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

Christophe.VanGinneken@cs.kuleuven.be

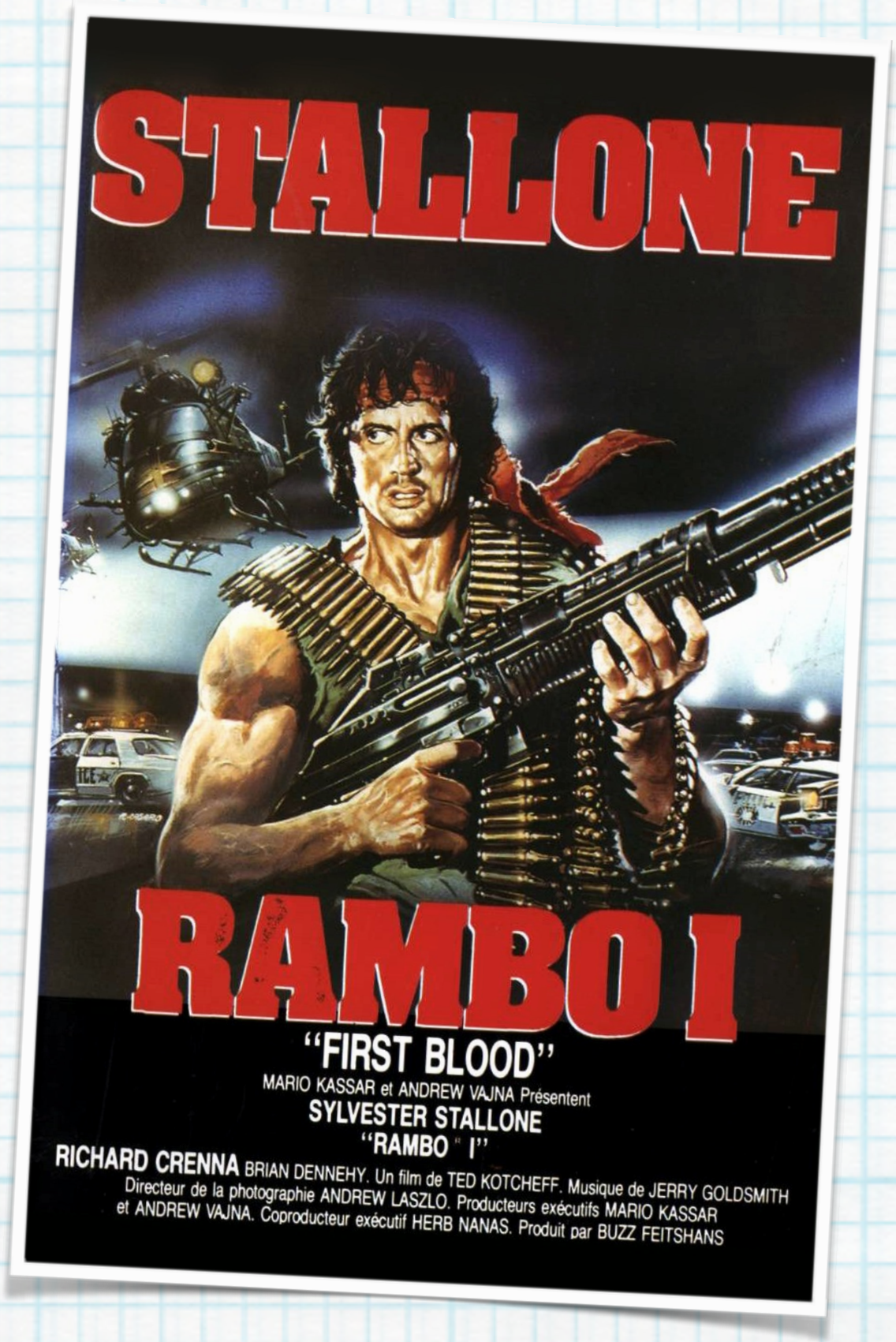
KU LEUVEN

DistriNet
iMinds KU LEUVEN

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…

christophe.vg

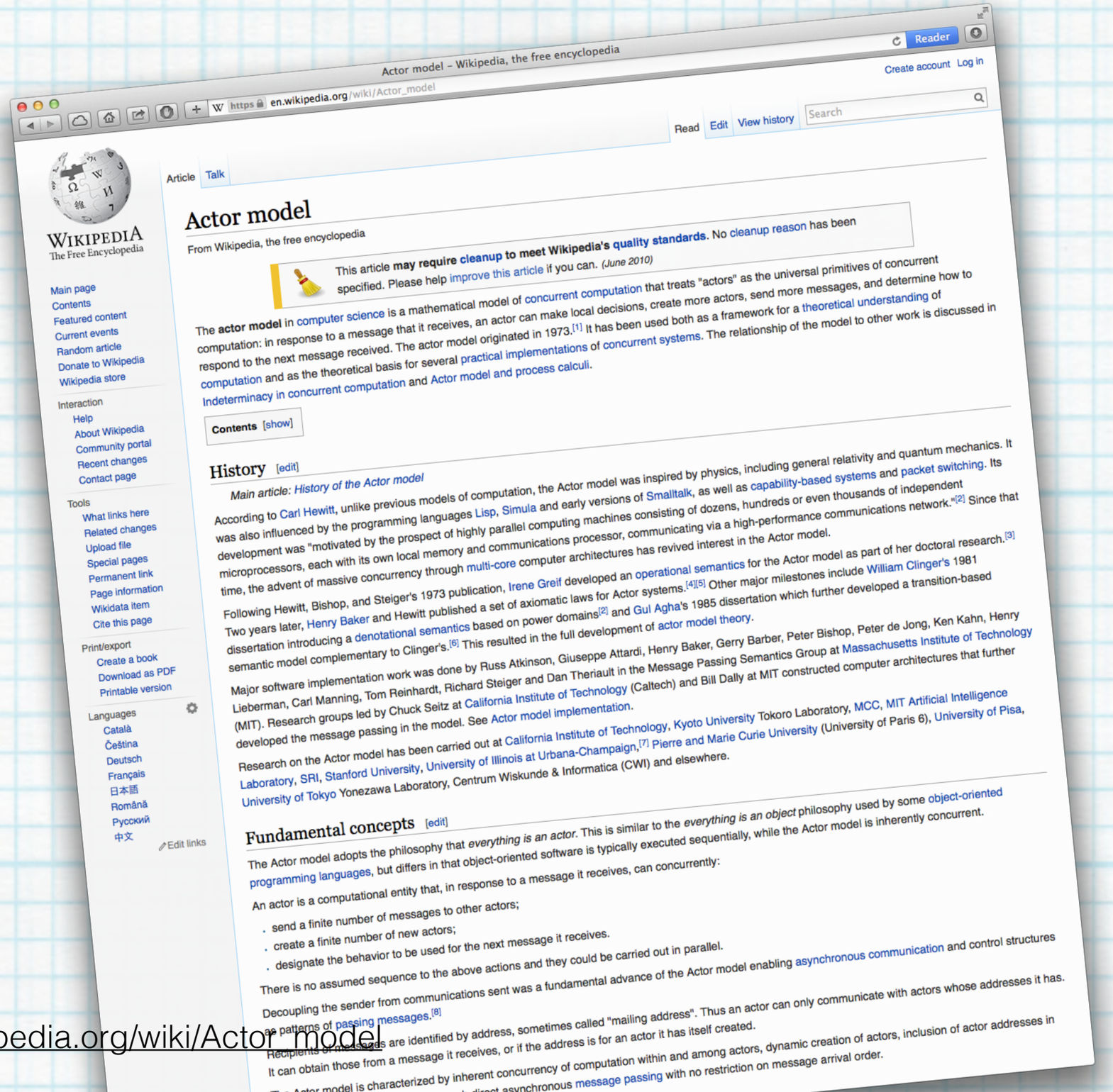
distrinet.cs.kuleuven.be


STALLONE
RAMBO I
"FIRST BLOOD"
MARIO KASSAR et ANDREW VAJNA Présentent
SYLVESTER STALLONE
"RAMBO I"
RICHARD CRENNA BRIAN DENNEHY. Un film de TED KOTCHEFF. Musique de JERRY GOLDSMITH
Directeur de la photographie ANDREW LASZLO. Producteurs exécutifs MARIO KASSAR
et ANDREW VAJNA. Coproducteur exécutif HERB NANAS. Produit par BUZZ FEITSHANS

What are Actors?

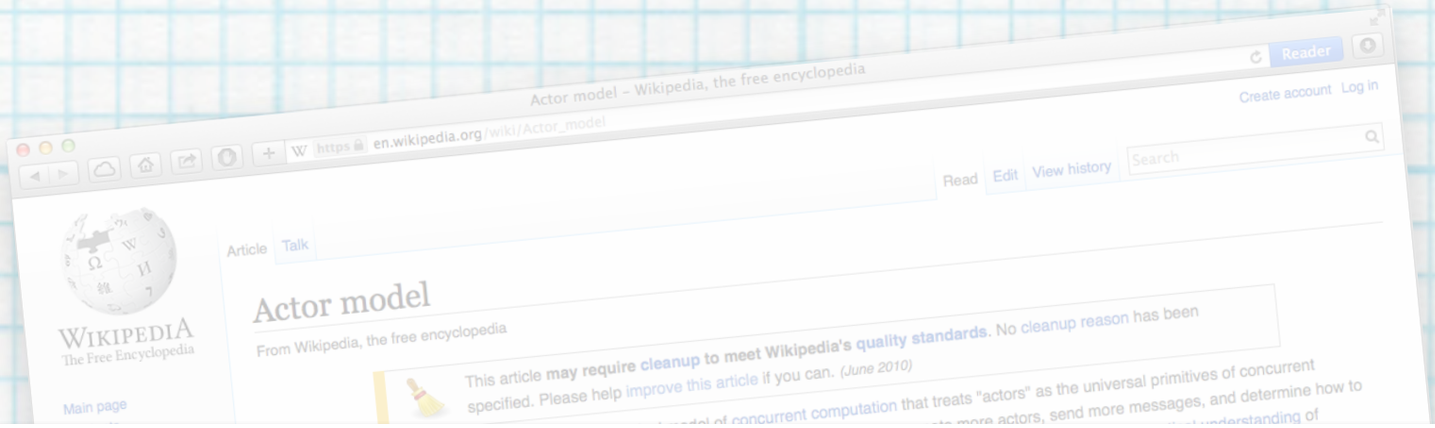
What are Actors?

The Actor Model

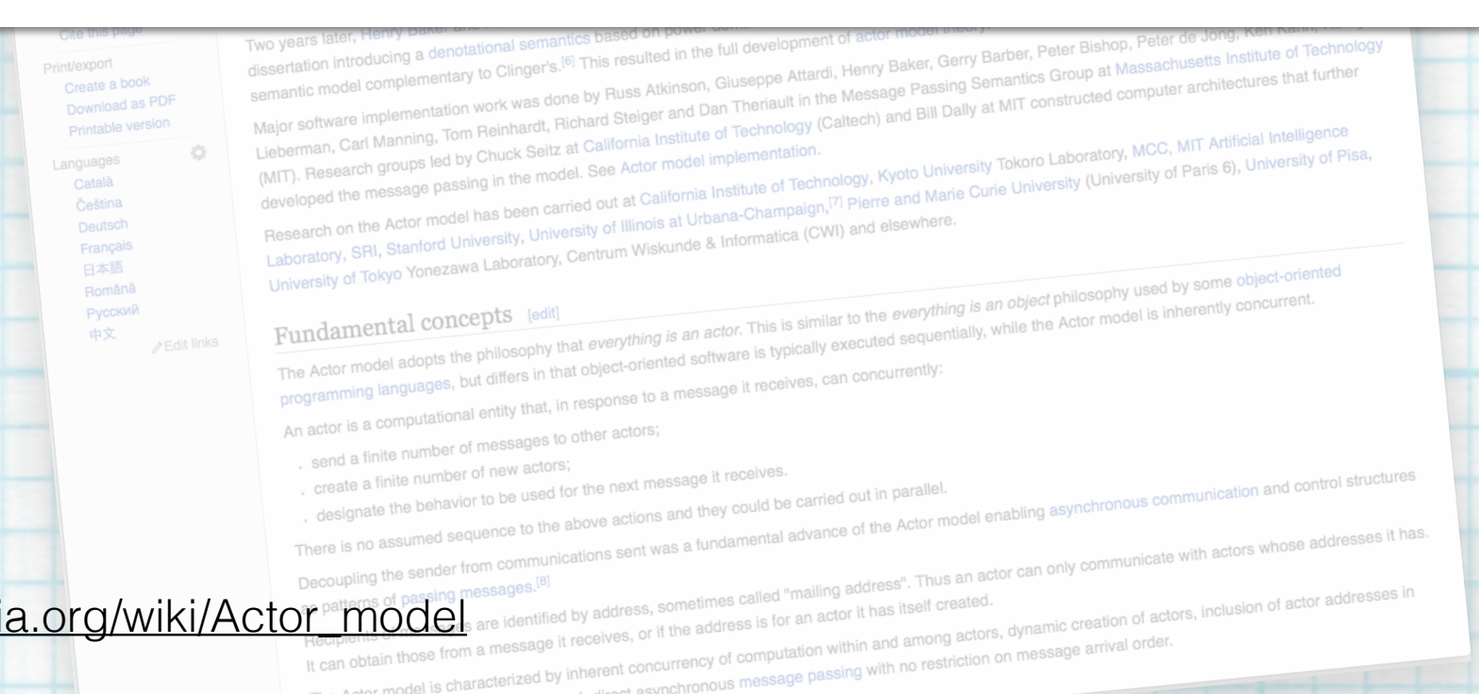


source: https://en.wikipedia.org/wiki/Actor_model

The Actor Model

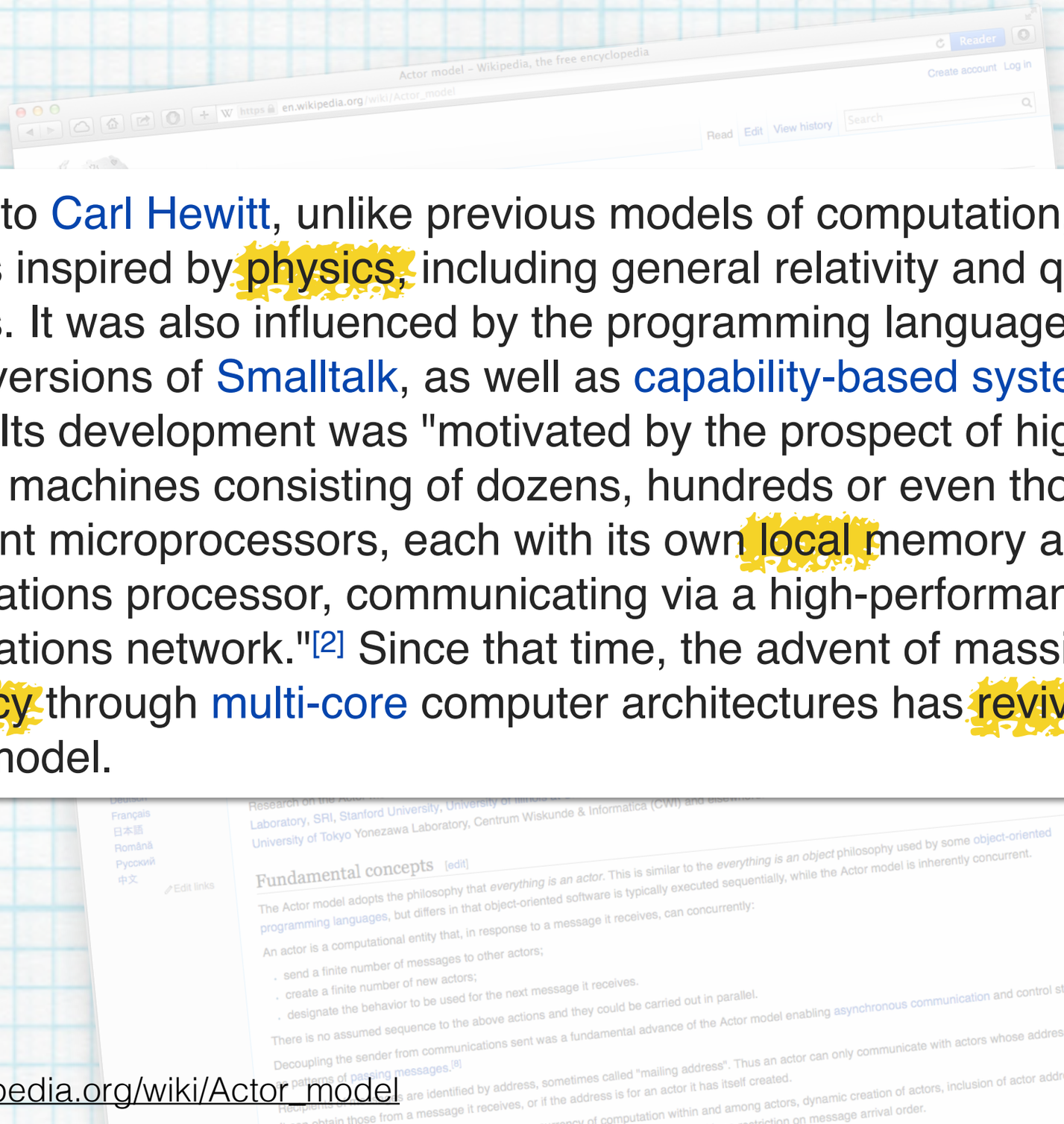


The **actor model** in computer science is a mathematical model of concurrent computation that treats "actors" as the universal primitives of concurrent computation: in response to a message that it receives, an actor can make local decisions, create more actors, send more messages, and determine how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



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.

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.

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.

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

The Actor Model

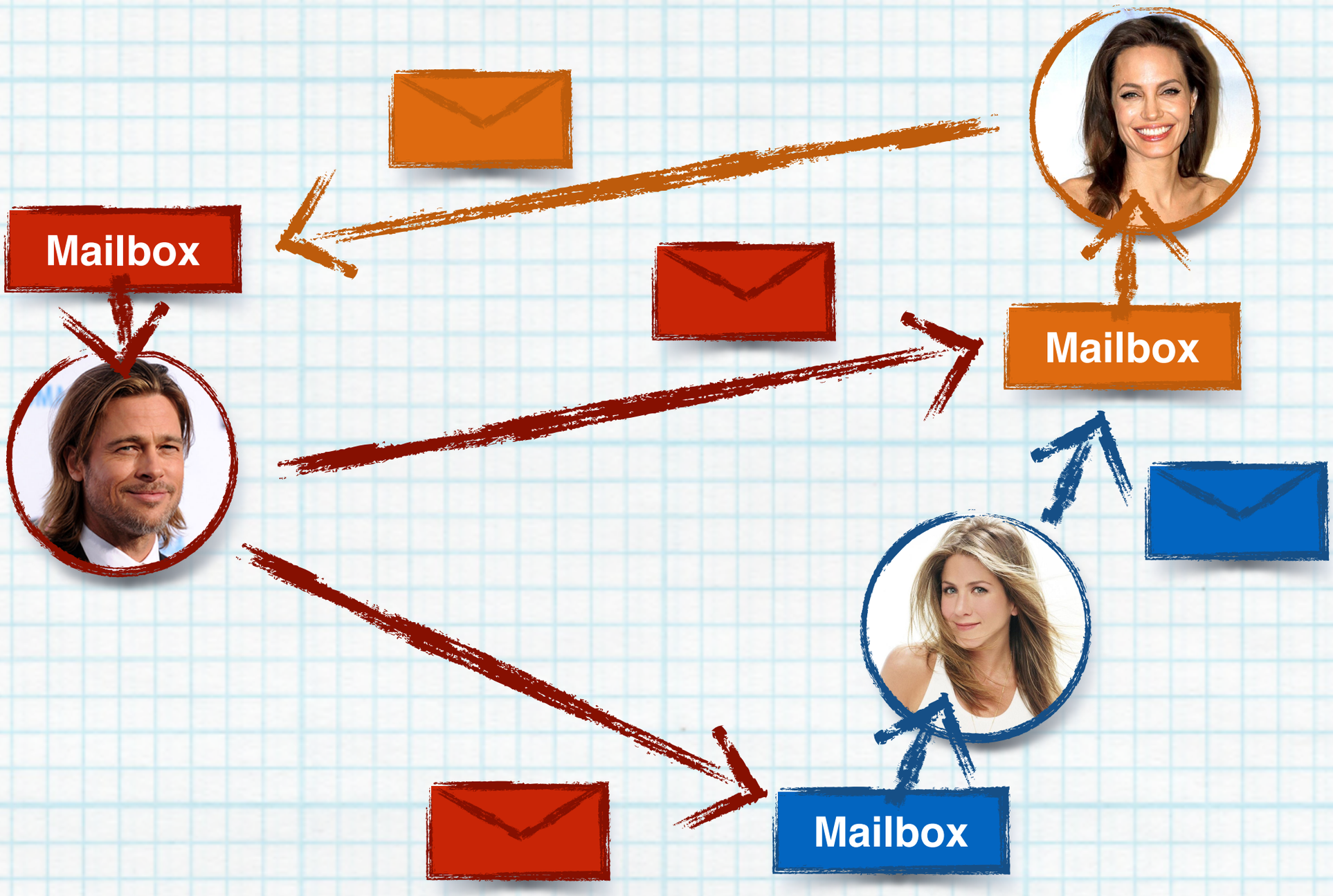


diagram based on: http://blog.scottlogic.com/2014/08/15/using-akka-and-scala-to-render-a-mandelbrot-set.html

The Actor Model

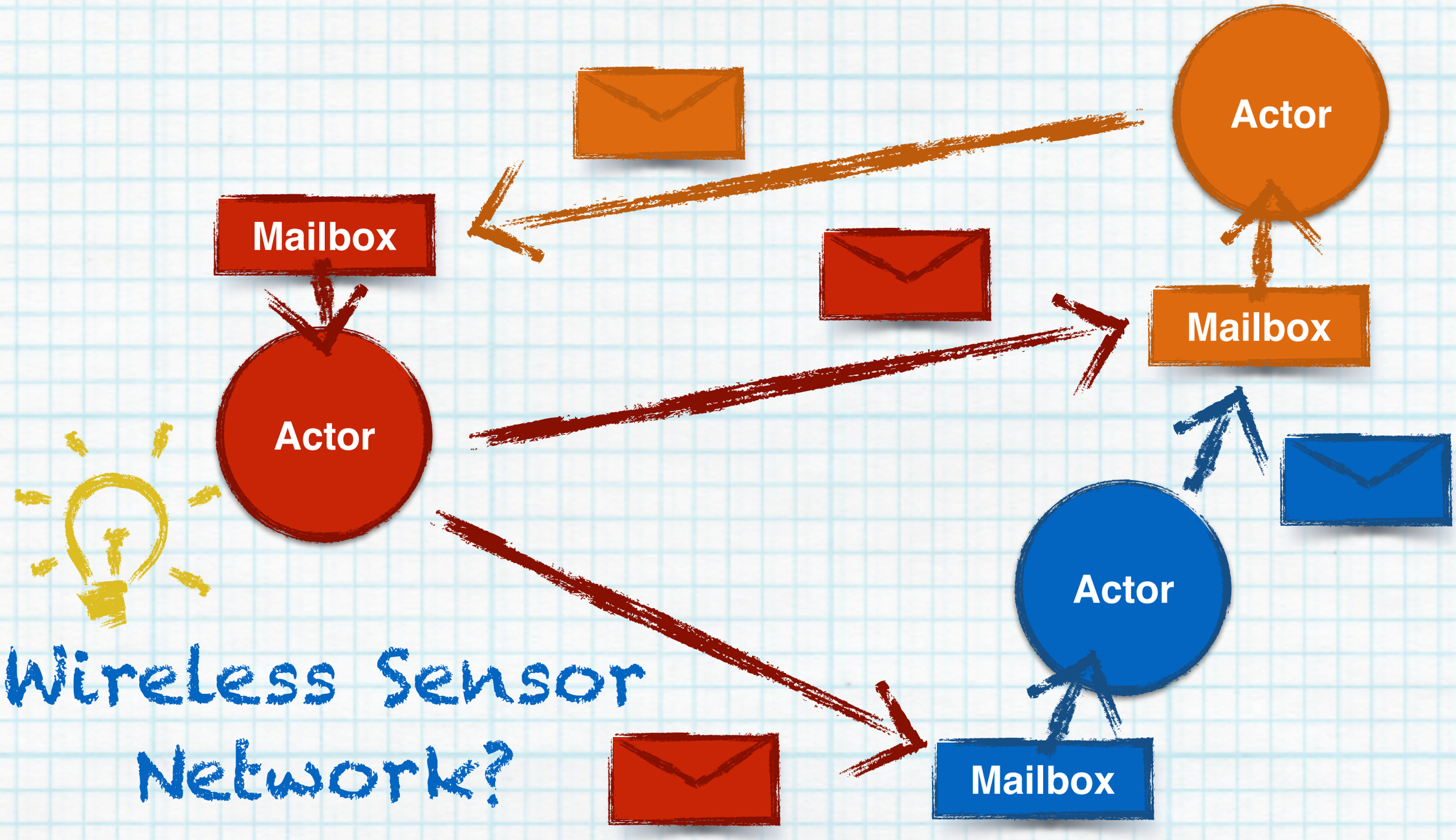
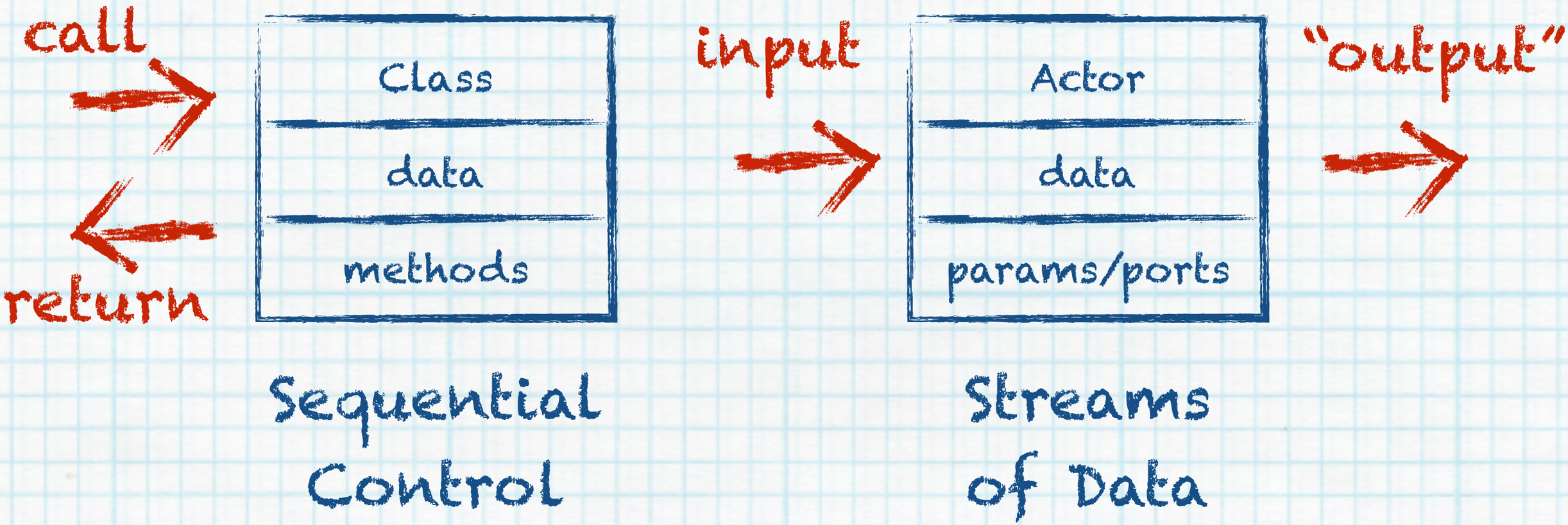


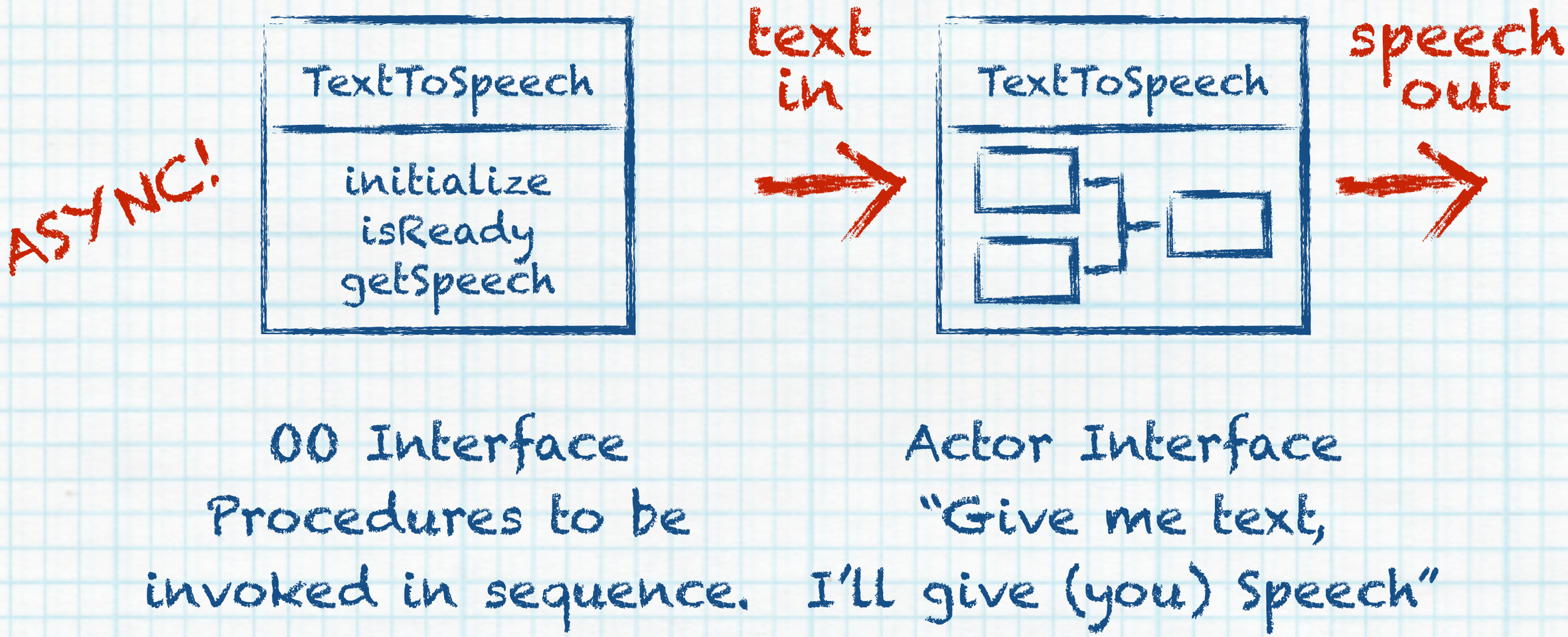
diagram based on: http://blog.scottlogic.com/2014/08/15/using-akka-and-scala-to-render-a-mandelbrot-set.html

OOP vs Actor Model



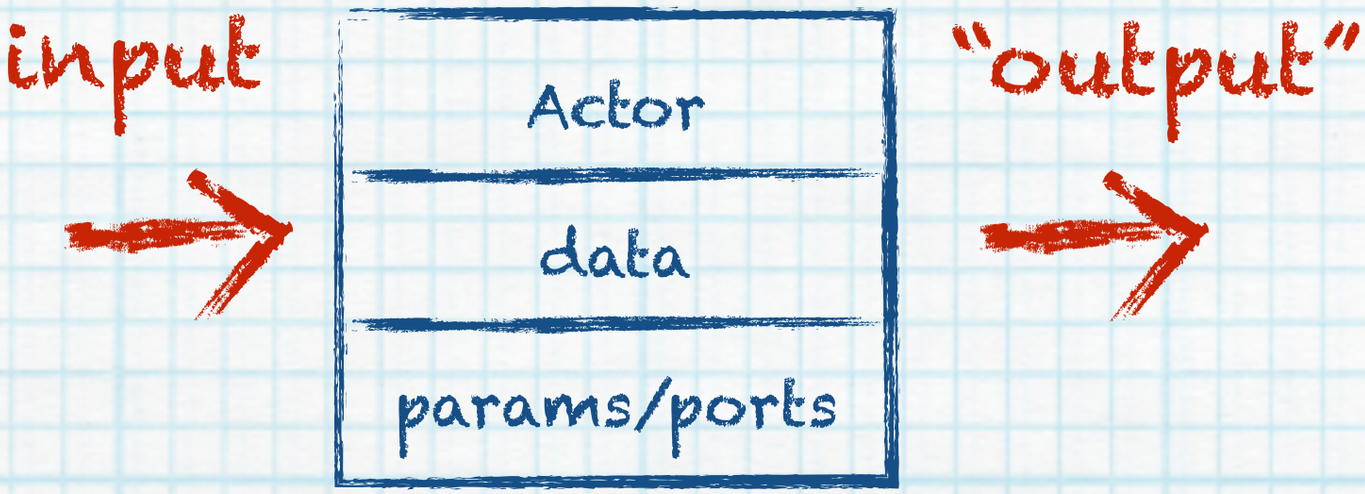
source: Lee, Edward A. "Model-driven development-from object-oriented design to actor-oriented design." Workshop on Software Engineering for Embedded Systems: From Requirements to Implementation (aka The Monterey Workshop), Chicago. 2003.

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

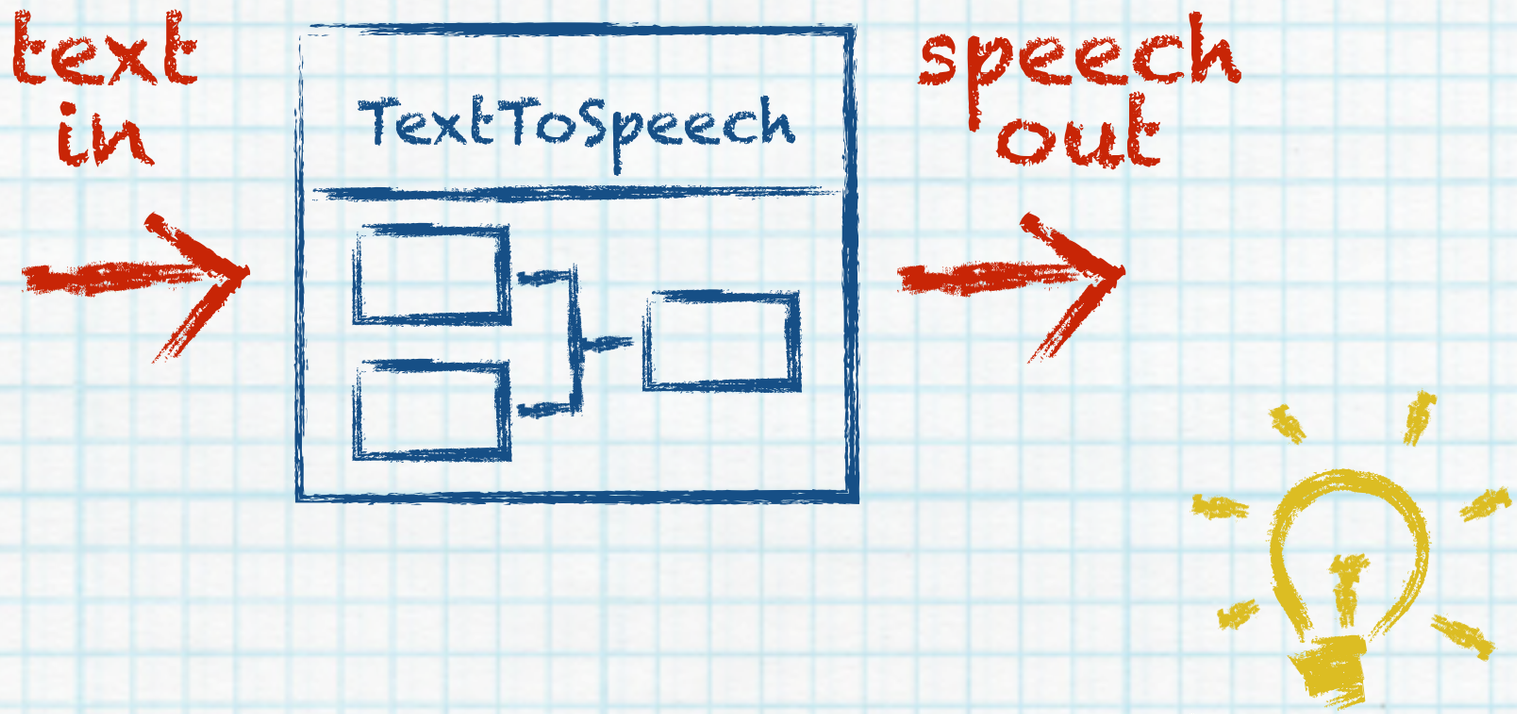


Streams of Data

Reactive Programming ?!

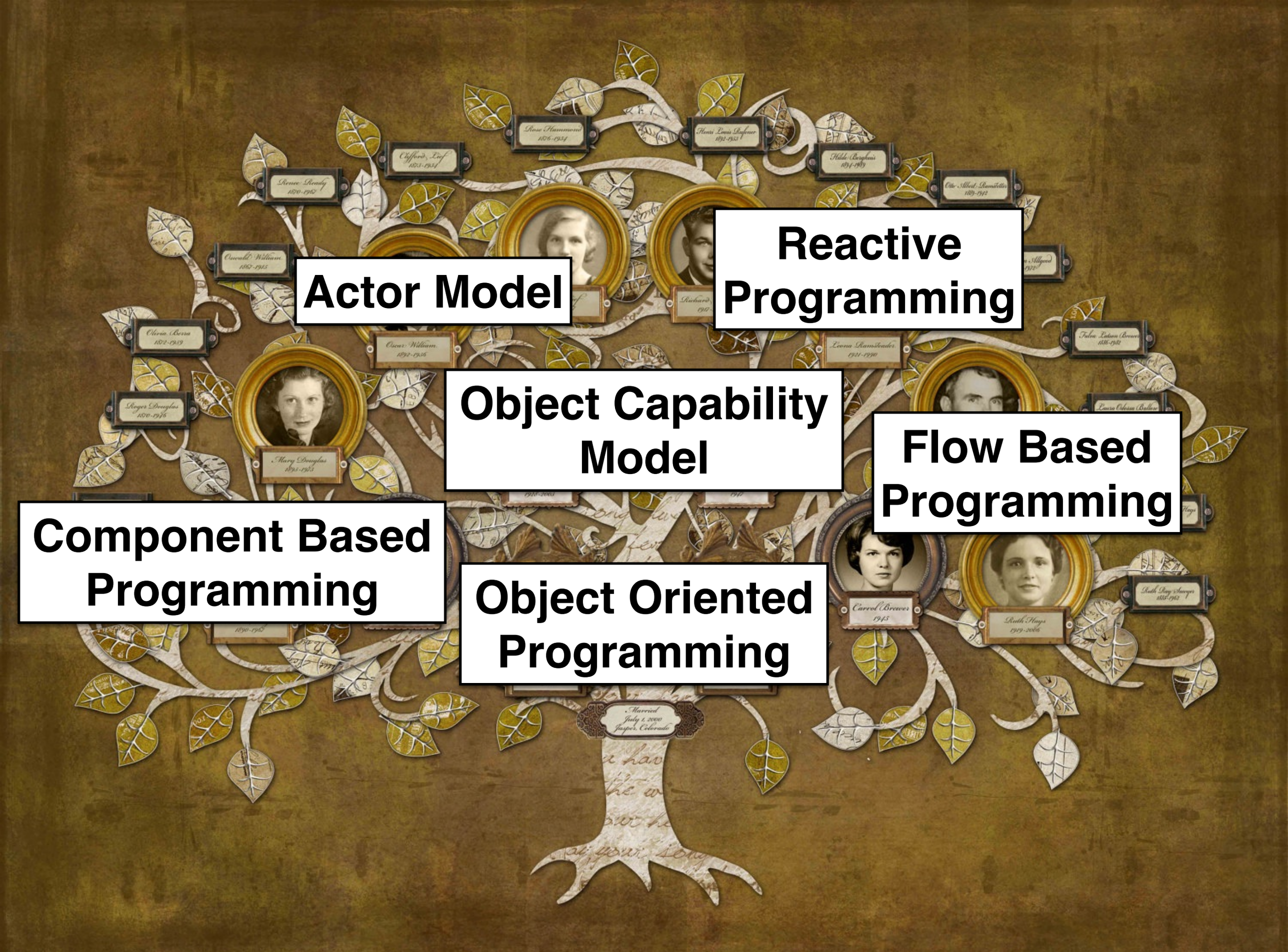
source: Lee, Edward A. "Model-driven development-from object-oriented design to actor-oriented design." Workshop on Software Engineering for Embedded Systems: From Requirements to Implementation (aka The Monterey Workshop), Chicago. 2003.

OOP vs Actor Model

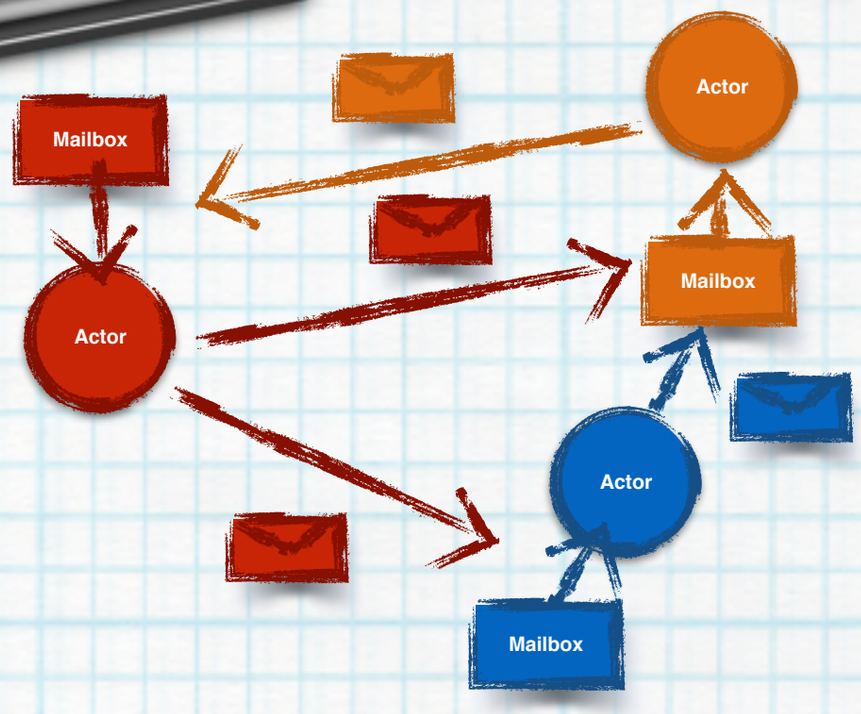


Component Based Programming ?!

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
Flow Based Programming
Component Based Programming
Object Oriented Programming

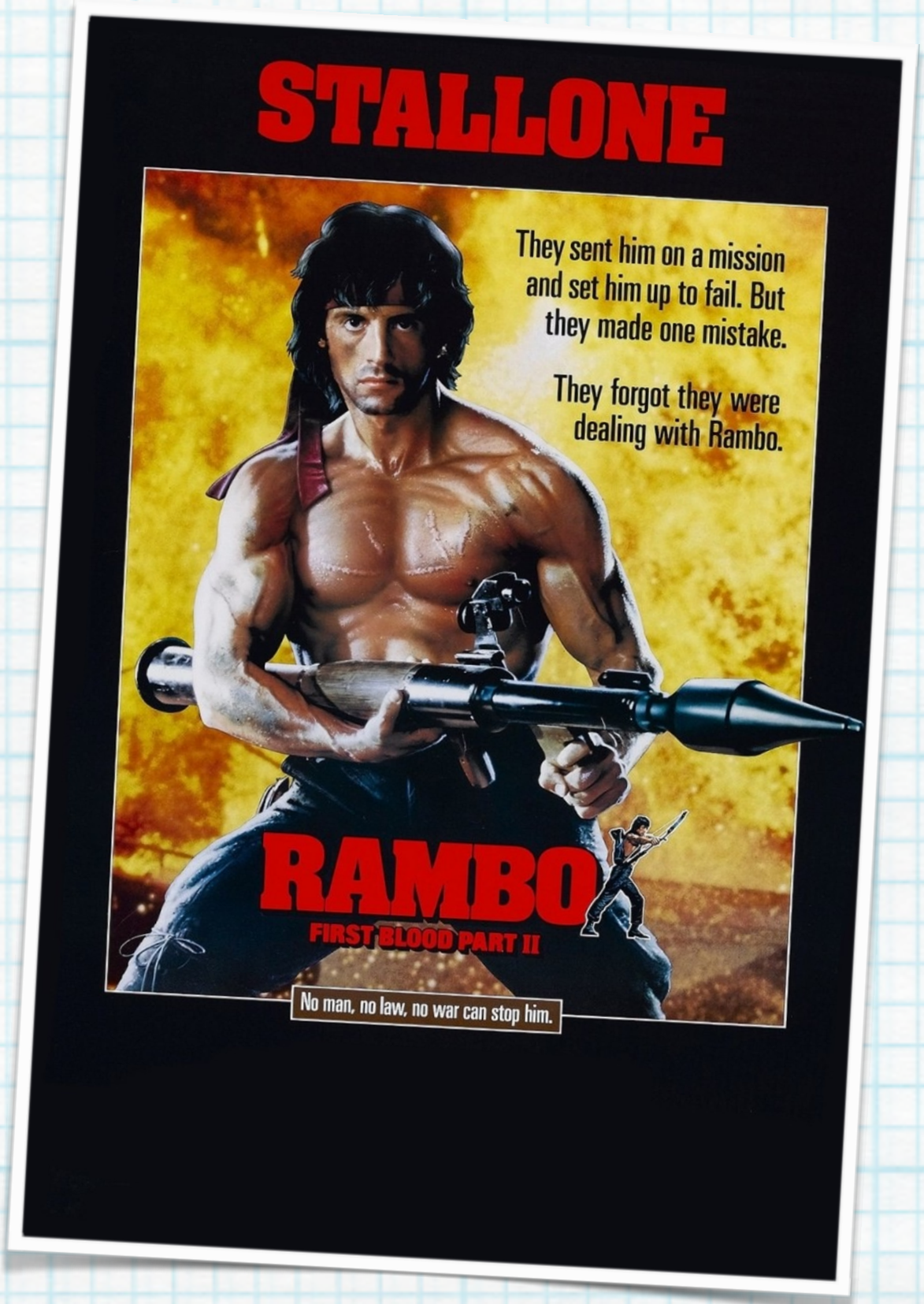
Success?



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

source: http://www.doc.ic.ac.uk/~nd/surprise_97/journal/vol2/pjm2/


STALLONE
They sent him on a mission and set him up to fail. But they made one mistake.
They forgot they were dealing with Rambo.
RAMBO
FIRST BLOOD PART II
No man, no law, no war can stop him.

What is Akka?

Scalable real-time transactio ocessing

We beli ing c ect c t, fault-tolerant and scalable ap ications i
Most im 's be e we are using the wron ls d t
evel of abstrac n. s here that. Usi r Mo we se
bstraction lev and p de a bet platfor scala resili an
onsive a cations—see the Rea e M festo for mo etails. r fa
w dopt the "let it cr " m el wh the ecom ustry u used with
s to bu ti th self-he d systems t never stop. Actors
lso provide the ab ction t nt dist d the basis for truly
scalable a fault-to nt a ca s.
A ka is O ource labl der the Apache 2 License.

akka

hello.java — Desktop

```java
package sample.hello;

import akka.actor.Props;
import akka.actor.UntypedActor;
import akka.actor.ActorRef;

public class HelloWorld extends UntypedActor {

  @Override
  public void preStart() {
    // create the greeter actor
    final ActorRef greeter = getContext().actorOf(Props.create(Greeter.class), "greeter");
    // tell it to perform the greeting
    greeter.tell(Greeter.Msg.GREET, getSelf());
  }

  @Override
  public void onReceive(Object msg) {
    if (msg == Greeter.Msg.DONE) {
      // when the greeter is done, stop this actor and with it the app
      getContext().stop(getSelf());
    } else
      unhandled(msg);
  }
}
```

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

greeter.java — Desktop

```java
package sample.hello;

import akka.actor.UntypedActor;

public class Greeter extends UntypedActor {

  public static enum Msg {
    GREET, DONE;
  }

  @Override
  public void onReceive(Object msg) {
    if (msg == Msg.GREET) {
      System.out.println("Hello World!");
      getSender().tell(Msg.DONE, getSelf());
    } else
      unhandled(msg);
  }

}

```

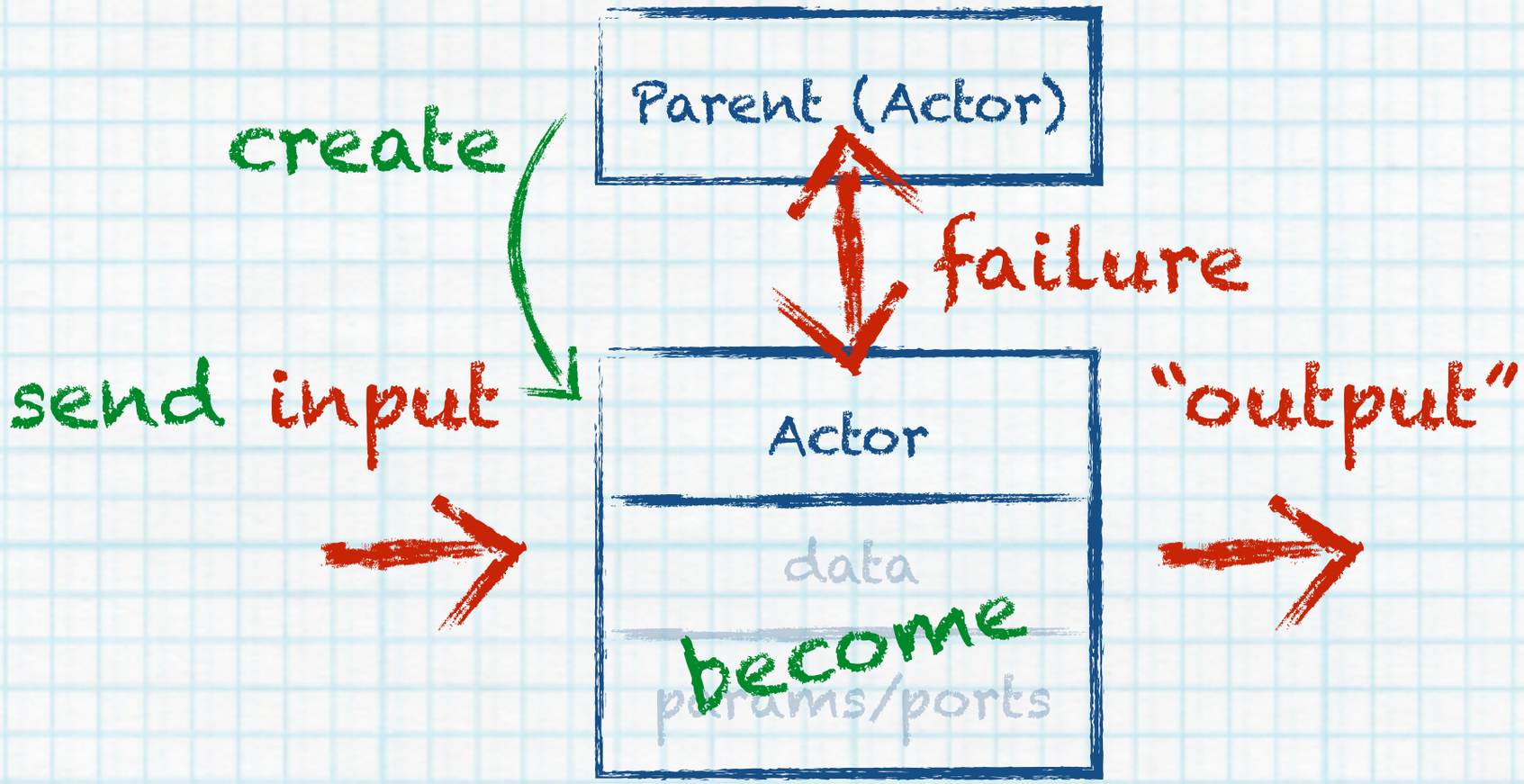
Line: 21 | Java | Soft Tabs: 2

main.java — Desktop

```java
package sample.hello;

public class Main {

  public static void main(String[] args) {
    akka.Main.main(new String[] { HelloWorld.class.getName() });
  }
}

```

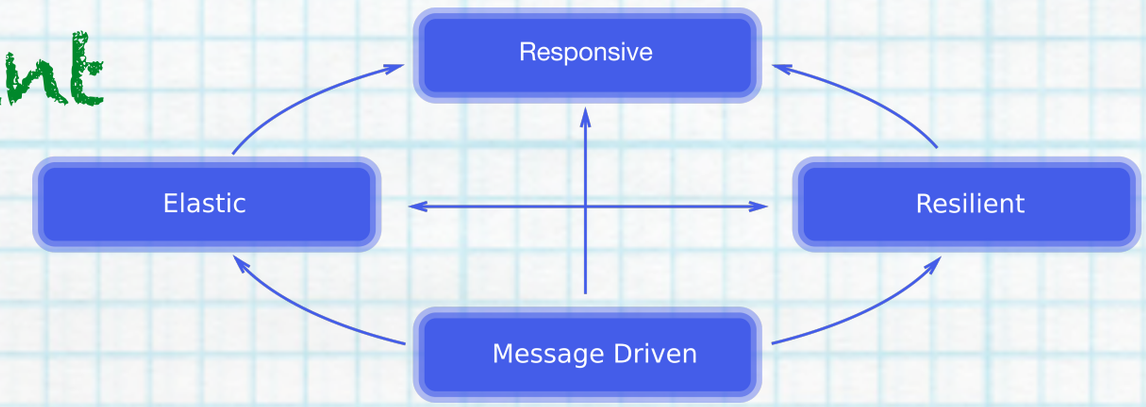
Line: 9 | Java | Soft Tabs: 2

Actors à la Akka



purely reactive component

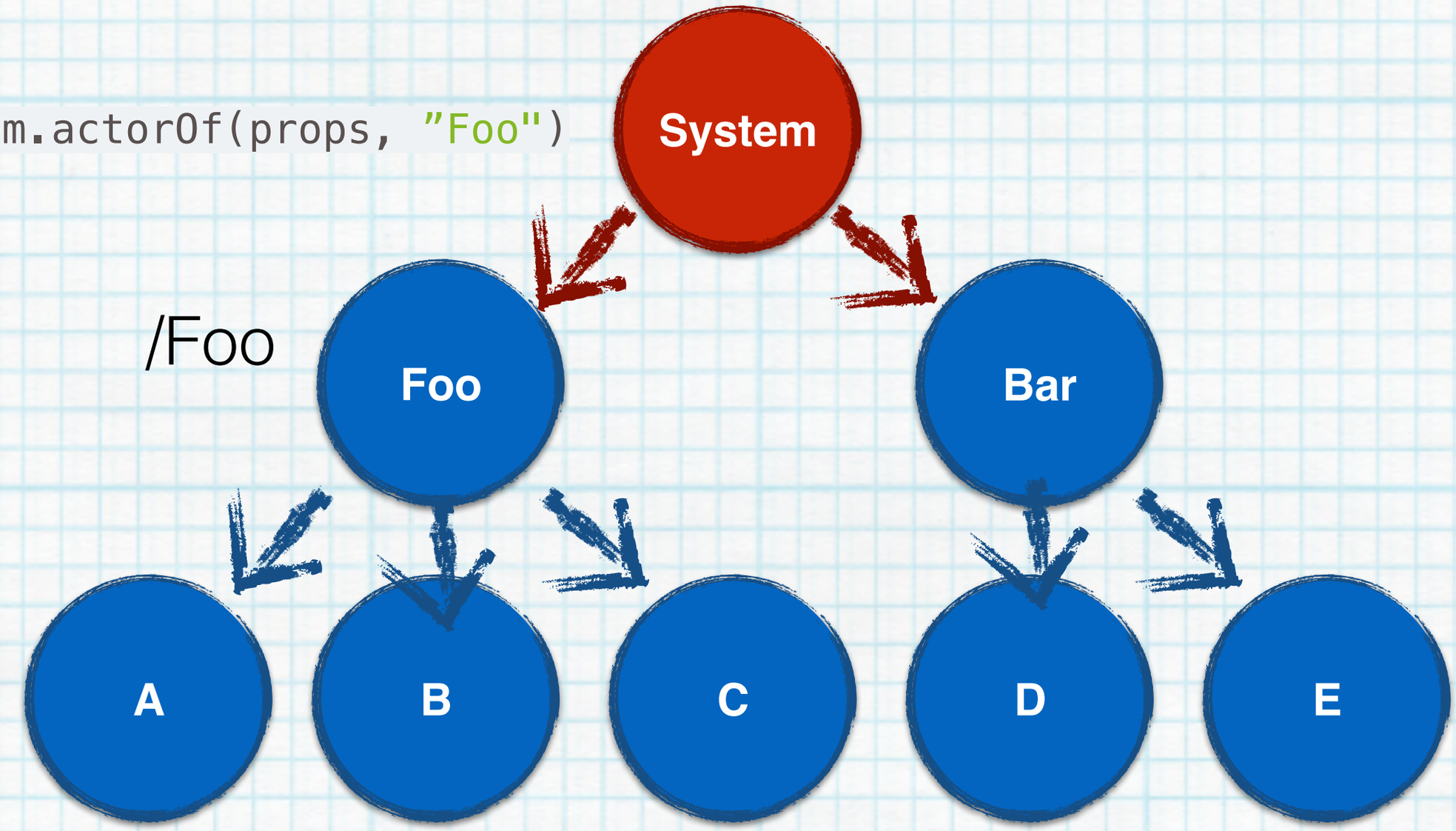
(act on receive)



create

Actors à la Akka

```
system.actorOf(props, ”Foo")
```

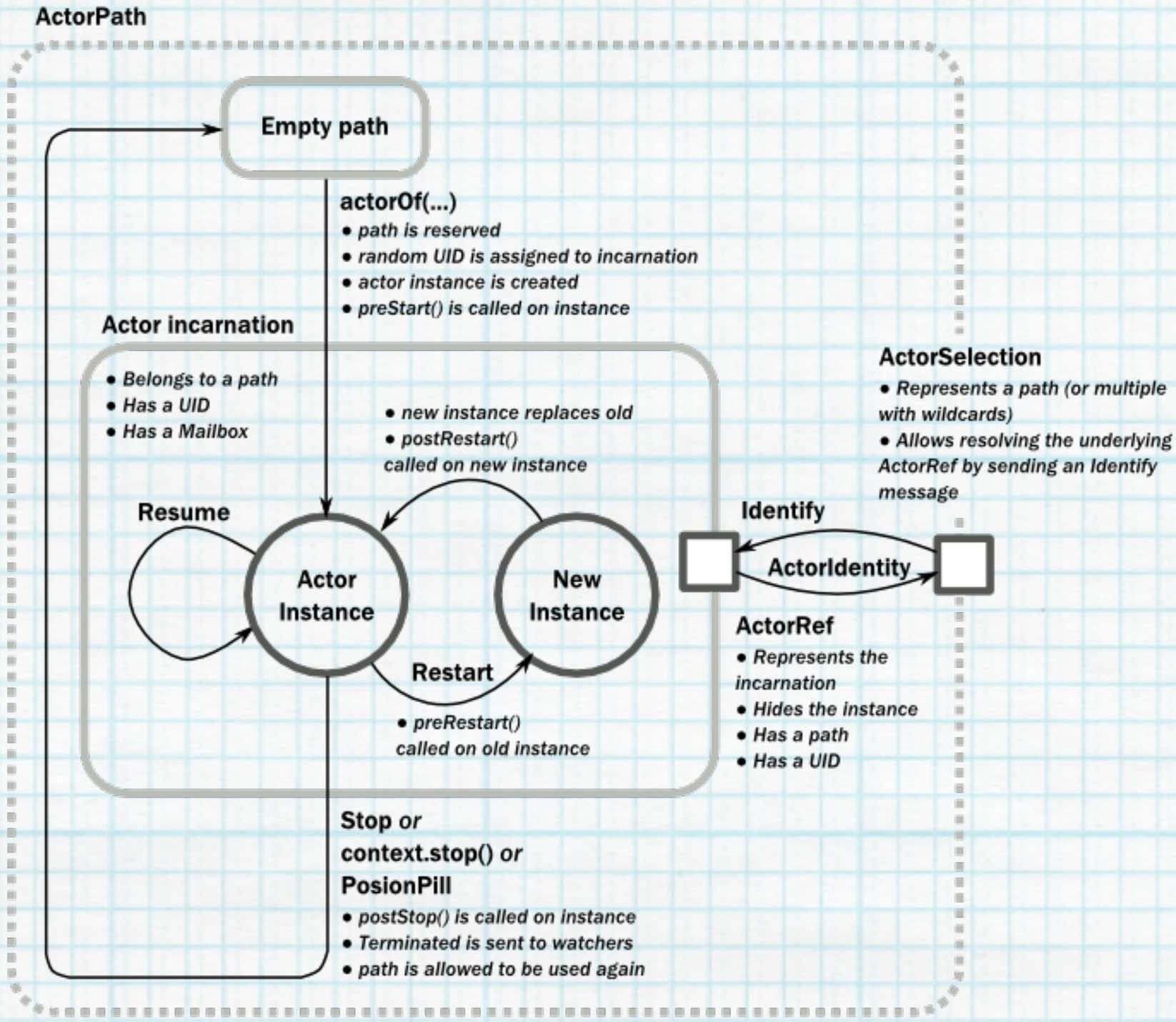


addressing/selection

```
context.actorSelection(“/Bar/E”)
```

create

Actors à la Akka



send

akka

Actors à la Akka

```
context.actorSelection(“/Foo/A”).send(msg)
```

Message Delivery Reliability

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

DEAL WITH IT.

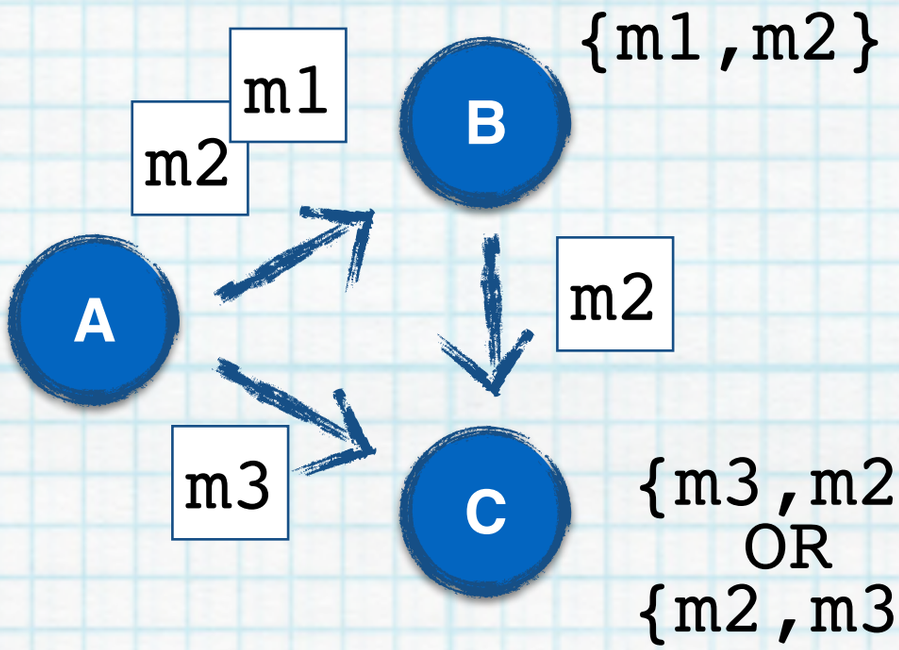
send

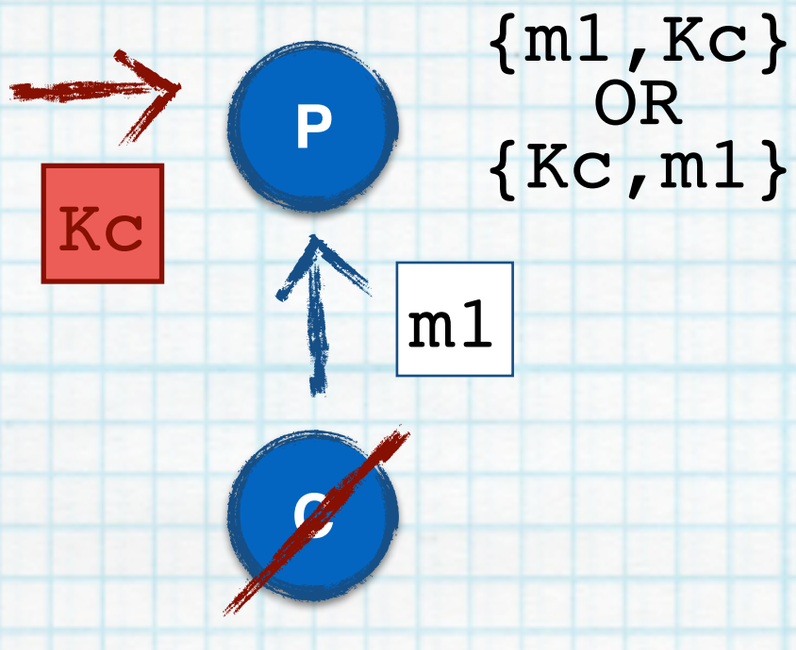
Actors à la Akka

```
context.actorSelection(“/Foo/A”).send(msg)
```

Message Delivery Reliability

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





become

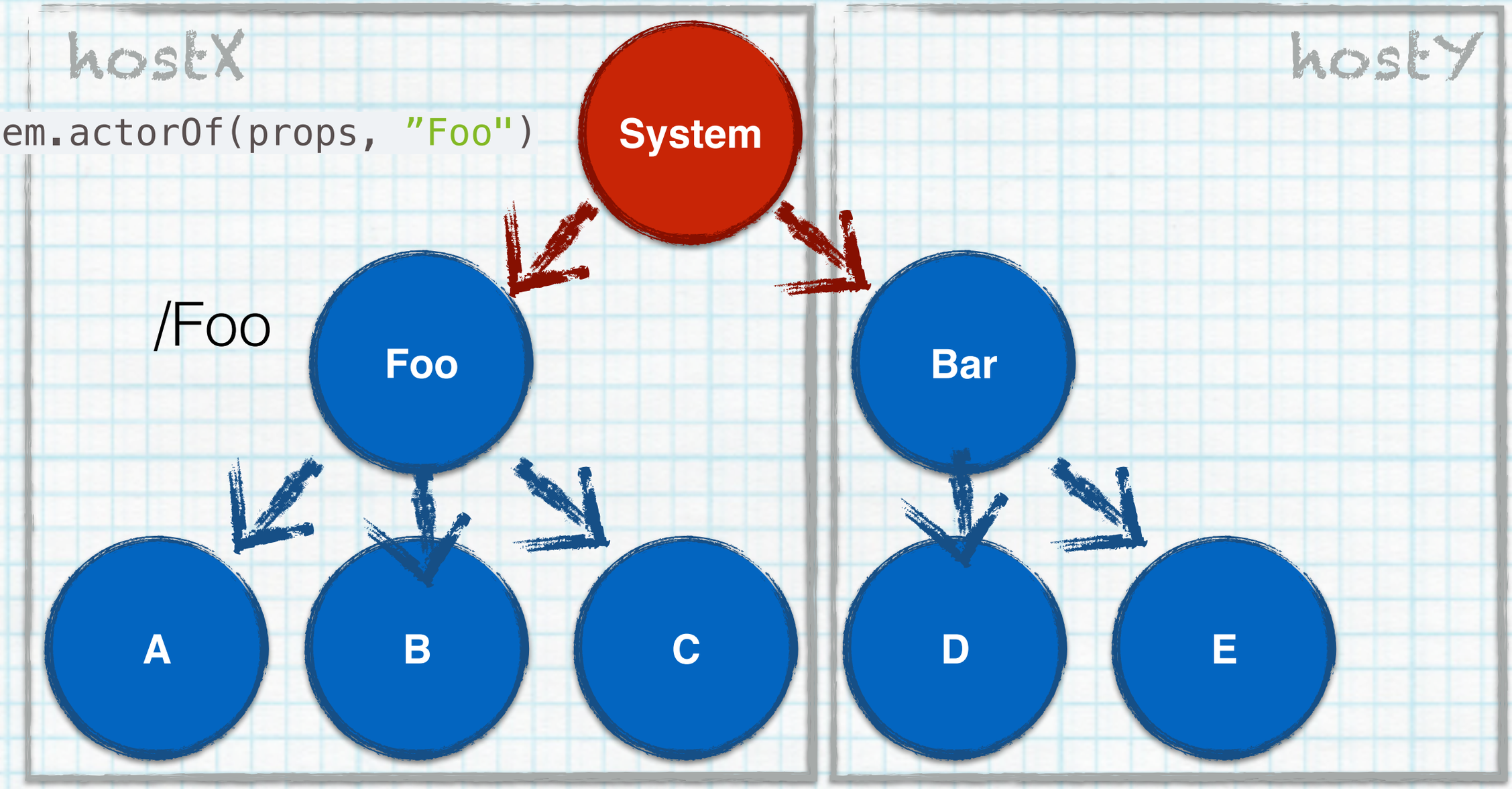
Actors à la Akka

become.java — Desktop

```java
public class HotSwapActor extends UntypedActor {

  Procedure<Object> angry = new Procedure<Object>() {
    @Override
    public void apply(Object message) {
      if (message.equals("bar")) {
        getSender().tell("I am already angry?", getSelf());
      } else if (message.equals("foo")) {
        getContext().become(happy);
      }
    }
  };

  Procedure<Object> happy = new Procedure<Object>() {
    @Override
    public void apply(Object message) {
      if (message.equals("bar")) {
        getSender().tell("I am already happy :-)", getSelf());
      } else if (message.equals("foo")) {
        getContext().become(angry);
      }
    }
  };

  public void onReceive(Object message) {
    if (message.equals("bar")) {
      getContext().become(angry);
    } else if (message.equals("foo")) {
      getContext().become(happy);
    } else {
      unhandled(message);
    }
  }
```

create remoting

Actors à la Akka



addressing/selection

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

remoting

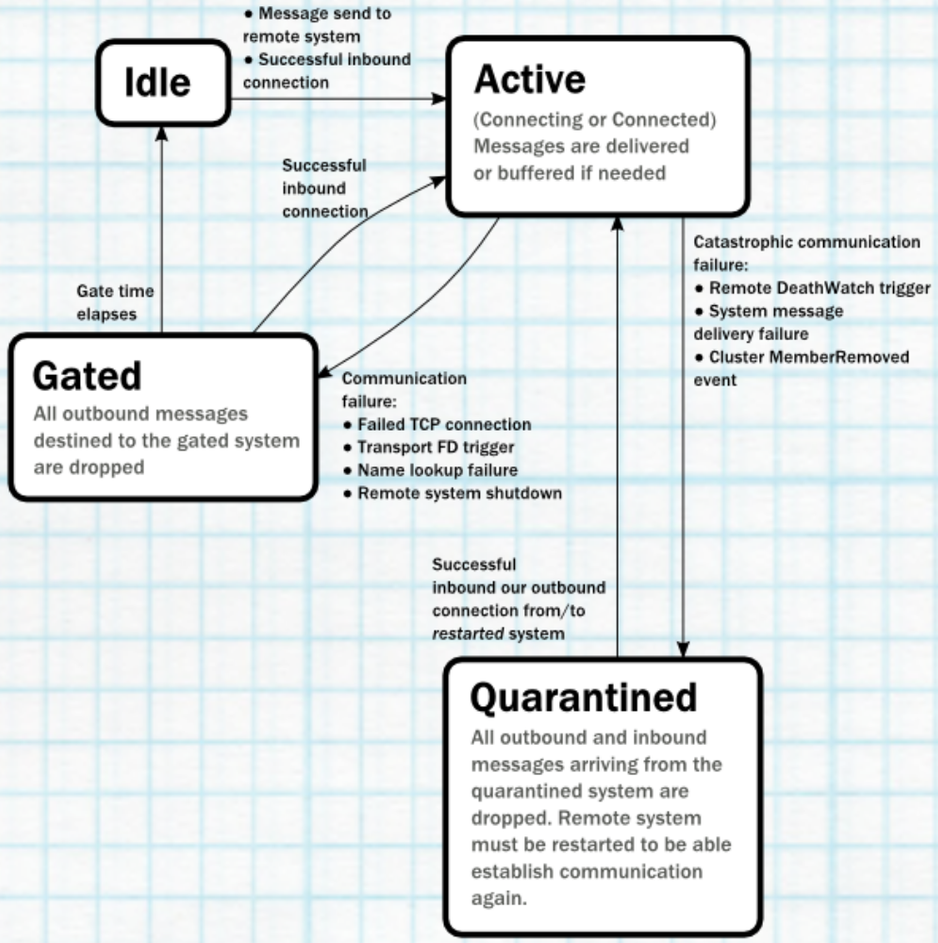
Actors à la Akka

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

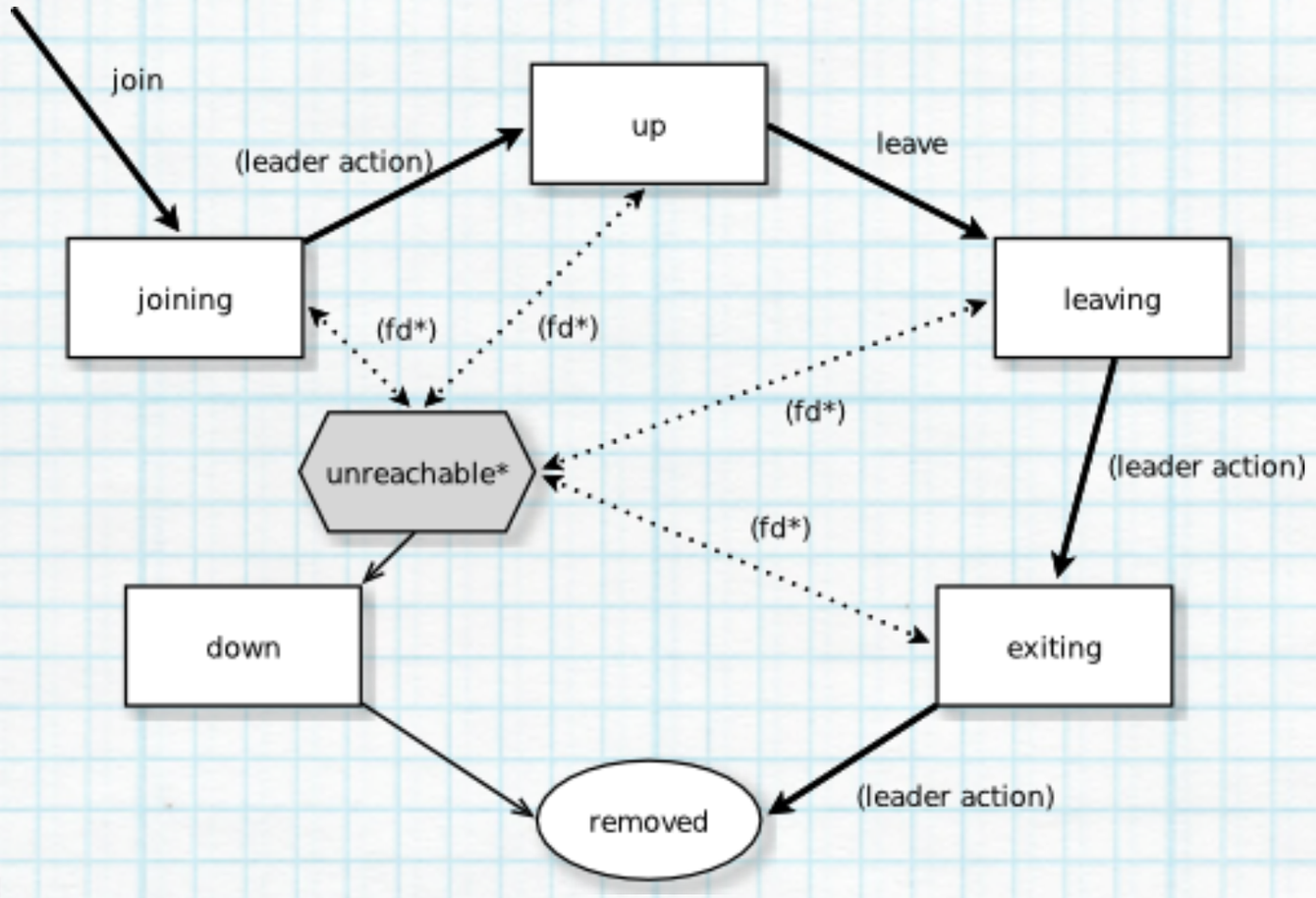
à la Dynamo, Riak

Gossip Protocol

for membership,
leader determination,
configuration

Vector Clocks

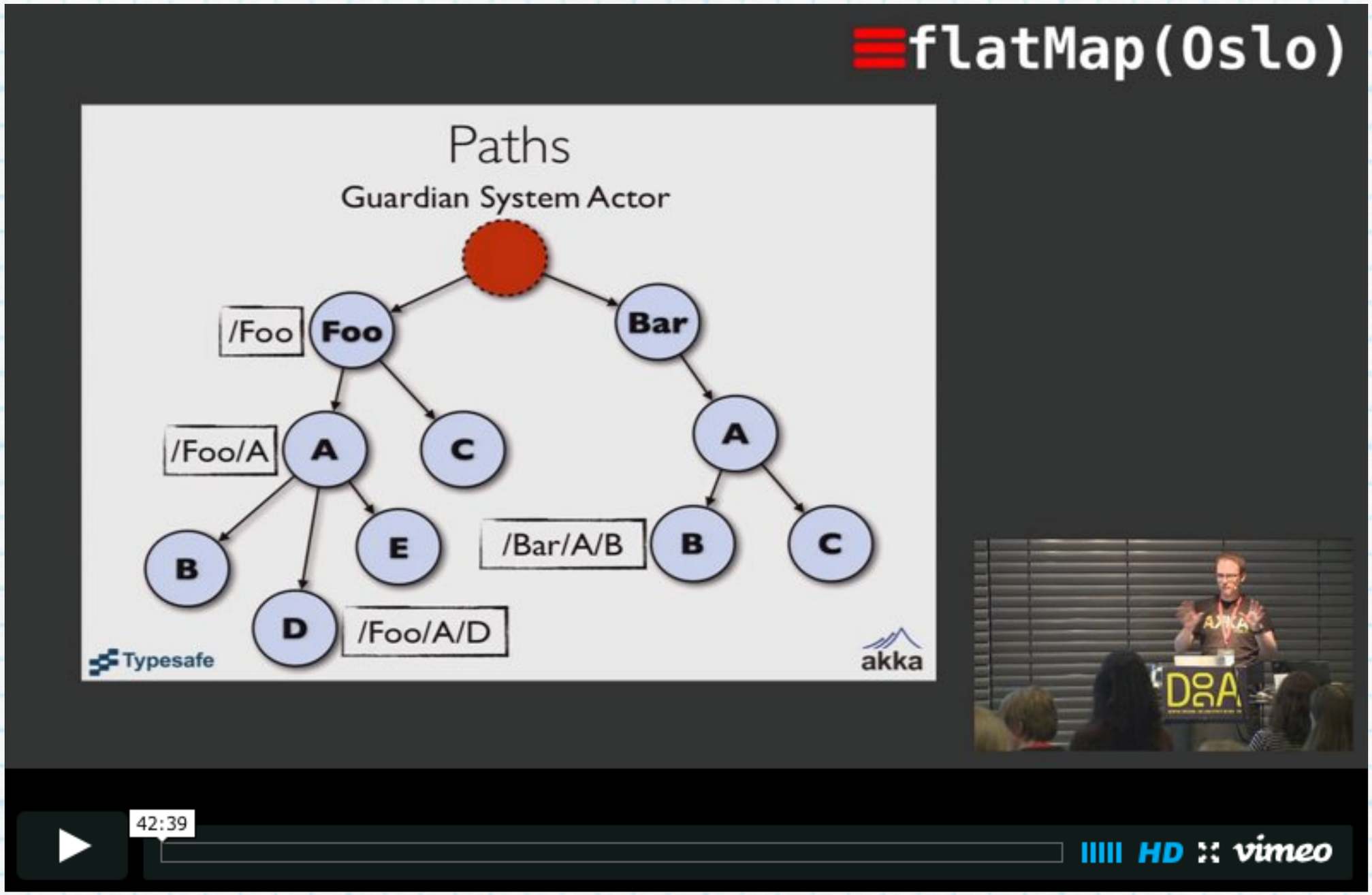
Leaders are ***not*** elected



Actors à la Akka

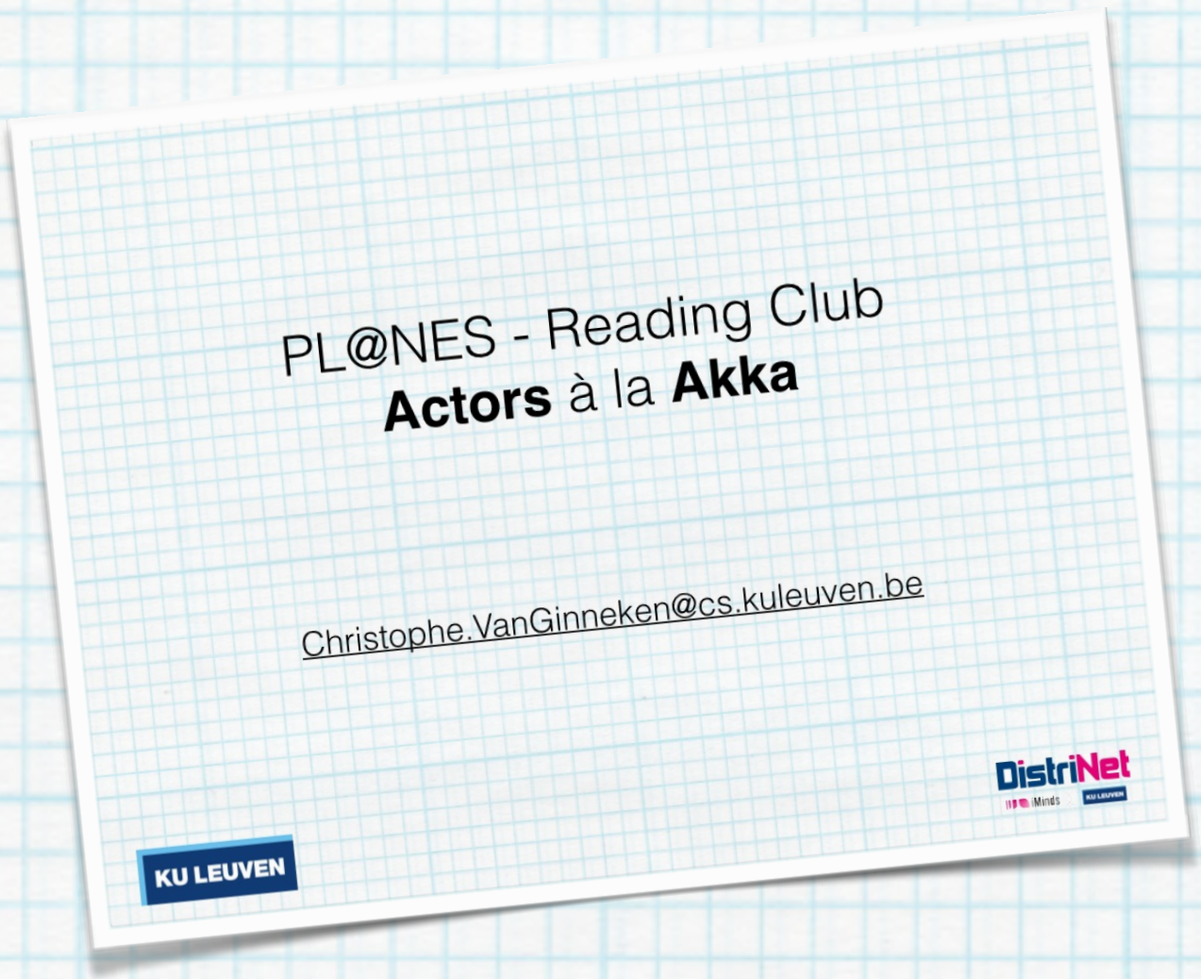
- Create, send, become
- Parents handle failures
- Purely reactive components
- Remoting with basic guarantees
- Clustering

Actors à la Akka



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


The first was for himself. The second was for his country.
This time is for his friend.
STALLONE
?
III
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 A TRI-STAR RELEASE
CAROLCO
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.



Christophe.VanGinneken@cs.kuleuven.be

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