Erlang

Aurojit Panda
apanda@cs.berkeley.edu
Who Uses Erlang*

*riak

CouchDB

relax

Amazon SimpleDB

RabbitMQ

*and is cool
Erlang as a Language

• Functional
• Designed to support applications which are
  • Distributed
  • Fault-tolerant
  • Soft-realtime
  • “Non-stop”
• Relies on Actors to provide concurrency
• Predates the cloud :)
History

• Built by Joe Armstrong and others
  • Initially at Ericsson

• First built in 1986 on top of Prolog

• Designed to help build concurrent, fault tolerant applications
  • Actually the design was solving a very concrete problem for telephony

• Open sourced in 1998 along with OTP
Stated Philosophy

• Processes are independent, no difference between processes on one machine, or on multiple machines

• Processes share nothing. Communicate through messages

• Processes have unforgeable names.

• If you know a process name, can send it a message.

• Send-and-pray message passing

• Processes can monitor remote processes
An actor is an entity which in response to a message can (with no implied order)
- Send out a finite set of messages
- Create a finite set of new actors
- Designate what it should do when it next receives a message
- Messages don’t carry the identity of the sender
Actors (contd...)

• The claim is that based on those simple rules Erlang can accomplish its design goals
• Ignoring soft-real time since that’s a property of the runtime
• Distributed is obvious
• “Non-stop” is basically covered by “hot swapping” one actor for another
• Essentially designate a new message handler
Fault Tolerance with Actors

• Actors make fault tolerance easy in some sense

• Erlang provides a mechanism to receive a message on the death of a certain process

• Thus chaining processes

Image from http://learnyousomeerlang.com/supervisors#from-bad-to-good
OTP

- The Open Telephony Platform makes a lot of these concepts easier to use.
- A set of library functions and behaviors based on common code patterns
- Designed and used by the folks at Ericsson
- Makes a lot of what was discussed before easier to use
Future

- Erlang is reasonably popular for some applications
- Akka (akka.io) and other frameworks are now adopting similar ideas
- From the Akka site:
  
  “Embrace failure. Write applications that self-heal using Erlang-style Actor supervisor hierarchies.”

- Erlang claims it philosophically avoids defensive programming, rather embracing failure
Acknowledgments

• Some of the content was stolen from various talks by Joe Armstrong’s slides.

• Some of my information came from “Programming Erlang” by Joe Armstrong

• Some of the other information, and a picture, came from http://learnyouosomeerlang.com/content
The End