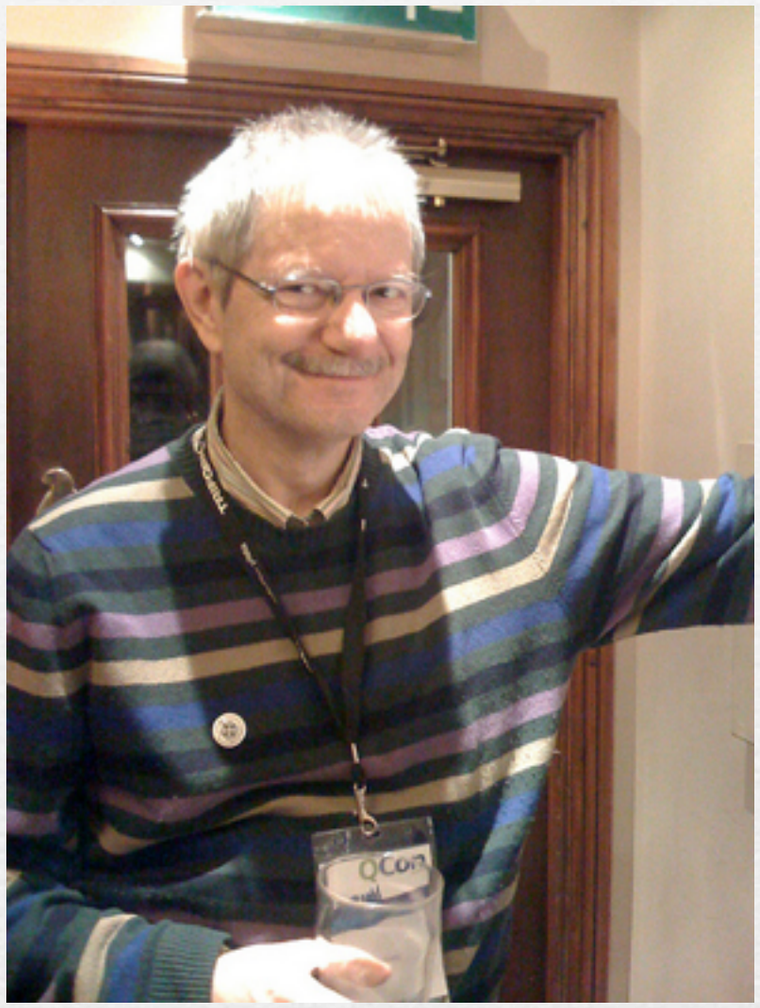
More than a century of programming

Joe Armstrong
Robert Virding
Mike Williams


QCon

History (early 1980's)

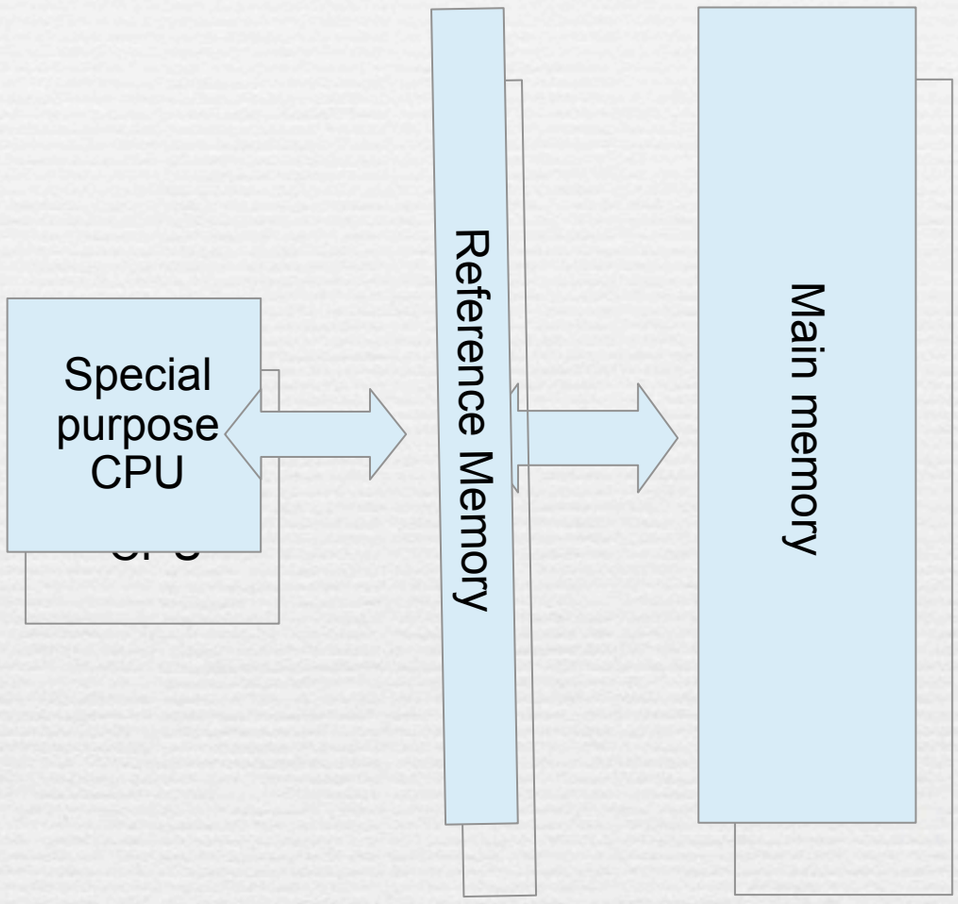
- Ericsson's “best seller” was AXE telephone exchanges (switches).
- Required large effort to develop and maintain software.
- The “job” was to make programming these types of application easier, but keeping the same characteristics

AXE Exchange, 1985
Guangzhou,China

AXE SW Characteristics

mike

- Massive concurrency (thousands of transactions)
- Array bounds and pointer checking in hardware
  - No wild pointers
  - Size changes of statically allocated arrays
  - Re-arrange memory
- Change code at runtime
- Modular
- Error handling and transactions



Special purpose synchronously duplicated AXE hardware

Do the same as AXE, but use

- Conventional processors, easy portability to new processors
- Conventional operating system (type UNIX)
- Distributed multi-processor system enabling scalability (more processing power == more processors)

And

- Make software development effort significantly easier.

How hardware changes how we think

- Large memory/small memory
- Always on line
- Fast CPU
- Parallel hardware
- Mobility

If the hardware doesn't change the software won't change

Hardware didn't change much in 1986-2004 so the software didn't change much

apart from clock speed and memory capacity still Von-Neumann non-distributed non-connected

1980 - 1984

- The state of computer science
- 3 MHz clock
- 80 MB disks
- 4MB memory ...
- (language reflect the hardware of the time when they were developed)

Problem Domain

joe

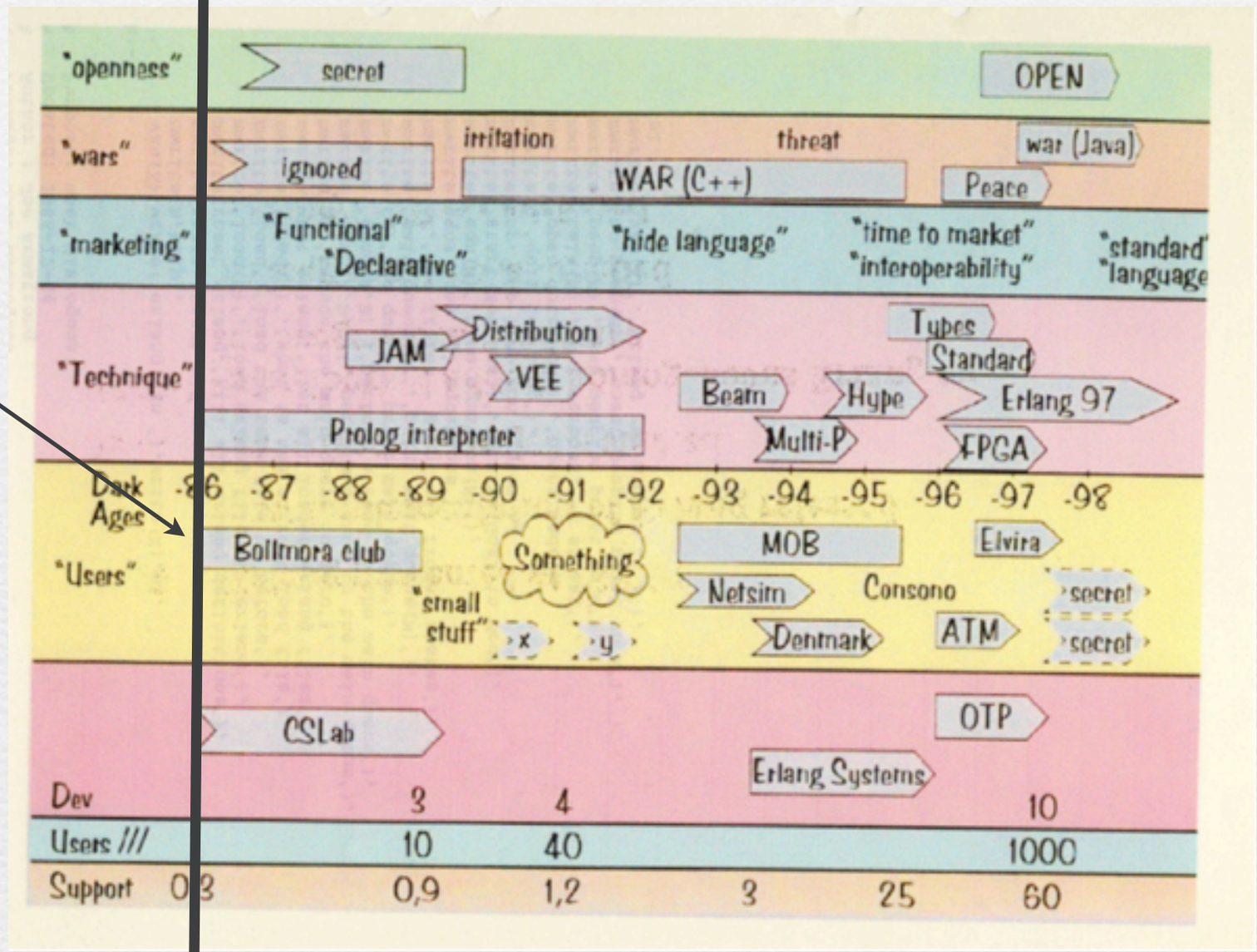
1. Actions must be performed at a certain point in time or within a certain time.
2. System may be distributed over several computers.
3. The system is used to control hardware.
4. The software system is very large.
5. The system exhibits complex functionality such as feature interaction.
6. The systems should be in continuous operation over many years.
7. Software maintenance (reconfiguration etc.) should be performed without stopping the system.
8. There are stringent quality, and reliability requirements.
9. Fault tolerance both to hardware failures, and software errors, must be provided.
10. The system *must* be able to handle very large numbers of concurrent activities.

Bjarne Däcker. November 2000 – Licentiate thesis

Reality check

joe

KERSTIN ÖDLING
Ericsson Business Networks AB



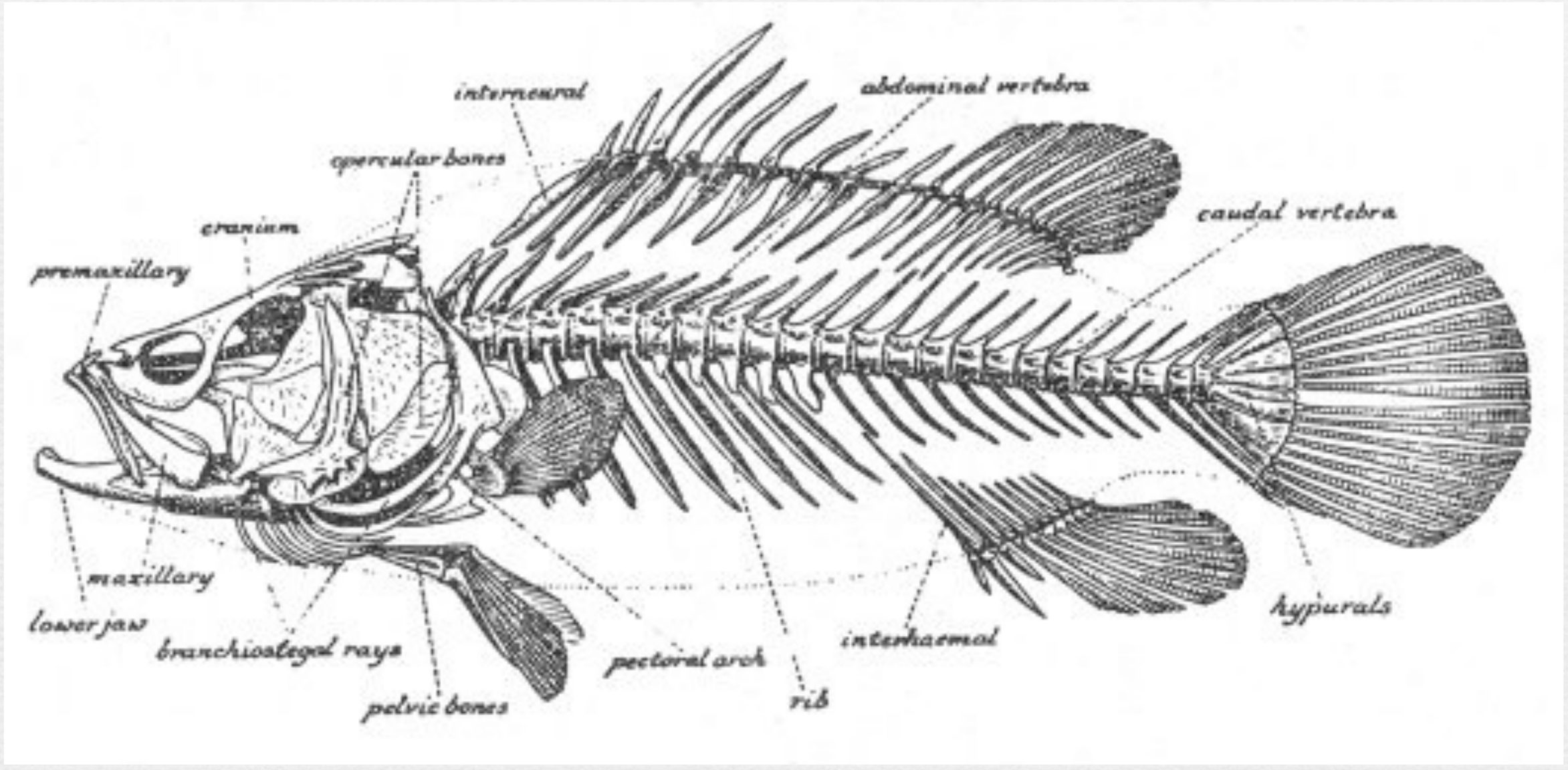
MD110

joe

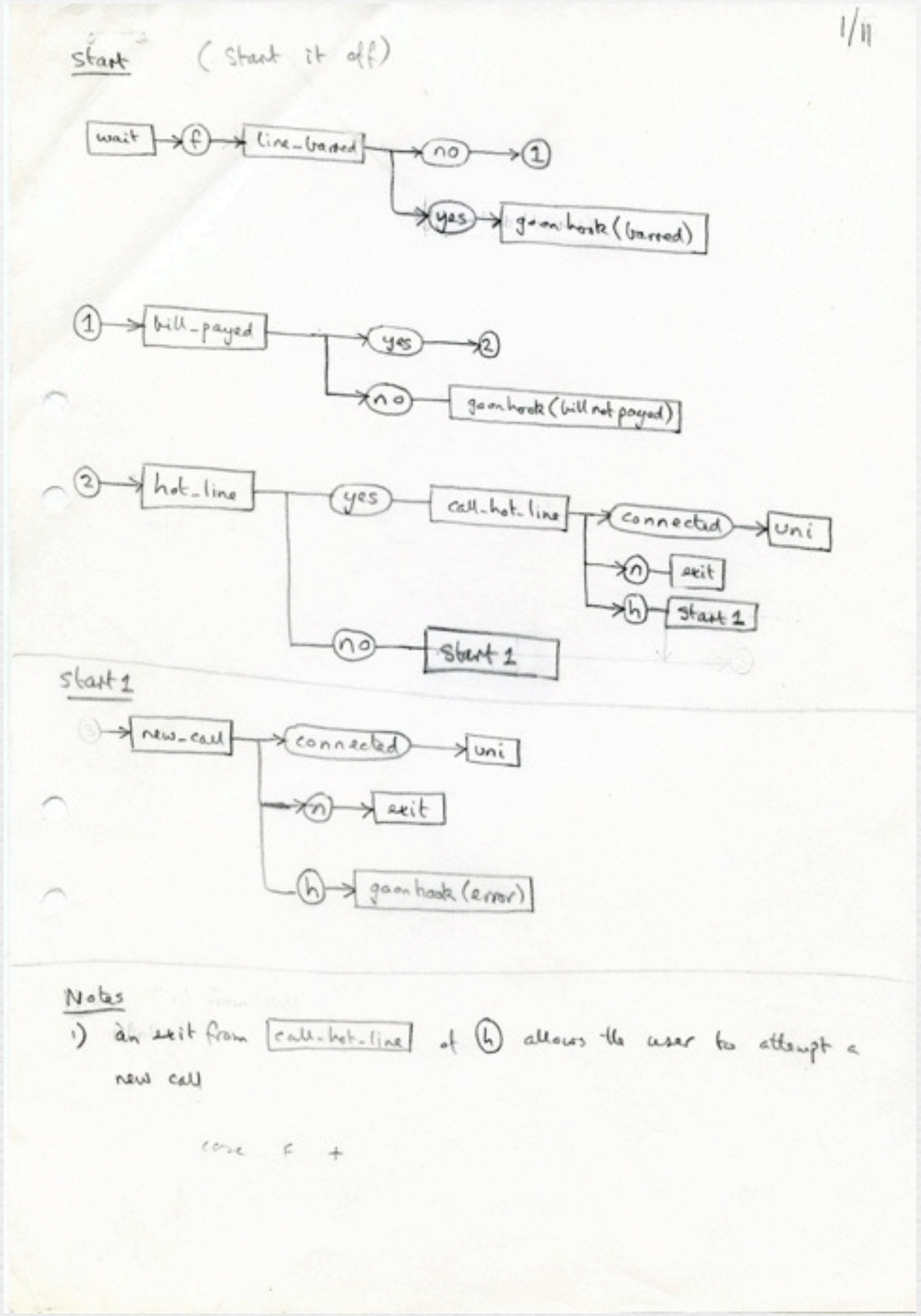
"It's the hardware stupid"

"Hardware is funny" - RV

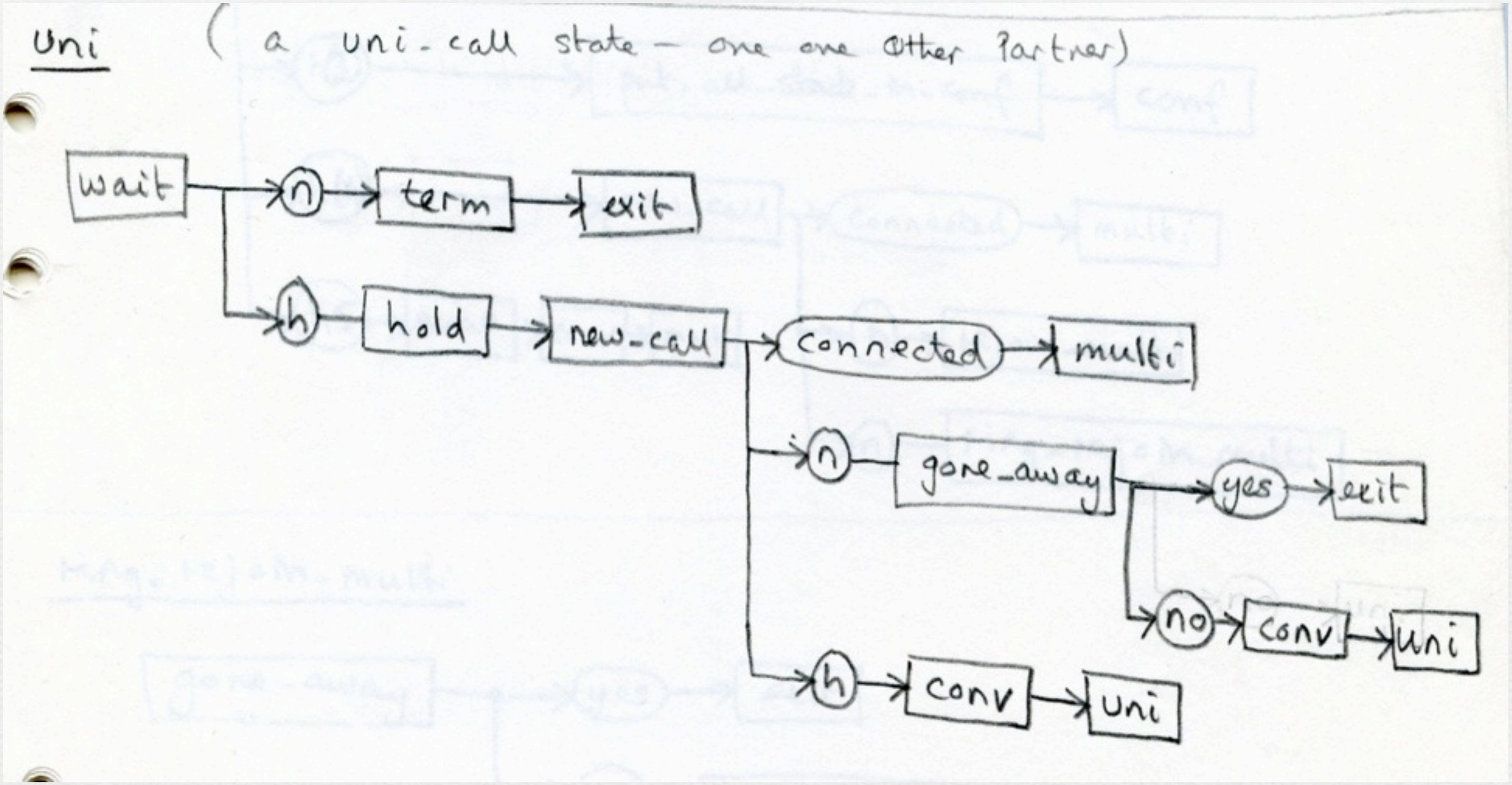
Erlang version 0



KERSTIN ÖDLING
Ericsson Business Networks AB



Fish bone diagrams



```
function uni returns none.

1 # uni --->
        case(wait,[
            n => [term,exit],
            h => [hold,
                  case(new_call,[
                    connected => multi,
                    n => case(gone_away,[
                      yes => exit,
                      no => [conv,uni]
                      ]),
                    h => [conv,uni]
                    ])
                ]
        ]).
```

Today

joe

```
uni() ->
 receive
    n ->
       term(),exit();
    h ->
       case new_call() of
          connected ->
             multi();
          n ->
             case gone_away() of
                yes -> exit();
                n   -> conv(), uni()
             end;
          h ->
             conv(), uni()
    end
end.
```

1988 - Robert Virding joins the team

4 days for a rewrite

Not so fast

88/12/16 12:44:20 — erlang.pl — 1

```
/*
 * $HOME/erlang.pro
 *
 *                Copyright (c) 1988 Ericsson Telecom
 *
 * Author: Joe Armstrong
 * Creation Date: 1988-03-24
 * Purpose:
 *      main reduction engine
 *
 * Revision History:
 *      88-03-24        Started work on multi processor version
 *                      of erlang
 *      88-03-28        First version completed (Without timeouts)
 *      88-03-29        Correct small errors
 *      88-03-29        Changed 'receive' to make it return the pair
 *                      msg(From,Mess)
 *      88-03-29        Generate error message when out of goals
 *                      i.e. program doesn't end with terminate
 *      88-03-29        added trace(on), trace(off) facilities
 *      88-03-29        Removed Var := {....} , this can be achieved
 *                      with {..}
 *      88-05-27        Changed name of file to erlang.pro
 *                      First major revision started - main changes
 *                      Complete change from process to channel
 *                      based communication
 *                      here we (virtually) throw away all the
 *                      old stuff and make a bloody great data base
 *      88-05-31        the above statements were incorrect much better
 *                      to go back to the PROPER way of doing things
 *                      long live difference lists
 *      88-06-02        Reds on run([et5]) = 245
 *                      changing the representation to separate the
 *                      environment and the process - should improve things
 *                      It did .... reds = 283 - and the program is nicer!
 *      88-06-08        All pipe stuff working (pipes.pro)
 *                      added code so that undefined functions can return
 *                      values
```

1988

- Let's make a product
  - Documentation ...
  - Community ...
  - Performance ...
  - Courses ...

Documentation

erlang vsn 1.05

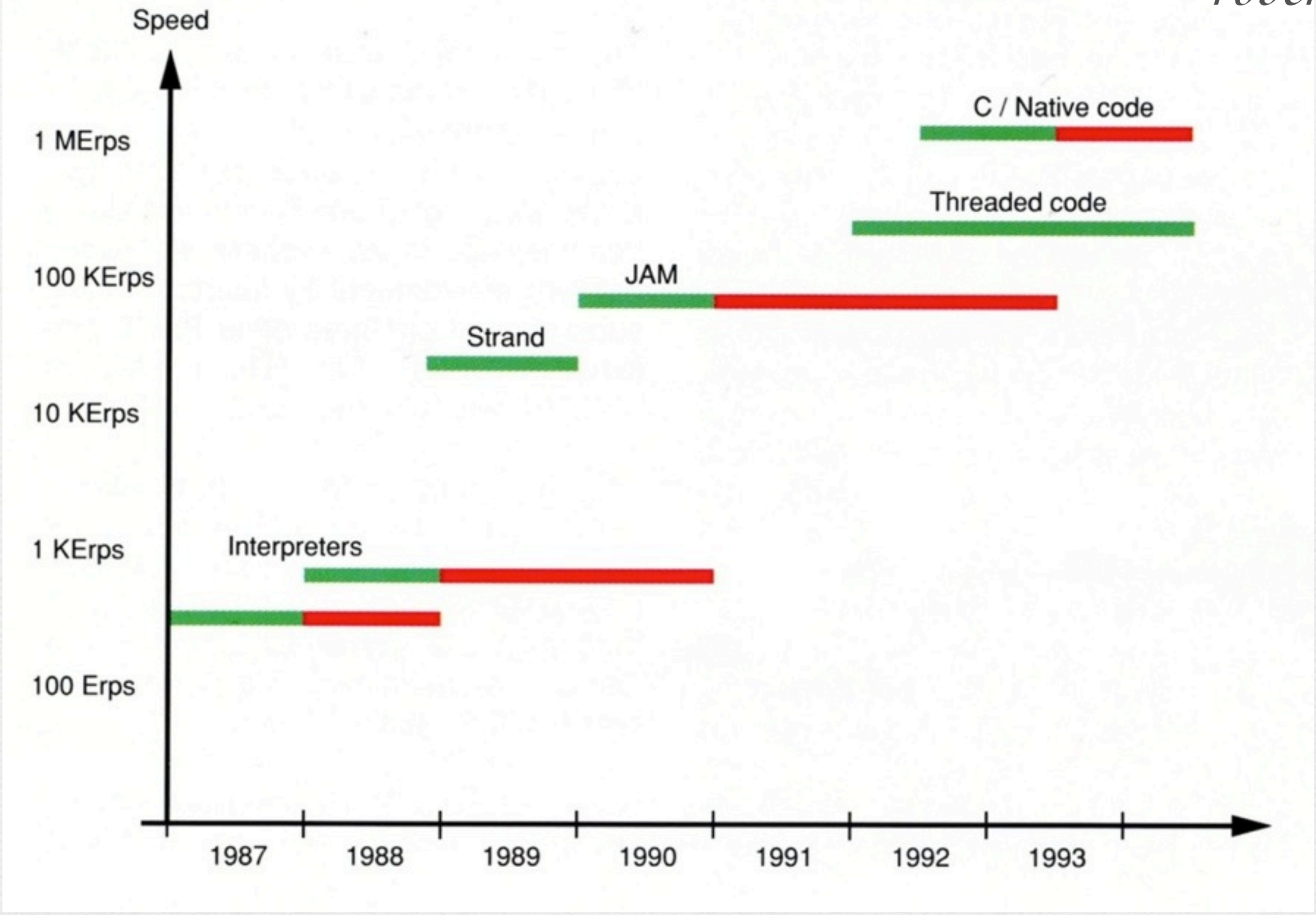
h	help
reset	reset all queues
reset_erlang	kill all erlang definitions
load(F)	load erlang file <F>.erlang
load	load the same file as before
load(?)	what is the current load file
what_erlang	list all loaded erlang files
go	reduce the main queue to zero
send(A,B,C)	perform a send to the main queue
send(A,B)	perform a send to the main queue
cq	see queue - print main queue
wait_queue(N)	print wait_queue(N)
cf	see frozen - print all frozen states
eqns	see all equations
eqn(N)	see equation(N)
start(Mod,Goal)	starts Goal in Mod
top	top loop run system
q	quit top loop
open_dots(Node)	opens Node
talk(N)	N=1 verbose, =0 silent
peep(M)	set peeping point on M
no_peep(M)	unset peeping point on M
vsn(X)	erlang vsn number is X

Performance

Why own VM?

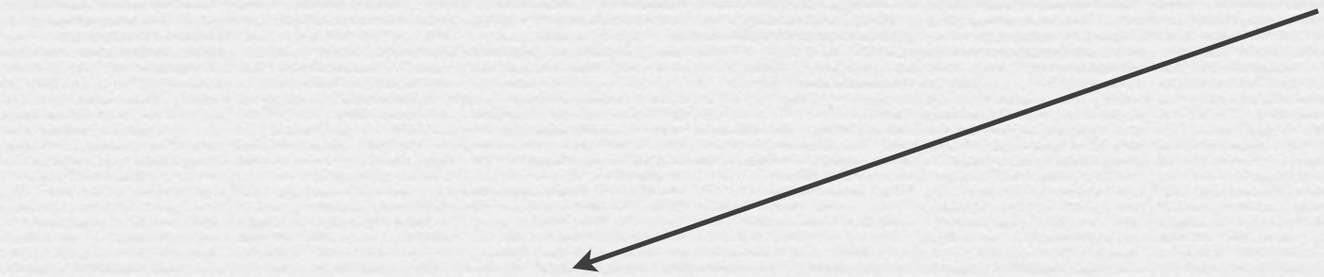
- Performance
- Semantics (code change)/isolation/real-time GC
- How?
  - byte code interpreter (inspiration P-code, WAM)


Excellent
Good
Average
Poor


Speed
1 MErps
100 KErps
10 KErps
1 KErps
100 Erps
C / Native code
Threaded code
JAM
Strand
Interpreters
1987
1988
1989
1990
1991
1992
1993

Improving Performance

- Experiments
- Parlog
- Strand
- ...
- Own VM
- Compiler and Emulator in Prolog



"do it in C"

35 ERPS

Mike Williams
reads Joe's C
and declares it it be "the
worse program ever
written"

The JAM Erlang virtual machine

- JAM = Joe's abstract machine
- Joe → Compiler (Prolog → Erlang) + Architecture
- Mike → VM in “CV”
- I thought I really knew how to program “C” until I started to program the JAM
- VM
  - Byte code instructions
  - 32 bits: 8 bits tag, 24 bits data/pointer
  - Each Erlang process has it’s own separate stack and heap
  - Garbage collection per Erlang process

JAM

First version on a VAX 11/750

- 3 MHz clock, 8 MByte memory, about 300 Mbyte disk

Second version on SUN workstation

- Motorola 68K processor
- Later Sparc

First use in product on “Mobility Server”

- OS – VXWorks, processor 68 K

ETS (Erlang Term Storage) added later

- Enabler for the Mnesia real time fault tolerant database.

Some Later products

- Anx – ADSL DSLAM
- AXD 301 ATM Switch
- SGSN MME (Data access for GSM GPRS, WCDMA and LTE)

Other VMs

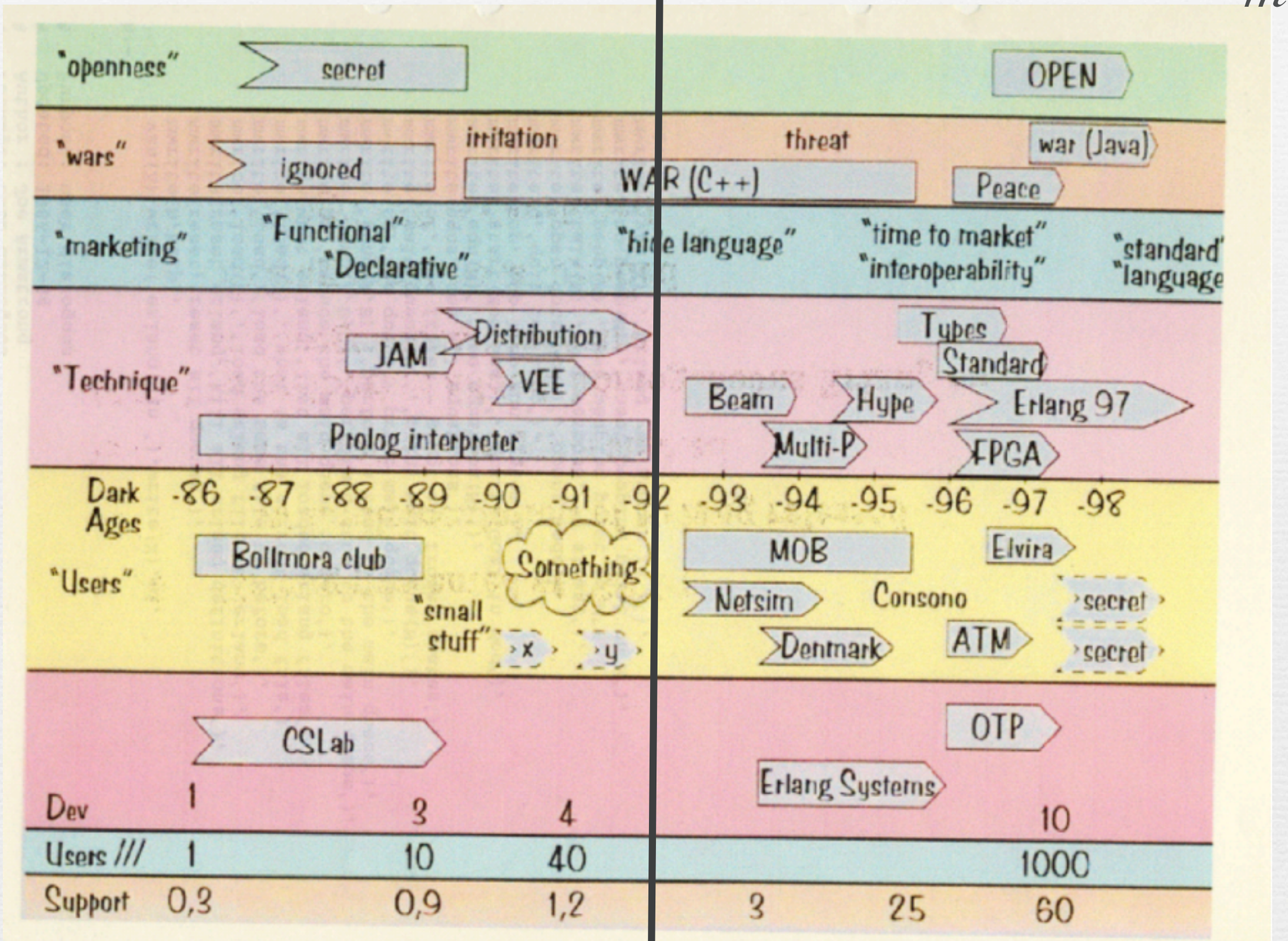
- VEE = Virding's Erlang Engine
- BEAM = Bogdan's (nowadays Björn's) Erlang abstract Machine
  - BEAM has replaced JAM in all Ericsson products

Fault and Failure Handling

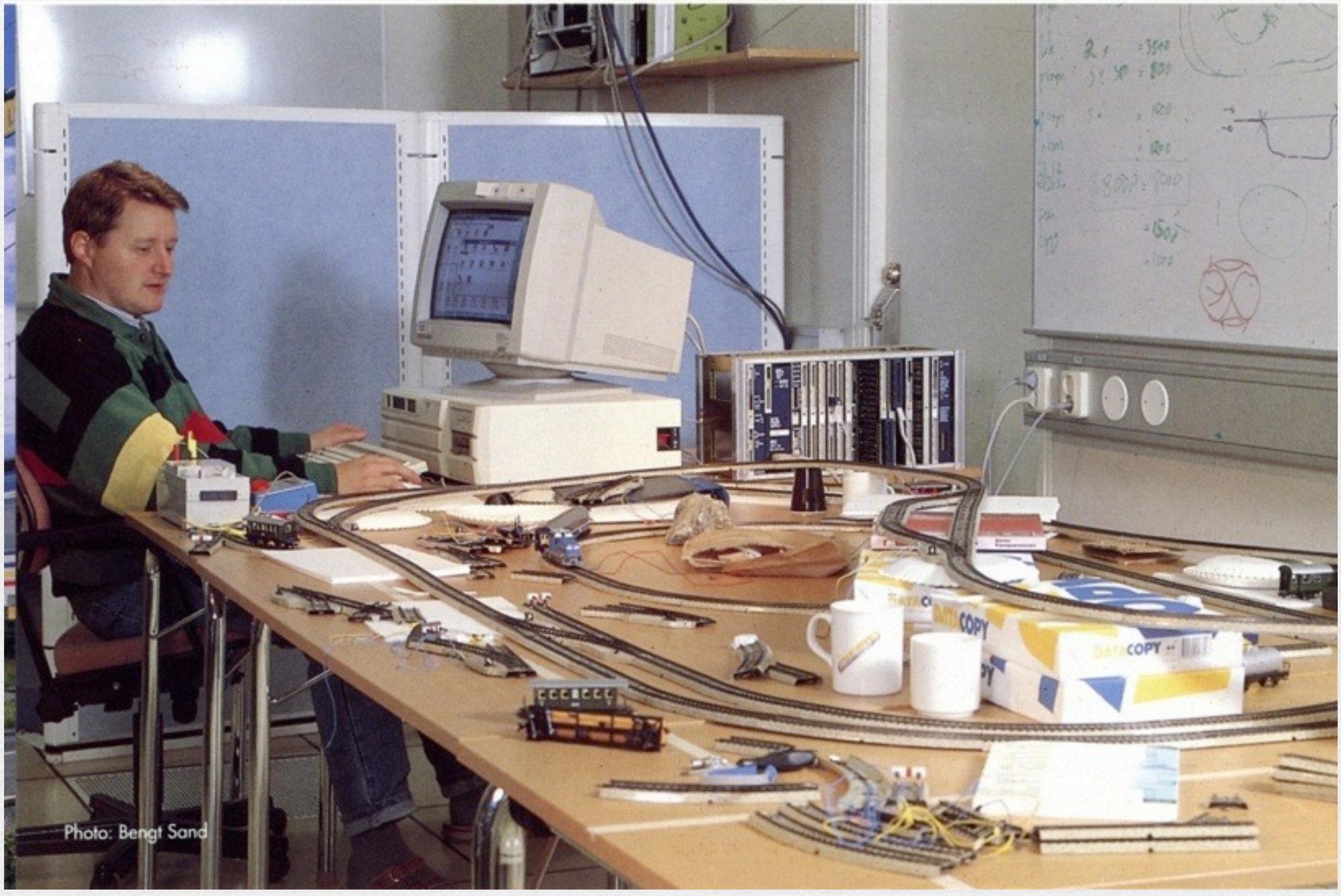
- Fault = bug in code
- Failure = hardware breaks
- Concept:
  - Faults cannot be handled in the same context (i.e. Erlang process) as they occur
  - Failures cannot be handled in the same hardware which is broken
  - Code which handles faults and failures must be as simple as possible.
- Error handling concepts inspired by the “C” wire in ancient relay based telephone exchanges
- Concept of linked process means that if one of them crashes (fault or failure) they all terminative
  - Except super simple recovery processes which receive information about the fault/failure and take remedial action.
- Often used principle:
  - Put steady state data in the Mnesia, let failing transactions crash, recovery processes use data in Mnesia to restore stable state.

Things you may not have thought about

- Dynamic typing
  - Makes tracing and debugging a lot easier as lot of symbolic information is retained
  - Makes mashalling of data for inter-machine communication easy at runtime
- Being able to change code “on the fly” at runtime greatly speeds up the
  code->test->debug->correct
  cycle
- Distribution is transparent in nearly all the code!
- Selective message reception greatly simplifies state machine code
- You can implement synchronous interprocess communication on top of asynchronous communication, the the inverse is very much harder!


"openness"
secret
OPEN
"wars"
ignored
irritation
WAR (C++)
threat
war (Java)
Peace
"marketing"
"Functional"
"Declarative"
"hi e language"
"time to market"
"interoperability"
"standard"
"language
"Technique"
JAM
Distribution
VEE
Prolog interpreter
Types
Standard
Beam
Hype
Erlang 97
Multi-P
FPGA
Dark Ages
-86 -87 -88 -89 -90 -91 -92 -93 -94 -95 -96 -97 -98
"Users"
Bollmora club
"small stuff"
Something
x
y
MOB
Elvira
Netsim
Consono
secret
Denmark
ATM
secret
CSLab
OTP
Erlang Systems
Dev 1 3 4 10
Users /// 1 10 40 1000
Support 0,3 0,9 1,2 3 25 60

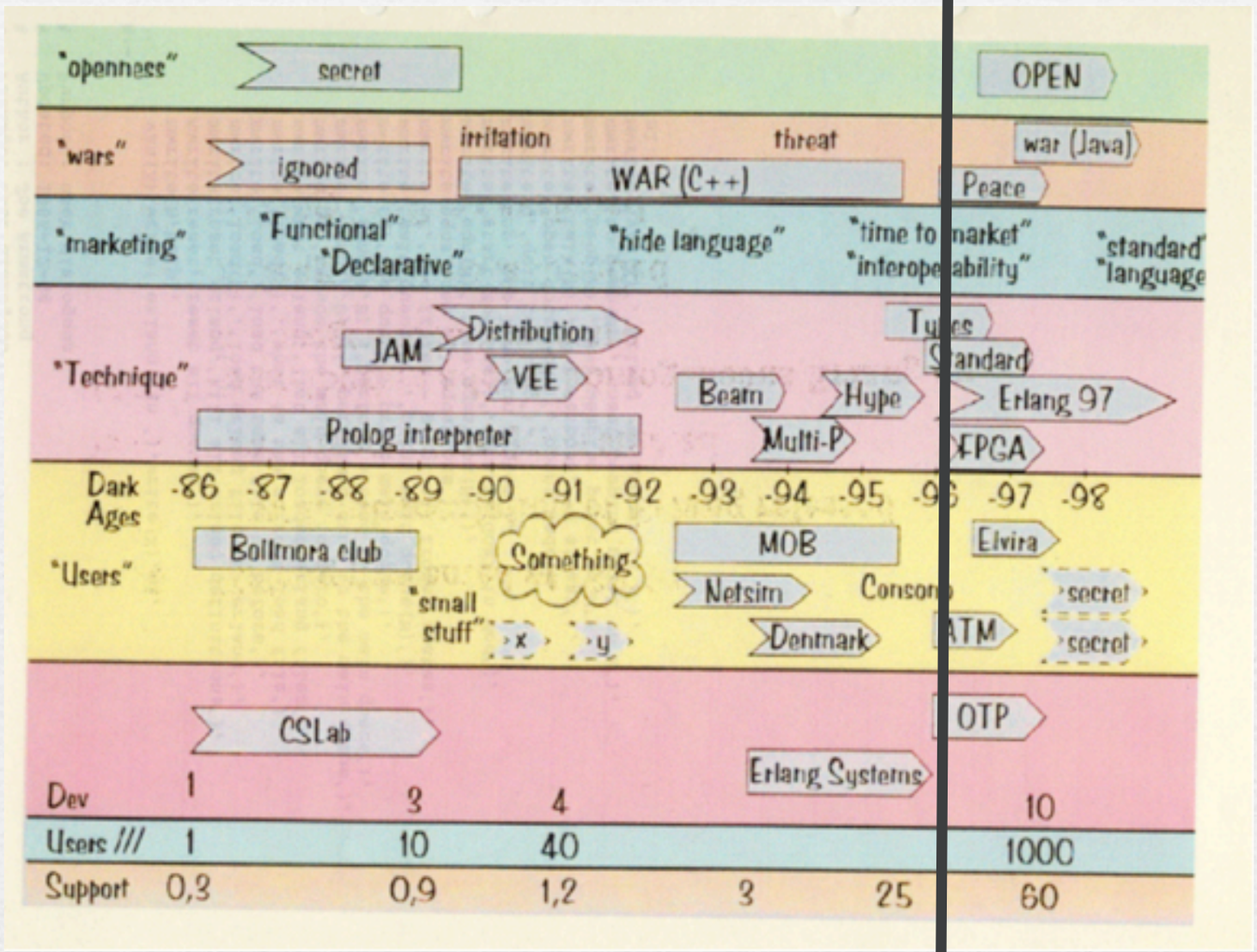
Rapid Prototyping c.1992



1992 - 1995 nothing much happens ...

8 Dec 1995

AXE-N Cancelled ...


"openness"
secret
OPEN
"wars"
ignored
irritation
WAR (C++)
threat
war (Java)
Peace
"marketing"
"Functional"
"Declarative"
"hide language"
"time to market"
"interoperability"
"standard language
"Technique"
JAM
Distribution
VEE
Prolog interpreter
Beam
Hype
Multi-P
Types
Standard
Erlang 97
FPGA
Dark Ages
-86 -87 -88 -89 -90 -91 -92 -93 -94 -95 -96 -97 -98
"Users"
Ballmora club
Something
"small stuff"
x
y
MOB
Netsim
Denmark
Elvira
secret
ATM
secret
CSLab
Erlang Systems
OTP
Dev 1 3 4 10
Users /// 1 10 40 1000
Support 0,3 0,9 1,2 3 25 60

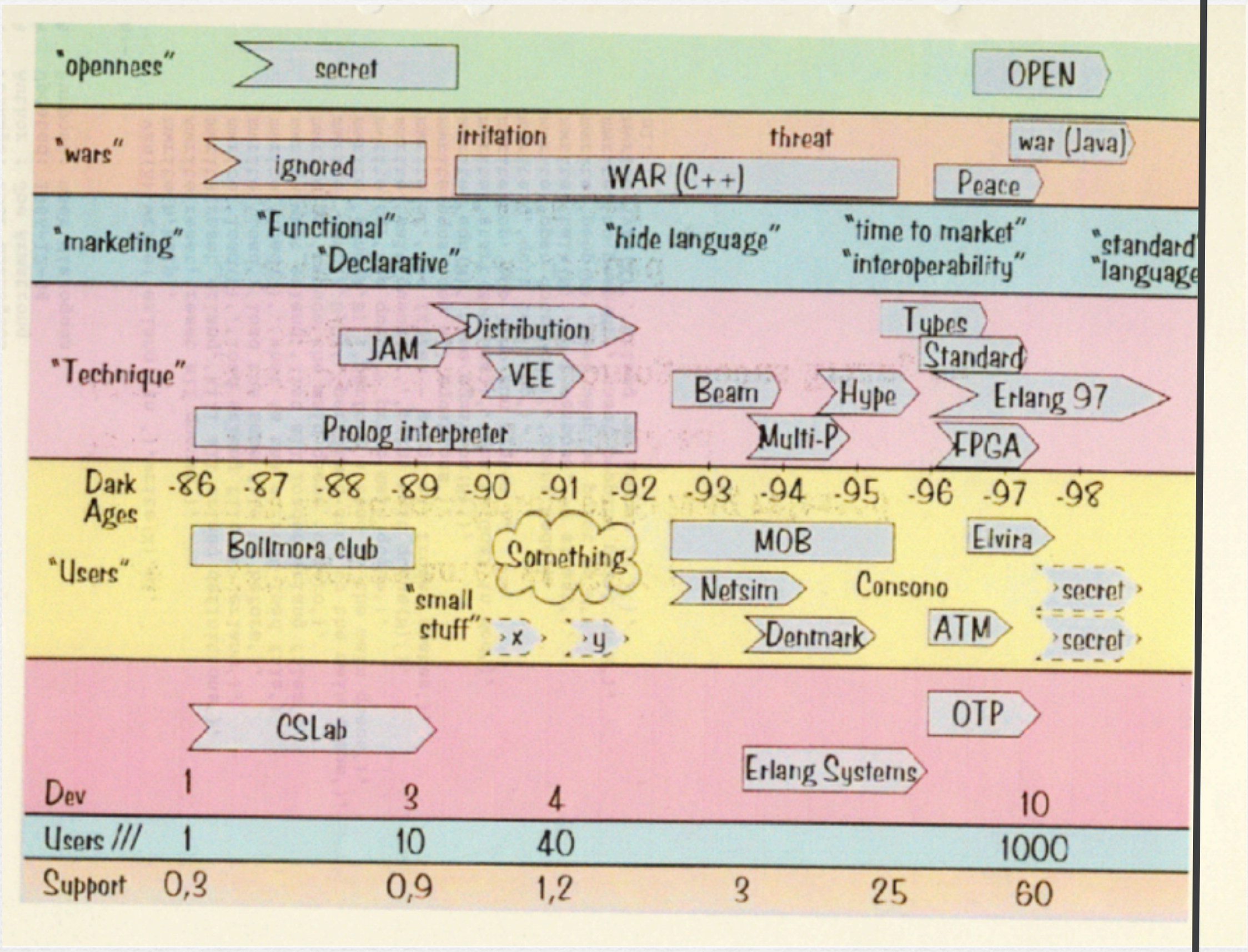
1996
AXD 301 starts

Lot's of stuff happens quickly

1996 - 1998
nothing much
happens ...

1998
AXD 301 is a great success ...

Still in use today in BT network


"openness"
secret
OPEN
"wars"
ignored
irritation
WAR (C++)
threat
war (Java)
Peace
"marketing"
"Functional"
"Declarative"
"hide language"
"time to market"
"interoperability"
"standard"
"language"
"Technique"
JAM
Distribution
VEE
Prolog interpreter
Beam
Hype
Multi-P
Types
Standard
Erlang 97
FPGA
Dark Ages
-86 -87 -88 -89 -90 -91 -92 -93 -94 -95 -96 -97 -98
"Users"
Bollmora club
"small stuff"
Something
x
y
MOB
Netsim
Denmark
Consono
Elvira
secret
ATM
secret
CSLab
OTP
Erlang Systems
Dev 1 3 4 10
Users /// 1 10 40 1000
Support 0,3 0,9 1,2 3 25 60

BANNED

1998 Stuff Happens

- Lot's of stuff happens quickly
- Erlang becomes Open Source ...
- Four days later ... Bluetail AB ...

10 years later...

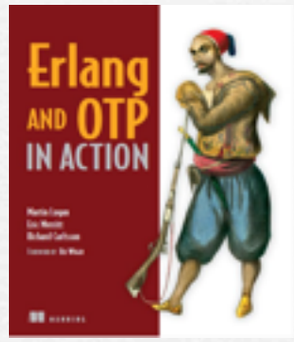
joe


ERLANG
Concurrent Programming in ERLANG
Second Edition
Joe Armstrong Robert Virding Claes Wikström Mike Williams


Erlang
Programming
O'REILLY


Programming Erlang
Software for a Concurrent World
Joe Armstrong


Erlang
语言编程
O'REILLY


Erlang
AND OTP
IN ACTION


プログラミング
Erlang


GOTOP
Erlang程式設計
Joe Armstrong


Concurrent Programming in ERLANG
Joe Armstrong
Robert Virding
Mike Williams


Building Web Applications with Erlang
Early Release
O'REILLY

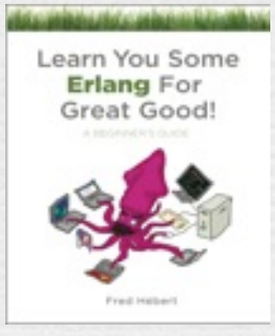

Erlang
programmation

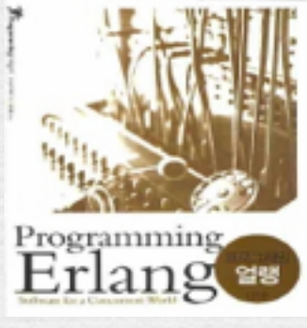

Erlang/OTP


Erlang
プログラミング
O'REILLY


Mastering
Erlang
Geoff Cant


プログラミング言語
Erlang
入門
ASCII

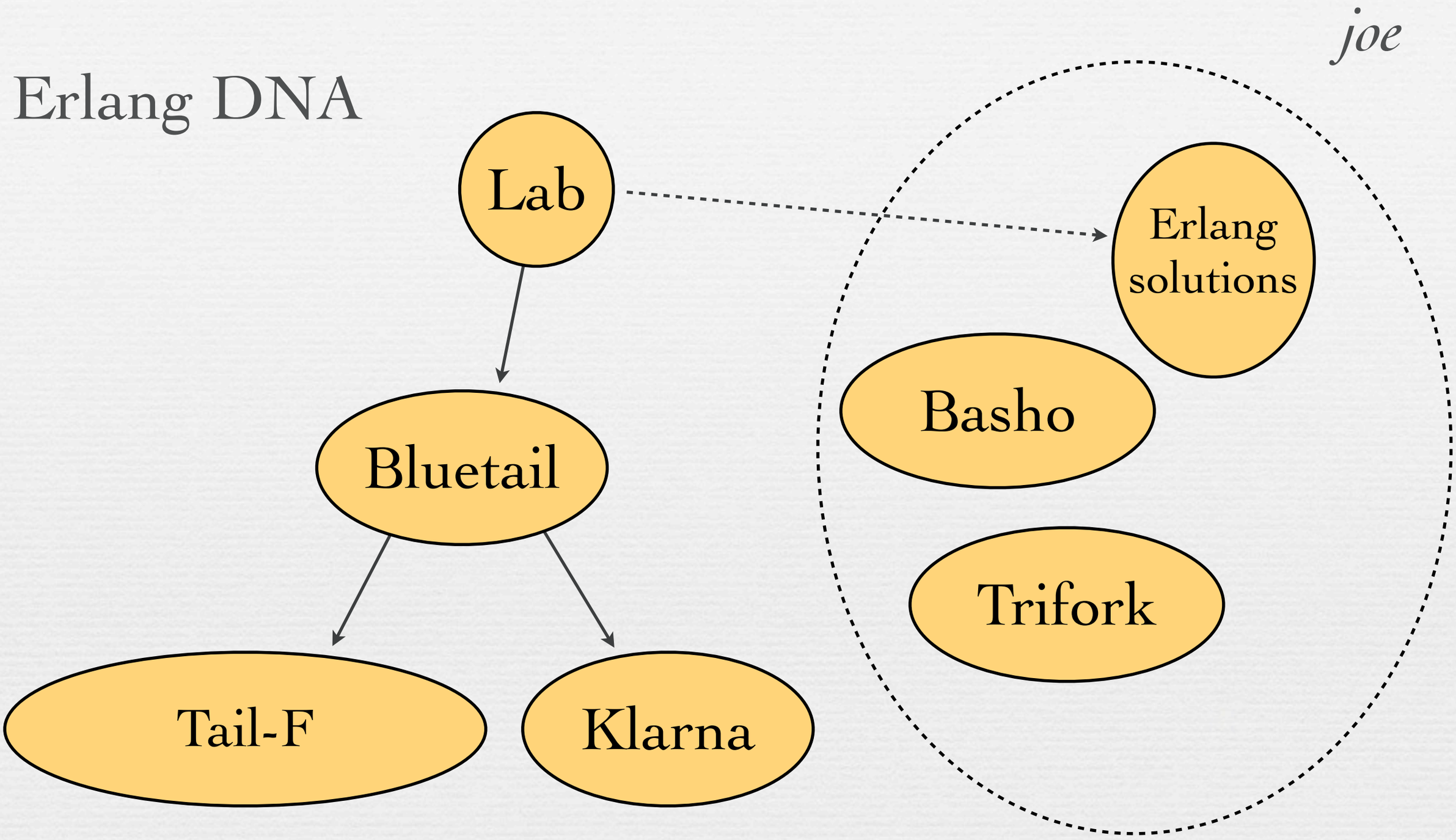

Learn You Some
Erlang For
Great Good!
Fred Hébert


Programming
Erlang
O'REILLY


Programmer en
Erlang

THE LINUX FOUNDATION
CLOUDANT
The Boston Globe
NOKIA Connecting People
ERICSSON
Erlang SOLUTIONS
AOL
QuviQ
hp
DuoMark
github
codecentric
CouchBase
PODIO
OPSCODE
BLOOM DIGITAL PLATFORMS
basho
klarna
CloudBees
bounday
FEUERLABS
LShift
heroku
Chango
2600hz OPEN SCALABLE TELECOM
rackspace HOSTING
vmware
afiniate
wooga world of gaming
vocalocity powering small business
Spawngrid
BARQUE ENERGY GROUP
KYOTO UNIVERSITY FOUNDED 1897
travelping

Erlang DNA



And loads more that we don't know about ... (ask Francesco)

The Future

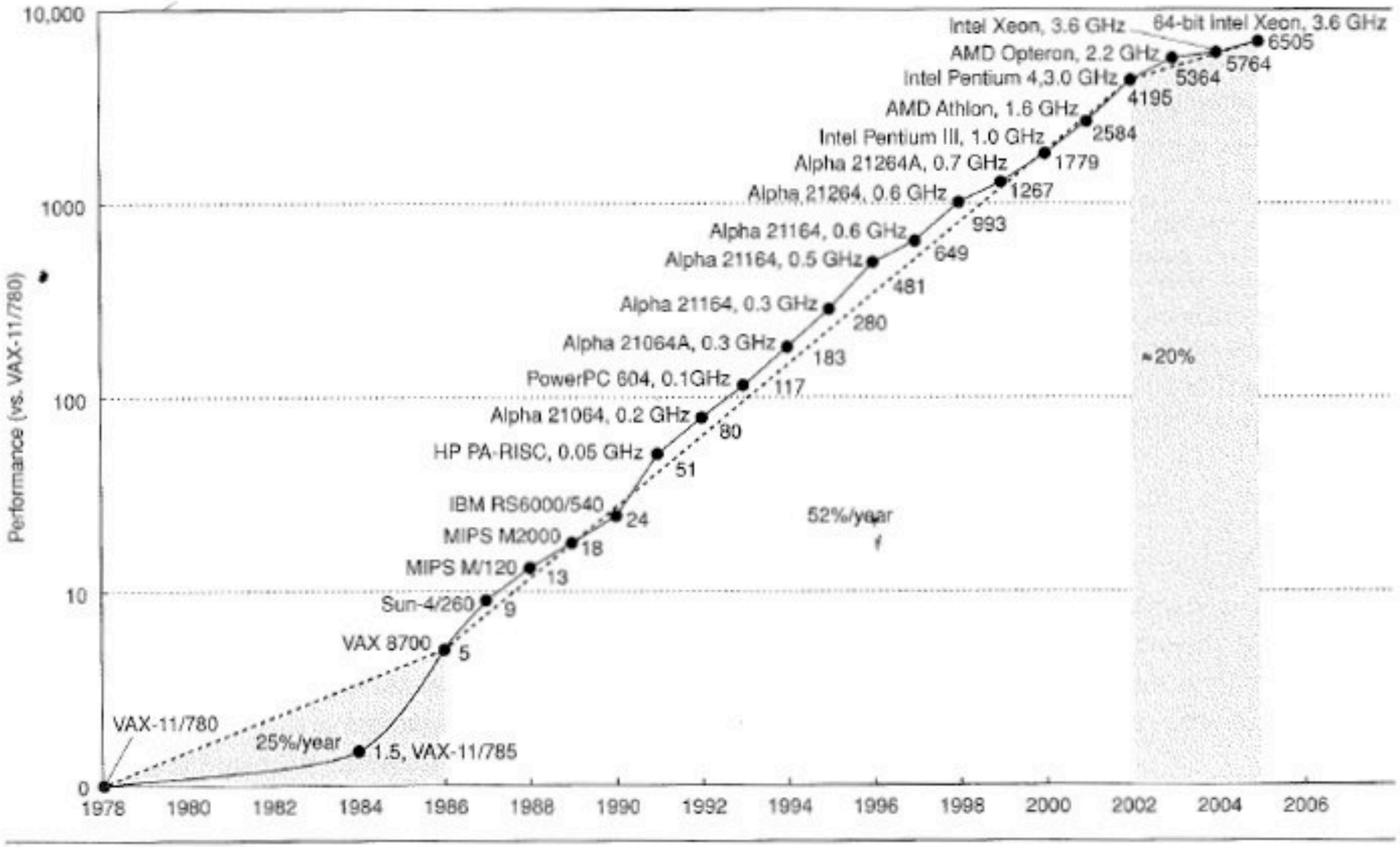
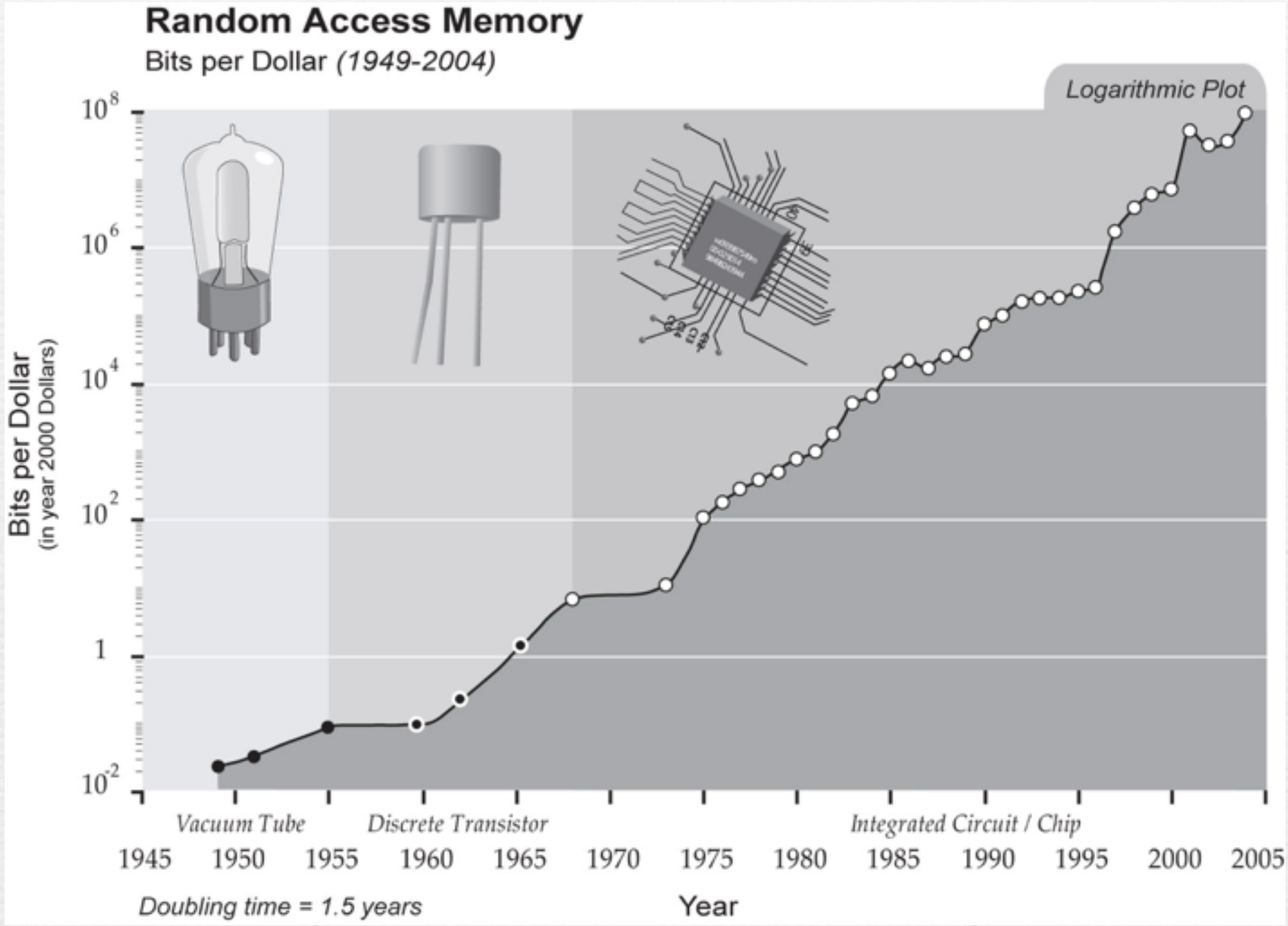


Figure 1.1 Growth in processor performance since the mid-1980s.



source: www.singularity.com

What happened?

joe

Software

- 1879 - Frege
- 1930 - Curry
- 1958 - LISP
- 1969 - agents/actors/smalltalk
- 1972 - Prolog
- 1978 - CSP
- 1983 - Occam (+hardware)
- 1986 - Parlog/Strand
- 1986 - Erlang
- 2011 - Elixir

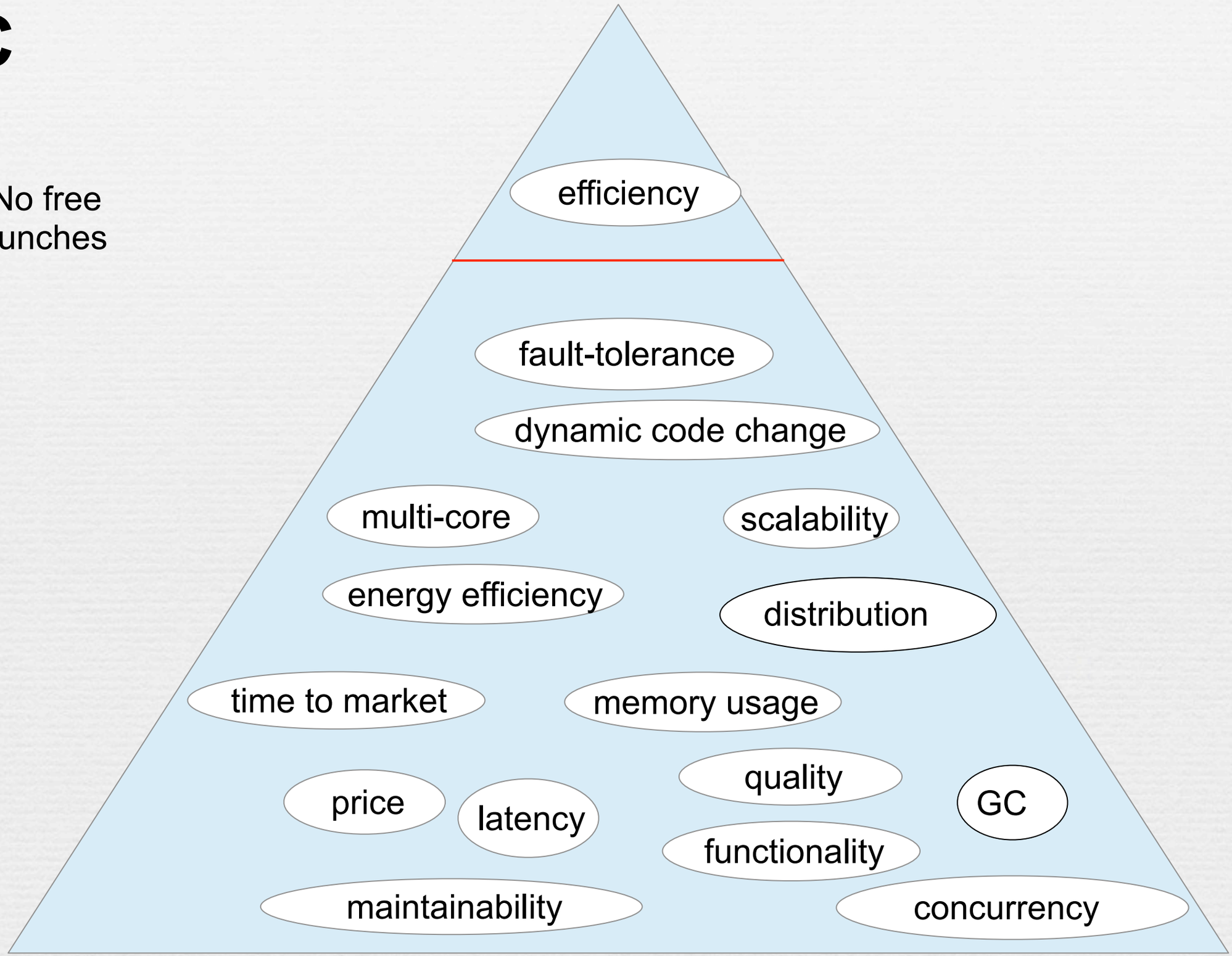
A heck of a lot of hardware stuff has happened in the last 10 years

Hardware - sort

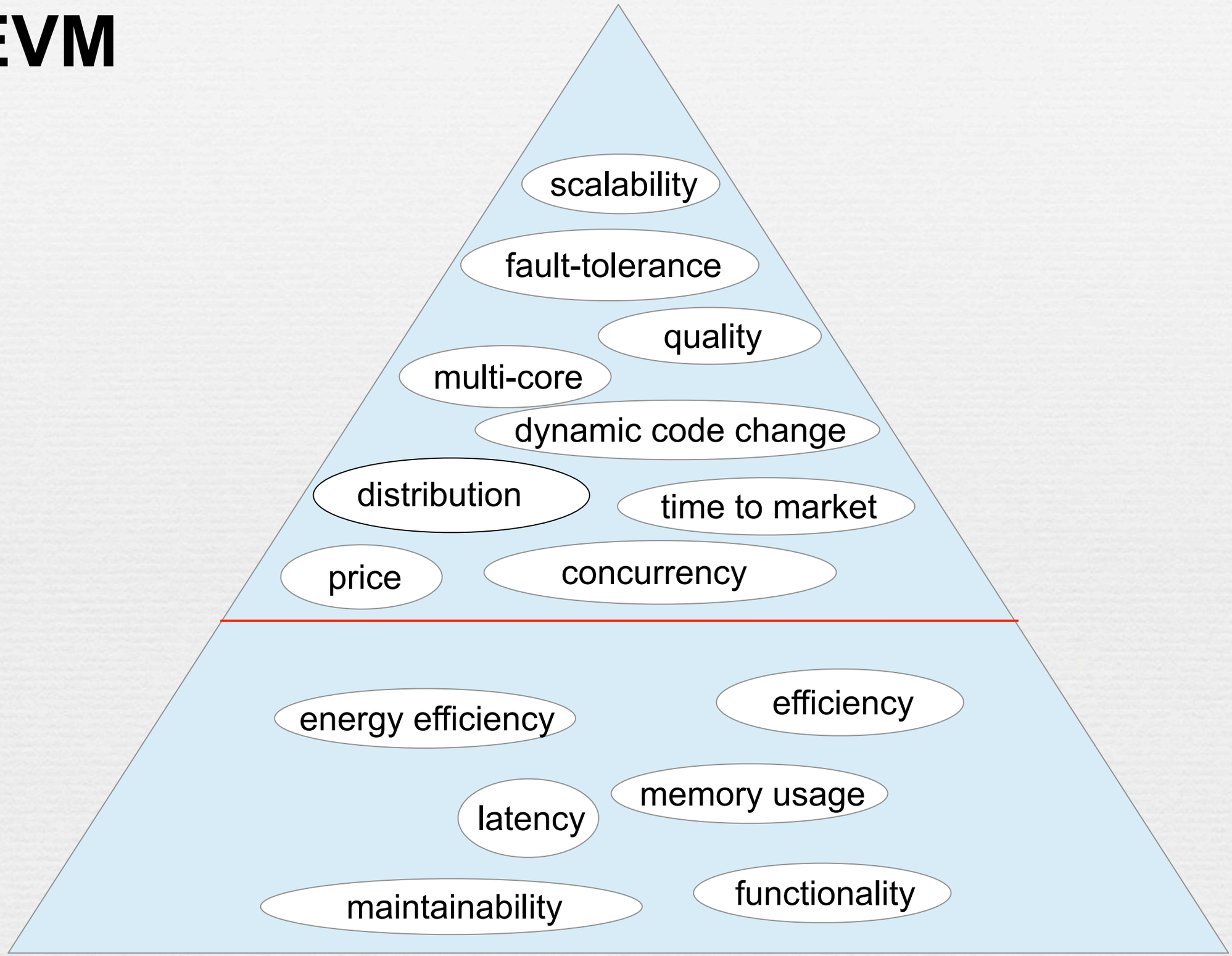
- 1879 - 0 Hz - 0 MB
- 1958 - ...
- 1980's TCP/Internet
- 1986 - 1.6 Mhz
- 2000 - 1GHz clocks
- 2000 - Always connected
- 2000 - Mobile revolution
- 2004 - GPRS/3G/WCDMA
- 2004 - multi cores /GB Ram/
- 2010 - 4G (LTE)/TB disk
- 2020 - Peta bytes?/K Cores

C

No free
lunches



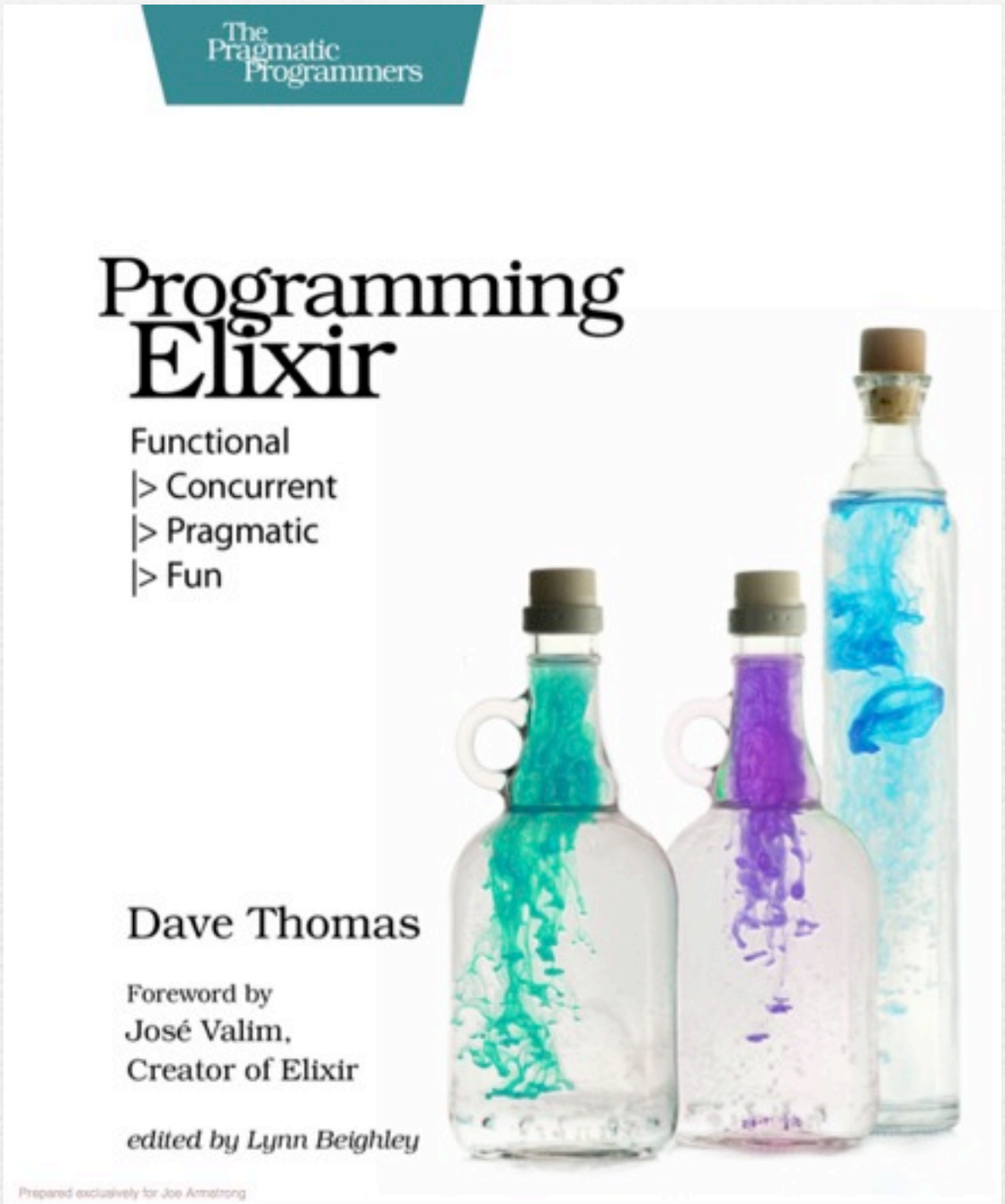
EVM


scalability
fault-tolerance
quality
multi-core
dynamic code change
distribution
time to market
price
concurrency
energy efficiency
efficiency
memory usage
latency
functionality
maintainability

Languages on the EVM

Erlang
Prolog
LFE
LUA
Elixir
Joxa (lisp)
Reia

Elixir


The Pragmatic Programmers
Programming Elixir
Functional
|> Concurrent
|> Pragmatic
|> Fun
Dave Thomas
Foreword by
José Valim,
Creator of Elixir
edited by Lynn Beighley
Prepared exclusively for Joe Armstrong


Introducing Elixir
Simon St. Laurent and J. David Eisenberg
O'REILLY
Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

Erlang

- Language discussion limited by geography (pre WWW)
- Closed source
- “Funny syntax”
- Started 1986
- Book1 1993
- Book2 2007 (21 years later)
- Not marketed

Elixir

- Language discussion on WWW
- Open source
- “Ruby syntax”
- Started 2011
- Books1+2 2013 (2 years later)
- Marketed

Tomorrow

joe

- 2020 - 1 Million cores
- 2020 - 1 TB flash in mobile / 1 GB/sec mobile PByte disks
- 2020 - 100 B connected devices / ubiquitous networking

Today

joe

- MME (=Mobile Management Entity (LTE/4G))
- SGSN (=Serving GPRS Support Node (GPRS = General Packet Radio Service, 3G))
- WhatsApp

How will we program all this new stuff?

Which X on the EVM?