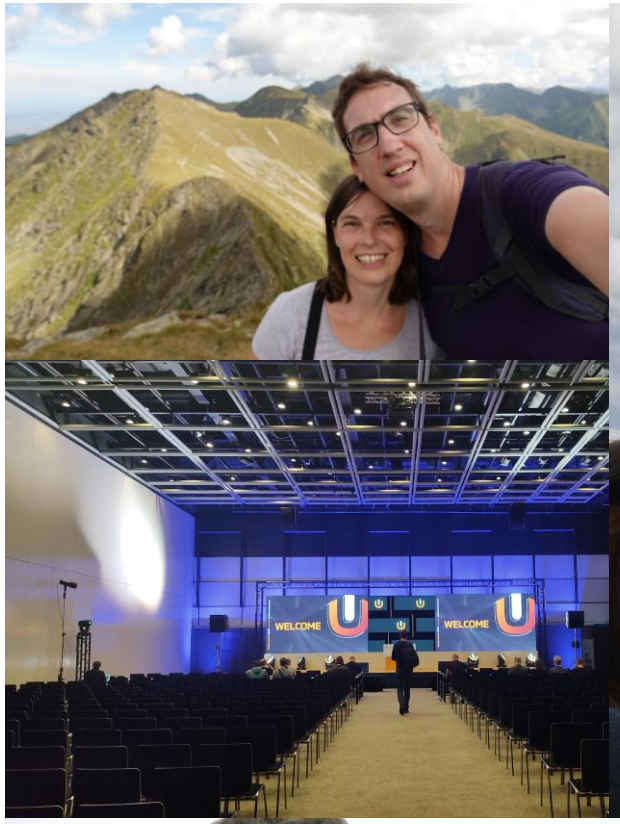
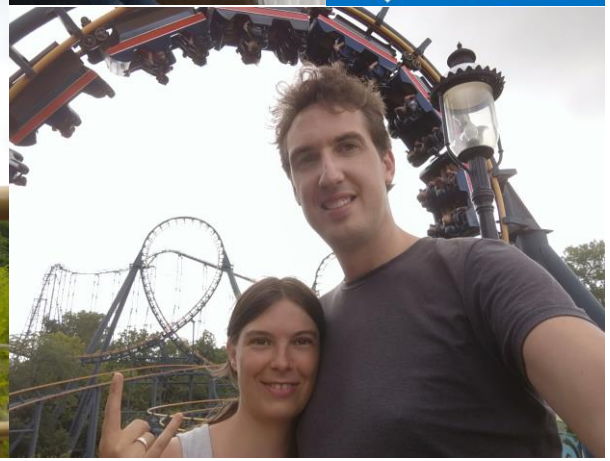

MTV
MUSIC TELEVISION
CELEBRITY
DEATHMATCH
akka.net
VS
Orleans

Who is Johnny?

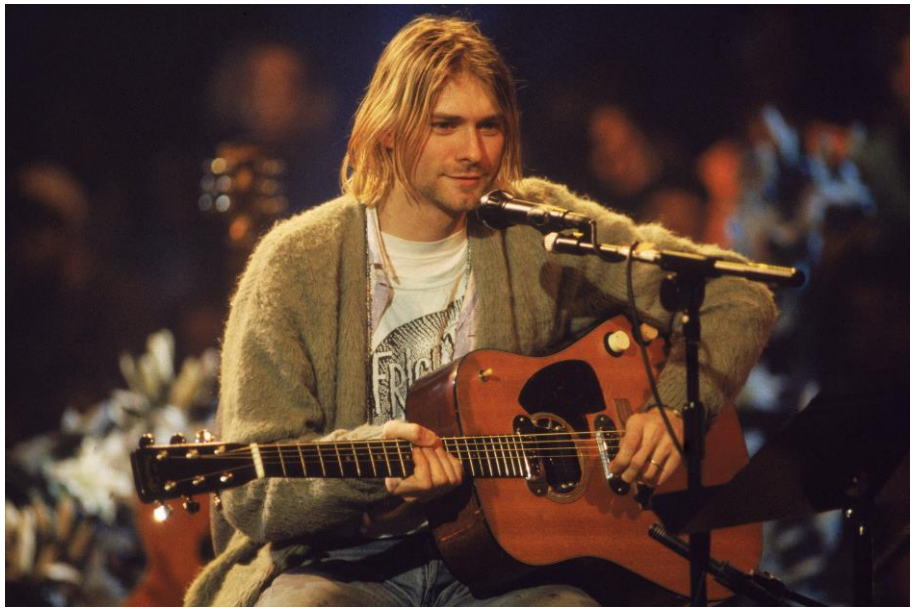
- Married to Marijke
- .NET Backend Consultant @ Involved
- Managing Partner @ Involved
- Visug Board Member
- Loves knowledge sharing
- Loves traveling the world and nature
- Addicted to Theme Parks & Rollercoasters
- Microsoft MVP





Who is Hannes?

- Father of Arne (12), Joren (9) and Marit (6)
- Partner of Barbara (?)
- Head of L&D @ Axxes
- .NET backend dev
- Loves knowledge sharing
- Amateur guitar builder
- Guitarist @
  Dylan Beattie & the Linebreakers
- Mountain biker
- Bad chess player
- Microsoft MVP



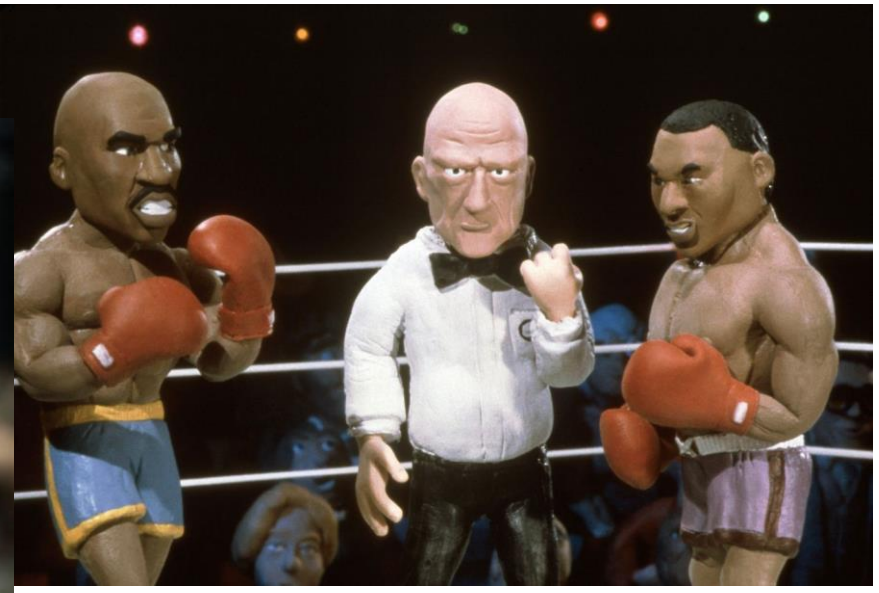
MTV
MUSIC TELEVISION®

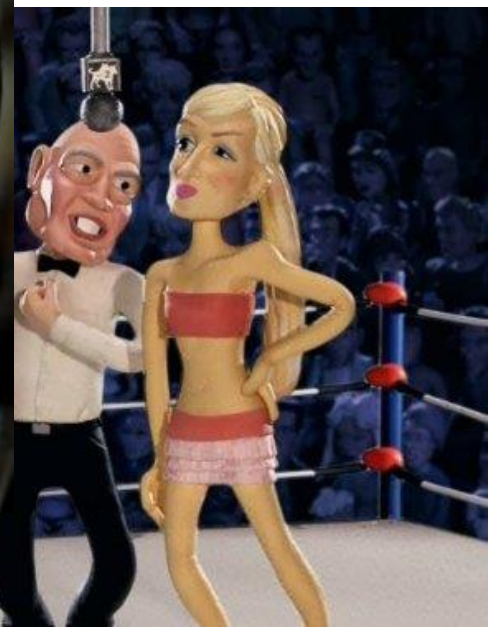
MTV
CELEBRITY DEATHMATCH

MTV BEAVIS AND BUTT-HEAD
MIKE JUDGE


DEATHMATCH


I WATCHED MTV
WHEN THEY WERE COOL
imgflip.com





Agenda

1 _ Why would you use actors?

2 _ Who is who?

3 _ General actor concepts

4 _ Akka.NET vs Orleans

5 _ Conclusion

Why would you use actors?

–

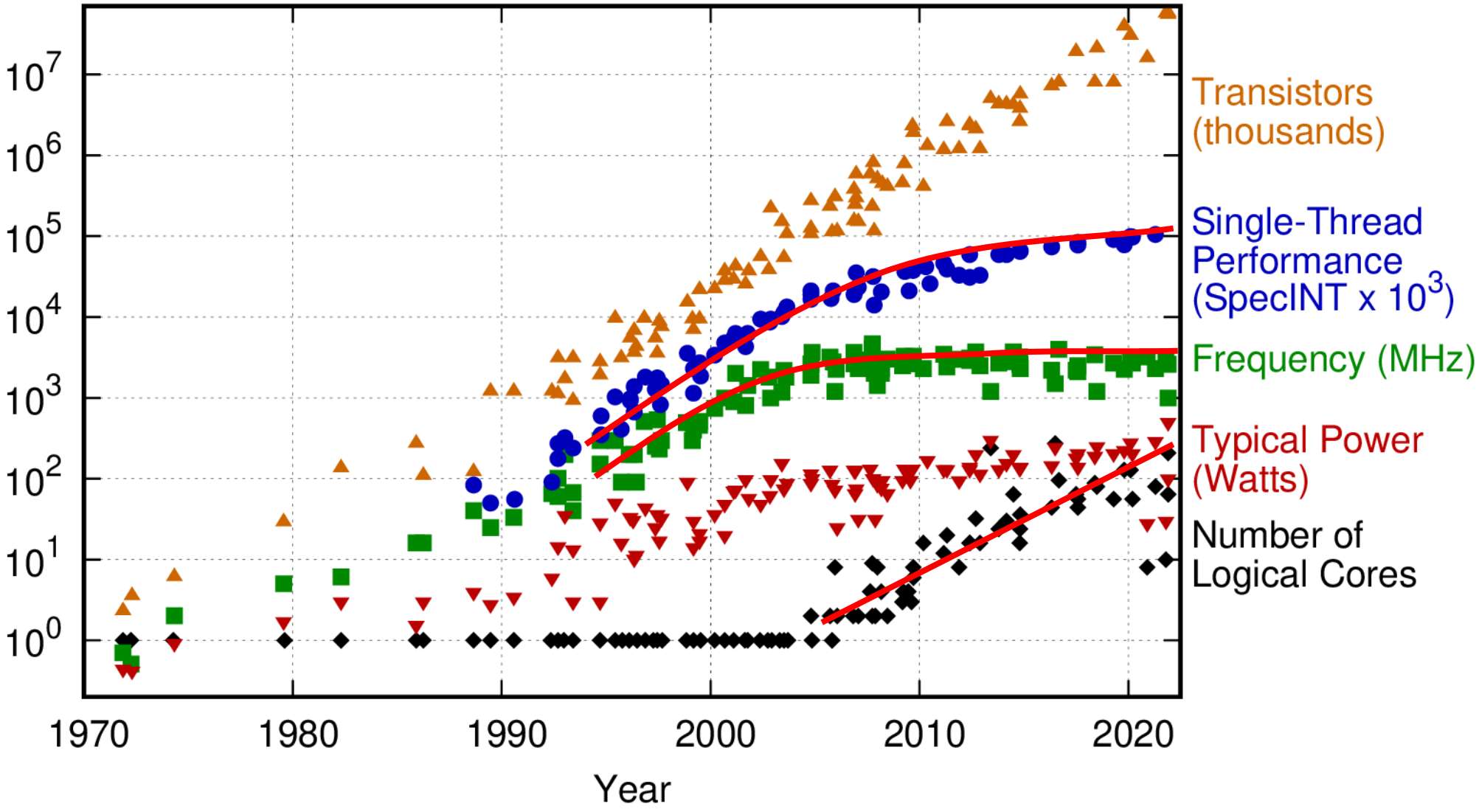
What has changed?

Software Scale

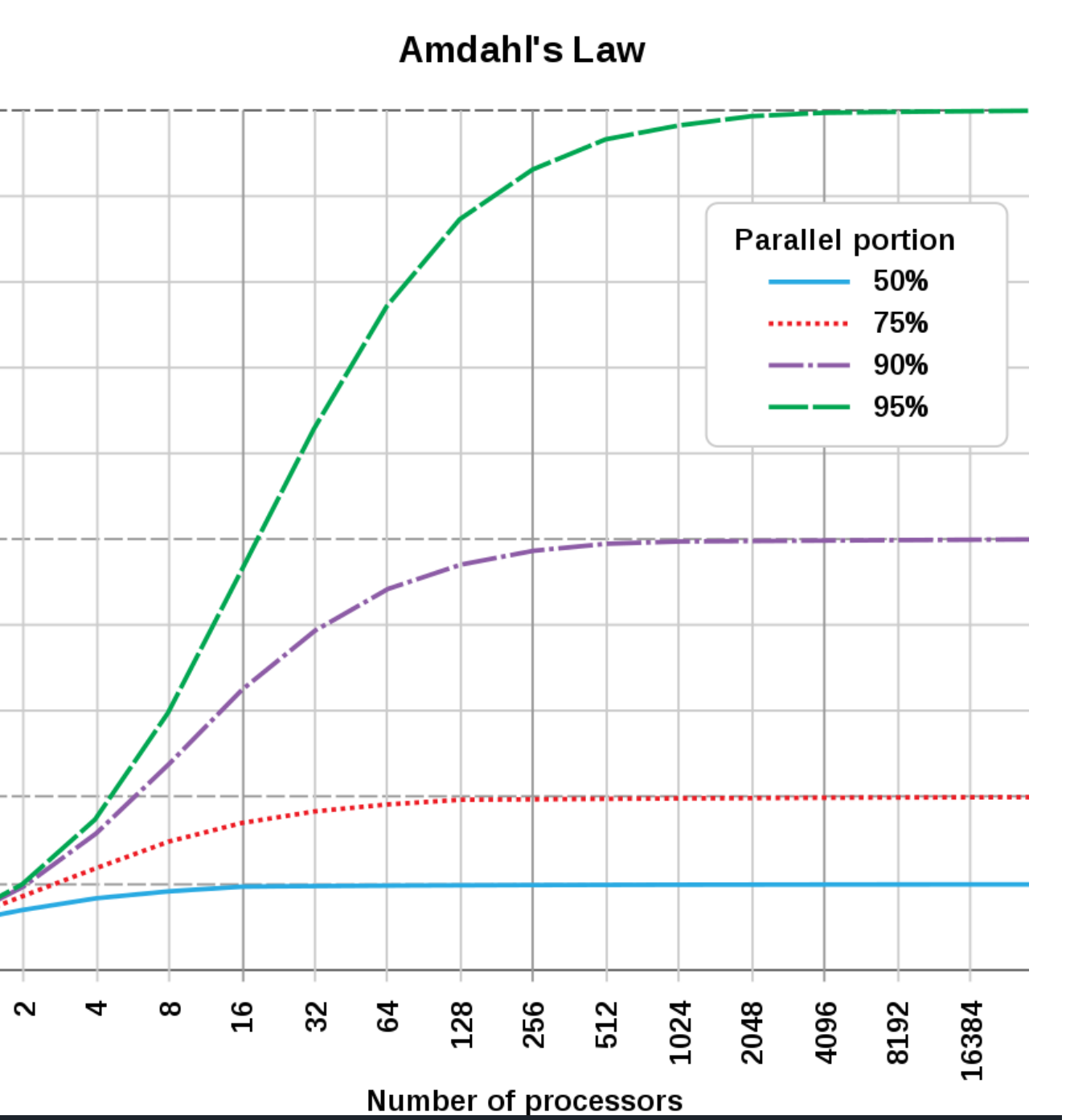
- Internet of Things
- App Backends
- Web Scale systems
- High throughput

... on commodity hardware (cloud)

50 years of microprocessor trend data



Source: 2022 - https://github.com/karlrupp/microprocessor-trend-data



Source: https://en.wikipedia.org/wiki/Amdahl%27s_law

Amdahl's Law

*The **theoretical speedup** of a workload being executed by **parallel processors**, based on the **percentage** of code that can be **parallelized**.*

Threads are hard

Shared state

↓

Blocking Calls

↓

Deadlocks

… and scheduling optimally

Actors

- Code: **single threaded**
- Parallelization: **high**
- Scale-out: **easy**
- State: **in-memory**
- Latency: **low**
- Resiliency: **high**

Who is who?

–

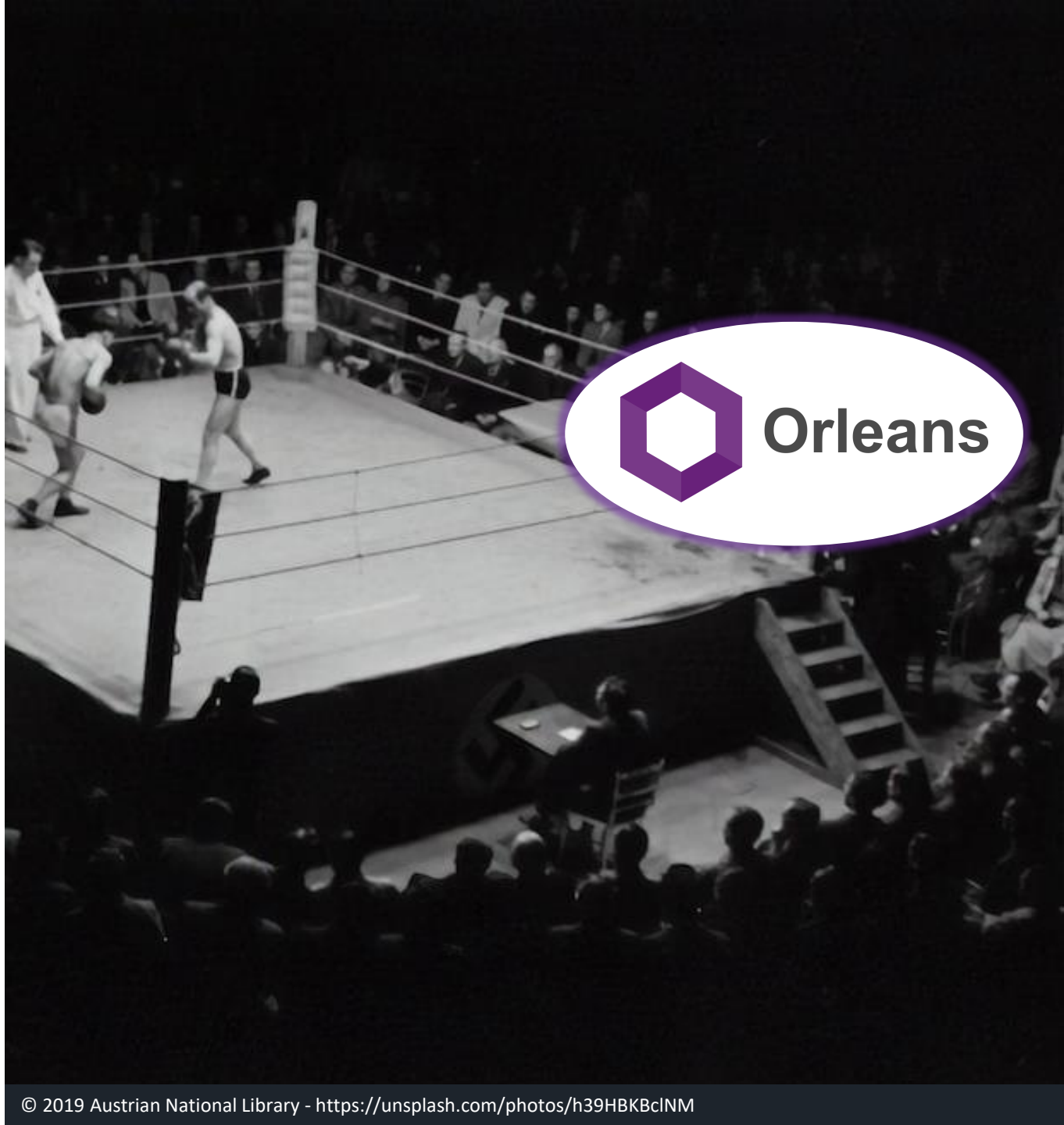
Meet the contestant in each corner

- eXtreme Computing Group
- Microsoft Research

- Open Source & Free
- Stable since 2015

- Virtual Actors
- Highly available
- Globally distributed
- Aimed at cloud

- Xbox



- Port of Akka (JVM)
- Aaron Stannard & Roger Johansson
- Petabridge

- Open Source & Free
- Paid: observability (Phobos)
- Stable since: 2015

- High performance
- Fault tolerant

General actor concepts

–

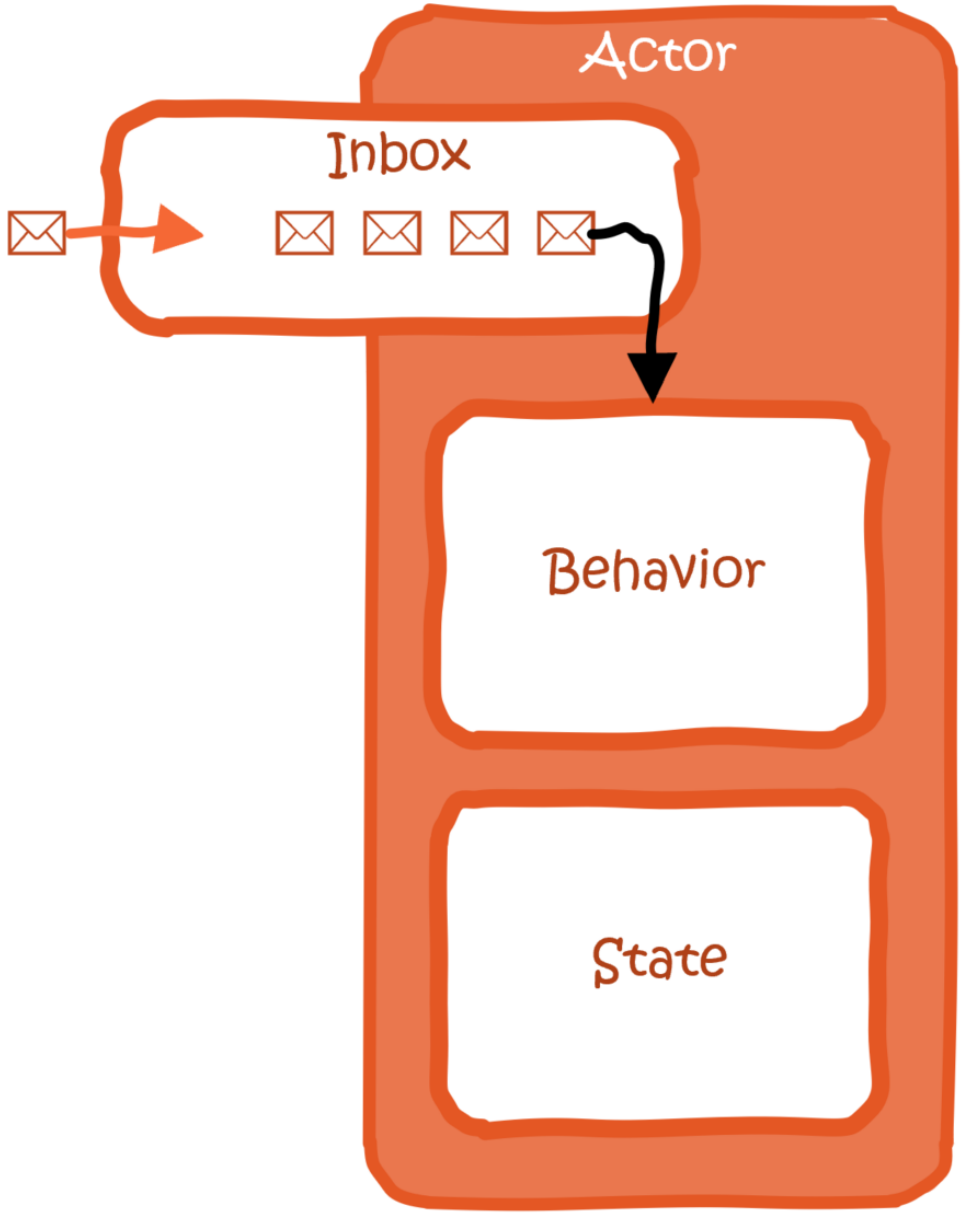
The things that unite them…

Actor / Grain

- CLR object

- State = **internal**
- Behavior = **code**
- Input = **messages**
- **Processed in order, 1 by 1**

→ **Guaranteed single threaded**



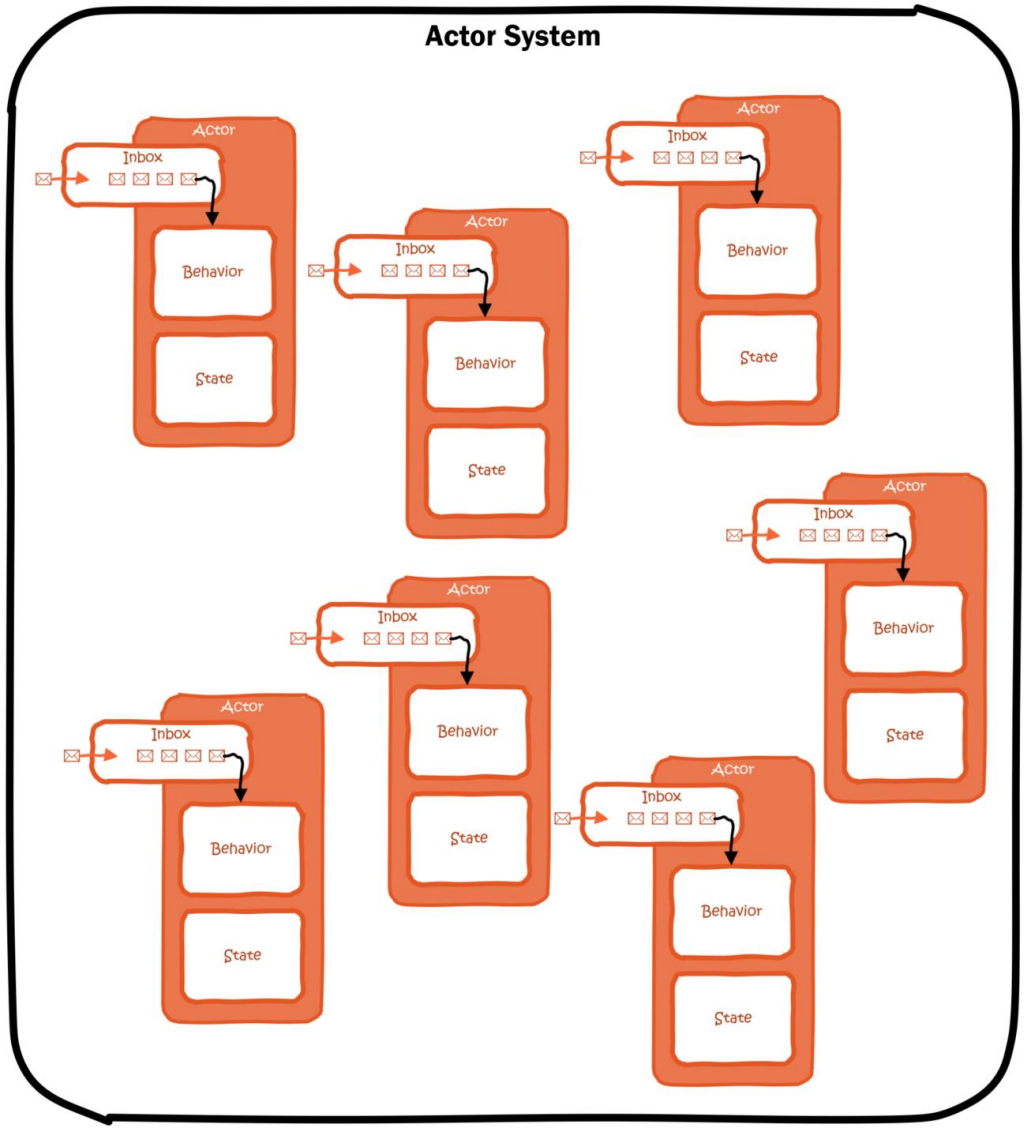
Messages

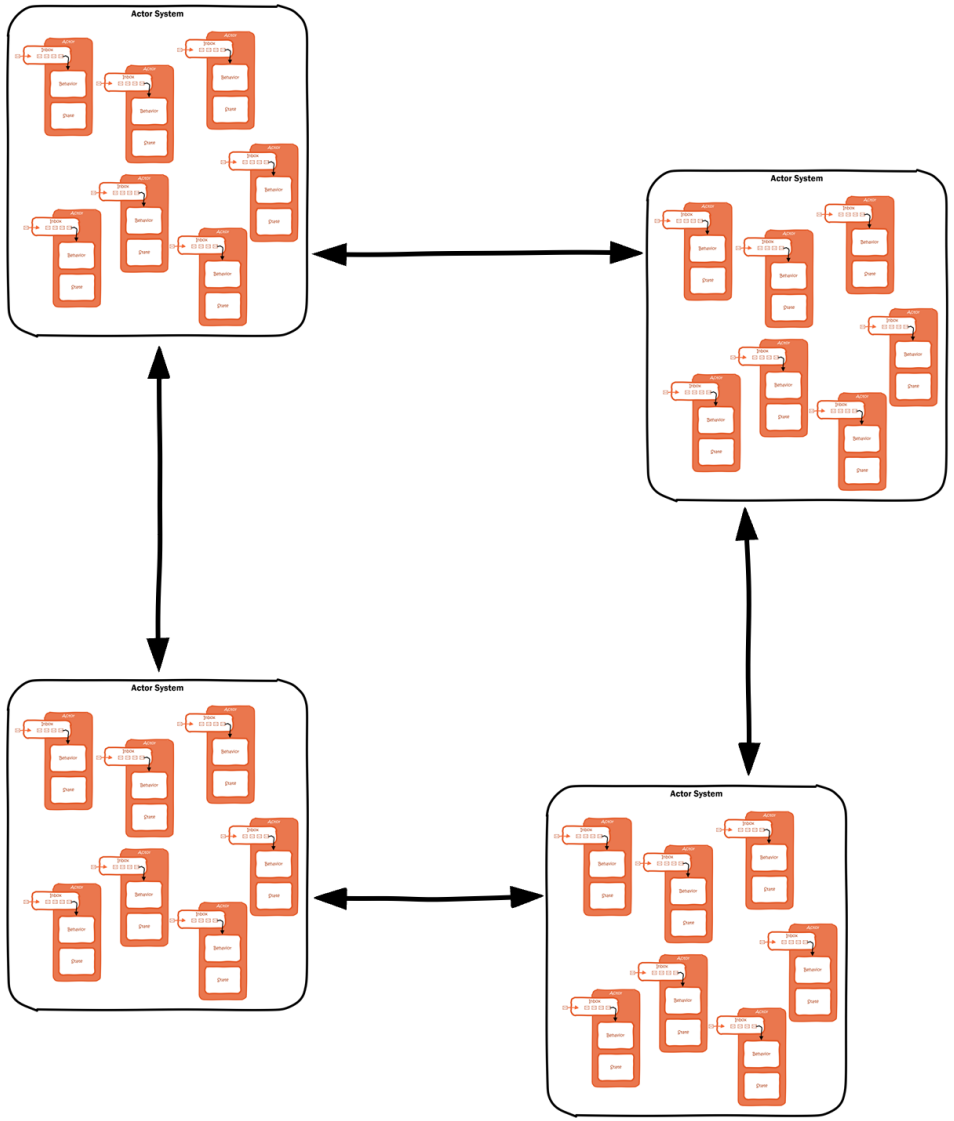
- Simple CLR objects
- Serialized to cross nodes
- Design for immutability

ActorSystem / Silo

“Actor Orchestrator”

- Actor life cycle
- Messaging
- Inboxes
- Thread scheduling
- Timers
- Event Bus
- …





Cluster

- Actor Systems or Silos can work together
- Sharding Actors across nodes
- Actors can be recreated on another node
- Resiliency against node failure

→ Enables a near-linear scale-out

Akka.NET vs Orleans

–

… and what separates them.

Writing your Actors

-

Round 1

Creating & Messaging

–

Round 2

Topology & Clusters

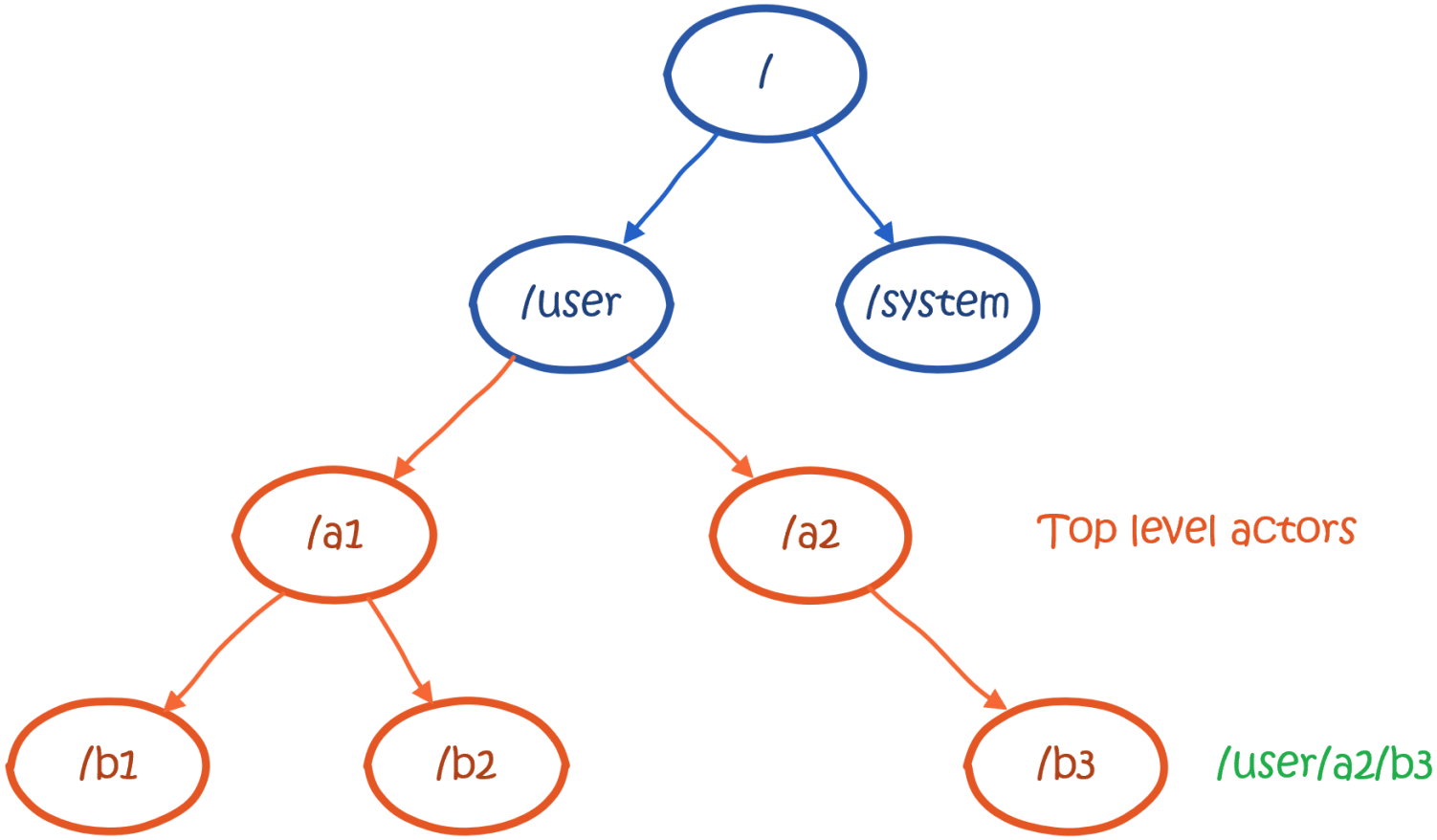
–

Round 3

Akka.NET

- Organized in a tree
- Position = address
- Address = unique

- Parents & Children



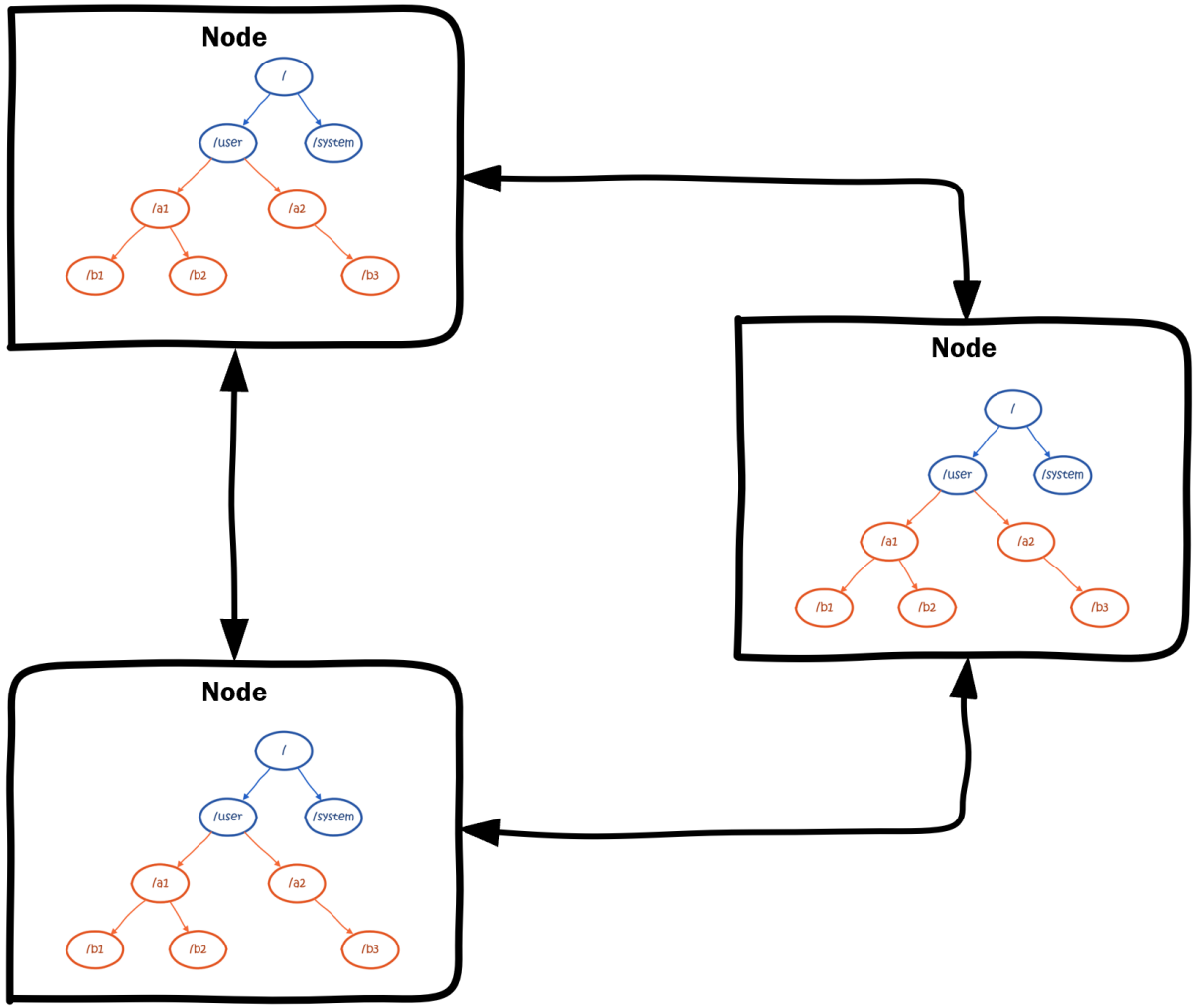
Akka.NET

```
// Create a top level actor
var props = Props.Create<DeathMatchActor>(Guid.NewGuid());
IActorRef actorRef = _actorSystem.ActorOf(props, "name");

// Inside an actor, creates a child
var props = Props.Create<DeathMatchActor>(Guid.NewGuid());
IActorRef actorRef = Context.ActorOf(props, "child-name");
```

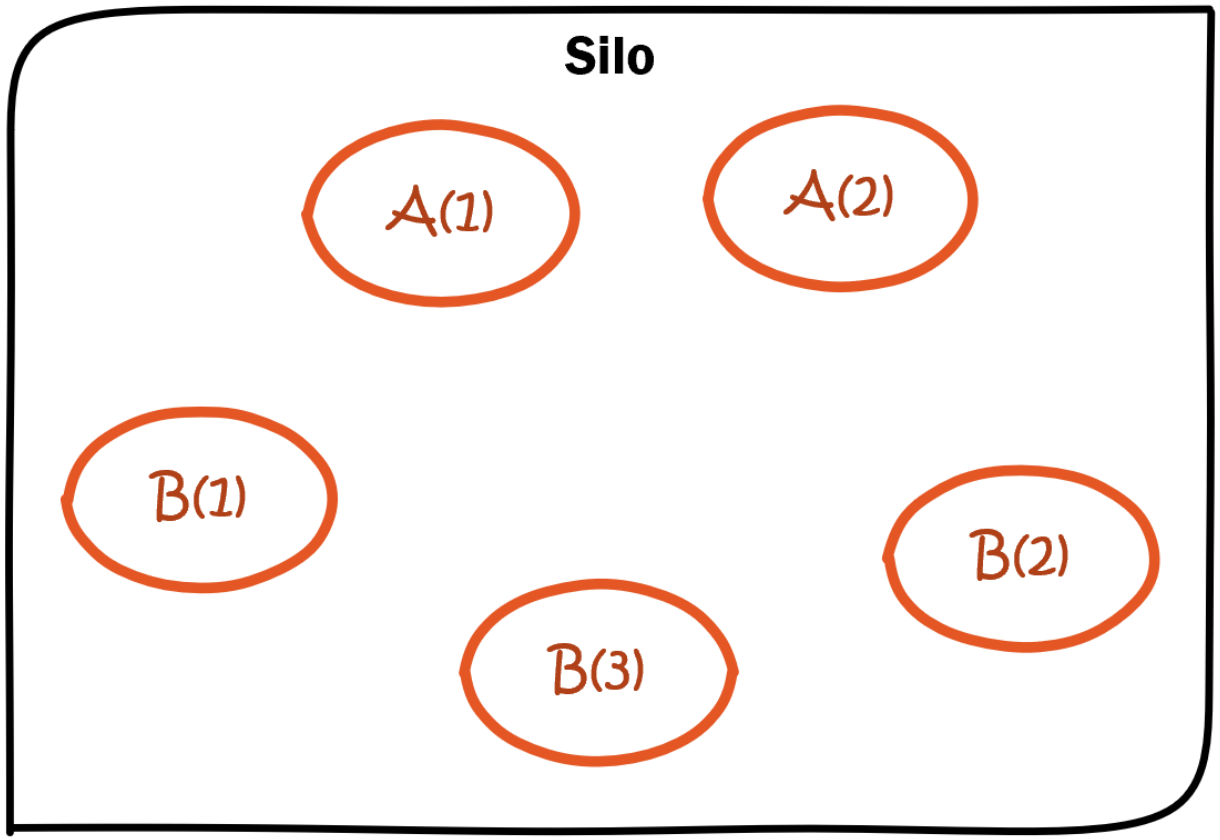
Akka.NET Cluster

- Node = independent ActorSystem
- Sharding = full control
  - Own algorithm
  - Routers
  - Hashing
  - Akka.Cluster.Sharding
- Nodes can have roles



Orleans

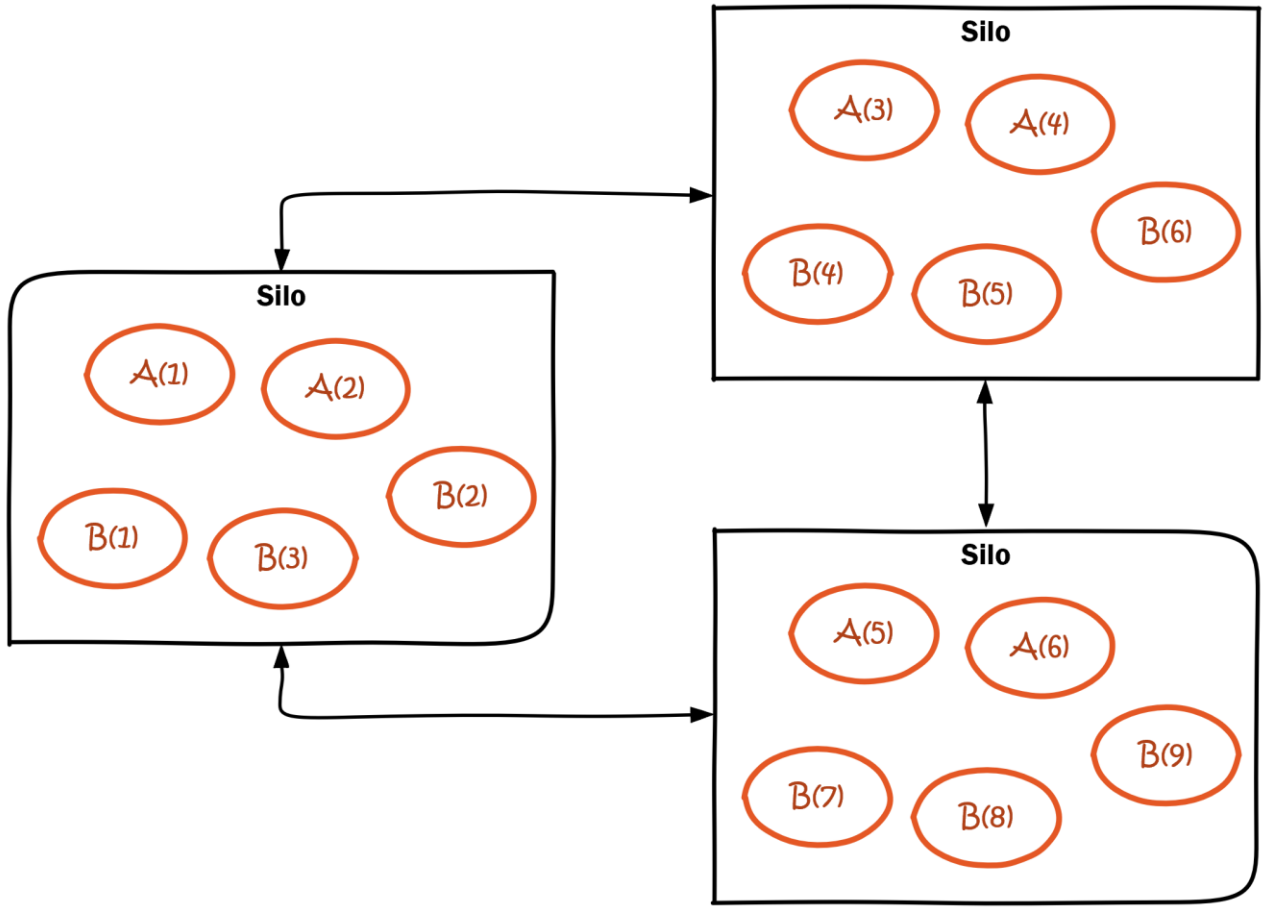
- Grains live in a Silo
- Identity = type + ID
- No hierarchy



Orleans Cluster

- Silos are cluster-aware
- Sharding = built-in
- Limited control over Silo a Grain goes in

- Virtual Actors make discovery & addressing easier



Error Handling

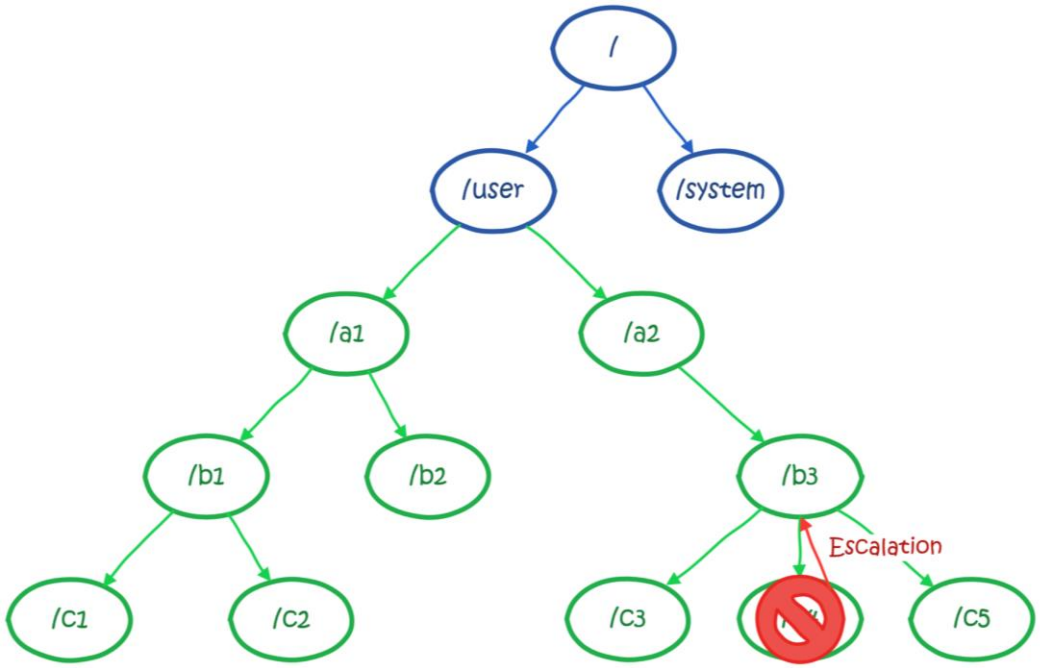
–

Round 4

Orleans

- Exceptions sent back to caller:
  - Serialized
  - Deserialized
  - Thrown
- Catch exceptions on the call side
- Downside: wait for response

Akka.NET



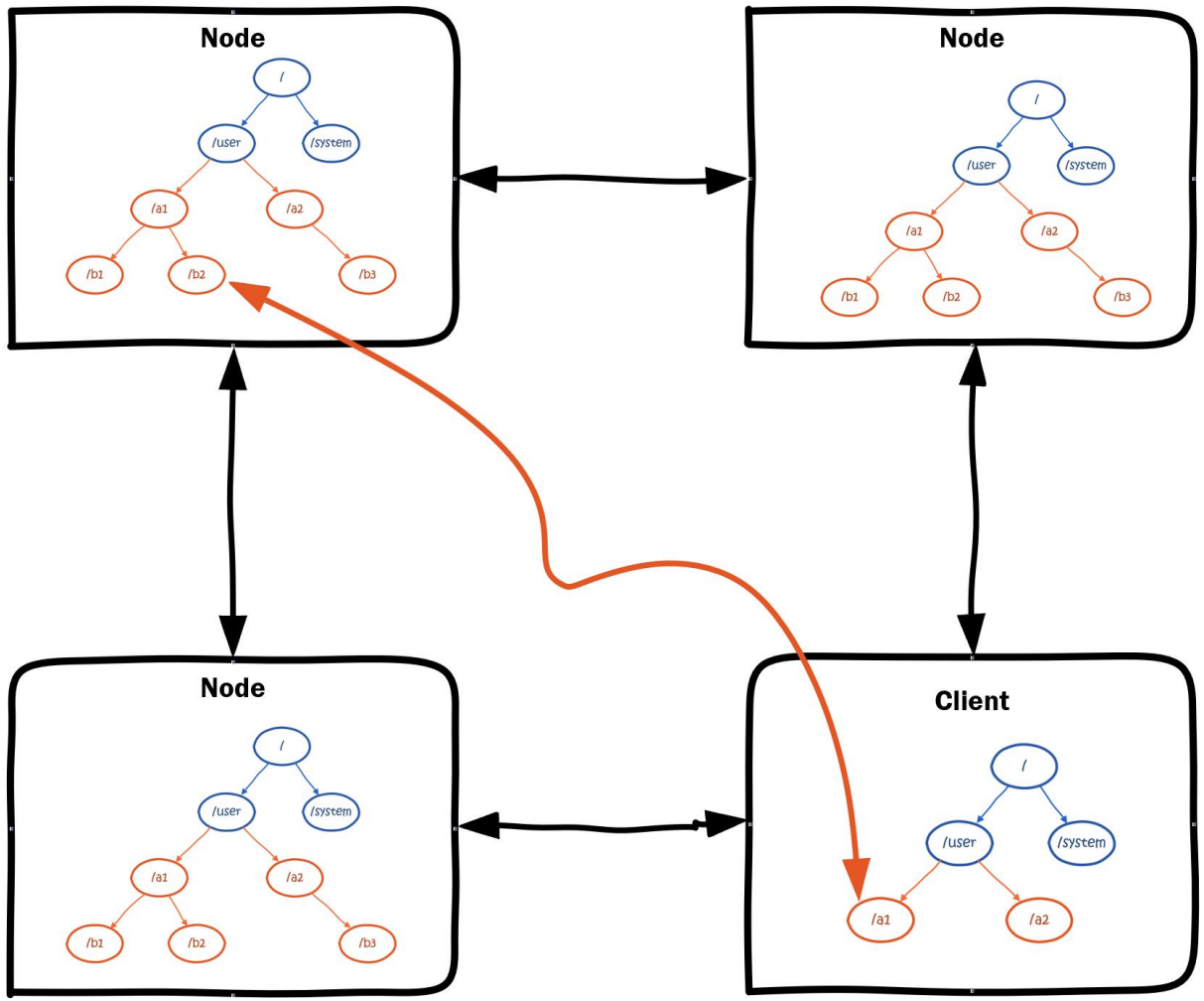
Supervision

- Errors escalated to the parent
- Parent decides OR escalates
- Action:
  - Resume
  - Stop
  - Restart
- Strategy:
  - OneForOne: only the failing actor
  - OneForAll: all children

Clients

-

Round 5



Akka.NET

- ActorSystems talk to each other
  - Akka.Remote
  - Akka.Cluster
- Clients spin up an ActorSystem
- The client Actors communicate to the other actors

Orleans

- Client Library
- Use ClusterClient to get proxies

- Proxies contact the right Silo & Grain

Persistence

-

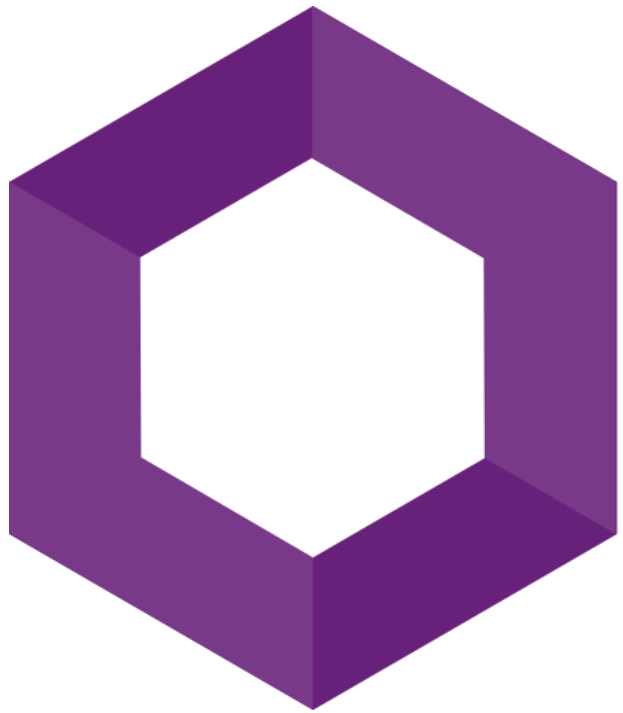
Round 6

Conclusion

–

Let’s wrap this up

And Today’s winner is …



Orleans

akka.net

Our take on this

- Akka.NET
  - Full control
  - More explicit in code
  - Better performance
- Orleans
  - Opinionated implementation
  - Sensible abstractions
  - Closer to the C# we're used to

→ Both will work in most situations