

YARV

Progress Report

RubyConf 2005 Oct. 14

SASADA Koichi

Tokyo University of Agriculture and Technology

Nihon Ruby no Kai

Ko1@atdot.net

Agenda

- Self Introduction and Japanese Activities
- Overview of YARV
- Goal of YARV
- Current YARV Status
 - YARV Design, Optimization Review
 - Evaluation
- Conclusion

Self Introduction

- “SASADA” the family name
- “Koichi” is given name → “ko1”
- A Student for Ph.D. 2nd grade
 - Systems Software for Multithreaded Arch.
 - SMT / CMP or other technologies
 - i.e.: Hyper threading (Intel), CMT (Sun), Power (IBM)
 - OS, Library, Compiler and Interpreter
 - YARV is my first step for Parallel interpreter (?)
 - Computer Architecture for Next Generation

Not a
Son-shi

At Public Position

Self Introduction (c

Well known as
Takahashi
Method

- Nihon Ruby no Kai
 - Organized by Mr. Takahashi (maki)
- Rubyist Magazine (<http://jp.rubyst.net/magazine>)
 - vol. 10 at 10th Oct. 2005
 - 1st anniversary at 6th Sep. 2005 (vol. 9)
- Ruby-dev summary
- English Diary some days
 - But retired...

Overview of YARV

Overview: Background

- **Ruby** is used world-wide,
~~one of~~ **the Most Comfortable
Programming Language**
- Ruby is slow, because interpreter
doesn't use Virtual Machine techniques
→ **We need RubyVM!**

Overview: YARV

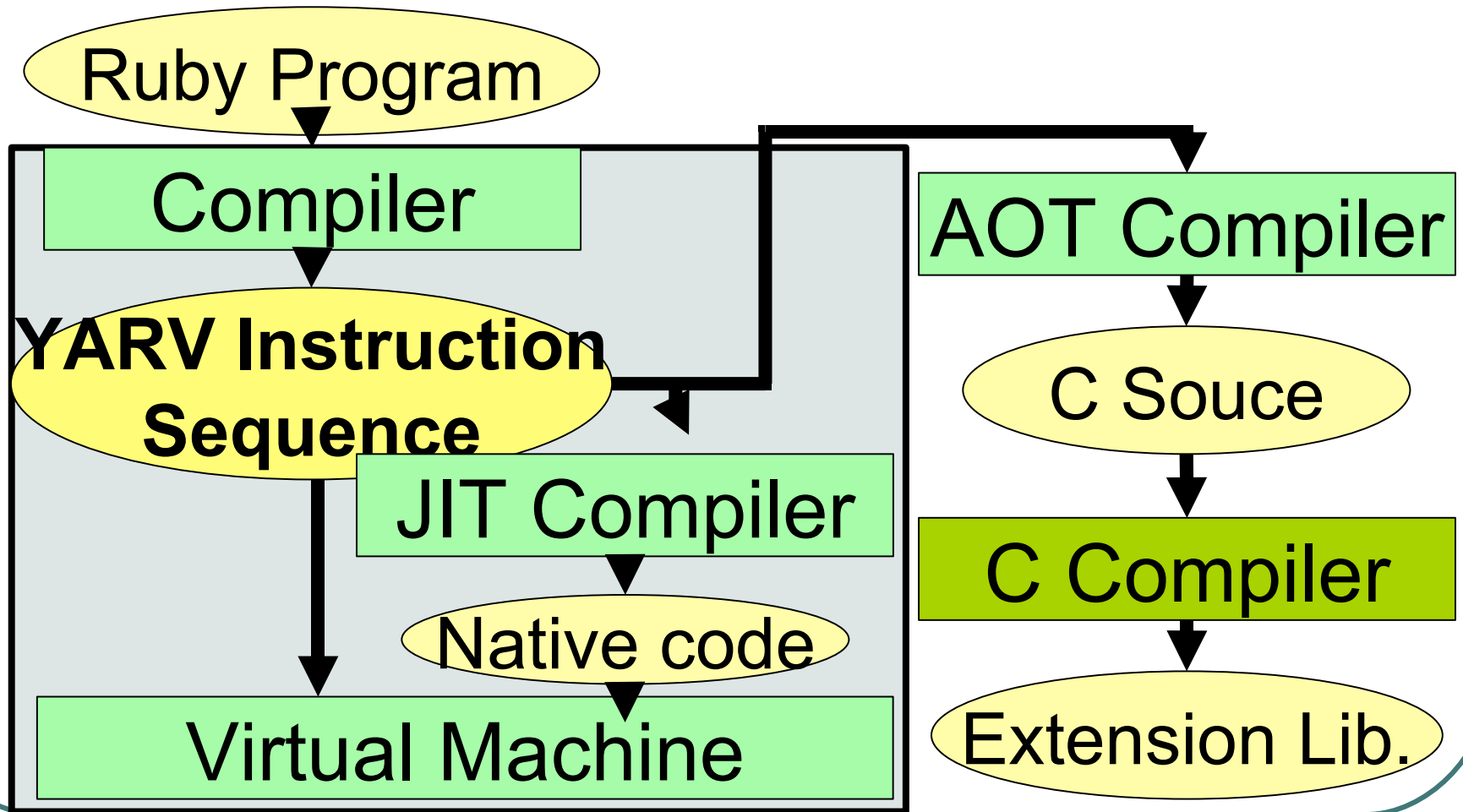
- YARV: Yet Another RubyVM
 - Started development on 1st Jan. 2004
 - At that time, there were some VMs for Ruby
 - Simple Stack Virtual Machine
- <http://www.atdot.net/yarv/>
- Ruby's license, of course

Overview:

FAQ (review of last year FAQ)

- Q: How does “YARV” pronounce?
- A: You can pronounce “YARV” what you like.
- Q: Should I remember the name of “YARV”?
- A: No. If YARV succeeds, it renames to Rite, if doesn't, no one remember YARV.
 - **About YARV, name is NOT important**
- Q: YARV will be Ruby 2.0?
- A: I hope so. But Matz will decide it.

Overview: YARV System



Overview: Current Interpreter

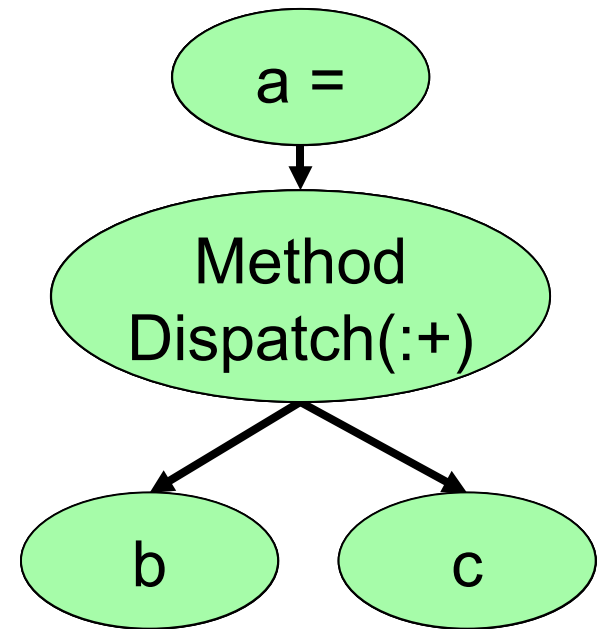
Ruby Program

`a = b + c`

Parse



Abstract Syntax Tree



Current Interpreter
traverses AST
directly

Overview

YARV - Stack Machine

Ruby Program
 $a = b + c$

Compile

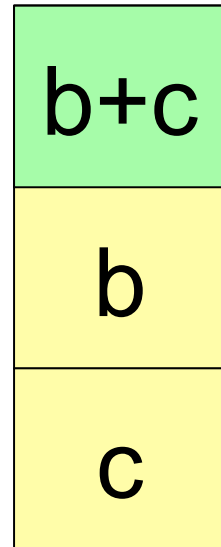
YARV Instructions

getlocal b

getlocal c

send +

setlocal a



YARV Stack

The Goal of YARV

The Goal of YARV

- YARV: Yet Another RubyVM
 - The RubyVM
 - To be the Ruby 2.0 VM Rite
- Fastest Ruby Interpreter
 - Easy to beat current fastest VM

The Goal of YARV (cont.)

- Support all Ruby features
 - Include Ruby 2.0 new syntaxes
- Native Thread Support
 - Concurrent execution (Giant VM lock)
 - Parallel execution on parallel machine
- Multi-VM Instance
 - Same as MVM in Java

New features

Goal: Ruby 2.0 syntax

- Matz will decide it 😊
- “{|...| ... }” == “->(...){ ... }”
Really?
- Multiple-values
 - Same as Array? Or first class multiple-values support?
- Selector-namespace?

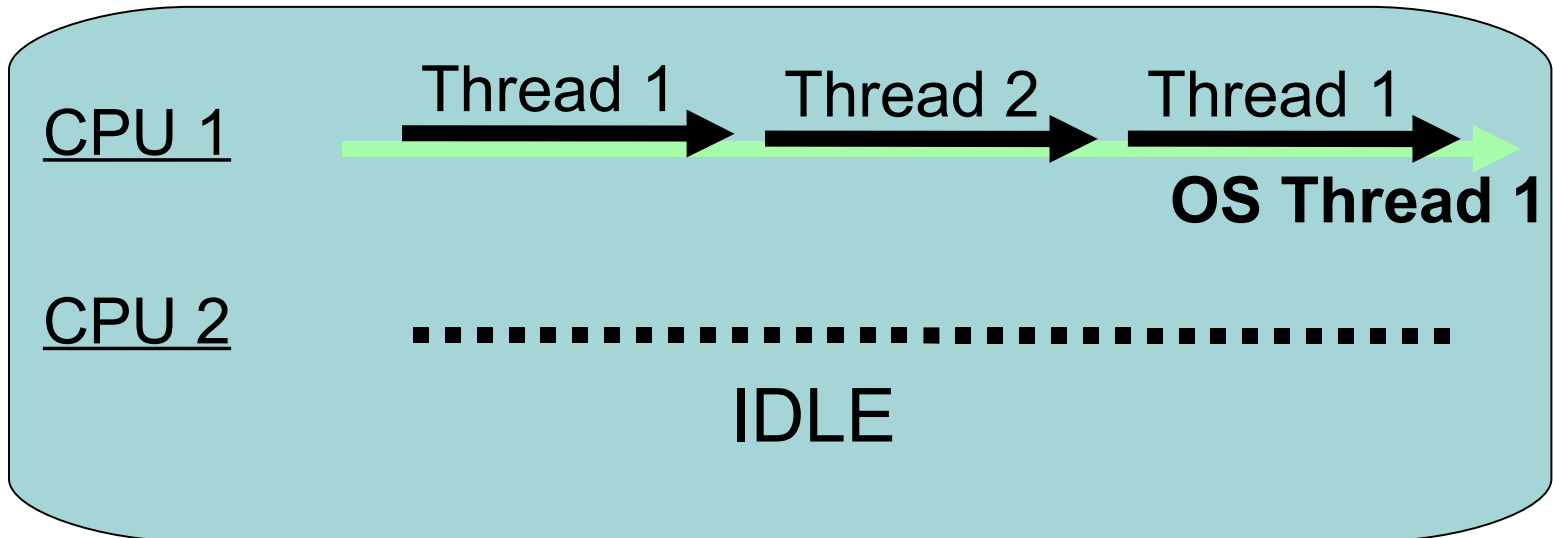
Goal:

Native Thread Support

- Three different thread models
- Model 1: User-level thread (Green Thread)
 - Same as current Ruby interpreter
- Model 2: Native-thread with giant VM lock
 - Same as current Ruby interpreter
 - Easy to implement
- Model 3: Native-thread with fine grain lock
 - Run ruby threads in parallel
 - For enterprise?

Goal: Native Thread Support (cont.)

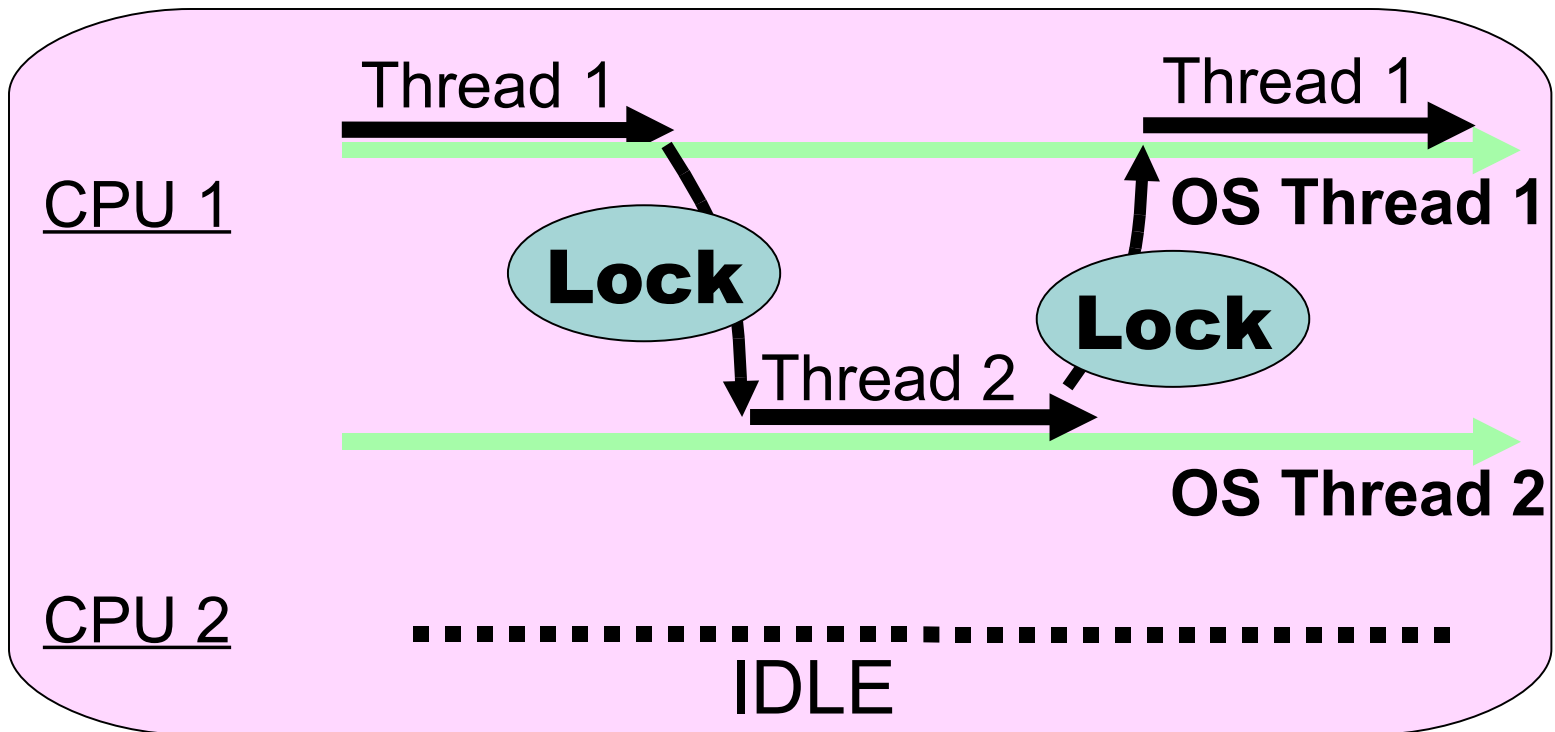
Current Ruby Interpreter
& Model 1



Goal: Native Thread Support (cont.)

Model 2:

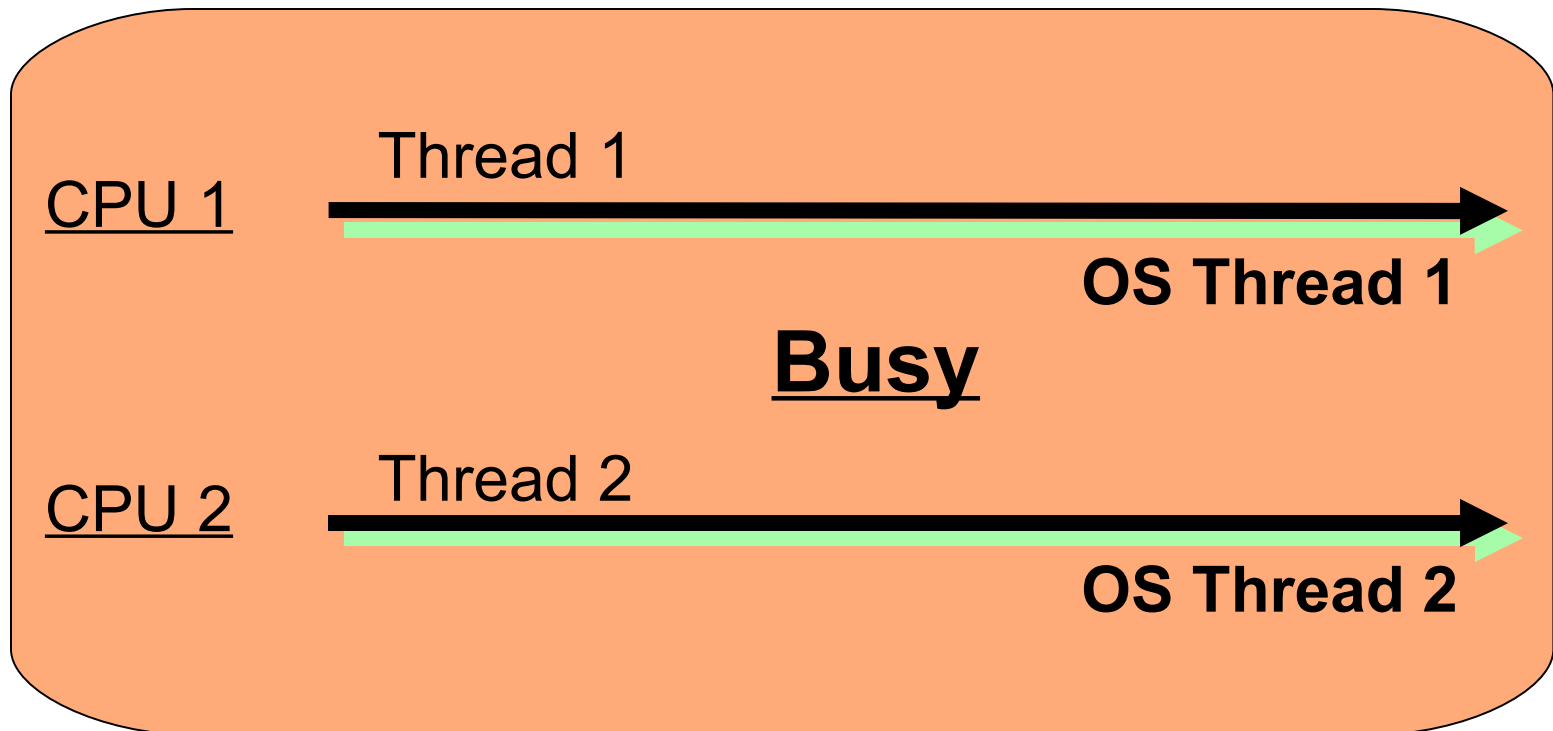
Native thread with Giant VM Lock



Goal: Native Thread Support (cont.)

Model 3:

Native thread with Fine Grain Lock



Goal: Native Thread Support Summary

	Model 1	Model 2	Model 3
Scalability	Bad	Bad?	Best
Lock overhead	No	Some	High
Impl. Difficulty	Norm.	Easy	Hard
Portability	Good	Bad	Bad

Goal:

Multi-VM Instance

Current Ruby Process

Process

Ruby Interpreter (VM)

Ruby Process with Multi-VM Instance

Process

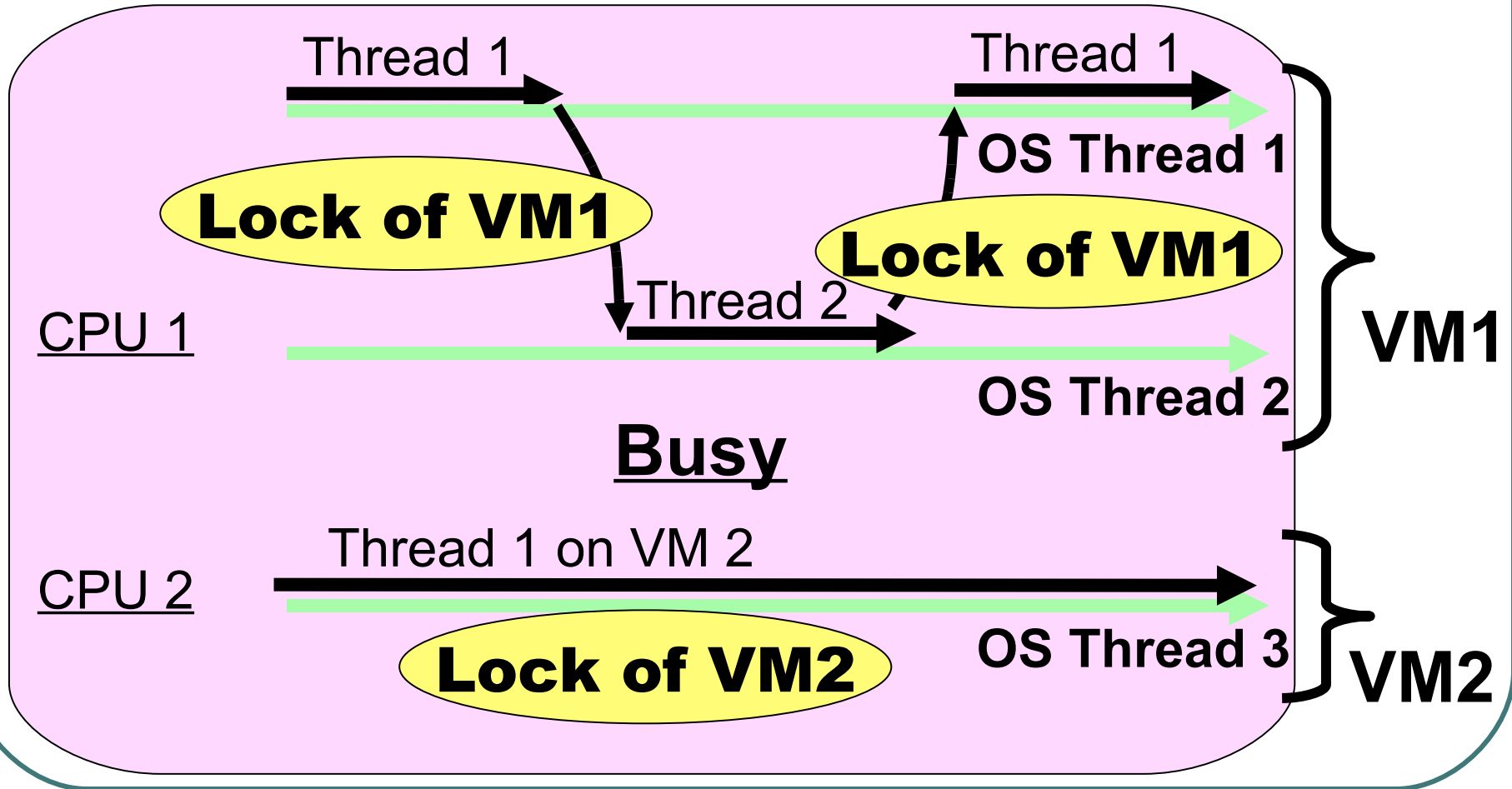
Ruby Interpreter (VM)



Goal: Multi-VM Instance (cont.)

- Current Ruby can hold only 1 interpreter in 1 process
 - Interpreter structure causes this problem
 - Using many global variables
- Multiple-VM instance
 - Running some VM in 1 process
 - It will help ruby embedded applications
 - **mod_ruby**, etc

Multi-VM Instance + Thread Model 2



Review

Summary with MV

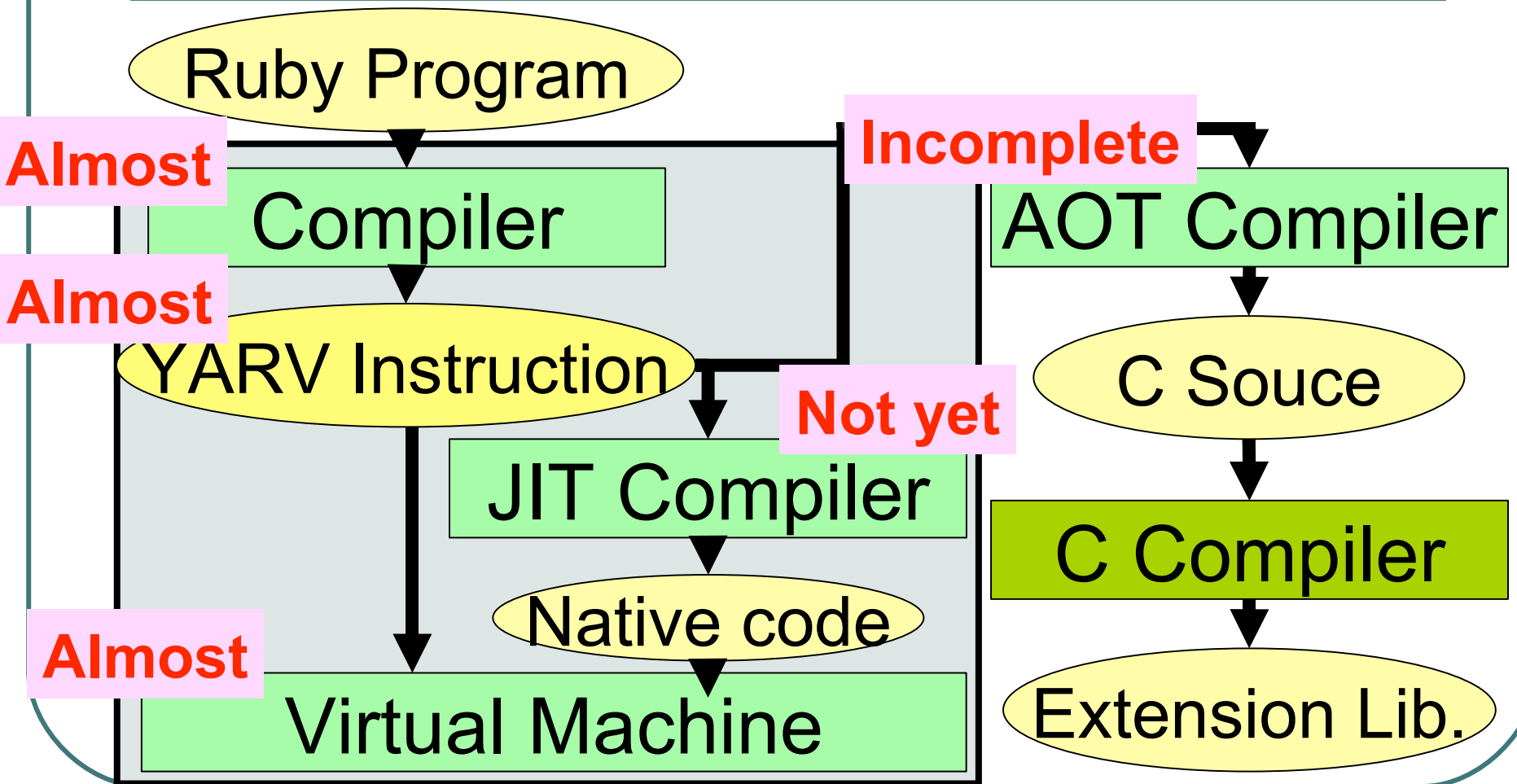
	M1	M2	M2+MV	M3
Scalability	Bad	Bad?	Good	Best
Lock overhead	None	Some	Some	High
Impl. Difficulty	Norm.	Easy	Easy	Hard
Portability	Good	Bad	Bad	Bad

Goal: Load Map

- All Ruby features support
 - Feb. 2006 ...?
- Native Thread Support
 - Experimental: Dec. 2005
 - Complete: 2006?(model 2) 2007?(model 3)
- Multi-VM Support
 - Experimental: Feb. 2006
 - Complete: 2006?

Status of YARV

Status: System



Status:

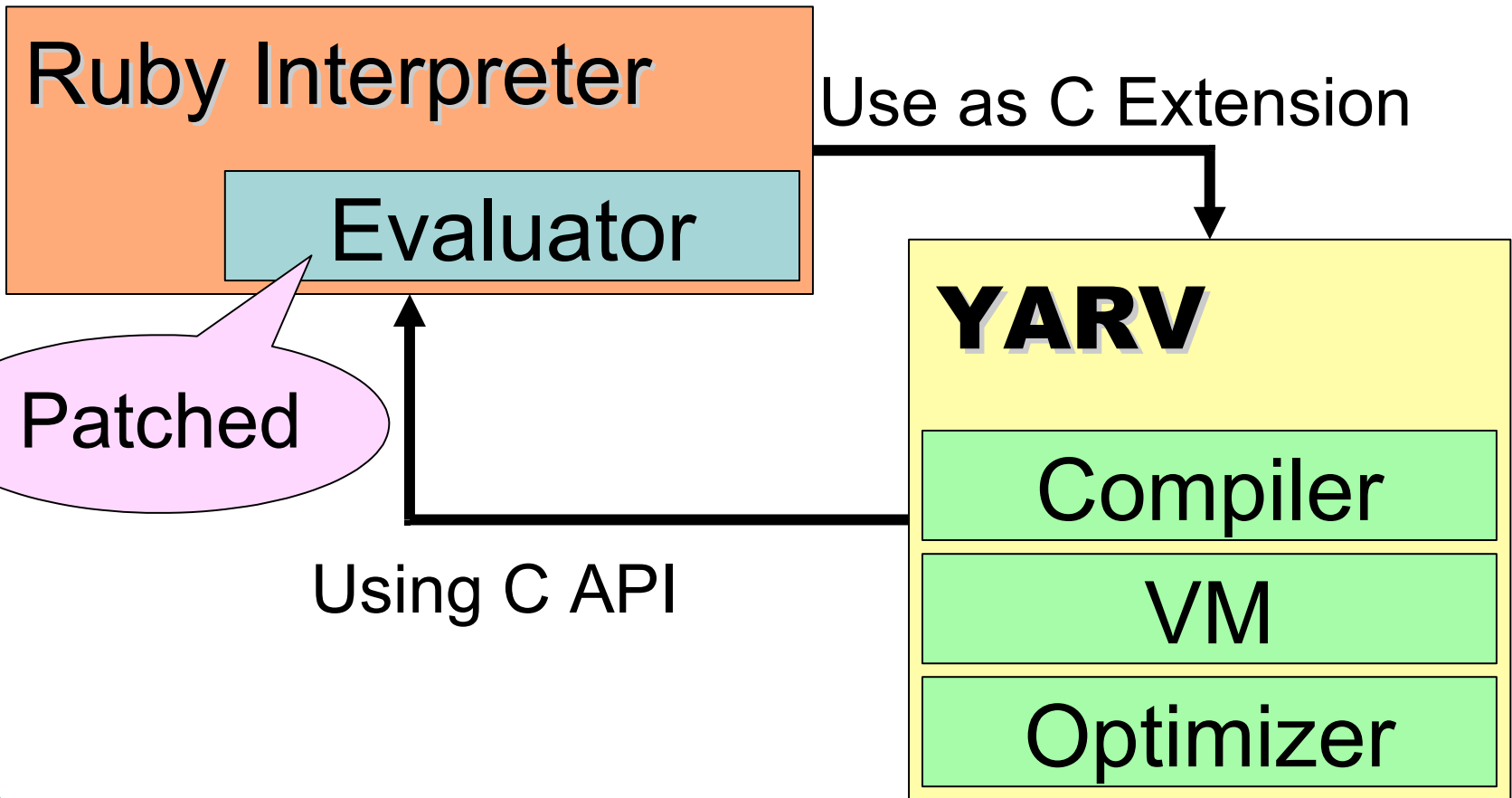
Supported Ruby Features

- Almost all Ruby features
- Not supported:
 - Few syntaxes ... `{|*arg| ...}`
 - Visibility
 - Safe level (`$SAFE`)
 - Some methods written in C for current Ruby implementation
 - Around Signal
 - C extension libraries
 - Because `yarv` can't run `"mkmf.rb"`

Status: Versions

- 0.2: YARV as C Extension
 - Need a patch to Ruby interpreter
- 0.3 (2005-8): YARV as Ruby Interpreter
 - Merged to Ruby source code (Ruby 1.9 HEAD)
 - Maintained on my Subversion repository
- Latest version: 0.3.2
 - Native thread (pthread / win32) supports on model 2

YARV 0.2.x



YARV 0.3.x

YARV merged with Ruby Interpreter

Future Work

Generational GC

m17n

Selector Namespace

...

YARV

Compiler

VM

Optimizer

Status:

Compile & Disasm CGI

- <http://www.atdot.net/yc/>

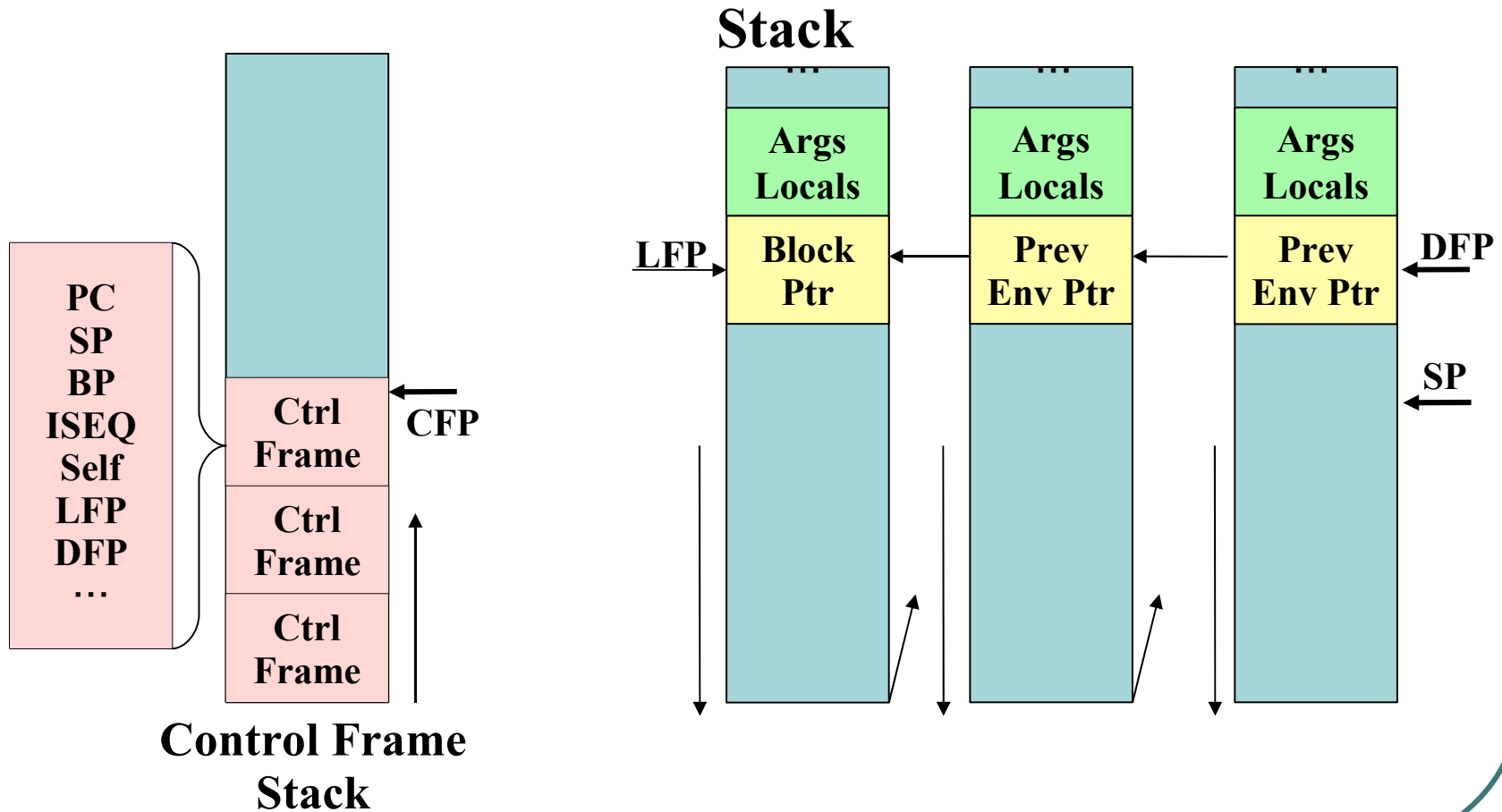
Status:

VM Design

- 5 registers
 - PC: Program Counter
 - SP: Stack Pointer
 - CFP: Control Frame Pointer
 - LFP: Local Frame Pointer
 - DFP: Dynamic Frame Pointer
- Some stack frame
- Control stack and value stack

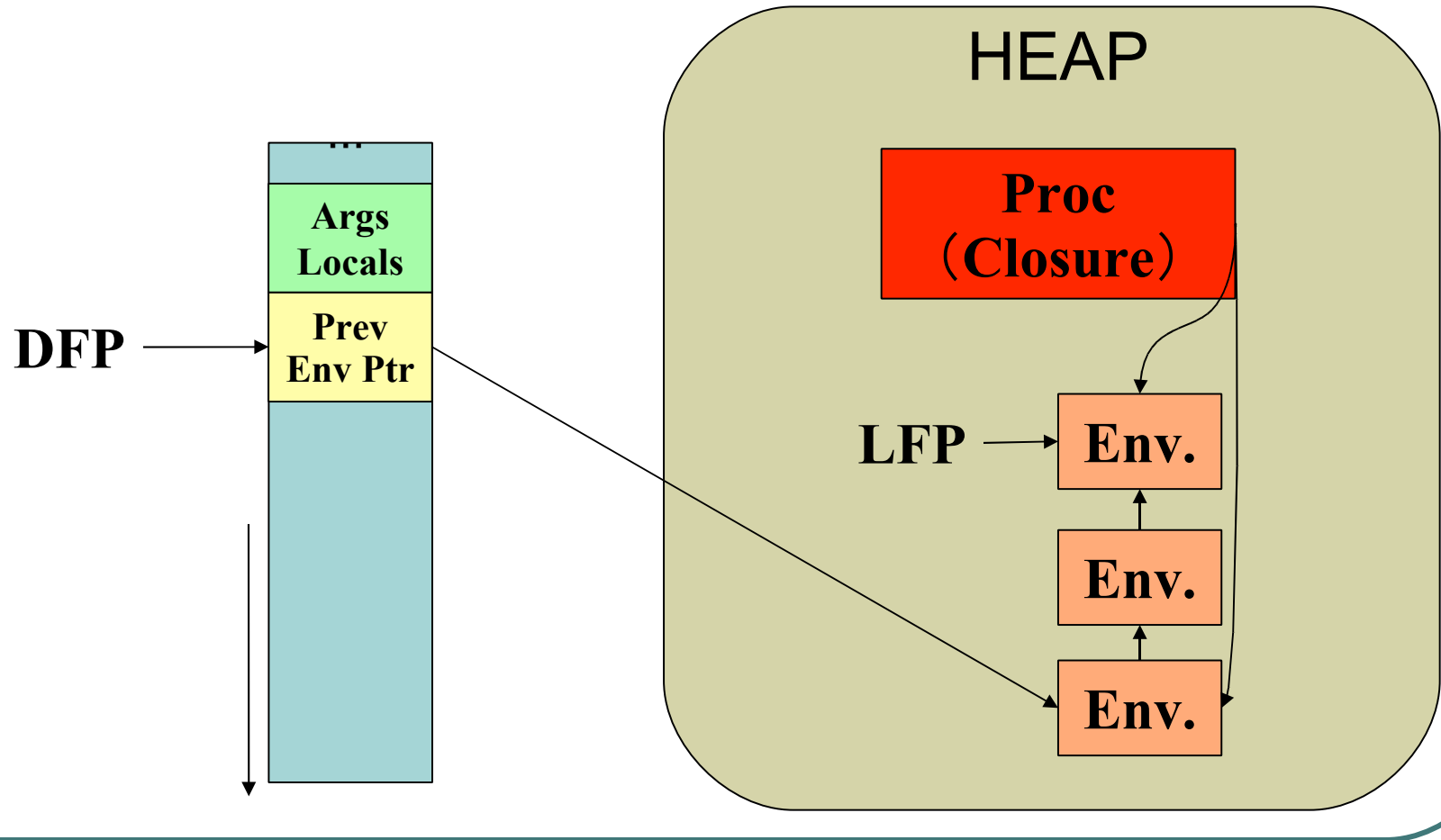
Status:

VM Design – Stack Frame



Status:

VM Design – Closure



Status:

VM Design – Exception

Call Graph of YARV Execution

VM handler Func (setjmp)

VM Func

C Func

VM handler Func(setjmp)

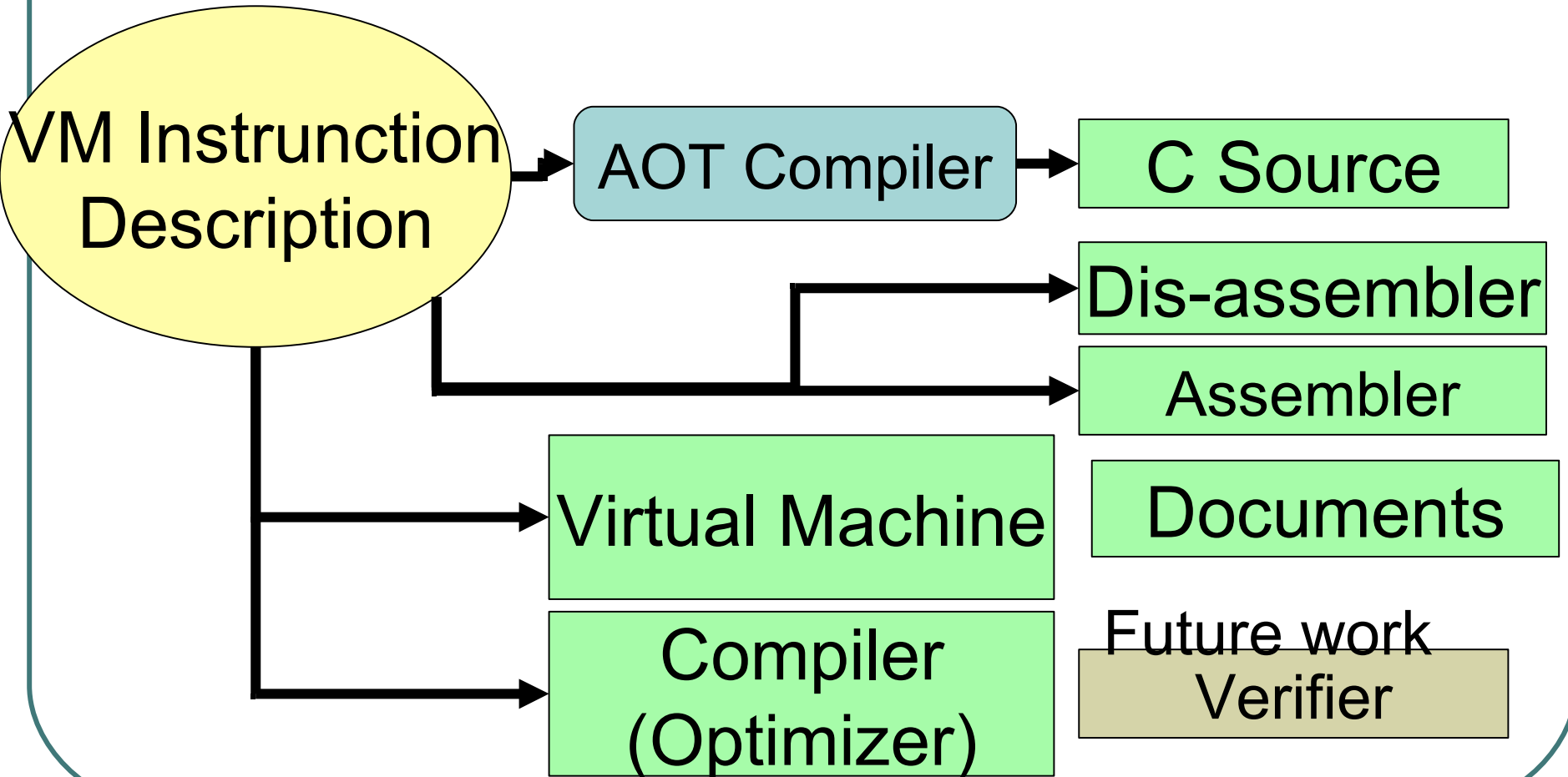
VM Func

C Func

Search
Exception
Table
If not match,
longjmp

Raise with
longjmp

Status: VM Generator (in Ruby)



Status:

Optimization

- Simple Stack Virtual Machine
 - Re-design Exception handling
- Peep-hole optimization on compile time
 - I gave up static program analysis
 - Dynamicity is your friend, but my **ENEMY**
- Direct Threaded code with GCC

Status: Optimization (cont.)

- Specialized Instruction
 - i.e.) Ruby program “x+y” compiled to special instruction instead of a method dispatch instruction

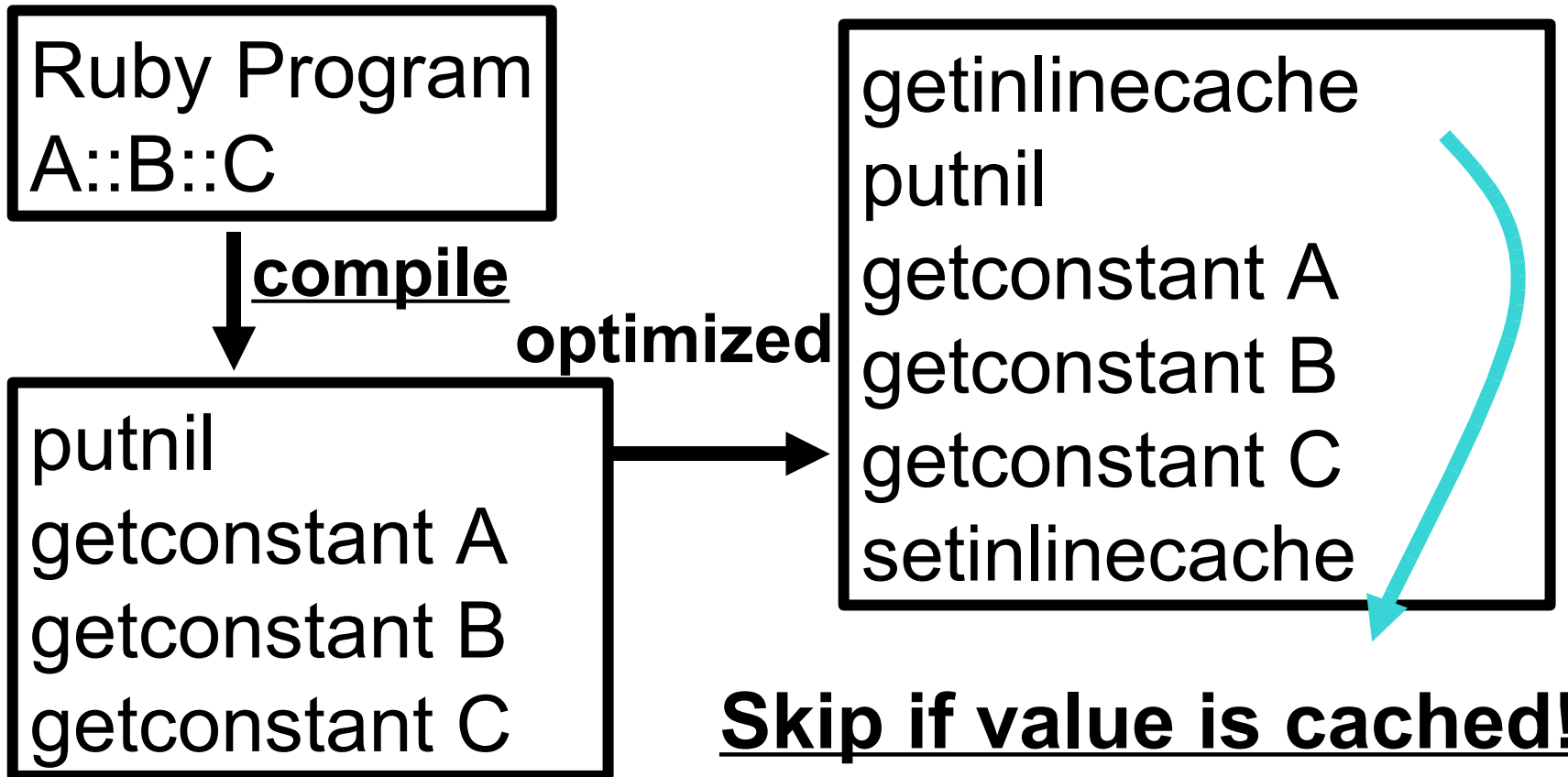
```
// Specialized “+” instruction  
instruction opt_plus(x, y){  
  if(x is Fixnum && y is Fixnum)  
    if(Fixnum#+ is not re-defined)  
      return x+y;  
  return x.+(y);
```

Status: Optimization (cont.)

- In-line Cache
 - In-line Method Cache
 - In-line Constant Value Cache
 - Because Ruby's "Constant Variable" is not Constant!
- Embed values in an Instruction sequence

Status: Optimization (cont.)

In-line Constant Value cache

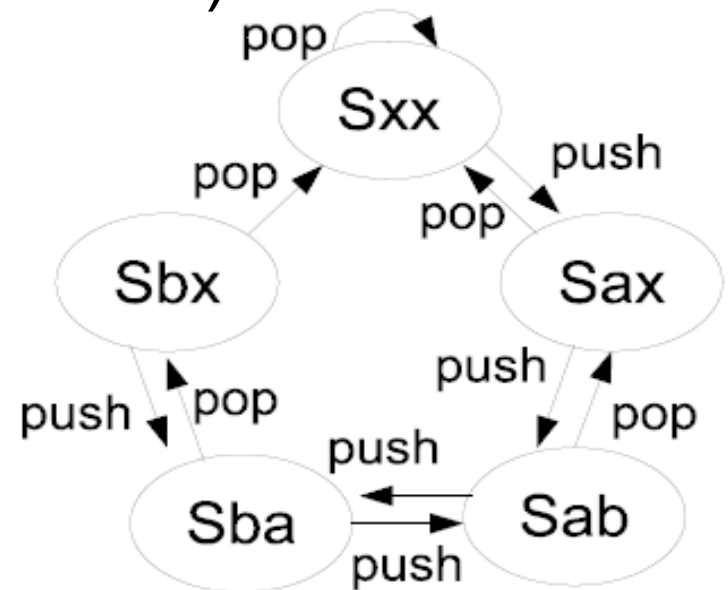


Status: Optimization (cont.)

- Unified Instruction
 - Operands Unification
 - $\text{Insn_A } x \rightarrow \text{Insn_A_}x$
 - Instructions Unification
 - $\text{Insn_A, Insn_B} \rightarrow \text{Insn_A_B}$
- Unified instructions are auto generated by VM generator
 - I only decide which instructions should be combined.

Status: Optimization (cont.)

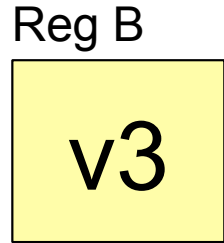
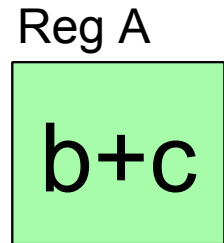
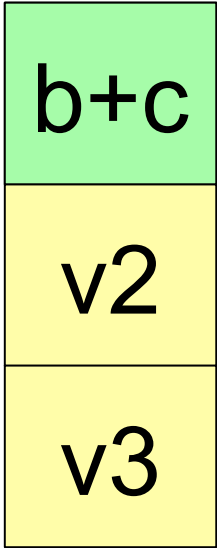
- Stack Caching
 - 2 registers, 5 states
 - putobject (put 1 values on stack)
 - putobject_xx_ax
 - putobject_ax_ab
 - putobject_bx_ba
 - putobject_ab_ba
 - putobject_ba_ab



Status: Optimization (cont.)

Stack Caching

Ruby Program
 $v1 = v2 + v3$



```
getlocal xx ax v2  
getlocal ax ab  
send_ab_ax +  
setlocal_ax_xx
```

No need to touch the Stack

RV
ack

Status: Optimization (cont.)

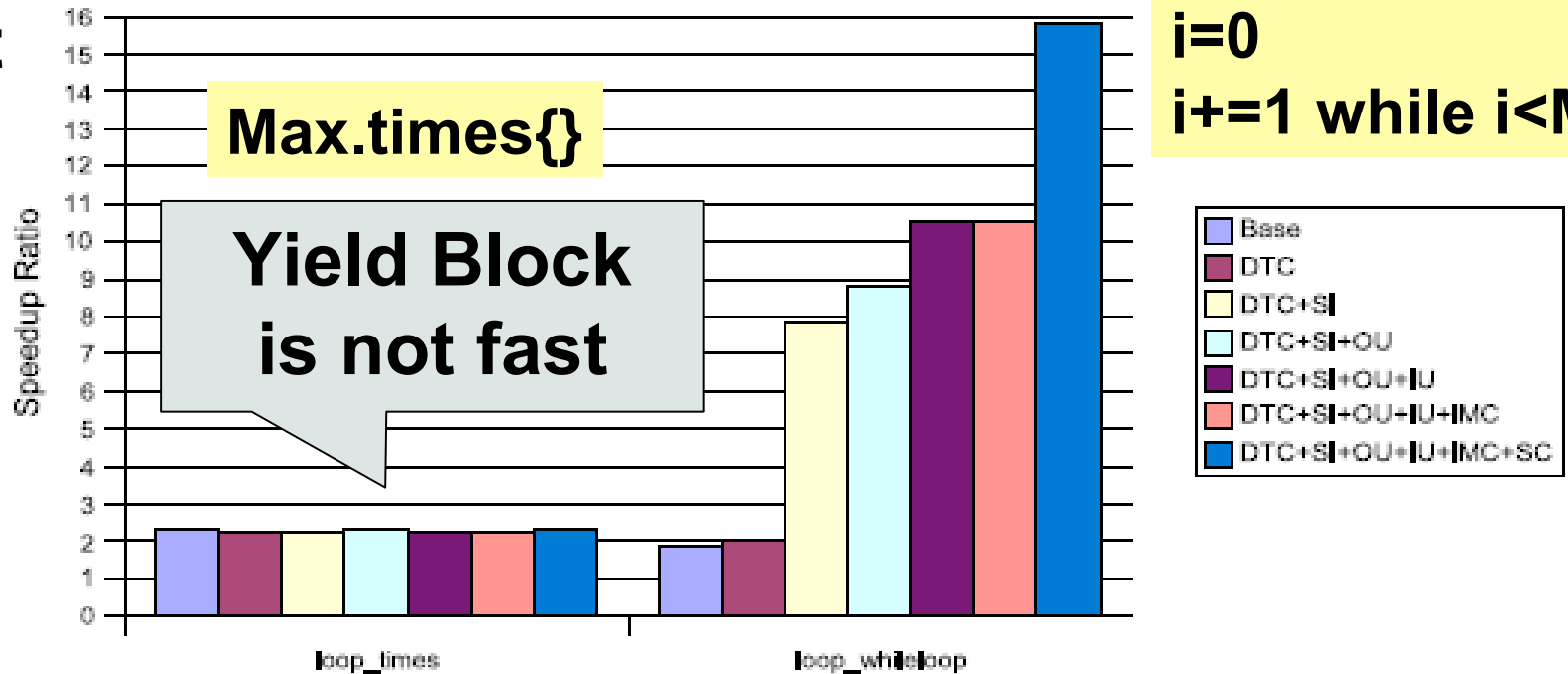
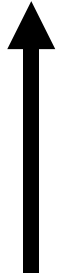
- JIT Compilation
 - I made easy one for x86, but...
 - Too hard to do alone. I retired.
- AOT Compilation
 - YARV bytecode → C Source
 - Easy to develop
 - Hard to support exception

Status: Demo

- YARV Building Demo?
- YARV Running Demo?

Status: Evaluation

Fast



i=0
i+=1 while i<Max

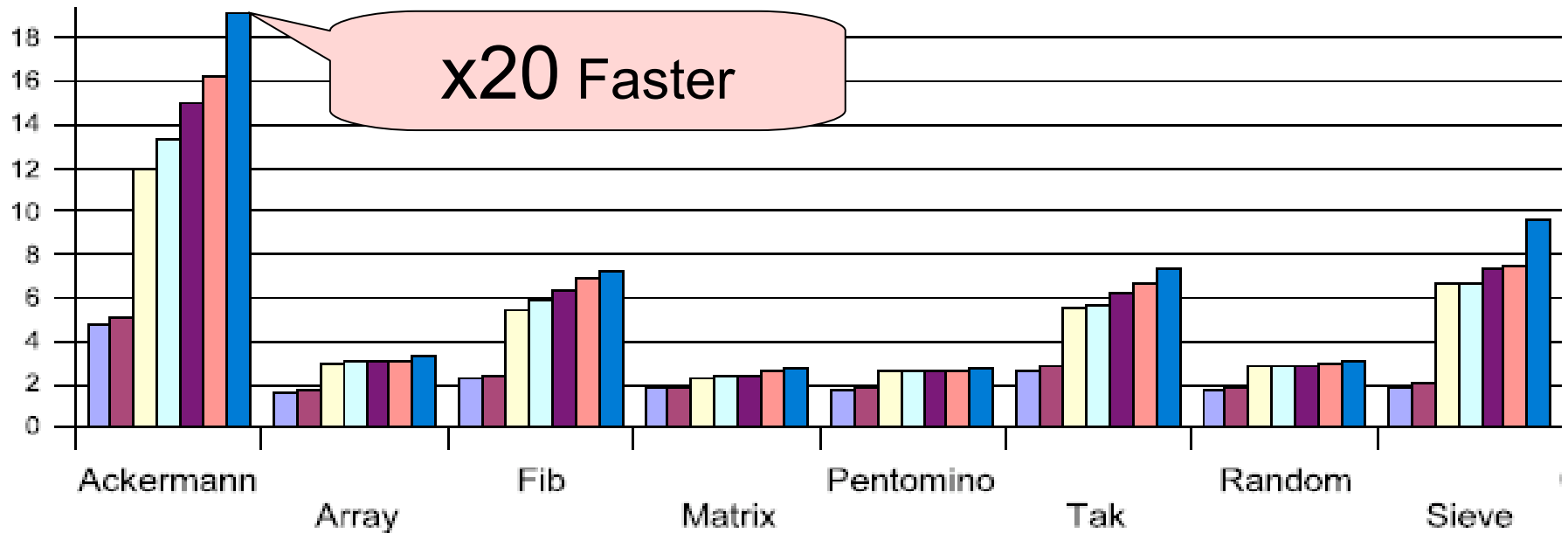
Max.times{ }

Yield Block is not fast

- Base
- DTC
- DTC+SI
- DTC+SI+OU
- DTC+SI+OU+IU
- DTC+SI+OU+IU+IMC
- DTC+SI+OU+IU+IMC+SC

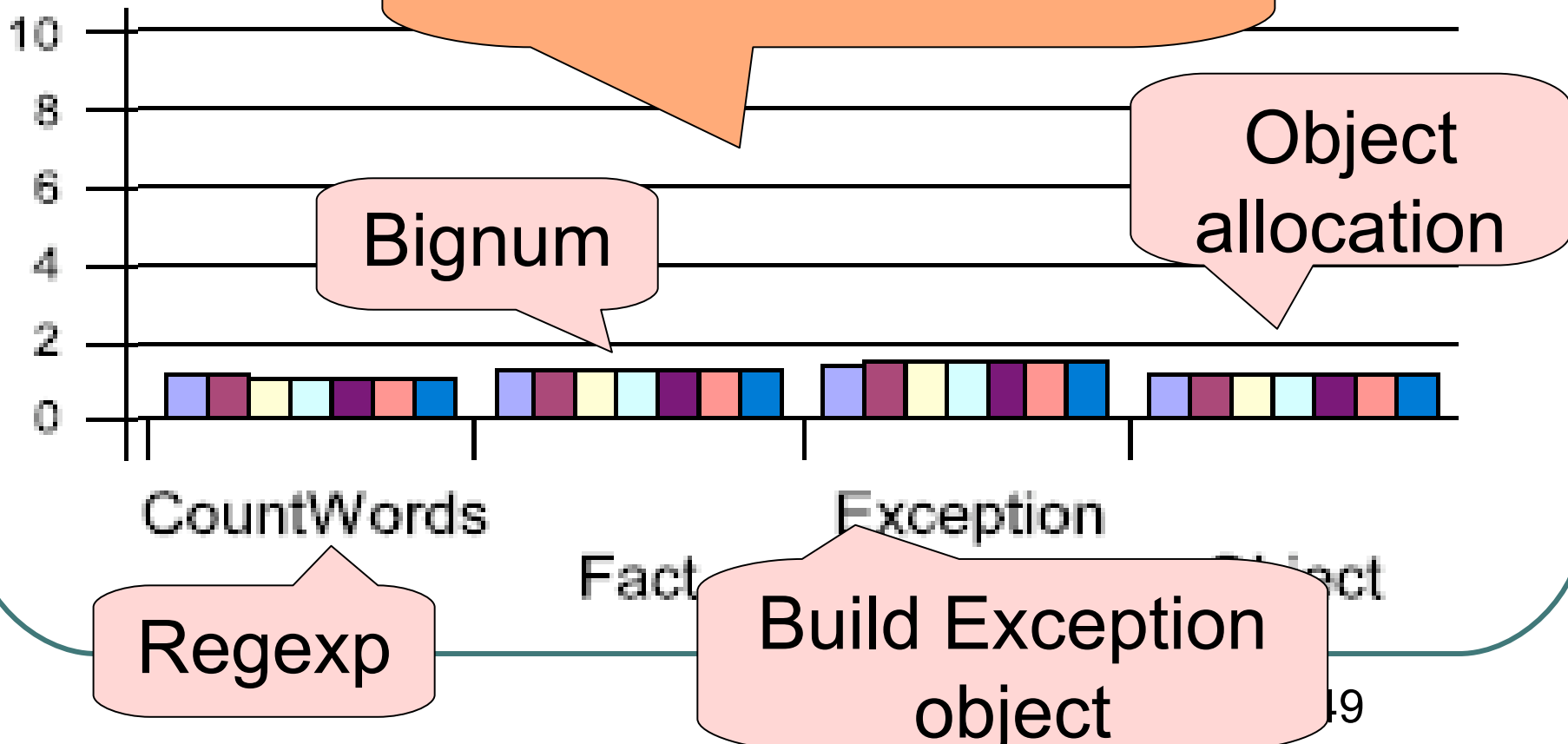
Base: only base VM **OU:** Operand Unification **IMC:** Inline Method Cache
DTC: Direct Threaded Code **IU:** Instruction Unification **SC:** Stack Caching
SI: Specialized Instruction

Status: Evaluation (cont.)



Status: Evaluation (cont.)

No speed-up.
VM is not bottleneck.



Regexp

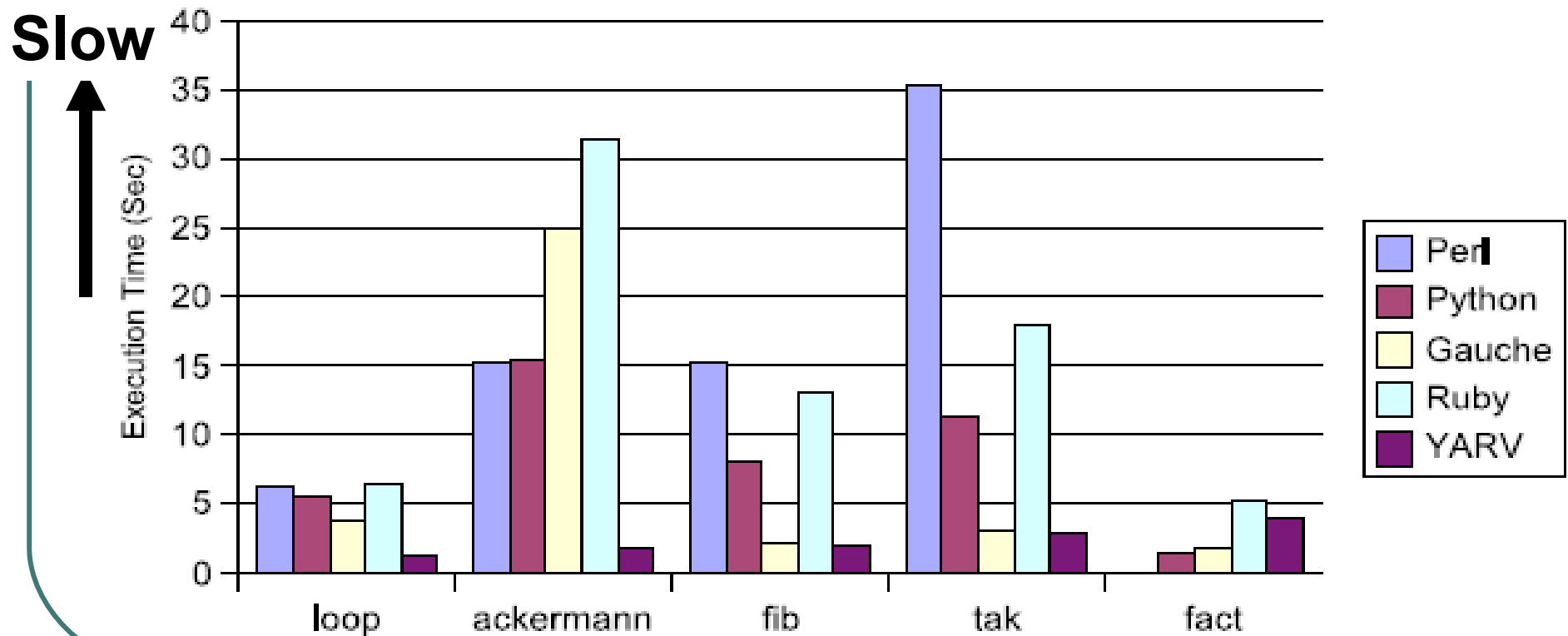
Bignum

Object allocation

Build Exception object

Status: Evaluation (cont.)

Compare with other languages



Status: Awards

- 2004: Funded by IPA Exploratory Software Development “Youth”
 - IPA: Information-technology Promotion Agency, Japan
- 2005: Funded by IPA Exploratory Software Development (continuance)
 - I can't walk away from the development ☹️
- 2004: Got Award as “Super Creator” from IPA

Conclusion

Conclusion

- YARV supports **almost** Ruby syntaxes
- YARV supports some Ruby libraries
 - But can't build Extension Libraries
 - Because YARV can't run "mkmf.rb"
- YARV 0.3.2 supports native thread
- YARV achieves **significant speedup** for Ruby programs execution which have VM bottleneck
 - This means that we can enjoy Symbol Programming with Ruby

Conclusion: Future work

- Support all Ruby features
 - At least, YARV must work with “mkmf.rb”
- Support every Thread Model
 - Especially model 2 and 3
- Support Multi-VM Instance

How Can You Help me

- Any comments are welcome
 - Build reports, Bug reports, architecture reports, ...
- yarv-devel Mailing List
 - English ML for YARV development
 - Matz and other Japanese also join
- YARVWiki
 - <http://yarv.rubyforge.org/pukiwiki/pukiwiki.php>
- Give me a job (I'll finish my course 2 years later)

Special Thanks

- Matz the architect of Ruby
- IPA: Information-technology Promotion Agency, Japan (my sponsor)
- Gabriele Renzi, Ippei Tate
- YARV development ML subscribers
 - Yarv-dev (Japanese)
 - Yarv-devel (English)
- All rubyists

Finish

“YARV Progress Report”

Thank you for your attention.
Any Questions?

SASADA Koichi
ko1@atdot.net