Ruby Runtime

by @jimmynguyc

Hi

- Jimmy Ngu (@jimmynguyc)
- Engineering Team Lead @ RapidRiver Software
- KL Ruby Brigade, RubyConf MY (not anymore)
- We are hiring !! :D

What I'll Cover

- How Ruby runs your code
- Different Ruby Implementations
- MJIT

How Ruby runs Code

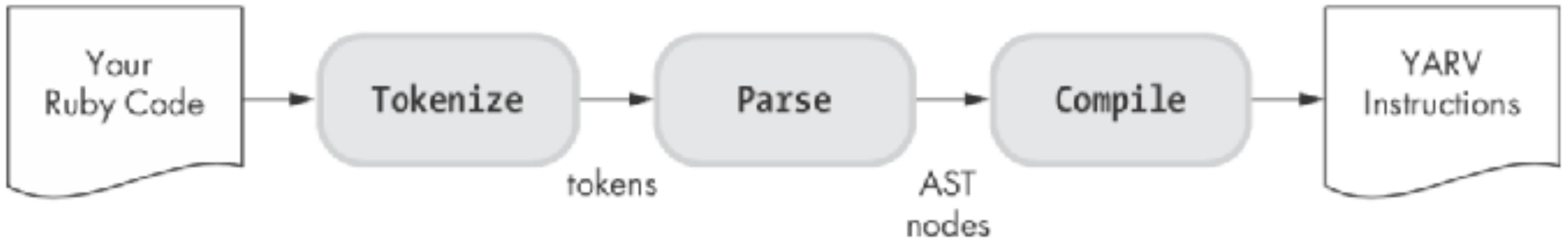


Figure 1-1: Your code's journey through Ruby

Tokenizer

- Breaks up sentences / code into pieces
- Words, keywords, phrases, symbols (tokens)
- Words used in Ruby
- parse.y (parser_yylex function)
  https://github.com/ruby/ruby/blob/master/parse.y#L8625

Ripper

```
require 'ripper'
require 'pp'
code = <<STR
  10.times do |n|
    puts n
  end
STR
pp Ripper.lex(code)
```

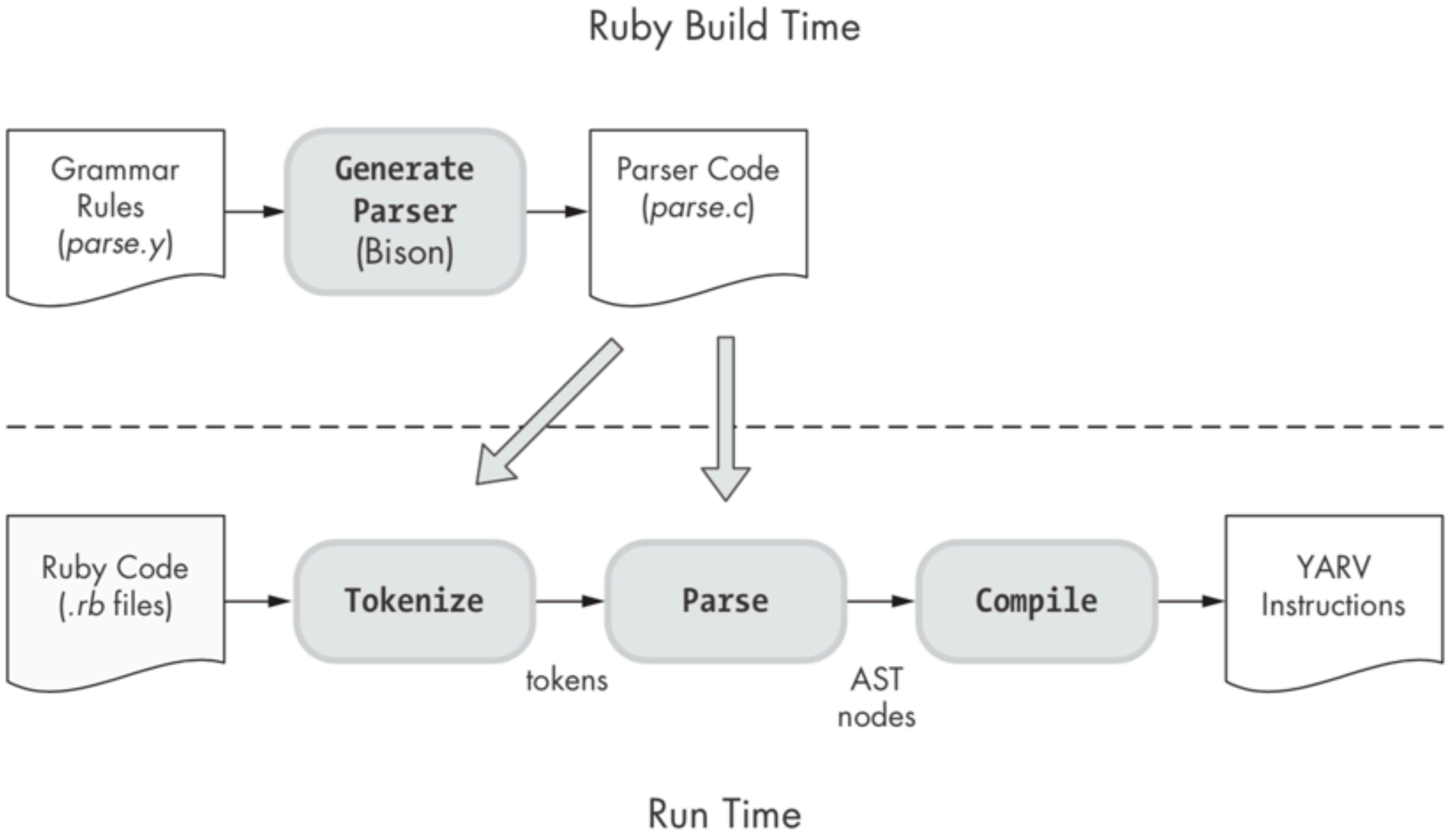
```
[[[1, 0], :on_sp, "  ", EXPR_BEG],
 [[1, 2], :on_int, "10", EXPR_END],
 [[1, 4], :on_period, ".", EXPR_DOT],
 [[1, 5], :on_ident, "times", EXPR_ARG],
 [[1, 10], :on_sp, " ", EXPR_ARG],
 [[1, 11], :on_kw, "do", EXPR_BEG],
 [[1, 13], :on_sp, " ", EXPR_BEG],
 [[1, 14], :on_op, "|", EXPR_BEG|EXPR_LABEL],
 [[1, 15], :on_ident, "n", EXPR_ARG],
 [[1, 16], :on_op, "|", EXPR_BEG|EXPR_LABEL],
 [[1, 17], :on_ignored_nl, “\n", EXPR_BEG|EXPR_LABEL],
 [[2, 0], :on_sp, "    ", EXPR_BEG|EXPR_LABEL],
 [[2, 4], :on_ident, "puts", EXPR_CMDARG],
 [[2, 8], :on_sp, " ", EXPR_CMDARG],
 [[2, 9], :on_ident, "n", EXPR_END|EXPR_LABEL],
 [[2, 10], :on_nl, "\n", EXPR_BEG],
 [[3, 0], :on_sp, "  ", EXPR_BEG],
 [[3, 2], :on_kw, "end", EXPR_END],
 [[3, 5], :on_nl, "\n", EXPR_BEG]]
```

Ripper

- Ripper symbol (e.g. :on_ident) is not actual Ruby token symbols (e.g. tIDENTIFIER)
- Gives a sense of what tokens are found

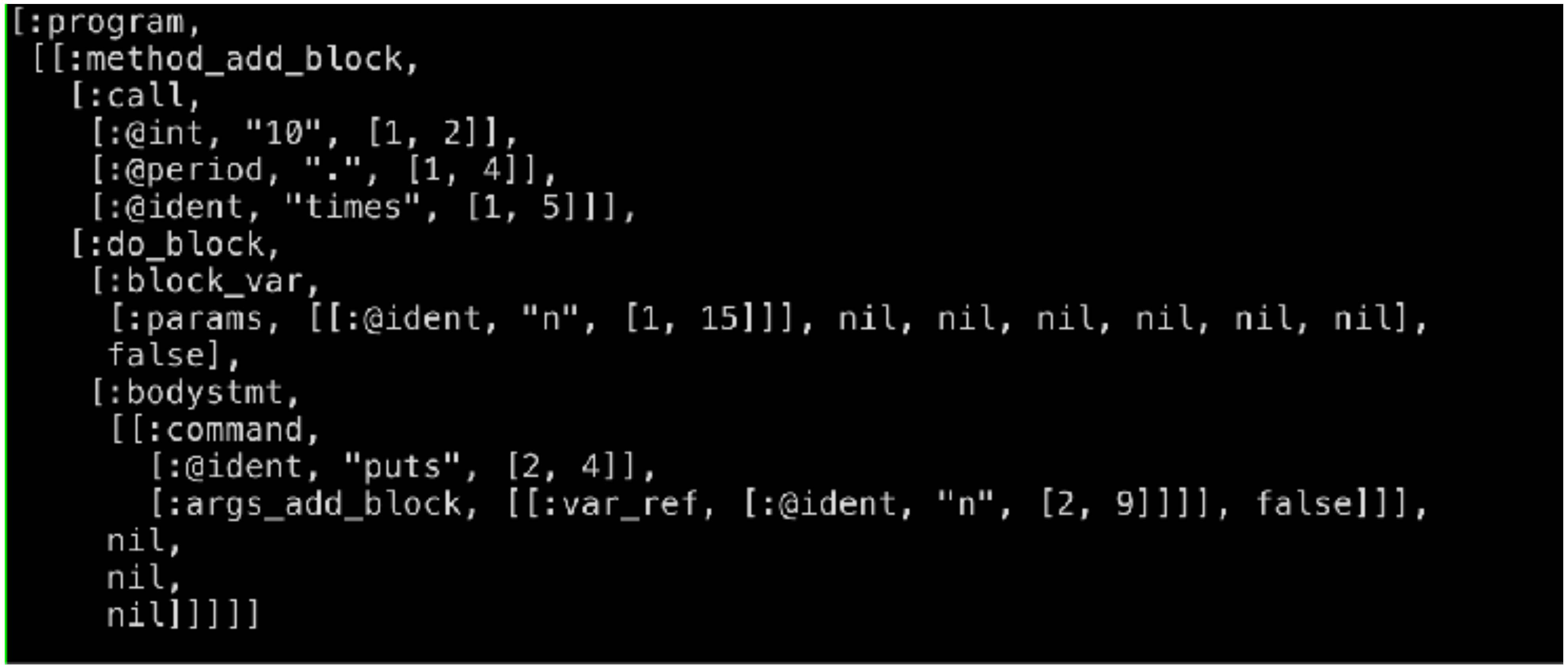
Parser

- Group token into sentences / phrases that makes sense to Ruby
- Ruby uses a parser generator called Bison (newer version of Yacc - Yet Another Compiler Compiler)
- parse.y -> parse.c
- Compiled during Ruby build time


Ruby Build Time
Grammar Rules (parse.y)
Generate Parser (Bison)
Parser Code (parse.c)
Ruby Code (.rb files)
Tokenize
tokens
Parse
AST nodes
Compile
YARV Instructions
Run Time

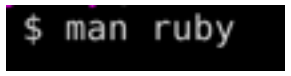
AST

```
require 'ripper'
require 'pp'
code = <<STR
  10.times do |n|
    puts n
  end
STR
pp Ripper.sexp(code)
```

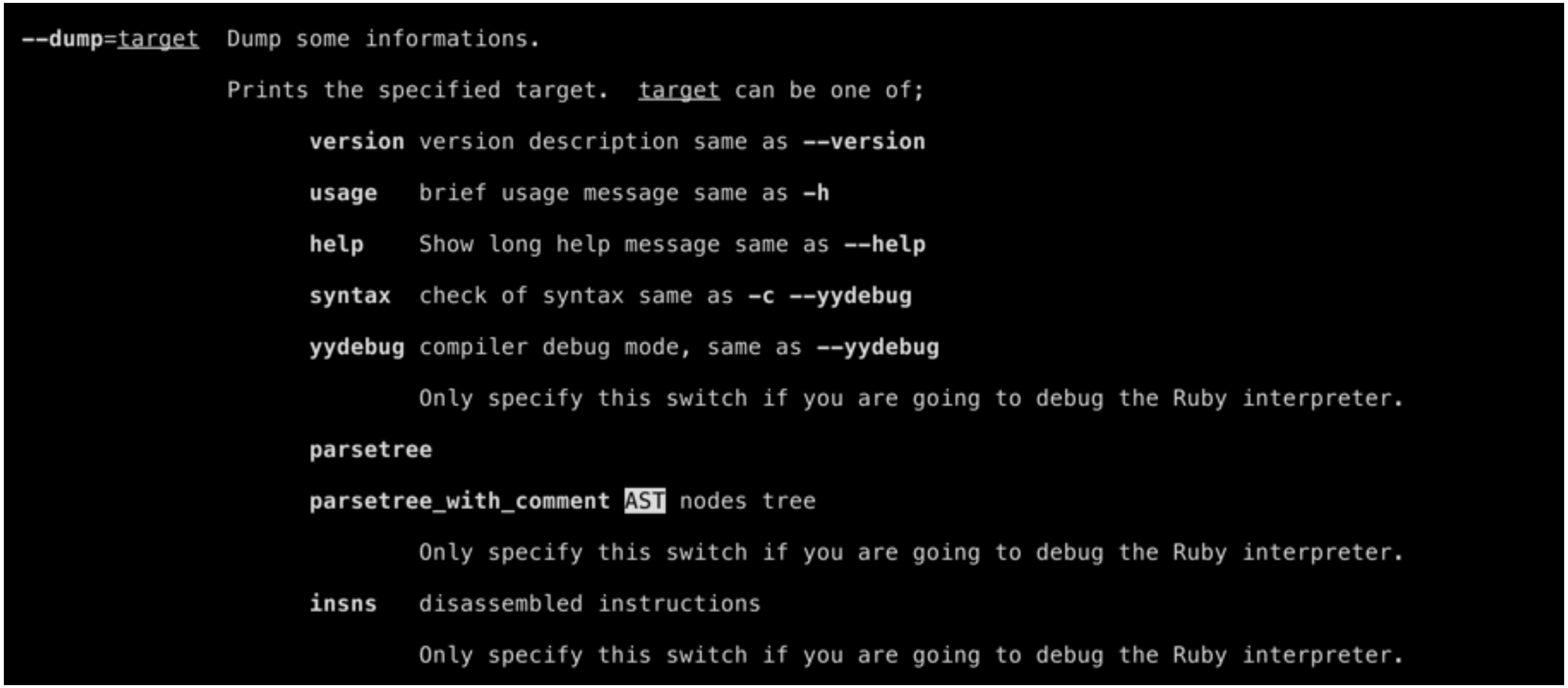


```
[:program,
 [[:method_add_block,
   [:call,
    [:@int, "10", [1, 2]],
    [:@period, ".", [1, 4]],
    [:@ident, "times", [1, 5]]],
   [:do_block,
    [:block_var,
     [:params, [[:@ident, "n", [1, 15]]], nil, nil, nil, nil, nil, nil],
     false],
    [:bodystmt,
     [[:command,
       [:@ident, "puts", [2, 4]],
       [:args_add_block, [[:var_ref, [:@ident, "n", [2, 9]]]], false]]],
     nil,
     nil,
     nil]]]]]
```

ruby --dump



```
$ man ruby
```



```
--dump=target  Dump some informations.

               Prints the specified target.  target can be one of;

                     version version description same as --version

                     usage   brief usage message same as -h

                     help    Show long help message same as --help

                     syntax  check of syntax same as -c --yydebug

                     yydebug compiler debug mode, same as --yydebug

                             Only specify this switch if you are going to debug the Ruby interpreter.

                     parsetree

                     parsetree_with_comment AST nodes tree

                             Only specify this switch if you are going to debug the Ruby interpreter.

                     insns   disassembled instructions

                             Only specify this switch if you are going to debug the Ruby interpreter.
```

Compiler

- Translate code -> Another code

- E.g.

  Compile C -> Machine language
  Compile Java -> Java bytecode

< 1.9

- No compiler
- Executes AST tree directly
- Interpreted into C > Machine code

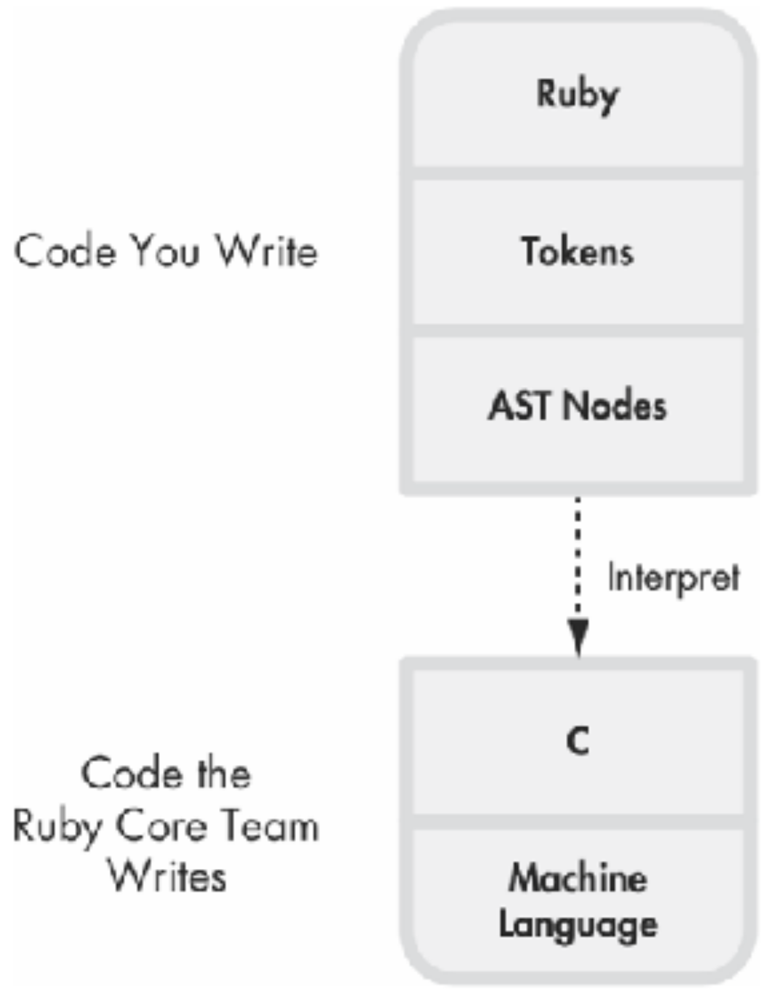


Figure 2-1: In Ruby 1.8, your code is converted into AST nodes and then interpreted.

>= 1.9

- YARV (Yet Another Ruby Virtual Machine)
- Compiles AST into bytecode (YARV Instructions)
- Then Intepreted into C > Machine Code

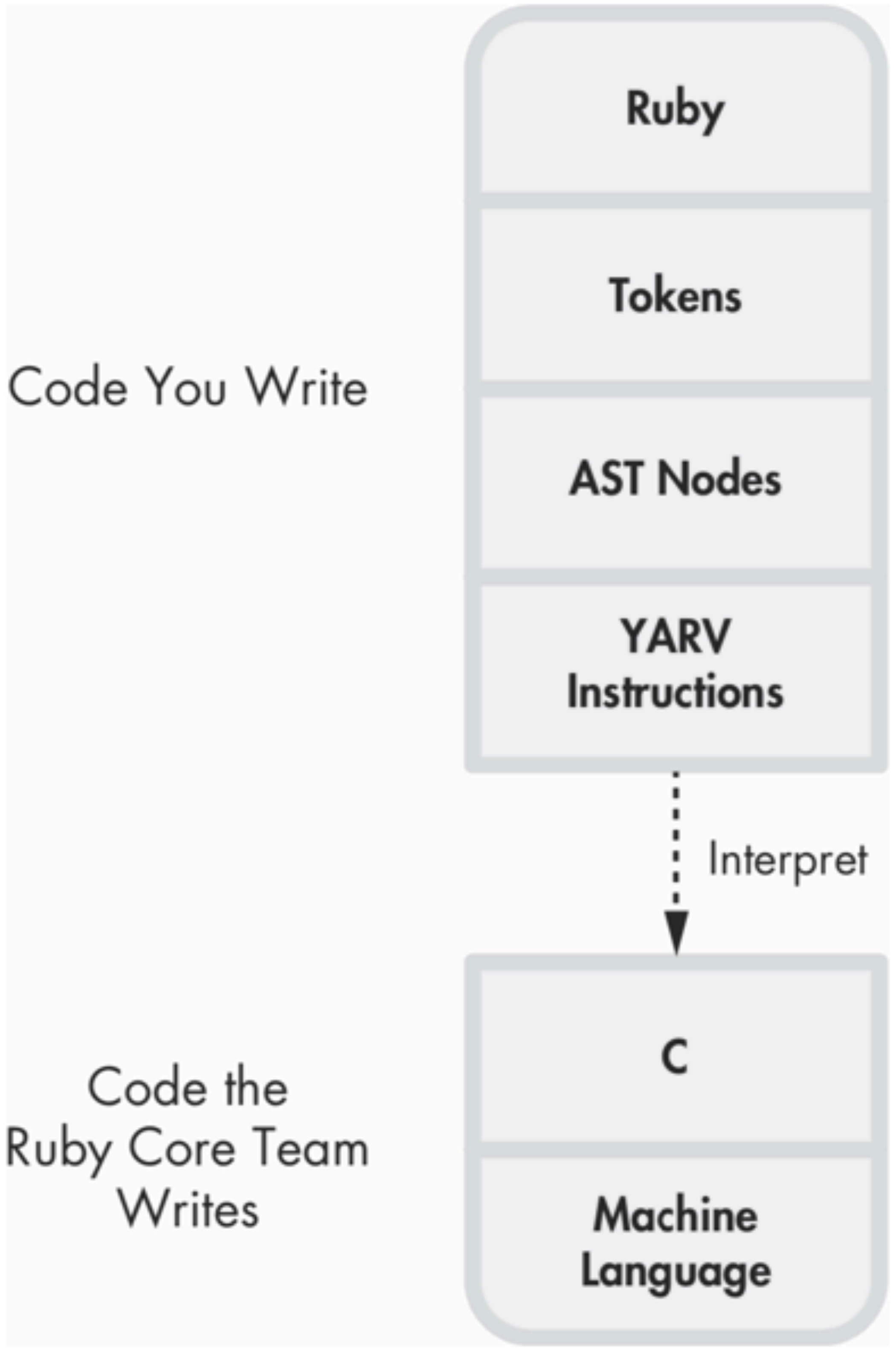


Figure 2-2: Ruby 1.9 and 2.0 compile the AST nodes into YARV instructions before interpreting them.

YARV

Reason: SPEED !!!

YARV

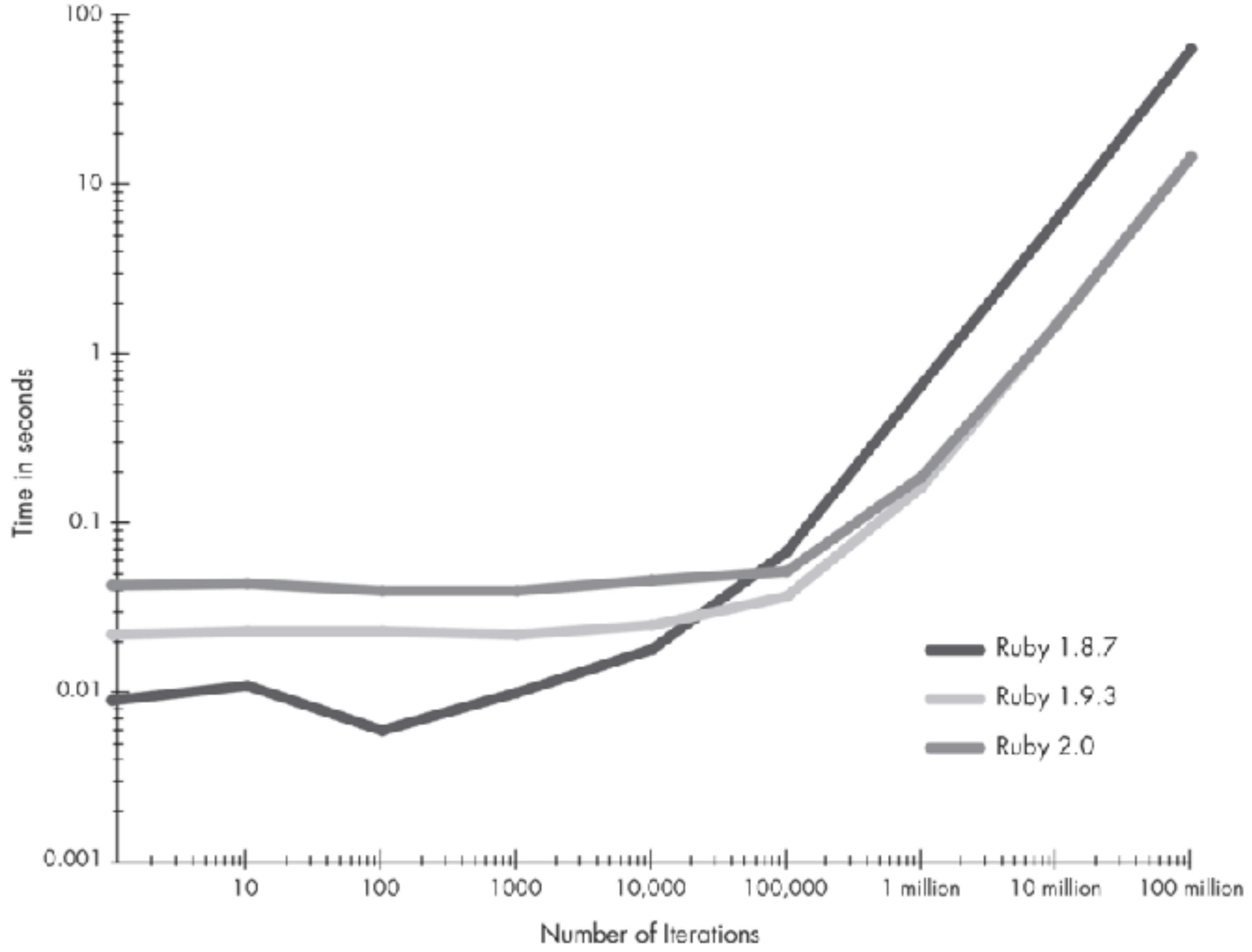
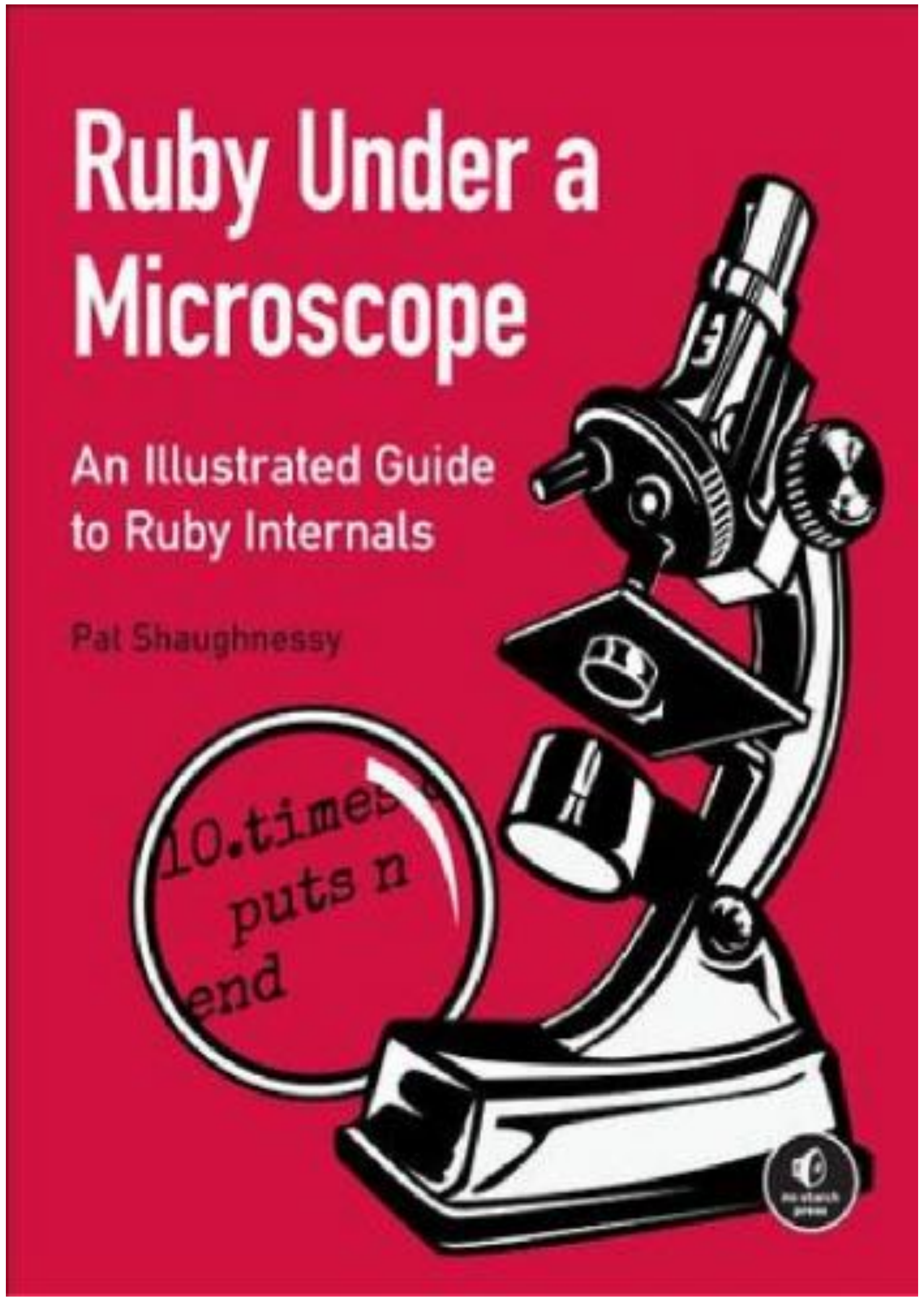


Figure 3-14: Performance of Ruby 1.8.7 vs. Ruby 1.9.3 and Ruby 2.0; time (in seconds) vs. number of iterations on a logarithmic scale

Reference

- Ruby Under a Microscope (Chapter 1-4)



Ruby Implementations

- Standard implementation - C (a.k.a. CRuby)
- Switch out parser & compiler
- Run Ruby on all environments !!!

Ruby Implementations

- JRuby - Java VM
- Rubinius - Rubinius VM (C++) / LLVM
- Truffle Ruby - Truffle + GraalVM
- MacRuby - RubyMotion VM (Objective C)
- Opal - Ruby to JS transpiler (Browser JS Runtime / V8)

MJIT

- MJIT - Method JIT
- Ruby’s new compiler

What’s wrong with YARV?

- Stack-oriented vs. Register-Oriented architecture
- CPU Registers - fastest storage locations
- Simpler instructions
- Optimizing compilers e.g. GCC

MJIT

- JIT - Just-in-time Compilation
- Vladamir Makarov from RedHat
- New RTL (Register Transfer Language) generator - generates register-oriented IRs (bytecode-like instructions), replacing YARV
- Uses GCC
- Method JIT (vs. Tracing JIT), which is better
- Ruby’s own JIT

However

- Far from complete
- We are all impatient / immediate-gratifiers

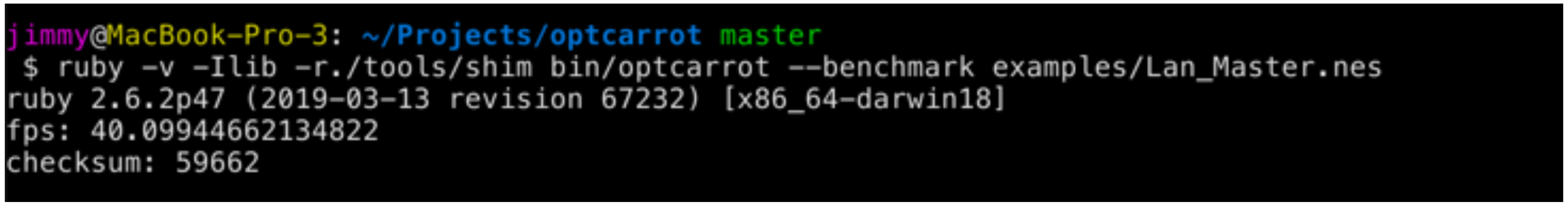
Enter YARV-MJIT

- Takashi Kokubun
- Uses existing YARV instructions
- Ports over MJIT implementations
- Available in Ruby 2.6 using *--jit*

Demo

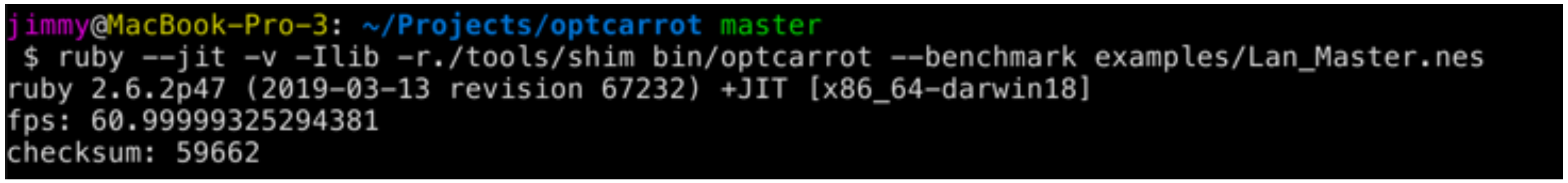
optcarrot bench

Without MJIT



```
jimmy@MacBook-Pro-3: ~/Projects/optcarrot master
 $ ruby -v -Ilib -r./tools/shim bin/optcarrot --benchmark examples/Lan_Master.nes
ruby 2.6.2p47 (2019-03-13 revision 67232) [x86_64-darwin18]
fps: 40.09944662134822
checksum: 59662
```

With MJIT



```
jimmy@MacBook-Pro-3: ~/Projects/optcarrot master
 $ ruby --jit -v -Ilib -r./tools/shim bin/optcarrot --benchmark examples/Lan_Master.nes
ruby 2.6.2p47 (2019-03-13 revision 67232) +JIT [x86_64-darwin18]
fps: 60.99999325294381
checksum: 59662
```

Performed on MBP 2.9 GHz Intel Core i7, 16 GB 2133 MHz LPDDR3
(with who-knows-what running on background)

Conclusion

- Ruby is a long running project, but not considered a high-priority language (compared to JS, Python) by RedHat
- Relies on community to thrive
- Vlad chose Ruby because of its codebase, and was pleasantly surprised by its community
- Let’s use more Ruby !!!

Rapid River is Hiring !!

- We are a development consultancy

  We build web applications for clients. We do DevOps and Data Engineering too

Rapid River is Hiring !!

- We are remote first

  We're constantly looking for ways to make remote first more convenient for our clients and teams

Rapid River is Hiring !!

- We help each other in our work

  We go out of our way to make our environment collaborative & non-toxic

Rapid River is Hiring !!

- We value strong communication skills & technical excellence

  We actively improve ourselves and our company to stay fresh & current

duck typing

Rapid River is Hiring !!

- Open positions
    - Lead Rails Developer
    - Data Engineer
    - QA Lead
    - Web Analytics Tech Project Manager
- https://rrsoft.co/careers

Thank You