TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

Programming Distributed Systems

Erlang's design principles

Annette Bieniusa, Albert Schimpf

FB Informatik
TU Kaiserslautern

Summer Term 2020

Problem domain

- Need to handle very large number of **concurrrent** activities
- **Soft real-time**: Tasks need to be handled within a specific time
- **Distributed** over several computers
- **Continuous operations** for many years
  - Maintenance without stoppping the system
- **Fault tolerance**

Design goals[1]

- Organize system as set of communicating processes
- Processes are identifiable by unique, unforgeable Pid
- No sharing of data between processes
- Supports both multi-processor and distributed systems
  - *Caveat:* Might be inefficient if strong data dependencies between processes
- Message passing is assumed to be asynchronous, but ordered and atomic
- Must be able to identify failure in a process (and its potential reason)

Erlang View of the World[1]

1. Everything is a process.
2. Processes are strongly isolated.
3. Process creation and destruction is a lightweight operation.
4. Message passing is the only way for processes to interact.
5. Processes have unique names.
6. If you know the name of a process you can send it a message.
7. Processes share no resources.
8. Error handling is non-local.
9. Processes do what they are supposed to do or fail.

Erlang's Concurrent programming in a Nutshell

```
Pid = spawn(F)

Pid ! Msg

receive
    MsgPattern1 -> Expr1;
    MsgPattern2 -> Expr2;
    MsgPattern3 -> Expr3;
    ...
    % optional timeout
end
```

The Actor model[2]

- Mathematical model of concurrent computation
- Erlang's philosophy is very close to the actor model
  - Differences: FIFO ordering of message sending, processes are sequential, ...

Actor

- Computational entity that can
  - send finite number of messages to other actors it knows
  - create finite number of new actors
  - designate what to do with the next message

Further reading I

[1] Joe Armstrong. „Making reliable distributed systems in the presence of software errors“. Diss. Royal Institute of Technology, Stockholm, Sweden, 2003. URL: http://erlang.org/download/armstrong_thesis_2003.pdf.

[2] Carl Hewitt, Peter Boehler Bishop und Richard Steiger. „A Universal Modular ACTOR Formalism for Artificial Intelligence“. In: *Proceedings of the 3rd International Joint Conference on Artificial Intelligence. Standford, CA, USA, August 20-23, 1973*. Hrsg. von Nils J. Nilsson. William Kaufmann, 1973, S. 235–245. URL: http://ijcai.org/Proceedings/73/Papers/027B.pdf.