goto; conference aarhus

ERLANG EVOLVES

FOR MULTI-CORE AND CLOUD ENVIRONMENTS

Torben Hoffmann

Erlang Solutions Ltd.

@LeHoff

http://musings-of-an-erlang-priest.blogspot.dk/

INTERNATIONAL SOFTWARE DEVELOPMENT CONFERENCE

gotocon.com

Agenda

- Erlang fundamentals

- Challenges

© 1999-2012 Erlang Solutions Ltd.

2

Warning 1: The Truth

© 1999-2012 Erlang Solutions Ltd.

I will do a few simplifications in order to get the main points across.

Warning 1: The Truth

© 1999-2012 Erlang Solutions Ltd.

3

I will do a few simplifications in order to get the main points across.

Warning 1: The Truth

Will you tell the truth?

© 1999-2012 Erlang Solutions Ltd.

I will do a few simplifications in order to get the main points across.

Warning 1: The Truth

Will you tell the truth? Yes

Erlang SOLUTIONS
© 1999-2012 Erlang Solutions Ltd.

I will do a few simplifications in order to get the main points across.

Warning 1: The Truth

Will you tell the truth? Yes

The whole truth?

© 1999-2012 Erlang Solutions Ltd.

3

I will do a few simplifications in order to get the main points across.

Warning 1: The Truth

Will you tell the truth?	Yes
The whole truth?	No

© 1999-2012 Erlang Solutions Ltd.

I will do a few simplifications in order to get the main points across.

Warning 1: The Truth

Will you tell the truth?	Yes
The whole truth?	No
So help you OTP?	

© 1999-2012 Erlang Solutions Ltd.

I will do a few simplifications in order to get the main points across.

Warning 1: The Truth

Will you tell the truth?	Yes
The whole truth?	No
So help you OTP?	Yes

Erlang SOLUTIONS

© 1999-2012 Erlang Solutions Ltd.

I will do a few simplifications in order to get the main points across.

Warning 2: Serious Love Ahead

I love Erlang!

© 1999-2012 Erlang Solutions Ltd.

4

It was part of a major career shift and I have never looked back.
Apologies if I get too intense.

Realities Of Software Development

- Time-to-market pressure
- Utilisation of computing resources
- Scaling successes
- Maintenance burden

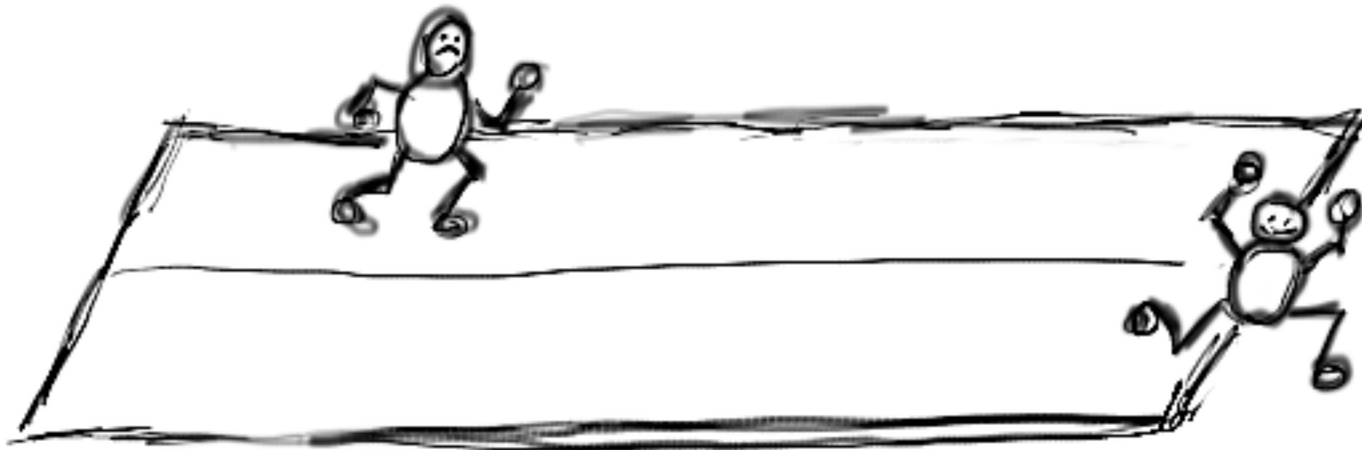
Erlang SOLUTIONS
© 1999-2012 Erlang Solutions Ltd.

What Could Be...

© 1999-2012 Erlang Solutions Ltd.

3x productivity over C++/Java
Seamless scaling on multicore
Scaling nicely over machines
Less code per feature
The future is here today – it's called Erlang!

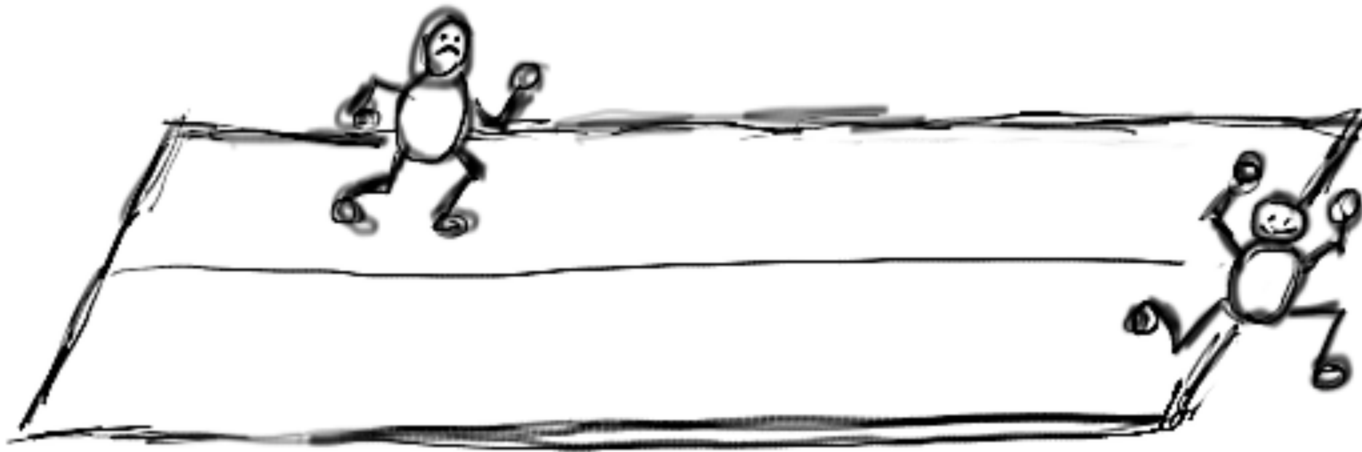
What Could Be...



© 1999-2012 Erlang Solutions Ltd.

3x productivity over C++/Java
Seamless scaling on multicore
Scaling nicely over machines
Less code per feature
The future is here today – it's called Erlang!

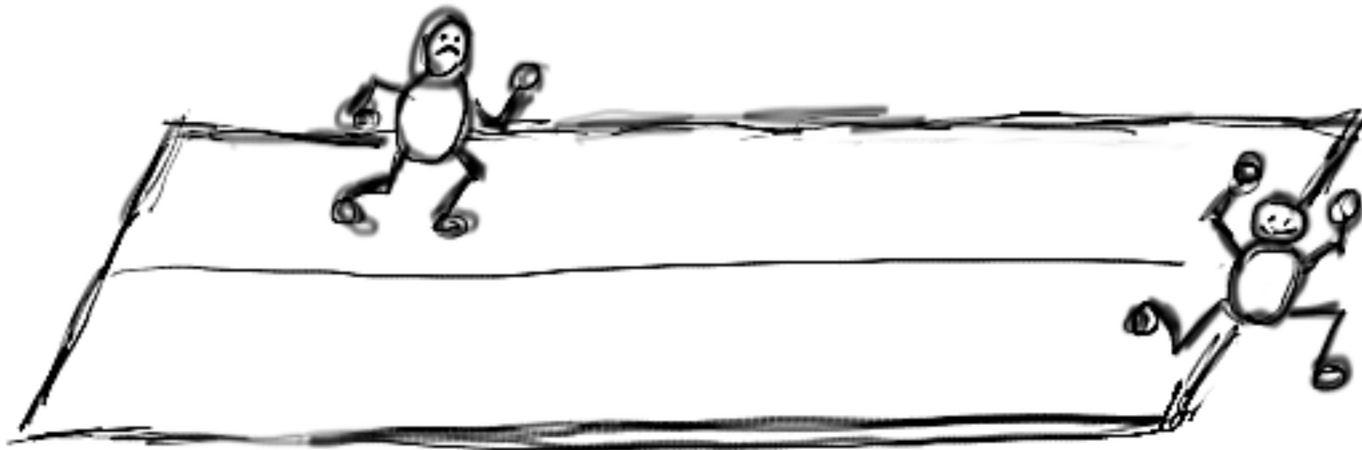
What Could Be...

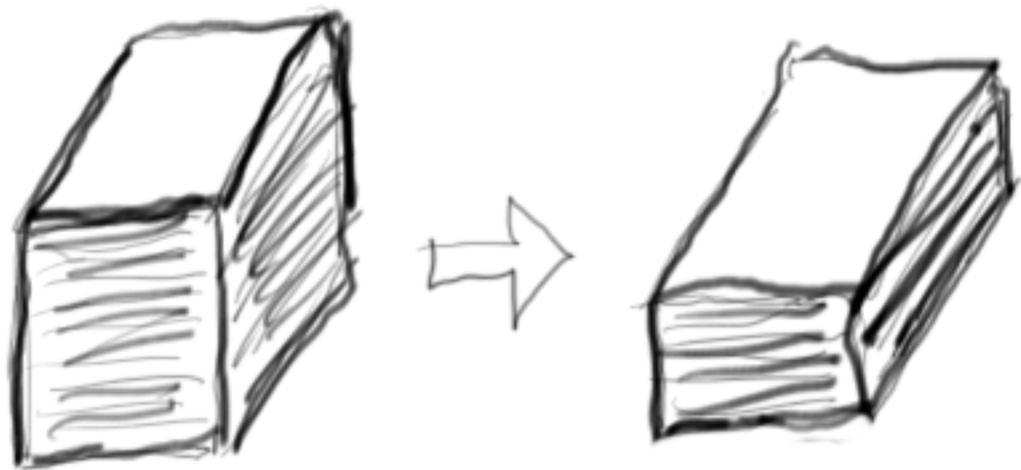


© 1999-2012 Erlang Solutions Ltd.

3x productivity over C++/Java
Seamless scaling on multicore
Scaling nicely over machines
Less code per feature
The future is here today – it's called Erlang!

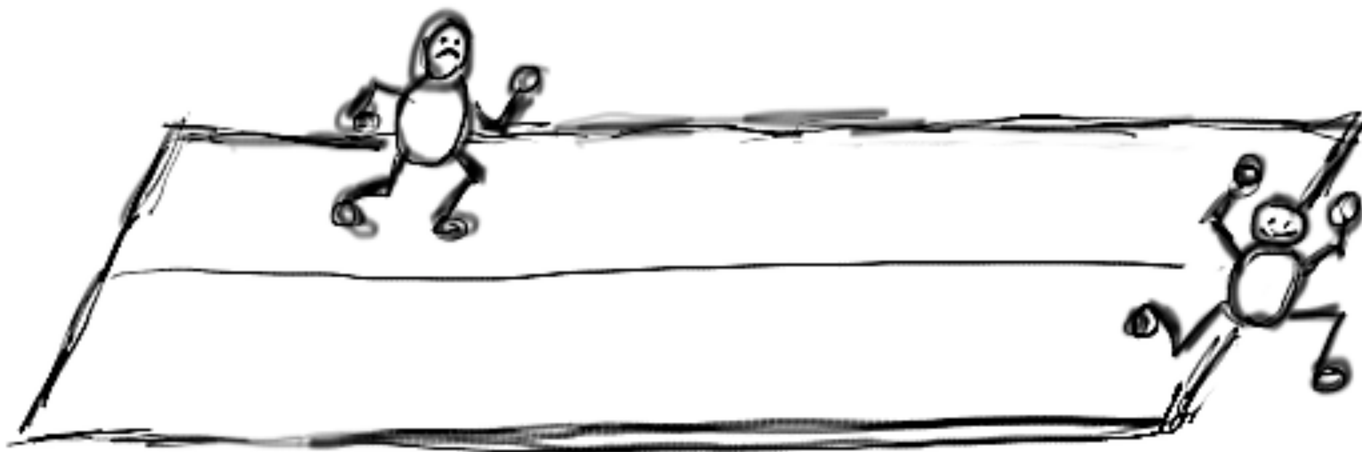
What Could Be...

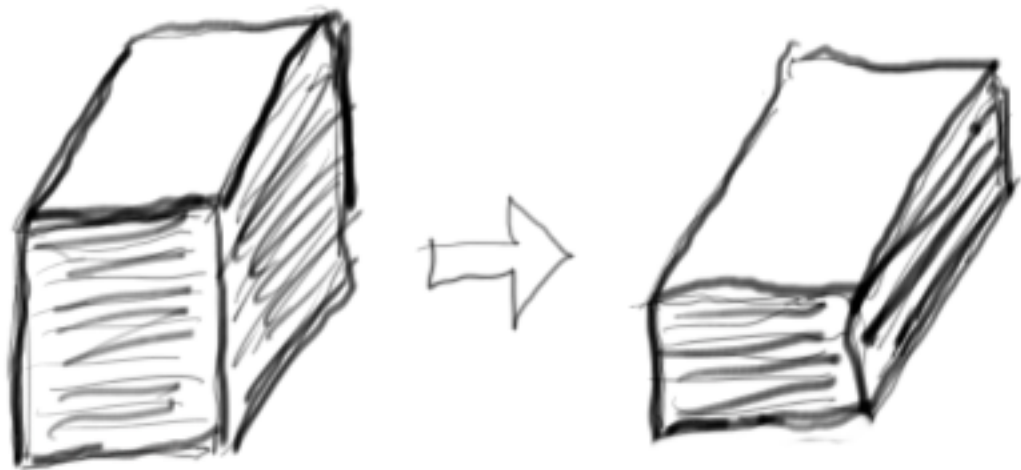




3x productivity over C++/Java
Seamless scaling on multicore
Scaling nicely over machines
Less code per feature
The future is here today – it's called Erlang!

What Could Be...

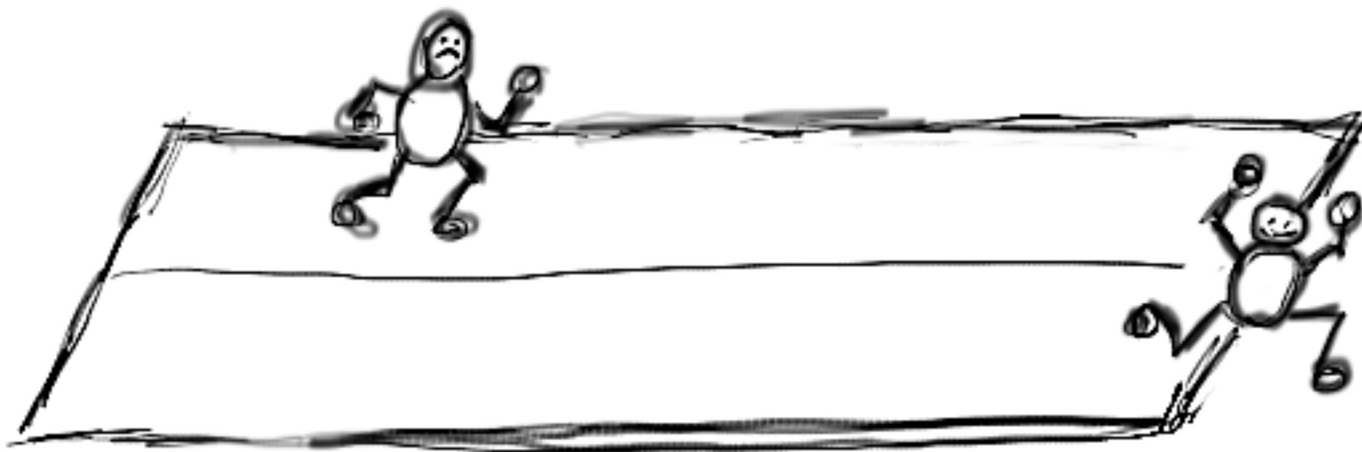




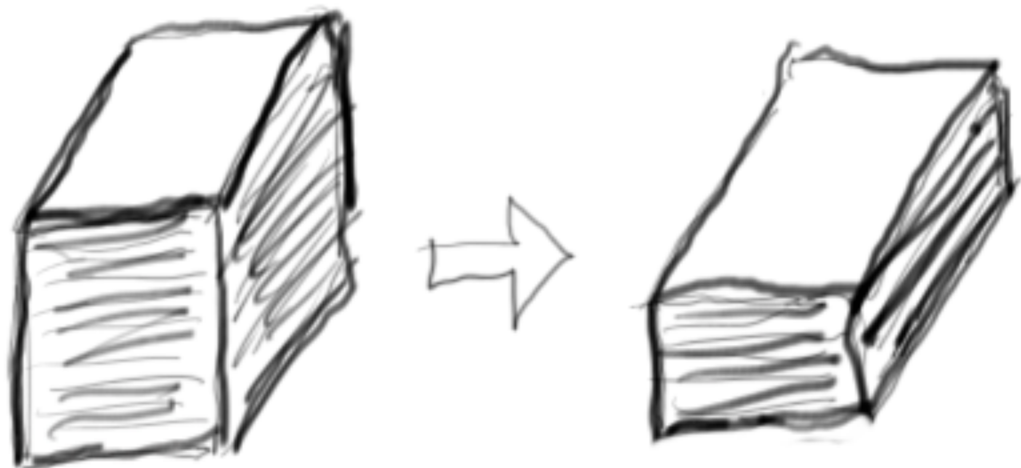
Erlang Solutions Ltd.

3x productivity over C++/Java
Seamless scaling on multicore
Scaling nicely over machines
Less code per feature
The future is here today - it's called Erlang!

What Could Be...

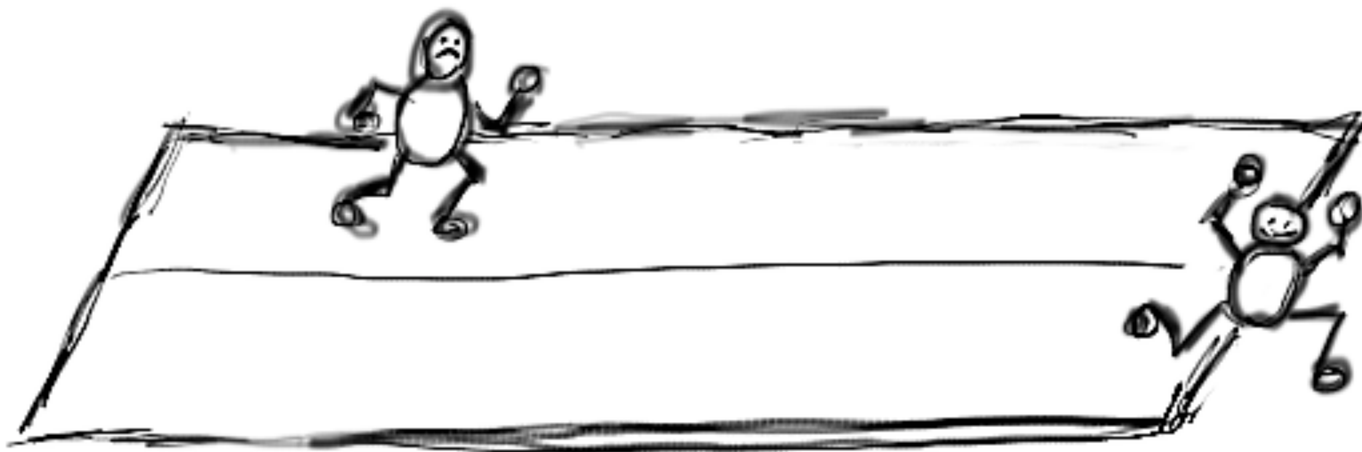


The future is here...



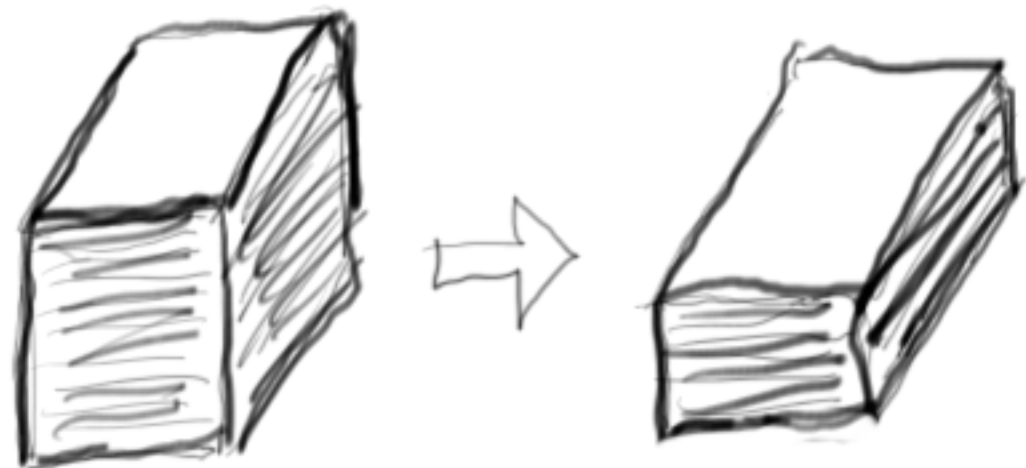
3x productivity over C++/Java
Seamless scaling on multicore
Scaling nicely over machines
Less code per feature
The future is here today - it’s called Erlang!

What Could Be...



The future is here...

The future is Erlang!



Erlang Solutions Ltd.

3x productivity over C++/Java
Seamless scaling on multicore
Scaling nicely over machines
Less code per feature
The future is here today – it’s called Erlang!

Erlang's Original Requirements

© 1999-2012 Erlang Solutions Ltd.

Erlang's Original Requirements

- Large scale concurrency

© 1999-2012 Erlang Solutions Ltd.

7

Erlang's Original Requirements

- Large scale concurrency
- Soft real-time

Erlang SOLUTIONS

© 1999-2012 Erlang Solutions Ltd.

7

Erlang's Original Requirements

- Large scale concurrency
- Soft real-time
- Distributed systems

Erlang SOLUTIONS

© 1999-2012 Erlang Solutions Ltd.

7

Erlang's Original Requirements

- Large scale concurrency
- Soft real-time
- Distributed systems
- Hardware interaction

Erlang SOLUTIONS

© 1999-2012 Erlang Solutions Ltd.

Erlang's Original Requirements

- Large scale concurrency
- Soft real-time
- Distributed systems
- Hardware interaction
- Very large software systems

Erlang SOLUTIONS
© 1999-2012 Erlang Solutions Ltd.

Erlang's Original Requirements

- Large scale concurrency
- Soft real-time
- Distributed systems
- Hardware interaction
- Very large software systems
- Complex functionality

Erlang SOLUTIONS
© 1999-2012 Erlang Solutions Ltd.

Erlang's Original Requirements

- Large scale concurrency
- Soft real-time
- Distributed systems
- Hardware interaction
- Very large software systems
- Complex functionality
- Continuous operation for many years

Erlang SOLUTIONS

© 1999-2012 Erlang Solutions Ltd.

Erlang's Original Requirements

- Large scale concurrency
- Soft real-time
- Distributed systems
- Hardware interaction
- Very large software systems
- Complex functionality
- Continuous operation for many years
- Software maintenance on-the-fly

Erlang SOLUTIONS

© 1999-2012 Erlang Solutions Ltd.

Erlang's Original Requirements

- Large scale concurrency
- Soft real-time
- Distributed systems
- Hardware interaction
- Very large software systems
- Complex functionality
- Continuous operation for many years
- Software maintenance on-the-fly
- High quality and reliability

Erlang SOLUTIONS

© 1999-2012 Erlang Solutions Ltd.

Erlang's Original Requirements

- Large scale concurrency
- Soft real-time
- Distributed systems
- Hardware interaction
- Very large software systems
- Complex functionality
- Continuous operation for many years
- Software maintenance on-the-fly
- High quality and reliability
- Fault tolerance

Erlang SOLUTIONS

© 1999-2012 Erlang Solutions Ltd.

Erlang's Original Requirements

- Large scale concurrency
- Soft real-time
- Distributed systems
- Hardware interaction
- Very large software systems
- Complex functionality
- Continuous operation for many years
- Software maintenance on-the-fly
- High quality and reliability
- Fault tolerance

Sounds familiar?

Erlang SOLUTIONS

© 1999-2012 Erlang Solutions Ltd.

Erlang's Original Requirements

- Large scale concurrency
- Soft real-time
- Distributed systems
- Hardware interaction
- Very large software systems
- Complex functionality
- Continuous operation for many years
- Software maintenance on-the-fly
- High quality and reliability
- Fault tolerance

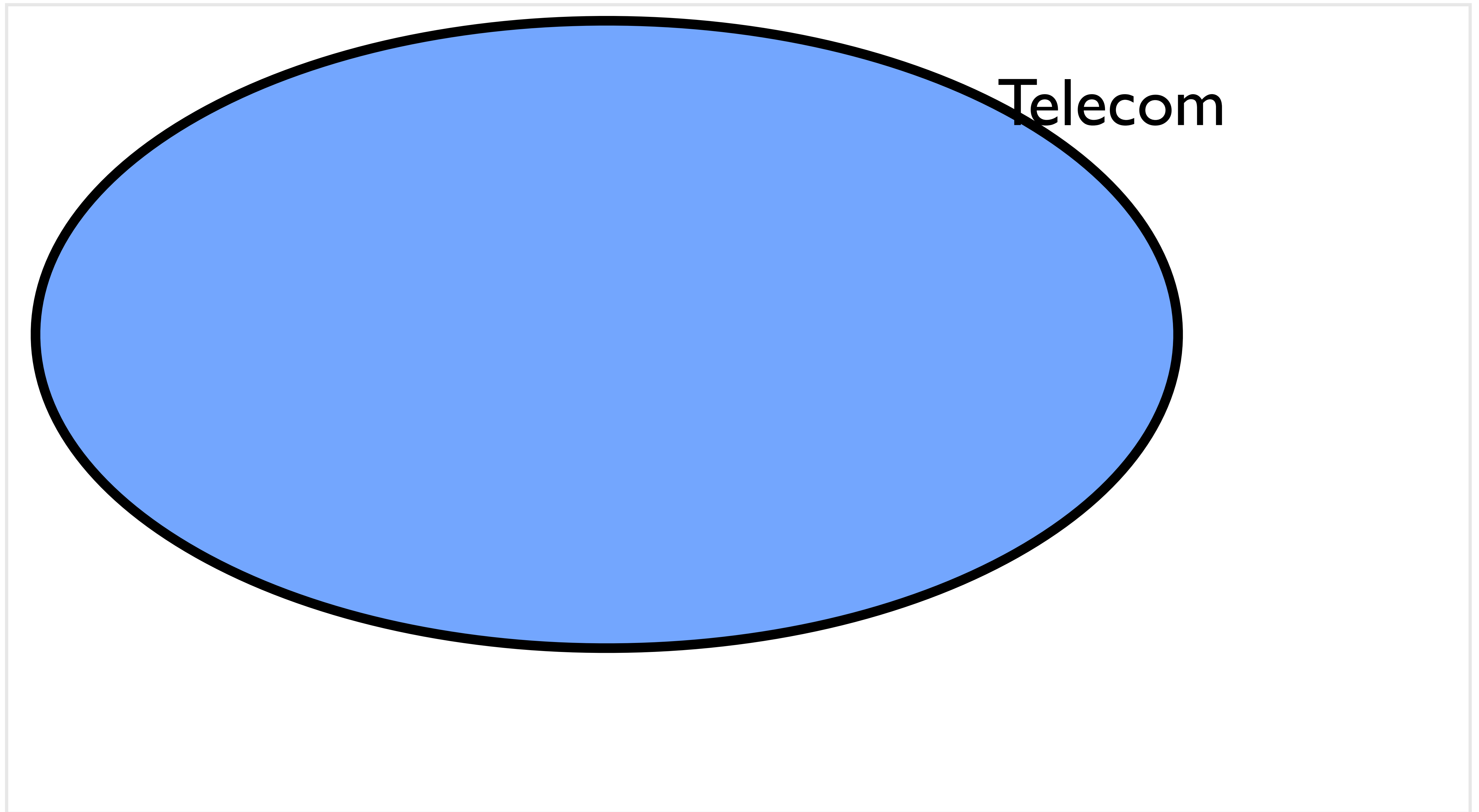
Sounds familiar?

Sounds good, right?

Erlang SOLUTIONS

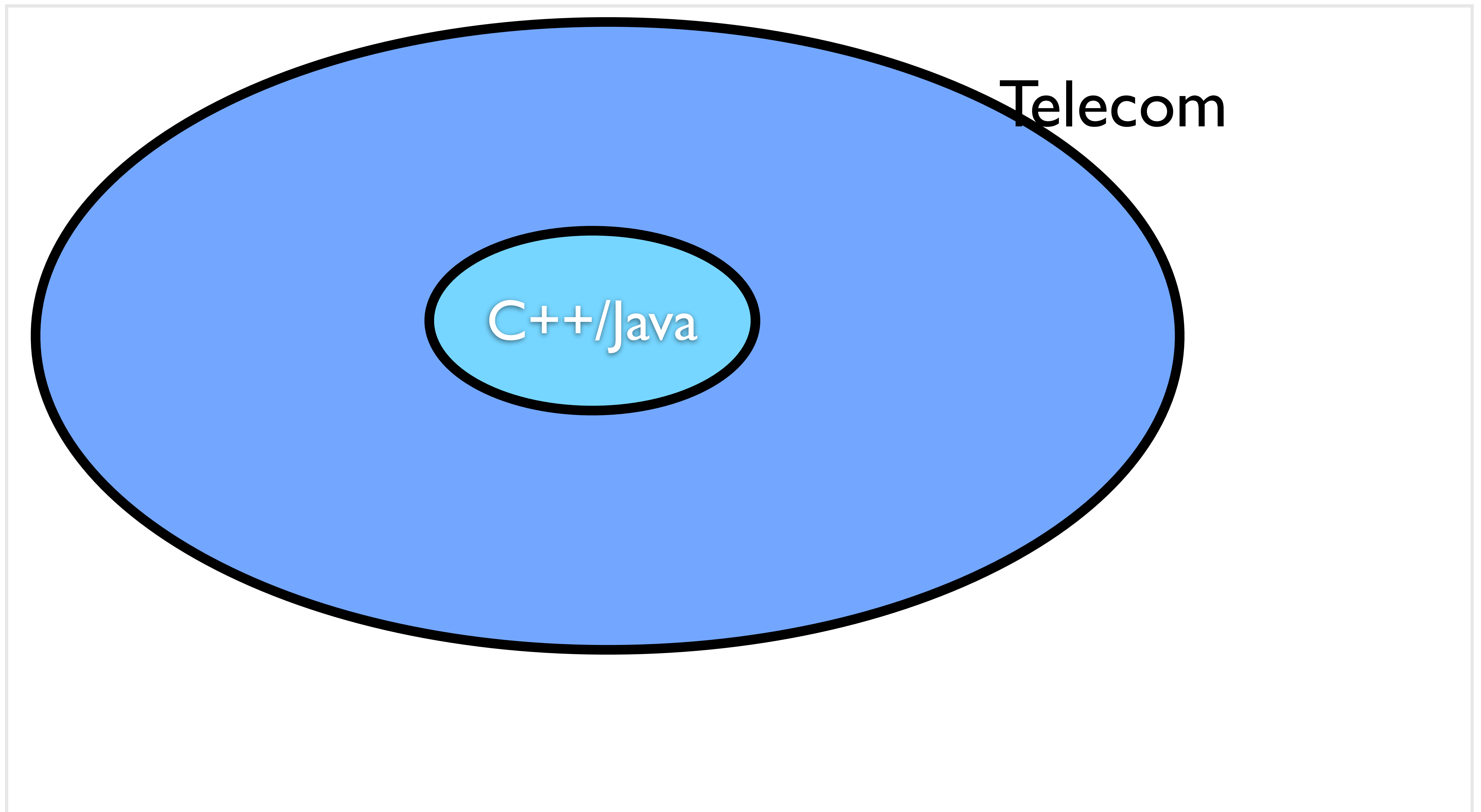
© 1999-2012 Erlang Solutions Ltd.

General vs Domain Specific



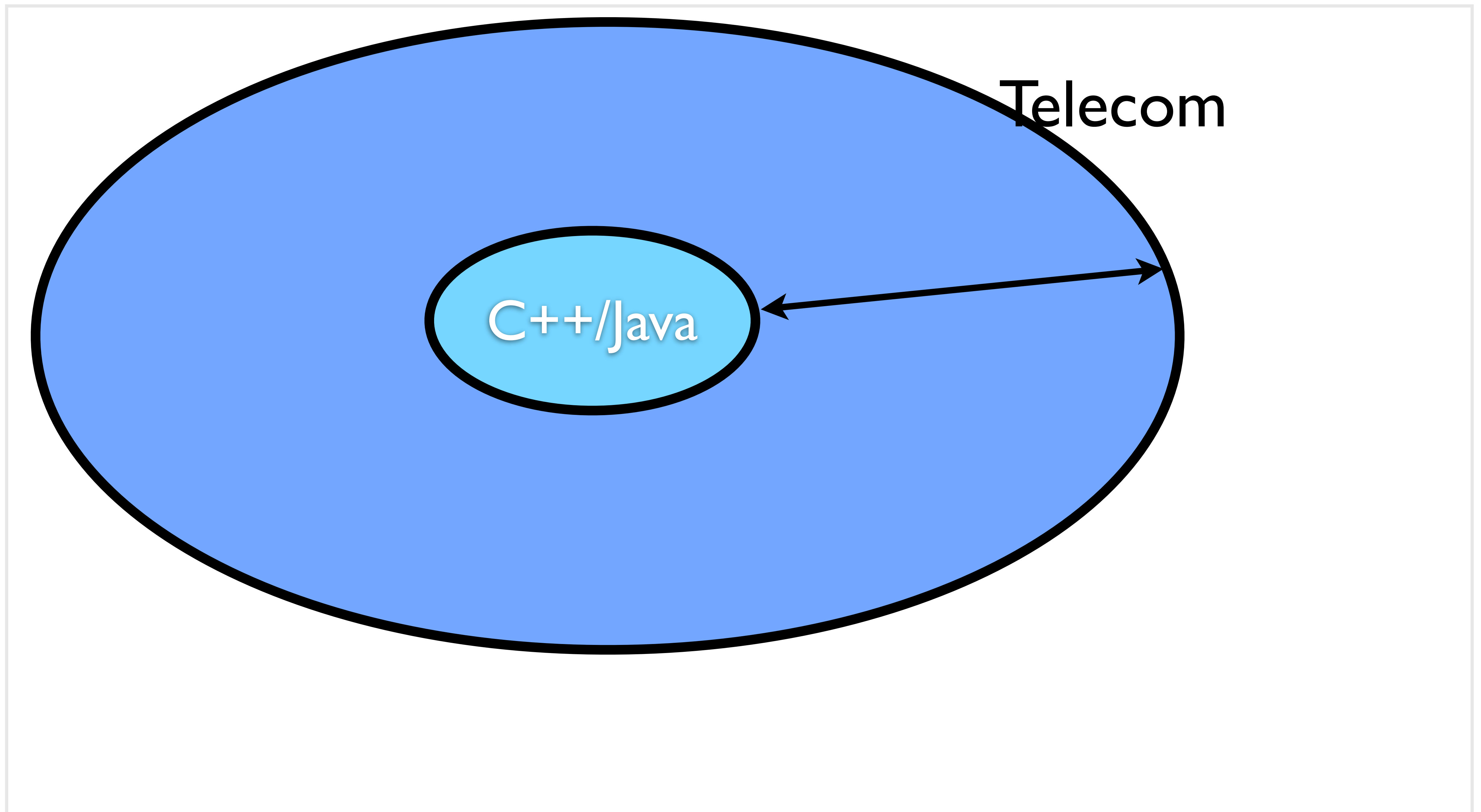
© 1999-2012 Erlang Solutions Ltd.

General vs Domain Specific



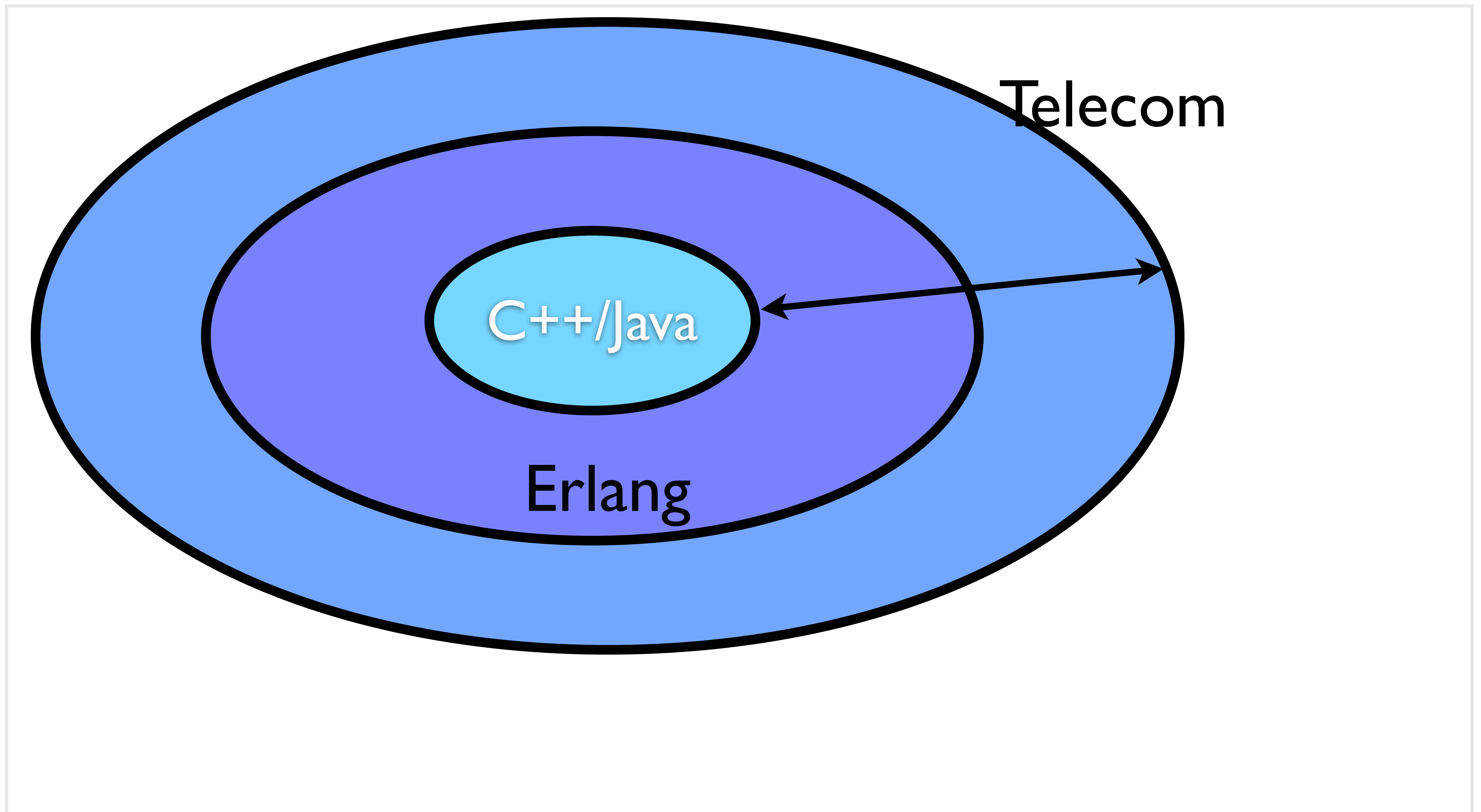
© 1999-2012 Erlang Solutions Ltd.

General vs Domain Specific



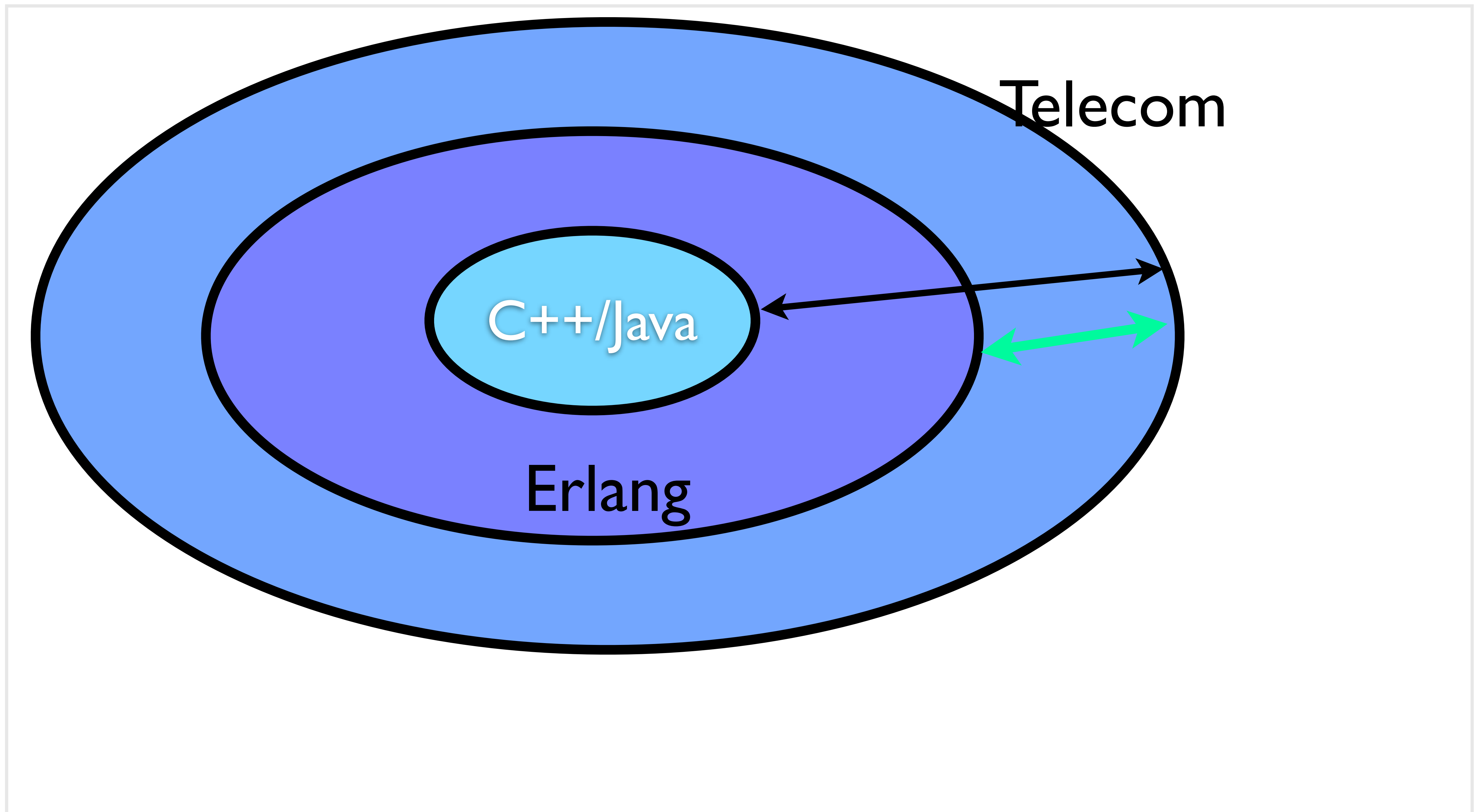
© 1999-2012 Erlang Solutions Ltd.

General vs Domain Specific



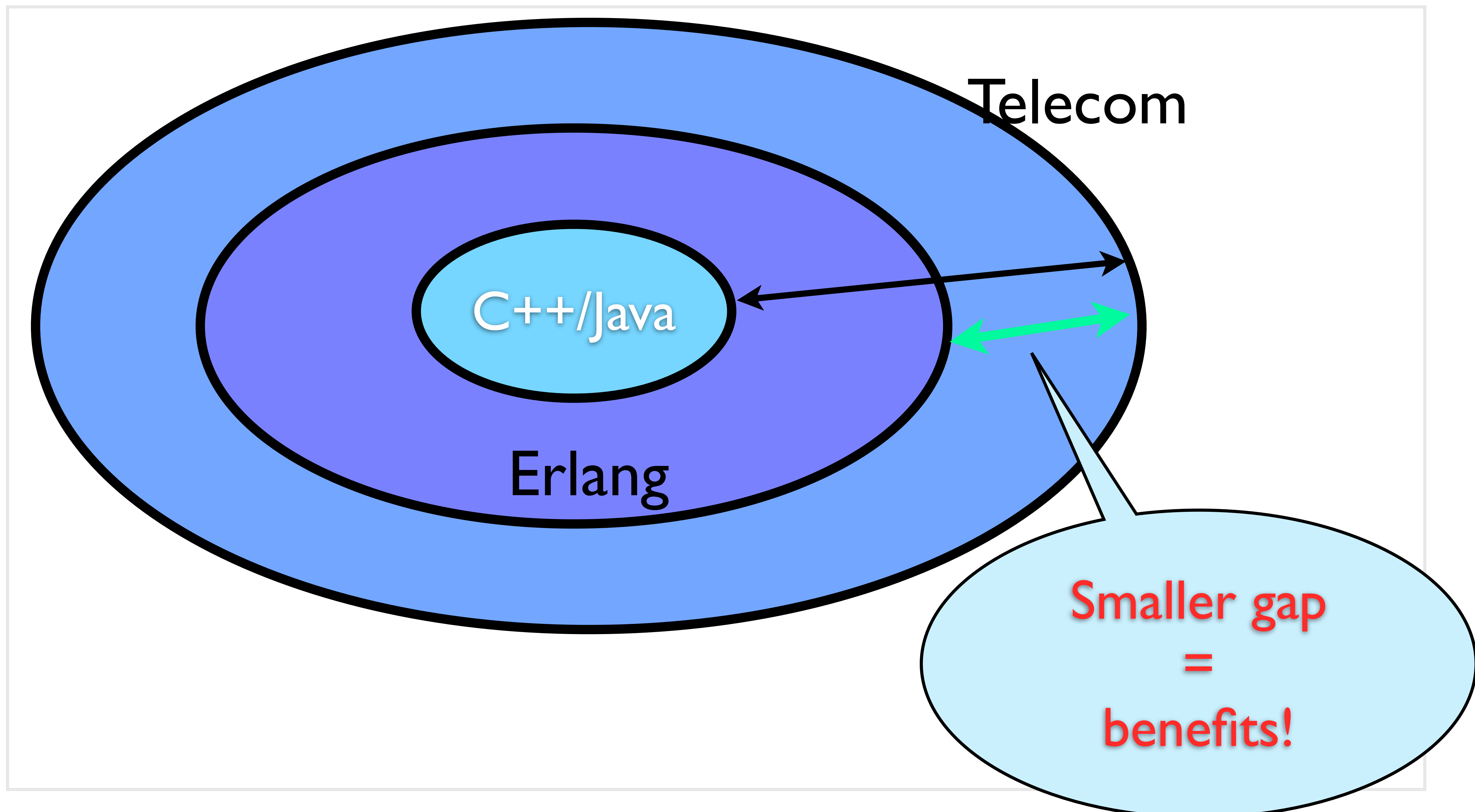
© 1999-2012 Erlang Solutions Ltd.

General vs Domain Specific



© 1999-2012 Erlang Solutions Ltd.

General vs Domain Specific



© 1999-2012 Erlang Solutions Ltd.

Erlang's Sweet Spot

GUI

Erlang

Middleware
Coordination
Control

Drivers

© 1999-2012 Erlang Solutions Ltd.

Erlang was intended to deal with the control plane in telecom, which is all about orchestration of what goes on.
GUI and low-level things are not what Erlang was created for - hence Erlang has good support for integration with other languages.
Read the wonderful doctor thesis by Bjarne Däcker if you want to learn more: http://

Other Erlang Domains

- Messaging - XMPP et al
  - ejabberd, MongooseIM
- Webservers
  - Yaws, Chicago Boss
- Payment switches & soft switches
  - Vocalink, OpenFlow/LINC
- Distributed Databases
  - Riak, CouchDB, Scalaris

Erlang SOLUTIONS

© 1999-2012 Erlang Solutions Ltd.

Other Erlang Domains

- Messaging - XMPP et al
  - ejabberd, MongooseIM
- Webservers
  - Yaws, Chicago Boss
- Payment switches & soft switches
  - Vocalink, OpenFlow/LINC
- Distributed Databases
  - Riak, CouchDB, Scalaris

If the tool fits,
you must select!
Tech Mesh Conference
4-5 December
London

Erlang SOLUTIONS

© 1999-2012 Erlang Solutions Ltd.

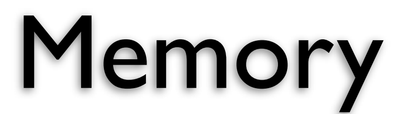
To Share Or Not To Share

© 1999-2012 Erlang Solutions Ltd.

11

Death propagates in shared memory unless you do a ton of defensive programming.
Due to the actor model with no shared memory it is custom in Erlang to do fail-fast programming.
No shared memory allows you to fail fast when suitable.
Erlang uses message passing between processes to exchange information.

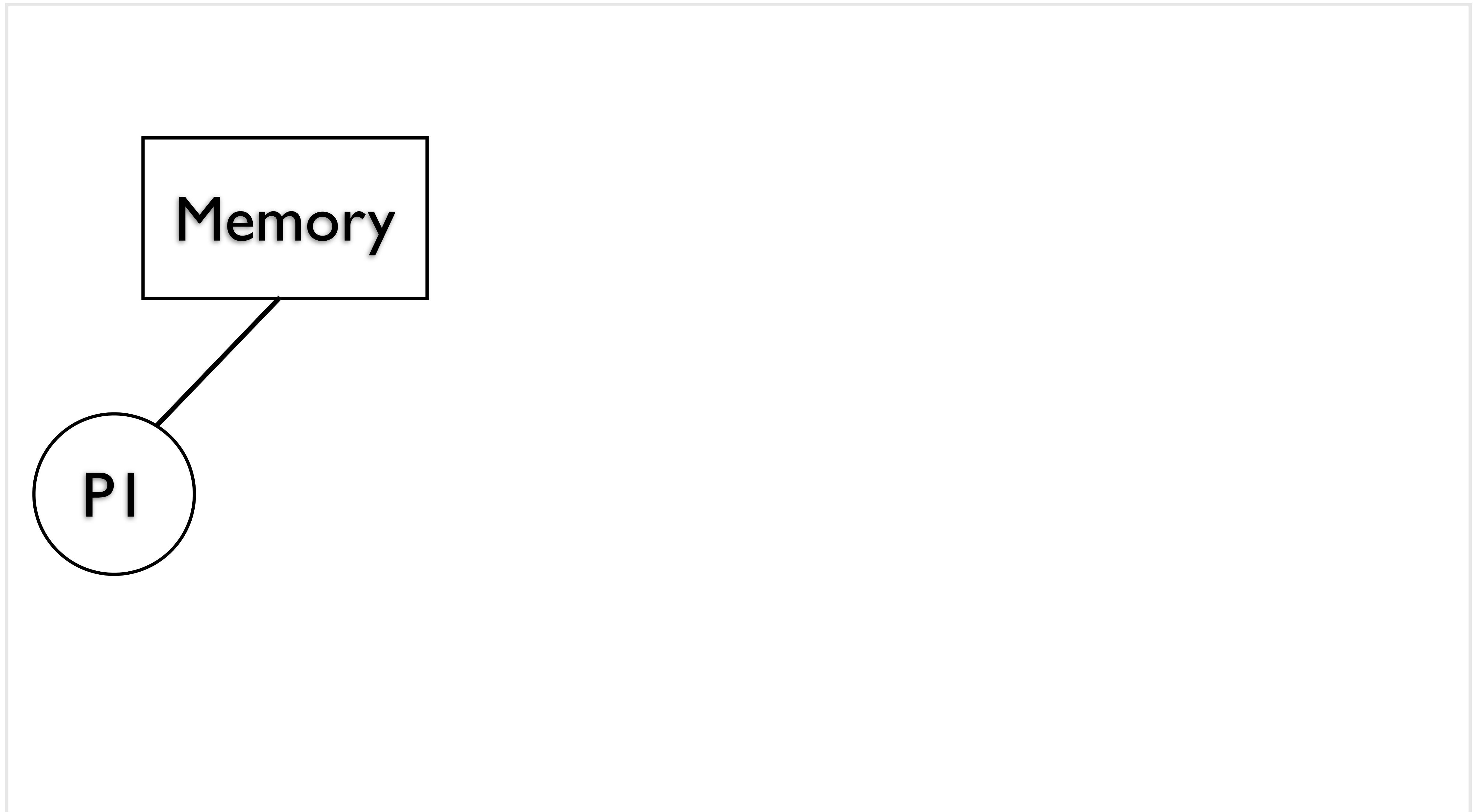
To Share Or Not To Share



Erlang SOLUTIONS
© 1999-2012 Erlang Solutions Ltd.

Death propagates in shared memory unless you do a ton of defensive programming.
Due to the actor model with no shared memory it is custom in Erlang to do fail-fast programming.
No shared memory allows you to fail fast when suitable.
Erlang uses message passing between processes to exchange information.

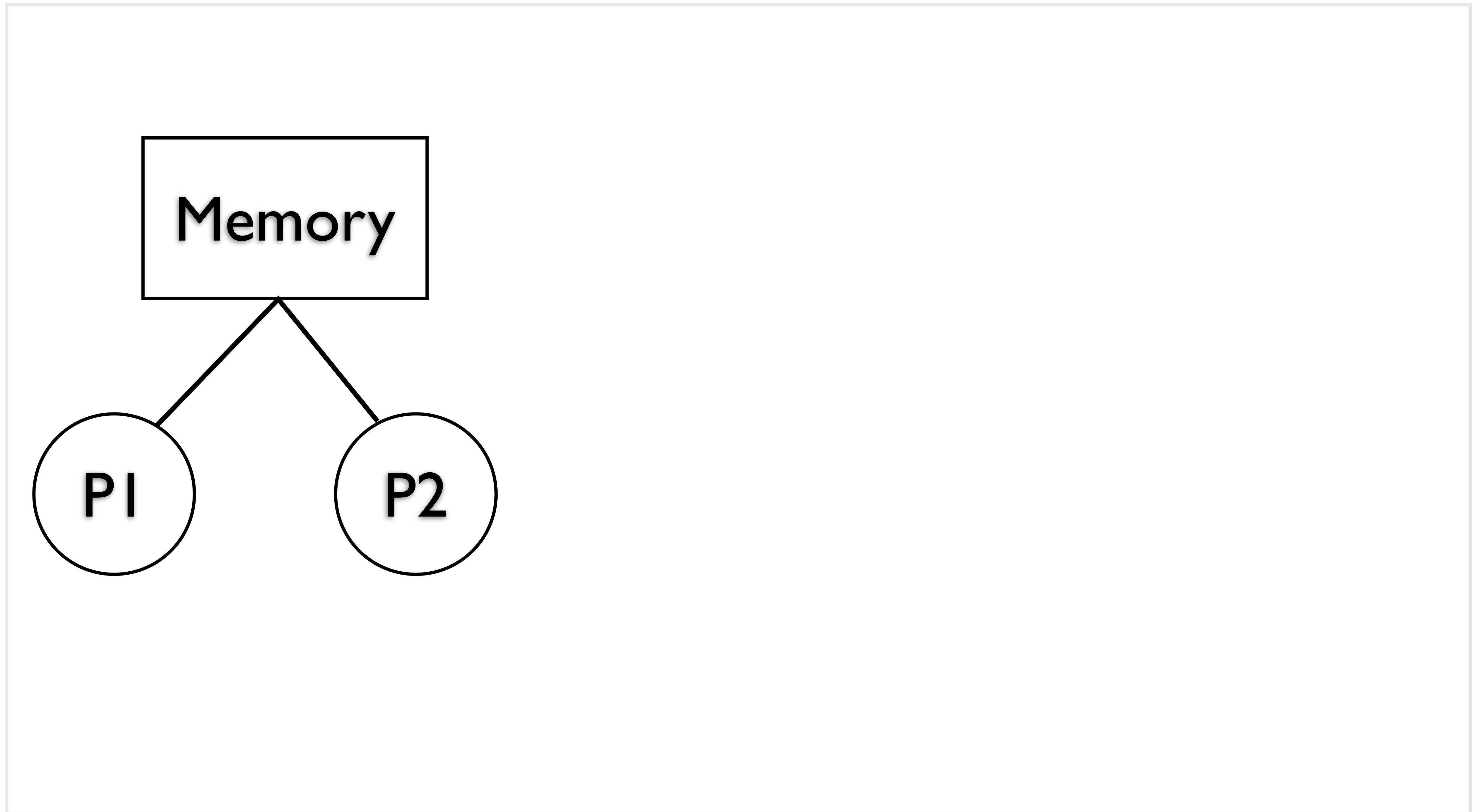
To Share Or Not To Share



© 1999-2012 Erlang Solutions Ltd.

Death propagates in shared memory unless you do a ton of defensive programming.
Due to the actor model with no shared memory it is custom in Erlang to do fail-fast programming.
No shared memory allows you to fail fast when suitable.
Erlang uses message passing between processes to exchange information.

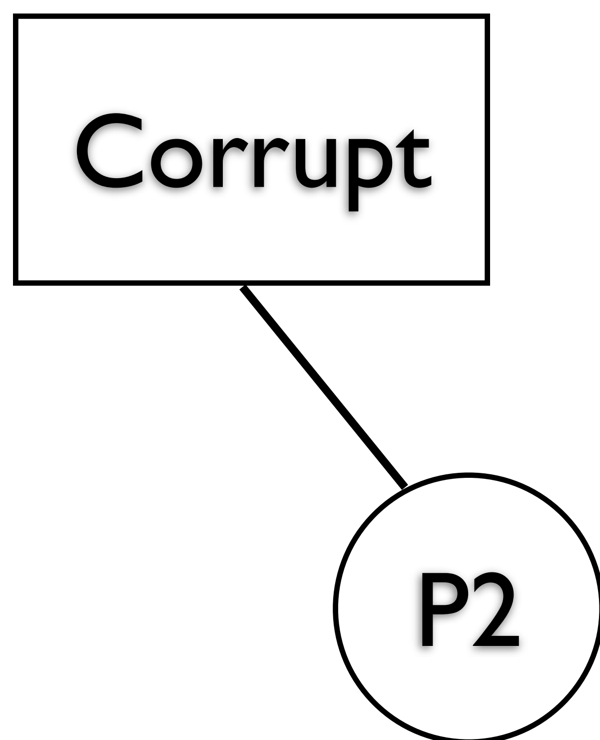
To Share Or Not To Share



Erlang SOLUTIONS
© 1999-2012 Erlang Solutions Ltd.

Death propagates in shared memory unless you do a ton of defensive programming.
Due to the actor model with no shared memory it is custom in Erlang to do fail-fast programming.
No shared memory allows you to fail fast when suitable.
Erlang uses message passing between processes to exchange information.

To Share Or Not To Share



© 1999-2012 Erlang Solutions Ltd.

Death propagates in shared memory unless you do a ton of defensive programming.
Due to the actor model with no shared memory it is custom in Erlang to do fail-fast programming.
No shared memory allows you to fail fast when suitable.
Erlang uses message passing between processes to exchange information.

To Share Or Not To Share

Corrupt

Erlang SOLUTIONS

© 1999-2012 Erlang Solutions Ltd.

Death propagates in shared memory unless you do a ton of defensive programming.
Due to the actor model with no shared memory it is custom in Erlang to do fail-fast programming.
No shared memory allows you to fail fast when suitable.
Erlang uses message passing between processes to exchange information.

To Share Or Not To Share

Corrupt

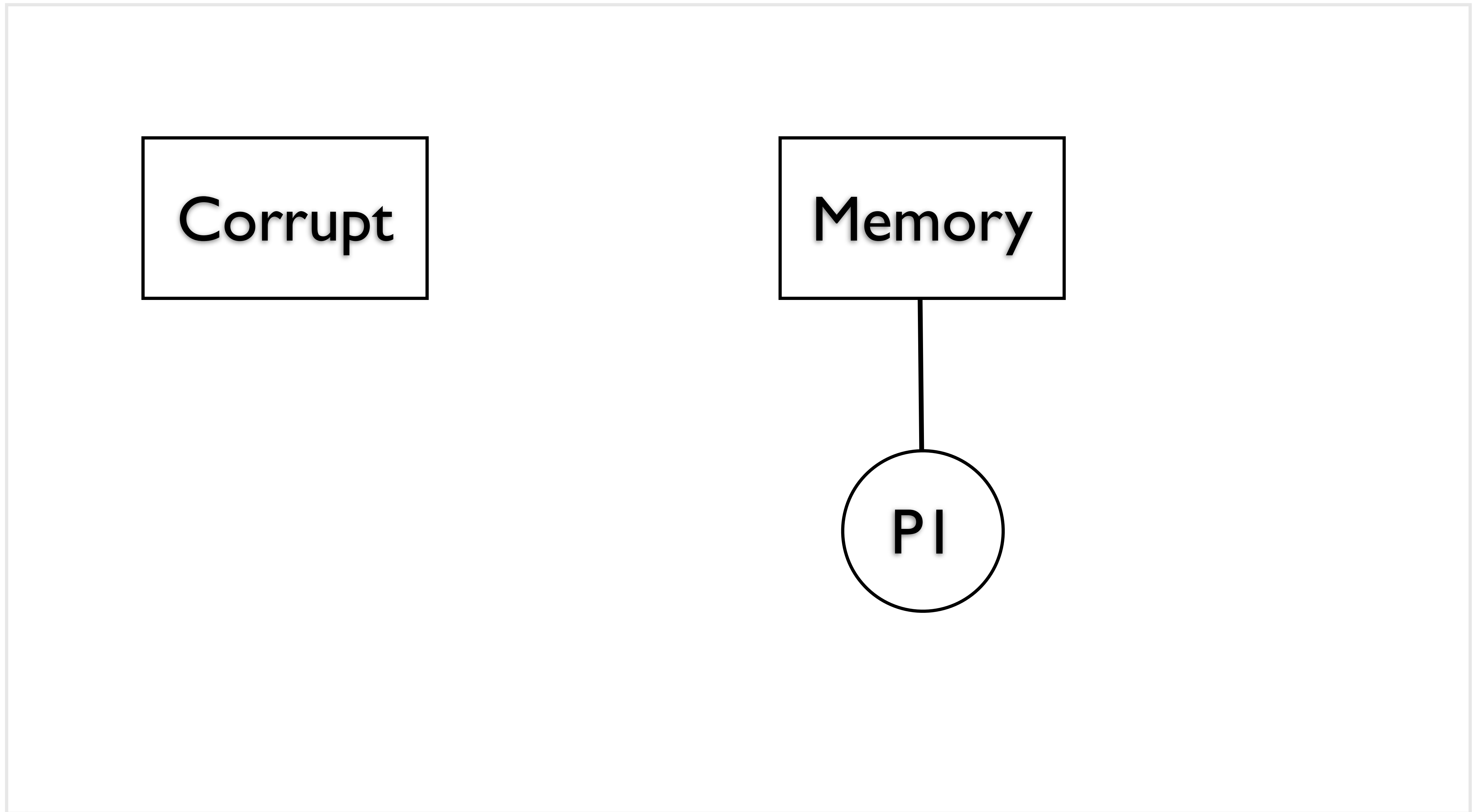
Memory

© 1999-2012 Erlang Solutions Ltd.

11

Death propagates in shared memory unless you do a ton of defensive programming.
Due to the actor model with no shared memory it is custom in Erlang to do fail-fast programming.
No shared memory allows you to fail fast when suitable.
Erlang uses message passing between processes to exchange information.

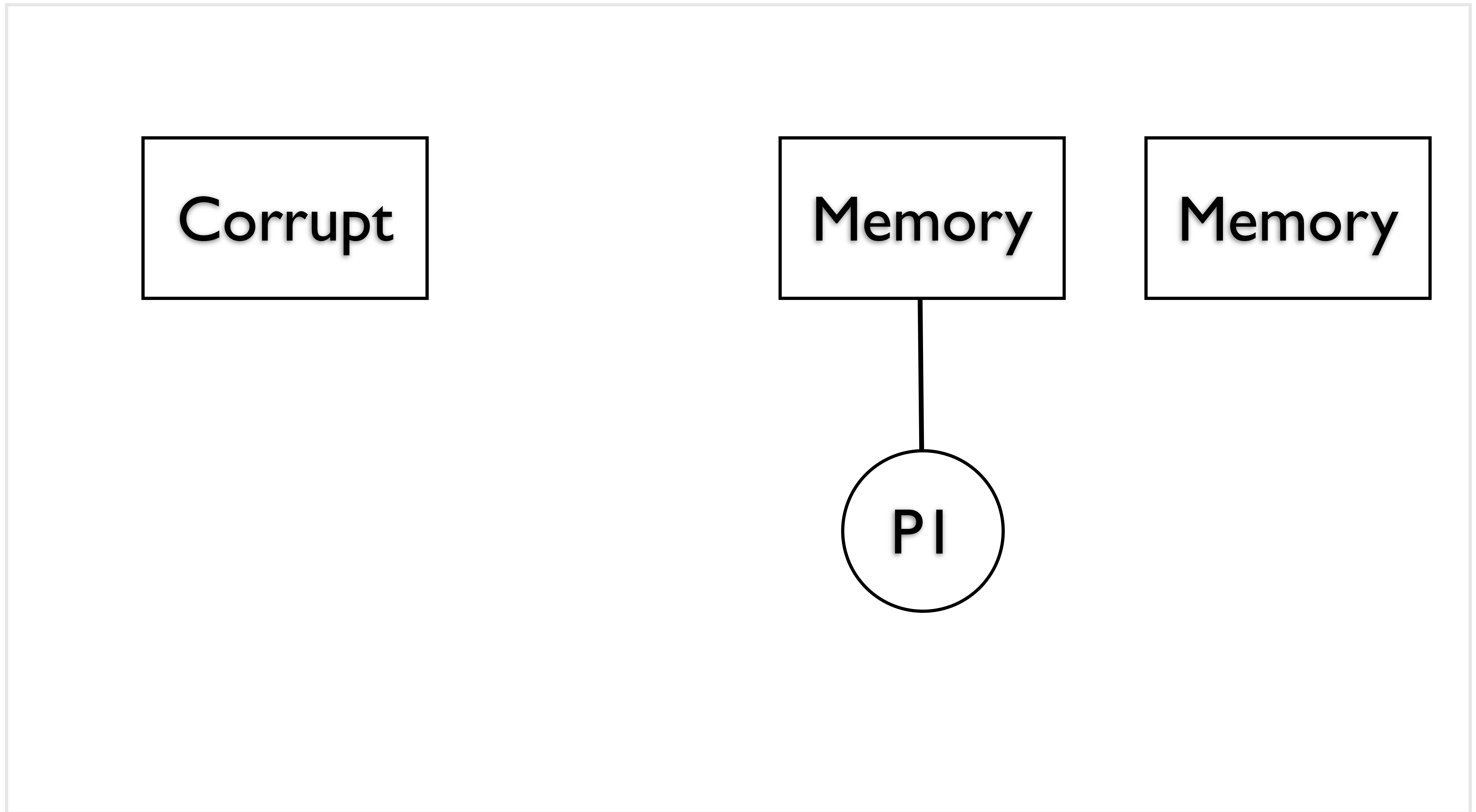
To Share Or Not To Share



© 1999-2012 Erlang Solutions Ltd.

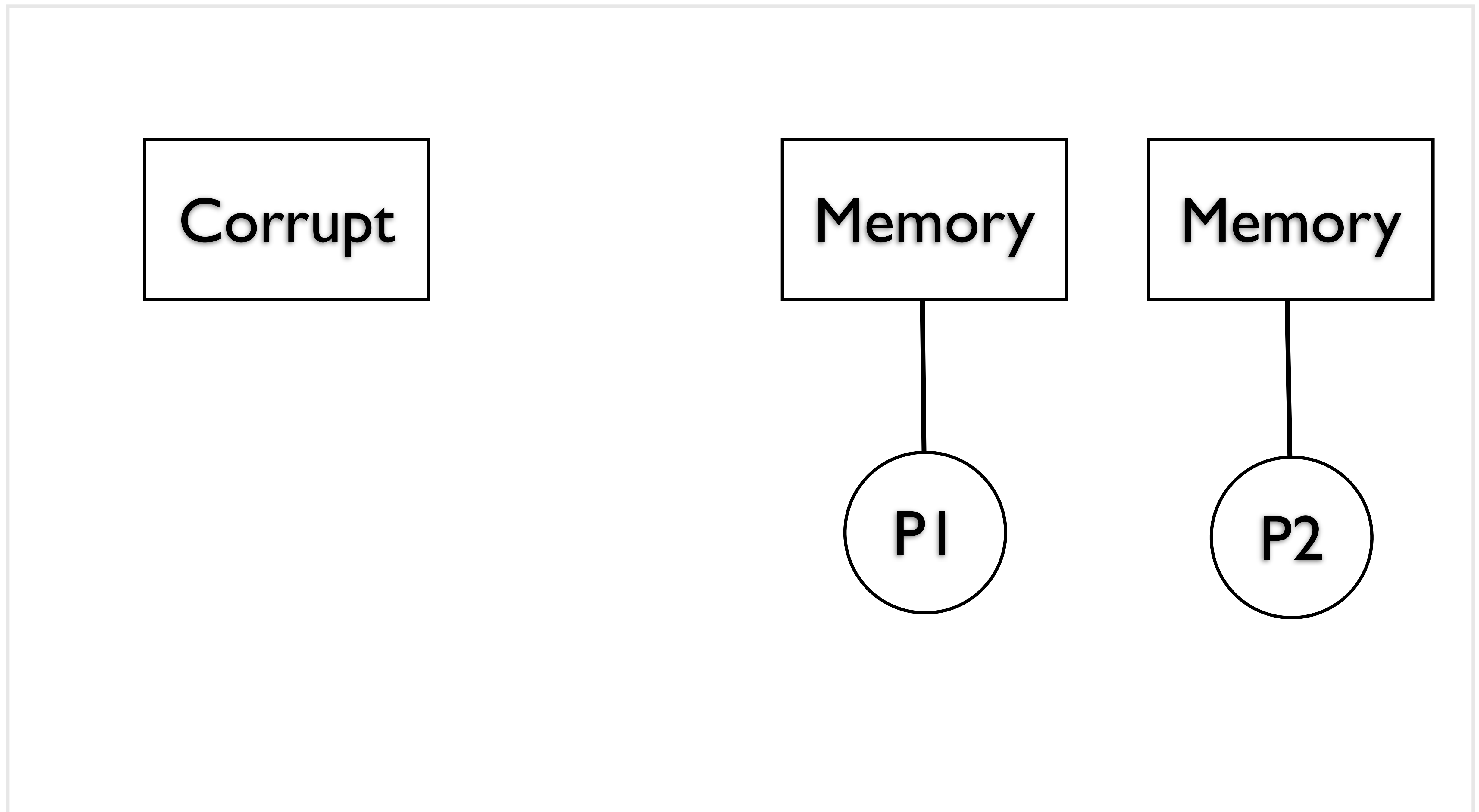
Death propagates in shared memory unless you do a ton of defensive programming.
Due to the actor model with no shared memory it is custom in Erlang to do fail-fast programming.
No shared memory allows you to fail fast when suitable.
Erlang uses message passing between processes to exchange information.

To Share Or Not To Share



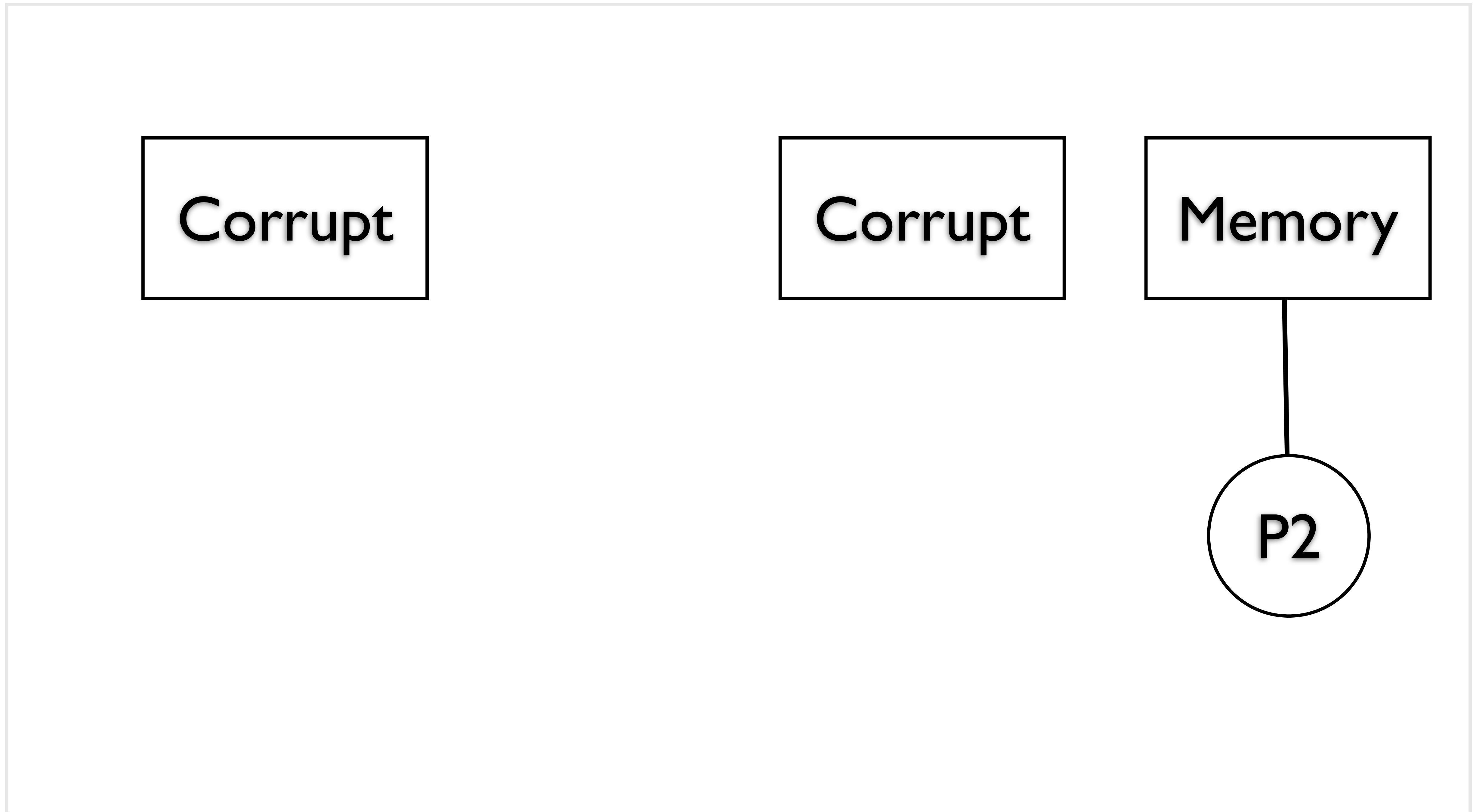
© 1999-2012 Erlang Solutions Ltd.

Death propagates in shared memory unless you do a ton of defensive programming.
Due to the actor model with no shared memory it is custom in Erlang to do fail-fast programming.
No shared memory allows you to fail fast when suitable.
Erlang uses message passing between processes to exchange information.



Death propagates in shared memory unless you do a ton of defensive programming.
Due to the actor model with no shared memory it is custom in Erlang to do fail-fast programming.
No shared memory allows you to fail fast when suitable.
Erlang uses message passing between processes to exchange information.

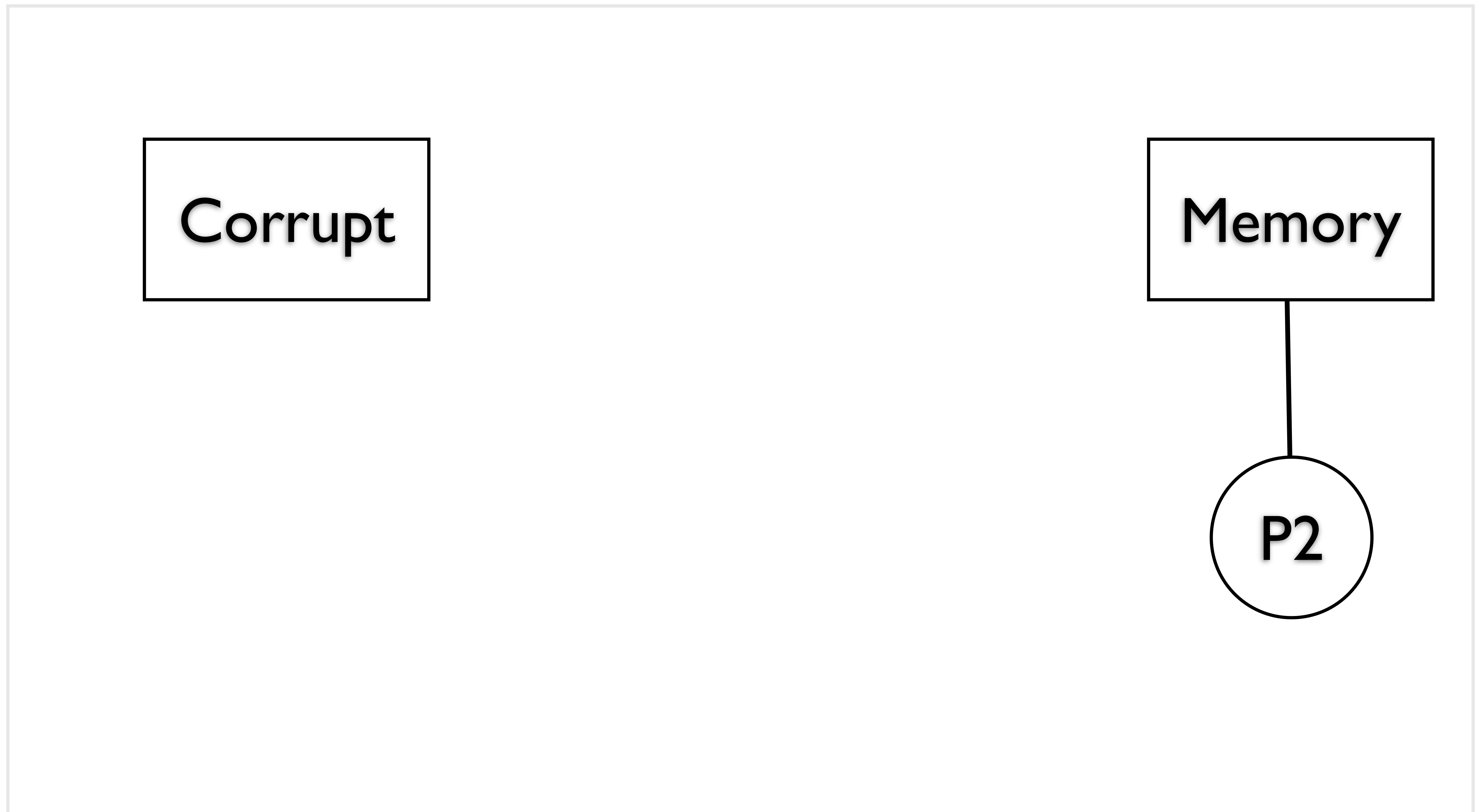
To Share Or Not To Share



© 1999-2012 Erlang Solutions Ltd.

Death propagates in shared memory unless you do a ton of defensive programming.
Due to the actor model with no shared memory it is custom in Erlang to do fail-fast programming.
No shared memory allows you to fail fast when suitable.
Erlang uses message passing between processes to exchange information.

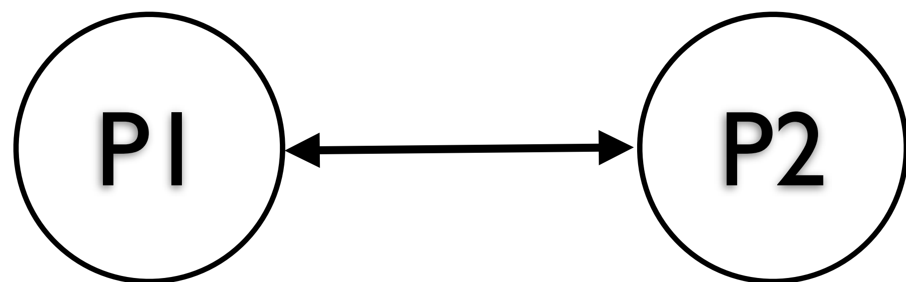
To Share Or Not To Share



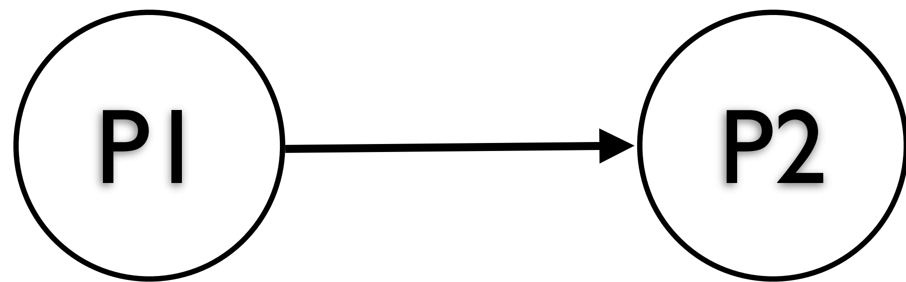
© 1999-2012 Erlang Solutions Ltd.

Death propagates in shared memory unless you do a ton of defensive programming.
Due to the actor model with no shared memory it is custom in Erlang to do fail-fast programming.
No shared memory allows you to fail fast when suitable.
Erlang uses message passing between processes to exchange information.

Dealing With Failures



link =
die together

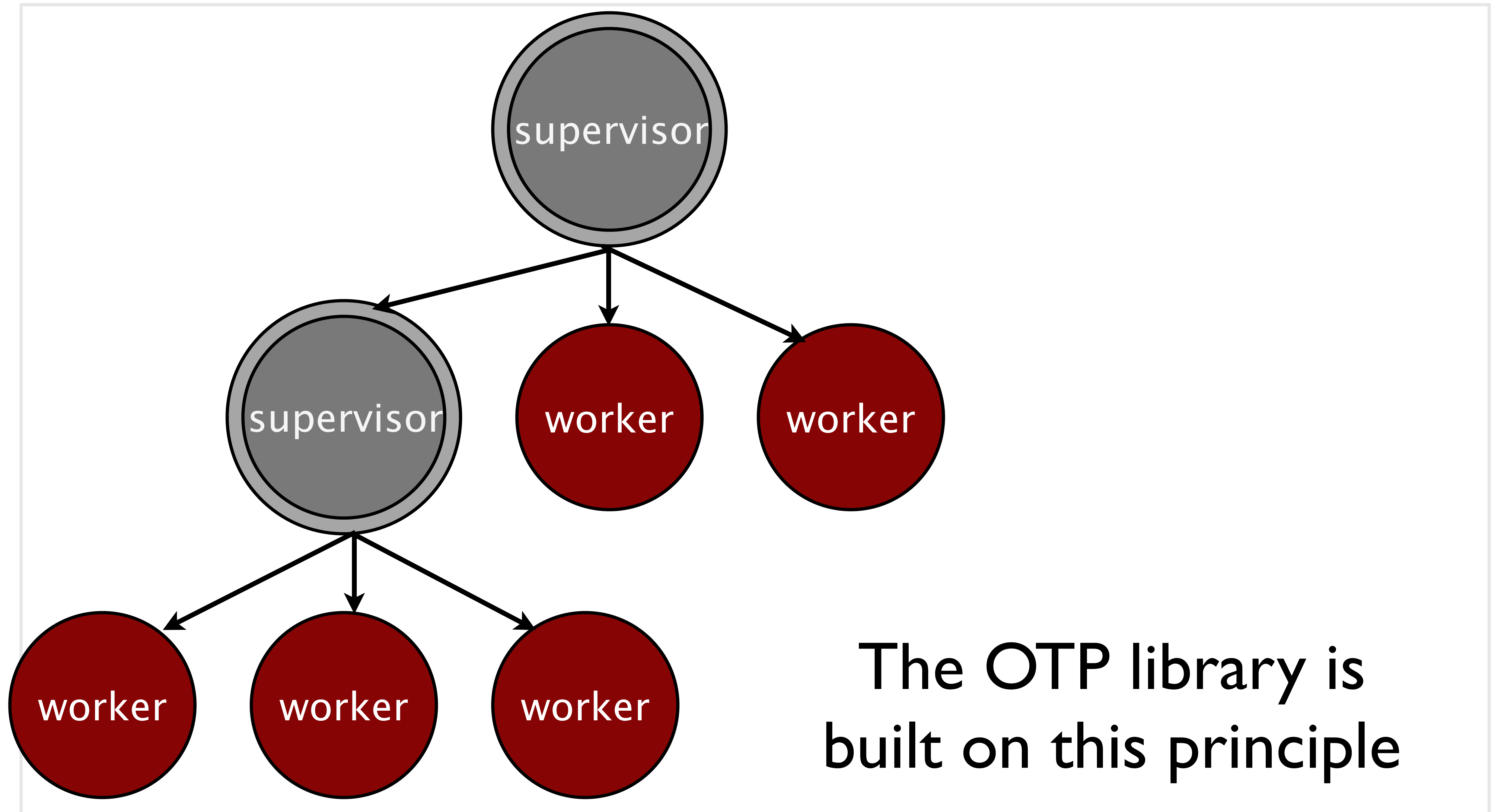


monitor =
notification
of death

© 1999-2012 Erlang Solutions Ltd.

The ability to link processes and monitor them is the key to adopting fail-fast fully.
Link & monitor works across machines!

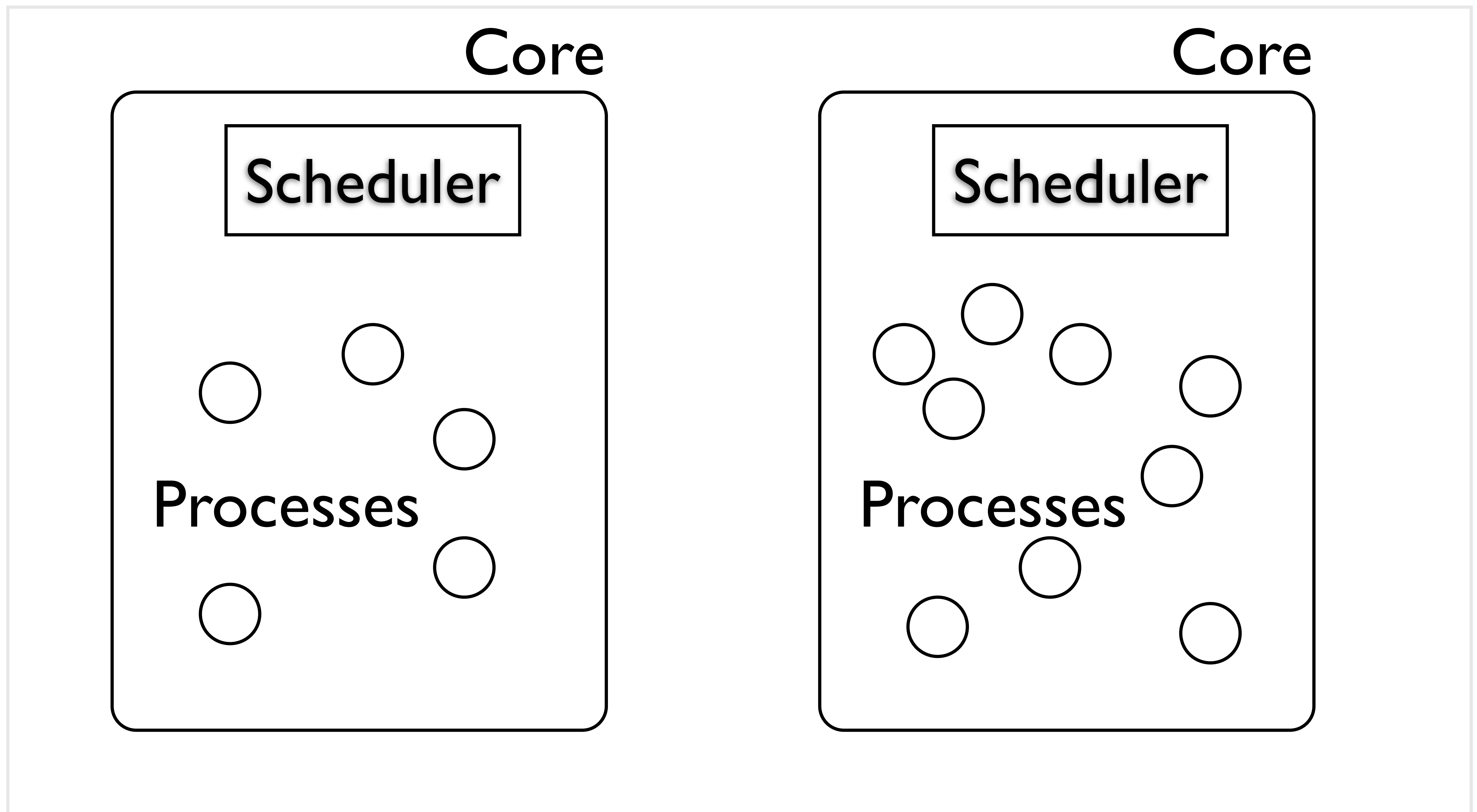
Supervision Trees



Erlang SOLUTIONS
© 1999-2012 Erlang Solutions Ltd.

Robust systems does not happen by accident – even in Erlang!
You have to think about the consequences of a worker process that fails and let the supervisor take appropriate actions.
Using the OTP library's components makes it straightforward to implement the supervision tree, which has the added benefit that all things are started in the right order.

Distribution Over Cores



Erlang SOLUTIONS
© 1999-2012 Erlang Solutions Ltd.

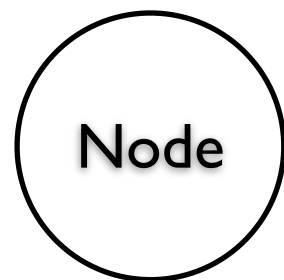
There is 1 scheduler per core.
The VM tries to load balance across the available cores.
Scales extremely well with the addition of extra cores – WITHOUT changing the programs!

Distribution Over Machines

© 1999-2012 Erlang Solutions Ltd.

Each instance of the Erlang runtime is called a node.
There can be several nodes on one machine if you fancy that.
Nodes detect when other nodes are not around any more - the programmer can then decide what to do.
If you have the PID (Process Identifier) of a process you do not care which node it is on. You

Distribution Over Machines



© 1999-2012 Erlang Solutions Ltd.

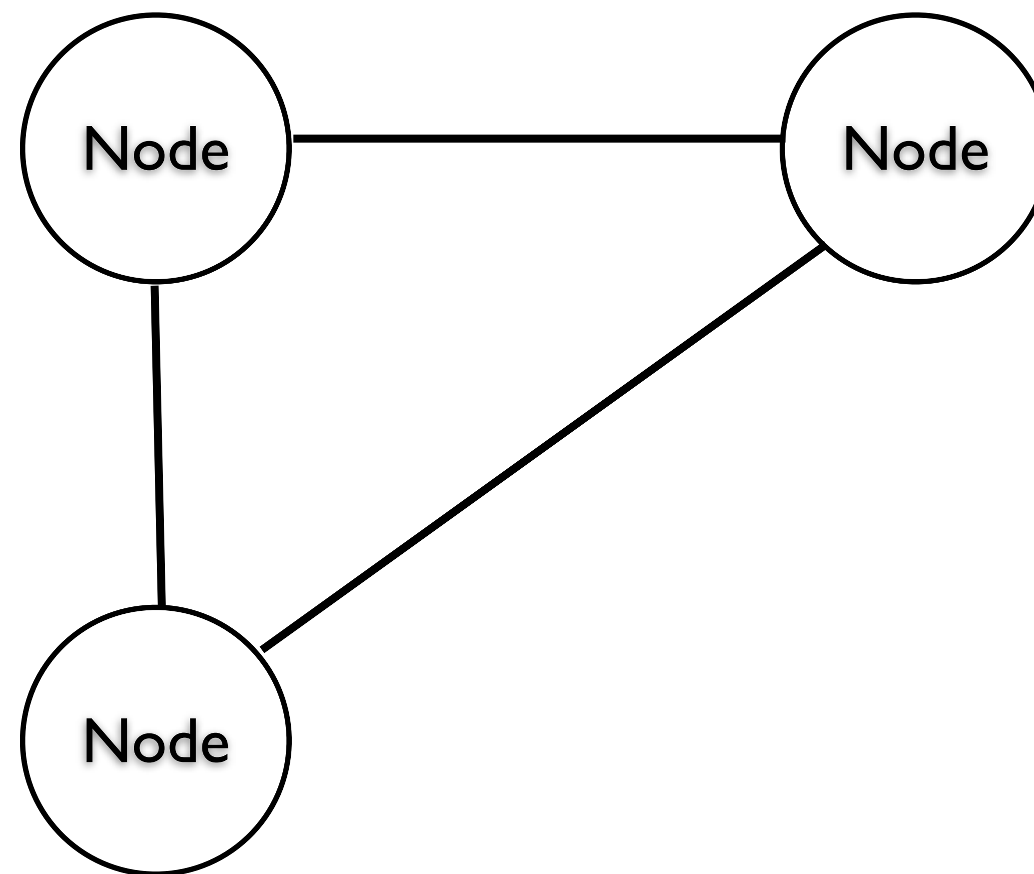
Each instance of the Erlang runtime is called a node.
There can be several nodes on one machine if you fancy that.
Nodes detect when other nodes are not around any more – the programmer can then decide what to do.
If you have the PID (Process Identifier) of a process you do not care which node it is on. You

Distribution Over Machines

© 1999-2012 Erlang Solutions Ltd.

Each instance of the Erlang runtime is called a node.
There can be several nodes on one machine if you fancy that.
Nodes detect when other nodes are not around any more – the programmer can then decide what to do.
If you have the PID (Process Identifier) of a process you do not care which node it is on. You

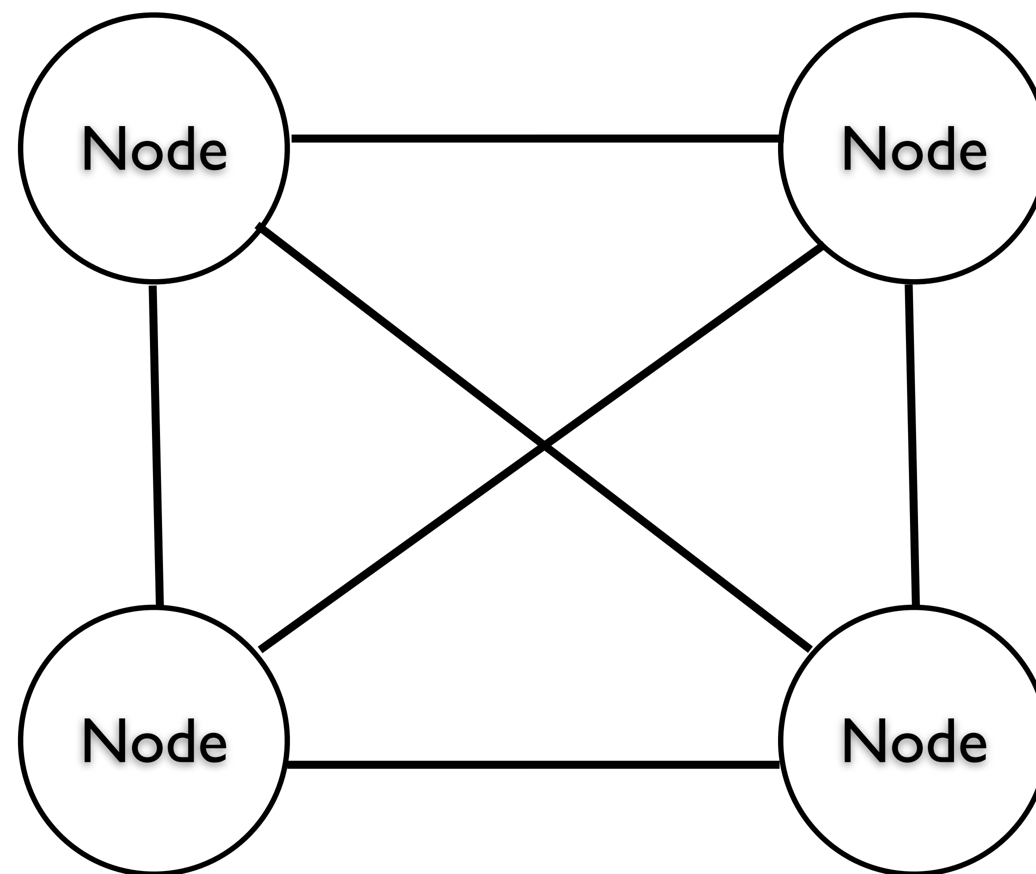
Distribution Over Machines



© 1999-2012 Erlang Solutions Ltd.

Each instance of the Erlang runtime is called a node.
There can be several nodes on one machine if you fancy that.
Nodes detect when other nodes are not around any more – the programmer can then decide what to do.
If you have the PID (Process Identifier) of a process you do not care which node it is on. You

Distribution Over Machines



© 1999-2012 Erlang Solutions Ltd.

Each instance of the Erlang runtime is called a node.
There can be several nodes on one machine if you fancy that.
Nodes detect when other nodes are not around any more - the programmer can then decide what to do.
If you have the PID (Process Identifier) of a process you do not care which node it is on. You

Staying Alive...

Learn New Moves!

© 1999-2012 Erlang Solutions Ltd.

As style changed a new group entered the top of the hip scale.
For most it meant a serious restart or a stop altogether.

Staying Alive...

Learn New Moves!

© 1999-2012 Erlang Solutions Ltd.

As style changed a new group entered the top of the hip scale.
For most it meant a serious restart or a stop altogether.

Staying Alive...

Learn New Moves!

© 1999-2012 Erlang Solutions Ltd.

As style changed a new group entered the top of the hip scale.
For most it meant a serious restart or a stop altogether.

Staying Alive...

Learn New Moves!

© 1999-2012 Erlang Solutions Ltd.

As style changed a new group entered the top of the hip scale.
For most it meant a serious restart or a stop altogether.

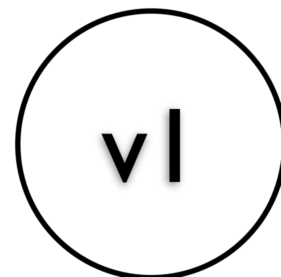
Staying Alive Erlang Style

© 1999-2012 Erlang Solutions Ltd.

With Erlang you can survive upgrades without loosing service.
Along with the code change signal you specify how the internal state of the process should be updated before continuing.

Staying Alive Erlang Style

Process running

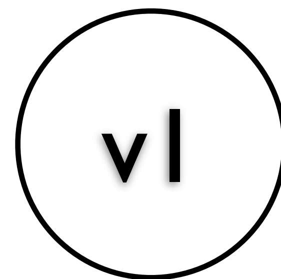


© 1999-2012 Erlang Solutions Ltd.

With Erlang you can survive upgrades without loosing service.
Along with the code change signal you specify how the internal state of the process should be updated before continuing.

Staying Alive Erlang Style

Process running



Code loaded: v2

© 1999-2012 Erlang Solutions Ltd.

With Erlang you can survive upgrades without loosing service.
Along with the code change signal you specify how the internal state of the process should be updated before continuing.

Staying Alive Erlang Style

Process running

Code loaded: v2

© 1999-2012 Erlang Solutions Ltd.

With Erlang you can survive upgrades without loosing service.
Along with the code change signal you specify how the internal state of the process should be updated before continuing.

Staying Alive Erlang Style

Process running

Code loaded: v2

© 1999-2012 Erlang Solutions Ltd.

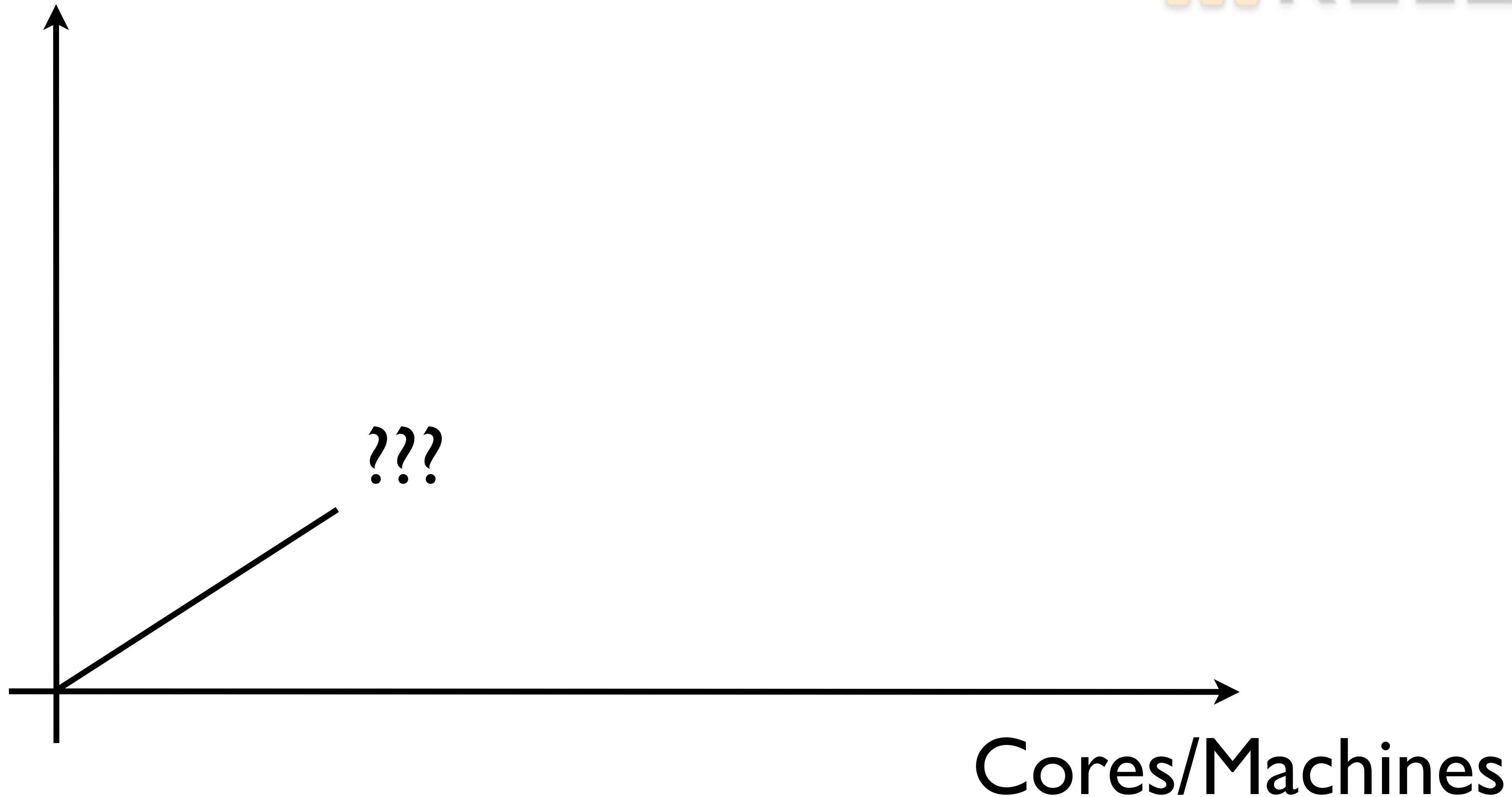
With Erlang you can survive upgrades without loosing service.
Along with the code change signal you specify how the internal state of the process should be updated before continuing.

Challenges

© 1999-2012 Erlang Solutions Ltd.

18

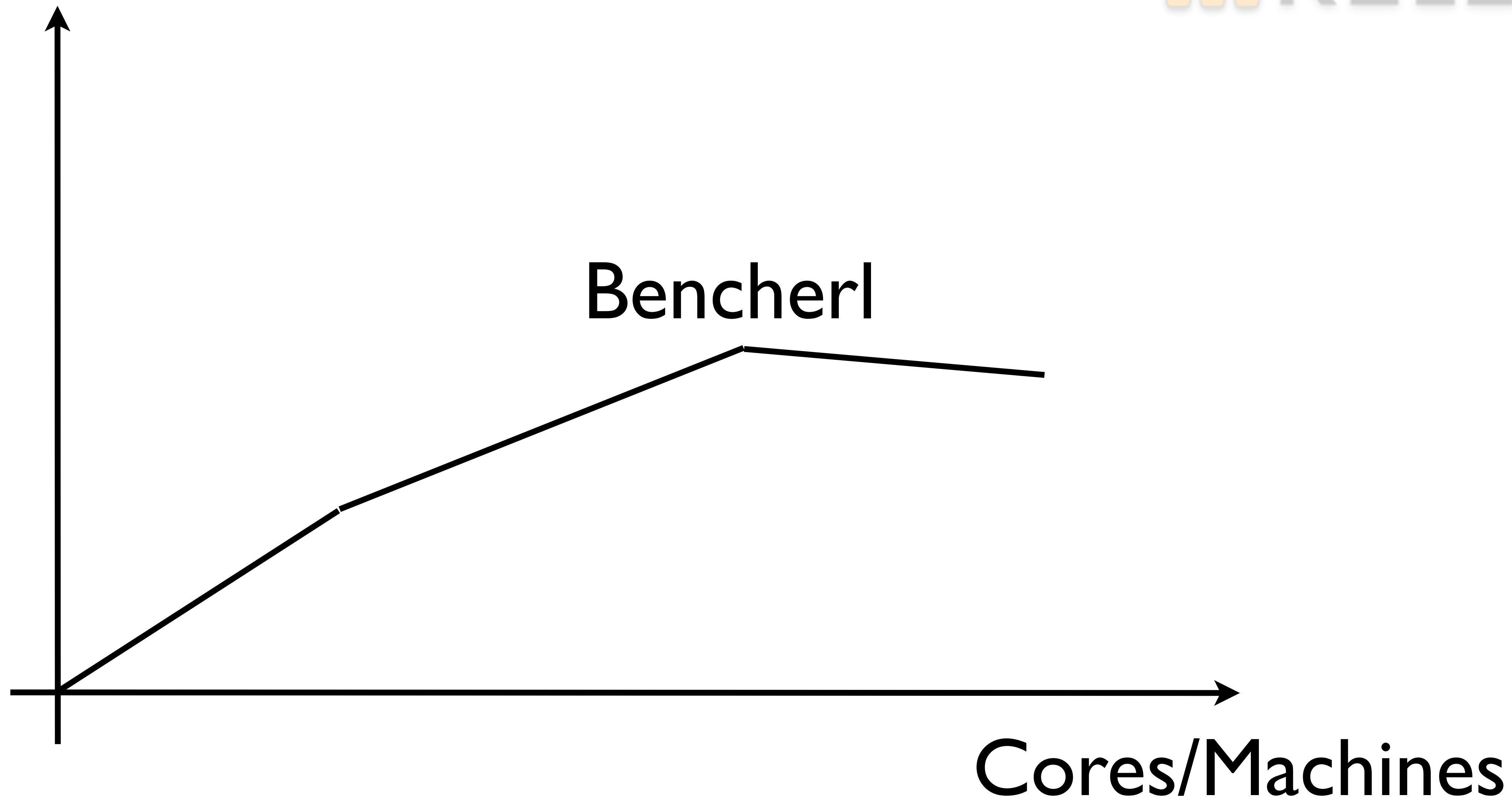
How Will I Know If It Really Scales?



© 1999-2012 Erlang Solutions Ltd.

A Scalability Benchmark Suite for Erlang/OTP.
http://www.softlab.ntua.gr/release/bencherl/index.html
A number of synthetic benchmarks plus real-world (dialyzer and scalaris).
Extendable to test your own application.

How Will I Know If It Really Scales?

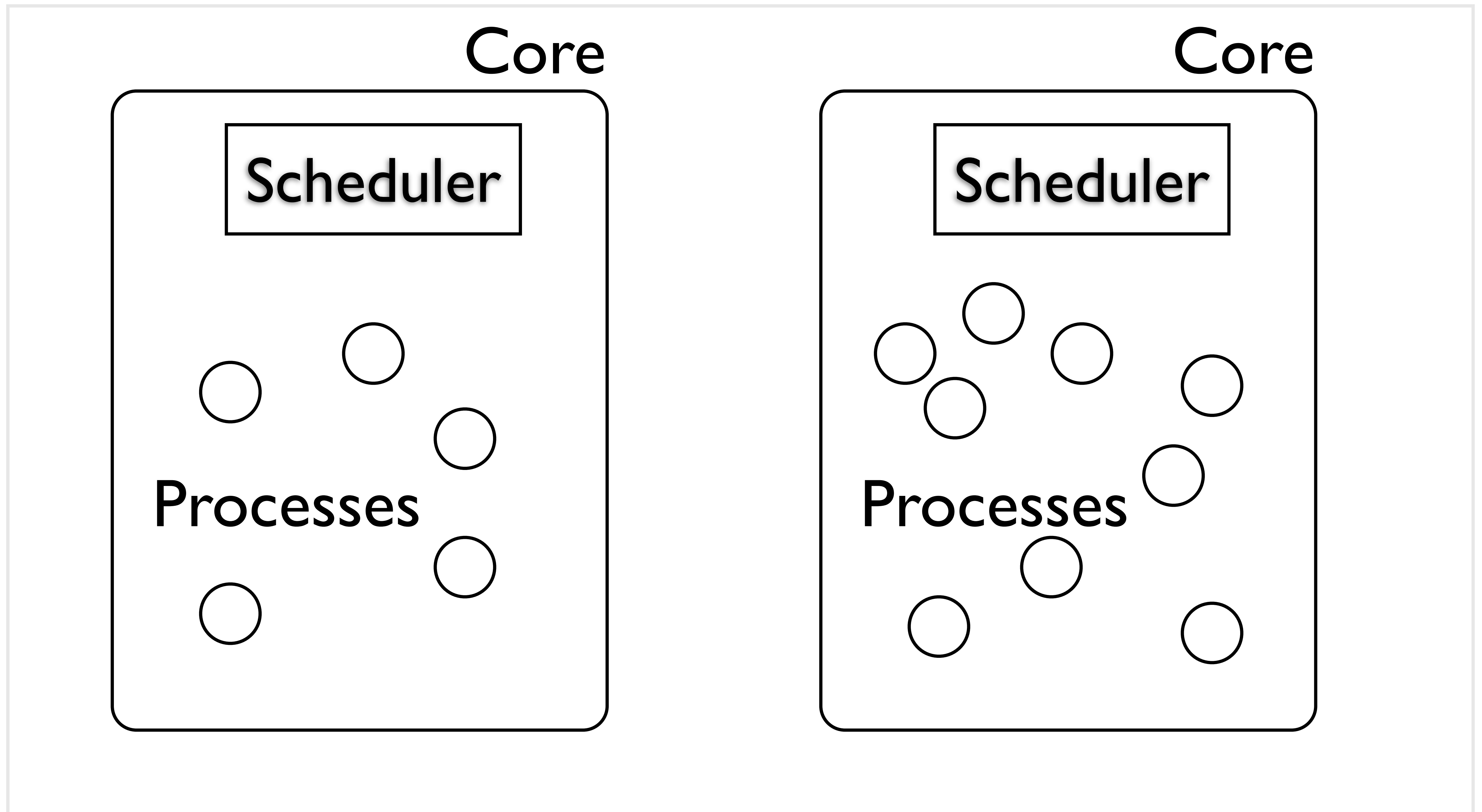


© 1999-2012 Erlang Solutions Ltd.

A Scalability Benchmark Suite for Erlang/OTP.
http://www.softlab.ntua.gr/release/bencherl/index.html
A number of synthetic benchmarks plus real-world (dialyzer and scalaris).
Extendable to test your own application.

Who Is Doing What?

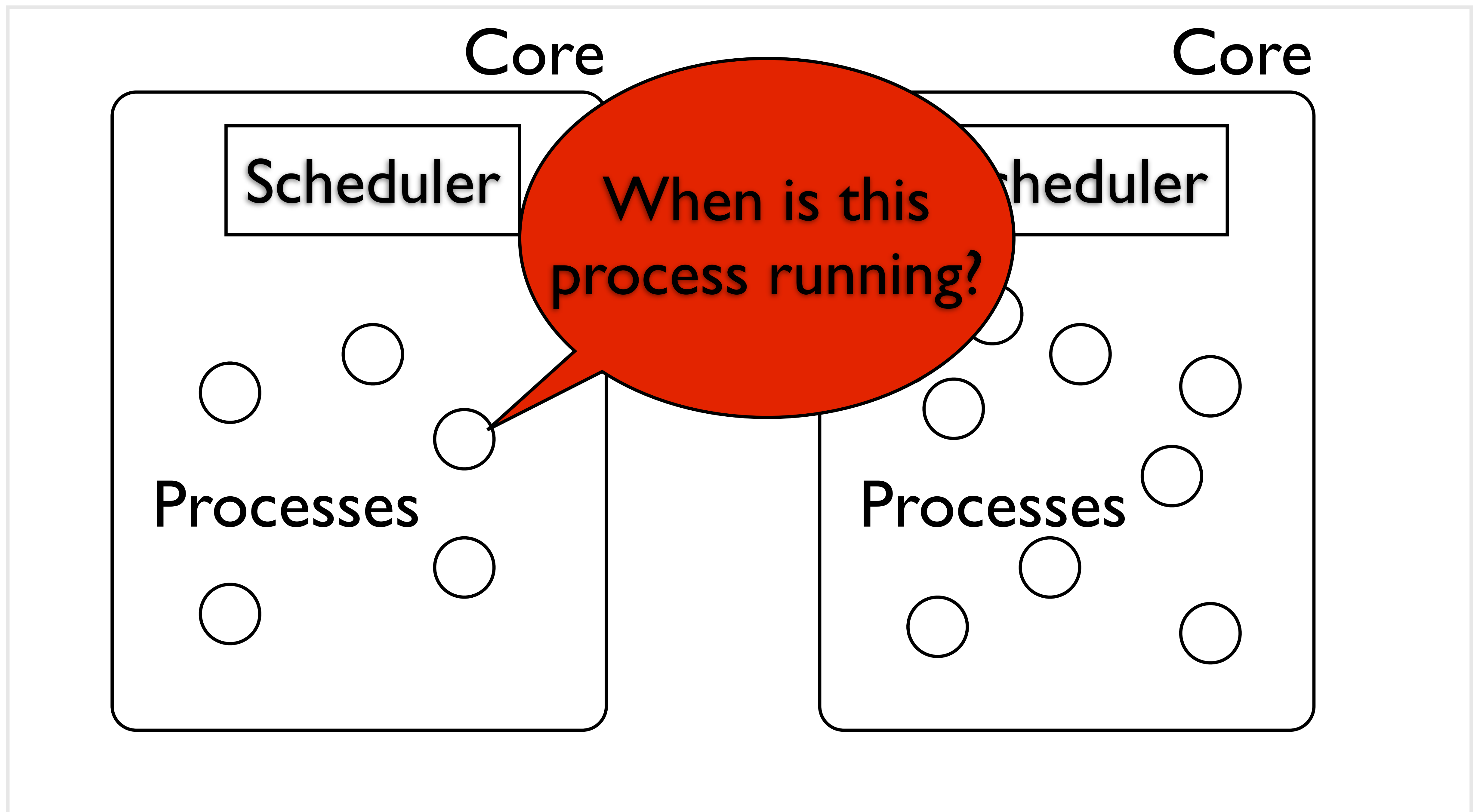
RELEASE



Erlang SOLUTIONS
© 1999-2012 Erlang Solutions Ltd.

Who Is Doing What?

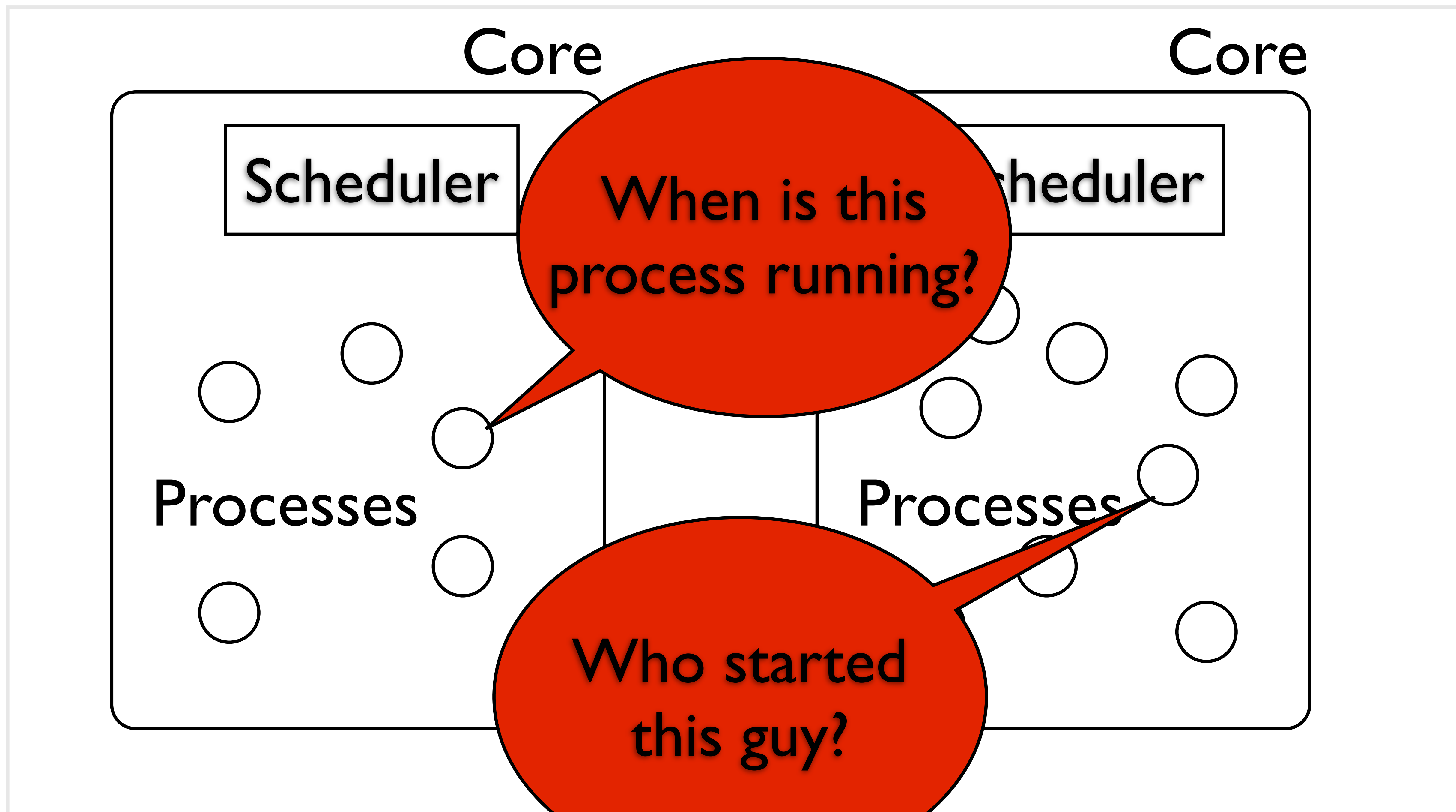
RELEASE



© 1999-2012 Erlang Solutions Ltd.

Who Is Doing What?

RELEASE



© 1999-20 olutions Ltd.

Percept2 To The Rescue

Active functions

overview | processes | ports | function activities | inter-node messaging

pid	module:function/arity	activity	function start/end secs	monitor start/end secs
<0.36.0>	{sim_code,sim_code_detection,8}		{1.0e-6,2.854873}	{0.419,1.1255}
<0.36.0>	{sim_code,sim_code_detection,4}		{0.016239,2.839082}	{0.419,1.1255}
<0.36.0>	{sim_code,sim_code_detection_1,6}		{0.016253,2.607782}	{0.419,1.1255}
<0.36.0>	{sim_code,generalise_and_hash_ast,6}		{0.016274,2.607782}	{0.419,1.1255}
<0.36.0>	{sim_code,pforeach,2}		{0.016275,2.607782}	{0.419,1.1255}
<0.775.0>	{sim_code,pforeach_1,3}		{0.016385,2.607765}	{0.419,1.1255}
<0.775.0>	{sim_code,'-generalise_and_hash_ast/6-fun-0-',6}		{0.016386,2.607761}	{0.419,1.1255}
<0.775.0>	{sim_code,generalise_and_hash_file_ast_1,7}		{0.016387,2.607761}	{0.419,1.1255}
<0.775.0>	{sim_code,pforeach,2}		{1.107858,2.607761}	{0.419,1.1255}
<0.776.0>	{sim_code,pforeach_1,3}		{0.016393,0.454145}	{0.419,1.1255}
<0.776.0>	{sim_code,'-generalise_and_hash_ast/6-fun-0-',6}		{0.016394,0.454131}	{0.419,1.1255}
<0.776.0>	{sim_code,generalise_and_hash_file_ast_1,7}		{0.016395,0.454131}	{0.419,1.1255}
<0.776.0>	{sim_code,pforeach,2}		{0.265244,0.454131}	{0.419,1.1255}
<0.780.0>	{sim_code,pforeach_1,3}		{0.016425,0.454381}	{0.419,1.1255}
<0.780.0>	{sim_code,'-generalise_and_hash_ast/6-fun-0-',6}		{0.016426,0.454377}	{0.419,1.1255}
<0.780.0>	{sim_code,generalise_and_hash_file_ast_1,7}		{0.016427,0.454377}	{0.419,1.1255}
<0.780.0>	{sim_code,pforeach,2}		{0.32833,0.454377}	{0.419,1.1255}
<0.891.0>	{sim_code,pforeach_0,3}		{0.26539,0.454128}	{0.419,1.1255}
<0.891.0>	{sim_code,pforeach_wait,2}		{0.265718,0.454116}	{0.419,1.1255}
<0.891.0>	{sim_code,pforeach_wait,2}		{0.265979,0.454116}	{0.419,1.1255}
<0.893.0>	{sim_code,pforeach_1,3}		{0.265735,0.454025}	{0.419,1.1255}
<0.893.0>	{sim_code,'-generalise_and_hash_file_ast_1/7-fun-1-',2}		{0.265736,0.45402}	{0.419,1.1255}
<0.893.0>	{sim_code,'-generalise_and_hash_file_ast_1/7-fun-0-',6}		{0.265737,0.45402}	{0.419,1.1255}
<0.893.0>	{sim_code,generalise_and_hash_function_ast,6}		{0.265738,0.45402}	{0.419,1.1255}

© 1999-2012 Erlang Solutions Ltd.

Percept: Erlang Concurrency Profiling Tool, utilizes trace informations and profiler events to form a picture of the processes's and ports runnability.
Percept2 is an extension of Percept (part of the OTP release).
Extensions: # of schedulers active, active functions, process migration, message passing stats, inter-node communication

Memory Alloc Previously

R12B-1

Thread 1 ... Thread N

Scheduler 1 | Scheduler 2 | ... | Scheduler N

sl_alloc
eheap_alloc
binary_alloc

sl_alloc
eheap_alloc
binary_alloc

...

sl_alloc
eheap_alloc
binary_alloc

ets_alloc | ets_alloc | ets_alloc

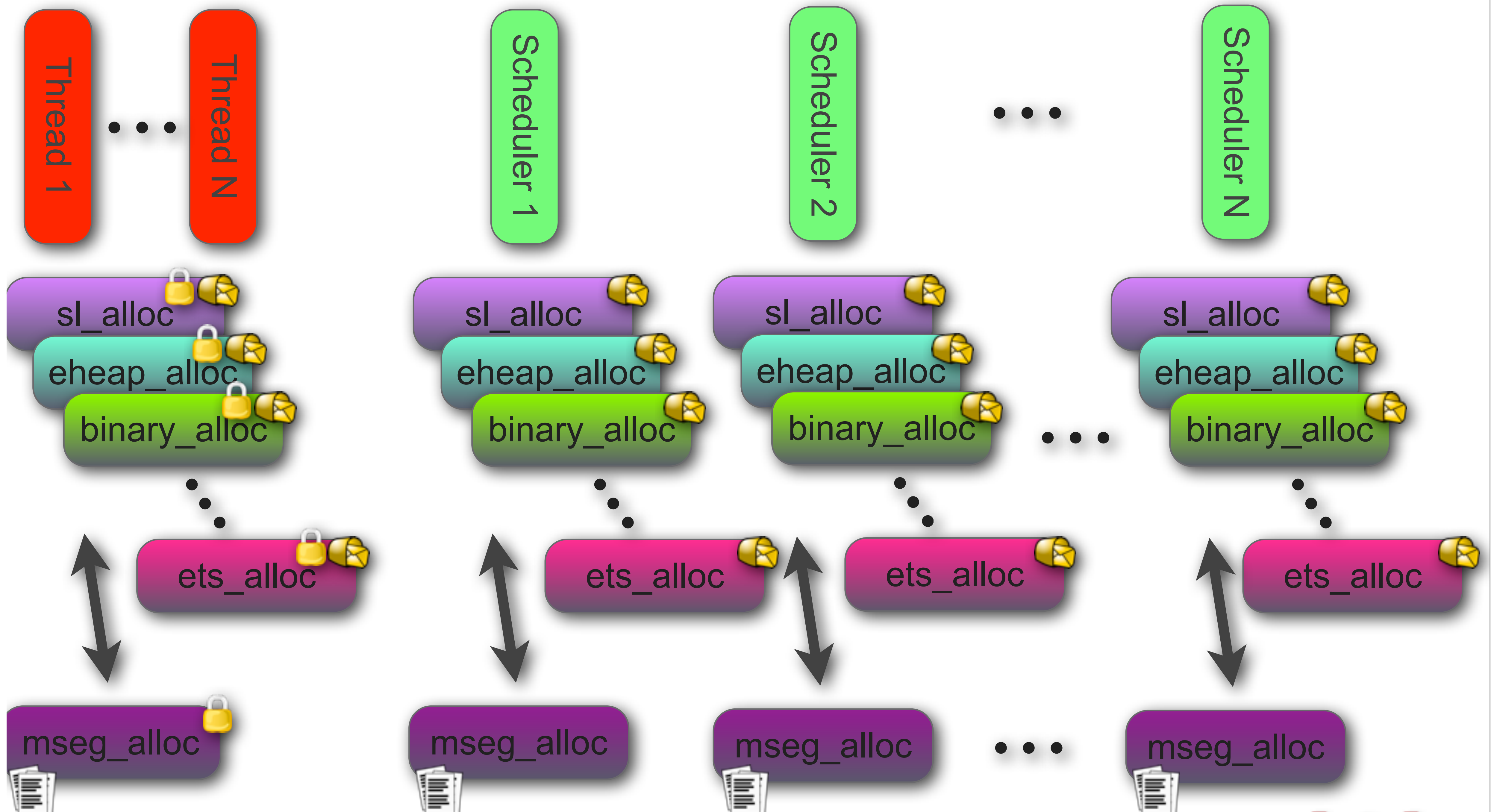
mseg_alloc

RELEASE

© 1999-2012 Erlang Solutions Ltd.

One central memory allocator for all schedulers on the same machine

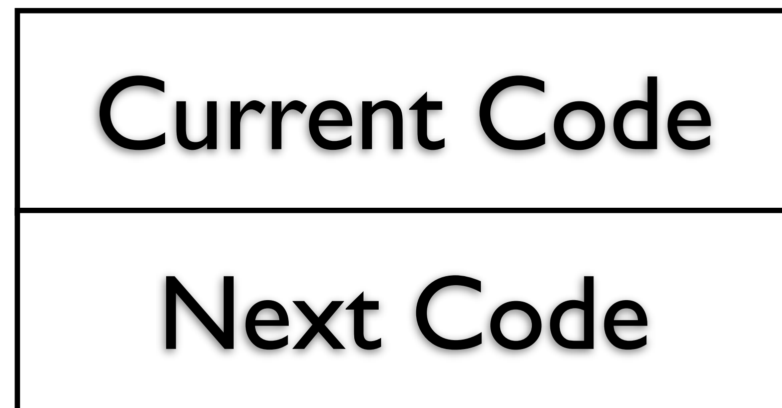
Memory Alloc Now

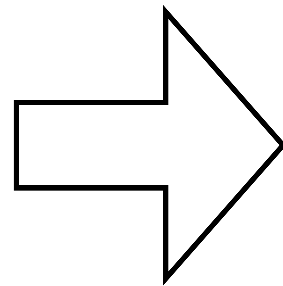


© 1999-2012 Erlang Solutions Ltd.

One central memory allocator for all schedulers on the same machien

Upgrading Blocks

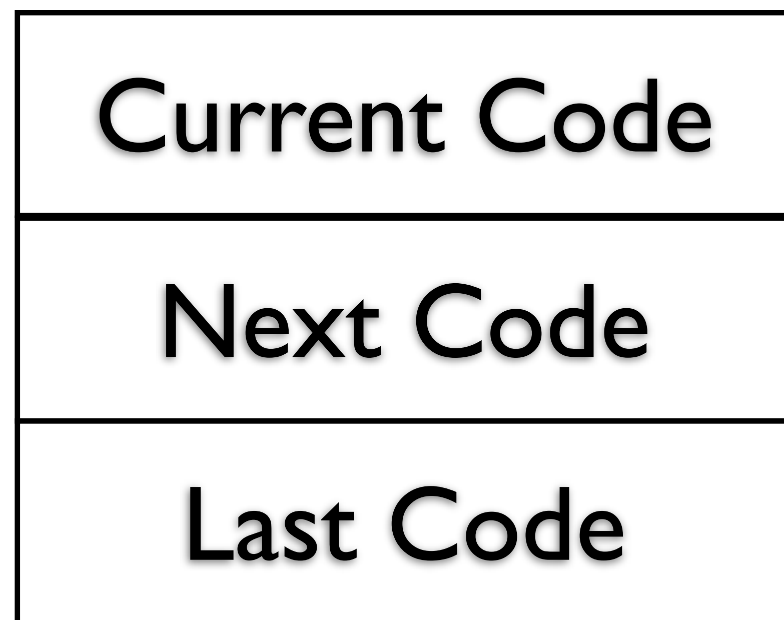


Load code  All schedulers blocked

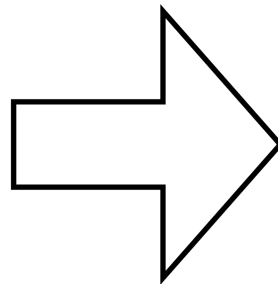
© 1999-2012 Erlang Solutions Ltd.

24

Upgrade Without Blocking



Load code

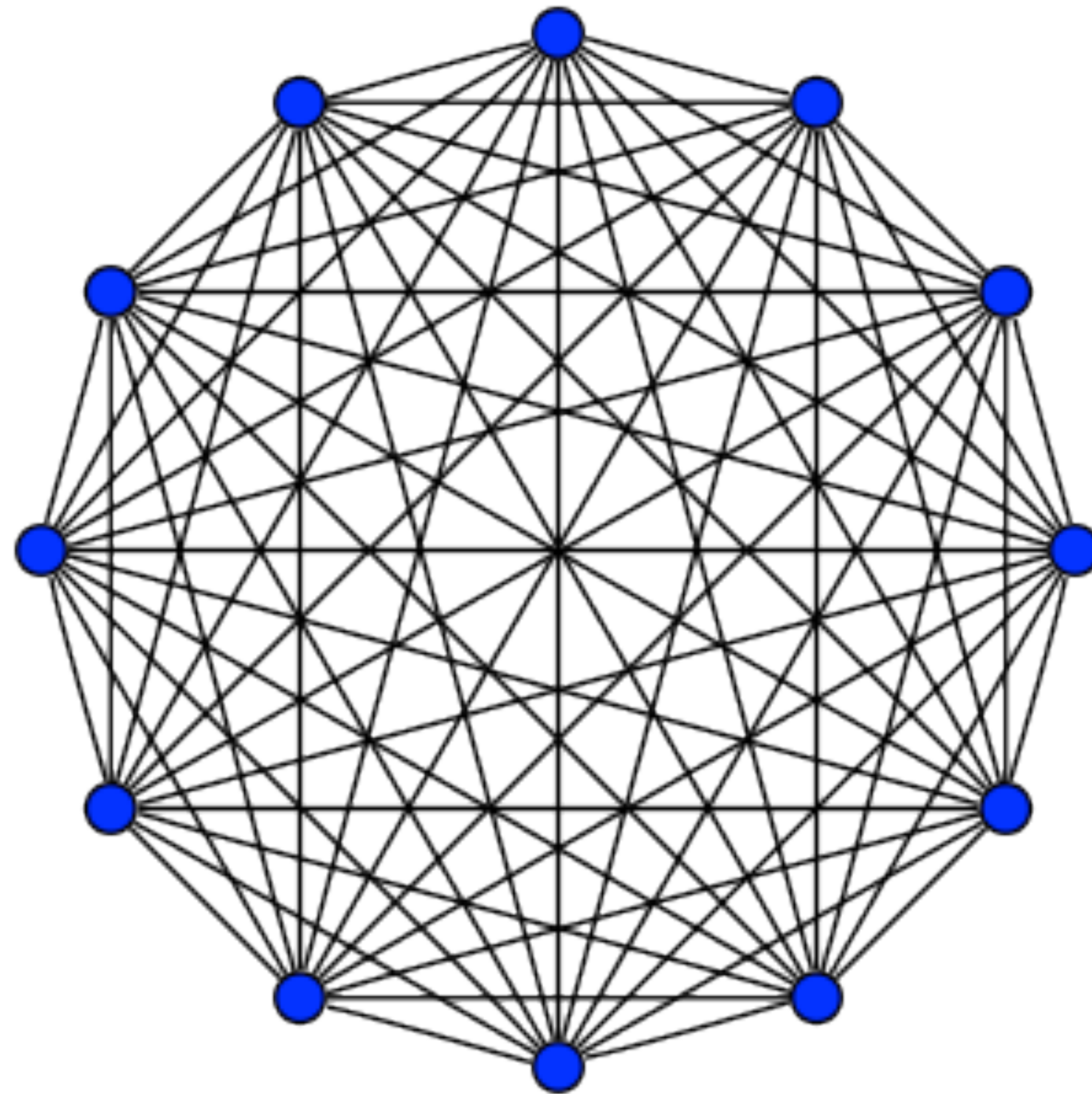


Each scheduler does it when needed

© 1999-2012 Erlang Solutions Ltd.

Coming in R16!

Fully Connected



Erlang SOLUTIONS

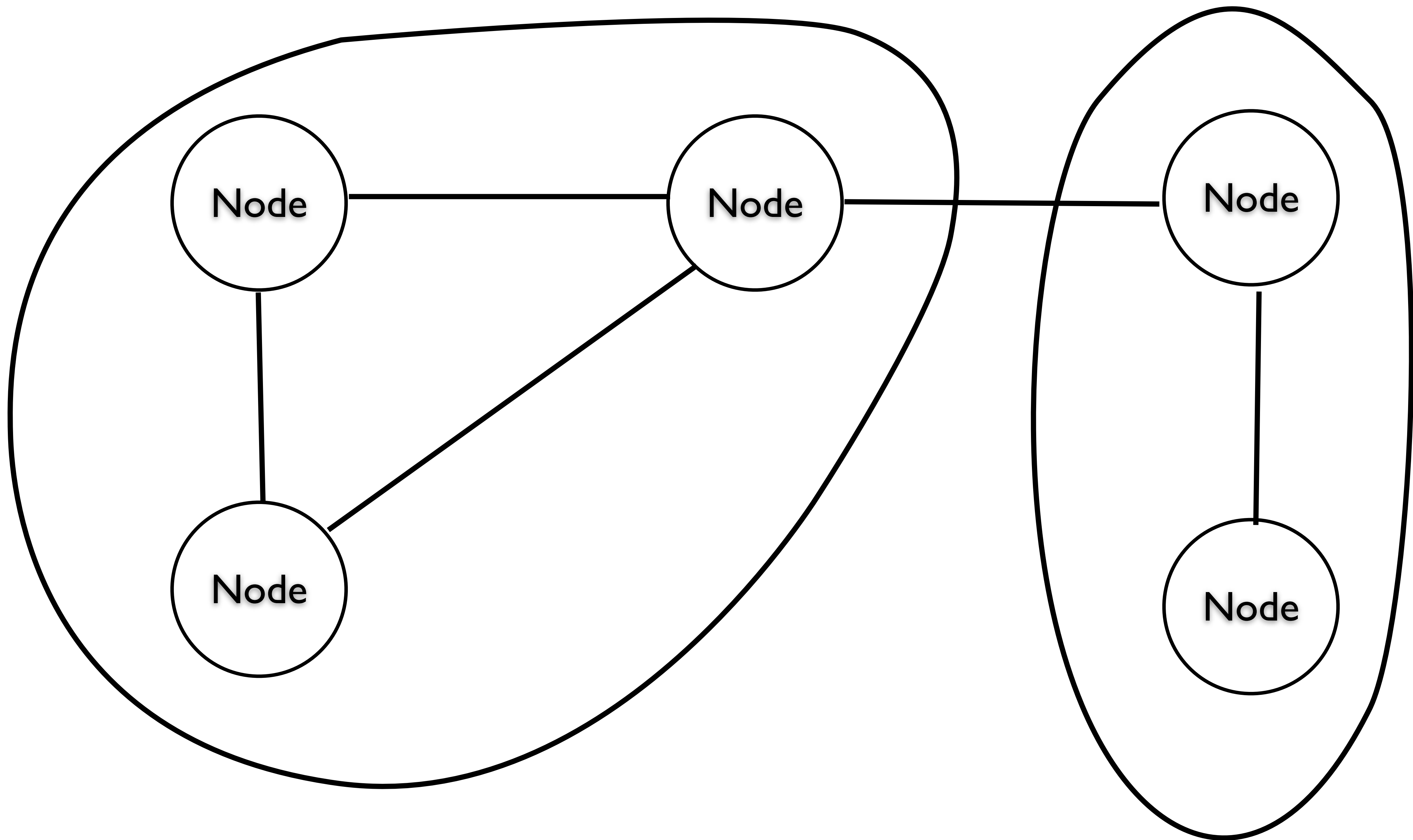
© 1999-2012 Erlang Solutions Ltd.

26

Erlang connects all nodes fully.
So you get a lot of connections.

Scaling Over Machines

RELEASE



© 1999-2012 Erlang Solutions Ltd.

A fully connected system might not be right for every problem.
s_groups allows you to create clusters of nodes.
Nodes inside a cluster are fully connected.
Connections between clusters can be arbitrary.

Managing Erlang Systems

© 1999-2012 Erlang Solutions Ltd.

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.
This is no different from any other language/technology!

Managing Erlang Systems

- Provision machines

© 1999-2012 Erlang Solutions Ltd.

28

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.
This is no different from any other language/technology!

Managing Erlang Systems

- Provision machines
- Deploy Erlang application

© 1999-2012 Erlang Solutions Ltd.

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.
This is no different from any other language/technology!

Managing Erlang Systems

- Provision machines
- Deploy Erlang application
- Attach to node

© 1999-2012 Erlang Solutions Ltd.

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.
This is no different from any other language/technology!

Managing Erlang Systems

- Provision machines
- Deploy Erlang application
- Attach to node
- Dig out metrics

© 1999-2012 Erlang Solutions Ltd.

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.
This is no different from any other language/technology!

Managing Erlang Systems

- Provision machines
- Deploy Erlang application
- Attach to node
- Dig out metrics

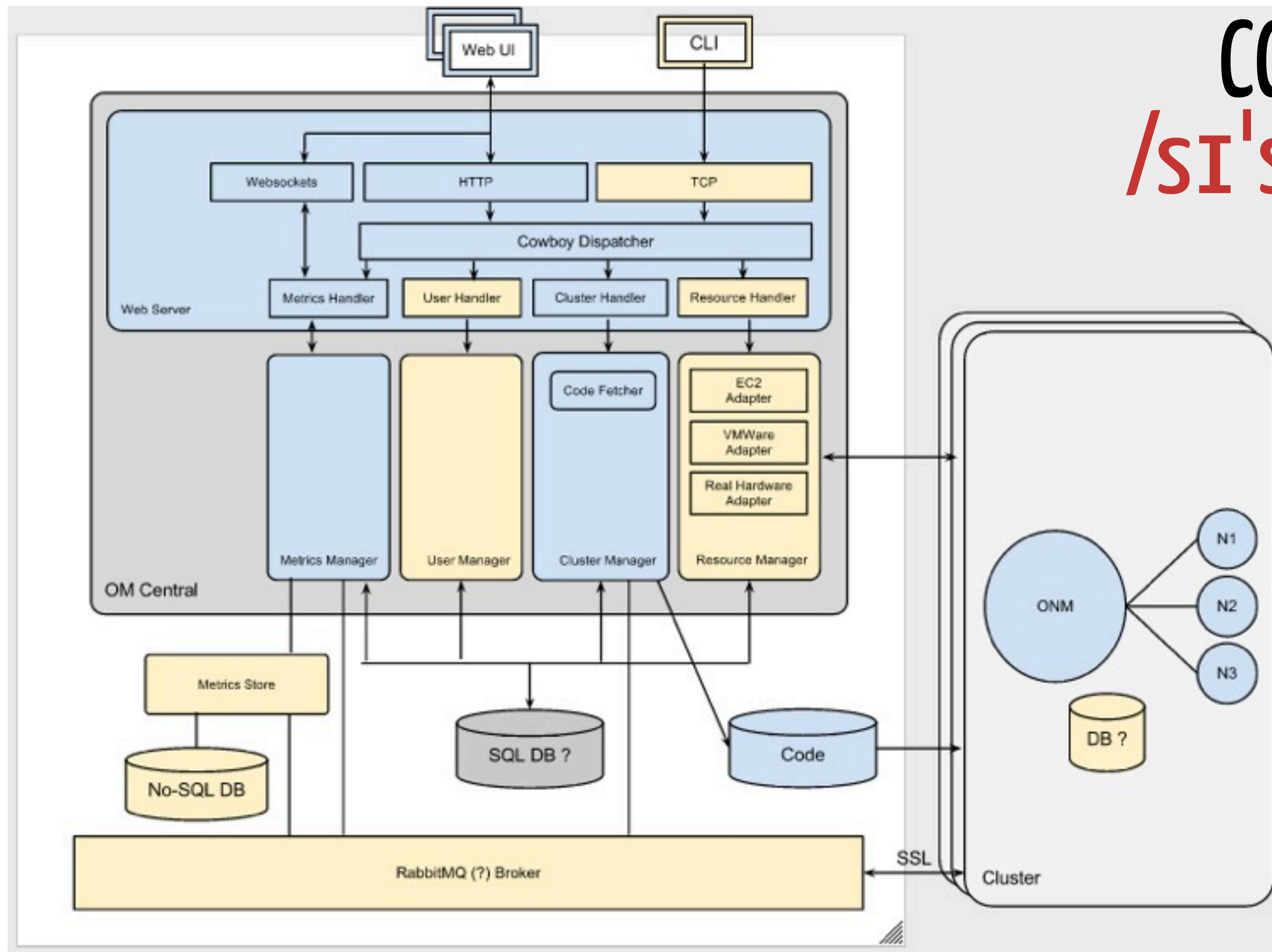
Memory usage
CPU load
Process hierarchy

© 1999-2012 Erlang Solutions Ltd.

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.
This is no different from any other language/technology!

Managing Big Systems

RELEASE



CCL
/sɪˈsɪlɪ/

© 1999-2012 Erlang Solutions Ltd.

Erlang SOLUTIONS

Erlang Solutions are building a tool as part of the RELEASE project to manage and operate big Erlang systems.
CCL = Cloud Computing Lace or Cloud Cuckoo Land depending on your mood.

Erlang And Parallelism

© 1999-2012 Erlang Solutions Ltd.

Erlang And Parallelism

- Created for

© 1999-2012 Erlang Solutions Ltd.

Erlang And Parallelism

- Created for
  - explicit concurrency

© 1999-2012 Erlang Solutions Ltd.

Erlang And Parallelism

- Created for
  - explicit concurrency
  - fault tollerance

Erlang SOLUTIONS
© 1999-2012 Erlang Solutions Ltd.

Erlang And Parallelism

- Created for
  - explicit concurrency
  - fault tollerance
  - highly concurrent systems

Erlang SOLUTIONS

© 1999-2012 Erlang Solutions Ltd.

Erlang And Parallelism

- Created for
  - explicit concurrency
  - fault tollerance
  - highly concurrent systems
- No direct support for

Erlang SOLUTIONS
© 1999-2012 Erlang Solutions Ltd.

Erlang And Parallelism

- Created for
  - explicit concurrency
  - fault tollerance
  - highly concurrent systems
- No direct support for
  - matrix multiplication

Erlang SOLUTIONS

© 1999-2012 Erlang Solutions Ltd.

Erlang And Parallelism

- Created for
  - explicit concurrency
  - fault tollerance
  - highly concurrent systems
- No direct support for
  - matrix multiplication
  - ray tracing

Erlang SOLUTIONS

© 1999-2012 Erlang Solutions Ltd.

Erlang And Parallelism

- Created for
  - explicit concurrency
  - fault tollerance
  - highly concurrent systems
- No direct support for
  - matrix multiplication
  - ray tracing
  - coarse grained parallel problems

Erlang SOLUTIONS
© 1999-2012 Erlang Solutions Ltd.

Intensional Parallelism

© 1999-2012 Erlang Solutions Ltd.

31

We are taking the good things from what has been learnt in the Haskell & Data-flow language communities and building a DSL which helps us leverage these types of parallel optimisations"

Intensional Parallelism

- Lucid like: demand-driven data computation
- Find short comings in the Erlang VM
- Variables are infinite streams of values

© 1999-2012 Erlang Solutions Ltd.

Erlang SOLUTIONS

We are taking the good things from what has been learnt in the Haskell & Data-flow language communities and building a DSL which helps us leverage these types of parallel optimisations"

Intensional Parallelism

- Lucid like: demand-driven data computation
- Find short comings in the Erlang VM
- Variables are infinite streams of values

```
running_avg
  where
     sum = first(input) fby sum + next(input);
     n = 1 fby n + 1;
     running_avg = sum / n;
  end;
```

Erlang SOLUTIONS

© 1999-2012 Erlang Solutions Ltd.

We are taking the good things from what has been learnt in the Haskell & Data-flow language communities and building a DSL which helps us leverage these types of parallel optimisations"

Intensional Parallelism

- Lucid like: demand-driven data computation
- Find short comings in the Erlang VM
- Variables are infinite streams of values

```
running_avg
  where
     sum = first(input) fby sum + next(input);
     n = 1 fby n + 1;
     running_avg = sum / n;
  end;
```

Black magic

© 1999-2012 Erlang Solutions Ltd.

We are taking the good things from what has been learnt in the Haskell & Data-flow language communities and building a DSL which helps us leverage these types of parallel optimisations"

Going Forward

© 1999-2012 Erlang Solutions Ltd.

Going Forward

- Consider Erlang when the problem fits

Erlang SOLUTIONS

© 1999-2012 Erlang Solutions Ltd.

Going Forward

- Consider Erlang when the problem fits
- More focus on right tool for the job

© 1999-2012 Erlang Solutions Ltd.

Going Forward

- Consider Erlang when the problem fits
- More focus on right tool for the job

Erlang SOLUTIONS

© 1999-2012 Erlang Solutions Ltd.