Erlang Solutions Ltd

An Introduction to Erlang

From behind the trenches…

GOTO Amsterdam
Amsterdam, May 25th 2012

Francesco Cesarini
Founder, Technical Director

@FrancescoC
francesco@erlang-solutions.com

So Here I Am….

Erlang

Telecom Applications: Issues

Complex

No down time

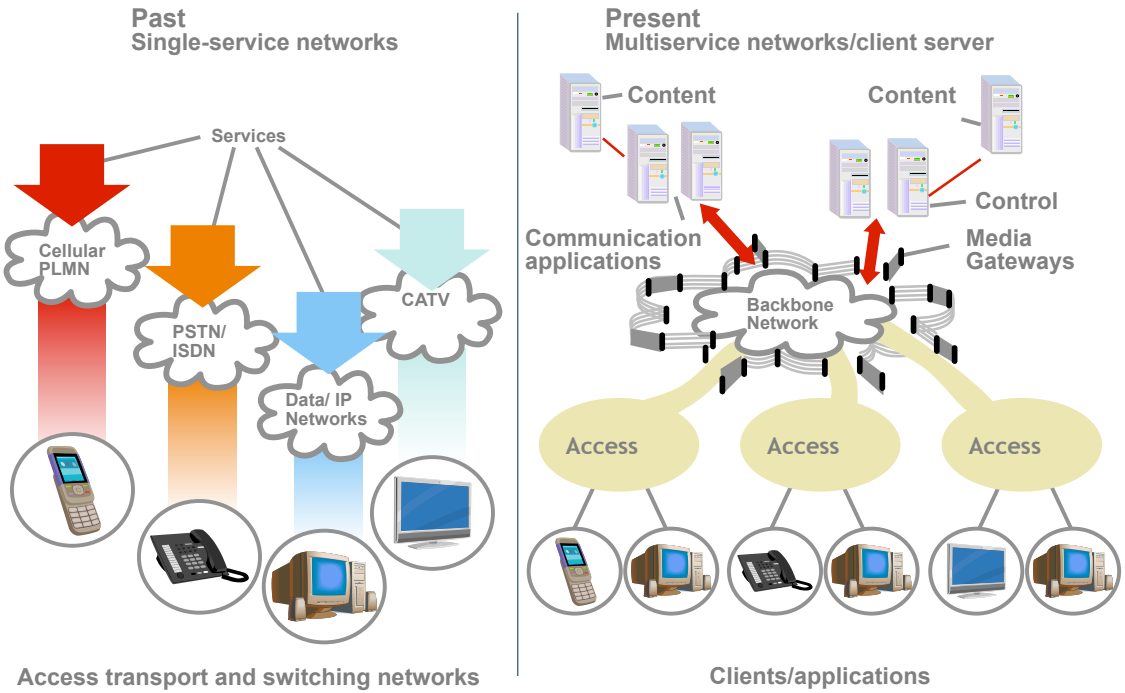
Scalable

Maintainable

Distributed

vs

Time to Market



Erlang

The Ancestors

Languages like SmallTalk, Ada, Modula or Chill

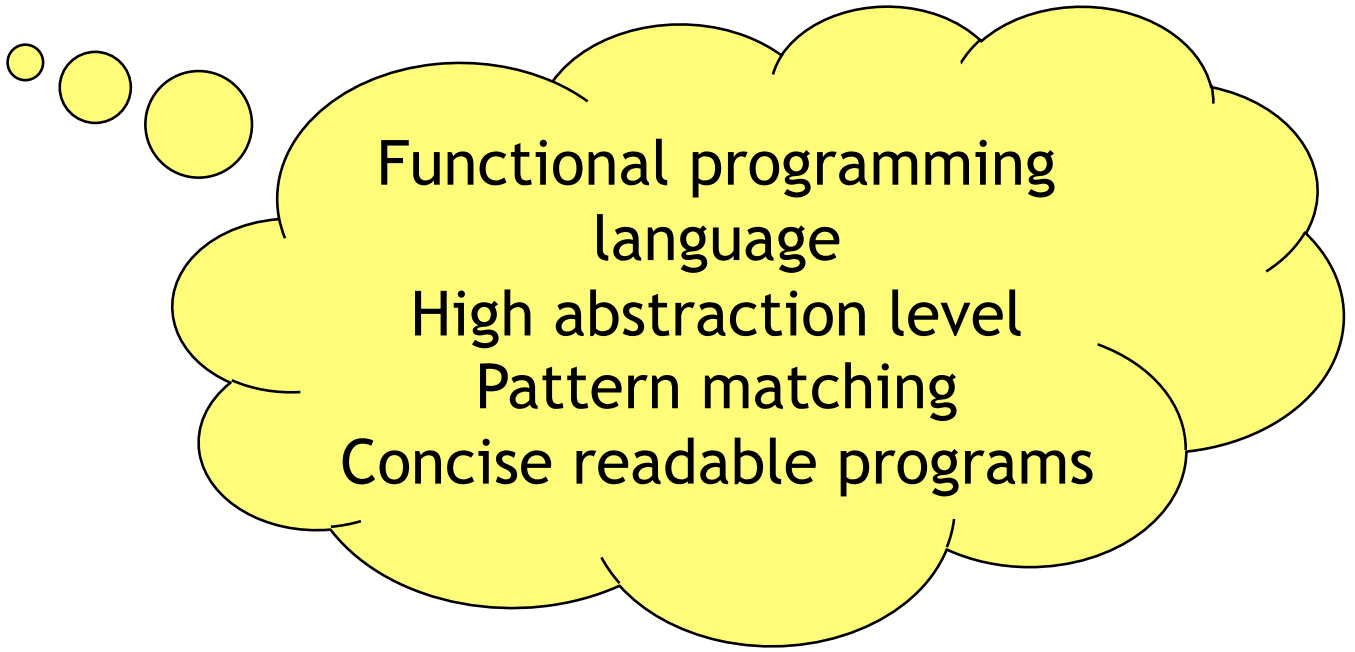
Functional languages like ML or Miranda

Logical languages like Prolog

Erlang

Erlang Highlights

- Declarative
- Concurrent
- Robust
- Distributed
- Hot code loading
- Multicore Support
- OTP



Erlang

Erlang Highlights: Factorial

Factorial using Recursion

Definition

$$n! = \begin{cases} 1 & n = 0 \\ n*(n-1)! & n \geq 1 \end{cases}$$

```
Eshell V5.0.1  (abort with ^G)
1> c(ex1).
{ok,ex1}
2> ex1:factorial(6).
720
```

Implementation

```
-module(ex1).
-export([factorial/1]).

factorial(0) ->
   1;
factorial(N) when N >= 1 ->
   N * factorial(N-1).
```


Erlang

Erlang Highlights: High-level Constructs

QuickSort using List Comprehensions

```
-module(ex2).
-export([qsort/1]).

qsort([Head|Tail]) ->
   First = qsort([X || X <- Tail, X =< Head]),
   Last  = qsort([Y || Y <- Tail, Y > Head]),
   First ++ [Head] ++ Last;
qsort([]) ->
   [].
```

```
Eshell V5.0.1  (abort with ^G)
1> c(ex2).
{ok,ex2}
2> ex2:qsort([7,5,3,8,1]).
[1,3,5,7,8]
```

"all objects Y taken from the list Tail, where Y > Head"

Erlang

Erlang Highlights: High-level Constructs

Parsing a TCP packet using the Bit Syntax

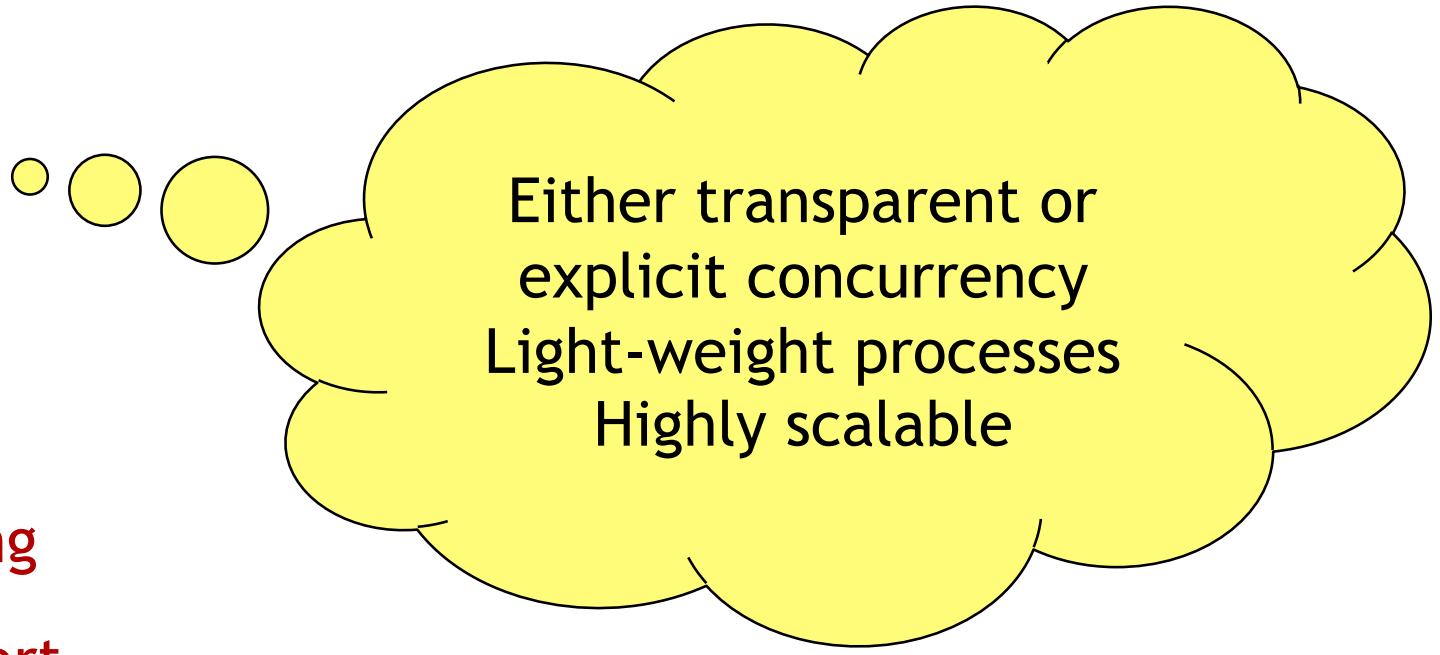
```
<< SourcePort:16, DestinationPort:16, SequenceNumber:32,
   AckNumber:32, DataOffset:4, _Reserved:4, Flags:8,
   WindowSize:16, Checksum:16, UrgentPointer:16,
   Payload/binary>> = Segment,

OptSize = (DataOffset - 5)*32,
<< Options:OptSize, Message/binary >> = Payload,
<< CWR:1, ECE:1, URG:1, ACK:1, PSH:1,
   RST:1, SYN:1, FIN:1>> = <<Flags:8>>,

%% Can now process the Message according to the
%% Options (if any) and the flags CWR, ..., FIN
```

etc…

Erlang

Erlang Highlights

- Declarative
- Concurrent
- Robust
- Distributed
- Hot code loading
- Multicore Support
- OTP



Erlang

Erlang Highlights: Concurrency

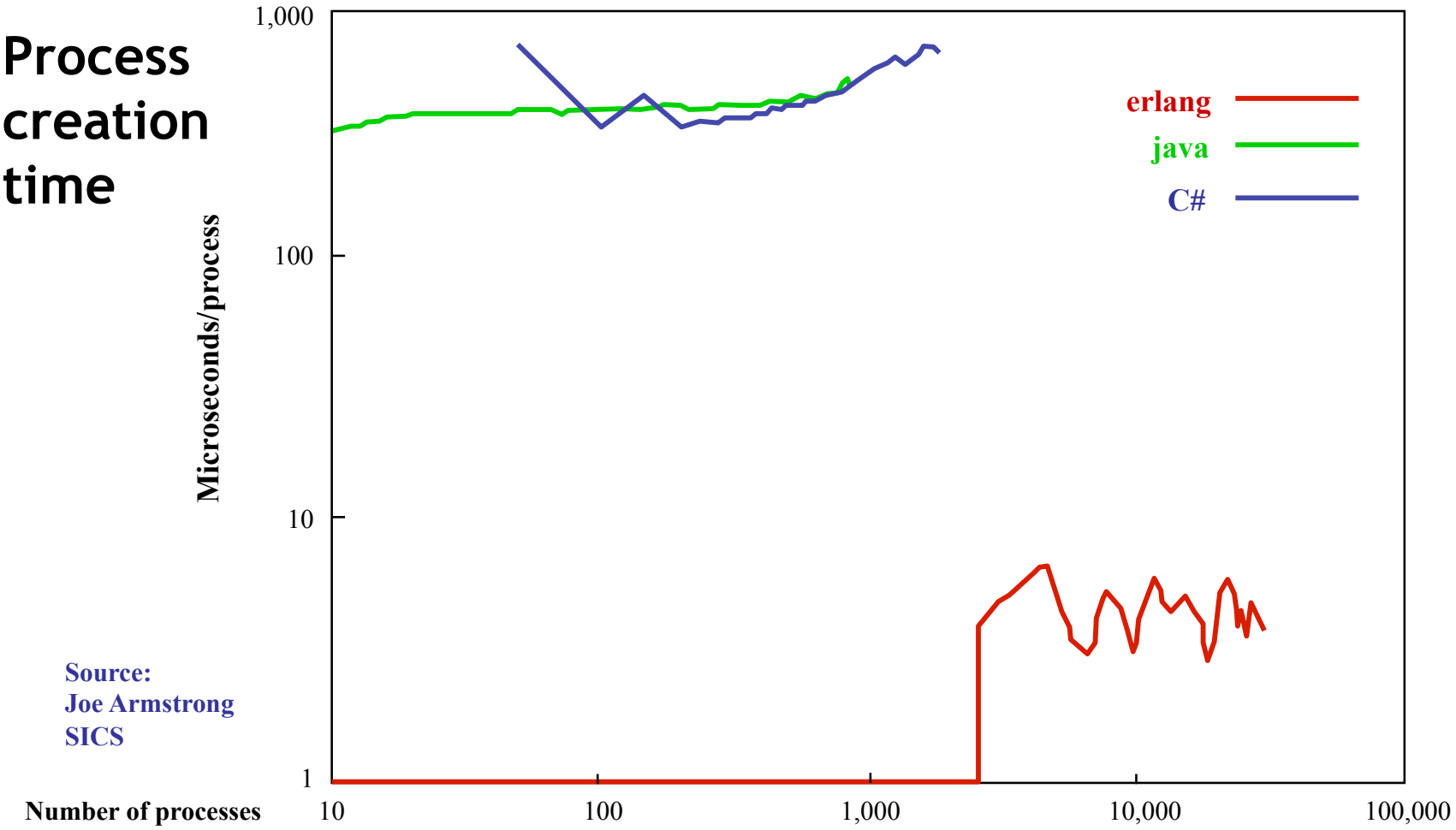
Creating a new process using spawn

```
-module(ex3).
-export([activity/3]).

activity(Name,Pos,Size) ->
    ..........
```


```
Pid = spawn(ex3,activity,[Joe,75,1024])
```

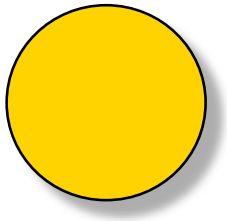
Erlang Highlights: Concurrency

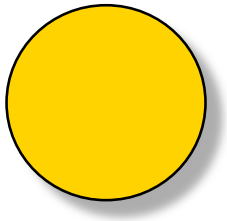
Process creation time



Erlang Highlights: Concurrency

Processes communicate by asynchronous message passing





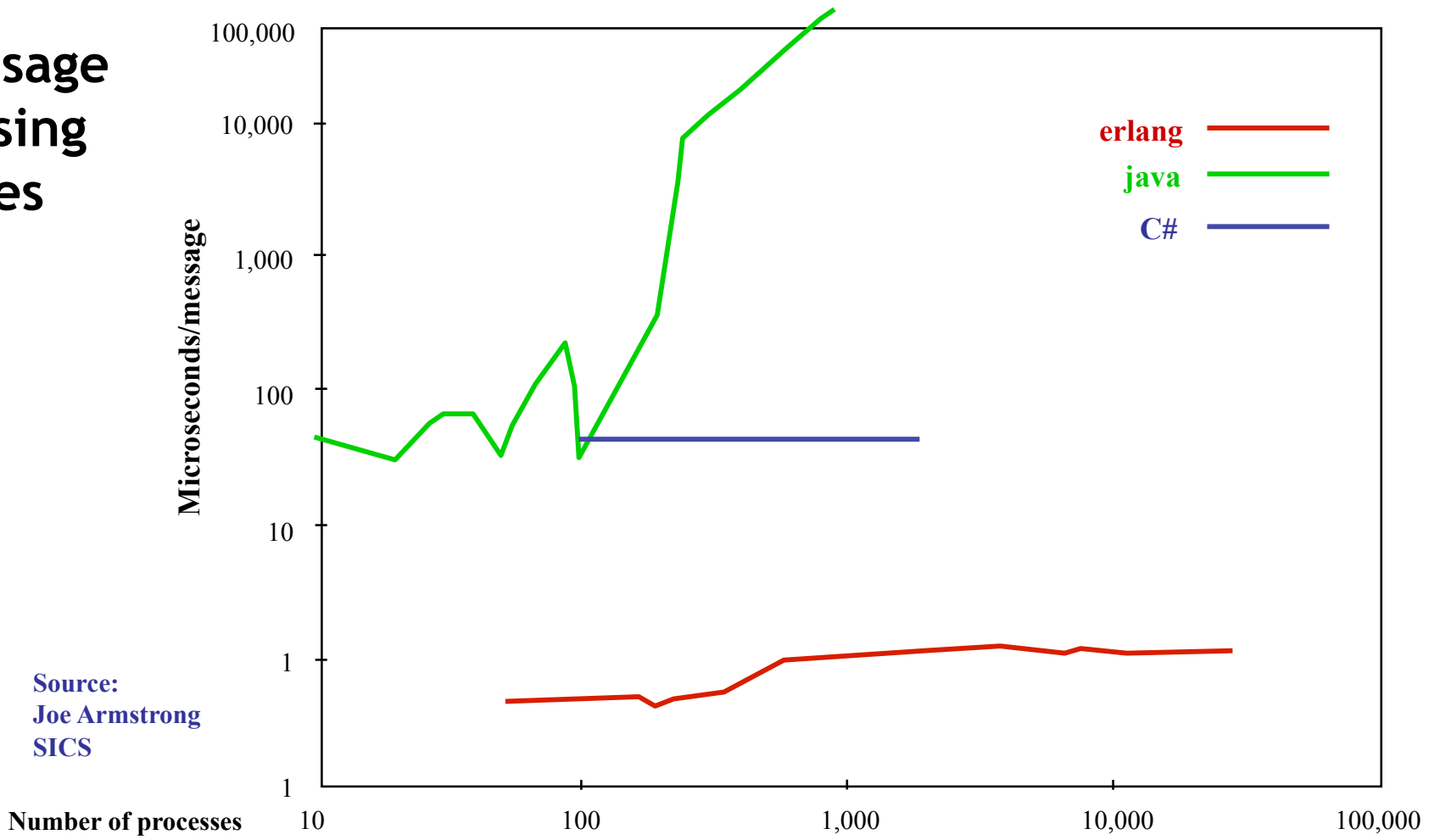
```
Pid ! {data,12,13}
```

```
receive
     {start} -> ……
     {stop} -> ……
     {data,X,Y} -> ……
end
```


Erlang

Erlang Highlights: Concurrency

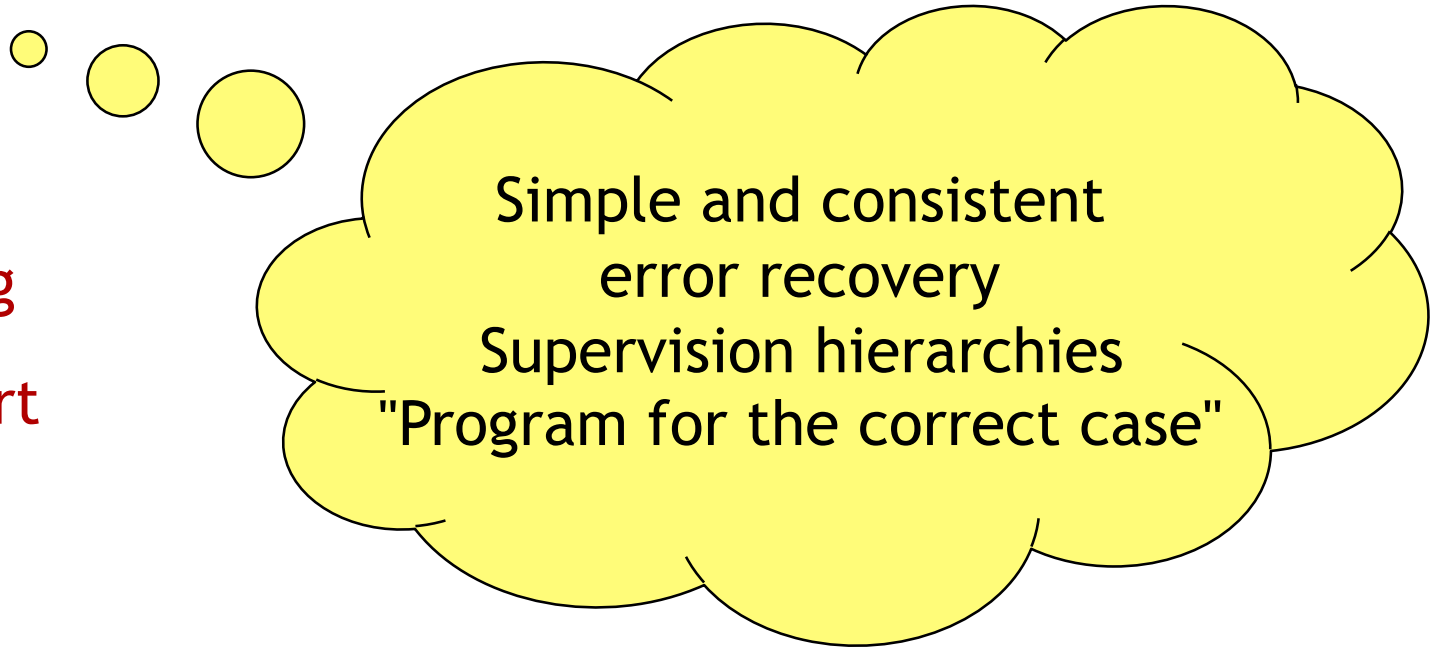
Message passing times



Erlang

Erlang Highlights

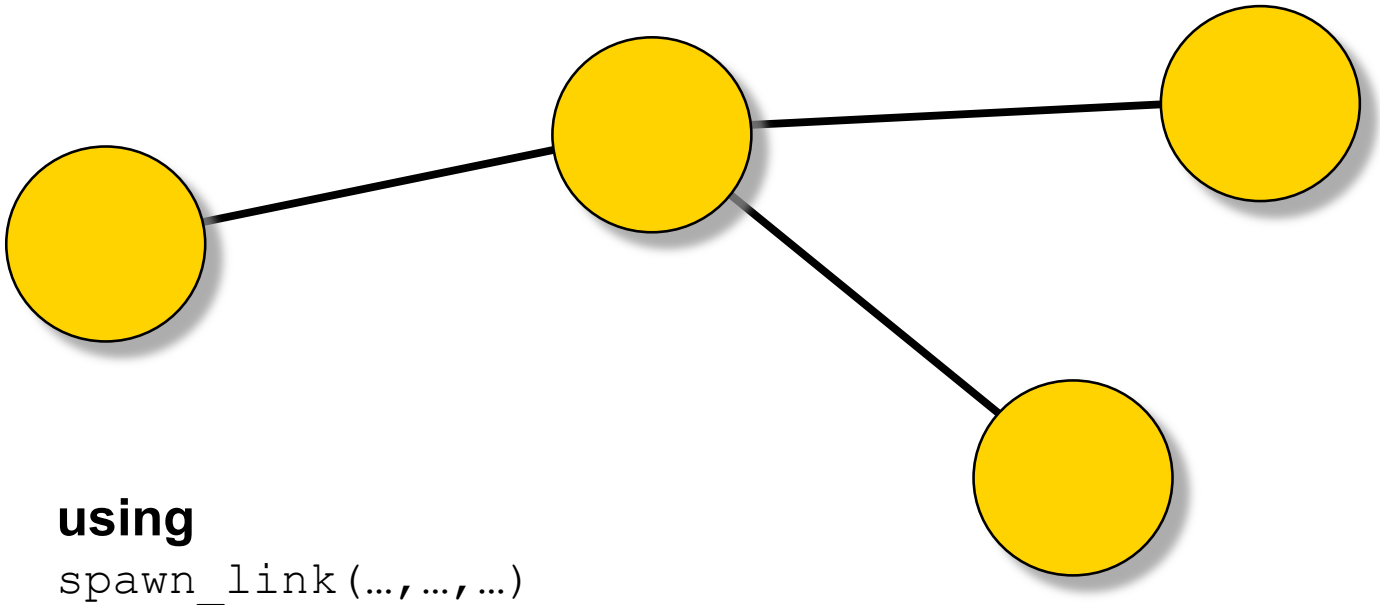
- Declarative
- Concurrent
- Robust
- Distributed
- Hot code loading
- Multicore Support
- OTP



Erlang

Erlang Highlights: Robustness

Cooperating processes may be linked together



using

```
spawn_link(…,…,…)
```

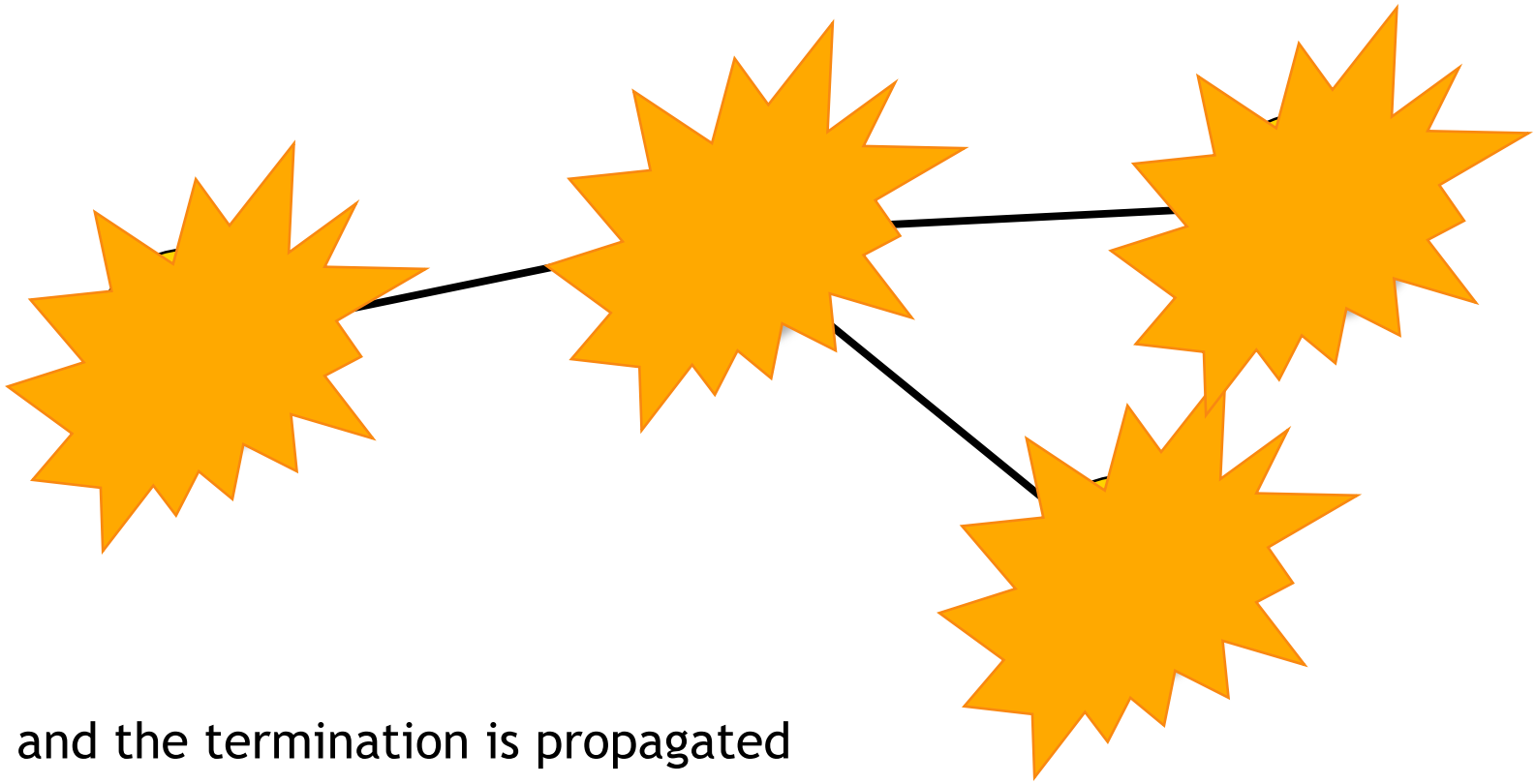
or

```
link(Pid)
```


Erlang

Erlang Highlights: Robustness

When a process terminates, an exit signal is sent to all linked processes

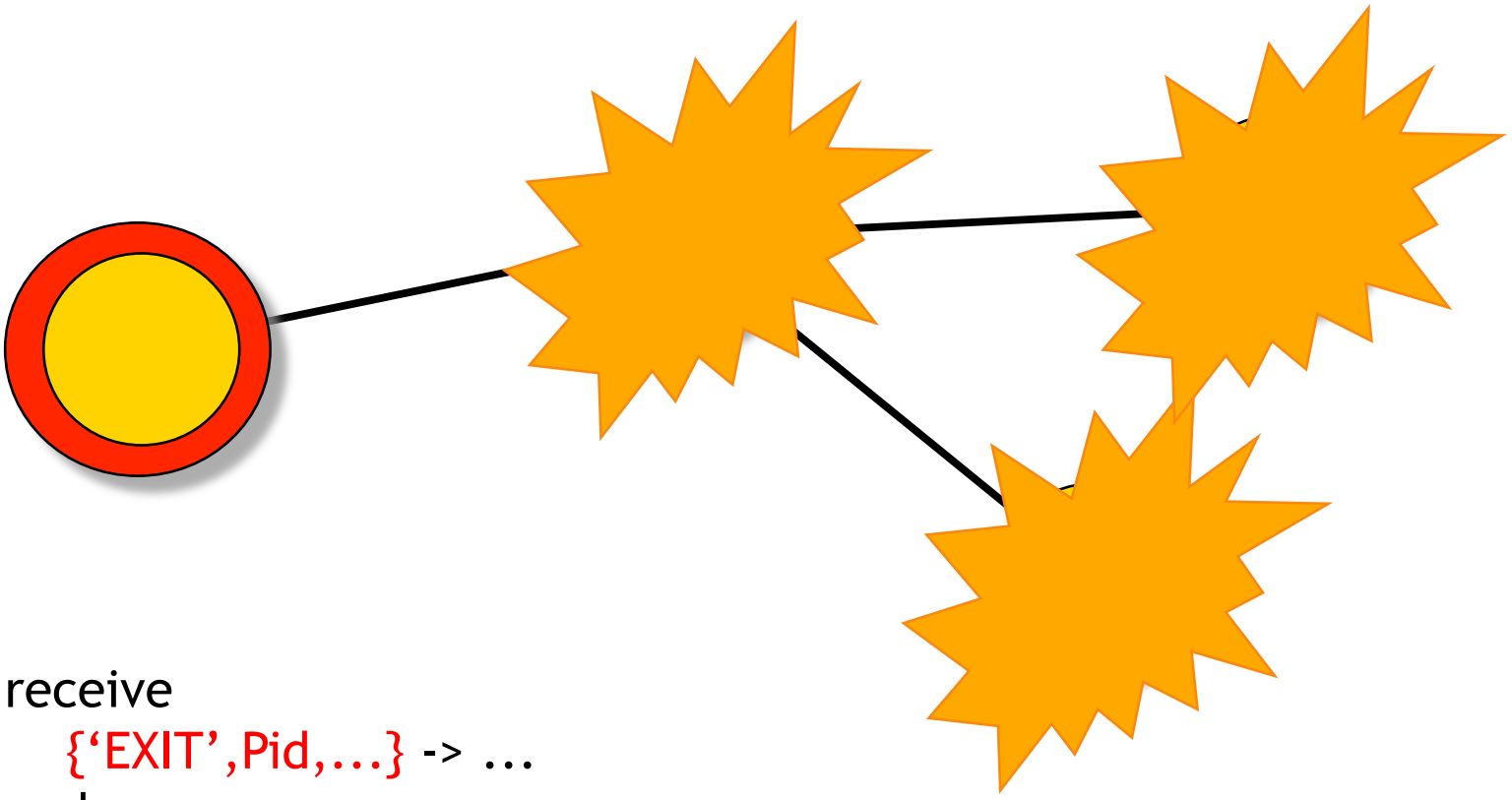


… and the termination is propagated

Erlang

Erlang Highlights: Robustness

Exit signals can be trapped and received as messages

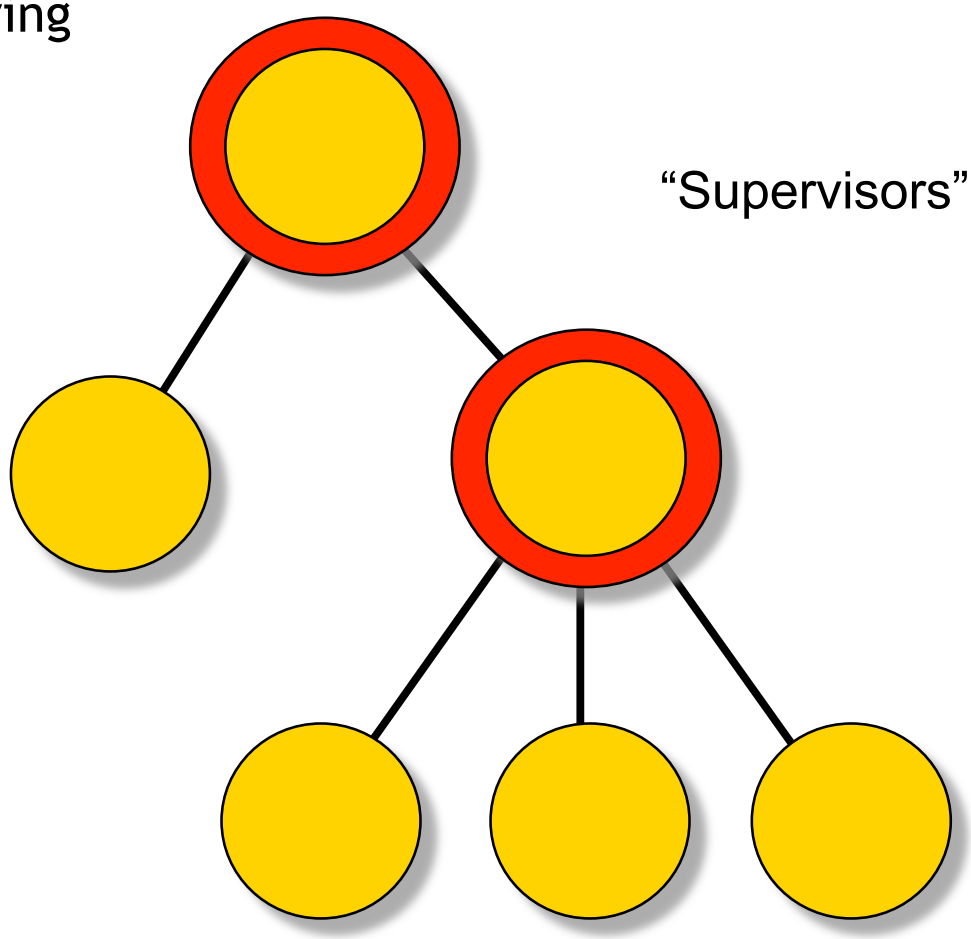


```
receive
   {‘EXIT’,Pid,...} -> ...
end
```


Erlang

Erlang Highlights: Robustness

Robust systems can be built by layering



Erlang

Erlang Highlights

Declarative

Concurrent

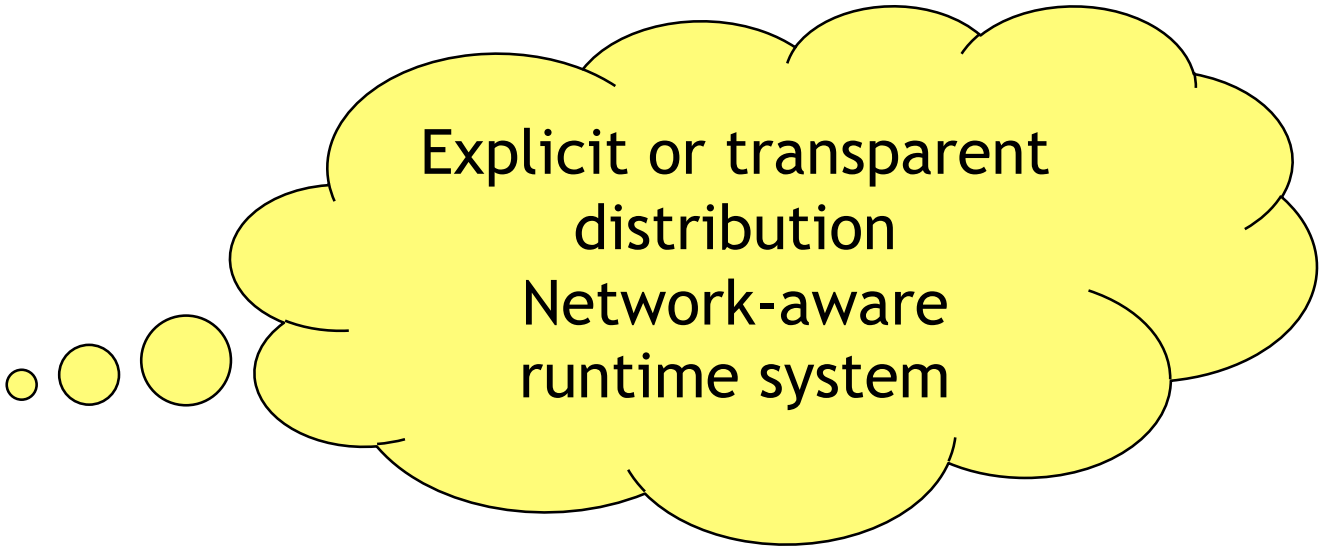
Robust

Distributed

Hot code loading

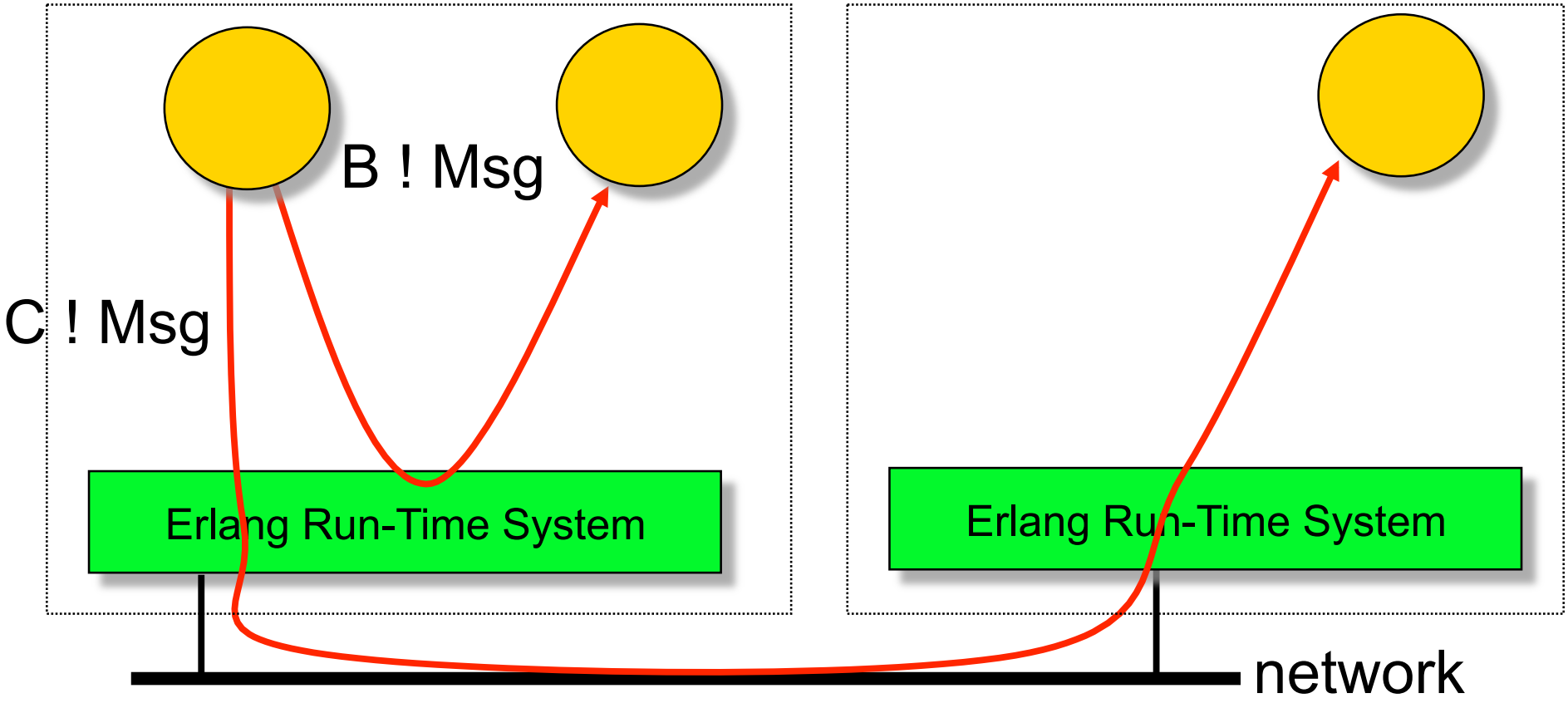
Multicore Support

OTP



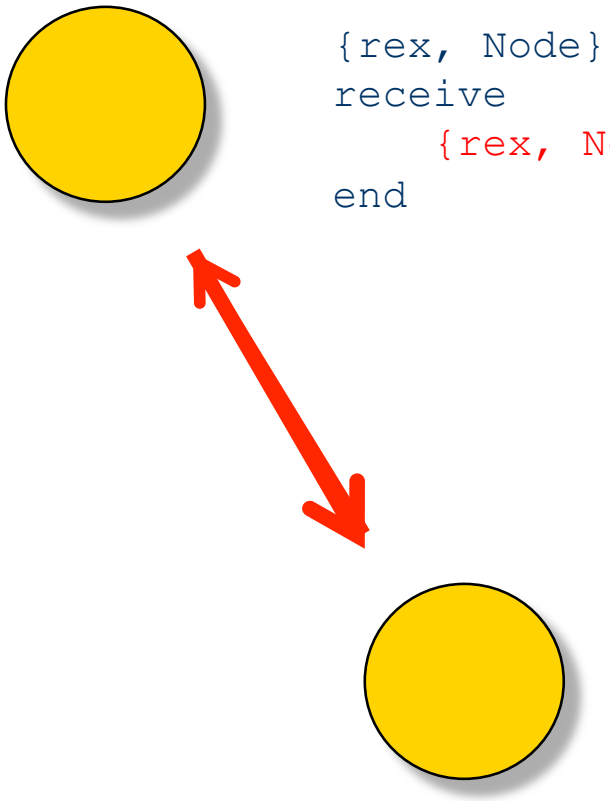
Erlang

Erlang Highlights: Distribution



Erlang Highlights: Distribution

Simple Remote Procedure Call



```
{rex, Node} ! {self(), {apply, M, F, A}},
receive
    {rex, Node, What} -> What
end
```

```
loop() ->
    receive
        {From, {apply, M, F, A}} ->
            Answer = apply(M, F, A),
            From ! {rex, node(), Answer}
            loop();
        _Other -> loop()
    end.
```


Erlang

Erlang Highlights

Declarative

Concurrent

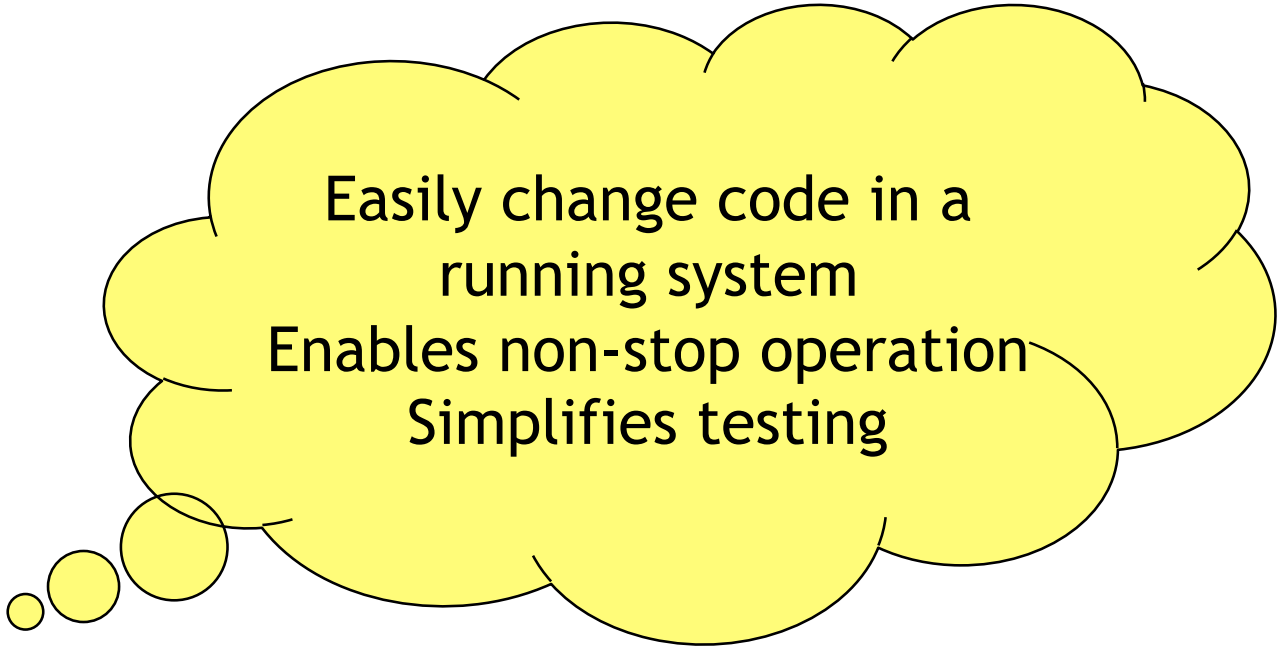
Robust

Distributed

Hot code loading

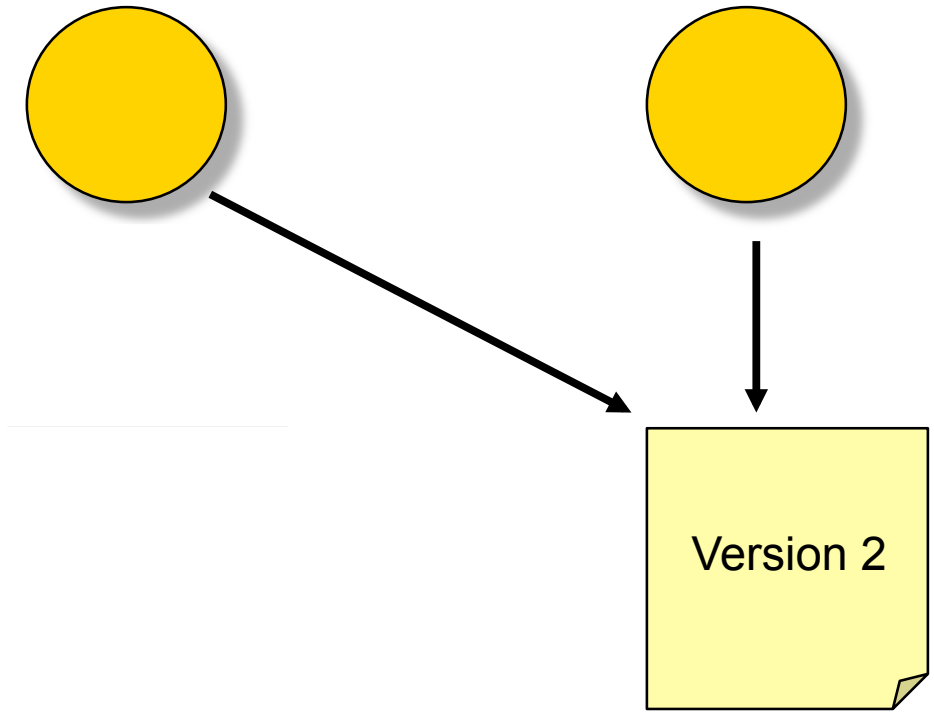
Multicore Support

OTP



Erlang

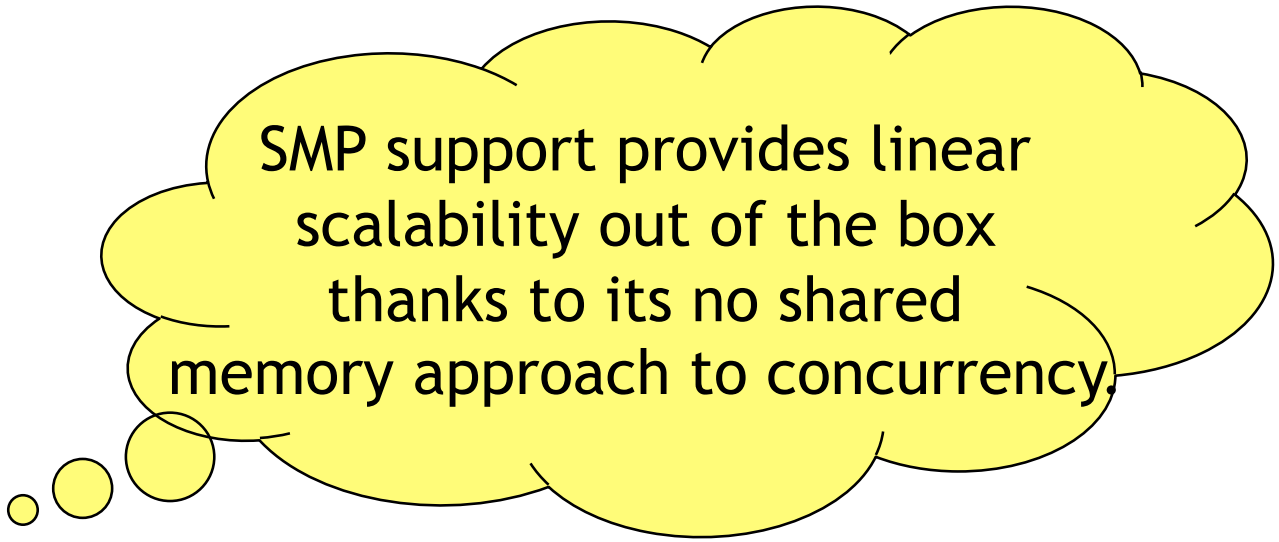
Erlang Highlights: Hot Code Swap



Erlang

Erlang Highlights

- Declarative
- Concurrent
- Robust
- Distributed
- Hot code loading
- Multicore Support
- OTP



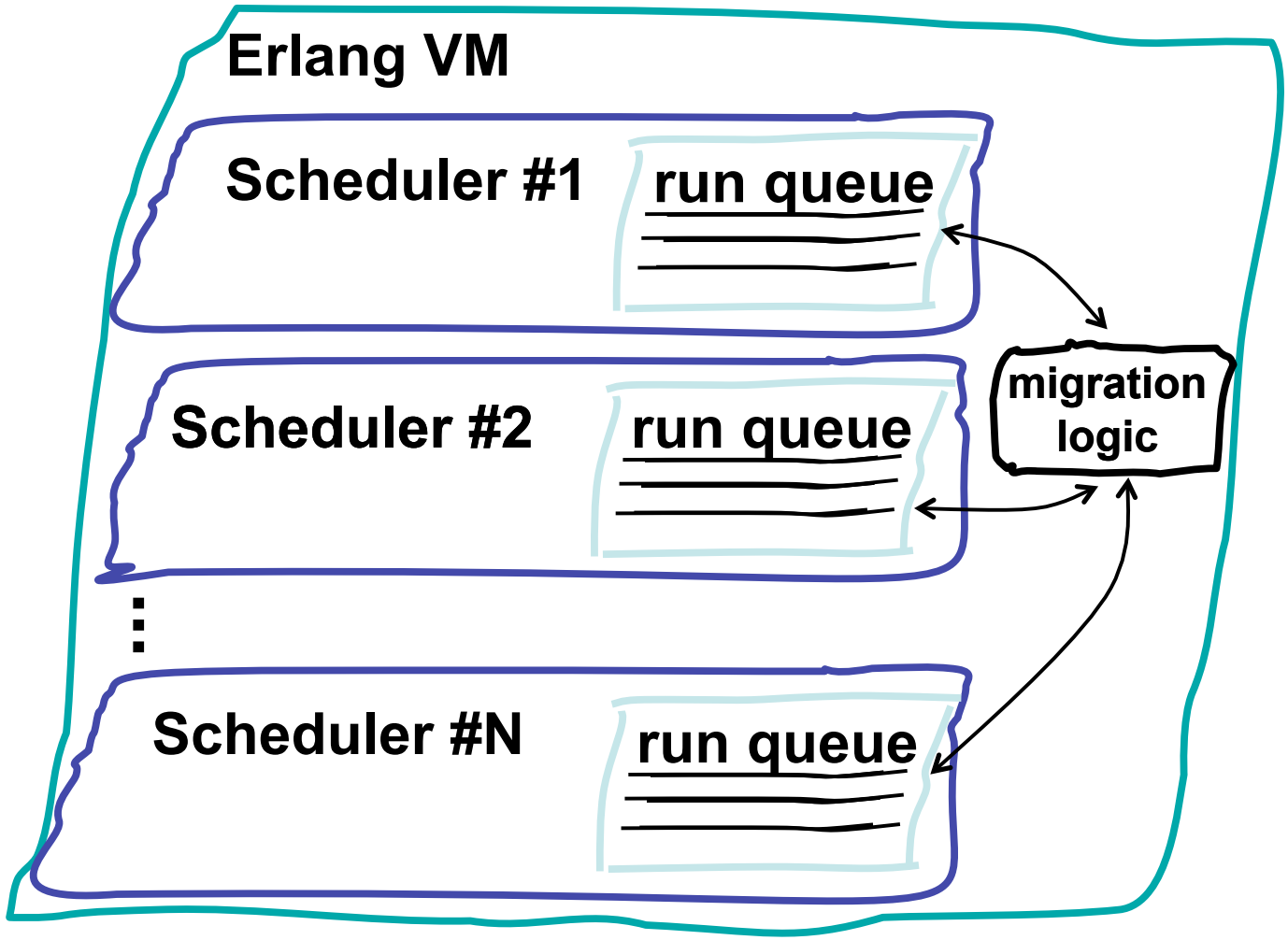
Erlang

Ericsson's strategy with SMP

Hide the problems and awareness of SMP from the programmer
Programmed in the normal style using processes for encapsulation and parallelisation

Erlang

Multicore Erlang



Erlang

Telephony Gateway Controller

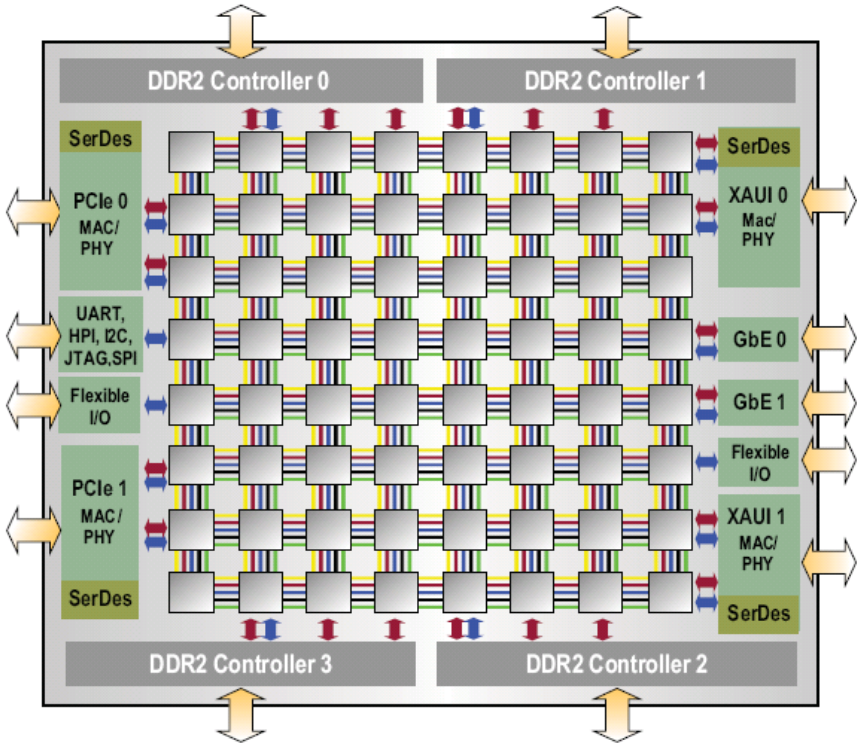
Traffic scenario	IS/GCP 1slot/board	IS/GEP Dual core One core running 2slots/board	IS/GEP Dual core Two cores running 2slots/board	AXD CPB5	AXD CPB6
POTS-POTS / AGW	X call/sec	2.3X call/sec One core used	4.3X call/sec OTP R11_3 beta +patches	0.4X call/sec	2.1X call/sec
ISUP-ISUP /Inter MGW	3.6X call/sec	7.7X call/sec One core used	13X call/sec OTP R11_3 beta +patches	1.55X call/sec	7.6X call/sec
ISUP-ISUP /Intra MGW	5.5X call/sec		26X call/sec	3.17X call/sec	14X call/sec

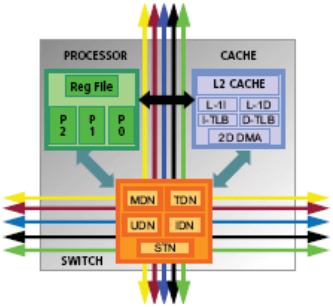
Tilera “Tile64”

Chatty

500 processes created

Each process randomly sends messages and receives a response from all other processes





Erlang

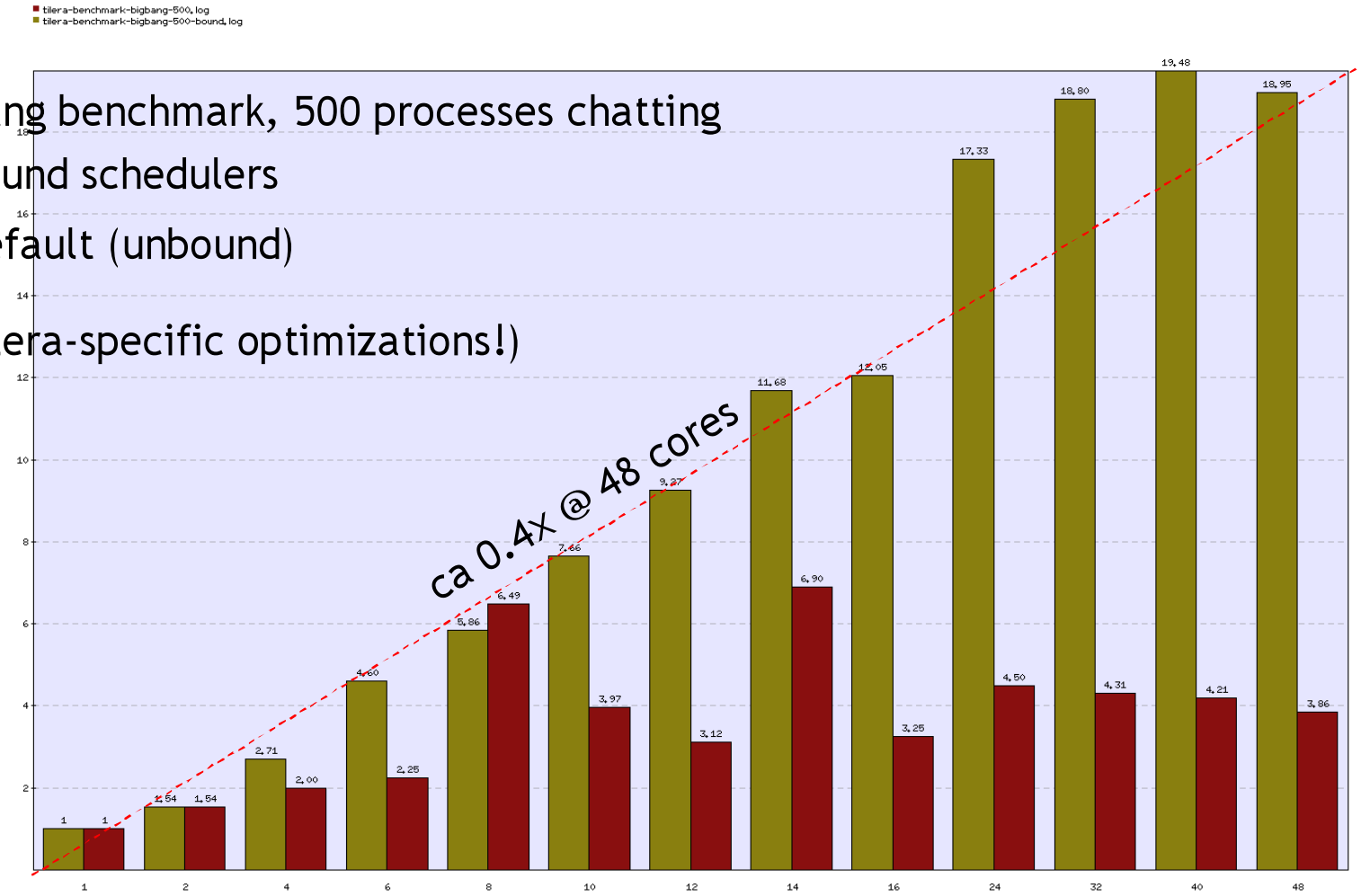
Multicore Benchmark – Big Bang

Big_bang benchmark, 500 processes chatting

Bound schedulers

Default (unbound)

(No Tilera-specific optimizations!)



Erlang/OTP R13B on Tilera Pro 64-core

Erlang Highlights

Declarative

Concurrent

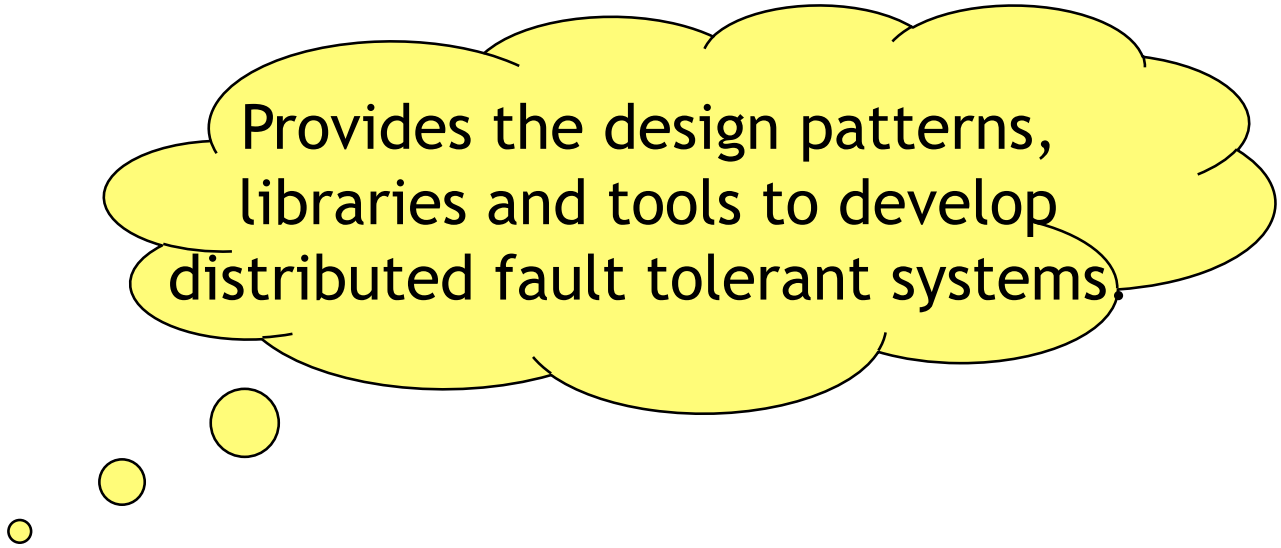
Robust

Distributed

Hot code loading

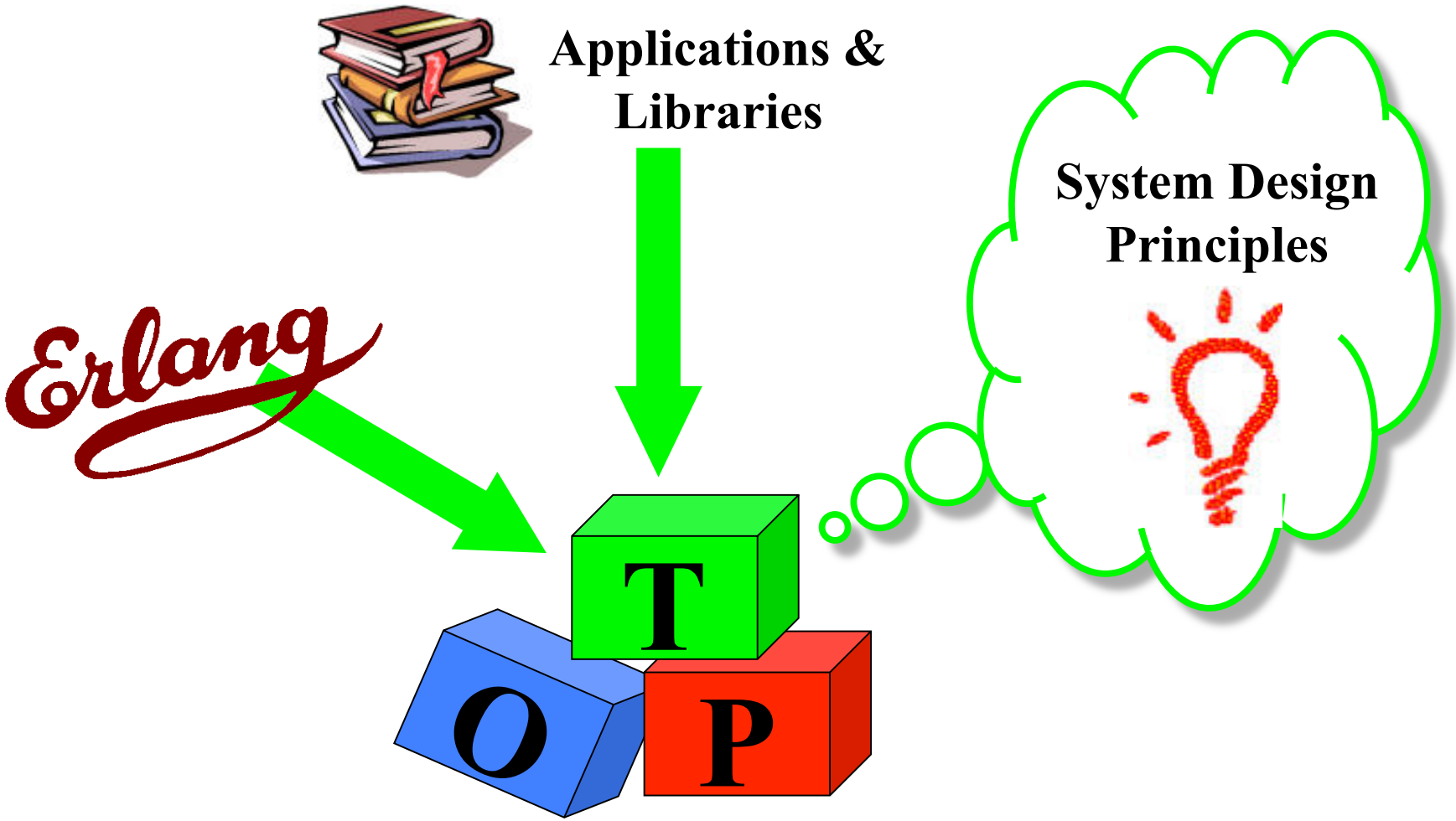
Multicore Support

OTP



Erlang

OTP Middleware



Erlang

Erlang Highlights

Declarative

Concurrent

Robust

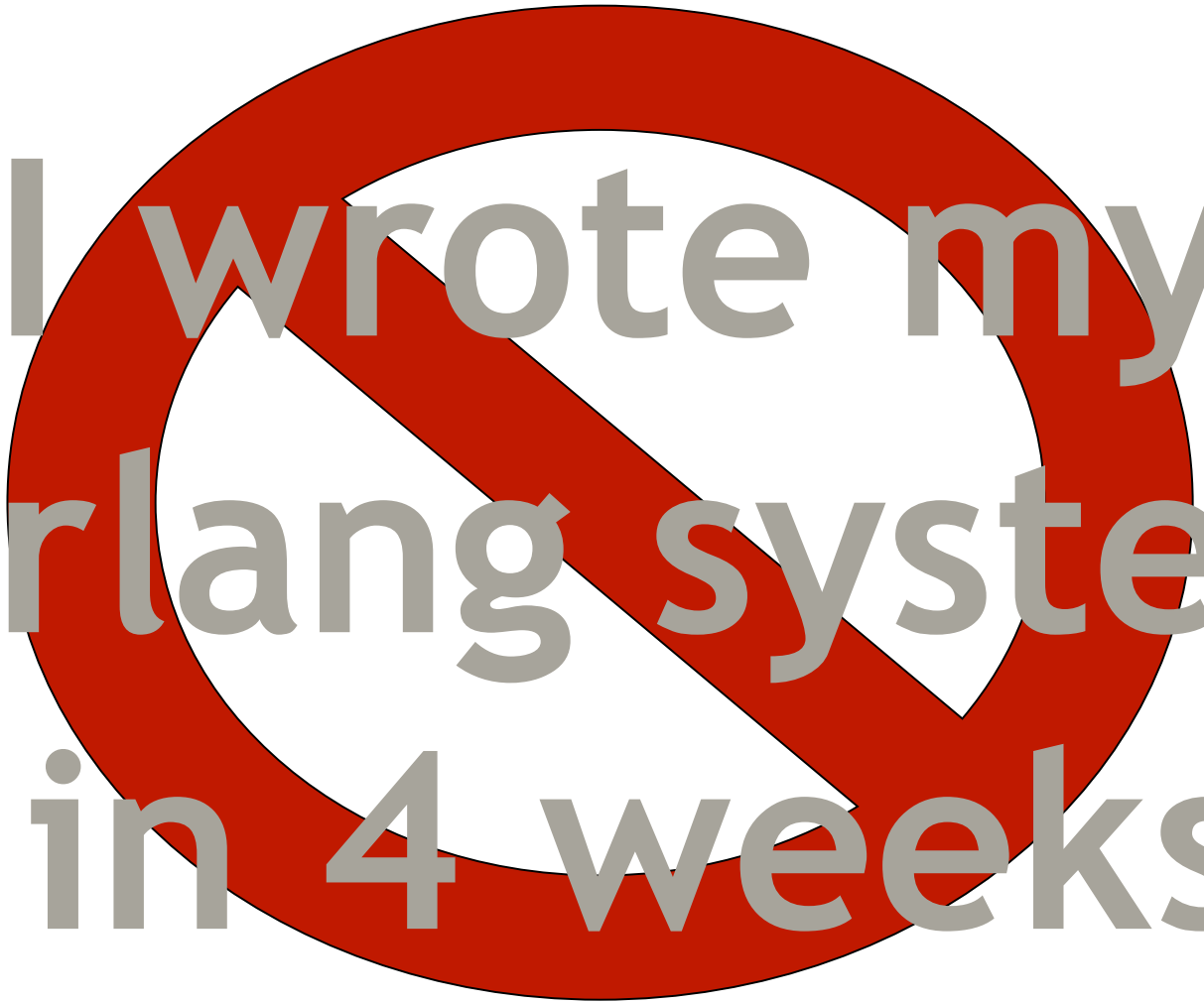
Distributed

Hot code loading

Multicore Support

OTP

Erlang



I wrote my Erlang system in 4 weeks!

Erlang

The Myths of Erlang….

Is it Documented?

Is the developer supporting it?

What visibility does support staff have into what is going on?

- SNMP
- Live Tracing
- Audit Trails
- Statistics
- CLI / HTTP Interface

How much new code was actually written?

Erlang

Erlang

The Myths of Erlang….

Yes, it is easy for

- Simple patches
- Adding functionality without changing the state

Non backwards compatible changes need time time

- Database schema changes
- State changes in your processes
- Upgrades in distributed environments

Test, Test, Test

- A great feature when you have the manpower!

Erlang

Erlang

The Myths of Erlang….

"As a matter of fact, the network performance has been so reliable that **there is almost a risk that our field engineers do not learn maintenance skills**"

Bert Nilsson, Director NGS-Programs Ericsson

Ericsson Contact, Issue 19 2002

Erlang

The Myths of Erlang….

99,999 (Five Nines) is a more like it!

- Achieved at a fraction of the effort of Java & C++

Upgrades are risky!

Non Software related issues

- Power Outages
- Network Failures, Firewall Configurations
- Hardware Faults

Erlang

Who is using Erlang?

Erlang: It's Happening!

CouchDB
Distributed Robust document database

Riak
Distributed, partition tolerant and scalable database

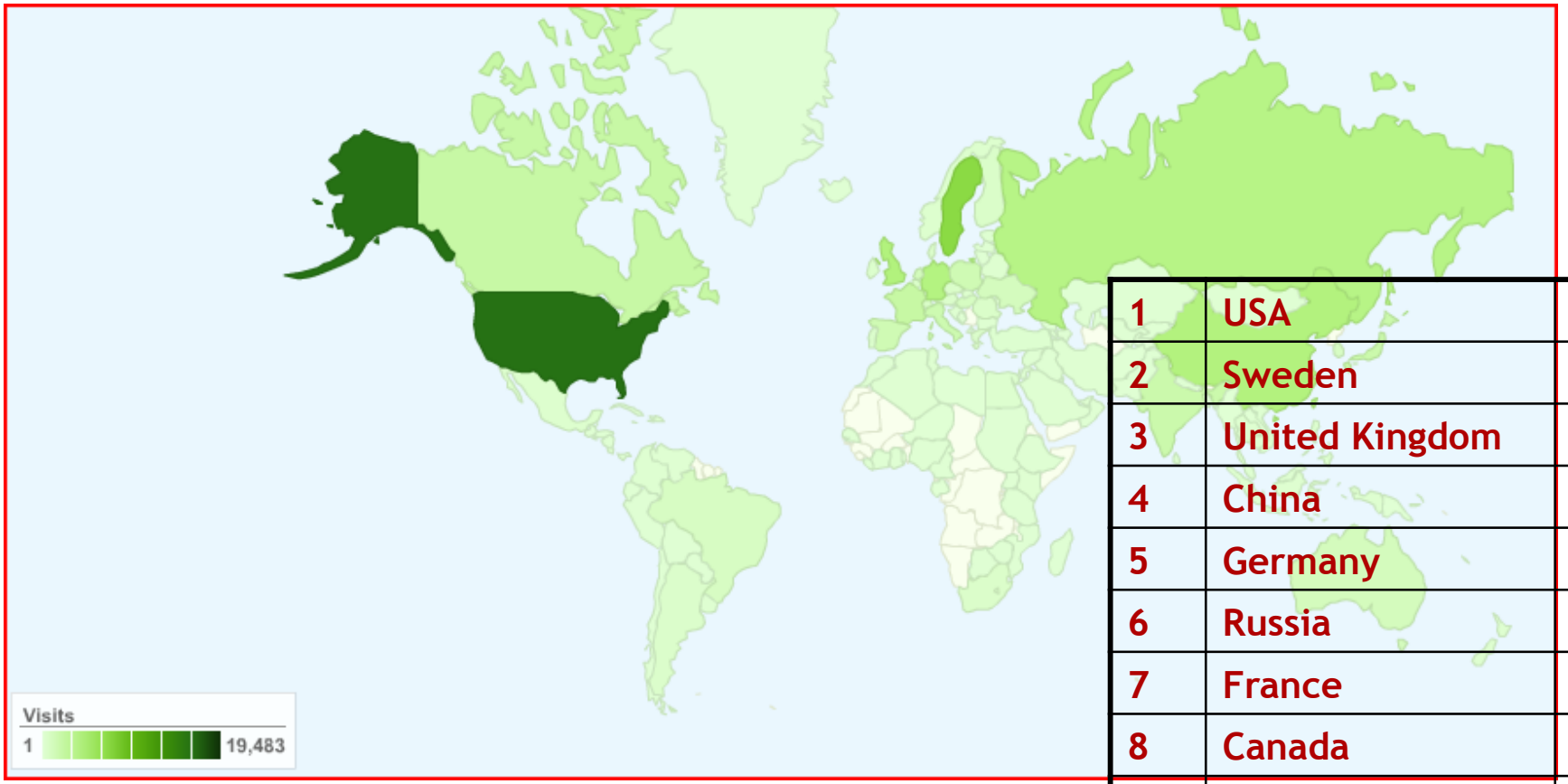
YAWS
Yet Another Web Server

RabbitMQ
High performance enterprise messaging

Ejabberd
XMPP instant messaging server

Erlang

erlang.org site usage (Unique visits, 30 days)



1	USA	19,483
2	Sweden	8,465
3	United Kingdom	4,984
4	China	4,897
5	Germany	4,745
6	Russia	4,465
7	France	2,722
8	Canada	2,647
9	India	2,299
10	Japan	2,261

Erlang

Books

Erlang

More Information

Programming Erlang

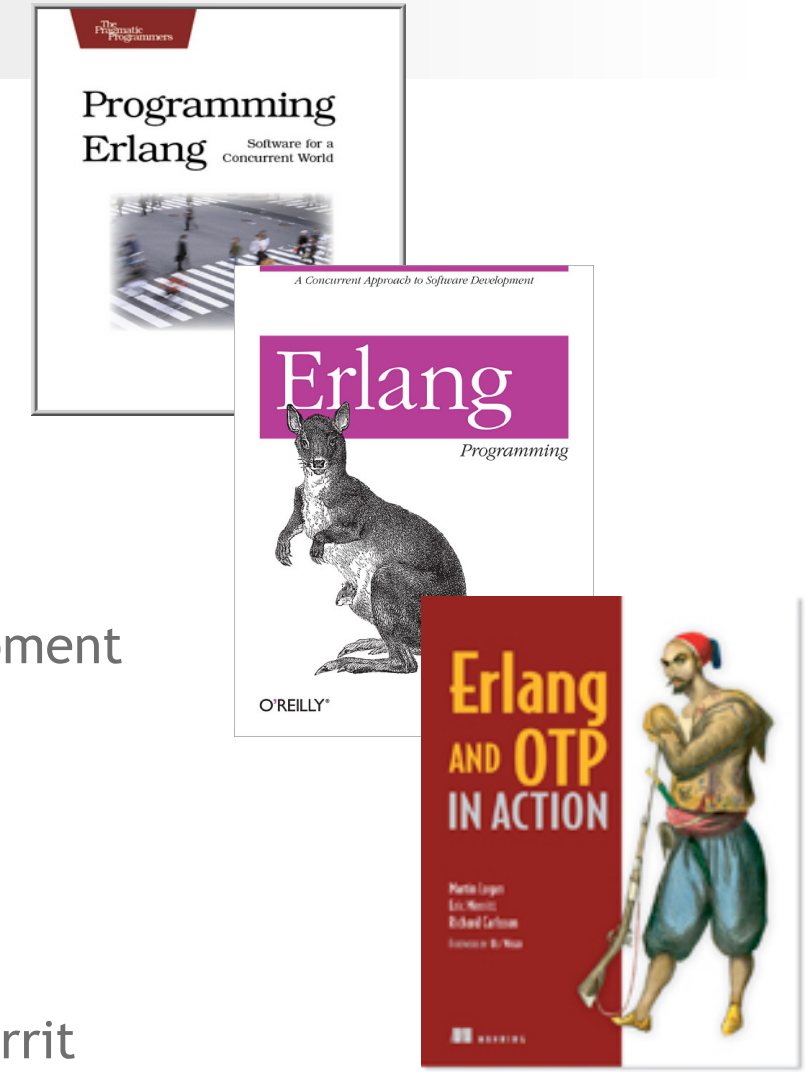
- Software for a Concurrent World
- by Joe Armstrong

Erlang Programming

- A Concurrent Approach to Software Development
- by Francesco Cesarini & Simon Thompson

Erlang and OTP in Action

- Large-scale software design with OTP
- by Richard Carlsson, Martin Logan & Eric Merrit



Erlang

Questions

Erlang

Thank You!

@FrancescoC on Twitter or
francesco@erlang-solutions.com

Erlang