2018 Global Azure BOOTCAMP VERONA

2018
Global Azure
BOOTCAMP

CloudGen
ominidi verso l'evoluzione

Gli sponsor Mondiali

2018 Global Azure BOOTCAMP

CloudGen

opsgility

JET BRAINS

SERVICEBUS 360

Microsoft

myget

cerebrata

CloudMonix

Gli sponsor del nostro evento

Platinum Sponsor

Gli sponsor del nostro evento

Gold Sponsor

Gli sponsor del nostro evento

2018 Global Azure BOOTCAMP

CloudGen ominidi verso l'evoluzione

digital joy

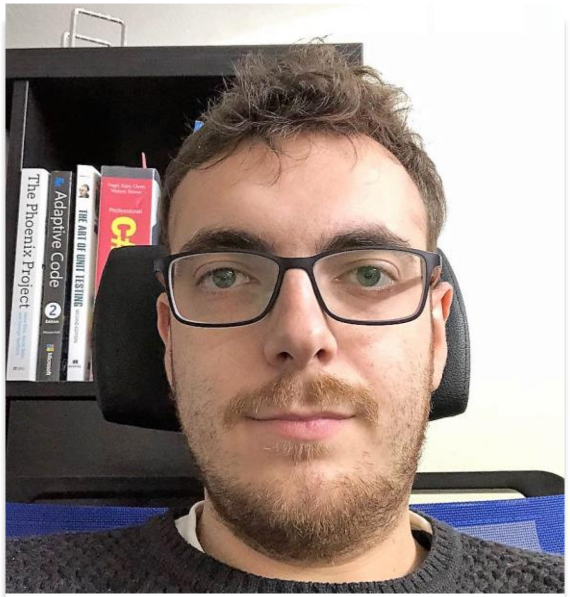
dgroove Tecnologia emozionale.

Basic Sponsor

Tweet della giornata

#GlobalAzure

@cloudgen_verona



MATTEO DEFANTI

@MatteoDefanti

Matteo Defanti

ARGOMENTO

Actor Model

Che cos’è l’Actor Model?

The **Actor Model** is a conceptual model to deal with concurrent computation.

It defines some general rules for how the system’s components should behave and interact with each other.

- Is inherently concurrent
- Manages concurrency through message passing
- Adopts the philosophy that everything is an actor

Che cos'è un actor?

One ant is no ant and one actor is no actor. They come in systems.

An actor is the primitive unit of computation. It's the *thing* that receives a message and do some kind of computation based on it.

- Should be lightweight
- Never shares state
- Communicates through asynchronous messages
- Has a mailbox to buffer messages
- Processes one message at a time
- Is a single-thread object

An Actor Can not exist on its own

Cosa può fare un actor?

The fundamental unit of computation that embodies:

- Processing
- Storage
- Communication

An actor can:

- Create new Actors
- Send messages to Actors
- Designate how to handle the next message

Che cosa sono i messaggi?

A message is the way that an Actor use to communicate.

In OOP, your objects communicate with each-other via function calls.
In the Actor model, actors communicate with each-other by sending messages.

Messages are stored in the Actor’s mailbox until they are processed.

Message passing is asynchronous *- the actor who sent the message can continue to do other work while the receiving actor processes the sender’s message.*

So in effect, every interaction one actor has with any other actor is going to be asynchronous by default.

Gli Actor inviano messaggi a degli Address, non direttamente agli Actor

Decoupling the sender from communications sent was a fundamental advance of the Actor model enabling asynchronous communication.

Recipients of messages are identified by address, sometimes called "mailing address".

It doesn’t matter if the actor that I’m sending a message to is running locally or in another node.

```
    //local actor
    var actorRef = MyActorSystem.Selection("/user/myActor");
    actorRef.Tell("HI!");

    //remote actor
    var remoteActorRef = MyActorSystem.Selection("akka.tcp://MyActorSystem@localhost:1001/user/myActor");
    remoteActorRef.Tell("HI!");
```

Actor Model VS OOP from concurrency point of view

Object Oriented Programming

- You need to care about shared memory
- Identifying and fixing all concurrency problems requires high level of knowledge
- Object communicate with each other via function call that usually are blocking

The Actor Model

- Higher abstraction level
- No shared memory between actors
- Simpler concurrent programming model
- Write single-threaded code (easier to understand)
- Maximize CPU utilization
- Easy to distribute

akka.net

Orleans

proto actor

Microsoft Azure
Service Fabric

ARGOMENTO

Akka.NET

It’s time for some Actors in action

akka.net

2018 Global Azure BOOTCAMP

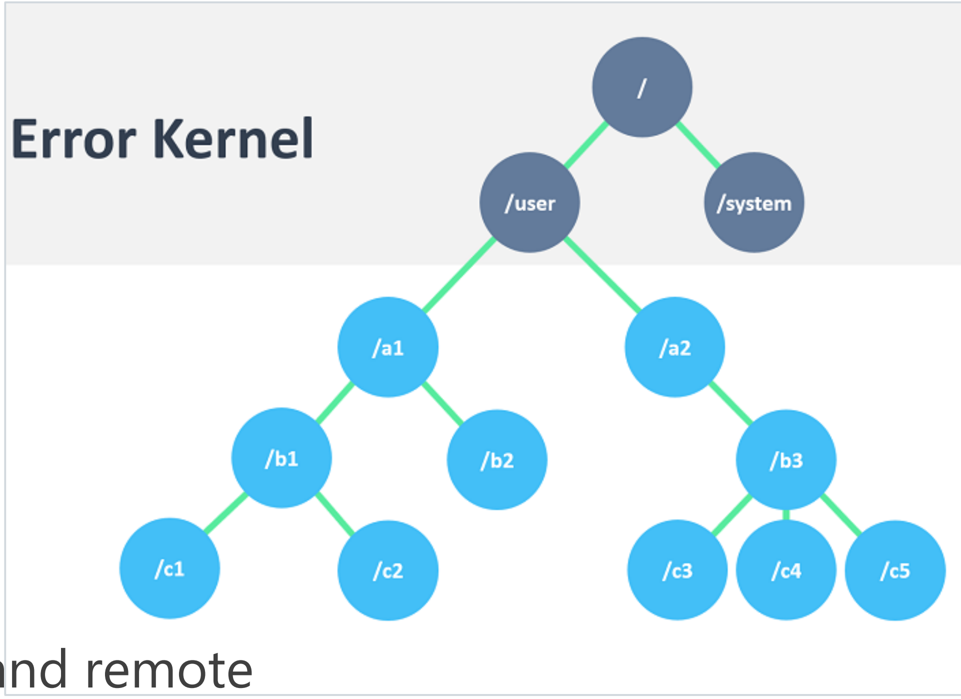
CloudGen ominidi verso l'evoluzione

Akka.Net Fundamentals

- Actors form an hierarchical structure

- Actor Lifecycle
  - Actors needs to be created and destroyed
  - Fault handling is done via supervision hierarchies and several available supervision strategies

- Location Transparency
  - Actors can be created remotely
  - Actors are called via an actor reference, same for local and remote



Install Akka.Net Actor Framework

To install Akka.NET Distributed Actor Framework, run the following command in the Package Manager Console

```
PM> Install-Package Akka

PM> Install-Package Akka.Remote
```

Cos’è ActorSystem?

```
public ActorSystem ActorService { get; private set; }
public ActorManagerService(string actorSystemName)
        {
                ActorService = ActorSystem.Create(actorSystemName);

        }
```

An actor system is a hierarchical group of actors which share common configuration, e.g. dispatchers, deployments, remote capabilities and addresses. It is also the entry point for creating or looking up actors. There are several possibilities for creating actors.

I messaggi devono essere immutable

Defining a message

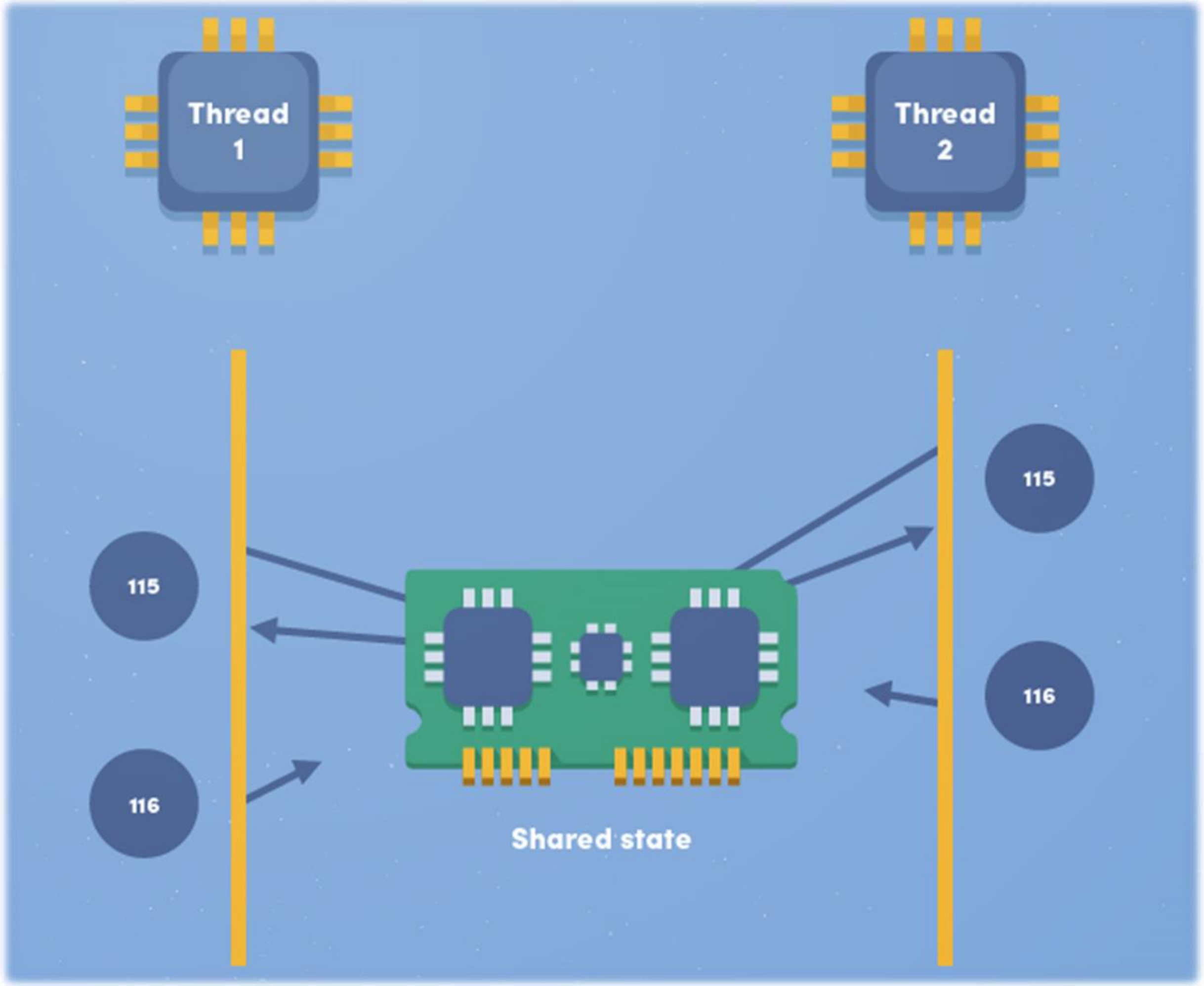
Actor state is always private, modified when handling messages, only shared via immutable messages.

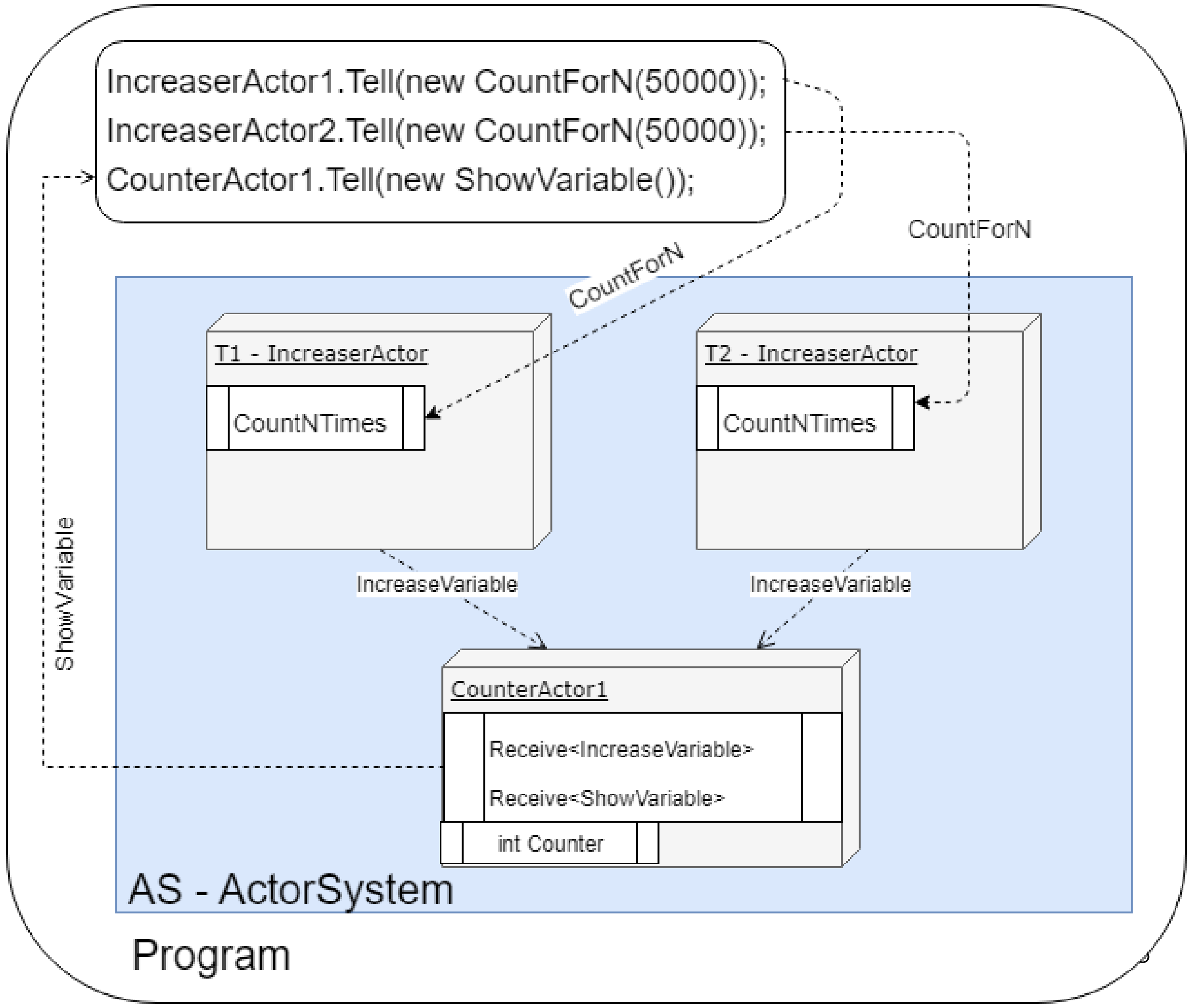
Immutable messages are inherently thread-safe. No thread can modify the content of an immutable message, so a second thread receiving the original message doesn’t have to worry about a previous thread altering the state in an unpredictable way.

```
public class InputSuccess
{
        public InputSuccess(string reason)
        {
                Reason = reason;
        }

        public string Reason { get; private set; }
}
```

Demo





```
IncreaserActor1.Tell(new CountForN(50000));
IncreaserActor2.Tell(new CountForN(50000));
CounterActor1.Tell(new ShowVariable());
```

ReceiveActor

Defining an Akka actor:

```
public class MyActor: ReceiveActor
    {
        private readonly ILoggingAdapter log = Context.GetLogger();

            public MyActor()
            {
                Receive<string>(message => {
                                log.Info("Received String message: {0}", message);
                                Sender.Tell(message);
                        });
                Receive<SomeMessage>(message => {...});
            }
    }
```

Istanziare un actor

```
Props props2 = Props.Create(() => new MyActor(param1, param2));

IActorRef myActor = MyActorSystem.ActorOf(props2, "myfirstactor");
```

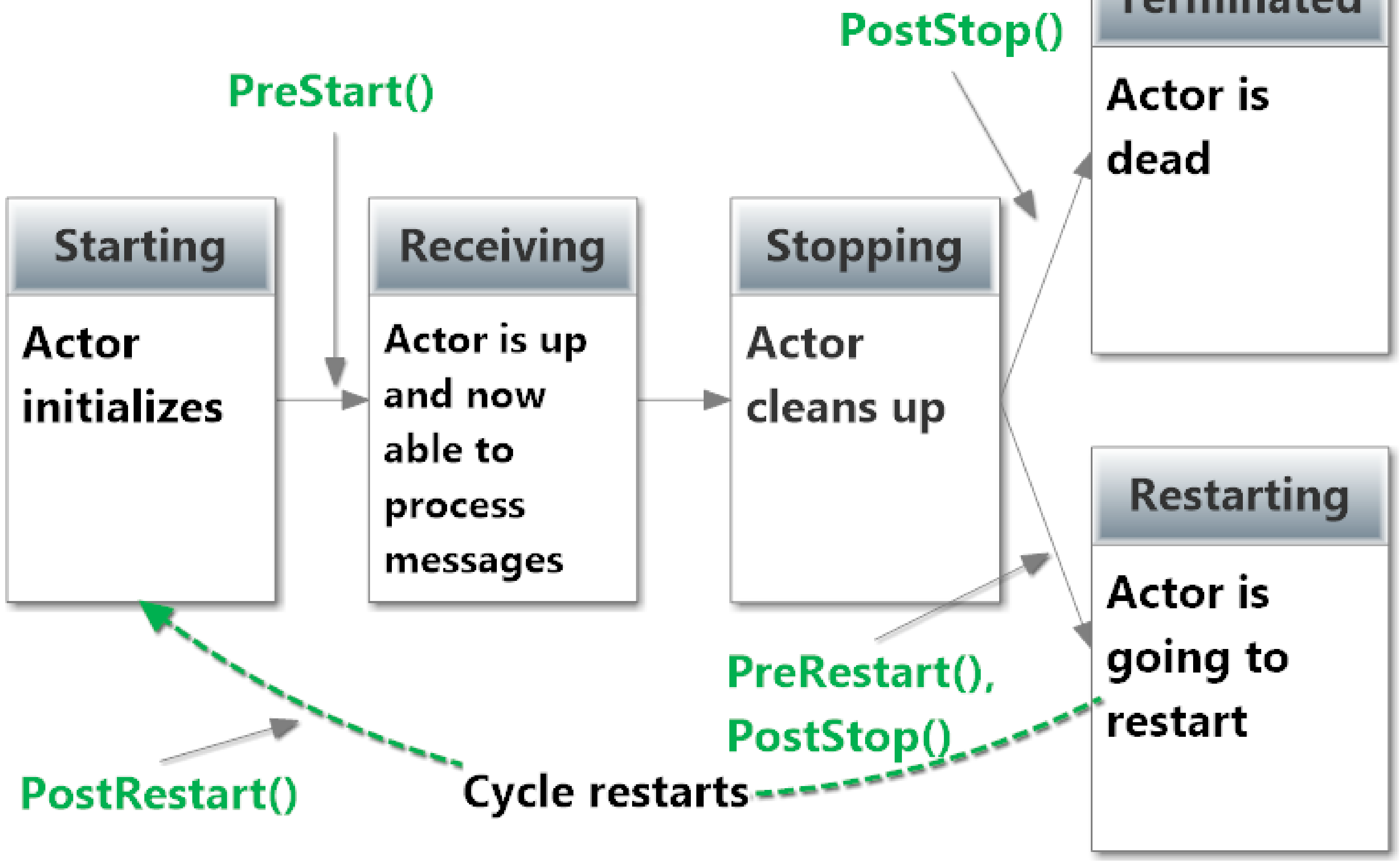
What are Props?

Think of Props as a recipe for making an actor. Technically, Props is a configuration class that encapsulates all the information needed to make an instance of a given type of actor.

What is an IActorRef?

An IActorRef is a reference or handle to an actor. The purpose of an IActorRef is to support sending messages to an actor through the ActorSystem. You never talk directly to an actor—you send messages to its IActorRef and the ActorSystemtakes care of delivering those messages for you.

Il ciclo di vita di un Actor



Fault Tollerance di un Actor

Let it crash!

Each actor is the supervisor of its children, and as such each actor defines fault handling supervisor strategy.

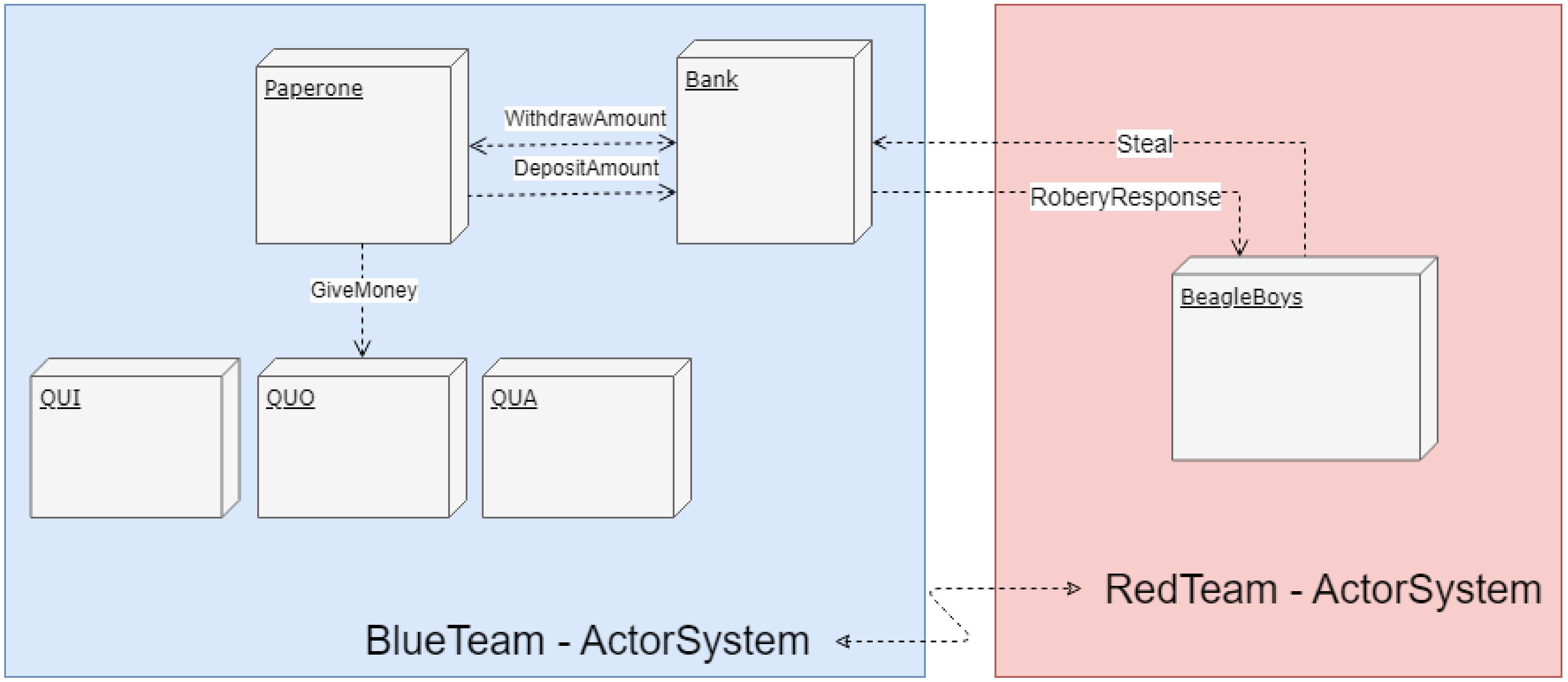
When does supervision come into play? Errors!

When things go wrong, that's when! Whenever a child actor has an unhandled exception and is crashing, it reaches out to its parent for help and to tell it what to do.
Specifically, the child will send its parent a message that is of the Failure class. Then it's up to the parent to decide what to do.

How can the parent resolve the error?

There are two factors that determine how a failure is resolved:
1. How the child failed (what type of Exception did the child include in its Failure message to its parent.)
2. What Directive the parent actor executes in response to a child Failure. This is determined by the parent's SupervisionStrategy.

Demo


Paperone
Bank
WithdrawAmount
DepositAmount
Steal
RoberyResponse
GiveMoney
QUI
QUO
QUA
BeagleBoys
RedTeam - ActorSystem
BlueTeam - ActorSystem

Resources

https://getakka.net/

https://petabridge.com/blog/

http://bartoszsypytkowski.com/dont-ask-tell-2/

https://www.brianstorti.com/the-actor-model/

https://www.slideshare.net/petabridge/akkanet-the-future-of-distributed-programming-in-net

Grazie

Domande?

@MatteoDefanti

Matteo Defanti

Coffee Break