Paxos

DIDA: Class 03

Paxos

- Algorithm invented by L. Lamport to solve consensus without a perfect failure detector

- Leader based algorithm: the algorithm assumes that one of the processes becomes the leader and coordinates the selection of the output value

Leaders in Paxos

- Processes pre-agree on a order to become leader
- Let the set of processes be {p1, p2, …, pn}
- p1 is the 1st leader
- If p2 suspects p1 has crashed, p2 becomes the second leader
- If p3 suspects that p1 and p2 have crashed, p3 becomes the 3rd leader
- Etc

Leaders in Paxos

- Because failure detection is imperfect, p2 may suspect p1 while p1 is still live, etc, and the last process pn can also become slow or crash
- Thus, if pn is suspectes, p1 becomes leader again
- Assume a set of 5 processes {p1, p2, p3, p4, p5}
  - p1 is leader 1, 6, 11, 16, etc
  - p2 is leader 2, 7, 12, 17, etc
  - …
  - p5 is leader 5, 10, 15, 20, etc

Leaders in Paxos

- The leader numbers work as a logical clock that marks the passge of logical time
  - Leader 3 runs “after” leader 2
  - Leader 5 runs “after” leader 3
  - Leader 4 runs runs the “past” of leader 6

Algorithm of the Leader (intuition)

- Each leader performs two steps:
  - Step 1: Checks for evidence of the activity of past leaders and selects a value that is consistent with any decision that previous leaders may have taken
  - Step 2: Attempts to have a majority of nodes to adopted the value it has selected

  - If at the end of Step 2, a majority of nodes have adopted the value proposed by the leader, that value is decided!

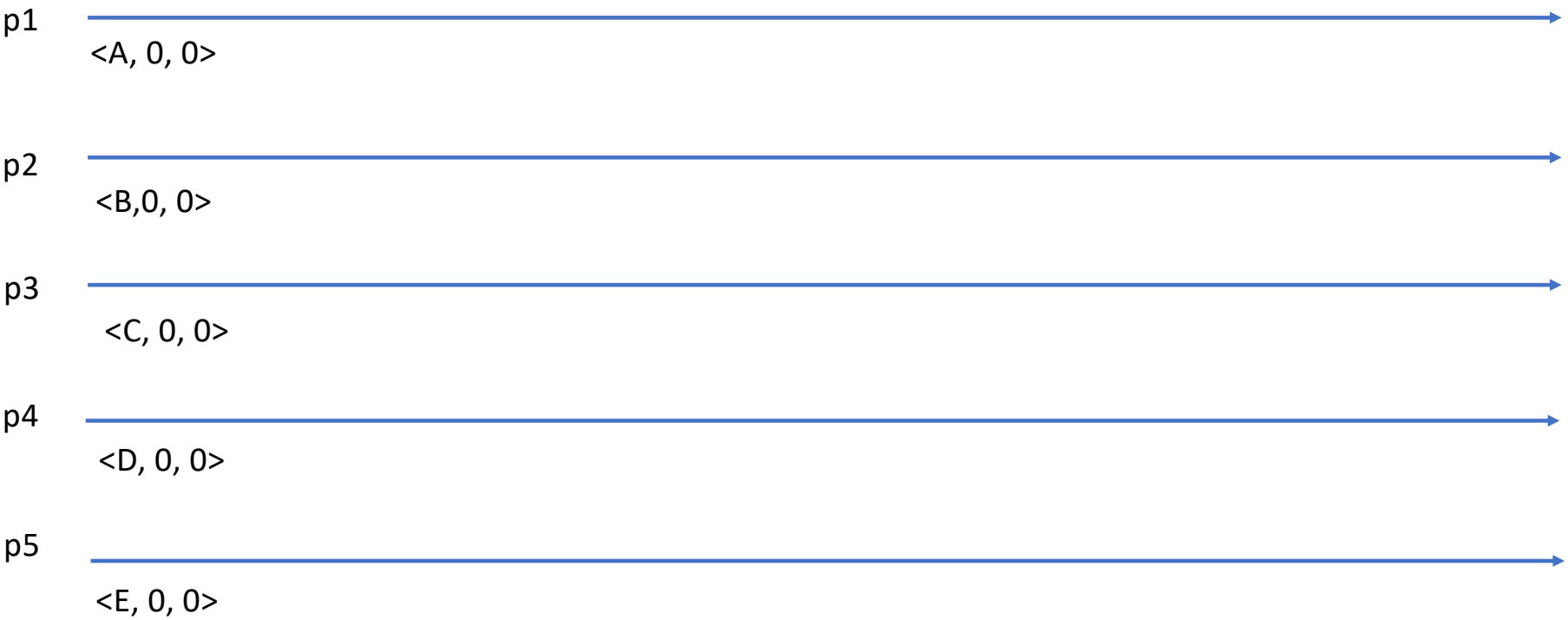
  - Note: The first leader can skip Step 1, because there was no previous leader

Adopted value

- Each process keeps a tuple with the last adopted value:
  - <value, write_timestamp, read_timestamp>

- Initially:
  - <m_proposed_value, 0, 0>

Example: initial state

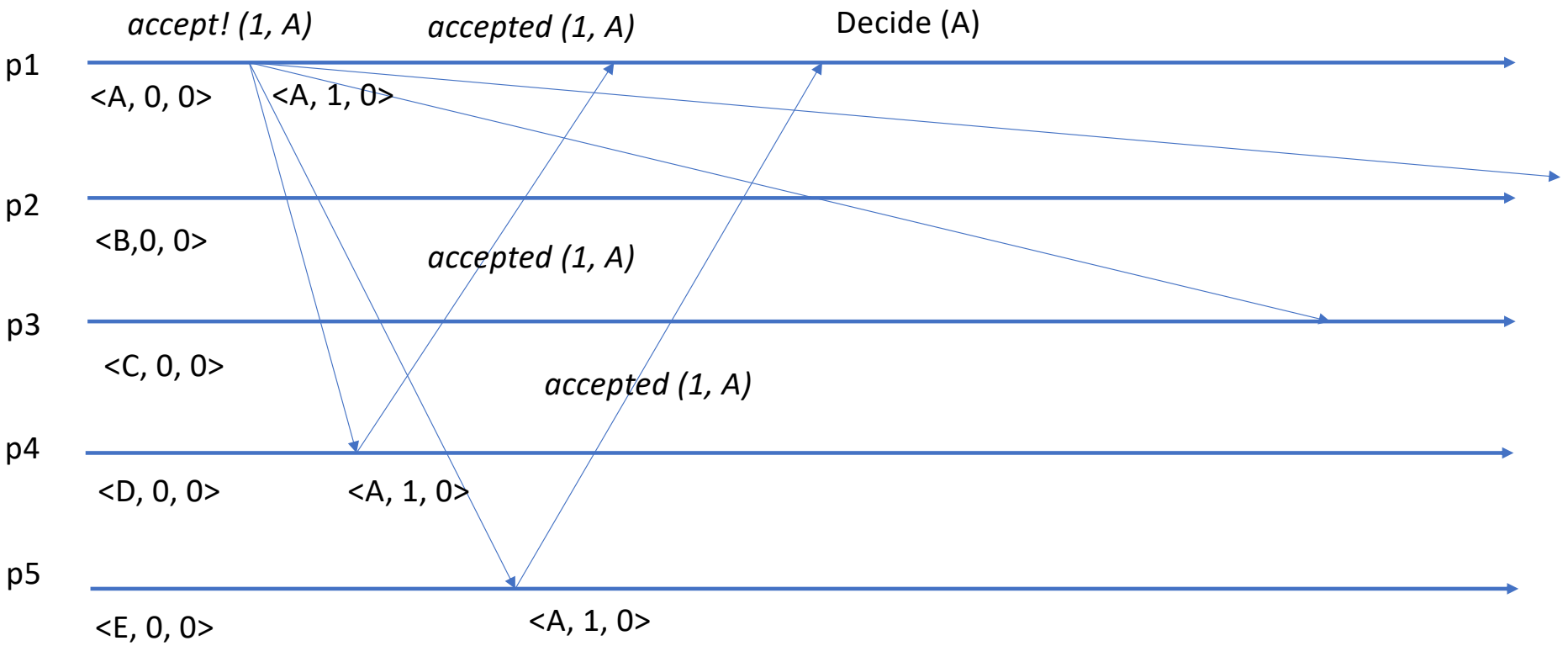


Work of 1st leader

- As noted before, the first leader can skip step 1
- It starts executing step 2 as follows:
  - It sends an "*accept! (1, my_value)"* to all other processes
  - It waits for "*accepted (1, my_value)"* from a majority of processes
  - If a majority accepts, *"my_value"* is decided!

- When a node accepts the value of the leader, it changes its tuple acordingly

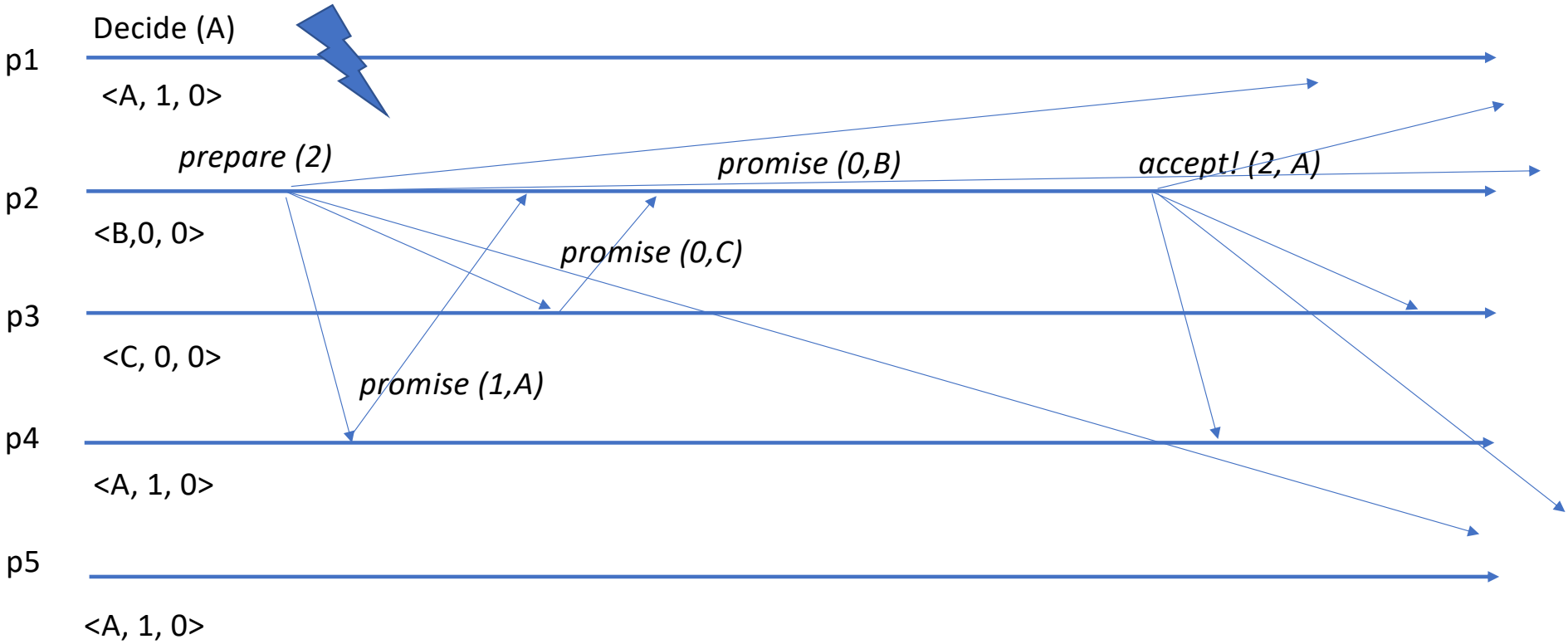
Example: successful run of leader 1



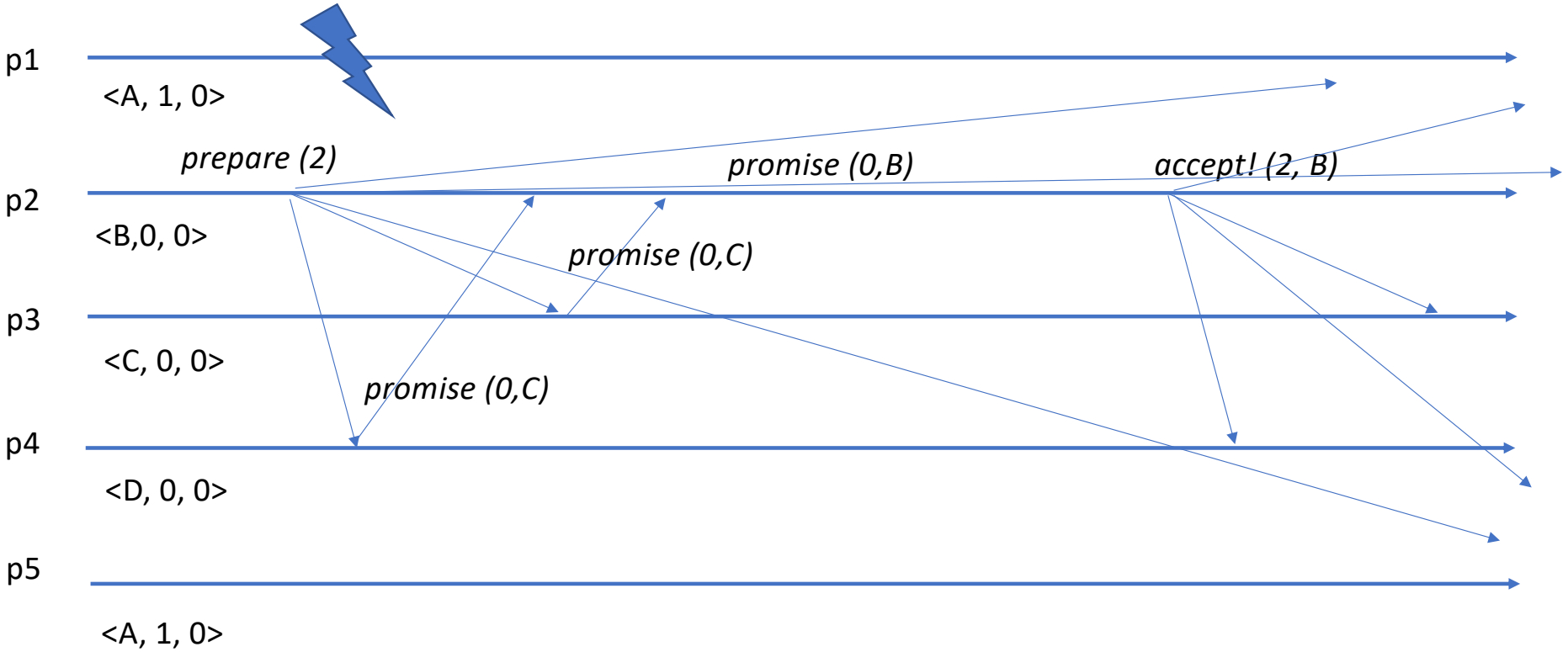
Work of 2nd leader

- Step 1:
  - Sends “prepare (2)” to all nodes.
  - Waits for “promise (write_timestamp, value)” from a majority of nodes
  - Adopts the most recent value in the set of the responses received (i.e., the value with highest write_timestamp)
  - If all values (in the majority of replonses) are still the initial values (with timsetamp 0), it adopts its own value.
  - Proceeds to execute Step 2 as before.

Example: run of leader 2 after leader 1



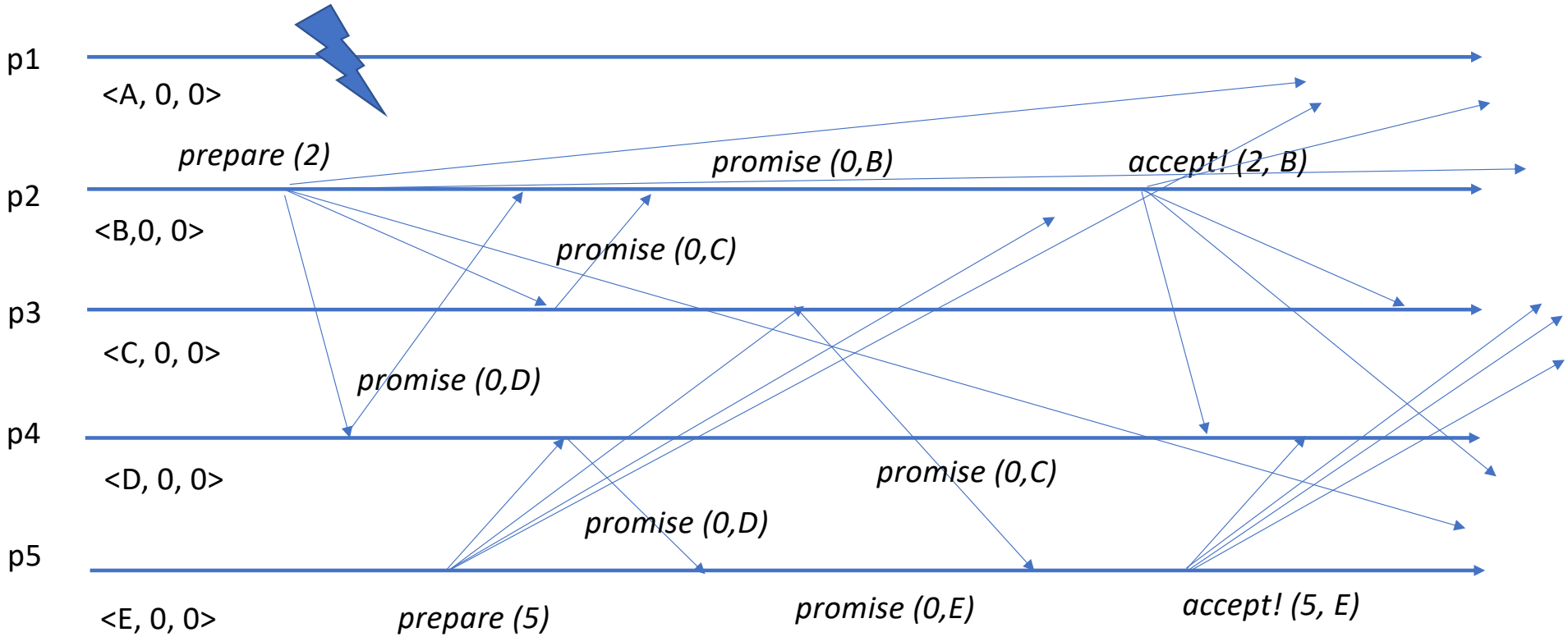
Example: run of leader 2 without evidence of leader 1



If two leaders attempt to execute concurrently?

- Unless with add some additional mechanims to the algorithm, two concurrent leaders may propose conflicting values

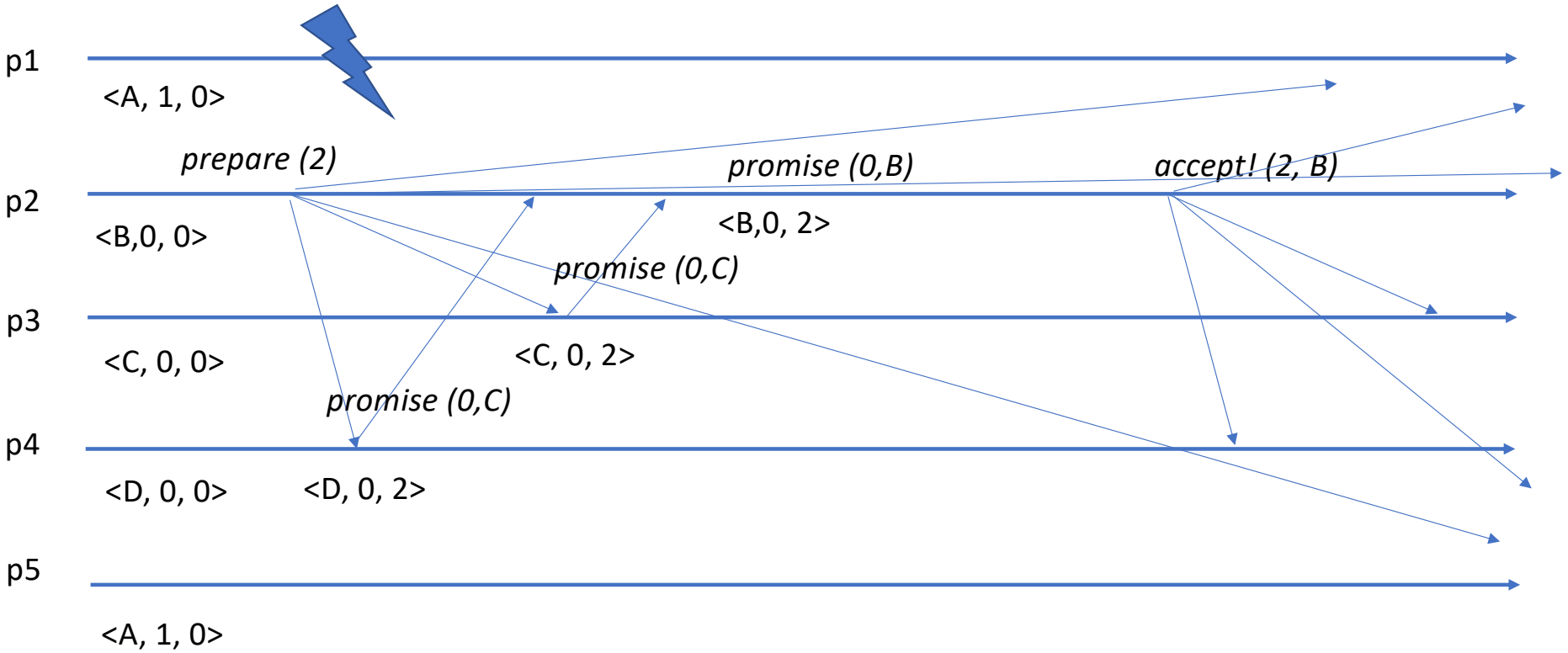
Example: concurrent leaders



Read timestamp

- When a leader executes step 1, and asks the state of each process, a process that replies, “memorizes” who was the last leader to read the state, using the “*read_timestamp*” field.

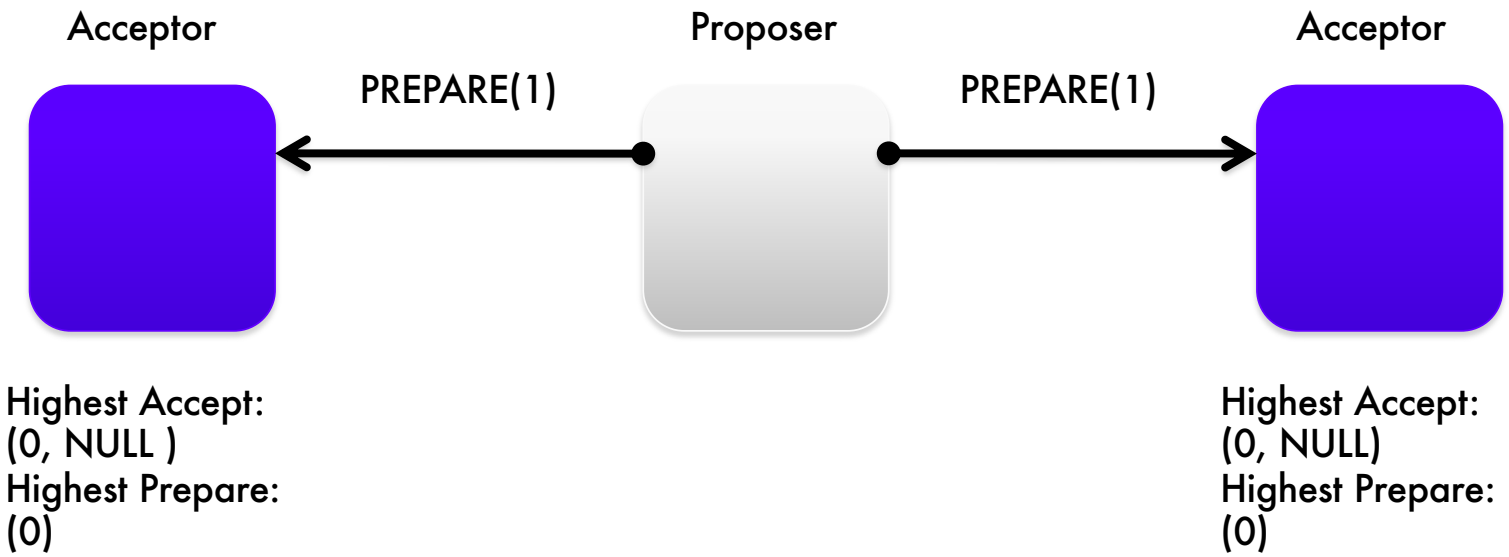
Example: read timestamp



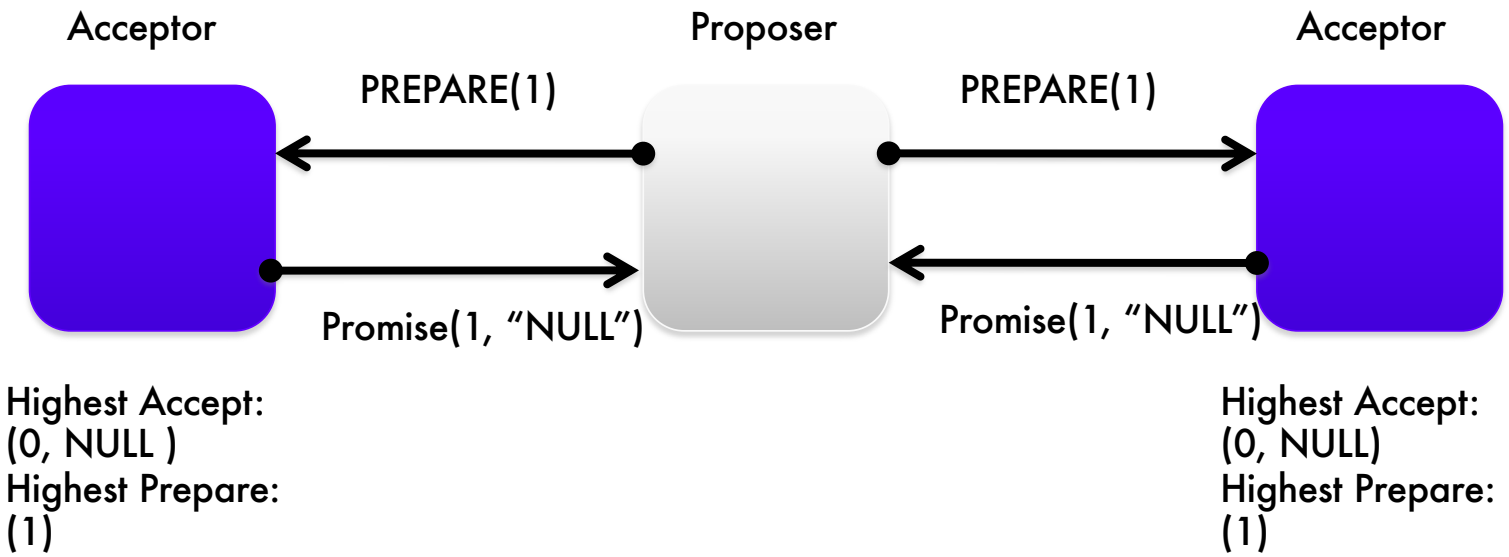
Read timestamp

- The "*read_timestamp*" field is used to prevent two concurrent leaders from commiting conflictiong values:
  - Nodes only send "promise" messages to leaders that use a timetsamp larger than any timestamp seen in the past (if a leader is running in the past of another process, it must give up, and try again using a timestamp in the future)
  - Nodes only accept values that are proposed by the last leader to read the tuple, i.e., a leader must read and write the tuple alone, without the interference of other leaders, to be able to commit a value.
  - This implements something that is similar to a fault-tolerant, distributed, "compare-and-swap".

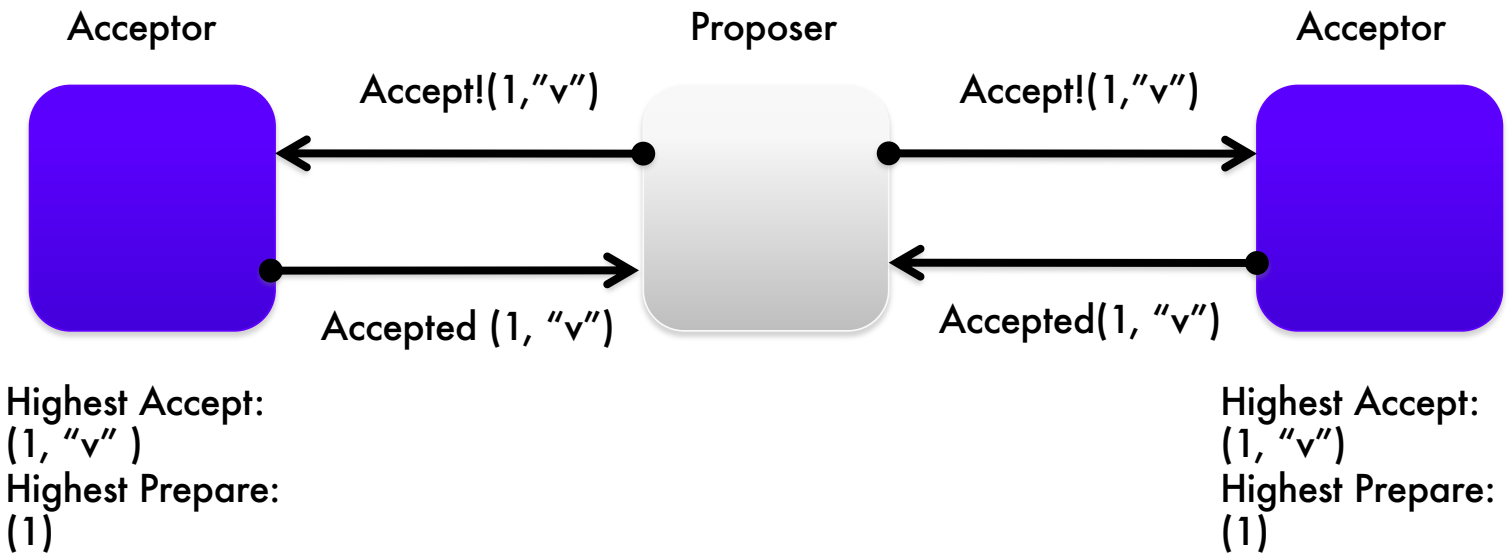
Example: Normal behavior
Phase 1



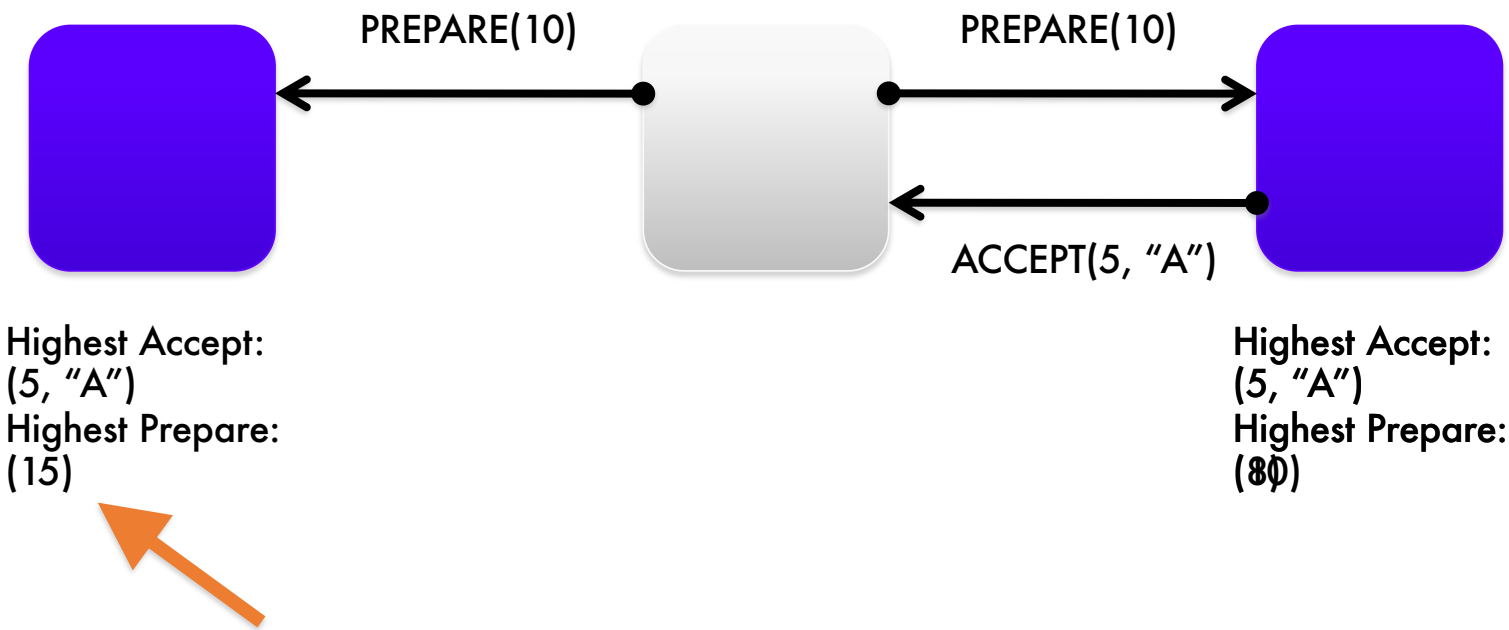
Example: Normal behavior
Phase 1



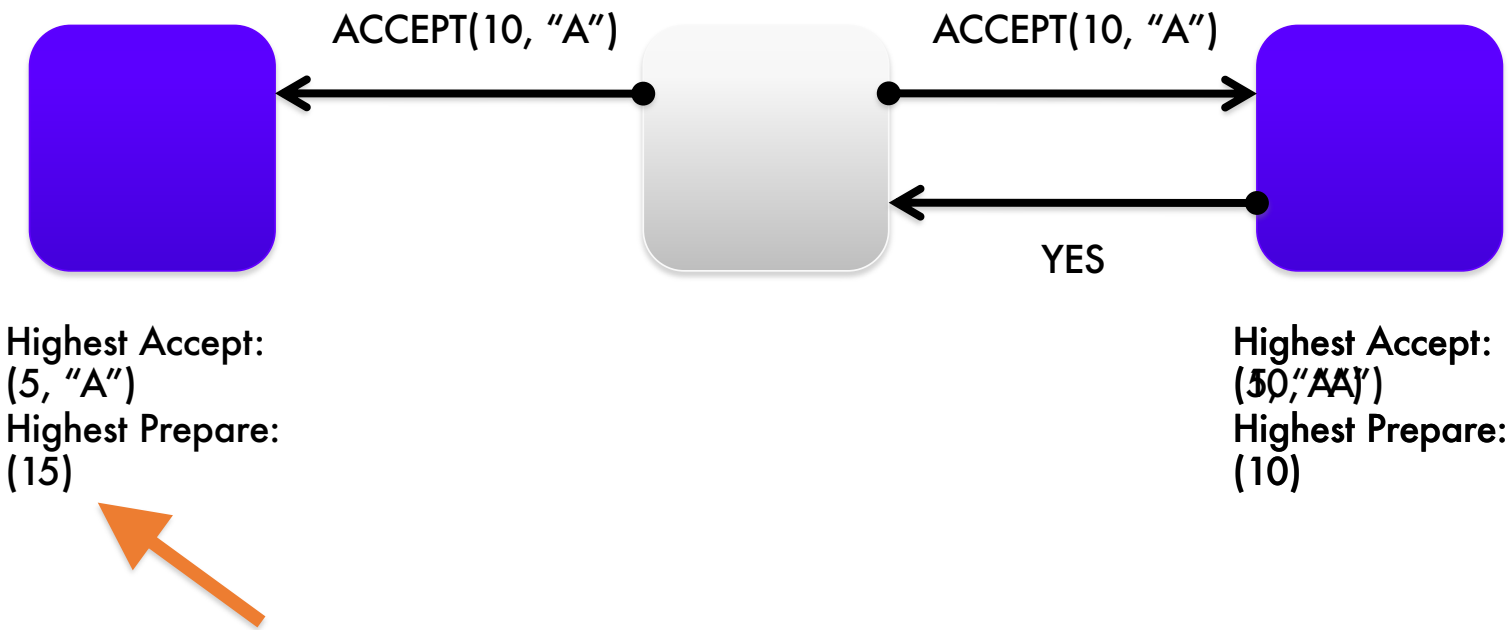
Example: Normal behavior
Phase 2



Example:
Prepare refused due to concurrent leader



Example:
Accept refused due to concurrent leader



Example: Livelock