mruby de Hello World!

HASUMI Hitoshi @hasumikin

Monstar Lab, Shimane office

富山Ruby会議01

2019-11-03

Matsue.rb

Today is

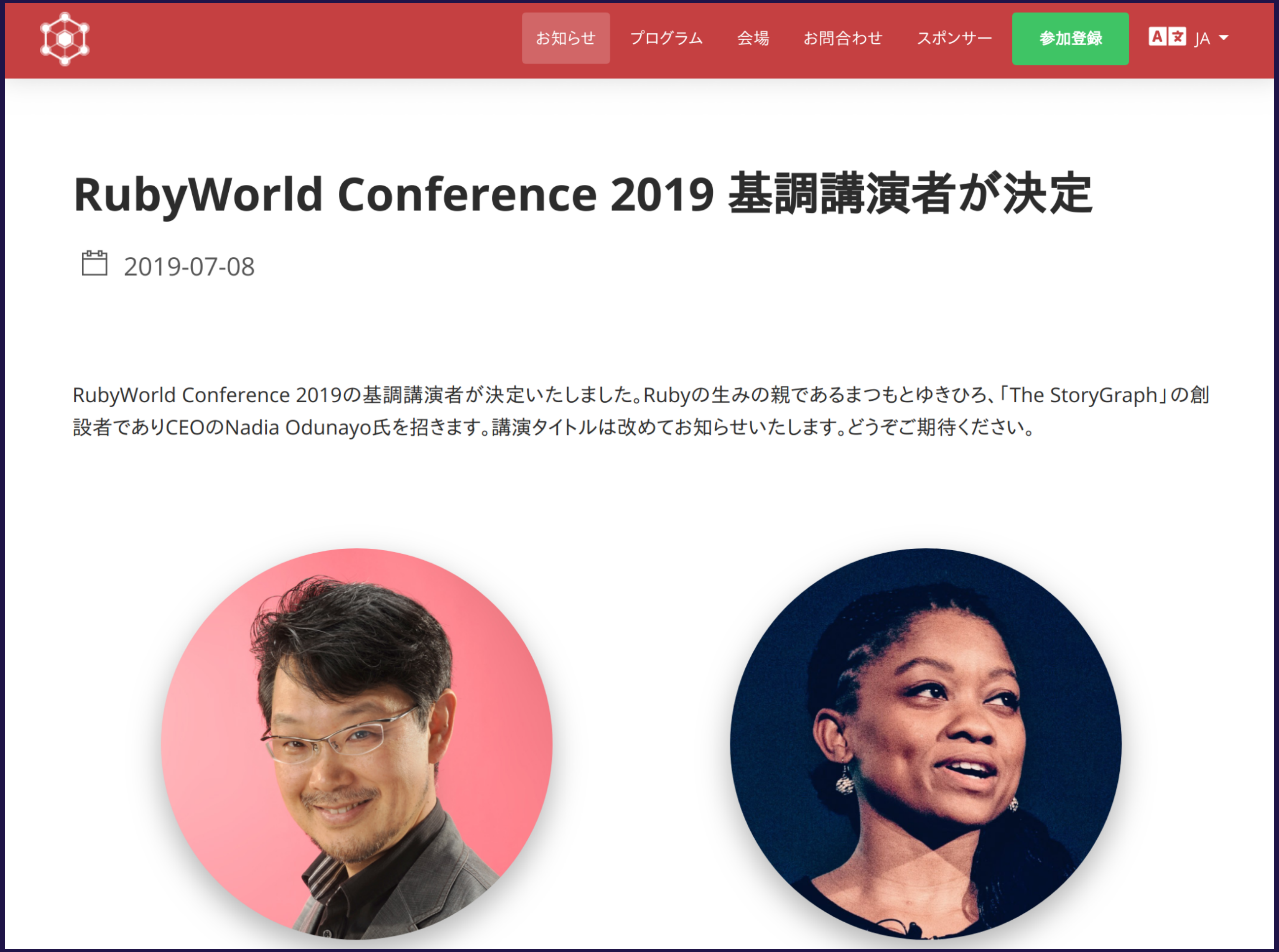
Today is

My birthday 🎂

me

- HASUMI Hitoshi
- @hasumikin
- Microcontroller detective
- RubyKaigi 2018, 2019
- RubyWorld 2018
- KRKRB 2019 (Poland)
- RubyConf 2019 (the US)

RubyWorld Conference 2019



RubyWorld Conference 2019 基調講演者が決定

2019-07-08

RubyWorld Conference 2019の基調講演者が決定いたしました。Rubyの生みの親であるまつもとゆきひろ、「The StoryGraph」の創設者でありCEOのNadia Odunayo氏を招きます。講演タイトルは改めてお知らせいたします。どうぞご期待ください。

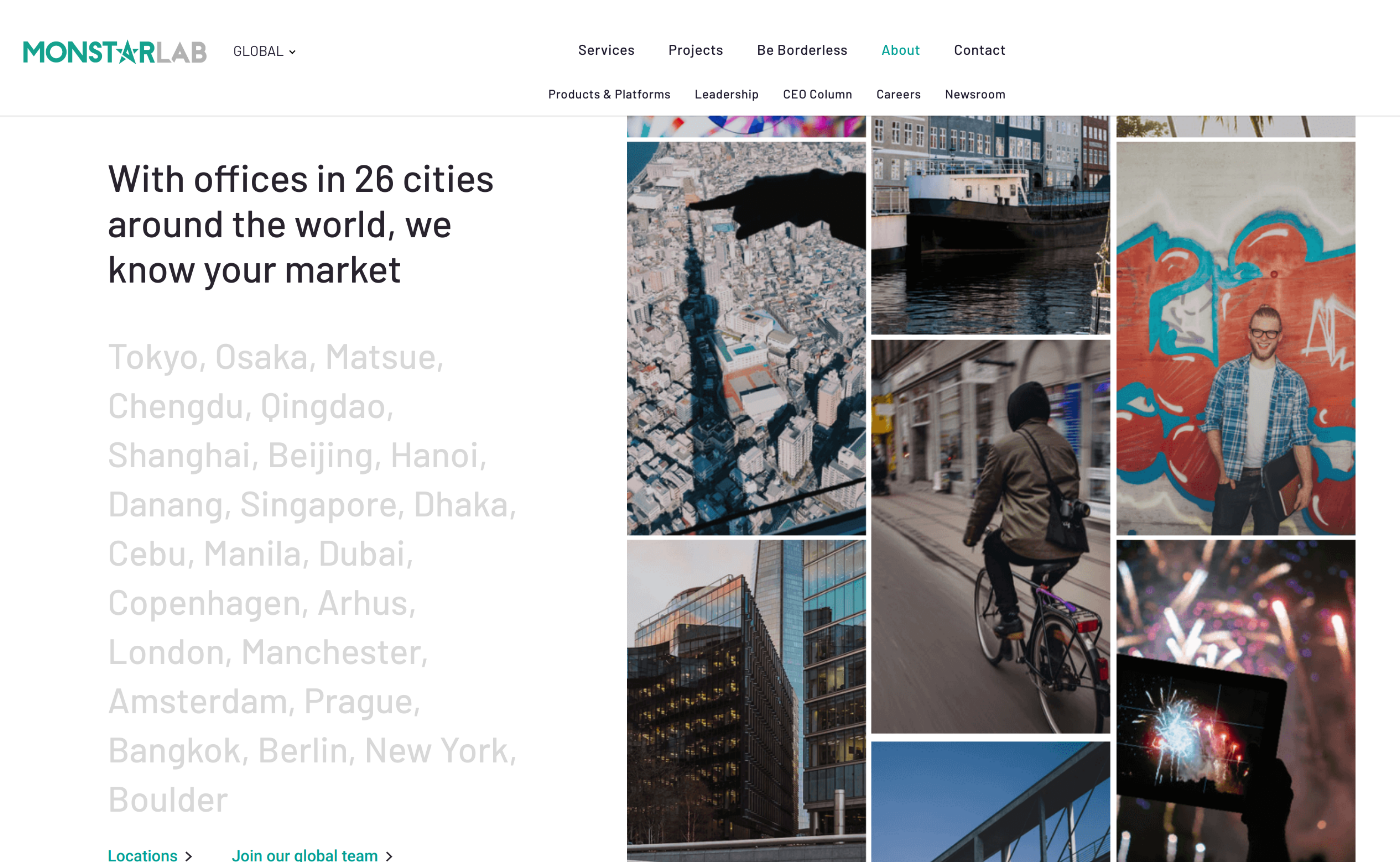
Nov. 7-8, 2019 / Matz江, the Holy City of Ruby + 🍶

RubyWorld Conference 2019

🍶熱燗 sponsored

by Monstar Lab

Monstar Lab



With offices in 26 cities around the world, we know your market

Tokyo, Osaka, Matsue, Chengdu, Qingdao, Shanghai, Beijing, Hanoi, Danang, Singapore, Dhaka, Cebu, Manila, Dubai, Copenhagen, Arhus, London, Manchester, Amsterdam, Prague, Bangkok, Berlin, New York, Boulder

Monstar Lab

We don't have Toyama office, though

Monstar Lab

@noboru_i

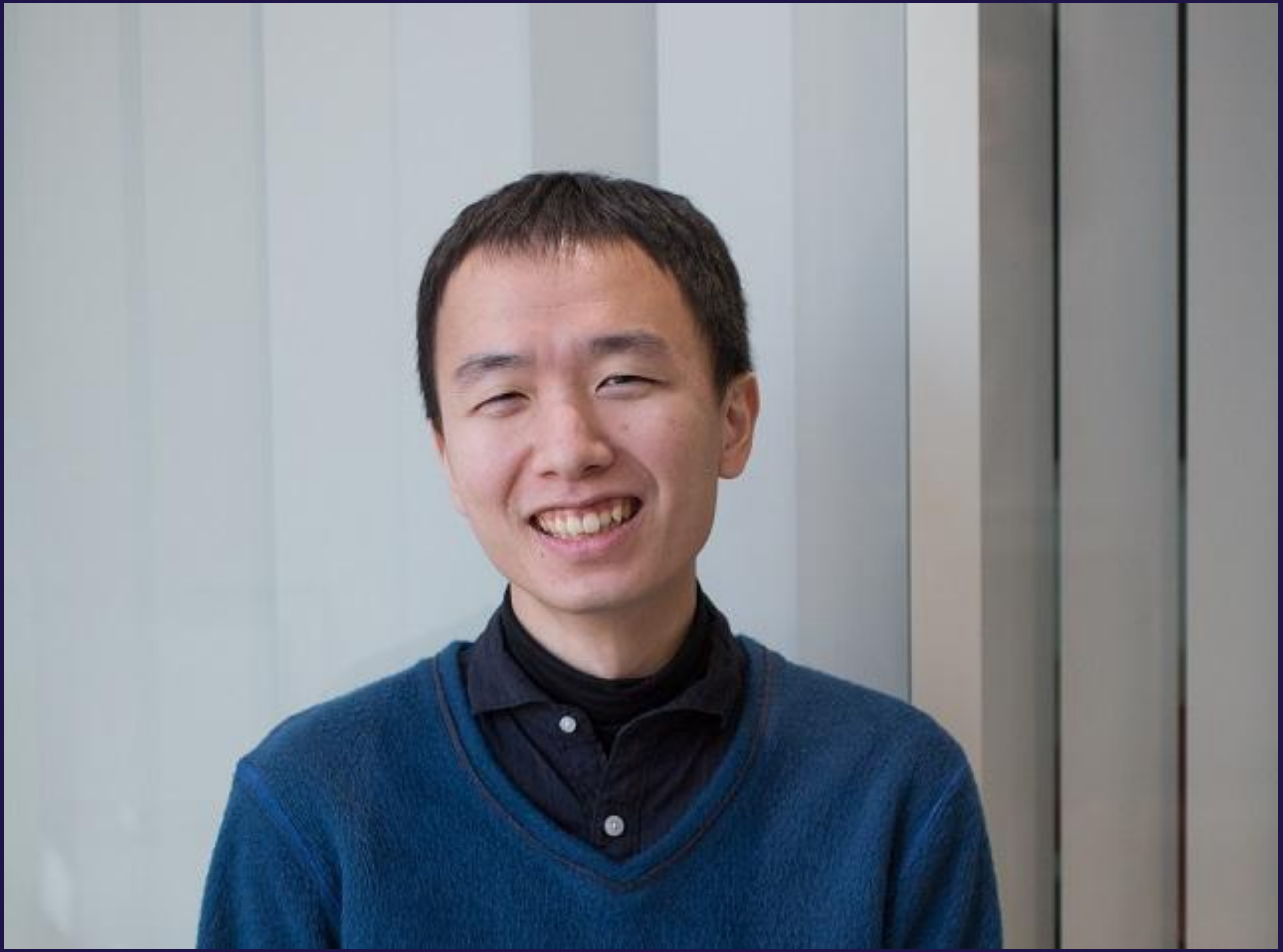
Monstar Lab

石倉神

Monstar Lab

いわゆるゴッド

Monstar Lab

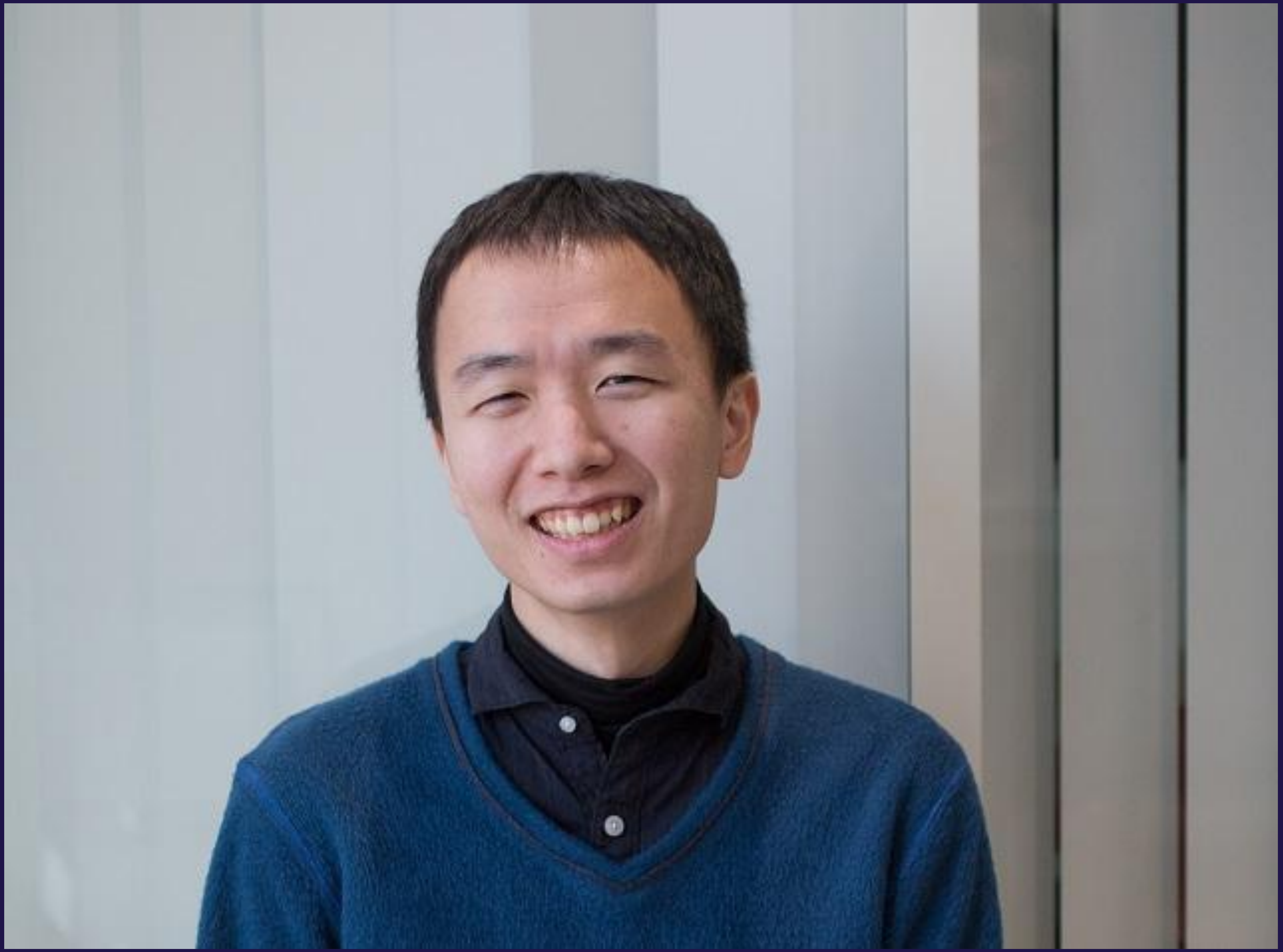


この人です

Monstar Lab

WE ARE HIRING!

Monstar Lab



採用してます

mruby de Hello World!

How to code

mruby de Hello World!

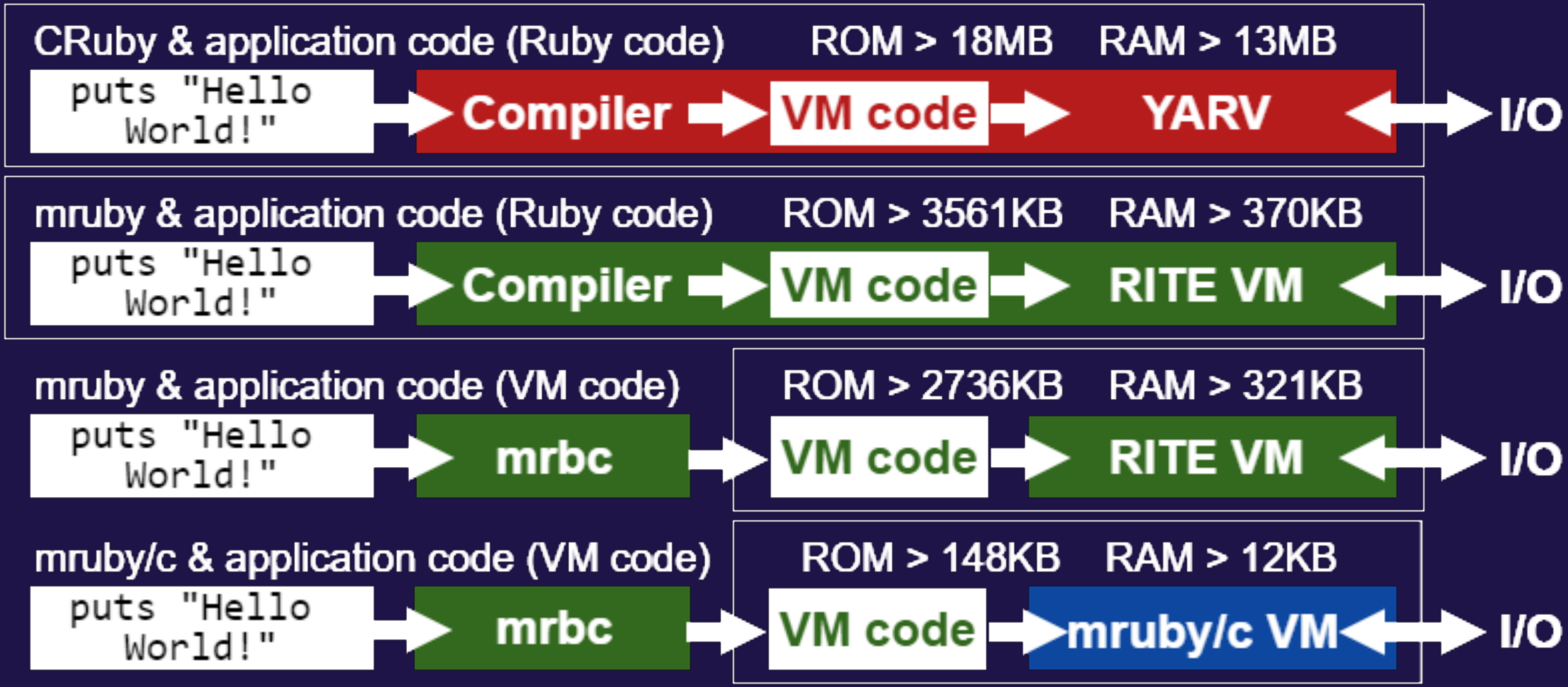
```
5249 5445 3030 3036 9a78 0000 0062 4d41
545a 3030 3030 4952 4550 0000 0044 3030
3032 0000 0060 0001 0004 0000 0000 000c
1001 4f02 002e 0100 0137 0167 0000 0001
0000 0c48 656c 6c6f 2057 6f72 6c64 2100
0000 0100 0470 7574 7300 454e 4400 0000
0008
```

puts "Hello World!"

```
5249 5445 3030 3036 9a78 0000 0062 4d41  RITE0006.x...bMA
545a 3030 3030 4952 4550 0000 0044 3030  TZ0000IREP...D00
3032 0000 0060 0001 0004 0000 0000 000c  02...`..........
1001 4f02 002e 0100 0137 0167 0000 0001  ..O......7.g....
0000 0c48 656c 6c6f 2057 6f72 6c64 2100  ...Hello World!.
0000 0100 0470 7574 7300 454e 4400 0000  .....puts.END...
0008                                     ..
```

VM code

CRuby, mruby and mruby/c



CRuby & application code (Ruby code) ROM > 18MB RAM > 13MB

```
puts "Hello
  World!"
```

Compiler → VM code → YARV ↔ I/O

mruby & application code (Ruby code) ROM > 3561KB RAM > 370KB

```
puts "Hello
  World!"
```

Compiler → VM code → RITE VM ↔ I/O

mruby & application code (VM code) ROM > 2736KB RAM > 321KB

```
puts "Hello
  World!"
```

mrbc → VM code → RITE VM ↔ I/O

mruby/c & application code (VM code) ROM > 148KB RAM > 12KB

```
puts "Hello
  World!"
```

mrbc → VM code → mruby/c VM ↔ I/O

CRuby(2.6.4) + Ruby code

```ruby
puts "Hello World!"
rss = `ps -o rss= -p #{Process.pid}`.to_f / 1024
vsz = `ps -o vsz= -p #{Process.pid}`.to_f / 1024
puts "RSS: #{rss} MB"
puts "VSZ: #{vsz} MB"

# $ ruby hello.rb
# Hello World!
# RSS: 13.63671875 MB
# VSZ: 78.6328125 MB
```

mruby(2.0.1) + Ruby code

```
#include <mruby.h>
#include <mruby/compile.h> // compile at runtime
int main(void) {
  mrb_state *mrb = mrb_open();
  char code[] = "puts 'Hello World!'";
  mrb_load_string(mrb, code);
  mrb_close(mrb);
  return 0;
}
// $ valgrind ./hello_ruby
// (...)
// Hello World!
// ==18802==
// ==18802== HEAP SUMMARY:
// ==18802==     in use at exit: 0 bytes in 0 blocks
// ==18802==   total heap usage: 3,067 allocs, 3,067 frees,
//               379,851 bytes allocated
```

mruby(2.0.1) + VM code

```c
#include <mruby.h>
#include <mruby/irep.h>
#include "hello.c"      // compiled by mrbc
int main(void) {
  mrb_state *mrb = mrb_open();
  mrb_load_irep(mrb, hello);
  mrb_close(mrb);
  return 0;
}
// $ valgrind ./hello_vm
// (...)
// Hello World!
// ==18858==
// ==18858== HEAP SUMMARY:
// ==18858==     in use at exit: 0 bytes in 0 blocks
// ==18858==   total heap usage: 3,057 allocs, 3,057 frees,
//                329,083 bytes allocated
```

mruby/c(2.0) + VM code

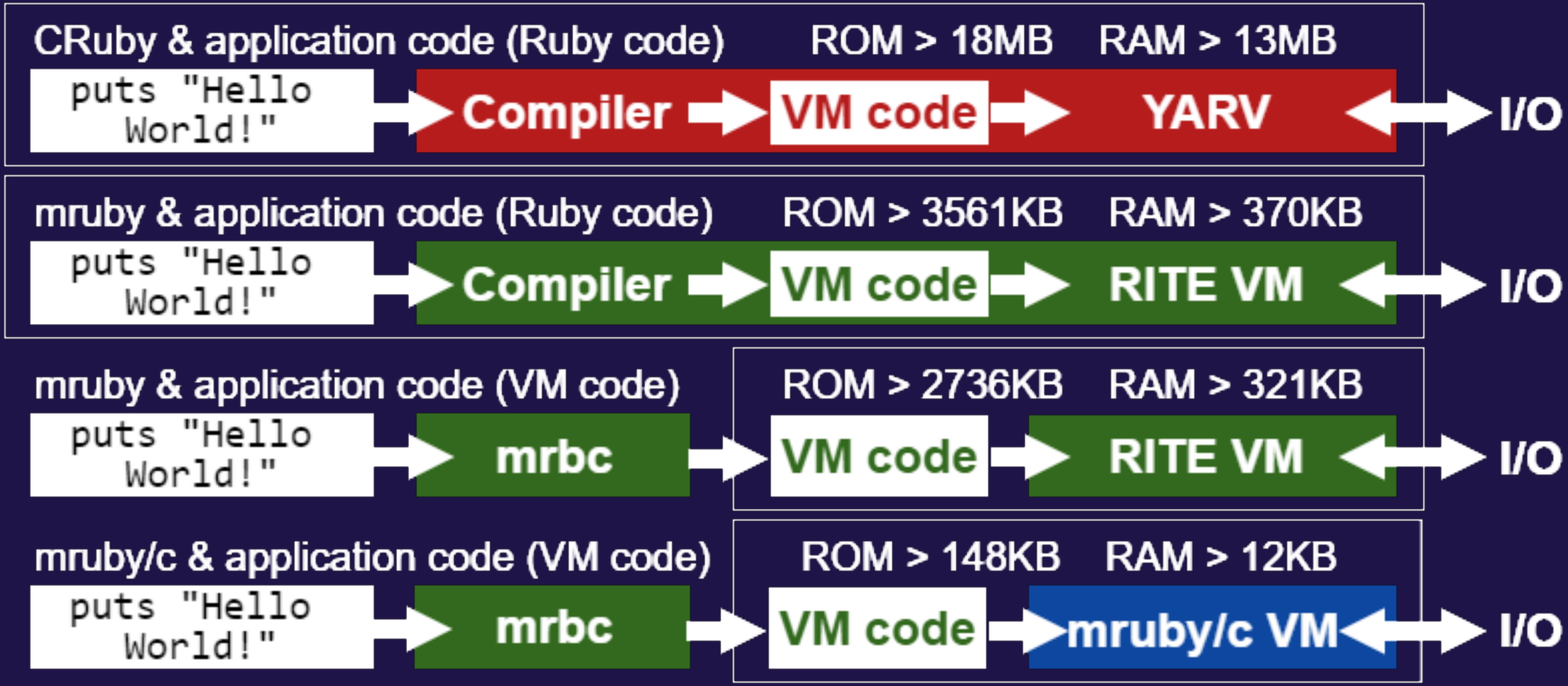
```c
#include ”mrubyc/src/mrubyc.h”
#include ”hello.c”
#define MEMORY_SIZE (1024 * 12) // RAM:12KB
static uint8_t my_memory_pool[MEMORY_SIZE];
int main(void) {
  mrbc_init(my_memory_pool, MEMORY_SIZE);
  mrbc_create_task(hello, 0);
  mrbc_run();
  return 0;
}
```

Hello memory usage

	CRuby + Ruby code	mruby + Ruby code	mruby + VM code	mruby/c + VM code
ROM	18MB(*)	3561KB	2736KB	148KB
RAM	13MB	370KB	321KB	12KB
ROM / RAM	1.3	9.6	8.5	12.3

(*)...binary size of `bin/ruby` itself

CRuby, mruby and mruby/c



CRuby & application code (Ruby code) ROM > 18MB RAM > 13MB

mruby & application code (Ruby code) ROM > 3561KB RAM > 370KB

mruby & application code (VM code) ROM > 2736KB RAM > 321KB

mruby/c & application code (VM code) ROM > 148KB RAM > 12KB

mruby compiler

How to code?

Steps of coding a compiler

- Tokenize (Scan, Lexical analyze)
- Parse
- Generate Code

Not detailed enough 😵

Steps of coding a compiler

- Tokenize (Scan, Lexical analyze)
- Find keywords
- Classify tokens
- Parse
- Make syntax tree
- Make symbol table
- Make literal pool
- Count local variables and registers
- Make each scopes（文字数

Steps of coding a compiler

- Tokenize (Scan, Lexical analyze)
- Parse
- Generate Code

Just outlines for today

mruby compiler written in CRuby

github.com/hasumikin/mmrbc.gem

mruby compiler written in CRuby

github.com/hasumikin/mmrbc.gem

for only `puts "Hello World!"` 😂
`[identifier] "[string literal]"`

mruby compiler written in CRuby

github.com/hasumikin/mmrbc.gem

for only `puts "Hello World!"` 😂
`[identifier] "[string literal]"`

but no cheat 😤

Tokenizer

- FLEX is a tokenizer generator
- You can write tokenizer from scratch w/o FLEX
  - CRuby, mruby and mmrbc.gem, too

```
puts("Hello World!")
....................
p
pu
put
puts
puts(  # look ahead
puts   # detemine a token
```

Tokenizer of Ruby

- It has state

```
$ irb
irb(main):001:0> [1, 2, 3].each do |n|
irb(main):002:1*
```

- `do` keyword sets tokenizer_state as EXPR_BEG
- irb can delay parsing until it becomes EXPR_END with `end` keyword

Parser and Parser generator

- Parser
  - Syntactic analysis of token list
- Parser generator
  - Generates C code of parser by syntactic definition (and "reduction" code)
- Parse algorithms
  - LL(n), LR(n), etc.

Parse algorithm - LL(1)/LR(1)

- LL(k) = Left to right, Leftmost derivation
  - You can write LL parser from scratch
- LR(k) = Left to right, Rightmost derivation
  - You can hardly write LR parser from scratch
  - You should use parser generator
- LALR(k) is a variation of LR
- (k) = length of lookahead symbols

Parser generator

YACC/BISON

YACC/BISON

- Most popular parser genarator
  - Used in CRuby, mruby, bash, Blawn, etc.
- BISON is a GNU version of YACC
  - Thread safe (Reentrant)
- Genarates LALR(1) parser

However

However

I don't use YACC/BISON

So?

So?

LEMON, instead

LEMON?

- Parser generator of SQLite
  - A part of SQLite project
- Generates LALR(1) parser code
  - as well as YACC/BISON does
- Doesn't use global variable to pass information between parser and tokenizer
  - YACC/BISON does
- Tokenizer calls parser in LEMON
  - Parser calls tokenizer in YACC/BISON

Parser calls tokenizer in YACC/ BISON

```
int yyparse(parser_state *p) {
  ...
  yynewstate:
    ...
    yychar = yylex (&yylval, p); // calls tokenizer
    ...
    goto yynewstate;
  ...
}
```

Tokenizer calls parser in LEMON

```c
void Tokenize(char *code) {
  ...
  while (token == get_token(code)) {
    ...
    Parse(parser, token, value); // calls parser
    ...
  }
  Parse(parser, 0, NULL);
}
```

Parsing "Hello World!" in YACC

```
# an excerpt from mruby/mrbgems/mruby-compiler/core/parse.y
primary         : literal
                ¦ string
                (...)
                ;
literal         : numeric
                (...)
                ;
string          : string_fragment;
                ¦ string string_fragment
                    { $$ = concat_string(p, $1, $2); }
                ;
string_fragment : tSTRING_BEG string_rep tSTRING
                    { $$ = new_dstr(p, push($2, $3)); };
string_rep      : string_interp
                ¦ string_rep string_interp
                    { $$ = append($1, $2); };
string_interp   : tSTRING_MIĐ
                    { $$ = list1($1); };
```

Parsing "Hello World!" in LEMON

```
# an excerpt from mmrbc.gem/ext/mmrbc/parse.y
primary            ::= literal.
primary            ::= string.
literal            ::= numeric.
string             ::= string_fragment.
string_fragment(A) ::= STRING_BEG string_rep(C) STRING.
                         { A = new_dstr(p, list3(atom(ATOM_string_add),
                           list1(atom(ATOM_string_content)), C)); }
string_rep         ::= string_interp.
string_rep(A)      ::= string_rep(B) string_interp(C).
                         { A = append(B, C); }
string_interp(A)   ::= STRING_MIÐ(B).
                         { A = list2(atom(ATOM_at_tstring_content),
                           literal(B)); }
```

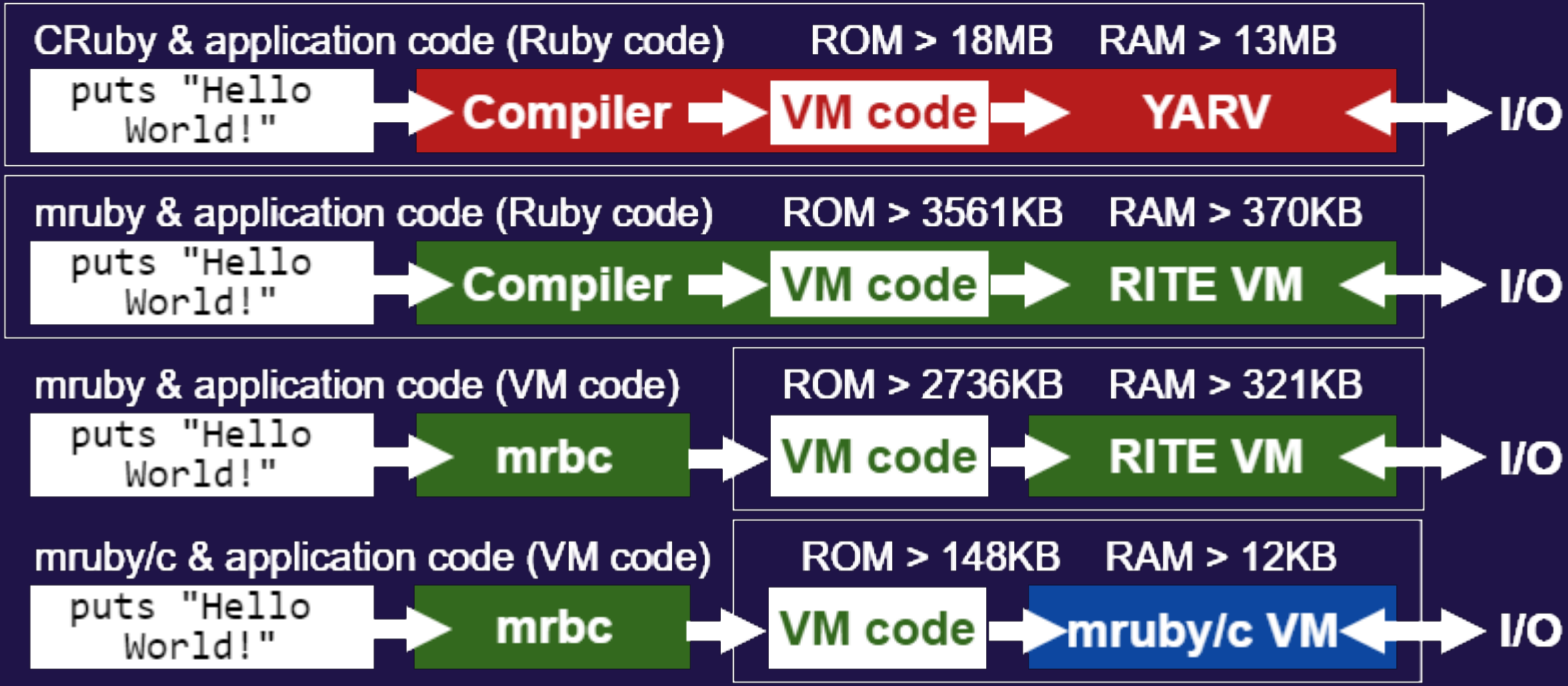
DEMO?

Future work

Future work ->{mmrbc + mruby/c}

- More syntax than `puts "Hello World!"`
- Rewrite mmrbc.gem into C
- And embed it with mruby/c in one-chip microcontroller which has less than 512KB ROM and 128KB RAM

Future work ->{mmrbc + mruby/c}



CRuby & application code (Ruby code) ROM > 18MB RAM > 13MB

puts "Hello World!" → Compiler → VM code → YARV ↔ I/O

mruby & application code (Ruby code) ROM > 3561KB RAM > 370KB

puts "Hello World!" → Compiler → VM code → RITE VM ↔ I/O

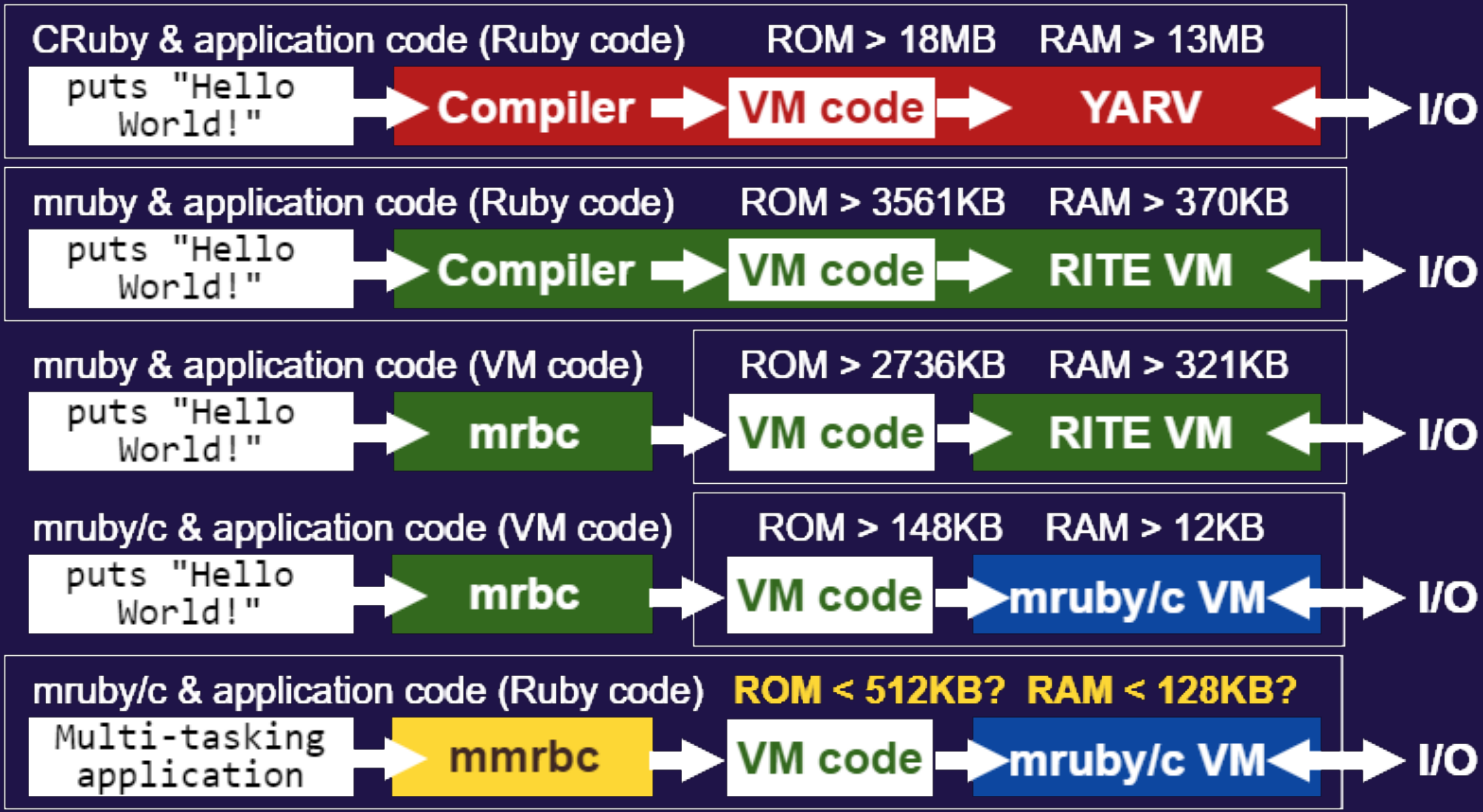
mruby & application code (VM code) ROM > 2736KB RAM > 321KB

puts "Hello World!" → mrbc → VM code → RITE VM ↔ I/O

mruby/c & application code (VM code) ROM > 148KB RAM > 12KB

puts "Hello World!" → mrbc → VM code → mruby/c VM ↔ I/O

Future work ->{mmrbc + mruby/c}



CRuby & application code (Ruby code) ROM > 18MB RAM > 13MB

puts "Hello World!" → Compiler → VM code → YARV ↔ I/O

mruby & application code (Ruby code) ROM > 3561KB RAM > 370KB

puts "Hello World!" → Compiler → VM code → RITE VM ↔ I/O

mruby & application code (VM code) ROM > 2736KB RAM > 321KB

puts "Hello World!" → mrbc → VM code → RITE VM ↔ I/O

mruby/c & application code (VM code) ROM > 148KB RAM > 12KB

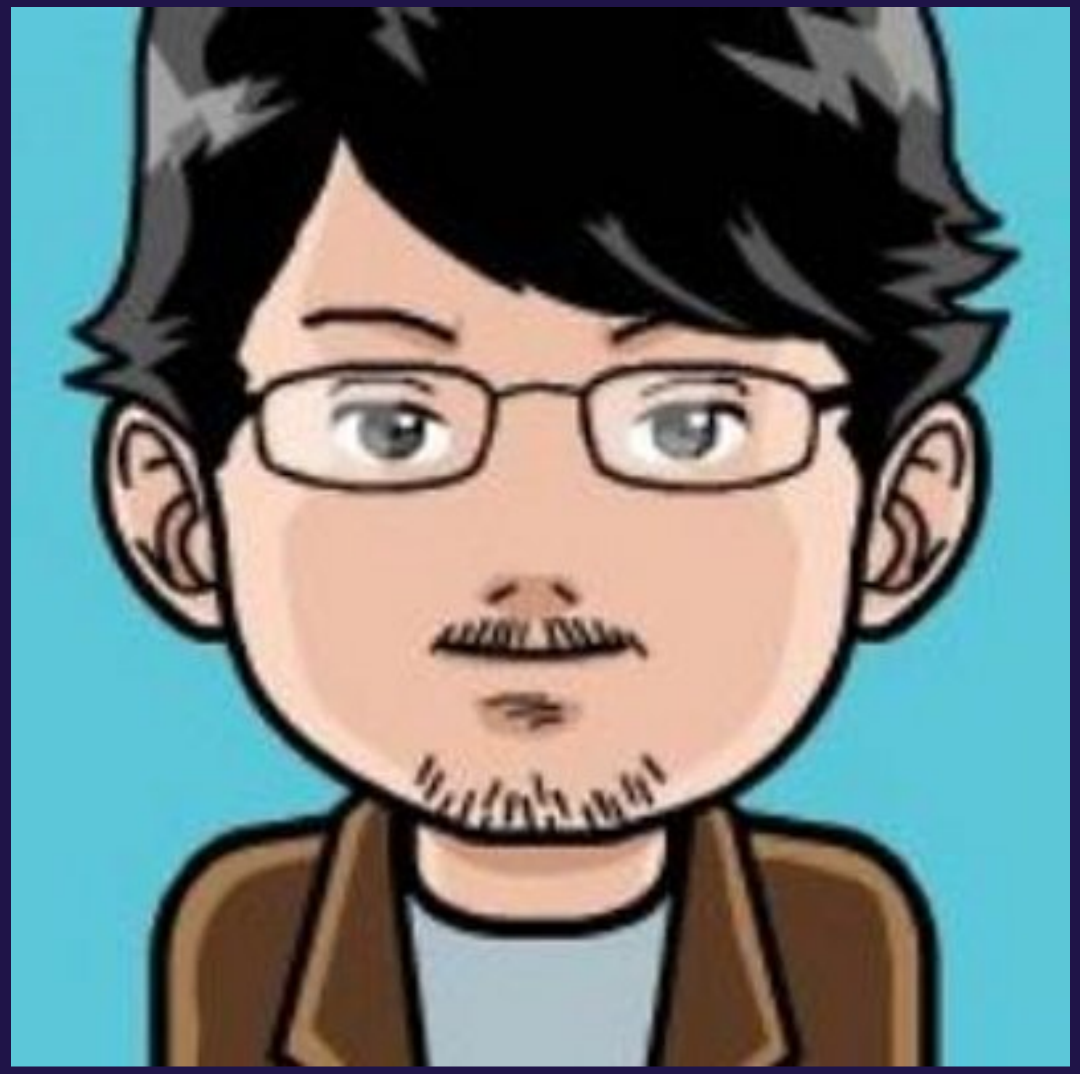
puts "Hello World!" → mrbc → VM code → mruby/c VM ↔ I/O

mruby/c & application code (Ruby code) ROM < 512KB? RAM < 128KB?

Multi-tasking application → mmrbc → VM code → mruby/c VM ↔ I/O

Future work ->{mmrbc + mruby/c}

"LEMON would generate smaller binary than YACC."



Future work ->{mmrbc + mruby/c}

Possibly and hopefully,
I will see you on
RubyKaigi 2020 at Matsumoto

Thank you!