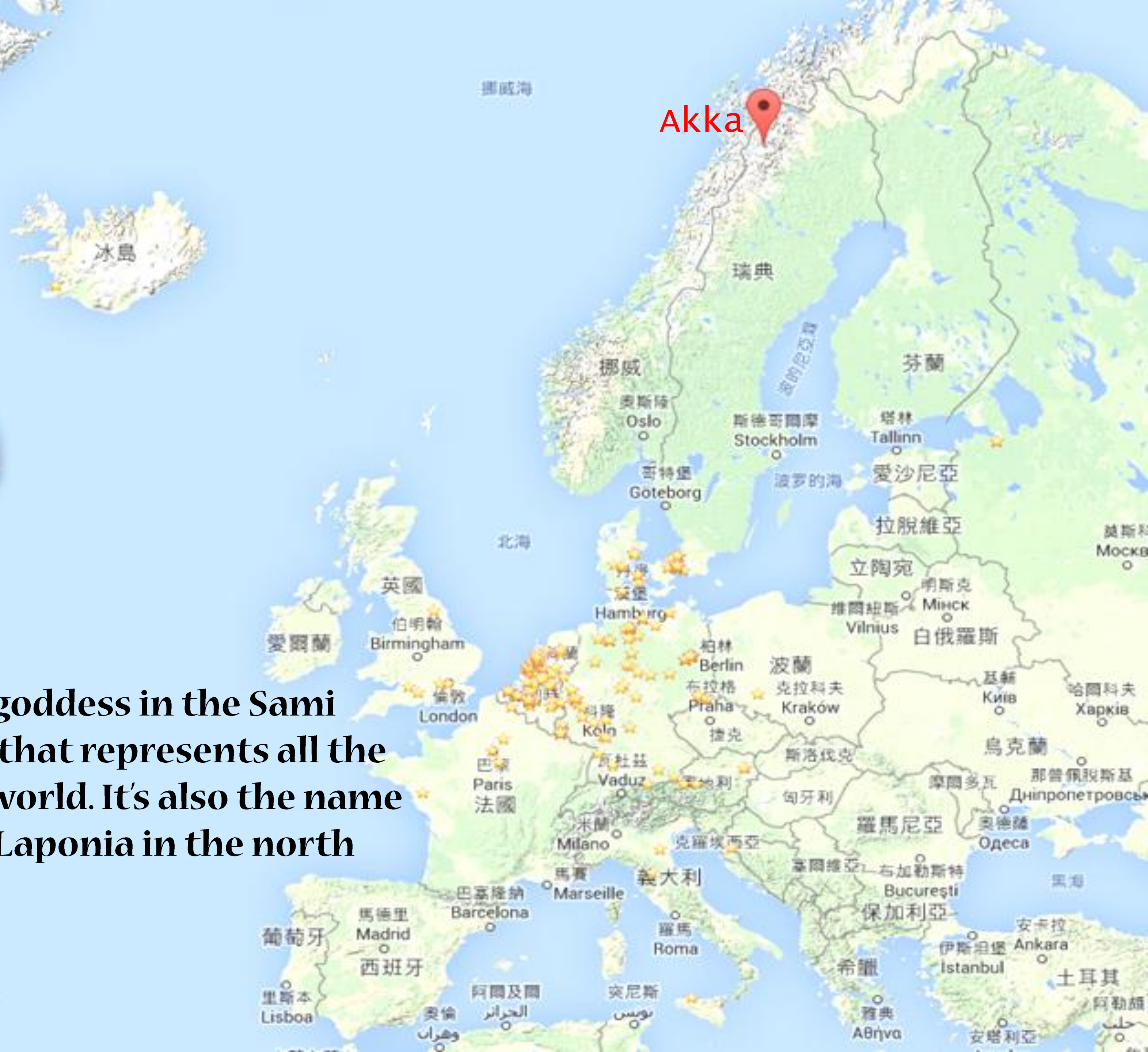
Akka Actor Introduction

Gene



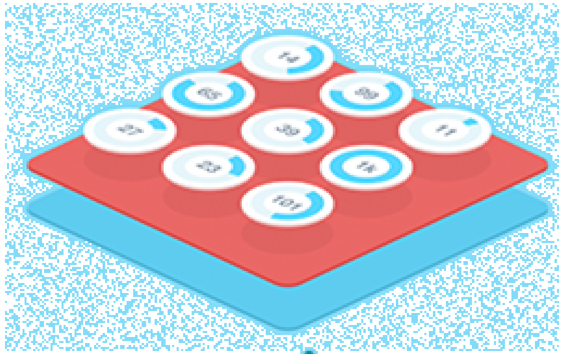
Akka

The name comes from the goddess in the Sami (Native swedes) mythology that represents all the wisdom and beauty in the world. It's also the name of a beautiful mountain in Laponia in the north part of Sweden.

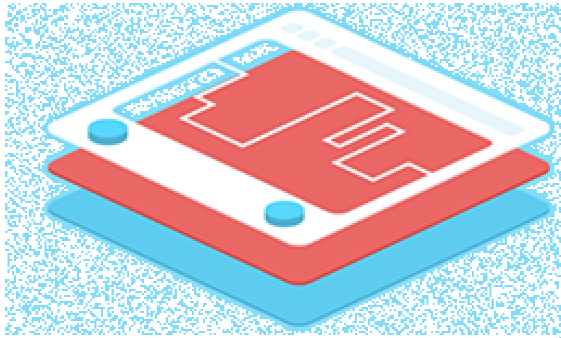
Agenda

1. Akka Introduction
2. Core Operations
3. WebSocket with Actor
4. Remote Actor

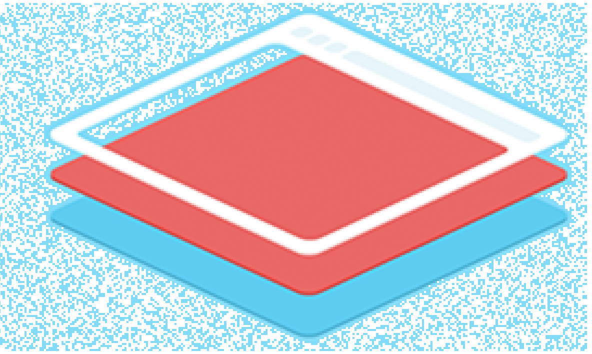
Typesafe



akka



Scala



play

→Toolkit/Library(*.jar)

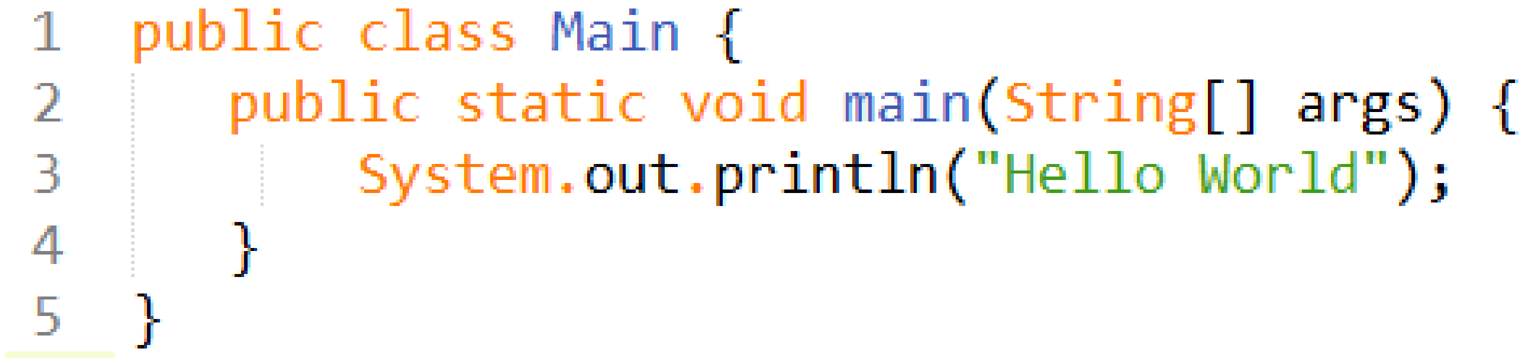
→Web Application Framework

→Programming Language

Scala

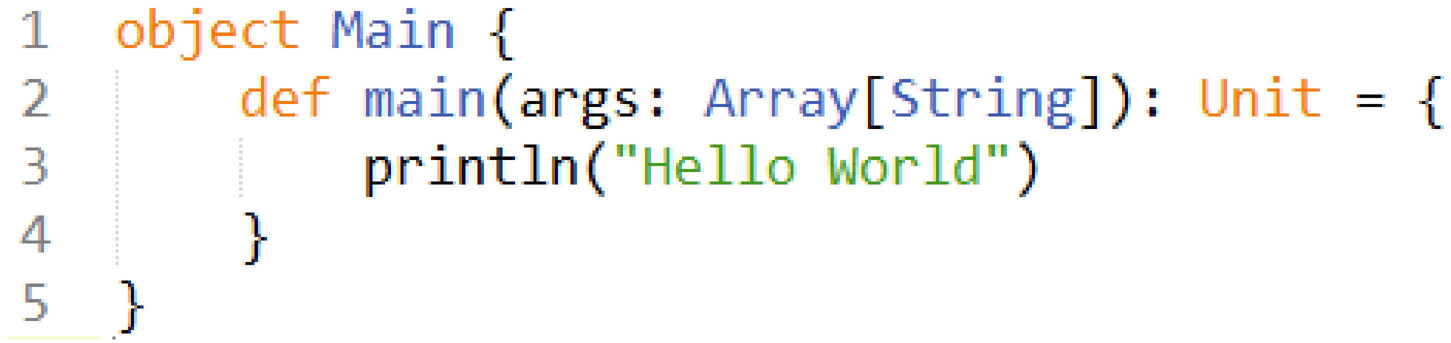
- Functional as well as Object-Oriented
- Scala is compatible with Java
- Scala is compiled to Java byte-codes and run on Java Virtual Machine

Java



```java
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```

Scala



```scala
object Main {
    def main(args: Array[String]): Unit = {
        println("Hello World")
    }
}
```

1.Akka Introduction

akka

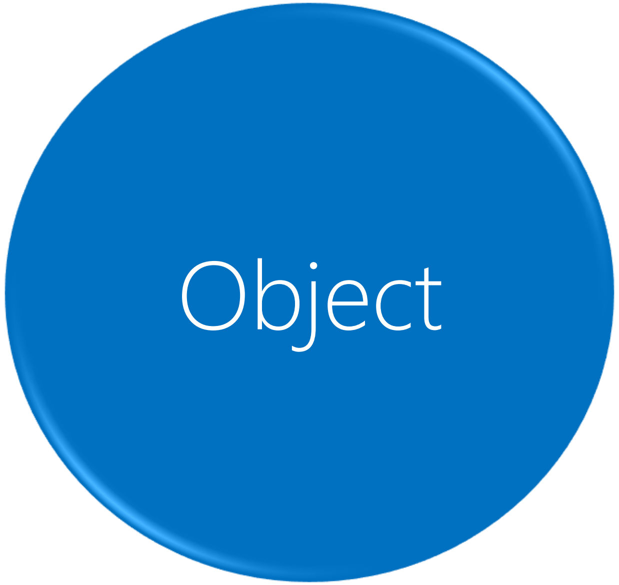
TDC
Telefónica
CSC
htc quietly brilliant
hp
BLIZZARD ENTERTAINMENT
UBS
SIEMENS
amazon.com
KLOUT
W3C
HSBC
cisco
HUAWEI
banksimple
svt
Autodesk
CREDIT SUISSE
IGN
AtoS
O2
vmware
dialog Smart Stream Platform
SEVEN Networks
xerox
UNIBET
DRW TRADING GROUP
OOYALA
novus
BBC
navirec
moshi monsters
T8 Webware
CARTOMAPIC
JUNIPER NETWORKS
Maritime Poker
ngmoco:)
zeebox
Answers.com
azavea

Akka

- 一個 JVM 上 Actor Model 的實作

1. Concurrency	Actors
2. Distribution	Remoting
3. Fault-tolerance	Supervision

Java

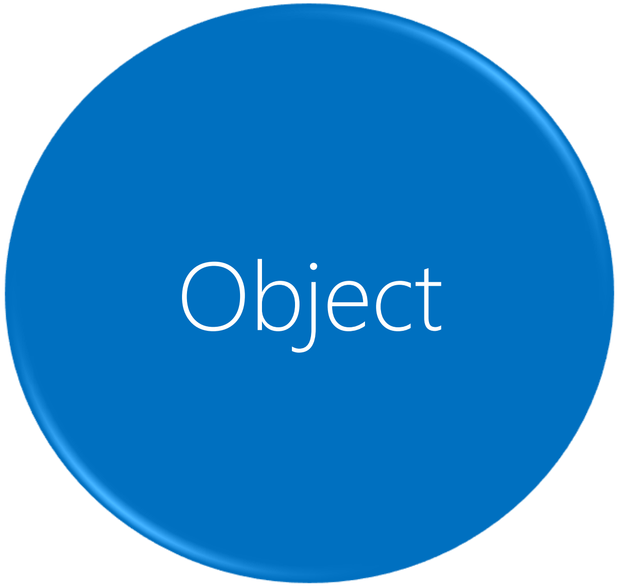


Object

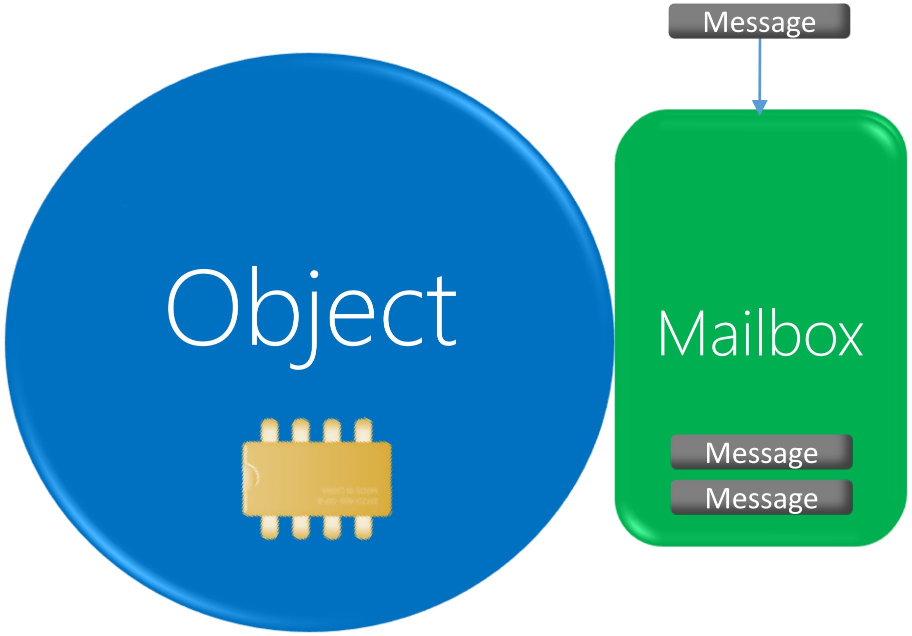
AKKA

Actor

Java

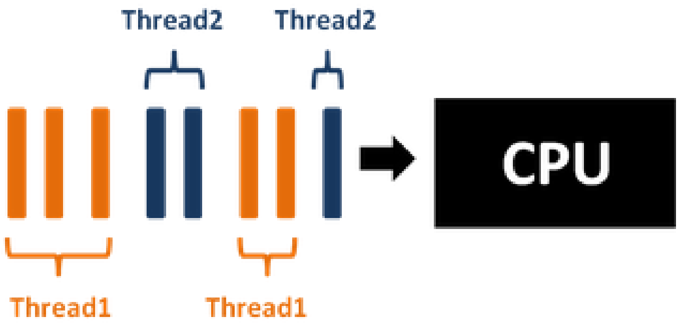


AKKA



Concurrent programming

Concurrent



Thread2

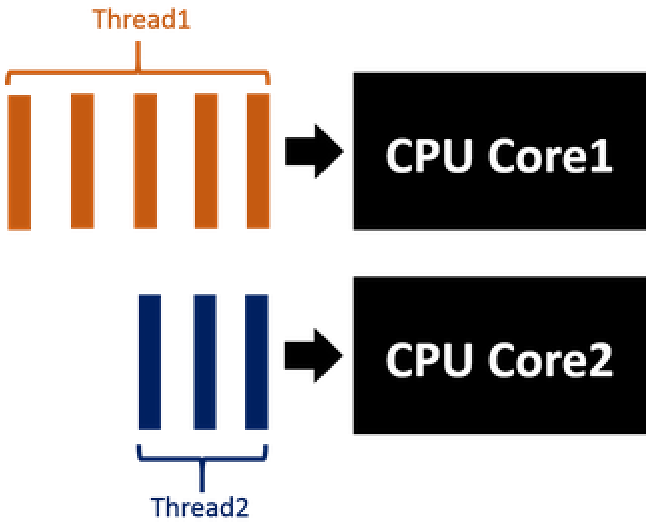
Thread2

CPU

Thread1

Thread1

Parallel



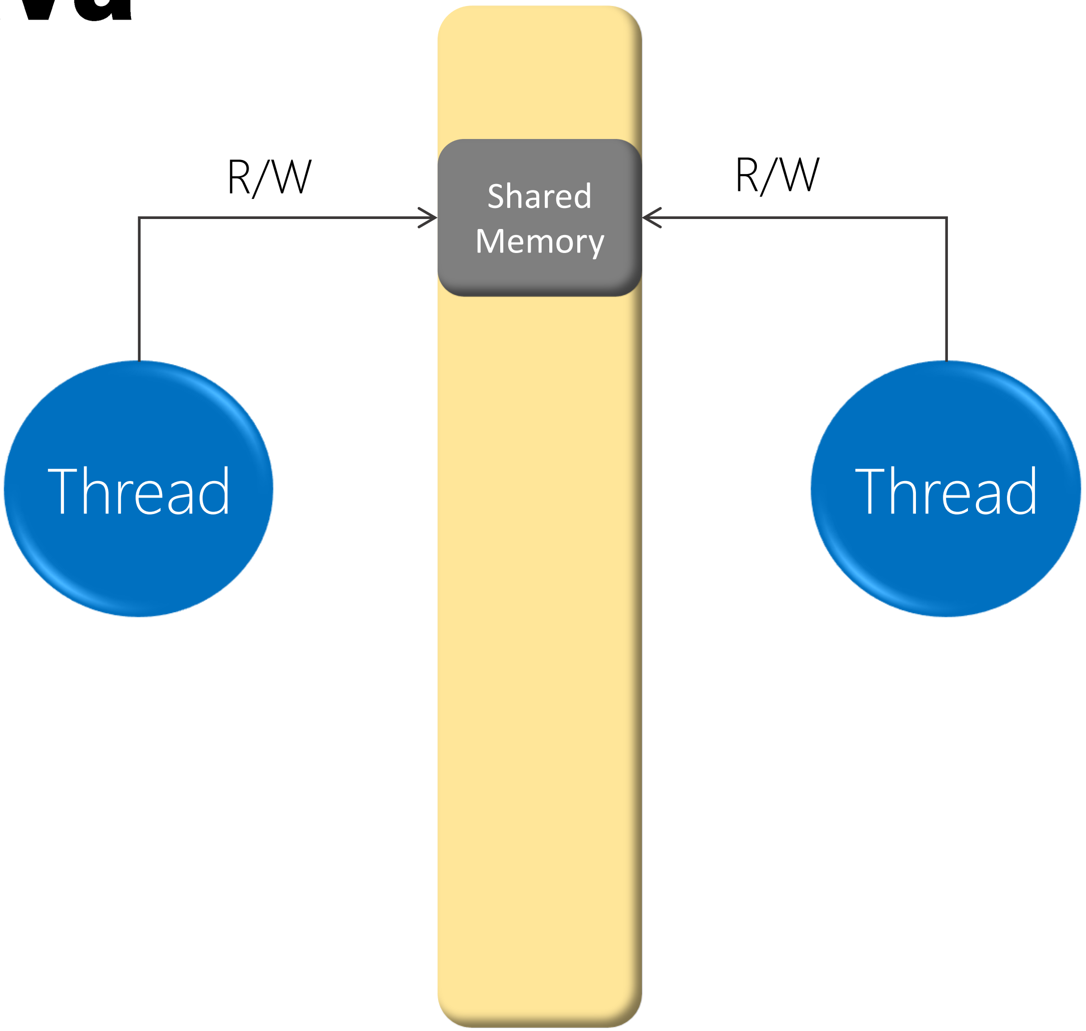
Thread1

CPU Core1

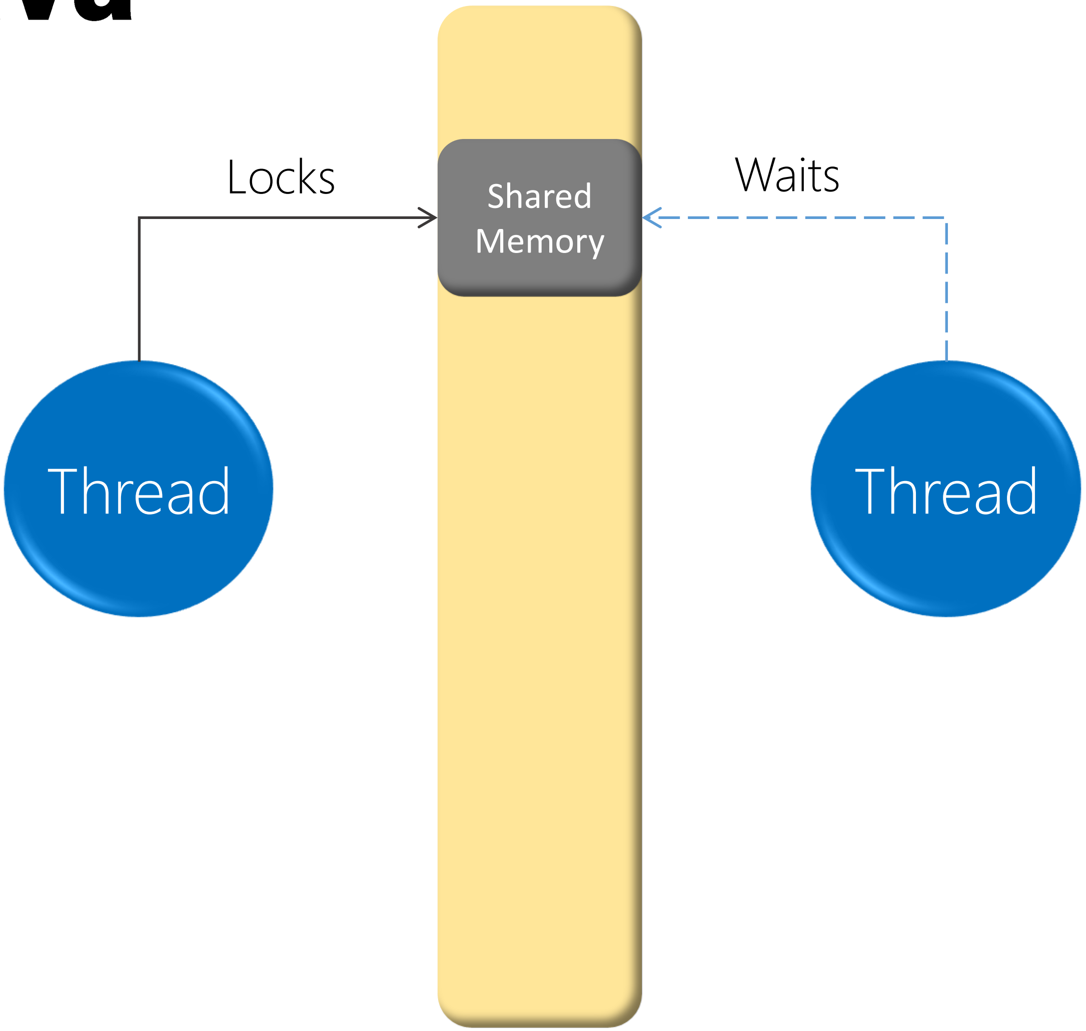
CPU Core2

Thread2

Java

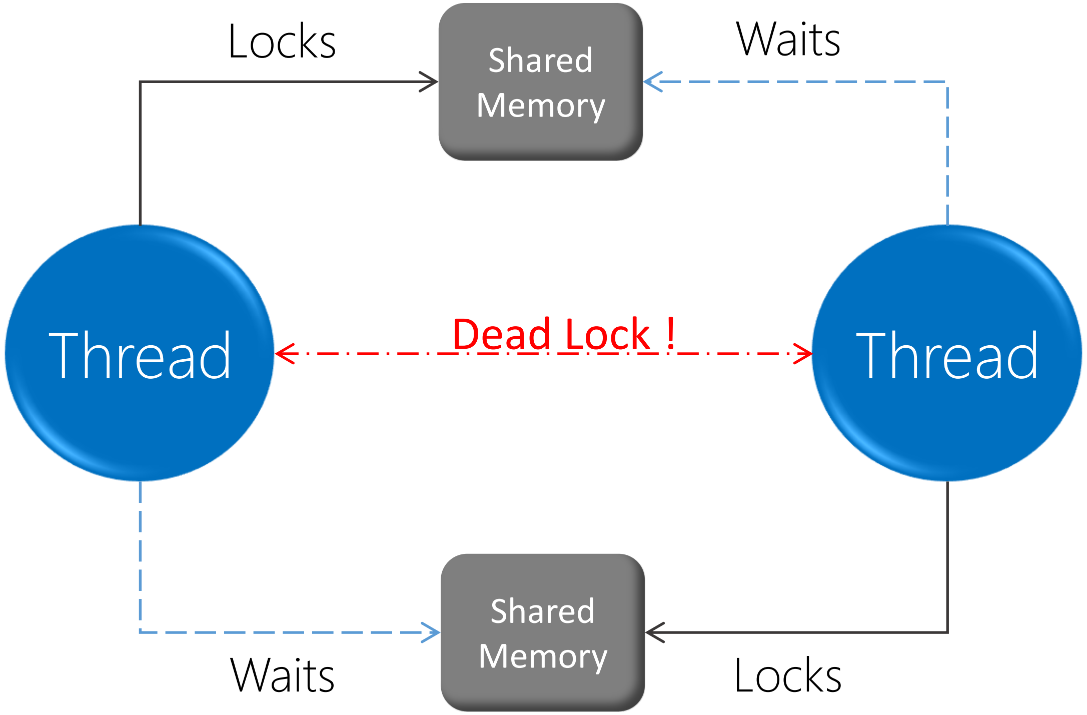


Java

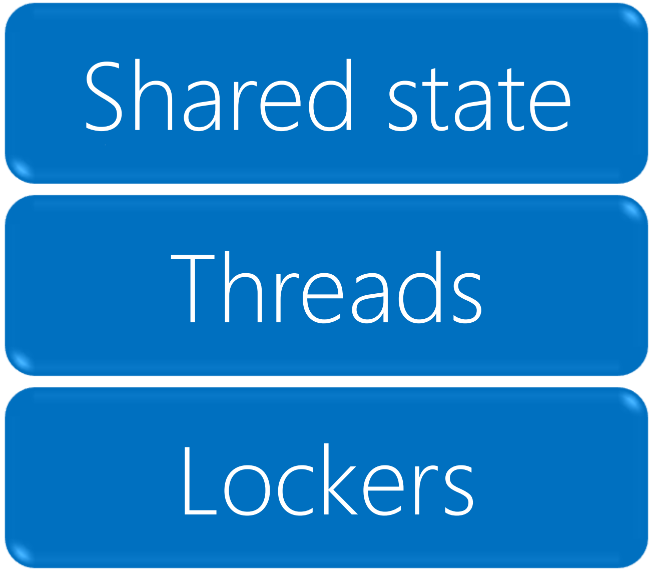


Java

How to avoid?



Java

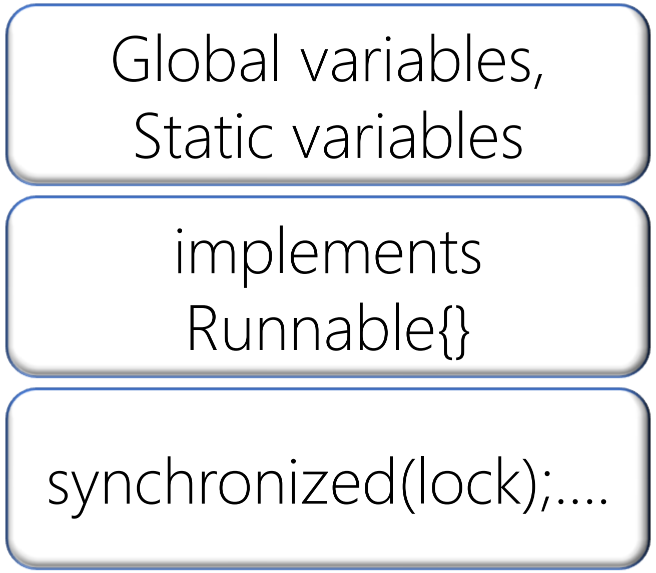


Shared state

Threads

Lockers

Java

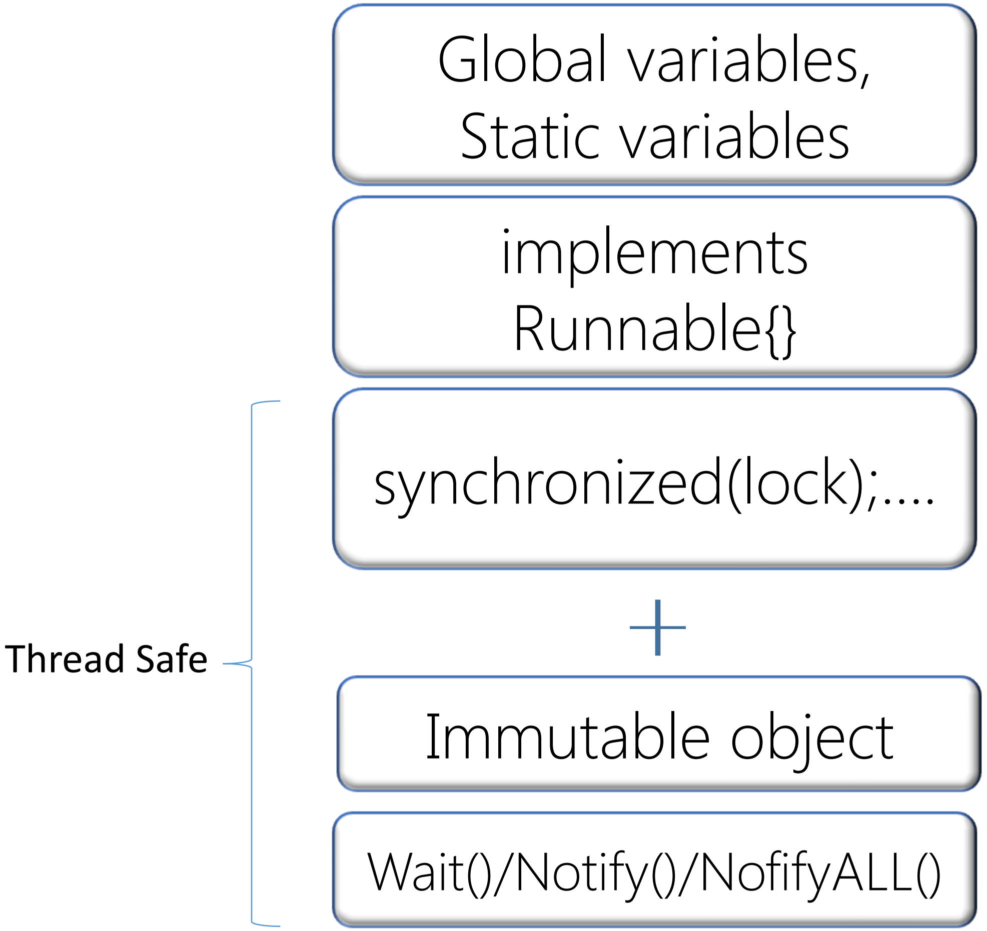


Global variables, Static variables

implements Runnable{}

synchronized(lock);....

Java



Global variables,
Static variables

implements
Runnable{}

synchronized(lock);….

+

Immutable object

Wait()/Notify()/NofifyALL()

Thread Safe

Java

AKKA

Global variables,
Static variables

implements
Runnable{}

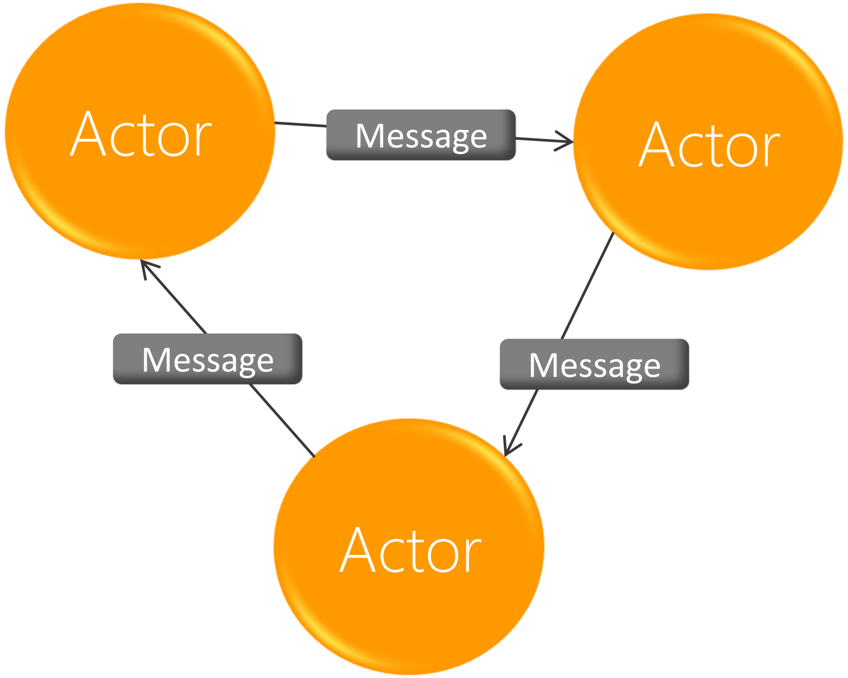
synchronized(lock);....

+

Immutable object

Wait()/Notify()/NofifyALL()

Thread Safe



Java

AKKA

Global variables,
Static variables

implements
Runnable{}

synchronized(lock);....

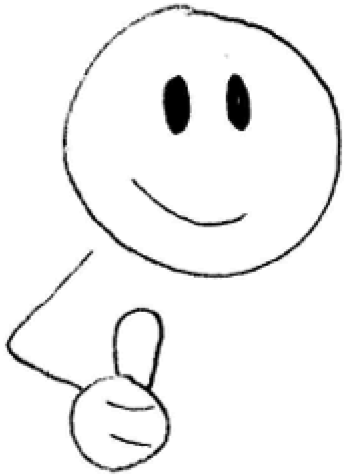
+

Immutable object

Wait()/Notify()/NofifyALL()

Thread Safe

Message flow

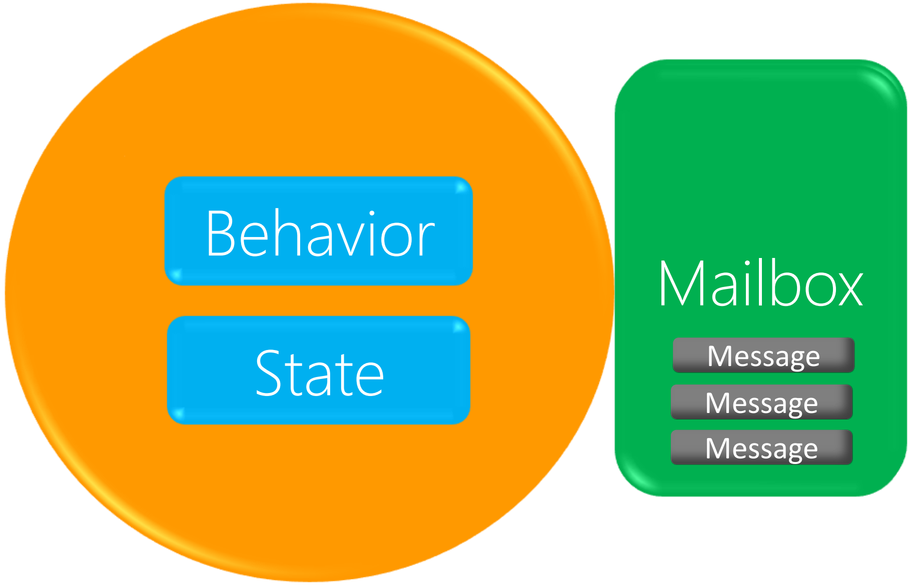


Actor

Actor having

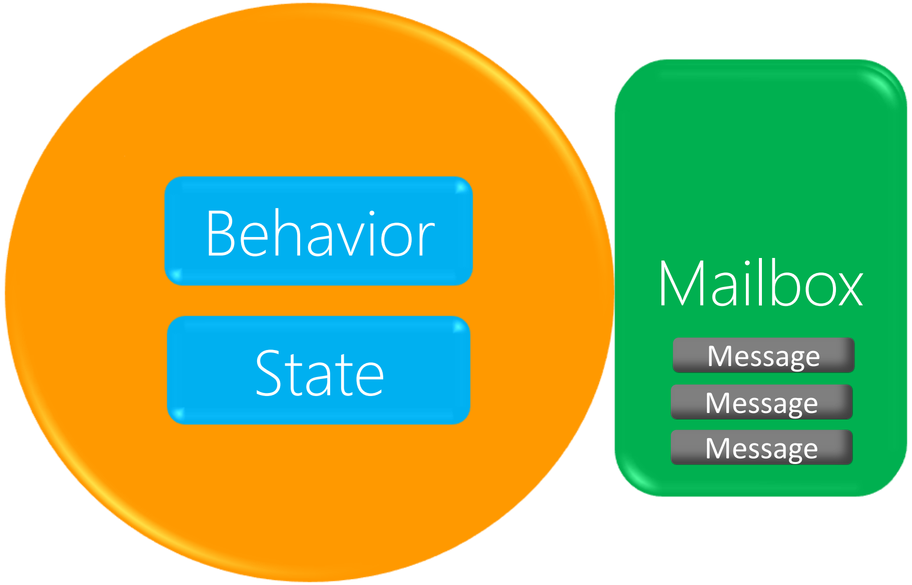
1. Behavior (Processing)
2. State (Storage)
3. Mailbox (Message Queue)

*State is not shared, only accessible through…messages passing.



Actor

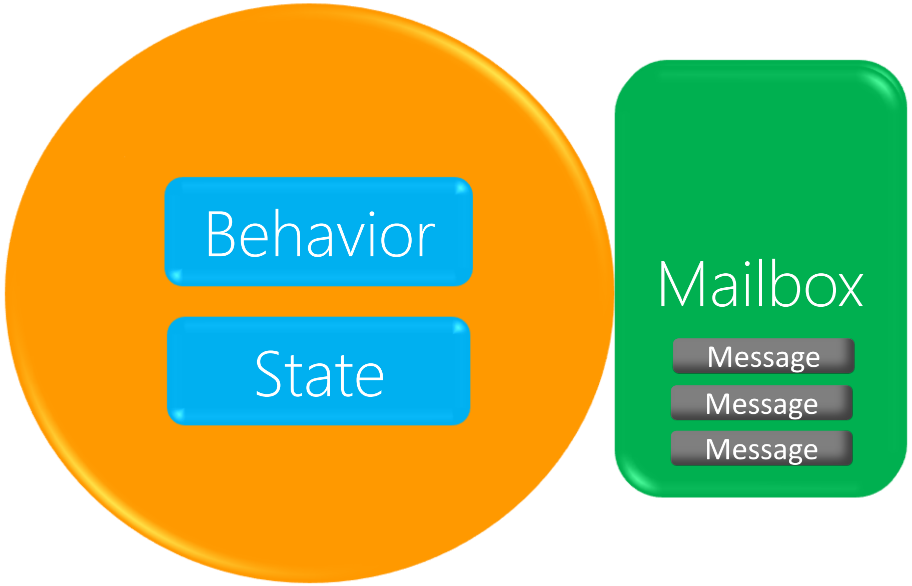
➔Messages are in mailbox.



Actor

➔Thread is allocated to the Actor.

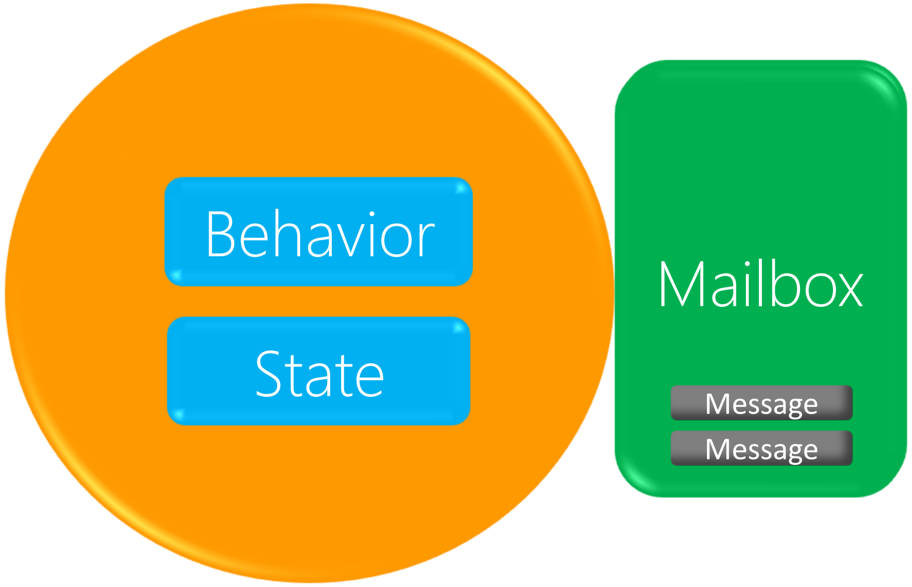
➔ It has read message and is applying behavior (in OnReceive() function).



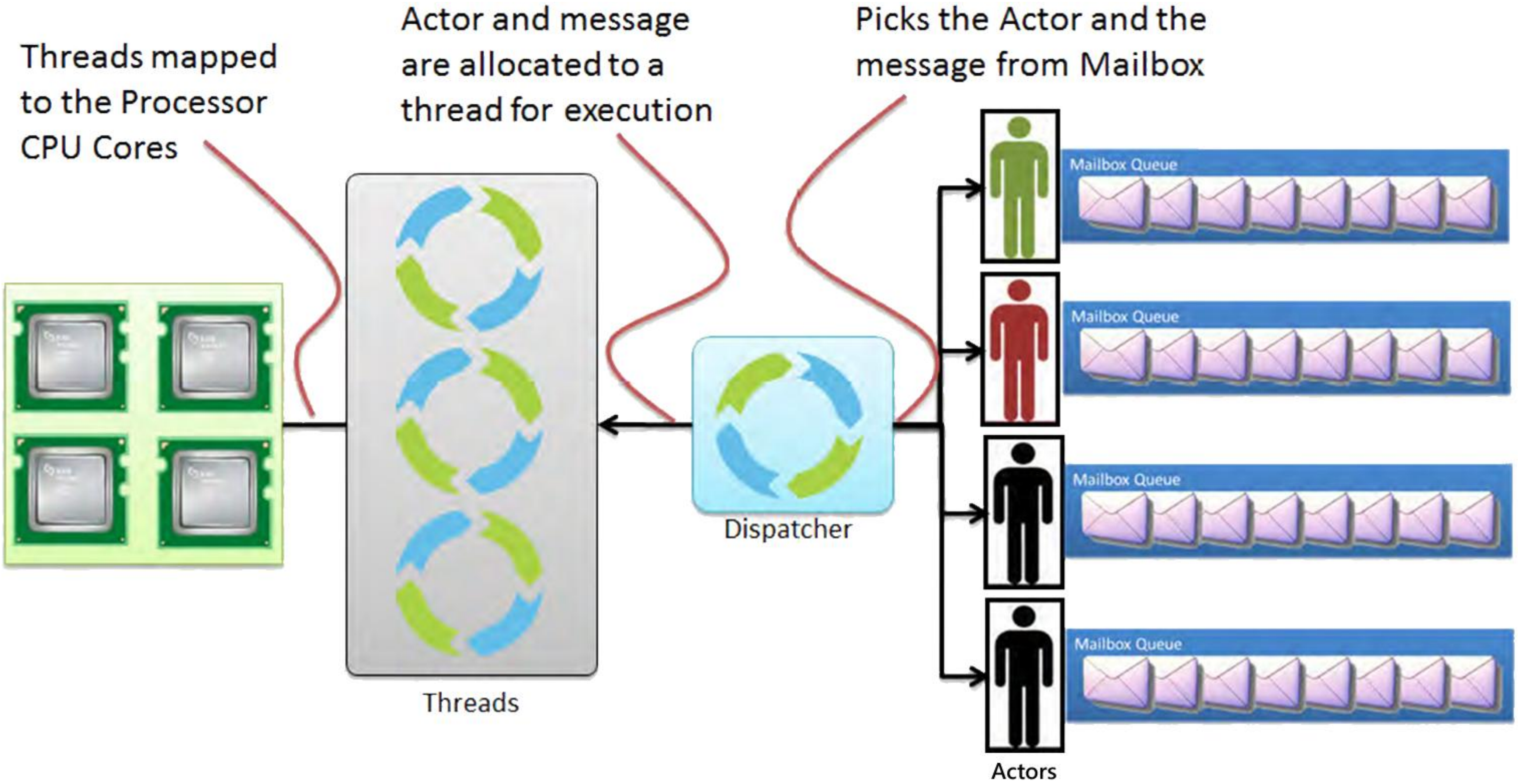
AKKA

➔ Actor has handled message*.
➔Thread is deallocated.

*One message is executed at a time
& messages are processed sequentially .



Dispatcher

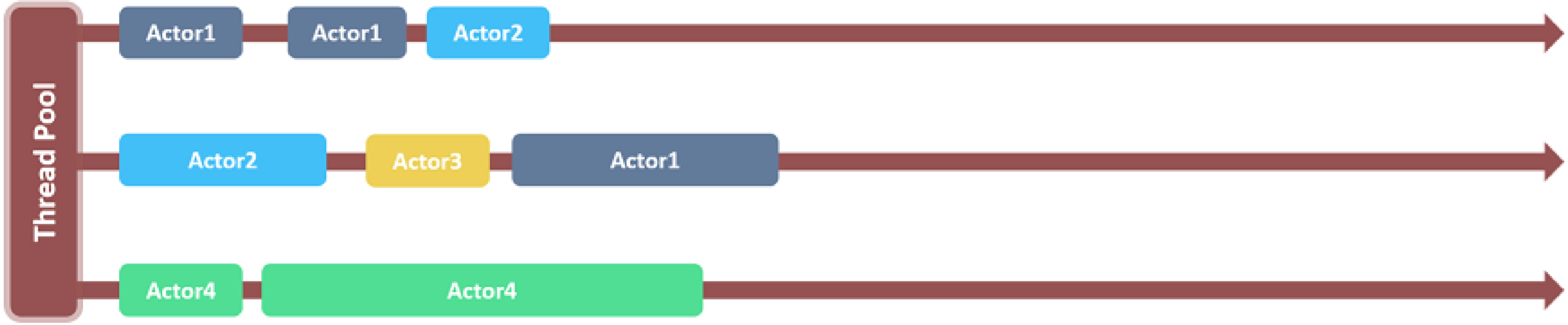


Threads mapped to the Processor CPU Cores

Actor and message are allocated to a thread for execution

Picks the Actor and the message from Mailbox

Dispatcher

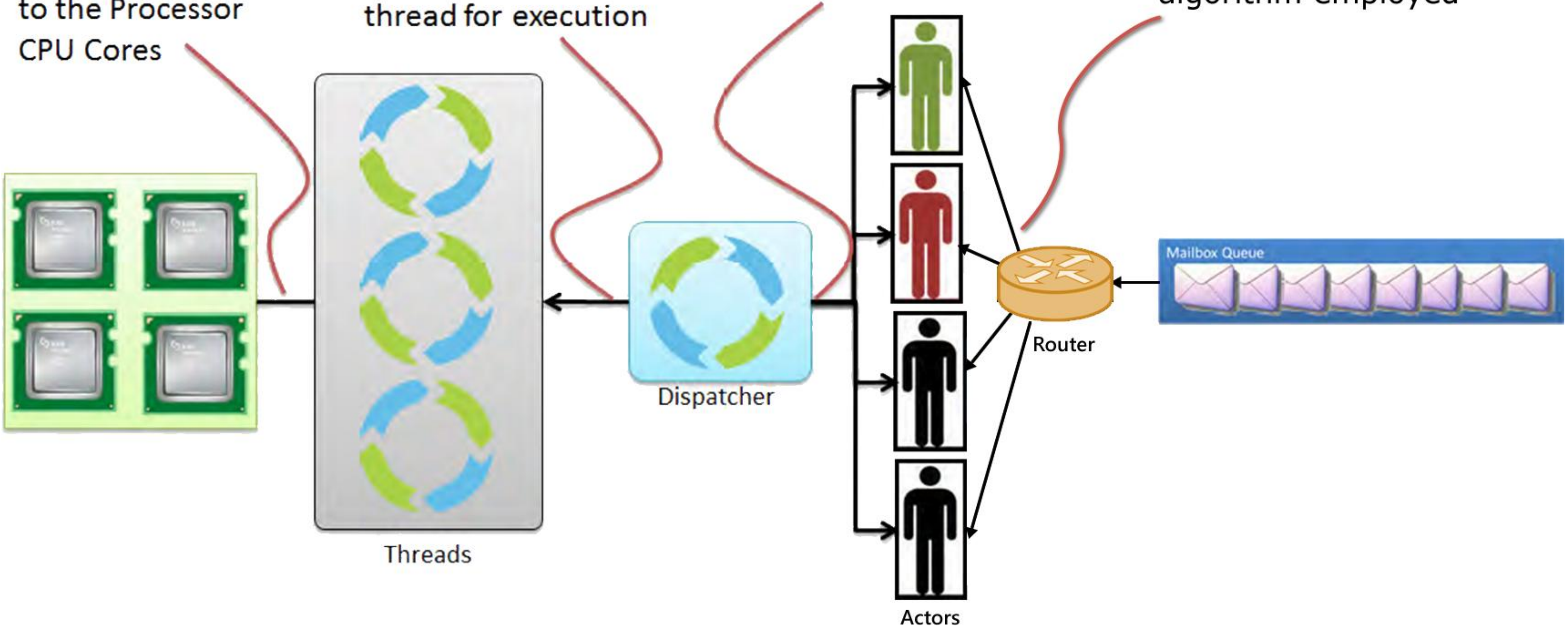

Thread Pool
Actor1
Actor1
Actor2
Actor2
Actor3
Actor1
Actor4
Actor4

4 types of dispatchers

1. Dispatcher (default)
2. Pinned dispatcher
3. Balancing dispatcher (Deprecated*)
4. Calling thread dispatcher

*Instead by BalancingPool of Router.

Router



Threads mapped to the Processor CPU Cores

Actor and message are allocated to a thread for execution

Picks the Actor and the message from Mailbox

Messages are routed to actors based on the algorithm employed

Router Types

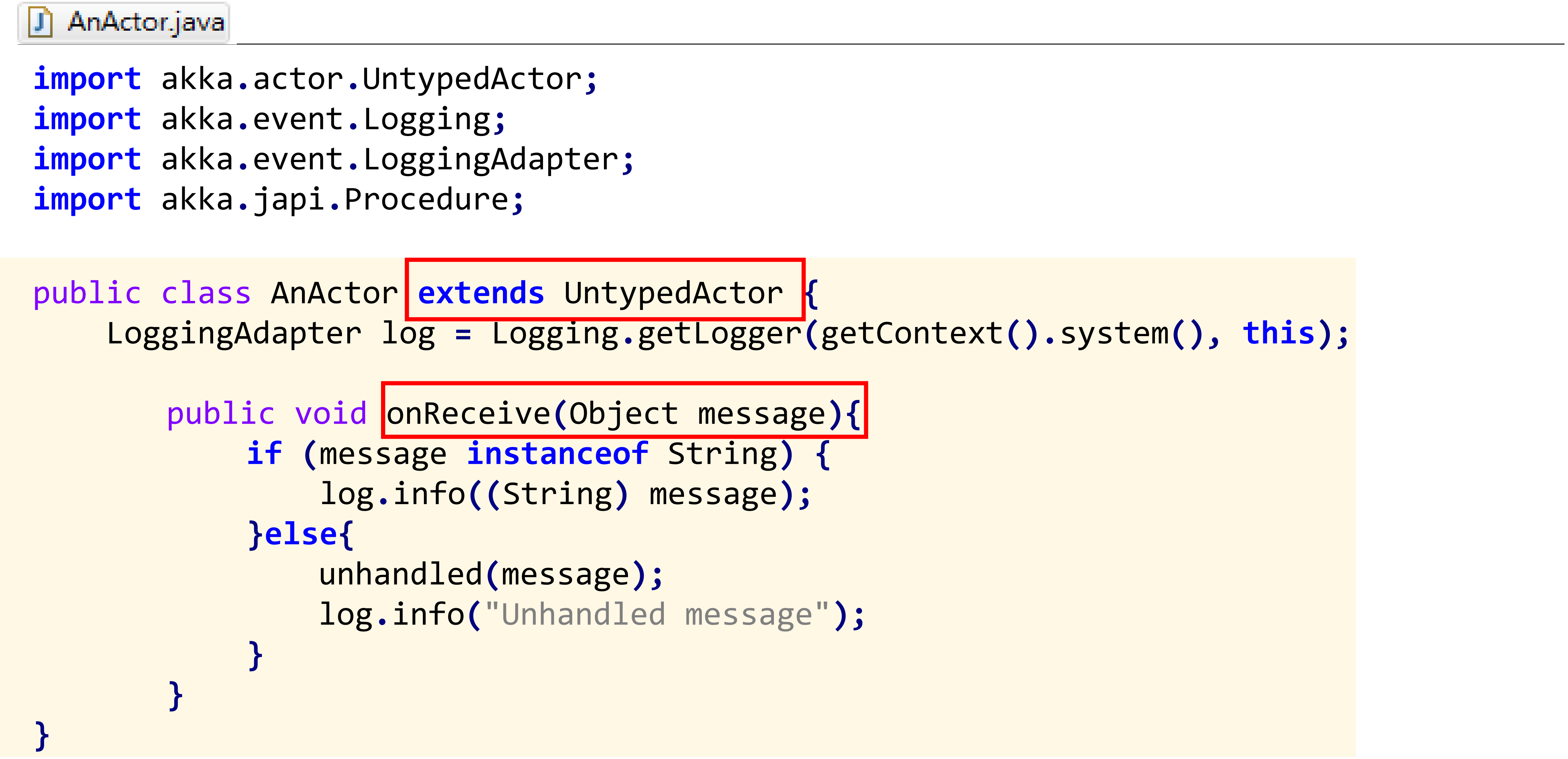
- RoundRobinPool & RoundRobinGroup
- RandomPool & RandomGroup
- BalancingPool- shared mailbox
- SmallestMailboxPool
- BroadcastPool & BroadcastGroup
- ScatterGatherFirstCompletedPool & ScatterGatherFirstCompletedGroup
- TailChoppingPool & TailChoppingGroup
- ConsistentHashingPool & ConsistentHashingGroup

2.Core Operations

5 Core Actor Operations

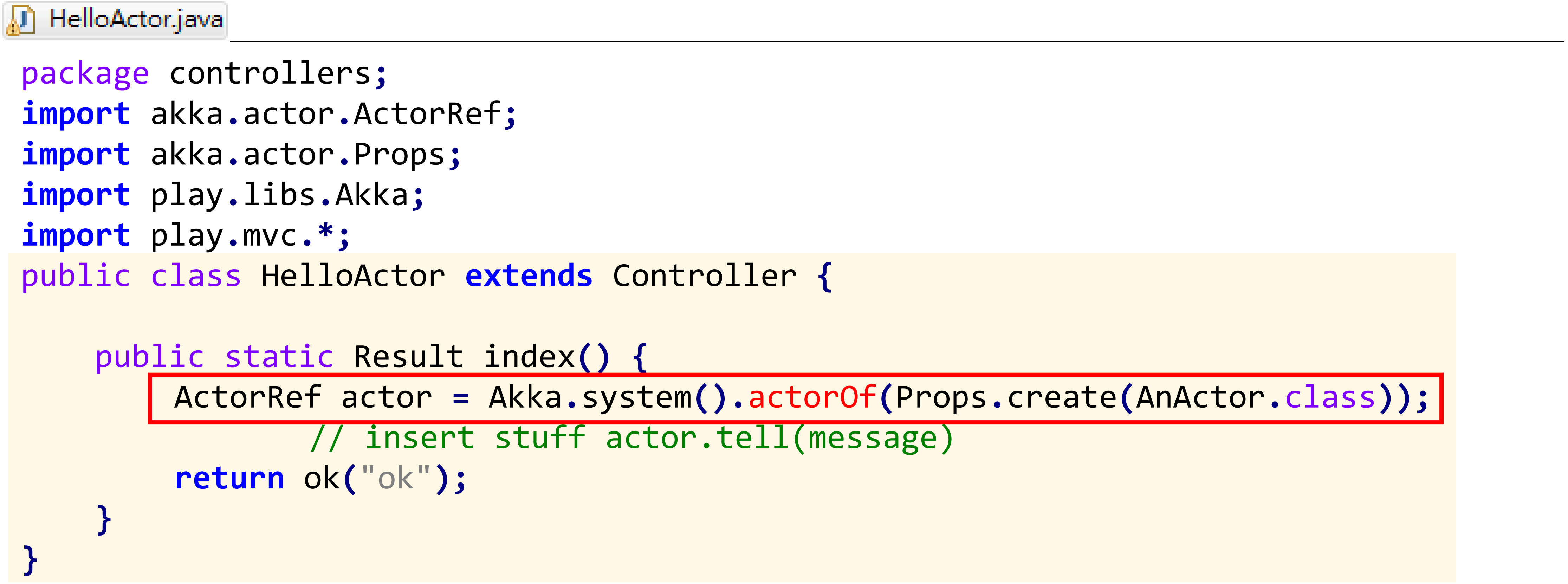
0. Define → Define Actors
1. Create → Create new Actors
2. Send → Send messages to other Actors
3. Become → Change the behavior for handling the next message
4. Supervise → Manage another Actors failure

0.DEFINE



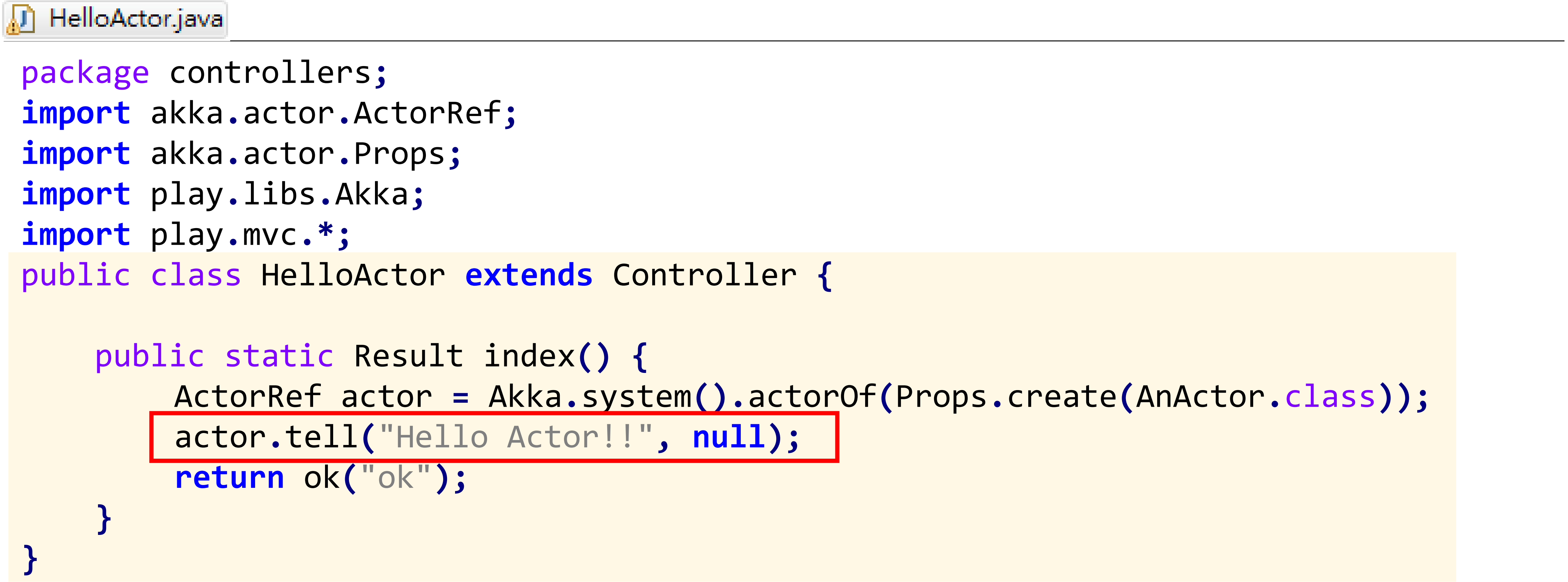
```java
import akka.actor.UntypedActor;
import akka.event.Logging;
import akka.event.LoggingAdapter;
import akka.japi.Procedure;

public class AnActor extends UntypedActor {
    LoggingAdapter log = Logging.getLogger(getContext().system(), this);

        public void onReceive(Object message){
            if (message instanceof String) {
                log.info((String) message);
            }else{
                unhandled(message);
                log.info("Unhandled message");
            }
        }
}
```

1.CREATE



```
package controllers;
import akka.actor.ActorRef;
import akka.actor.Props;
import play.libs.Akka;
import play.mvc.*;
public class HelloActor extends Controller {

    public static Result index() {
        ActorRef actor = Akka.system().actorOf(Props.create(AnActor.class));
                // insert stuff actor.tell(message)
        return ok("ok");
    }
}
```

2.SEND



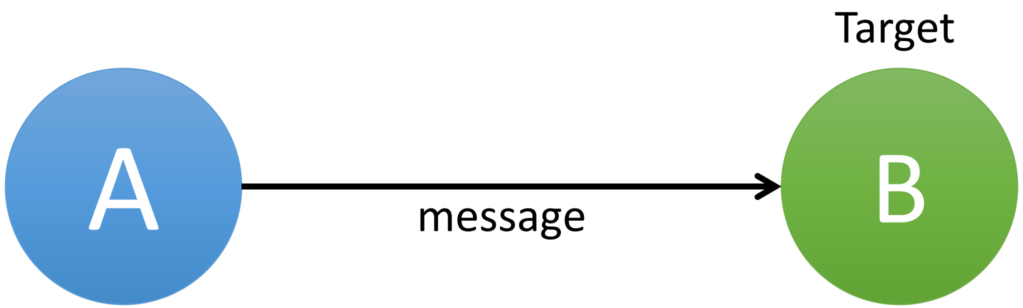
```java
package controllers;
import akka.actor.ActorRef;
import akka.actor.Props;
import play.libs.Akka;
import play.mvc.*;
public class HelloActor extends Controller {

    public static Result index() {
        ActorRef actor = Akka.system().actorOf(Props.create(AnActor.class));
        actor.tell("Hello Actor!!", null);
        return ok("ok");
    }
}
```

```
[INFO] [03/13/2015 22:14:01.442] [application-akka.actor.default-dispatcher-2] [akka://application/user/$a] Hello Actor!!
```

3 ways to sending messages

1. Fire-Forget → Tell
2. Ask and Reply → Ask
3. Forward → Forward

Tell



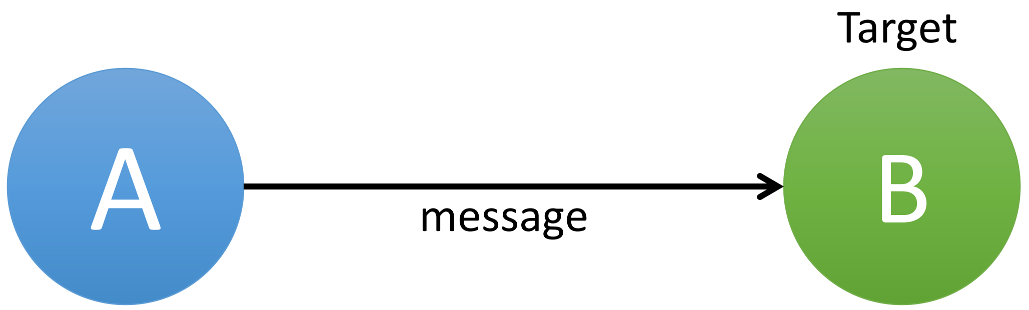
```
Target.tell(message, sender);
```

ActorRef Object ActorRef

1. To send a message to an actor, you need a Actor reference
2. Asynchronous and Non-blocking (Fire-and-forget)

1. null
2. ActorRef.noSender()
3. getSelf()
4. …

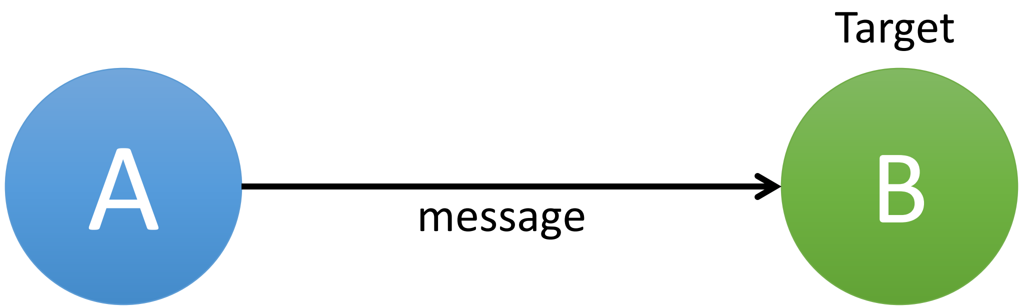
Tell



```
Target.tell(message, sender);
```

```
public void onReceive(Object message){
     if (message instanceof String) {
        log.info((String) message);
        log.info("getSender()="+getSender());
     }
   }
```

Tell



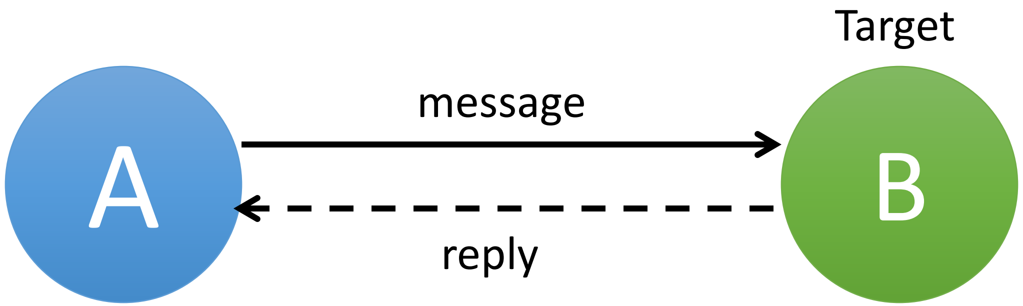
```
Target.tell(message, sender);
```

ActorRef Object ActorRef

EXAMPLE:

```
B.tell(“Hello Actor”,ActorRef.noSender());
B.tell(new Person(“David”,”Chang”),getSelf());
```

Ask



1

```
Future<Object> rt = Patterns.ask(Target, message, timeout);
```

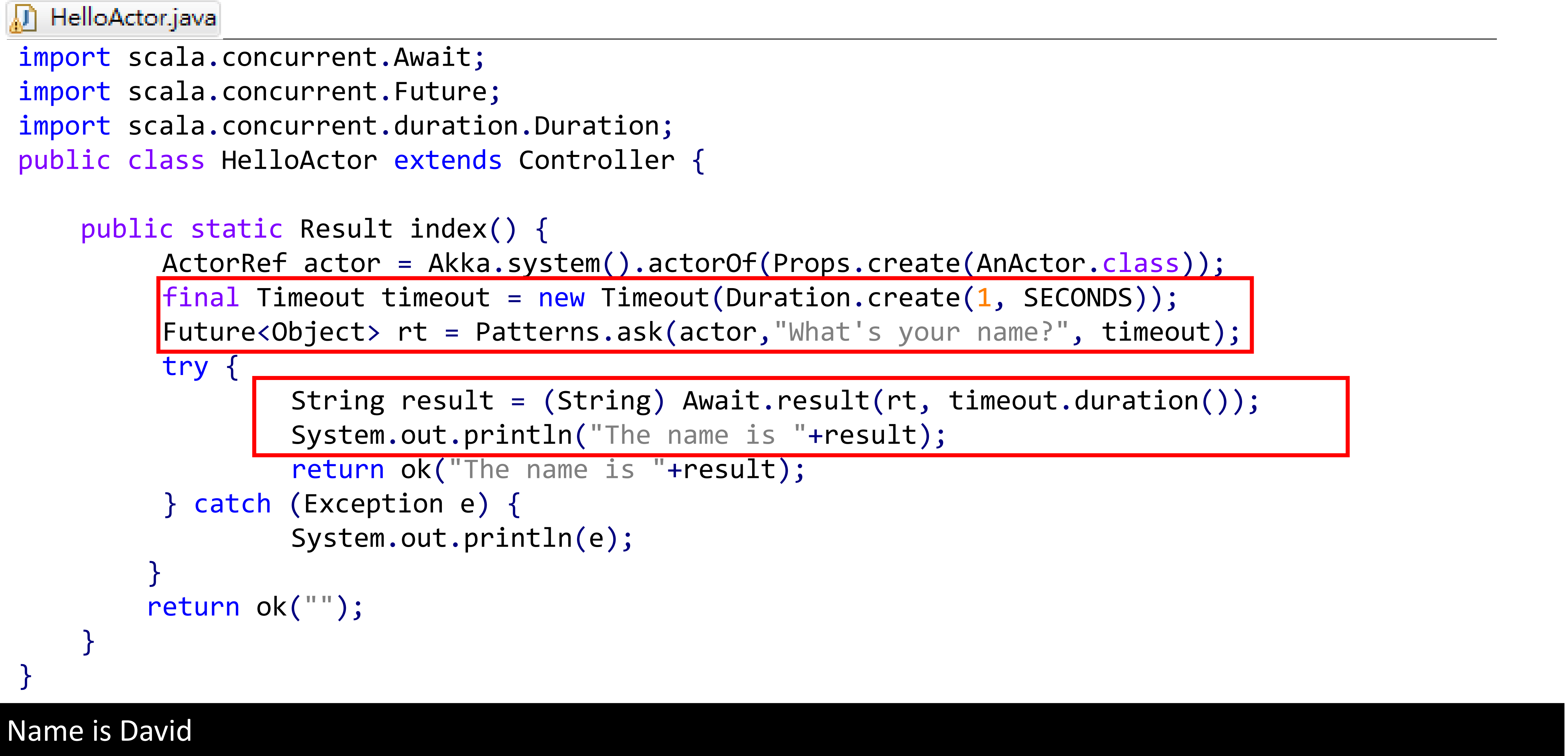
2

```
getSender().tell(reply_message, getSelf());
```

3

```
String result = Await.result(rt , timeout.duration);
```

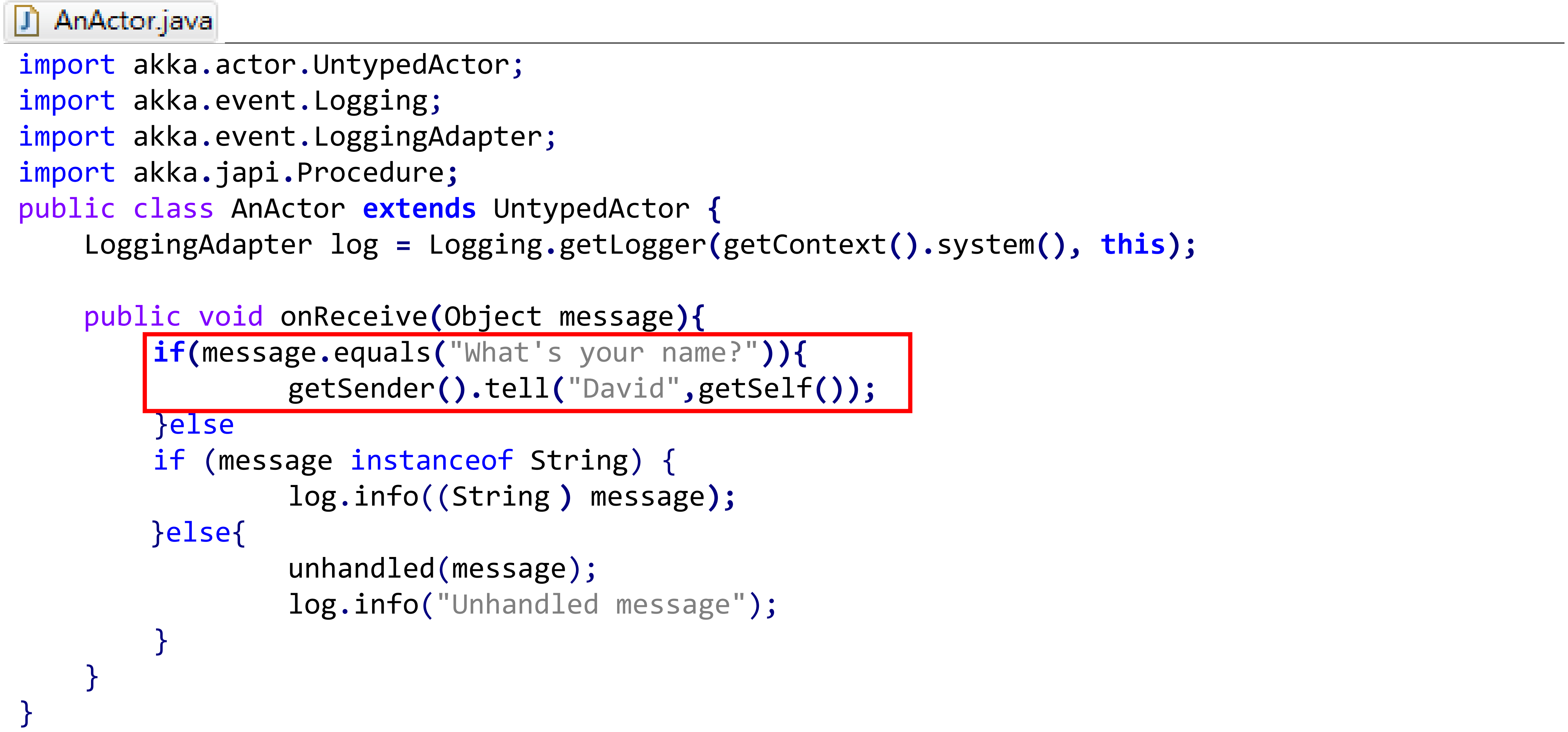
Ask



```java
import scala.concurrent.Await;
import scala.concurrent.Future;
import scala.concurrent.duration.Duration;
public class HelloActor extends Controller {

    public static Result index() {
         ActorRef actor = Akka.system().actorOf(Props.create(AnActor.class));
         final Timeout timeout = new Timeout(Duration.create(1, SECONDS));
         Future<Object> rt = Patterns.ask(actor,"What's your name?", timeout);
         try {
                 String result = (String) Await.result(rt, timeout.duration());
                 System.out.println("The name is "+result);
                 return ok("The name is "+result);
         } catch (Exception e) {
                 System.out.println(e);
        }
        return ok("");
    }
}
```

```
The Name is David
```

Ask

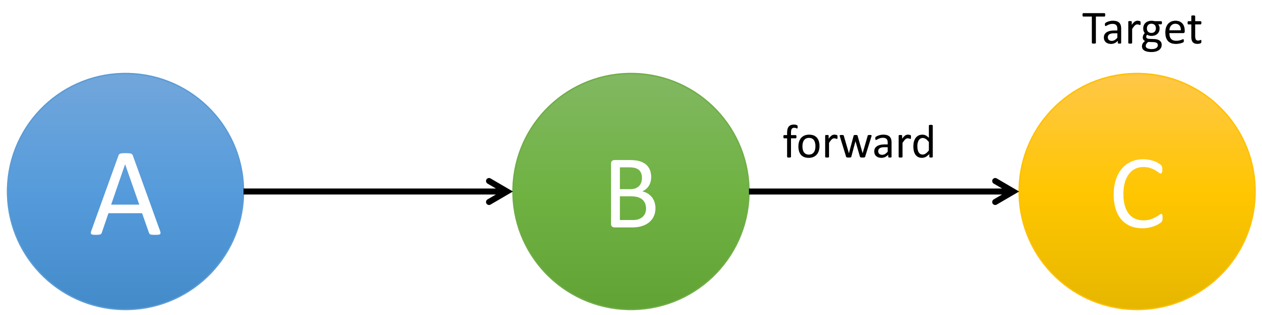


AnActor.java

```
import akka.actor.UntypedActor;
import akka.event.Logging;
import akka.event.LoggingAdapter;
import akka.japi.Procedure;
public class AnActor extends UntypedActor {
    LoggingAdapter log = Logging.getLogger(getContext().system(), this);

    public void onReceive(Object message){
        if(message.equals("What's your name?")){
                getSender().tell("David",getSelf());
        }else
        if (message instanceof String) {
                log.info((String ) message);
        }else{
                unhandled(message);
                log.info("Unhandled message");
        }
    }
}
```

The Name is David

Forward



```
Target.forward(message, getContext());
```

ActorContext

‖

```
Target.tell(message, getSender());
```

ActorRef

3.BECOME

```
getContext().become(Procedure<Object>);
```

1. Dynamically redefines actor behavior
2. Reactively triggered by message
3. Behaviors are stacked & can be pushed and popped

→ `getContext().unbecome();`

BECOME

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

```
public class HotSwapActor extends UntypedActor {
  Procedure<Object> angry = new Procedure<Object>() {
    @Override
    public void apply(Object message) {
      if (message.equals(“work")) {
        getSender().tell(“I am angry ☹",getSelf());
      } else if (message.equals(“play")) {
        getContext().become(happy);
      }
    }
  };
  Procedure<Object> happy = new Procedure<Object>() {
    @Override
    public void apply(Object message) {
      if (message.equals(“play")) {
        getSender().tell("I am happy ☺", getSelf());
      } else if (message.equals(“work")) {
        getContext().become(angry);
      }
    }
  };
  public void onReceive(Object message) {
    if (message.equals(“work")) {
      getContext().become(angry);
    } else if (message.equals(“play")) {
      getContext().become(happy);
    } else {
      unhandled(message);
    }
  }
}
```

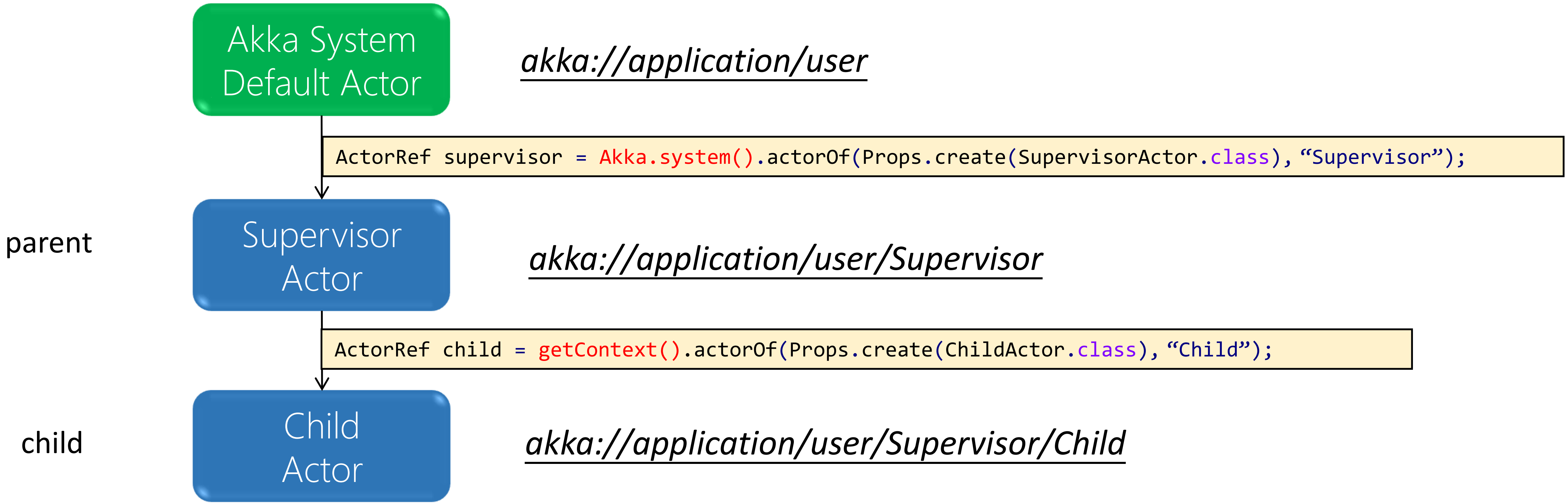
BECOME

```
public class HotSwapActor extends UntypedActor {
  Procedure<Object> angry = new Procedure<Object>() {
    @Override
    public void apply(Object message) {
      if (message.equals(“work")) {
        getSender().tell(“I am angry ☹",getSelf());
      } else if (message.equals(“play")) {
        getContext().become(happy);
      }
    }
  };
  Procedure<Object> happy = new Procedure<Object>() {
    @Override
    public void apply(Object message) {
      if (message.equals(“play")) {
        getSender().tell("I am happy ☺", getSelf());
      } else if (message.equals(“work")) {
        getContext().become(angry);
      }
    }
  };
  public void onReceive(Object message) {
    if (message.equals(“work")) {
      getContext().become(angry);
    } else if (message.equals(“play")) {
      getContext().become(happy);
    } else {
      unhandled(message);
    }
  }
}
```

```
Procedure<Object> angry = new Procedure<Object>() {
    @Override
    public void apply(Object message) {
      if (message.equals(“work")) {
        getSender().tell("I am angry ☹", getSelf());
      } else if (message.equals(“play")) {
        getContext().become(happy);
      }
    }
  };
```

Hierarchy

- Actors can form hierarchies

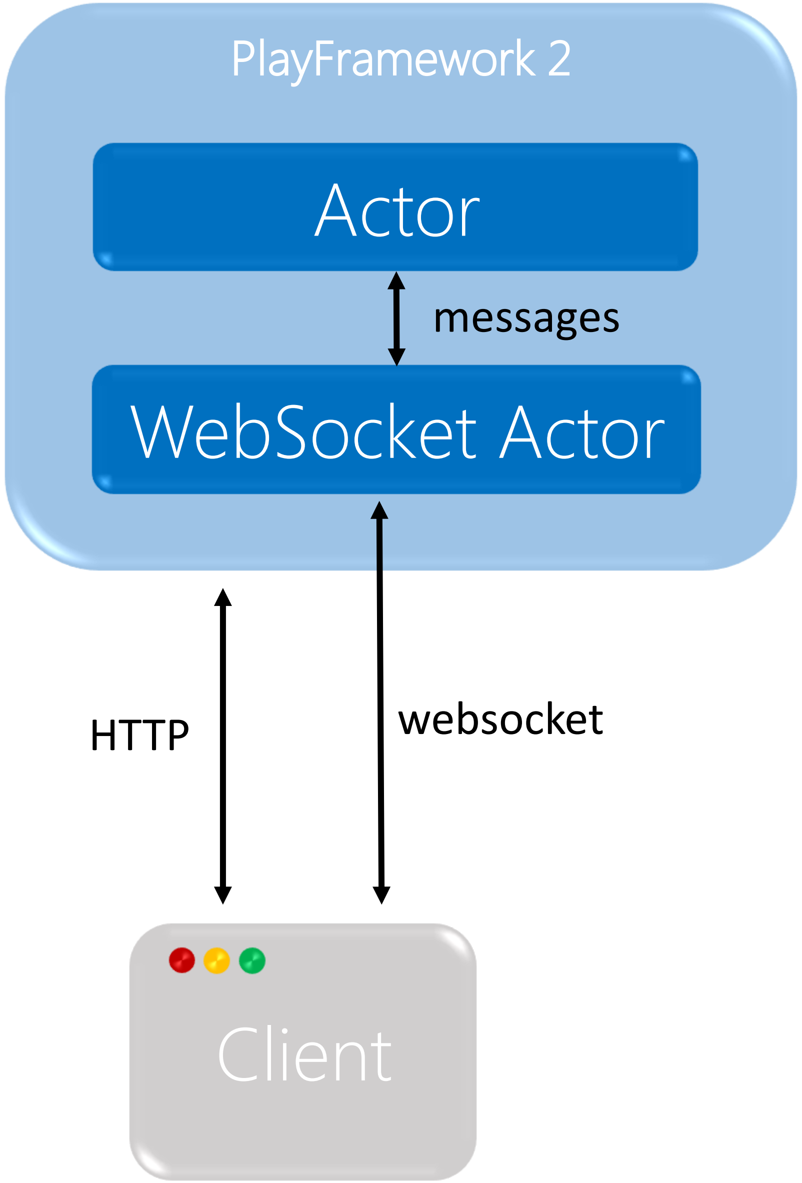

Akka System Default Actor
akka://application/user
parent
Supervisor Actor
akka://application/user/Supervisor
child
Child Actor
akka://application/user/Supervisor/Child

```
ActorRef supervisor = Akka.system().actorOf(Props.create(SupervisorActor.class), “Supervisor”);
```

```
ActorRef child = getContext().actorOf(Props.create(ChildActor.class), “Child”);
```

3.WebSocket with Actor

WebSocket with Actor



WebSocket with Actor

- **Controller**

```
import play.mvc.WebSocket;
public class Application extends Controller {

    public static WebSocket<JsonNode> chat(final String username)  {
       return WebSocket.withActor(new Function<ActorRef, Props>() {
           public Props apply(ActorRef out) throws Throwable {
                return ChatWebSocketActor.props(out, username);
           }
        });
    }
}
```

- **Routes**

```
GET    /room/chat                    controllers.Application.chat(username)
```

- **URL**

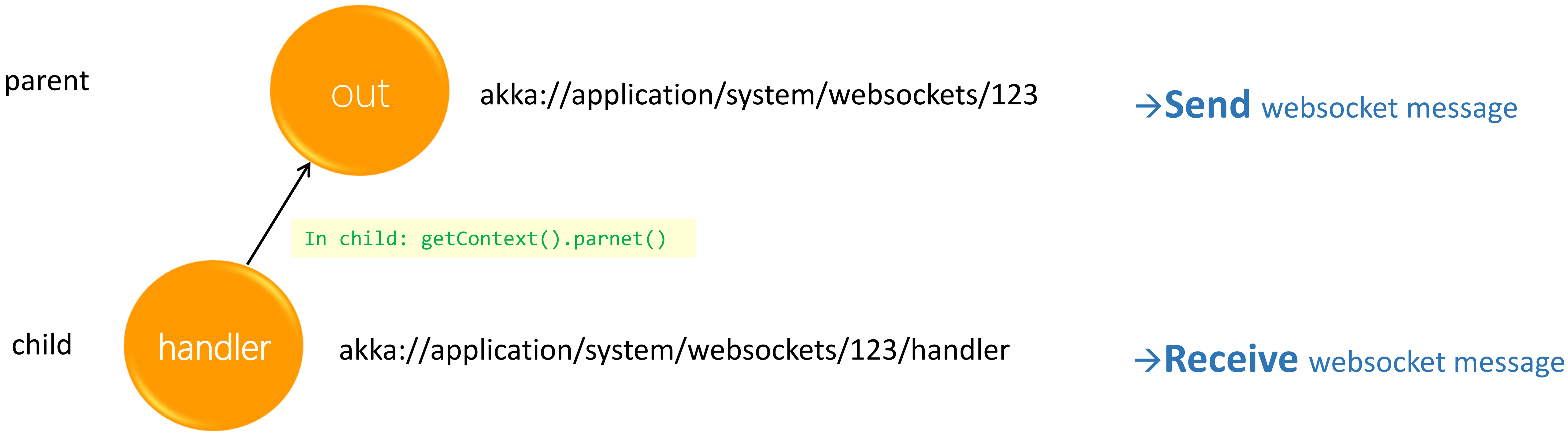
  ws://127.0.0.1:9000/room/chat?username=XXX

WebSocket with Actor

```
public class ChatWebSocketActor extends UntypedActor {
        LoggingAdapter log = Logging.getLogger(getContext().system(), this);

    public static Props props(ActorRef out, String username) {
        return Props.create(ChatWebSocketActor.class, out, username);
    }
    private final ActorRef out;
    private final String username;

    public ChatWebSocketActor(ActorRef out,String username) {
        this.out = out;
        this.username = username;
    }
    public void preStart(){
                //do something
    }
    public void onReceive(Object message) throws Exception {
                //do something
    }
    public void postStop() throws Exception {
                //do something
    }
}
```

Receive websocket message

WebSocket with Actor



parent

out

akka://application/system/websockets/123

→**Send** websocket message

`In child: getContext().parnet()`

child

handler

akka://application/system/websockets/123/handler

→**Receive** websocket message

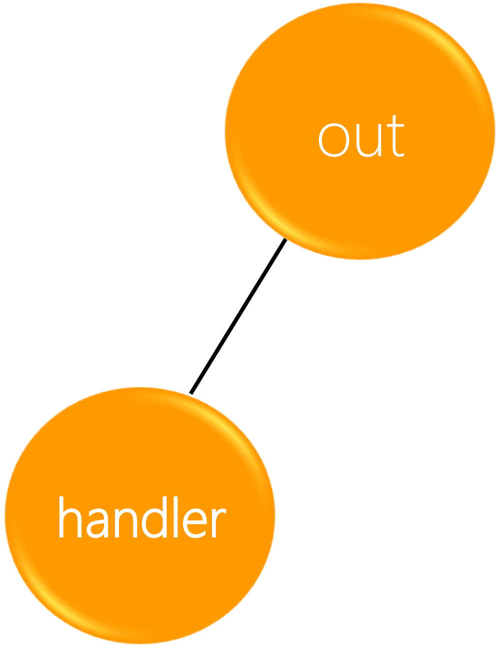
WebSocket with Actor

- Send websocket message to client

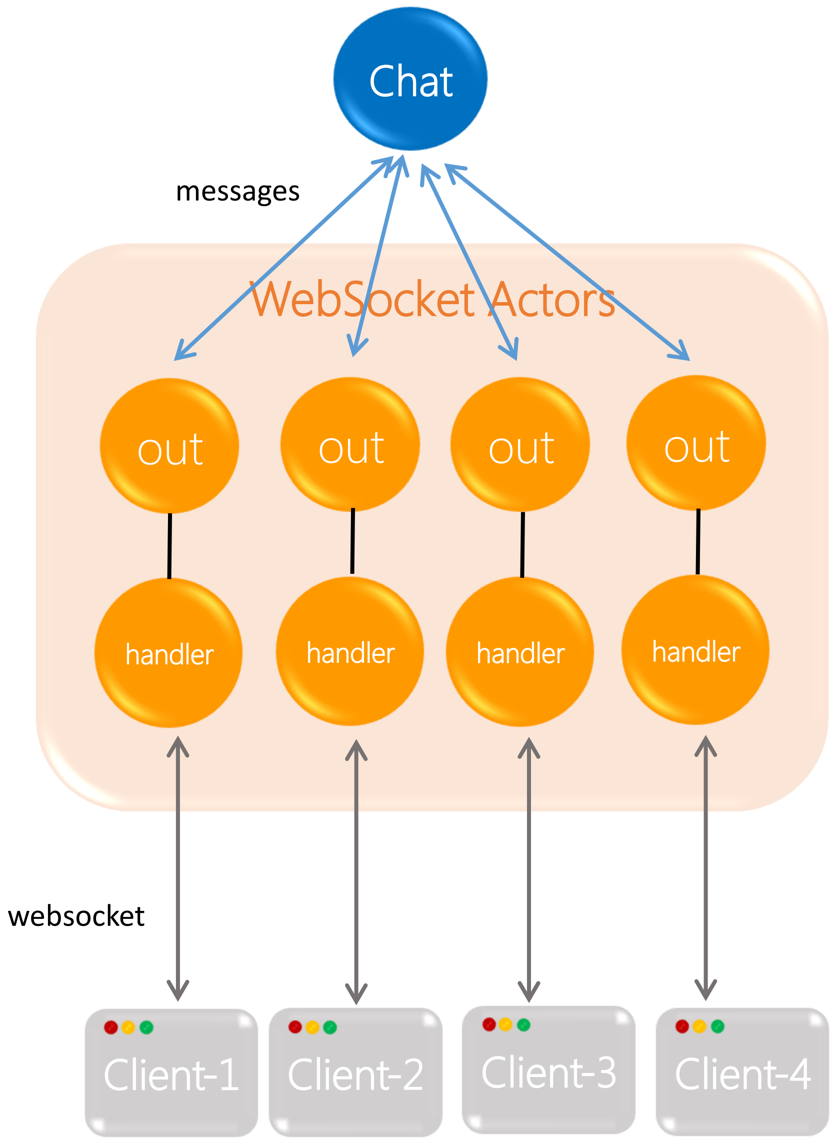
```
out.tell(message, null);
```

- Closing a websocket

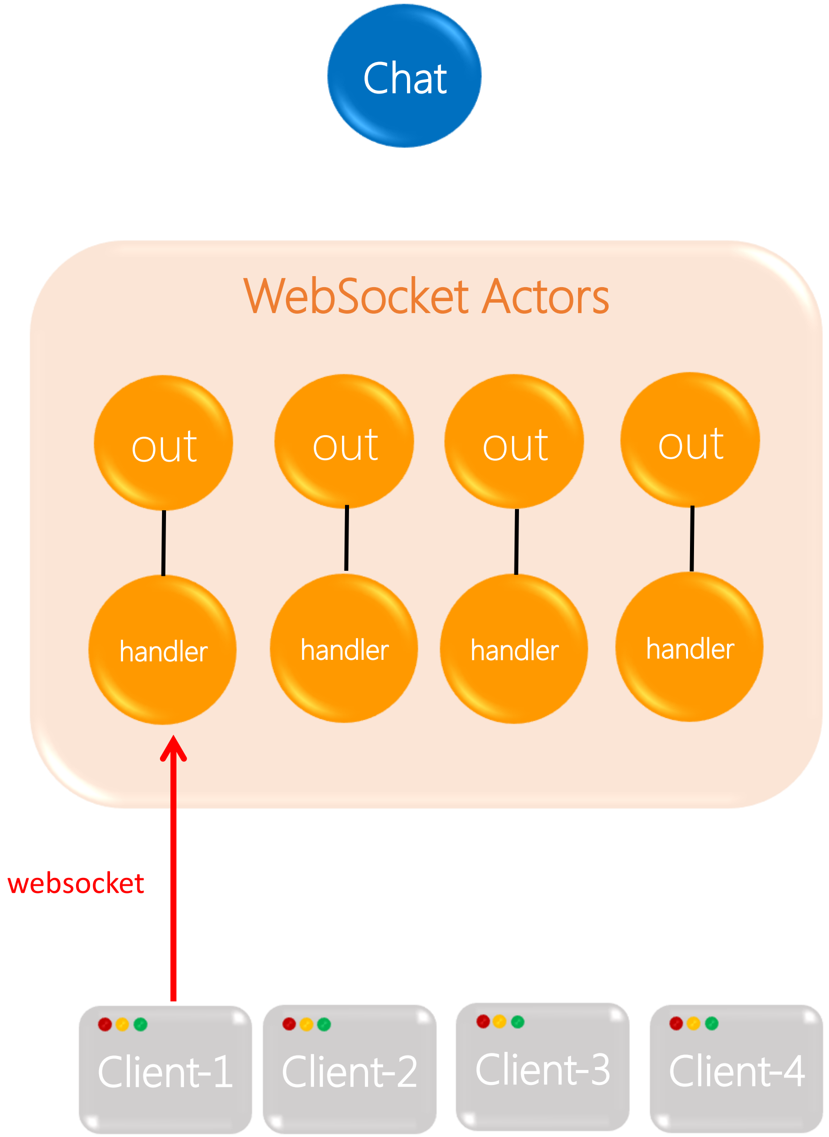
```
out.tell(PoisonPill.getInstance(), self());
```



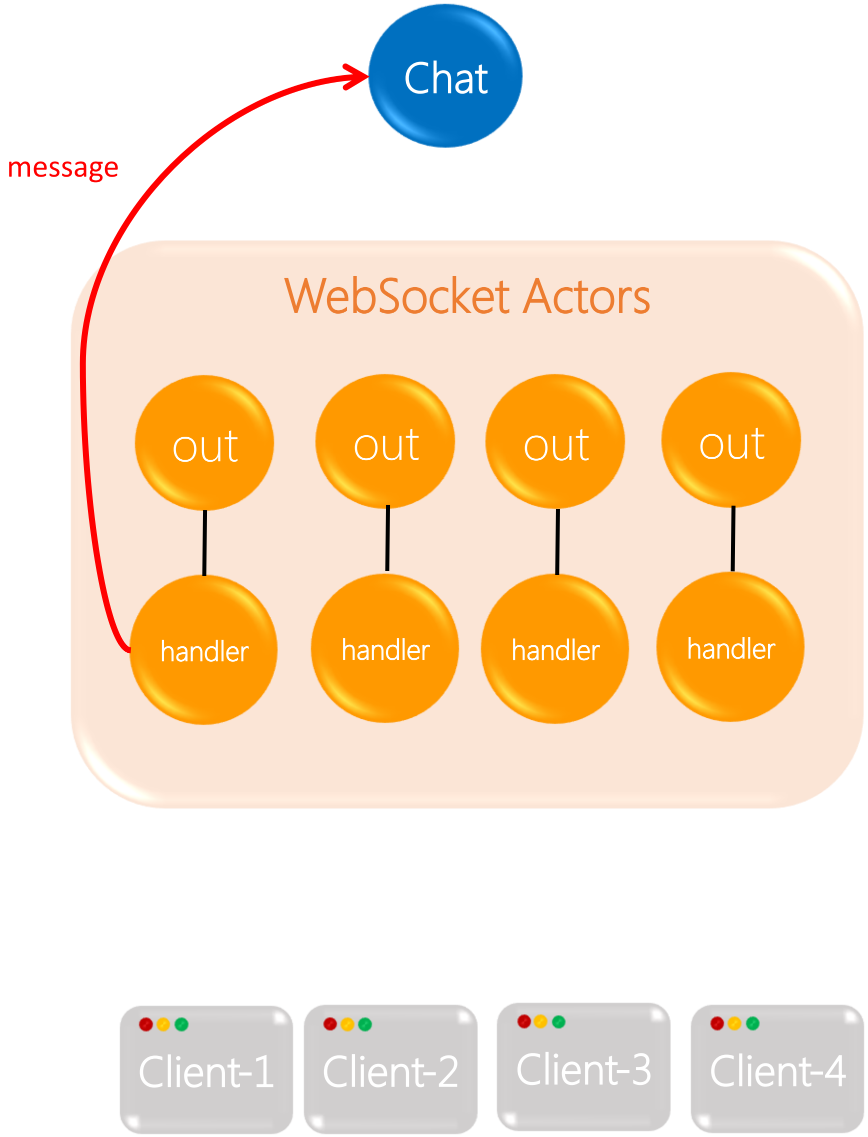
Chat Actor


Chat
messages
WebSocket Actors
out
out
out
out
handler
handler
handler
handler
websocket
Client-1
Client-2
Client-3
Client-4

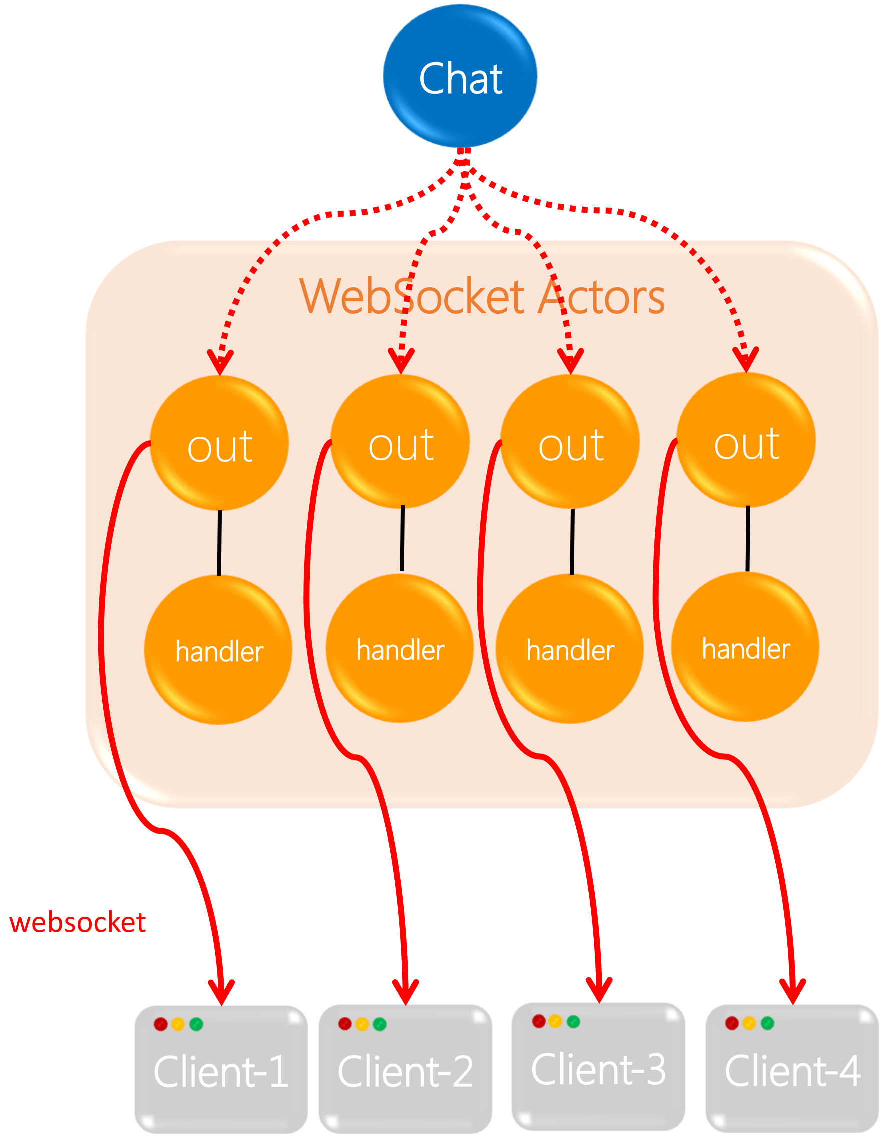
Chat flow – (1)


Chat
WebSocket Actors
out
out
out
out
handler
handler
handler
handler
websocket
Client-1
Client-2
Client-3
Client-4

Chat flow – (2)

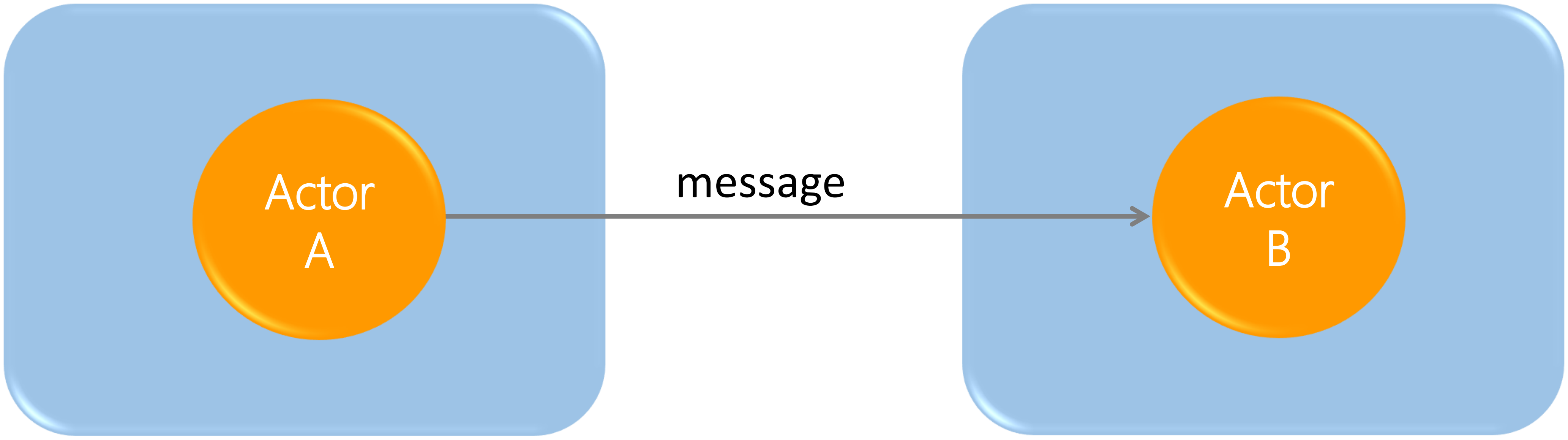

Chat
message
WebSocket Actors
out
out
out
out
handler
handler
handler
handler
Client-1
Client-2
Client-3
Client-4

Chat flow – (3)


Chat
WebSocket Actors
out
out
out
out
handler
handler
handler
handler
websocket
Client-1
Client-2
Client-3
Client-4

4.Remote Actor

Remote Actors



192.168.0.1

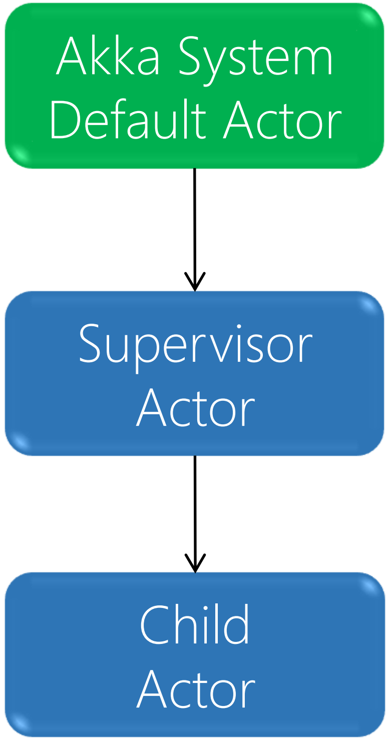
192.168.0.2

➔ akka.tcp://application@192.168.0.1/user/ActorA

➔ akka.tcp://application@192.168.0.2/user/ActorB

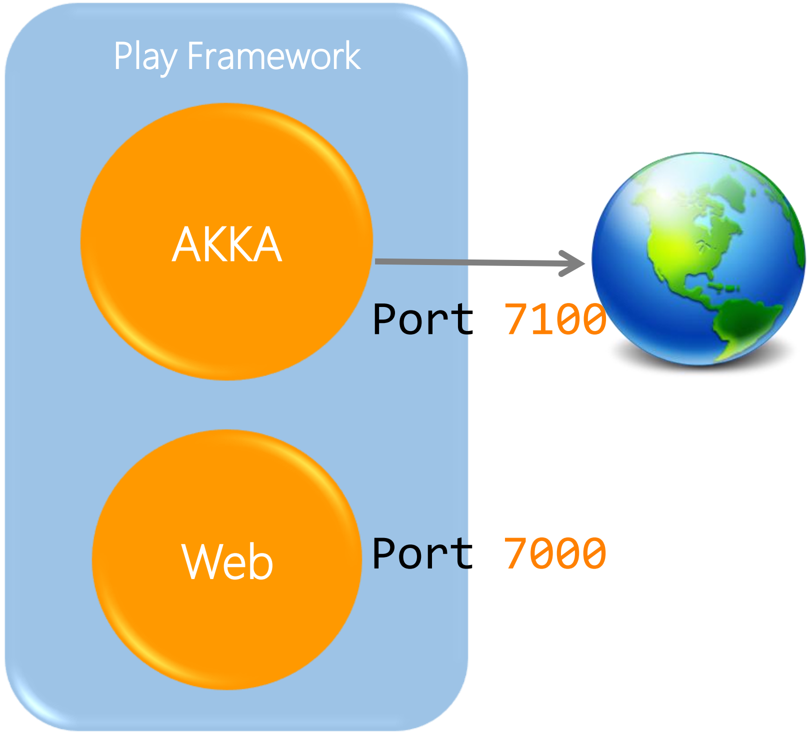
Preparing your ActorSystem for Remoting

- Each Actor has a Path, but an ActorSystem can be publish in an Address.



Local Path	Remote Path
akka://application/user	*akka://application@127.0.0.1/user*
akka://application/user/Supervisor	*akka://application@127.0.0.1/user/Supervisor*
akka://application/user/Supervisor/Child	*akka://application@127.0.0.1/user/Supervisor/Child*

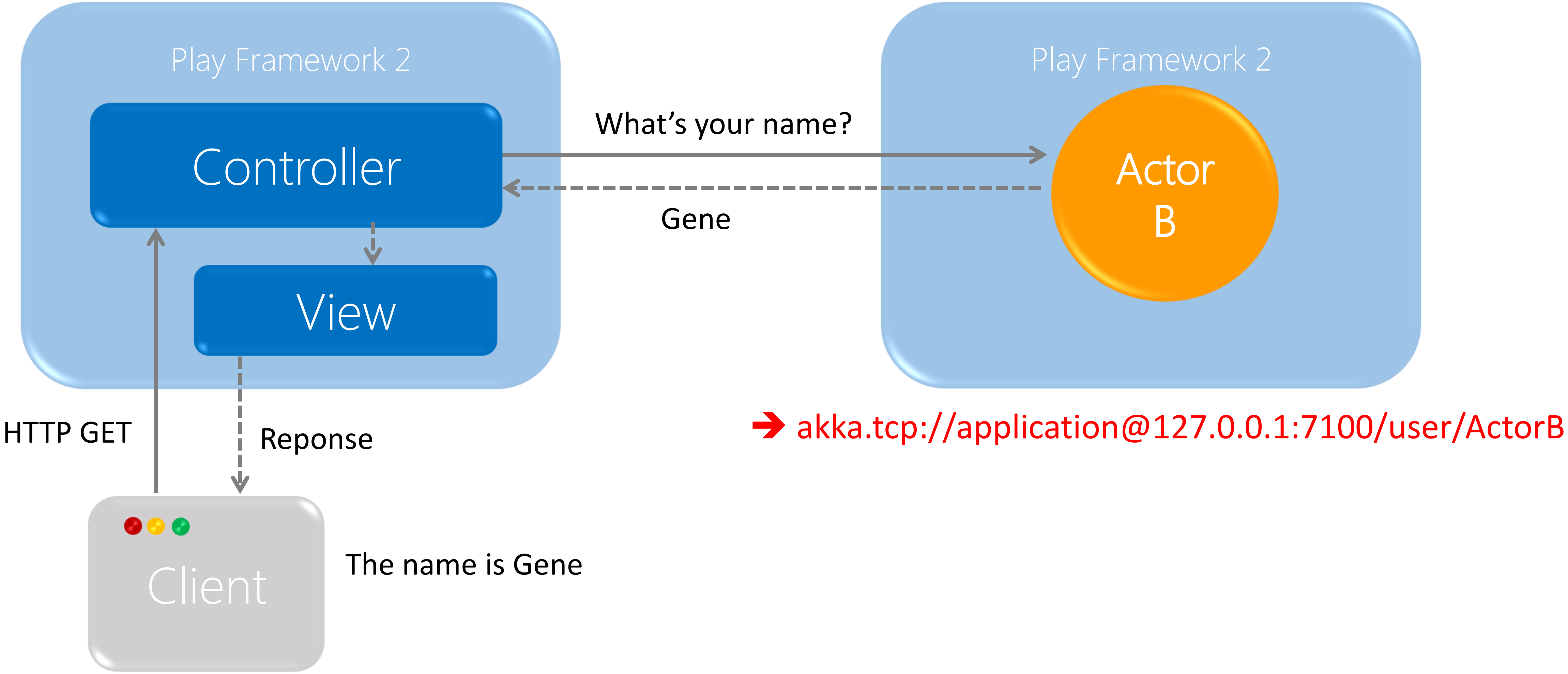
Preparing your ActorSystem for Remoting



application.conf

```
akka {
  actor {
    provider = "akka.remote.RemoteActorRefProvider"
  }
  remote {
    enabled-transports = ["akka.remote.netty.tcp"]
    netty.tcp {
      hostname = "127.0.0.1"
      port = 7100
    }
  }
}
```

Remote Actors



➔ akka.tcp://application@127.0.0.1:7100/user/ActorB

Send Messages to Remote Actors

Retrieve remote actor

```
ActorSelection selection =
            Akka.system().actorSelection("akka.tcp://application@127.0.0.1:7100/user/ActorB");
```

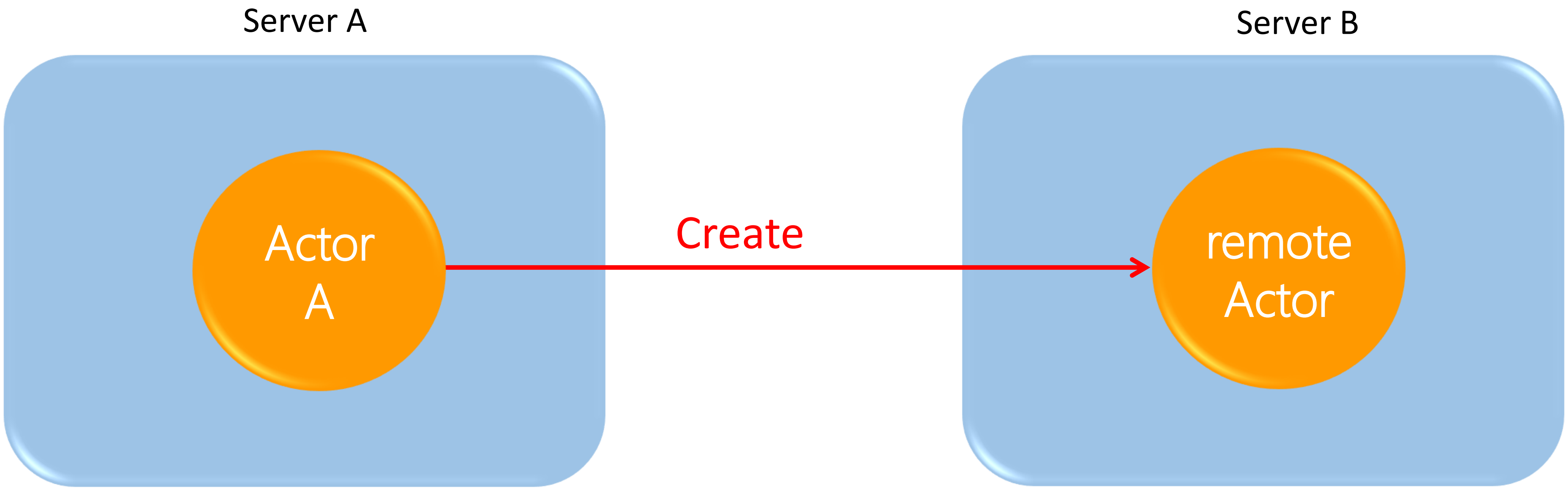
Tell message

```
selection.tell("Hello Remote",null);
```

Ask message

```
Future<Object> rt = Patterns.ask(selection,"What's your name?", timeout);
```

Creating Actor Remotely



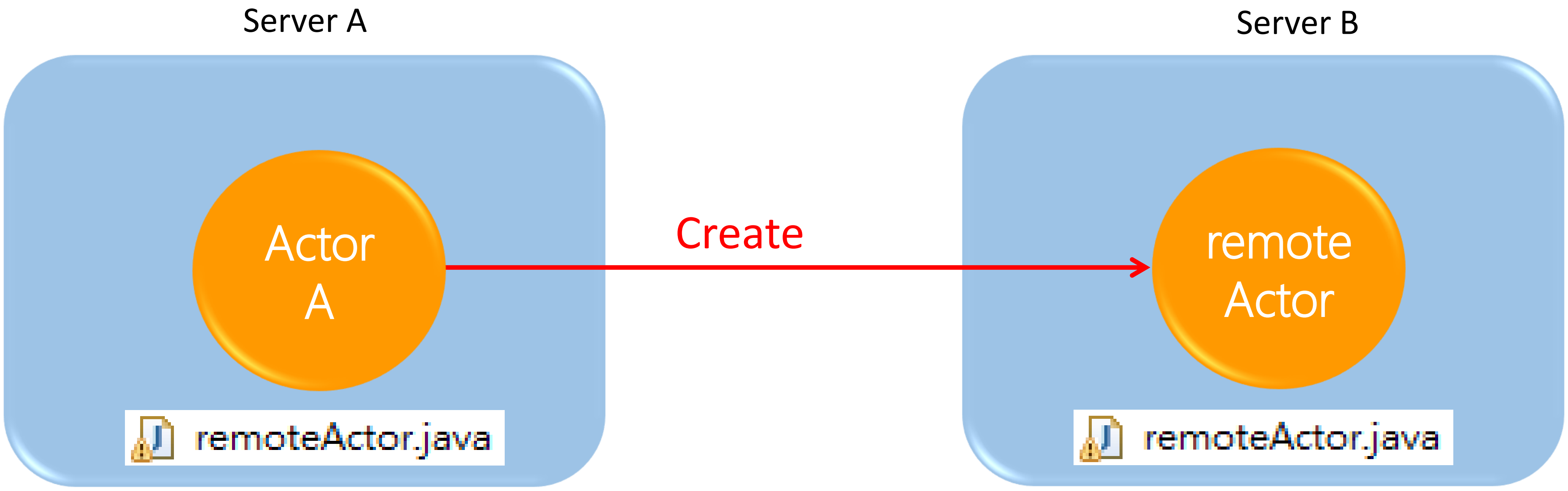
192.168.0.1

192.168.0.2

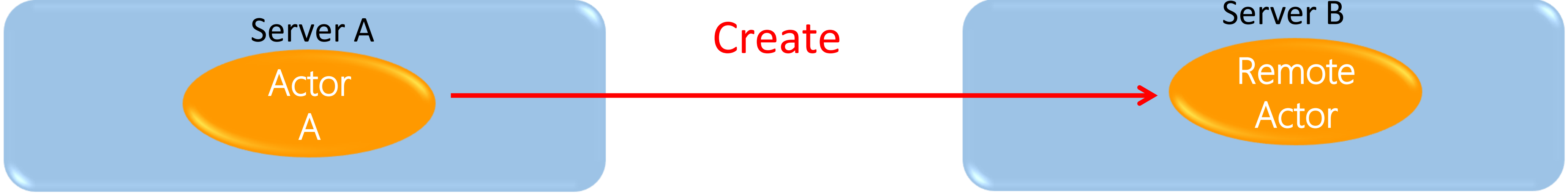
➔ akka.tcp://application@192.168.0.1/user/ActorA

➔ akka.tcp://application@192.168.0.2/user/ActorB

Creating Actor Remotely – How to create?



1 → Both server must have same Actor class to be remote



application.conf for server A

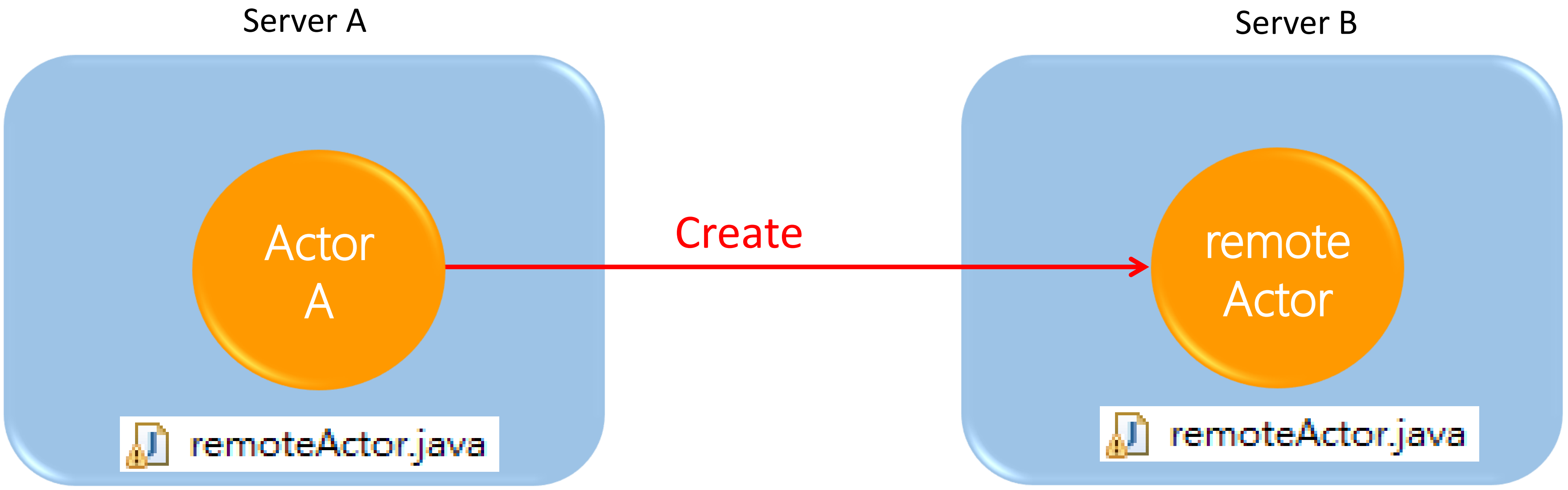
```
akka {
  actor {
    provider = "akka.remote.RemoteActorRefProvider“
    deployment {
      /remoteActor {
        remote = "akka.tcp://application@127.0.0.1:9100"
      }
    }
  }
  remote {
    enabled-transports = ["akka.remote.netty.tcp"]
    netty.tcp {
      hostname = "127.0.0.1"
      port = 7100
    }
 }
}
```

application.conf for server B

```
akka {
  actor {
    provider = "akka.remote.RemoteActorRefProvider
  }
  remote {
    enabled-transports = ["akka.remote.netty.tcp"]
    netty.tcp {
      hostname = "127.0.0.1"
      port = 9100
    }
 }
}
```

2 → Amend the application.conf file for Server A

Creating Actor Remotely – How to create?



```
ActorRef actor = Akka.system().actorOf(Props.create(remoteActor.class), "remoteActor");
actor.tell("Hello Remote",null);
```

3

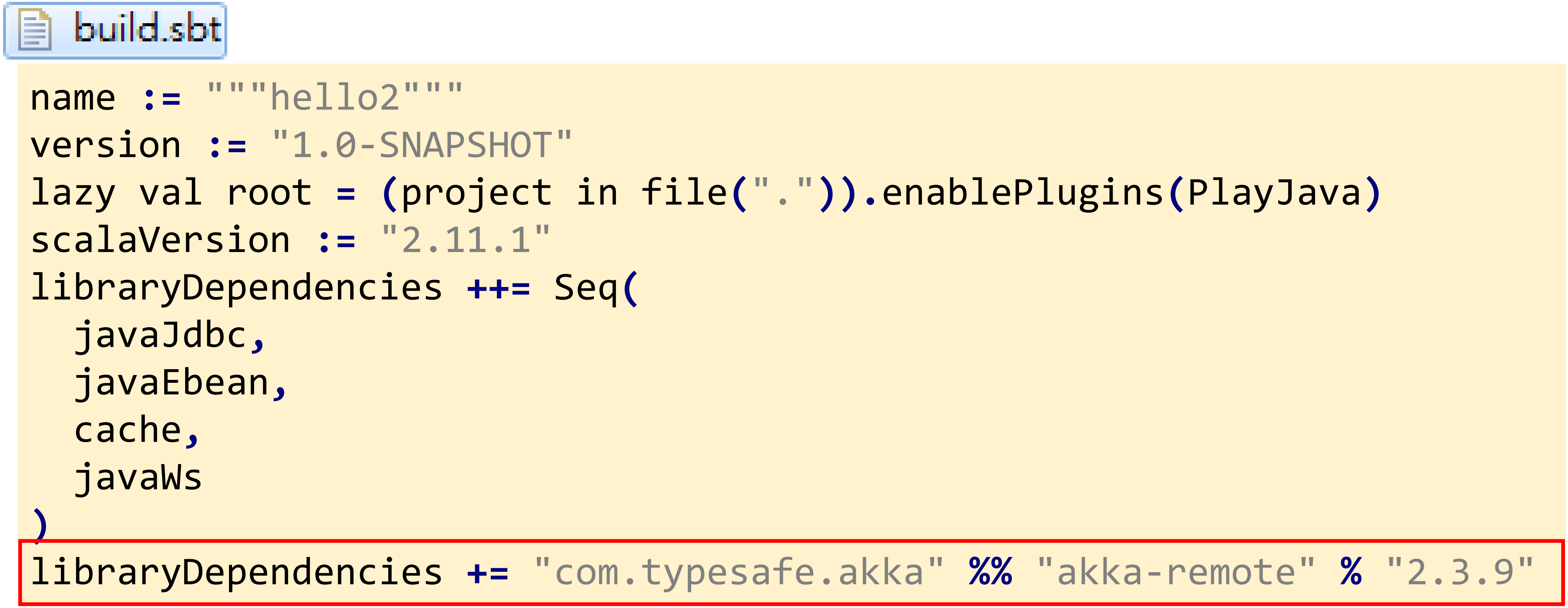
→ Use actorOf () to create a remote actor on Server A.

Another way to Create Actor Remotely

for server A

```
import akka.actor.ActorSelection;
import akka.actor.Address;
import akka.actor.AddressFromURIString;
import akka.actor.Deploy;
import akka.remote.RemoteScope;

public class HelloActor extends Controller {

    public static Result index() {
       Address addr = AddressFromURIString.parse("akka.tcp://application@127.0.0.1:9100");
       ActorRef actor = Akka.system().actorOf(Props.create(remoteActor.class).withDeploy(
                       new Deploy(new RemoteScope(addr))));
       actor.tell("Hello Remote",null);
    }
}
```

Remote Actors



```
name := """hello2"""
version := "1.0-SNAPSHOT"
lazy val root = (project in file(".")).enablePlugins(PlayJava)
scalaVersion := "2.11.1"
libraryDependencies ++= Seq(
  javaJdbc,
  javaEbean,
  cache,
  javaWs
)
libraryDependencies += "com.typesafe.akka" %% "akka-remote" % "2.3.9"
```

THANKS