From WhatsApp
to
Outer Space

Joe, Robert and Mike

Where are we today?

Many enterprises use Erlang for a signification part of their infrastructure. This gives them a commercial advantage in terms of time-to-market, slimmer code base, fewer programmers.

WhatsApp (Facebook)
Machine Zone
Ericsson
Tail-f (Cisco)
Klarna
Basho
Bet365
Whisper
T-Mobile
AddRoll

Sqor
Process one
Erlang Solutions
Vocalink
RabbitMQ
Opscode (Chef11)
Infoblox
And a lot of others!!

And there are some BIG names we can't mention

What do they do?

- Messaging
- Control mobile telecom
- Network management
- Banking backends
- Data base management
- Online gaming
- Online advert brokering
- *And a lot of other things some we don't even know about*

What do they have in common?

- They are all server applications
- Several are multi-processor
- All serve huge numbers of concurrent transactions
- “Zero” downtime and fault tolerance are important

It's not surprising,
that's what we
developed
Erlang for!

Most “serious” Applications Need Several Software Technologies!

- Erlang fits the bill for highly concurrent, distributed fault tolerant applications
- Erlang doesn't work so well for DSP or other computationally critical applications but is good for coordination of products using DSPs, FPGAs
- You can use Erlang for graphics, but it is maybe easier to use something else?
- Maybe we should work to expand the areas where Erlang wins?

Teaching Erlang
Training
How to get people to understand Erlang

The Short Answer

It's easy!

Honestly it is!

The Long Answer

The Major Problems

The Lesser Problems

The Major Problems

- Erlang is functional
- The concurrency model
- Developing an Application Architecture

Each a major rethink!

Erlang is Functional

- Immutable data
- Pattern Matching
- Recursion
- ...

Concurrency Model

- Asynchronous Communication
- Selective receiving of messages
- No shared state
- ...

The Application Architecture

- How to build a system architecture

The Lesser Problems

- Syntax
- ...

Syntax

Yes, most people think the syntax strange

BUT

Syntax

Most functional languages have a strange syntax

All languages you don't know have a strange syntax

Syntax

By the time most people have grasped the basic concepts they know how to express them

Projects using Erlang

- It is easy to build prototypes with Erlang!
- Try things out “in the small” before you start development with a large team!
- Re-write code and develop architectures until you are happy with them
- Keep development of automated test suites running at the same pace as development of the application software
- Make sure that tools like Dialyzer are used for the start.

Starting up and maintaining

- There is a huge difference between:
  - Starting a project for a new application
  - Maintaining an existing product
- You need different
  - people
  - goals
  - Schedules and project management

Erlang

Where are we now?
Where are we going?

Where is the software industry going?

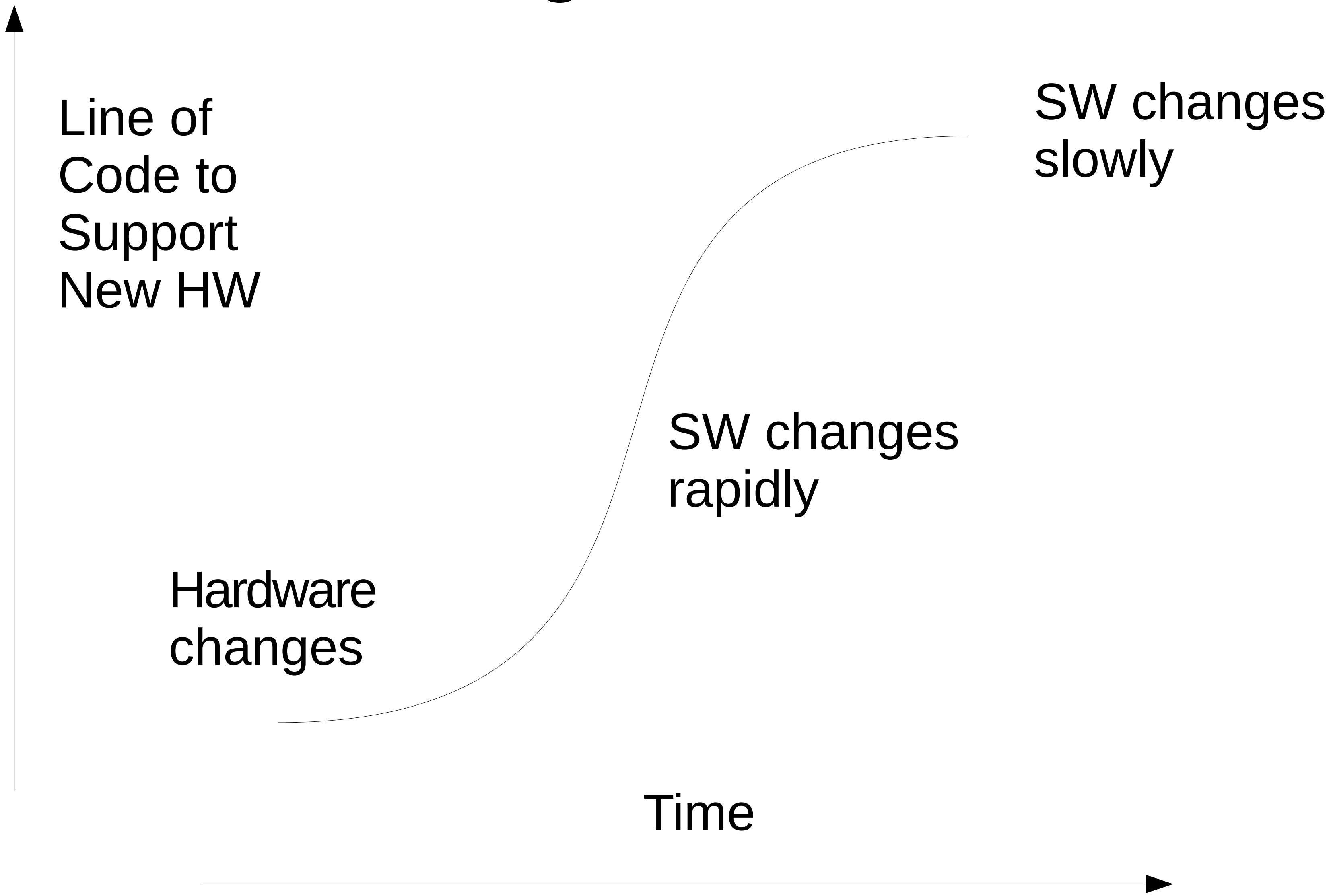
Erlang's initial design reflects the hardware of the mid 1980's

- Small memory (a few MB RAM – single CPUs)
- Small number of nodes (10's of nodes)
- Closed (Behind a firewall – poor security)

1985: How do we map a large number of parallel activities onto a small number of CPUs?

2015: How do we map a large number of parallel activities onto a massive number of CPUs?

Changes in SW follow changes in HW



Line of Code to Support New HW

SW changes slowly

SW changes rapidly

Hardware changes

Time

- If the hardware changes the SW will change
- Big changes to hardware = Big changes to SW
- SW takes a long time to catch up (long tail to the S curve)
- “Good enough” catch-up is rapid
- Commercial advantage is in the middle of the S curve

Changes in theory take even longer ...

- 1933 - Alonzo Church – Lambda Calculus
- 2014 – Java 8 – gets Lambdas

  (only took 91 years)

- 1879 – Gottlob Frege - *Begriffsschrift*
- 1986 – Erlang :-)

What are the big changes to hardware/theory that will drive SW in the future?

Hardware

- Massive NOC/SOC chips

  1000 General purpose CPUs 100 hardware accelerators – 1 MB/CPU

- Peta – Exabytes of local storage

  Non Volatile memory?

- High speed communication
- Massive numbers of connected devices
- Memory is free, CPU is Free, Communication Costs

Data rates drive the industry

- 2G – GSM 10Kb/s → 64 Kb/s
- 3G – WCDMA 64Kb/s, 384Kb/s, 2Mb/s

  HSPA up to 15Mb/s
- 4G – LTE 100 Mb/s. LTE-A up to 1 GB/s (nomadic)
- 5G – 10Gb/s “hot spots” 100Mb/s “everywhere”

- 1000 x increase per ten years (10^6 price/bit decrease over 20 years)
- 2014 – wireless access exceeds wireline (which is why WhatsApp was worth so much)

Mobile data is the bottleneck – this is where the shoe pinches – get to the fibre and the rest is easy

The New Problems

- 50 Billion connected devices
- 1000+ core computers
- Limited Energy (how much energy does it take to store 1GBye of data in the cloud? - what is the energy cost of a big data analysis – what does a Google search cost – in Joules)
- Limited radio bandwidth – data rates are high BUT total capacity is limited by laws of physics

The New Hardware Landscape

- 50 Billion connected devices
- 5G (I have to say this :-)
- 1000+ core CPUs
- 5 Gb/s radio access
- CPU power is free
- Memory is free
- Communication costs
- Energy costs

The New software problems

- Manage 50 Billion crypto keys
- Manage the SW versioning on 50 Billion devices and the network
- Program 1000+ core CPUs with HW accelerators
- Fix the “old mess” - be backwards compatible
- Save energy (zero environmental impact)
- Increased system complexity

The Future

- Self managing

  Version control/security/authentication/privacy become

  key problems

- Self repairing

  What comes after supervision trees? Machine learning?? AI??? neural nets????, Chaos Monkeys?????

- Energy Aware

  Measure Measure Measure -

- Privacy

Where is Erlang today?

- Language of choice for programming “soft” RT distributed applications.
- Language of choice for multicore (excluding GP/DSP type SW)
- 20 years battle tested experience with highly concurrent fault tolerant systems with self-repair
- Great for rapid prototyping distributed programs
- Can manage “a few” million devices from one Erlang machine – need DHTs etc to scale to billions

“The best way to predict the future is to invent it”

— Alan Kay