# Building Concurrent Applications with the Actor Model in Akka.NET

## Introducing Actor Models and Akka.NET

Jason Roberts

@robertsjason | dontcodetired.com

# Overview

- Why use Actor Models
- Classes of applications
- Akka.NET history and Reactive Manifesto
- Understanding actors and messages
- Actor systems and fault tolerance
- Akka.NET NuGet packages
- Getting started in Visual Studio

Simplify the building of scalable, concurrent, high-throughput and low latency systems

# Why Use Actor Models?

No manual thread management

Higher abstraction level

Scale up

Scale out

Fault tolerance and handling

Common single architecture

# Classes of Applications

**Transactional**

- Financial & statistical
- Betting / gambling
- Social media
- Telecoms

**Batch**

- Divide workloads between actors

**General Service**

- REST, SOAP
- System integration

# Classes of Applications

| | |
|---|---|
| Communications | • Chat applications<br>• Real time notifications |
| Multiplayer Gaming | • Manage players / interactions |
| Traffic Management | • Road traffic flow<br>• Asset management / location |
| Numerical Processing | • Business intelligence<br>• Data mining |
| Internet of Things | • Incoming streams of data (sensor data) |

# Using Akka.NET in Different .NET Application Types

Services backend (Web API, WCF, etc)

Web app backend (MVC, Web Forms)

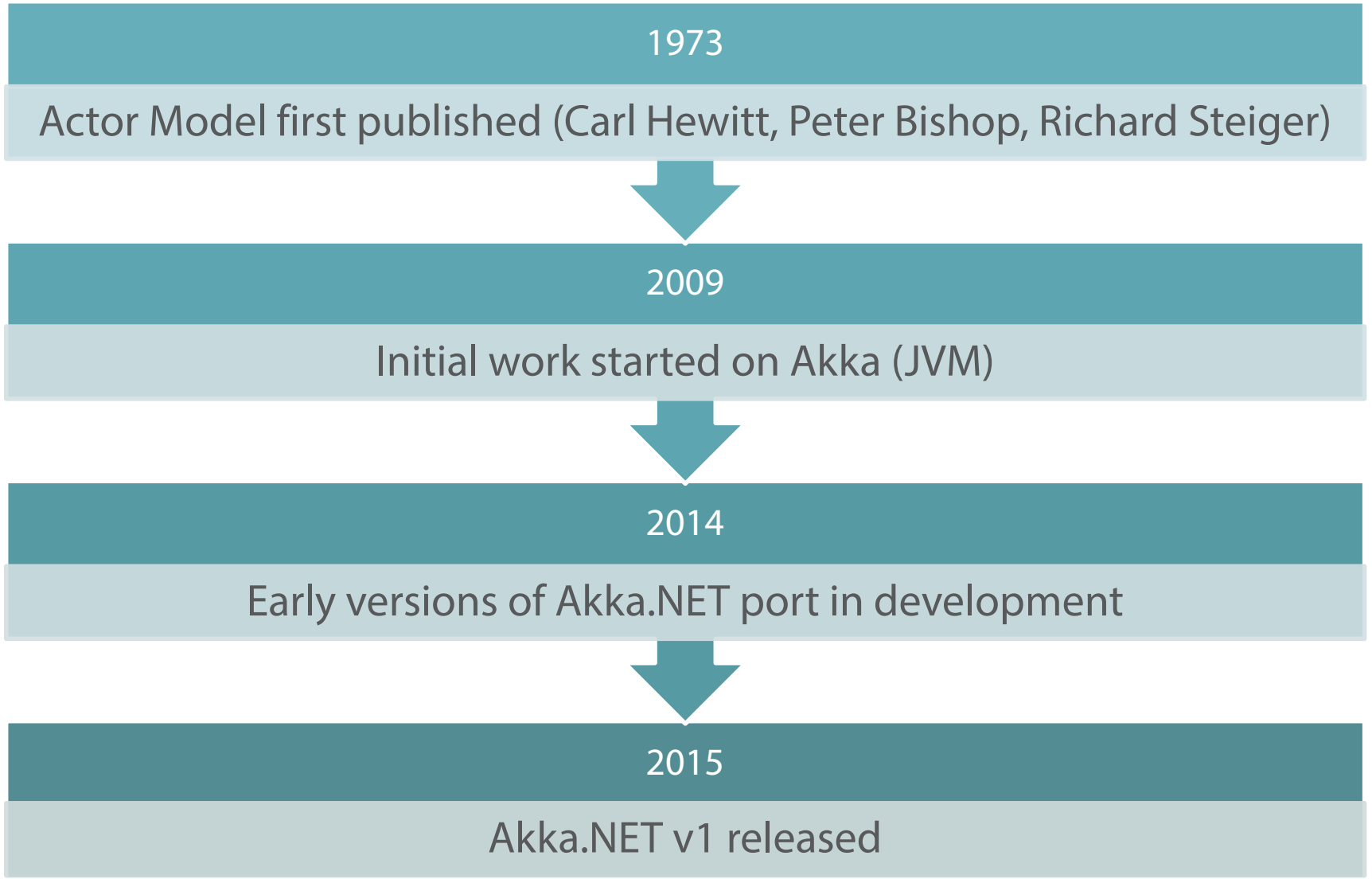
Windows Service

Console application

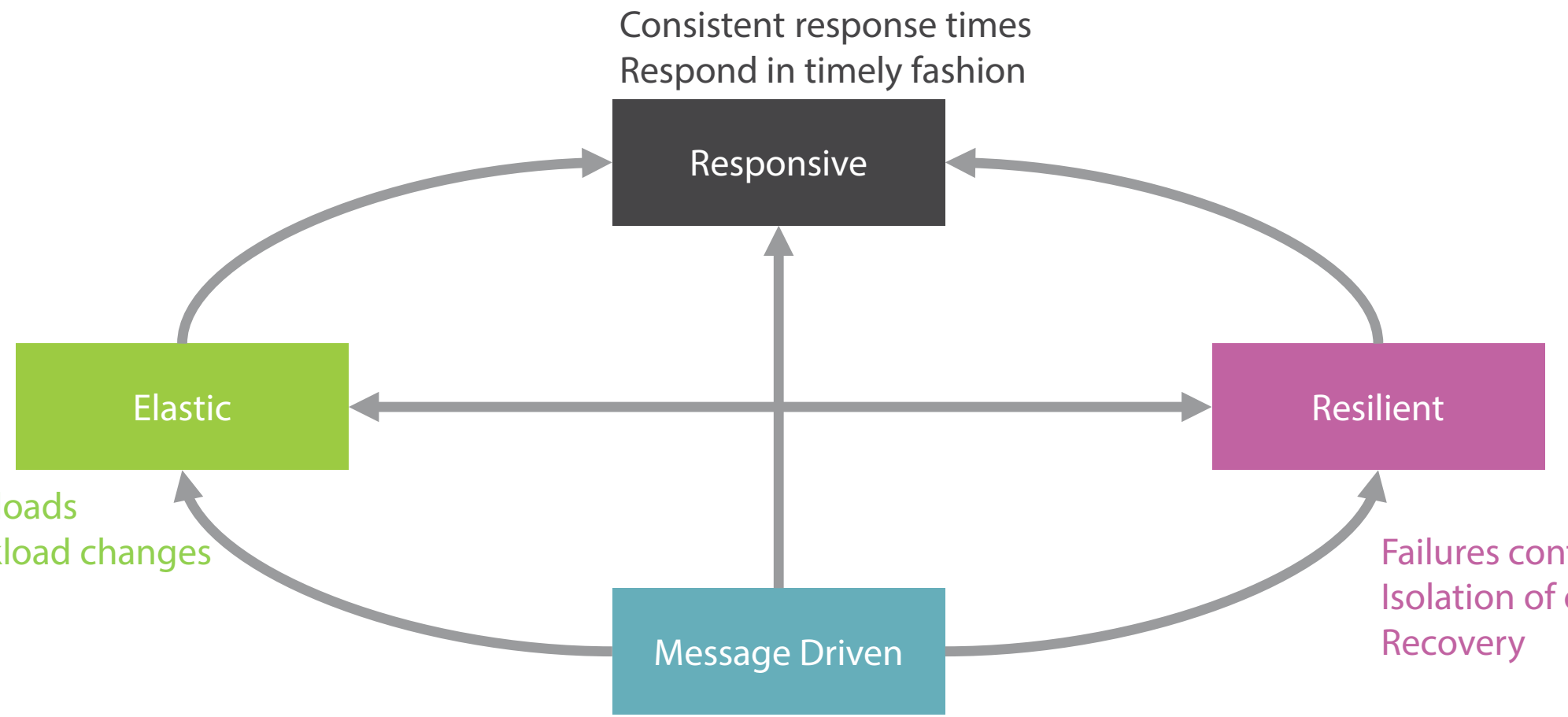
WinForms application

WPF application

# A Brief History of Akka.NET



# The Reactive Manifesto




www.reactivemanifesto.org

# Key Features of Akka.NET

**Simple Concurrency**

- High level abstractions (actors, messages, FSMs)
- Asynchronous

**Simple Distribution**

- Location transparency
- Remote deployment using configuration

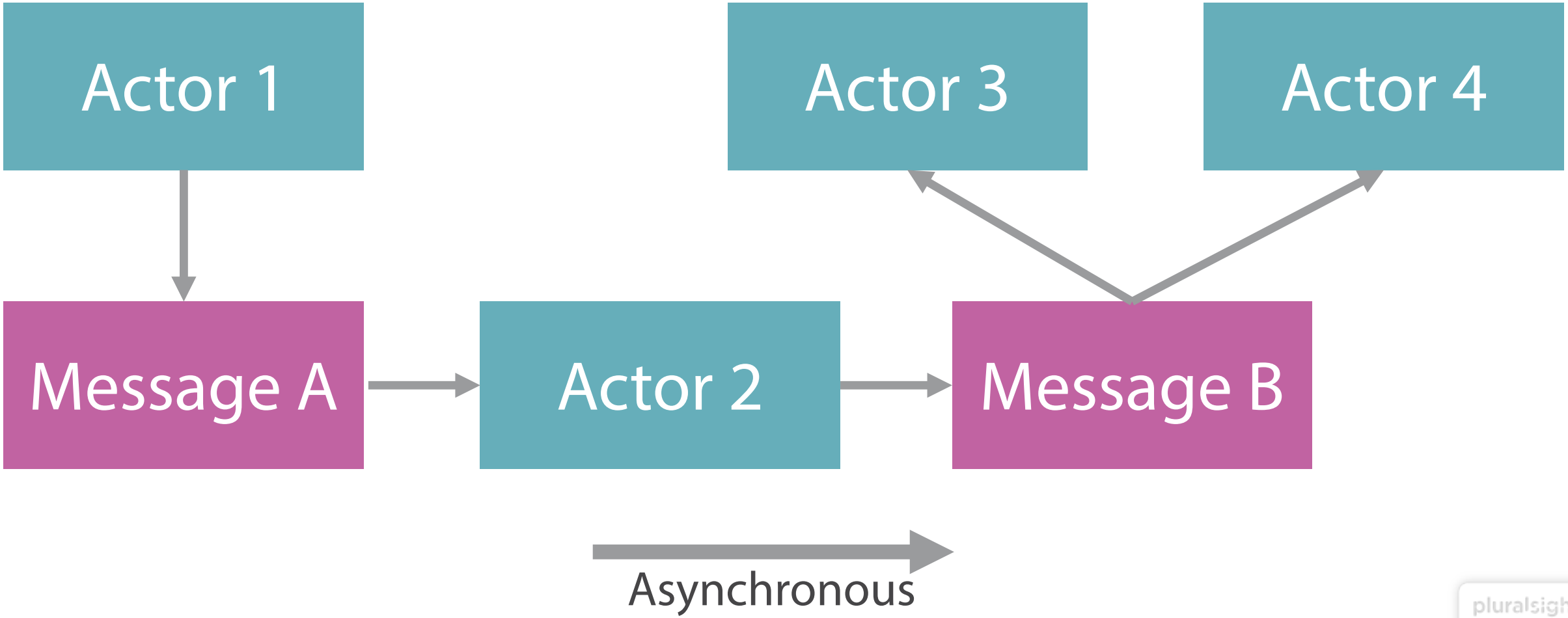
**High Performance**

- ~50 million messages per second per machine
- 1GB heap memory = ~2.5 million actors
- Load balancing / routing

**Resilient**

- Self-healing
- Supervisory hierarchies and fault isolation

# Actors and Messages



# Actors

"Everything is an Actor"

Fundamental primitive computational unit

Perform small well-defined tasks

Big piece of work broken down into smaller pieces

Actor code is the same whether it's local or distributed

Every actor instance has an address

Actors communicate via messages

Actor instances are lazy

~300 bytes overhead per instance

# Four Things an Actor Can Do

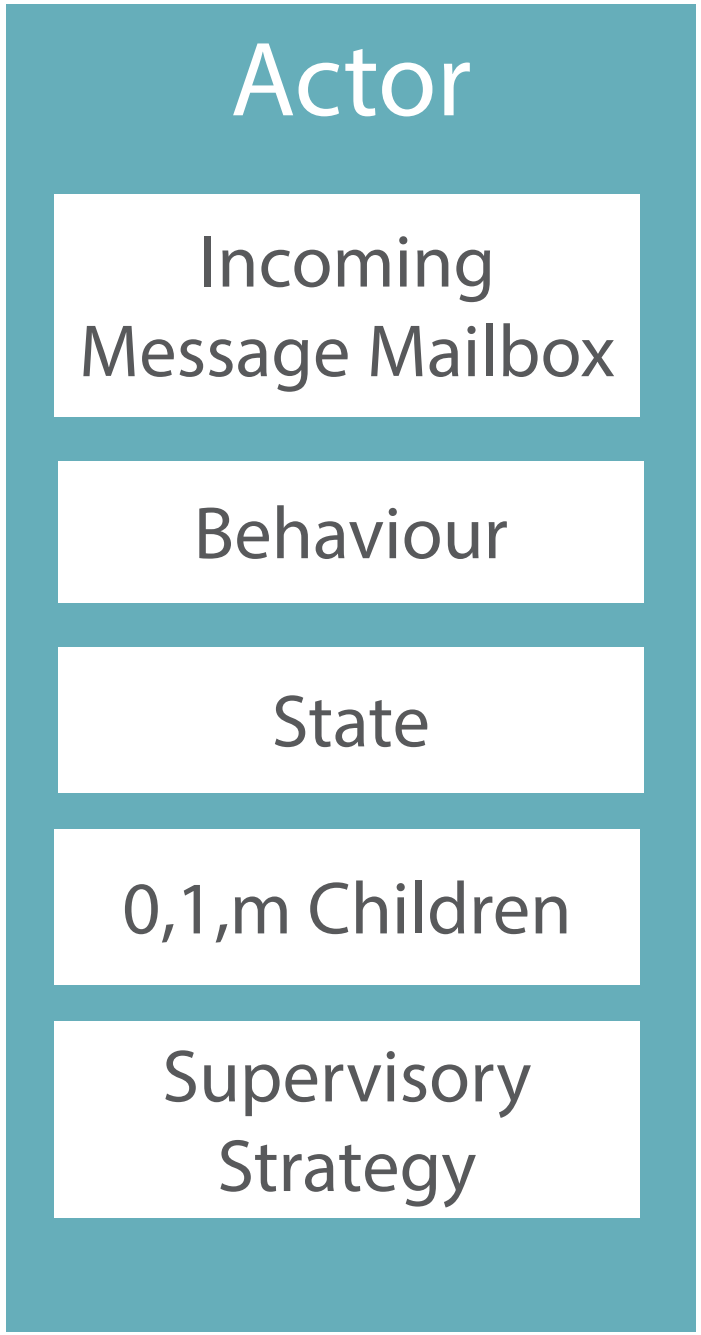
Receive & react to messages

Change behaviour for next message

Actor

Create more actors

Send messages to other actors




Messages sent to the actor get queued here for processing

React to an incoming message and perform defined action

The current state of the actor (user ID, request counter, etc.)

Other child actors that are being supervised

How to handle faults in children if they occur

An actor processes one message at a time

Message

Simple POCO class

Actors can change state when responding to messages

Message instances should be immutable

The passing of messages is asynchronous

A message is an item of data that is sent to a specific destination… In a message-driven system addressable recipients await the arrival of messages and react to them, otherwise lying dormant.

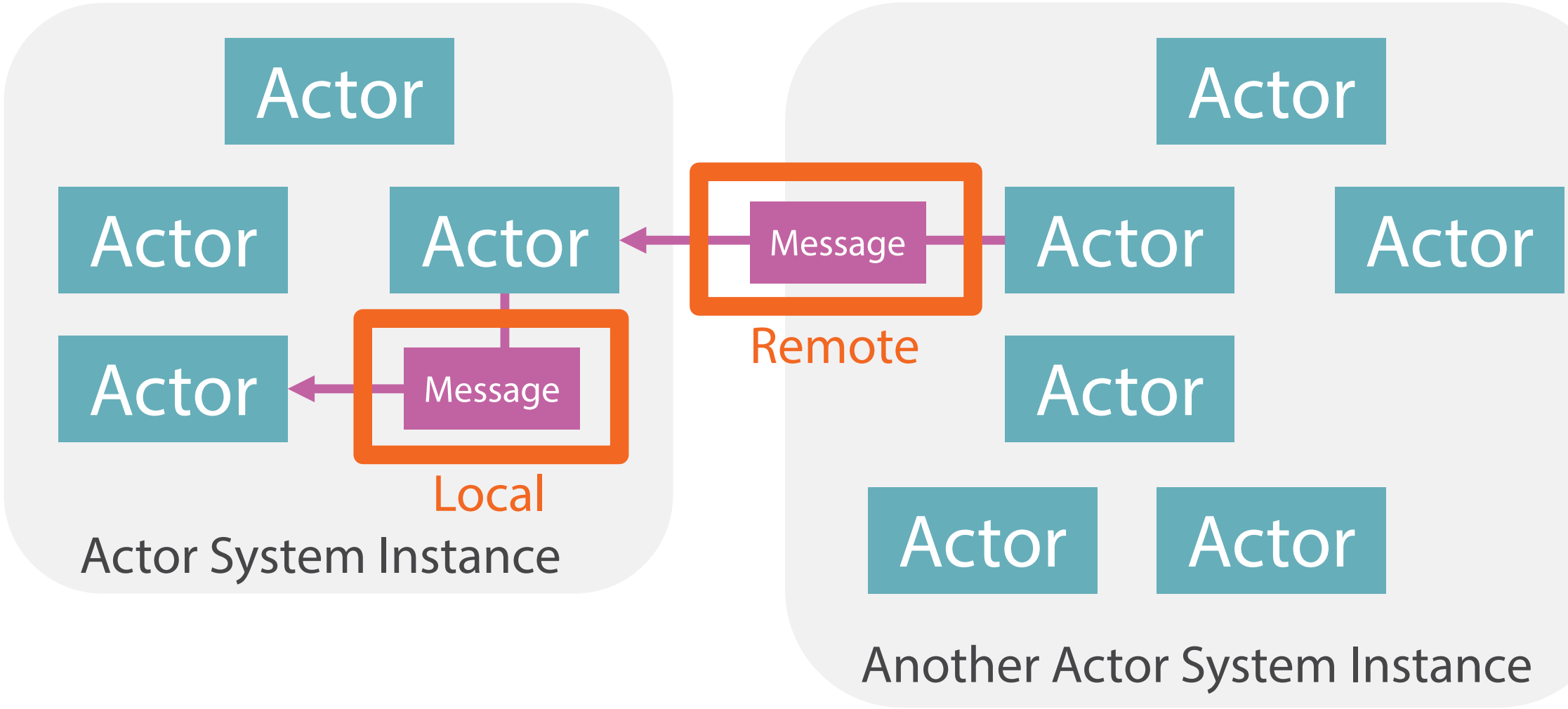
— The Reactive Manifesto

# An Example of a Message

```
public class ExampleMessage
{
    public ExampleMessage(int customerId)
    {
        CustomerId = customerId;
    }

    public int CustomerId { get; private set; }
}
```

# Actor Systems



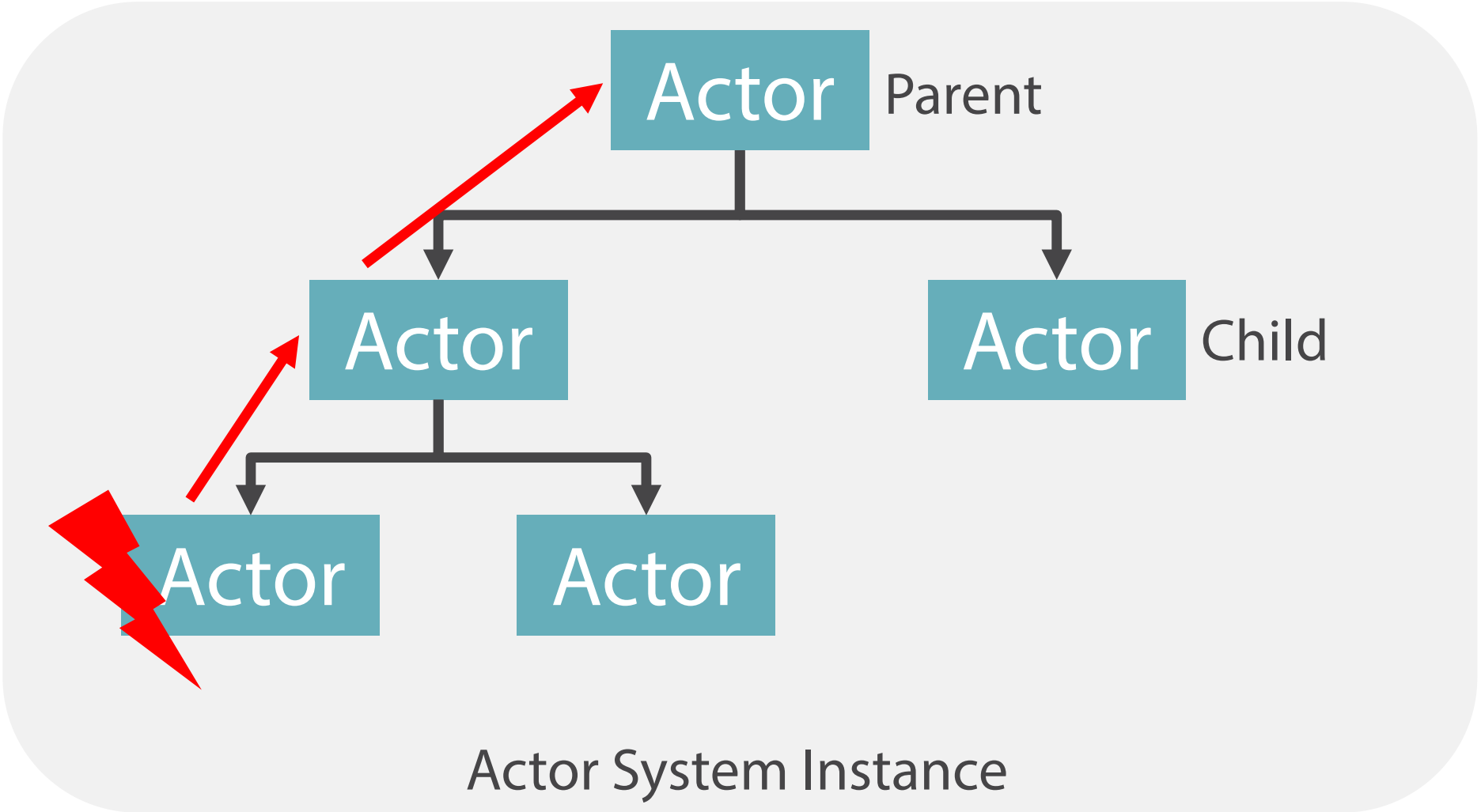
## Location Transparency

The ability to send a message to another actor without needing to know where it is; whether it be in the same actor system instance on the same computer, or another actor system instance on the other side of the world.

# Location Transparency

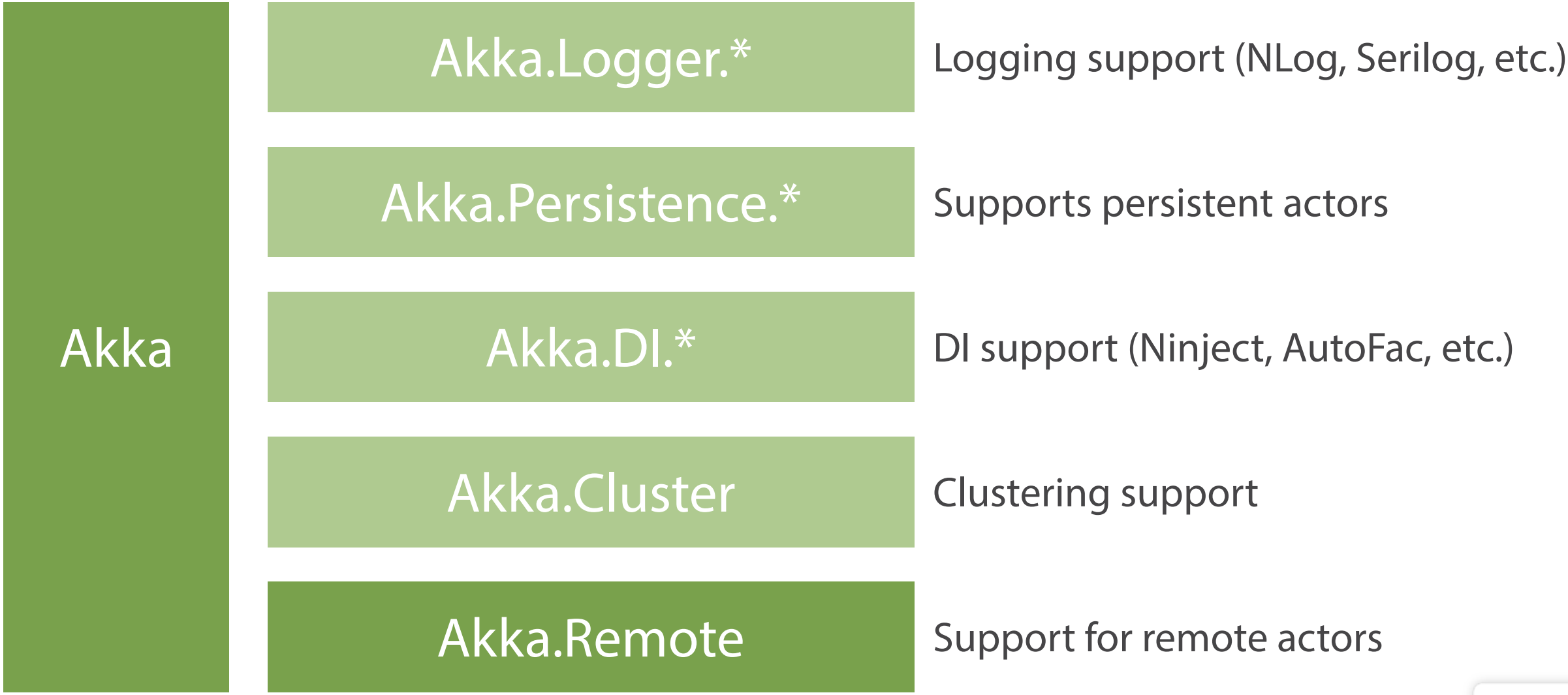
Phone Number

# Overview of Actor Supervision Hierarchies



Supervision hierarchies allow the system to not only deal with faults but become self-healing

# Akka.NET NuGet Packages



# Getting Started in Visual Studio

Start with empty console application

Install the Akka.NET NuGet package

Create an actor system instance

# Course Outline

- Now: Introducing Actor Models and Akka.NET
- Next: Defining and Using Actors and Messages
- Understanding Actor Lifecycles and States
- Creating Actor Hierarchies and Isolating Faults
- Deploying and Messaging Remote Actors

# Other Actor Model Frameworks and Libraries

- Microsoft Project Orleans
  - Designed for use in the cloud (also run on-premise)
  - Optimized for Microsoft Azure
  - Raised abstraction level of the Actor Model
  - Requires Orleans SDK to be installed for development
  - Uses some code generation behind the scenes
  - Used to build Halo 4 cloud services
  - dotnet.github.io/Orleans
  - "Introduction to Microsoft Orleans" Pluralsight course (by Richard Astbury)

# Other Actor Model Frameworks and Libraries

- Other .NET Actor Model Libraries
  - Stact - github.com/phatboyg/stact
  - NAct - code.google.com/p/n-act/
  - Remact.Net - github.com/steforster/Remact.Net

# Summary

- Why use Actor Models
- Classes of applications
- Akka.NET history and Reactive Manifesto
- Understanding actors and messages
- Actor systems and fault tolerance
- Akka.NET NuGet packages
- Getting started in Visual Studio

Next:

Defining and Using Actors and Messages