The Golden Trinity of Erlang
How Something Simple Has Real Business Value

Torben Hoffmann
CTO, Erlang Solutions
torben.hoffmann@erlang-solutions.com
@LeHoff

Why this talk?

Why this talk?

Introduce The Golden Trinity of Erlang

Why this talk?

Introduce The Golden Trinity of Erlang

Show the business value of the simple concepts that makes Erlang great

Why this talk?

Introduce The Golden Trinity of Erlang

Show the business value of the simple concepts that makes Erlang great

Spread the Erlang love

Some Erlang SOLUTIONS Customers

T-Mobile

ERICSSON

cellfind

VOCALINK safer payments, smarter partner

WhatsApp

CASSIDIAN

Lavelle Energy

medical-objects

E*TRADE

basho

ooVoo

19,000,000,000 reasons to use Erlang

Some Erlang Customers

WhatsApp

19,000,000,000 reasons to use Erlang

University Relations

UPPSALA UNIVERSITET

HERIOT WATT UNIVERSITY

UNIVERSITY OF GOTHENBURG

BUDAPESTI EÖTVÖS LORÁND TUDOMÁNYEGYETEM

AGH

Università degli Studi di Catania

UNIVERSITY OF OXFORD

The University of Nottingham

University of St Andrews

University of Kent

CHALMERS

KTH VETENSKAP OCH KONST

Erlang History

wanted

Bjarne Däcker’s licentate thesis: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.88.1957

wanted

short time-to-market

Bjarne Däcker's licentate thesis: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.88.1957

wanted

short time-to-market

on-the-fly upgrades

Bjarne Däcker's licentate thesis: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.88.1957

wanted

short time-to-market

on-the-fly upgrades

quality and reliability

and some other stuff…

Bjarne Däcker's licentate thesis: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.88.1957

wanted

productivity

on-the-fly upgrades

quality and reliability

and some other stuff…

Bjarne Däcker's licentate thesis: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.88.1957

wanted

productivity

no downtime

quality and reliability

and some other stuff…

Bjarne Däcker's licentate thesis: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.88.1957

wanted

productivity

no downtime

something that always work

Bjarne Däcker's licentate thesis: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.88.1957

wanted

money

no downtime

something that always work

Bjarne Däcker's licentate thesis: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.88.1957

wanted

money

money

something that always work

Bjarne Däcker's licentate thesis: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.88.1957

wanted

money

money

money

Bjarne Däcker's licentate thesis: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.88.1957

wanted

money

money

money

it’s a rich mans world

If our basic tool, the language in which we design and code our programs, is also complicated, the language itself becomes part of the problem rather than part of its solution.

- C.A.R. Hoare

Good Erlang Domains

Good Erlang Domains

Low latency over throughput

Good Erlang Domains

Low latency over throughput

Stateful (in contrast to being stateless)

Good Erlang Domains

Low latency over throughput

Stateful (in contrast to being stateless)

Massively concurrent

Good Erlang Domains

Low latency over throughput

Stateful (in contrast to being stateless)

Massively concurrent

Distributed

Good Erlang Domains

Low latency over throughput

Stateful (in contrast to being stateless)

Massively concurrent

Distributed

Fault tolerant

Good Erlang Domains

Low latency over throughput

Stateful (in contrast to being stateless)

Massively concurrent

Distributed

Fault tolerant

Uses OTP

Good Erlang Domains

Low latency over throughput

Stateful (in contrast to being stateless)

Massively concurrent

Distributed

Fault tolerant

Uses OTP

Non-stop operation

Good Erlang Domains

Low latency over throughput

Stateful (in contrast to being stateless)

Massively concurrent

Distributed

Fault tolerant

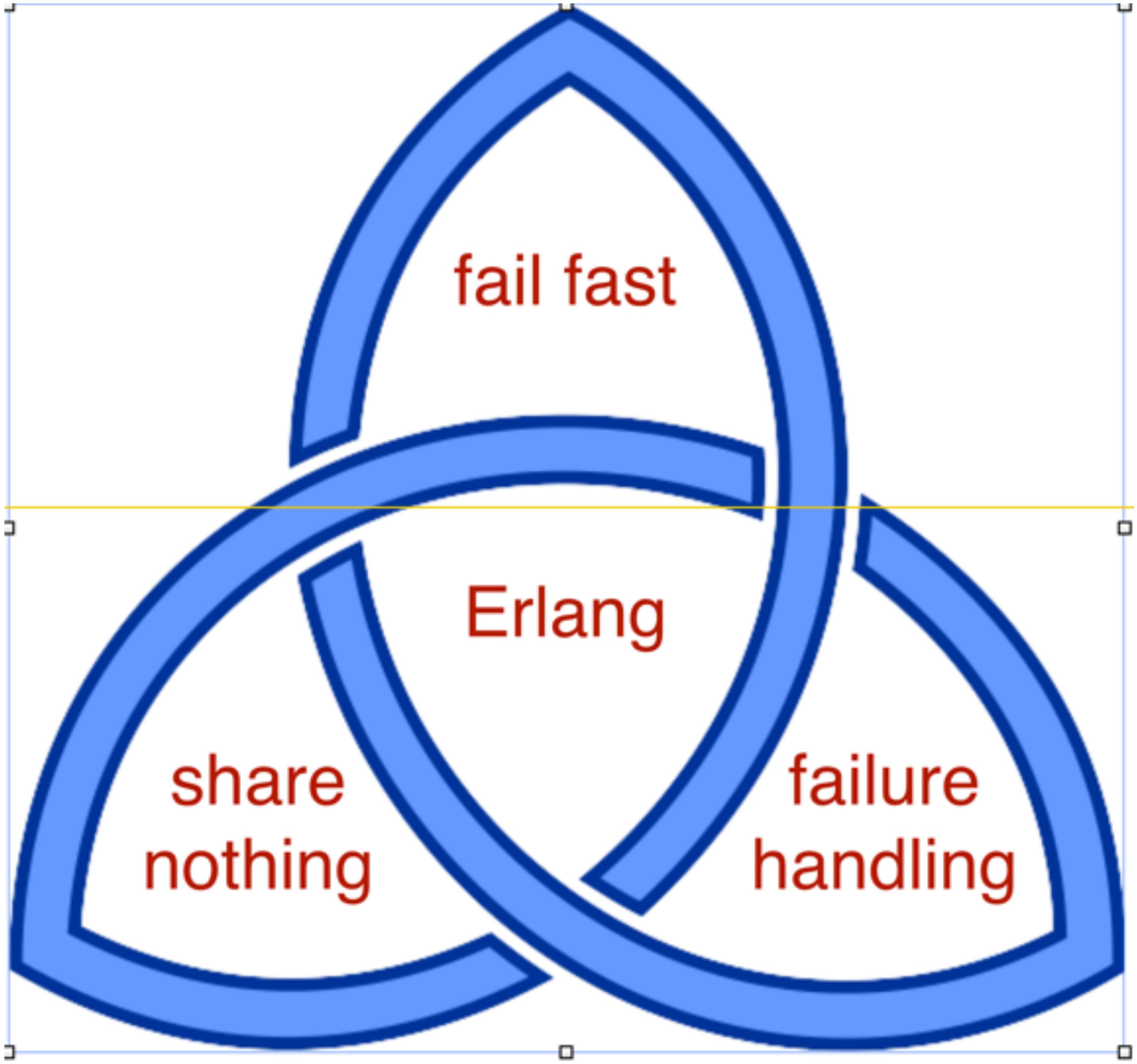
Uses OTP

Non-stop operation

Under load, Erlang programs usually performs as well as programs in other languages, often way better.

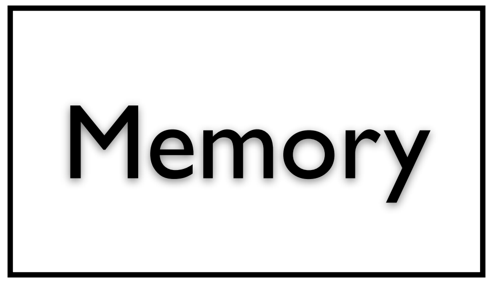
Jesper Louis Andersen

The Golden Trinity Of Erlang

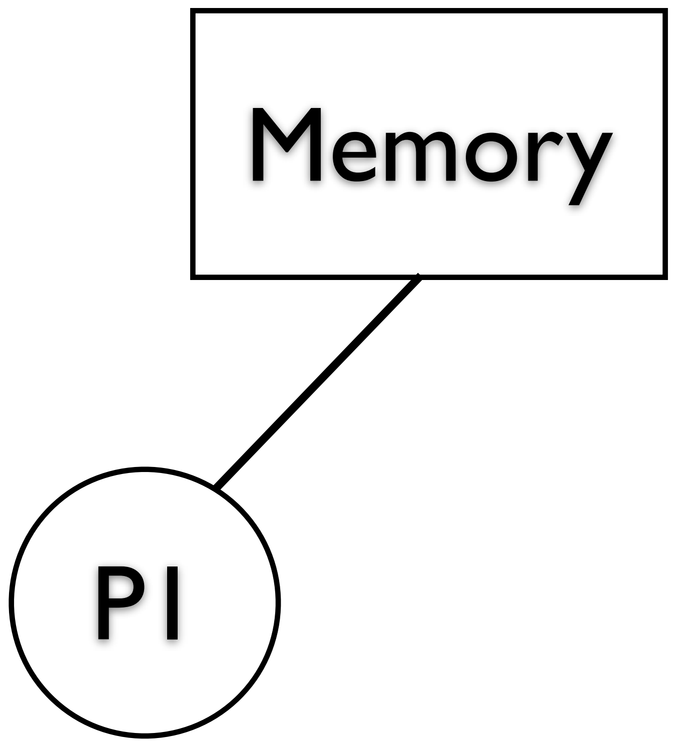


To Share Or Not To Share

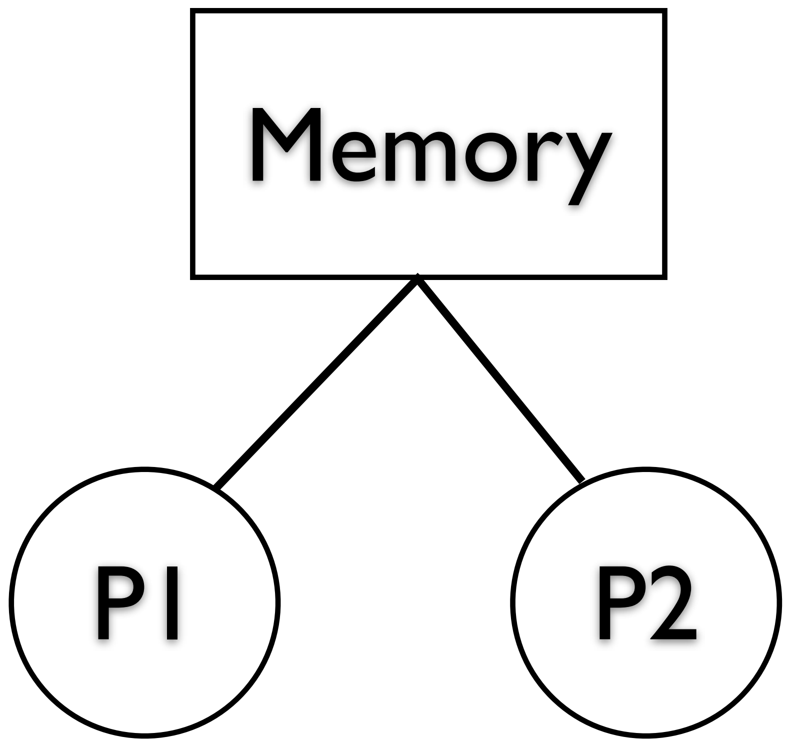
To Share Or Not To Share



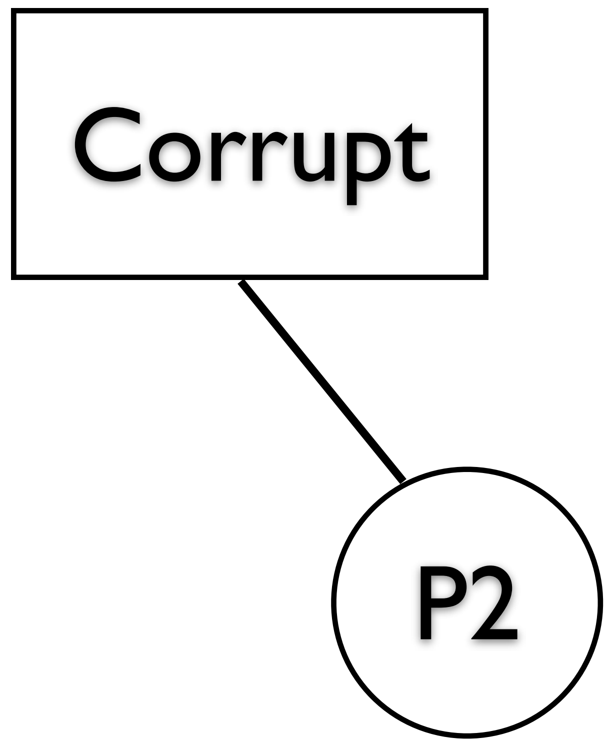
To Share Or Not To Share



To Share Or Not To Share



To Share Or Not To Share



To Share Or Not To Share

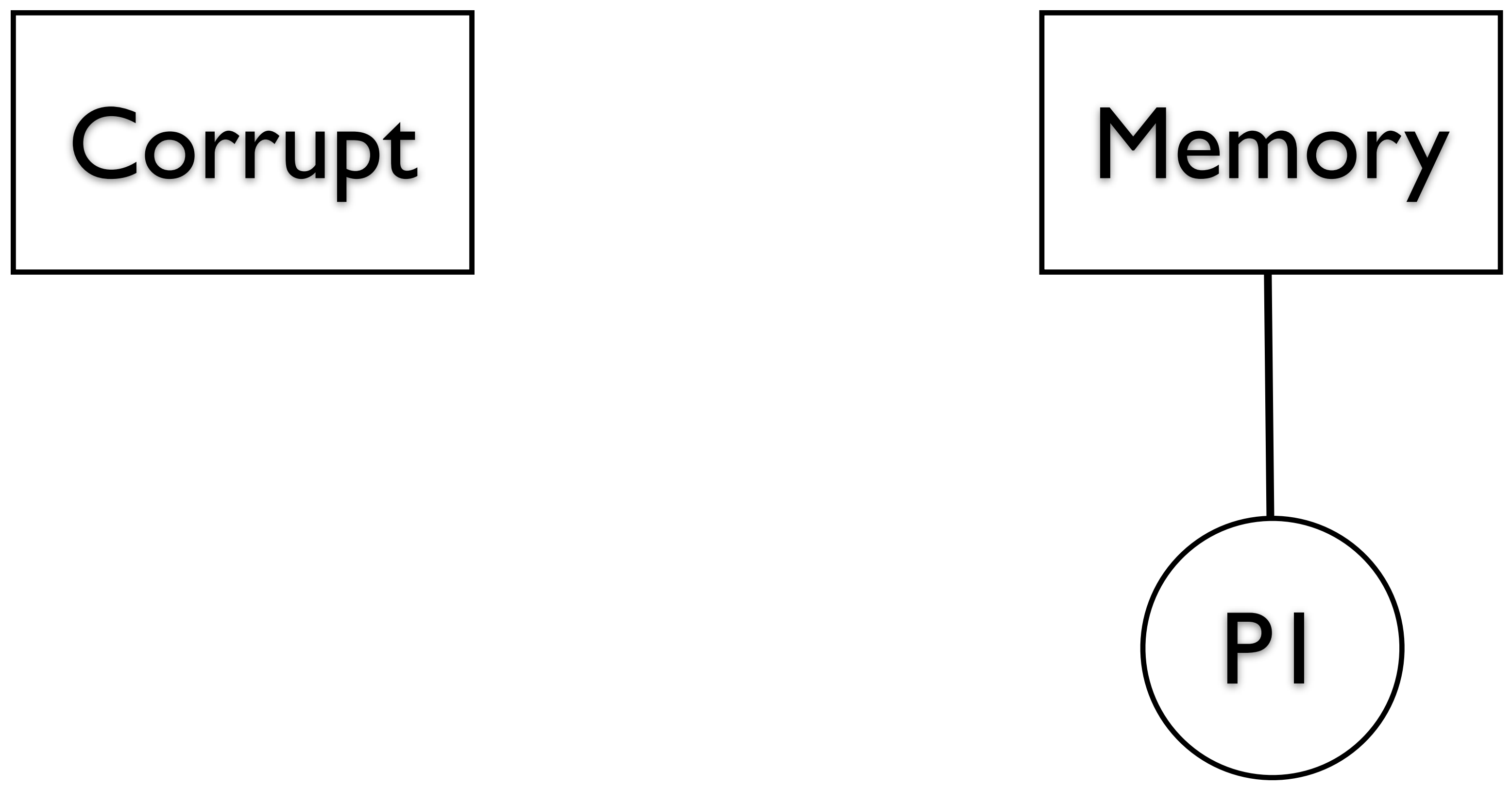
Corrupt

To Share Or Not To Share

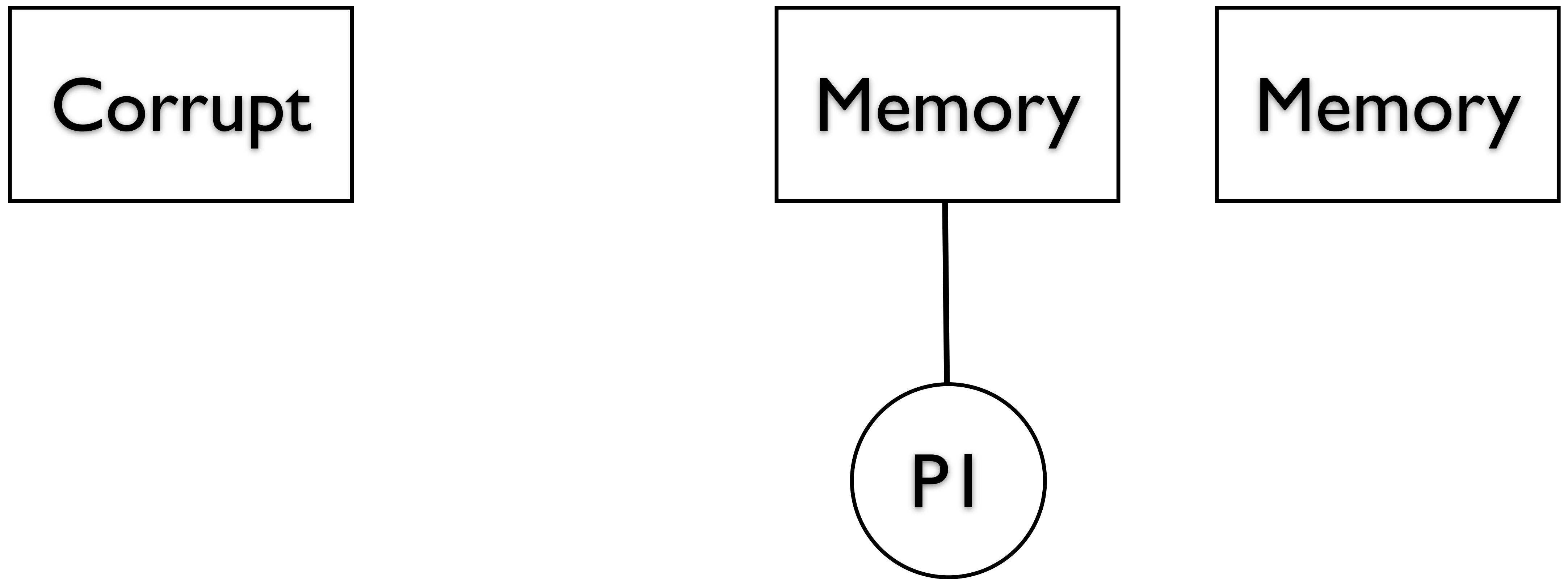
Corrupt

Memory

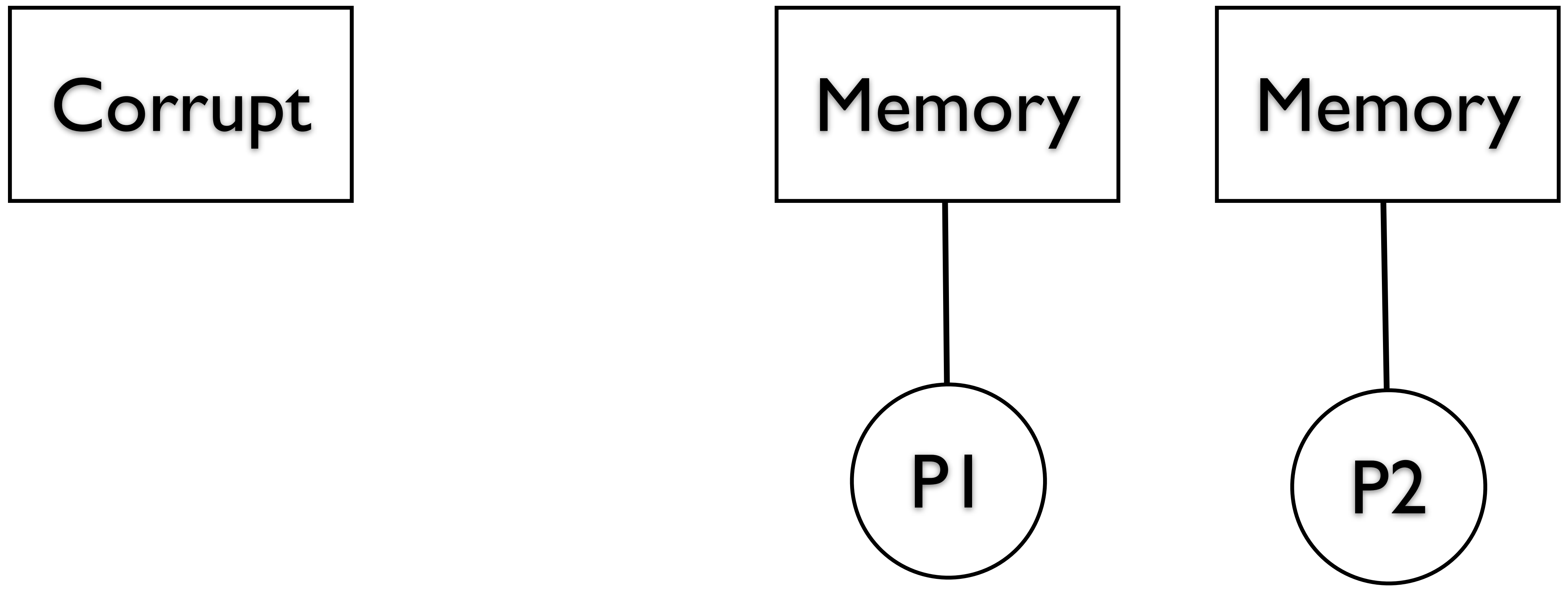
To Share Or Not To Share



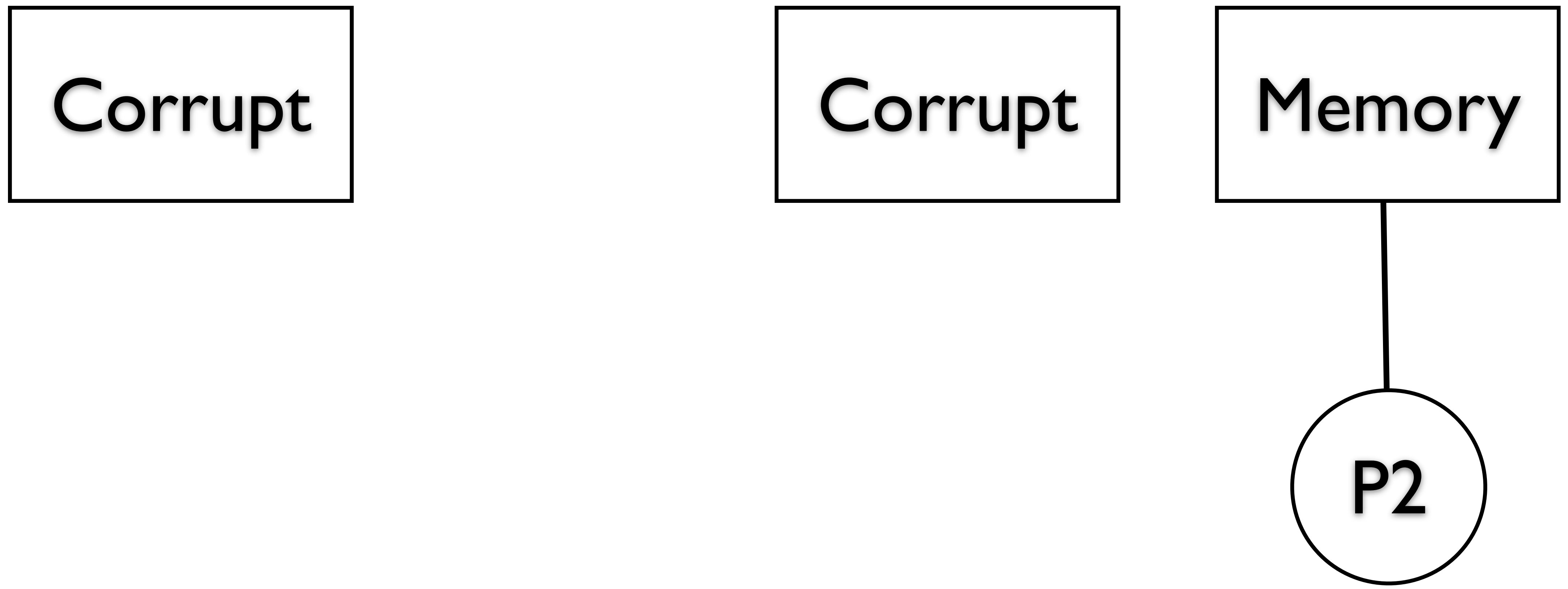
To Share Or Not To Share



To Share Or Not To Share



To Share Or Not To Share



To Share Or Not To Share

Corrupt
Memory
P2

Failures

Anything that can go wrong, will go wrong

Murphy

Failures

Programming errors

Anything that can go wrong, will go wrong

Murphy

Failures

Programming errors
Disk failures

Anything that can go wrong,
will go wrong

Murphy

Failures

Programming errors
Disk failures
Network failures

Anything that can go wrong,
will go wrong

Murphy

Failures

Anything that can go wrong,
will go wrong

Murphy

Programming errors
Disk failures
Network failures

Most programming paradigmes are *fault in-tolerant*

Failures

Anything that can go wrong, will go wrong

Murphy

Programming errors
Disk failures
Network failures

Most programming paradigmes are *fault in-tolerant*
⇒ must deal with all errors or die

Failures

Anything that can go wrong,
will go wrong

Murphy

Programming errors
Disk failures
Network failures

Most programming paradigmes are *fault in-tolerant*
⇒ must deal with all errors or die

Erlang is *fault tolerant* by design

Failures

Anything that can go wrong,
will go wrong

Murphy

Programming errors
Disk failures
Network failures

Most programming paradigmes are *fault in-tolerant*
⇒ must deal with all errors or die

Erlang is *fault tolerant* by design
⇒ failures are embraced and managed

Let It Fail

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

Let It Fail

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

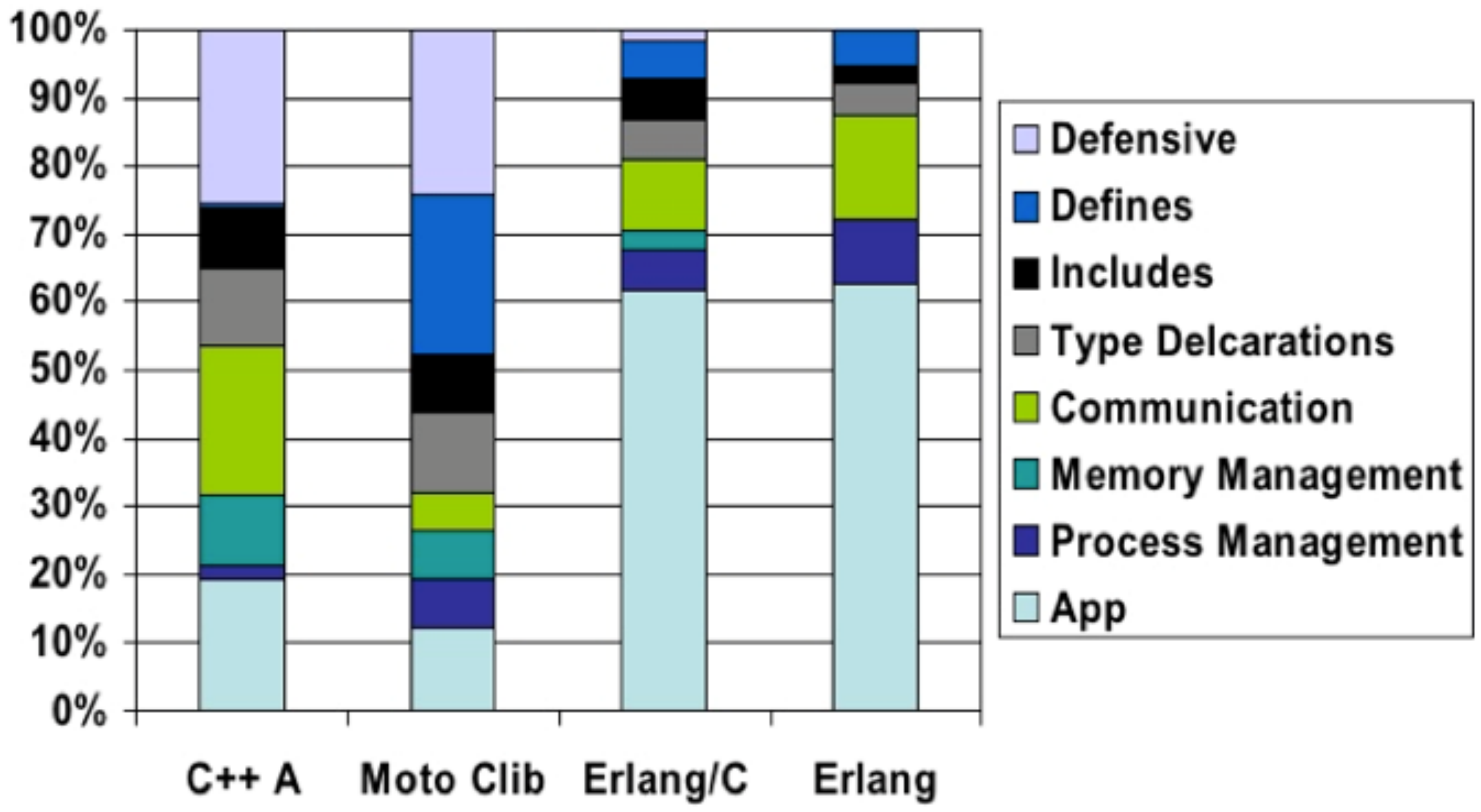
Let It Fail

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

Erlang encourages agressive/offensive programming

Benefits of let-it-fail

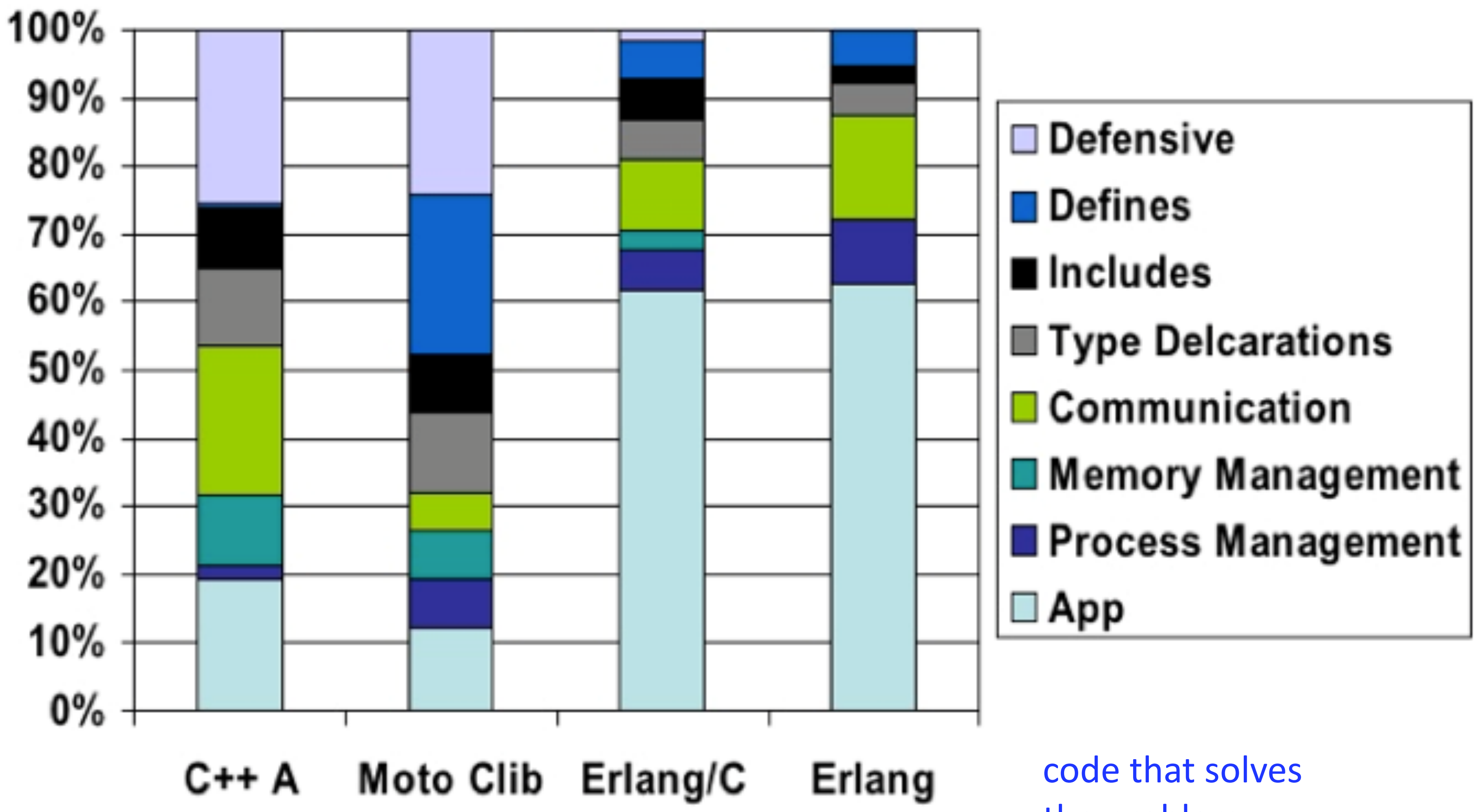
Data Mobility component breakdown



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

Benefits of let-it-fail

Data Mobility component breakdown

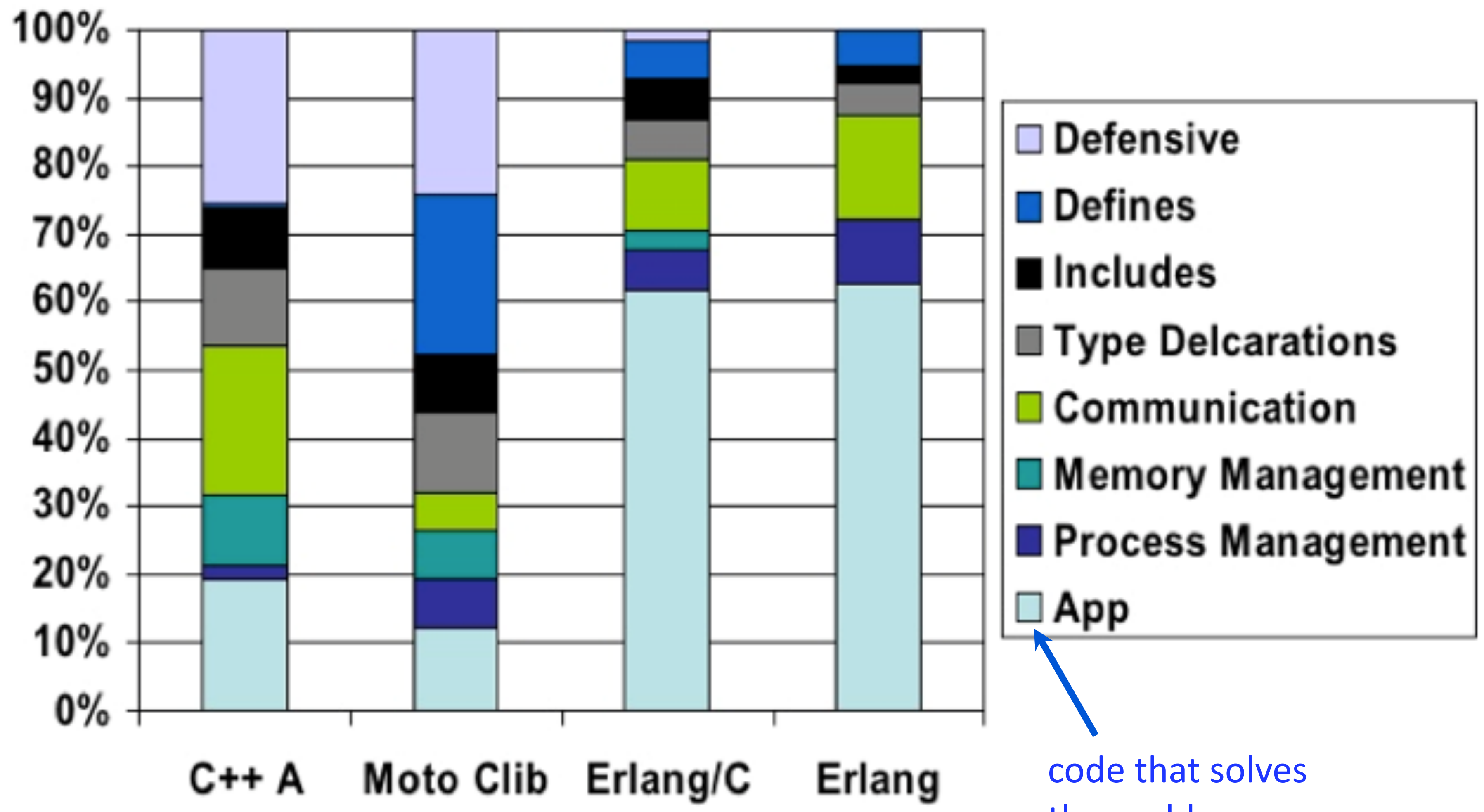


code that solves the problem

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

Benefits of let-it-fail

Data Mobility component breakdown

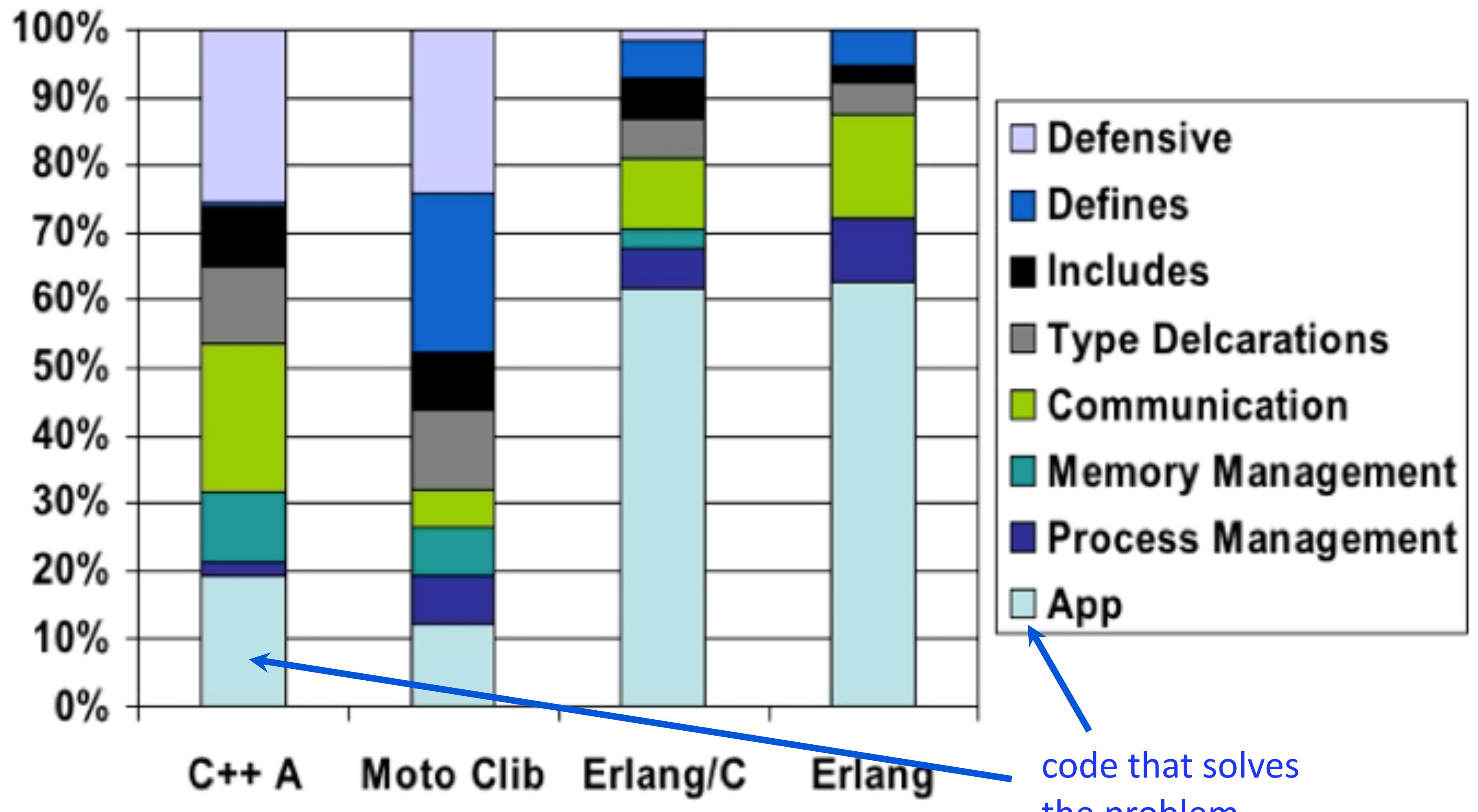


code that solves the problem

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

Benefits of let-it-fail

Data Mobility component breakdown

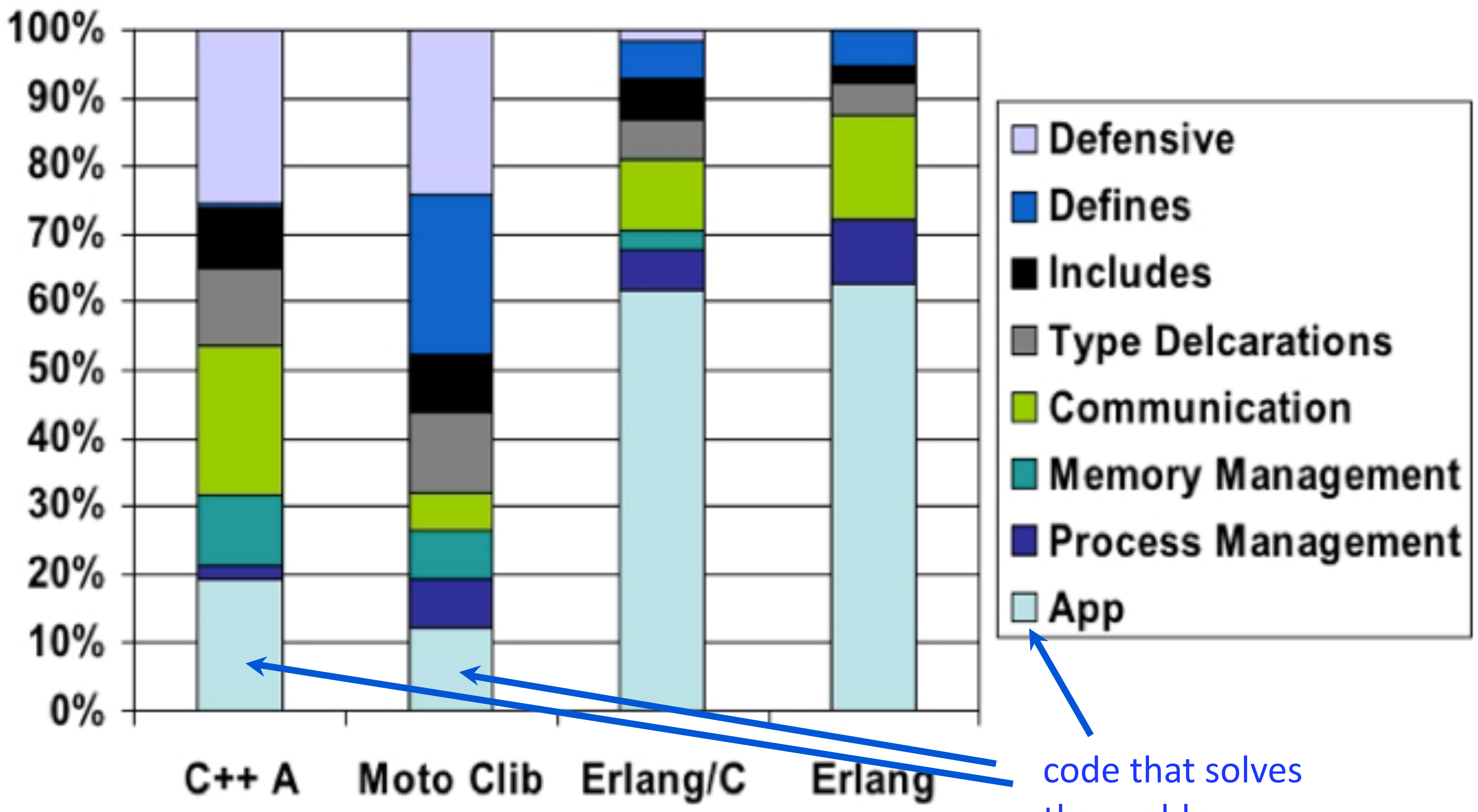


code that solves the problem

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

Benefits of let-it-fail

Data Mobility component breakdown

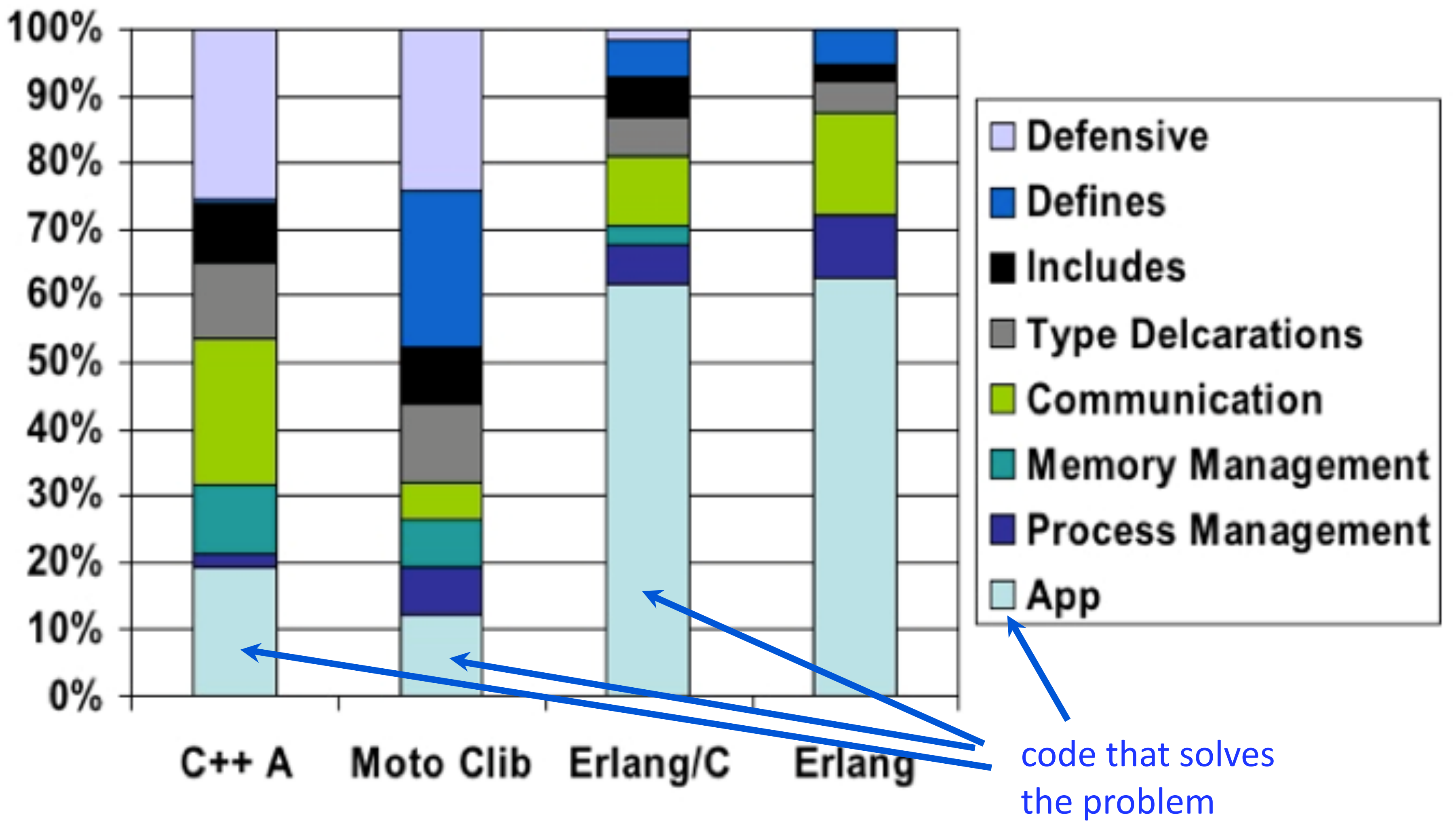


code that solves
the problem

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

Benefits of let-it-fail

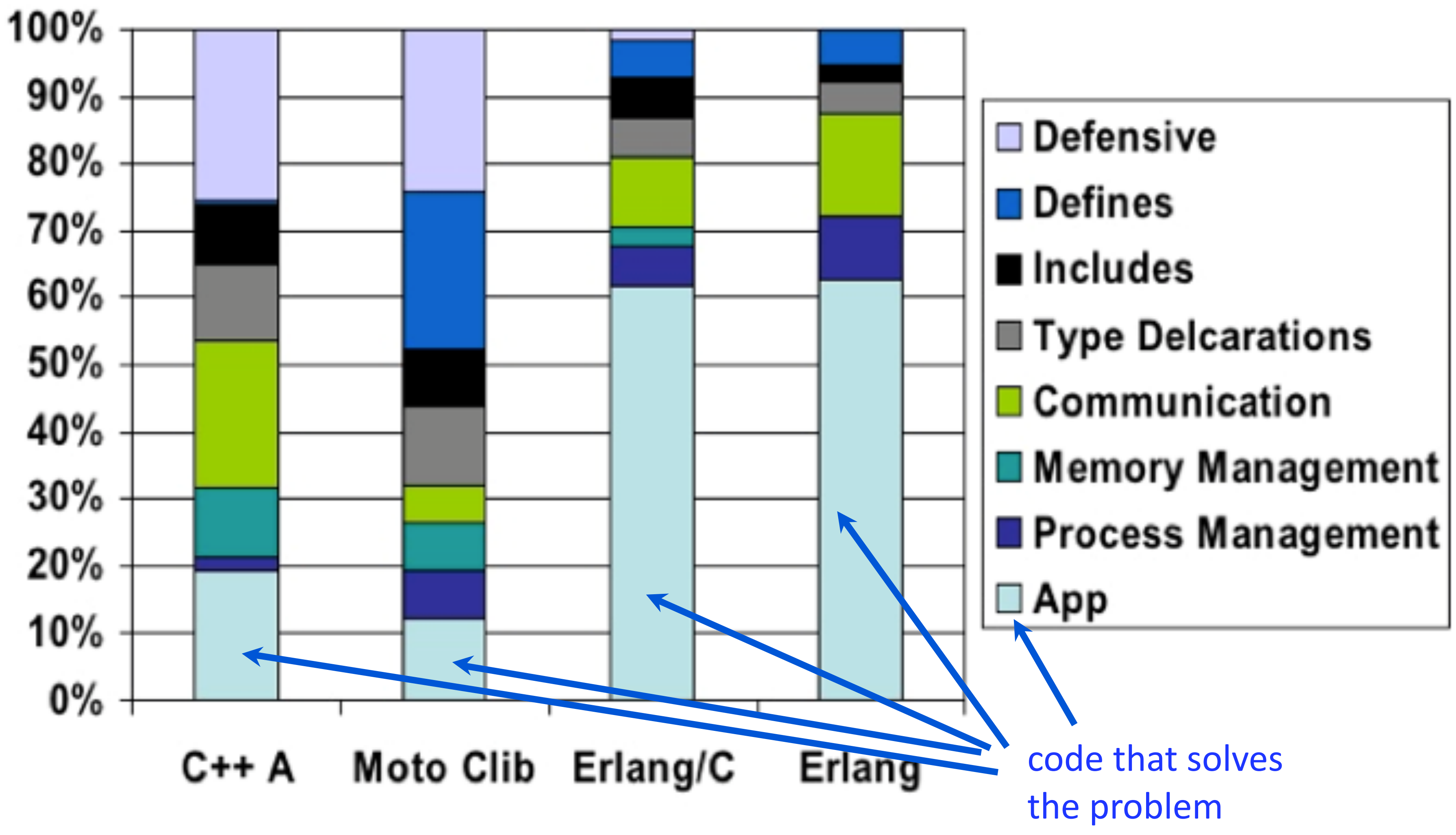
Data Mobility component breakdown



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

Benefits of let-it-fail

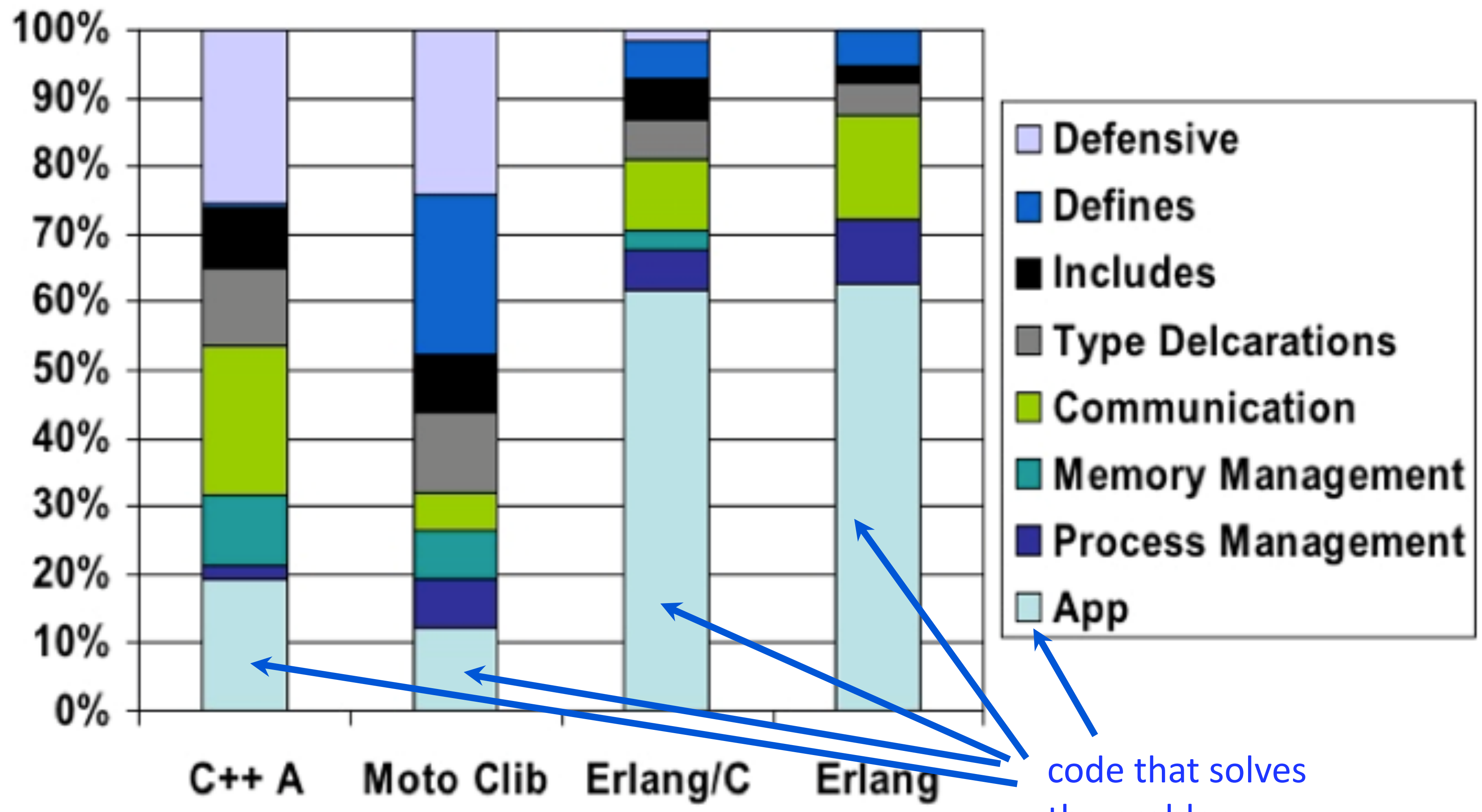
Data Mobility component breakdown



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

Benefits of let-it-fail

Data Mobility component breakdown

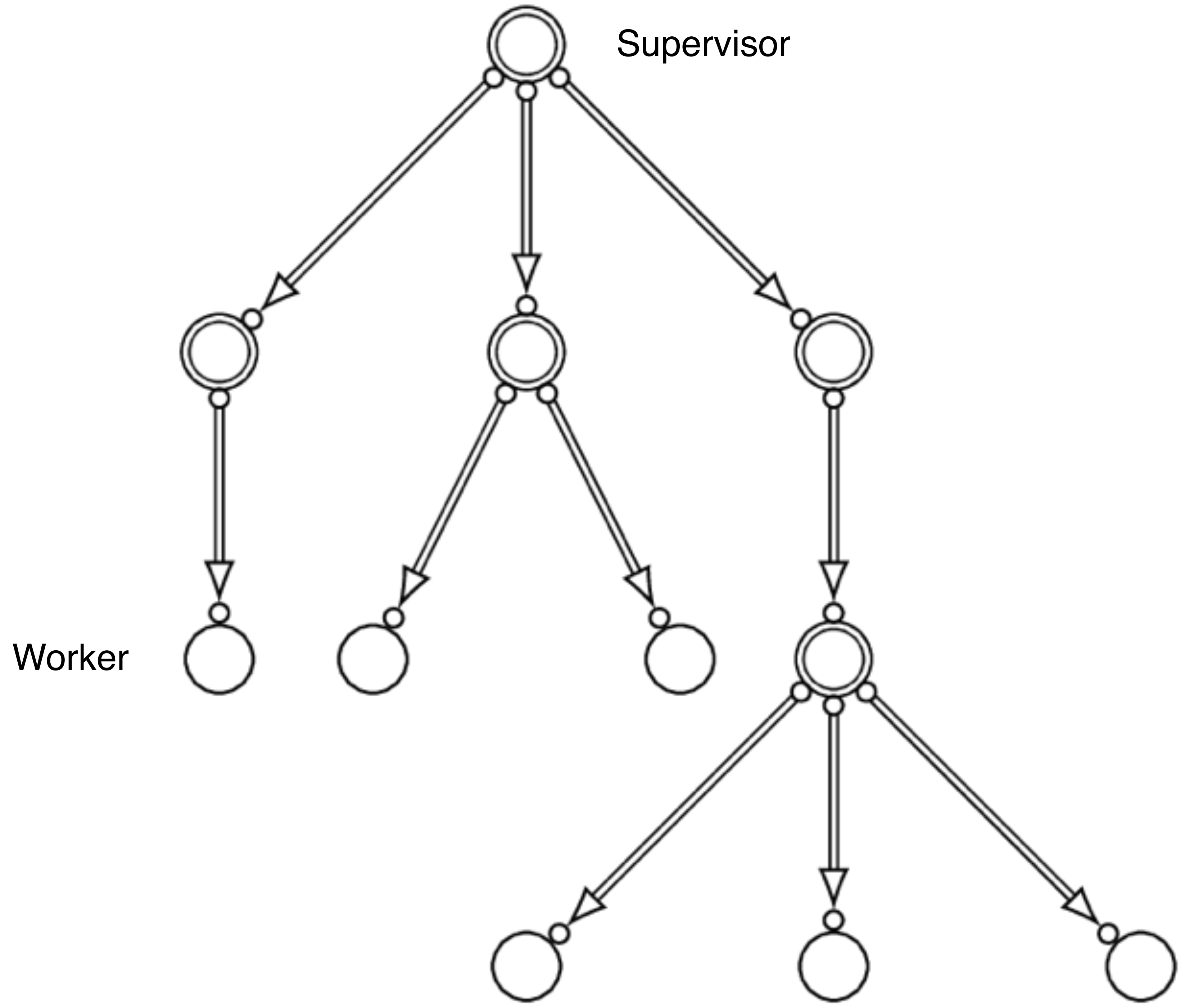


code that solves the problem

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

Erlang @ 3x

Failure Handling with Supervisors



Business benefits of supervisors

Business benefits of supervisors

Only one process dies

Business benefits of supervisors

Only one process dies

- isolation gives continuous service

Business benefits of supervisors

Only one process dies

- isolation gives continuous service

Everything is logged

Business benefits of supervisors

Only one process dies

- isolation gives continuous service

Everything is logged

- you know what is wrong

Business benefits of supervisors

Only one process dies

    isolation gives continuous service

Everything is logged

    you know what is wrong

Corner cases can be fixed at leisure

Business benefits of supervisors

Only one process dies

  isolation gives continuous service

Everything is logged

  you know what is wrong

Corner cases can be fixed at leisure

  Product owner in charge!

Business benefits of supervisors

Only one process dies

  isolation gives continuous service

Everything is logged

  you know what is wrong

Corner cases can be fixed at leisure

  Product owner in charge!

  Not the software!

Business benefits of supervisors

Only one process dies

  isolation gives continuous service

Everything is logged

  you know what is wrong

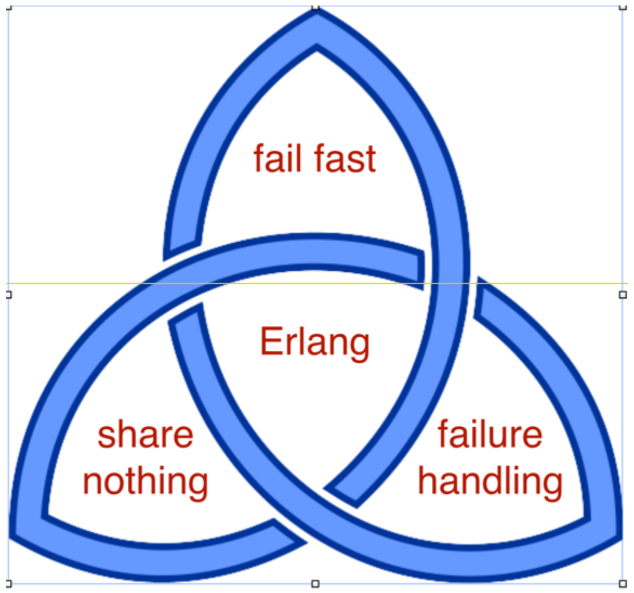
Corner cases can be fixed at leisure

  Product owner in charge!

  Not the software!

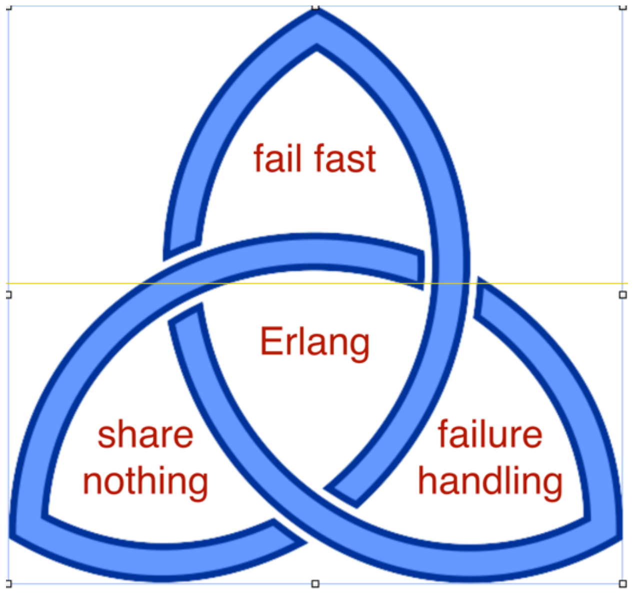
Software architecture that supports iterative development

Cruising with Erlang



Cruising with Erlang

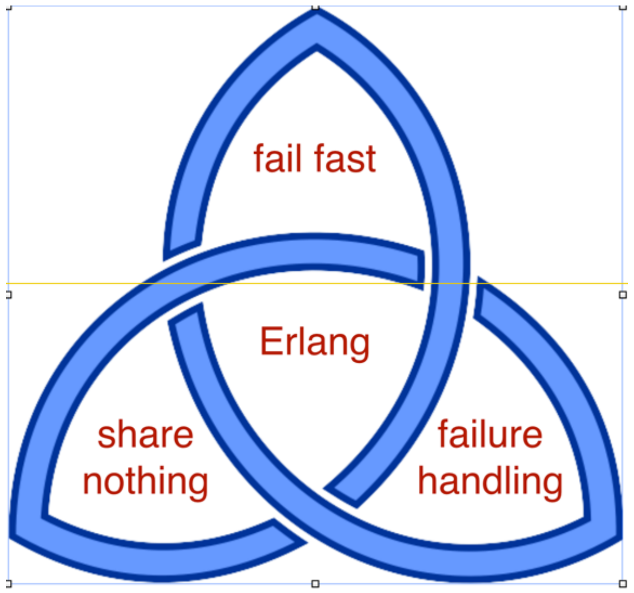
Understand the failure model



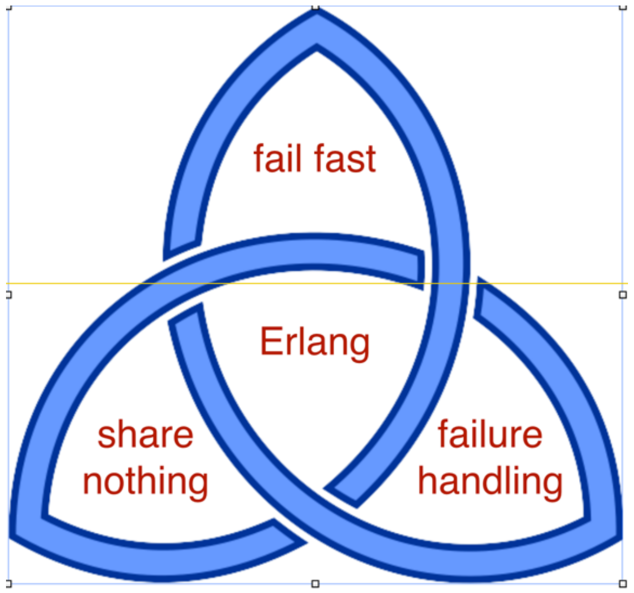
Cruising with Erlang

Understand the failure model

Embrace failure!



Cruising with Erlang

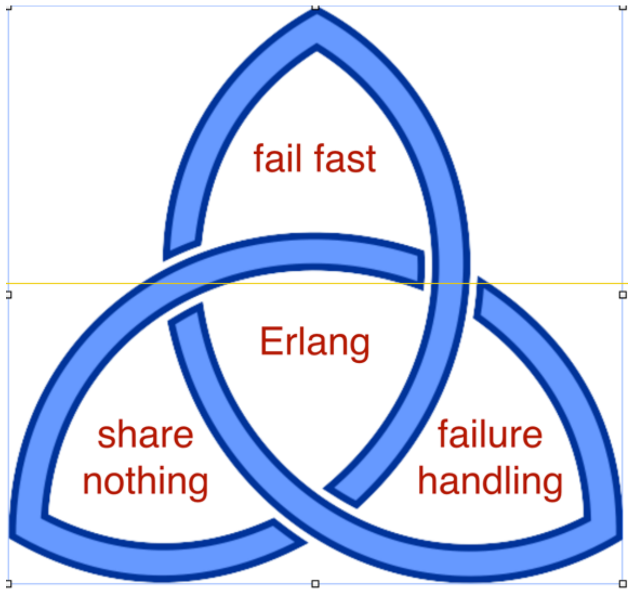


Understand the failure model

Embrace failure!

Use patterns to deliver business value

Cruising with Erlang



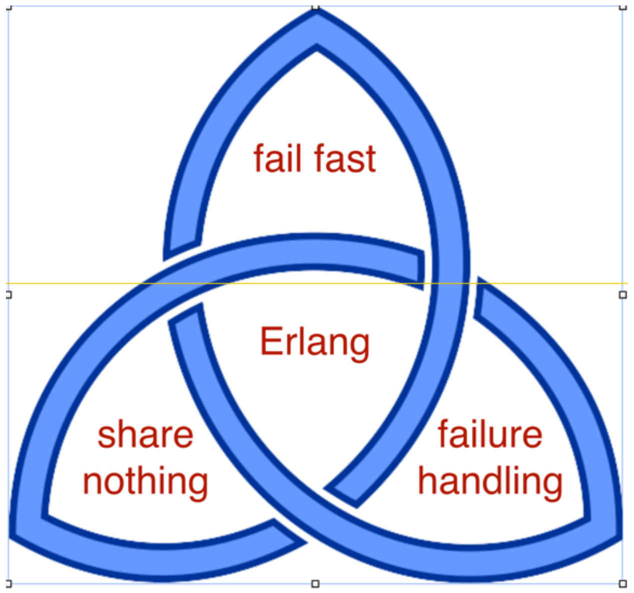
Understand the failure model

Embrace failure!

Use patterns to deliver business value

Stay in charge!

Cruising with Erlang



Understand the failure model

Embrace failure!

Use patterns to deliver business value

Stay in charge!