Erlang Solutions Ltd.

The Erlang Rationale

Robert Virding

A Rationale

Rationale – n. 1. Fundamental reasons; the basis. 2. An exposition of principles or reasons.

Why would we want one?

- Help users understand how/why to use various features
- Help language designers
- Help implementors
- Help people wishing to extend language

First Principles

- Lightweight concurrency
  - Must handle a large number of processes
  - Process creation, context switching and inter-process communication must be cheap and fast.
- Asynchronous communication
- Process isolation
  - What happens in one process must not affect any other process.
- Error handling
  - The system must be able to detect and handle errors.
- Continuous evolution of the system
  - We want to upgrade the system while running and with no loss of service.

First Principles

Also

- High level language to get real benefits.
- The language should be simple
  - Simple in the sense that there should be a small number of basic principles, if these are right then the language will be powerful but easy to comprehend and use. Small is good.
  - The language should be simple to understand and program.
- Provide tools for building systems, not solutions
  - We would provide the basic operations needed for building communication protocols and error handling

Trivial code example

```
ringing_a_side(Addr, B_Pid, B_Addr) ->
    receive
        on_hook ->
            B_Pid ! cleared,
            tele_os:stop_tone(Addr),
            idle(Addr);
        answered ->
            tele_os:stop_tone(Addr),
            tele_os:connect(Addr, B_Addr),
            speech(Addr, B_Pid, B_Addr);
        {seize,Pid} ->
            Pid ! rejected,
            ringing_a_side(Addr, B_Pid, B_Addr);
        _ ->
            ringing_a_side(Addr, B_Pid, B_Addr)
    end.
```

Trivial code example

```
ringing_b_side(Addr, A_Pid) ->
    receive
        cleared ->
            tele_os:stop_ring(Addr),
            idle(Addr);
        off_hook ->
            tele_os:stop_ring(Addr),
            A_Pid ! answered,
            speech(Addr, A_Pid, not_used);
    {seize,Pid} ->
            Pid ! rejected,
            ringing_b_side(Addr, A_Pid);
    _ ->
            ringing_b_side(Addr, A_Pid)
end.
```

Erlang “Things”

Only two basic types of things in Erlang

- Immutable data structures
  - Normal Erlang terms
- Processes
  - Everything with internal state

- Yes, the process dictionary is a mutable data structure but not the data in it, and we never really liked it!

Processes

- A process is something which obeys process semantics:
  - Parallel independent execution
  - Communicates through asynchronous message passing
  - Links/monitors for error detection/handling
  - Obey/transmit exit signals

N.B. Implementation and internal details irrelevant!

Processes

- Everything is run within a process
- All processes are equal – no special or system processes
- No process hierarchy – flat process space

- Processes are used for many things
  - Concurrency
  - Managing state

Process communication

- ***All*** process communication by messages
- ***All*** process communication asynchronous
- Process BIFs asynchronous
  - Only check arguments
  - One exception then: sending to registered name!
- Works the same with distribution!

Ports

- ”Processes” for communicating with the outside world
- Obey process semantics
  - Message based interface
  - Obeys links and exit signals
  - Fits in with rest of erlang
- Ports talk to hardware
- Ports need connected process to communicate with.

Error handling

Errors will ALWAYS occur!

Error handling

- Robust systems must always be aware of errors
- Want to avoid writing error checking code everywhere
- Want to be able to handle process crashes among cooperating processes
- System must detect, contain and handle errors
- Interact well with process communication

Error handling

We just want to

Let it crash!

Error handling

- Process based
- If one process crashes then all should crash
  - Cooperating processes are *linked* together
  - Process crashes propagate along links

- ”System” processes can monitor them and restart them when necessary
- But sometimes we do need to handle errors locally

Modules, code and code loading

- Only compiled code
- Module is both the unit of compilation and of all code handling
  - Relatively efficient compilation
  - More consistent system when loading code
- Multiple versions of a module

- No inter-module dependencies

Modules, code and code loading

- All functions belong to a module
- All modules are equal – no system or special
- No module hierarchy – flat module space

Things missing in early Erlang

- Code handling
- Binaries
- ETS
- Funs
- OTP

- NIFs

Distribution

- Based on loosely coupled nodes – like processes
- Completely transparent if desired (almost true)
- Easier with asynchronous communication, so keep communication and error handling asynchronous

Patterns, pattern matching, guards

- Patterns are a Big Win™ and ubiquitous.
- Data constructors and patterns are the same.
  - Rule not broken by new data types!
- Guards added to provide simple tests for extending pattern matching.
- Adding boolean operators a Good Thing but has has made the difference between guard tests and expressions less distinct.

Variables, scoping and =

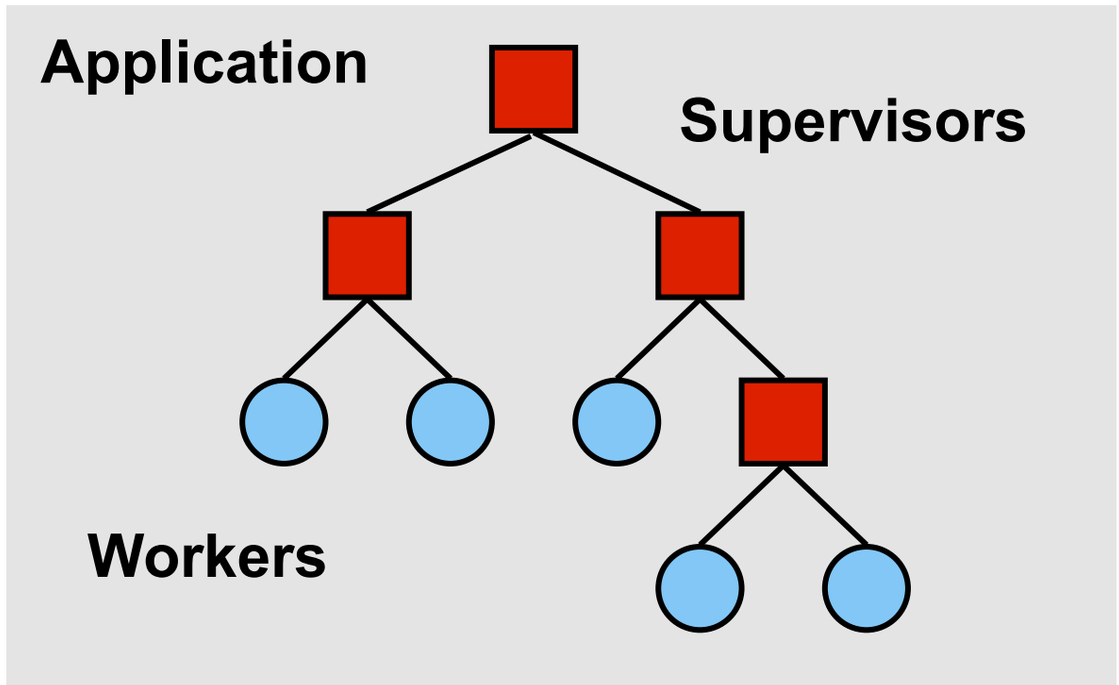
- Variables are just bind-once references to values
- Also inherited Prologs scoping, or rather lack of scoping, a variable's scope is the whole function clause
- Affects pattern matching as already occurring variables means testing existing value
- = started its life as simple assignment
  - Practical to use it to pull apart return values

OTP (Open Telecoms Platform)

Erlang just a language, for building large scale applications you need:

- A large set of standard libraries
- A set of rules and design patterns for building robust systems
  - Generic behaviours
- And patterns for building new behaviours
- Tools

OTP (Open Telecoms Platform)



An application, its supervision tree and its workers

- Supervisors ensure robust system by restarting workers

Thank you

Robert Virding: robert.virding@erlang-solutions.com