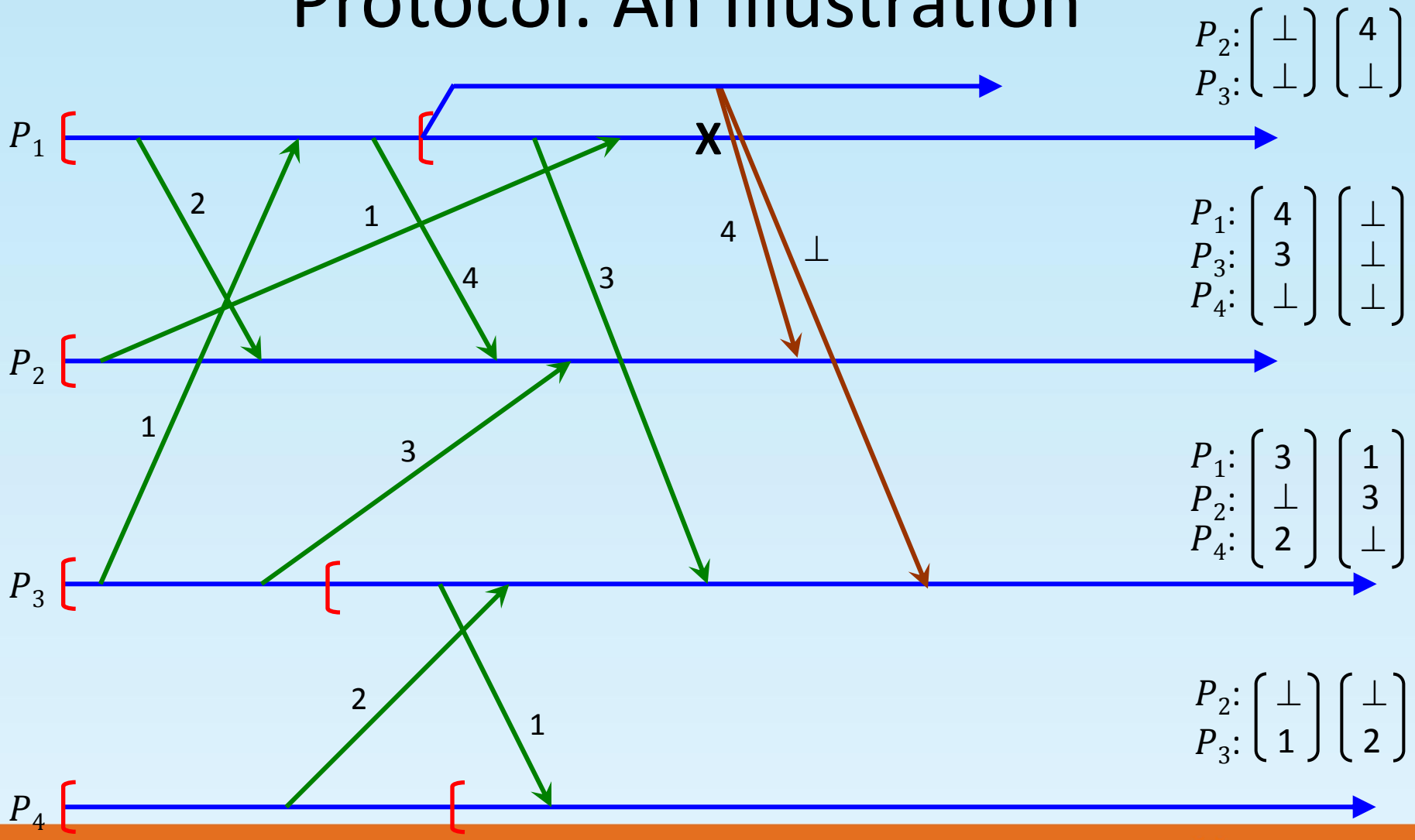
P_2 receives the message from P_1 and processes it

Koo and Toueg's Recovery Protocol: An Illustration



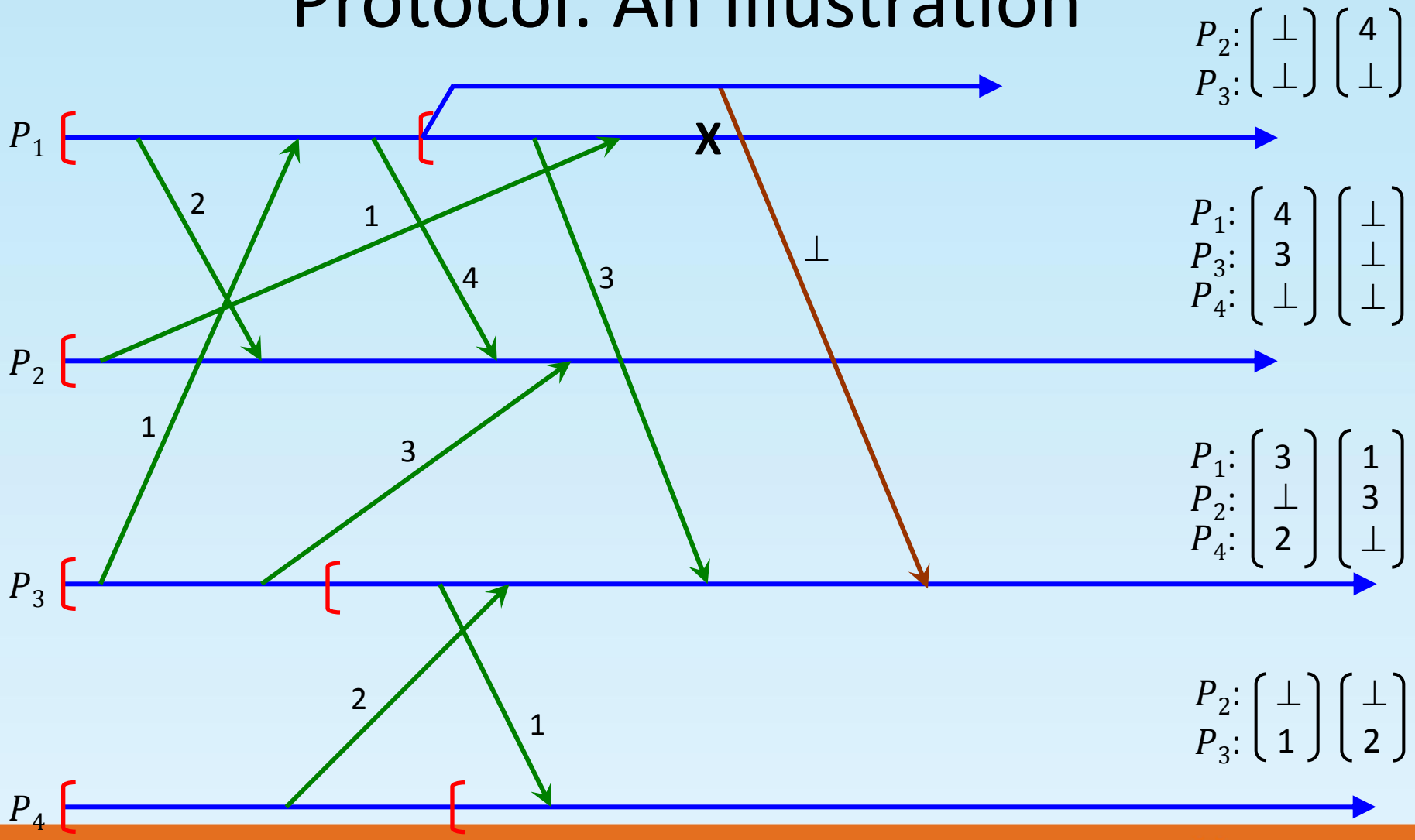
P_2 tests whether $last\_label\_rcvd_2[1] > last\_label\_sent_1[2]$ holds

Koo and Toueg's Recovery Protocol: An Illustration



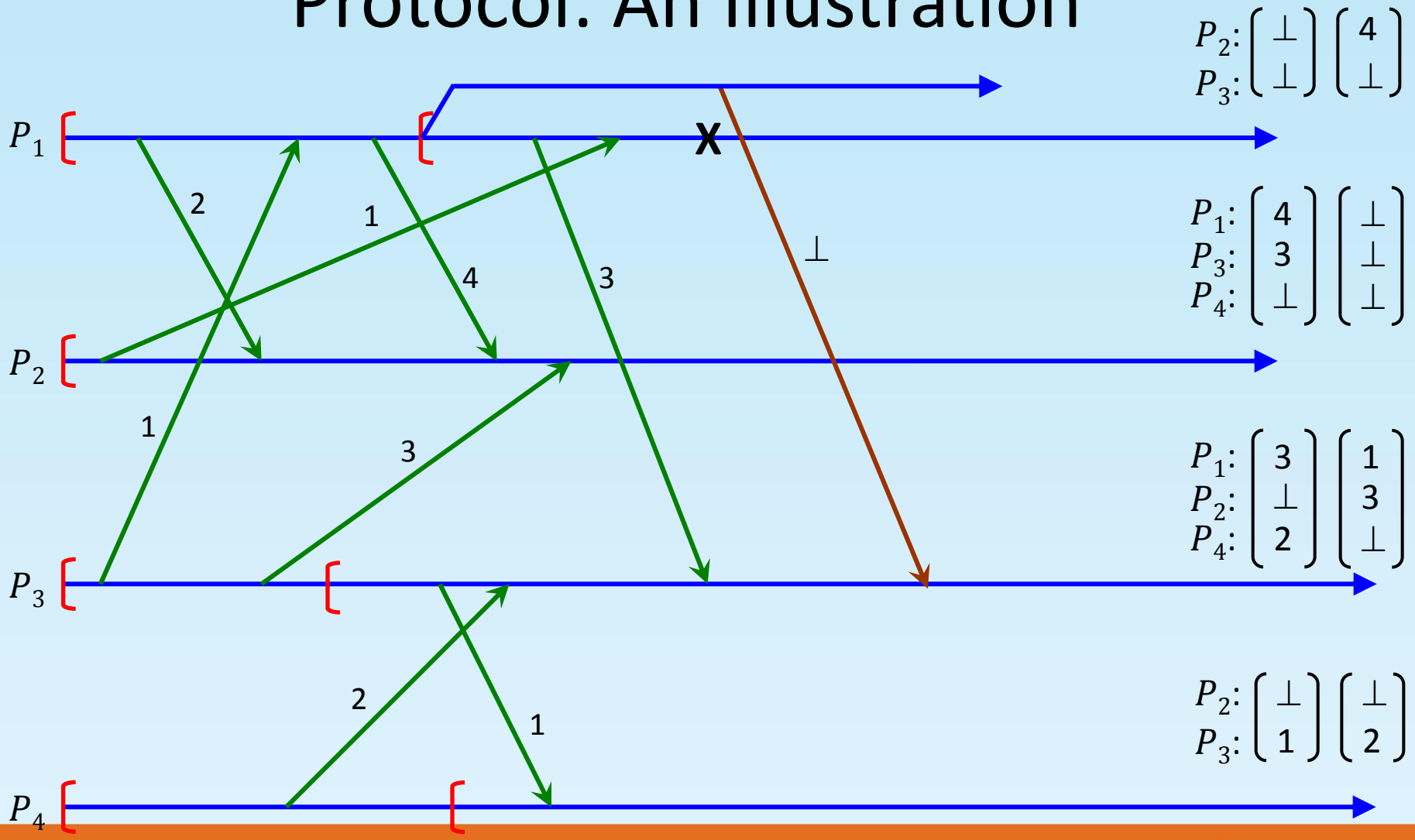
The test fails and P_2 does not have to rollback

Koo and Toueg's Recovery Protocol: An Illustration



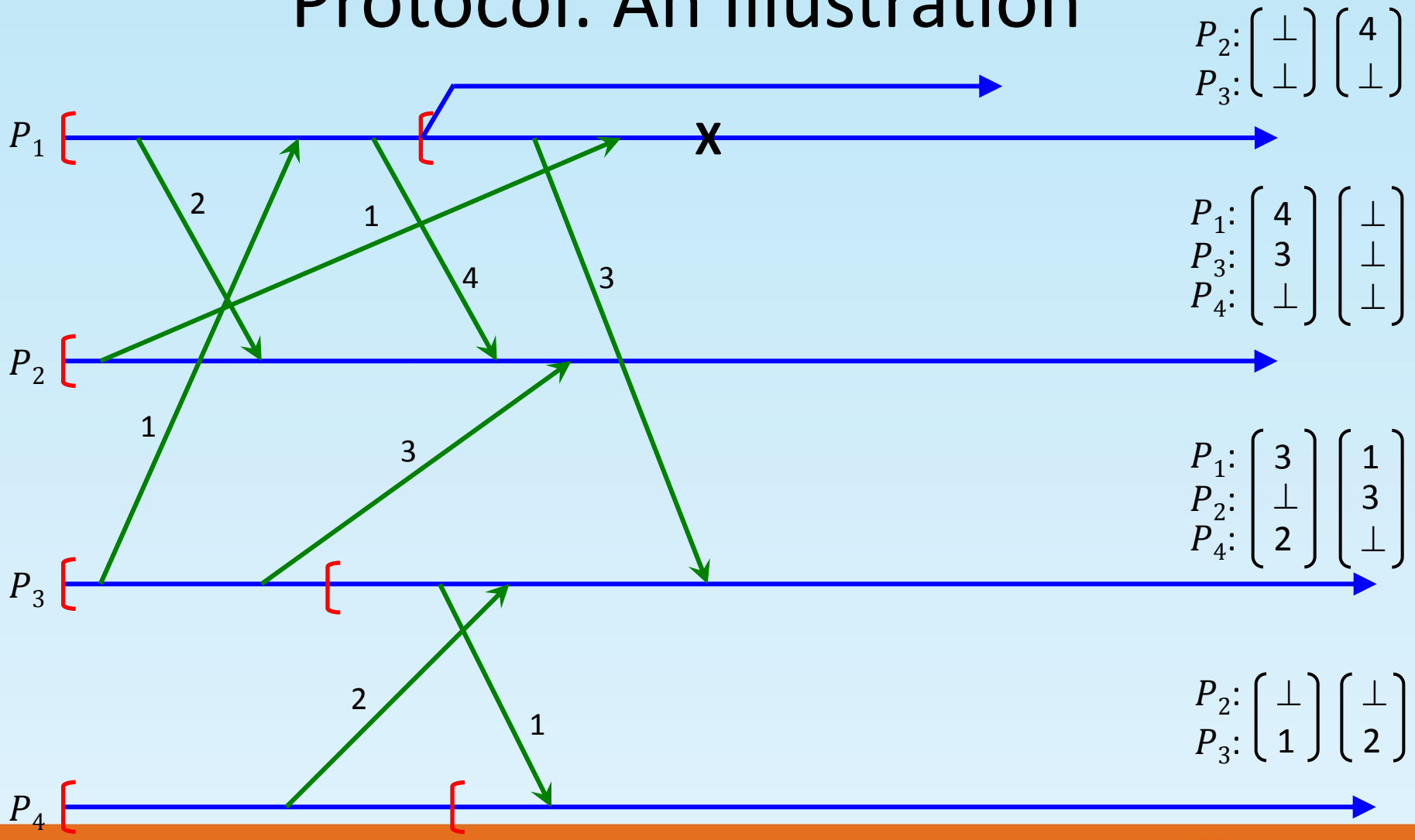
P_3 receives the message from P_1 and processes it

Koo and Toueg's Recovery Protocol: An Illustration



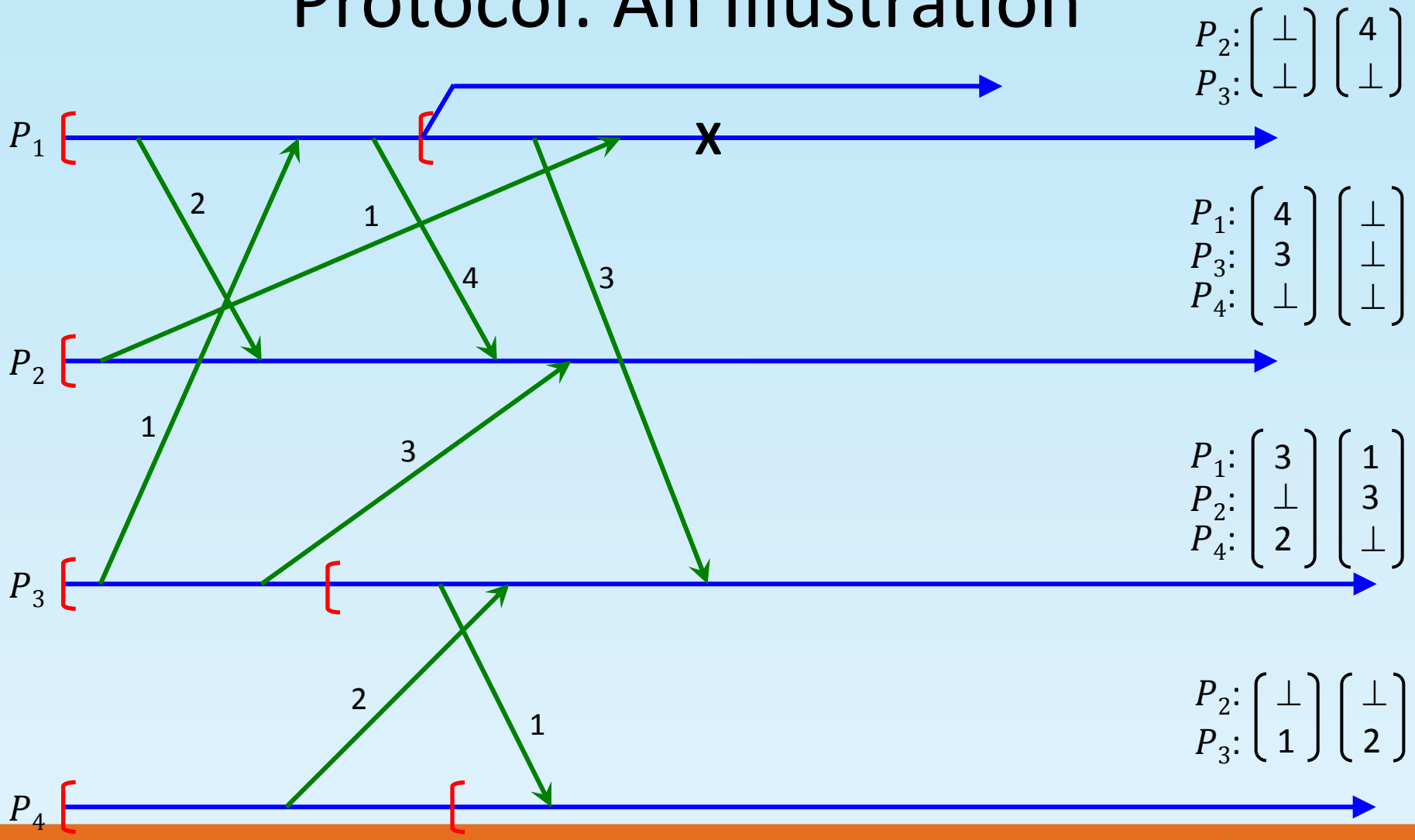
P_3 tests whether $last\_label\_rcvd_3[1] > last\_label\_sent_1[3]$ holds

Koo and Toueg's Recovery Protocol: An Illustration



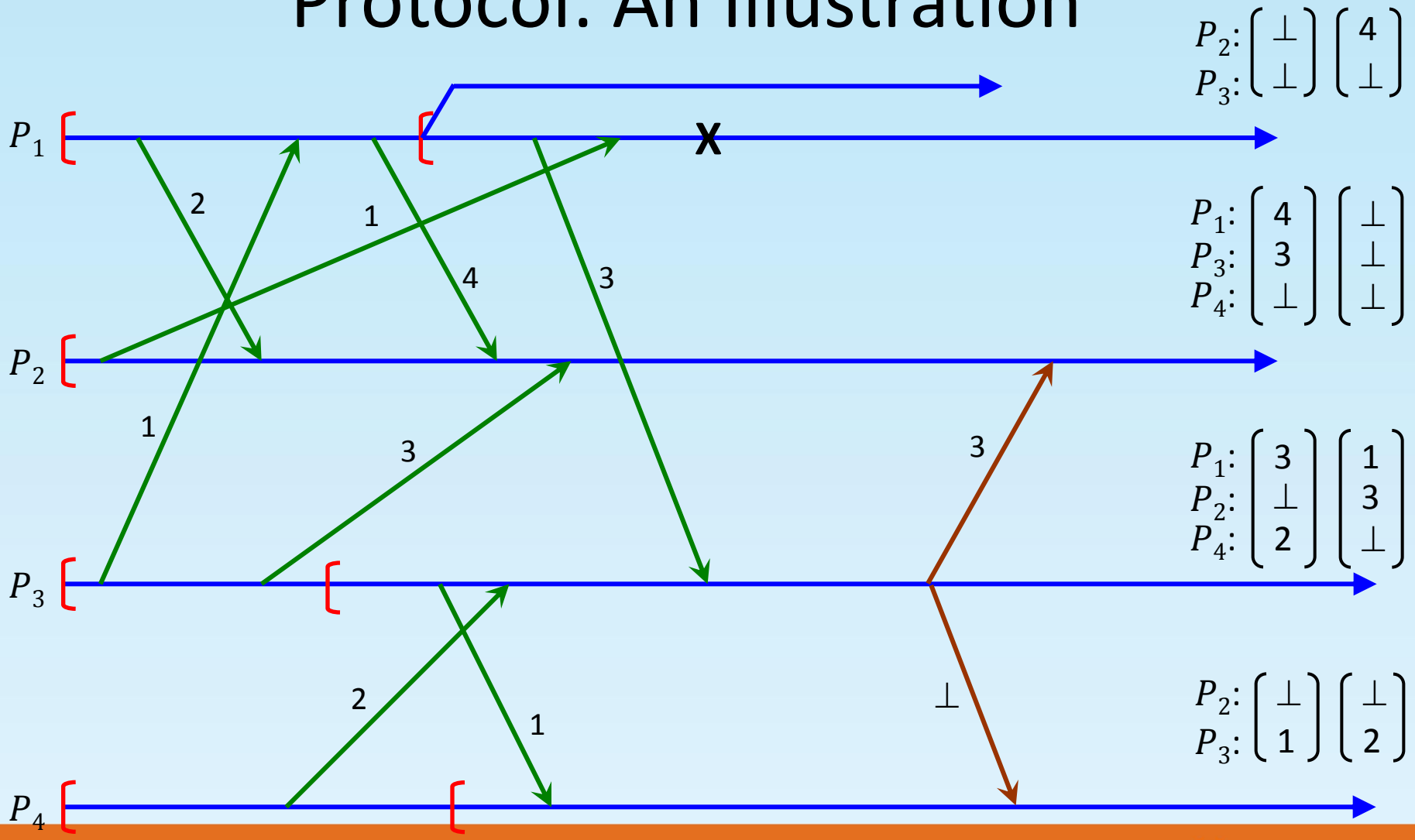
The test evaluates to true and P_3 agrees to rollback

Koo and Toueg's Recovery Protocol: An Illustration



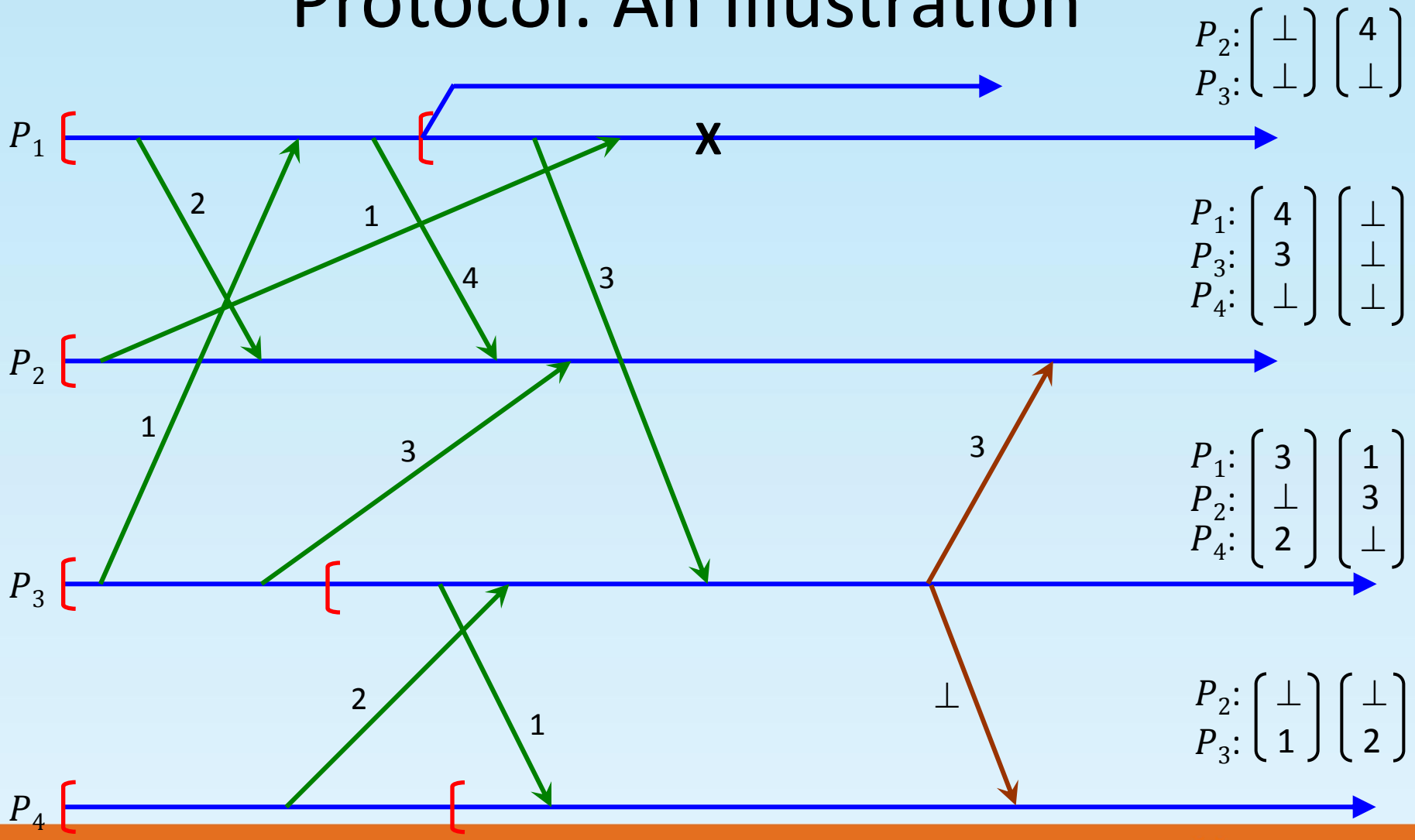
P_3 sends the value of $last\_label\_sent$ entries to its neighbors

Koo and Toueg's Recovery Protocol: An Illustration



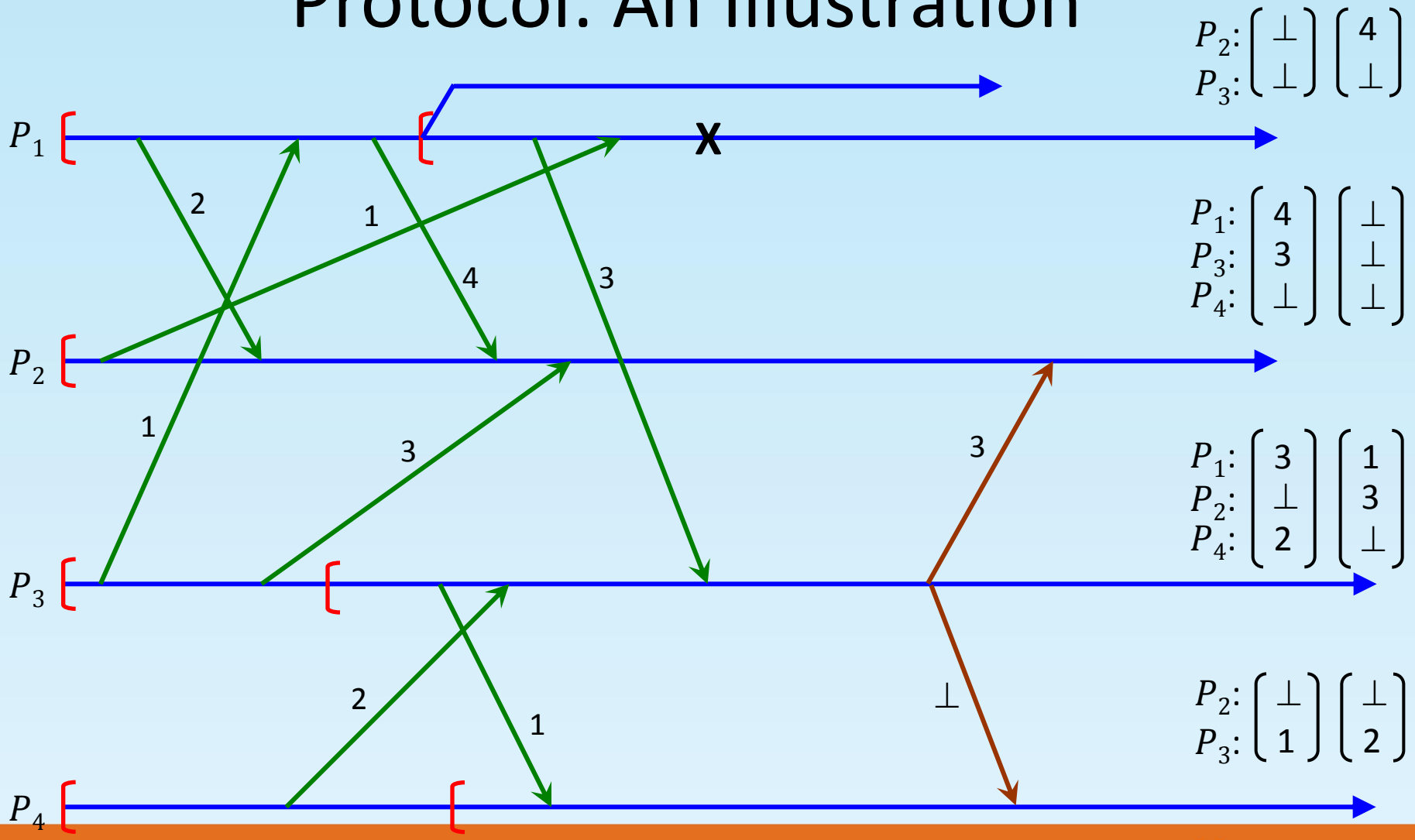
Specifically, P_3 sends 3 to P_2 and $\perp$ to P_4

Koo and Toueg's Recovery Protocol: An Illustration



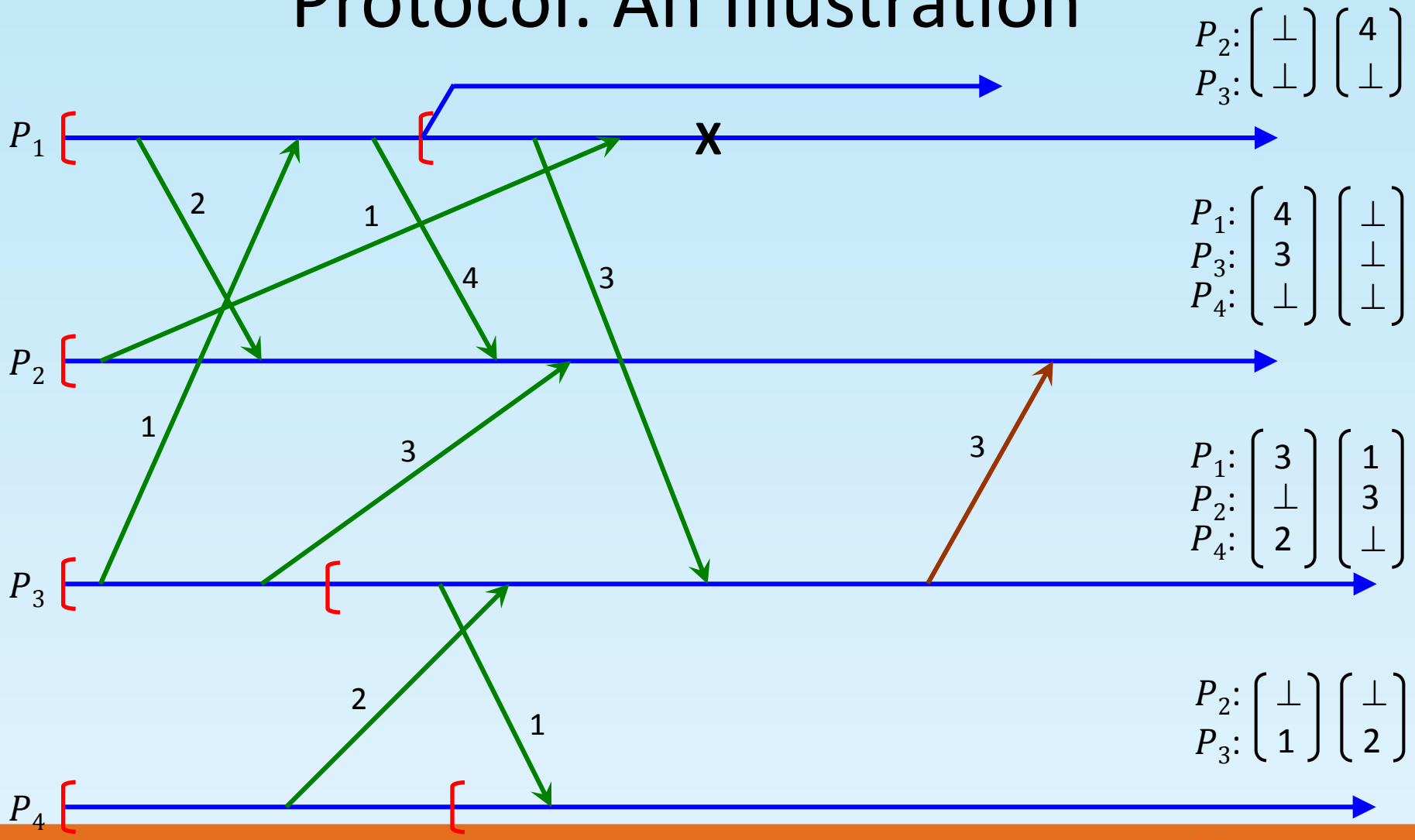
P_4 receives the message from P_3 and processes it

Koo and Toueg's Recovery Protocol: An Illustration



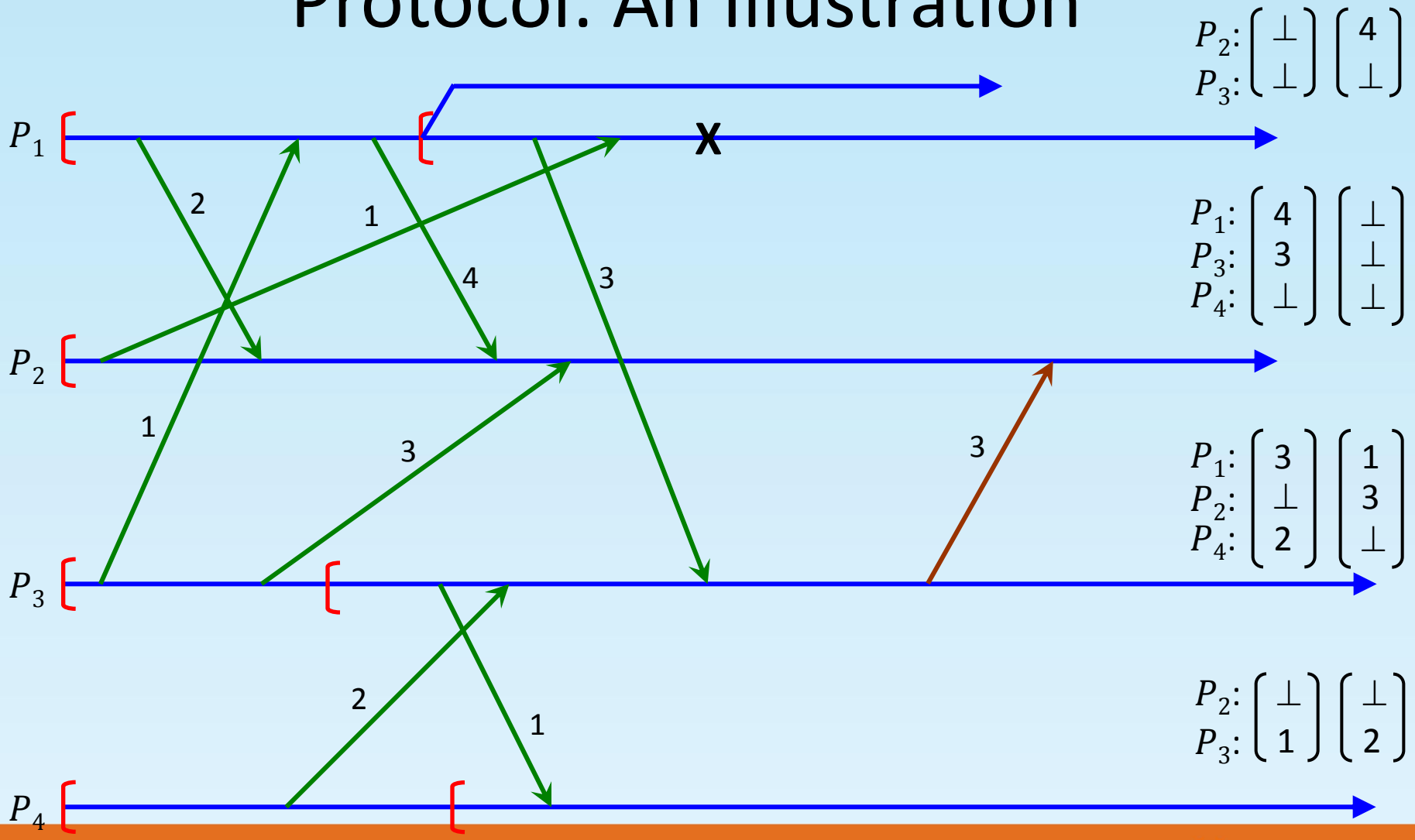
P_4 tests whether $last\_label\_rcvd_4[3] > last\_label\_sent_3[4]$ holds

Koo and Toueg's Recovery Protocol: An Illustration



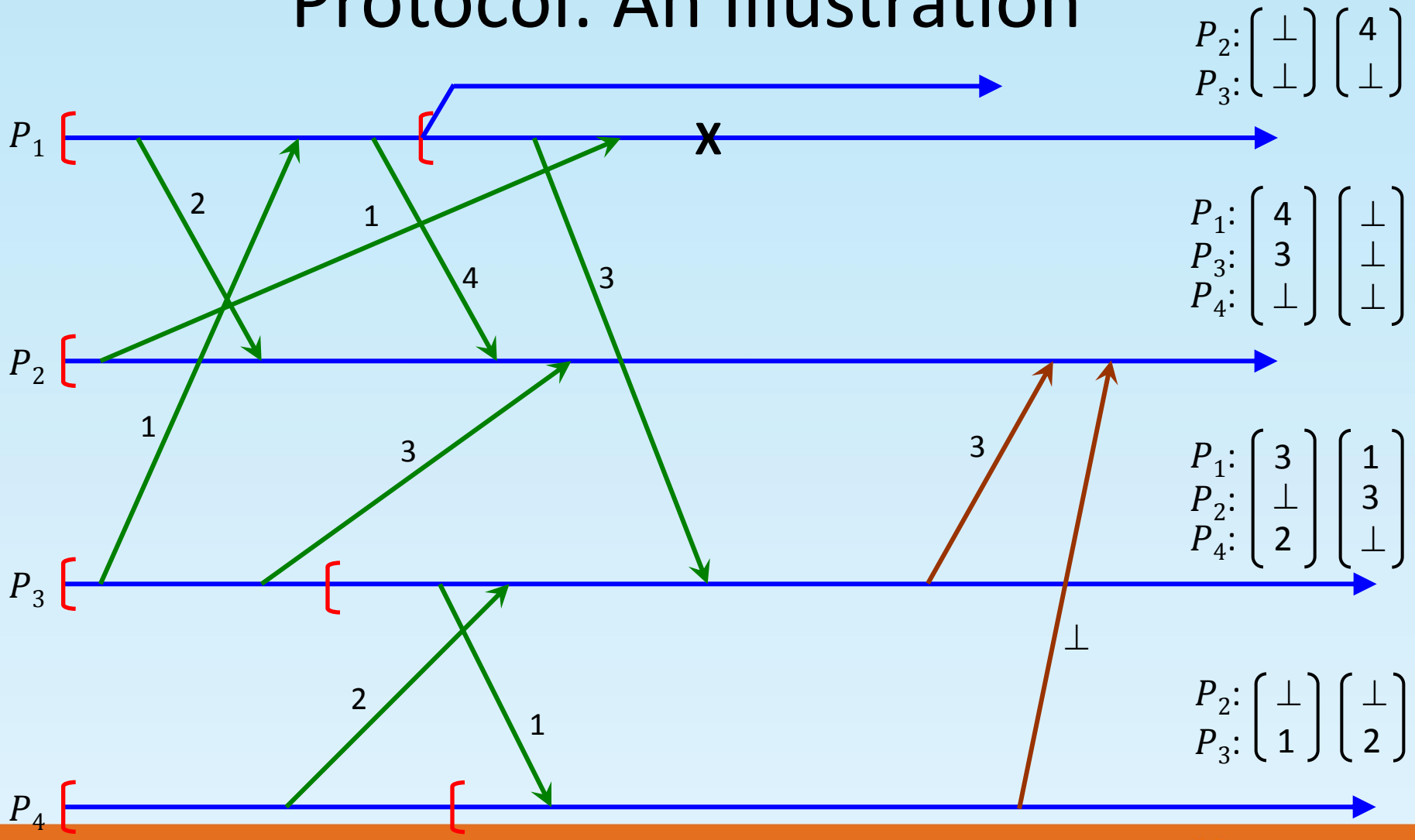
The test evaluates to true and P_4 agrees to rollback

Koo and Toueg's Recovery Protocol: An Illustration



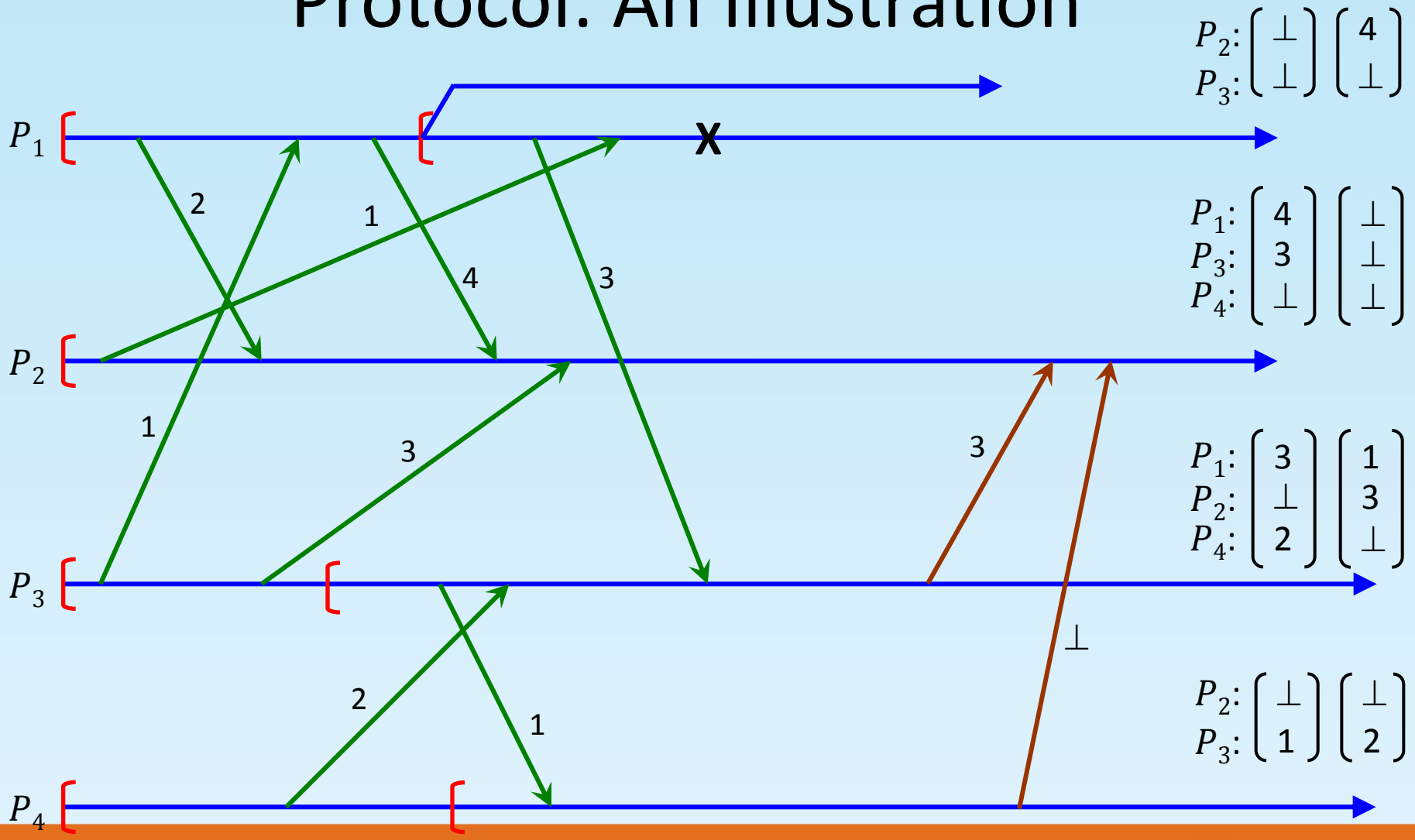
P_4 sends the value of $last\_label\_sent$ entries to its neighbors

Koo and Toueg's Recovery Protocol: An Illustration



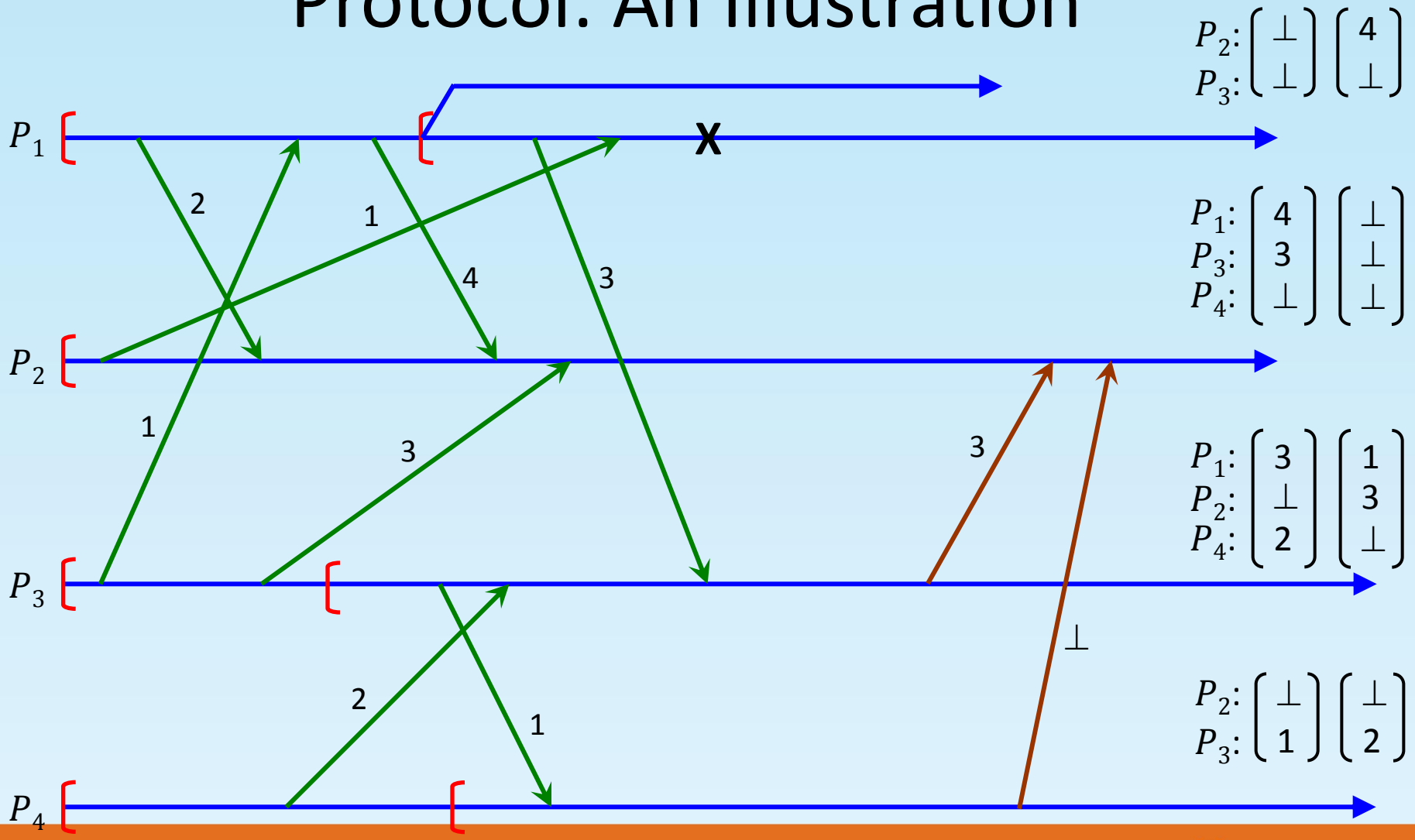
Specifically, P_4 sends $\perp$ to P_2

Koo and Toueg's Recovery Protocol: An Illustration



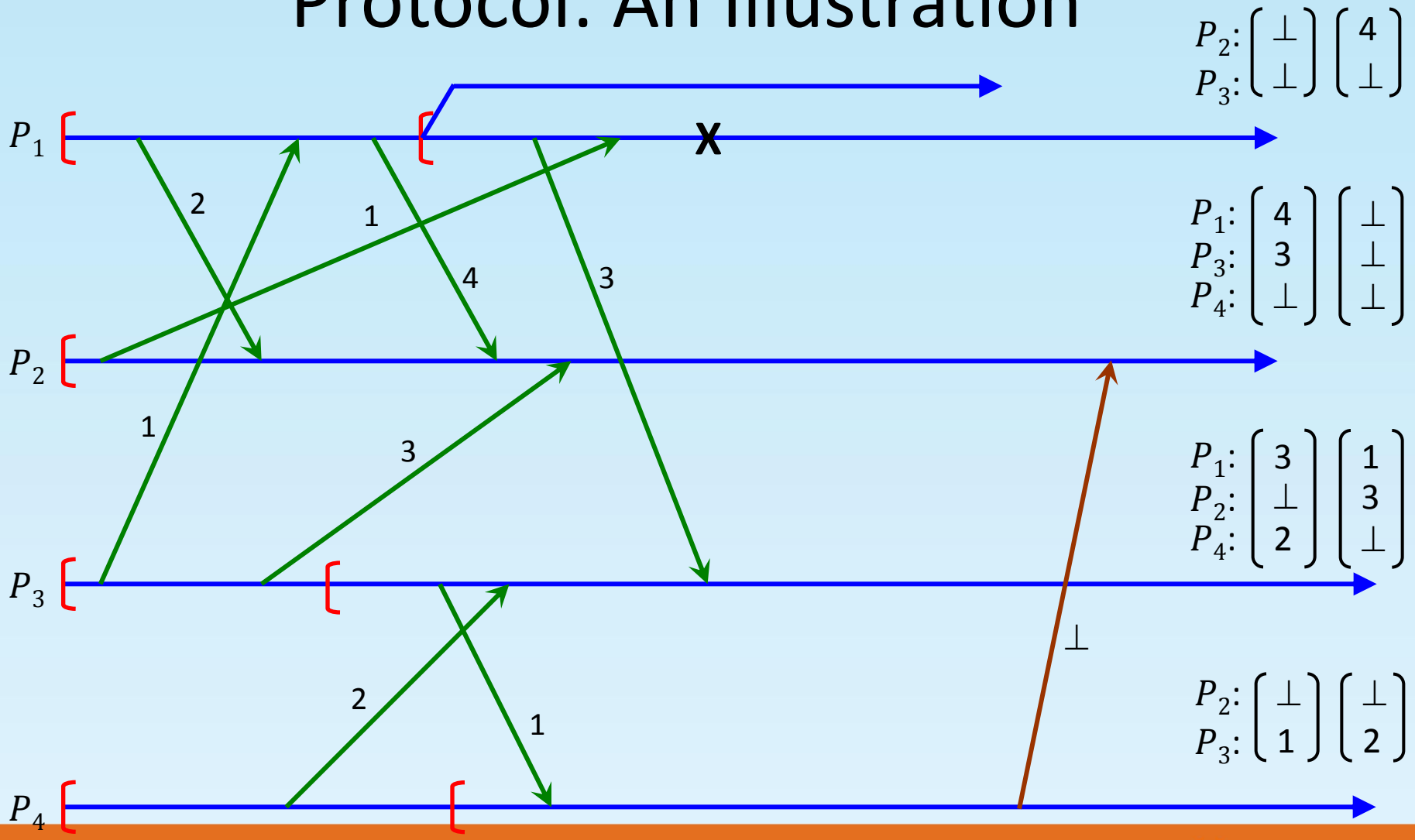
P_2 receives the message from P_3 and processes it

Koo and Toueg's Recovery Protocol: An Illustration



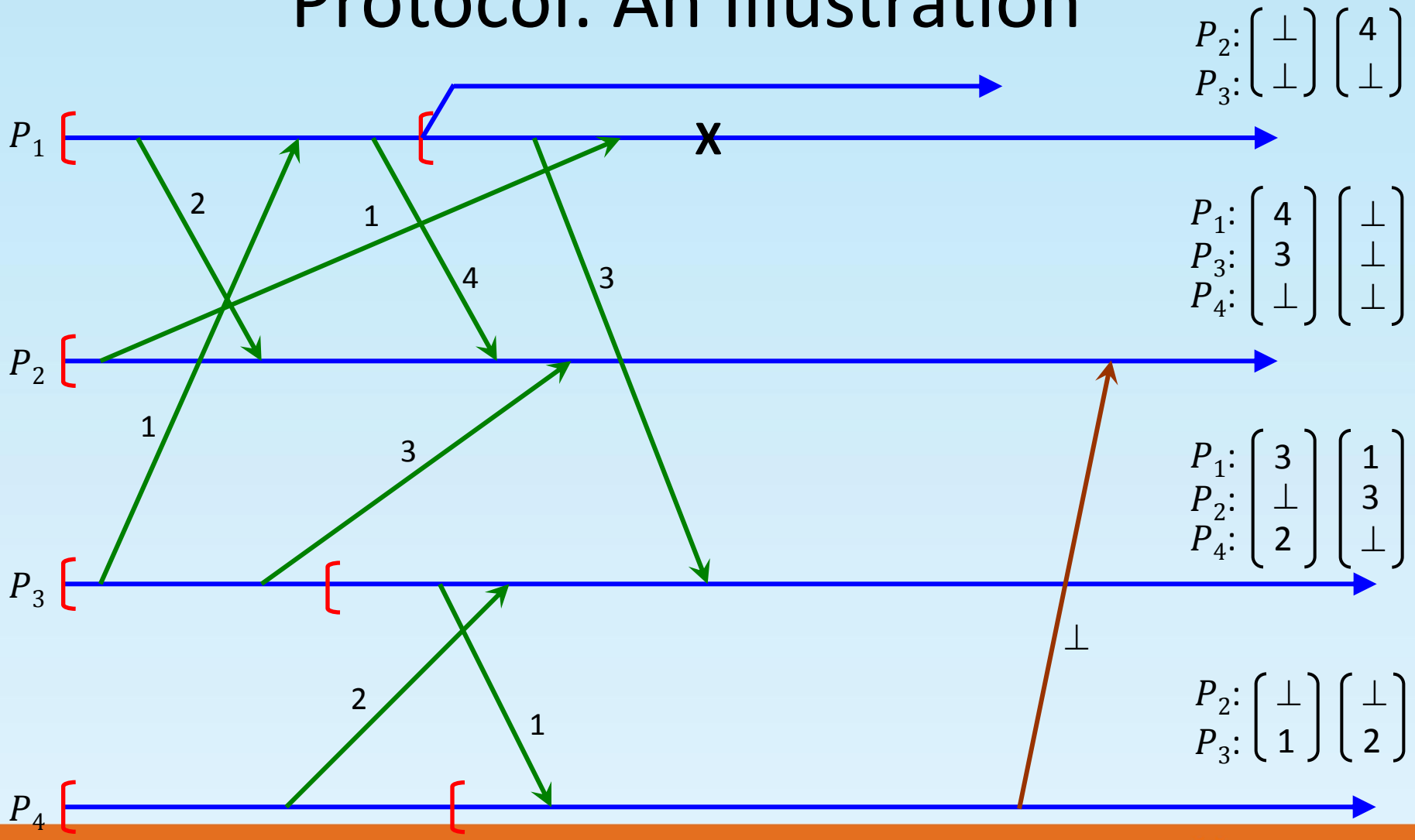
P_2 tests whether $last\_label\_rcvd_2[3] > last\_label\_sent_3[2]$ holds

Koo and Toueg's Recovery Protocol: An Illustration



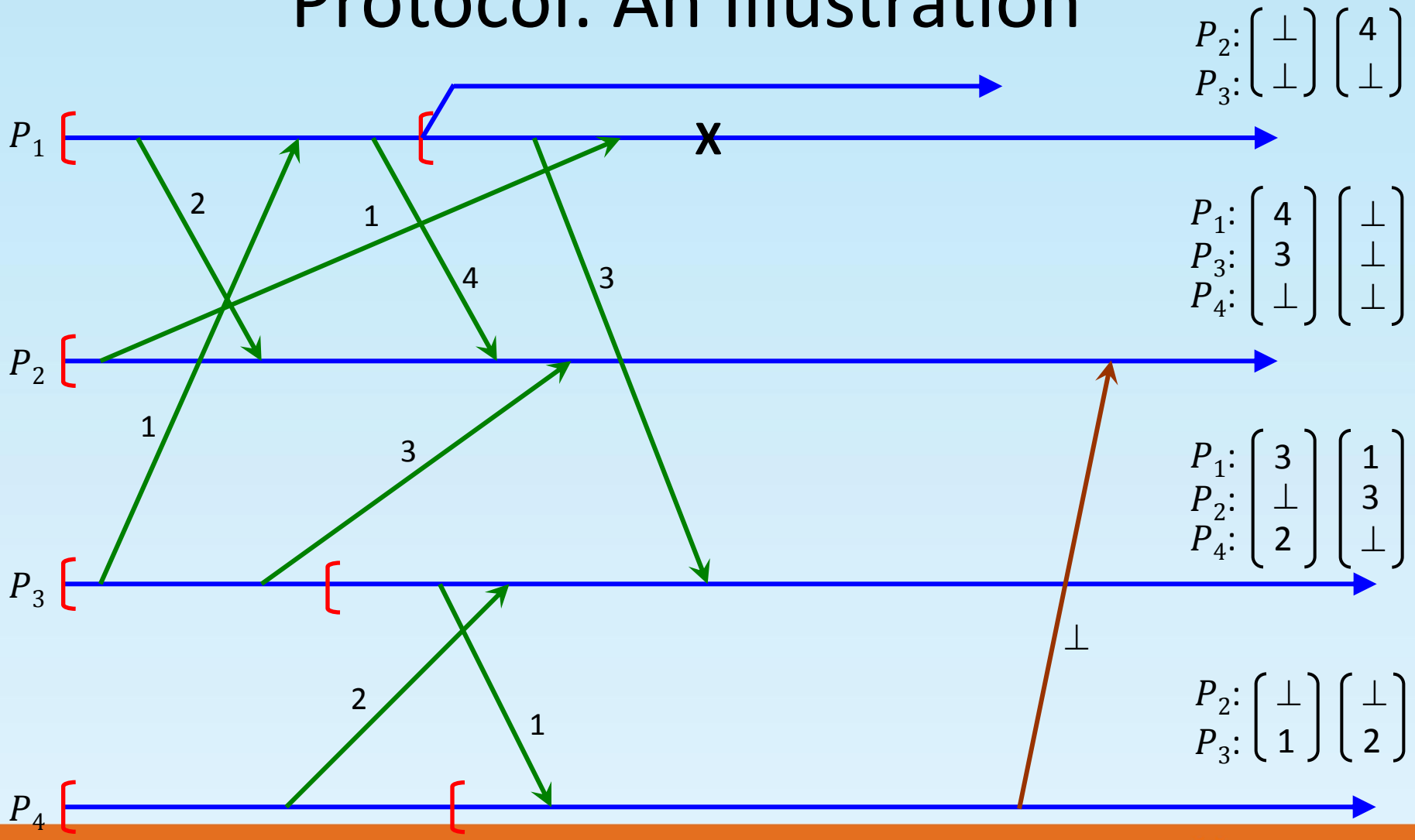
The test fails and P_2 does not have to rollback

Koo and Toueg's Recovery Protocol: An Illustration



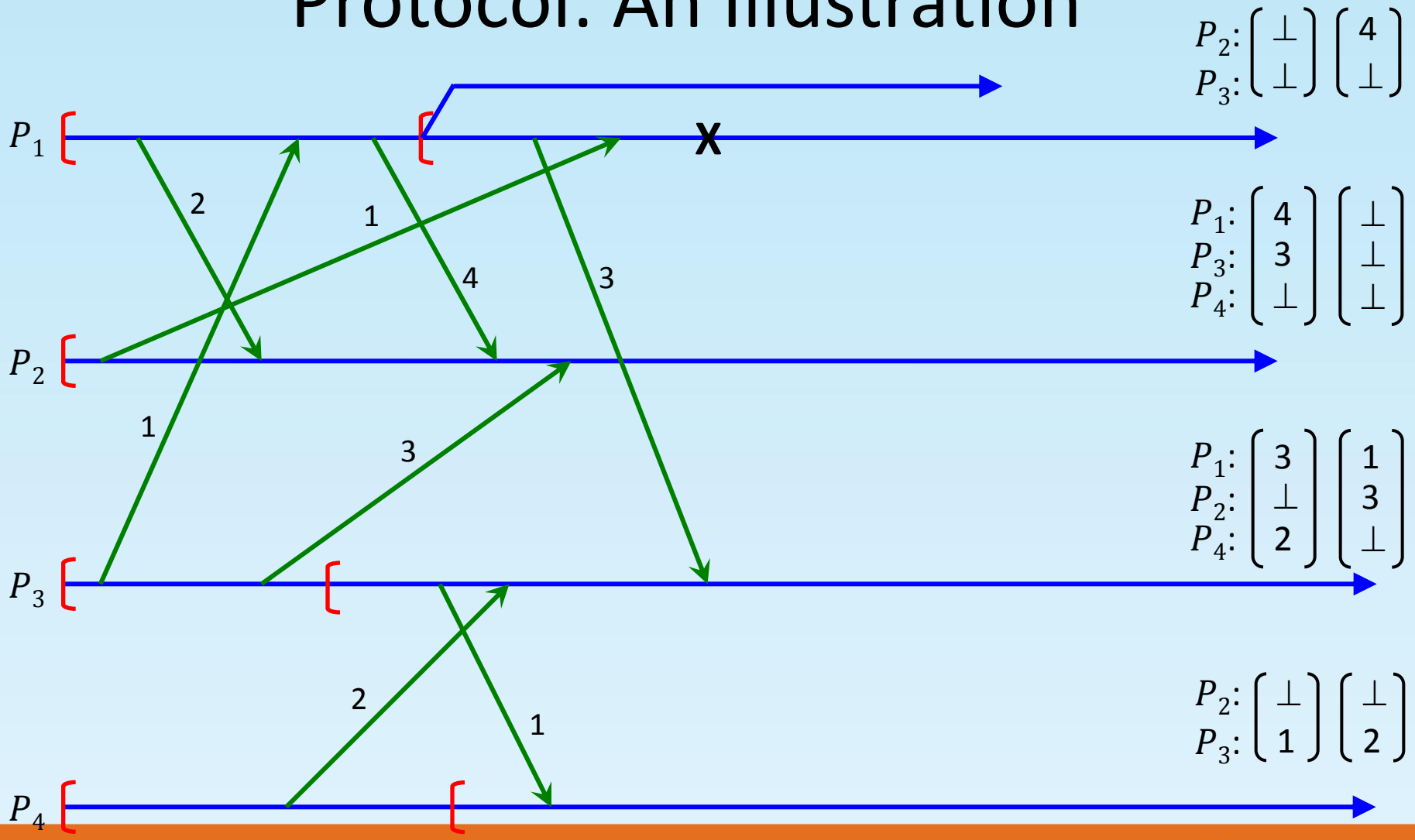
P_2 receives the message from P_4 and processes it

Koo and Toueg's Recovery Protocol: An Illustration



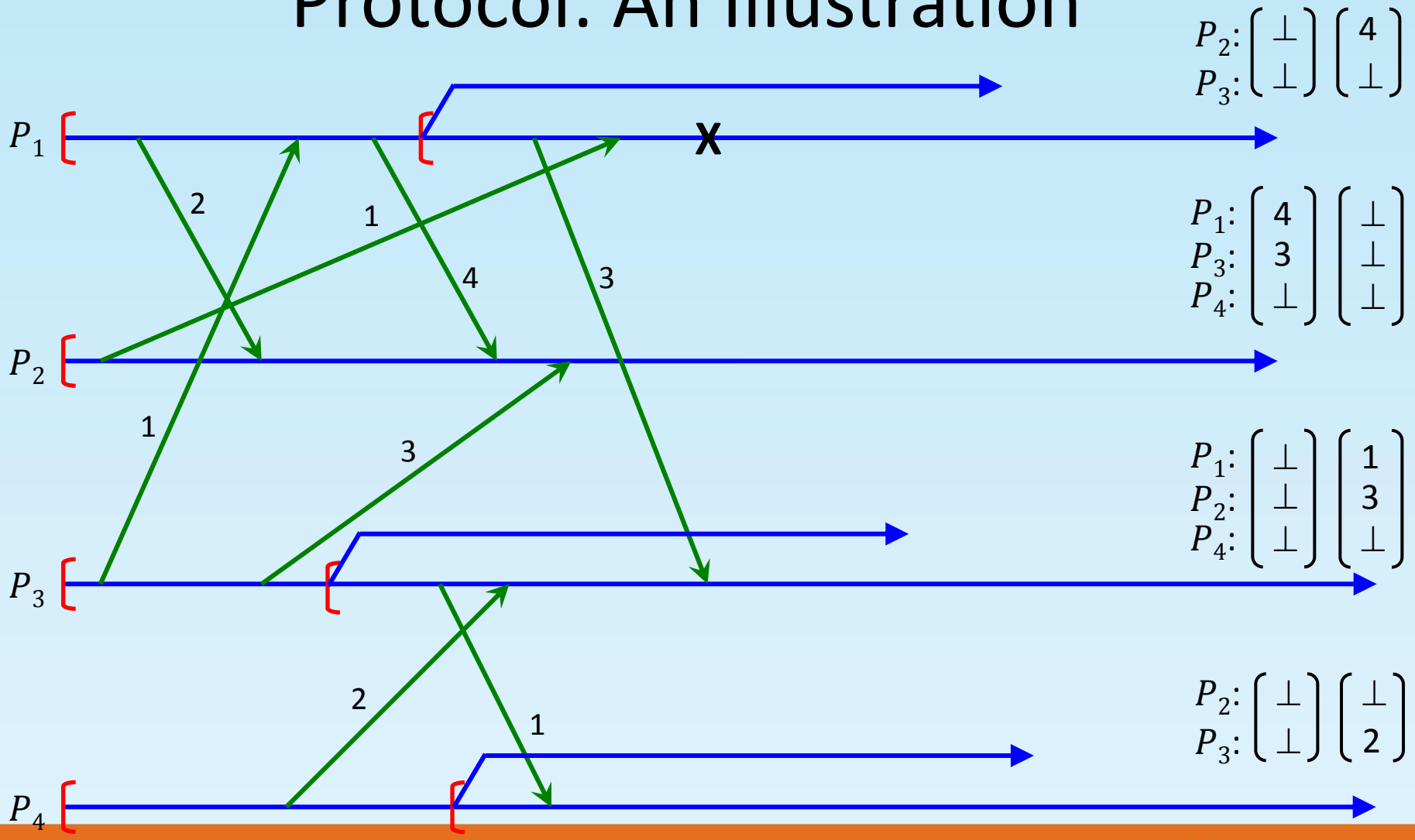
P_2 tests whether $last\_label\_rcvd_2[4] > last\_label\_sent_4[2]$ holds

Koo and Toueg's Recovery Protocol: An Illustration



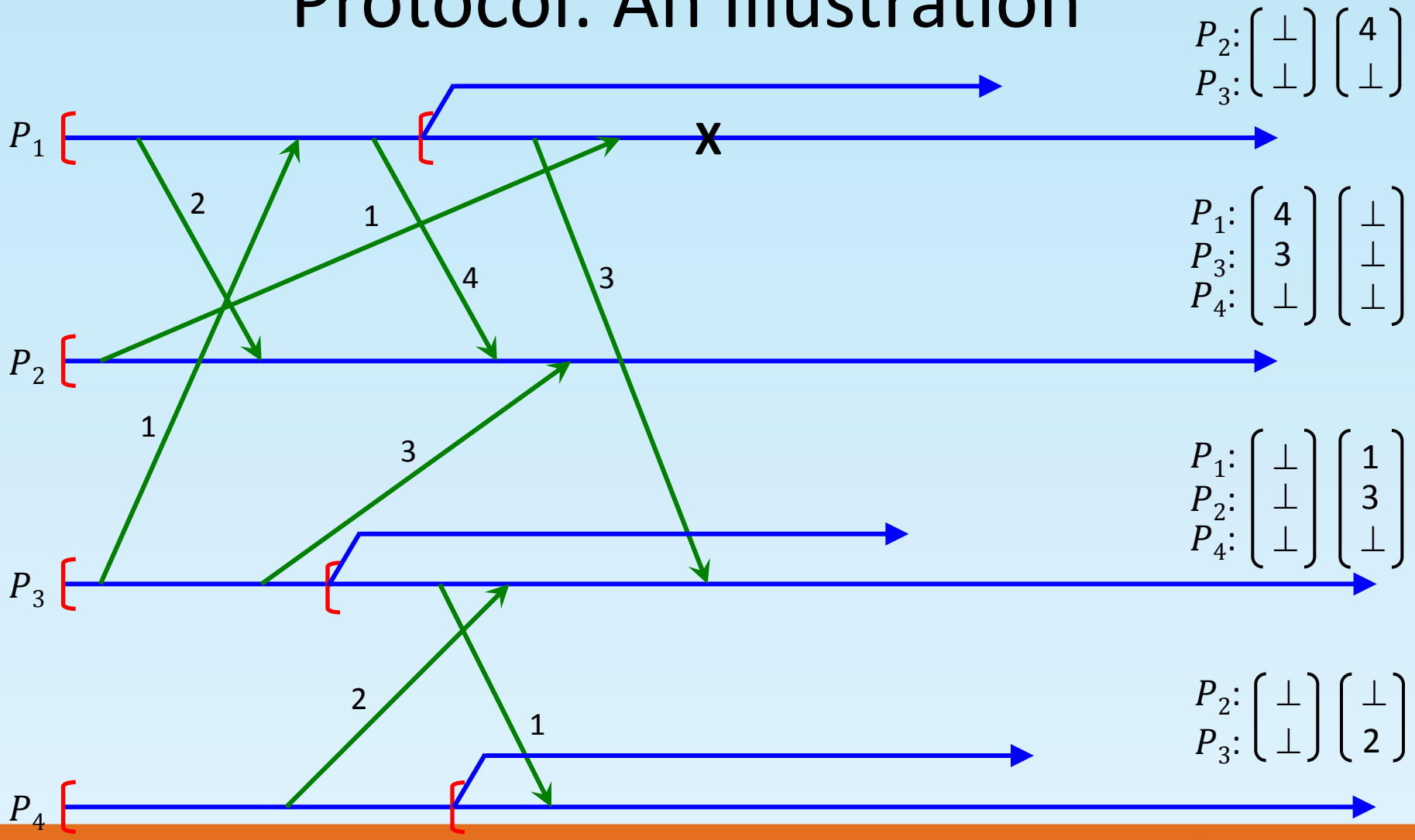
The test fails and P_2 does not have to rollback

Koo and Toueg's Recovery Protocol: An Illustration



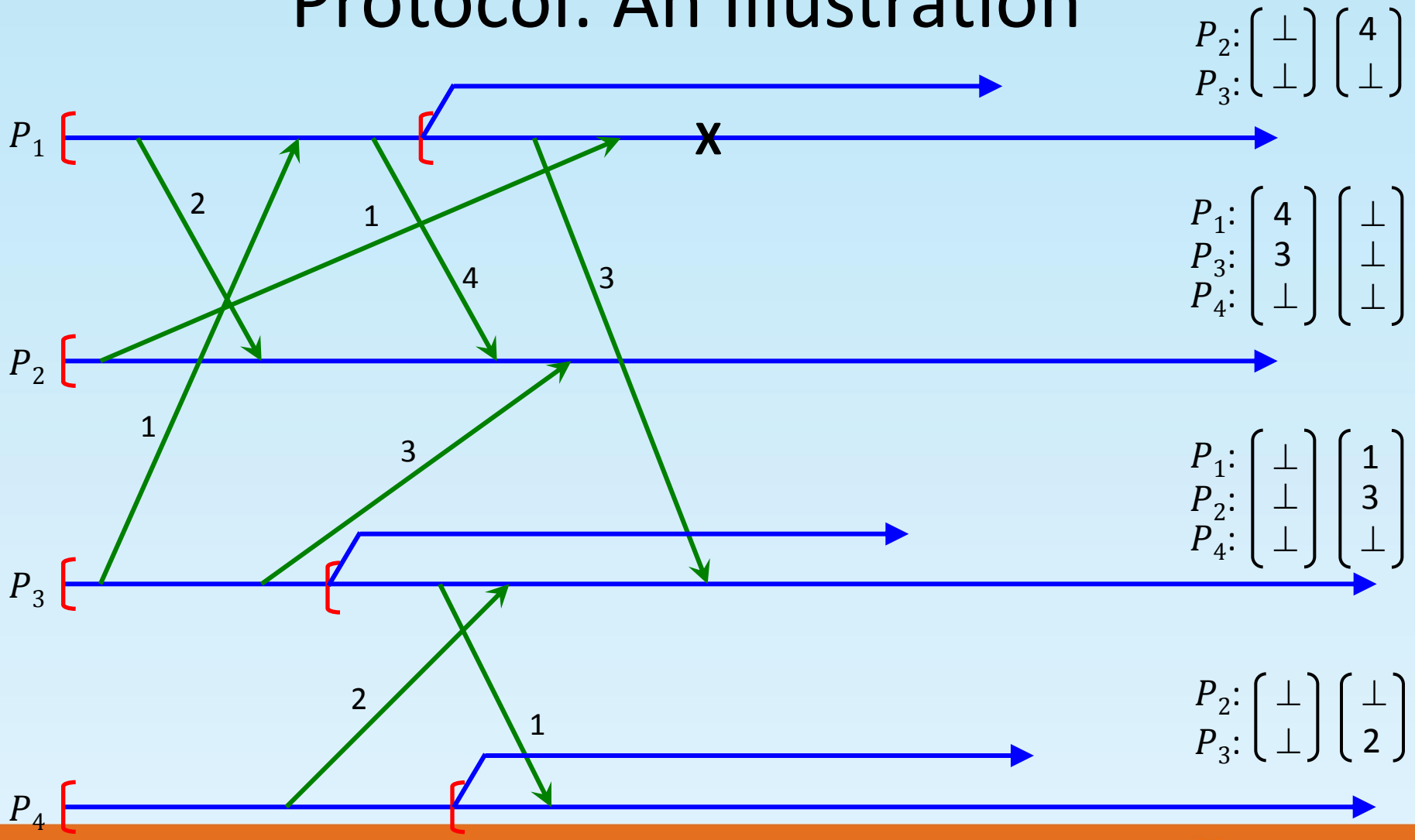
P_3 and P_4 rollback and restart from their last permanent checkpoint

Koo and Toueg's Recovery Protocol: An Illustration



On rolling back, P_3 and P_4 reset all entries in $last\_label\_rcvd$ vectors

Koo and Toueg's Recovery Protocol: An Illustration



Outline

- Overview
- Main Issues
- Checkpointing and Recovery Protocols
  - Koo and Toueg's Protocol
  - Juang and Venkatesan's Protocol

Juang and Venkatesan's Checkpointing Protocol

- A process takes a checkpoint every time it executes a communication event
  - Checkpoints are taken in volatile storage
  - Periodically checkpoints in volatile storage are flushed to stable storage
  - A process loses checkpoints in its volatile storage when it fails
  - Other processes (that did not fail) still have access to checkpoints in volatile storage

Juang and Venkatesan's Recovery Protocol

- Each process maintains two vectors with one entry for every neighbor:
  - $SENT$: stores the number of messages sent to each neighbor so far
  - $RCVD$: stores the number of messages received from each neighbor so far

Juang and Venkatesan's Recovery Protocol (Contd.)

- Consider a collection of checkpoints, one from each process $P_1, P_2, \ldots, PN$, given by:

$$ckpt_1, ckpt_2, \ldots, ckpt_N$$

- Checkpoints form a consistent global state if, for each pair of neighbors P_i and P_j, the following holds:

$$SENT(ckpt_i)[j] \geq RCVD(ckpt_j)[i]$$

- Otherwise, P_j's state is inconsistent with that of P_i
  - The inconsistency can be removed by rolling back P_j

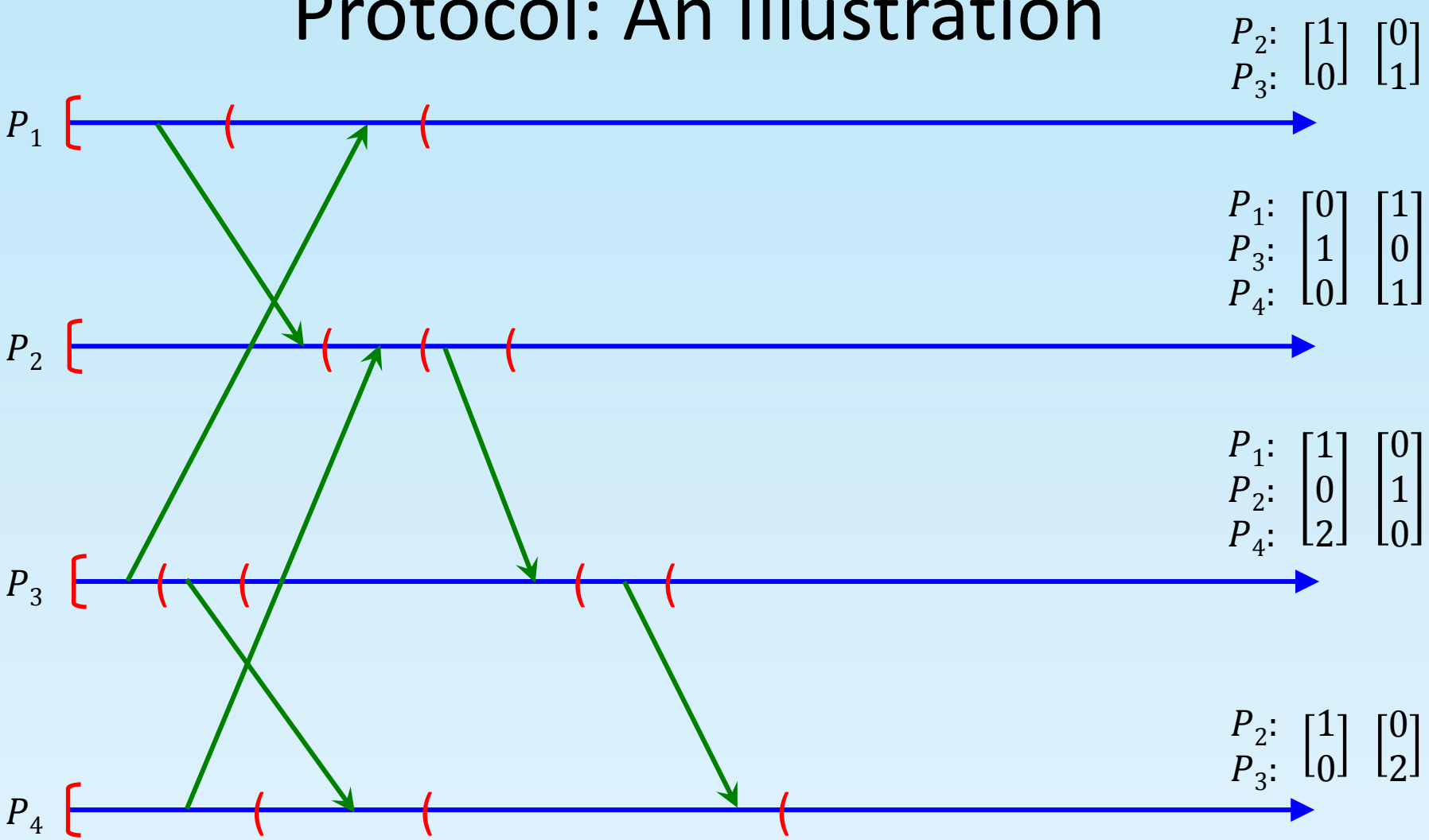
Juang and Venkatesan's Recovery Protocol (Contd.)

- All processes participate in recovery
  - Failed process, on restarting, rolls back to its last stable checkpoint and instructs all processes to start recovery using flooding
- Recovery protocol executes in iterations
- In each iteration, every process sends to each of its neighbors the number of messages it has sent to it as per the current state
  - A process rolls back if its state is inconsistent with that of its neighbor. It rolls back to its latest checkpoint (volatile or stable) that removes the inconsistency

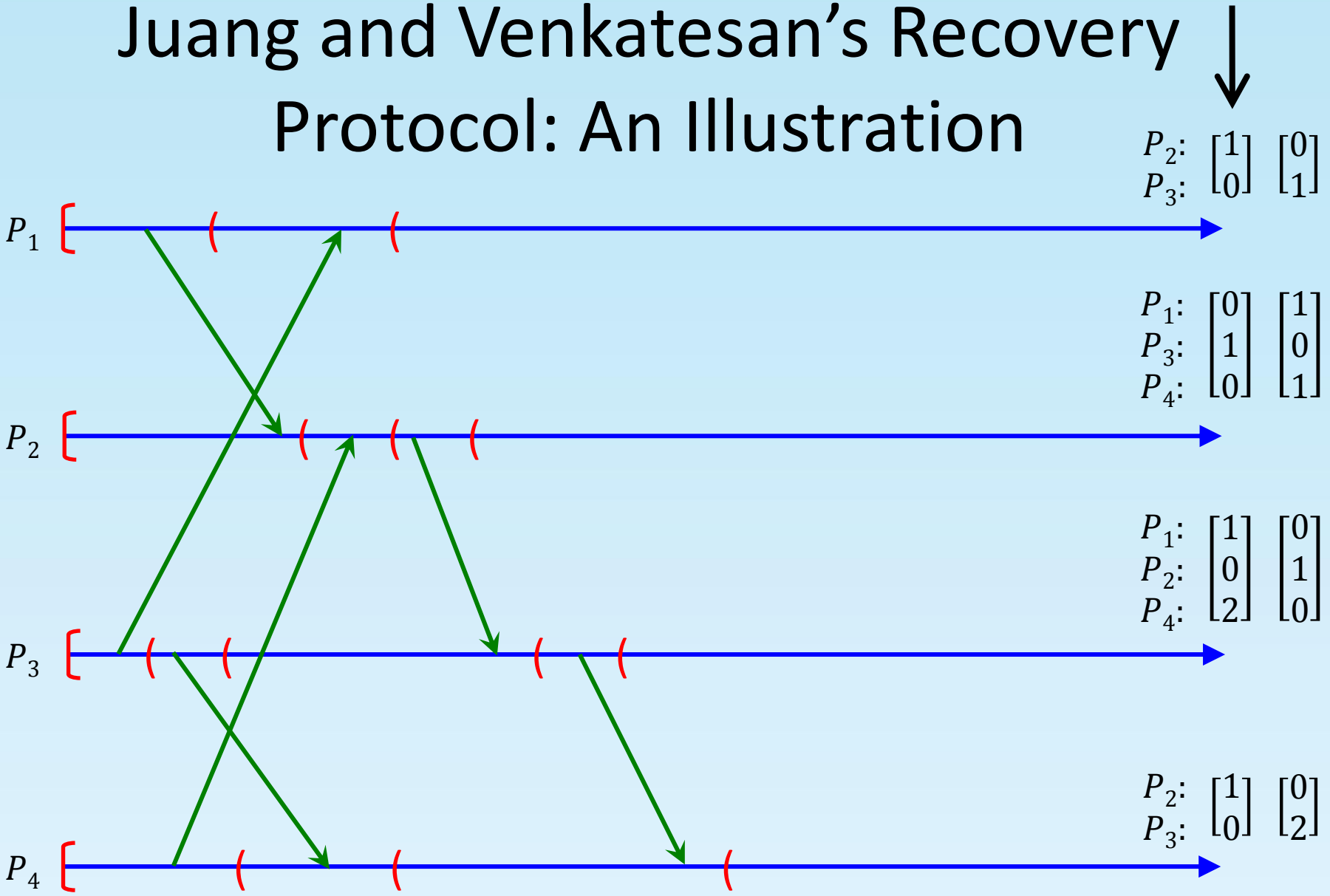
Juang and Venkatesan's Recovery Protocol (Contd.)

- Let N denote the number of processes in the system
  - The system is guaranteed to be in a consistent state after $N - 1$ iterations

Juang and Venkatesan's Recovery Protocol: An Illustration

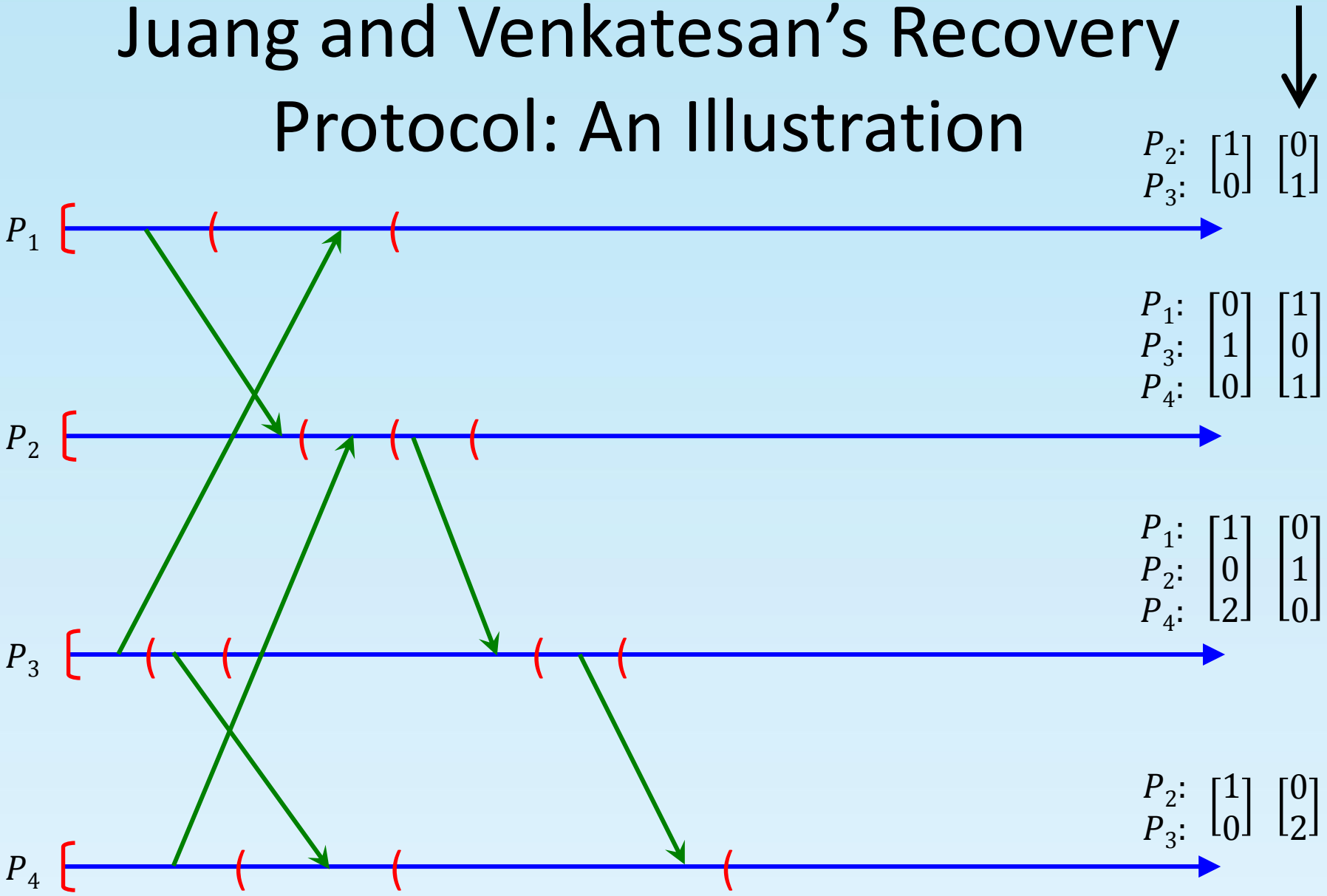


Juang and Venkatesan's Recovery Protocol: An Illustration



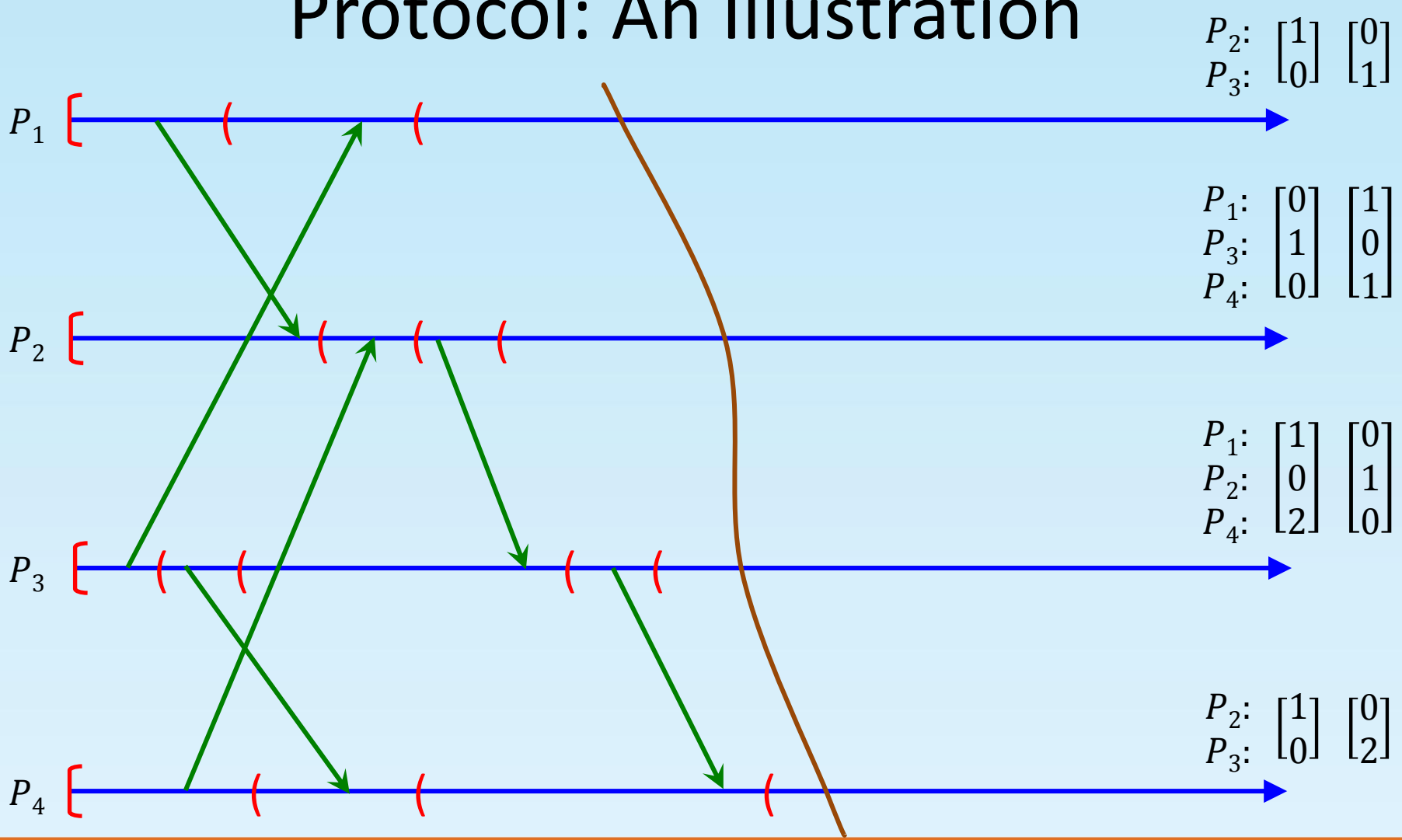
First vector: $SENT$

Juang and Venkatesan's Recovery Protocol: An Illustration



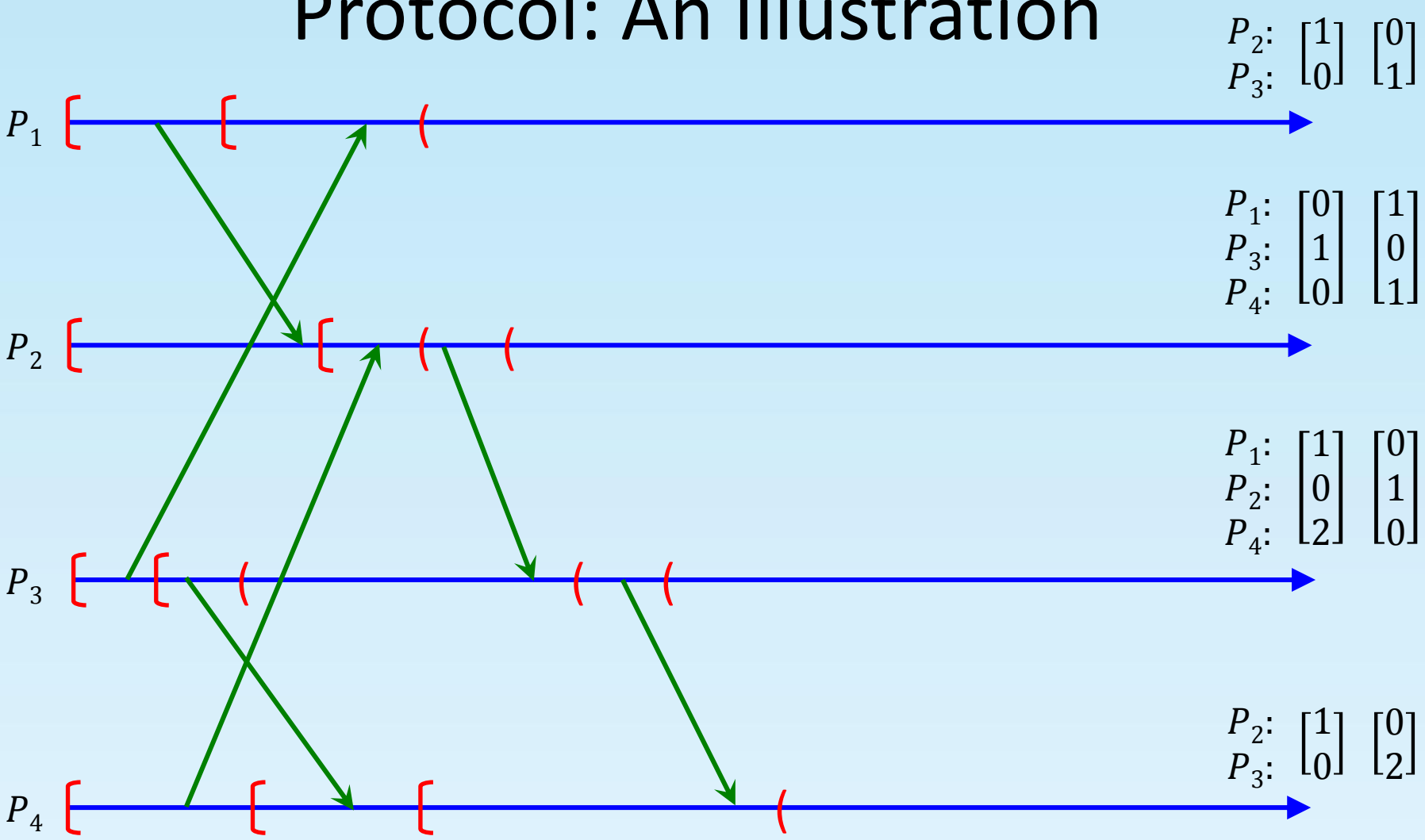
Second vector: $RCVD$

Juang and Venkatesan's Recovery Protocol: An Illustration



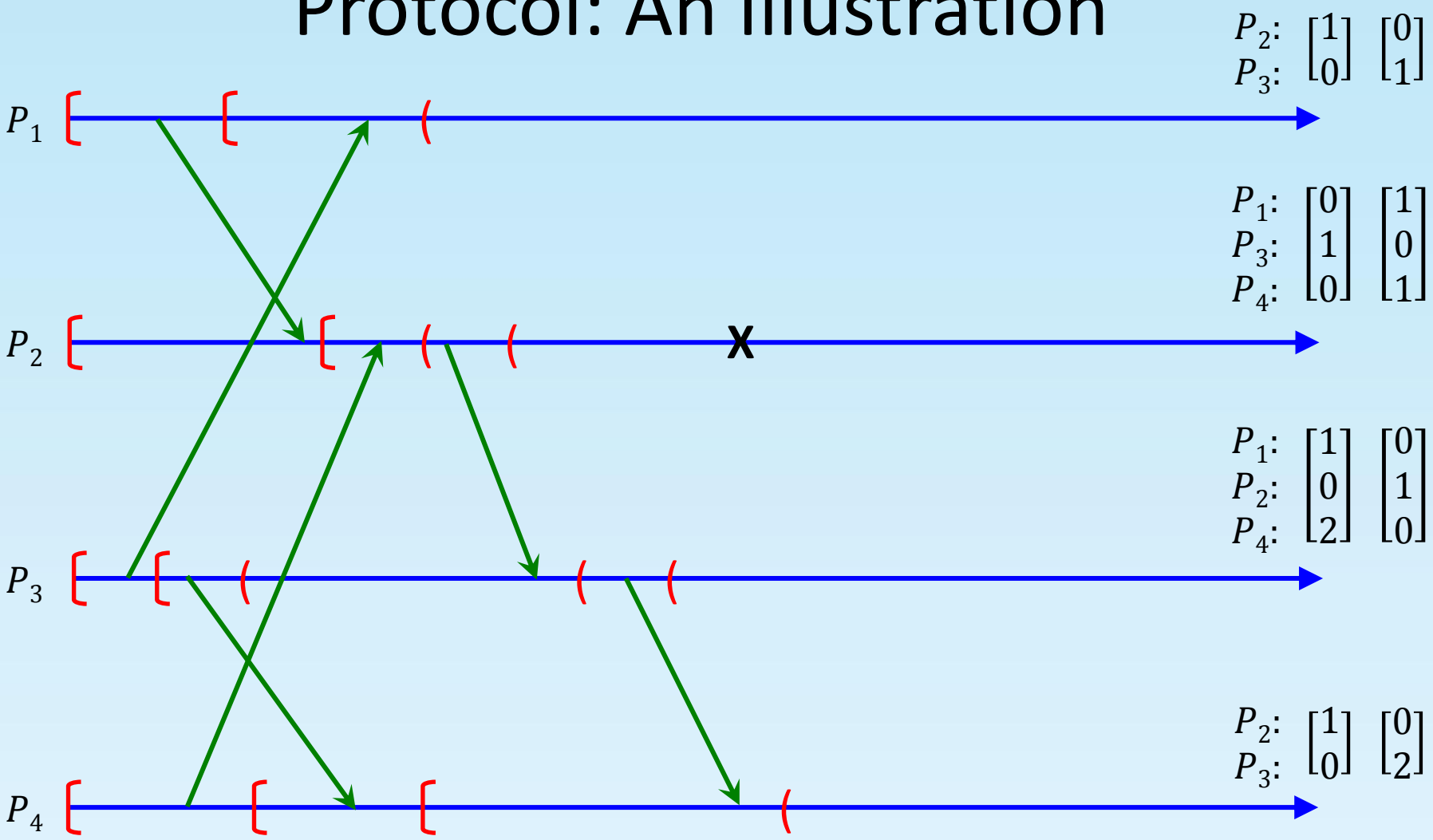
Current system state

Juang and Venkatesan's Recovery Protocol: An Illustration



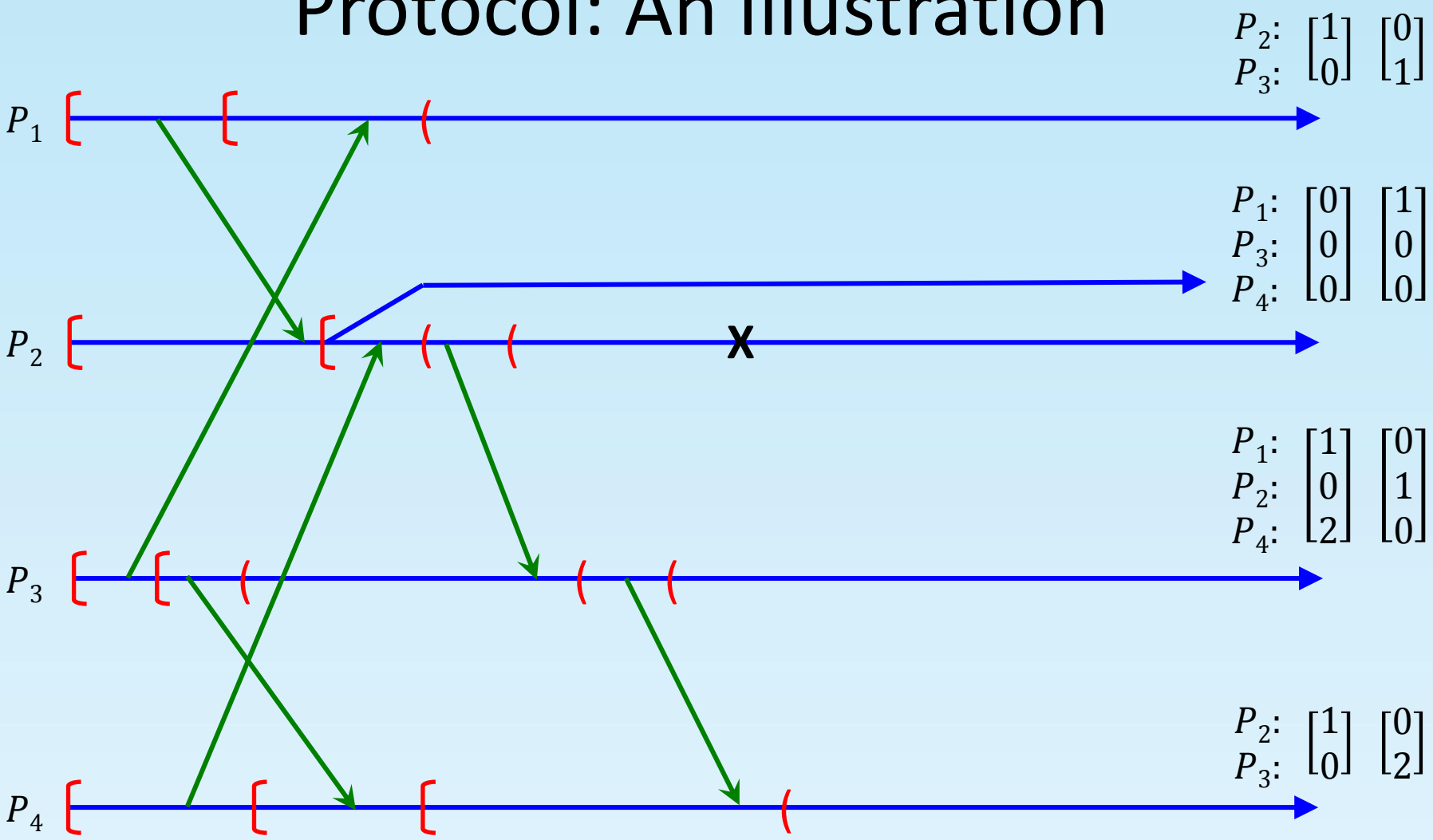
Some volatile checkpoints become permanent

Juang and Venkatesan's Recovery Protocol: An Illustration



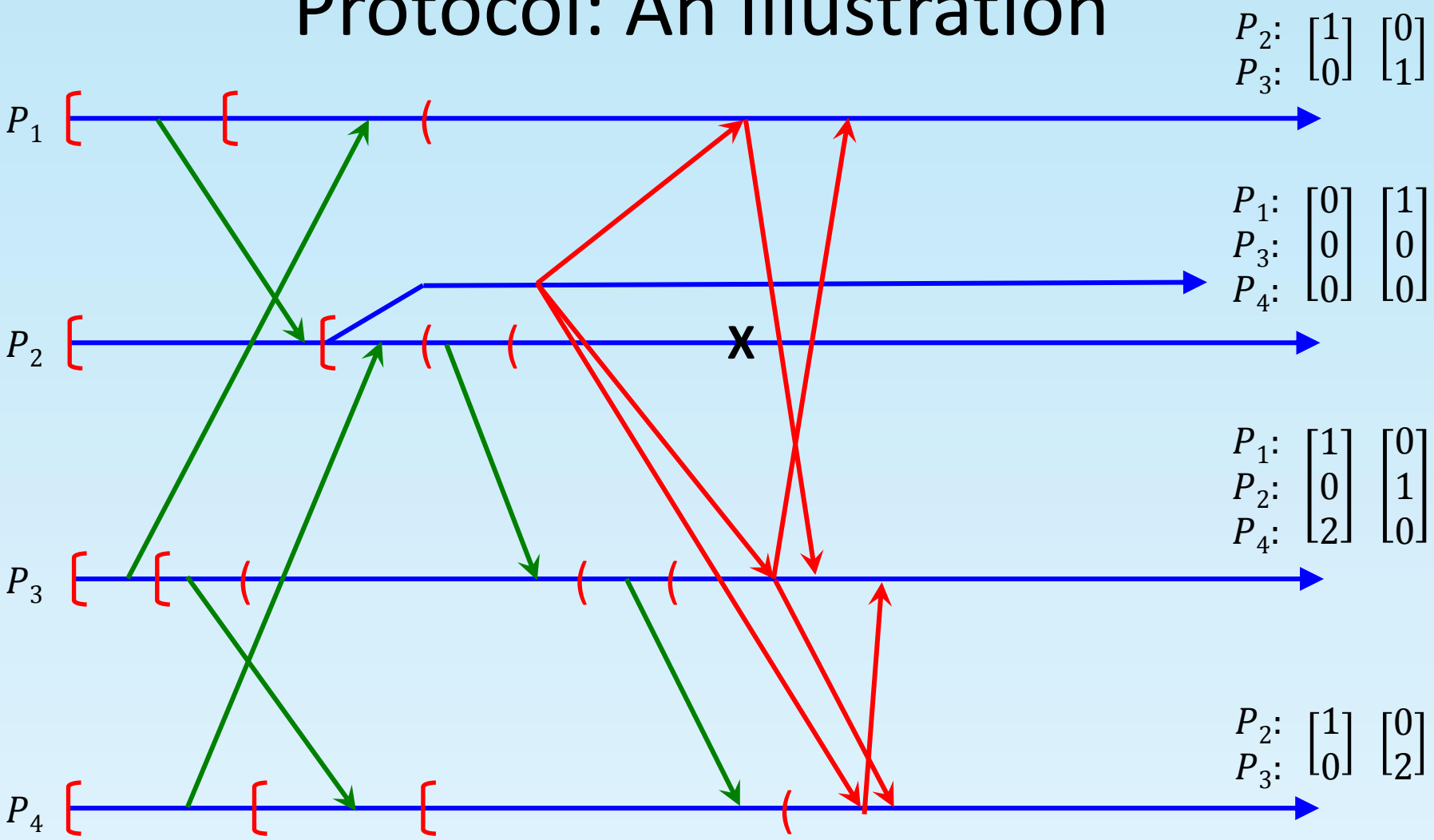
P_2 crashes

Juang and Venkatesan's Recovery Protocol: An Illustration



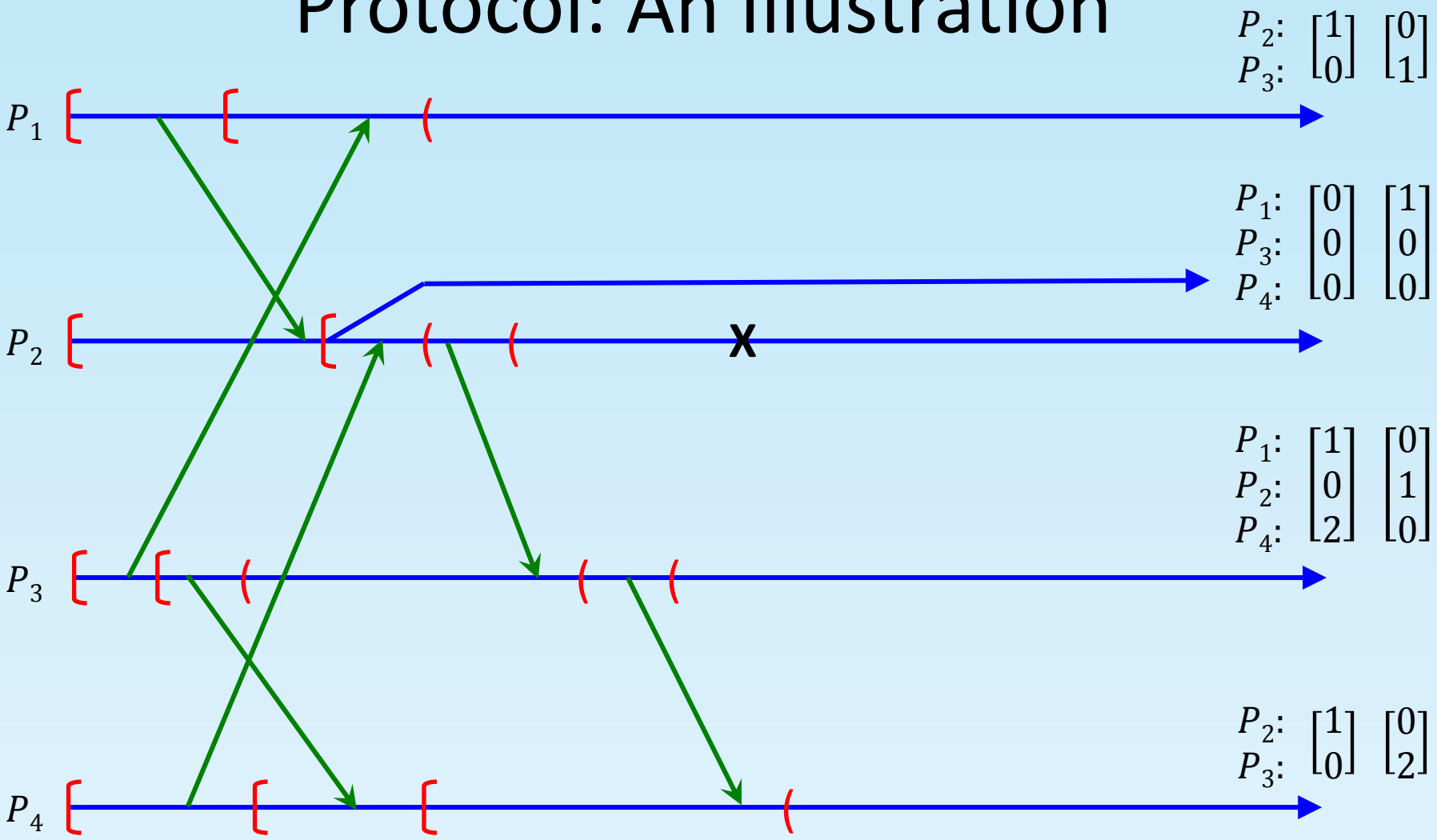
P_2 restarts from its last stable checkpoint

Juang and Venkatesan's Recovery Protocol: An Illustration



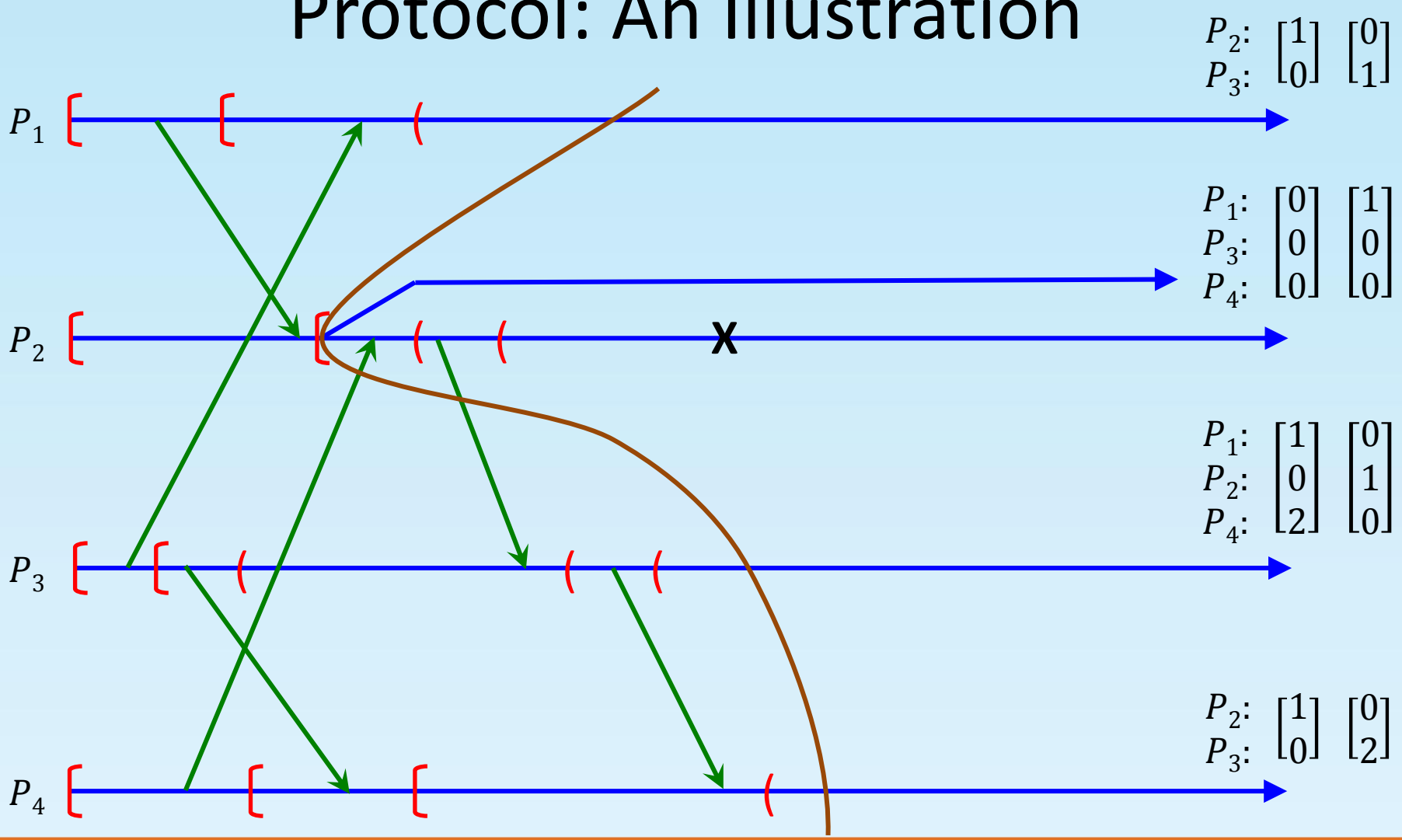
P_2 informs all processes of its failure using flooding

Juang and Venkatesan's Recovery Protocol: An Illustration



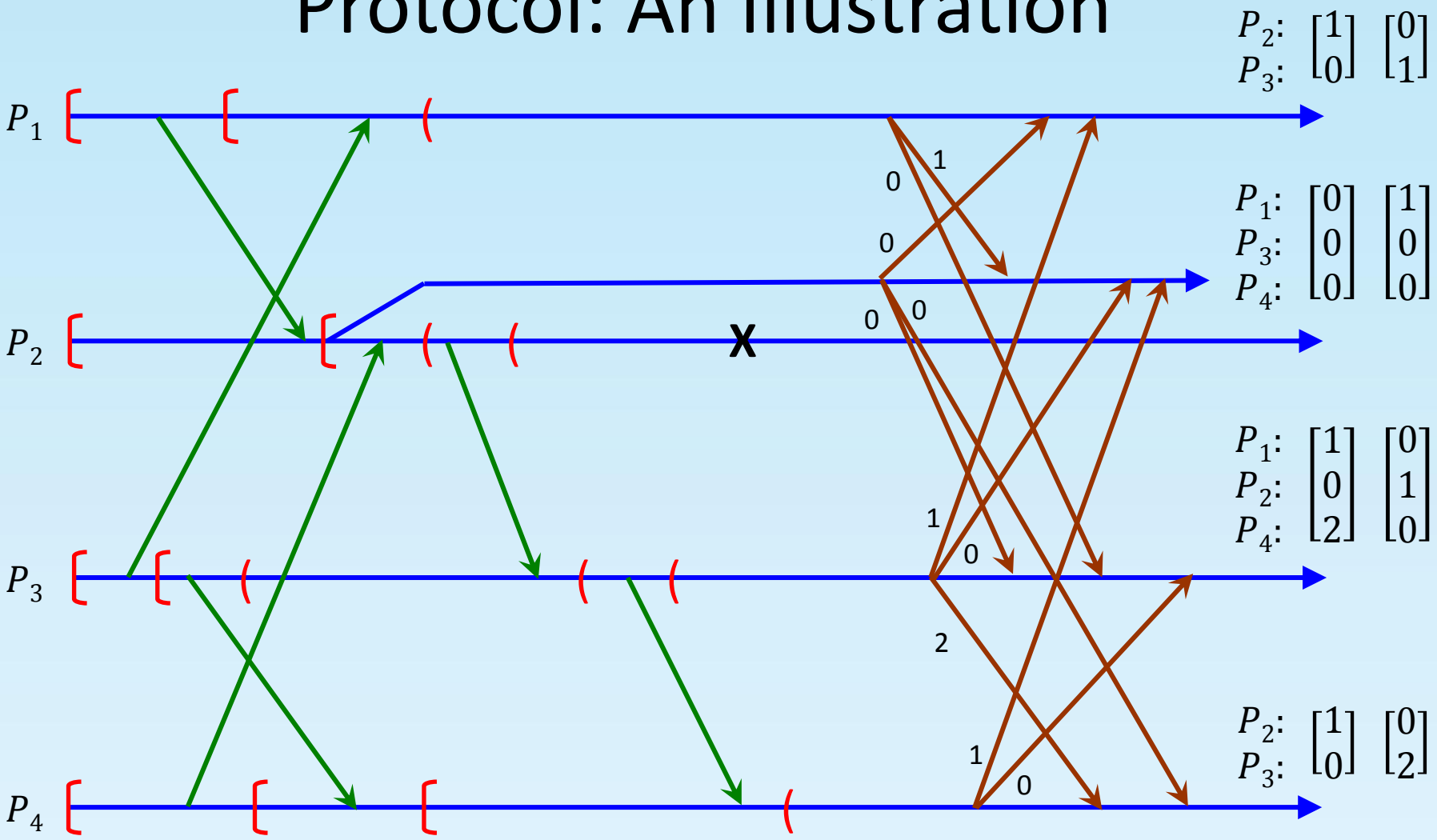
All processes enter recovery mode

Juang and Venkatesan's Recovery Protocol: An Illustration



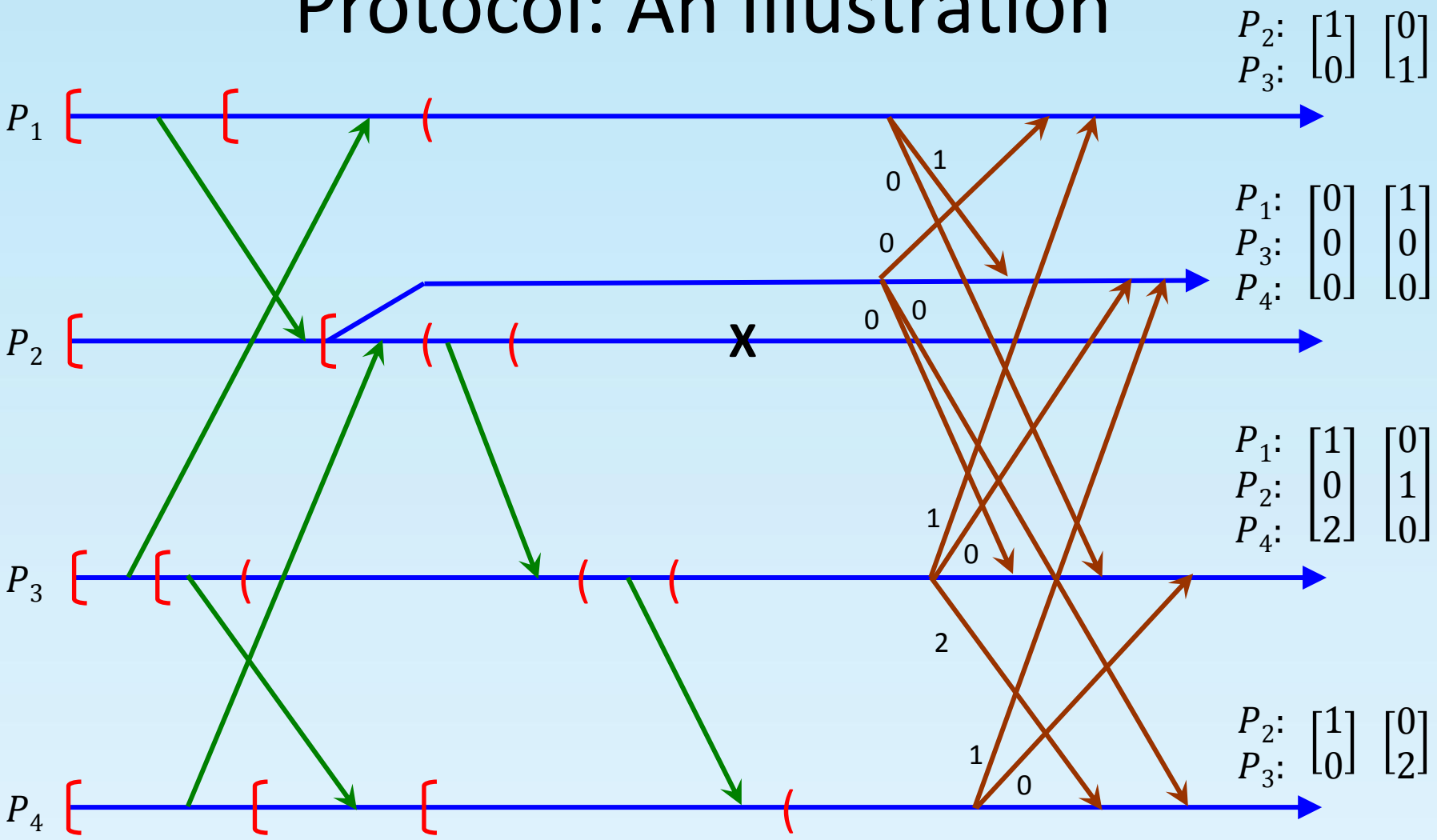
System state at the beginning of iterations; it is an inconsistent global state

Juang and Venkatesan's Recovery Protocol: An Illustration



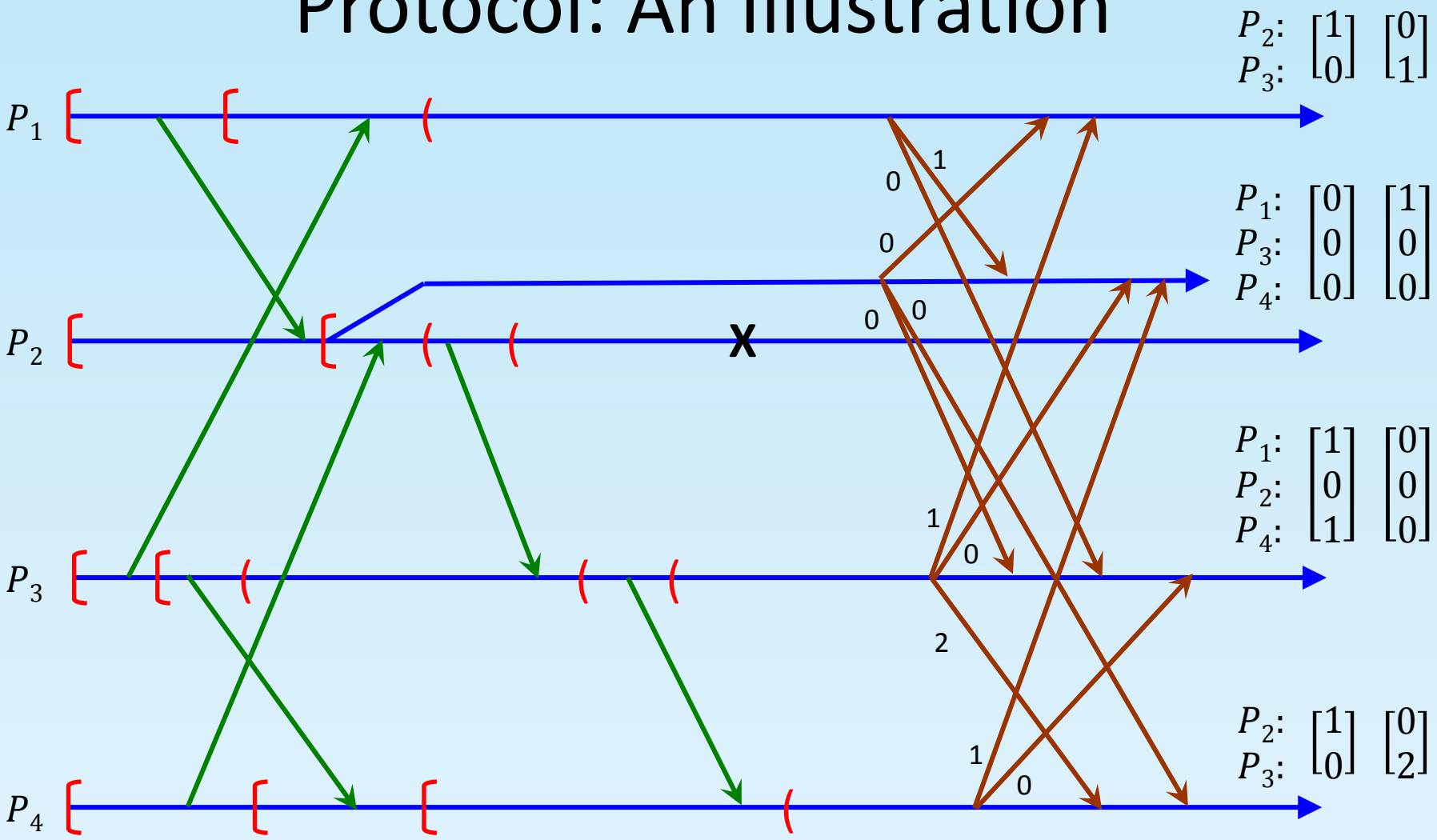
Iteration 1: every process sends entries of $SENT$ vector to its neighbors

Juang and Venkatesan's Recovery Protocol: An Illustration



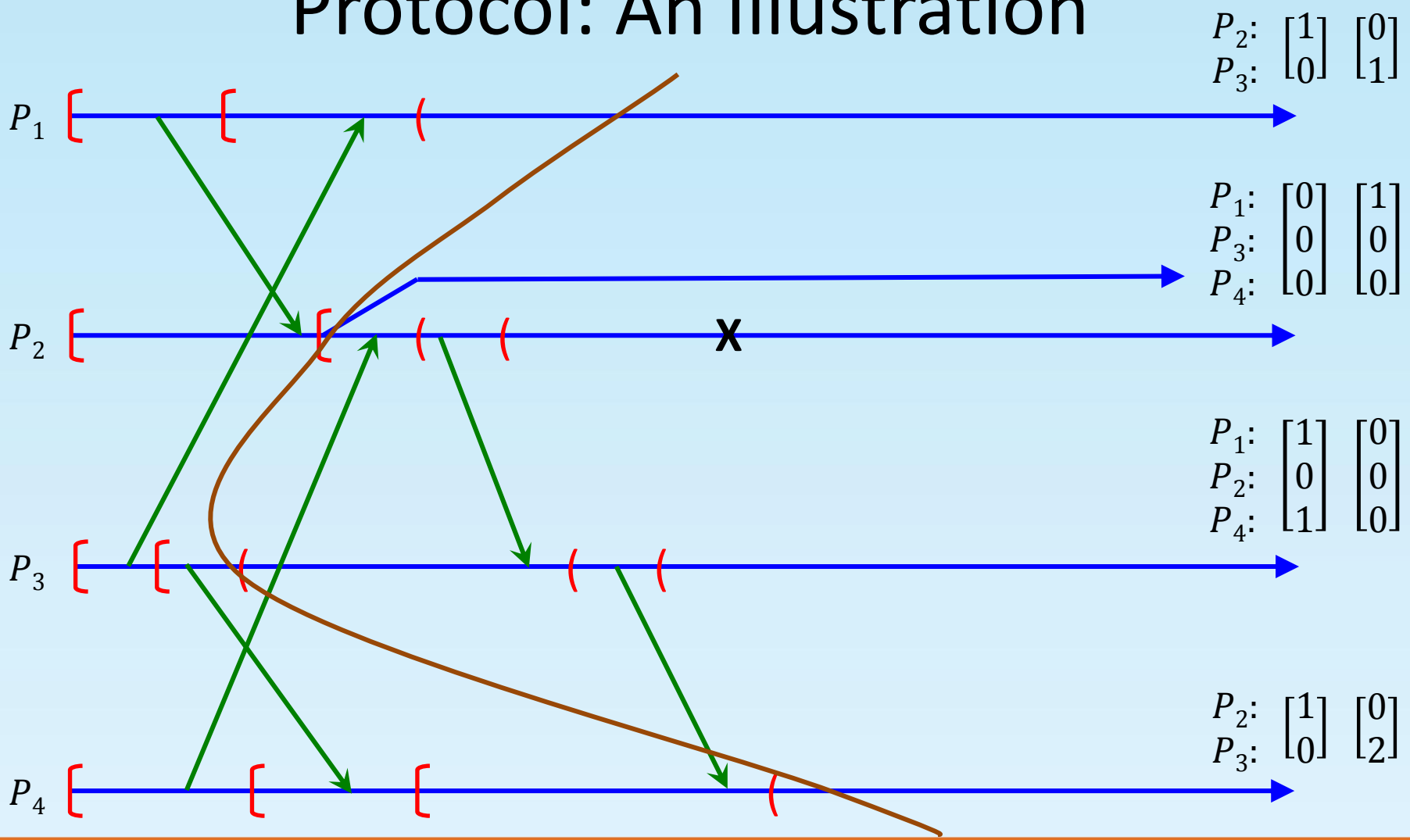
Each process checks if its current state is inconsistent with that of its neighbors

Juang and Venkatesan's Recovery Protocol: An Illustration



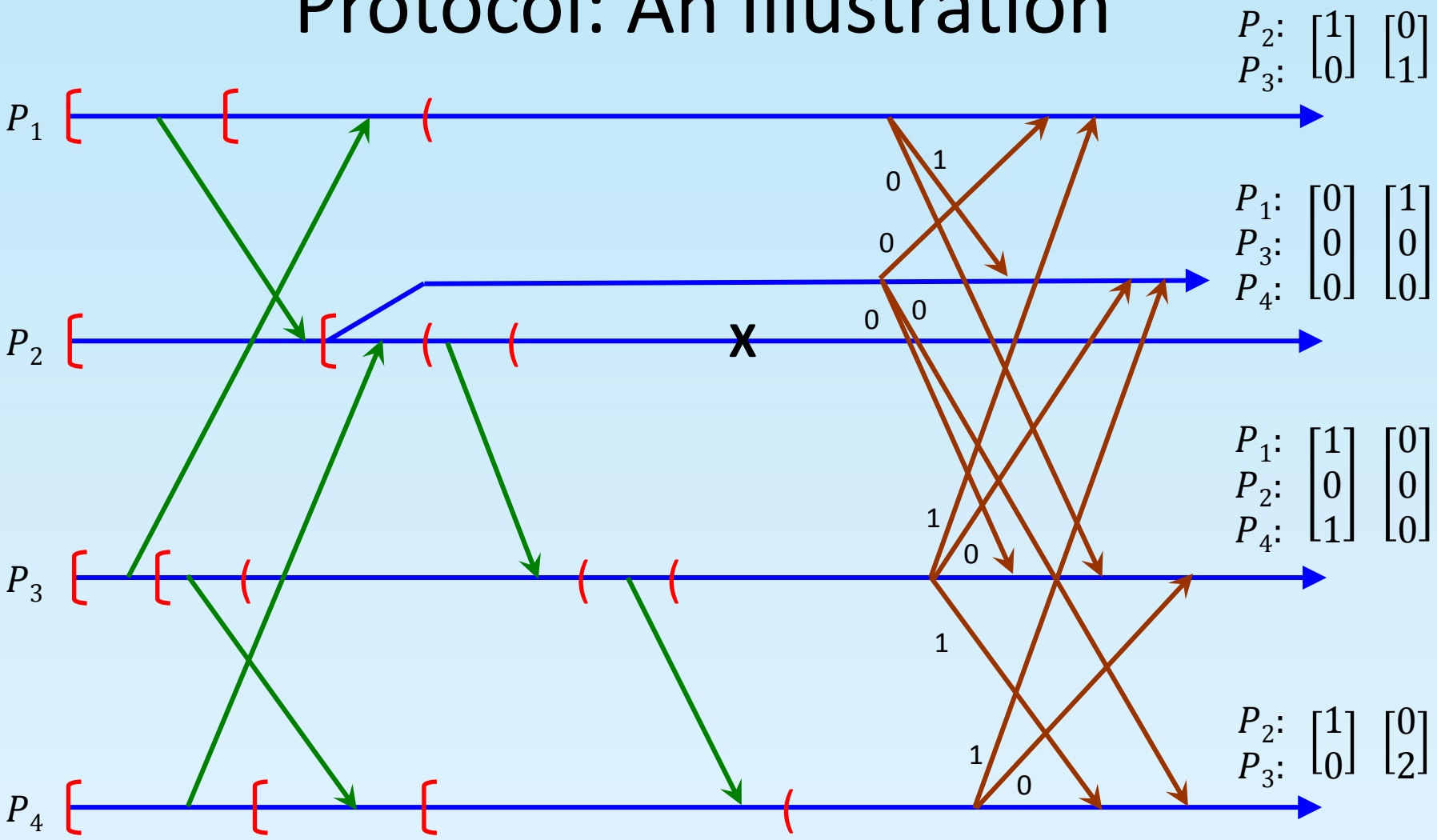
P_3 rolls back; P_1, P_2 and P_4 do not need to rollback

Juang and Venkatesan's Recovery Protocol: An Illustration



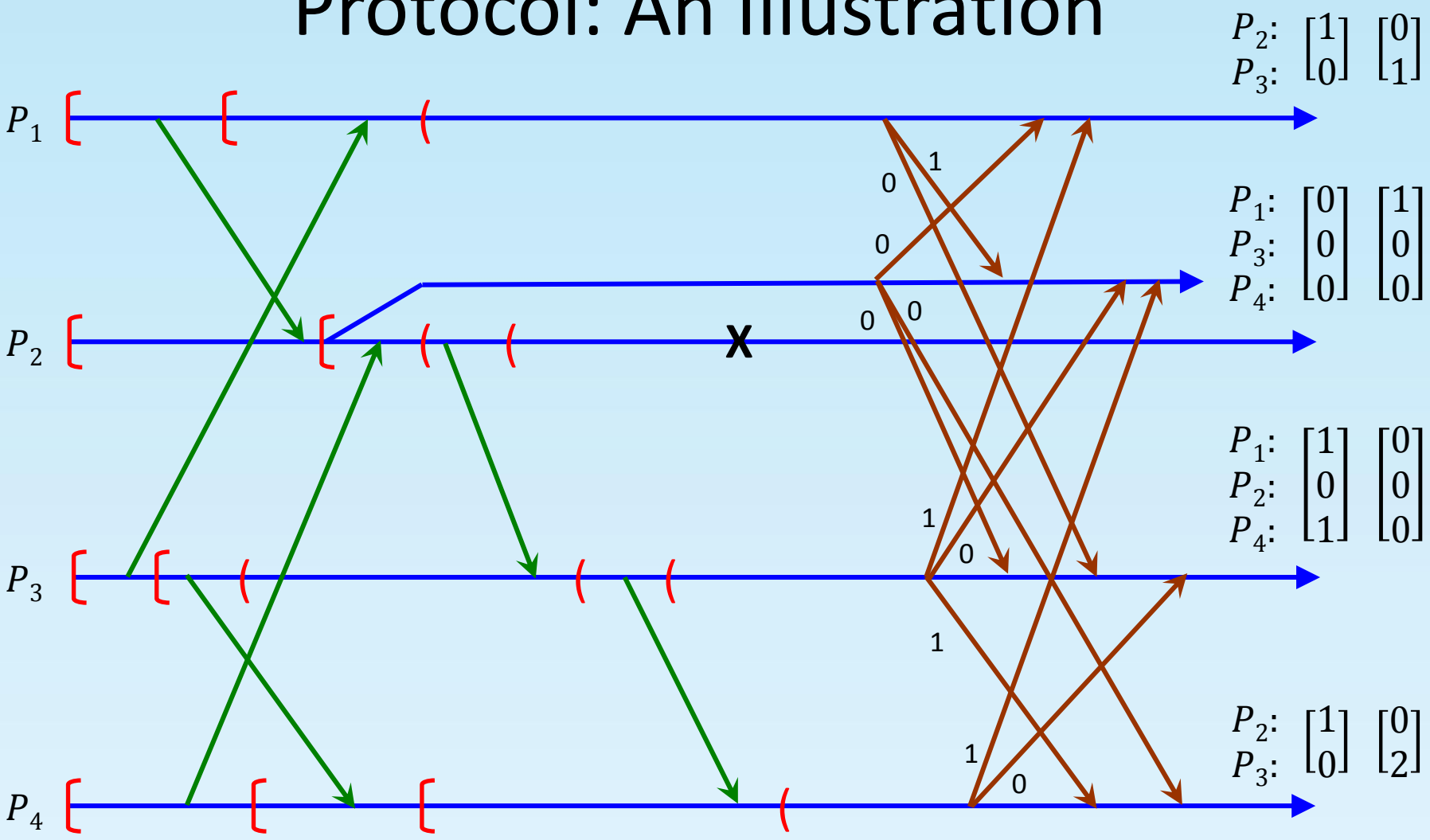
System state at the end of iteration 1

Juang and Venkatesan's Recovery Protocol: An Illustration



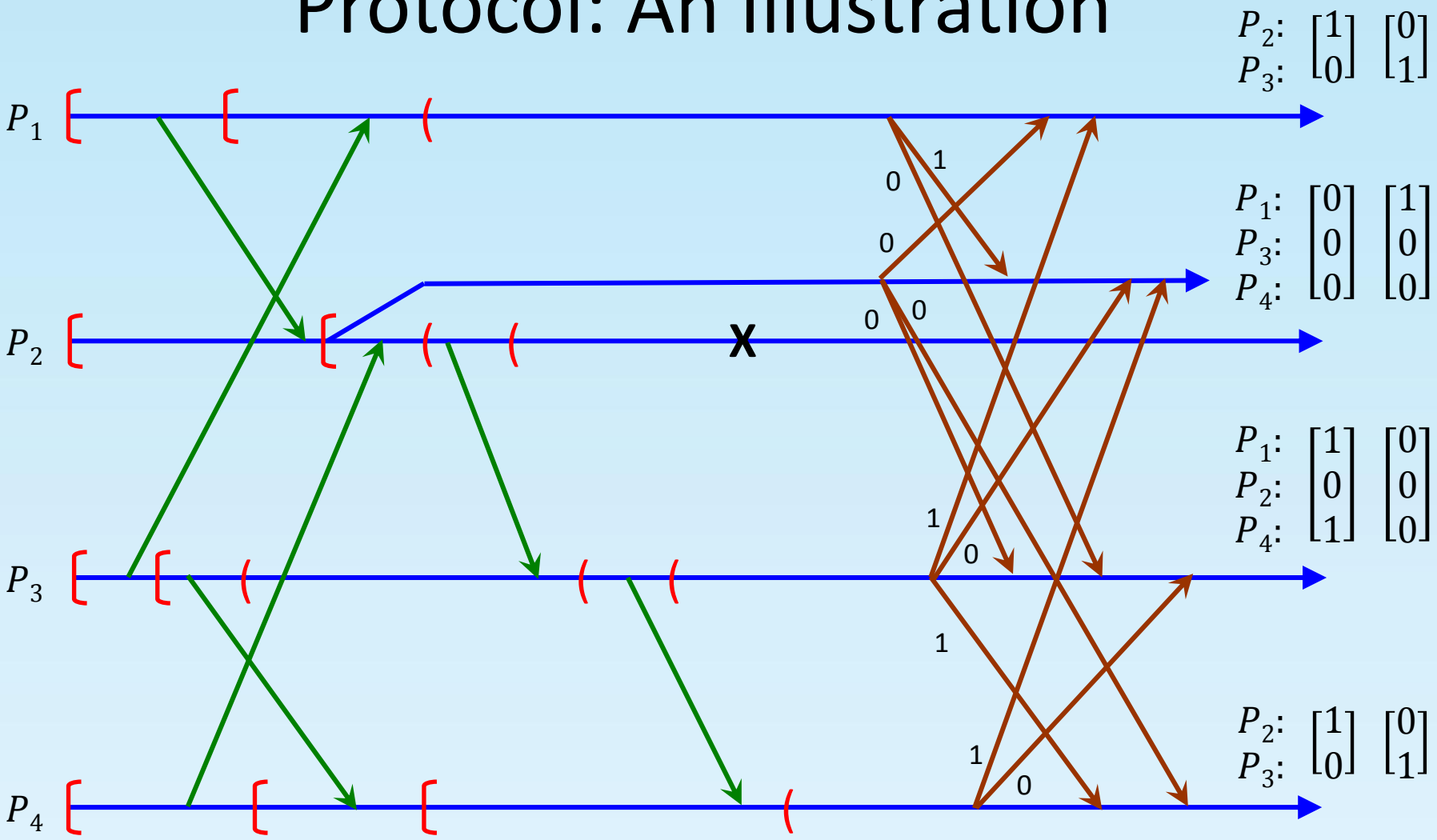
Iteration 2: every process sends entries of $SENT$ vector to its neighbors

Juang and Venkatesan's Recovery Protocol: An Illustration



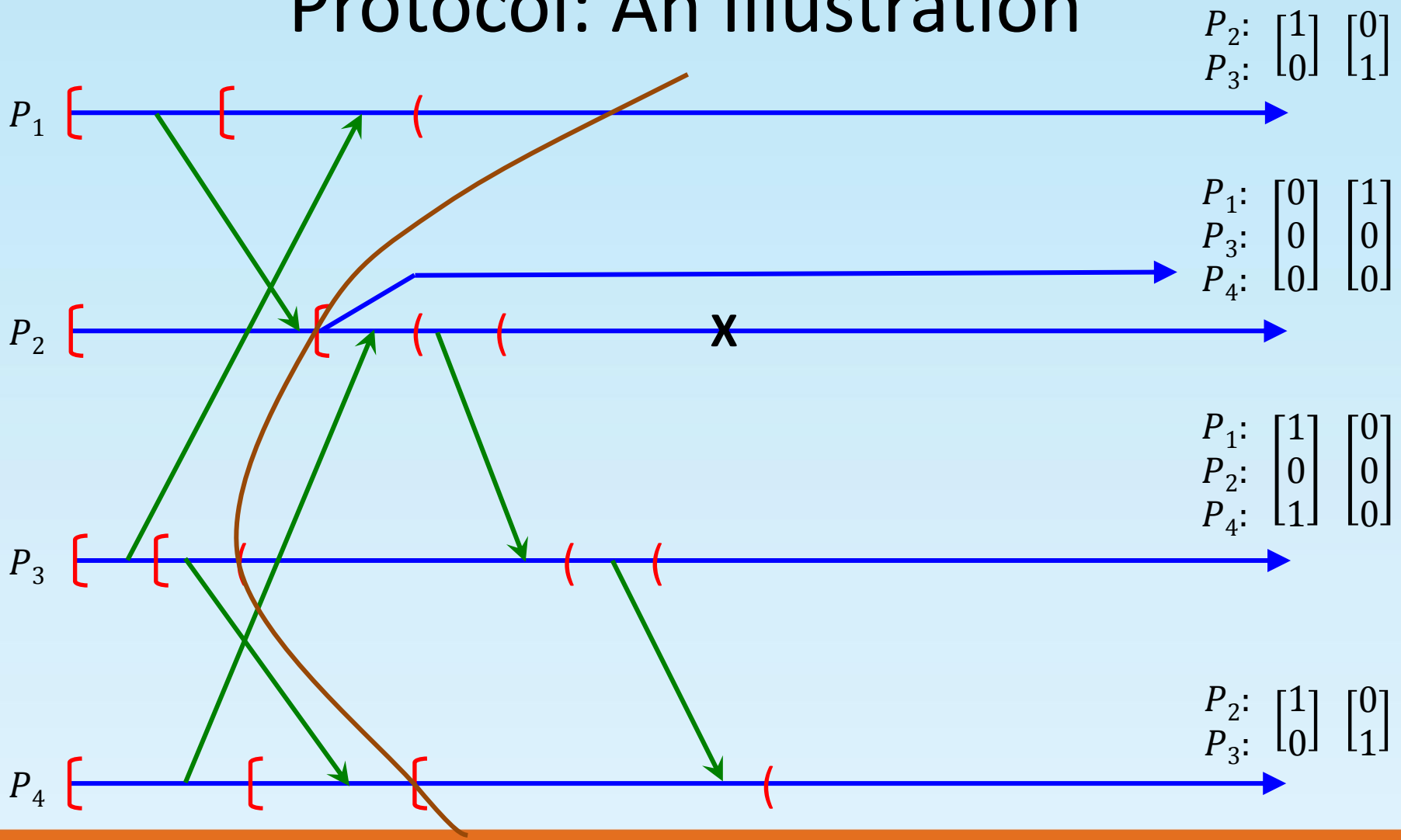
Each process checks if its current state is inconsistent with that of its neighbors

Juang and Venkatesan's Recovery Protocol: An Illustration



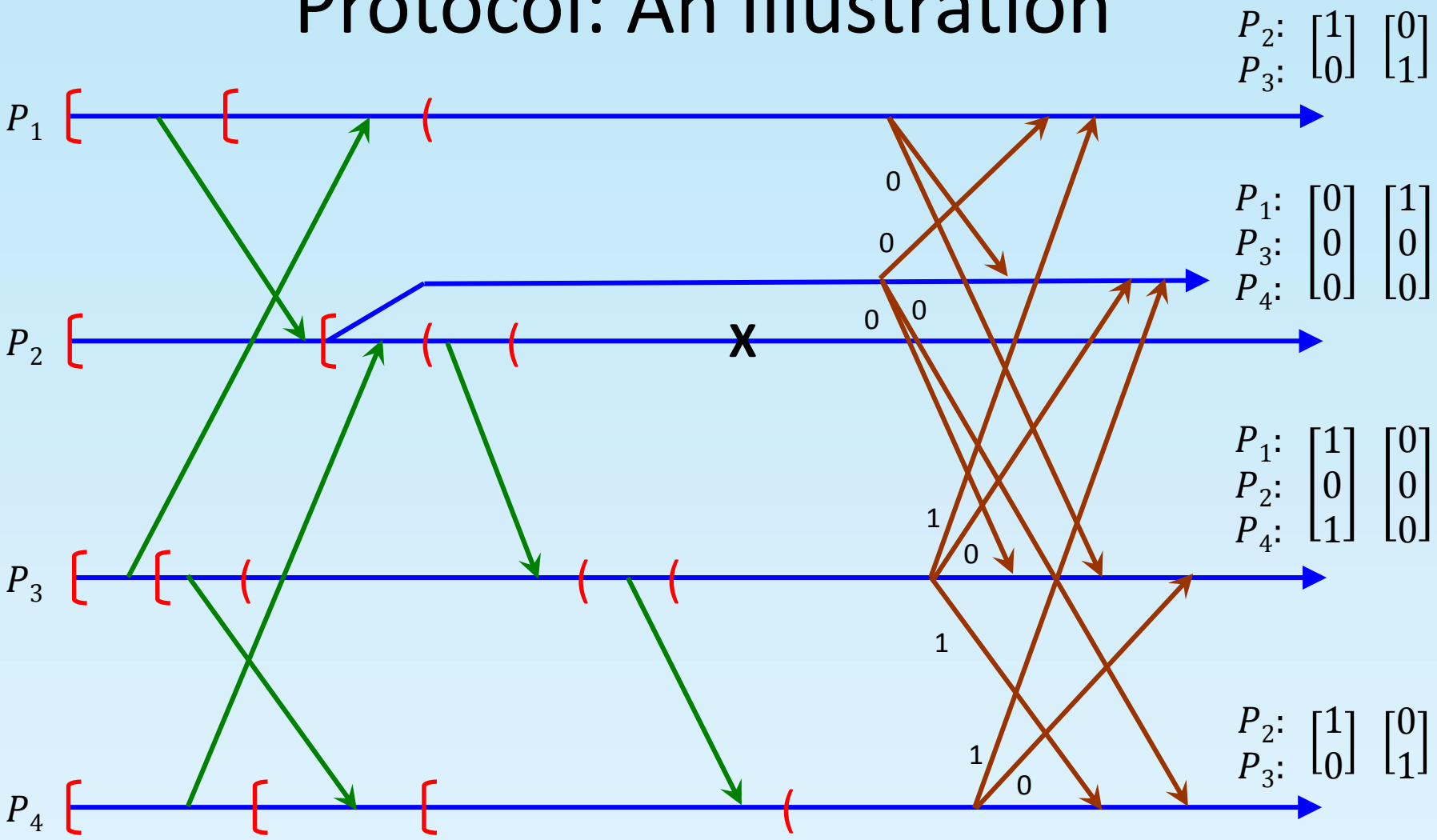
P_4 rolls back; P_1, P_2 and P_3 do not need to rollback

Juang and Venkatesan's Recovery Protocol: An Illustration



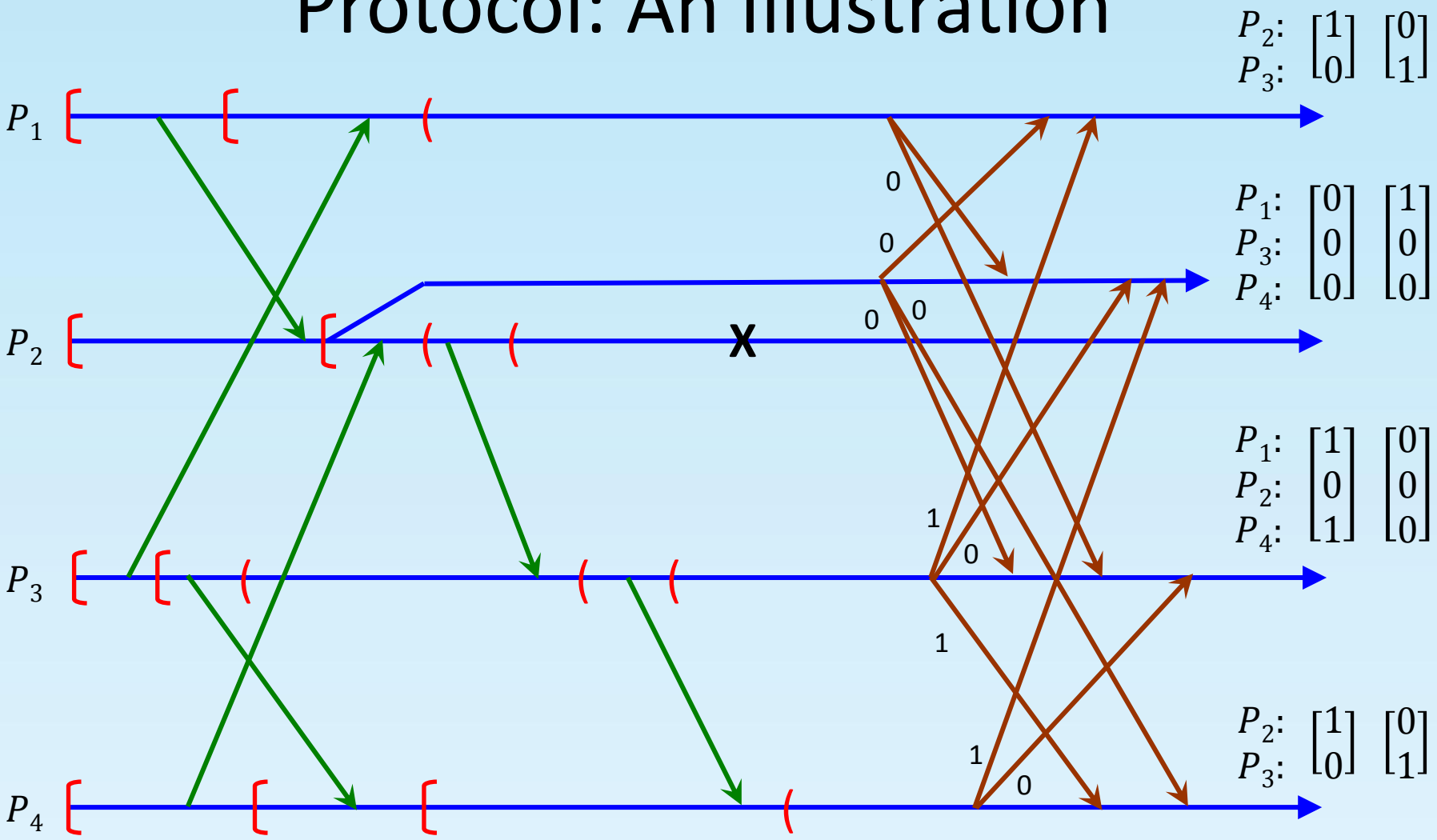
System state at the end of iteration 2

Juang and Venkatesan's Recovery Protocol: An Illustration



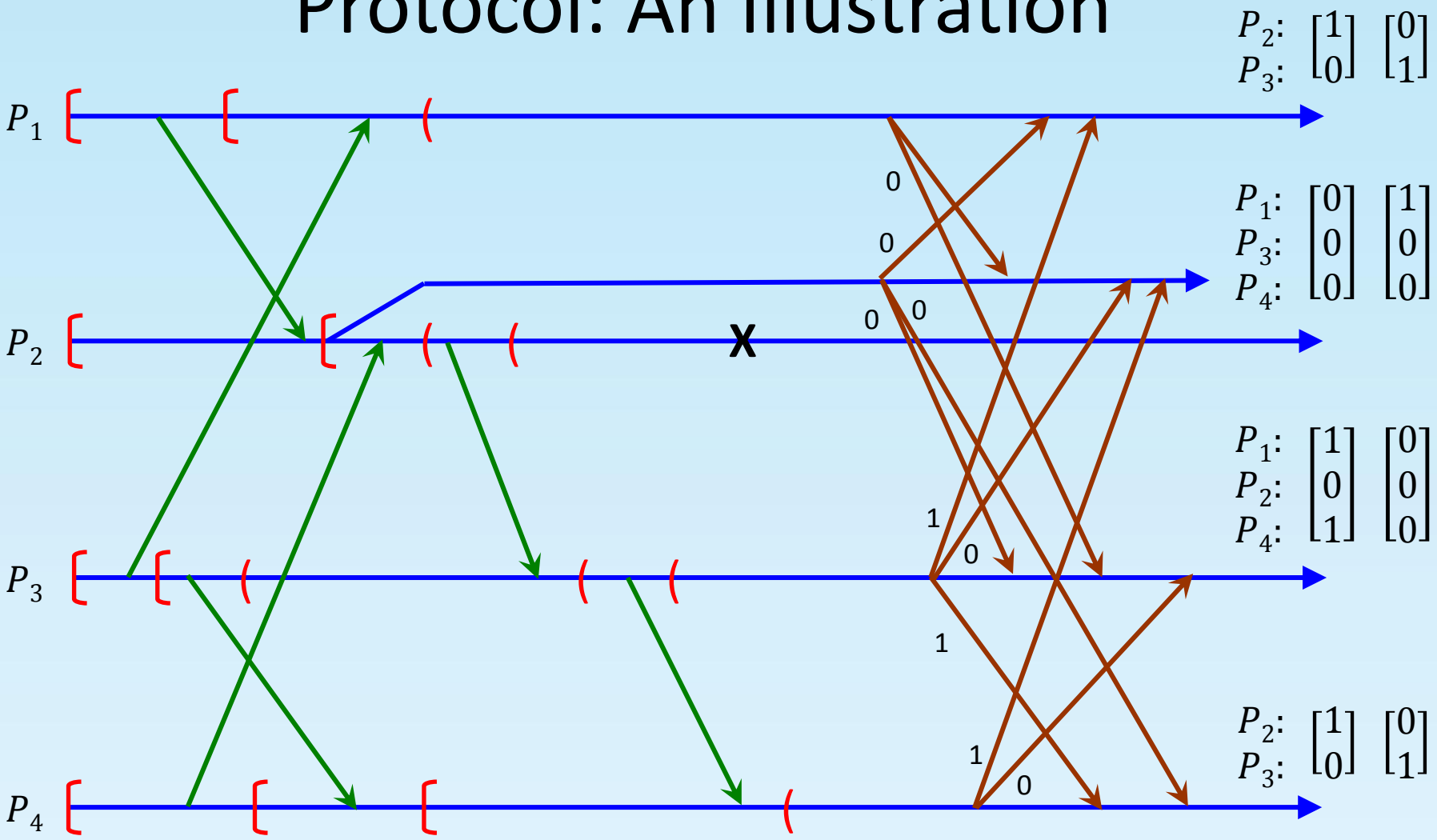
Iteration 3: every process sends entries of $SENT$ vector to its neighbors

Juang and Venkatesan's Recovery Protocol: An Illustration



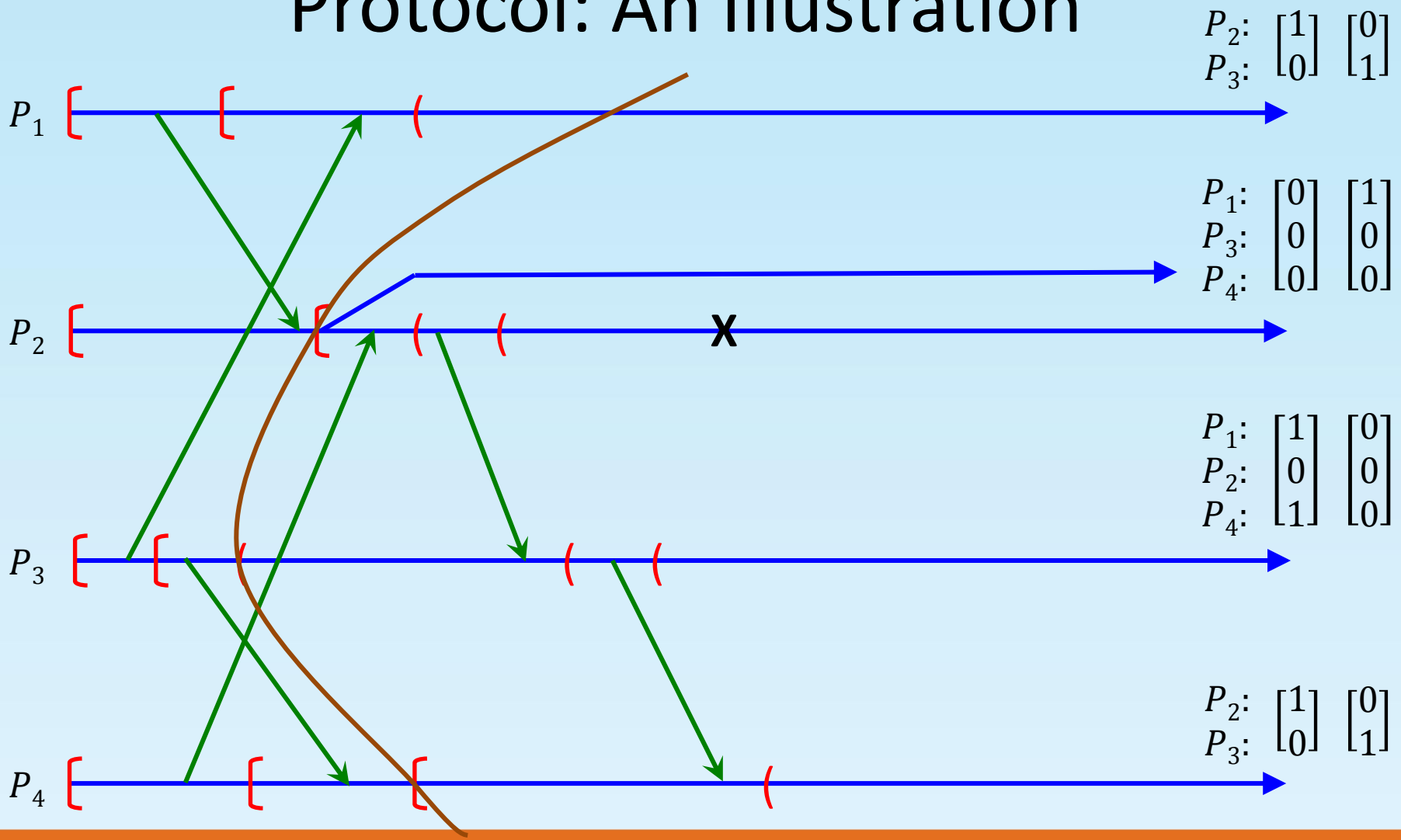
Each process checks if its current state is inconsistent with that of its neighbors

Juang and Venkatesan's Recovery Protocol: An Illustration



No process needs to rollback

Juang and Venkatesan's Recovery Protocol: An Illustration



System state at the end of iteration 3; it is a consistent global state

Juang and Venkatesan's Recovery Protocol: An Illustration