Platinum Sponsor

LUXOFT
Engineer Your Future

REACTIVE PROGRAMMING WITH AKKA

- LESSONS LEARNED -

Daniel Deogun & Daniel Sawano
Twitter: @DanielDeogun, @DanielSawano

33rd DEGREE

Who We Are

Daniel Deogun

Daniel Sawano

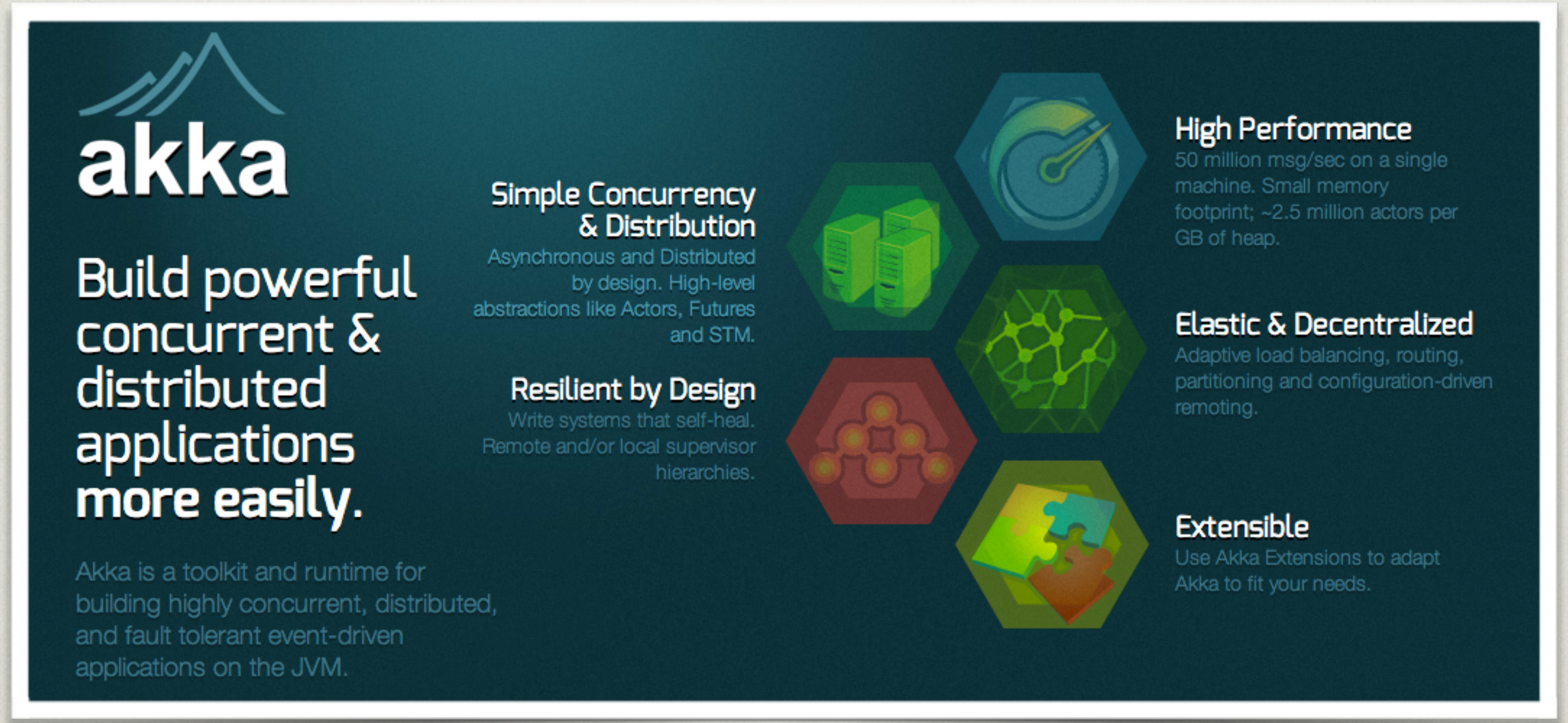
Omegapoint

Stockholm - Gothenburg - Malmoe - Umea - New York

Agenda

- Akka in a nutshell
- Akka & Java
- Akka and Java 8 Lambdas
- Domain influences
- Lessons learned from building real systems with Akka

Akka in a Nutshell



akka

Build powerful concurrent & distributed applications **more easily.**

Akka is a toolkit and runtime for building highly concurrent, distributed, and fault tolerant event-driven applications on the JVM.

Simple Concurrency & Distribution
Asynchronous and Distributed by design. High-level abstractions like Actors, Futures and STM.

Resilient by Design
Write systems that self-heal. Remote and/or local supervisor hierarchies.

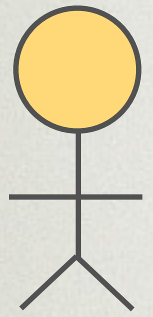
High Performance
50 million msg/sec on a single machine. Small memory footprint; ~2.5 million actors per GB of heap.

Elastic & Decentralized
Adaptive load balancing, routing, partitioning and configuration-driven remoting.

Extensible
Use Akka Extensions to adapt Akka to fit your needs.

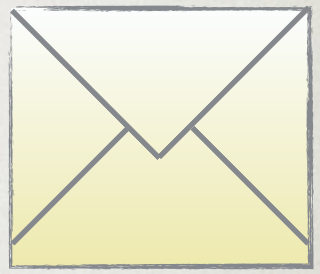
http://akka.io/

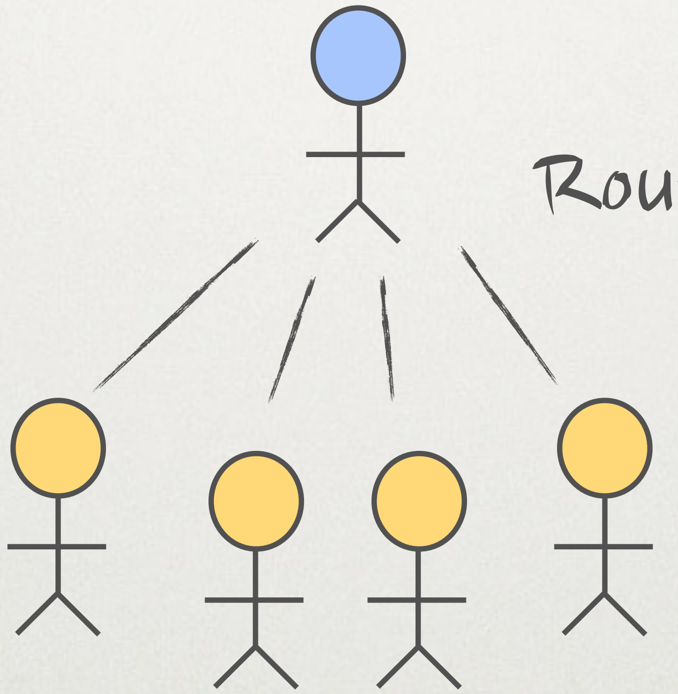
Akka in a Nutshell



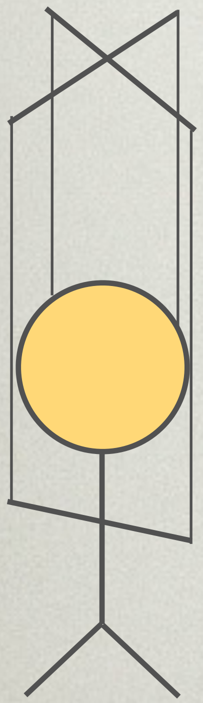
Actors

Messages





Routers



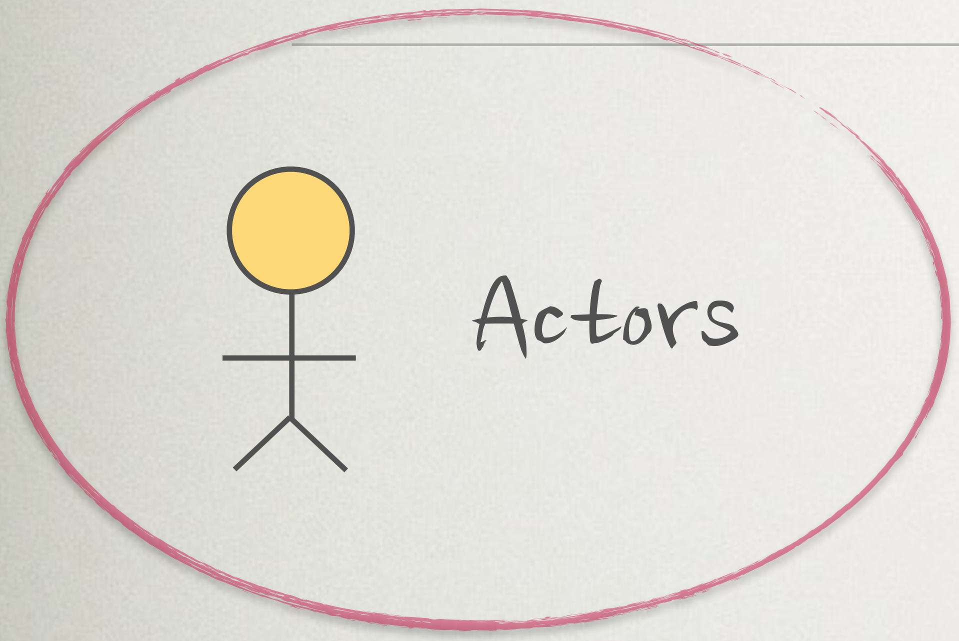
Actor System

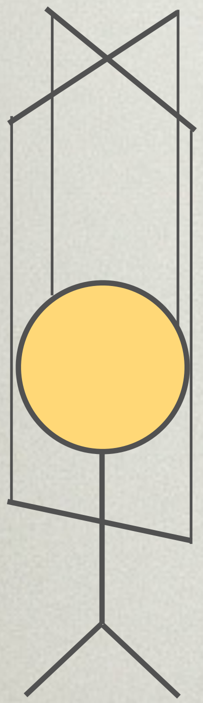
Mailbox

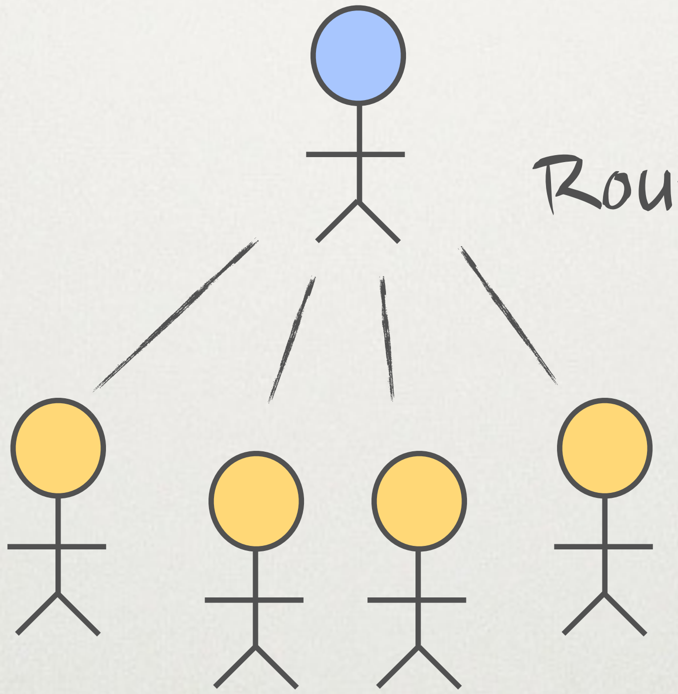
[1]

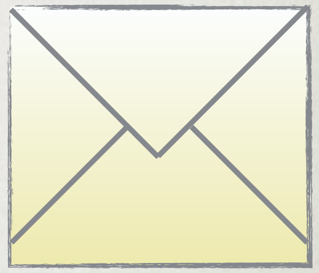
[1] By Dickelbers (Own work) [CC-BY-SA-3.0 (http://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons
http://upload.wikimedia.org/wikipedia/commons/0/09/Sweden_postbox.JPG

Akka in a Nutshell







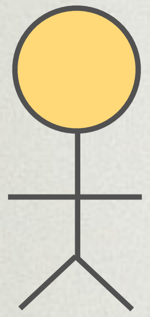


[1]

[1] By Dickelbers (Own work) [CC-BY-SA-3.0 (http://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons
http://upload.wikimedia.org/wikipedia/commons/0/09/Sweden_postbox.JPG

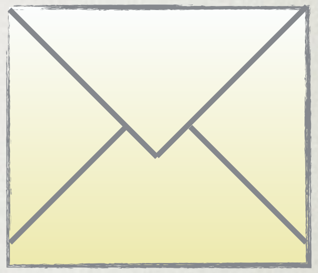
omega point.

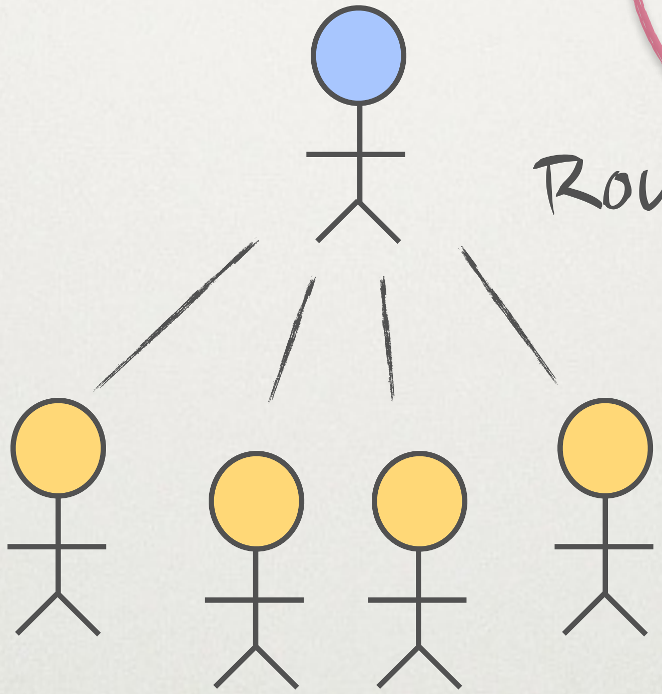
Akka in a Nutshell



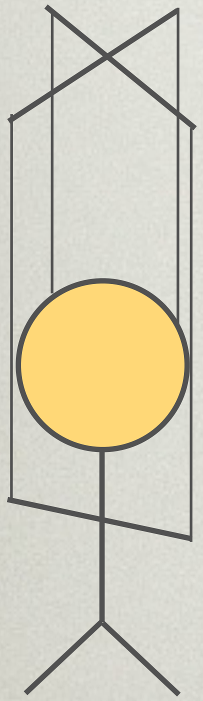
Actors

Messages





Routers



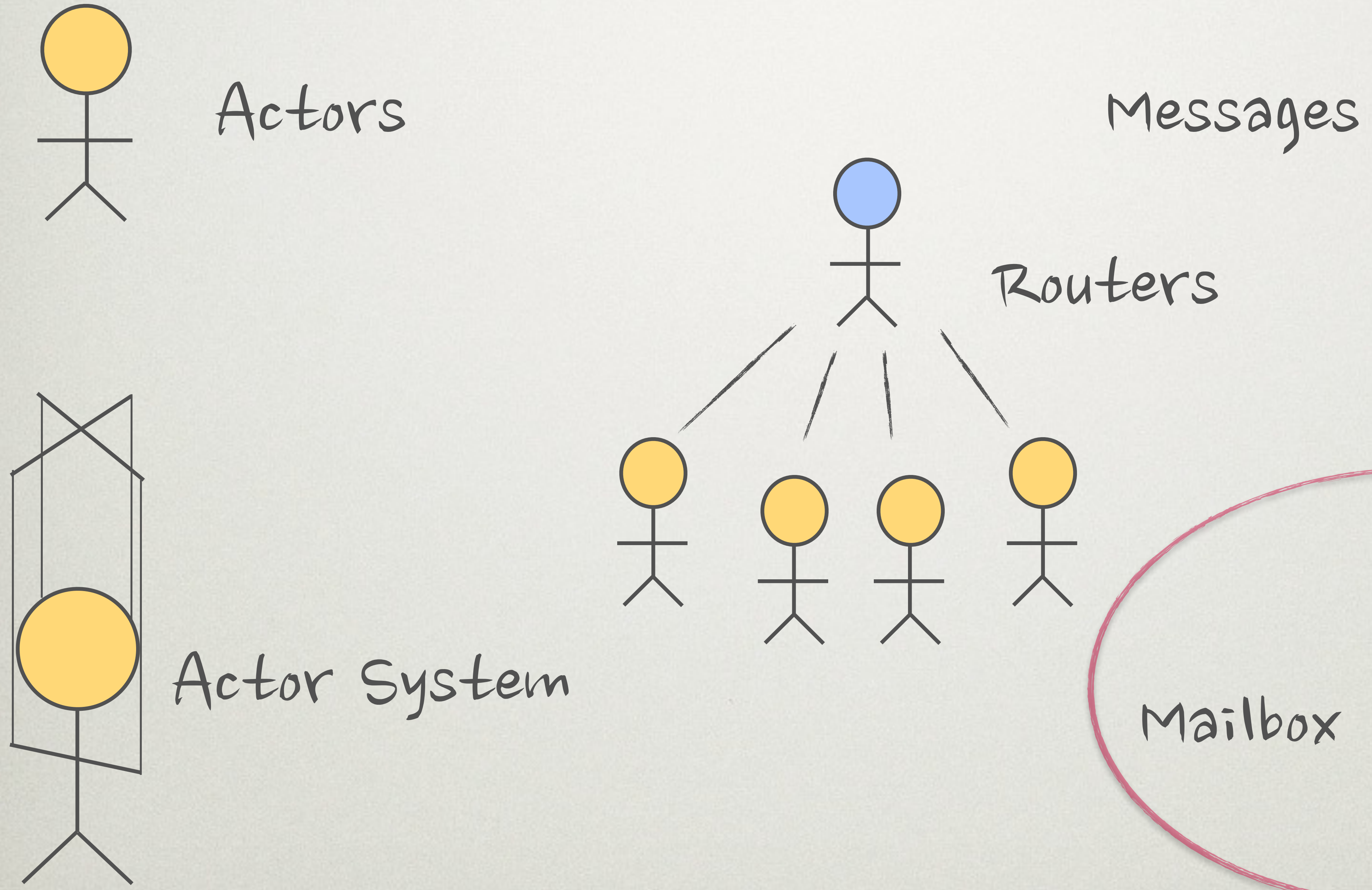
Actor System

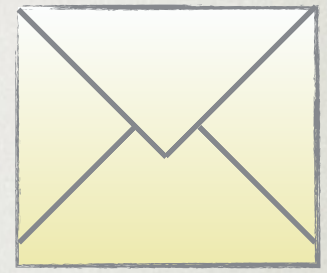
Mailbox

[1]

[1] By Dickelbers (Own work) [CC-BY-SA-3.0 (http://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons
http://upload.wikimedia.org/wikipedia/commons/0/09/Sweden_postbox.JPG

Akka in a Nutshell

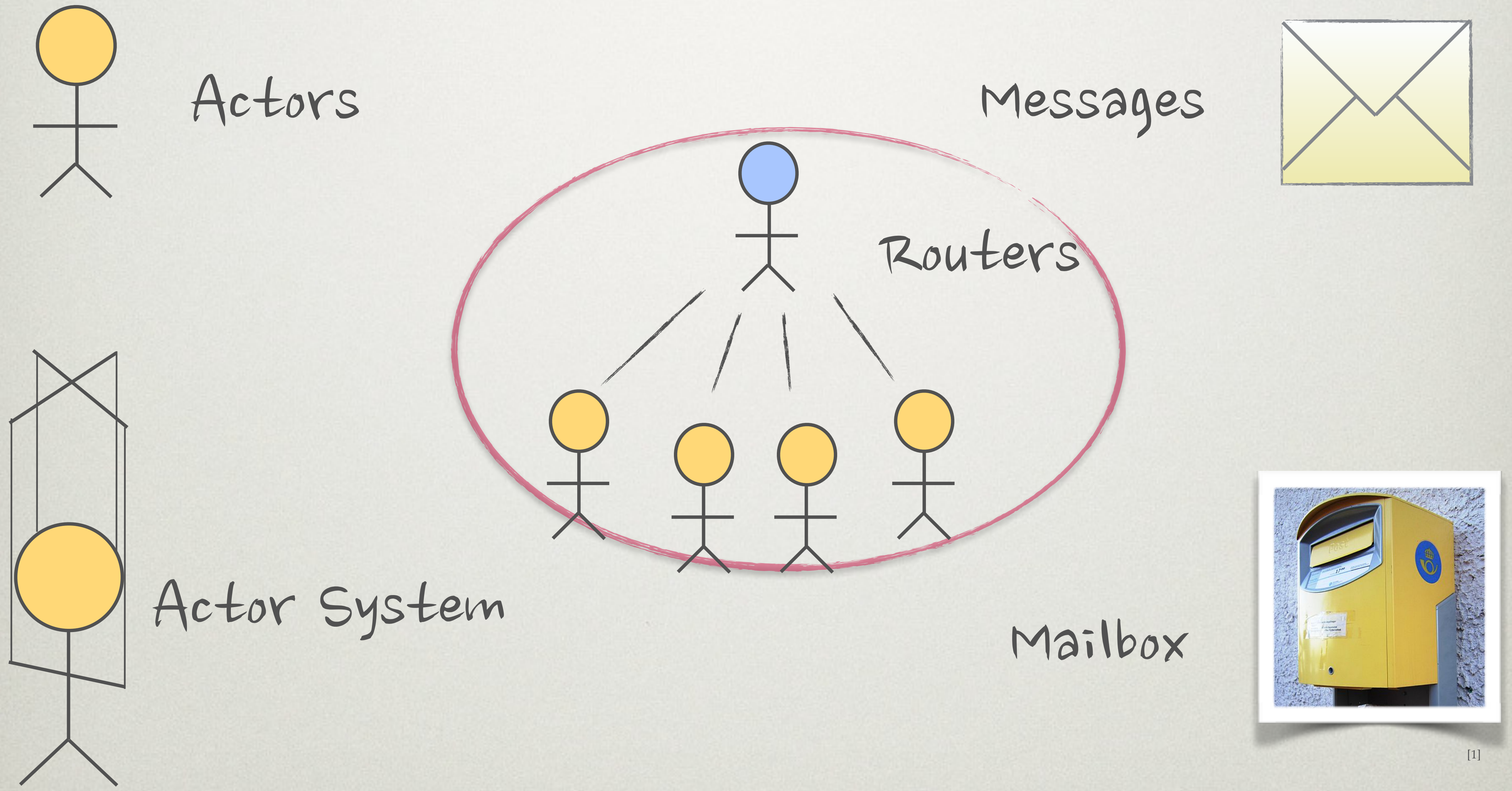




[1]

[1] By Dickelbers (Own work) [CC-BY-SA-3.0 (http://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons
http://upload.wikimedia.org/wikipedia/commons/0/09/Sweden_postbox.JPG

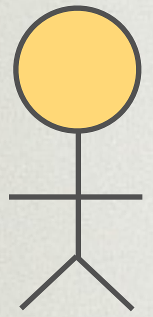
Akka in a Nutshell



[1]

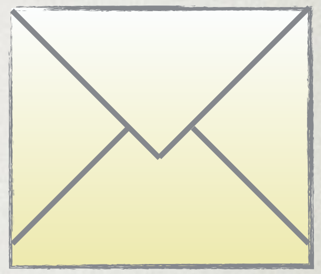
[1] By Dickelbers (Own work) [CC-BY-SA-3.0 (http://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons
http://upload.wikimedia.org/wikipedia/commons/0/09/Sweden_postbox.JPG

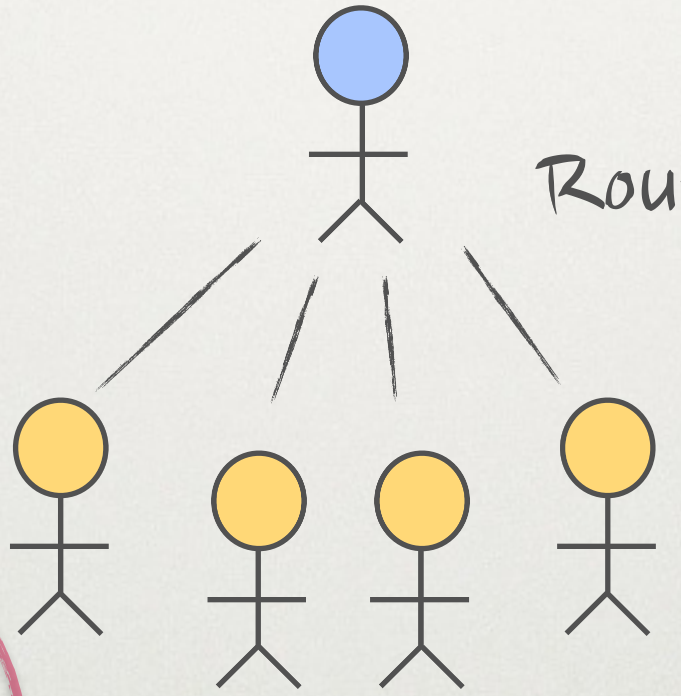
Akka in a Nutshell



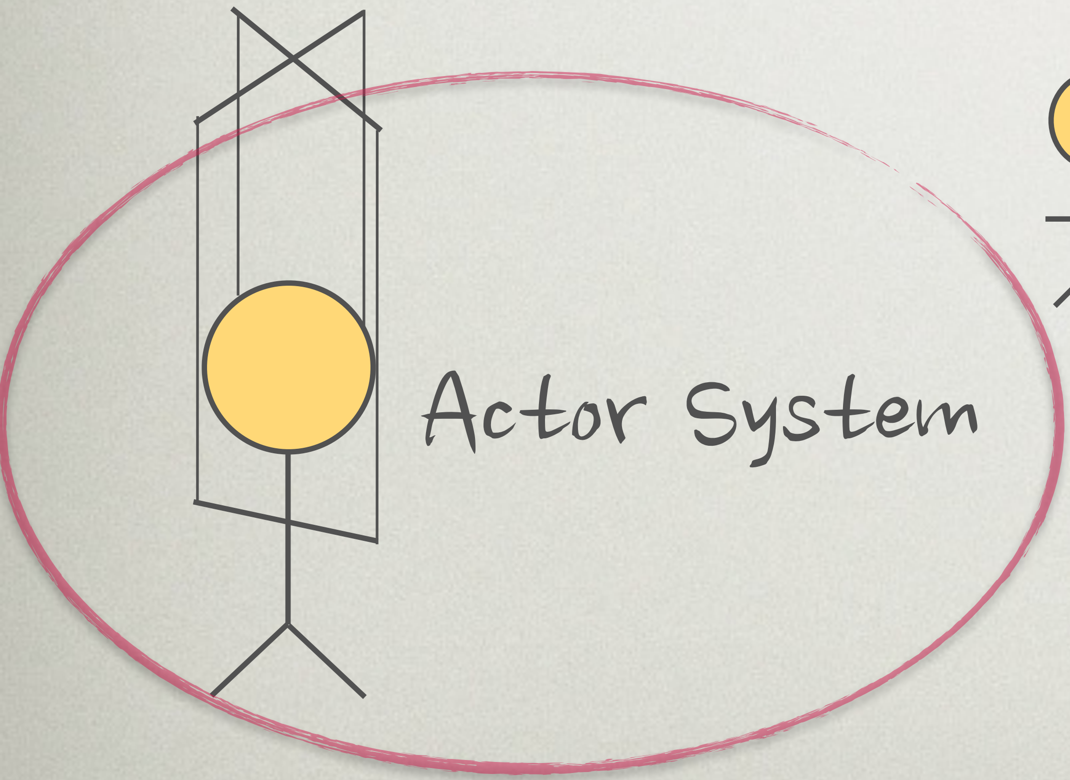
Actors

Messages





Routers



Actor System

Mailbox

[1]

[1] By Dickelbers (Own work) [CC-BY-SA-3.0 (http://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons
http://upload.wikimedia.org/wikipedia/commons/0/09/Sweden_postbox.JPG

Our definition of Legacy Code

Legacy \ˈle-gə-sē\

": something that happened in the past or that comes from someone in the past"

- Merriam-Webster

omega point.

Our definition of Legacy Code

Legacy \ˈle-gə-sē\

": something that happened in the past or that comes from someone in the past"

- Merriam-Webster

Legacy Code \ˈle-gə-sē\ \ˈkōd\

": code that does not satisfy the characteristics of a reactive system"

- Deogun-Sawano

omega point.

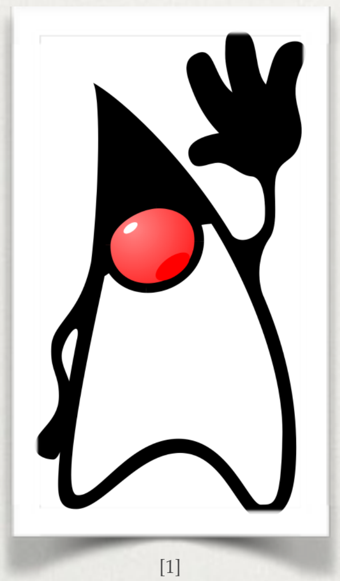
What is Legacy Code?

Characteristics of a reactive system, as defined by the reactive manifesto:

- responsive
- scalable
- resilient
- event-driven

reactivemanifesto.org

Java or Scala


[1]

[2]

I want to build an application with Akka, should I use Java or Scala?

Well, it depends...

[1] https://duke.kenai.com/wave/.Midsize/Wave.png.png
[2] http://www.scala-lang.org/

omega point.

Java or Scala

Assume we want to build a machine M to solve a problem P where,

- Efficiency is imperative
- Sequential computations shall be independent
- Implementation of M shall be platform independent
- Complexity and boilerplate code shall be reduced
- M's behavior shall be verifiable
- Time to Market is essential and risks minimized

Java or Scala Pros & Cons

Scoreboard

Java:

Scala:

Assume we want to build a machine M to solve a problem P where,

- ☐ Efficiency is imperative
- ☐ Sequential computations shall be independent
- ☐ Implementation of M shall be platform independent
- ☐ Complexity and boilerplate code shall be reduced
- ☐ M's behavior shall be verifiable
- ☐ Time to Market is essential and risks shall be minimized

omega point.

Java or Scala Pros & Cons

Scoreboard

Java: 1

Scala: 1

Assume we want to build a machine M to solve a problem P where,

- [x] Efficiency is imperative
- [ ] Sequential computations shall be independent
- [ ] Implementation of M shall be platform independent
- [ ] Complexity and boilerplate code shall be reduced
- [ ] M's behavior shall be verifiable
- [ ] Time to Market is essential and risks shall be minimized

Java or Scala Pros & Cons

Assume we want to build a machine M to solve a problem P where,

- [✗] Efficiency is imperative
- [✗] Sequential computations shall be independent
- [ ] Implementation of M shall be platform independent
- [ ] Complexity and boilerplate code shall be reduced
- [ ] M's behavior shall be verifiable
- [ ] Time to Market is essential and risks shall be minimized

Scoreboard

Java: ll

Scala: ll

omega point.

Java or Scala Pros & Cons

Assume we want to build a machine M to solve a problem P where,

- [x] Efficiency is imperative
- [x] Sequential computations shall be independent
- [x] Implementation of M shall be platform independent
- [ ] Complexity and boilerplate code shall be reduced
- [ ] M's behavior shall be verifiable
- [ ] Time to Market is essential and risks shall be minimized

Scoreboard

Java: III

Scala: III

omega point.

Java or Scala
Pros & Cons

Assume we want to build a machine M to solve a problem P where,

- [x] Efficiency is imperative
- [x] Sequential computations shall be independent
- [x] Implementation of M shall be platform independent
- [x] Complexity and boilerplate code shall be reduced
- [ ] M's behavior shall be verifiable
- [ ] Time to Market is essential and risks shall be minimized

Scoreboard

Java: III

Scala: IIII

Java or Scala Pros & Cons

Assume we want to build a machine M to solve a problem P where,

- ☒ Efficiency is imperative
- ☒ Sequential computations shall be independent
- ☒ Implementation of M shall be platform independent
- ☒ Complexity and boilerplate code shall be reduced
- ☒ M's behavior shall be verifiable
- ☐ Time to Market is essential and risks shall be minimized

Scoreboard

Java: IIII

Scala: ~~IIII~~

Java or Scala Pros & Cons

Assume we want to build a machine M to solve a problem P where,

- ☒ Efficiency is imperative
- ☒ Sequential computations shall be independent
- ☒ Implementation of M shall be platform independent
- ☒ Complexity and boilerplate code shall be reduced
- ☒ M's behavior shall be verifiable
- ☒ Time to Market is essential and risks shall be minimized

Scoreboard

Java: 卌

Scala: 卌

omega point.

Java or Scala Conclusion

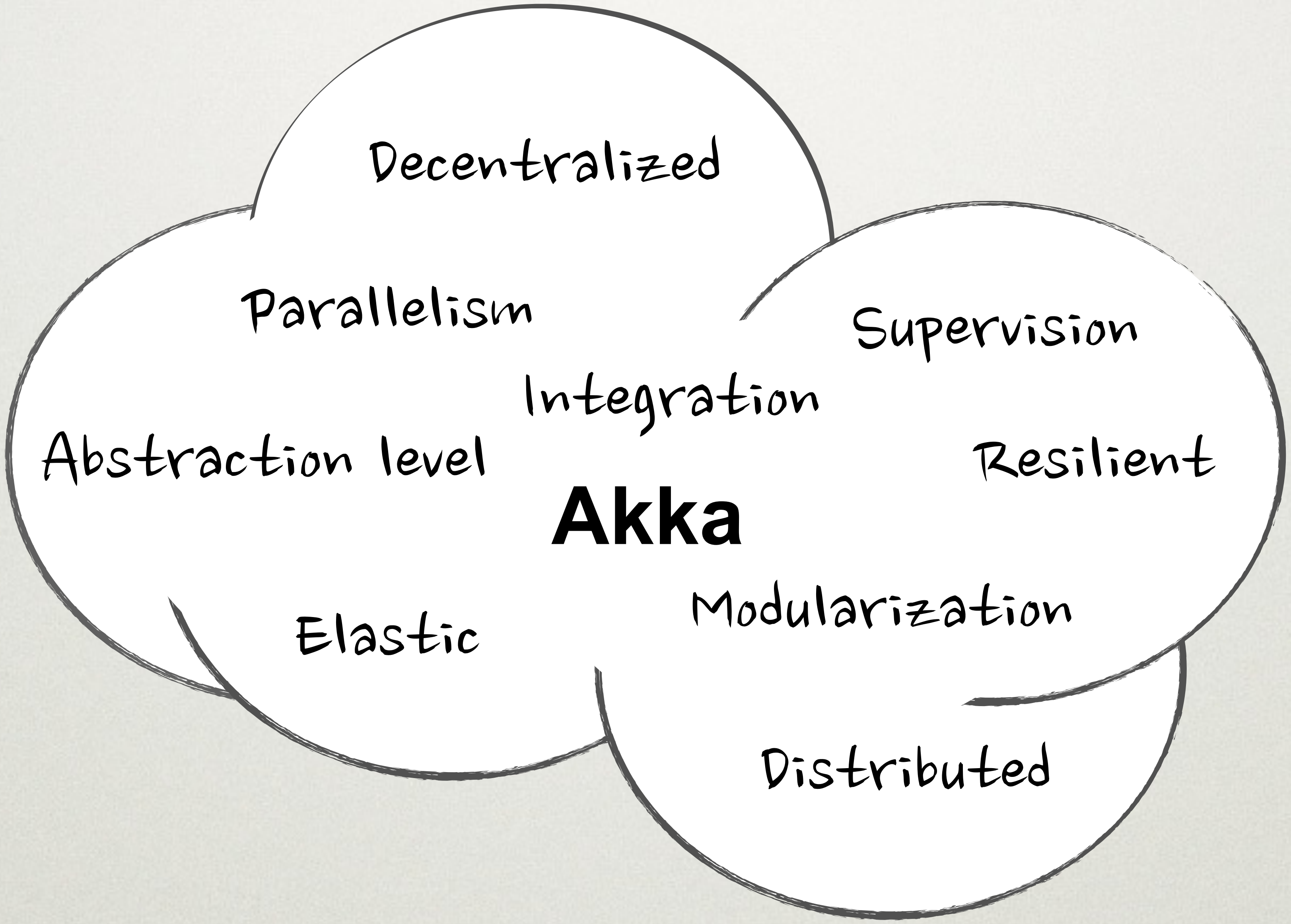
Scoreboard

Java: 𝍸

Scala: 𝍸

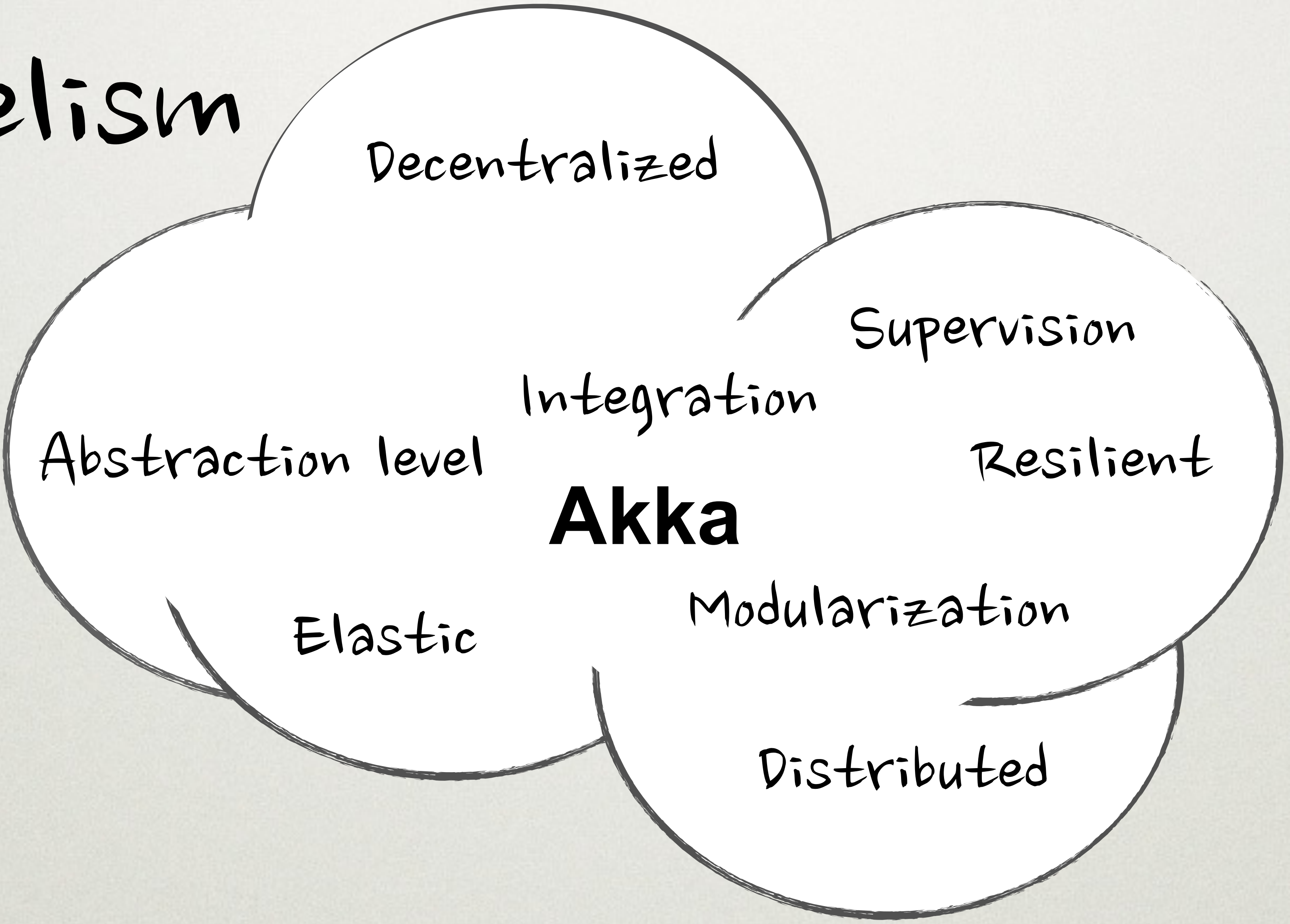
- Both Java and Scala works well with Akka
- Choose the language that makes most sense
- Don't add unnecessary risk

Akka
All or nothing?

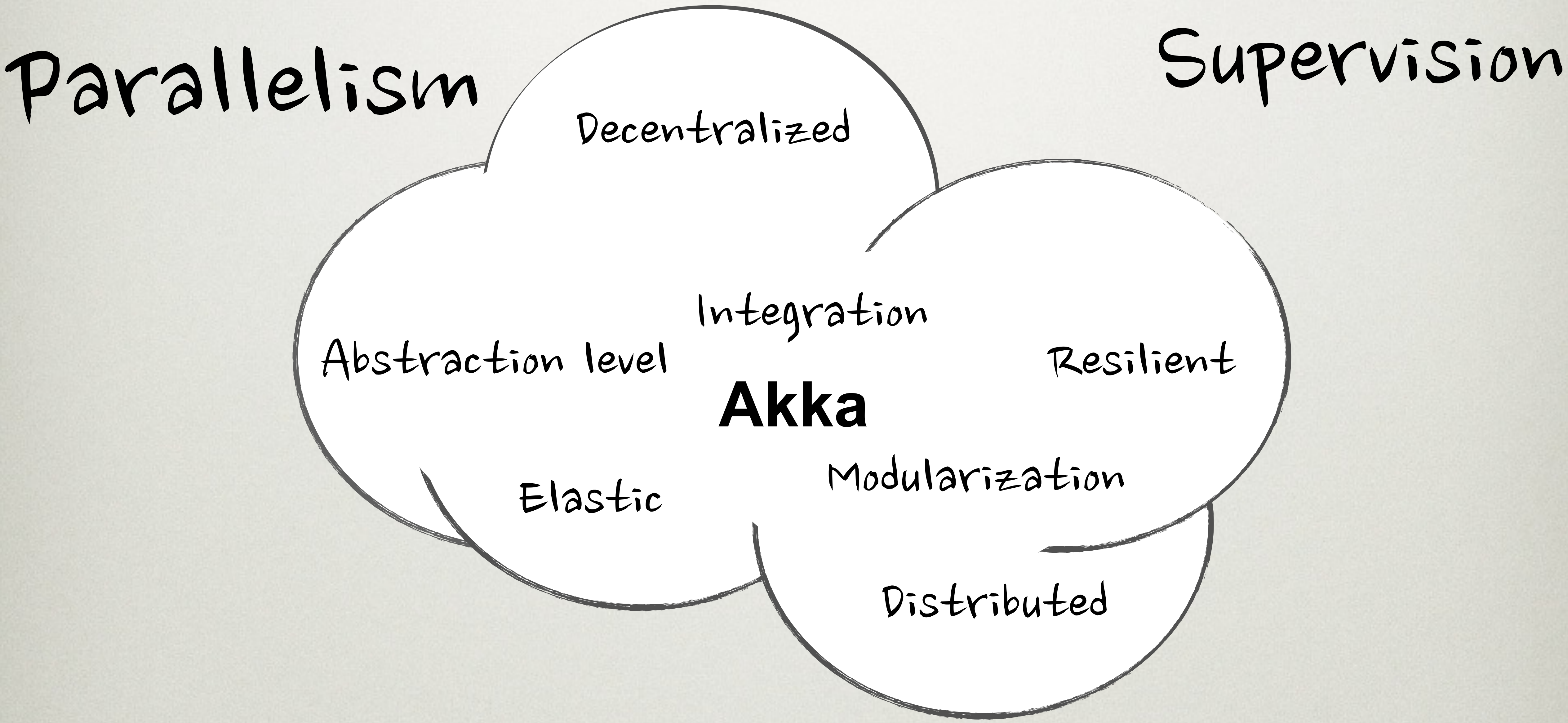


Akka
All or nothing?

Parallelism



Akka
All or nothing?

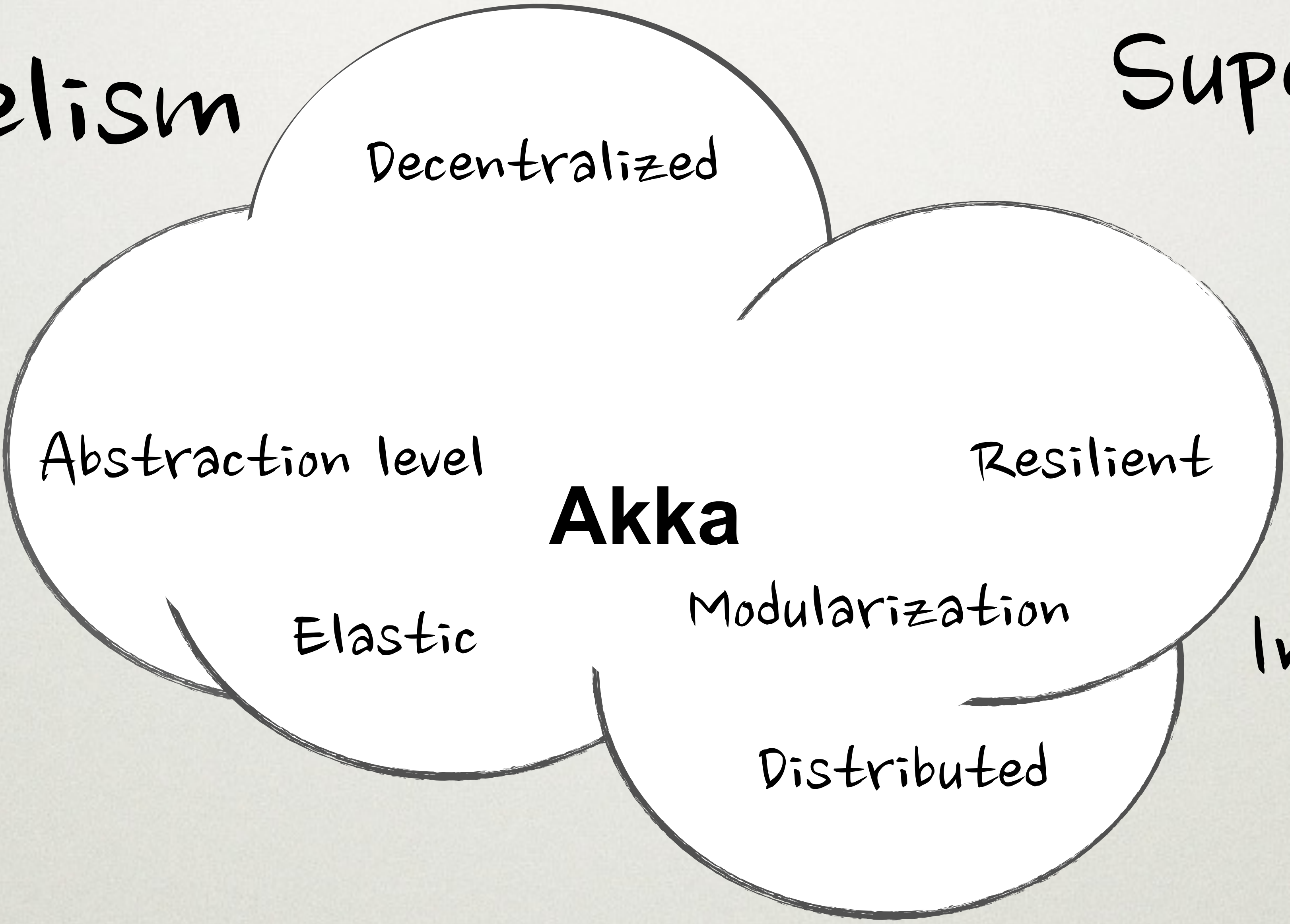


omega point.

Akka
All or nothing?

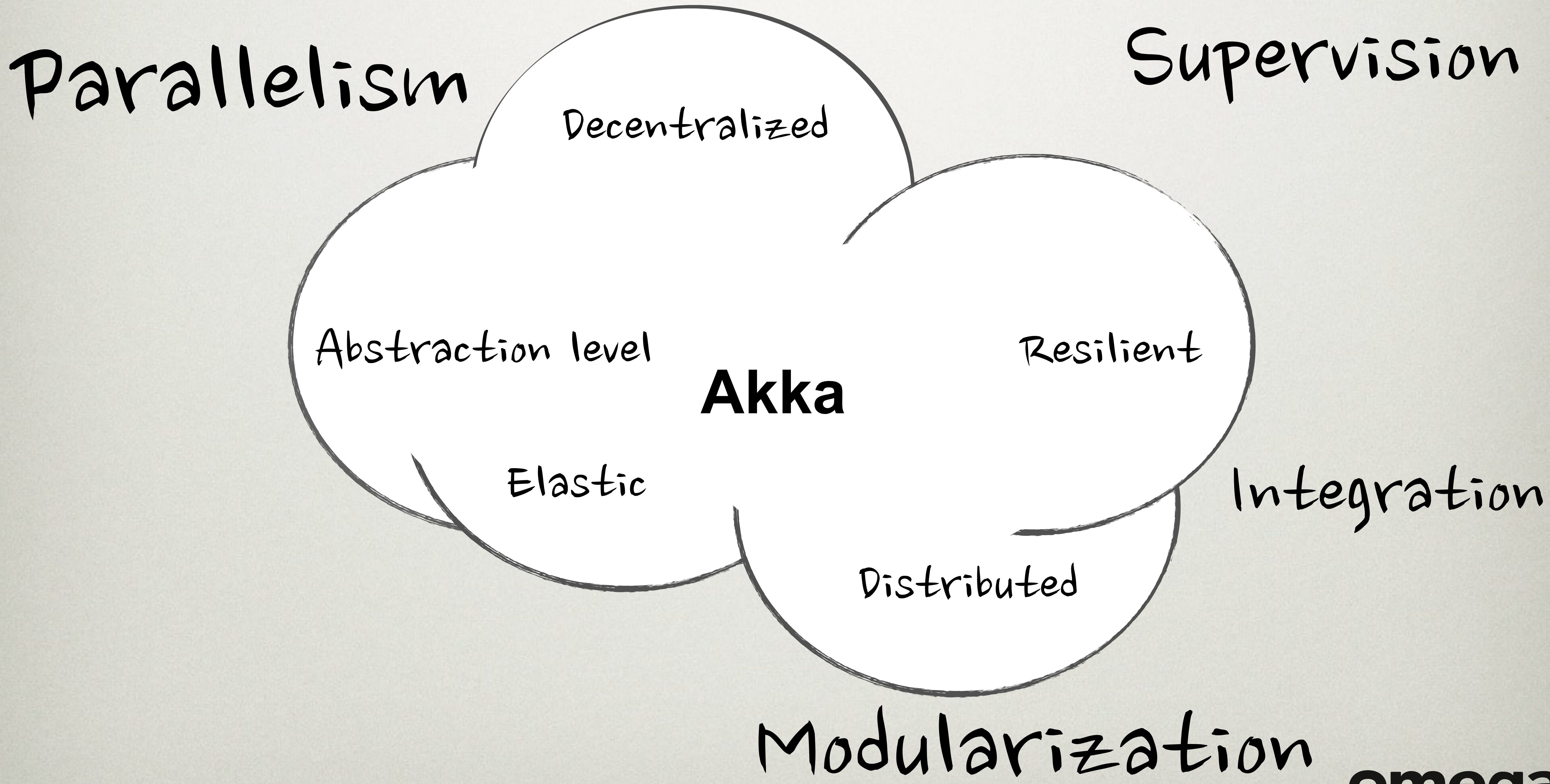
Parallelism

Supervision



Integration

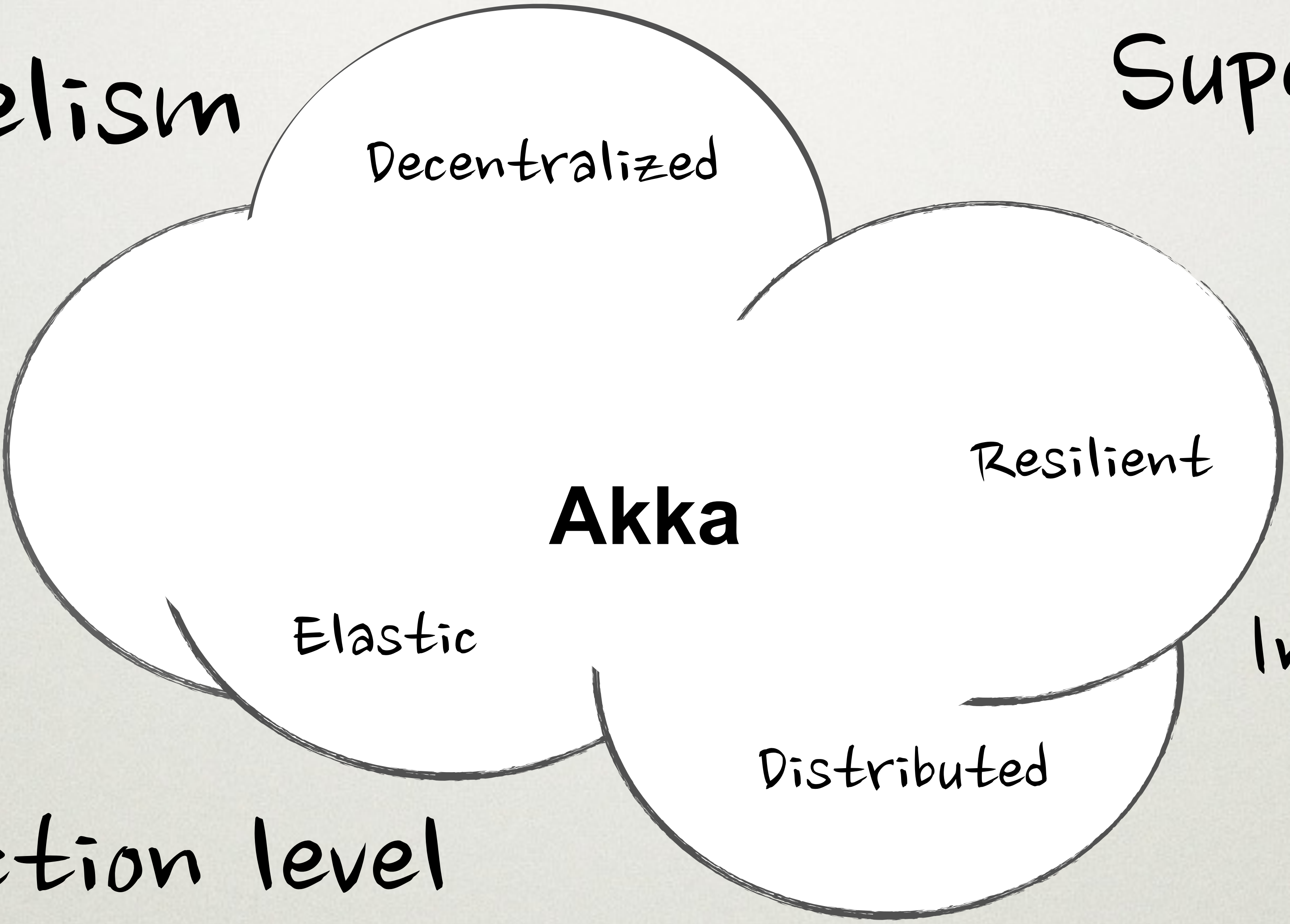
Akka
All or nothing?



Akka
All or nothing?

Parallelism

Supervision



Integration

Abstraction level

Modularization

omega point.

Domain Specific Requirements

- Akka is more or less a perfect match for all parallelizable domains

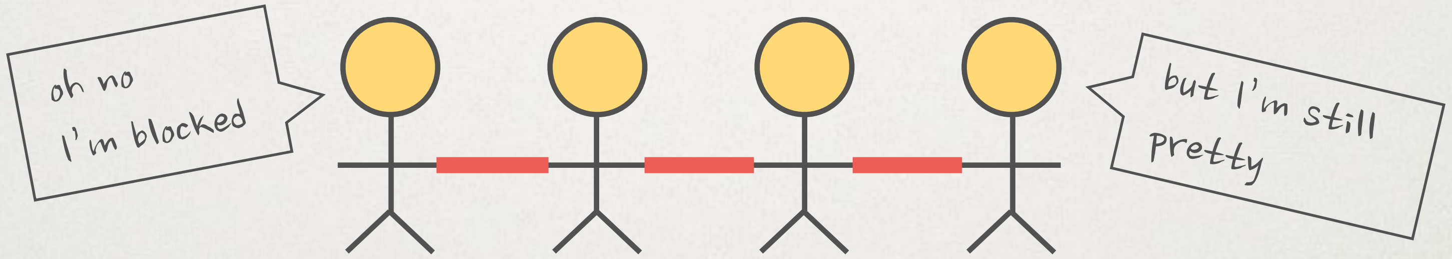
- But what about
  - reactive domains with blocking parts?
  - legacy domains with reactive parts?

[1]

[1] By Renesis (Own work) [CC-BY-SA-2.5 (http://creativecommons.org/licenses/by-sa/2.5)], via Wikimedia Commons. http://upload.wikimedia.org/wikipedia/commons/1/19/Stop2.png

Blocking in a Reactive Environment

- If we choose Akka, we need to block actors, yuck!

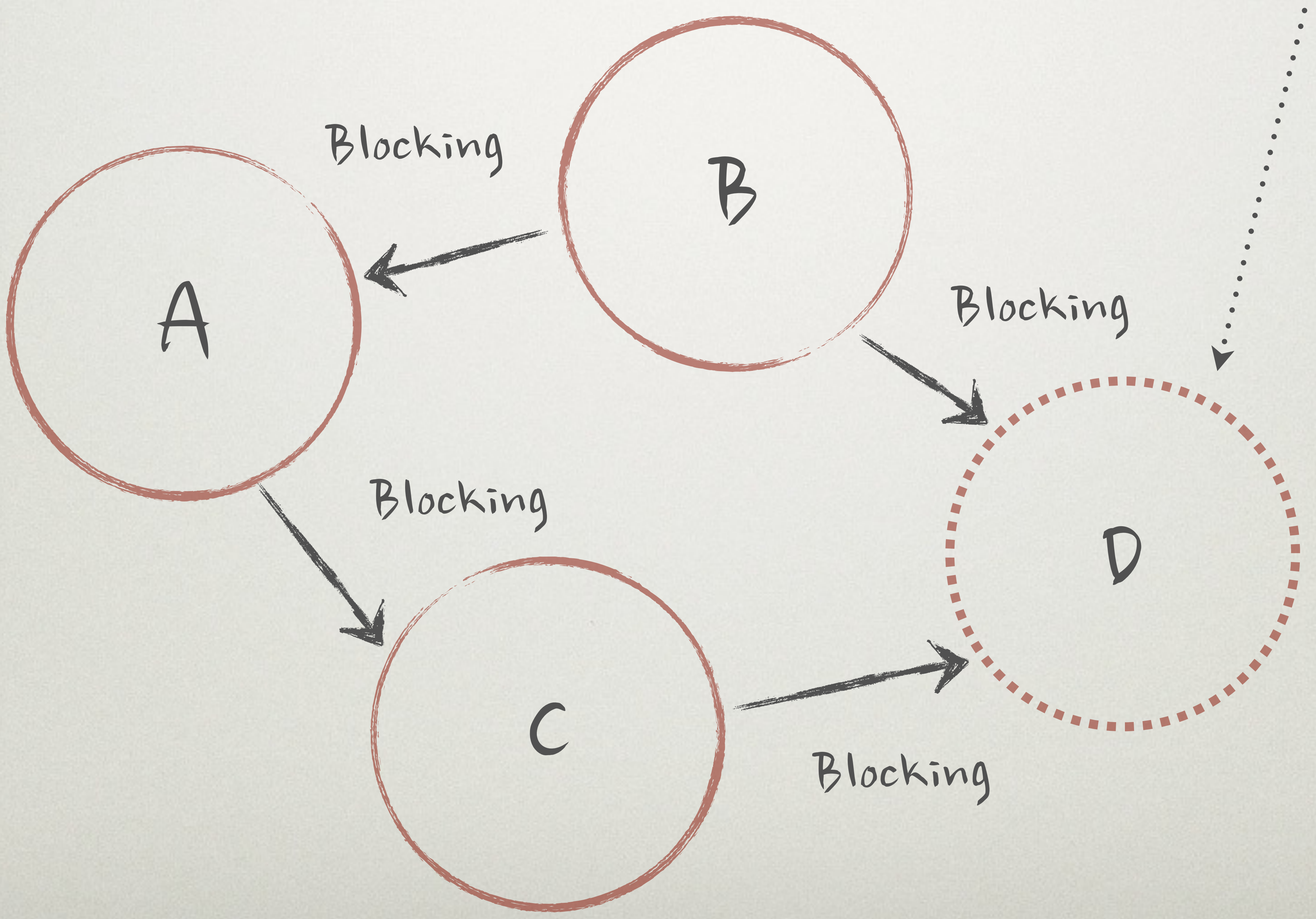


- The main advantage is that we can reuse actors from parallelizable parts but are there any downsides?

- The other option is to use legacy design

Reactive in a Legacy Environment?

Is it possible to replace D by a component implemented with Akka?

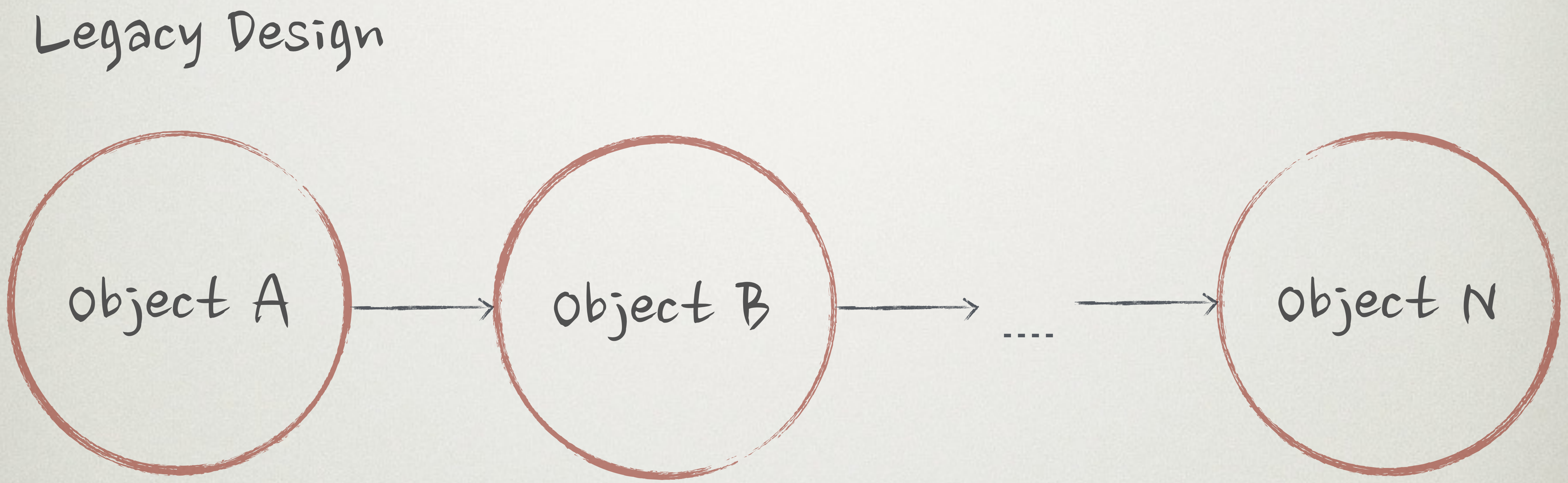


Debugger
Friend or Foe?

We often get the question:

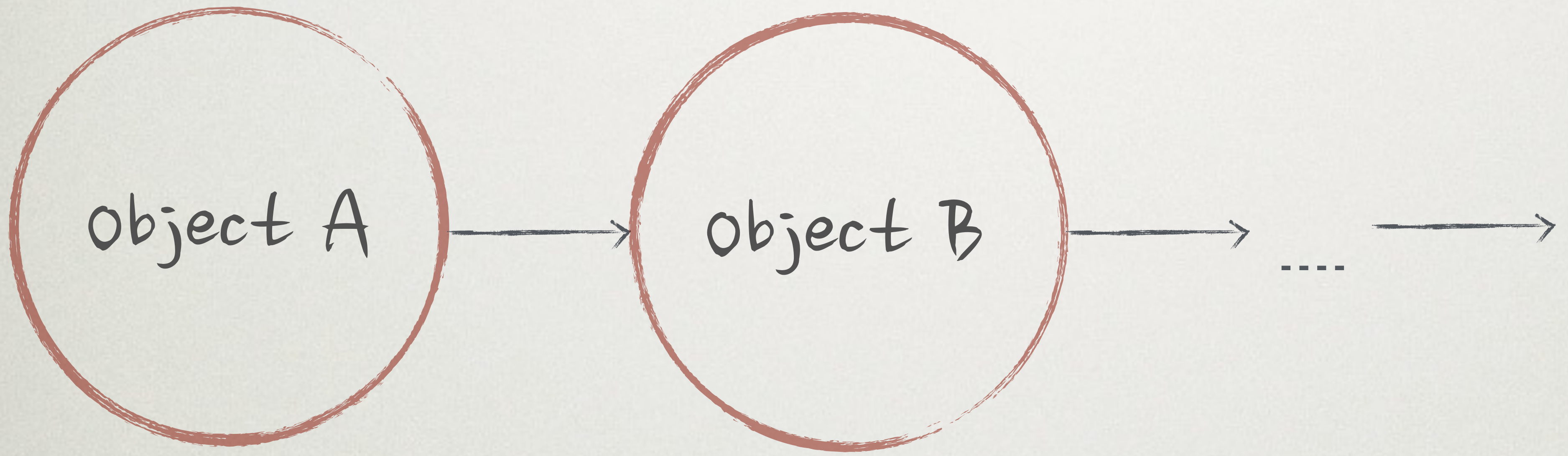
"The asynchrony in Akka makes it very hard to use the debugger, Am I doing it wrong?"

DEBUGGER IN LEGACY DESIGN



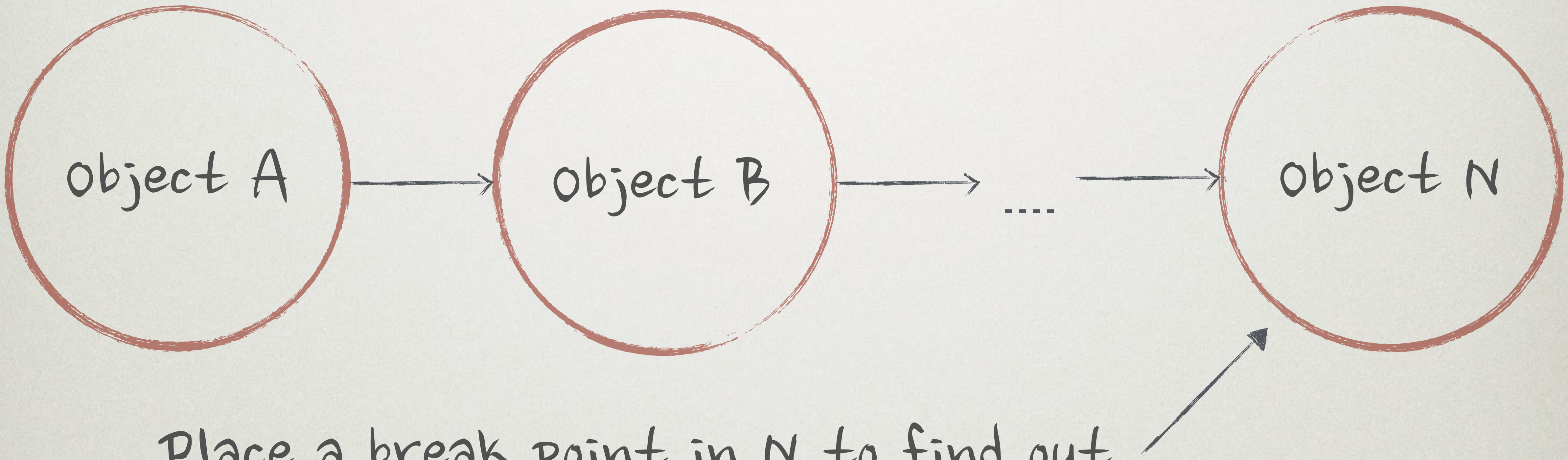
DEBUGGER IN LEGACY DESIGN

Legacy Design



DEBUGGER IN LEGACY DESIGN

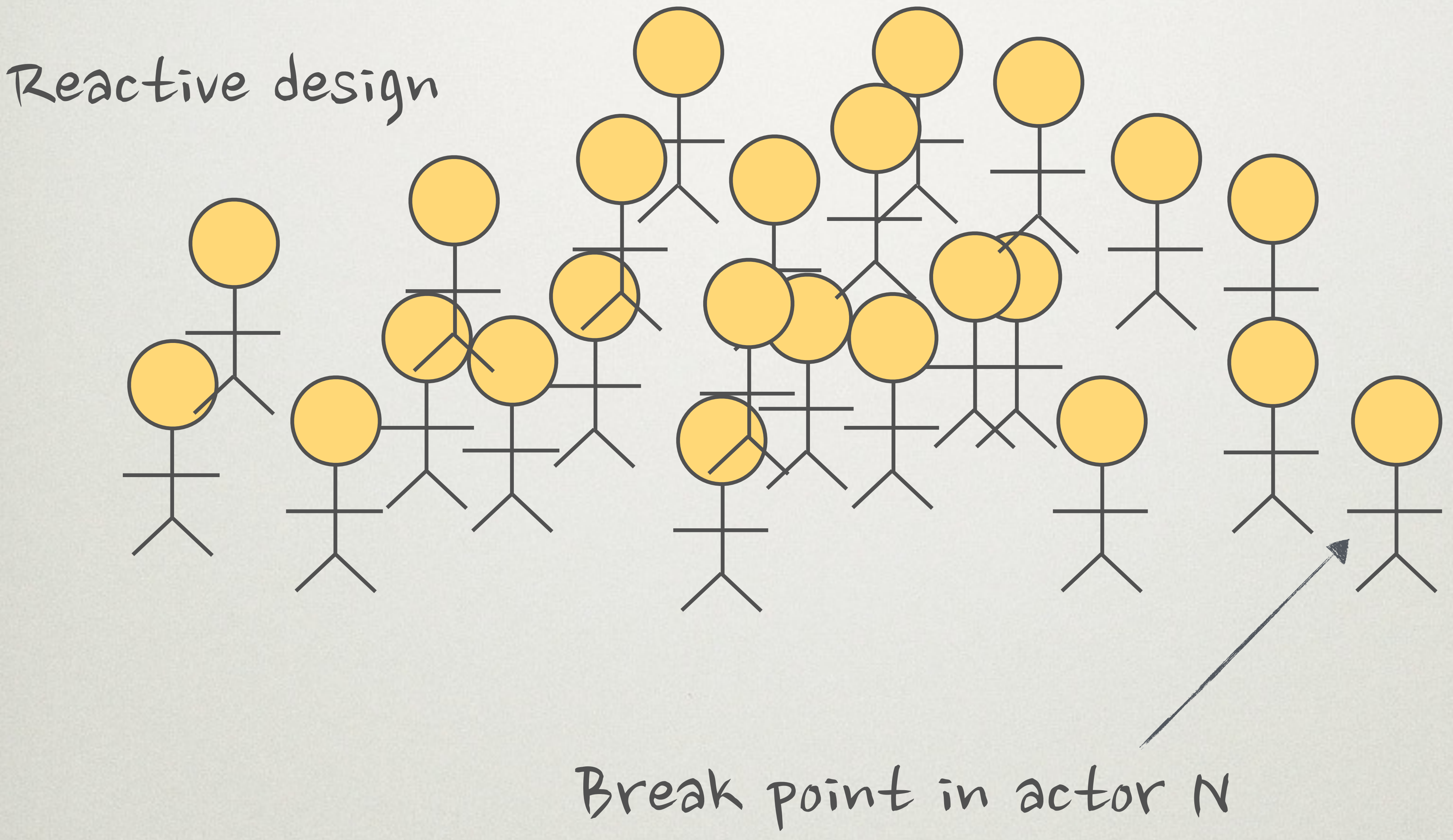
Legacy Design



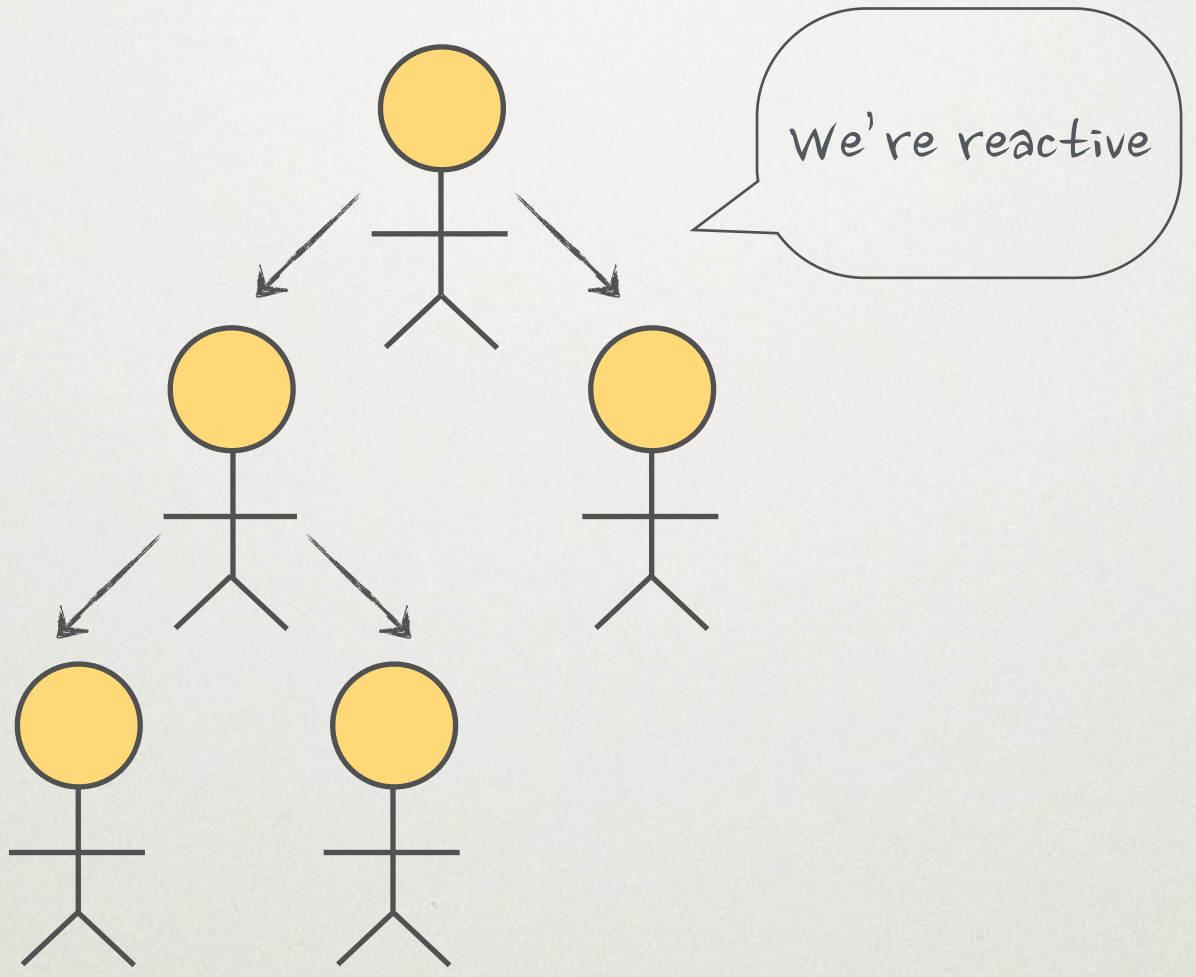
Place a break point in N to find out

- Which value caused the crash?
- Who created it?

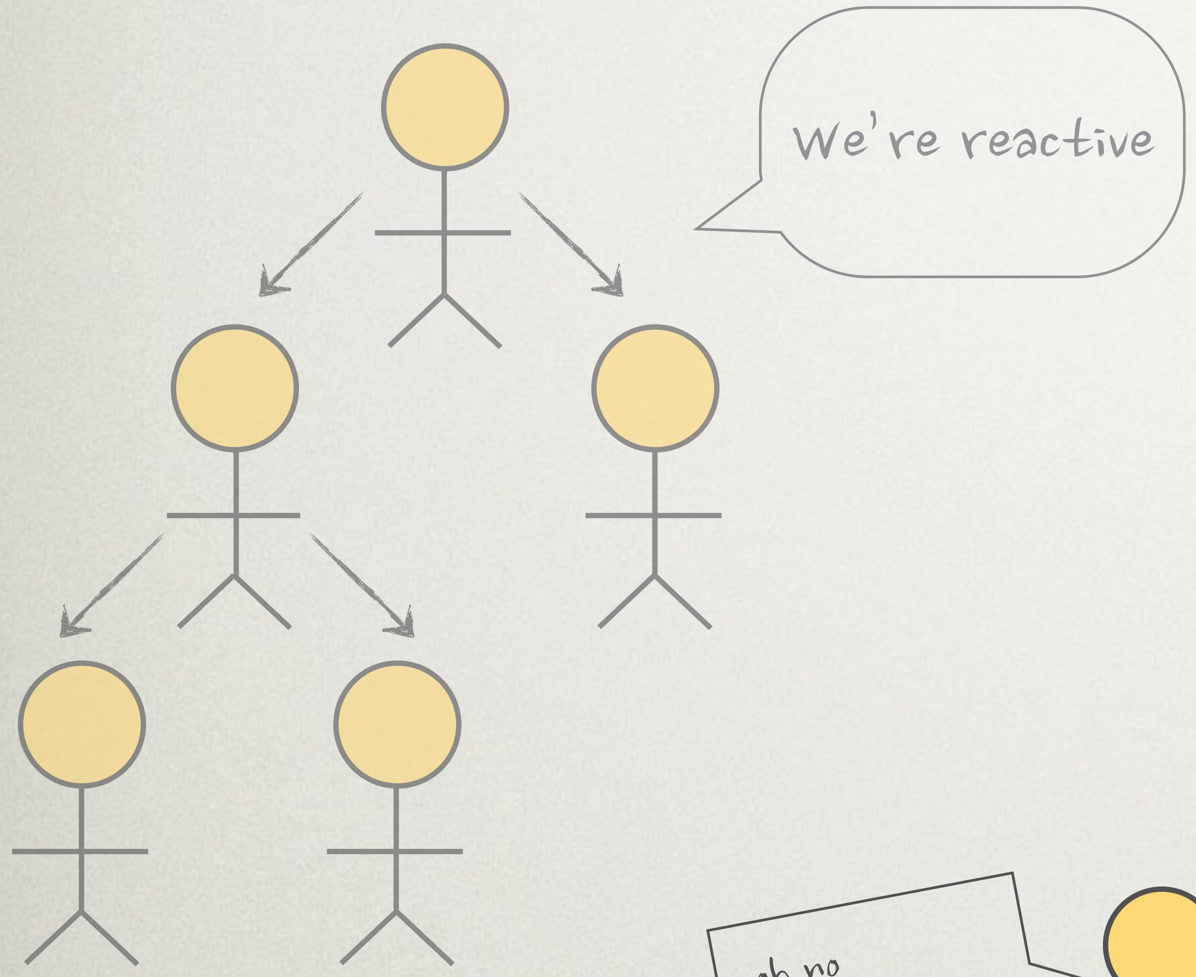
DEBUGGER IN
REACTIVE DESIGN

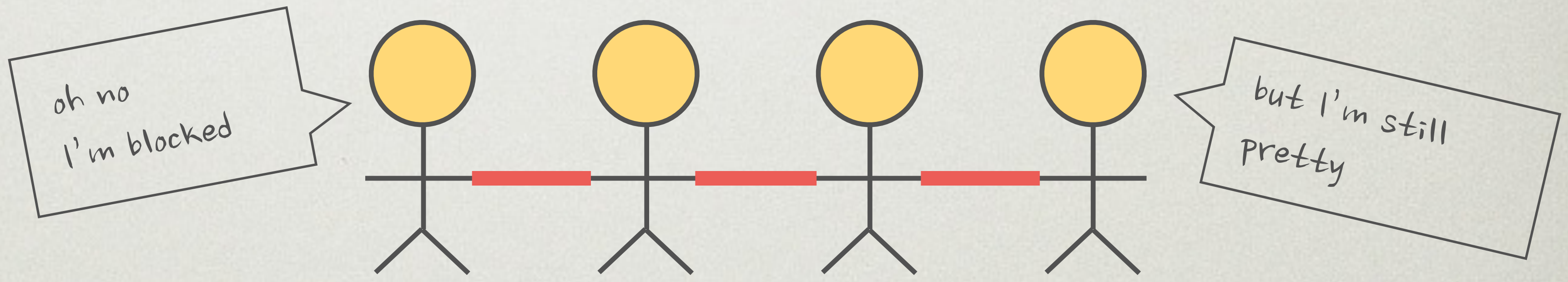


Supervision



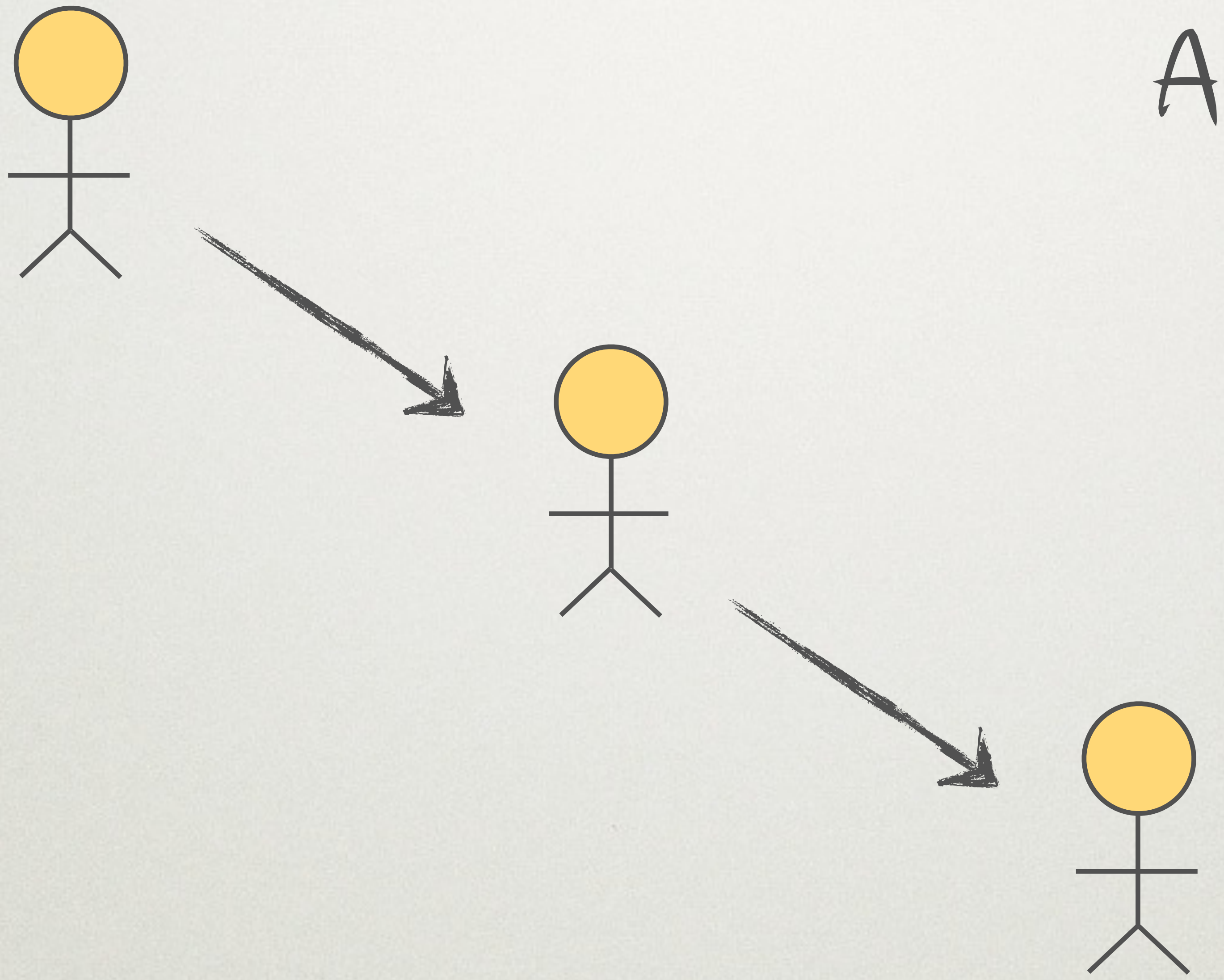
Supervision





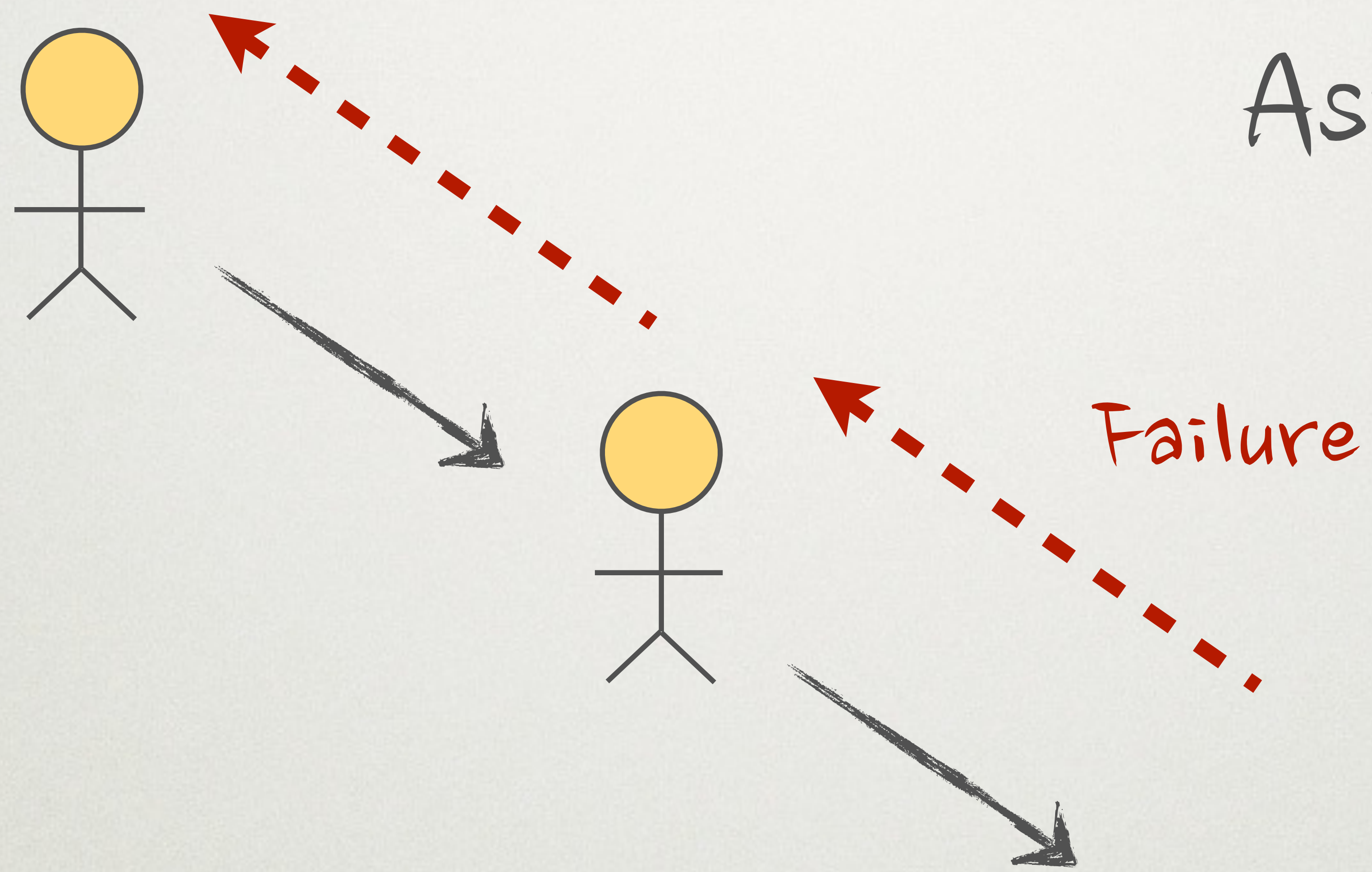
Supervision

Asynchronous



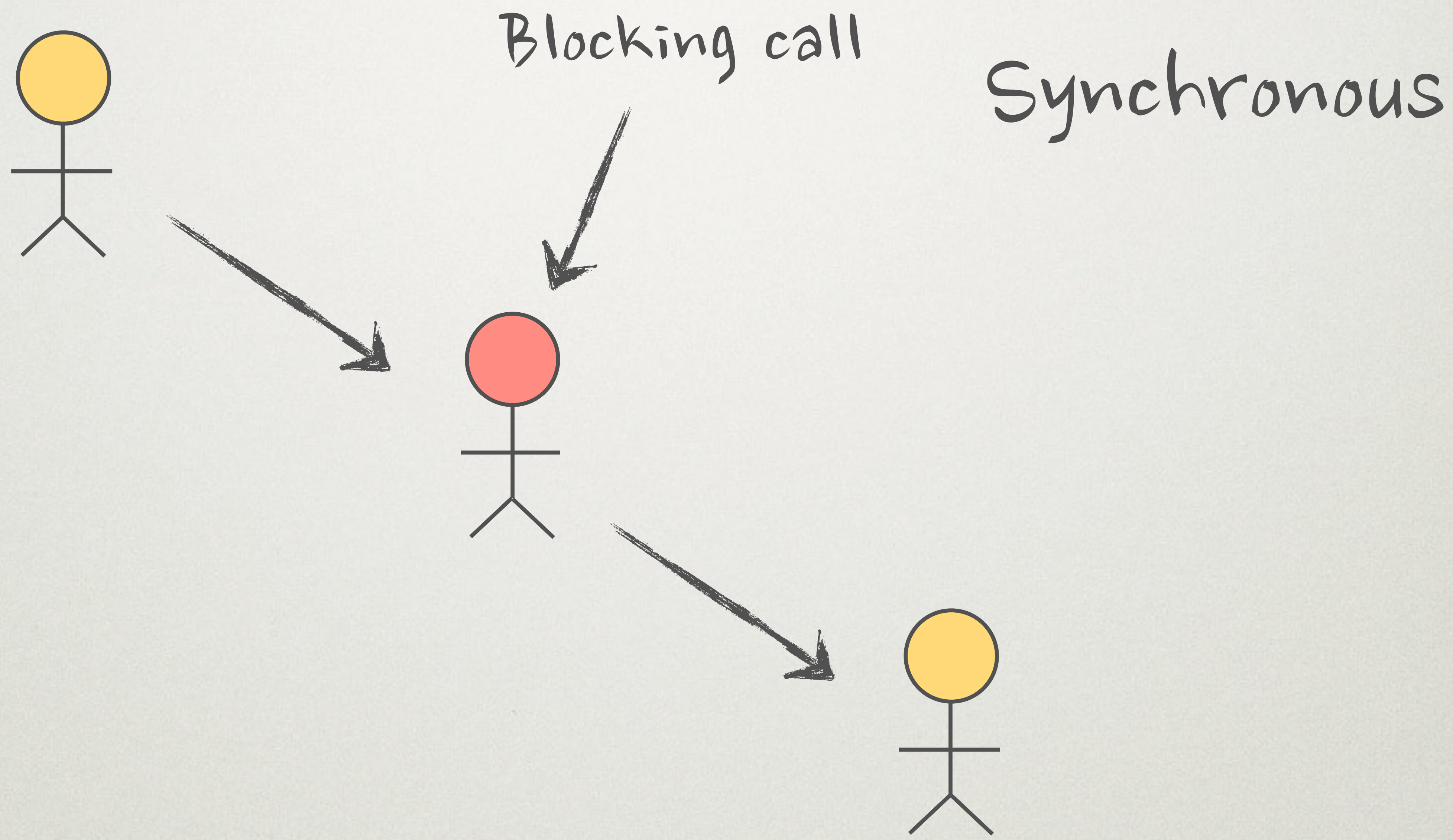
omega point.

Supervision



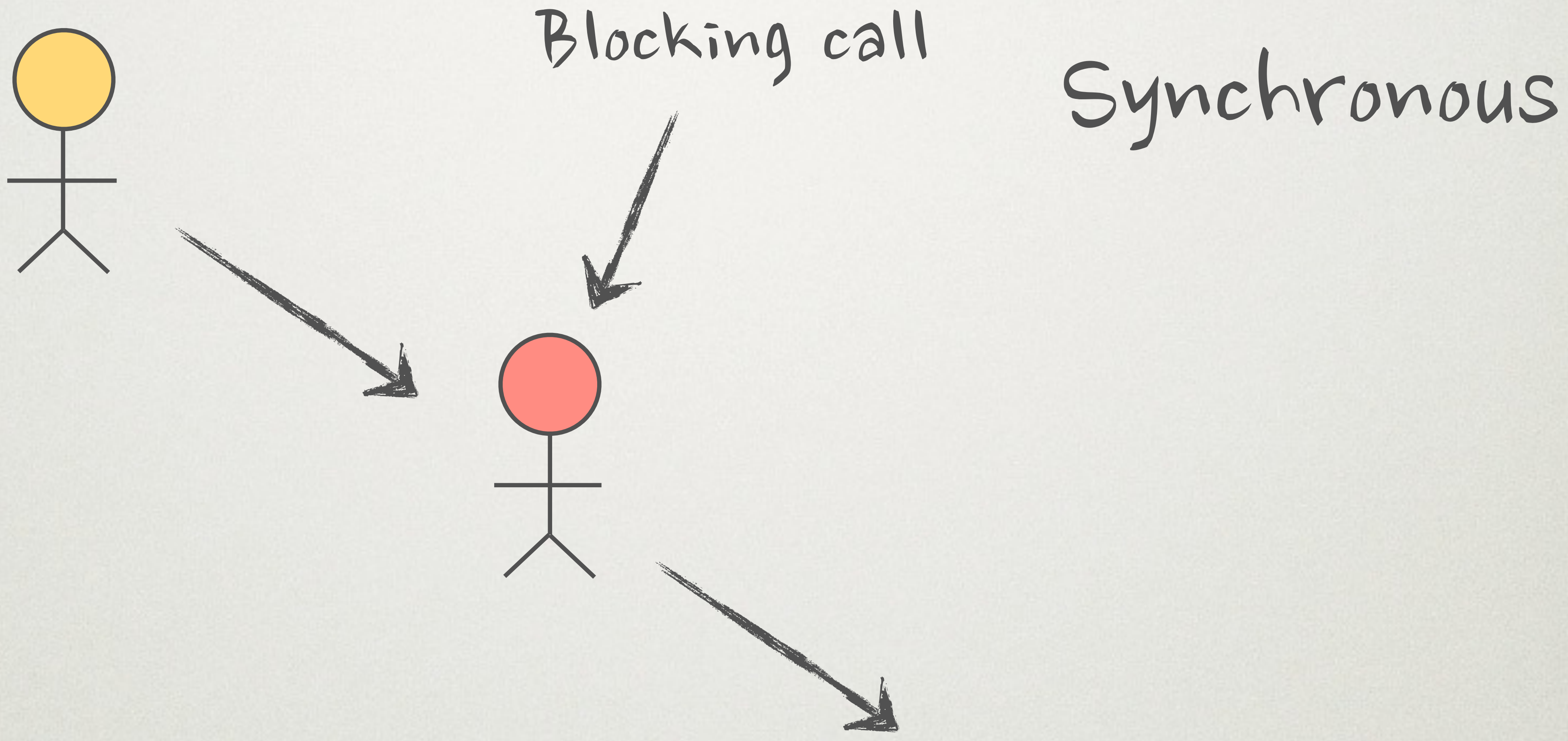
Asynchronous

Supervision

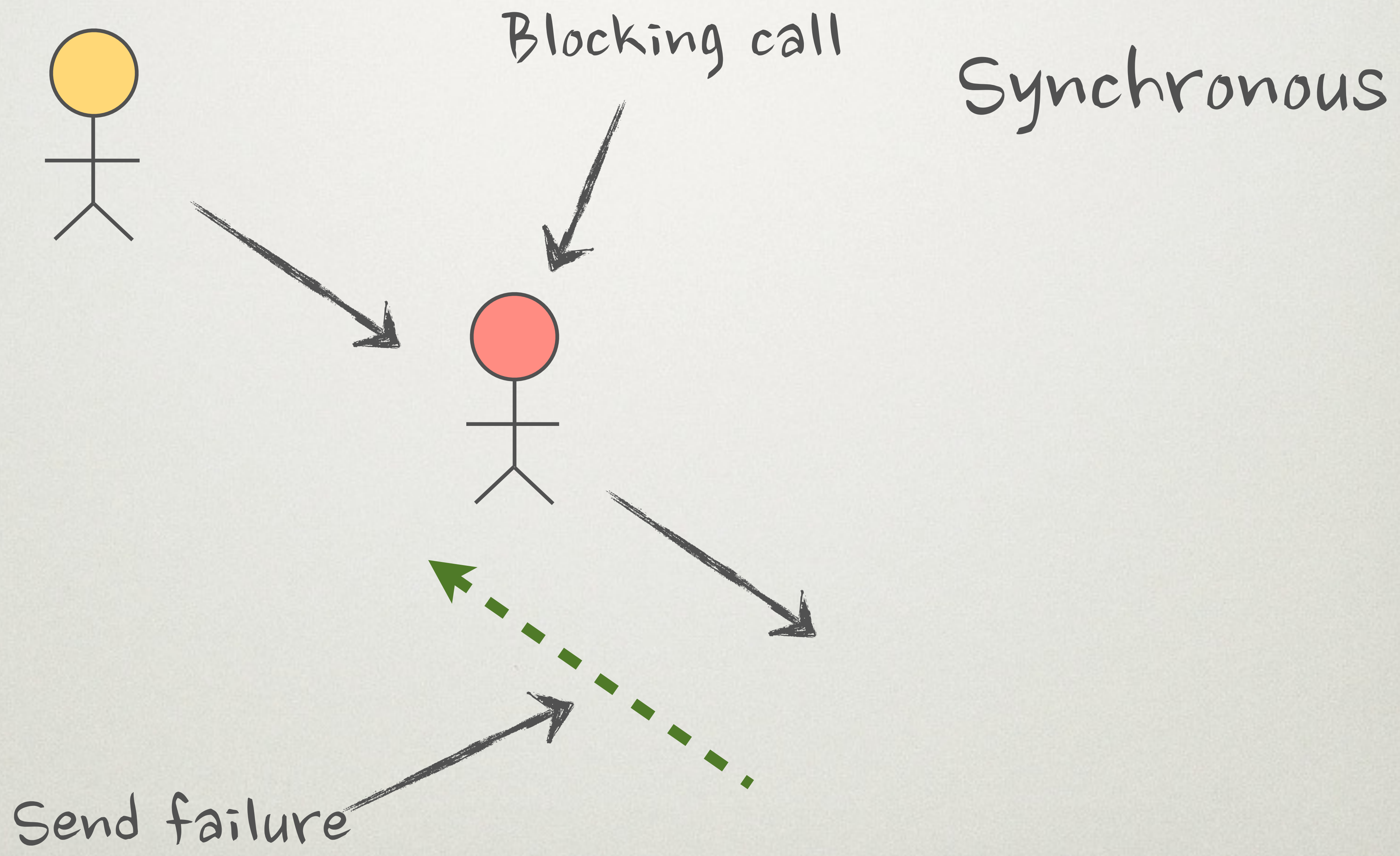


omega point.

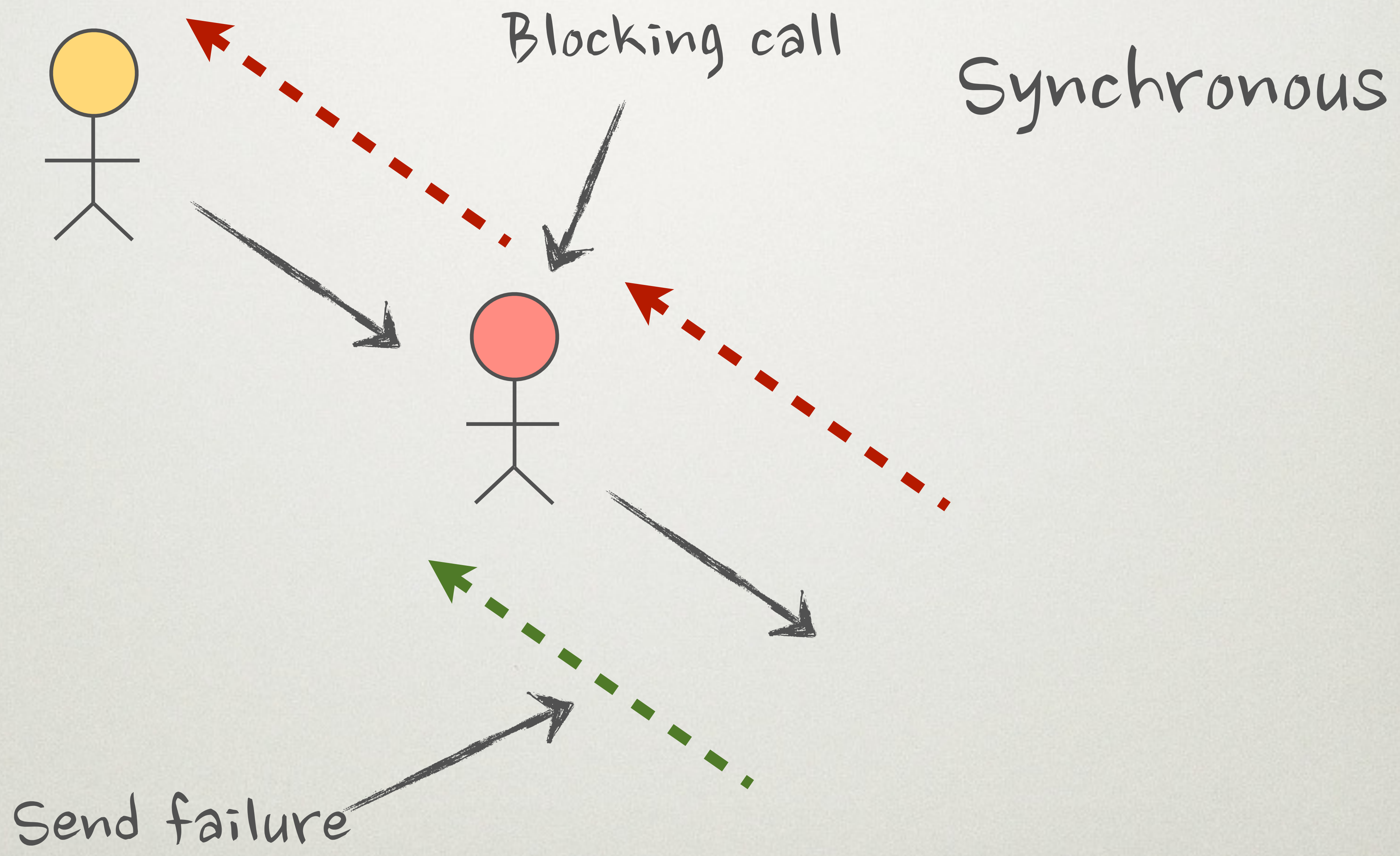
Supervision



Supervision



Supervision



Supervision

```
private ActorRef targetActor;
private ActorRef caller;
private Timeout timeout;
private Cancellable timeoutMessage;

@Override
public SupervisorStrategy supervisorStrategy() {
    return new OneForOneStrategy(0, Duration.Zero(), new Function<Throwable, SupervisorStrategy.Directive>() {
        public SupervisorStrategy.Directive apply(Throwable cause) {
            caller.tell(new Failure(cause), self());
            return SupervisorStrategy.stop();
        }
    });
}

@Override
public void onReceive(final Object message) throws Exception {
    if (message instanceof AskParam) {
        AskParam askParam = (AskParam) message;
        timeout = askParam.timeout;
        caller = sender();
        targetActor = context().actorOf(askParam.props);
        context().watch(targetActor);
        targetActor.forward(askParam.message, context());
        final Scheduler scheduler = context().system().scheduler();
        timeoutMessage = scheduler.scheduleOnce(askParam.timeout.duration(), self(), new AskTimeout(), context().dispatcher(), null);
    }
    else if (message instanceof Terminated) {
        sendFailureToCaller(new ActorKilledException("Target actor terminated."));
        timeoutMessage.cancel();
        context().stop(self());
    }
    else if (message instanceof AskTimeout) {
        sendFailureToCaller(new TimeoutException("Target actor timed out after " + timeout.toString()));
        context().stop(self());
    }
    else {
        unhandled(message);
    }
}

private void sendFailureToCaller(final Throwable t) {
    caller.tell(new Failure(t), self());
}
```

Supervision

- Write legacy code
- Use Akka actors

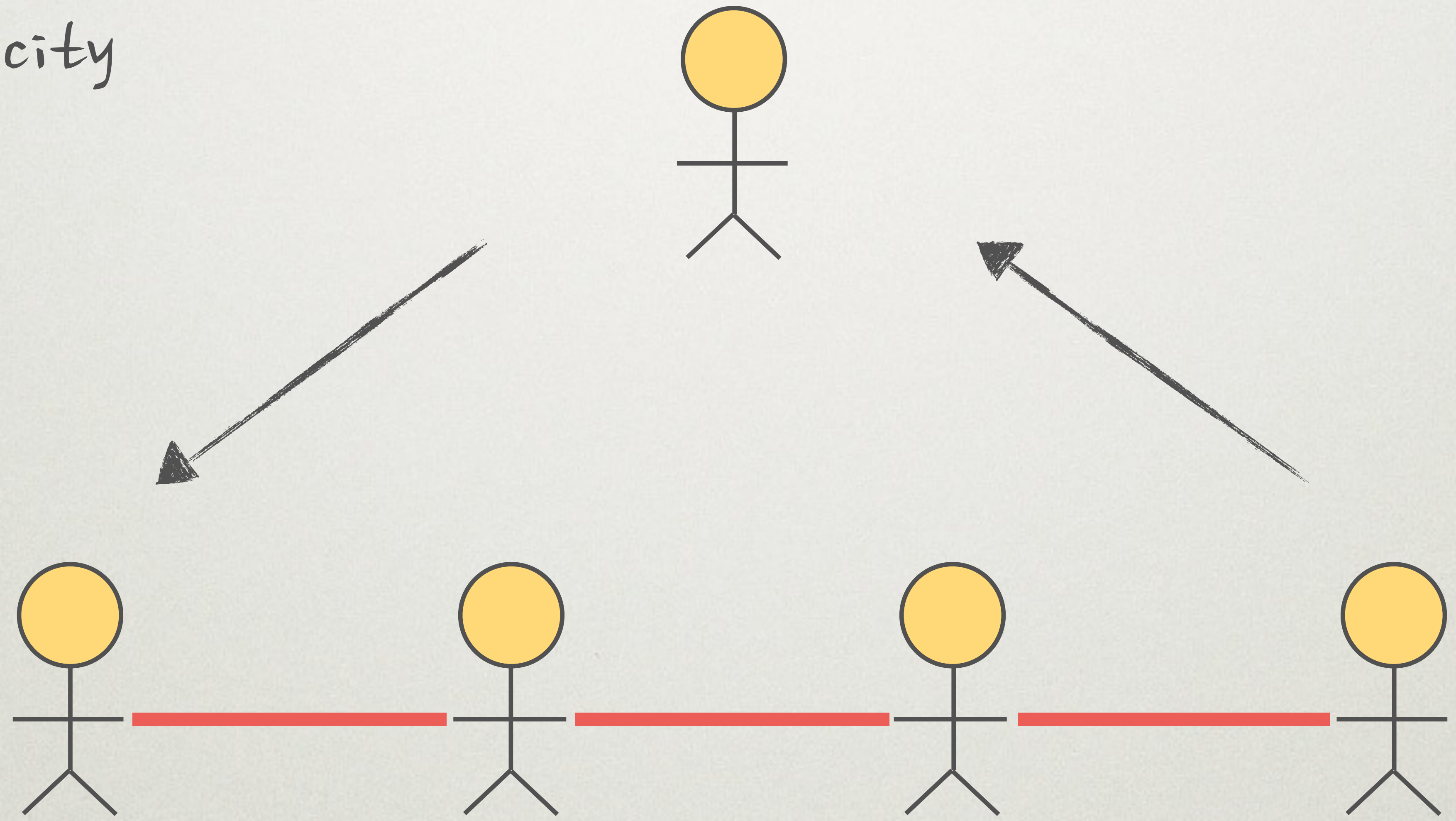
Supervision

- Write legacy code
- Use Akka actors

Sequential does not imply synchronicity

Changing mindset

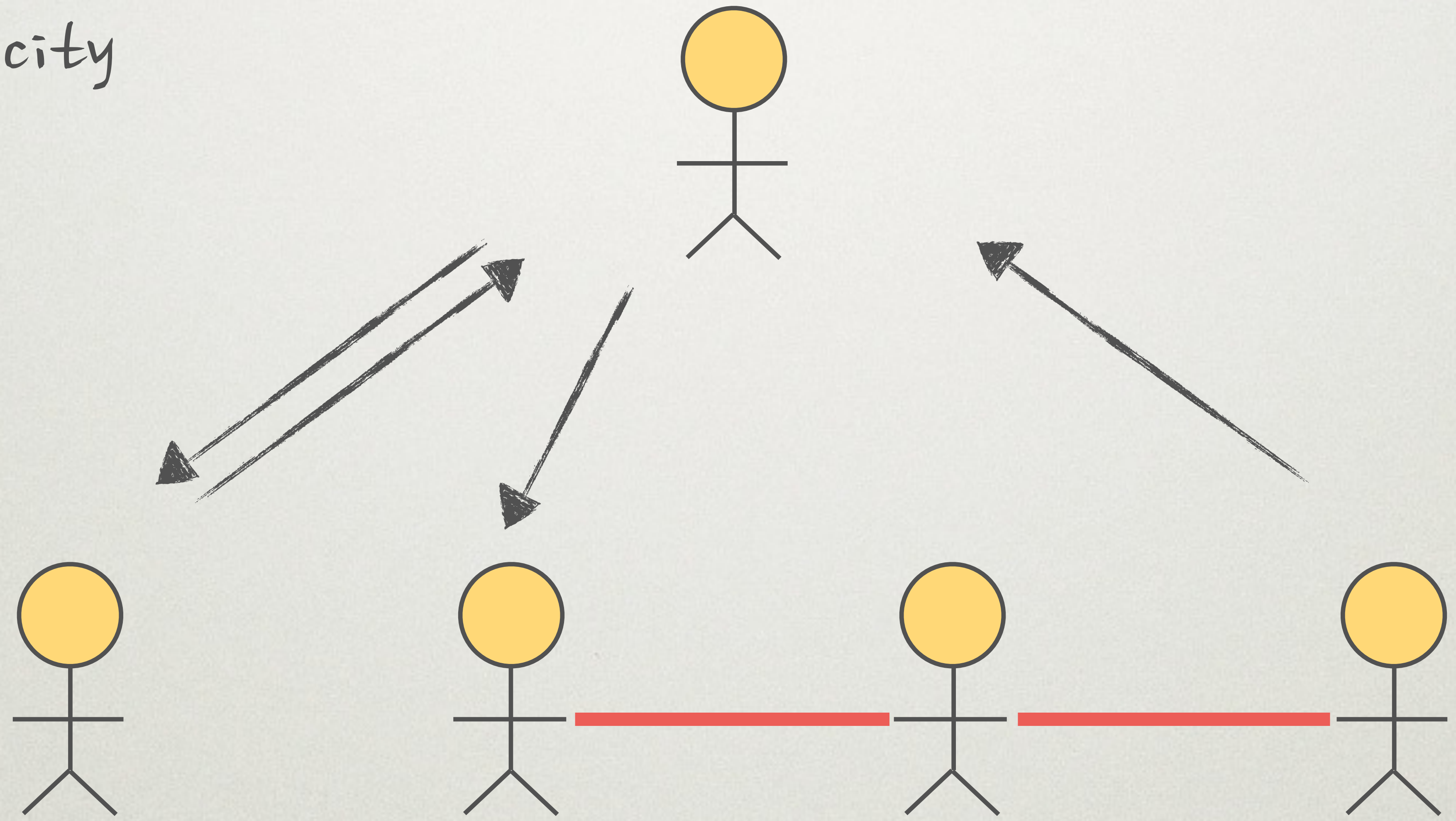
Sequential does not imply synchronicity



omega point.

Changing mindset

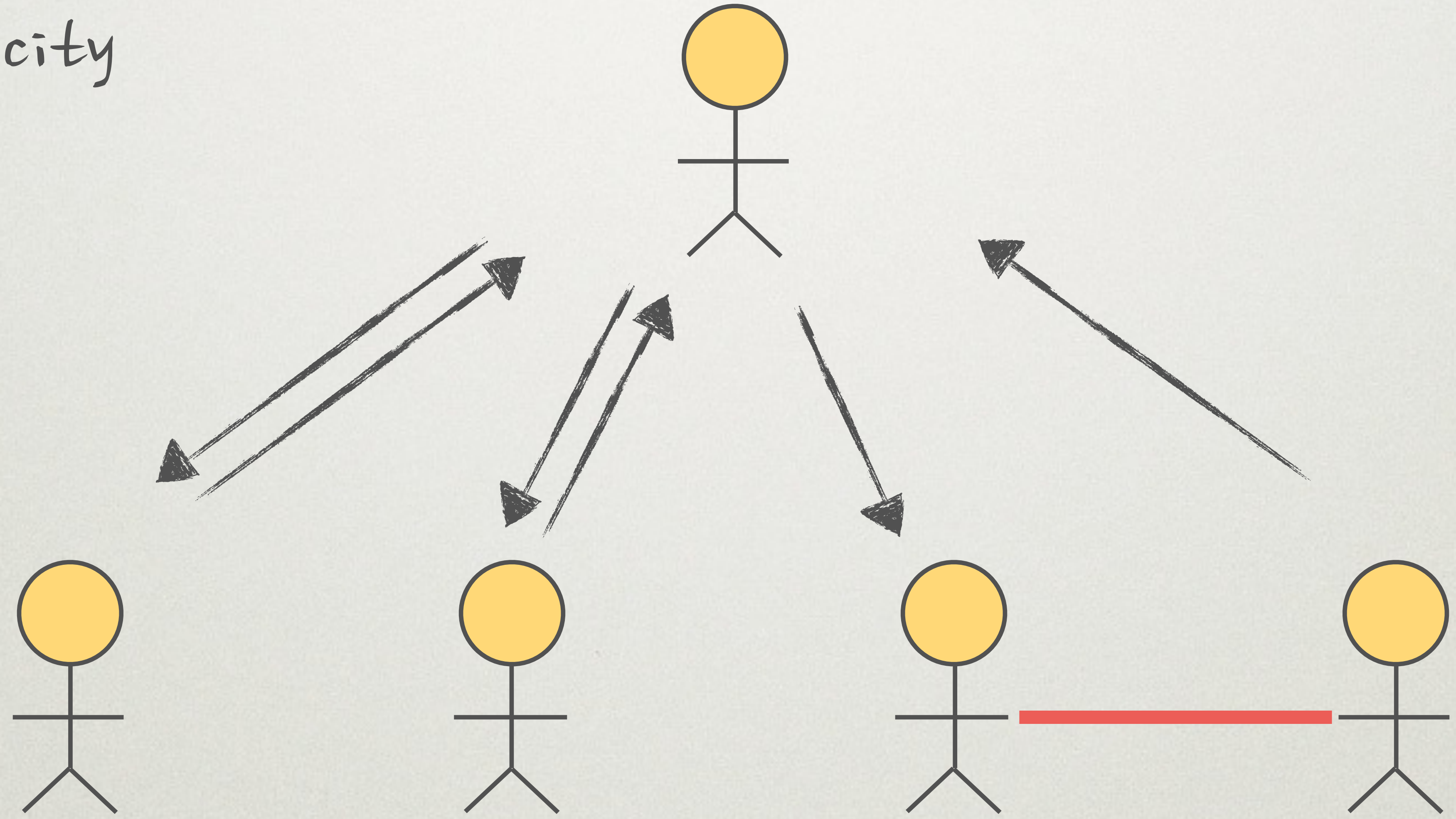
Sequential does not imply synchronicity



omega point.

Changing mindset

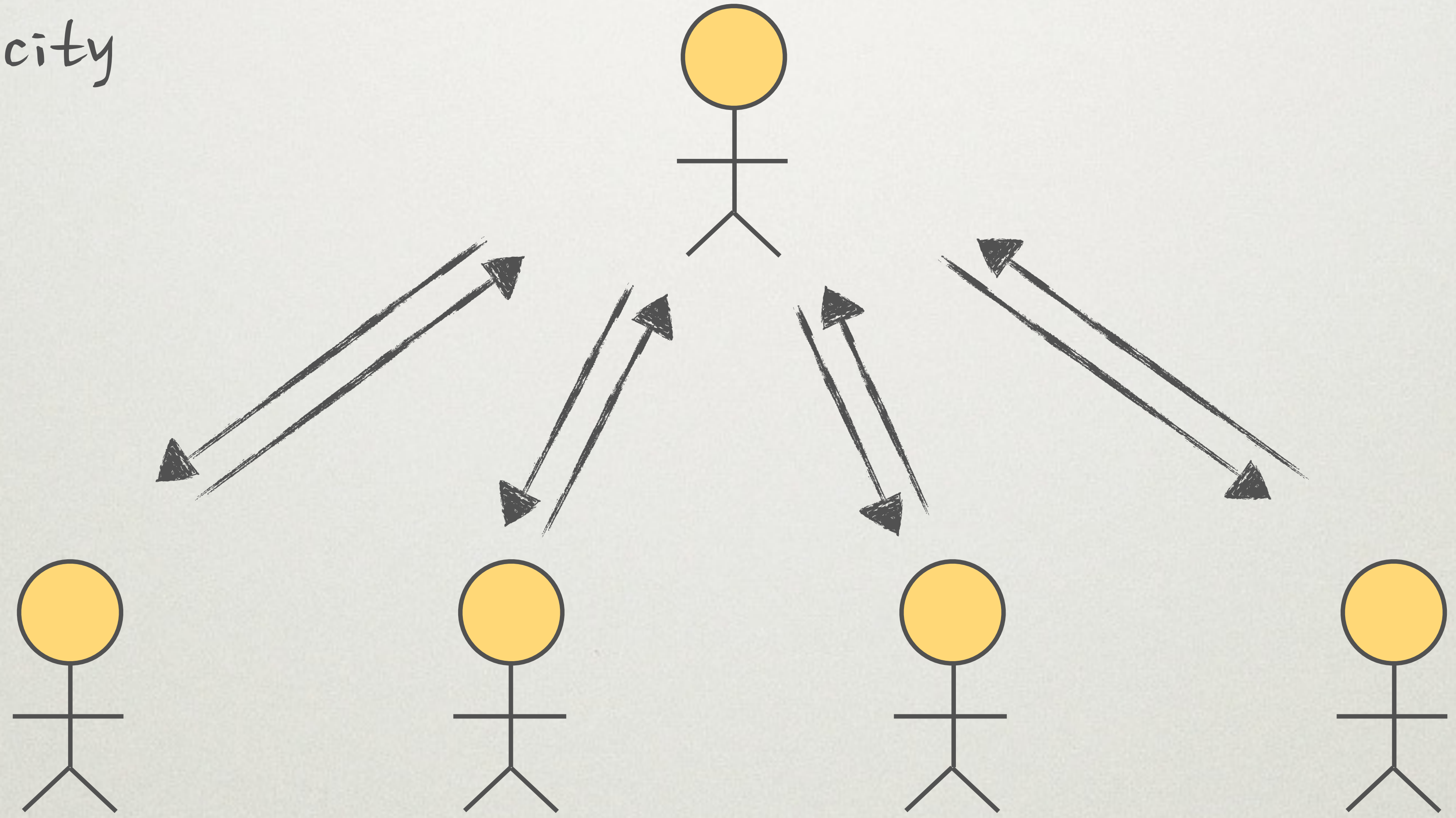
Sequential does not imply synchronicity



omega point.

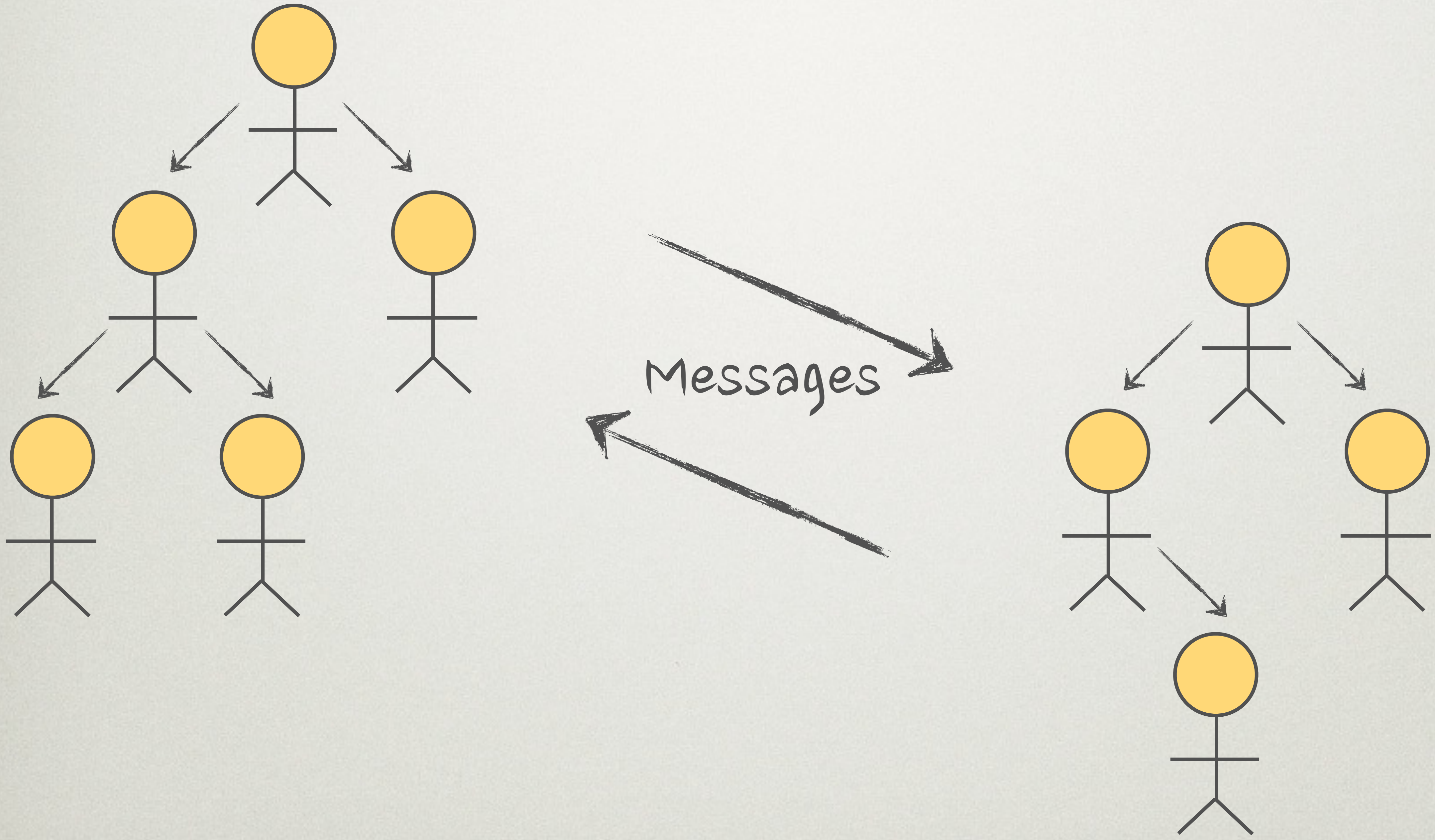
Changing mindset

Sequential does not imply synchronicity



omega point.

Loose Coupling?

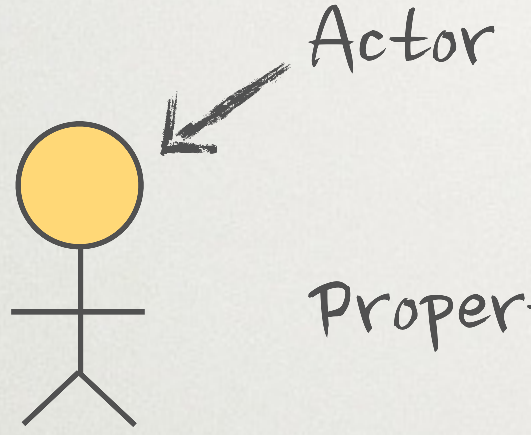


omega point.

The Shared Mutable State Trap

Keeping state between messages in an actor is extremely dangerous because it may cause a shared mutable state

The Shared Mutable State Trap



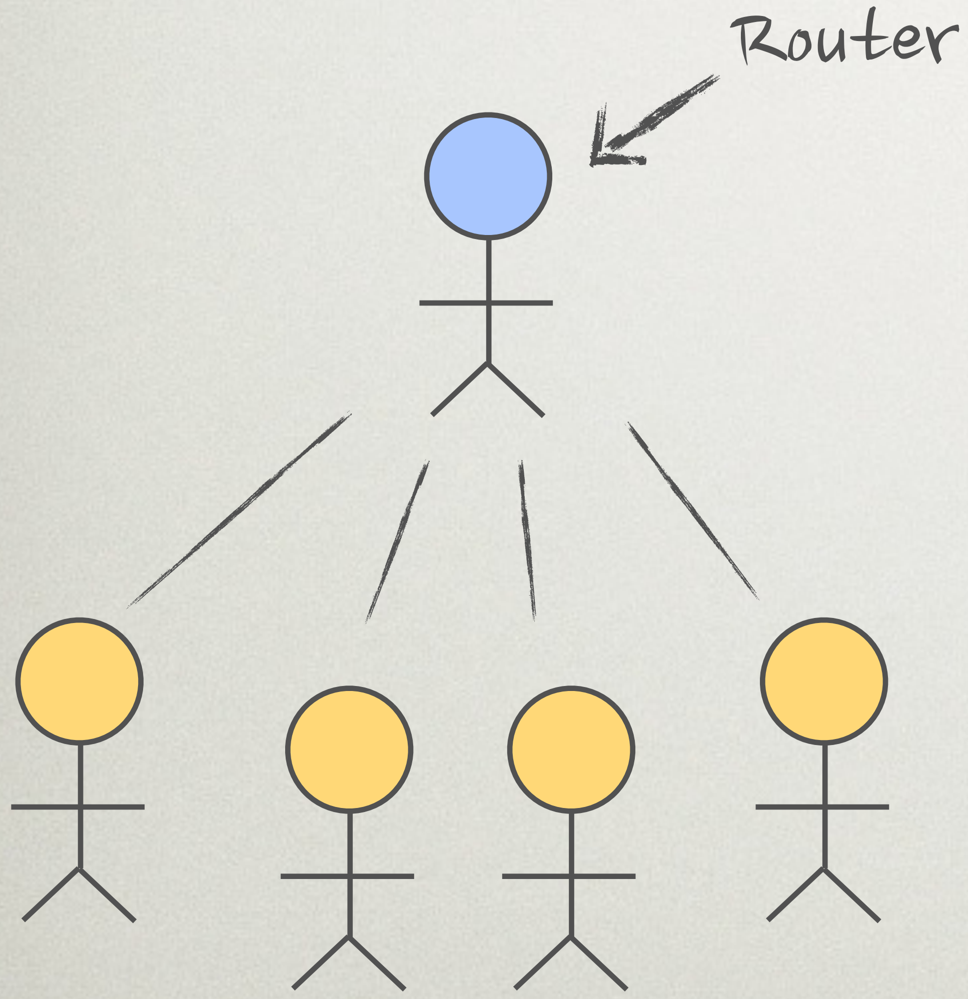
Properties

- One message queue
- Only one message is processed at a time
- Messages are processed in order received
- An actor may choose to divide and conquer a task by calling other actors

omega point.

The Shared Mutable State Trap

Actors may be grouped in a router

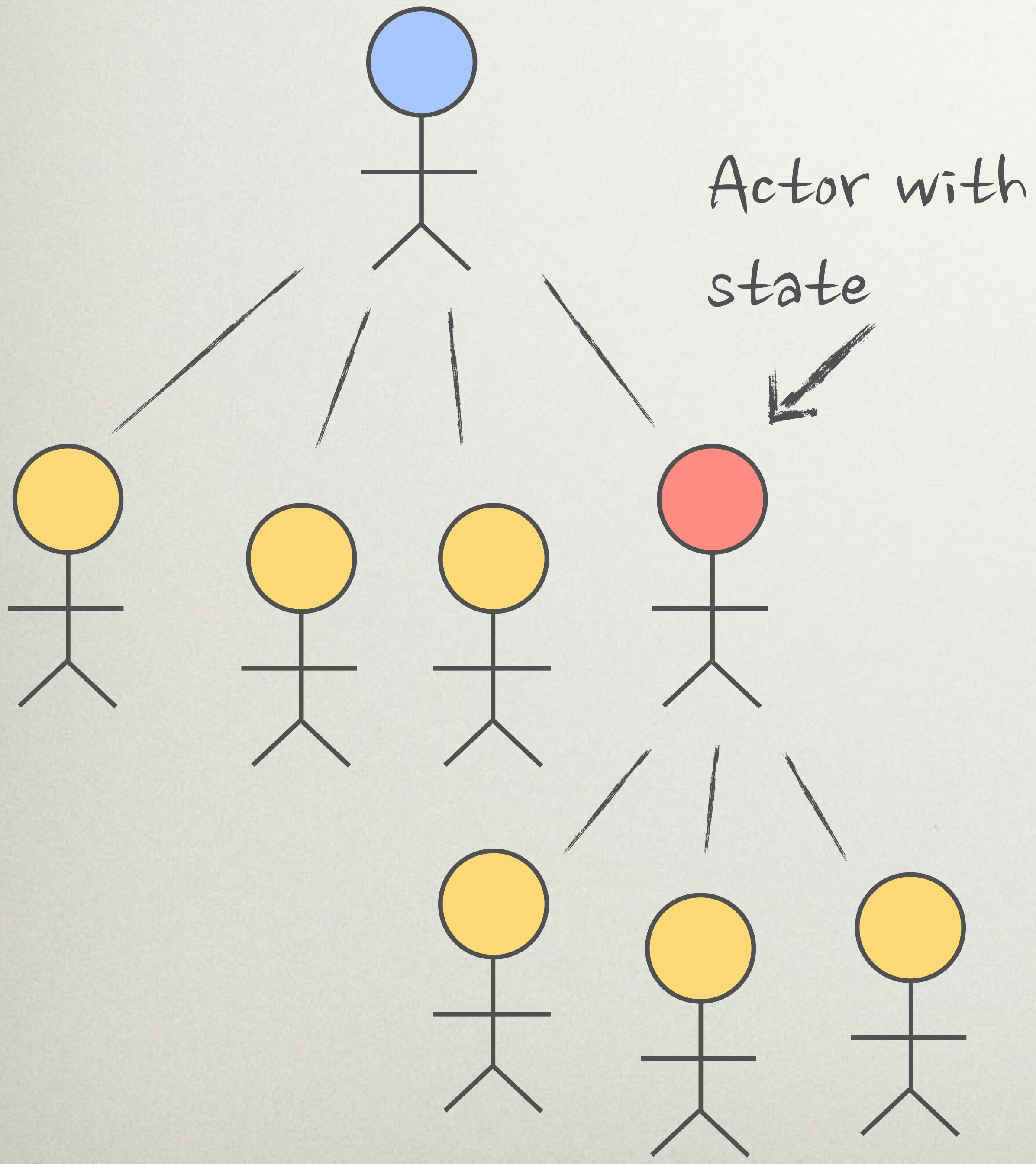


Properties

- Messages are distributed among actors according to some message delivery scheme, eg Round Robin or Smallest Mailbox

- A router may receive a lot of messages

The Shared Mutable State Trap

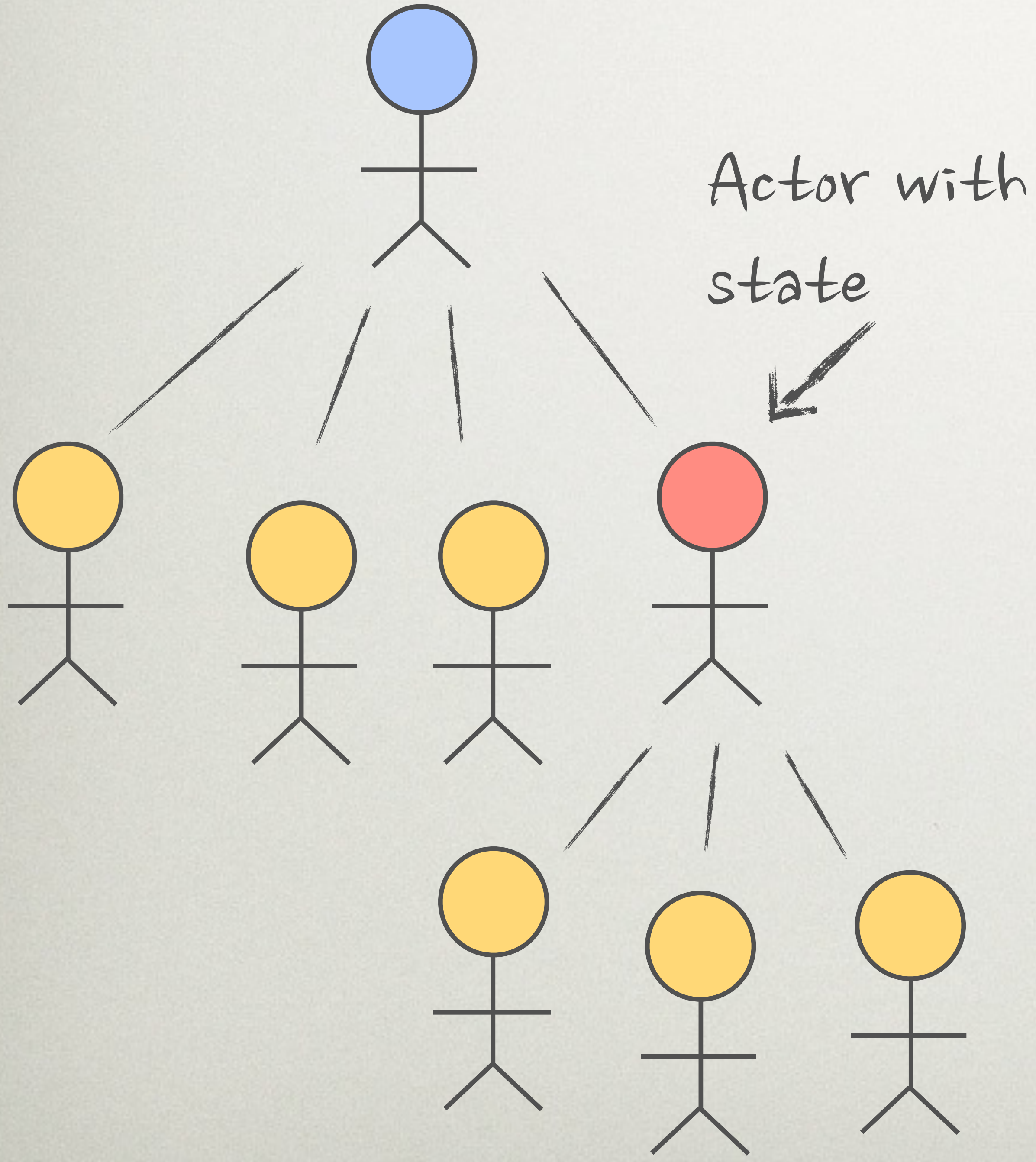


Scenario

- A stateful actor is part of a router

- It uses divide and conquer to solve its task

The Shared Mutable State Trap



Scenario

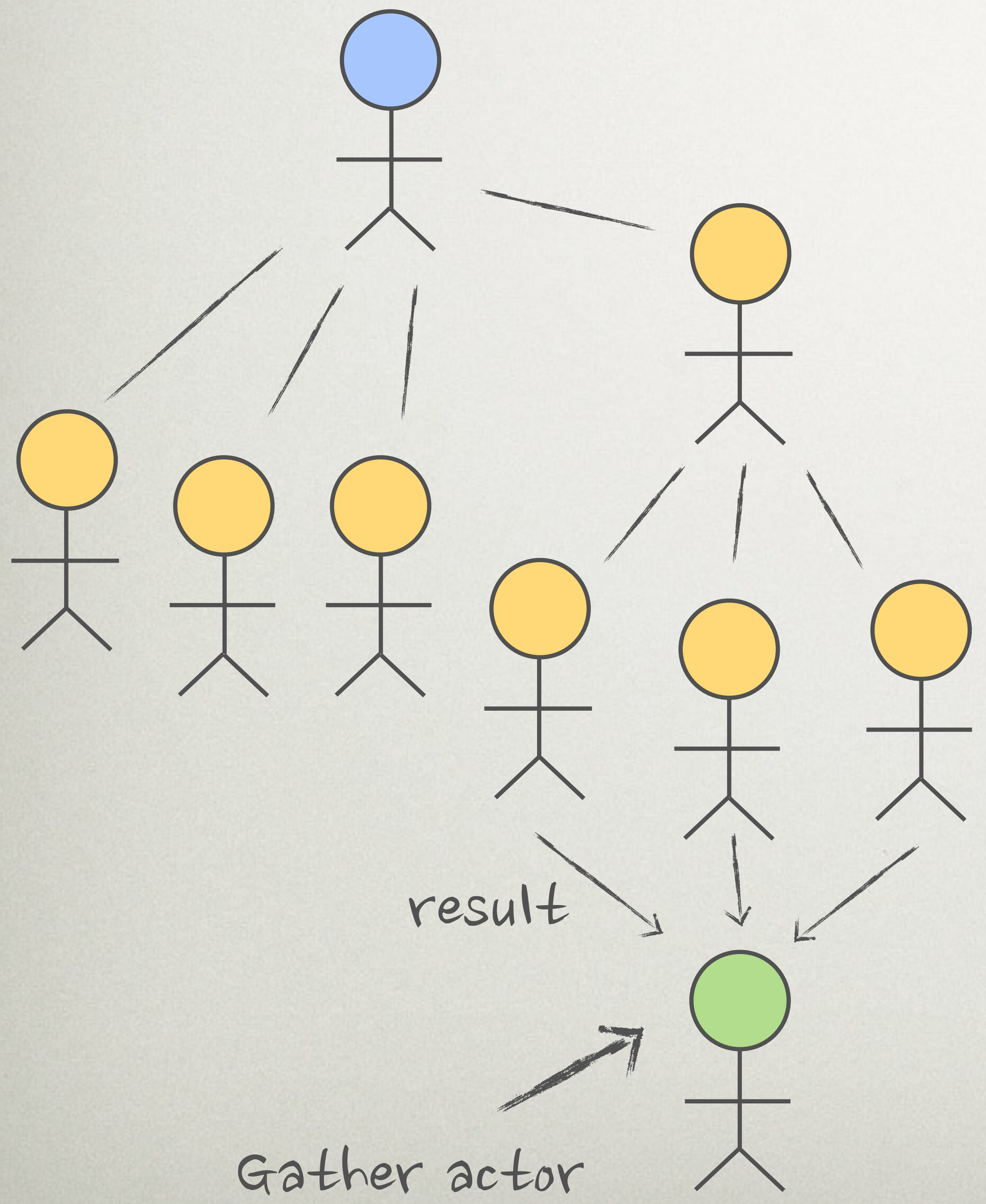
- Each message recived from the router resets the state

- New messages are inter mixed with child responses

- Hence, we have a shared mutable state

omega point.

The Shared Mutable State Trap



Solution

- Each request sent by the router results in a "Gather" actor

- The Gather actor is responsible for collecting the result

- A Gather actor is NEVER reused

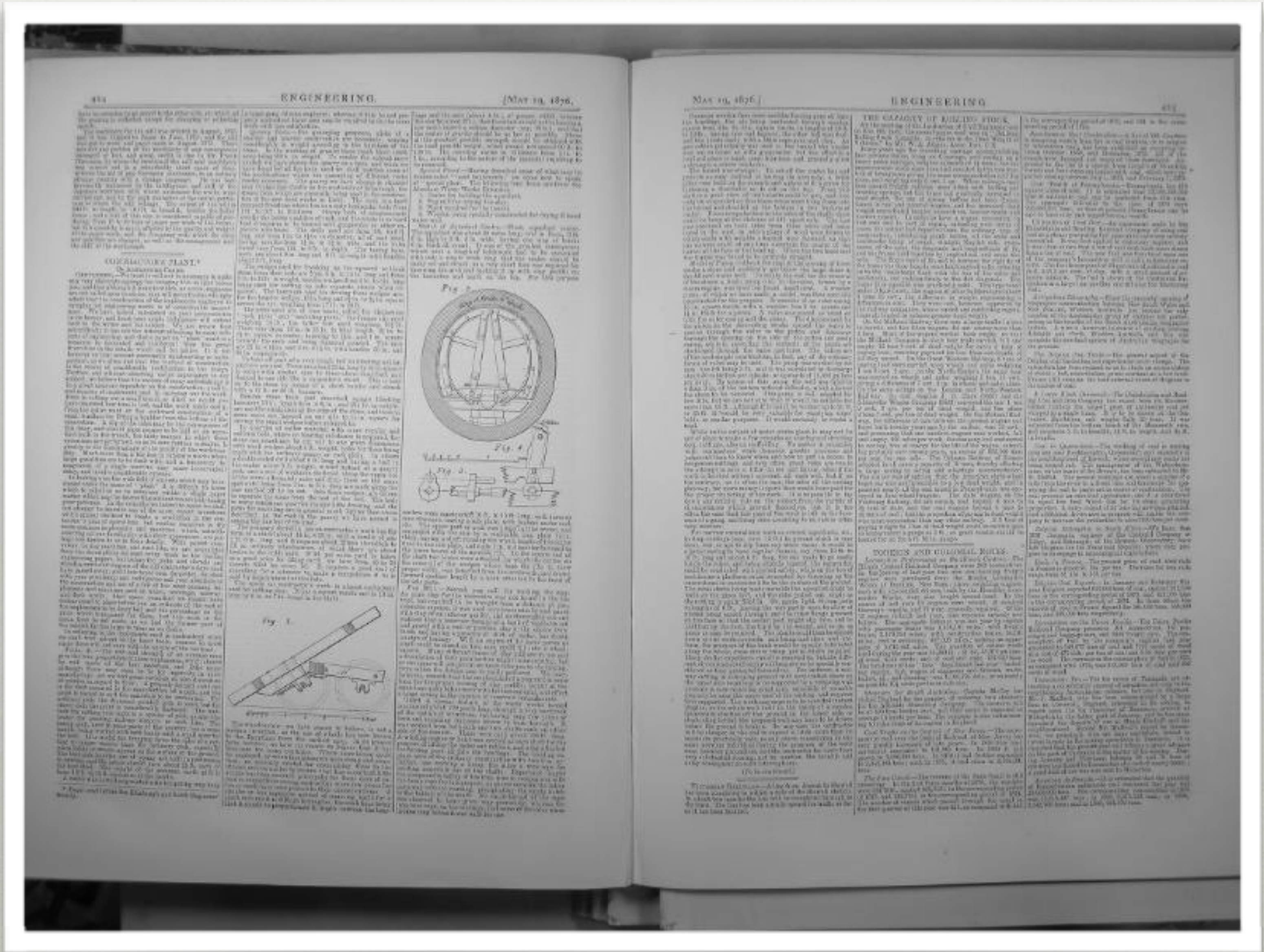
Key Take-Aways

- It's very easy to accidentally fall back to sequential "thinking" with state

- Often one does not realize this until the system is placed under heavy load

- Try to avoid state!

Readability


ENGINEERING. [MAY 19, 1876.

MAY 19, 1876.] ENGINEERING.

[illegible]

By Various. Edited by: W H Maw and J Dredge (Abandoned library clearance. Self photographed.) [Public domain], via Wikimedia Commons

Implications of untyped actors

Akka actors are untyped - by design

```
public void onReceive(Object message)
```

Implications of untyped actors

Akka actors are untyped - by design

```
public void onReceive(Object message)
```

- Readability
- No support from compiler/IDE

Message Handling V 1.0

```
@Override
public void onReceive(Object message) {
    if (message instanceof SomeMessage) {
        doStuff();
    }
    else {
        unhandled(message);
    }
}
```

V 1.0 - If-else contd.

```
@Override
public void onReceive(Object message) {
    if (message instanceof SomeMessage) {
        doStuff();
    }
    else if (message instanceof SomeOtherMessage) {
        doSomeOtherStuff();
    }
    else {
        unhandled(message);
    }
}
```

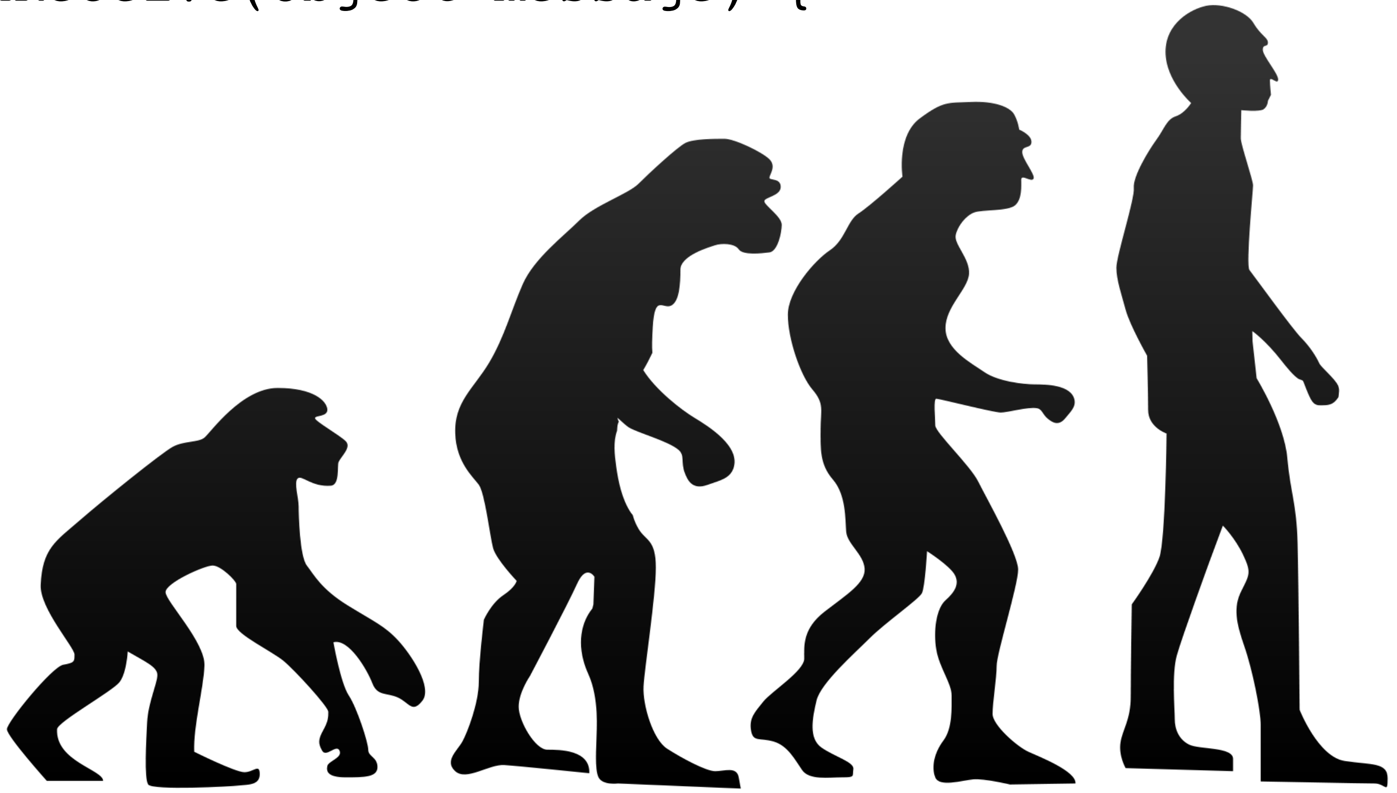
omega
point.

V 1.0 - If-else mess

```
@Override
public void onReceive(Object message) {
    if (message instanceof SomeMessage) {
        doStuff();
    }
    else if (message instanceof SomeOtherMessage) {
        doSomeOtherStuff();
    }
    else if (message instanceof YetAnotherMessage) {
        doEvenMoreStuff();
    }
    else {
        unhandled(message);
    }
}
```

Evolution of message handling

```
@Override
public void onReceive(Object message) {
    ...
}
```



http://upload.wikimedia.org/wikipedia/commons/thumb/6/69/Human_evolution.svg/2000px-Human_evolution.svg.png

V 2.0 - Overloading

```
public void onMessage(SomeMessage message) {...}

public void onMessage(SomeOtherMessage message) {...}
```

V 2.0 - Overloading

```
public void onMessage(SomeMessage message) {...}

public void onMessage(SomeOtherMessage message) {...}
```

```
@Override
public void onReceive(Object message) {
    ...

    methods.get(message.getClass()).invoke(this, message);

    ...
}
```

V 2.1 - Annotations

```
class Worker extends BaseActor {

   @Response
   public void onMessage(SomeMessage message) {...}


   public void onMessage(SomeOtherMessage message) {...}
}
```

V 3.0 - Contract

```
interface Messages {

    void doSomething(SomeMessage message);

    void doSomethingElse(SomeOtherMessage message);

}
```

V 3.0 - Contract

```
interface Messages {

    void doSomething(SomeMessage message);

    void doSomethingElse(SomeOtherMessage message);

}



SomeActor extends BaseActor implements Messages {...}
```

V 3.0 - Contract

```
class Worker extends BaseActor implements Messages {

   public void handleResponse(SomeMessage message) {...}

   public void handleRequest(SomeOtherMessage message) {...}
}
```

V 3.0 - Contract

```
BaseActor extends UntypedActor {

    @Override
      public void preStart() {
        methodDelegate = new MethodDelegate(this);
    }

    @Override
    public void onReceive(Object message) {
        if (methodDelegate.onReceive(message)) {
            return;
        }
        unhandled(message);
    }
}
```

omega point.

Before

```
class Worker extends BaseActor implements Messages {
   @Override
   public void onReceive(Object message) {
       if (message instanceof SomeMessage) {
           doStuff();
       }
       else if (message instanceof SomeOtherMessage) {
           doSomeOtherStuff();
       }
       else if (message instanceof YetAnotherMessage) {
           doEvenMoreStuff();
       }
       else {
           unhandled(message);
       }
```

omega point.

After

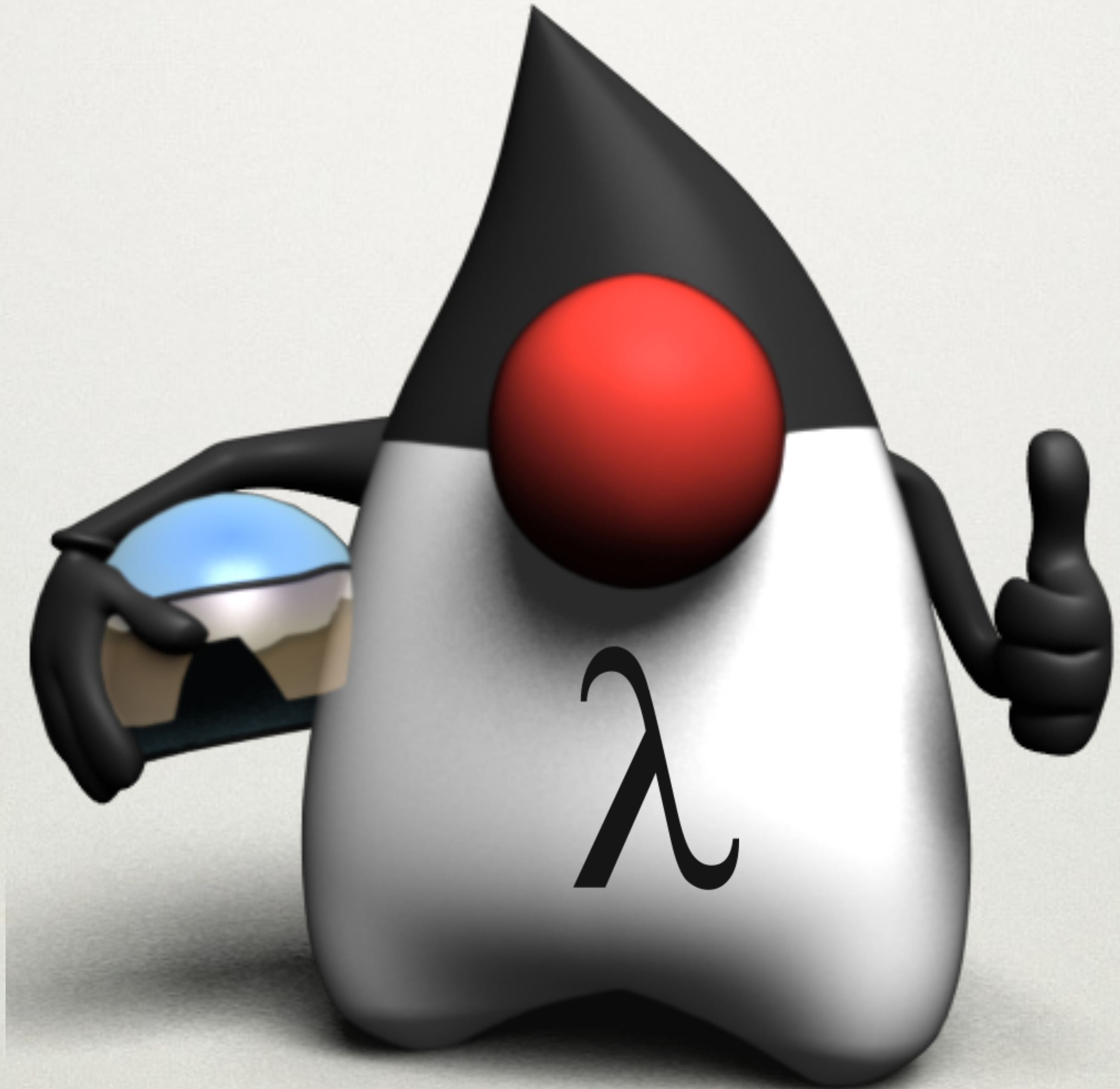
```
class Worker extends BaseActor implements Messages {

   public void handleResponse(SomeMessage message) {
      doStuff();
   }

   public void handleRequest(SomeOtherMessage message) {
      doSomeOtherStuff();
   }

   public void process(YetAnotherMessage message) {
      doEvenMoreStuff();
   }
}
```

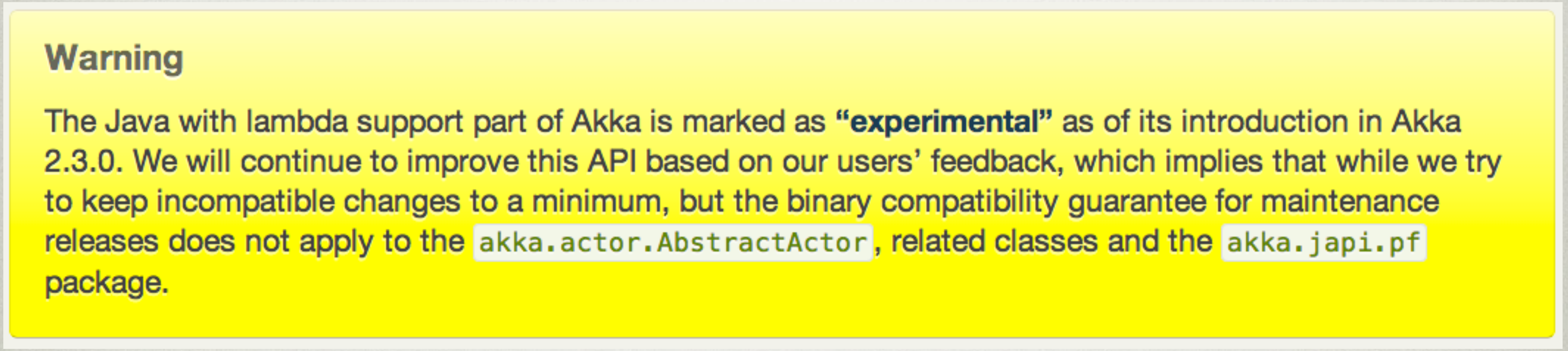
Akka & Java 8 Lambdas



omega point.

- Duke: https://duke.kenai.com/thumbsup/DukeWithHelmet.png
- Lambda: http://en.wikipedia.org/wiki/Lambda_calculus

Warning


Warning

The Java with lambda support part of Akka is marked as **“experimental”** as of its introduction in Akka 2.3.0. We will continue to improve this API based on our users’ feedback, which implies that while we try to keep incompatible changes to a minimum, but the binary compatibility guarantee for maintenance releases does not apply to the `akka.actor.AbstractActor`, related classes and the `akka.japi.pf` package.

http://doc.akka.io/docs/akka/2.3.3/java/lambda-actors.html

omega point.

The Receive Builder

```
public class MyActor extends AbstractActor {
    public MyActor() {
        receive(ReceiveBuilder.
                match(SomeMessage.class, m -> {
                    doStuff();
                }).
                match(SomeOtherMessage.class, m -> {
                   doSomeOtherStuff();
                }).
                match(YetAnotherMessage.class, m -> {
                   yetSomeMoreStuff();
                }).
                matchAny(this::unhandled).
                build());
    }
}
```

omega
point.

A Quick Note On Performance

- The partial functions created by the ReceiveBuilder consist of multiple lambda expressions for every match statement which may be hard for the JVM to optimize

- The resulting code may not be as performant as the corresponding untyped actor version

Conclusions

- Contracts to explicitly define behavior works pretty well
- Can be a useful tool in your toolbox
- Java 8 and lambdas provides additional approaches
- Scala can give similar support via traits & match

Testing Akka Code

Interact by sending messages

BDD Scenarios

Integration tests

Scenarios

Case A1: Sort any sequence of numbers in increasing order

Given a sequence of numbers in decreasing order

When applying the sort algorithm

Then the resulting sequence of numbers is in increasing order

The Test

```
@Test
public void caseA1() {

    given(sequence(9,8,7,6,5,4,3,2,1,0));

    whenSorting();

    thenResultIs(sequence(0,1,2,3,4,5,6,7,8,9));

}
```

omega point.

Key Take-Aways

- It's possible to use Akka in legacy code
- Akka is Scala & Java compliant
- Akka is a toolkit with a lot of goodies
- Stop writing legacy code

Thank you

@DanielDeogun @DanielSawano

omega point.