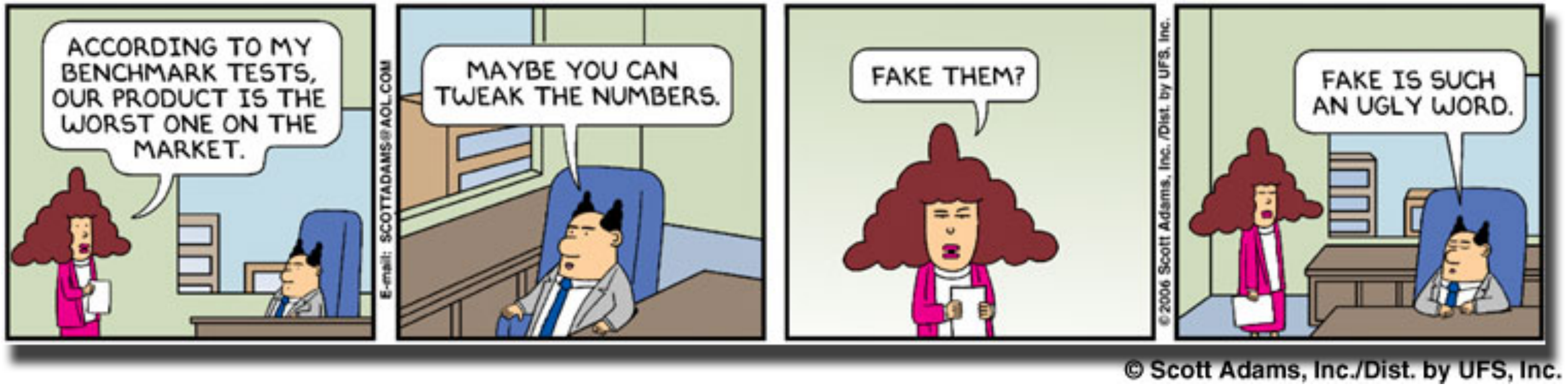
Wake Up and Smell The Coffee

Evaluation Methodology for the 21st Century

Stephen M Blackburn, Kathryn S McKinley, Robin Garner, Chris Hoffmann, Asjad M Khan, Rotem Bentzur, Amer Diwan, Daniel Feinberg, Daniel Frampton, Samuel Z Guyer, Martin Hirzel, Antony Hosking, Maria Jump, Han Lee, J Eliot B Moss, Aashish Phansalkar, Darko Stefanovic, Thomas VanDrunen, Daniel von Dincklage, Ben Wiedermann


ACCORDING TO MY BENCHMARK TESTS, OUR PRODUCT IS THE WORST ONE ON THE MARKET.
E-mail: SCOTTADAMS@AOL.COM
MAYBE YOU CAN TWEAK THE NUMBERS.
FAKE THEM?
© 2006 Scott Adams, Inc./Dist. by UFS, Inc.
FAKE IS SUCH AN UGLY WORD.
© Scott Adams, Inc./Dist. by UFS, Inc.

“...improves throughput by up to 41x”

“speed up by 10-25% in many cases...”

“...about 2x in two cases...”

“...more than 10x in two small benchmarks”

“speedups of 1.2x to 6.4x on a variety of benchmarks”

“can reduce garbage collection time by 50% to 75%”

“...demonstrating high efficiency and scalability”

“our prototype has usable performance”

There are lies, damn lies, and benchmarks

“sometimes more than twice as fast”

“our algorithm is highly efficient”

“garbage collection degrades performance by 70%”

“speedups.... are very significant (up to 54-fold)”

“our .... is better or almost as good as .... across the board”

“the overhead .... is on average negligible”

The success of most systems innovation hinges on benchmark performance.

Predicate 1. Benchmarks reflect current (and ideally, future) reality.

Predicate 2. Methodology is appropriate.

Benchmarks & Reality

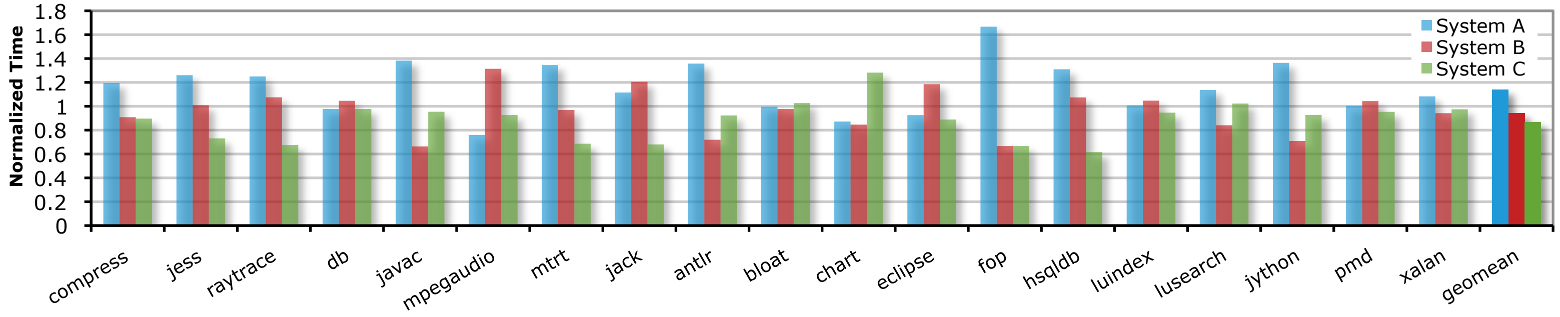
1. JVM design & implementation
   – SPECjvm98 is small and jbb is relatively simple
     - *Q: What has this done to GC research?*
     - *Q: What has this done to compiler research?*
2. Computer architecture
   – ISCA & Micro rely on SPEC CPU
     - *Q: What does this mean for Java and C# performance on modern architectures?*
3. C#
   – Public benchmarks are almost non-existant
     - Q: How has this impacted research?

Benchmarks & Methodology

- We're not in Kansas anymore!
  - JIT compilation, GC, dynamic checks, etc
- Methodology has not adapted
  - Needs to be codified and mandated

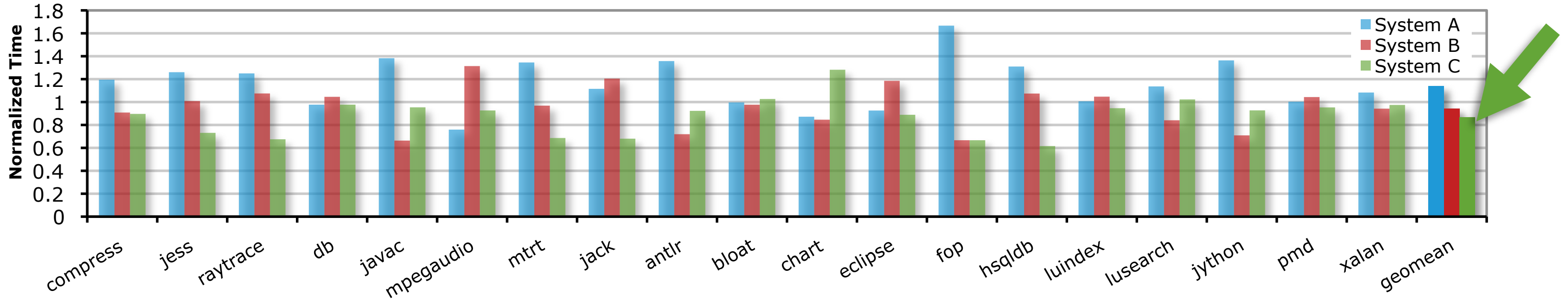
"...this sophistication provides a significant challenge to understanding complete system performance, not found in traditional languages such as C or C++"
[Hauswirth et al OOPSLA '04]

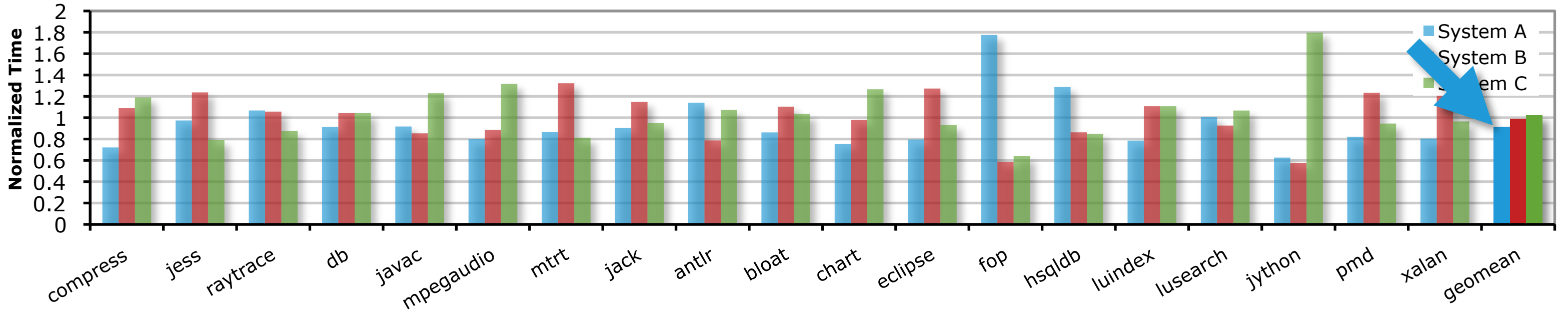
Benchmarks & Methodology

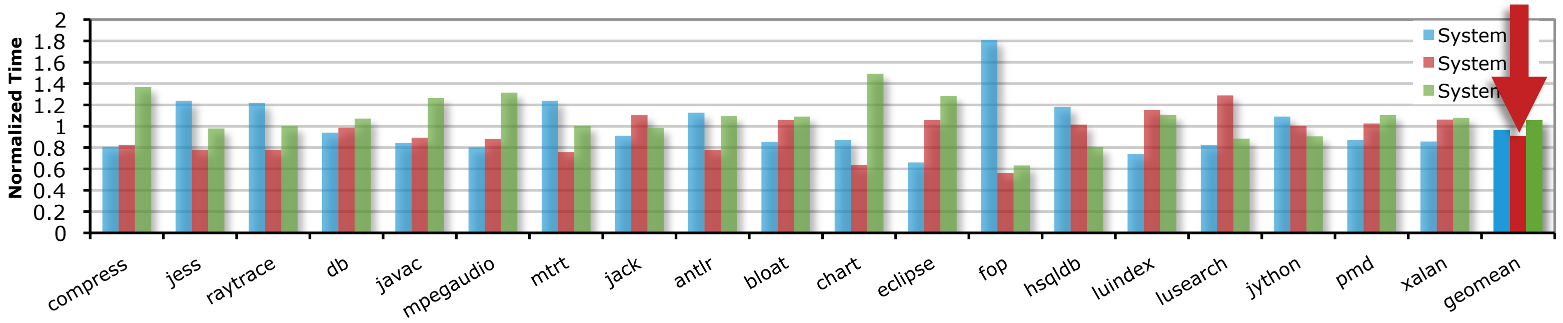


- Comprehensive comparison
  - 3 state-of-the-art JVMs
  - Best of 5 executions
  - 19 benchmarks
  - 1 platform
- 3 students perform the same evaluation…

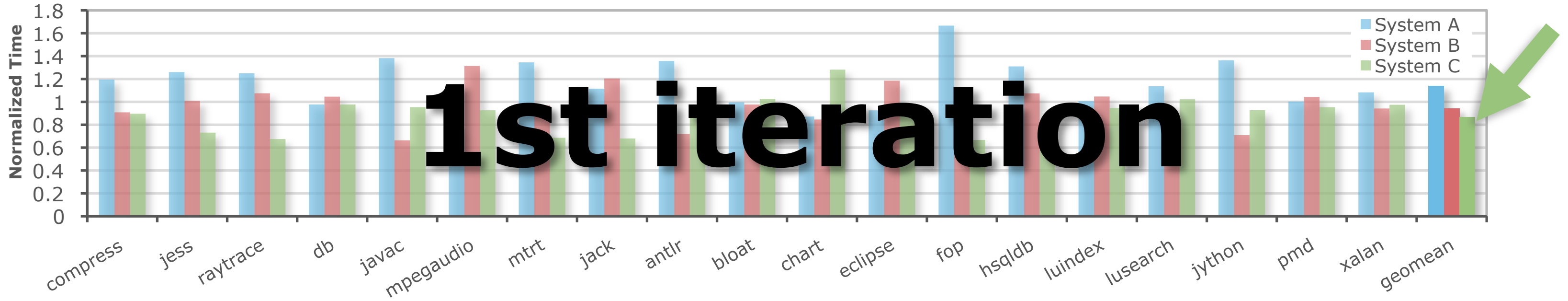
Benchmarks & Methodology

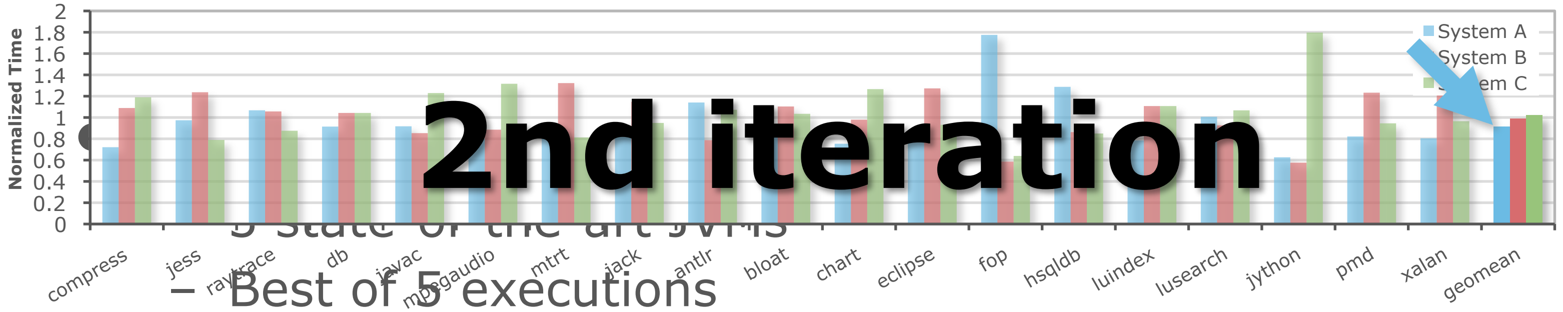






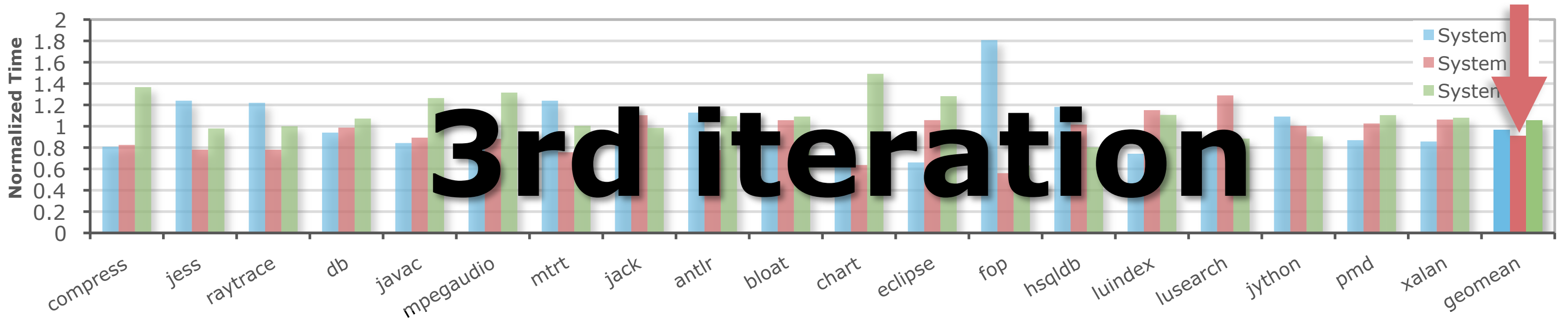
Benchmarks & Methodology



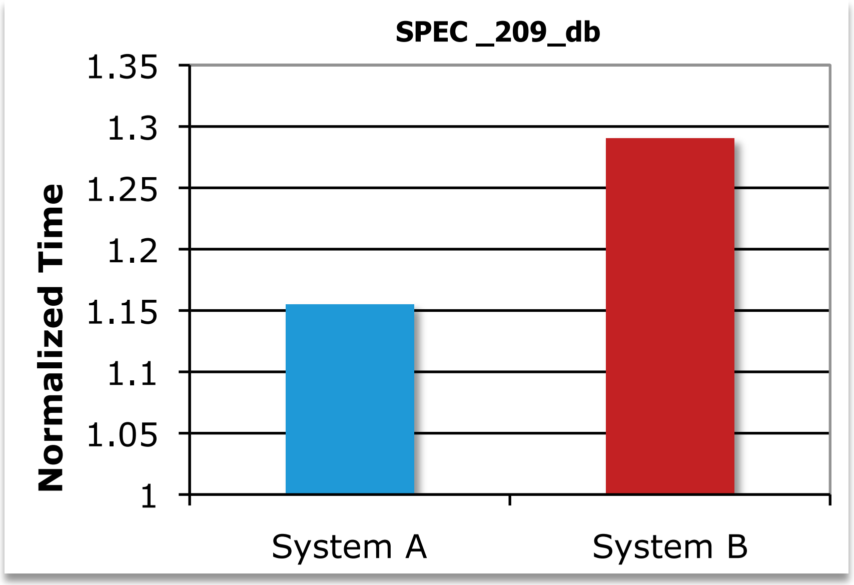


3 state of the art JVMs

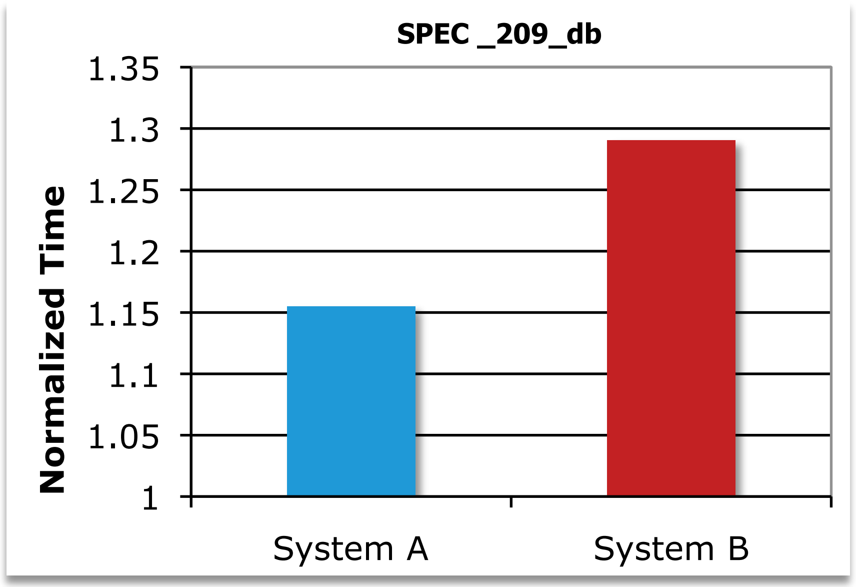
– Best of 5 executions

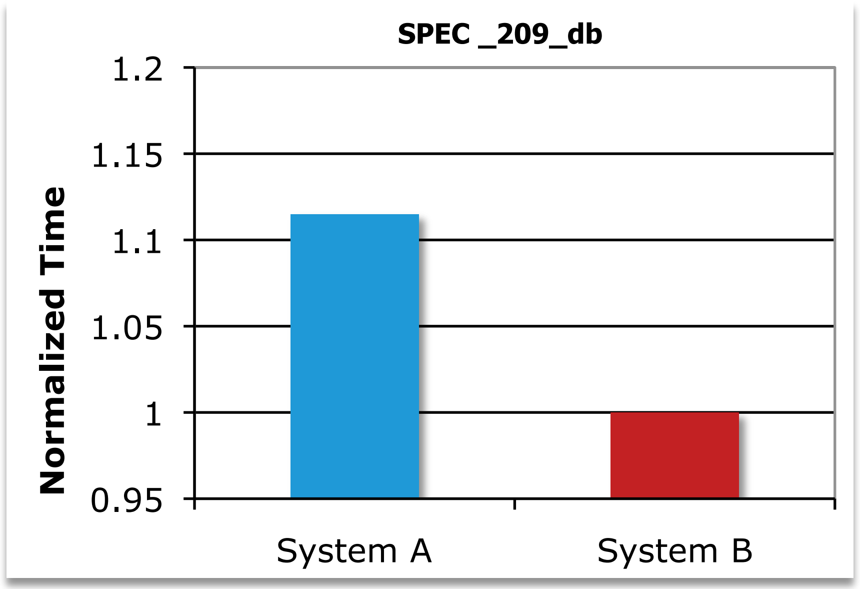


Benchmarks & Methodology



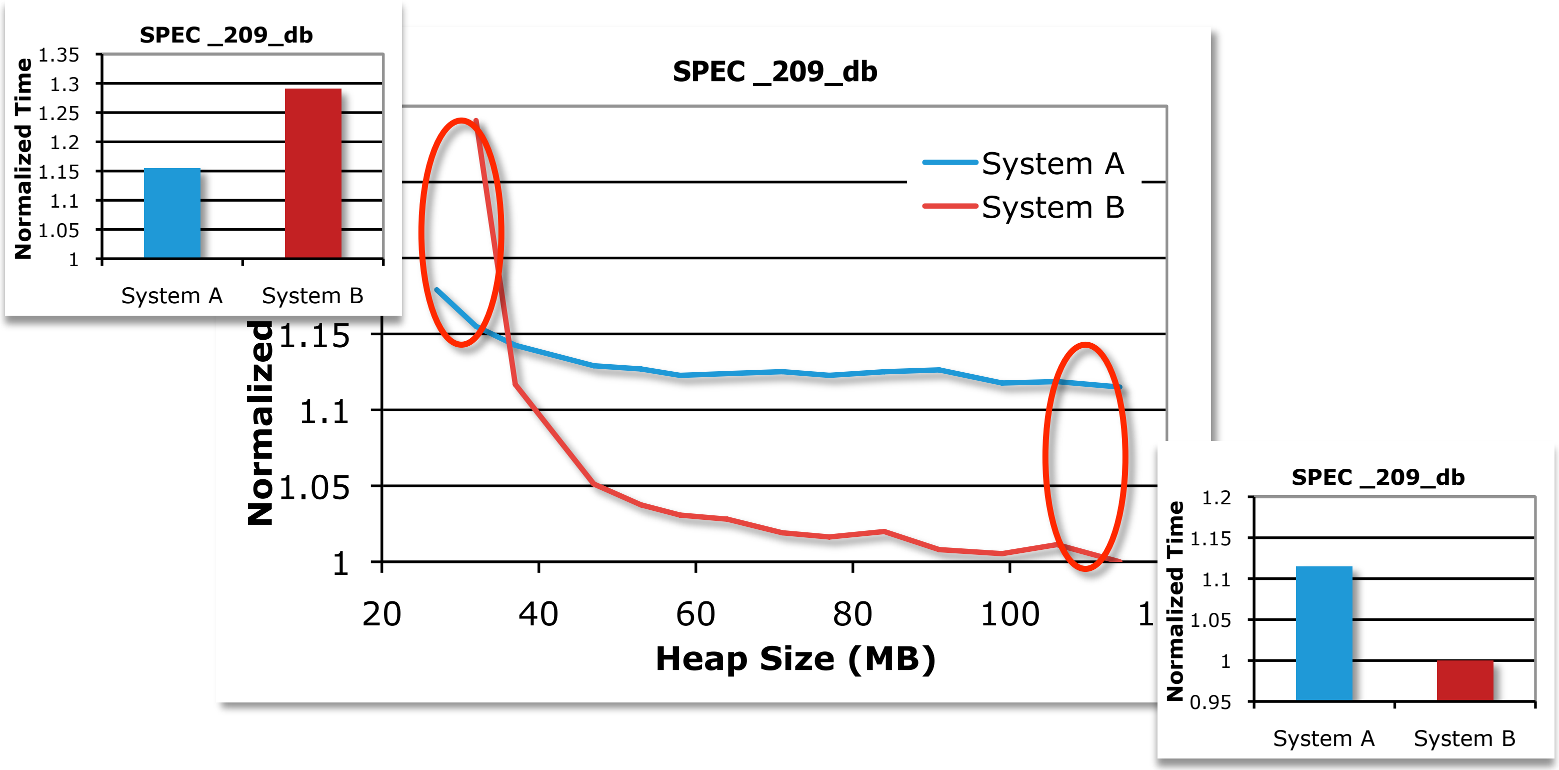
Benchmarks & Methodology



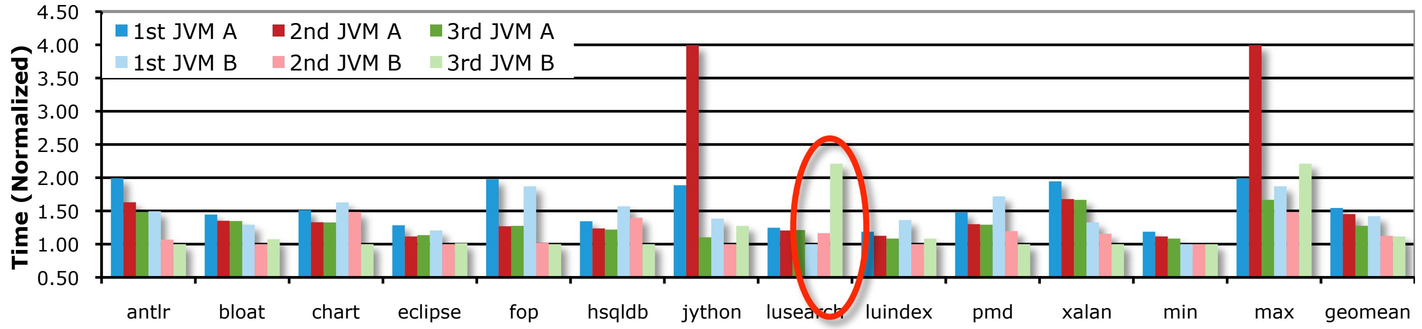


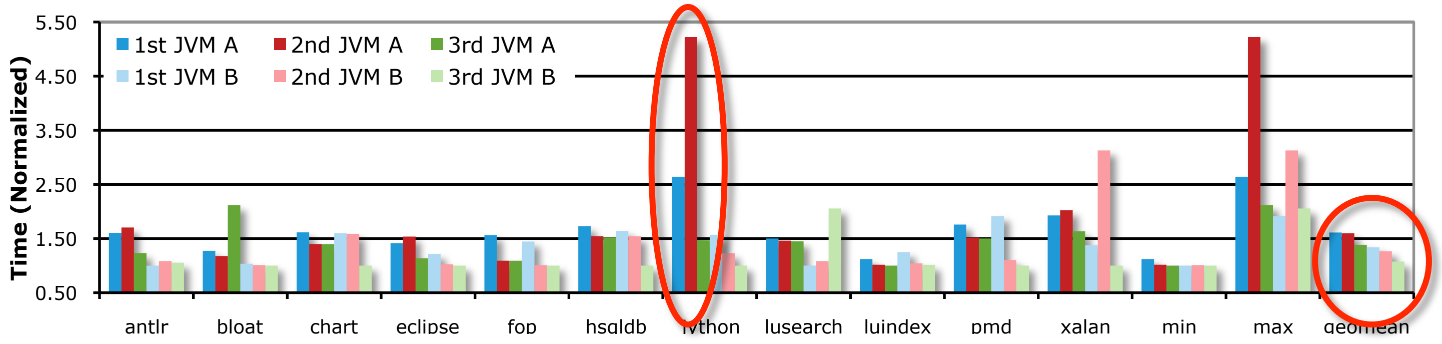
Another evaluation of the same systems, same hardware, same iteration measured….

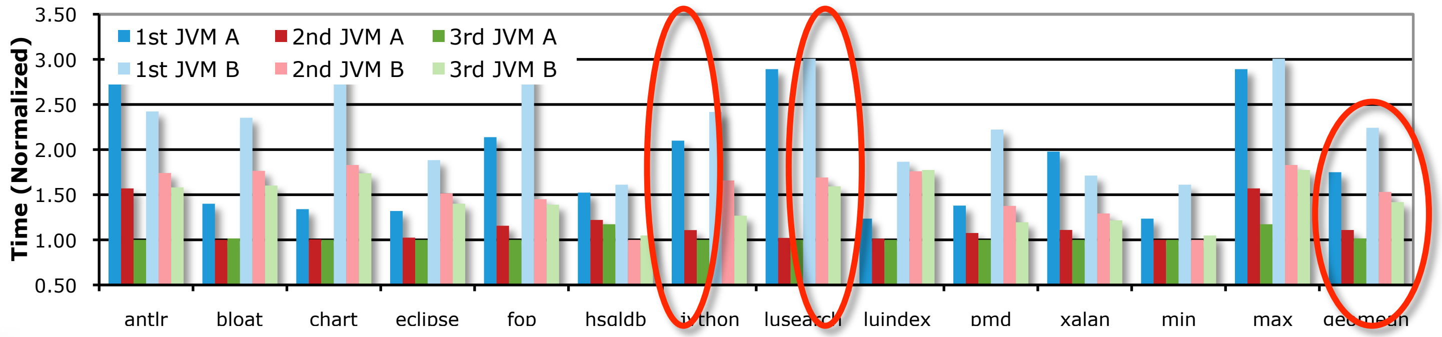
Benchmarks & Methodology



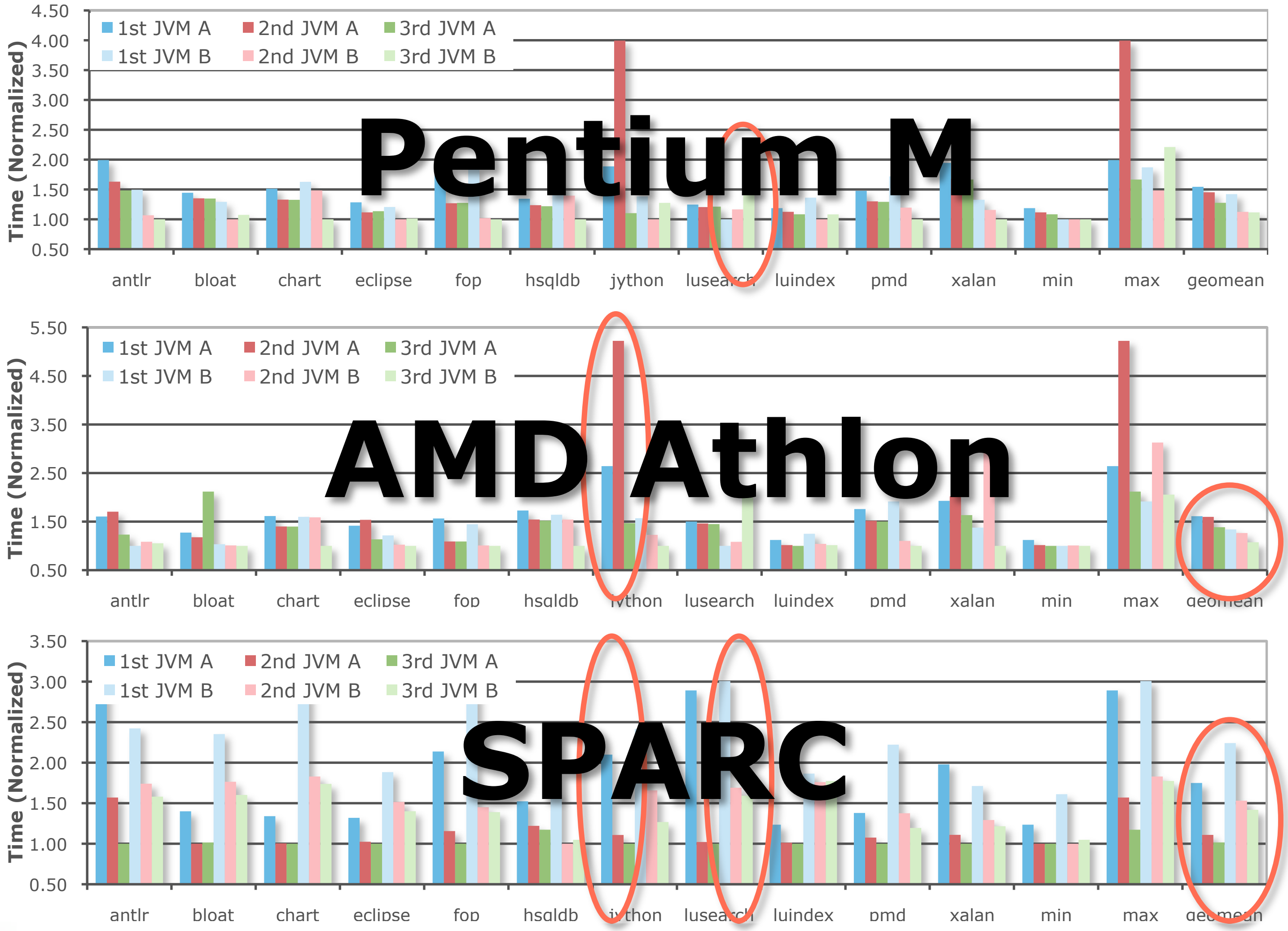
Benchmarks & Methodology







Benchmarks & Methodology



The success of most systems innovation hinges on benchmark performance.

Predicate 1. Benchmarks reflect current (and ideally, future) reality.

Predicate 2. Methodology is appropriate.

The success of most systems innovation hinges on benchmark performance.

Predicate 1. Benchmarks reflect current (and ideally, future) reality.

Predicate 2. Methodology is appropriate.

The success of most systems innovation hinges on benchmark performance.

Predicate 1. Benchmarks reflect current (and ideally, future) reality.

Predicate 2. Methodology is appropriate.

Innovation Trap

- Innovation is gated by benchmarks
- Poor benchmarking retards innovation
  - Reality: inappropriate, unrealistic benchmarks
  - Reality: poor methodology
- Concrete, contemporary instances
  - Architectural tuning to managed languages
  - Software transactional memory
  - C#
  - GC avoided in SPEC performance runs

How Did This Happen?

- Researchers depend on SPEC
  - Primary purveyor & de facto guardian
  - Industry body
  - Concerned with product comparison
    - Little involvement from researchers
  - Historically C & Fortran benchmarks
    - Did not update/adapt methodology for Java
- Researchers tend not to create their own suites
  - Enormously expensive exercise

Enough Whining. How Do We Respond?

- Critique our benchmarks & methodology
  - Not enough to "set the bar high" when reviewing!
  - Need appropriate benchmarks & methodology
- Develop new benchmarks
  - NSF review challenged us
- Maintain and evolve those benchmarks
- Establish new, appropriate methodologies
- Attack problem as a community
  - Formally (SIGs?) and ad hoc (e.g. DaCapo)

The DaCapo Suite Background & Scope

- Motivation (mid 2003)
  - We wanted to do good Java runtime and compiler research
  - An NSF review panel agreed that the existing Java benchmarks were limiting our progress
- Non-goal: product comparison (SPEC does a fine job)
- Scope
  - Client-side, real-world, measurable Java applications
    - Real world data and coding idioms, manageable dependencies
- Two-pronged effort
  - New candidate benchmarks
  - New suite of analyses to characterize candidates

The DaCapo Suite: Goals

- **Open source**
  - Encourage (& leverage) community feedback
  - Enable analysis of benchmark sources
  - Freely available, avoid intellectual property restrictions
- **Real, non-trivial applications**
  - Popular, non-contrived, active applications
  - Use analysis to ensure non-trivial, good coverage
- **Responsive, not static**
  - Adapt the suite as circumstances change
- **Easy to use**

The DaCapo Suite: Today

- **Open source (www.dacapobench.org)**
- Significant community-driven improvements already
- **11 real, non-trivial applications**
  - Compared to JVM98, JBB2000, on average:
    - 2.5 X classes, 4 X methods, 3 X DIT, 20 X LCOM, 2 X optimized methods, 5 X icache load, 8 X ITLB, 3 X running time, 10 X allocations, 2 X live size.
  - Uncovered bugs in product JVMs
- **Responsive, not static**
  - Have adapted the suite
- **Easy to use**
  - Single jar file, OS-independent, output validation

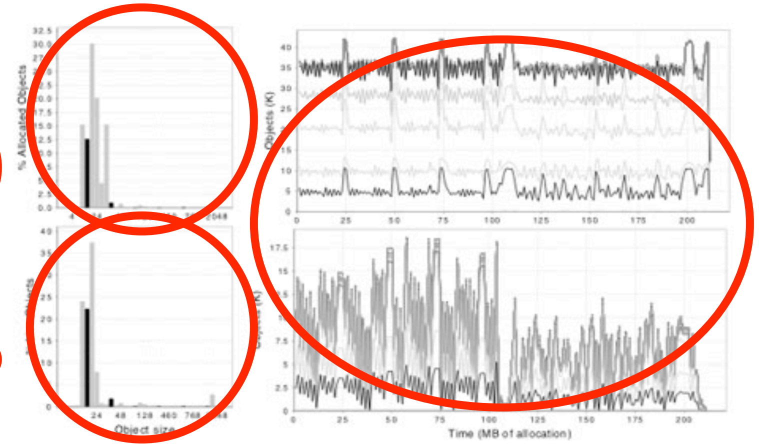
Some of our Analyses

Benchmark Description and Origin

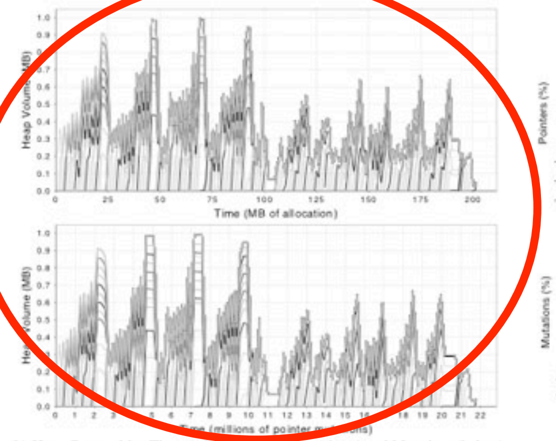
Short Description	A text indexing tool
Long Description	Indexes a set of documents, the works of Shakespeare and the King James Bible
Threads	Single threaded
Repeats	Single iteration indexes two multi-file documents
Version	1.9.1
Copyright	Copyright (C) The Apache Software Foundation
Author	Lucene Project Management Committee
License	Apache Public License

Benchmark Characteristics

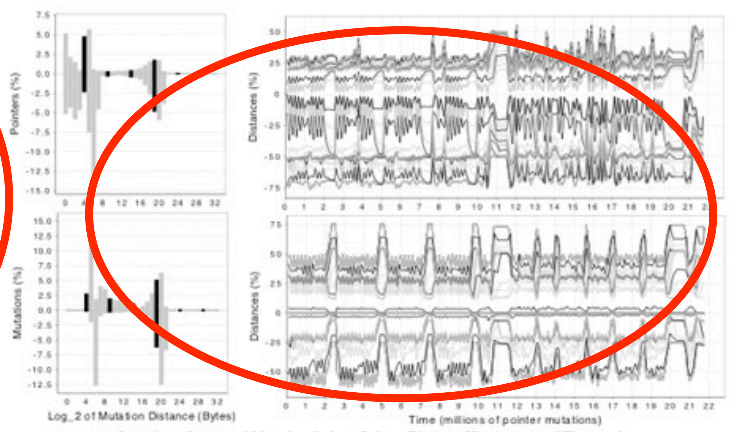
Total Allocation (MB)	201.4
(Obj)	7,202,623
Maximum Live (MB)	1.0
(Obj)	18,566
Pointer Mutations (M)	21.75
Classes Loaded	128



(a) Allocated (above) and Live (below) Object Size Histograms and Time-series



(b) Heap Composition Time-series, in Allocations (above) and Mutations (below)



(c) Snapshot (above) and Mutation (below) Pointer Distance Histograms and Time-series

Figure 11. Benchmark Characteristics: luindex

Broader Impact

- Just the tip of the iceberg?
  - *Q: How many good ideas did not see light of day because of jvm98?*
- A problem unique to Java?
  - *Q: How has the lack of C# benchmarks impacted research?*
- What's next?
  - Multicore architectures, transactional memory, Fortress, dynamic languages, …
  - *Q: Can we evaluate TM versus locking?*
  - *Q: Can we evaluate TM implementations? (SPLASH & JBB???)*
- Are we prepared to let major directions in our field unfold at the whim of inadequate methodology?

Developing a New Suite

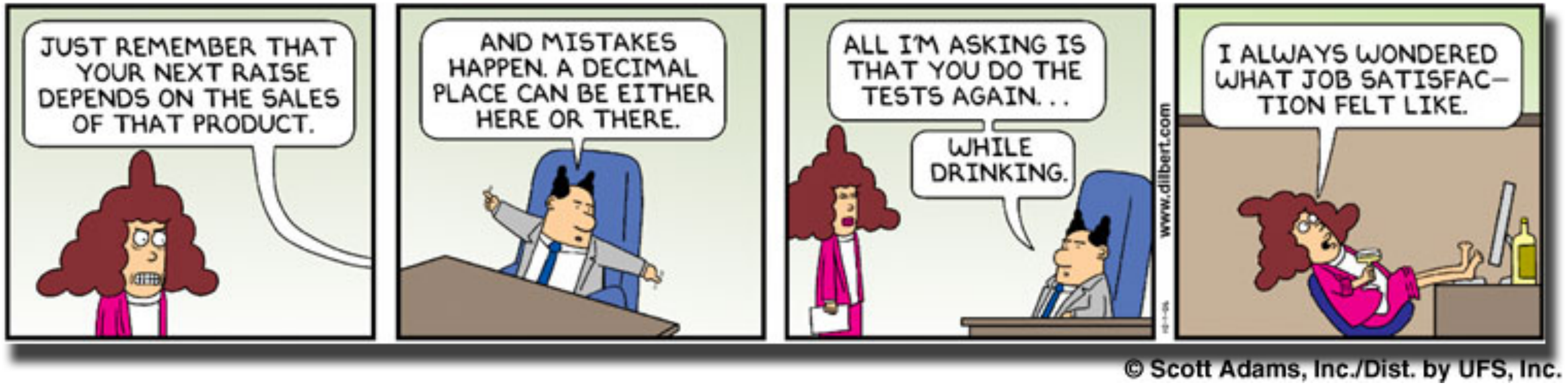
- Establish a community consortium
  - Practical and qualitative reasons
  - DaCapo grew to around 12 institutions
- Scope the project
  - What qualities do you most want to expose?
- Identify realistic candidate benchmarks
  - … and iterate.
- Identify/develop many analyses and metrics
  - This is essential
- Analyze candidates & prune set, engaging community
  - An iterative process
- Use PCA to verify coverage

Conclusions

- Systems **innovation is gated by benchmarks**
  - Benchmarks & methodology can retard or accelerate innovation, focus or misdirect energy.
- As a community, **we have failed**
  - We have unrealistic benchmarks and poor methodology
- We have **a unique opportunity**
  - Transactional memory, multicore performance, dynamic languages, etc…
- We need to **take responsibility** for benchmarks & methodology
  - Formally (eg SIGPLAN) or via ad hoc consortia (eg DaCapo)

Acknowledgements

- **Andrew Appel**, **Randy Chow**, **Frans Kaashoek** and **Bill Pugh** who encouraged this project at our three year ITR review.
- **Mark Wegman** who initiated the public availability of Jikes RVM, and **the developers of Jikes RVM**
- **Fahad Gilani** for writing the original version of the measurement infrastructure for his ANU Masters Thesis
- **Kevin Jones** and **Eric Bodden** for significant feedback and enhancements
- **Vladimir Strigun** and **Yuri Yudin** for extensive testing and feedback
- **The rest of the DaCapo research consortium** for their long term assistance and engagement with this project


JUST REMEMBER THAT YOUR NEXT RAISE DEPENDS ON THE SALES OF THAT PRODUCT.
AND MISTAKES HAPPEN. A DECIMAL PLACE CAN BE EITHER HERE OR THERE.
ALL I'M ASKING IS THAT YOU DO THE TESTS AGAIN. . .
WHILE DRINKING.
www.dilbert.com
I ALWAYS WONDERED WHAT JOB SATISFAC– TION FELT LIKE.
© Scott Adams, Inc./Dist. by UFS, Inc.

www.dacapobench.org

Extra Slides

Experimental Design

Best Practices

- Measuring **JVM** innovations
- Measuring **JIT** innovations
- Measuring **GC** innovations
- Measuring **Architecture** innovations

JVM Innovation

Best Practices

- Examples:
  - Thread scheduling
  - Performance monitoring
- Workload triggers differences
  - real workloads & perhaps micro-benchmarks
  - e.g. control frequency of thread switching
- Measure and report multiple iterations
  - start-up, steady-state (aka server-mode)
  - do not configure the VM to not optimize!!
- Use a modest, or multiple heap sizes
  - a function of minimum in which application will run
- Use & report multiple architectures

JIT Innovation

Best Practices

Example: new compiler optimization

- **Code quality:** Does it improve the application code?
- **Compile time:** How much does it add to compile time?
- **Total time:** Compiler and application time together
- **Problem:** Adaptive compilation responds to compilation load and code quality
- **Question:** how do we tease all these effects apart?

JIT Innovation Cont.

Best Practices

Teasing apart **compile time** and **code quality** requires multiple experiments.

Total Time

- Run adaptive system as intended
  - Result: mixture of optimized and unoptimized code
  - First & Nth iterations (startup and steady-state)
  - Set and report heap size as a function of minimum for the application
  - Report: mean and statistical error

Code Quality

OK: Run iterations until performance stabilizes, or

Better: Run several iterations, turn off JIT, measure iteration with no JIT activity

Best: Replay mix compilation

Compile time

- Requires the compiler to be deterministic
- *Replay* mix compilation

Replay Compilation

Force Determinism From the JIT

An adaptive compilation **profiler** and **replayer**.

Profiler

- Profile JIT on multiple multi-iteration executions; pick best or median
- Record in profile key optimization *inputs* and *outcomes* (eg dynamic edge profiles, final optimization levels)

Replayer

- Use profile to directly compile methods to final optimization level
- Use profile is input to optimization (eg edge profiles)

Result

- Controlled, deterministic, and repeatable compiler behavior
- Removes largest source of statistical variance
- Not perfect (eg inlining)

GC Innovation

Best Practices

- Requires more than one experiment…
- Explore space–time trade-off
  - Use & report a range of heap sizes
  - Express heap size relative to minimum
  - VMs should report total memory, not just application memory
    - GC may require substantial meta-data
    - JIT & VM use memory

GC Innovation Cont.

Best Practices

- Measure time for constant workload
  - Throughput experiments don't hold workload constant
- Replay: hold compiler activity constant
  - Choose best profile
    - This will minimize VM and highlight GC costs
- Ideally: evaluate with adaptive system
  - Overcome non-determinism with statistical brute force

Architecture Innovation

Best Practices

- Requires more than one experiment…
- Use more than one VM
- Include GC: set modest heap size or measure multiple heap sizes
- Include a mix of optimized and unoptimized code
- Minimize non-determinism
  - Replay
    - Good, but not available in product JVMs
  - Roll-forward from snapshot
    - For strictly microarchitectural change
  - Statistical brute force
    - Intractable given overhead of simulation