

ORACLE[®]

C++, Java and .NET:

Lessons learned from the Internet Age, and What it Means for the Cloud and Emerging Languages

Cameron Purdy

Vice President, Development

Overview

A retrospective of the trade-offs compared to C++ illustrated by Java, C# and other VM-based programming languages with Garbage-Collection, why scripting languages simultaneously thrived, and what this teaches us about the applicability of technology to emerging challenges and environments such as cloud computing.



Overview (Alternate Version)

A study of the claim that you can get any talk accepted at JAOO '09 so long as the submission mentions cloud computing.



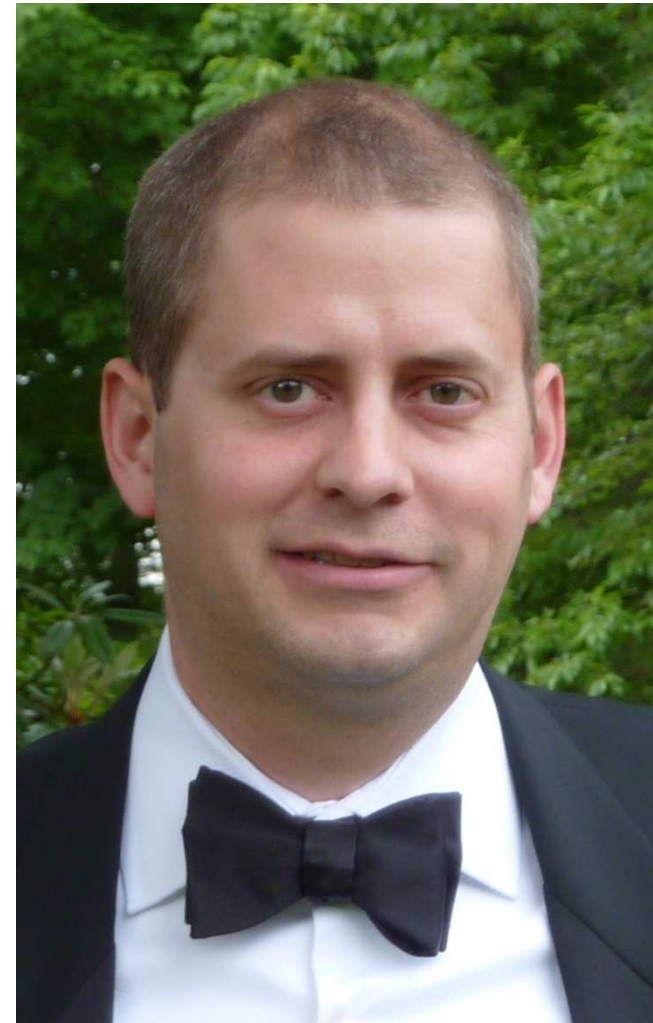
About the Speaker

- **PRESIDENT BARACK OBAMA**
- Barack H. Obama is the 44th President of the United States.
- His story is the American story — values from the heartland, a middle-class upbringing in a strong family, hard work and education as the means of getting ahead, and the conviction that a life so blessed should be lived in service to others.



About the Speaker

- Cameron Purdy is the Vice President of Development for Oracle Fusion Middleware, responsible for the Coherence Data Grid product, which has Java, C# and C++ versions
- Cameron was a founder and the CEO of Tangosol, acquired by Oracle in 2007
- Cameron has been working with C++ since 1994, Java since 1996, Java EE since 1999, and C# since 2001

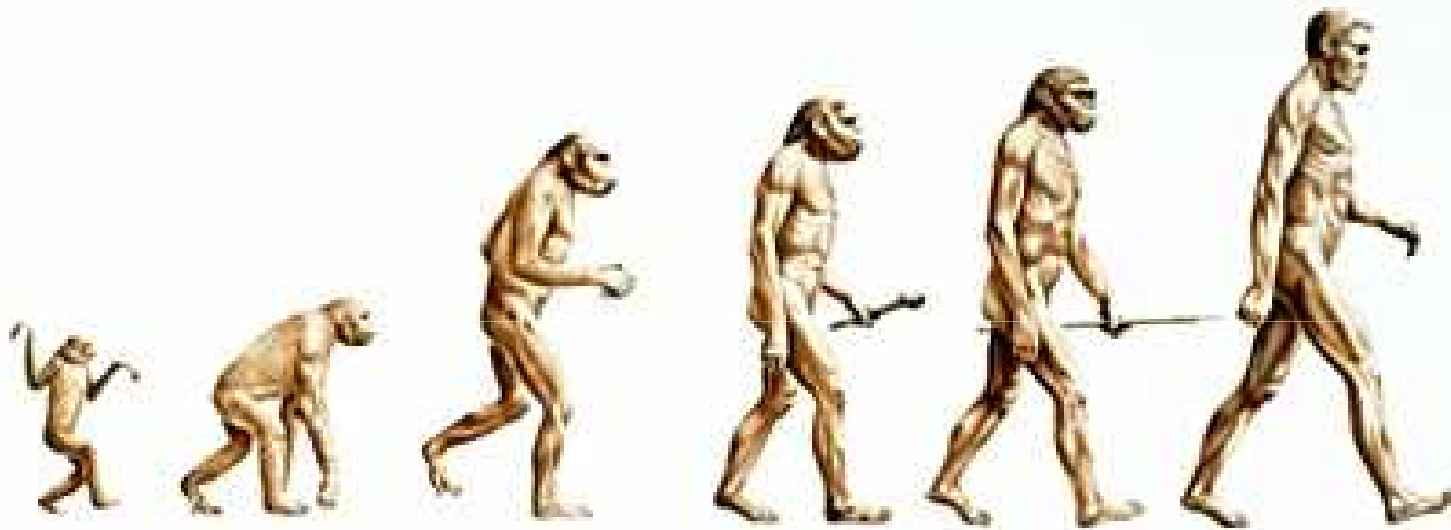




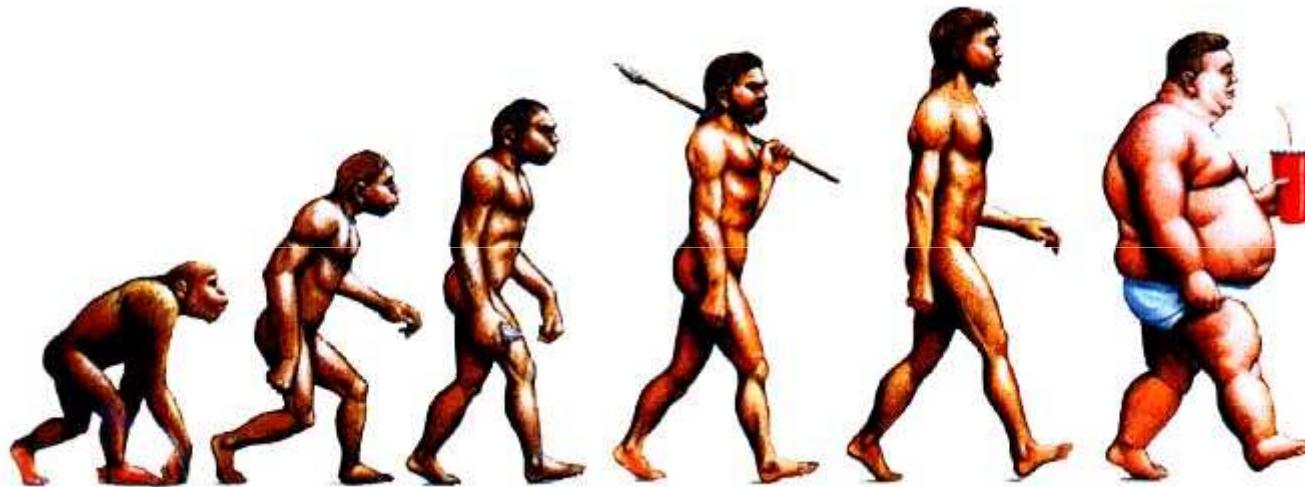
Disclaimer

The views expressed in this presentation are my own and do not necessarily reflect the views of my employer.

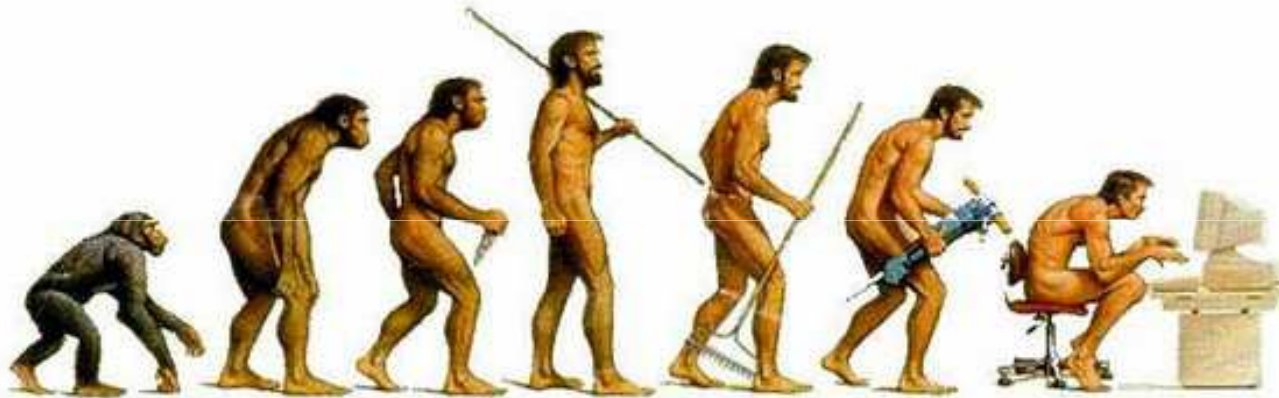
Java in the eyes of a Java Programmer



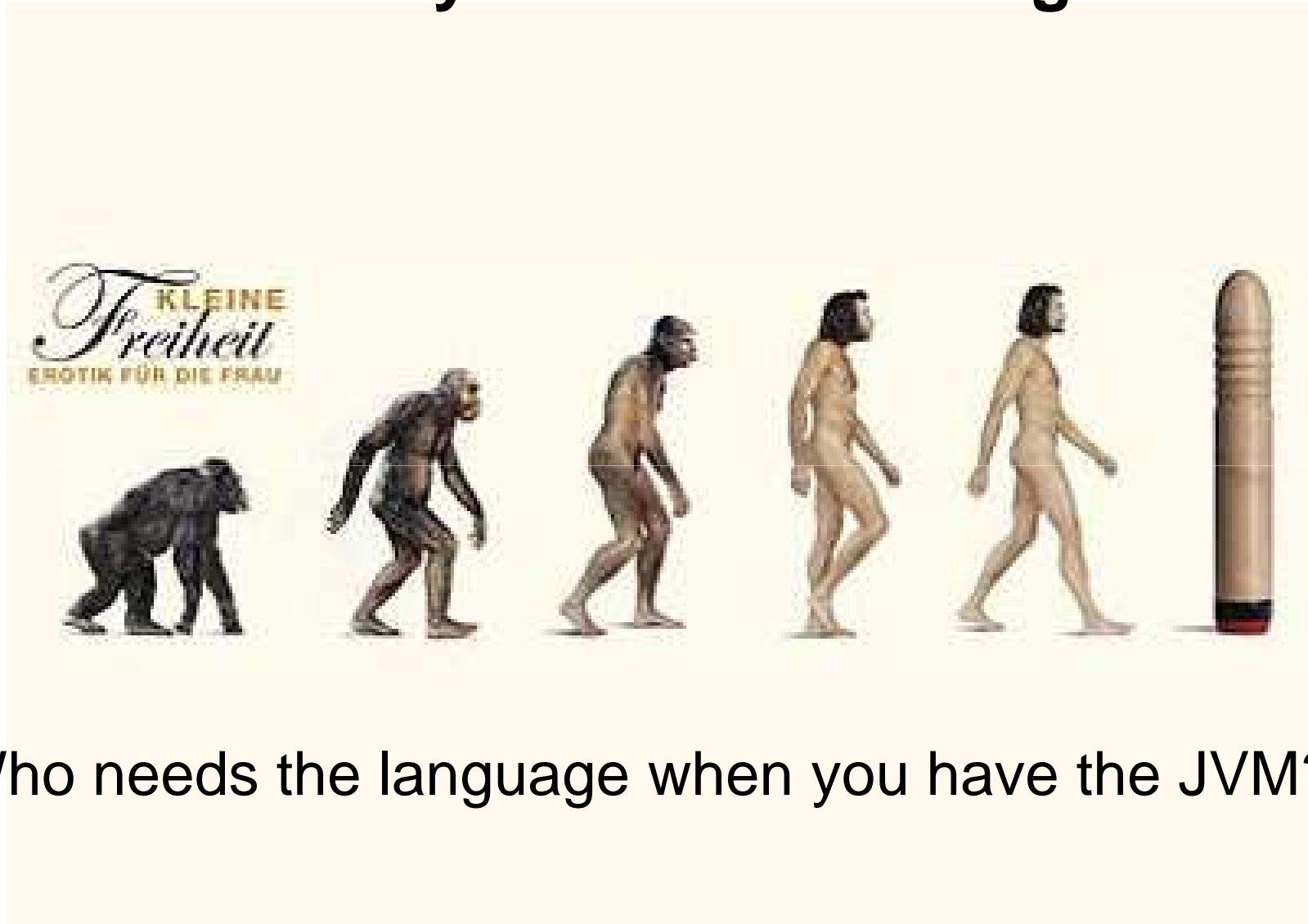
Java in the eyes of a C++ Programmer



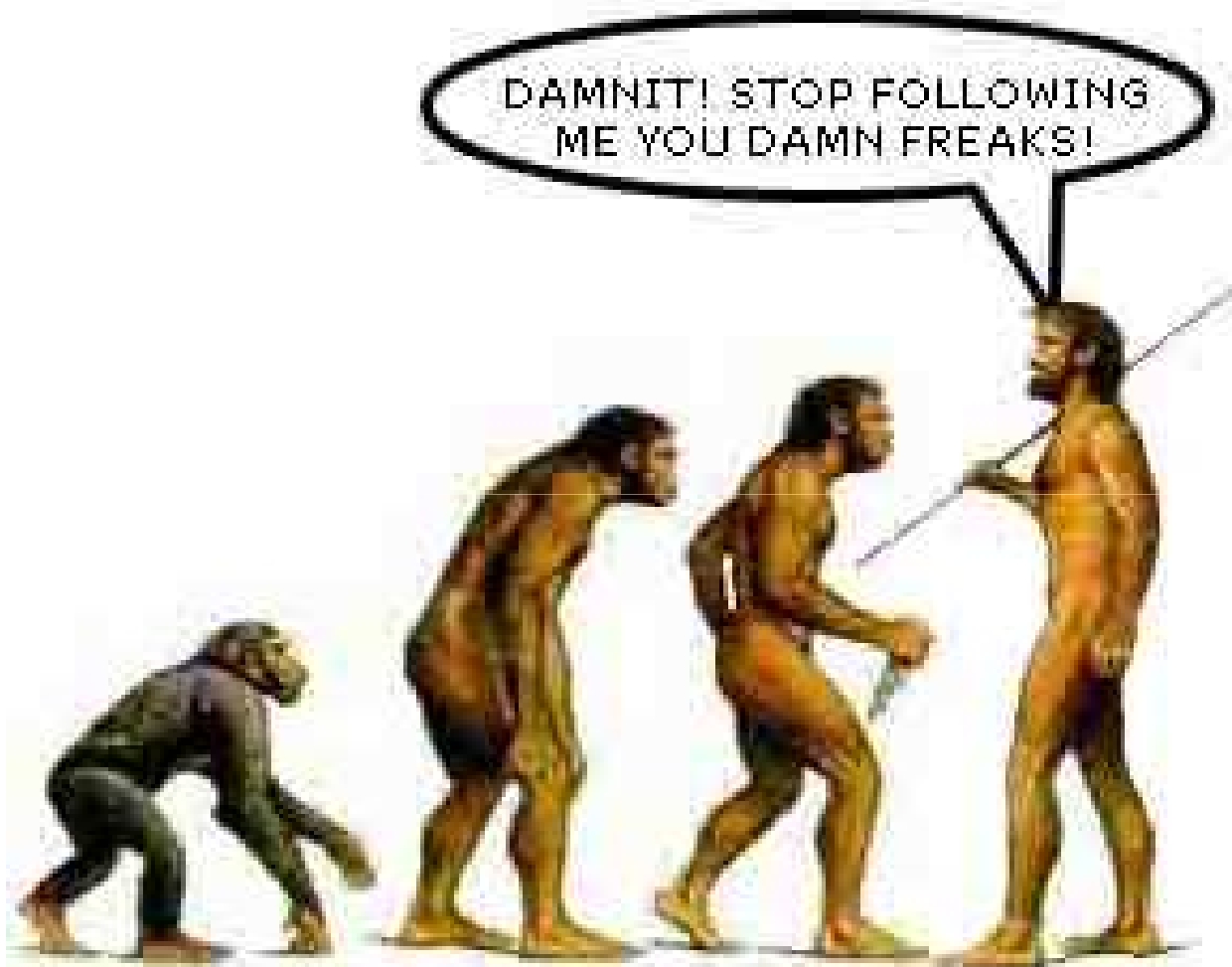
Java in the eyes of a Ruby Programmer



Java in the eyes of a Scala Programmer



Who needs the language when you have the JVM?





Top 10 Reasons Why Java *has* been able to supplant C++

Do not seek to follow in the footsteps of the men of old; seek what they sought.

- *Matsuo Basho*

Warning to Language Fanbois



Yes, I know that there are third party GC implementations for C++ and that you can get POSIX support on Windows from Cygwin, and
Don't go all Comicbook Guy on me. I'm describing reality as it *is*, not as how it *could* be.

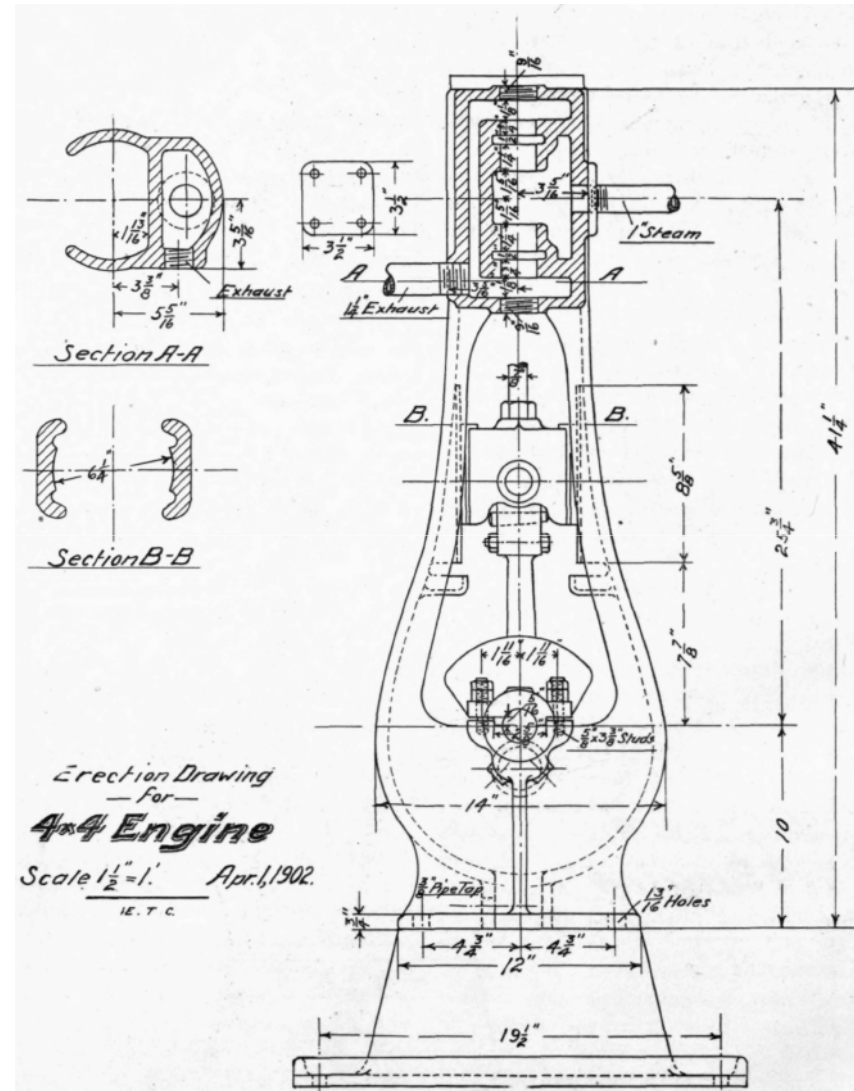
10. Automated Garbage Collection

- A significant portion of C++ code is dedicated to memory management
- Cross-component memory management does not exist in C++
 - Libraries and components are harder to build and have less natural APIs
- Faster time to market, lower bug count



9. The Build Process

- C++ builds are slow and complicated
- Personal example: 20 hour full build in C++ compared to 7 minutes in Java
- Multiply that times two: For debugging C++, you need a second build
- Tools such as Ant & Maven are available in Java
 - *make? nmake? Atrocious!*



8. Simplicity of Source Code and Artifacts

- C++ splits source into header and implementation files
 - Header trawling
 - Require big monitor to see .hpp and .cpp at the same time
 - Code in multiple places: Some inlined in the header, some in the .cpp
- Artifacts are compiler-specific, but there are many of them
 - Java: Just one .java, one .class



7. Binary Standard

- In addition to being loadable as a class by a JVM, a Java Classfile can be used to compile against
 - C++ has no binary standard
 - Precompiled headers?
Compiler and platform specific
 - C++ requires a large amount of source to be shipped in order to compile against it; brittle!
- Java defers platform-specific stuff to the runtime



6. Dynamic Linking

- No standard way to dynamically link to C++ classes
 - Compiler and platform specific
 - Messy at best
- Java allows arbitrary collections of classes to be packaged together and dynamically loaded and linked as needed
 - No DLL hell!



5. Portability

- Java is portable with very little effort; C++ is portable in theory, but in practice you have to build another language (#ifdef'd types, etc.) on top of it
- C++ has significant differences from vendor to vendor, e.g. standards support
 - Some *unnamed* major vendors have horrid support for the C++ standard, particularly templates
- “In theory, there’s no difference between theory and practice”



4. Standard Type System

- Java has:
 - Specified, portable primitive types
 - Built in, specified, portable runtime library
 - Rich support for I/O, networking, XML/HTML, database connectivity
- C++ has:
 - 16-bit? 32-bit? 64-bit? *80-bit!?!?*
 - Multi-threading? You must be joking
 - STL? Maybe some day
 - *Basically nothing is standard!*



3. Reflection

- Full runtime capability to look at the runtime
 - C++ has optional RTTI but no reflection
- Enables extremely powerful generic frameworks
 - Ability to learn about, access and manipulate any object



2. Performance

- GC can make memory management much more efficient
 - Slab allocators
 - Escape analysis
- Multi-threaded? No support in C++
- Thread safe smart pointers 3x slower than Java references
- Hotspot can do massive inlining
 - Very important for dealing with layers of virtual invocation



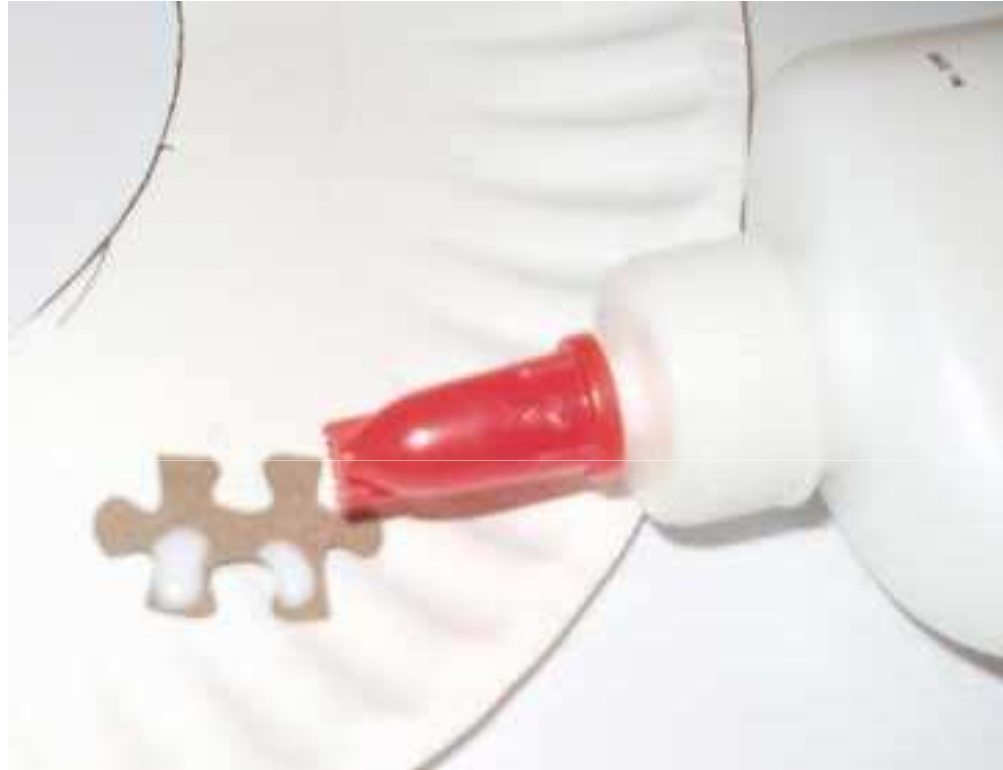
1. Safety

- Elimination of pointers
 - Arbitrary memory access
 - Ability to easily crash the process (core dump)
- No buffer overruns
 - Code and data cannot be accidentally mixed
- Bounds checked
 - All raw access via arrays and NIO buffers
 - Impossible to read/write outside of the bounds of either



Honorable Mention: C++ Templates

- Fugly
 - Atrocious syntax (IMHO)
 - Metaprogramming is near unreadable
 - Advanced features are not explainable to mortals
 - Only Bjarne understands
- Bloat
 - The compiler does the cut & paste for you
 - Personal example: 80MB binary





Top 10 Reasons Why Java has *not* been able to supplant C++

Old ideas give way slowly; for they are more than abstract logical forms and categories, they are habits, predispositions, deeply ingrained attitudes of diversion and preference.

- *John Dewey*

10. Startup Time



- The graph of initially loaded classes is pretty large
 - Loading
 - Validation
 - JITting
 - Initializer code
- The code is either getting JITted to native code each time the JVM starts, or it's initially being interpreted (Hotspot)
- Conclusion: Not good for “instant” and short-running processes

9. Memory Footprint



- Java uses significantly more memory than C++, particularly for “small” applications
 - Easily two orders of magnitude more at the extreme end of the scale
- Memory requirements limit Java adoption on some devices

8. Full GC Pauses



PRINCE FLORIMOND FINDS THE SLEEPING BEAUTY

- Sooner or later, there is a part of GC that can't be run in the background and can't be avoided
 - Results in a temporary halt
 - Makes real time impossible
- Havoc for distributed systems
 - Is the process dead? Locked up? Or in GC?
- Possible partial amelioration by increasing the memory footprint

7. No Deterministic Destruction



- No support for RAII
- Cannot count on finalizers
 - What if you own a socket? A thread? A mutex!?!?
- Not even a “using” construct in Java (there is one in C#)

6. Barriers to Native Integration



- Operating Systems are built in C/C++
 - APIs are typically C
 - Native GUIs and other OS-level functionality often require C (e.g. cdecl or pascal) callbacks
- JNI allows Java to call ... native code that was written explicitly to be called by JNI



So it wasn't a Top 10 list ...

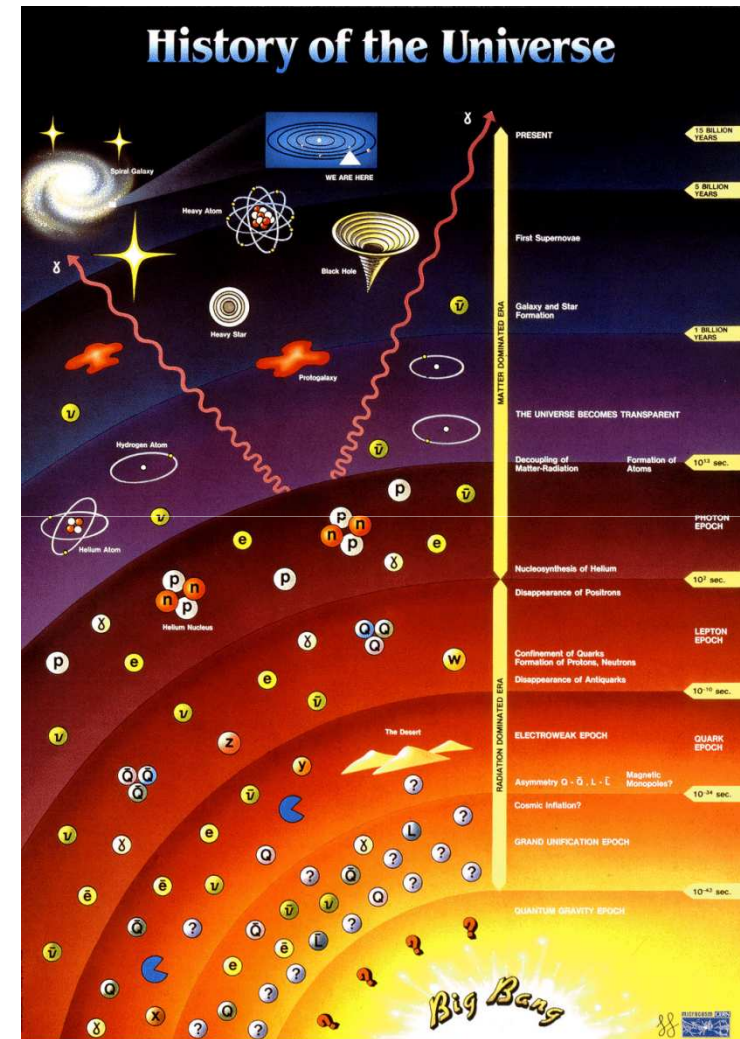
Top ~~10~~ 5 Reasons Why Java has *not* been able to supplant C++

Old ideas give way slowly; for they are more than abstract logical forms and categories, they are habits, predispositions, deeply ingrained attitudes of diversion and preference.

- John Dewey

Understanding the Shift to Java and C#

- Shift Happens
 - Internet & the World Wide Web
 - HTML Browser: No client-side software per application, no installation, no configuration
 - Eliminated startup time, memory footprint and native (e.g. GUI) integration benefits of C++
 - Required faster iterative development, better class libraries, modularity, long running capability (no memory leaks), multi-threading support and safety on the server



Scripting Languages

- Simplicity & Approachability
 - Hooks up to databases
 - Manages state on behalf of the user
 - Produces HTML
- Rapid Application Development
 - No OO Architectural Requirements
 - No compile step; save and refresh
- High Density
 - Designed for shared hosting
 - Much lower initial memory footprint
 - Stateless processes are easy to scale, easy to cycle





Cloud Computing

New roads; new ruts.
- *G. K. Chesterton*

What are we missing?

- Virtual Machine
 - Modularity, Lifecycle & Isolation
 - Lower memory footprint
 - Predictable GC pauses
- Platform
 - Distributed system as a system
 - Provisioning and Metering
 - Cloud Operating System APIs
 - Persistence (including key/value)
 - Map/Reduce-style processing
- Application Definition
 - Packaging, Resource Declaration
 - Security



Rethinking the Container for the Cloud

- What's changed in the world since Java was introduced?
 - Hardware Virtualization
 - Stateful Grid Infrastructure
 - Capacity On Demand ISPs (EC2)
- What's coming in Java?
 - NIO pluggable file systems
- Conclusion: Java will either step up or get replaced





Emerging Languages

And from the discontent of man
The world's best progress springs.
- *Ella Wheeler Wilcox*



Scala

- The Scala language runs on the JVM
 - Compiles to Java byte code
 - Scala code can easily use existing Java libraries
- Similar to Java, but:
 - Eliminates primitives: Every value is an object
 - Supports functional programming
 - Function objects
 - Closures
 - Higher order functions
 - Includes XML processing capability
 - Adds simple tail recursion
- Expect to see Scala talks at the next JA00

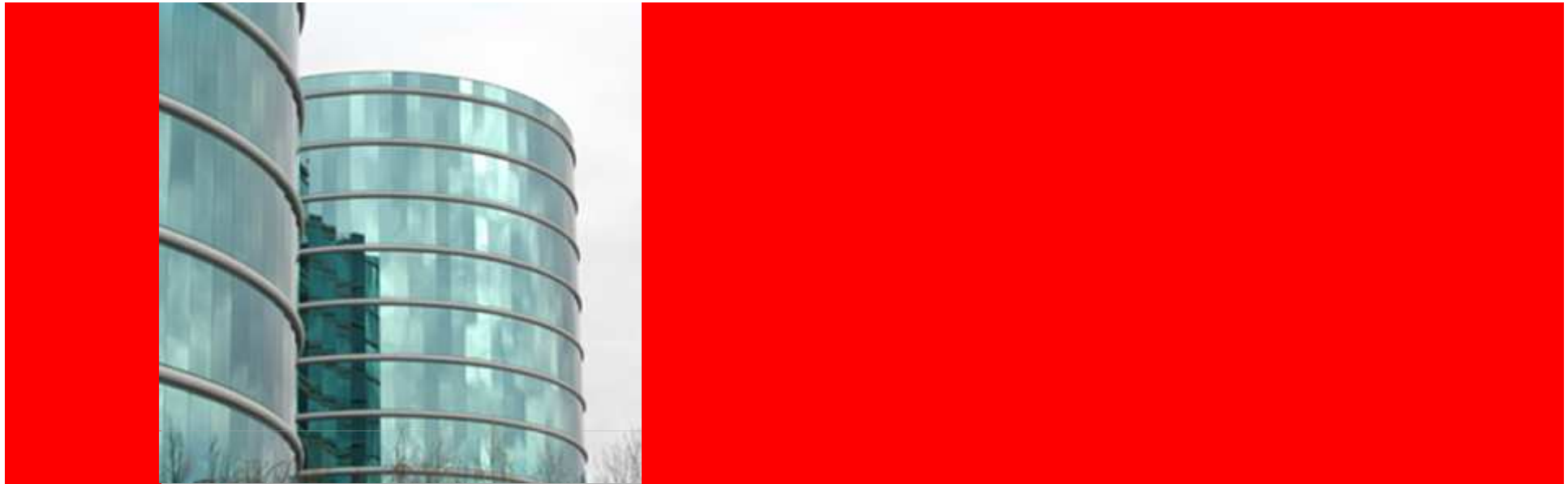


Scala Traits

- Traits are like sub-classes
- Allow overriding of methods
- Instead of inheriting from a class, a trait is used to extend any number of classes

```
abstract class IntQueue {  
  def get(): Int def put(x: Int)  
}  
trait Doubling extends IntQueue {  
  abstract override def put(x: Int) {  
    super.put(2 * x)  
  }  
}
```





ORACLE[®]

C++, Java and .NET:

Lessons learned from the Internet Age, and What it Means for the Cloud and Emerging Languages

Cameron Purdy

Vice President, Development