Microsoft

Microsoft Ignite

Lights, Camera, Akka!

The Actor Model & Agentic AI Orchestra

An exploration of Actors, Akka.NET, and Agentic AI orchestration patterns.

Prabh Singh

Agenda

- What is the Actor Model?
- Core Concepts
- Akka.NET Ecosystem
- Message Processing
- Supervision & Fault Tolerance
- Real-World Use Cases & Applications
- Agentic AI Orchestration Patterns
- Orchestration Patterns Comparison
- Various Actor Model Implementations

What is the Actor Model?

A Conceptual Model for Concurrent Computation

Origins

- Born from Erlang in the 1970s-80s
- Whitepaper published in 1977
- Proven in telecommunications for decades

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

AI Working Paper 134A May 10, 1977

Laws for Communicating Parallel Processes

by

Carl Hewitt and Henry Baker

This paper presents some laws that must be satisfied by computations involving communicating parallel processes. The laws are stated in the context of the *actor theory*, a model for distributed parallel computation, and take the form of stating plausible restrictions on the histories of parallel computations to make them physically realizable. The laws are justified by appeal to physical intuition and are to be regarded as falsifiable assertions about the kinds of computations that occur in nature rather than as proven theorems in mathematics. The laws are used to analyze the mechanisms by which multiple processes can communicate to work effectively together to solve difficult problems.

Since the causal relations among the events in a parallel computation do not specify a total order on events, the actor model generalizes the notion of computation from a *sequence of states* to a *partial order of events*. The interpretation of unordered events in this partial order is that they proceed concurrently. The utility of partial orders is demonstrated by using them to express our laws for distributed computation.

Core Concepts

Actors

Fundamental units of computation that encapsulate state and behavior

Mailboxes

Message queues where actors receive and process messages asynchronously

Fault Tolerance

Supervision hierarchies that handle failures gracefully

Distribution

Seamless communication across network boundaries

Location Transparency

Actors communicate the same way whether they are local or remote - location doesn't matter to the code

Akka.NET: The Orchestration Platform

Most popular Actor Model implementation for .NET

Akka Core

Base package with essential actor functionality

Akka.Remote

Cross-process communication across networks

Akka.Persistence

Event sourcing for data persistence and recovery

Akka.Cluster

Highly available actor networks

Akka.Streams

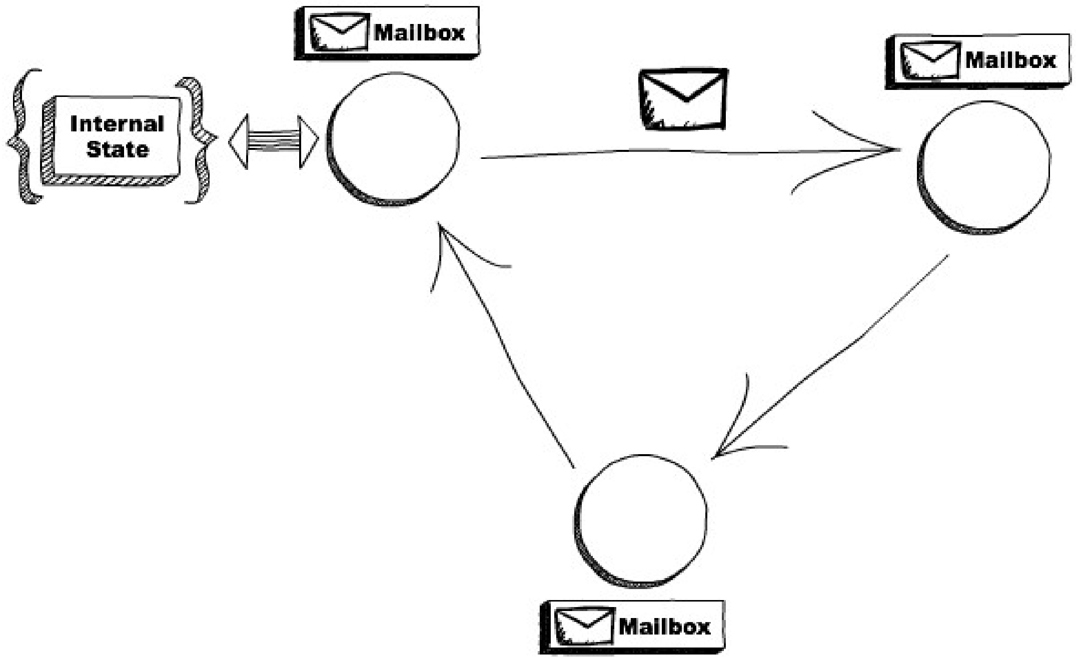
High-performance streaming: Kafka, SignalR, Azure Event Hub

Message Processing

How Actors Communicate

1. Actor receives message in mailbox
2. Processes one message at a time (sequential)
3. Can send messages, create actors, or change behavior
4. No shared state = No race conditions!

🎯 **Key Advantage: Eliminates traditional concurrency problems**


Mailbox
Internal State
Mailbox
Mailbox

Let It Crash: Supervision Hierarchies

The "Let It Crash" Philosophy

• Parent actors supervise child actors

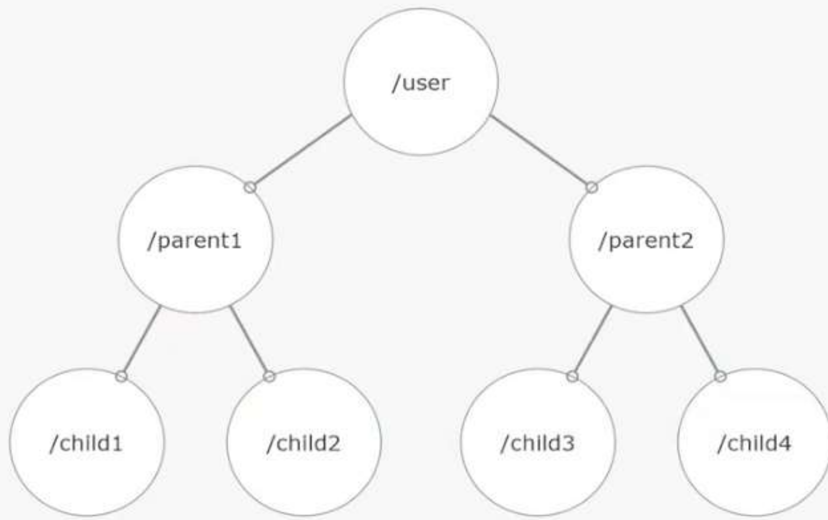
• Failures are isolated and contained

• Supervisors decide: Restart, Resume, Stop, or Escalate

Reactive & Resilient

System continues operating even when individual components fail. Self-healing architecture that recovers automatically.

Actors Live in Supervision Hierarchies



Creating Actors

```
// create ActorSystem (allows actors to talk in-memory)
var actorSystem = ActorSystem.Create("PingPong");

// Props == formula used to start an actor.
var pingActorProps = Props.Create(factory: () => new PingActor());

// start pingActor and get actor reference (IActorRef)
IActorRef pingActor = actorSystem.ActorOf(pingActorProps, name: "ping");

// tell pingActor a message
pingActor.Tell(new Ping(count: 0));
```

Basic Akka.NET Actor

```
public class PingActor : Akka.Actor.ReceiveActor
{
    private readonly ILoggingAdapter _log = Context.GetLogger();

    public PingActor()
    {
        Receive<Ping>(handler: p =>
        {
            _log.Info(format: "Received {0}", p);

            // reply back at a random, short interval
            var replyTime = TimeSpan.FromSeconds(
                ThreadLocalRandom.Current.Next(1, 5));

            Context.System.Scheduler.ScheduleTellOnce(
                replyTime, // delay
                Sender, // target
                message: p.Next(), // message
                Self); // sender (optional)
        });
```

Actor base type

Handle to built-in logging system
(automatically thread-safe)

Message handler for
messages of type Ping.

Sender = reference to actor
who sent us the Ping message.

Real-World Applications

🏗️ Microservices Architecture

Loosely coupled, independently deployable services

📡 IoT Data Streaming

Real-time processing of sensor data at scale

🎬 Event-Driven Systems

Reactive architectures that respond to events instantly

📊 Big Data Processing

Distributed processing of massive datasets

🤖 AI Model Orchestration

Coordinating multiple AI models and agents

Agentic AI orchestration

AI Orchestration Patterns

🎭 Coordinator Pattern

Central orchestrator delegates tasks to specialized AI agents

🔗 Pipeline Pattern

Agents process data sequentially, each adding intelligence

🌐 Swarm Pattern

Multiple agents work collaboratively on complex problems

🎯 Specialist Pattern

Domain-specific AI actors handle specialized tasks

How These Patterns Compare

Pattern	Control	Flexibility	Best For
🎭 Coordinator	Centralized	Medium	Complex tasks needing oversight
🔗 Pipeline	Sequential	Low	Predictable multi-stage processes
🌐 Swarm	Distributed	High	Creative, open-ended problems
🎯 Specialist	Routed	Medium	Domain-specific expertise

💡 **Pro Tip:** Combine patterns for complex AI workflows. For example, use a Coordinator to manage multiple Specialist actors, or chain Pipelines within a Swarm.

The Natural Synergy: Actor Pattern + Agentic AI

Both focus on independent, self-contained entities that communicate via messages

Concurrency
Independent processes with autonomous goals

Message-Driven
Communication via protocols

Encapsulation
Private state and beliefs

Distribution
Network-agnostic systems

Resilience
Let it crash with isolation

Perfect Match
Infrastructure + Intelligence

Where Actor Pattern + AI Excel Together

🎮 Multi-Agent Simulations

Complex simulations where each agent is an autonomous actor

Traffic systems, logistics, gaming NPCs

🔧 Distributed AI Services

Microservices with embedded AI logic

Specialized AI capabilities per service

📡 IoT & Edge Computing

Distributed devices with local AI decisions

Graceful concurrency and failure handling

🤖 Robotics Coordination

Multiple robots coordinating via async messaging

Local autonomy while collaborating

AutoGen Group Chat Pattern

How AutoGen Orchestrates Agents

AutoGen uses a GroupChatManager to coordinate multi-agent conversations, similar to an actor supervisor

🎭 Speaker Selection

- **Auto:** LLM selects next speaker
- **Round-robin:** Sequential turns
- **Manual:** Human selection
- **Custom:** User-defined logic

🔄 Conversation Patterns

- **Two-agent:** Simple back-and-forth
- **Sequential:** Chained conversations
- **Group chat:** Multi-agent coordination
- **Nested:** Hierarchical workflows

🔍 Key Insight: Actor-Like Behavior

✓ Each agent is like an actor with a mailbox

✓ Messages trigger responses (message handlers)

✓ GroupChatManager acts as supervisor

Actor Model Implementations Across the Ecosystem

🎯 Traditional Actor Frameworks

- **Erlang/OTP**
  The original (1980s)
- **Elixir**
  Modern Erlang VM
- **Akka (Scala/JVM)**
  Industry standard
- **Akka.NET**
  .NET implementation

☁️ Cloud & Distributed

- **Microsoft Orleans**
  Virtual actors
- **Cloudstate**
  Serverless actors
- **Dapr**
  Distributed apps
- **Pico**
  Lightweight actors

🦀 Modern Implementations

- **Actix (Rust)**
  High-performance
- **Ray**
  Distributed Python
- **Proto.Actor**
  Cross-platform
- **CAF (C++)**
  C++ Actor Framework

🎨 UI & State Machines

- **XState**
  State machines for JavaScript/TypeScript UI
- **Redux-Observable**
  Actor-like patterns for React
- **Elm**
  Functional reactive UI

🤖 AI Agent Frameworks

- **AutoGen**
  Multi-agent conversations (Microsoft)
- **Magentic-One**
  Generalist multi-agent system
- **LangGraph**
  Agent orchestration graphs
- **CrewAI**
  Role-based AI agents

💡 **The actor model pattern transcends languages and platforms—from embedded systems to cloud-native applications to AI orchestration!**

The Future is Reactive

Build Scalable, Resilient, High-Performing Applications

⚡ **Reactive**

🔄 **Resilient**

📈 **Scalable**

The Actor Model + AI = The perfect orchestra for building intelligent, distributed systems

Thank You to our sponsors

Microsoft

Morgan Stanley

INFRAGISTICS

apidays

uno PLATFORM

And next time -
Your Company!

Please share your event and session feedback!

http://aka.ms/MSIgniteNYCSurvey

http://aka.ms/MSIgniteNYCSessionsSurvey

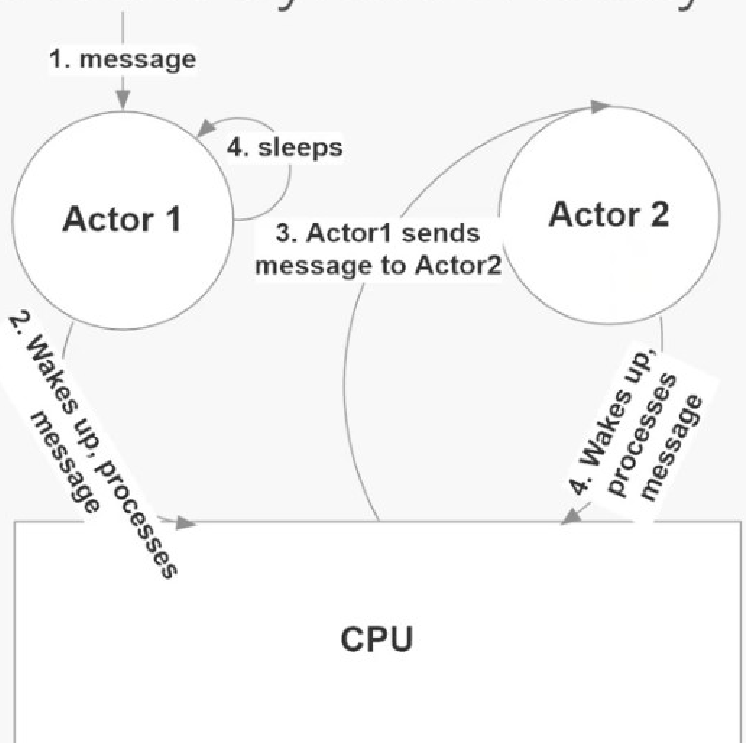
Day 1 – Nov 17

8:00am – 8:45am	🥯 **Check In & Breakfast** 🥯
8:45am – 9:00am	**Kickoff & Welcome**
9:00am – 9:30am	**Generative Pages in Power Apps**
9:30am – 10:00am	**Lights, Camera, Akka! The Actor Model & Agentic AI Orchestra**
10:00am – 10:30am	**How to create Moonshot solutions with AI**
10:30am – 10:45am	☕ **Break** ☕
10:45am – 11:15am	**Elevating Construction: Real-Time Optimization with Azure Digital Twins and AI**
11:15am – 11:45am	**Transforming Facility, Network and Organization Management with Visio and Power BI**
11:45am – 12:45pm	🍴 **Lunch** 🍴
12:45pm – 1:15pm	**Adventures in AI**
1:15pm – 1:45pm	**Building Agents in AI Foundry!**
1:45pm – 2:15pm	**What's new with Azure Load Balancer, NAT Gateway, and Public IP Addresses**
2:15pm – 2:30pm	☕ **Break** ☕
2:30pm – 3:00pm	**.NET Apps Everywhere!**
3:00pm – 3:30pm	**Accelerating Web Application Development with AI-Powered Tools: From Design to Deployment**
3:30pm – 4:00pm	**Agentic AI: Strategies for Success and Paths to Failure**
4:00pm – 4:30pm	**How (and why) Microsoft's upstream teams engage with multi-stakeholder open source projects**
4:30pm – 5:00pm	🛜 **Networking / Mingle** 🛜

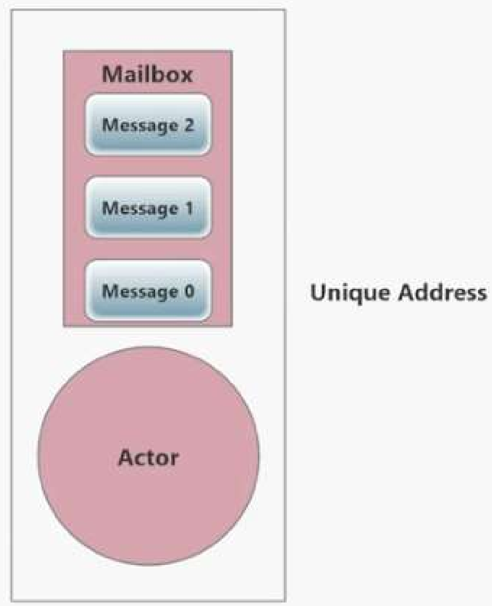
Day 2 – Nov 18

8:00am – 9:00am	🥯 **Check In & Breakfast** 🥯
9:00am – 9:30am	**Leveling Up Agents: Copilot Studio for Enterprise Solutions**
9:30am – 10:00am	**RAG Hero: Fast-Track Vector Search in .NET**
10:00am – 10:30am	**Building Resilient Systems**
10:30am – 11:00am	**Agentic Orchestration: Building Scalable, Open Source Automation with A2A, MCP and RAG Patterns**
11:00am – 12:00pm	🍴 **Lunch** 🍴
12:00pm – 2:00pm	📺 **Keynote Watch** 📺
2:00pm – 3:00pm	🎙️ **MVP Panel** 🎙️
3:00pm – 5:00pm	🛜 **Networking / Mingle** 🛜

Actors Always Run Asynchronously



Actors Process Messages One at a Time



Actors Reduce Big Problems into Small Ones