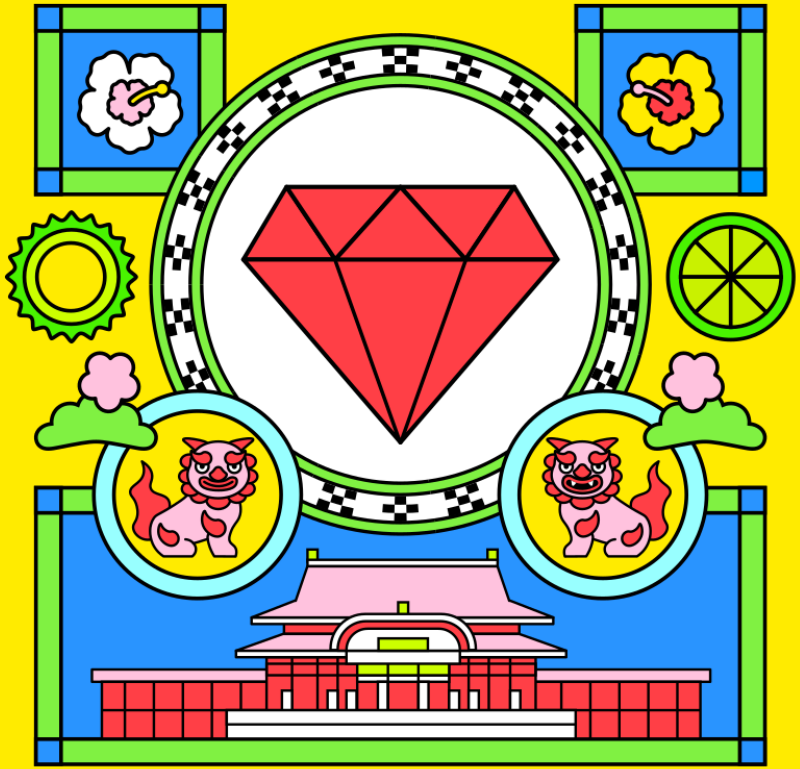


RubyKaigi 2024 MAY15-17 @NAHA,OKINAWA

Unlock The Universal Parsers

A New PicoRuby Compiler

@hasumikin

Monstarlab

[AD] “n-monthly Lambda Note” Vol.4, No.1

Learning How Programming Languages Control Electrical Circuits with PicoRuby

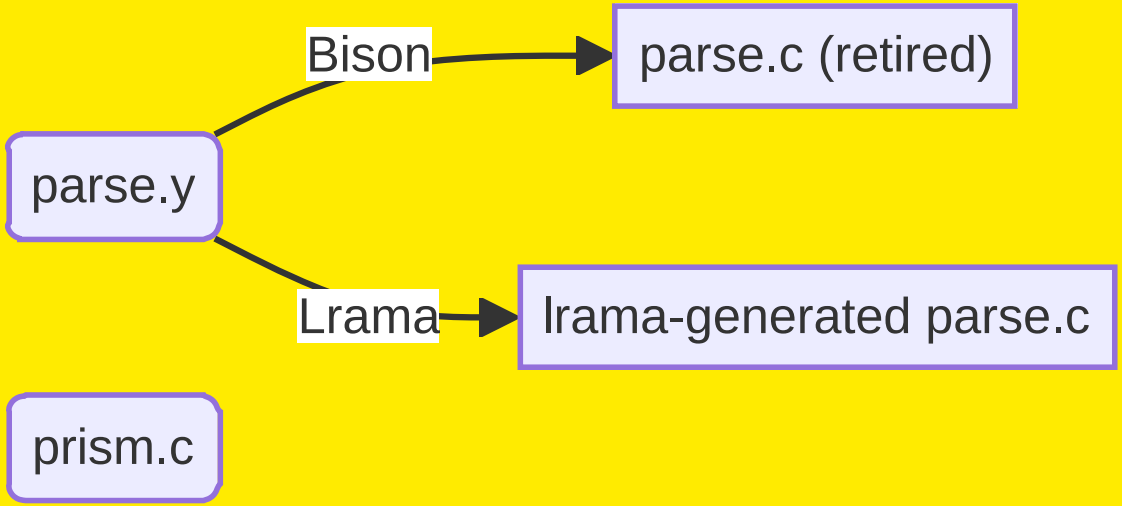
(Japanese only)

1,760 yen (tax included)

RubyKaigi 2024

CRuby's parser

- parse.y and prism.c: human-writable
- parse.c: generated by Bison or Lrama
- prism.c: no generator needed



Prism and Lrama-generated parser

- Prism
  - Newly developed universal Ruby parser
  - Portable and small footprint
  - Easy to integrate
  - Available in form of a gem
- Lrama-generated parser (parse.c)
  - 100% compliant with CRuby in any aspect
  - Not portable by nature

Parsers in Ruby (as of May 2024)

Ruby	Parser
CRuby (default)	Lrama-generated
CRuby (experimental)	Prism
JRuby	Prism
TruffleRuby	Prism
mruby	**by Matz**
PicoRuby	**by hasumikin (me)**
IRB(katakata_irb)	(gem) Prism
RuboCop	(gem) Parser → Prism

Prism in RuboCop

< Back

Koichi ITO

@koic

Koichi Ito is a member of RuboCop core team and open source software maintainer. He is a long time practitioner of Ruby/Rails application development with eXtreme Programming. He is also Engineering Manager and Distinguished Engineer at ESM, Inc.

JA

RuboCop: LSP and Prism

Do you remember the "Smarter, Faster" concept for Ruby 4.0?

RuboCop now includes the built-in LSP as an experimental feature. This feature was essential to meet modern developer experience demands.

Ruby has some LSP implementations and among them, I will focus on the "Smarter, Faster" concept that RuboCop, the de facto standard Linter and Formatter, is aiming for.

Currently, RuboCop uses the Parser gem for Ruby syntax parsing. In addition to this, there is a plan to introduce the Prism Ruby parser as an experimental option. I will also talk about their purposes and designs.

RuboCop will enhance your developer experience by incorporating its built-in LSP. You can receive RuboCop in its current state and future vision.

Parsers in Ruby (I'm working on)

Ruby	Parser
CRuby (default)	Lrama-generated
CRuby (optional)	Prism
JRuby	Prism
TruffleRuby	Prism
mruby	**???????**
PicoRuby	**???????**
IRB(katakata_irb)	(gem) Prism
RuboCop	(gem) Parser → Prism

mruby compiler generates VM code

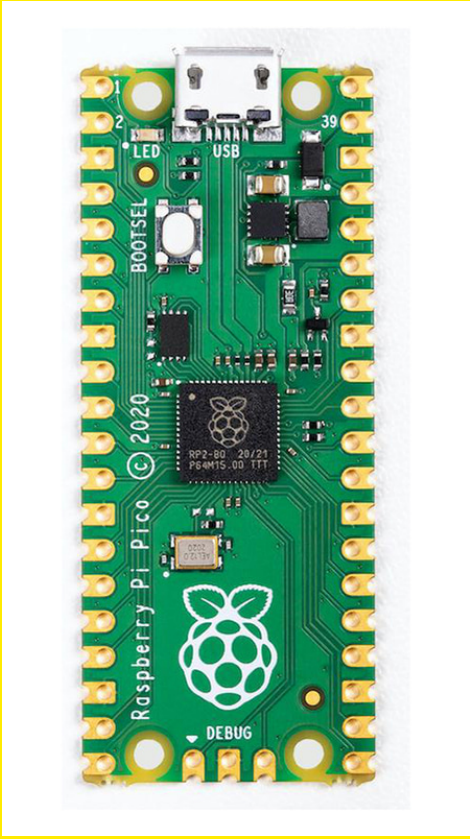
```
$ echo 'puts "Hello, World!"' | bin/mrbc - | xxd # hexdump
00000000: 5249 5445 3033 3030 0000 005e 4d41 545a  RITE0300...^MATZ
00000010: 3030 3030 4952 4550 0000 0042 3033 3030  0000IREP...B0300
00000020: 0000 0036 0000 0004 0000 0000 0000 000a  ...6............
00000030: 5102 002d 0100 0138 0169 0001 0000 0d48  Q..-...8.i.....H
00000040: 656c 6c6f 2c20 576f 726c 6421 0000 0100  ello, World!....
00000050: 0470 7574 7300 454e 4400 0000 0008       .puts.END.....

$ echo 'puts "Hello, World!"' | bin/mrbc - | bin/mruby
Hello, World!
```

(PicoRuby compiler does the same)

PicoRuby compiler must be small footprint

- Raspberry Pi Pico
  - 133 MHz Arm Cortex-M0+ (dual)
  - **264 KB SRAM**
  - 2 MB Flash ROM



RubyKaigi 2024

PicoRuby compiler must be small footprint

- Raspberry Pi Pico
  - 133 MHz Arm Cortex-M0+ (dual)
  - **264 KB SRAM**
  - 2 MB Flash ROM

If a universal parser is not small enough, it won't be "universal"

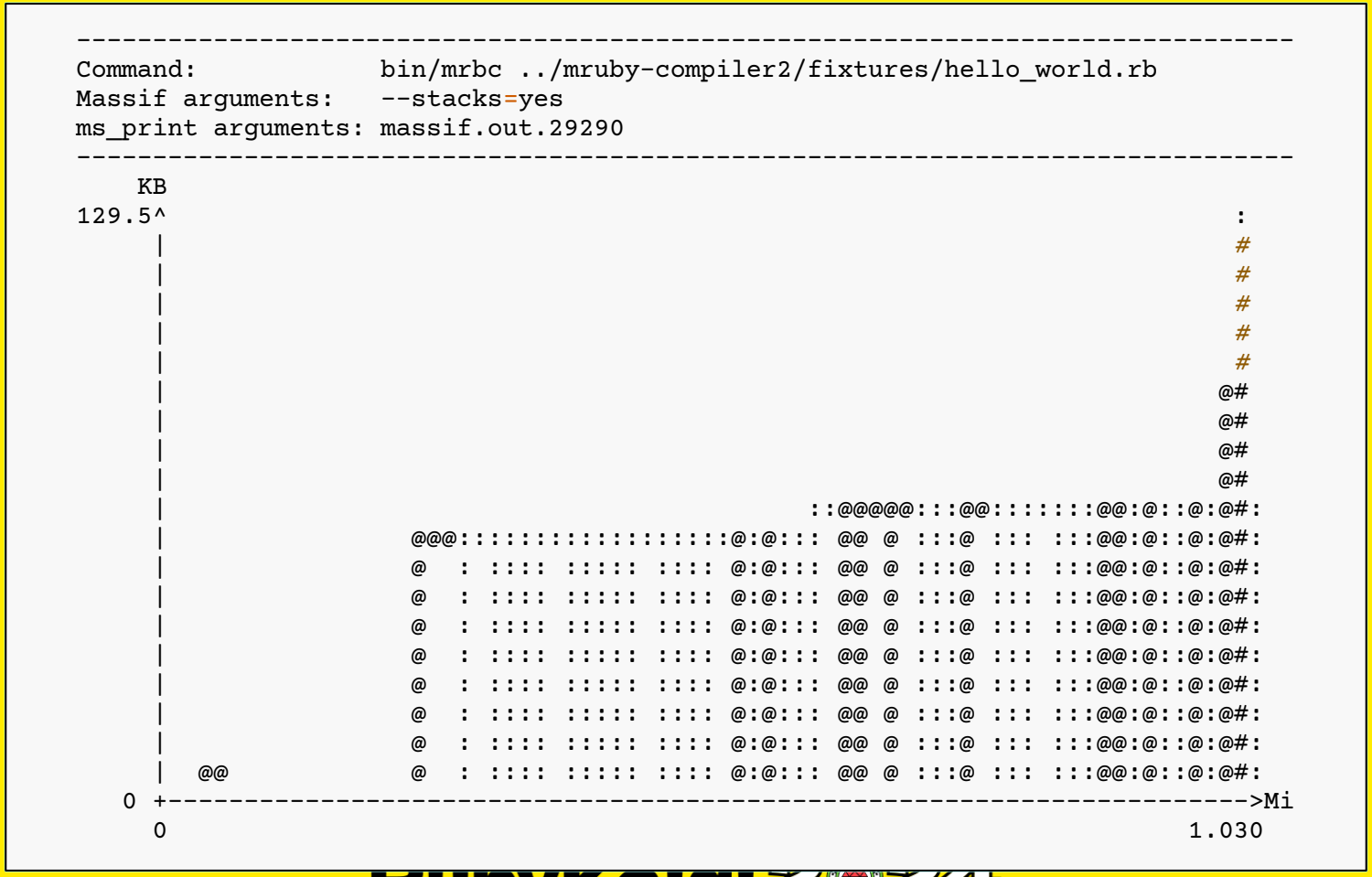
RubyKaigi 2024

PicoRuby compiler must be small footprint

```
$ valgrind \
    --tool=massif \
    --stacks=yes \
    path/to/mrbc hello_world.rb
#=> massif.out.12345

$ ms_print massif.out.12345 | less
```

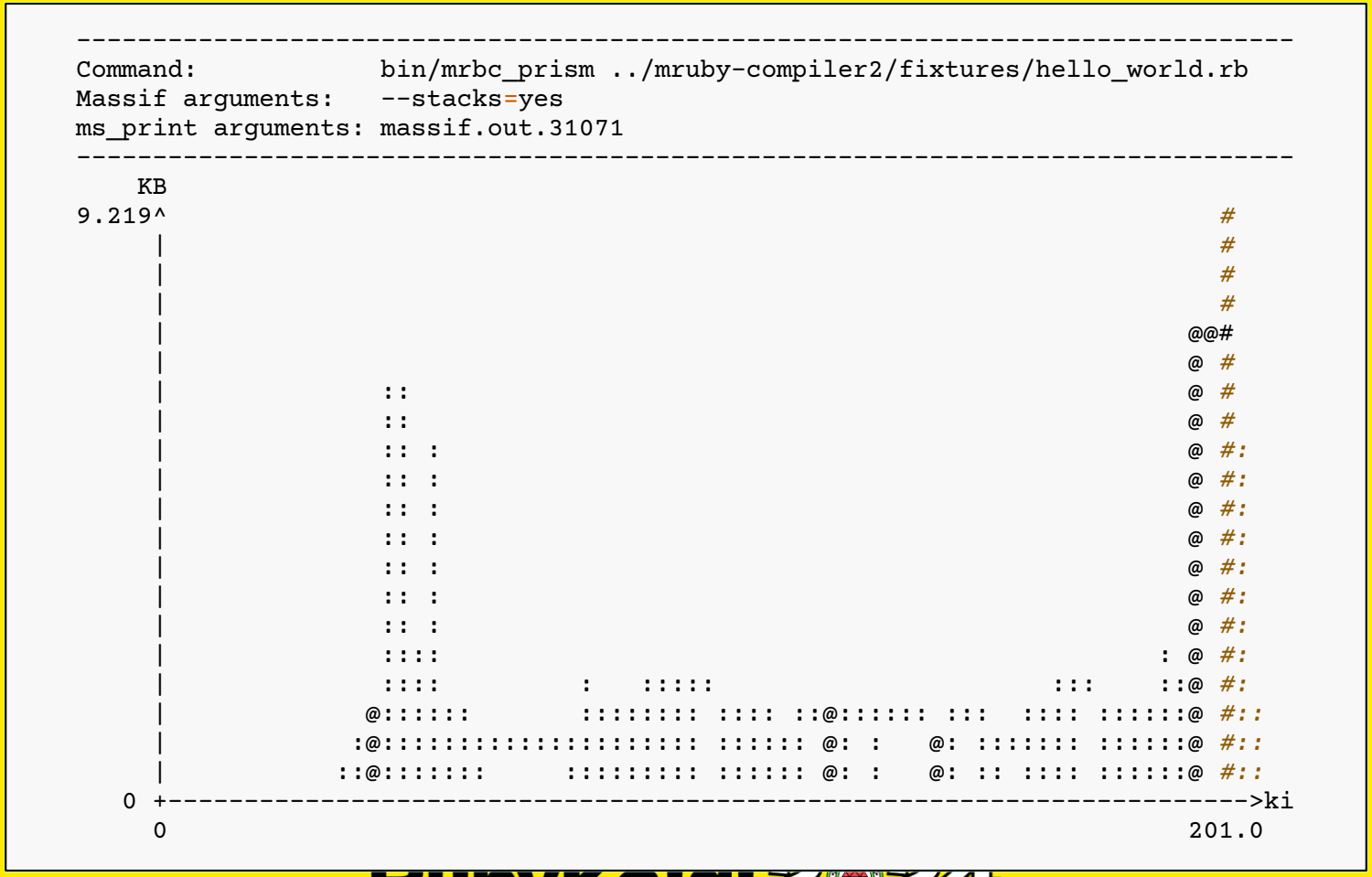
The original mruby Compiler



Existing PicoRuby Compiler

```
--------------------------------------------------------------------------------
Command:            bin/picorbc ../mruby-compiler2/fixtures/hello_world.rb
Massif arguments:   --stacks=yes
ms_print arguments: massif.out.28004
--------------------------------------------------------------------------------
    KB
16.70^                                                                    :
     |                                                                  #:::::::::@
     |                                                                  #:::::::::@:
     |                                                                  #:::::::::@:
     |                                                                  #:::::::::@:
     |                                                                  #:::::::::@:
     |                                                                  #:::::::::@::
     |                                                                  #:::::::::@::
     |                                                                  #:::::::::@::
     |                                                                  #:::::::::@::
     |                                                                  #:::::::::@::
     |                                                                  #:::::::::@::
     |                                                                  #:::::::::@::
     |                                                                  #:::::::::@::
     |             ::                                                   #:::::::::@::
     |             ::   :                                              #:::::::::@::
     |             ::   :                                              #:::::::::@::
     |             ::   :                                              #:::::::::@::
     |             ::   :                                              #:::::::::@::
     |             :::::              :  :::::: @@  :::: :            #:::::::::@::
     |            :::: ::::::::::::::::::::::::: :@ ::::::::@::::@#:::::::::@::
   0 +--------------------------------------------------------------------------->ki
     0                                                                       219.6
```

mrbc_prism


Command: bin/mrbc_prism ../mruby-compiler2/fixtures/hello_world.rb
Massif arguments: --stacks=yes
ms_print arguments: massif.out.31071
KB
9.219^
0 +>ki
0
201.0

RAM consumption

Compiler	RAM consumption
mrbc (original)	129.5 KB
picorbc	16.70 KB
mrbc_prism	9.219 KB*

Figures are measured in 64 bit Linux

* mrbc_prism's small footprint is also due to a new code generator and possibly going to be bigger with development

mrbc_prism

```
mrc_irep *
compile_ruby_to_mruby_vm_code_with_prism(mrc_ccontext *c)
{
    pm_parser_t parser;
    uint8_t *string = "puts 'Hello, World!'"
    pm_parser_init(&parser,
                   string, strlen((const char *)string),
                   NULL);
    pm_node_t *root = pm_parse(&parser);
    mrc_irep *irep = mrc_load_exec(c, (mrc_node *)root);
    pm_node_destroy(&parser, root);
    pm_parser_free(&parser);
    return irep;
}
```

(For illustrative purposes only and non-functional)

Prism, really good stuff

- Well designed APIs and NODE
- Portable
- Small footprint

Still, I'm going to implement Lrama-generated parser. Why?

In Taipei, Taiwan last year...


parse.y is a labyrinth

@ydah_

< Back

Yudai Takada

@ydah_

A contributor for ruby/lrama. Co-Founder of Kyobashi.rb. A Software engineer at ANDPAD, Inc.

JA

Does Ruby Parser dream of highly expressive grammar?

The ruby parser grammar file has been subjected to many complicated workarounds due to the limitations by Bison. You've probably thought at least once or twice that if richer DSL support would make grammar files more readable.

For example, pattern involving multiple repetitions separated by commas, like method call arguments, are quite common. It is only natural to desire a concise way to express them given their frequency. In addition, resolving cases where the added grammar conflicts with the existing one is challenging in ruby, which boasts a flexible grammar. It would be easier to add/change the grammar if there are a solution that could easily resolve this conflict.

In this talk, cover into the outcomes of incorporating additional functionality to extend the DSL in the parser generator. I believe this extension aims to enhance the maintainability of the grammar, thereby contributing to the future development of ruby's syntax.

RubyKaigi 2024

@junk0612

< Back

Junichi Kobayashi

@junk0612

Working as a Rails programmer at ESM, Inc., with a keen interest in Ruby parsers and their related technologies.

JA

From LALR to IELR: A Lrama's Next Step

In parse.y, there is a variable that represents the state of the Lexer, based on the idea that the Parser and the Lexer can be separated. However, in reality, the state of Parser and Lexer are shared and cannot be said to be separated. Also, the Lexer state is manually managed, but historical history has clouded the view, and additions and modifications must be done with care. With the replacement of the parser generator from Bison to Lrama, the time has come to attack parse.y from the parser generator side, and Lrama is trying to solve this problem by generating an algorithmic parser called PSLR. As a first step, I will show how Lrama can generate a parser for a new algorithm called IELR, which is a prerequisite for PSLR. IELR is an improved version of LALR and can parse grammars that LALR cannot. In this presentation, I will explain the implementation of Lrama and how the parser is actually generated.

RubyKaigi 2024


No longer a labyrinth

Why Lrama-generated parser?

- Making parse.y readable and maintainable
- Integrated error tolerance
- Innovative state management of the tokenizer
- In short, a new era of LALR parser generator

Computer-science-wide achievement

RubyKaigi 2024

OK, let me use both

Prism and **Lrama-gen parser**

in my PicoRuby compiler💪

RubyKaigi 2024

Parsers in Ruby (I'm working on)

Ruby	Parser
CRuby (default)	Lrama-generated
CRuby (optional)	Prism
JRuby	Prism
TruffleRuby	Prism
mruby	**Prism and Lrama-gen**
PicoRuby	**Prism and Lrama-gen**
IRB(katakata_irb)	(gem) Prism
RuboCop	(gem) Parser → Prism

AST of Lrama-generated parser

```
$ ruby --dump=p -e'puts "Hello, World!"'
################################################################
## Do NOT use this node dump for any purpose other than  ##
## debug and research.  Compatibility is not guaranteed. ##
################################################################

# @ NODE_SCOPE (id: 3, line: 1, location: (1,0)-(1,20))
# +- nd_tbl: (empty)
# +- nd_args:
# |   (null node)
# +- nd_body:
#     @ NODE_FCALL (id: 0, line: 1, location: (1,0)-(1,20))*
#     +- nd_mid: :puts
#     +- nd_args:
#         @ NODE_LIST (id: 2, line: 1, location: (1,5)-(1,20))
#         +- as.nd_alen: 1
#         +- nd_head:
#         |   @ NODE_STR (id: 1, line: 1, location: (1,5)-(1,20))
#         |   +- nd_lit: "Hello, World!"
#         +- nd_next:
#             (null node)
```

Version 0.0.0.0.1 of mrbc_lrama

(A mruby compiler using Lrama-generated parser)

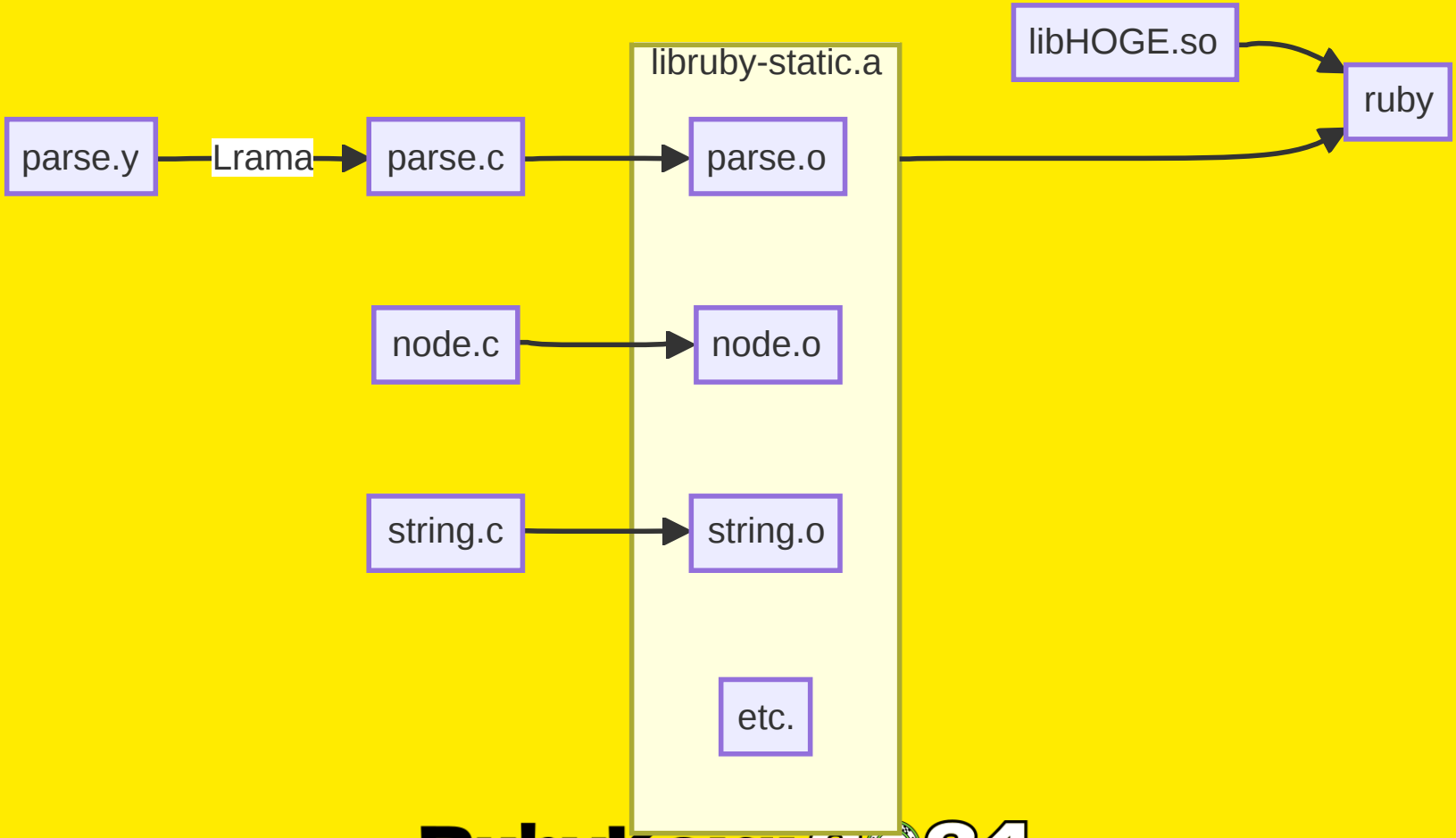
RubyKaigi 2024

How to integrate Lrama-gen parser

```
$ cd ruby/ruby
$ ./autogen.sh #=> configure
$ ./configure  #=> common.mk etc.
$ make
$ ls -l *.a
-rw-r--r-- 1 145797150 libruby-static.a
```

An archive file containing all the object files of the Ruby interpreter

libruby-static.a



Even CRuby can be embedded

```c
// hello_world.c
#include <ruby.h>
int
main(int argc, char **argv)
{
    ruby_init();
    rb_eval_string("puts 'Hello, World!'");
    ruby_finalize();
    return 0;
}
```

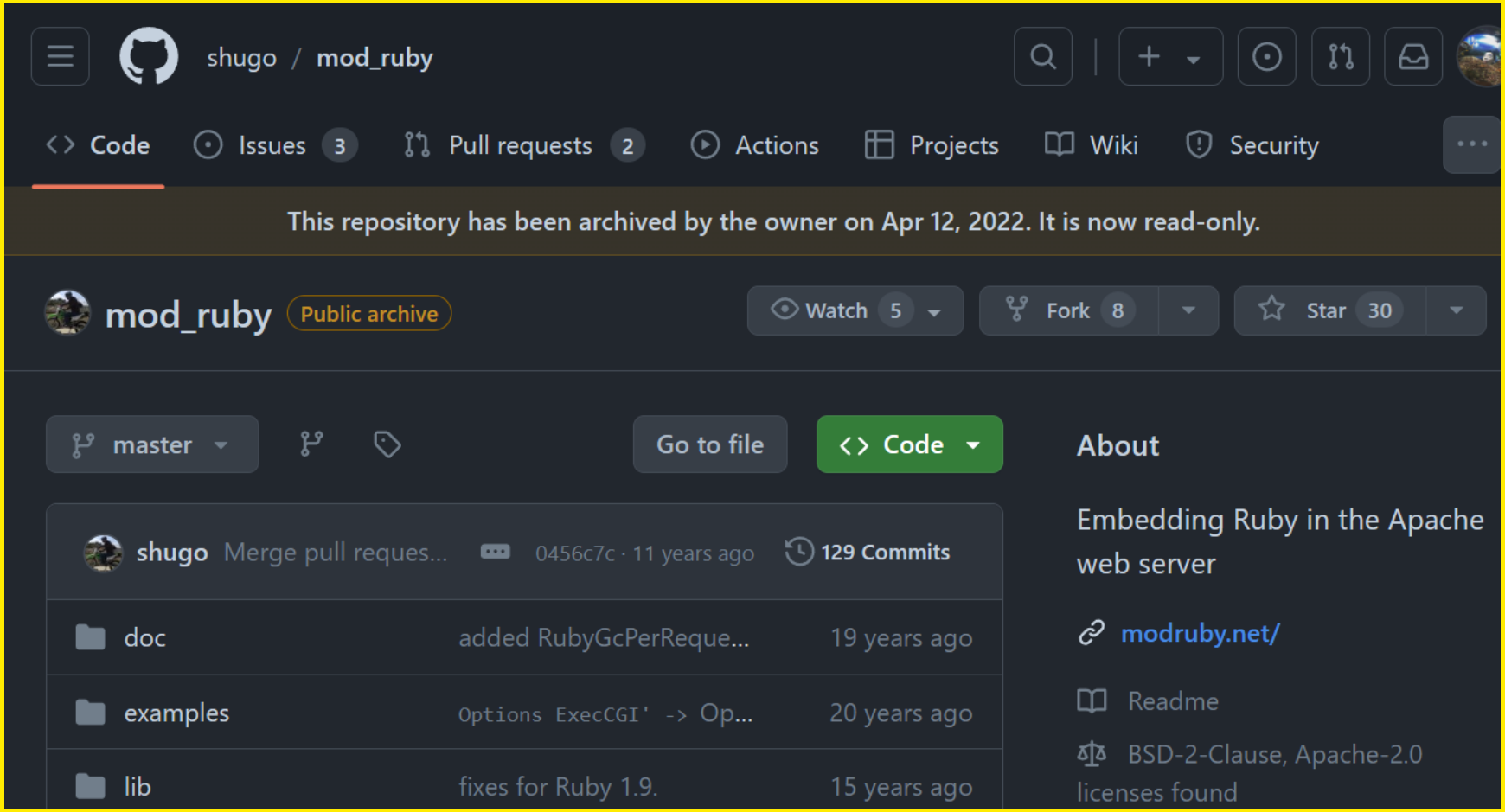
(For illustrative purposes only and non-functional)

Even CRuby can be embedded

```
$ cc -o hello_world \
  hello_world.c \
  -I/path/to/ruby/include \
  -L/path/to/ruby \
  -l:libruby-static.a
#    ^^^^^^^^^^^^^^^^^

$ ./hello_world
#=> Hello, World!
```

Legenday work of embedded CRuby

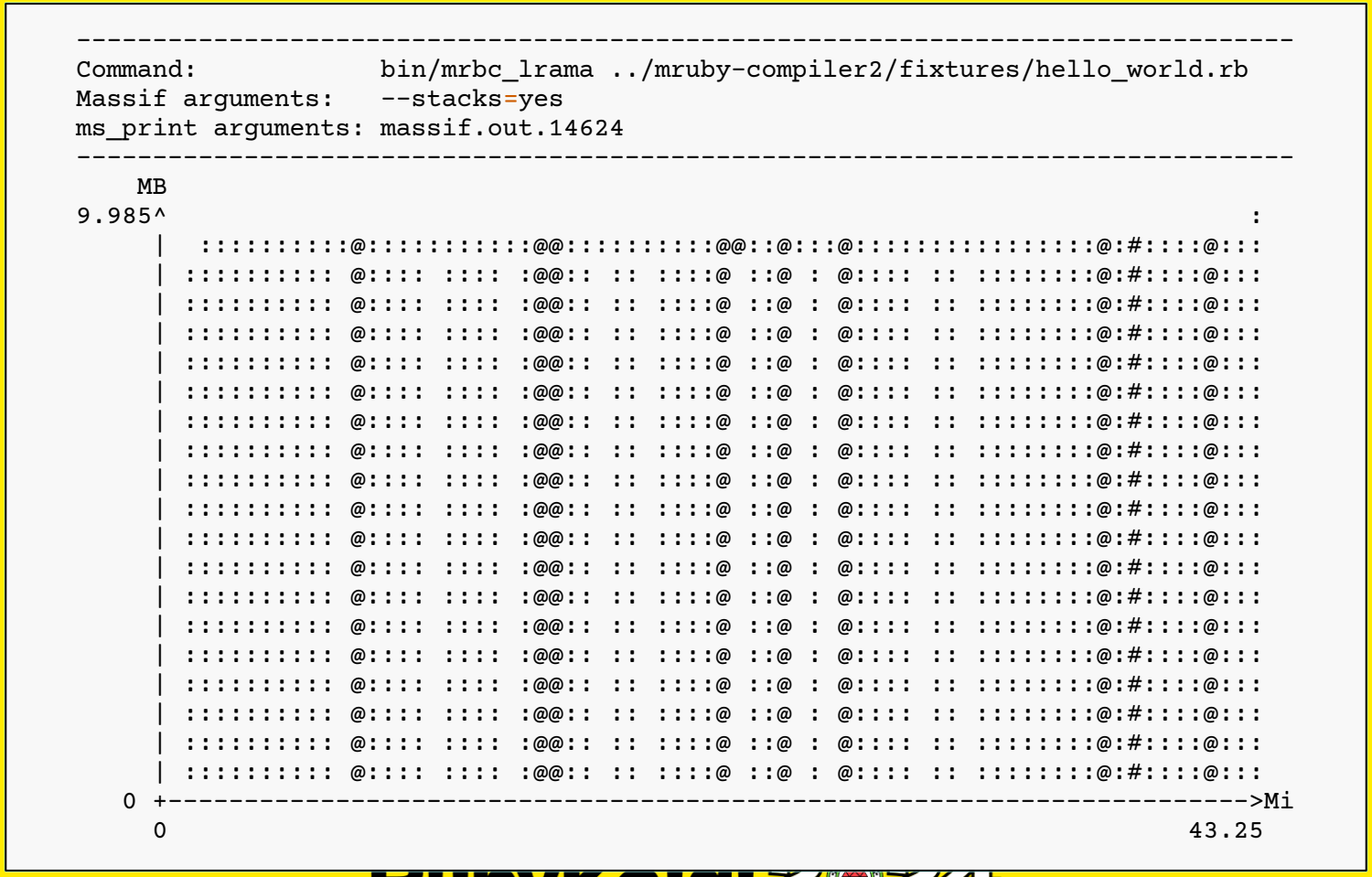


RubyKaigi 2024

Let’s see the memory consumption of mrbc_lrama

RubyKaigi 2024

Memory consumption of mrbc_lrama



9.985 MB 

RubyKaigi 2024

RAM consumption

Compiler	RAM consumption
mrbc (original)	129.5 KB
picorbc	16.70 KB
mrbc_prism	9.219 KB
mrbc_lrama	9.985 **MB**

Why consumes a lot of RAM?

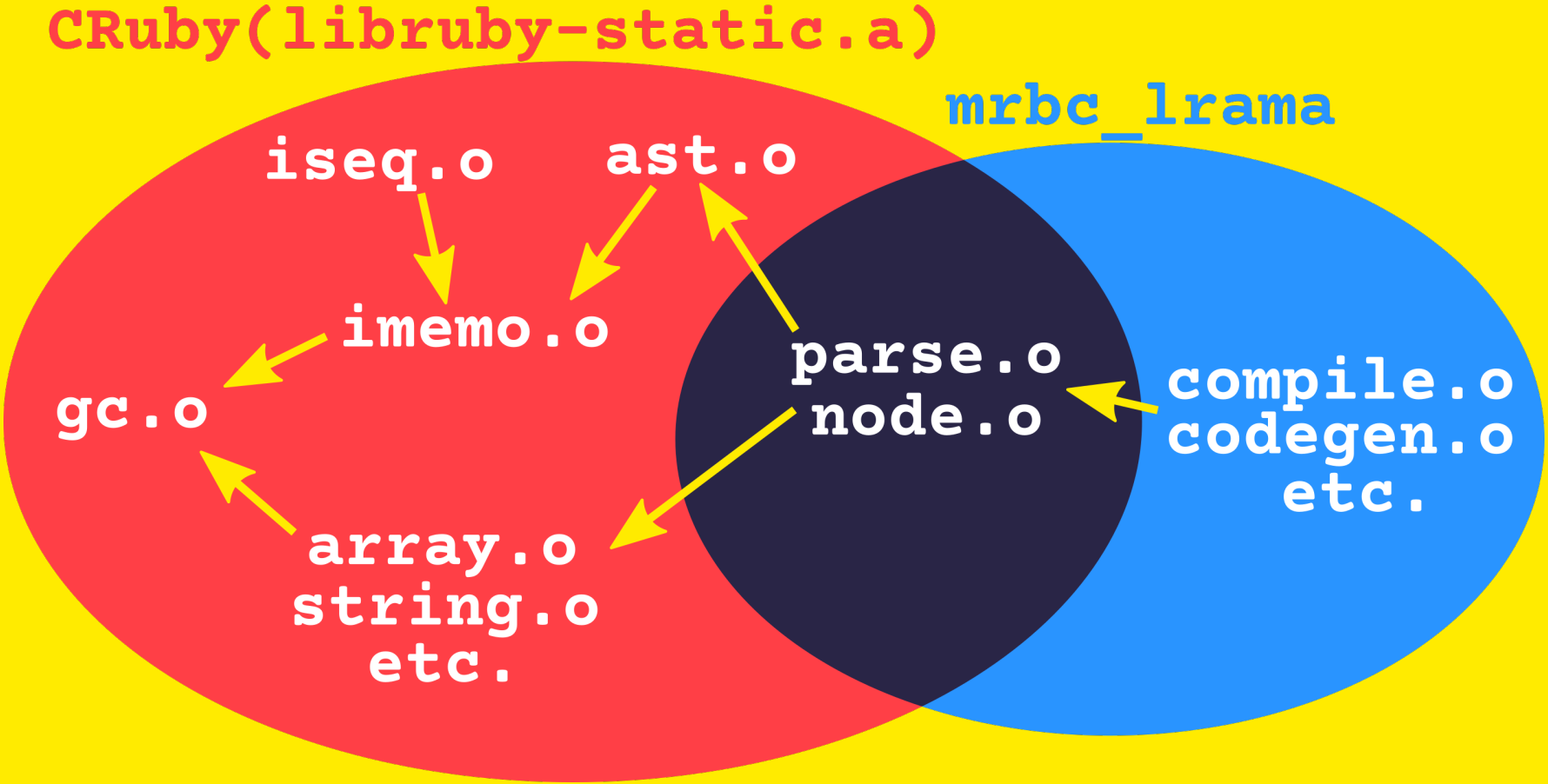
```
mrc_irep *
compile_ruby_to_mruby_vm_code_with_lramagen(mrc_ccontext *c)
{
    ruby_init(); // 👍👍👍👍👍👍👍👍
    rb_parser_t *parser = rb_ruby_parser_new();
    rb_ast_t *ast = rb_ruby_parser_compile_string(
                    parser, "", "puts 'Hello, World!'", 0);
    mrc_irep *irep = mrc_load_exec(c, (mrc_node *)ast->body.root);
    ruby_finalize();
    return irep;
}
```

(For illustrative purposes only and non-functional)

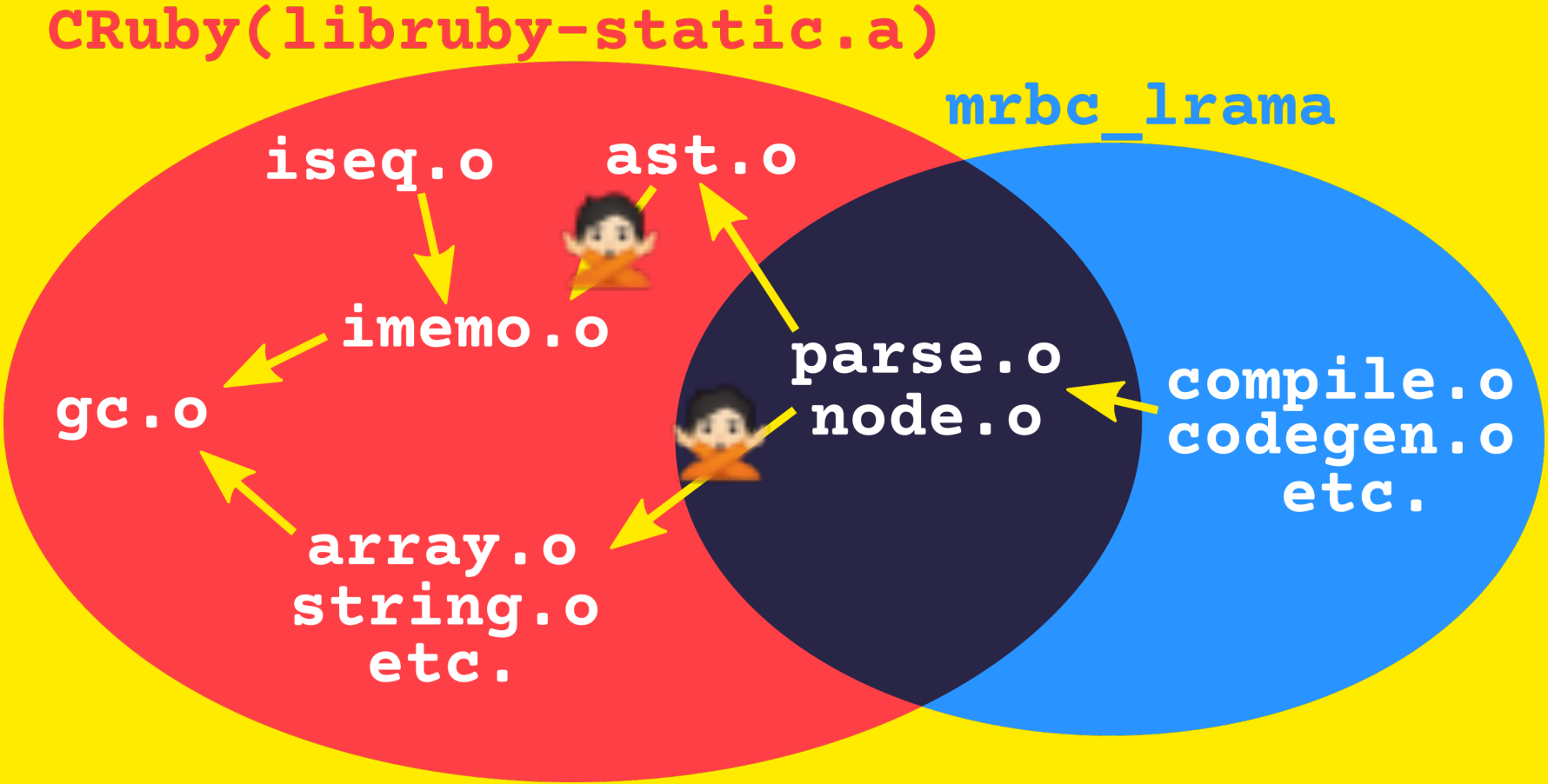
`ruby_init()` prepares the CRuby's GC.

However, if you remove it, you get **SEGV**

Dependencies (actual)



Dependencies (goal)



DeGC is what we need

- DeVALUE
  - **Decouple VALUE** that is a C-level representation of a Ruby object such as RString and RArray
- DeIMEMO
  - **Decouple IMEMO** that is a C-level object taken care of by GC
- DeGC
  - After all, we are going to **decouple GC** from the parser

Relevant work of DeVALUE

S.H.
@S-H-GAMELINKS

Software engineer at ESM, Inc.

Contributing to the Ruby Parser

How I learned to stop worrying and contributed the Ruby parser.

Lightning Talks today

RubyKaigi 2024

rb_parser_ary_t* instead of RArray

```
enum rb_parser_ary_data_type {
    PARSER_ARY_DATA_AST_TOKEN,
    PARSER_ARY_DATA_SCRIPT_LINE
};

typedef struct rb_parser_ary {
    enum rb_parser_ary_data_type data_type;
    rb_parser_ary_data *data; // typedef of void*
    long len;                 // current size
    long capa;                // capacity
} rb_parser_ary_t;
```

e.g. DeVALUE of tokens

```
node = RubyVM::AbstractSyntaxTree.
         parse('puts "Hello, World!"',
               keep_tokens: true) # 👈

node.tokens
#=>
[[0, :tIDENTIFIER, "puts", [1, 0, 1, 4]],
 [1, :tSP, " ", [1, 4, 1, 5]],
 [2, :tSTRING_BEG, "'", [1, 5, 1, 6]],
 [3, :tSTRING_CONTENT, "Hello, World!", [1, 6, 1, 19]],
 [4, :tSTRING_END, "'", [1, 19, 1, 20]]]
```

e.g. DeVALUE of tokens

```c
// parse.y
static void parser_append_tokens(struct parser_params *p,...
{
    VALUE ary = rb_ary_new2(4); // A token
    (...)
    rb_ary_push(p->tokens, ary);
    (...)
}

(...)

static VALUE yycompile0(VALUE arg)
{
    (...)
    if (p->keep_tokens) {
        p->ast->node_buffer->tokens = tokens;
        p->tokens = NULL;
    }
    (...)
}
```

e.g. DeVALUE of tokens

```
struct parser_params {
    (...)
    VALUE tokens;
}
```

```
struct parser_params {
    (...)
    rb_parser_ary_t *tokens;
}
```

e.g. DeVALUE of script_lines

- `ast->body.script_lines`
- `iseq->variable.script_lines`
- `SCRIPT_LINES__`

Three kinds of script_lines

script_lines in AST

```ruby
node = RubyVM::AbstractSyntaxTree.
         parse("puts 'Hello, World!'",
               keep_script_lines: true) # 👈

node.script_lines
#=>
["puts 'Hello, World!'"]
```

script_lines in ISEQ

```
iseq = RubyVM::InstructionSequence.
          compile("puts 'Hello, World!'",
                  keep_script_lines: true) # 👈

iseq.script_lines
#=>
["puts 'Hello, World!'"]
```

SCRIPT_LINES__ in top-level

```ruby
# hello_world.rb
puts 'Hello, World!'
```

```ruby
SCRIPT_LINES__ = {}
require './hello_world'
p SCRIPT_LINES__
#=>
{"./hello_world"=>["puts 'Hello, World!'"]}
```

debug.rb and tracer.rb used to use SCRIPT_LINES__

Internally, they were the same RArray

- `ast->body.script_lines`
- `iseq->variable.script_lines`
- `SCRIPT_LINES__[filename]`

Build script_lines as a **`rb_parser_ary_t`**
during parsing, then convert it to a **VALUE**
or when **`Node#script_lines`** is called

IMEMO (Internal MEMOry Object)

- IMEMO is a C-level object taken care of by GC
- Not a VALUE
- Five-word restriction to be GC-ed
  - The first word has to be the flags
  - You can only use the five words in total

e.g. DelMEMO of AST (before)

```
typedef struct rb_ast_struct {
    VALUE flags;                    // To be removed
    node_buffer_t *node_buffer;
    struct {
        const NODE *root;
        VALUE *script_lines; // To be deVALUEd
        signed int frozen_string_literal:2;
        signed int coverage_enabled:2;
    } body;
} rb_ast_t;
```

Five words

e.g. DelMEMO of AST (after)

```
typedef struct rb_ast_struct {
    node_buffer_t *node_buffer;
    struct {
        const NODE *root;
        rb_parser_ary_t *script_lines;
        int line_count;              // Separeted from script_lines
        signed int frozen_string_literal:2;
        signed int coverage_enabled:2;
    } body;
#ifdef UNIVERSAL_PARSER
    const rb_parser_config_t *config; // Moved from node_buffer_t
#endif
} rb_ast_t;
```

...Still five words?

RubyKaigi 2024

e.g. DelMEMO of AST (after)

```
typedef struct rb_ast_struct {
    node_buffer_t *node_buffer;
    struct {
        const NODE *root;
        rb_parser_ary_t *script_lines;
        int line_count;                 // One word in 32 bit
        signed int frozen_string_literal:2;
        signed int coverage_enabled:2;
    } body;
#ifdef UNIVERSAL_PARSER
    const rb_parser_config_t *config; // Moved from node_buffer_t
#endif
} rb_ast_t;
```

Six words in 32 bit 💡

DeVALUE and DeIMEMO automatically complete DeGC😊

Amount of diff in ruby/ruby by me

```
$ git log --author=hasumikin \
  --since=2024-01-01 --until=2024-4-30 \
  --oneline

ddd8da4b6b [Universal parser] Improve AST structure
9ea77cb351 Remove unnecessary assignment to ast->body.line_count
55a402bb75 Add line_count field to rb_ast_body_t
2244c58b00 [Universal parser] Decouple IMEMO from rb_ast_t
9b1e97b211 [Universal parser] DeVALUE of p->debug_lines and ast...
f5e387a300 Separate SCRIPT_LINES__ from ast.c
8aa8fce320 Fix return-type warning in compile.c
ce544f8dbd [ruby/prism] [Compatibility] Improve printf format
40ecad0ad7 [Universal parser] Fix -Wsuggest-attribute=format warnings
9a19cfd4cd [Universal Parser] Reduce dependence on RArray in parse.y
b95e2cdca7 [ruby/prism] Additional fix of adding `x` prefix after ...
54f27549e2 [ruby/prism] Chage some names
c4bd6da298 [ruby/prism] Make alloc interface replaceable
f0a46c6334 [ruby/prism] Include unistd.h before cheching _POSIX_...
15b53e901c [ruby/prism] Use `_POSIX_MAPPED_FILES` and `_WIN32` to ...
```

Amount of diff in ruby/ruby by me

```
$ git log --author=hasumikin \
  --since=2024-01-01 --until=2024-4-30 \
  --pretty=tformat: --numstat \
  | awk '{add += $1; del += $2} END
         {print "++:", add, "\n--:", del}'

++: 1321
--: 818
```

Despite 2000+ lines of changes,
CRuby does not alter its behavior.

Let's try `valgrind --tool=massif` again

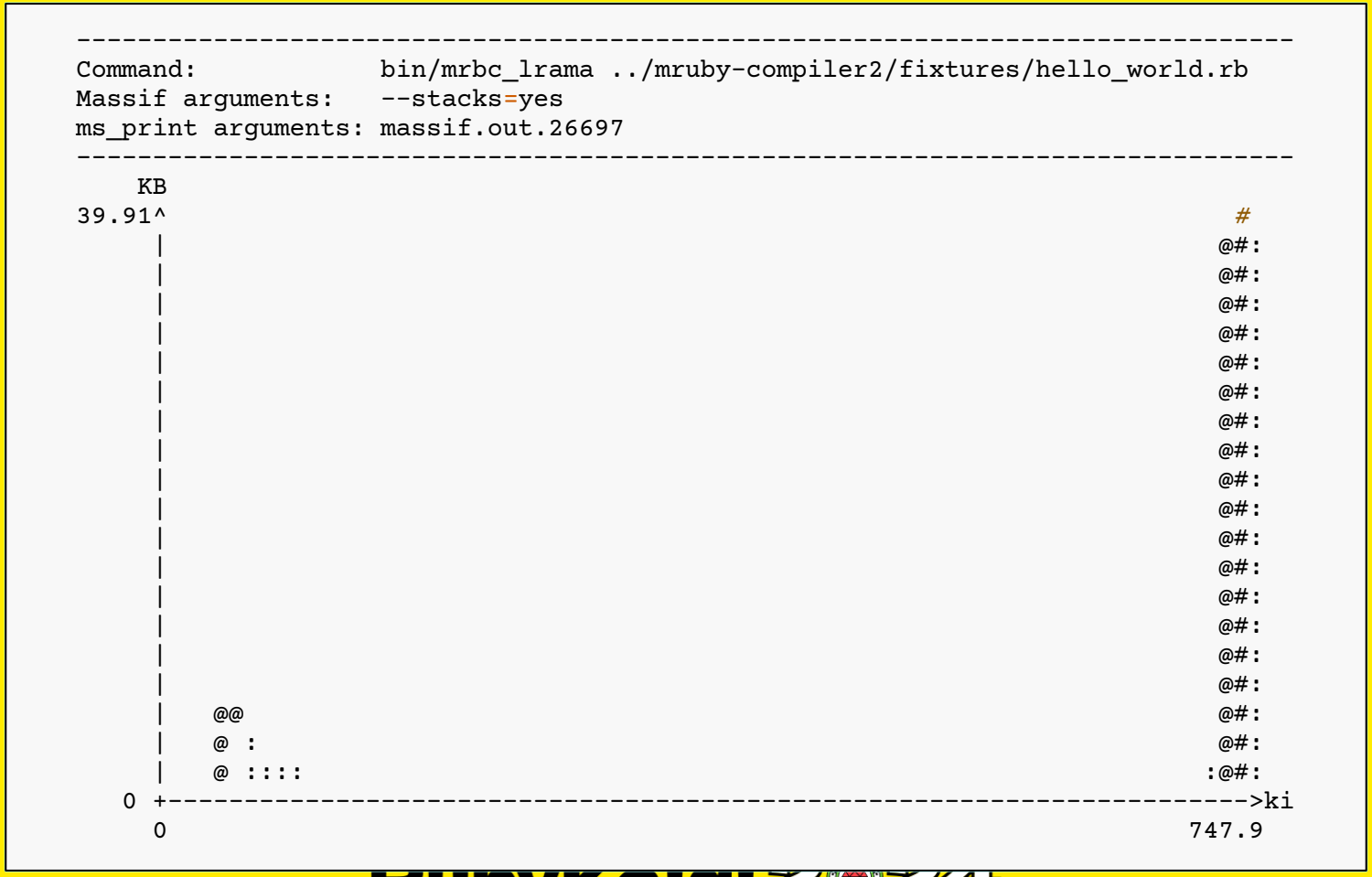
```
mrc_irep *
compile_ruby_to_mruby_vm_code_with_lramagen(mrc_ccontext *c)
{
    // ruby_init(); 👈 Remove this
    rb_parser_t *parser = rb_ruby_parser_new();
    rb_ast_t *ast = rb_ruby_parser_compile_string(
                    parser, "", "puts 'Hello, World!'", 0);
    mrc_irep *irep = mrc_load_exec(c, (mrc_node *)ast->body.root);
    // ruby_finalize(); 👈 Remove this
    return irep;
}
```

(For illustrative purposes only and non-functional)

Now we can remove

`ruby_init()` and **`ruby_finalize()`**

RAM consumption of Lrama-gen parser



RAM consumption of Lrama-gen parser

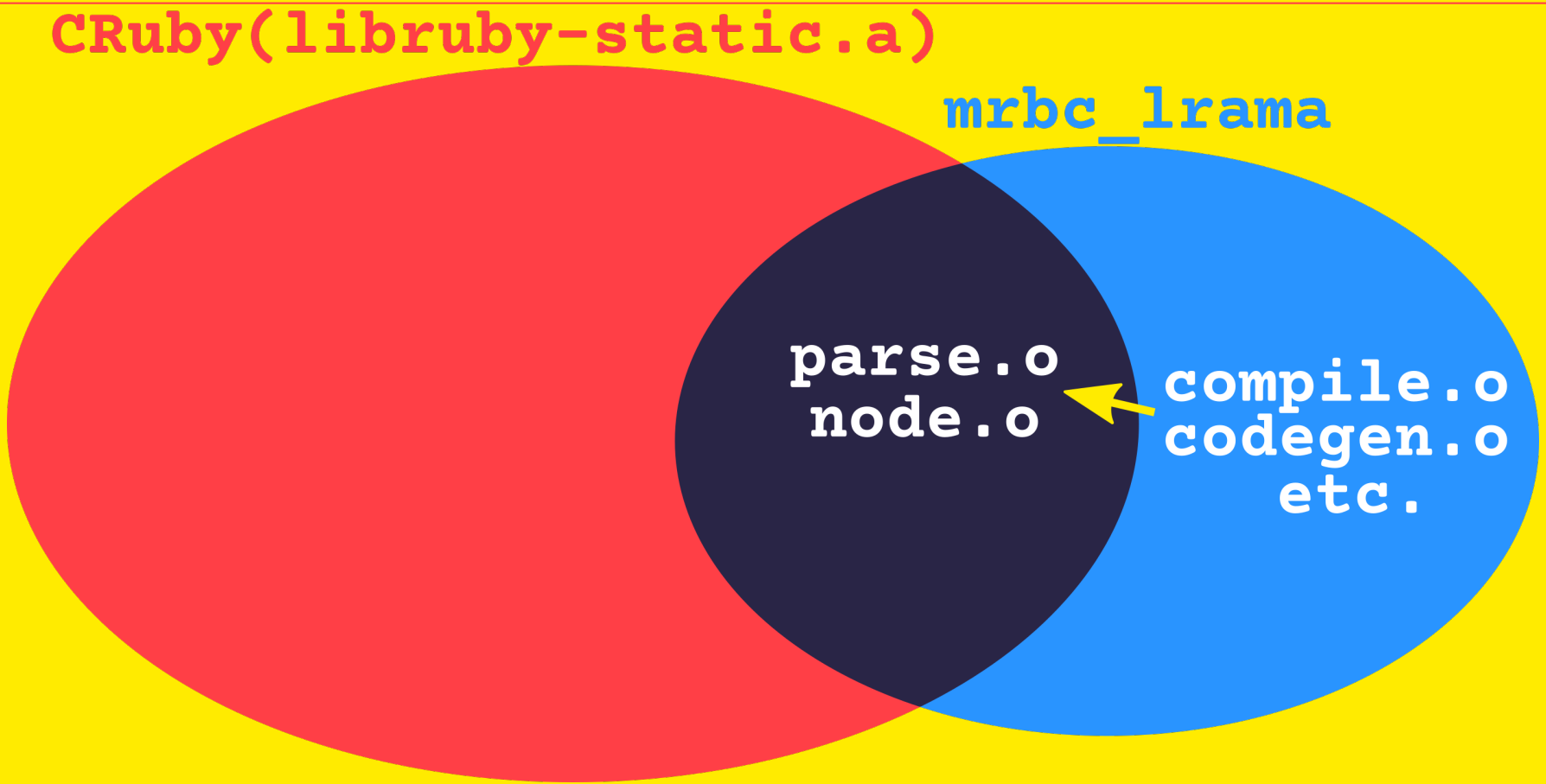
Compiler	RAM consumption
mrbc (original)	129.5 KB
picorbc	16.70 KB
mrbc_prism	9.219 KB
mrbc_lrama	9.985 MB 👉 **39.91 KB** 🎉

96% reduction!!!

(Ready to fight with Prism)

RubyKaigi 2024

Finally, DeGCed!!!



Remaining issues of Lrama-gen parser

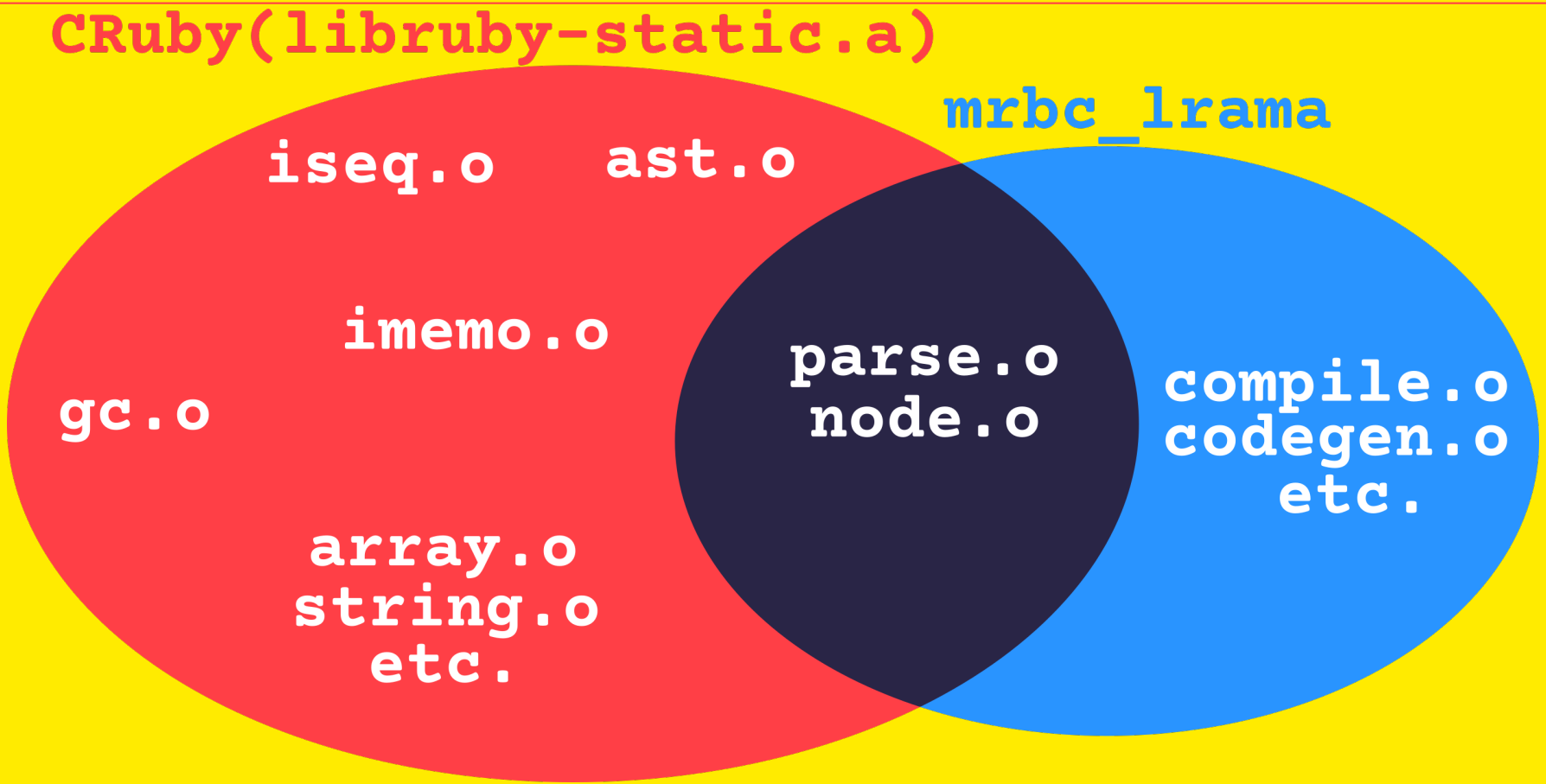
- Make NODE(AST) design compatible with Prism
- Fine-tuning of the RAM consumption
  - 39.91 KB is the result of just **deGCed**
  - There is still more that can be done 💪
- **ROM size** and **cross compilation**

ROM size also matters

```
$ ls -lSr picoruby/bin/
-rwxr-xr-x   851648 picorbc
-rwxr-xr-x  2512528 mrbc_prism
-rwxr-xr-x  3386824 mrbc # original
-rwxr-xr-x 25080800 mrbc_lrama # 24 MB
```

- picorbc solely consists of mruby compiler (ascii only)
- mrbc_prism contains all encodings (can be omitted)
- mrbc contains mruby-core
- mrbc_lrama contains CRuby (miniruby)

mrbc_lrama: 24 MB (on x86-64)



To fix, we need cross compilation

```
$ cd ruby
$ ./autogen.sh
$ ./configure
#=> .ext/include/x86_64-linux/ruby/config.h

$ ./configure --host=arm-linux-eabi
#=> .ext/include/arm-linux-eabi/ruby/config.h
```

What is `ruby/config.h`?

ruby/config.h is platform-specific config

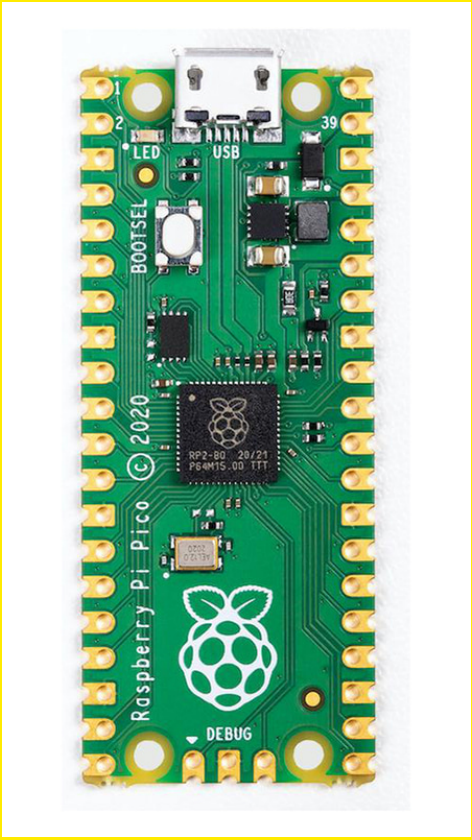
```
// .ext/include/arm-linux-eabi/ruby/config.h
...
#define RUBY_ABI_VERSION 0
#define HAVE_STDIO_H 1
...
#define SIZEOF_INT 4
#define SIZEOF_SHORT 2
...
#define SIZEOF_VOIDP 4   // 👈 32 bit
#define SIZEOF_FLOAT 4
#define SIZEOF_DOUBLE 8
...
```

Target triplets [CPU]-[ambiguous]-[ambiguous]

- x86-64-linux-gnu:
  - x86 64 bit Linux with GNU libc
- aarch64-linux-gnu:
  - Arm 64 bit Linux with GNU libc
- arm-linux-gnueabi(hf):
  - Arm 32 bit Linux with GNU libc for EABI (hard-float)
    EABI: Embedded Application Binary Interface
- arm-none-eabi:
  - Arm 32 bit **bare-metal** with newlib

arm-none-eabi

- RP2040 (Raspberry Pi Pico)
  - 133 MHz Arm Cortex-M0+ (dual)



Configurable with arm-none-eabi? No

```
$ cd ruby
$ ./autogen.sh #=> configure
$ ./configure --host=arm-none-eabi
#=> error😢😢😢😢😢😢
```

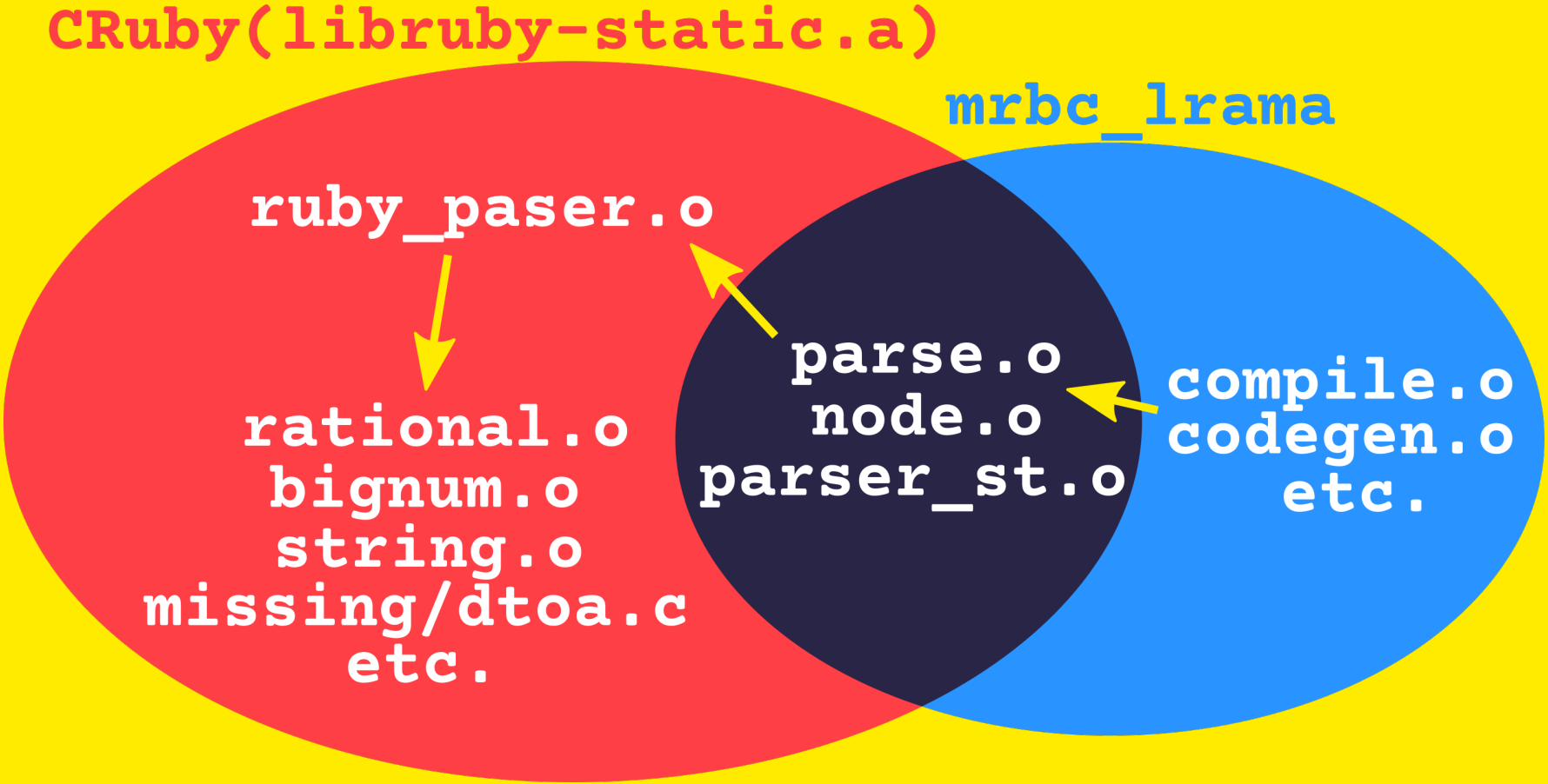
Why?

Because “none-eabi” doesn’t support things like `stdio.h` that CRuby requires

The next step is to make an effective build

- Configurable against bare-metal
  - Arm, ESP32, and PIC32, etc.
- Compile only what we need instead of making libruby-static.a
  - `parse.o` + `node.o` + `parser_st.o` + `?.o`
  - Resolving the lack of dependencies is another challenge

Still some deVALUE to do



Functions to be deVALUEd

```
VALUE rb_node_file_path_val(const NODE *node);
VALUE rb_node_str_string_val(const NODE *node);
VALUE rb_node_line_lineno_val(const NODE *node);
VALUE rb_node_encoding_val(const NODE *node);
VALUE rb_node_integer_literal_val(const NODE *n);
VALUE rb_node_float_literal_val(const NODE *n);
VALUE rb_node_rational_literal_val(const NODE *n);
VALUE rb_node_imaginary_literal_val(const NODE *n);
VALUE rb_node_regx_string_val(const NODE *node);
VALUE rb_node_sym_string_val(const NODE *node);
VALUE rb_str_new_parser_string(rb_parser_string_t *str);
VALUE rb_str_new_mutable_parser_string(rb_parser_string_t *str);
VALUE rb_sym2id(VALUE sym);
```

Further deVALUE is needed

Build with dummy functions (PoC)

```
# Linking libruby-static.a
-rwxr-xr-x  2512528 mrbc_prism
-rwxr-xr-x 25080800 mrbc_lrama
```

```
# Linking dummy functions
-rwxr-xr-x  2512528 mrbc_prism
-rwxr-xr-x  1927440 mrbc_lrama
```

Now mrbc_lrama is barely able to compile

```
puts 'Hello, World!'
```

Working in progress...

https://github.com/picoruby/mruby-compiler2

https://github.com/picoruby/mruby-bin-mrbc2

FYI, PicoRuby talk at #rubykaigiC 16:40

< Back

Ryo Kajiwara

@sylph01

A wild (freelance) programmer. I can do cryptography and authentication/authorization a little.

EN

Adding Security to Microcontroller Ruby

"Actual" Internet of Things with Ruby is here!

Continuing on the journey through Ruby's cryptographic libraries, I gave it a shot to make Raspberry Pi Pico W speak TLS in PicoRuby, making it even closer to other languages' environments.

This talk will cover how networking, cryptography, and HTTP/HTTPS is implemented both in your normal Ruby and mruby world.

RubyKaigi 2024

Also, PicoRuby appears in Lightning Talks

Hayao Kimura
@hachiblog

Co-Founder of Kyobashi.rb.
Organizer of Kaigi on Rails. A
Software engineer at freee K.K.

Drive Your Code: Building an RC Car by Writing Only Ruby

Electronics projects often come with the daunting task of setting up development environments, which can deter many from getting started. However, as Rubyists, we have PicoRuby and the software implemented with it, called R2P2. With R2P2, you can embark on electronics projects by simply writing Ruby code in your editor, no setup required. This Talk will introduce you to the basics of electronic projects using R2P2 and showcase how to create a functioning RC car using this technology. Learn how effortlessly you can bring your code to life and drive an RC car, bridging the gap between software and hardware with Ruby's simplicity and power.

This guy may give you a Raspi Pico

https://twitter.com/hanachin_/status/1790713159906681259

Catch him at the venue

RubyKaigi 2024

Lrama-gen univ-parser is almost there!!!

- Embed libruby-static.a in PicoRuby compiler
  - → RAM consumption: **10 MB** 😨
  - → ROM size: **24 MB** 😫
- → DeVALUE, DeIMEMO, DeGC
  - → RAM consumption: **40 KB** 🥳
- → Improve build process
  - → ROM size: **1.9 MB** 😊

Wrap-up

- PicoRuby compiler will integrate both Prism and Lrama-gen parser
- Rest of the work
  - Cross compile
  - DeVALUE (a little more)
  - Symbol table (ID)
  - Fine-tuning

github.com/picoruby/picoruby