Quality Assurance in Performance: Evaluating Mono Benchmark Results

Tomas Kalibera, **Lubomir Bulej**, Petr Tuma

DISTRIBUTED SYSTEMS RESEARCH GROUP
http://nenya.ms.mff.cuni.cz

CHARLES UNIVERSITY PRAGUE
Faculty of Mathematics and Physics

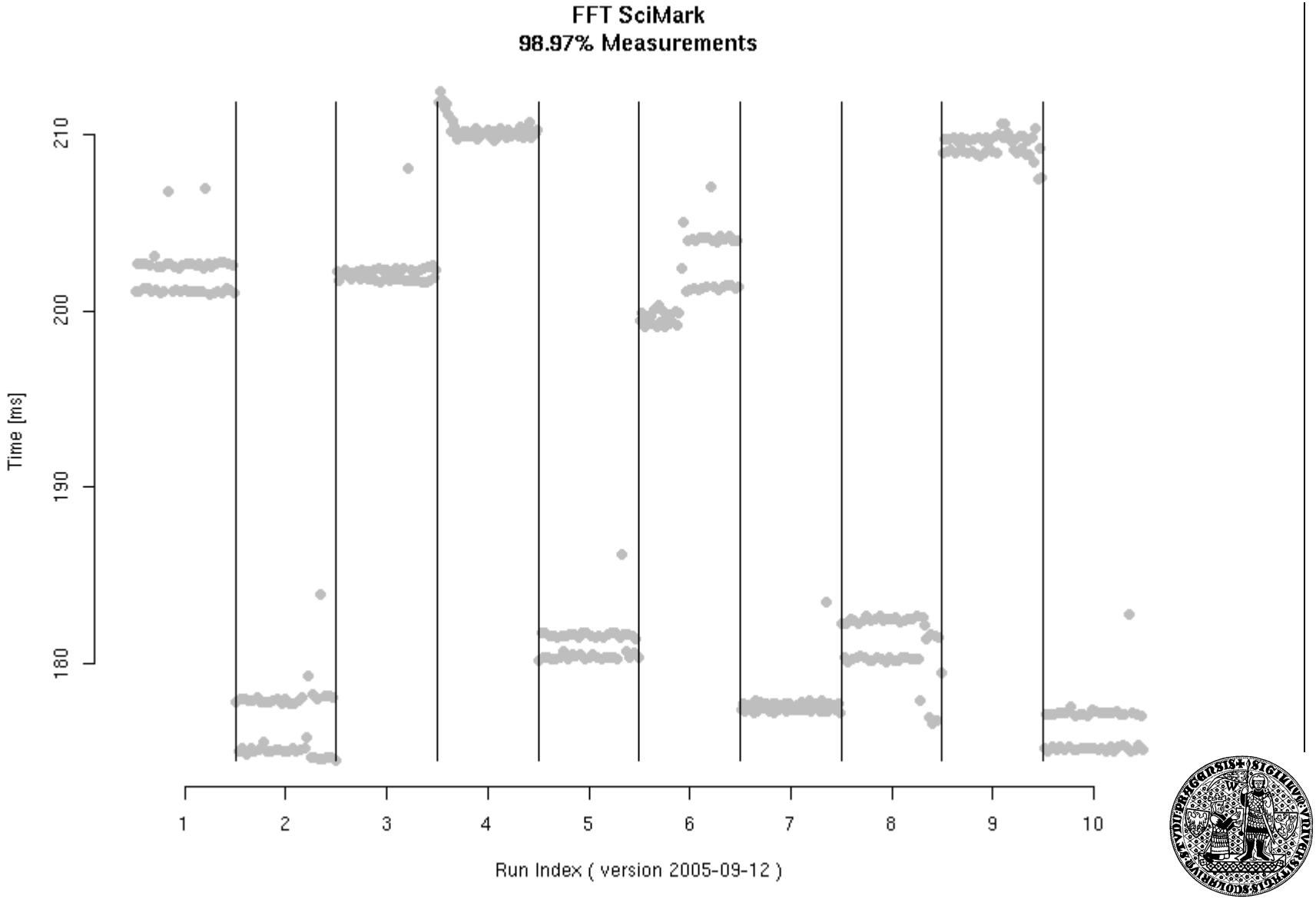
Agenda

- Regression benchmarking
  - Motivation, basic idea, requirements
  - Expectations and surprises
  - Statistical evaluation
- Application to Mono project
  - Selected benchmarks and results
  - Tracing changes back to code
  - Identified and verified regressions
- Conclusion
  - Evaluation of the approach
  - Future work

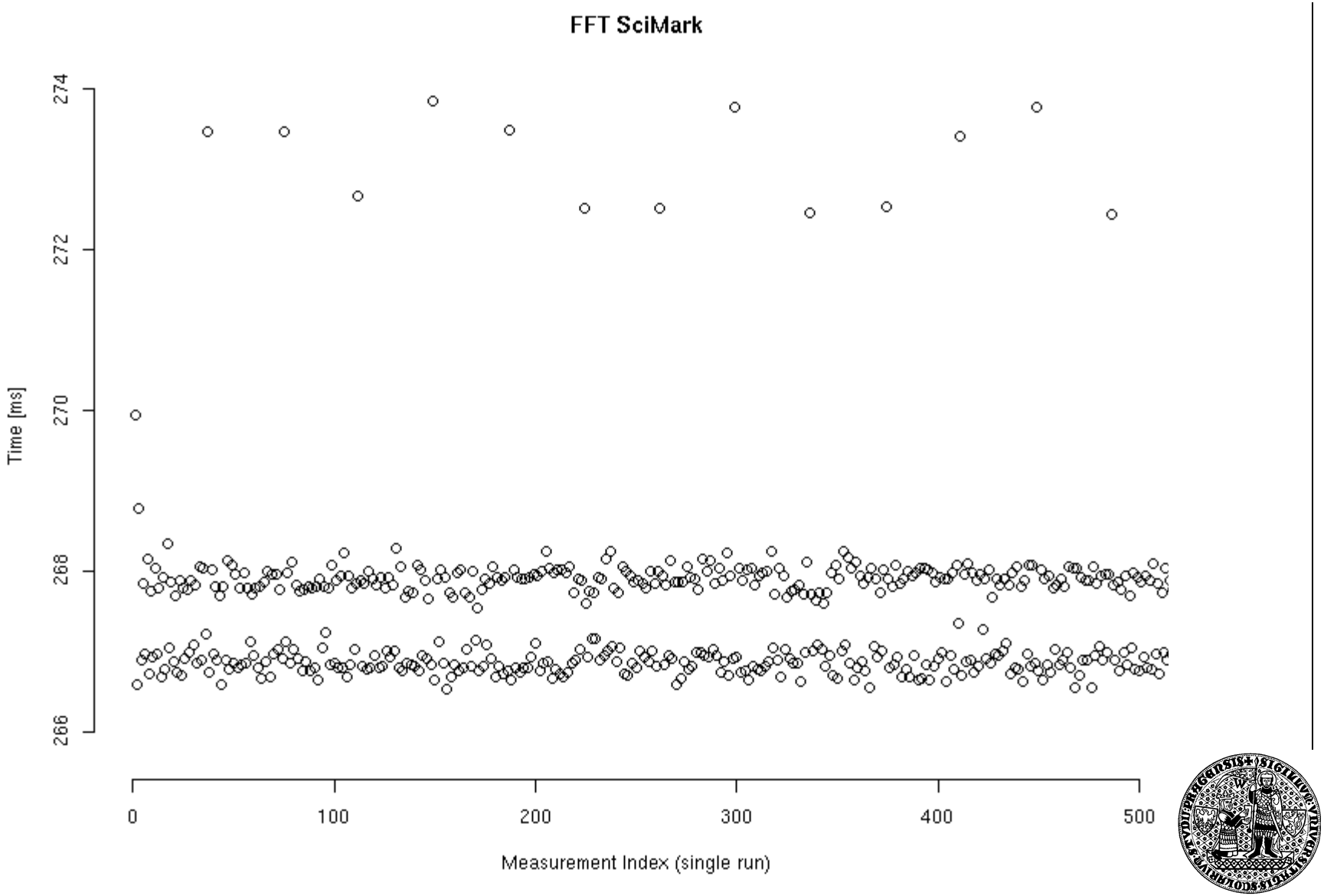
Performance: A Neglected Aspect of Quality.

- Motivation
  - Functional regression/unit testing
  - Nonfunctional/performance testing neglected
- The goal: regression benchmarking
  - Regularly test software performance
  - Detect and report performance changes
- Basic idea
  - Benchmark daily development versions
  - Detect changes in benchmark results
- Requirements
  - Fully automatic
  - Reliable and easy to use

Surprise: Repeating operations does not help.



Expectation: Repeating operations helps.



Even Worse: The instability has layers.

- Download a new software version
- Build a benchmark with the new version
- Run a benchmark **m** times
  - Start a new operating system process
  - Warm-up the benchmark
  - Invoke the same operation **n** times
  - Report individual operation response times
- Collect and analyze the results

Solution to Instability: Statistics.

- Model benchmark as a random process
  - Model instability by randomness
  - Model layers of instability by hierarchical random variables
- Collect representative data
  - Repeat builds, runs and operations
  - Benchmark result is estimate of a model parameter of interest (i.e. overall mean)
  - Result precision – precision of the estimate

Statistical Evaluation: Current solution.

- Statistical model
  - Two-layer hierarchical, robust
  - Parameter of interest is the mean, estimated by average, precision is confidence interval length
  - Allows to specify optimum number of operations for maximum precision
- Change detection
  - Non-overlapping confidence intervals

Mono Benchmarking: Proof of Concept.

- Mono Project
  - Open-source .NET platform by Novell, http://www.mono-project.com
  - Includes C# compiler, virtual machine, application libraries
- Mono Benchmarking Project
  - Fully automated benchmarking of Mono with detection of performance changes
  - Daily updated results since August 2004, http://nenya.ms.mff.cuni.cz/projects/mono

Mono Benchmarks.

- FFT SciMark
  - Uses floating point operations, memory
  - Measures FFT computation time
- Rijndael
  - Uses .NET Cryptography
  - Measures Rijndael encryption/decryption time
- TCP Ping and HTTP Ping
  - Use .Net Remoting
  - Measure single remote method invocation

HTTP Ping: Detected performance changes.

HTTP Ping: Detected performance changes.

Newer Version	Older Version	Change Impact[%]
2004-08-17	2004-08-13	-9.67 %
2004-08-18	2004-08-17	-10.44 %
2004-12-20	2004-12-01	19.64 %
2005-03-02	2005-02-28	-7.81 %
2005-03-07	2005-03-04	7.77 %
2005-04-05	2005-04-04	39.29 %
2005-05-03	2005-04-12	-47.47 %

Mono: Finding causes of performance changes.

- Manual inspection
  - Focus on modified source files, change logs
- Modifications in application libraries
  - Focus on source files used by the benchmark code (automated restricted diffs)
  - If it does not help, look into VM or compiler
- Verification
  - Create intermediate versions (1-2)
  - Benchmark and detect changes with new versions

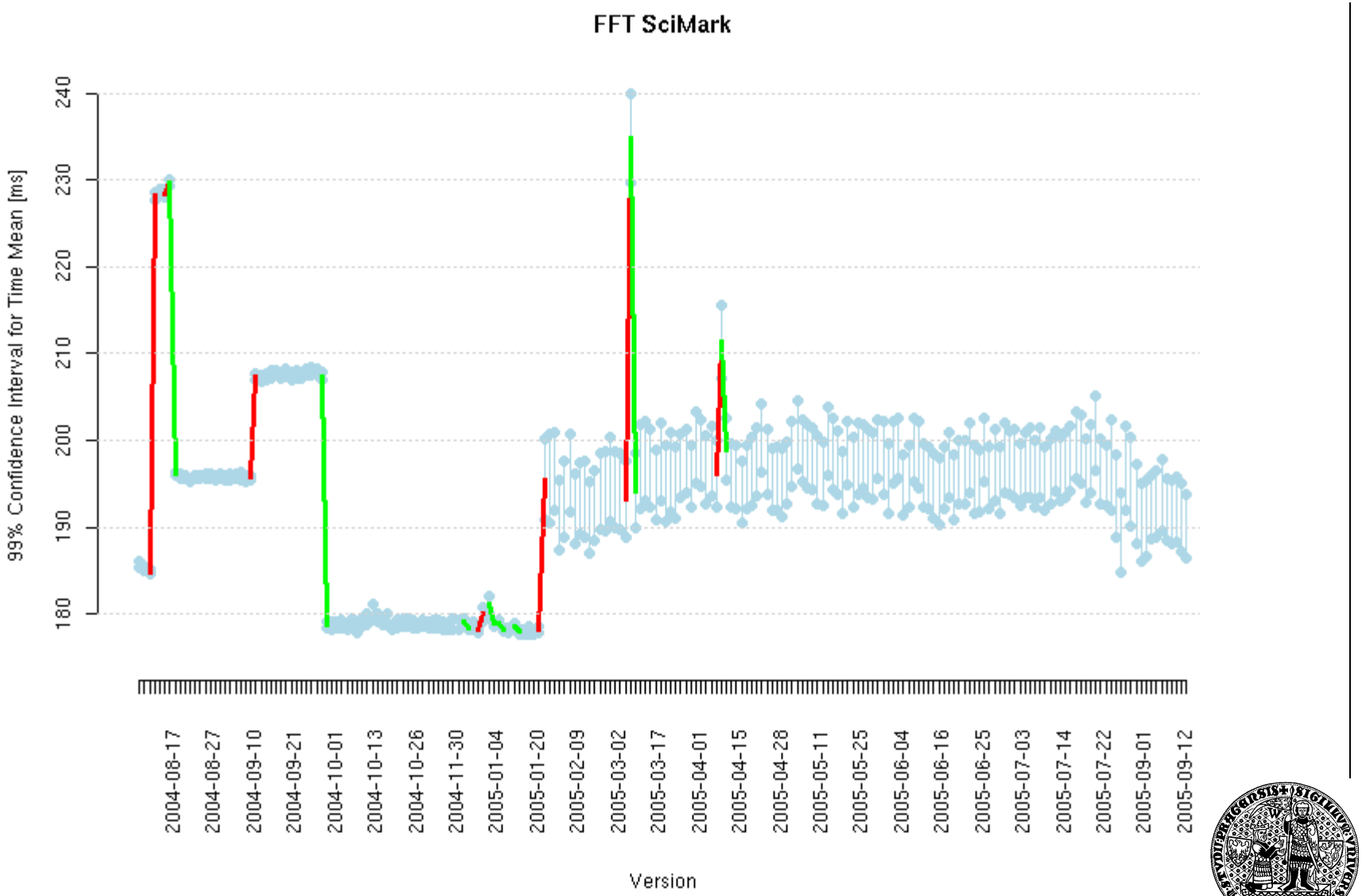
Mono: Verified causes of performance changes.

- Performance improvements
  - 99% - buffering network communication, TCP Ping
  - 17% - improved switching between native and managed code, FFT SciMark
- Performance degradations
  - 40% - introducing i18n in string case conversion, HTTP Ping
  - 24% - introducing loop optimization in JIT into default options, FFT SciMark

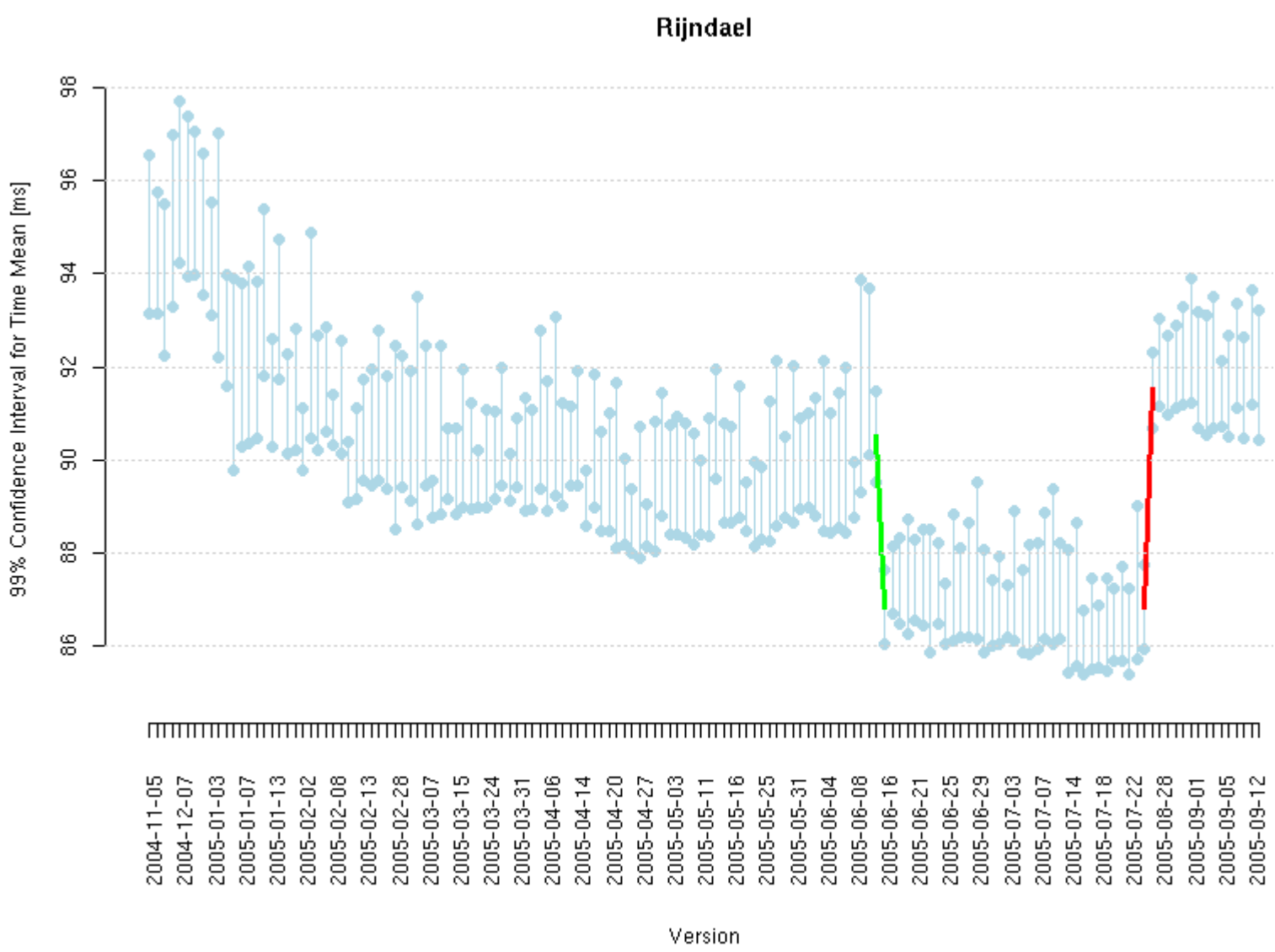
Conclusion.

- Mono benchmarking suite
  - Fully automated benchmarking with detection of changes, publicly available results
- Automated analysis
  - Independent on Mono, robust, allows planning of experiments
- Future Work
  - Even more robust analysis method
  - Semi-automated tools for discovering causes of performance changes

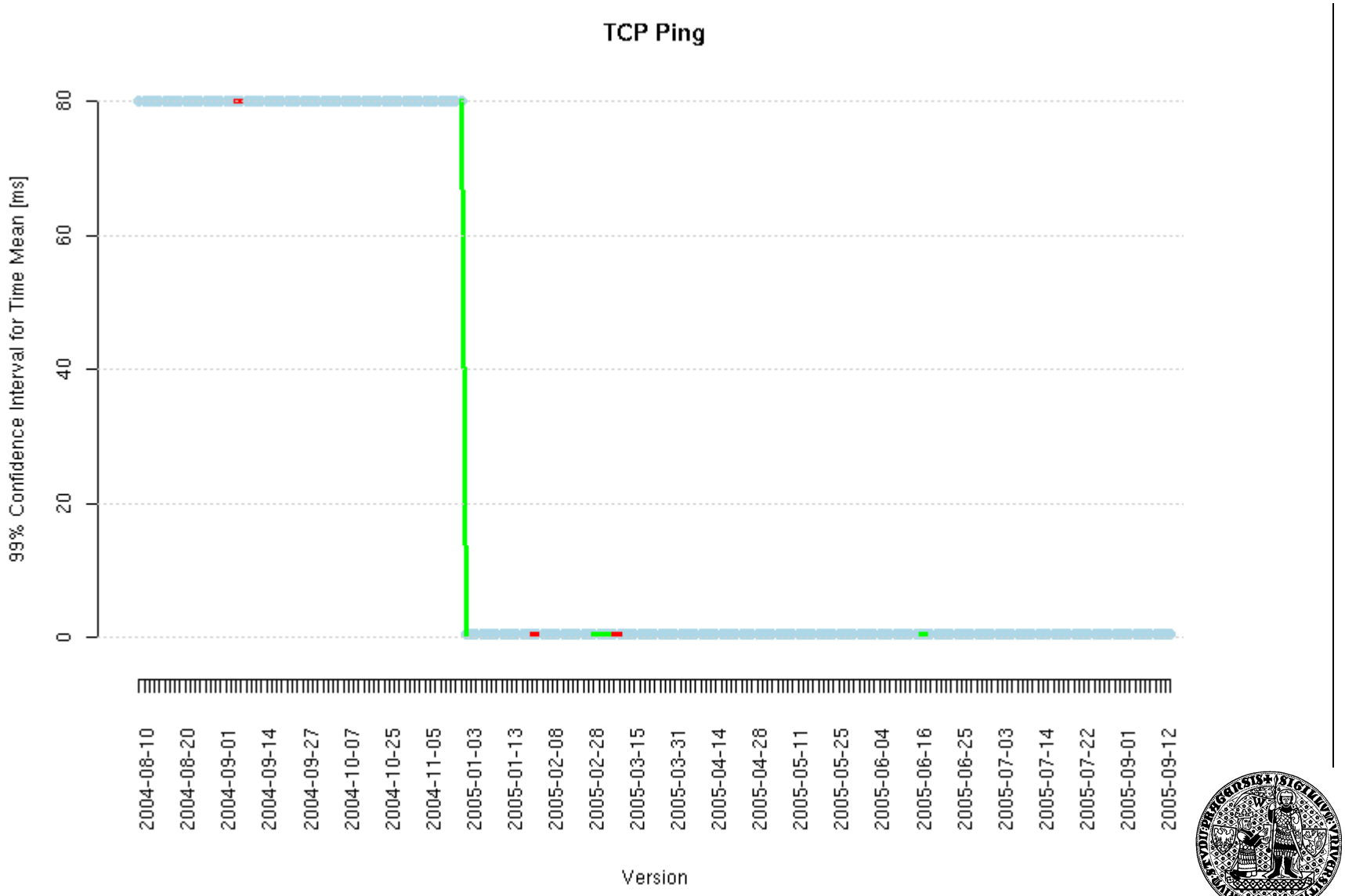
FFT SciMark: Detected performance changes.



Rijndael: Detected performance changes.



TCP Ping: Detected performance changes.



Impact of process initialization random effects.

Benchmark	Platform	Impact Factor
FFT	Pentium/Windows	94.74
FFT	Itanium/Linux	35.91
FFT	Pentium/Linux	25.81
FFT	Pentium/DOS	1.06
RPC Marshaling	Pentium/Linux	2.61
RPC Ping	Pentium/Linux	1.10
RUBiS	Pentium/Linux	1.01

Publications.

- Kalibera, T., Bulej, L., Tůma, P.: *Quality Assurance in Performance: Evaluating Mono Benchmark Results*, accepted as a full paper on Second International Workshop on Software Quality (SOQUA 2005), Erfurt, Germany
- Kalibera, T., Bulej, L., Tůma, P.: *Benchmark Precision and Random Initial State*, in Proceedings of the 2005 International Symposium on Performance Evaluation of Computer and Telecommunications Systems (SPECTS 2005), SCS 2005
- Bulej, L., Kalibera, T., Tůma, P.: *Repeated Results Analysis for Middleware Regression Benchmarking*, Performance Evaluation: An International Journal, Performance Modeling and Evaluation of High-Performance Parallel and Distributed Systems, Elsevier, 2005
- Bulej, L., Kalibera, T., Tůma, P.: *Regression Benchmarking with Simple Middleware Benchmarks*, proceedings of IPCCC 2004 Middleware Performance Workshop, IEEE 2004
- Kalibera, T., Bulej, L., Tůma, P.: *Generic Environment for Full Automation of Benchmarking*, in proceedings of First International Workshop on Software Quality (SOQUA 2004), LNI 2004