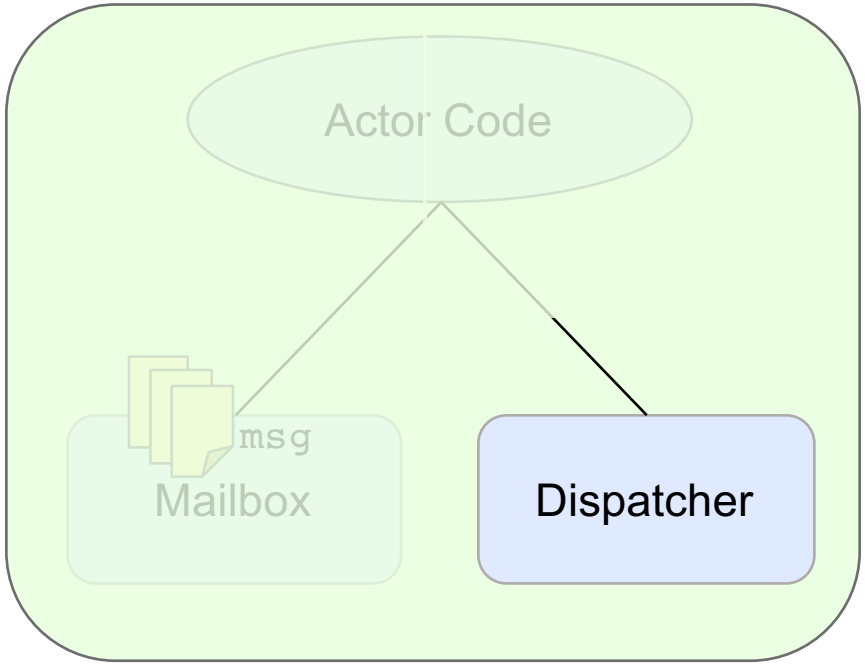
Dispatchers

Dispatchers

- Dispatcher forms the engine of an Akka application
  - provides an Execution Context for actors
  - supports Futures
  - allows internal scheduling of tasks
- Implemented using JVM concurrency constructs
  - fork-join executor
  - thread pool
- Highly configurable



Working with Dispatchers

- `ActorSystem` has a default dispatcher
  - built on fork/join executor
  - uses between 8 and 64 threads
- More dispatchers can be added
  - more execution contexts
  - associated with actors
  - different types for different execution characteristics
- Can localise failures or performance issues
  - "Bulkheading"

Dispatcher Types

- `Dispatcher`
  - default
  - event based
  - binds a group of actors to a thread pool
- `PinnedDispatcher`
  - associates a unique thread with each actor
  - normally thread pool containing a single thread
- `BalancingDispatcher`
  - attempts to redistribute work from busy actors to idle actors
  - works with actors of the same type only
- `CallingThreadDispatcher`
  - for testing

Dispatchers Example

- Default configuration

```
object DispatchersProg extends App {
```

```
case class SimpleMsg(c: Int)

class SimpleActor extends Actor with ActorLogging {
  log.info("Creating")
  override def receive = {
    case SimpleMsg(counter) =>
           Thread sleep (scala.util.Random.nextInt(10)*100);
           log.info(s"Msg: $counter");
  }
```

Dispatchers Example

- Default configuration

```
object DispatchersProg extends App {

  val aSystem = ActorSystem("Dispatchers")
  val actor1 = aSystem.actorOf(Props[SimpleActor], "actor1")
  1 to 10 foreach ( actor1 ! SimpleMsg(_) )
}
```

```
[INFO] … [Dispatchers-akka.actor.default-dispatcher-2] …
Creating
[INFO] … [Dispatchers-akka.actor.default-dispatcher-2] … Msg: 1
[INFO] … [Dispatchers-akka.actor.default-dispatcher-2] … Msg: 2
…
```

Dispatchers Example

- Default configuration

```
val actor1 = aSystem.actorOf(Props[SimpleActor]
                             .withDispatcher("my-dispatcher"),
                              "actor1")
```

```
my-dispatcher {
 type = Dispatcher
 executor = "thread-pool-executor"
 thread-pool-executor {
  core-pool-size-min = 2
  core-pool-size-factor = 2.0
  core-pool-size-max = 4
 }
 throughput = 1
}
```

No of messages processed on thread before moving to next thread

```
[INFO] … [Dispatchers-my-dispatcher-5] … Creating
[INFO] … [Dispatchers-my-dispatcher-5] … Msg: 1
…
```

The Scheduler

- Allows activity to be scheduled for future execution
  - once only or repeated
- Requires Execution Context
  - normally `Executor`
  - use `ActorSystem` dispatcher

```
import context.dispatcher
import scala.concurrent.Duration._
val pinger = context.system.scheduler.schedule(
                                    0 milliseconds,
                                    500 milliseconds,
                                    self,
                                    Ping )
```

Get access to executor

Delay before start

Delay between firing

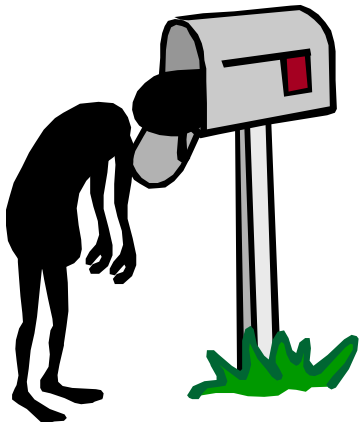
Actor to send to

Message to send

  - `cancel` method stops repetition
- Use `scheduler.scheduleOnce` for one time task

Dead Letters

- Messages sent to an actor after it has terminated or died
  - or when a bounded mailbox is full
- Messages wrapped in `DeadLetter` message
  - contains original sender, receiver and message
- Published on special `ActorSystem` event stream
- Can listen for `DeadLetters`



Listening for DeadLetter Messages

- Subscribe to `ActorSystem` event stream

```
val dlListener = aSystem.actorOf(Props(new Actor {
   def receive = {
     case d: DeadLetter => println(
            s"Dead Letter: From ${d.sender}
                        To ${d.recipient} Msg: ${d.message}")
   }
 })
)
aSystem.eventStream.subscribe(dlListener, classOf[DeadLetter])
…
```

Listening for DeadLetter Messages

- Detecting dead letters

```
…
1 to 5 foreach ( actor1 ! SimpleMsg(_) )
aSystem.stop(actor1)
Thread sleep 1000     // Give the actor time to stop…

1 to 5 foreach ( actor1 ! SimpleMsg(_) )
…
```

```
[INFO] … Msg: 1
[INFO] … Msg: 2
[INFO] … Msg: 3
[INFO] … Msg: 4
[INFO] … Msg: 5
Dead Letter: From Actor[akka://Dispatchers/deadLetters] To
         Actor[akka://Dispatchers/user/actor1] Msg: SimpleMsg(1)
Dead Letter: From Actor[akka://Dispatchers/deadLetters] To
         Actor[akka://Dispatchers/user/actor1] Msg: SimpleMsg(2)
…
```