# Leader Election

**Synchronization** can be quite costly: if each algorithm step involves contacting each other participant, we can end up with a significant **communication overhead**. This is particularly true in large and geographically distributed networks.

To reduce synchronization overhead and the number of message round-trips required to reach a decision, some algorithms rely on the existence of the **leader** (sometimes called coordinator) process, responsible for executing or coordinating steps of a distributed algorithm.

Generally, processes in distributed systems are uniform, and **any process** can take over the leadership role. Processes assume leadership for long periods of time, but this is not a permanent role. Usually, the process remains a leader **until it crashes**. After the crash, any other process can start a new **election** round, assume leadership, if it gets elected, and continue the failed leader's work.

Leader election is the simple idea of giving one thing (a process, host, thread, object, or human) in a distributed system some special powers.

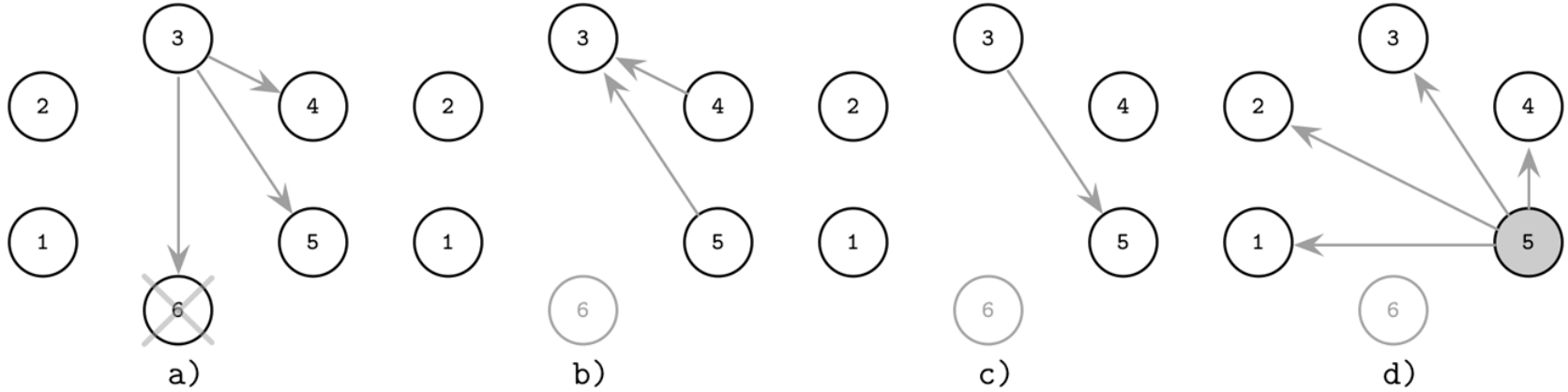
Election is triggered when the system initializes, and the leader is elected for the first time, or when the previous leader crashes or fails to communicate. Election has to be **deterministic**: **exactly one leader** has to emerge from the process. This decision needs to be effective for all participants.

## Bully Algorithm

Bully algorithm uses process ranks to identify the new leader. Each process gets a unique rank assigned to it. During the election, the process with the highest rank becomes a leader.
This algorithm is known for its simplicity. The algorithm is named bully because the highest-ranked node "bullies" other nodes into accepting it.

Election starts if one of the processes notices that there's no leader in the system (it was never initialized) or the previous leader has stopped responding to requests, and proceeds in three steps:

1. The process **sends** election messages to processes with higher identifiers.
2. The process **waits**, allowing higher-ranked processes to respond. If no higher-ranked process responds, it proceeds with step 3. Otherwise, the process notifies the highest-ranked process it has heard from, and allows it to proceed with step 3.
3. The process assumes that there are no active processes with a higher rank, and **notifies** all lower-ranked processes about the new leader.




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.

One of the apparent problems with this algorithm is that it **violates the safety guarantee** (that at most one leader can be elected at a time) in the presence of network partitions. It is quite easy to end up in the situation where nodes get split into two or more independently functioning subsets, and each subset elects its leader. This situation is called split brain.

Another problem with this algorithm is a strong preference toward **high-ranked nodes**, which becomes an issue if they are **unstable** and can lead to a permanent state of reelection. An unstable high-ranked node proposes itself as a leader, fails shortly thereafter, wins reelection, fails again, and the whole process repeats. This problem can be solved by distributing **host quality metrics** and taking them into consideration during the election.

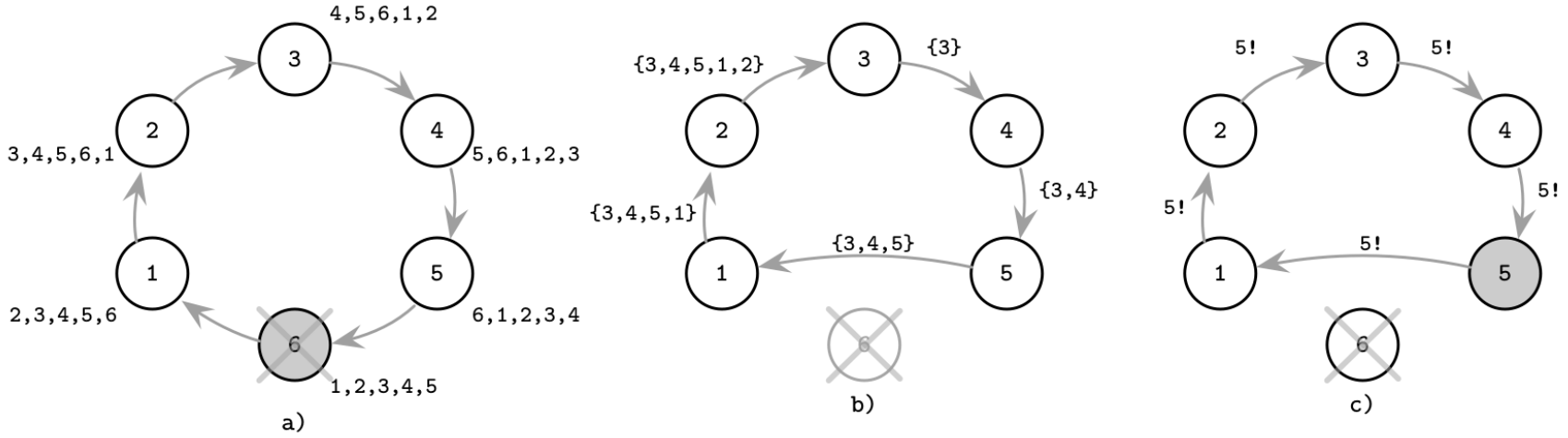
# Ring Algorithm

This algorithm applies to systems organized as a ring (logically or physically). In this algorithm, we assume that the link between the process is unidirectional and every process can message to the process on its right only.

When the process detects the leader failure, it starts the new election. The **election message is forwarded across the ring**: each process contacts its successor (the next node closest to it in the ring). If this node is unavailable, the process skips the unreachable node and attempts to contact the nodes after it in the ring, until eventually one of them responds.

Nodes contact their siblings, following around the ring and **collecting the live node set**, adding themselves to the set before passing it over to the next node.

The algorithm proceeds by **fully traversing the ring**. When the message comes back to the node that started the election, the highest-ranked node from the live set is chosen as a leader.



## Summary

Leader election is an important subject in distributed systems, since using a designated leader helps to reduce coordination overhead and improve the algorithm's performance. Election rounds might be costly but, since they're infrequent, they do not have a negative impact on the overall system performance. A single leader can become a bottleneck, but most of the time this is solved by partitioning data and using **per-partition** leaders or using different leaders for different **actions**.

Unfortunately, all the algorithms we've discussed in this chapter are prone to the split brain problem: we can end up with two leaders in independent subnets that are not aware of each other's existence. To avoid split brain, we have to obtain a cluster-wide majority of votes.

## Application

How Amazon elects a leader

References

https://en.wikipedia.org/wiki/Leader_election
https://www.geeksforgeeks.org/election-algorithm-and-distributed-processing/
https://aws.amazon.com/builders-library/leader-election-in-distributed-systems/