Implementation Details of Ruby 2.0 VM

Koichi Sasada

heroku

Disclaimer

http://www.flickr.com/photos/andosteinmetz/2901325908/

- (As you can see) I can speak English little.
  - It's my 8th RubyConf
  - 7th time disclaimer
- Ask me an questions in 日本語 Japanese (**WELCOME!**), **Ruby** or **SLOW** English
- All of I want to say is on the screen. You can read them.

Who am I ?

- Koichi Sasada (笹田耕一）
  - Matz team at Heroku, Inc.
    - Full-time CRuby development
  - CRuby/MRI committer
    - Virtual machine (YARV) from Ruby 1.9
    - YARV development since 2004/1/1
  - 2.0 Release manager **assistant**
    - Organizing feature request
      - 200 tickets → 100 tickets last week!
    - Over 150 mails to ruby-core/ruby-dev
      - Sorry for spamming

PROGRAMMING Language

Ruby 2.0

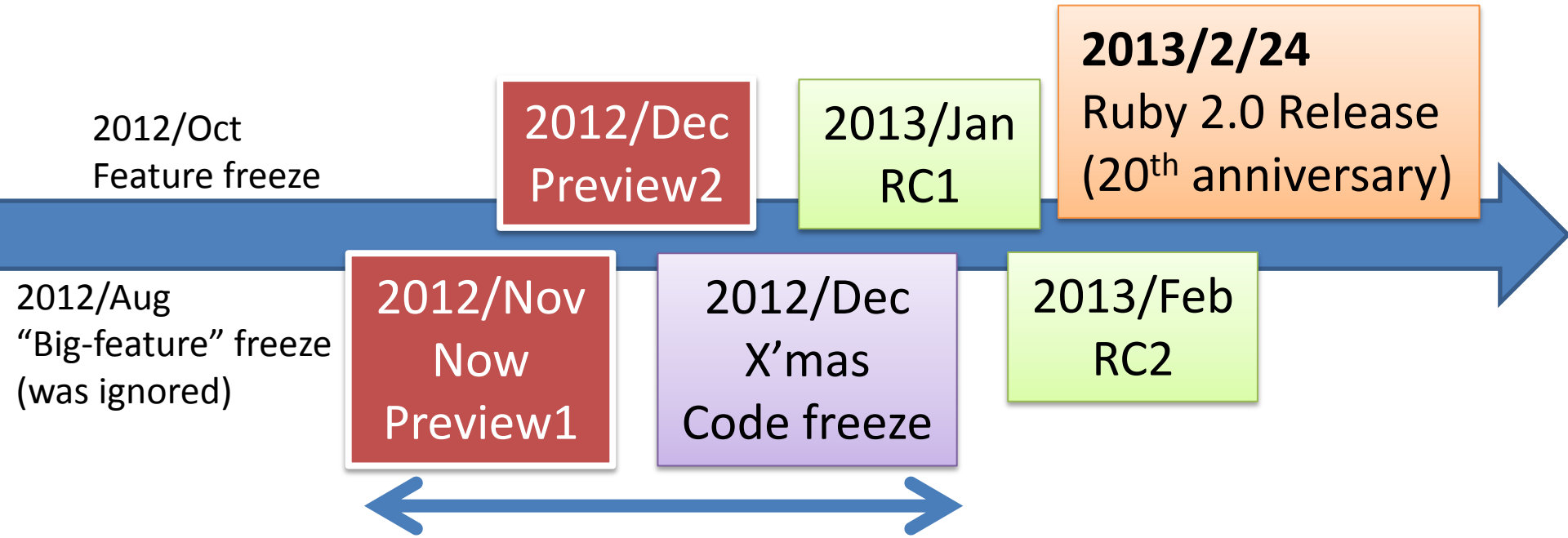
20th Anniversary Release
of Ruby language

Ruby 2.0 Release policy

- Compatibility (Ruby level)
- Compatibility (Ruby level)
- Compatibility (Ruby level)
- Usability
- Performance

ADD (Anniversary Driven Development)

Ruby 2.0 Roadmap



Only about *2 months*
to introduce new codes

“[ruby-core:40301] A rough release schedule for 2.0.0”
and Endo-san’s (release manager) leak

Introduction of Ruby 2.0 features

What is introduced?

-*- rd -*-

= NEWS

This document is a list of user visible feature changes made between releases except for bug fixes.

Note that each entry is kept so brief that no reason behind or reference information is supplied with. For a full list of changes with all sufficient information, see the ChangeLog file.

== Changes since the 1.9.3 release

=== C API updates

* NUM2SHORT() and NUM2USHORT() added. They are similar to NUM2INT, but short.

=== Library updates (outstanding ones only)

* builtin classes

  * Array
    * incompatible changes:
      * random parameter of Array#shuffle! and Array#sample now will be called with one argument, maximum value.

  * Enumerable
    * added method:
      * added Enumerable#lazy method for lazy enumeration.

  * ENV
    * aliased method:
      * ENV.to_h is a aliased method of ENV.to_hash

  * Hash
    * added method:
      * added Hash#to_h as explicit conversion method, like Array#to_a.
    * extended method:
      * Hash#default_proc= can be passed nil to clear the default proc.

  * Kernel
    * added method:
      * added Kernel#Hash conversion method like Array() or Float().
      * added Kernel#using, which imports refinements into the current scope.
      * [experimental]
    * extended method:
      * Kernel#warn accepts multiple args in like puts.
      * Kernel#caller accepts second optional argument `n' which specify required caller size.
    * incompatible changes:
      * system() and exec() closes non-standard file descriptors (The default of :close_others option is changed to true by default.)
      * respond_to? against a protected method now returns false unless the second argument is true.
      * __callee__ has returned to the original behavior, and now returns the called name but not the original name in an aliased method.
      * Kernel#inspect does not call #to_s anymore (it used to call redefined #to_s).

  * LoadError
    * added method:
      * added LoadError#path method to return the file name that could not be loaded.

  * Module
    * added method:
      * added Module#prepend which is similar to Module#include, however a method in the prepended module overrides the corresponding method in the prepending module.
      * added Module#refine, which extends a class or module locally.
      * [experimental]
      * added Module#refinements, which returns refinements defined in the receiver. [experimental]
      * added Module#using, which imports refinements into the receiver. [experimental]
    * extended method:
      * Module#define_method accepts a UnboundMethod from a Module.
      * Module#const_get accepts a qualified constant string, e.g. Object.const_get("Foo::Bar::Baz")

  * NilClass
    * added method:
      * added nil.to_h which returns {}

  * Signal
    * incompatible changes:
      * Signal.trap raises ArgumentError when :SEGV, :BUS, :ILL, :FPE, :VTALRM are specified.

  * Struct
    * added method:
      * added Struct#to_h returning values with keys corresponding to the instance variable names.

  * Thread
    * added method:
      * added Thread#thread_variable_get for getting thread local variables (these are different than Fiber local variables).
      * added Thread#thread_variable_set for setting thread local variables.
      * added Thread#thread_variables for getting a list of the thread local variable keys.
      * added Thread#thread_variable? for testing to see if a particular thread variable has been set.

  * Time
    * change return value:
      * Time#to_s returned encoding defaults to US-ASCII but automatically transcodes to Encoding.default_internal if it is set.

  * Fiber
    * incompatible changes:
      * Fiber#resume cannot resume a fiber which invokes "Fiber#transfer".

* net/http
  * new features:
    * Proxies are now automatically detected from the http_proxy environment variable. See Net::HTTP::new for details.
    * gzip and deflate compression are now requested for all requests by default. See Net::HTTP for details.
    * SSL sessions are now reused across connections for a single instance. This speeds up connection by using a previously negotiated session.
  * new methods:
    * Net::HTTP#local_host
    * Net::HTTP#local_host=
    * Net::HTTP#local_port
    * Net::HTTP#local_port=
  * extended method:
    * Net::HTTP#connect uses local_host and local_port if specified.

* net/imap
  * new methods:
    * Net::IMAP.default_port
    * Net::IMAP.default_implicit_port
    * Net::IMAP.default_tls_port
    * Net::IMAP.default_ssl_port
    * Net::IMAP.default_imaps_port

* ostruct
  * new methods:
    * OpenStruct#[], []=
    * OpenStruct#each_pair
    * OpenStruct#eql?
    * OpenStruct#hash
    * OpenStruct#to_h converts the struct to a hash.
  * extended method:
    * OpenStruct.new also accepts an OpenStruct / Struct.

* pathname
  * new method:
    * Pathname#find returns an enumerator if no block is given.

* resolv
  * new method:
    * Resolv::DNS#timeouts=
    * Resolv::DNS::Config#timeouts=

* shellwords
  * Shellwords#shellescape() now stringifies the given object using to_s.
  * Shellwords#shelljoin() accepts non-string objects in the given array, each of which is stringified using to_s.

* syslog
  * Added Syslog::Logger which provides a Logger API atop Syslog.
  * Syslog::Priority, Syslog::Level, Syslog::Option and Syslog::Macros are introduced for easy detection of available constants on a running system.

* lib/tmpdir.rb
  * incompatible changes:
    * Dir.mktmpdir uses FileUtils.remove_entry instead of FileUtils.remove_entry_secure. This means that applications should not change the permission of the directory to make it temporarily inaccessible from other users.

* zlib
  * Added streaming support for Zlib::Inflate and Zlib::Deflate. This allows processing of a stream without the use of large amounts of memory.
  * Added support for the new deflate strategies Zlib::RLE and Zlib::FIXED.
  * Zlib streams are now processed without the GVL. This allows gzip, zlib and deflate streams to be processed in parallel.

* openssl
  * Consistently raise an error when trying to encode nil values. All instances of OpenSSL::ASN1::Primitive now raise TypeError when calling to_der on an instance whose value is nil. All instances of OpenSSL::ASN1::Constructive raise NoMethodError in the same case. Constructing such values is still permitted.
  * TLS 1.1 & 1.2 support by setting OpenSSL::SSL::SSLContext#ssl_version to :TLSv1_2, :TLSv1_2_server, :TLSv1_2_client or :TLSv1_1, :TLSv1_1_server :TLSv1_1_client. The version being effectively used can be checked with OpenSSL::SSL::SSLSocket#ssl_version. Furthermore, it is also possible to blacklist the new TLS versions with OpenSSL::SSL::OP_NO_TLSv1_1 and OpenSSL::SSL::OP_NO_TLSv1_2.
  * Added OpenSSL::SSL::SSLContext#renegotiation_cb. A user-defined callback may be set which gets called whenever a new handshake is negotiated. This also allows to programmatically decline (client) renegotiation attempts.
  * Support for "0/n" splitting of records as BEAST mitigation via OpenSSL::SSL::OP_DONT_INSERT_EMPTY_FRAGMENTS.
  * OpenSSL requires decrypting PEM-encoded files to be at least four characters long. This led to awkward situations where an export with a password with fewer than four characters was possible, but accessing the file afterwards failed. OpenSSL::PKey::RSA, OpenSSL::PKey::DSA and OpenSSL::PKey::EC therefore now enforce the same check when exporting a private key to PEM with a password - it has to be at least four characters long.
  * Support for the Next Protocol Negotiation extension. Supported with OpenSSL 1.0.1 and higher.
  * OpenSSL::OPENSSL_FIPS allows client applications to detect whether OpenSSL is running in FIPS mode and to react to the special requirements this might impy.

* yaml
  * Syck has been removed. YAML now completely depends on libyaml being installed.

* objspace
  * new method:
    * ObjectSpace.reachable_objects_from(obj)

=== Language changes

  * Added %i and %I for symbol list creation (similar to %w and %W).

=== Compatibility issues (excluding feature bug fixes)

  * Signal.trap

    See above.

  * Merge Onigmo.

    https://github.com/k-takata/Onigmo

  * The :close_others option is true by default for system() and exec().

    Also, the close-on-exec flag is set by default for all new file descriptors.

    This means file descriptors doesn't inherit to spawned process unless explicitly requested such as system(..., fd=>fd).

  * Kernel#respond_to? against a protected method now returns false unless the second argument is true.

  * Dir.mktmpdir in lib/tmpdir.rb

    See above.

  * OpenStruct new methods can conflict with custom attributes named "each_pair", "eql?", "hash" or "to_h".

You can read them at NEWS file

goto :next_topic

Change the title of this presentation to...

Lecture series of Computer Science

How to make interpreter?
#3 Method dispatch

Prof. Koichi Sasada (*1)

Akihabara University (*2)

*1: Prof. means ...

*2: Of course, joking. No such University ☺

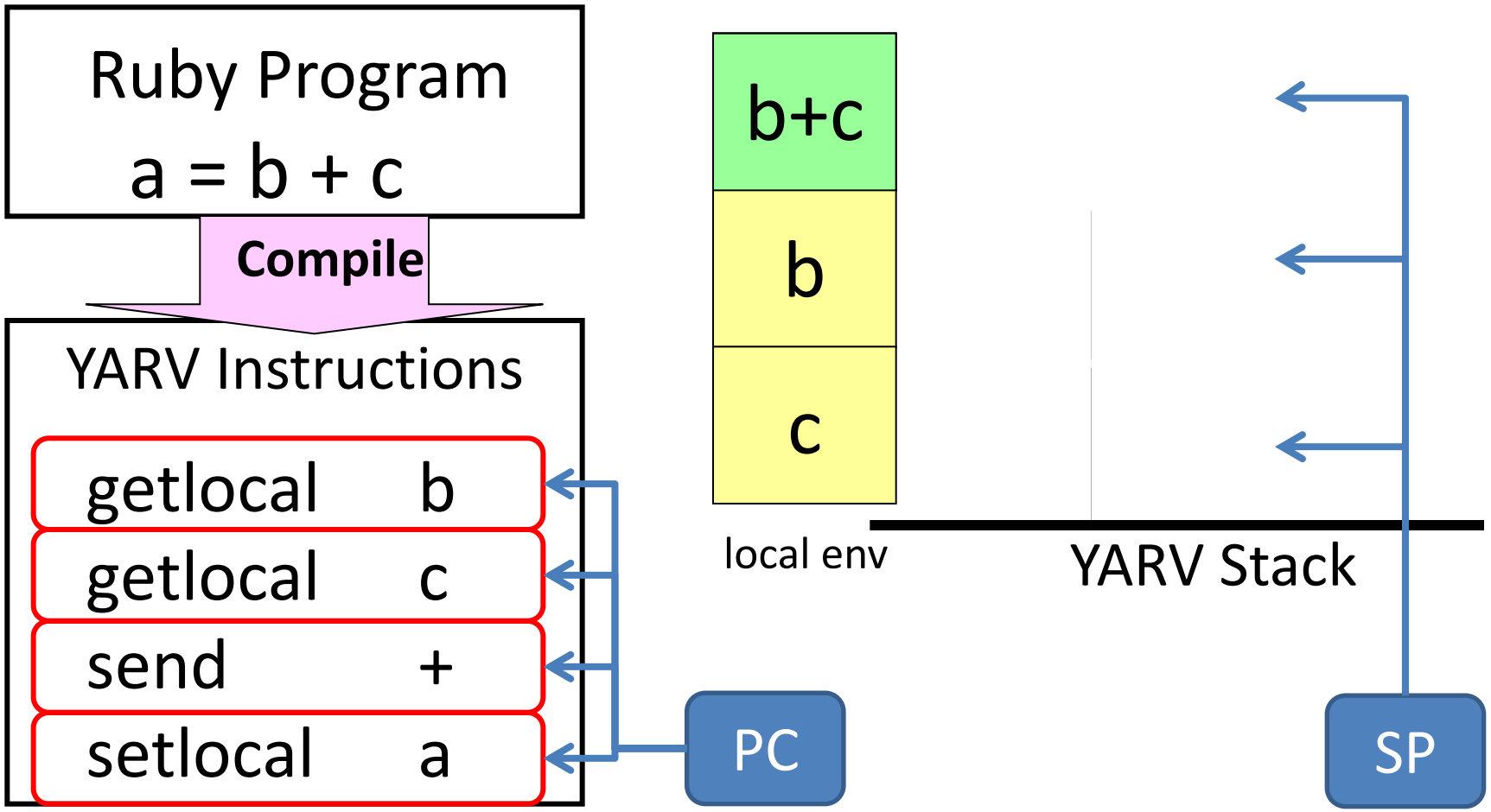
Review slide
Requirement and Assumption

- You need to finish **“Ruby language basic”** course
- This course uses “Ruby” language/interpreter
  - One of the most popular languages
  - Used in world-wide programming
    - Web application
    - Text processing
    - and everything!!
  - CRuby
    - Ruby has many alternative implementations
    - CRuby has their own VM

Review slide
How to implement virtual machine?

- Execute instructions
  - Execute compiled instructions (bytecodes)
  - Pointed by “Program counter” (PC)
- Stack machine architecture
  - All of values on the stack
  - Stack top is pointed by “Stack pointer” (SP)
  - V.S. Register machine architecture
    - Advantages and disadvantages
    - Yunhe Shi, et al: “Virtual machine showdown: stack versus registers” (2005)

Review slide
Stack machine execution (basic)

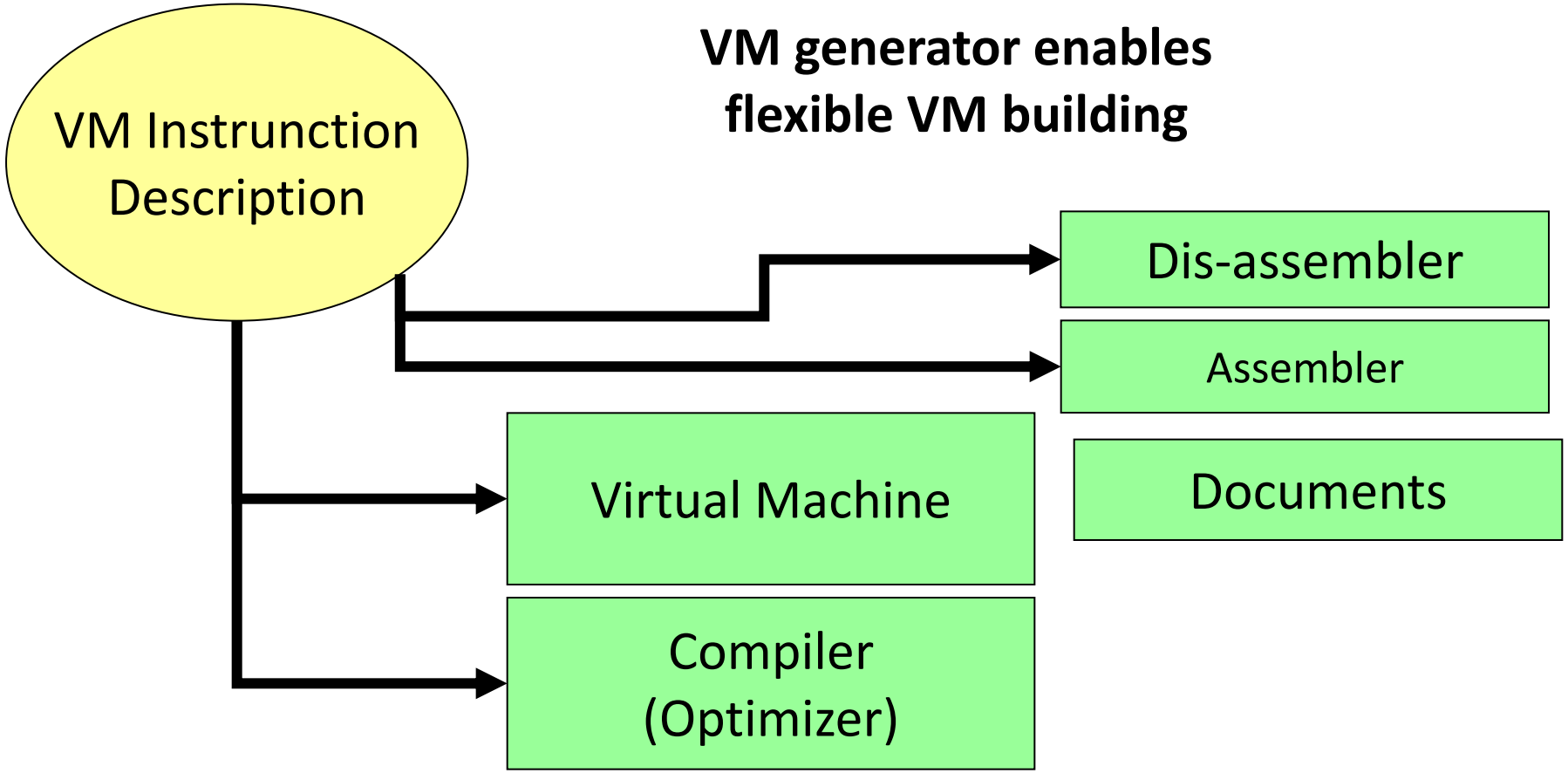


Animation with Powerpoint

Review slide
[Advanced] Optimization techniques

- Peephole optimizations (compiler technique)
  - Reduce instruction number
- Make macro instructions
  - Operand unification
  - Instruction unification
- Direct threading
  - Using GCC specific feature
- Stack caching
  - n-level stack caching
  - Impact on CPU's branch prediction

Review slide
[Advanced] VM generator

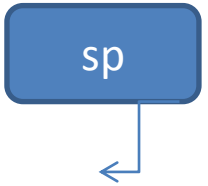


Today's lecture: Method dispatch

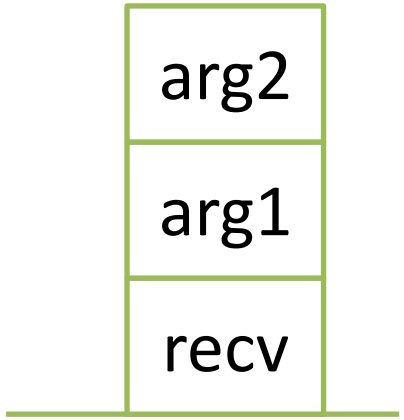
```
# Example
recv.selector(arg1, arg2)
```

- recv: receiver
- selector: method id
- arg1, arg2: arguments

Before method dispatch



1. Evaluate `recv'
2. Evaluate `arg1' and `arg2'
3. Method dispatch (`selector')



Stack after #2

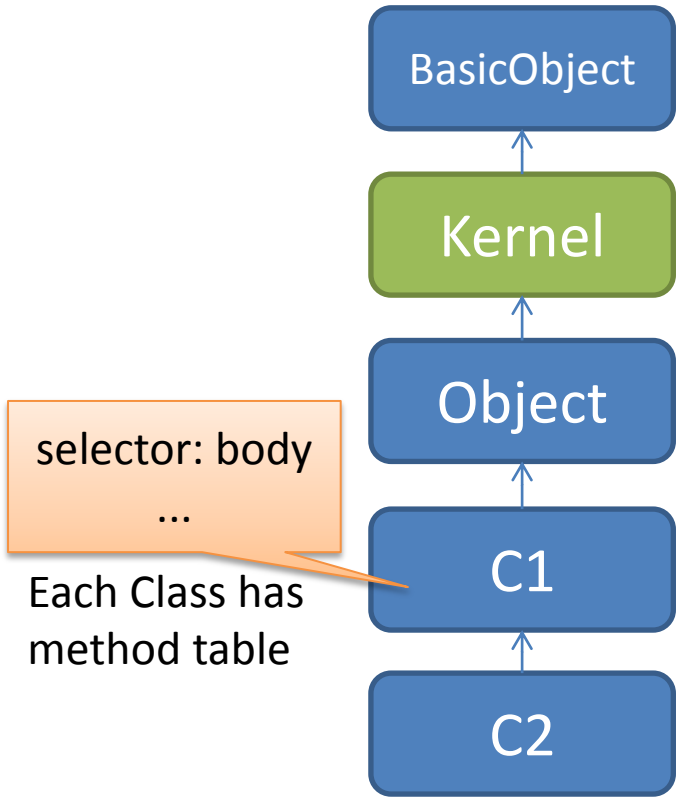
```
# Ruby's disassembled bytecodes of Ruby 2.0 trunk
0016 getlocal       recv, 0   # 1 receiver
0019 getlocal       arg1, 0  # 2 arg1
0022 getlocal       arg2, 0  # 2 arg2
0025 send           <callinfo!mid:selector, argc:2, ARGS_SKIP>
```

Method dispatch Overview

1. Get class of `recv' (`klass')
2. Search method `body' named `selector' from `klass'
   - Method is not fixed at compile time
   - **"Dynamic"** method dispatch
3. Dispatch method with `body'
   1. Check visibility
   2. Check arity (expected args # and given args #)
   3. Store `PC' and `SP' to continue after method returning
   4. Build `local environment'
   5. Set program counter
4. And continue VM execution

Overview
Method search

- Search method from `klass'
  1. Search method table of `klass'
     1. if method `body' is found, return `body'
     2. `klass' = super class of `klass' and repeat it
  2. If no method is given, exceptional flow
     - In Ruby language, `method_missing' will be called



Overview
Cheking arity and visibility

- Checking arity
  - Compare with given argument number and expected argument number
- Checking visibility
  - In Ruby language, there are three visibilities (can you explain each of them ?:-p)
    - public
    - private
    - protected

Overview
Building `local environment’

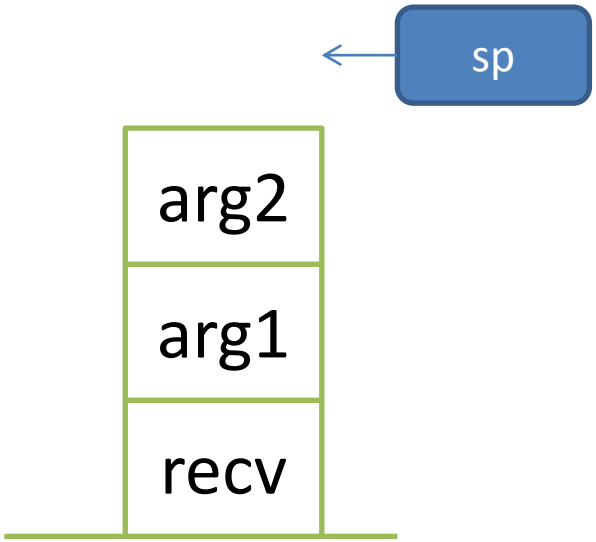
- How to maintain local variables?

→ Prepare `local variables space’ in stack

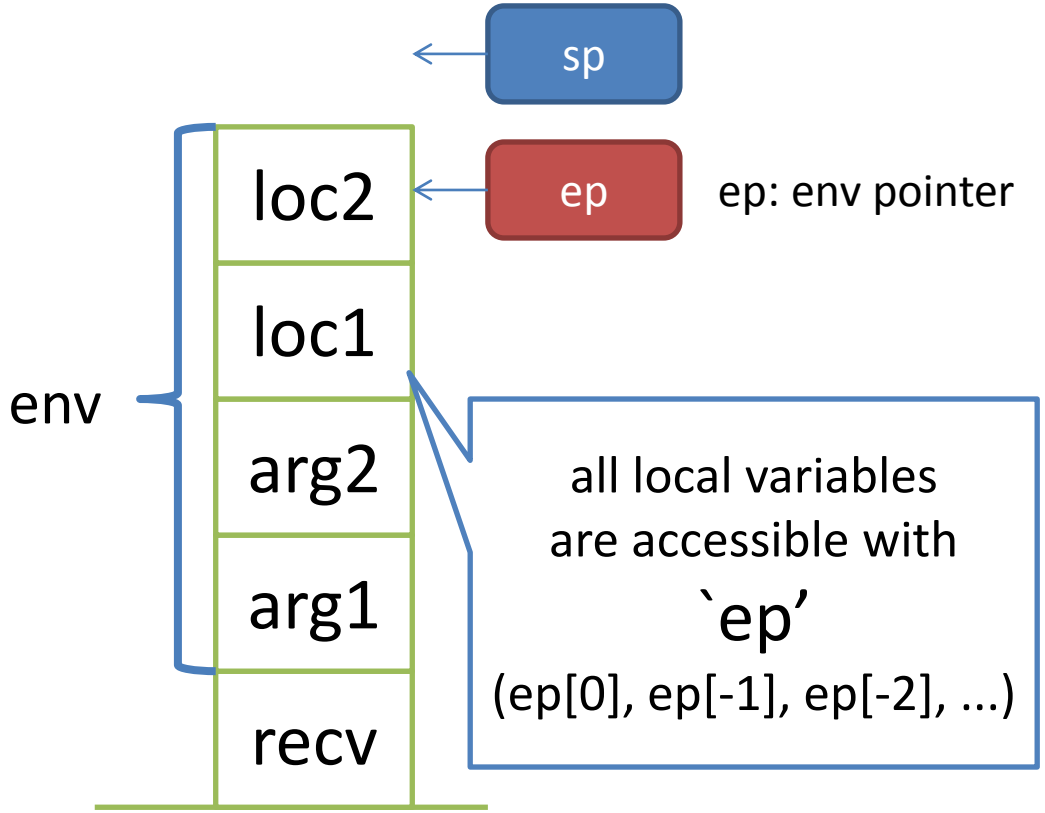
→ `local environment’ (short `env’)

- Parameters are also in `env’

Overview
Building `local environment'



Stack before method dispatch



Stack after method dispatch

Method dispatch Overview (again)

1. Get class of `recv' (`klass')
2. **Search method `body' `selector' from `klass'**
   - Method is not fixed at compile time
   - **"Dynamic"** method dispatch
3. **Dispatch method with `body'**
   1. Check visibility
   2. Check arity (expected args # and given args #)
   3. Store `PC' and `SP' to continue after method returning
   4. Build `local environment'
   5. Set program counter
4. And continue VM execution

About 7 steps

It seems very **easy and simple!** and slow...

Method dispatch
Ruby's case

- Quiz: How many steps in Ruby's case?
  - Hint: More complex than I explained overview

① 8 steps

② 12 steps

③ 16 steps

④ 20 steps

Answer is
About ④ 20 steps

Method dispatch
Ruby's case

1. Check caller's arguments
   1. Check splat (*args)
   2. Check block (given by compile time or block parameter (&block))
2. Get class of `recv' (`klass')
3. **Search method `body' `selector' from `klass'**
   – Method is not fixed at compile time
   – **"Dynamic"** method dispatch
4. **Dispatch method with `body'**
   1. Check visibility
   2. Check arity (expected args # and given args #) and process
      1. Post arguments
      2. Optional arguments
      3. Rest argument
      4. Keyword arguments
      5. Block argument
   3. Push new control frame
      1. Store `PC' and `SP' to continue after method returning
      2. Store `block information'
      3. Store `defined class'
      4. Store bytecode info (iseq)
      5. Store recv as self
   4. Build `local environment'
   5. Initialize local variables by `nil'
   6. Set program counter
5. And continue VM execution

(*) Underlined items are additonal process

Ruby's case
4. Dispatch method with `body'

- Previous explanation is for Ruby methods
  - `body' (defined as rb_method_definition_t in method.h) has several types at least the following two types:
    - Method defined by Ruby code
    - Method defined by C function (in C-extension)
- Quiz: How many method types in CRuby?
  - Hint: At least 2 types (Ruby method and C method)

① 3 types
② 6 types
③ 9 types
④ 11 types

Answer is
About ④ 11 types

Ruby's case
Method types

1. VM_METHOD_TYPE_ISEQ: Ruby method (using `def' keyword)
2. VM_METHOD_TYPE_CFUNC: C method
3. VM_METHOD_TYPE_ATTRSET: defined by @attr_accessor
4. VM_METHOD_TYPE_IVAR: defined by @attr_reader
5. VM_METHOD_TYPE_BMETHOD: defind by `define_method'
6. VM_METHOD_TYPE_ZSUPER: used in internal
7. VM_METHOD_TYPE_UNDEF: `undef'ed method
8. VM_METHOD_TYPE_NOTIMPLEMENTED: not implemet
9. VM_METHOD_TYPE_OPTIMIZED: optimization
10. VM_METHOD_TYPE_MISSING: method_missing type
11. VM_METHOD_TYPE_CFUNC_FRAMELESS: optimization two

There are 11th different method dispatch procedure (dispatch by switch/case statement)

Ruby's case

- Quiz: I introduce (virtual) registers `pc', `sp' and `ep'. How many registers in virtual machine (in Ruby 1.9.x)?
  - ① 4 registers
  - ② 6 registers
  - ③ 9 registers
  - ④ 11 registers

Answer is
About ④ 11 registers
↓
Need to store/restore
11 registers
each method call

Ruby's case
Store registers

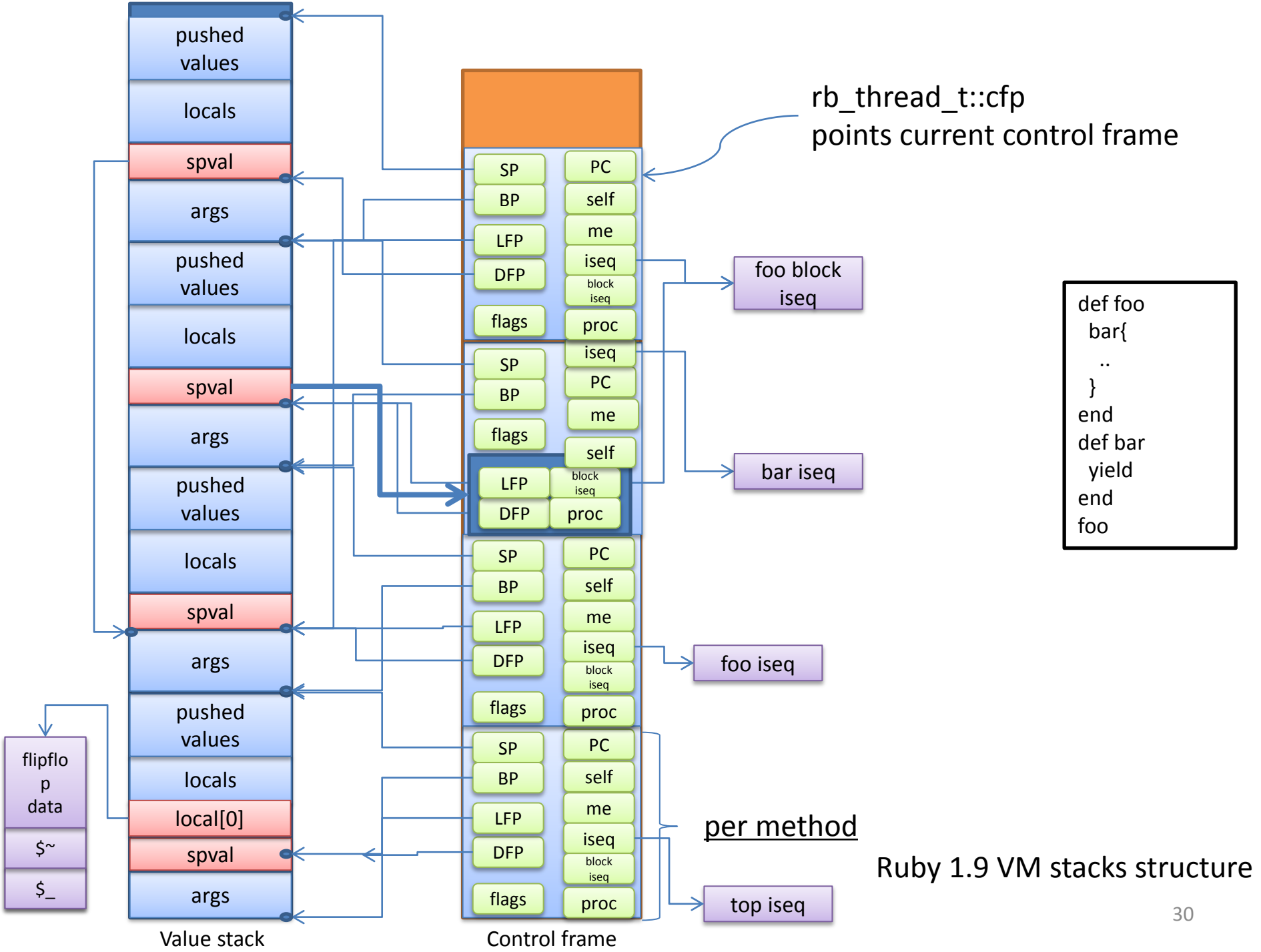
- Introduce "control frame stack" to store registers
  - To store `pc', `sp', `ep' and other information, VM has another stack named "control frame stack"
  - Not required structure, but it makes VM simple → Easy to maintain

11 regs

```
/* 1.9.3 */
typedef struct {
    VALUE *pc;                          /* cfp[0] */
    VALUE *sp;                          /* cfp[1] */
    VALUE *bp;                          /* cfp[2] */
    rb_iseq_t *iseq;                    /* cfp[3] */
    VALUE flag;                         /* cfp[4] */
    VALUE self;                         /* cfp[5] / block[0] */
    VALUE *lfp;                         /* cfp[6] / block[1] */
    VALUE *dfp;                         /* cfp[7] / block[2] */
    rb_iseq_t *block_iseq;      /* cfp[8] / block[3] */
    VALUE proc;                         /* cfp[9] / block[4] */
    const rb_method_entry_t *me;/* cfp[10] */
} rb_control_frame_t;
```

10 regs
reduced, but many yet

```
/* 2.0 */
typedef struct {
    VALUE *pc;                          /* cfp[0] */
    VALUE *sp;                          /* cfp[1] */
    rb_iseq_t *iseq;                    /* cfp[2] */
    VALUE flag;                         /* cfp[3] */
    VALUE self;                         /* cfp[4] / block[0] */
    VALUE klass;                /* cfp[5] / block[1] */
    VALUE *ep;                          /* cfp[6] / block[2] */
    rb_iseq_t *block_iseq;      /* cfp[7] / block[3] */
    VALUE proc;                         /* cfp[8] / block[4] */
    const rb_method_entry_t *me;/* cfp[9] */
}
```



Ruby 1.9 VM stacks structure

Ruby's case
Complex parameter checking

- "def foo(m1, m2, o1=..., o2=...,
  p1, p2, *rest, &block)"
  - m1, m2: mandatory parameter
  - o1, o2: optional parameter
  - p1, p2: post parameter
  - rest: rest parameter
  - block: block parameter
- From Ruby 2.0, keyword parameter is supported

Method dispatch
Ruby's case

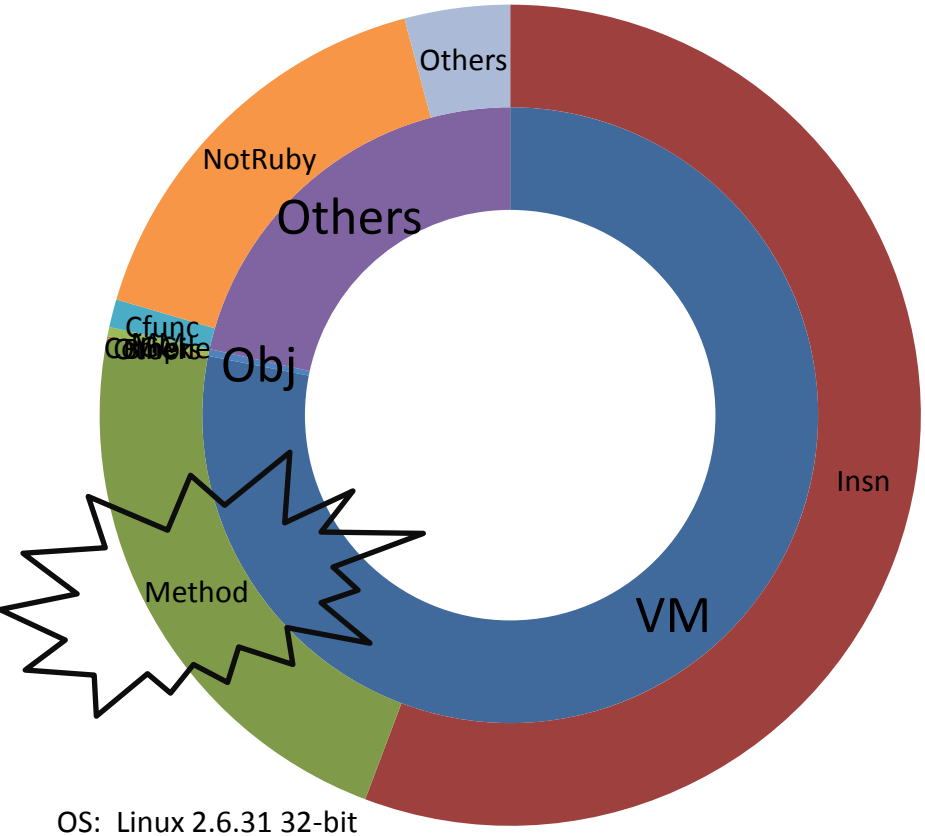
1. CHeck caller's arguments
   1. Check splat (*args)
   2. Check block (given by compile time or block parameter (&block))
2. Get class of `recv' (`klass')
3. **Search method `body' `selector' from `klass'**
   - Method is not fixed at compile time
   - **"Dynamic"** method dispatch
4. **Dispatch method with `body'**
   1. Check visibility
   2. Check arity (expected args # and given args #) and process
      1. Post arguments
      2. Optional arguments
      3. Rest argument
      4. Keyword arguments
      5. Block argument
   3. Push new control frame
      1. Store `PC' and `SP' to continue after method returning
      2. Store `block information'
      3. Store `defined class'
      4. Store bytecode info (iseq)
      5. Store recv as self
   4. Build `local environment'
   5. Initialize local variables by `nil'
   6. Set program counter
5. And continue VM execution

Complex and Slow!!!

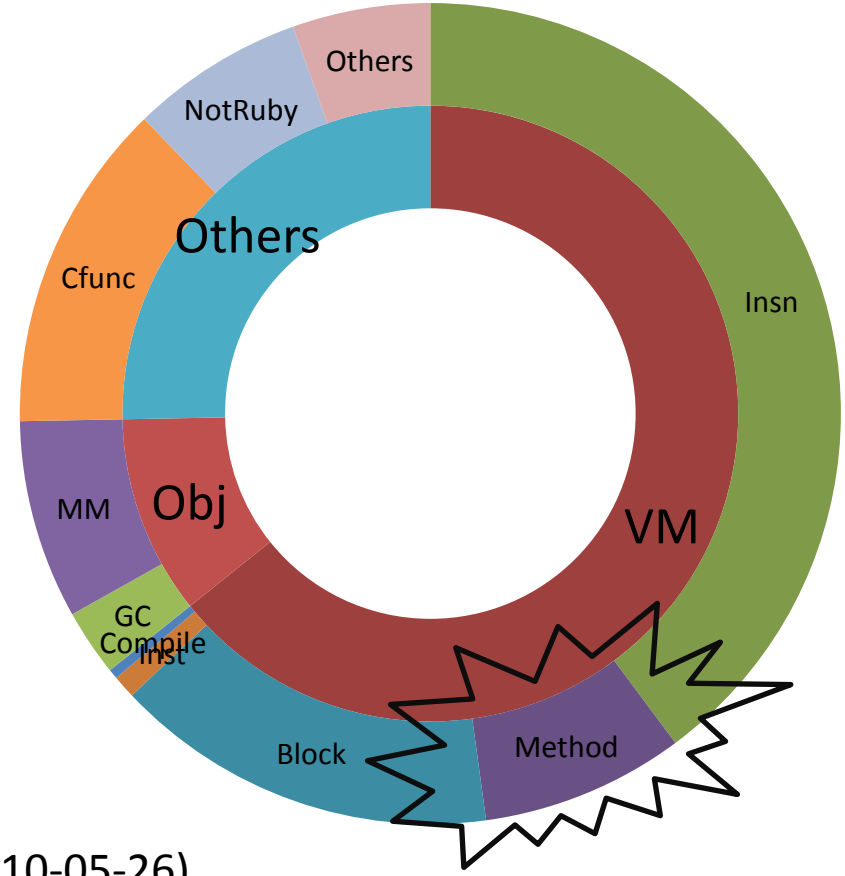
Method dispatch Overhead

Method dispatch overhead is big especially on micro-benchmarks ☺

Fib



Pentomino



OS: Linux 2.6.31 32-bit
CPU: IntelCore2Quad 2.66GHz
Mem: 4GB
C Compiler: GCC 4.4.1, -O3
Profiled by Oprofile

ruby 1.9.3dev (2010-05-26)
Profiled by Mr. Shiba

Homework

- Report about “Method Dispatch speedup techniques”
  1. Analyze method dispatch overhead on your favorite application
  2. Survey method dispatch speed-up techniques
  3. Propose your optimization techniques to improve method dispatch performance
  4. Implement techniques and evaluate their performance

- Deadline: 2012/11/15 (Thu) 23:59 JST
- Submit to: Koichi Sasada <ko1@rvm.jp>
- This report is important for your grade of this course!

Lecture was finished ☺

Presentation is not finished

Back to the presentation “Implementation Details of Ruby 2.0 VM”

Report “Optimization techniques for Ruby’s method dispatch”

Koichi Sasada

Speedup techniques for method dispatch

1. Specialized instructions
2. Method caching
3. Caching checking results
4. Frameless CFUNC method
5. Special path for `send' and `method_missing'

Note that these optimizations may not be my original.

Introduced techniques from Ruby 2.0
Today's main subject ☺

Method dispatch overheads

1. Check caller’s arguments
2. Search method `body’ `selector’ from `klass’
3. Dispatch method with `body’
   1. Check visibility and arity
   2. Push new control frame
   3. Build `local environment’
   4. Initialize local variables by `nil’

Optimization
Specialized instruction (from 1.9)

- Make special VM instruction for several methods
  - +, -, *, /, ...

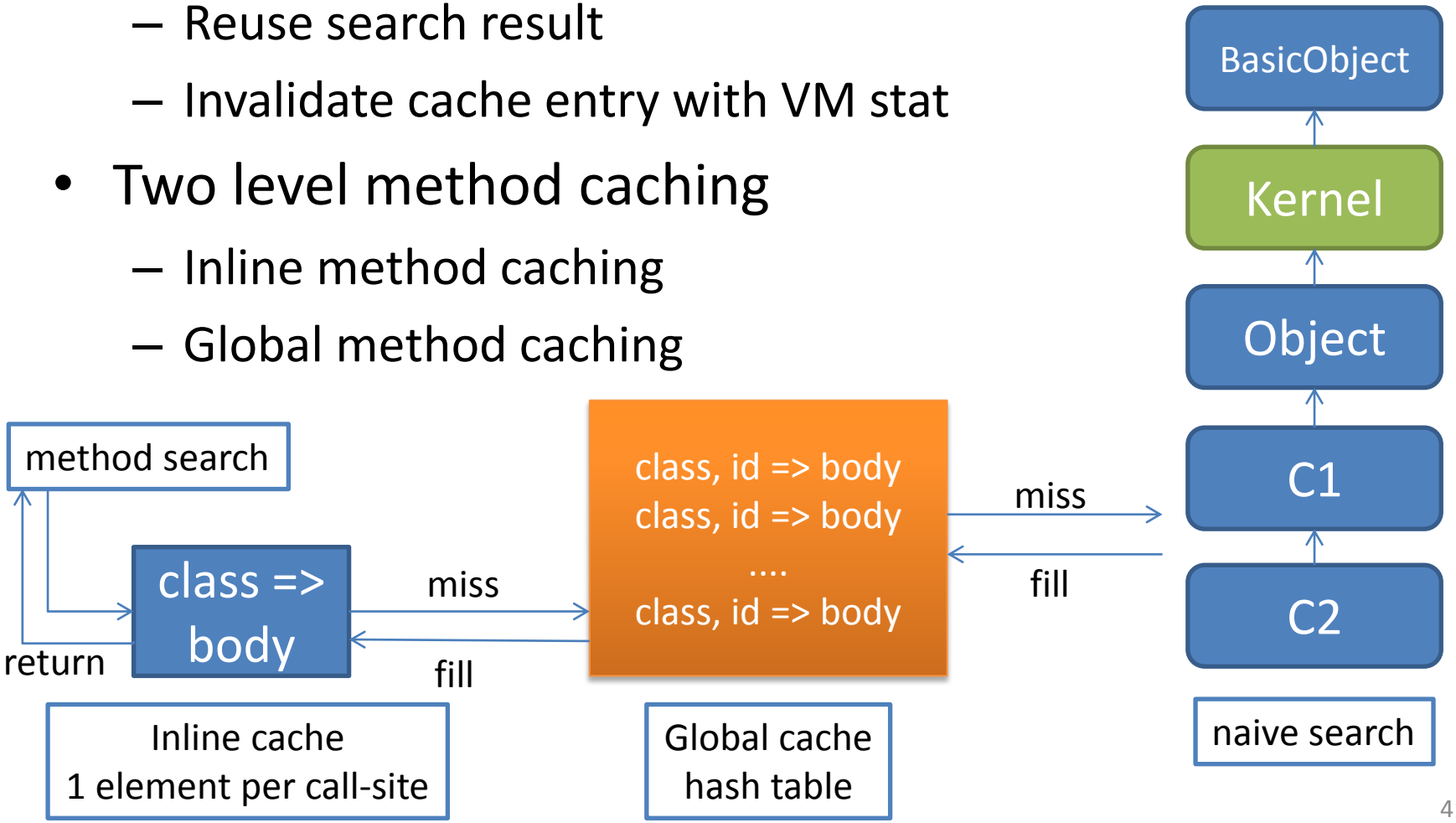
```
def opt_plus(recv, obj)
  if recv.is_a(Fixnum) and obj.is_a(Fixnum) and
      Fixnum#+ is not redefined
      return Fixnum.plus(recv, obj)
  else
    return recv.send(:+, obj) # not prepared
  end
end
```

Optimization
Specialized instruction

- Pros.
  - Eliminate all of dispatch cost **(very effective)**
- Cons.
  - Limited applicability
    - Limited classes, limited selectors
    - Tradeoff of VM instruction numbers
  - Additional overhead when not prepared class

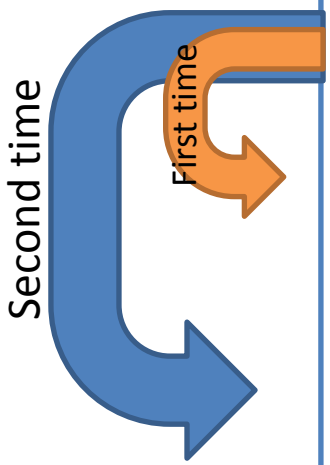
Optimization
Method caching

- **Eliminate method search overhead**
  - Reuse search result
  - Invalidate cache entry with VM stat
- Two level method caching
  - Inline method caching
  - Global method caching



Optimization
Caching checking results (from 2.0)

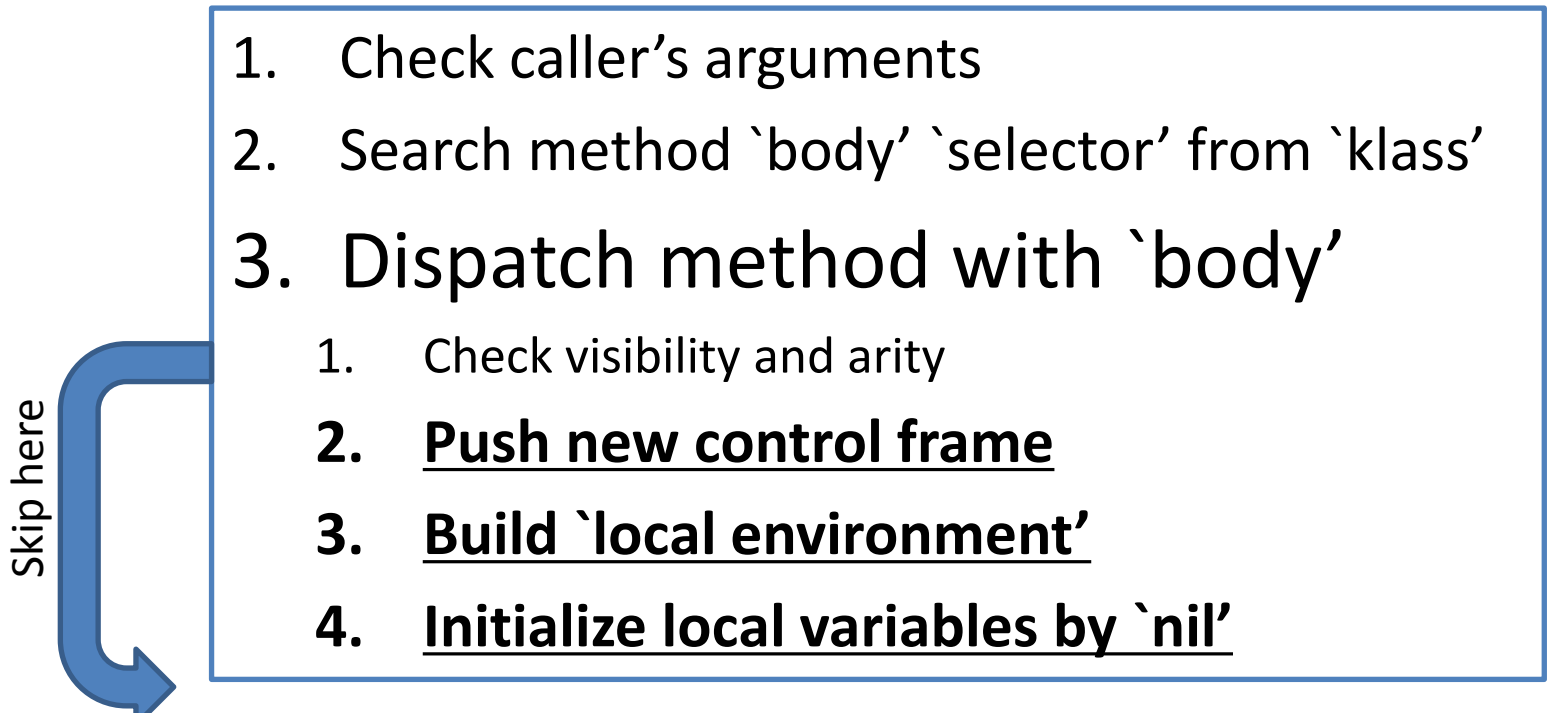
- Idea: Visibility and arity check can be skipped after first checking
  - Store result in inline method cache



1. Check caller's arguments
2. Search method `body' `selector' from `klass'
3. Dispatch method with `body'
   1. **Check visibility and arity**
      1. **Cache result into inline method cache**
   2. Push new control frame
   3. Build `local environment'
   4. Initialize local variables by `nil'

Optimization
Frameless CFUNC (from 2.0)

- Introduce “Frameless” CFUNC methods
  - Idea: Several CFUNC doesn’t need method frame
    - For example, String#length doesn’t need method frame. It only return the size of given String

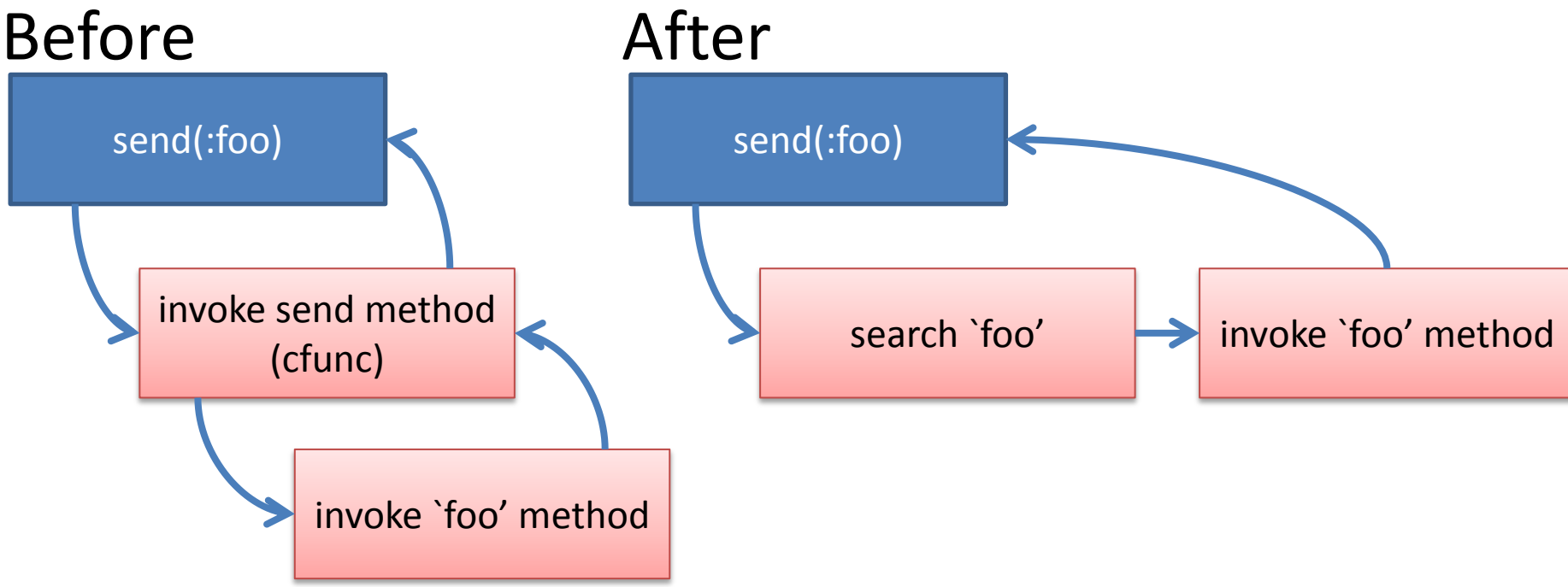


Optimization
Eliminate frame building (from 2.0)

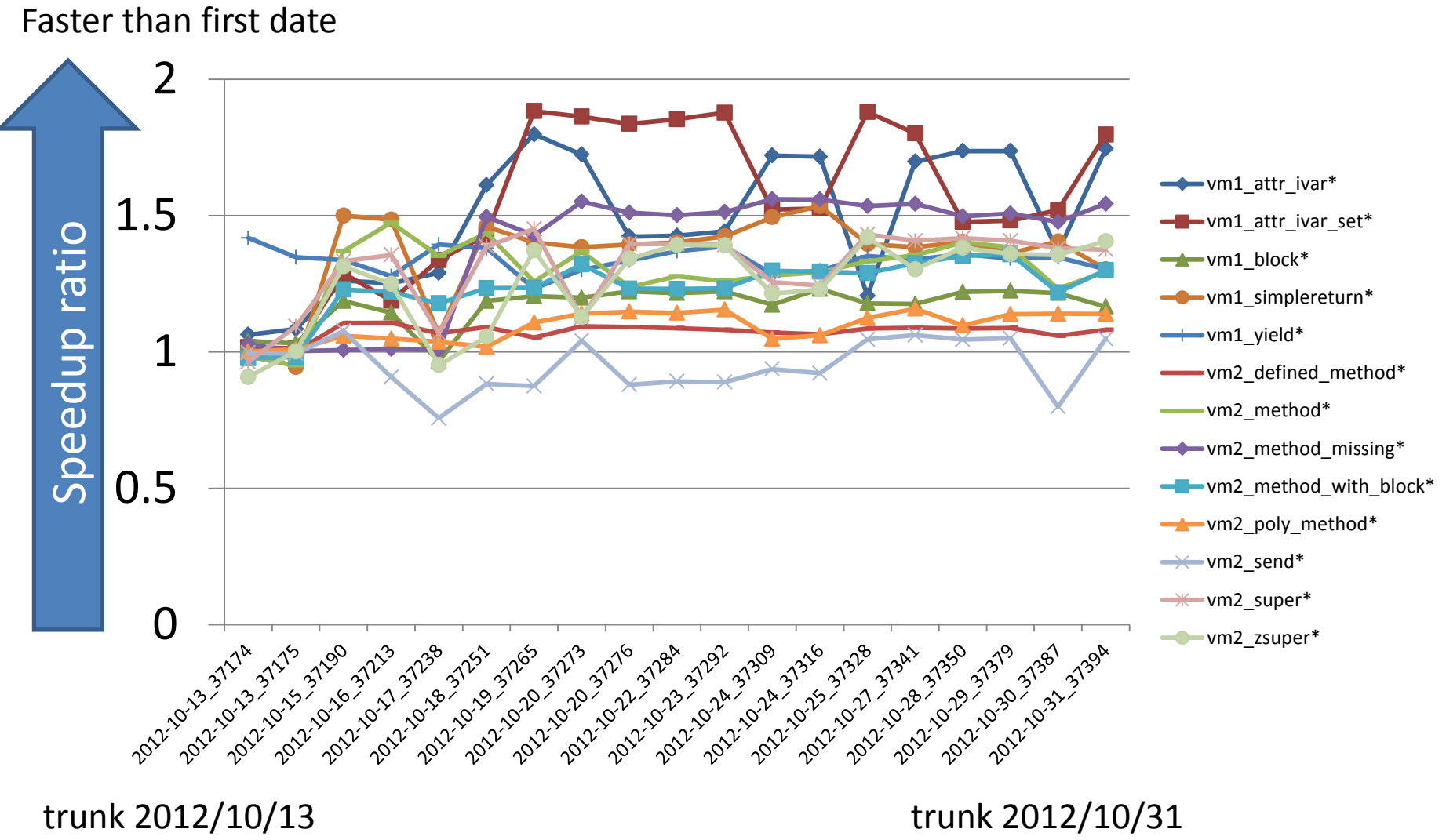
- Compare with specialized instruction
  - Pros.
    - You can define unlimited number of frameless methods
  - Cons.
    - A bit slow compare with specialized instruction
- Note that evaluation result I will show you doesn't include this technique

Optimization

Special path for `send' and `method_missing' (from2.0)



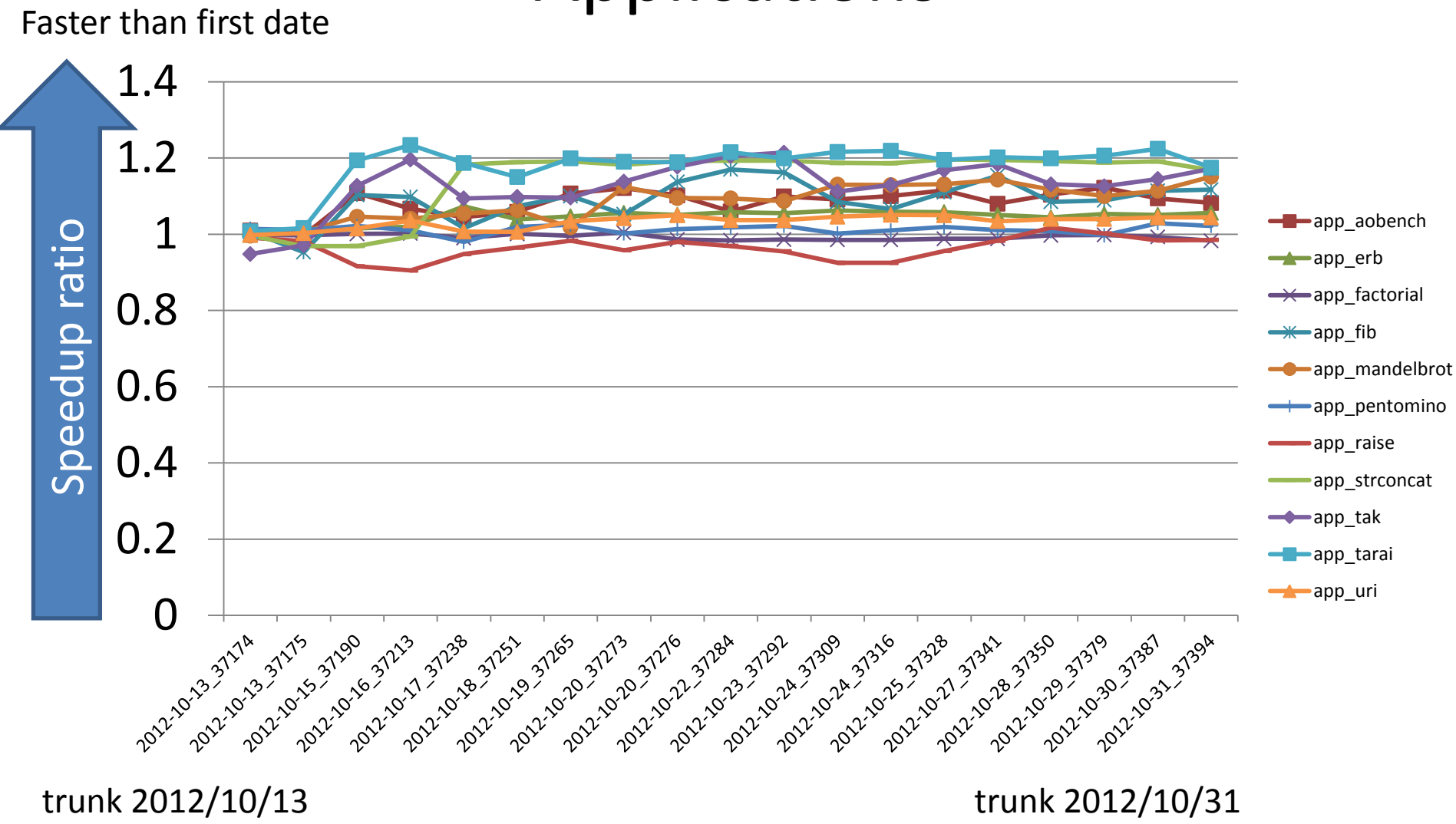
Evaluation result
Micro benchmarks



trunk 2012/10/13

trunk 2012/10/31

Evaluation results
Applications



trunk 2012/10/13

trunk 2012/10/31

Future work

- Restructure “method frame”
  - Reduce required information per frame
- Improve “yield” performance
  - Using something cached

Conclusion

Method dispatch speed-up

- Ruby's method dispatch is nightmare
  - Too complex
- Speedup upto 50% at simple method dispatch with new optimizations
- Need more effort to achieve performance improvements

Other optimizations from 2.0

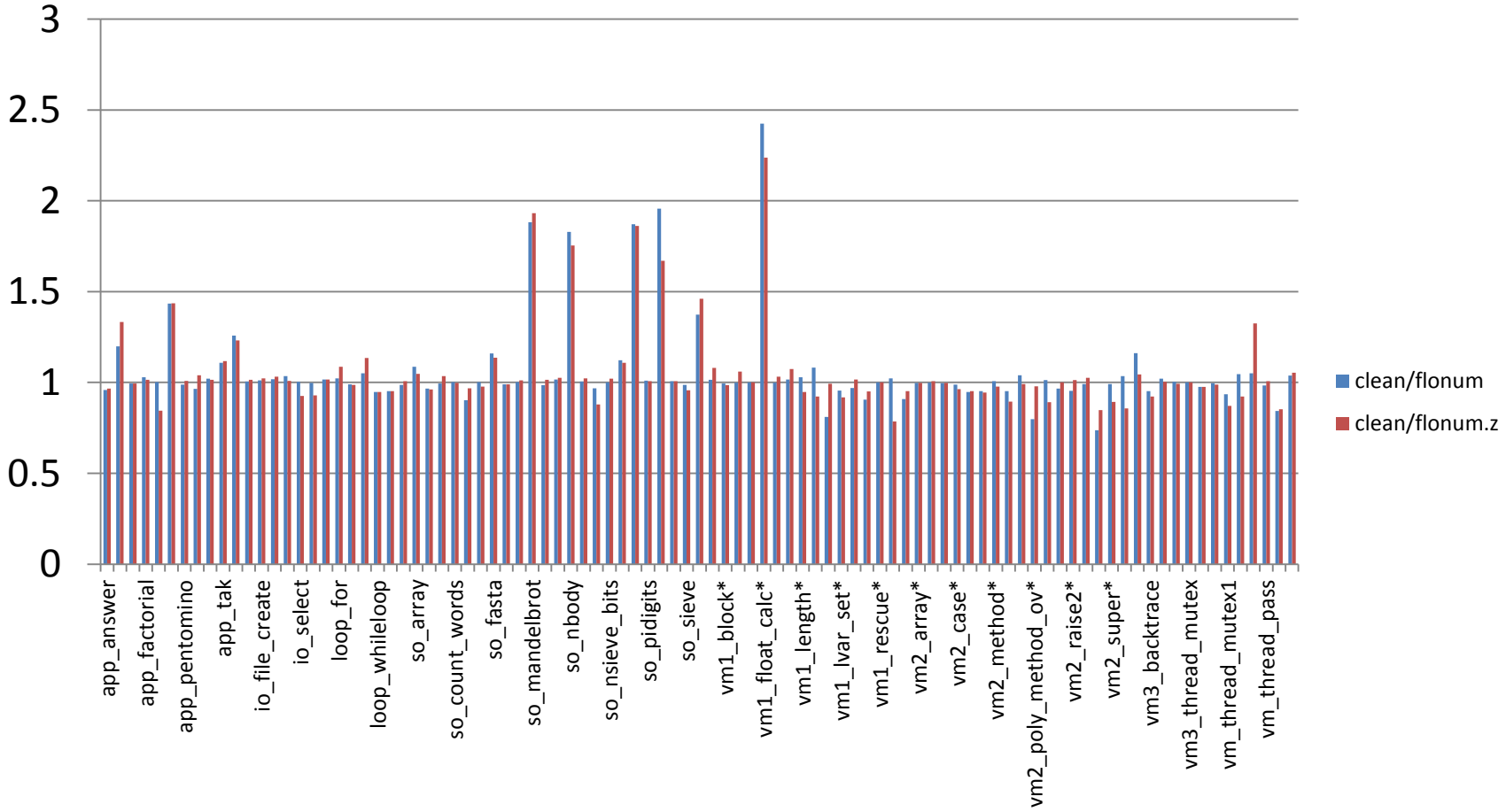
- Introducing Flonum (only on 64bit OSs)
- Lightweight Backtrace capturing
- Re-structure VM stacks/ISeq data

- Bitmap marking garbage collection (by nari3)
- “require” performance (not by me)

Introducing Flonum
(only on 64bit CPU)

- Problem: Float objects are not immediate on Ruby 1.9
  - It causes GC overhead problem
- **To speedup floating calculation, represent Float object as immediate object**
  - Specified range Float objects are represented as immediate object (Flonum) like Fixnum
    - 1.72723e-77 < |f| < 1.15792e+77 (approximately) and +0.0
    - Out of this range and all Floats on 32bit CPU are allocated in heap
  - No more GCs! (in most of case)
  - Flonum and old Float are also Float classes
  - Proposed by [K.Sasada 2008]
  - On 64bit CPU, object representation was changed

Benchmark results

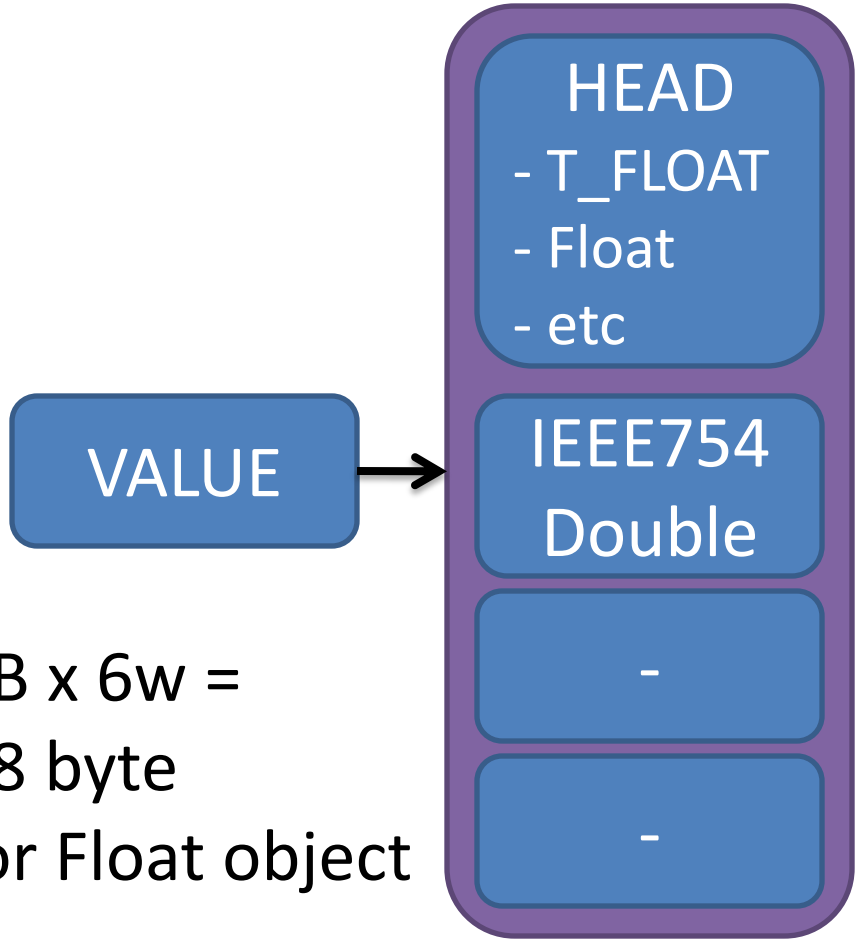


Flonum: Float in Heap (1.9 or before)

All of Float object
are allocated in heap

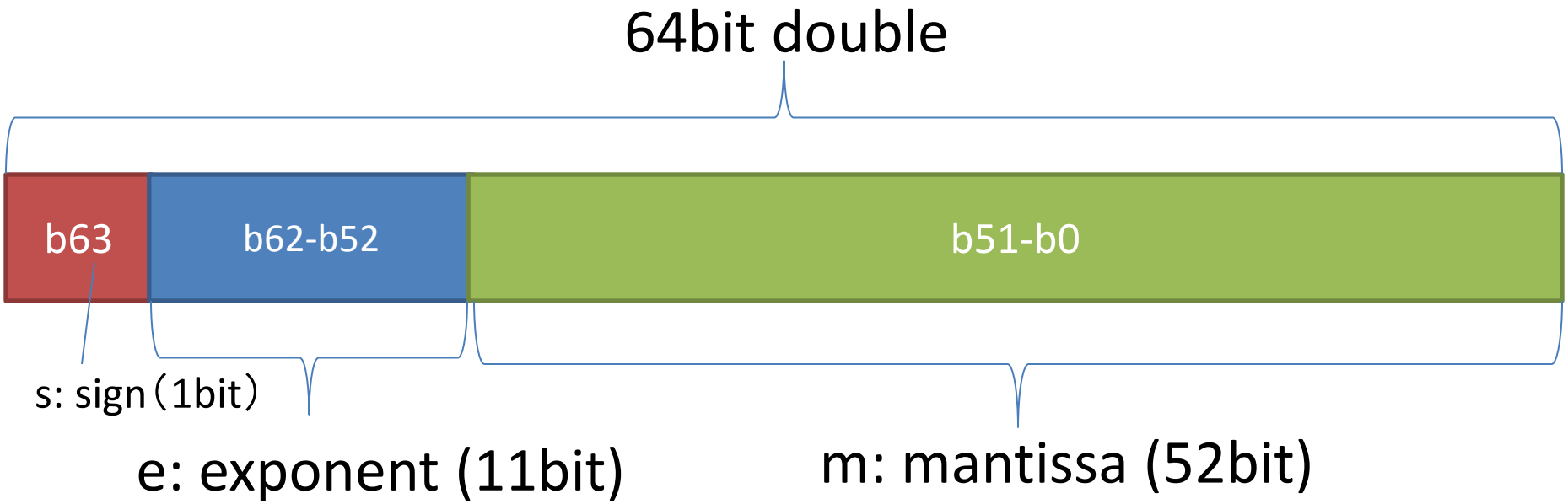
Data structure in heap
contains IEEE754/double

On 64bit CPU



8B x 6w =
48 byte
for Float object

Flonum: Encoding IEEE754 double floating number



$$-1^{s}2^{e-1024}m \quad (m = 1 + \sum_{i=0}^{51} \frac{b_i}{2^{52-i}})$$

Flonum: Range

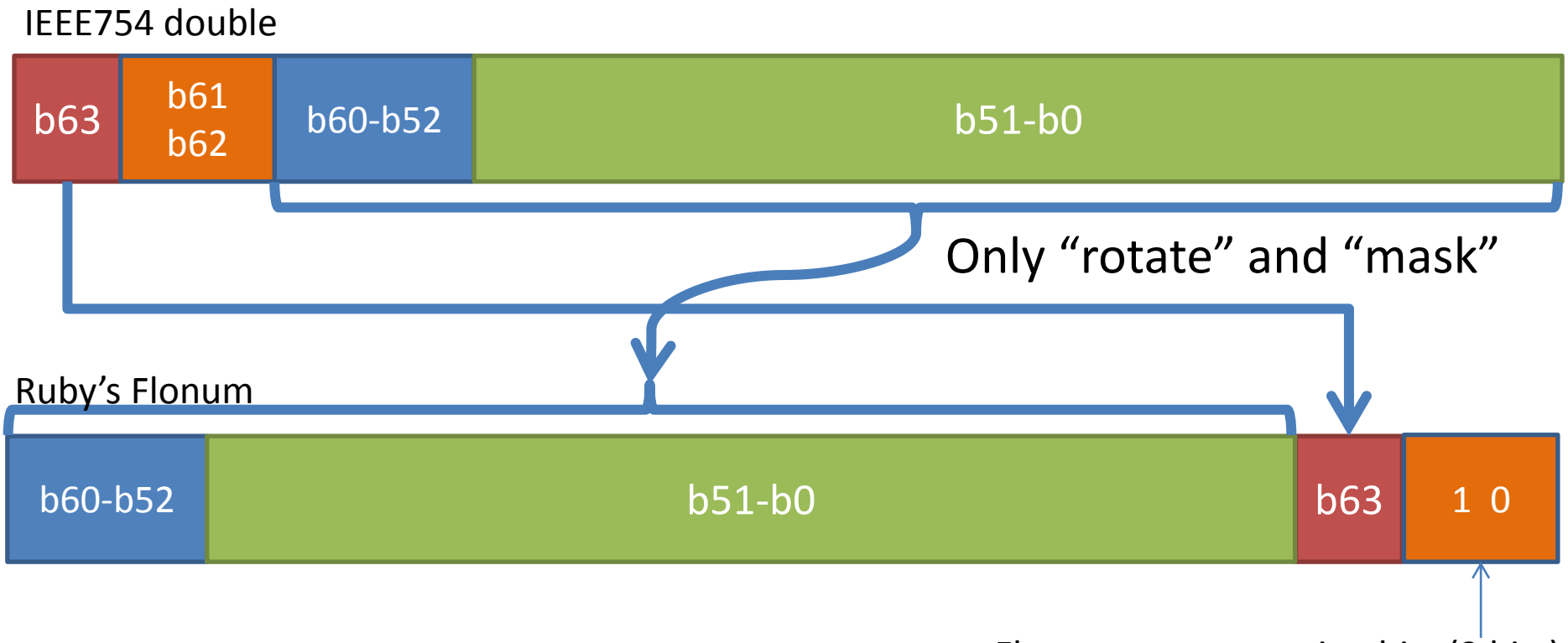
IEEE754 double

Check if e (b52 to b62) is with-in 768 to 1279, then it can be represent in Flonum.

This check can be done with b60-b62.

(+0.0 (0x00) is special case to detect)

Flonum: Encoding



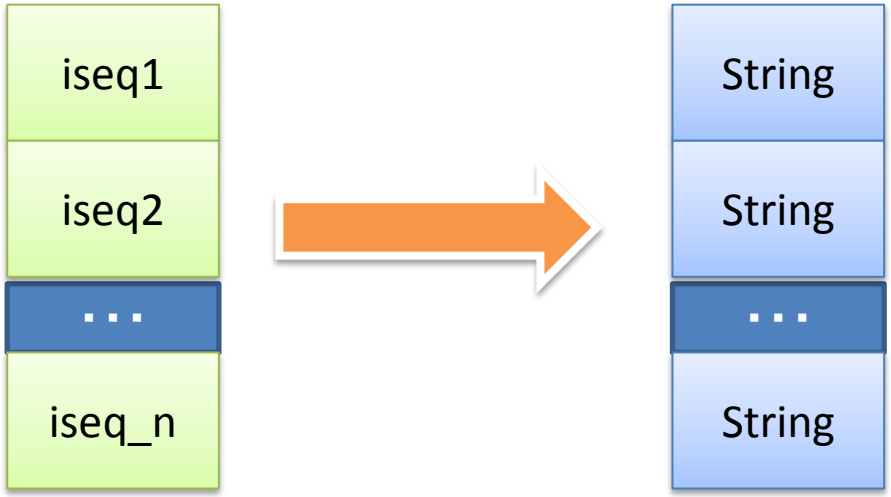
☆ +0.0 is special case (0x02)

Flonum:
Object representation on VALUE

	Non Flonum	Flonum
Fixnum	...xxxx xxx1	...xxxx xxx1
Flonum	N/A	...xxxx xx10
Symbol	...xxxx 0000 1110	...xxxx 0000 1100
Qfalse	...0000 0000	...0000 0000
Qnil	...0000 0100	...0000 1000
Qtrue	...0000 0010	...0001 0100
Qundef	...0000 0110	...0011 0100
Pointer	...xxxx xx00	....xxxx x000

Lightweight Backtrace capturing

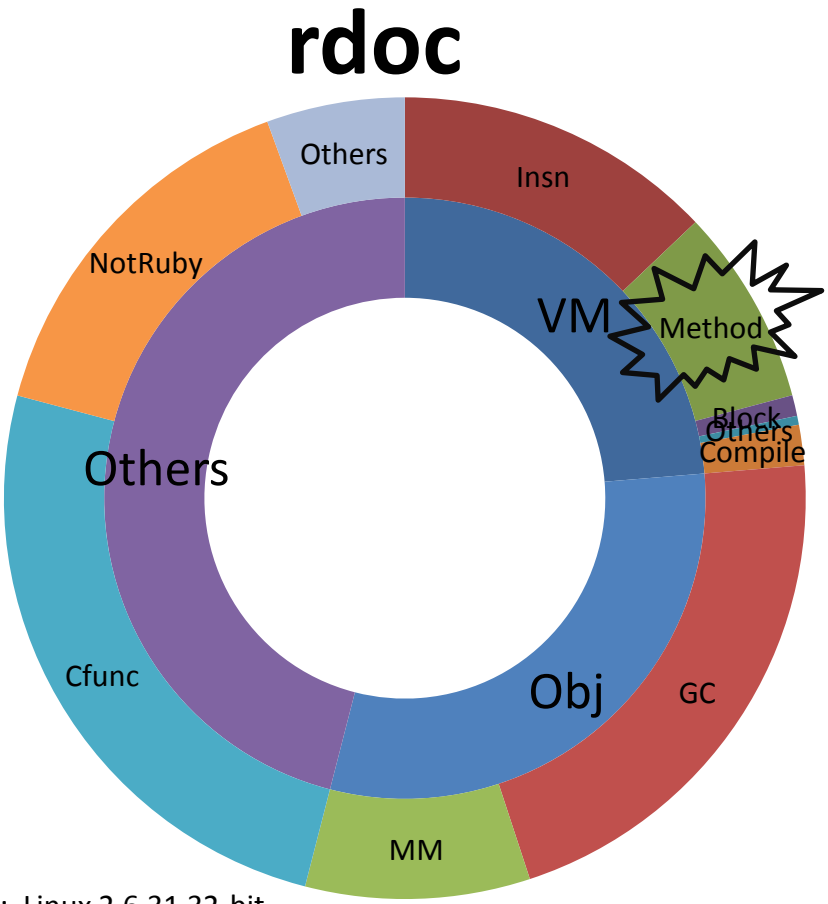
- Backtrace is Array of String objects
  - ["file:lineno method", ...]
- Idea: Capture only ISeqs and translate to String (file and line) only when it is accessed
  - Backtrace information may be ignored

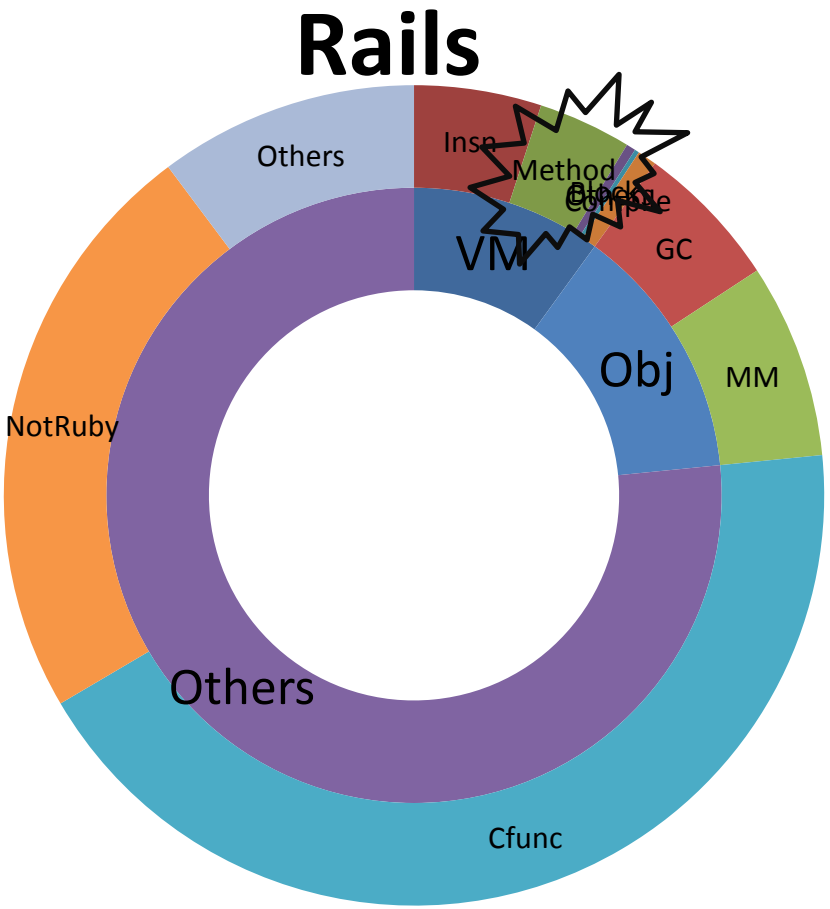


After Ruby 2.0

What we should do?

Performance overhead





OS: Linux 2.6.31 32-bit
CPU: IntelCore2Quad 2.66GHz
Mem: 4GB
C Compiler: GCC 4.4.1, -O3
Profiled by Oprofile

ruby 1.9.3dev (2010-05-26)
Profiled by Mr. Shiba

VM techniques

- On Rails and other applications, VM is not an bottleneck
- On Mathematic, Symbolic computation, VM is matter
  - To speedup then, we need compilation framework
    - 2.0?

Object Allocation and garbage collection

- Lightweight object allocation
  - Sophisticate object creation
  - Create objects in non-GC managed area
- Sophisticate Garbage collection
  - Per-type garbage collection
  - Generational garbage collection
    - Introduce write barriers with dependable techniques

Parallelization

- Multiple processes
- Multiple VMs
- Multiple Threads

No time and space to discuss about them!

Conclusion

Our challenge has just begun!!

俺たちの戦いはまだ始まったばかりだ!

Thank you for your attention

Koichi Sasada

Heroku, Inc.

ko1@heroku.com