Azure Service Fabric

...a gentle introduction

Alessandro Melchiori

Solution architect @ Particular Software

alessandro [dot] melchiori [at] particular [dot] net

twitter: @amelchiori

blog: http://melkio.codiceplastico.com

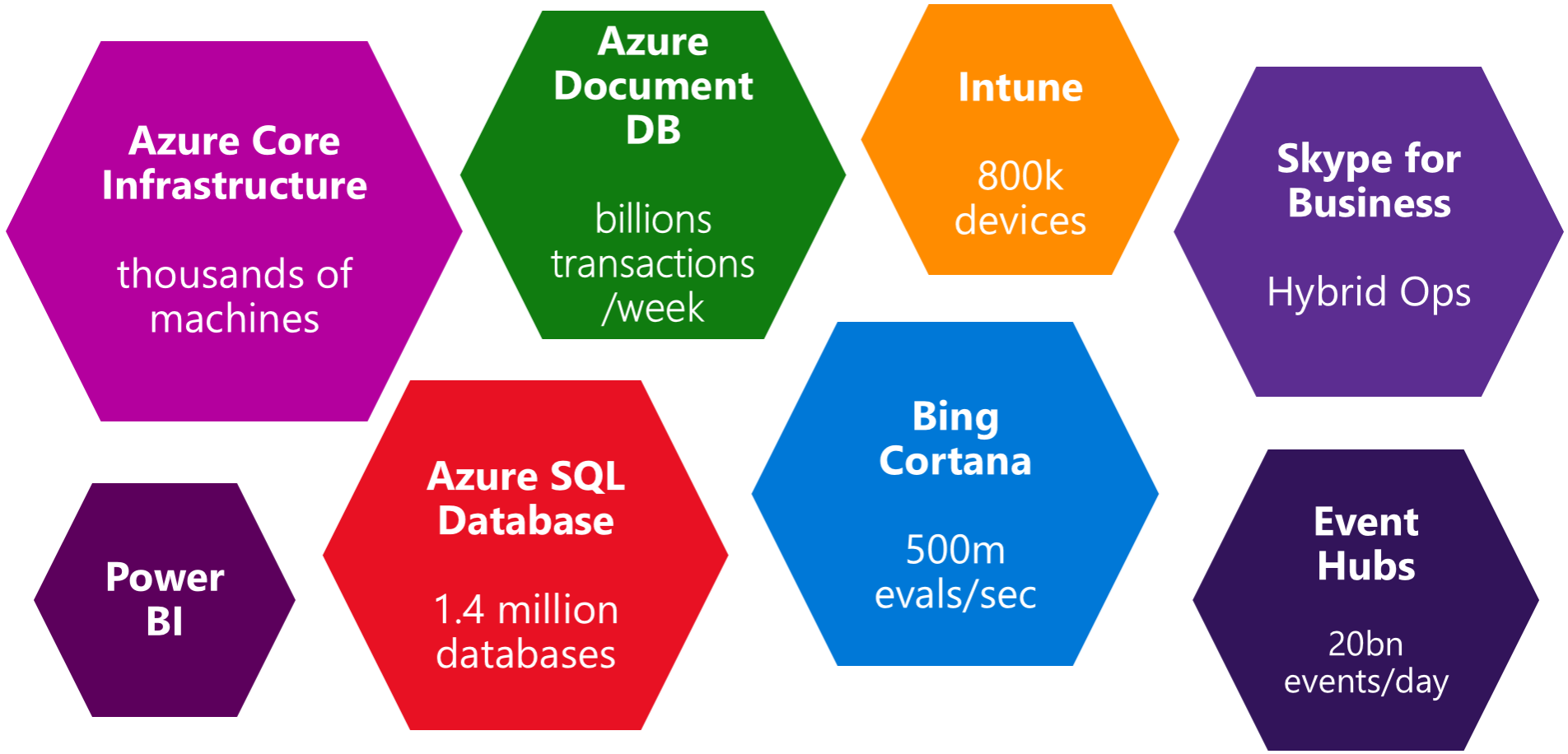
github: http://github.com/melkio

Agenda

- Why Service Fabric?
- Service Fabric overview
- What is a microservice?
  - Actor model
  - Stateless and Stateful
- Monitoring
  - Service Fabric Explorer
- Upgrade application

Why Service Fabric?

Where Service Fabric?

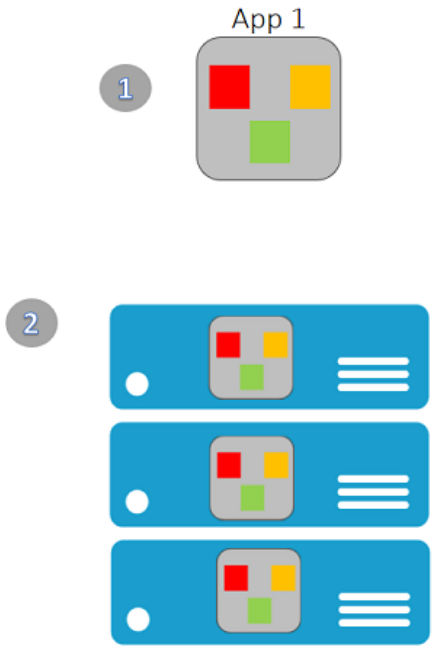


I MICROSERVICES...

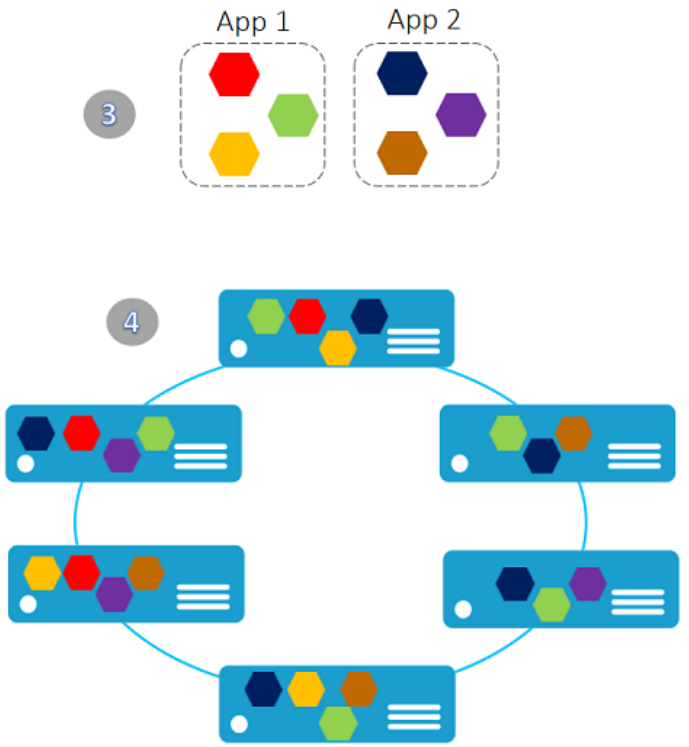
ΣΟΝΟ ΥΝΑ ΧΑΓΑΤΑ ΠΑΖΖΕΣΧΑ!!!

Why Service Fabric?

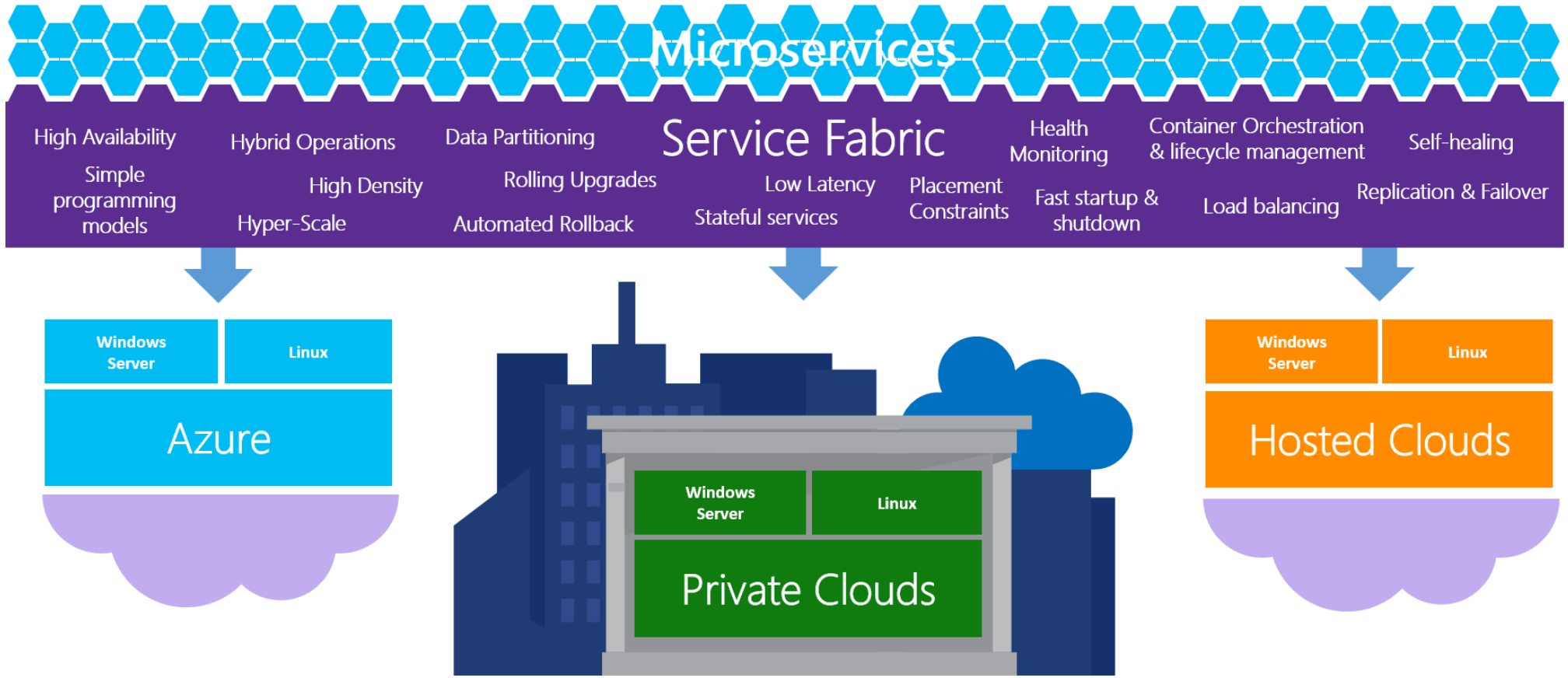
Monolithic application approach



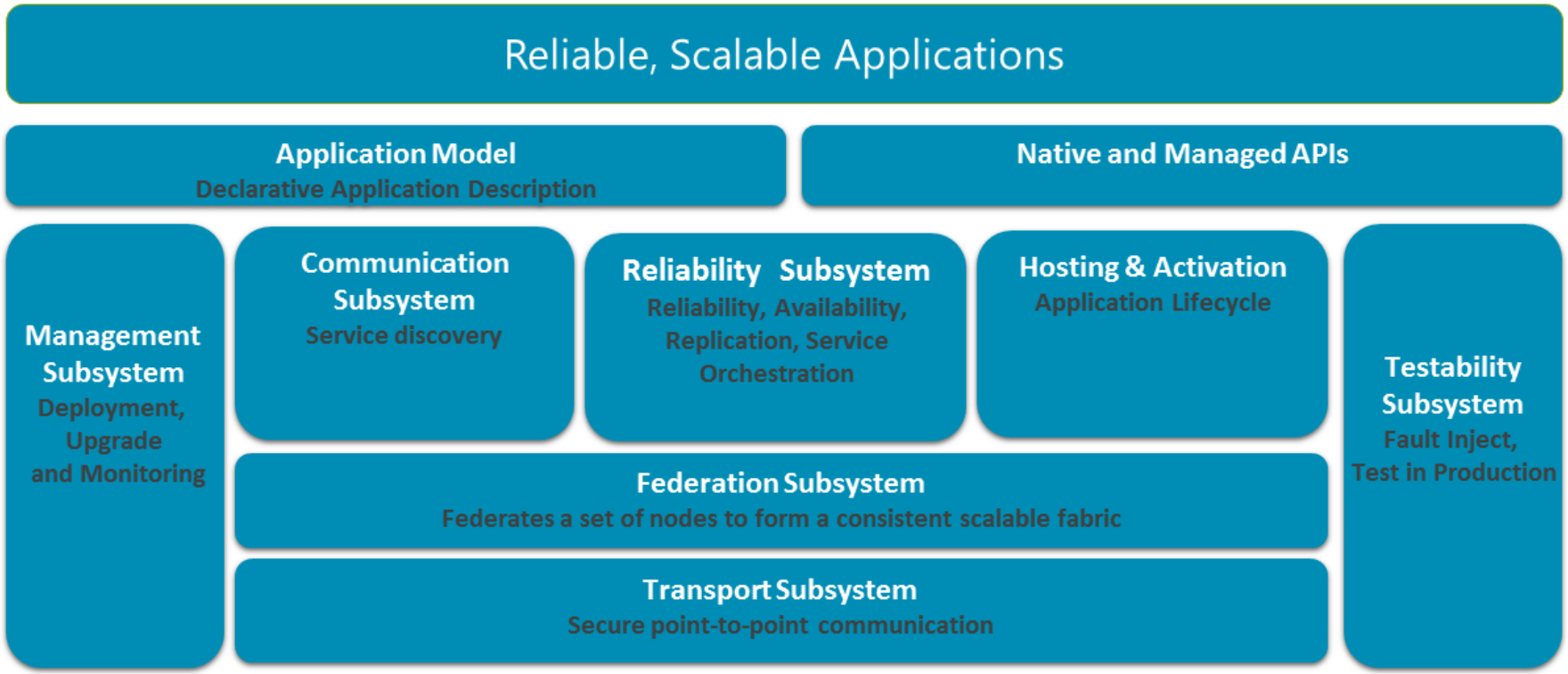
Microservices application approach



Why Service Fabric?

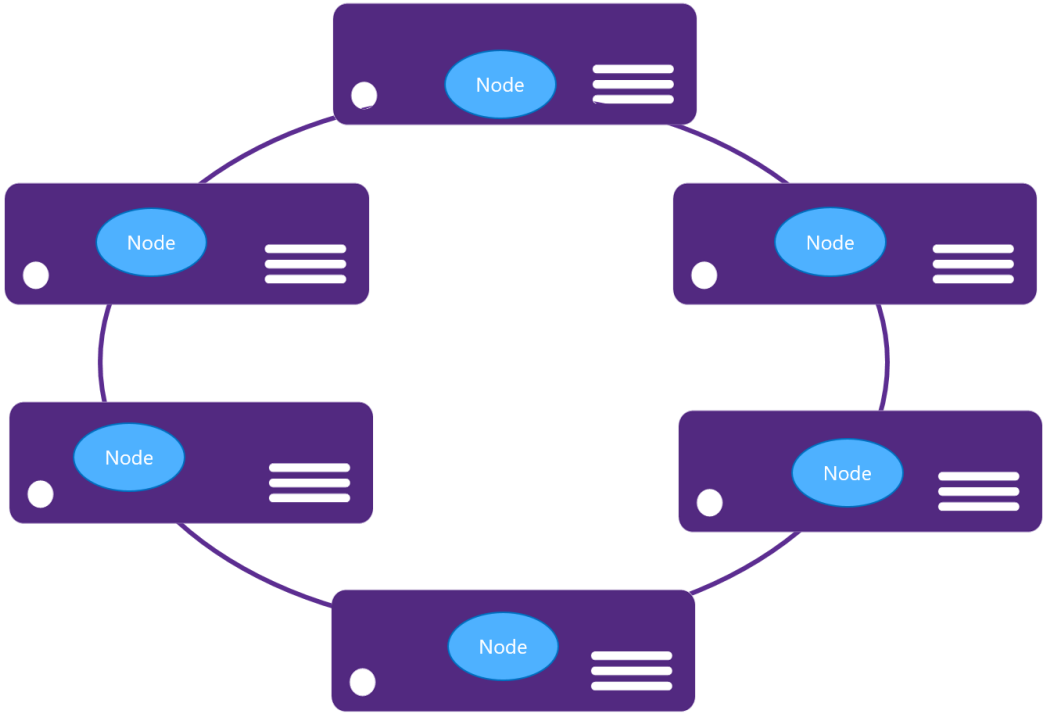


Service Fabric Architecture



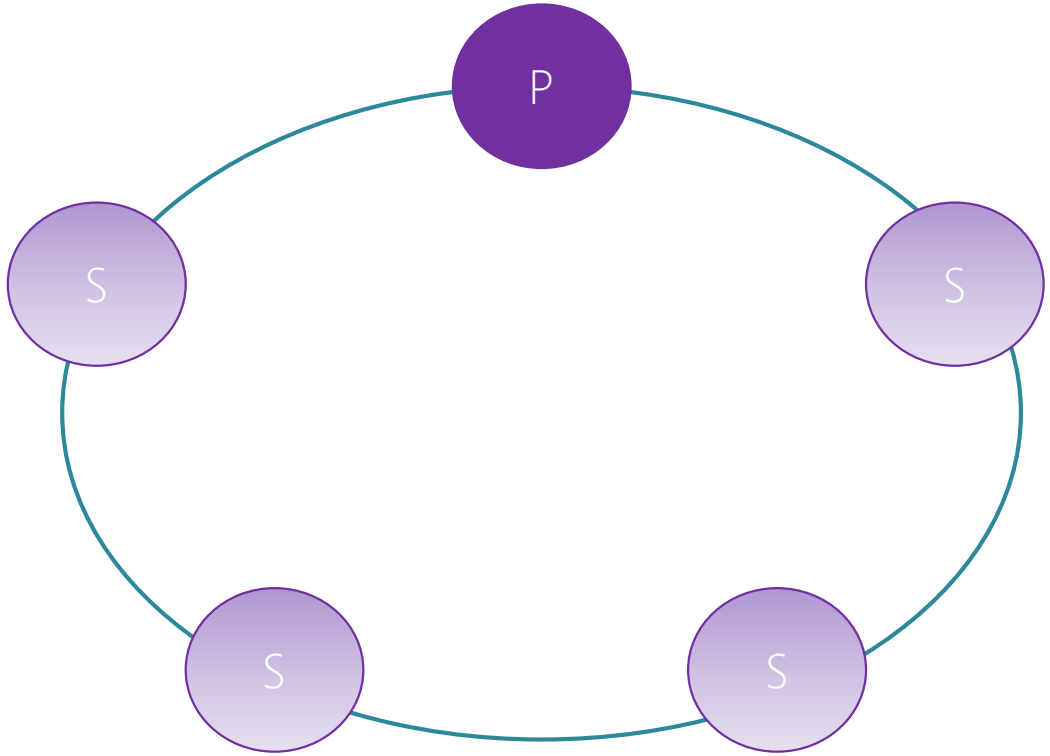
Service Fabric architecture: cluster

- Cluster is a federation of machines
- Cluster can scale to 1000s of machines



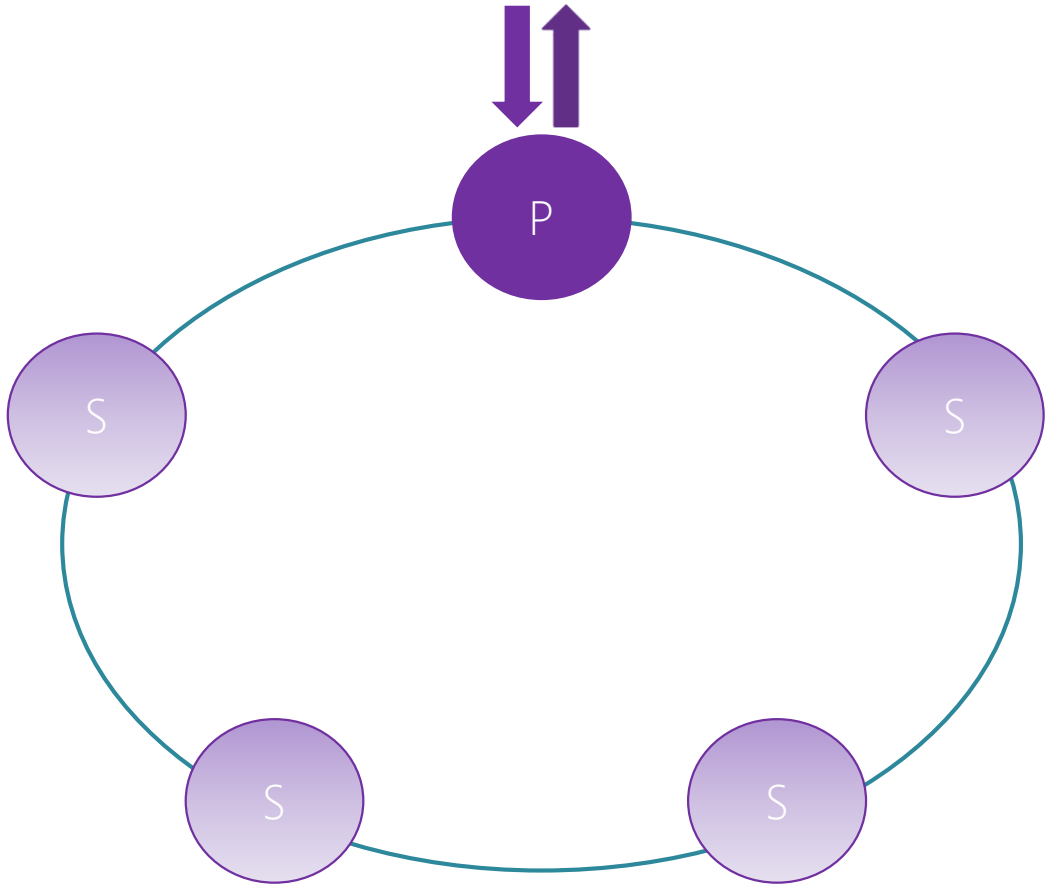
Service Fabric architecture: replication system

- Replica states
  - None
  - Idle secondary
  - Active secondary
  - Primary



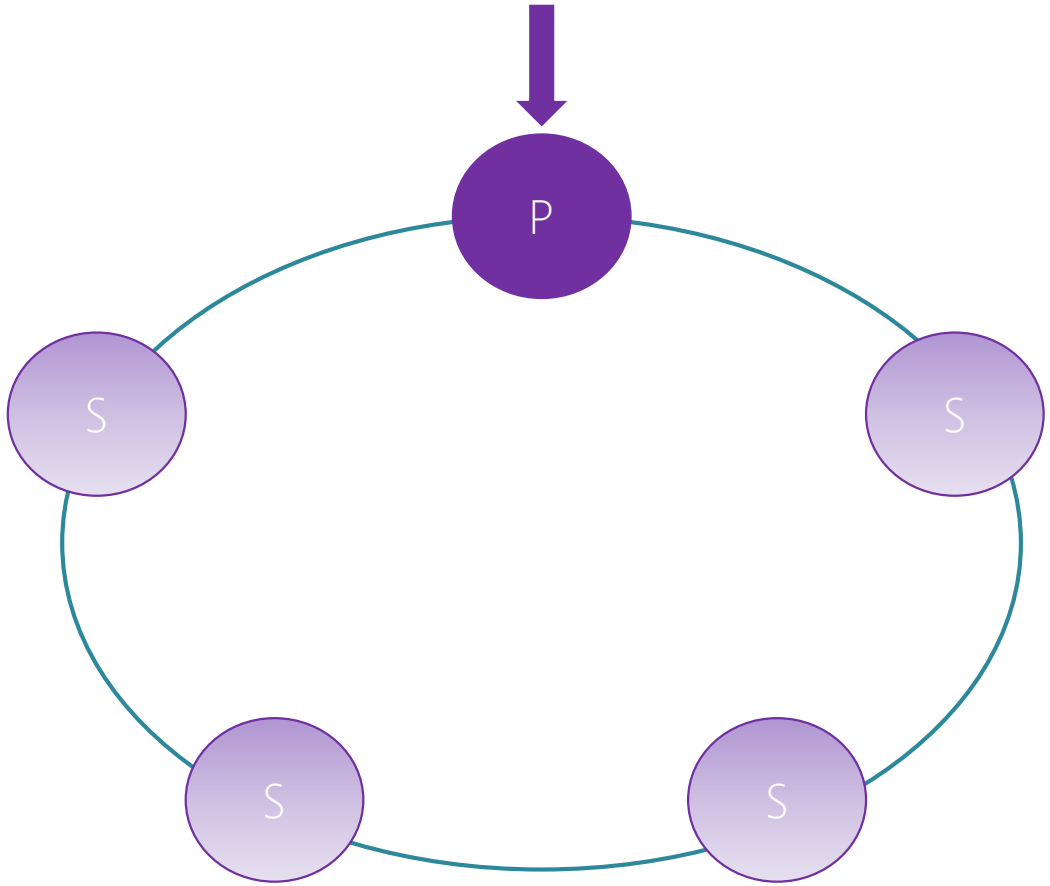
Service Fabric architecture: replication system

- Reads are completed at the primary



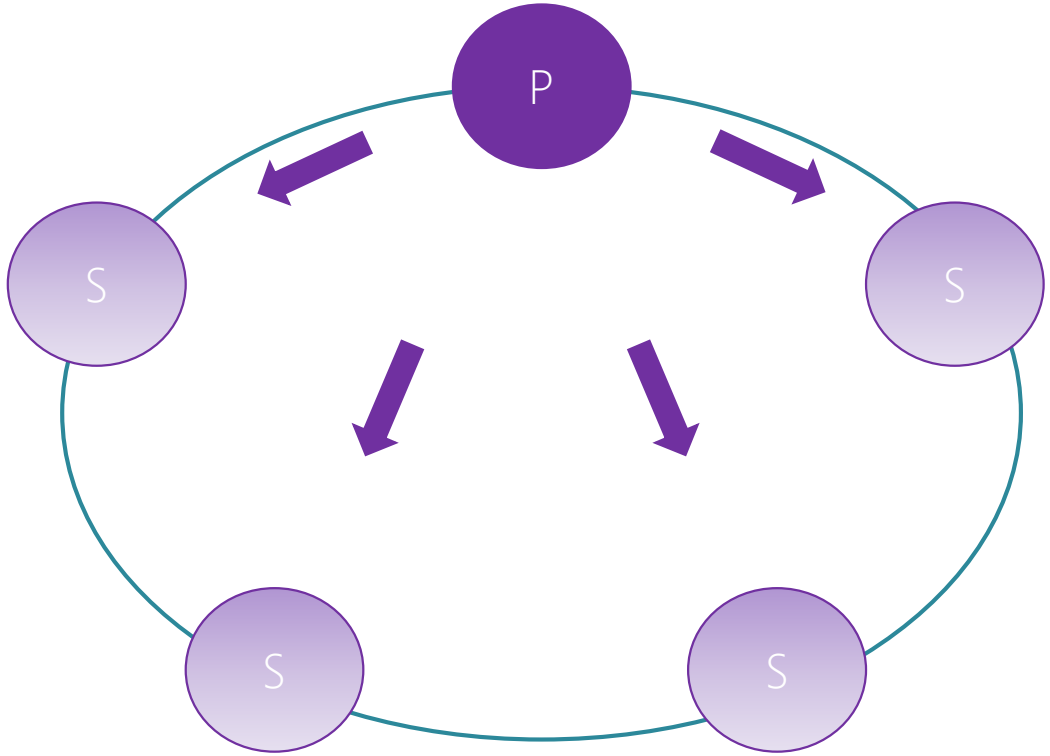
Service Fabric architecture: replication system

- Writes are replicated to the write quorum of secondaries



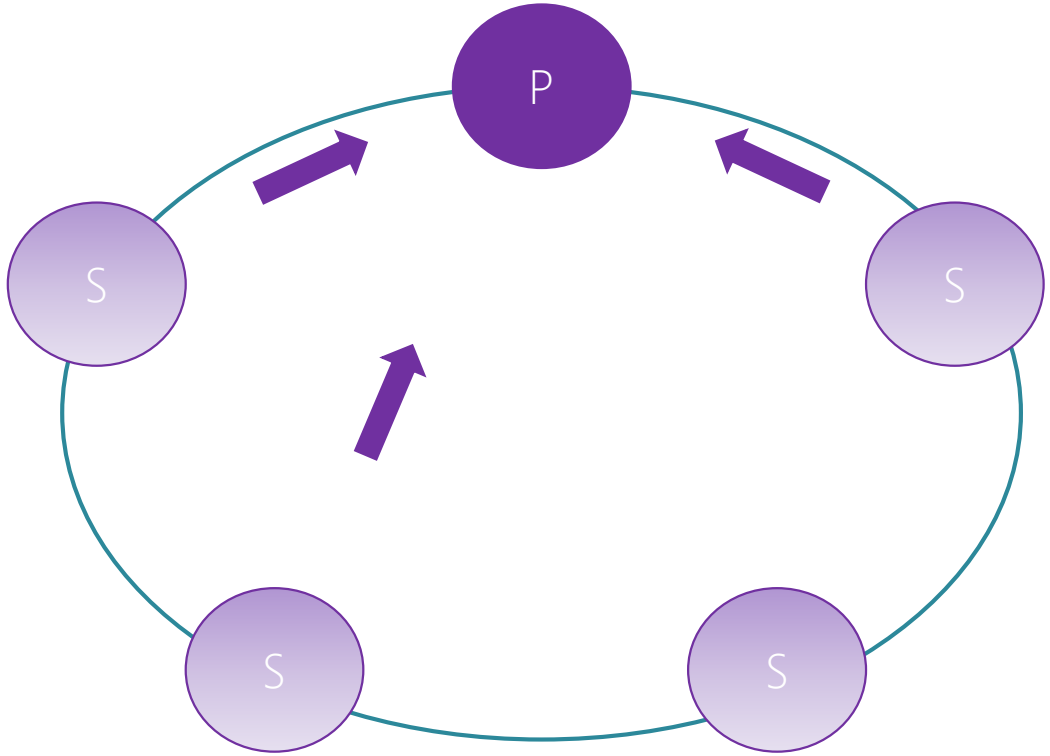
Service Fabric architecture: replication system

- Writes are replicated to the write quorum of secondaries



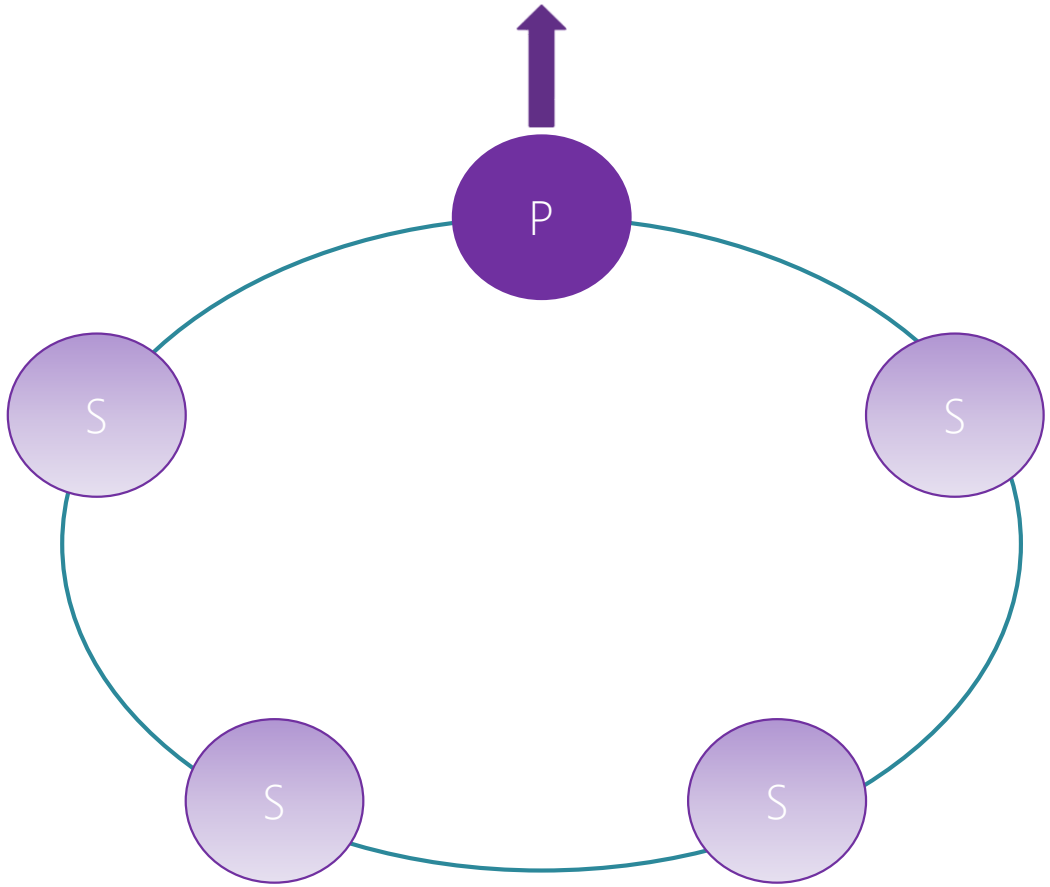
Service Fabric architecture: replication system

- Writes are replicated to the write quorum of secondaries



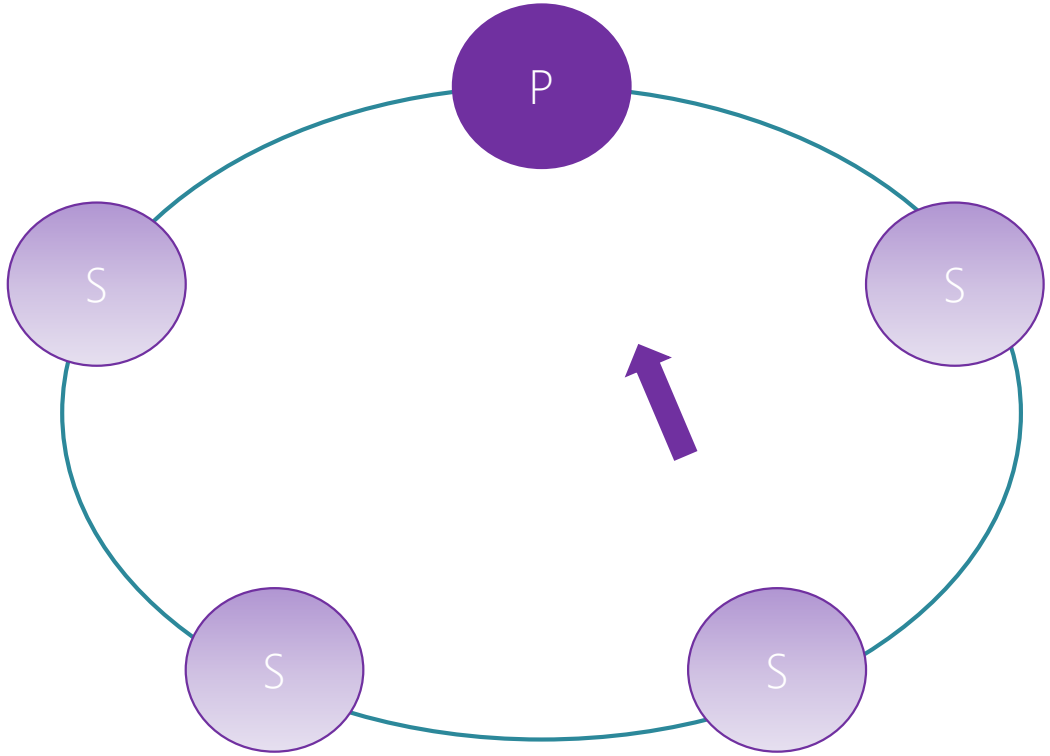
Service Fabric architecture: replication system

- Writes are replicated to the write quorum of secondaries



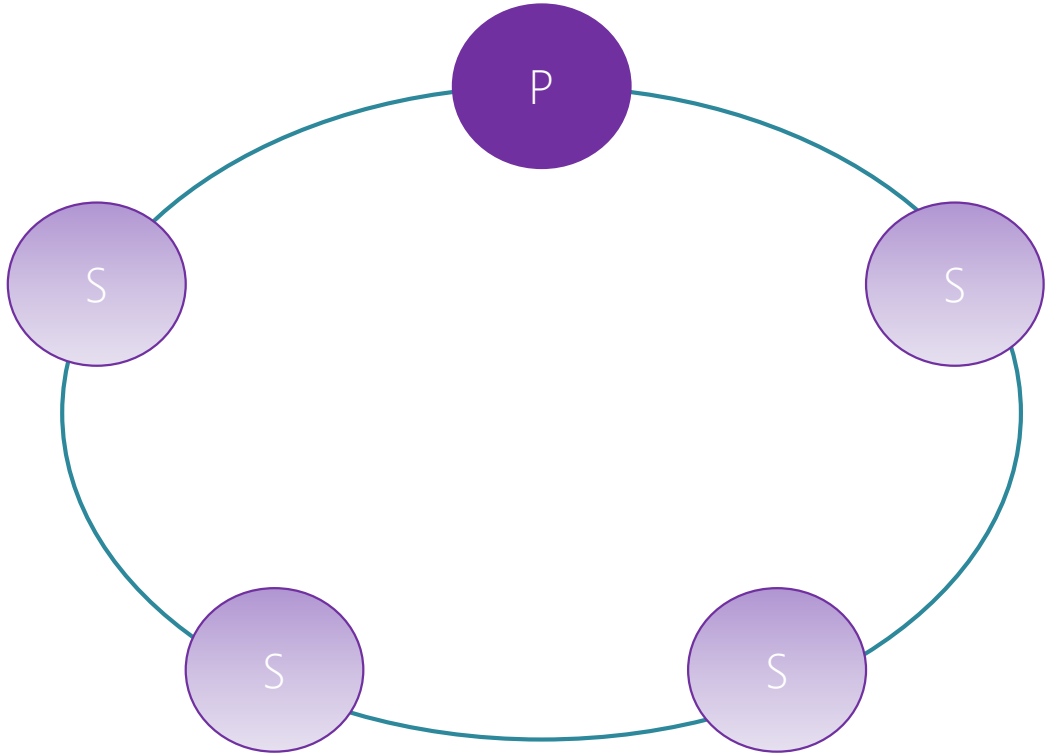
Service Fabric architecture: replication system

- Writes are replicated to the write quorum of secondaries



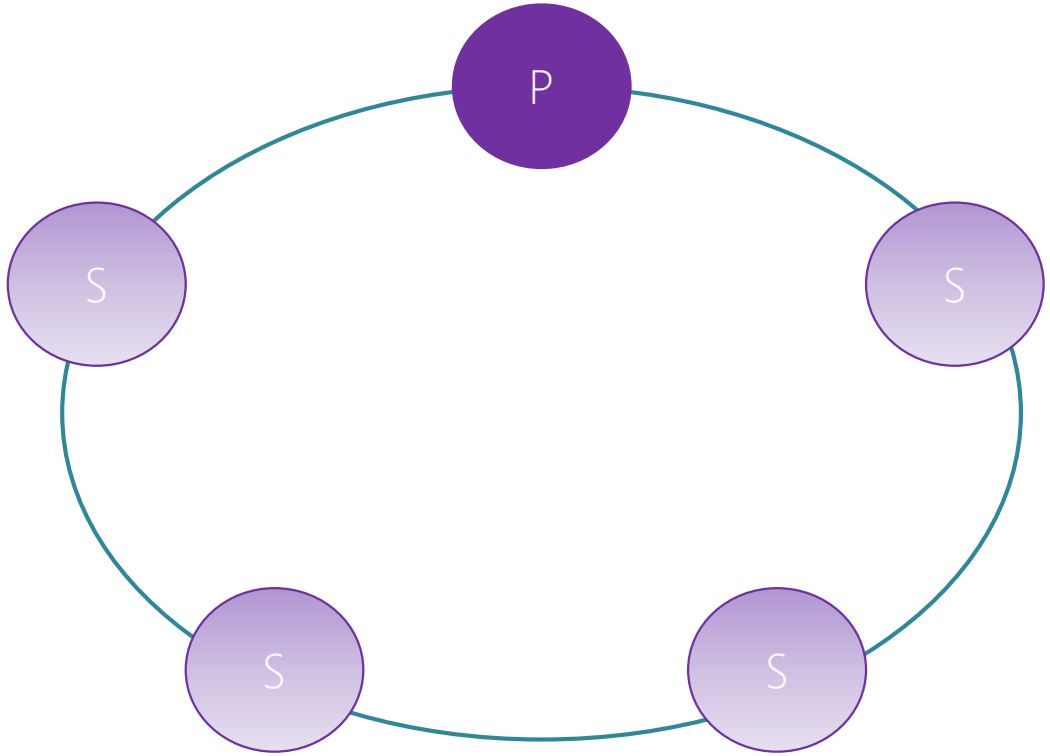
Service Fabric architecture: replication system

- Writes are replicated to the write quorum of secondaries



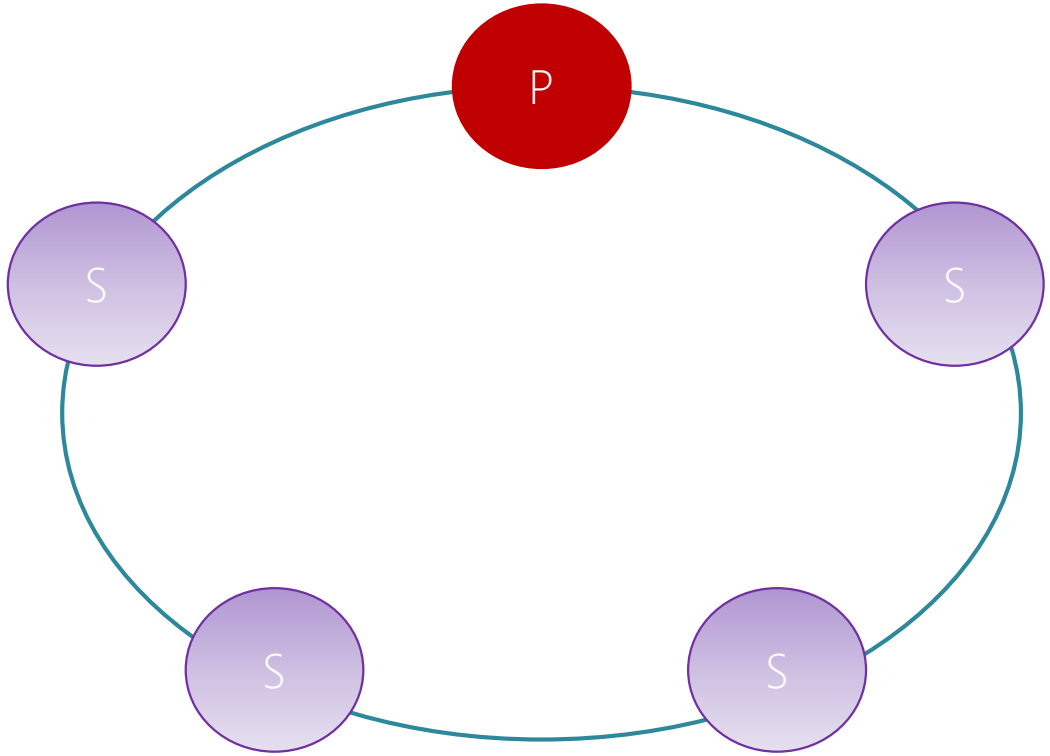
Service Fabric architecture: replication system reconfig

- Types of reconfiguration
  - Primary failover
  - Removing a failed secondary
  - Adding recovered replica
  - Building new secondary



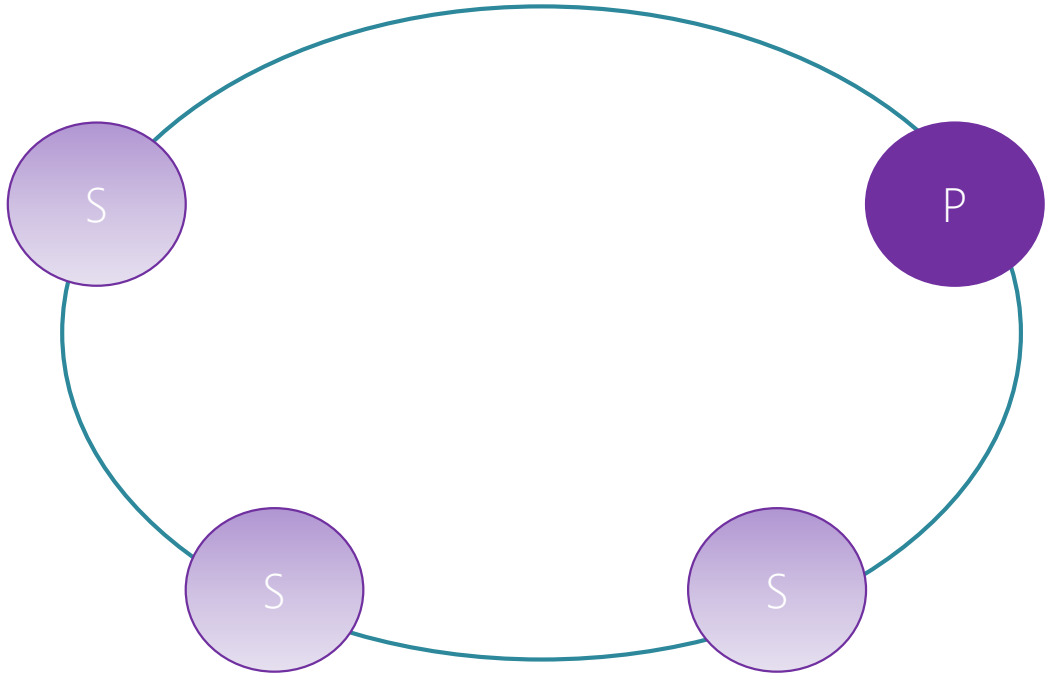
Service Fabric architecture: replication system reconfig

- Types of reconfiguration
  - Primary failover
  - Removing a failed secondary
  - Adding recovered replica
  - Building new secondary



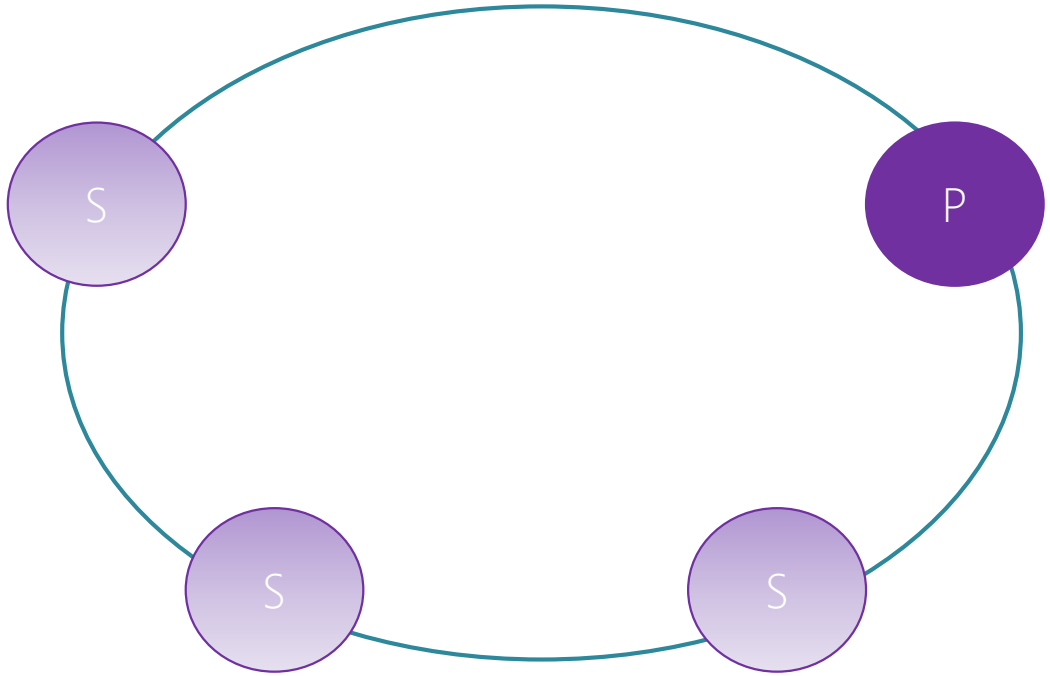
Service Fabric architecture: replication system reconfig

- Types of reconfiguration
  - Primary failover
  - Removing a failed secondary
  - Adding recovered replica
  - Building new secondary



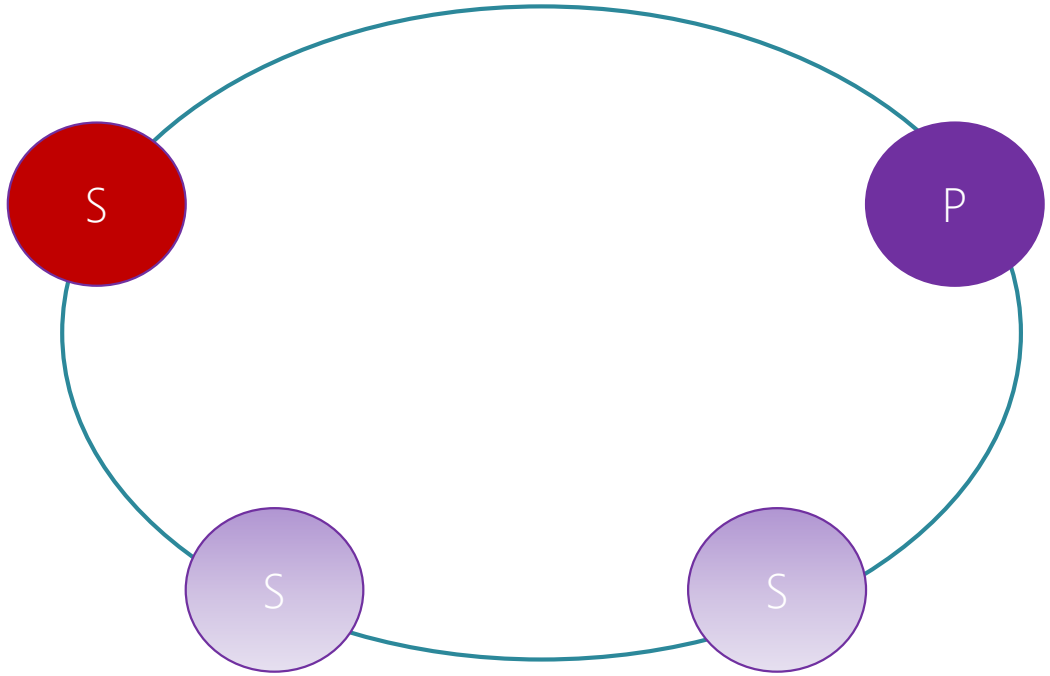
Service Fabric architecture: replication system reconfig

- Types of reconfiguration
  - Primary failover
  - **Removing a failed secondary**
  - Adding recovered replica
  - Building new secondary



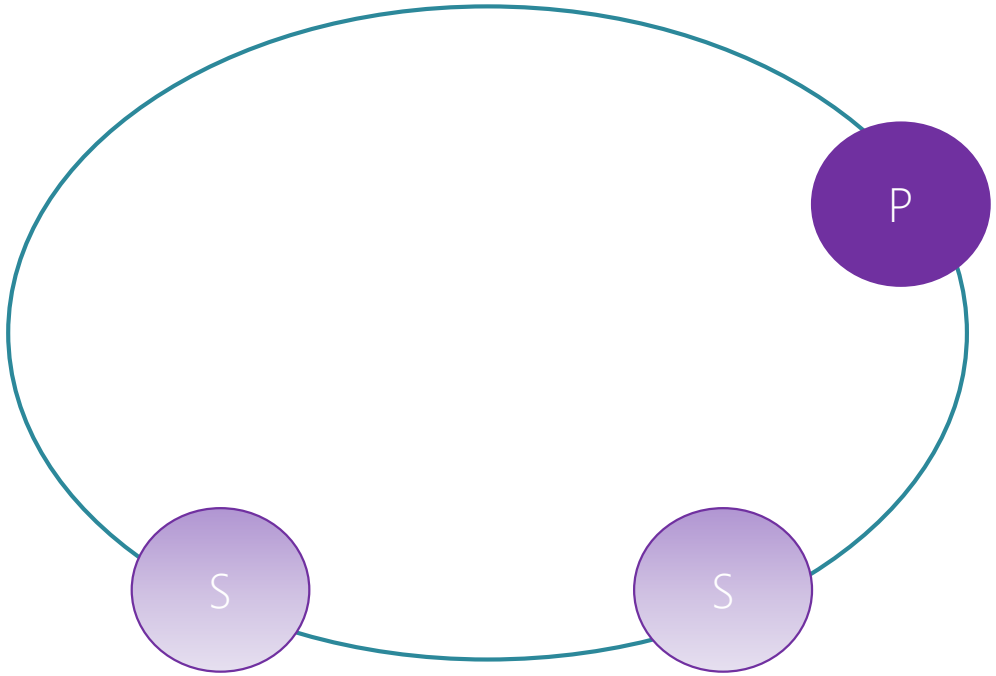
Service Fabric architecture: replication system reconfig

- Types of reconfiguration
  - Primary failover
  - **Removing a failed secondary**
  - Adding recovered replica
  - Building new secondary



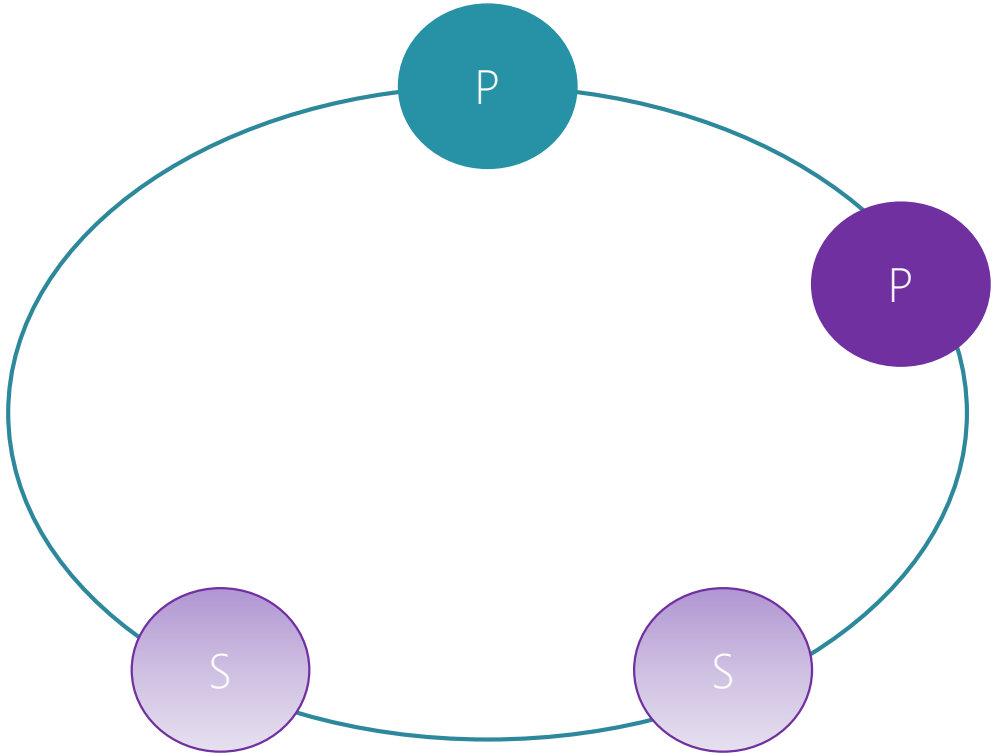
Service Fabric architecture: replication system reconfig

- Types of reconfiguration
  - Primary failover
  - **Removing a failed secondary**
  - Adding recovered replica
  - Building new secondary



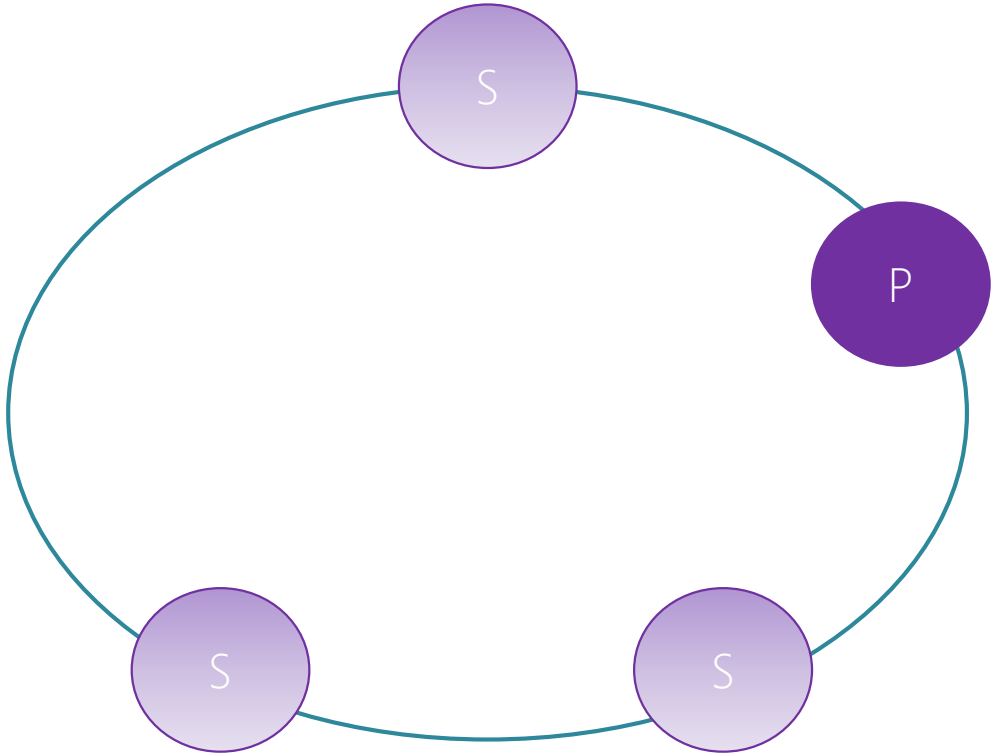
Service Fabric architecture: replication system reconfig

- Types of reconfiguration
  - Primary failover
  - Removing a failed secondary
  - Adding recovered replica
  - Building new secondary



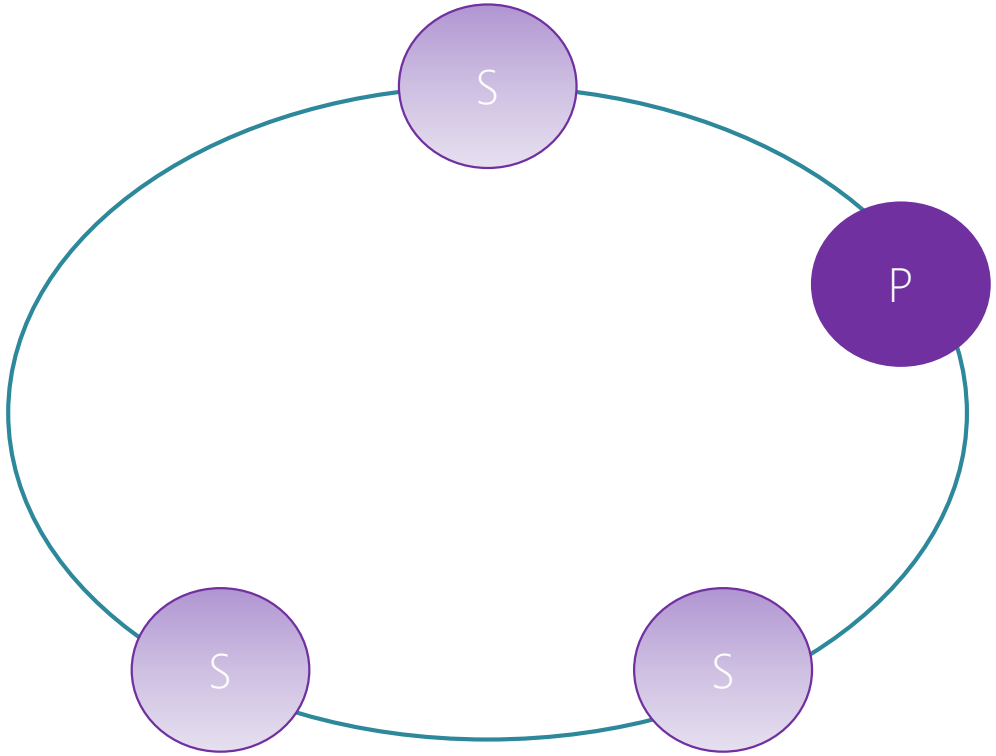
Service Fabric architecture: replication system reconfig

- Types of reconfiguration
  - Primary failover
  - Removing a failed secondary
  - Adding recovered replica
  - Building new secondary



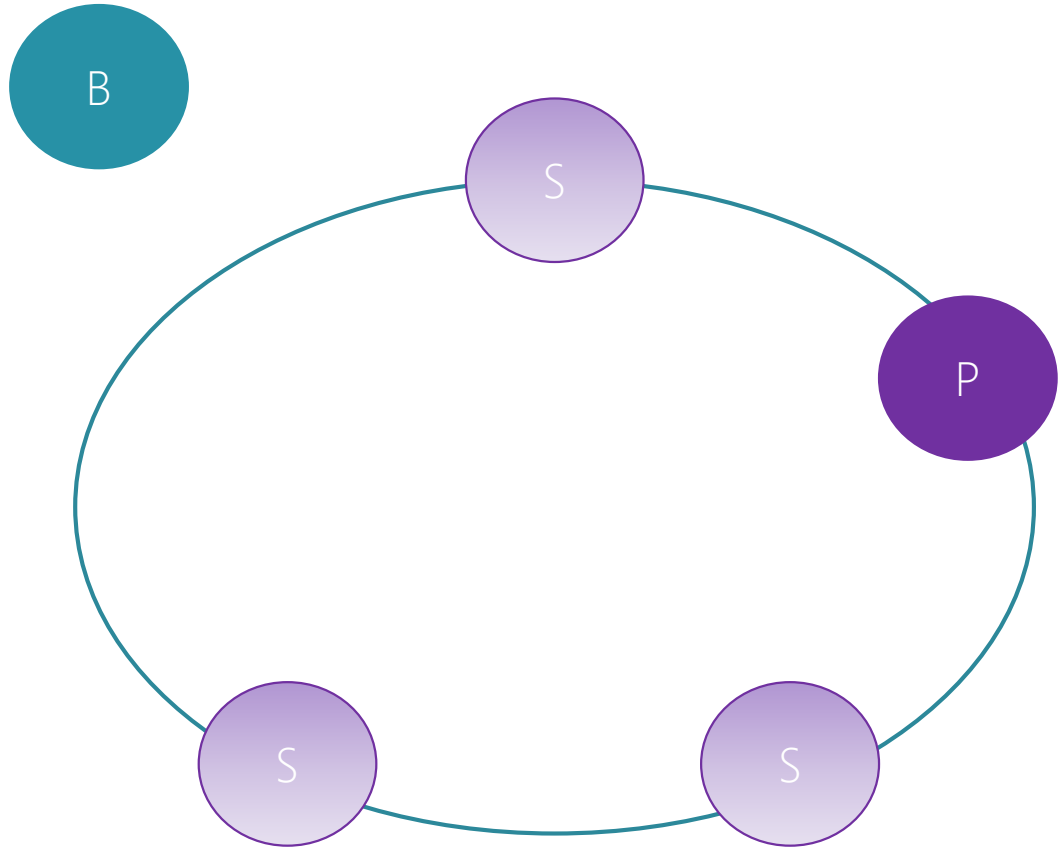
Service Fabric architecture: replication system reconfig

- Types of reconfiguration
  - Primary failover
  - Removing a failed secondary
  - Adding recovered replica
  - **Building new secondary**



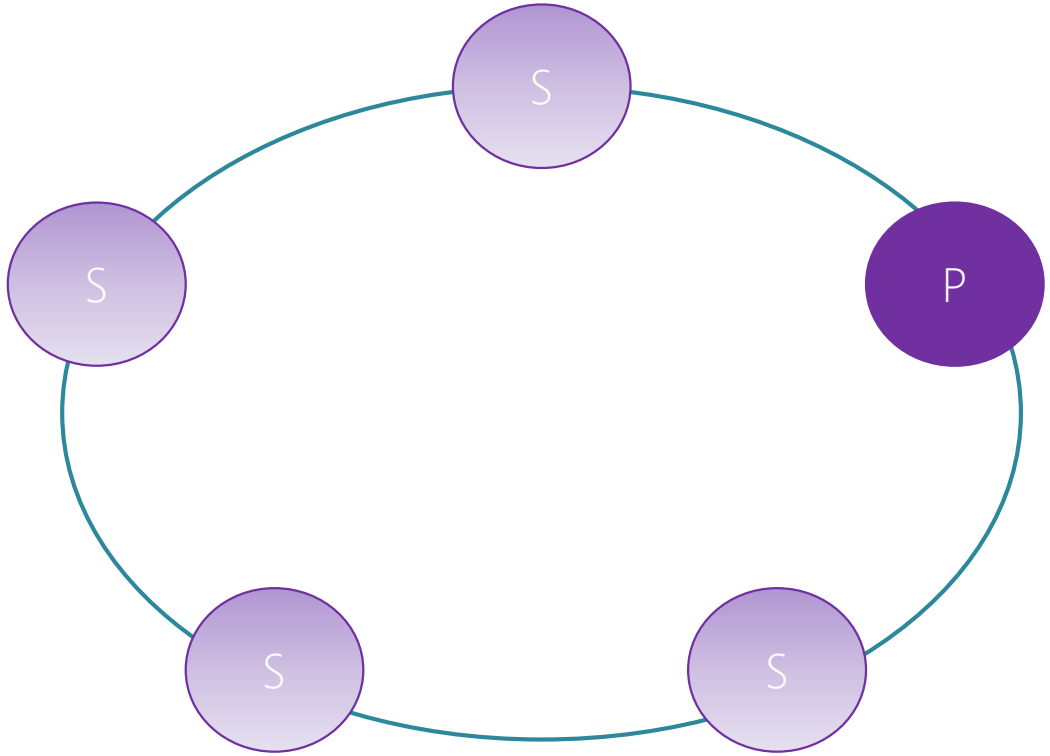
Service Fabric architecture: replication system reconfig

- Types of reconfiguration
  - Primary failover
  - Removing a failed secondary
  - Adding recovered replica
  - **Building new secondary**



Service Fabric architecture: replication system reconfig

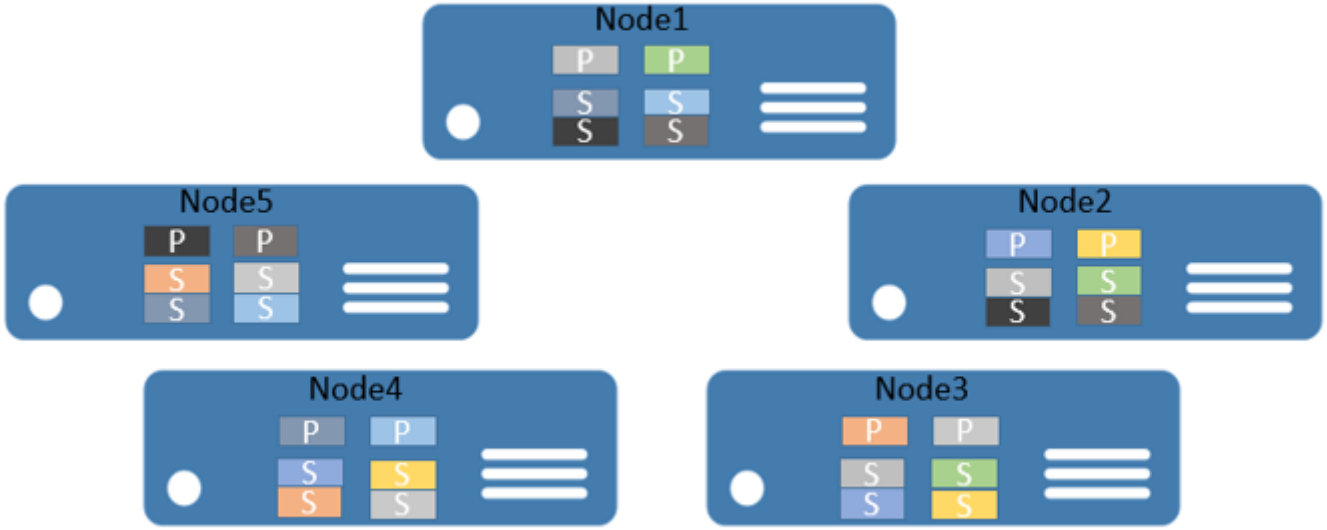
- Types of reconfiguration
  - Primary failover
  - Removing a failed secondary
  - Adding recovered replica
  - **Building new secondary**



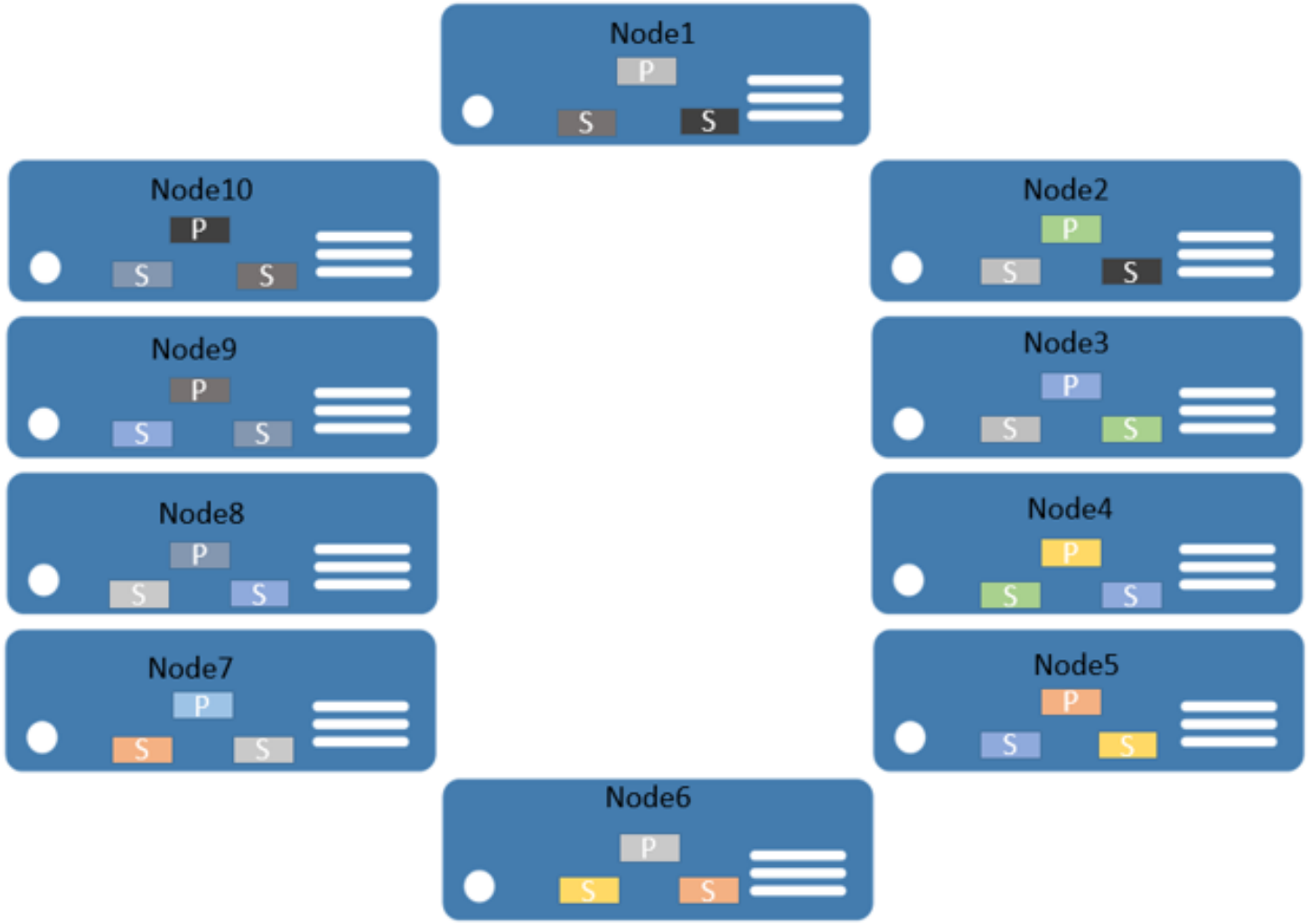
Service Fabric architecture: partitioning

- Allow data/computation to be spread across nodes
- A partition must fit in 1 node / 1 node can hold multiple partitions
- Cross-partitions operations requires network hops and different transactions
- SF balances partitions across nodes

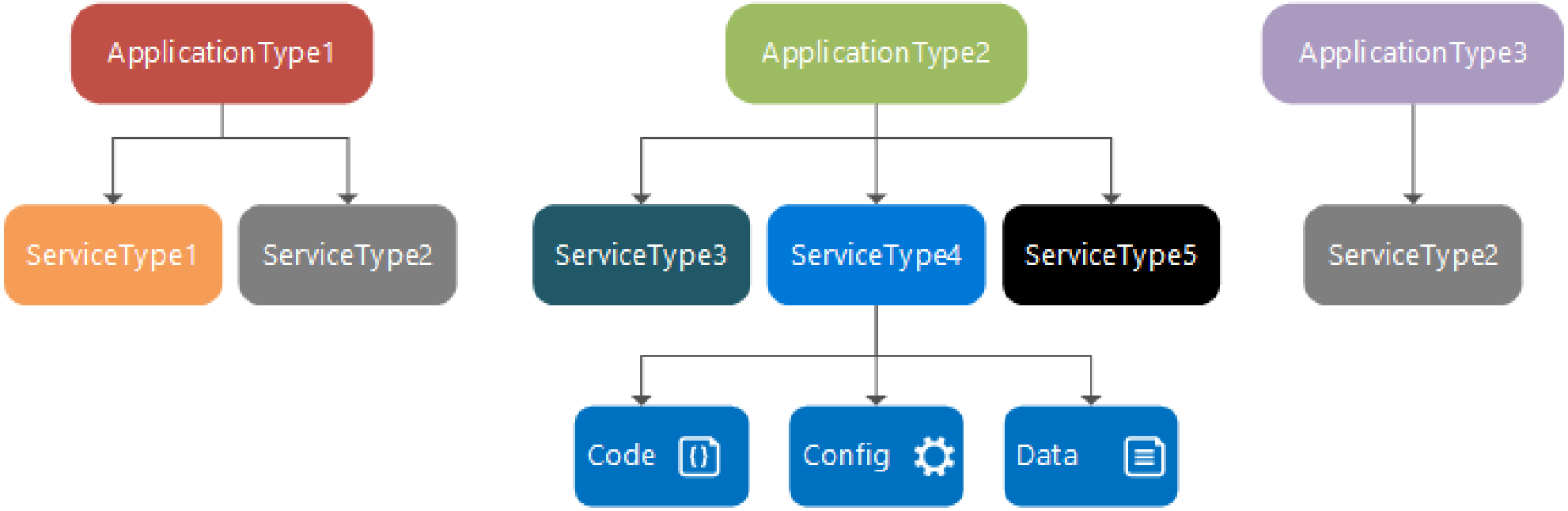
Service Fabric architecture: partitioning



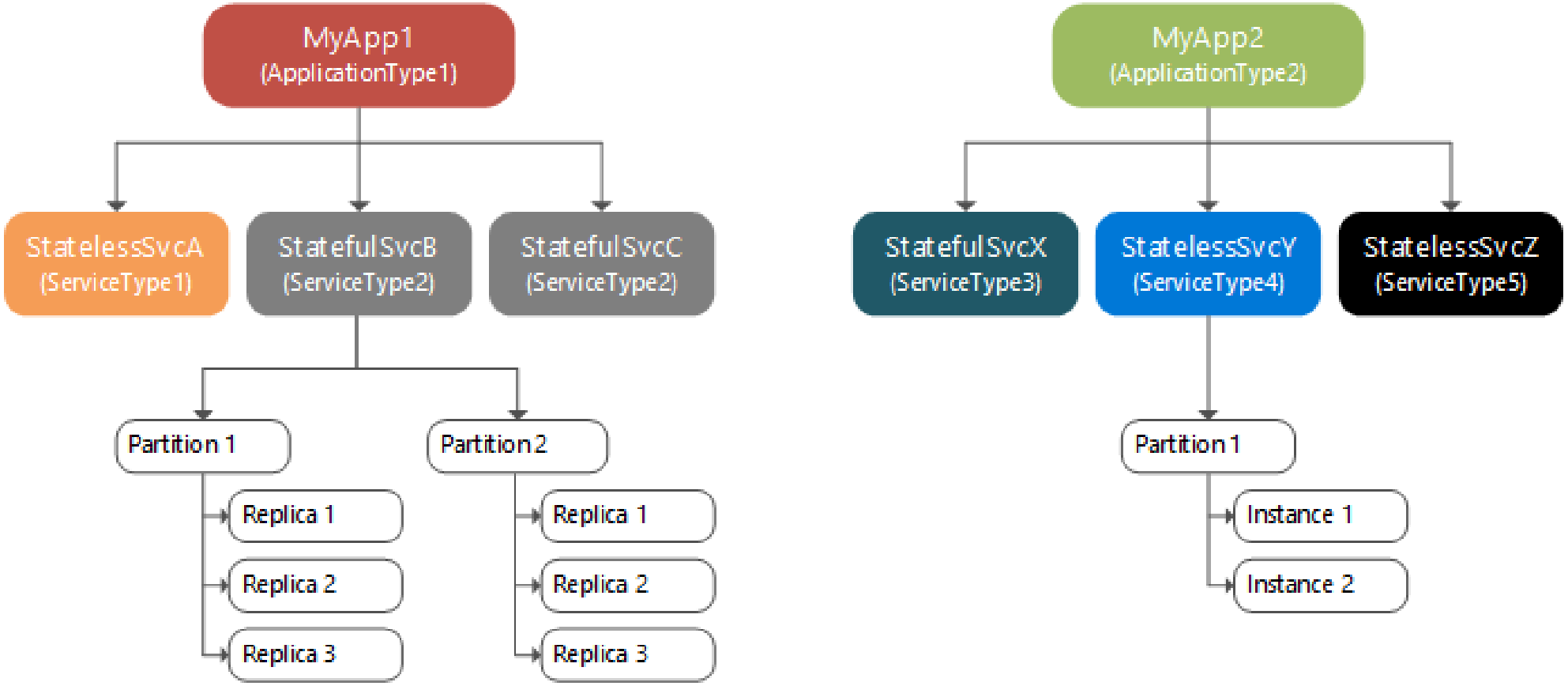
Service Fabric architecture: partitioning



Service Fabric application model



Service Fabric application model



DEMO

run your local cluster

What is a microservice?

- Encapsulate a business scenario
- Can be written in any programming language
- Consist of code and (optionally) state that is independently versioned, deployed and scaled
- Has a unique name, used to resolve its location
- Remains consistent and available in the presence of failure
- Interacts with other microservices

Actor model

The actor model in computer science is a mathematical model of concurrent computation that treats "actors" as the universal primitives of concurrent computation: in response to a message that it receives, an actor can make local decisions, create more actors, send more messages, and determine how to respond to the next message received

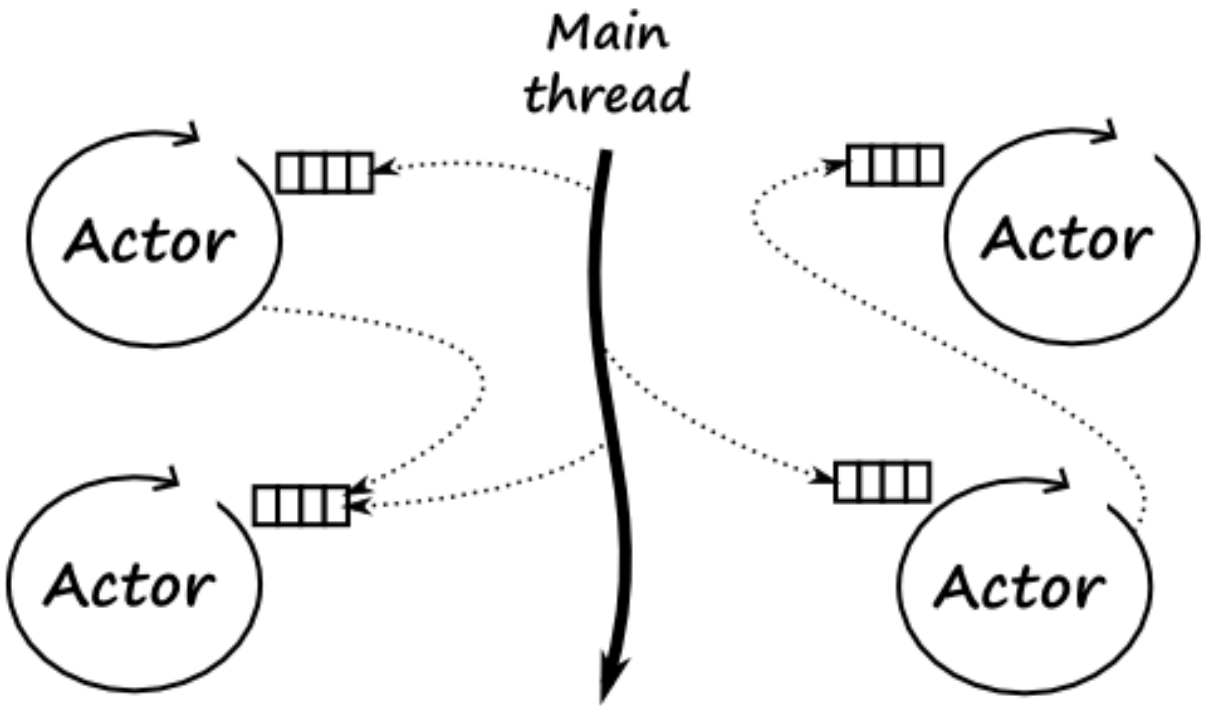
https://en.wikipedia.org/wiki/Actor_model

Actor model

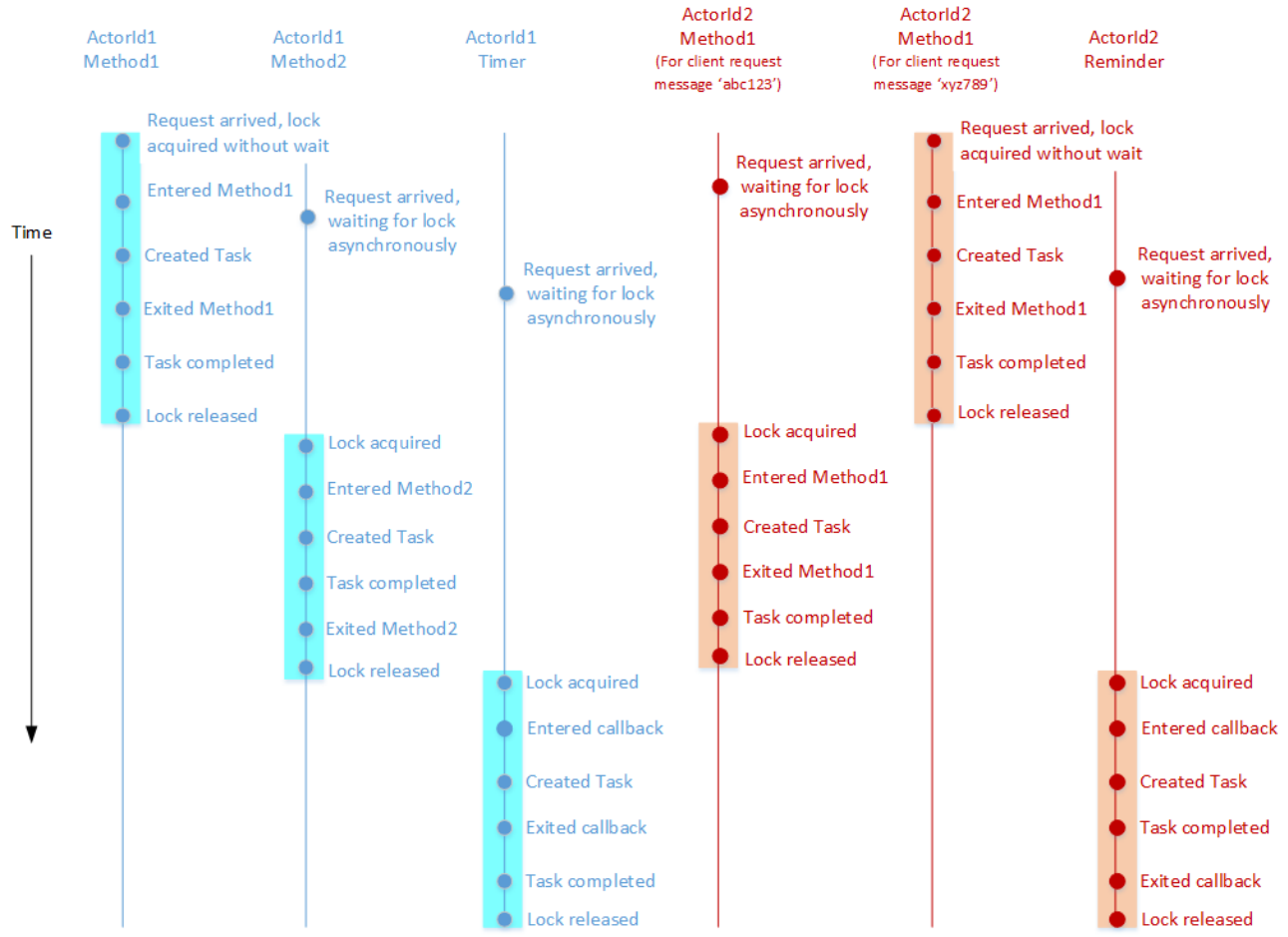
The actor model in computer science is a mathematical model of concurrent computation that treats "actors" as the universal primitives of concurrent computation: in response to a message that it receives, an actor can make local decisions, create more actors, send more messages, and determine how to respond to the next message received

https://en.wikipedia.org/wiki/Actor_model

Actor model



Actor model


ActorId1
Method1
ActorId1
Method2
ActorId1
Timer
ActorId2
Method1
(For client request
message 'abc123')
ActorId2
Method1
(For client request
message 'xyz789')
ActorId2
Reminder
Time
Request arrived, lock acquired without wait
Entered Method1
Created Task
Exited Method1
Task completed
Lock released
Request arrived, waiting for lock asynchronously
Lock acquired
Entered Method2
Created Task
Task completed
Exited Method2
Lock released
Request arrived, waiting for lock asynchronously
Lock acquired
Entered callback
Created Task
Exited callback
Task completed
Lock released
Request arrived, waiting for lock asynchronously
Lock acquired
Entered Method1
Created Task
Exited Method1
Task completed
Lock released
Request arrived, lock acquired without wait
Entered Method1
Created Task
Exited Method1
Task completed
Lock released
Request arrived, waiting for lock asynchronously
Lock acquired
Entered callback
Created Task
Task completed
Exited callback
Lock released

Service Fabric Reliable Actors API

- Actors are isolated, single-threaded components that encapsulate both state and behavior
- Each such actor is uniquely identified by an actor ID
- Actors interact with rest of the system, including other actors, by passing asynchronous messages using a request-response pattern

Stateless actor: definition

```
public interface ICalculatorActor : IActor
{
    Task Increment();
    Task<Int32> GetValue();
}
```

Stateless actor: definition

```
public interface ICalculatorActor : IActor
{
    Task Increment();
    Task<Int32> GetValue();
}
```

Stateless actor: definition

```
public interface ICalculatorActor : IActor
{
    Task Increment();
    Task<Int32> GetValue();
}
```

Stateless actor: definition

```
public interface ICalculatorActor : IActor
{
    Task Increment();
    Task<Int32> GetValue();
}

public class CalculatorActor : StatelessActor, ICalculatorActor
{
    ...
}
```

Stateless actor: definition

```
public interface ICalculatorActor : IActor
{
    Task Increment();
    Task<Int32> GetValue();
}

public class CalculatorActor : StatelessActor, ICalculatorActor
{
    ...
}
```

Stateful actor: definition

```
public interface ICalculatorActor : IActor
{
    Task Increment();
    Task<Int32> GetValue();
}
```

Stateful actor: definition

```
public interface ICalculatorActor : IActor
{
    Task Increment();
    Task<Int32> GetValue();
}

public class CalculatorActor : StatefulActor<ActorState>,
                               ICalculatorActor
{
    ...
}
```

Stateless actor: definition

```
public interface ICalculatorActor : IActor
{
    Task Increment();
    Task<Int32> GetValue();
}

public class CalculatorActor : StatefulActor<ActorState>,
                               ICalculatorActor
{
    ...
}
```

Actor communication: the actor proxy

```
var actorId = ActorId.NewId();

var applicationName = "fabric:/CalculatorActorApp";

var actor = ActorProxy.Create<ICalculatorActor>(actorId,
applicationName)
```

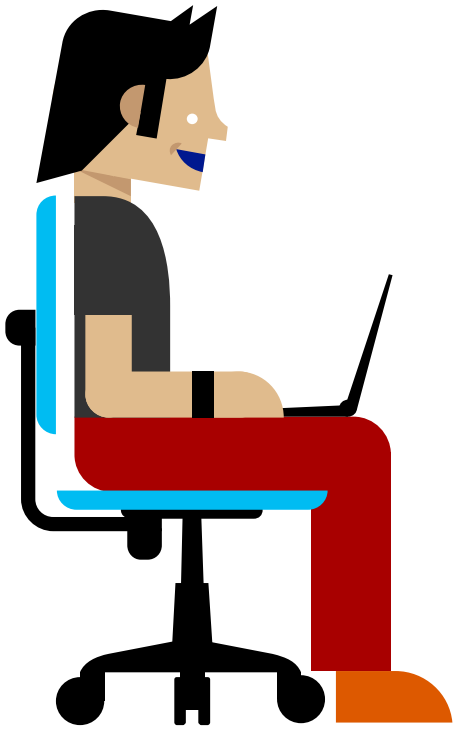
Actor communication: the actor proxy

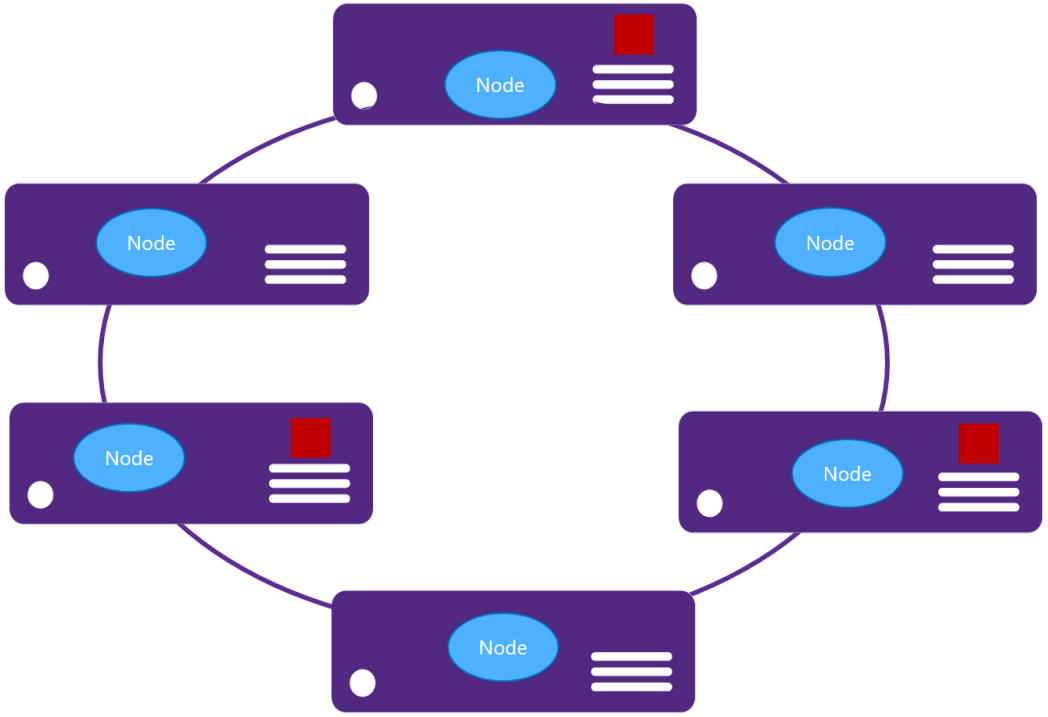
```
var actorId = ActorId.NewId();

var applicationName = "fabric:/CalculatorActorApp";

var actor = ActorProxy.Create<ICalculatorActor>(actorId,
applicationName)
```

DEMO

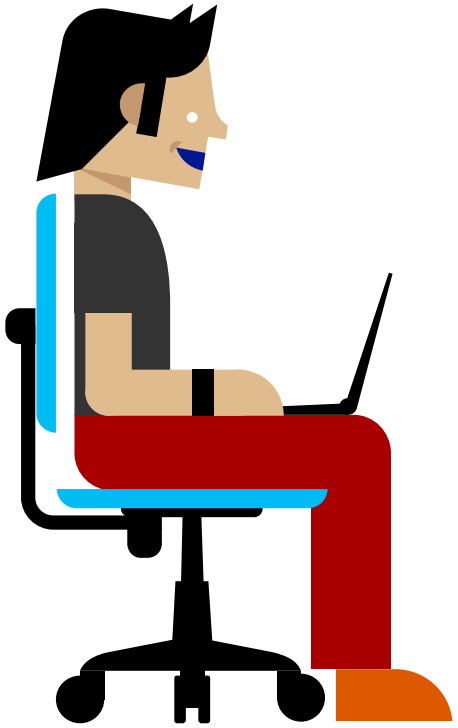
stateless actor vs stateful actor

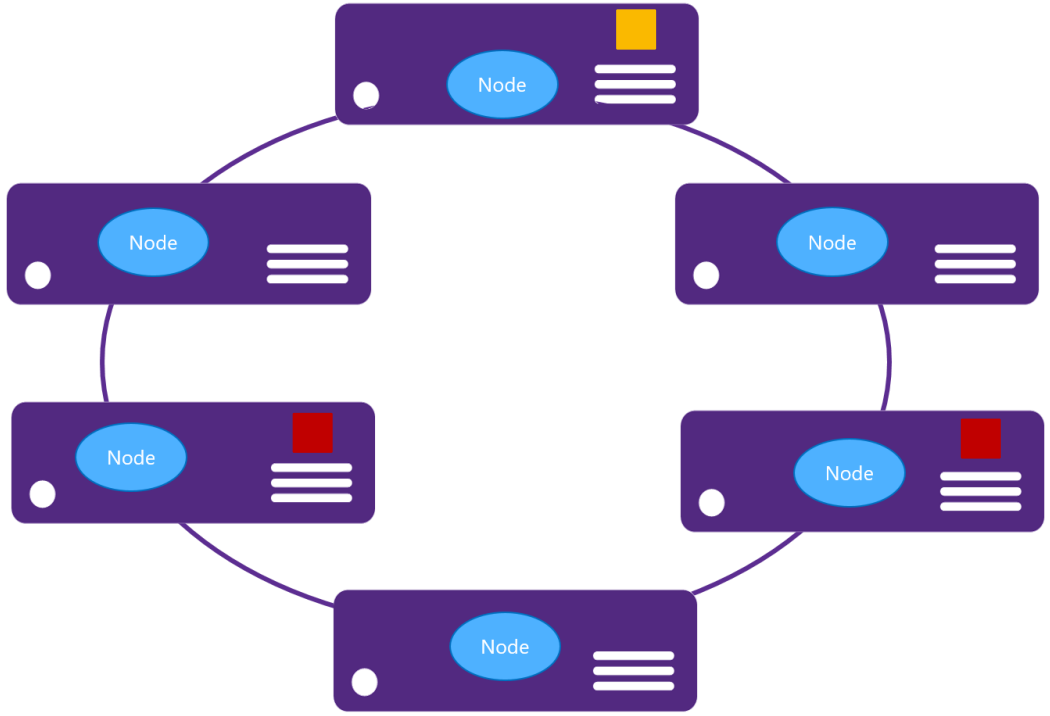
Upgrade application (with zero downtime)



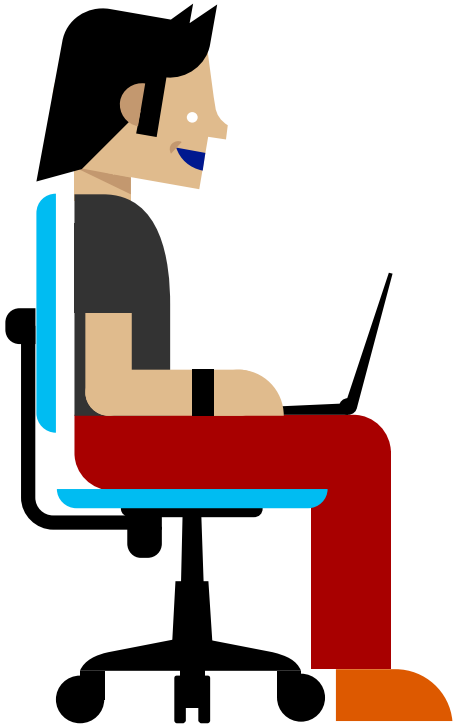


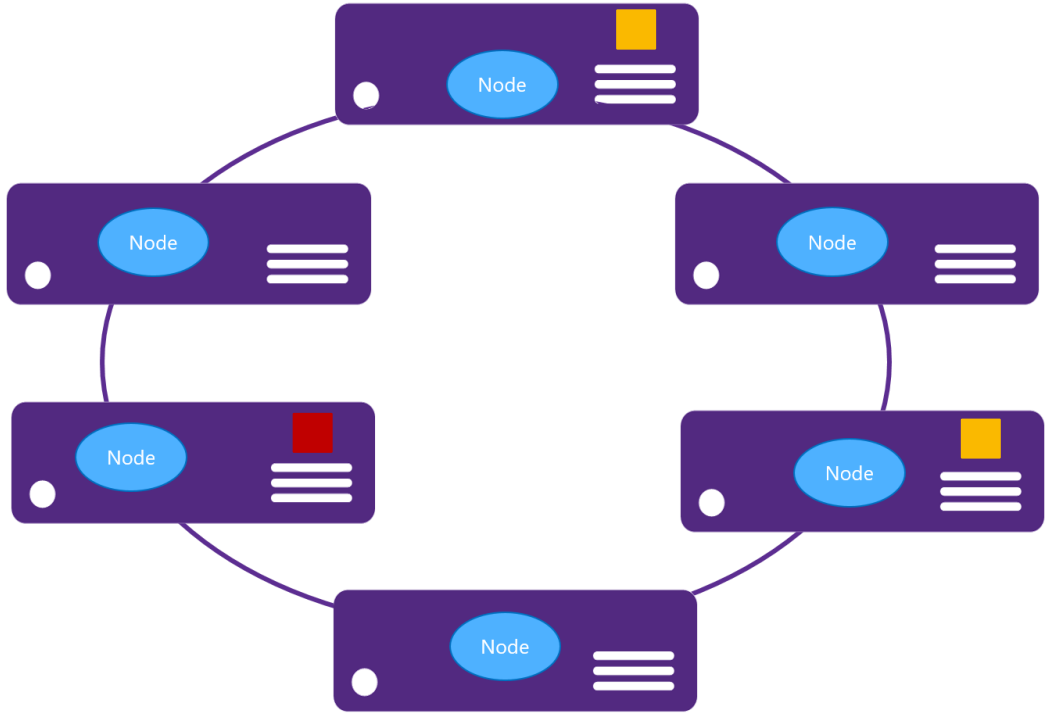
Upgrade application (with zero downtime)



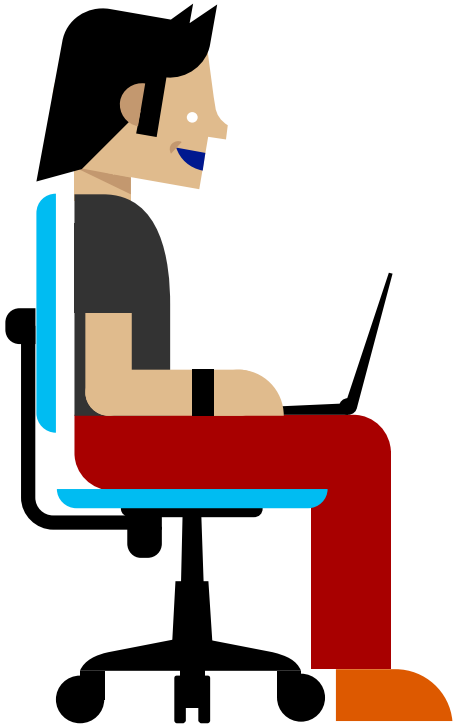


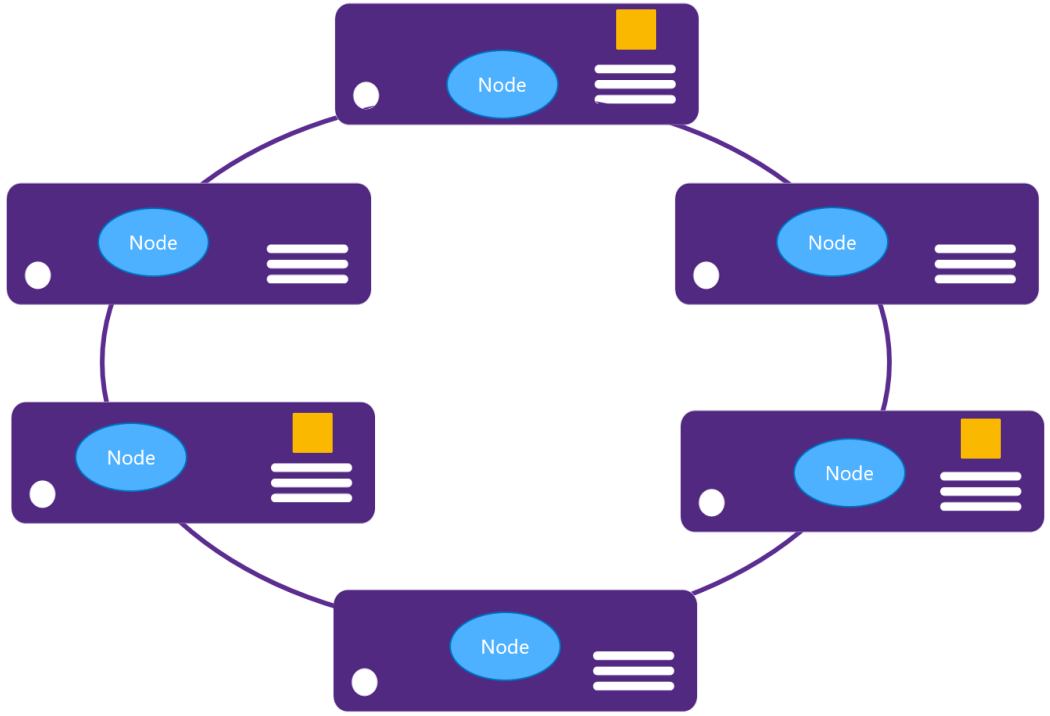
Upgrade application (with zero downtime)





Upgrade application (with zero downtime)

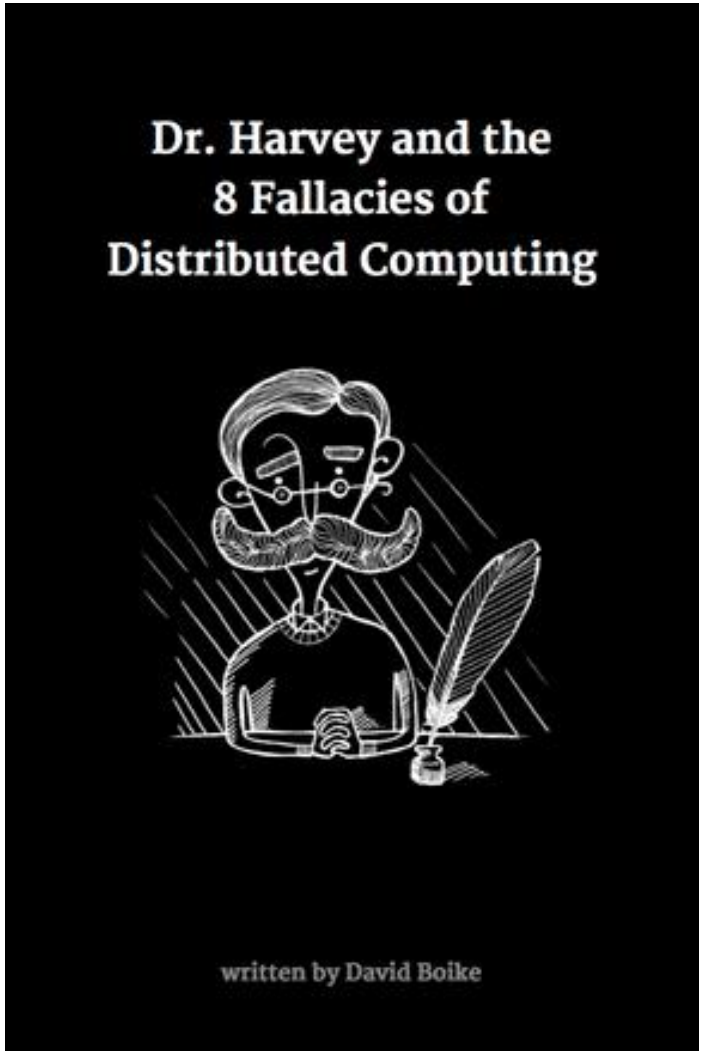




DEMO

upgrade application

Curiosi?



Free e-book available at:

http://go.particular.net/Liguria