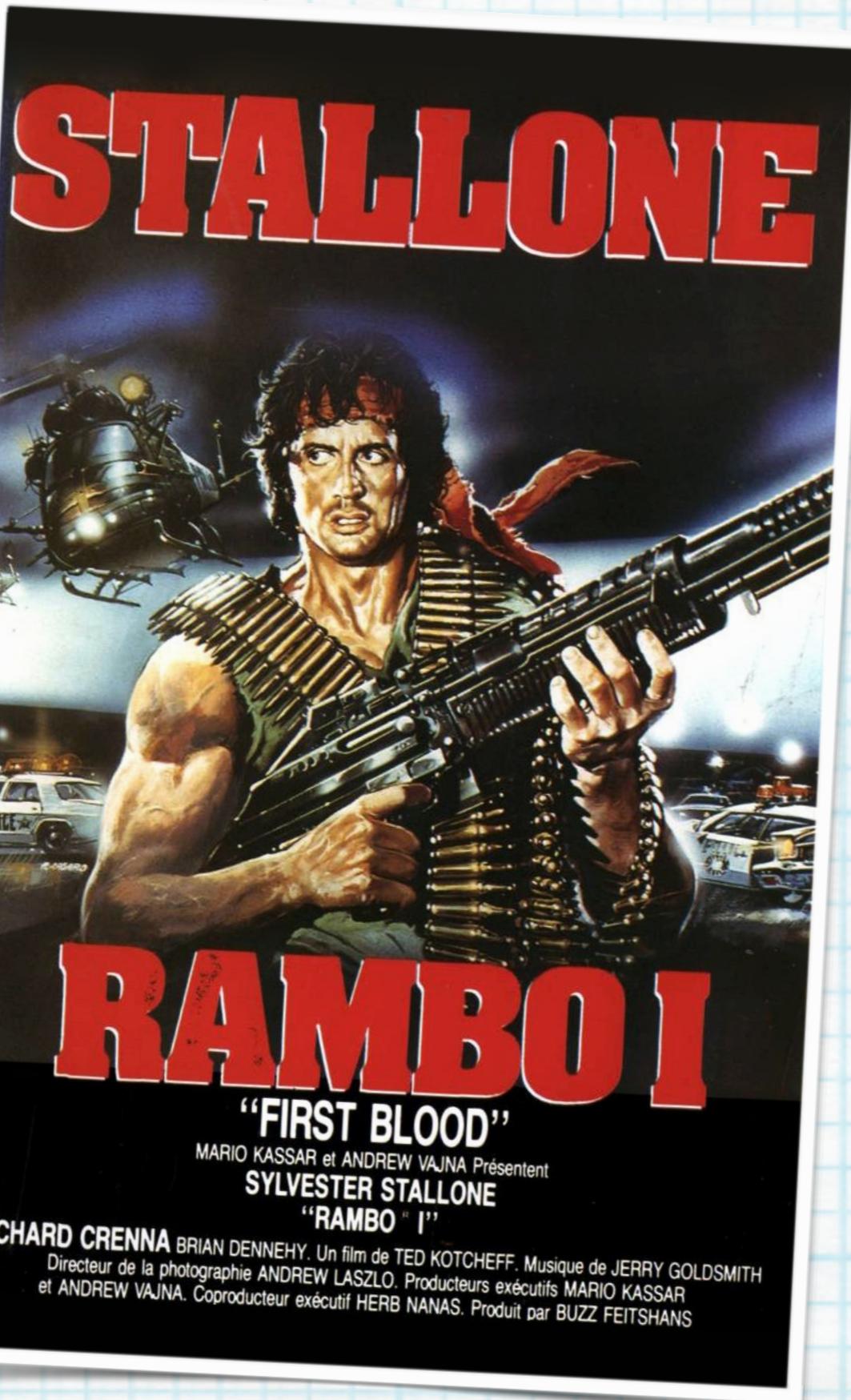


PL@NES - Reading Club **Actors à la Akka**

Christophe.VanGinneken@cs.kuleuven.be

THE FOLLOWING **PRESENTATION** IS NOT ABOUT A
SCIENTIFIC PAPER. IT INTRODUCES AN **INDUSTRY-GRADE**
TECHNOLOGY THAT IS REPRESENTATIVE FOR
THE LEVEL OF ADOPTION OF **THE ACTOR MODEL**.
OR MAYBE NOT...





What are Actors?



What are Actors?

Actor (disambiguation) – Wikipedia, the free encyclopedia

Create account Log in

Article Talk Read Edit View history Search

Actor (disambiguation)

From Wikipedia, the free encyclopedia

An **actor** is a person who plays a role in theater, cinema or television.

Actor can also refer to:

- Actants, also called **actors**, in actor-network theory (a general theory of sociological behaviour), the one who performs the act
- in **Interactions of Actor**, theory, excitations in any medium able to produce action, a theory of cybernetics
- in computing:
 - Actor (UML)**, requirements analysis and UML
 - Actor model**, in concurrency, refers to a model of concurrent computation
- Actor**, integrated development environment (IDE) for the Windows operating system
- Actor (law)**
- Actor (mythology)**, in Greek mythology, refers to a number of characters, including the father of Menoetius and Astyoche
- Actor (policy debate)**, the entity that enacts a certain policy action
- Actor (album)**, a 2009 album by St. Vincent

Look up **actor** in Wiktionary, the free dictionary.

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file

The Actor Model

The screenshot shows a web browser displaying the English Wikipedia page for "Actor model". The page title is "Actor model" and it is categorized under "Computer science". The page content discusses the Actor model as a mathematical model of concurrent computation. It highlights its origins in 1973, its use as primitives of concurrent computation, and its relationship to other models like Indeterminacy in concurrent computation and Actor model and process calculi. The page also covers its history, mentioning Carl Hewitt's work and subsequent milestones like Irene Greif's operational semantics and Gul Agha's dissertation. It details major software implementations at MIT and other institutions like Caltech and Stanford. The "Fundamental concepts" section explains the philosophy of the Actor model, noting its concurrency and decoupling of senders from receivers. The page includes a sidebar with navigation links and a footer with copyright information.

Actor model – Wikipedia, the free encyclopedia

https://en.wikipedia.org/wiki/Actor_model

Reader Create account Log in

Article Talk

Actor model

From Wikipedia, the free encyclopedia

This article may require cleanup to meet Wikipedia's quality standards. No cleanup reason has been specified. Please help improve this article if you can. (June 2010)

The actor model in computer science is a mathematical model of concurrent computation that treats "actors" as the universal primitives of concurrent computation: in response to a message that it receives, an actor can make local decisions, create more actors, send more messages, and determine how to respond to the next message received. The actor model originated in 1973.^[1] It has been used both as a framework for a theoretical understanding of computation and as the theoretical basis for several practical implementations of concurrent systems. The relationship of the model to other work is discussed in Indeterminacy in concurrent computation and Actor model and process calculi.

Contents [show]

History [edit]

Main article: History of the Actor model

According to Carl Hewitt, unlike previous models of computation, the Actor model was inspired by physics, including general relativity and quantum mechanics. It was also influenced by the programming languages Lisp, Simula and early versions of Smalltalk, as well as capability-based systems and packet switching. Its development was "motivated by the prospect of highly parallel computing machines consisting of dozens, hundreds or even thousands of independent microprocessors, each with its own local memory and communications processor, communicating via a high-performance communications network."^[2] Since that time, the advent of massive concurrency through multi-core computer architectures has revived interest in the Actor model.

Following Hewitt, Bishop, and Steiger's 1973 publication, Irene Greif developed an operational semantics for the Actor model as part of her doctoral research.^[3] Two years later, Henry Baker and Hewitt published a set of axiomatic laws for Actor systems.^{[4][5]} Other major milestones include William Clinger's 1981 dissertation introducing a denotational semantics based on power domains^[2] and Gul Agha's 1985 dissertation which further developed a transition-based semantic model complementary to Clinger's.^[6] This resulted in the full development of actor model theory.

Major software implementation work was done by Russ Atkinson, Giuseppe Attardi, Henry Baker, Gerry Barber, Peter Bishop, Peter de Jong, Ken Kahn, Henry Lieberman, Carl Manning, Tom Reinhardt, Richard Steiger and Dan Theriault in the Message Passing Semantics Group at Massachusetts Institute of Technology (MIT). Research groups led by Chuck Seitz at California Institute of Technology (Caltech) and Bill Dally at MIT constructed computer architectures that further developed the message passing in the model. See Actor model implementation.

Research on the Actor model has been carried out at California Institute of Technology, Kyoto University Tokoro Laboratory, MCC, MIT Artificial Intelligence Laboratory, SRI, Stanford University, University of Illinois at Urbana-Champaign,^[7] Pierre and Marie Curie University (University of Paris 6), University of Pisa, University of Tokyo Yonezawa Laboratory, Centrum Wiskunde & Informatica (CWI) and elsewhere.

Fundamental concepts [edit]

The Actor model adopts the philosophy that everything is an actor. This is similar to the everything is an object philosophy used by some object-oriented programming languages, but differs in that object-oriented software is typically executed sequentially, while the Actor model is inherently concurrent.

An actor is a computational entity that, in response to a message it receives, can concurrently:

- send a finite number of messages to other actors;
- create a finite number of new actors;
- designate the behavior to be used for the next message it receives.

There is no assumed sequence to the above actions and they could be carried out in parallel.

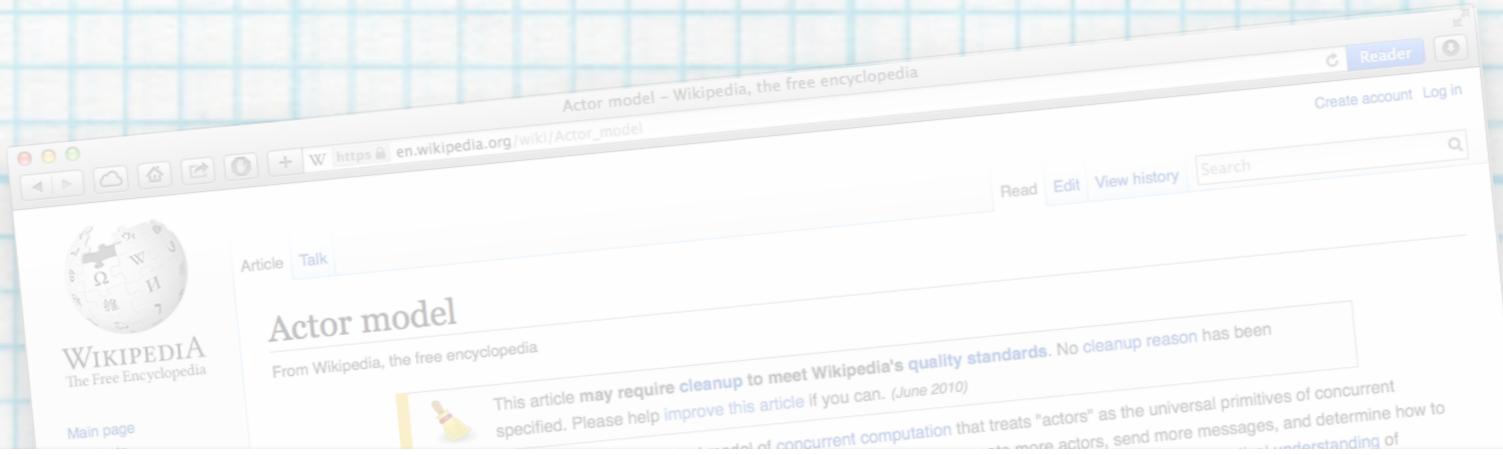
Decoupling the sender from communications sent was a fundamental advance of the Actor model enabling asynchronous communication and control structures as patterns of passing messages.^[8]

Recipients of messages are identified by address, sometimes called "mailing address". Thus an actor can only communicate with actors whose addresses it has. It can obtain those from a message it receives, or if the address is for an actor it has itself created.

The Actor model is characterized by inherent concurrency of computation within and among actors, dynamic creation of actors, inclusion of actor addresses in direct asynchronous message passing with no restriction on message arrival order.

source: https://en.wikipedia.org/wiki/Actor_model

The Actor Model

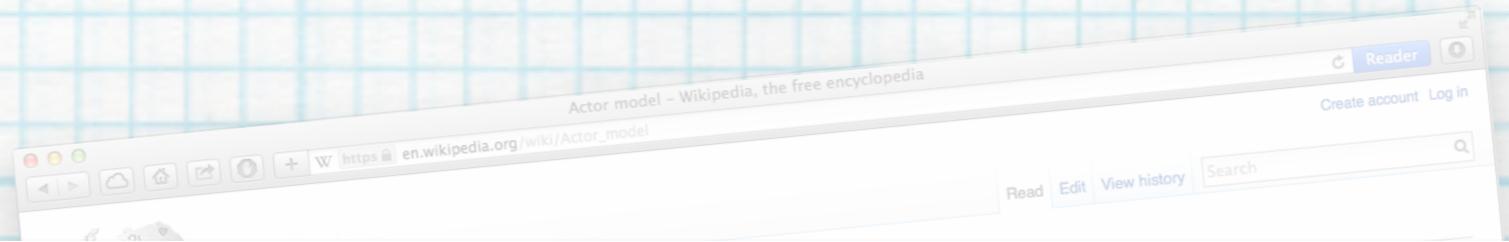


The **actor model** in computer science is a mathematical model of concurrent computation that treats "actors" as the universal primitives of concurrent computation: in response to a message that it receives, an actor can make local decisions, create more actors, send more messages, and determine how to respond to the next message received.

A detailed view of the Wikipedia 'Actor model' page. On the left, there is a sidebar with links for 'One time page', 'Print/export', 'Create a book', 'Download as PDF', 'Printable version', 'Languages' (with options for Català, Čeština, Deutsch, Français, 日本語, Română, Русский, 中文), and 'Edit links'. The main content area starts with a section titled 'Fundamental concepts' [edit]. It discusses the philosophy of the Actor model, noting that it is similar to the 'everything is an object' philosophy but is inherently concurrent. It lists three actions an actor can perform in response to a message: sending messages, creating new actors, and designating behavior for the next message. Below this, there is a section on 'Decoupling the sender from communications' [edit], which explains that actors are identified by address and can receive messages from different senders. The page continues with sections on 'Implementation' [edit] and 'See also' [edit].

source: https://en.wikipedia.org/wiki/Actor_model

The Actor Model

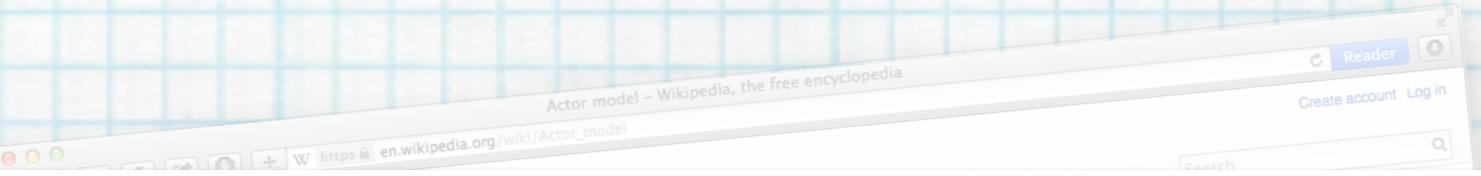


According to [Carl Hewitt](#), unlike previous models of computation, the Actor model was inspired by [physics](#), including general relativity and quantum mechanics. It was also influenced by the programming languages [Lisp](#), [Simula](#) and early versions of [Smalltalk](#), as well as [capability-based systems](#) and [packet switching](#). Its development was "motivated by the prospect of highly parallel computing machines consisting of dozens, hundreds or even thousands of independent microprocessors, each with its own local memory and communications processor, communicating via a high-performance communications network."^[2] Since that time, the advent of massive [concurrency](#) through [multi-core](#) computer architectures has [revived](#) interest in the Actor model.

A screenshot of the "Fundamental concepts" section of the Wikipedia page for the Actor model. The section starts with a note about the Actor model's philosophy: "The Actor model adopts the philosophy that *everything is an actor*. This is similar to the *everything is an object* philosophy used by some object-oriented programming languages, but differs in that object-oriented software is typically executed sequentially, while the Actor model is inherently concurrent." Below this, it describes an actor as a computational entity that can concurrently perform actions like sending messages, creating new actors, and designating behavior. A note states that there is no assumed sequence to these actions. Further down, it discusses decoupling senders from receivers and the pattern of passing messages. At the bottom, it mentions that actors are identified by address and can receive messages from other actors.

source: https://en.wikipedia.org/wiki/Actor_model

The Actor Model



Fundamental concepts

The Actor model adopts the philosophy that **everything is an actor**. This is similar to the **everything is an object** philosophy used by some **object-oriented programming languages**, but differs in that object-oriented software is typically **executed sequentially**, while the Actor model is inherently **concurrent**.

An actor is a computational entity that, in response to a message it receives, can concurrently:

- send a finite number of messages to other actors;
- create a finite number of new actors;
- designate the behavior to be used for the next message it receives.

There is no assumed sequence to the above actions and they could be carried out in **parallel**.

A screenshot of the 'Fundamental concepts' section of the Wikipedia Actor model article. The text describes the philosophy of everything being an actor, comparing it to object-oriented programming. It then lists three actions an actor can perform concurrently: sending messages, creating new actors, and designating behavior. Below this, a note states there is no assumed sequence and actions can be carried out in parallel. Further down, it discusses decoupling senders from communications and the identification of actors by address.

source: https://en.wikipedia.org/wiki/Actor_model

The Actor Model

Applications



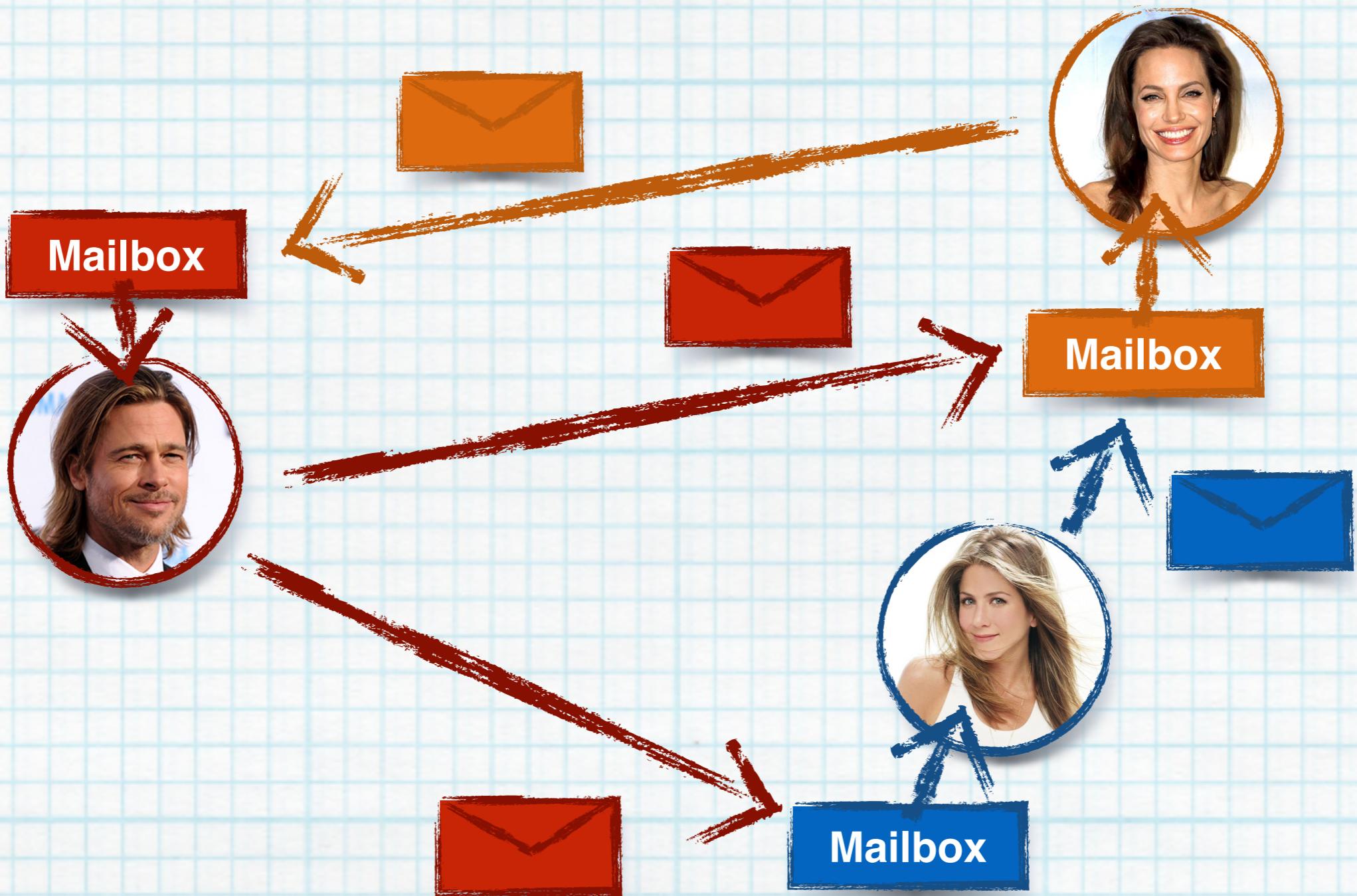
This article **needs additional citations for verification**. Please help [improve this article](#) by [adding citations to reliable sources](#). Unsourced material may be challenged and removed. *(December 2006)*

The Actors model can be used as a framework for modelling, understanding, and reasoning about, a wide range of [concurrent systems](#). For example:

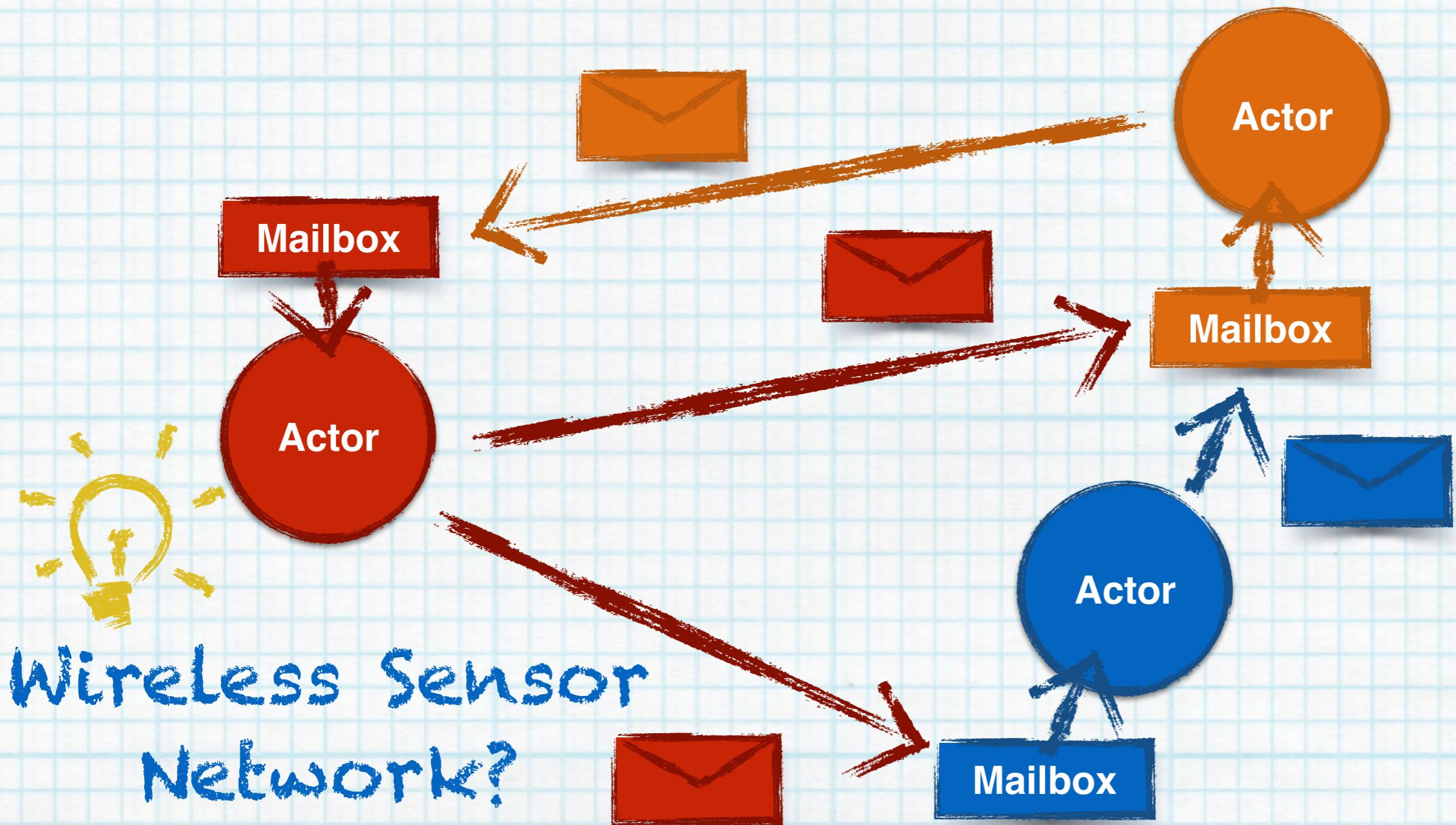
- [Electronic mail](#) (e-mail) can be modeled as an Actor system. Accounts are modeled as Actors and [email addresses](#) as Actor addresses.
- [Web Services](#) can be modeled with [SOAP](#) endpoints modeled as Actor addresses.
- [Objects with locks](#) (e.g., as in [Java](#) and [C#](#)) can be modeled as a **Serializer**, provided that their implementations are such that messages can continually arrive (perhaps by being stored in an internal queue). A serializer is an important kind of Actor defined by the property that it is continually available to the arrival of new messages; every message sent to a serializer is guaranteed to arrive.

send a finite number of messages
create a finite number of new actors;
designate the behavior to be used for the next message it receives.
There is no assumed sequence to the above actions and they could be carried out in parallel.
Decoupling the sender from communications sent was a fundamental advance of the Actor model enabling asynchronous communication and control structures
patterns of passing messages.^[8]
Recipients of messages are identified by address, sometimes called "mailing address". Thus an actor can only communicate with actors whose addresses it has.
It can obtain those from a message it receives, or if the address is for an actor it has itself created.
The Actor model is characterized by inherent concurrency of computation within and among actors, dynamic creation of actors, inclusion of actor addresses in direct asynchronous message passing with no restriction on message arrival order.

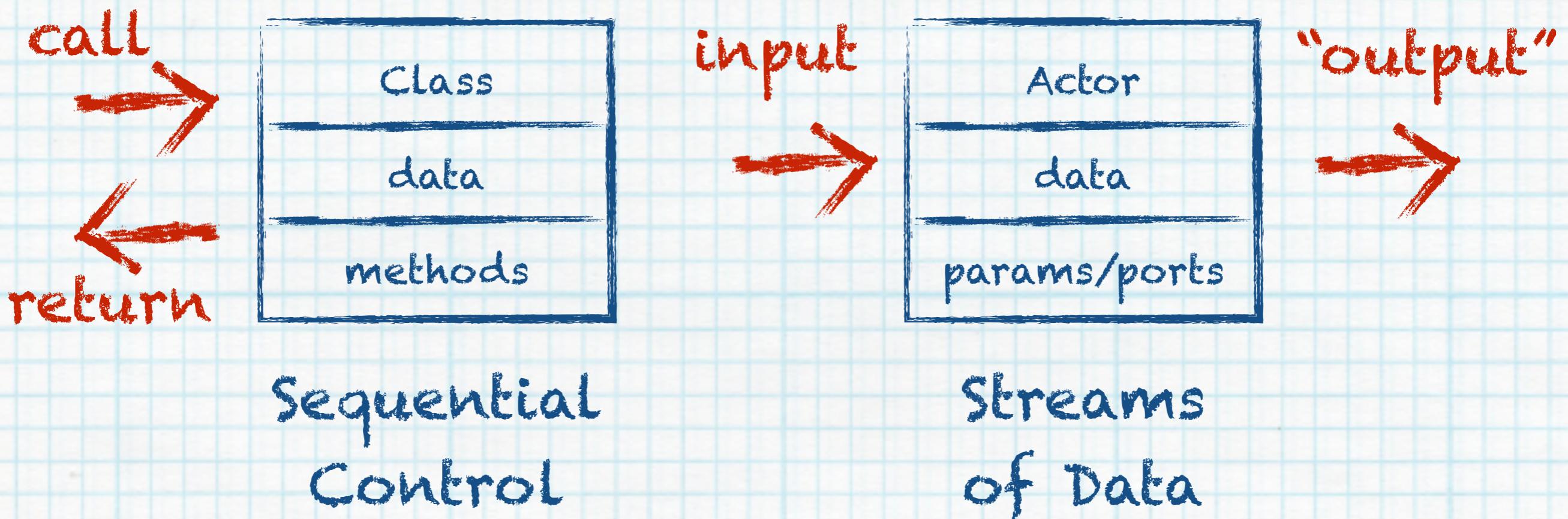
The Actor Model



The Actor Model

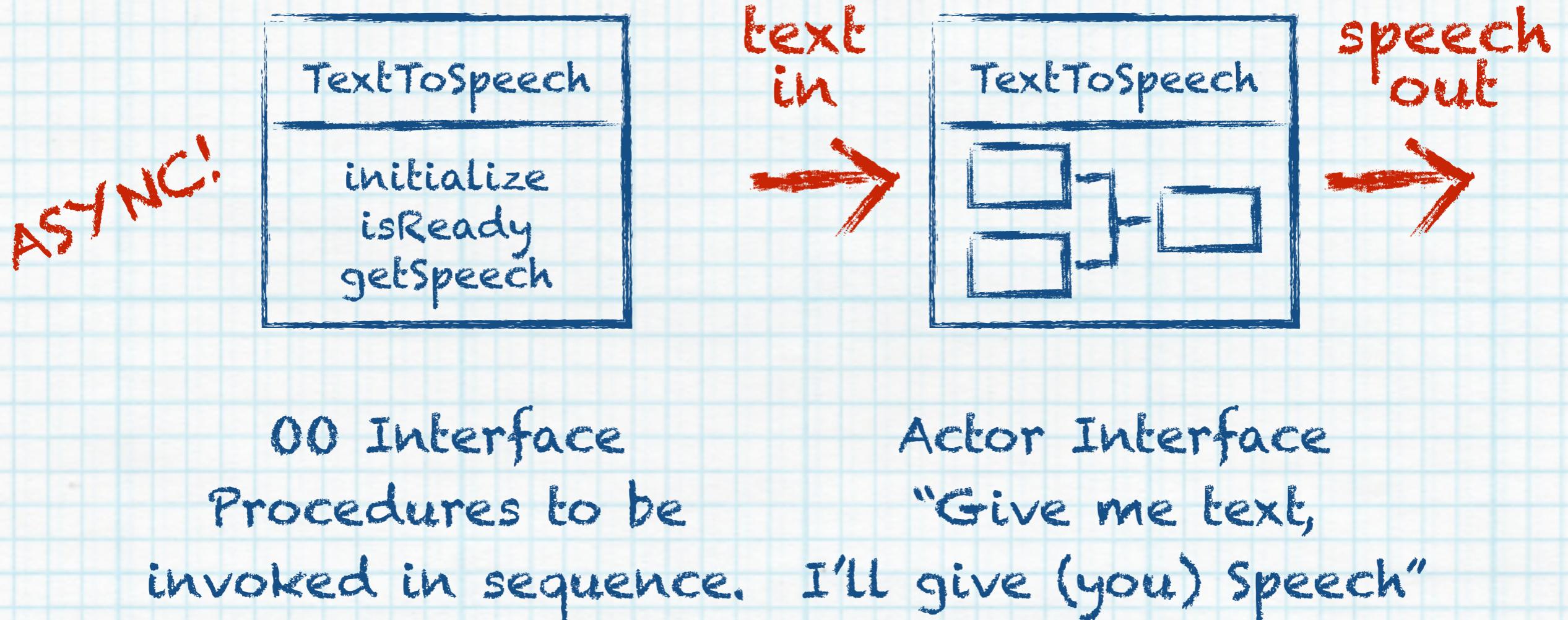


OOP vs Actor Model



source: Lee, Edward A. "Model-driven development-from object-oriented design to actor-oriented design." Workshop on Software Engineering for Embedded Systems: From Requirements to Implementation (aka The Monterey Workshop), Chicago. 2003.

OOP vs Actor Model



source: Lee, Edward A. "Model-driven development-from object-oriented design to actor-oriented design." Workshop on Software Engineering for Embedded Systems: From Requirements to Implementation (aka The Monterey Workshop), Chicago. 2003.

OOP vs Actor Model

input
→



"output"
→

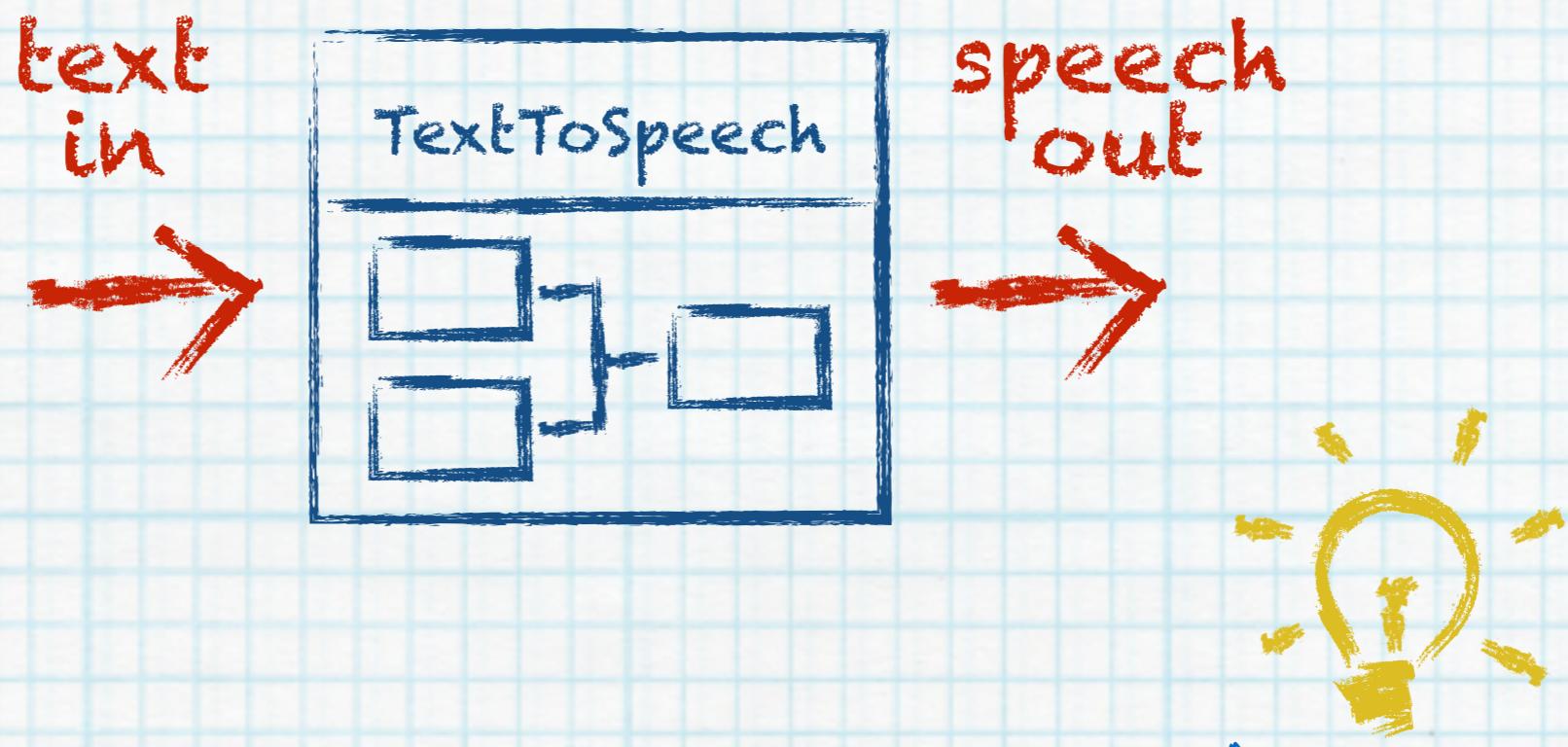
Streams
of Data

Reactive
Programming ?!



source: Lee, Edward A. "Model-driven development-from object-oriented design to actor-oriented design." Workshop on Software Engineering for Embedded Systems: From Requirements to Implementation (aka The Monterey Workshop), Chicago. 2003.

OOP vs Actor Model



Component Based
Programming ?!

source: Lee, Edward A. "Model-driven development-from object-oriented design to actor-oriented design." Workshop on Software Engineering for Embedded Systems: From Requirements to Implementation (aka The Monterey Workshop), Chicago. 2003.



Actor Model

**Reactive
Programming**

**Object Capability
Model**

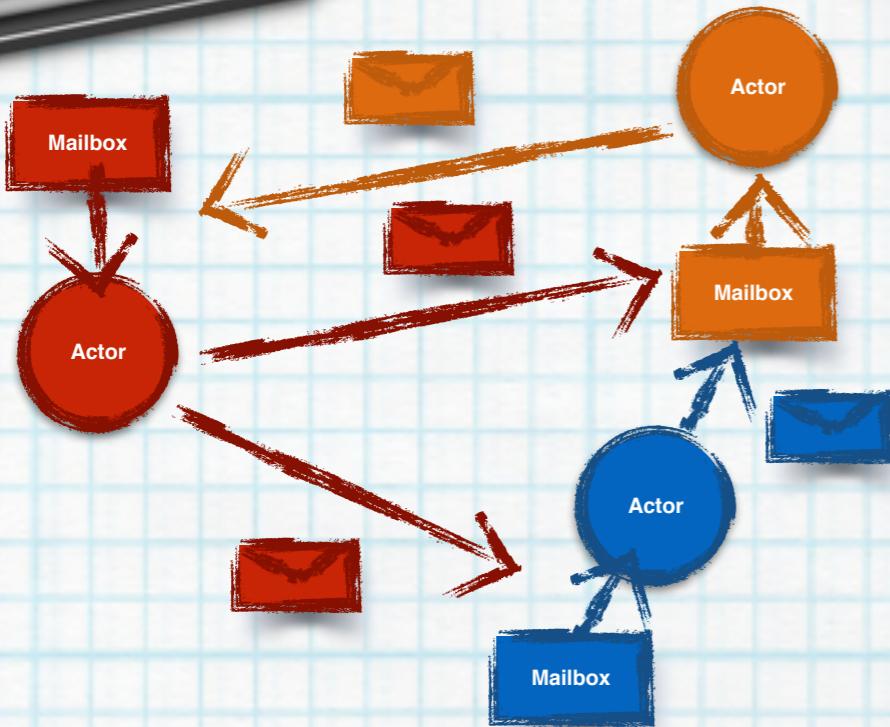
**Component Based
Programming**

**Object Oriented
Programming**

**Flow Based
Programming**

1973
1997

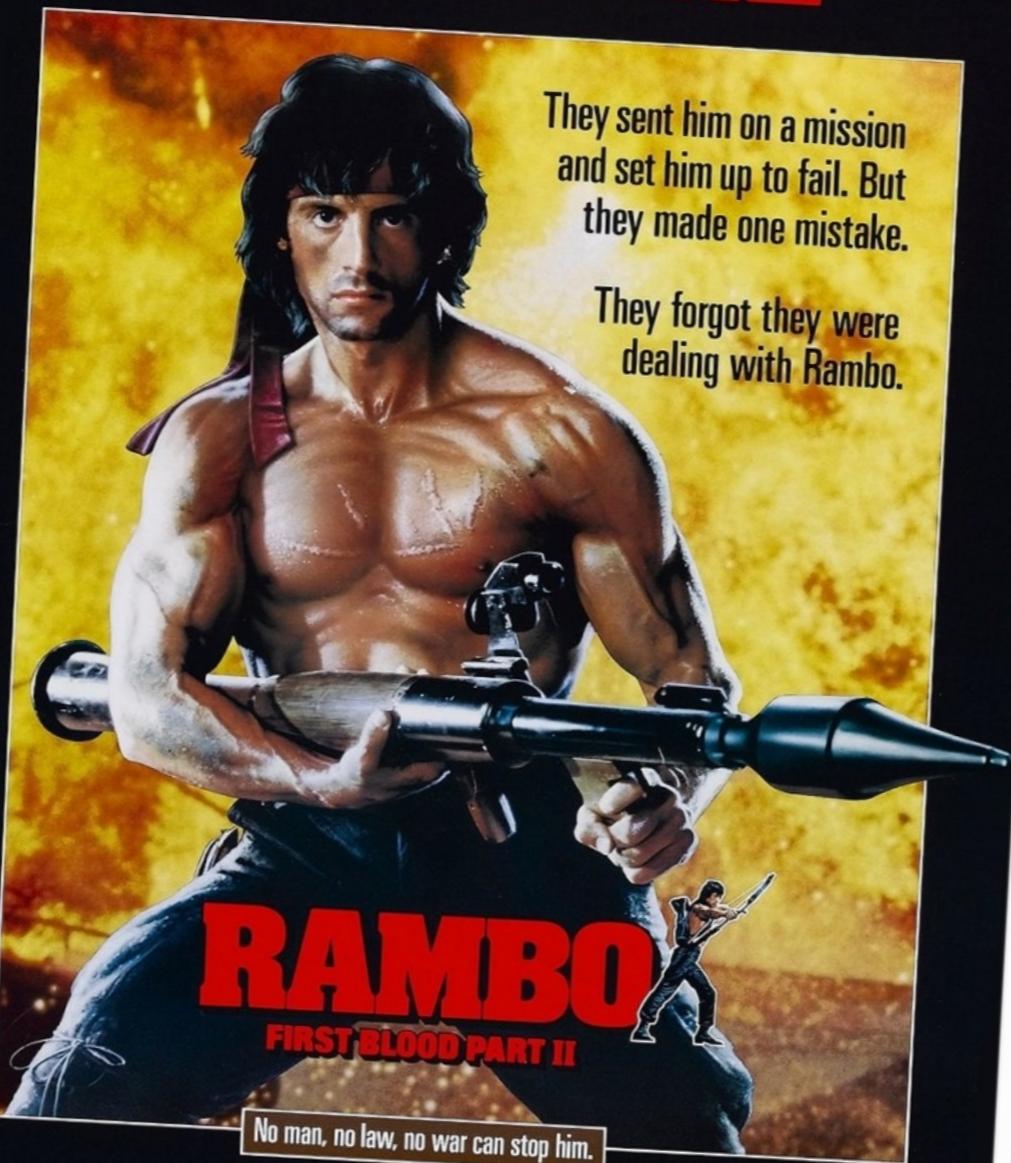
Success?



When the actor model was first proposed, the development of distributed networks was in its infancy. The conceptual model of actors is easy to understand as it allows state to be directly expressed. Also the only side effects of an actor are to send communications and to set a new behaviour. The simplicity of this model suggests that it would make programming for a distributed system simpler, but there proved to be difficulties associated with its implementation.

- No notion of inheritance/hierarchy
- Changing behaviour (storage,...)
- Dynamic behaviour versus static languages
- Asynchronous messaging versus algorithms

STALLONE





akka

What is Akka?

Scalable real-time transaction processing

We believe in writing correct distributed, fault-tolerant and scalable applications in Scala or Java. Most of the time it's been "we are using the wrong tools and the wrong abstraction level". Akka is here to change that. Using Akka, we can write more expressive applications—see the [Real-Time Manifesto](#) for more details. In fact, we adopt the "let it crash" model which telecom industry has used with success to build systems that self-heal and systems that never stop. Actors also provide the abstraction for transparent distributed and the basis for truly scalable and fault-tolerant applications.

Akka is Open Source and available under the Apache 2 License.

I ❤ MARKETING



hello.java — Desktop

```
1 package sample.hello;
2
3 import akka.actor.Props;
4 import akka.actor.UntypedActor;
5 import akka.actor.ActorRef;
6
7 public class HelloWorld extends UntypedActor {
8
9     @Override
10    public void preStart() {
11        // create the greeter actor
12        final ActorRef greeter = getContext().actorOf(Props.create(Greeter.class), "greeter");
13        // tell it to perform the greeting
14        greeter.tell(Greeter.Msg.GREET, getSelf());
15    }
16
17    @Override
18    public void onReceive(Object msg) {
19        if (msg == Greeter.Msg.DONE) {
20            // when the greeter is done, stop this actor and with it the app
21            getContext().stop(getSelf());
22        } else
23            unhandled(msg);
24    }
25}
26
```

Line: 10:27 | Java | Soft Tabs: 2 | preStart()

main.java — Desktop

```
1 package sample.hello;
2
3 public class Main {
4
5     public static void main(String[] args) {
6         akka.Main.main(new String[] { HelloWorld.class.getName() });
7     }
8 }
9
```

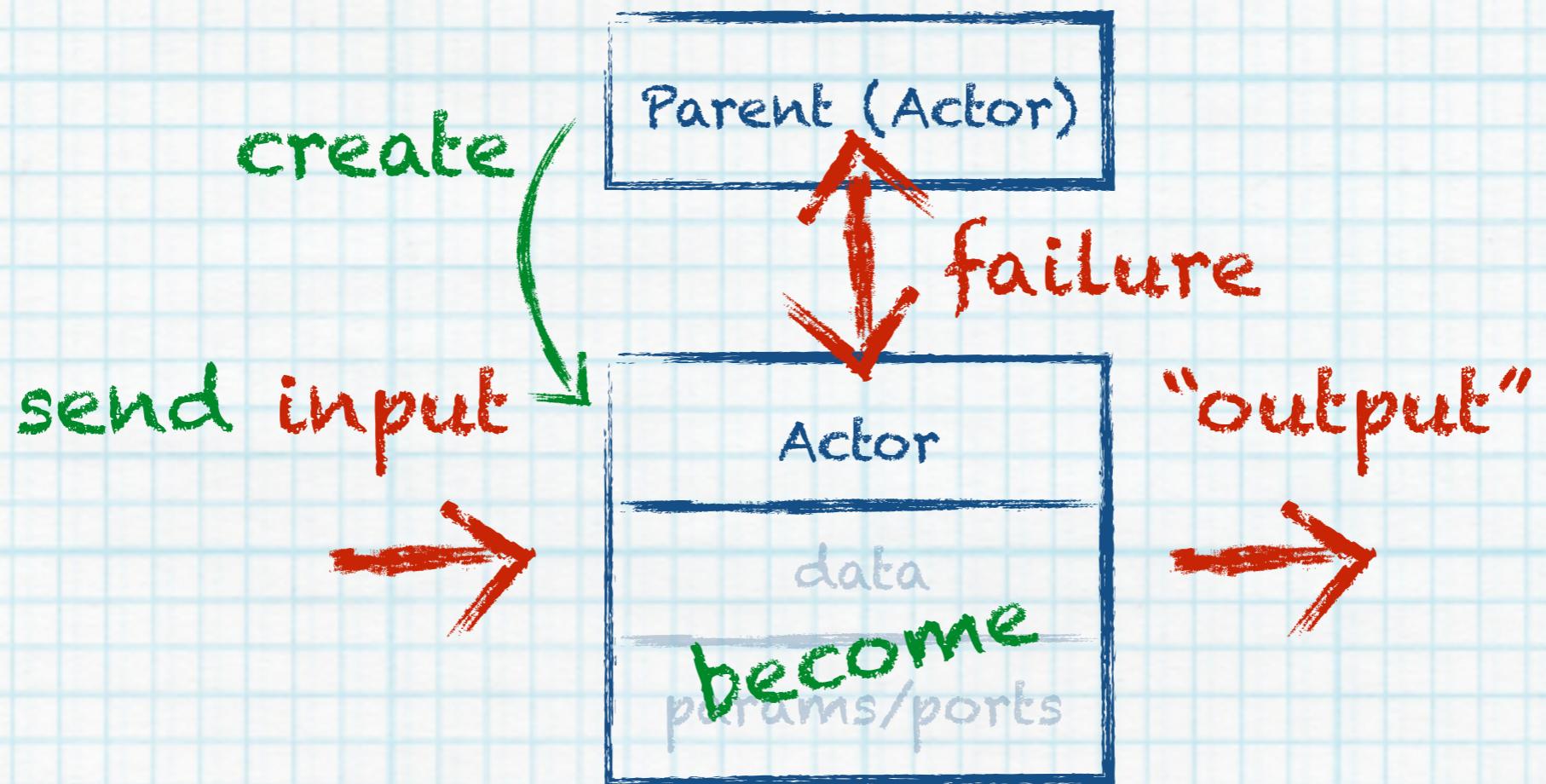
Line: 9 | Java | Soft Tabs: 2 |

greeter.java — Desktop

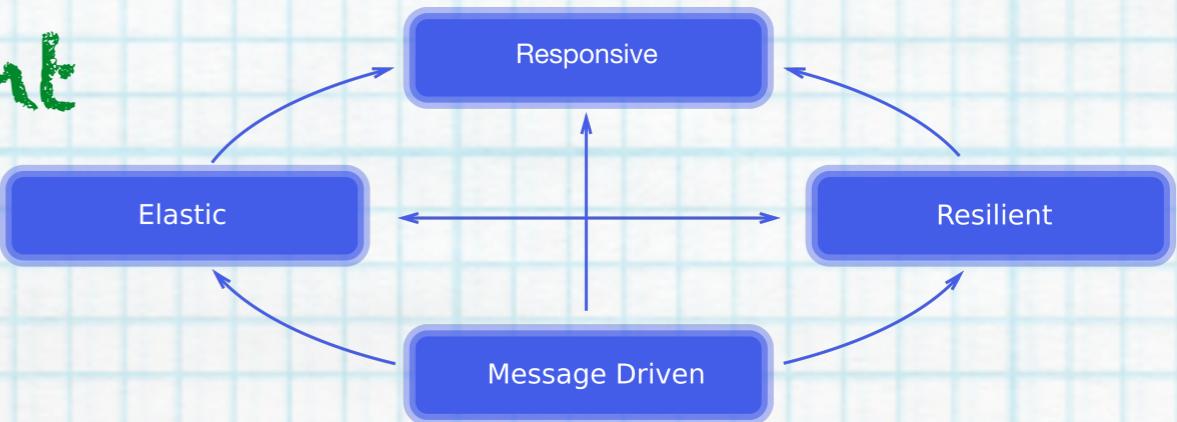
```
1 package sample.hello;
2
3 import akka.actor.UntypedActor;
4
5 public class Greeter extends UntypedActor {
6
7     public static enum Msg {
8         GREET, DONE;
9     }
10
11    @Override
12    public void onReceive(Object msg) {
13        if (msg == Msg.GREET) {
14            System.out.println("Hello World!");
15            getSender().tell(Msg.DONE, getSelf());
16        } else
17            unhandled(msg);
18    }
19
20}
21
```

Line: 21 | Java | Soft Tabs: 2 |

Actors à la Akka



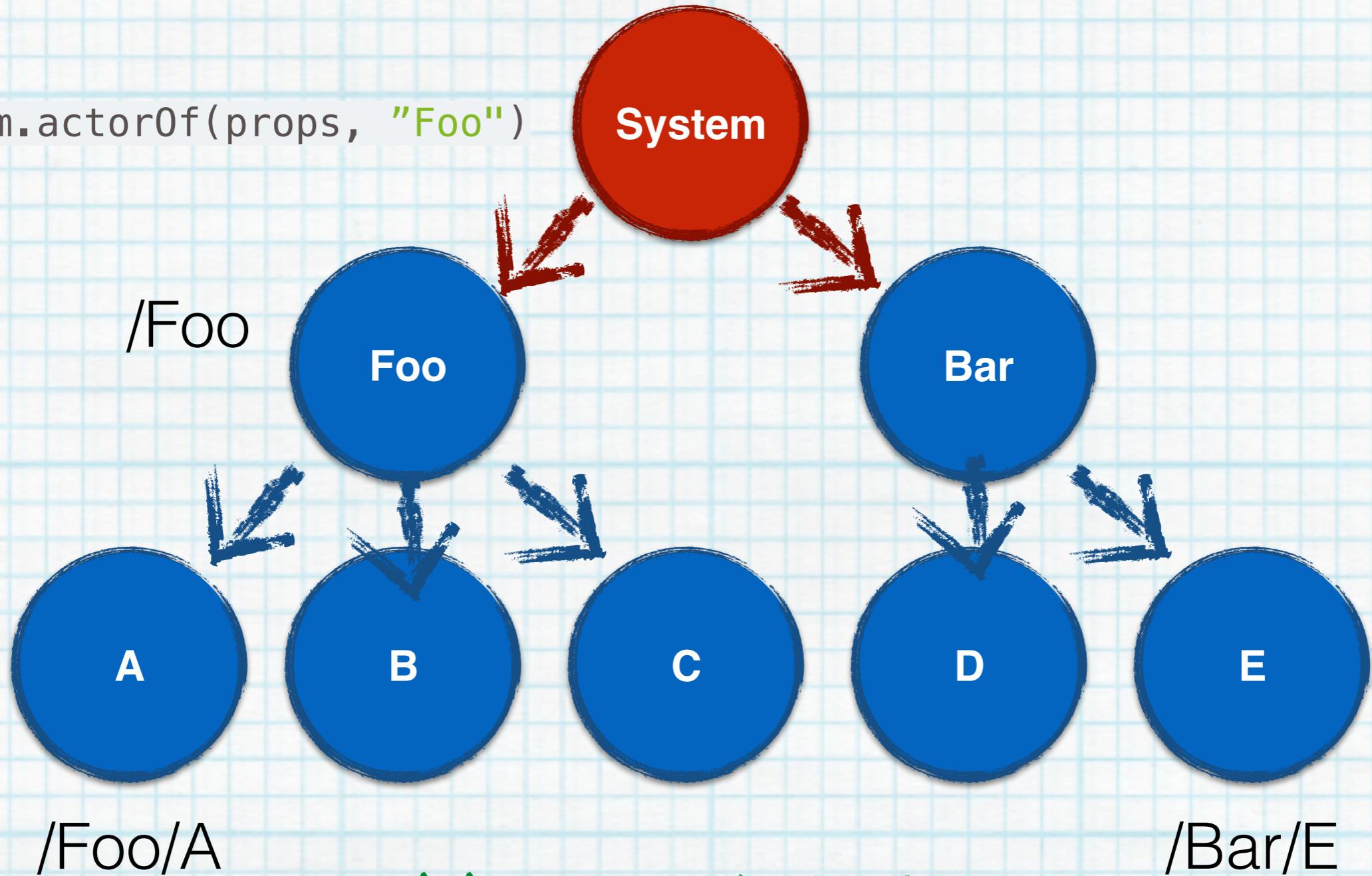
purely reactive component
 (act on receive)



create

Actors à la Akka

```
system.actorOf(props, "Foo")
```



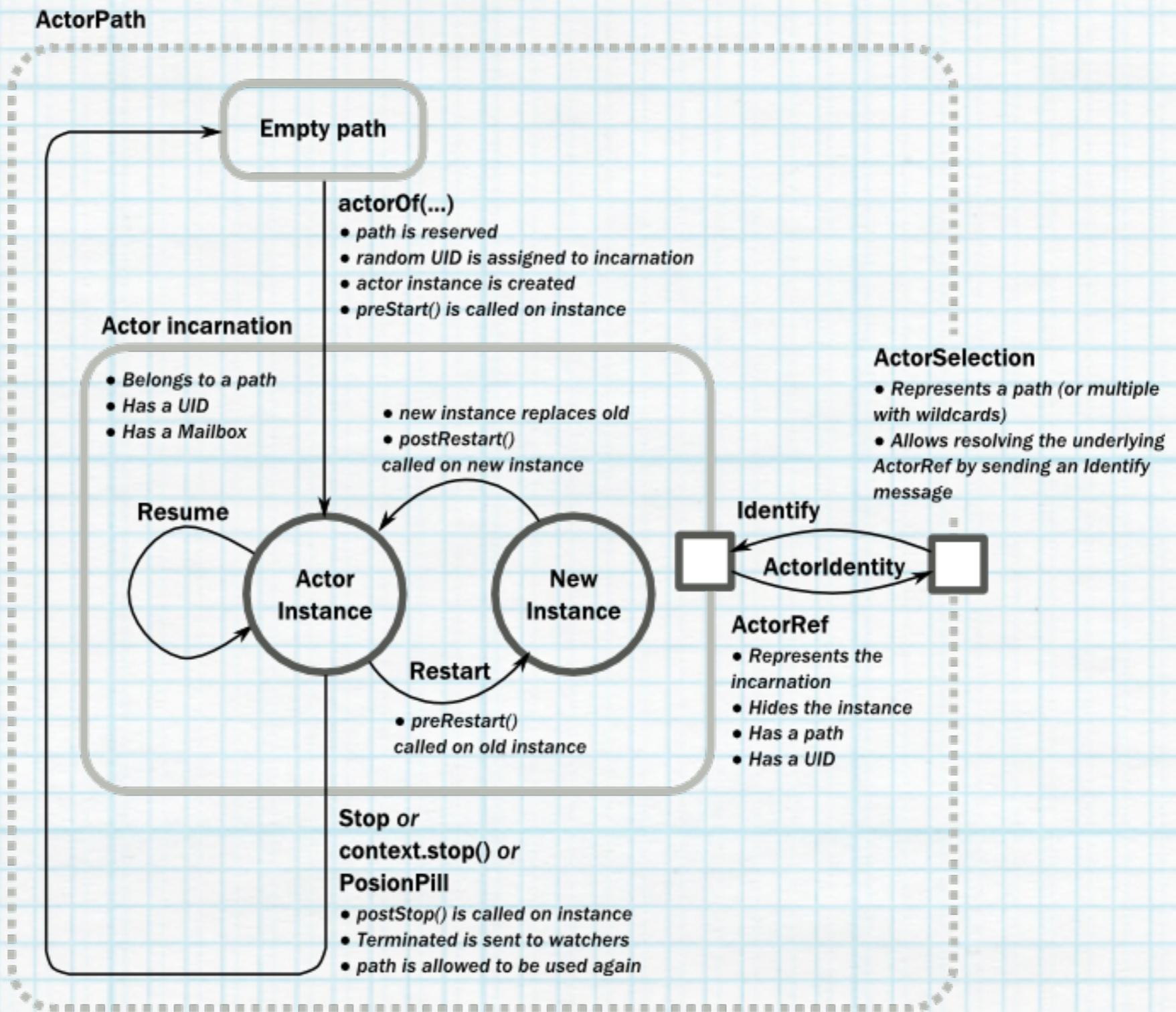
addressing/selection

```
context.actorSelection("/Bar/E")
```

create



Actors à la Akka



send

Actors à la Akka

```
context.actorSelection("/Foo/A").send(msg)
```

Message Delivery Reliability

1. **at-most-once** delivery





**DEAL
WITH
IT.**

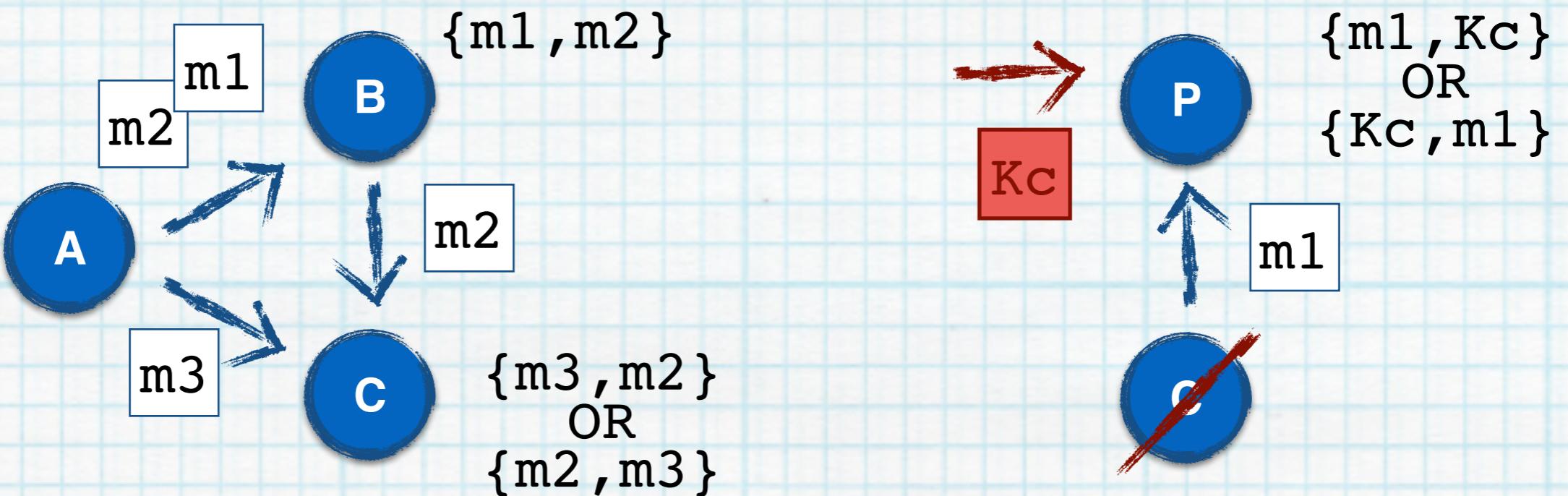
send

Actors à la Akka

```
context.actorSelection("/Foo/A").send(msg)
```

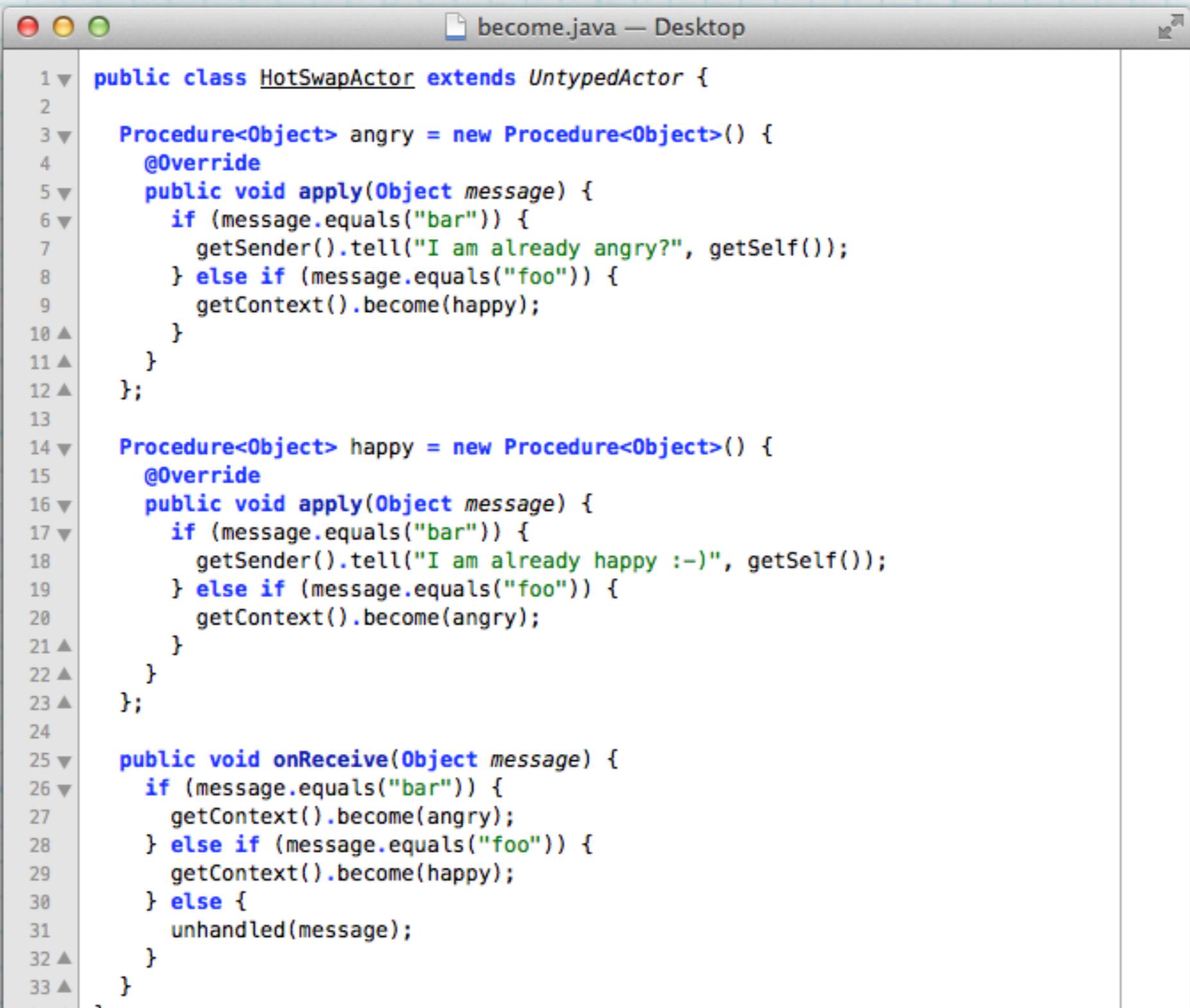
Message Delivery Reliability

1. **at-most-once** delivery
2. message ordering per sender-receiver pair



become

Actors à la Akka



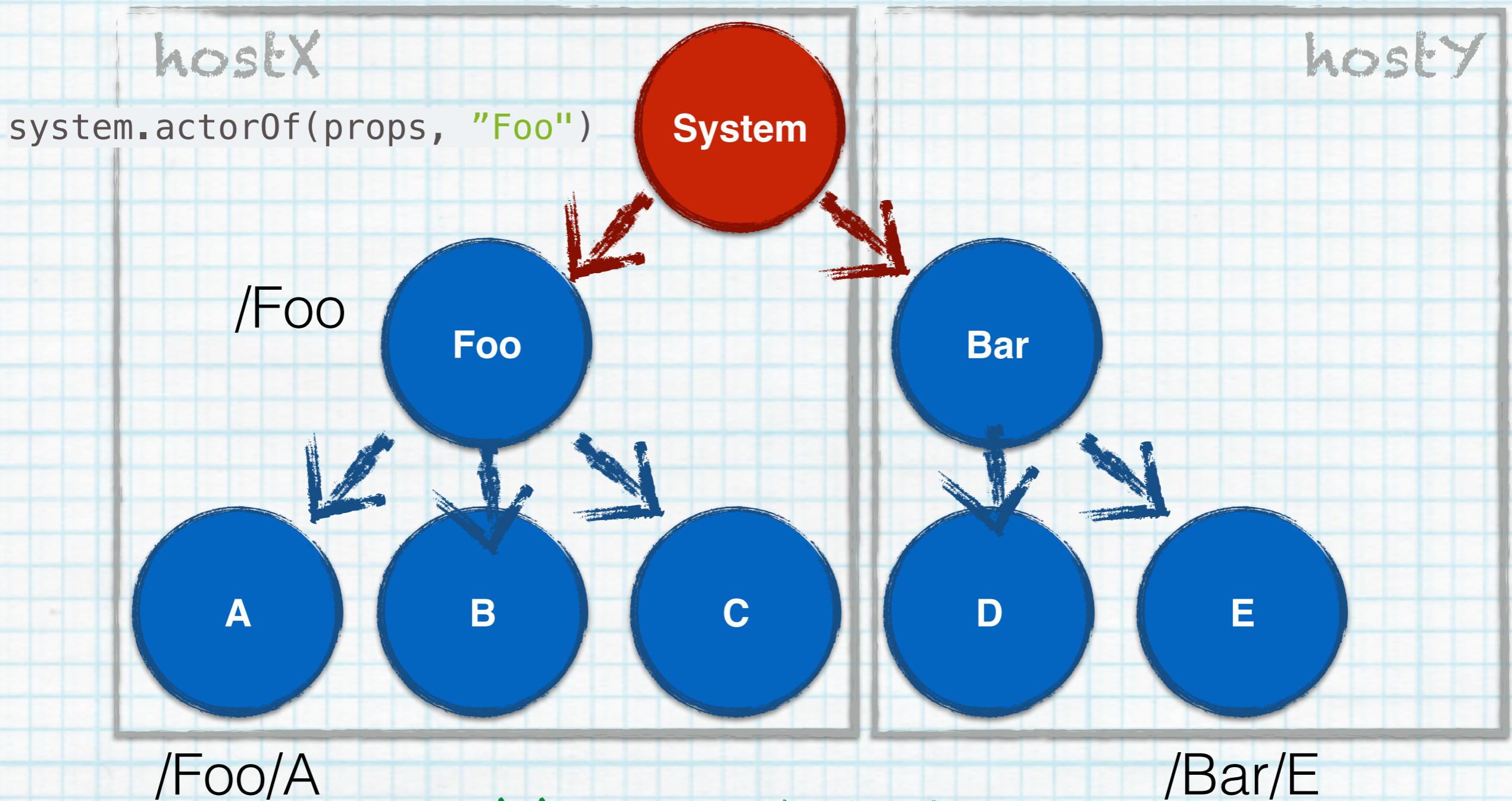
The screenshot shows a Java code editor window titled "become.java — Desktop". The code implements a hot-swapping actor behavior:

```
1 public class HotSwapActor extends UntypedActor {  
2  
3     Procedure<Object> angry = new Procedure<Object>() {  
4         @Override  
5         public void apply(Object message) {  
6             if (message.equals("bar")) {  
7                 getSender().tell("I am already angry?", getSelf());  
8             } else if (message.equals("foo")) {  
9                 getContext().become(happy);  
10            }  
11        }  
12    };  
13  
14    Procedure<Object> happy = new Procedure<Object>() {  
15        @Override  
16        public void apply(Object message) {  
17            if (message.equals("bar")) {  
18                getSender().tell("I am already happy :-)", getSelf());  
19            } else if (message.equals("foo")) {  
20                getContext().become(angry);  
21            }  
22        }  
23    };  
24  
25    public void onReceive(Object message) {  
26        if (message.equals("bar")) {  
27            getContext().become(angry);  
28        } else if (message.equals("foo")) {  
29            getContext().become(happy);  
30        } else {  
31            unhandled(message);  
32        }  
33    }  
}
```

create +remoting



Actors à la Akka



addressing/selection

`context.actorSelection("akka.tcp://system@hostY:1234/Bar/E")`

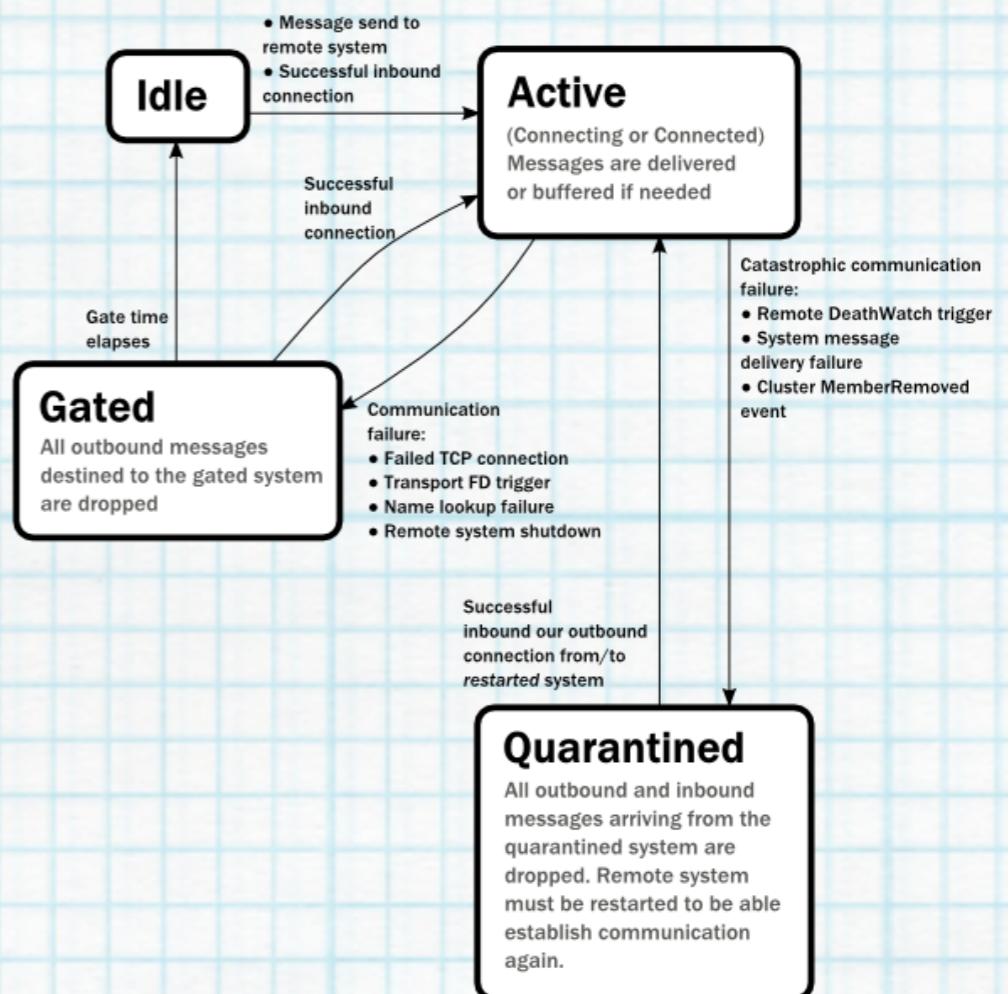
remoting Actors à la Akka

The Phi Accrual Failure Detector

<http://ddg.jaist.ac.jp/pub/HDY+04.pdf>

Routers

Remote Events



Actors à la Akka

clustering

Ring-structured Cluster

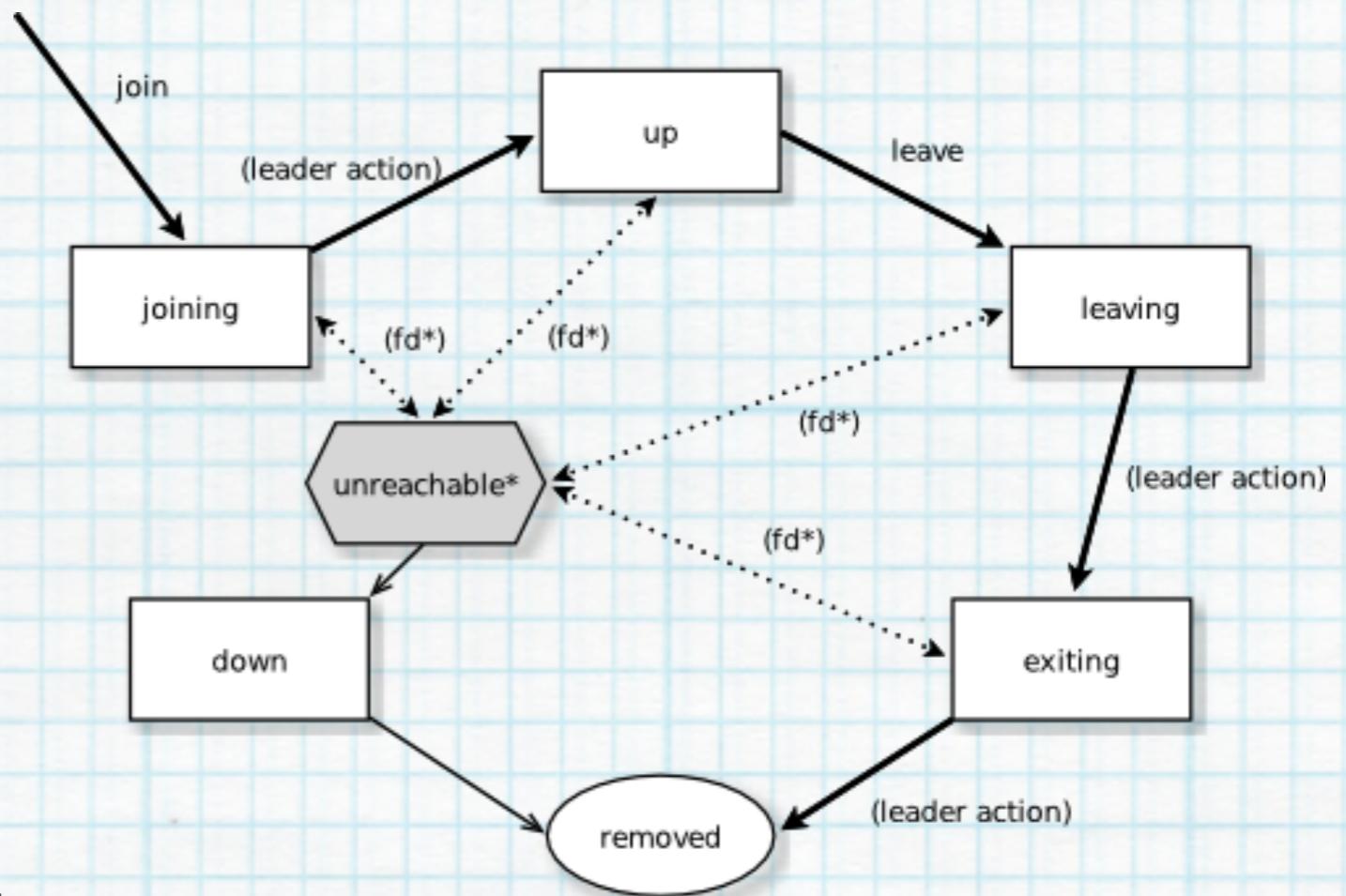
à la Dynamo, Riak

Gossip Protocol

for membership,
leader determination,
configuration

Vector Clocks

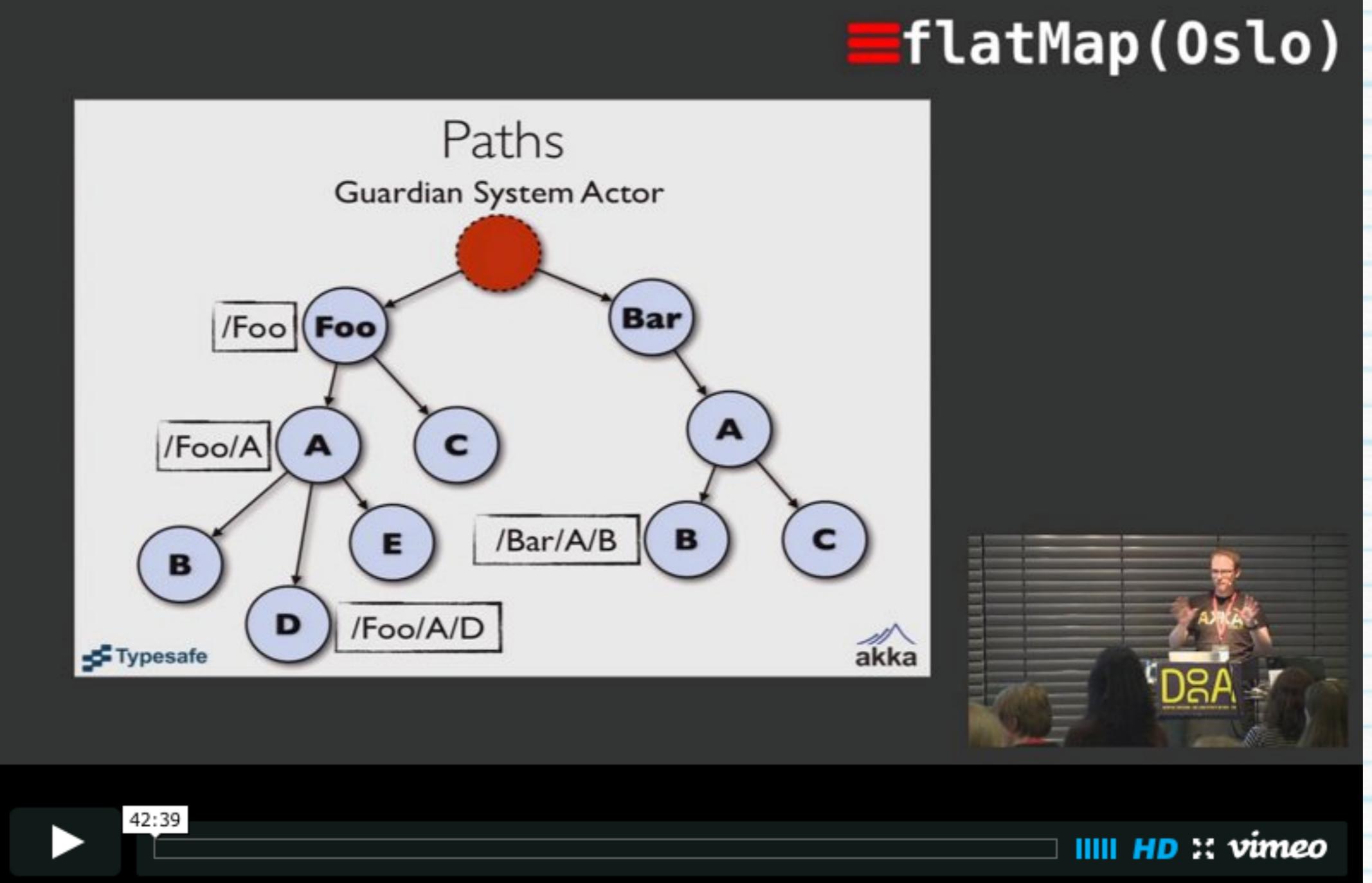
Leaders are ***not*** elected



Actors à la Akka

- Create, send, become
- Parents handle failures
- Purely reactive components
- Remoting with basic guarantees
- Clustering

Actors à la Akka



The diagram illustrates the hierarchical structure of Akka actors. At the top is a red circular node labeled "Guardian System Actor". It has two children: a blue circle labeled "Foo" and a blue circle labeled "Bar". The "Foo" node has three children: a blue circle labeled "A", a blue circle labeled "C", and a blue circle labeled "E". The "A" node under "Foo" has two children: a blue circle labeled "B" and a blue circle labeled "D". The "B" node under "A" has one child: a blue circle labeled "C". The "D" node under "A" has one child: a blue circle labeled "E". The "Bar" node has one child: a blue circle labeled "A". This "A" node under "Bar" has two children: a blue circle labeled "B" and a blue circle labeled "C". The "B" node under this final "A" has one child: a blue circle labeled "C". Labels for actor paths are placed near their respective nodes: "/Foo" is near the "Foo" node; "/Foo/A" is near the first "A" node; "/Bar/A/B" is near the "B" node under the final "A"; and "/Foo/A/D" is near the "D" node under the first "A". The "Typesafe" logo is in the bottom left corner, and the "akka" logo is in the bottom right corner. The video player interface at the bottom shows a play button, a timestamp of "42:39", a progress bar, and a "vimeo" logo.

<http://2013.flatmap.no/klang.html>

The first was for himself. The second was for his country.
This time is for his friend.

STALLONE

?

RAMBO

MARIO KASSAR and ANDREW VAJNA Present
SYLVESTER STALLONE
RAMBO® III

RICHARD CRENNA Music by JERRY GOLDSMITH Director of Photography JOHN STANIER, G.B.C.T.
Associate Producer TONY MUNAFO Executive Producers MARIO KASSAR and ANDREW VAJNA
Based on Characters Created by DAVID MORRELL Written by SYLVESTER STALLONE and SHELDON LETTICH
Produced by BUZZ FEITSHANS Directed by PETER MACDONALD ATRI-STAR RELEASE



R
RESTRICTED
UNDER 17 REQUIRES ACCOMPANYING
PARENT OR ADULT GUARDIAN

Read the Jove Novel by DAVID MORRELL
Rambo® is a Registered Trademark owned by CAROLCO.

DOLBY STEREO
IN SELECTED THEATRES

©1988 Tri-Star Pictures, Inc. All Rights Reserved.



PL@NES - Reading Club
Actors à la Akka

Christophe.VanGinneken@cs.kuleuven.be

KU LEUVEN

DistriNet
Minds KU LEUVEN

Christophe.VanGinneken@cs.kuleuven.be

<http://www.slideshare.net/christophevg/actors-la-akka>