nova net

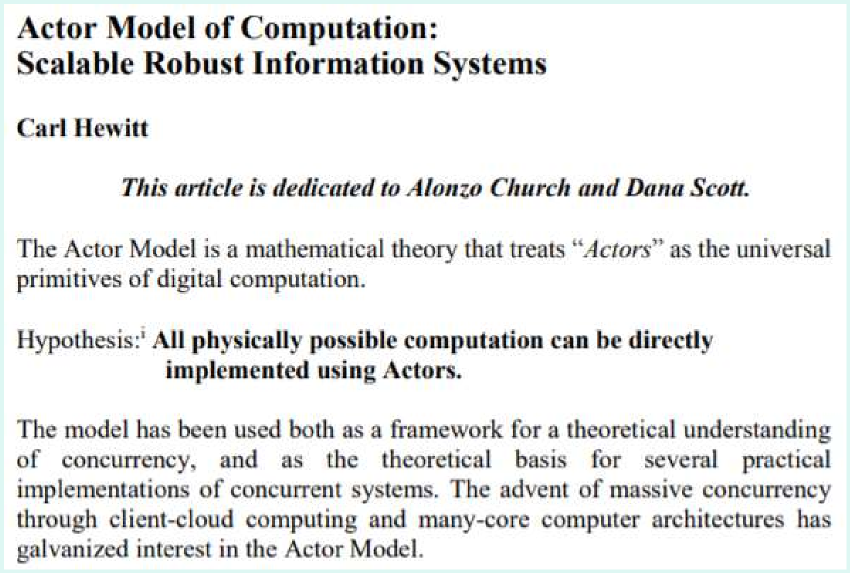
nova
net

Akka.NET

Actor Model Framework for .NET Core

Actor Model – Background and history

- Introduced by Carl Hewitt in a 1973 paper, with Peter Bishop and Richard Steiger
- A mathematical theory of computation based on the concept of an **Actor** – a fundamental unit of computation
- Motivated by the anticipation of parallelized hardware with a large number of processors with independent storage
- Has been actualized in later years, as multicore and manycore processors have become common



Actor Model of Computation:
Scalable Robust Information Systems

Carl Hewitt

This article is dedicated to Alonzo Church and Dana Scott.

The Actor Model is a mathematical theory that treats "*Actors*" as the universal primitives of digital computation.

Hypothesis:[i] **All physically possible computation can be directly implemented using Actors.**

The model has been used both as a framework for a theoretical understanding of concurrency, and as the theoretical basis for several practical implementations of concurrent systems. The advent of massive concurrency through client-cloud computing and many-core computer architectures has galvanized interest in the Actor Model.

Actor Model – Background and history

- **Early Actor Model languages:**
- Act 1, 2 and 3, Acttalk, Ani, Cantor, Rosette

- **Later Actor Model languages:**
- ABCL, AmbientTalk, Axum, CAL, D, Dart, E, Elixir, Erlang (Ericsson), Fantom, Humus, Io, LFE, Encore, Pony, Ptolemy Project, P, P#, Rebeca modeling Languiage, Reia, SALSA, Scala, TNSDL

- **Actor Model Frameworks and Libraries:**
- **.NET:**
- ActorFx, Akka.NET, F# MailboxProcessor, NAct, Orleans, PostSharp, protoactor, Retlang, Remact.Net

- **C/C++:**
- Actor-CPP, C++ Actor Framework (CAF), Cloudl, Clutter, czmq, Libactor, libagents, libprocess, OOSMOS, Orleans, QP, rotor, Skynet, SObjectizer, Theron

- **Java:**
- Actor, Actor4j, ActorFoundry, Actr, Akka, Ateji PX, FunctionalJava, JActor, Jetlang, Kilim[49], Korus, Orbit, Peernetic, Quasar, S4, Vert.x, vlingo

- **Others:**
- Acteur, Actix, ActorKit, Akka, Aojet, Bastion, Celluloid, Cloud Haskell, Comedy, GPars, Haskell-Actor, LabVIEW Actor Framework, LabVIEW Messenger Library, Nact, PARLEY, protoactor, Pulsar, Pulsar, Pykka, Reactors.IO, Riker, Termite Scheme, Thespian, Vert.x, vlingo, waSCC

Evolution of Akka

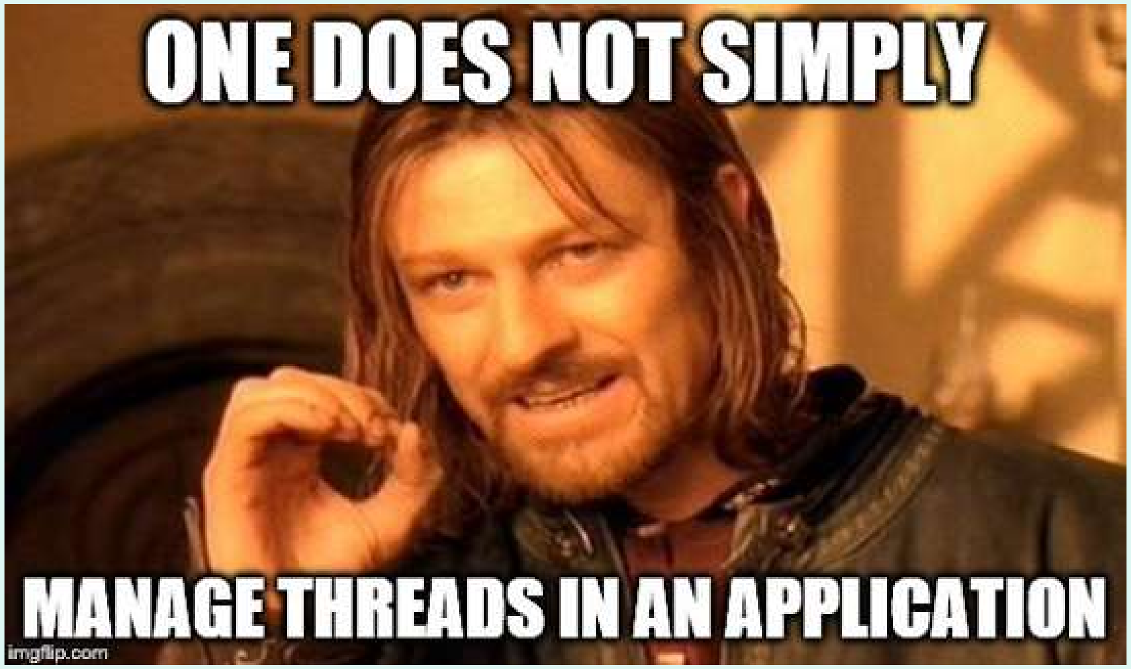
- Akka
- Release 2008 as part of Scala 2.1.7

- Ported to .NET in 2013, now an open source project hosted on github.

nova
net

Problems with parallelization

- Shared State
  - Race Conditions
  - Blocking calls
  - Deadlocks

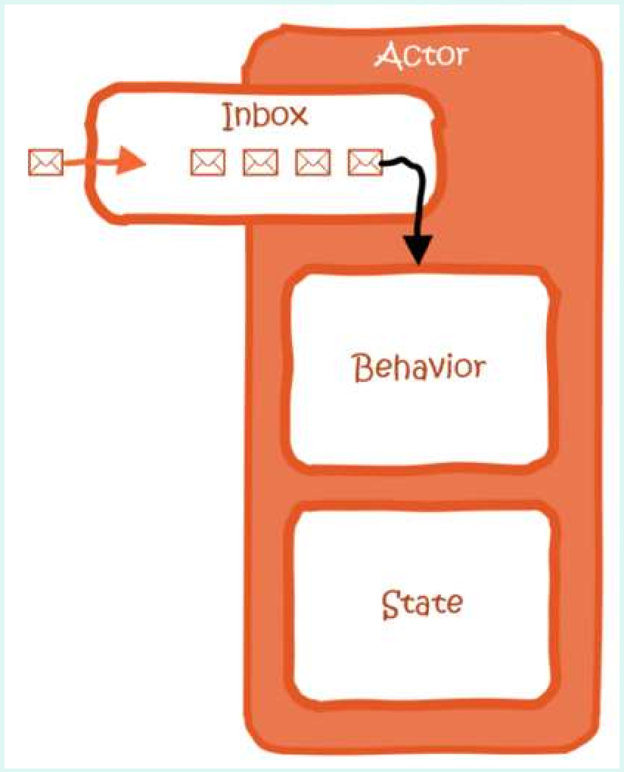


nova
net

The Actor

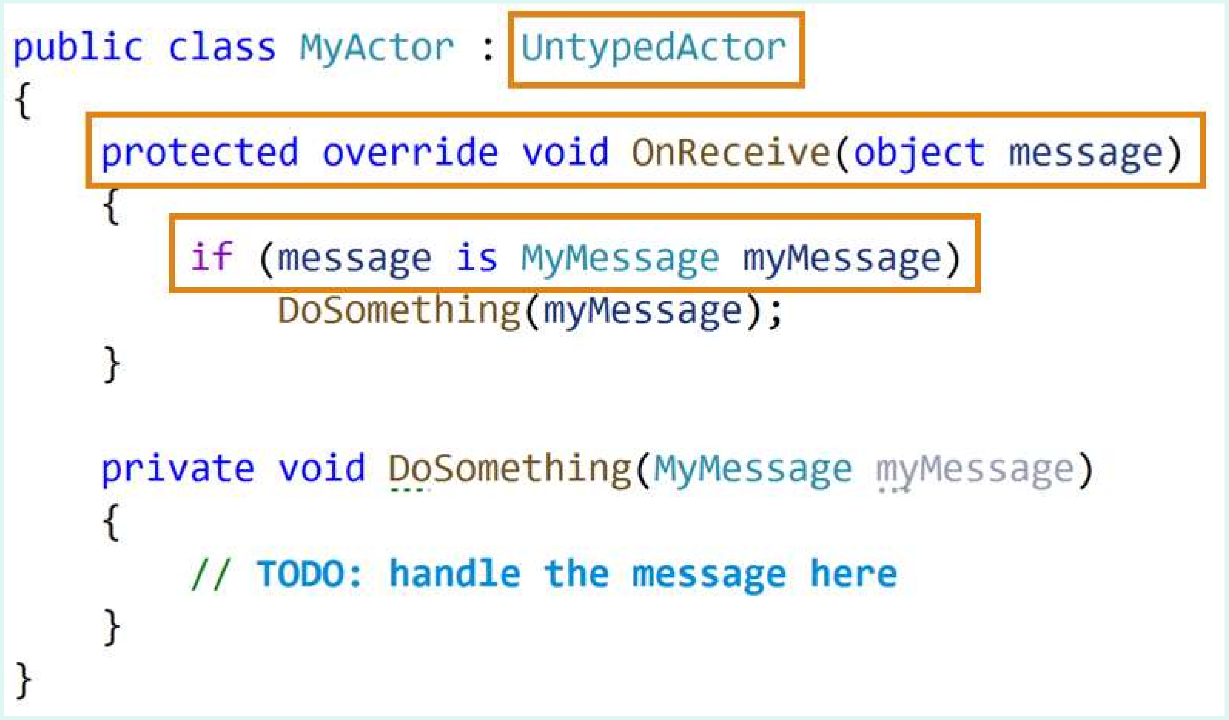
- Simple object
  - Holds its own state
  - Inbox
    - Messages (the only input)
    - Processed in order
    - 1 message at a time

- Guaranteed single threaded



nova
net

The simplest actor



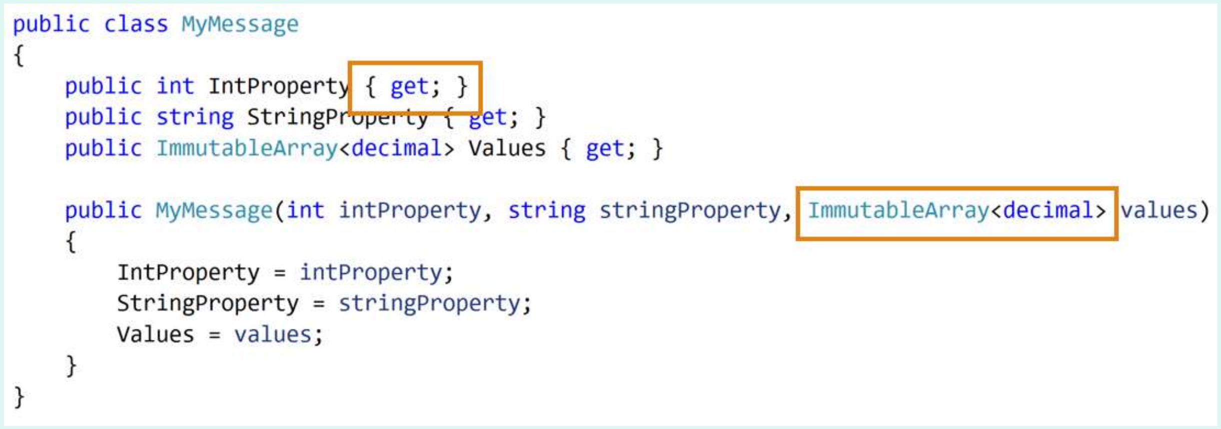
```
public class MyActor : UntypedActor
{
    protected override void OnReceive(object message)
    {
        if (message is MyMessage myMessage)
            DoSomething(myMessage);
    }

    private void DoSomething(MyMessage myMessage)
    {
        // TODO: handle the message here
    }
}
```

nova
net

Messages

- Simple objects
- Immutable
  - Note: Akka.NET does not enforce this
- Might cross machine boundaries



```
public class MyMessage
{
    public int IntProperty { get; }
    public string StringProperty { get; }
    public ImmutableArray<decimal> Values { get; }

    public MyMessage(int intProperty, string stringProperty, ImmutableArray<decimal> values)
    {
        IntProperty = intProperty;
        StringProperty = stringProperty;
        Values = values;
    }
}
```

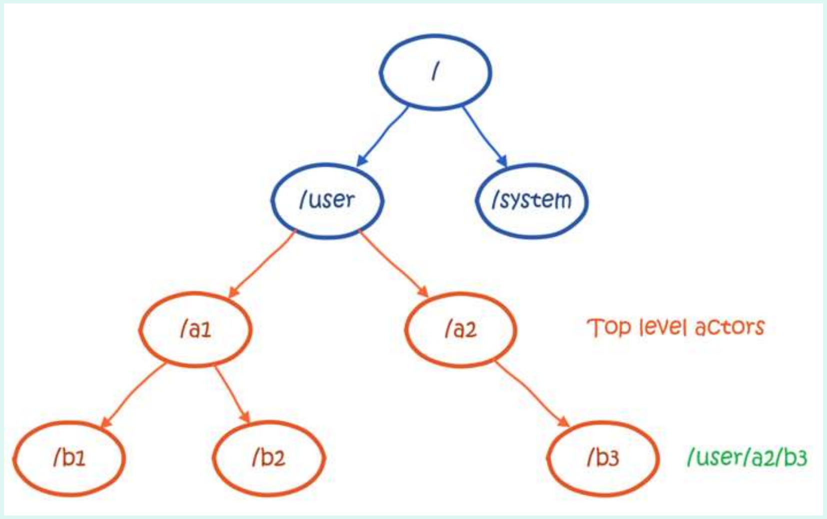
nova
net

The ActorSystem

- The ActorSystem manages:
  - Actor life cycles
  - Messaging
  - Inboxes
  - Thread scheduling
  - The system event bus
  - Persistence
  - Remoting
  - Clustering

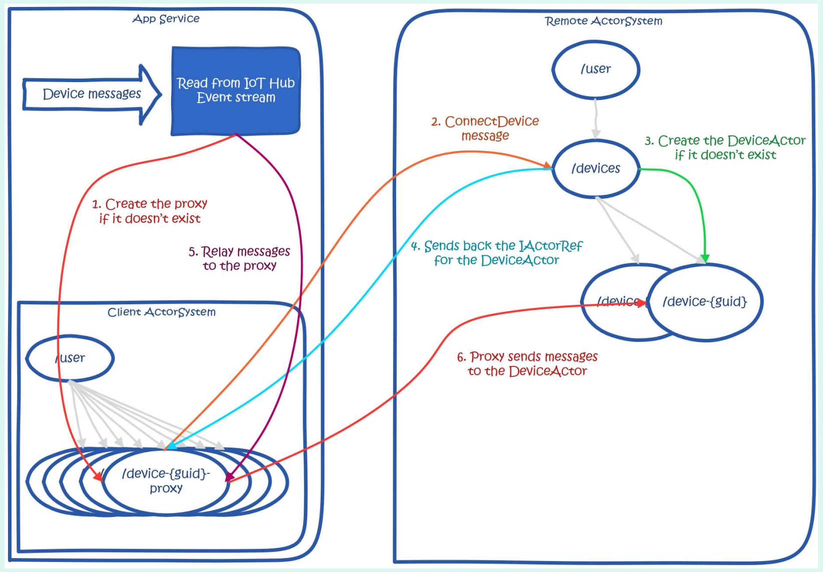
nova
net

Actor hierarchy



- Actors can have children
- Position = address
- 3 default actors:
  - /
  - /user
  - /system

Demo: processing IoT readings


App Service
Device messages
Read from IoT Hub Event stream
1. Create the proxy if it doesn't exist
5. Relay messages to the proxy
Client ActorSystem
/user
/device-{guid}-proxy
Remote ActorSystem
/user
2. ConnectDevice message
/devices
3. Create the DeviceActor if it doesn't exist
4. Sends back the IActorRef for the DeviceActor
/device
/device-{guid}
6. Proxy sends messages to the DeviceActor