JRuby: You've got Java in my Ruby

Thomas E. Enebo
<tom.enebo@gmail.com>
JRuby Architect, Engine Yard

Differing Goals

- Two Audiences?
  - Ruby - "Why JVM is good for Ruby impl"
  - Java - "Appreciate how the JVM complements another language and learn a little Ruby along the way"

Who am I?

- JRuby co-lead
- Java guy (since the beginning?)
- Ruby guy 7-8 years
- Employed by Engine Yard to work on JRuby!
  - “It’s my day job”

What is JRuby?

- Ruby on the JVM (Java 5+)
  - Open-Source: GPL/CPL/LGPL
  - 1.8.7 compatible
  - Has 1.9-mode (--1.9)

JRuby Boasts...

- Great Compatibility
- Fast!
- Native Threads
- All the JVM Buzzwords
  - More on this a little later

Note on Compatibility

- ~37,000 passing rubyspecs
- ~22,000 passing assertions
- CI Runs
  - Java versions, platforms, common libraries

Incompatibilities

- Missing some POSIX behavior (e.g. no fork())
- No continuations (callcc)
- Slower Startup
- Cannot run native C extensions <-- Biggest :(
  - Java Native Extensions (ar-jdbc, yaml, ...)
  - Foreign Function Interface (FFI)

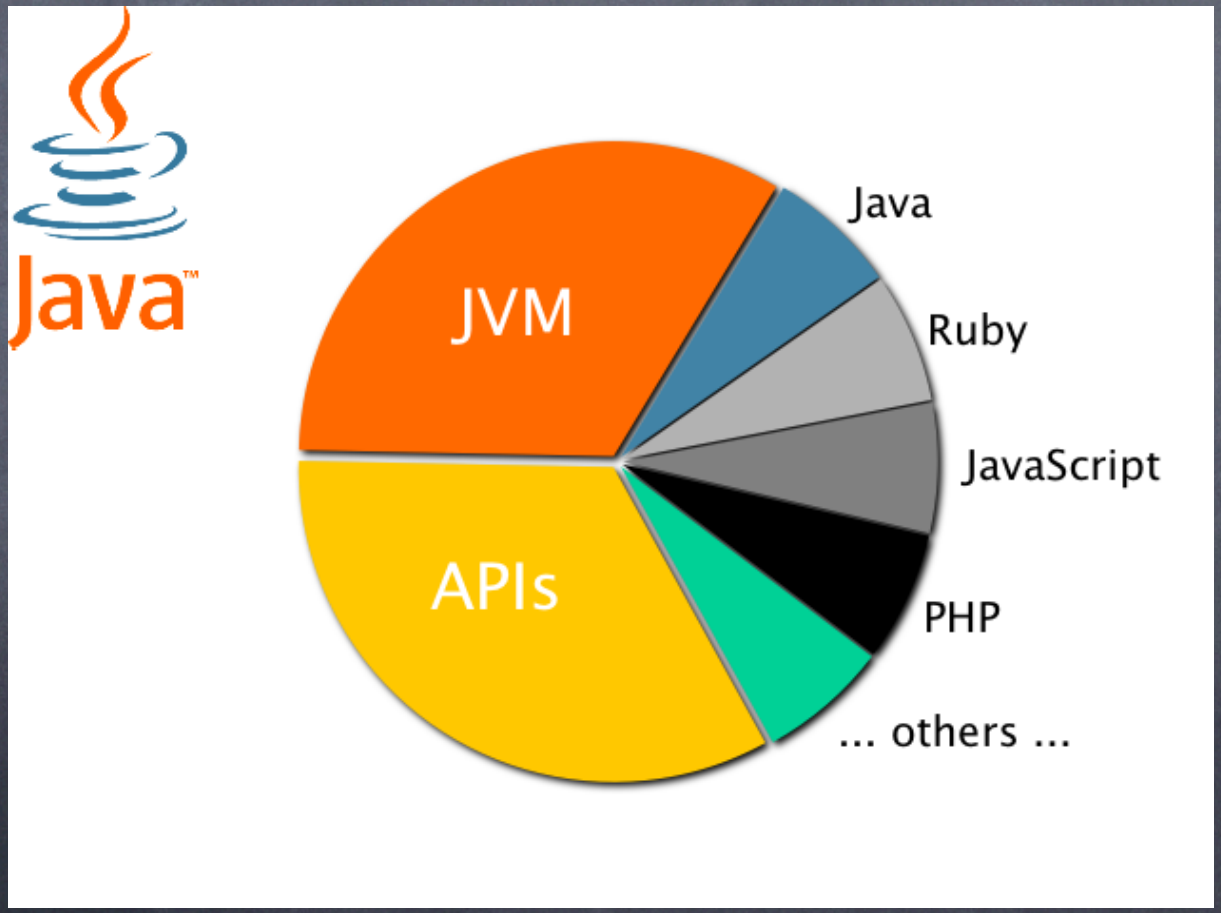
Foreign Function Interface (FFI)

```ruby
require 'ffi'

module POSIX
  extend FFI::Library
  # not usually necessary since libc is always linked
  ffi_lib 'c'

  attach_function :getuid, :getuid, [], :uint
  attach_function :getpid, :getpid, [], :uint
end
puts "Process #{POSIX.getpid}, user #{POSIX.getuid}"
```

JRuby Status Update

- JRuby 1.4.0 released (November 2, 2009)
  - New Embedding framework: RedBridge
  - Improved windows support + installer
  - New bug-for-bug YAML parser: Yecht
  - 400+ issues resolved since 1.3.1
- JRuby 1.5 coming around new years

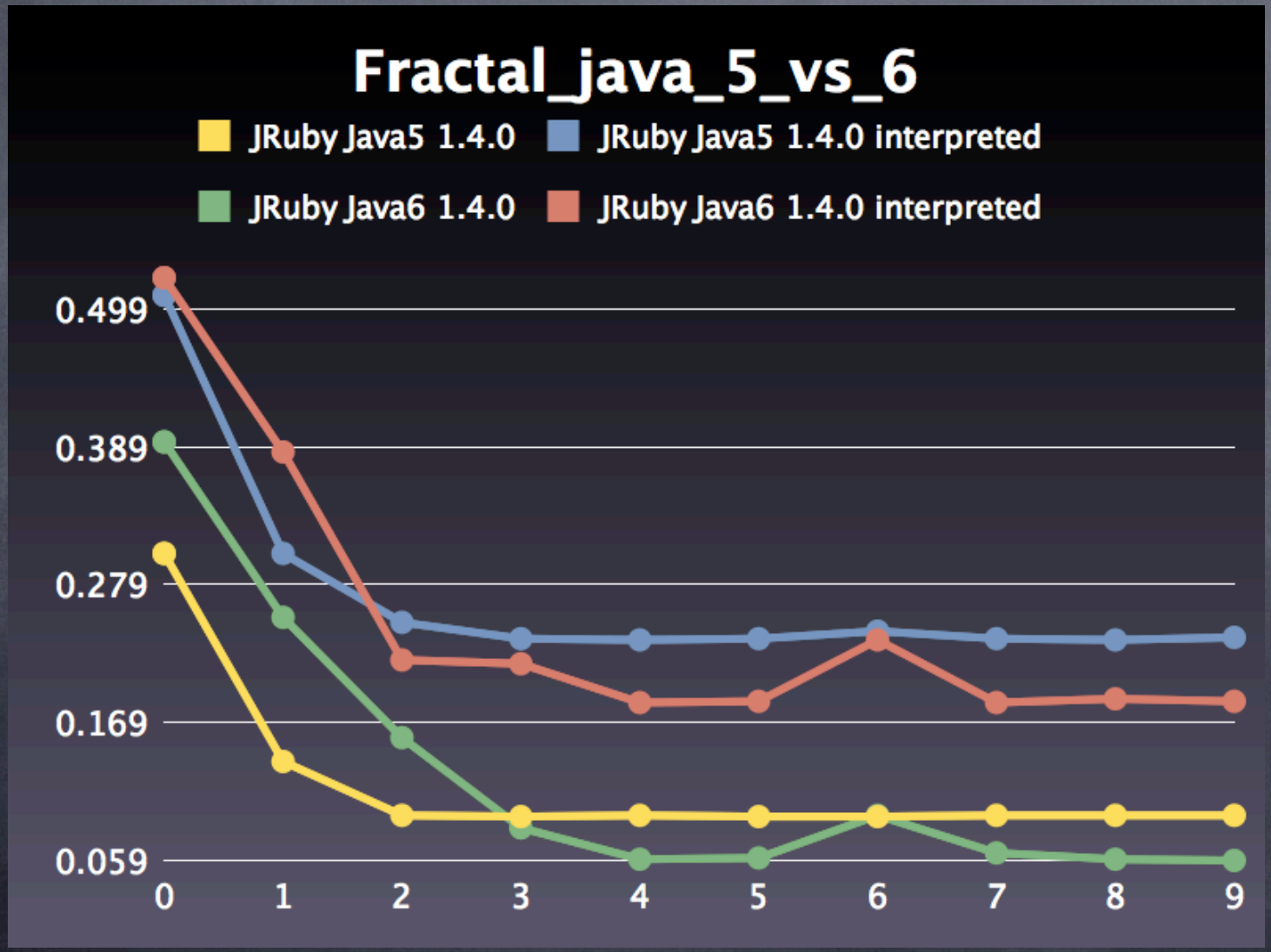
JVM Appreciation



JVM is Mature...

- "silver-back" implementations
  - Decades of debugging and optimizations
  - Capable of incredibly long uptimes
  - Keeps improving over time...

JVM keeps improving...



JVM is Pervasive...

- Every OS you know runs JVM including a few you don't
- Most machines already have JVM installed

JVM Hotspot

- Dynamic profiling your application to optimize the chunks of code which matter
- Can runtime profiling be smarter than a static compiler?
- Are you smarter than your runtime?

Hotspot Session

- Disclaimer: All optimizations shown can happen, but this is merely representative

Hotspot Session: Initial Code

```
Vector v = new Vector(3);  // Thread-safe
list
....
reset(v); // Called many times
....
void reset(Vector v) {
   for (int i = 0; i < v.size(); i++) {
      v.set(i) = 0;
   }
}
```

Hotspot Session: Inlining

```
void reset(Vector v) {
    fast_guard(v) {
        for (int i = 0; i < lock { arr.length }; i++) {
            lock { arr[i] = 0; }
        }
    }
}
```

Hotspot Session: Simple Optz (loop unroll)

```
void reset(Vector v) {
    fast_guard(v) {
        lock { arr[0] = 0; }
        lock { arr[1] = 0; }
        lock { arr[2] = 0; }
     }
}
```

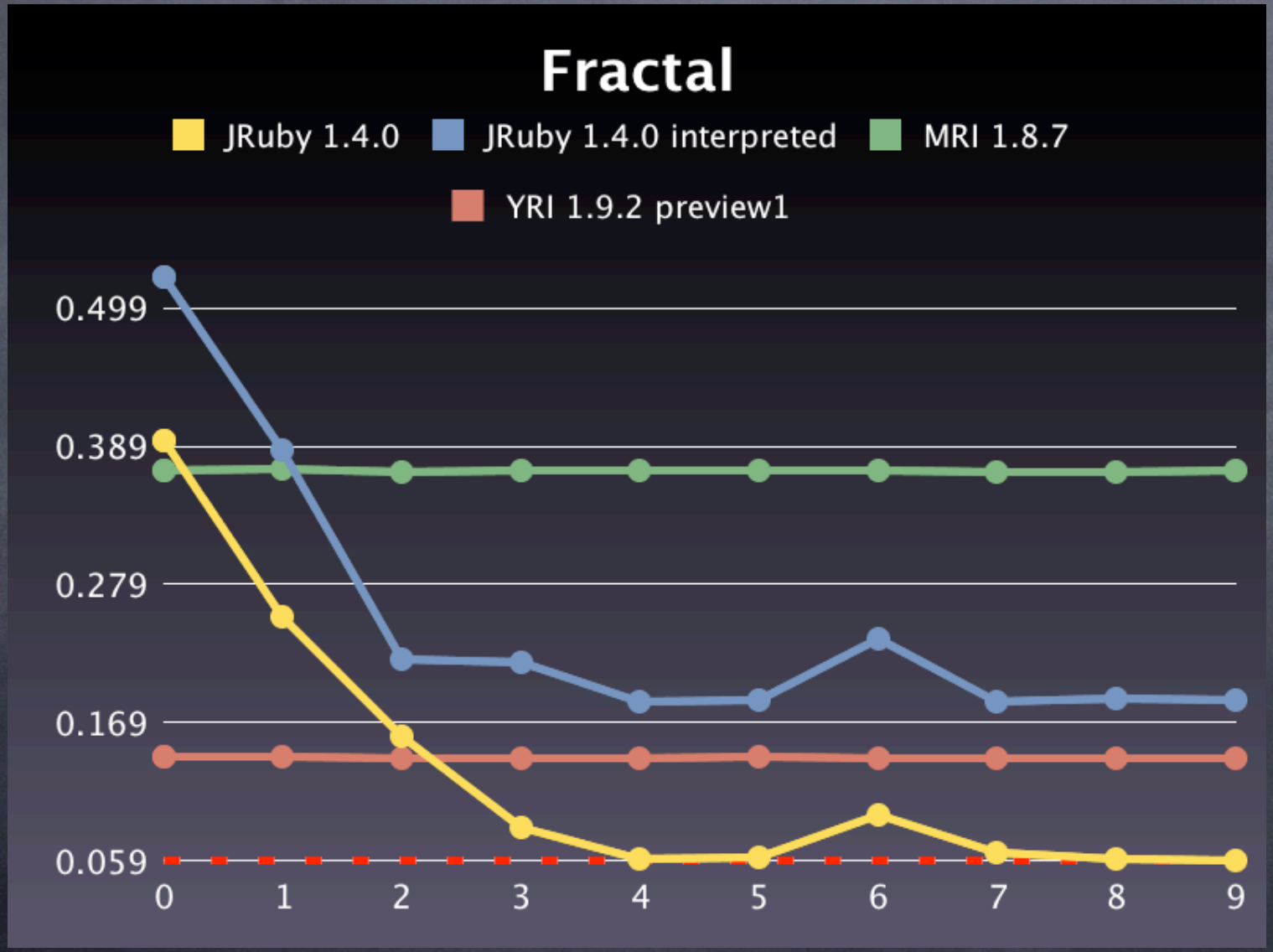
Hotspot Session: Lock Coarsening

```
void reset(Vector v) {
    fast_guard(v) {
        lock {
            arr[0] = 0;
            arr[1] = 0;
            arr[2] = 0;
        }
    }
}
```

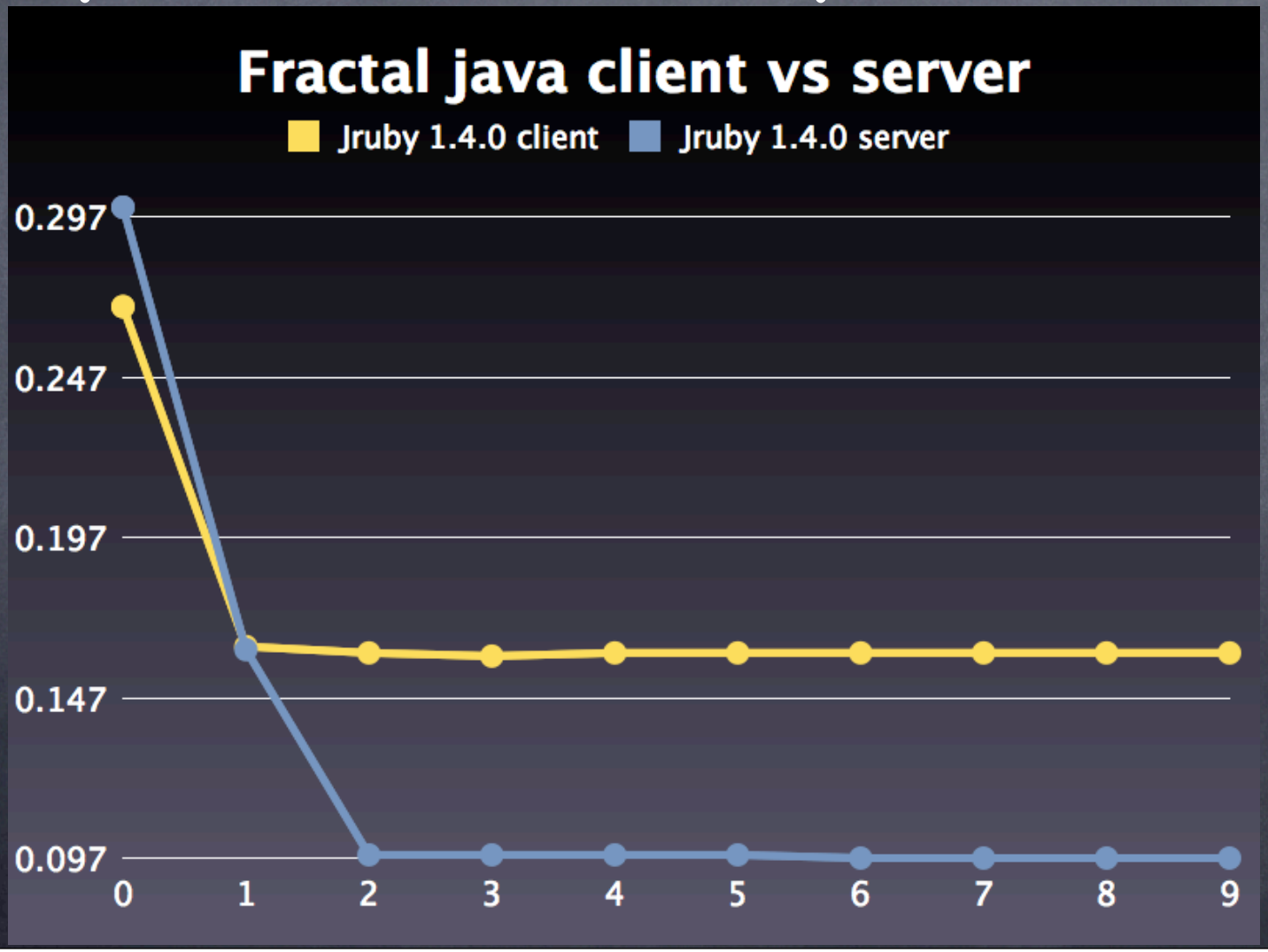
Hotspot Session: Array Copy Stubs

```
void reset(Vector v) {
   fast_guard(v) {
       lock {
           arrayCopyStub(v, [0,0,0])
       }
    }
}
```

Hotspot: Fractal



JVM: Multiple performance profiles



JVM and Garbage!

- Many Garbage Collectors to fit your workload
  - Army of engineers working on them
  - Incremental, Compacting, Generational, Concurrent, Parallel
  - Tons of tunables

JVM GCs: Incremental

- Faster partial GC
  - Smaller discrete phases to reduce GC pauses
  - Sometimes concurrent phases for no pause
- C Ruby is stop-the-world

JVM GCs: Compacting

- No fragmentation
  - Runtime does not gobble all your memory over time
  - No Fragmentation == Long runtimes
- C Ruby is not compacting

JVM GCs: Generational

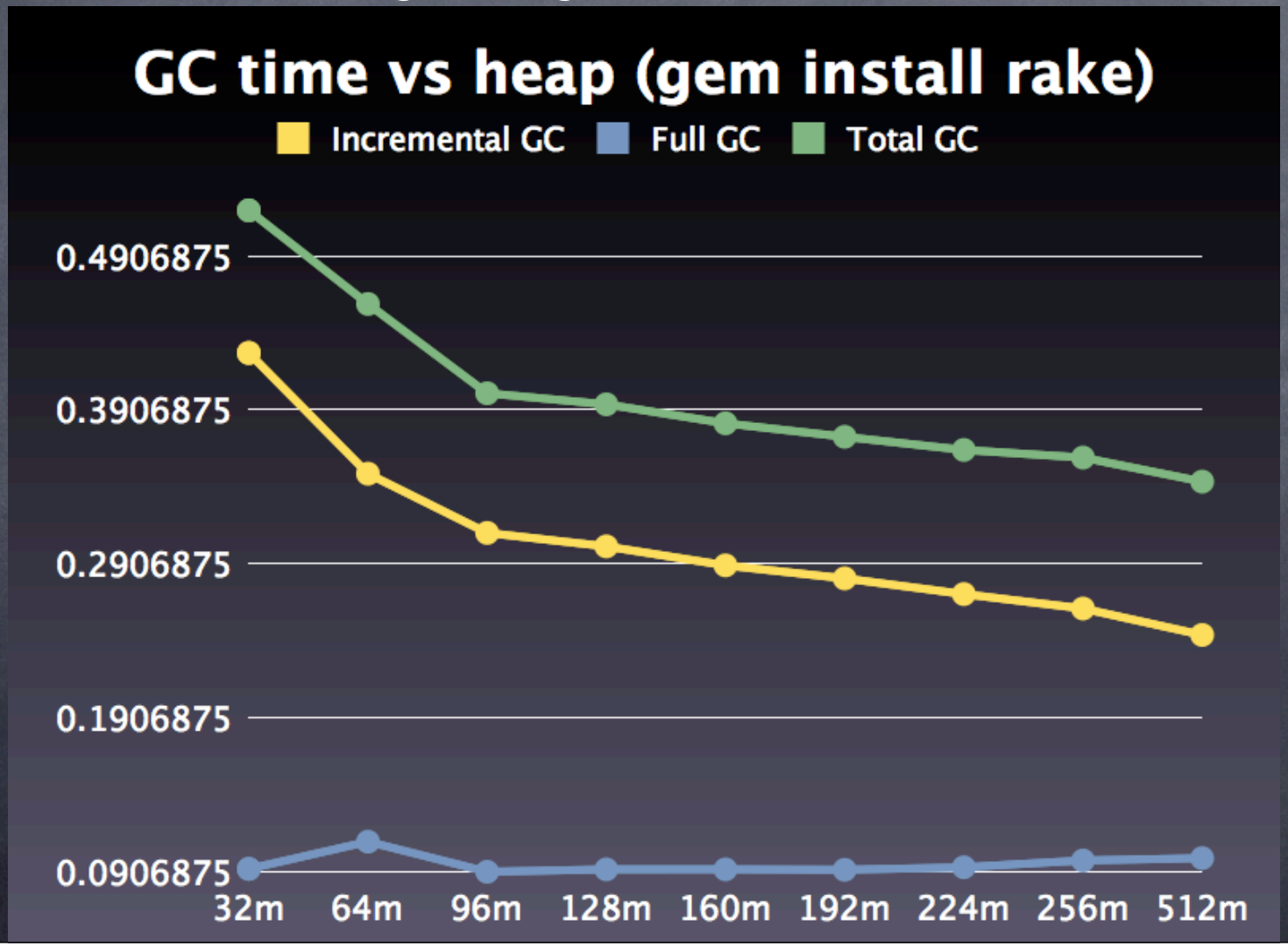
- Short-lived objects collect EXTREMELY fast via incremental collections
- Long-lived object get promoted to different object pool(s)
- Ruby creates tons of short-lived garbage
- C Ruby is not generational

Garbage Collection: Parallel, Concurrent

- Splitting GC across multiple cores
- GC'ing while execution is still happening
  - Dark magic
- Becoming more and more relevant in Multi-core world

JVM GCs: Tunables!

- -J-verbose:gc to give live GC information...



JVM Tools

- Profilers
- Debuggers
- Verbose runtime information from VM
- jconsole + JMX

JProfile Demo

Java IRB Demo

- Midi and Swing in cut time!

Java Library Demo

- JMonkeyEngine + JMEPhysics
  - 3D-accelerated Scene-graph library
  - Is Ruby fast enough?
  - MADNESS!

Cleaning Up Java APIs

- Ruby Language
  - Has less ceremony
  - Has features which Java doesn't
    - Blocks
    - DSLs (aka Syntactic Gymnastics)

Less Ceremony == Easier to consume

- No type declarations
- No checked exceptions
- Much richer core libraries
  - Common tasks simplified

Blocks Remove Boilerplate

```
def read(filename)
  open_file = Reader.new(JFile.new(filename))
  yield open_file
ensure
  open_file.close
end
```

```
read("my_data_file") do |fd|
   fd.read(30)
   # ... more stuff ...
end
```

Blocks & Reusability

```
forward = ForwardAndBackwardAction.new(node,
  ForwardAndBackwardAction::FORWARD)
add_action forward, "forward", true
backward = ForwardAndBackwardAction.new(node,
  ForwardAndBackwardAction::BACKWARD)
add_action backward, "backward", true
```

Blocks & Reusability

```
class ForwardAndBackwardAction < KeyInputAction
  FORWARD = 0
  BACKWARD = 1

  def initialize(node, direction)
    super()
    @node, @direction = node, direction
  end

  def performAction(evt)
    if (@direction == FORWARD)
      @node.accelerate evt.time
    elsif (@direction == BACKWARD)
      @node.brake evt.time
    end
  end
end
```

Blocks & Reusability

```
forward = KeyInputAction.impl { |event| node.accelerate event.time }
add_action forward, "forward", true
backward = KeyInputAction.impl { |event| node.brake event.time }
add_action backward, "backward", true
```

```
@drift = KeyInputAction.impl { |event| node.drift(event.time) }
```

Blocks & Reusability

```
class KeyInputAction
  def self.impl(&block)
    ConcreteKeyInputAction.new(&block)
  end
end

class ConcreteKeyInputAction < KeyInputAction
  def initialize(&block)
    super()
    @block = block
  end

  def performAction(event)
    @block.call event
  end
end
```

DSLs (Syntactic Gym.)

- Use Ruby syntax features to dress up Java APIs

```
/* Missing try/catches.... */
DynamicPhysicsNode iceQube = getPhysicsSpace().createDynamicNode();
iceQube.attachChild(new Box("Icecube", Vector3f.new, CUBE_SIZE, CUBE_SIZE, CUBE_SIZE));
iceQube.generatePhysicsGeometry();
iceQube.setMaterial(Material.ICE);
TextureState textureState = DisplaySystem.getDisplaySystem().getRenderer().createTextureState
();
URL url = System.getResource("data/images/Monkey.jpg");
Texture texture = TextureManager.loadTexture(url, Texture::MinificationFilter:Trilinear,
Texture::MagnificationFilter::Bilinear, true);
texture.setWrap(Texture::WrapMode::Repeat);
textureState.setTexture(texture);
setRenderState(textureState);
iceQube.computeMass();
iceQube.getLocalTranslation().set(START_X, START_Y, START_Z);
```

```
@icecube = physics_space.create_dynamic do
      geometry Cube("Icecube", CUBE_SIZE)
      made_of Material::ICE
      texture "data/images/Monkey.jpg"
      at *START
 end
```

Conclusions

- JVM is a great base for languages
- Java libraries are easy to Rubify

Thanks

- jruby: http://www.jruby.org/
- email: tom.enebo@gmail.com
- twitter: tom_enebo
- blog: http://blog.enebo.com/
- jrme: http://www.kenai.com/projects/jrme