WELCOME TO PERL11

5 + 6 = 11

http://perl11.org/p2/

Perl

PIMC

Stavanger 2012 Moose + p5-mop Workshop

Preikestolen

Will Braswell
Austin 2012

Ingy döt net

PERL 11

5 + 6 = 11

perl11.org

Will Braswell, Ingy döt net, Reini Urban,
Flavio Glock, Audrey Tang, Wendy + Liz, ...

ofun.pm

Orlando 2013

PERL IS NOT DEAD, IT IS A DEAD END

Stevan Little
Orlando Perl Workshop 2013
stevan.little@iinteractive.com

Tuesday, August 13, 13

PERL8.ORG

pugs in scala - moe

INFORMATION
IN GIFT SHOP
Deer
$1,500

Tuesday, August 13, 13

Baby Boy

perl11

simple
features
performance
threads
sanity
future (?)

perl11

Pluggable PERL5 (+6)

1 **Parser** -> AST

2 **Compiler** AST -> ops

3 **VM** - Execute ops

PARSER

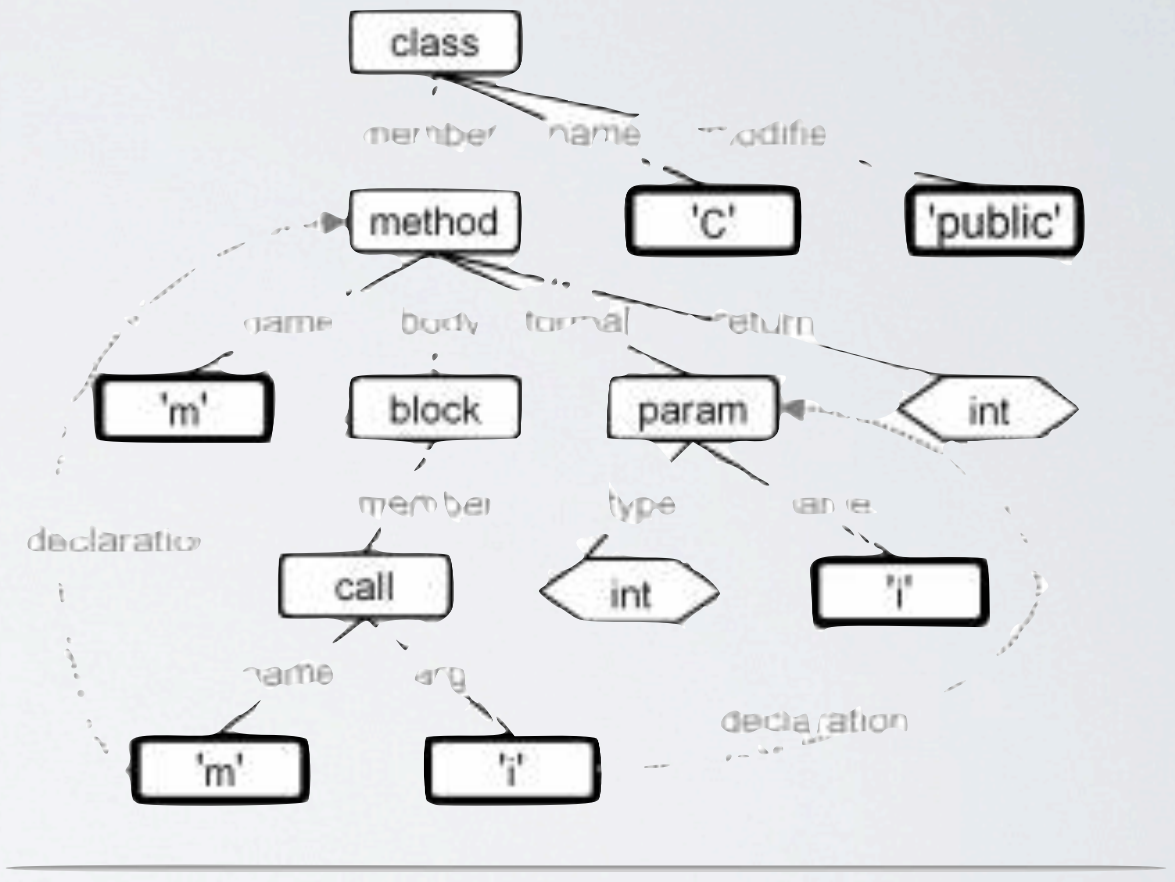
* YACC

* PEG / packrat

Marpa / ANTLR /

PGE, parsec / ...

* Handwritten



COMPILER

- **AST -> ops linearization**
- **Data Structures native vs library**
- **pluggable**
  bytecode vm, jit, c, native, jvm, js

Data structures: native vs library tradeoff

VM (S)

- Compile & execute compiled code
- Bytecode
- JIT
- call-out/in native libs
- Debugging/profiling support

DESIGN PRINCIPLES

Frequent case

- Math
- Conditionals
- Function calls
- Method dispatch
- Local variables
- Strings, build + compare
- Memory allocation

Not

- New methods
- Creation of classes
- Deep scoping situations
- Change inheritance tree
- Global variables
- Eval
- Code allocation

EFFICIENCY

- **Raw**
- **JVM / CLR / LLVM**
- **ML, LISP, LUA, Go, Smalltalk, V8**
- **Smaller or slower VMs**

LEARN FROM THE GOOD

1236 loc, 86K

- 30MB static libs for **LLVM** just for a **JIT**?
- 1GB of ugly junk for a JVM/.NET with huge startup overhead? Safe but not practical
- Java's main competitor: Lucent Inferno OS/Limbo/**Dis** VM
- All "good" VMs use their approach: GC, register based, three-address coding, tagged small data, word-size ops

JIT: 1236 loc, 115K

PARROT

- **Right, catchy ideas**

PARROT

- Pluggable syntax
- Pluggable types
- Pluggable ops

PARROT

- Pluggable syntax - parse to common AST - easy
- Pluggable types - like loadable C++ objects - framework
- Pluggable ops - same MOP framework (strict rules)

PARROT

- once it was fast
- then it was de-optimized by non-technicians
- threads the best, but still not used
- dead end. suicidal tendencies

POTION

- why the lucky stiff -
  famous ruby, eclectic, online suicide
- lua VM
- io / soda objmodel (smalltalk based)
- GC Cheney two-finger loop from QISH
- JIT self-written, very elegant

POTION

- common number interface

-

POTION

- common number interface

IV or NV, possibly auotmatic bignums

- common hash/array interface

tables and tuples, interchangable (i.e. casting)

-

- IV or NV, possibly automatic bignums with CPU-specific overflow checks
- tables and tuples interchangable (i.e. automatic casting)

POTION

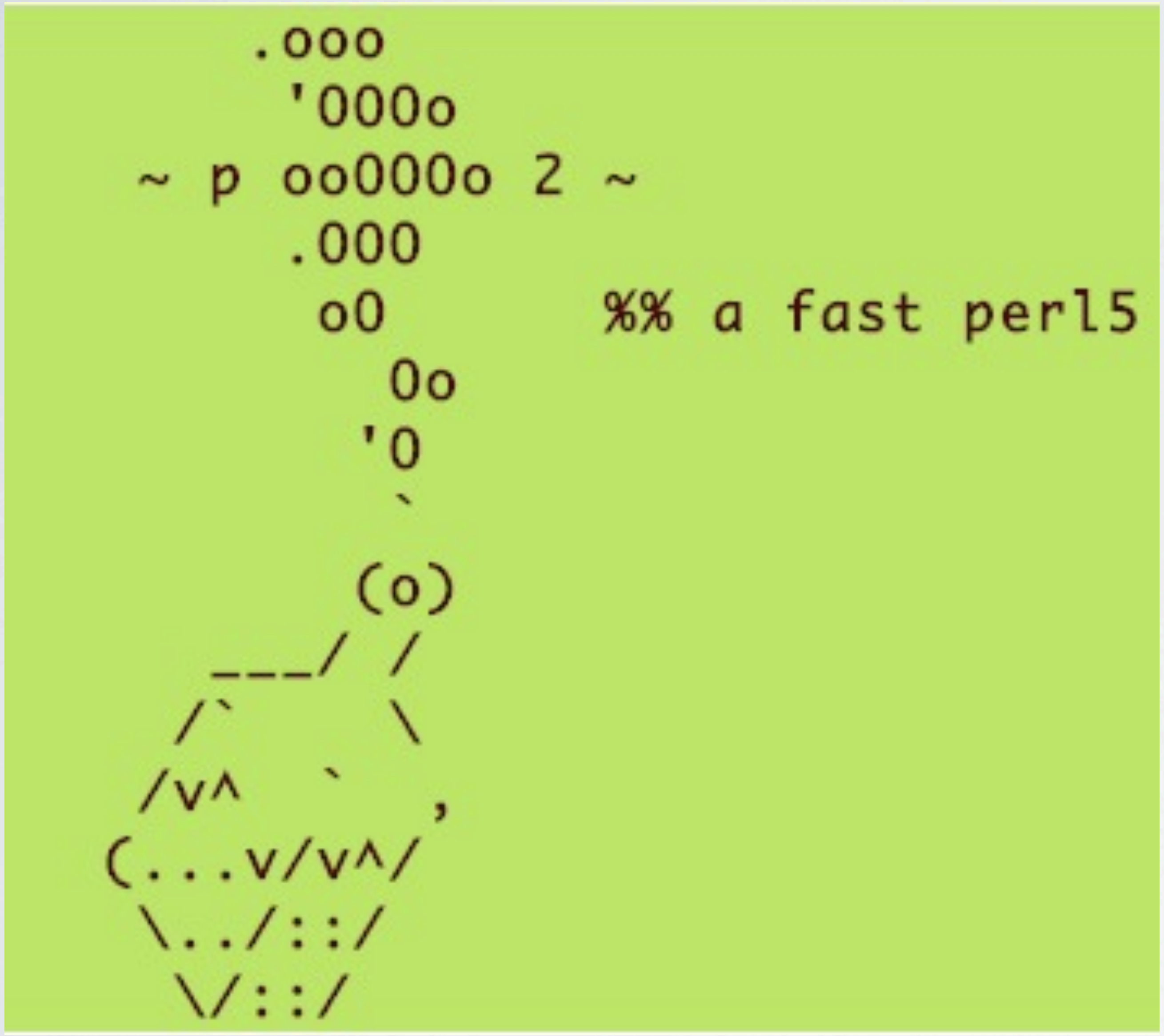
- common number interface
- common hash/array interface
- everything is an object, every object is a word
-

POTION

- common number interface
- common hash/array interface
- everything is an object, every object is a word
- every op is a word

potion

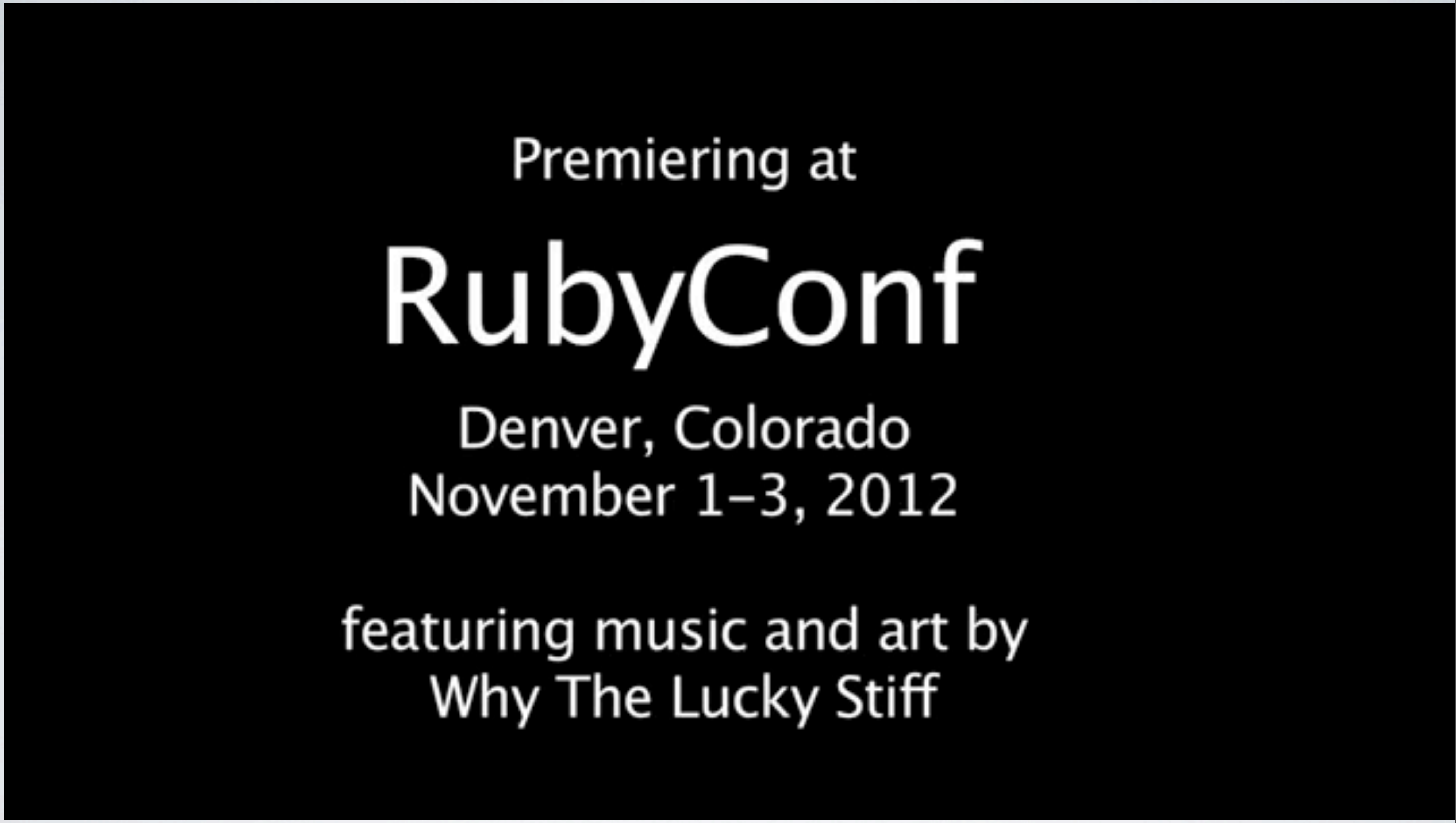
- **looks good**
- **smells good**
- **makes fun**



```
      .ooo
       '000o
   ~ p oo000o 2 ~
       .000
        o0       %% a fast perl5
          0o
         '0
          `

         (o)
     ___/ /
    /`    \
   /v^  `  ,
  (...v/v^/
   \../::/
    \/::/
```

play


Premiering at

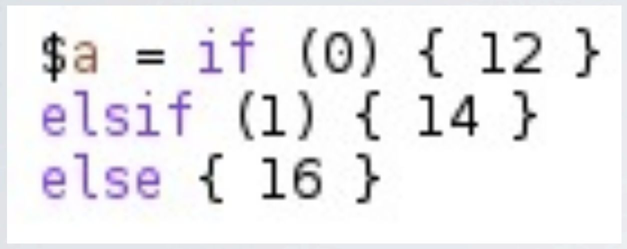
RubyConf

Denver, Colorado
November 1–3, 2012

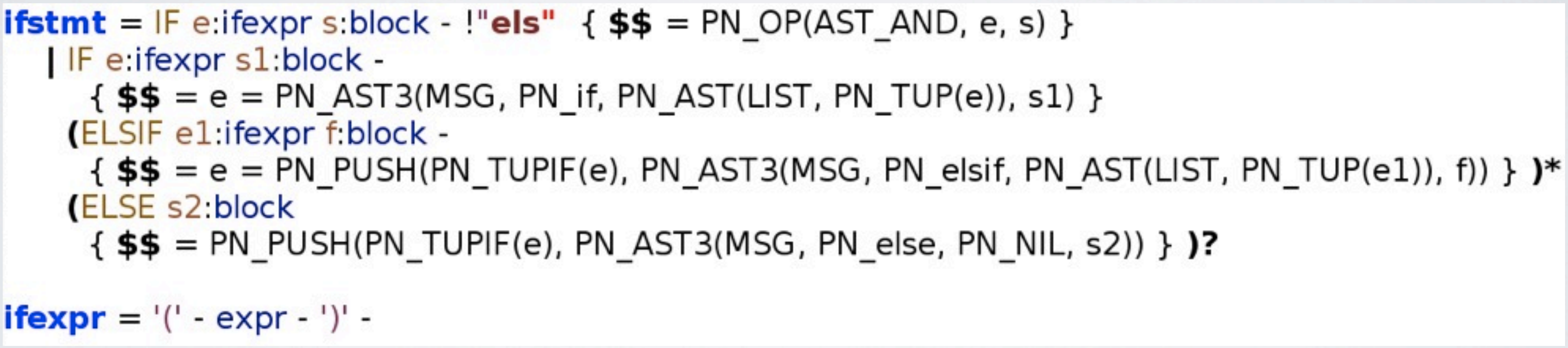
featuring music and art by
Why The Lucky Stiff

PARSER

- PEG (enhanced to greg)
- Syntax tree of PNSource objects (max 3 nodes)

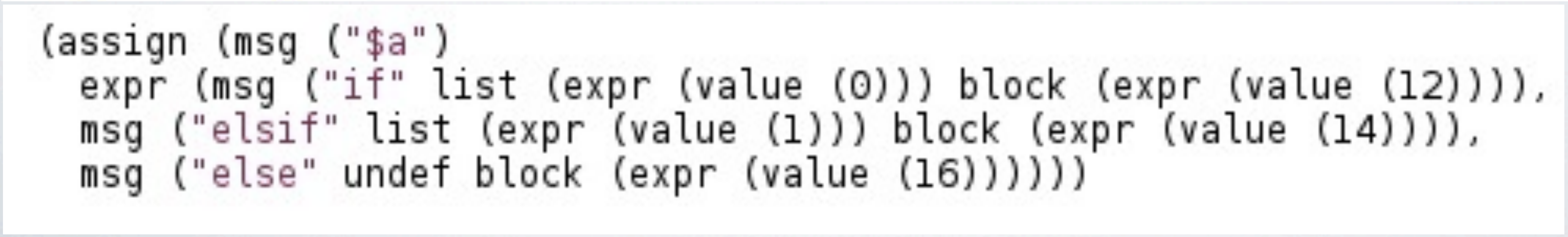


```
$a = if (0) { 12 }
elsif (1) { 14 }
else { 16 }
```



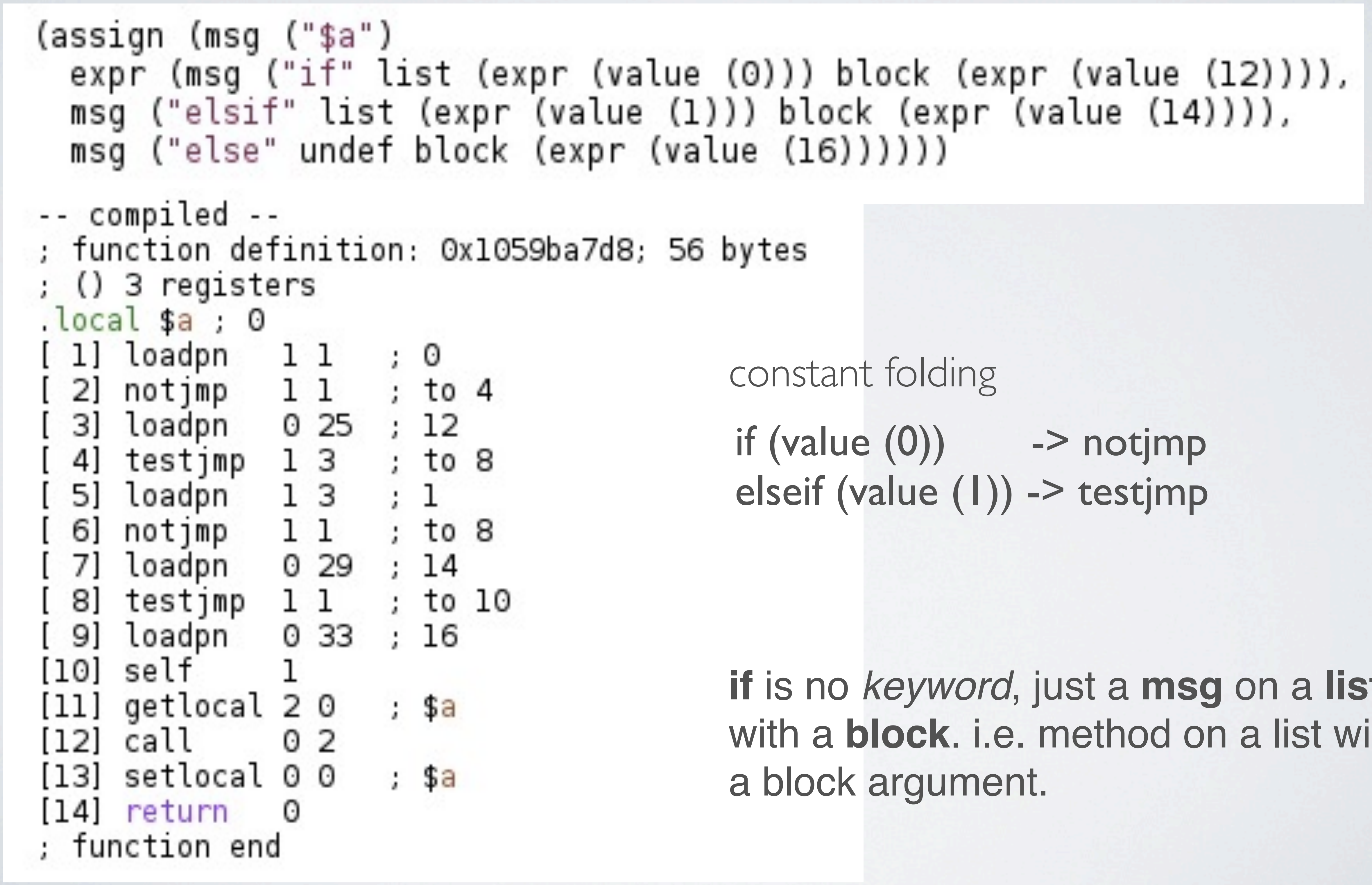
```
ifstmt = IF e:ifexpr s:block - !"els"  { $$ = PN_OP(AST_AND, e, s) }
   | IF e:ifexpr s1:block -
       { $$ = e = PN_AST3(MSG, PN_if, PN_AST(LIST, PN_TUP(e)), s1) }
     (ELSIF e1:ifexpr f:block -
       { $$ = e = PN_PUSH(PN_TUPIF(e), PN_AST3(MSG, PN_elsif, PN_AST(LIST, PN_TUP(e1)), f)) } )*
     (ELSE s2:block
       { $$ = PN_PUSH(PN_TUPIF(e), PN_AST3(MSG, PN_else, PN_NIL, s2)) } )?

ifexpr = '(' - expr - ')' -
```



```
(assign (msg ("$a")
  expr (msg ("if" list (expr (value (0))) block (expr (value (12)))),
  msg ("elsif" list (expr (value (1))) block (expr (value (14)))),
  msg ("else" undef block (expr (value (16))))))
```

COMPILER



```
(assign (msg ("$a")
  expr (msg ("if" list (expr (value (0))) block (expr (value (12)))),
  msg ("elsif" list (expr (value (1))) block (expr (value (14)))),
  msg ("else" undef block (expr (value (16))))))
```

```
-- compiled --
; function definition: 0x1059ba7d8; 56 bytes
; () 3 registers
.local $a ; 0
[ 1] loadpn   1 1   ; 0
[ 2] notjmp   1 1   ; to 4
[ 3] loadpn   0 25  ; 12
[ 4] testjmp  1 3   ; to 8
[ 5] loadpn   1 3   ; 1
[ 6] notjmp   1 1   ; to 8
[ 7] loadpn   0 29  ; 14
[ 8] testjmp  1 1   ; to 10
[ 9] loadpn   0 33  ; 16
[10] self     1
[11] getlocal 2 0   ; $a
[12] call     0 2
[13] setlocal 0 0   ; $a
[14] return   0
; function end
```

•

constant folding

if (value (0))   -> notjmp
elseif (value (1)) -> testjmp

if is no *keyword*, just a **msg** on a **list** with a **block**. i.e. method on a list with a block argument.

COMPILER

- Control constructs are not parser special.
  Expanded by the compiler, like a macro
- Macros are compile-time parser extensions, no parser keywords
- Most perl-level ops are just methods on objects
- Compiler is extendable.
  `--compile=c,opts` loads and calls a external compile-c library

VM

- Everything is an object, every object is a function (lambda)
- Every variable is a function, reacts to methods. (get, set, string, ...)
- Every block is a function, with lexical scoped variables and env
- Every call is a method call, even on nil or any

MOP

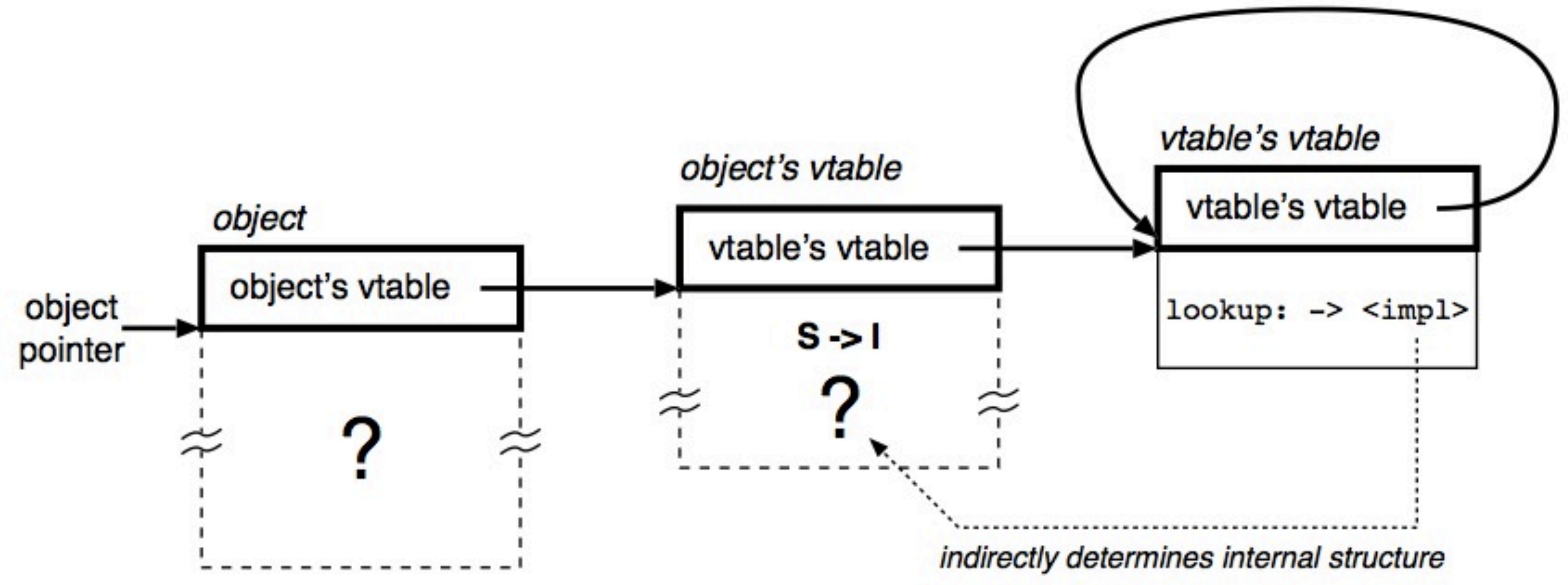
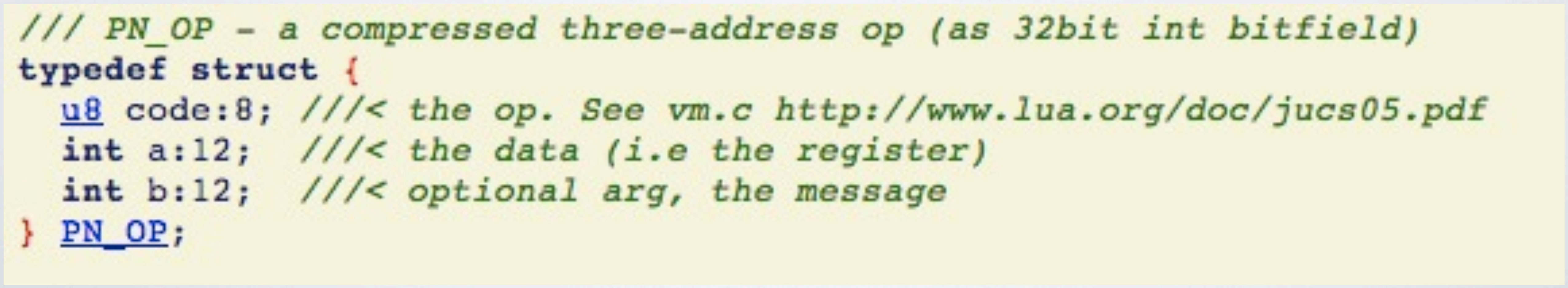


Figure 6. Everything is an object. Every object has a vtable that describes its behaviour. A method is looked up in a vtable by invoking its `lookup` method.

VM

- JIT default, for intel and powerpc. arm not yet.
- Bytecode for unsupported CPUs, and for debugging
- Very simple. From lua ~50 ops. Do complicated stuff in methods, such as array, hash, io, syscalls methods.
- Each op consists of 3 numbers `code,a,b` in one word



```c
/// PN_OP - a compressed three-address op (as 32bit int bitfield)
typedef struct {
  u8 code:8; ///< the op. See vm.c http://www.lua.org/doc/jucs05.pdf
  int a:12;  ///< the data (i.e the register)
  int b:12;  ///< optional arg, the message
} PN_OP;
```

DATA

- Primitive obj (in one word) vs extended objects (vt, uniq, size, data).
- INT, BOOL, NIL as primitives, everything else is an object.
- last bits 00 => foreign ptr or our obj (in our memory pages)
- last bits 10 => bool (true or false)
- last bit 1 => int (shifted by 1)
- Note: Different to dart, which has native int and shifts ptrs.

CALLING CONVENTION

only native, no stddecl, or foreign decl yet

- Native C cdecl (32bit) and fastcall (64bit) layout
- Fast, and easy to interface, call-out and call-in.
  Fast function calls, no function call overhead (as in LISP)
- OO: Every potion method prepends 2 args.
  interpreter, environment (a closure), self, optional args

only native, no stddecl, or foreign decl yet

GC - CHENEY LOOP

- walks the stack, not the heap, use volatile
- copying (i.e. compacting), thread-friendly
- gc friendly data, chain of fwd ptr,
  also for thread-shared data - parrot "proxy"
- i.e. essentially a tri-color algo

- just not stop-the-world and mark&sweep, uses no private stack. data knows about threads, proxies

Tuesday, August 13, 13

- just not stop-the-world and mark&sweep, uses no private stack.
  data knows about threads, proxies

GC

- 3 memory areas:
    - protected segment (boot + core)
    - birth segment (fast generation, minor collections)
    - main segment (major collections)
- old segment

- old: swapped out with live segments during GC, mprotected

DESIGN DECISIONS

- support 90% but do not sacrifice for the rest
- gmake and c99 gcc/clang are everywhere
- no MSVC, bsd make, no strict C++-only compilers
- early testing with cross-compiling and threads

not afterwards

FUNCTIONAL

- use destruction with care.
  I use LISP names: nreverse, delete

- return copies, do not change arguments
- Str immutable, Buf bytebuffers for io
- no functions. pass a message to everything
- no statements. everything is an expression

- returns something and can be stacked

- use destruction with care. I use LISP names: nreverse, nsort, delete

- returns something and can be stacked

MACROS

- With non-lisp languages
- **parser macros**
  in parse context, use existing parser syntax. <rule> ...
- **compiler macros**
  like a function call. evaluate not all args, only some.
  body unquoting with `expr`

you can do everything: control constructs, like when, foreach, unless
start getting messy, where to be added into the parser state machine, fragile (messes with existing parser rules), and look bad because of the <rule> syntax.

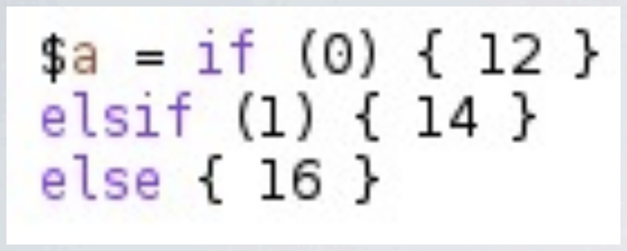
limited to calls. but if your parser does nothing else then calls (like lisp does), its the perfect point to add it.
do not change the parser, just hook into the compiler.

Tuesday, August 13, 13

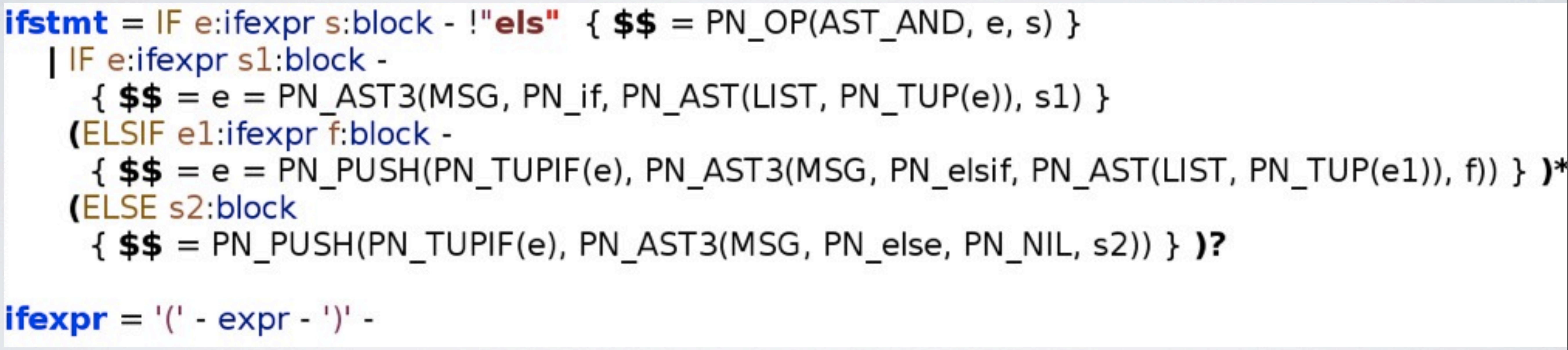
you can do everything: control constructs, like while, foreach, unless.
starts getting messy. where to be added into the parser state machine, fragile (messes with existing parser rules), and look bad because of the <rule> syntax. needs parser support, not pre-compiled.

limited to calls. but if your parser does nothing else then calls (like lisp does), its the perfect point to add it.
do not change the parser, just hook into the compiler.

MACROS

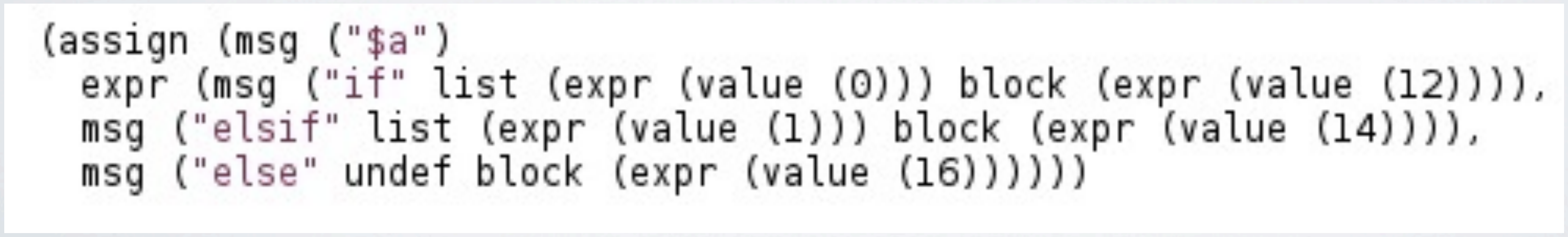


```
$a = if (0) { 12 }
elsif (1) { 14 }
else { 16 }
```



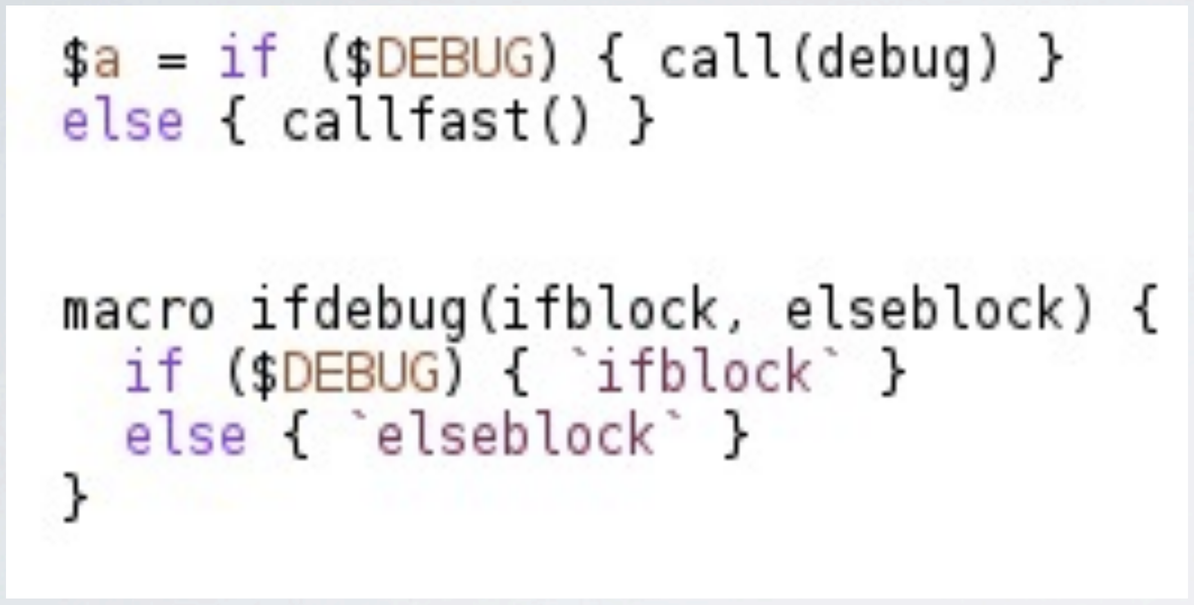
```
ifstmt = IF e:ifexpr s:block - !"els"  { $$ = PN_OP(AST_AND, e, s) }
   | IF e:ifexpr s1:block -
      { $$ = e = PN_AST3(MSG, PN_if, PN_AST(LIST, PN_TUP(e)), s1) }
     (ELSIF e1:ifexpr f:block -
      { $$ = e = PN_PUSH(PN_TUPIF(e), PN_AST3(MSG, PN_elsif, PN_AST(LIST, PN_TUP(e1)), f)) } )*
     (ELSE s2:block
      { $$ = PN_PUSH(PN_TUPIF(e), PN_AST3(MSG, PN_else, PN_NIL, s2)) } )?

ifexpr = '(' - expr - ')' -
```



```
(assign (msg ("$a")
  expr (msg ("if" list (expr (value (0))) block (expr (value (12)))),
  msg ("elsif" list (expr (value (1))) block (expr (value (14)))),
  msg ("else" undef block (expr (value (16))))))
```

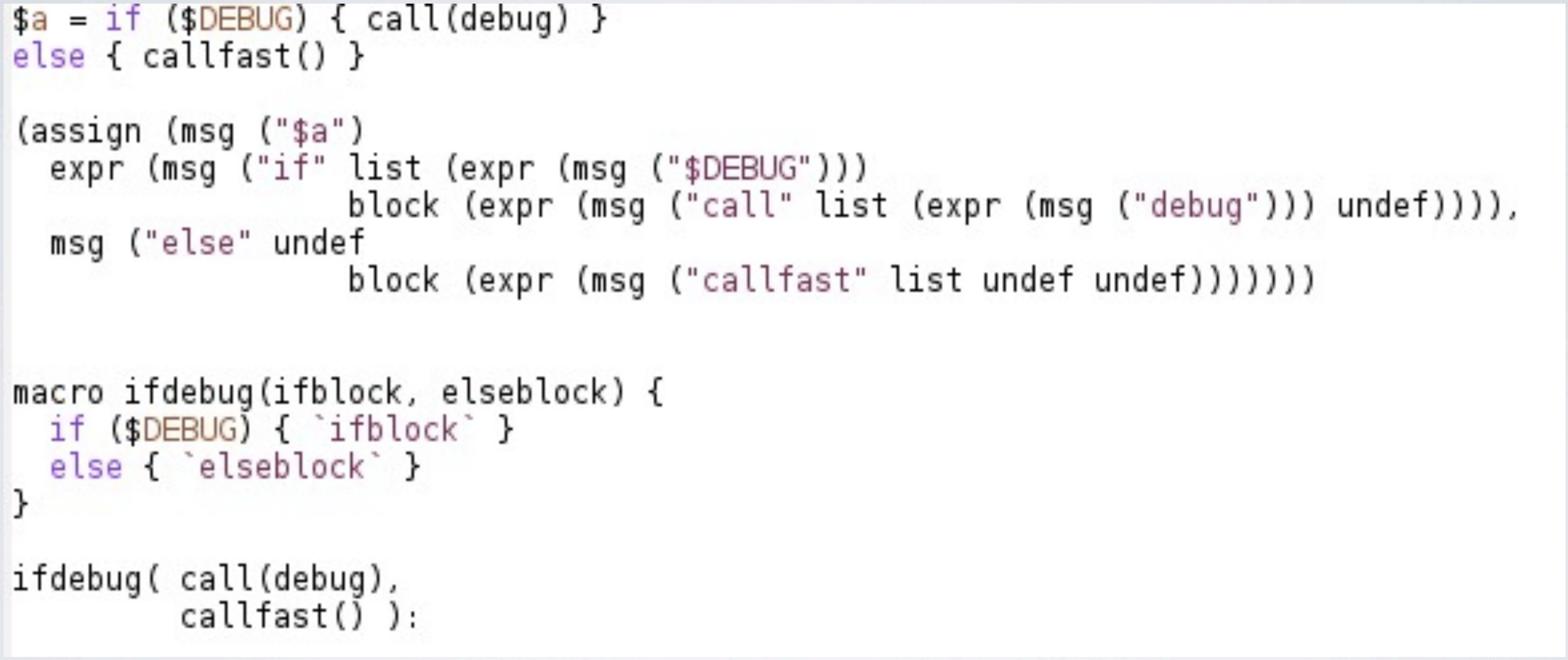
MACROS



```
$a = if ($DEBUG) { call(debug) }
else { callfast() }


macro ifdebug(ifblock, elseblock) {
  if ($DEBUG) { `ifblock` }
  else { `elseblock` }
}
```

MACROS



```
$a = if ($DEBUG) { call(debug) }
else { callfast() }

(assign (msg ("$a")
  expr (msg ("if" list (expr (msg ("$DEBUG")))
                  block (expr (msg ("call" list (expr (msg ("debug"))) undef)))),
  msg ("else" undef
                  block (expr (msg ("callfast" list undef undef)))))))


macro ifdebug(ifblock, elseblock) {
  if ($DEBUG) { `ifblock` }
  else { `elseblock` }
}

ifdebug( call(debug),
         callfast() ):
```

STATUS

- potion maintenance moved to perl11 org
- greg upstream commits: better error handling, diagnostics
- release potion 0.1 soon (await move, docs and one VM bug)
- more potion examples and features: ffi, threads, UI bindings, shootout samples

TODO

- GOAL
- Parser
- Compiler
- VM
- Libs

TODO

- GOAL: run 50% of p5 by Summer 2013, 90% by 2014.
- Parser
- Compiler
- VM
- Libs

TODO

- GOAL: run 50% of p5 by Summer 2013, 90% by 2014.
- Parser: 30%, work on expr and calls. Can’t call functions yet, no p5-weirdness (proto, dynamic namespaces)
- Compiler
- VM
- Libs

TODO

- GOAL: run 50% of p5 by Summer 2013, 90% by 2014.
- Parser: 30%, work on expr and calls. Can’t call functions yet, no p5-weirdness (proto, dynamic namespaces)
- Compiler: only to bytecode serialization, vm and jit. not to C or native yet. No macros.
- VM
- Libs

TODO

- GOAL: run 50% of p5 by Summer 2013, 90% by 2014.
- Parser: 30%, work on expr and calls. Can't call functions yet, no p5-weirdness (proto, dynamic namespaces)
- Compiler: only to bytecode serialization, vm and jit. not to C or native yet. No macros.
- VM: arm jit, threads, callcc stability, ffi.
- Libs

TODO

- GOAL: run 50% of p5 by Summer 2013, 90% by 2014.
- Parser: 30%, work on expr and calls. Can’t call functions yet, no p5-weirdness (proto, dynamic namespaces)
- Compiler: only to bytecode serialization, vm and jit. not to C or native yet. No macros yet.
- VM: arm jit, threads, stabilize callcc, ffi.
- Libs: aio ✓, buffile ✓, sprintf (20%), pcre (10%), bignum (20%) bindings, p5 compat.