Particular Software

Welcome

Workshop

Chain of Responsibility

Particular
Software

Solution Architect
Enthusiastic Software Engineer
Microsoft Azure MVP

@danielmarbach
particular.net/blog
planetgeek.ch

Pattern

Build It

WrapUp

OWIN

```
appBuilder.Use(async (ctx, next) =>
{
  // do some things here
  await next();
  // or here
});
```

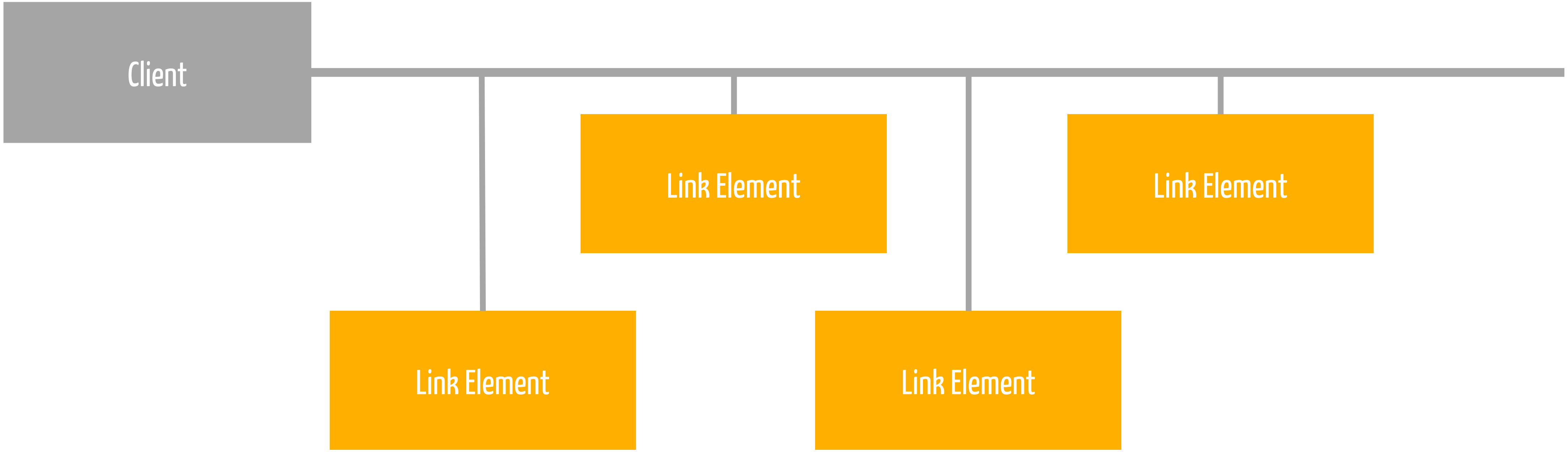
WebApi

```
 class FilterOutInvalidOperationException : IActionFilter  {
     public bool AllowMultiple { get; }
     public async Task<HttpResponseMessage>
ExecuteActionFilterAsync(HttpActionContext actionContext,
CancellationToken cancellationToken,
Func<Task<HttpResponseMessage>> continuation) {
        try {
           var response = await continuation();
           return response;
        } catch (InvalidOperationException) {
        }
        return new HttpResponseMessage();
     }
  }
```

Goals

target

Chain of Responsibility


Client
Link Element
Link Element
Link Element
Link Element

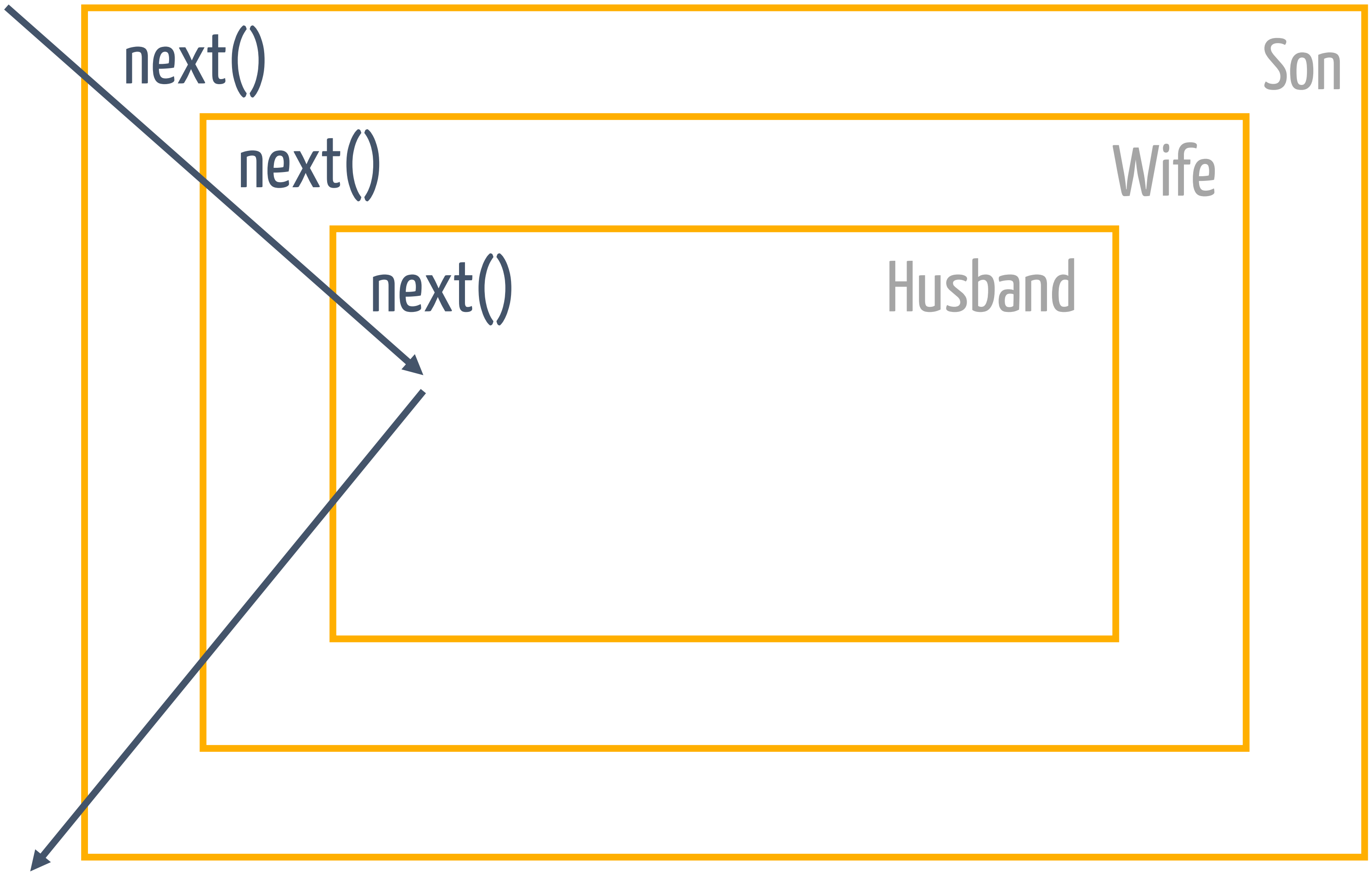
son wife husband

son wife husband

```
static void Person(Action next)
{
    // Implementation
    next();
}
```

```
public void ManualDishwasherUnloading()
{
    Son(() => Wife(() => Husband(() => Done())));
}
```

Coding time

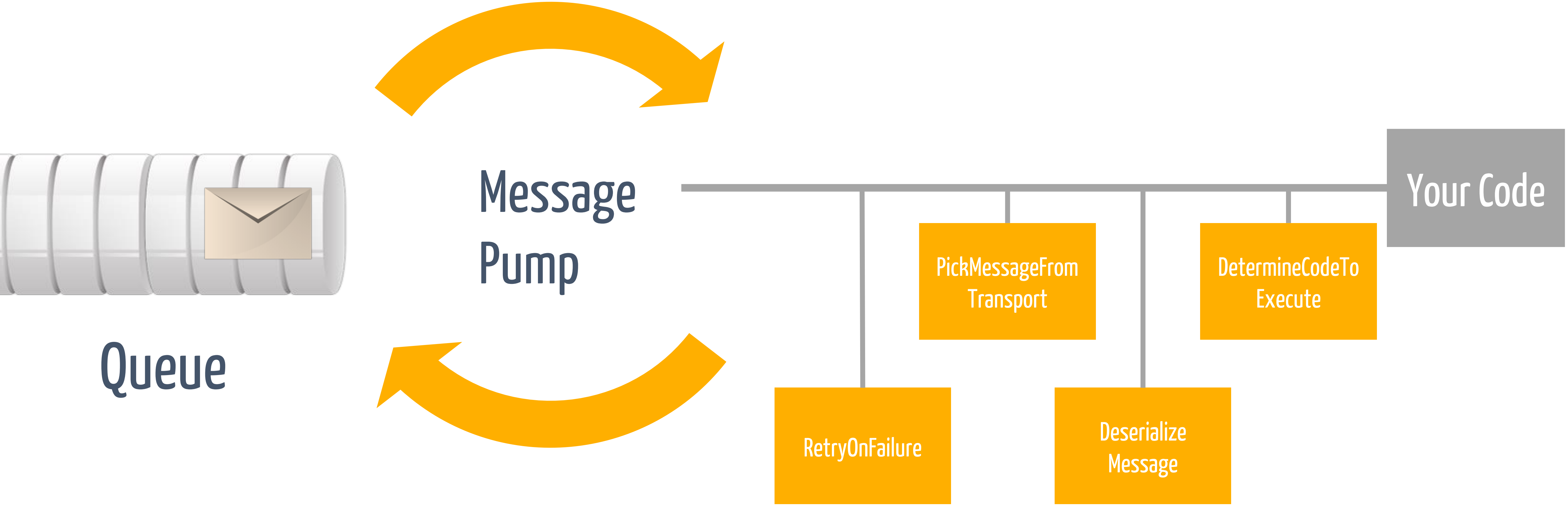

next()
Son
next()
Wife
next()
Husband

cumbersome

Coding time

```
static void IgnoreDishStillWetException(Action next))
{
    try {
        next();
    }
    catch(DishStillWetException) { }
}
```

Coding time



Queue

Message Pump

Your Code

PickMessageFrom Transport

DetermineCodeTo Execute

RetryOnFailure

Deserialize Message

Task

IO-bound

```
await IoBoundMethod();

static async Task IoBoundMethod() {
  using (var stream = new FileStream(...))
  using (var writer = new StreamWriter(stream)) {
    await writer.WriteLineAsync("42");
    ...
  }
}
```

Task

CPU-bound

```
Parallel.For(0, 1000, CpuBoundMethod);
Parallel.ForEach(Enumerable.Range(1000, 2000), CpuBoundMethod);

await Task.Run(() => CpuBoundMethod(2001));
await Task.Factory.StartNew(() => CpuBoundMethod(2002));
```

Avoid

async void

```
try {
  AvoidAsyncVoid();
}
catch (InvalidOperationException e) { }

await Task.Delay(100);

static async void AvoidAsyncVoid() {
  await Task.Delay(10);
  throw new InvalidOperationException("Gotcha!");
}
```

Don’t mix

blocking & async

```
Delay(15);

static void Delay(int milliseconds) {
  DelayAsync(milliseconds).Wait();
}

static async Task DelayAsync(int milliseconds) {
  await Task.Delay(milliseconds);
}
```

Recap

reminder

Use Task.Run, Factory.StartNew for CPU-bound work

Use Task directly for IO-bound work

Use async Task instead of async void

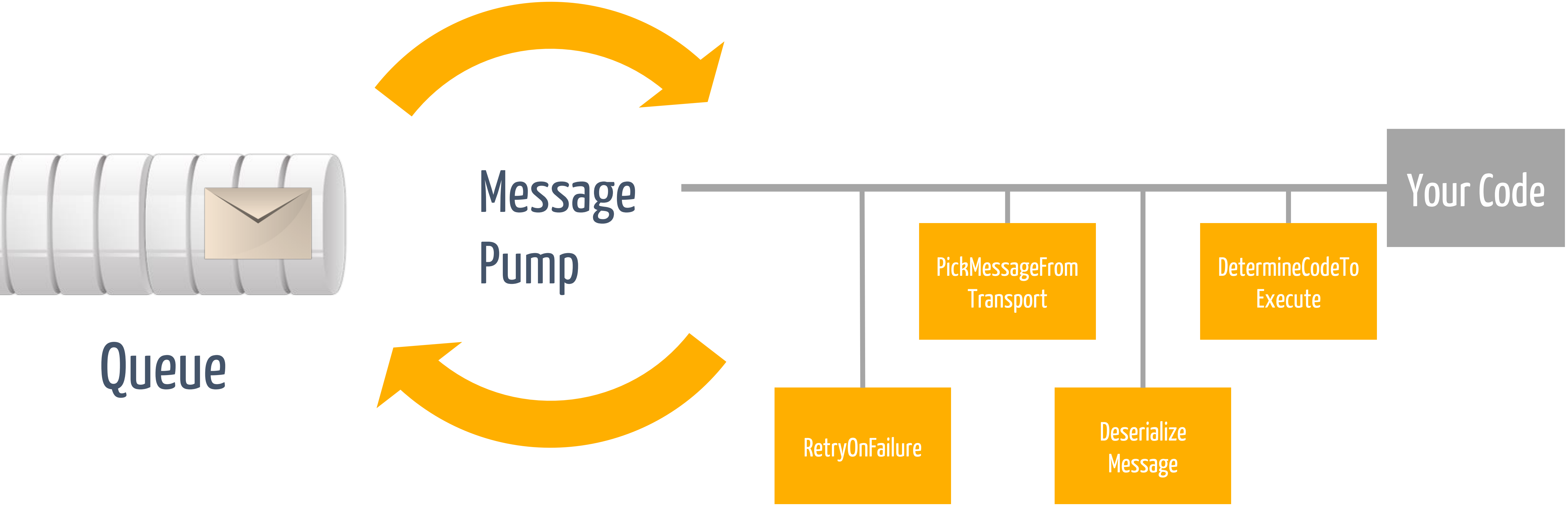
Recap

reminder

Libraries and frameworks should use ConfigureAwait(false)

Async all the way, don't mix blocking and asynchronous code

Coding time



Queue

Message Pump

Your Code

PickMessageFrom Transport

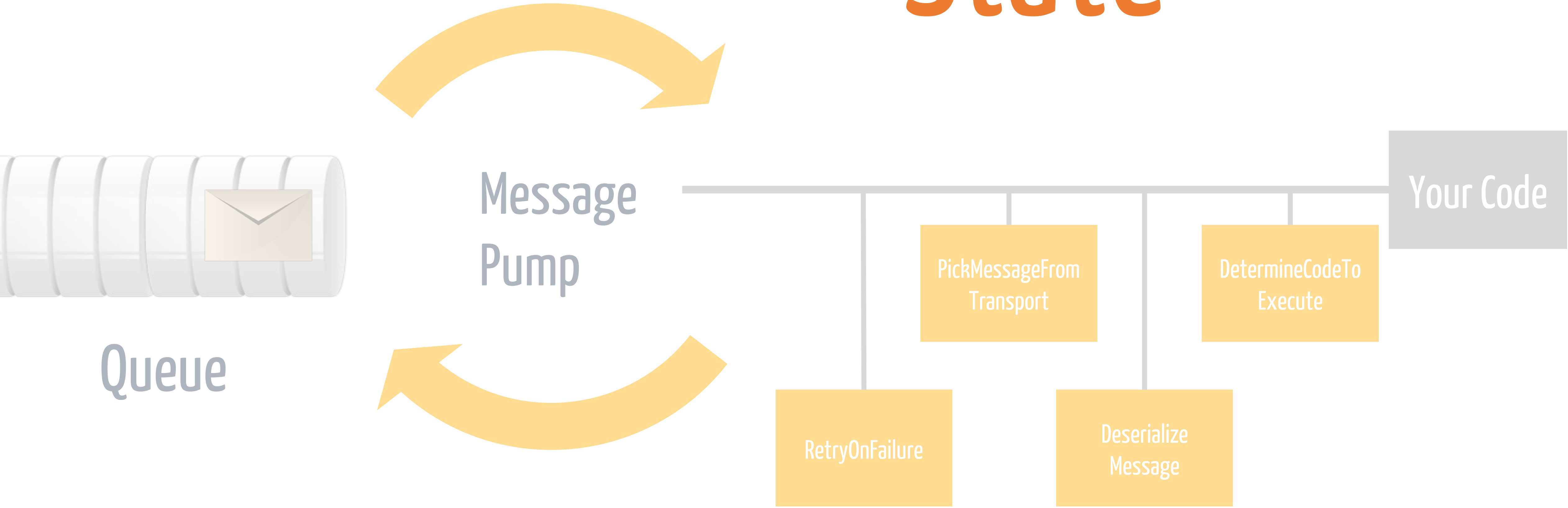
DetermineCodeTo Execute

RetryOnFailure

Deserialize Message

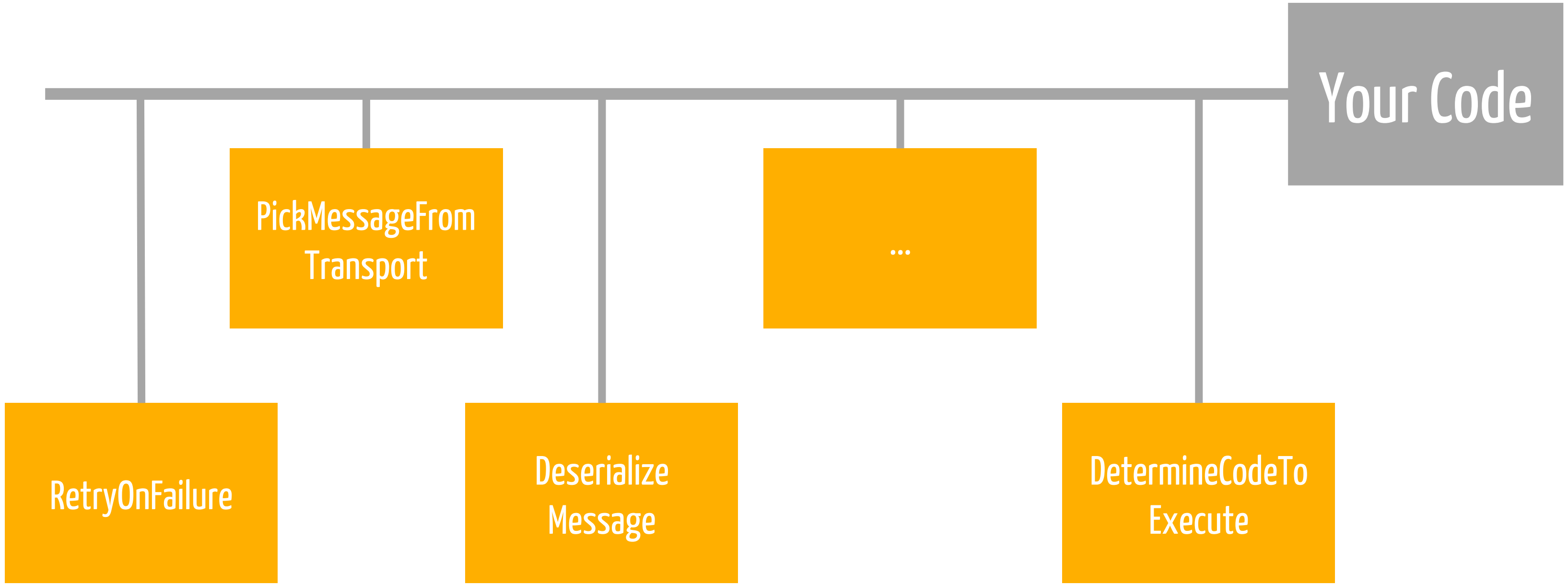
Coding time

State

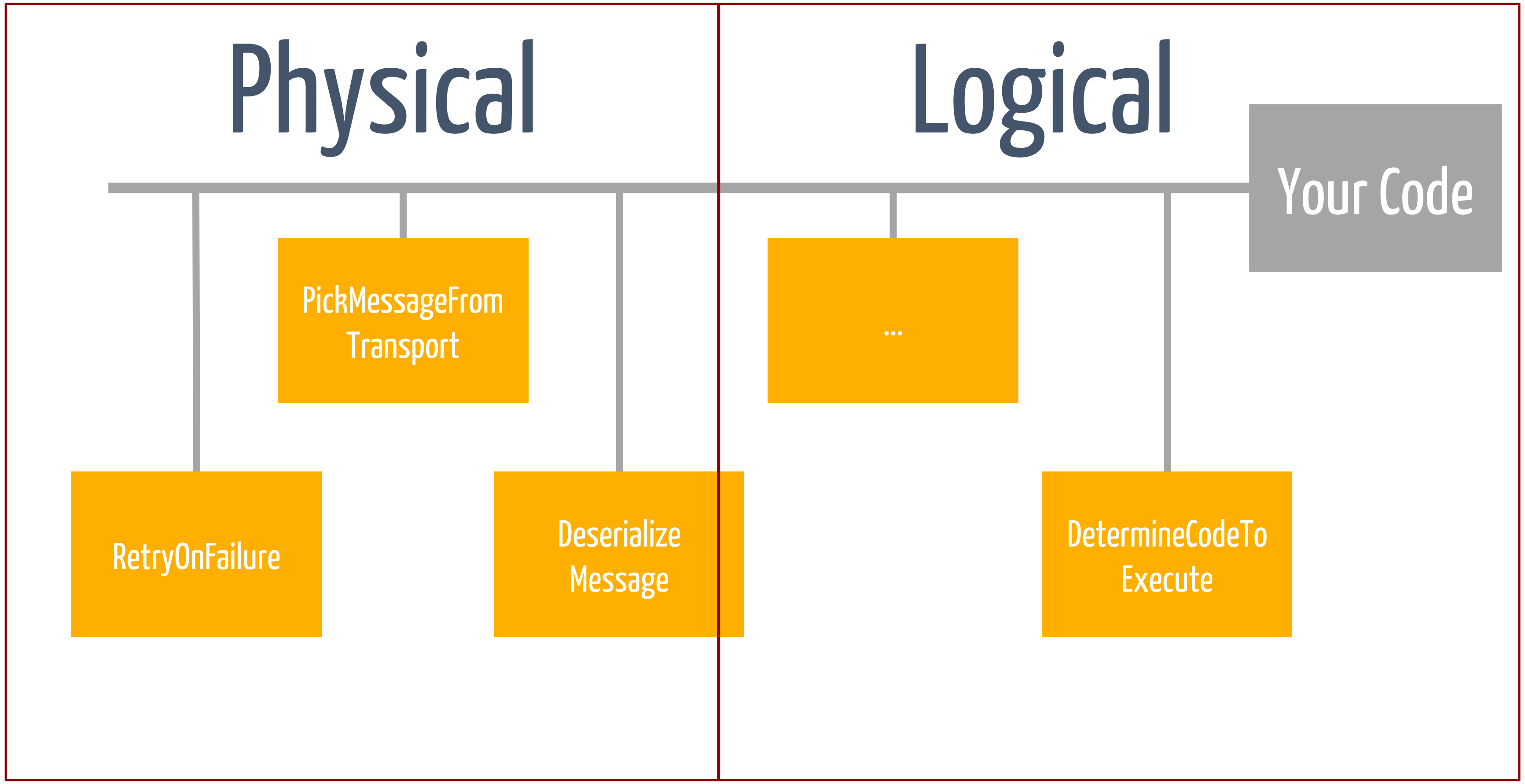


Queue
Message Pump
RetryOnFailure
PickMessageFrom Transport
Deserialize Message
DetermineCodeTo Execute
Your Code

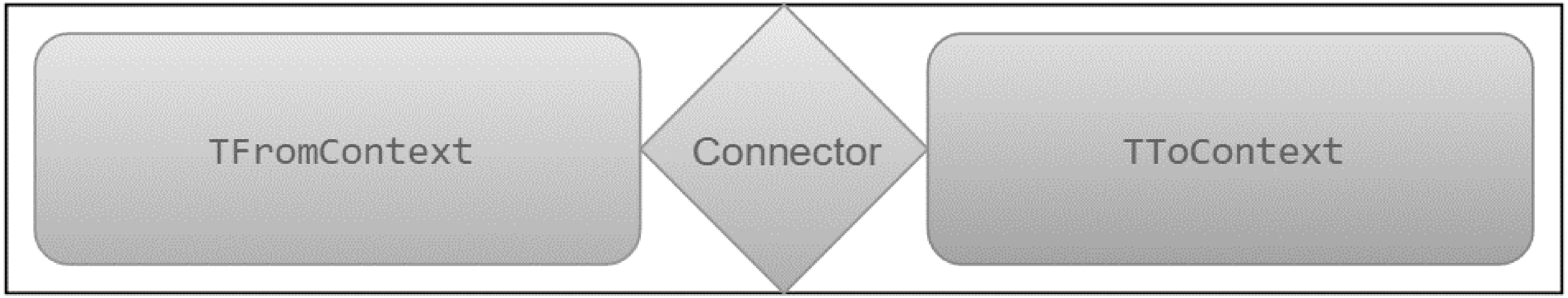
Coding time



Where to place links?

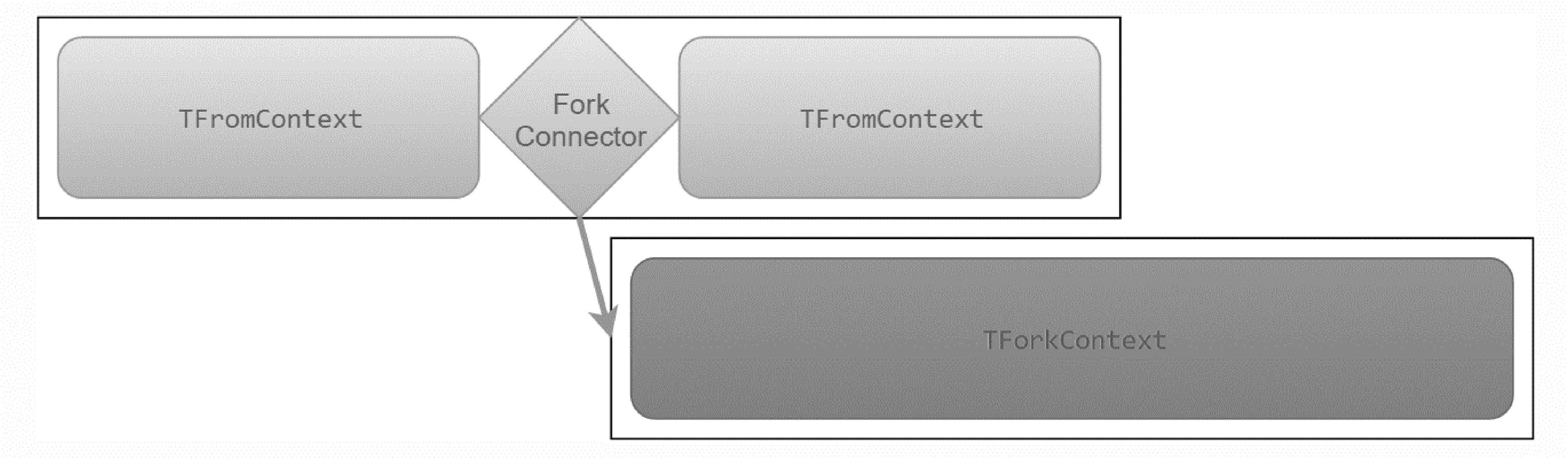

Physical
Logical
Your Code
PickMessageFrom Transport
...
RetryOnFailure
Deserialize Message
DetermineCodeTo Execute

Stages

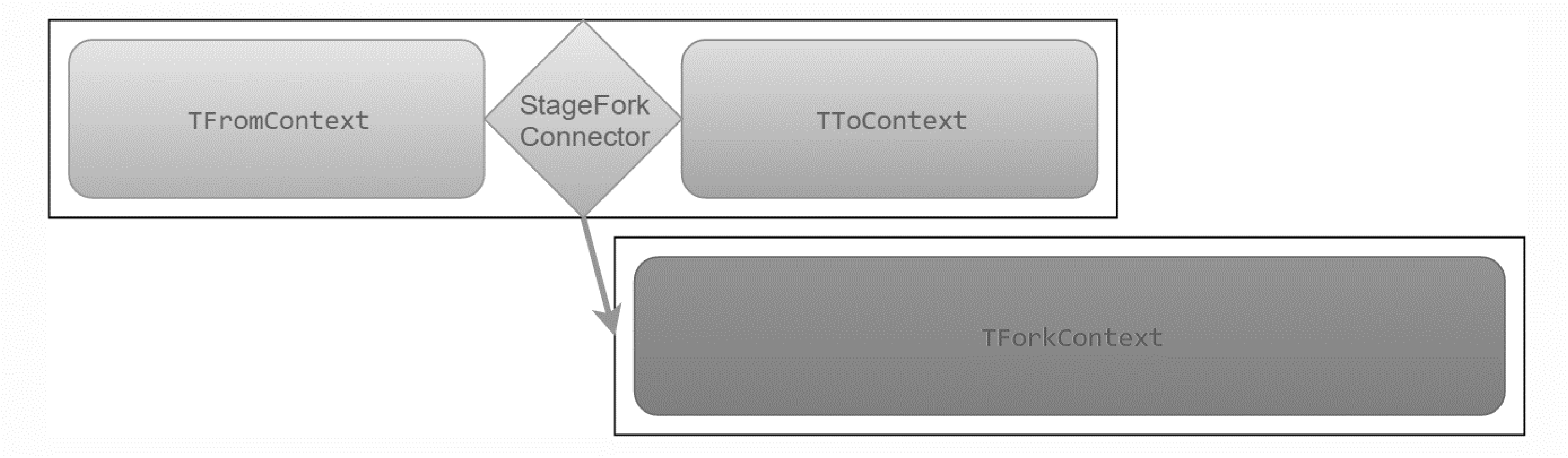


Stage Connector

await Demo



Fork Connector



Stage Fork Connector

Tree of Responsibility

Keep calm and let your head explode

Pattern

Build It

WrapUp

NSB v6

Will be Async all the way

Uses the chain of responsibility pattern heavily

particular.net/blog/async-await-its-time

docs.particular.net/nservicebus/pipeline/customizing-v6

Recap

reminder

Chain of Responsibility or Russian Dolls is a flextensible pattern ideally suited to build robust IO bound pipelines

The pattern is used in many OSS projects

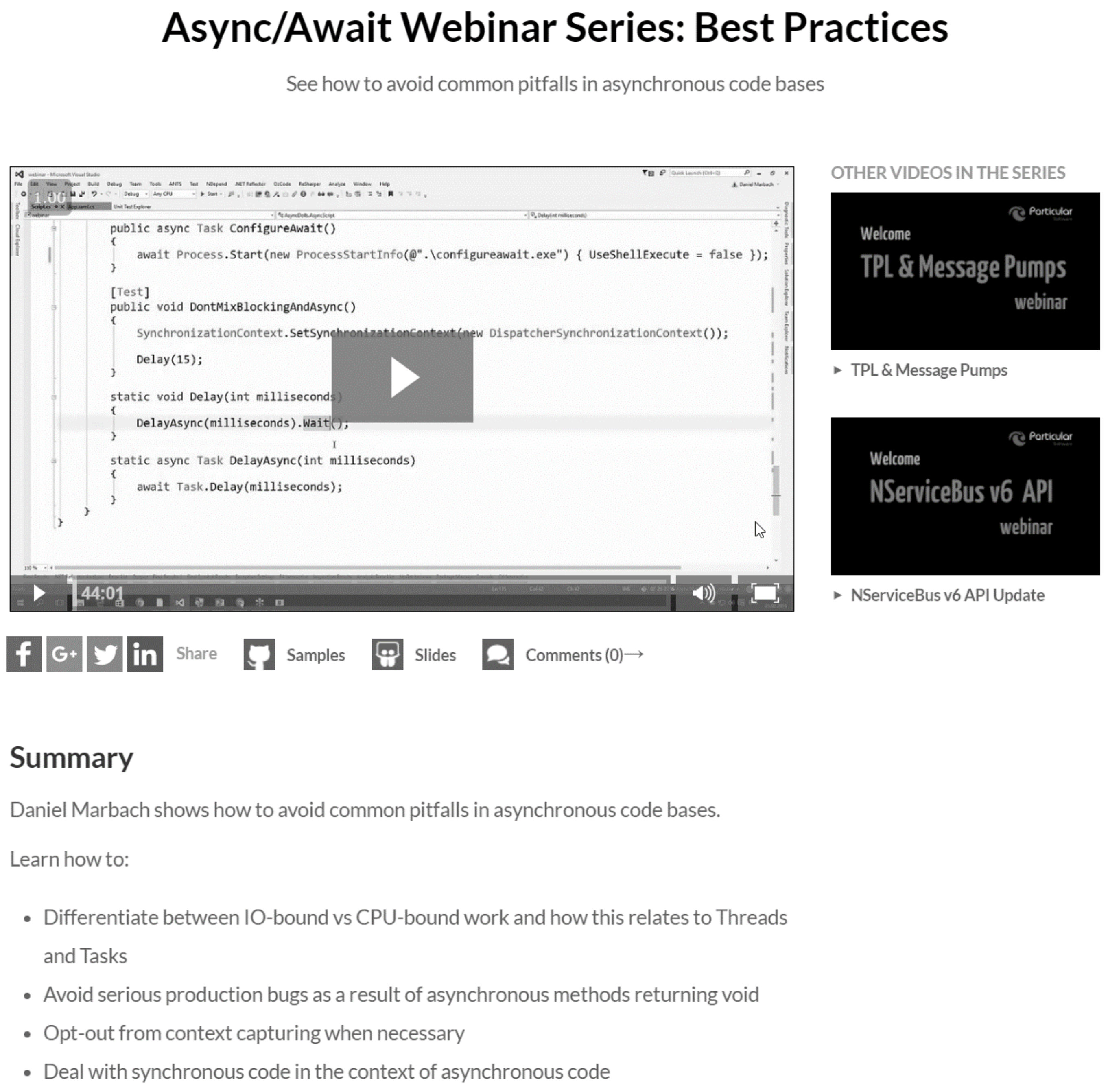
Know it, learn it, love it *

Probably copyrighted by the .NET Rocks guys ;)*

Slides, Links...

github.com/danielmarbach/async-dolls

github.com/danielmarbach/dwx16.async-chain go.particular.net/async-await



Async/Await Webinar Series: Best Practices

See how to avoid common pitfalls in asynchronous code bases

```
public async Task ConfigureAwait()
{
    await Process.Start(new ProcessStartInfo(@".\configureawait.exe") { UseShellExecute = false });
}

[Test]
public void DontMixBlockingAndAsync()
{
    SynchronizationContext.SetSynchronizationContext(new DispatcherSynchronizationContext());

    Delay(15);
}

static void Delay(int milliseconds)
{
    DelayAsync(milliseconds).Wait();
}

static async Task DelayAsync(int milliseconds)
{
    await Task.Delay(milliseconds);
}
```

OTHER VIDEOS IN THE SERIES

▸ TPL & Message Pumps

▸ NServiceBus v6 API Update

Summary

Daniel Marbach shows how to avoid common pitfalls in asynchronous code bases.

Learn how to:

- Differentiate between IO-bound vs CPU-bound work and how this relates to Threads and Tasks
- Avoid serious production bugs as a result of asynchronous methods returning void
- Opt-out from context capturing when necessary
- Deal with synchronous code in the context of asynchronous code

Particular Software

await Q & A

Particular
Software

Thanks