Distributed Systems: Election Algorithms

Dr Aidan Murphy
aidan.murphy@ucd.ie

School of Computer Science and Informatics
University College Dublin
Ireland

From Previous Lecture…

- Some Distributed Systems require that one of the processes play a particular role.
  - For example, selecting a process to play the role of “central-server” in a variant of the Centralized Mutual Exclusion algorithm.

- In such situations, we need to employ a mechanism for selecting the “leader” process.
  - This mechanism must allow all relevant processes to participate in the choice.
  - It must also produce a single choice that is accepted by all the processes.
  - Once chosen the “leader” performs the assigned role until either they “retire” or fail.
- We term such a mechanism an Election Algorithm.

Election Algorithms

- In this course we will consider two algorithms:
  - The Ring Algorithm
  - The Bully Algorithm

- Both algorithms use the ID of the processes to reach agreement.
  - Each ID is basically a (large) integer number

- We measure the performance of these algorithms by:
  - Their total network bandwidth utilization, which is proportional to the total number of messages sent.
  - The turnaround time for the algorithm, which is the number of serialized message transmission times between the initiation and termination of a single run.

Ring Algorithm

- The purpose of the algorithm, developed by Chang and Roberts in 1979, is to elect a single process, known as the coordinator, which is the process with the largest ID.

- Processes are organized in a logical ring.
  - Each process receives messages from their neighbour on the left and sends messages to their neighbour on the right.
  - Each process stores a local "active list" that contains the process ID of each active process in the system.

Ring Algorithm

The ring algorithm uses the same ring arrangement as in the token ring mutual exclusion algorithm, but does not employ a token. Processes are physically or logically ordered so that each knows its successor.

If any process detects failure, it constructs an election message with its process I.D. (e.g. network address and local process I.D.) and sends it to its successor.

If the successor is down, it skips over it and sends the message to the next party. This process is repeated until a running process is located.

At each step, the process adds its own process I.D. to the list in the message.

Ring Algorithm

Eventually, the message comes back to the process that started it:

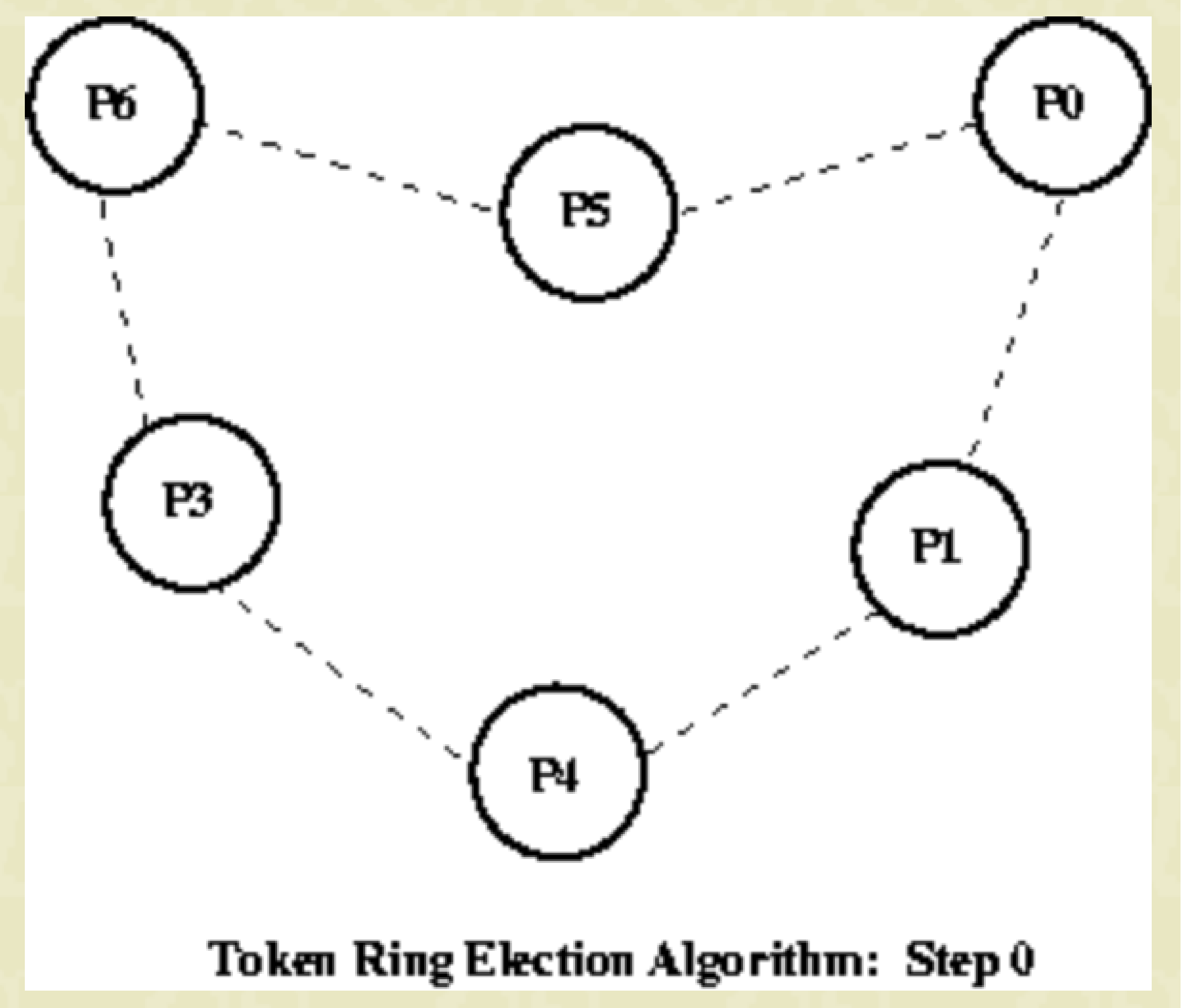
The process sees its ID in the list.

It changes the message type to coordinator.

The list is circulated again, with each process selecting the highest numbered ID in the list to act as coordinator.

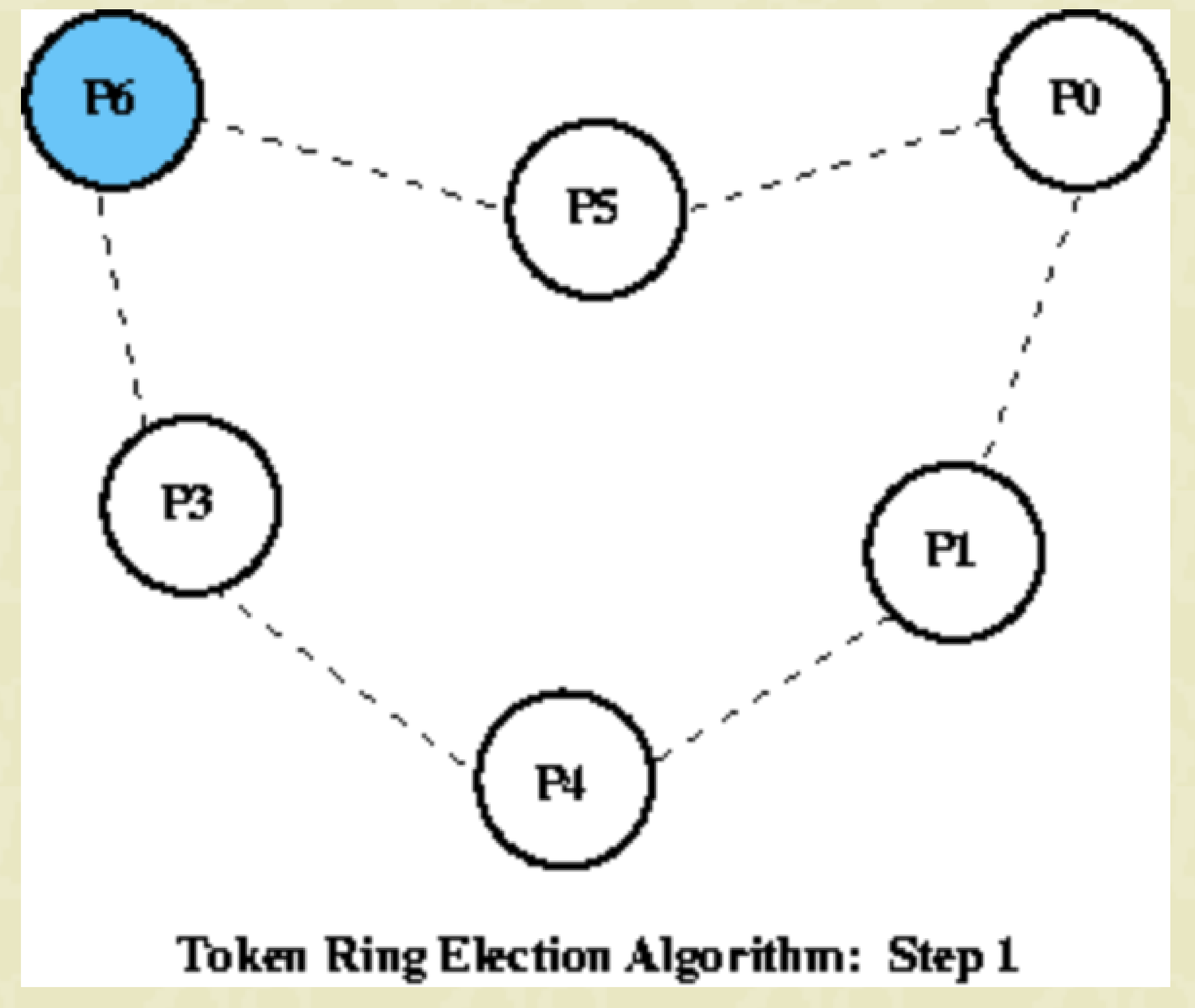
When the coordinator message has circulated fully, it is deleted.

Example



We start with 6 processes, connected in a logical ring. Process 6 is the leader, as it has the highest number.

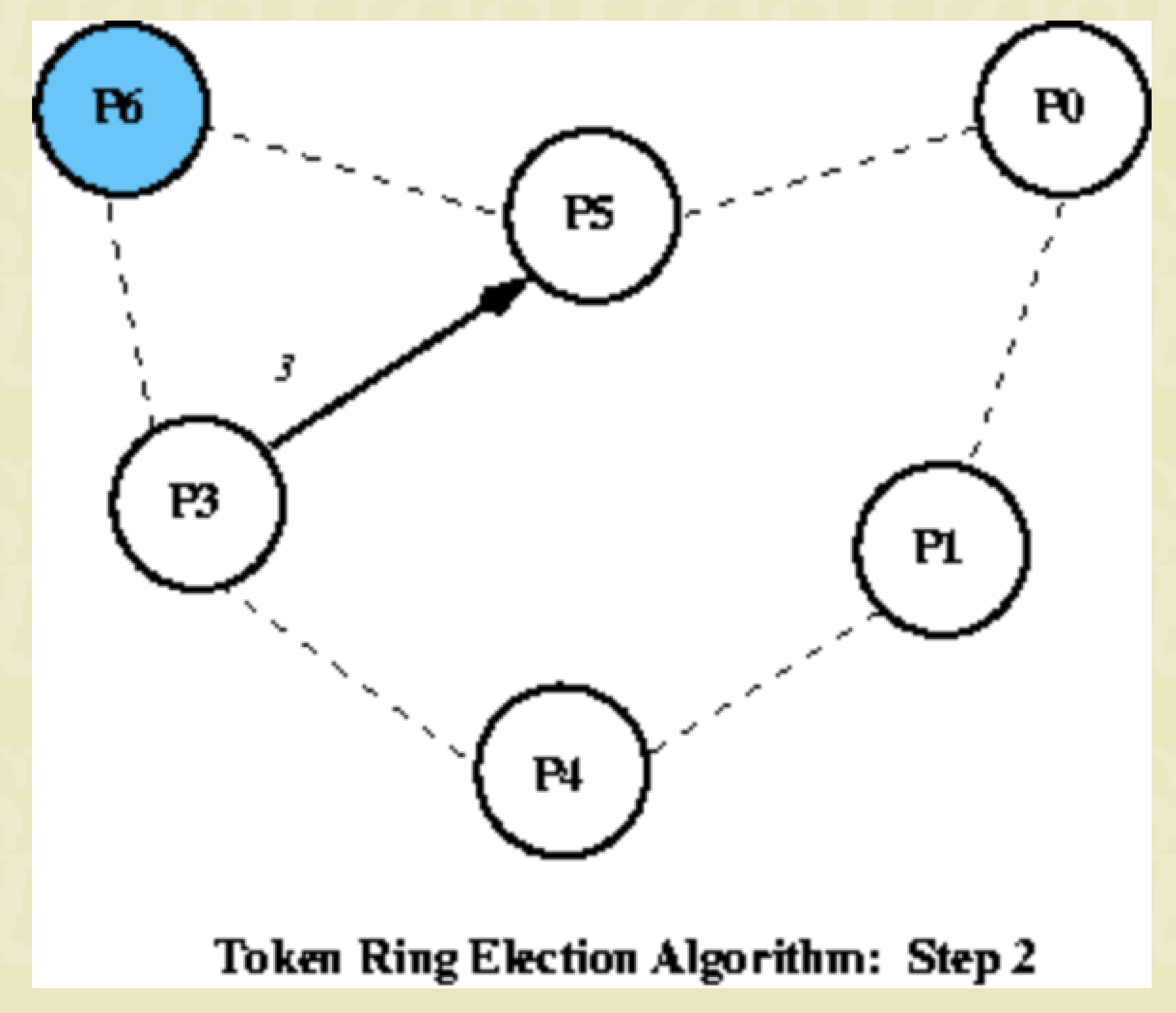
Example



Token Ring Election Algorithm: Step 1

Process 6 fails.

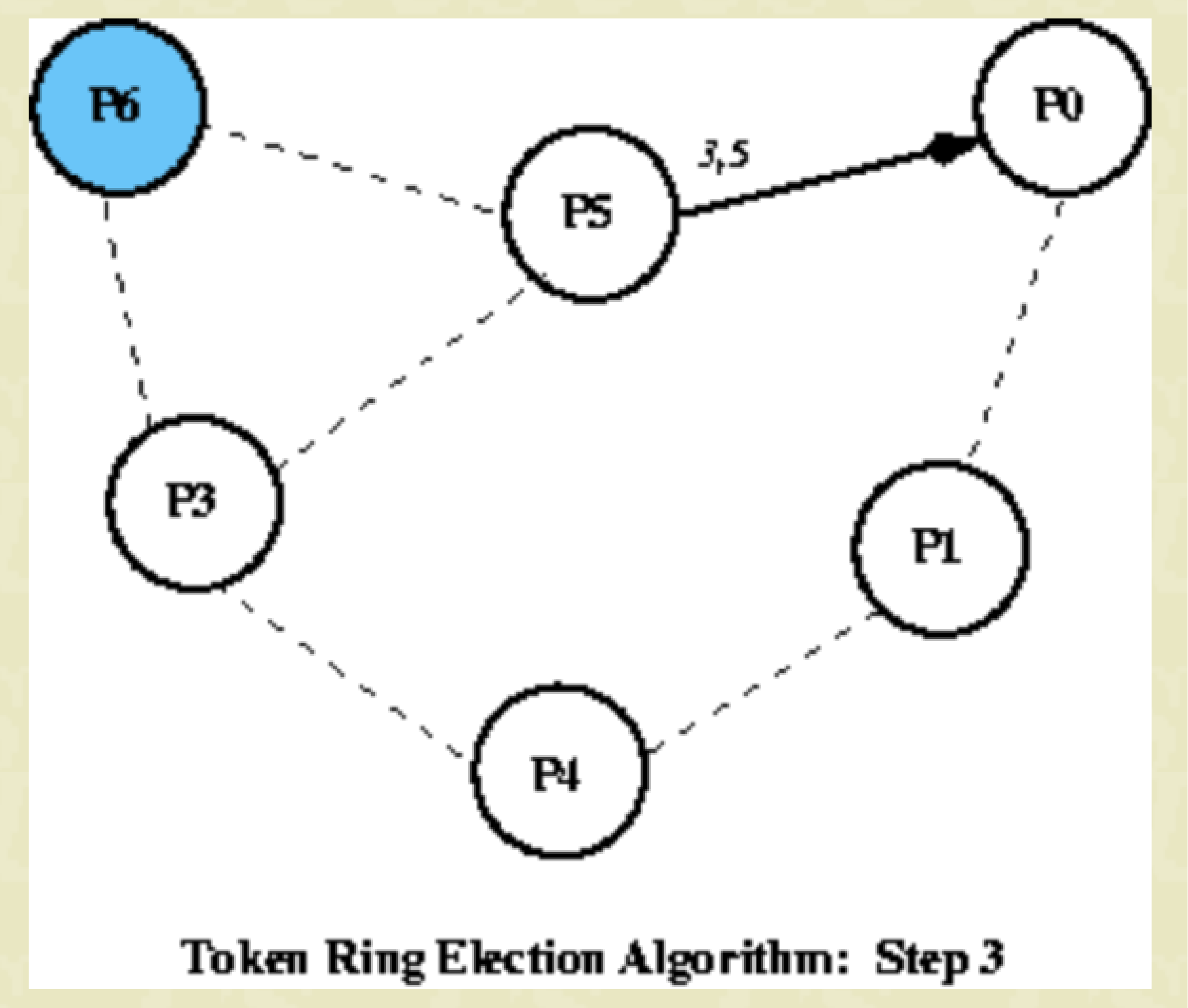
Example



Token Ring Election Algorithm: Step 2

Process 3 notices that Process 6 does not respond.
So it starts an election,
sending a message containing its id
to the next node in the ring.

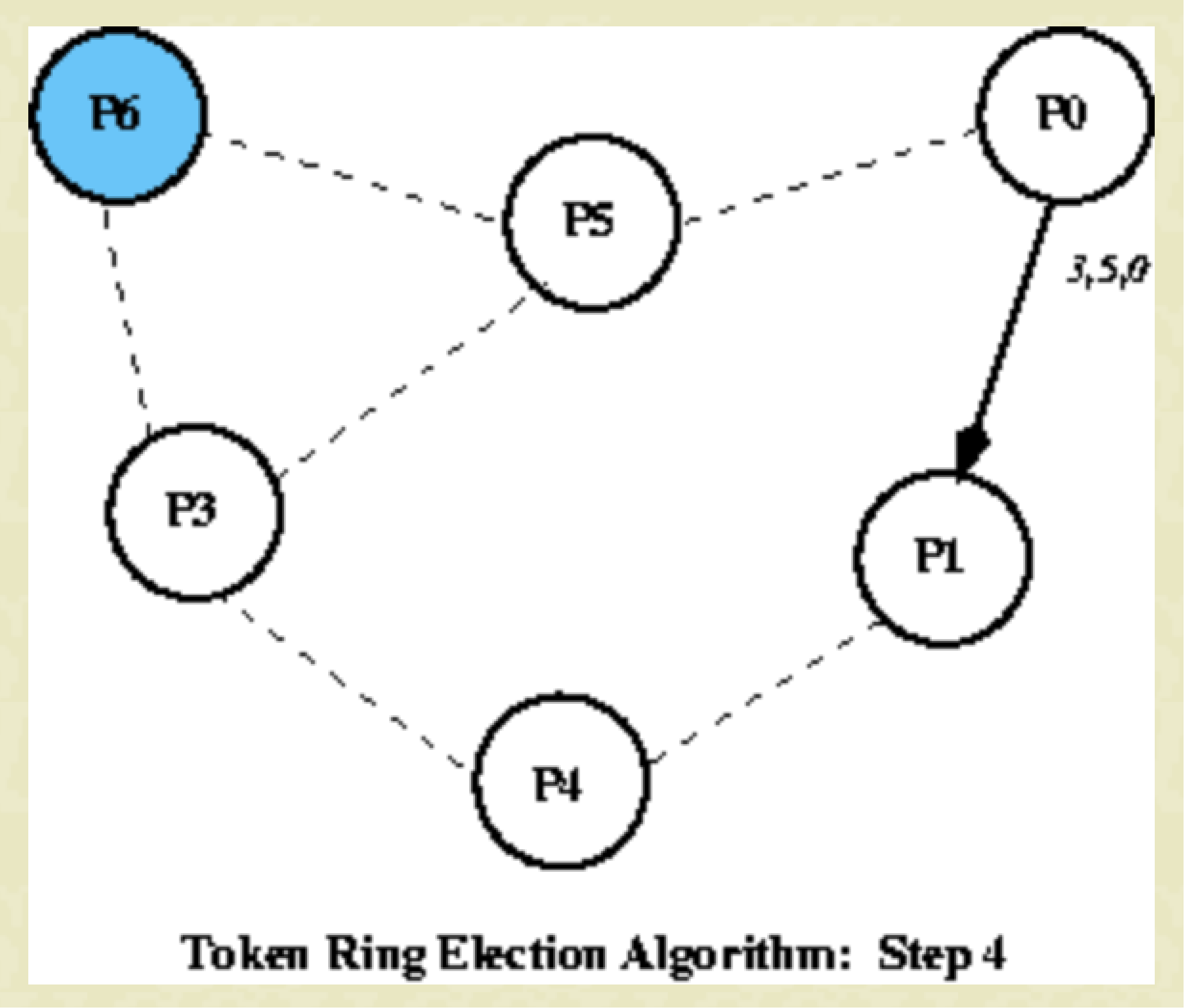
Example



Token Ring Election Algorithm: Step 3

Process 5 passes the message on, adding its own id to the message.

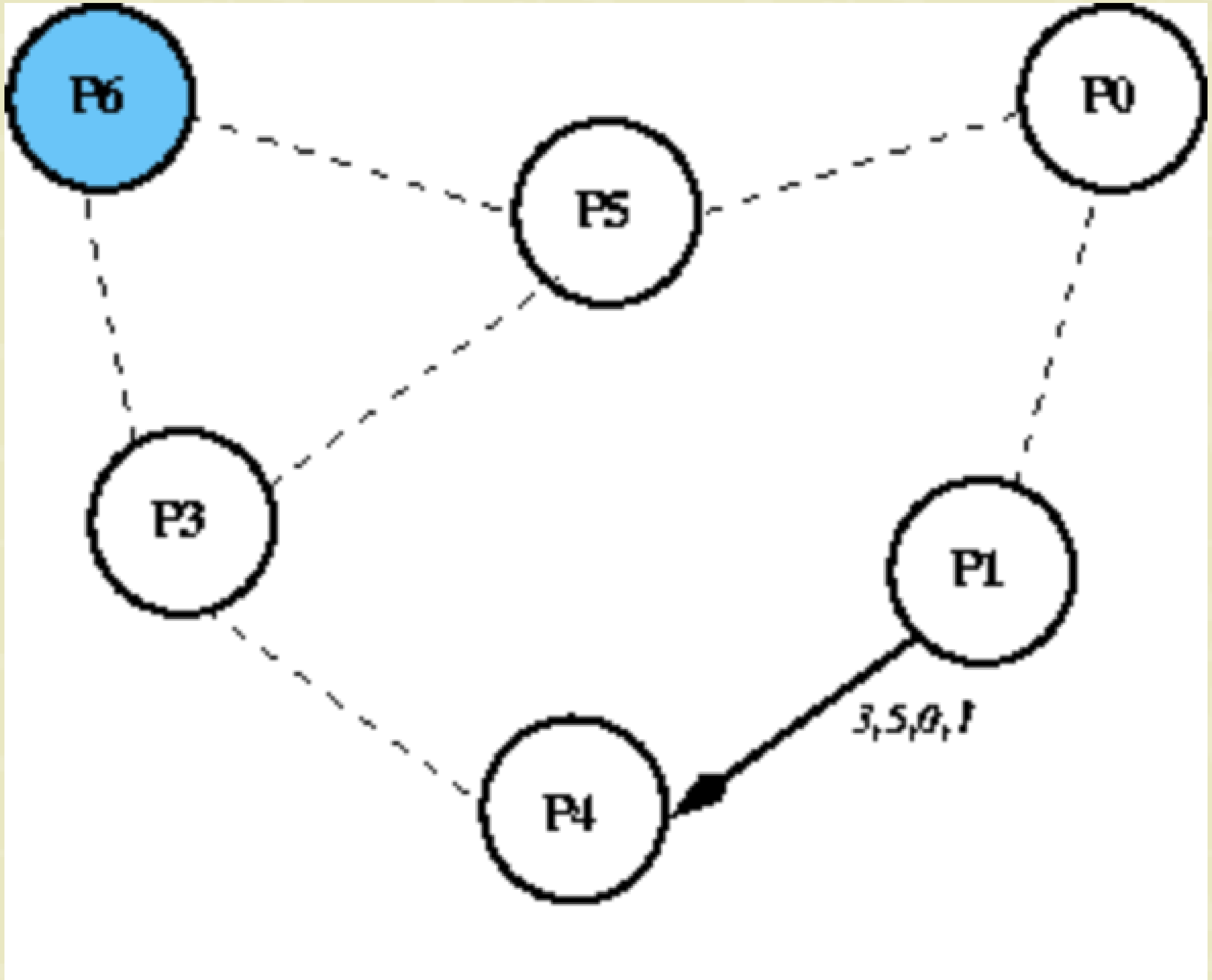
Example



Token Ring Election Algorithm: Step 4

Process 0 passes the message on, adding its own id to the message.

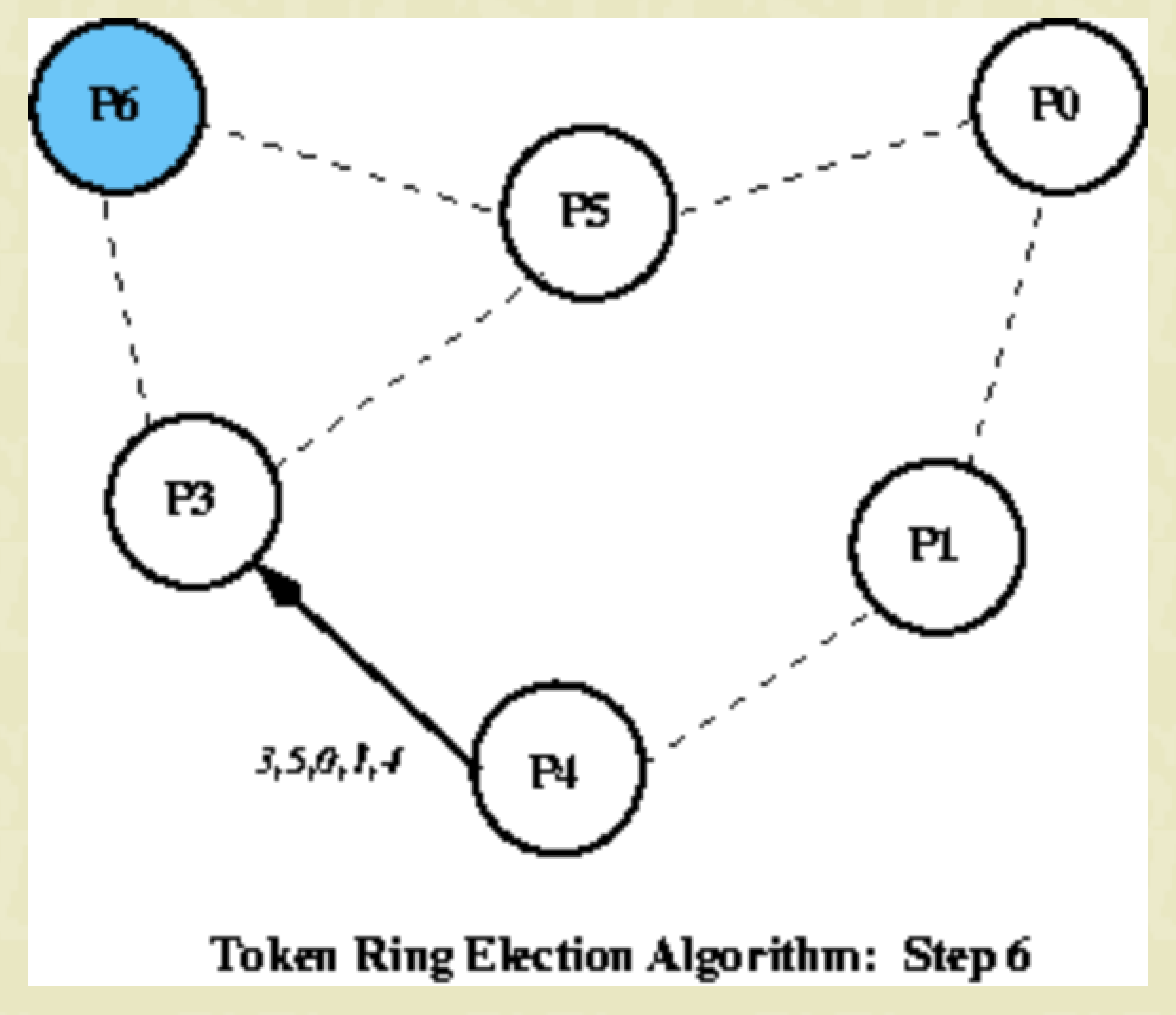
Example



Token Ring Election Algorithm: Step 5

Process 1 passes the message on, adding its own id to the message.

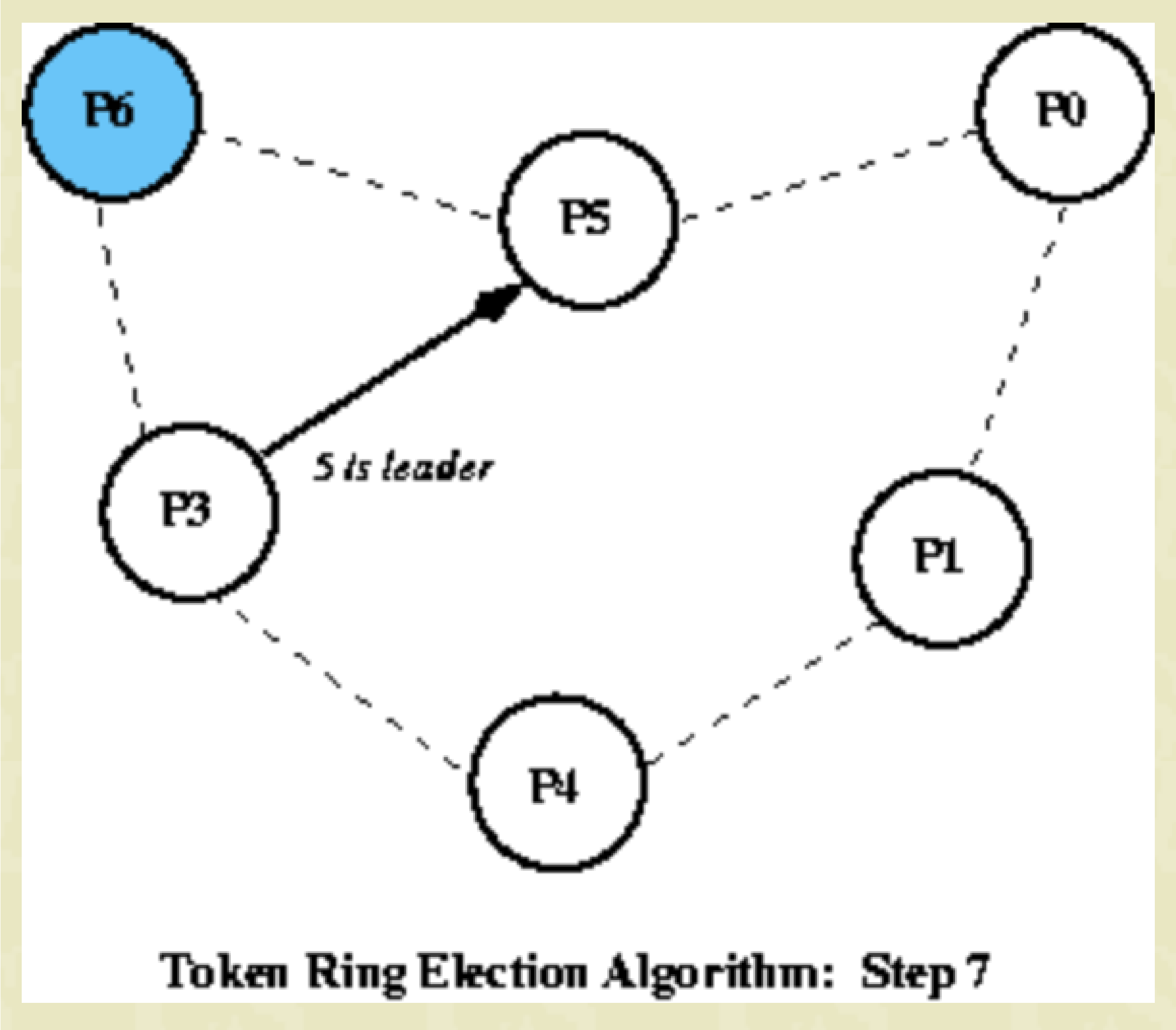
Example



Token Ring Election Algorithm: Step 6

Process 4 passes the message on, adding its own id to the message.

Example

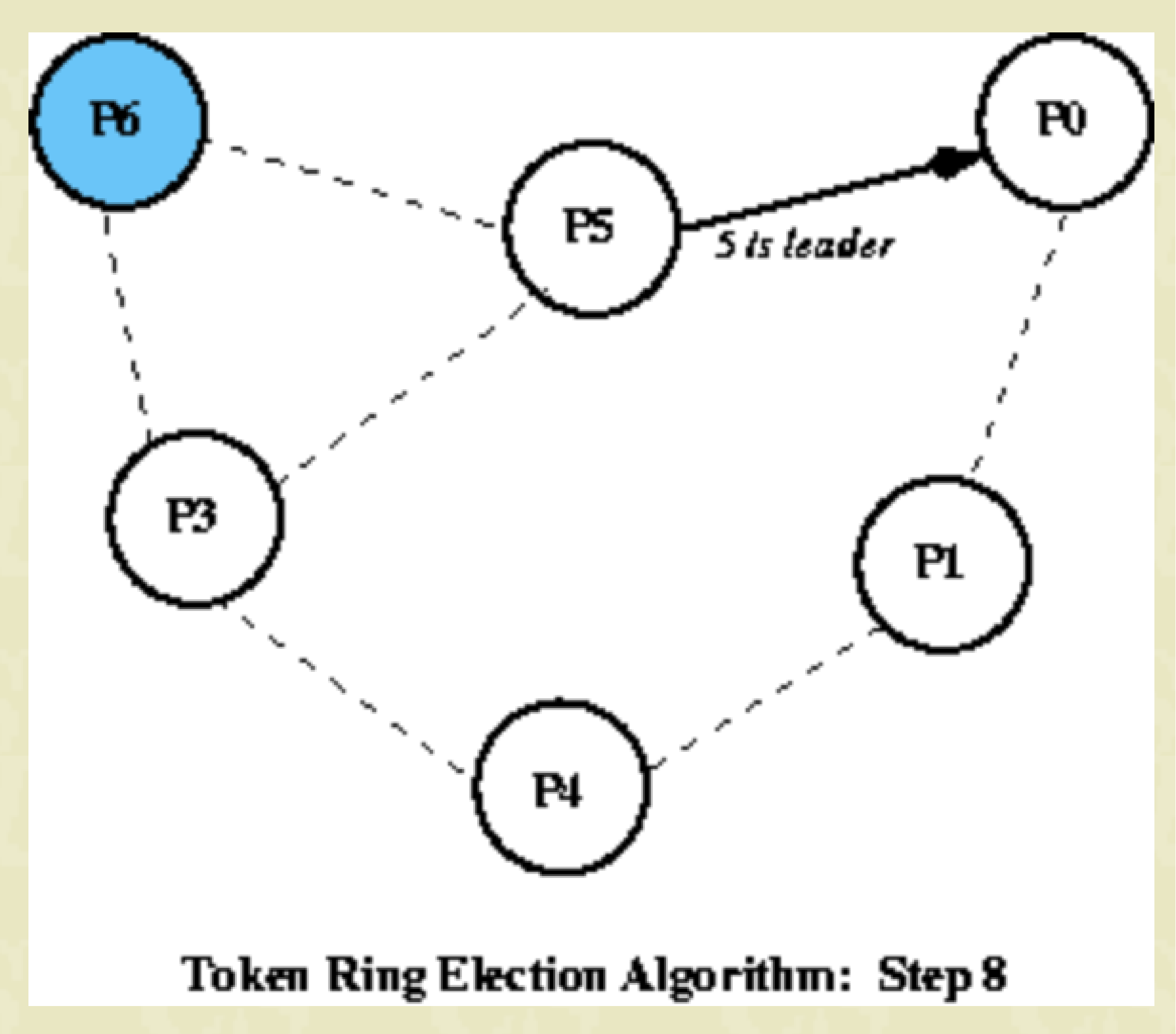


Token Ring Election Algorithm: Step 7

When Process 3 receives the message back, it knows the message has gone around the ring, as its own id is in the list.

Picking the highest id in the list, it starts the coordinator message "5 is the leader" around the ring.

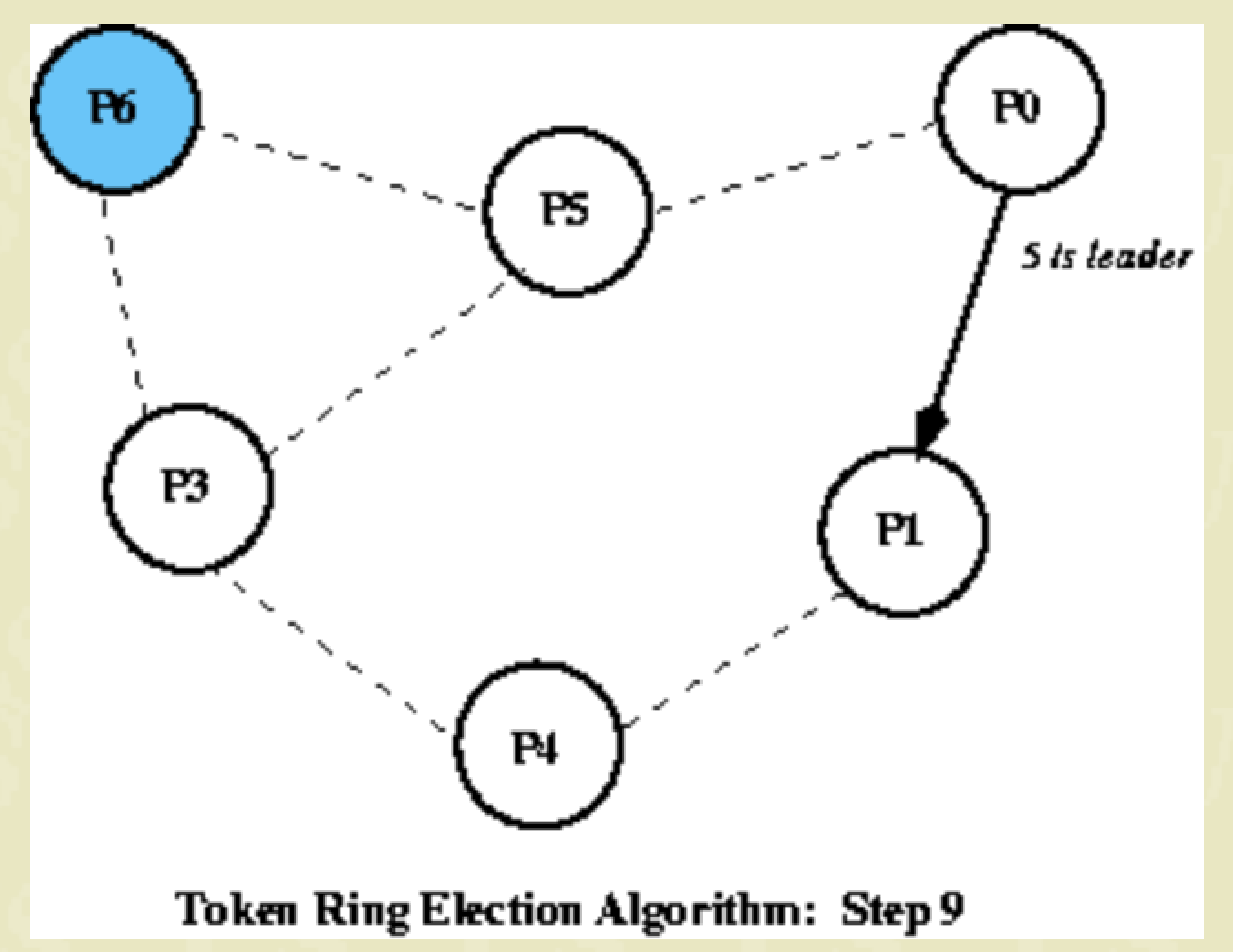
Example



Token Ring Election Algorithm: Step 8

Process 5 passes on the coordinator message.

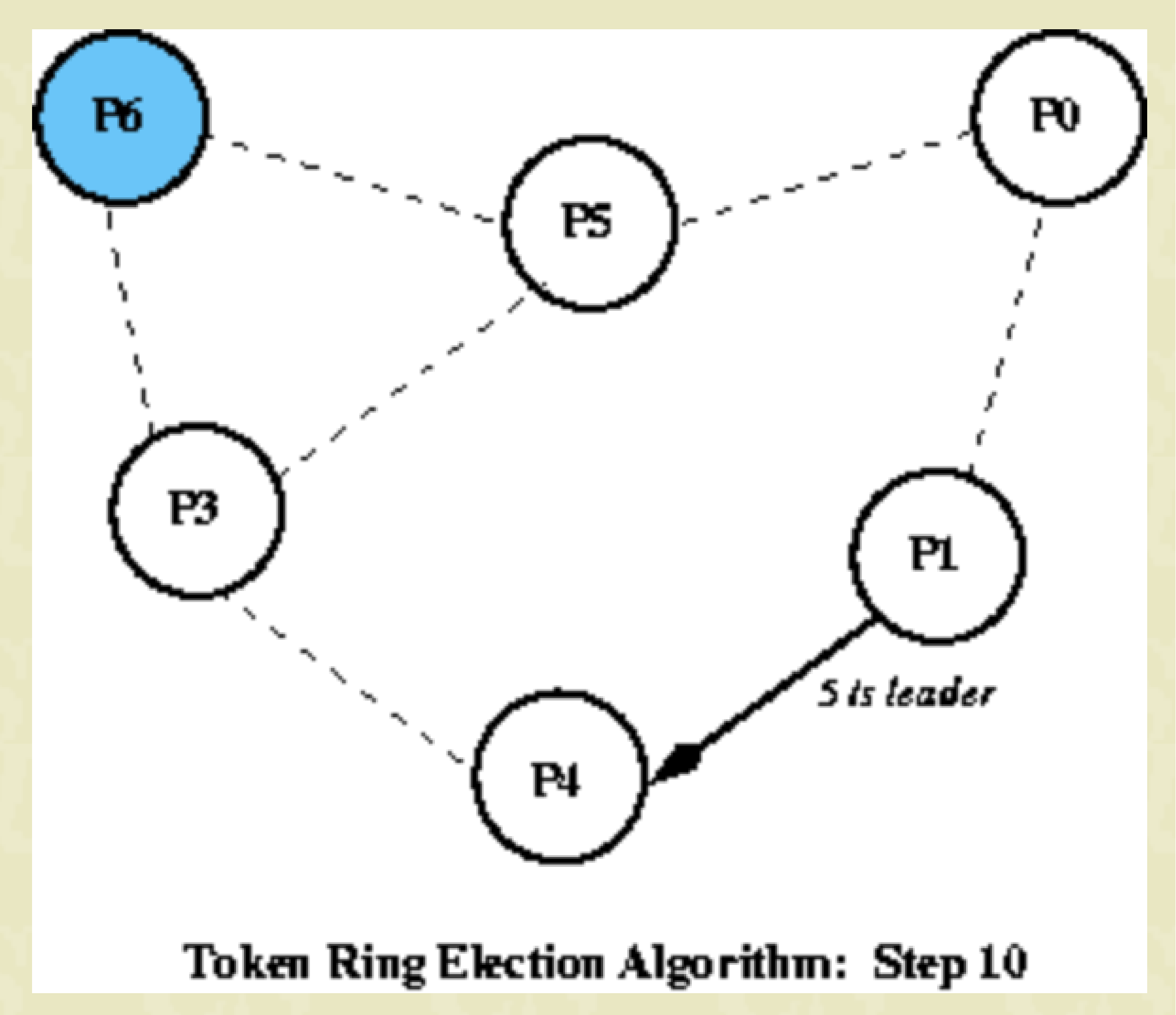
Example



Token Ring Election Algorithm: Step 9

Process 0 passes on the coordinator message.

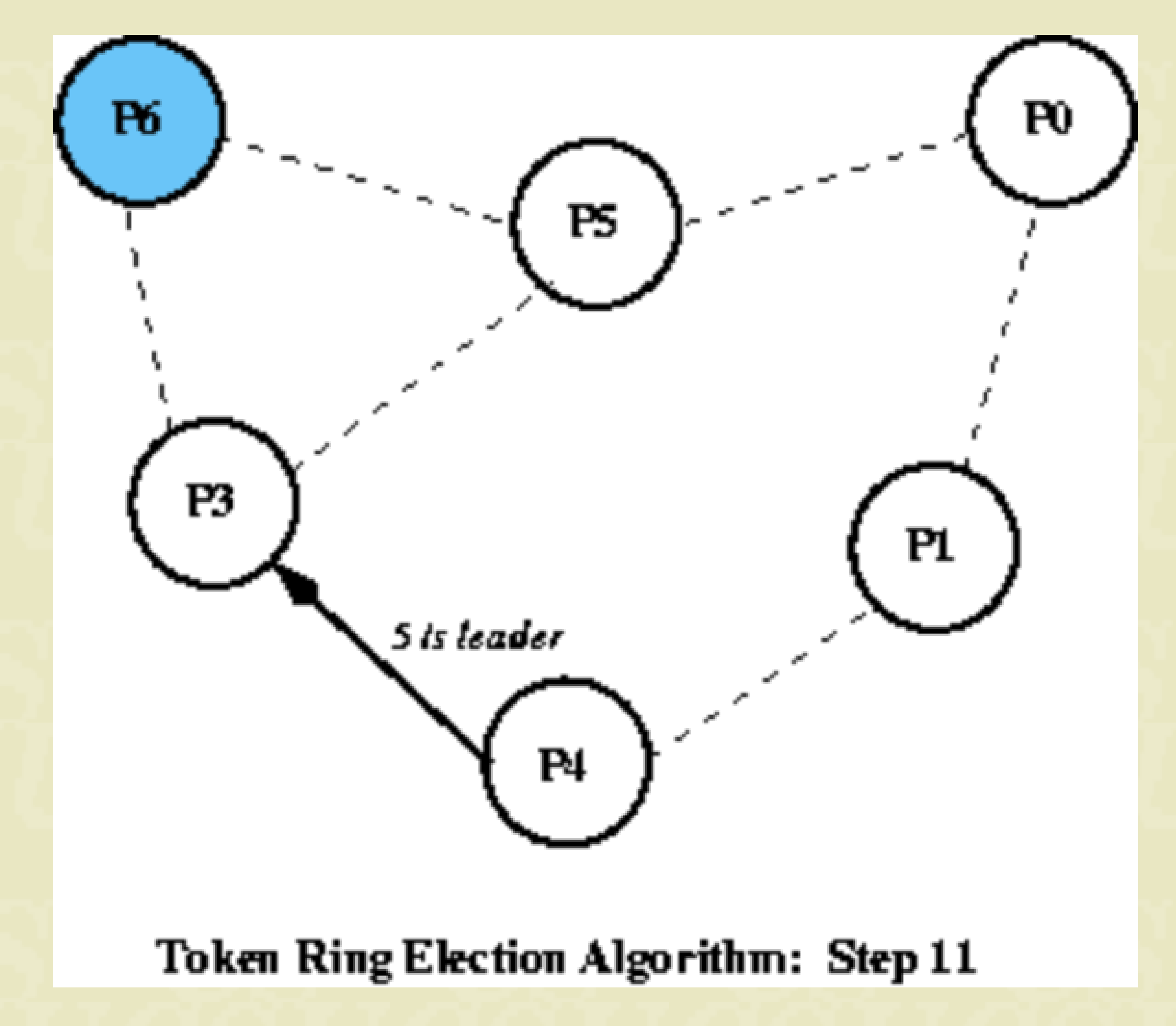
Example



Token Ring Election Algorithm: Step 10

Process 1 passes on the coordinator message.

Example



Token Ring Election Algorithm: Step 11

Process 4 passes on the coordinator message.

Process 3 receives the coordinator message, and stops it.

Analysis: Ring Algorithm

- Bandwidth:
  - Worst case scenario, N-1 messages are required for the ultimate winner to receive the elect message.
  - A further N messages are required for the elect message to come back around so the winner knows it has won.
  - A further N messages are used to inform everybody of the result (the elected message).
  - Therefore the bandwidth is (N –1) + N + N = 3N – 1

- Turnaround Time:
  - This is the number of serialized messages sent during the election process.
  - Each node transmits each message in sequence.
  - Therefore the turnaround time is the also 3N – 1.

Bully Algorithm

- Developed by Garcia-Molina in 1982.

- Assumptions:
  - Each process knows the ID and address of every other process.
  - Communication is reliable.

Bully Algorithm

The algorithm uses the following message types:

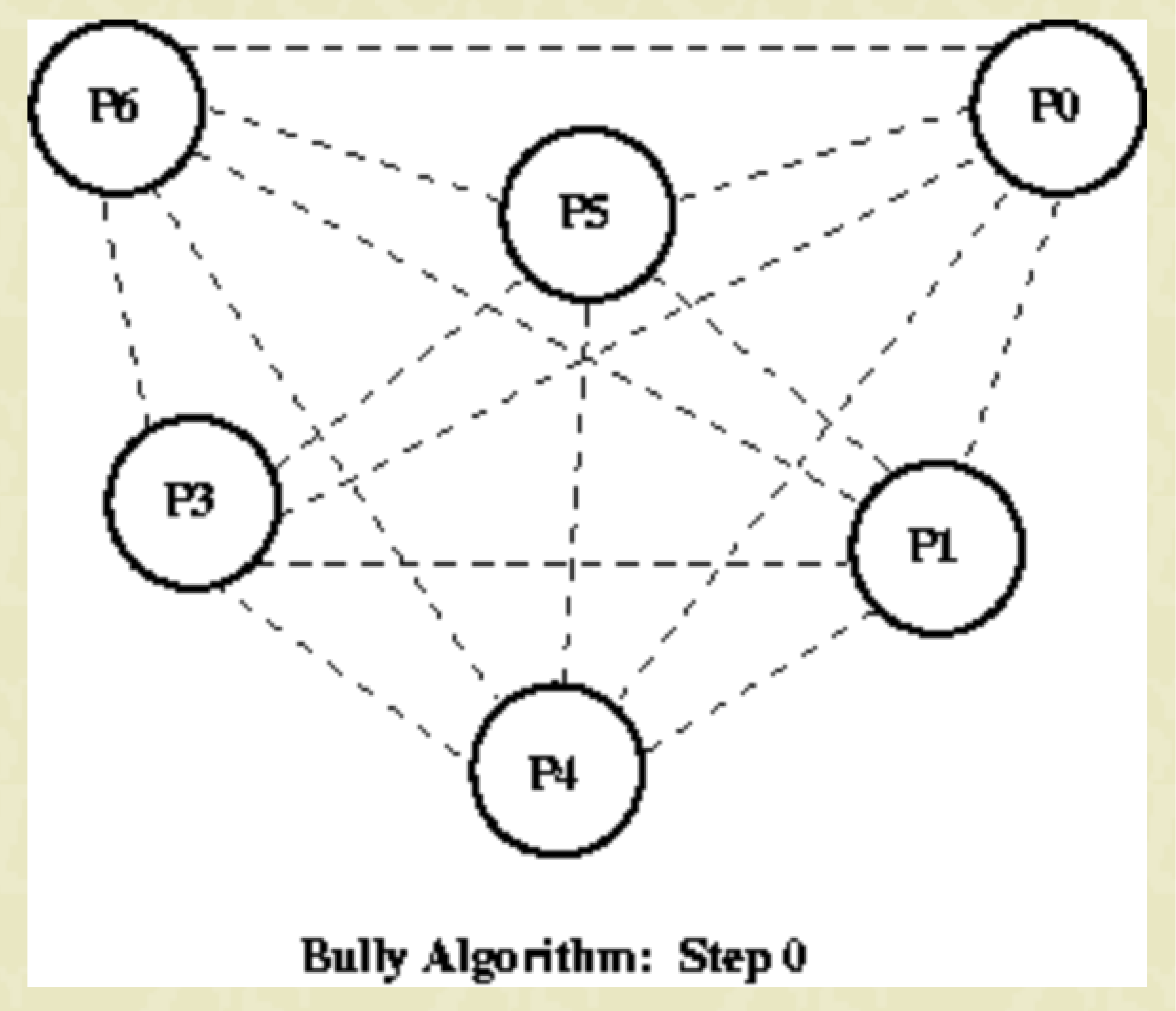
Election Message: Sent to announce election.
Answer (Alive) Message: Responds to the Election message.
Coordinator (Victory) Message: Sent by winner of the election to announce victory.

Bully Algorithm

When a process P recovers from failure, or the failure detector indicates that the current coordinator has failed, P performs the following actions:

1. If P has the highest process id, it sends a Victory message to all other processes and becomes the new Coordinator. Otherwise, P broadcasts an Election message to all other processes with higher process IDs than itself.
2. If P receives no Answer after sending an Election message, then it broadcasts a Victory message to all other processes and becomes the Coordinator.
3. If P receives an Answer from a process with a higher ID, it sends no further messages for this election and waits for a Victory message.
4. If P receives an Election message from another process with a lower ID it sends an Answer message back and starts the election process at the beginning, by sending an Election message to higher-numbered processes.
5. If P receives a Coordinator message, it treats the sender as the coordinator.

Bully Algorithm

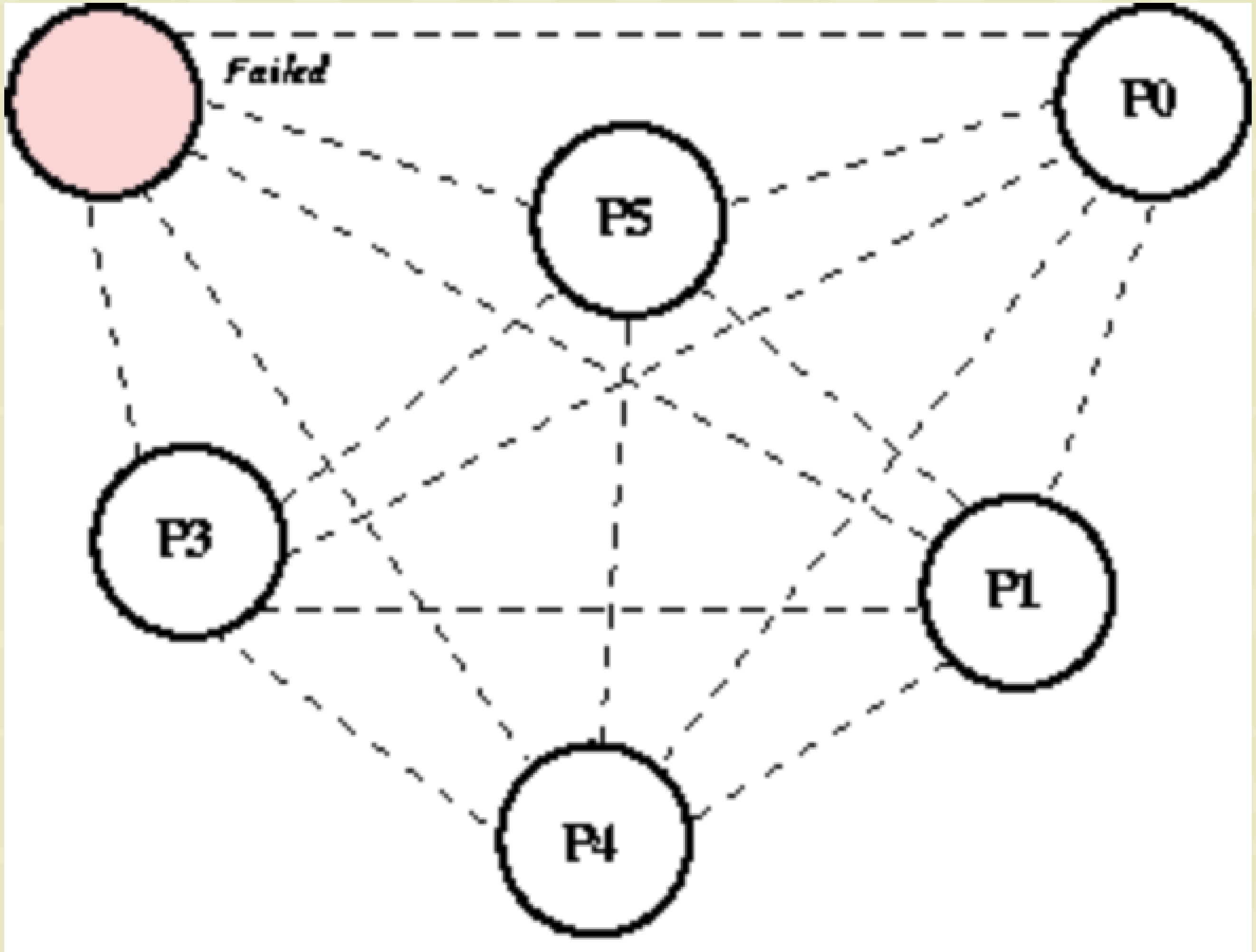


Bully Algorithm: Step 0

We start with 6 processes,
all directly connected to each other.

Process 6 is the leader,
as it has the highest number.

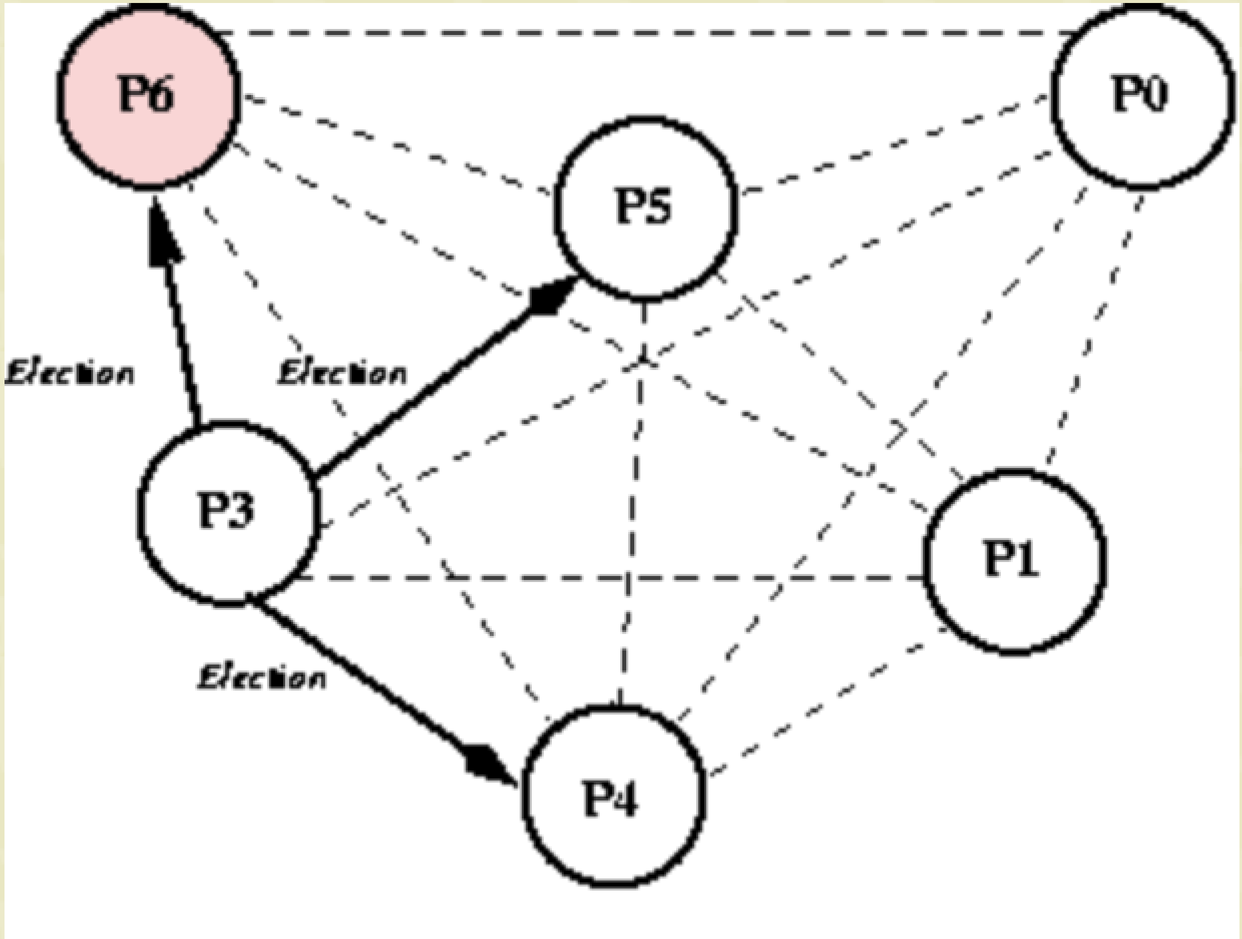
Bully Algorithm



Bully Algorithm: Step 1

Process 6 fails.

Bully Algorithm

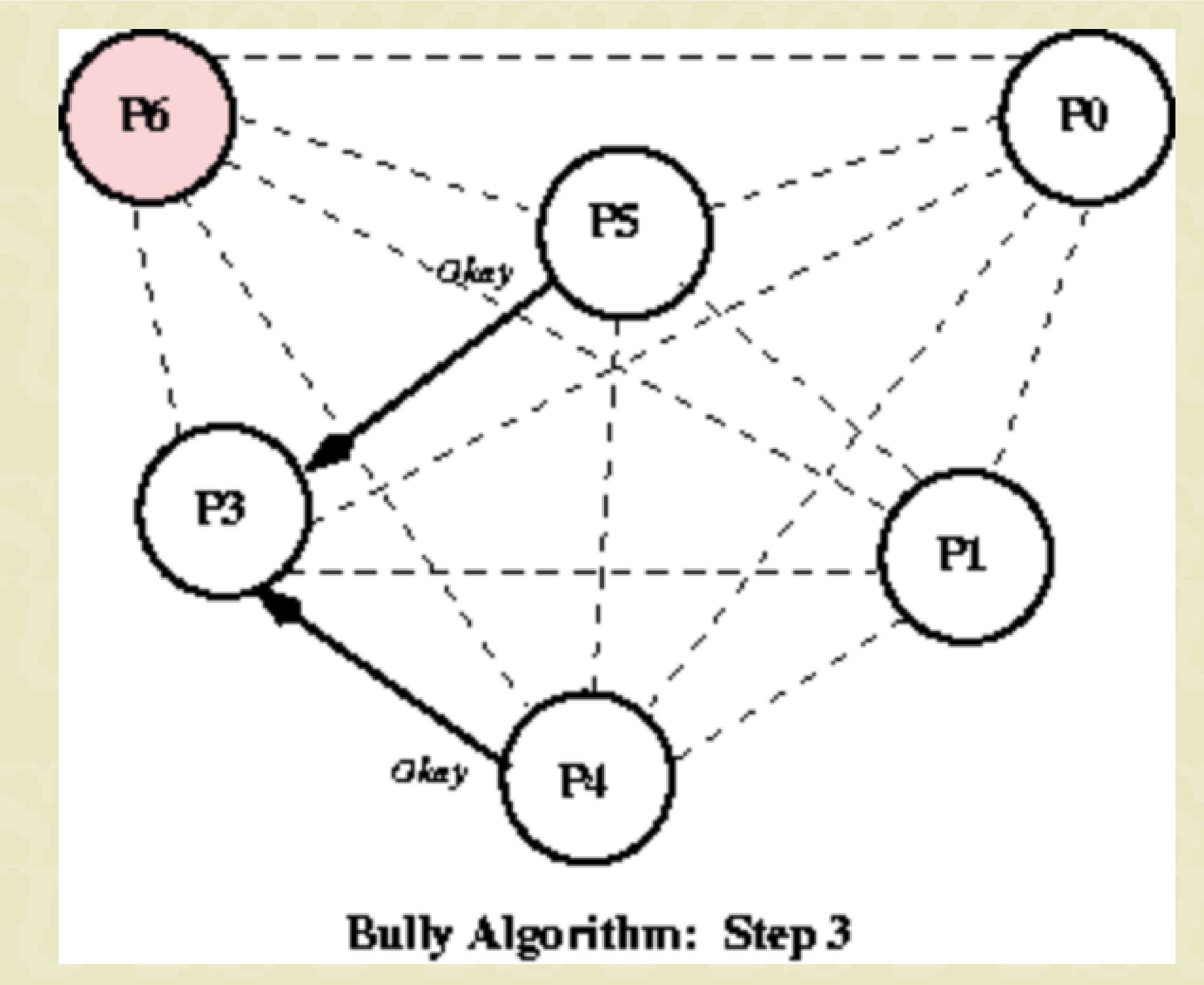


Bully Algorithm: Step 2

Process 3 notices that Process 6 does not respond

So it starts an election, notifying those processes with ids greater than 3.

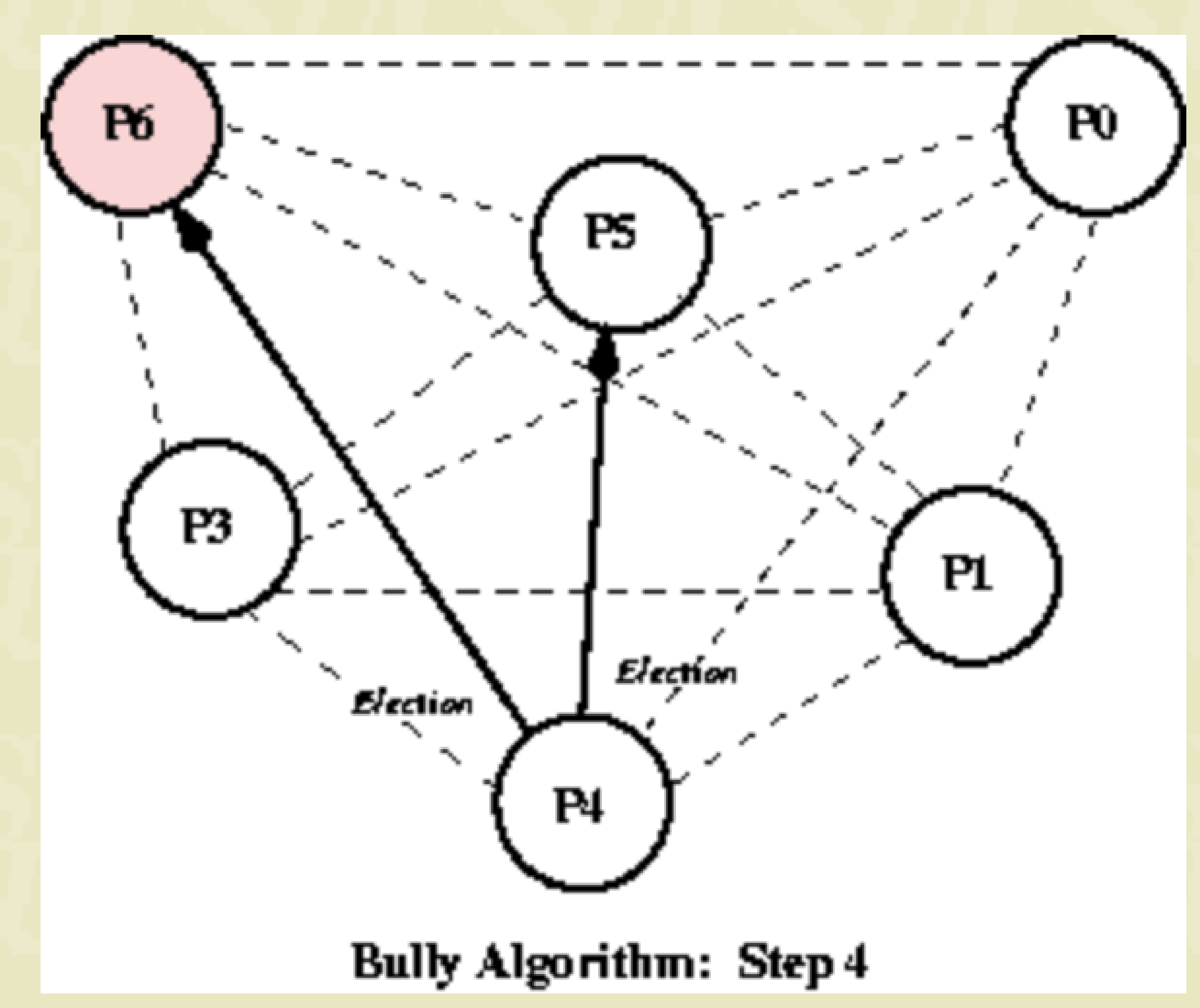
Bully Algorithm



Bully Algorithm: Step 3

Both Process 4 and Process 5 respond, telling Process 3 that they'll take over from here.

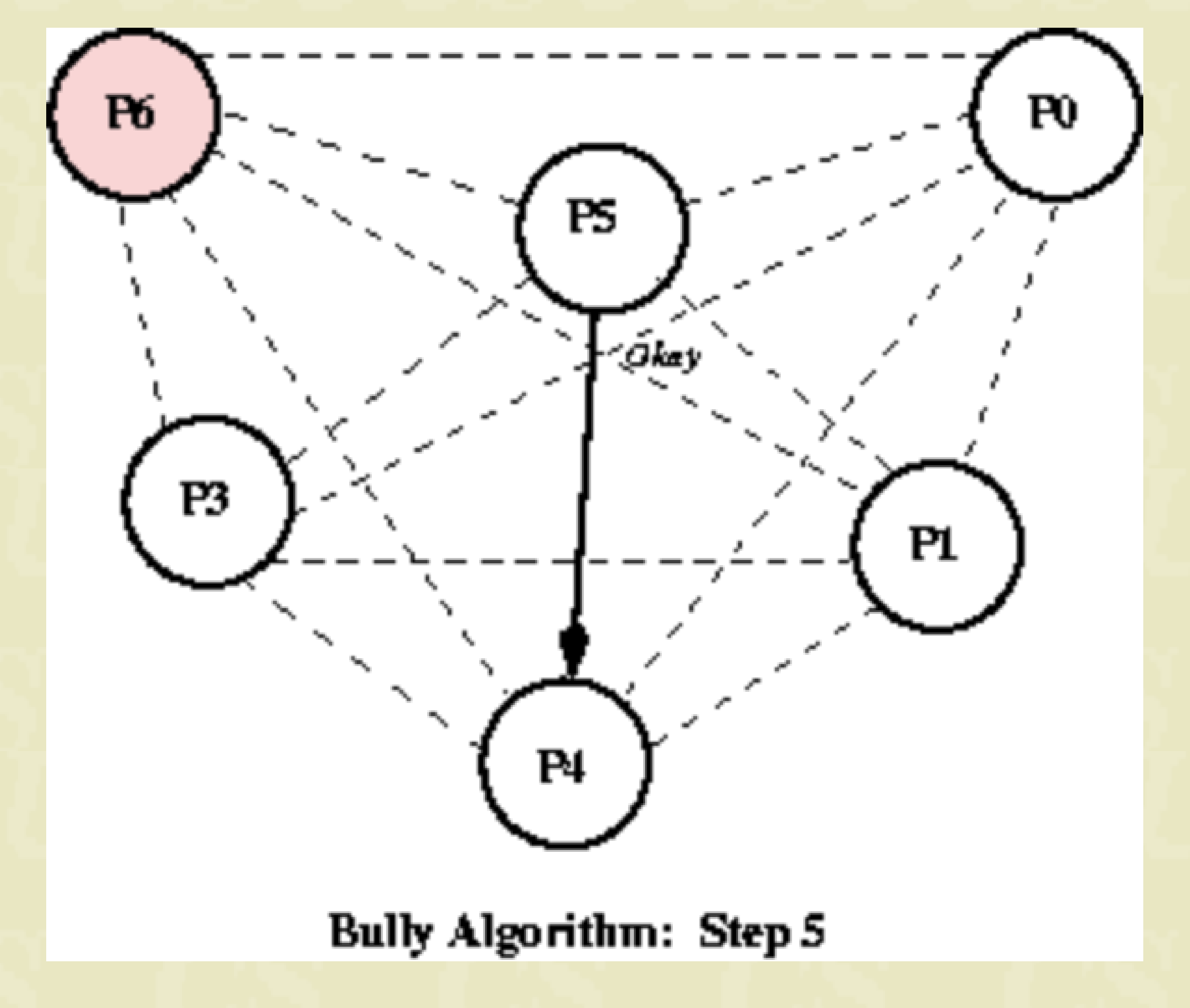
Bully Algorithm



Bully Algorithm: Step 4

Process 4 sends election messages to both Process 5 and Process 6.

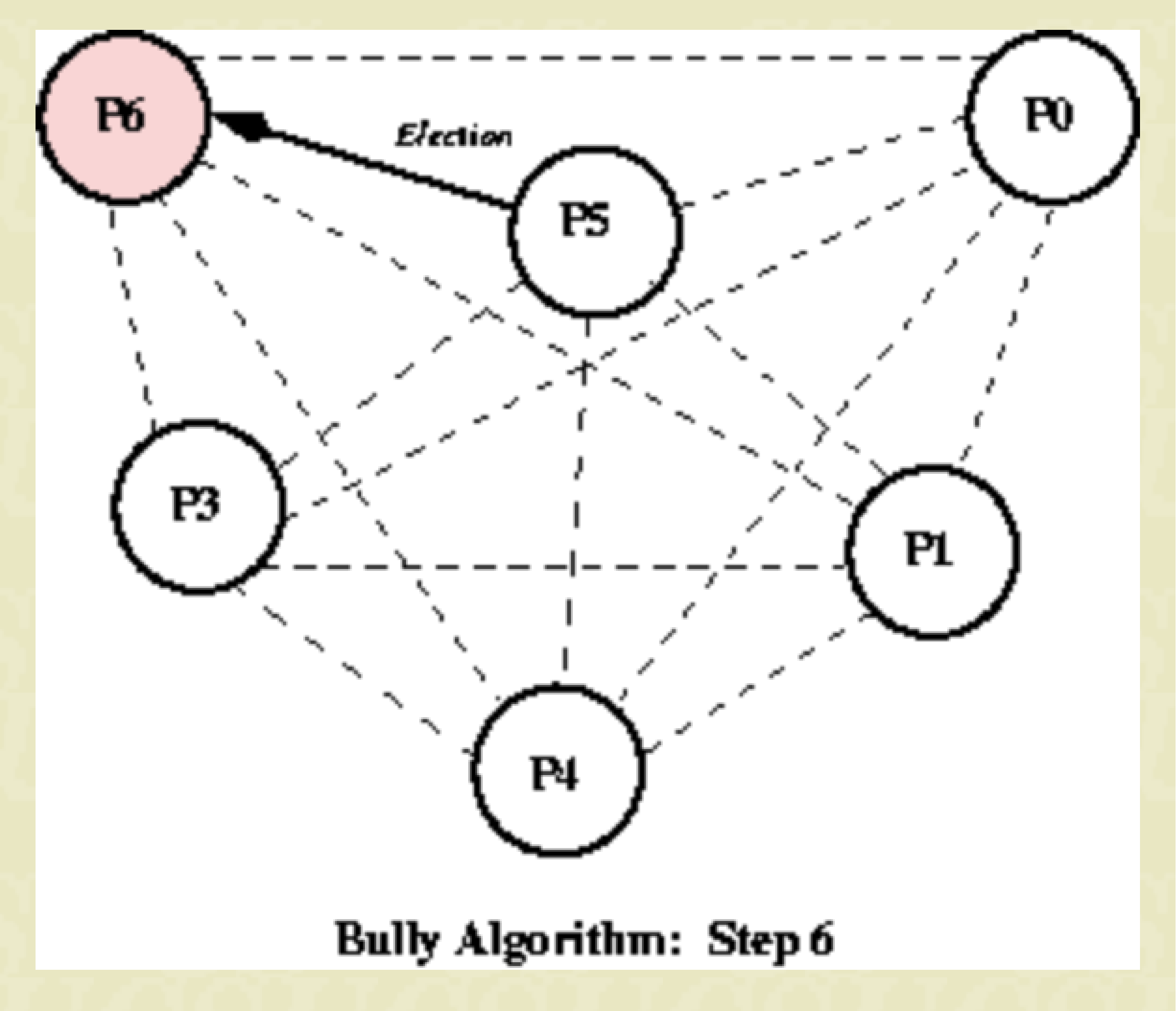
Bully Algorithm



Bully Algorithm: Step 5

Only Process 5 answers and takes over the election.

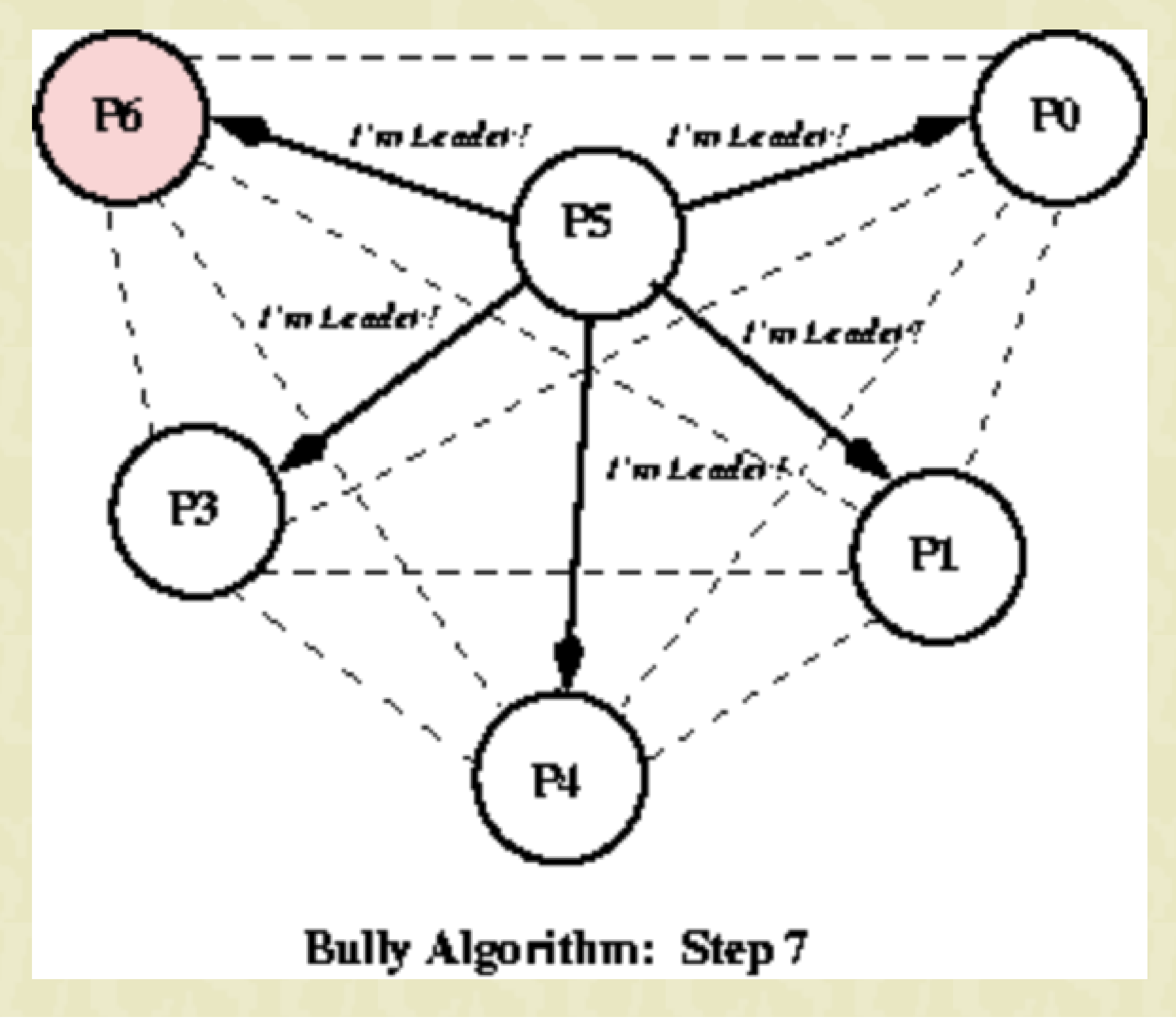
Bully Algorithm



Bully Algorithm: Step 6

Process 5 sends out only one election message to Process 6.

Bully Algorithm



Bully Algorithm: Step 7

When Process 6 does not respond

Process 5 declares itself the winner.

Analysis: Bully Algorithm

- *Bandwidth:*
  - In the worst case, the process with the lowest ID detects the failure
    - It sends N-1 election messages, and receives N-1 OK responses
  - This in turn causes causes the other N-1 processes to start elections
    - Which result in 2*(N-2), 2*(N-3), , 2 messages respectively
  - Finally, the winning process sends N-1 COORDINATOR messages.

$$\begin{aligned} \text{Bandwidth} \quad &= 2 \text{ x } \left(\textstyle\sum_{j=1, \quad , N-1} j\right) + (n-1) \\ &= (N-1)^2 + (N-1) + (N-1) \\ &= N^2 - 2N + 1 + 2N - 2 \\ &= N^2 - 1 \end{aligned}$$

$$\sum_{1 \le i \le n} i = (n^2+n)/2$$

- *Turnaround Time:*
  - Worst case N-1 elections held giving a turnaround time of N-1 (messages are not actually sent sequentially)

Analysis: Bully Algorithm

- Bandwidth:
  - In the best case, the process with the second-highest identifier notices the failure of the existing co-ordinator, so it can immediately elect itself and send N-2 co-ordinator messages.
- Turnaround Time:
  - Turnaround time is just 1 message, as these are sent concurrently.

Election Algorithms

- The examples used the process identifier as a score for the election process.

- In practice, we can use any score that we want to, for example:
  - Load-based scores: score for process p, S_p = 1/$load_p$ (will elect the process that has the minimum load).
  - Bandwidth-based scores: the score for each process is the maximum bandwidth available to that process.

Distributed Systems:

Distributed File Systems

Introduction

- Sharing of resources is a key goal for distributed systems.
  - Perhaps the most common type of shared resource is stored information.
- This can be done in many ways:
  - Databases
  - Distributed Shared Memory
  - Remote Objects
  - …
- Underlying many of these approaches is the ability to store data persistently.
- To do this, we need a File System!

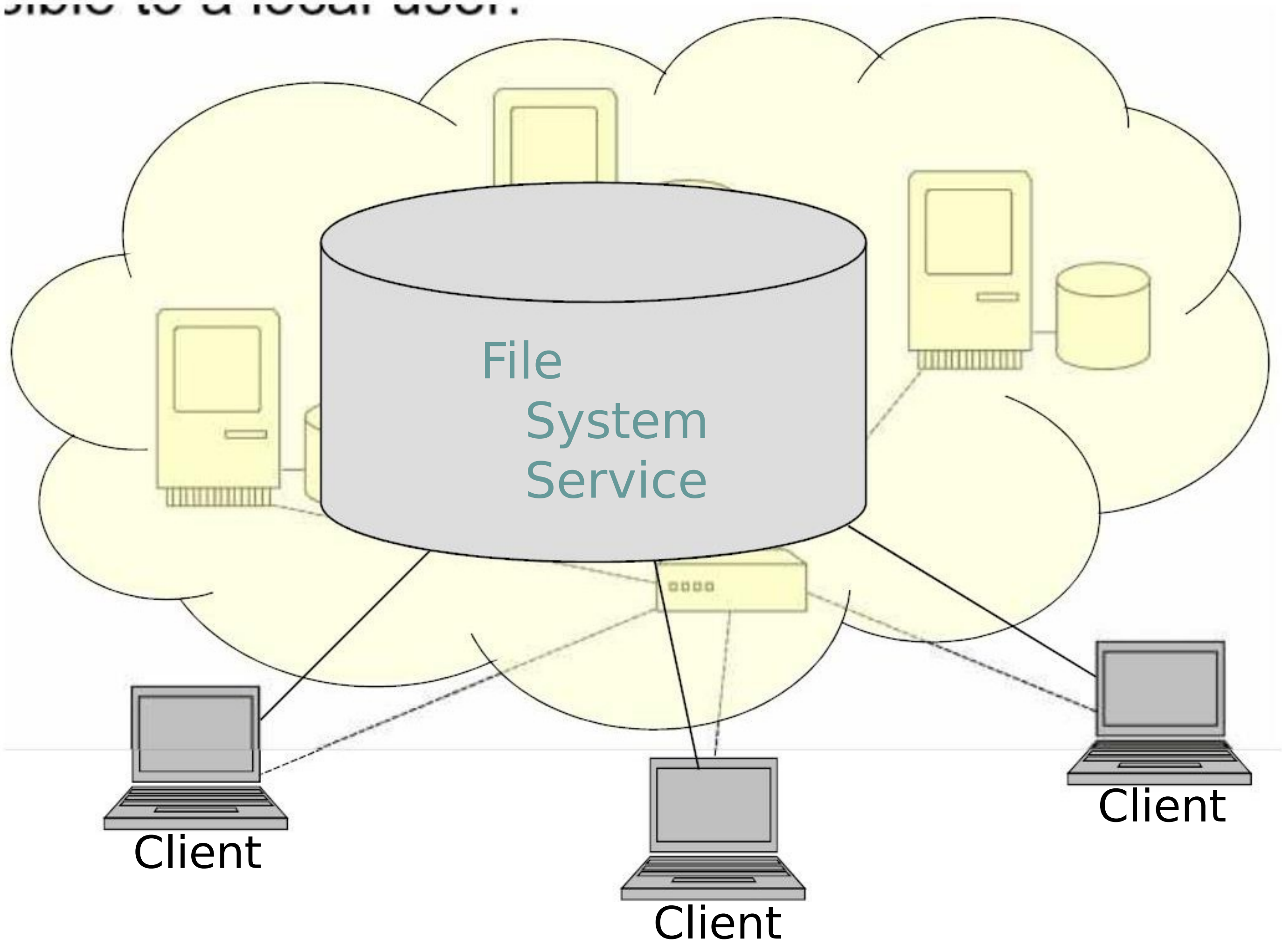
File Systems

- In general, file systems are responsible for:
  - the organisation, storage, retrieval, naming, sharing, and protection of files.

Directory module:	Relates file names to file Ids
File module:	Relates file Ids to particular files
Access control module:	Checks permission for operation requested
File access module:	Reads or writes file data or attributes
Block module:	Accesses and allocates disk blocks
Device module:	Disk I/O and buffering

Distributed File Systems (DFS)

- A Distributed File System aims to make remote files accessible to a local user.



DFS: Components

- File Servers:
  - These are a combination of hardware and software.
  - The hardware includes secondary storage resources (e.g. magnetic disks).
  - The software handles file service requests.

- Clients:
  - This is used to transmit file service requests to the file servers.
  - It also includes some form of user interface for the service.

- File Services:
  - These specify the primitive file operations of the DFS.
  - For example: create a file, delete a file, read from a file, …

Examples of DFS

- Novell Netware:
  - It initially used cooperative multitasking to run various services on a personal computer, using the IPX network protocol.
  - Client integrates into Windows OS
    - Remote directories are made accessible as separate drives (e.g. H:, F:, …)
    - Modifies “Start Menu” to allow the execution of remote applications.

- Windows File Sharing:
  - Built-in feature (service) of Windows OS
    - When running, users specify the folders that are to be shared.
    - File access is through a common interface (shared folders can also be mounted as separate drives)
  - Every machine can be both a client and a server!

Outline

- Introduction

- DFS Issues
  - Naming and Transparency
  - Remote File Access
  - Stateful versus Stateless
  - File Replication
  - Security

- Example Systems

Naming

- Naming refers to:
  - the labels that we associate with files and their mapping to physical objects on some storage medium.
  - the structures that we use to organize those files into meaningful groups.

- While naming schemes differ from OS to OS, broadly speaking they follow the same approach:
  - File names are textual labels that (usually) have some relevance to the corresponding file (identified by a file ID)
  - A directory-based hierarchical structure is commonly used to organize the files into meaningful groups (often implemented as special files)
    - E.g. /etc/passwd
  - This structure acts as a context for each file and is needed together with the name in order to identify the correct physical objects.

Transparency of Naming Schemes

- Naming schemes enforce a separation of concerns between the logical objects that we work with and the corresponding physical objects.

- Location transparency – file name does not reveal the file's physical storage location.
  - File name still denotes a specific, although hidden, set of physical disk blocks.
  - Convenient way to share data.
  - Can expose correspondence between component units and machines.

- Location independence – file name does not need to be changed when the file's physical storage location changes.
  - Better file abstraction.
  - Promotes sharing the storage space itself.
  - Separates the naming hierarchy from the storage-devices hierarchy.

Transparency of Naming Schemes

- The problem is that many of our current File Systems are still bound to the stand-alone computer model.

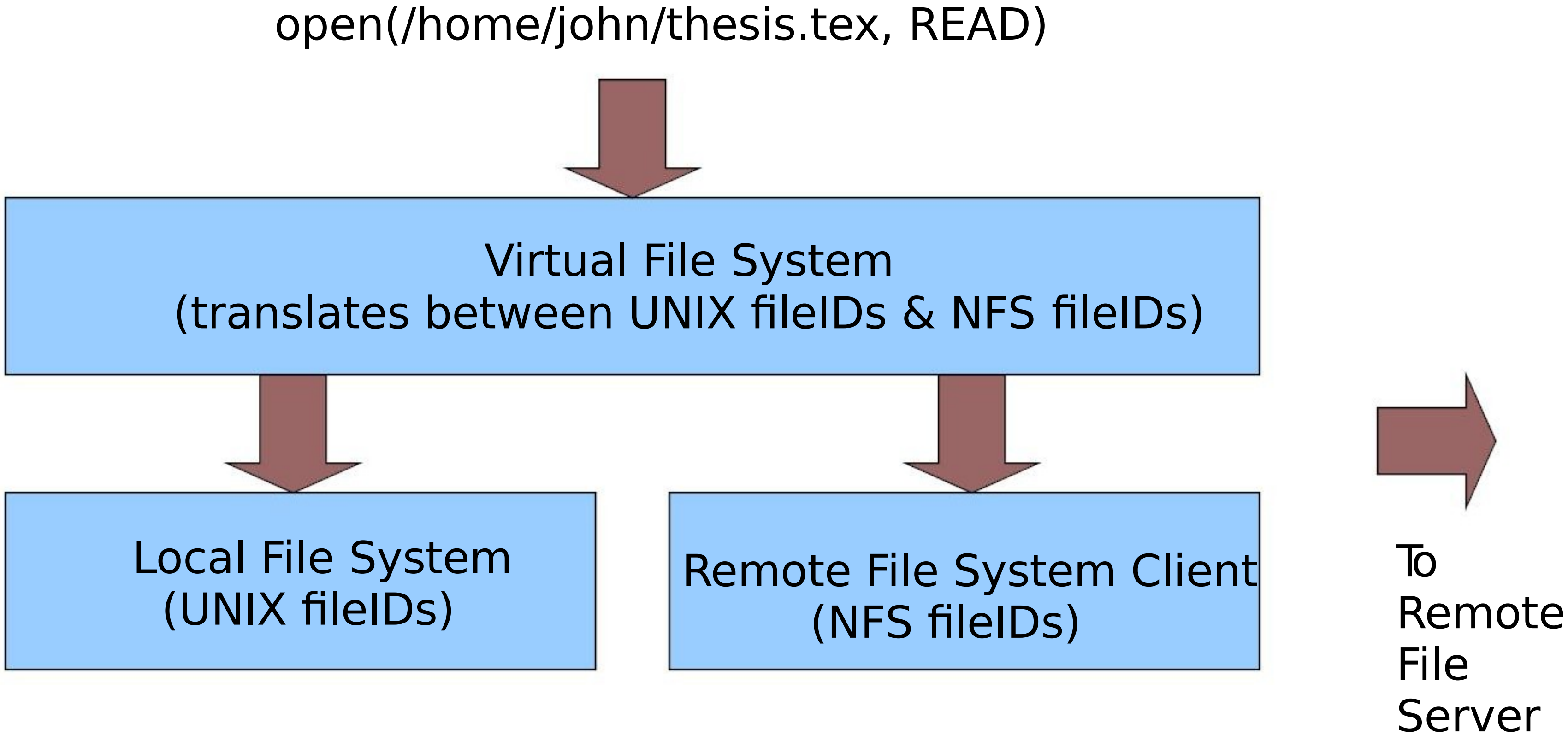
- When we want to access a remote resource, we must explicitly connect to that resource before we can access the files transparently.

- In Sun's NFS, this is known as mounting the remote directory.
  - Remote directories are generally mounted to the /mnt/nfs directory

- Once mounted, you can access the remote files in exactly the same way as the local files
  - This is known as access transparency!

Access Transparency

- Access transparency requires that a common set of operations be used for both local and remote file management.
- Virtual File System (UNIX)



open(/home/john/thesis.tex, READ)

Virtual File System
(translates between UNIX fileIDs & NFS fileIDs)

Local File System
(UNIX fileIDs)

Remote File System Client
(NFS fileIDs)

To
Remote
File
Server

Global Naming

- Another issue in DFS Naming Schemes is how to provide a global context for every file name.

- In UNIX we can mount a remote directories anywhere!
  - How I organize my remote directories may differ from how you organize them…

- When we talk about a file, how do we know which file we are talking about?
  - The file /home/john/thesis.tex on my machine could be the same as the file /mnt/nfs/johns_desktop/thesis.tex on yours.

Global Naming

- Solution: Total integration of the component file systems.

- Employ a single global name structure that spans all the files in the system.
  - Examples include X.500 Naming Scheme and the Andrew File System (AFS)

- Each machine has the same view of the DFS.
  - Ideally, the composed file system structure is isomorphic to the structure of a conventional file system.
  - Unfortunately, certain files (device files and machine specific files) make this difficult

- If a server is unavailable, some arbitrary set of directories becomes unavailable on some different machines.

Outline

- Introduction
- DFS Issues
  - Naming and Transparency
  - Remote File Access
  - Stateful versus Stateless
  - File Replication
  - Security
- Example Systems

Remote File

- Distributed File Systems require mechanisms for accessing remote files.
  - Here we must understand how we will read and write data to a remote file.

- There are two basic approaches:
  - Upload/Download Model.
    - Downloads the whole file to the client, modifies it and then uploads it back to the server.
  - Remote Access Model
    - Download only the bits that you need, and send back the bits that you change.

Upload/Download Model

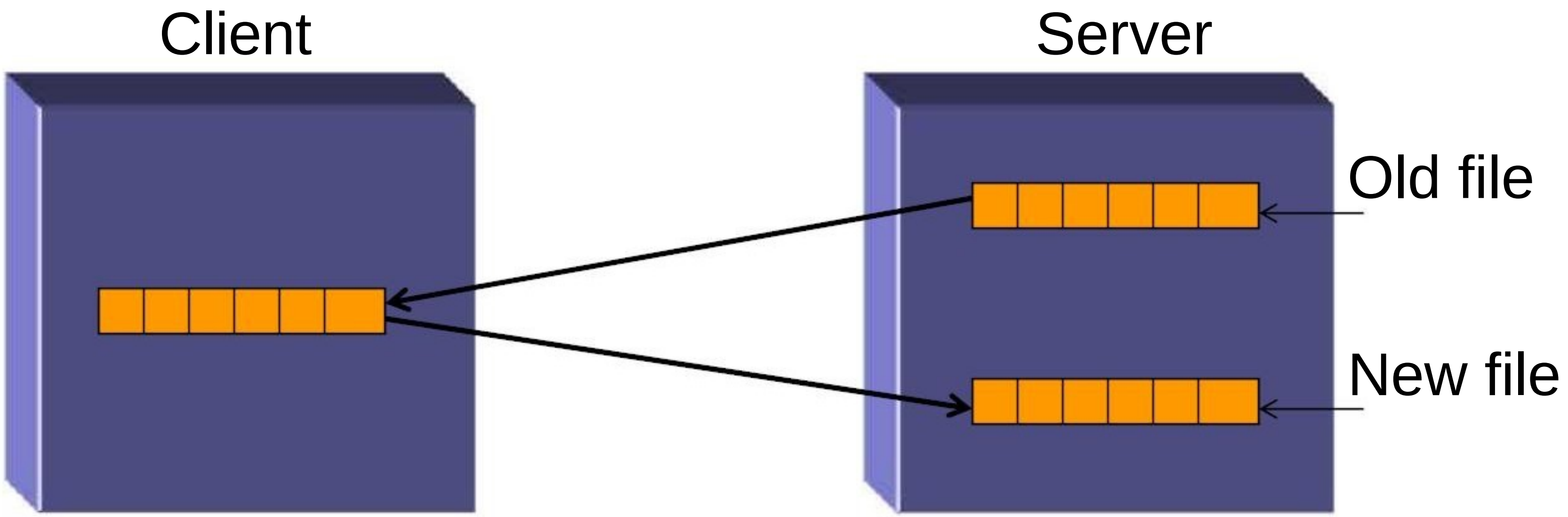
- IDEA: Move the file to the file system

- Provides only two major operations: read and write
  - The read operation transfers the entire file from a file server to the requesting client
  - The write operation transfers the entire file on the way back

- Advantages:
  - Simple file service interface

- Disadvantages
  - Enough storage must be available on the client
  - Moving the whole file is sometimes wasteful!
    - Particularly for large files.

Upload/Download Model



Remote Access

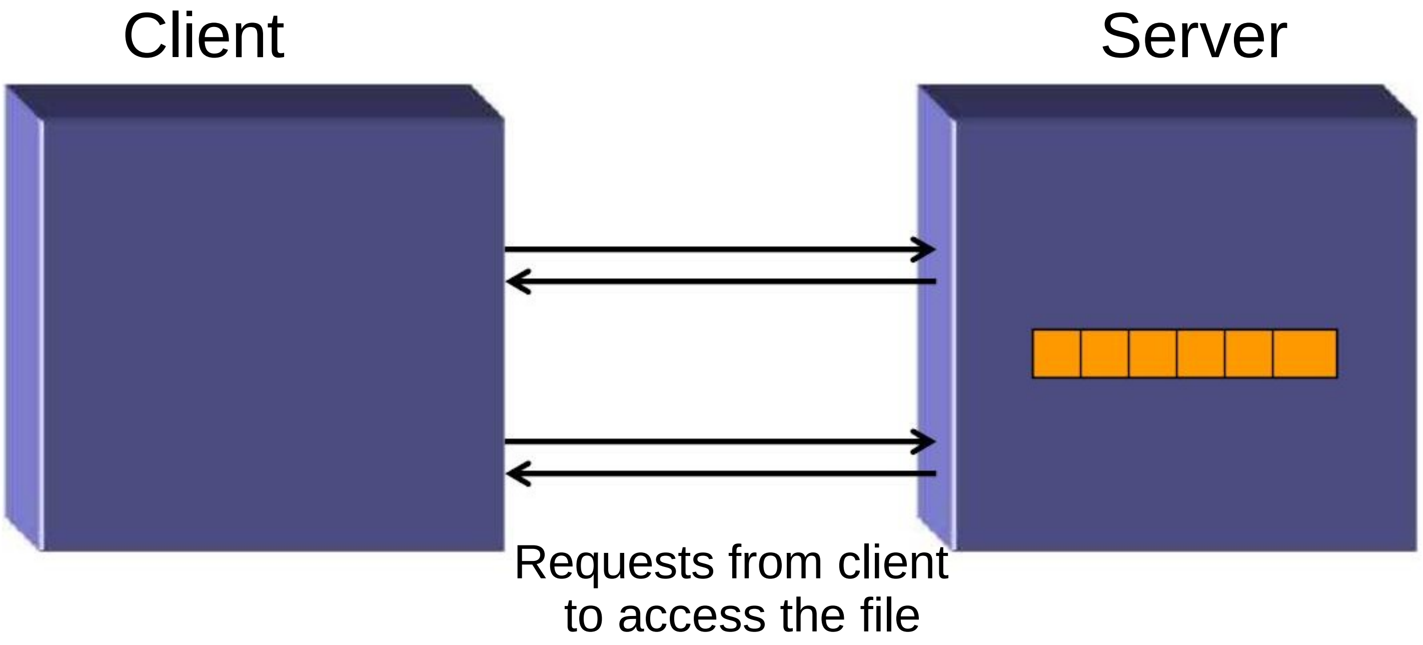
- IDEA: Remotely Control the File System

- The file system operations carried out on the servers
  - All the operations are supported

- Advantages:
  - Not requiring much space on the clients
  - Eliminates the need to pull in entire files when only small pieces are needed

- Disadvantages:
  - Accessing the same data repeatedly (caching can deal with this)

Remote Access



Client

Server

Requests from client to access the file

Remote Access Model

- Reduce network traffic by retaining recently accessed disk blocks in a cache, so that repeated accesses to the same information can be handled locally.
  - If needed data not already cached, a copy of data is brought from the server to the user.
  - Accesses are performed on the cached copy.
  - Files identified with one master copy residing at the server machine, but copies of (parts of) the file are scattered in different caches.
  - Cache-consistency problem – keeping the cached copies consistent with the master file.

Cache Location - Disk vs. Memory

- Advantages of disk caches
  - More reliable.
  - Cached data kept on disk are still there during recovery and don’t need to be fetched again.

- Advantages of main-memory caches:
  - Permit workstations to be diskless.
  - Data can be accessed more quickly.
  - Performance speedup in bigger memories.
  - Server caches (used to speed up disk I/O) are in main memory regardless of where user caches are located.
    - Using main-memory caches on the user machine permits a single caching mechanism for servers and users.

Cache Update Policy

- Write-through – write data through to disk as soon as they are placed on any cache.
  - Reliable, but suffers from poor performance.

- Delayed-w rite – modifications written to the cache and then written through to the server later.
  - Good Performance; Write accesses complete quickly; some data may be overwritten before they are written back, and so need never be written at all.

  - Poor reliability; unwritten data will be lost whenever a user machine crashes.

Cache consistency

- A key dilemma with the use of caching arises from the cache-consistency problem:
  - Is locally cached copy of the data consistent with the master copy?

- Client-initiated approach
  - Client initiates a validity check.
  - Server checks whether the local data is consistent with the master copy.

- Server-initiated approach
  - Server records, for each client, the (parts of) files it caches.
  - When server detects a potential inconsistency, it must react.
    - For example, it can turn off caching for the “offending” file.

Benefits of Caching

- In caching, many remote accesses handled efficiently by the local cache.
  - Therefore, most remote accesses will be served as fast as local ones.

- Caching is superior in access patterns with infrequent writes.
  - With frequent writes, substantial overhead incurred to overcome cache- consistency problem.

Benefits of Caching

- Lower total network overhead
  - Disk access routines on the server can be optimized if it is known that requests are always for large, contiguous segments of data, rather than many requests for random disk blocks.

- Benefit from caching when execution carried out on machines with either local disks or large main memories.
  - Caching should not be used to support remote access on diskless, small-memory-capacity machines.

Thank you

For general enquries, contact:

Please contact aidan.murphy@ucd.ie