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!!!**
TO THE RESCUE
• 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.
WHAT IS MIDDLEWARE?
Middleware

Design Patterns

Fault Tolerance

Distribution

Upgrades

Packaging
What are Libraries?
WHAT TOOLS?
OPEN SOURCE

OTP IS

PART OF THE ERLANG DISTRIBUTION
Less Code
Less Bugs
More Solid Code
More Tested Code
More Free Time

Servers
Finite State Machines
Event Handlers
Supervisors
Applications
Behaviours
Less Code
Less Bugs
More Solid Code
More Tested Code
More Free Time

Servers
Finite State Machines
Event Handlers
Supervisors
Applications

OTP
call(Name, Message) ->
    Name ! {request, self(), Message},
    receive
        {reply, Reply} -> Reply
    end.

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

Timeouts
Deadlocks
Tracing
Monitoring
Distribution
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.
ISOLATE THE ERROR!
PROPAGATING EXIT SIGNALS

{'EXIT', PidA, Reason}

{'EXIT', PidB, Reason}

PidA

PidB

PidC
TRAPPING AN EXIT SIGNAL

{'EXIT', PidA, Reason}
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}}}
“To scale the radical concurrency-oriented programming paradigm to build reliable general-purpose software, such as server-based systems, on massively parallel machines (10^5 cores).”
The Runtime Queues

Erlang VM

Scheduler #1
run queue

Scheduler #2
run queue

... run queue...

Scheduler #N
run queue

migration
logic
LIMITATIONS ARE PRESENT AT THREE LEVELS

WP4 Scalable Infrastructure
WP3 SD Erlang Language
WP2 Virtual Machine

WP5 Tools

WP6 Case Studies
- Push the **responsibility for scalability** from the programmer to the VM
- Analyze **performance** and scalability
- Identify **bottlenecks** and prioritize changes and extensions
- Tackle **well-known scalability issues**
  - ETS tables (shared global data structure)
  - Message passing, copying and frequently communicating processes
• **Two major issues**
  - Fully connected clusters
  - Explicit process placement

• **Scalable Distributed (SD) Erlang**
  - Nodes grouping
  - Non-transitive connections
  - Implicit process placement
  - Part of the **standard** Erlang/OTP package

• **New concepts introduced**
  - Locality, Affinity and Distance
Wombat O&M

- Middleware layer
- Set of Erlang applications
- Create and manage clusters of (heterogeneous) Erlang nodes
- API to monitor and control Erlang distributed systems
- Existing tracing/logging/debugging tools pluggable
- Broker layer between users and cloud providers
- Auto-scaling

... And Much More
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 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
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/OTP**
Questions?

@francescoC