Actor Clustering with Docker Containers and Akka.NET

RICCARDO TERRELL

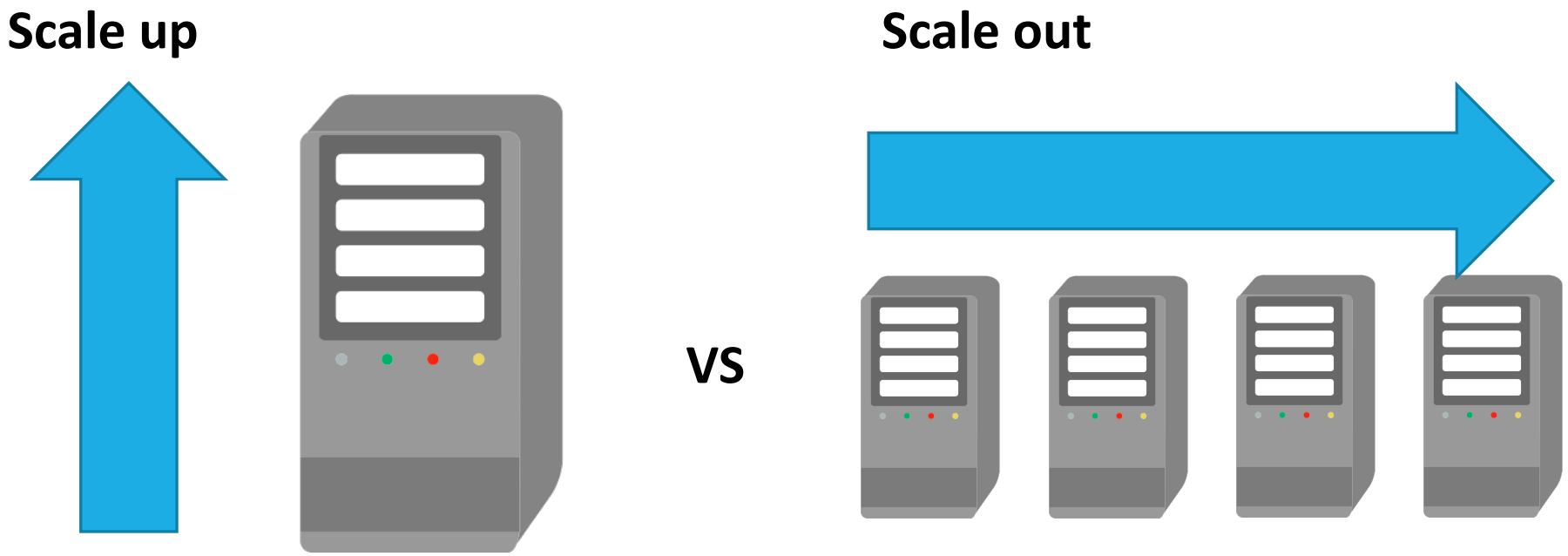
Actor Clustering with Docker Containers and Akka.NET

RICCARDO TERRELL

“Why should we use the Actor programming model?

- Fault Tolerant
- Elastic
- Decentralized
- Scalable

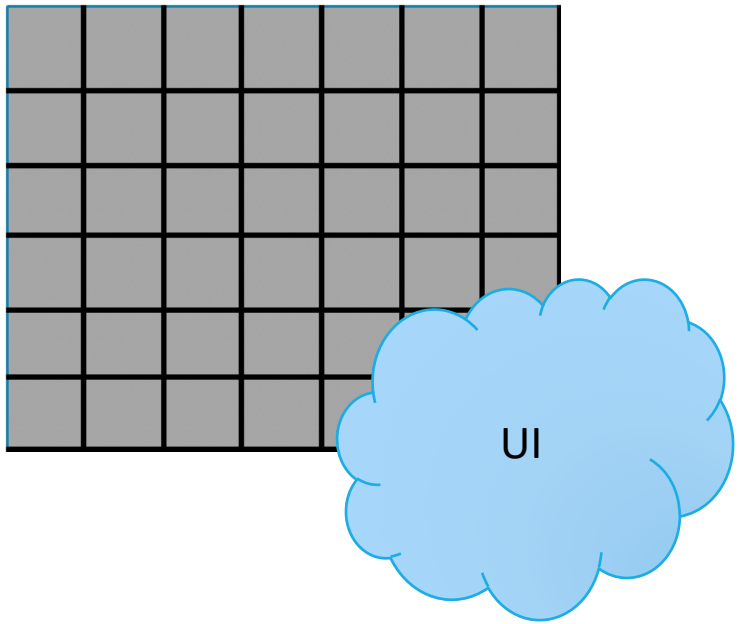
Its about maximizing resource use

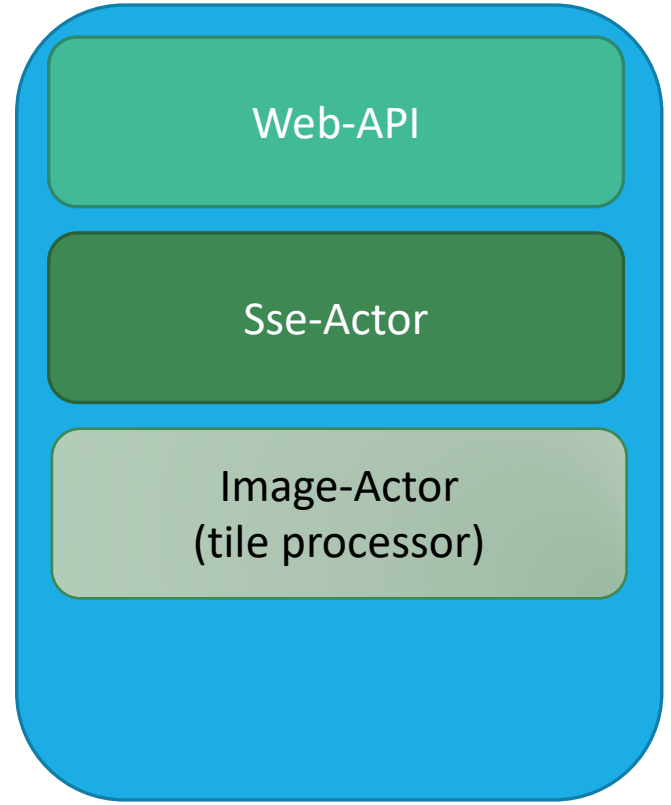


What to expect

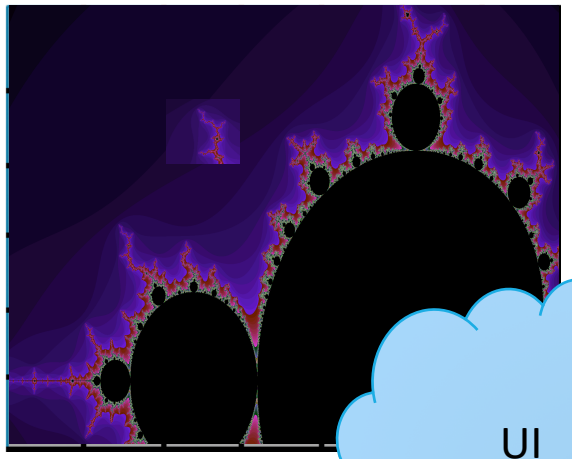
- **Akka** – core actor library
- **Akka.Persistence** – event-sourcing, durable actor state & recovery
- **Akka.Remote** – cross-node actor deployment / communication
- **Akka.Cluster** – elastic actor networks, distributed system
- **Docker** - platform-as-a-service products that use OS-level virtualization to deliver software in packages called containers.

Project Remote Distributed Image (fractal) Processing

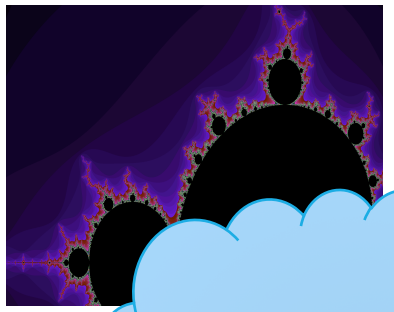




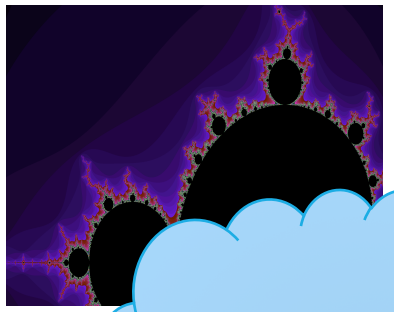
Project Remote Distributed Image (fractal) Processing



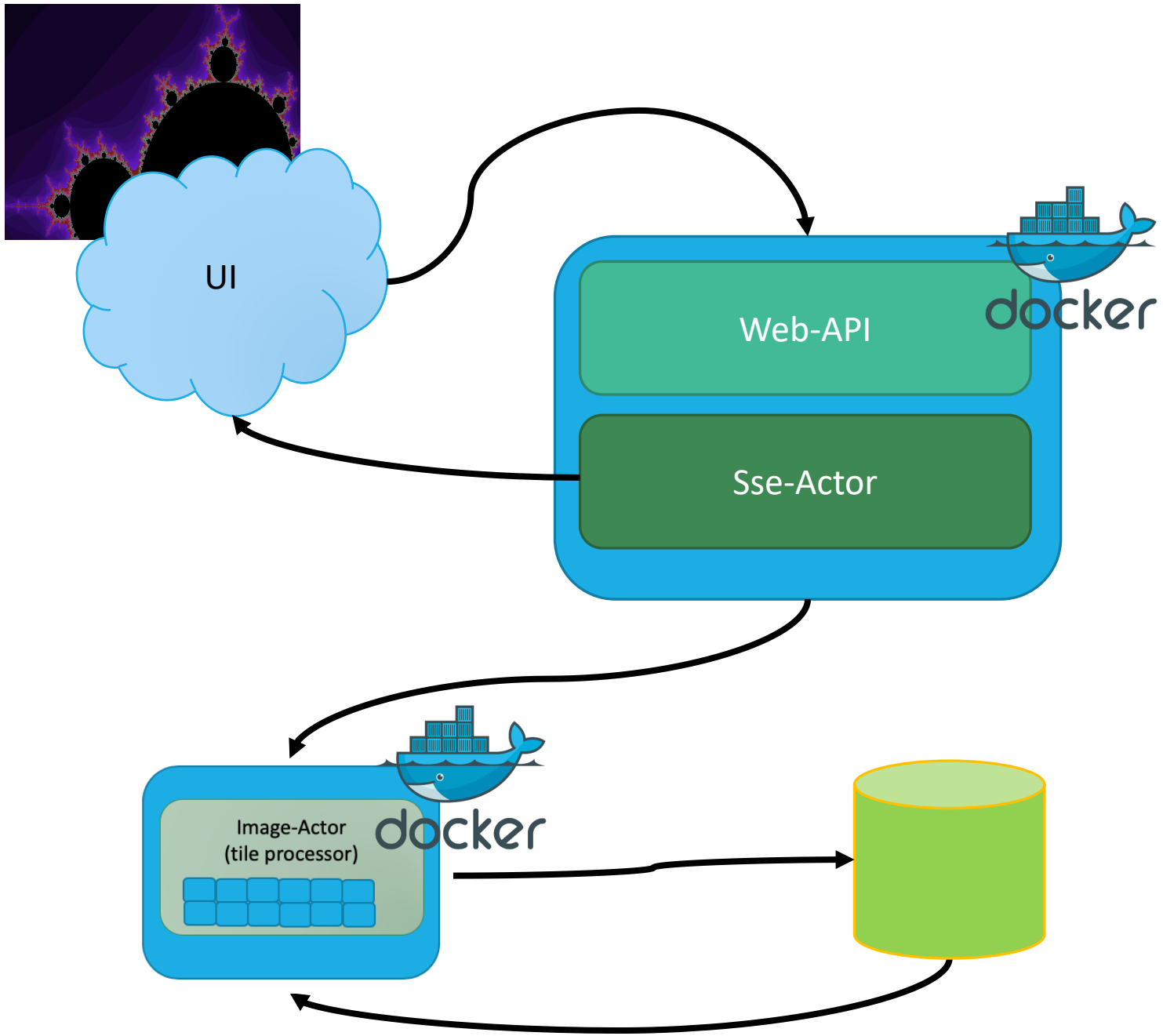
Project Remote Distributed Image (fractal) Processing



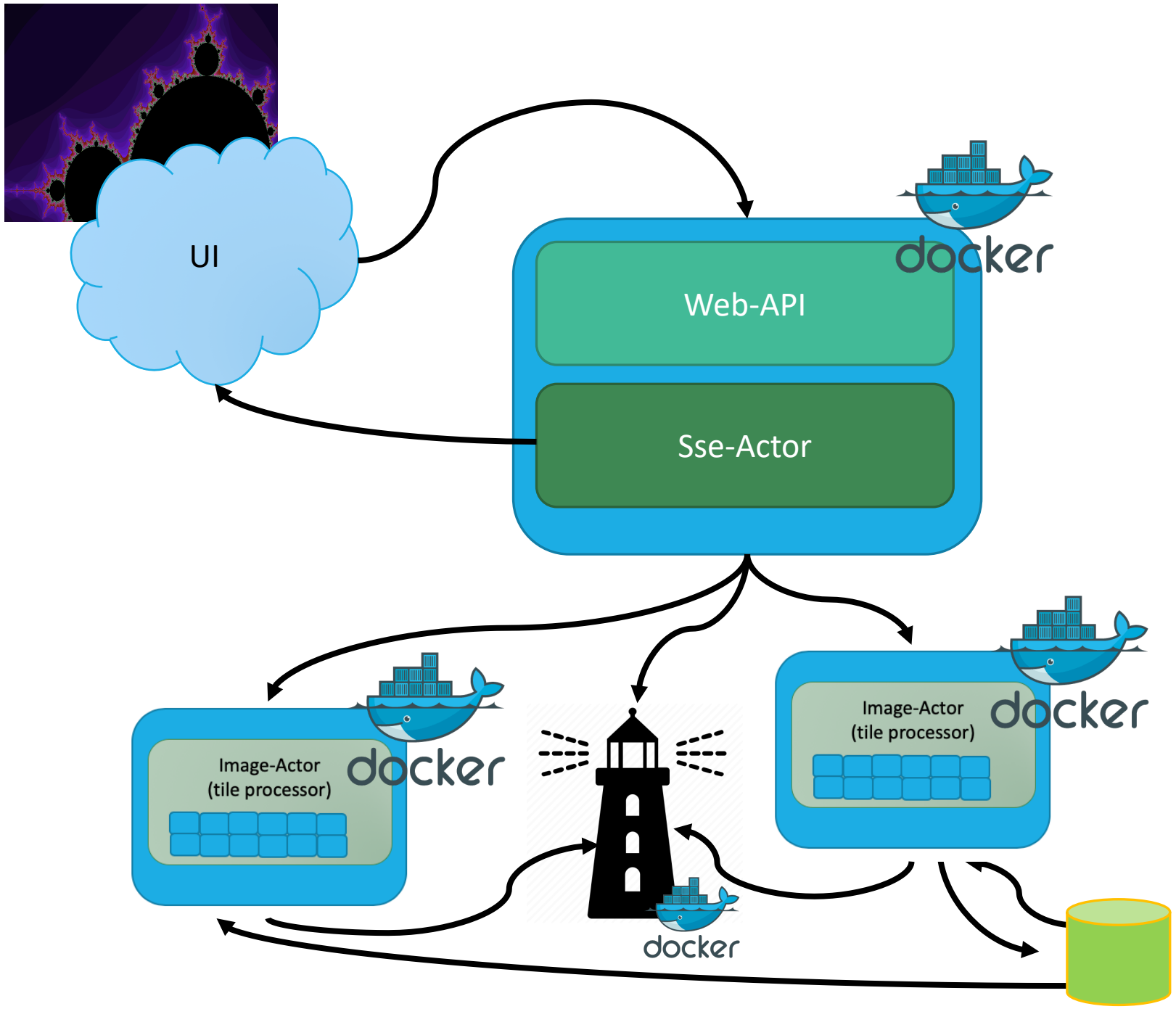
Project Remote Distributed Image (fractal) Processing



Project Remote Distributed Image (fractal) Processing

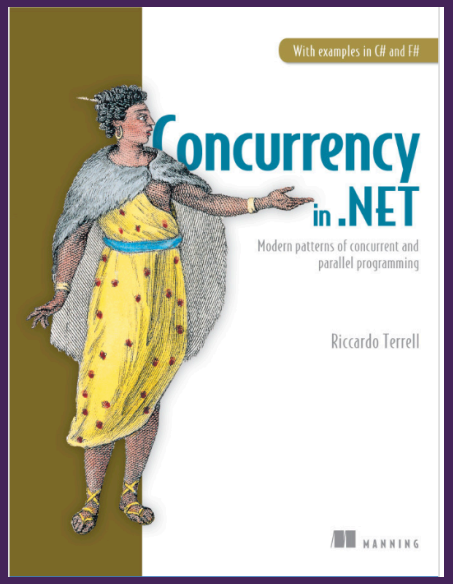


Project Remote Distributed Image (fractal) Processing



Introduction - Riccardo Terrell

- Originally from Italy, currently - Living/working in Charlotte NC
- +/- 20 years in professional programming
  - C++/VB → Java → .Net C# → Scala → Haskell → C# & F# → ??
- Author of the book “Concurrency in .NET” – Manning Pub.
- *Polyglot programmer*
  - *believes in the art of finding the right tool for the job*
- *Organizer of the Pure Functional User Group*



@trikace www.rickyterrell.com tericcardo@gmail.com

MODERN PATTERNS OF PARALLEL AND DISTRIBUTED SYSTEMS IN .NET

2DAY COURSE

THURSDAY, DEC. 5, 2019 AT 9:00 AM-FRIDAY, DEC. 6, 2019 AT 5:00 PM (EST)
RALEIGH, NC

Next course in Raleigh

December 5 and 6

use code RaleighCC

for 10% discount

Course link

https://bit.ly/2GeC3zD

For questions

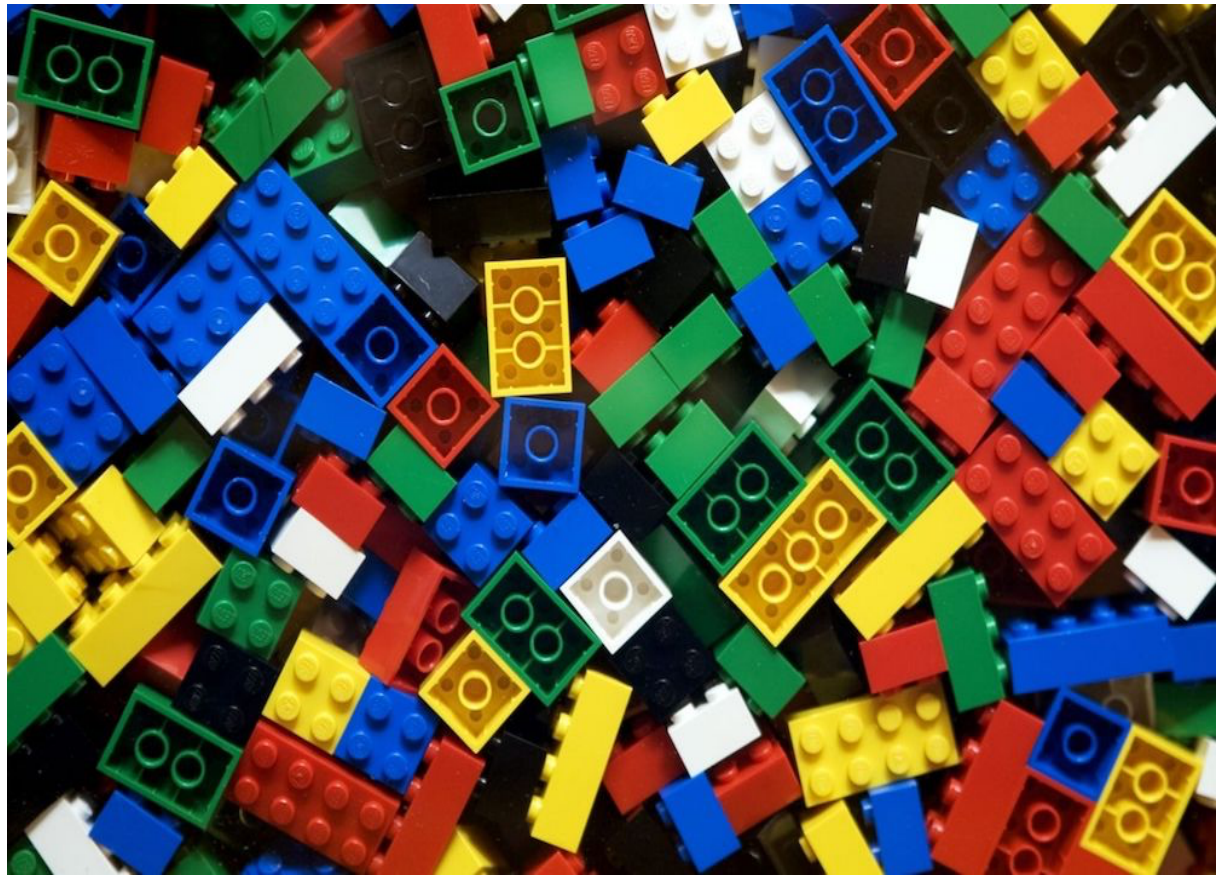
email

info@techstudiosoftware.com

Strategies to parallelize the code

The first step in designing any parallelized system is **Decomposition**.

Decomposition is nothing more than taking a problem space and breaking it into discrete parts. When we want to work in parallel, we need to have at least two separate things that we are trying to run. We do this by taking our problem and decomposing it into parts.



The issue is Shared of Memory

Shared Memory Concurrency

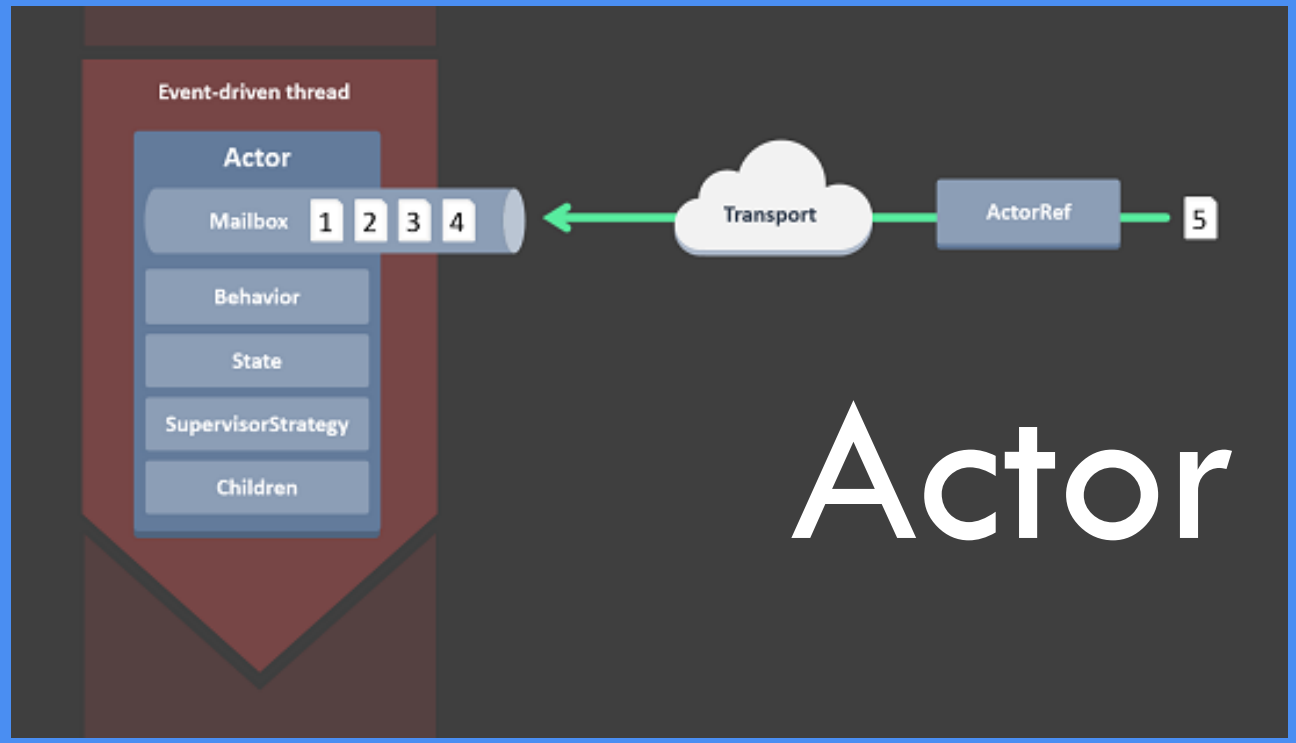
Data Race / Race Condition

Works in sequential single threaded environment

Not fun in a multi-threaded environment

Not fun trying to parallelize

Locking, blocking, call-back hell



Actor Model

Actor Model Three axioms:

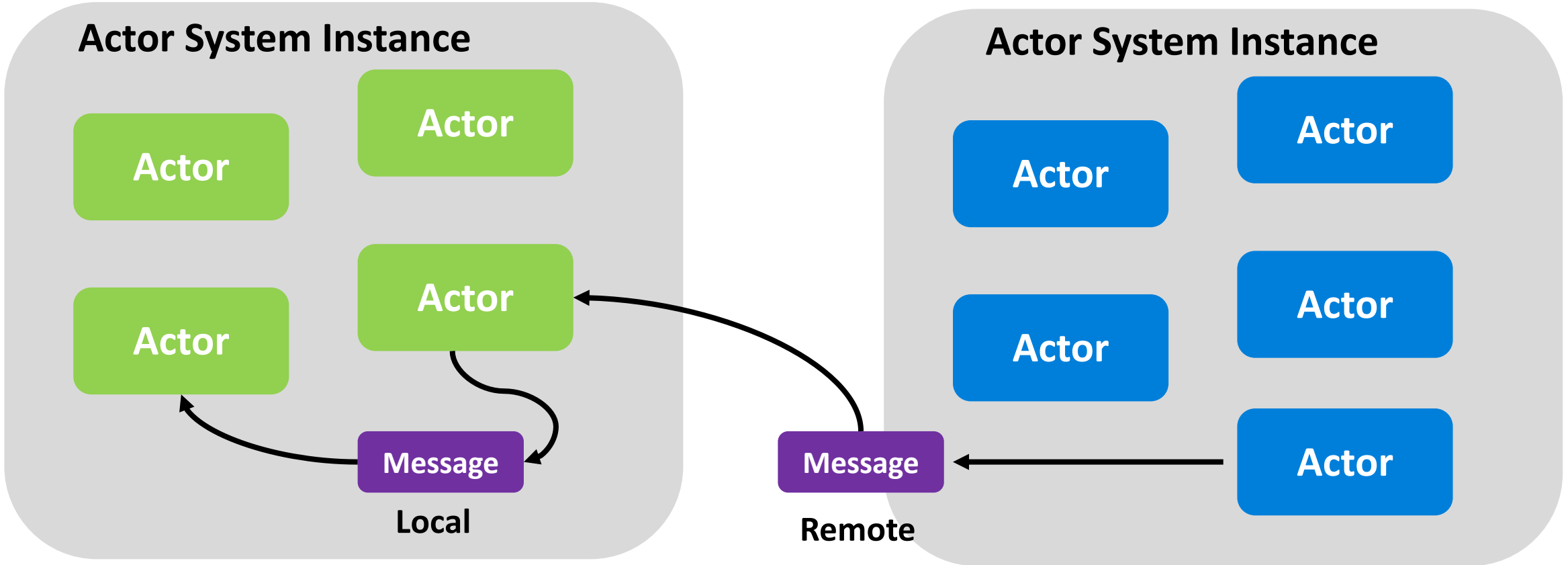
1. Send messages to other Actors

   - One Actor is not Actor -
2. Create other Actor
3. Decide how to handle the next message

What is an Actor?

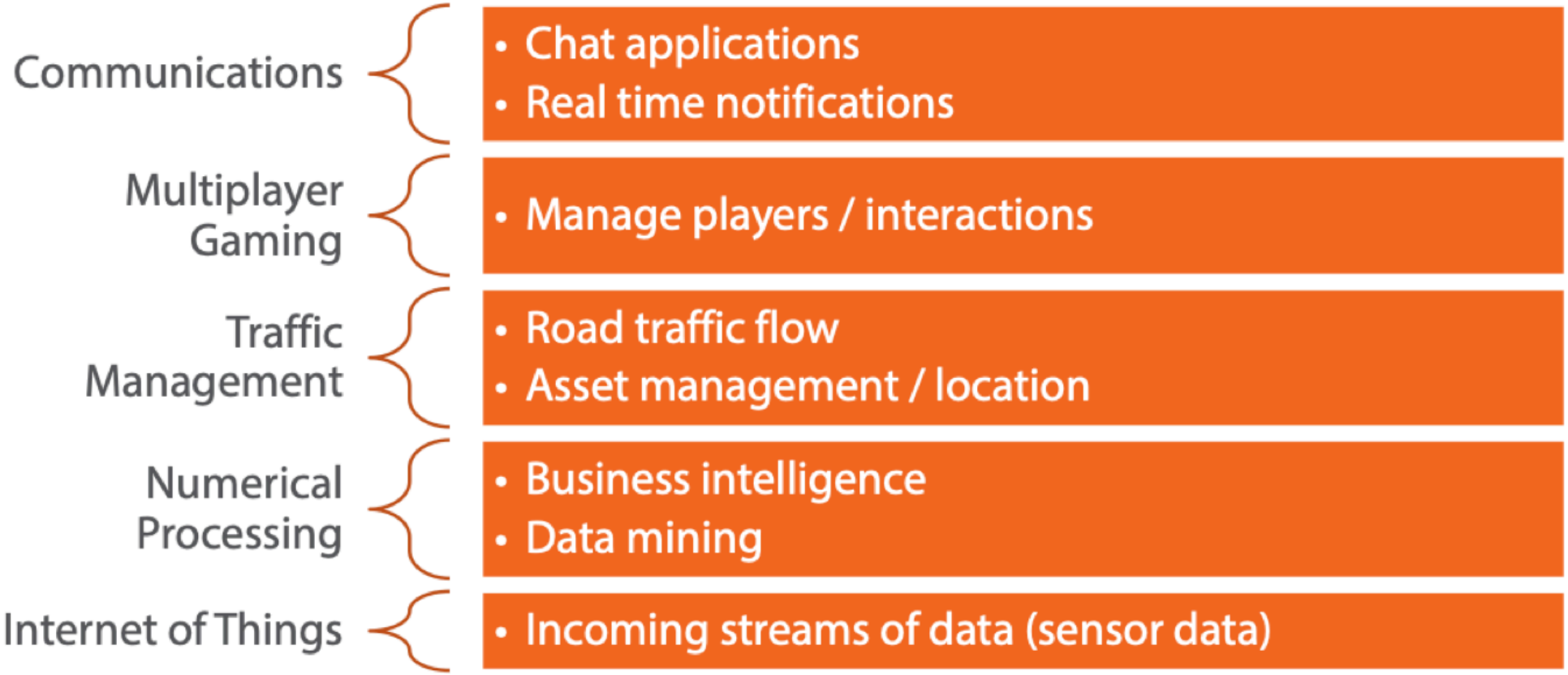
- Processing
- Storage – State
- Communication only by messages
- Share Nothing
- Message are passed by value
- Lightweight object.
- Running on it’s own thread.
- No shared state.
- Messages are kept in mailbox and processed in order.
- Massive scalable and lightening fast because of the small call stack.

Actor Systems



Classes of Applications

Classes of Applications



Akka.NET

- Akka.Remote
- Akka.Cluster
- Akka.Persistence

- Supervision
- Routing

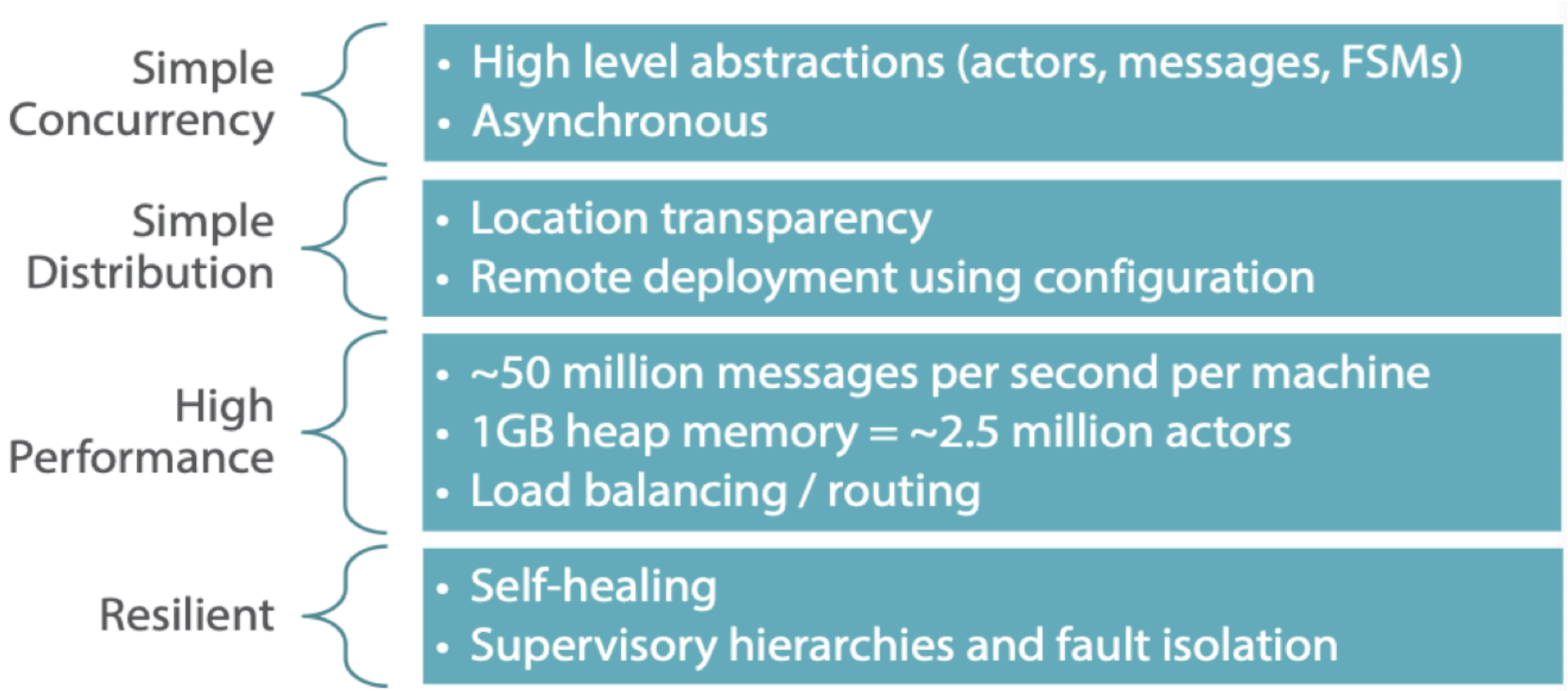
** Concurrent*

** Resilient*

** Distributed*

** Scalable*

Key Features of Akka.NET



Actor – Akka.NET in C#


```
var system = ActorSystem.Create("fizz-buzz");

public class FizzBuzzActor : ReceiveActor {
    public FizzBuzzActor()     {
        Receive<FizzBuzzMessage>(msg => {
            // code to handle the message
        });
    }
}

var actor = system.ActorOf(Props.Create<FizzBuzzActor>(), "fb-actor");
actor.Tell(new FizzBuzzMessage(5));
```

Actor – Akka.NET in F#


```
let system = ActorSystem.Create("FSharp")

type EchoServer =
    inherit Actor

    override x.OnReceive (message:obj) =
        match message with
        | :? string as msg -> printfn "Hello %s" msg
        | _ ->  printfn "What should I do with this thing?”

let echoServer = system.ActorOf(Props(typedefof<EchoServer>))
echoServer.Tell 42
```

Types of Message Sending

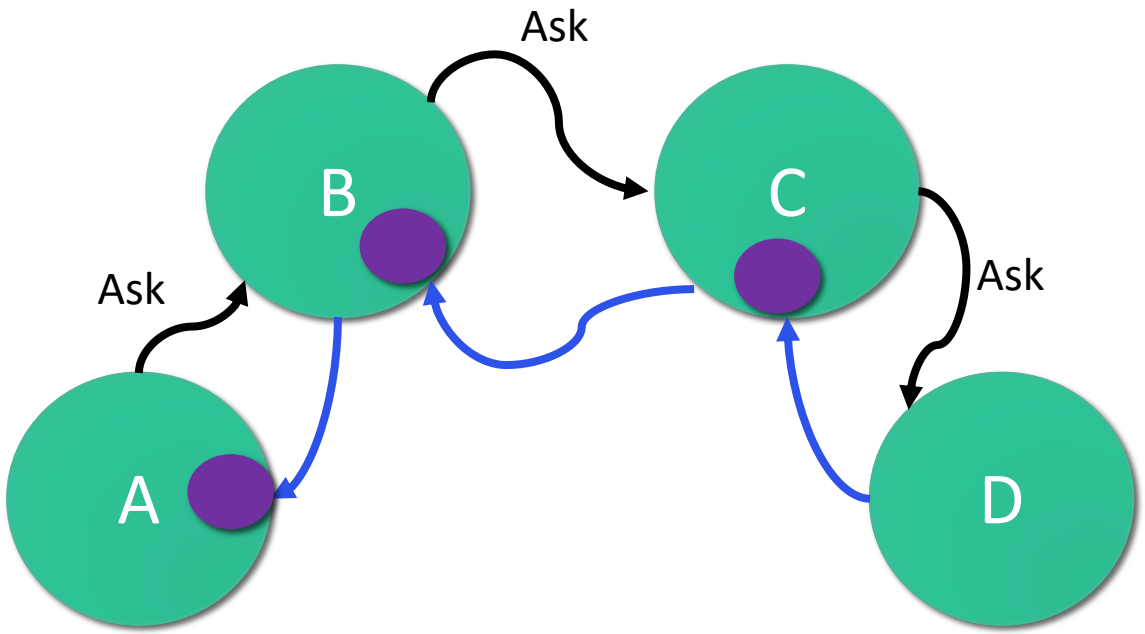
Tell

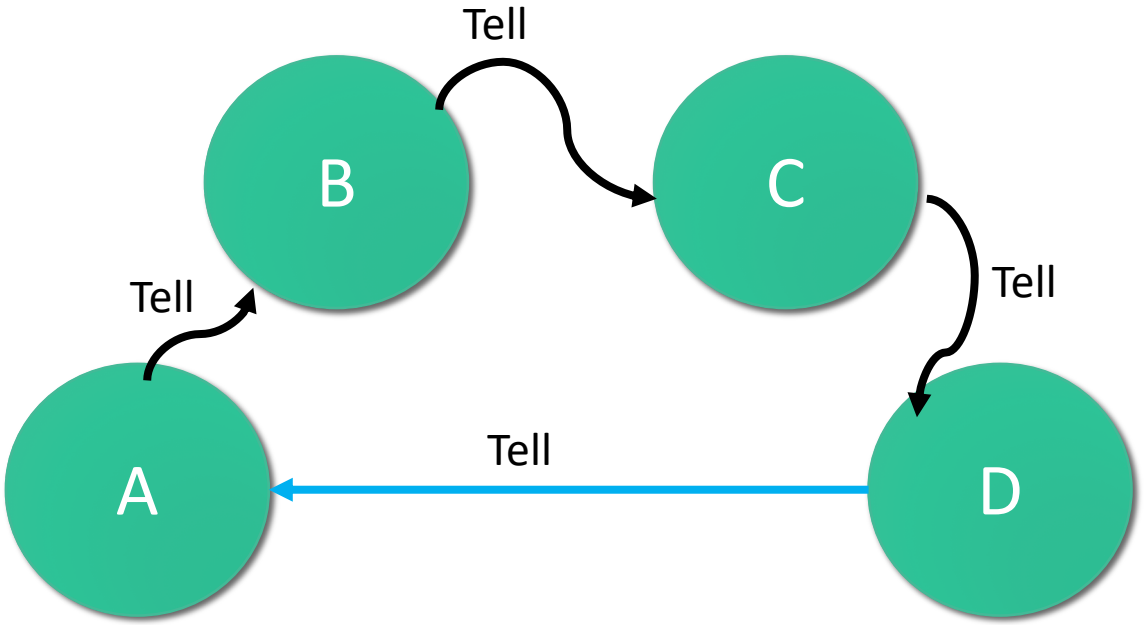
- Tell target actor something
- Don’t expect a response
- “Fire and forget”
- Doesn’t block waiting for a reply
- Better scaling and concurrency
- Use most of the time

Ask

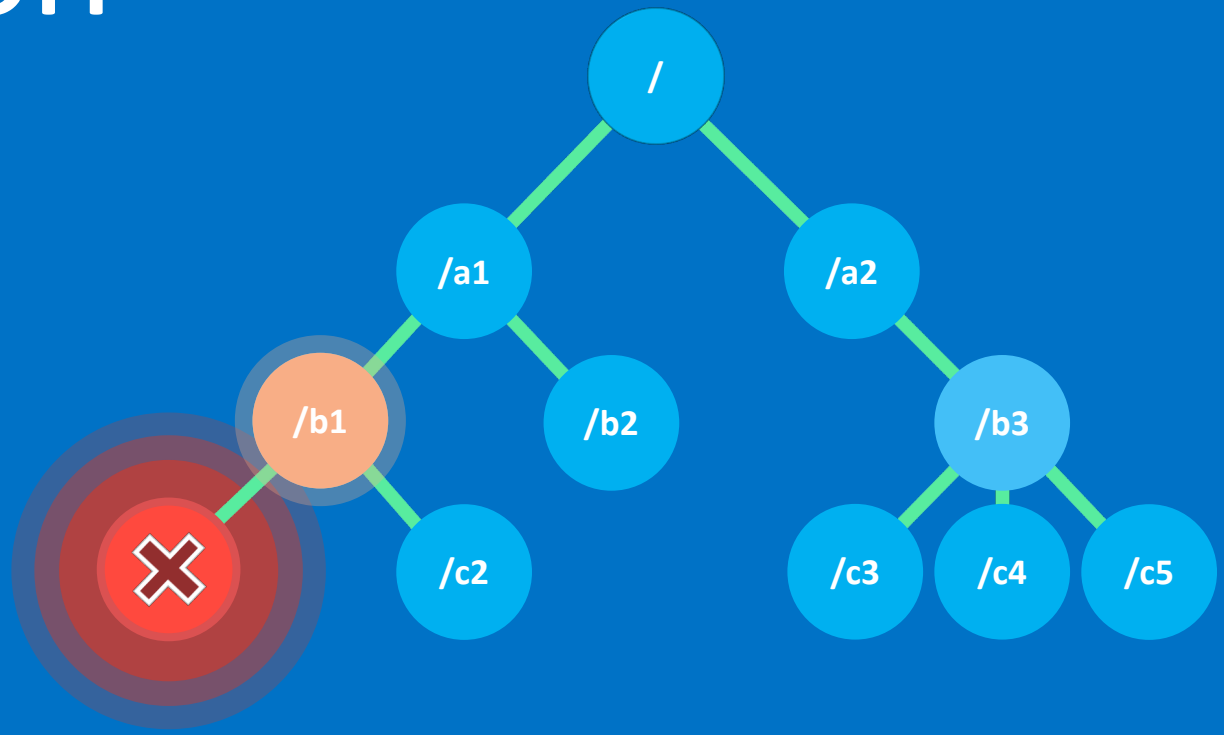
- Ask target actor for something
- Expect a response
- Target actor has to reply to sender
- Timeout (TaskCancelledException)
- Worse scaling and concurrency
- Only use when absolutely necessary

Don't Ask, Tell

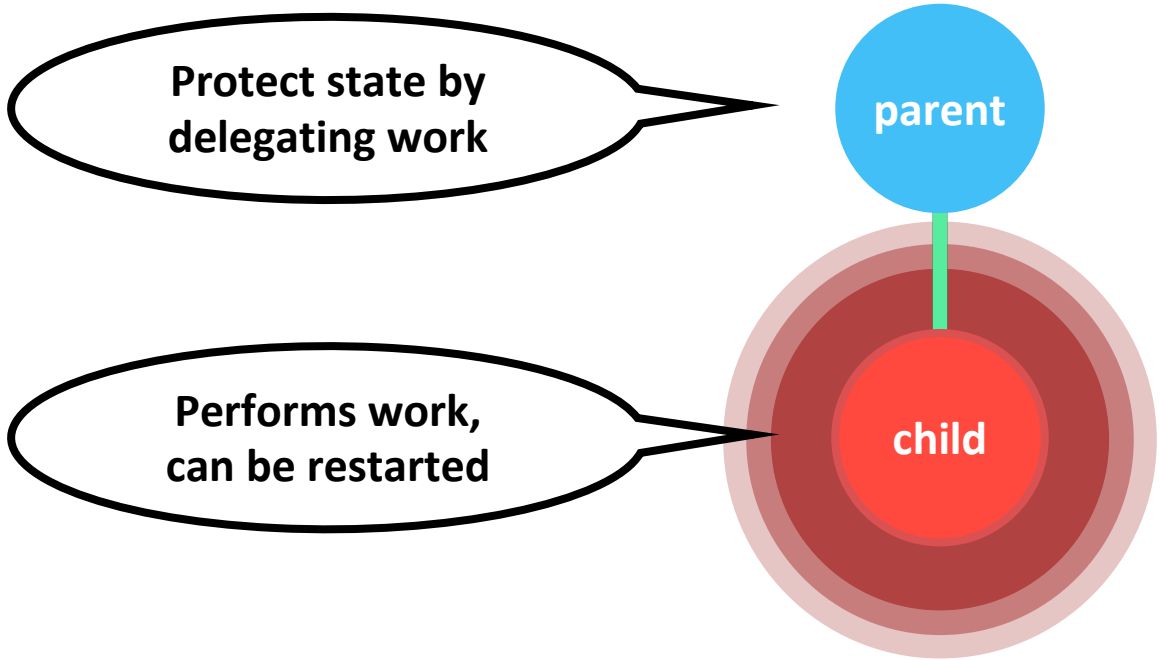


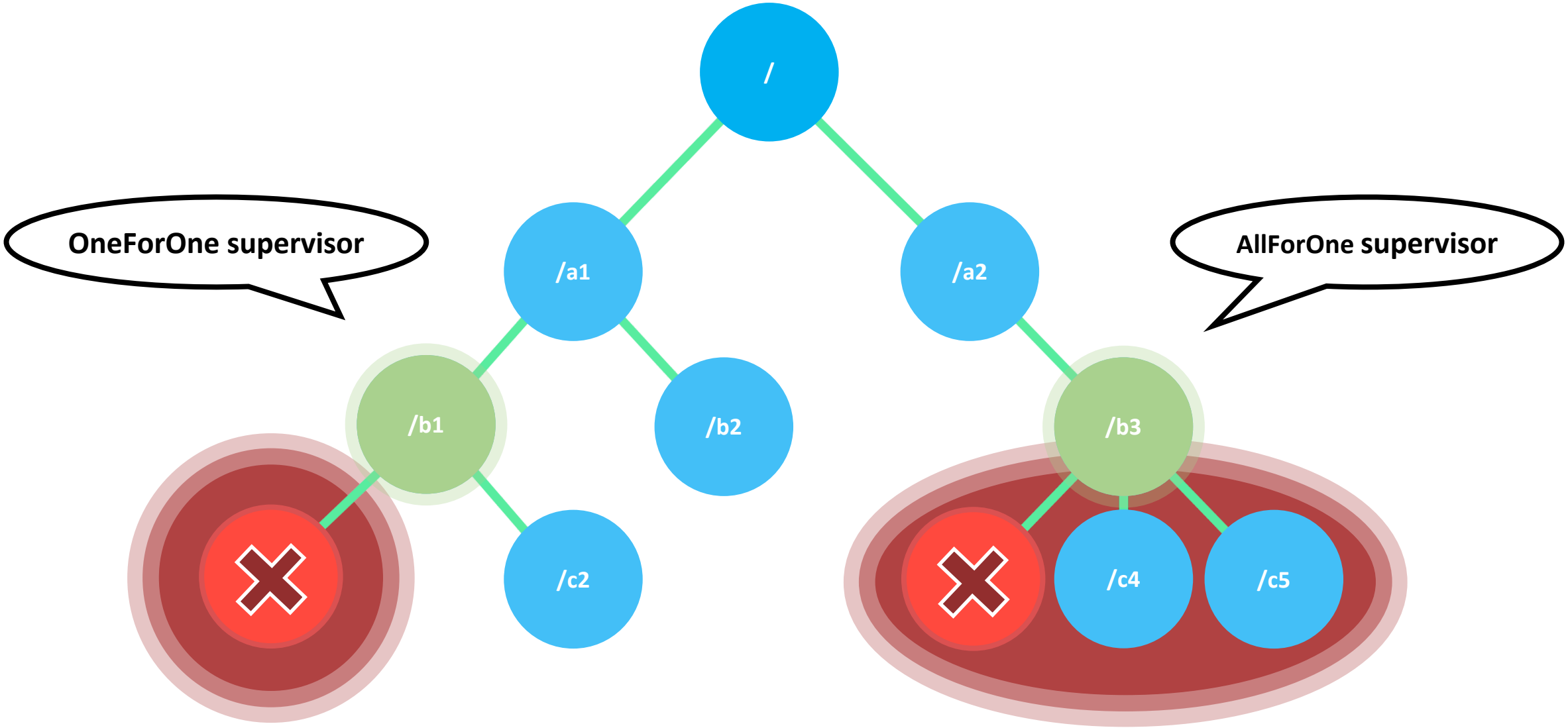


Supervision



Let it crash


Protect state by
delegating work
parent
Performs work,
can be restarted
child


/
/a1
/a2
/b1
/b2
/b3
/c2
/c4
/c5
OneForOne supervisor
AllForOne supervisor

Actor Supervision

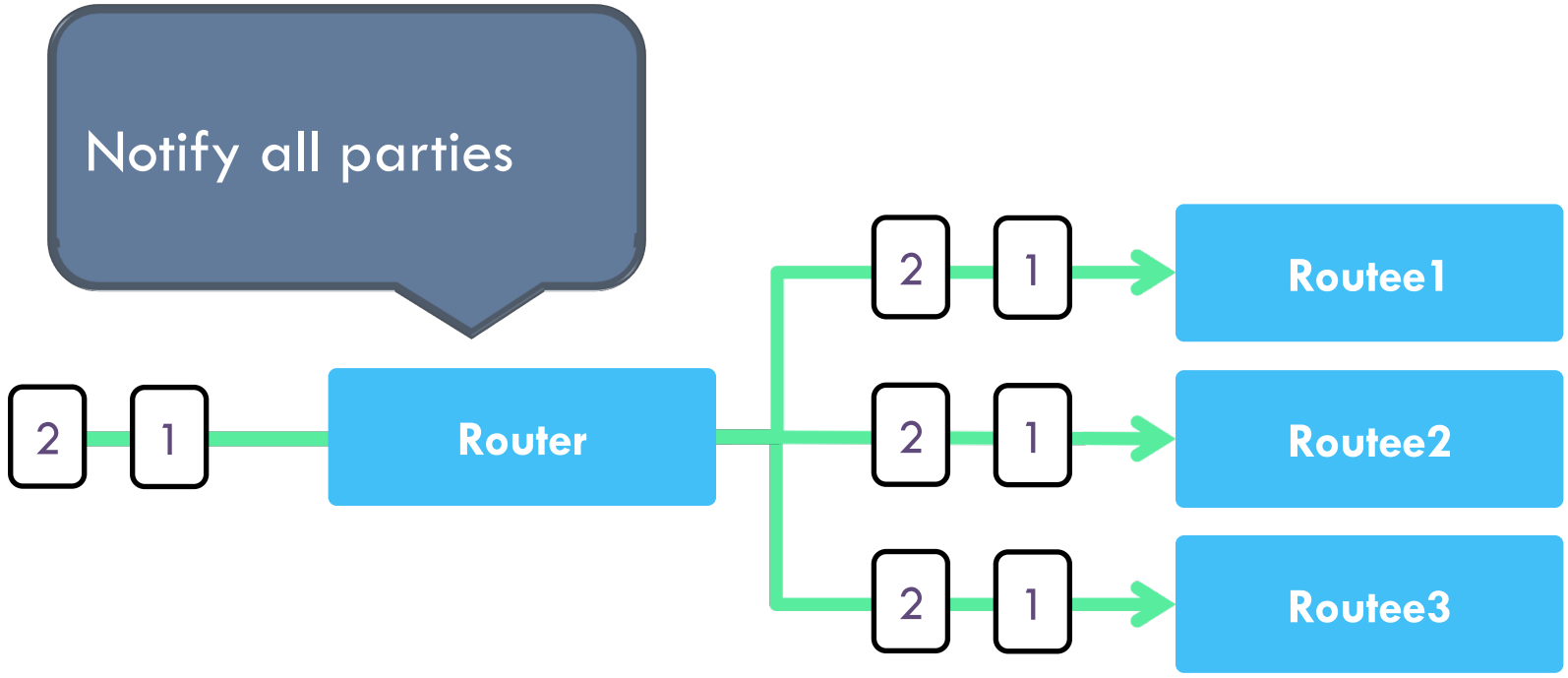
```
protected override SupervisorStrategy SupervisorStrategy()
{
    return new OneForOneStrategy(
        maxNrOfRetries: 10,
        withinTimeRange: TimeSpan.FromMinutes(1),
        localOnlyDecider: ex =>
        {
            if (ex is ArithmeticException)
            {
                return Directive.Resume;
            }

            return
Akka.Actor.SupervisorStrategy.DefaultStrategy.Decider.Decide
(ex);
        });
}
```

Akka.Game

Routing

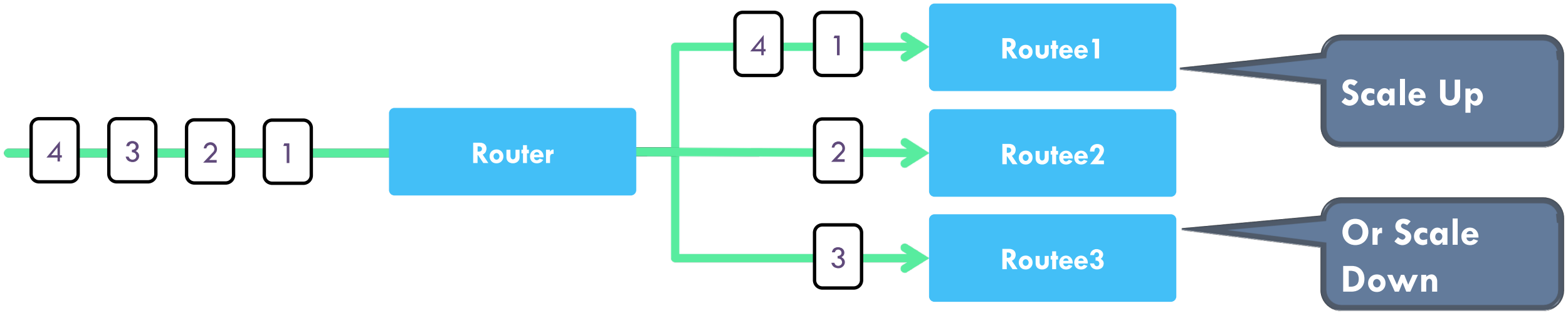
Akka.Net Routing Strategies

Broadcast router will as the name implies, broadcast any message to all of its routees



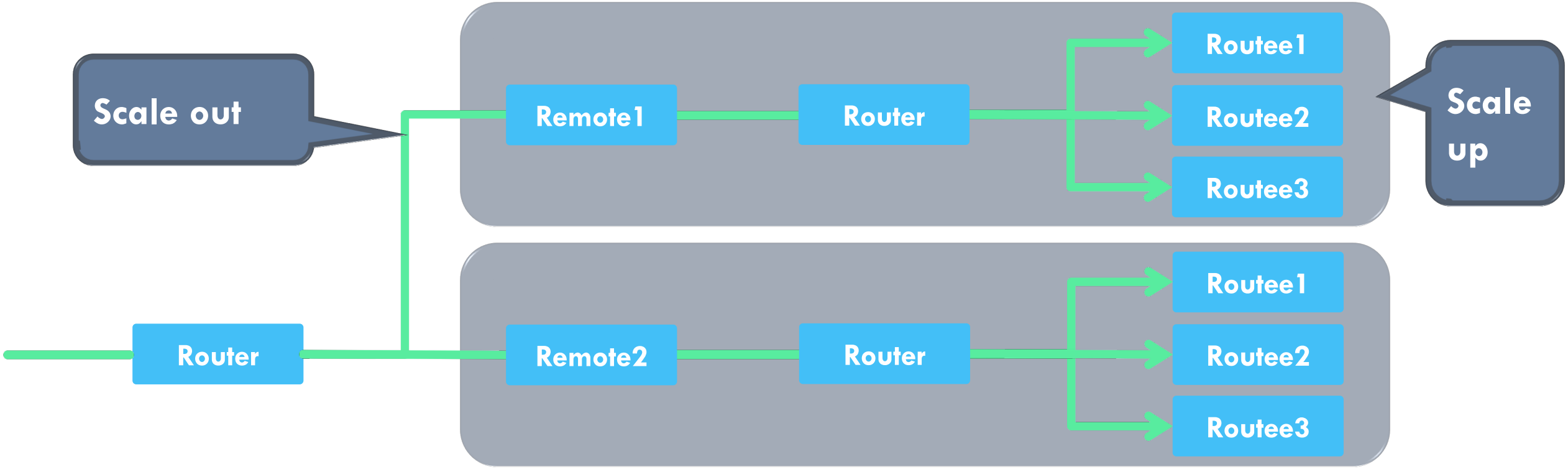
Akka.Net Routing Strategies

RoundRobin-Pool router uses round-robin to select a connection. For concurrent calls, round robin is just a best effort.



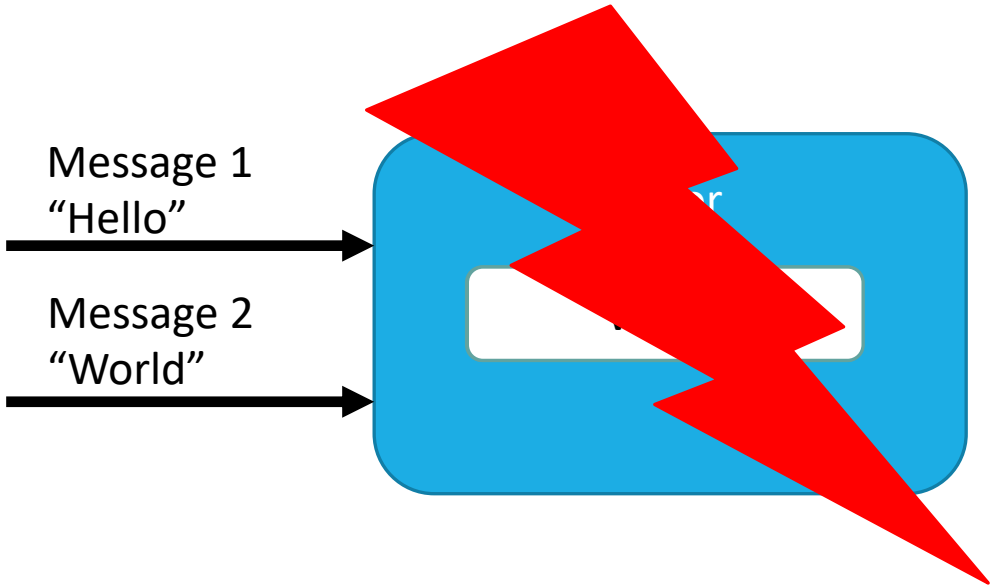
Akka.Net Routing Strategies

RoundRobin-Group router uses round-robin to select a connection. For concurrent calls, round robin is just a best effort.



Actor Persistence

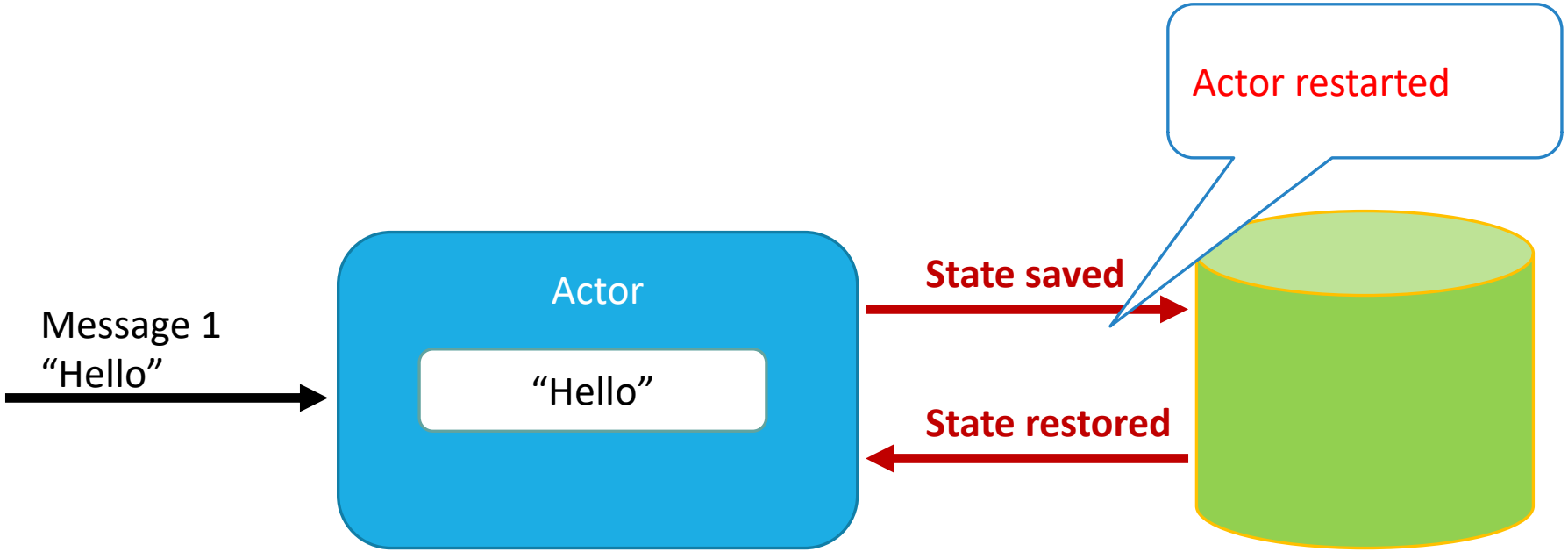
Actor persistence



If for example there is an internal exception, the process crash and force the server and actor to restart

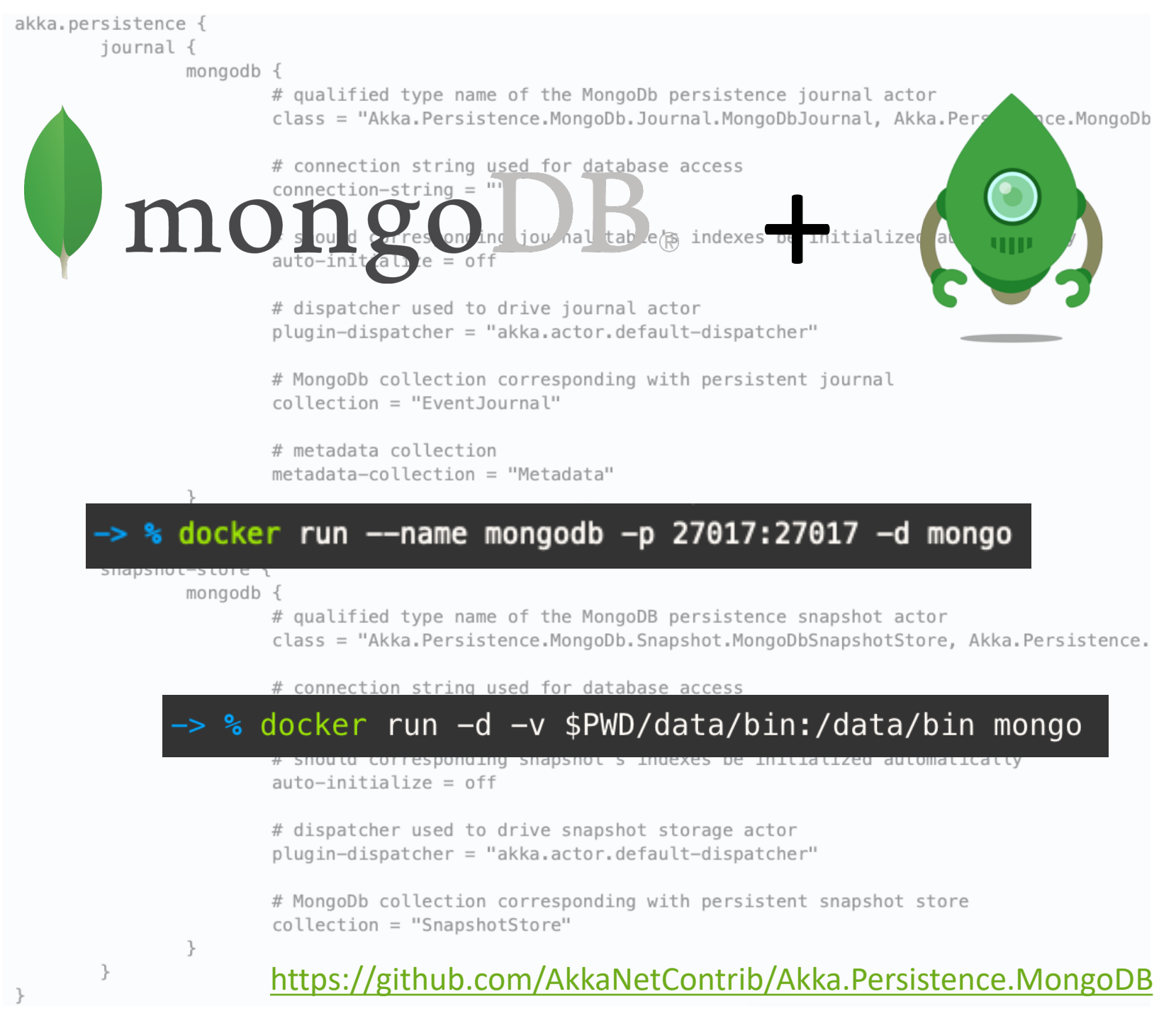
The internal stare is lost

Actor persistence



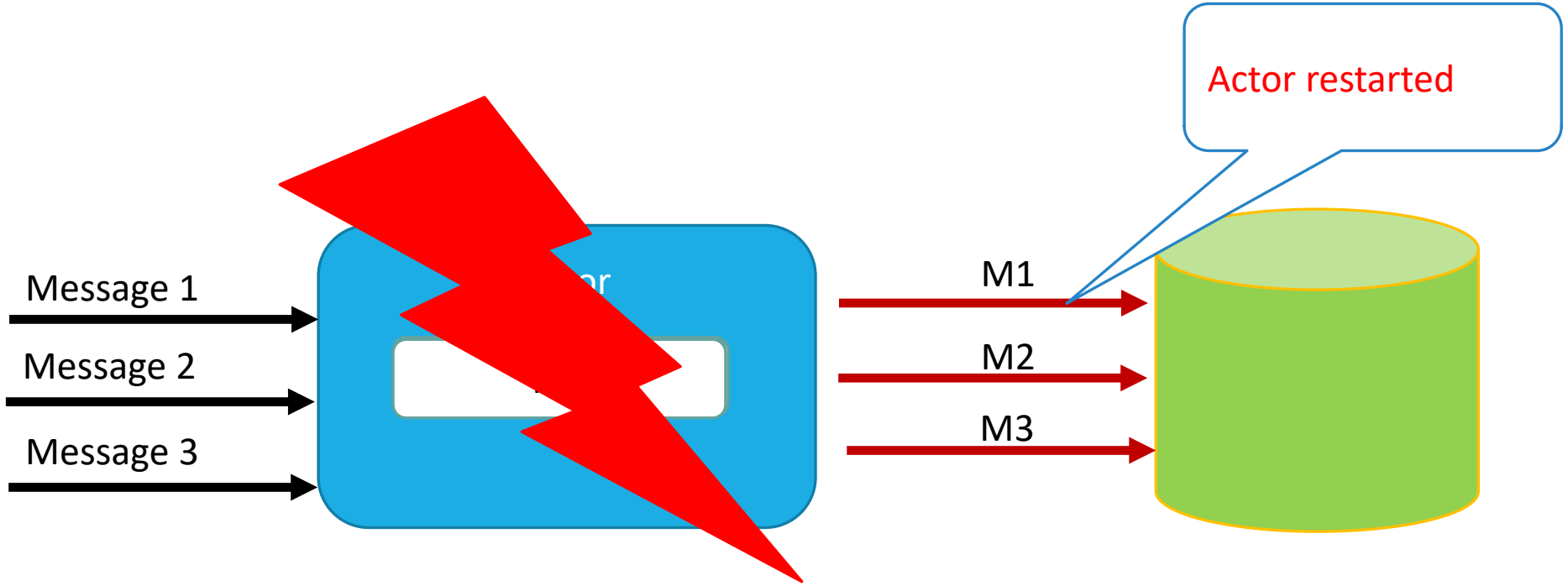
Supported Persistence Stores

- Sql
- Postgres
- Mongo
- Redis

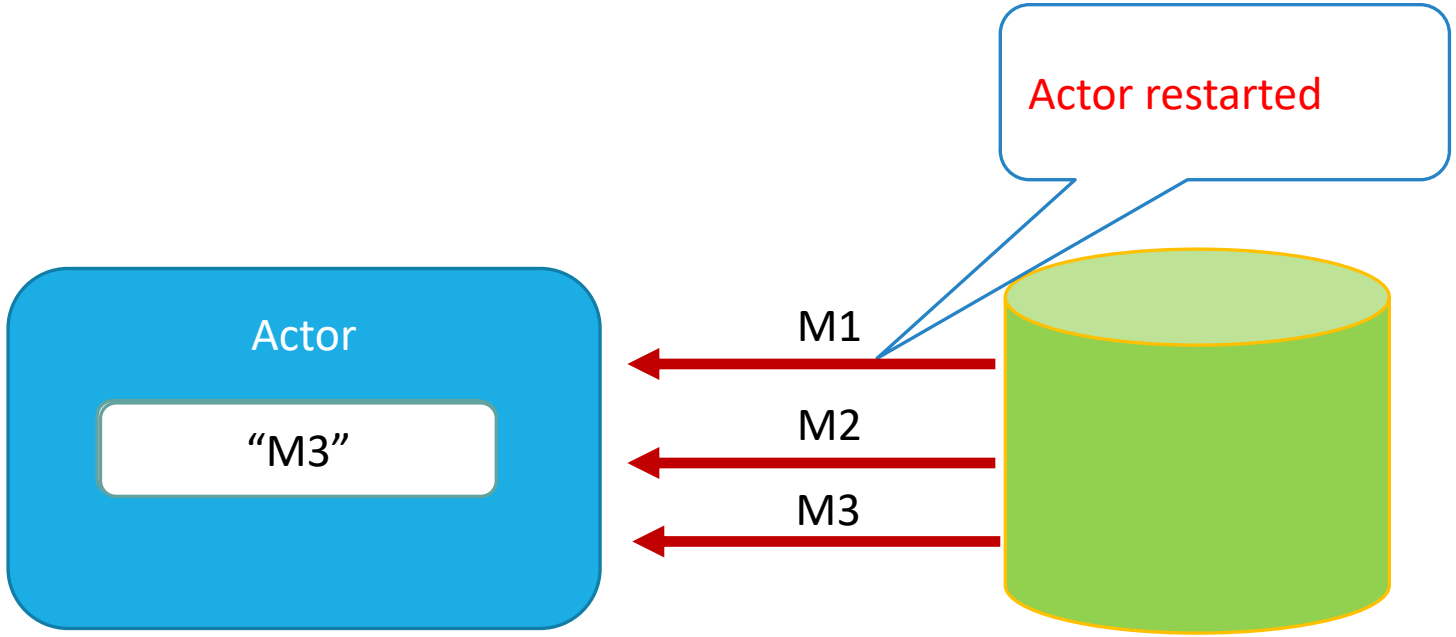


https://github.com/AkkaNetContrib/Akka.Persistence.MongoDB

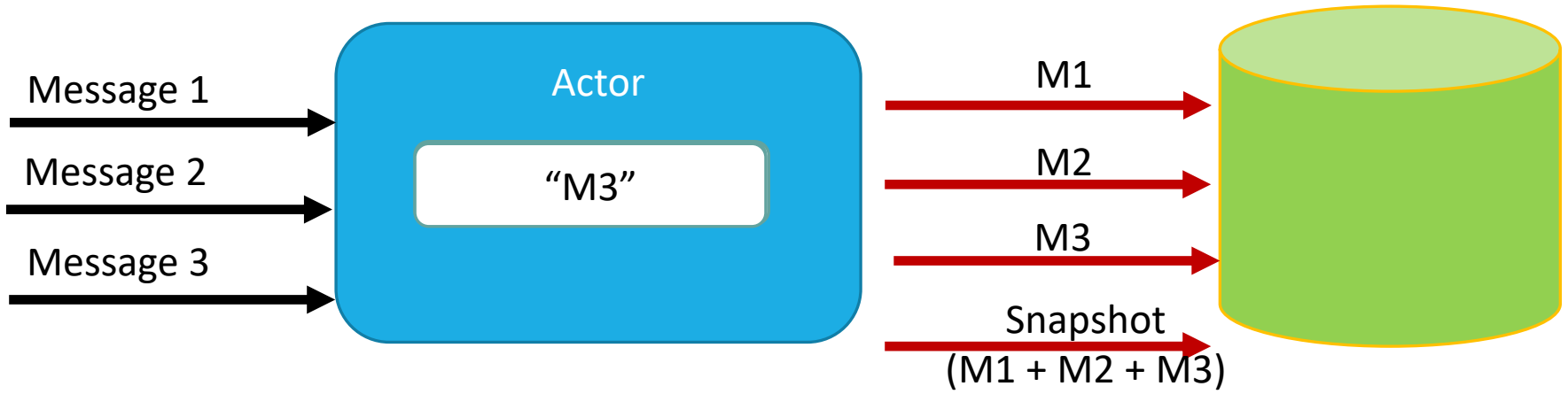
Actor persistence



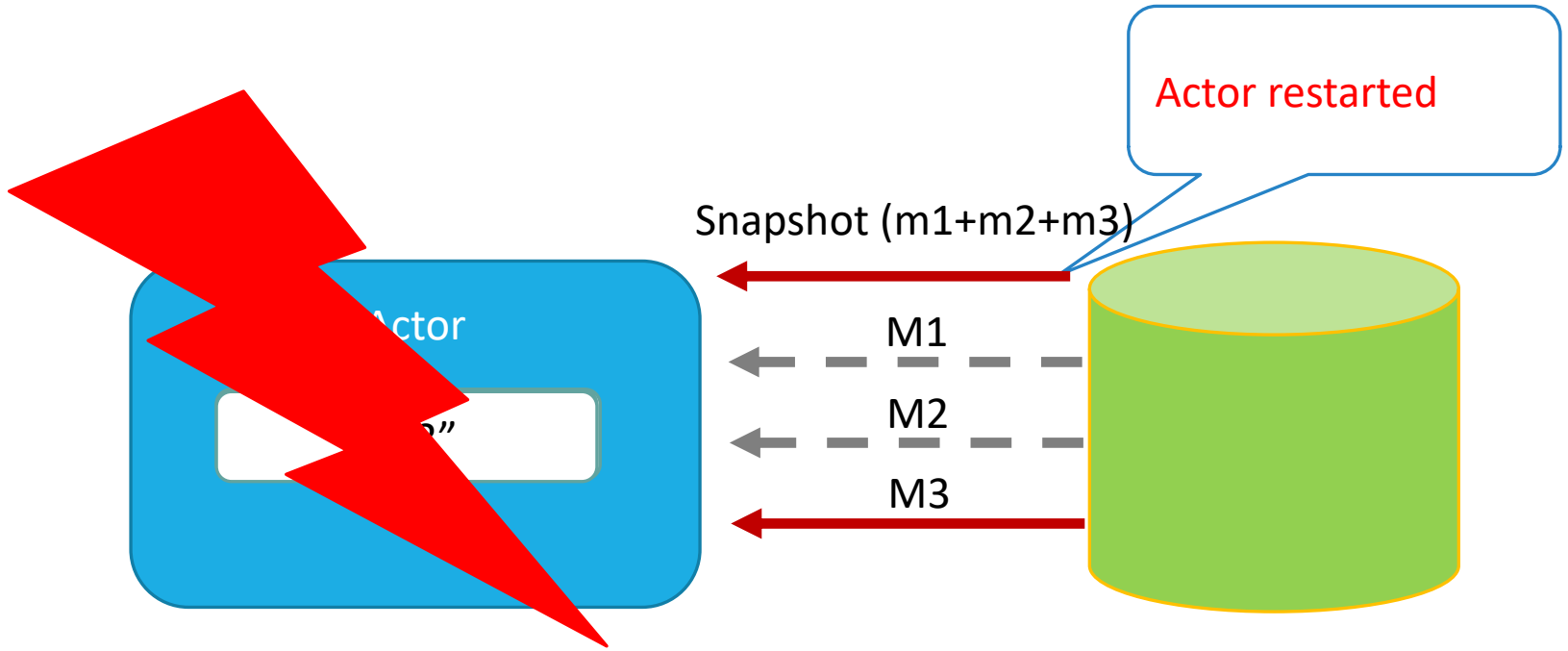
Actor persistence



Actor persistence - Snapshot



Actor persistence - Snapshot



Remoting

Location Transparency

Protocol | Actor System | Address | Path

Akka.tcp://System@10.55.1.25:9234/actorName

Remotely Deploying Actors

Deploying an actor means two things simultaneously:

Creating an actor instance with specific, explicitly configured properties

Getting an ActorRef to that actor

```
using (var system = ActorSystem.Create("Deployer", ConfigurationFactory.ParseString(@" // CLIENT
        akka {  ... CONFIG AS BEFORE ... }
            remote {
                helios.tcp {
                    port = 0
                    hostname = localhost
                }
            }
        }")))
    {
    var remoteEcho1 = system.ActorOf(Props.Create(() => new EchoActor()), "remoteecho");

    var echoActor = system.ActorOf(Props.Create(() => new HelloActor(remoteEcho1)));
    echoActor.Tell("Hello")
```

Remote deployment configuration

Human-Optimized Config Object Notation is a configuration format that aims to provide extensibility and flexibility, while at the same time making configuration values easy to edit by humans

Hocon configuration

```
akka {
    actor {
        provider = "Akka.Remote.RemoteActorRefProvider, Akka.Remote"
        deployment {
            /remoteactor {
                router = round-robin-pool
                nr-of-instances = 1
                remote = "akka.tcp://RemoteSystem@0.0.0.0:8090"
            }
        }
    }
    remote {
        dot-netty.tcp {
          port = 8091
          hostname = 127.0.0.1
    }
}
```

Remote node

```
let main args =
    use system = System.create "remote-system" config
    System.Console.ReadLine()
    0
```

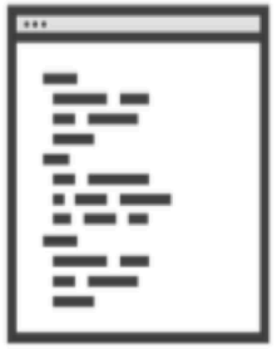
docker

What is Docker ?

- Lightweight, open, secure platform
- Simplify building, shipping, running apps
- Shipping container system for code
- Runs natively on Linux or Windows Server
- Runs on Windows or Mac Development machines (with a virtual machine)
- Relies on "images" and "containers"

Dockerfile

```
FROM mono:latest

MAINTAINER Riccardo Terrell

ENV CODEDIR /var/code
RUN mkdir -p $CODEDIR
copy . $CODEDIR
WORKDIR $CODEDIR

RUN chmod +x $CODEDIR
RUN xbuild ./ActorRemote.sln /property:Configuration=Release

EXPOSE 9234

ENTRYPOINT ["mono", "./AkkaExamples/bin/Release/AkkaExamples.exe"]
```



Dockerfile

Docker Image

Docker-Compose

```
version: "3.7"
services:
  webapp:
    build:
      context: ./dir
      dockerfile: Dockerfile-alternate
      args:
        buildno: 1
     volumes:
      - type: volume
        source: mydata
        target: /data
        port: '8090:8095'
        volume:
          nocopy: true
```

Akka.Fractal

Clustering

- Load balancing
- Fault-tolerant
- Scalability

Actor Clustering

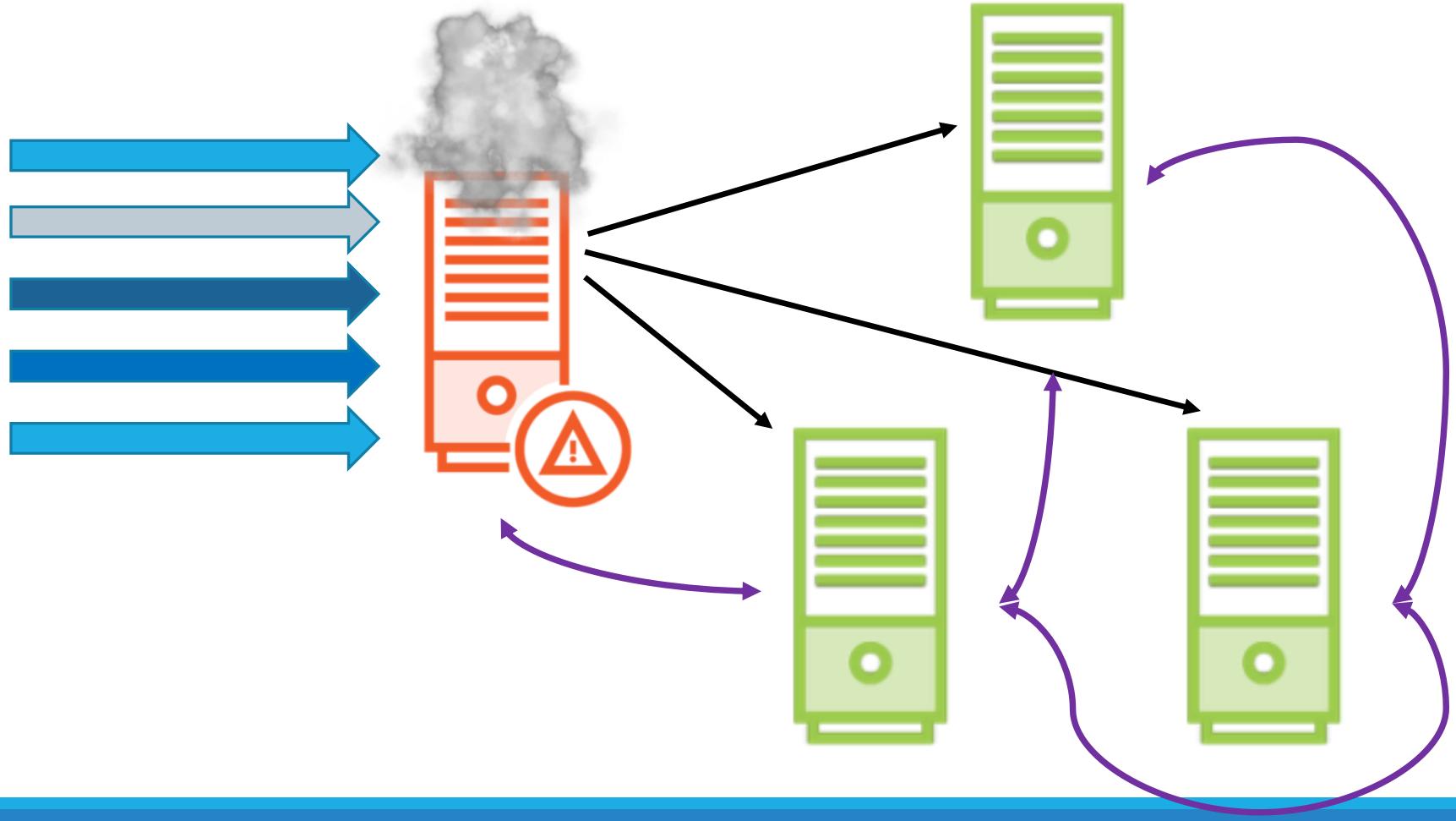
“Anything that can go wrong, will go wrong”

-- Murphy’s Law

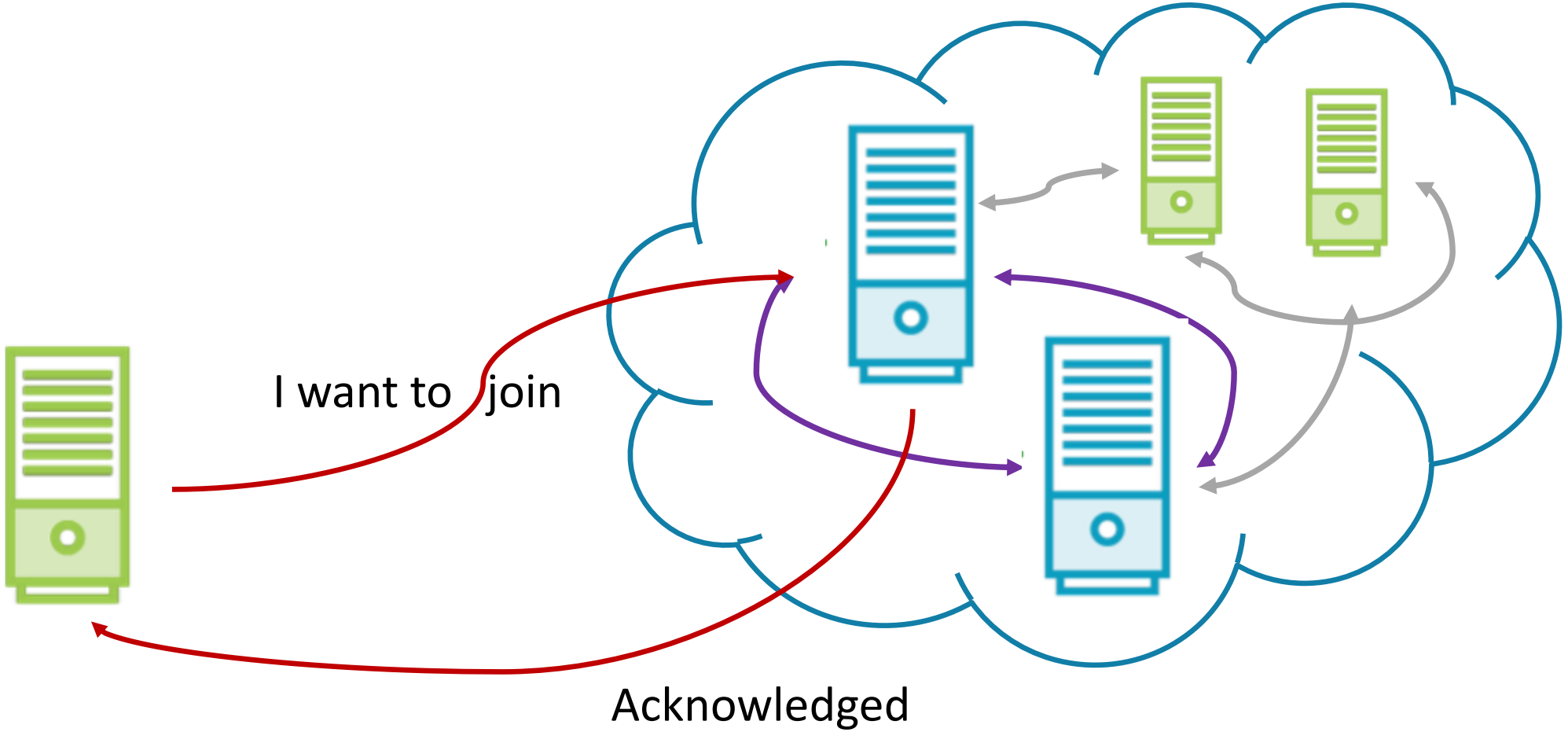
Cluster Terminologies

- **Node**: a logical member of a cluster.
- **Cluster**: a set of nodes joined through the membership service. Multiple Akka.NET applications can be a part of a single cluster.
- **Gossip**: underlying messages powering the cluster itself.
- **Leader**: single node within the cluster who adds/removes nodes from the cluster.
- **Role**: a named responsibility or application within the cluster. A cluster can have multiple Akka.NET applications in it, each with its own role. A node may exist in 0+ roles simultaneously.
- **Convergence**: when a quorum (simple majority) of gossip messages agree on a change in state of a cluster member.

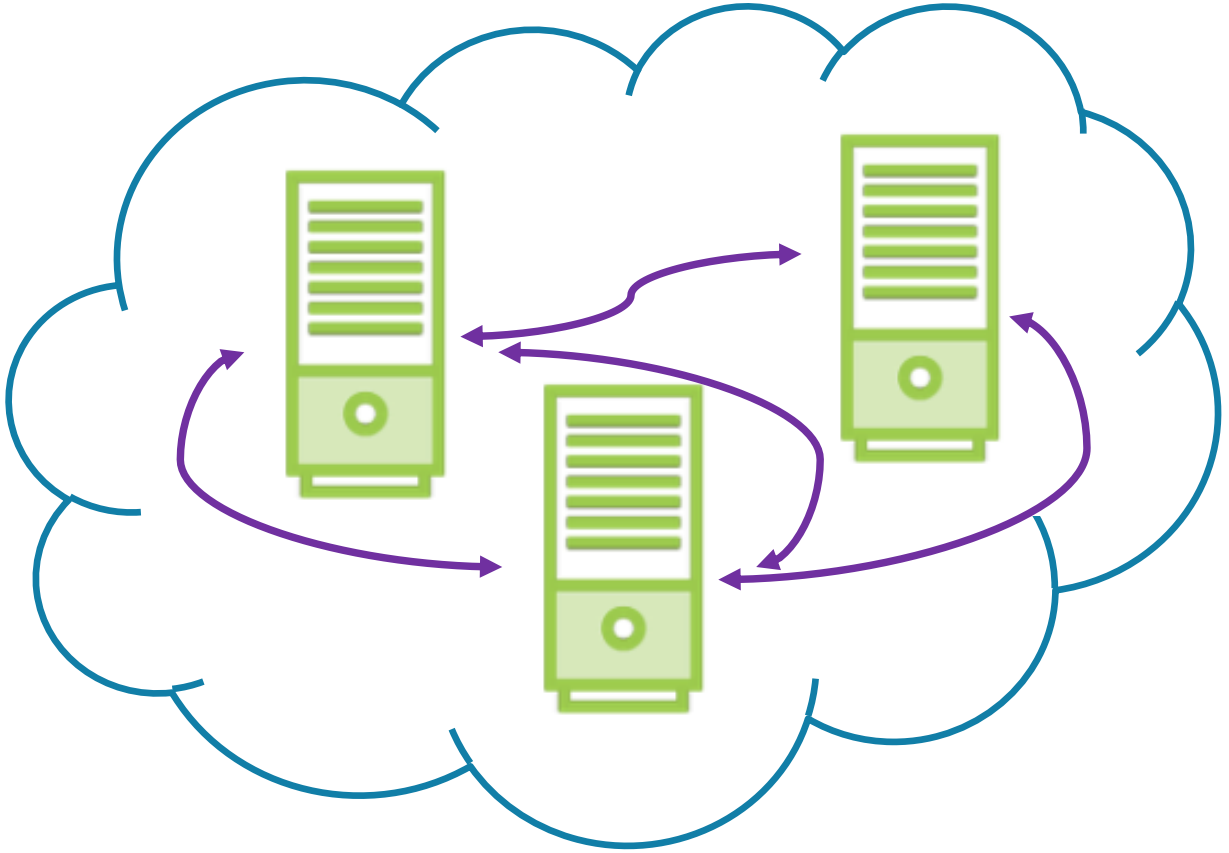
Actor Clustering



Joining the Actor Cluster



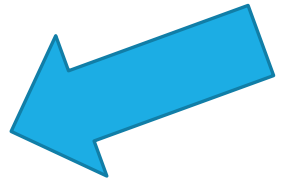
Joining the Actor Cluster



Gossip

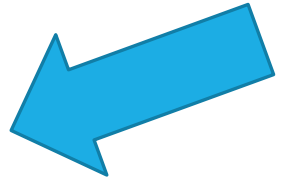
Clustering – Just a configuration matter

```
let workerSystem = System.create "cluster" <| Configuration.parse """
akka {
    actor {
        provider = "Akka.Cluster.ClusterActorRefProvider, Akka.Cluster"
        }
    }
    remote {
        helios.tcp {
            hostname = "localhost"
            port = 0
        }
    }
    cluster {
        seed-nodes = [ "akka.tcp://cluster@localhost:8091/" ]
    }
}
```



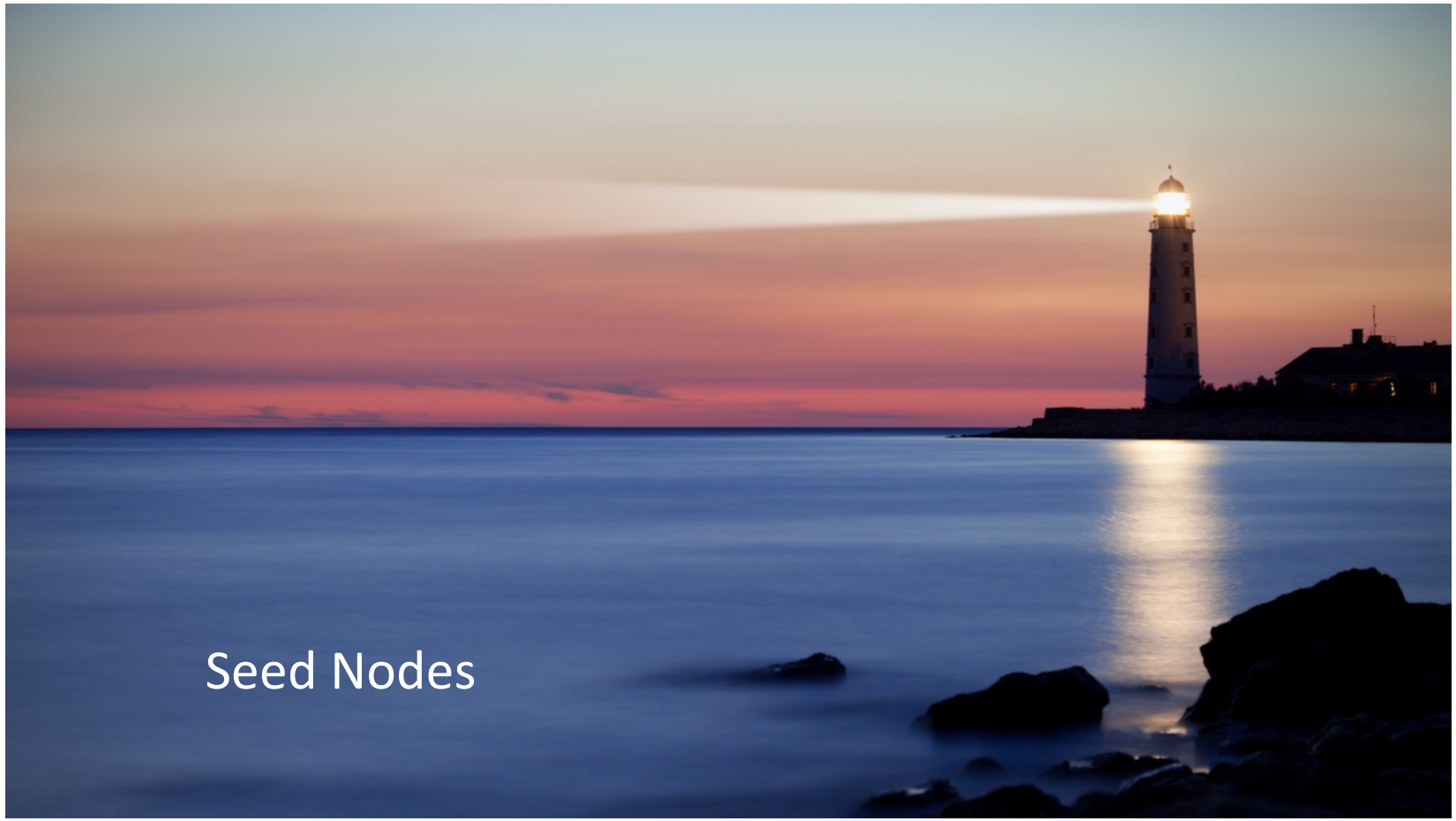
Clustering – Just a configuration matter

```
let masterSystem = System.create "cluster" <| Configuration.parse """
akka {
    actor {
        provider = "Akka.Cluster.ClusterActorRefProvider, Akka.Cluster"
        }
    }
    remote {
        helios.tcp {
            hostname = "localhost"
            port = 8091
        }
    }
    cluster {
        seed-nodes = [ "akka.tcp://cluster@localhost:8091/" ]
    }
}
```

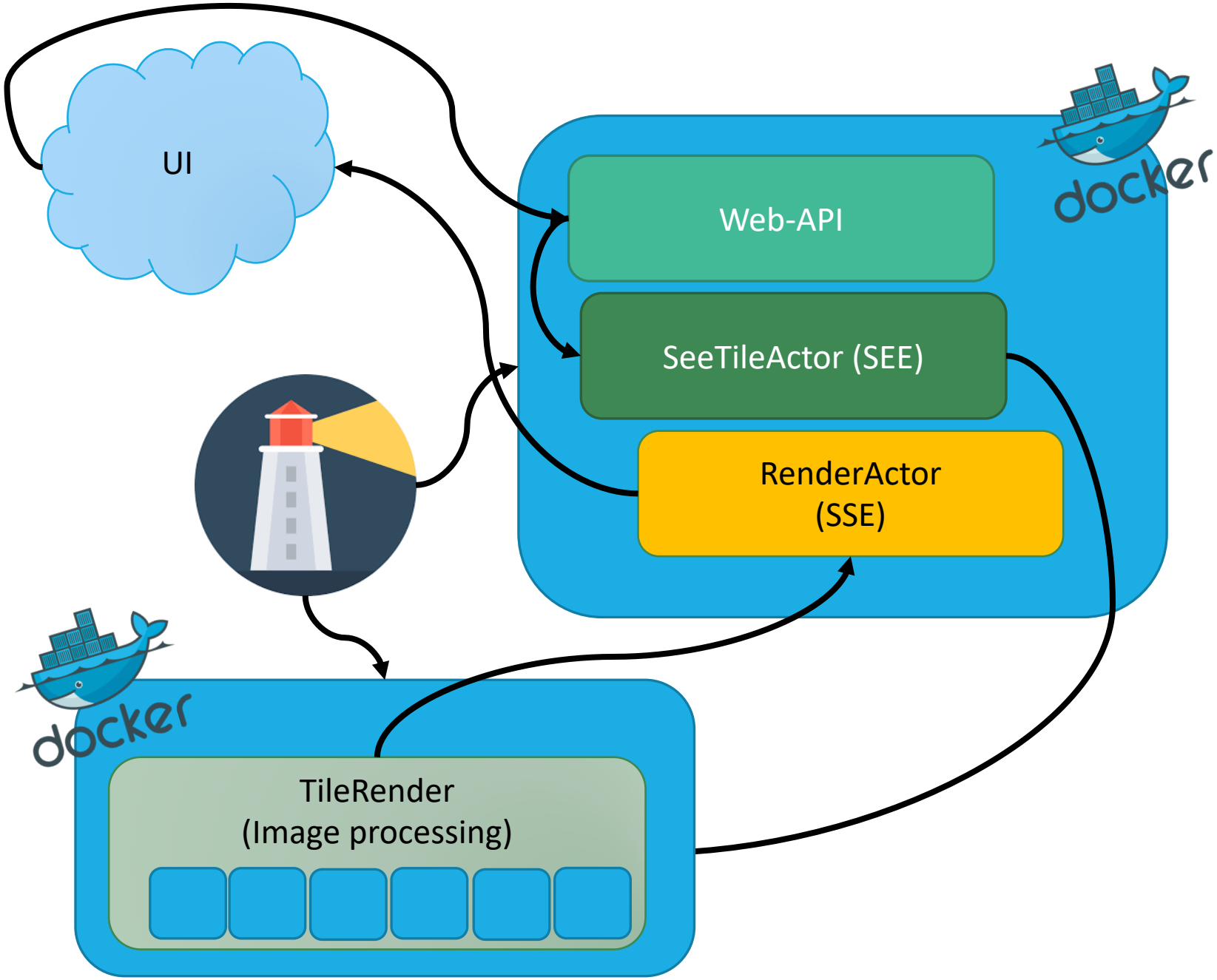


Lighthouse

Lighthouse dedicated is a simple but effective seed node



Project Remote Distributed Image (fractal) Processing



Summary

- Message passing programming model allows to easily scale up and/or scale out your application
- Supervision allows to build fault tolerant and resilient systems
- You can distribute the work and deploy actor cross network using remoting
- Routing provides different work distribution strategies to maximize/optimize the resources consumption
- Clustering allows the collaboration of different Actor-Systems to work as single unit, enabling a system to be responsive, resilient, elastic and message driven.

contacts

Source	https://github.com/rikace/raleighcc2019
Twitter	*@trikace*
Blog	*www.rickyterrell.com*
Email	*tericcardo@gmail.com*
Github	*www.github.com/rikace/*

The tools we use have a profound (and devious!) influence on our thinking habits, and, therefore, on our thinking abilities.

-- Edsger Dijkstra


That's all Folks!