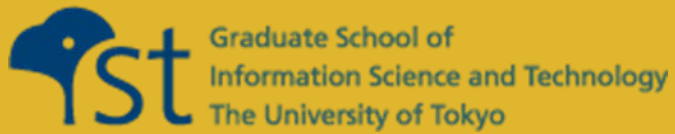
Ruby Memory Management Hacks

Ruby のメモリ管理とかなんとか



SASADA Koichi <ko1@rvm.jp>

Department of Creative Informatics,
Graduate School of Science and Technology,
The University of Tokyo

Summary of this talk

- Ruby 1.9.2 may improve several performance
  - Memory management efficiencies
  - Fiber performance
  - Memory profiler feature
- Researches are progressing
  - Memory Profiler
  - MVM

Agenda

- **(1) Memory Management Hacks**
  - OS dependent Memory Management
  - Memory profiler (including Student's work)
- **(2) VM performance improvement**
  - Fiber performance improvement (Student's work)
  - Other progress
- **(3) Multi-VM (MVM) Progress**

(1) Memory Management Hacks

1. Trivial Optimizations
2. Memory Profiler
   - “objspace” library
   - New T_DATA representation
3. OS dependent Memory Management

Memory Management Importance

- Memory is VERY BIG, EXPENSIVE!
- malloc()/free() hell
- Garbage collection overhead
- Long running services
  - e.g. 30 years continuous running
    - 20/Nov/1979 JST

Memory Management Problems

- CRuby/MRI has several issues about M/M
  - Naive Garbage Collection
    - Not a JVM, .NET or other VM including smart garbage collection
    - Programmer friendly, GC unfriendly C extension interfaces
  - No memory profilers
  - Object Fragmentation

Solution (1)
Reduce GC Pressure

- CRuby.GC != Generational GC
- Old, long life object might be GC pressure (increase GC frequency)
  - NODE, InstructionSequence
    - Inline cache entries (ICE) are NODE.
  - Modules, Classes
- **[1.9.2 Feature]** ICE is no longer GC managed object (manage it explicitly).
  - Reducing long lived NODE object.
    → Reducing GC pressure.
  - Planning to adopt other NODE object.

Solution (2)
Memory Profiler

- Memory Profiler Support
  - ObjectSpace#memsize_of(obj)
  - Experimental Memory Profiler

- **[NOTE] These methods are not expected to work except C Ruby.**

Memory Profiler ObjctSpace#memsize_of

- **[1.9.2 Feature]** 1.9.2 introduces "objspace" library:
  - This library adds the following methods
    - ObjectSpace.memsize_of(obj)
    - ObjectSpace.count_objects_size
    - ObjectSpace.count_nodes
    - ObjectSpace.count_tdata_objects

Memory Profiler ObjctSpace#memsize_of example

```
require ‘objspace’
ObjectSpace.memsize_of(‘.’*100)
#=> 100
ObjectSpace.memsize_of(Time.now)
#=> 60
ObjectSpace.memsize_of(Thread.new{})
#=> 524888
ObjectSpace.memsize_of(STDOUT)
#=> 124
```

Memory Profiler
New T_DATA Representation

- T_DATA: Wrapping C structure
- Traditional T_DATA representation includes:
  - pointer to memory object what T_DATA wrap
  - pointer to mark function
  - pointer to free function
- Make a T_DATA value with Data_Wrap/Make_Struct(klass, mark, free, sval)
- T_DATA lacks:
  - Type Identity
  - Memory size function

Memory Profiler
New T_DATA representation (2)

- **[1.9.2 Feature]** We introduce the type rb_data_type_t:
  - wrap_struct_name
  - pointer to mark function
  - pointer to free function
  - pointer to memsize_of function
- Make T_DATA object with TypedData_Wrap/Make_Struct(klass, type, data_type, sval)

Memory Profiler
New T_DATA representation (3)

- Example:

```
static const rb_data_type_t time_data_type = {
  "time",
  time_mark, time_free, time_memsize,
};
...
obj = TypedData_Make_Struct(klass, struct time_object,
                            &time_data_type, tobj);
```

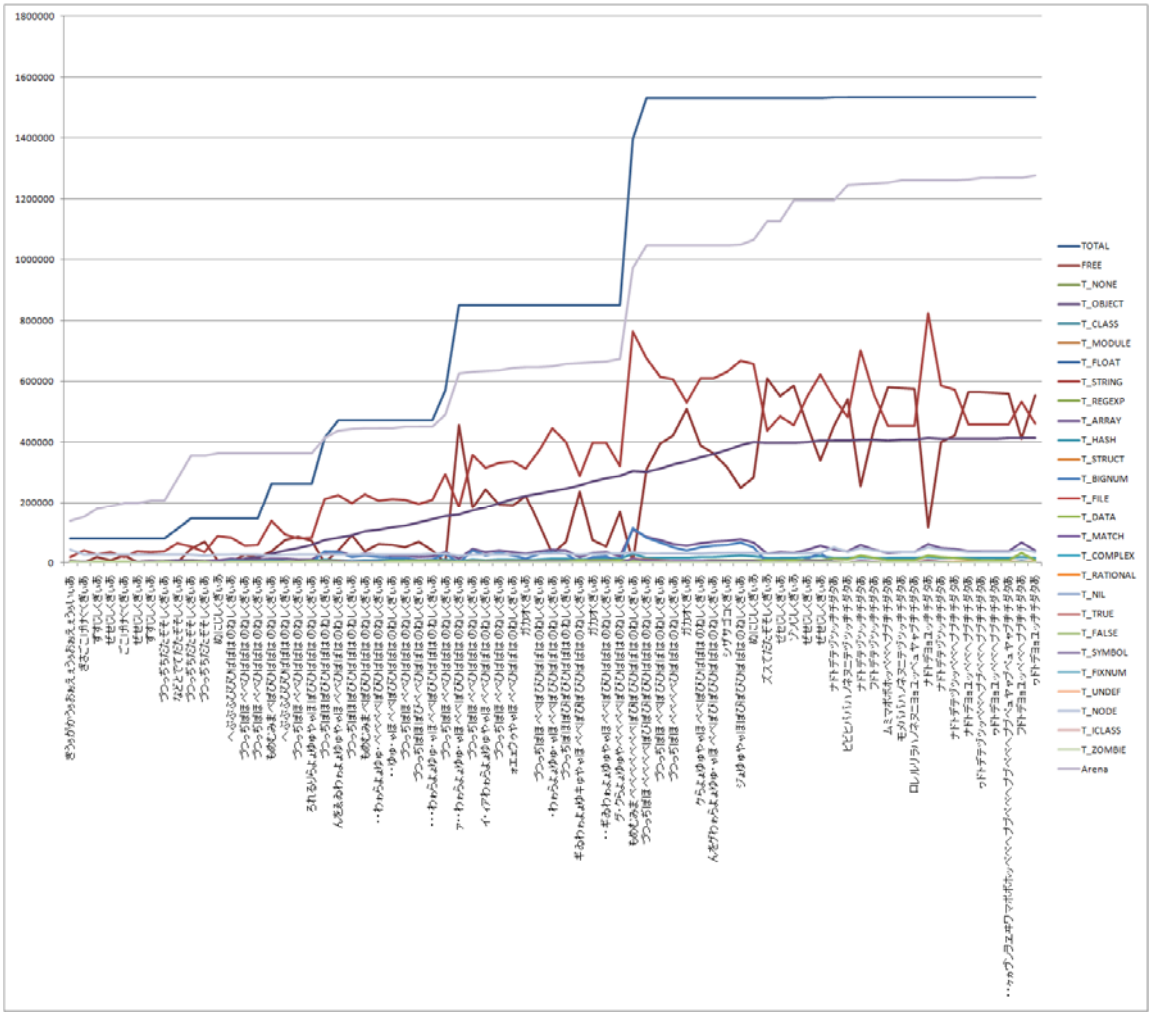
Memory Profiler
New T_DATA representation (4)

- New C APIs
  - rb_objspace_data_type_memsize(obj)
  - rb_objspace_data_type_name(obj)
  - RTYPEDDATA_DATA(obj)
  - RTYPEDDATA_P(obj)
  - RTYPEDDATA_TYPE(obj)

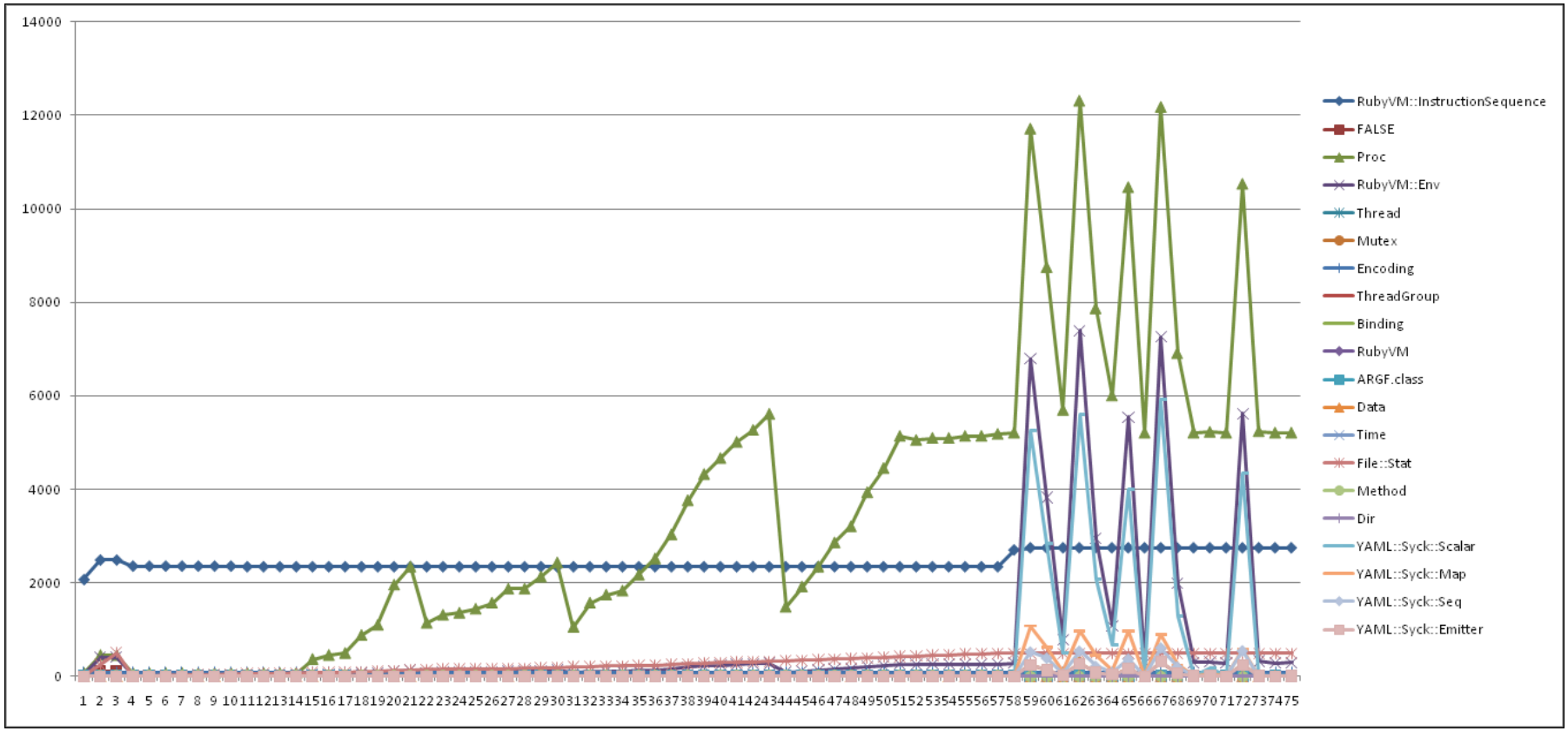
Memory Profiler
Statistic Profiler

- Sampling memory status periodically
- We already get tools

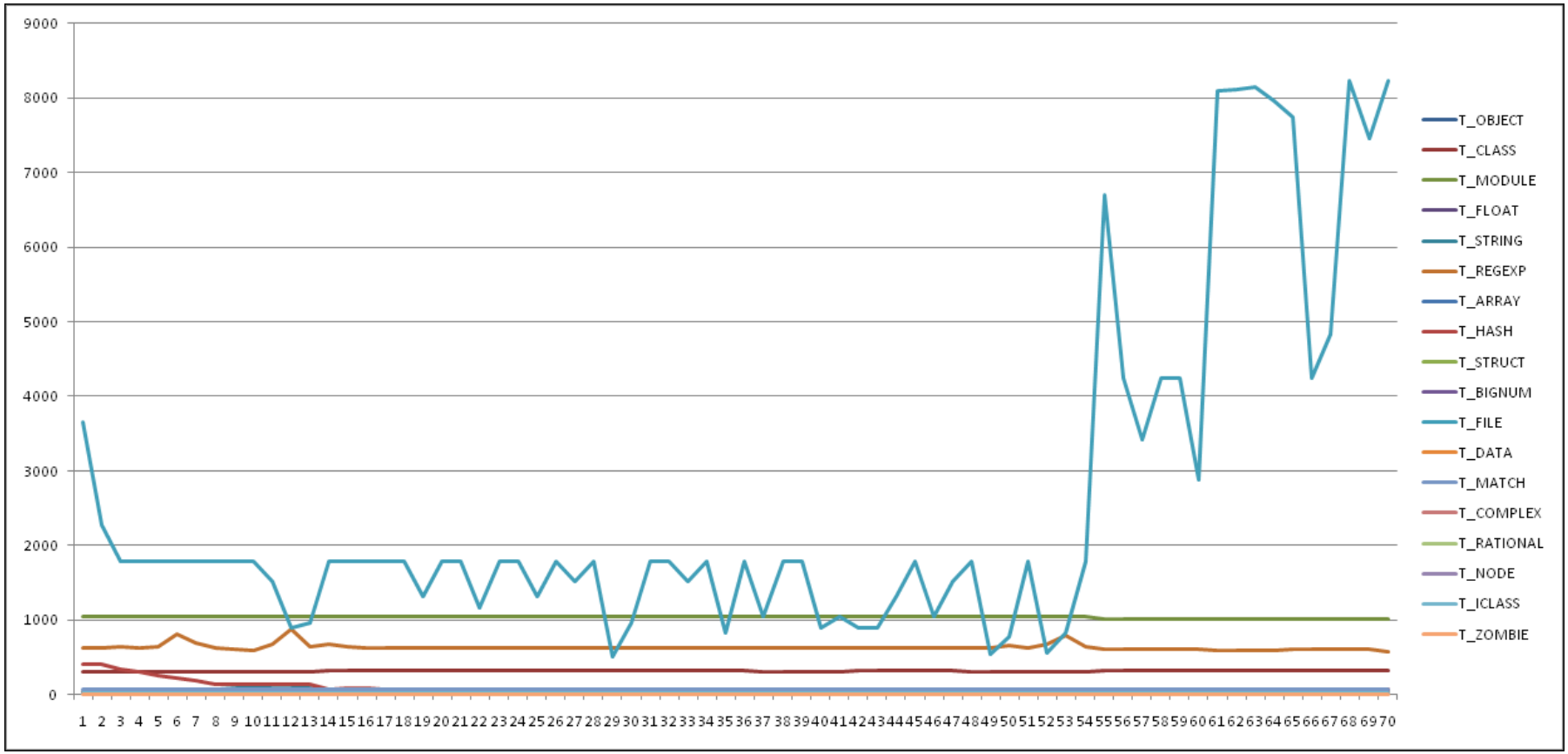
Statistic Profiler Examples (per T_xxx + Stack frame)



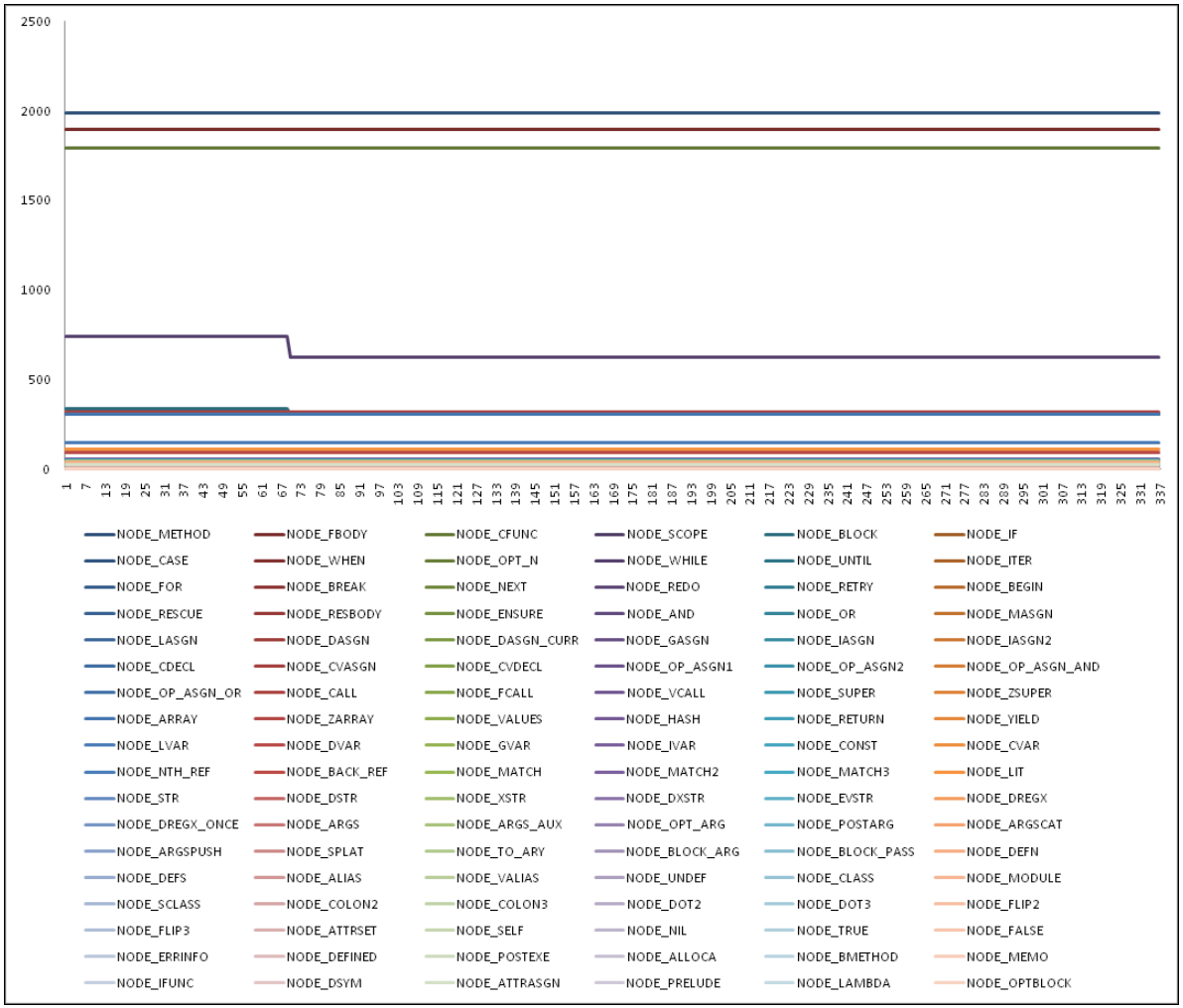
Statistic Profiler Examples (per Classes)



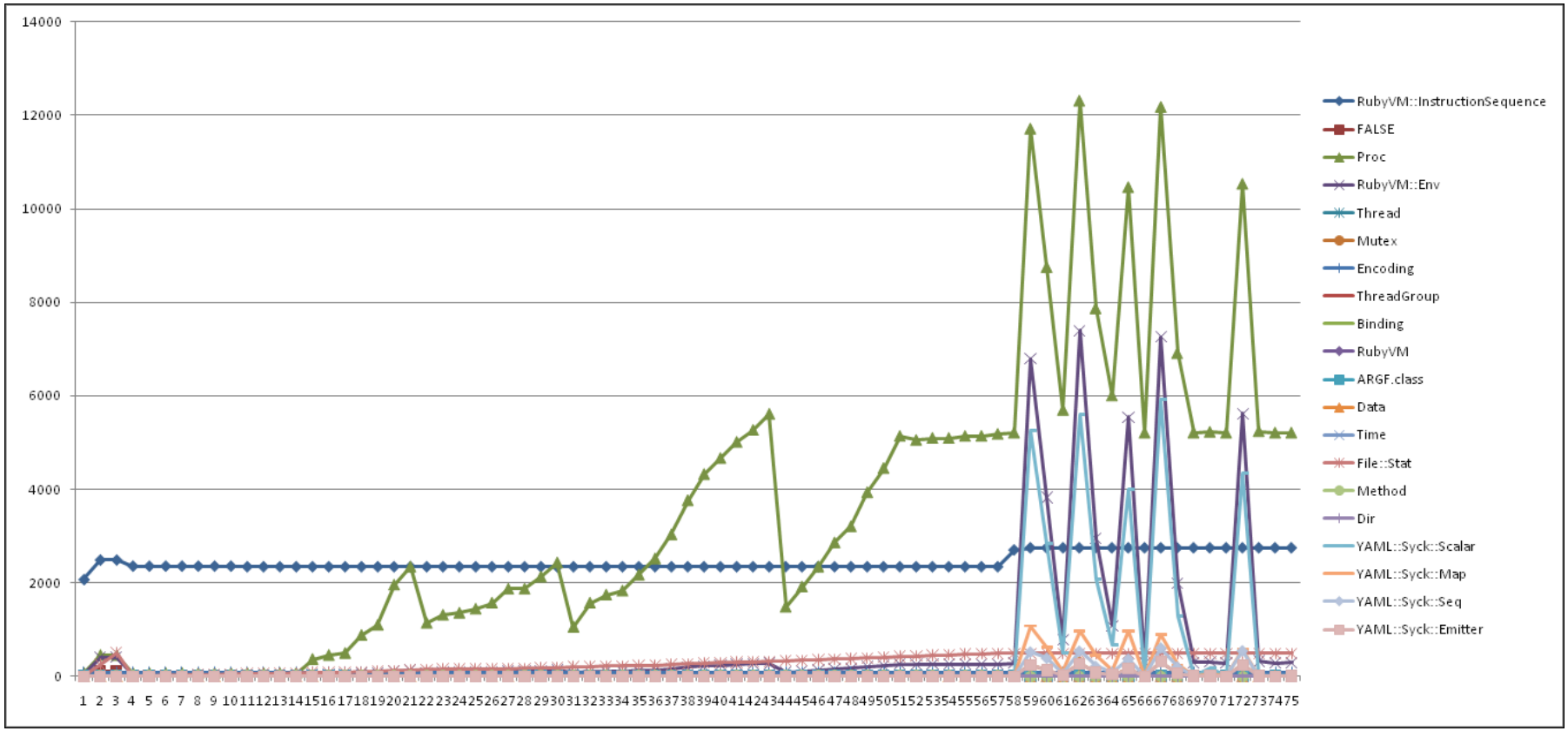
Statistic Profiler Examples (T_xxx / count)



Statistic Profiler Examples (NODE)



Statistic Profiler Examples (T_DATA)



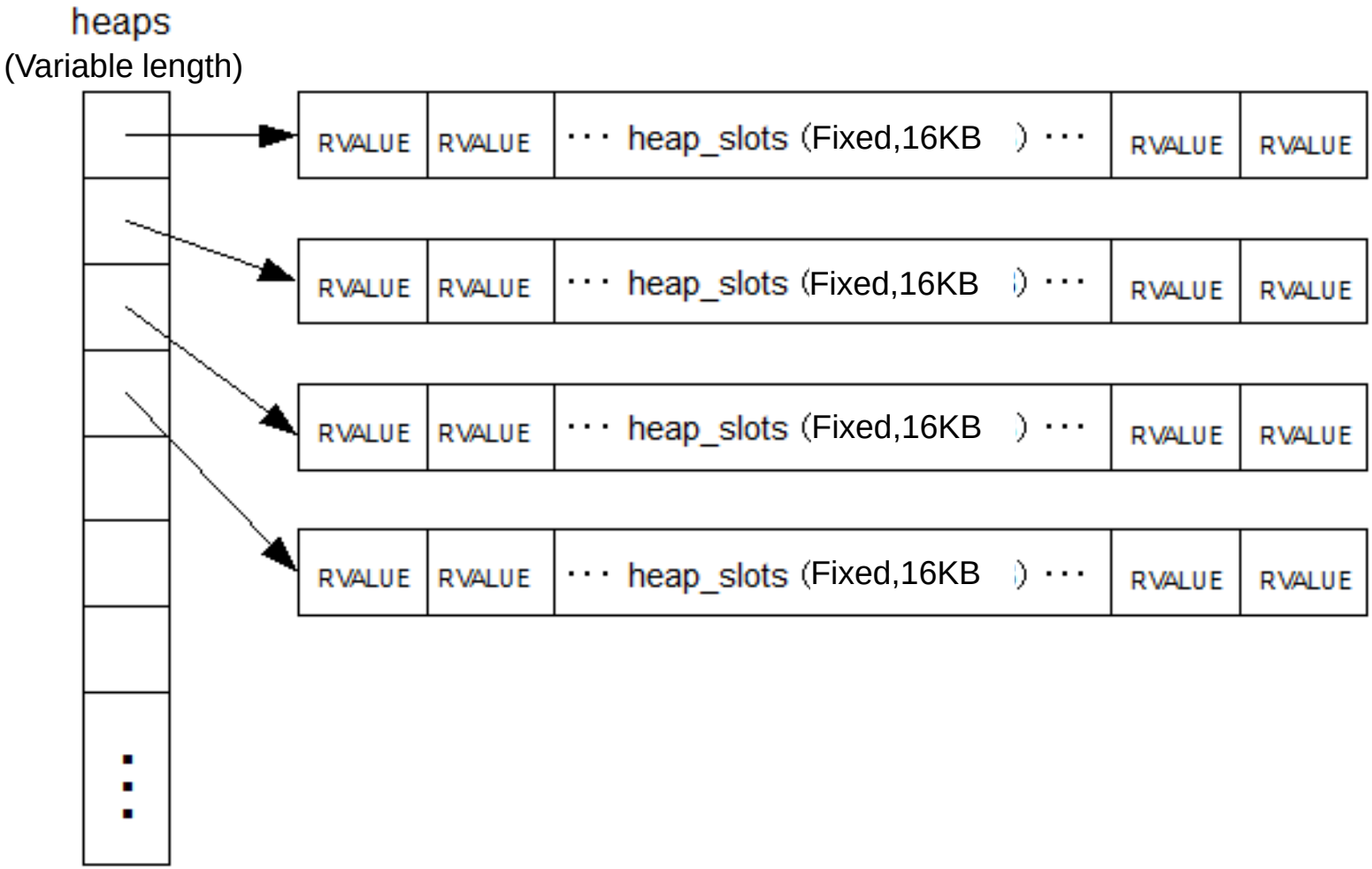
Memory Profiler
Deterministic Profiler

- **[Research]** We are working on deterministic memory profiler
  - **Explicit realtime memory usage**
  - **Explicit object lifetime**
  - **Explicit object generated place (file:line)**
  - Prepare probe points and some APIs
- Inspired from Java realtime memory profiler, valgrind/massif profiler

Solution (3)
OS dependent Memory Management

- Problem: Object Fragmentation
  - Memory block can not be de-allocate if at least one object exists.

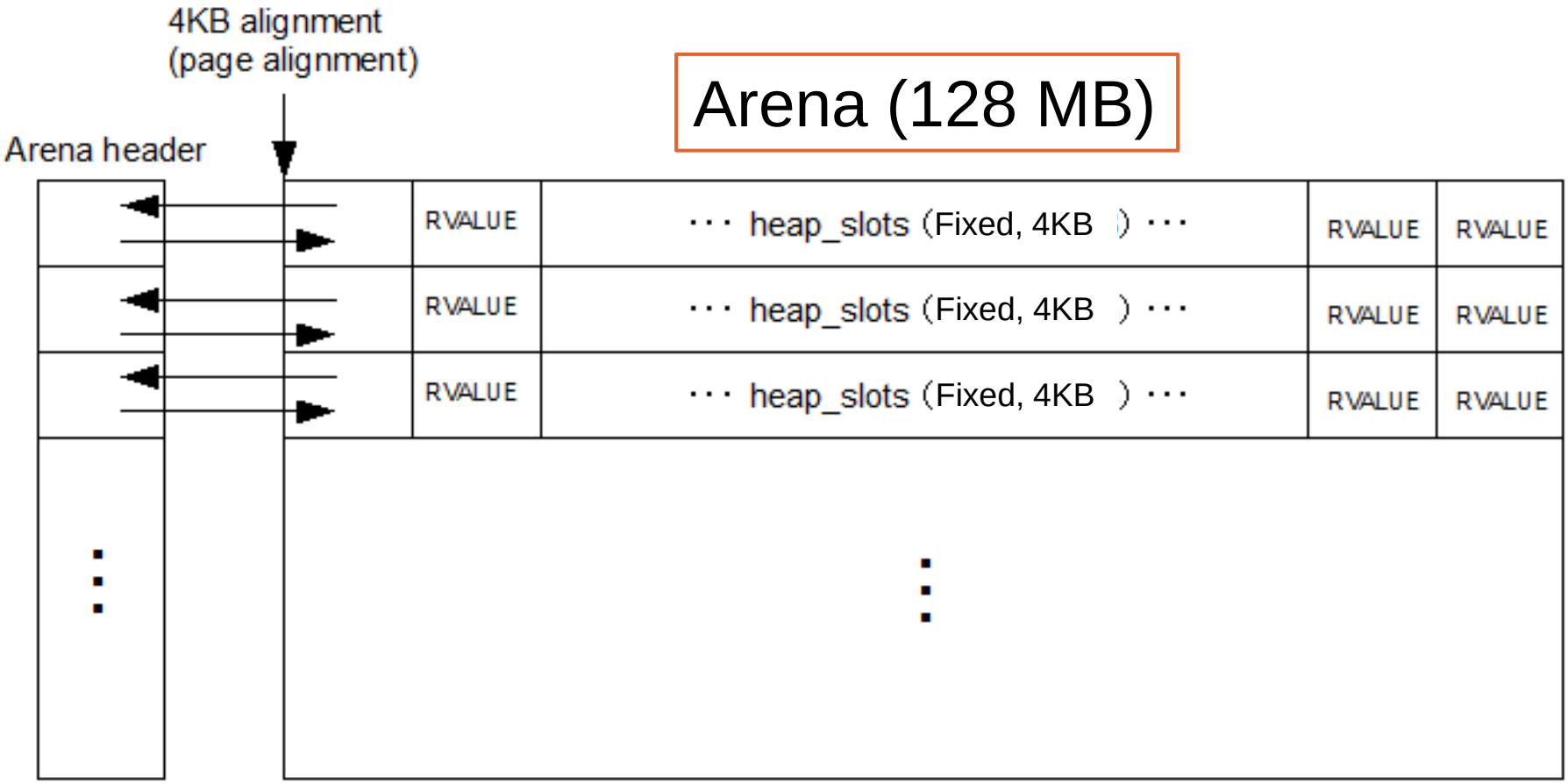
1.9 Memory Management Data Structure



Memory Management
OS dependent Memory Management

- Make more small block
- Use OS dependent memory management API
  - mmap(), munmap(), madvise() on UNIX
  - Similar APIs on Windows

1.9.2 Memory Management Data Structure



Memory Management
OS dependent Memory Management

- Many advantages
  - Low object fragmentation
  - Easy to manage with OS M/M
  - 4KB alignment page helps bitmap marking
  - 128MB bulk allocation helps is_pointer_to_heap() checking (reduce GC overhead)

Memory Management
OS dependent Memory Management
Evaluation

- Not yet
- HELP!

(2) VM Performance Improvement

1. Change VM instruction set a little
2. Fiber performance improvement

(2.1) VM instruction set changes

- Make instance variable index cache in inline cache entry

  → Need Instruction cache entry in operand of getinstance/setinstance
- Change instruction cache treatment

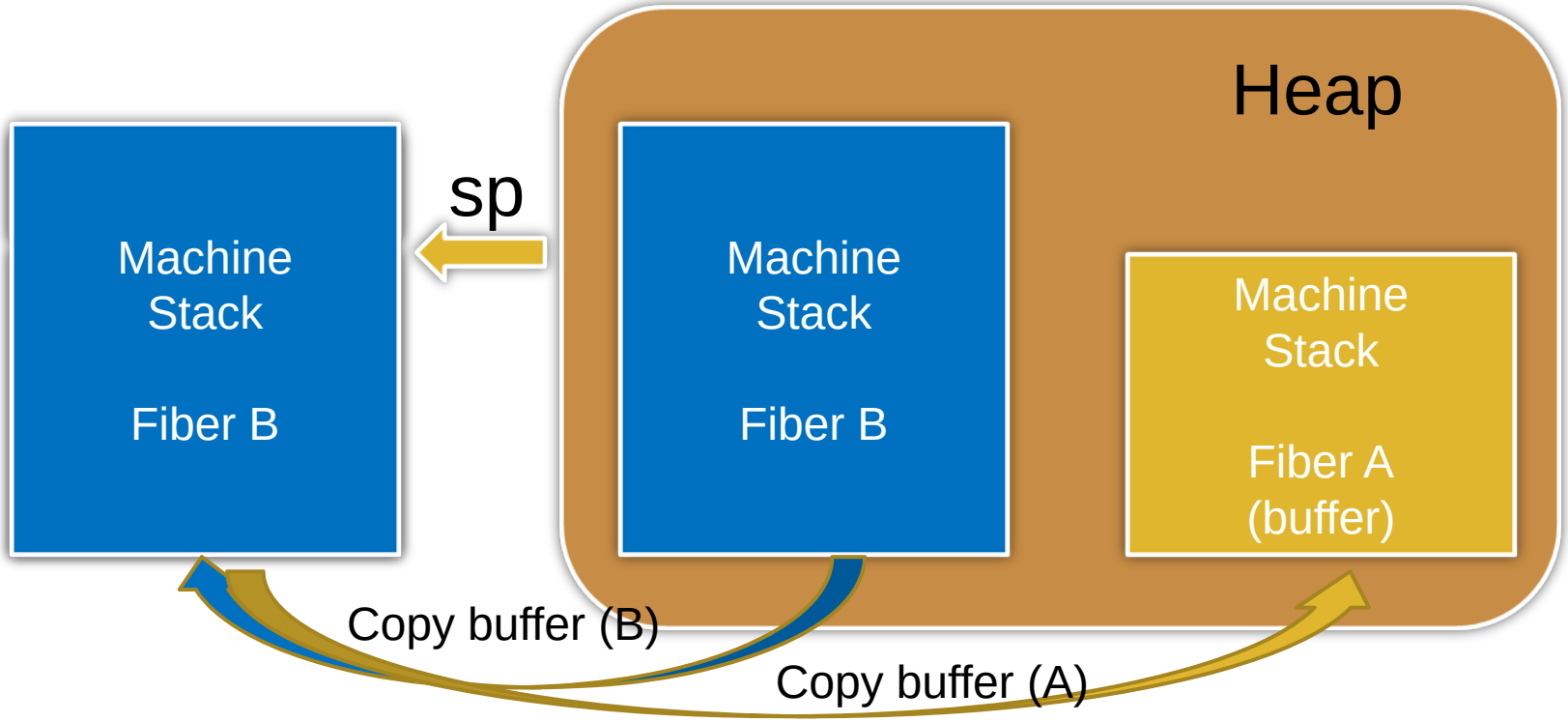
- **This point is NOTICE for YARV assembler users (… who use it?)**

(2.2) Fiber Performance Improvement

- Fiber is:
  - 1.9 Feature
  - Coroutine, Semi-Coroutine
  - Primitive for concurrent programming
  - Used in Enumerator#next
- Problem is:
  - Slow context switching

Fiber
Current implementation

- Fiber context switching needs **machine stack copy**



Fiber
Current implementation

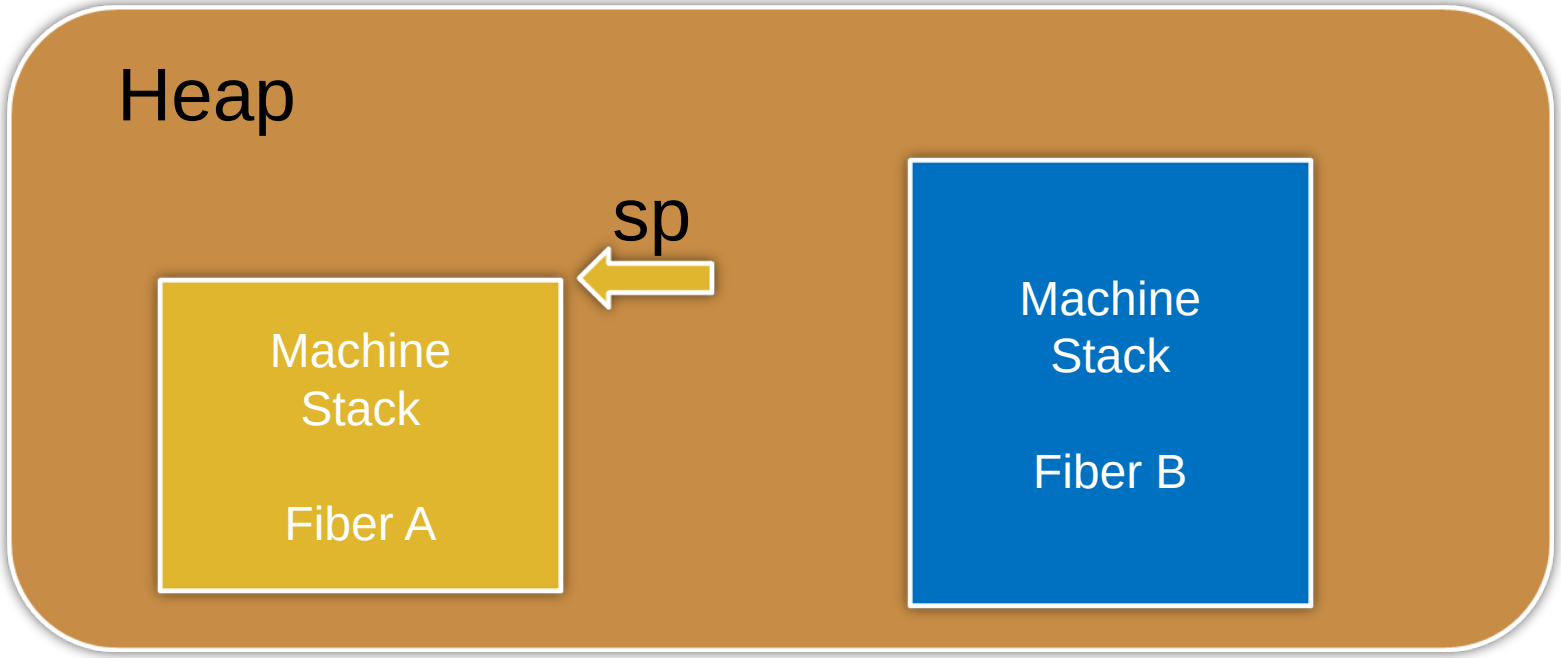
- Why such a expensive method?
  - The reason is **Portability**
  - There are no general method to switch machine stack
  - Assembler approach has portability, maintainability problems.
    - e.g. [ruby-core:14149]

Fiber performance improvement Solution

- Use OS dependent features
  - OS features are well-maintained
  - getcontext/setcontext on UNIX
    - Linux (2.6-), Solaris
      FreeBSD, Mac OSX
  - Fiber API on Windows
  - And machine stack copy for other OSs

Fiber performance improvement Switching SP with OS features

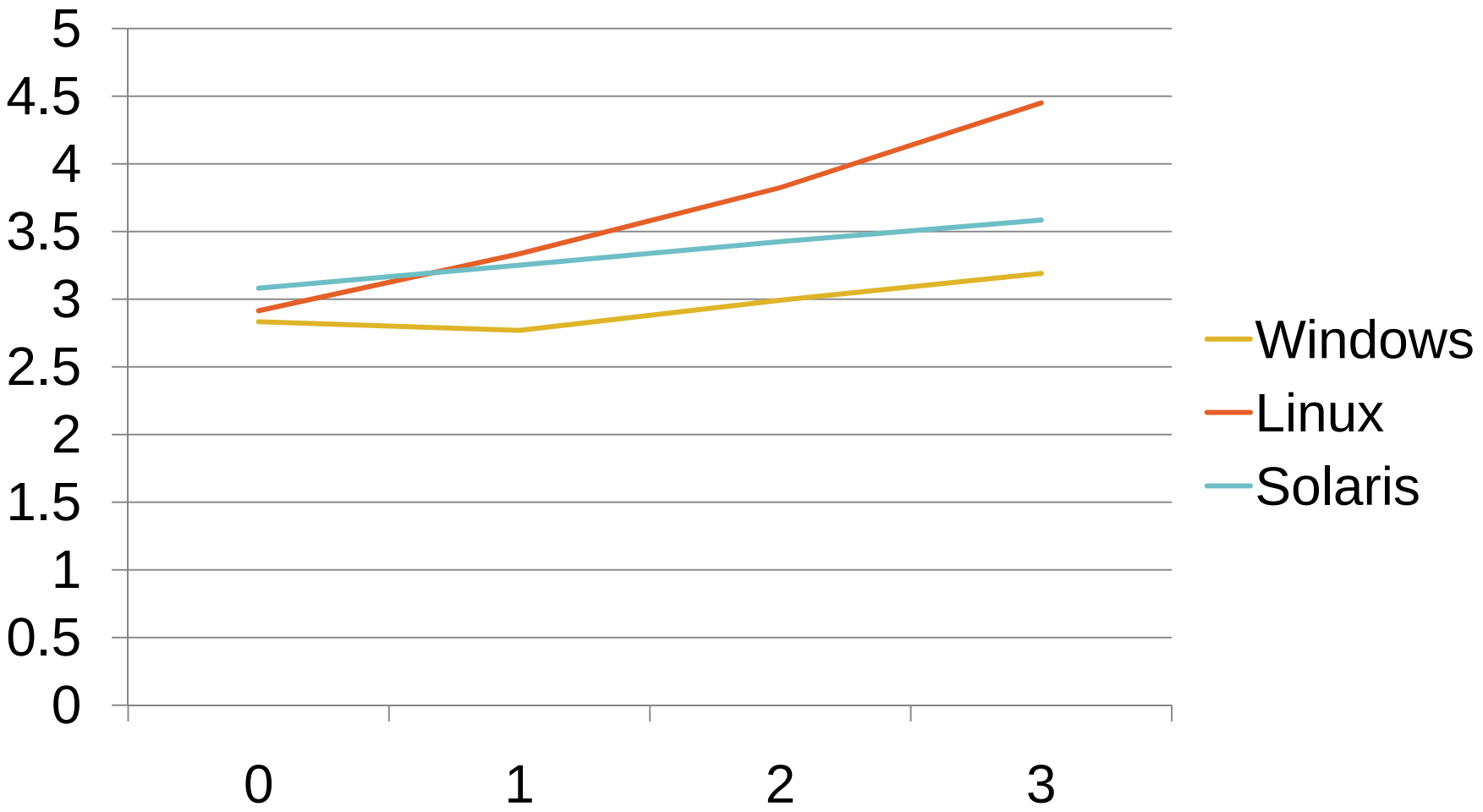
- **[1.9.2 Feature]** Lightweight fiber context switching



Fiber performance evaluation Context switch cost (sec)

Platform	Machine stack depth			
	0	1	2	3
Windows (1.9.1)	5.304	5.226	5.694	6.021
Windows (1.9.2)	1.872	1.887	1.903	1.887
Linux (1.9.1)	4.882	5.587	6.398	7.392
Linux (1.9.2)	1.675	1.675	1.673	1.661
Solaris (1.9.1)	130.995	137.903	144.062	151.819
Solaris (1.9.2)	42.507	42.408	42.056	42.357

Fiber performance evaluation
Context switch cost (speedup ratio)

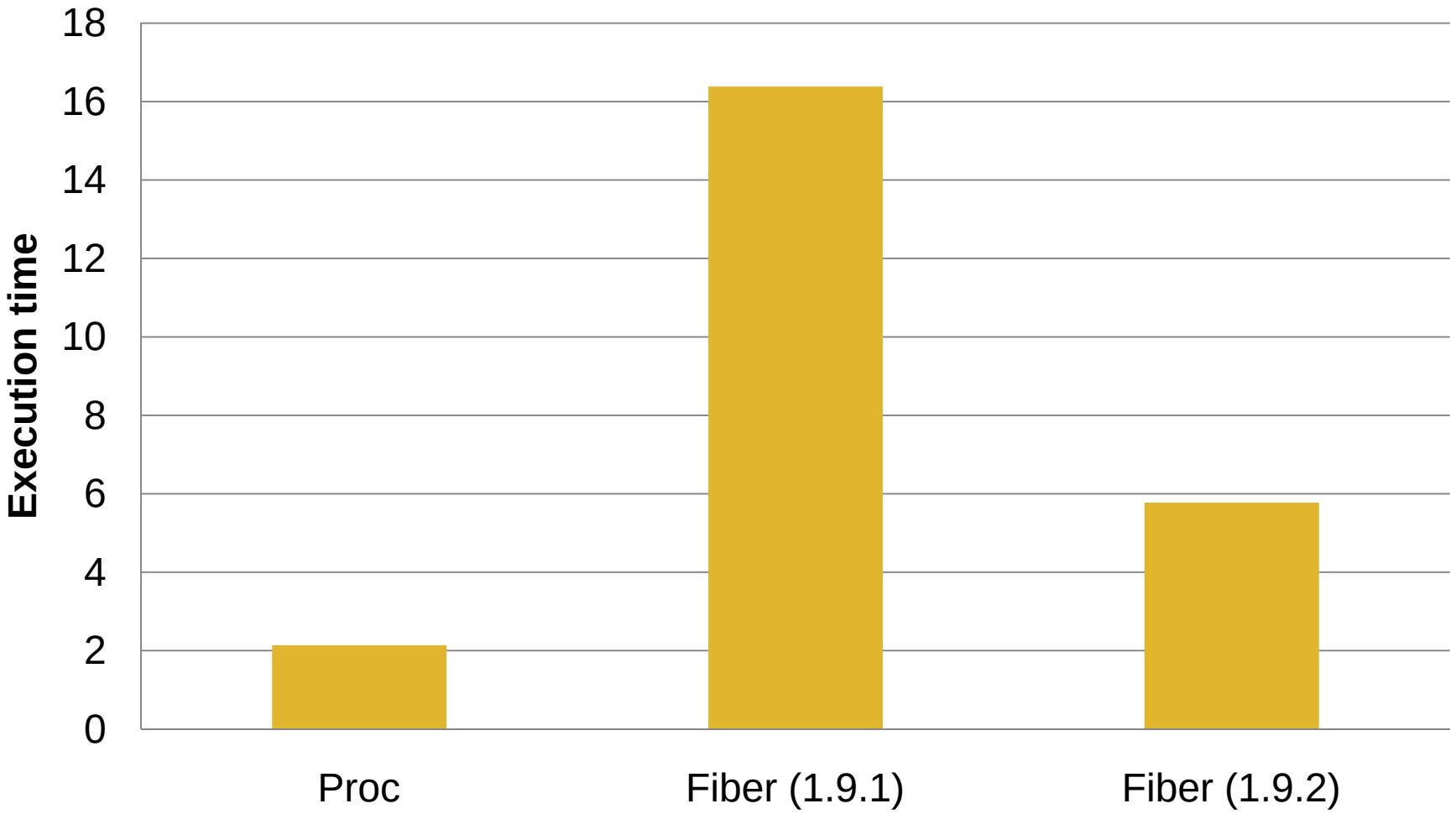


Fiber performance
Proc vs. Fiber

```
# Proc
  pr = Proc.new{};
  10000000.times{pr.call}

# Fiber
  f = Fiber.new{
    Fiber.yield while true};
  10000000.times{f.resume}
```

Fiber performance
Proc vs. Fiber



Fiber performance
Fiber generation speed

- 3000000.times do
  Fiber.new{}.resume
  end
- Ruby 1.9.1
  - User time : 13.649
  - System time:0.008
  - Real time: 13.657
- Ruby 1.9.2:
  - User time: 10.541
  - System time:1.136
  - Real time: 11.677

Most of time consumed by GC overhead.
This is future work.

in Sec

(3) Multi-VM (MVM)

- Overview
  - Motivation
  - Model
  - Challenges
  - API Design
- Progress
  - Features
  - Evaluations

(3) Multi-VM (MVM)

We are working on

Distibuted programming

- ✓ dRuby / Rinda
- ✓ Actor libraries
- ✓ MVM (multi-VM in a process)

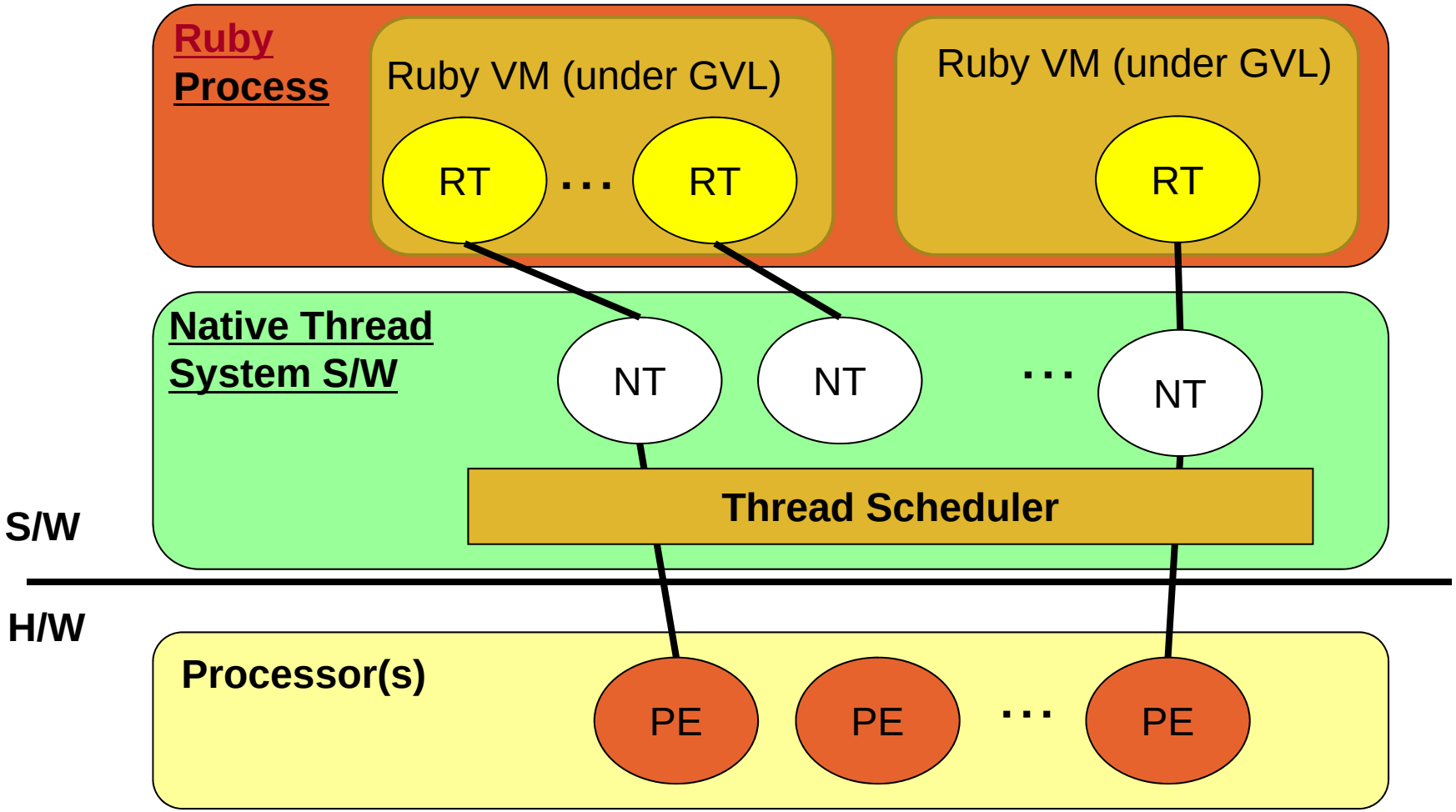
This one!

Quoted from Matz keynote

MVM
Motivation

- Parallel execution
  - Multi-core, many core, …
  - Parallel thread seems bad idea
  - **Isolated** VMs can run in parallel
- Application embedded VM
  - Easy to handle VM by your application

Model of MVM



PE: Processor Element, NT: Native Thread

MVM Challenges

- Compatibility
  - Process global information
    - C global variables
    - File related information (e.g. w/dir)
    - Process information (including signal)
    - C extension
- Communication interface design

MVM API

- C API /* include/ruby/vm.h */
  - ruby_vm_create(…)
  - ruby_vm_run(…)
  - ruby_vm_start(…)
- Ruby API # not mature
  - vm = RubyVM.new(…)
  - vm.push(…) / vm.pop() # comm.
  - Now under consideration

MVM
C Extension

- Introduce new convention
  - Init_foo(void)
  - InitVM_foo(void) ← New!
  - Do not use global variable
- If C extension doesn't have InitVM_foo() function, restricted to use by only main (first) VM

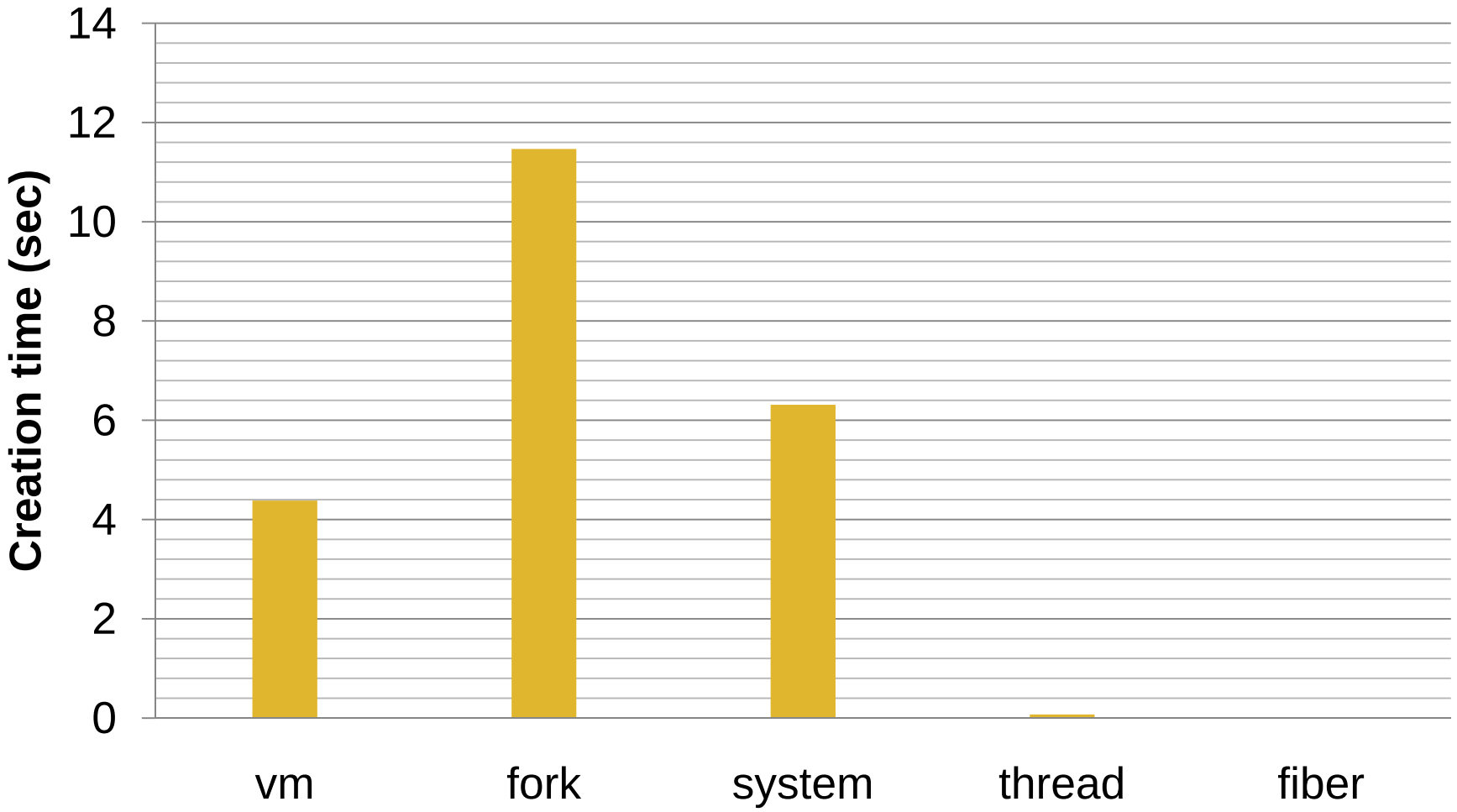
MVM Progress
Current Features

- Create VM by Ruby/C API
- Run in parallel each VMs
- Simple VM communication API
  - Not mature

MVM - Preliminary evaluation Evaluation environment

- MacBook
- Intel Core 2 Duo 2.1 GHz, 2 Cores
- L2$ 3 MB
- Memory 4 GB
- Bus speed 800 MHz

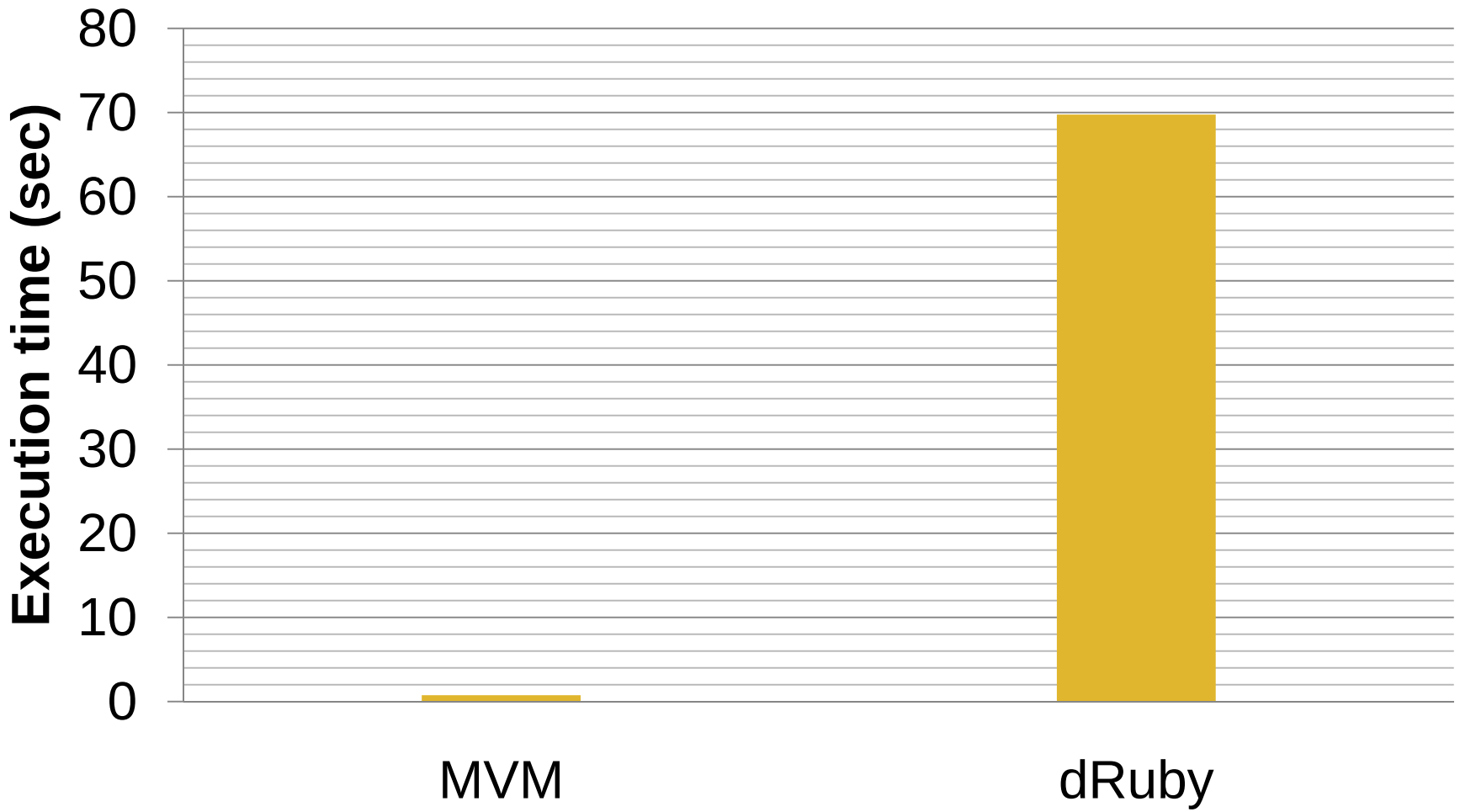
MVM - Preliminary evaluation VM creation time



MVM Progress Evaluation (ping/pong)

```
require 'benchmark'
N = ARGV[0] ? ARGV[0].to_i : 100
puts "N = #{N}"
W = 10
Benchmark.bm(W) {|bm|
  vm = RubyVM.new("ruby", "-e", %q{
    vm1 = RubyVM.current
    vm2 = RubyVM.parent
    #{N}.times {vm2.send(vm1.recv)}
  }).start
  data = :"foo"
  bm.report("send/recv") {
    N.times {|i| vm.send(data)}; vm.join
  }
}
```

MVM Progress
Evaluation (symbol ping/pong)



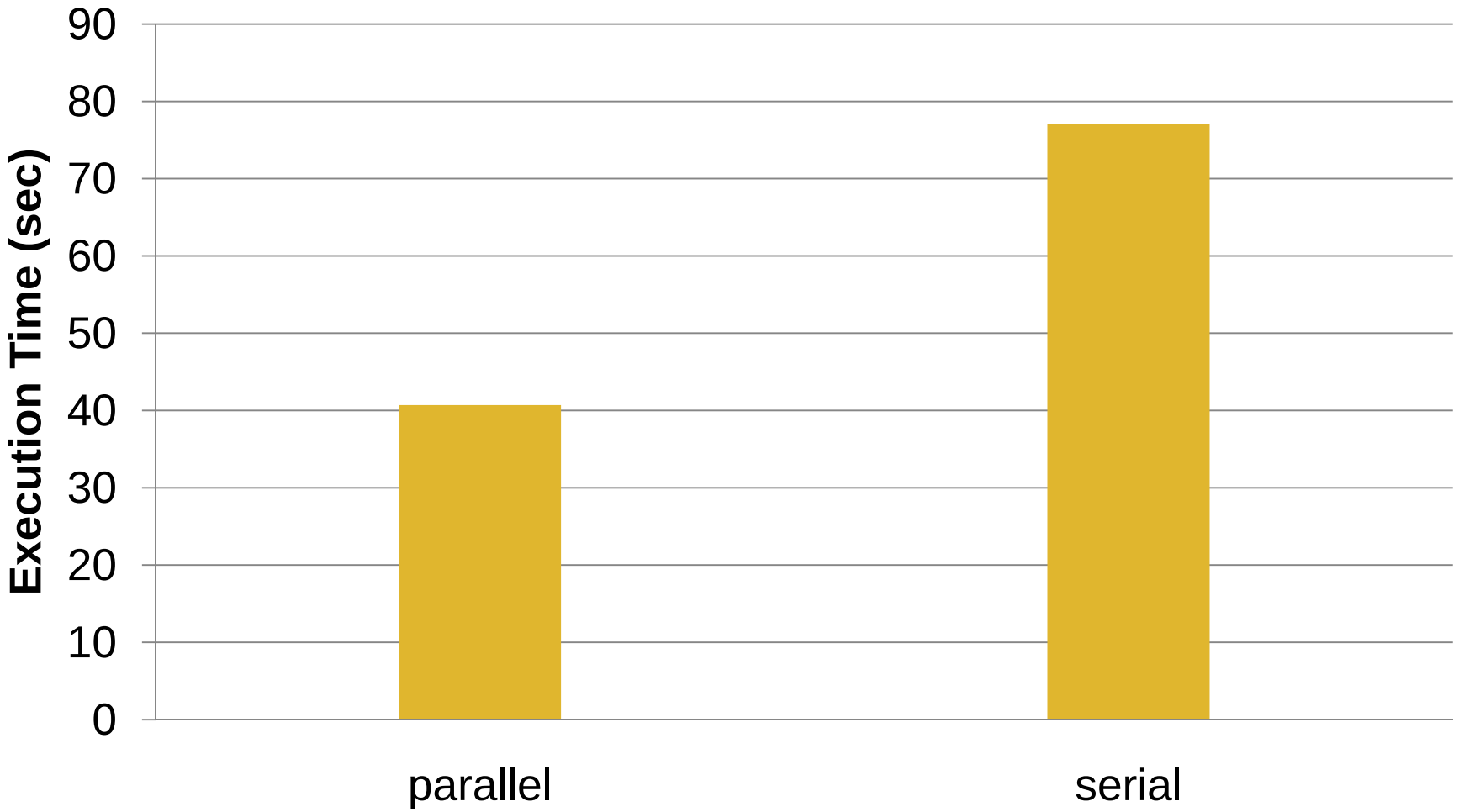
MVM Progress
Evaluation (parallel execution)

```
fibdef = <<EOS
def fib(n)
  if n<2
    n
  else
    fib(n-2)+fib(n-1)
  end
end
EOS
eval fibdef

require 'benchmark'
N = ARGV[0] ? ARGV[0].to_i : 100
F = ARGV[1] ? ARGV[1].to_i : 30
puts "N = #{N}, F = #{F}"
W = 8
```

```
Benchmark.bm(W) {|bm|
  script = fibdef + %q{
    vm1 = RubyVM.current
    vm2 = RubyVM.parent
    vm2.send(fib(vm1.recv))
  }
  bm.report("parallel") {
    (1..N).map{|i| RubyVM.new('ruby', '-e',
    script).start}.each {|vm|
      vm.send(F)
    }.each{|vm|
      vm.join
    }
  }
  bm.report("serial") {
    (1..N).map{|i| fib(F)}
  }
}
```

MVM Progress Evaluation
Parallel execution on 2 cores



Summary

- Ruby 1.9.2 may improve several performance. Stay tuned.
  - Memory management efficiencies
  - Fiber performance
  - Memory profiler feature
- Researches are progressing
  - Memory Profiler
  - MVM

Fin.

Thank You for Your Attention.

Any Questions?

SASADA Koichi

<ko1@rvm.jp>

Department of Creative Informatics,
Graduate School of Science and Technology,
The University of Tokyo

Graduate School of Information Science and Technology, The University of Tokyo

創造情報学専攻 Creative Informatics