TORBEN HOFFMANN

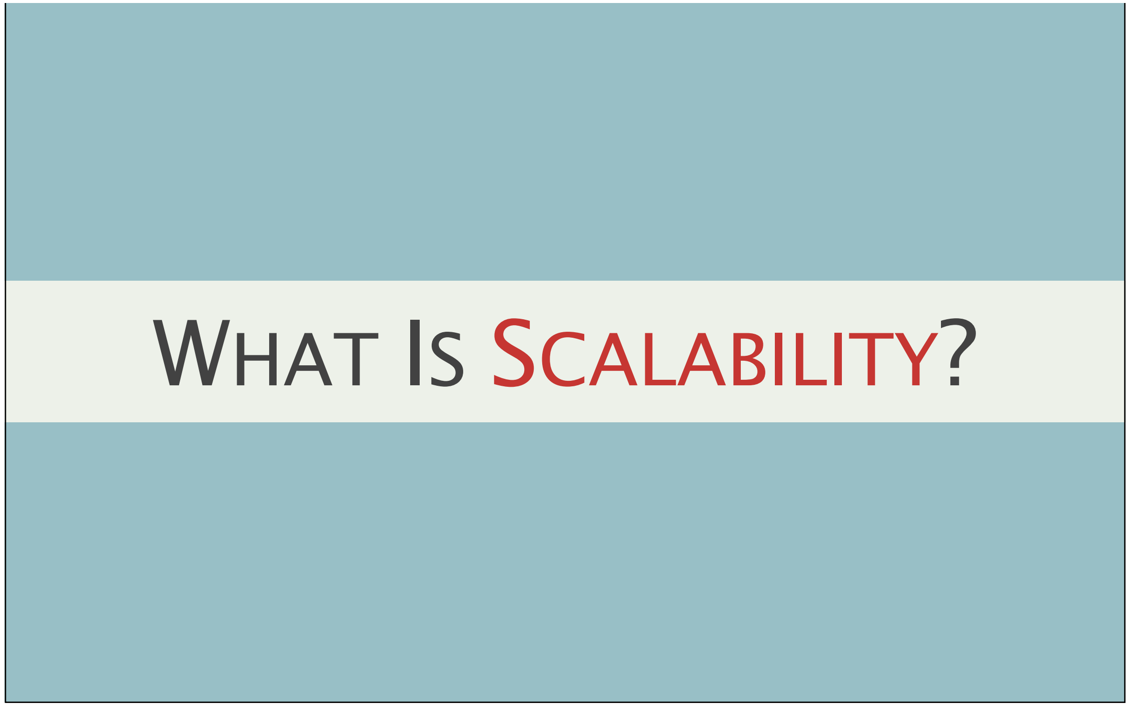
presents

ERLANG/OTP

Torben Hoffmann
Erlang Solutions

@LeHoff
torben@erlang-solutions.com
www.erlang-solutions.com

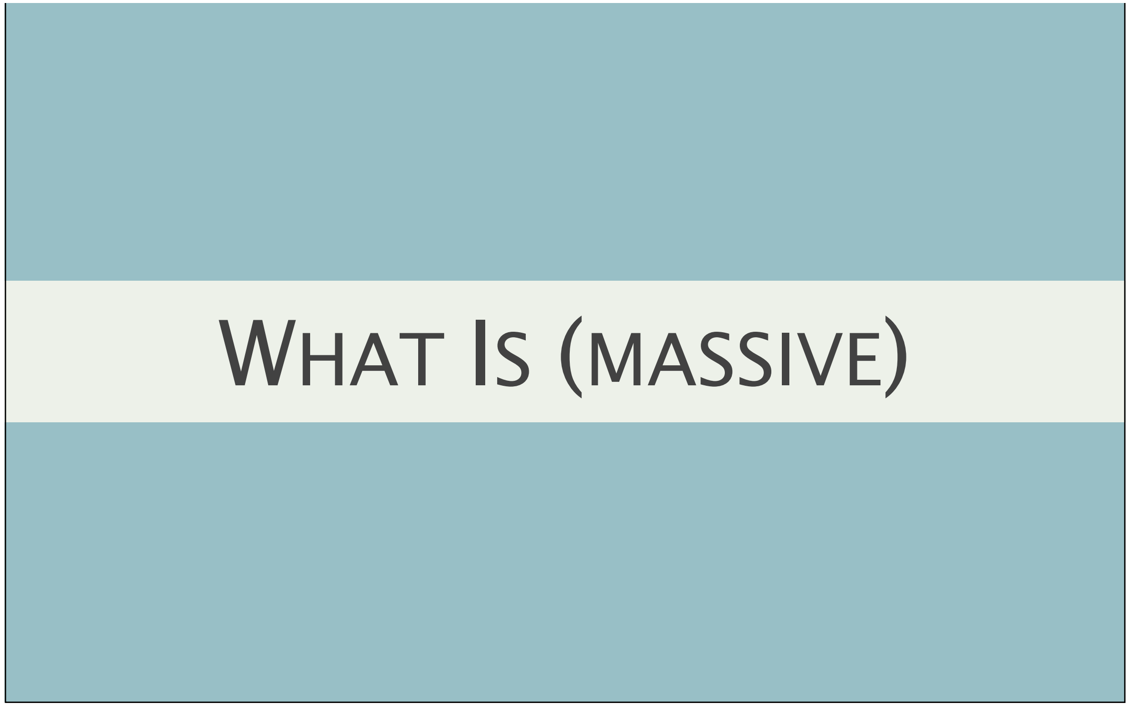
Erlang
SOLUTIONS



What Is Scalability?

handle traffic spike
Behave predictably under extended heavy load
Carry the traffic it was designed to handle

handle traffic spike
Behave predictably under extended heavy load
Carry the traffic it was designed to handle



What Is (massive)

millions of simultaneous requests being handled
Requests running independently of each other
SMS TV voting spile

millions of simultaneous requests being handled
Requests running independently of each other
SMS TV voting spile

What Is High

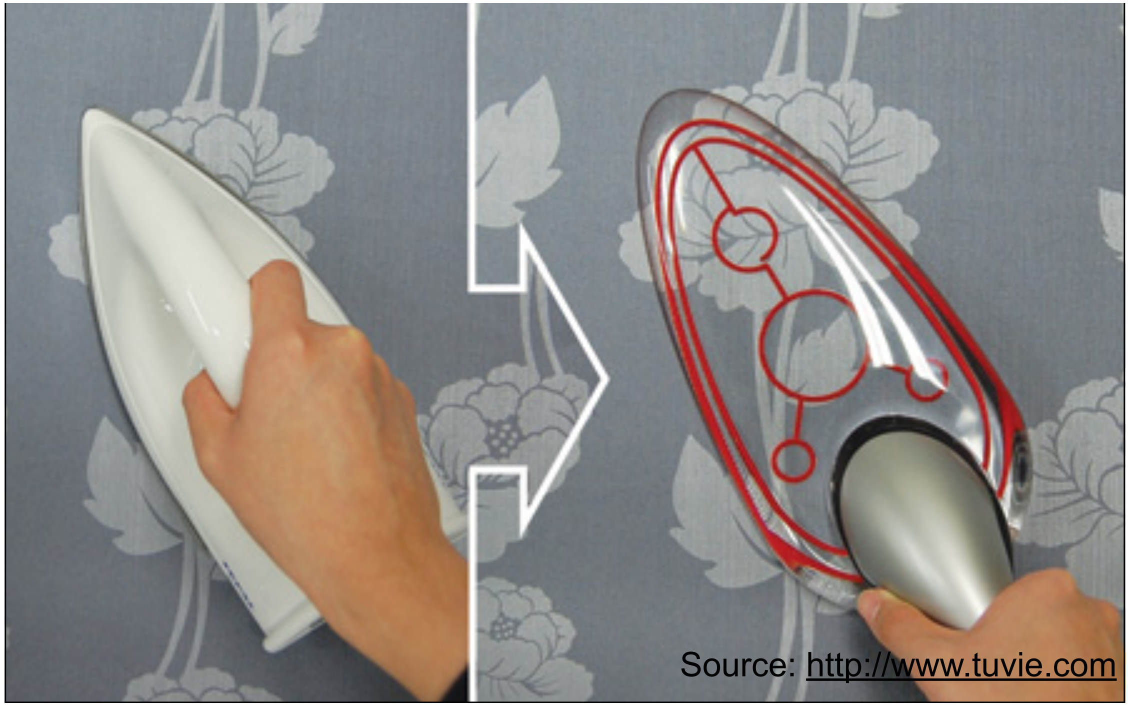
No single point of failure.
Two Computers (Joe Armstrong) Three if you ask Leslie Lamport
Redundant network - Sys admin tripping on a network cable not an excuse
Battery backup / generators. Hardware failure
Distribute your software and data. Software is important, but it is not only about Software.

What Is Fault

Even if things go wrong continue working and not affect other things in the system.
Ability to isolate the error. Regain control.

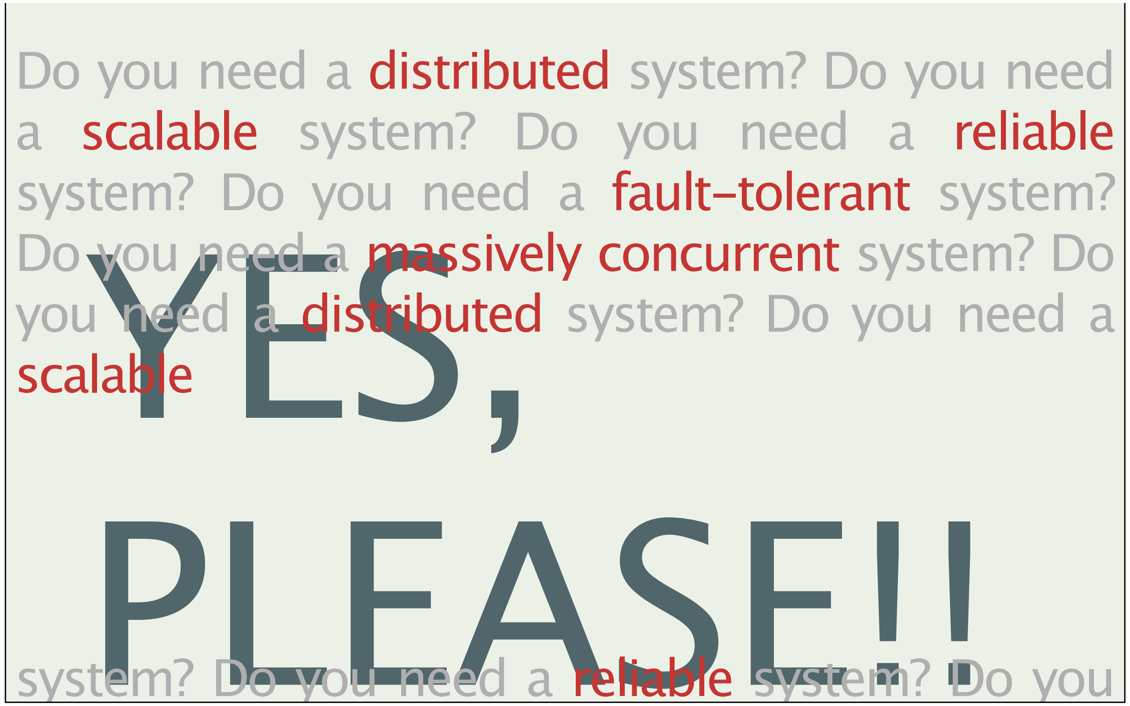
What Is Distribution

Simplicity in designing your system. Scalability and fault tolerance. Language with built in distribution.


Source: http://www.tuvie.com

With Erlang you can hide how the distribution over machines is taking place or you can decide to peek inside if you want to know more. Flexibility

Simplicity in designing your system. Scalability and fault tolerance. Language with built in distribution.



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

YES, PLEASE!!

So, you have all these requirements. What is it you actually need?

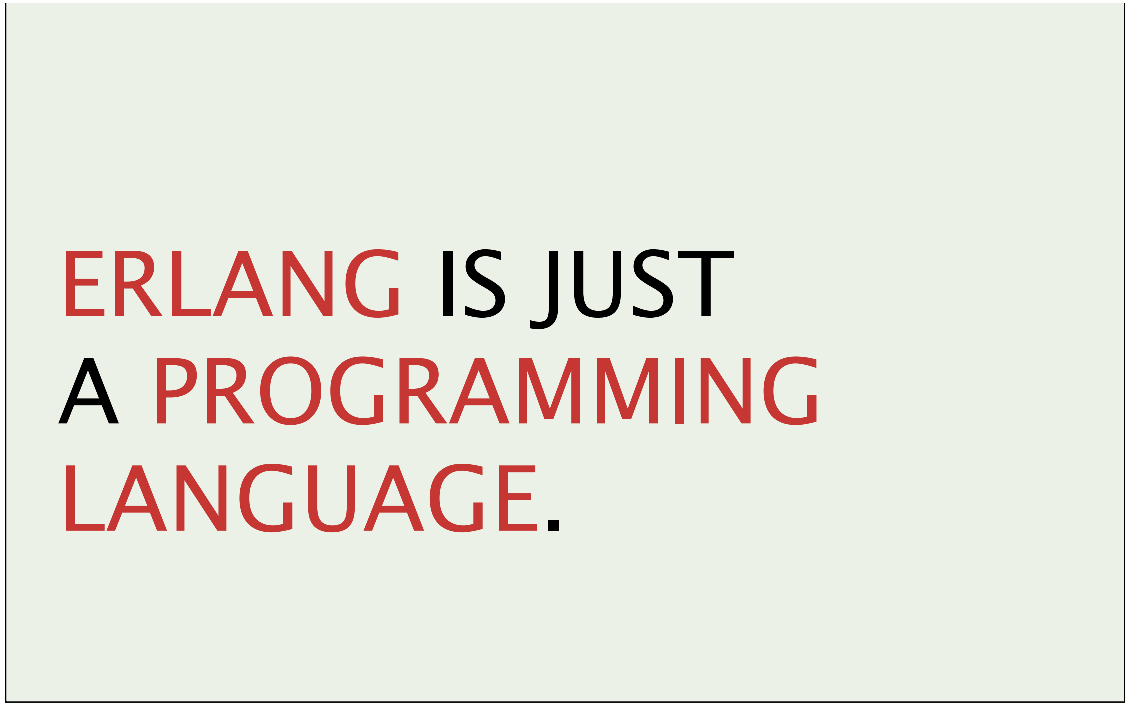
ERLANG

TO THE RESCUE

What is Erlang

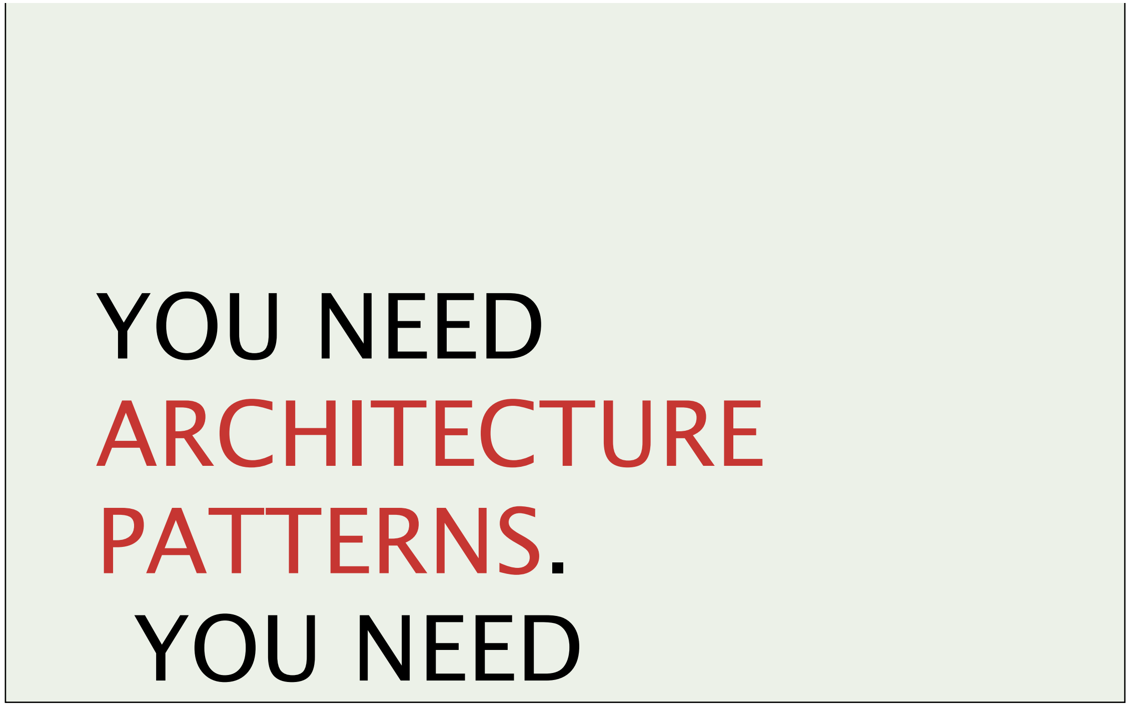
- Open source
- Concurrency-oriented
- Lightweight processes
- Asynchronous message passing
- Share-nothing model
- Process linking / monitoring
- Supervision trees and recovery strategies
- Transparent distribution model
- Soft-real time
- Let-it-fail philosophy
- Hot-code upgrades

WELL, IN FACT YOU NEED MORE.



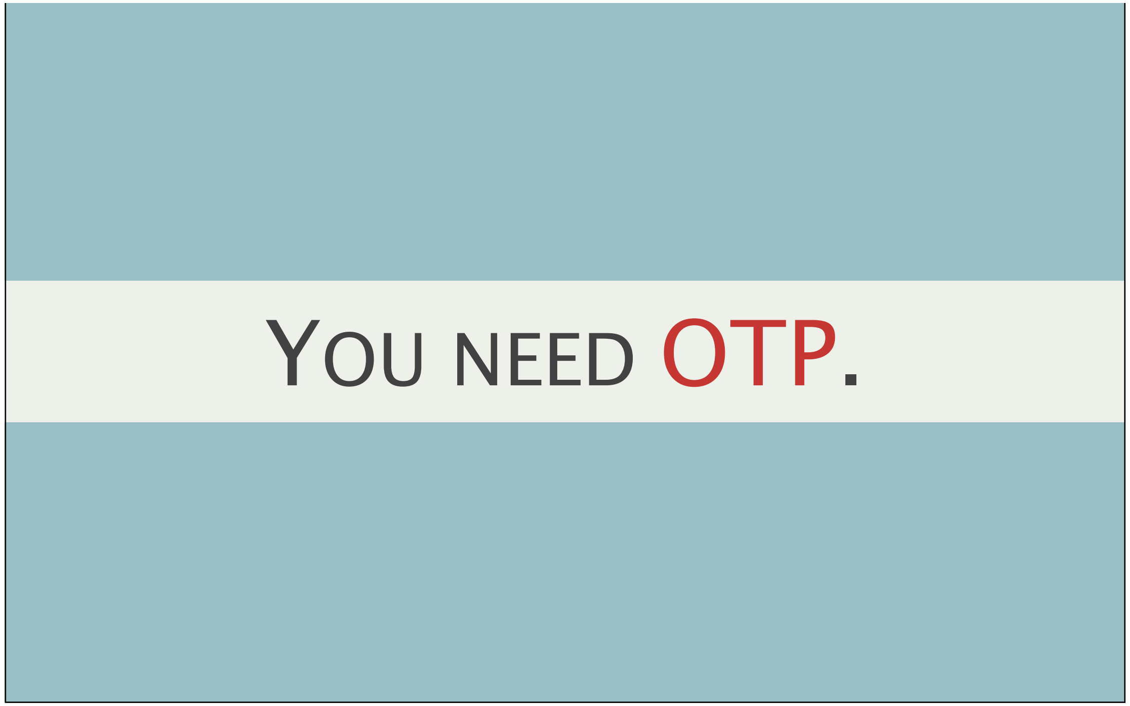
ERLANG IS JUST
A PROGRAMMING
LANGUAGE.

If you need to develop a highly complex system which never goes down, has built in fault tolerance, distribution mechanisms and manages millions of simultaneous transactions, you need more than just a programming language.



YOU NEED
ARCHITECTURE
PATTERNS.
YOU NEED

Erlang solves many software related problems.
It is still just a programming language
Lots of problems you solve are the same.
Don’t want to reinvent the wheel.
Development, deployment and monitoring tools.



You need OTP.

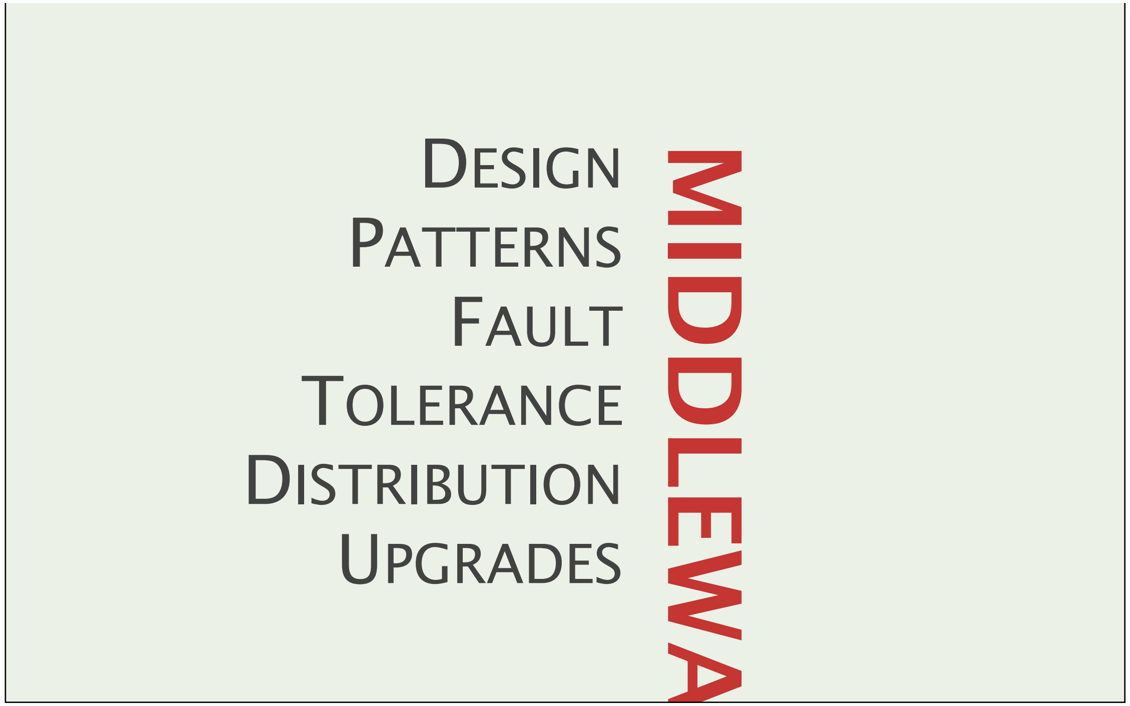
BOS – 1993, merged with Erlang in 1995.
Erlang is only 33% of your strength. VM, OTP
What does OTP Stand for? Rather not tell you.
On The Phone, One True Pair, **Oh, This is Perfect**

Ministry of Propaganda at Ericsson
Openness - JSON, XML, ASN.1, SNMP, Java, C, Ports.
Telecom - Distributed, Massively concurrent soft realtime systems with requirements
on scalability
Platform -

What is Middleware?

A set of abstract principles and design rules
They describe the software architecture of an Erlang System
Needed so existing tools will be compatible with them
Facilitate the understanding of the system among teams

Leave Architectural Patterns to Last



DESIGN
PATTERNS
FAULT
TOLERANCE
DISTRIBUTION
UPGRADES

MIDDLEWA

Systems will do very different things. But the issues are still the same. Glue to manage your distribution and communication layers. Your fault tolerance layers. Deploy and upgrade your systems.

What are Libraries?

Basic Applications Erlang Runtime System, Kernel, Compiler, Standard Lib,

System Architecture Support Library (SASL)

Database Applications

Mnesia (Distributed relational database) ODBC (Interface for accessing SQL databases)

STORAGE
O&M
INTERFACES
COMMUNICATIO
N

LIBRARIES

Operations and Maintenance Applications

Operating System Monitor, SNMP, OTP MIBs

Interface and communication Applications

- Corba ORB, ASN1 Compiler, Crypto, (Wx widgets), Inets (TCP, UDP, HTTP, FTP),
  Java Interface & Erlang to C Interface, SSH/SSL, XML Parsing

WHAT TOOLS?

OTP TOOLS

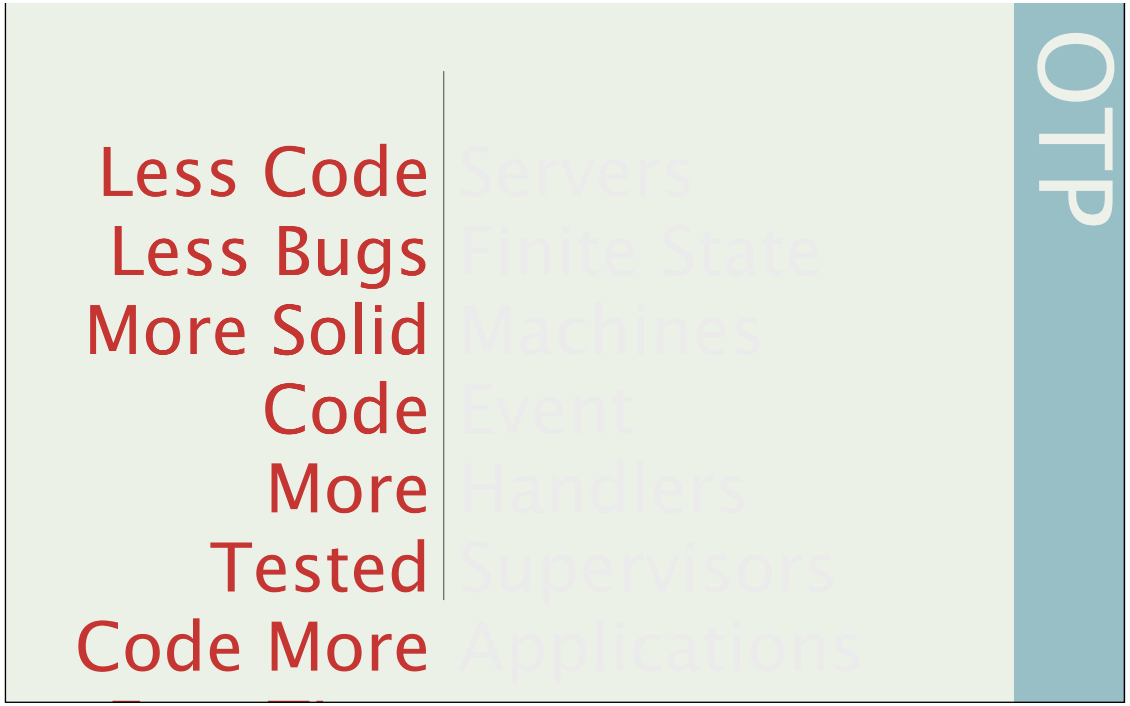
DEVELOPMENT
TEST
FRAMEWORKS
RELEASE &
DEPLOYMENT
DEBUGGING &

Eunit, Common test. No mocking frameworks, several OS.
Release and upgrade tools. Worth the hassle?
Low level debugging tools. dbg, trace local & global calls
Percept - Concurrency bottlenecks/profiling
Observer - web front end to other tools, e.g. crash dump viewer.
etop, crash dump viewer

OPEN SOURCE

OTP IS

PART OF THE ERLANG DISTRIBUTION



OTP

Less Code
Less Bugs
More Solid
Code
More
Tested
Code More

Servers
Finite State
Machines
Event
Handlers
Supervisors
Applications

Cons: Steeper learning curve, affects performance

Fail Safe, Fail Early

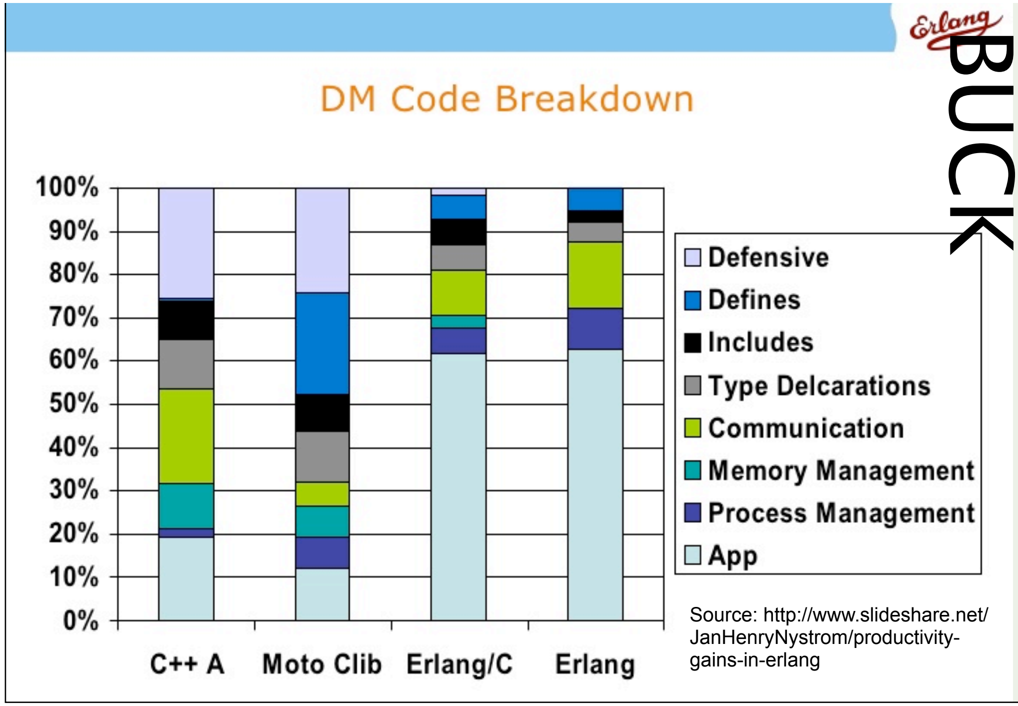
* Hide tricky parts of Concurrency. Mutexes, deadlocks, race conditions

* Stress 9-5 programmers

Let It Fail

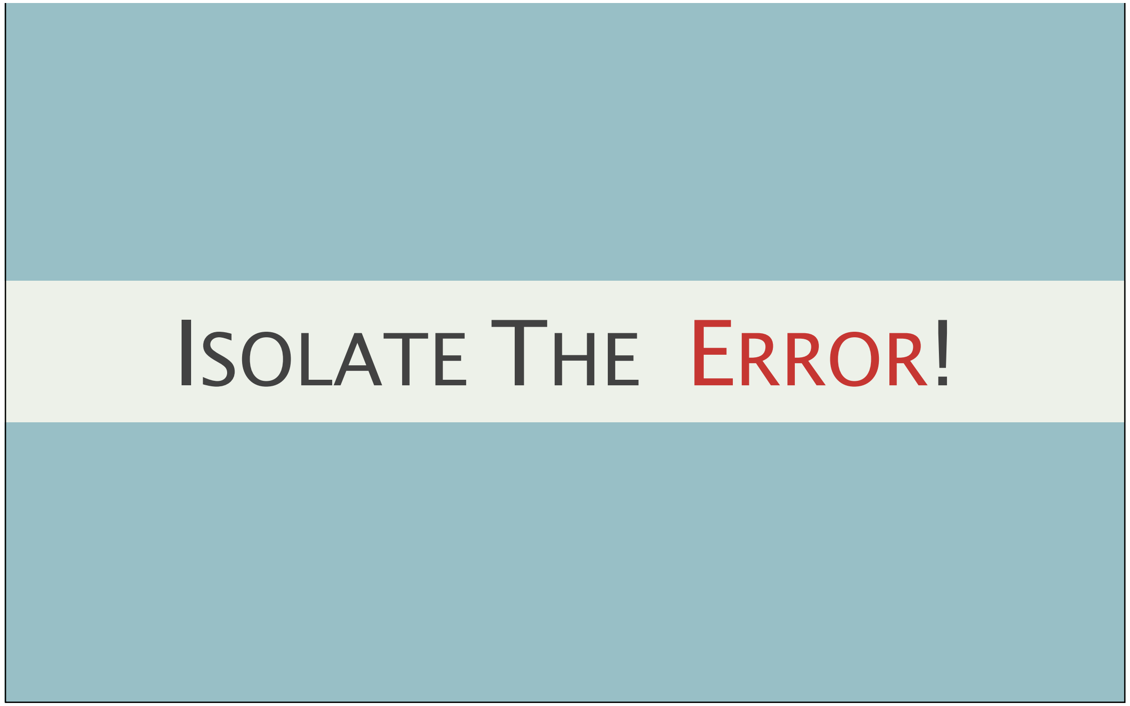
```
convert(Day) ->
  case Day of
      monday    -> 1;
      tuesday   -> 2;
      wednesday -> 3;
      thursday  -> 4;
      friday    -> 5;
      saturday  -> 6;
      sunday    -> 7;
      Other ->
         {error, unknown_day}
  end.
```

BANG FOR THE BUCK



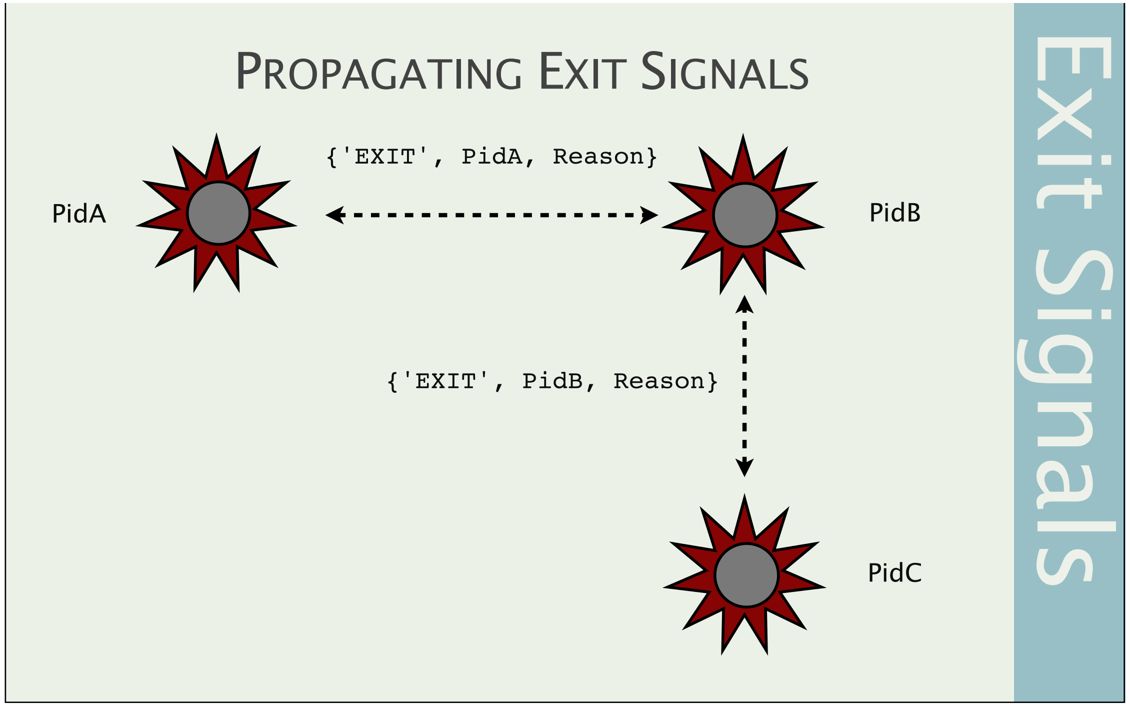
Source: http://www.slideshare.net/JanHenryNystrom/productivity-gains-in-erlang

You spend 3x the time on solving the actual problem (App) and much less on all sorts of other things.



ISOLATE THE ERROR!

Runtime Error
Do not use the word crash.
No shared memory -> Restart the process. Recreate the State.



Propagating Exit Signals

PidA

{'EXIT', PidA, Reason}

PidB

{'EXIT', PidB, Reason}

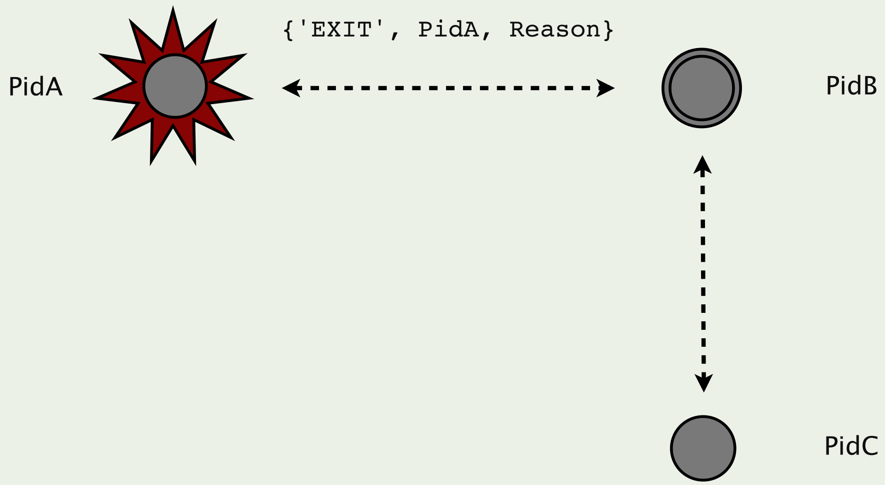
PidC

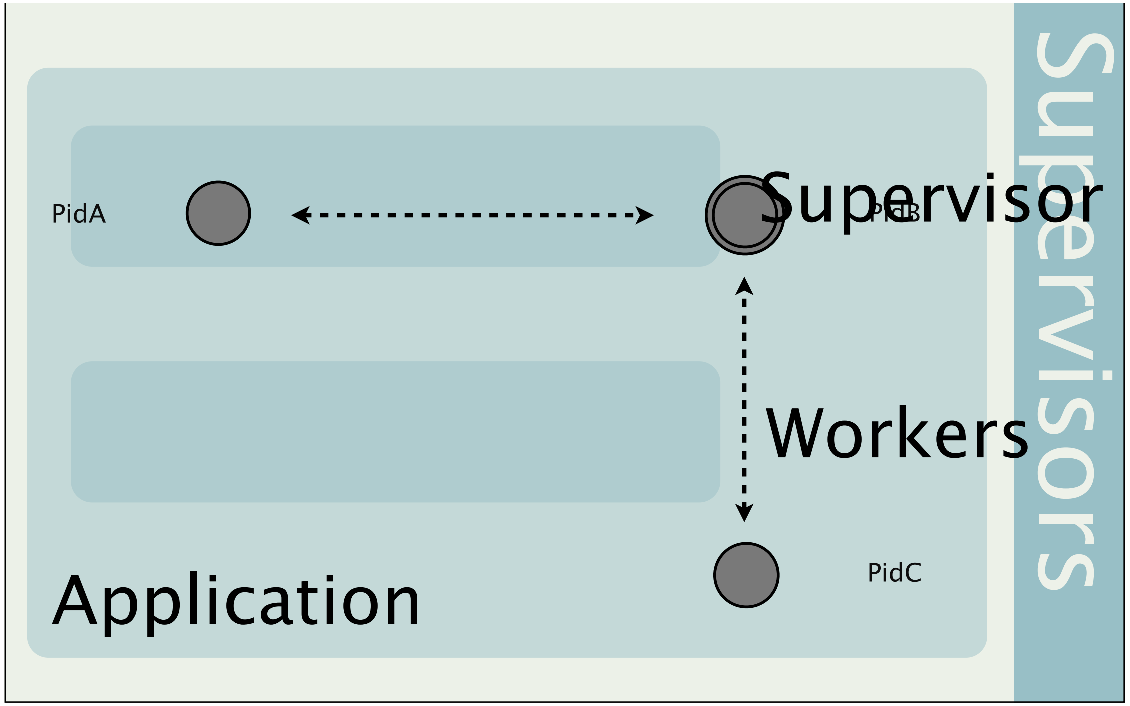
Exit Signals

Explain

Links, Exit Signals and trapping exits

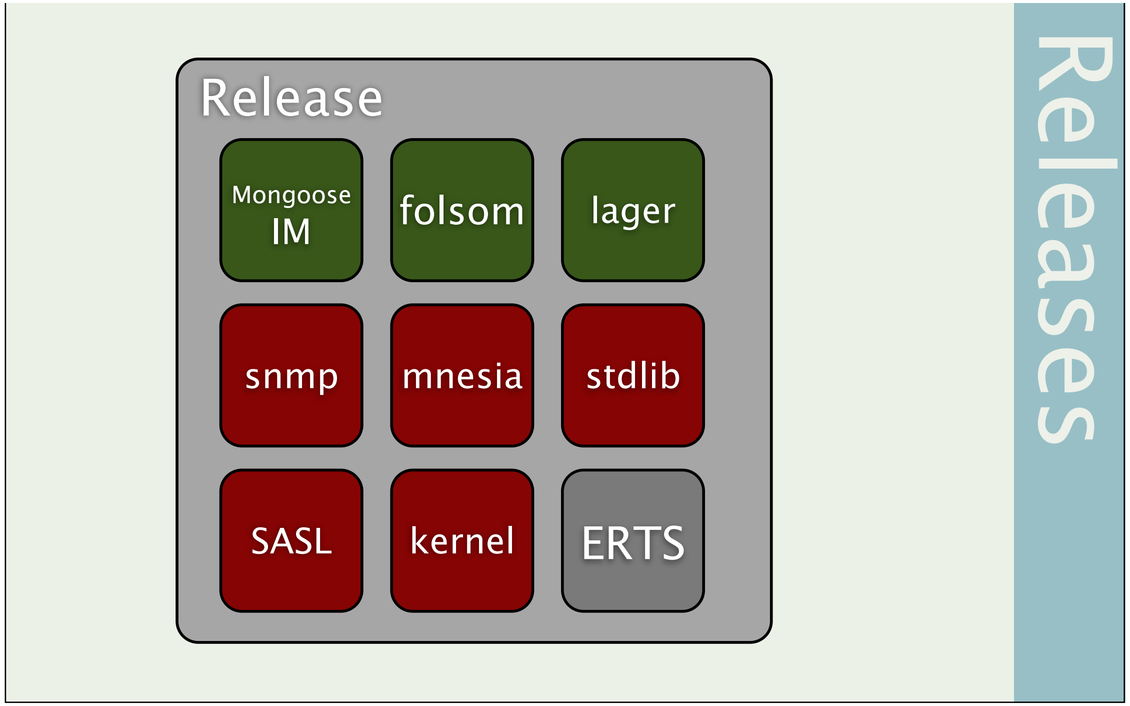
Trapping an Exit Signal





Handle dependencies.

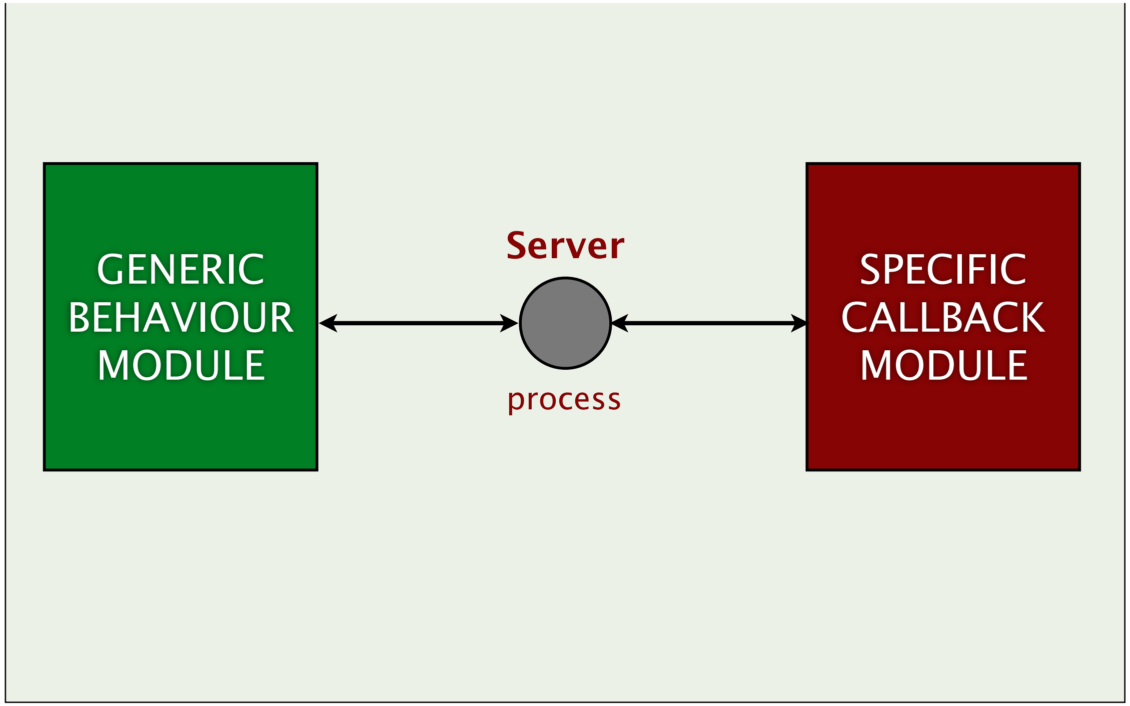
- An application is a logical unit of processes and modules grouped together to perform a given task
- Application = Collection of resources loaded, started and stopped as one
- Contains supervision tree. Workers can be implemented using generic behaviours



- Complete Erlang systems are built as releases
- A release is: a version of the Erlang Run Time System (ERTS). A set of OTP applications that work together
- Releases allow to start, stop, and manage applications in a standard manner
- Releases can be upgraded or downgraded as a unit
- Applications which come as part of OTP
- Applications the programmer writes

BEHAVIOURS

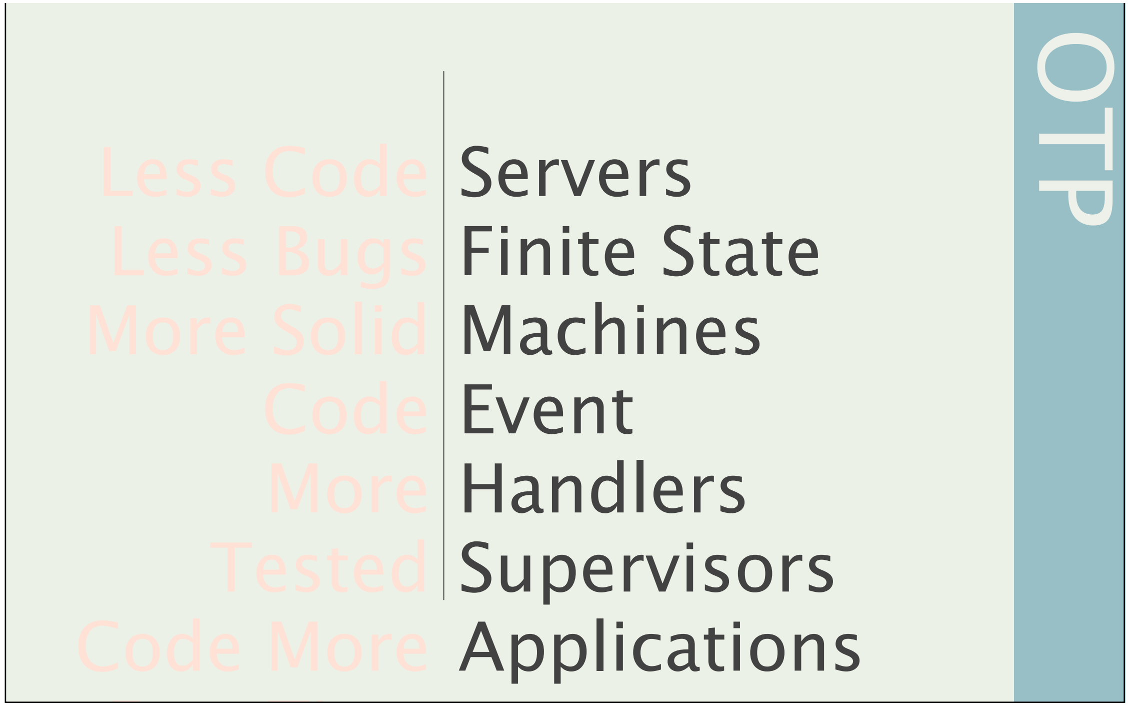
OTP Behaviours are a formalisation of design patterns
Processes share similar structures and life cycles , started, receive messages & send replies, terminate
Even if they perform different tasks, they will perform them following a set of patterns
Each design pattern solves a specific problem



The idea is to split the code in two parts

The generic part is called the **generic behaviour,**provided as library modules

The specific part is called the **callback module, implemented by programmer**

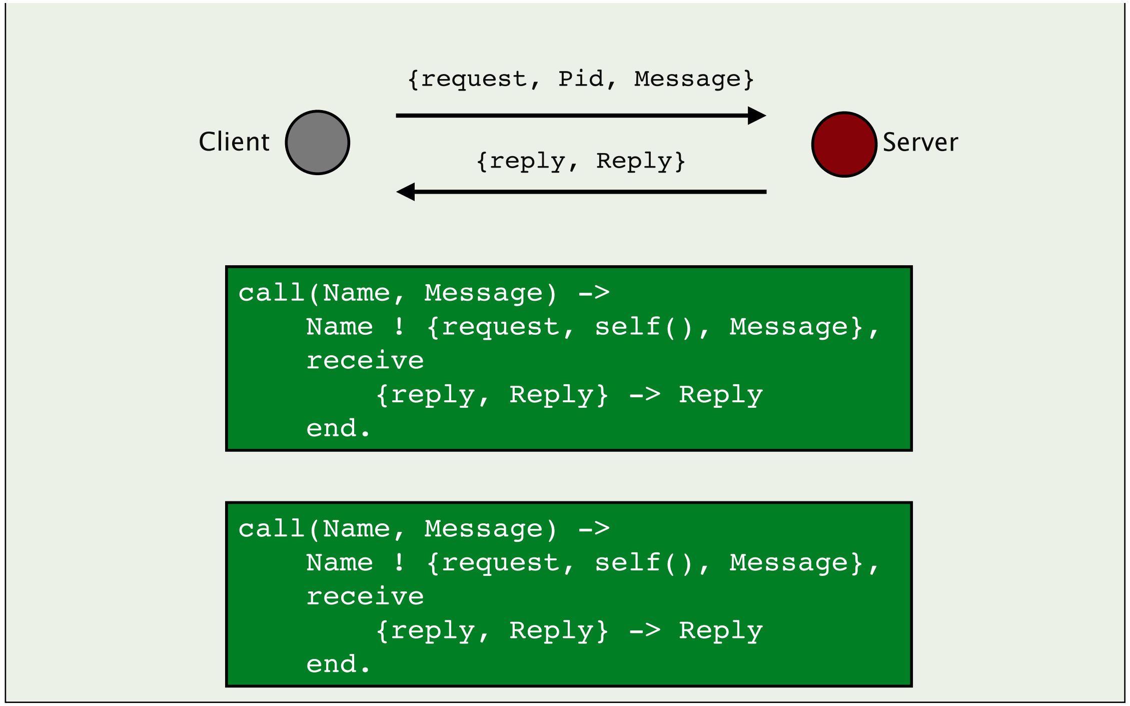


OTP

Less Code
Less Bugs
More Solid
Code
More
Tested
Code More

Servers
Finite State
Machines
Event
Handlers
Supervisors
Applications

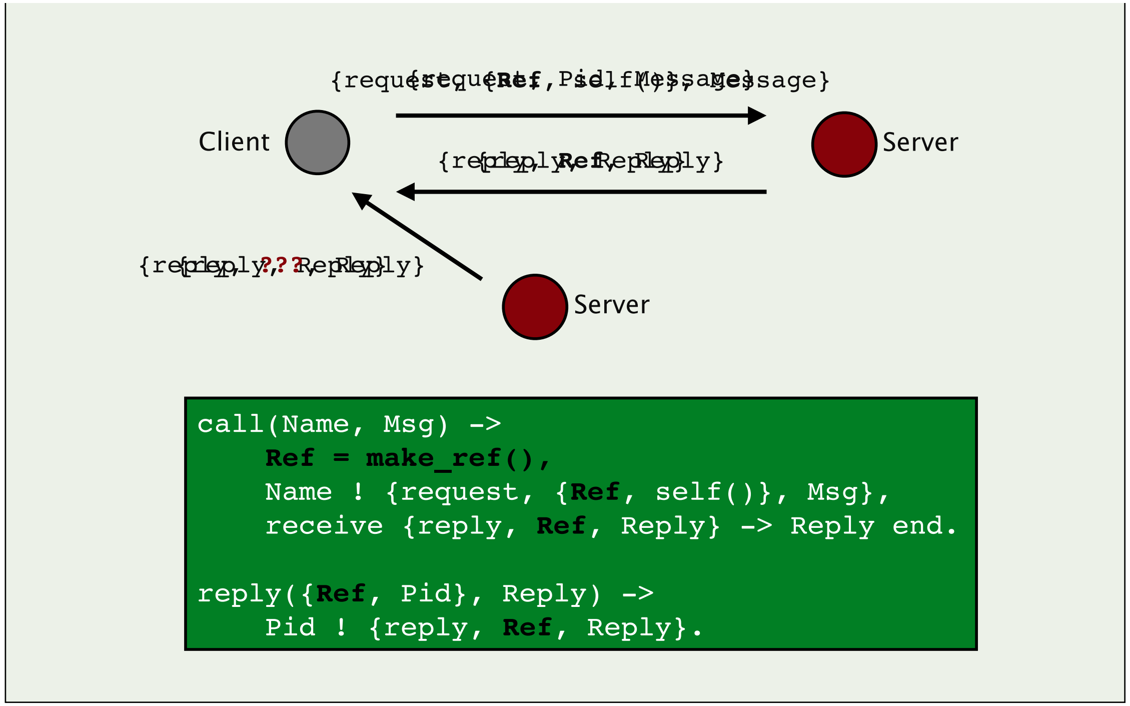
Generic: start, stop, receive and send messages.
Specific: Server state, messages, handling requests (+reply)
Specific know nothing about the generic.
generic servers, fsm, event handlers, supervisors, roll out your own



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

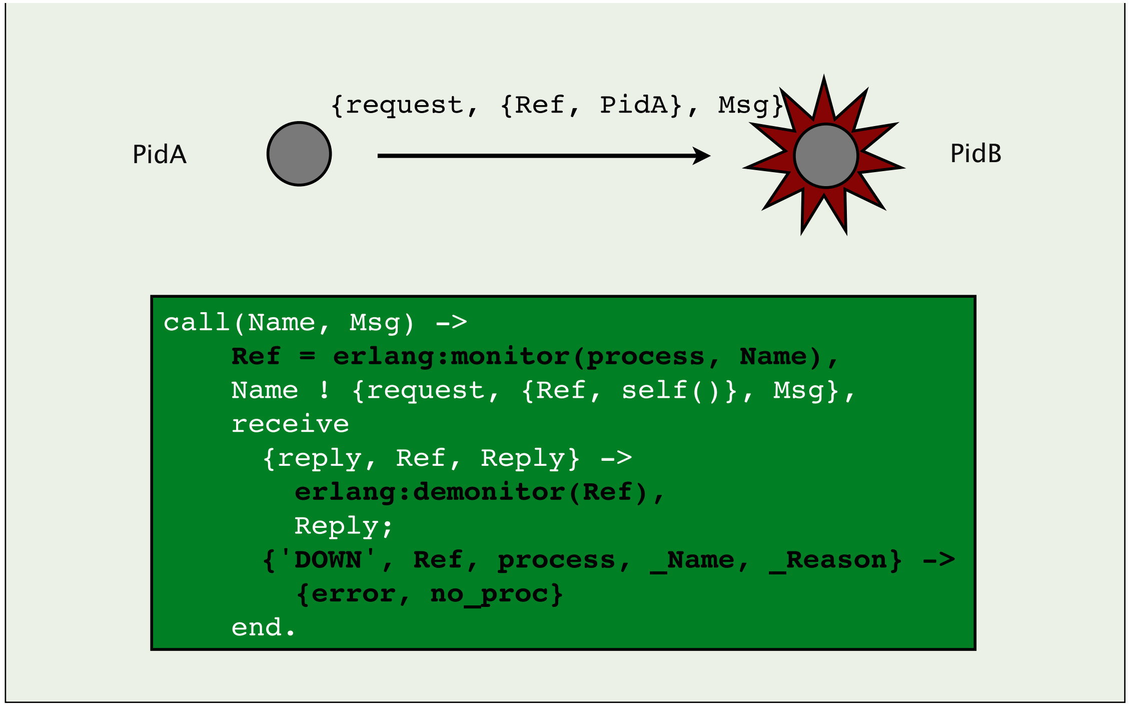
```
call(Name, Message) ->
    Name ! {request, self(), Message},
    receive
        {reply, Reply} -> Reply
    end.
```

9-5 programmer will not think of all error cases.
Concurrency is tricky. Deadlocks, race conditions, mutexes, critical sections.



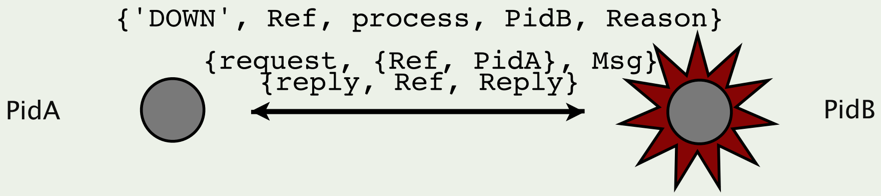
```
call(Name, Msg) ->
    Ref = make_ref(),
    Name ! {request, {Ref, self()}, Msg},
    receive {reply, Ref, Reply} -> Reply end.

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

TODO Fix Animation



```
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.
```

Fix animation



{'DOWN', Ref, process, PidB, Reason}

{request, {Ref, PidA}, Msg}

{reply, Ref, Reply}

PidA

PidB

```
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

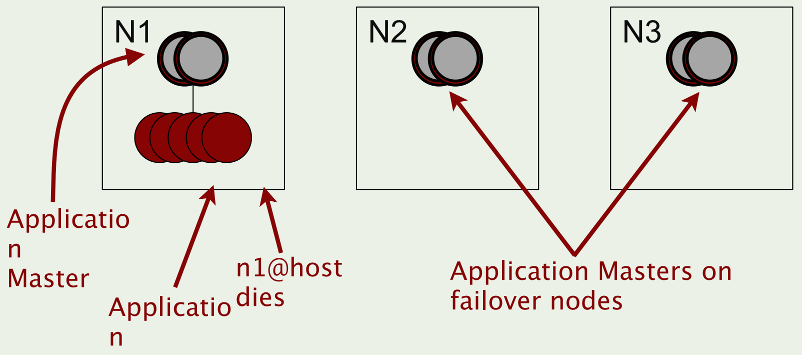
Monitoring

Distribution

BEHAVIOU

Automatic Takeover

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



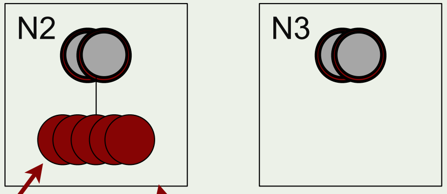
Applicatio
n
Master

Applicatio
n

n1@host
dies

Application Masters on
failover nodes

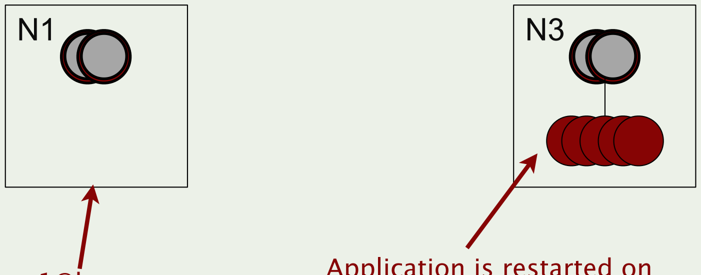
```
{myApp, 2000, {n1@host, {n2@host, n3@host}]}
```



Application is
restarted on n2@host

n2@host
dies

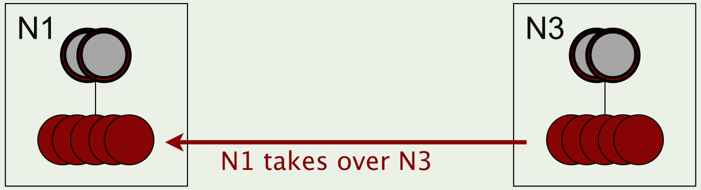
```
{myApp, 2000, {n1@host, {n2@host, n3@host}]}
```

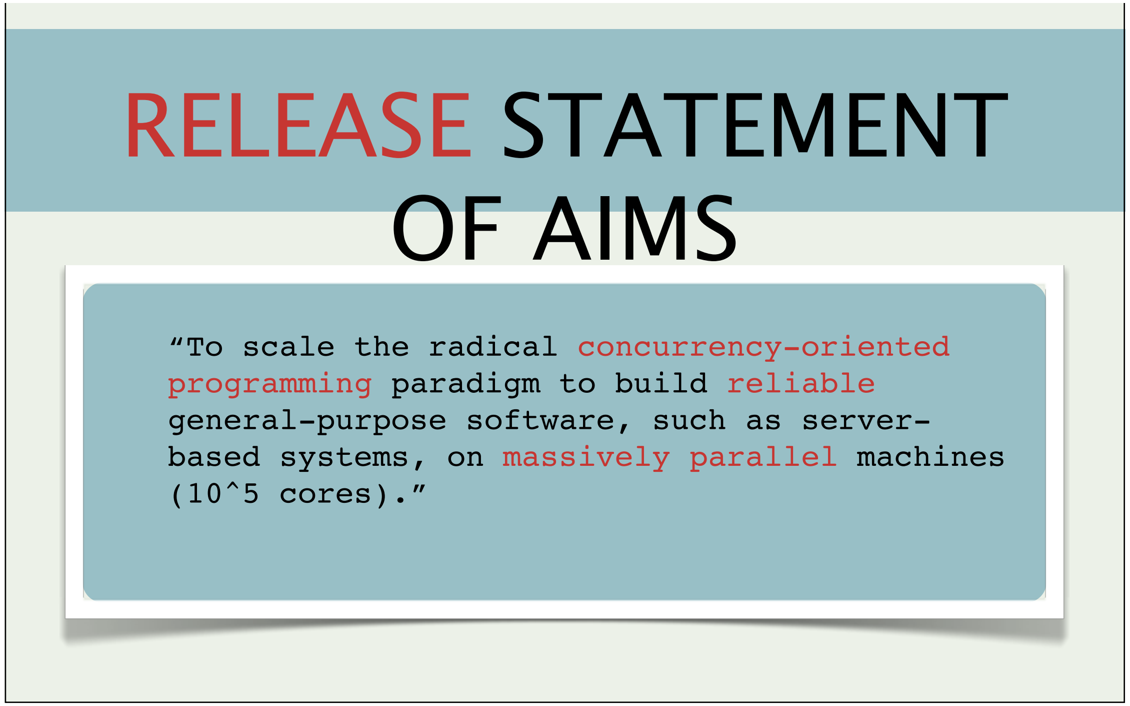


n1@host comes back up

Application is restarted on n3@host

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

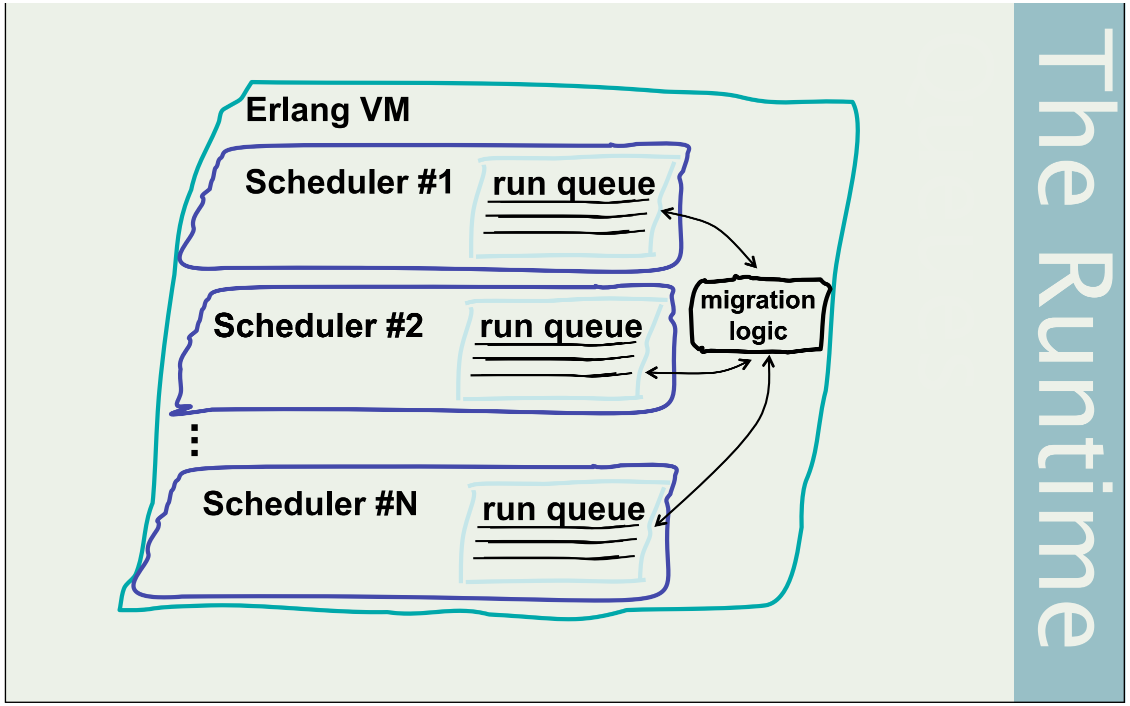




RELEASE STATEMENT OF AIMS

> “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).”

Until recently, every 18 months, computing power doubled. Moore’s law came about.
Million cores within our lifetime, 100,000s will become common place.
Consortium of companies and universities.
Bring OTP to the next level
European Union Seventh Framework Programme (FP7/2007-2013) , aprox. 3.5 million Euro

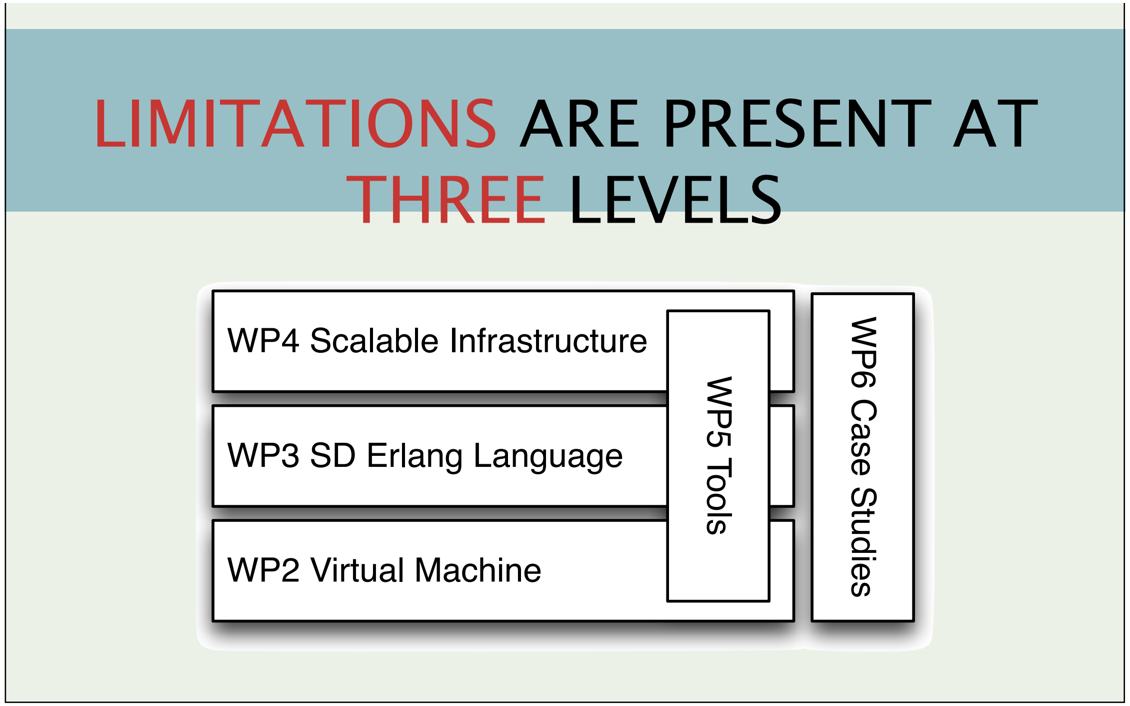


The Runtime

1 scheduler per core

Effort in migration logic among the cores.

Heriot-Watt, University of Kent, Uppsala University,
Institute of Communications & Computer Systems (Athens)
Electricite de France, Erlang Solutions (Case Studies), Ericsson



LIMITATIONS ARE PRESENT AT THREE LEVELS

WP4 Scalable Infrastructure

WP3 SD Erlang Language

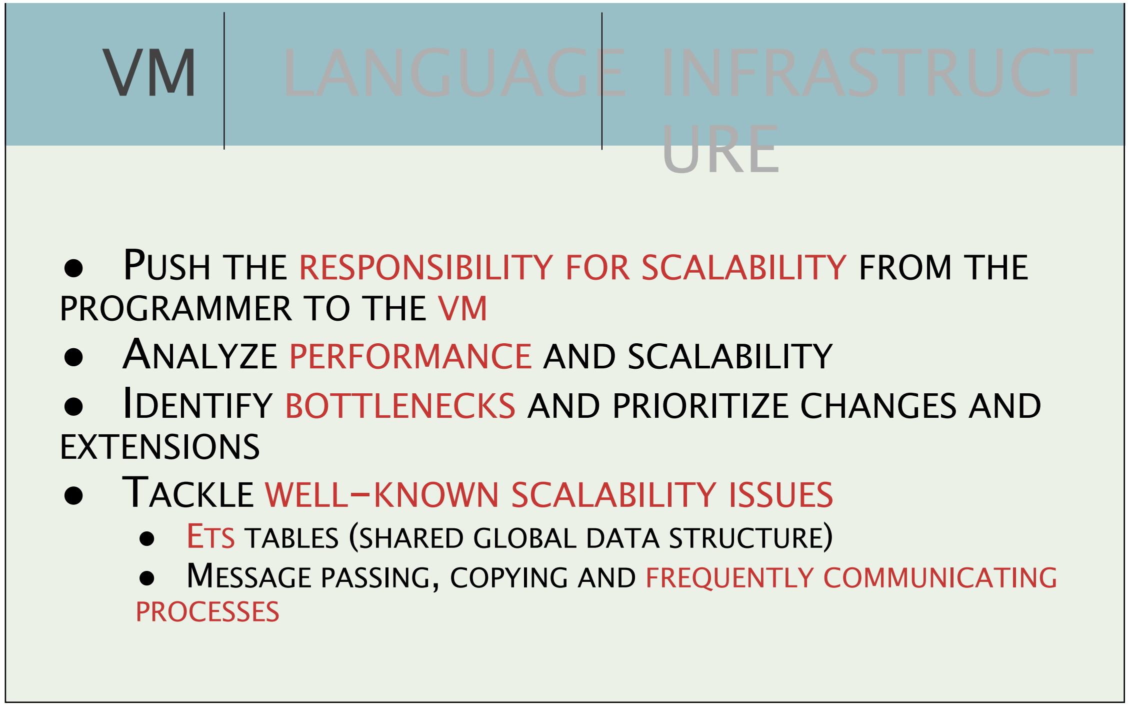
WP2 Virtual Machine

WP5 Tools

WP6 Case Studies

Erlang is too much small cluster focused.

* Cover / Stratch across
* There might be some overlap between layers

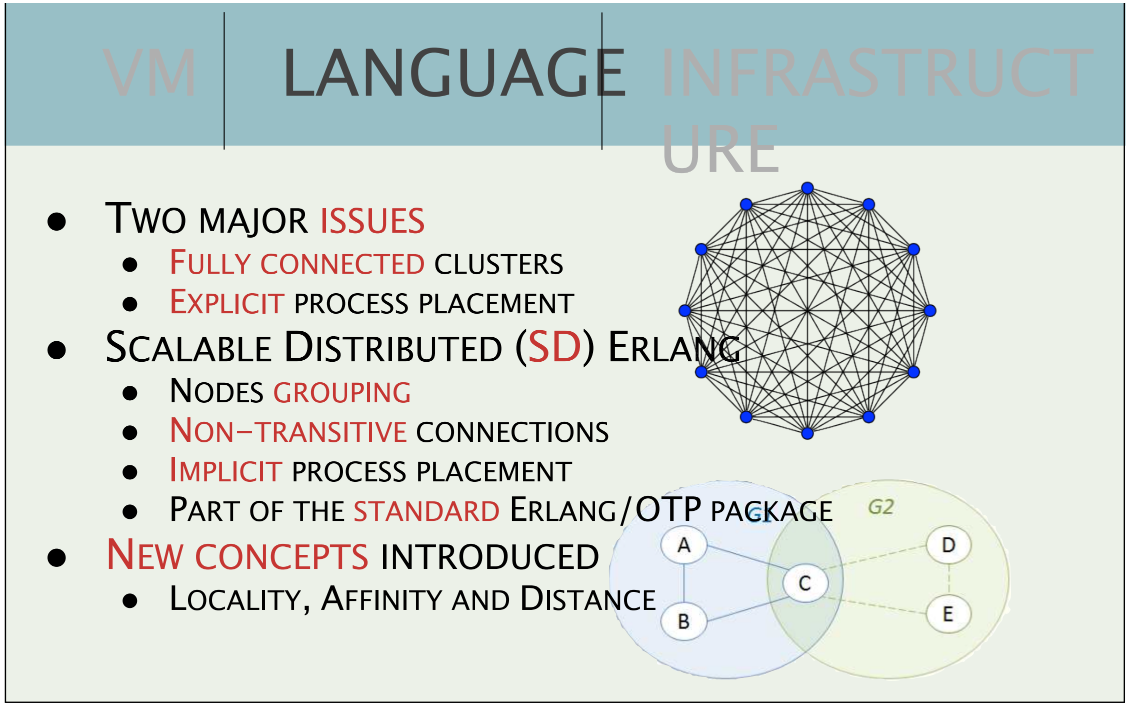


VM | LANGUAGE | INFRASTRUCTURE

- 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

Evolve the Erlang virtual machine – which implements Erlang on each core – so that it can work effectively in large-scale multicore systems.

Percept2 - visualisation

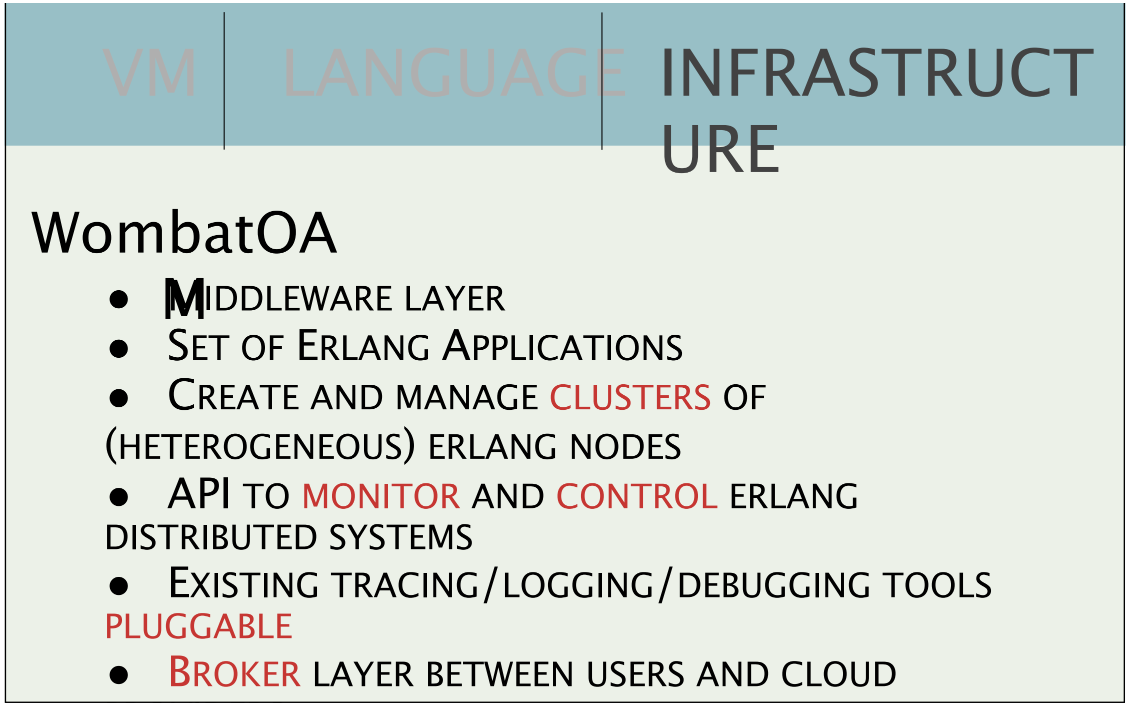


VM | LANGUAGE | INFRASTRUCTURE

- 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

Scalable Distributed (SD) Erlang, provides constructs to control how computations are spread across multicore platforms, and coordination patterns to allow SD Erlang to effectively describe computations on large platforms, while preserving performance portability.

Tools - Scheduler, visualising process migration.



VM | LANGUAGE | INFRASTRUCTURE

WombatOA

- 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

* Basic Erlang has the ability to go in and monitor what is going on in any node you can attach yourself to.
* But no tool exists to manage a big number of nodes in a coherent fashion.
* Cloud Provider. Analyse metrics which are on an OS level. CPU load, memory, etc
* Scaling should however be based on the application layer
* O&M which monitor. Hidden nodes.
* Nagios & other tools with plugins.

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

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

Evaluate Now!

@LeHoff