Testing a Service Fabric solution and live happy!!!

Massimo Bonanni

Paranormal Developer, with the head in the Cloud and all the REST in microservices!

massimo.bonanni@microsoft.com

@massimobonanni

#CodeGen

#dotnetconf

@cloudgen_verona

Main sponsor

24/CO

Community sponsors

CloudGen

Syncfusion®

Packt

MANNING

elmah.io

Udemy

JET BRAINS

Local Event

It's FREE!
.NET Conf
Discover the world of .NET
September 12-14, 2018
Tune in: www.dotnetconf.net
.NET

Testing Service Fabric …
It ….. could …. work!!!!!

Unit testing

Service Fabric: Microservices platform

Build and deploy applications and microservices on Windows and Linux, at any scale, on any cloud

ASP.NET Core

Reliable Actors (.NET, Java)

Reliable Services (.NET, Java)

Containers

Executables

- Programming Models
- Dev & Ops Tooling
- Orchestration
- Lifecycle Management
- Health & Monitoring
- Always On Availability
- Auto Scaling

Dev machine

Any cloud

On-premises infrastructure

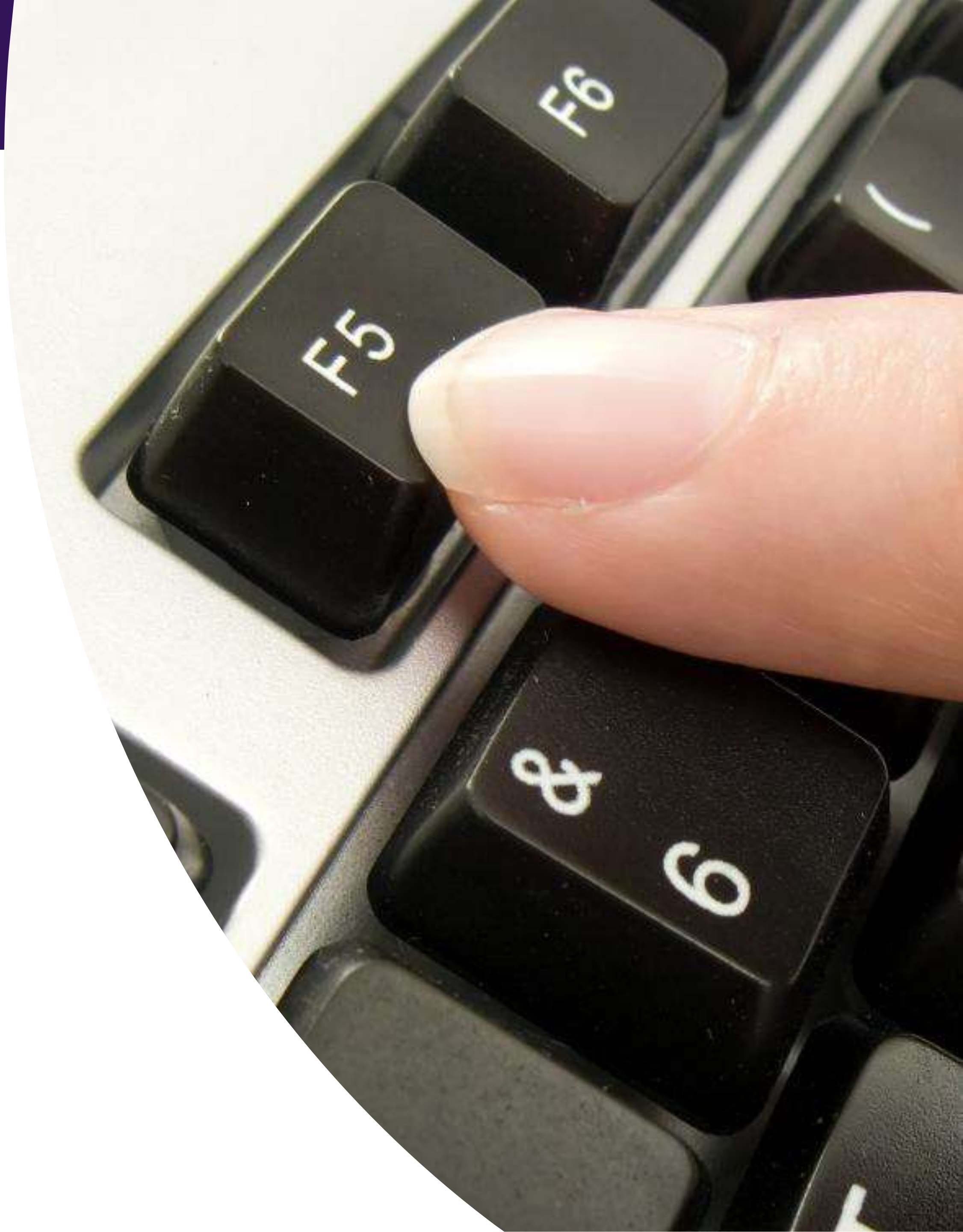
Azure

F5 is our friend....

You don’t have to write any additional code

It's free and you can find it in all Visual Studio versions

It is the most used test tools



F5 is not useful …. with Service Fabric

Resources – you need a local cluster

Time – you need to publish the app

Code complete – you need all the microservices you interact with

Why unit test…

A unit test is an automated piece of code that invokes a unit of work in the system and then checks a single assumption about the behavior of that unit of work.

Roy Osherove – The Art Of Unit Testing

Microservices are small, perform single functionality and loosely coupled.

Microservices Architecture

Why unit test…

A unit test is an automated piece of code that invokes a unit of work in the system and then checks a single assumption about the behavior of that unit of work.

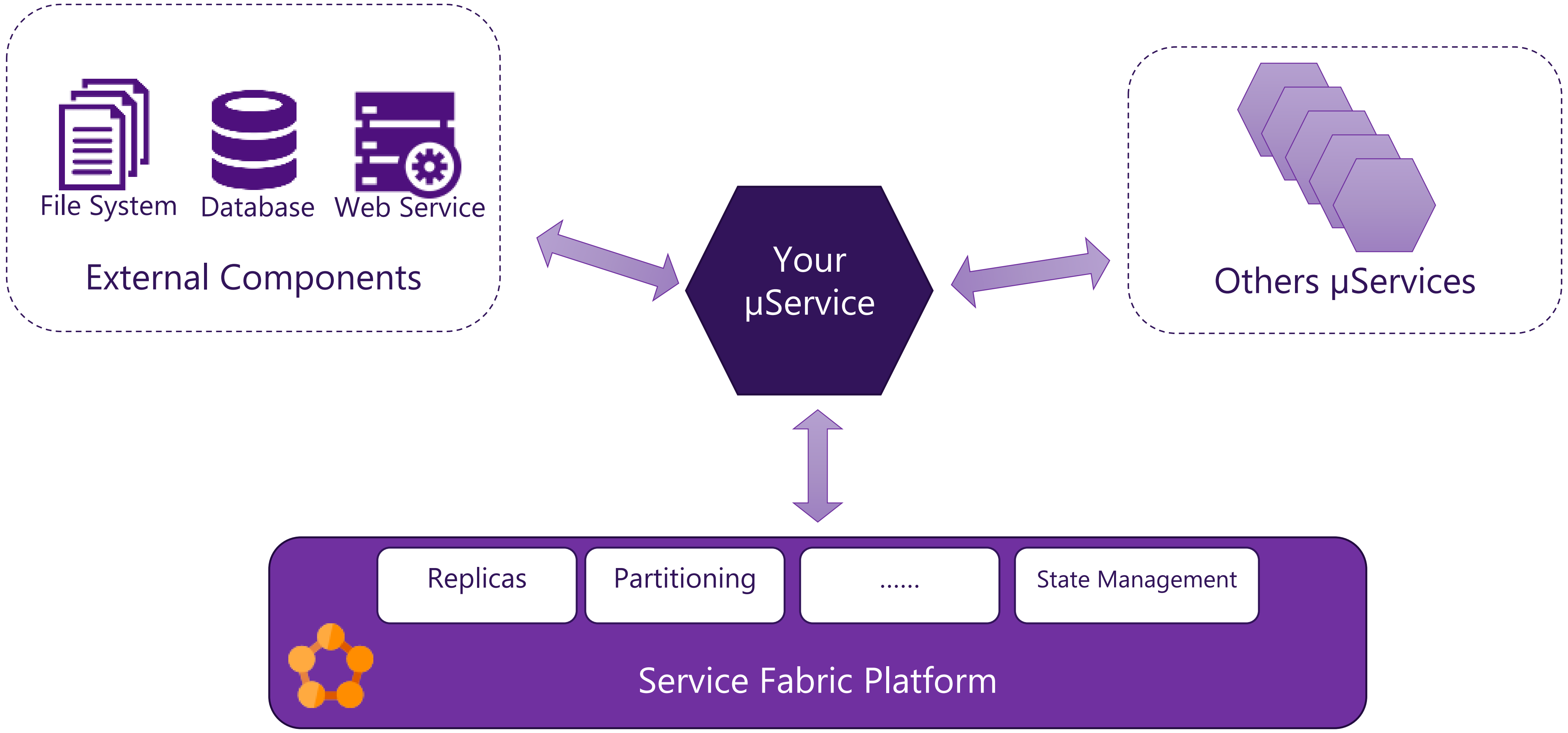
Roy Osherove – The Art Of Unit Testing

Microservices are small, perform single functionality and loosely coupled.

Microservices Architecture

A microservice is your Unit of Work to test!!!

The three obstacles



Real case: Shopping Cart & Order

We model 3 typical shopping cart scenarios

Creating the shopping cart

- The cart can be created only if it is a new cart

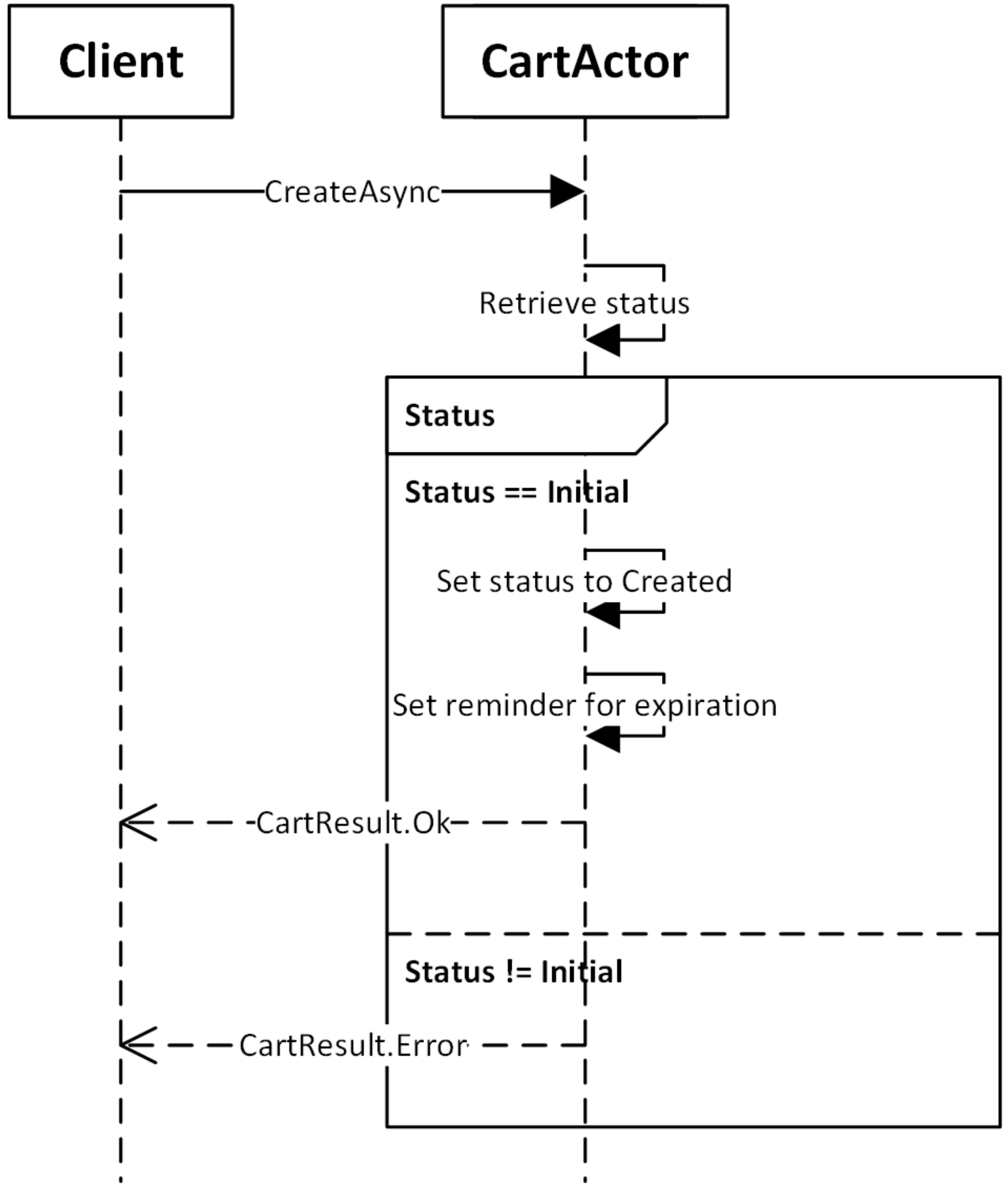
Adding a product to the cart

- Need to check if the product is still available

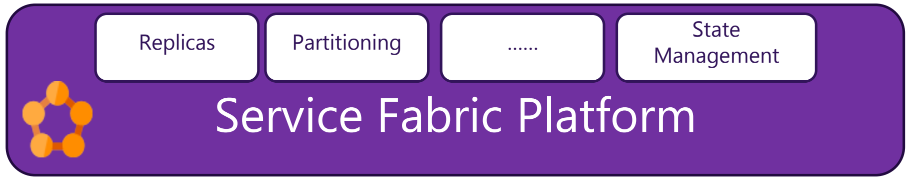
Creating an order from the cart

- The order is created starting from the cart and only if the order is new

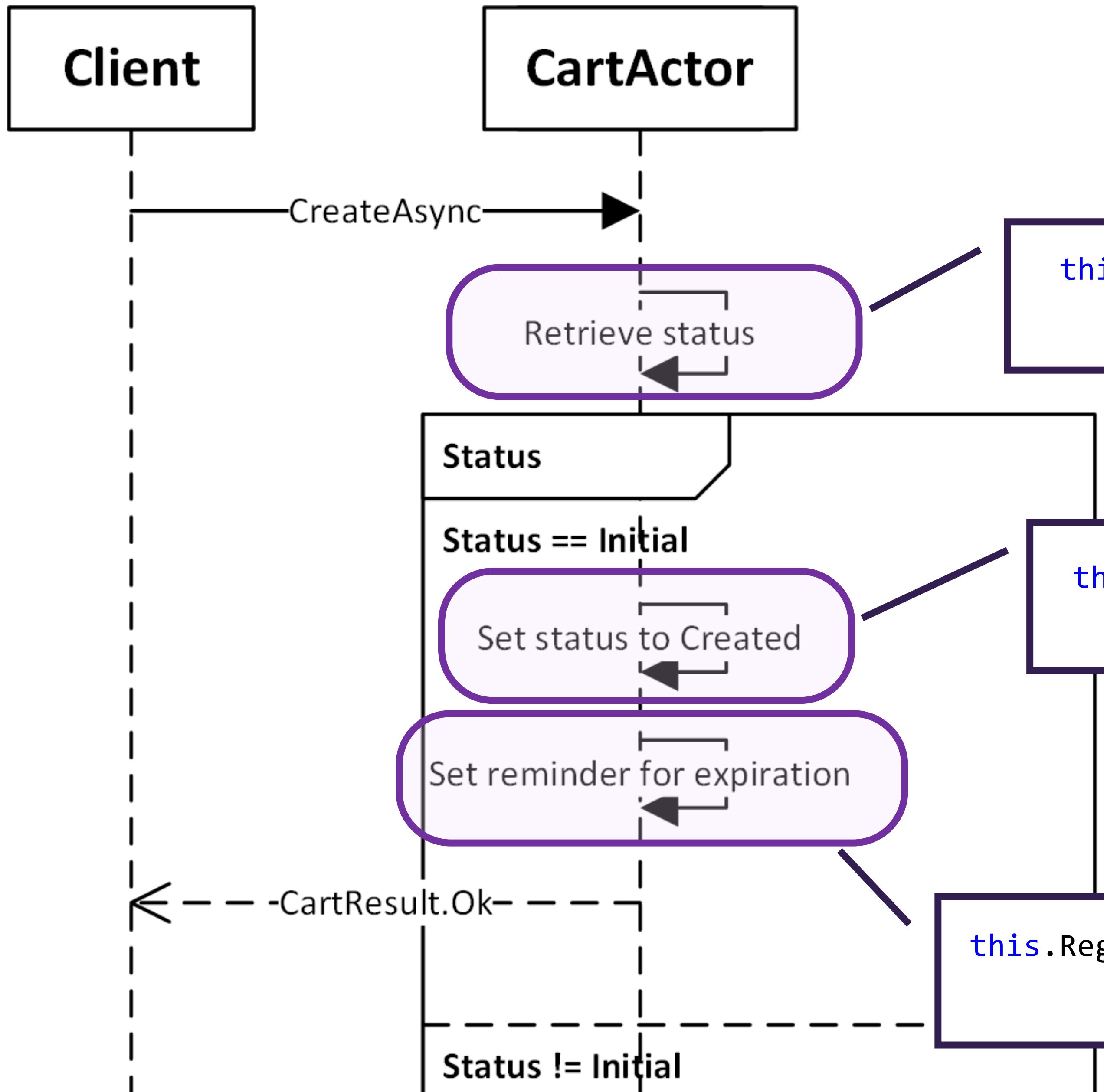
Scenario : Creating the shopping cart



We must "replace" the Service Fabric platform, so our tests doesn’t need the cluster.



Scenario : Creating the shopping cart

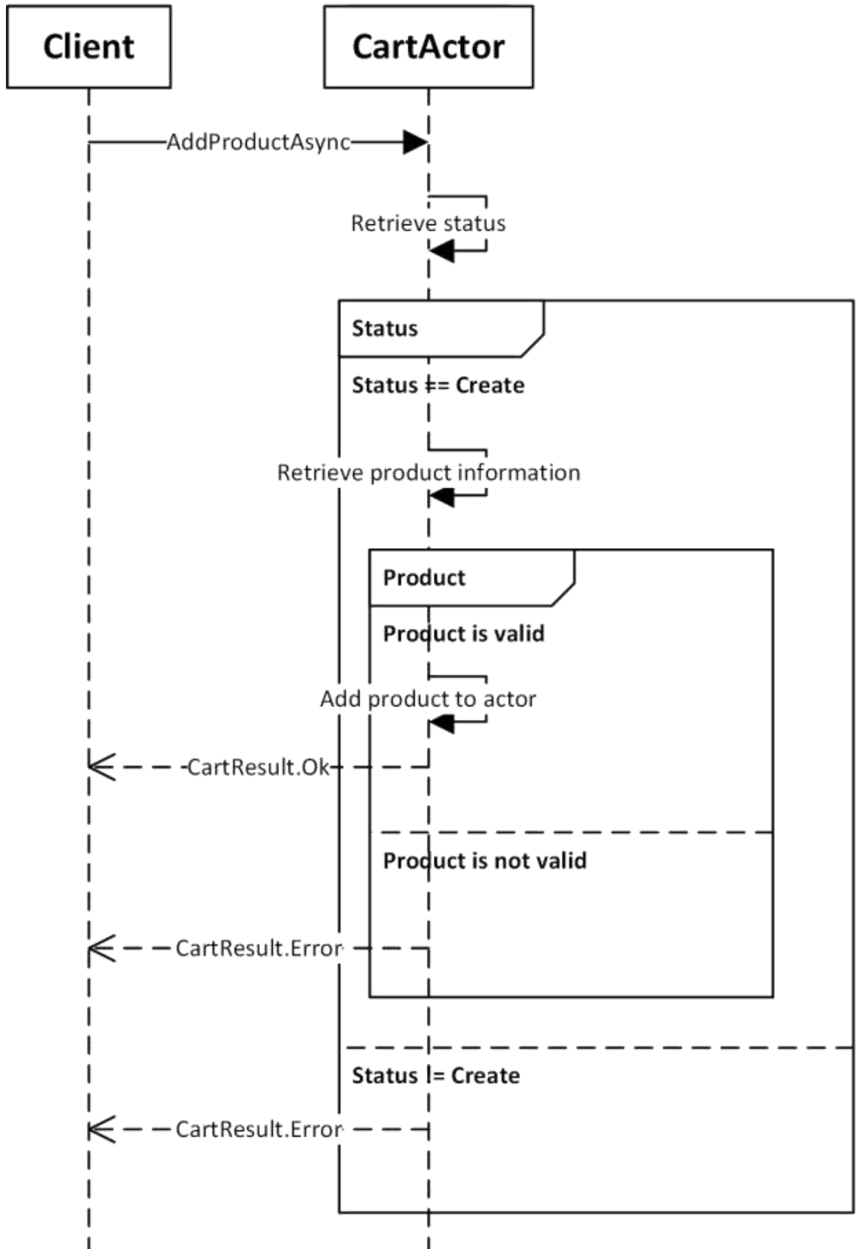


```
this.StateManager.TryGetStateAsync<State>(StateKeyName,
         cancellationToken);
```

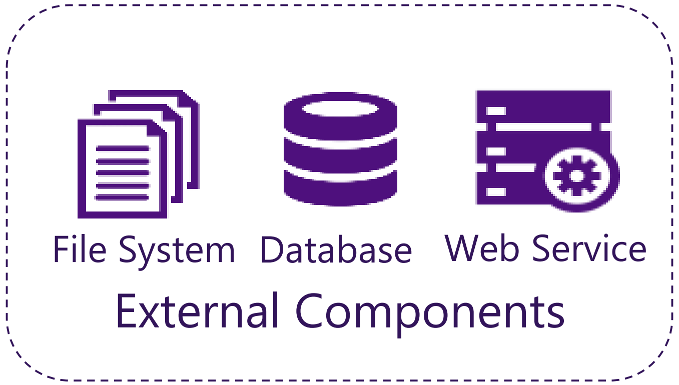
```
this.StateManager.SetStateAsync(StateKeyName,
         state, cancellationToken);
```

```
this.RegisterReminderAsync(ExpiredReminderName, null,
         CartExpiredTimeout, TimeSpan.FromMilliseconds(-1));
```

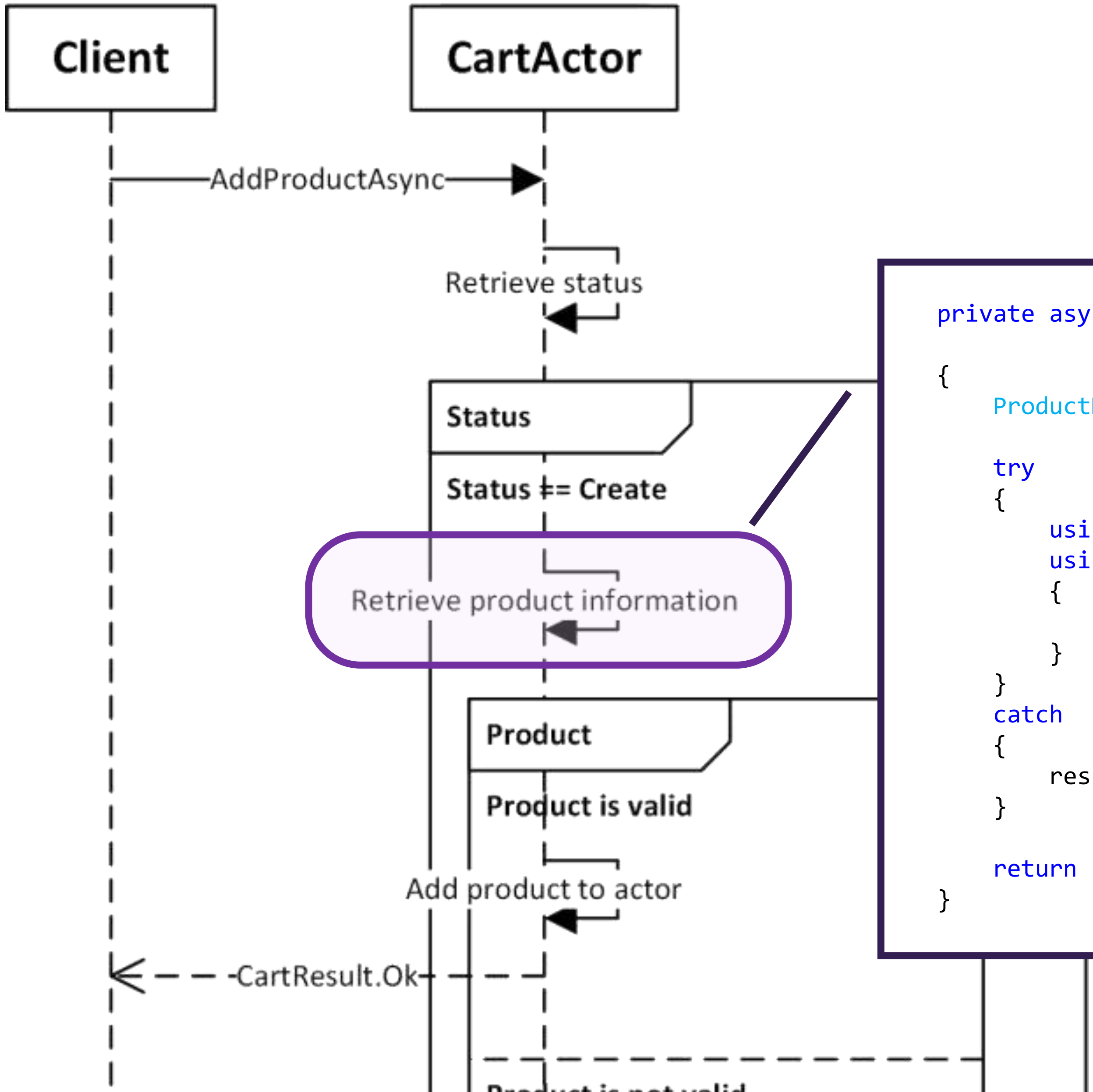
Scenario : Adding a product to the cart



We need to decouple our microservice from external components, so that our tests can verify only the actor's logic.

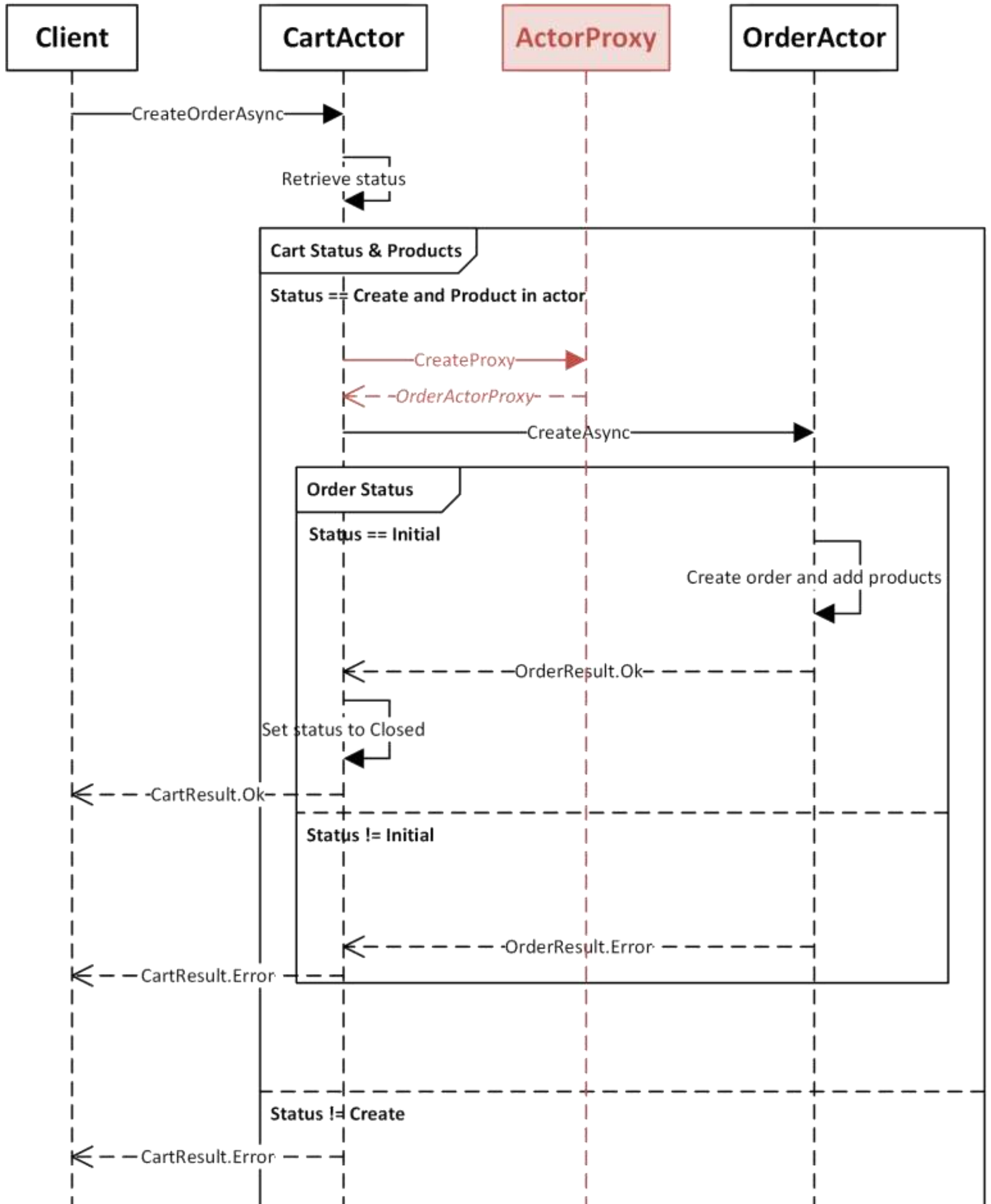


Scenario : Adding a product to the cart



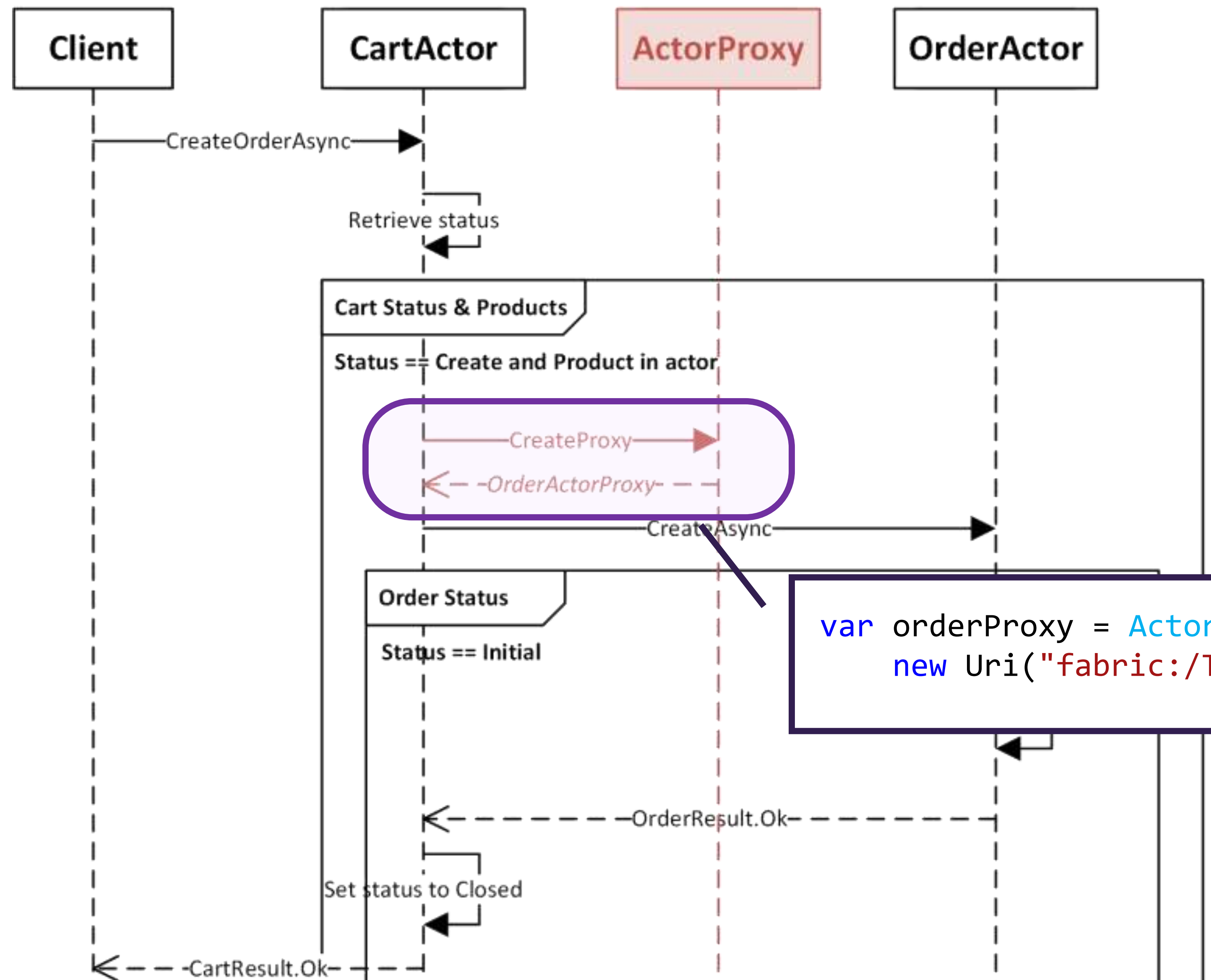
```
private async Task<ProductData> GetProductFromStorageAsync(string productId,
            double quantity, CancellationToken cancellationToken)
{
    ProductData result = null;

    try
    {
        using (SqlConnection conn = new SqlConnection(connectionString))
        using (SqlCommand cmd = new SqlCommand("CheckProduct", conn))
        {
            // Database code
        }
    }
    catch
    {
        result = null;
    }

    return result;
}
```

Scenario : Creating an order from the cart



We must to replace the static classes `ActorProxy` and `ServiceProxy` used in the creation of communication proxies.

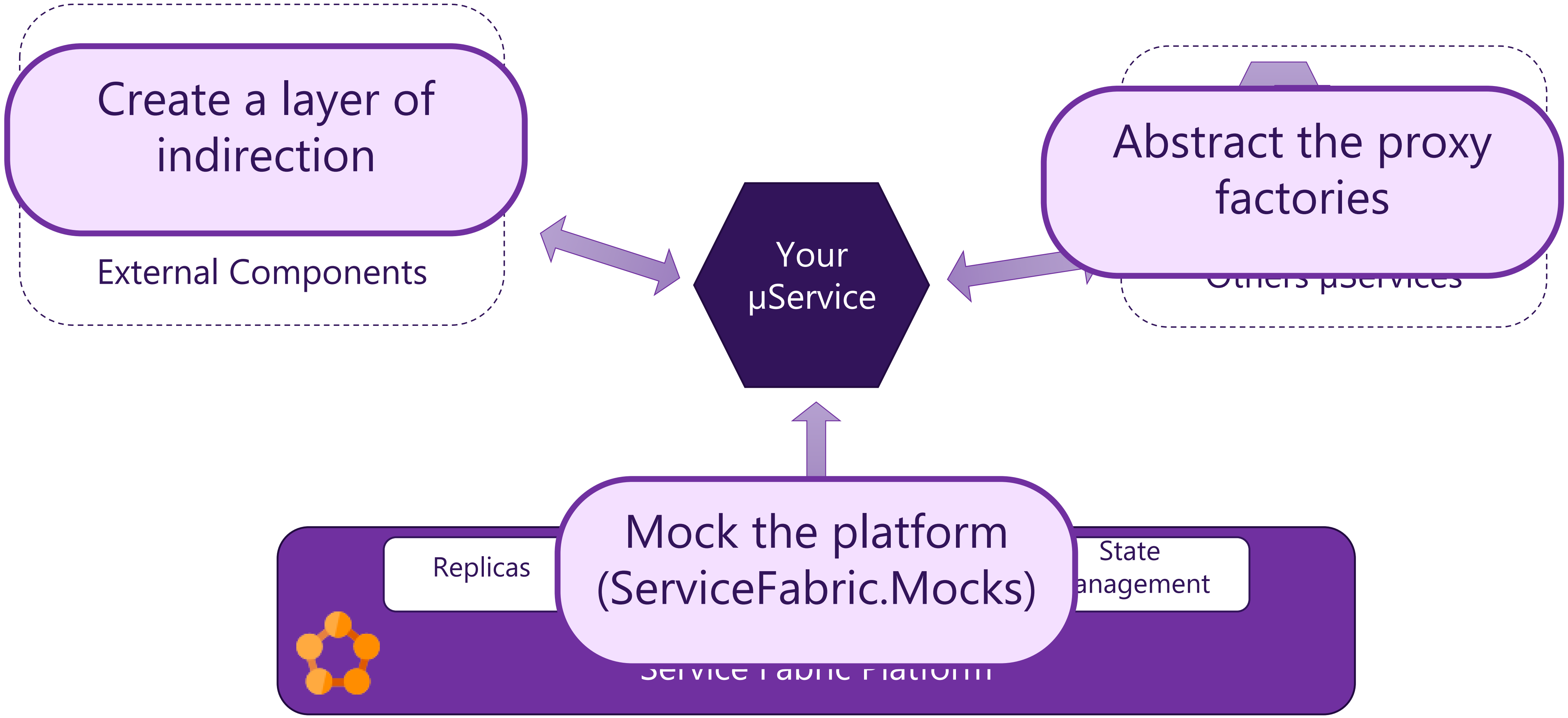
Scenario : Creating an order from the cart



```
var orderProxy = ActorProxy.Create<IOrderActor>(this.Id,
    new Uri("fabric:/TestingApp/OrderActor"));
```

Demo:
Unit test

The three obstacles….three solutions!


Create a layer of indirection
External Components
Your µService
Abstract the proxy factories
Others µServices
Replicas
State
anagement
Mock the platform (ServiceFabric.Mocks)
Service Fabric Platform

The CHAOS

Keep the stability!!!!

Solutions based on distributed architectures such as cloud infrastructures must be:

Resilient	•able to withstand or recover quickly from difficult conditions
Reliable	• its status does not get corrupted as a result of a problem

This is why you need to be able to test the stability of your solutions when complex state transitions due to errors occur in the underlying infrastructure.

What can go wrong in production?

Moving a primary replica

Removing a replica

Restarting a replica

Restarting a distributed code package

Restarting a node

Moving a secondary replica

Remove a node

Add a node

At my cmdlet,
… unleash CHAOS!!!

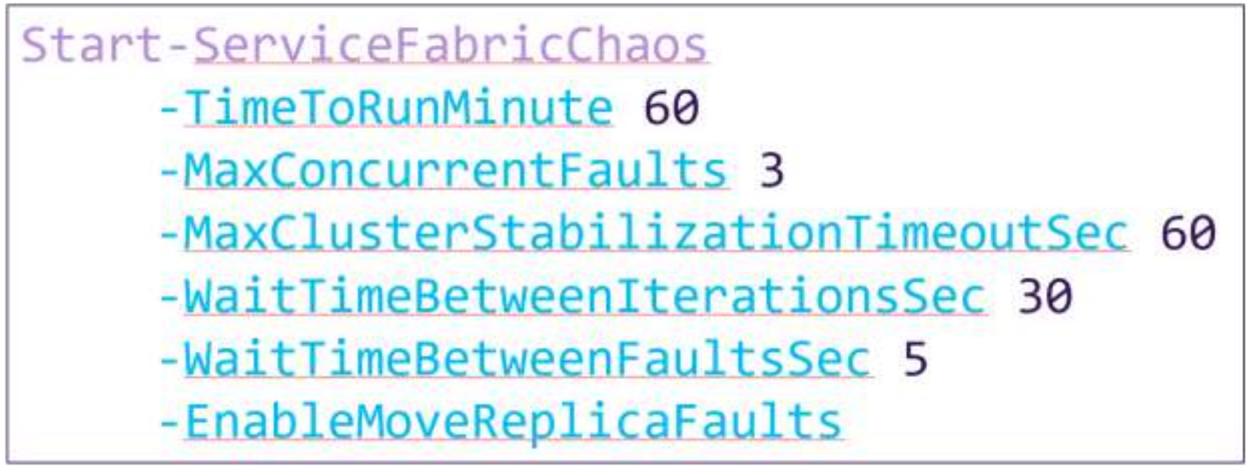
Start-ServiceFabricChaos

Chaos induces faults in the cluster based on the received input parameters.

Chaos runs in multiple iterations: each iteration consists of faults and cluster validation.

You can control:

- how long Chaos runs,
- how long it waits between iterations,
- how many faults it can induce during an iteration,
- how long it waits between faults.
- ….



```
Start-ServiceFabricChaos
        -TimeToRunMinute 60
        -MaxConcurrentFaults 3
        -MaxClusterStabilizationTimeoutSec 60
        -WaitTimeBetweenIterationsSec 30
        -WaitTimeBetweenFaultsSec 5
        -EnableMoveReplicaFaults
```

Demo:
Chaos test

Take away....

Keep the microservices simple....they will be testability simply!

When design a solution, keep in mind the testability!

Mock, fake and shim are your best friends!

If car companies launch expensive cars against a wall to test them, why should not you do it with your code?

Grazie

massimobonanni

@massimobonanni

massimobonanni