code://Rubinius/technical

/GC, /cpu, /organization, /compiler

Evan Phoenix

weeee!!

Evan Phoenix

Rubinius

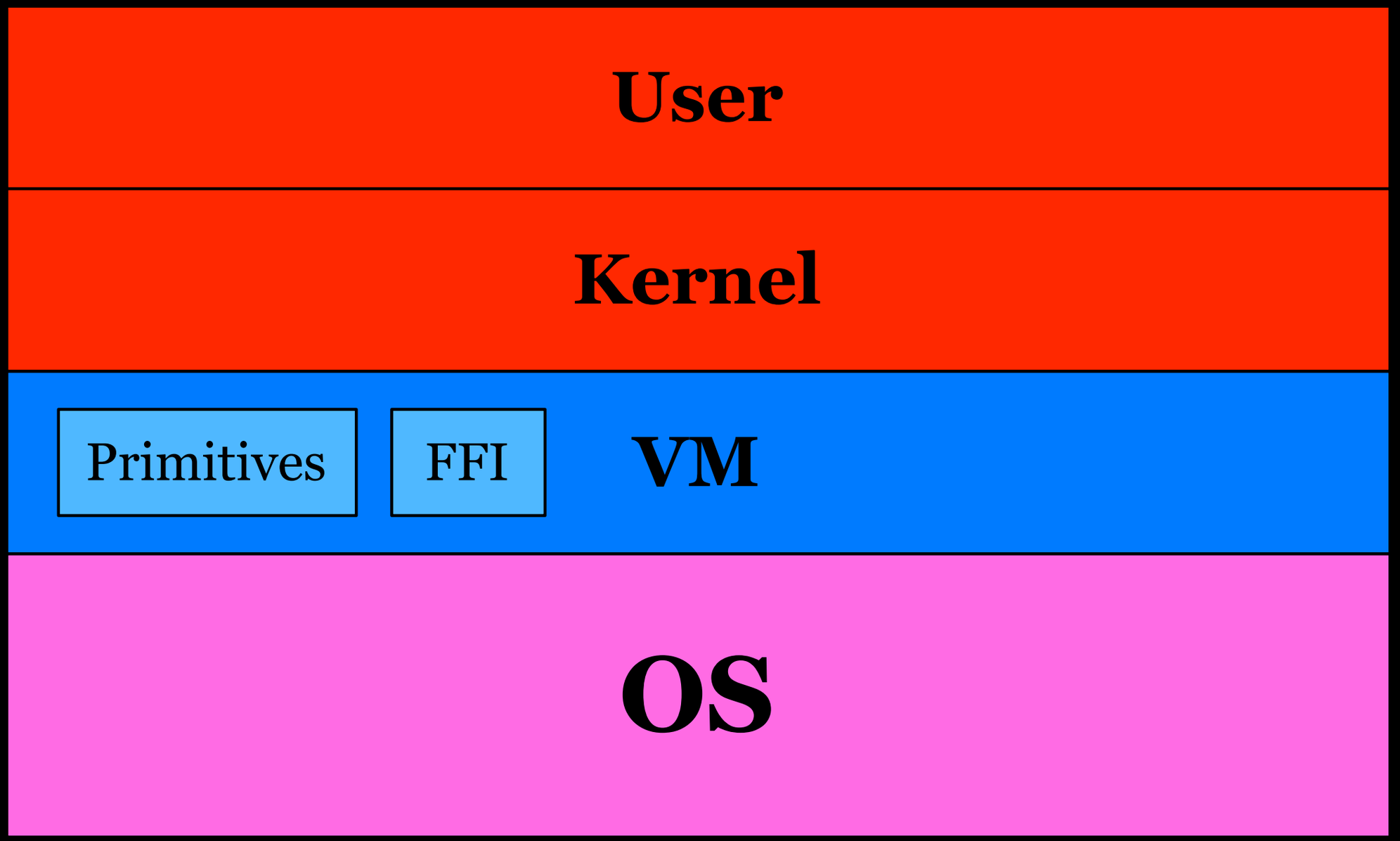
- New, custom VM for running ruby code
- Small VM written in not ruby
- Kernel and everything else in ruby

Evan Phoenix

http://rubini.us

git://rubini.us/code

Evan Phoenix



Evan Phoenix

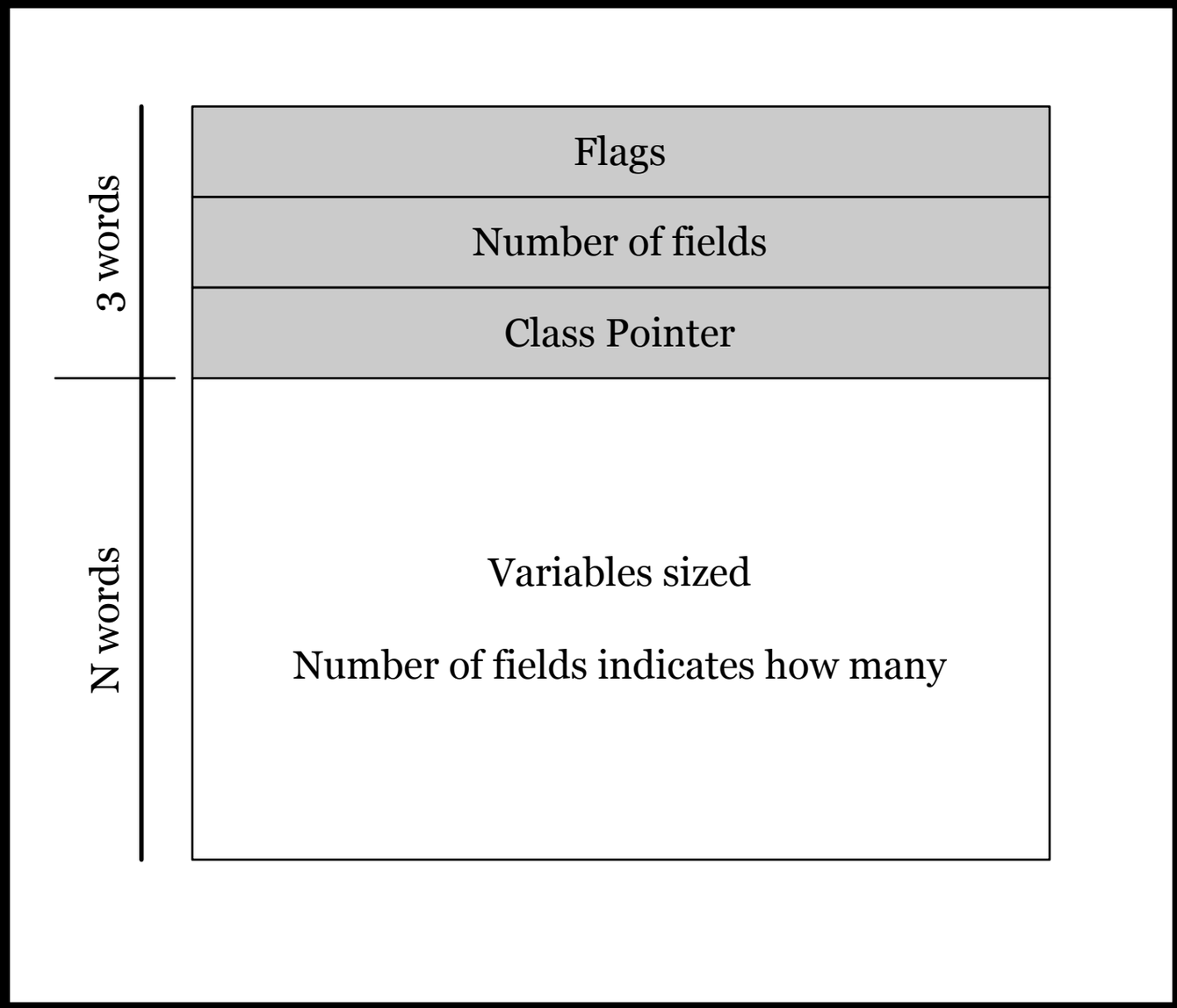
VM Services

- ObjectMemory
  - Generational GC
- Virtual CPU
  - Custom Instruction Set
- Primitive operations

ObjectMemory

- Allocation
  - Variable sized objects
    - Opaque objects
- Accurate collection of all garbage objects
  - No C stack / register walking

Object Layout



Object Flags

- Used by GC
  - Forwarded, Remember, Mark, ForeverYoung
- Allow for opaque objects
  - StoresBytes
- Allow for weak references
  - RefsAreWeak

Opaque Objects

- Allows GC to raw bytes
  - ByteArray class used by String primary is the example
- Allows GC to store C structs
  - Used by VM to implement some objects
  - MethodContext, SendSite, etc.

Generational

- Young object space (YOS) uses Baker/ Cheney copy collector
  - Made simple using accurate collection
- Mature objectp space (MOS) uses simple mark/sweep collector
- Write barrier keeps GC sane

Write Barrier

- Piece of C code run everytime an object reference is stored into another object
- Allows the VM to be sure it knows all objects that point into YOS
- Allows GC to collect YOS independently

Allocation Steps

- YOS has 2 halves
- YOS current half tried first
- if fails, use other half
- if fails, use MOS (never fails)

Accurate Collection

- VM is able to see every object reference in system
- Makes copy collector possible
- Collection is only performed at ‘safe points’
  - Only one defined currently
- At a safe point, there are no hidden object references

No Platform Specific GC code!

/cpu

Evan Phoenix

Virtual CPU

- ‘bytecode’ interpreter
  - ‘registers’ provide ability to implement flow control
  - Uses some techniques to improve performance
    - Direct threading, integer opcodes

Instructions

- 114 instructions
- Instructions created as needed
- Each instruction is 4 bytes (a 32bit integer)
- Allows for up to 2 operands, defined statically per instruction
- Most are flow control related, very few manipulate objects directly
  - Differs from Java in this way

MethodContext

- First class ‘stack frame’ objects
- Contains all information about the current of a method
- Data is copied between a MC and the CPU when the MC is run
- If new MC is created, information in CPU is copied back to original MC

```
def silly
    a = 3
    mc = MethodContext.current
    mc.locals[0] = 18
    p a # => 18
end
```

Evan Phoenix

```
def evil_and_silly
    a = 3
    mc = MethodContext.current.sender
    mc.locals[0] = 18
end

def poor_parent
    a = 3
    evil_and_silly()
    p a # => 18
end
```

Evan Phoenix

Spaghetti Stack

- The ‘call stack’ is a linked list
- Each MethodContext has a field call sender
- Each sender points to the MC to restore when this MC returns
- Toplevel MC has nil sender, causing the VM to exit.

Task Objects

- Some ‘registers’ of the CPU are global, i.e. not stored in each MC
- These are saved and restored from Task objects
- Each Task object represents the complete state of the CPU
- Used as the muscle in the Thread and Continuation classes

Primitives

- Basic operations that the VM provides
- Most are simple chunks of code with fixed number of arguments and one return
- Some reconfigure the CPU in a new way to provide unique functionality
- Hooked up to a method using syntax
- Can succeed or fail

Hooking Up

```
class Fixnum
  def +
    Ruby.primitive :fixnum_add
  end
end
```

- Compiler detects syntax and saves the name of the primitive in the CompiledMethod object

Simple - fixnum_add

```
ARITY(1);
GUARD(FIXNUM_P(self));
OBJECT t1 = stack_pop();
if (FIXNUM_P(t1)) {
  stack_push(fixnum_add(state, self, t1));
  return TRUE;
} else {
  return FALSE;
}
```

Primitive Failure

- Code after Ruby.primitive is run, allowing the method to try again, provide a different implementation, or fail.

```
class Fixnum
  def +(other)
    Ruby.primitive :fixnum_add
    raise “damnit!”
  end
end
```

FFI

- Implemented using a couple of primitives
- Allows Ruby code to bind and directly call C functions
- Automatically converts between Ruby and C types

```ruby
module OutsideRuby
  attach_function ‘strlen’, [:string], :int
end
```

```ruby
str = “hello denmark”
p OutsideRuby.strlen(str) # => 13
```

Evan Phoenix

- Allows for faster development of methods tied directly to native libraries
- getpwnam, socket, etc.

Threads

- Similar to 1.8
- Green threads built on Task objects
- Preemption based on simple timer
- API directly to VM thread scheduler
- Channel objects provide scheduler notifications

Dispatch

- Largest amount of time spent in calling methods
- Any performance benefits have big pay offs

Caching

- Finding the correct method takes the most time
- Caching provides ways to shortcut searching
- Multiple layers of caching

Global Cache

- For each send, the class and method name are hashed
- Hash value is clamped and used as index into large table
- Value is validated and used

Send Site Cache

- Each place where a method is performed is called is called a send site
- Observations about code usage find interesting patterns
  - Most code is NOT polymorphic
  - Each time a method is called, self, arguments, and locals are the same ‘type’

- Initially, send site is empty
  - Causes the global cache to be consulted
  - Information within the send site is updated
- Next time send site is used
  - Information contained within is validated and used directly

Locality

- Locality contains very rich information
- Any exploitation of locality can increase performance
- Java's Hotspot uses locality as the primary exploitation mechanism

Other VM operations

- Ability to directly save, load, and execute a .rbc file
- Simple multi-VM spawn and communication
- Basic abilities to manipulate builtin classes such as Hash, Array, etc.

Division of Ruby Code

Evan Phoenix

Kernel-land

- All ruby code located in kernel/
- VM loads code directly without require
- Phase order: bootstrap, platform, kernel
- Load order of .rb files determined by special dependencies comments

User-land

- All code loaded by the kernel
- No strict division from kernel code like an OS
- Label used primarily to group code when doing Rubinius development

/compiler

Evan Phoenix

VM / Compiler boundary

- VM provides ability to execute CompiledMethod objects
- CompiledMethod objects are normal, first class objects
  - Can easily be built up from scratch, one bit at a time

Sexp

- 2 VM primitives
- Provide ability to parse and emit code as data
  - Thanks to ParseTree
- In the future, the parser will be all in ruby
  - ruby_parser (Thanks Ryan Davis)

Ruby Code:

```
"aoeu".count
```

Sexp:

```
[:call, [:str, "aoeu"], :count]
```

- Sexps are the input to the compiler
  - Allows for custom Sexp composition
- Compiler transforms Sexp into internal tree
- Compiler walks tree, using visitor pattern to generate bytecode