



JRuby: Not Just Another Ruby Impl

Charles Nutter
Thomas Enebo

Introductions: Who am I?

- Charles Oliver Nutter – charles.nutter@sun.com
- Thomas Enebo - thomas.enebo@sun.com
- Engineers at Sun, JRuby core developer
- 10+ years Java experience each
 - > Also C/C++, Win32, C#, .NET, JavaME/CLDC, Perl, LiteStep, JINI, JavaEE
- Working to make Ruby a first-class JVM language
- Working to make JVM a better home for dynlangs

Agenda

- What Is JRuby
- Ruby Design Issues
- The JRuby Way
- JRBuilder/Cheri
- NetBeans Ruby Support
- Compatibility Metrics
- JRuby Compiler and Performance Metrics
- Conclusion and Q/A

What Is JRuby

- Started in 2002
- Java implementation of Ruby language
- Open source, many active contributors
- Aiming for compatibility with current Ruby version
- Easy integration with Java libraries, infrastructure
 - > Call to Ruby from Java via JSR223, BSF
 - > Use Java classes from Ruby (e.g. Script Java)
- Growing set of external projects based on JRuby
 - > JRuby-extras (ActiveRecord-JDBC, rails-integration, ...)

Why Ruby on Java?

- Ruby only supports green threads
- Unicode story is rather weak
- Performance often considered “slow”
- Too great a change for many organizations
- Still relatively new, perceived as “untried”
- Existing investments in Java infrastructure
- Vast, proven collection of Java libraries
- Complete platform independence

Ruby Design Issues

- Ruby 1.8: Green threading
 - > Can't scale across processors/cores
 - > C libraries won't or can't yield to Ruby threads
 - > One-size-fits-all scheduler doesn't fit all
- Ruby 1.9: Native, non-parallel threading
 - > Core classes, extensions not ready for parallel exec
 - > Lots of work to ensure thread-safe internals
 - > Move to native threads, but not running in parallel
 - **May change before 2.0

Ruby Design Issues

- Ruby 1.8: Partial Unicode support
 - > Internet-connected apps **must** have solid Unicode
 - > Ruby provides partial support, but not consistent
 - > App developers must roll their own: Rails MultiByte
- Ruby 1.9: Full Unicode (M17N) but drastic change
 - > String interface changes to per-char, not per-byte
 - Breaks existing consumers of those methods
 - Duplicate methods to allow per-byte access
 - > Each string can have its own encoding
 - Mixed encoding app behavior unclear
 - Much more complicated than “just support Unicode”

Ruby Design Issues

- Ruby 1.8: Slower than most languages
 - > 1.8 is usually called “fast enough”
 - > ...but routinely finishes last on benchmarks
 - > ...and no plans to improve in 1.8
- Ruby 1.9: Improvement, but still more to do
 - > 3-4x faster on some targeted tests
 - > ...but no change on others
 - > AOT compilation
 - > More to do: no JIT; GC, threading still slow;
 - > Not likely to be “done” soon (1.9.1 in December)

Ruby Design Issues

- Ruby 1.8: Memory management
 - > Simple design
 - > Good for many apps, but (probably?) doesn't scale
 - > Stop-the-world GC, no parallel or generational collection
- Ruby 1.9: No change
 - > Improved performance means more garbage
 - > Multiple native threads, potentially more contention
 - > GC problems could multiply, cutting into profits

Ruby Design Issues

- Ruby 1.8: C language extensions
 - > C is difficult to write well
 - > Badly-behaved extensions can crash Ruby runtime
 - > Threading, GC issues
 - > Portable, but often must be recompiled
 - > No security restrictions
- Ruby 1.9: No change
 - > Can't afford to break extensions
 - > Ruby C API exposure limits changes to Ruby core
 - > Common performance advice: “Write it in C”

Other Ruby Issues

- Politics
 - > You want me to switch to what?
 - > ...and it needs new servers/software/training?
 - > Potentially better accepted by 1.9
- Legacy
 - > Lots of Java apps and code in the world
 - Extensive library of Java frameworks
 - Many experienced users, developers, administrators
 - > Large existing investment in Java infrastructure
 - Server software, developer and admin training
 - Existing services, frameworks

The JRuby Way

- Native threading
 - > Scale across all processors/cores in system
 - > Concurrent execution, even in extensions
 - > Allow system to schedule threads
 - > Ensure reasonable safety in core classes

The JRuby Way

- World-class, native Unicode support
 - > Provide Ruby's byte[]-based String
 - ...but also provide native Rails multibyte “Chars” class
 - ...and you can also use Java's UTF-16 String
 - > Java has complete, reliable Unicode support
 - ...and all libraries are Unicode-ready
 - ...and all IO channels support many encodings

The JRuby Way

- Scalable Performance
 - > Make interpreter as fast as possible
 - No reason we can't interpret as fast as Ruby 1.8
 - > Support Ruby 1.9/2.0 bytecode engine
 - Same resulting performance gains in JRuby as in Ruby 1.9
 - Future-proof
 - > Compile to Java bytecode
 - AOT and JIT compilation
 - Let HotSpot take over
 - JIT compilation
 - Code inlining
 - Dynamic optimization

The JRuby Way

- Let Java manage memory
 - > Best memory management and GC in the world
 - > Wide variety of GC options
 - Concurrent
 - Generational
 - Real-time
 - > Scales up to enormous applications and loads
 - > Battle-tested: millions of deployments worldwide

The JRuby Way

- Java-based extensions
 - > Easier to write than C
 - > Nearly impossible to crash runtime or VM
 - > Truly portable: write once, run anywhere
 - > No GC, threading, or security issues: it's all Java
 - > Clean separation between core and extension API
 - > Easier to expose Java libraries than C libraries
 - > “Everybody” knows Java

The JRuby Way

- Politics don't get in the way
 - > JRuby is “just another Java library”
 - > Easier to change web framework than app architecture
 - > Minimal impact to dev, admin processes
 - > Ten years of mainstream Java
- Legacy integrates just fine
 - > Call existing services, libraries directly from Ruby
 - > Implement services in Ruby
 - > Deploy to existing servers
 - > Same proven scalability, reliability...just a new language

What Can It Do?

- Most “pure” Ruby code runs just fine now
- Rake runs well, though no official testing effort
- RubyGems, ditto, but it's hit more often
- Projects using JRuby + RSpec now
- Rails looking better and better
 - > Some small-scale production apps in the wild
 - > Rails unit tests above 98% passing in JRuby
 - > More and more of Rails in our regression suites
- Other combinations of JRuby + X popping up daily

Yeah, But What Else?

- Ruby + Java = Awesome
 - > Java provides libraries and VM Ruby needs
 - Extensive, flexible libraries
 - Often too complicated to use...why?
 - Perhaps Java is good for frameworks, but not for calling them?
 - Perhaps there's a better option for “gluing” libraries together?
 - > Ruby provides agile, flexible, fun language Java needs
 - Unix metaphor: libraries use C, apps often use dynlangs
 - Applied to Java: JVM is our kernel, Java is our C
 - Ruby and its ilk finally complete the platform
 - > More powerful than either alone

JRBuilder/Cheri

- JRBuilder/Cheri project by Bill Dortch
 - > A Ruby “SwingBuilder”
 - > Groovy-like builder syntax for Swing
 - But written entirely in Ruby
 - ...and only a couple kloc
 - > Also inspired by F3
 - Declarative syntax for Swing/Java2D UIs
 - > Data binding support similar to JSR 295 “Beans Binding”
 - > Additional Java integration enhancements
 - Java array and primitive manipulation
 - Some merged into JRuby proper, more to come



Demo: JRBuilder/Cheri - A Ruby SwingBuilder

NetBeans Ruby Support

- Tor Norbye started work in September
- Uses JRuby's parser
- Available in NetBeans 6 Milestone 7
 - > Ruby support in “Update Center”
 - > Still early, but very promising
- Editor features: syntax coloring, completion (methods, string escapes, regex, variables...), variable renaming, hyperlinking (go to definition)
- IDE features: running tests and scripts, Ruby projects, Rails projects, in-IDE WEBrick server



Demo: Netbeans Ruby Support

Measuring Compatibility

- How do you implement a language with no spec?
- ...and no complete suite of functional tests?
- Make do with what's available
 - > Partial suites (MRI's tests, BFTS, Rubicon, ruby_test)
 - > Applications' own suites (RSpec, Rails so far)
 - > Our own suite of tests, expanded over time
 - > Other implementations' tests (Rubinius, others soon?)
- Document as we go
 - > Community spec: www.headius.com/rubyspec
 - > JRuby mailing list archives, www.headius.com/jrubywiki

Compatibility: Ruby Tests

- Ruby's sample test (`ruby_src/sample/test.rb`)
 - > 95% or better passing
 - > Remaining is mostly POSIXy stuff we can't support
 - > Popular measure of compatibility (XRuby, Ruby.NET)
 - > ...but very limited in scope
- Ruby's language unit tests (`ruby_src/test/ruby`)
 - > 90% or better passing
 - > “second step” toward compatibility?
- Rubicon (rubyforge.org/projects/rubytets)
 - > Many tests 100%; perhaps 80% passing overall
 - > Not the most reliable suite, but oldest, largest

Compatibility: Rails 1.2.1

- > ActiveSupport
 - 498 tests, 1849 assertions, 4 failures, 0 errors (99.19%)
- > ActionPack
 - 1157 tests, 4868 assertions, 6 failures, 2 errors (99.30%)
- > ActionMailer
 - 64 tests, 133 assertions, 4 failures, 2 errors (90.63%)
- > ActionWebService
 - 96 tests, 531 assertions, 0 failures, 0 errors (100%)
- > ActiveRecord
 - 1012 tests, 3658 assertions, 41 failures, 6 errors (95.35%)
- > 2827 tests, 62 failures or errors (97.70%)

Compatibility: Others

- BFTS (rubyforge.org/projects/bfts)
 - > Maybe 90% passing
 - > Fairly complete, but for very few core classes
- ruby_test (rubyforge.org/projects/ruby_test)
 - > Unknown; just started looking at it recently
- RSpec's specs (rspec.rubyforge.org)
 - > 99% passing
- Our own regression suite: 100%!!!
 - > Coverage exceeds 80%

Measuring Performance

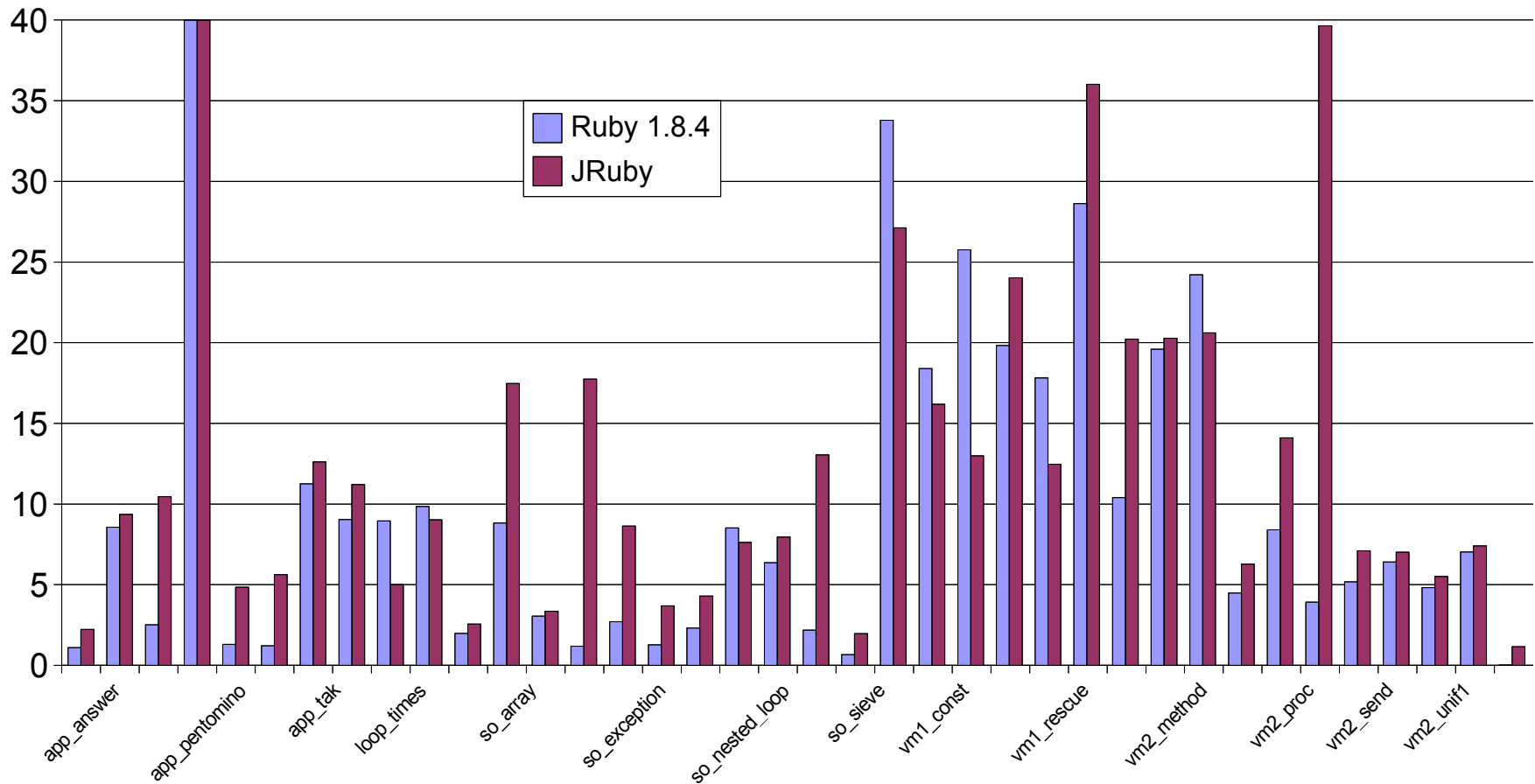
- No standard set of benchmarks for Ruby
- Alioth “Shootout” tests
 - > Wide range of algorithms tested
 - > Most are memory-intensive
 - > Much-maligned among Rubyists
- Ruby 1.9 benchmarks
 - > Narrow but solid range of tests
 - > Primarily testing areas YARV was designed to improve
 - > Subject of recent “Ruby shootout”
- Rails requests-per-second

Performance Metrics

- Rails requests-per-second (WEBrick)
- Still 100% interpreted, but some playing with JIT
- Ruby 1.8.4
 - > static - 301.09 req/s
 - > dynamic - 121.74 req/s
- JRuby 0.9.8 (Java 6 server VM)
 - > static – 180.03 req/s
 - > Dynamic – 50.53 req/s

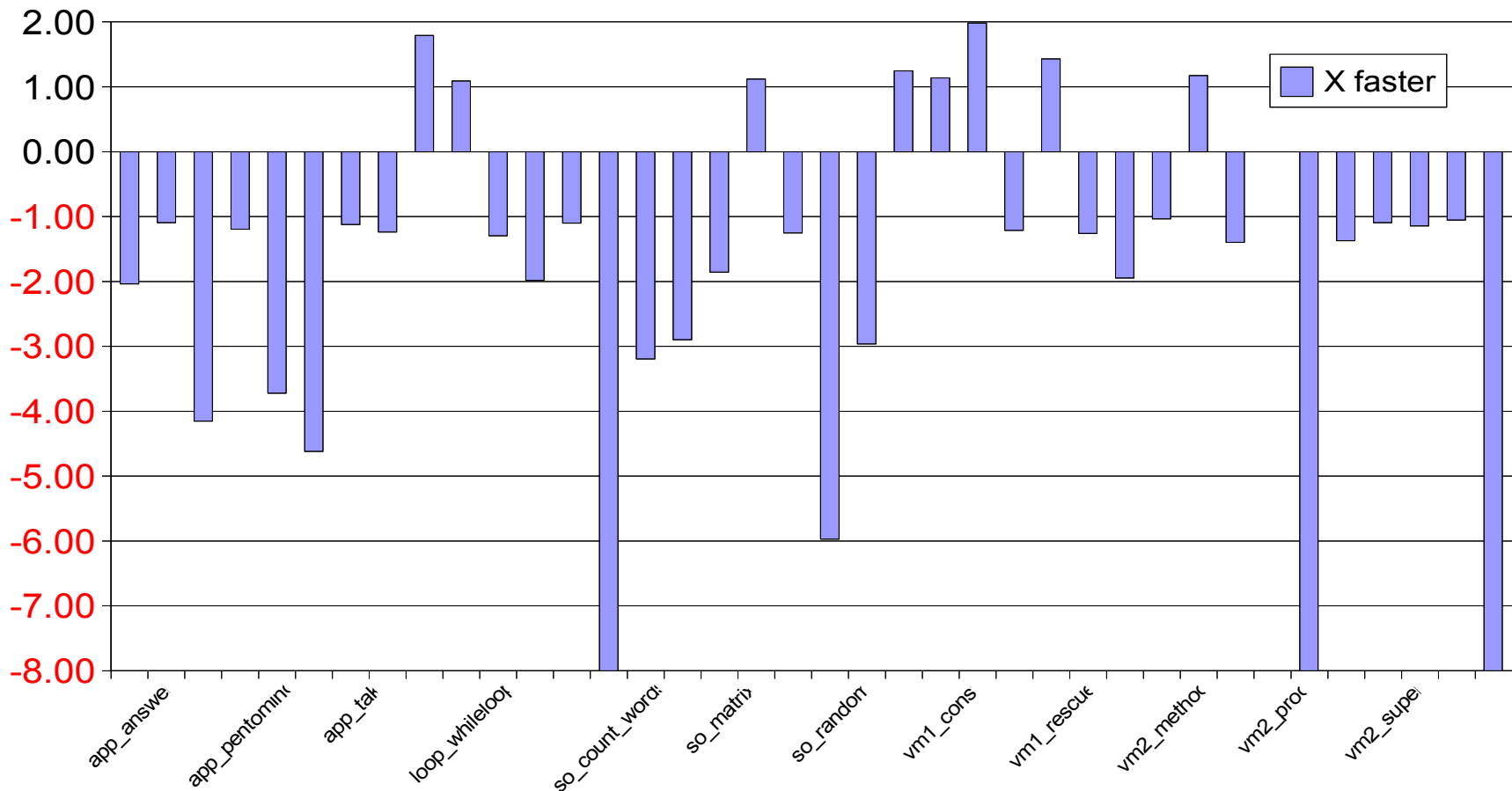
Performance: JRuby

Run Time



Performance: JRuby

Times Faster



JRuby Compiler

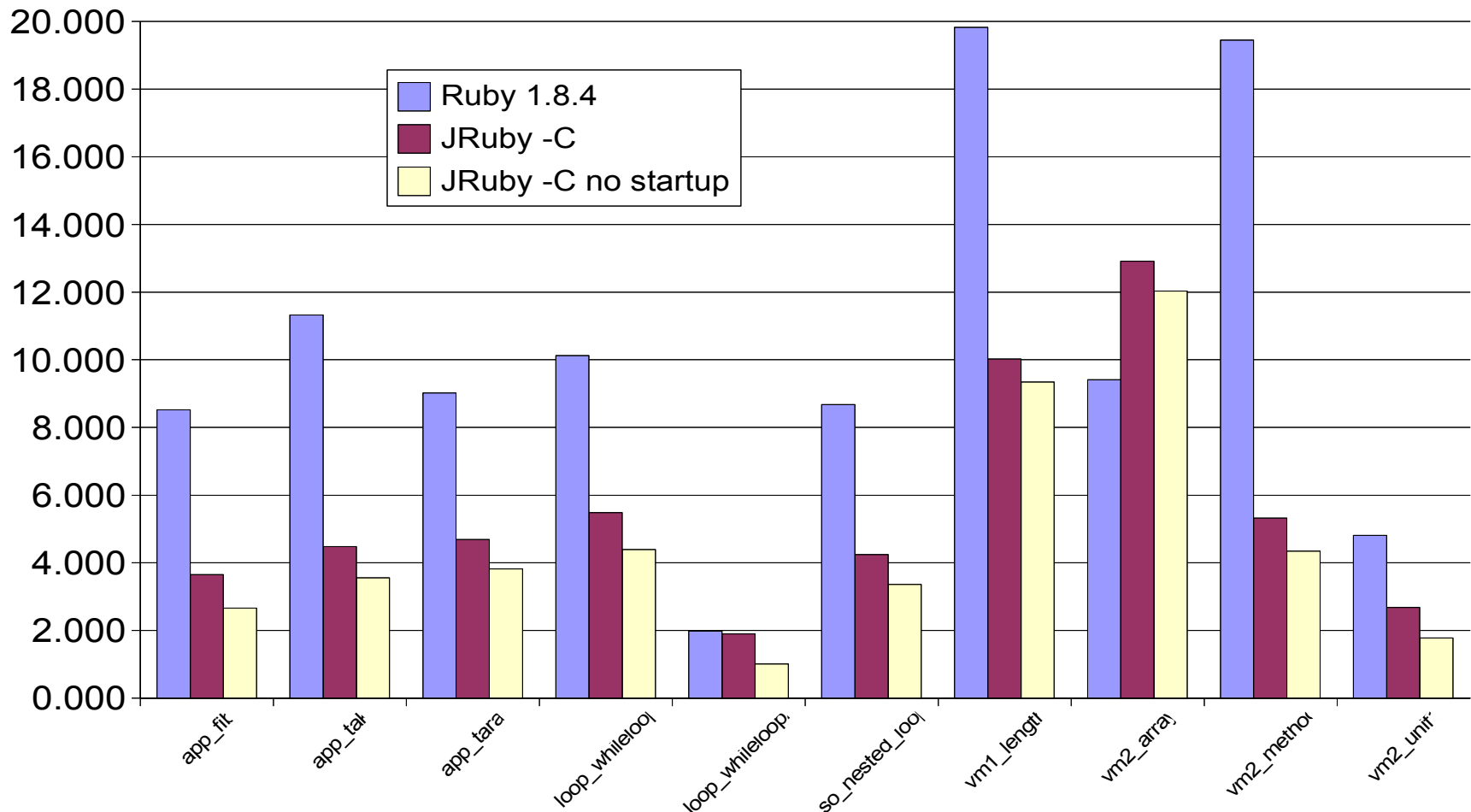
- Ahead-of-time
 - > Compile .rb file to .class file
 - > Directly executable, or require/loadable
 - > Package compiled Ruby like compiled Java
- Just-in-time
 - > Heavily hit methods compiled at runtime
 - > Run existing apps, scripts without modification
 - > Optimize methods generated at runtime
- Incremental design
 - > Fall back on interpreter for unimplemented bits



Demo: JRuby Compiler

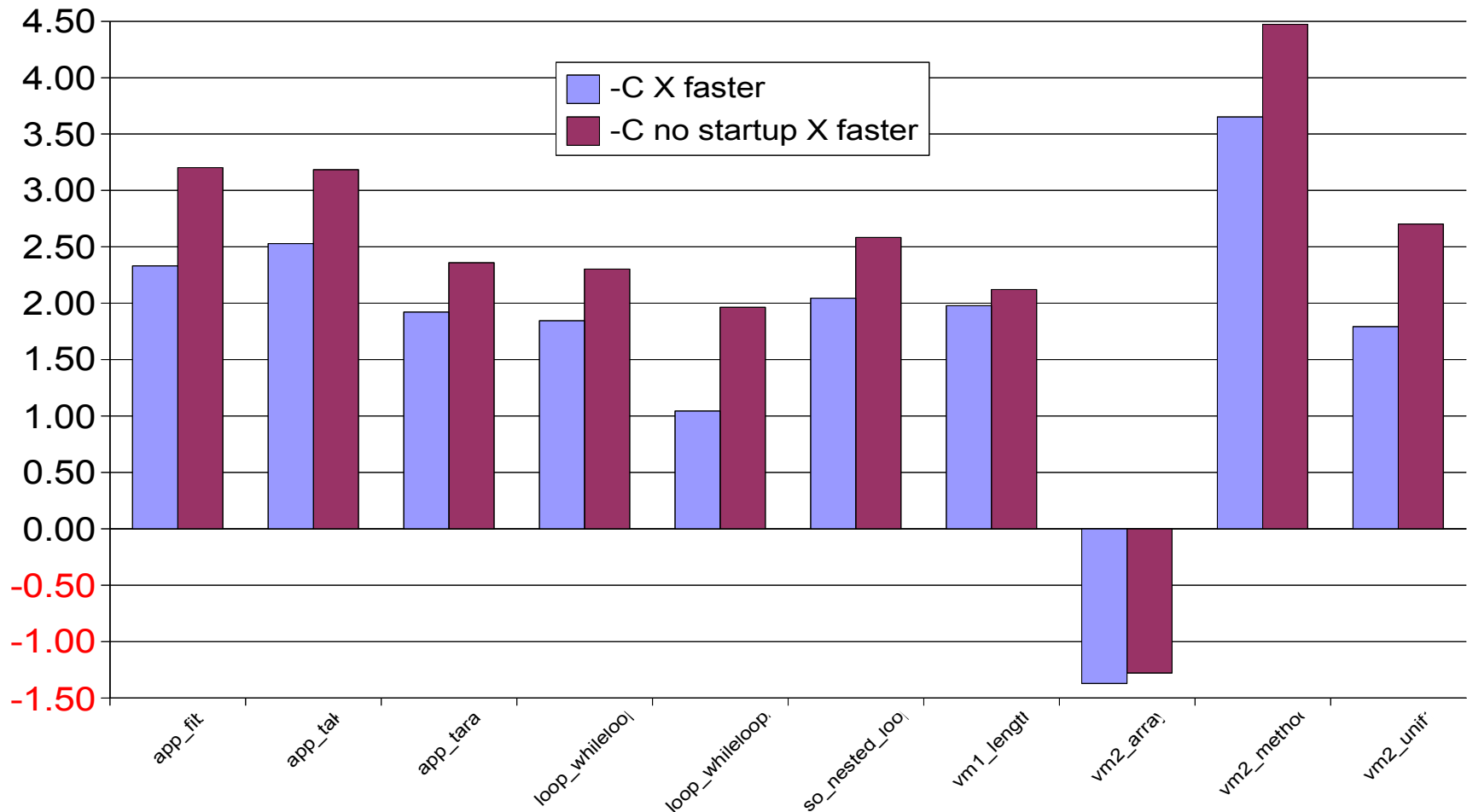
Performance: JRuby Compiler

Run Time



Performance: JRuby Compiler

Times Faster



What Next for JRuby?

- Viable for development today
 - > Not perfect, but what is?
- Rapidly approaching MRI on many fronts
- Many capabilities beyond MRI today
- More and more apps running
 - > Does yours?
 - > Shouldn't you ensure that it does?
- Bright future for Ruby on the Java platform

JRuby Roadmap

- Early March, 0.9.8 release
 - > Official “Rails Support”
 - > First release of experimental compiler
 - > Hundreds of bug fixes, some complete lib rewrites
- April timeframe, 0.9.9
 - > JIT compiler enabled, AOT handles “most” scripts
 - > Continuing app compatibility work
 - > Expanding Rails with WAR, enterprise API support
- May timeframe, 1.0RC?
 - > Wrapping up, smoothing the edges

More Information

- JRuby: www.jruby.org
- JRuby Wiki: headius.com/jrubywiki
- RubySpec Wiki: headius.com/rubyspec
- JRBuilder: www2.webng.com/bdortch/jrbuilder
- Cheri: www2.webng.com/bdortch/cheri
- NetBeans Ruby: wiki.netbeans.org/wiki/view/Ruby
- Charlie's Blog: headius.blogspot.com
- Tom's Blog: bloglines.com/blog/ThomasEEnebo



Become Super Powerful with JRuby

Q/A

Thank You!

charles.nutter@sun.com