



ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Testing Actor Systems

Principles of Reactive Programming

Roland Kuhn

Testing Actors

Tests can only verify externally observable effects.

Testing Actors

Tests can only verify externally observable effects.

```
class Toggle extends Actor {  
  def happy: Receive = {  
    case "How are you?" =>  
      sender ! "happy"  
      context become sad  
  }  
  def sad: Receive = {  
    case "How are you?" =>  
      sender ! "sad"  
      context become happy  
  }  
  def receive = happy  
}
```

Akka's TestKit (1)

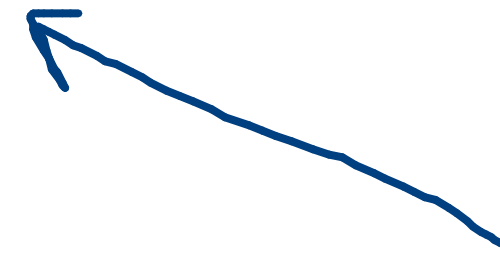
TestProbe as remote-controlled actor.

```
implicit val system = ActorSystem("TestSys")
val toggle = system.actorOf(Props[Toggle])
val p = TestProbe()
p.send(toggle, "How are you?")
p.expectMsg("happy")
p.send(toggle, "How are you?")
p.expectMsg("sad")
p.send(toggle, "unknown")
p.expectNoMsg(1.second)
system.shutdown()
```


Akka's TestKit (2)

Running a test within a TestProbe:

```
new TestKit(ActorSystem("TestSys")) with ImplicitSender {  
  val toggle = system.actorOf(Props[Toggle])  
  toggle ! "How are you?"  
  expectMsg("happy")  
  toggle ! "How are you?"  
  expectMsg("sad")  
  toggle ! "unknown"  
  expectNoMsg(1.second)  
  system.shutdown()  
}
```



sent from testActor

Testing Actors with Dependencies

Accessing the real DB or production web services is not desirable:

- ▶ one simple solution is to add overridable factory methods

Testing Actors with Dependencies

Accessing the real DB or production web services is not desirable:

- ▶ one simple solution is to add overridable factory methods

```
class Receptionist extends Actor {  
  def controllerProps: Props = Props[Controller]  
  ...  
  def receive = {  
    ...  
    val controller = context.actorOf(controllerProps, "controller")  
    ...  
  }  
}
```

Testing Actors with Dependencies

Accessing the real DB or production web services is not desirable:

- ▶ one simple solution is to add overridable factory methods

```
class Getter extends Actor {  
  ...  
  def client: WebClient = AsyncWebClient  
  client get url pipeTo self  
  ...  
}
```

Testing Interaction with the Parent

Create a step-parent:

```
class StepParent(child: Props, probe: ActorRef) extends Actor {  
  context.actorOf(child, "child")  
  def receive = {  
    case msg => probe.tell(msg, sender)  
  }  
}
```

Inserting a Foster-Parent

For when parent–child communication should occur, but monitored:

```
class FosterParent(child: Props, probe: ActorRef) extends Actor {  
  val child = context.actorOf(child, "child")  
  def receive = {  
    case msg if sender == context.parent =>  
      probe forward msg  
      child forward msg  
    case msg =>  
      probe forward msg  
      context.parent forward msg  
  }  
}
```

Testing Actor Hierarchies

Start verifying leaves, work your way up:

- ▶ “Reverse Onion Testing”