Akka

Daniel Hinojosa

Who is this for?

- Java/Groovy/Scala/Clojure Developers interested in Akka

In One Sentence

> “To help the audience understand Actors, Supervision, Futures, Routers in both Scala and Java”

A Note About My Presentation Style

- Trying to achieve the right mix of demonstration and slides.
- Code for this presentation is available on GitHub: http://github.com/dhinojosa/akka-study.git (http://github.com/dhinojosa/akka-study.git)
- Goal is to provide you with code that you can use and reference long after the presentation.
- Presentation slides are also updated as a pdf on the repository.

About Akka

- Set of libraries used to create concurrent, fault-tolerant and scalable applications.
- It contains many API packages:
- Actors, Logging, Futures, STM, Dispatchers, Finite State Machines, and more…
- We are going to only focus on some of the core items.
- Akka’s managing processes can run on
  - in the Same VM
  - in a Remote VM
- Akka’s Actor’s will replace Scala’s Actors

Game Changer?

Making the case

- No `synchronized`, no `wait`, no `notifyAll`
- No Boilerplate
- Failure Tolerance
- Better Scaling (Up or Out)
- Divide and Conquer
- Scala Actors to be deprecated out of the language in favor of Akka

Starting with Actors

- Based on the Actor Model from Erlang.
- Encapsulates State and Behavior
- Concurrent processors that exchange messages.
- Each message is immutable (cannot be changed, this is required!)
- Each message should not be a closure
- Breath of fresh air if you have suffered concurrency

Recognizing Immutable

What does immutable look like in Java?

```
public class Person {
    private final String firstName;
    private final String lastName;

    public Person(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    public String getFirstName() {
        return firstName;
    }

    public String getLastName() {
        return lastName;
    }

    @Override
    public String toString() {...}

    @Override
    public boolean equals(Object o) {...}

    @Override
    public int hashCode() {...}
}
```

What does immutable look like in Scala?

```
case class Person(firstName:String, lastName:String)
```

or

```
class Person(val firstName:String, val lastName:String) {
  override def toString() = {...}
  override def equals(x:AnyRef):Boolean = {...}
  override def hashCode:Int = {...}
}
```

Recognizing closures

What is a closure in Java?

```
import java.util.function.Function;

public class Closures {

  public static Integer foo(Function<Integer, Integer> w) {
     return w.apply(5);
  }

  public static void main(String[] args) {
      Integer x = 3;
      Function<Integer, Integer> y = (Integer z) ->  x + z;
      System.out.println(foo(y)); //8
  }
}
```

What is a closure in Scala?

```
var x = 3
val y = {z:Int => x + z}
def foo(w: Int => Int) = w(5)
println(foo(y)); //8
```

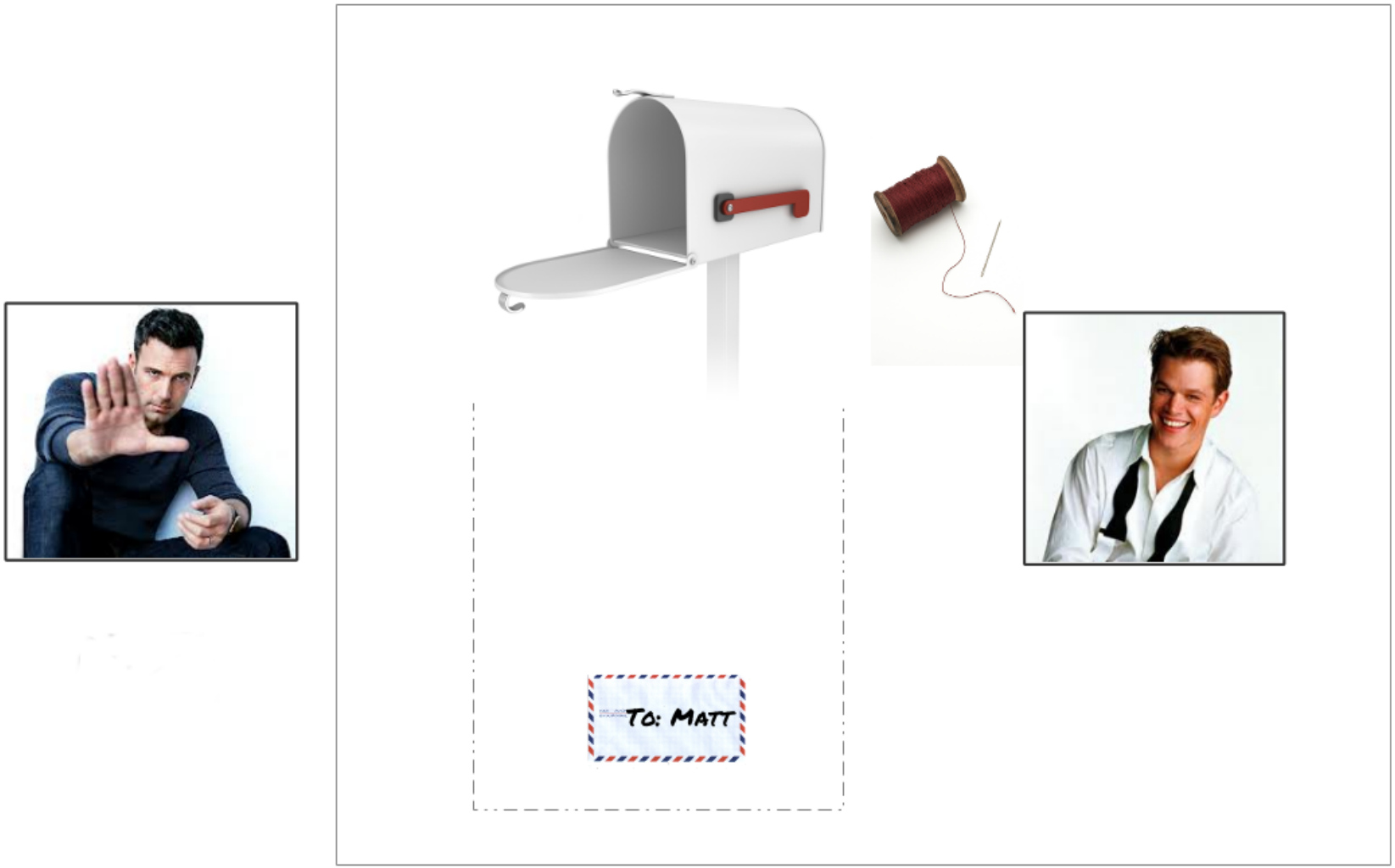
Inside the Actor’s Studio

Love Note Written

Love Note Sent

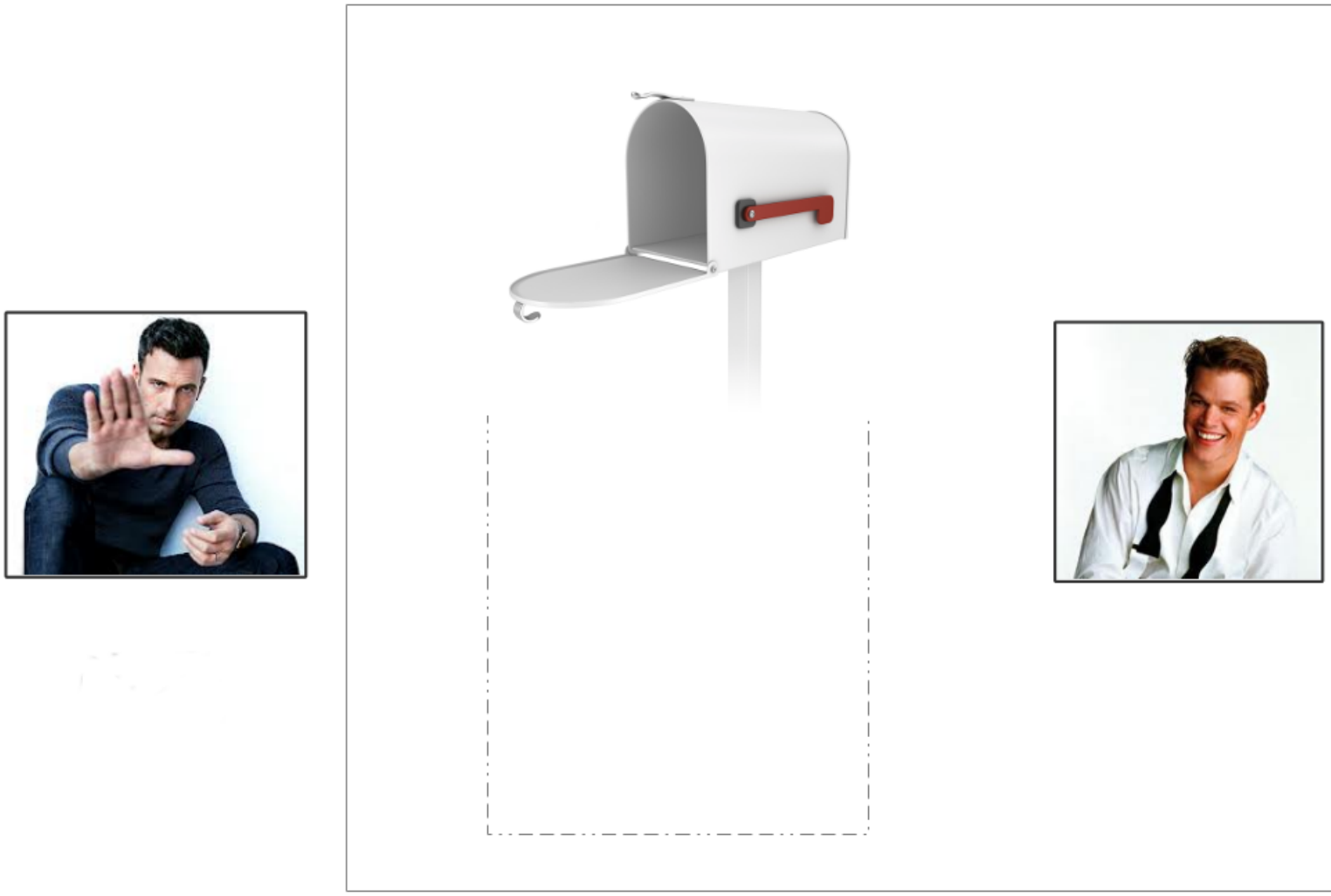
To: Matt

Love Note Processed By Thread

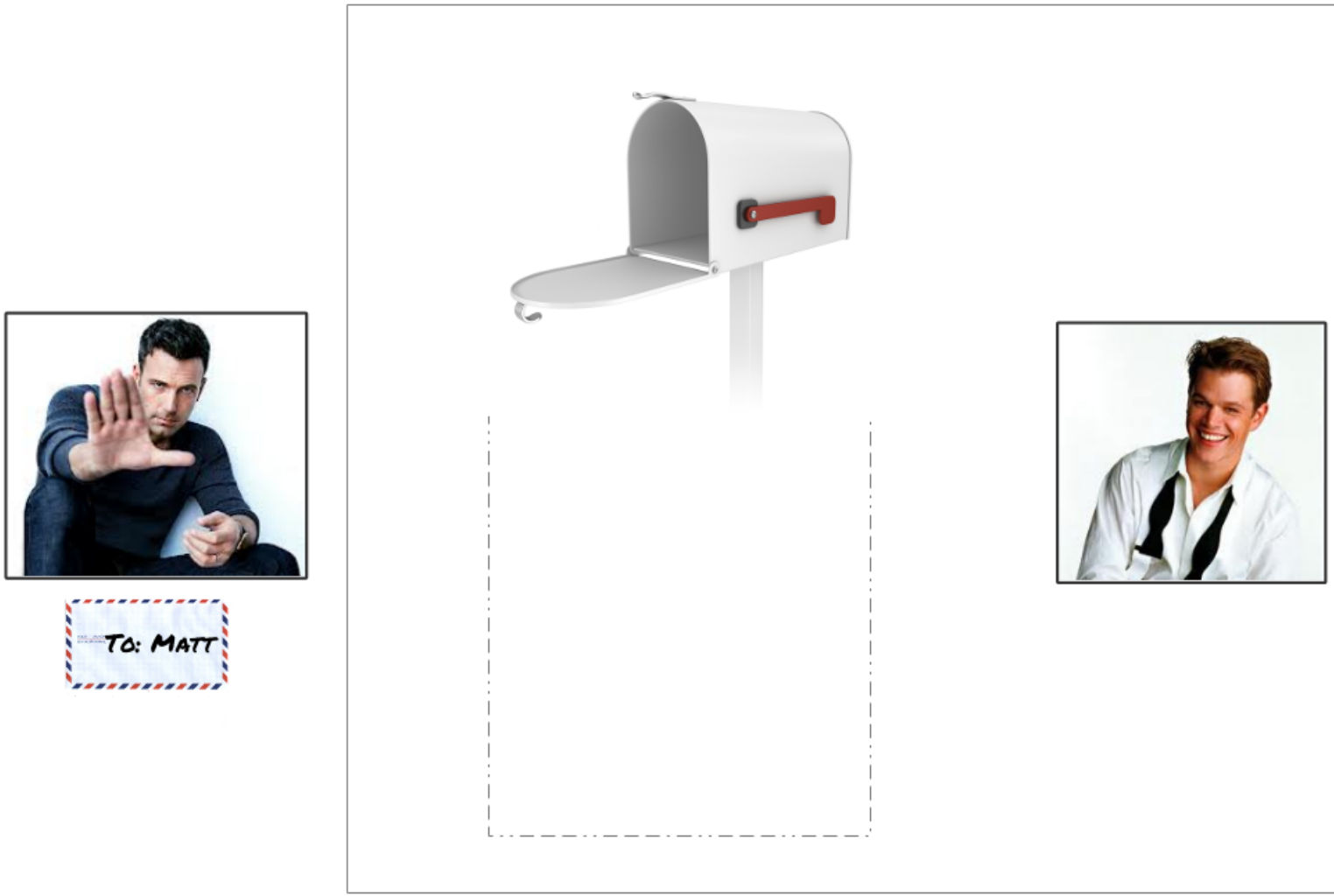


Love Note Processed By Thread

Thread goes away



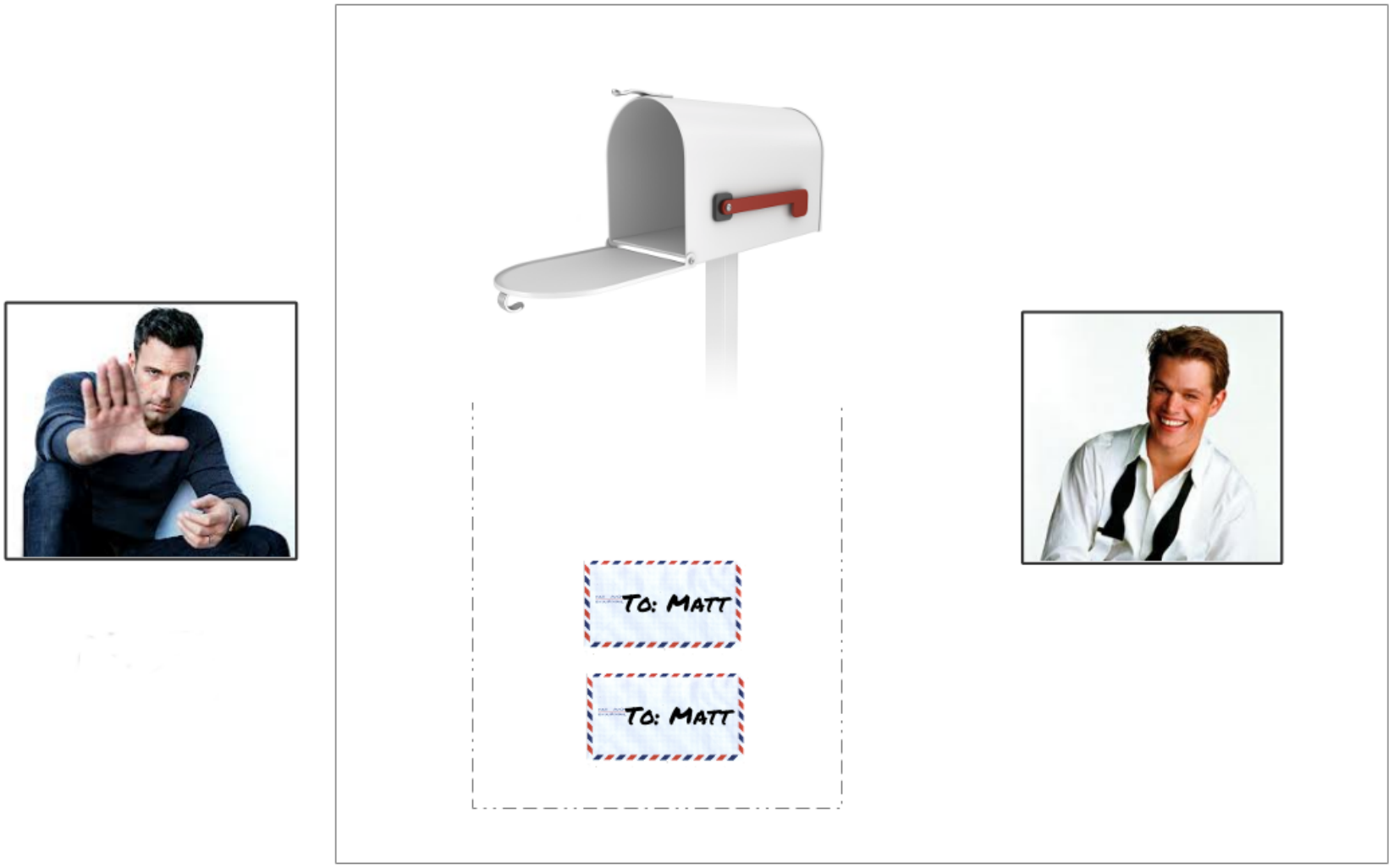
Flooding Matt Damon with Messages



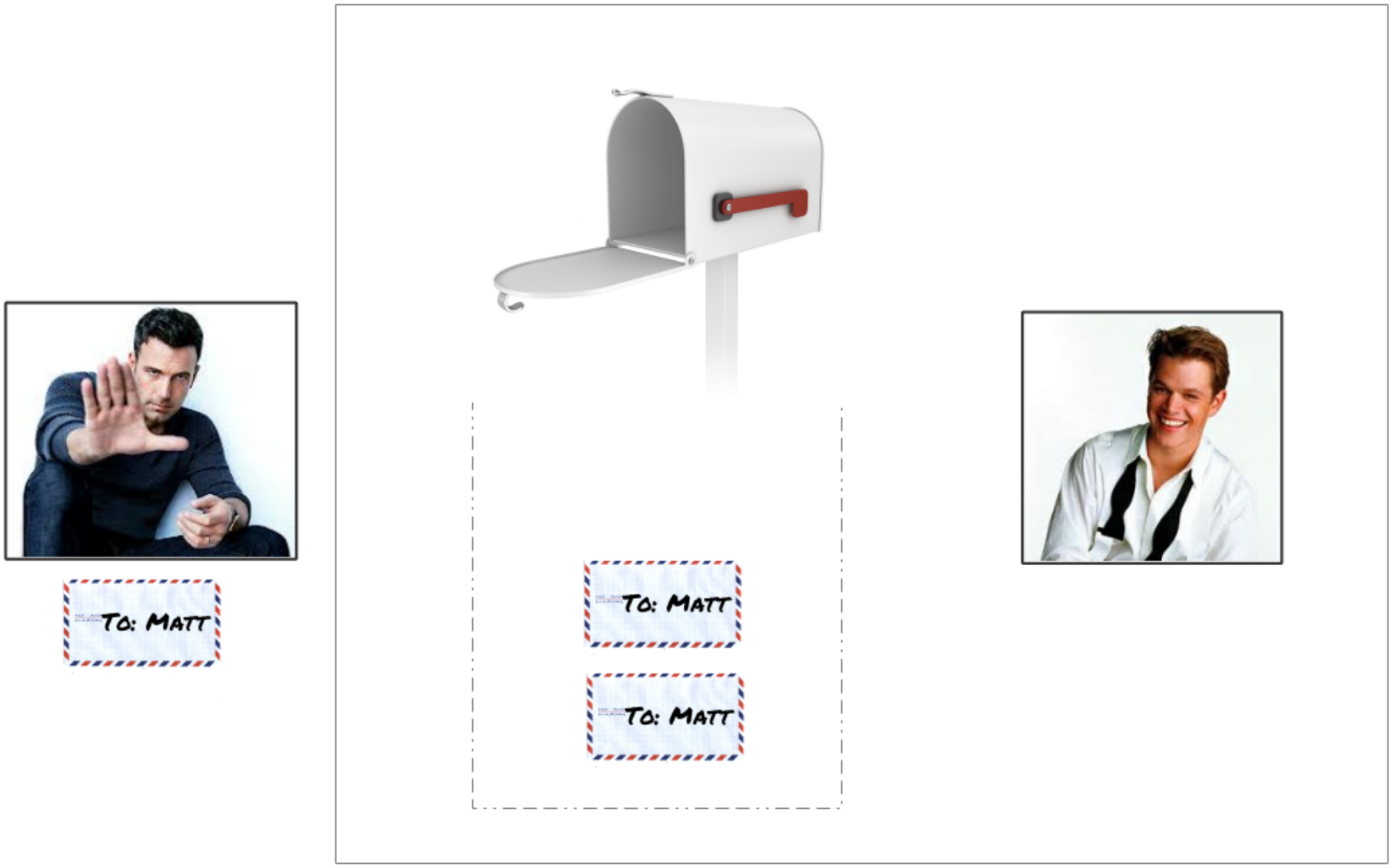
Flooding Matt Damon with Messages

Flooding Matt Damon with Messages

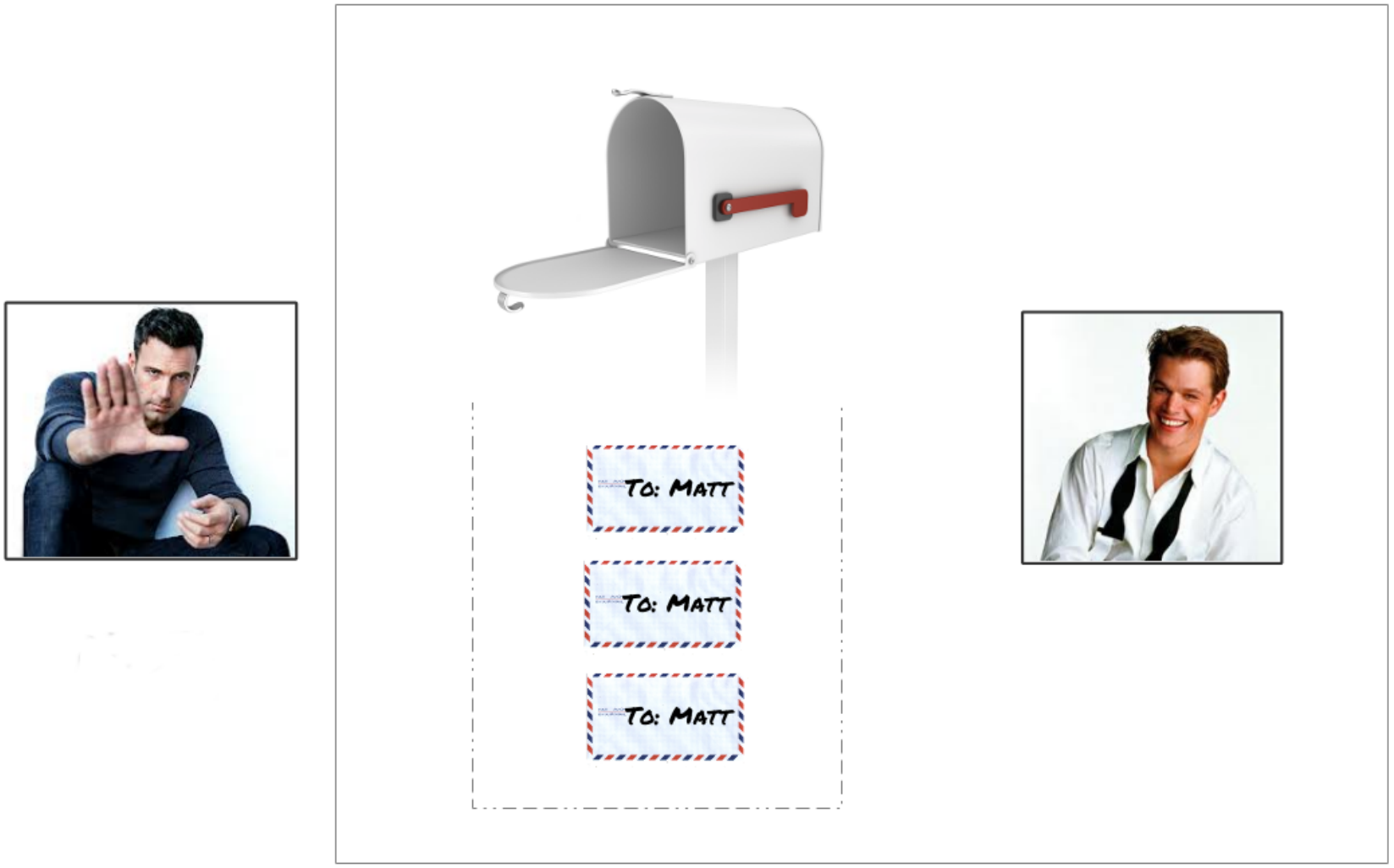
Flooding Matt Damon with Messages



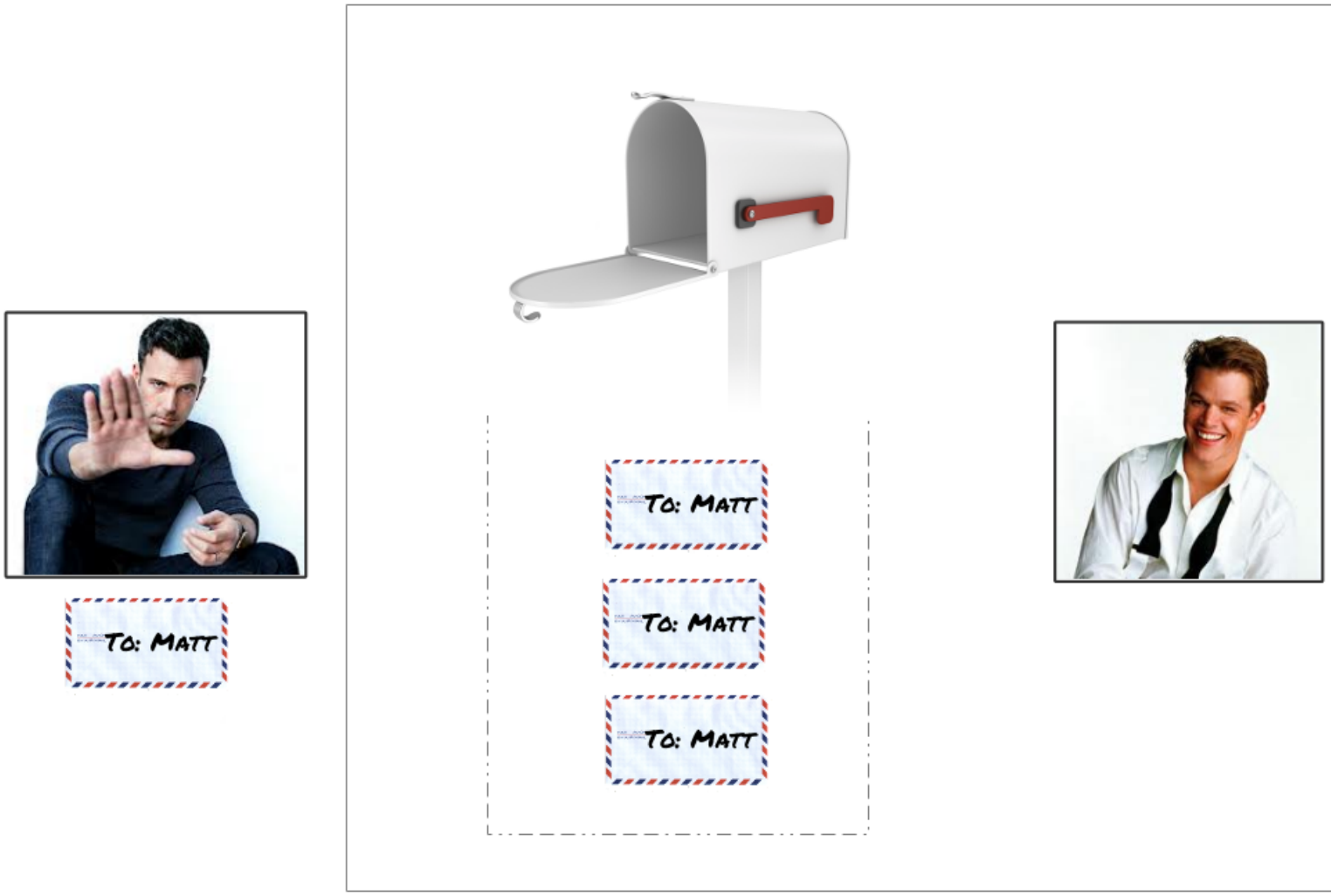
Flooding Matt Damon with Messages



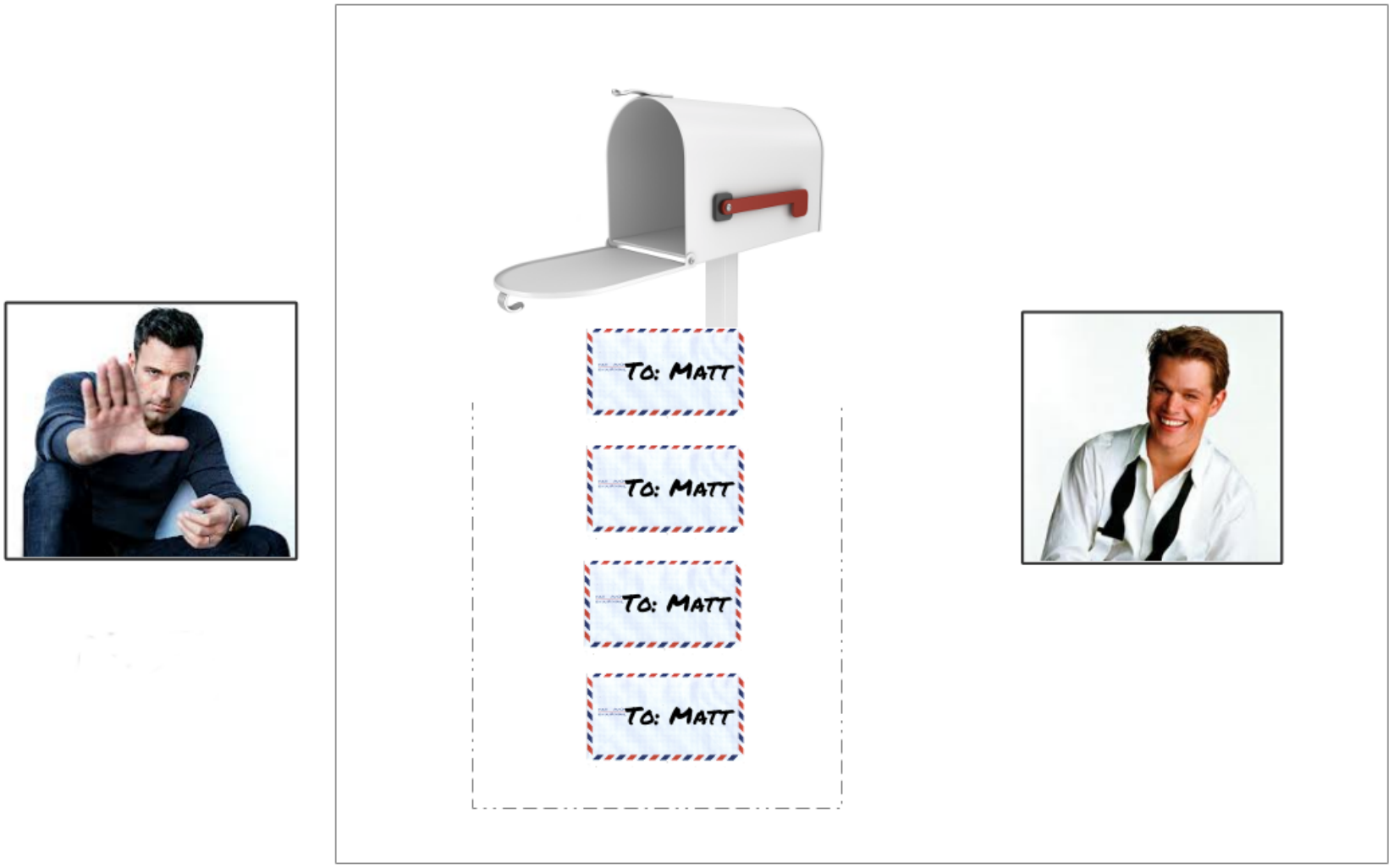
Flooding Matt Damon with Messages



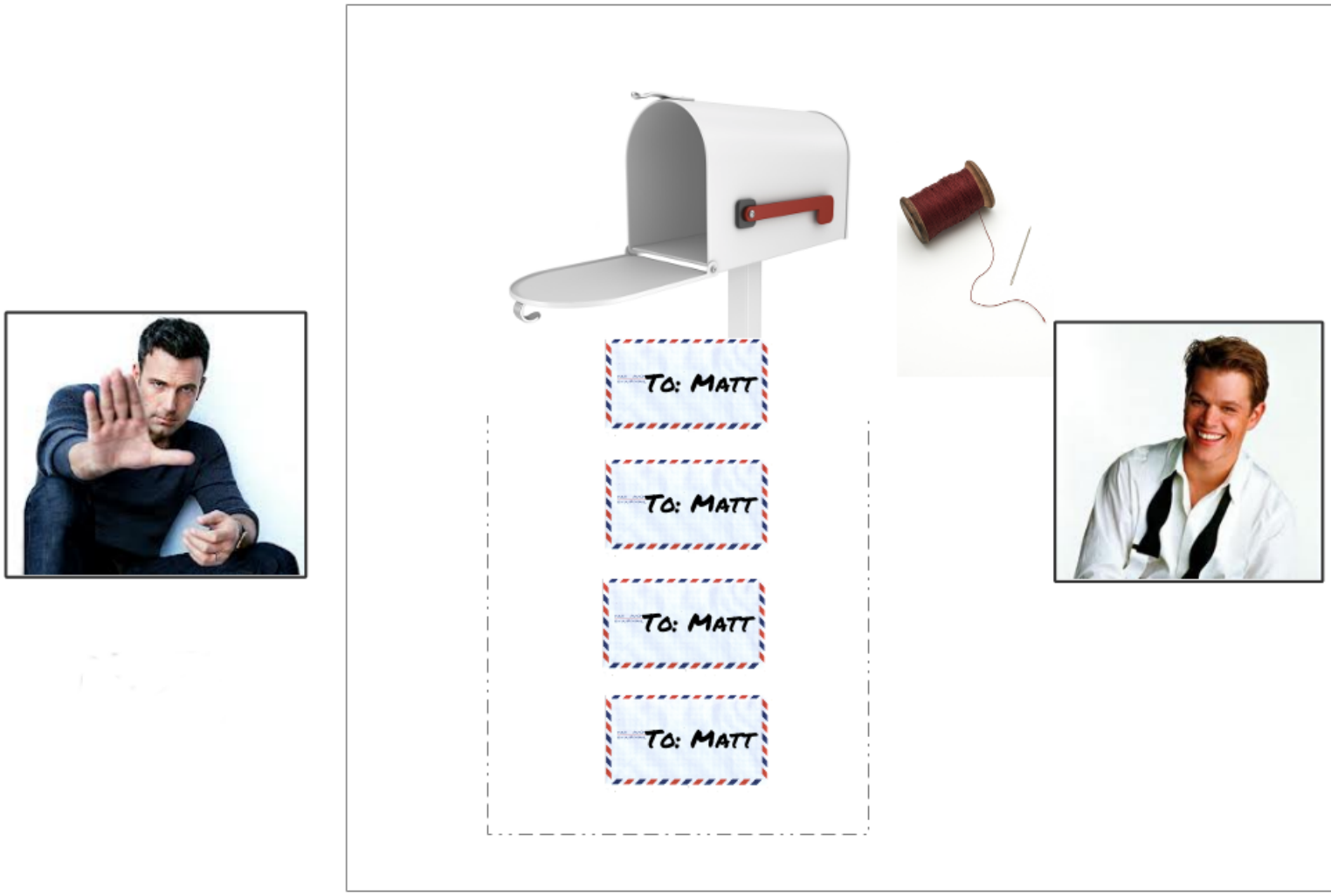
Flooding Matt Damon with Messages



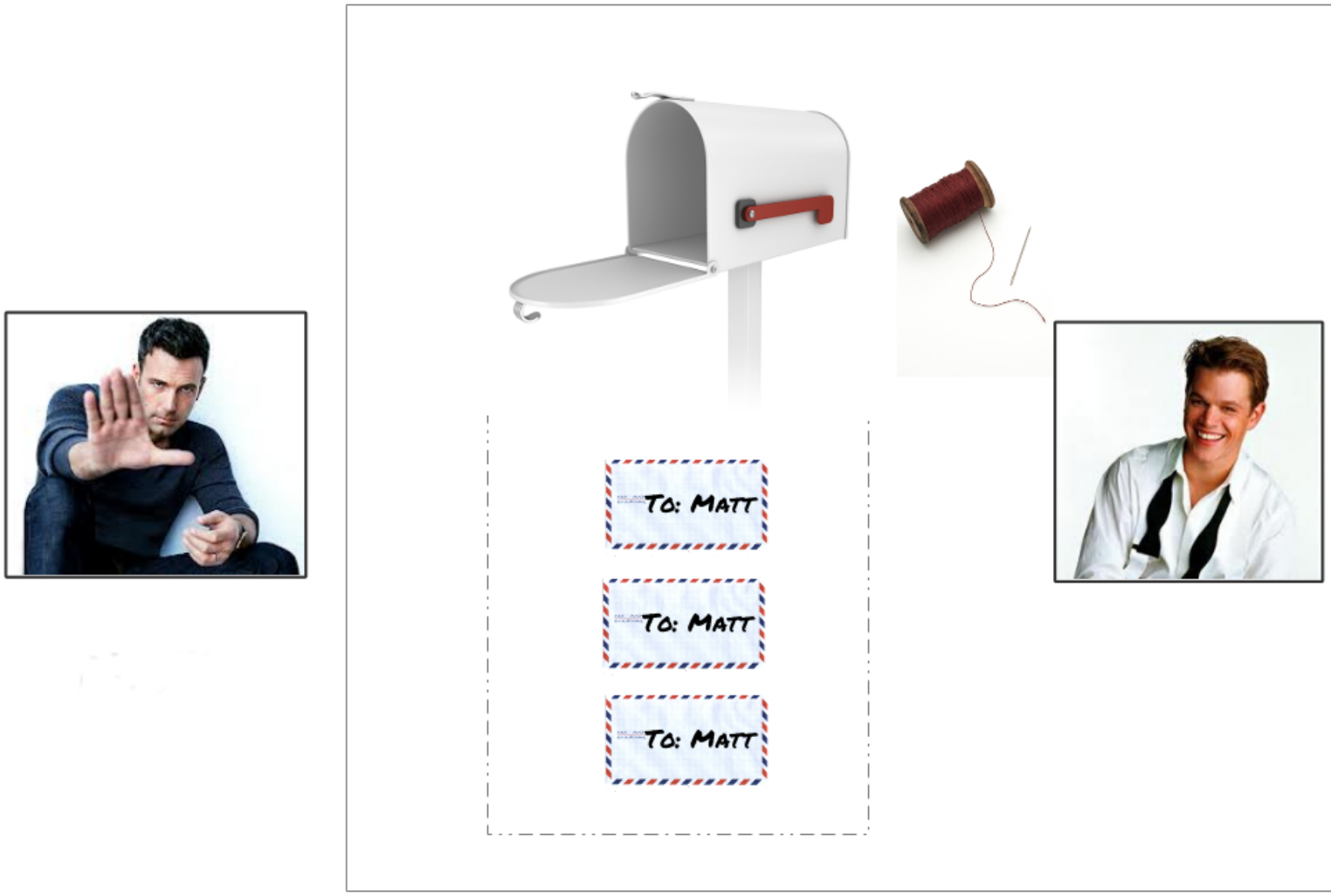
Flooding Matt Damon with Messages



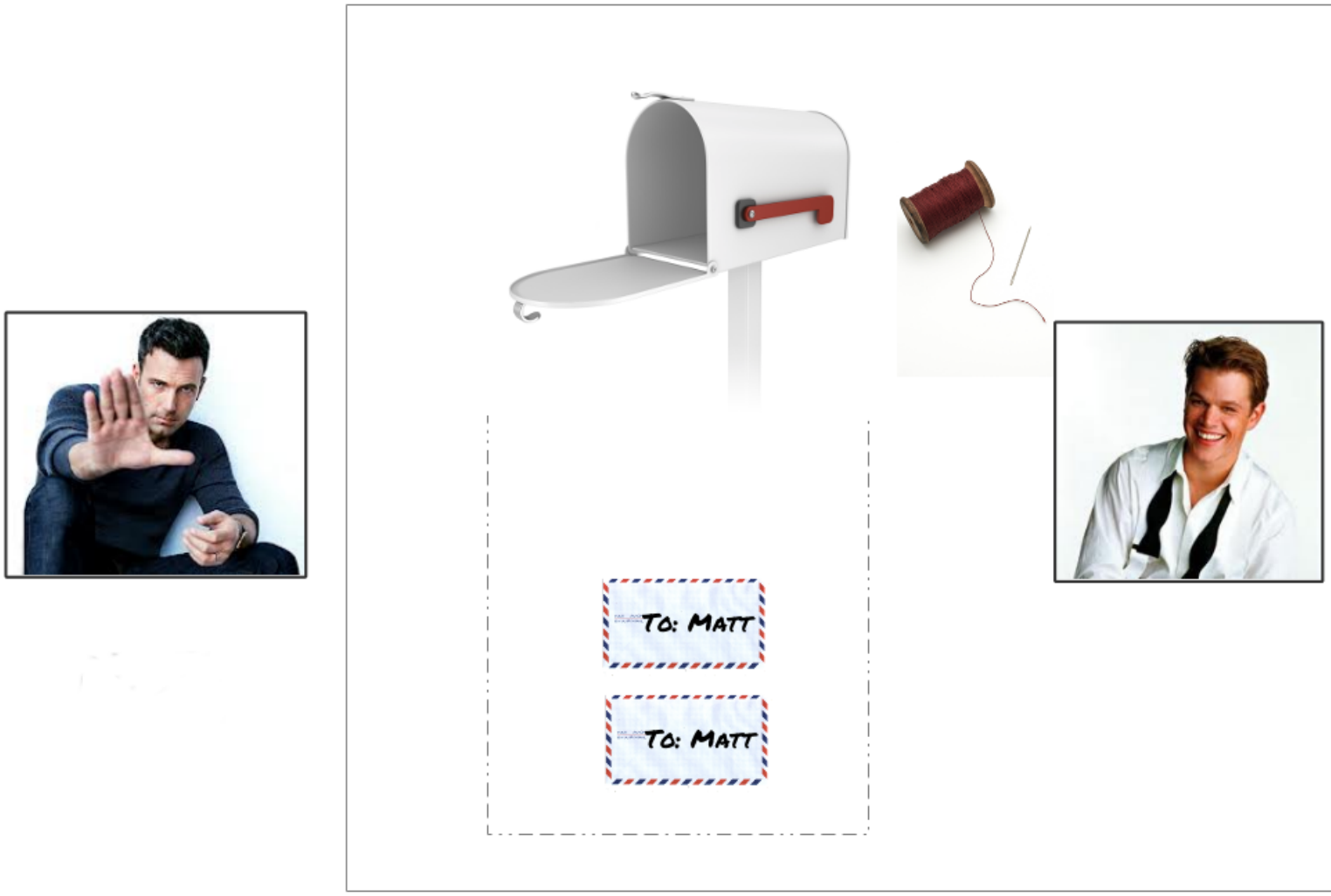
Flooding Matt Damon with Messages



Flooding Matt Damon with Messages



Flooding Matt Damon with Messages

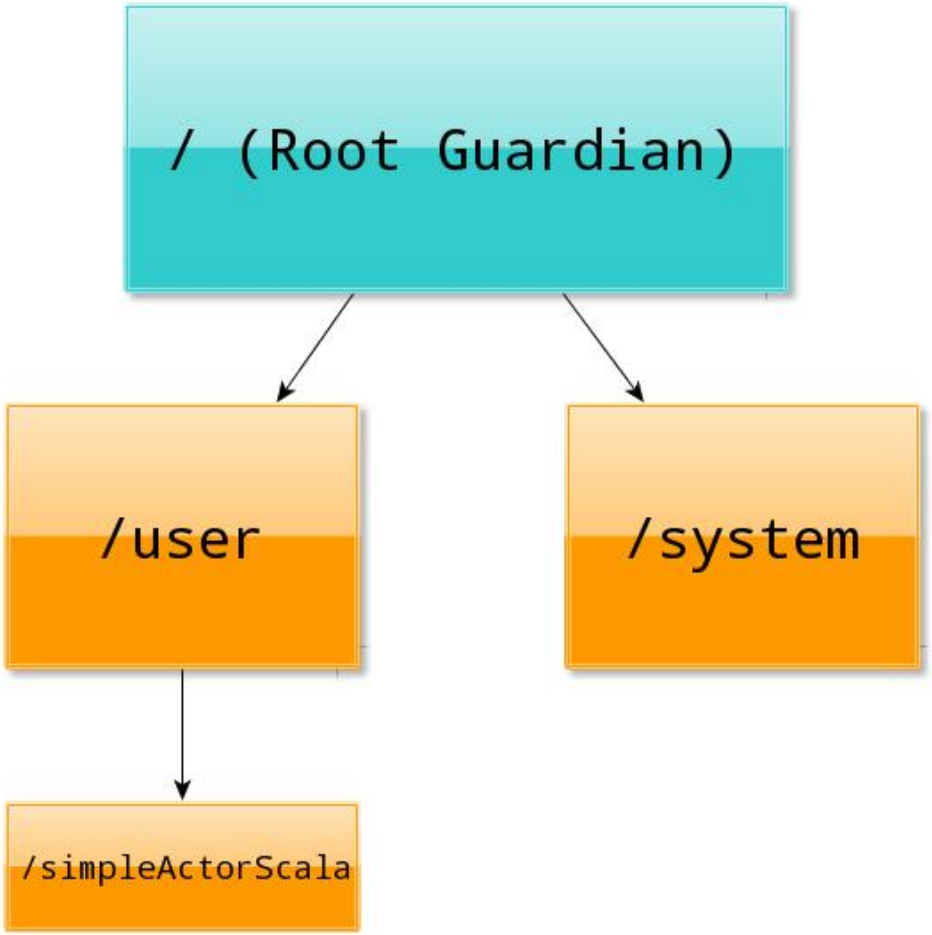


Flooding Matt Damon with Messages

Flooding Matt Damon with Messages

Demo: Setting up Actors

Anatomy of the Actor System




```
akka://{system.name}/user/{actor.name}
```

Paths

- Can be Local

```
akka://my-sys/user/service-a/worker1
```

- Can be Remote

```
akka://my-sys@host.example.com:5678/user/service
```

- Can be Clustered

```
cluster://my-cluster/service-c
```

- Can be Relative

```
../brother-actor
```

Location Transparency

- Vertical and Horizontal Growth
- Horizontal Growth driven by Remoting systems
- Vertical Growth driven by routers
- All configuration based

More about `actorOf()`

- Creates an actor onto a `ActorSystem`
- Creates an actor given the a set of properties to describe the Actor
- Creates an Actor with an identifiable name, if provided
- Returns an `ActorRef`
- If `actorOf` is called *inside* of another actor, that new actor becomes a child of that actor

`actorSelection()`

- Was `ActorFor` which is now deprecated
- Actor references can be looked up using `actorSystem.actorSelection(...)` method.
- `actorSystem.actorSelection` returns a `ActorSelection` object that abstracts over local or remote reference.
- `ActorSelection` can be used as long as the actor is alive.
- Only ever looks up an existing actor, i.e. does not create one.
- The actor must exist or you will receive an EmptyLocalActorRef
- For Remote Actor References, a search by path on the remote system will occur.

About `ActorRef`s

- Any subtype of ActorRef
- ActorRef is the only way to interact with an Actor
- Intent is to send messages to Actor that it represents, proxy.
- Each actor has reference to `self()` refers to it's own reference.
- Each actor also has reference to the `sender()`, the actor that sent the message.
- Any reference can be sent to another actor so that actor can send messages to it.

Other Types of References

- Pure Local Actor References
- Local Actor References
- Local Actor References with Routing
- Remote Actor References
- Promise Actor References
- Dead Letter Actor References
- Cluster Actor References

Actor References within Actor

- `self()` reference to the `ActorRef` of the actor
- `sender()` reference sender Actor of the last received message, typically used as described in Reply to messages
- `context()` exposes contextual information for the actor and the current message.

Lifecycle Methods

- `preStart()` - Lifecycle call when the actor is started
- `preRestart()` - Lifecycle call before the actor is restarted
- `postRestart()` - Lifecycle call after the actor is restarted
- `postStop()` - Lifecycle call after the actor stopped

Futures and Promises

`java.util.concurrent`

- Contains an `Executor`
- `Executor` manages independent threading tasks and decouples task submission from task execution
- Contains an `ExecutorService` that takes `Runnable` or `Callable` tasks and comes complete with a lifecycle and monitoring systems

The `Executor`

```
public interface Executor {
   void execute(Runnable command);
}
```

`ExecutorService`

- An `ExecutorService` is a subinterface of `Executor`
- Provides the ability to create and track `Future`s from `Runnable` and `Callable`s
- Services can be started up and shut down
- Can create your own
- Likely will use `Executors` class to create `ExecutorServices`

Snippet of `ExecutorService`

```
public interface ExecutorService extends Executor {
  ...
   <T> Future<T> submit(Callable<T> task);
   <T> Future<T> submit(Runnable task, T result);
   Future<?> submit(Runnable task);
  ...
}
```

Executors

- Utility Factory that can create `ExecutionServices`

`FixedThreadPool`

- `Executors.newFixedThreadPool()`
- *“Creates a thread pool that reuses a fixed number of threads operating off a shared unbounded queue.”* according to the API
- Keeps threads constant and uses the queue to manage tasks waiting to be run
- If a thread fails, a new one is created in its stead
- If all threads are taken up, it will wait on an unbounded queue for the next available thread

`SingleThreadExecutor`

- `Executors.newSingleThreadExecutor()`
- *“Creates an Executor that uses a single worker thread operating off an unbounded queue.”*
- If a thread terminates due to a failure during execution prior to shutdown, a new one will take its place if needed to execute subsequent tasks.

`CachedThreadPool`

- `Executors.newCachedThreadPool()`
- Factory method is a flexible thread pool implementation that will reuse previously constructed threads if they are available
- If no existing thread is available, a new thread is created and added to the pool
- Threads that have not been used for sixty seconds are terminated and removed from the cache

`ScheduledThreadPool`

- `Executors.newScheduledThreadPool()`
- Can run your tasks after a delay or periodically
- This method does not return an `ExecutorService`
- Returns a ScheduledExecutorService which contains methods to help you set not only the task but the delay or periodic schedule

`Future`

- An asynchronous computation
- Contains methods determine completion of said computation
- Can be in running state, completed state, or have thrown an Exception

`Future` interface

```
public interface Future<V> {
    boolean cancel(boolean mayInterruptIfRunning);
    boolean isCancelled();
    boolean isDone();
    V get() throws
         InterruptedException, ExecutionException;
    V get(long timeout, TimeUnit unit)
         throws InterruptedException,
           ExecutionException, TimeoutException;
}
```

Demo: `Future`s with `java.util.concurrent`

Futures in `scala.concurrent`

- Data structures used to retrieve the result of some concurrent operation.
- It can be retrieved “synchronously” (blocked) or “asynchronously” (unblocked)
- `Future`s need an `ExecutionContext` in scope in order to work

Promises


TICKET
KEEP THIS COUPON
TICKET
KEEP THIS COUPON
TICKET
KEEP THIS COUPON

Demo: `Future`s and `Promise`s with `scala.concurrent`

Ask

- Actors can be asked for return information from an actor using the ask pattern
- Performance Hit especially with remote actors
- Will return a `Future` so that you can wait for an answer

Demo: Ask an `Actor` in Java and Scala

A quick intro to HOCON

- "Human-Optimized Config Object Notation"
- Uses Typesafe Configuration Library https://github.com/typesafehub/config (https://github.com/typesafehub/config)
- JSON Like Features
- Typical configuration file: *application.conf*

Sample HOCON Configuration

```
db.default.driver=org.h2.Driver
db.default.url="jdbc:h2:mem:play"
db.default.user=sa
db.default.password=""
```

Sample HOCON Configuration (Alternate 1)

```
db {
  default.driver=org.h2.Driver
  default.url="jdbc:h2:mem:play"
  default.user=sa
  default.password=""
}
```

Sample HOCON Configuration (Alternate 2)

```
db {
  default {
    driver=org.h2.Driver
    url="jdbc:h2:mem:play"
    user=sa
    password=""
  }
}
```

`application.conf`

- Contains Settings for Actor Systems
- All HOCON (Human Optimized Config Object Notation)
- One *application.conf* per application
- Typically stored src/main/resources

Logical and Physical Paths

- A logical path is seen as if one element is a without recognizing if one actor is remote
- A physical path is the direct path from an actor to it’s ActorSystem. Never spans the JVM

Demo: Setting up and running remote actors on remote systems

Fault Tolerance

- Each actor
  - Can be a parent, and create children
  - Only an actor can create a child
  - Is responsible for their children (a.k.a. subordinate)
- When failure occurs
  - A child or all children are suspended
  - Parent determines the next course of action
  - Mailbox contents are maintained

One for One Strategy

- Each child is treated is separately
- Typically the normal one that should be used
- Default if no strategy is defined

All for One Strategy

- Each child will be given the same treatment
- If one fails, they all essentially "fail"
- Used if tight coupling between children is required

Default Fault Tolerance Behavior

- `ActorInitializationException` will stop the failing child actor
- `ActorKilledException` will stop the failing child actor
- `Exception` will restart the failing child actor
- Other types of `Throwable` will be escalated to parent actor

Actor References within Actor (Repeat)

- `self()` reference to the `ActorRef` of the actor
- `sender()` reference sender Actor of the last received message, typically used as described in Reply to messages
- `context()` exposes contextual information for the actor and the current message.

Demo: Fault Tolerance

Routers

- Actor that reroutes message to other actors
- Different Strategies available
- Create your own

Out of the Box Routers

- `RoundRobinRoutingLogic`
  - Round Robin Distribution
  - Chooses either number of Routees **or** a list of Routees **not both**.
- `RandomRoutingLogin`
  - Chooses a routee randomly
- `SmallestMailboxRouter`
  - Chooses non-suspended routee with the least messages in its mailbox
- `BroadcastRouter`
  - Sends messages to all the routees!

Out of the Box Routers (Continued)

- `ScatterGatherFirstCompletedRouter`
  - Sends messages to all routees, gets a `Future`,
  - Whichever routee responds first, that reponse will be sent to the `sender()`
- `ConsistentHashingRouter`
  - Consistent Hashing
  - Special type of map where remapping of keys is limited
  - K/n mapping keys will be remapped
    - K is the number of keys
    - n is the number of slots

Demo: Routers

Questions?

Thank You

- Email: dhinojosa@evolutionnext.com (mailto:dhinojosa@evolutionnext.com)
- Github: https://www.github.com/dhinojosa (https://www.github.com/dhinojosa)
- Twitter: http://twitter.com/dhinojosa (http://twitter.com/dhinojosa)
- Google Plus: http://gplus.to/dhinojosa (http://gplus.to/dhinojosa)
- Linked In: http://www.linkedin.com/in/dhevolutionnext (http://www.linkedin.com/in/dhevolutionnext)

Last updated 2014-06-27 09:52:27 MDT