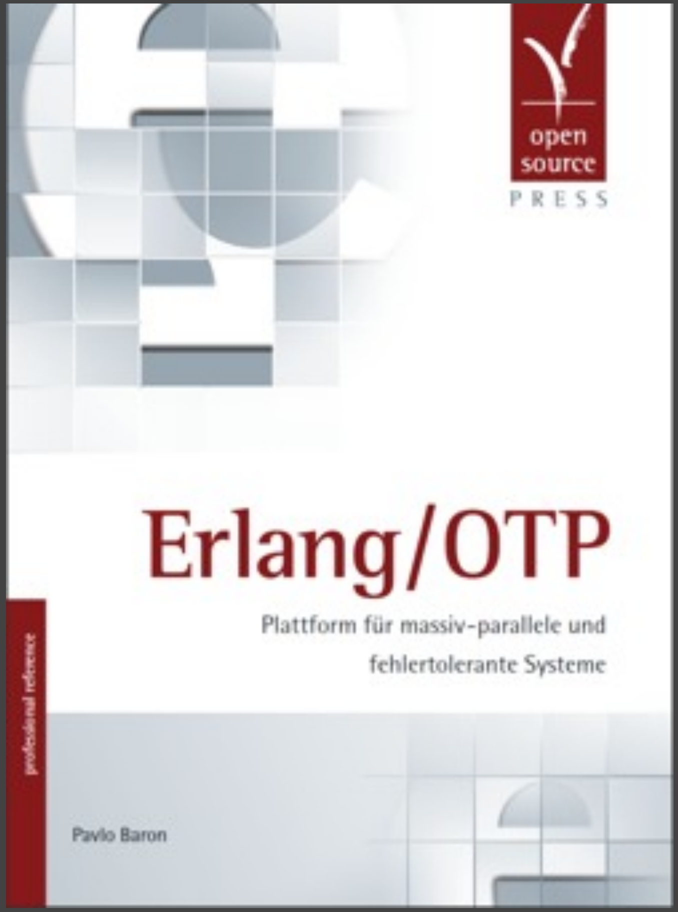
The almost zero-slide story of my journey into Erlang, and why I'll never be the same again

Pavlo Baron, codecentric

entwickler.press
Pragmatische IT-Architektur
Schlanke Technologien und Konzepte für dynamische Unternehmen
Pavlo Baron



pavlo BARON
BIG DATA
FÜR IT-ENTSCHEIDER
RIESIGE DATENMENGEN UND MODERNE TECHNOLOGIEN GEWINNBRINGEND NUTZEN
EXTRA: Mit kostenlosem E-Book
HANSER

Pavlo Baron

@pavlobaron

it began in the 70s,
as I learned to tinker...

natural speaking wasn’t really mine, so some time later I discovered the most expressive language around

x86 assembly

At some point, I realized that there is C (guess it was about growing up or something).

So I started coding in C:

```
while (abc) {}

    asm {
        push    bx
        push    es
        mov     bx, 9
        mov     ax, 0
...
```

But puberty was over (I was like 18), and I had to do some higher level programming. So I discovered C++

The golden age of my C++ coding: OO thinking, abstractions, objects:

```
long SomeClass::getFoo()
{
    _asm
    {
        mov eax,dword ptr[ecx]
    }
}
...
```

But (enterprise) Java's advance was unstoppable. I had to jump in for some money.

I loved coding Java - finally patterns and real abstract thinking:

```
package com.stuff.jni;
public class Magic
{
  static {
    System.loadLibrary("strlen");
  }
  public static native int strlen(String s);
}
...
```

Out of nowhere, someone came along and asked me to write a tiny-footprint database for these weird Palm-PDAs.

And I found myself coding for memory maximum of 64kB yet again.

But (enterprise) Java was where the money lived (fun lived in nightly hacks).

oh, look, an interest calculator that's perfect for us! Let's buy it for our Java platform!!!

What, it's
written in C???
What, it can't do
parallel calculation?????

How parallelize this thing in our enterprise Java stack? Threads? EJB doesn't like threads. Machines through message-driven beans, JMS, whatever?..

Now, the bullshit part of the slides is over. Here is where Erlang kicks in.

I spent 3 months of my life researching how Erlang can help parallelize this weird piece of C code.

It could help.
It really could.

On one machine, on several machines.

But their ops guys said it's not Java. So they won't take it (as if they knew what real serious Java is).

And what happened? I just used every chance to play more with Erlang. This is what happened.

I decided to write a book only to learn this thing a lot deeper. Took 2.5 years of my life, worth every single minute.

I toured through Germany with an Erlang live hacking session for Java user groups. And yes, they (mostly) liked it.

My Java code started looking like this (and, man, why didn't it compile?):

```
public void doit(String s) ->
...
```

I don’t even want to touch a language not having list comprehensions and lambdas. Tooth grinding about those that don’t have pattern matching

Erlang made my code better in any language I use. I don't reassign variables. I work with deep copies. I sometimes use recursion instead of loops. I cut code into tiny, parallelizable functions

I judge every platform by the fact if I can spawn 2^27 actors on it and pass asynchronous messages between them

I am questioning every home-brewed non-Erlang distributed system or programming framework with a simple question: why reinvent the wheel?

For specific use cases such as messaging or distributed data management, an Erlang-based technology has automatic credit of trust when I look at it

I met some of the smartest people I ever met who are directly or indirectly related to Erlang

No, Erlang is not perfect.

But to be honest, nothing is.

Erlang is just the only considerable platform to solve some specific problems.

I have it in my toolbox. It’s a precise machine, very sensitive to doing things wrong. But I’m glad I learned to use it.

You should, too.

The End.

Thanks to the WWW for helping out with some code snippets. I lost my own ones during a house moving