ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Failure Handling with Actors

Principles of Reactive Programming

Roland Kuhn

Failure Handling in Asynchronous Systems

Where shall failures go?

- reify as messages
- send to a known address

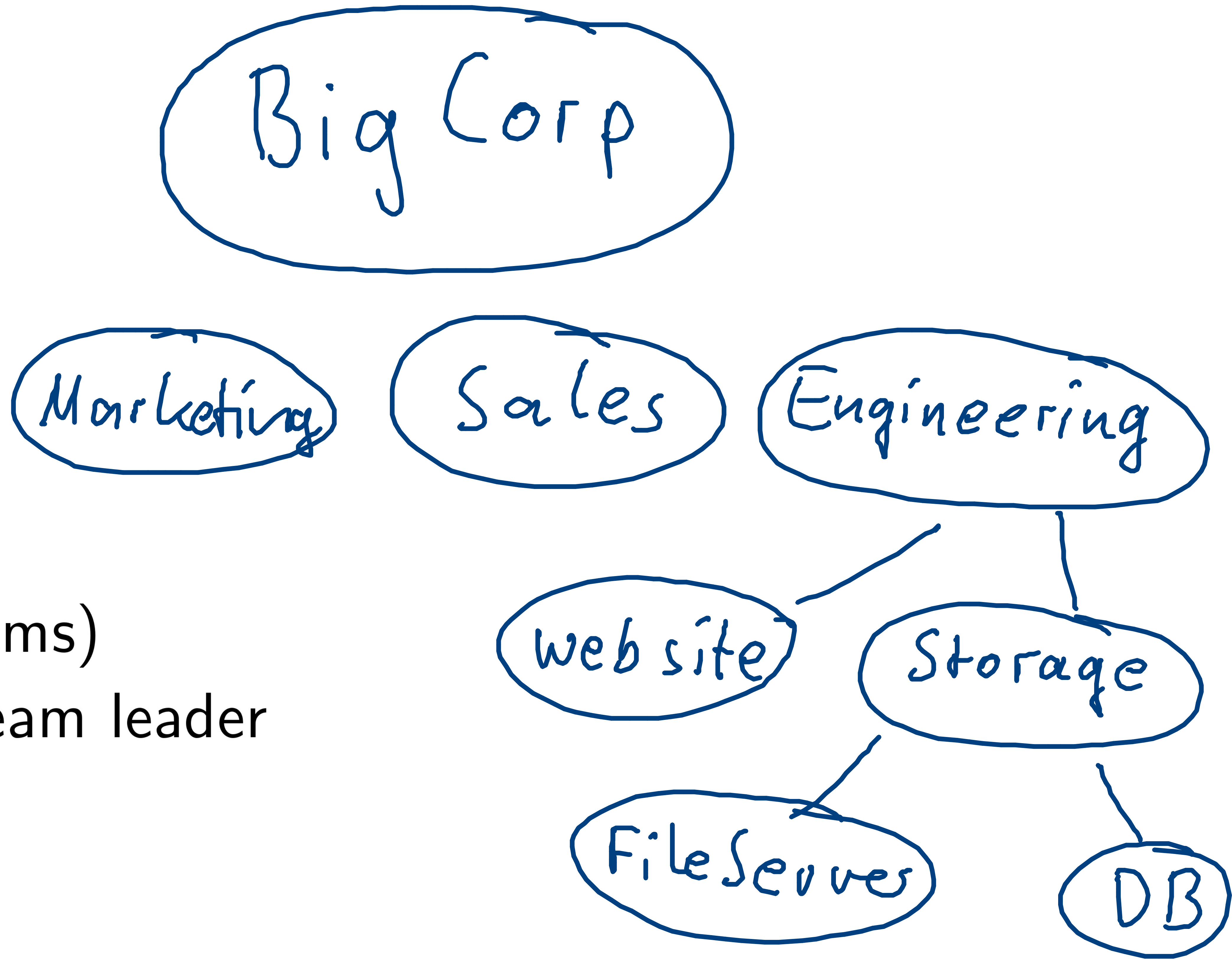
Failure Handling in Asynchronous Systems

Where shall failures go?

- reify as messages
- send to a known address

The Actor Model is anthropomorphic:

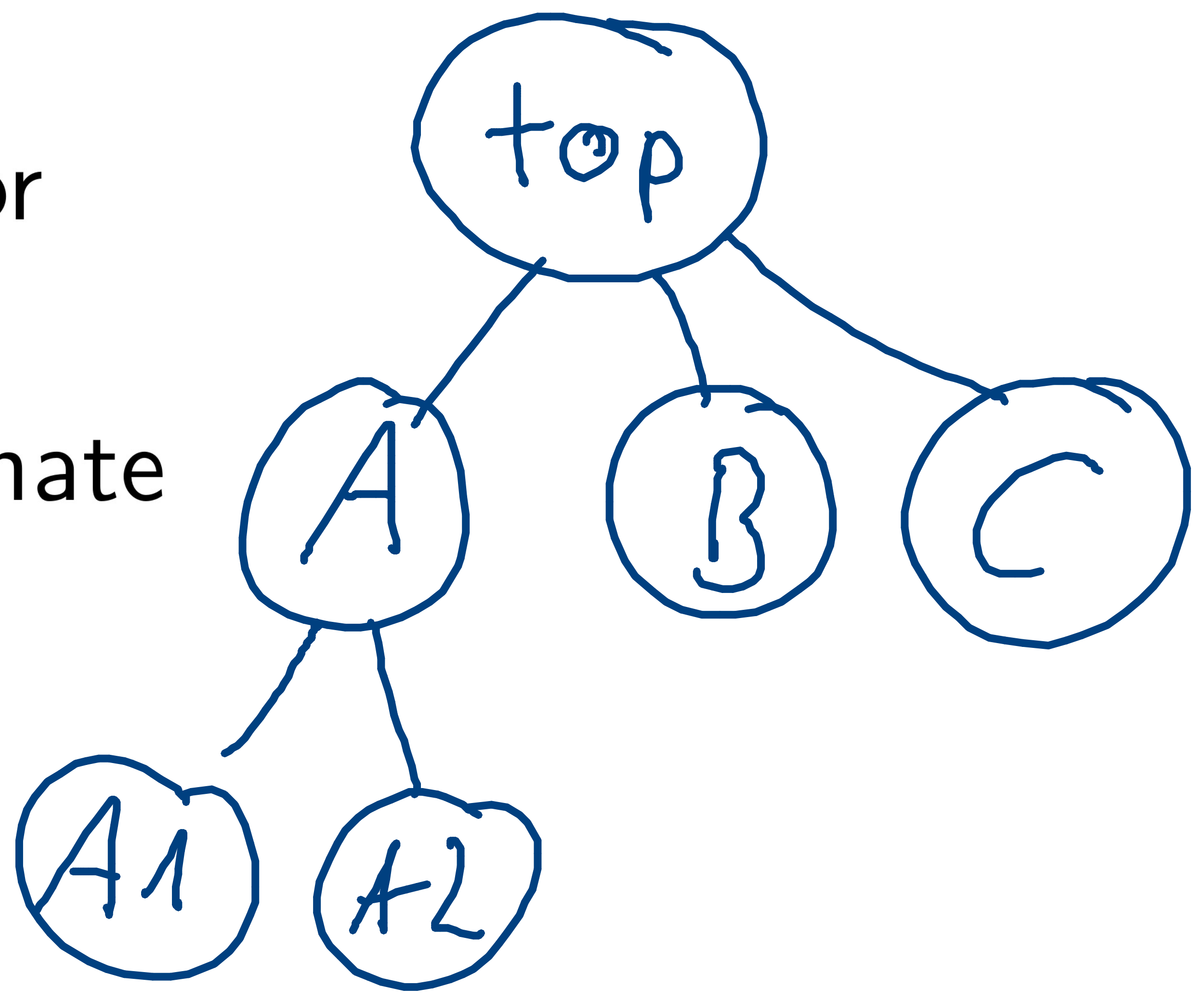
- Actors work together in teams (systems)
- individual failure is handled by the team leader



Supervision

Resilience demands *containment* of and *automatic response* to failure.

- failed Actor is terminated or restarted
- decision must be taken by one other Actor
- supervised Actors form a tree structure
- the supervisor needs to create its subordinate



Supervisor Strategy

In Akka the parent declares how its child Actors are supervised:

```
class Manager extends Actor {
  override val supervisorStrategy = OneForOneStrategy() {
    case _: DBException          => Restart // reconnect to DB
    case _: ActorKilledException => Stop
    case _: ServiceDownException => Escalate
  }
  ...
  context.actorOf(Props[DBActor], ”db”)
  context.actorOf(Props[ImportantServiceActor], ”service”)
  ...
}
```

Supervisor Strategy (cont'd)

Failure is sent and processed like a message:

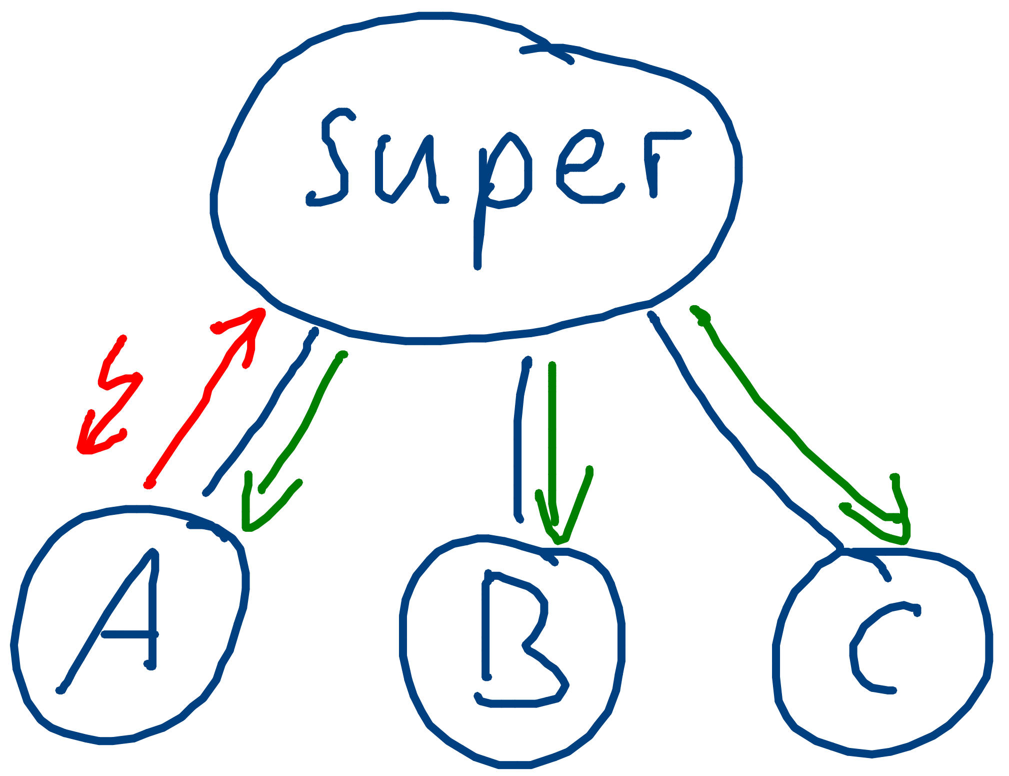
```
class Manager extends Actor {
  var restarts = Map.empty[ActorRef, Int].withDefaultValue(0)
  override val supervisorStrategy = OneForOneStrategy() {
    case _: DBException =>
      restarts(sender) match {
        case toomany if toomany > 10 =>
          restarts -= sender; Stop
        case n =>
          restarts = restarts.updated(sender, n + 1); Restart
      }
  }
}
```

Supervisor Strategy (cont'd)

If decision applies to all children: `AllForOneStrategy`

Simple rate trigger included:

- allow a finite number of restarts
- allow a finite number of restarts in a time window
- if restriction violated then `Stop` instead of `Restart`



Actor Identity

Recovery by restart requires stable identifier to refer to the service:

- in Akka the ActorRef stays valid after a restart
- in Erlang a name is registered for the current PID

Actor Identity

Recovery by restart requires stable identifier to refer to the service:

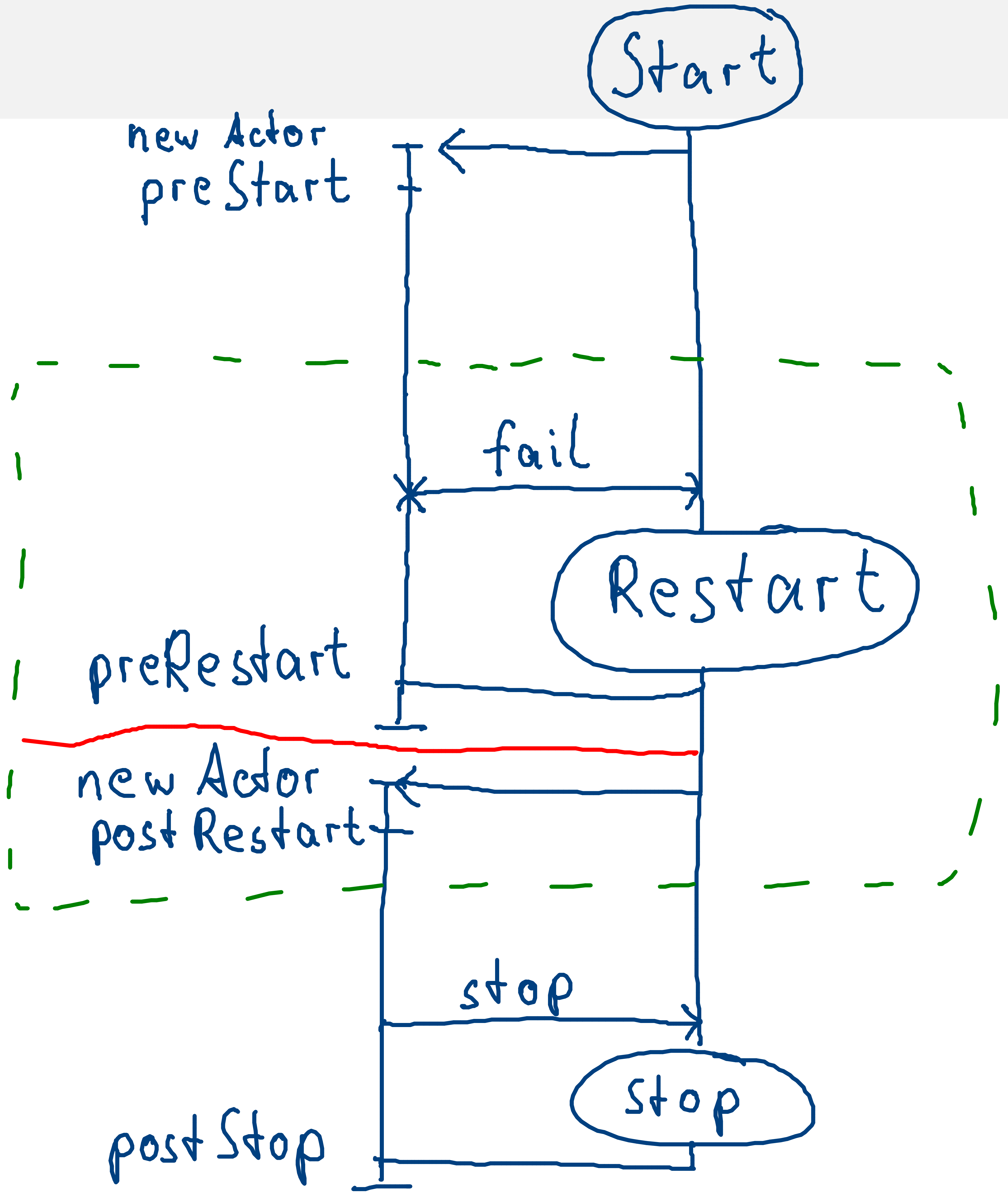
- in Akka the ActorRef stays valid after a restart
- in Erlang a name is registered for the current PID

What does restart mean?

- expected error conditions are handled explicitly
- unexpected error indicate invalidated actor state
- restart will install initial behavior / state

Actor Lifecycle

- start
- (restart)*
- stop



Actor Lifecycle Hooks

```
trait Actor {
  def preStart(): Unit = {}
  def preRestart(reason: Throwable, message: Option[Any]): Unit = {
    context.children foreach (context.stop(_))
    postStop()
  }
  def postRestart(reason: Throwable): Unit = {
    preStart()
  }
  def postStop(): Unit = {}
  ...
}
```

The Default Lifecycle

```
class DBActor extends Actor {
  val db = DB.openConnection(...)
  ...
  override def postStop(): Unit = {
    db.close()
  }
}
```

In this model the actor is fully reinitialized during restart.

Lifecycle Spanning Restarts

```
class Listener(source: ActorRef) extends Actor {
  override def preStart() { source ! RegisterListener(self) }
  override def preRestart(reason: Throwable, message: Option[Any]) {}
  override def postRestart(reason: Throwable) {}
  override def postStop() { source ! UnregisterListener(self) }
}
```

Actor-local state cannot be kept across restarts, only external state can be managed like this.

Child actors not stopped during restart will be restarted recursively.