AKKA.NET 101

A BRIEF INTRODUCTION TO ACTOR PROGRAMMING IN .NET USING AKKA.NET

ABOUT ME

Herdy Handoko

- Senior Analyst, Information Systems @ Rio Tinto
- Occasional open-source contributor
  - ServiceStack LightSpeed ORM adapter
  - ServiceStack Yammer OAuth
  - Yammer .NET API wrapper
- Working in .NET (C#), Scala, Java, with some Python knowledge.

 _hhandoko

 hhandoko

WHAT WILL BE COVERED

Actors and Actor programming model overview

Akka(.NET) overview and fundamentals:

1. Creating actors
2. Sending and handling messages
3. Actor hierarchy and supervision strategy
4. Behaviour and hotswap

TARGET AUDIENCE

.NET developers of all levels of experience

Related concepts:

- Asynchrony
- Concurrency
- Message Queue
- Parallelism

WHAT HAS CHANGED?

Multi-core computing[1]:

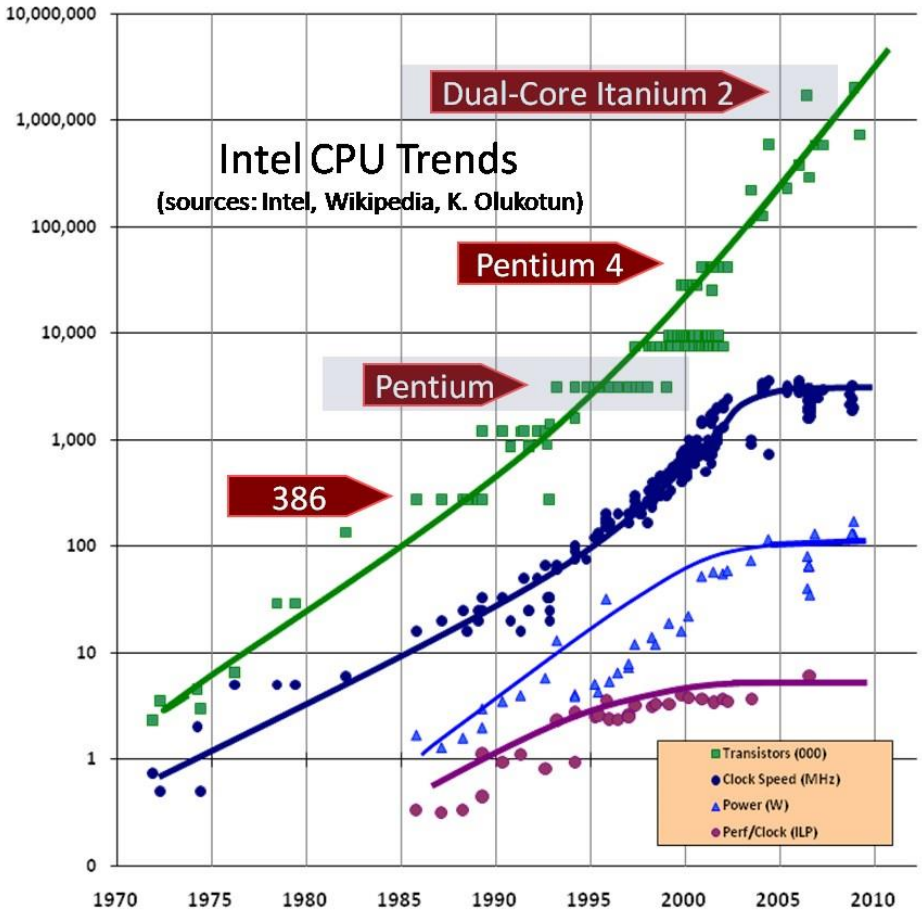
CPU speed increases (GHz) have plateaued, single-threaded performance increases is slowing down[2]

CPU development focus shifted on more cores with better efficiency

Cloud computing[1,3]:

Design for failure

Design to scale horizontally



Notes:

[1] – Hewitt 2010, p.15.

[2] – http://preshing.com/20120208/a-look-back-at-single-threaded-cpu-performance/

[3] – Mackenzie 2014, p.7.

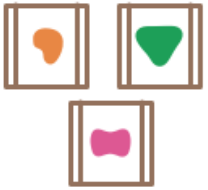
Images:

[1] – http://www.extremetech.com/wp-content/uploads/2012/02/CPU-Scaling.jpg

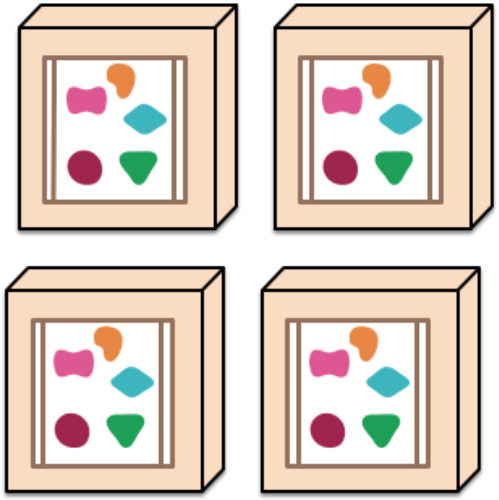
WHAT HAS CHANGED?

A monolithic application puts all its functionality into a single process...

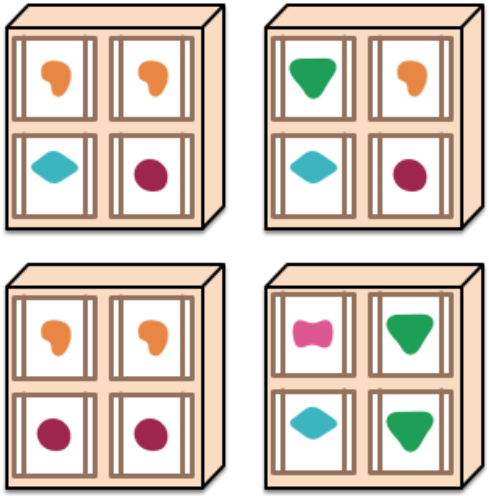
A microservices architecture puts each element of functionality into a separate service...



... and scales by replicating the monolith on multiple servers



... and scales by distributing these services across servers, replicating as needed.



Microservices!

Images:
[1] – http://martinfowler.com/articles/microservices/images/sketch.png

HOW CAN WE ADAPT?

Build applications with parallelism and concurrency to take advantage of multi-core computing

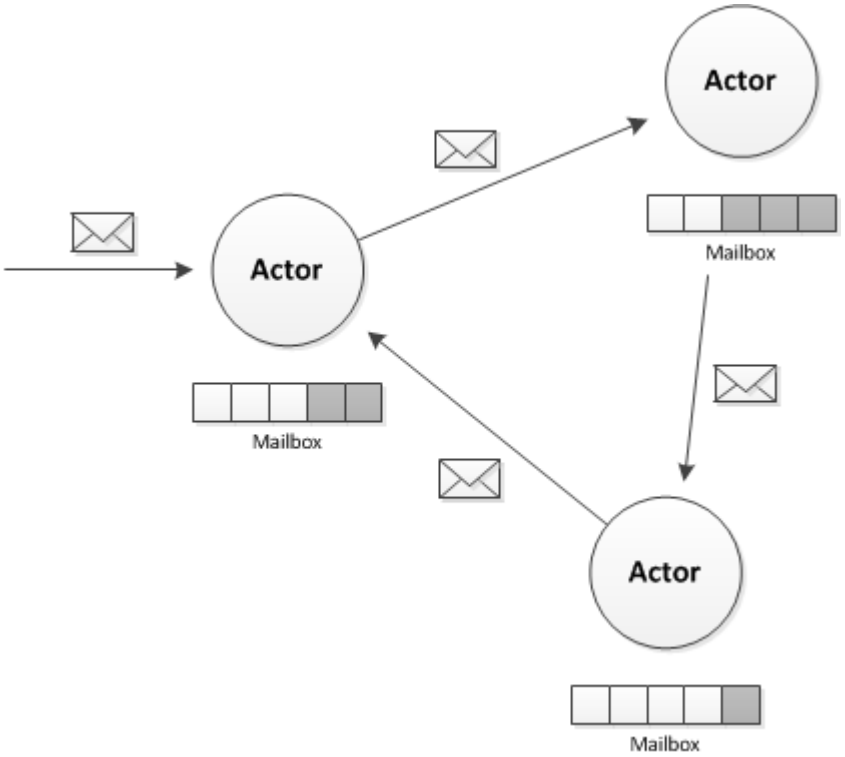
Build applications that is resilient in the face of failure[1]:

- Isolate component from each other
- Contain failures within each component
- Ensure parts of the system can fail and recover without compromising the system as a whole

Notes:
[1] – http://www.reactivemanifesto.org/

WHAT ARE ACTORS?

Actors are self-contained, interactive, independent components of a computing system that communicate by asynchronous message passing[1].



Notes:
[1] – Agha, Houck & Panwar 1992, p.2.

WHY THE ACTOR MODEL?

The bulk of our tools rely on Shared State concurrency. They perform really well, but hard to do properly:

- deadlocks
- race conditions
- memory corruptions
- resource contentions
- etc.

Actors model provides a simplification / abstraction layer. It is developed as an inherently concurrent model, based on Message Passing concurrency.

Notes:
[1] – Wyatt 2013, pp.2 - 3.

KEY PRINCIPLES

Actors react to received message by executing a behaviour / function[1,2]:

- Actors can modify its internal state, but do not share state with other actors
- Actors have a well-defined life cycle
- Actors can exchange data by sending **immutable** messages, asynchronously
- Actors send messages to addresses, not directly to actors

Notes:
[1] – Bereznitsky 2010, pp.22 - 24.
[2] – http://petabridge.com/blog/akkadotnet-what-is-an-actor/

KEY PRINCIPLES, CONT'D

Messages are buffered in an actor's mailbox[1,2]:

- Mailbox is a queue with multiple producers and a single consumer, also known as a channel

- Actors process only one message at a time

Notes:
[1] – Bereznitsky 2010, p.25.
[2] – http://petabridge.com/blog/akkadotnet-what-is-an-actor/

AKKA.NET

HTTP://GETAKKA.NET/

What is Akka.NET?

- A community-driven .NET port of Java / Scala Akka Actor framework.

- Open source code (Apache 2.0), available on GitHub: https://github.com/akkadotnet/akka.net

Notes:
[1] – https://github.com/akkadotnet/akka.net/milestones

DEMO / TUTORIAL

PART 1:
CREATING ACTORS

Goals:

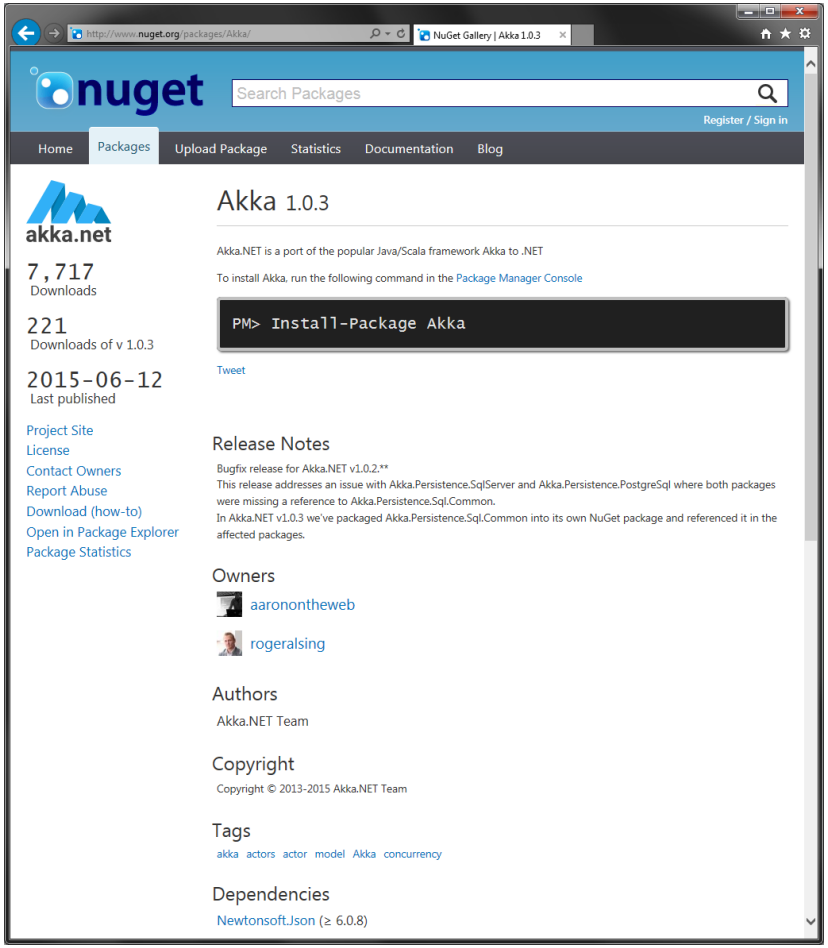
Understand how to create Actors in an ActorSystem.

Steps:

1. Install Akka.NET from NuGet
2. Setup an actor system
3. Setup an actor
4. Send message to an actor

Extras:

- Different Actor types
- Actor's properties and context



PART 2:

SENDING AND HANDLING MESSAGES

Goals:

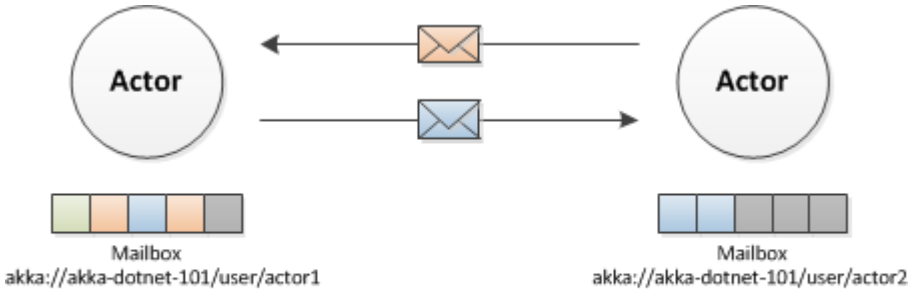
Understand how Actors pass information between one another.

Steps:

1. Setup a second actor
2. Send messages (`tell`)

Extras:

- Message path
- Message ‘Ask’ pattern
- Handler priority
- Handler predicates
- Handler switching



akka://<actor system>/user/<top level actor(s)>/<child actor(s)>

PART 3:

ACTOR HIERARCHY AND SUPERVISION STRATEGY

Goals:

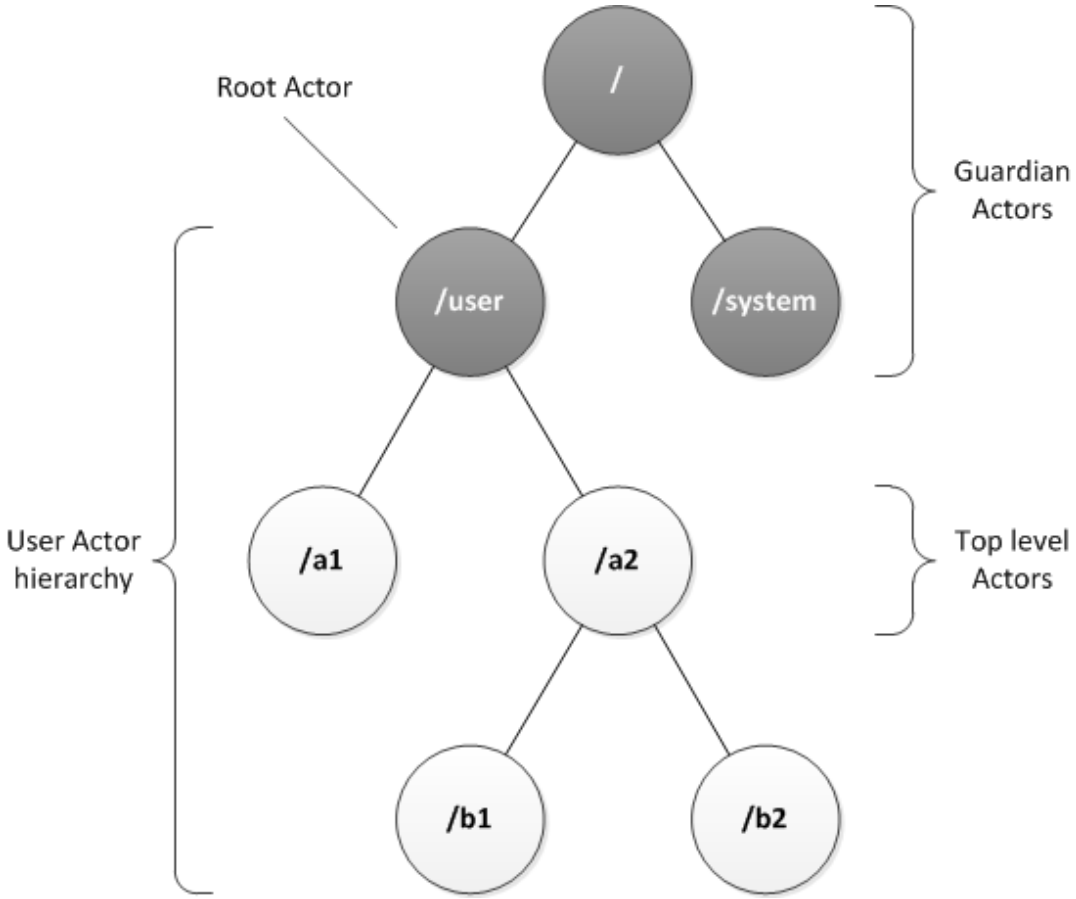
Understand how to create child Actor(s), and apply supervision strategy to handle failures.

Steps:

1. Setup child Actor(s)
2. Setup supervision strategy
3. Let Actors crash

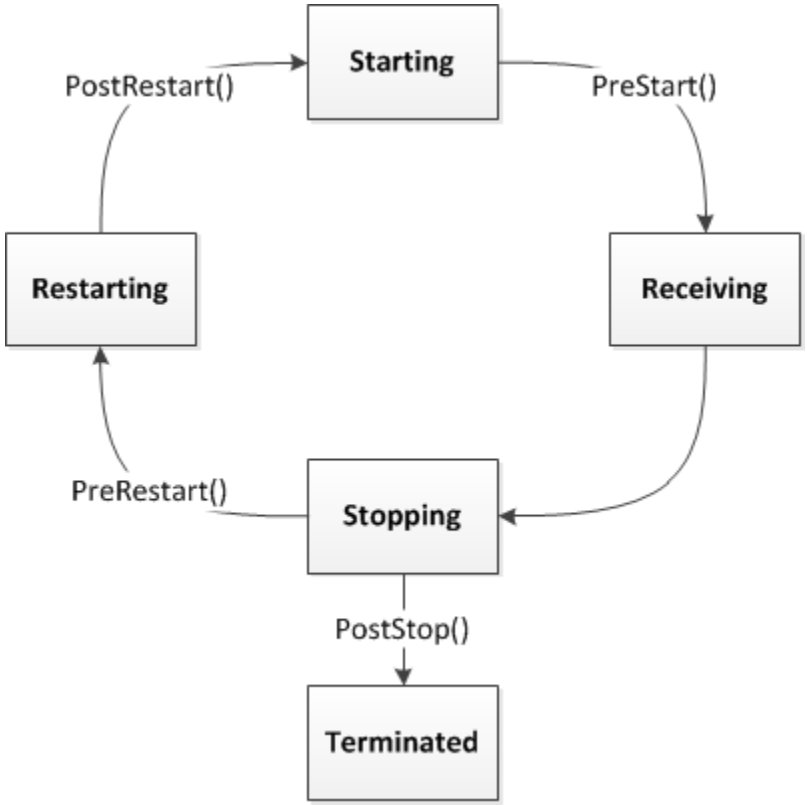
Extras:

- Failure directives
- Hooking into the actor's lifecycle



PART 3:

ACTOR HIERARCHY AND SUPERVISION STRATEGY



PART 4:

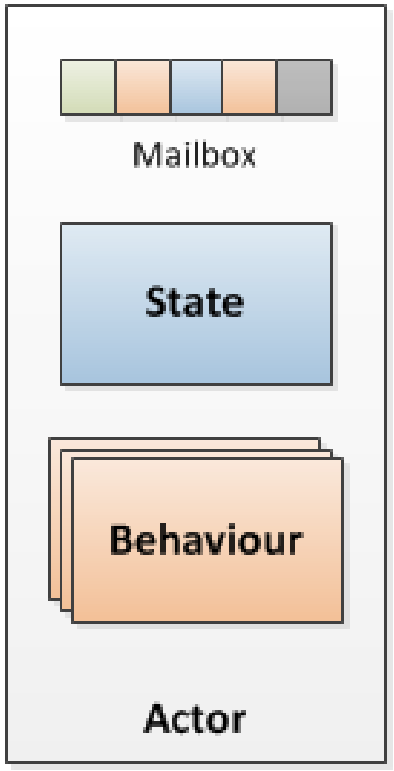
BEHAVIOUR AND HOTSWAP

Goals:

Understand how to control change the actor's behaviour at runtime.

Steps:

1. Use `Become` method
2. Use `BecomeStacked` and `UnbecomeStacked` methods



QUESTIONS?

APPENDIX A

.NET ACTOR SYSTEM ALTERNATIVES

.NET ACTOR MODEL LIBS & FX[1]

Name	License	Link	Comments
ActorFx	Apache 2.0	https://actorfx.codeplex.com/	
Akka.NET	Apache 2.0	http://getakka.net/	A port of Akka on JVM
F# Mailbox Processor	Apache 2.0	http://en.wikibooks.org/wiki/F_Sharp_Programming/MailboxProcessor	Part of F# core library
NAct	LGPL 3.0	http://code.google.com/p/n-act/	
Project Orleans	MIT	http://research.microsoft.com/en-us/projects/orleans/	
PostSharp	Commercial	http://doc.postsharp.net/actor	
Remact.Net	MIT	https://github.com/steforster/Remact.Net	
RetLang	New BSD	http://code.google.com/p/retlang/	

Notes:
[1] – http://en.wikipedia.org/wiki/Actor_model#Actor_libraries_and_frameworks

F# MAILBOX PROCESSOR

HTTP://EN.WIKIBOOKS.ORG/WIKI/F_SHARP_PROGRAMMING/MAILBOXPROCESSOR

What is F# Mailbox Processor?

- A dedicated message queue built as part of the F# core libraries. Found in *Fsharp.Core.dll*:
  - Namespace: *Microsoft.Fsharp.Control.MailboxProcessor*
  - Source code: https://github.com/fsharp/fsharp/blob/master/src/fsharp/FSharp.Core/control.fs#L2124

- Open source code (Apache 2 license), part of F# repository on GitHub: https://github.com/fsharp/fsharp

PROJECT ORLEANS

HTTP://RESEARCH.MICROSOFT.COM/EN-US/PROJECTS/ORLEANS/

What is Project Orleans?

- .NET implementation of (distributed) Actor model by Microsoft Research.

- Open source code (MIT license), available on GitHub: https://github.com/dotnet/orleans

APPENDIX B

ADDITIONAL RESOURCES

AKKA.NET BOOTCAMP

https://learnakka.net
https://github.com/petabridge/akka-bootcamp

PAPERS AND ARTICLES

Agha, G, Houck, C & Panwar, R 1992, 'Distributed Execution of Actor Programs', *Springer Berlin Heidelberg*.

Bereznitsky, D 2010, 'The Actor Model – Towards Better Concurrency', *SlideShare*, 27 January, viewed 20 January 2015 , <http://www.slideshare.net/drorbr/the-actor-model-towards-better-concurrency>

Hewitt, C 2010, 'Actor Model of Computation: Scalable Robust Information Systems', *arXiv preprint arXiv:1008.1459*.

Ho, YL 2011, 'Introduction to Actor Model and Akka', *SlideShare*, 22 August, viewed 20 January 2015, <http://www.slideshare.net/YungLinHo/introduction-to-actor-model-and-akka>

Mackenzie, N 2014, 'Project Orleans – Actor Model Framework', *SlideShare*, 18 June, viewed 20 January 2015, <http://www.slideshare.net/nmackenzie/project-orleans>

PAPERS AND ARTICLES

Wyatt, D 2013, 'Akka Concurrency: Building Reliable Software in a Multi-Core World', *Artima Press, Walnut Creek, California*.