Erlang
The Driver behind WhatsApp's Success

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

Erlang SOLUTIONS

Background

Background

Erlanger since 2006

Background

Erlanger since 2006

Happyness

Background

Erlanger since 2006

Happyness

Mission critical gateway for Tetra

Background

Erlanger since 2006

Happyness

Mission critical gateway for Tetra

Hard work

Background

Erlanger since 2006

Happyness

Mission critical gateway for Tetra

Hard work

Major learnings

Background

Erlanger since 2006

Happyness

Mission critical gateway for Tetra

Hard work

Major learnings

Why this talk?

Why this talk?

Explain the tech foundation for WhatsApp's succes

Why this talk?

Explain the tech foundation for WhatsApp's succes

Understand the design decisions that makes Erlang unique

Why this talk?

Explain the tech foundation for WhatsApp's succes

Understand the design decisions that makes Erlang unique

Show how Erlang's features delivers business value

Why this talk?

Explain the tech foundation for WhatsApp's succes

Understand the design decisions that makes Erlang unique

Show how Erlang's features delivers business value

Spread the Erlang love

Religious Connection

Er Lang Shen - Chinese God w/ a 3rd truth seeing eye

source:http://www.taoistsecret.com/taoistgod.html

Erlang Temple in Zunhua

source: http://www.tripadvisor.com/Attraction_Review-g1152320-d1799218-Reviews-Erlang_Temple-Zunhua_Hebei.html

Dealing with disbelievers…

Source: http://2.bp.blogspot.com/-qNM3LGTtUYM/UIFLJGd_MLI/AAAAAAAAAnU/GCtI5SYfbCs/s320/orc-army.jpg

source: http://images1.wikia.nocookie.net/__cb20110119125642/villains/images/e/ef/Saruman.jpg

source: http://asset3.cbsistatic.com/cnwk.1d/i/tim2/2013/08/12/Larry_Ellison_Oracle_Open_World_2009_610x407.jpg

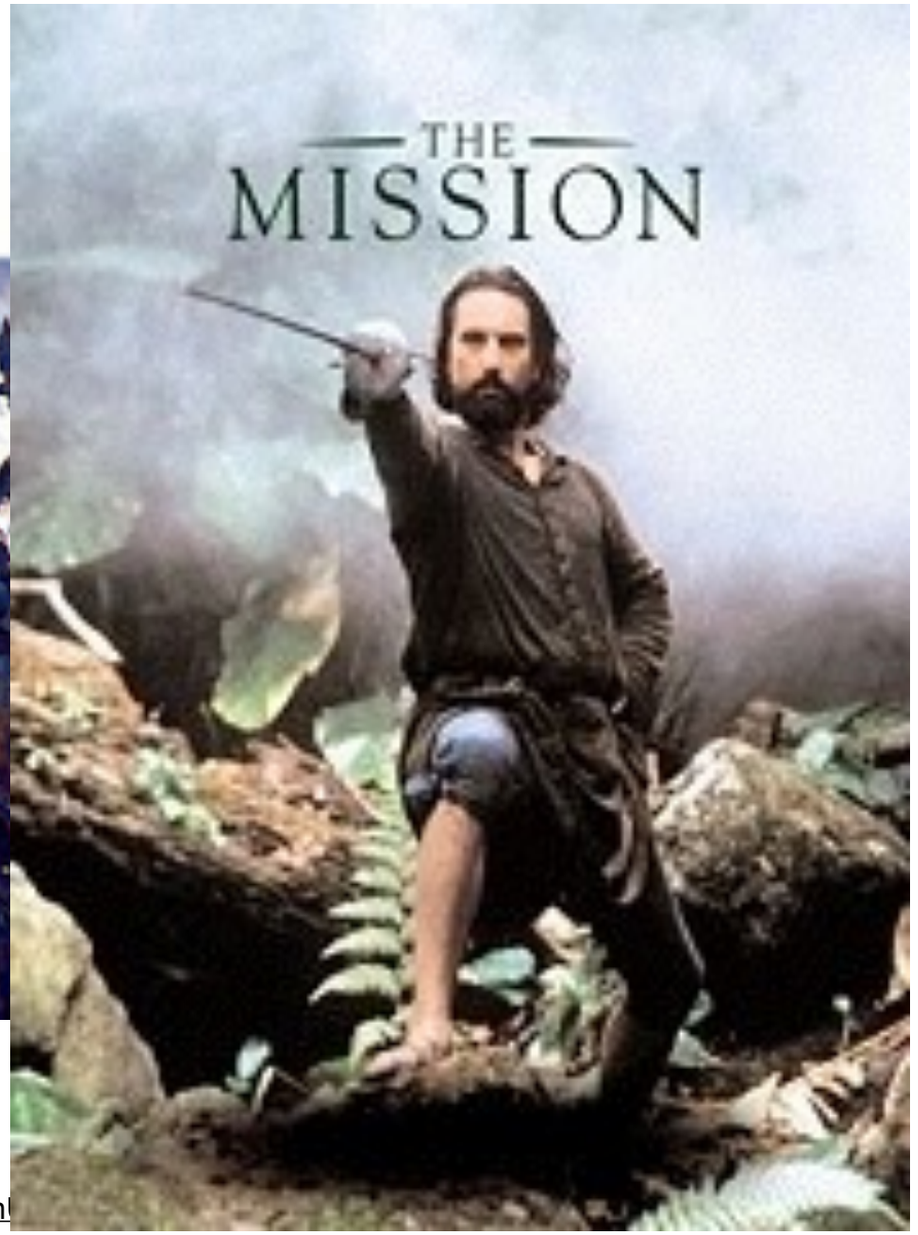
Dealing with disbelievers…

Source: http://2.bp.blogspot.com/-qNM3LGTtUYM/UIFLJGd_MLI/AAAAAAAAAnU/GCtI5SYfbCs/s320/orc-army.jpg

source: http://images1.wikia.nocookie.net/__cb20110119125642/villains/images/e/ef/Saruman.jpg

source: http://asset3.cbsistatic.com/cnwk.1d/i/tim2/2013/08/12/Larry_Ellison_Oracle_Open_World_2009_610x407.jpg

Dealing with disbelievers…



Source: http://2.bp.blogspot.com/-qNM3LGTtUYM/UIFLJGd_MLI/AAAAAAAAAn

source: http://www.rottentomatoes.com/m/1014027-mission/

source: http://images1.wikia.nocookie.net/__cb20110119125642/villains/images/e/ef/Saruman.jpg

source: http://asset3.cbsistatic.com/cnwk.1d/i/tim2/2013/08/12/Larry_Ellison_Oracle_Open_World_2009_610x407.jpg


Erlang
SOLUTIONS

Founded in 1999 soon after Erlang was released as open source

Founded in 1999 soon after Erlang was released as open source

Experts at building bespoke scalable, high availability, high performance systems

Founded in 1999 soon after Erlang was released as open source

Experts at building bespoke scalable, high availability, high performance systems

Only company of its kind totally focused on Erlang and Erlang community

Founded in 1999 soon after Erlang was released as open source

Experts at building bespoke scalable, high availability, high performance systems

Only company of its kind totally focused on Erlang and Erlang community

Over 300 clients.

Founded in 1999 soon after Erlang was released as open source

Experts at building bespoke scalable, high availability, high performance systems

Only company of its kind totally focused on Erlang and Erlang community

Over 300 clients.

Headquartered in London, U.K.

Founded in 1999 soon after Erlang was released as open source

Experts at building bespoke scalable, high availability, high performance systems

Only company of its kind totally focused on Erlang and Erlang community

Over 300 clients.

Headquartered in London, U.K.

- Offices: Stockholm, Krakow, Copenhagen, Aarhus, Budapest, Seattle and Zurich

Founded in 1999 soon after Erlang was released as open source

Experts at building bespoke scalable, high availability, high performance systems

Only company of its kind totally focused on Erlang and Erlang community

Over 300 clients.

Headquartered in London, U.K.

- Offices: Stockholm, Krakow, Copenhagen, Aarhus, Budapest, Seattle and Zurich

Organically growing and continually investing in R & D

Founded in 1999 soon after Erlang was released as open source

Experts at building bespoke scalable, high availability, high performance systems

Only company of its kind totally focused on Erlang and Erlang community

Over 300 clients.

Headquartered in London, U.K.

- Offices: Stockholm, Krakow, Copenhagen, Aarhus, Budapest, Seattle and Zurich

Organically growing and continually investing in R & D

Core focus

Core focus

Speed to market

Core focus

Speed to market

Low lifetime cost

Erlang SOLUTIONS

Core focus

Speed to market

Low lifetime cost

Extreme reliability

Erlang SOLUTIONS

Core focus

Speed to market

Low lifetime cost

Extreme reliability

True scalability

Erlang SOLUTIONS

Core focus

Speed to market

Low lifetime cost

Extreme reliability

True scalability

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

Some Erlang SOLUTIONS Customers

T-Mobile

ERICSSON

cellfind

VOCALINK
safer payments, smarter partner

WhatsApp

CASSIDIAN

Lavelle Energy

medical objects

E*TRADE

basho

oovoo

Some Erlang SOLUTIONS Customers

19,000,000,000 reasons to use Erlang

WhatsApp

Real-time Messaging

Text and Pictures

Group chat

WhatsApp Numbers

WhatsApp Numbers

10 Erlang engineers

WhatsApp Numbers

10 Erlang engineers

~500M monthly users

WhatsApp Numbers

10 Erlang engineers

~500M monthly users

19B msg/day in / 40B msg/day out

WhatsApp Numbers

10 Erlang engineers

~500M monthly users

19B msg/day in / 40B msg/day out

147M concurrent connections

WhatsApp Numbers

10 Erlang engineers

~500M monthly users

19B msg/day in / 40B msg/day out

147M concurrent connections

peak: 324K msg/s in / 712K msg/s out

WhatsApp Numbers

10 Erlang engineers

~500M monthly users

19B msg/day in / 40B msg/day out

147M concurrent connections

peak: 324K msg/s in / 712K msg/s out

WhatsApp Hardware

~550 servers

2x2690v2 Ivy Bridge 10-core (40 threads total)

64-512 GB RAM

SSD

>11,000 cores

WhatsApp Software

FreeBSD 9.2

Erlang R16B01 (with patches)

Other Users of Erlang

Bet365

Bet365

Resarch team Re-did a Java system in Erlang as a POC

Bet365

Resarch team Re-did a Java system in Erlang as a POC

Results:

5x connected users in load test

4x rate of data change

Better utilisation of CPU resources

Bet365

Resarch team Re-did a Java system in Erlang as a POC

Results:

- 5x connected users in load test
- 4x rate of data change
- Better utilisation of CPU resources

How to convince the developers to switch?

Bet365 showed that Erlang…

https://www.erlang-solutions.com/resources/webinars/webinar-recording-erlang-gamblingonline-betting

Bet365 showed that Erlang…

makes programming fun

https://www.erlang-solutions.com/resources/webinars/webinar-recording-erlang-gamblingonline-betting

Bet365 showed that Erlang…

makes programming fun

scales and is reliable

https://www.erlang-solutions.com/resources/webinars/webinar-recording-erlang-gamblingonline-betting

Bet365 showed that Erlang…

makes programming fun

scales and is reliable

has enough depth to be interesting

https://www.erlang-solutions.com/resources/webinars/webinar-recording-erlang-gamblingonline-betting

Bet365 showed that Erlang…

makes programming fun

scales and is reliable

has enough depth to be interesting

solves difficult problems with simple code

https://www.erlang-solutions.com/resources/webinars/webinar-recording-erlang-gamblingonline-betting

Bet365 showed that Erlang…

makes programming fun

scales and is reliable

has enough depth to be interesting

solves difficult problems with simple code

Results:

https://www.erlang-solutions.com/resources/webinars/webinar-recording-erlang-gamblingonline-betting

Bet365 showed that Erlang…

makes programming fun

scales and is reliable

has enough depth to be interesting

solves difficult problems with simple code

Results:

production teams quickly started to appreciate the benefits of Erlang

https://www.erlang-solutions.com/resources/webinars/webinar-recording-erlang-gamblingonline-betting

Bet365 showed that Erlang…

makes programming fun

scales and is reliable

has enough depth to be interesting

solves difficult problems with simple code

Results:

- production teams quickly started to appreciate the benefits of Erlang
- did not want to go back to Java

https://www.erlang-solutions.com/resources/webinars/webinar-recording-erlang-gamblingonline-betting

Bet365 showed that Erlang…

makes programming fun

scales and is reliable

has enough depth to be interesting

solves difficult problems with simple code

Results:

- production teams quickly started to appreciate the benefits of Erlang
- did not want to go back to Java

https://www.erlang-solutions.com/resources/webinars/webinar-recording-erlang-gamblingonline-betting

Riak from Basho

Distributed NoSQL database

- Dynamo inspired key/value store

Erlang & C/C++

Powers Rovio, Danish Health Services and many more

Erlang History

There are two ways of constructing a software design:

There are two ways of constructing a software design:

One way is to make it so simple that there are *obviously* no deficiencies

There are two ways of constructing a software design:

One way is to make it so simple that there are *obviously* no deficiencies

and the other way is to make it so complicated that there are no *obvious* deficiencies.

There are two ways of constructing a software design:

One way is to make it so simple that there are *obviously* no deficiencies and the other way is to make it so complicated that there are no *obvious* deficiencies.

- C.A.R. Hoare

Erlang's Original Requirements

Erlang's Original Requirements

Large scale concurrency

Erlang's Original Requirements

Large scale concurrency

Soft real-time

Erlang's Original Requirements

Large scale concurrency

Soft real-time

Distributed systems

Erlang's Original Requirements

Large scale concurrency

Soft real-time

Distributed systems

Hardware interaction

Erlang's Original Requirements

Large scale concurrency

Soft real-time

Distributed systems

Hardware interaction

Very large software systems

Erlang's Original Requirements

Large scale concurrency

Soft real-time

Distributed systems

Hardware interaction

Very large software systems

Complex functionality

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'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'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'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

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

wanted

wanted

productivity

wanted

productivity

no down-time

wanted

productivity

no down-time

something that always works

wanted

wanted

money

wanted

money

money

wanted

money

money

money

wanted

money

money

money

it’s a rich man’s world!

wanted

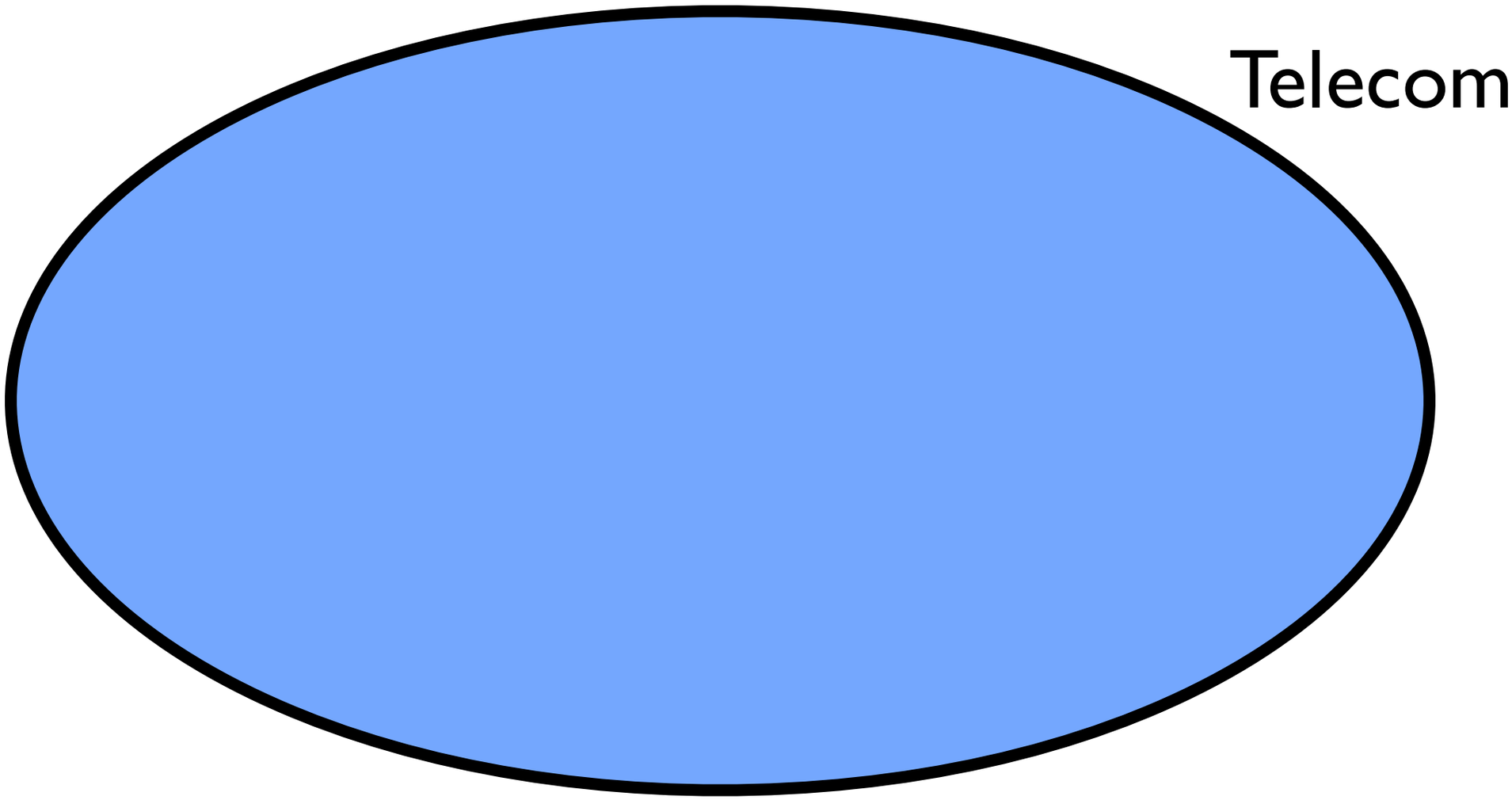
money

money

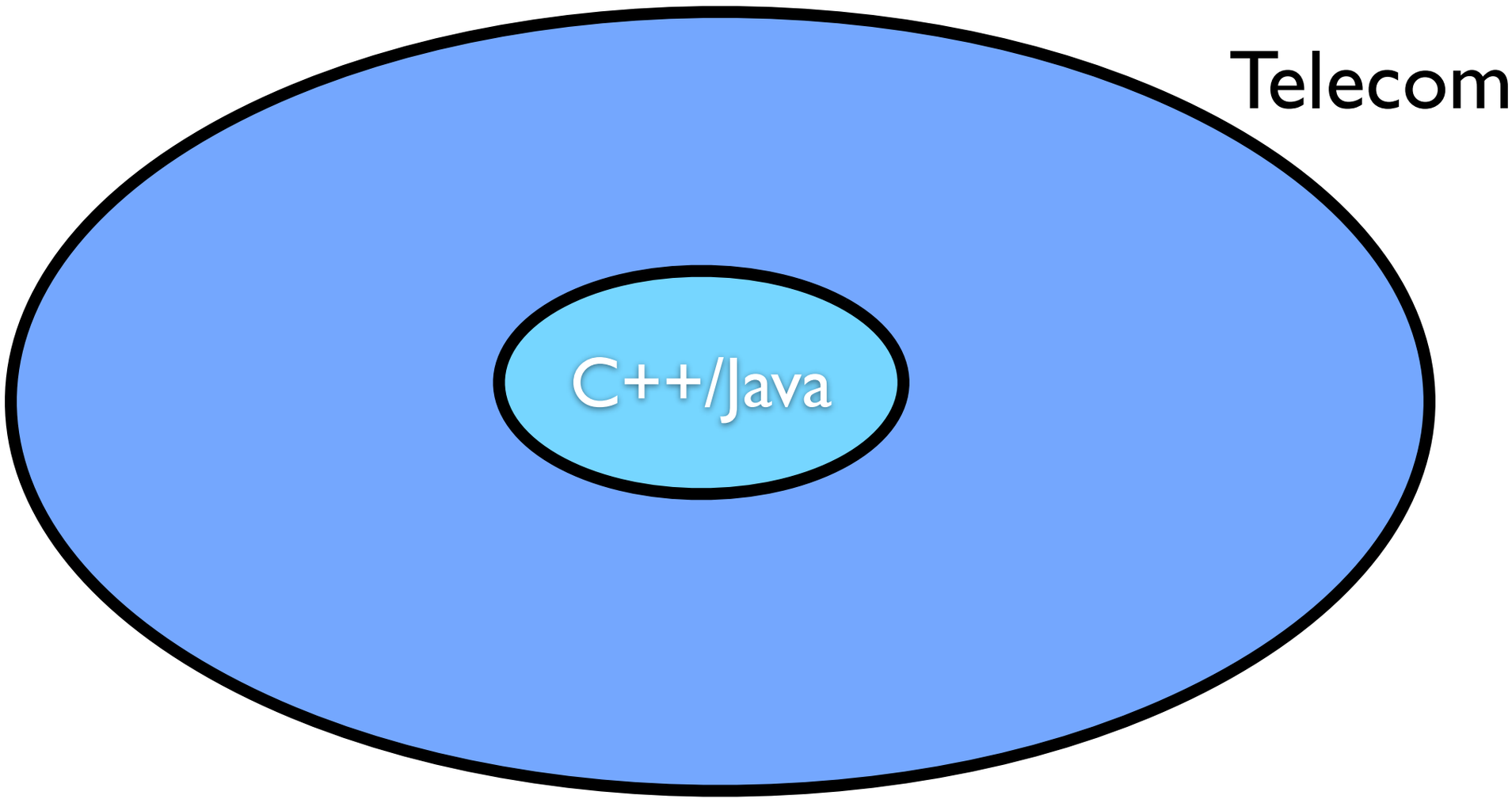
money

it’s a rich man’s world!

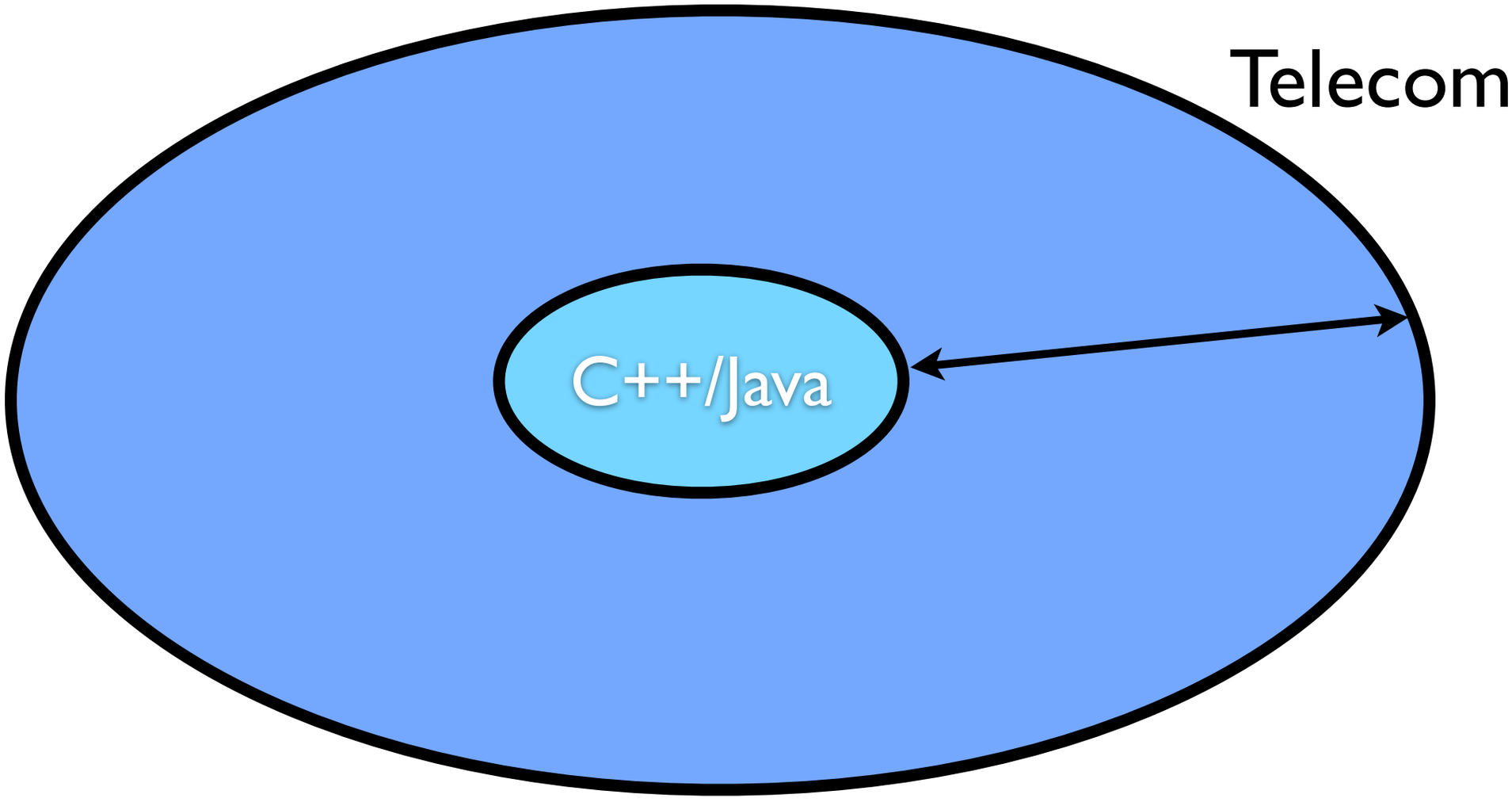
General vs Domain Specific



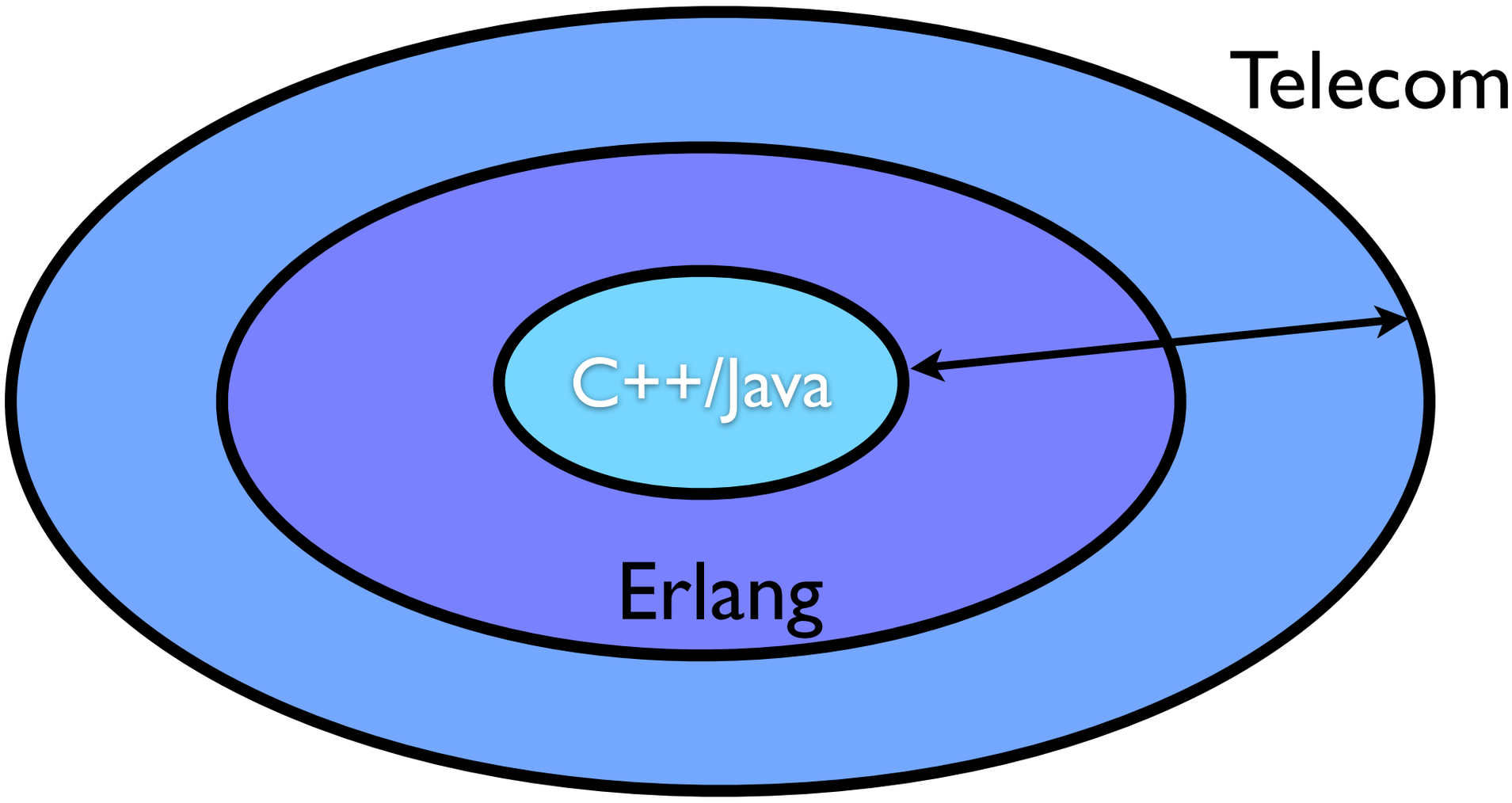
General vs Domain Specific



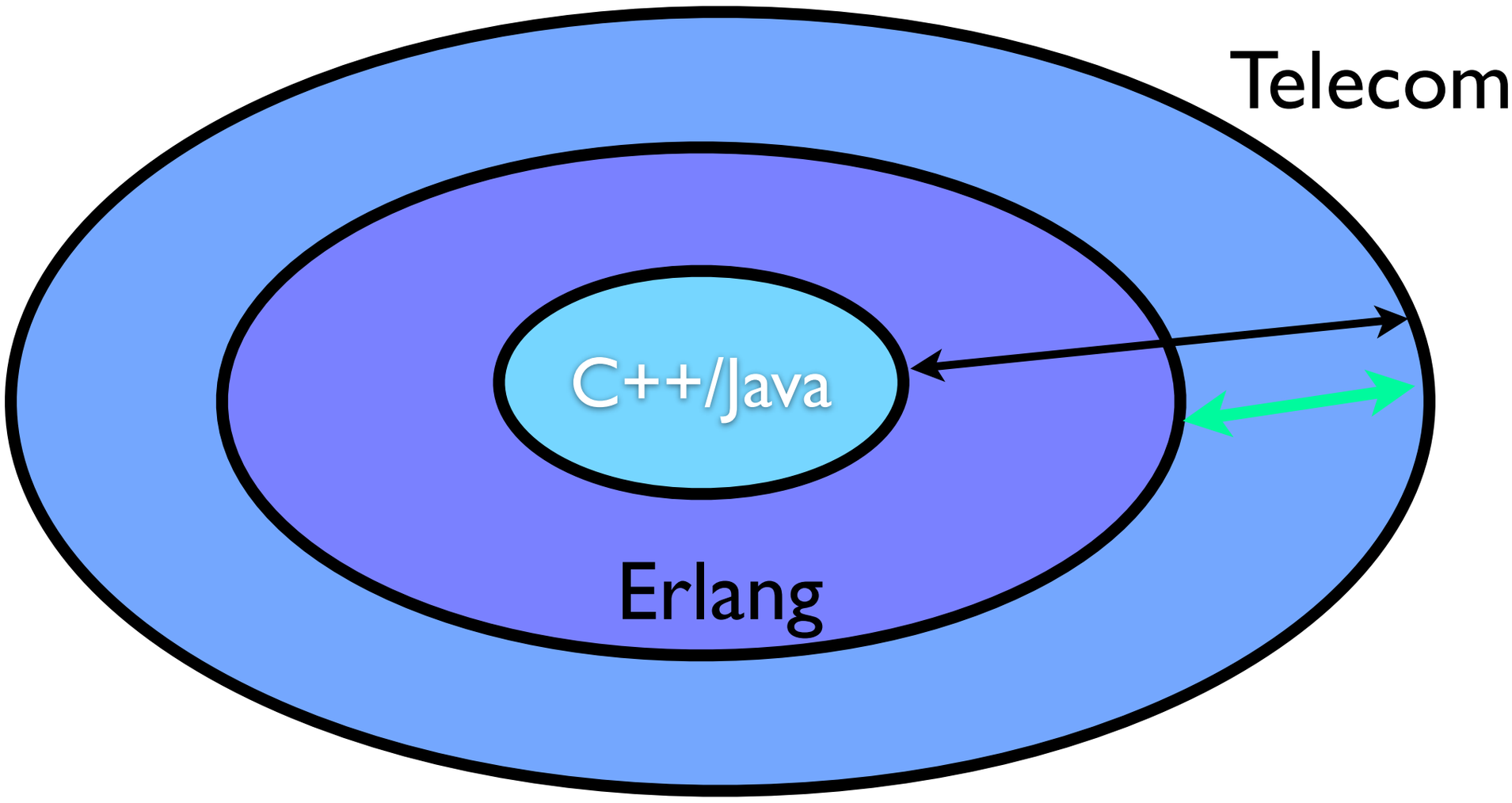
General vs Domain Specific



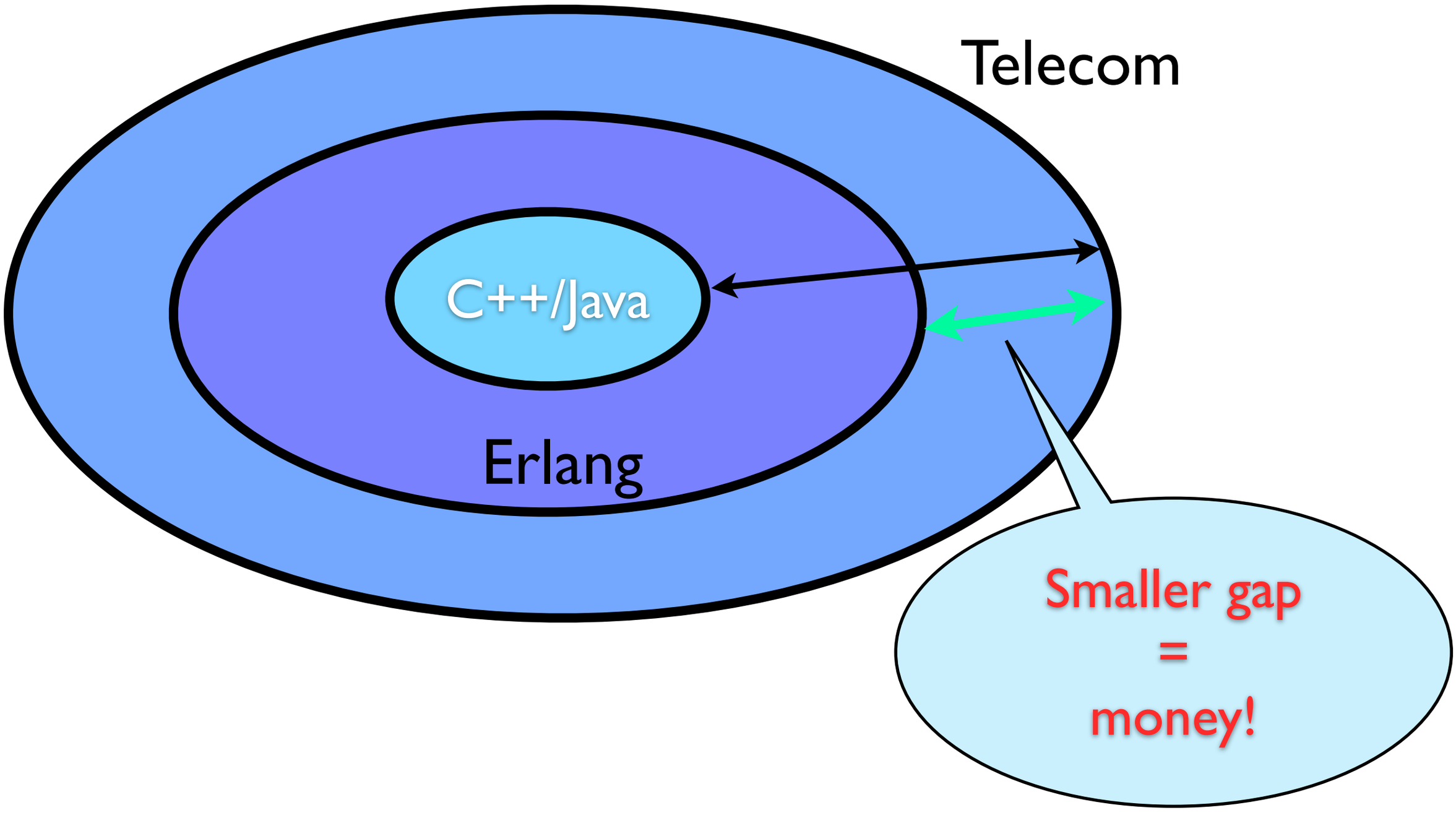
General vs Domain Specific



General vs Domain Specific



General vs Domain Specific



The Sweet Spot

GUI

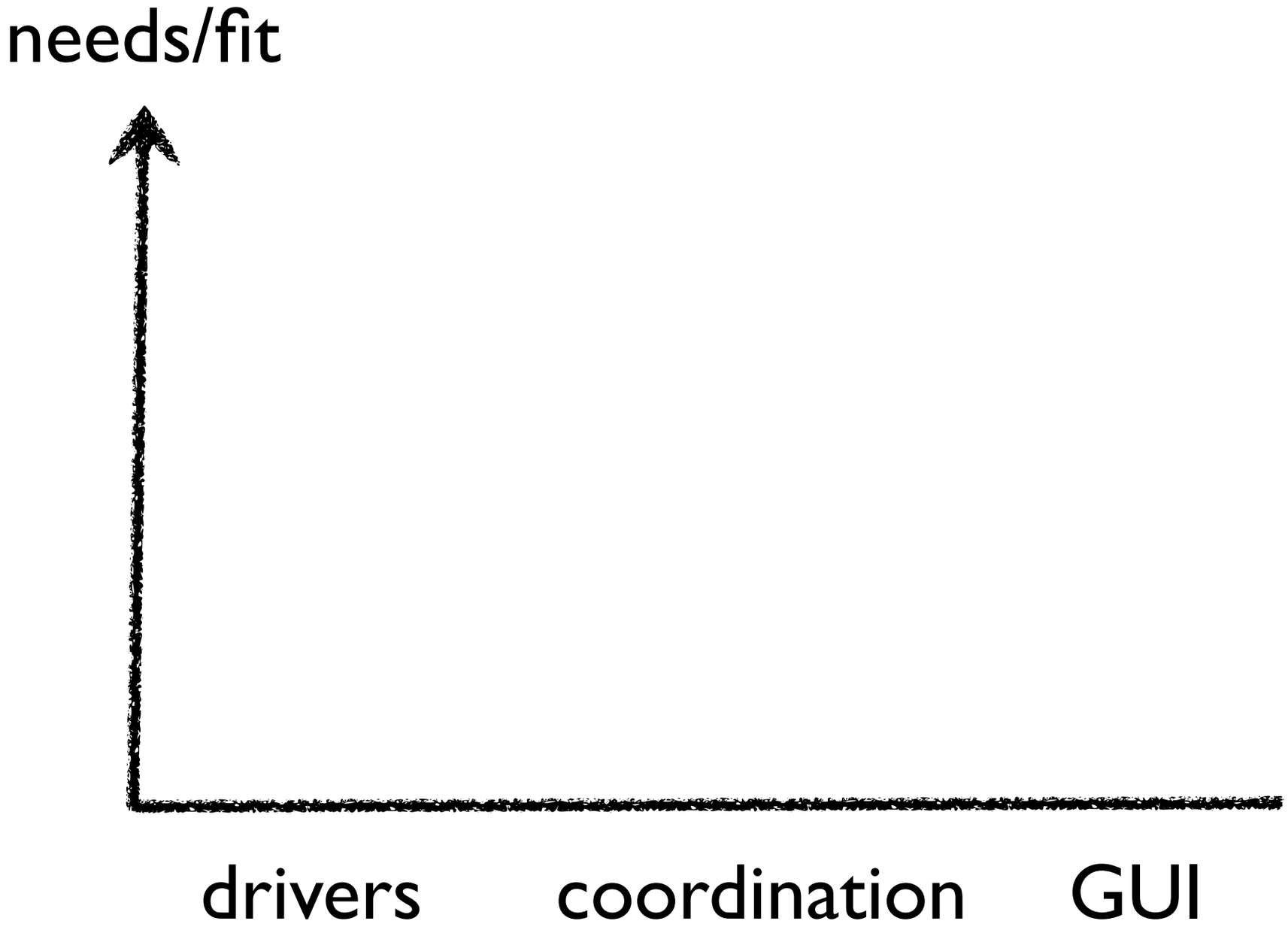
Middleware

Coordination

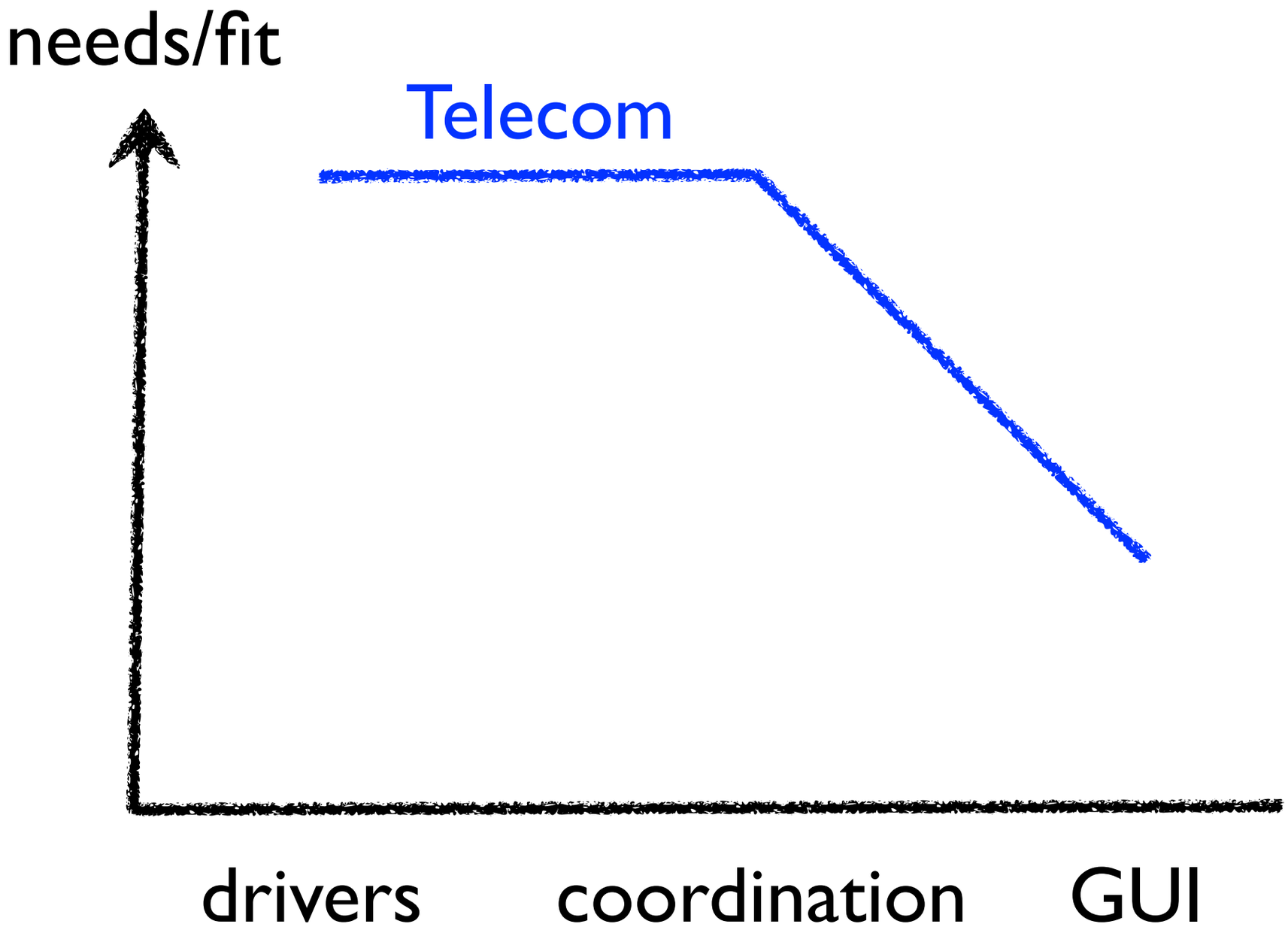
Control

Drivers

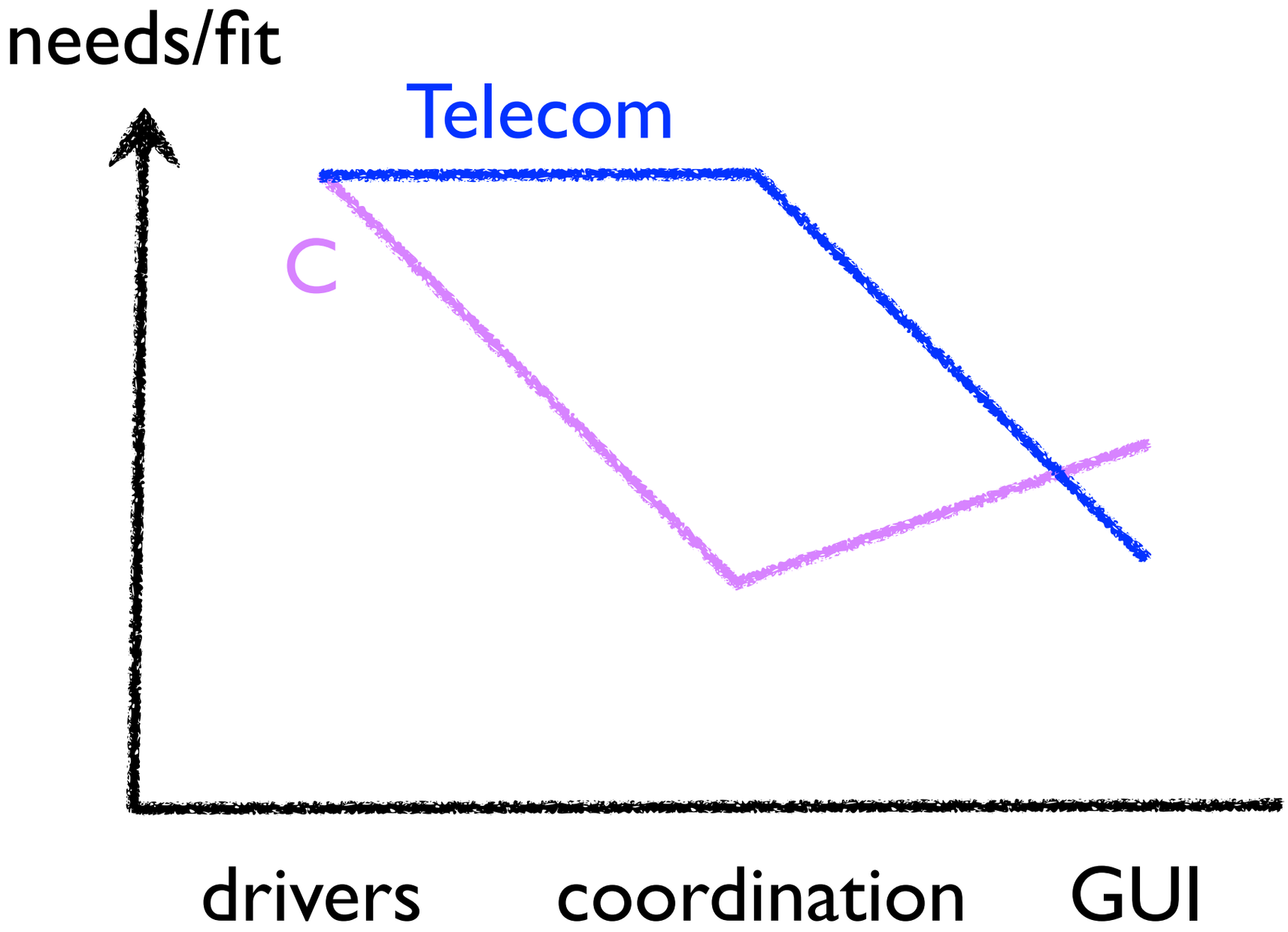
If the glove fits...



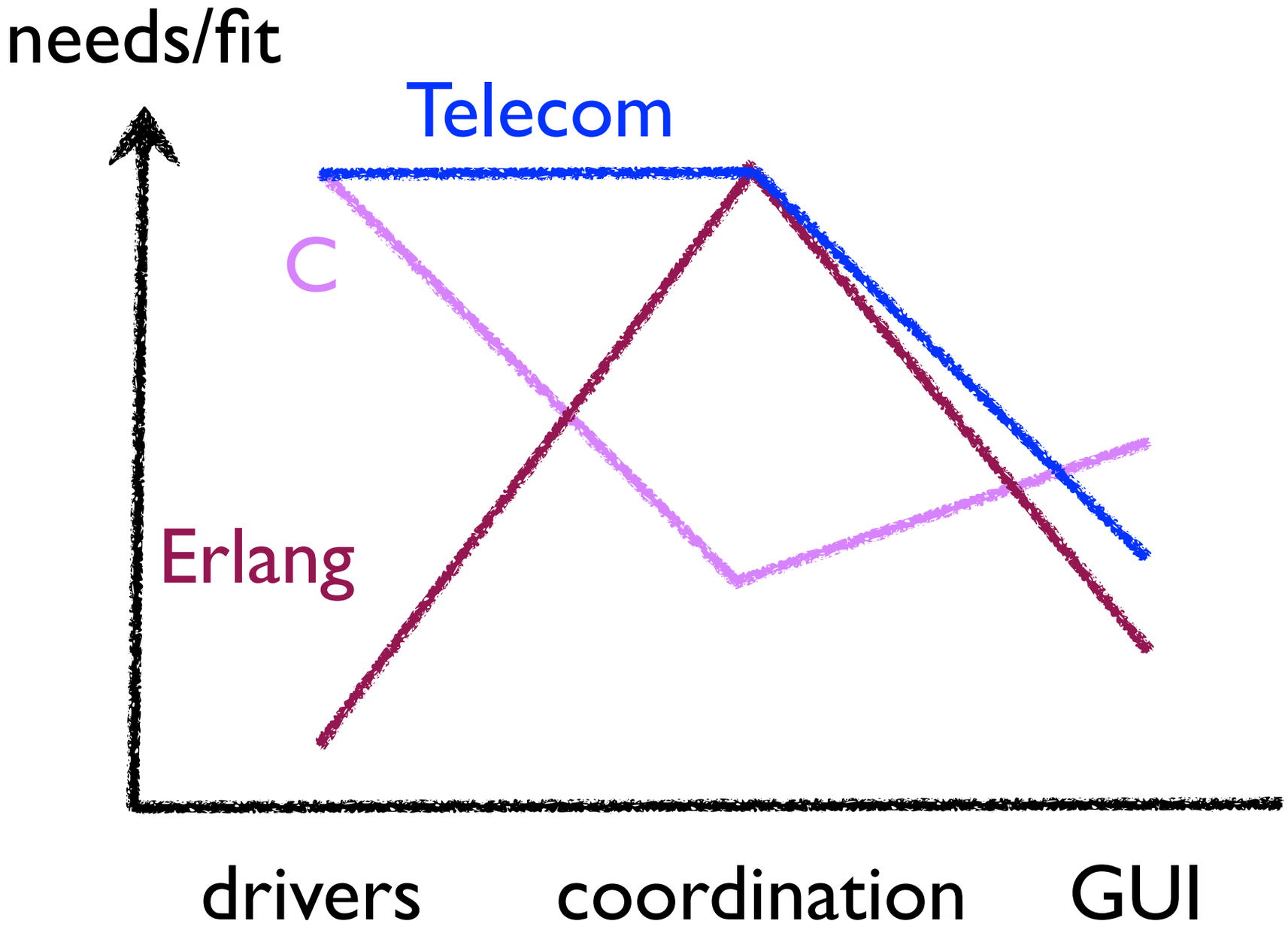
If the glove fits...



If the glove fits...



If the glove fits...



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

Other Erlang Domains

Other Erlang Domains

Messaging - XMPP et al

Other Erlang Domains

Messaging - XMPP et al

ejabberd, MongooseIM

Other Erlang Domains

Messaging - XMPP et al

ejabberd, MongooseIM

Webservers

Other Erlang Domains

Messaging - XMPP et al

ejabberd, MongooseIM

Webservers

Yaws, Chicago Boss, Cowboy

Other Erlang Domains

Messaging - XMPP et al

ejabberd, MongooseIM

Webservers

Yaws, Chicago Boss, Cowboy

Payment switches & soft switches

Other Erlang Domains

Messaging - XMPP et al

ejabberd, MongooseIM

Webservers

Yaws, Chicago Boss, Cowboy

Payment switches & soft switches

Vocalink, OpenFlow/LINC

Other Erlang Domains

Messaging - XMPP et al

ejabberd, MongooseIM

Webservers

Yaws, Chicago Boss, Cowboy

Payment switches & soft switches

Vocalink, OpenFlow/LINC

Distributed Databases

Other Erlang Domains

Messaging - XMPP et al

ejabberd, MongooseIM

Webservers

Yaws, Chicago Boss, Cowboy

Payment switches & soft switches

Vocalink, OpenFlow/LINC

Distributed Databases

Riak, CouchDB, Scalaris

Other Erlang Domains

Messaging - XMPP et al

ejabberd, MongooseIM

Webservers

Yaws, Chicago Boss, Cowboy

Payment switches & soft switches

Vocalink, OpenFlow/LINC

Distributed Databases

Riak, CouchDB, Scalaris

Queueing systems

Other Erlang Domains

Messaging - XMPP et al

ejabberd, MongooseIM

Webservers

Yaws, Chicago Boss, Cowboy

Payment switches & soft switches

Vocalink, OpenFlow/LINC

Distributed Databases

Riak, CouchDB, Scalaris

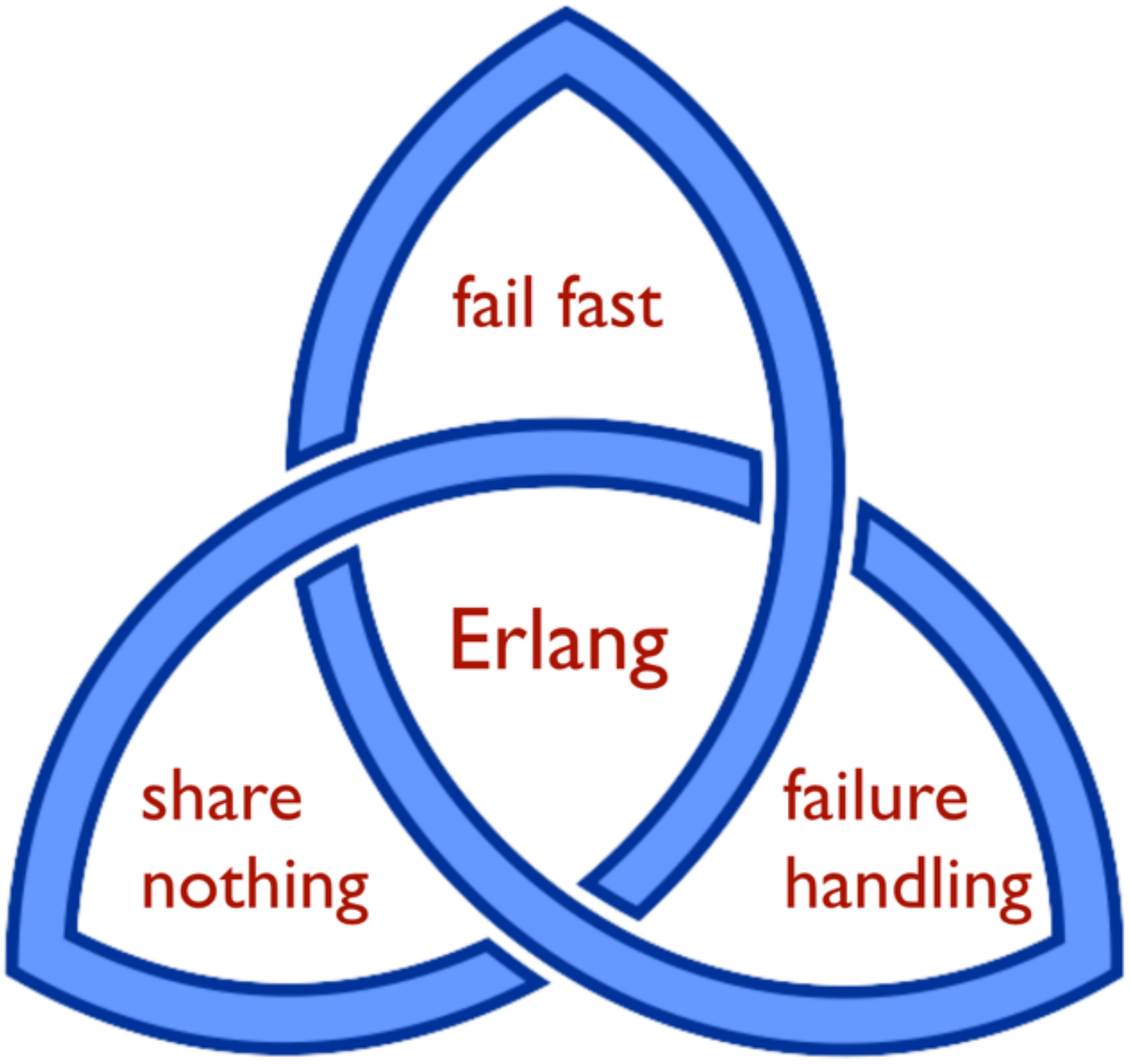
Queueing systems

RabbitMQ (AMQP)

The glove fits!

	Low latency	Stateful	Massively concurrent	Distributed	Fault tolerant
Messaging	✓	✓	✓	✓	✓
Webservers	✓	✓	✓		✓
Soft switches	✓	✓	✓	✓	✓
Distributed DBs	✓	✓	✓	✓	✓
Queueing systems	✓	✓	✓	✓	✓

The Golden Trinity Of Erlang

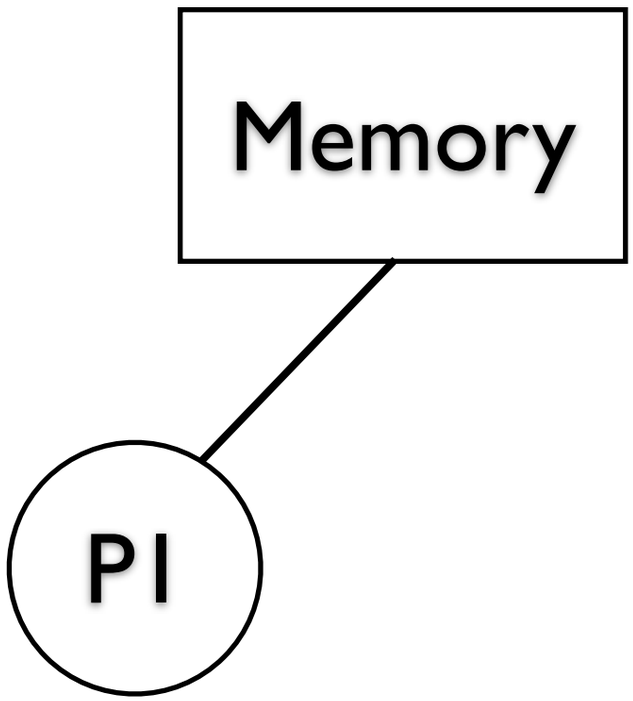


To Share Or Not To Share

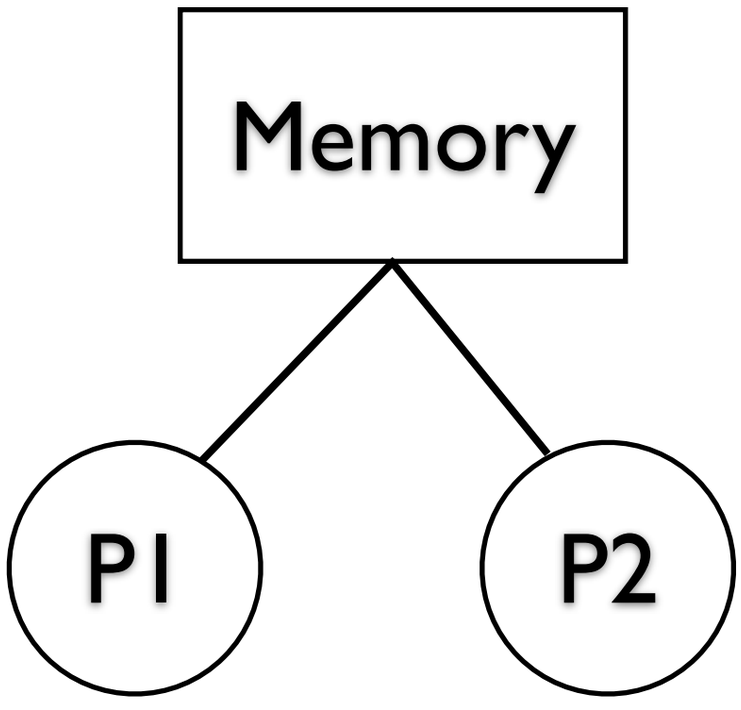
To Share Or Not To Share

Memory

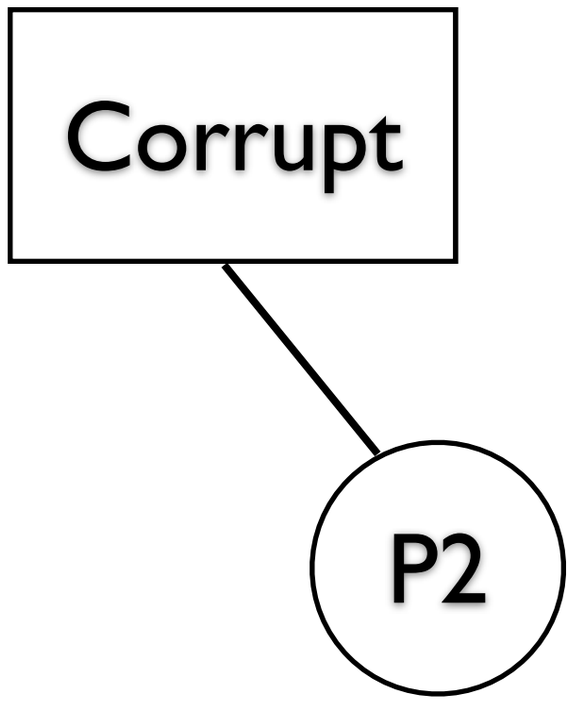
To Share Or Not To Share



To Share Or Not To Share



To Share Or Not To Share



To Share Or Not To Share

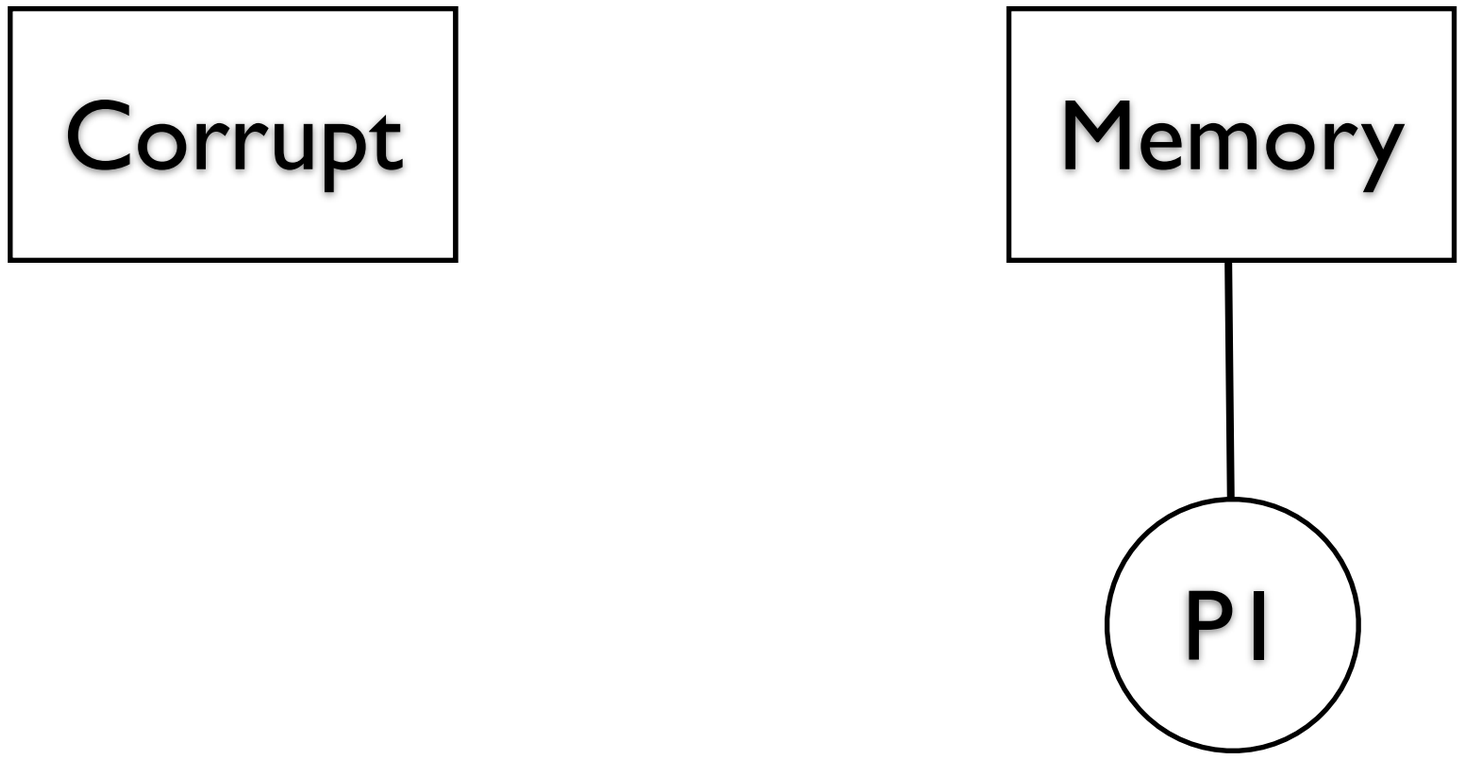
Corrupt

To Share Or Not To Share

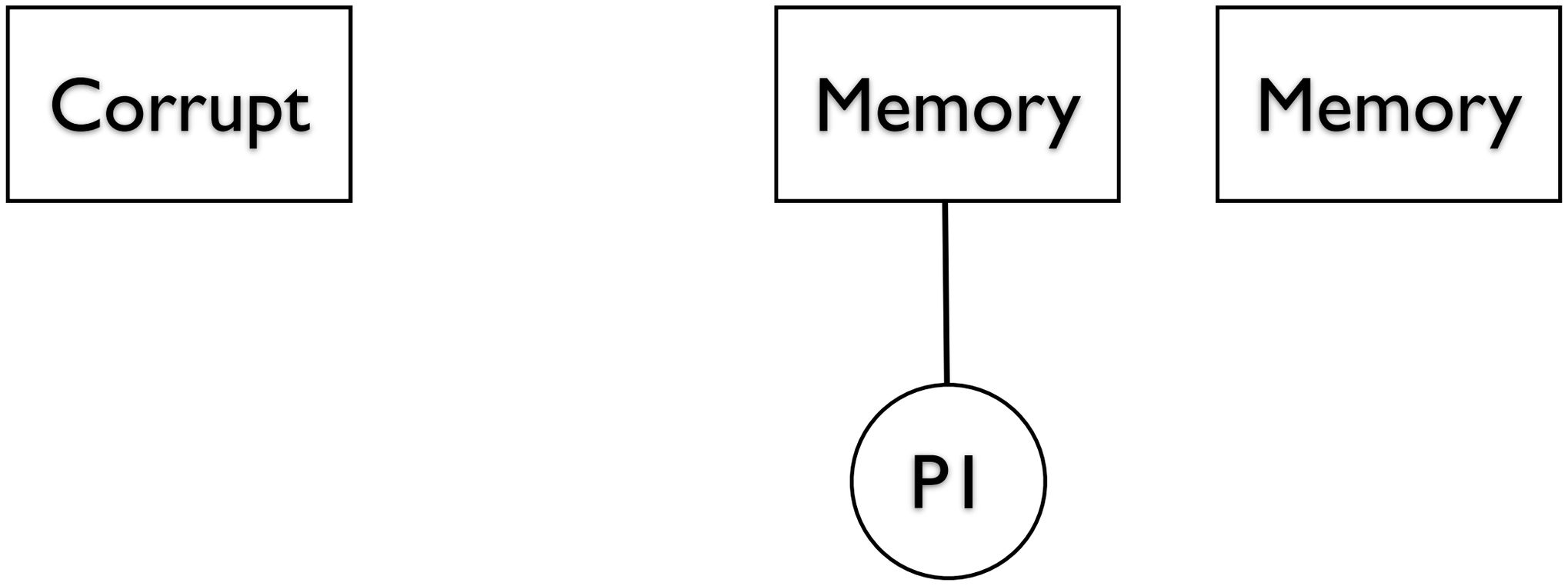
Corrupt

Memory

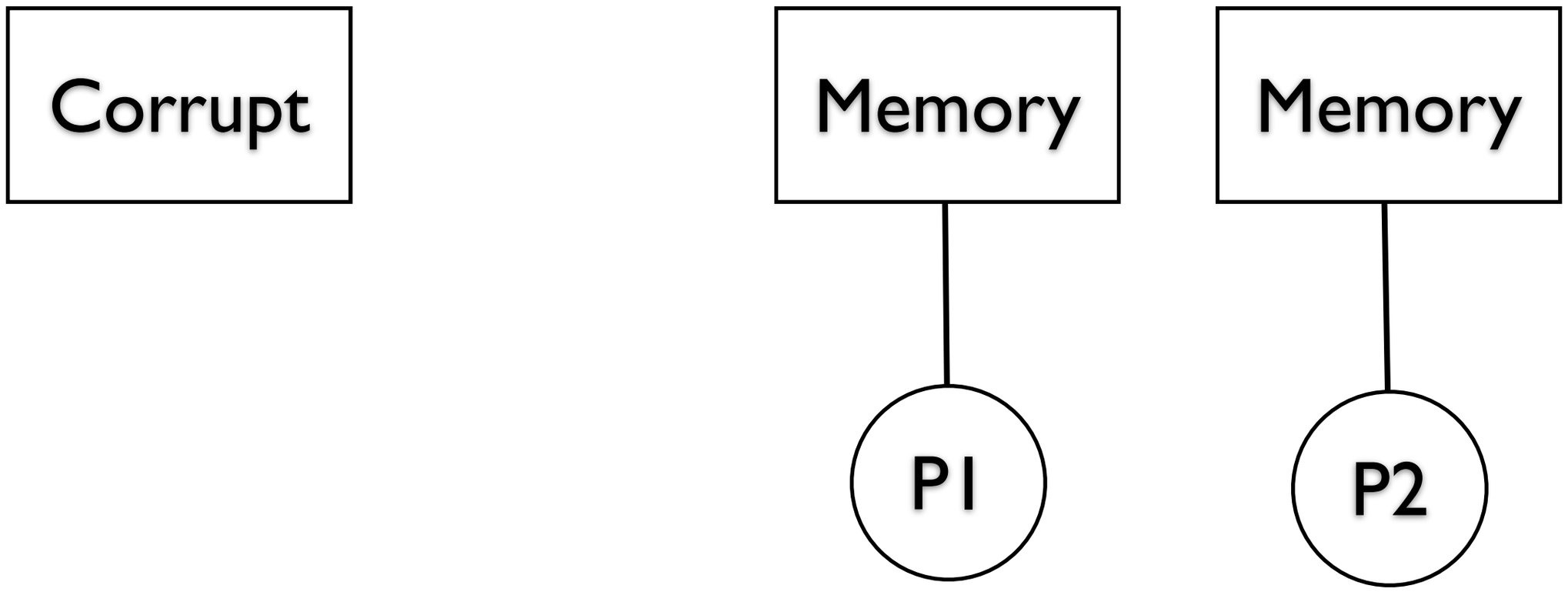
To Share Or Not To Share



To Share Or Not To Share



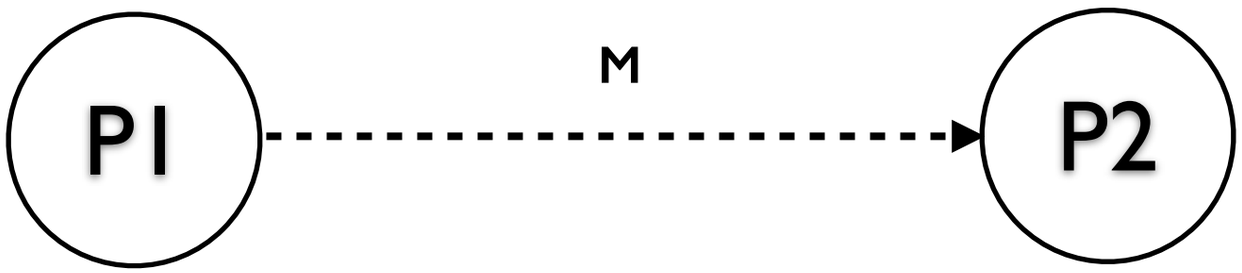
To Share Or Not To Share



To Share Or Not To Share

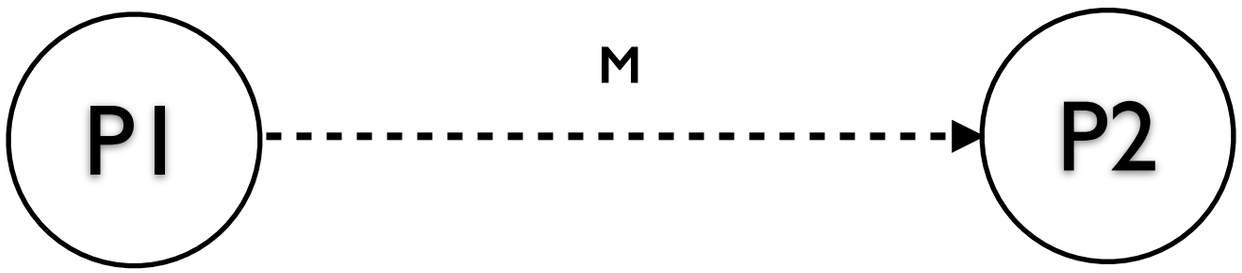
To Share Or Not To Share

Message Passing



P1 sends M to P2.

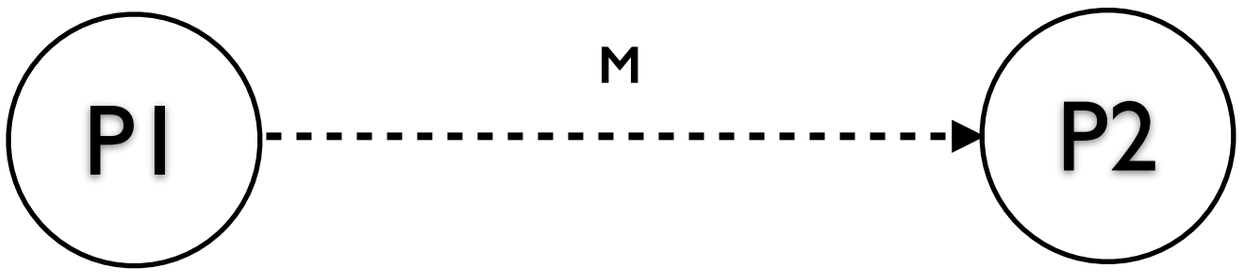
Message Passing



P1 sends M to P2.

Every process has a mailbox

Message Passing



P1 sends M to P2.

Every process has a mailbox

Messages are received:

```
receive
    {tag, Value} -> Value;
    N when is_integer(N) -> N + 42
end
```

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

Programming errors
Disk failures
Network failures

Anything that can go wrong,
will go wrong

Murphy

Most programming paradigmes are
fault in-tolerant

Failures

Programming errors
Disk failures
Network failures

Anything that can go wrong,
will go wrong

Murphy

Most programming paradigmes are
fault in-tolerant

⇒ must deal with all errors or die

Failures

Programming errors
Disk failures
Network failures

Anything that can go wrong,
will go wrong

Murphy

Most programming paradigmes are *fault in-tolerant*

⇒ must deal with all errors or die

Failures

Programming errors
Disk failures
Network failures

Anything that can go wrong,
will go wrong

Murphy

Most programming paradigmes are *fault in-tolerant*

⇒ must deal with all errors or die

Erlang is *fault tolerant* by design

Failures

Programming errors
Disk failures
Network failures

Anything that can go wrong,
will go wrong

Murphy

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

Failures

Programming errors
Disk failures
Network failures

Anything that can go wrong,
will go wrong

Murphy

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

source: http://www.thelmagazine.com/BrooklynAbridged/archives/2013/05/14/should-we-be-worried-about-this-brooklyn-measles-outbreak

source: http://johnkreng.wordpress.com/tag/jean-claude-van-damme/

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

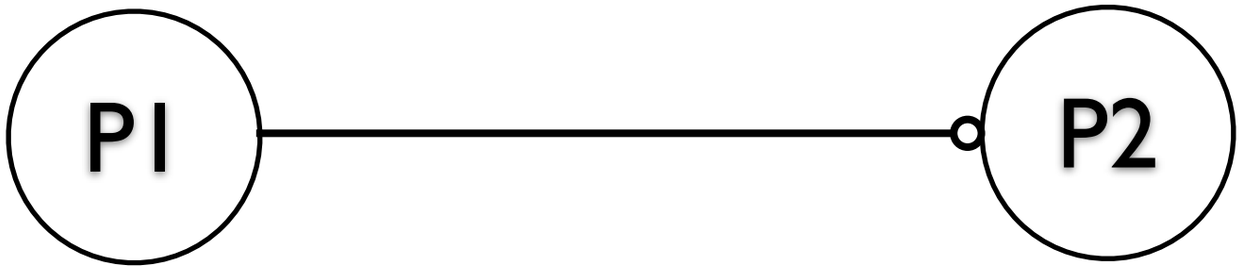
```
convert(monday)    -> 1;
convert(tuesday)   -> 2;
convert(wednesday) -> 3;
convert(thursday)  -> 4;
convert(friday)    -> 5;
convert(saturday)  -> 6;
convert(sunday)    -> 7.
```

Let It Fail

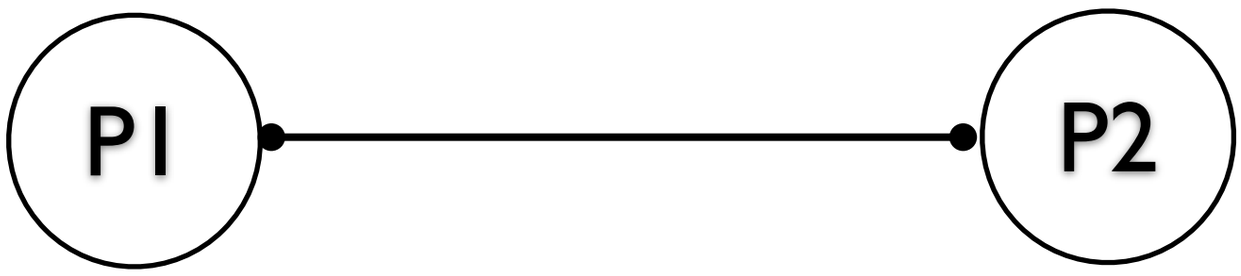
```
convert(monday)    -> 1;
convert(tuesday)   -> 2;
convert(wednesday) -> 3;
convert(thursday)  -> 4;
convert(friday)    -> 5;
convert(saturday)  -> 6;
convert(sunday)    -> 7.
```

Erlang encourages offensive programming

Handling Failure



P1 monitors P2.



P1 and P2 are linked.

Intentional Programming

a style of programming where the reader of a program can easily see what the programmer intended by their code. [1]

[1] http://www.erlang.org/download/armstrong_thesis_2003.pdf

Intentional Dictionary

data retrieval - `dict:fetch(Key, Dict) = Val | EXIT`

the programmer knows a specific key should be in the dictionary and it is an error if it is not.

search - `dict:find(Key, Dict) = {ok, Val} | error.`

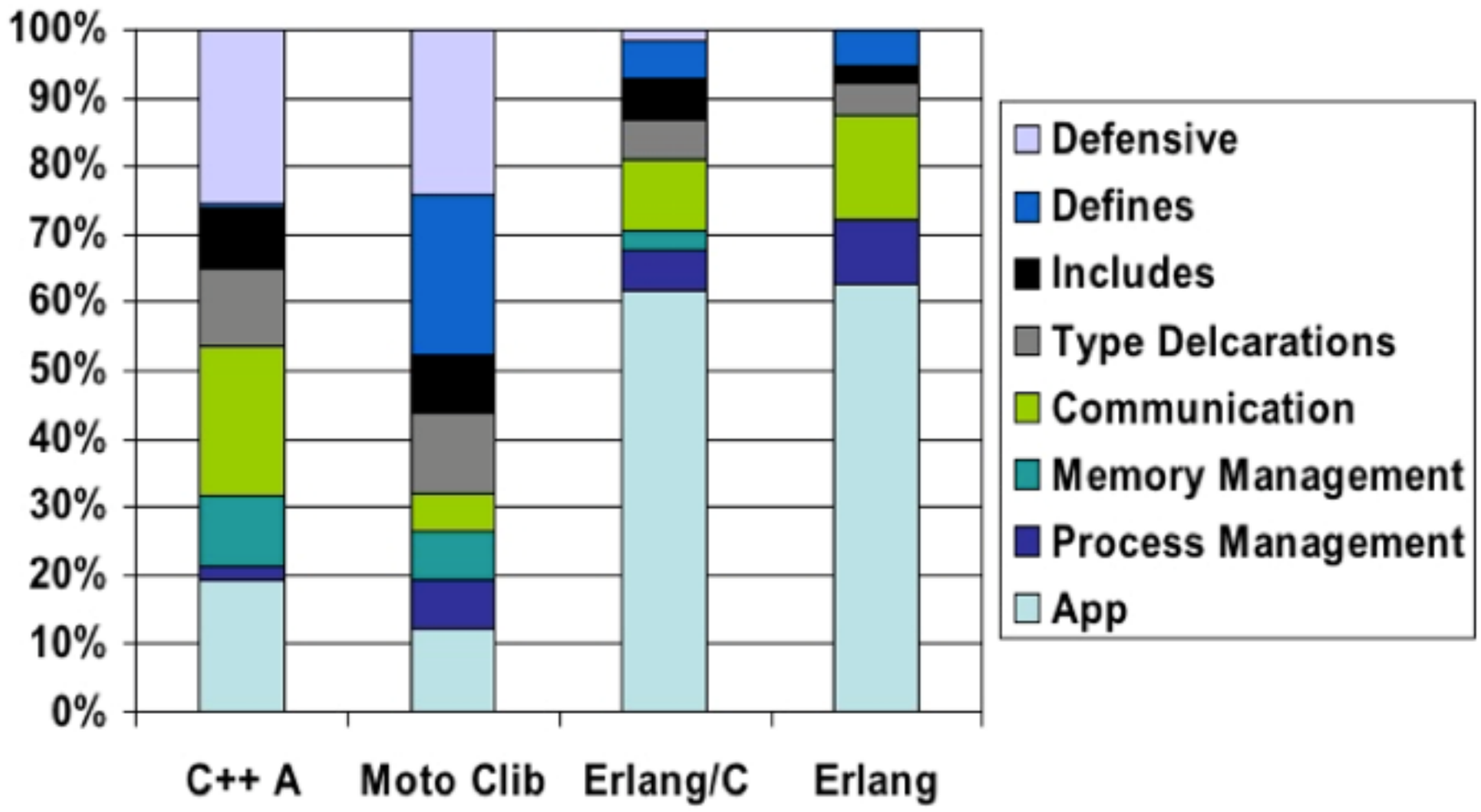
it is unknown if the key is there or not and both cases must be dealt with.

test - `dict:is_key(Key, Dict) = Boolean`

knowing if a key is present is enough.

Benefits of let-it-fail

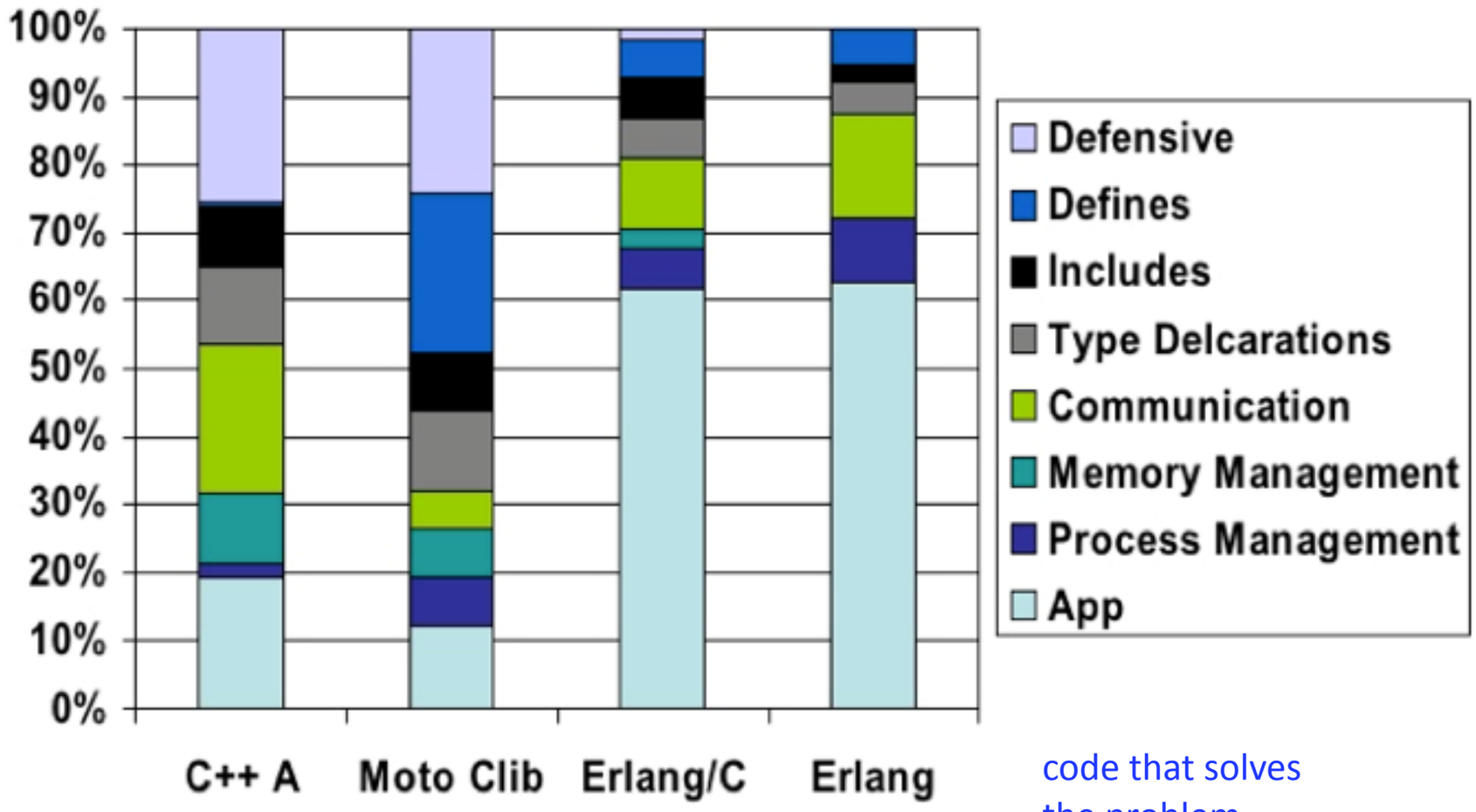
Data Mobility component breakdown



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

Benefits of let-it-fail

Data Mobility component breakdown

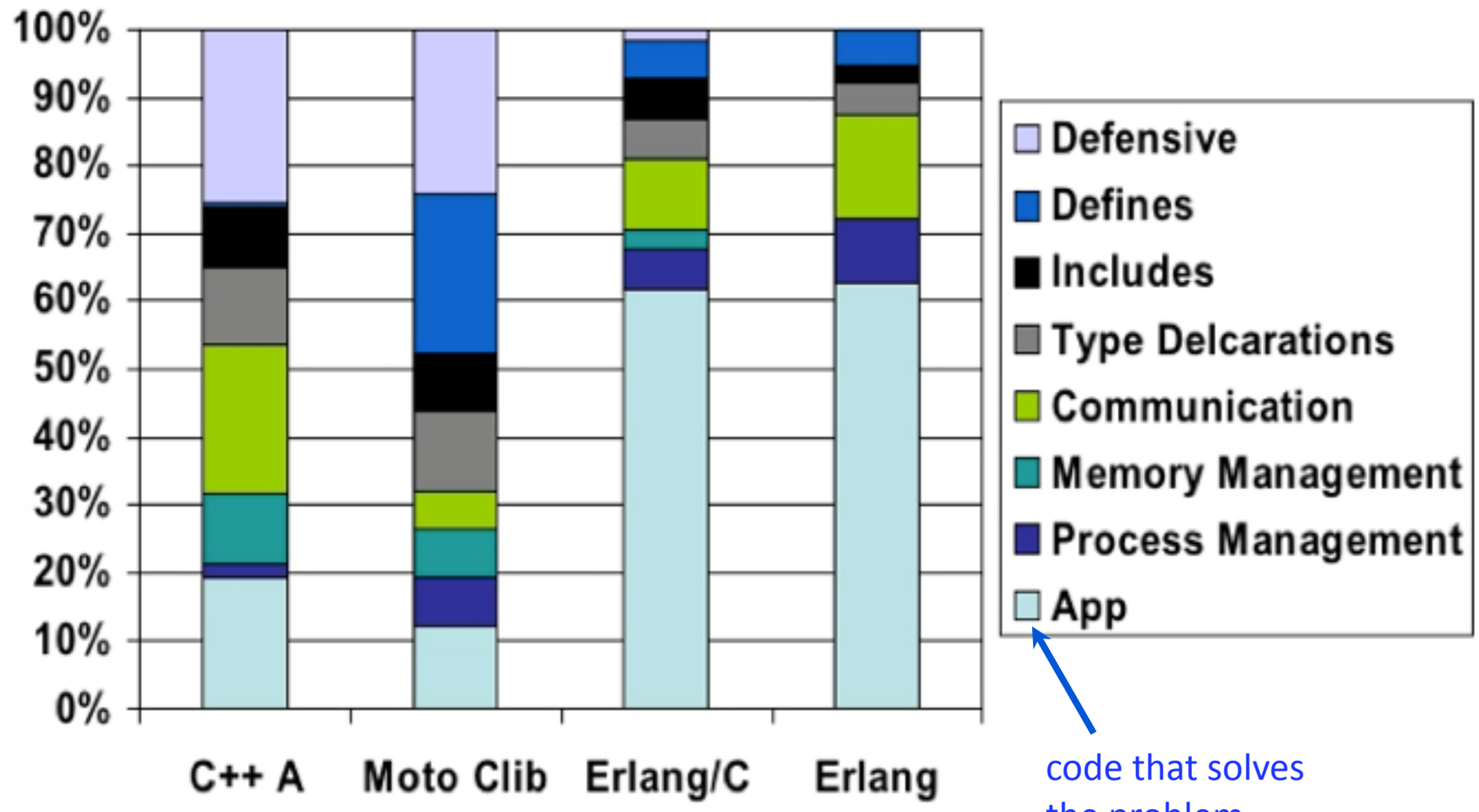


code that solves the problem

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

Benefits of let-it-fail

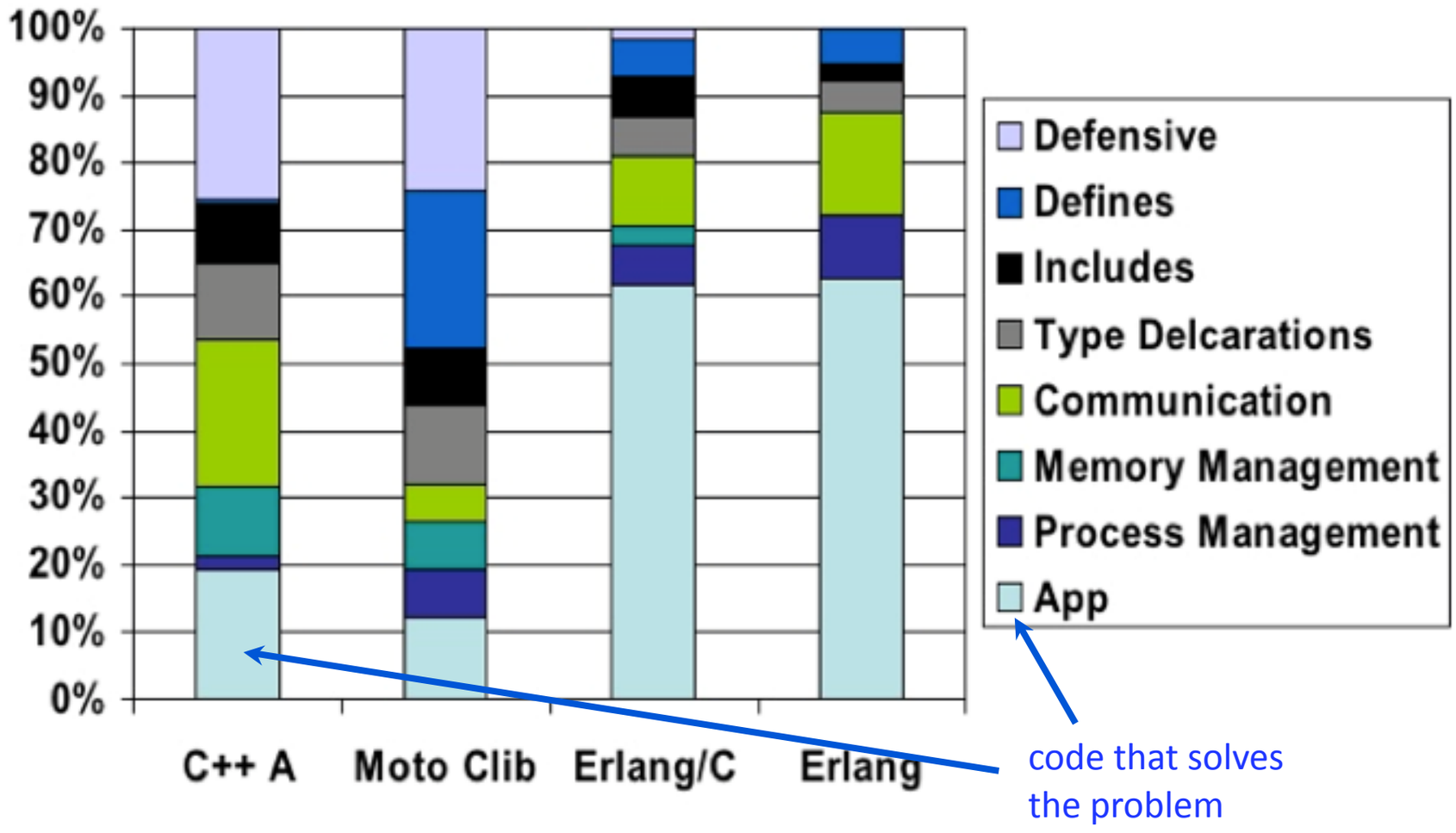
Data Mobility component breakdown



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

Benefits of let-it-fail

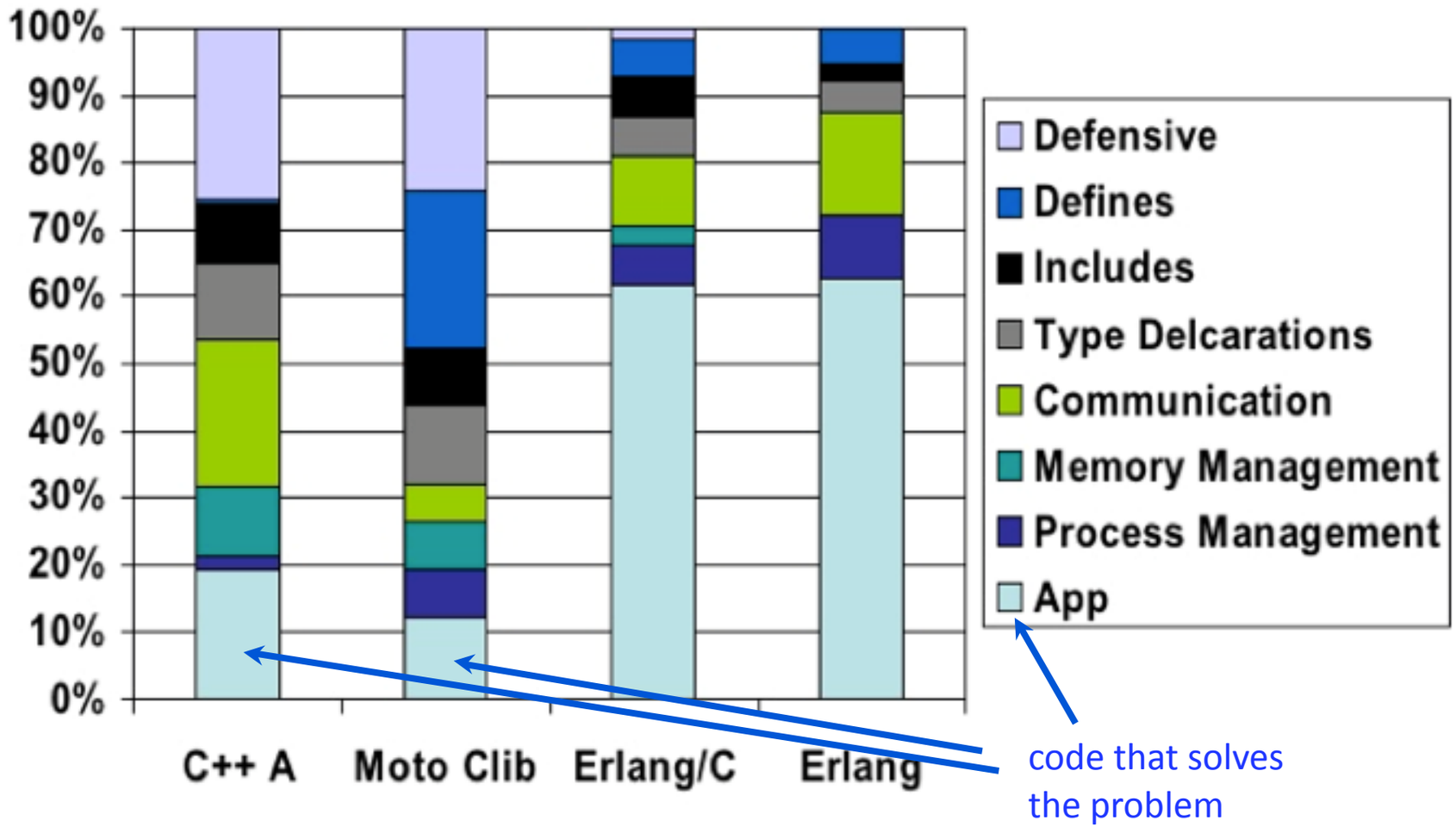
Data Mobility component breakdown



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

Benefits of let-it-fail

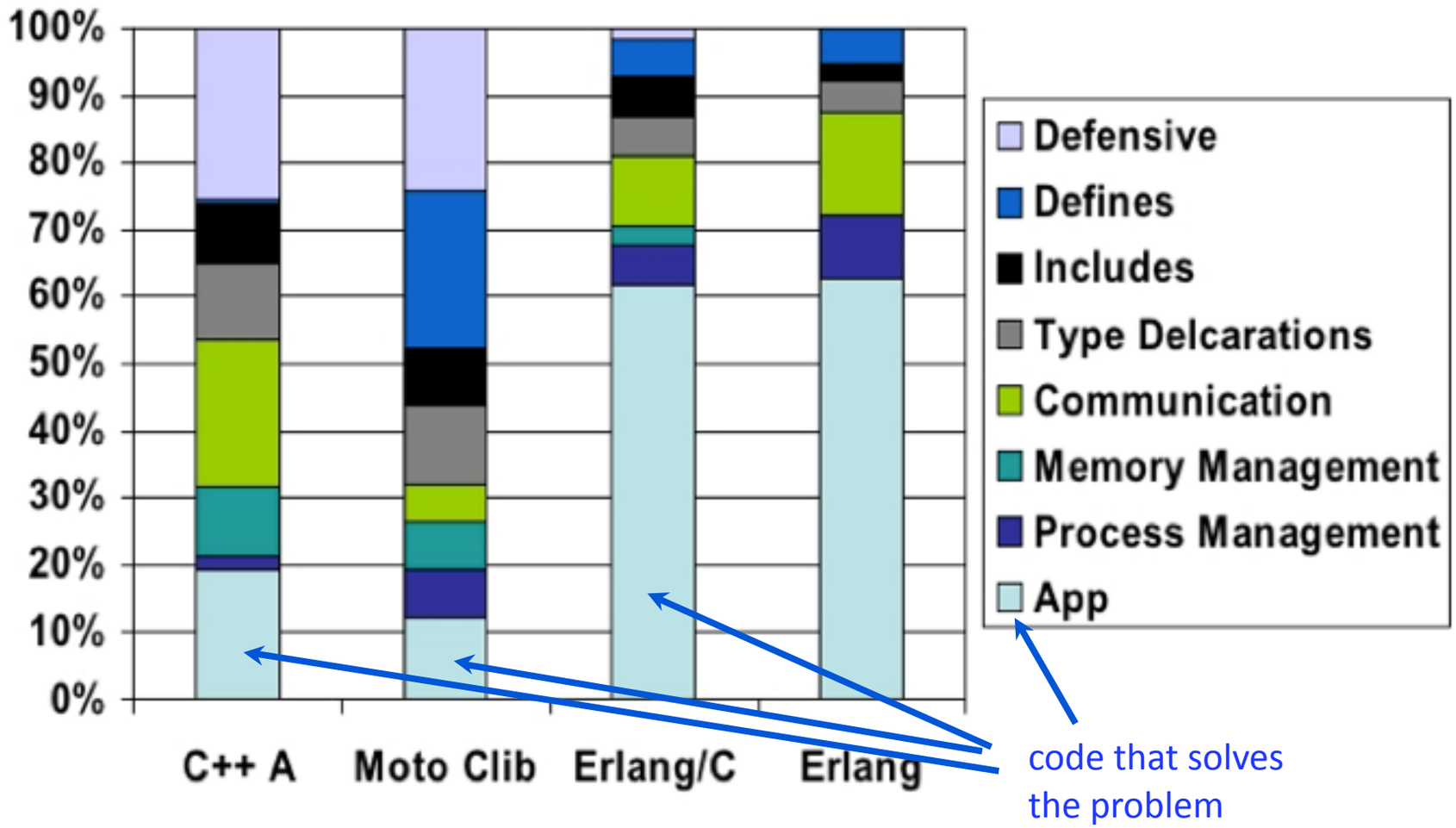
Data Mobility component breakdown



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

Benefits of let-it-fail

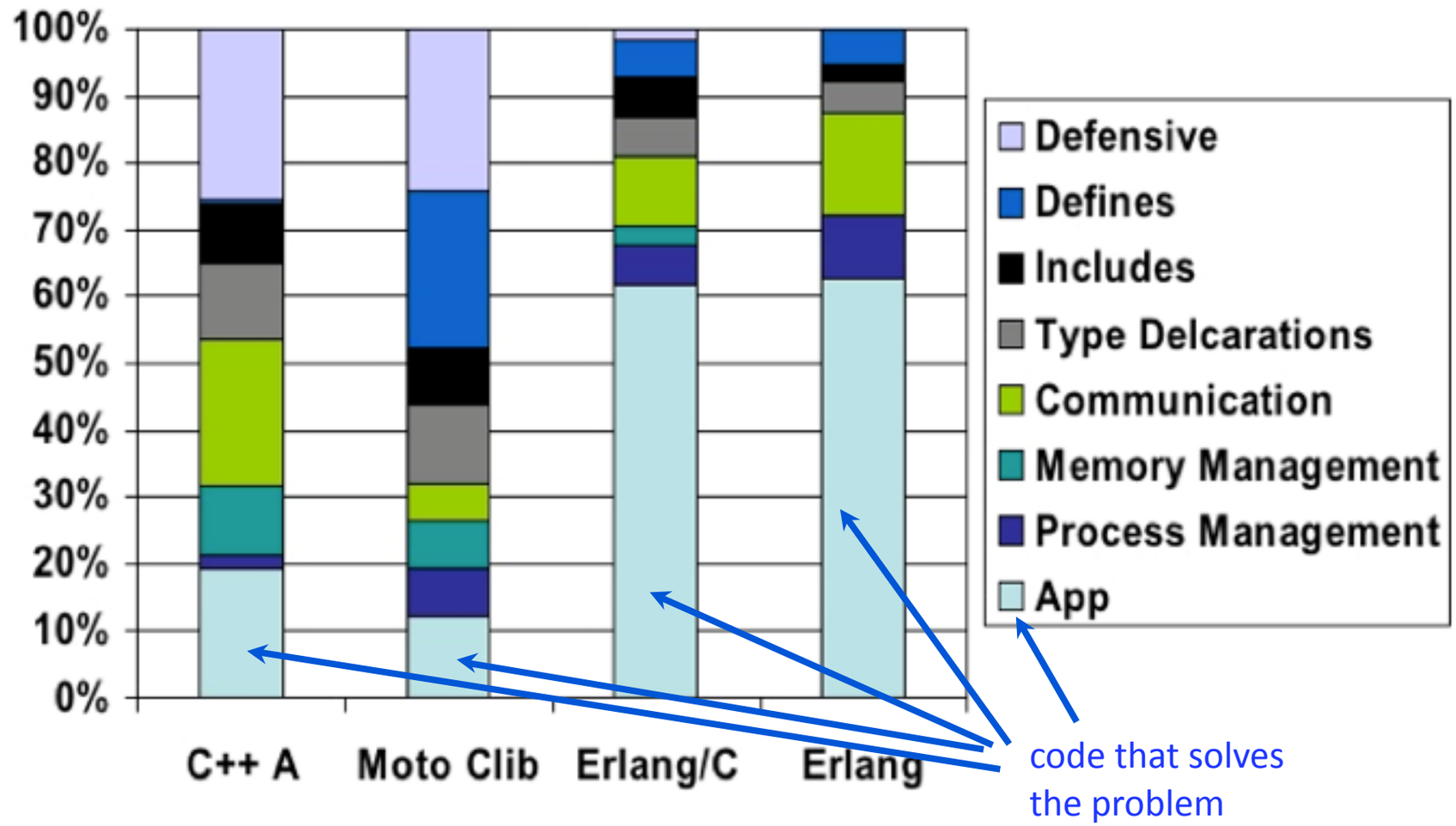
Data Mobility component breakdown



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

Benefits of let-it-fail

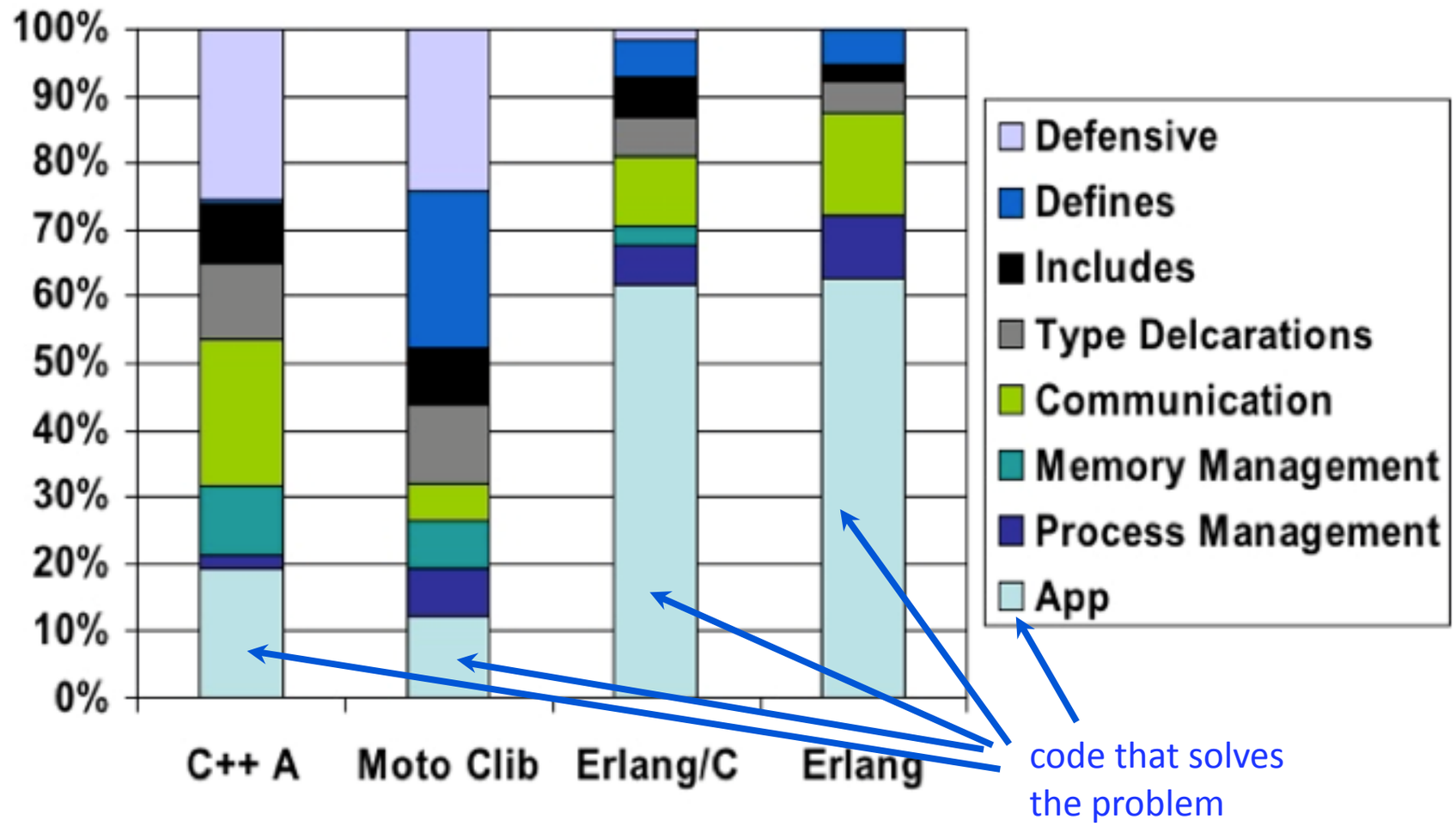
Data Mobility component breakdown



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

Erlang @ 3x

Productivity with Erlang @ Motorola

Goal: Development of a mission-critical telecom gateway for TETRA

To be developed from scratch using Erlang and some drivers in C

The gateway translates between proprietary protocols and the ISI standard

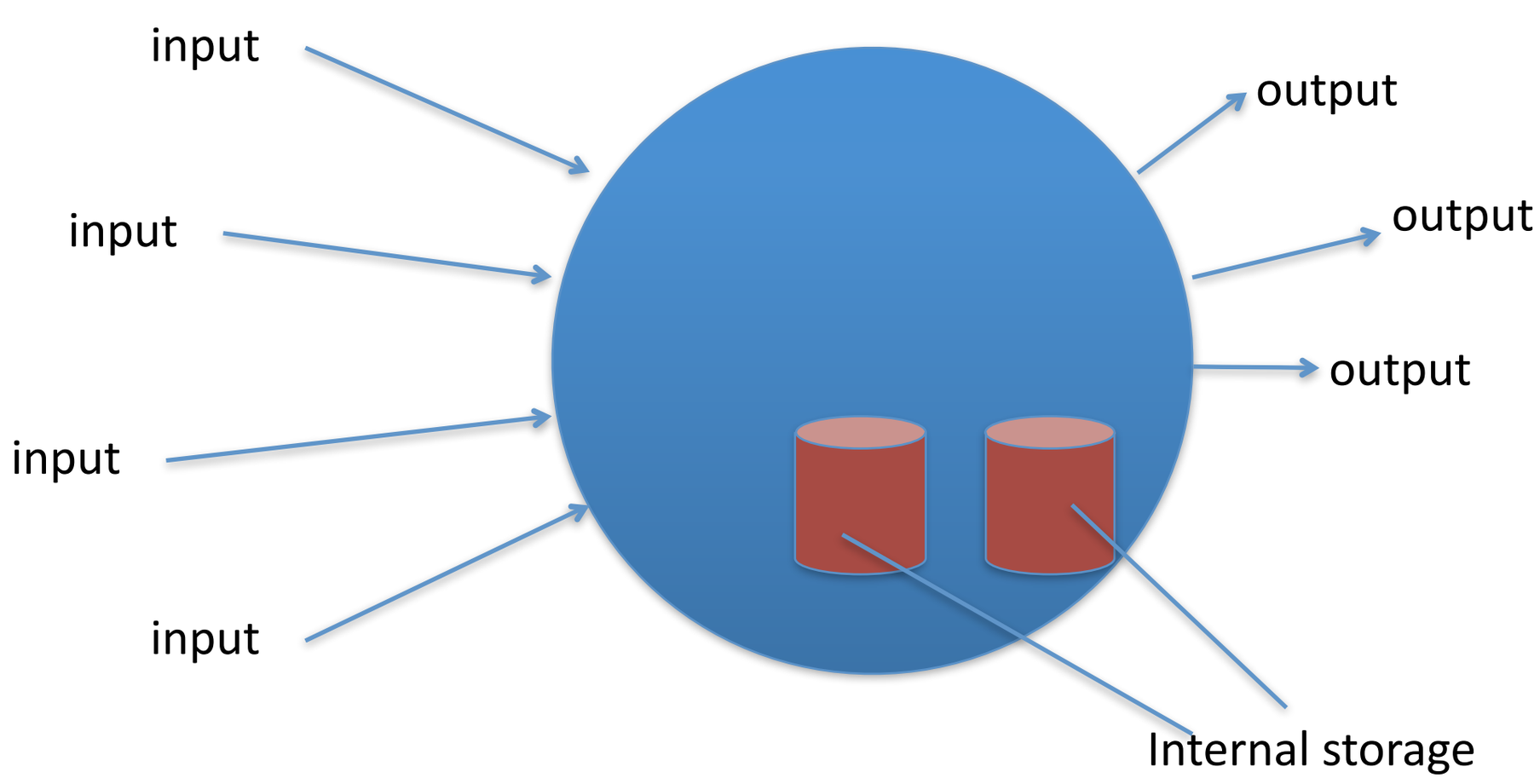
Developed using a two – six man team over four year

A total of 72 staff months used on the work

Productivity with Erlang @ Motorola

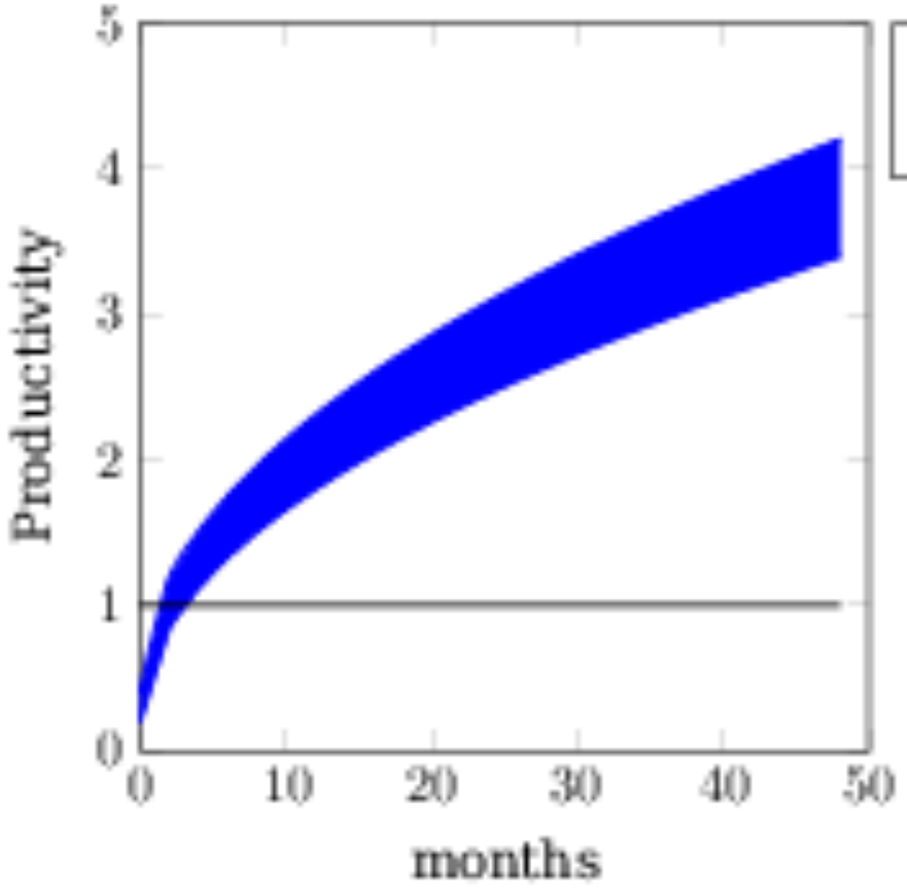
Function point analysis

Language agnostic measurement of problem size



Erlang SOLUTIONS

Show me the money!



Function Point Analysis of the size of the problem

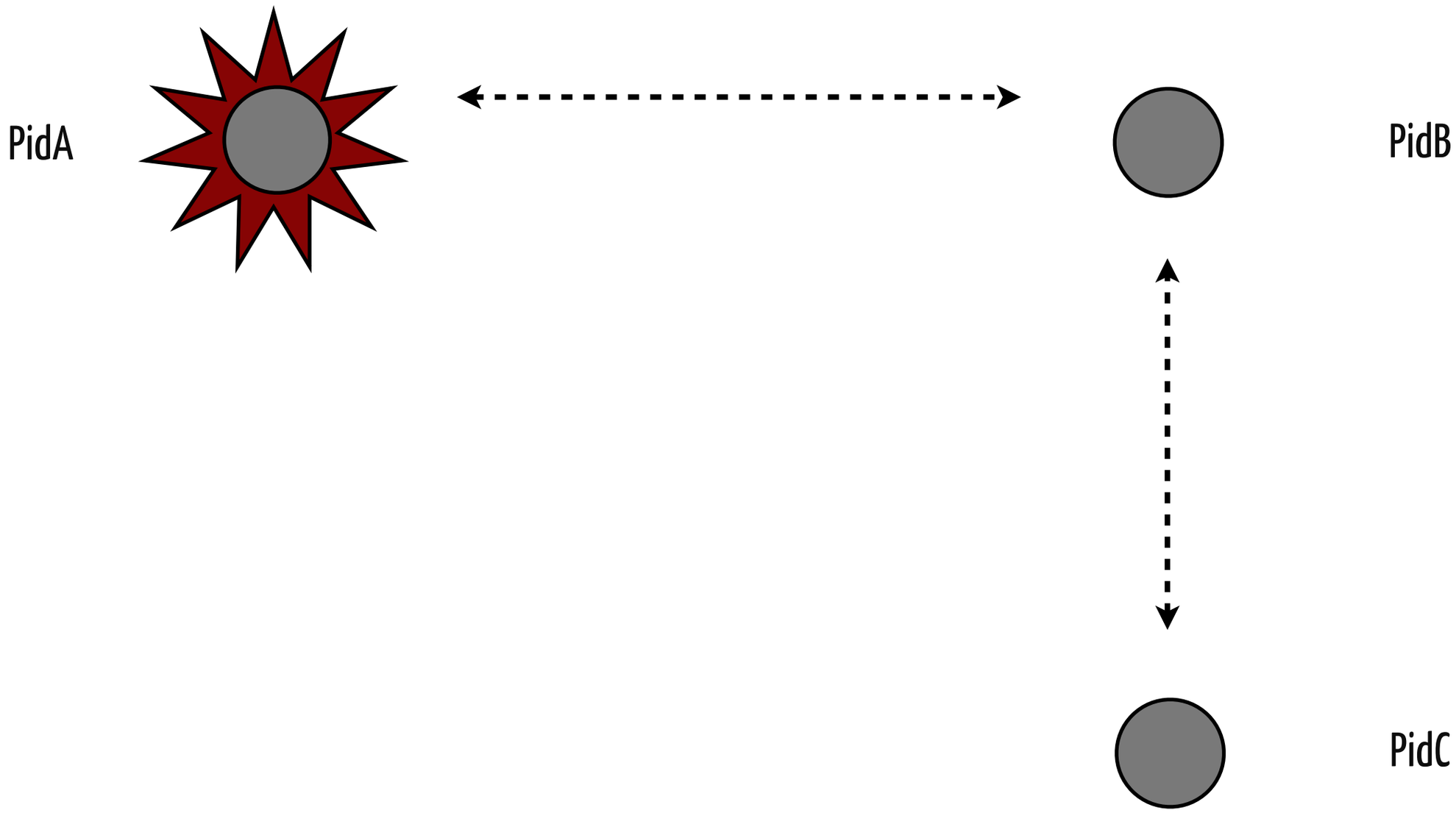
Conservative estimation of the number of inputs, outputs and internal storage

Includes design, box test, system test, project management efforts

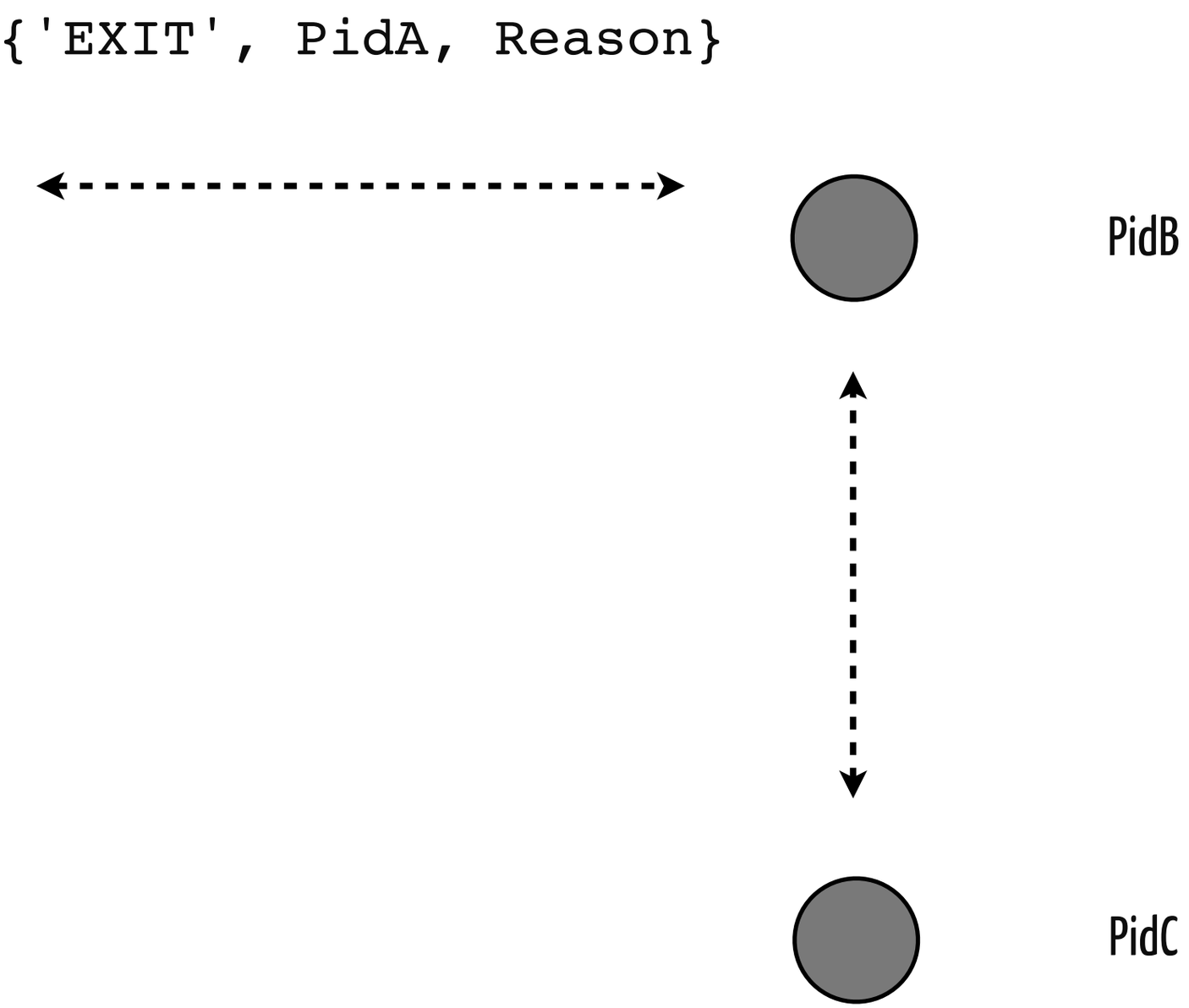
Isolate Errors

PROPAGATING EXIT SIGNALS

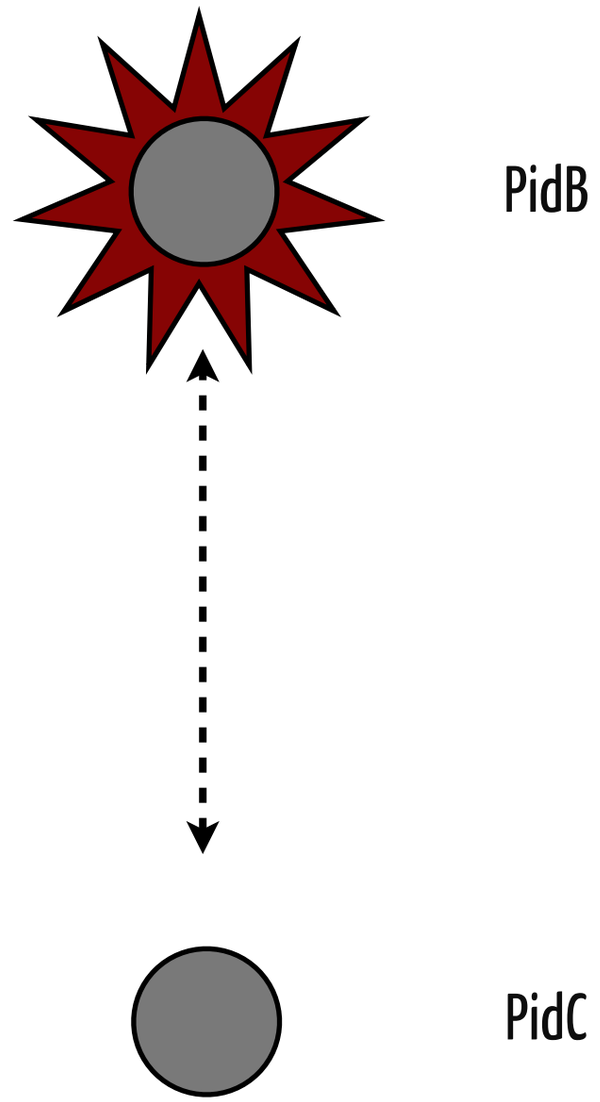
Propagating Exit Signals



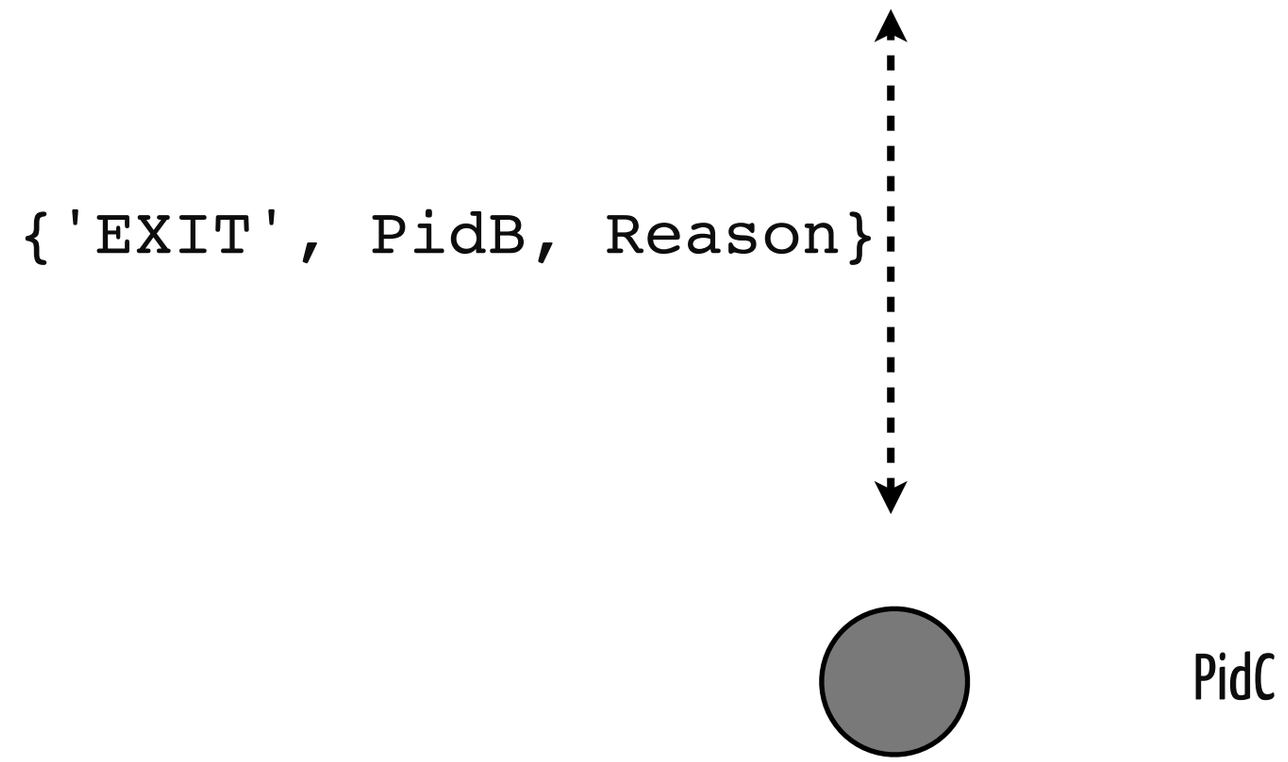
PROPAGATING EXIT SIGNALS



PROPAGATING EXIT SIGNALS



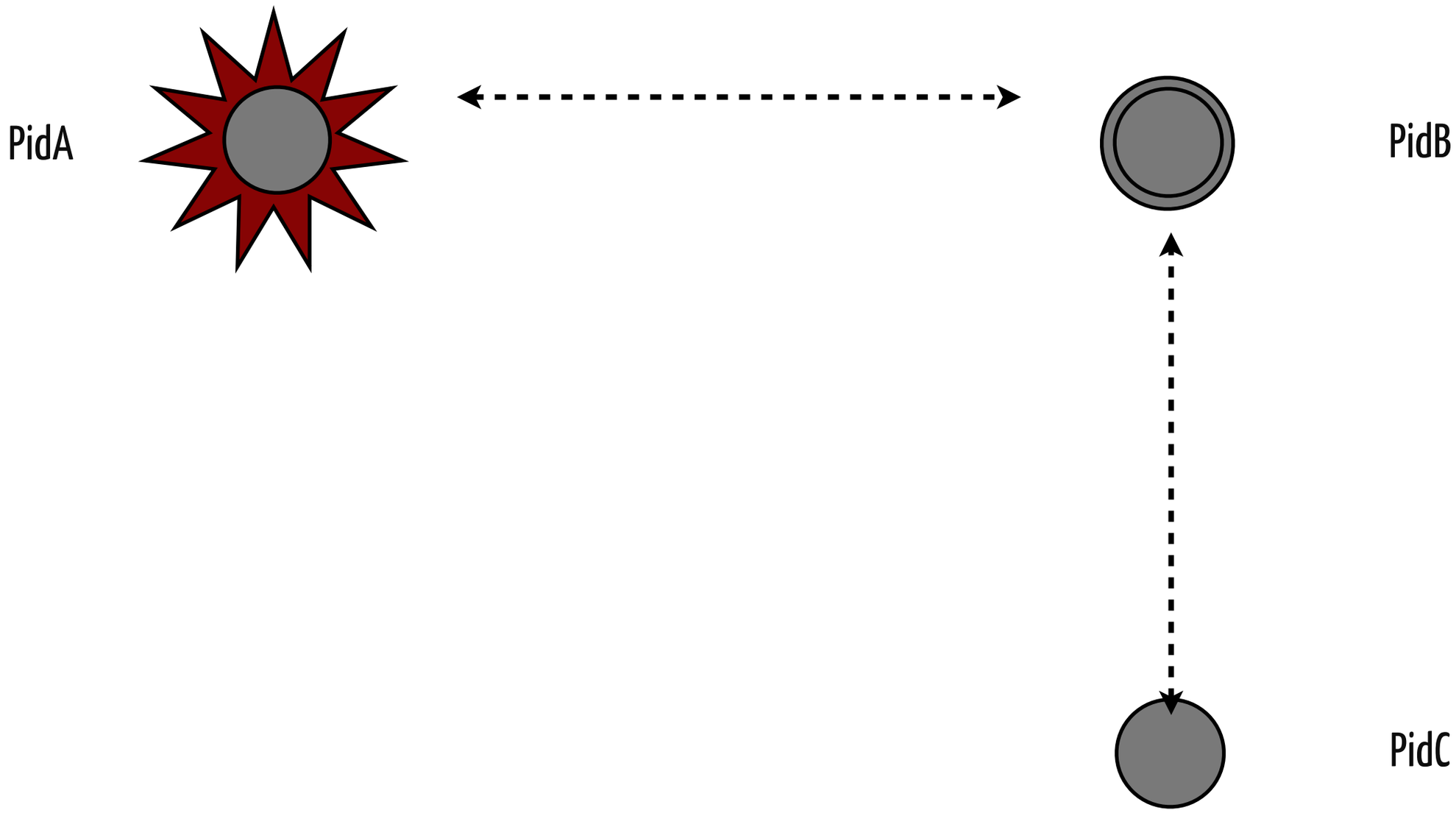
PROPAGATING EXIT SIGNALS



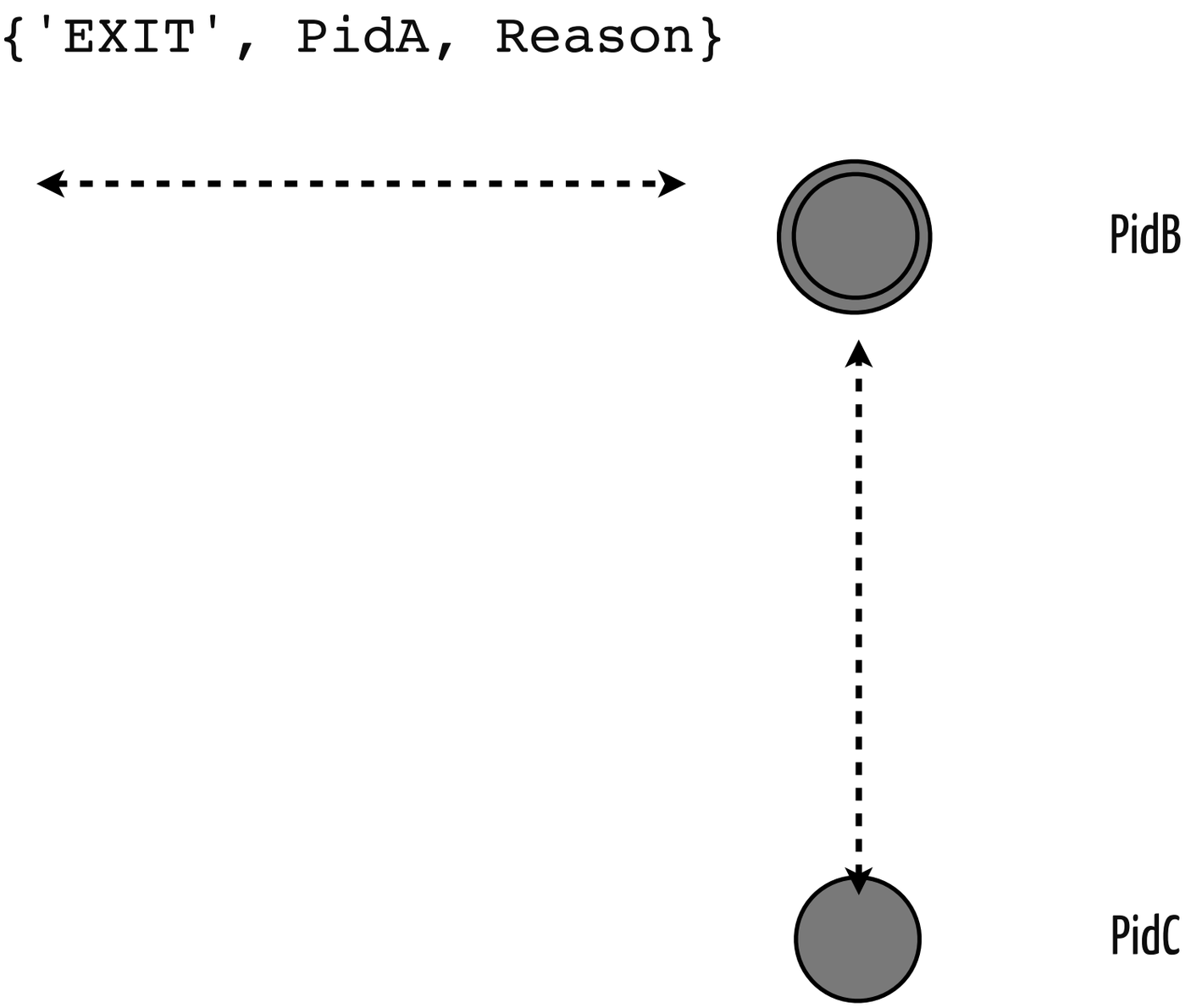
Propagating Exit Signals

TRAPPING AN EXIT SIGNAL

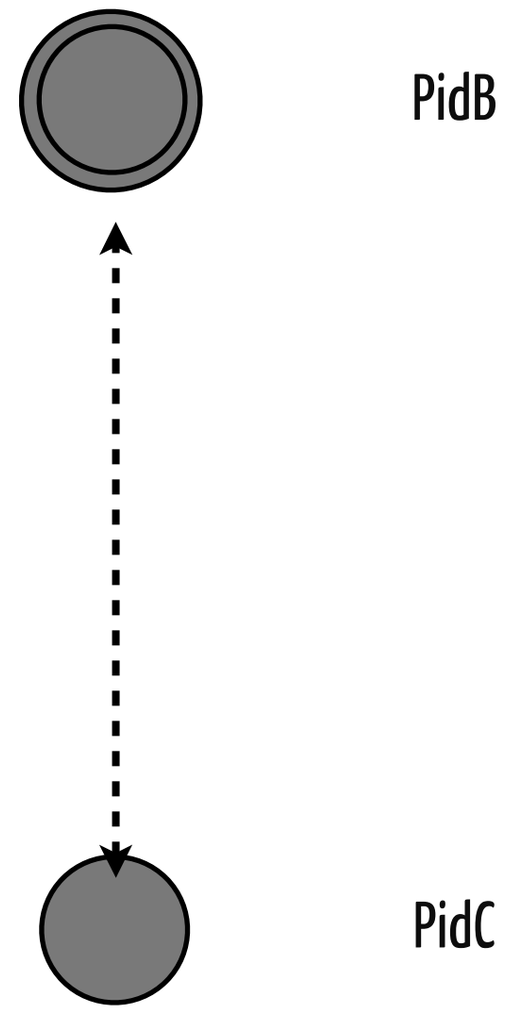
Trapping an Exit Signal



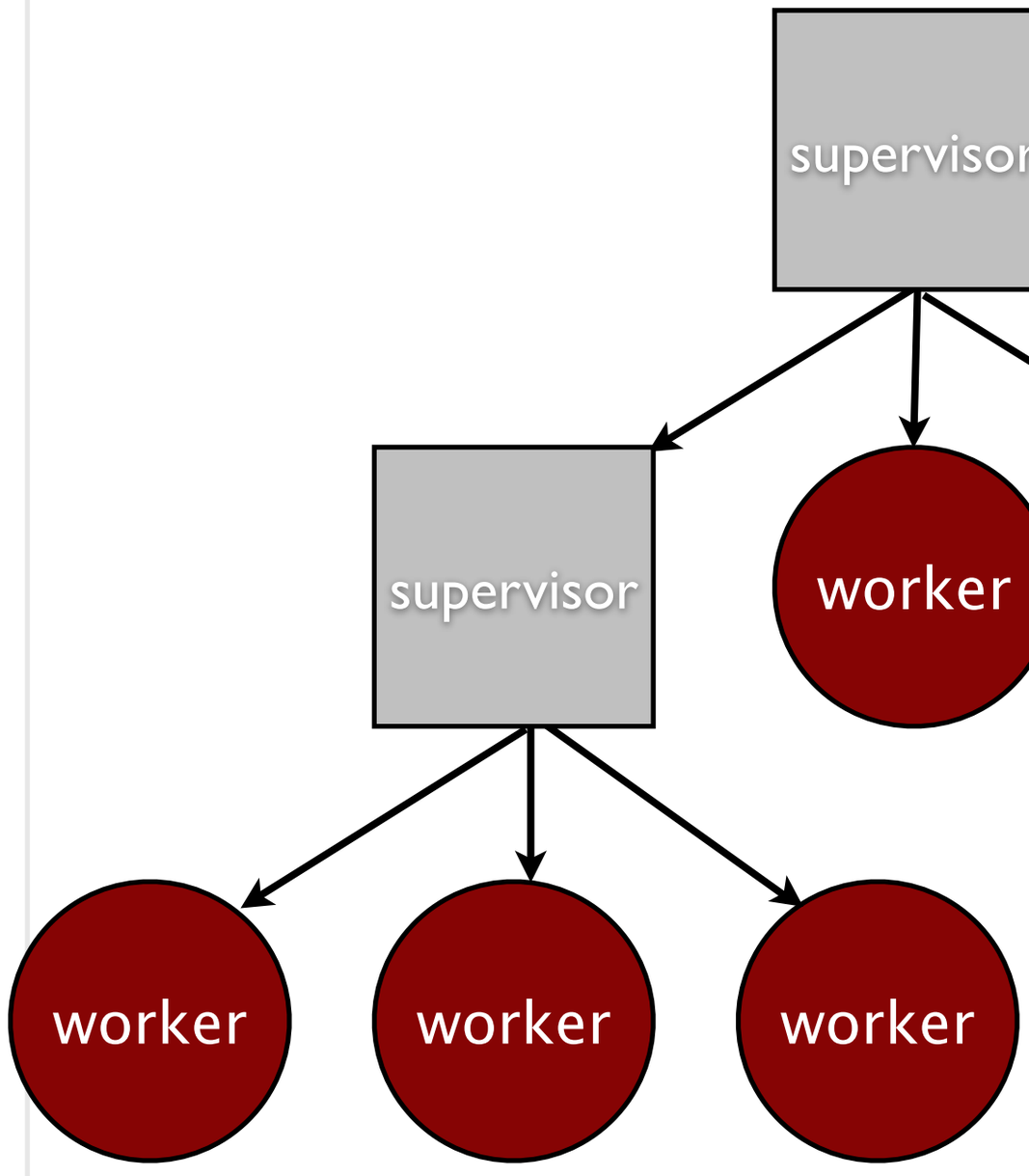
TRAPPING AN EXIT SIGNAL



TRAPPING AN EXIT SIGNAL

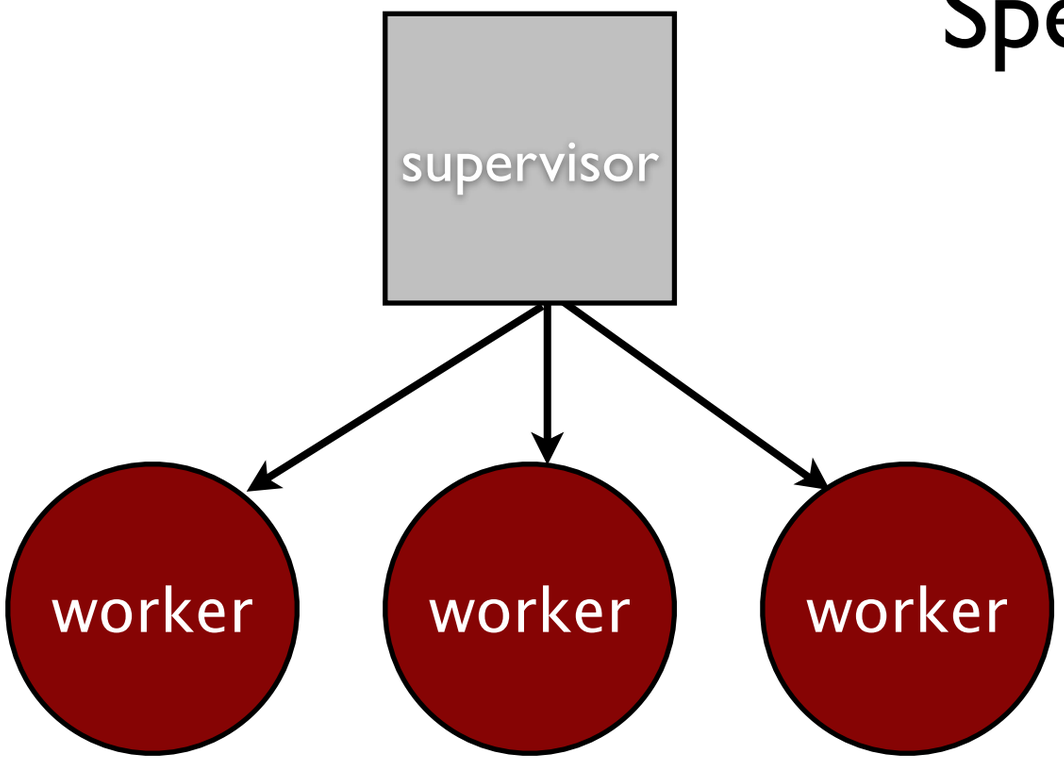


Supervision Trees



The OTP library is built on this principle

The OTP Supervisor



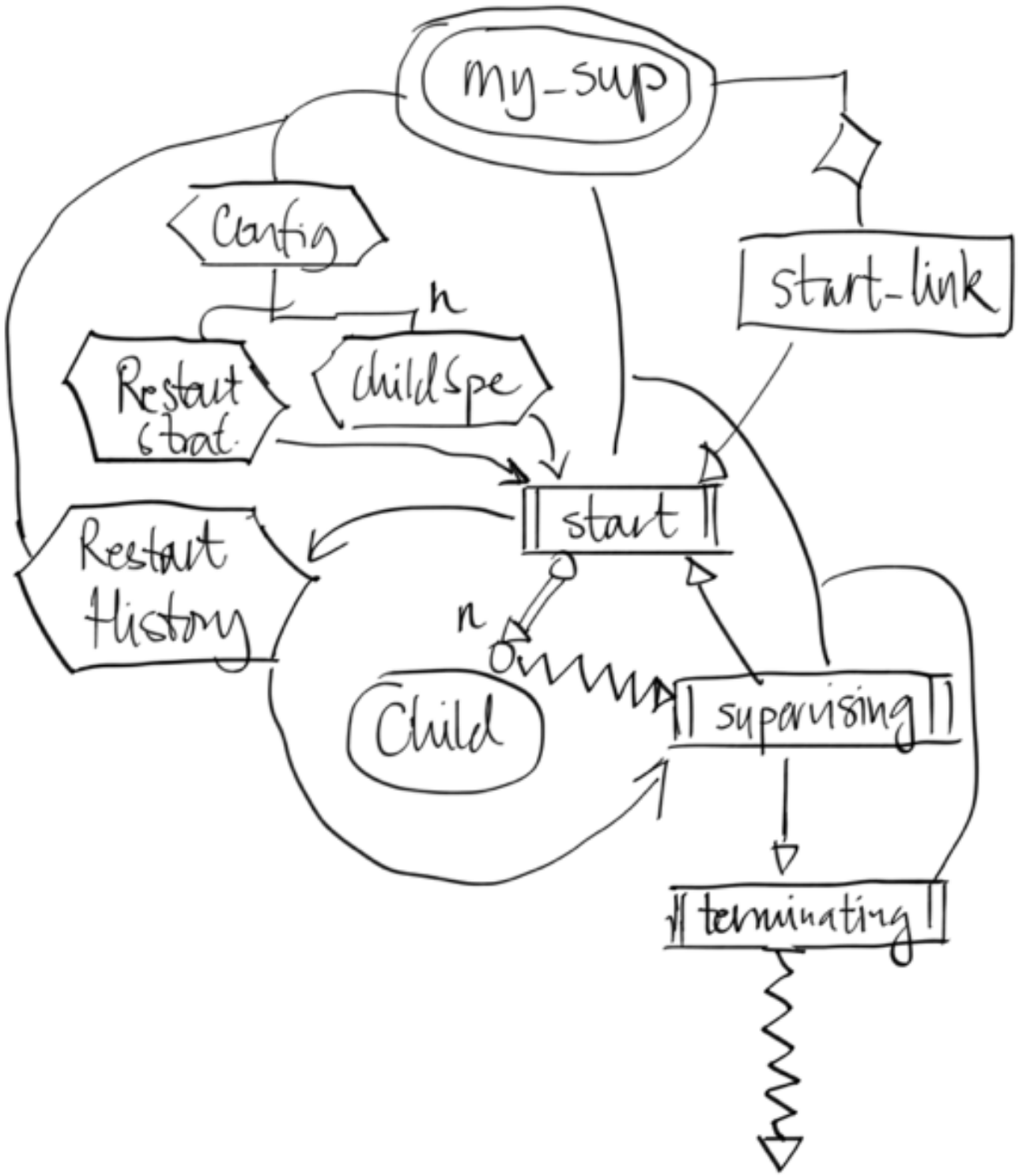
Specifies a default restart strategy

```
one_for_one
one_for_all
rest_for_one
simple_one_for_one
```

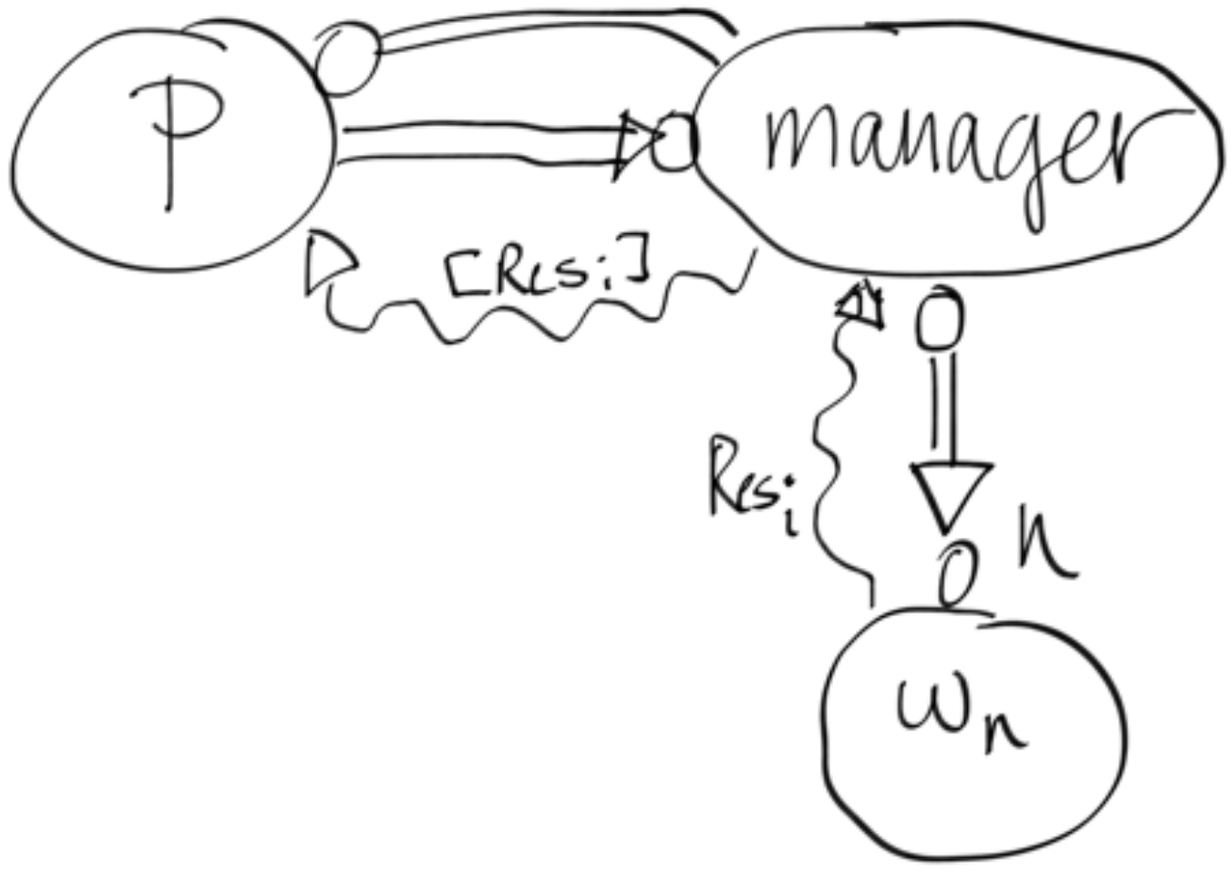
Child spec for how a child is restarted

```
permanent | transient | temporary
```

Supervisors



Simple Manager/Worker Pattern



Realities of software development

????

Source: http://www.thejournal.ie/readme/lunch-atop-skyscraper-photo-men-irish-shanaglish-518110-Jul2012/

Realities of software development

Product Owner

Source: http://www.thejournal.ie/readme/lunch-atop-skyscraper-photo-men-irish-shanaglish-518110-Jul2012/

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

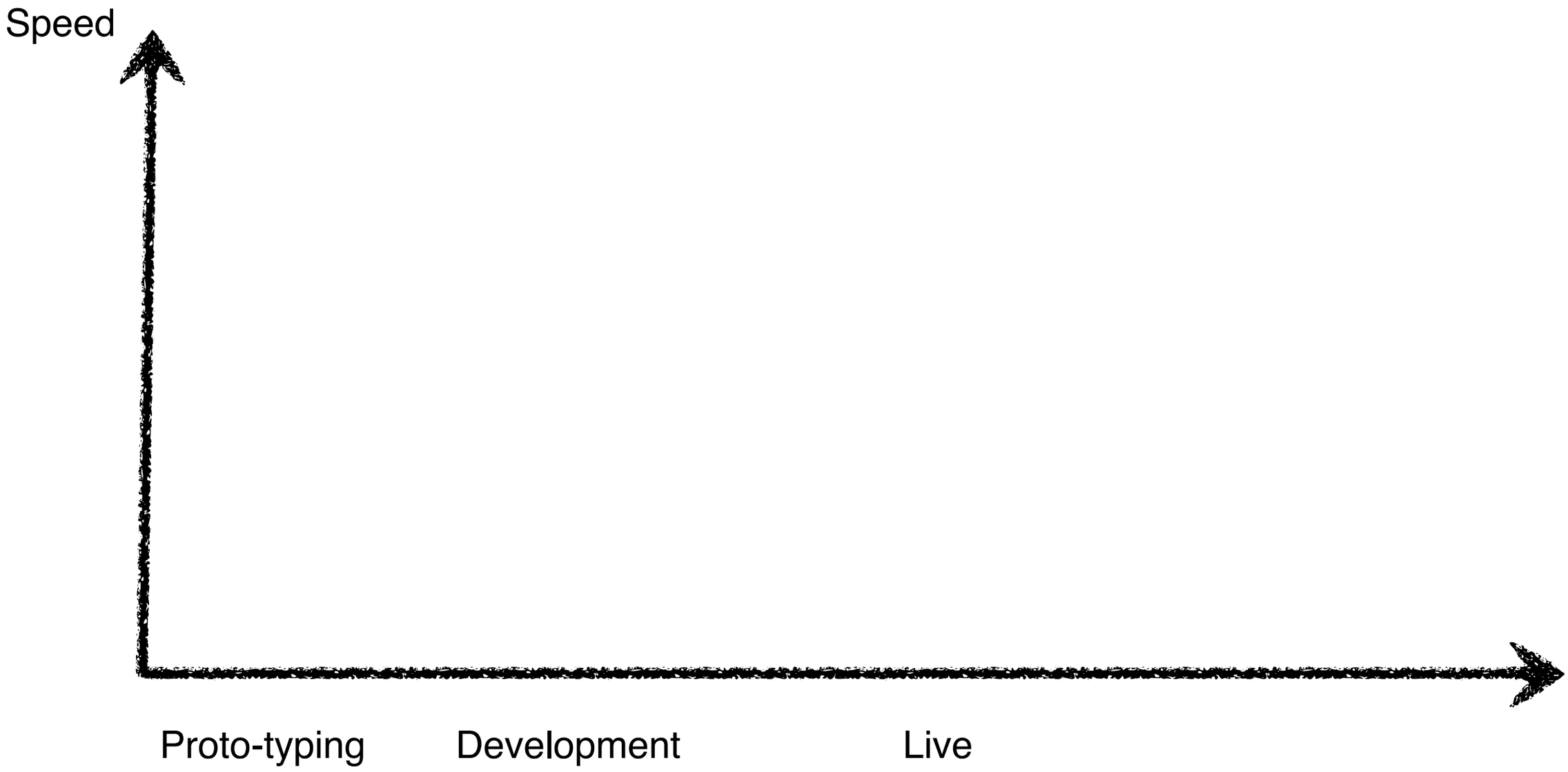
Preparing for battle

DANZA DI GUERRA DEI SIÙ

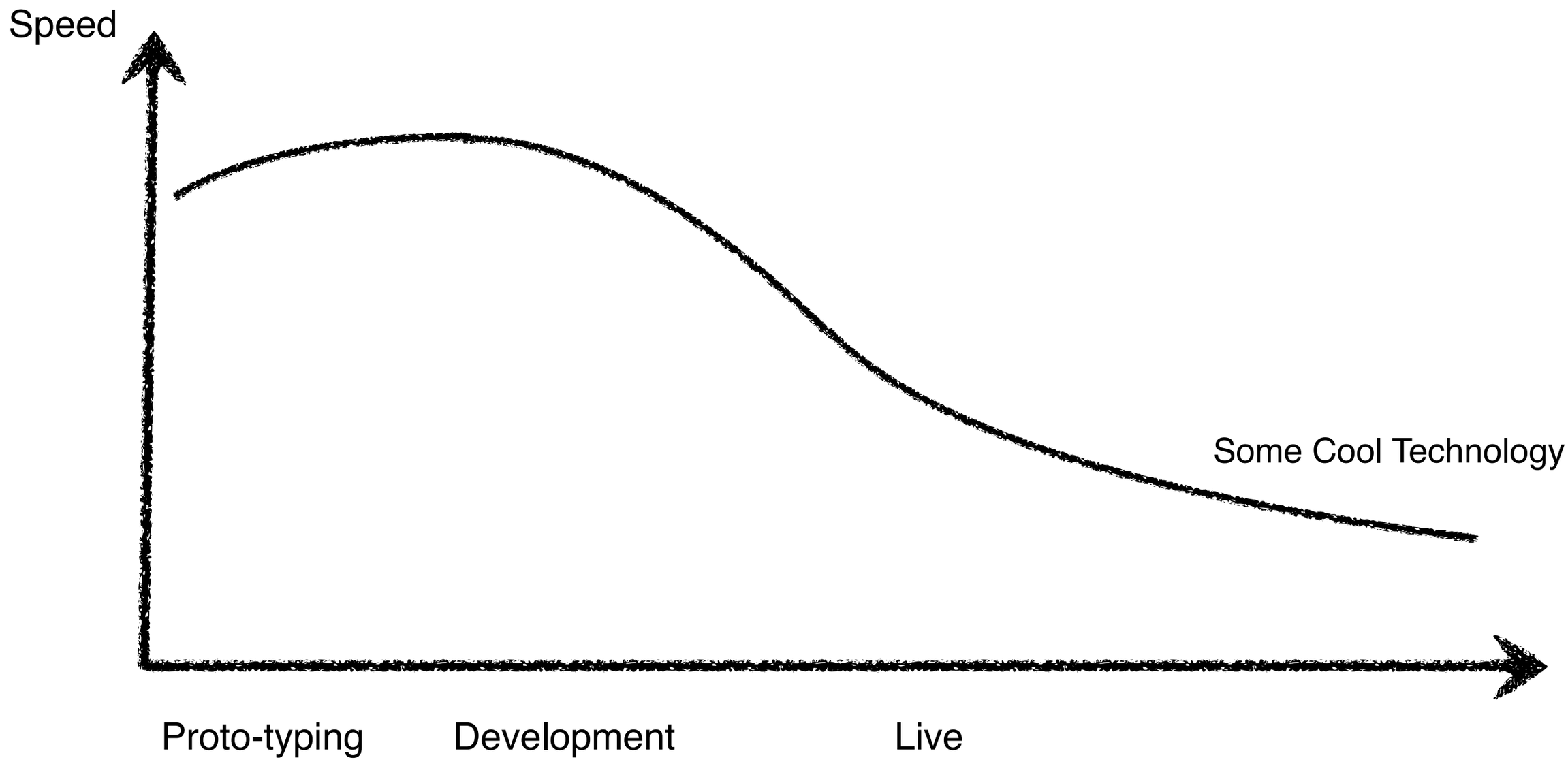
(Presa dal vero da Rodolfo Cronau).

Source: http://www.finerareprints.com/print_detail.html?stock_no=17122

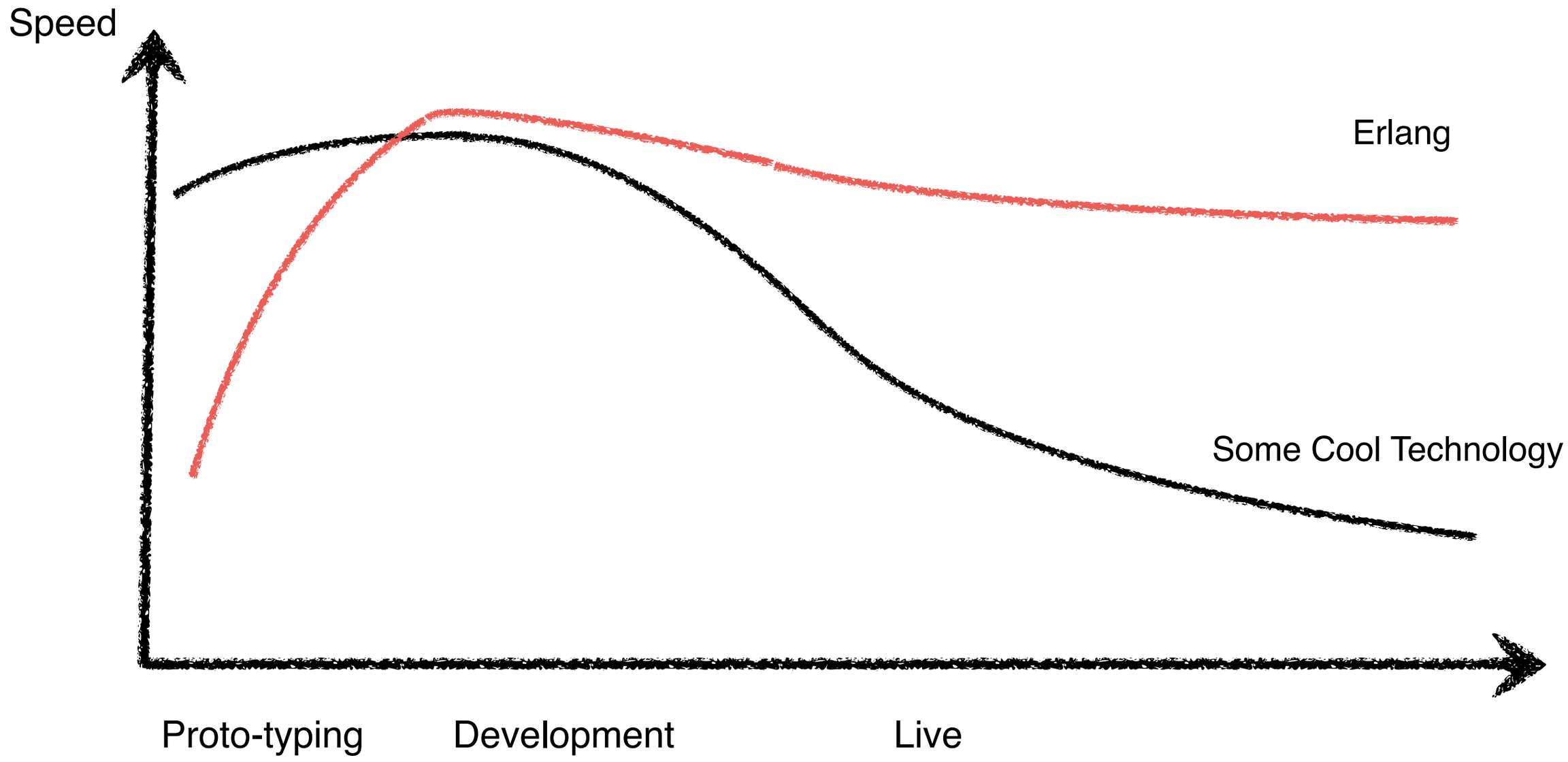
When do I get my ROI?



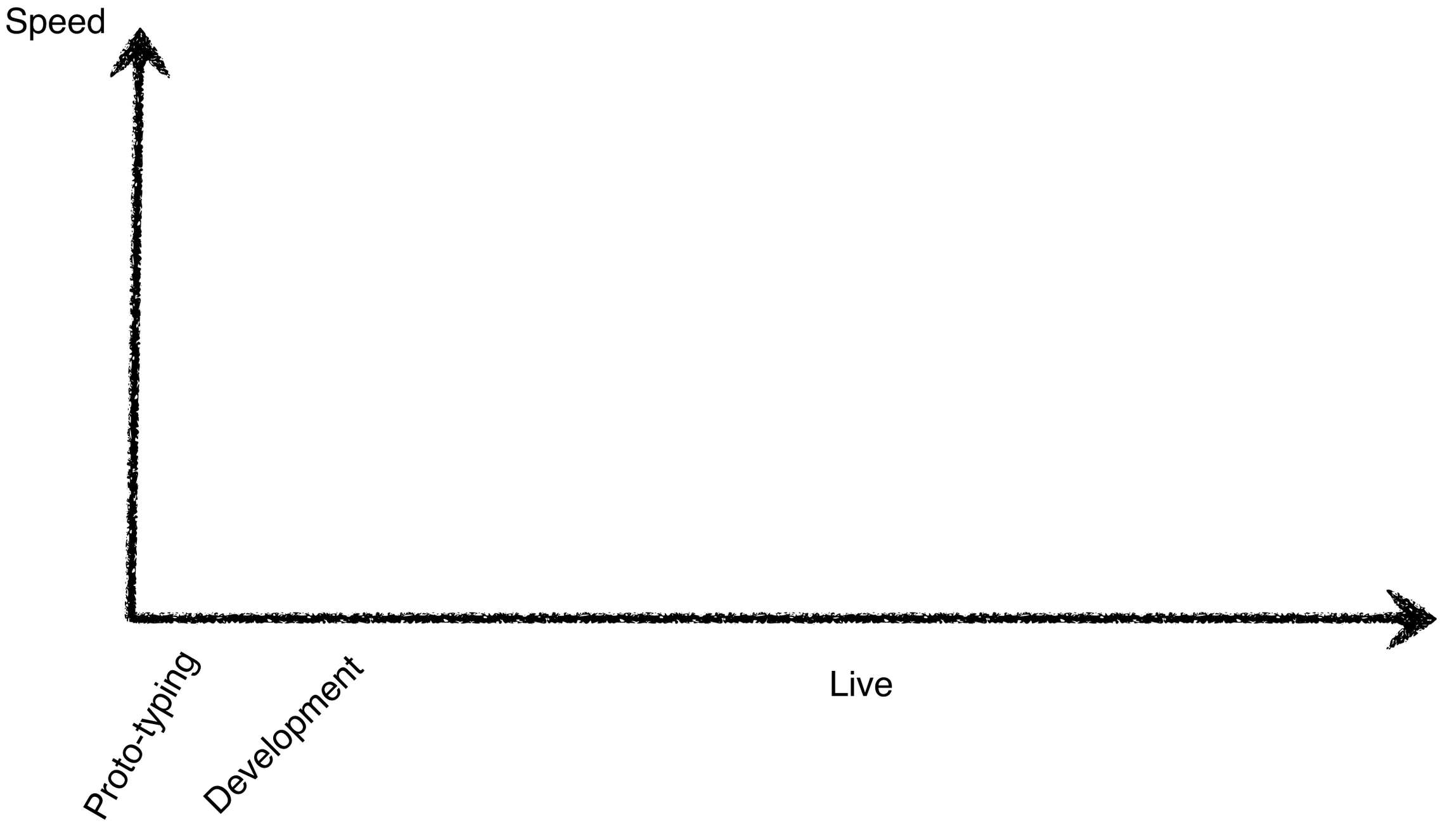
When do I get my ROI?



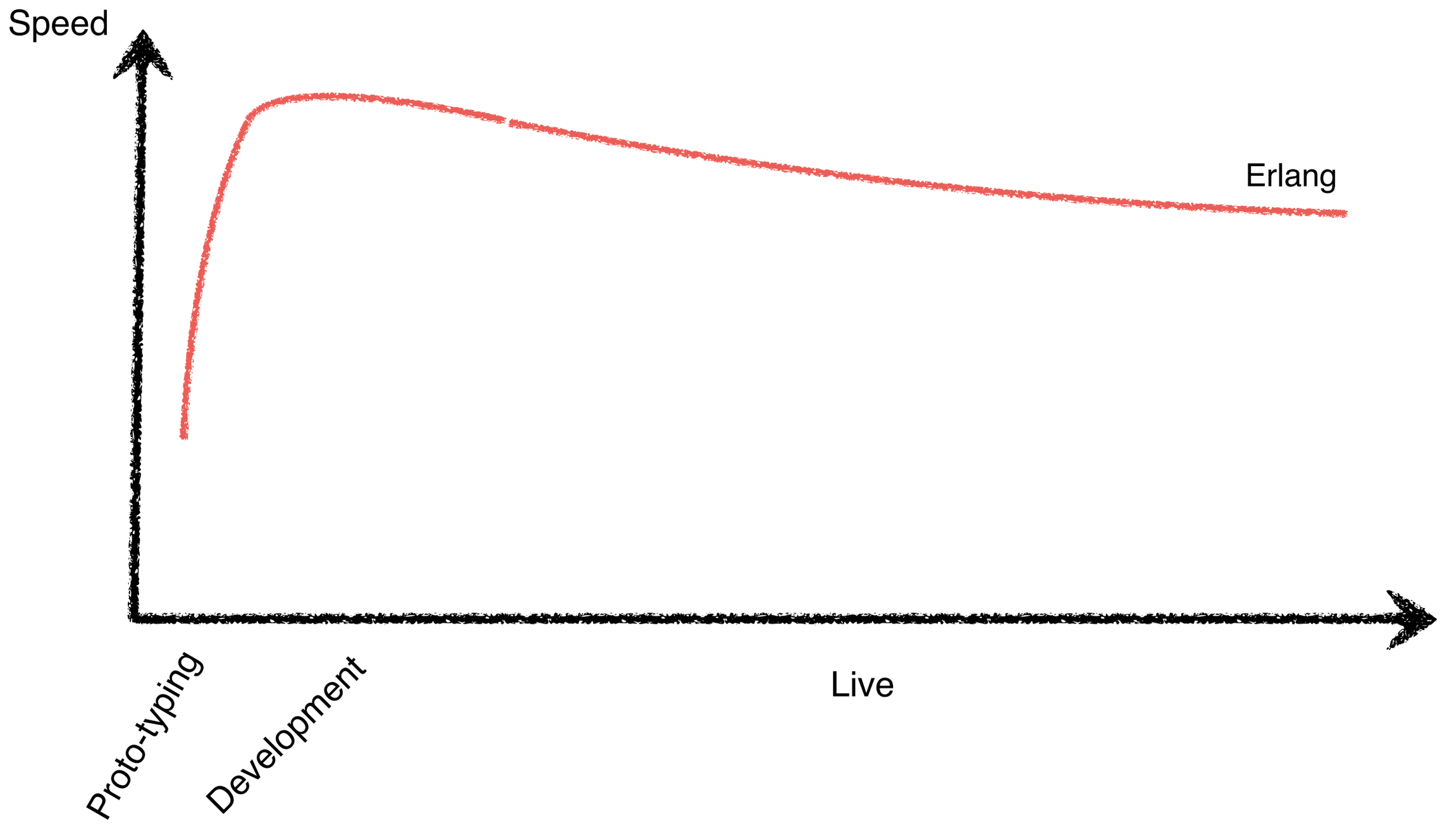
When do I get my ROI?



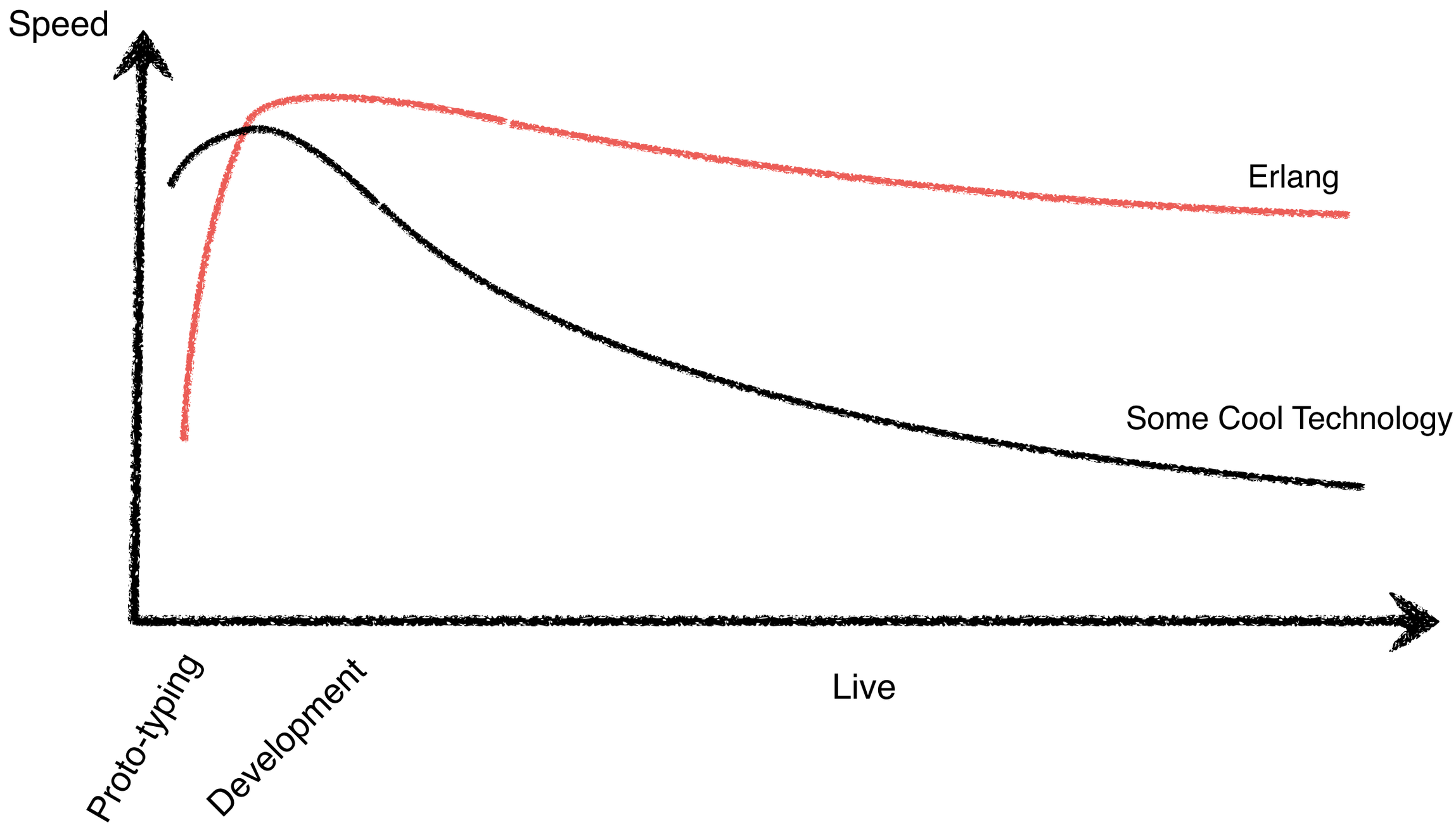
When do I get my ROI?



When do I get my ROI?



When do I get my ROI?



Erlang/OTP

Middleware

Middleware

Design Patterns

Middleware

Design Patterns

Fault Tolerance

Middleware

Design Patterns

Fault Tolerance

Distribution

Middleware

Design Patterns

Fault Tolerance

Distribution

Upgrades

Middleware

Design Patterns

Fault Tolerance

Distribution

Upgrades

Packaging

Middleware

Design Patterns

Fault Tolerance

Distribution

Upgrades

Packaging

OTP Tools

OTP Tools

Development

OTP Tools

Development

Test Frameworks

OTP Tools

Development

Test Frameworks

Release & Deployment

OTP Tools

Development

Test Frameworks

Release & Deployment

Debugging & Monitoring

OTP Benefits

OTP Benefits

Less Code

OTP Benefits

Less Code

Less Bugs

OTP Benefits

Less Code

Less Bugs

More Solid Code

OTP Benefits

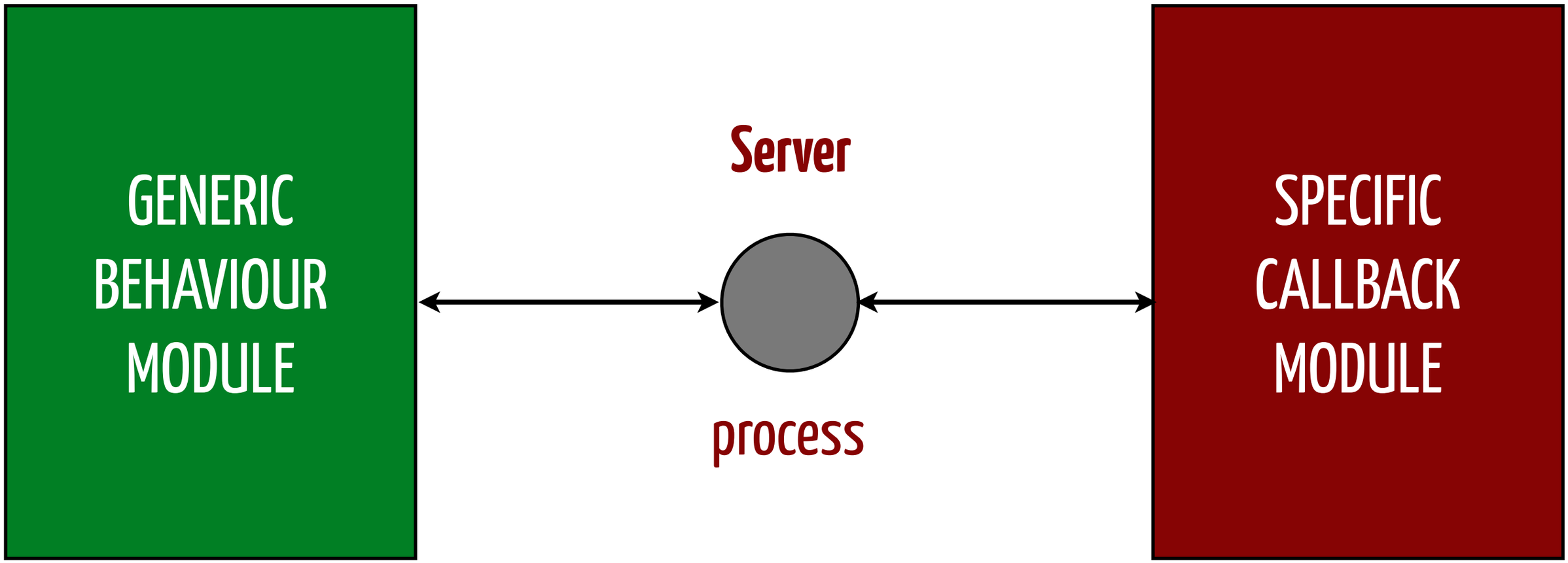
Less Code

Less Bugs

More Solid Code

More Tested Code More Free Time

Behaviours


GENERIC
BEHAVIOUR
MODULE
Server
process
SPECIFIC
CALLBACK
MODULE

OTP Behaviours

OTP Behaviours

Servers

OTP Behaviours

Servers

Finite State Machines

OTP Behaviours

Servers

Finite State Machines

Event Handlers

OTP Behaviours

Servers

Finite State Machines

Event Handlers

Supervisors

OTP Behaviours

Servers

Finite State Machines

Event Handlers

Supervisors

Applications

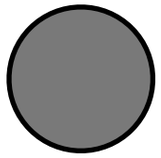
Nasty Things

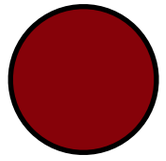
Handled by OTP

Nasty Things Handled by OTP

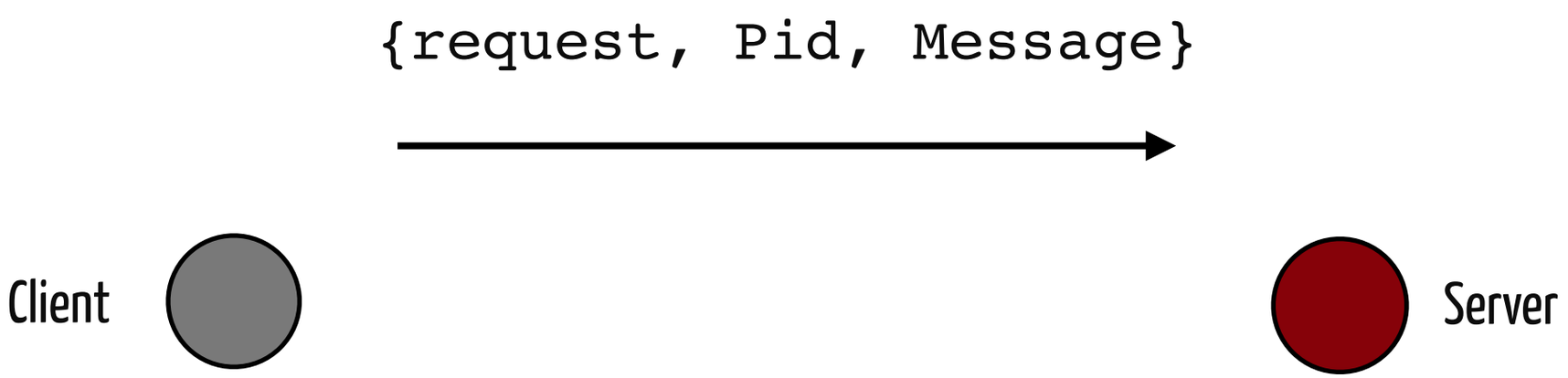
Who are you gonna call?

Client

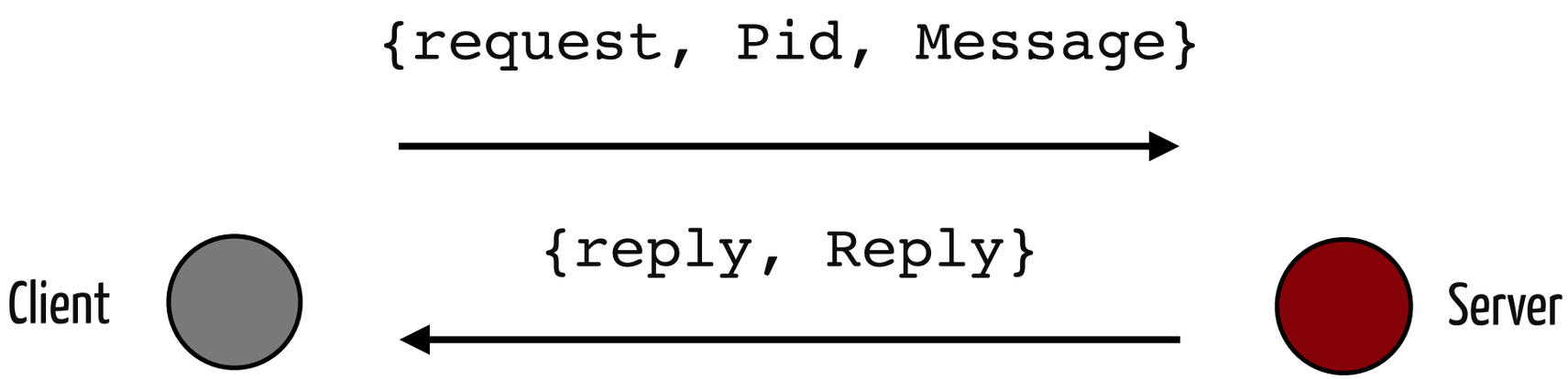




Server

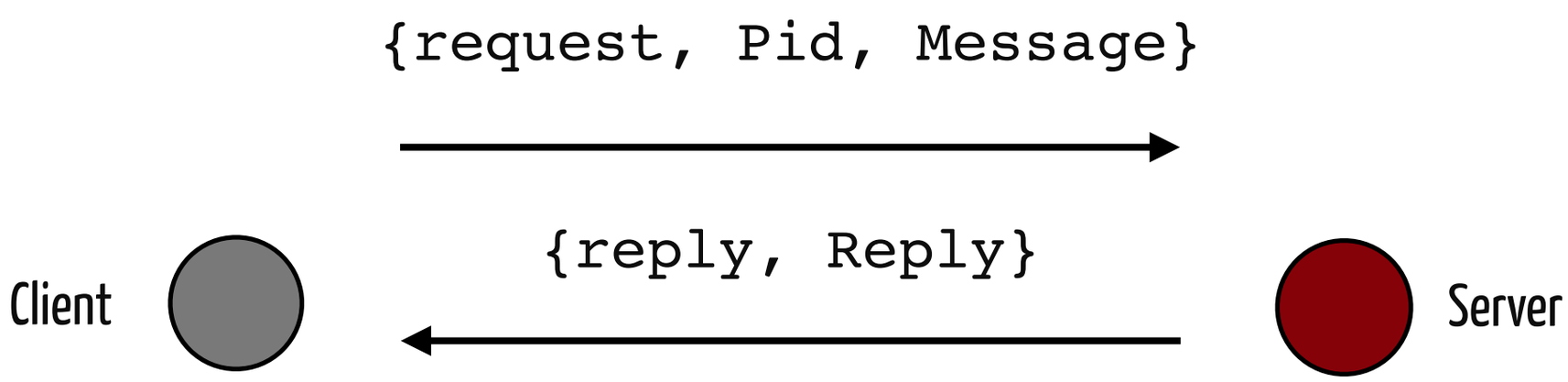


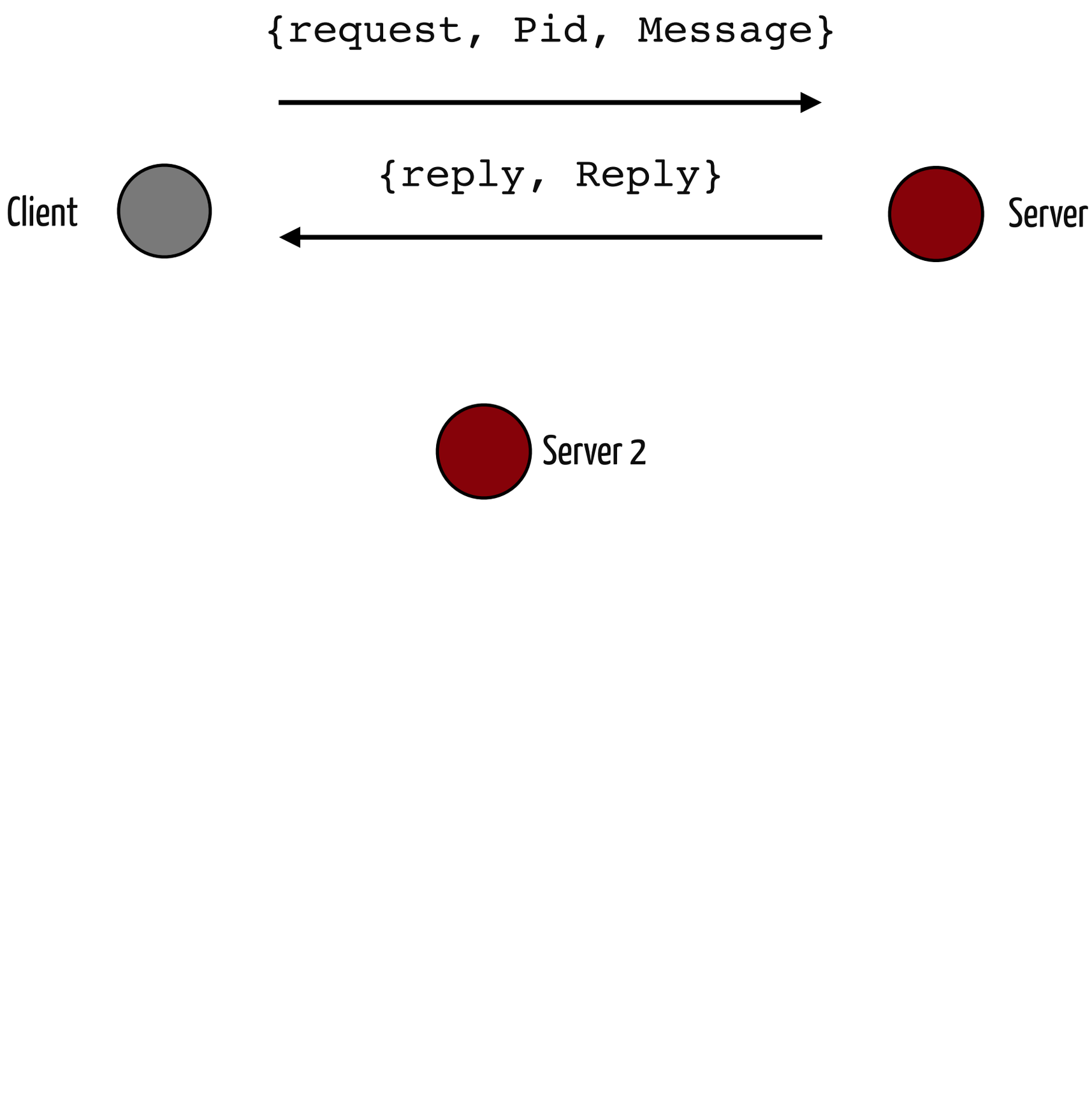

```
call(Name, Message) ->
    Name ! {request, self(), Message},
    receive
        {reply, Reply} -> Reply
    end.
```

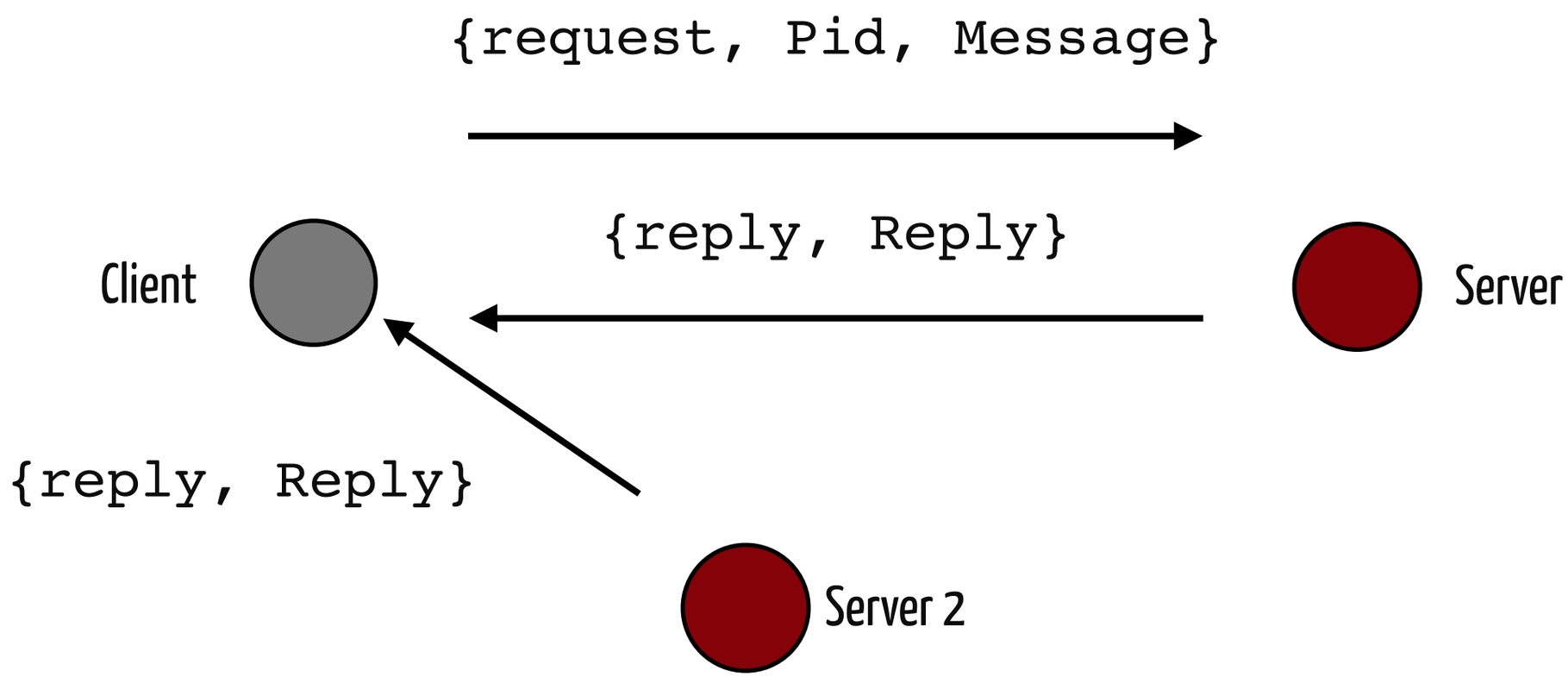


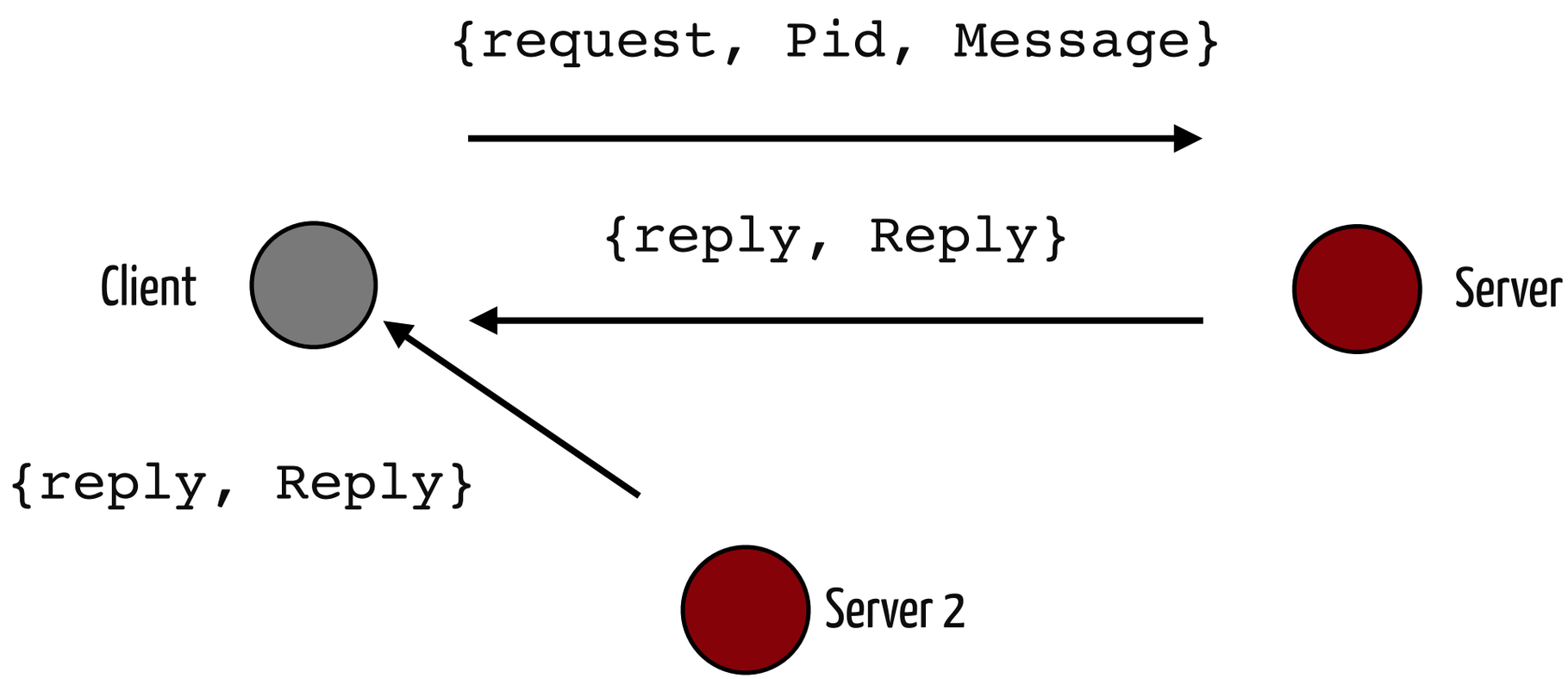

```
call(Name, Message) ->
    Name ! {request, self(), Message},
    receive
        {reply, Reply} -> Reply
    end.
```

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

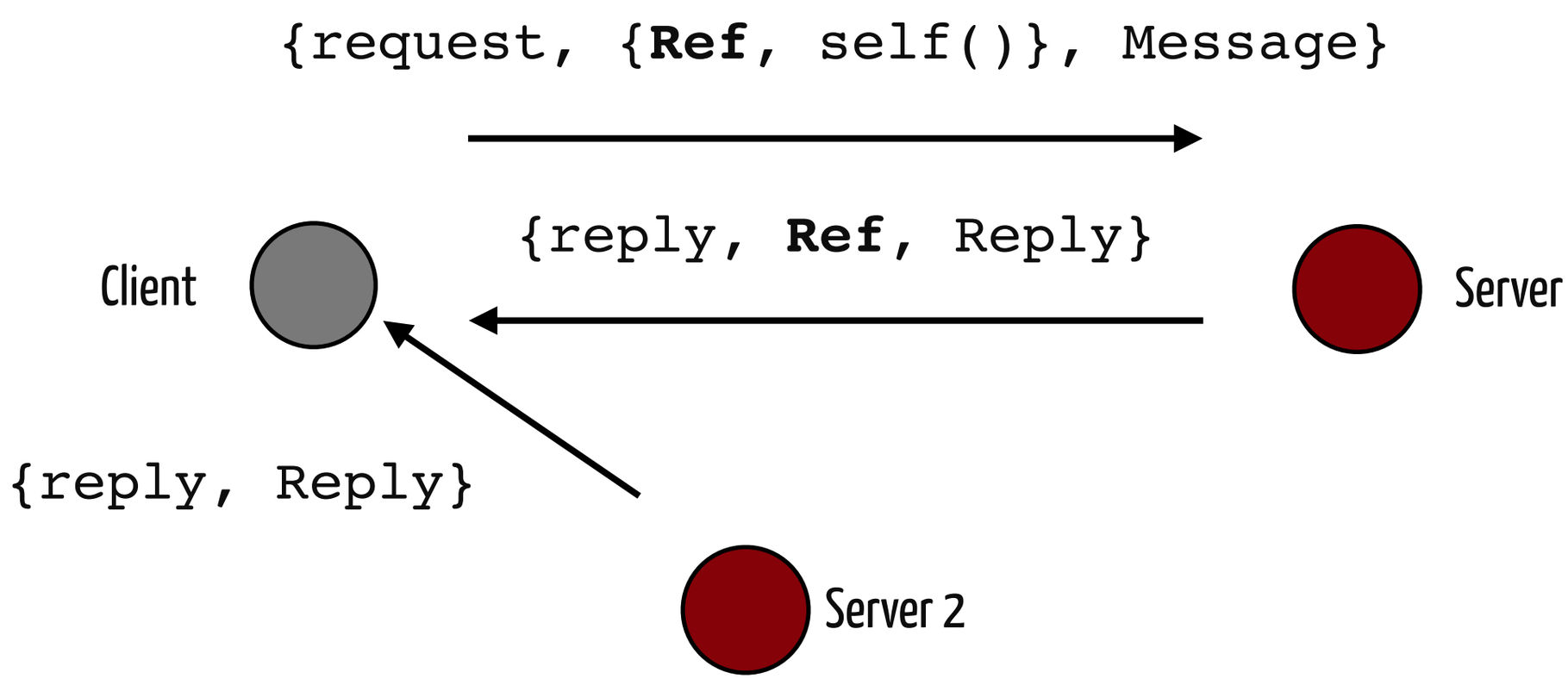

{request, Pid, Message}
{reply, Reply}
Client
Server


{request, Pid, Message}
{reply, Reply}
Client
Server
Server 2

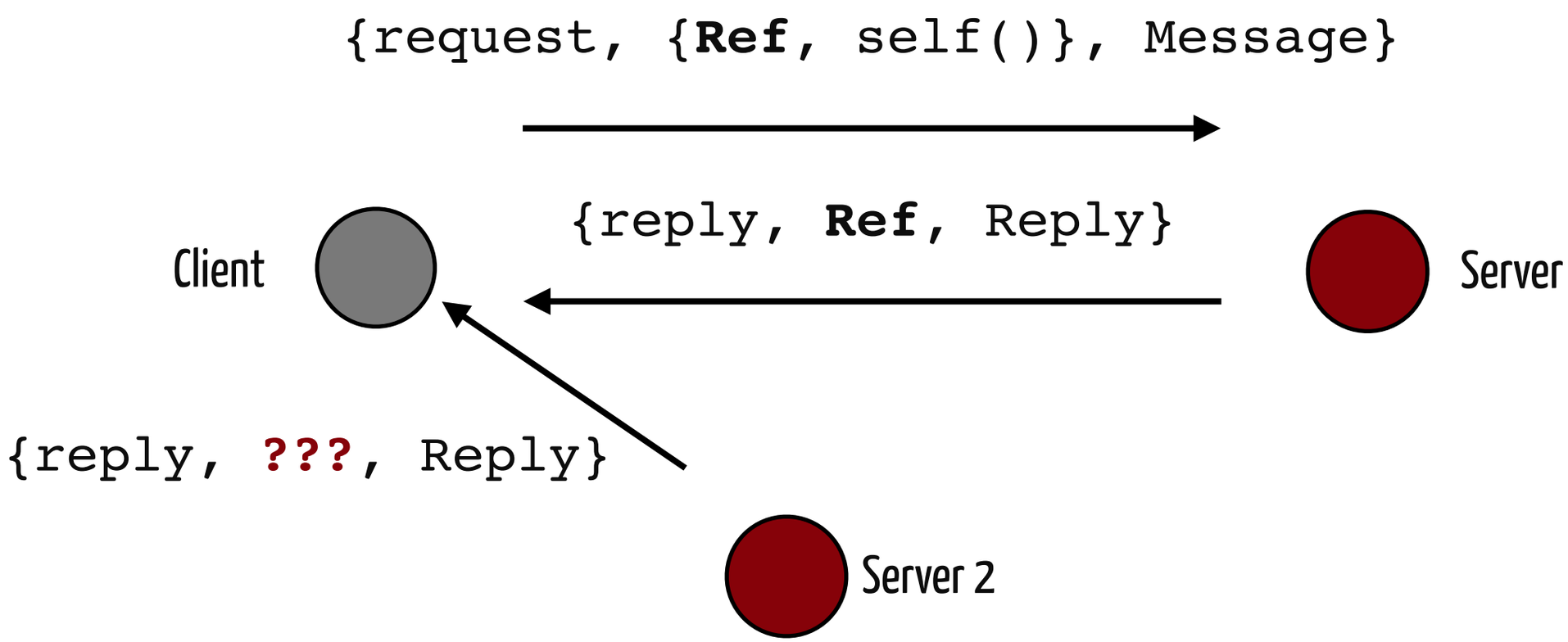

{request, Pid, Message}
{reply, Reply}
Client
Server
{reply, Reply}
Server 2




```
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}.
```

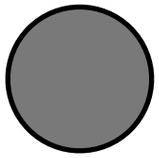


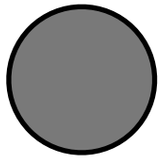

```
call(Name, Msg) ->
    Ref = make_ref(),
    Name ! {request, {Ref, self()},    Msg},
    receive {reply, Ref, Reply} -> Reply end.

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



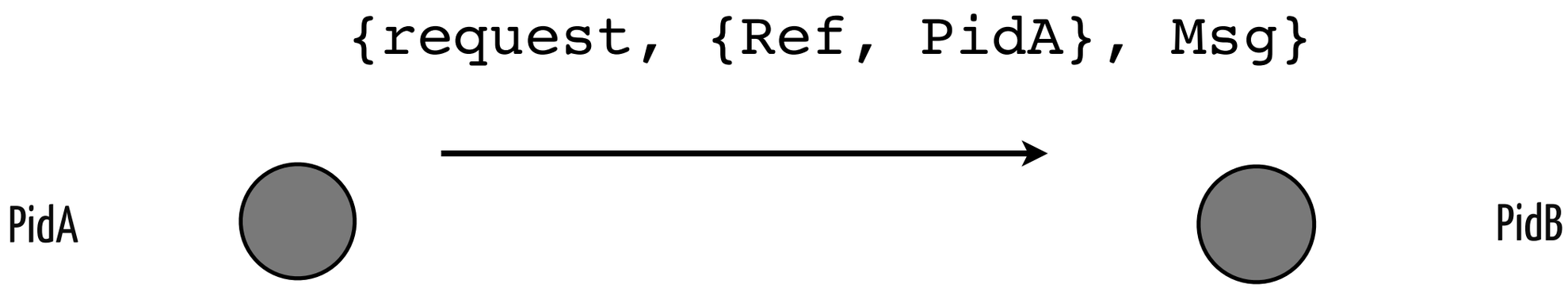

```
call(Name, Msg) ->
    Ref = make_ref(),
    Name ! {request, {Ref, self()},    Msg},
    receive {reply, Ref, Reply} -> Reply end.

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

PidA

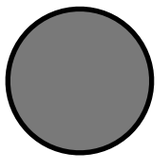




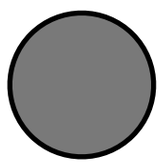
PidB


{request, {Ref, PidA}, Msg}
PidA
PidB

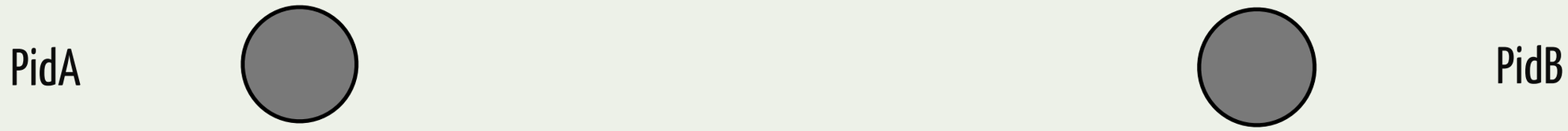
PidA

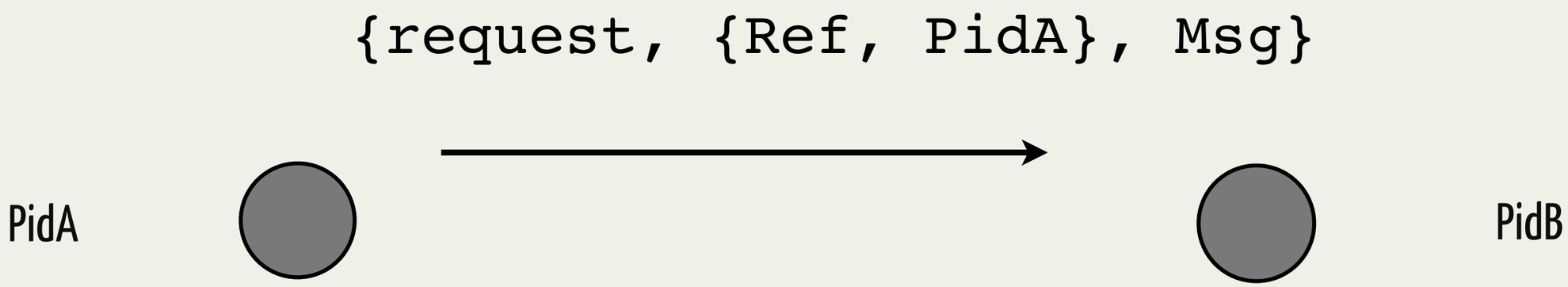


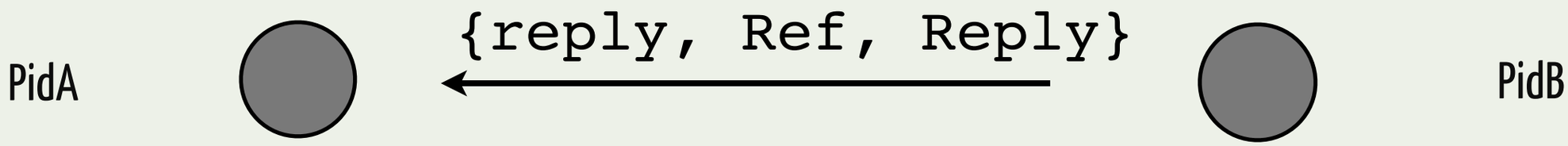
PidA



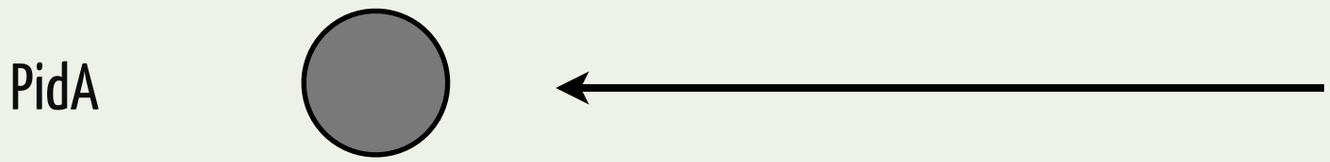
```
call(Name, Msg) ->
    Ref = erlang:monitor(process, Name),
    Name ! {request, {Ref, self()}, Msg},
    receive
      {reply, Ref, Reply} ->
        erlang:demonitor(Ref),
        Reply;
      {'DOWN', Ref, process, _Name, _Reason} ->
        {error, no_proc}
    end.
```


PidA
PidB

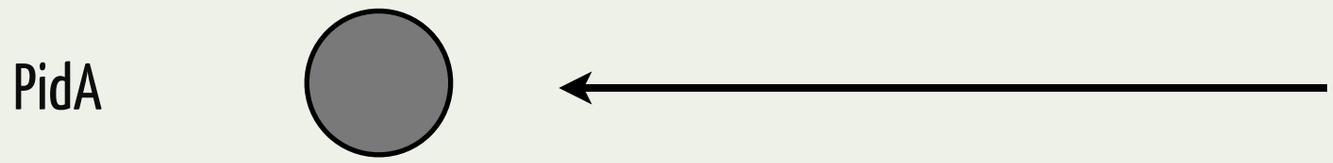

{request, {Ref, PidA}, Msg}
PidA
PidB


{reply, Ref, Reply}
PidA
PidB

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



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




```
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.
```

Dealing with deadlocks

Dealing with deadlocks

7 years of coding Erlang

Dealing with deadlocks

7 years of coding Erlang

Time spent on deadlock issues….

Dealing with deadlocks

7 years of coding Erlang

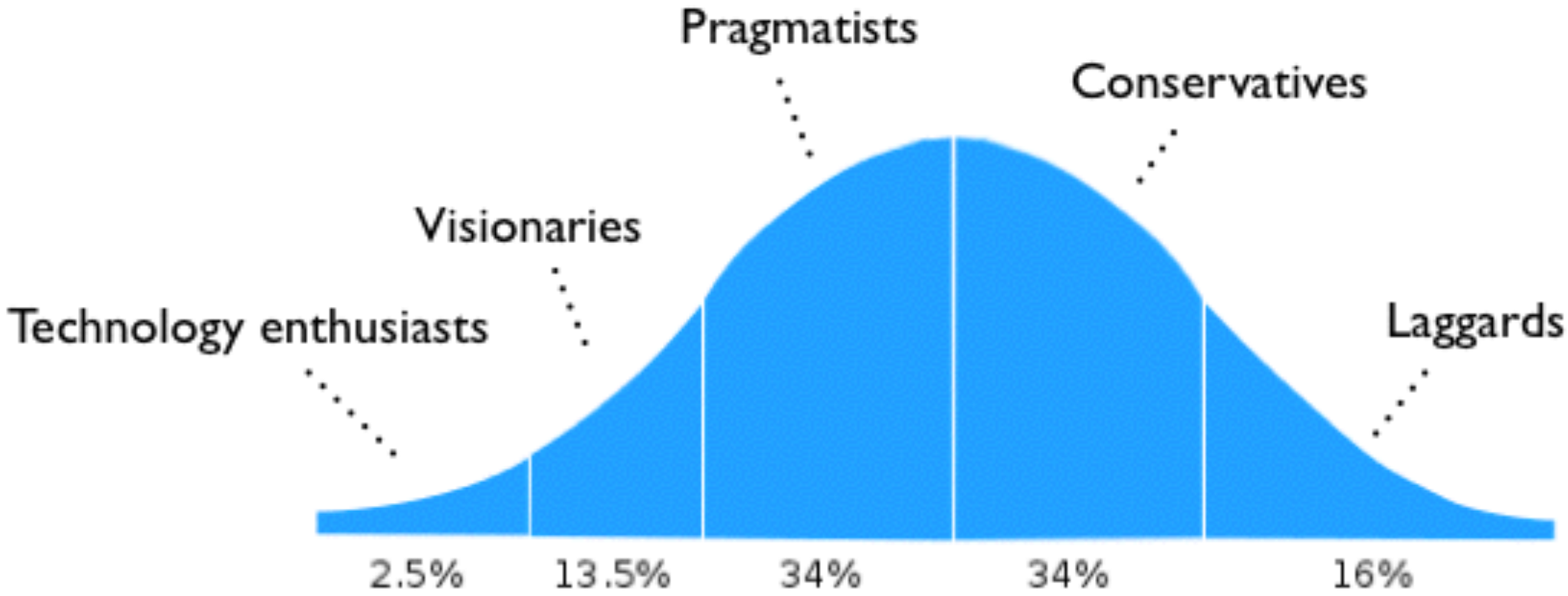
Time spent on deadlock issues….

1 hour (due to lack of experience with OTP)

Selling Others on Erlang

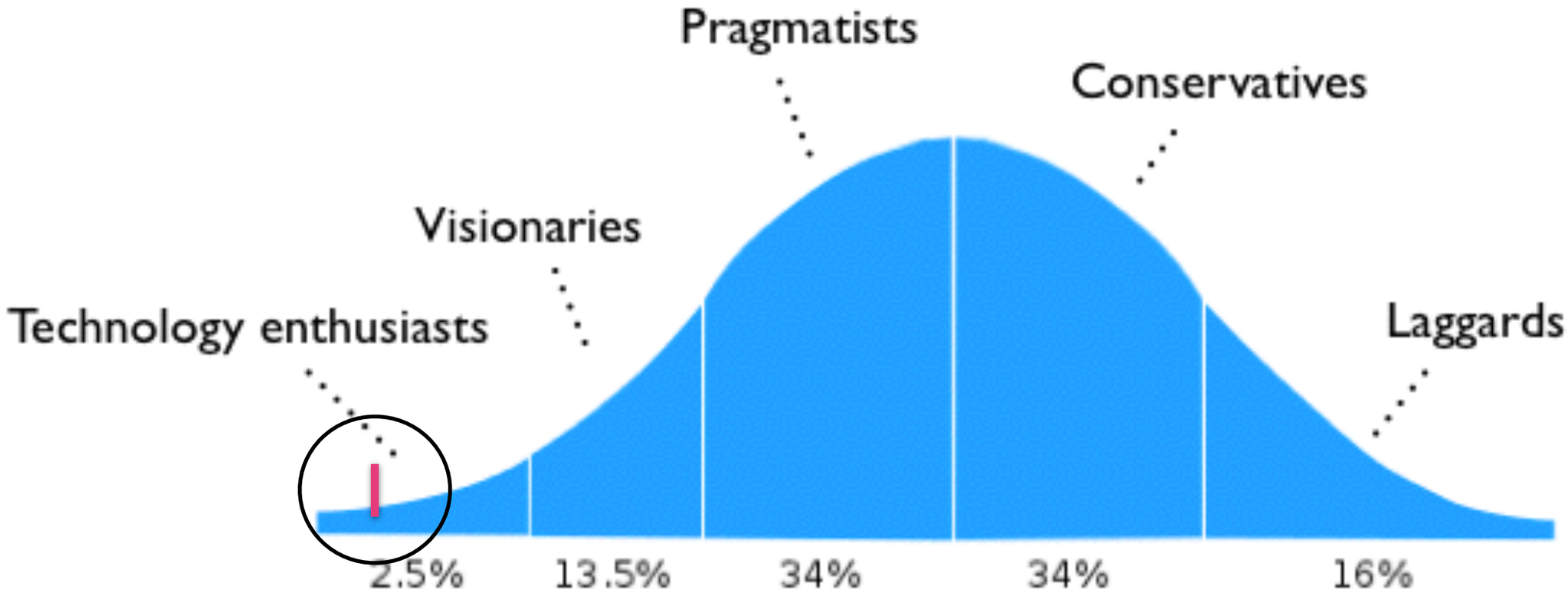
In Theory...

You conquer the TALC group by group in one smooth motion



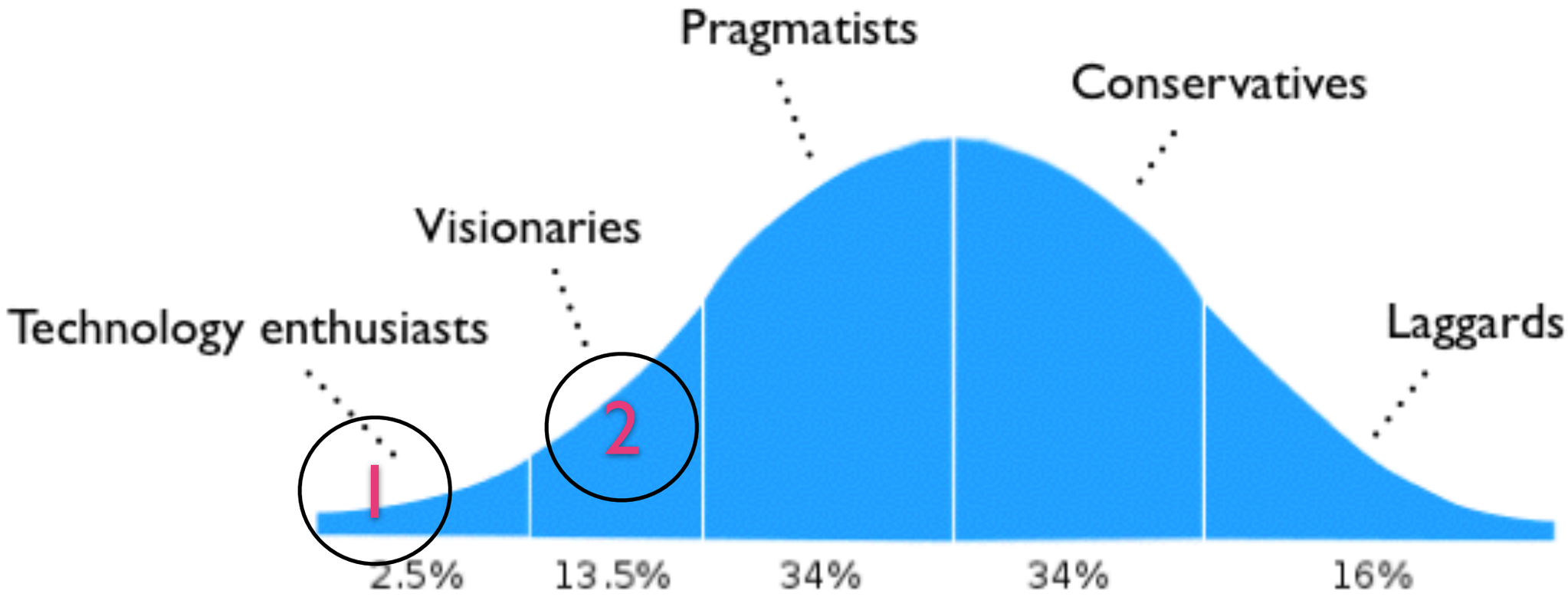
In Theory...

You conquer the TALC group by group in one smooth motion



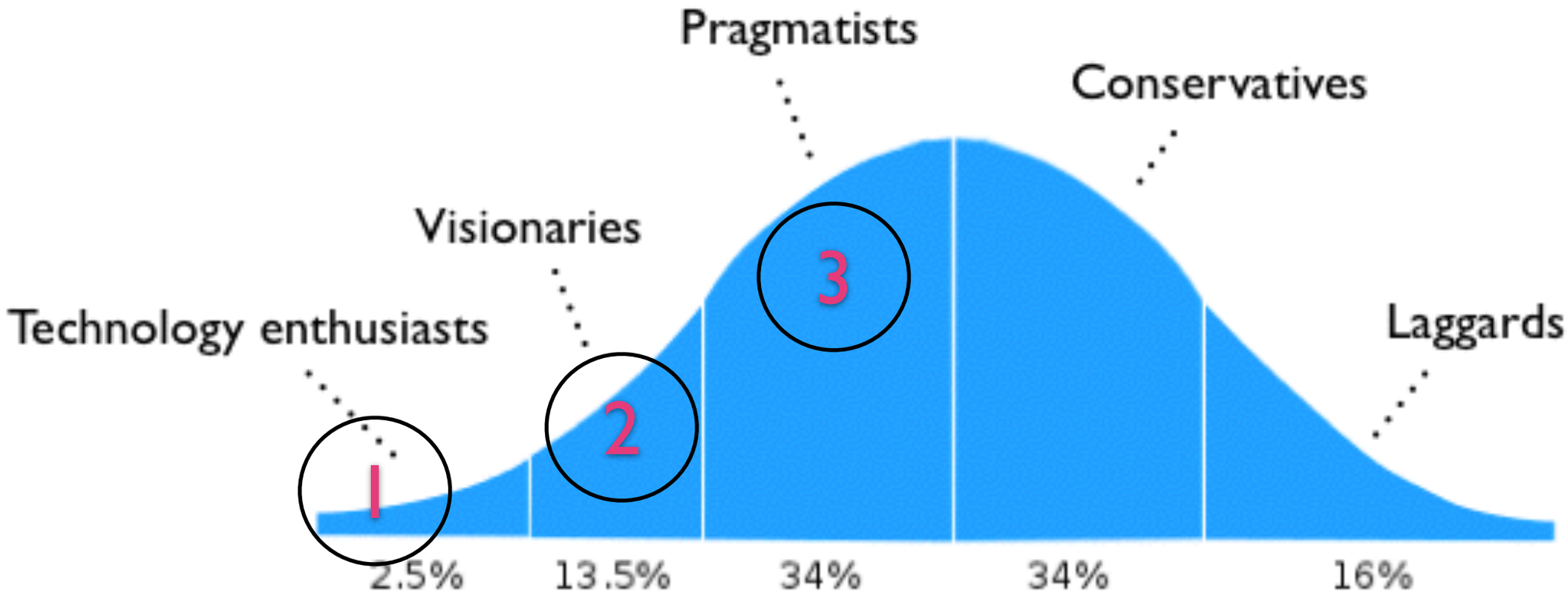
In Theory...

You conquer the TALC group by group in one smooth motion



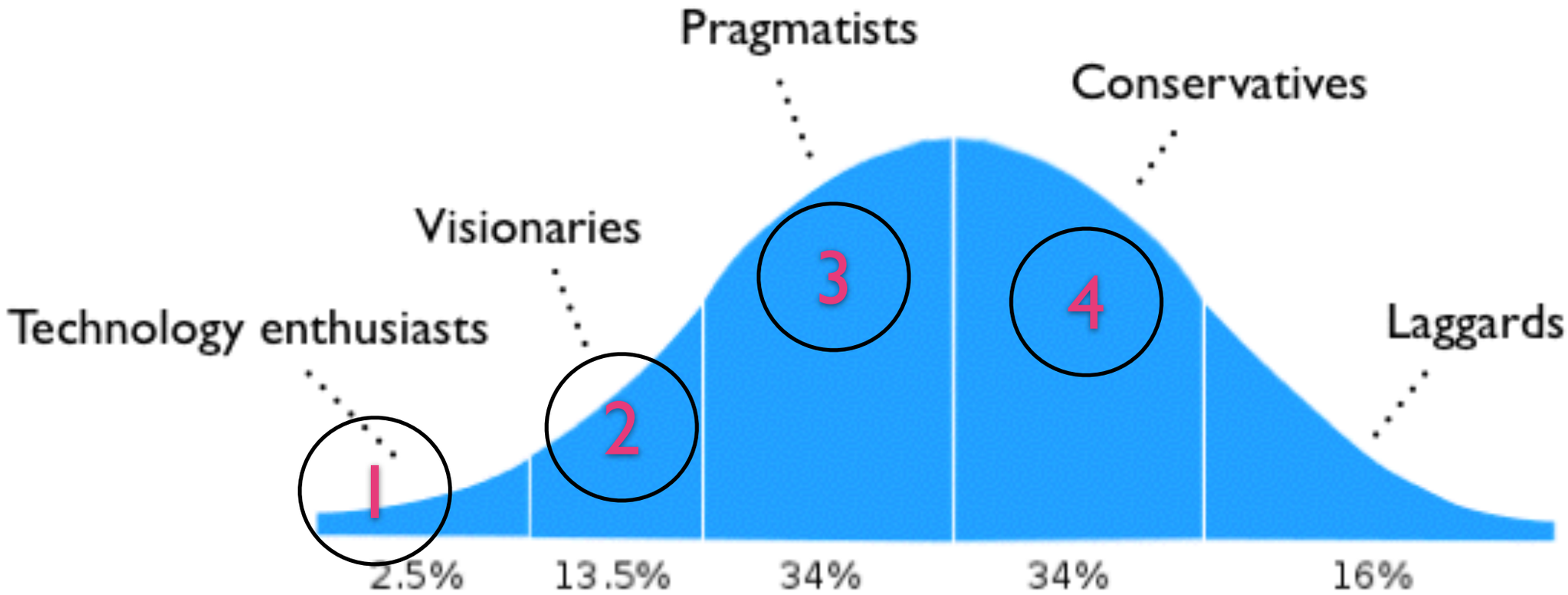
In Theory...

You conquer the TALC group by group in one smooth motion



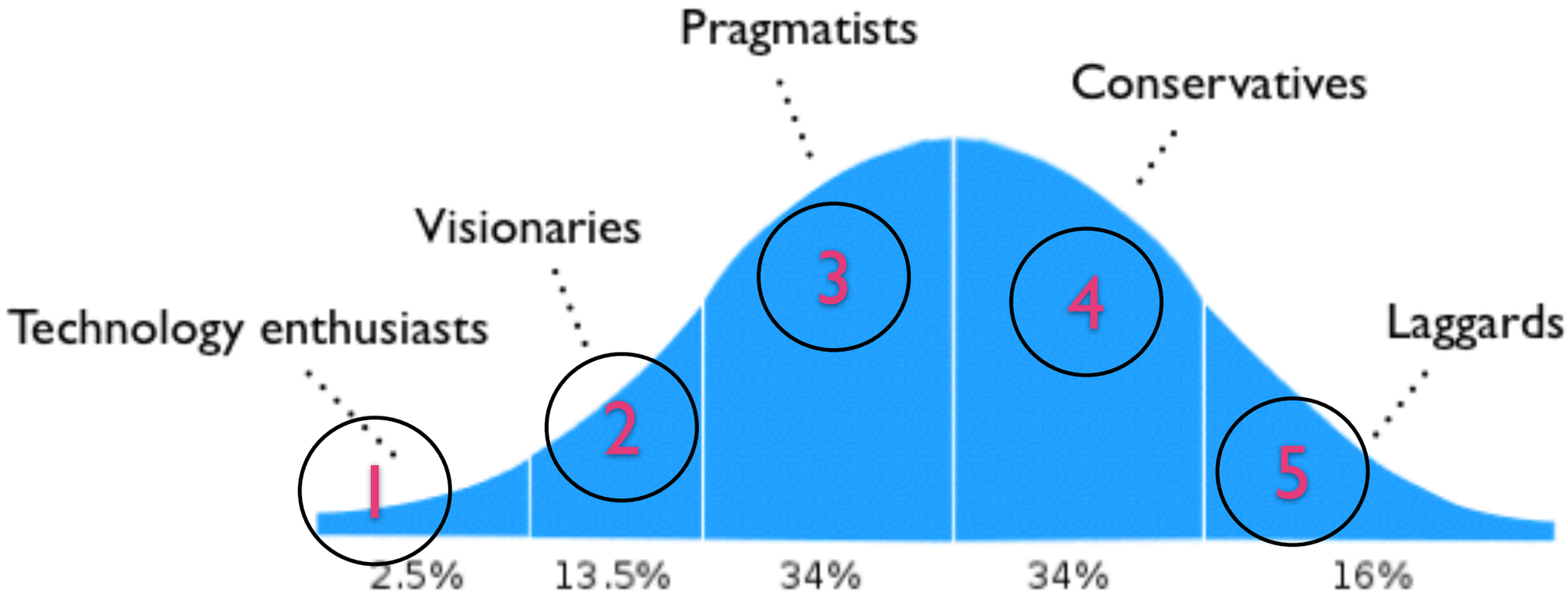
In Theory...

You conquer the TALC group by group in one smooth motion



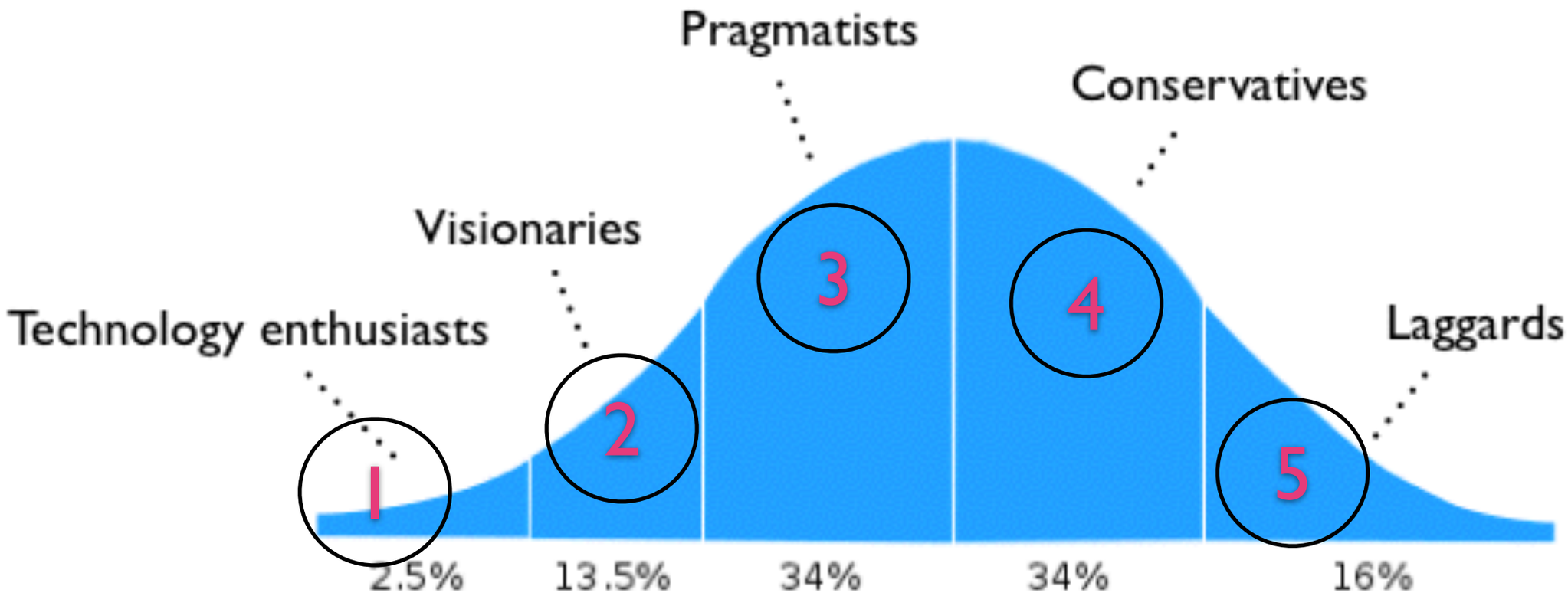
In Theory...

You conquer the TALC group by group in one smooth motion



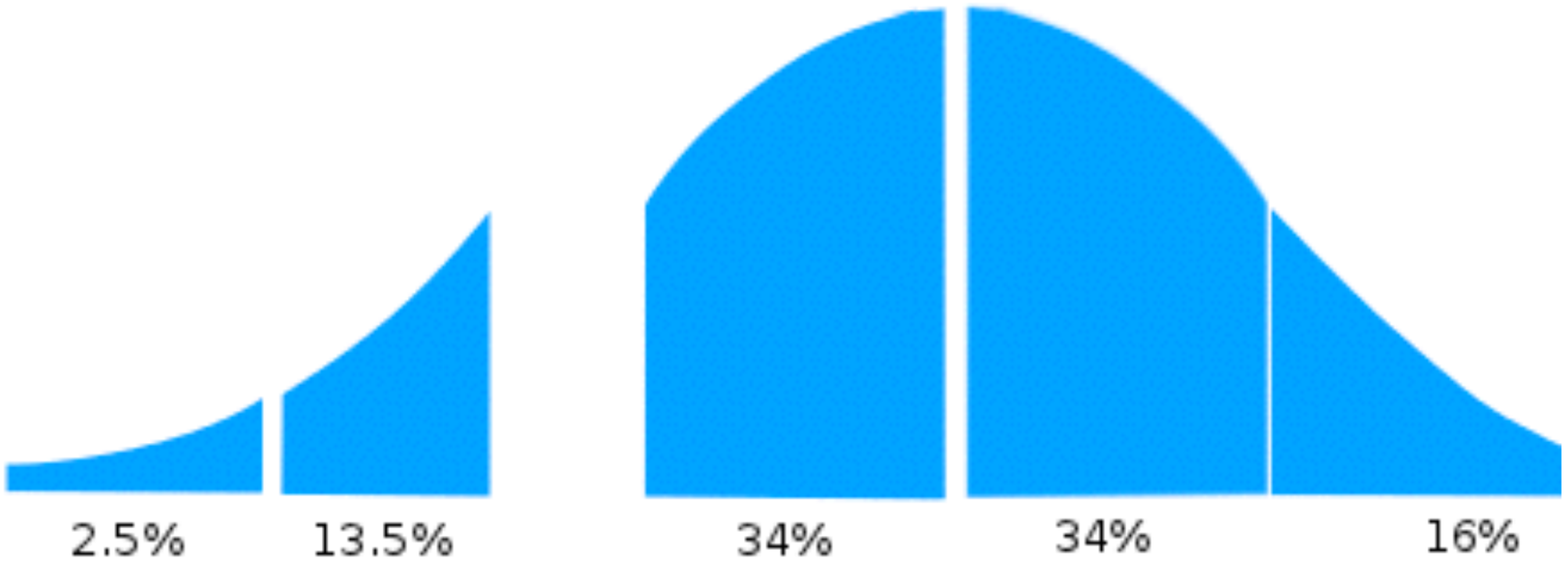
In Theory...

You conquer the TALC group by group in one smooth motion

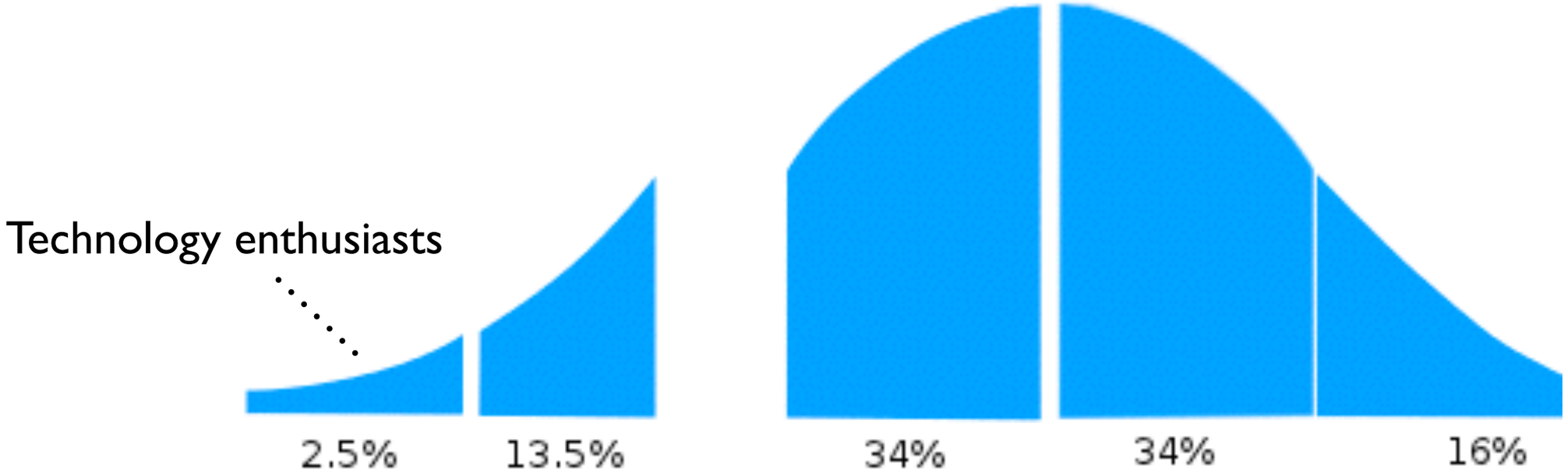


but it's just an illusion :-(

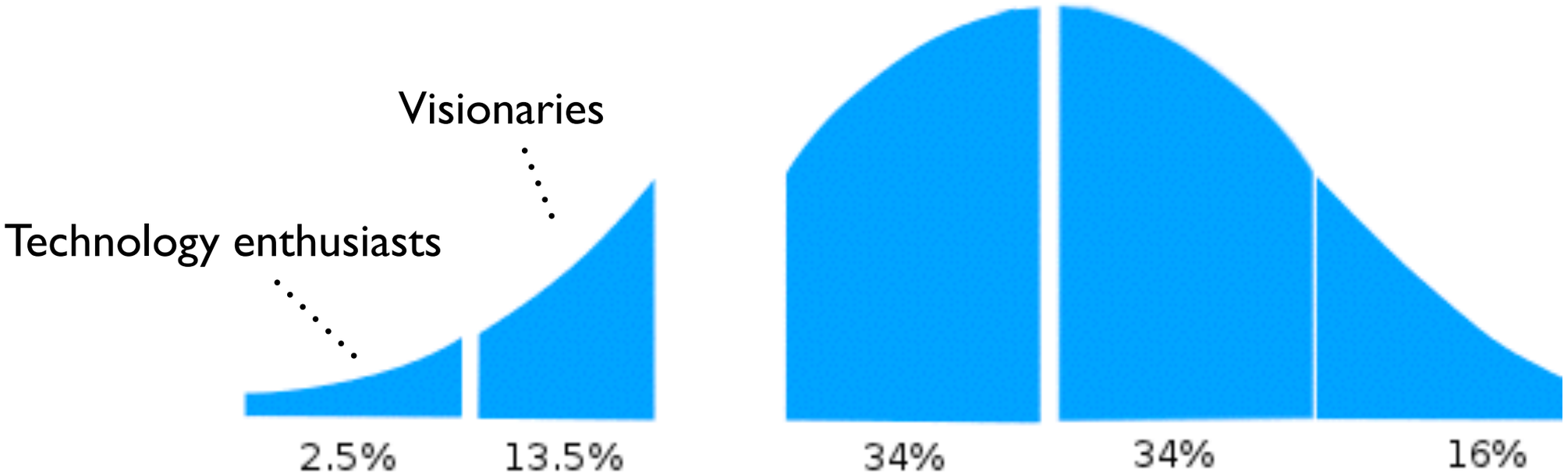
Cracks and a Chasm



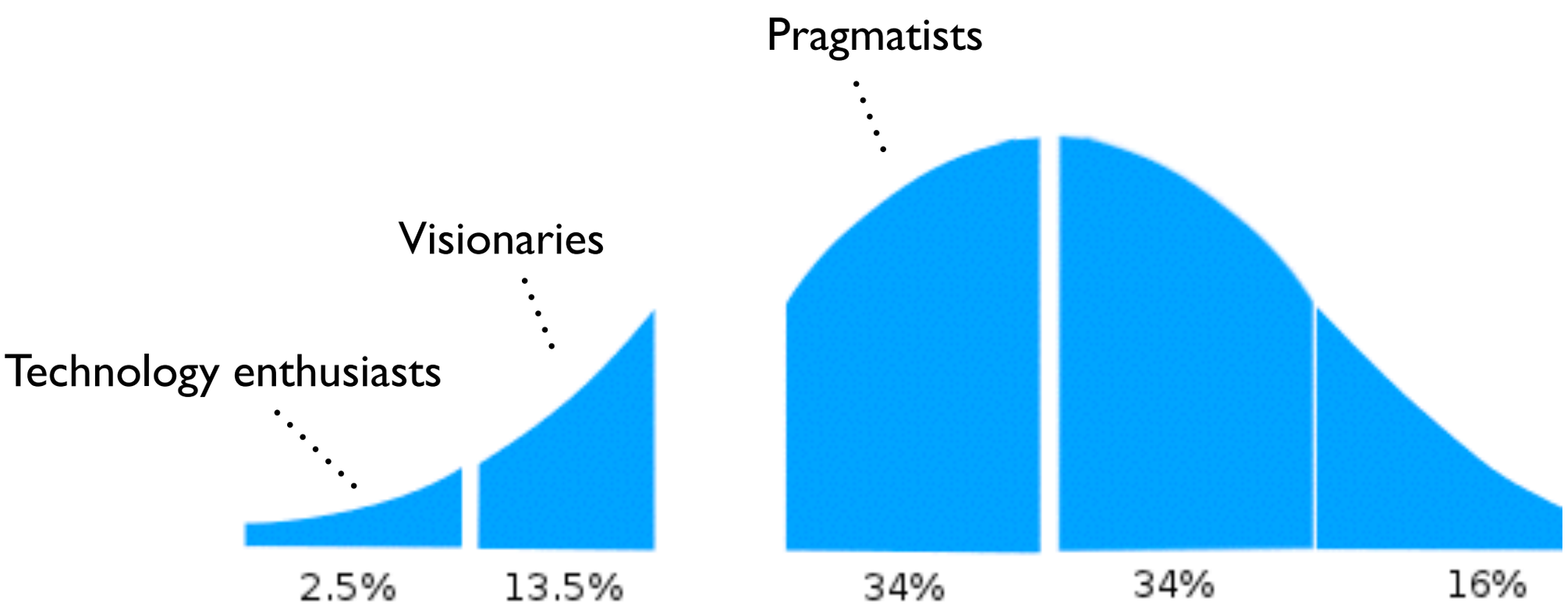
Cracks and a Chasm



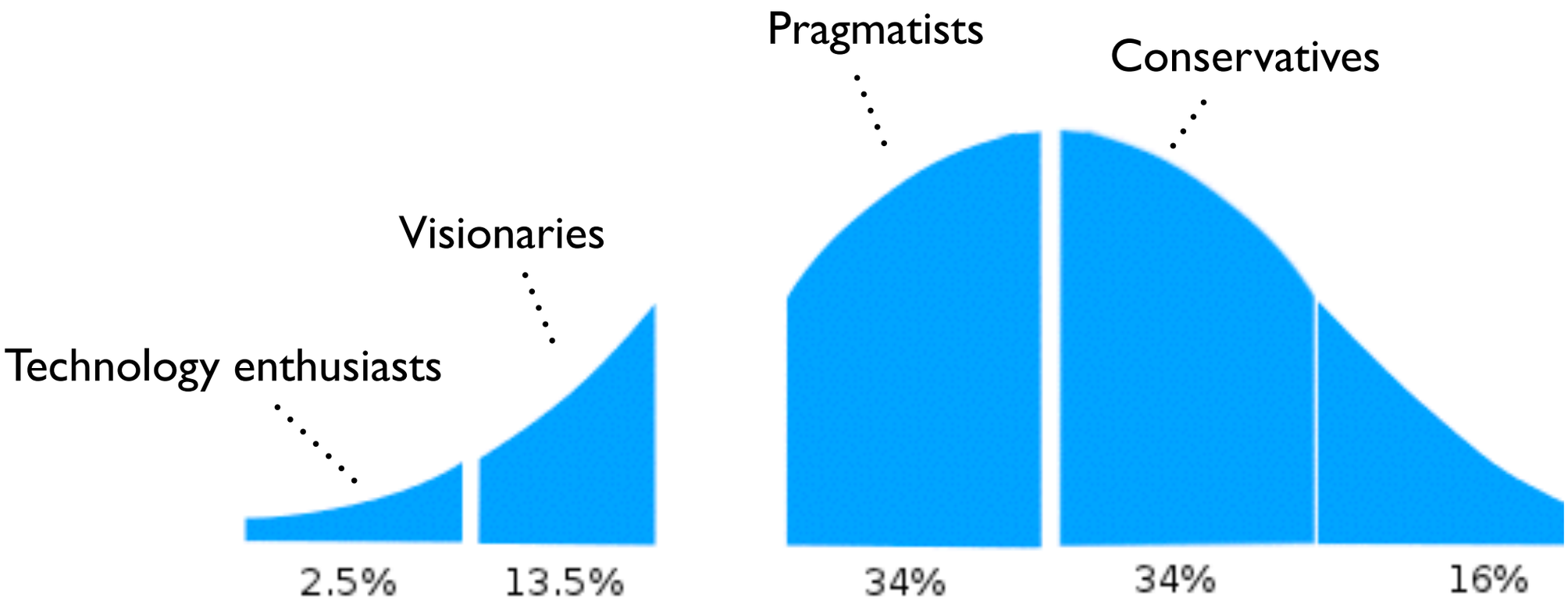
Cracks and a Chasm



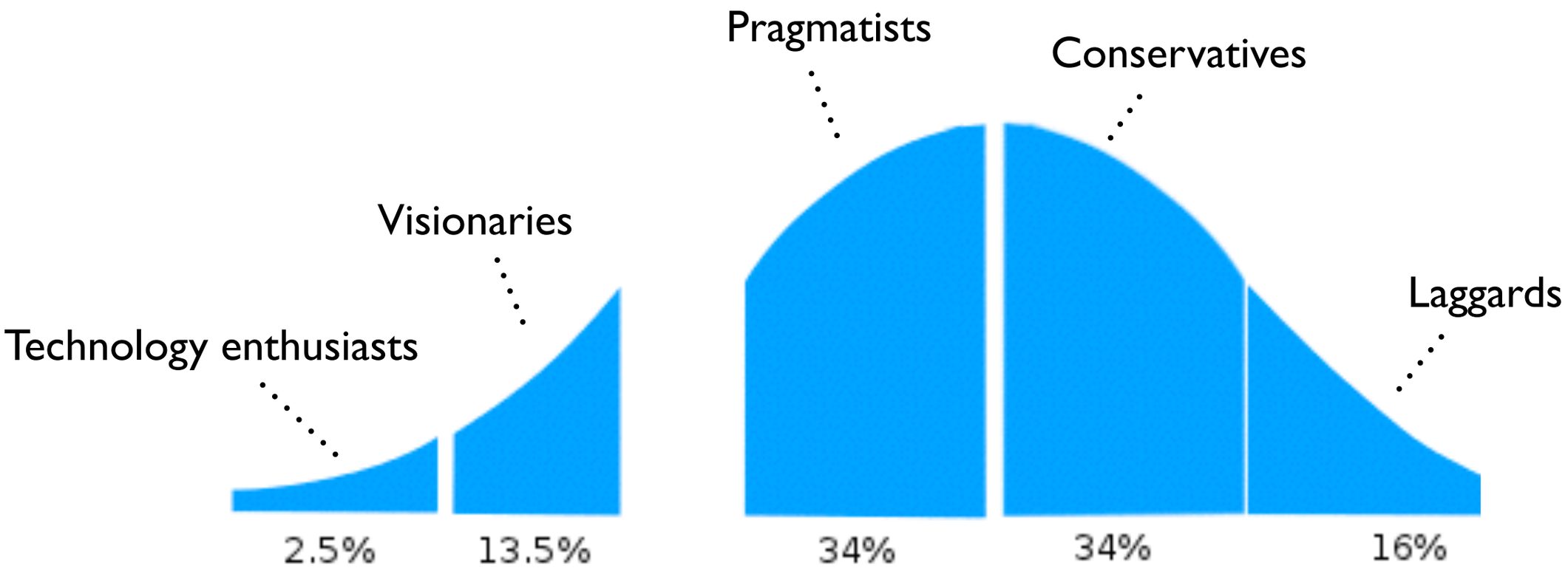
Cracks and a Chasm



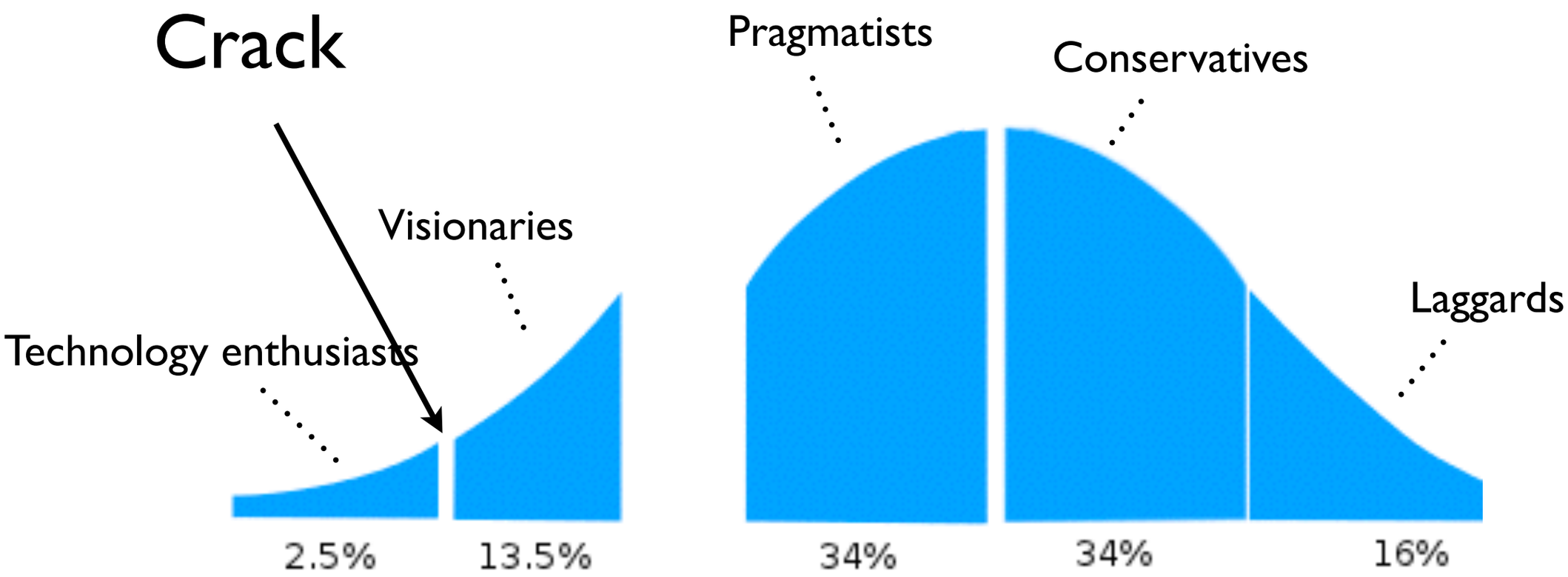
Cracks and a Chasm



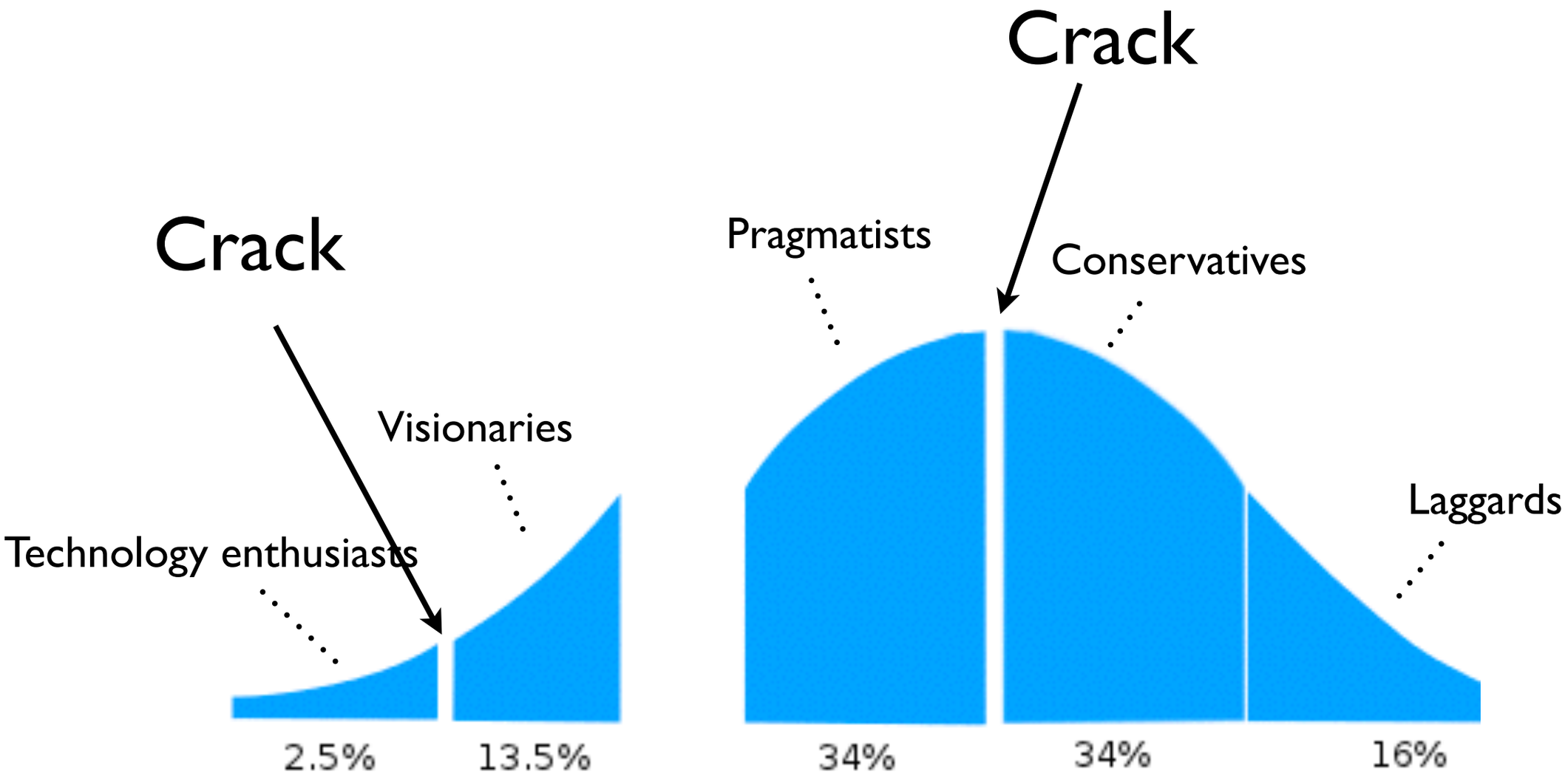
Cracks and a Chasm



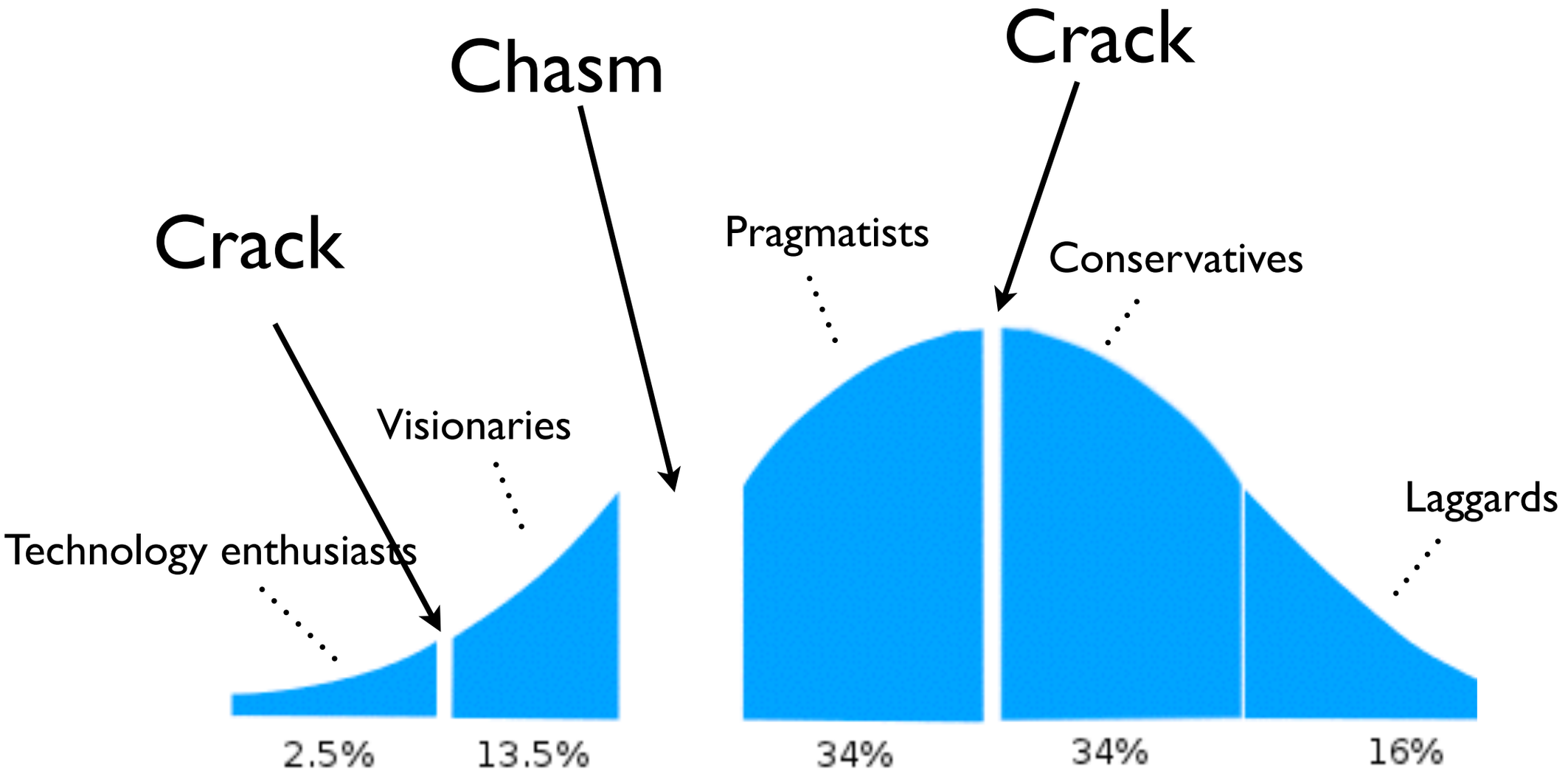
Cracks and a Chasm



Cracks and a Chasm



Cracks and a Chasm



The Chasm

The Chasm

From Visionaries to Pragmatists

The Chasm

From Visionaries to Pragmatists

Visionaries buy a *change agent* to get a radical discontinuity

The Chasm

From Visionaries to Pragmatists

Visionaries buy a *change agent* to get a radical discontinuity

Pragmatists want a *productivity improvement* for existing operations

The Chasm

From Visionaries to Pragmatists

Visionaries buy a *change agent* to get a radical discontinuity

Pragmatists want a *productivity improvement* for existing operations

Pragmatists want evolution, not revolution

The Chasm

From Visionaries to Pragmatists

Visionaries buy a *change agent* to get a radical discontinuity

Pragmatists want a *productivity improvement* for existing operations

Pragmatists want evolution, not revolution

Pragmatists wants references

Visionaries vs Pragmatists

Visionaries have four characteristics that alienate Pragmatists:

Visionaries vs Pragmatists

Visionaries have four characteristics that alienate Pragmatists:

- Lack of respect for their colleagues' experiences

Visionaries vs Pragmatists

Visionaries have four characteristics that alienate Pragmatists:

- Lack of respect for their colleagues' experiences
- Takes greater interest in technology than in their industry

Visionaries vs Pragmatists

Visionaries have four characteristics that alienate Pragmatists:

- Lack of respect for their colleagues' experiences
- Takes greater interest in technology than in their industry
- Fail to recognise the importance of existing product infrastructure

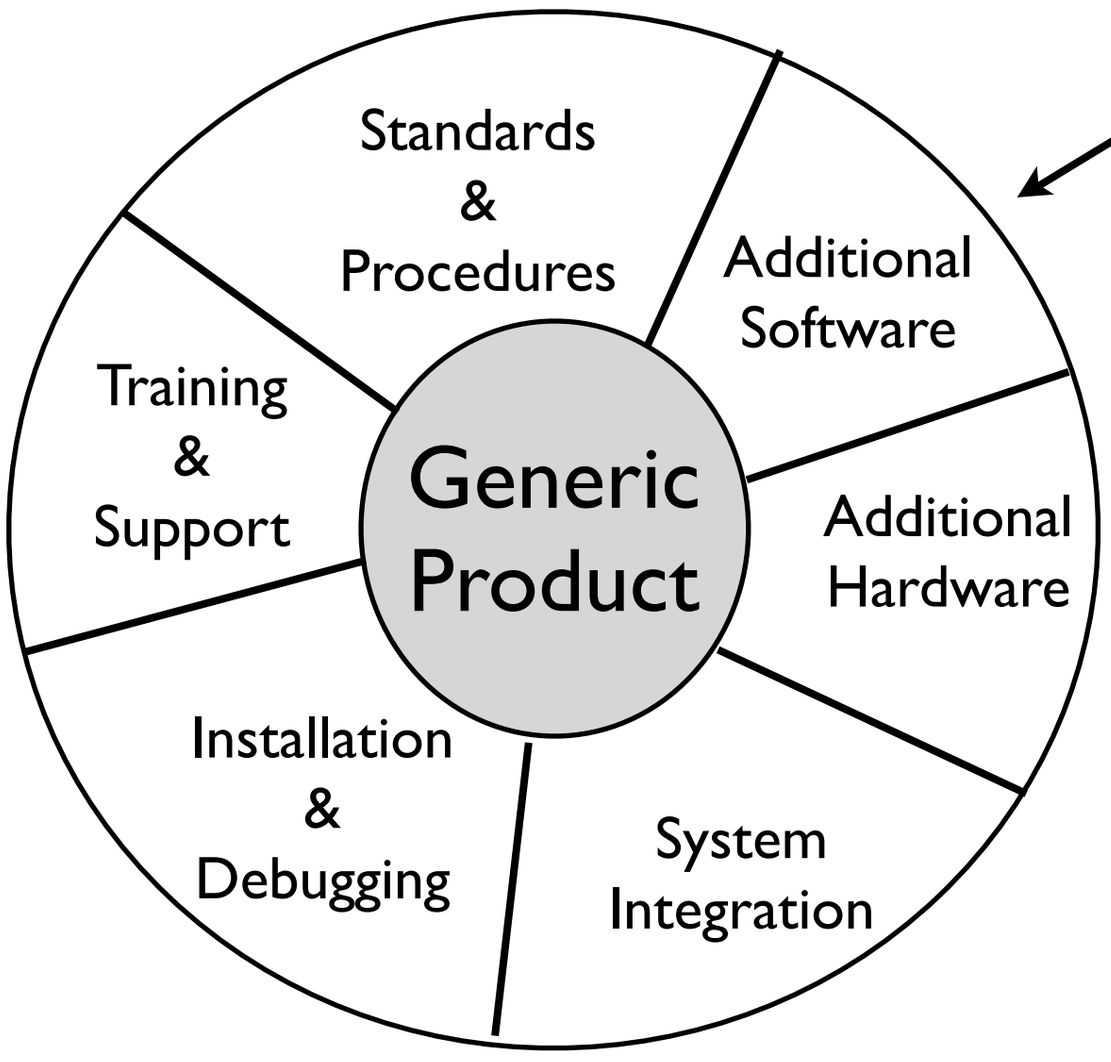
Visionaries vs Pragmatists

Visionaries have four characteristics that alienate Pragmatists:

- Lack of respect for their colleagues' experiences
- Takes greater interest in technology than in their industry
- Fail to recognise the importance of existing product infrastructure
- Overall disruptiveness

Whole Product Planning

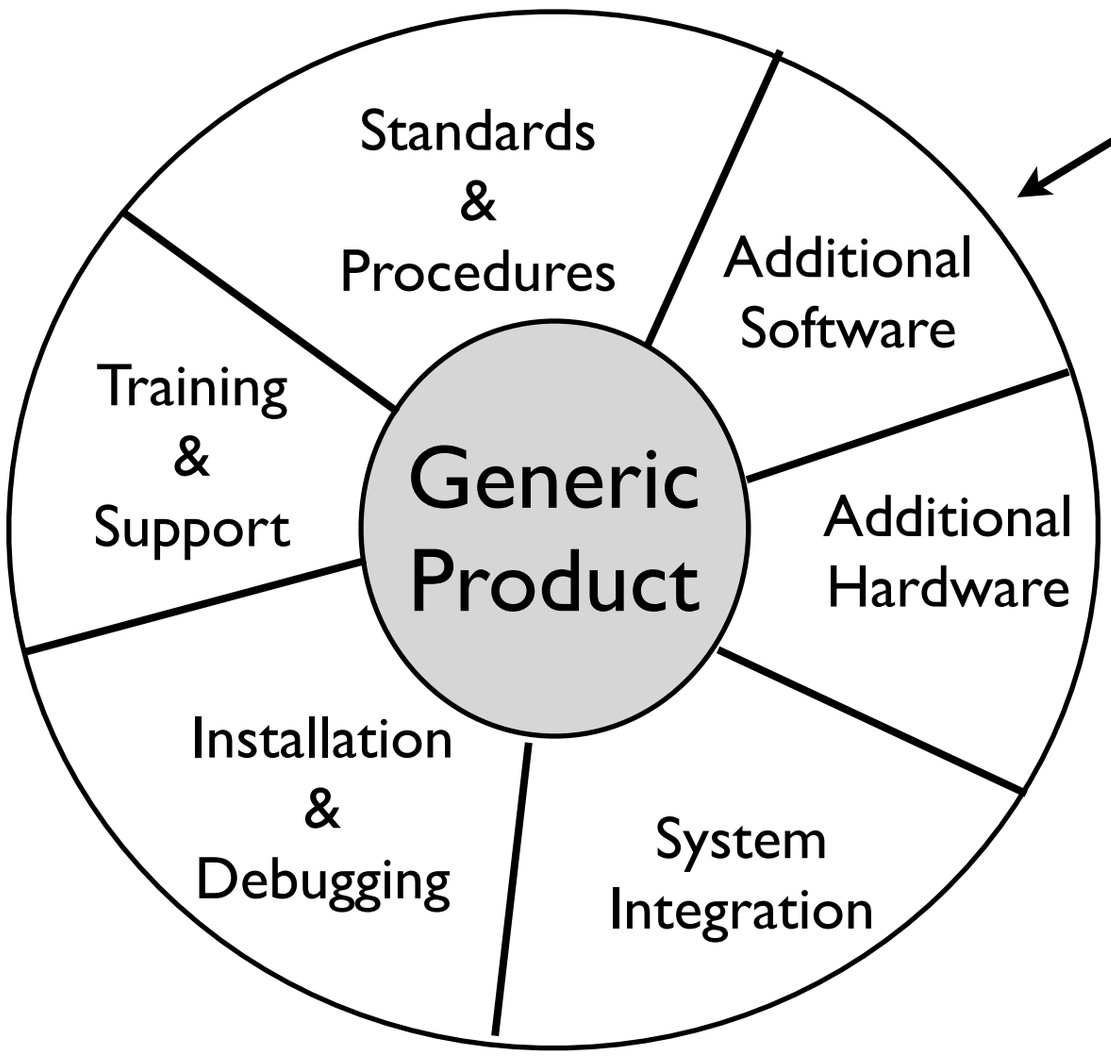
Simplified for chasm crossing



Anything else
to achieve your
compelling reason to buy

Whole Product Planning

Simplified for chasm crossing



Anything else
to achieve your
compelling reason to buy

= *marketing promise*
to win the sale

Erlang Whole Product 1/2

Generic product:

- Erlang compiler and runtime

Additional software:

- rich library shipped with each release
- many open source libraries

Training & support:

- ESL provides many courses
- ErlangCamp also provides training

Erlang Whole Product 2/2

System integration:

- mostly case-by-case
- few public success stories

Installation & Debugging:

Adequate functionalities for installing applications
Wombat: ESL tool for Operations and Maintenance
Good debugging tools, but not well publicised

The EuroVision Link

Or why Java is used more than Erlang…

source: http://i1.cdnds.net/14/19/618x411/uktv-eurovision-song-contest-2014-25.jpg

source: http://sawyerspeaks.com/wp-content/uploads/2011/04/larry_ellison_oracle_ceo.jpg

source: http://cache1.asset-cache.net/gc/489477445-conchita-wurst-of-austria-performs-on-stage-gettyimages.jpg?v=1&c=IWSAsset&k=2&d=X7WJLa88Cweo9HktRLaNXrTYXDJ3BsApHeprRdCRbmiAEiSzJ73mXeJ82T5OvCcz

The EuroVision Link

Or why Java is used more than Erlang…

Lots of wrapping

source: http://i1.cdnds.net/14/19/618x411/uktv-eurovision-song-contest-2014-25.jpg

source: http://sawyerspeaks.com/wp-content/uploads/2011/04/larry_ellison_oracle_ceo.jpg

source: http://cache1.asset-cache.net/gc/489477445-conchita-wurst-of-austria-performs-on-stage-gettyimages.jpg?v=1&c=IWSAsset&k=2&d=X7WJLa88Cweo9HktRLaNXrTYXDJ3BsApHeprRdCRbmiAEiSzJ73mXeJ82T5OvCcz

The EuroVision Link

Or why Java is used more than Erlang…

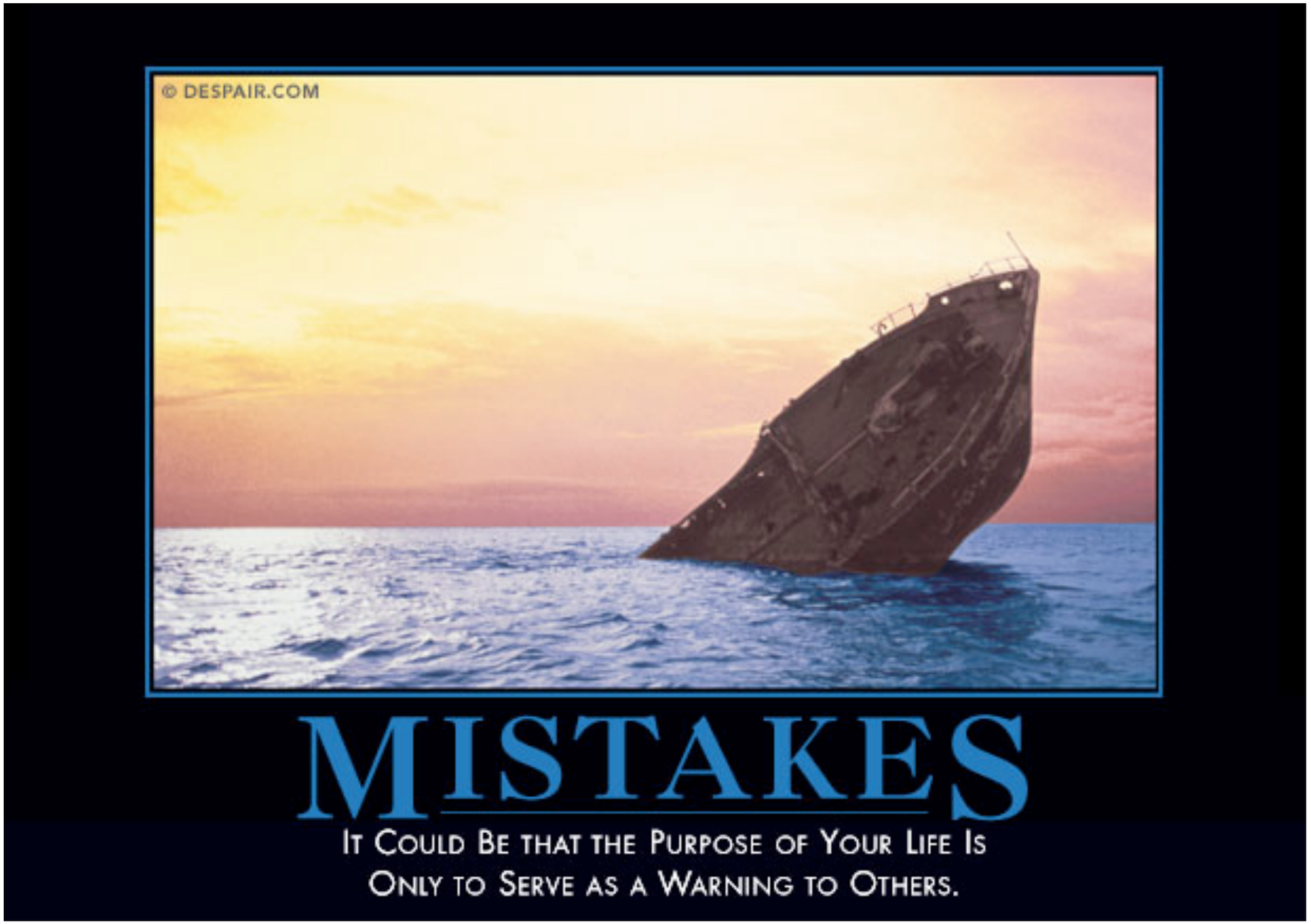
Lots of wrapping

The real McCoy

source: http://i1.cdnds.net/14/19/618x411/uktv-eurovision-song-contest-2014-25.jpg

source: http://sawyerspeaks.com/wp-content/uploads/2011/04/larry_ellison_oracle_ceo.jpg

source: http://cache1.asset-cache.net/gc/489477445-conchita-wurst-of-austria-performs-on-stage-gettyimages.jpg?v=1&c=IWSAsset&k=2&d=X7WJLa88Cweo9HktRLaNXrTYXDJ3BsApHeprRdCRbmiAEiSzJ73mXeJ82T5OvCcz



Source: http://www.despair.com/mistakes.html

Learning Erlang

Learning Erlang

ESL training courses

Learning Erlang

ESL training courses

Learn You Some Erlang

http://learnyousomeerlang.com/

Learning Erlang

ESL training courses

Learn You Some Erlang

http://learnyousomeerlang.com/

Use the erlang-questions mailing list

Learning Erlang

ESL training courses

Learn You Some Erlang

http://learnyousomeerlang.com/

Use the erlang-questions mailing list

Do it hands-on

Learning Erlang

ESL training courses

Learn You Some Erlang

http://learnyousomeerlang.com/

Use the erlang-questions mailing list

Do it hands-on

Give it time to sink in!!!

Elixir

Elixir

Built on top of the Erlang VM

Elixir

Built on top of the Erlang VM

More Ruby-like syntax

Elixir

Built on top of the Erlang VM

More Ruby-like syntax

Hygienic macros - easy to do DSLs

Elixir

Built on top of the Erlang VM

More Ruby-like syntax

Hygienic macros - easy to do DSLs

But… you still have to learn the Erlang programming model

Key building blocks

Key building blocks

Share nothing processes

Key building blocks

Share nothing processes

Message passing

Key building blocks

Share nothing processes

Message passing

Fail fast approach

Key building blocks

Share nothing processes

Message passing

Fail fast approach

Link/monitor concept

Key building blocks

Share nothing processes

Message passing

Fail fast approach

Link/monitor concept

You can deal with failures in a sensible manner because you have a language for them.

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 supervision patterns to deliver business value

Cruising with Erlang

Understand the failure model

Embrace failure!

Use supervision patterns to deliver business value

Stay in charge!

Cruising with Erlang

Understand the failure model

Embrace failure!

Use supervision patterns to deliver business value

Stay in charge!