Testing and monitoring distributed applications build using Azure Functions, CosmosDB and Service Bus

Henry Been

LESS

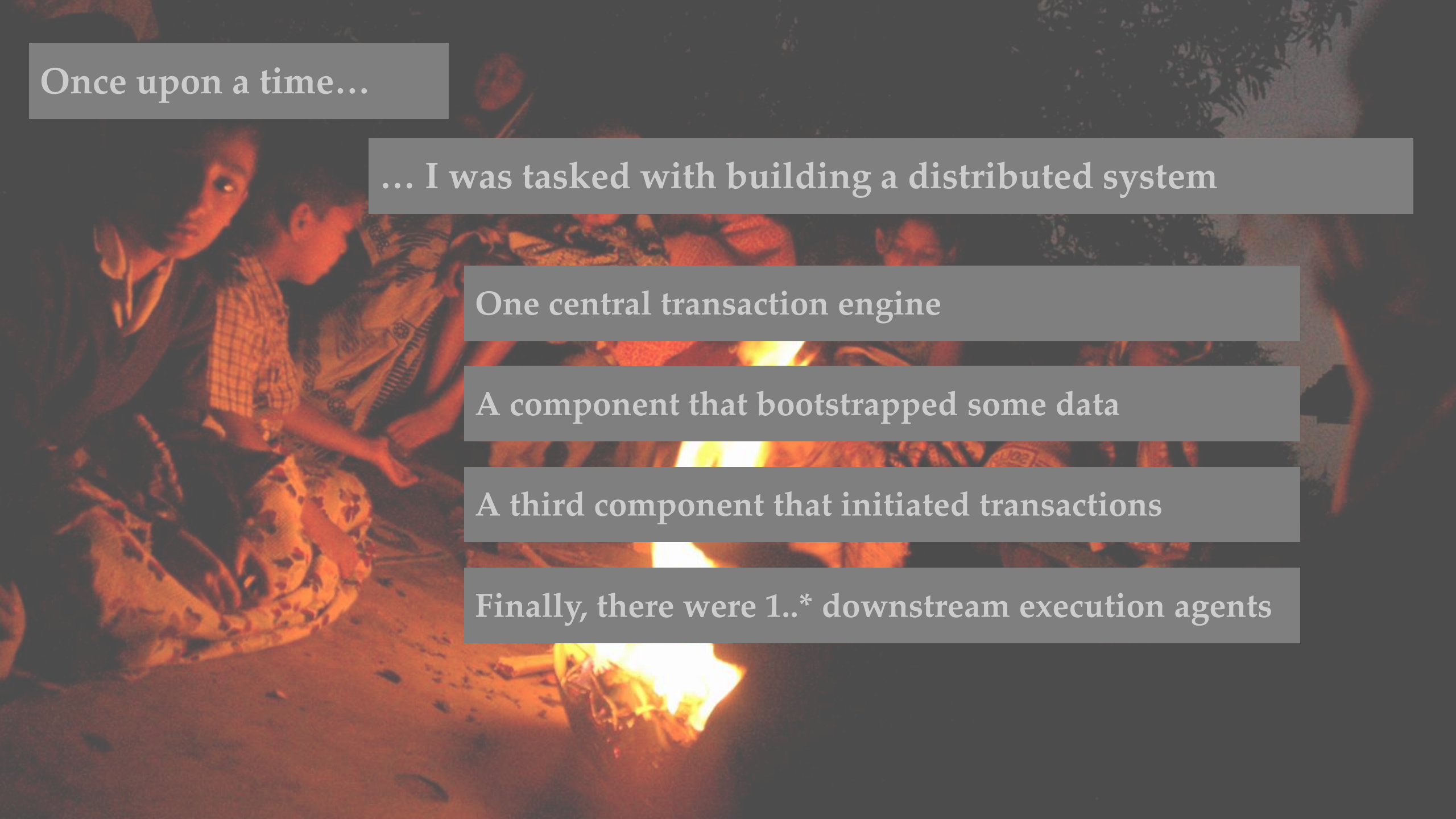
MORE

WONDERING **WHO** IS THAT GUY?

HENRY BEEN
Independent Devops & Azure Architect
Microsoft MVP & MCT

henry@azurespecialist.nl
@henry_been
linkedin.com/in/henrybeen
henrybeen.nl

Microsoft® Most Valuable Professional



Once upon a time…

… I was tasked with building a distributed system

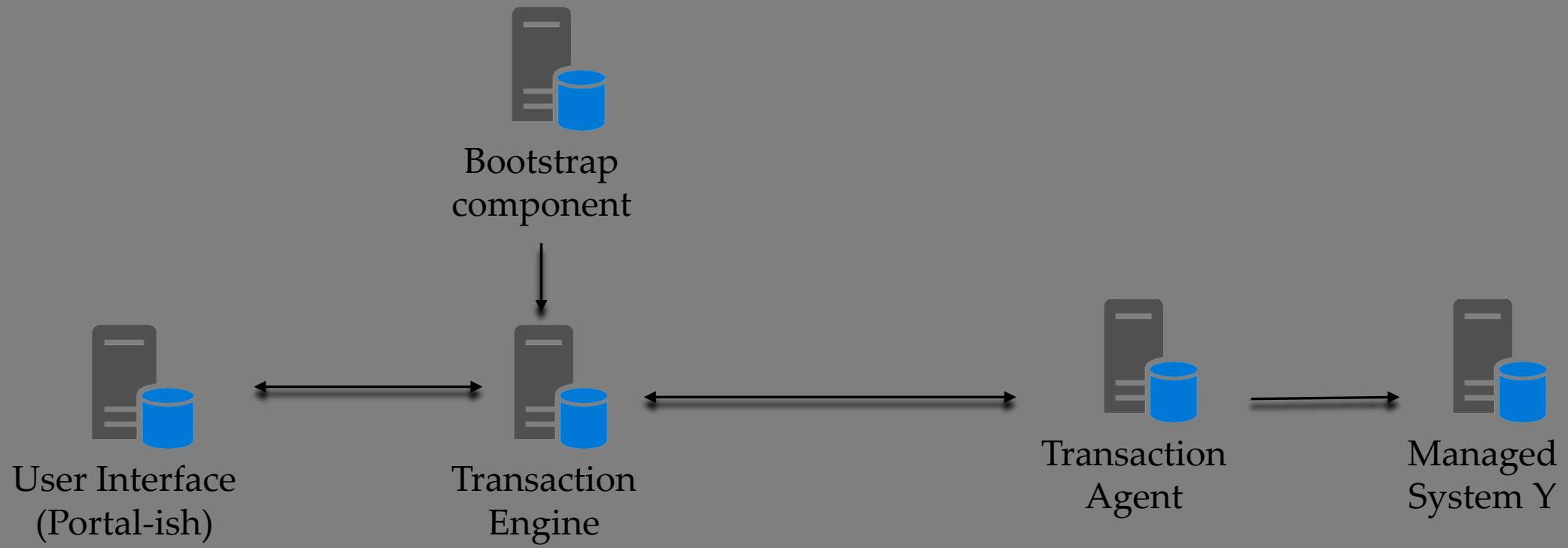
One central transaction engine

A component that bootstrapped some data

A third component that initiated transactions

Finally, there were 1..* downstream execution agents

Something like this…



Once upon a time…

… I was tasked with building a distributed system

Three weeks for a POC

Three months to build

Six FTE to run and further develop and extend

“Good decisions come from experience. Experience comes from making bad decisions.”

Mark Twain

"Good decisions come from experience. Experience comes from making bad decisions. **This is life. So, never regret. Learn from mistakes and go ahead"**

Mark Twain

NO WAY!

You shouldn't build distributed systems

You shouldn’t build distributed systems

Communication is complex

Testing, monitoring and troubleshooting is **HARD**!

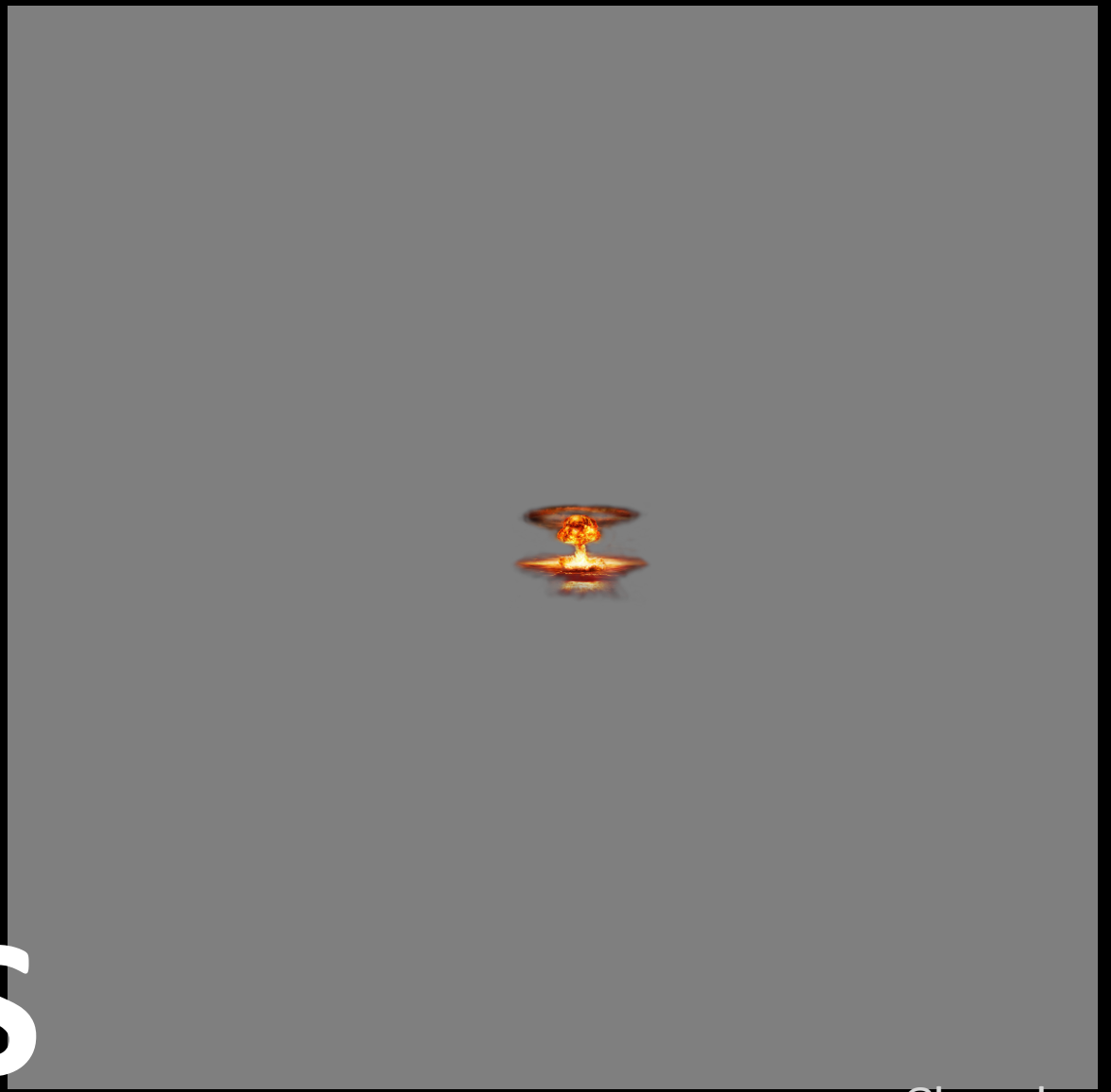
More components, more costs

Requires a different skill- and mindset

Then why are we forced to build distributed systems?

Less impact of errors

VS



Focus for developers

This system we own, understand, develop and monitor. We know it well!

We don't have time to understand 10000s LOC

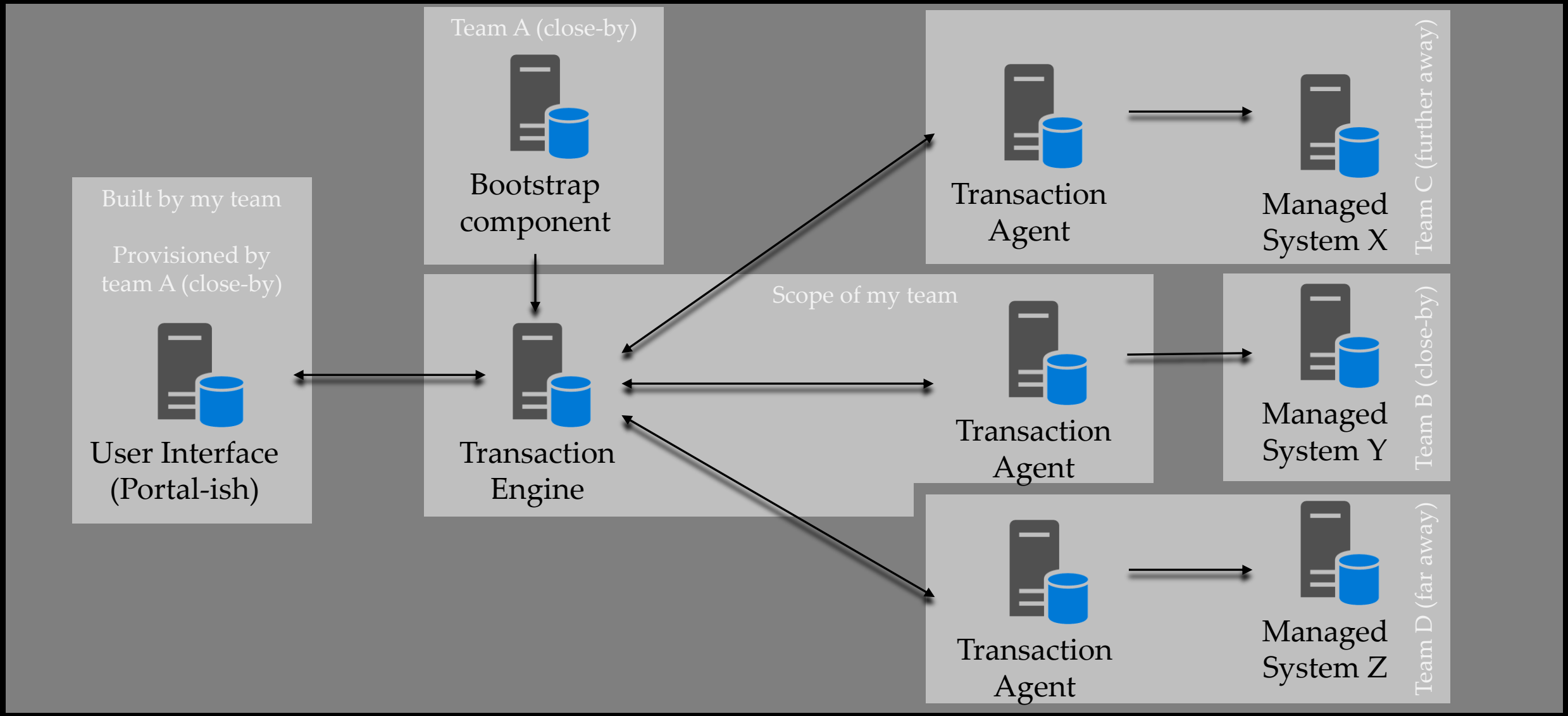
Less impact of errors

Focus for developers

The need for speed

- Smaller systems change faster
- Smaller systems deploy faster
- Smaller teams learn and develop faster
- Smaller components come and go faster

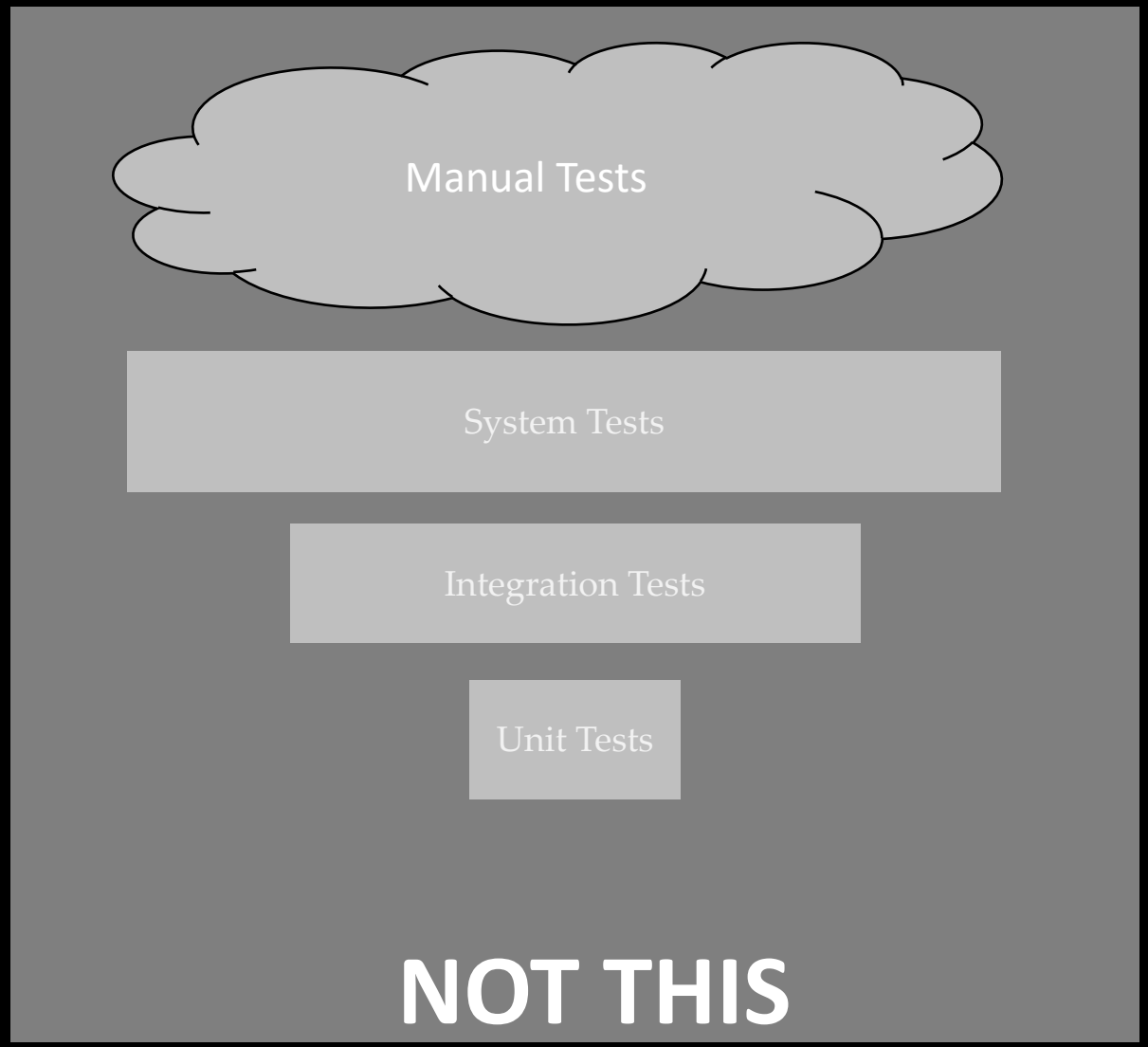
Large organizations are complex



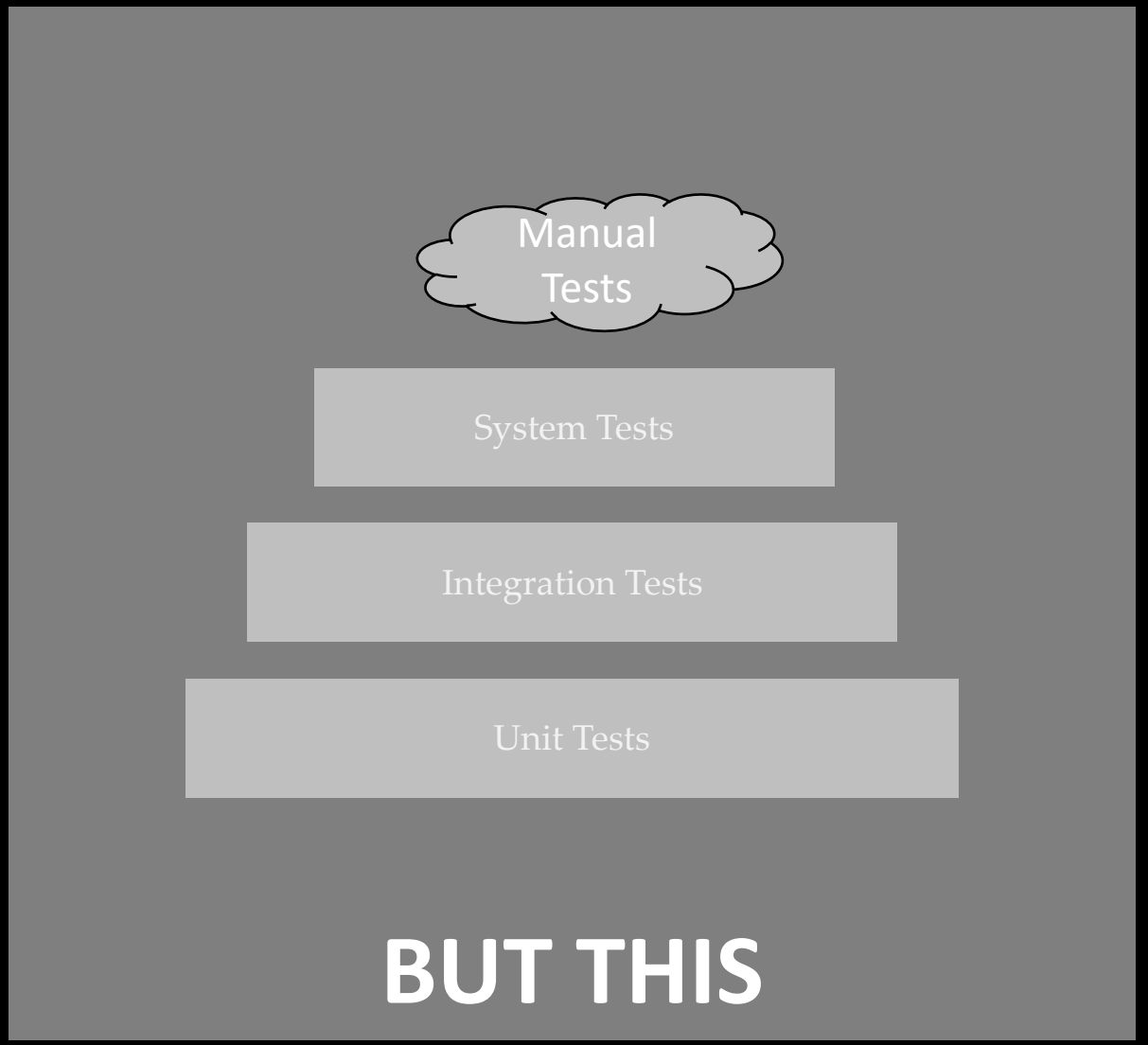
How to test distributed systems

@henry_been

A proper pyramid of tests…



NOT THIS



BUT THIS

How many tests do I need?

Unit test everything you can

Write integration tests against other components..

... that verify your assumptions

... that are as atomic as possible

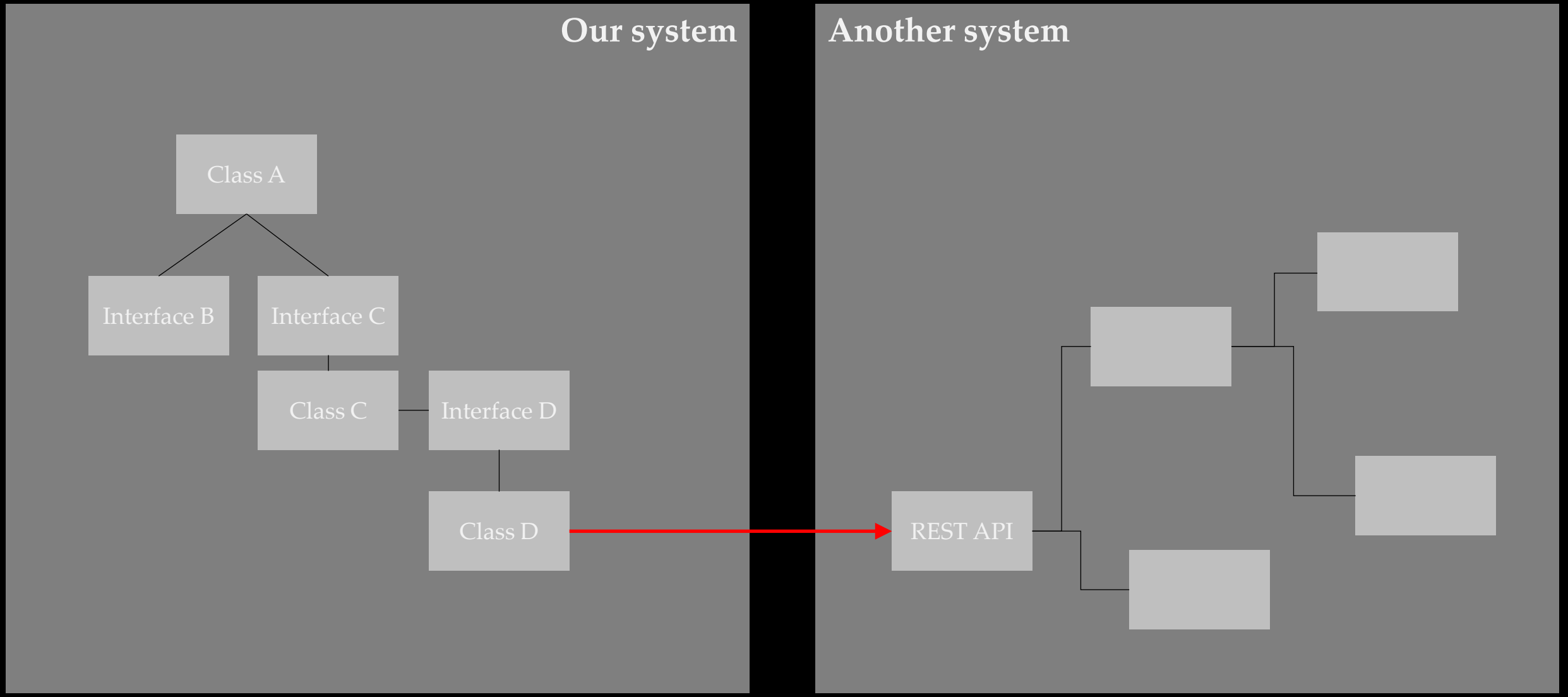
System tests are a necessary evil

A ratio of 1 : 5 – 15 for every level often works, YMMV

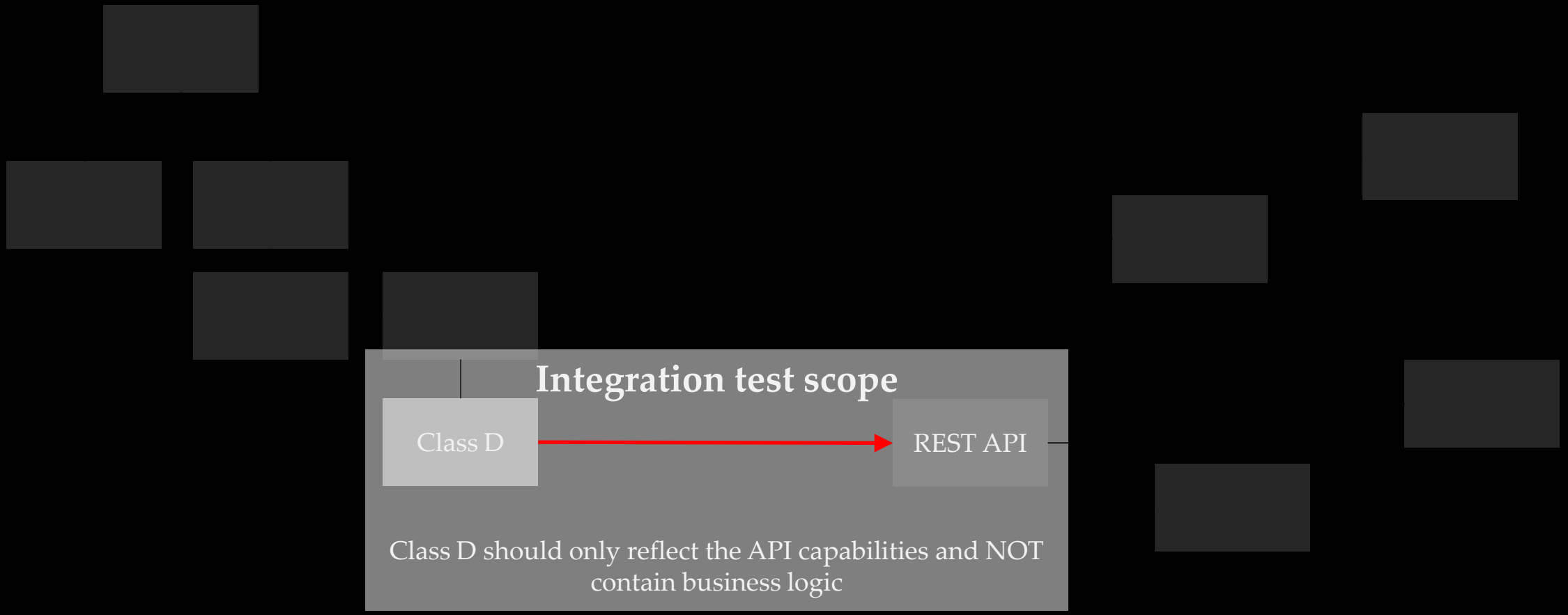
Unit tests

```
[TestFixture]
Public class StuffTest
{
    [TestCase(0)] [TestCase(3)] [TestCase(7)] [TestCase(17)] [TestCase(20)]
    public void WhenUsernameIsOfInvalidLength_ThenItThrows(int usernamelength)
    {
        var username = new string("a", usernamelength)

        TestDelegate act = () => new User(username);

        Assert.Throws<InvalidUsernameException>(act);
    }
}
```

Integration tests



Integration tests



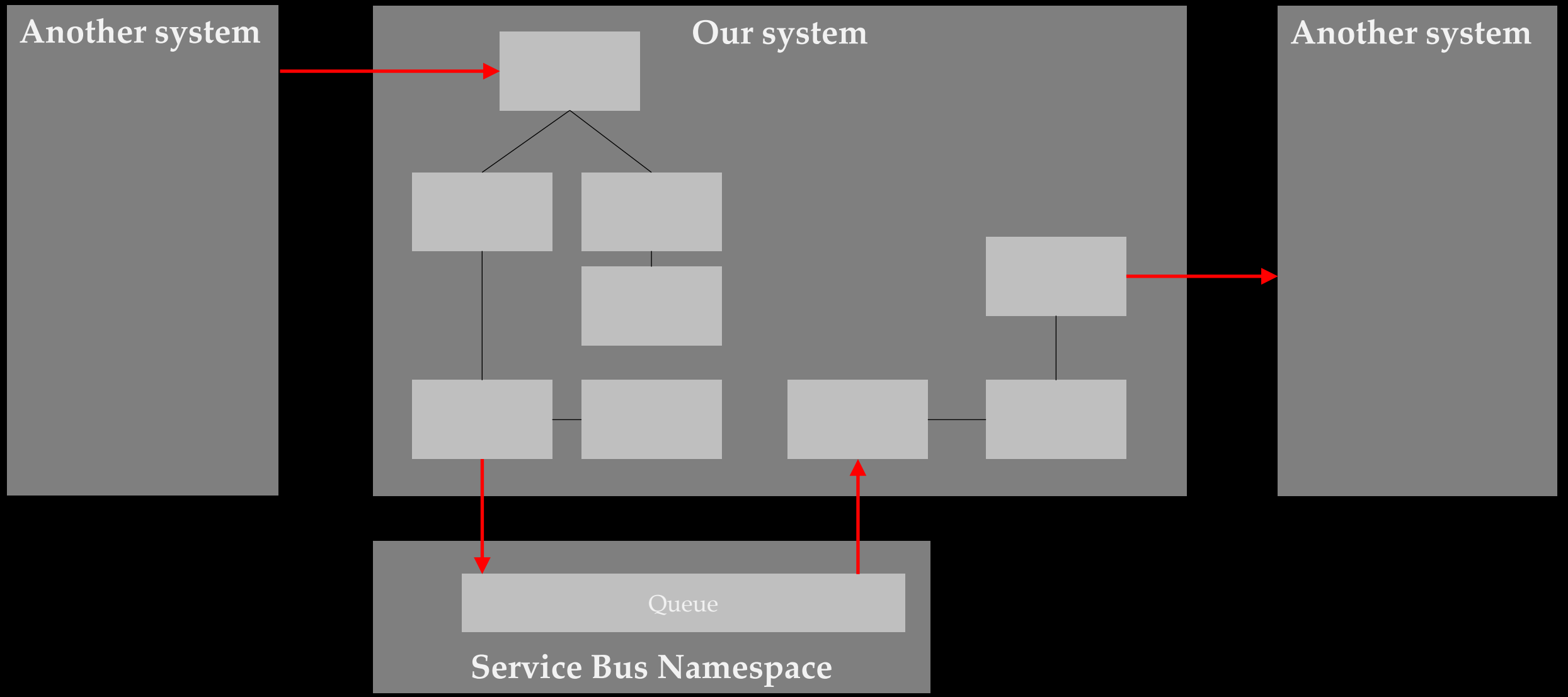
Integration tests

```
[Test]
public async task WhenCreatingUserInExternalApi_ItCanBeRetrieved()
{
    var subject = new ClassD(TestContext.GetClassDCredentials());

    var username = Guid.newGuid().ToString();

    await  subject.CreateUser(username);

    var actual = subject.FindUser(username);

    Assert.IsNotNull(actual);
}
```

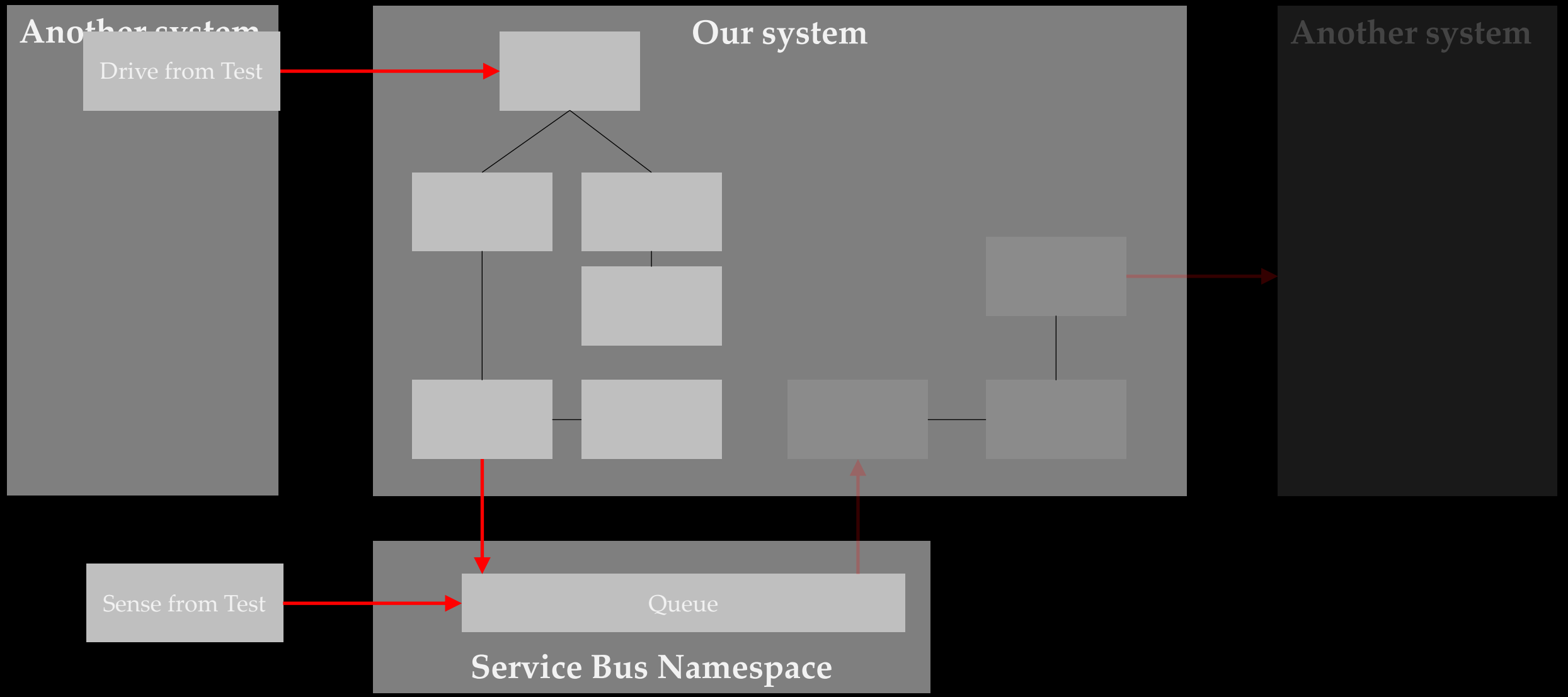
My ***opinion***: connect to the production environment of the other component

or when ***necessary***: create and destroy a new environment for every test case!

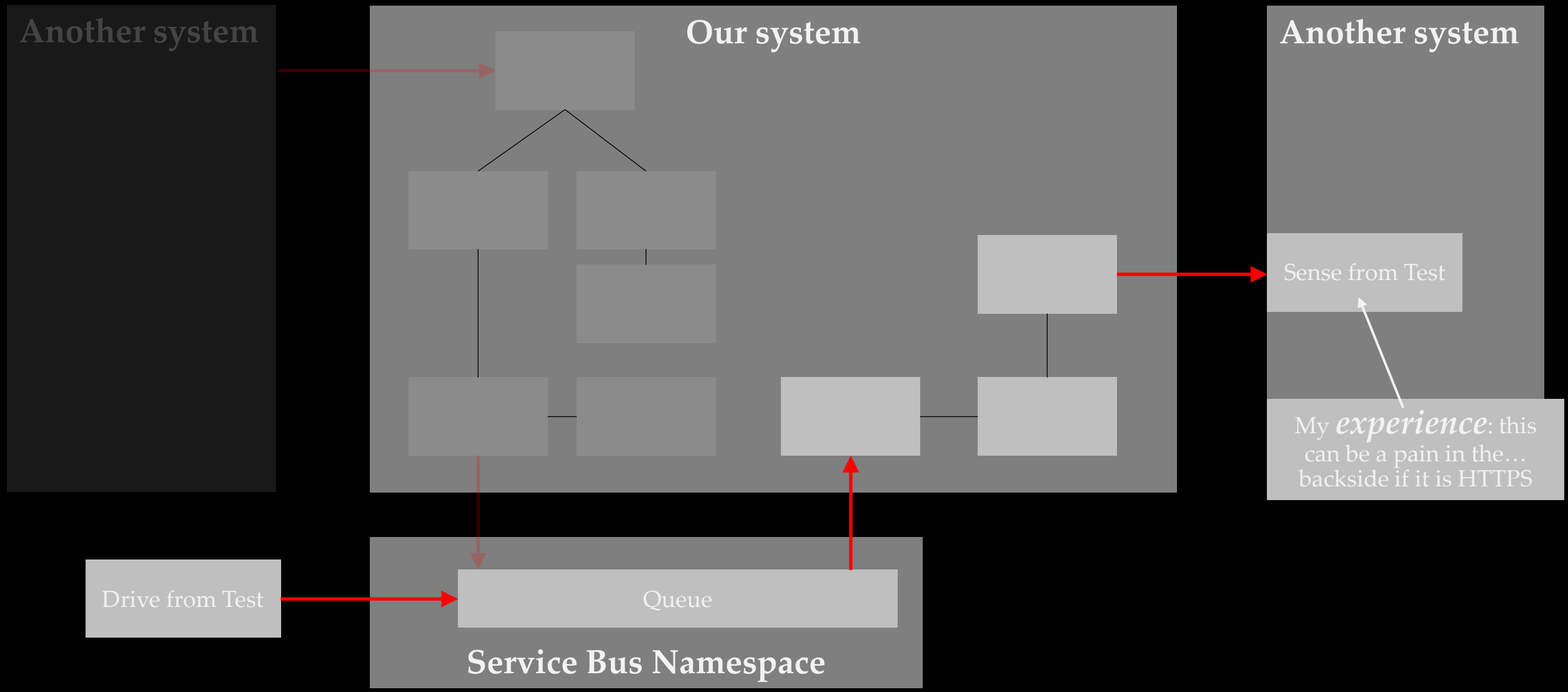
System tests



System tests



System tests




MORE DEMOS,
PLEASE!
SPEED:

Testing distributed systems is hard.

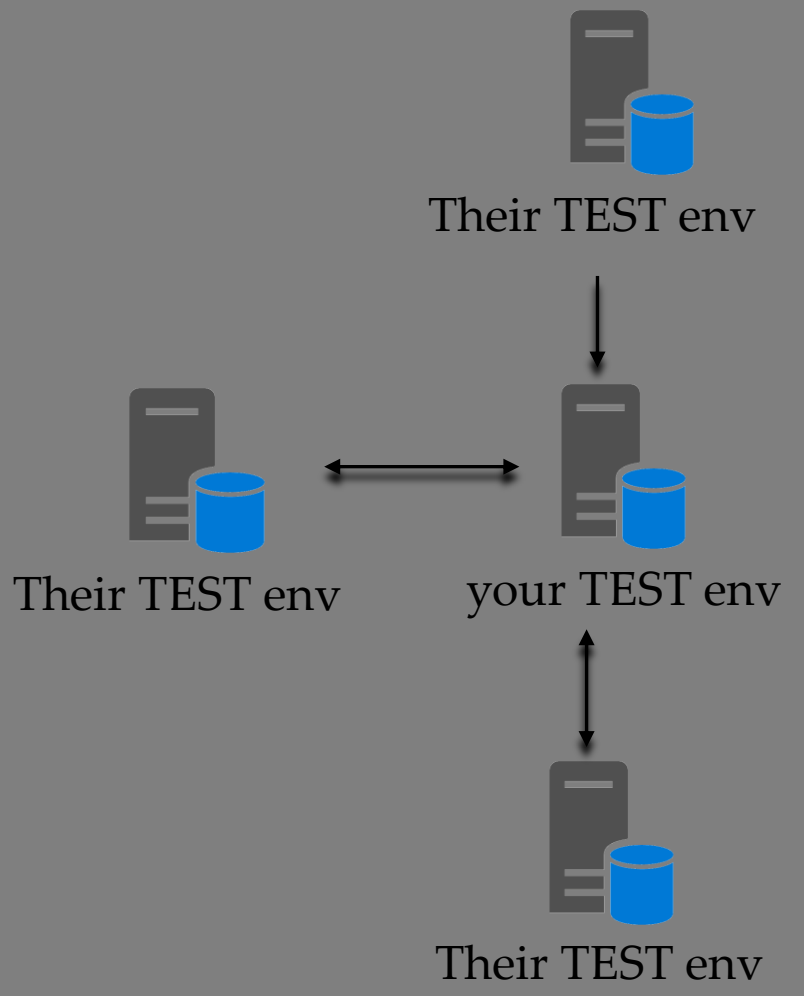
But then there are those that believe in manual testing…

@henry_been

“We need to verify that our system integrates correctly with all systems we depend on and all systems that depend upon us!”

- Your random risk-averse person

Connected test environments



Should we connect test environments?

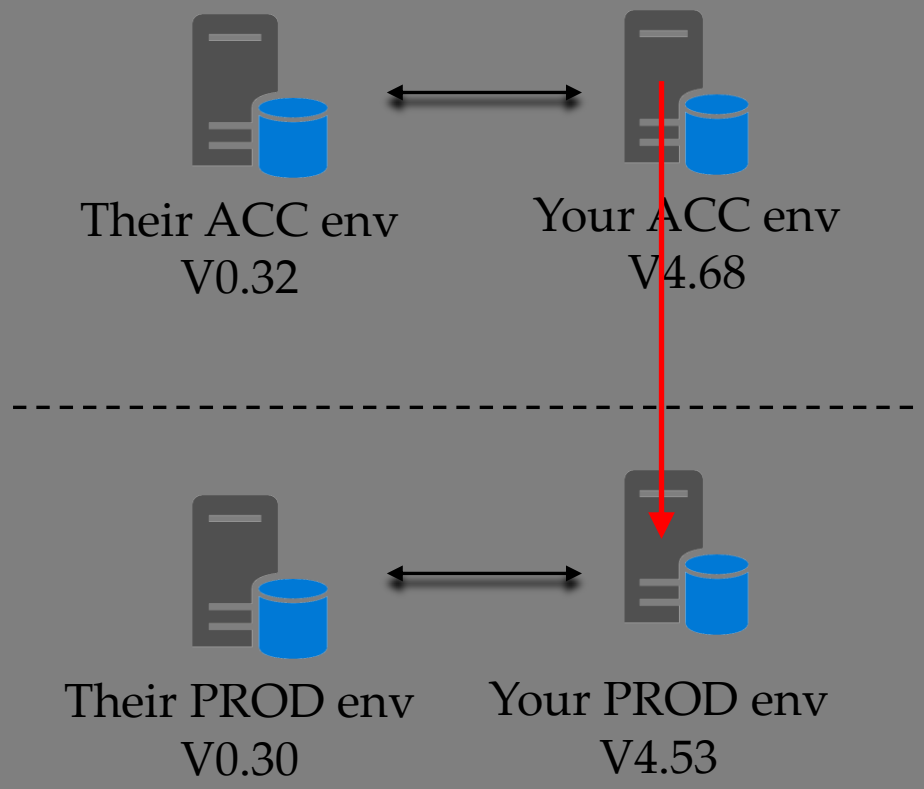
HELL NO!

They will not stop before all test systems are connected!

You will get complaints when YOUR test environment is down

State inconsistencies left and right will invalidate all test results

Connected acceptance environments instead



Should we connect acceptance environments?

HELL NO!

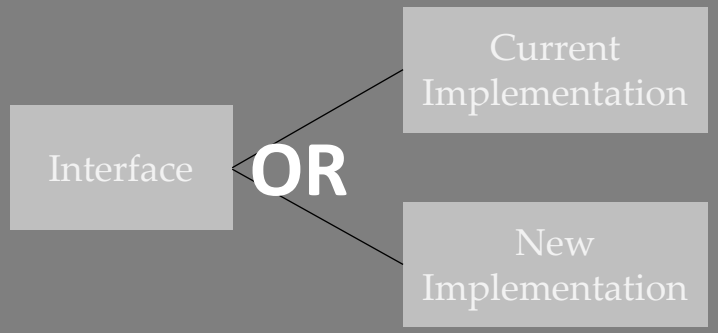
They will not stop before all acceptance systems are connected!

You will only know if your software works with future versions!

“East, West,
~~Home~~ Production Best!!”

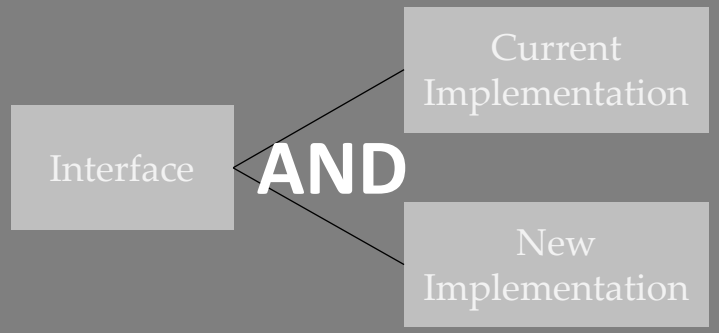
Testing in production

Feature Toggles



Execute current OR new implementation, depending on configuration

Experiments



Execute both implementations, but only return result from current implementation

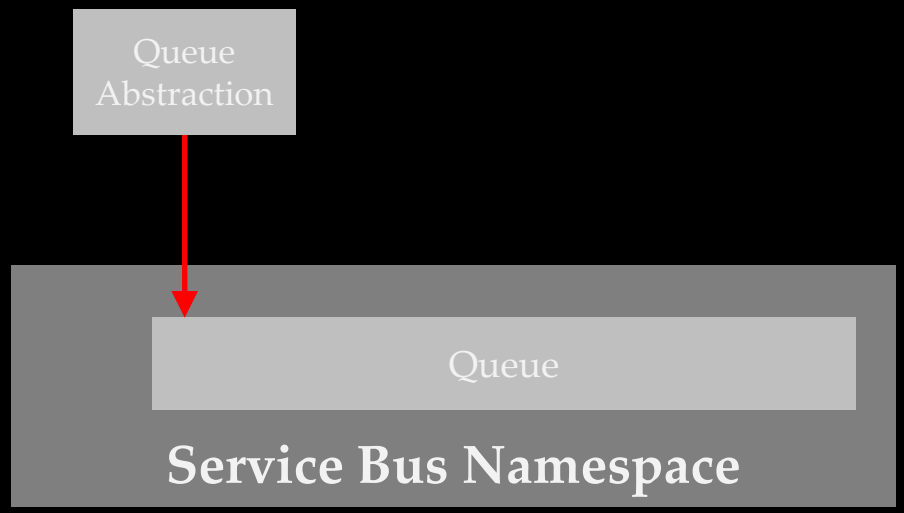
Testing in production

```
public async Task OnPost()
{
  Position = FibonacciInput.Position;

  Result = await Scientist.ScienceAsync<int>("fibonacci-implementation", experiment =>
  {
    experiment.Use(async () => await _recursiveFibonacciCalculator.CalculateAsync(Position));
    experiment.Try(async () => await _linearFibonacciCalculator.CalculateAsync(Position));

    experiment.AddContext("Position", Position);
  });
}
```

https://www.henrybeen.nl/running-experiments-in-production/

Contract testing

Both the producer and consumer of messages, constantly validate that all messages they send or receive confirm to the agreed contract and if not decline the message. This also in production.




```
Class QueueAbstraction
{
    private readonly ISdkQueueSender _sdkObject;
    private readonly  IMessageContractValidator _messageValidator;

    // ctr with DI for validator
    // and construction of SDK object from injected credentials

    public async Task SendMessage(Message message)
    {
        await _messageValidator.ValidateAsync(message);

        await  _sdkObject.SendMessageAsync(message);
    }

}
```

Contract testing

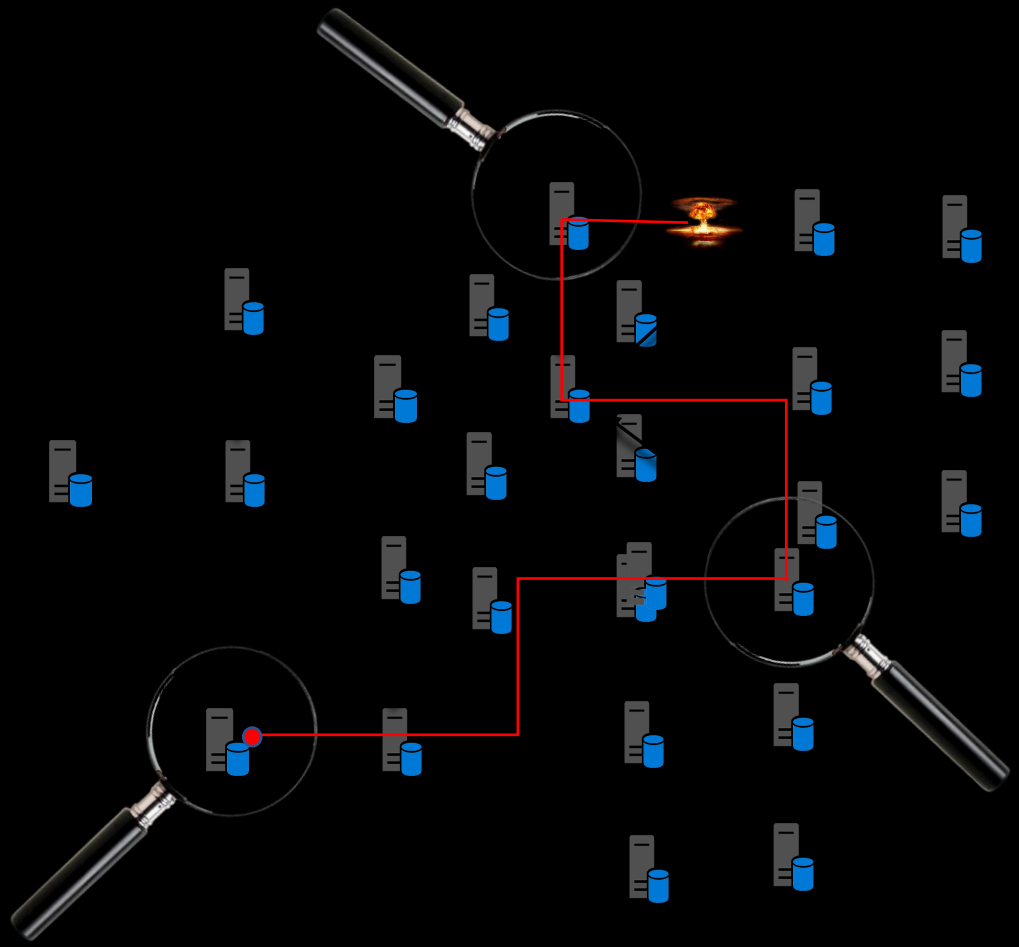
Both the producer and consumer of messages, constantly validate that all messages they send or receive confirm to the agreed contract and if not decline the message. This also in production.

Shared Message Descriptor (JSON Schema, AVRO schema, …)

Published by consumer


```
Class QueueFunction
{
   private readonly IBusinessLogic _ myBusinessLogic;
   private readonly  IMessageContractValidator _messageValidator;

   // ctr with DI for validator  and business logic

   [FunctionName("QueueConsumer")]
   public async Task Run(… Message message)
   {
      await _messageValidator.ValidateAsync(message);

      await  _myBusinessLogic.ProcessMessageAsync(message);
   }

}
```

How to monitor distributed systems



@henry_been

Monitoring distributed systems

Alert on every backend invocation failure

Have distributed tracing in place

Have deadletter queues everywhere

Use synthetic monitoring / use case monitoring

Alert on every deadletter message

Alert on resource consumption limits

Alert on critical log entries

Fix all alert causes / remove all alert noise


NEEDS MORE…
DEMOS!
SPEED:

Only built distributed systems when you really, really have to!

Do not connect your non-production environments with other components

Build a proper pyramid of tests for your system

Provide a means for integration testing to your consumers

Consider contract testing

Monitor every component at the individual level

Implement distributed tracing

DO TRY THIS **AT HOME!**

HENRY BEEN
Independent Devops & Azure Architect

E: henry@azurespecialist.nl
T: @henry_been
L: linkedin.com/in/henrybeen
W: henrybeen.nl

DO TRY THIS AT HOME!

HENRY BEEN
Independent Devops & Azure Architect

E: henry@azurespecialist.nl
T: @henry_been
L: linkedin.com/in/henrybeen
W: henrybeen.nl

Questions?

QUESTIONS?

Now is the time!

DO TRY THIS AT HOME!

HENRY BEEN
Independent Devops & Azure Architect

E: henry@azurespecialist.nl
T: @henry_been
L: linkedin.com/in/henrybeen
W: henrybeen.nl