

Ask Ko1

YARV

Yet Another RubyVM

SASADA Koichi

Tokyo University of Agriculture and Technology

Nihon Ruby no Kai

Ko1 に聞け

YARV

Yet Another RubyVM

ささだ こういち
東京農工大学大学院
日本Rubyの会

Agenda

- From Japanese Rubyist
- About me
- About YARV
 - Background
 - Design overview
 - Implementation
 - Current status

Messages from Japanese Rubyists

- From Matz (Mr. Matsumoto)
 - よろしく。来年は会いたいね
 - Regards. I hope to meet you next year.
- From eban (Mr. Watanabe)
 - Learn Japanese
- From NaHi (Mr. Nakamura)
 - Don't pick a fight.

Q. Who are you?

A. Self Introduction

- A Student for Ph.D. 1st degree
 - Systems Software for Multithreaded Arch.
 - SMT / CMP or other technologies
 - i.e.: Hyper threading (Intel), CMT (Sun)
 - OS, Library, Compiler and Interpreter
 - YARV is first step for Parallel interpreter (?)
 - Computer Architecture for Next Generation

At Public Position

A. Self Introduction (cont.)

- Nihon Ruby no Kai
 - Rubyist community in Japan founded on 8th Aug.
 - Organized by Mr. Takahashi (maki)
- Rubyist Magazine (10th Sep.)
 - <http://jp.rubyst.net/magazine>
- Favorite method
 - `__send__`
 - Recently, Japanese rubyist must say it

A. Self Introduction (cont.)

- Ko1?
 - “Koichi” → 「こういち」 → 「耕一」
 - 「一」 means 1 in Japanese
- Works (part time job)
 - Kahua: Application Server framework in Scheme
 - Coins: a COmpiler INfra Structure
 - Compiler for some languages in Java
 - I don't like Java language. But eclipse is quite good (If I have a high performance machine)

A. Self Introduction (cont.)

- Software
 - Rava: A JavaVM in Ruby
 - My best joke software
 - Rucheme: A Scheme interpreter in Ruby
 - I like scheme
 - Nadoka: IRC Client Server software
 - Ruby's killer application (... only in my head)

A. Self Introduction (cont.)

- Home page
 - <http://www.namikilab.tuat.ac.jp/~sasada/>
 - Of course in Japanese :)
- Organize poll about some topics
 - Like that...

Q. Why do you make YARV?

A. Project Background

- Ruby - Object Oriented Scripting Language
 - Very easy to use, but still powerful
 - Used world-wide
 - From Japan (This words for Japanese public officer)
- But... Current Ruby interpreter is slow
 - Traverse Abstract Syntax Tree in runtime
 - Because ruby is too slow to use it, some projects should choose other language (like Java)
 - And everyone says “Ruby? Ah, slow language” (myth)

A. Project Background (cont.)

- Byte code Interpreter seems to be good
 - Like Lisp, Java, .Net, Smalltalk, ...
- Some bytecode interpreter → not enough
 - Matz' try → incomplete
 - ByteCodeRuby (Mr. Marrows) → Slow (old Info?)
 - And other incomplete interpreters

Q. What is YARV?

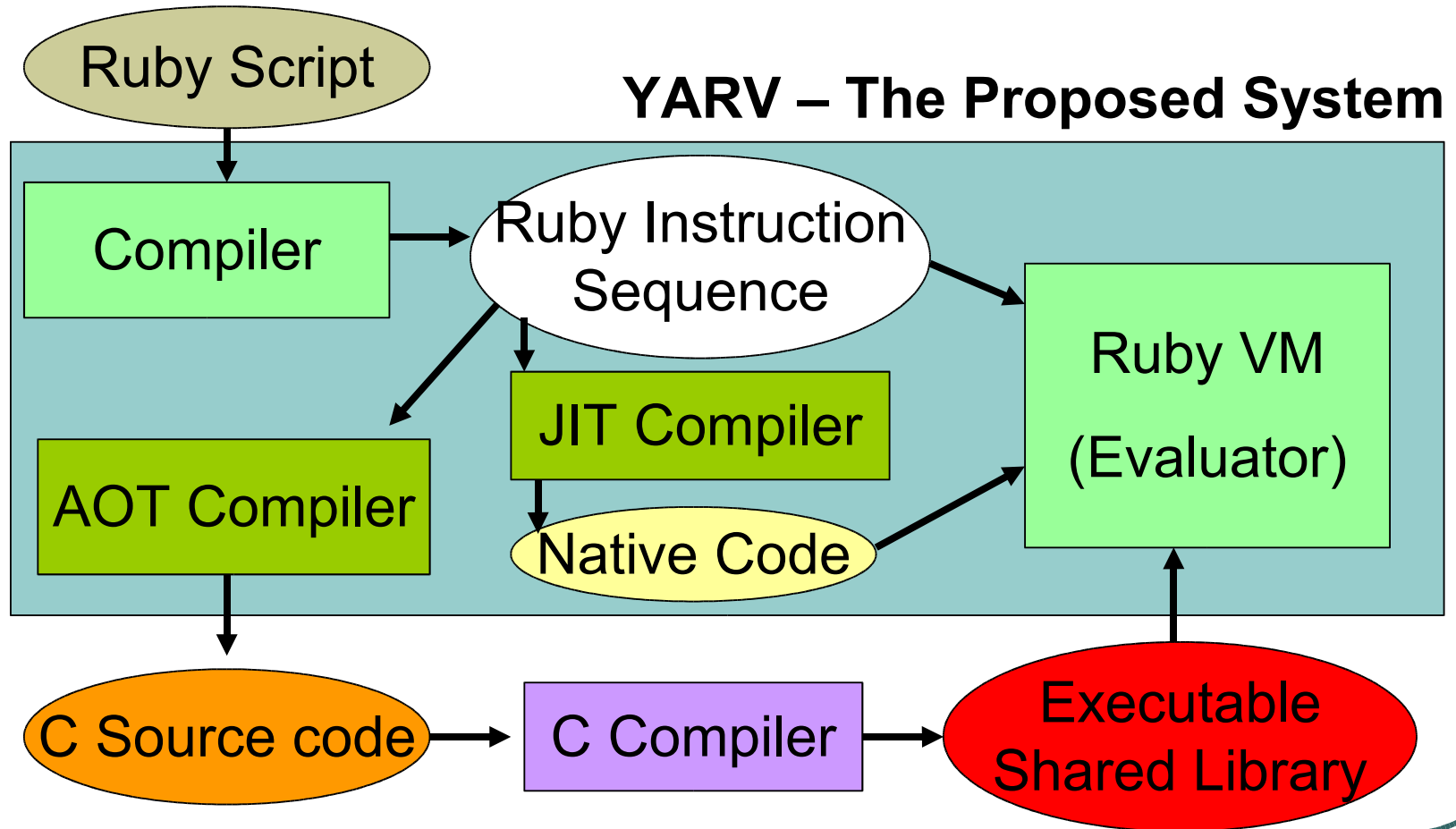
A. Project overview

- Creating Ruby VM
- Accepted by “Exploratory Software Development” – “Exploratory youth”
 - 未踏ソフトウェア創造事業 – 未踏ユース
 - By IPA – Information-technology Promotion Agency, Japan.
 - Another Ruby Project is accepted in this year
 - Ruby Compiler for .Net by Mr. Asakawa

A. Project overview (cont.)

- VM Instruction (insn.) set “design”
- Compiler design and implementation(impl.)
- VM design and impl.
- JIT (Just In Time) and AOT (Ahead Of Time) compiler design and impl.
- Published as Open Source Software

A. Project overview (cont.)



Q. What's the goal of YARV?

A. Goal of YARV

- To be Rite
 - If YARV accomplished (Matz promised)
- To be Fastest RubyVM in the world
 - Now, rival is only current ruby interpreter :)
- To enable all Ruby features
 - Very important to call “RubyVM”
 - It's easy to make “Ruby Subset VM”
 - ... but is it really Ruby?

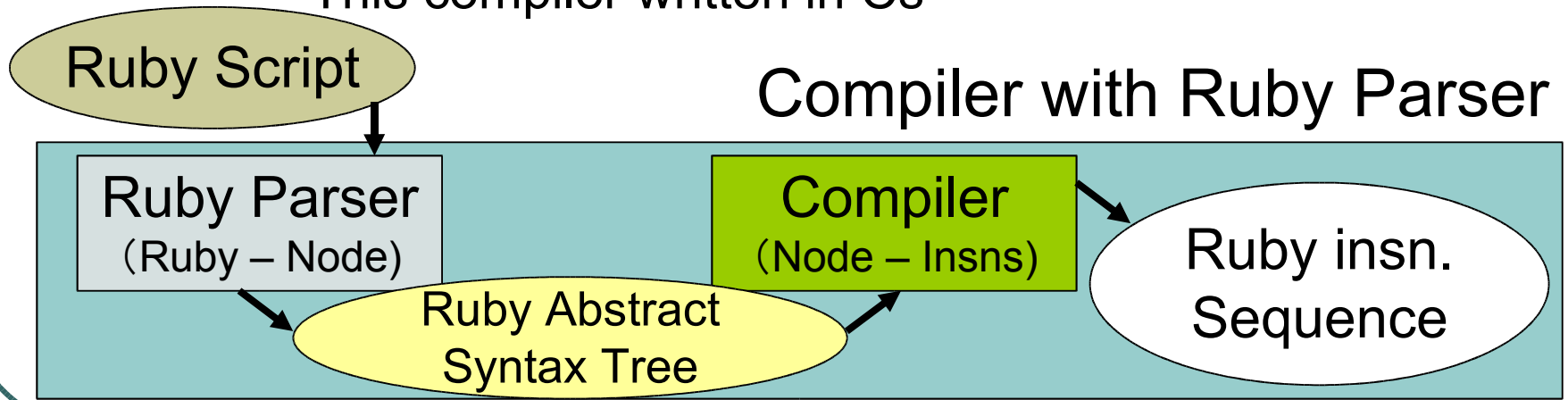
Q. How to implement YARV?

A. Development Policy

- Simple stack machine
- YARV Implemented as Ruby C Extension
- Not “Bytecode” but “Wordcode”
- Use Ruby’s existing Infrastructure
 - GC
 - Ruby Script parser
 - Ruby API is very useful to C programming
 - i.e) Memory Management
 - using Array without “free()” is very happy

A. Development Policy

- Compiler parse Ruby's AST
 - Ruby Script Parser creates Node tree
 - Traverse this tree and translate to YARV instructions (insns)
 - This compiler written in Cs



Q. How to implement YARV?(2)

A. Implementation - Registers

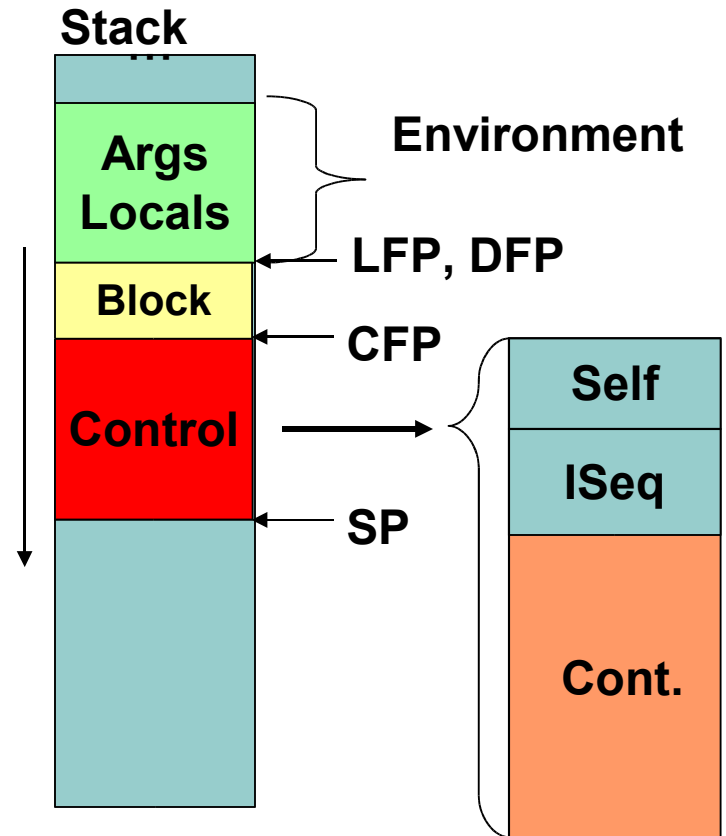
- 5 registers
 - PC: Program Counter
 - SP: Stack Pointer
 - LFP: Local Frame Pointer
 - DFP: Dynamic Frame Pointer
 - CFP: Control Frame Pointer

A. Implementation - Frames

- Some frames
 - Method Frame
 - Block Frame
 - Class Frame

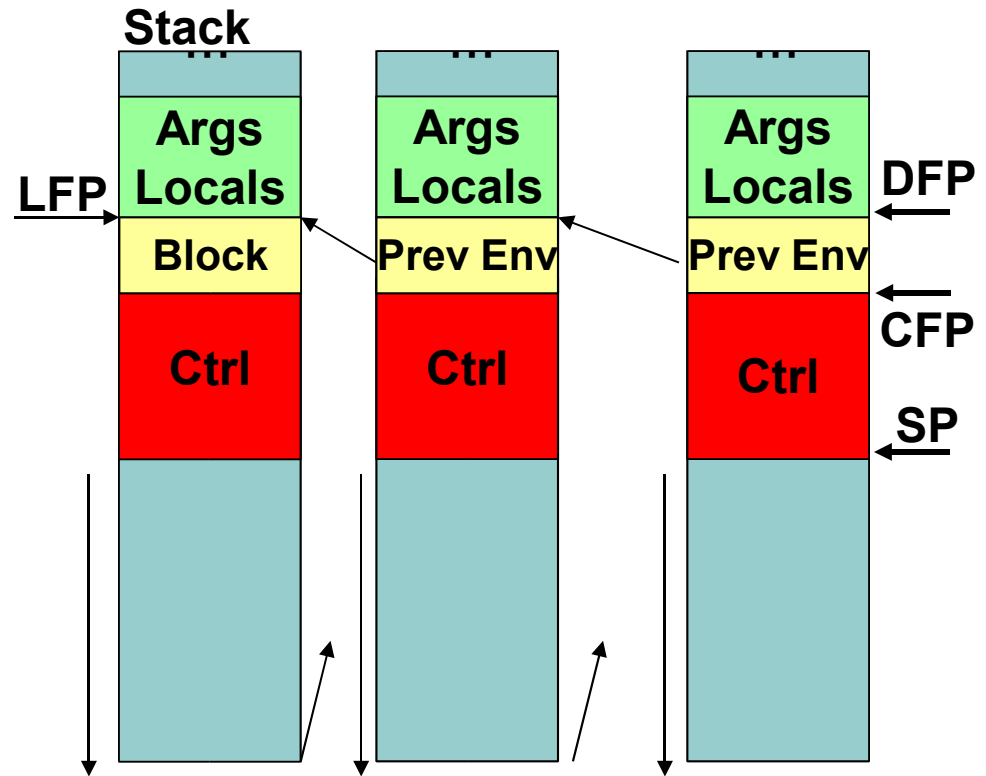
A. Implementation - Frames (.cont)

- Method Frame
 - Same as other VM
 - Class frame is same
- Control frame
 - Every frame has this
 - Includes “self”,
instruction sequence information,
continuation(keep last regs)
 - `CFP[0] == self`



A. Implementation - Frames (cont.)

- Block Frame
 - Created when 'yield'
 - LFP point method local environment
 - DFP point current environment
 - DFP[0] point previous environment

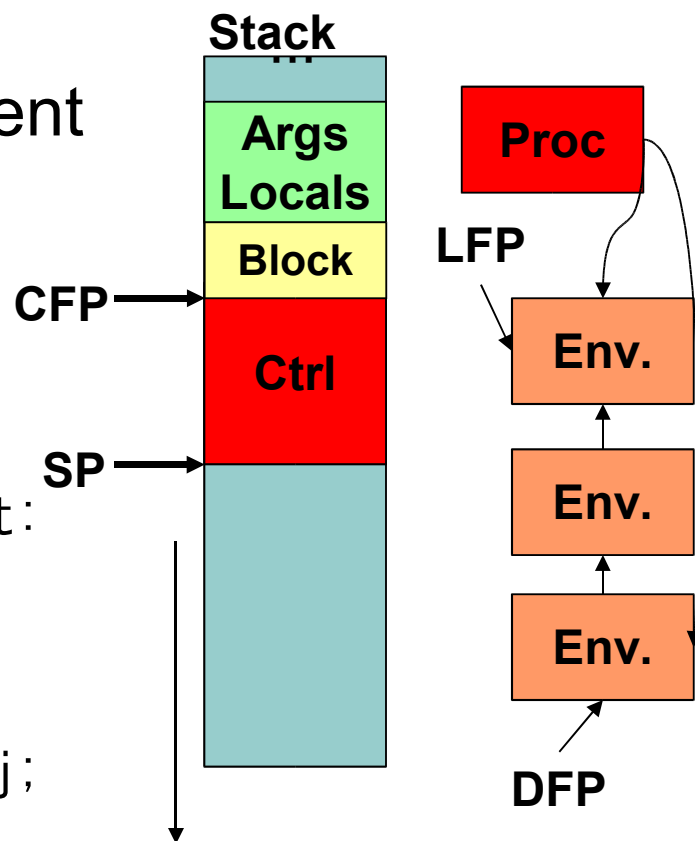


A. Implementation - Proc

- Creating Proc Object
 - Proc enables indefinite extent
 - Environments to heap
 - LFP and DFP point env. in heap

```
# Proc sample
def m arg; x = 0
  iter{|a| i=1
    iter{|b| j=2
      Proc.new
    }; end
  }; end
```

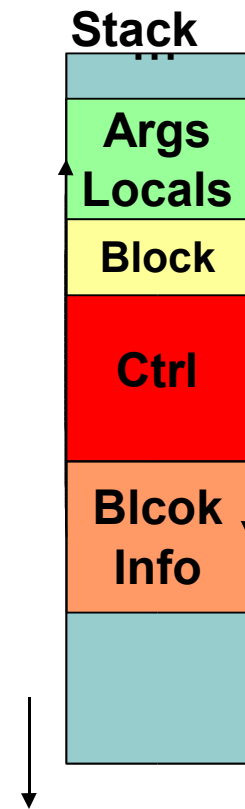
```
Struct ProcObject:
  VALUE self;
  VALUE *lfp;
  VALUE *dfp
  VALUE iseqobj;
```



A. Implementation - Block

- Block is stacked
 - Like 'alloca()' area allocation
 - Use by 'yield' insn.

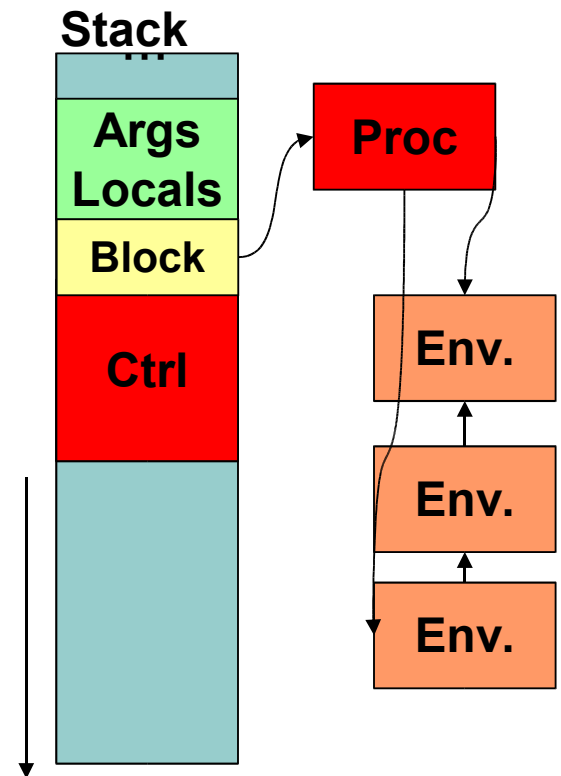
```
# Block sample  Struct
iter {
  ...
}
BlockObject:
  VALUE self;
  VALUE *lfp;
  VALUE *dfp;
  VALUE iseqobj;
```



A. Implementation - Block (Proc)

- Proc is stacked too
 - Used by 'yield' insn.
 - Same data structure as Proc
 - Can treat as Block object

```
# Proc sample      # cont.
def m arg; x = 0
  iter{|a| i=1      iter(m(arg))
    iter{|b| j=2
      Proc.new
    }}; end
```



A. Implementation

Exception / Jump

- Use exception table to handle
 - Like JavaVM
 - Some entries
 - Rescue clause
 - Ensure clause
 - Retry point
 - Each entries have
 - PC range
 - Continuation PC and SP
- If jump occurred, rewind stack and check this table

A. Implementation

Exception / Jump (cont.)

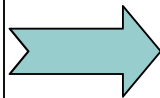
- Different from Java and usually VM
 - Must manage continuation SP register

```
# Java can do this?  
v = begin  
    F00  
    rescue  
    BAR  
ensure  
    BAZ  
end
```

A. Implementation - Ensure

- If no error/jump occurred: ensure is done by normal instruction flow (copied / like recent Java compiler)

```
# sample  
begin  
  A  
ensure  
  B  
end
```



```
Compiled:  
  Start_A:  
    A  
  End_A:  
    B  
  End_B:  
end
```

```
ExceptionTable:  
entry:  
  type: ensure  
  range: Start_A - End_A  
  do: B  
  restart point: End_B  
  restart sp: 0
```

Q. What Insn does YARV has?

A. Insn Category Lists

- **Insn names are not abbreviated**
- Stack control
 - Swap, dup, ...
- Accessor
 - get/setlocal, get/setconstant, ...
- Put something
 - putobject, putnil, putarray, ...
- Apply some change to object
 - concatstrings, ...

A. Insn Category Lists

- Method definition
 - methoddef
- Method call, Class/Module def
 - send, classdef, moduledef, yield, super, ...
- Control flow
 - goto, if, unless, throw
- Some for optimization
 - get/setinlinecache, opt_plus, opt_..., ...
- And others

Q. How to write each insn?

A. Insn Description Language

- Instruction Description Language
 - Body is written in C
 - Declare variables
 - Operands
 - Stack popped and pushed values
 - Parsed by Ruby
 - This scheme enables to flexible VM creation
 - Apply some optimization techs
 - Insert debug print
 - Make document automatically (like RubyDoc)

Q. Does YARV have optimizer?

A. YARV Optimization Tech.

- Inline cache
 - Set for “VM state version” counter
 - It's incremented when some side-effect change
 - (Re)Definition of Constant
 - (Re)Definition of Method
 - Cache some values with this count
 - If kept counter equals current counter you can use cached value
 - This scheme is used by Constant access and method search

A. YARV Optimization Tech. (cont.)

- Inline cache (cont.)
 - Constant access needs some insns
 - A::B::C needs 4 insns
 - Using inline cache achieve to short to 1 insn
 - Method search
 - Current Ruby using Global method cache (works good)
 - I'll try to use this scheme and measure effect

A. YARV Optimization Tech. (cont.)

- Stack caching
 - 2 level stack caching
 - Cache 2 stack top values
 - Using insn description make it automatically

A. YARV Optimization Tech. (cont.)

- Super instructions
 - Union some insns
 - Insns sequence frequently appeared
- Make Special instruction
 - le: putobject true → puttrue
- These techs are effective
 - Apply powerful C compiler optimization
 - I want to make these automatically by brows statistics data. But difficult?

A. YARV Optimization Tech. (cont.)

- JIT compile
 - I'm searching easy way
 - Using existing libraries
 - Using copy code technique
 - Compile in C, and copy with label information
 - Seems difficult or bother

A. YARV Optimization Tech. (cont.)

- AOT compile
 - Substitute insn to C implementation code and compile with C compiler
 - Description language will helps this
 - Easy and C compiler optimization is powerful
 - Output will be normal C extension method
 - Very very simple exam shows faster x100

Q. Does YARV move?

A. Current Status

- Variables
 - Method local, block local, global, instance, Constants, ...
- Class/Module definition
- Control flow
 - if/unless, while/until, case/when
 - begin/rescue/ensure, return, break/retry/next/redo
- Method invocation and yield block
 - Call and yield with some arguments
 - Call with block

Q. What's the limitation on YARV?

A. Follows

- Ruby call from C method
 - It's mean that “10000.times{ ... }” doesn't move
 - To enable that, I must patch ruby/eval.c
- Some useful ruby schemes
 - Stack trace, set_trace_func, method_missing, call with optional/with initials call, Proc as block, method visibility check, ...
 - And many many schemes :-P

Q. How fast YARV work?

A. Benchmark result

- Everyone love it :)
- I'll show you at conference

Q. Why Original System?

A. Comparing to other systems

- v.s. JavaVM, .Net
 - They have very nice library and optimizer
 - Totally, mapping to Ruby spec to VM's models
 - Trade off between optimizer and ruby stub
 - **Is it fun?**
- v.s. Parrot
 - Register model VM really fast in “Interpreter”?
 - **Is it fun?**

Q. What are the rest tasks?

A. Future Work

- Implement Ruby specification
- Implement Optimizers
 - JIT/AOT compiler
 - Implement other interesting optimize tech.
- Collect benchmark program

Q. How to join YARV development?

A. YARV Development community

- Home page
 - <http://www.atdot.net/yarv/>
 - Install instructions and some information
- Mailing list
 - Yarv-dev (in Japanese)
 - Yarv-devel (in English) ... no one using 😊

Q. Finish?

A. Yes.

- Thank you
- Special Thanks
 - Alexander Kellett, Sanimir Agovic
 - Ruby-talk, Yarv-dev subscriber
 - Matz and other rubyists
 - IPA (my sponsor :))
- Any other questions? Please “Ask Ko1” :)