Actor Programming model with Akka.NET (and F#)

BY RICCARDO TERRELL- @TRIKACE

Actor Programming model with Akka.NET (and F#)

BY RICCARDO TERRELL- @TRIKACE

Akka.NET

- Akka.Remote
- Akka.Cluster
- Akka.Persistence
- Akka.Streams
- Supervision
- Routing

* *Concurrent*
* *Resilient*
* *Distributed*
* *Scalable*

Akka.NET

- Akka.Remote
- Akka.Cluster
- Akka.Persistence
- Akka.Streams
- Supervision
- Routing

* Concurrent
* Resilient
* Distributed
* Scalable

MULTITHREADING

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

Concurrency Challenges

Shared State

Difficult to maintain and test

Very difficult to parallelize!

Locking is fundamentally error prone:

- Must guess where parallelism will be needed
- All consumers need to participate

I/O Parallelism

Scaling Up

The Solution is Immutability and Isolation

Concurrency Challenges

Shared State

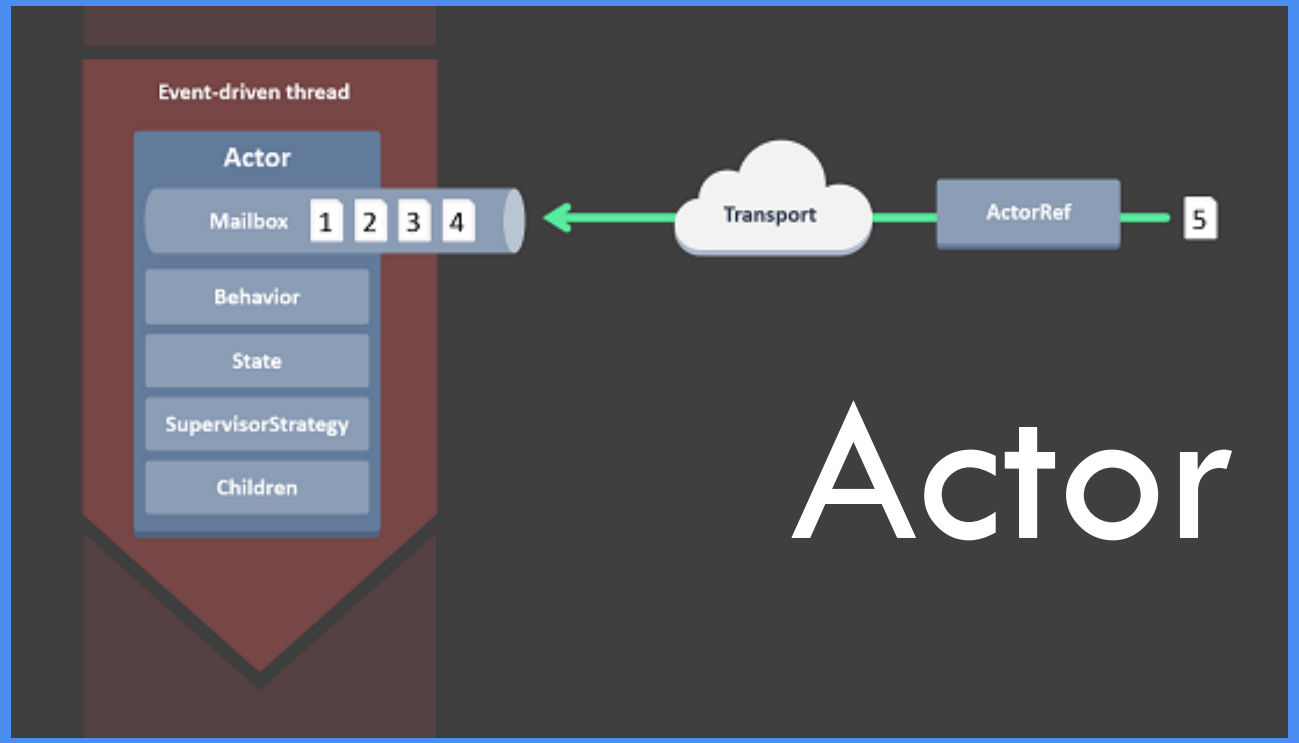
I/O Parallelism

Scaling Up

Programs must be written with this in mind
This means structuring with messaging and actors (agents)

Scaling Out

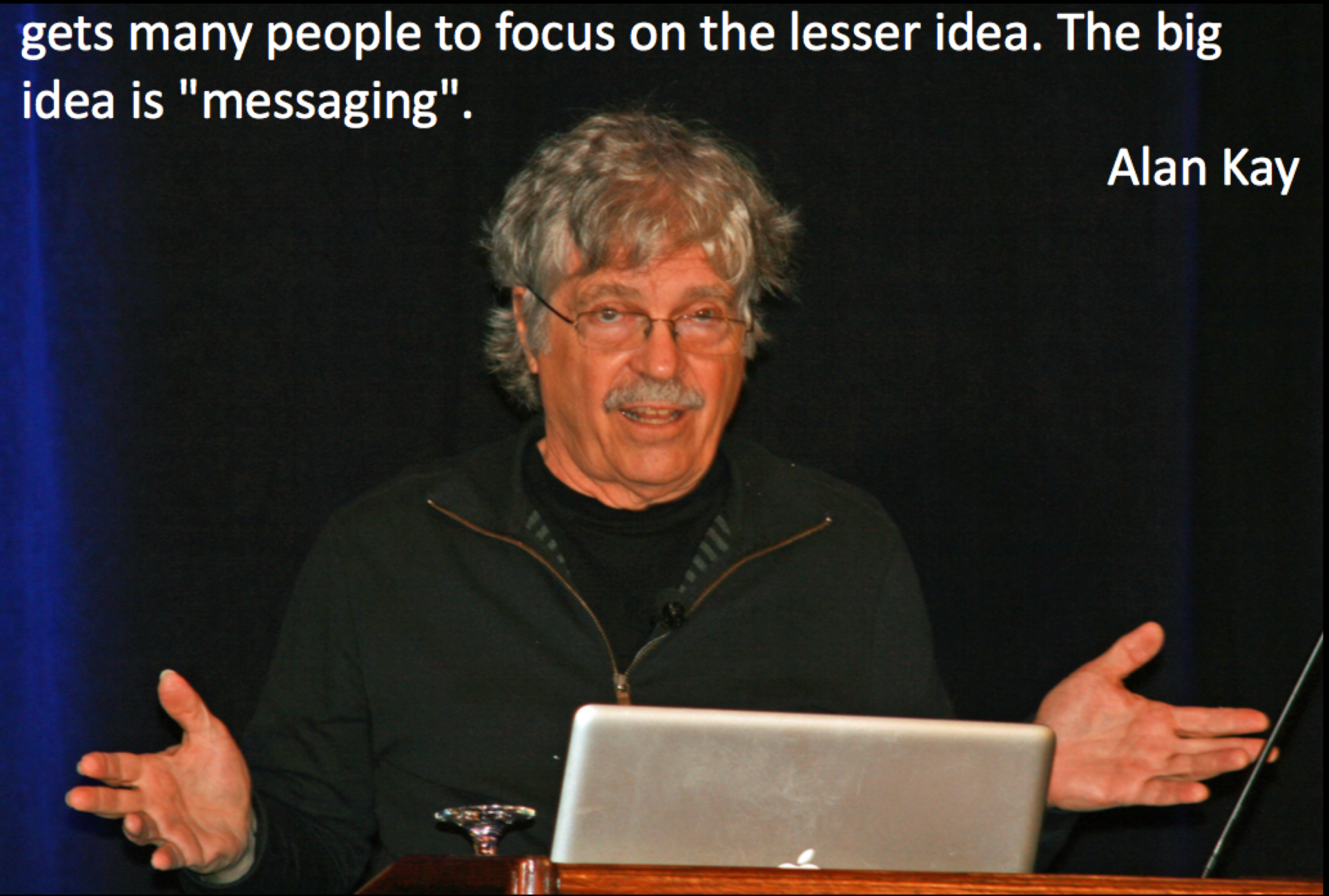
Roll-you-own clusters with cheap hardware
On-demand cloud compute with Azure


Event-driven thread
Actor
Mailbox
1
2
3
4
Behavior
State
SupervisorStrategy
Children
Transport
ActorRef
5

Actor Model

Actor is a single and independent unit of computation

Actor Model Message-Driven


I'm sorry that I coined the term "objects", because it gets many people to focus on the lesser idea. The big idea is "messaging".
Alan Kay

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.

Carl Hewitt's Actor Model

The fundamental unit of computations that embodies:

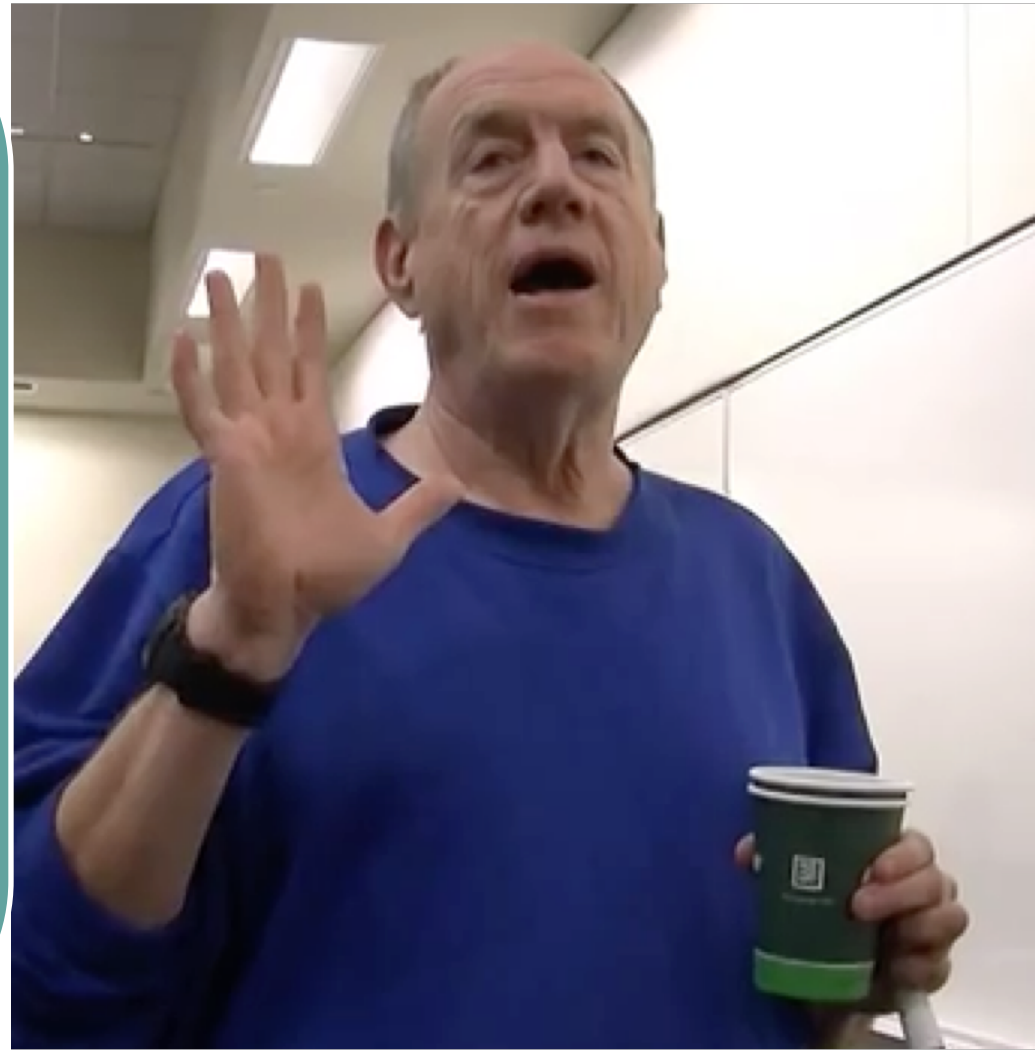
- Processing
- Storage
- Communication

Actor Model Three axioms:

1. Send messages to other Actors

   - *One Actor is not Actor -*

2. Create other Actor
3. Decide how to handle next message



Akka.NET

- Akka.Remote
- Akka.Cluster
- Akka.Persistence
- Akka.Streams
- Supervision
- Routing

* Concurrent
* Resilient
* Distributed
* Scalable

What is Akka.Net

Akka.NET is a port of the popular Java/Scala framework Akka to .NET.

"Akka is a toolkit and runtime for building highly concurrent, distributed, and resilient message-driven applications on the JVM."

- Typesafe

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
```

Actor – Akka.NET in better F#

```
let system = System.create "System" <| Configuration.load()

let greeter =
   // ActorFactory -> Name -> f(Actor<Message> -> Cont<'Message, 'return>) -> ActorRef
    spawn system "Greeter-Functional"
    <| fun mailbox ->
        let rec loop() = actor {
            let! msg = mailbox.Receive()
            match msg with
            | Greet(w) ->printfn "Hello %s" w
            return! loop() }
        loop()

greeter <! GreetMsg.Greet("AKKA.Net!!")
```

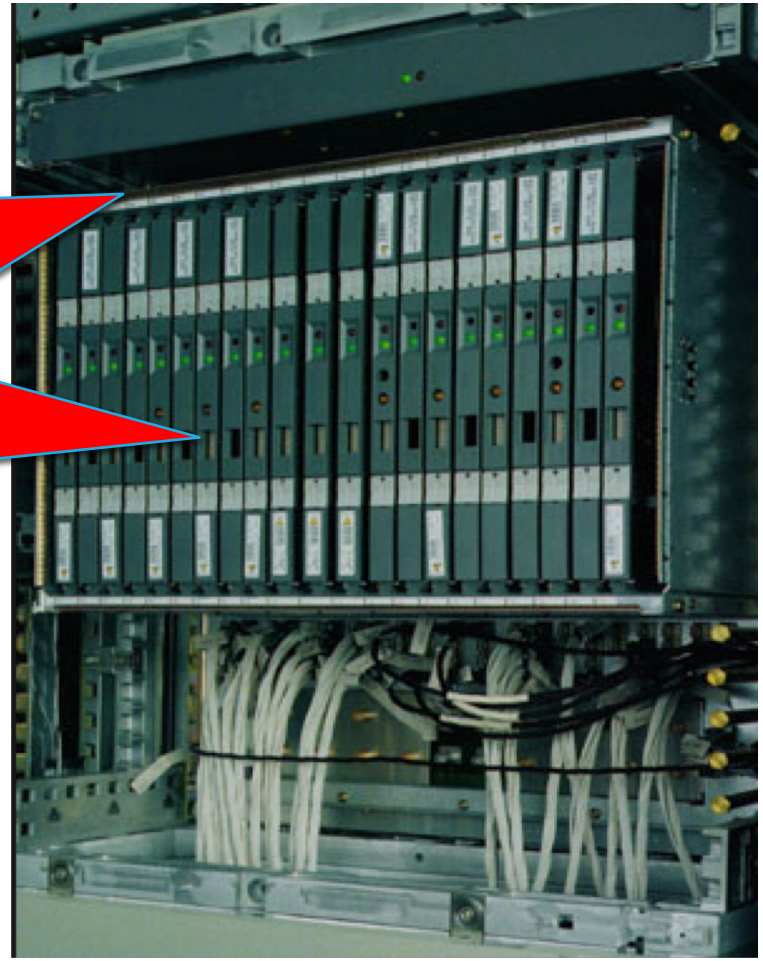
Multi-Threads vs Multi Actors

1 MB per thread (4 Mb in 64 bit) vs 2.7 million Actors per Gigabyte

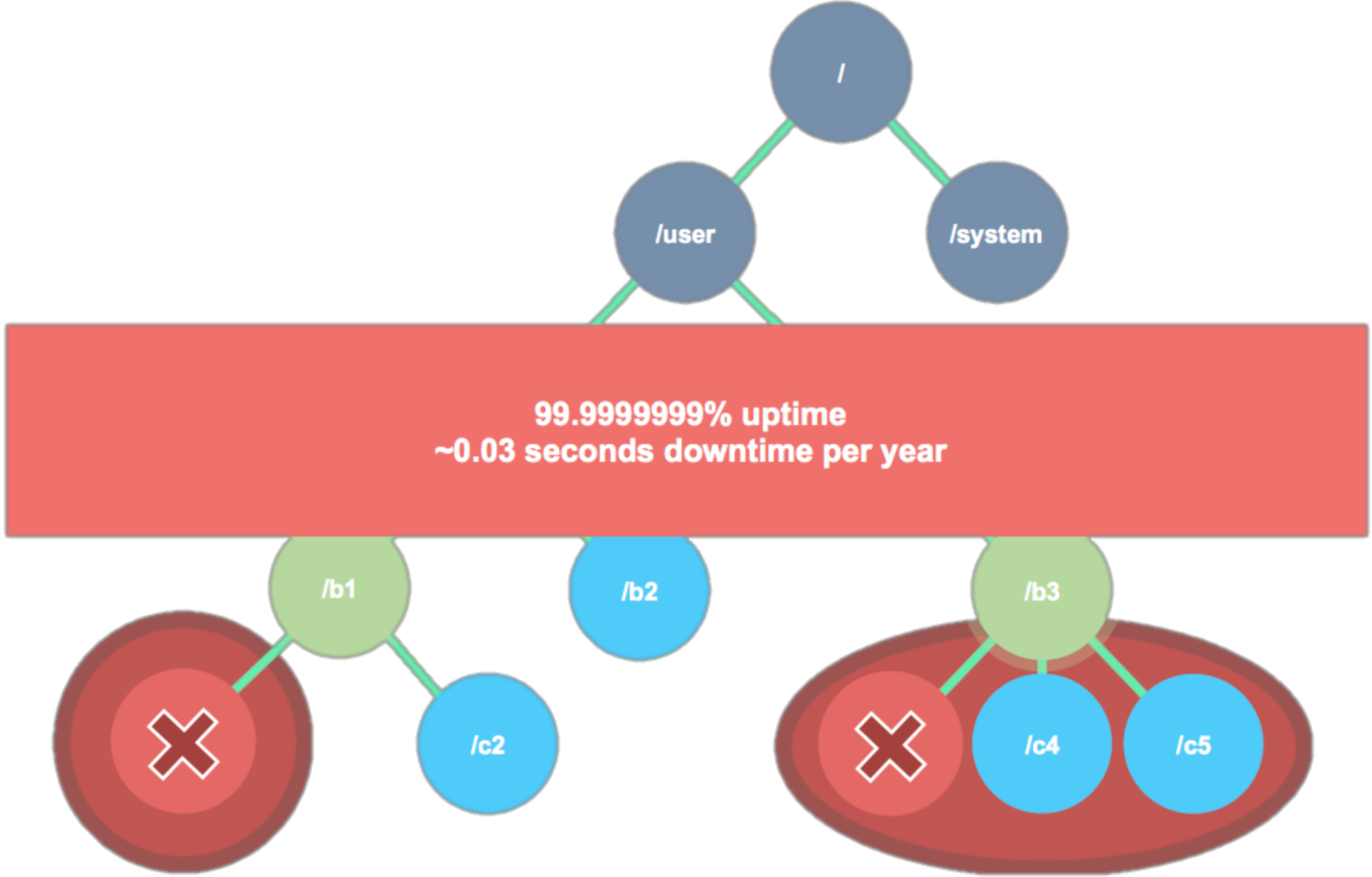
VS

Ericson AXD 301 Switch - 1996

99.9999999 percent uptime



Actor S


/
/user
/system
99.9999999% uptime
~0.03 seconds downtime per year
/b1
/b2
/b3
/c2
/c4
/c5

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

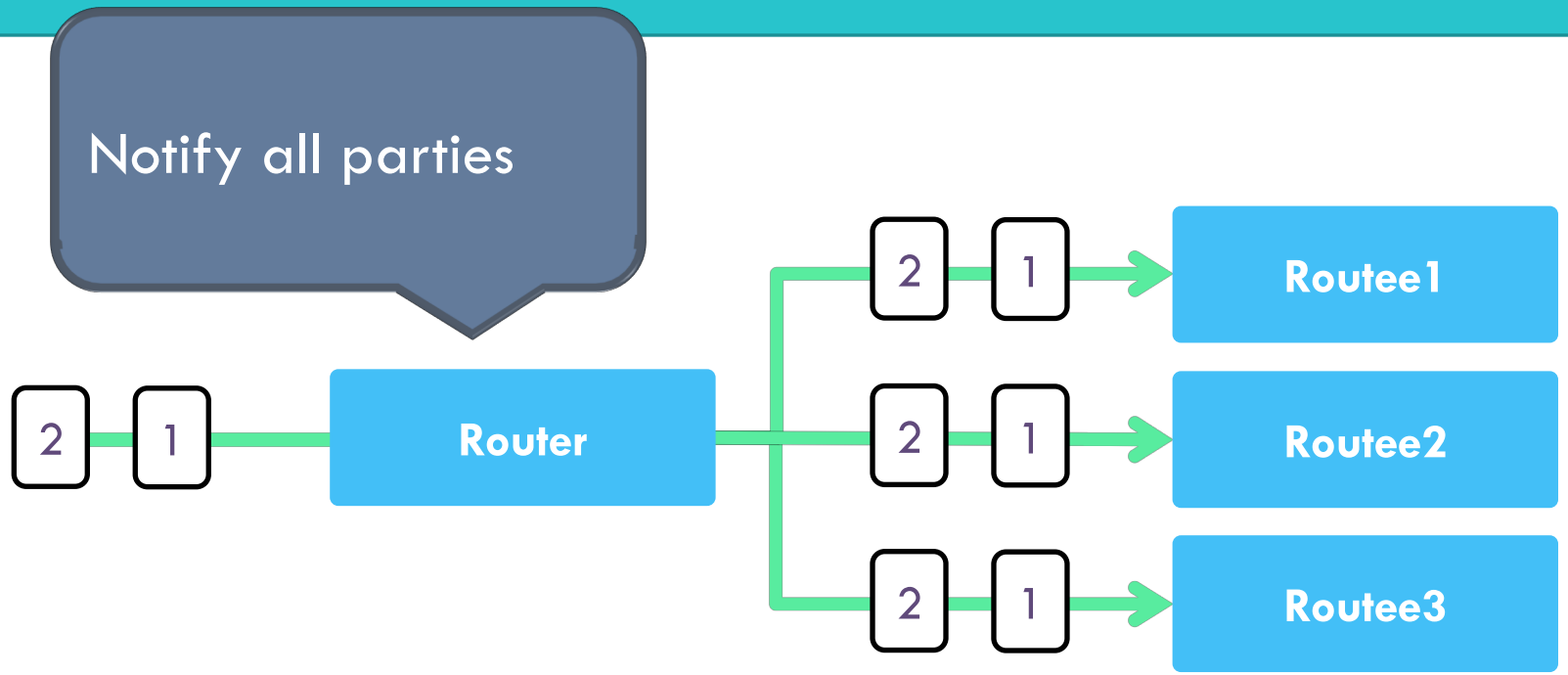
```
using (var system = ActorSystem.Create("Deployer", ConfigurationFactory.ParseString(@" // CLIENT
        akka {  ... CONFIG AS BEFORE ... }
            remote {
                helios.tcp {
                    port = 0
                    hostname = localhost
                }
            }
    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

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

Routing

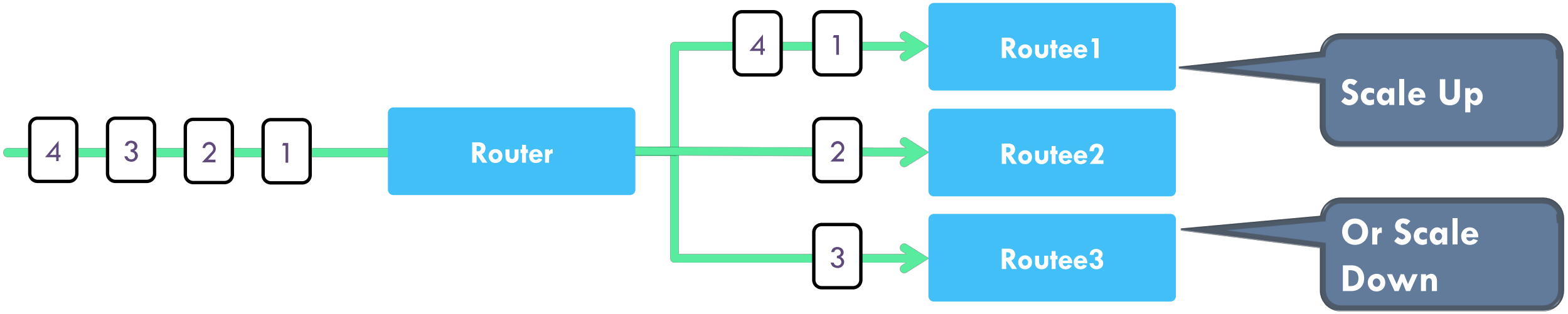
Akka.Net Routing Strategies

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



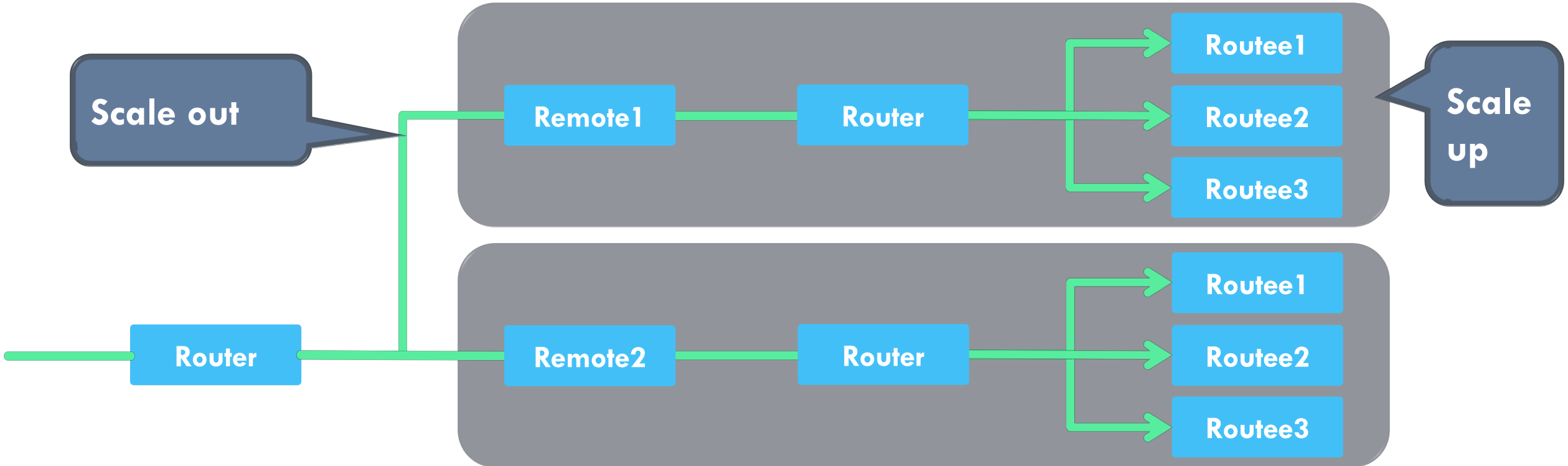
Akka.Net Routing Strategies

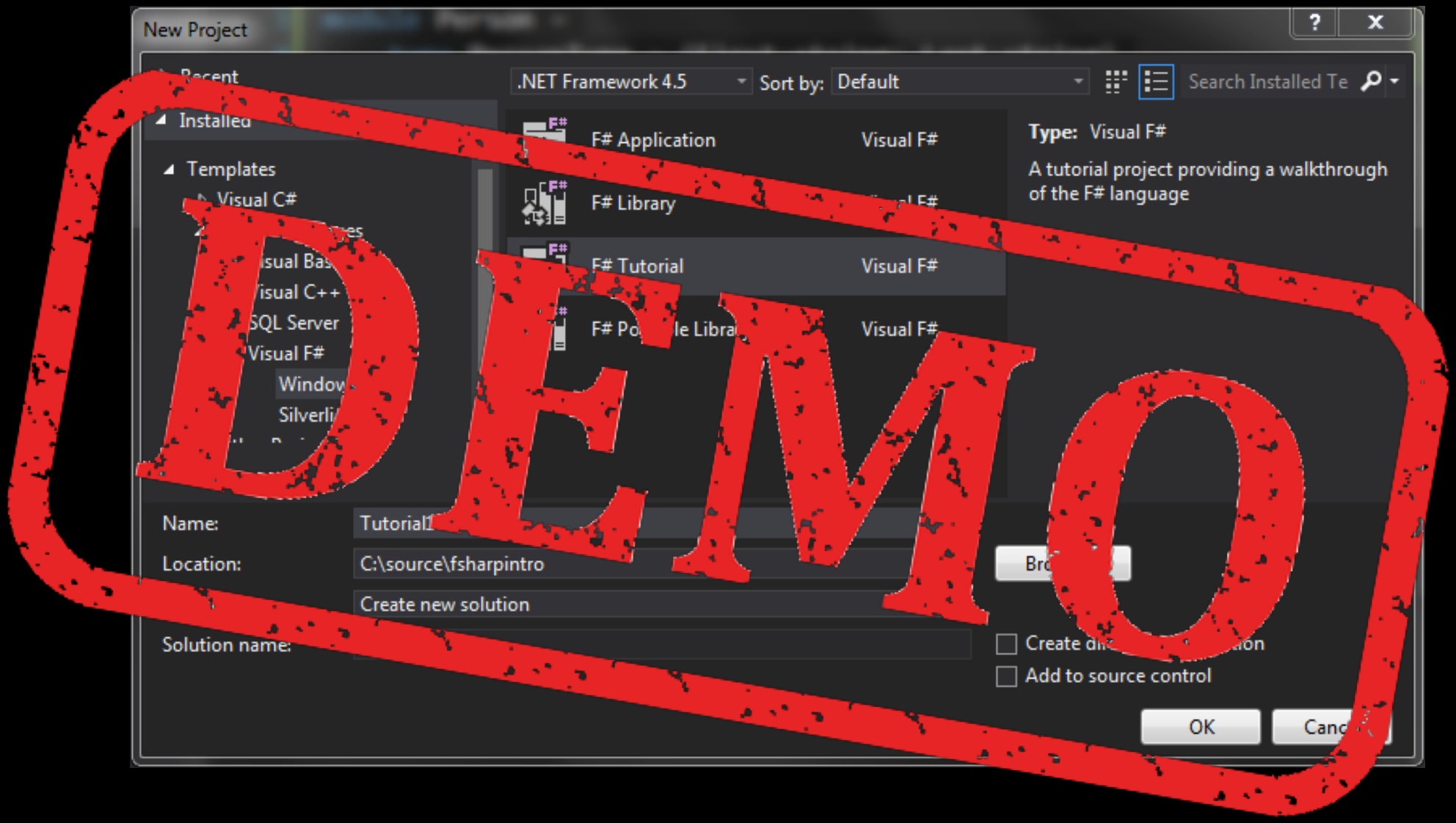
RoundRobin-Pool router uses round-robin to select a connection. For concurrent calls, round robin is just a best effort.



Akka.Net Routing Strategies

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

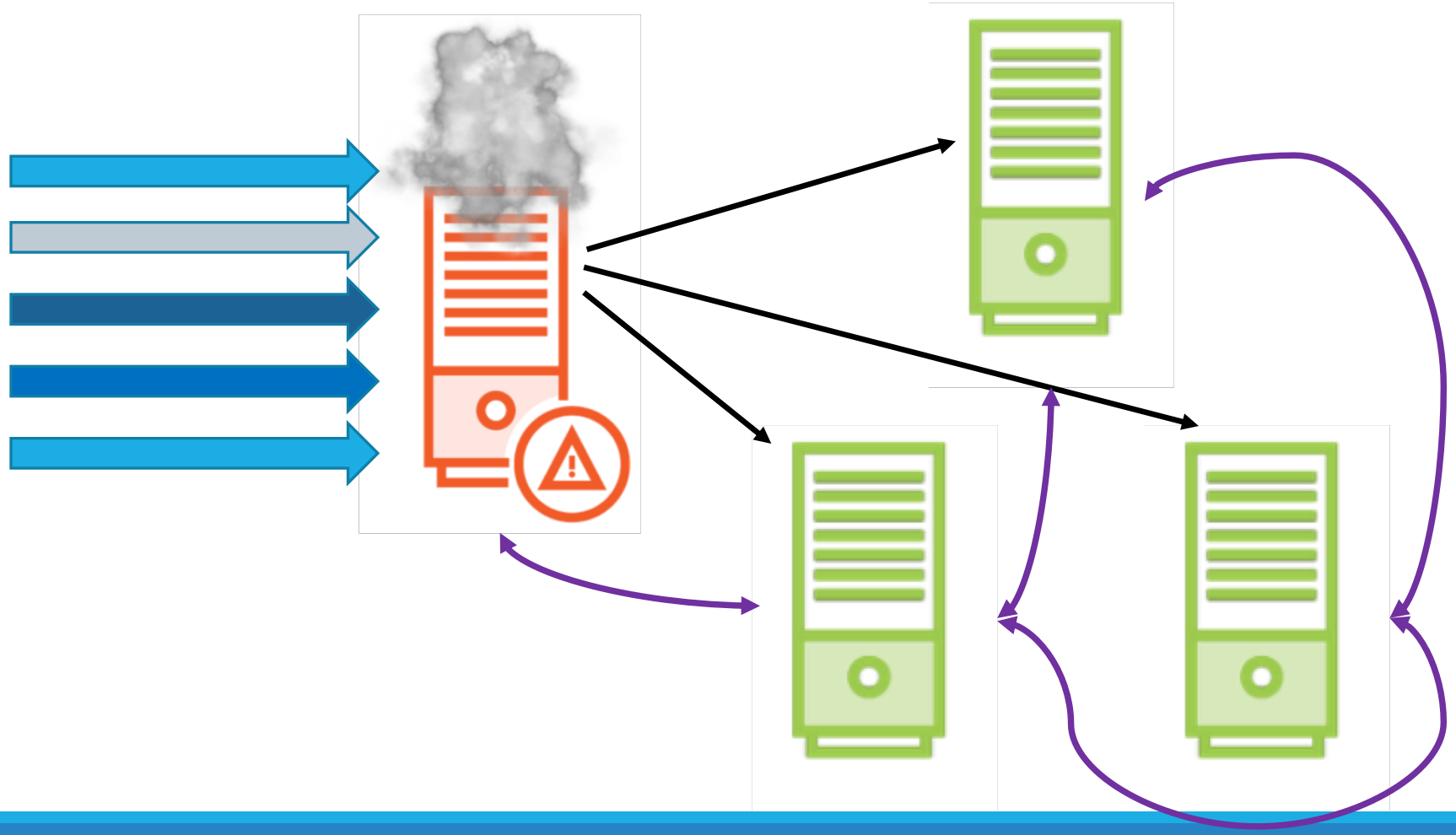



New Project
Installed
Templates
Visual C#
Visual F#
SQL Server
.NET Framework 4.5
Sort by:
Default
Search Installed Te
F# Application
Visual F#
F# Library
F# Tutorial
Visual F#
Type: Visual F#
A tutorial project providing a walkthrough of the F# language
Name:
Tutorial1
Location:
C:\source\fsharpintro
Create new solution
Solution name:
Add to source control
OK
DEMO

Clustering

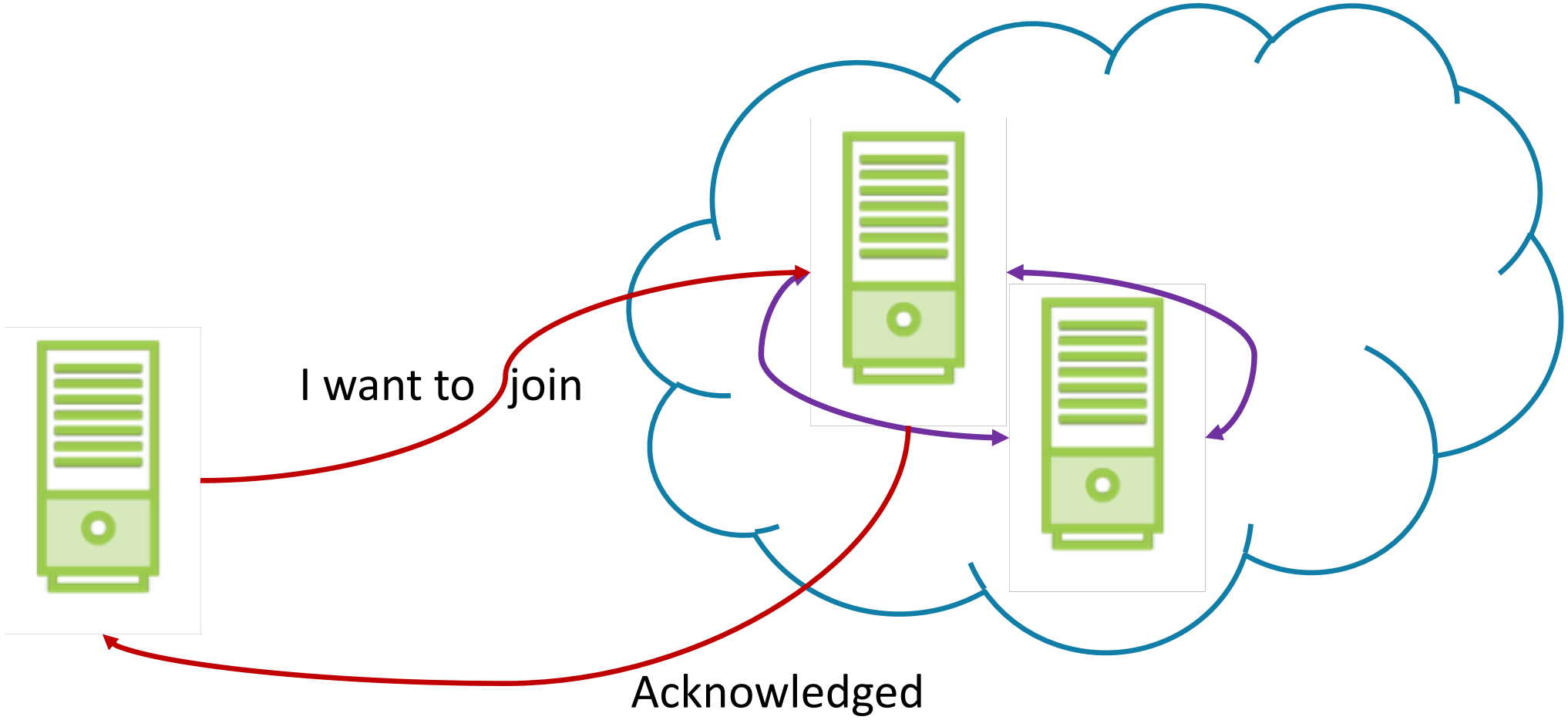
- Load balancing
- Fault-tolerant
- Scalability

Actor Clustering

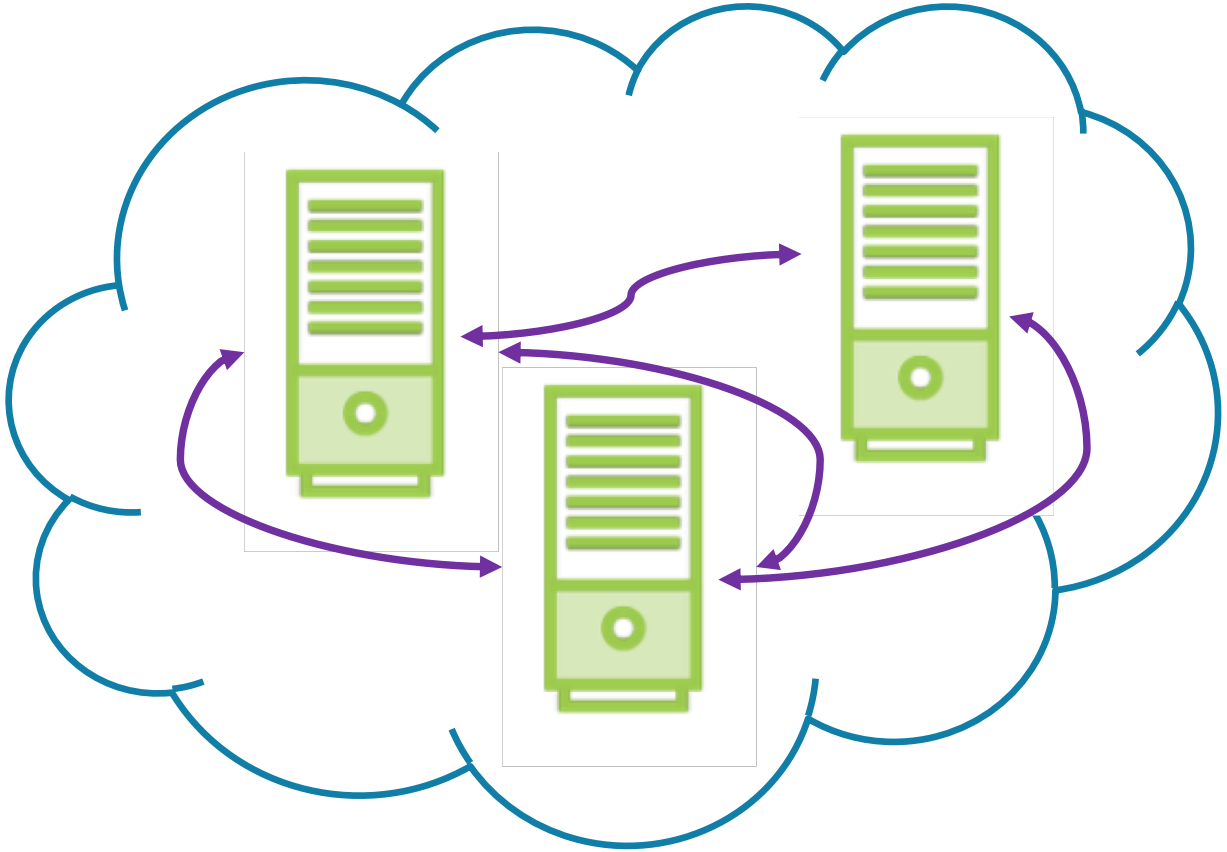


Gossip

Joining the Actor Cluster



Joining the Actor Cluster

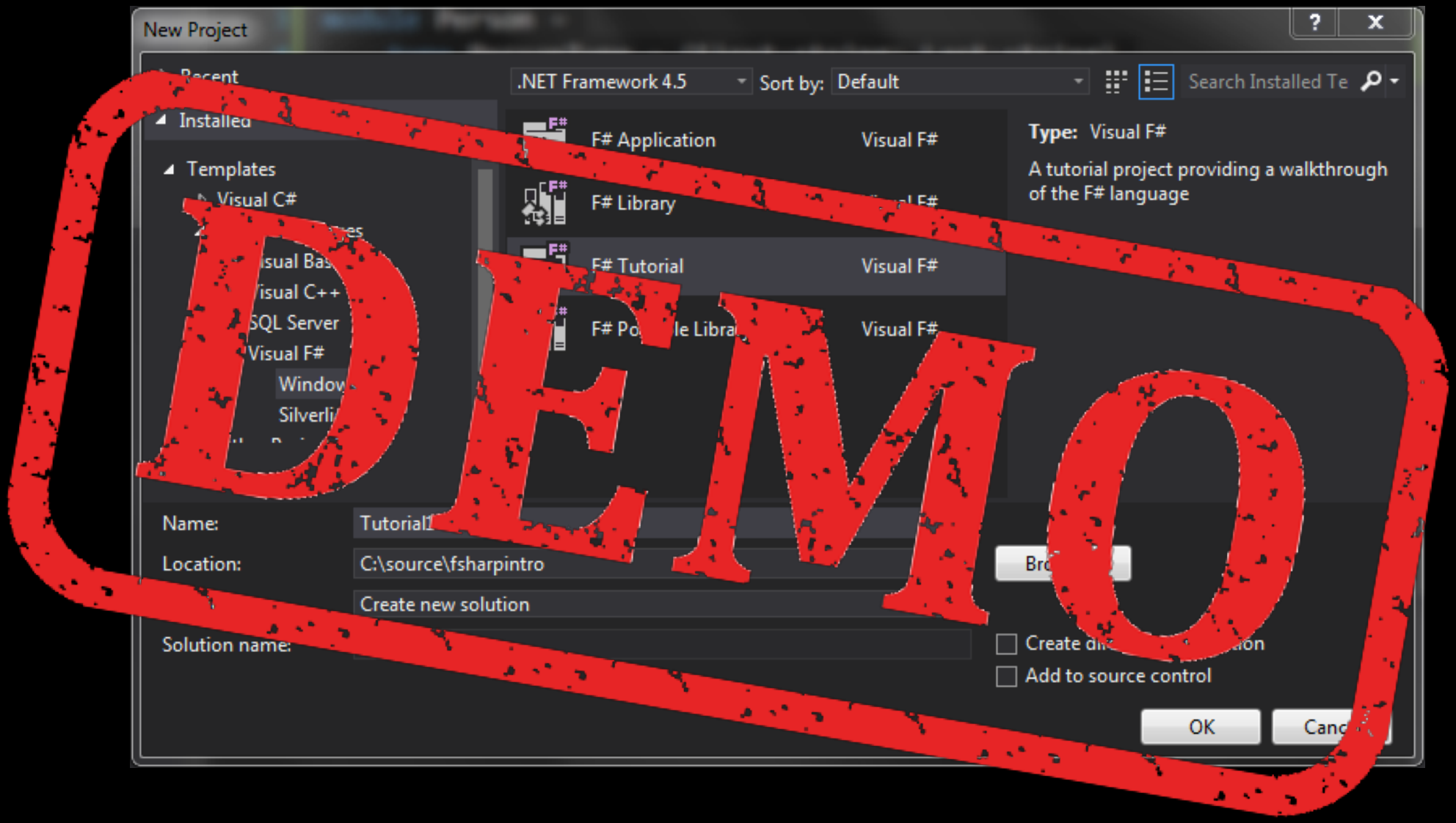


Clustering – Just a configuration matter

```
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/" ]
    }
}
```


New Project
Installed
Templates
Visual C#
Visual F#
SQL Server
.NET Framework 4.5
Sort by:
Default
Search Installed Te
F# Application
Visual F#
F# Library
F# Tutorial
Visual F#
Type: Visual F#
A tutorial project providing a walkthrough of the F# language
Name:
Tutorial1
Location:
C:\source\fsharpintro
Create new solution
Solution name:
Add to source control
OK
DEMO

<@ Code Quotations @>

<@ F# Code Quotations @>

```
// A typed code quotation.
let sum5 : Expr<int> = <@ 2 + 3 @>
```

```
val sum5 : Quotations.Expr<int> =
  Call (None, op_Addition, [Value (2), Value (3)])
```

<@ F# Code Quotations @>

```
let rec print expr =
    match expr with
    | Call(exprOpt, methodInfo, exprList) ->
        match exprOpt with
        | Some expr -> print expr
        | None -> printf "%s" methodInfo.DeclaringType.Name
        for expr in exprList.Tail do  print expr
    | Int32(n) ->printf "%d" n
    | Value(value, typ) -> printf ”%d” value
```

<@ F# Code Quotations @>

```
let fsum42 = fun x -> x + 42
```

<@ F# Code Quotations @>

```
let fsum42 = fun x -> x + 42
let serializer:BinarySerializer = FsPickler.CreateBinarySerializer()

let sum42:byte[] = serializer.Pickle<@  fsum42  @>
```

<@ F# Code Quotations @>

```
let fsum42 = fun x -> x + 42
let serializer:BinarySerializer = FsPickler.CreateBinarySerializer()

let sum42:byte[] = serializer.Pickle<@  fsum42  @>
let sum42 = serializer.UnPickle<Quotations.Expr<int -> int>> sum42
```

<@ F# Code Quotations @>

```
let fsum42 = fun x -> x + 42
let serializer:BinarySerializer = FsPickler.CreateBinarySerializer()

let sum42:byte[] = serializer.Pickle<@  fsum42  @>
let sum42 = serializer.UnPickle<Quotations.Expr<int -> int>> sum42

let sumfun = QuotationEvaluator.Evaluate sum42

printfn "%d" (sumfun  4) // print 46
```

<@ F# Code Quotations @>

```
let remoter =
        spawne system "remote"
                fun mailbox ->
                let rec loop(): Cont<int * string, unit> =
                    actor {
                        let! msg = mailbox.Receive()
                        match msg with
                        | (REQ, m) ->
                            printfn "Remote actor received: %A" m
                            mailbox.Sender() <! (RES, "ECHO " + m)
                        | _ -> logErrorf "Received unexpected message: %A" msg
                        return! loop()
                    }
                loop()
                [ SpawnOption.Deploy(remoteDeploy remoteSystemAddress) ]
```

<@ F# Code Quotations @>

```
let remoter =
      spawne system "remote"
            <@  fun mailbox ->
                let rec loop(): Cont<int * string, unit> =
                    actor {
                        let! msg = mailbox.Receive()
                        match msg with
                        | (REQ, m) ->
                            printfn "Remote actor received: %A" m
                            mailbox.Sender() <! (RES, "ECHO " + m)
                        | _ -> logErrorf "Received unexpected message: %A" msg
                        return! loop()
                    }
                loop()
             @> [ SpawnOption.Deploy(remoteDeploy remoteSystemAddress) ]
```

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

-- Edsger Dijkstra