FRANCESCO CESARINI

presents

OTP

the Middleware for Concurrent Distributed Scalable Architectures

Francesco Cesarini
Erlang Solutions

@FrancescoC
francesco@erlang-solutions.com
WHAT IS SCALABILITY?
WHAT IS (MASSIVE) CONCURRENCY?
WHAT IS HIGH AVAILABILITY?
What Is Fault Tolerance?
What Is Distribution Transparency?
Do you need a **distributed** system? Do you need a **scalable** system? Do you need a **reliable** system? Do you need a **fault-tolerant** system? Do you need a **massively concurrent** system? Do you need a **distributed** system? Do you need a **scalable** system? Do you need a **reliable** system? Do you need a **fault-tolerant** system? Do you need a **massively concurrent** system?

**YES, PLEASE!!!**
• **Open Source**
• **Concurrency-oriented**
• **Lightweight Processes**
• **Asynchronous Message Passing**
• **Share-nothing Model**
• **Process Linking / Monitoring**
• **Supervision Trees and Recovery Strategies**
• **Transparent Distribution Model**
• **Soft-real Time**
• **Let-it-fail Philosophy**
• **Hot-code Upgrades**
WELL, IN FACT YOU NEED MORE.
ERLANG IS JUST A PROGRAMMING LANGUAGE.
YOU NEED ARCHITECTURE PATTERNS.
YOU NEED MIDDLEWARE.
YOU NEED LIBRARIES.
YOU NEED TOOLS.
You need OTP.
LIBRARIES

STORAGE

O&M

INTERFACES

COMMUNICATION
OTP TOOLS

Development
Test Frameworks
Release & Deployment
Debugging & Monitoring
Less Code
Less Bugs
More Solid Code
More Tested Code
More Free Time

Servers
Finite State Machines
Event Handlers
Supervisors
Applications
OPEN SOURCE

OTP IS

PART OF THE ERLANG DISTRIBUTION
Behaviours
BEHAVIOURS

TIMEOUTS
DEADLOCKS
TRACING
MONITORING
DISTRIBUTION
Server process

- **Generic Behaviour Module**
- **Specific Callback Module**
call(Name, Message) ->
    Name ! {request, self(), Message},
    receive
        {reply, Reply} -> Reply
    end.

reply(Pid, Reply) ->
    Pid ! {reply, Reply}.

Client  {request, Pid, Message}  Server
       ^   |                        
       |   |                        
       v   v                        
        {reply, Reply} .
call(Name, Msg) ->
    Ref = make_ref(),
    Name ! {request, {Ref, self()}, Msg},
    receive {reply, Ref, Reply} -> Reply end.

reply({Ref, Pid}, Reply) ->
    Pid ! {reply, Ref, Reply}.
call(Name, Msg) ->
    Ref = erlang:monitor(process, Name),
    Name ! {request, {Ref, self()}, Msg},
    receive
        {reply, Ref, Reply} ->
            erlang:demonitor(Ref),
            Reply;
        {'DOWN', Ref, process, _Name, _Reason} ->
            {error, no_proc}
    end.
call(Name, Msg) ->
    Ref = erlang:monitor(process, Name),
    Name ! {request, {Ref, self()}, Msg},
    receive
        {reply, Ref, Reply} ->
            erlang:demonitor(Ref, [flush]),
            Reply;
        {'DOWN', Ref, process, _Name, _Reason} ->
            {error, no_proc}
    end.
Less Code
Less Bugs
More Solid Code
More Tested Code
More Free Time

Servers
Finite State Machines
Event Handlers
Supervisors
Applications
Isolate The Error!
convert(Day) ->
  case Day of
    monday      -> 1;
    tuesday     -> 2;
    wednesday   -> 3;
    thursday    -> 4;
    friday      -> 5;
    saturday    -> 6;
    sunday      -> 7;
    Other       ->
      {error, unknown_day}
  end.
convert(Day) ->
    case Day of
    monday    -> 1;
tuesday   -> 2;
wednesday -> 3;
thursday  -> 4;
friday    -> 5;
saturday  -> 6;
sunday    -> 7
end.
Propagating Exit Signals

PidA

{'EXIT', PidA, Reason}

PidB

{'EXIT', PidB, Reason}

PidC
TRAPPING AN EXIT SIGNAL

{'EXIT', PidA, Reason}
Release

- Mongoose IM
- folsom
- lager
- snmp
- mnesia
- stdlib
- SASL
- kernel
- ERTS
AUTOMATIC TAKEOVER AND FAILOVER
{myApp, 2000, {n1@host, {n2@host, n3@host}}}
{myApp, 2000, [n1@host, [n2@host, n3@host]]}
\{myApp, 2000, \{n1@host, \{n2@host, n3@host\}\}\}
{myApp, 2000, {n1@host, {n2@host, n3@host}]]

N1 takes over N3
Conclusions
Do you need a **distributed** system? Do you need a **scalable** system? Do you need a **reliable** system? Do you need a **fault-tolerant** system? Do you need a **massively concurrent** system? Do you need a **distributed** system? Do you need a **scalable** system? Do you need a **reliable** system? Do you need a **fault-tolerant** system? Do you need a **massively concurrent** system?

USE ERLANG
USE ERLANG/OTP

Do you need a **distributed** system? Do you need a **scalable** system? Do you need a **reliable** system? Do you need a **fault-tolerant** system? Do you need a **massively concurrent** system? Do you need a **distributed** system? Do you need a **scalable** system? Do you need a **reliable** system? Do you need a **fault-tolerant** system? Do you need a **massively concurrent** system?
Questions?

Francesco Cesarini
Erlang Solutions

@FrancescoC
francesco@erlang-solutions.com
www.erlang-solutions.com