Rubinius 1.0
the Ruby VM that could
I am a ...

☐ MD
☐ Lawyer
☐ Physicist
☐ Chemist
☐ Rocket Scientist
In my work, I ... 

- use Ruby
- use Rails
- wish I could use Ruby
- hate being forced to use Ruby and wish I could use Java instead
Wow, I should be contributing to Rubinius!
Rubinius is so hot
Rubinius is better than sliced toast
Rubinius is the cat’s meow
Rubinius is an implementation of the Ruby programming language.
Part II
Why Rubinius?
Why water?
Why Ruby?
Ruby is ...

extremely dynamic
- no decision before its time
- supports changing assumptions

humane
- aliases

useful
- versatile core library
- flat hierarchies

malleable
- DSLs
- extensible programs
... but, Ruby is slow
so these language features must make it too hard to implement
Bullsh*t
Rubinius will demonstrate that Ruby can be a fast language
Part III
Enumerating the elements of a Hash

- MRI C
- MRI Ruby
- RBX
- RBX Jit
- RBX Jit Inline

- 50 items
- 1000 items
Enumerating the elements of a Hash

- MRI C
- MRI Ruby
- RBX
- RBX Jit
- RBX Jit Inline

50 items
1000 items
Getting, setting and deleting the elements of a Hash

![Bar chart showing performance comparison between MRI C, MRI Ruby, RBX, RBX Jit, and RBX Jit Inline for 10 items and 100 items. MRI Ruby performs significantly worse than the other languages.]

- MRI C
- MRI Ruby
- RBX
- RBX Jit
- RBX Jit Inline

Seconds

- 10 items
- 100 items
Getting, setting and deleting the elements of a Hash
Creating 1,000,000 Hash instances

- MRI C
- MRI Ruby
- RBX
- RBX Jit
- RBX Jit Inline

Seconds

<table>
<thead>
<tr>
<th></th>
<th>MRI C</th>
<th>MRI Ruby</th>
<th>RBX</th>
<th>RBX Jit</th>
<th>RBX Jit Inline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seconds</td>
<td>2.5</td>
<td>5</td>
<td>2.5</td>
<td>2.5</td>
<td>2.5</td>
</tr>
</tbody>
</table>
Creating 1,000,000 Hash instances
Your Ruby code on MRI
Your Ruby code on Rubinius
Your Ruby code on Rubinius

Ruby  ➔  Optimized Machine Code  ➔  Ruby
Part IV
A brief history of Rubinius

Evan announces Rubinius
A brief history of Rubinius

Engine Yard hires Evan
A brief history of Rubinius

Engine Yard hires more
A brief history of Rubinius

Rubinius runs Rails #61684750
A brief history of Rubinius

C++ VM runs mspec
A brief history of Rubinius

“stackfull” branch merged
A brief history of Rubinius

New LLVM JIT #a3526242
A brief history of Rubinius

Rubinius runs Rails again
A brief history of Rubinius
Part V
def [](key)
    if entry = find_entry(key)
        entry.value
    else
        default key
    end
end
def concat(other)
    Ruby.primitive :array_concat
    concat Type.coerce_to(other, Array, :to_ary)
end
// Ruby.primitive :array_concat
Array* concat(STATE, Array* other);
Array* Array::concat(STATE, Array* other) {
    size_t osize = other->size();
    ...
}

vm/builtin/array.cpp
Bytecode compiler

source -> IR -> symbolic bytecode -> bytecode
class Flower
    def bloom(*a)
        b a
        end
    end
end

Bytecode compiler
Bytecode compiler

bin/rbx describe f.rb
Bytecode compiler

=============== :Flower ===============
contains 1 CompiledMethods
object_id: 0xa8
total args: 0 required: 0
stack size: 5, local count: 0
lines: #<Rubinius::Tuple: 0, 1, 1, 2, 16>

0000: push_self
0001: add_scope
0002: push_const_fast :Rubinius, nil
0005: push_literal :bloom
0007: push_literal #<Rubinius::CompiledMethod bloom file=f.rb>
0009: push_scope
0010: push_variables
0011: send_method :method_visibility
0013: send_stack :add_defn_method, 4
0016: ret

-----------------------------
Bytecode compiler

============== :bloom ===============
object_id: 0xda
total args: 0 required: 0
(splatted)stack size: 3, local count: 1
lines: #<Rubinius::Tuple: 0, 3, 7>

0000: push_self
0001: push_local 0  # a
0003: allow_private
0004: send_stack :b, 1
0007: ret

--------------------------------------

Bytecode compiler
Bytecode interpreter

stackful vs stackless
Garbage collector

Generational

semi-space copying collector
Immix mark-region collector
C extensions

Object handles

persistent
non-moving
integrated with the GC
C extensions

Extensions

readline
digest
bigdecimal
syck
mongrel
sqlite3
melbourne
JIT compiler

bytecode -> LLVM IR -> machine code
class Flower
  def bloom(*a)
    b a
  end
end

JIT compiler
JIT compiler

bin/rbx -Xjit.enabled -Xjit.show_code=4
JIT compiler

0x2280010 push %esi
0x2280011 sub %0x68, %esp
0x2280014 mov %0x78(%esp), %eax
0x2280018 mov %0x8(%eax), %ecx
0x228001b mov %edi, 0x3c(%esp)
0x2280023 mov %eax, 0x44(%esp)
0x2280027 mov %ecx, 0x48(%esp)
0x228002b mov %esi, 0x4c(%esp)
0x2280033 mov %edi, 0x50(%esp)
0x2280036 mov %esi, 0x54(%esp)
0x228003f lea 0x24(%esp), %ecx
0x2280043 mov %esi, 0x58(%esp)
0x228004b mov %edi, 0x5c(%esp)
0x2280053 mov %edi, 0x60(%esp)
0x2280056 mov %edi, 0x64(%esp)
0x228005d mov %esi, 0x24(%esp)
0x2280063 mov %esi, 0x68(%esp)
0x2280067 mov (%edi), %edx
0x228006d mov %edi, 0x6c(%esp)
0x2280070 mov %edi, 0x70(%esp)
0x2280074 mov %edi, 0x74(%esp)
0x2280077 mov %edi, 0x78(%esp)
0x228007b mov %edi, 0x7c(%esp)
0x2280083 mov %edi, 0x80(%esp)
0x2280087 mov %edi, 0x84(%esp)
0x228008f mov %edi, 0x88(%esp)
0x2280091 mov %edi, 0x8c(%esp)
0x2280099 mov %edi, 0x90(%esp)
0x228009e call 0xfffffffffde9f6c0 ; 0x11f6d0 rbx_check_interrupts
0x22800a3 mov %edi, 0x94(%esp)
0x22800a6 mov %edi, 0x98(%esp)
0x22800ab mov %edi, 0x9c(%esp)
0x22800ba mov %esi, (%esp)
0x22800b2 call 0xfffffffffde9f6c0 ; 0x11f6d0 rbx_check_interrupts
0x22800b7 test %esi, %eax
0x22800ba9 jnz 0x22800e1
0x22800b4 cmp %edi, 0x24(%esp)
0x22800b8 jz 0xca ; 0x22800da
0x22800c4 cmp %edi, 0x24(%esp)
0x22800c8 jz 0xca ; 0x22800da
0x22800ca lea %edi, 0x24(%esp)
0x22800cb mov %esi, %edi
0x22800dd mov %esi, %edi
0x22800e4 call 0xfffffffffde9f6c0 ; 0x11f6d0 rbx_check_interrupts
0x22800e9 xor %edi, %eax
0x22800eb add %edi, 0x24(%esp)
0x22800f0 pop %esi
0x22800f6 ret
class Flower
  def bloom(*a)
    a
    end
  end
end
JIT compiler

0x2280010  sub $0xc, %esp
0x2280013  mov 0x1c(%esp), %eax
0x2280017  mov %eax, 0x4(%esp)
0x228001b  mov 0x10(%esp), %eax
0x228001f  mov %eax, (%esp)
0x2280022  mov $0x0, 0x8(%esp)
0x228002a  call 0xfffffffffde9d640 ; 0x11d650 rbx_construct_splat
0x228002f  add $0xc, %esp
0x2280032  ret
class Flower
  def bloom(*a)
    1
  end
end
JIT compiler

0x2280010  mov $0x3, %eax
0x2280015  ret
JIT compiler

bin/mspec ci -T -Xjit.enabled \  
-T -Xjit.inline.generic \  
-T -Xjit.inline.debug
Part VI
How can I help?

1. Run your Ruby code
How can I help?

2. Write specs for failures
How can I help?

3. File issues on Github
How can I help?

4. Send patches
Rubinius commit policy

If your patch is accepted, you get a commit bit
git clone git://github.com/evanphx/rubinius.git

cd rubinius

rake

less doc/getting_started.txt
#rubinius on irc.freenode.net

http://rubini.us
http://github.com/evanphx/rubinius/issues

ephoenix@engineyard.com
bford@engineyard.com
Thank you