

FRANCESCO CESARINI

presents



Jvřshj xht Gj xfvon

Erlang Solutions

@FrancescoC

francesco@erlang-solutions.com

www.erlang-solutions.com

Erlang
SOLUTIONS



Erlang
SOLUTIONS

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**

YES, PLEASE!!!

system? Do you need a **reliable** system? Do you need a **fault-tolerant** system? Do **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**



TO THE **RESCUE**

WHAT IS ERLANG

- 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?

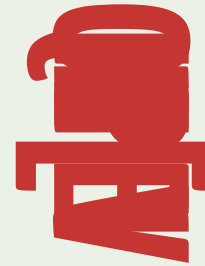
DESIGN PATTERNS

FAULT TOLERANCE

DISTRIBUTION

UPGRADES

PACKAGING



WHAT ARE LIBRARIES?

STORAGE
O&M
INTERFACES
COMMUNICATION



WHAT TOOLS?

DEVELOPMENT
TEST FRAMEWORKS
RELEASE & DEPLOYMENT
DEBUGGING & MONITORING



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



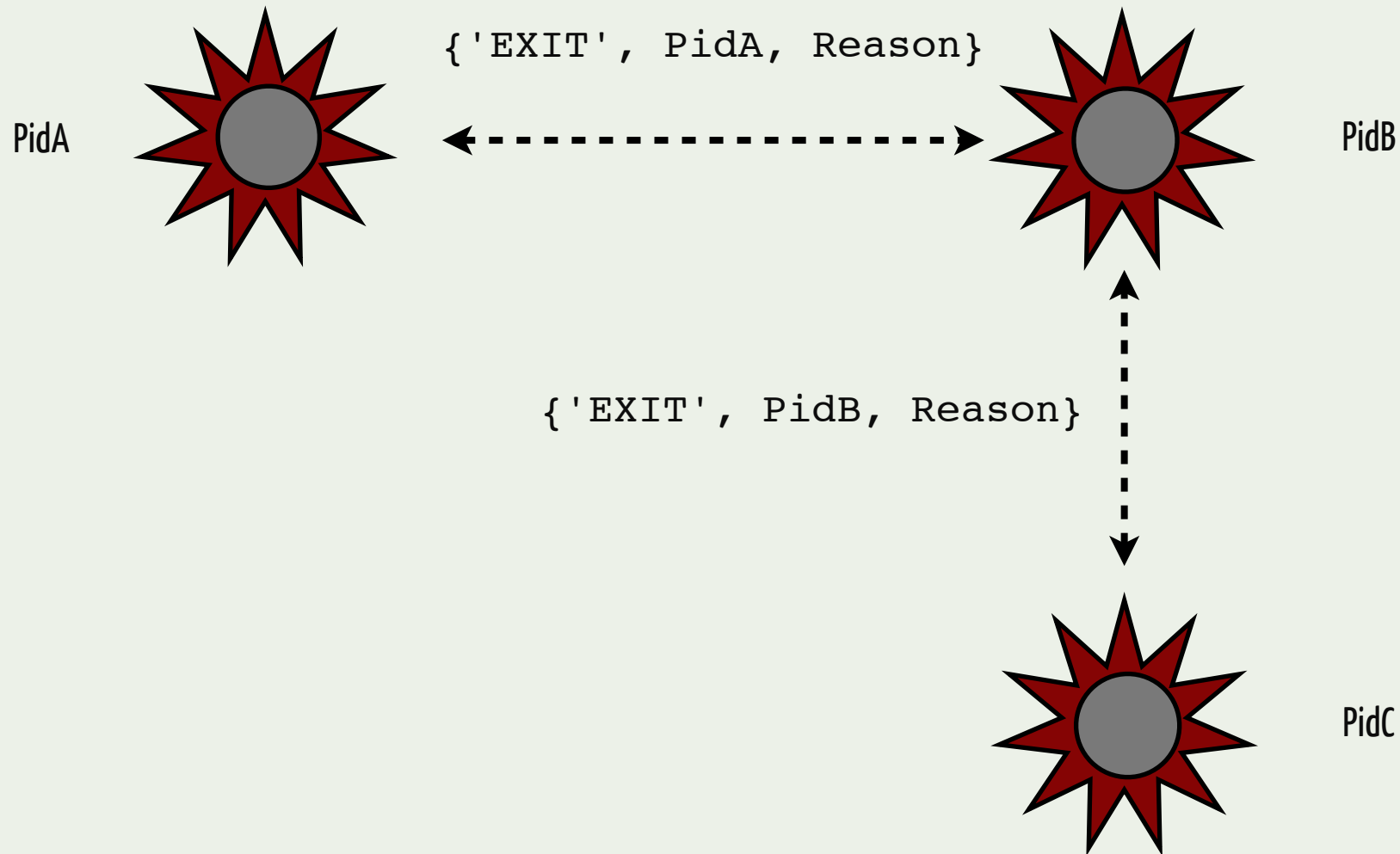
```
convert(Day) ->  
  case Day of  
    monday      -> 1;  
    tuesday     -> 2;  
    wednesday   -> 3;  
    thursday    -> 4;  
    friday      -> 5;  
    saturday    -> 6;  
    sunday      -> 7;  
    Other ->  
      {error, unknown_day}  
  end.
```

```
convert(Day) ->  
  case Day of  
    monday      -> 1;  
    tuesday     -> 2;  
    wednesday   -> 3;  
    thursday    -> 4;  
    friday      -> 5;  
    saturday    -> 6;  
    sunday      -> 7;  
  
end.
```

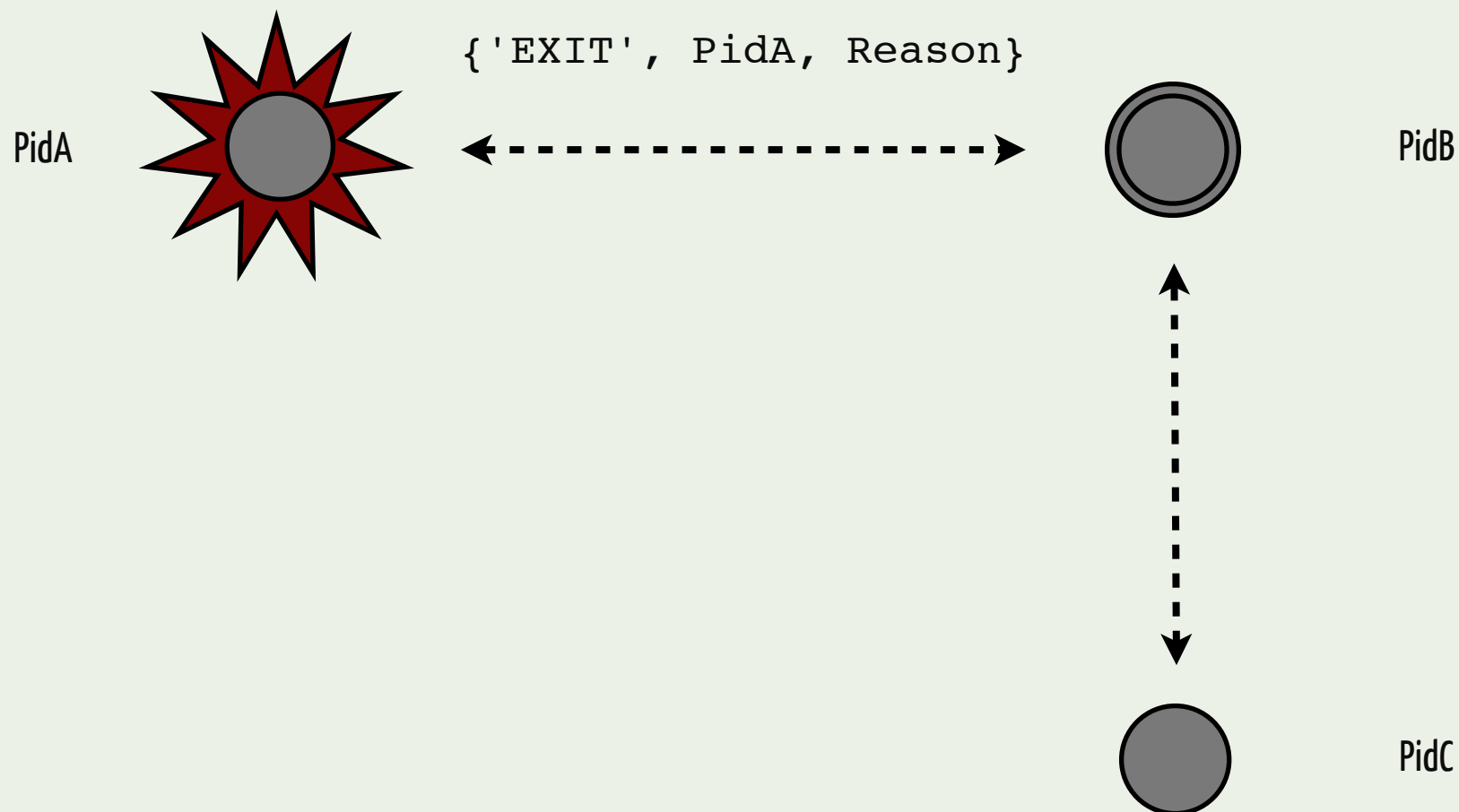
ISOLATE THE **ERROR!**

Exit Signals

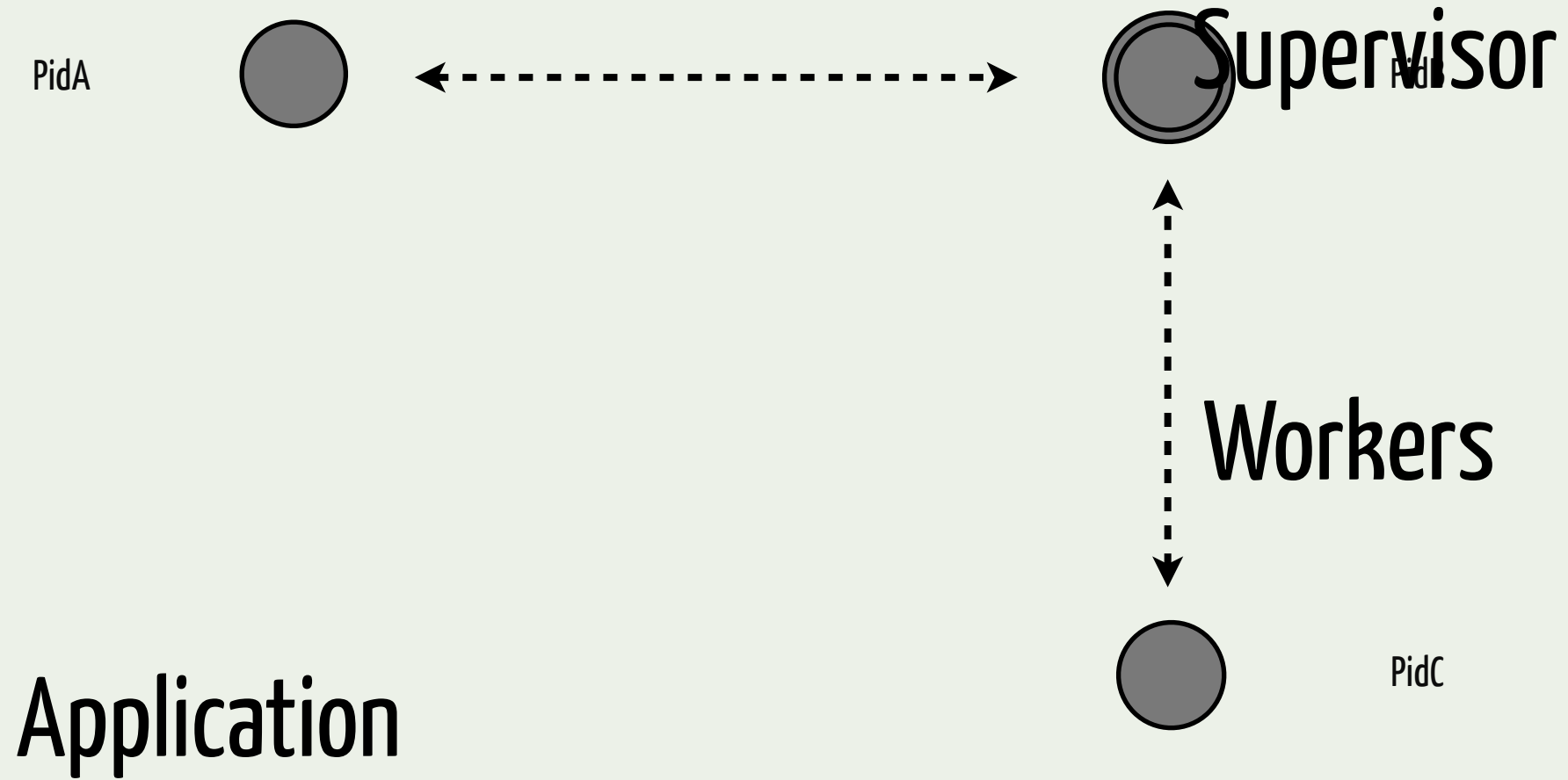
PROPAGATING EXIT SIGNALS



TRAPPING AN EXIT SIGNAL



Supervisors



Releases

Release

Mongoose
IM

folsom

lager

snmp

mnesia

stdlib

SASL

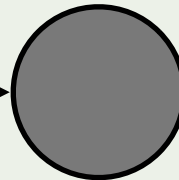
kernel

ERTS

BEHAVIOURS

GENERIC
BEHAVIOUR
MODULE

V_j/W_j



process

SPECIFIC
CALLBACK
MODULE

Less Code

Servers

Less Bugs

Finite State Machines

More Solid Code

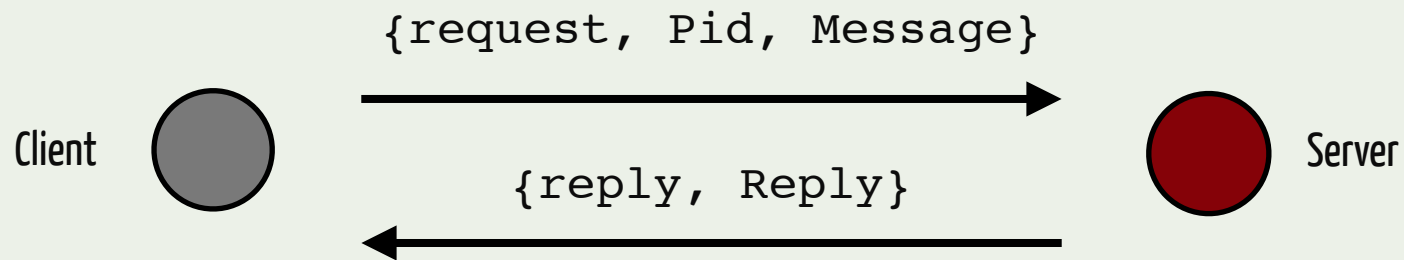
Event Handlers

More Tested Code

Supervisors

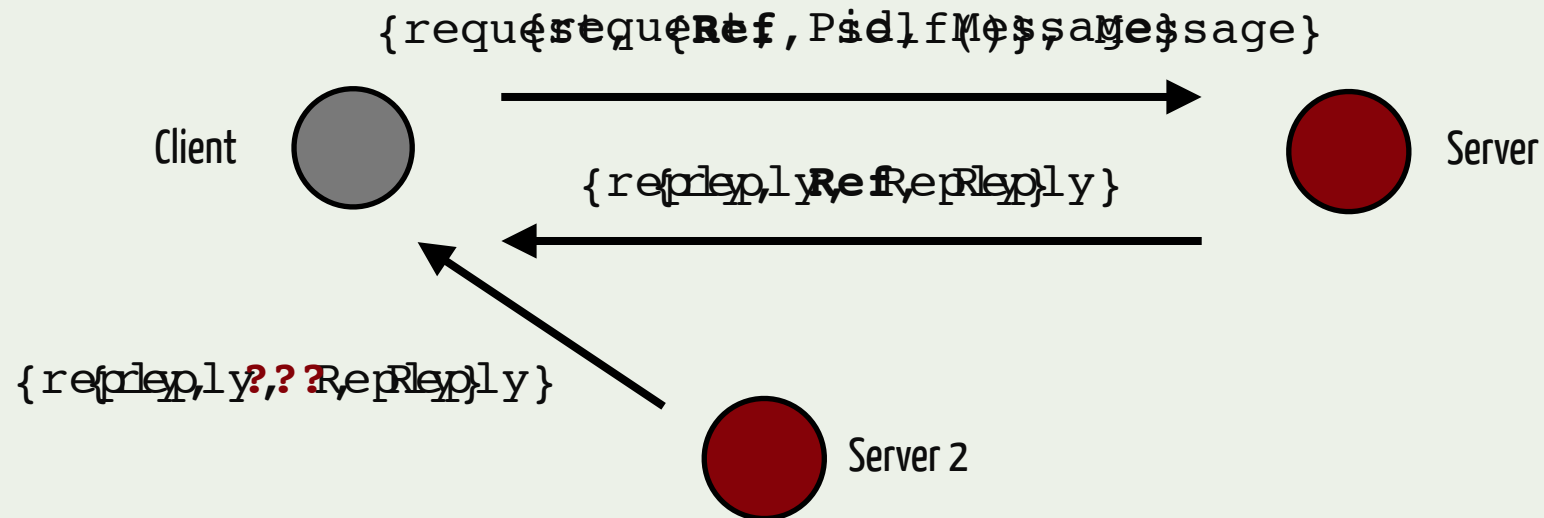
More Free Time

Applications



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

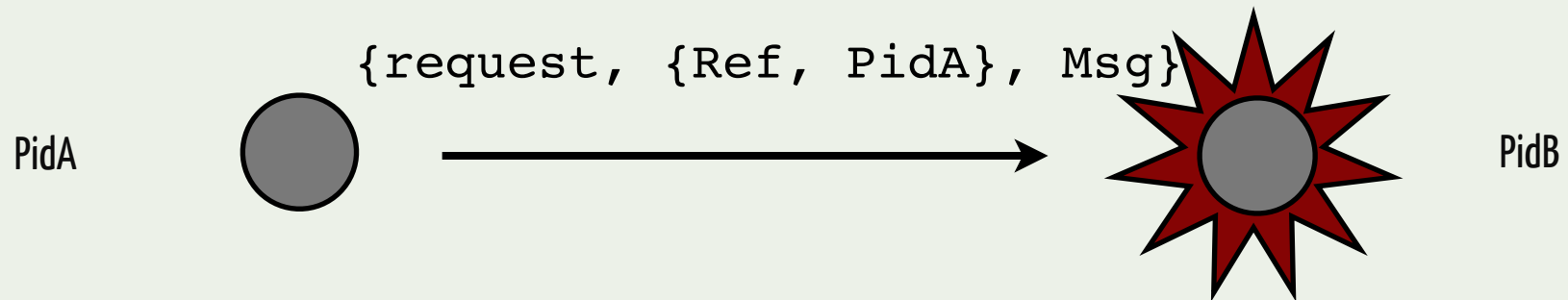
```
reply(Pid, Reply) ->
  Pid ! {reply, Reply}.
```



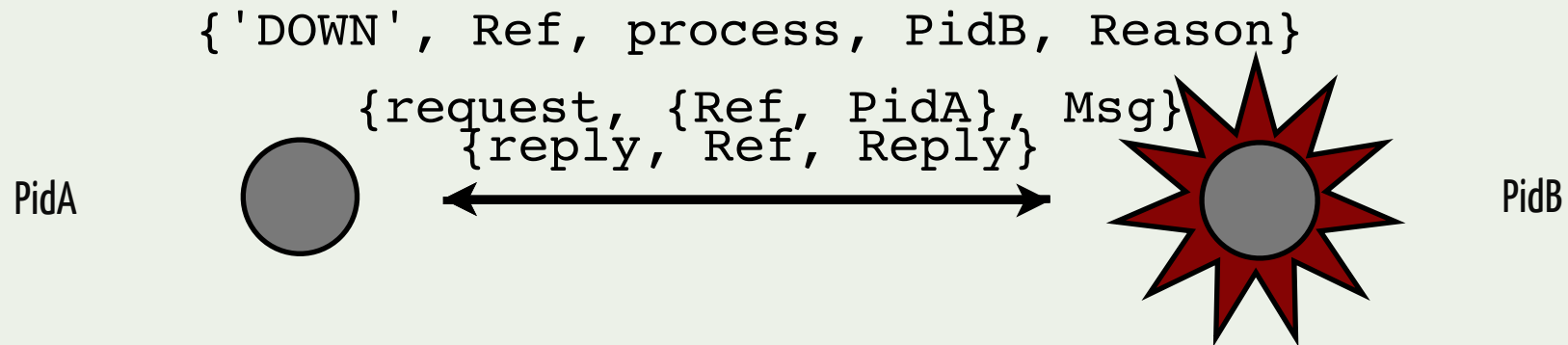
```

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.
```



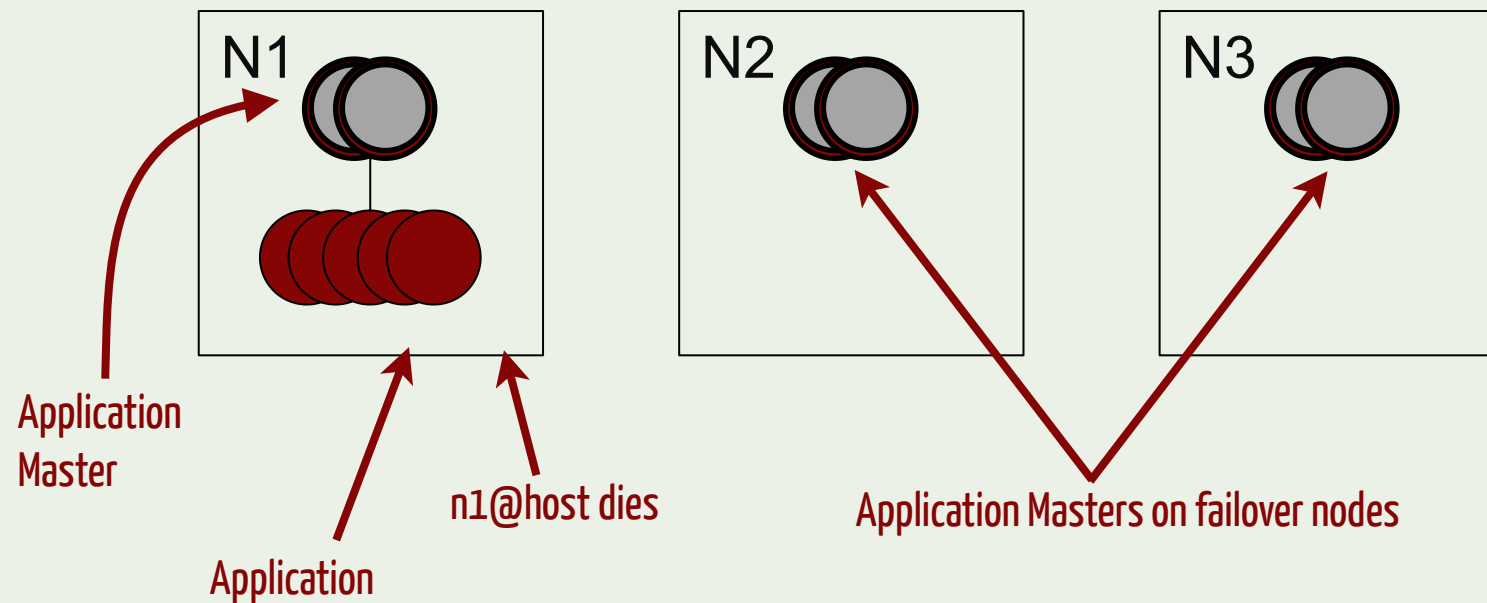
```
call(Name, Msg) ->
  Ref = erlang:monitor(process, Name),
  Name ! {request, {Ref, self()}, Msg},
  receive
    {reply, Ref, Reply} ->
      erlang:demonitor(Ref, [flush]),
      Reply;
    {'DOWN', Ref, process, _Name, _Reason} ->
      {error, no_proc}
  end.
```

TIMEOUTS
DEADLOCKS
TRACING
MONITORING
DISTRIBUTION

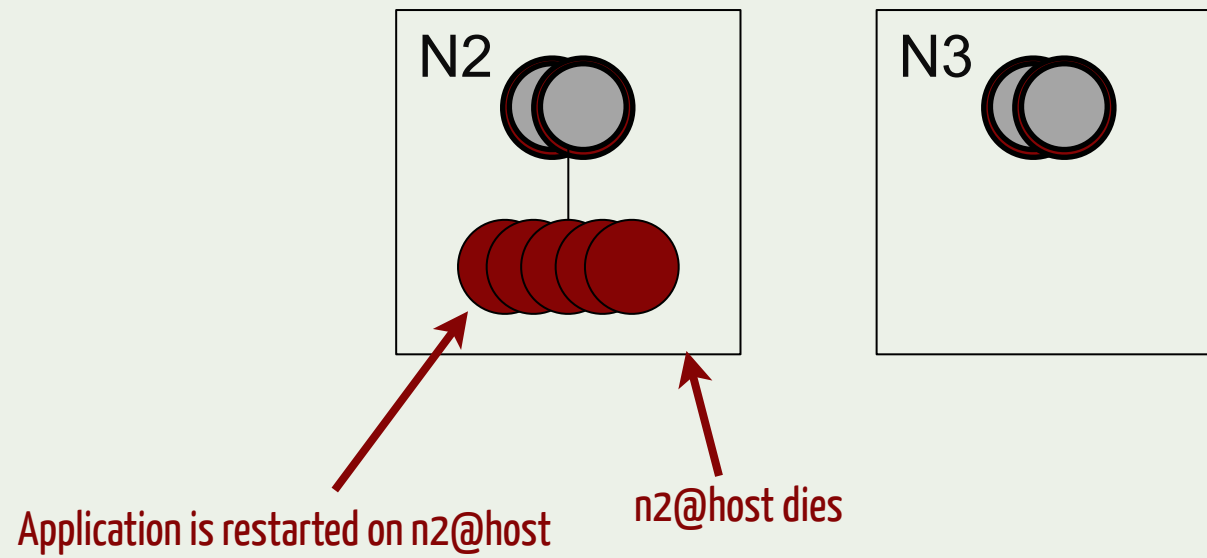


AUTOMATIC TAKEOVER AND FAILOVER

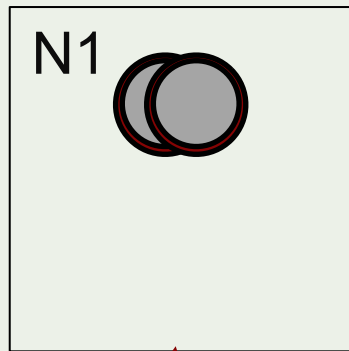
`{myApp, 2000, {n1@host, {n2@host, n3@host}}}`



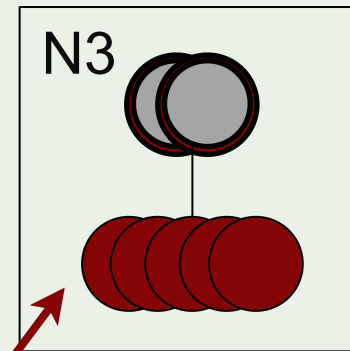
`{myApp, 2000, {n1@host, {n2@host, n3@host}}}`



`{myApp, 2000, {n1@host, {n2@host, n3@host}}}`

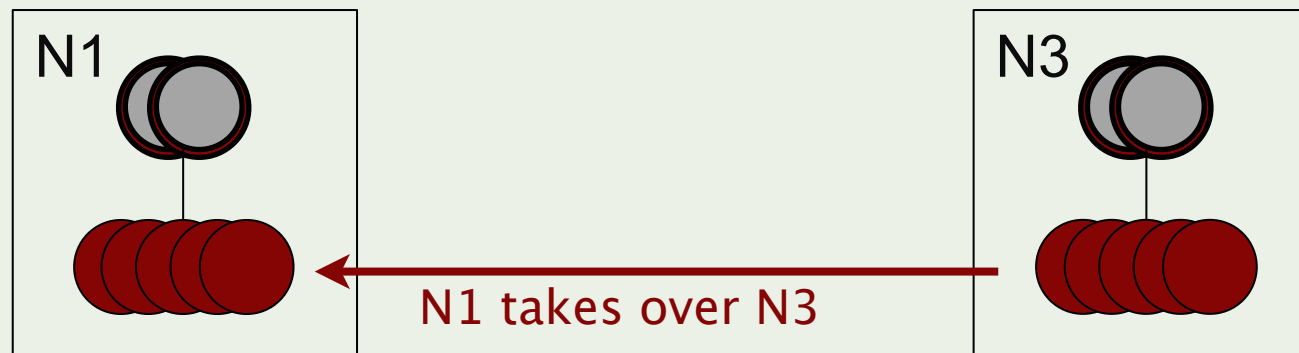



n1@host comes back up




Application is restarted on n3@host

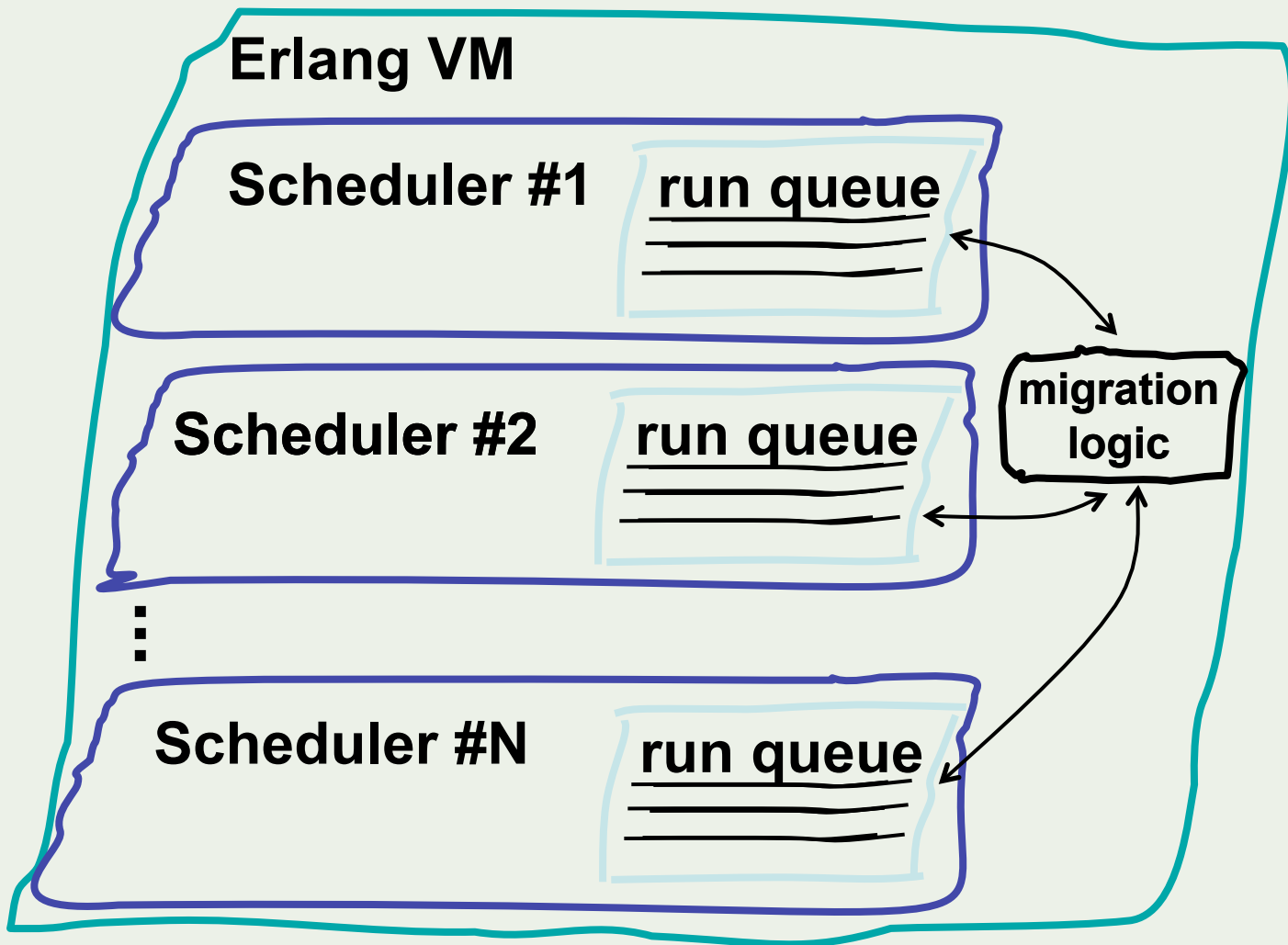
`{myApp, 2000, {n1@host, {n2@host, n3@host}}}`



RELEASE STATEMENT OF AIMS

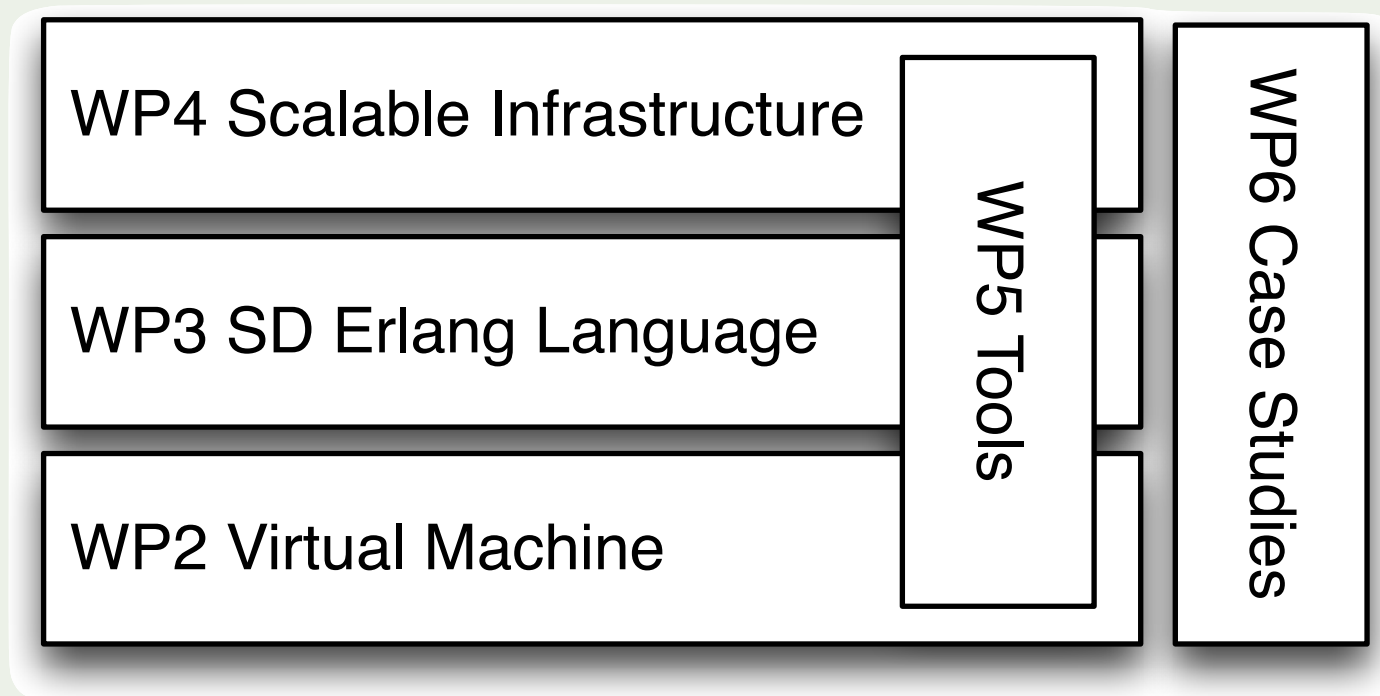
"To scale the radical **concurrency-oriented programming** paradigm to build **reliable** general-purpose software, such as server-based systems, on **massively parallel** machines (10^5 cores)."

The Runtime Queues





LIMITATIONS ARE PRESENT AT THREE LEVELS



VM

LANGUAGE

INFRASTRUCTURE

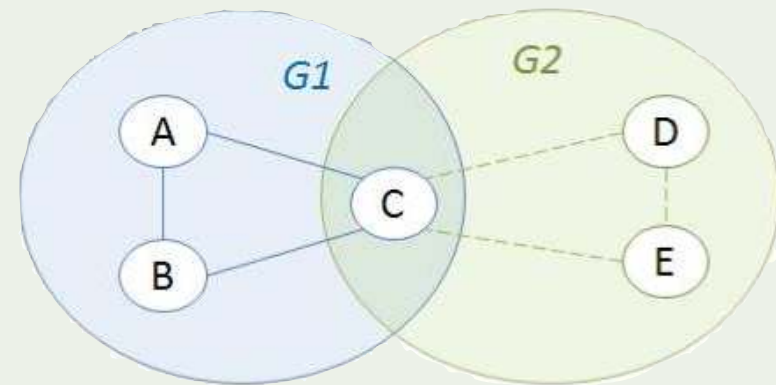
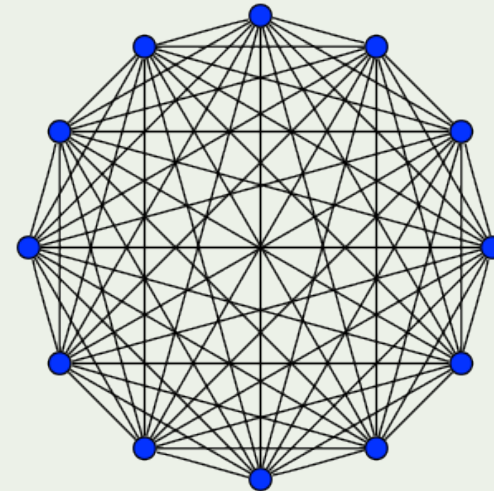
- PUSH THE **RESPONSIBILITY FOR SCALABILITY** FROM THE PROGRAMMER TO THE **VM**
- ANALYZE **PERFORMANCE** AND SCALABILITY
- IDENTIFY **BOTTLENECKS** AND PRIORITIZE CHANGES AND EXTENSIONS
- TACKLE **WELL-KNOWN SCALABILITY ISSUES**
 - **ETS** TABLES (SHARED GLOBAL DATA STRUCTURE)
 - MESSAGE PASSING, COPYING AND **FREQUENTLY COMMUNICATING PROCESSES**

VM

LANGUAGE

INFRASTRUCTURE

- 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



VM

LANGUAGE

INFRASTRUCTURE

CCL /sɪ'sɪlɪ/

- 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**

USE ERLANG

system? Do you need a **reliable** system? Do you need a **fault-tolerant** system? Do **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**

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**

USE ERLANG/OTP

system? Do you need a **reliable** system? Do you need a **fault-tolerant** system? Do **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**

QUESTIONS?

@francescoC