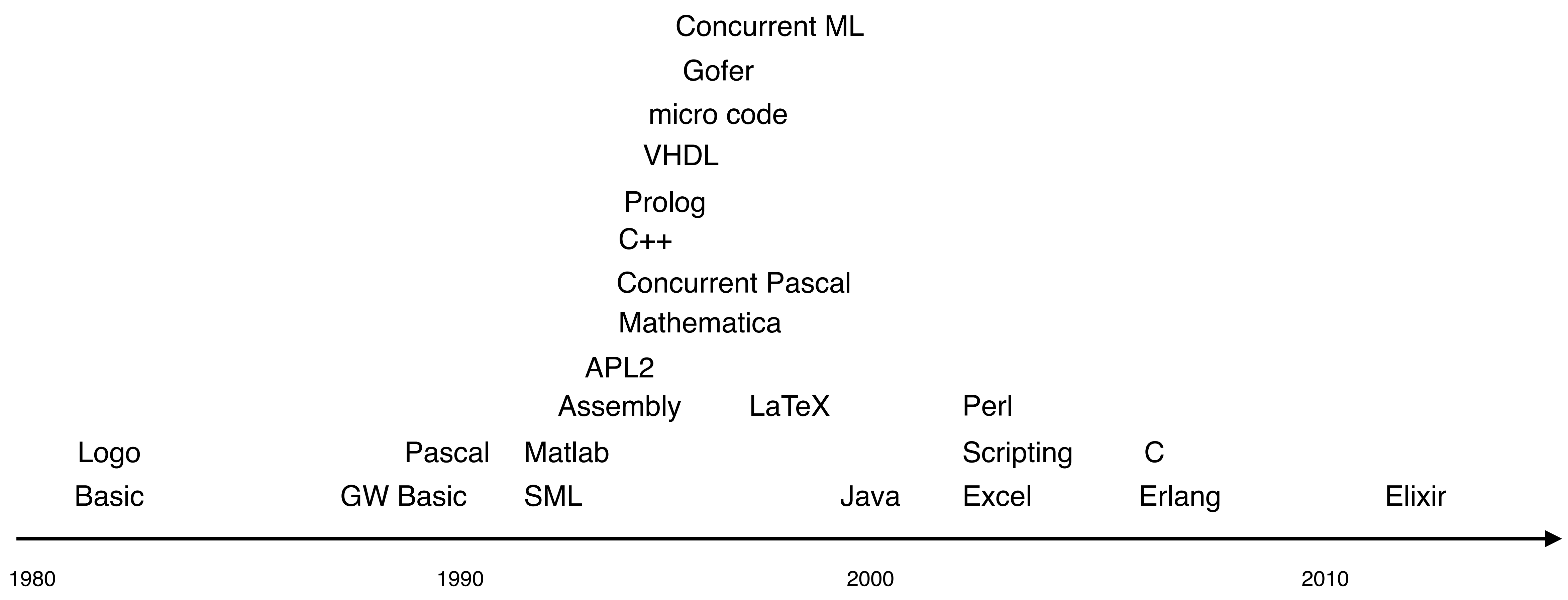
& Idioms

Erlang Patterns Matching Business Needs

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

Erlang SOLUTIONS

Background


Concurrent ML
Gofer
micro code
VHDL
Prolog
C++
Concurrent Pascal
Mathematica
APL2
Assembly
LaTeX
Perl
Logo
Pascal
Matlab
Scripting
C
Basic
GW Basic
SML
Java
Excel
Erlang
Elixir
1980
1990
2000
2010

Why this talk?

Show the business value of Erlang

Introduce Erlang Patterns

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

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

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

wanted

short time-to-market

on-the-fly upgrades

quality and reliability

and more...

wanted

productivity

no down-time

something that always works

wanted

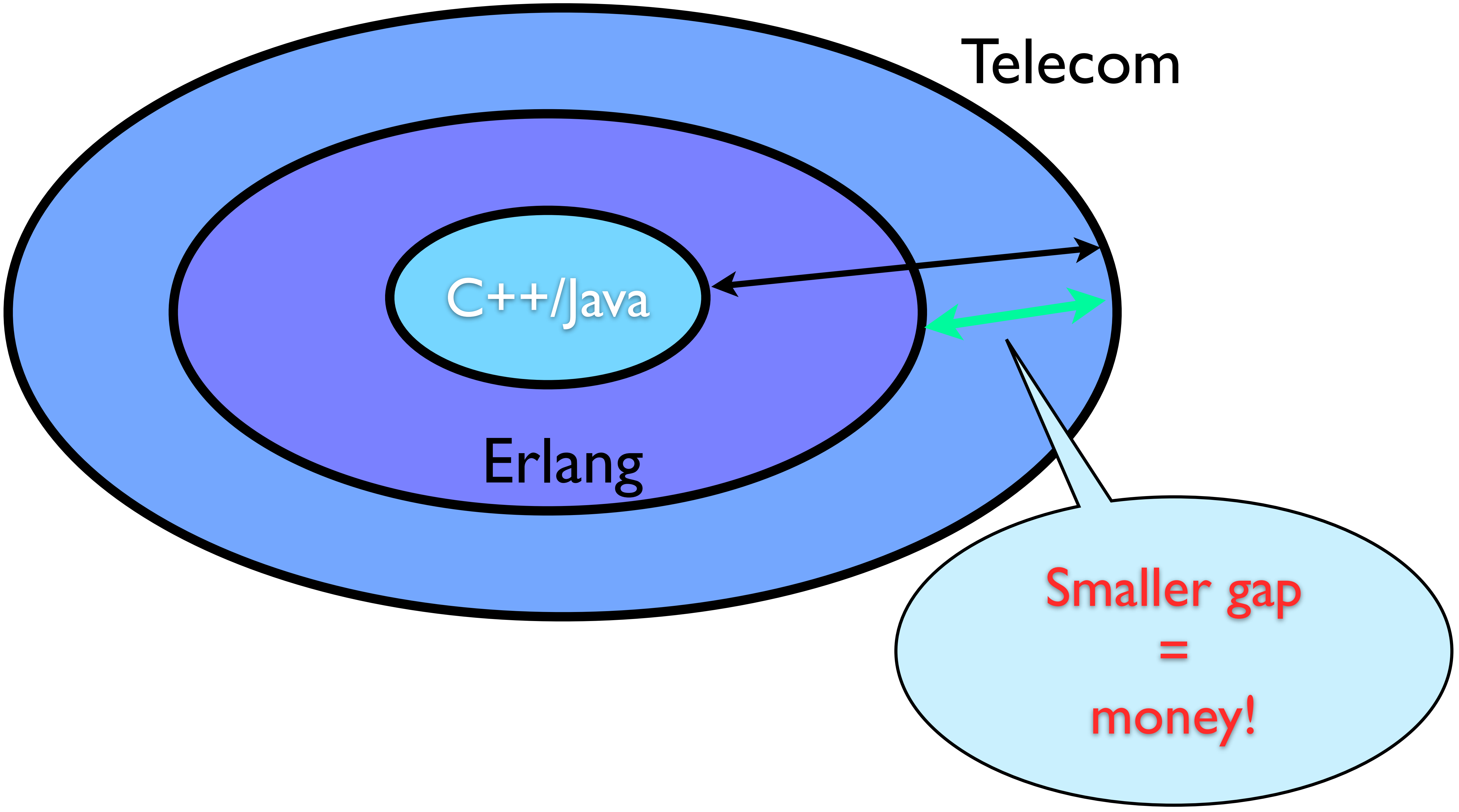
money

money

money

it’s a rich man’s world!

General vs Domain Specific



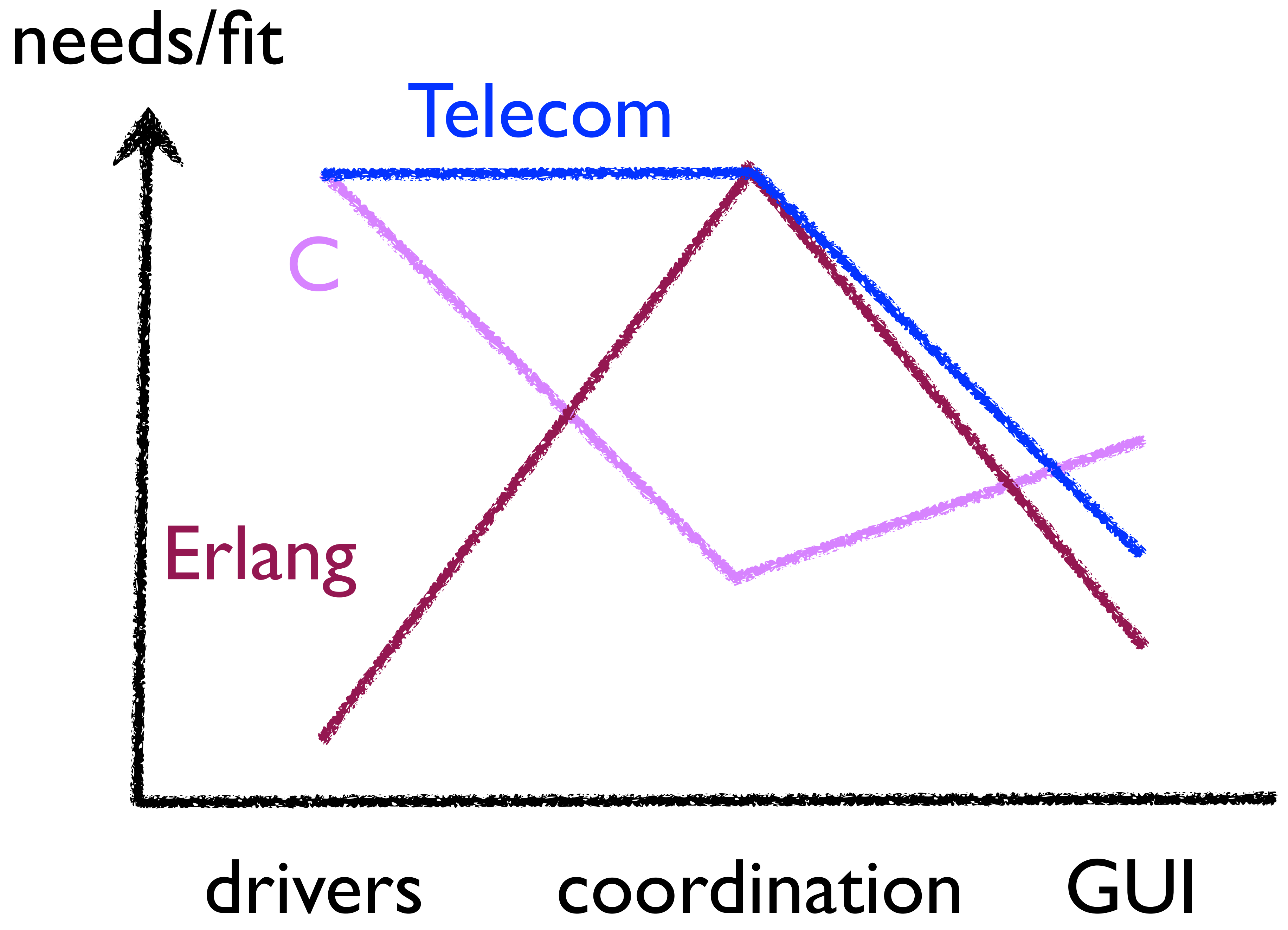
The Sweet Spot

GUI

Middleware
Coordination
Control

Drivers

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

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)

Good Erlang Domains

Low latency over throughput

Stateful (in contrast to being stateless)

Massively concurrent

Distributed

Fault tolerant

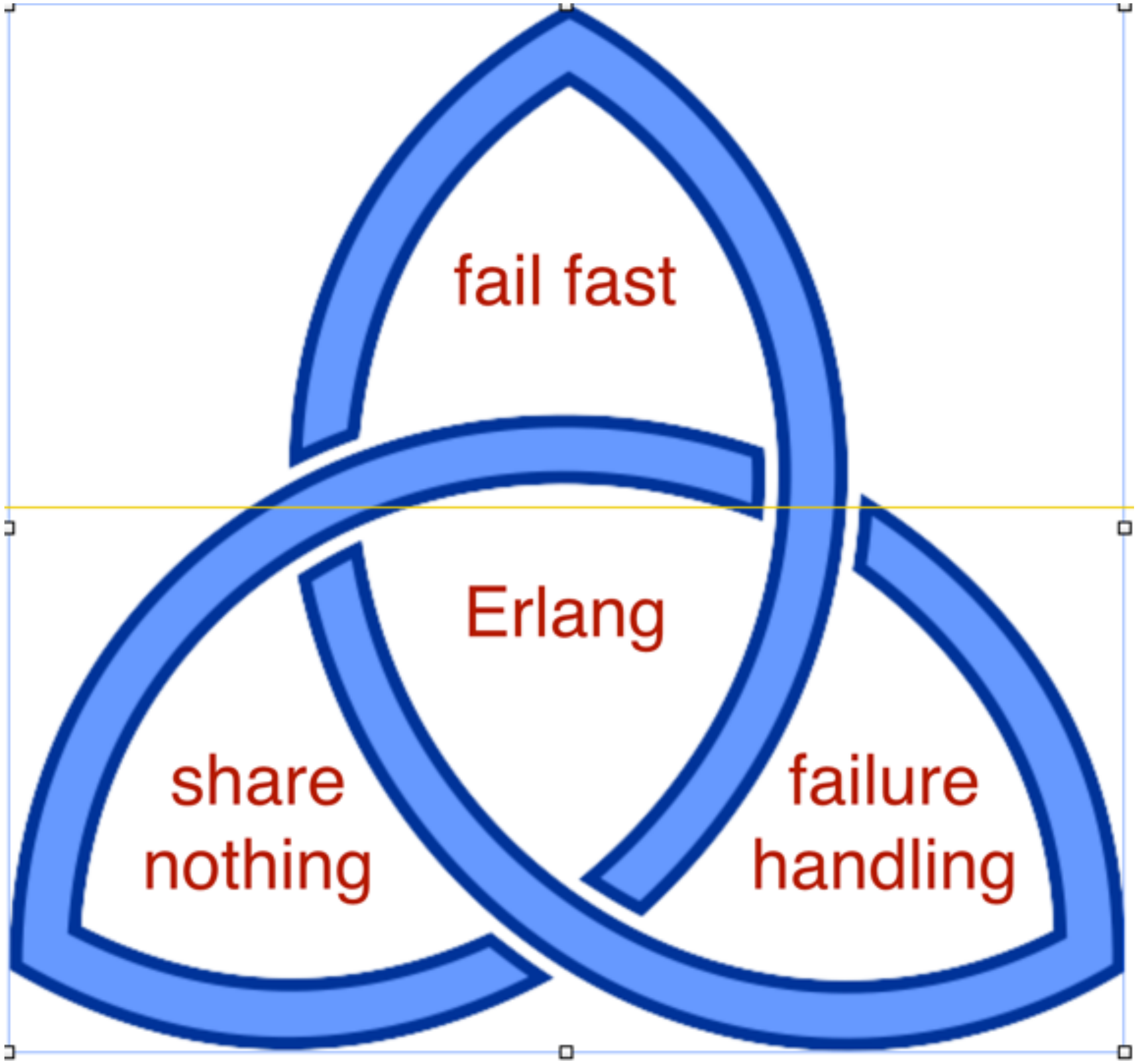
Uses OTP

Non-stop operation

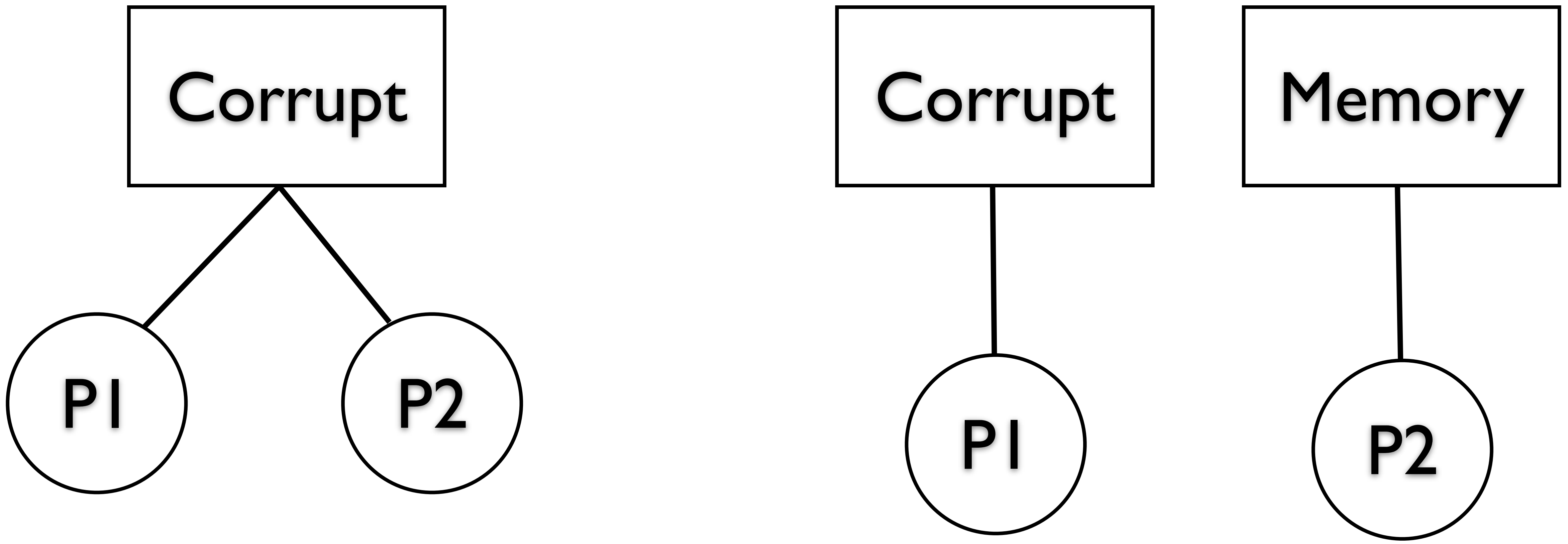
Under load, Erlang programs usually performs as well as programs in other languages, often way better.

Jesper Louis Andersen

The Golden Trinity Of Erlang



To Share Or Not To Share



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

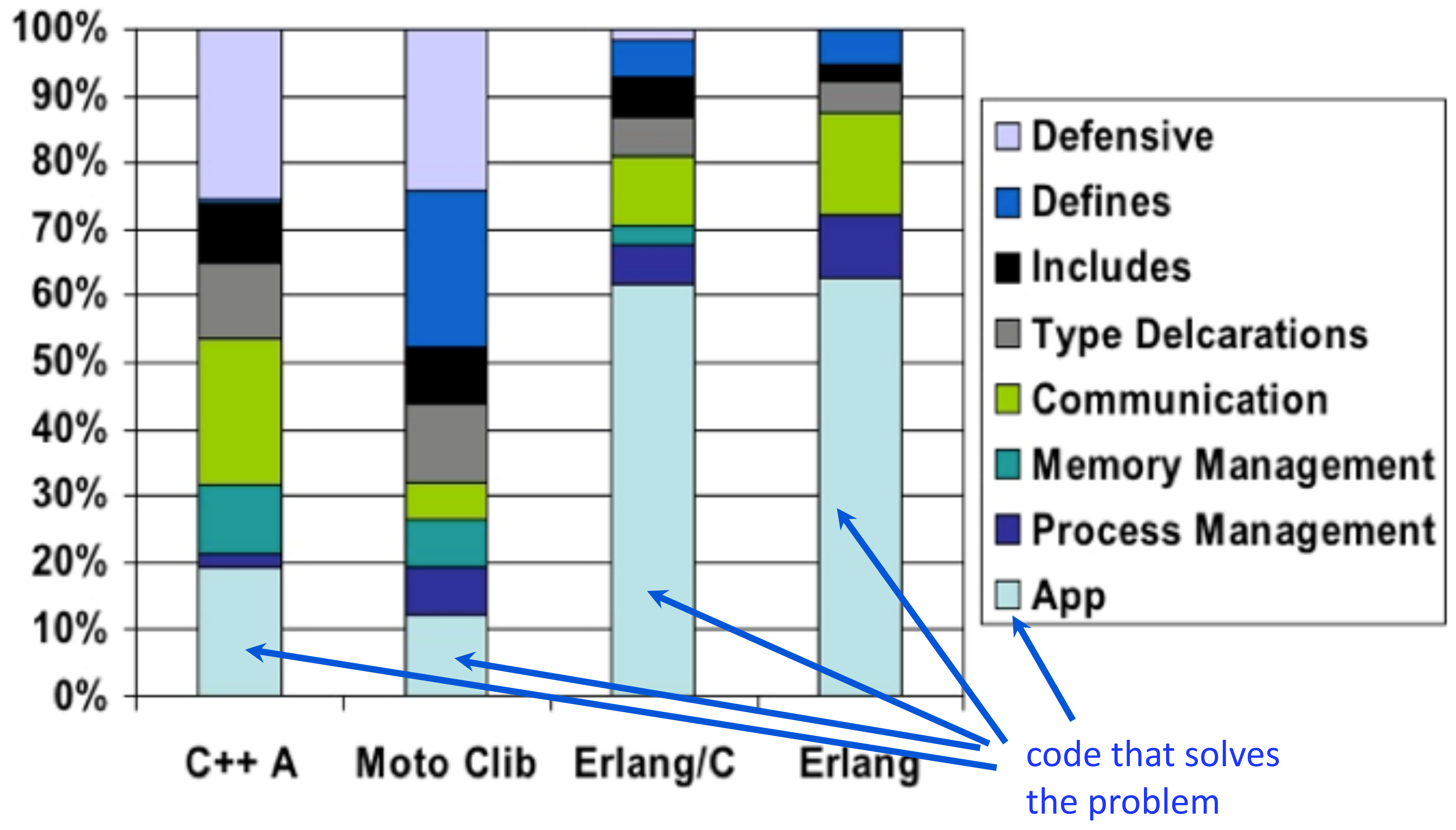
Let It Fail

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

Erlang encourages agressive/offensive programming

Benefits of let-it-fail

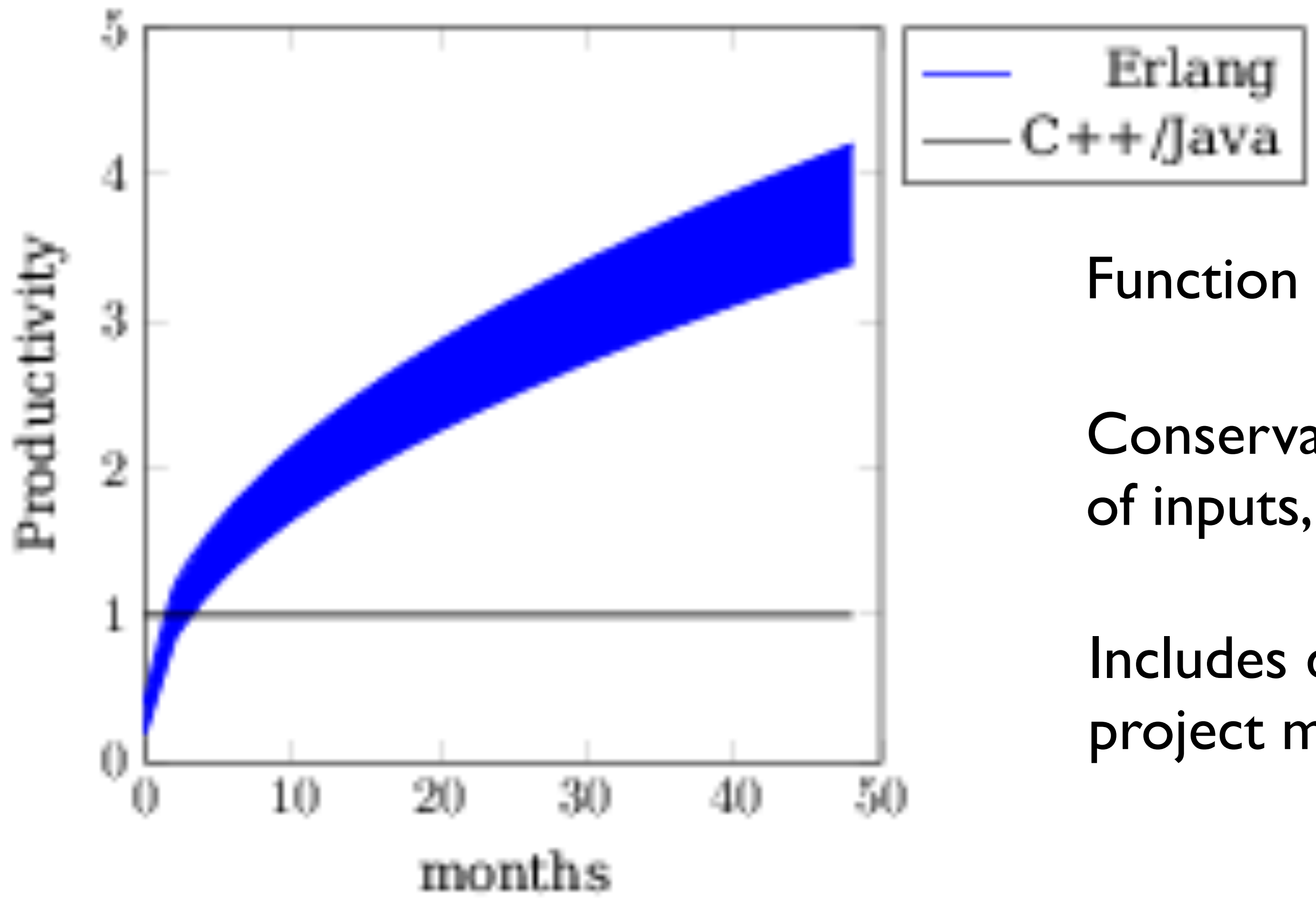
Data Mobility component breakdown



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

Erlang @ 3x

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

Visual Erlang

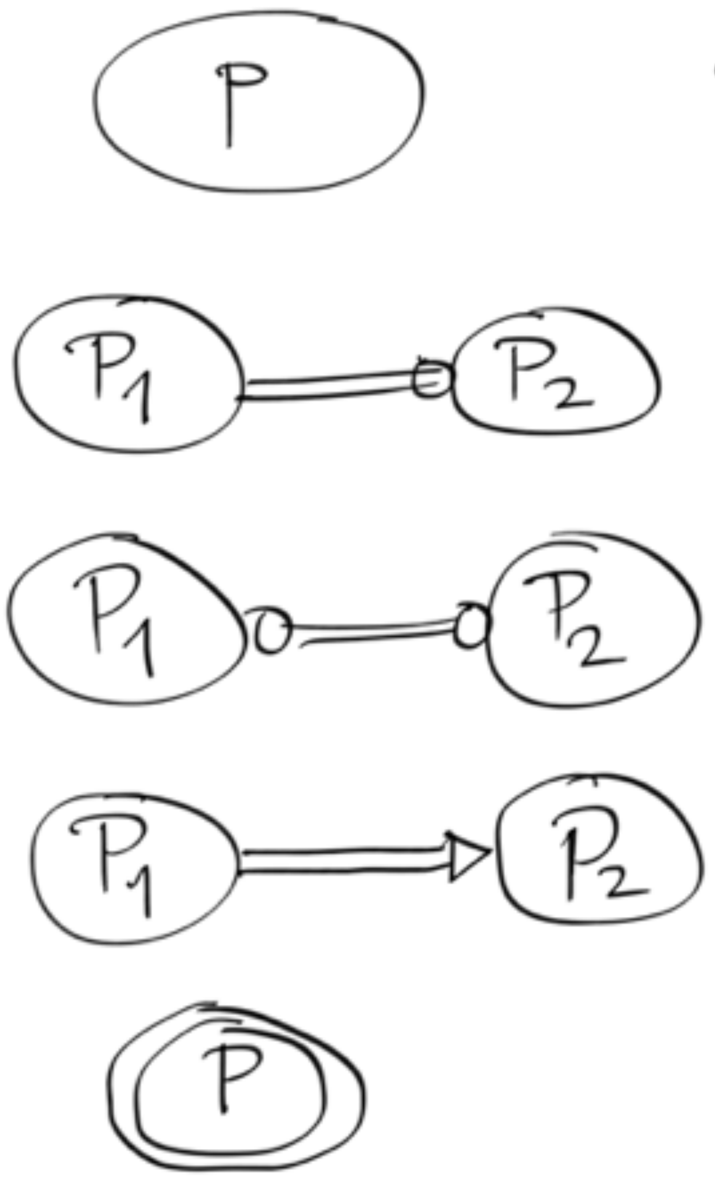
Visual Erlang Objectives

Detailed enough to capture important aspects

Not suited for 100% explanation of Erlang

Standardise on how we show Erlang architecture

Processes in Visual Erlang



Process P

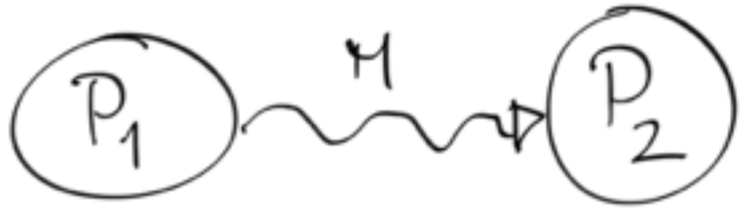
P1 monitors P2

P1 & P2 are linked

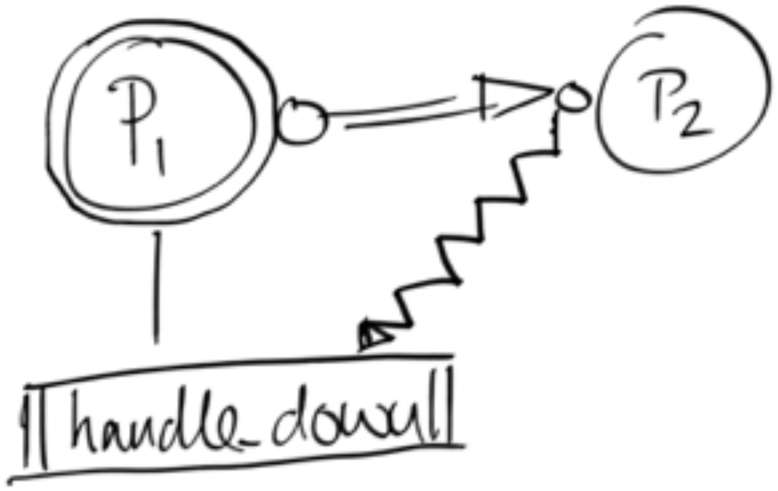
P1 spawns P2.

Process P traps exits.

Messages and Functionality

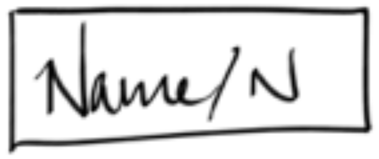


P_1 sends M to P_2

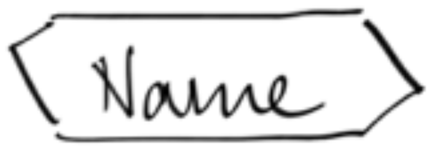


P_1 has functionality
handle_down to
handle DOWN
msgs from P_2

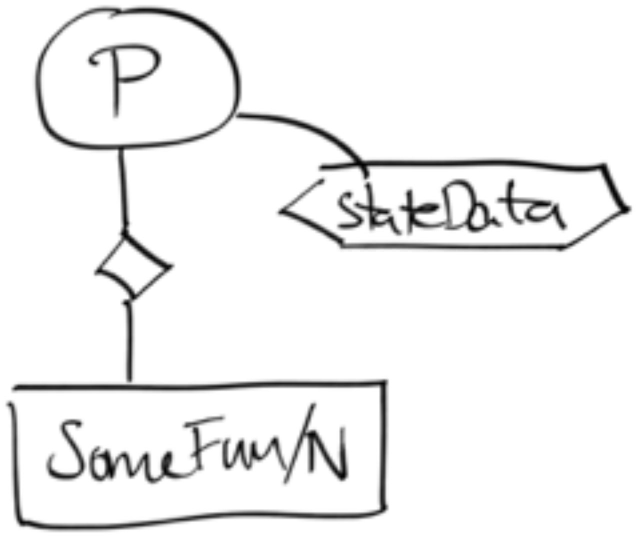
Functions & State Data



Function Name
w/ arity N



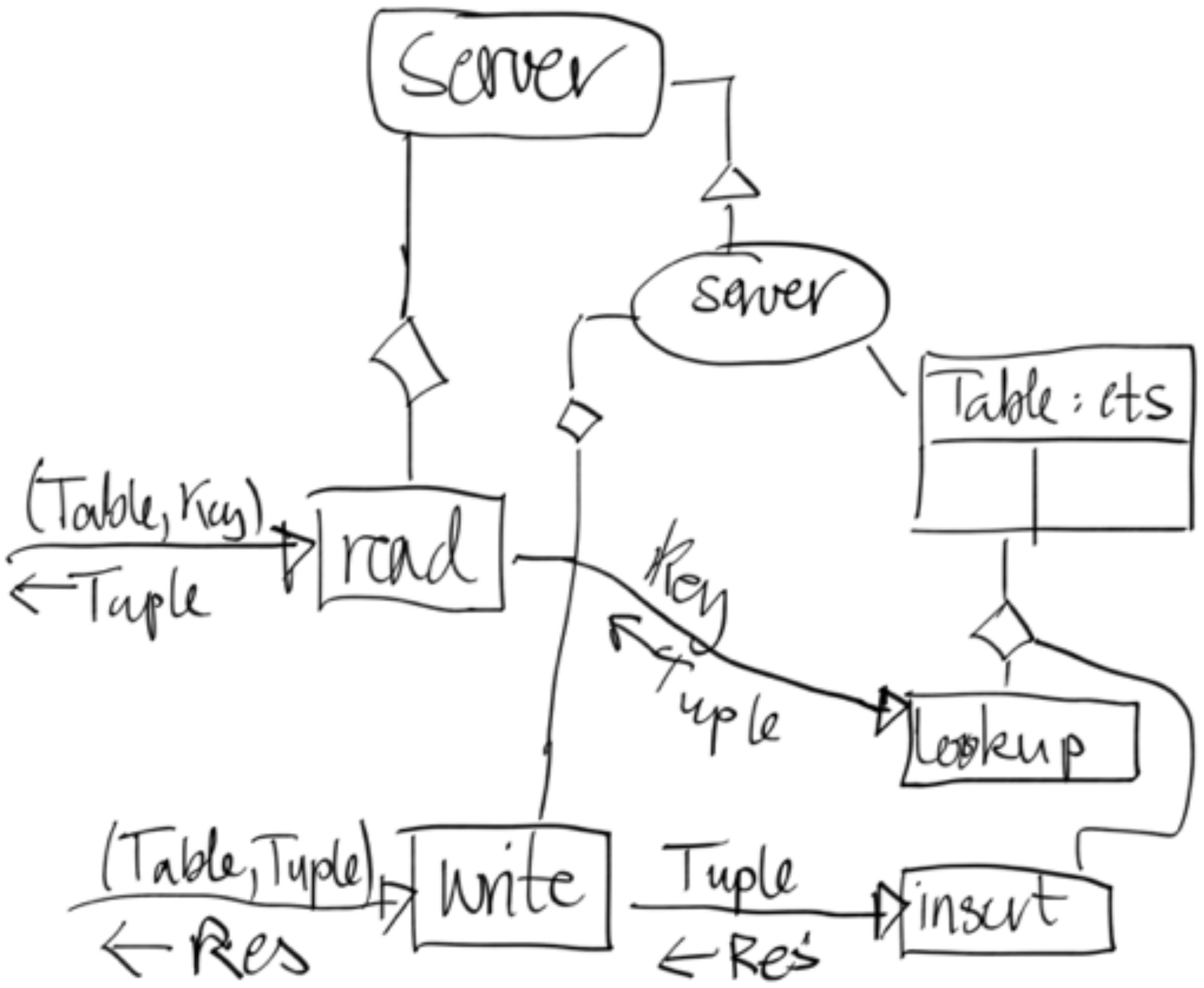
State data for
a function



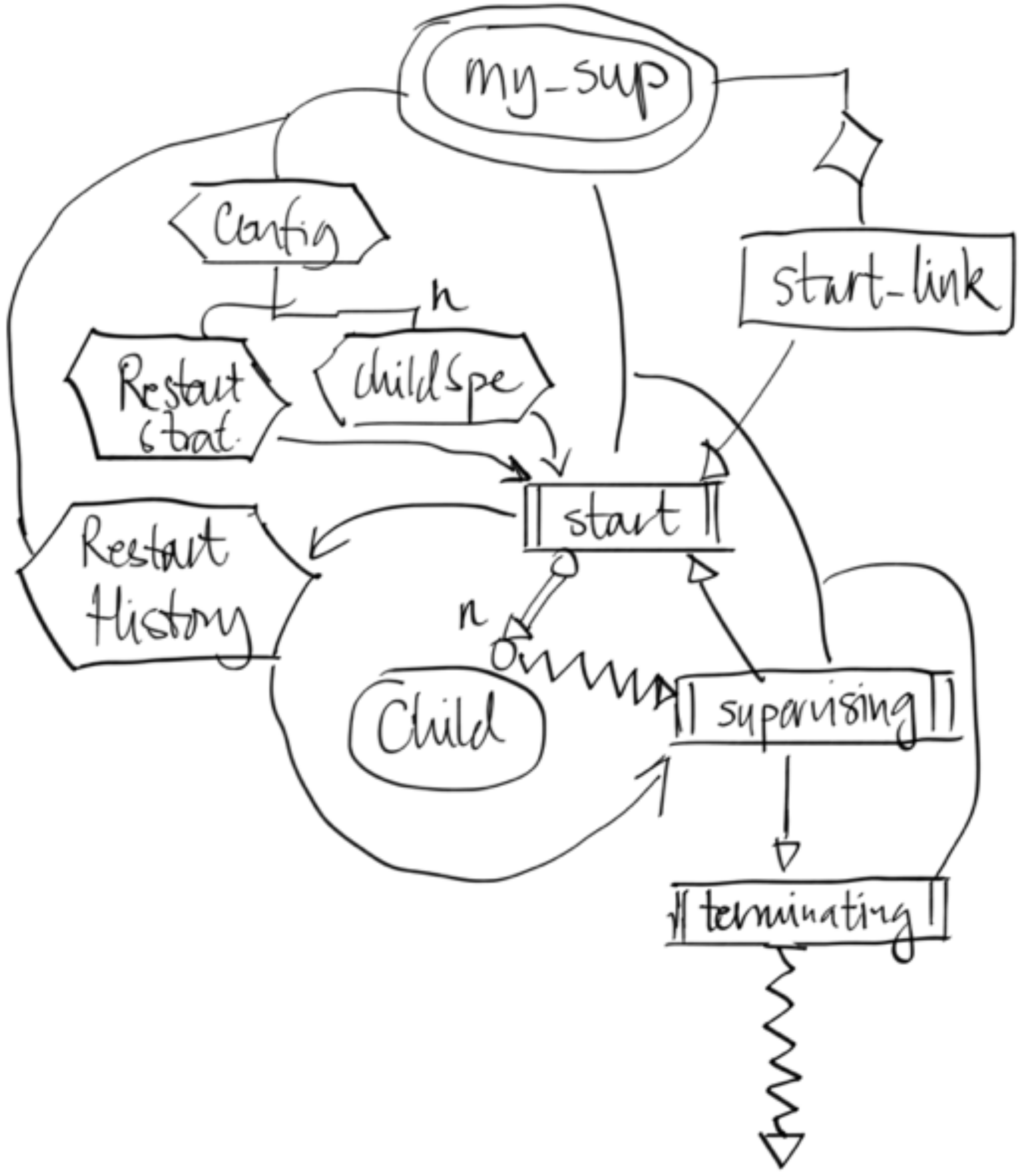
Process P has
public API
SomeFun/N &
state data
StateData.

Erlang Patterns

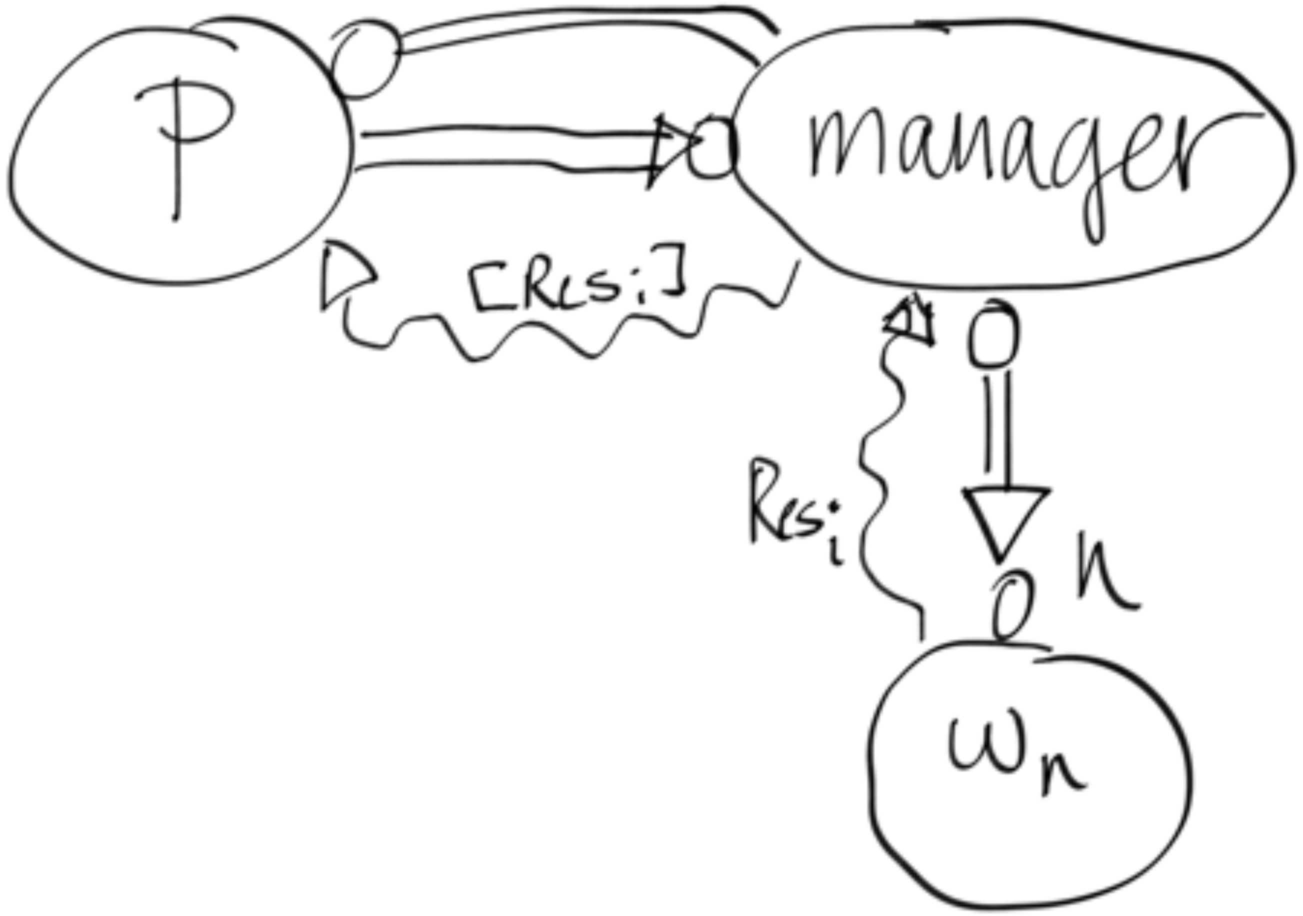
Tuple Space Storage Pattern



Supervisors



Simple Manager/Worker Pattern



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

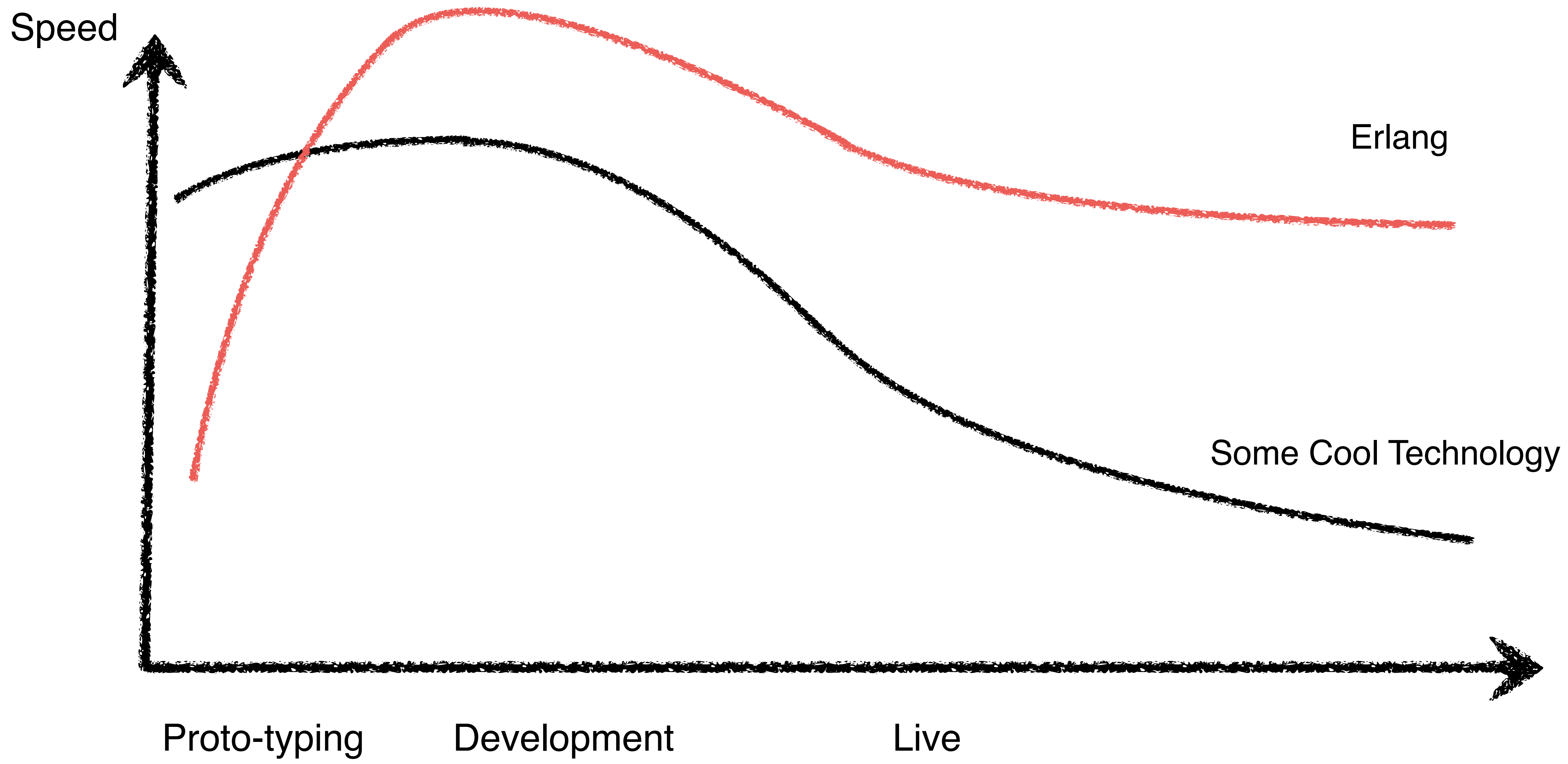
Visual Erlang Patterns

Adds vocabulary about architecture

- Share insights

Consider failures while designing

When do I get my ROI?



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.

Elixir

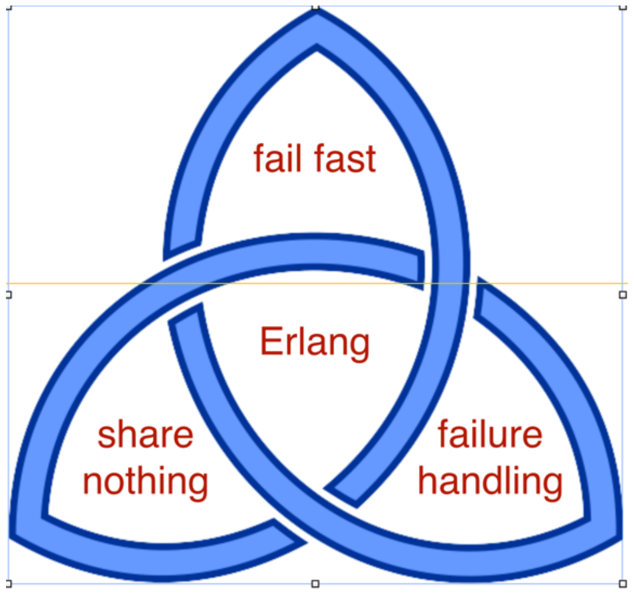
Built on top of the Erlang VM

More Ruby-like syntax

Hygienic macros - easy to do DSLs

But… you still have to learn the Erlang programming model

Cruising with Erlang



Understand the failure model

Embrace failure!

Use patterns to deliver business value

Stay in charge!