YARV
Yet Another RubyVM

SASADA Koichi

Tokyo University of Agriculture and Technology

Nihon Ruby no Kai

Ko1@atdot.net

RubyConf2004 Oct. 2

Ask Ko1
YARV
Yet Another RubyVM

SASADA Koichi

Tokyo University of Agriculture and Technology

Nihon Ruby no Kai

Ko1@atdot.net

RubyConf2004 Oct. 2

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
Ko1@atdot.net

RubyConf2004 Oct. 2

Caution!

- I can’t use English well
  - If I say strange English, you can see the slide page
    - This slide checked by other guys ☺
  - If you have any question, ask me with:
    - Japanese (recommended)
    - Ruby, C, Scheme, Java, …
    - IRC (@freenode#rubyconf)
    - Easy English (less than 10 words, only easy words)

Agenda

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

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 my 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
- Ruby-dev summary
- Favorite method
  - __send__
  - Recently, Japanese rubyist must say so

A. Self Introduction (cont.)

- Ko1?
  - “Koichi” →「こういち」→「耕一」
  - 「一」means “one” in Japanese
- Works (part time job)
  - Kahua: Application Server framework in Scheme
  - Coins: a Compiler Infrastructure
    - A compiler framework languages in Java supporting various language
    - 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
    - Compile to instruction set which I designed
    - I like Scheme
  - Nadoka: IRC Client Server software
    - IRC proxy/bouncer
    - Ruby's killer application (… nobody else may agree)

A. Self Introduction (cont.)

- Home page
  - http://www.namikilab.tuat.ac.jp/~sasada/
  - Of course in Japanese :)
- Organize polls on many topics

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 (to make my sponsor (officials) happy)
- But… Current Ruby interpreter is slow
  - Traverse Abstract Syntax Tree in runtime
  - Some projects chose other languages (e.g. Java) because Ruby was just too slow for them
  - And everyone says “Ruby? Ah, slow language” (myth)

A. Project Background (cont.)

- Bytecode Interpreter seems to be good
  - Like Lisp, Java, .Net, Smalltalk, …
- Existing bytecode interpreter for ruby

  → Not enough
  - Matz' try → incomplete
  - ByteCodeRuby (Mr. Marrows) → Slow (old info?)
  - And other incomplete interpreters

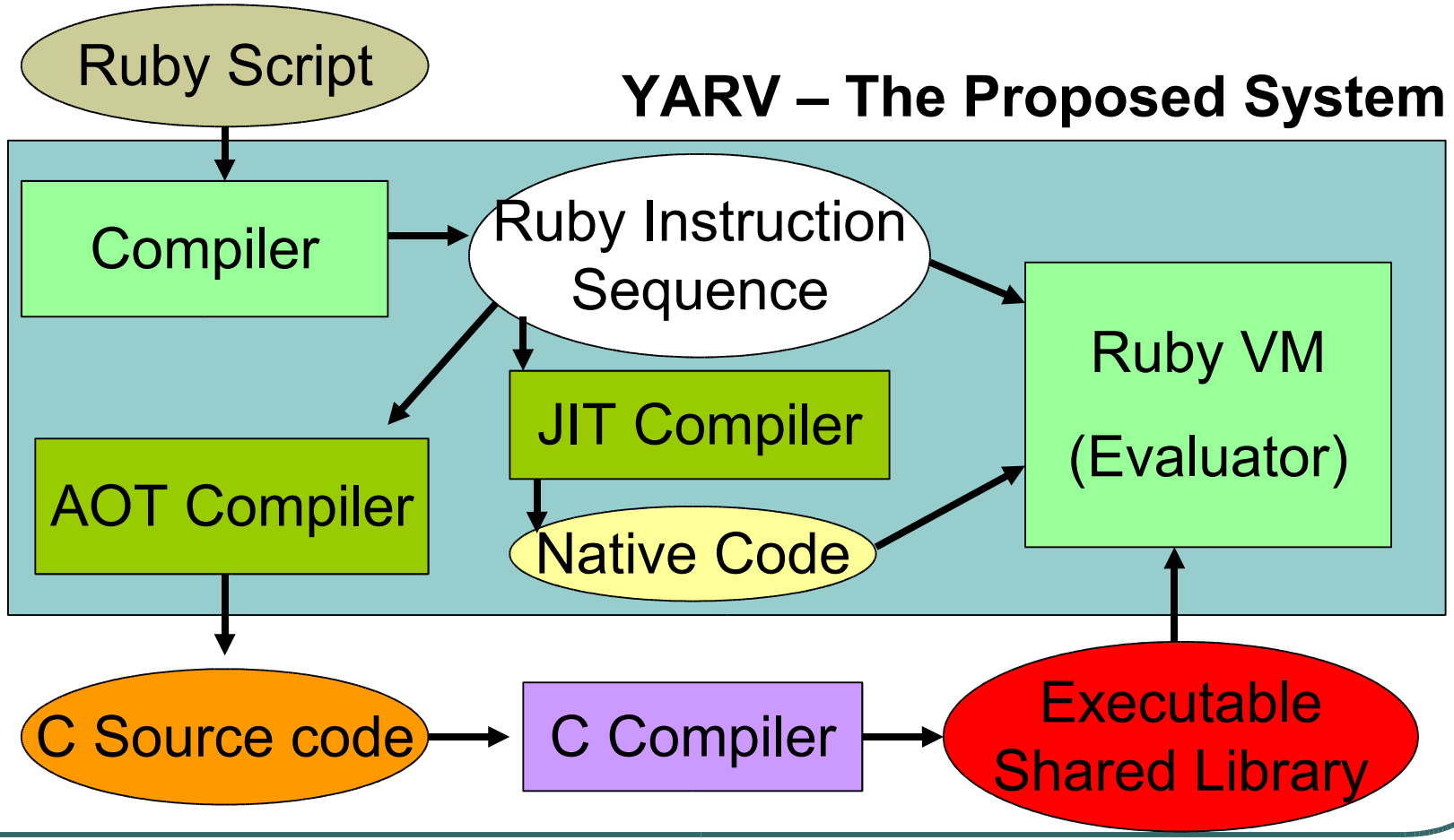
Q. What is YARV?
A. Project overview

- Creating a Ruby VM
- Funded 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.
- Will be published as Open Source Software
- http://www.atdot.net/yarv/

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 'the Fastest RubyVM in the world'
  - Now, rival is only current ruby interpreter :)
- To enable all Ruby features
  - Very important for claiming "RubyVM" name
  - It's easy to make "Ruby Subset VM"
  - … but is it really Ruby?

Q. How to pronounce “YARV”?
A. I say like that, but...

- “Name is important” (Matz)
- But “YARV” name is not important
- Because “YARV” will become “Rite”

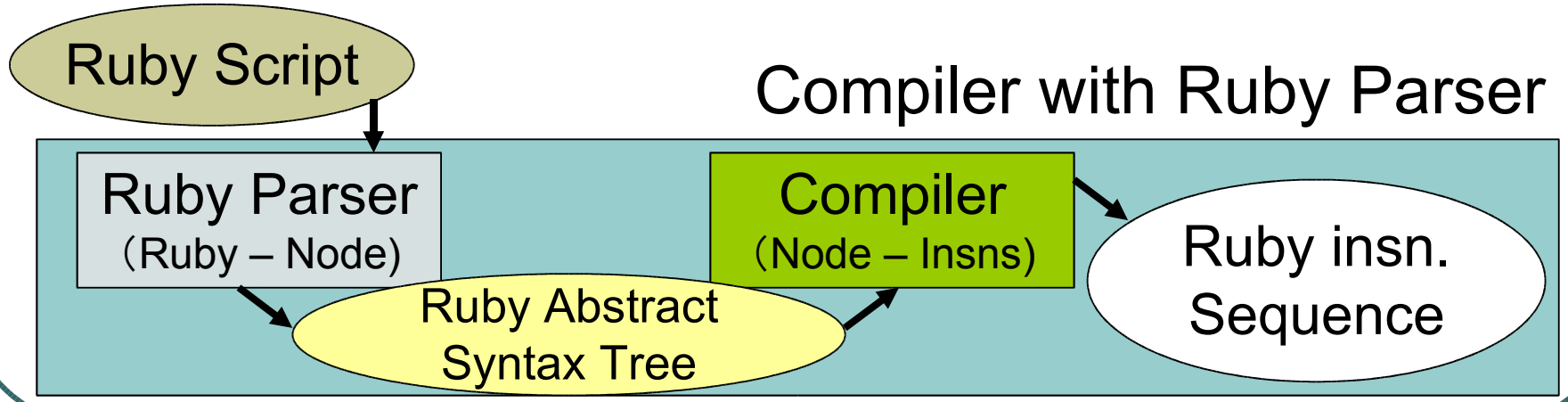
  … if this project succeed
- If failed, no one remember “YARV”
- You can call “YARV” at your pleasure

Q. How to implement YARV?
A. Development Policy

- Simple stack machine
- YARV Implemented as Ruby C Extension
- Not “Bytecode” but “Wordcode”
  - Easy to access from Processor
- Use Ruby’s existing Infrastructure
  - GC
  - Ruby Script parser
  - Ruby API is very useful in 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 is written in C



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

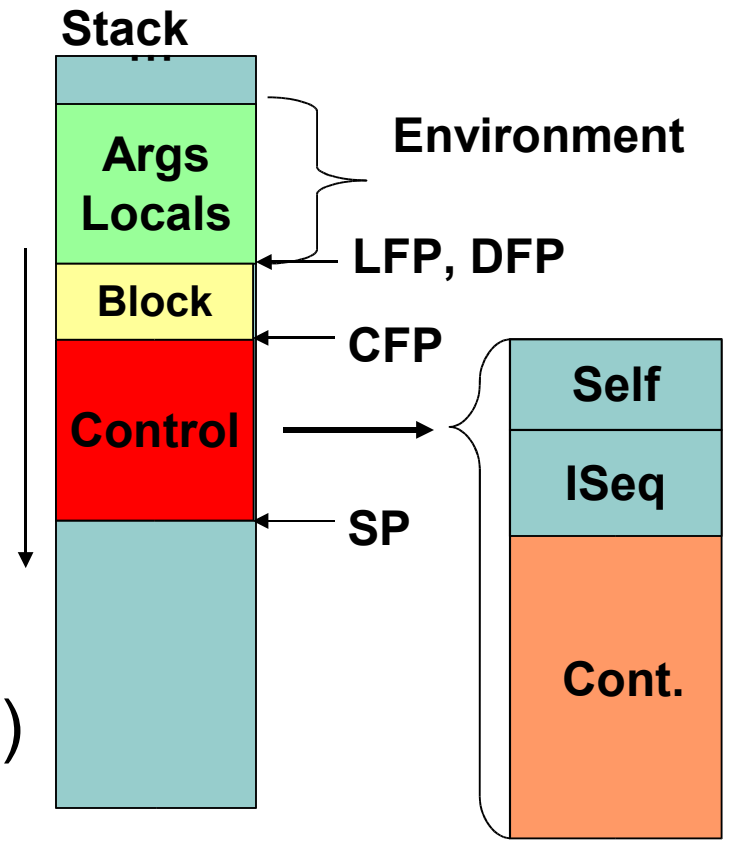
- Frame types
  - Method Frame
  - Block Frame
  - Class Frame
- Save environment to stack

A. Implementation - Frames (.cont)

- Method Frame
  - Same as other VMs
  - Identical to Class Frame
- Control frame
  - Every frame has this
  - Includes “self”,

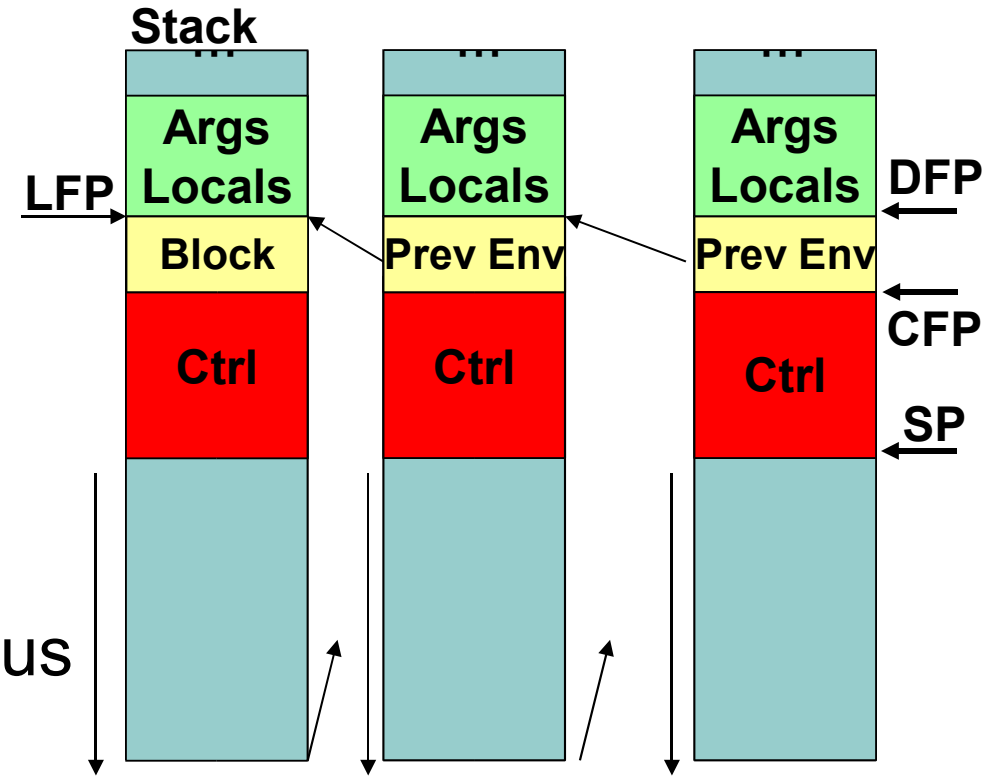
    instruction sequence information,

    continuation(keep last regs)
  - CFP[0] == self



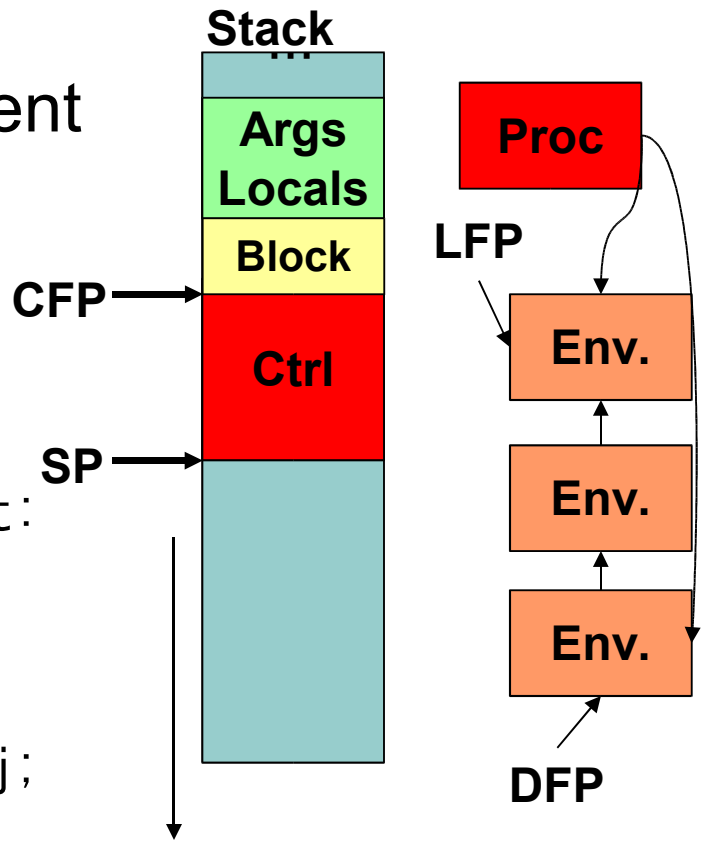
A. Implementation - Frames (cont.)

- Block Frame
  - Created when ‘yield’
  - LFP points to method local environment
  - DFP point to current environment
  - DFP[0] point to previous environment



A. Implementation - Proc

- Creating Proc Object
  - Proc enables indefinite extent
    - Moving environment 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
```

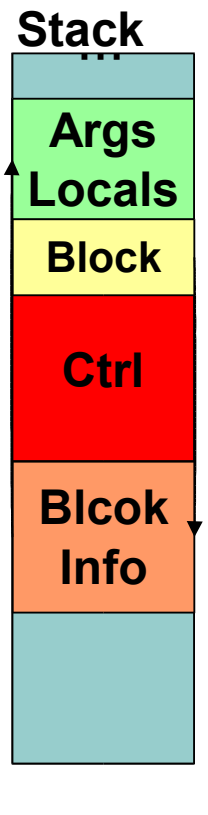
```
Struct ProcObject:
    VALUE self;
    VALUE *lfp;
    VALUE *dfp
    VALUE iseqobj;
```

A. Implementation - Block

- Blocks are pushed on stack
  - A Block body is allocated by area allocation like “alloca()”
  - Used by ‘yield’ insn.

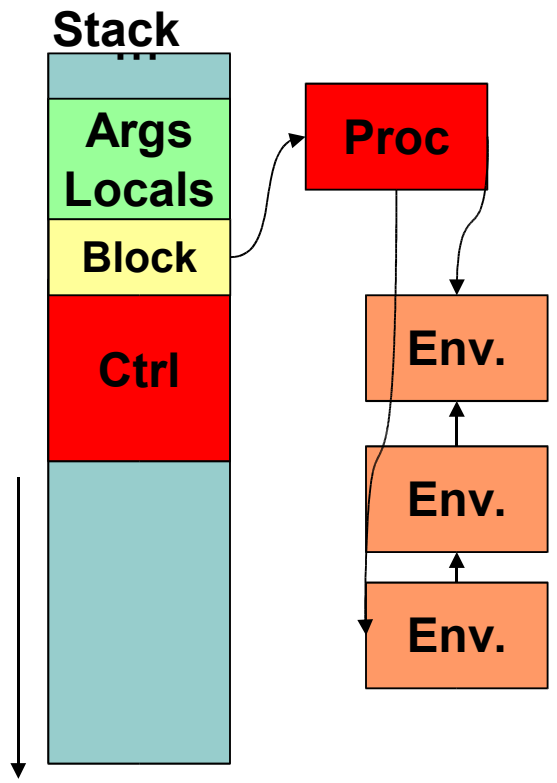
```
# Block sample
iter{
  ...
}
```

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



A. Implementation - Block (Proc)

- Procs are pushed on stack
  - Used by ‘yield’ insn.
  - Same data structure as Proc
  - Can treat as Block object




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

```
# cont.
iter(m(arg))
```

A. Implementation Exception / Jump

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

A. Implementation Exception / Jump (cont.)

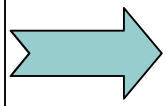
- Different from Java and other ordinary VM
  - Must manage continuation SP register

```
# Java can do this?
V = 1 + begin
          FOO
        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 List

- **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
- 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
    - Values popped from or pushed to the stack
  - Parsed by Ruby
  - This scheme enables flexible VM creation
    - Apply some optimization techs
    - Insert debug print
    - Make document automatically (similar to rdoc)

A. Insn Description Language (cont.)

```
/**
  @c put
  @e put self.
  @j self を置く。
 */
DEFINE_INSN
putself
()
()
(VALUE val)
{
  val = GET_SELF();
}
```

A. Insn Description Language (cont.)

```
/**
  @c variable
  @e get local variable(which is pointed by idx).
  @j idx で指定されたローカル変数をスタックに置く。
 */
DEFINE_INSN
getlocal
(ulong idx)
()
(VALUE val)
{
  val = *(GET_LFP() - idx);
}
```

Q. Does YARV have optimizer?
A. YARV Optimization Tech.

- Inline cache
  - Global “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
    - With this inline cache, this can be shortened to 1 insn
  - Method search
    - Current using Global method cache (which works wells)
    - Inline caching: planned (to be measured first)

A. YARV Optimization Tech. (cont.)

- Stack caching
  - 2 level stack caching
  - Cache 2 stack top values
  - With insn description, this can be automated
- Direct threaded code
  - Using GCC feature (label as value)

A. YARV Optimization Tech. (cont.)

- Super instructions
  - Merge two (or more) insns to one insn
  - Replace frequent insn sequence with super insn.
- Make Special instruction
  - putobject true → puttrue
  - 1 + 1 → put 1; put 1; opt_plus
- These techs are very effective because:
  - give C compiler more opportunity for optimization
- I want to do these automatically from statistics data, but difficult?

A. YARV Optimization Tech. (cont.)

- JIT compile
  - I'm searching for an easy way
    - Using existing libraries
    - Using copy code technique
      - Compile in C, and copy with label information
  - Seems to need much more effort

A. YARV Optimization Tech. (cont.)

- AOT compile
  - Substitute insn to C implementation code and compile it with C compiler
  - Description language will helps
  - Easy. Can rely on powerful C optimizer
  - Output will be normal C extension method
  - Very very simple experiment shows x100 speedup

Q. Is YARV working now?
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 arguments, optional/rest arguments
  - Call with block

Q. Current limitations on YARV? A. Many

- Can’t call Ruby from C
  - It mean that “10000.times{ … }” doesn’t work
  - To enable that, I must patch ruby/eval.c
- Missing some useful Ruby features
  - Stack trace, set_trace_func, method_missing
  - Proc as method visibility check, creating Proc object, …
  - And many many schemes :-P

Q. How fast will YARV run?
A. Benchmark result

- Everyone loves benchmarking!
  - Of course, me too!
  - and Everyone will love the result!
- (omitted)
  - Try on your machine ☺

Q. Why Original System?
A. Comparison to other systems

- v.s. JavaVM, .Net, Squeak, …
  - They have very nice library and optimizer
  - Just a mapping of Ruby Specification to VM's own 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. Schedule of YARV?
A. Schedule

- 2004
  - - Oct: implement basic feature of Ruby
  - Aug. 30, 31 meeting with Matz
  - Oct. 1-3 RubyConf 2004
  - - Nov: implement JIT compiler
  - - Dec: implement AOT compiler
- 2005
  - Debug debug debug!
  - - Mar: finish the fund
  - -?: Rite release

Q. What are the rest tasks?
A. Future Work

- Implement complete Ruby Specification
- Implement Optimizers
  - JIT/AOT compiler, other interesting optimize tech.
- Collect benchmark program
  - Do you have any program for it?
- Some other features
  - Marshaling YARV instruction sequence, …
- Implement other dynamic language on YARV
  - Scheme, ECMA Script, …, Python, Perl 6? ☺

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. Finished?
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” ☺

SASADA Koichi
Ko1 at atdot dot net