

Computer Science 425

Distributed Systems

CS 425 / CSE 424 / ECE 428

Fall 2012

Indranil Gupta (Indy)

October 2, 2012

Lecture 11

Leader Election

Reading: Sections 15.3

NCSA Innovation Fellowships

- **SPIN = Students Pushing Innovation**
- **15 paid fellowships by NCSA**
- **You'll be working with an NCSA mentor**
- **Oct 24 4pm-6pm meet and greet with NCSA researchers**
- **Submit proposal afterwards**
- **go.illinois.edu/opportunities**

Not this kind of election



Why Election?

❖ **Example 1: Your Bank maintains multiple servers in their cloud, but for each customer, one of the servers is responsible, i.e., is the **leader****

❖ **What if there are two leaders per customer?**

❖ **Inconsistency**

❖ **What if servers disagree about who the leader is?**

❖ **Inconsistency**

❖ **What if the leader crashes?**

❖ **Unavailability**

Why Election?

- ❖ **Example 2: (few lectures ago) In the sequencer-based algorithm for total ordering of multicasts, the "sequencer" = leader**
- ❖ **Example 3: Group of cloud servers replicating a file need to elect one among them as the primary replica that will communicate with the client machines**
- ❖ **Example 4: Group of NTP servers: who is the root server?**

What is Election?

- ❖ In a group of processes, elect a **Leader** to undertake special tasks.
- ❖ What happens when a leader fails (crashes)
 - ❖ Some process detects this (how?)
 - ❖ Then what?
- ❖ Focus of this lecture: **Election algorithm**
 1. Elect one leader only among the non-faulty processes
 2. All non-faulty processes agree on who is the leader

System Model/Assumptions

- ❖ 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.
 - ❖ All of them together must yield a single leader only
- ❖ The result of an election should not depend on which process calls for it.
- ❖ Messages are eventually delivered.

Problem Specification

- ❖ At the end of the election protocol, the non-faulty process with the best (highest) election attribute value is elected.
- ❖ Attribute examples: leader has highest id or address, or fastest cpu, or most disk space, or most number of files, etc.
- ❖ A *run* (execution) of the election algorithm must always guarantee at the end:
 - **Safety:** \forall non-faulty p : (p 's `elected` = (q : a particular non-faulty process with the best attribute value) or \perp)
 - **Liveness:** \forall election: (election terminates)
& \forall p : non-faulty process, p 's `elected` is not \perp

Algorithm 1: Ring Election

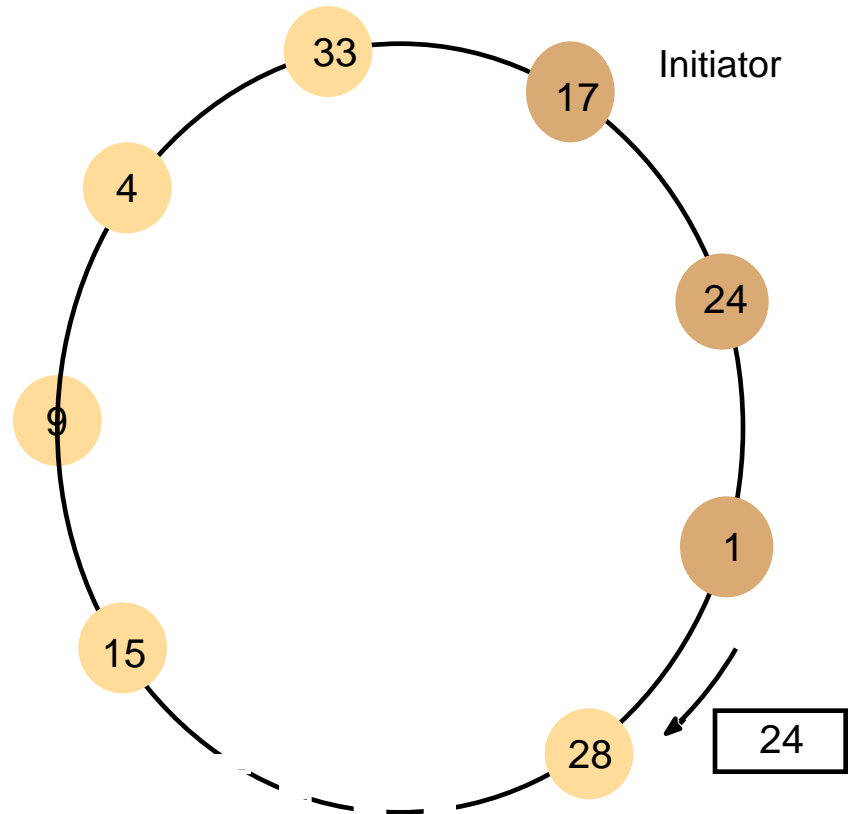
- ❖ N Processes are organized in a logical ring (similar to ring in Chord p2p system)
 - ❖ p_i has a communication channel to $p_{(i+1) \bmod N}$
 - ❖ All messages are sent clockwise around the ring.
- ❖ Any process p_i that discovers the old coordinator has failed initiates an “election” message that contains p_i ’s own id:attr. This is the *initiator* of the election.
- ❖ When a process p_i receives an *election* message, it compares the attr in the message with its own attr.
 - ❖ If the arrived attr is greater, p_i forwards the message.
 - ❖ If the arrived attr is smaller and p_i has not forwarded an election message earlier, it overwrites the message with its own id:attr, and forwards it.
 - ❖ If the arrived id:attr matches that of p_i , then p_i ’s attr must be the greatest (why?), and it becomes the new coordinator. This process then sends an “elected” message to its neighbor with its id, 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.

Ring-Based Election: Example

(In this example, $\text{attr} = \text{id}$)

❖ The worst-case scenario occurs when the counter-clockwise neighbor (@ the initiator) has the highest attr.

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

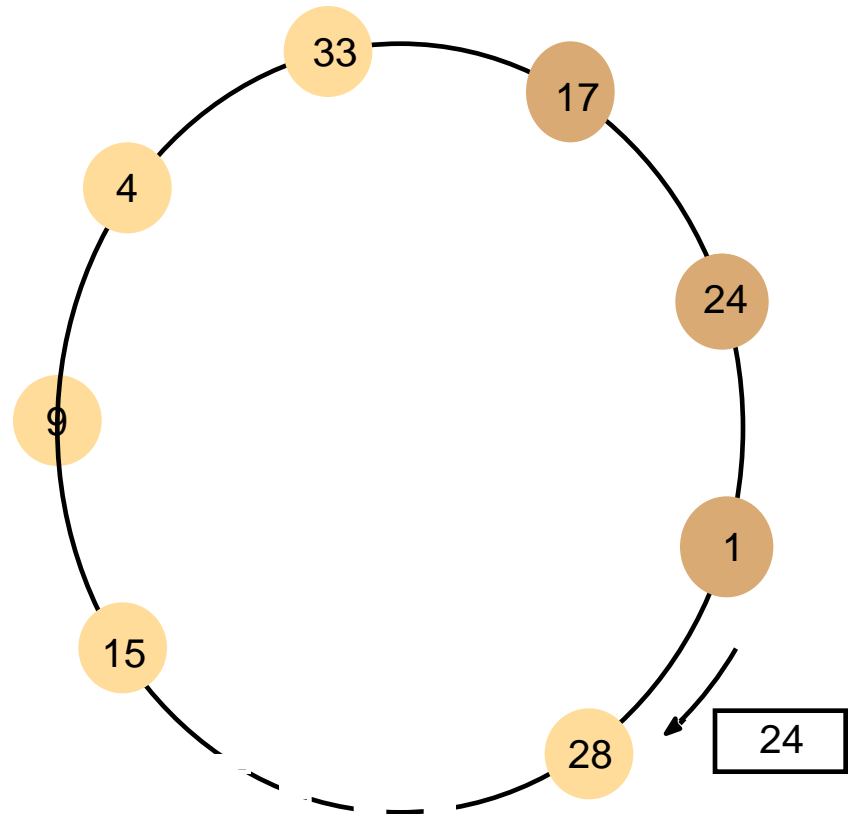


Ring-Based Election: Analysis

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

In a ring of N processes, in the worst case:

- ❖ A total of N-1 messages are required to reach the new coordinator-to-be (election messages).
- ❖ Another N messages are required until the new coordinator-to-be ensures it is the new coordinator (election messages – no changes).
- ❖ Another N messages are required to circulate the elected messages.
- ❖ Total Message Complexity = $3N-1$
- ❖ Turnaround time = $3N-1$



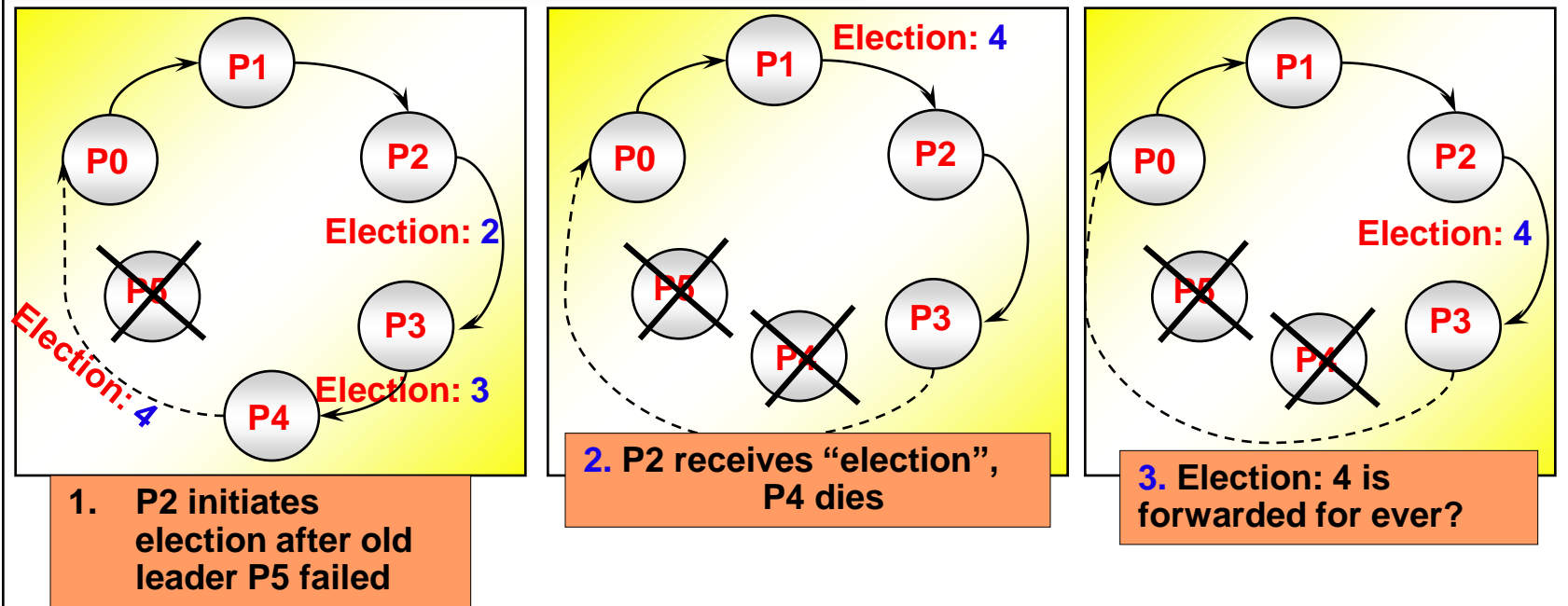
Correctness?

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



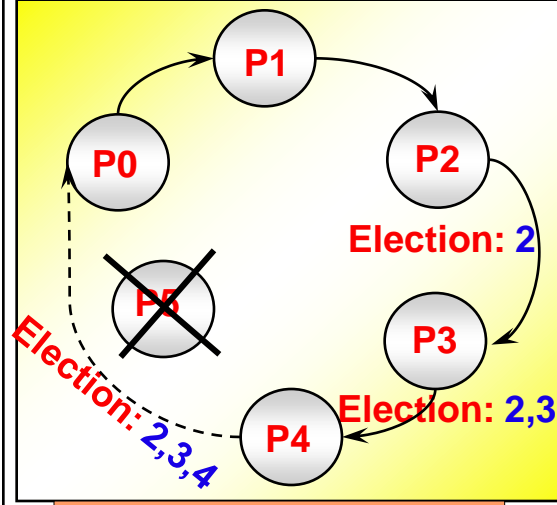
May not terminate when process failure occurs during the election!
Consider above example where $attr == id$

Does not satisfy liveness

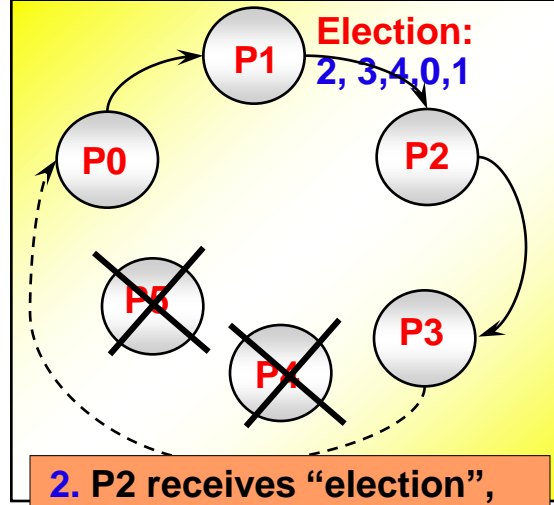
Algorithm 2: Modified Ring Election

- ❖ Processes are organized in a logical ring.
- ❖ Any process that discovers the coordinator (leader) has failed initiates an “election” message.
- ❖ The message is circulated around the ring, bypassing failed processes.
- ❖ Each process appends (adds) its id:attr to the message as it passes it to the next process (without overwriting what is already in the message)
- ❖ Once the message gets back to the initiator, it elects the process with the best election attribute value.
- ❖ It then sends a “coordinator” message with the id of the newly-elected coordinator. Again, each process adds its id to the end of the message, and records the coordinator id locally.
- ❖ Once “coordinator” message gets back to initiator,
 - ❖ election is over if would-be-coordinator's id is in id-list.
 - ❖ else the algorithm is repeated (handles election failure).

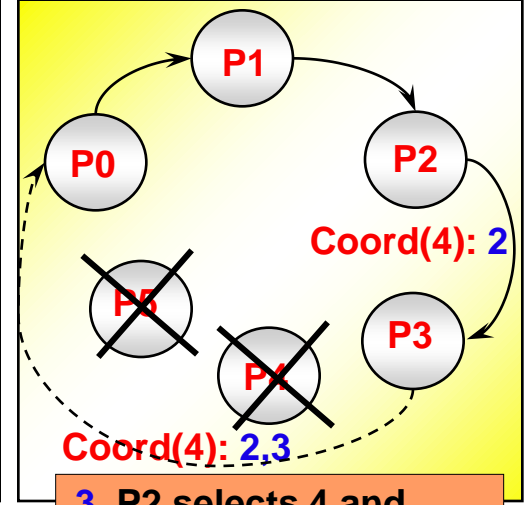
Example: Ring Election



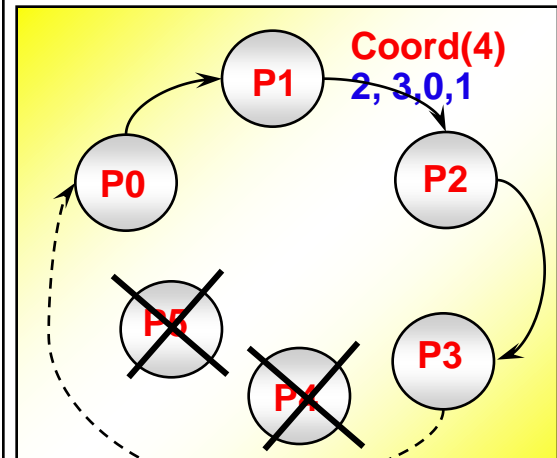
1. P2 initiates election



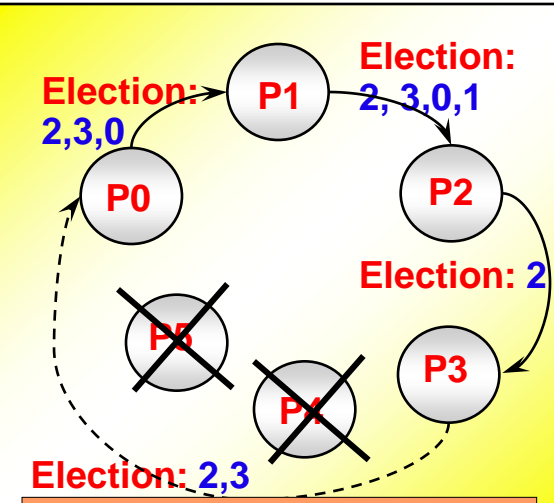
2. P2 receives "election", P4 dies



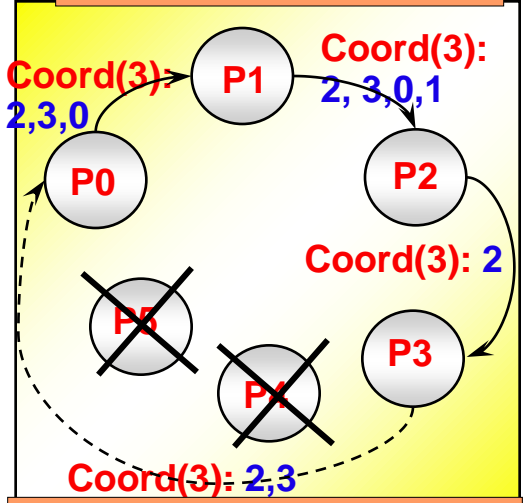
3. P2 selects 4 and announces the result



4. P2 receives "Coord", but P4 is not included



5. P2 re-initiates election



6. P3 is finally elected

Modified Ring Election

- **Supports concurrent elections – an initiator with a lower id blocks other initiators' election messages**
- **Reconfiguration of ring upon failures**
 - Can be done if all processes “know” about all other processes in the system (Membership list! – MP2)
- **If initiator non-faulty ...**
 - How many messages? $2N$
 - What is the turnaround time? $2N$
 - Size of messages? $O(N)$
- **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, timeout, then re-initiate a new election. Do the same for this successor's successor, and so on...
 - What if timeouts are too short... starts to get messy

Leader Election Is Hard

- The Election problem is a special form of the *consensus problem*
- Consensus is impossible to solve with 100% guarantee in an asynchronous system with no bounds on message delays and arbitrarily slow processes
- So is leader election in fully asynchronous system model
- Where does the modified Ring election start to give problems with the above asynchronous system assumptions?
 - p_i may just be very slow, but not faulty (yet it is not elected as leader!)
 - Also slow initiator, ring reorganization

Algorithm 3: Bully Algorithm

❖ Assumptions:

❖ Synchronous system

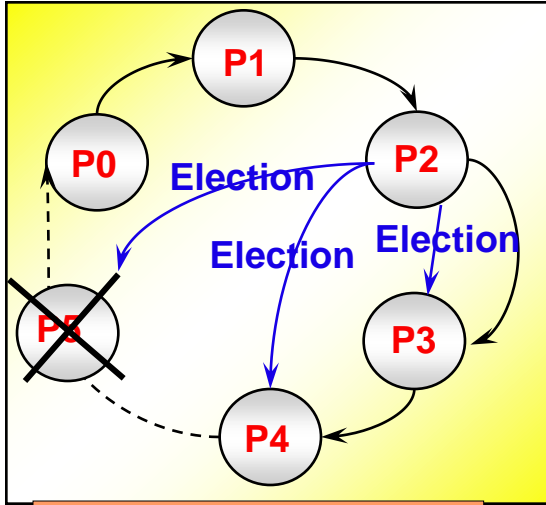
- ❖ All messages arrive within T_{trans} units of time.
- ❖ A reply is dispatched within $T_{process}$ units of time after the receipt of a message.
- ❖ if no response is received in $2T_{trans} + T_{process}$, the process is assumed to be faulty (crashed).
- ❖ $attr == id$
- ❖ Each process knows all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

Algorithm 3: Bully Algorithm

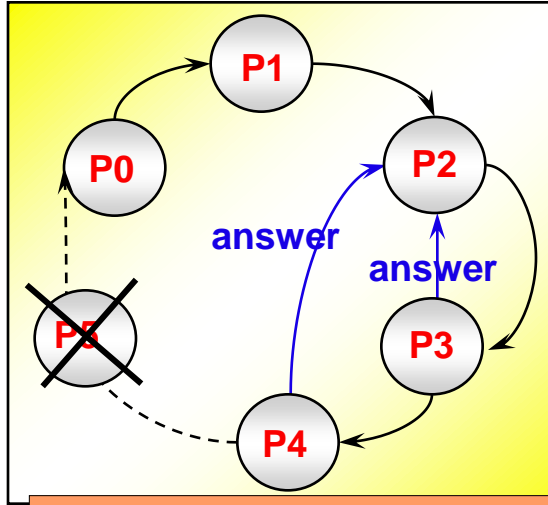
- ❖ When a process finds the coordinator has failed, if it knows its id is the highest, it elects itself as coordinator, then sends a **coordinator** message to all processes with lower identifiers.
- ❖ A process initiates election by sending an **election** message to only processes that have a higher id than itself.
 - ❑ If no answer within timeout, send coordinator message to lower id processes
 - ❑ if any answer received, then there is some non-faulty higher process → so, wait for coordinator message. If none received after another timeout, start a new election.
- ❖ A process that receives an “election” message replies with **answer** message, & starts its own election protocol (unless it has already done so)

Example: Bully Election

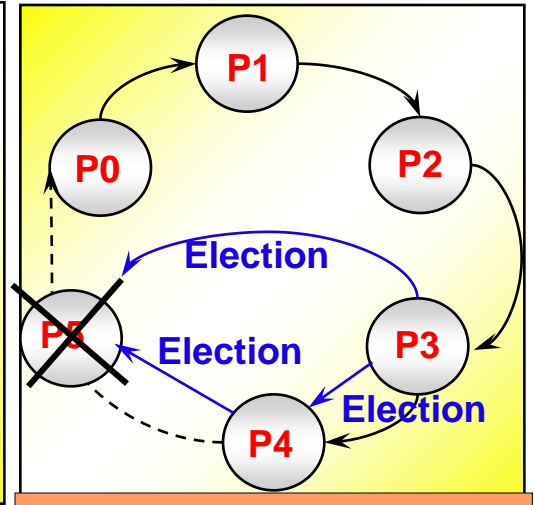
answer=OK



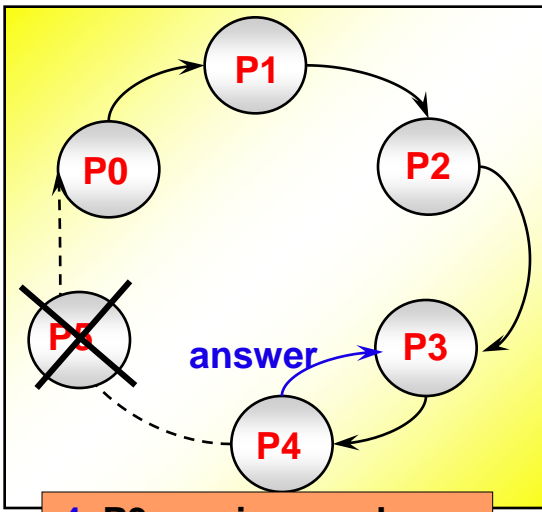
1. P2 initiates election



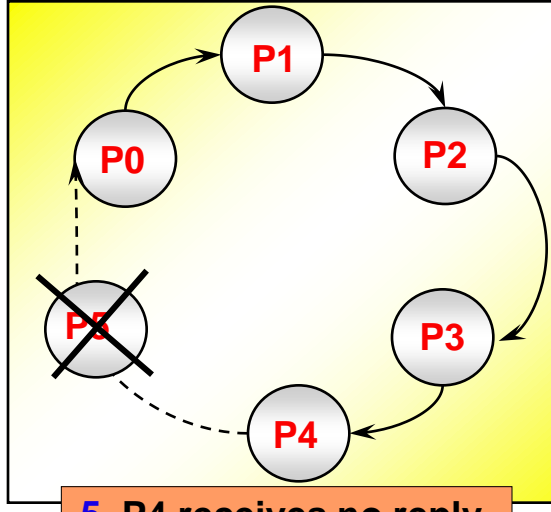
2. P2 receives answers



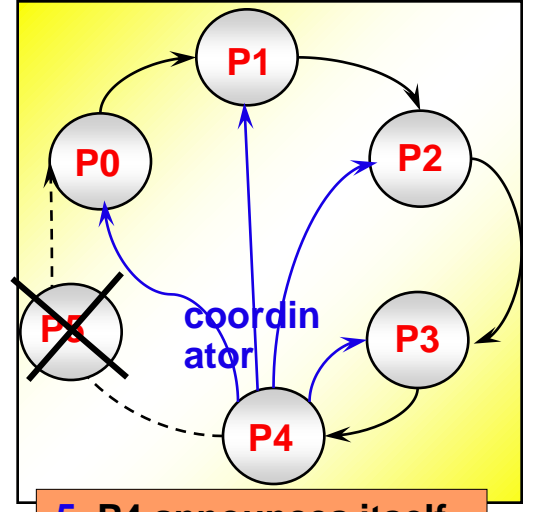
3. P3 & P4 initiate election



4. P3 receives reply



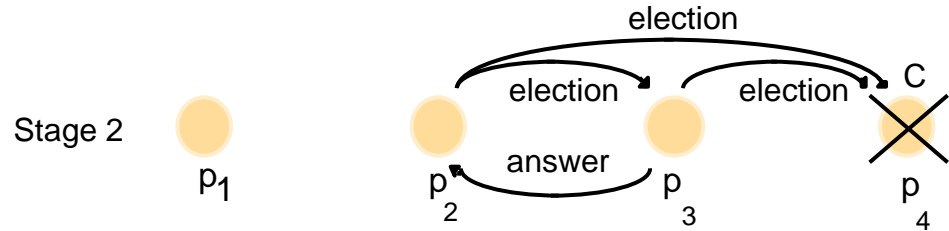
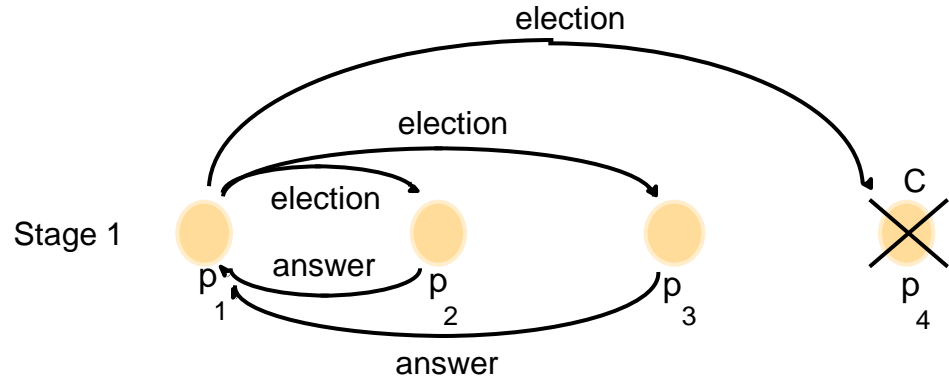
5. P4 receives no reply



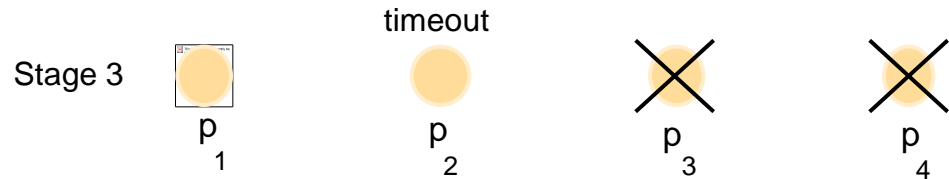
5. P4 announces itself

The Bully Algorithm with Failures

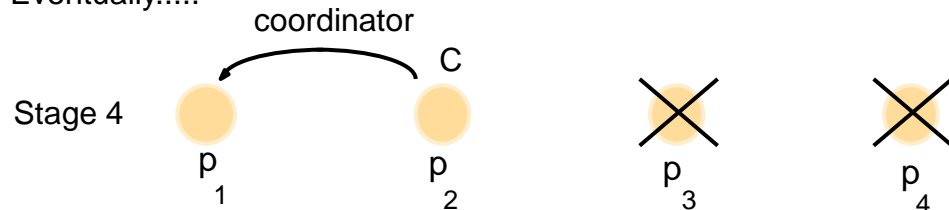
The coordinator p_4 fails and p_1 detects this



p_3 fails



Eventually.....



Analysis of The 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.

Analysis of The Bully Algorithm

- **Worst case scenario: When the process with the lowest id in the system detects the failure.**
 - N-1 processes altogether begin elections, each sending messages to processes with higher ids.
 - » i-th highest id process sends i-1 election messages
 - The message overhead is $O(N^2)$.
 - Turnaround time is approximately 5 message transmission times if there are no failures during the run:
 1. Election from lowest id process
 2. Answer to lowest id process from 2nd highest id process
 3. Election from 2nd highest id process
 4. Timeout for answers @ 2nd highest id process
 5. Coordinator from 2nd highest id process

Summary

- **Coordination in distributed systems requires a leader process**
- **Leader process might fail**
- **Need to (re-) elect leader process**
- **Three Algorithms**
 - **Ring algorithm**
 - **Modified Ring algorithm**
 - **Bully Algorithm**

Readings and Announcements

- **For Thursday: 15.2**
- **Midterm Exam is Oct 16th during class hours**
 - All material until Lecture 12 (this Thursday)
 - Location may be same or different (watch Piazza)
- **MP2**
 - By now, you should have a working failure detector. If you have a fully working system, you're doing good!