Distributed Algorithms and Design Patterns

Distributed Computing
(SE5090)

Lasitha Petthawadu

Founder | CEO

Tetranyde

@lasitha_ishan

Distributed Algorithms

Algorithms that make use of distributed nodes and decisions derived from them.

Distributed Algorithms
Usages

Distributed systems generally need rudimentary algorithms that help them solve other problems.

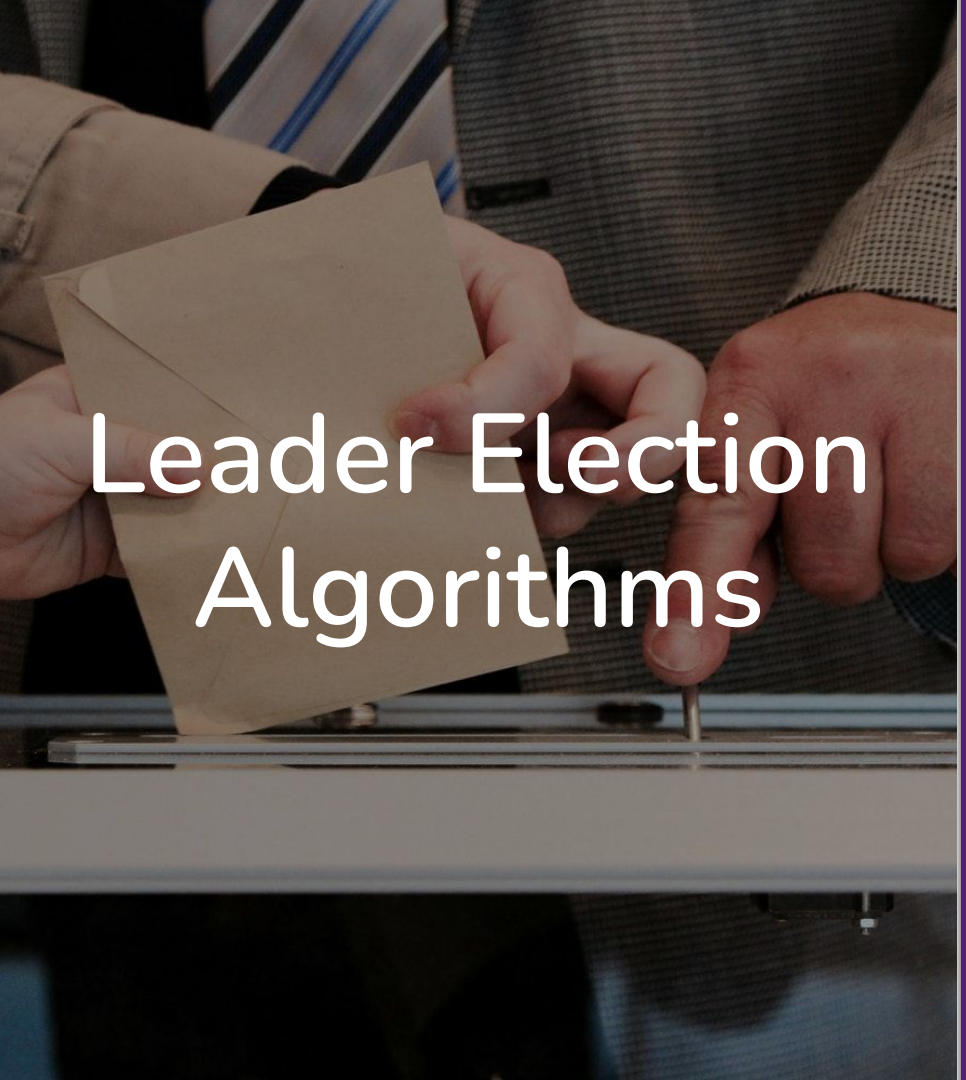
These algorithms are the building blocks of other larger algorithms.

Distributed Algorithms
Usages

These algorithms help solve common problems.
In distributed systems.

Distributed Algorithms
Types of Algorithms

- Leader Election Algorithms
- Consensus Algorithms



Leader Election Algorithms

Leader election algorithms allow nodes to decide who the leader is

Leader Election Algorithms
Why Leader election algorithms

- Deciding a coordinator is important in many distributed systems.
- Allows to re-elect a leader in case the main leader dies.

Leader Election Algorithms
Types of Leader Election Algorithms

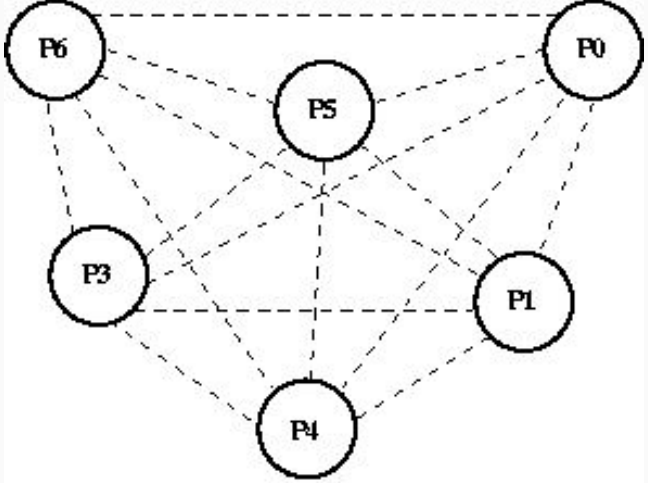
- Bully Algorithm
- Leader Ring Algorithm

Leader Election Algorithms
Bully Algorithm

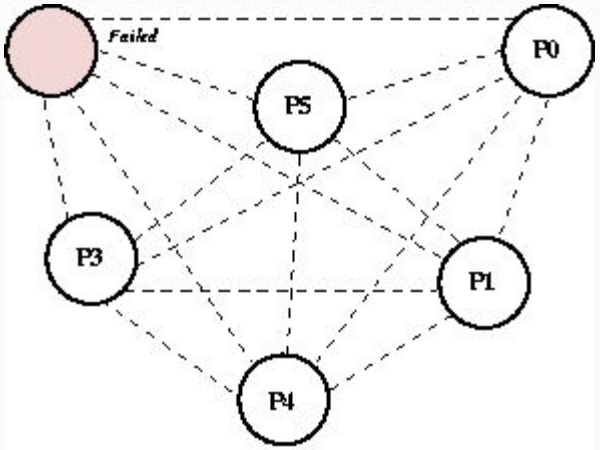
1. Each process has a unique Id
2. They broadcast their Ids
3. The process with the **highest Id** would become the leader.

Leader Election Algorithms

Bully Algorithm



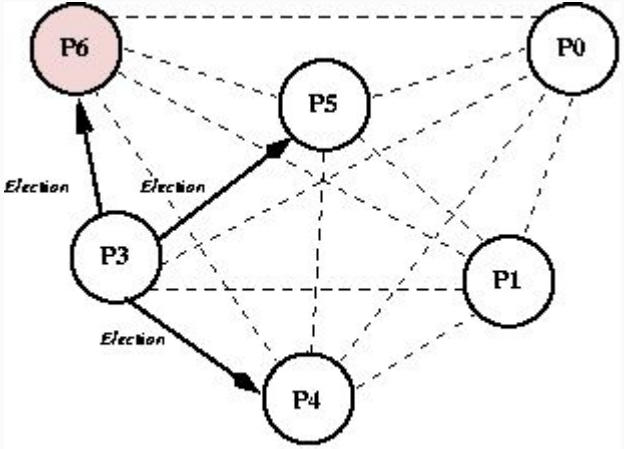
Bully Algorithm: Step 0



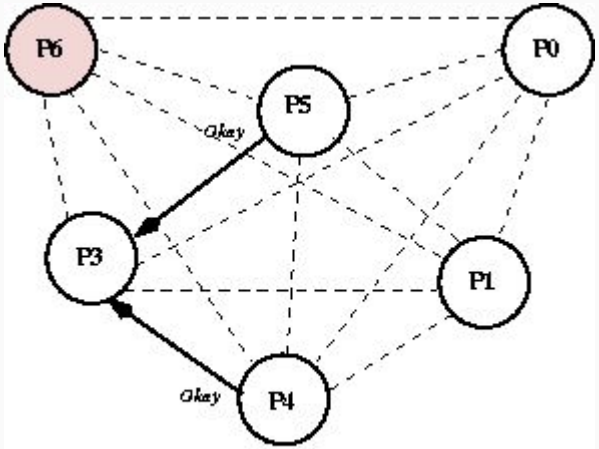
Bully Algorithm: Step 1

Leader Election Algorithms

Bully Algorithm



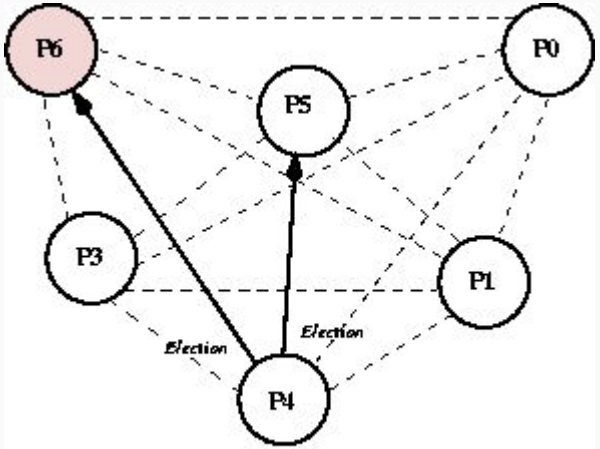
Bully Algorithm: Step 2



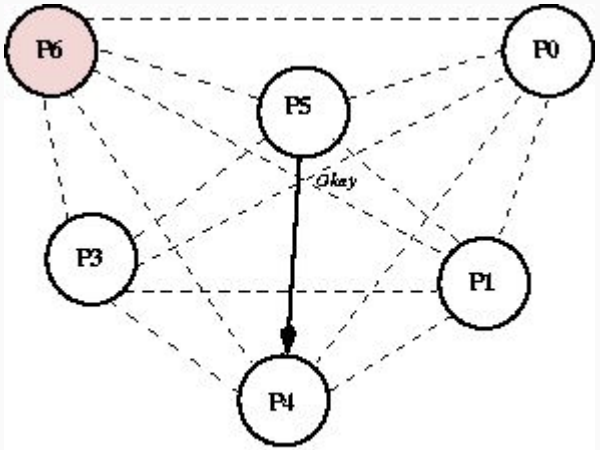
Bully Algorithm: Step 3

Leader Election Algorithms

Bully Algorithm



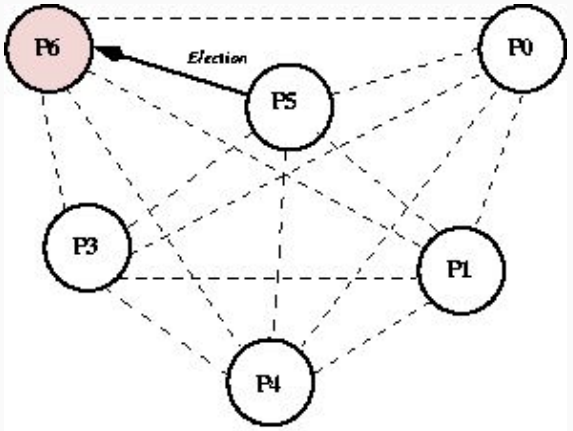
Bully Algorithm: Step 4



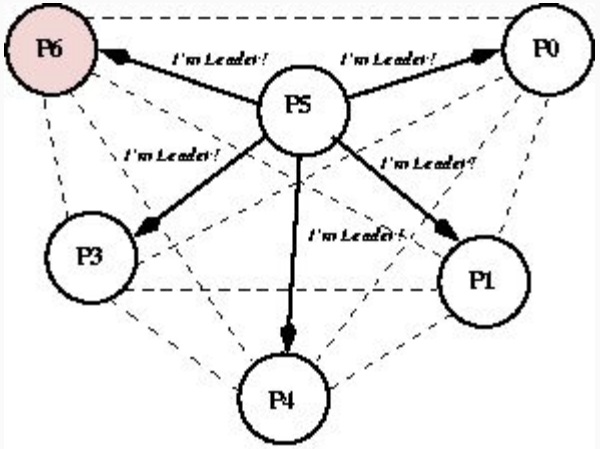
Bully Algorithm: Step 5

Leader Election Algorithms

Bully Algorithm

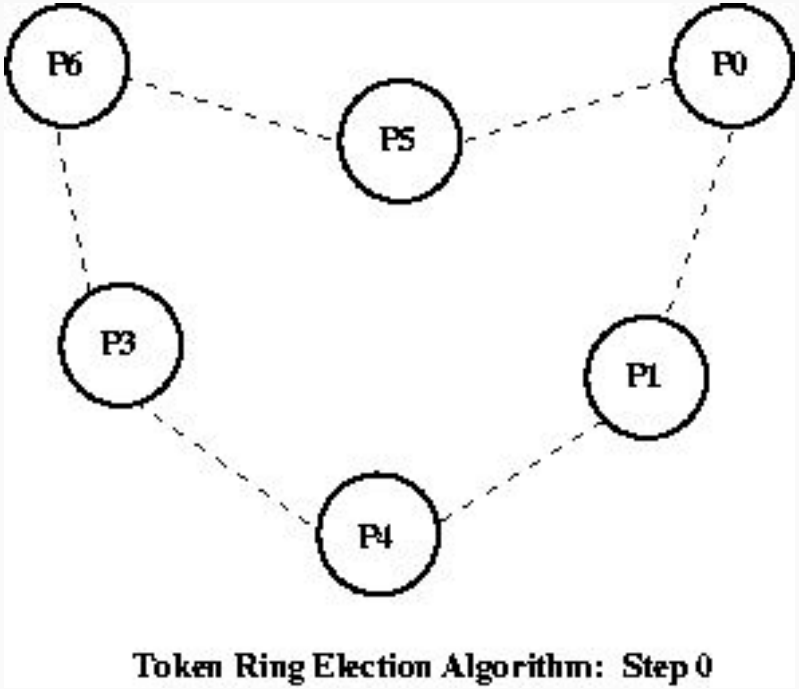


Bully Algorithm: Step 6

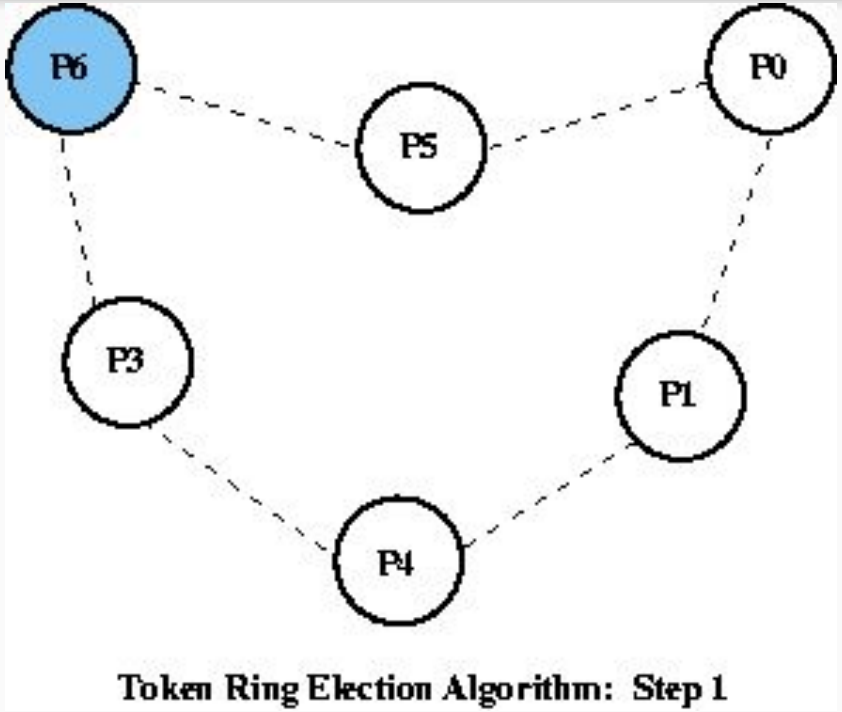


Bully Algorithm: Step 7

Leader Election Algorithms
Ring Algorithm

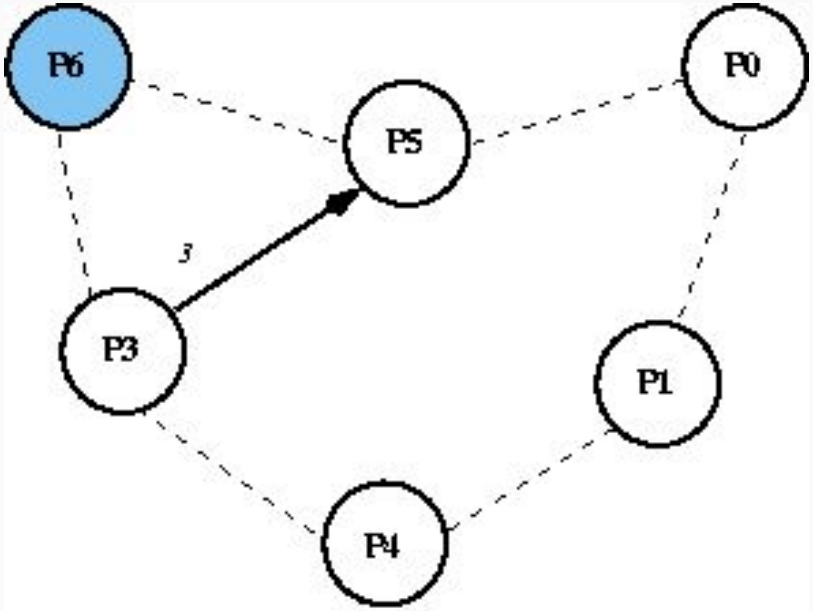


Token Ring Election Algorithm: Step 0

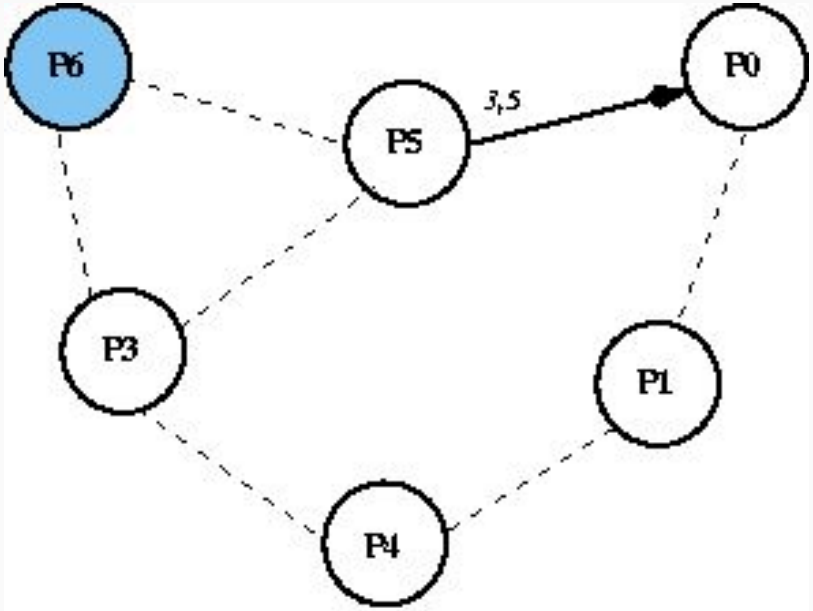


Token Ring Election Algorithm: Step 1

Leader Election Algorithms
Ring Algorithm

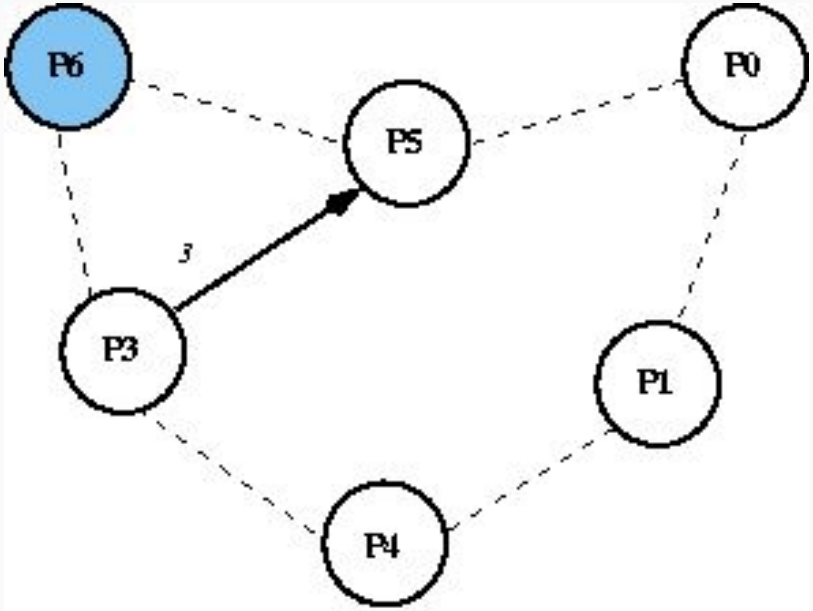


Token Ring Election Algorithm: Step 2

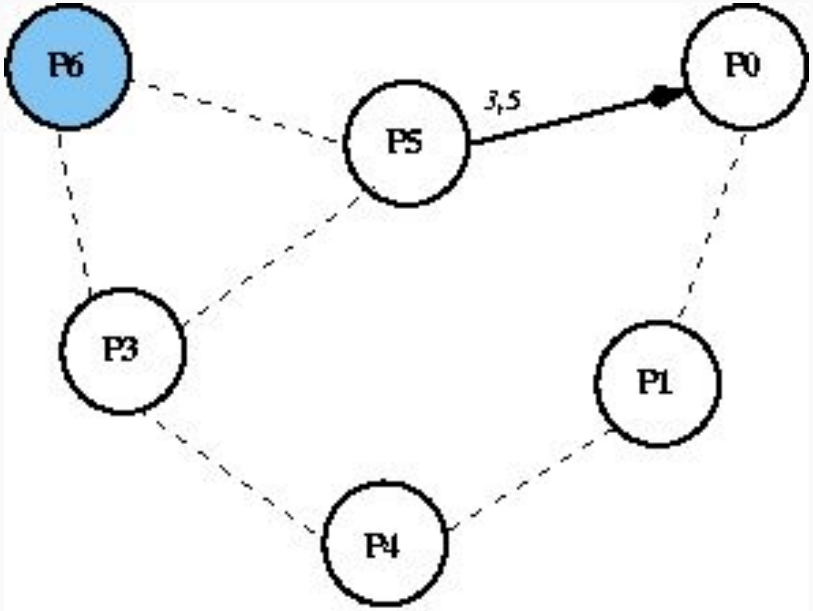


Token Ring Election Algorithm: Step 3

Leader Election Algorithms
Ring Algorithm

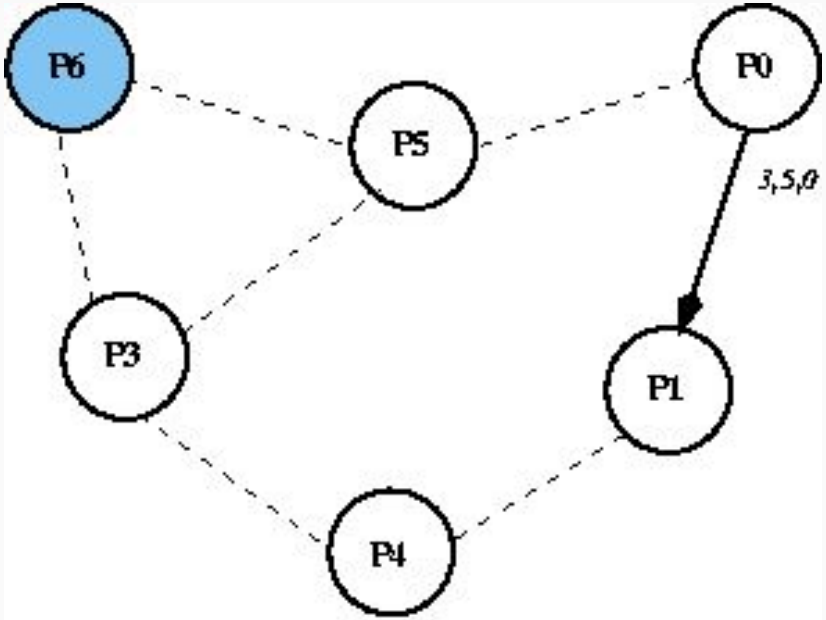


Token Ring Election Algorithm: Step 2

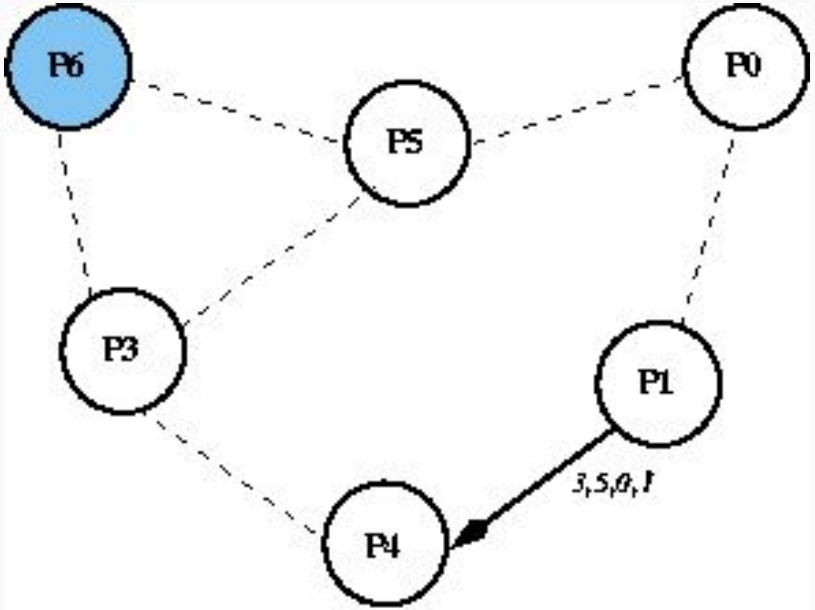


Token Ring Election Algorithm: Step 3

Leader Election Algorithms
Ring Algorithm

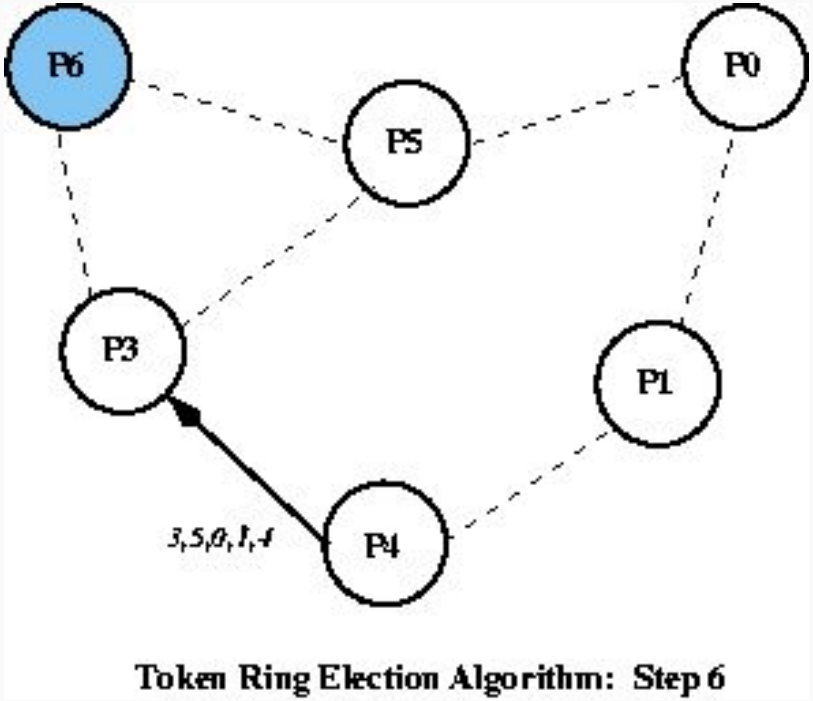


Token Ring Election Algorithm: Step 4

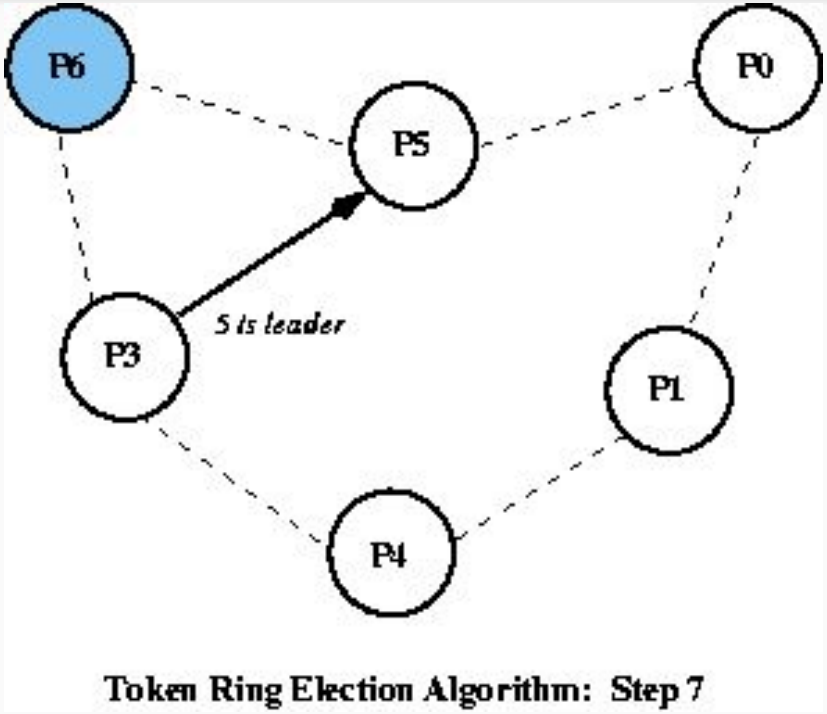


Token Ring Election Algorithm: Step 5

Leader Election Algorithms
Ring Algorithm

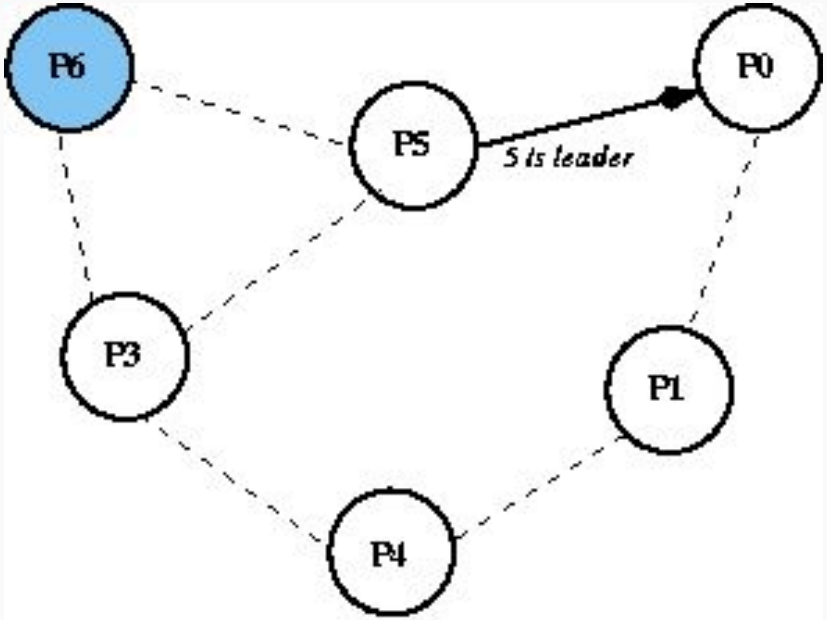


Token Ring Election Algorithm: Step 6

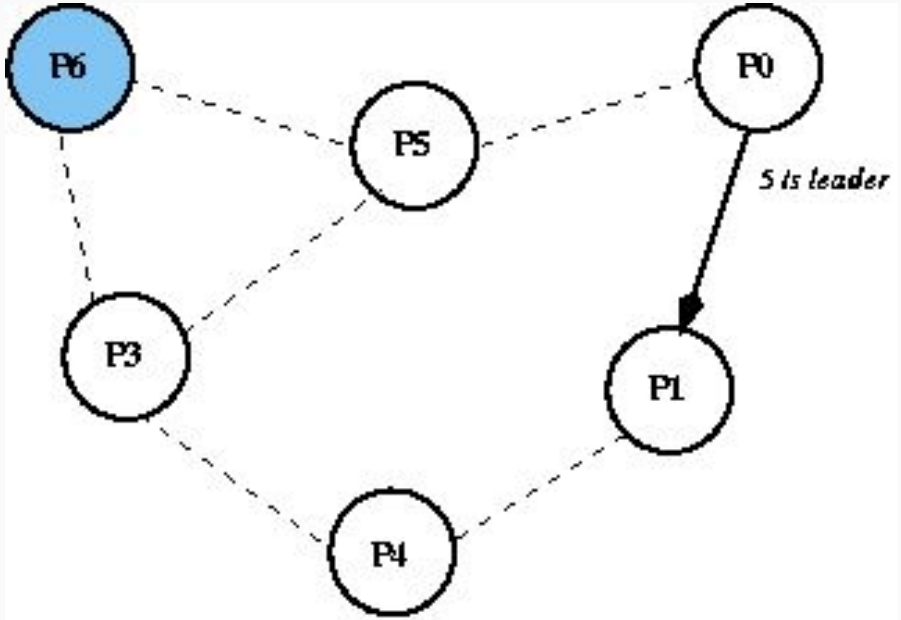


Token Ring Election Algorithm: Step 7

Leader Election Algorithms
Ring Algorithm

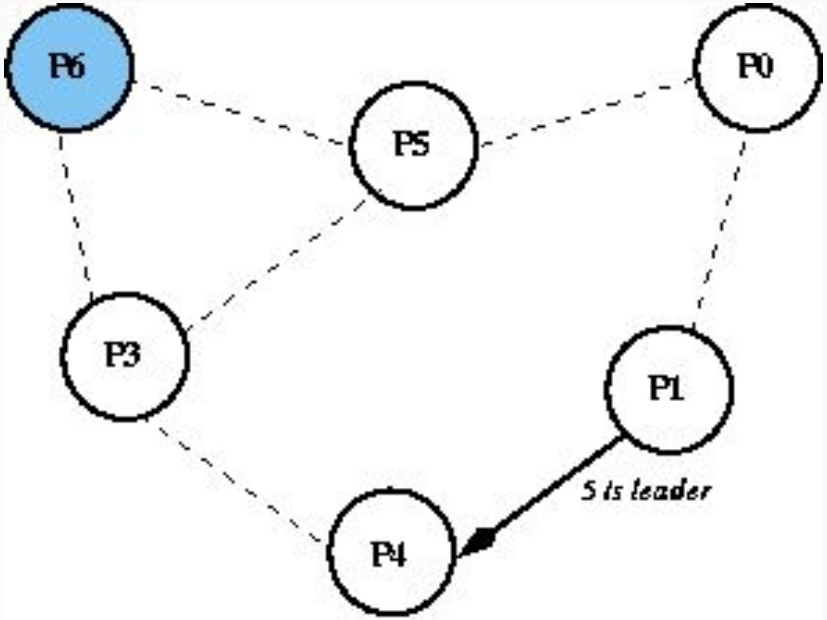


Token Ring Election Algorithm: Step 8

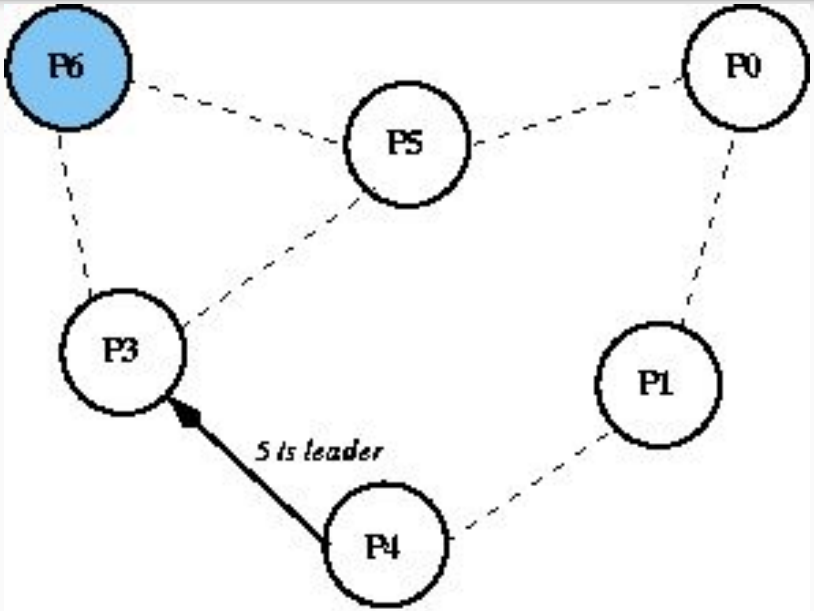


Token Ring Election Algorithm: Step 9

Leader Election Algorithms
Ring Algorithm



Token Ring Election Algorithm: Step 10



Token Ring Election Algorithm: Step 11

Consensus Algorithms

Consensus algorithms allow a collection of nodes to come to a common agreement.

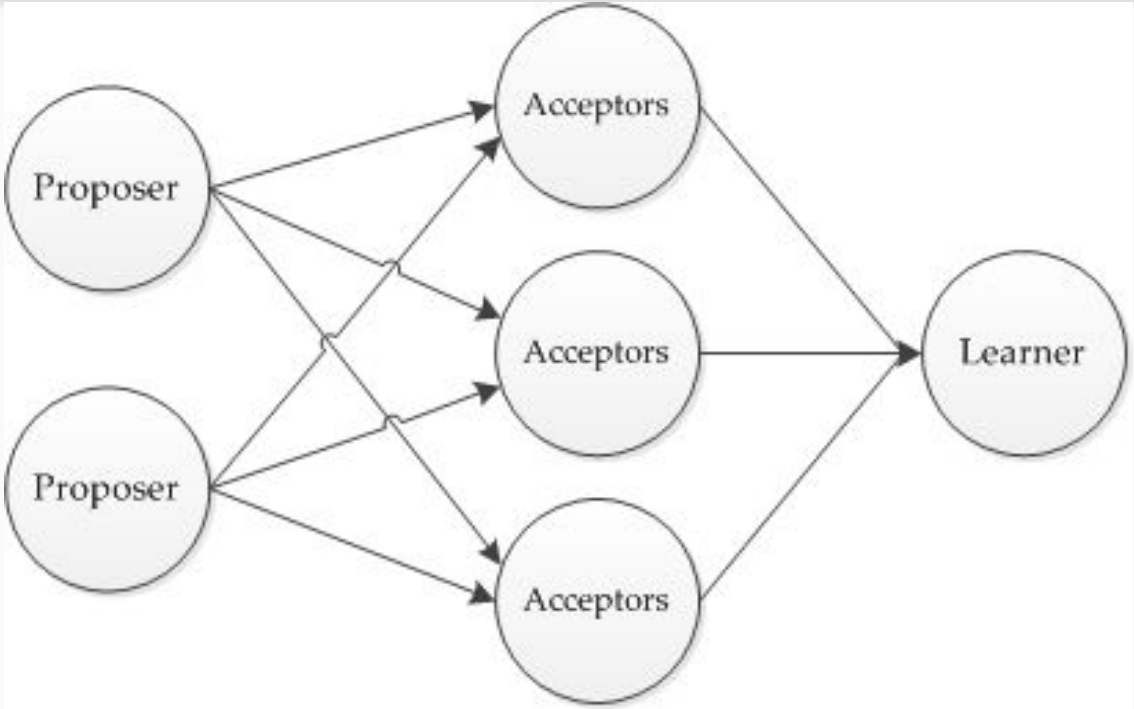
Consensus Algorithms
Real-world - Usages

- Deciding between nodes whether to commit a distributed transaction.
- Designating a node is a leader.
- Synchronizing replicas and ensuring consistency.

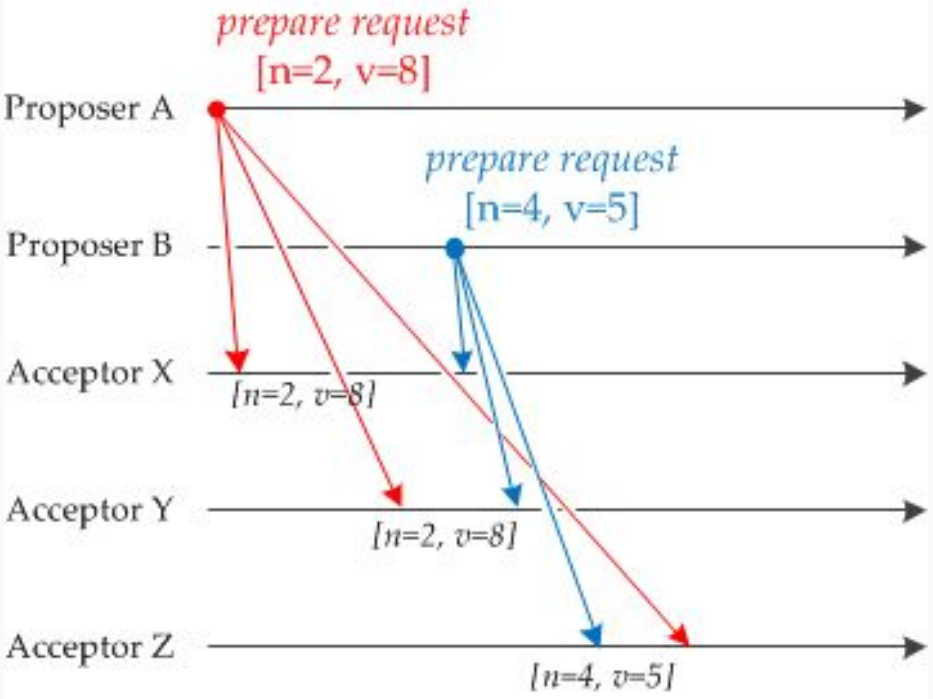
Consensus Algorithms
Types of Algorithms

- Paxos Algorithm
- Raft Algorithm

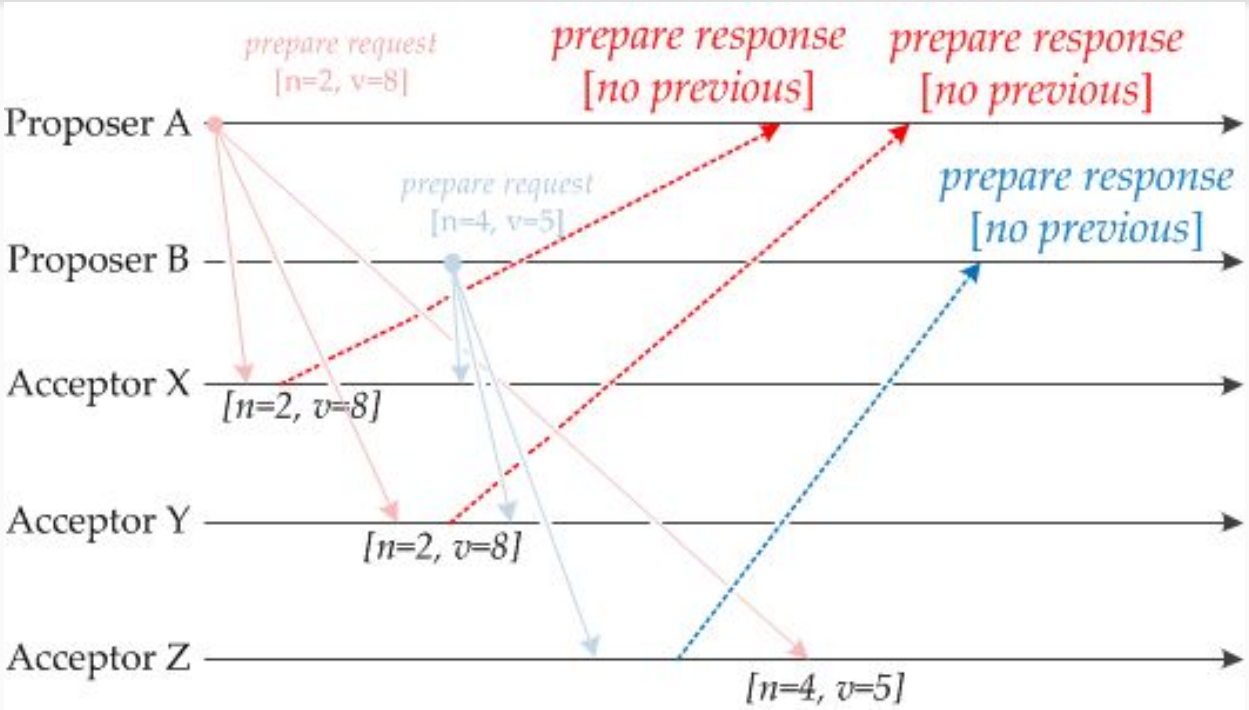
Consensus Algorithms
Paxos Algorithm



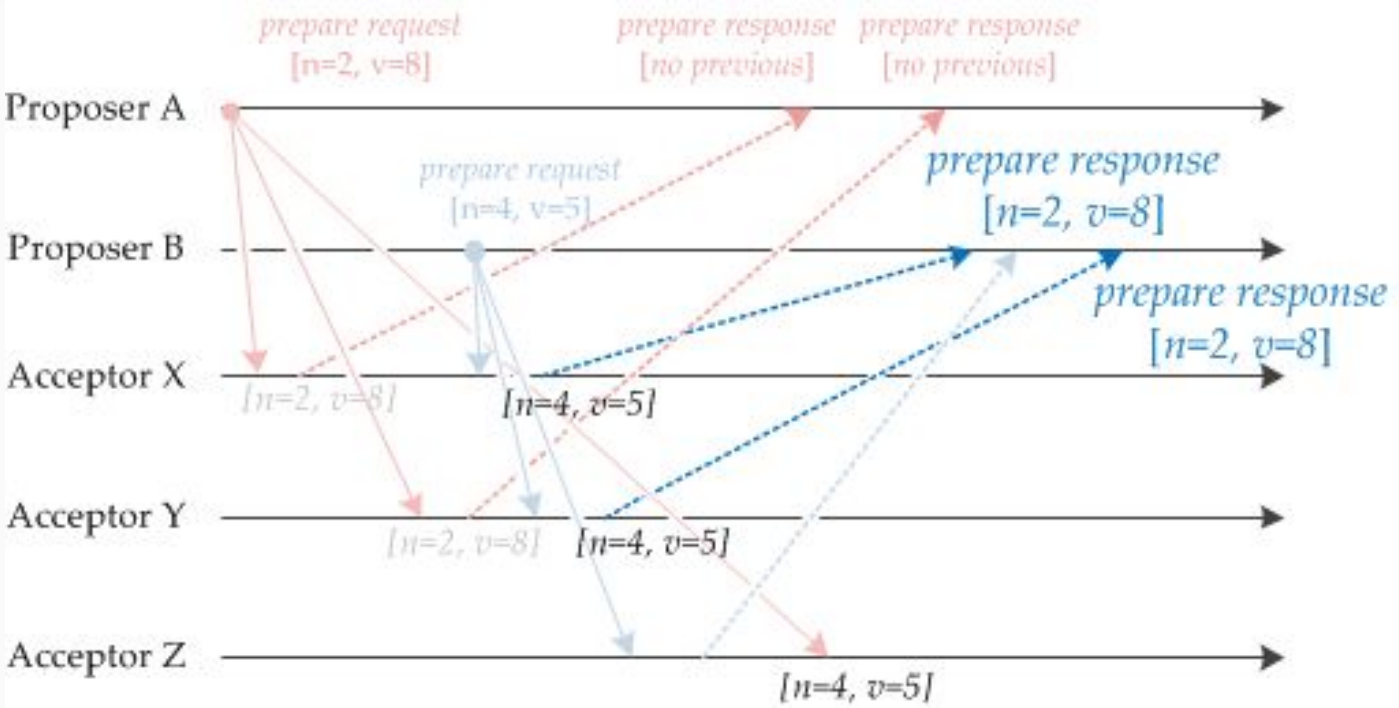
Consensus Algorithms
Paxos Algorithm



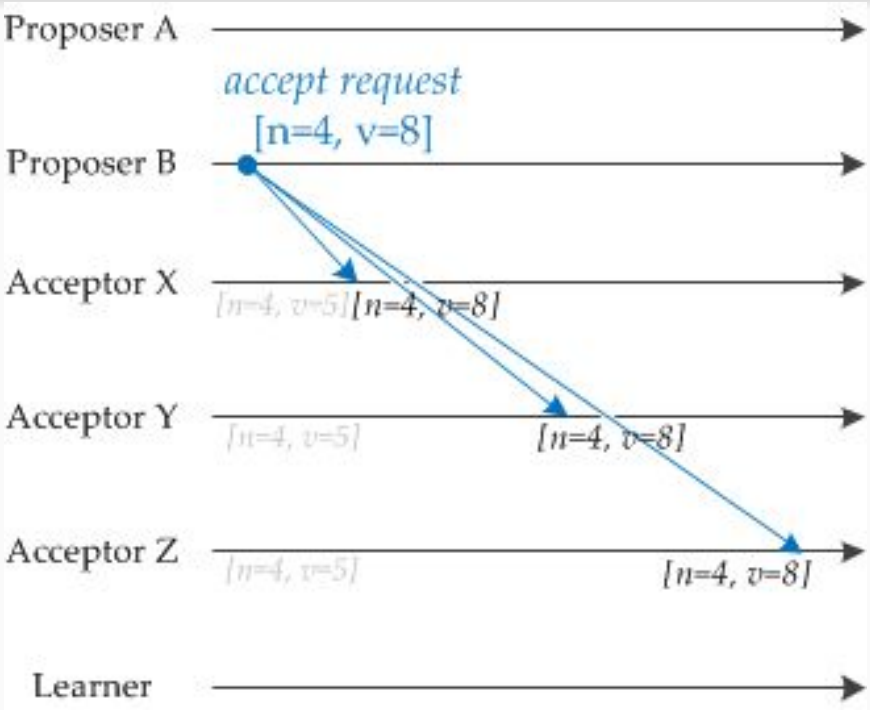
Consensus Algorithms
Paxos Algorithm



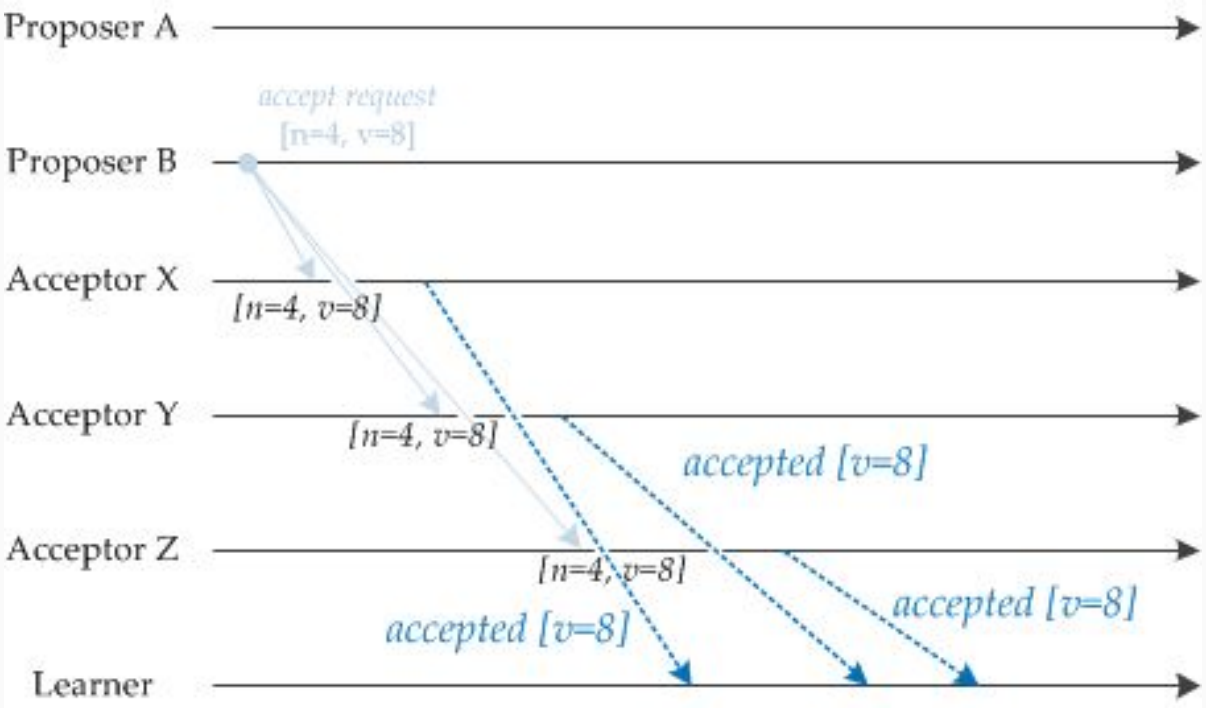
Consensus Algorithms
Paxos Algorithm



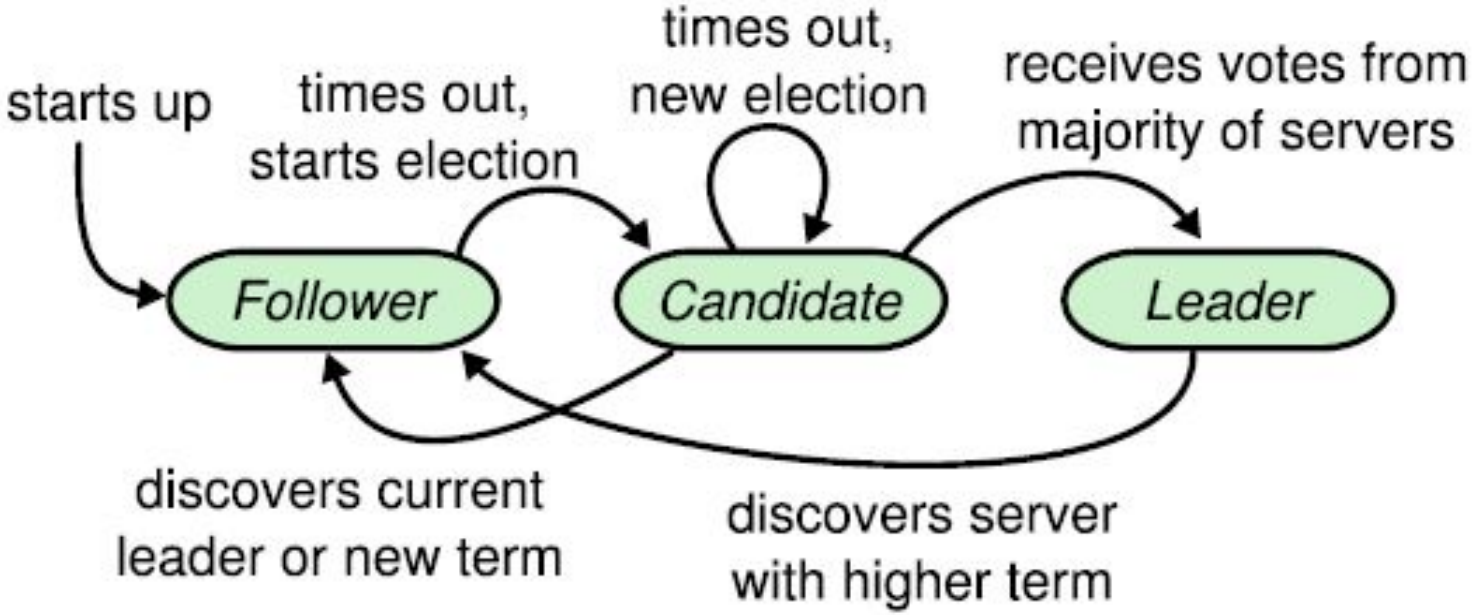
Consensus Algorithms
Paxos Algorithm

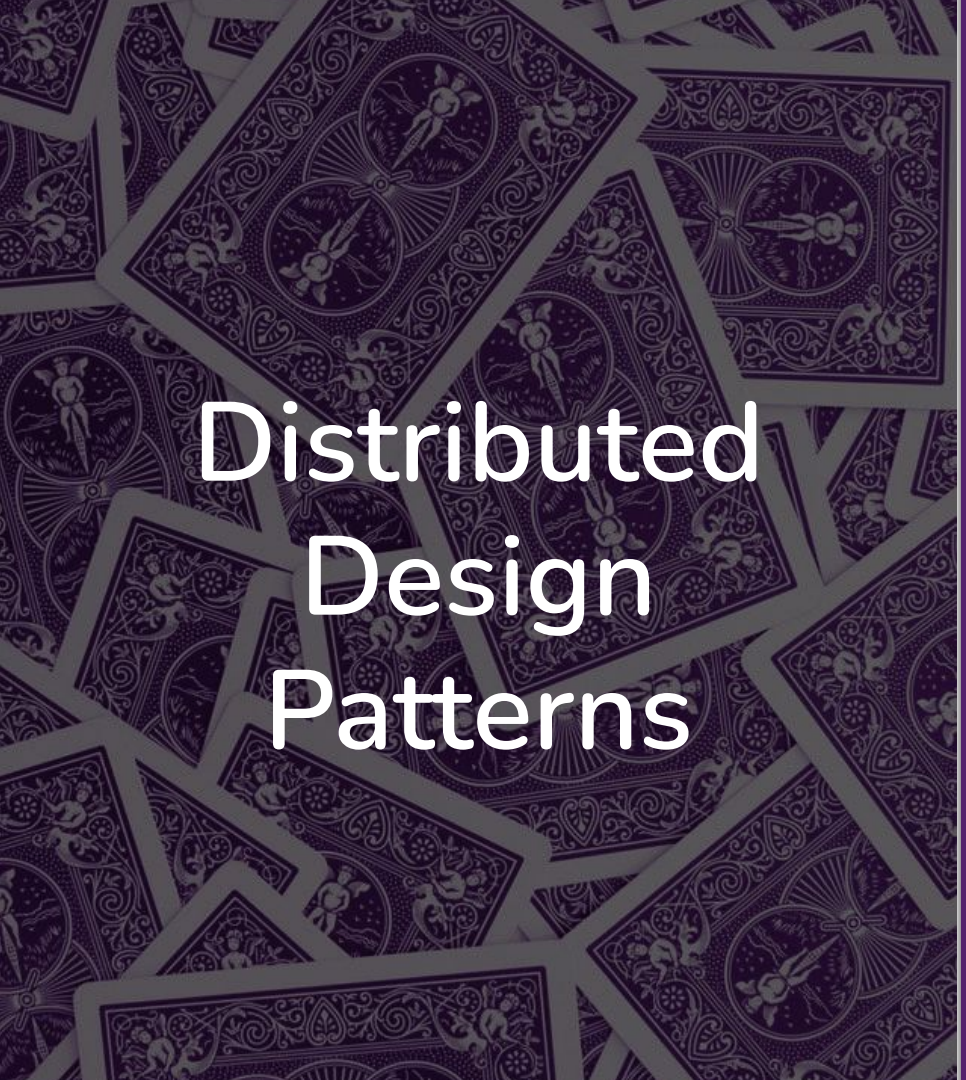


Consensus Algorithms
Paxos Algorithm



Consensus Algorithms
Raft Algorithm





Distributed Design Patterns

Design Patterns
WHY?

Distributed Design Patterns
Why?

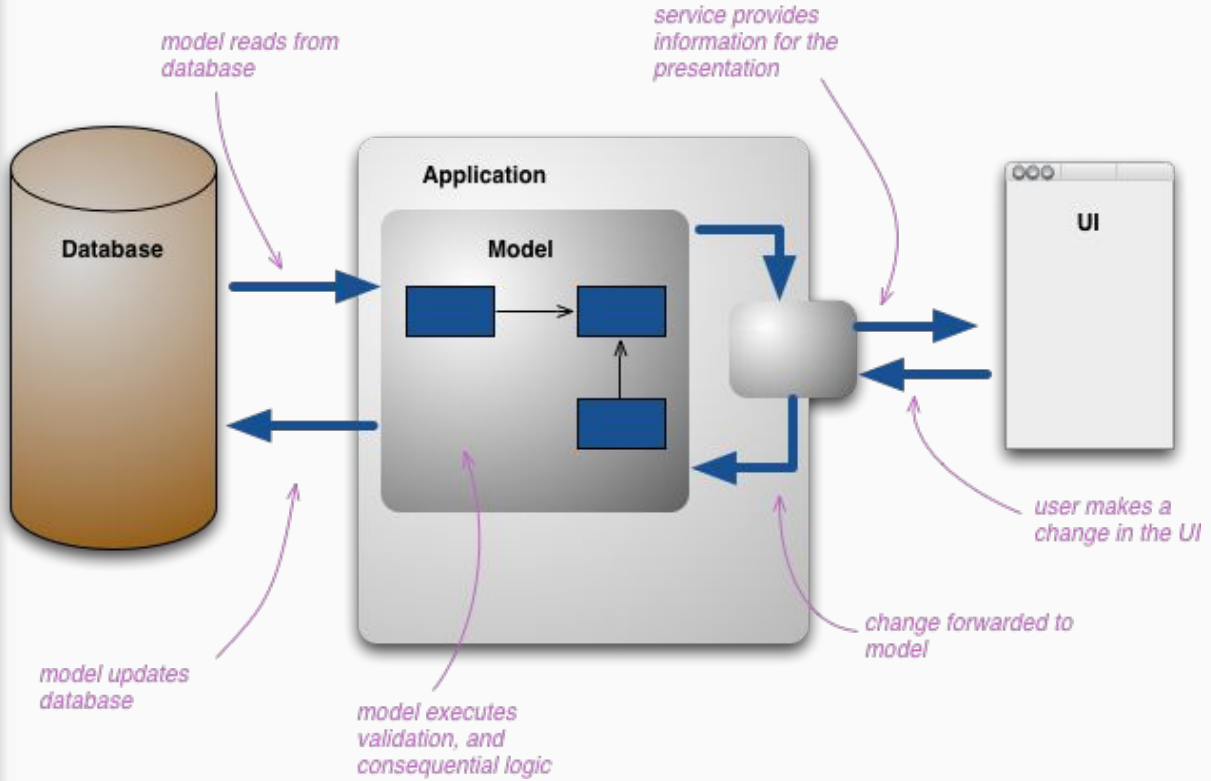
- Solves common recurring problems.
- Based on best practices.
- Templated solutions which could be customized and implemented.

Distributed Design Patterns

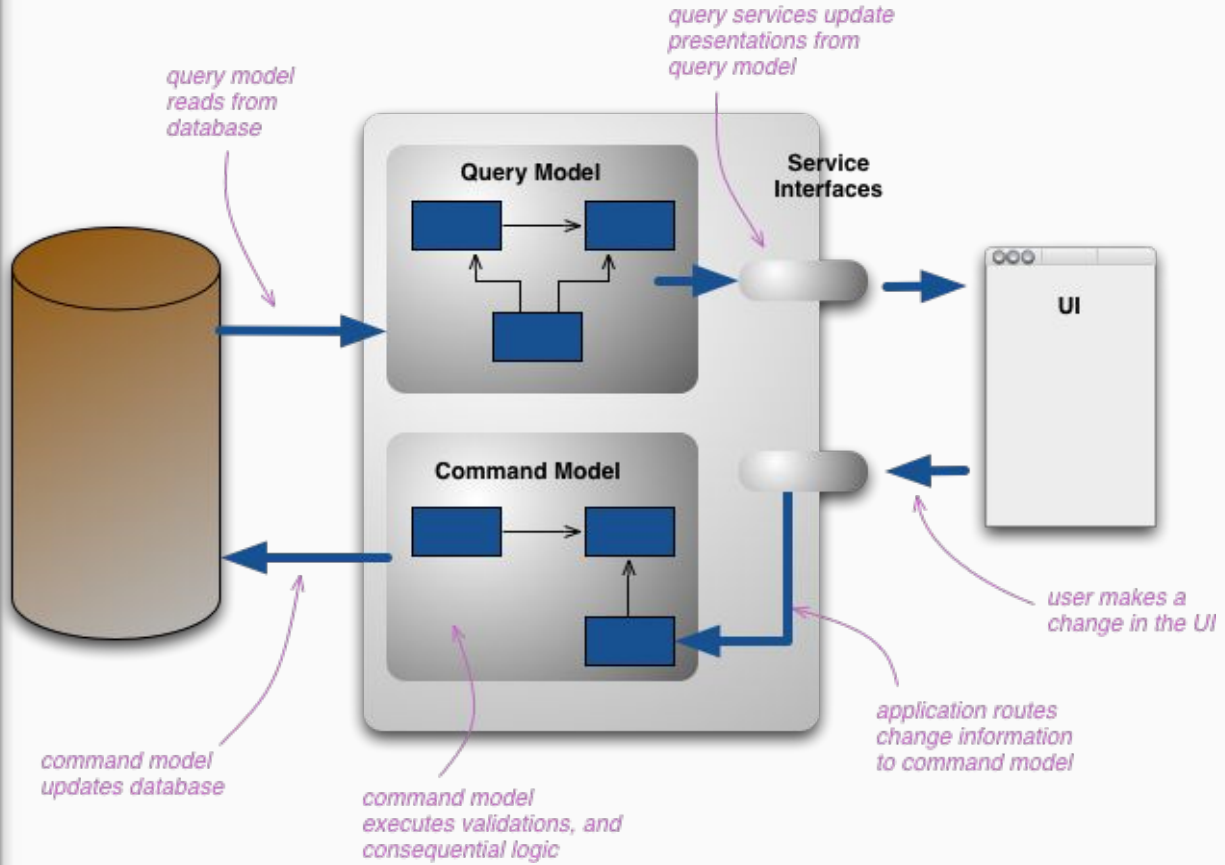
Command and Query Responsibility Segregation

- Allows to separate data querying and data write operations.
- Allows to use two different domain models for data write and data read operations.

Command and Query Responsibility Segregation (CQRS)



Command and Query Responsibility Segregation (CQRS)



Distributed Design Patterns

Command and Query Responsibility Segregation

- Don’t use for every solution!
- Check if the following requirements are needed.

Distributed Design Patterns
Command and Query Responsibility Segregation

- Use it when the write domain model is stacked with complex validations, business logics and entirely different from the read model

Distributed Design Patterns
Command and Query Responsibility Segregation

- Integration with other systems, especially in combination with ***event sourcing***, where the temporal failure of one subsystem shouldn't affect the availability of the others.

Distributed Design Patterns
Command and Query Responsibility Segregation

- UI is tasks based and user is guided through a complex process involving multiple steps.

Distributed Design Patterns
Command and Query Responsibility Segregation

- UI is tasks based and user is guided through a complex process involving multiple steps with partial writes.

Distributed Design Patterns
Command and Query Responsibility Segregation

- Performance of write operations and read operations need to be improved.

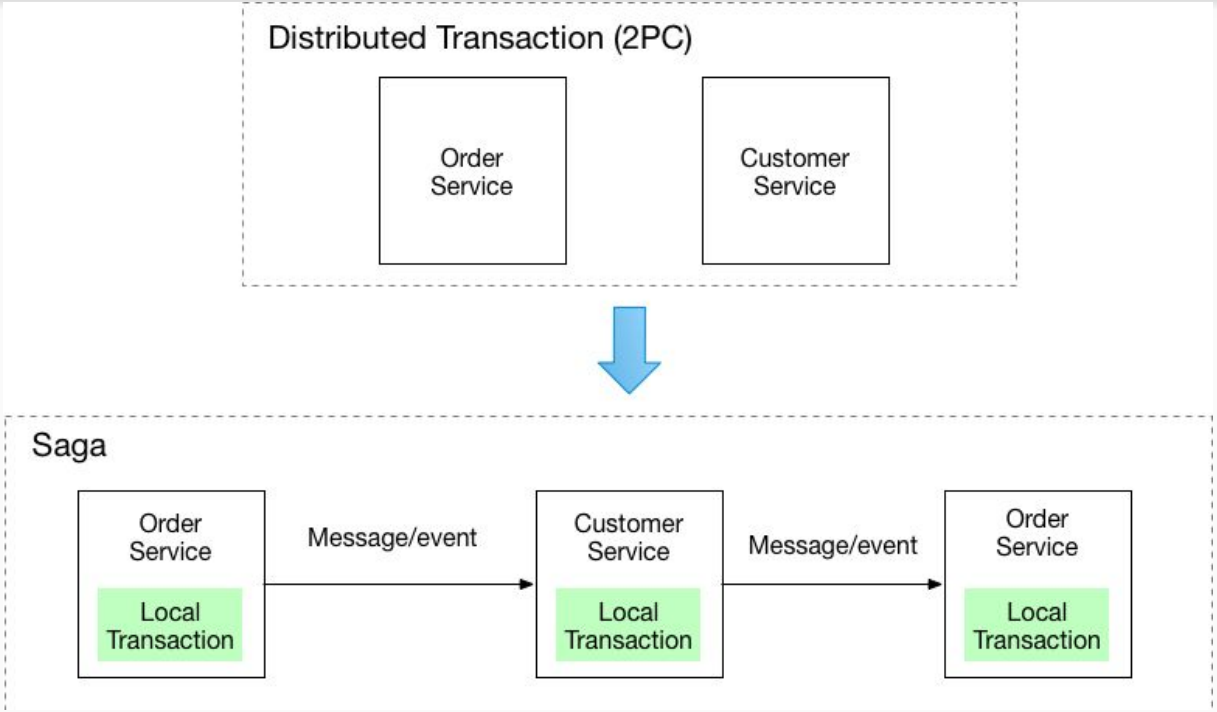
Distributed Design Patterns

Saga Design Pattern

A saga is a sequence of **local transactions** where each transaction updates data within a single service. The first transaction is initiated by an external request corresponding to the system operation, and then each subsequent step is triggered by the completion of the previous one.

Distributed Design Patterns

Saga Design Pattern



Distributed Design Patterns

Sidecar Pattern

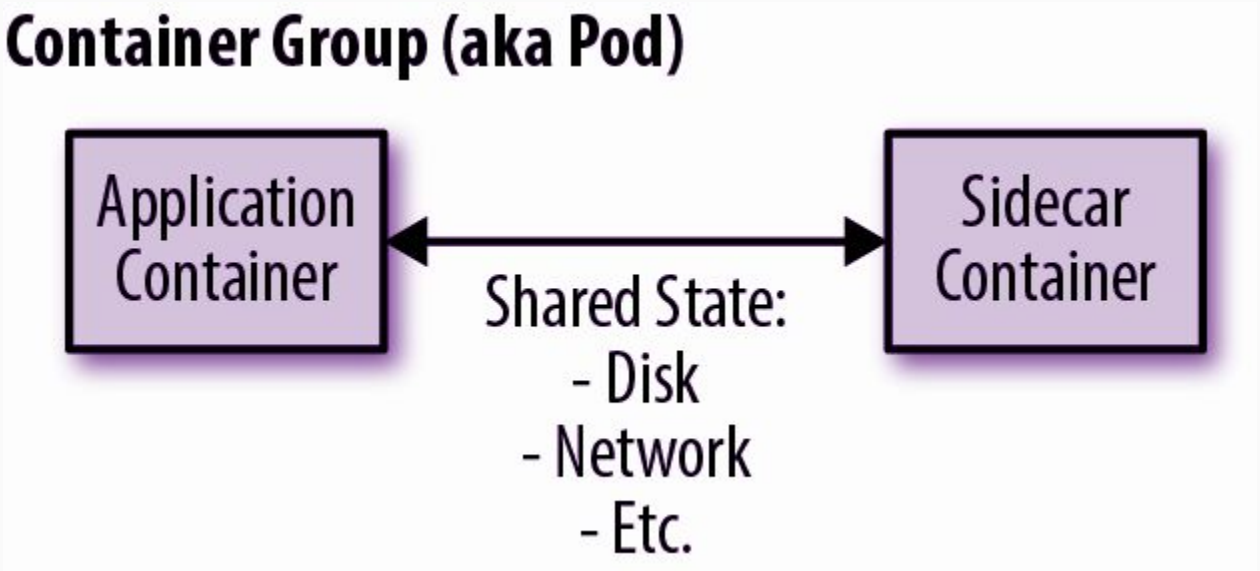
Distributed Design Patterns

Sidecar Pattern

- The sidecar pattern is a single-node pattern made up of two containers
- The first is the application container having the application
- The role of the sidecar is to augment and improve the application container, often without the application container's knowledge.

Distributed Design Patterns

Sidecar Pattern



Distributed Design Patterns

Sidecar Pattern - Usages

Your primary application uses a **heterogeneous** set of languages and frameworks. A component located in a sidecar service can be consumed by applications written in different languages using different frameworks.

Distributed Design Patterns

Sidecar Pattern - Usages

You need fine-grained control over resource limits for a particular resource or component. For example, you may want to restrict the amount of memory a specific component uses. You can deploy the component as a sidecar and manage memory usage independently of the main application.

Distributed Design Patterns

Sidecar Pattern - Usages

- Resource Logging
- Proxying
- Monitoring
- Caching
- Security Augmentation

Distributed Design Patterns

Strangler Pattern

Incrementally migrate a legacy system by gradually replacing specific pieces of functionality with new applications and services.

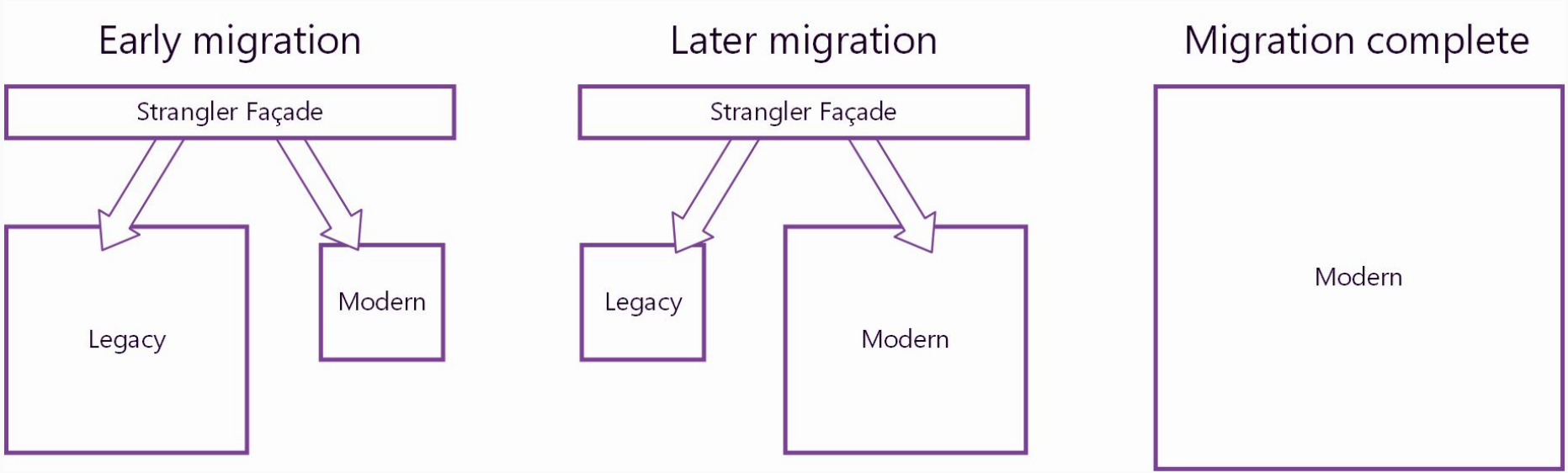
Distributed Design Patterns

Strangler Pattern

As features from the legacy system are replaced, the new system eventually replaces all of the old system's features, strangling the old system and allowing you to decommission it.

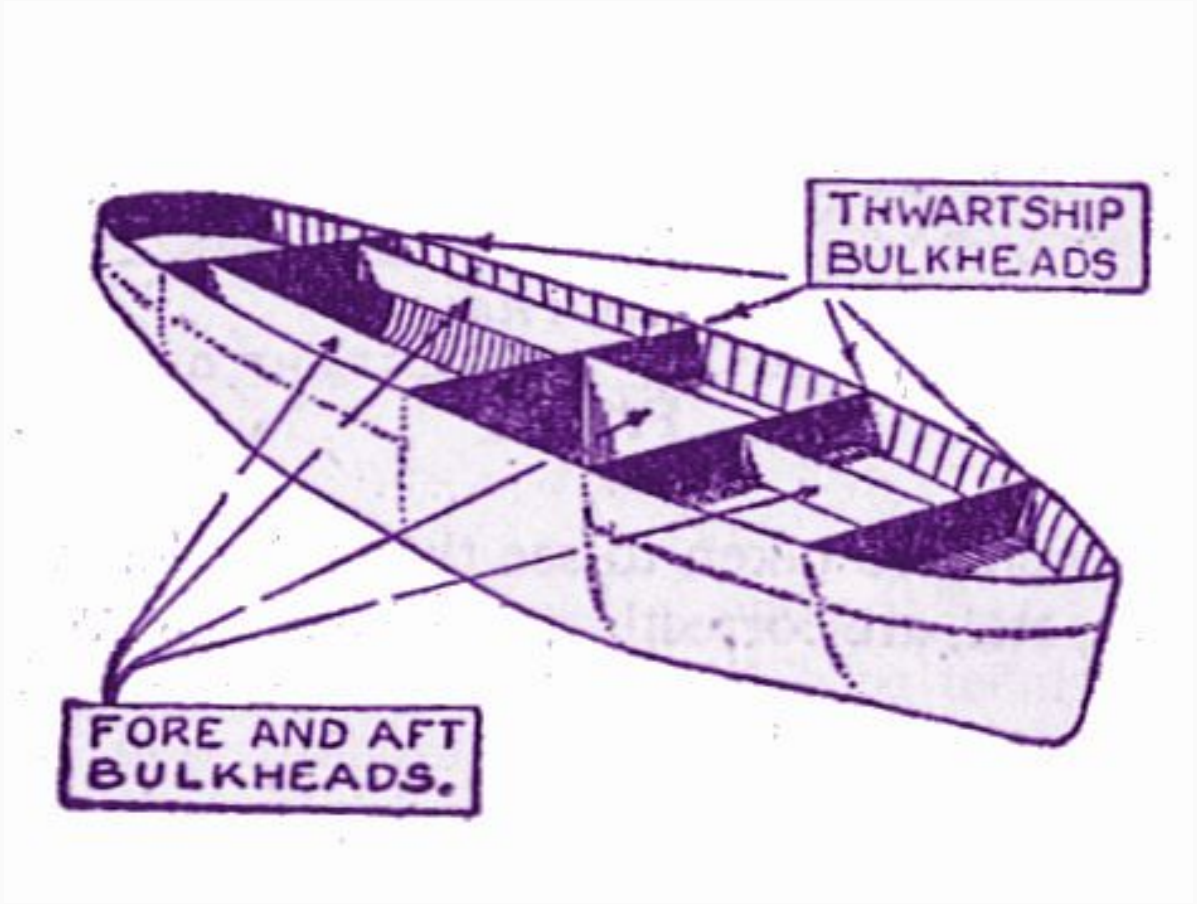
Distributed Design Patterns

Strangler Pattern



Distributed Design Patterns

Bulkhead Pattern



Distributed Design Patterns

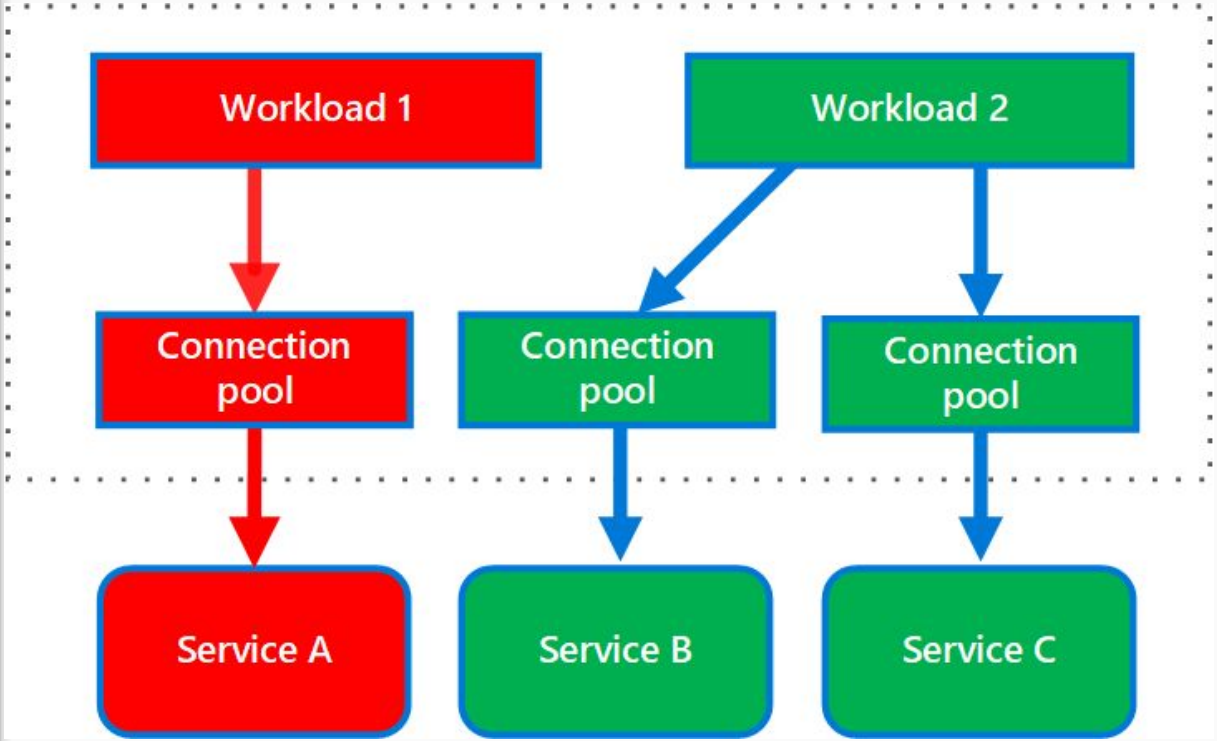
Bulkhead Pattern

The Bulkhead pattern is a type of application design that is tolerant of failure.

In a bulkhead architecture, elements of an application are **isolated into pools** so that if one fails, the others will continue to function.

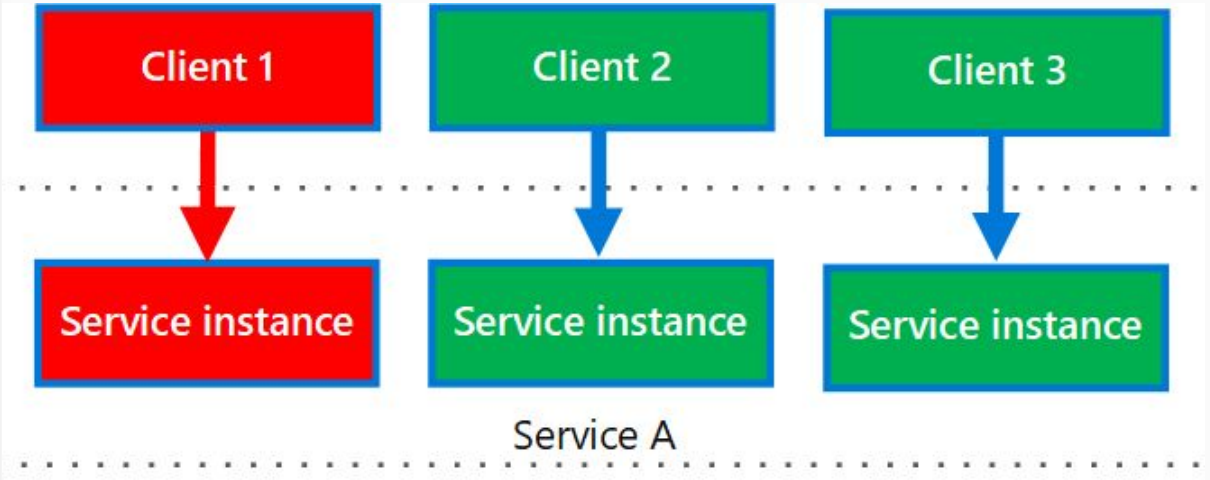
Distributed Design Patterns

Bulkhead Pattern



Distributed Design Patterns

Bulkhead Pattern



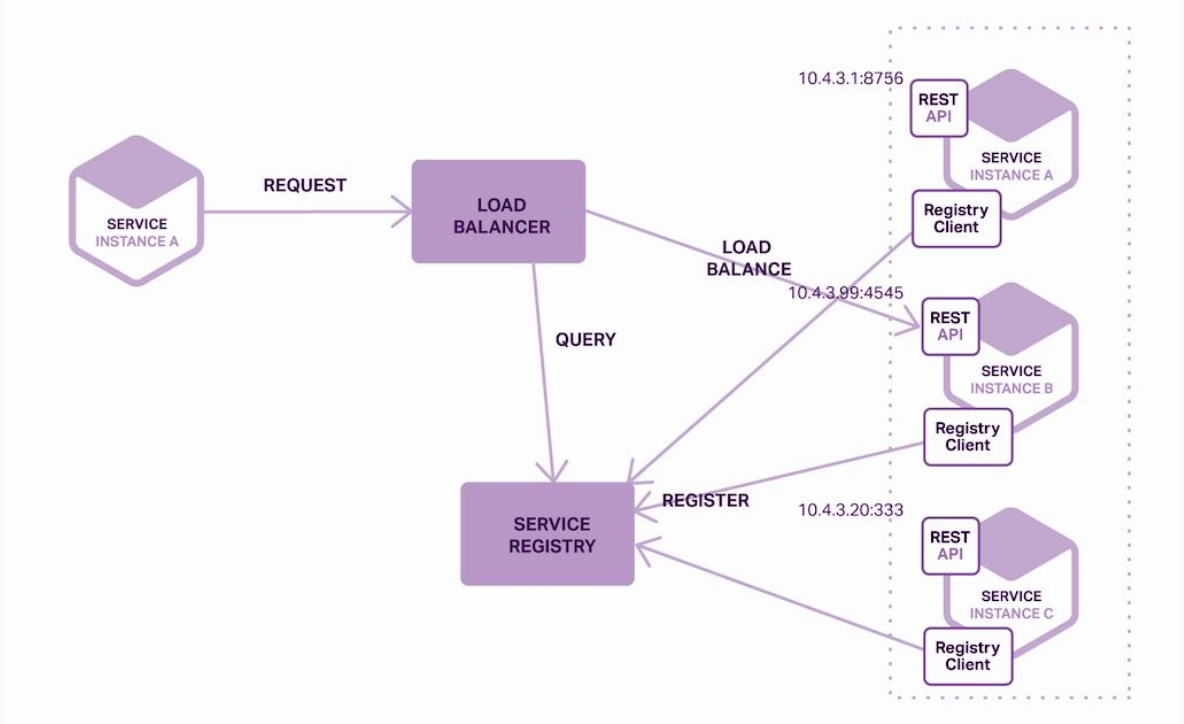
Distributed Design Patterns

Service Discovery Pattern

With distributed components its difficult to find out the the IP addresses and where services are located.

Distributed Design Patterns

Service Discovery Pattern



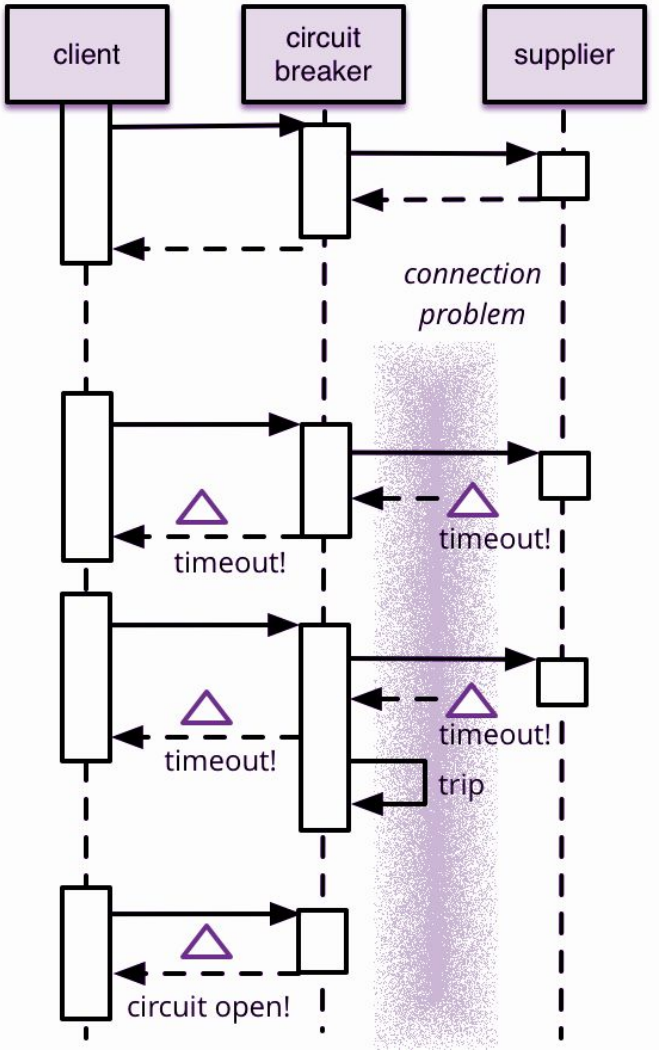
Distributed Design Patterns

Circuit Breaker Pattern

Circuit breaker is used to **detect failures and encapsulates the logic of preventing a failure from constantly recurring**, during maintenance, temporary external system failure or unexpected system difficulties.

Distributed Design Patterns

Circuit Breaker Pattern



Distributed Deployment Patterns

Distributed Design Patterns

Blue-green Deployment pattern

Allows to update distributed nodes with zero downtime.

All servers are marked as blue, these run the older version, next incrementally the new version is deployed while routing users to the old servers. Once the new versions are ready users are routed to them.

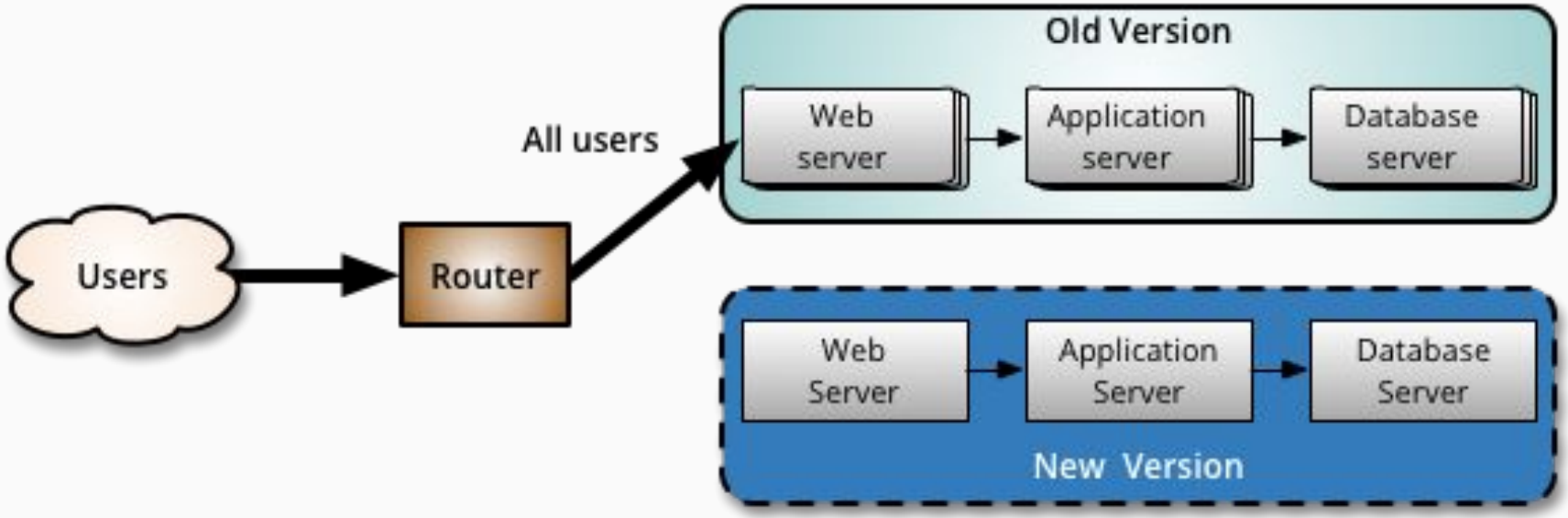
Distributed Design Patterns

Canary Deployment pattern

It is a technique to reduce the risk of introducing a new software version in production by slowly rolling out the change to a small subset of users before rolling it out to the entire infrastructure and making it available to everybody.

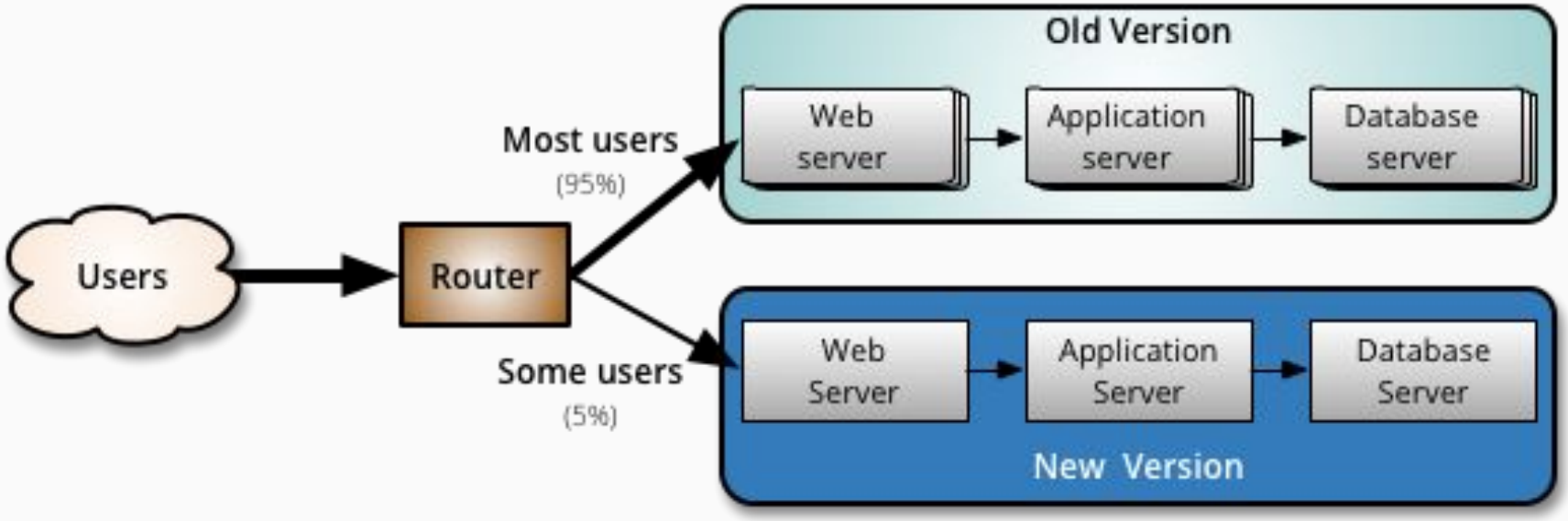
Distributed Design Patterns

Canary Deployment pattern



Distributed Design Patterns

Canary Deployment pattern



Distributed Design Patterns

Canary Deployment pattern

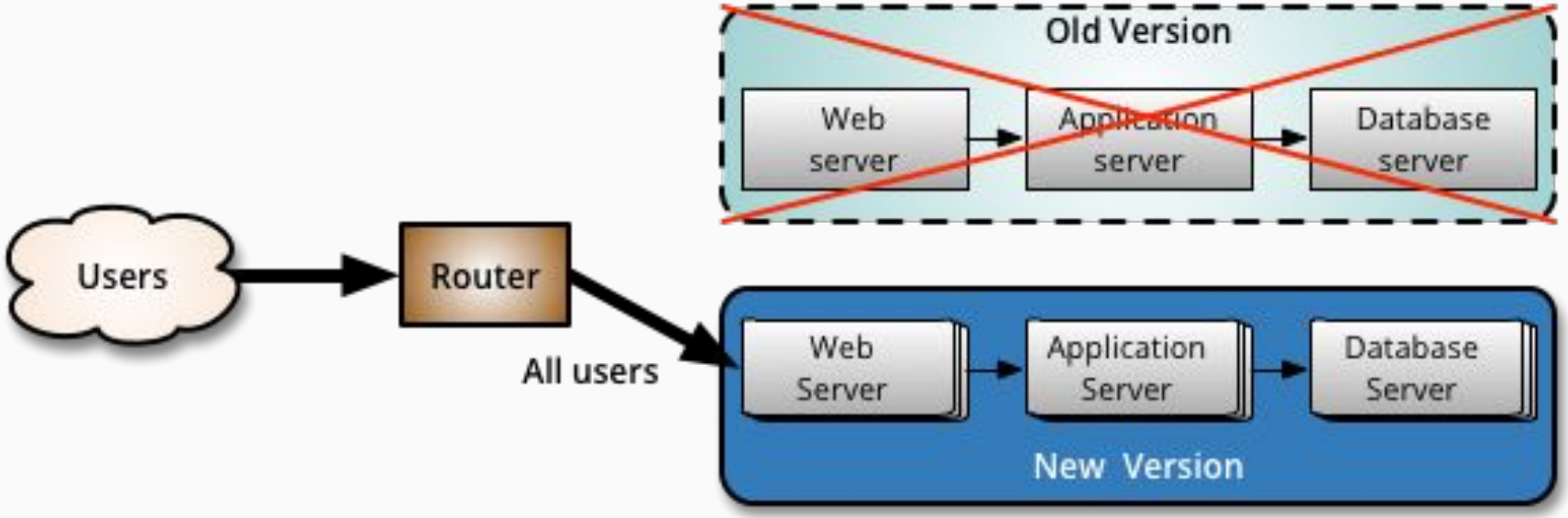


Photo Credits

Unsplash for awesome free photos!

https://unsplash.com

Thank you !

Contact