



# What's Your Workload? and Why You Care

Stephen M Blackburn, Robin Garner, Chris Hoffmann, Asjad M Khan, Kathryn S McKinley, 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

**OOPSLA--ACM Conference on Object-Oriented Programming, Systems, Languages, & Applications, Portland OR, October 2007**



“...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 statistics**

“sometimes more than twice as fast”

“our algorithm is highly efficient”

KS S

“garbage collection degrades performance by 70%” *Disraeli*

“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.

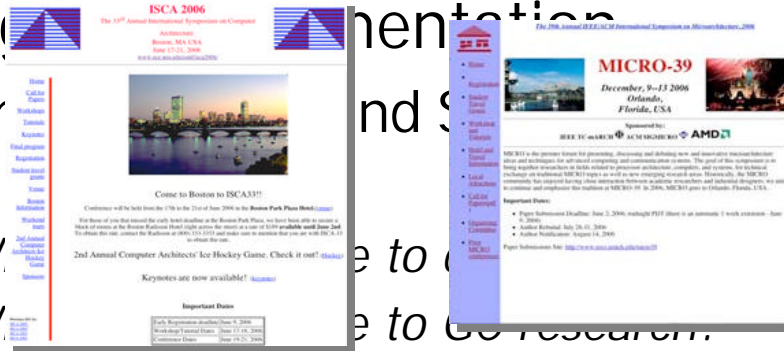
Predicate 1. Benchmarks reflect current (and ideally, future) reality.

Predicate 2. Methodology is appropriate.

## Predicate 1.

# Benchmarks & Reality

- JVM design and implementation
  - SPECjvr simple and relatively simple
    - Q: Why?
    - Q: Why?



## Computer architecture

	CK metrics		Instruction Misses/ms		Heap (MB)	
	Methods	Inheritance	L1/ms	ITLB/ms	Allocated	Live
min	152	12	34	2	0.7	0.6
max	1011	186	6356	759	271	21.1
geomean	366	40	860	56	86.5	3.8

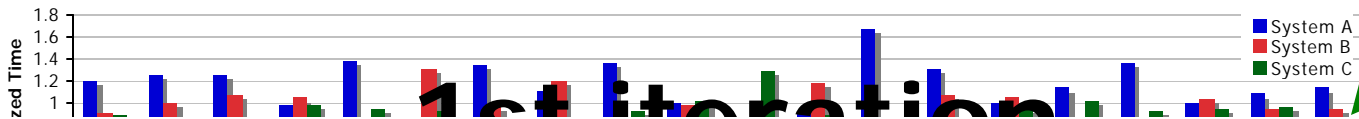
The success of most systems innovation hinges on benchmark performance.

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

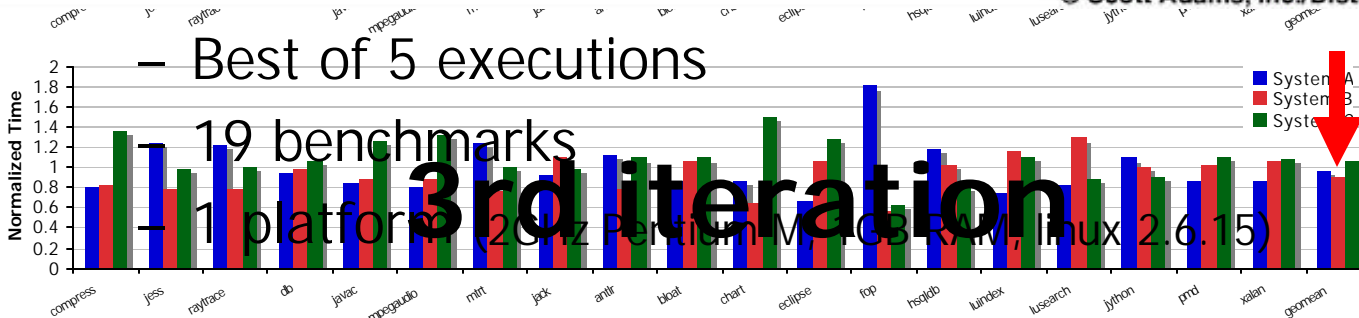
Predicate 2. Methodology is appropriate.

Predicate 2.

# Benchmarks & Methodology



© Scott Adams, Inc./Dist. by UFS, Inc.



in PSLA '(



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 & misdirects energy**
  - Reality: inappropriate, unrealistic benchmarks
  - Reality: poor methodology
- Examples
  - GC is avoided when doing SPEC performance runs
  - No architectural exploration with Java

# How Did This Happen?

- Researchers depend on SPEC
  - Primary purveyor & de facto guardian
  - Historically C & Fortran benchmarks
    - SPEC did not significantly modify methodology for Java
  - Industry body concerned with *product* comparison
    - Minimal involvement from researchers
    - SPEC is not concerned with research analysis/methodology
  - *But* the research community depends on them!
    - 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 panel challenged us
- Maintain and evolve those benchmarks
- Establish new, appropriate methodologies
- Attack problem as a community
  - Formally (SIGPLAN?) and ad hoc (eg 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 framework (see SPEC)
- 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

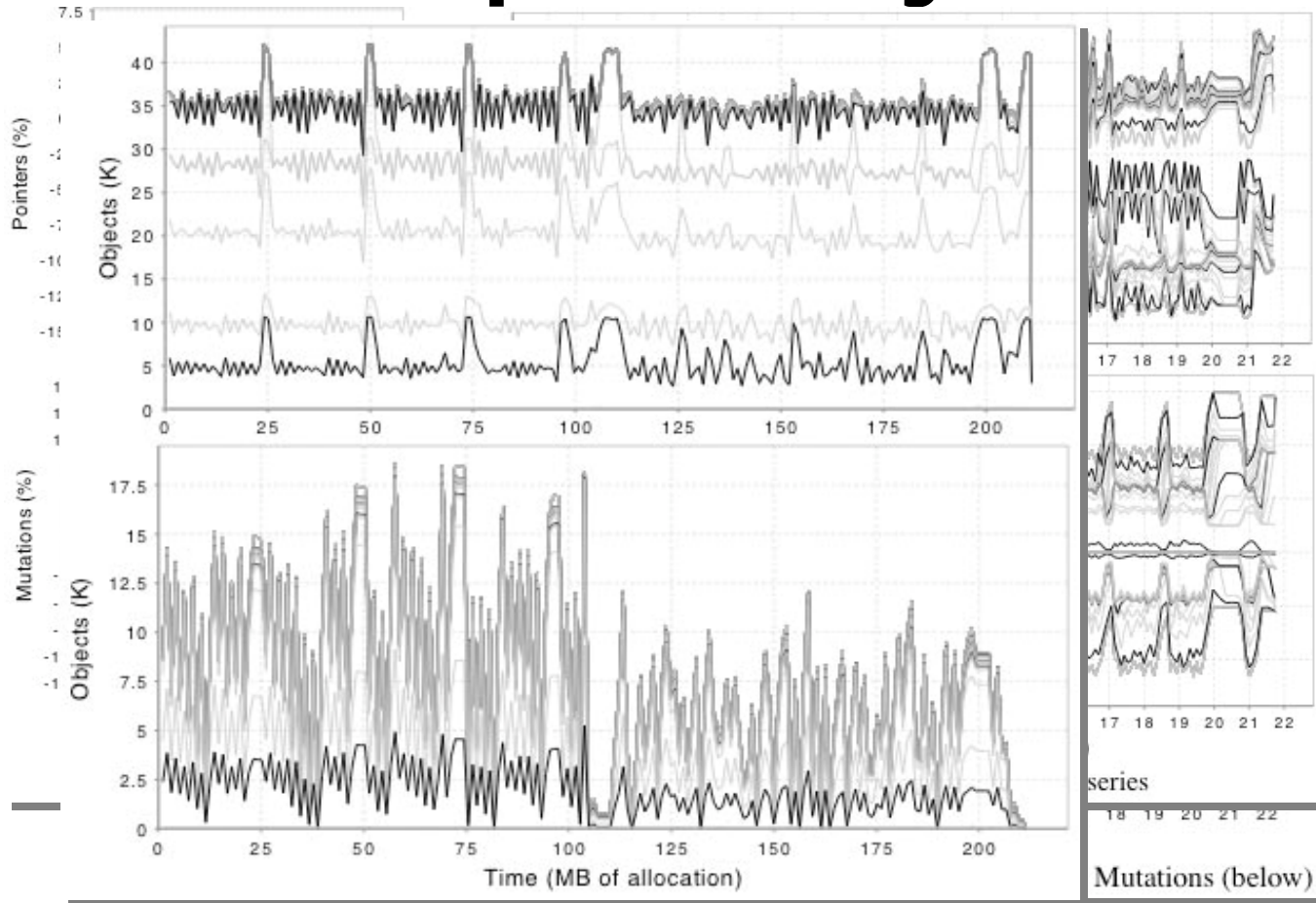
- **Open source ([www.dacapobench.org](http://www.dacapobench.org))**
  - Significant community-driven improvements already
    - *Examples: enable whole program analysis (McGill) , Xalan revision (Intel)*
- **11 real, non-trivial applications**
  - Compared to JVM98, JBB2000; on average:
    - *2.5 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
    - *Examples: addition of eclipse, lusearch, luindex and revision of Xalan*
- **Easy to use**
  - Single jar file, OS-independent, output validation



# Methodology Recommendations

- Improved methodology for **JVM**
  - Measure & report multiple iterations
  - Use & report multiple arch. when measuring JVM
  - Use & report multiple JVMs when measuring arch.
- Improved methodology for **JIT**
  - Determinism is crucial to some analyses (use “replay”)
- Improved methodology for **GC**
  - Use & report a range of fixed heap sizes
  - Hold workload (cf time) constant
  - Hold compiler activity constant (use “replay”)

# Example Analyses



# 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)



# Acknowledgments

- **Andrew Appel, Randy Chow, Frans Kaashoek and Bill Pugh** who encouraged this project at our three year ITR review.
- **Intel and IBM** for their participation in this project
- The **US National Science Foundation (NSF) and Australian Research Council (ARC)** for funding this work
- **Mark Wegman** who initiated the public availability of Jikes RVM, and the developers of Jikes RVM
- **Fahad Gilani** for writing the original version of our measurement infrastructure for his ANU Masters Thesis
- **Kevin Jones and Eric Boddien** for significant feedback and enhancements
- **Vladimir Strigun and Yuri Yudin** for extensive testing and feedback
- **The entire DaCapo research consortium** for their long term assistance and engagement with this project



# Extra Slides



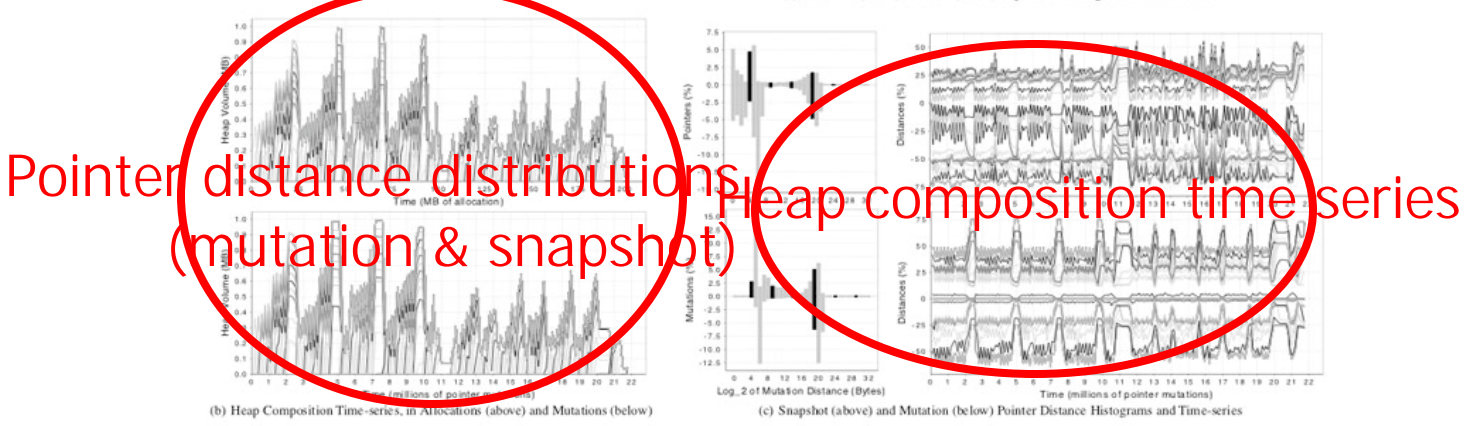


© Scott Adams, Inc./Dist. by UFS, Inc.

# Example Analyses



Object size distribution  
 Benchmark size distribution  
 time series (alloc & live)  
 Allocated object size distribution  
 Live object size distribution  
 Vital statistics distribution



Pointer distance distributions (mutation & snapshot)  
 Heap composition time series

Figure 11. Benchmark Characteristics: luindex