The True story about why we invented Erlang and A few things you don’t want to tell your Manager

History

- It all started in the 1980's.
  - Ericsson lived from selling AXE - a telecoms switch
  - Own developed processor (APZ)
  - Own programming language (PLEX)
  - Own Software development environment
  - Own Operating system
- Sounds nuts today, but at that time it made a lot of sense
  - We earned (and still earn) a huge amount of money (many billions) from AXE systems.

What did AXE and PLEX provide?

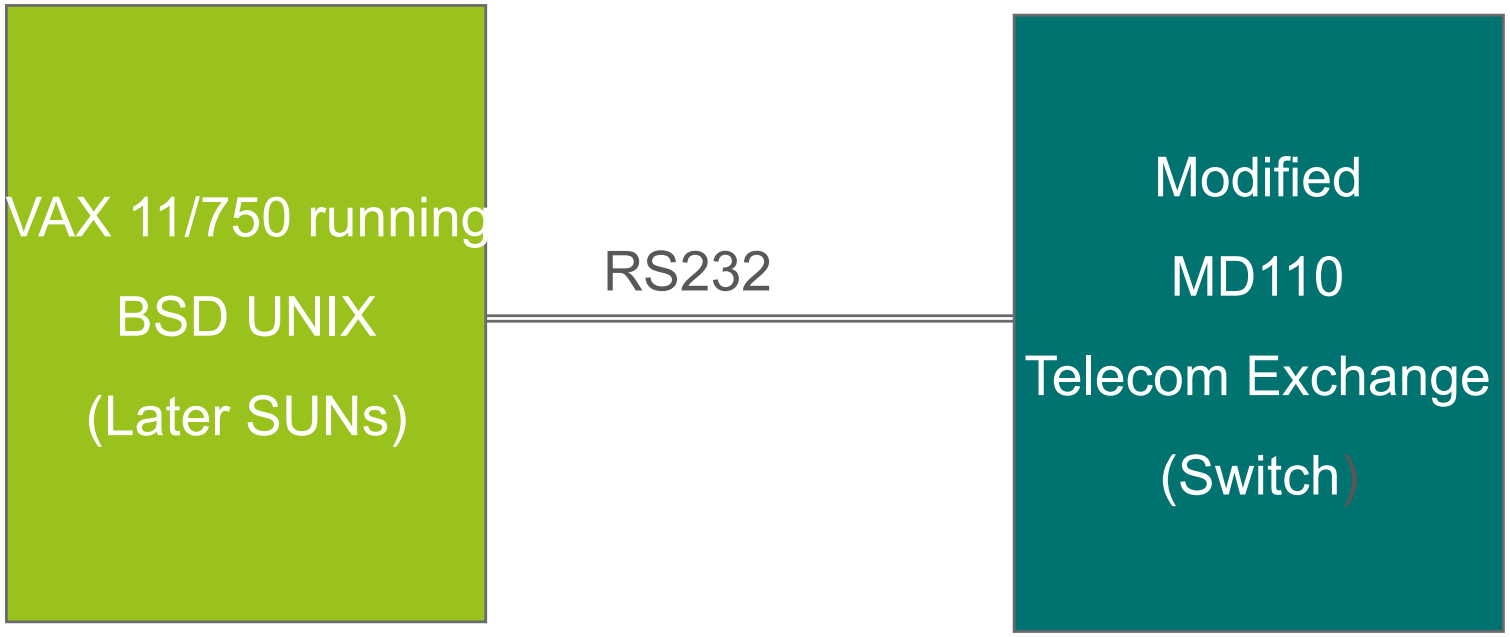
- PLEX is rather low level language (like Basic) but:
- Safe pointers
  - All addresses via one degree of software controlled hardware indirection
- Ability to change size of arrays etc without memory leaks
  - Memory allocation via software controlled hardware indirection
- Fine grained massive concurrency
  - Finite state machines supported in microcode
- Ability to develop software in “blocks” that are independent and can’t interfere with each other
- Ability to change (update) code at runtime without stopping (both by patching and replacing whole blocks)
- Advanced tracing ability at runtime
- Mechanisms to recover software and hardware failure
  - Transactions (Forlopp) (Came later)
  - Restart mechanisms (small and large restart)

Göran Hemdahl

Why Change? What did we do?

- Development of software in PLEX was (is) very time consuming
- Efforts to turn AXE into a multi-processor system failed
- Developing own processors is costly (own special purpose ASIC)
- Mission given to Ericsson's Computer Science Laboratory was
  - Invent a better way to program telecom type applications but retain the same features as PLEX.
- What we did:
  - Practically programmed a small telephone exchange (MD110) in a variety of languages (Prolog, CHILL, Ada, Concurrent Euclid, Rules Based Systems, AI systems, Functional Languages)
  - Scratched our heads very hard
- Conclusion
  - A lot of the approaches were good as regards abstraction etc, but:
  - We could not have the same characteristics as AXE using any of the languages we experimented with!

Hardware Set-UP



So we developed Erlang

- PLEX: Rather low level language
  - No subroutines with parameters
- Erlang: Functional but simple language

- PLEX: Safe pointers
  - pointers checked at run time via ´”reference memory”
- Erlang: No explicit pointers

- PLEX: Ability to change size of arrays etc without memory leaks
  - Memory statically allocated and accessed via reference memory
- Erlang: Dynamically typed, automatic garbage collection
  - automatic memory allocation

- PLEX: Fine grained massive concurrency
  - state machine / signal queue)
- Erlang: Massive processes based concurrency
  - true independent processes, each with a message queue

So we developed Erlang

- PLEX: Ability to develop software in “blocks” that are independent and can’t interfere with each other
- Erlang: Processes which cannot interfere with each other and modules which can be developed independently

- PLEX: Ability to change (update) code at runtime without stopping (both by patching and replacing whole blocks)
- Erlang: Ability to change (update) modules at runtime

- PLEX: Advanced tracing ability at runtime
- Erlang: Advanced tracing ability at run time

- PLEX: Mechanisms to recover software and hardware failure
  - Transactions (Forlopp) (Came later)
  - Restart mechanisms (small and large restart)
- Erlang: Mechanisms to recover from software and hardware failures
  - A processes can detect failures in another process or another computer

What's more

- PLEX used to need a special processor, but nowadays both PLEX and Erlang run in virtual machines on standard processors
- Unlike PLEX, Erlang has mechanisms for supporting distributed processing

Who Did What?

- Joe: Invented the language, wrote the first compiler
- Robert: Made the libraries and I/O system
- Mike: Wrote the first virtual (JAM) machine, stuff for error handling and code replacement
- Klacke: Added distribution and invented ETS, binaries & Mnesia
- Bogdan: Made the first Beam machine
- Per: Provided computer environment and kept the IS/IT people away
- Kostis & Co: Made Beam really fast
- Björn & Patrik: Made Beam even faster
- Kenneth: Keeps the Erlang team together
- Bjarne: Started it all off and provided inspiration (EriPascal)
- Lots more people: Made ASN.1, SMPT, Inets, Corba, etc. etc.

What happened next?

- We tried to market Erlang externally of Ericsson
  - Erlang Systems AB (I was the second CEO, but that's a long story)
  - Only one fully paying commercial customer (in Canada)
  - But great take up of Erlang at Universities world wide
  - Erlang used by Ericsson to develop an enterprise switch
- There was a very large project at Ellemtel to make an AXE replacement (AXE-N)
  - Cost approx (a lot of people).
  - Employed over (very many people)
  - Project failed and was abandoned in 1996
- Erlang took over and was chosen by Ericsson to develop
  - ATM switch
  - GPRS (SGSN & GGSN) for GSM (Data communication for 2G & 3G)
  - OTP developed, a large collection of libraries and components

Erlang Banned!

- Ericsson Radio banned the use of Erlang (1998)
  - Frightened of using little known programming language
  - Decided to focus on C++, CPP (OSE) and modeling with UML
- Erlang/OTP released as Open Source (1998)
  - Slow take up at first

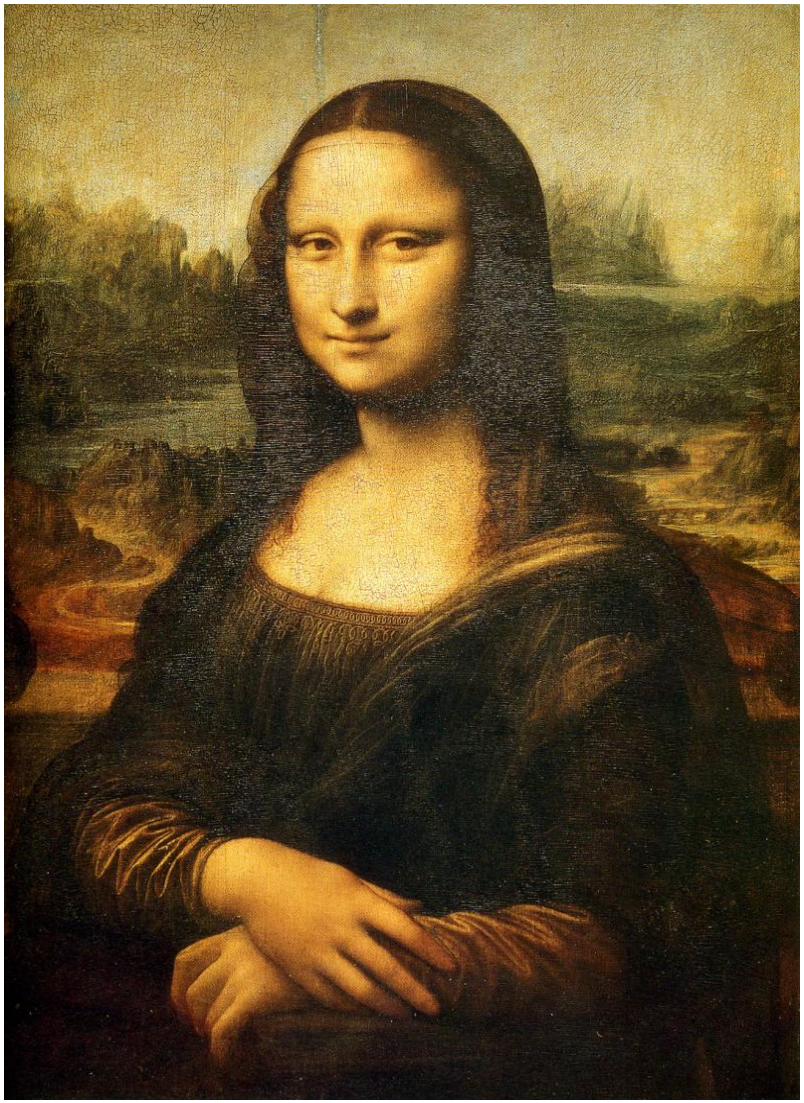
And then it happened

- Computers got cheaper (Moore's law)
- Expansion of the Internet explodes, programming distributed systems became "the norm".
- OTP got more components
- Erlang programming model is very well adapted to Multi / Many core
  - Erlang VM adapted to take full advantage of many / multi core
- Many companies develop products using Erlang / OTP
  - You probably use systems based on Erlang/OTP every day without knowing it!

But what shouldn't You TELL YOUR MANAGER?

SW development is an art, not a science

- Nobody (so far) has invented any reliable way to measure productivity of SW development.
  - Do we ever develop the same SW twice with the same pre-conditions?
- All attempts to use formal methods to derive software from requirement specifications break down as soon as we try to use them on realistically large and non-deterministic systems.
- The same is true of attempts to formally prove the correctness of programs.
- The most commonly used paradigm for SW development is “trial and error”.
- SW developers have almost religious beliefs in which operating system, programming language, editor, etc is best.
- **If you are going to produce “art” you needs the best tools - Erlang**



Mona Lisa Leonardo da Vinci

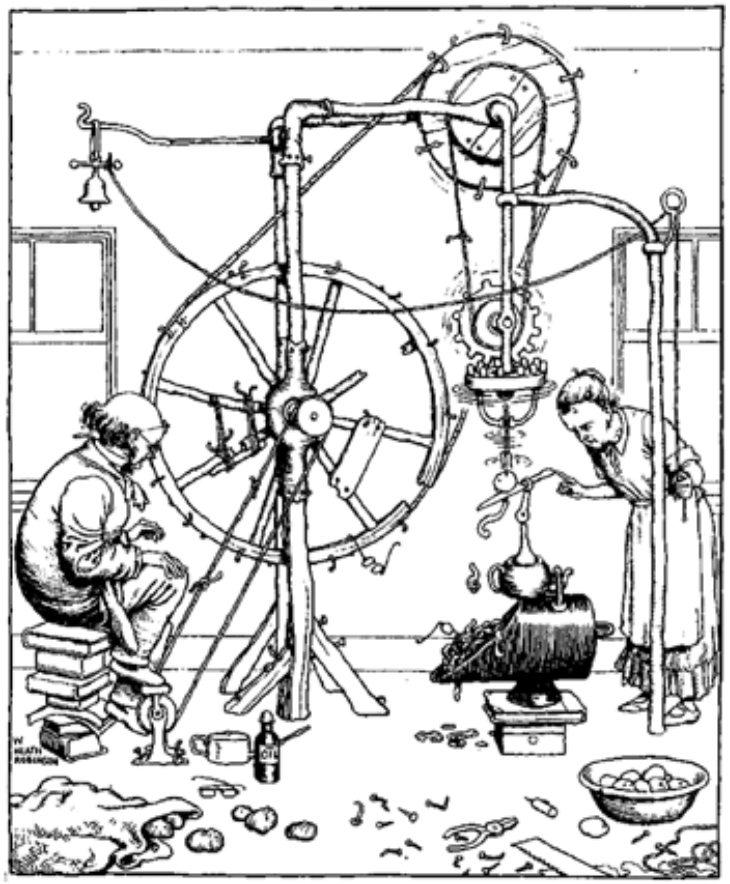
Because SW development is an Art:

- Managers dream about “Software factories”
- Estimating the needed time to develop software is an art!
- Lots of people, who have hardly done any programming at all, believe in wacko methods which are supposed to increase productivity
- There are as many ideas about good programming style as there are programmers!
- Don’t tell you manager that your work is “trial and error”, doesn’t sound very professional

Going to work - Lowry

It’s Not Object Oriented

- Your boss may be worried about funky functional programming languages
- Lot’s of pointy headed managers think object orientation is the same as development efficiency.
- Managers never get fired for following old trends (Java, C++, Micro$oft etc)
- So just tell you boss that Erlang/ OTP is used for financial and banking applications. Banking always seems solid – despite Leman Brothers!



The Professor's invention for peeling potatoes.

IT’s EASY to LEARN

- You don’t want your job to sound too easy, hard work means more pay.
- Best to say, “yes it’s hard to learn, but learning it will make me so much more productive” (I.e. pay me more for being so clever)
- But the fact is to truly understand Erlang is a heck of a lot easier than “C” or “C++”

i = 10;

j = i++ + i;

What is i? 20 or 21?

Erlang has left to right evaluation, in C it’s not even defined!

Machine for knitting socks

It’s OPEN SOURCE

- Lots of managers are worried about open source.
- People doing anything for free is communism!
- If this comes up, just ask you boss what “C++” compiler you are using. The odds are ten to one that it is gcc.
- DANGER: You boss may not know what a compiler is. In this case, you need not worry – your boss probably won’t know what Open Source is either

IT'S EASY TO INTERFACE TO OTHER LANGUAGES

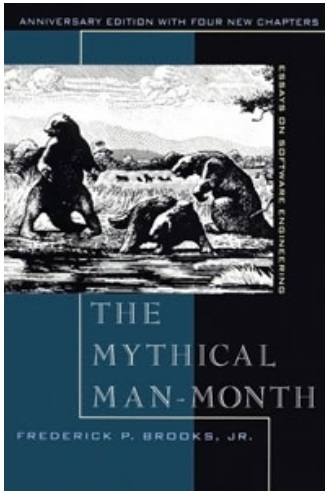
- This is the same as worrying about Erlang not being object oriented. You don't want to make your job sound too easy.
- The recent management trend is to think that software can always be developed using reusable components.
- Your boss may insist that you use some vastly expensive database, GUI builder, management system.
- Play it by ear.
  - Saying it is easy to interface to other systems may mean you are forced to use some wacko components
  - Say it is hard to interface may mean you won't be allowed to use Erlang
  - This is difficult

Re-use isn't free, it costs a lot more than the "purchase price"

Architectural Salvage – Hardware "Re-use"

YOU WoN'T NEED SO MANY PROGRAMMERS

- In many companies the status of a manager depends upon how many subordinates you have
- If it is easy to program, you won't need to many programmers
- You had better keep very quiet about this one!



Fred Brooks

Brooks's law: Adding manpower to a late software project makes it later

C++ and Java are Unnecessary

- If you want to do really low level stuff you may need to use assembly language
- If you have hard real time, or need to write device drivers, virtual machines etc, C is a good choice
- If you are into scripting then you can use Perl, Python etc
- If you want applications you have Erlang, Haskell, OCaml, Sceme etc

- What the heck are C++ and Java good for?

Just to Summarize

- Using Erlang is a very good idea – if you have an application which suits Erlang
- But new approaches always meets some opposition from “pointy haired managers”

Good Luck!