

# Scalaz Actors

---

Tom Adams

*With copious plagiarism from Rúnar Bjarnason*

# traditional concurrency

# In the past...

- ▶ Manual creation of threads
- ▶ Manual synchronisation
- ▶ One thread per process
- ▶ Processes communicate by shared mutable state

# Problems

- ▶ Threads do not compose
- ▶ Threads are expensive
- ▶ Too low level
- ▶ Too complicated
- ▶ Error prone
- ▶ New(er) APIs [1] cannot compose w/out blocking threads

[1] `java.util.concurrent.Future`

# background...

# Actors

- ▶ Light-weight thread-like process, that communicates by asynchronous messaging

# Actors

- ▶ Everything is an actor
- ▶ Actors have addresses
- ▶ An actor, in response to a message it receives, can:
  - ▶ send a finite number of messages to other actors
  - ▶ create a finite number of new actors
  - ▶ designate the behaviour to be used for the next message it receives

# Actors

- ▶ Has a mailbox
- ▶ Communicate only using messages (no shared state)

# Actors

- ▶ Processes receive messages on a queue
- ▶ Messages can be enqueued with no waiting
- ▶ An actor is always either suspended (waiting for messages) or working (acting on a message)
- ▶ An actor processes messages in some (but any) order

# scalaz actors

# Scalaz Actors

- ▶ Not Scala.actors
  - ▶ Simpler, just the essentials
  - ▶ Messages are typed
- ▶ Actor is sealed, and instantiated by supplying:
  - ▶ type A
  - ▶ effect: A => Unit
  - ▶ (implicit) strategy: Strategy[Unit]
  - ▶ (Optional) Error Handler: Throwable => Unit

strategy + effect = actor

# Strategy

- ▶ Abstracts over ways of evaluating expressions concurrently
- ▶ **Executor** - Evaluates the expression using an `ExecutorService`
- ▶ **Naive** - Starts a new thread for each expression
- ▶ **Sequential** - Evaluates each expression in the current thread (no concurrency)
- ▶ **Identity** - Performs no evaluation
- ▶ Can roll your own

# Effect

- ▶ A => Unit
- ▶ Does the work

# Error handler

- ▶ `Throwable => Unit`
- ▶ Optional, default throws the exception
- ▶ `(err : Throwable) => supervisor ! (actor, err)`

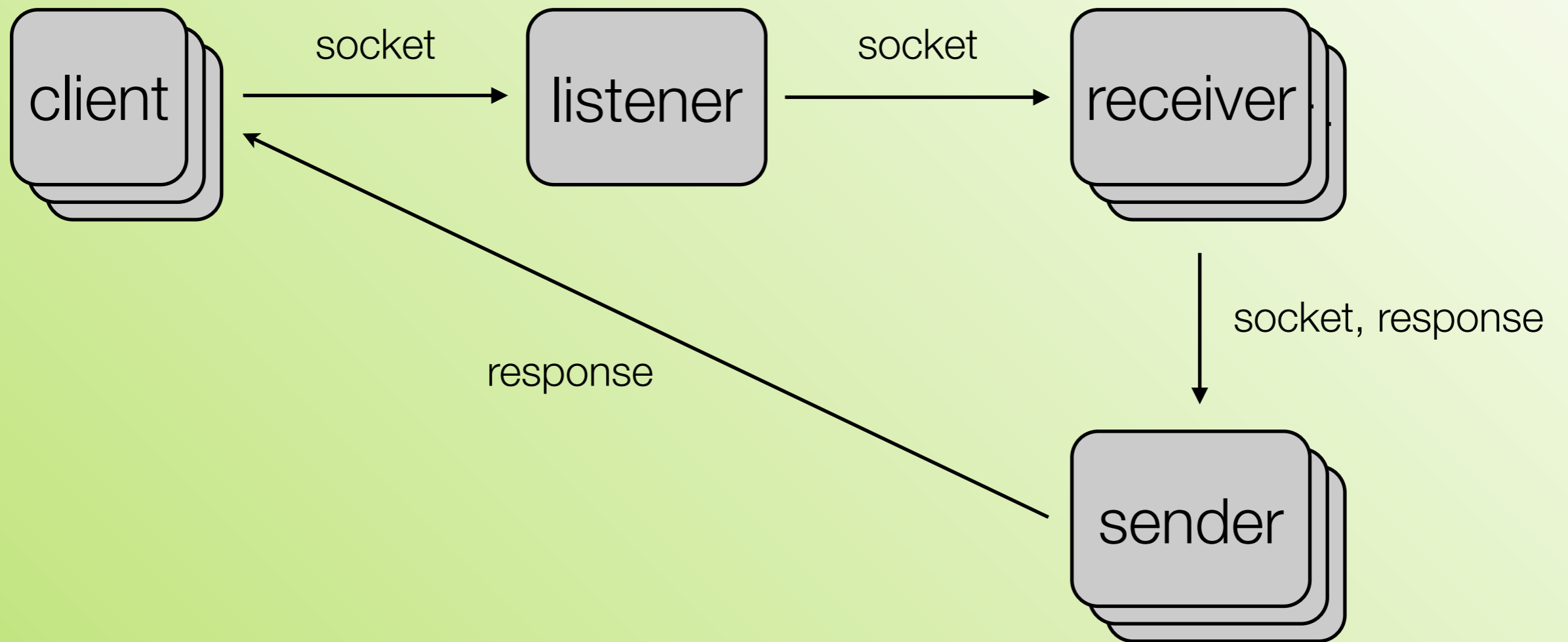
# Creating an actor

- ▶ `actor[A](err: Throwable => Unit, c: A => Unit)`
- ▶ `actor[A](c: A => Unit)`

# Sending messages

- ▶ Messages are sent to an actor using the ! “operator” (function)
- ▶ `actor ! message`

# example

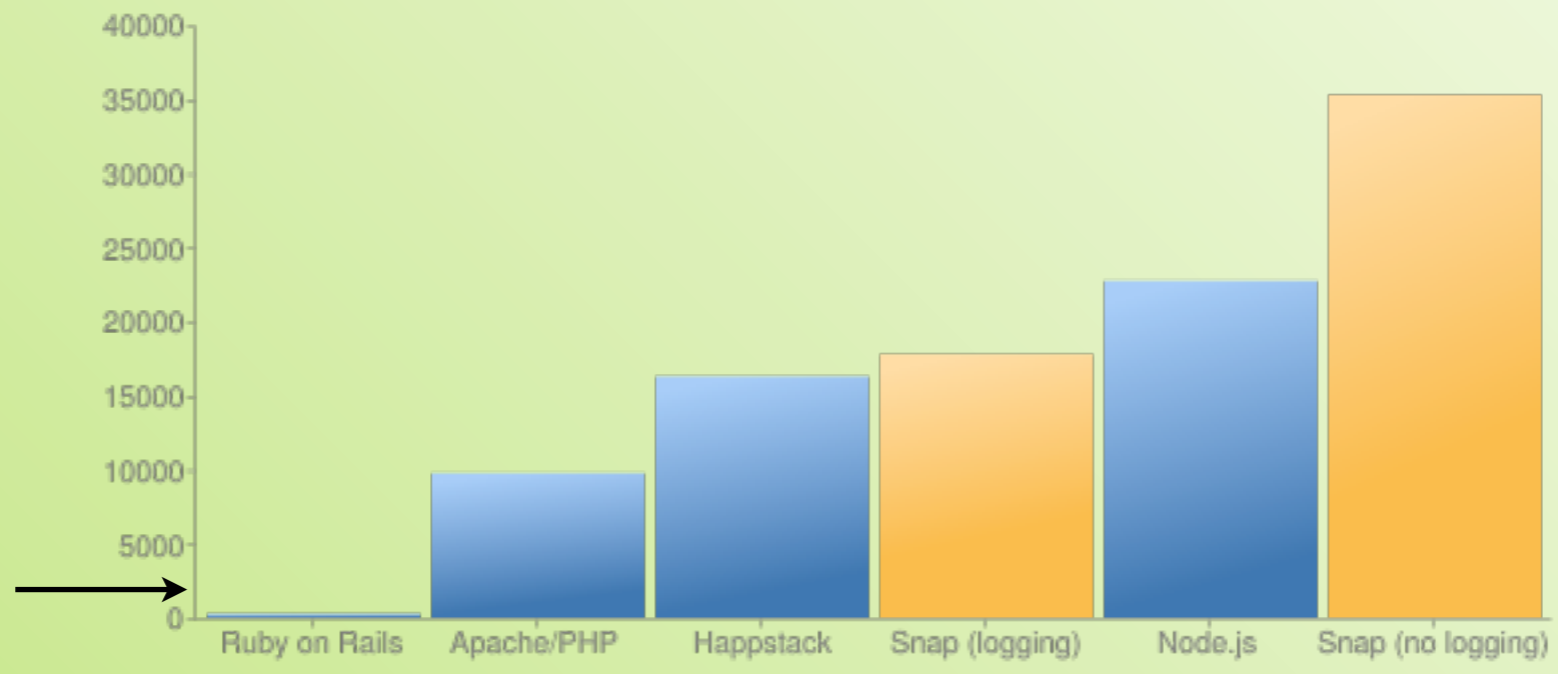


# the rest

# The Rest

- ▶ Lightweight
- ▶ No remote support
- ▶ No supervisor hierarchies
- ▶ `scalaz.concurrent.Promise`

# food for thought



you are here

Source: <http://snapframework.com/blog/2010/11/17/snap-0.3-benchmarks>

# references

# Further Reading

- ▶ Scalaz
  - ▶ <http://code.google.com/p/scalaz/>
  - ▶ Beyond Mere Actors: [http://docs.google.com/present/view?id=ddmk3f43\\_63zpg3jcgz](http://docs.google.com/present/view?id=ddmk3f43_63zpg3jcgz)
  - ▶ <http://stackoverflow.com/questions/tagged/scalaz>
- ▶ Corral: <https://github.com/tomjadams/corral>
- ▶ Scala Actors
  - ▶ Tutorial: <http://www.scala-lang.org/node/242>

# Contact



Tom Adams

Chief technologist & co-founder

+61 437 701 592

[tom@mogeneration.com](mailto:tom@mogeneration.com)