Advanced Hosted Service Concepts

Steve Gordon

MICROSOFT DEVELOPER TECHNOLOGIES MVP

@stevejgordon   www.stevejgordon.co.uk

Overview

Handling exceptions

Triggering application shutdown

Ordering background services

Overriding StartAsync and StopAsync

Testing worker services

BackgroundService Implementation Details

Demo

Adding exception handling

Why exception is handling is important

Demo

How lifetime is managed by the host

How the shutdown process works

Triggering shutdown

Configuring shutdown options

IHostApplicationLifetime

Allows consumers to be notified of application lifetime events.

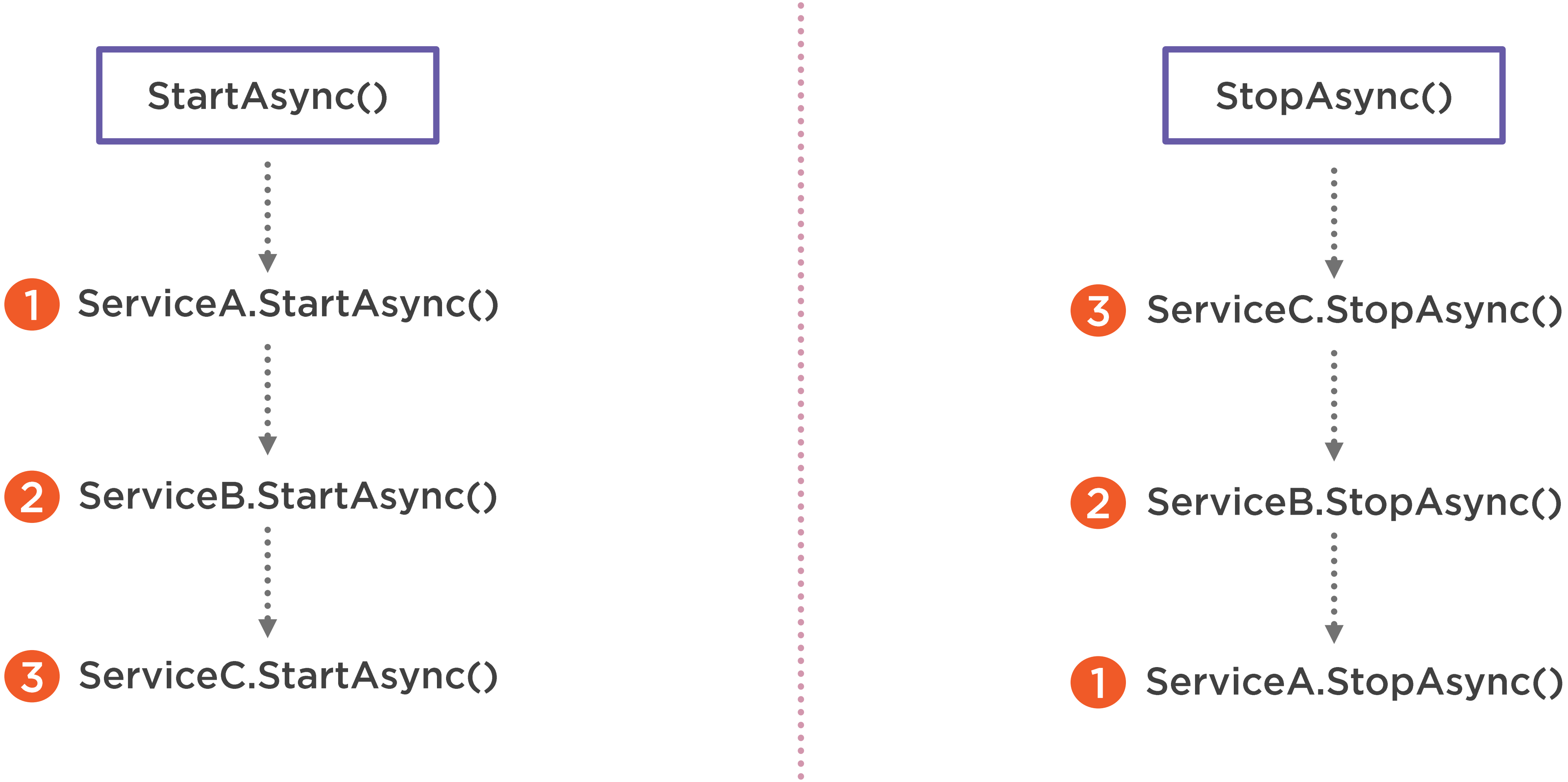
IHostApplicationLifetime.cs

```
public interface IHostApplicationLifetime
{
    CancellationToken ApplicationStarted { get; }
    CancellationToken ApplicationStopping { get; }
    CancellationToken ApplicationStopped { get; }
    void StopApplication();
}
```

Registration Order of Background Services

The registration order of background services can be important

Starting and Stopping Background Services



StartAsync()

1 ServiceA.StartAsync()

2 ServiceB.StartAsync()

3 ServiceC.StartAsync()

StopAsync()

3 ServiceC.StopAsync()

2 ServiceB.StopAsync()

1 ServiceA.StopAsync()

Configuring the Host

Host.CreateDefaultBuilder()

HostBuilder Defaults

Loads application configuration

appsettings.json

appsettings.{Environment}.json

Secret Manager (when the app runs in the development environment)

Environment variables

Command-line arguments

Adds logging providers

Console

Debug

EventSource

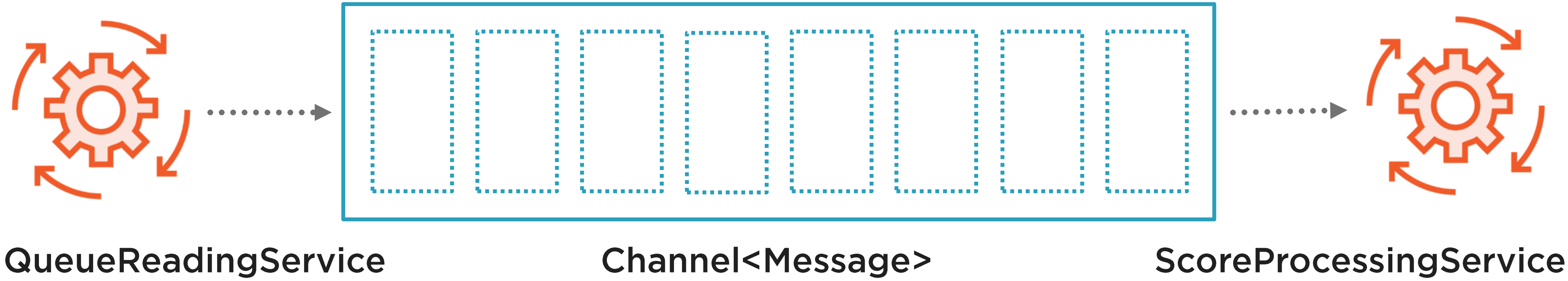
EventLog (only when running on Windows)

Program.cs

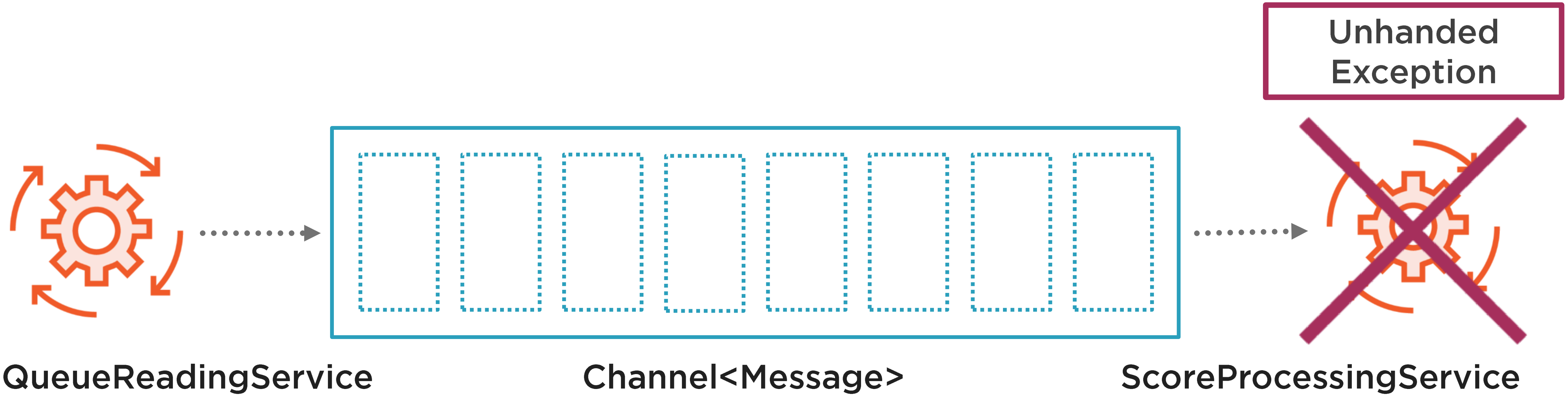
```
IHost host = Host.CreateDefaultBuilder()
                 .ConfigureServices(services =>
                 {
                     services.AddSingleton<ISomething, Something>();
                 })
                 .ConfigureAppConfiguration(config =>
                 {
                     config.AddInMemoryCollection(new[]
                     {
                         new KeyValuePair<string, string>("Key", "Value")
                     });
                 })
                 .ConfigureLogging(logging =>
                 {
                     logging.AddConsole(opt => opt.IncludeScopes = true);
                 })
                 .Build()
```

It’s safe to call the StopApplication method multiple times

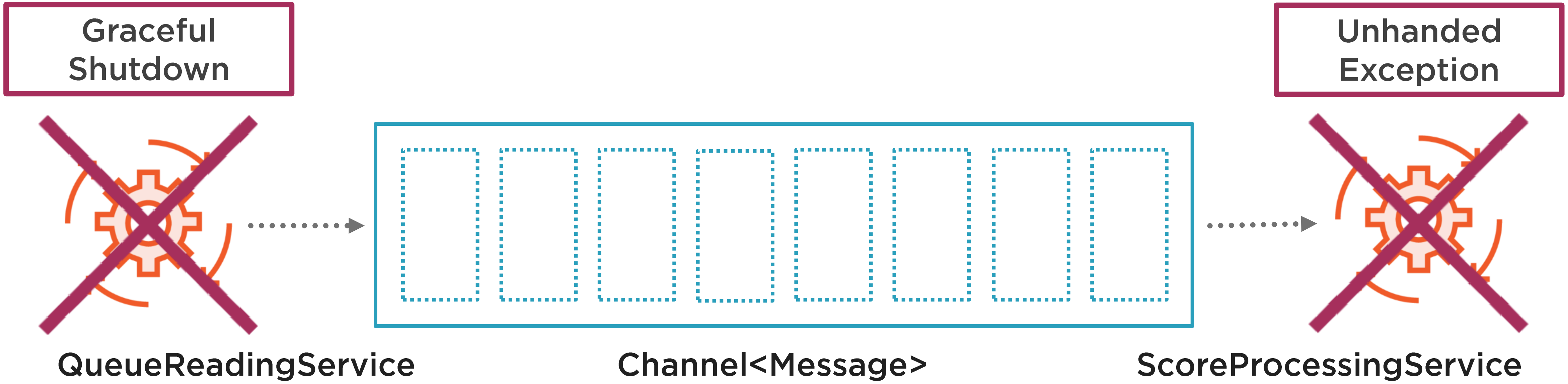
Failing to Call StopApplication



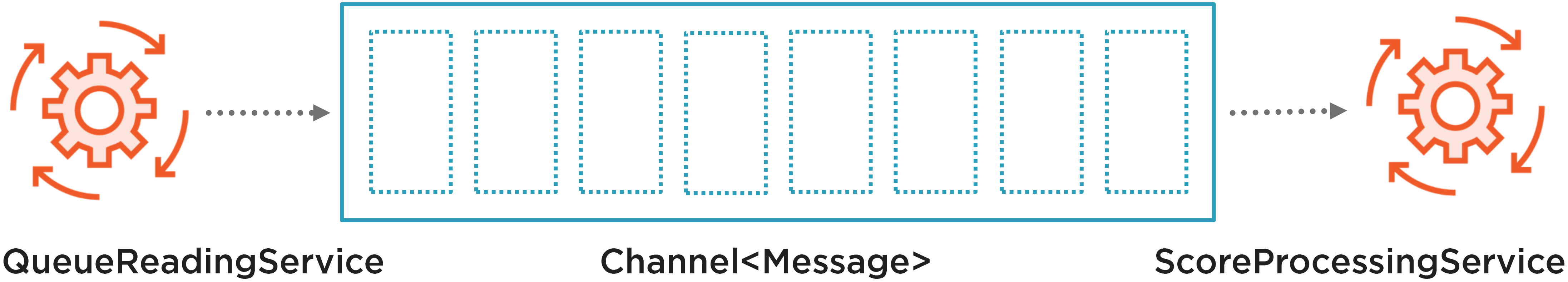
Failing to Call StopApplication



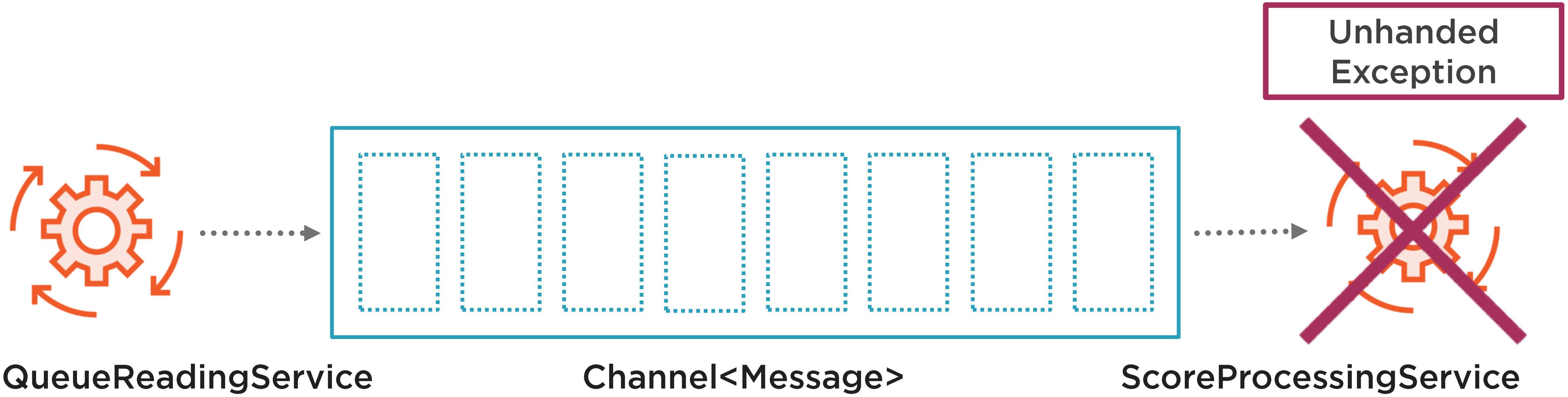
Failing to Call StopApplication



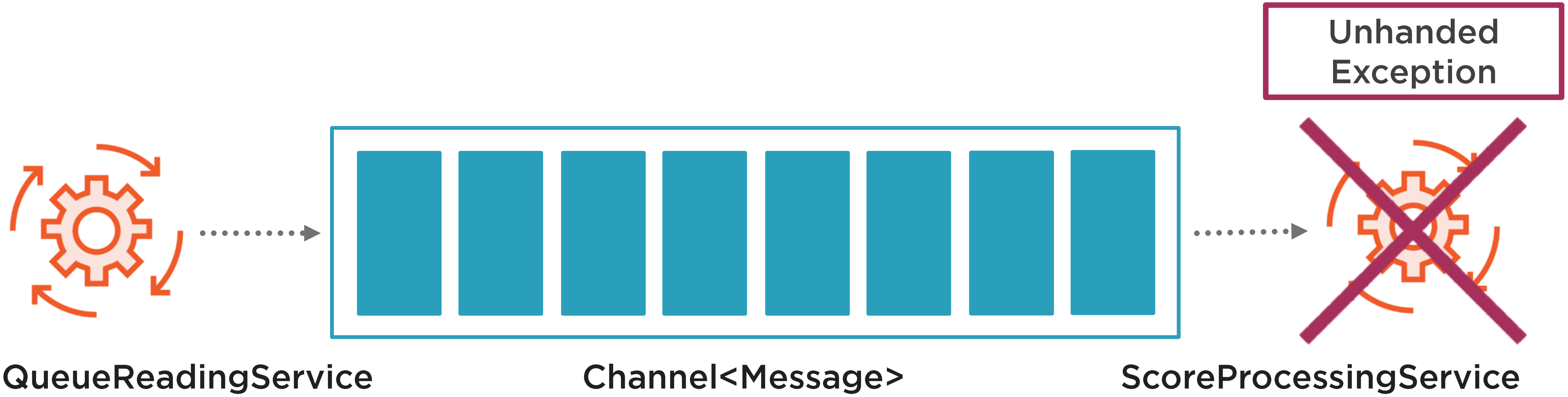
Failing to Call StopApplication



Failing to Call StopApplication



Failing to Call StopApplication



Unhanded Exception

QueueReadingService

Channel<Message>

ScoreProcessingService

Demo

Overriding StartAsync and StopAsync

- Adding timing logging to StopAsync

Demo

Unit testing background services

Avoiding Blocking Code in StartAsync

StartAsync blocks application startup

StartAsync should complete quickly

Avoid long-running work in StartAsync

ExecuteAsync also blocks startup until the first await

Awaiting an immediately completed Task is effectively synchronous

Summary

Handling exceptions

Triggering and handling shutdown

Registration order of hosted services

Overriding StartAsync and StopAsync

Unit testing hosted services

Up Next:
Running Worker Services in Production