Research Group
Distributed Systems

Christian-Albrechts-Universität zu Kiel
Technische Fakultät

Distributed Systems
Mutual Exclusion & Election

Olaf Landsiedel

Problem

- PreLab: 38 submissions
  - Who submitted alone?
  - ->Nearly 80 people in the labs

- We will have two lab sessions
  - Most likely, both: 14 to 16, Thursday
  - Alternative, second one: 16 to 18 Thursday
  - Depends on university: need to find rooms

Note

- No lecture this week Thursday
  - See course schedule
  - Please focus on lab 1

Last Time

- Processes
- Architectures

Today

- Mutual Exclusion
  - How to coordinate between processes that access the same resource?
- Election Algorithms
  - Here, a group of entities elect one entity as the coordinator for solving a problem

Mutual Exclusion

Mutual Exclusion

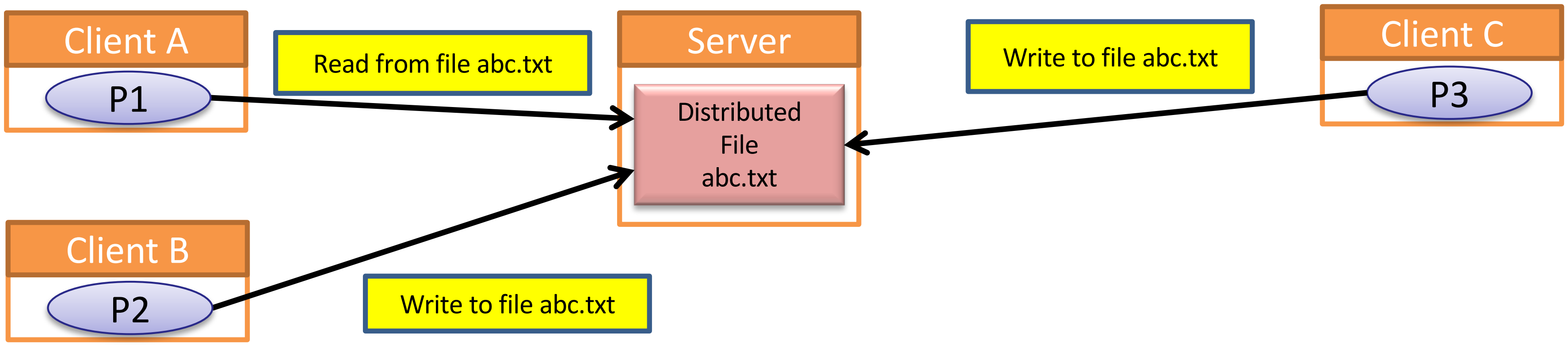
- What is it?
  - Manages access to (for example) resources
  - Goal: unique access
- Why do we need mutual exclusion?

Why Mutual Exclusion?

- Example: Bank's Servers in the Cloud:
  Two simultaneous deposits of $10,000 into your bank account, each from one ATM.
  What can go wrong?
  - Both ATMs read initial amount of $1000 concurrently from the bank's cloud server
  - Both ATMs add $10,000 to this amount (locally at the ATM)
  - Both write the final amount to the server
  - What's wrong?
- The ATMs need *mutually exclusive* access to your account

Need for Mutual Exclusion

- Distributed processes need to coordinate to access shared resources
- Example: Writing a file in a Distributed File System



In uniprocessor systems, mutual exclusion to a shared resource is provided through shared variables or operating system support: locks, mutexes, semaphores, …

However, such support is insufficient to enable mutual exclusion of distributed entities

In a Distributed System, processes coordinate access to a shared resource by passing messages to enforce *distributed mutual exclusion*

Requirements

- Safety
  - At most one process may execute in ***critical section (CS)*** at any time
- Liveness
  - Every request for a CS is eventually granted
- Ordering (desirable)
  - Requests are granted in the order they were made
  - Result: fairness

Performance Evaluation Criteria

- What makes a good algorithm?

- Number of messages send
  - To acquire access, to release access
- Delay
  - To acquire access, to release access

- -> Throughput
  - Number of operations per second

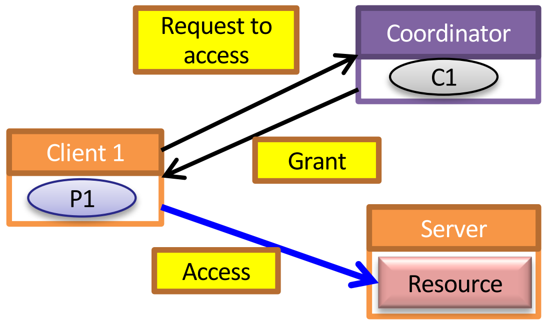
Assumptions/System Model

- Reliable communication
  - Each pair of processes is connected by reliable channels
    - such as TCP
  - But with arbitrary delay
  - We will later relax this
- Ordering
  - Messages are eventually delivered to recipient in FIFO order.
- Processes do not fail
  - We will later relax this

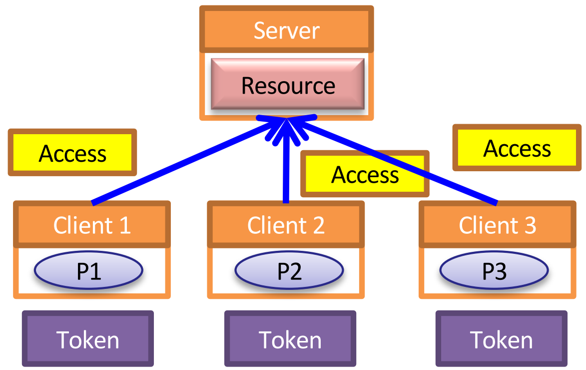
Types of Distributed Mutual Exclusion

- Mutual exclusion algorithms are classified into two categories

1. Permission-based Approaches
   + A process, which wants to access a shared resource, requests the permission from one or more coordinators



2. Token-based Approaches
   + Each shared resource has a token
   + Token is circulated among all the processes
   + A process can access the resource if it has the token



Overview

- Mutual Exclusion
  - **Permission-based Approaches**
  - Token-based Approaches

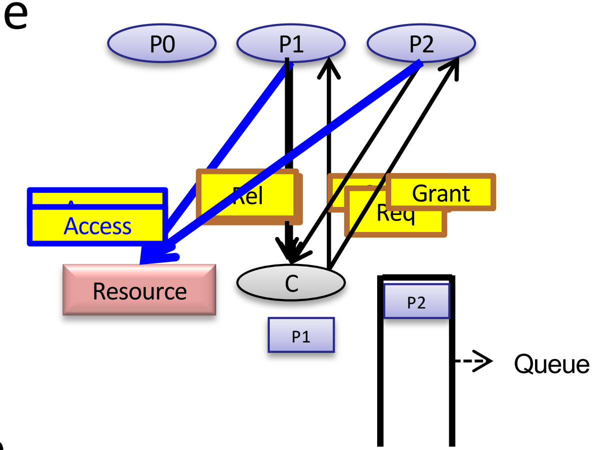
Permission-based Approaches

- There are three types of permission-based mutual exclusion algorithms
  – Centralized Algorithms
  – Decentralized Algorithms
  – Distributed Algorithms

- We will study an example of each type of algorithm

a. A Centralized Algorithm

- One process is elected as a coordinator (C) for a shared resource
- Coordinator maintains a Queue of access requests
- Whenever a process wants to access the resource, it sends a request message to the coordinator to access the resource
- When the coordinator receives the request:
  - If no other process is currently accessing the resource, it grants the permission to the process by sending a “grant” message
  - If another process is accessing the resource, the coordinator queues the request, and does not reply to the request
- The process releases the exclusive access after accessing the resource
- The coordinator will then send the “grant” message to the next process in the queue



Discussion: Centralized Algorithm

- Blocking vs. non-blocking requests
  - The coordinator can block the requesting process until the resource is free
  - Otherwise, the coordinator can send a "permission-denied" message back to the process
    - The process can poll the coordinator at a later time, or
    - The coordinator queues the request. Once the resource is released, the coordinator will send an explicit "grant" message to the process

- The algorithm guarantees mutual exclusion

- Advantages?
  - is simple to implement

- Disadvantages?
  - Fault-tolerance
    - Centralized algorithm is vulnerable to a single-point of failure (at coordinator)
      - Processes cannot distinguish between dead coordinator and request blocking

  - Performance
    - Bottleneck
    - In a large system, single coordinator can be overwhelmed with requests

Last Time

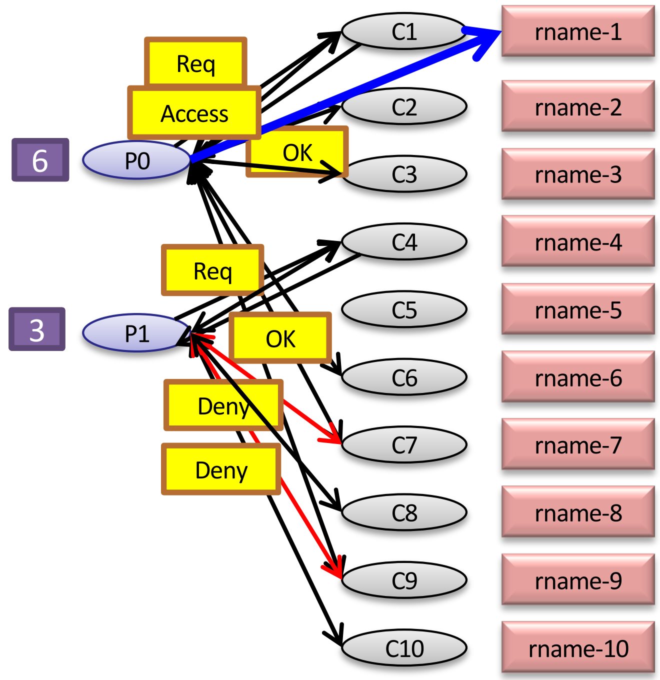
- Processes
- Mutual Exclusion

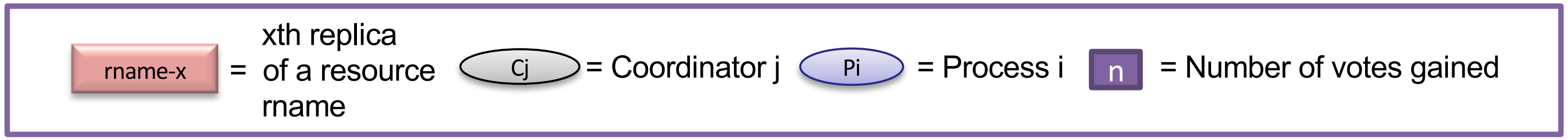
b. A Decentralized Algorithm

- To avoid the drawbacks of the centralized algorithm
  - decentralized mutual exclusion algorithm
- Assumptions
  - Multiple coordinators (n), each with a replica of the resource
  - Requester queries m coordinators
    - They reply concurrently: access granted or denied
- Approach:
  - Whenever a process wants to access the resource, it will have to get a majority vote from $m > n/2$ coordinators
  - If a coordinator does not want to vote for a process (because it has already voted for another process), it will send a "permission-denied" message to the process

A Decentralized Algorithm: Example

- If n=10 and m=6, then a process needs at-least 6 votes to access the resource



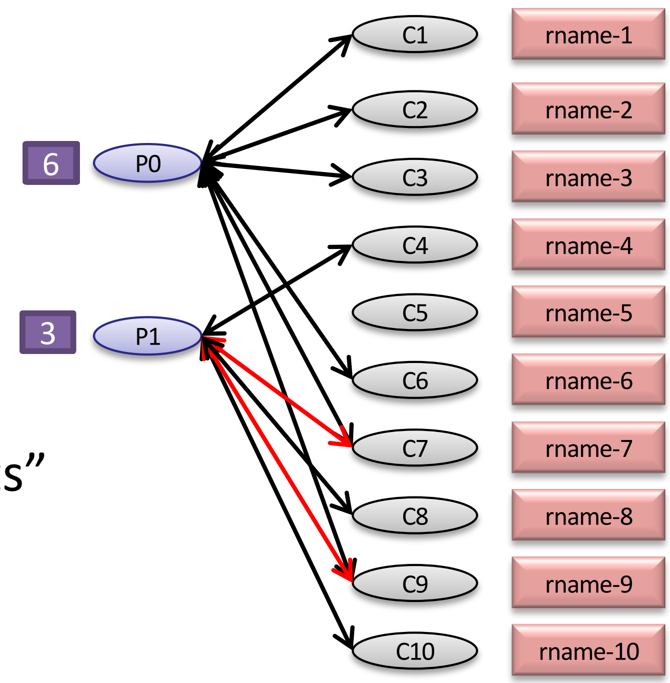


Fault-tolerance in Decentralized Algorithm

- The decentralized algorithm assumes that the coordinator recovers quickly from a failure
- However, the coordinator would have reset its state after recovery
  - Coordinator could have forgotten any vote it had given earlier
- Hence, the coordinator may incorrectly grant permission to the processes
  - Mutual exclusion cannot be deterministically guaranteed
  - But, the algorithm *probabilistically* guarantees mutual exclusion

Fault-tolerance in Decentralized Algorithm

- What can we do here?
  - Solution: Get more votes
  - For majority:
    - m > n/2
  - Choosing
    - M > n/2 + x
    - x = 1, 2, 3, ..
    - Will increase fault tolerance
    - If a coordinate reboots and “forgets” previous votes
  - Downside of this approach?
    - Overhead
      - -> need more votes
      - -> need to send more messages



Ricart & Agrawala's Algorithm

Basic Idea:

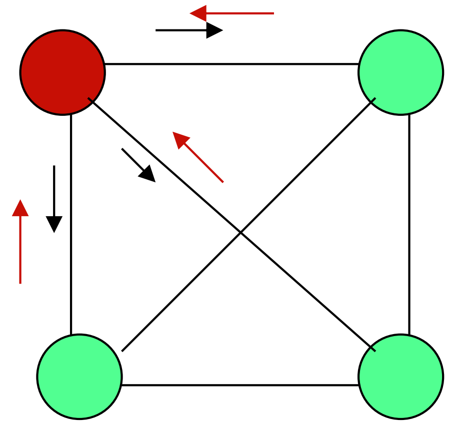
1. Broadcast a timestamped ***request*** to all.

2. Upon receiving a request, send ***ack*** if

-You do not want to enter your Critical Section (CS), or

-You are trying to enter your CS, but your timestamp is **larger** than that of the sender.

(If you are already in CS, then buffer the request)

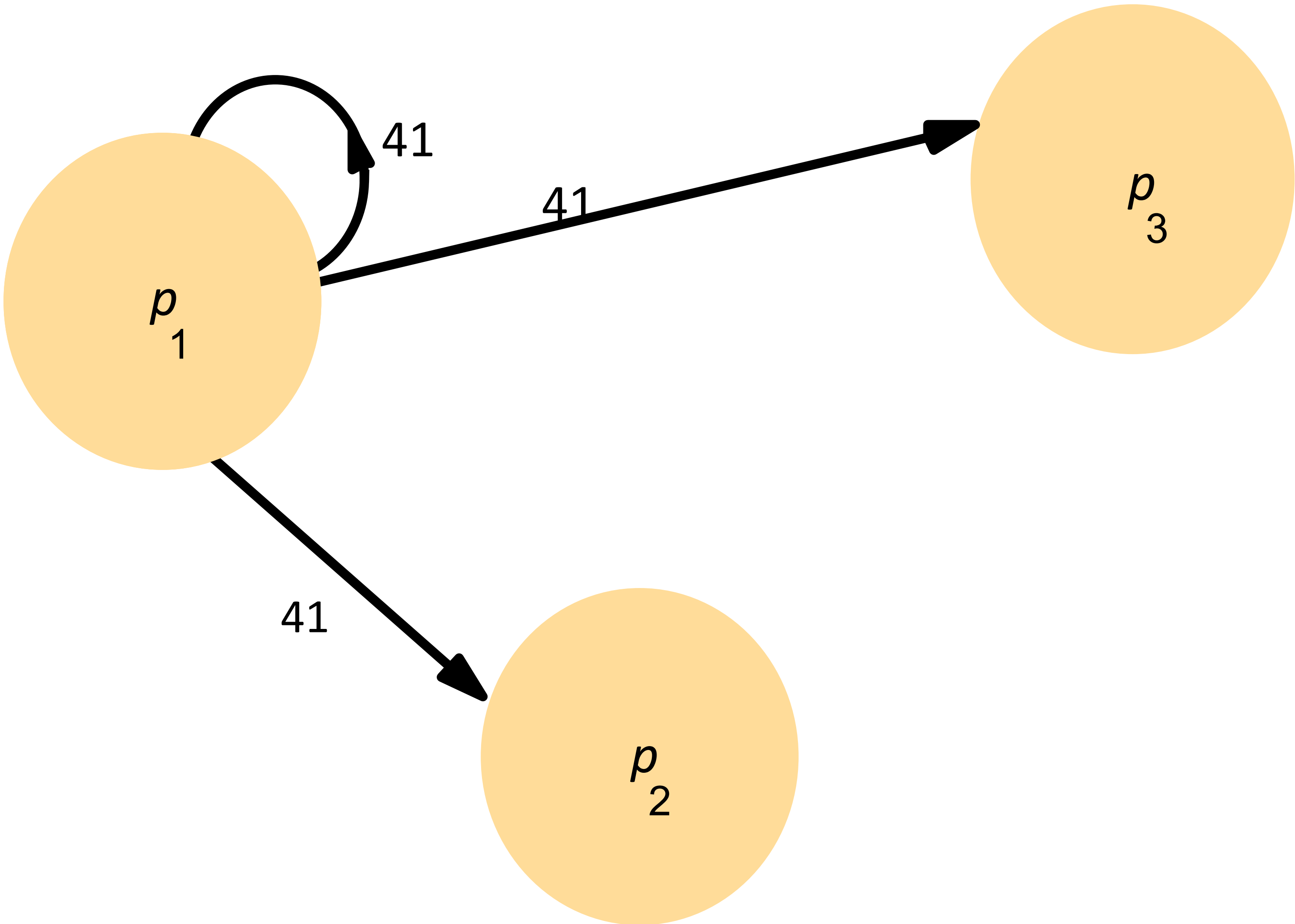


3. **Enter CS**, when you receive ***ack*** *from all*.

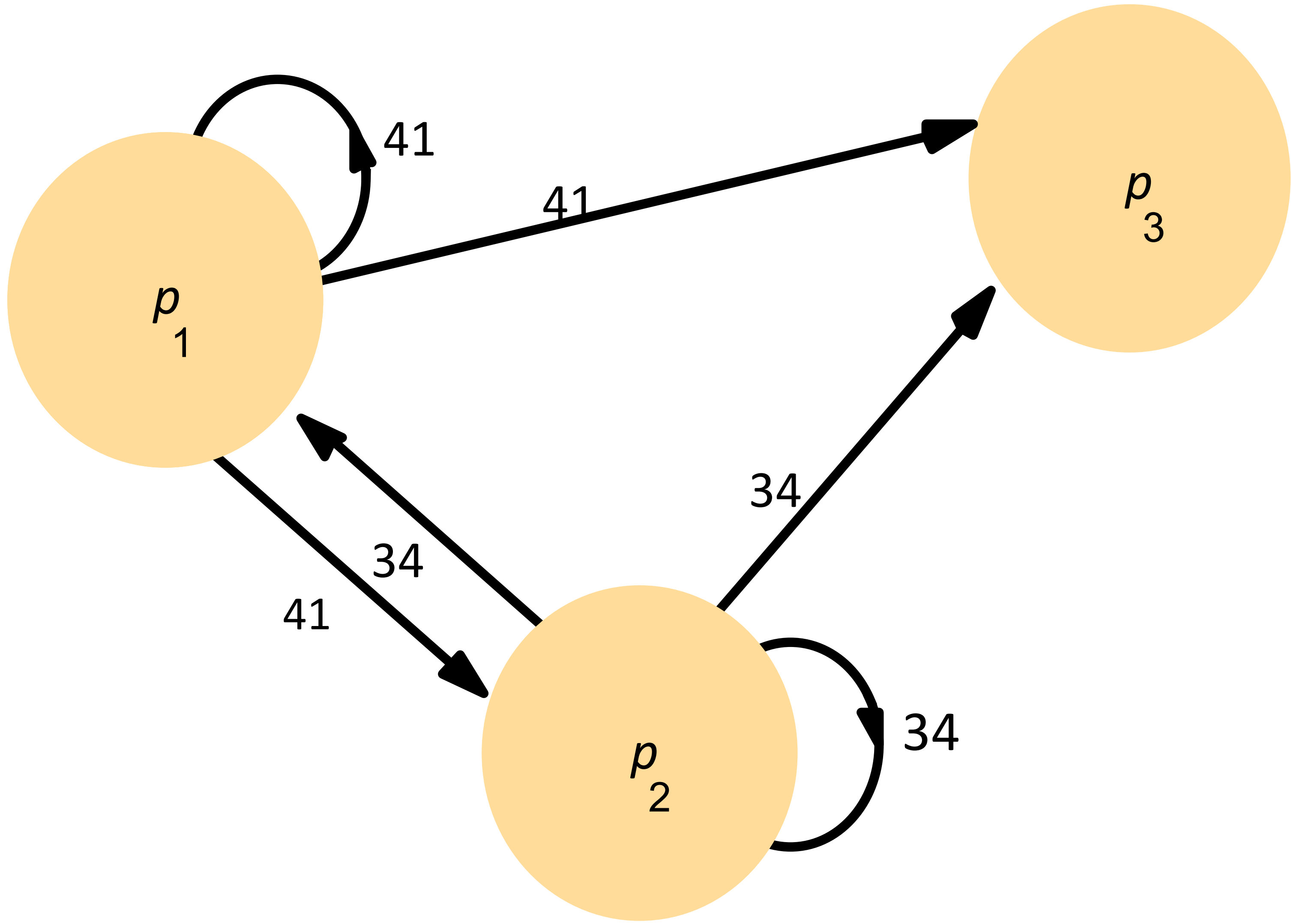
4. Upon **exit from CS**, send ***ack*** to each pending request(see comment) before making a new request.

(No release message is necessary)

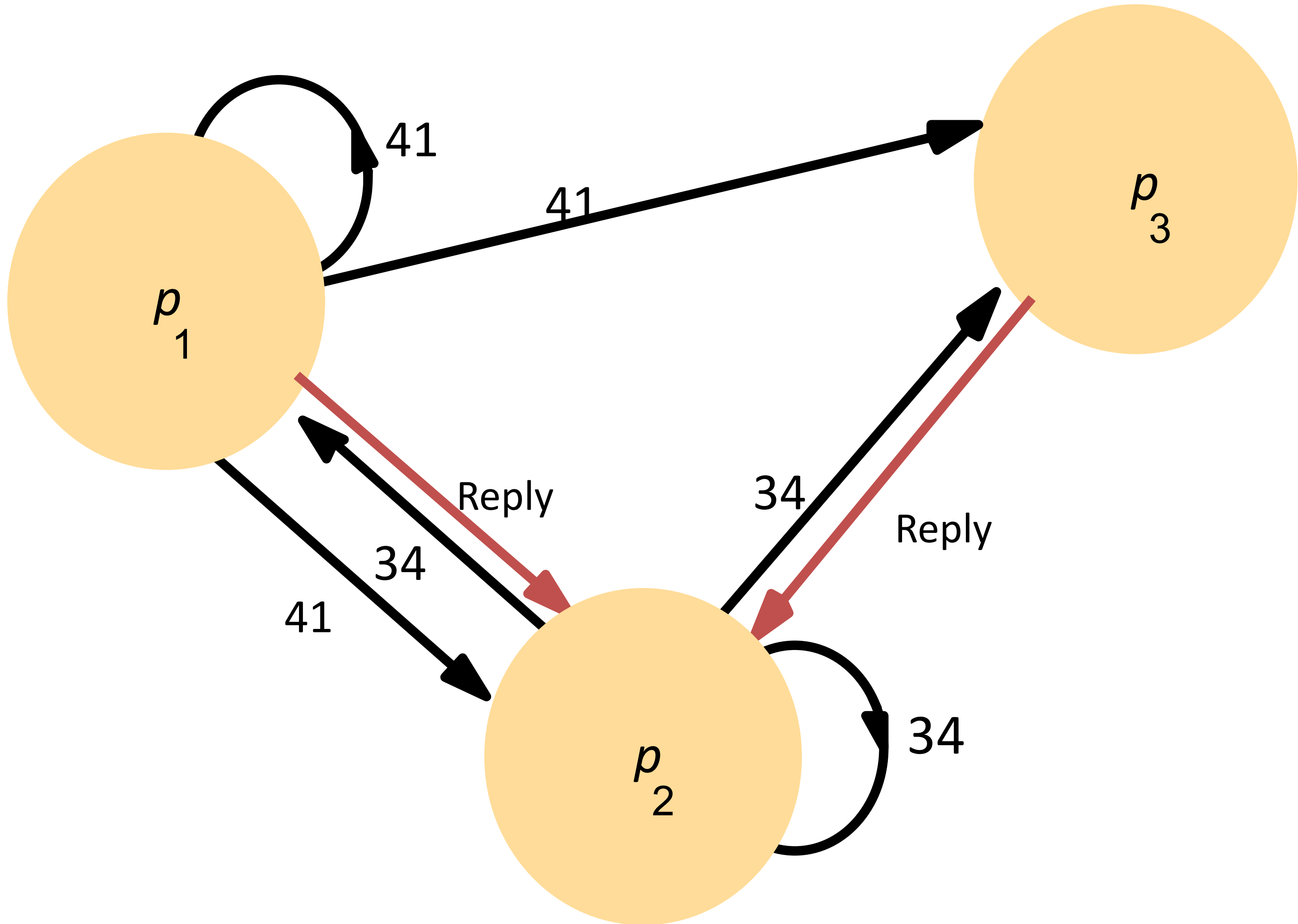
Ricart & Agrawala's Algorithm



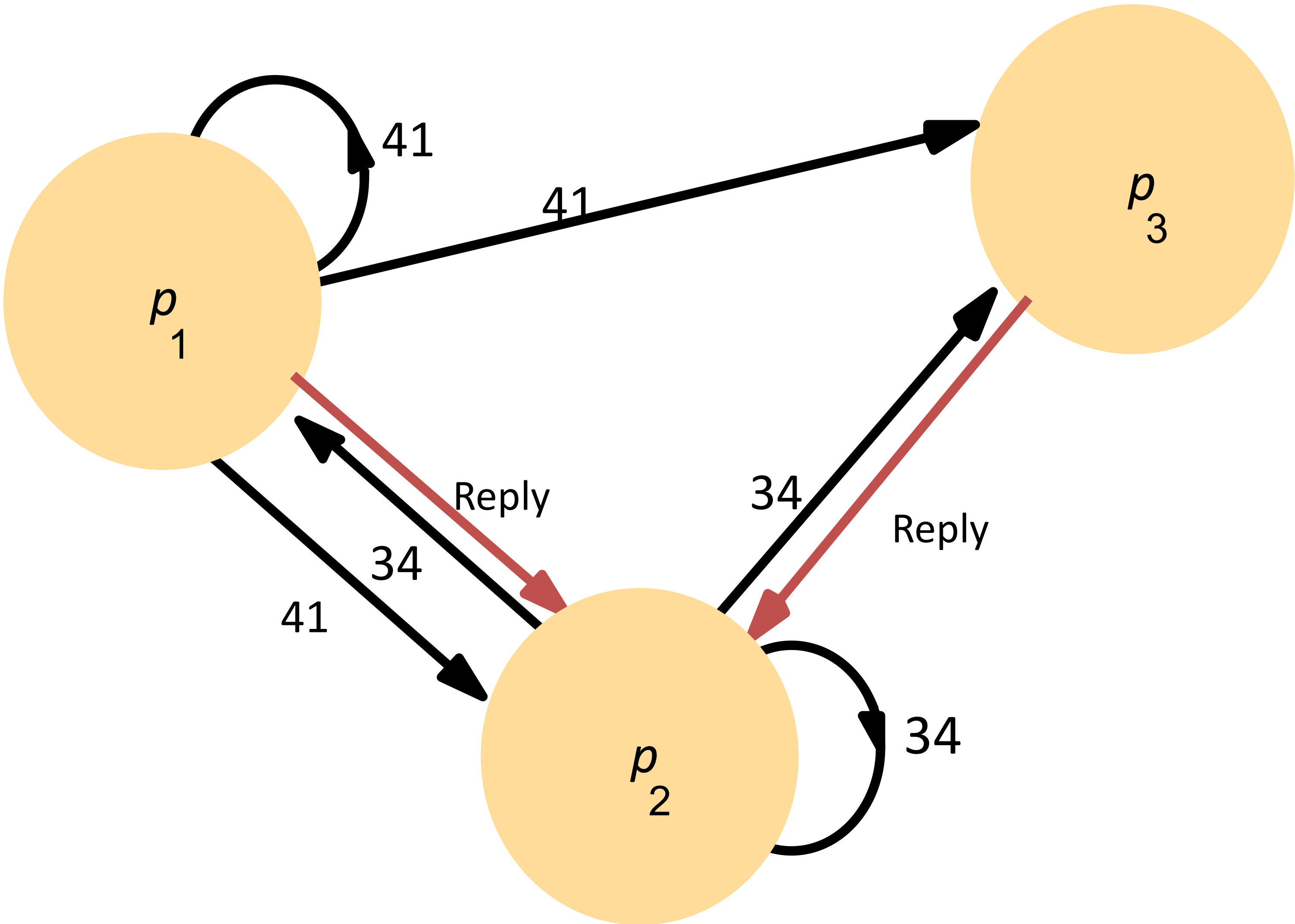
Ricart & Agrawala's Algorithm



Ricart & Agrawala's Algorithm



Ricart & Agrawala's Algorithm



Analysis: Ricart & Agrawala

- Bandwidth
- 2(N-1) messages per entry operation
  - N-1 unicasts for the multicast request + N-1 replies (send over the network)

Discussion: Permission based Approaches

- Centralized - Decentralized – Distributed
  - Can you summarize each approach?
  - Can we order them
    - Efficiency? Robustness to failure? Complexity?

Centralized -> Decentralized -> Distributed

Most efficient	Least efficient
Least robust	Most robust
Least complex	Most complex

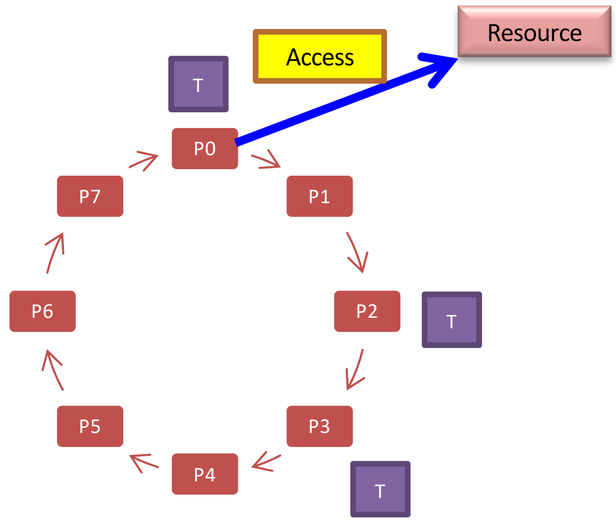
Overview

- Mutual Exclusion
  - Permission-based Approaches
  - **Token-based Approaches**

Token Ring

- In the Token Ring algorithm, each resource is associated with a token
- The token is circulated among the processes
- The process with the token can access the resource

- Circulating the token among processes:
  - All processes form a logical ring where each process knows its next process
  - One process is given a token to access the resource
  - The process with the token has the right to access the resource
  - If the process has finished accessing the resource OR does not want to access the resource:
    - it passes the token to the next process in the ring



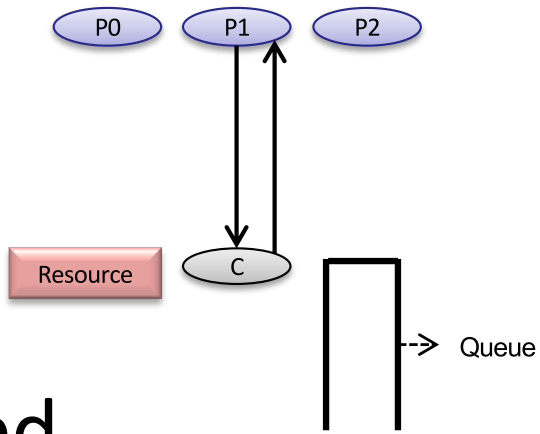
Discussion about Token Ring

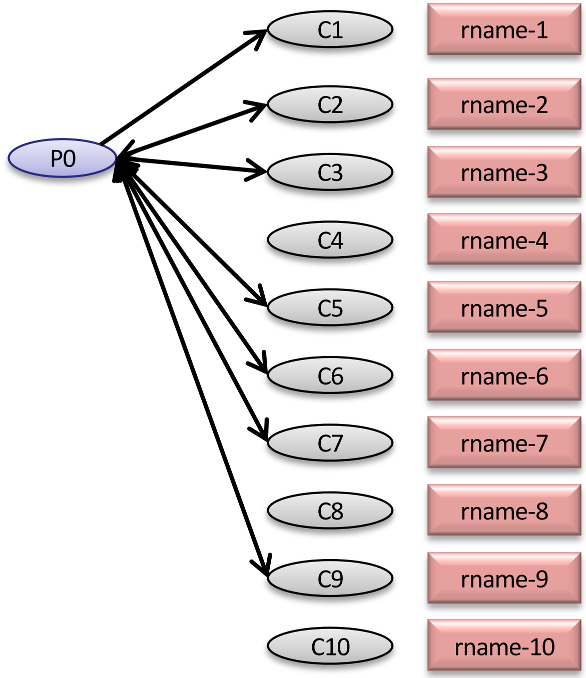
Pros and Cons?

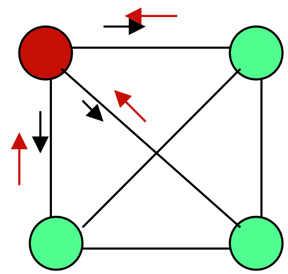
- ✓ Token ring approach provides deterministic mutual exclusion
  - There is one token, and the resource cannot be accessed without a token
- ✓ Token ring approach avoids starvation
  - Each process will receive the token
- ✗ Token ring has a high-message overhead
  - When no processes need the resource, the token circulates at a high-speed
- ✗ If the token is lost, it must be regenerated
  - Detecting the loss of token is difficult since the amount of time between successive appearances of the token is unbounded
- ✗ Dead processes must be purged from the ring
  - ACK based token delivery can assist in purging dead processes

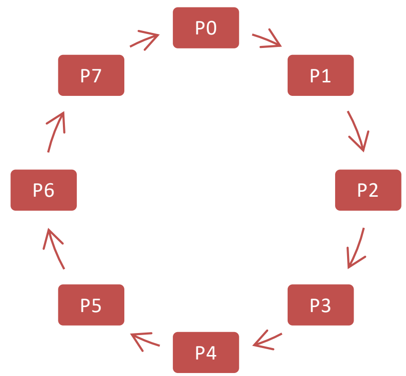
Summary: Mutual Exclusion

- Centralized
- Decentralized
- Distributed
- Token Ring









Questions

- Which algorithm is the easiest to implement?
- Which algorithm is most efficient?
  - Sends the least number of messages?
- Which algorithm is most robust to node failure?
- Which algorithm is the most scalable?
- Which algorithm makes the strongest assumptions about the system?

http://olafland.polldaddy.com/s/mutual-exclusion

Comparison of Mutual Exclusion Algorithms

Algorithm	Delay before a process can access the resource (in message times)	Number of messages required for a process to access and release the shared resource	Problems
Centralized	2	3	• Coordinator crashes
Decentralized	2mk	2mk + m; k=1,2,…	• Large number of messages
Distributed	2 (n-1)	2 (n-1)	• Crash of any process
Token Ring	0 to (n-1)	1 to ∞	• Token may be lost • Ring can cease to exist since processes crash

- Assume that:

  n = Number of processes in the distributed system

  For the Decentralized algorithm:

  m = minimum number of coordinators who have to agree for a process to access a resource

  k = average number of requests made by the process to a coordinator to request for a vote

Election Algorithms

Not this kind of election

Election Algorithms

- What is an election in a distributed system?

- What is it good for?

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:
  - Group of cloud servers replicating a file need to elect one among them as the primary replica that will communicate with the client machines
  - This how many google services (gmail etc.) work(ed)
    - 4 replicas of your data, one primary replica
- Example 3:
  - Group of NTP servers: who is the root server?

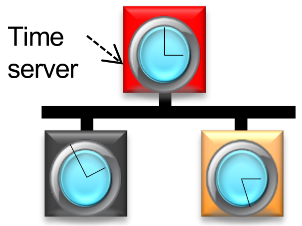
What is Election?

- 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

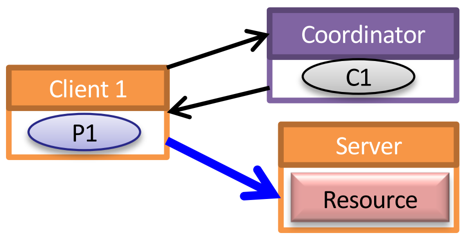
Election in Distributed Systems

- Many distributed algorithms require one process to act as a coordinator
  - Typically, it does not matter which process is elected as the coordinator
  - See Labs ;-)

- Example algorithms where coordinator election is required



Berkeley Clock Synchronization Algorithm



A Centralized Mutual Exclusion Algorithm

Election Process: Assumptions

- Initiation
  - Any process in a DS can initiate the election algorithm that elects a new coordinator
  - Multiple processes can call an election simultaneously.
    - All of them together must yield a single leader only
- Termination
  - At the termination of the election algorithm, the elected coordinator process should be unique
- Naming
  - Every process may know the process ID of every other processes, but it does not know which processes have crashed

Election Process: Goal

- At the end of the election protocol
  - Process with best (highest) election attribute value is elected
  - Attribute examples: leader has highest id or address, or fastest cpu, least CPU load, or most disk space, or most number of files, …

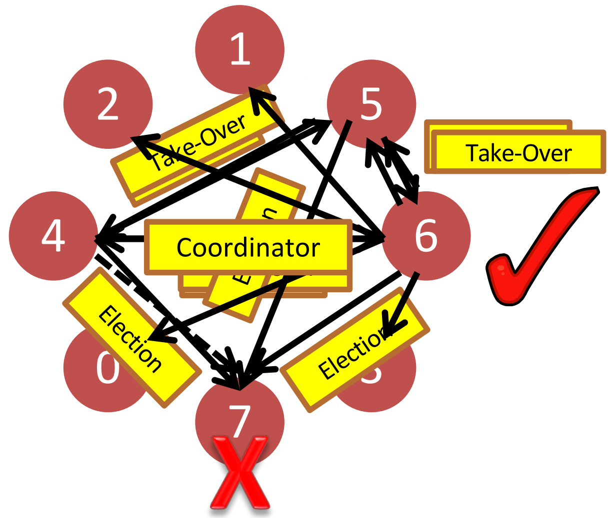
Election Algorithms

- We will study two election algorithms
  - Bully Algorithm
  - Ring Algorithm

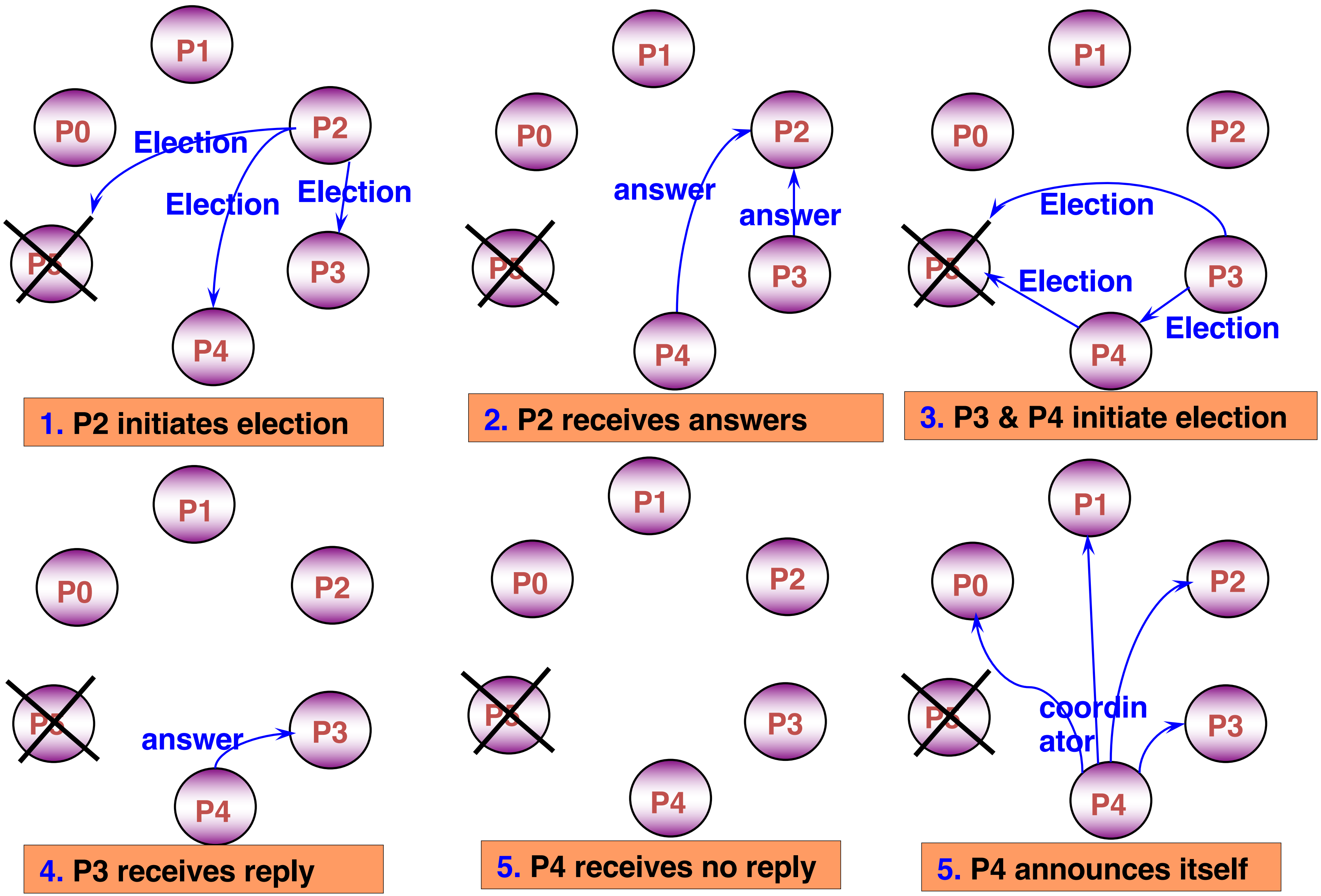
1. Bully Algorithm

- A process initiates election algorithm when it notices that the existing coordinator is not responding
- Process Pi calls for an election as follows:

1. **P_i** sends an "Election" message to all processes with higher process IDs
2. When process **P_j** with **j>i** receives the message, it responds with a "Take-over" message. **P_i** no more contests in the election
   i. Process **P_j** re-initiates another call for election. Steps 1 and 2 continue
3. If no one responds, **P_i** wins the election. **P_i** sends "Coordinator" message to every process



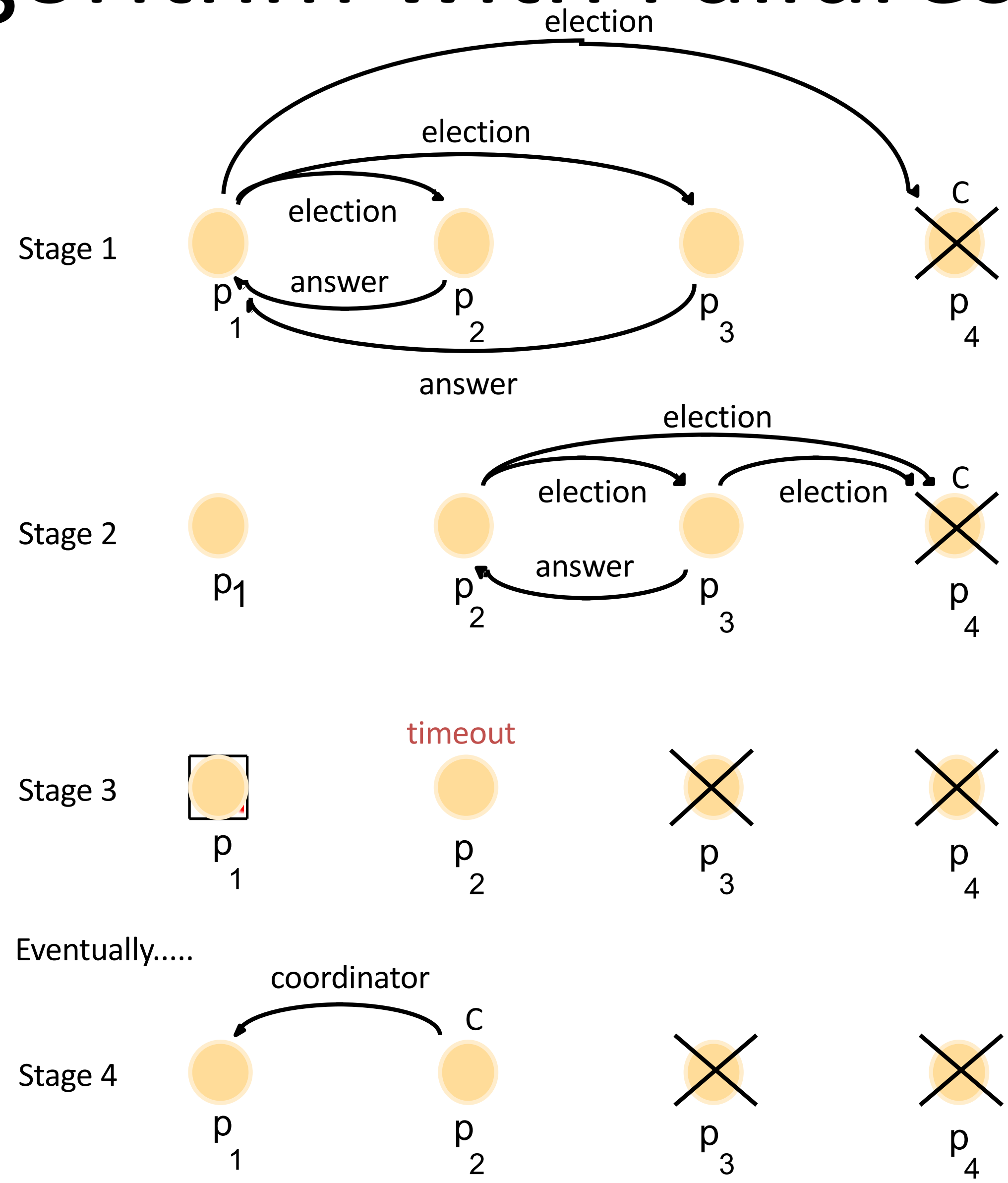
Example: Bully Election



The Bully Algorithm with Failures

The coordinator p_4 fails and p_1 detects this

p_3 fails



Analysis of The Bully Algorithm

- How many messages do we have to send
  - Best case scenario?
  - Worst case scenario?

- 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 $O(N^2)$.

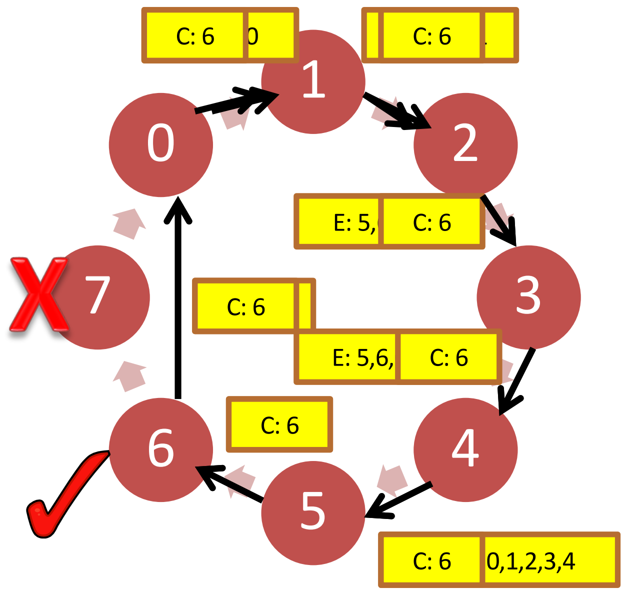
2. Ring Algorithm

- This algorithm is generally used in a ring topology
- When a process Pi detects that the coordinator has crashed, it initiates an election algorithm

1. P_i builds an "Election" message (**E**), and sends it to its next node. It inserts its ID into the Election message

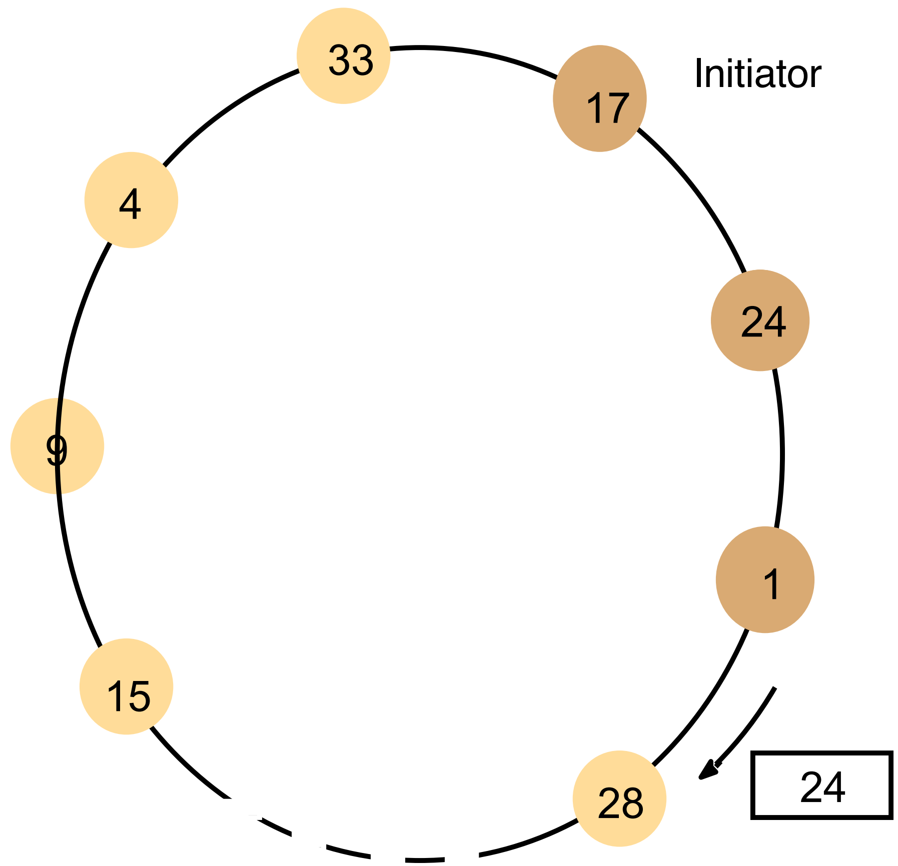
2. When process P_j receives the message, it appends its ID and forwards the message
   i. If the next node has crashed, P_j finds the next alive node

3. When the message gets back to the process that started the election:
   i. it elects process with highest ID as coordinator, and
   ii. changes the message type to "Coordination" message (**C**) and circulates it in the ring



Ring-Based Election: Analysis

- **Need to make two rounds around the ring.**
- In the example: The election was started by process 17.
  The highest process identifier encountered so far is 24.
  - (final leader will be 33)
- -> 2n-1 messages



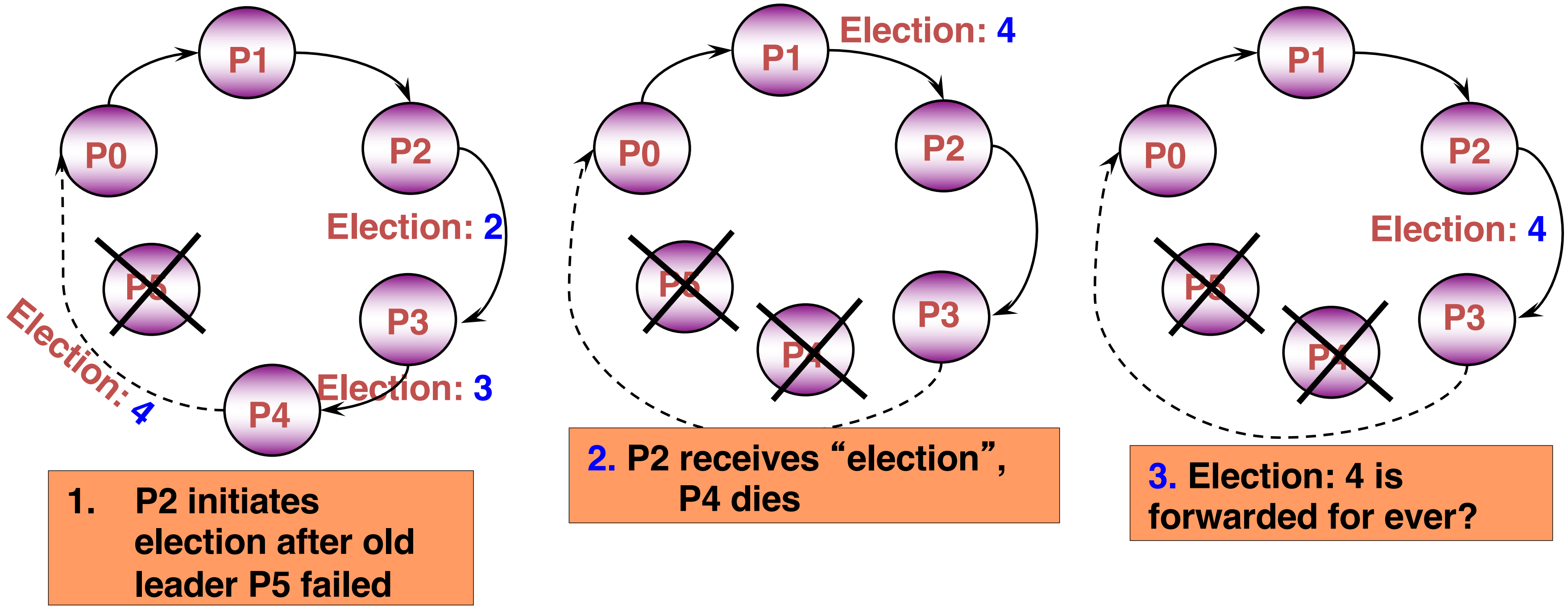
Correctness?

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 with Failure



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

Does not satisfy liveness

Need to add timeouts etc. to restart process and fix ring

Election Algorithms

- Which algorithm is more efficient (sends less messages)?
  - Bully or Ring Algorithm?

- Which algorithm makes stronger assumptions about the system?
  - Bully or Ring Algorithm?

olafland.polldaddy.com/s/election

Comparison of Election Algorithms

Algorithm	Number of Messages for Electing a Coordinator	Problems
Bully Algorithm	$O(n^2)$	• Large message overhead
Ring Algorithm	2n-1	• An overlay ring topology is necessary

- Assume that:
  n = Number of processes in the distributed system

Summary of Election Algorithms

- Election algorithms are used for choosing a unique process that will coordinate an activity
- At the end of the election algorithm, all nodes should uniquely identify the coordinator
- We studied two algorithms for election
  - Bully algorithm
    - Processes communicate in a distributed manner to elect a coordinator
  - Ring algorithm
    - Processes in a ring topology circulate election messages to choose a coordinator

Next Time

- Naming
  - Everything needs a name to be identifiable
  - Names on different layers
    - Host name vs. IP address

Questions?

In part, inspired from / based on slides from
Paul Krzyzanowski, Vinay Kolar, Indranil Gupta,
K. Nahrtstedt, S. Mitra, N. Vaidya, M. T. Harandi,
J. Hou, Sukumar Ghosh, and many others