Erlang on Microcontrollers

The path to a smaller BEAM

About me

Dmytro Lytovchenko

Employed by Erlang Solutions in Stockholm

20 years of experience with C/C++

7 years of Erlang

About me

/kvakvs/eia
/esl/erlang-handbook

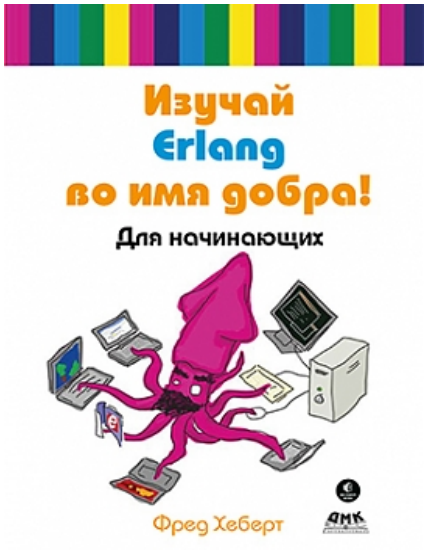
Russian translations on paper:

- Learn you some Erlang for Great Good
- Designing for Scalability

Russian translations:

- Erlang Handbook
- Erlang in Anger

- Created a website for VM internals:
  beam-wisdoms.clau.se



To run some Erlang

Where to begin?

To run some Erlang

Where to begin?

Code?

Loading, storing, interpreting

Data?

References and books?

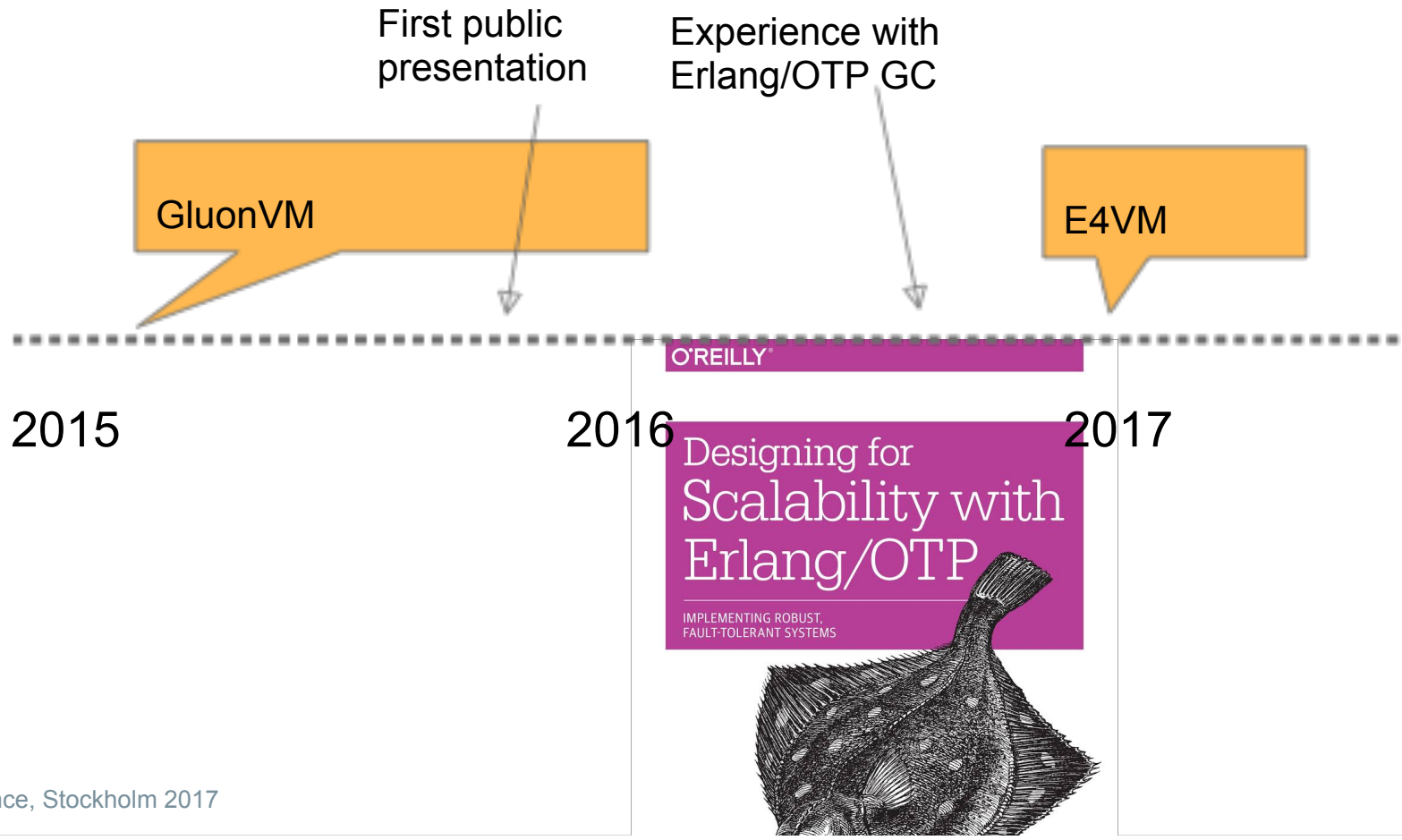
OTP emulator code

Ling emulator code

Other scarce documentation sources

The Beam Book (2017)
by Erik Stenman and the community

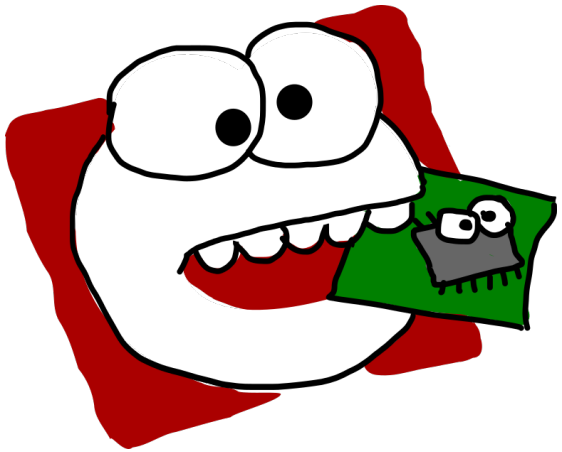
Timeline


First public presentation
Experience with Erlang/OTP GC
GluonVM
E4VM
2015
2016
2017
O'REILLY
Designing for Scalability with Erlang/OTP
IMPLEMENTING ROBUST, FAULT-TOLERANT SYSTEMS

GluonVM (2014 — 2015)

/kvakvs/gluonvm1

- A small project which had no one waiting for it
- A fairly successful attempt, though
- BEAM file compatibility
  - Could run some simpler code (lists, mochijson, …)

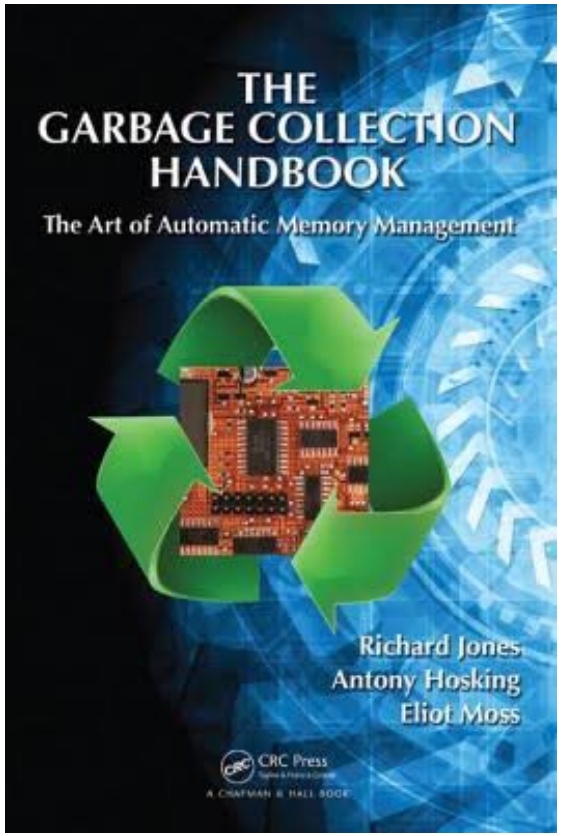


```
read fun table: init:-bs2as/1-fun-0-/1 offset=637
read fun table: init:-bs2ss/1-fun-0-/1 offset=635
read fun table: init:-boot/1-fun-0-/1 offset=633
read fun table: init:-notify/1-fun-0-/1 offset=631
read fun table: init:-do_boot/2-fun-0-/3 offset=623
read fun table: init:-par_load_modules/2-fun-0-/5 offset=612
read fun table: init:-shutdown_timer/1-fun-0-/1 offset=600
read fun table: init:-start_on_load_handler_process/0-fun-0-/0 offset=598
read fun table: init:-run_on_load_handlers/2-fun-0-/1 offset=596
Loading BEAM /usr/lib/erlang/lib/erts-7.3.1/ebin/erlang.beam
read fun table: erlang:-spawn_opt/5-fun-0-/2 offset=1097
Loading BEAM ../test/g_test2.beam
read fun table: g_test2:-test_inline_catch/0-fun-0-/0 offset=11
Process::jump_to_mfa -> 0x6c7530
---Scheduler::next() -> (Queue 3) #Pid<0x0>
[#Pid<0x0>;0x6c7530]: apply_mfargs_/0 args=()
apply_mfargs_: 'g_test2':'test' args=[]
---Scheduler::next() -> (Queue 3) #Pid<0x0>
[#Pid<0x0>;'g_test2':'test'/0]: allocate/2 args=(0;0)
[#Pid<0x0>;'g_test2':'test'/0]: call/2 args=(0;#CP<'g_test2':'test_inline_catch'/0>)
---Scheduler::next() -> (Queue 3) #Pid<0x0>
[#Pid<0x0>;'g_test2':'test_inline_catch'/0]: allocate_zero/2 args=(1;0)
[#Pid<0x0>;'g_test2':'test_inline_catch'/0]: make_fun2/1 args=(#Box<Tag=3;0x6f5d50>)
[#Pid<0x0>;'g_test2':'test_inline_catch'/0]: catch/2 args=(Y[0]=[];#CP<'g_test2':'test_inline_catch'/0>)
[#Pid<0x0>;'g_test2':'test_inline_catch'/0]: call_fun/1 args=(0)
---Scheduler::next() -> (Queue 3) #Pid<0x0>
[#Pid<0x0>;'g_test2':'-test_inline_catch/0-fun-0-'/0]: move/2 args=('test_exception';X[0]=#Fun<'g_test2'
-fun-0-'/0>)
[#Pid<0x0>;'g_test2':'-test_inline_catch/0-fun-0-'/0]: call_ext/2 args=(1;#Box<Tag=3;0x6f5b60>)
ctx.jump_ext -> 'erlang':'throw'/1
---Scheduler::next() -> (Queue 3) #Pid<0x0>
[#Pid<0x0>;'erlang':'throw'/1]: move/2 args=('undefined';X[0]='test_exception')
[#Pid<0x0>;'erlang':'throw'/1]: call_ext_only/2 args=(1;#Box<Tag=3;0x6e3560>)
ctx.jump_ext -> 'erlang':'nif_error'/1
Er
FAIL: belongs_ == ContextBelongsTo::VmLoop (/home/kv/proj/gluonvm1/emulator/src/process_ctx.h:82)
```

Gluon's End

/kvakvs/gluonvm1

- Unfinished
  - No GC
  - No binaries
  - No floats
  - No maps
  - A few BIFs
- Works with Erlang 17+
- Discontinued



Erl-Forth VM (2016 — 2017)

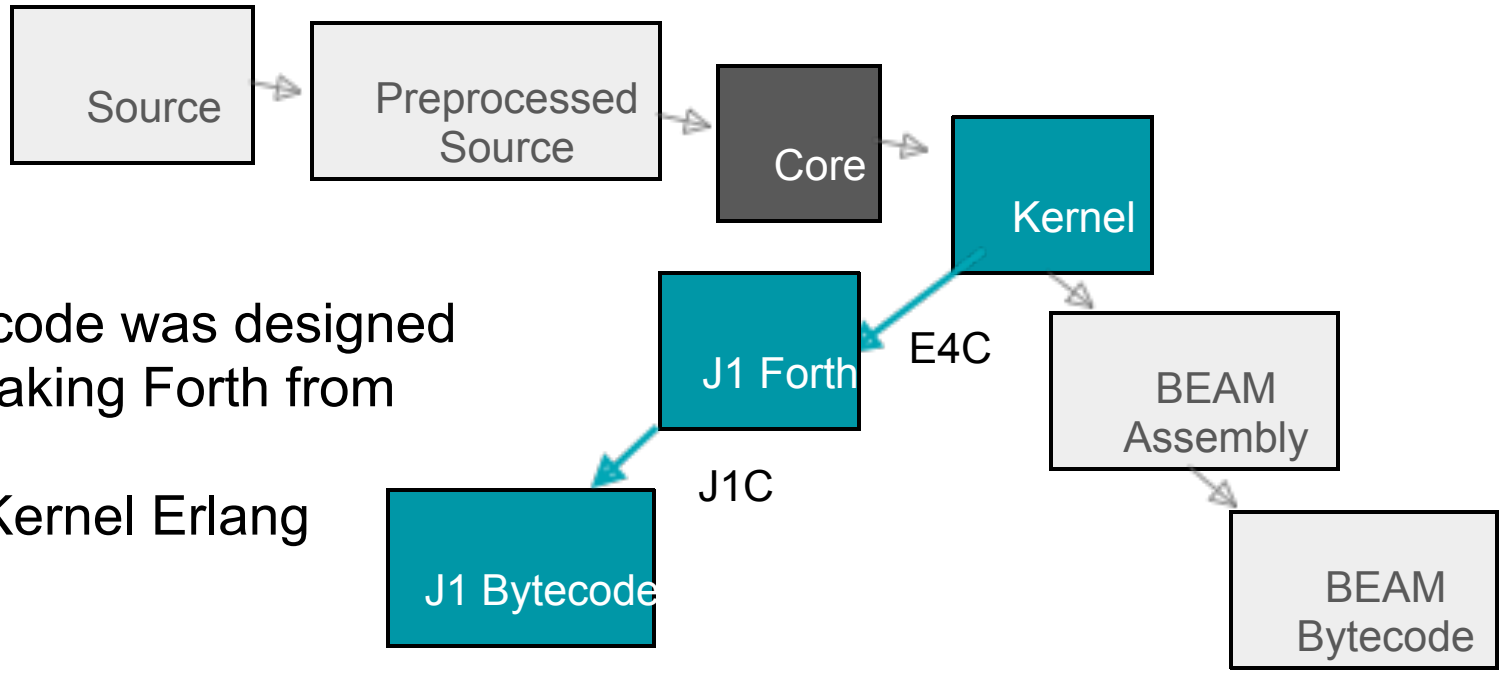
/kvakvs/E4VM
branch: ‘forth’

- Based on J1 Forth
- Simple, just 4 instructions
  - JUMP
  - JUMP_COND
  - CALL
  - ALU
- Has a VHDL implementation

```
hex : f* * e >>a ;
: sq over dup f* ;
4666 dup negate do 4000 dup 2* negate
do bl a + i j 1e 0 do sq sq 2dup + 10000 > if 2drop
3drop bl 0. leave then - j + -rot f* 2* k + loop 2drop
emit 268 +loop cr 5de +loop
```

Erl-Forth VM

- Experimental bytecode was designed
- A transpiler was making Forth from Core Erlang
- Later switched to Kernel Erlang



Erl-Forth VM

/kvakvs/E4VM
branch: ‘forth’

- Added more opcodes
- Packed them nicely into 16-bit words
- Invented some smart ways to encode longer opcode arguments

Erl-Forth VM

- And then I realized…
  - The opcodes form a new language of their own;
  - Not using J1
- It is not Forth anymore!

```
%%-define(J1INSTR_JUMP,      0).
%%-define(J1INSTR_JUMP_COND,  1).
-define(J1INSTR_CALL,      2).
-define(J1INSTR_ALU,       3).
-define(J1INSTR_GETELEMENT, 4).
-define(J1INSTR_LD,        5).
-define(J1INSTR_ST,        6).
-define(J1INSTR_ENTER,     7).
-define(J1INSTR_SMALL_POS,  9).
-define(J1INSTR_LD_SMALL,   10).
-define(J1INSTR_ST_SMALL,   11).
-define(J1LITERAL,         12).
-define(J1LIT_ATOM,        (?J1LITERAL+0)).
-define(J1LIT_LITERAL,     (?J1LITERAL+1)).
-define(J1LIT_INTEGER,     (?J1LITERAL+2)).
-define(J1INSTR_SINGLE_BYTE, 15).
-define(J1BYTE_INSTR_LEAVE,        16#F0).
-define(J1BYTE_INSTR_ERL_TAIL_CALL, 16#F1).
-define(J1BYTE_INSTR_ERL_CALL,     16#F2).
-define(J1BYTE_INSTR_NIL,          16#F3).
-define(J1BYTE_INSTR_JUMP,         16#F4).
-define(J1BYTE_INSTR_JUMP_COND,    16#F5).
-define(J1BYTE_INSTR_VARINT,       16#F6).
-define(J1BYTE_INSTR_VARINT_NEG,   16#F7).
```

LLVM Cross-compiler (2017)

/kvakvs/E4VM
branches: ‘llvm’
and ‘llvm-kernel’

- Another wild idea appears.
  - Erlang to LLVM IR
  - Also static single assignment (SSA)
    - A pleasant coincidence!
- Compiling from Erlang AST
  - Bad idea!

LLVM Cross-compiler

Afterthought

/kvakvs/E4VM
branches: ‘llvm’
and ‘llvm-kernel’

- Amazing amounts of work required
  - LLVM is smart but complicated
  - LLVM is huge
  - LLVM is slow to build and link
  - Must build LLVM source to make it work
- Hot code loading? Deal breaker!

Direct Threading

- The program consists of jump addresses
- VM loop is one large function with labels
- VM fetches another address and performs a goto

```
0000 OPCODE_A
0001 ArgA:8
0002 ArgA:16
0004 OPCODE_B
0005 ArgB:8
0006 OPCODE_C
```

```
0000 &opcodeA
0004 ArgA:32
0008 ArgA:32
000C &opcodeB
0010 ArgB:32
0014 &opcodeC
```

```
void vm_loop(bool init) {
  if (init) {
     goto do_init; }
opcodeA:
  do_something A…
  goto *(ip++);
opcodeB:
  do_something B…
  goto *(ip++);
do_init:
  labels[0] = &opcodeA;
  labels[1] = &opcodeB;
}
```

E4VM (2017)

/kvakvs/E4VM
branch: ‘master’

Best attempt so far

Custom BEAM-like file format

Possibly regular BEAM files are fine?

A direct-threaded emulator with reduced instruction set

E4VM Tradeoffs

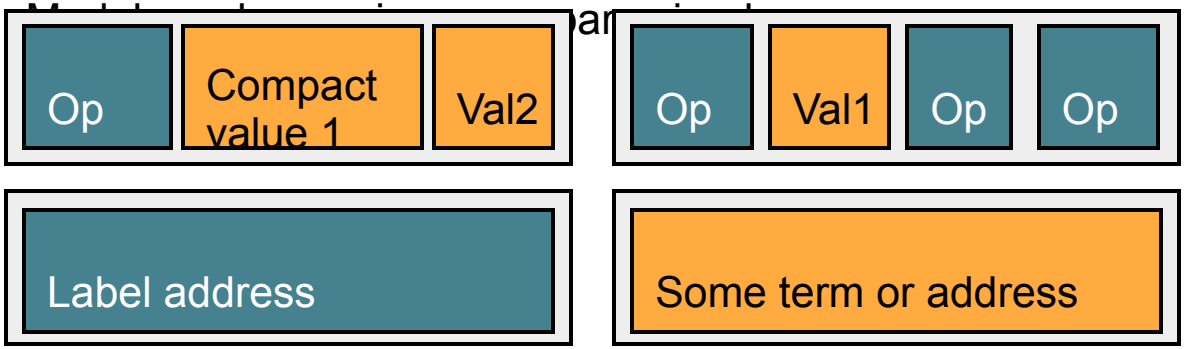
/kvakvs/E4VM
branch: ‘master’

Balance Performance vs. Code Size

Byte Opcodes + Switch vs. Threaded

Save memory

Run faster



A (32-bit) machine word

E4VM Library code

/kvakvs/E4VM
branch: ‘master’

Implementing all the major BIFs can be a lot of work

- Should be configurable per build

Erlang standard library is huge

- 47 Mb everything, 3 Mb stdlib, 400 kb ERTS
- Cut per-function
- Unload
- Compress

E4VM Current Status

/kvakvs/E4VM
branch: ‘master’

Transpiler	Data types, Heap	Major BIFs	**Ports**
Code Loader	Major Opcodes	**GC**	IO, Network
Modules	Minor Opcodes	**Running on ARM**	Hardware Drivers
VM Loop, Scheduling	**Distribution**	**Binary Opcodes**	Trimmed Erlang Libs

Challenges to overcome

/kvakvs/E4VM
branch: ‘master’

Knowing only approximate platform

Network stack? DNS? SSL?

Hardware drivers

Hard work (pays off)

/kvakvs/E4VM

Erlang Embedded?

Questions?