Introduction to Erlang Platform

Ali Hassan
Full stack engineer @ convo

Agenda

- Where it shines?
- Erlang history
- Erlang VM
- Erlang Language
- OTP (Open Telecom Platform)
- Who uses it?
- Resources for learning

Where it shines?

- Backend systems. (which are mostly IO bound)

e.g.

- Request handling
- Messaging systems
- Microservices
- Databases
- Background jobs
- Proxies

Erlang not suitable for

Systems which are CPU bound

e.g.

- Number crunching applications
- Graphics intensive systems

History

- It was created in computer science lab at Ericsson
- Their goal was to find a most suitable language for writing telecom systems.
- Their team implemented telecom systems in multiple languages. E.g. ML, Miranda, Smalltalk, Prolog, Ada etc.
- No single language had all the properties required for telecom systems.

History

Requirements for telecom systems

- Time to market
- Cocurrent
- Fault Tolerant
- Soft real time
- Distributed
- Declarative and Functional
- Inspect running systems
- Hot code swapping

History

- Erlang , like C, was created out of a need.
- C was created to write softwares in high-level language than assembler.
- Erlang was created to write concurrent, soft real time scalable and fault tolerant systems.
- Both languages were used internally for nearly a decade before general public heard about them.

History

- Ericsson deployed its first Erlang based product a telephony switch for British Telecom.

“the network performance has been so reliable that there is almost a risk that our field engineers do not learn maintenance skills.”

Why Erlang?

- Today’s internet systems requirements match with 1980’s telecom systems.

- Multi Core CPUs

Erlang Platform

- Erlang virtual machine
- Erlang Language
- OTP(Open Telecom Platform) framework

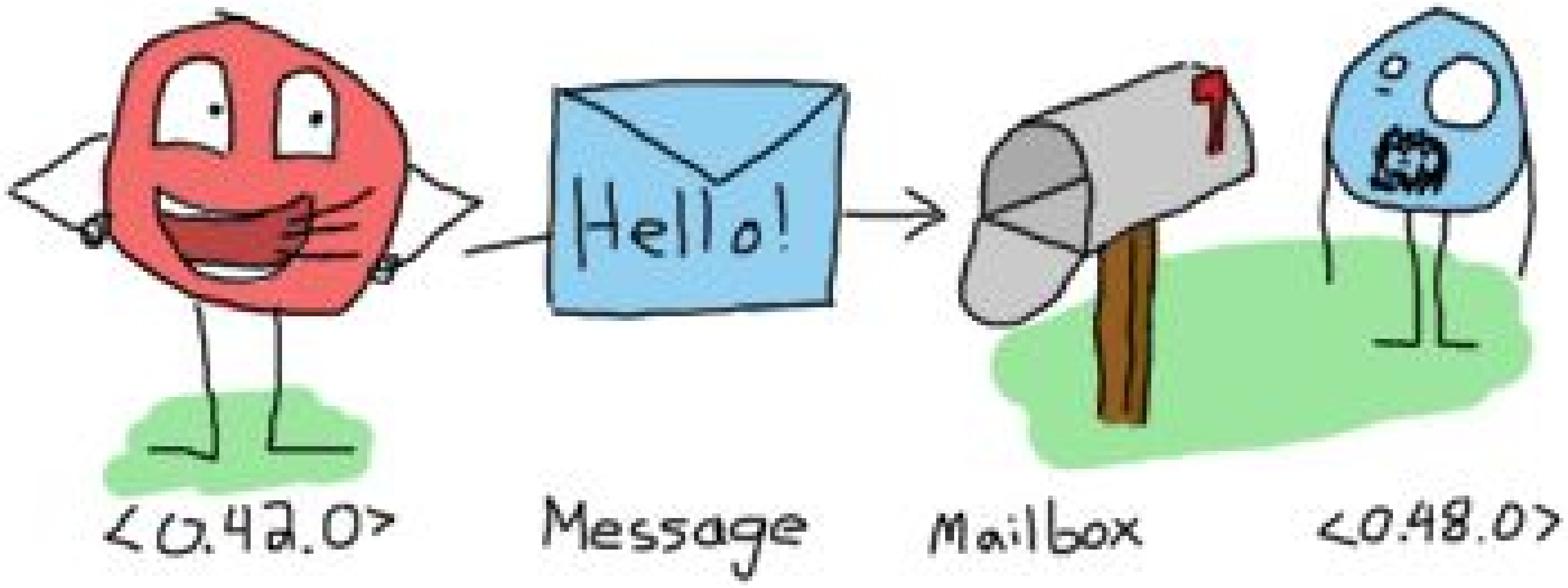
Erlang VM

- Lightweight massive concurrency using processes
- Asynchronous communication (message passing)
- Process Isolation
- Per process garbage collection
- Process linking
- Process scheduler
- Distribution
- Live updates
- Runtime inspection of systems running in production.

Erlang VM: Processes

- Lightweight processes managed by the VM
- Not mapped on OS thread/processes.
- No sharing between processes
- Communicate via message passing.
- Message passing is asynchronous
- Each processes has mailbox, where messages are received.

Erlang VM: Processes

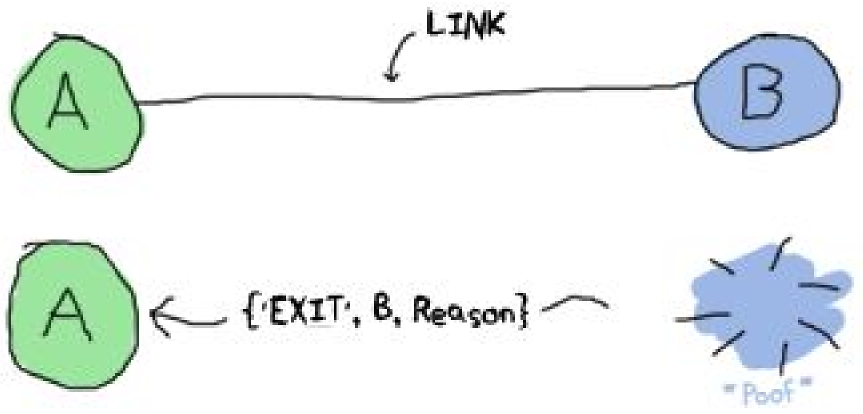


Erlang VM: Soft real time

- Per process garbage collection
- Erlang scheduler is preemptive.
- IOs are event based and do not block.
- Long running processes cannot stale others.
- Each process has reductions count, which determines for how much time this process can run before being suspended by scheduler.
- Downside: This behavior make erlang less ideal for cpu bound tasks.

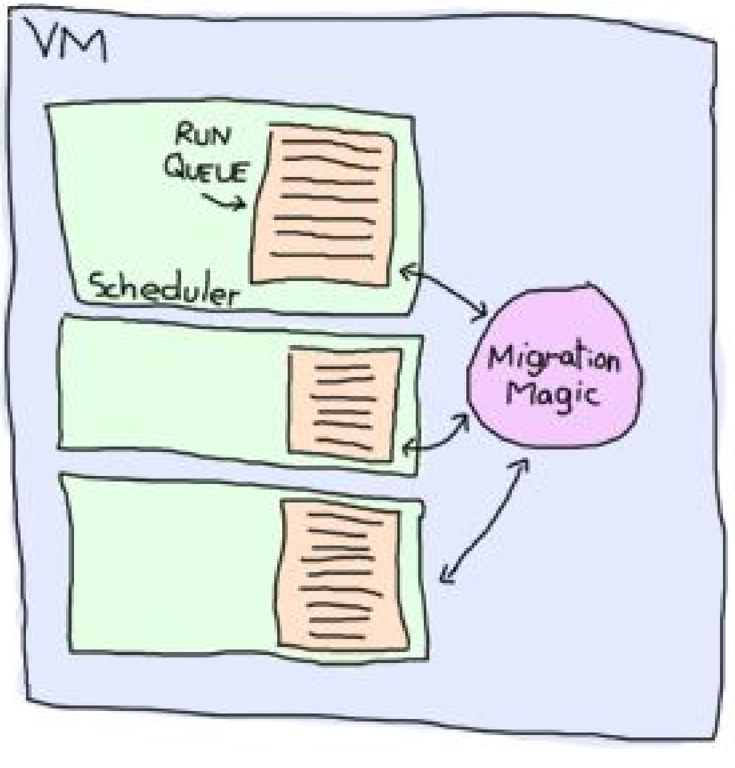
Erlang VM: Process Linking

- Two process can link together i.e. If one fails other receives a message.
- Process linking helps make app fault tolerant.



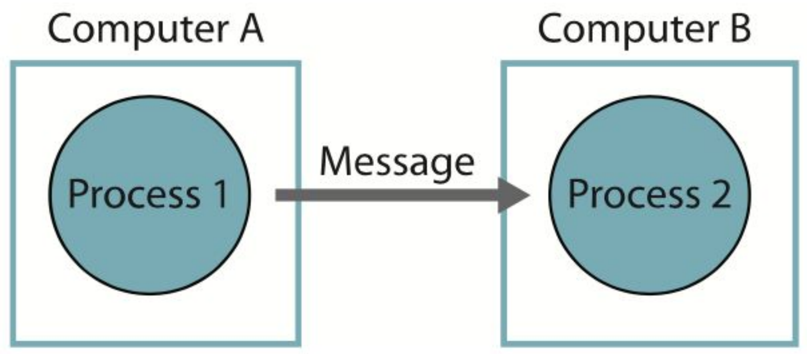
Erlang VM: Scheduler

- VM creates many schedulers equal to number of CPU cores.
- Fully utilize multi core
- Run queue per scheduler.



Erlang VM: Distribution

- VMs running on different machines connect with each other to form a cluster
- Nodes can be added and removed any time
- Location transparency: Same semantics in cluster mode.



Erlang VM: Live updates & Inspection

- Hot code swapping: running system can be updated without taking them down
- Provide tools to connect to a running system and inspect processes, mailboxes and messages.
- Helps fight heisenbugs

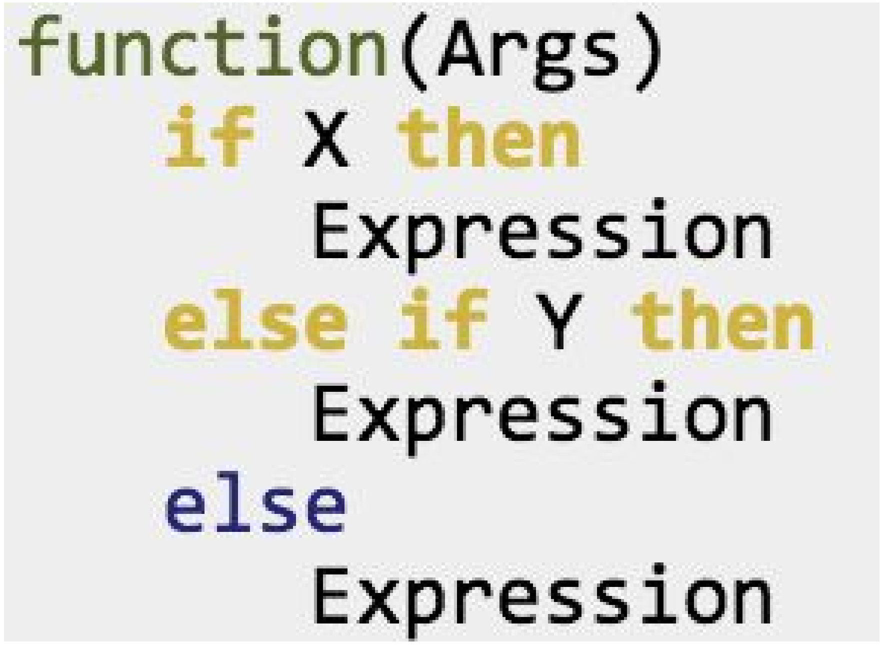
Erlang Language

- Declarative
- Functional
- Pattern matching
- Tail recursion
- Immutability

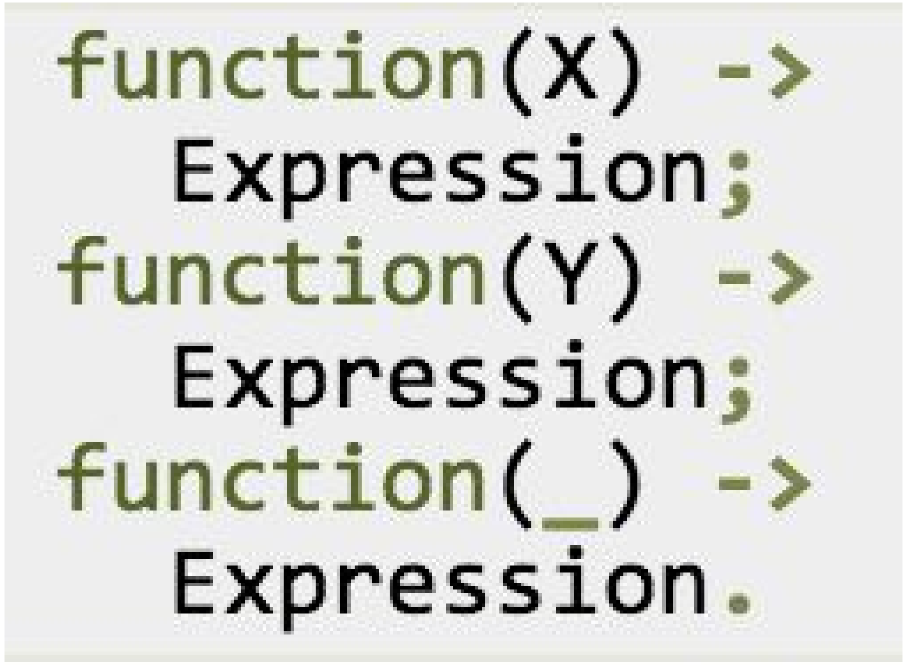
Erlang Language: Data types

- Numbers (integers and floats)
- Binaries/Bitstrings
- Atoms
- Tuples
- Lists (and strings)
- Records
- Unique identifiers (pids, ports, references)
- Funs

Erlang Language: Pattern Matching



```
function(Args)
   if X then
      Expression
   else if Y then
      Expression
   else
      Expression
```



```
function(X) ->
  Expression;
function(Y) ->
  Expression;
function(_) ->
  Expression.
```

Erlang Language: Tail Recursion

- No loop construct
- Looping is achieved using recursion.
- VM is optimized for tail recursive function.

```
sum_acc([],Sum) -> Sum;

sum_acc([Head|Tail], Sum) ->
          sum_acc(Tail, Head+Sum).
```

Erlang Language: Modules

- Code is written in modules
- Modules contain functions
- Only exported functions are accessible from other modules.

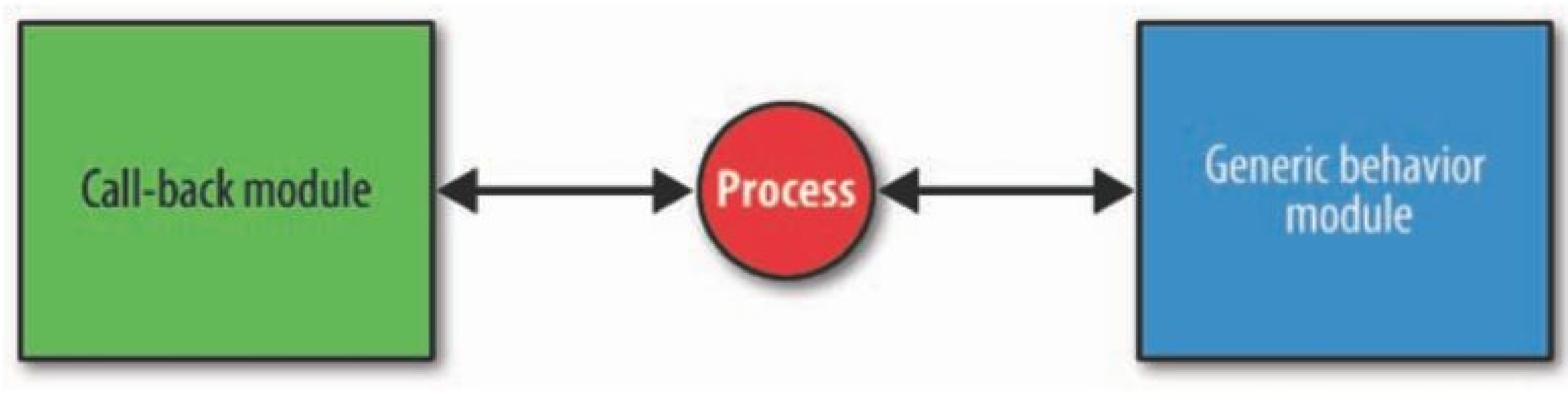
Erlang Language: Hello World

```
-module(hello_module).
-export([hello/0]).
hello() ->
        io:format(“Hello World!~n”).
```

OTP (Open Telecom platform)

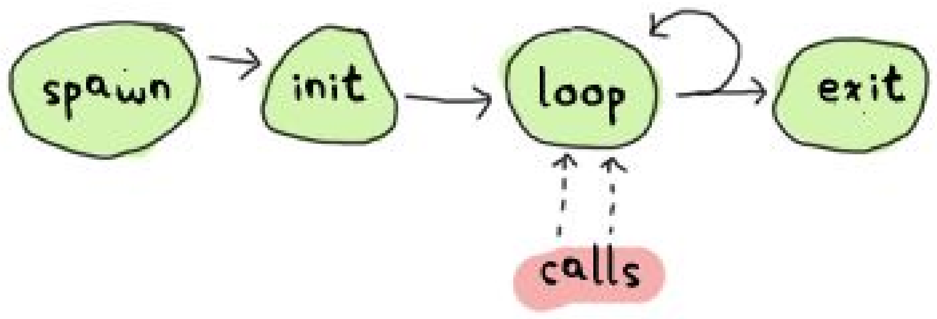
- OTP is a set of tools, libraries and design patterns to develop distributed applications.
- Provide generic behaviours which abstract away generic parts.
- Logging, Hot code loading, Packaging
- Less code to write, common coding style
- OTP is battle tested

OTP: Behaviours



OTP: Behaviours

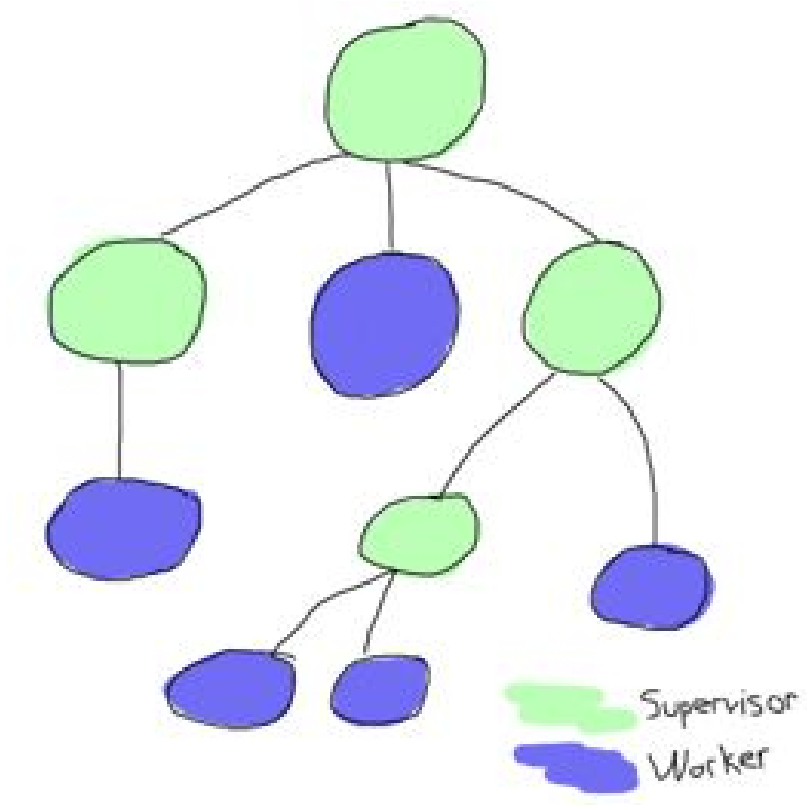
Generic Server (gen_server)



OTP: Behaviours

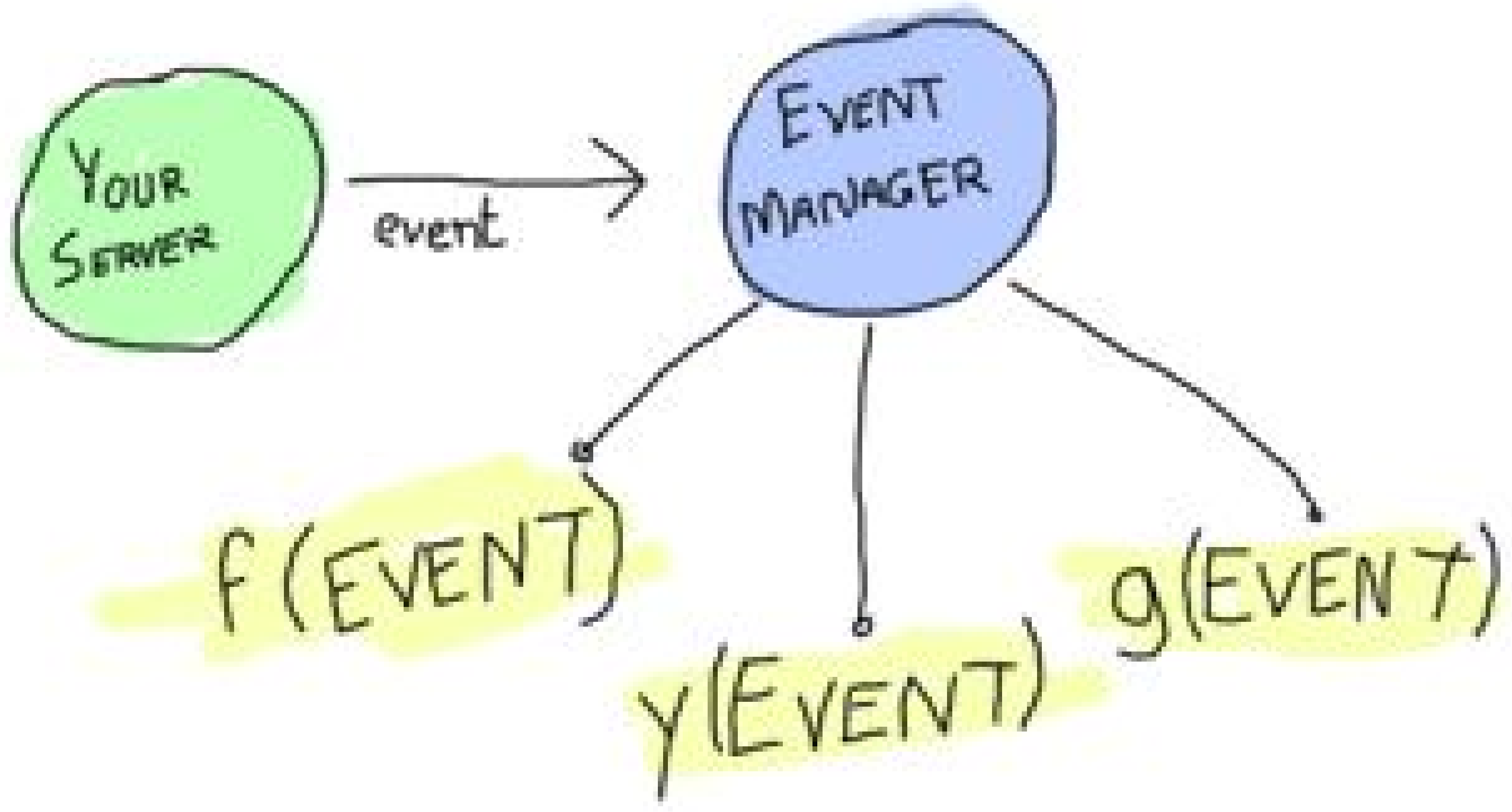
Supervisor (gen_supervisor):

Provides fault tolerance



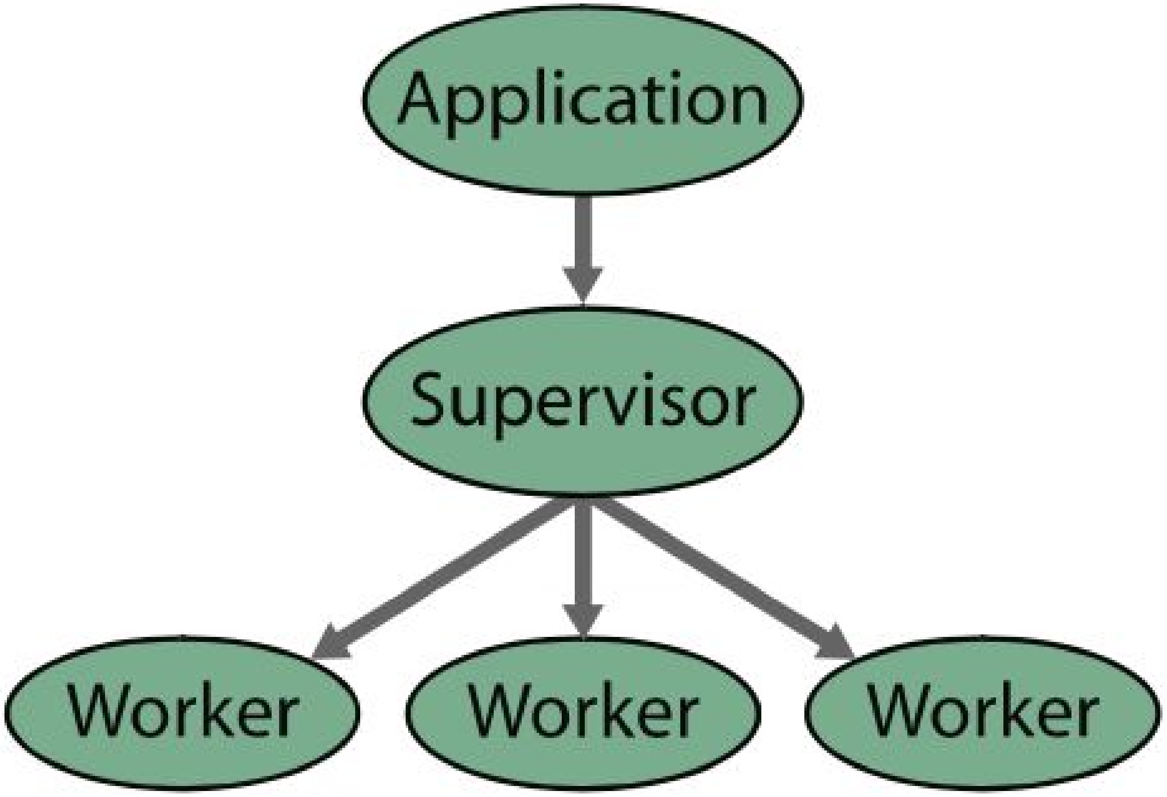
OTP: Behaviours

- Event Handler (gen_event)



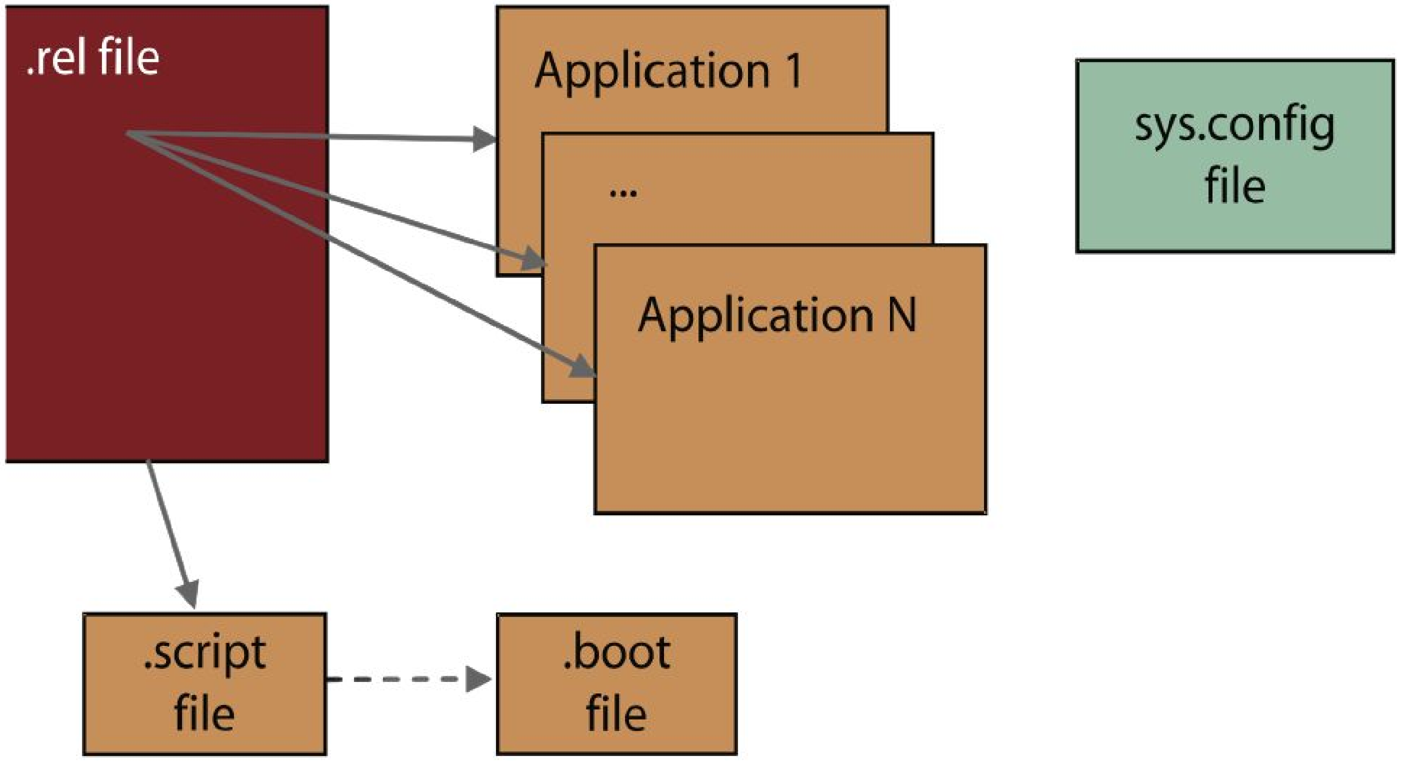
OTP: Behaviours

- Application



OTP: Packaging

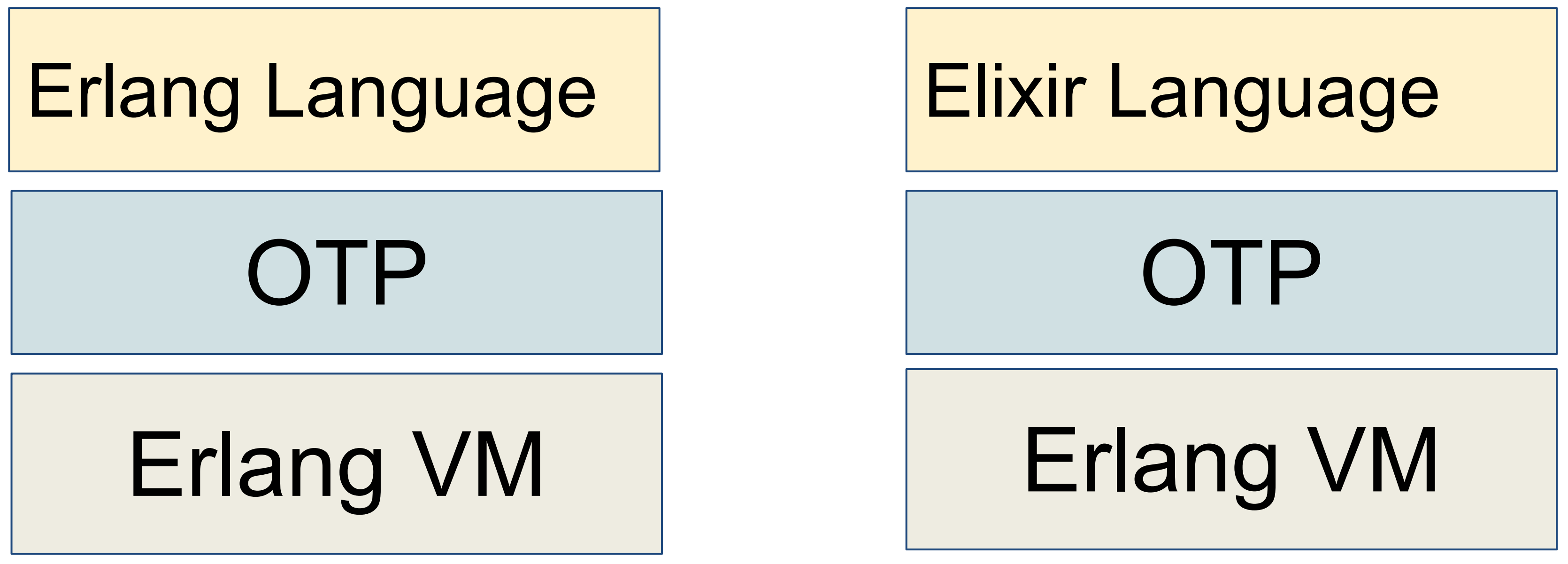
Group multiple apps into a release.



Where Erlang is used

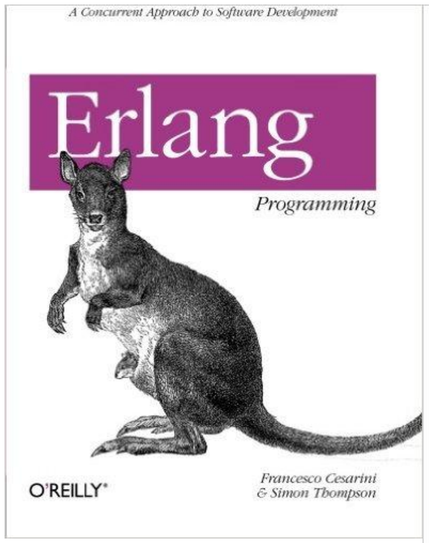
- Whatsapp
- Klarna (mobile payments)
- Adroll (advertisement)
- Wooga (gaming)
- RabbitMQ
- EjabberD
- CouchDB and Riak

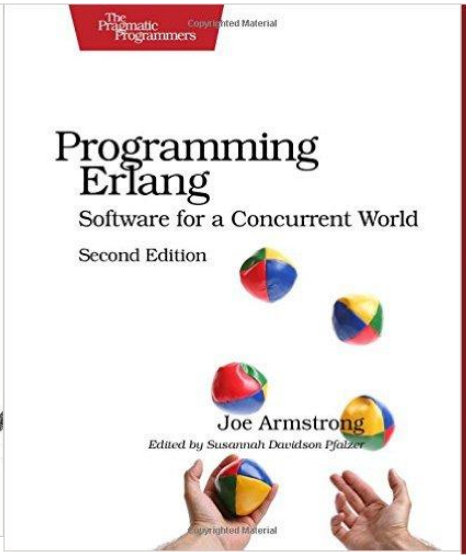
Where Elixir fits in?

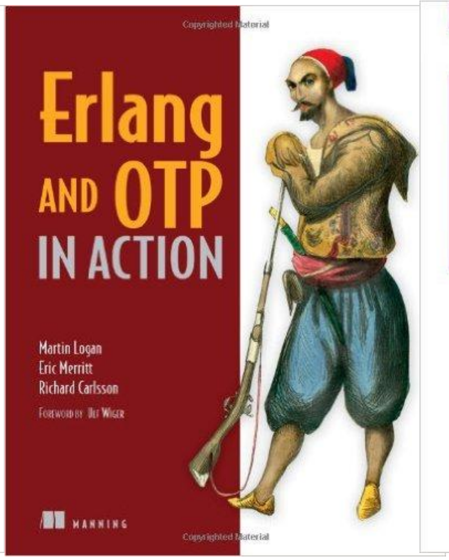


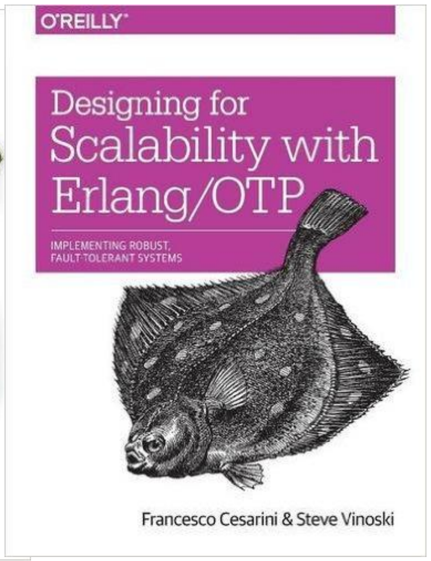
Resources for learning

- http://learnyousomeerlang.com/
- Books




The Pragmatic Programmers
Programming Erlang
Software for a Concurrent World
Second Edition
Joe Armstrong
Edited by Susannah Davidson Pfalzer




O'REILLY
Designing for Scalability with Erlang/OTP
IMPLEMENTING ROBUST, FAULT-TOLERANT SYSTEMS
Francesco Cesarini & Steve Vinoski

Questions?