Actors with Akka and Scala

Markus Jais

Built on a great fundation

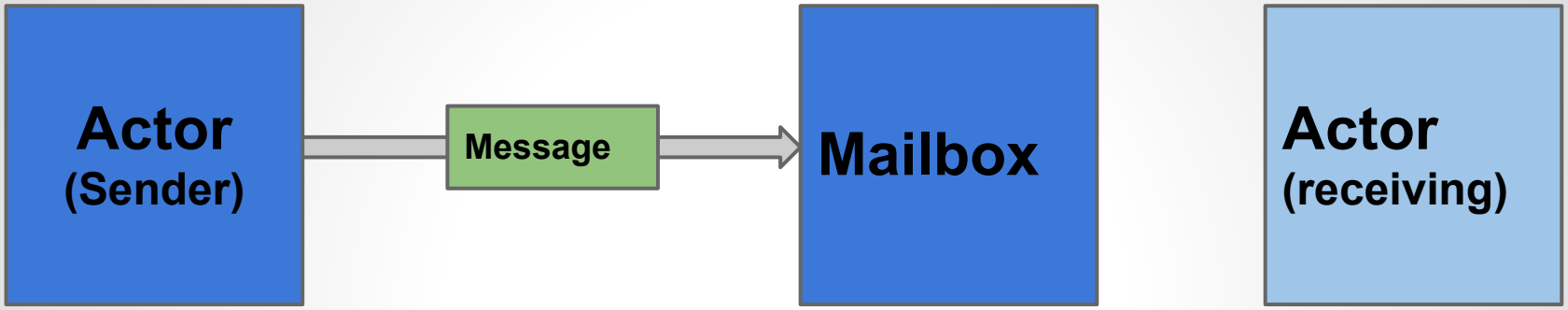
```
$ find . -iname "*a"|xargs grep -s "java.util.concurrent" |grep -v test  |wc -l
147
```

```
find . -iname "*a"|xargs grep -s "volatile" |grep -v test  |wc -l
79
```

Also:
- scala.concurrent (also builds on JDK, JVM)

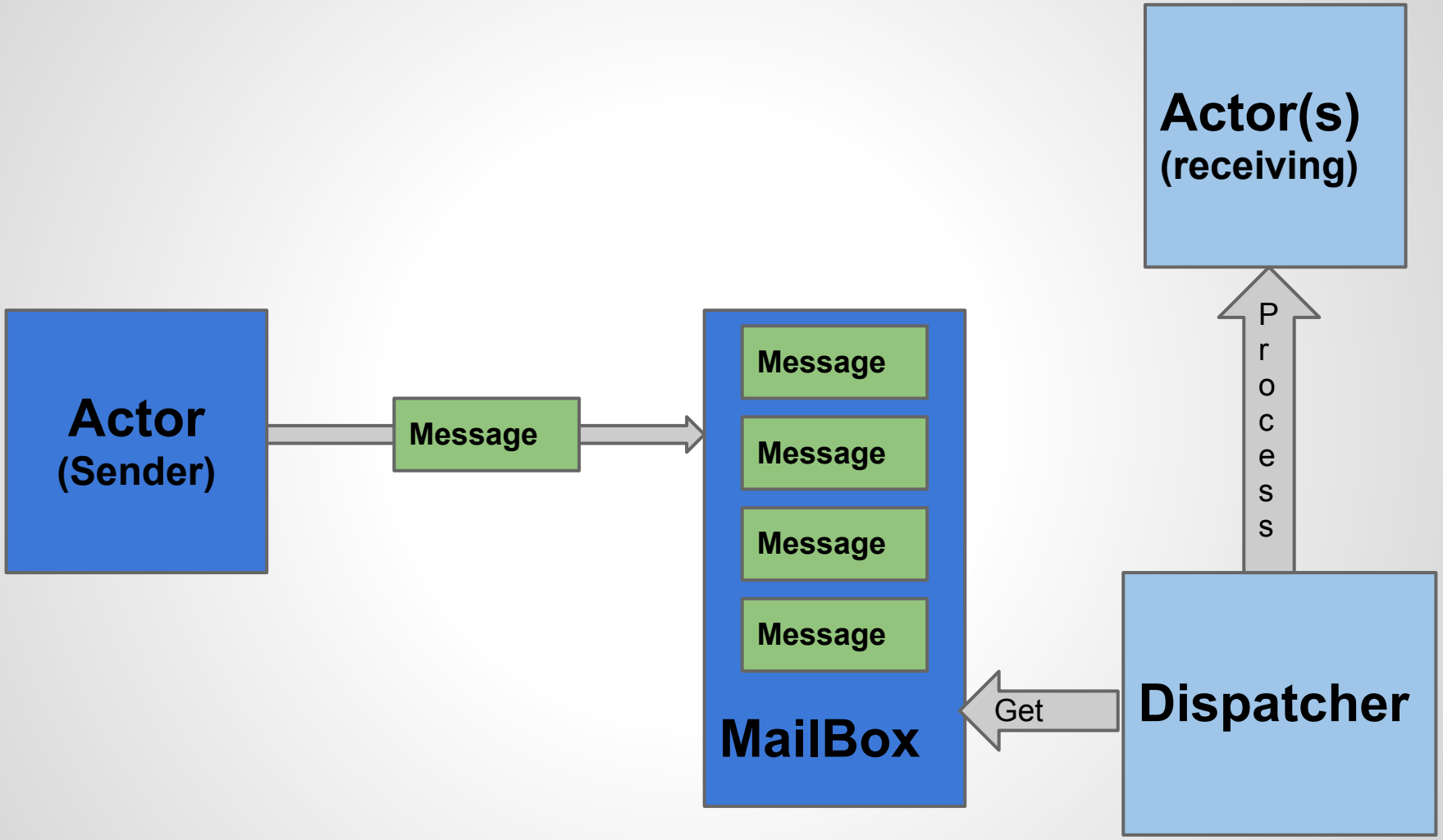
Don’t hate JVM/JDK/Java!
Without it there would be no Akka!

Message flow with Actors



- Message should be **immutable**
- Receiving actor does not need to be running
- Availability of receiving actor does not affect sender

Message flow with Actors



Actors can have state - but no race conditions possible!

Actors do one thing at a time!

Actor System

everything runs within an Actor System

- Create only one per application
- With a JVM several Actor Systems can run
- Actor Systems are independent from each otherder

You never work with an Actor itself.
You only work with an
ActorRef

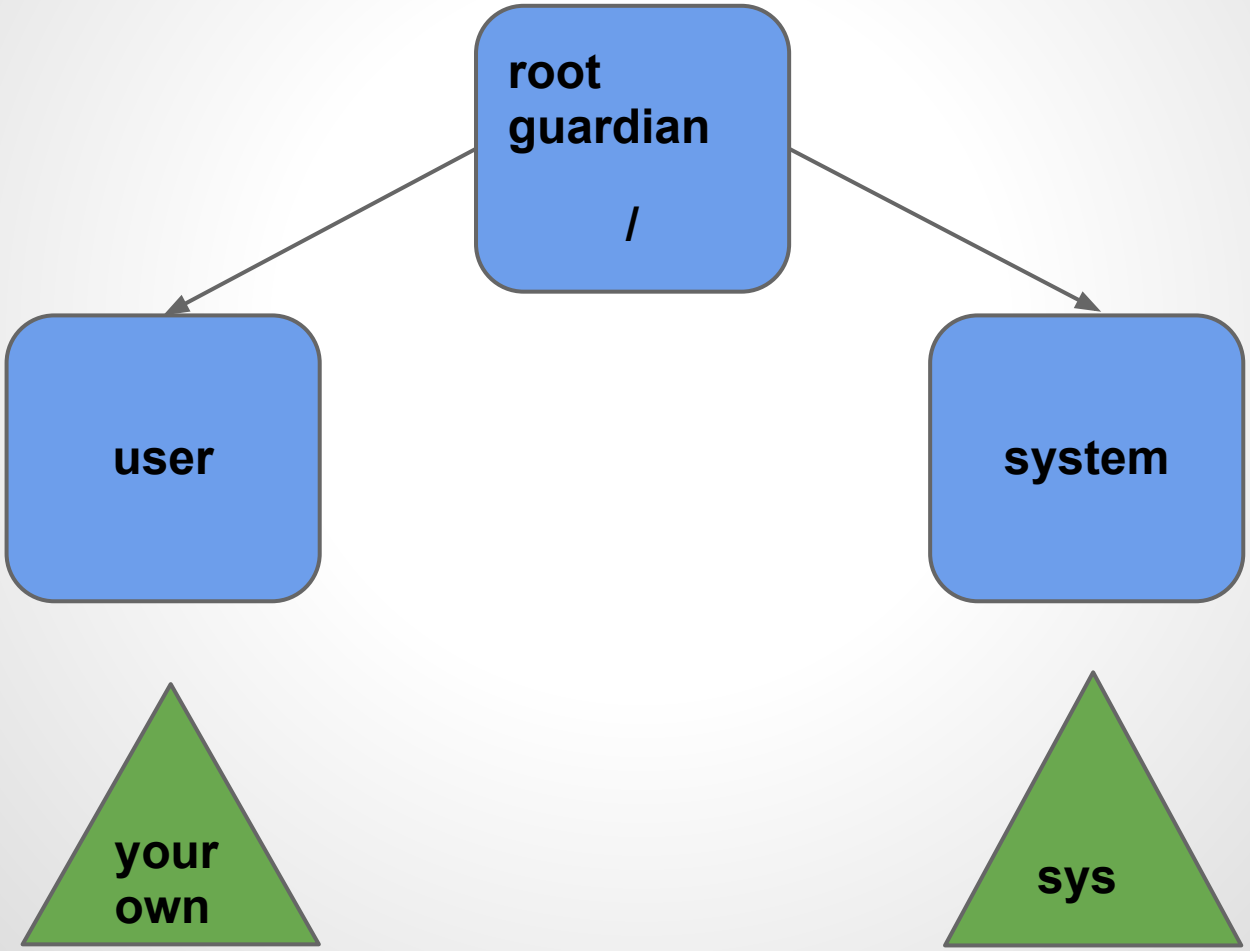
ActorRef:

- Each actor reference is a subtype of ActorRef
- for sending messages to “it’s” actor
- ActorRef can be accessed through “self”
- This reference is also the default “sender” for messages
- Receiving actor can use “sender” to send back messages

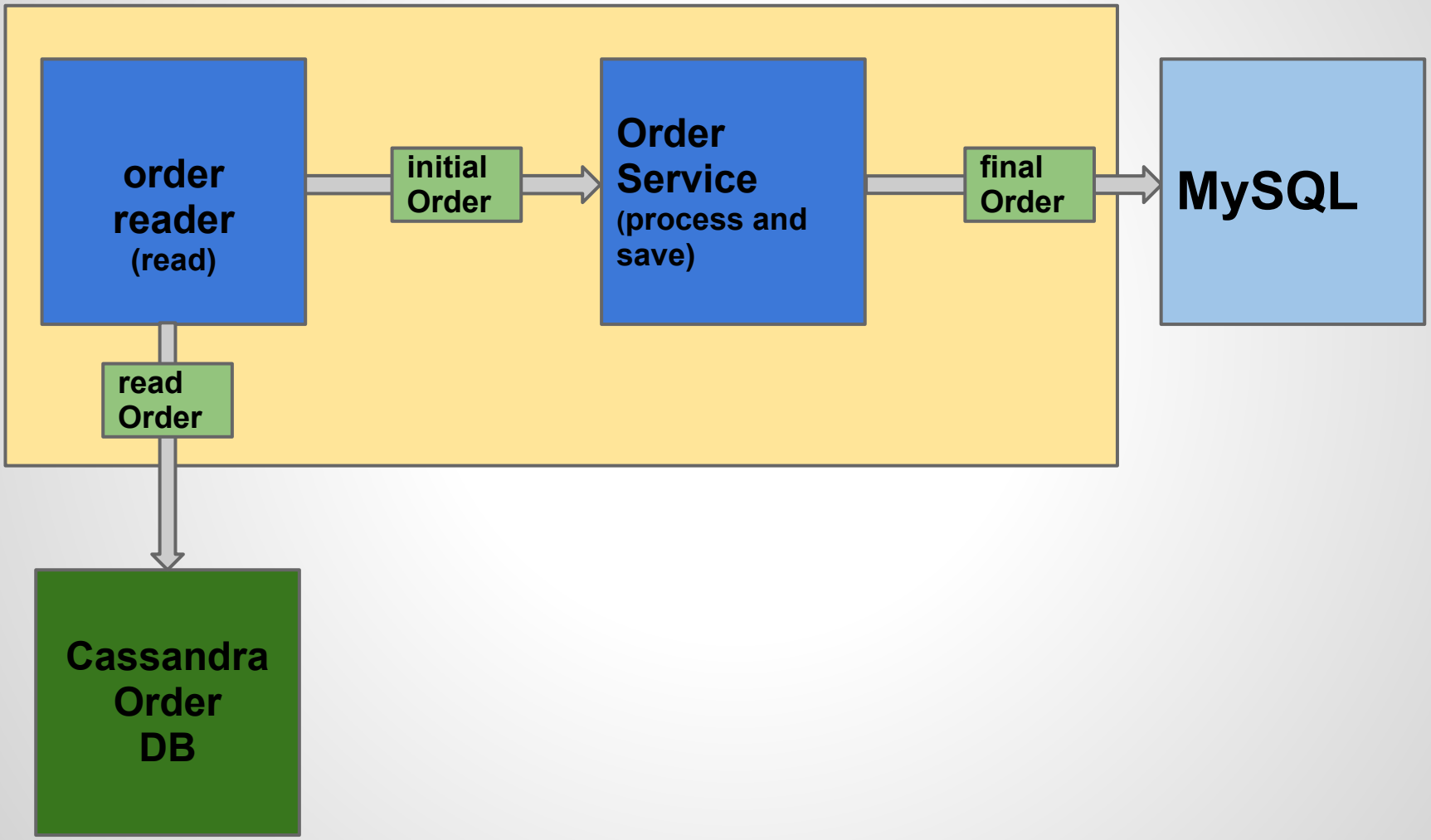
An Actor can change state

Let it crash

Fault Tolerance and Supervision



Fault Tolerance - an example



Fault Tolerance - an example

```
val orderService
val orderProcessor
val orderReader

val cassandraConnection = driver.getConnection
val initialOrders = cassandraConnection.getOrders

//for each inintialOrder
val finalOrder = orderProcessor.process(initinalOrder)
val mySQLConnection = mysqlDriver.getConnection
mySQLConnection.saveToMySQL(finalOrder)
```

Fault Tolerance - an example

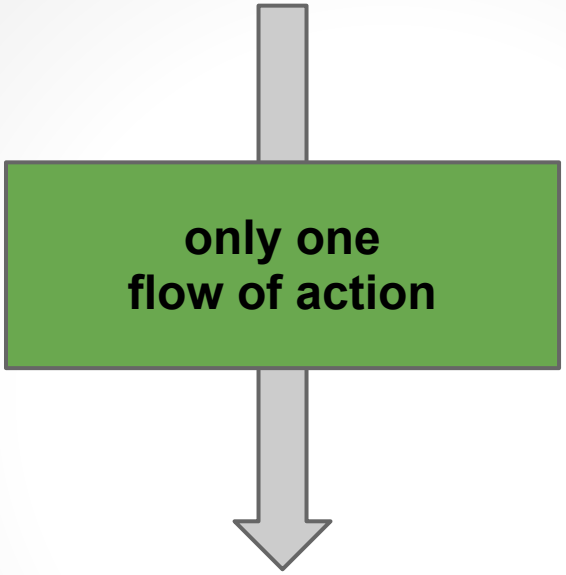
```
val orderService
val orderProcessor
val orderReader

val cassandraConnection = driver.getConnection
val initialOrders = cassandraConnection.getOrders

//for each inintialOrder
val finalOrder = orderProcessor.process(initinalOrder)
val mySQLConnection = mysqlDriver.getConnection
mySQLConnection.saveToMySQL(finalOrder)
```

What about exceptions?

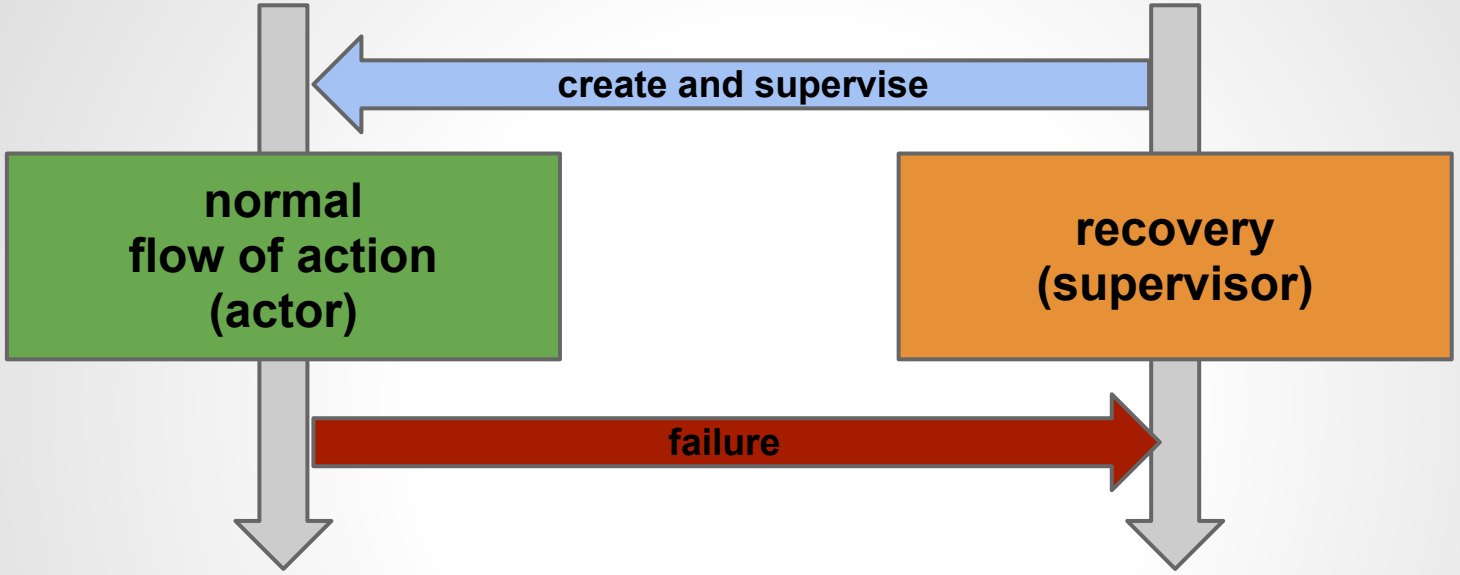
Fault Tolerance - problem with exceptions



everything on the executing thread

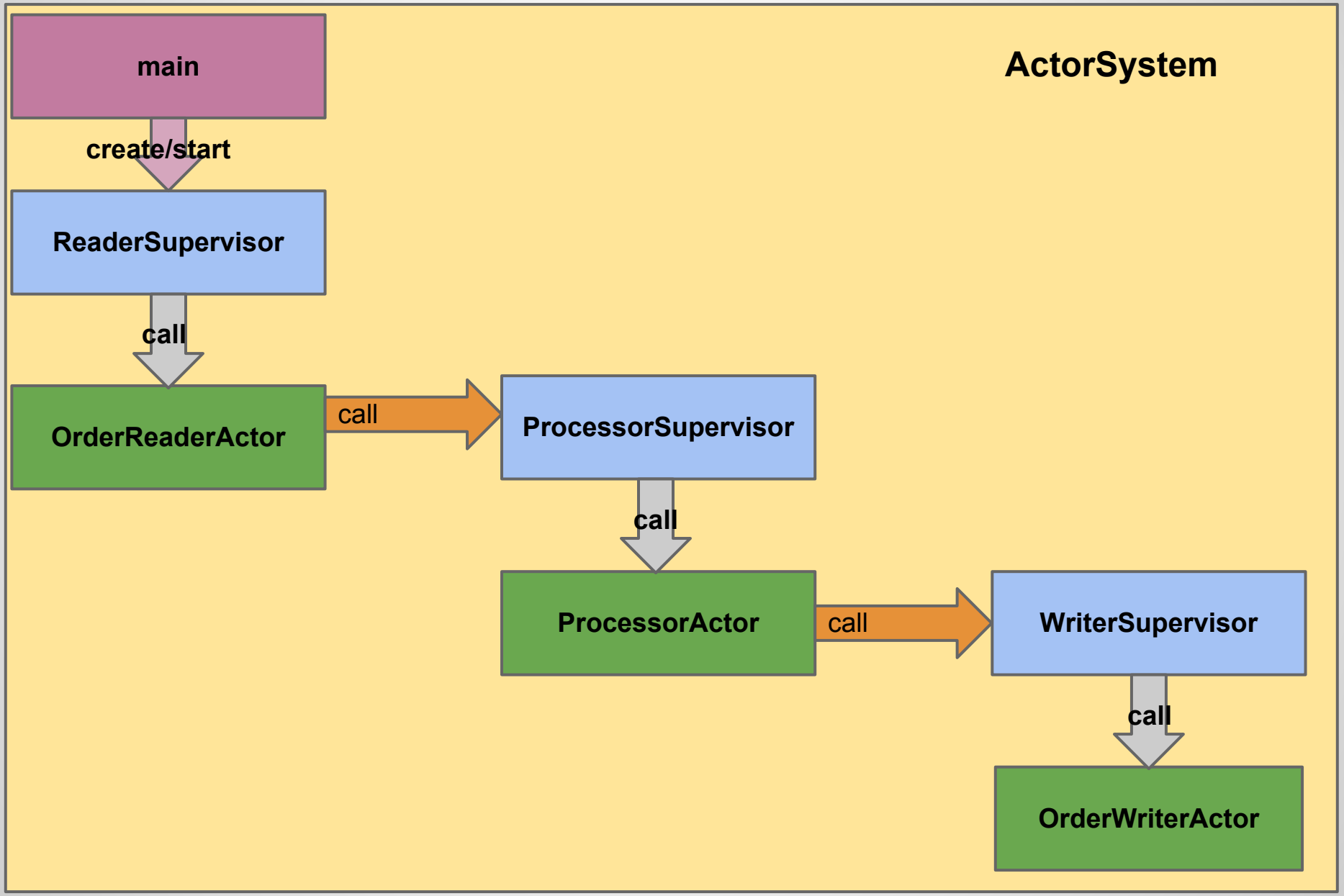
What happens with an exception?

Fault Tolerance - better solution



separating normal flow and recovery

Fault Tolerance - example



Fault Tolerance

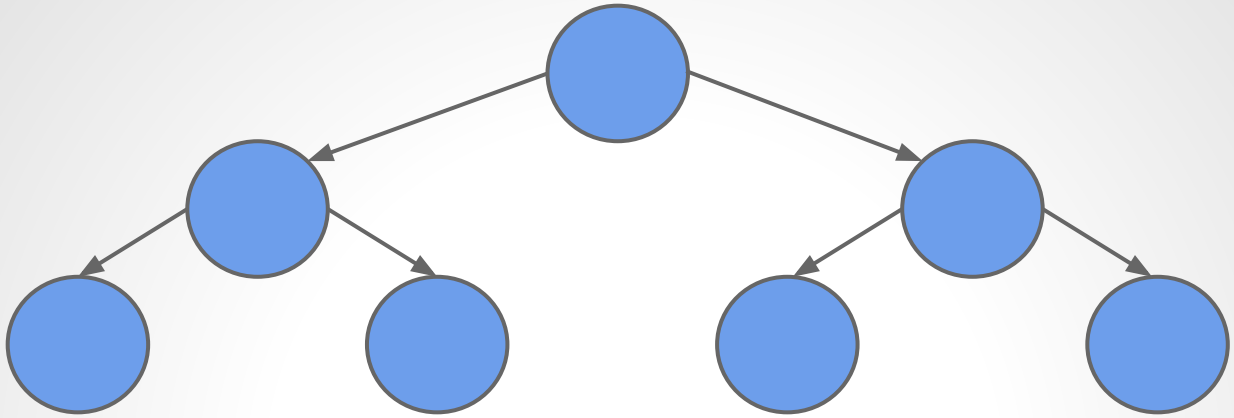
“Fix the error somewhere else”

Joe Armstrong,
http://www.infoq.com/presentations/self-heal-scalable-system

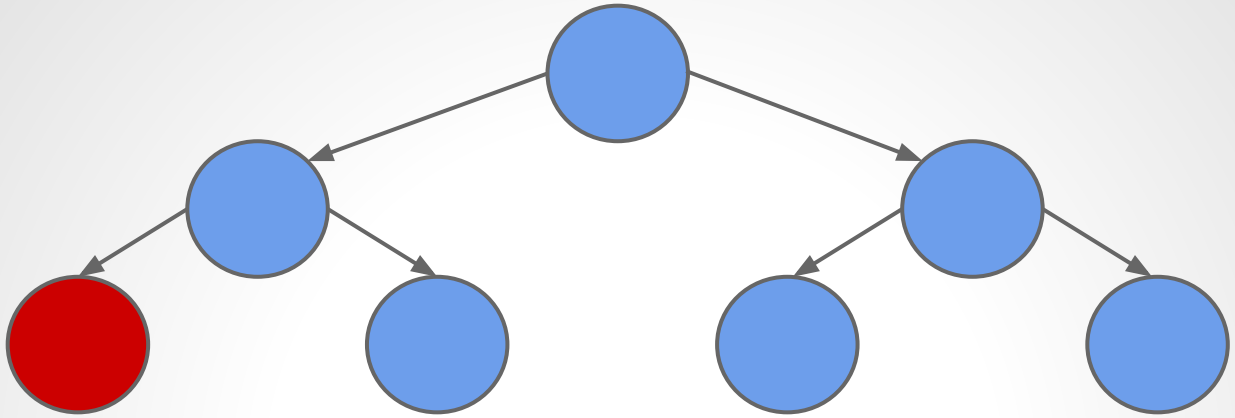
Supervisor Options

- **Restart** (can continue after recreation)
- **Resume** (same instance continues)
- **Stop** (will stop processing messages)
- **Escalate** (delegate to parent supervisor)

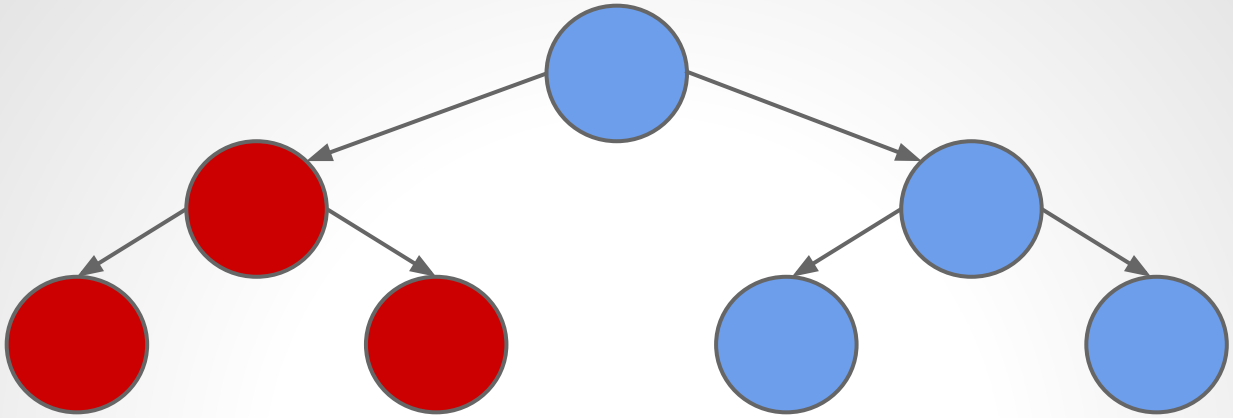
Fault tolerant architecture



Fault tolerant architecture



Fault tolerant architecture



Mailboxes

- **Unbounded**
  - **UnboundedMailbox**
    - *=> java.util.concurrent.ConcurrentLinkedQueue*
  - **Single consumer only**
- **Bounded**
- **Priority Mailboxes**
  - **bounded**
  - **unbounded**
- **Durable**
- **your own**

Dispatchers

- Dispatcher (Fork-Join)
- PinnedDispatcher
- BalancingDispatcher
- CallingThreadDispatcher
- your own

Routing

A *Router* is an Actor that acts like a proxy (and supervisor) for other Actors (*Routees*).

Routing is done on the calling thread!

Router implemenations

- RoundRobinRouter
- RandomRouter
- SmallestMailboxRouter
- BroadcastRouter
- ScatterGatherFirstCompletedRouter
- ConsistentHashingRouter
- your own

Remote Actors

akka.tcp://myActorSystem@192.168.0.0:2552/user/myActorName

New in 2.2: Clusters

- Basic features ready for use
- More planned
  - Partititions
  - Quorums
  - more
- Based on Gossip, similar to Cassandra

Futures

?

(it’s a method)

Rule number one

Don't block!

Rule number two

Isolate!

Rule number three

Keep actors simple!
(do one thing)

Book: *Akka in Action*

http://www.manning.com/roestenburg/

Book: *Akka Concurrency*

http://www.artima.com/shop/akka_concurrency

Book: *Java Concurrency in Practice*

http://jcip.net/

Further Information:

www.akka.io

Further Information:

https://github.com/MarkusJais