CS 425/ECE 428/CSE424 Distributed Systems (Fall 2009)

Lecture 8

Leader Election

Section 12.3

Klara Nahrstedt

Acknowledgement

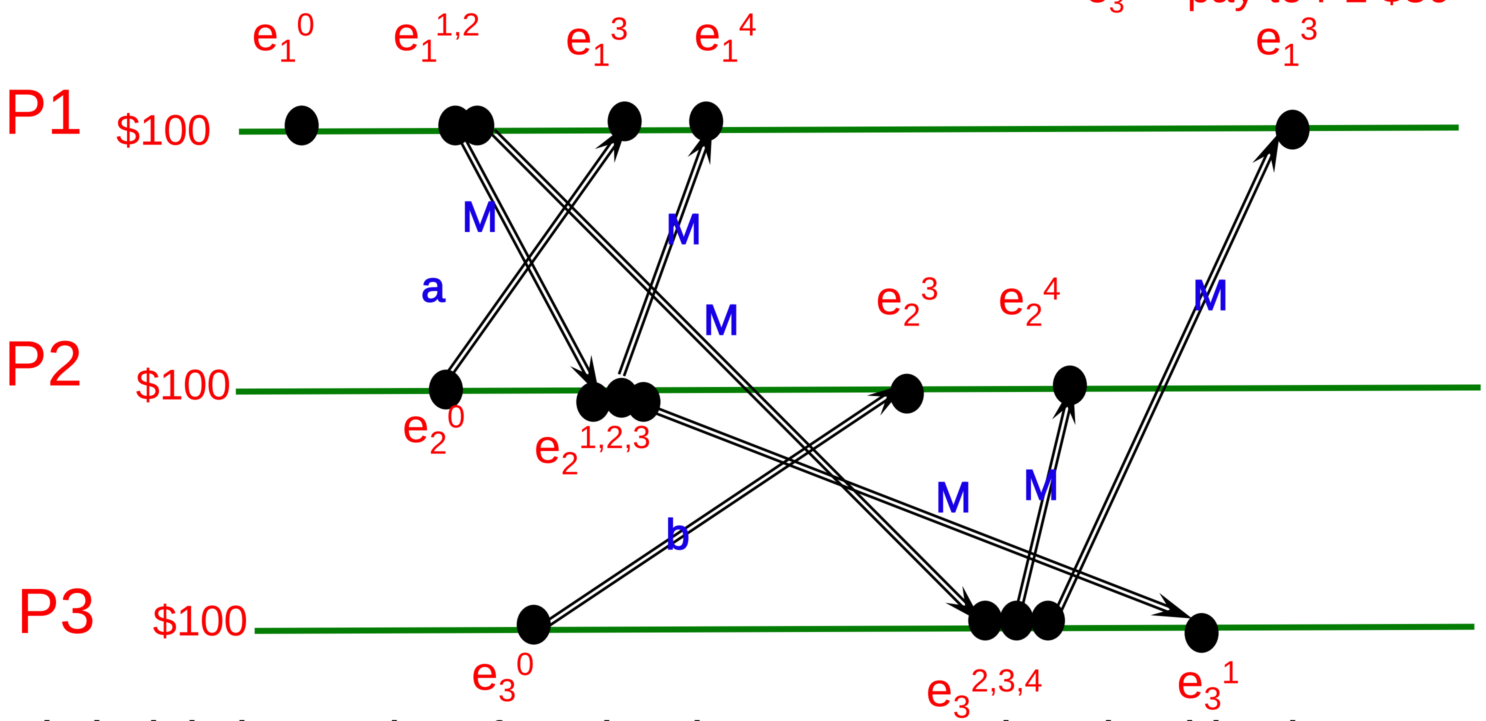
- **The slides during this semester are based on ideas and material from the following sources:**
  - **Slides prepared by Professors M. Harandi, J. Hou, I. Gupta, N. Vaidya, Y-Ch. Hu, S. Mitra.**
  - **Slides from Professor S. Gosh’s course at University o Iowa.**

Administrative

- **MP1 posted September 8, Tuesday**
  - **Deadline, September 25 (Friday), 4-6pm Demonstrations**

Re-Visit Snapshot Example

e_1^0 – deduct $10
e_2^0 – pay to P1 $20
e_3^0 – pay to P2 $30



Final Recorded Global Snapshot after Chandy-Lamport Marker Algorithm is over:

at P1 : {S1: <$90>, channel message ‘,<$20>’)
at P2 : (S2: <$80> ,channel message ‘<$30>’)
at P3 : (S3: <$70> ,channel message { })

We use couple of concepts: (a) reliable multicast of ‘markers’ M, , (b) causal ordering of markers; (c) closed group communication
If one wants the global snapshot at one place, one could have a coordinator to collect the snapshot states from each process.

Historical Comparison of Distributed Mutual Exclusion Algorithms

- **1978 – Introduction of logical Lamport clocks and event ordering (Lamport)**
- **1981 - Ricart and Agrawala – distributed mutual exclusion**
  - **Use of Happen-Before relation and Lamport clocks**
  - **Bandwidth :**
    - **Messages to entry: 2(N-1)**
    - **Message to exist: N-1**
  - **Client delay**
    - **one round trip**
  - **Synchronization delay**
    - **1 message**

Historical Comparison of Distributed Mutual Exclusion Algorithms

- **1985 – Maekawa - √N distributed mutual exclusion**
  - **Bandwidth :**
    - » **2√N messages per entry,**
    - » **√N messages per exit**
  - **Client delay:**
    - » **One round trip time**
  - **Synchronization delay:**
    - » **One round-trip time**
- **1989 – Raymond – O(log N) distributed mutual exclusion**
  - **Bandwidth:**
    - » **2D messages per entry (D – longest path length)**
    - » **D messages per exist**
  - **Best case – radiating star topology (D = log N)**
  - **Worst case ?**
  -

Plan for today

- **Election algorithms**
  - **Ring-based algorithm**
  - **Modified ring-based algorithm**
  - **Bully algorithm**

Why Election?

- **Example 1: Your Bank maintains multiple servers, but for each customer, one of the servers is responsible, i.e., is the leader**
- **Example 2: In the sequencer-based algorithm for total ordering of multicasts,**
  - **What happens if the "special" sequencer process fails?**
- **Example 3: Coordinator-based distributed mutual exclusion: need to elect (and keep) one coordinator**
- **In a group of processes, elect a *Leader* to undertake special tasks. Makes the algorithm design easy.**
- **But leader may fail (crash)**
  - **Some process detects this**
  - **Then what?**

Assumptions and Requirements

- Any process can call for an election.
- A process can call for at most one election at a time.
- Multiple processes can call an election simultaneously.
- The result of an election should not depend on which process calls for it.
- Each process has
  - Variable called *elected*
  - An attribute value called *attr,* e.g., id, MAC address, CPU
- The non-faulty process with the best (highest) election attribute value (e.g., highest id or address, or fastest cpu, etc.) is elected.
- **Requirement:** A *run* (execution) of the election algorithm must always guarantee at the end:
  - Safety: $\forall$ P (P's `elected` = (q: non-failed process with the best attribute value) or $\perp$)
  - Liveness: $\forall$ election((election terminates) & $\forall$ P: non-faulty process, P's `elected` is not $\perp$)

Ring Election

- N **Processes are organized in a logical ring.**
  - p_i **has a communication channel to** $p_{(i+1) \bmod N}$.
  - **All messages are sent clockwise around the ring.**
- **Any process** p_i **that discovers a coordinator has failed initiates an "election" message** $<i, p_i.attr>$
- **When a process** p_j **receives an** ***election*** **message** $<i, p_i.attr>$**, it compares the** ***attr*** **in the message with its own.**
  - **If the arrived** $p_i.attr > p_j.attr$ **, then receiver** p_j **forwards the message** $<i, p_i.attr>$**.**
  - **If the arrived** $p_i.attr < p_j.attr$ **and the receiver** p_j **has not forwarded an election message earlier, it substitutes its own** $<j, p_j.attr>$ **in the message and forwards it.**
  - **If the arrived** $p_i.attr = p_j.attr$ **, then this process's** $p_j.attr$ **must be the greatest, and it becomes the new coordinator. This process then sends an "elected" message to its neighbor announcing the election result.**
- **When a process** p_i **receives an elected message, it**
  - **sets its variable** $elected_i \leftarrow$ **id of the message.**
  - **forwards the message if it is not the new coordinator.**

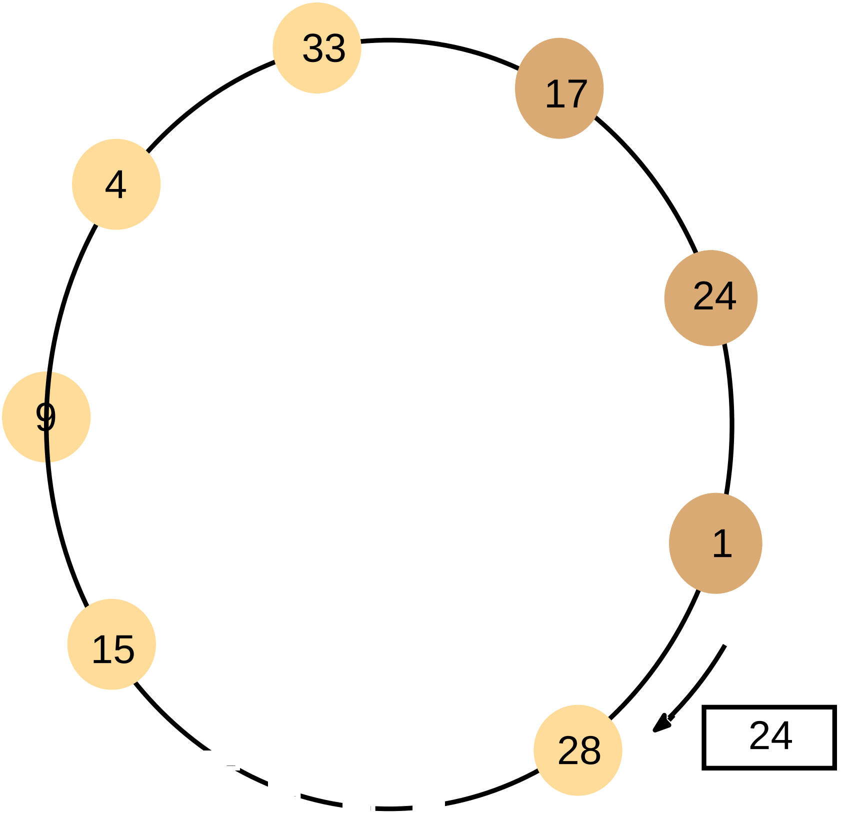
A Ring-Based Election in Progress

Ring of N processes

(attr:=id)

- **The worst-case scenario occurs when the counter-clockwise neighbor has the highest *attr***

- **A total of *N-1* messages is required to reach the new coordinator-to-be.**
- **Another *N* messages are required until the new coordinator-to-be ensures it is the new coordinator.**
- **Another *N* messages are required to circulate the elected messages.**



Note: The election was started by process 17.
The highest process identifier encountered so far is 24.
(final leader will be 33)

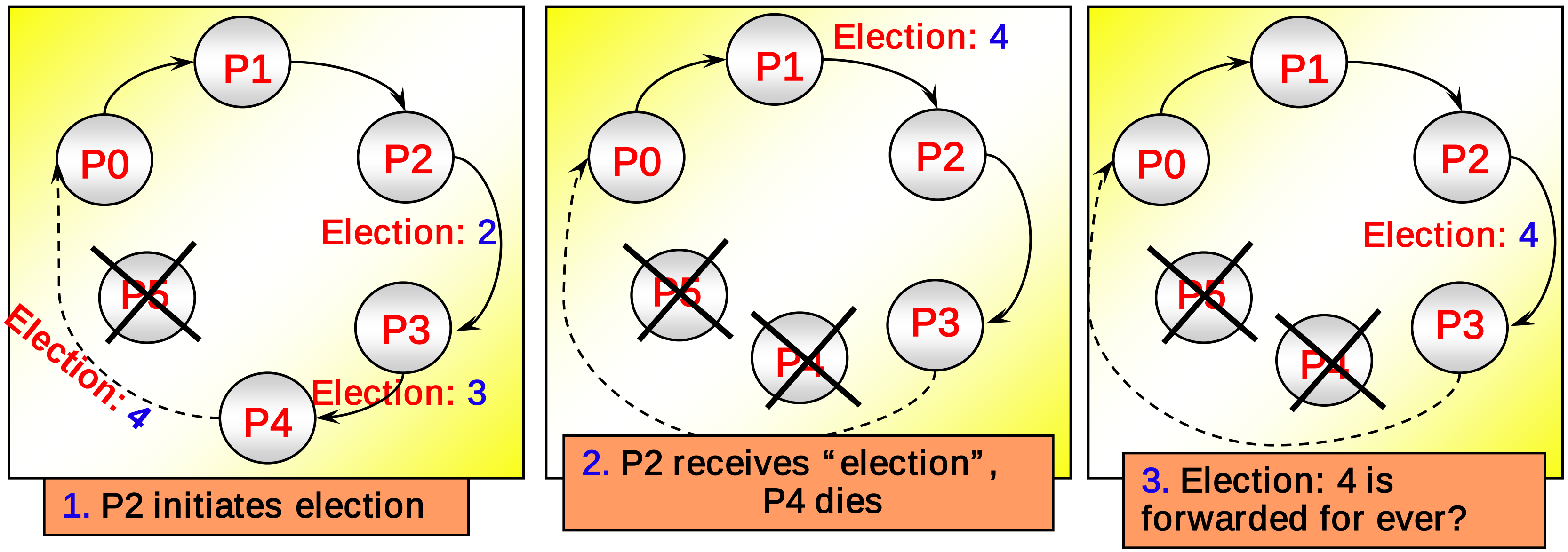
Ring-based Election

Assume – no failures happen during the run of the election algorithm

- **Safety and Liveness are satisfied.**

What happens if there are failures during the election run?

Example: Ring Election

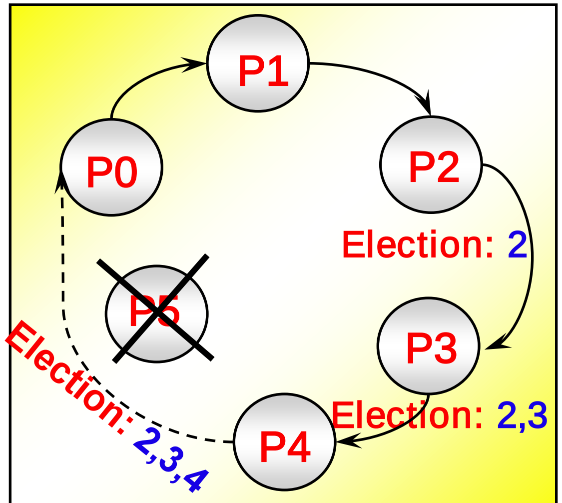


May not work when process failure occurs during the election!
Consider above example where attr==highest id

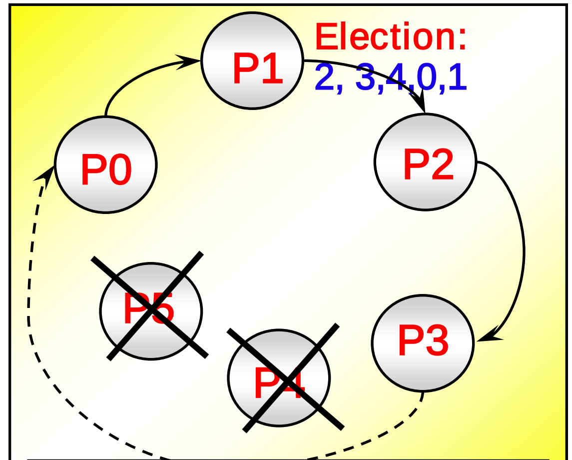
Modification to Ring Election

- **Processes are organized in a logical ring.**
- **Any process that discovers the coordinator (leader) has failed initiates an "election" message. This is the *initiator* of the election.**
- **The message is circulated around the ring, bypassing failed nodes.**
- **Each node <u>adds (appends)</u> its *id:attr* to the message as it passes it to the next node.**
- **Once the message gets to the initiator, it elects the node with the best election attribute value.**
- **It then sends a "coordinator" message with the id of the newly-elected coordinator. Again, each node adds (appends) its *id* to the end of the message.**
- **Once "coordinator" message gets back to initiator,**
  - **election is over if "coordinator" is in id-list.**
  - **else the algorithm is repeated (handles election failure).**

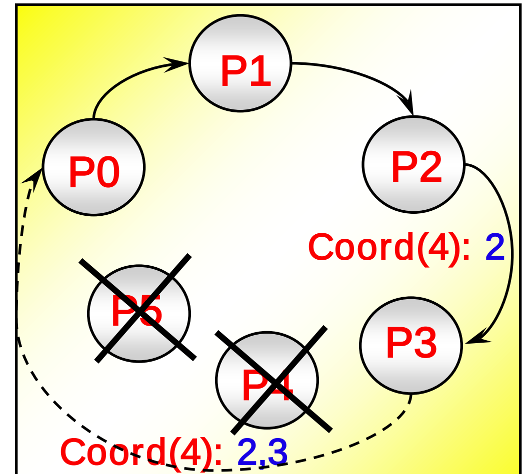
Example: Ring Election



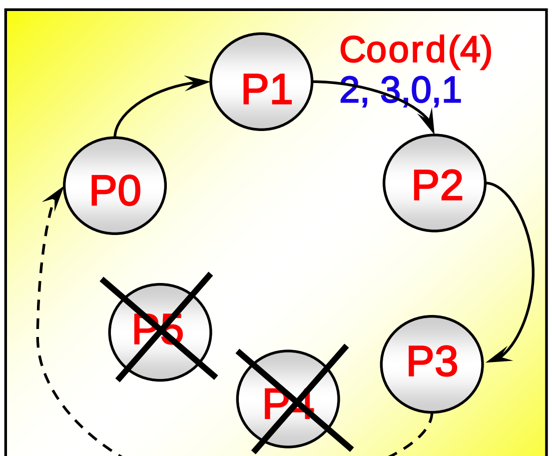
1. P2 initiates election



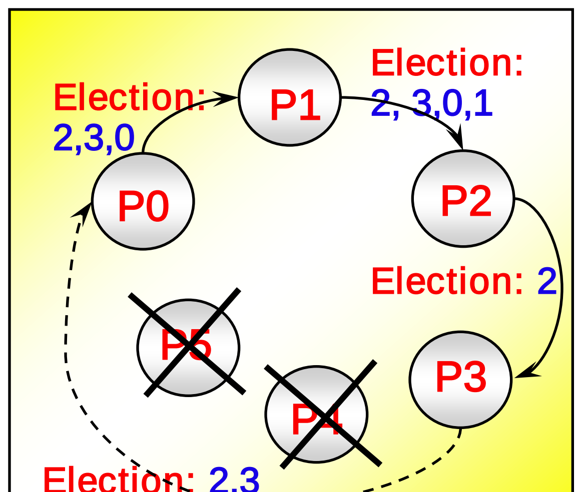
2. P2 receives “election”, P4 dies



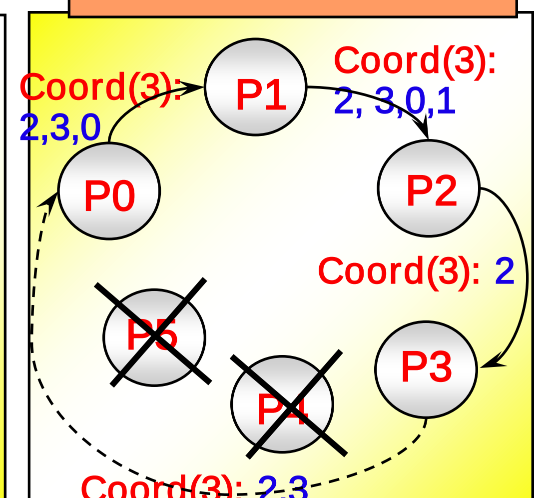
3. P2 selects 4 and announces the result



4. P2 receives “Coord”, but P4 is not included



5. P2 re-initiates election



6. P3 is finally elected

Modified Ring Election

- **How would you redesign the algorithm to be fault-tolerant to an initiator's failure?**
  - **One idea: Have the initiator's successor wait a while, then re-initiate a new election. Do the same for this successor's successor, and so on…**
- **Reconfiguration of ring upon failures**
  - **Ok if all processes "know" about all other processes in the system**

Election by the Bully Algorithm

- **Assumptions:**
  - **Synchronous system**
    - All messages arrive within T_{trans} units of time.
    - A reply is dispatched within $T_{process}$ units of time after the receipt of a message.
    - if no response is received in $2T_{trans} + T_{process}$, the node is assumed to be faulty (crashed).
  - attr=id
  - Each process knows all the other processes in the system (and thus their id's)

Election by the Bully Algorithm

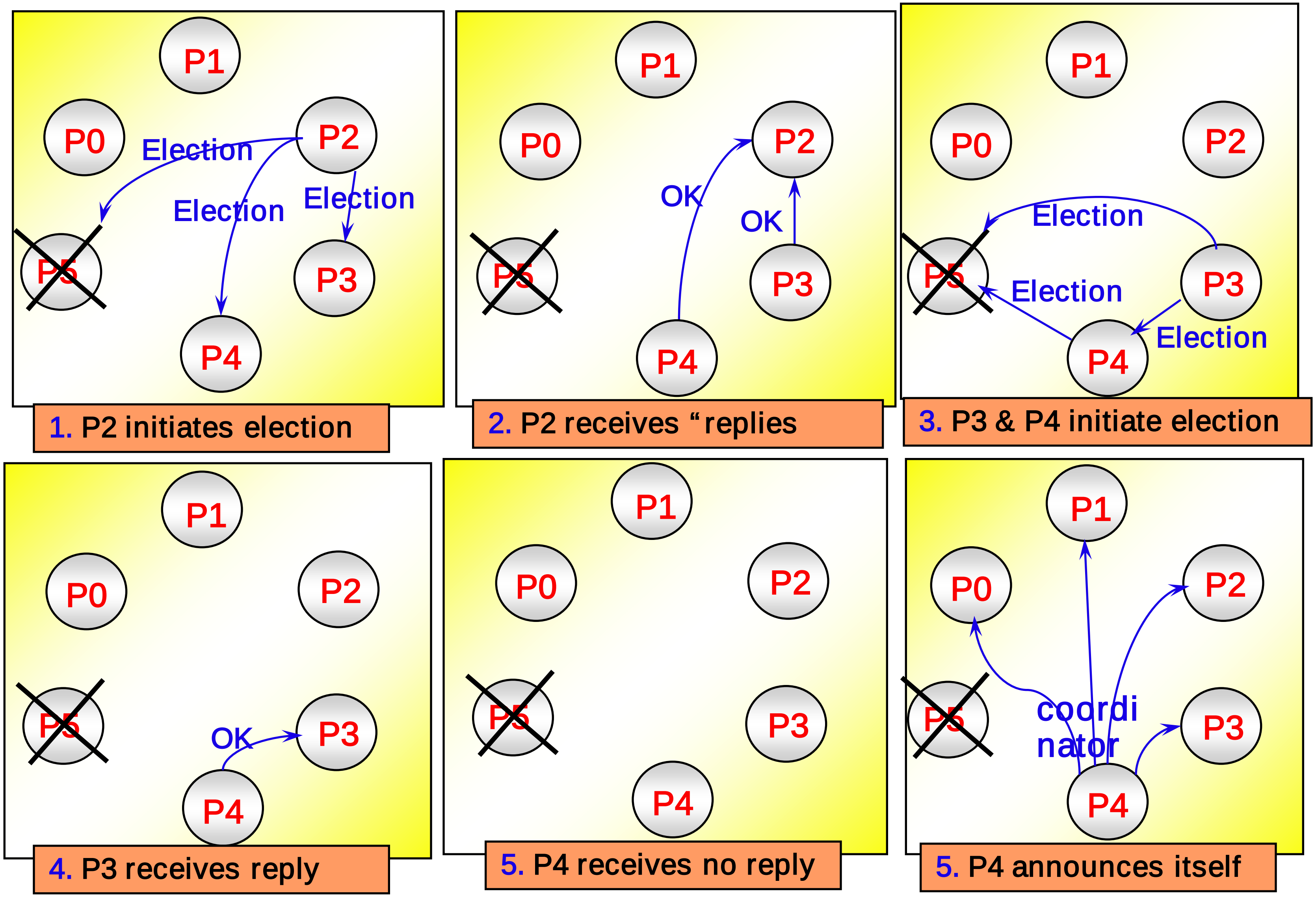
- A node initiates election by sending an "election" message to only nodes that have a higher id than itself.
  - If no answer, announce itself to lower nodes as coordinator.
  - if any answer, then there is some higher node active; wait for coordinator message. If none received after time out, start a new election.

- A node that receives an "election" message replies with answer, & starts an election – unless it has already.

- When a process finds the coordinator has failed, if it knows its id is the highest, it elects itself as coordinator, then sends a *coordinator* message to all processes with lower identifiers.

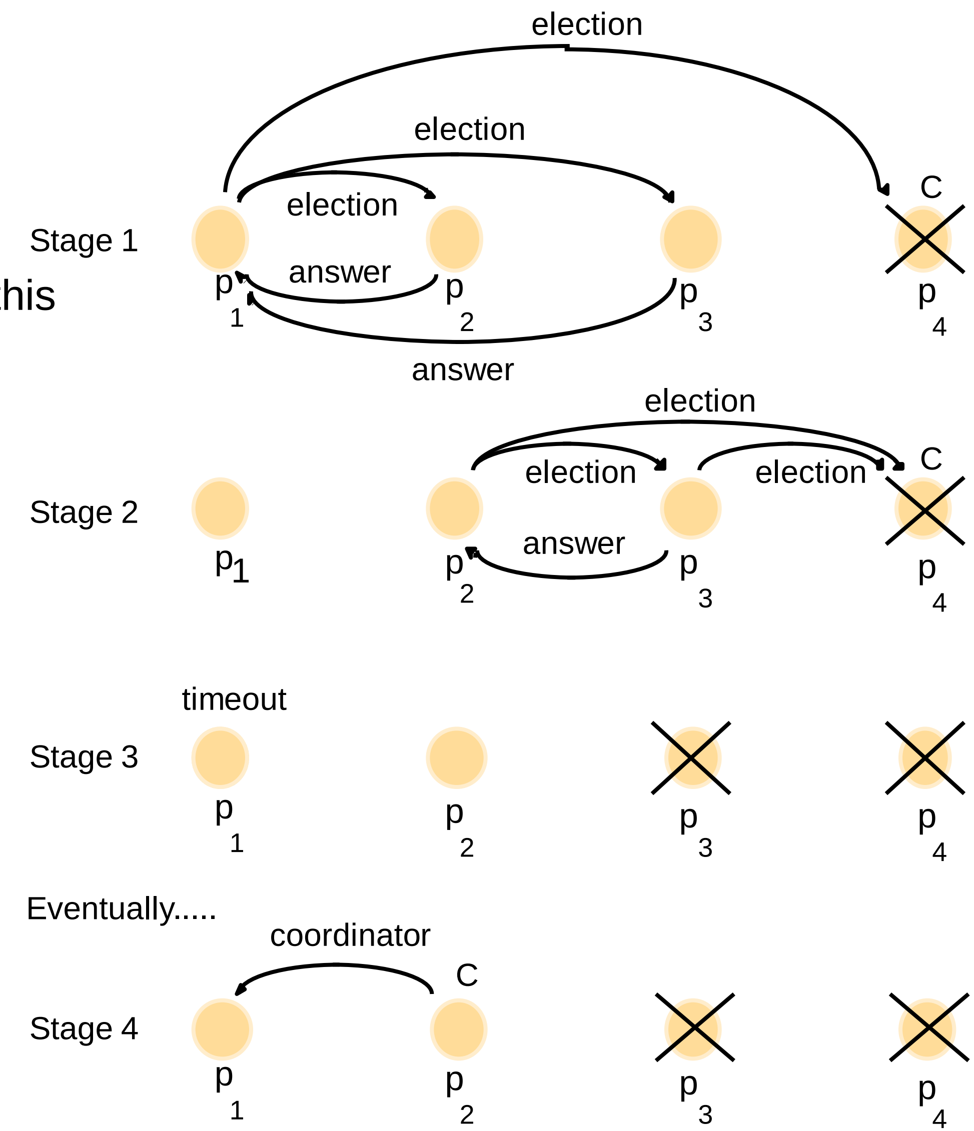
Example: Bully Election

answer=OK



The Bully Algorithm

The coordinator p_4 fails and p_1 detects this



p_3 fails

Performance of Bully Algorithm

- **Best case scenario: The process with the second highest id notices the failure of the coordinator and elects itself.**
  - *N-2* **coordinator messages are sent.**
  - **Turnaround time is one message transmission time.**
- **Worst case scenario: When the process with the least id detects the failure.**
  - **N-1 processes altogether begin elections, each sending messages to processes with higher ids.**
  - **The message overhead is** $O(N^2)$.
  - **Turnaround time is approximately 5 message transmission times if there are no failures during the run: election, answer, election, answer, coordinator**

What have we Learnt?

- **Coordination requires a leader process, e.g., sequencer for total ordering in multicasts, bank database example, coordinator-based mutual exclusion.**
- **Leader process might fail**
- **Need to (re-) elect leader process**
- **Three Algorithms**
  - **Ring algorithm**
  - **Modified Ring algorithm**
  - **Bully Algorithm**

Summary

- **Election algorithms**
  - **Ring-based algorithm**
  - **Modified ring-based algorithm**
  - **Bully algorithm**
- **Reading for Next Class: consensus**