# Java

# C/C++

Cliff Click
www.azulsystems.com/blogs

# Java vs C/C++

- "I declare a Flamewar!!!!"

  - ***Lots*** of noise & heat
  - Not many facts
  - Lots of **obvious** mistakes being made
- Situation is more subtle than expected
- This is my attempt to clarify the situation

# C/C++ Beats Java

- Very small footprint – under 300KB
  - e.g. Embedded controllers, cars, clocks
- Very deterministic or fast (re)boot times –
  - e.g. engine controllers, pacemakers
- Very big problems: Fortran optimizations
  - Array reshaping & tiling for cache
- Value types - Complex, Point
  - e.g. Overhead costs of 1b objects
  - vs array-of-doubles

# C/C++ Beats Java

- Direct Machine Access
  - e.g. OS's (special ops, registers), device drivers
    - Hard to do in Java (i.e. JavaOS effort)
  - AAA Games / First Person Shooter Games
  - Maxine Java-in-Java might be a counter-example
- Direct Code-Generation
  - gnu "asm"
  - Write bits to buffer & exec
    - '`sort`' inner loop key-compare
  - Interpreters

# C++ Beats Java

- Destructors vs finalizers
  - Destructors are reliable out-of-language cleanup
  - Finalizers will "eventually" run
    - But maybe after running out of e.g. file handles
    - So weird force-GC-cycle hooks to force cleanup
- Destructors vs & try/finally
  - Destructors are reliable exit-scope action
  - try/finally requires adding explicit exit-scope-action
    - For each new enter-scope-action
    - Maintenance mess

# Java Beats C/C++

- Most Programs - profiling pays off
  - But nobody bothers for C/C++, too hard
  - All JIT systems profile at least some
  - More profiling added as systems mature
- Very Large Programs >1MLOC
  - Large program tool chain is better
  - A **lot** more 1MLOC Java apps than C

# Java Beats C/C++

- GC is easier to get right than malloc/free
  - Faster time-to-market
  - Why so many variations on Regions, Arenas, Resource Areas?  Basically hand-rolled GC...
- GC is efficient
  - Parallel, concurrent
  - Good locality, fragmentation

- GC allows concurrent algorithms
  - Trivially track shared memory lifetimes
  - Fundamental change, can't "fake it"

# Java Beats C/C++

- Single CPU speed stalled
  - Bigger problem => parallel solution
- Better multi-threading support
  - Real Memory Model - synchronized, volatile
  - Threads are built-in
  - Large multi-threaded library base
    - JDK Concurrent Collections
  - GC vs concurrent malloc/free
- Tools for parallel coding, debugging

# Libraries

- Vast Java Library collection
  - Can COTS many many problems
- Downside: too many 3rd party libraries
  - Java Mentality: download from web, don't build
  - C Mentality: build before download
  - Too many layers of Java crap
  - Nobody knows what's going on
- Application plagued by failures no one understands

# Claims C-beats-Java But I Dont Think So

- Most modest sized programs
  - Fast enough is fast enough
- 16bit chars vs 8bit chars
  - Lots of noise here (and lots of optimizations)
  - Rarely makes a difference in practice
- Raw ***small*** benchmark speed
  - Usually *I don't care*
    - "C gets more BogoMips so it's better!"
  - OR broken testing methodology
    - "C makes a better WebServer because printf is faster!"

# Common Flaws When Comparing

- No Warmup
  - Only interesting for quick-reboot, e.g. Pacemakers
  - Most apps run for minutes to days
- Basic timing errors
  - API reports in nanos
  - OS rounds to millis (or 10's of millis)
- Caching Effects
  - CPU caches, OS-level, disk & network
  - DB cache, JIT/JVM level

vs

# Common Flaws When Comparing

- Basic Broken Statistics
  - Run-once-and-report
  - No averages, std-devs
  - Throwing out "outliers"
  - Not accounting for "compile plan"
    - "Statistically rigorous Java performance evaluation"
    - "Producing wrong data without doing anything obviously wrong!"

- Flags, version-numbers, env-factors all matter
  - "java" not same as "java -client" or "java -server"
  - Some JDK versions have 30% faster XML parsing

# Common Flaws When Comparing

- Varying Datasets or Constant-time workloads
  - Have seen cycles-per-work-unit vary by 10x
- Claiming X but testing Y
  - 209_db: claims DB test but is shell-sort test
  - SpecJBB: claims middleware test but is GC test
  - Lots more here
- Not comparing same program
  - e.g. Debian language shootout
    - http://shootout.alioth.debian.org

# Commonly Mentioned Non-Issues

- Stack Allocation "Does So" beat GC
  - Does Not. You got evidence? I got evidence of non-issue...

- Java has lots of casts
  - But they are basically free
    - load/compare/branch, roughly 1 clock
- Virtual & Interface calls are slow
  - And basically never taken (inline-cache)
- C# curious? I dunno, I don't track Microsoft

# Java-vs-C Examples

- Examples limited to what I can fit on slides
- In-Real-Life never get apples-to-apples
- Programs either very small
- Or new re-implementation
  - Generally better written 2nd go-round
- Or extremely bad (mis)use of language features

# Example: String Hash

- Java tied vs GCC -O2

```
int h=0;
for( int i=0; i<len; i++ )
  h = 31*h+str[i];
return h;
```

Here I ran it on a new X86 for 100 million loops:

```
> a.out         100000000
100000000 hashes in 5.636 secs
> java str_hash 100000000
100000000 hashes in 5.745 secs
```

- Key is loop unrolling
  - (i.e. JITs do all major compiler optimizations)

# Sieve of Erathosthenes

- Again tied

```
bool *sieve = new bool[max];
for (int i=0; i<max; i++) sieve[i] = true;
sieve[0] = sieve[1] = false;
int lim = (int)sqrt(max);
for (int n=2; n<lim; n++) {
  if (sieve[n]) {
    for (int j=2*n; j<max; j+=n)
      sieve[j] = false;
  }
}
```

```
I computed the primes up to 100million:

    > a.out      100000000
    100000000 primes in 1.568 secs
    > java sieve 100000000
    100000000 primes in 1.548 secs
```

# Silly Example

- Silly Example to Make a Point

```
int sum=0;
for (int i = 0; i < max; i++)
  sum += x.val();  // virtual call
return sum;
```

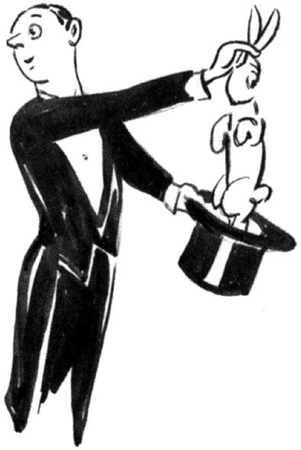
```
Here I run it on the same X86:

    > a.out      1000000000 0
    1000000000 adds in 2.657 secs
    > java vcall 1000000000 0
    1000000000 adds in 0.0 secs
```

- Zounds!  Java is "infinitely" faster than C

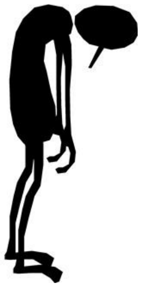
??? what happened here ???

# Silly Example Explained

- Command-line flag picks 1 of 2 classes for 'x'
- Type profiling at Runtime
  - Only 1 type loaded for 'x.val()' call
    - `"int val() { return 7; }"`
  - JIT makes the **virtual** call static, then inlines
    - `"for( int i=0; i<max; i++ ) { sum += 7/*x.val*/; }"`
  - Once inlined, JIT optimizes loop away
    - `"sum += max*7;"`



- True virtual call at static compile-time
  - No chance for a static compiler to optimize



# Why Silly Example Matters

- Only 1 implementing class for interface
- Common case for large Java programs
  - Single-implementor interfaces abound
  - Library calls with a zillion options
    - But only a single option choosen, etc
  - Can see 100+ classes collapsed this way
    - 10K call-sites optimized, 1M calls/sec optimized
- Major Optimization not possible without JIT'ing
- Lots more cool JIT tricks to come...

# Other Stuff That Matters

- Other Things Also Matter
    - Existing infrastructure, libraries, time-to-market
    - Programmer training, mind set
        - Lots of Java programmers Out There
    - Legal issues – open source or man-rating
    - Reliability, stability, scalability
- JVMs enabling new languages
    - Clojure, Scala, JRuby, Jython, many more
    - Much faster time-to-market

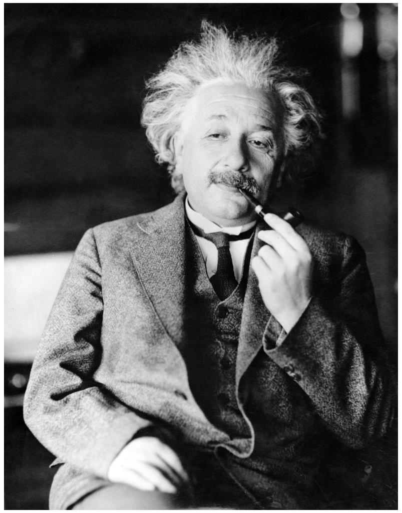
# Summary

- My Language is Faster!!!
  - Except when it's not
  - Ummm.... "fast" is not well-defined...
    - MOOPS/sec?  Faster than thy competitor? Faster-to-market?  Fits in my wrist watch?

- Other-things-matter more in many domains
  - If you got 500 X programmers, maybe should use X?
- Each language is a clear winner in some domains, neither going away soon
  - e.g. still room for trains in our auto-dominated world

# Summary

- Internet is a Great Amplifier
  - Of both the Good, the Bad AND the Ugly
- Real issue: Need Sane Discourse
  - Lots of Screaming & Flames
    - People with strong opinions, different vested interests, different experiences & goals
    - e.g. Do we even agree on what "faster" means?
  - Lots of Bad Science
    - Broken & missing statistical evidence
    - Misapplied testing, testing unrelated stuff



# Summary

- When the noise exceeds communication levels...
  - Back up, clarify, acknowlege each side has strengths
  - Chill out, think it through
- Recognize a lack-of-evidence for what it is
  - yelling louder about what you do know doesn't help
  - Good testing helps  (and bad testing hurts)
- Realize "faster" has different meanings
  - Junior Engineer thinks "faster than the competition"
  - Manager thinks "faster to market"
  - Senior Engineer thinks "that brick wall is approaching fast!"

# Summary

**It Depends.**

Cliff Click
http://www.azulsystems.com/blogs