Performance Benchmarking of Application Monitoring Frameworks

PhD Thesis Defense

Kiel University, Software Engineering Group

Jan Waller — December 12, 2014

MooBench

Kieker

Motivation (Monitoring & Overhead)

Measure Everything

> *At Facebook we collect an **enormous amount** of […] **application level** statistics […] the really interesting things only show up **in production**.*
>
> —Robert Johnson, "Scaling Facebook to 500 Million Users and Beyond"

Measurement influences the performance

→

Necessary trade-off [Reimer 2013]

- Detailed monitoring
- Monitoring overhead

Manage Overhead

- High overhead is common challenge [Plattner and Nievergelt 1981, Jeffery 1996, Shao et al. 2010]
- Customers expect minimal overhead [Siegl and Bouillet 2011]
- Especially important for frameworks [Bloch 2009, Kanstrén et al. 2011]

Further reading: Chap. 1, 2, 4 and [Smith and William 2001, Woodside et al. 2007, Jones 2010, van Hoorn et al. 2012, Eichelberger and Schmid 2014]

Research Questions & Methods

Christian-Albrechts-Universität zu Kiel

What is the **performance influence** an application-level **monitoring framework** has on the **monitored system**?

What are the **causes for observed changes** in the **response time** of a monitored method?

lab experiments

How to **develop a benchmark**?

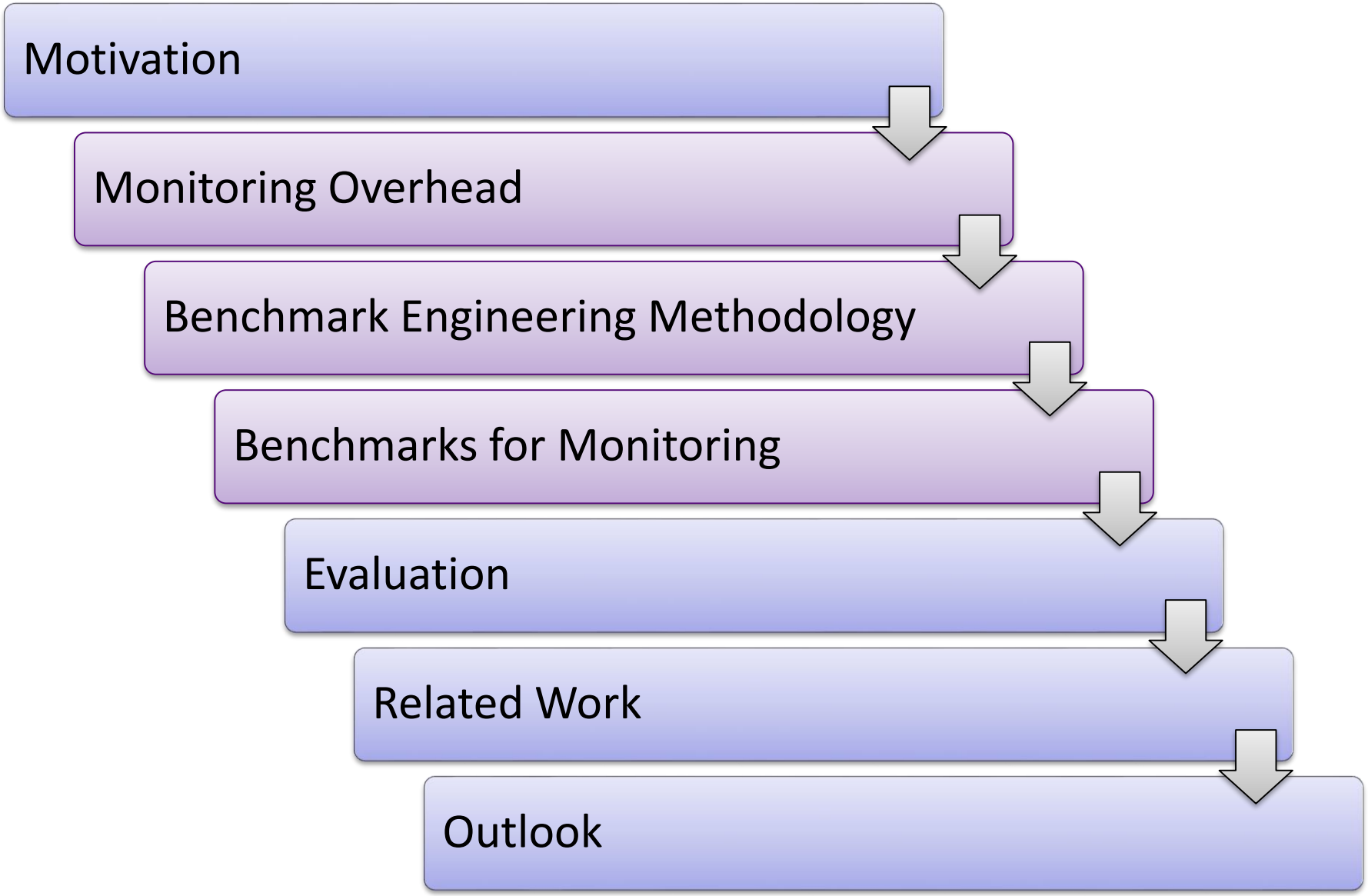
literature review
proof-of-concept

How to **measure monitoring overhead**?

lab experiments
goal, question, metric

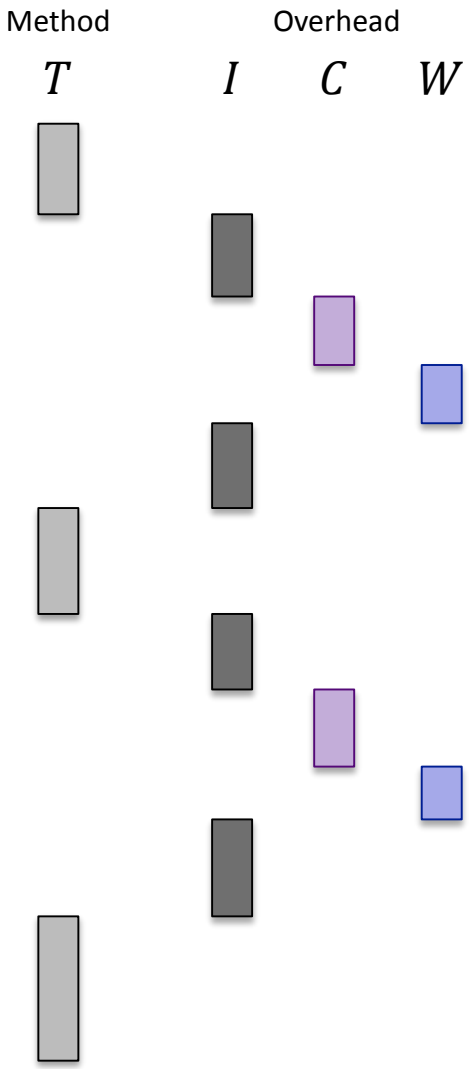
Further reading: Chap. 5 and [Waller 2013]

Outline


Motivation
Monitoring Overhead
Benchmark Engineering Methodology
Benchmarks for Monitoring
Evaluation
Related Work
Outlook

Approach & Contributions

Causes of Monitoring Overhead



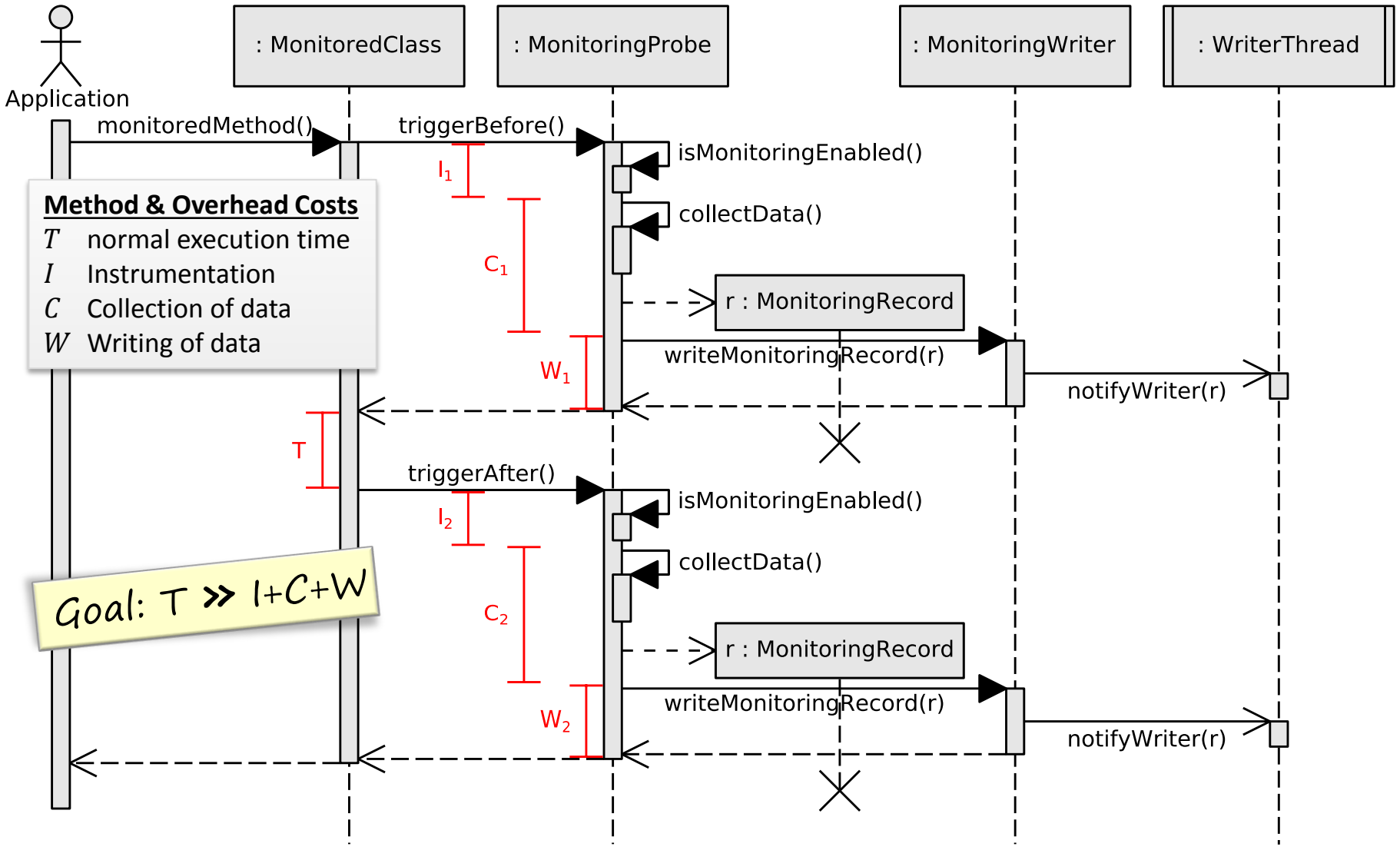

```
public boolean method() {

  if (isMonitoringEnabled(…)) {
    r = collectDataBefore();
    writeMonitoringData(r);
  }

  retval = businessMethod();

  if (isMonitoringEnabled(…)) {
    r = collectDataAfter();
    writeMonitoringData(r);
  }

  return retval;
}
```

Method & Overhead Costs

T	normal execution time
I	Instrumentation
C	Collection of data
W	Writing of data

Further reading: Chap. 6 and [van Hoorn et al. 2009, Waller and Hasselbring 2012, Waller and Hasselbring 2013, Waller et al. 2014]

Monitoring Overhead (cont.)

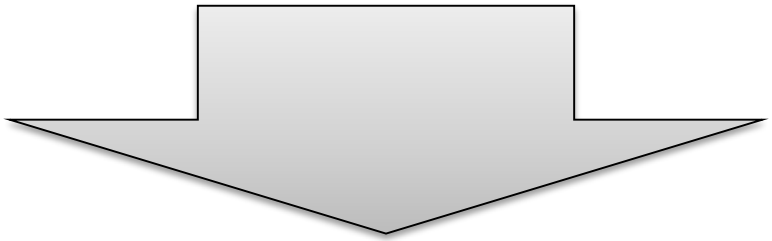


Further reading: Chap. 6 and [van Hoorn et al. 2009, Waller and Hasselbring 2012, Waller and Hasselbring 2013, Waller et al. 2014]

Benchmark Engineering

There is **no established methodology** for benchmarks

- [Hinnant 1988, Price 1989, Sachs 2011]



Benchmark Engineering Methodology in three phases:

including a total of 18 different **requirements** and **guidelines**

Further reading: Chap. 7 and [Waller 2013, Waller and Hasselbring 2013, Waller et al. 2014]

Benchmark Engineering (cont.)

Requirements & Guidelines	1965 – 2003	2004 – 2014	∑ (49)
R1: Representative / Relevant	21	21	42
R2: Repeatable	9	16	25
R3: Robust	10	18	28
R4: Fair	4	7	11
R5: Simple	10	13	23
R6: Scalable	4	8	12
R7: Comprehensive	10	9	19
R8: Portable / Configurable	8	9	17
S1: Specific	6	2	8
S2: Accessible / Affordable	2	4	6

Further reading: Chap. 7 and [Waller 2013, Waller and Hasselbring 2013, Waller et al. 2014]

Benchmark Engineering (cont.)

Requirements & Guidelines	1988 – 2003	2004 – 2014	∑ (31)
R9: Robust Execution	8	12	20
R10: Repeated Executions	3	12	15
R11: Warm-up / Steady State	2	14	16
R12 Idle Environment	2	4	6

Requirements & Guidelines	1987 – 2003	2004 – 2014	∑ (31)
R13: Statistical Analysis	7	12	19
R14: Reporting	6	16	22
R15: Validation	2	5	7
S3: Public Results Database	3	3	6

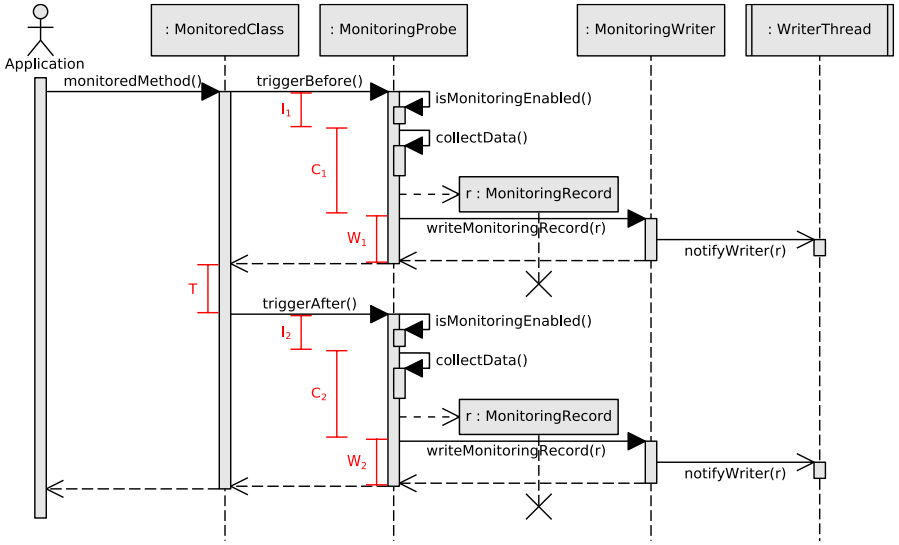
Further reading: Chap. 7 and [Waller 2013, Waller and Hasselbring 2013, Waller et al. 2014]

Benchmarks for Overhead

Three Portions of Overhead

Method & Overhead Costs

T	normal execution time
I	Instrumentation
C	Collection of data
W	Writing of data



Determine each portion (one at a time):

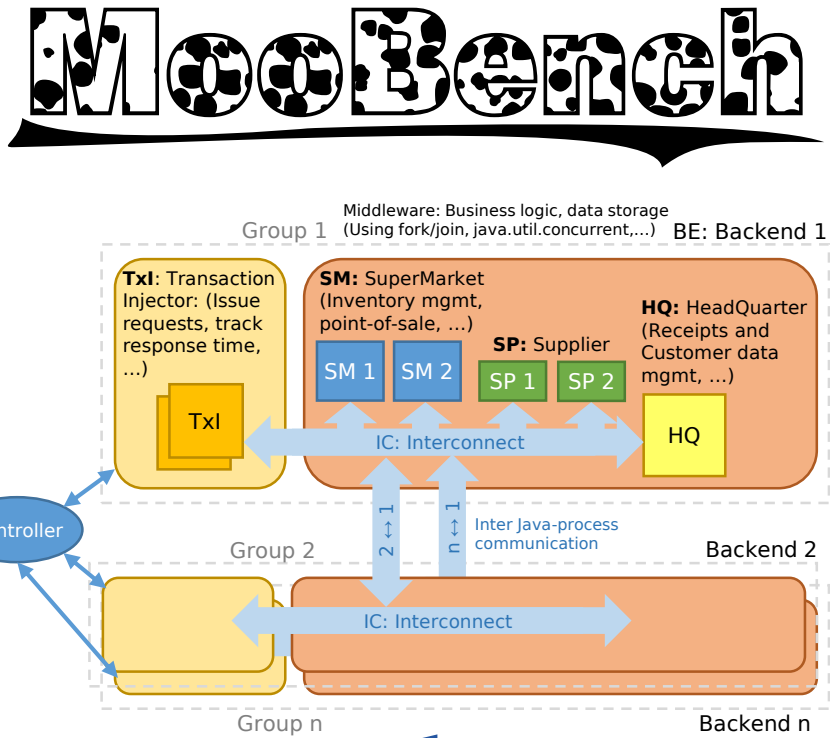
1. Determine T in the benchmark system — T
2. Add instrumentation I — $T + I$
3. Add data collection C — $T + I + C$
4. Add writing W — $T + I + C + W$

Further reading: Chap. 8 and [Waller and Hasselbring 2012, Waller 2013, Waller and Hasselbring 2013, Waller et al. 2014]

Benchmarks for Monitoring

Three evaluation steps

1. Micro-Benchmarks
   - **MooBench**
2. Macro-Benchmarks
   - Pet Store
   - SPECjvm2008
   - **SPECjbb2013**
3. Meta-Monitoring
   - **Kicker** for Kieker



Further reading: Chap. 8, 9, 10 and [Waller 2013]

MooBench (Monitoring overhead Benchmark)

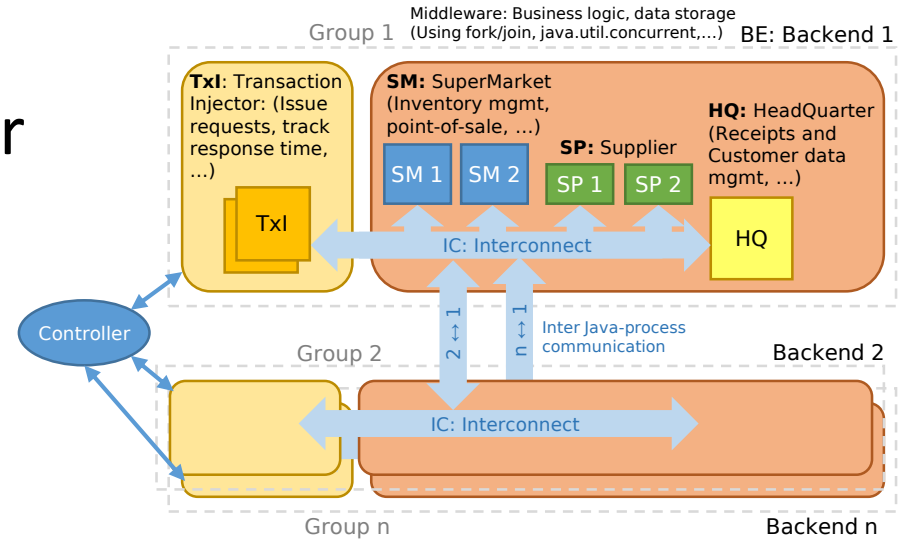
- Measures the **three causes of overhead**
- *Monitored Application*
  - single class; single method; fixed timing; configurable
- *Benchmark Driver*
  - initializes; executes; collects; records
- *Designed/implemented*, *executed*, and *analyzed/presented* according to our **benchmark engineering methodology**

Further reading: Chap. 8 and [Waller and Hasselbring 2012, Waller 2013, Waller and Hasselbring 2013, Waller et al. 2014]

SPECjbb2013 [Pogue et al. 2014]

Evaluate ***performance & scalability*** of ***environments*** for ***Java business applications***

- World-wide supermarket company IT infrastructure



spec

http://spec.org/
http://research.spec.org/

Further reading: Chap. 8 and [Waller 2013]

Meta-Monitoring

Monitoring the Monitoring Framework

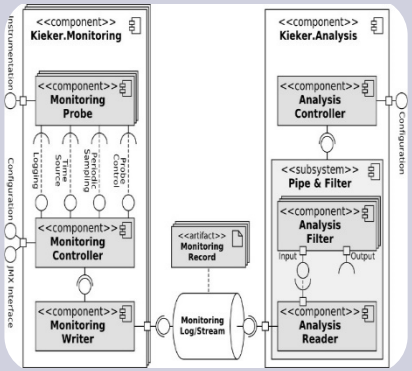
- based upon Kieker 1.10
- **Kicker** available as tagged version in git

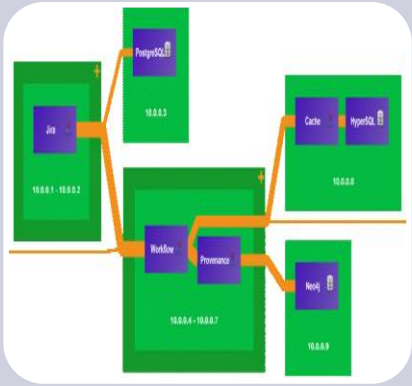
Challenges

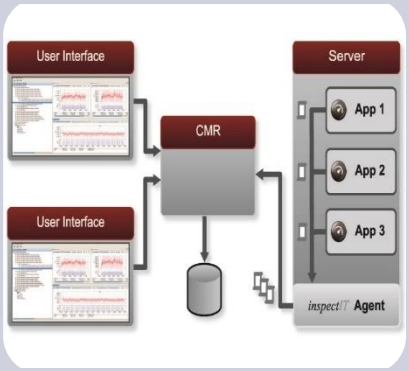
- Monitoring the monitoring
  - prevent endless loops
- Minimize perturbation
  - aka meta-monitoring overhead

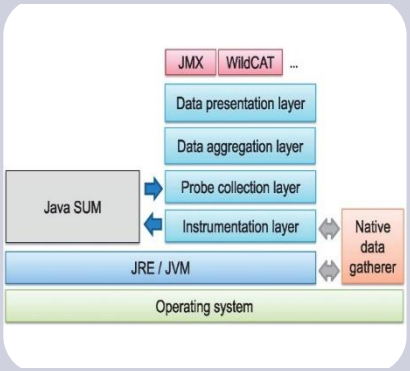
Further reading: Chap. 10 and [Waller 2013]

Systems Under Test (SUTs)









Kieker

http://kieker-monitoring.net

- Monitoring framework
- Research project
  - Oldenburg Univ.
  - Kiel University
  - Univ. of Stuttgart
- Focus on traces

ExplorViz

http://explorviz.net

- Monitoring tool
- Research project
  - Kiel University
- Focus on performance under high load

inspectIT

http://inspectit.eu

- Monitoring tool
- Commercial tool
  - NovaTec GmbH
- Focus on APM
- Integrated analysis

SPASS-meter

http://ssehub.github.com

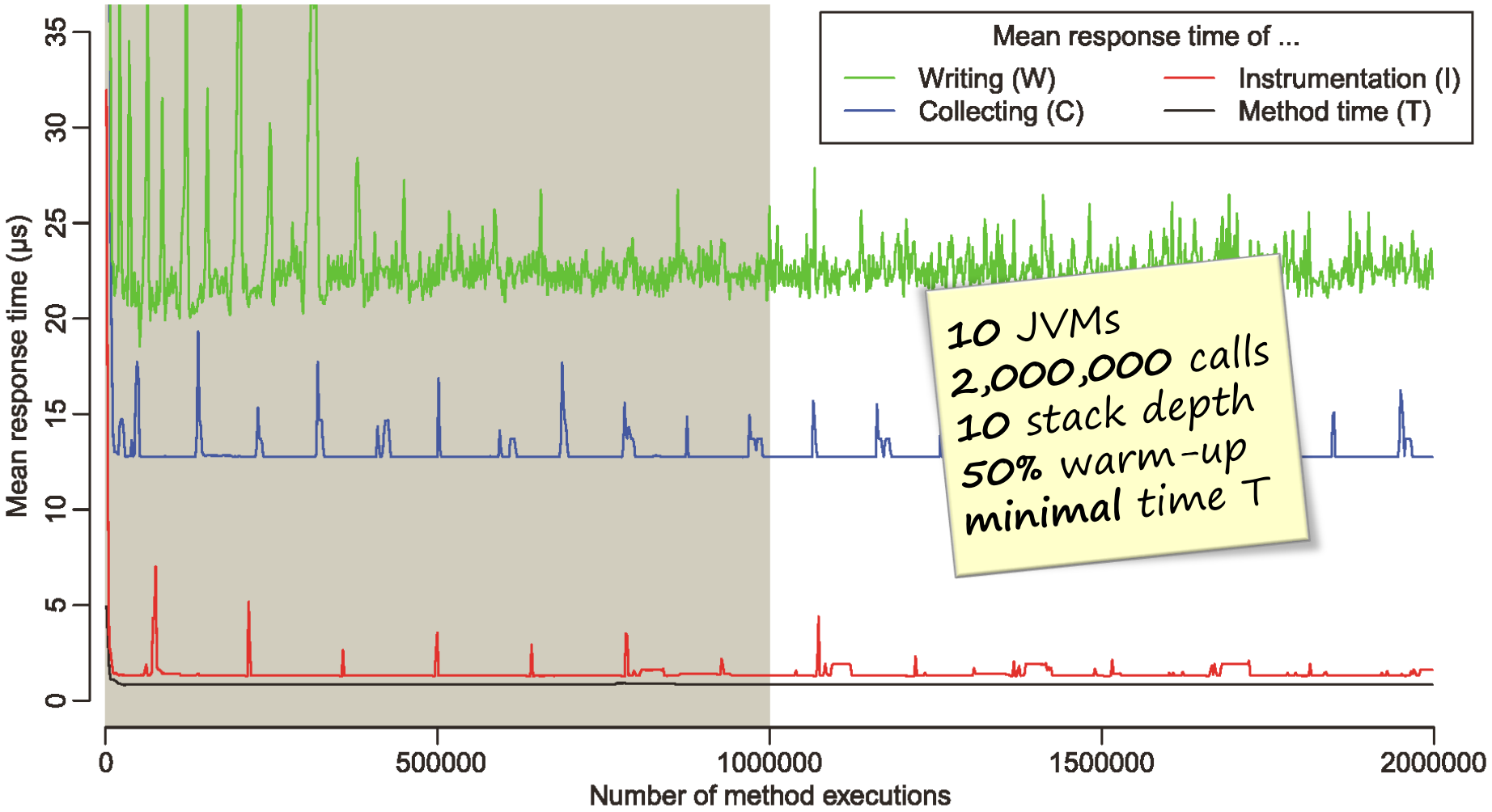
- Monitoring tool
- Research project
  - Univ. of Hildesheim
- Focus on resources
- Integrated analysis

Further reading: Chap. 4, 12 and [van Hoorn et al. 2012, Fittkau et al. 2013a, Siegl and Bouillet 2011, Eichelberger and Schmid 2014]

Evaluation

Warm-up vs. Steady State (Example)

MooBench

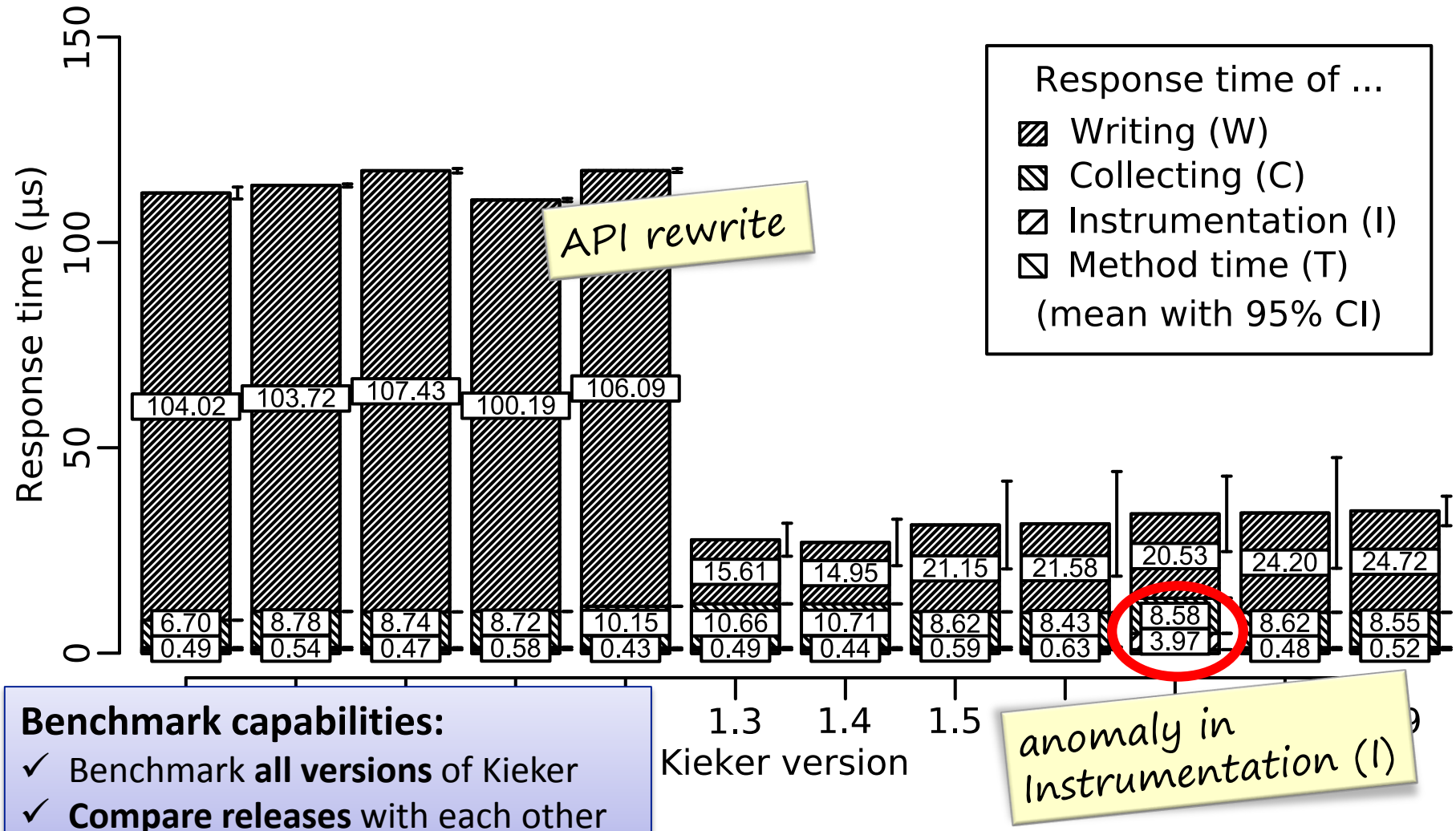


Further reading: Chap. 11 and [Waller and Hasselbring 2013]

Regression Benchmarks (Kieker)

Christian-Albrechts-Universität zu Kiel

MooBench



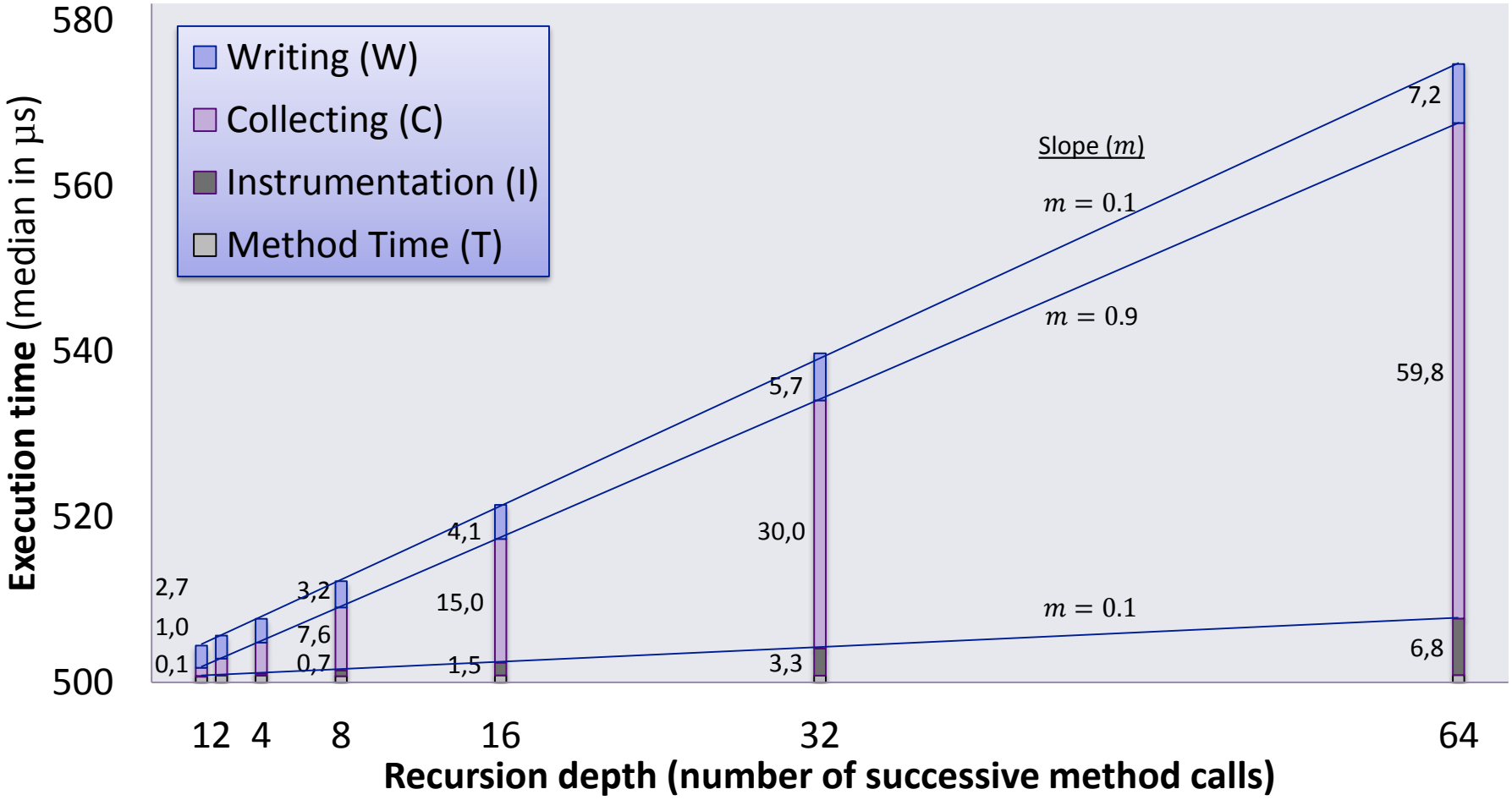
Benchmark capabilities:

- ✓ Benchmark **all versions** of Kieker
- ✓ **Compare releases** with each other
- ✓ Detect **performance regressions**

Further reading: Chap. 11 and [Waller and Hasselbring 2013]

Linear Increase of Overhead

MooBench



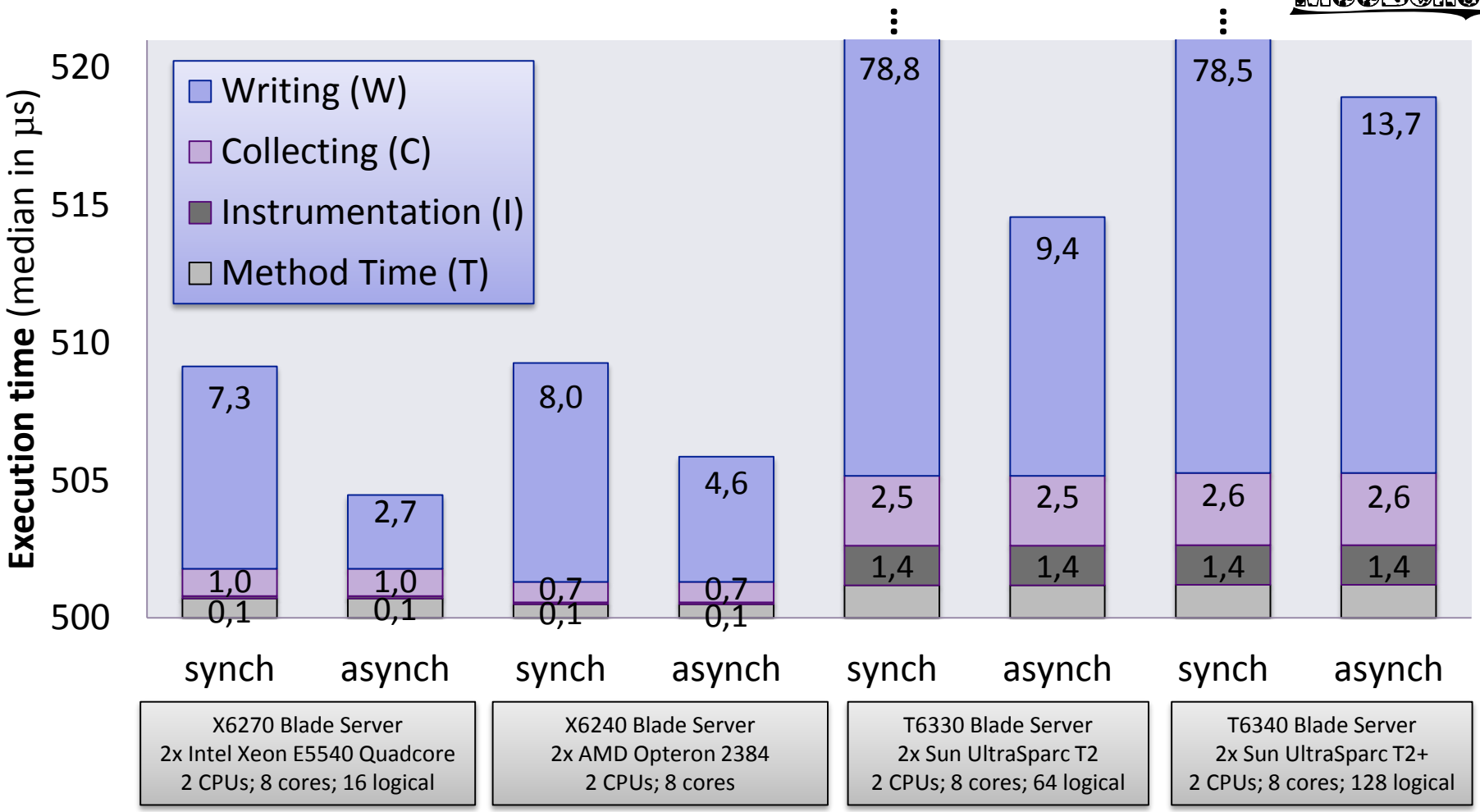
Benchmark capabilities:

✓ Benchmark with **scaling workloads**

Further reading: Chap. 11 and [Waller and Hasselbring 2012]

Multi-Core Architectures

MooBench



Benchmark capabilities:

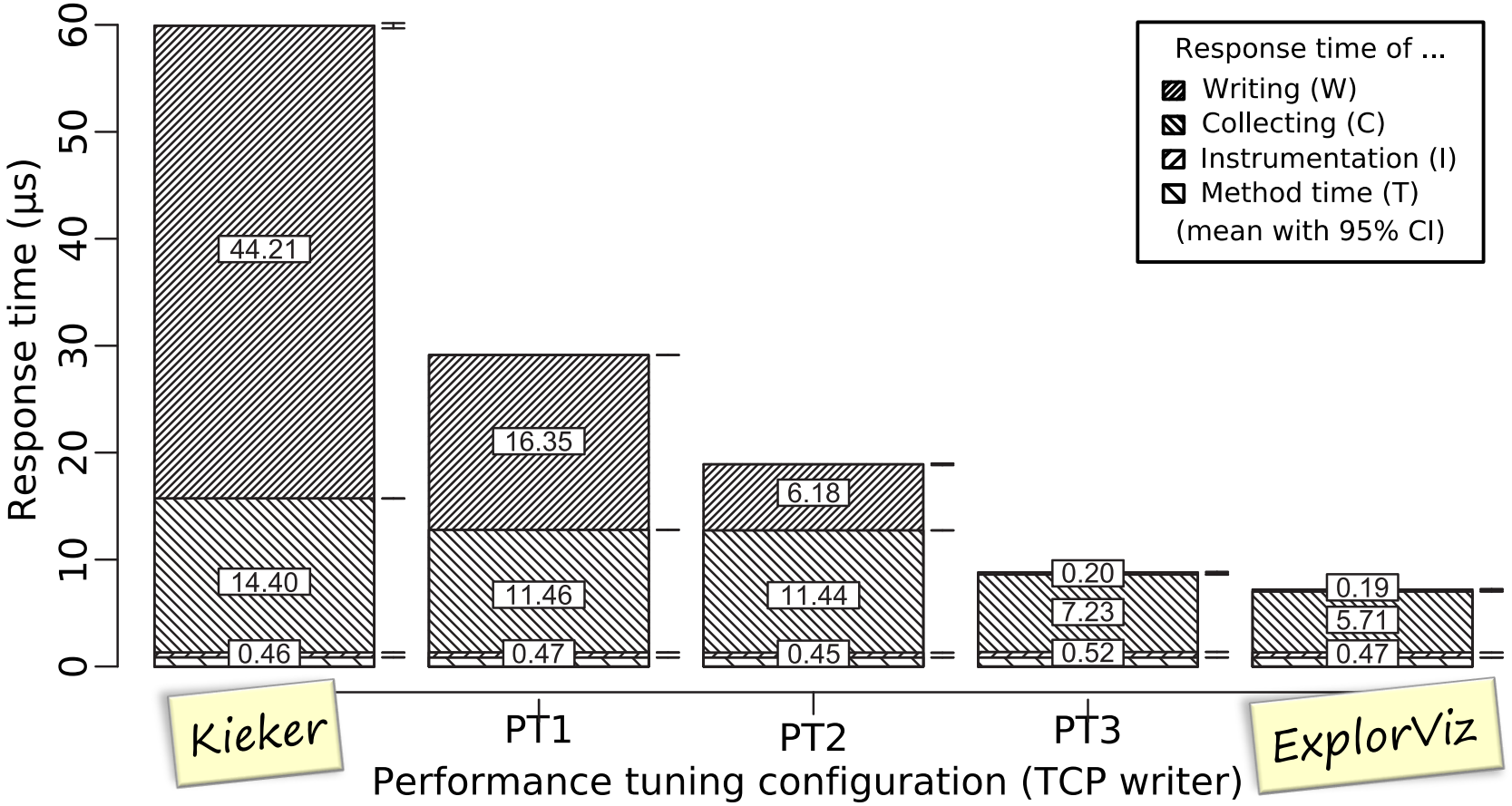
✓ Benchmark and **compare** different **environments**

Further reading: Chap. 11 and [Waller and Hasselbring 2012]

Performance Tuning (with MooBench)

C A U
Christian-Albrechts-Universität zu Kiel

MooBench



Benchmark capabilities:

- ✓ Benchmark to **guide a structured performance tuning** approach
- ✓ Benchmark **other tools** (ExplorViz monitoring)

Further reading: Chap. 11 and [Waller et al. 2014]

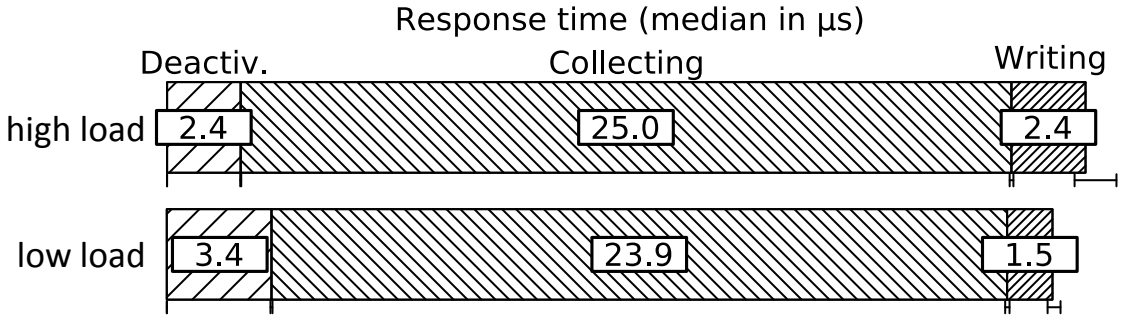
inspectIT [Siegl and Bouillet 2011]

MooBench

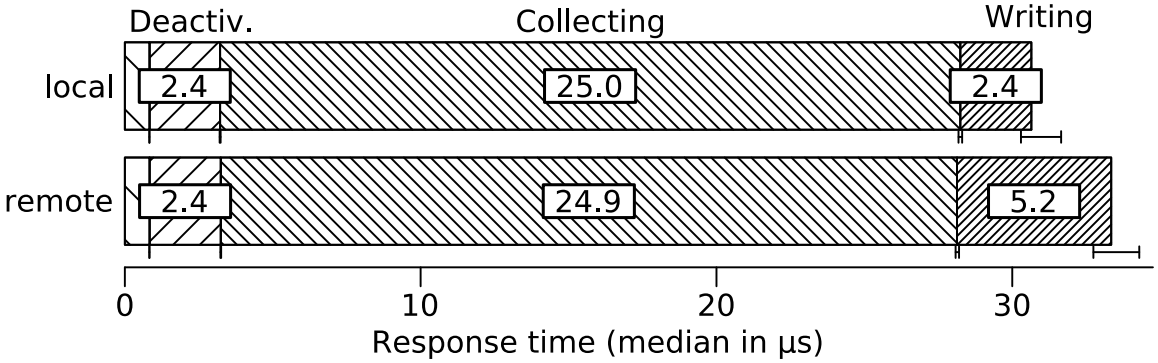
inspectIT ...because performance matters!

http://inspectit.eu

Compare high and low workloads



Compare local and remote analysis (under high workload)



Benchmark capabilities:

- ✓ Additional **commercial monitoring tools** (inspectIT)
- ✓ Only **minor adjustments** required

Adjustments integrated into future releases!

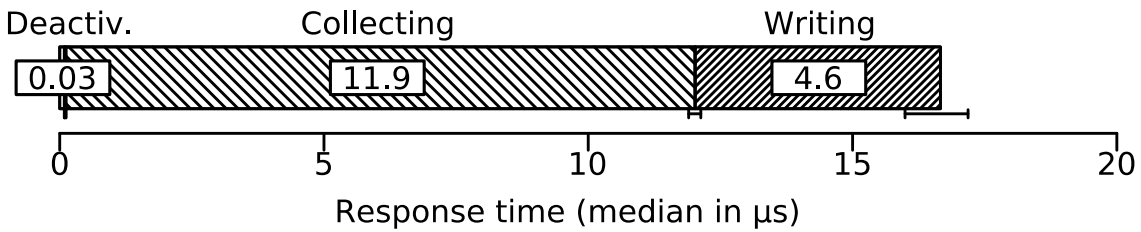
MooBench used in weekly load tests

Further reading: Chap. 12 and [Siegl and Bouillet 2011]

SPASS-meter [Eichelberger and Schmid 2014]

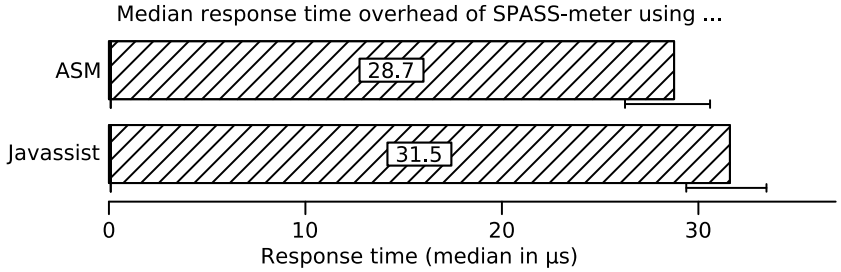
MooBench

Investigate causes of monitoring overhead

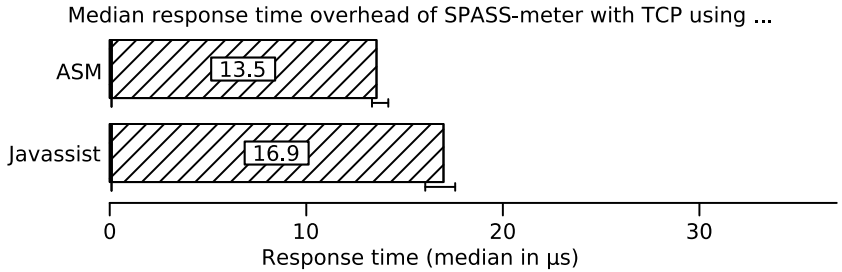


http://ssehub.github.com

Compare different technologies (local)



Compare different technologies (remote via TCP)



Benchmark capabilities:

- ✓ Additional **open-source monitoring tools** (SPASS-meter)
- ✓ Only **very minor adjustments** required

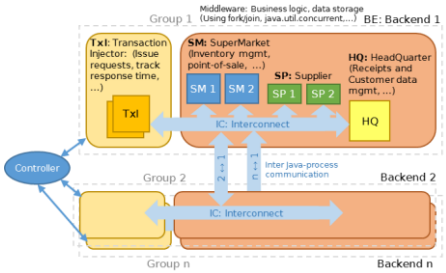
results similar to findings by [Eichelberger and Schmid 2014]

Further reading: Chap. 12 and [Eichelberger and Schmid 2014]

SPECjbb2013 [Pogue et al. 2014]

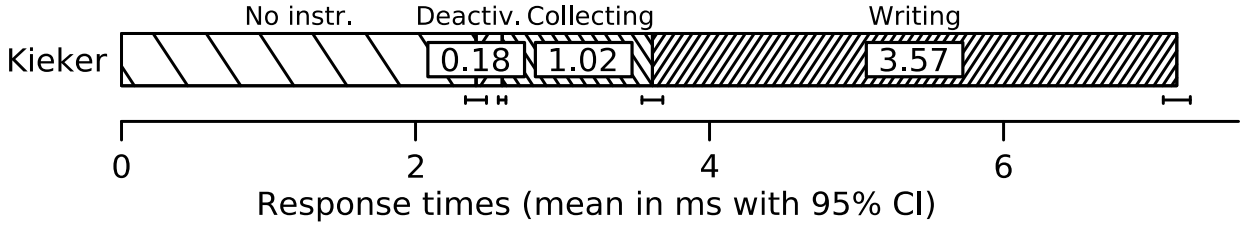
Determine capacity of system (workload as benchmark score)

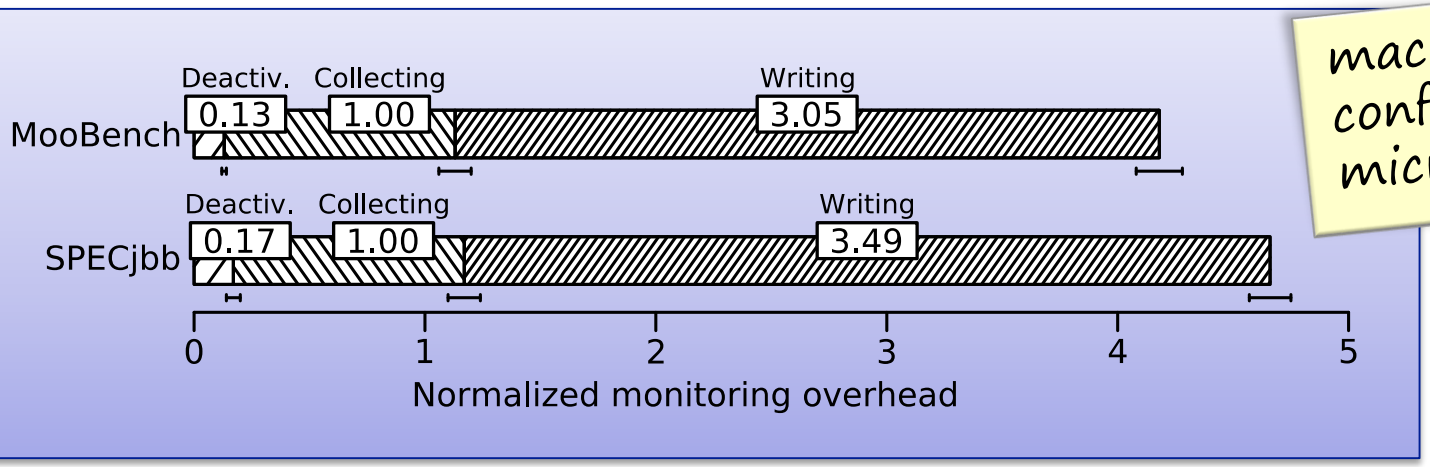
	No instr.	Deactiv.	Collect.	Writing
jOPS	268705	19490	2013	303



Kieker

Run experiments using determined capacity

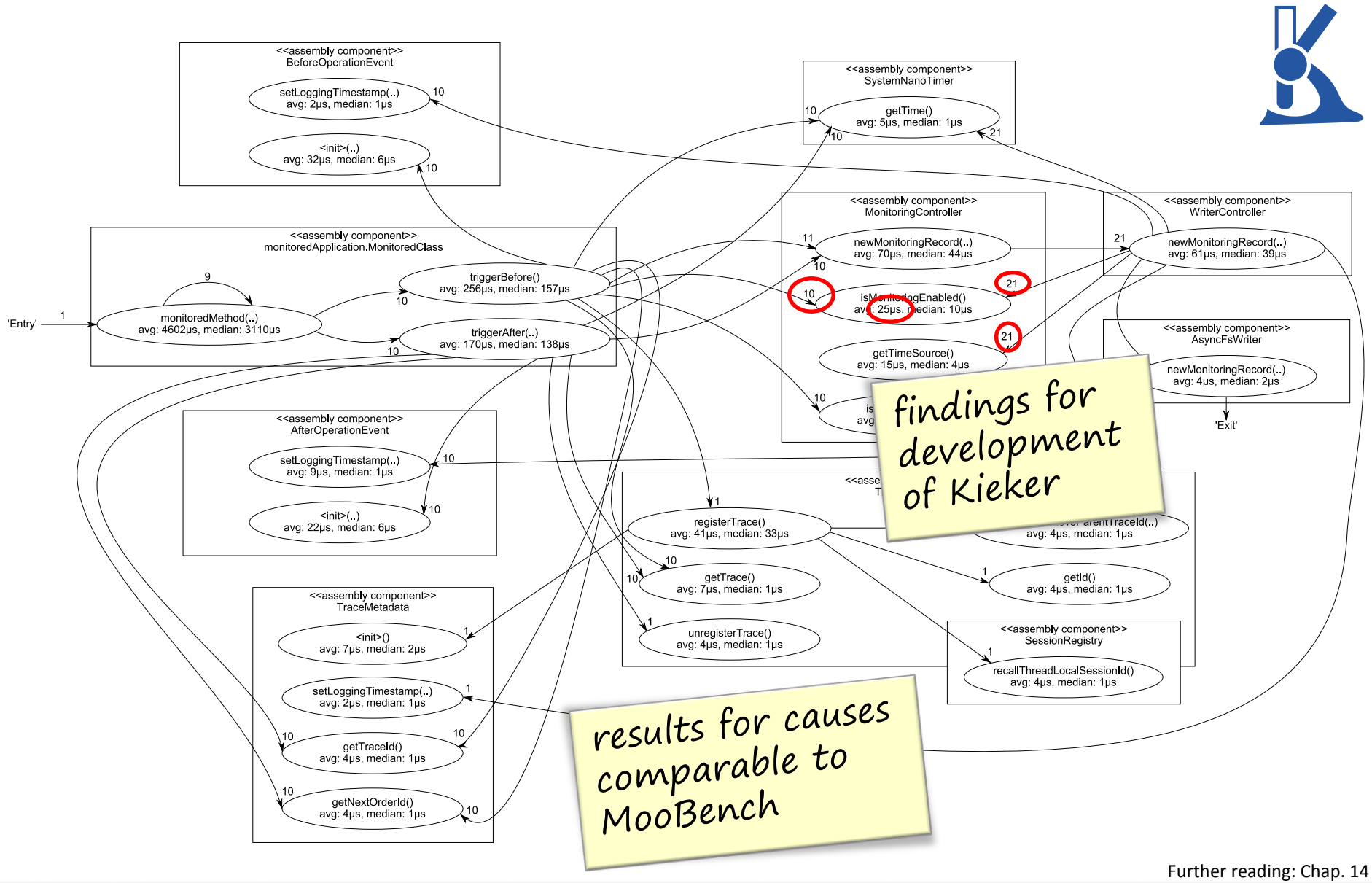




macro-benchmarks confirm findings of micro-benchmarks

Further reading: Chap. 13

Meta-Monitoring



Further reading: Chap. 14

Related Work & Outlook

Benchmark Engineering Methodology

- No **encompassing methodology**
  - see [Hinnant 1988, Price 1989, Sachs 2011]
- Only **15 of 50 publications** on **benchmark engineering**
  - E.g., [Gray 1993, Sim et al. 2003, Kounev 2005, Huppler 2009, Sachs 2011]
- **Execution of benchmarks mostly ignored in literature!**
  - Only recognized in more recent publication!
  - E.g., [Kounev 2005, Huppler 2009, Sachs 2011]

Further reading: Chap. 15

Measuring Monitoring Overhead

- Basic analysis (27 publications)
  - E.g., [Parsons et al. 2006, AppDynamics 2010]

no causes, no validation

- Causes of Overhead (13 publications)
  - E.g., [Kanstrén et al. 2011]

often more specialized

- Adaptive Monitoring (5 publications)
  - E.g., [Reiss 2008]

no causes

- Performance Evaluations of Kieker (4 publications)
  - E.g., [Focke 2006, Eichelberger and Schmid 2014]

less detailed

Further reading: Chap. 15

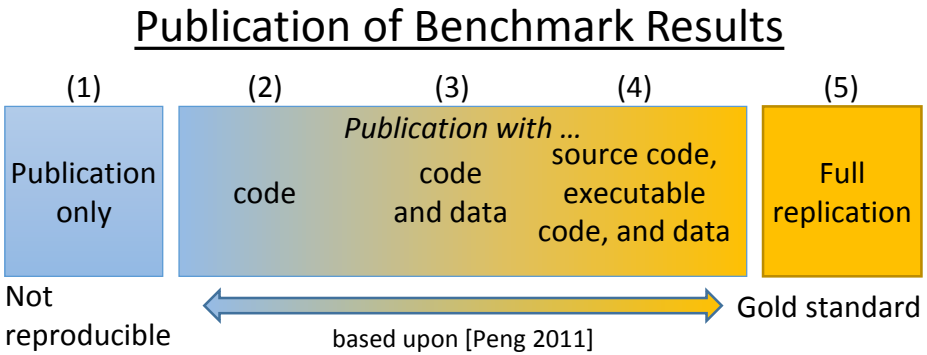
Outlook & Future Work

Replication and Validation

- MooBench as **open-source software**
- All **results available online**
  - Raw results and generated diagrams
  - Prepared experiments for all Kieker versions
  - Detailed description of experiments

MooBench
http://kieker-monitoring.net/MooBench

Kieker
http://kieker-monitoring.net



zenodo
http://zenodo.org/

Further reading: Chap. 16

Outlook & Future Work (cont.)

Benchmark Experiments

- Additional **monitoring frameworks/tools**
- Further **environments** besides Java

- Evaluation of **analysis overhead**
  - TeeTime [Wulf et al. 2014]
- Establish benchmarks in **community**
  - inspectIT
  - AIM [Flaig 2014, Schulz et al. 2014, Wert et al. 2015]
- Use in **continuous integration**
  - Automated regression benchmarks for Kieker [Waller et al. 2015]

Further reading: Chap. 16

Regression Benchmarks (Continuous Integration)

MooBench

Benchmark capabilities:

✓ Automated benchmarks in **continuous integration**

outlook on future work!

Further reading: Chap. 11 and [Waller et al. 2015]

Bibliography

[AppDynamics 2010] AppDynamics. AppDynamics Lite Performance Benchmark Report. May 2010. url: http://www.appdynamics.com/learn-more.

[Bloch 2009] J. Bloch. Performance Anxiety. Talk at JavaOne conference. June 2009.

[Eichelberger and Schmid 2014] H. Eichelberger and K. Schmid. Flexible resource monitoring of Java programs. Journal of Systems and Software 93 (July 2014), pages 163–186.

[Fittkau et al. 2013a] F. Fittkau, J. Waller, C. Wulf, and W. Hasselbring. Live trace visualization for comprehending large software landscapes: The ExplorViz approach. In: 1st IEEE International Working Conference on Software Visualization (VISSOFT 2013). IEEE Computer Society, Sept. 2013, pages 1–4.

[Fittkau et al. 2013b] F. Fittkau, J. Waller, P. C. Brauer, and W. Hasselbring. Scalable and live trace processing with Kieker utilizing cloud computing. In: Proceedings of the Symposium on Software Performance: Joint Kieker/Palladio Days (KPDays 2013). CEUR Workshop Proceedings, Nov. 2013, pages 89–98.

[Flaig 2014] A. Flaig. Dynamic Instrumentation in Kieker Using Runtime Bytecode Modification. Bachelor thesis. Institute of Software Technology, University of Stuttgart, Germany, Nov. 2014.

[Focke 2006] T. Focke. Performance Monitoring von Middleware-basierten Applikationen. German. Diploma thesis. University of Oldenburg, Mar. 2006.

[Gray 1993] J. Gray, editor. The Benchmark Handbook: For Database and Transaction Systems. 2nd edition. Morgan Kaufmann, May 1993.

[Hinnant 1988] D. F. Hinnant. Accurate Unix benchmarking: Art, science, or black magic? IEEE Micro 8.5 (Oct. 1988), pages 64–75.

[Huppler 2009] K. Huppler. The art of building a good benchmark. In: First TPC Technology Conference on Performance Evaluation and Benchmarking (TPCTC 2009). Springer, Aug. 2009, pages 18–30.

[Jeffery 1996] C. L. Jeffery. Program Monitoring and Visualization: An Exploratory Approach. Springer, June 1996.

[Jones 2010] D. Jones. The Five Essential Elements of Application Performance Monitoring. Quest Software. Nov. 2010.

[Kanstrén et al. 2011] T. Kanstrén, R. Savola, S. Haddad, and A. Hecker. An adaptive and dependable distributed monitoring framework. International Journal On Advances in Security 4.1&2 (Sept. 2011), pages 80–94.

[Kounev 2005] S. Kounev. Performance Engineering of Distributed Component-Based Systems – Benchmarking, Modeling and Performance Prediction. PhD thesis. TU Darmstadt, Germany, Dec. 2005.

[Parsons et al. 2006] T. Parsons, A. Mos, and J. Murphy. Non-intrusive end-to-end runtime path tracing for J2EE systems. IEEE Software 153.4 (Aug. 2006), pages 149 –161.

[Peng 2011] R. D. Peng. Reproducible research in computational science. Science 334.6060 (Dec. 2011), pages 1226–1227.

[Plattner and Nievergelt 1981] B. Plattner and J. Nievergelt. Special feature: Monitoring program execution: A survey. IEEE Computer 14.11 (Nov. 1981), pages 76–93.

[Pogue et al. 2014] C. Pogue, A. Kumar, D. Tollefson, and S. Realmuto. SPECjbb2013 1.0: An overview. In: Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering (ICPE '14). ACM, Mar. 2014, pages 231–232.

[Price 1989] W. J. Price. A benchmark tutorial. IEEE Micro 9.5 (Oct. 1989), pages 28–43.

[Reimer 2013] S. Reimer. Architekturzentriertes Monitoring für den Betrieb. Talk at Softwareforen Leipzig. Nov. 2013.

[Reiss 2008] S. P. Reiss. Controlled dynamic performance analysis. In: Proceedings of the 7th International Workshop on Software and Performance (WOSP '08). ACM, June 2008, pages 43–54.

[Sachs 2011] K. Sachs. Performance Modeling and Benchmarking of Event-Based Systems. PhD thesis. TU Darmstadt, Germany, Aug. 2011.

[Schulz et al. 2014] H. Schulz, A. Flaig, A. Wert, and A. van Hoorn. Adaptive Instrumentation of Java-Applications for Experiment-Based Performance Analysis. Talk at Symposium on Software Performance. Nov. 2014.

[Shao et al. 2010] J. Shao, H. Wei, Q. Wang, and H. Mei. A runtime model based monitoring approach for cloud. In: Proceedings of the 3rd International Conference on Cloud Computing (CLOUD'10). IEEE Computer Society, July 2010, pages 313–320.

[Siegl and Bouillet 2011] S. Siegl and P. Bouillet. inspectIT ...because performance matters! White paper. NovaTec, June 2011.

[Sim et al. 2003] S. E. Sim, S. Easterbrook, and R. C. Holt. Using benchmarking to advance research: A challenge to software engineering. In: Proceedings of the 25th International Conference on Software Engineering (ICSE 2003). IEEE Computer Society, May 2003, pages 74–83.

[Smith and William 2001] C. U. Smith and L. G. Williams. Performance Solutions – A Practical Guide to Creating Responsive, Scalable Software. Addison-Wesley, Sept. 2001.

[van Hoorn et al. 2009] A. van Hoorn, M. Rohr, W. Hasselbring, J. Waller, J. Ehlers, S. Frey, and D. Kieselhorst. Continuous Monitoring of Software Services: Design and Application of the Kieker Framework. Technical report TR-0921. Department of Computer Science, Kiel University, Germany, Nov. 2009.

[van Hoorn et al. 2012] A. van Hoorn, J. Waller, and W. Hasselbring. Kieker: A framework for application performance monitoring and dynamic software analysis. In: Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering (ICPE 2012). ACM, Apr. 2012, pages 247–248.

[Waller and Hasselbring 2012] J. Waller and W. Hasselbring. A comparison of the influence of different multi-core processors on the runtime overhead for application-level monitoring. In: Multicore Software Engineering, Performance, and Tools (MSEPT). Springer, June 2012, pages 42–53.

[Waller 2013] J. Waller. Benchmarking the Performance of Application Monitoring Systems. Technical report TR-1312. Department of Computer Science, Kiel University, Germany, Nov. 2013.

[Waller and Hasselbring 2013] J. Waller and W. Hasselbring. A benchmark engineering methodology to measure the overhead of application-level monitoring. In: Proceedings of the Symposium on Software Performance: Joint Kieker/Palladio Days (KPDays 2013). CEUR Workshop Proceedings, Nov. 2013, pages 59–68.

[Waller et al. 2014] J. Waller, F. Fittkau, and W. Hasselbring. Application performance monitoring: Trade-off between overhead reduction and maintainability. In: Proceedings of the Symposium on Software Performance: Joint Descartes/Kieker/Palladio Days (SoSP 2014). Nov. 2014, pages 1–24.

[Waller et al. 2015] J. Waller, N. C. Ehmke, and W. Hasselbring. Including performance benchmarks into continuous integration (2015). Submitted publication.

[Wert et al. 2015] A. Wert, H. Schulz, C. Heger, and R. Farahbod. AIM: Adaptable Instrumentation and Monitoring for automated software performance analysis. In: Submitted publication. 2015.

[Woodside et al. 2007] C. M. Woodside, G. Franks, and D. C. Petriu. The future of software performance engineering. In: International Conference on Software Engineering, Workshop on the Future of Software Engineering (FOSE 2007). IEEE Computer Society, May 2007, pages 171–187.

[Wulf et al. 2014] C. Wulf, N. C. Ehmke, and W. Hasselbring. A generic and concurrency-aware pipes-and-filters framework. In: Submitted publication. Nov. 2014.

Backup Slides

Goal, Question, Metric (GQM)

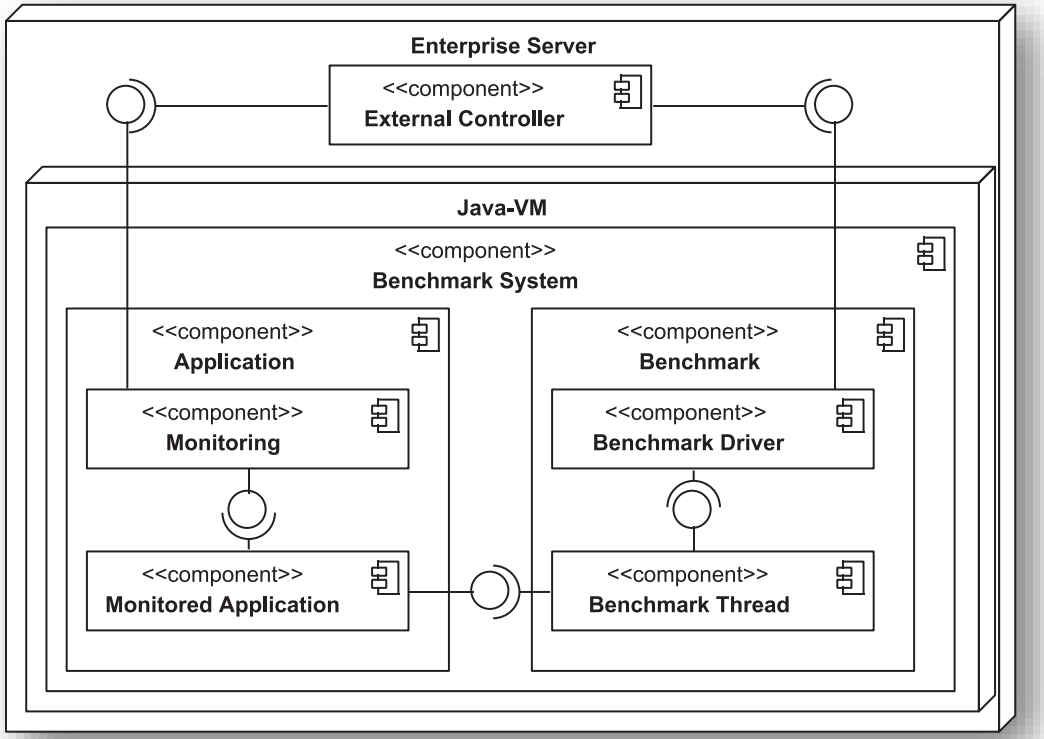
Goal	**G**	Measure and quantify the performance overhead of a monitoring framework with the MooBench micro-benchmark.
Question	**Q1**	Which monitoring tools can be benchmarked?
Metrics	**M1**	tools or frameworks
	M2	required changes for simple benchmarks
	M3	required changes for cause analysis
Question	**Q2**	What effort is required to benchmark?
Metrics	**M2**	required changes for simple benchmarks
	M3	required changes for cause analysis
	M4	required run-time of the benchmark
Question	**Q3**	Can the monitoring overhead be quantified?
Metrics	**M5**	different scenarios
	M6	configurability of the benchmark
	M7	reproducibility of benchmark results
Question	**Q4**	Are the benchmark results representative?
Metrics	**M7**	reproducibility of benchmark results
	M8	differences to other benchmarks

Micro-Benchmarks

Benchmark designed to measure individual portions

- *External Controller* configures *Monitoring* and *Driver*
- *Monitored Application* provides fixed *T*
- *Benchmark Threads*

  call *monitored method*
  - #totalCalls
  - #recodedCalls
- Run **4 times** to measure



complete source code available at:
http://kieker-monitoring.net

MooBench (Monitored Class)

Listing 8.2. Required Java interface for the Monitored Application

```
1  public interface MonitoredClass {
2    public long monitoredMethod(long methodTime, int recDepth);
3  }
```

Listing 8.3. Basic implementation of the MonitoredClass interface

```
1  public final class MonitoredClassSimple implements MonitoredClass {
2    public final long monitoredMethod(long methodTime, int recDepth) {
3      if (recDepth > 1) {
4        return this.monitoredMethod(methodTime, recDepth - 1);
5      } else {
6        final long exitTime = System.nanoTime() + methodTime;
7        long currentTime;
8        do {
9          currentTime = System.nanoTime();
10       } while (currentTime < exitTime);
11       return currentTime;
12     }
13   }
14 }
```

Further reading: Chap. 8 and [Waller and Hasselbring 2013, Waller et al. 2014]

MooBench (Benchmark Loop)

Listing 8.4. Excerpt of the Benchmark Thread's run method

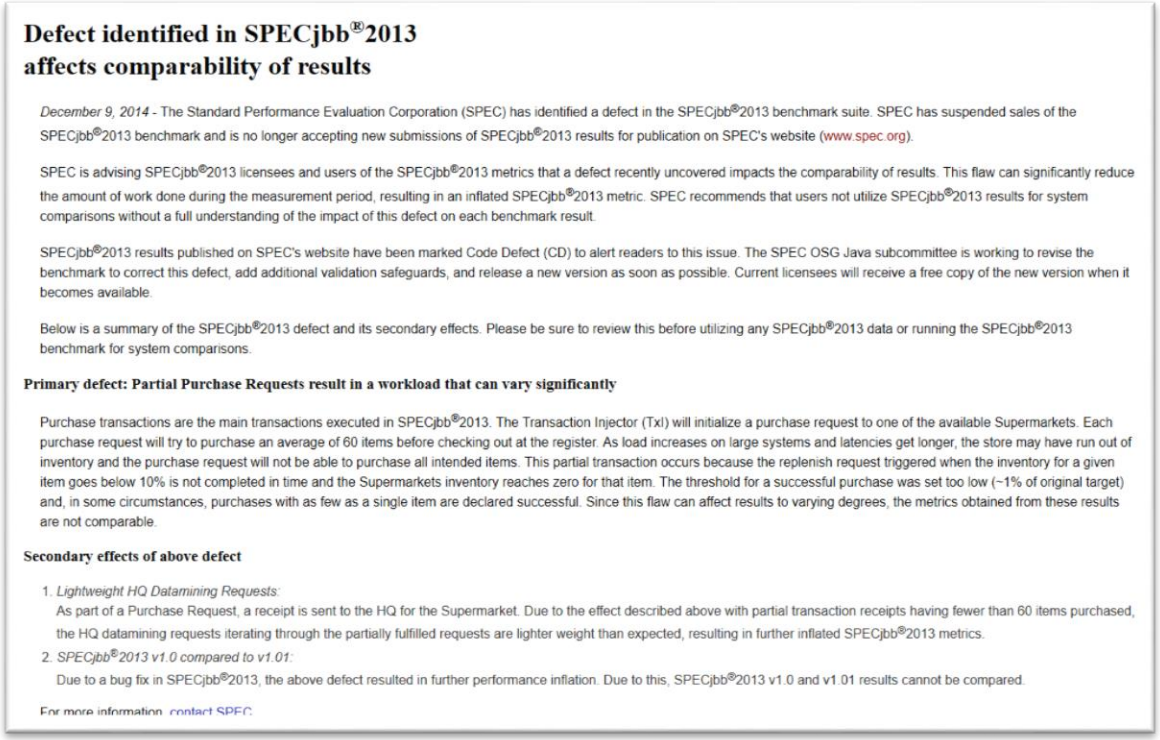
```
1  public final void run() {
2    long start_ns;
3    long stop_ns;
4    for (int i = 0; i < totalCalls; i++) {
5      start_ns = System.nanoTime();
6      monitoredClass.monitoredMethod(methodTime, recursionDepth);
7      stop_ns = System.nanoTime();
8      timings[i] = stop_ns - start_ns;
9    }
10 }
```

Further reading: Chap. 8 and [Waller and Hasselbring 2013, Waller et al. 2014]

SPECjbb2013 (Defect identified – Dec. 09, 2014)

Cause of Defect

- Supermarkets running out of wares under extremely high load
- Usual goal
  - generate high load
  - compare systems
- Our approach
  - utilizes rather low load
  - might be affected by defect
  - **however**, experiment easily repeatable, as soon as bug fix is released



Defect identified in SPECjbb®2013 affects comparability of results

December 9, 2014 - The Standard Performance Evaluation Corporation (SPEC) has identified a defect in the SPECjbb®2013 benchmark suite. SPEC has suspended sales of the SPECjbb®2013 benchmark and is no longer accepting new submissions of SPECjbb®2013 results for publication on SPEC's website (www.spec.org).

SPEC is advising SPECjbb®2013 licensees and users of the SPECjbb®2013 metrics that a defect recently uncovered impacts the comparability of results. This flaw can significantly reduce the amount of work done during the measurement period, resulting in an inflated SPECjbb®2013 metric. SPEC recommends that users not utilize SPECjbb®2013 results for system comparisons without a full understanding of the impact of this defect on each benchmark result.

SPECjbb®2013 results published on SPEC's website have been marked Code Defect (CD) to alert readers to this issue. The SPEC OSG Java subcommittee is working to revise the benchmark to correct this defect, add additional validation safeguards, and release a new version as soon as possible. Current licensees will receive a free copy of the new version when it becomes available.

Below is a summary of the SPECjbb®2013 defect and its secondary effects. Please be sure to review this before utilizing any SPECjbb®2013 data or running the SPECjbb®2013 benchmark for system comparisons.

Primary defect: Partial Purchase Requests result in a workload that can vary significantly

Purchase transactions are the main transactions executed in SPECjbb®2013. The Transaction Injector (TxI) will initialize a purchase request to one of the available Supermarkets. Each purchase request will try to purchase an average of 60 items before checking out at the register. As load increases on large systems and latencies get longer, the store may have run out of inventory and the purchase request will not be able to purchase all intended items. This partial transaction occurs because the replenish request triggered when the inventory for a given item goes below 10% is not completed in time and the Supermarkets inventory reaches zero for that item. The threshold for a successful purchase was set too low (~1% of original target) and, in some circumstances, purchases with as few as a single item are declared successful. Since this flaw can affect results to varying degrees, the metrics obtained from these results are not comparable.

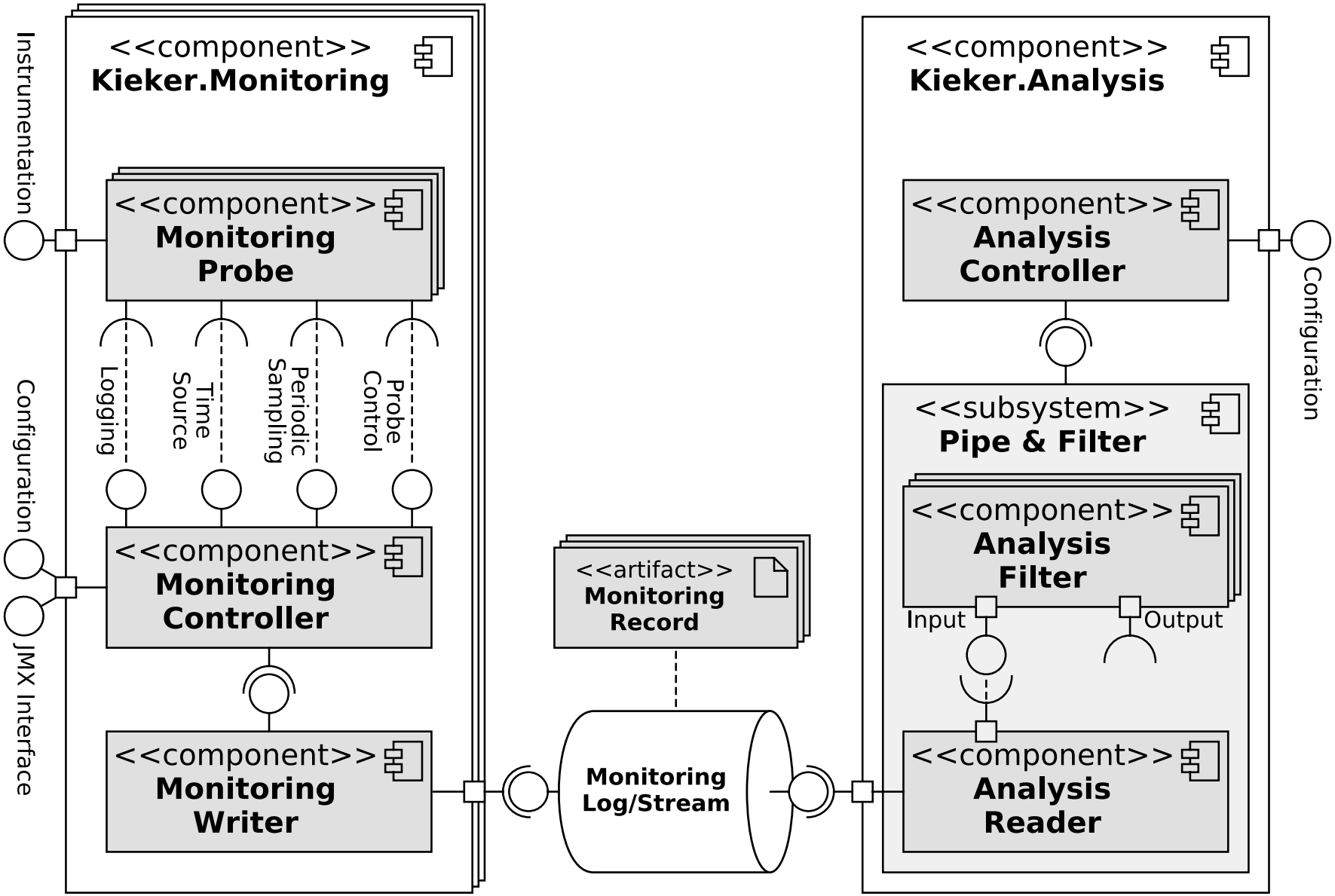
Secondary effects of above defect

1. *Lightweight HQ Datamining Requests:*
   As part of a Purchase Request, a receipt is sent to the HQ for the Supermarket. Due to the effect described above with partial transaction receipts having fewer than 60 items purchased, the HQ datamining requests iterating through the partially fulfilled requests are lighter weight than expected, resulting in further inflated SPECjbb®2013 metrics.
2. *SPECjbb®2013 v1.0 compared to v1.01:*
   Due to a bug fix in SPECjbb®2013, the above defect resulted in further performance inflation. Due to this, SPECjbb®2013 v1.0 and v1.01 results cannot be compared.

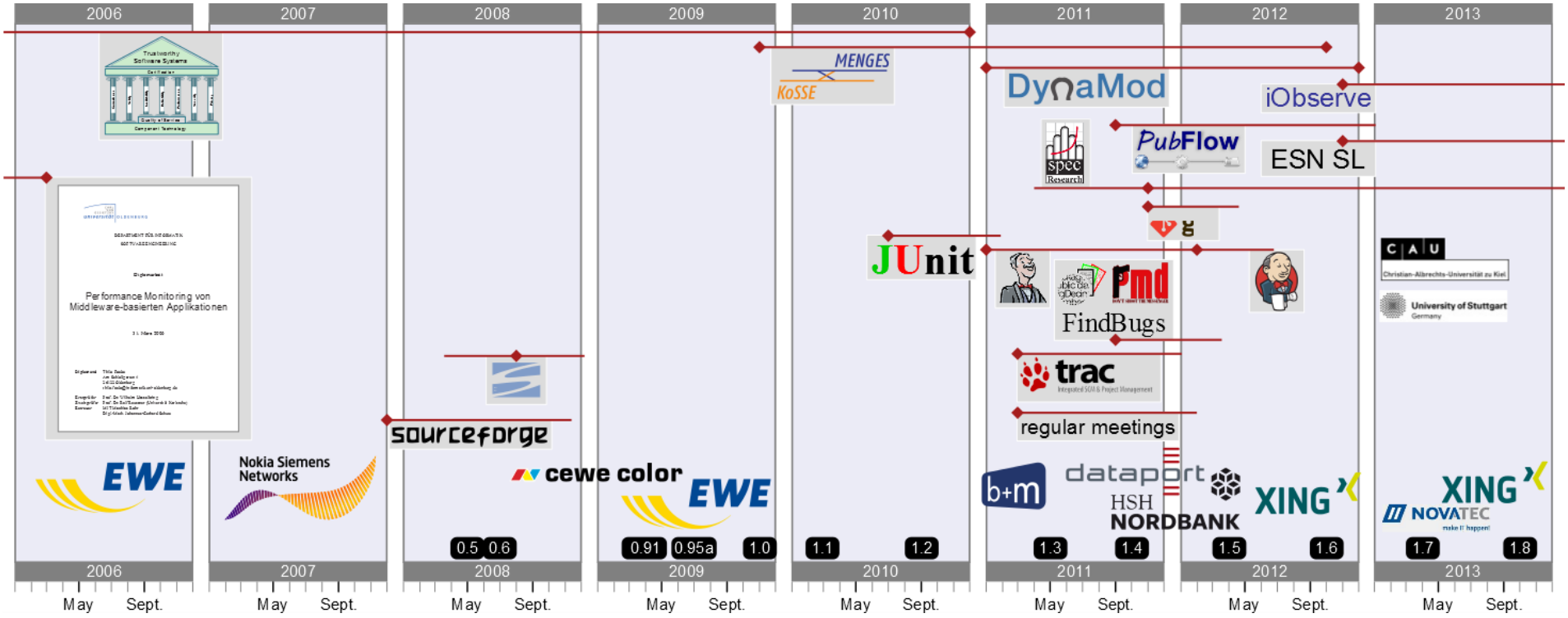
For more information, contact SPEC

http://www.spec.org/jbb2013/defectnotice.html

Kieker Architecture



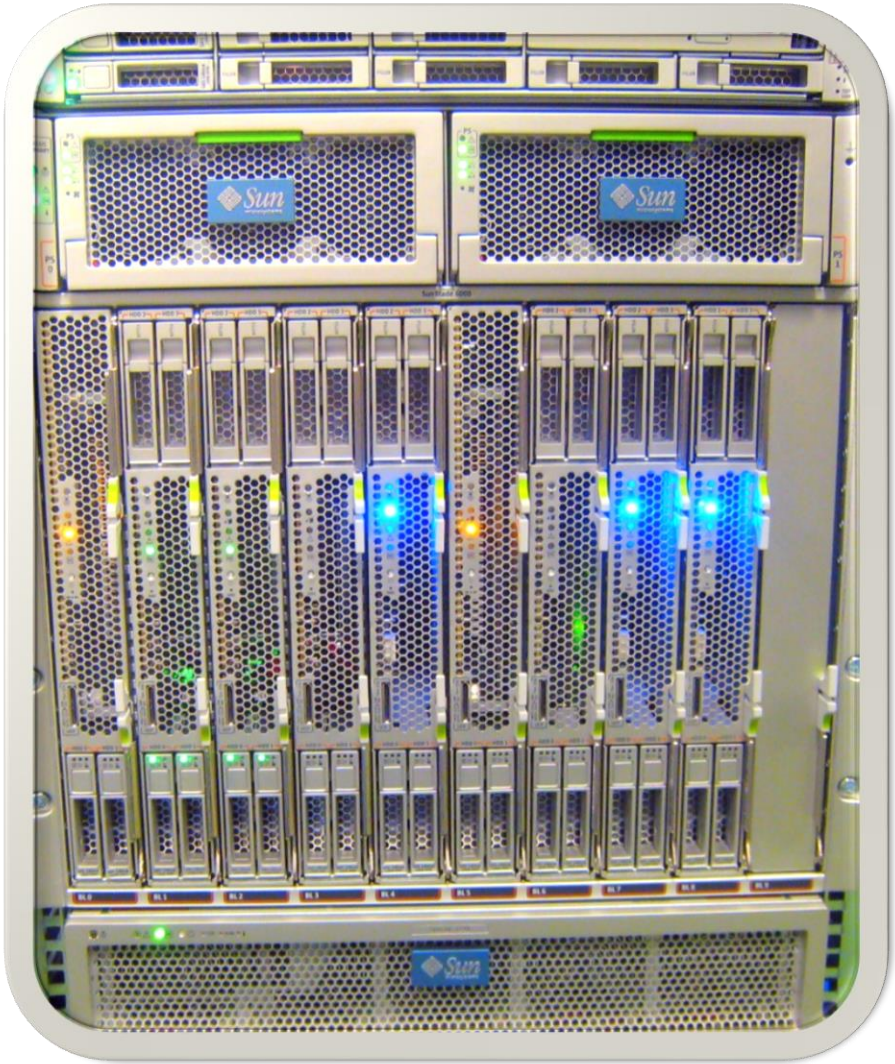
Kieker: Small Moments



Performance comparison with MooBench

Experimental Setup (usually)

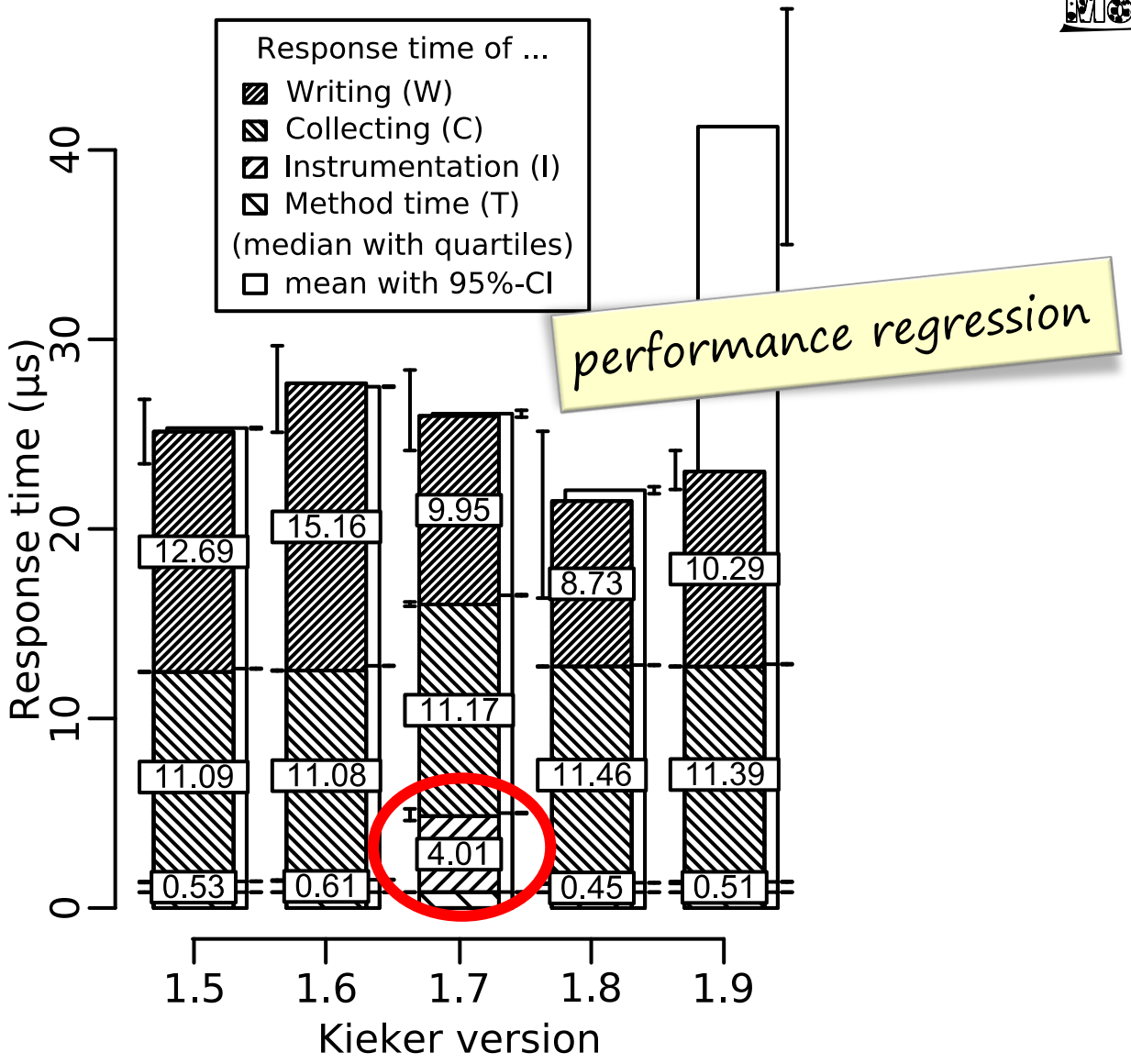
- X6270 Blade Server
  - 2x Intel Xeon 2.53 GHz
  - 24 GiB RAM
  - Solaris 10
  - Java 1.5 – 1.7 (64bit)
  - AspectJ
- X6240 AMD
- T6330/6340 SUN



Further reading: Chap. 11 and [Waller and Hasselbring 2012, Waller and Hasselbring 2013, Waller 2013, Waller et al. 2014]

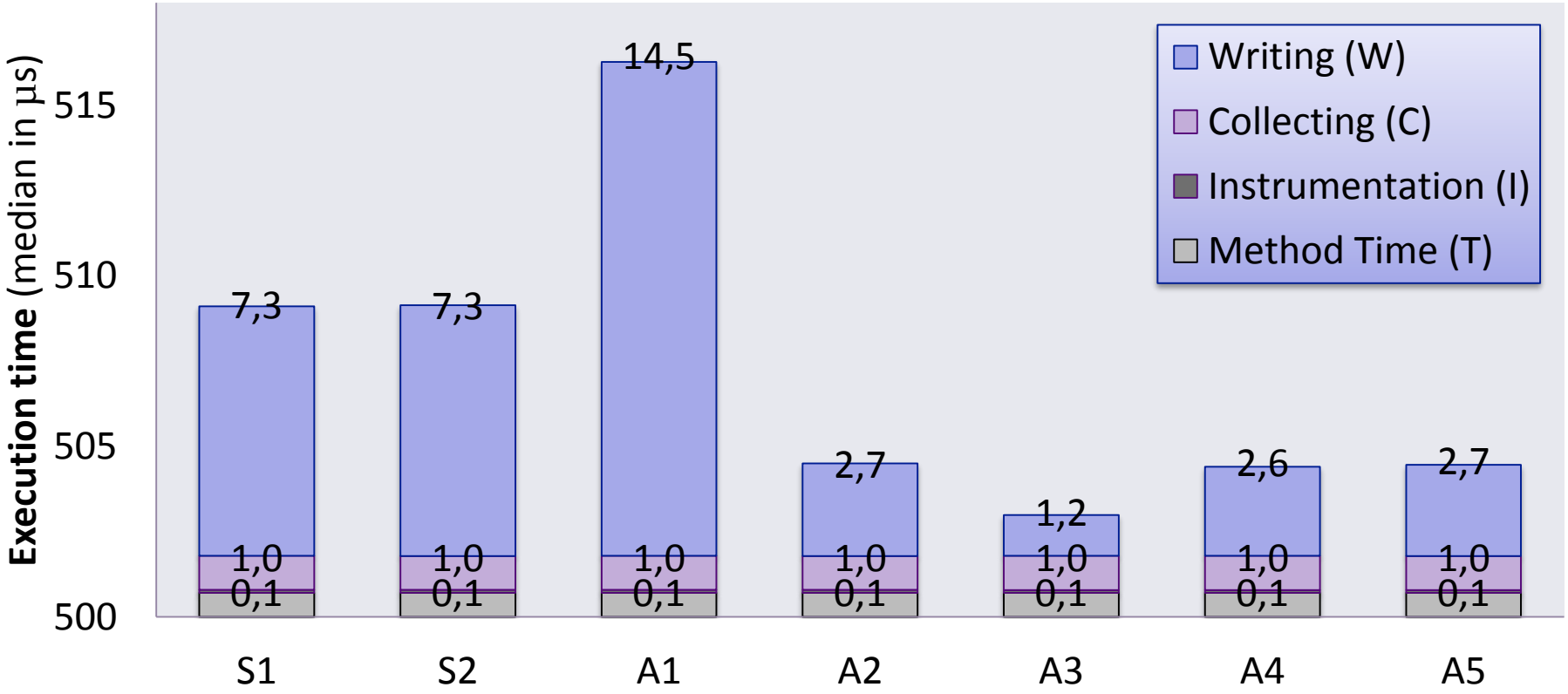
Regression Benchmarks (Kieker)

MooBench



Further reading: Chap. 11 and [Waller and Hasselbring 2013]

Multi-Core Experiments

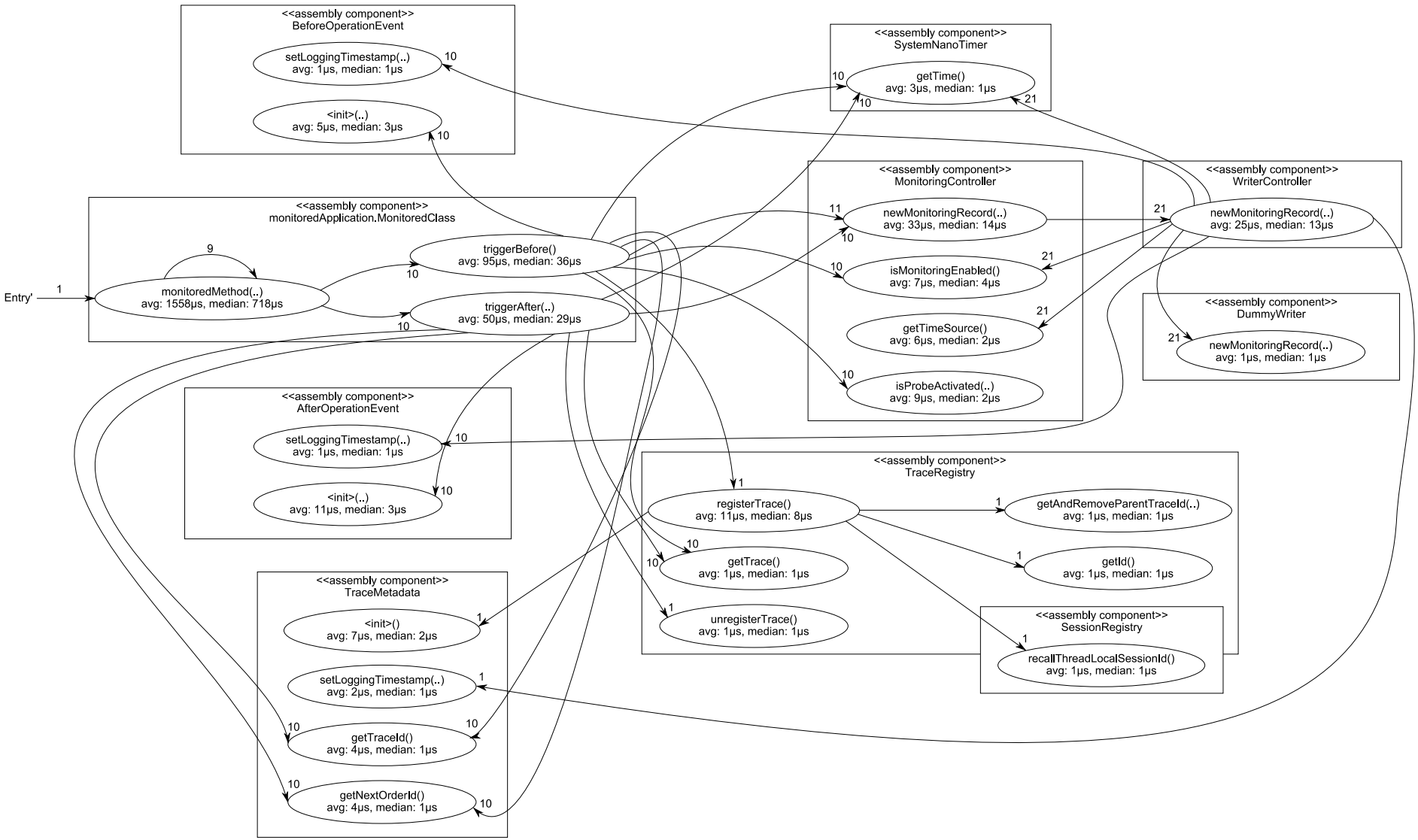


Exp	Writer	Cores	Notes
S1	SyncFS	1	single physical core
S2	SyncFS	2	two logical cores on the same physical core

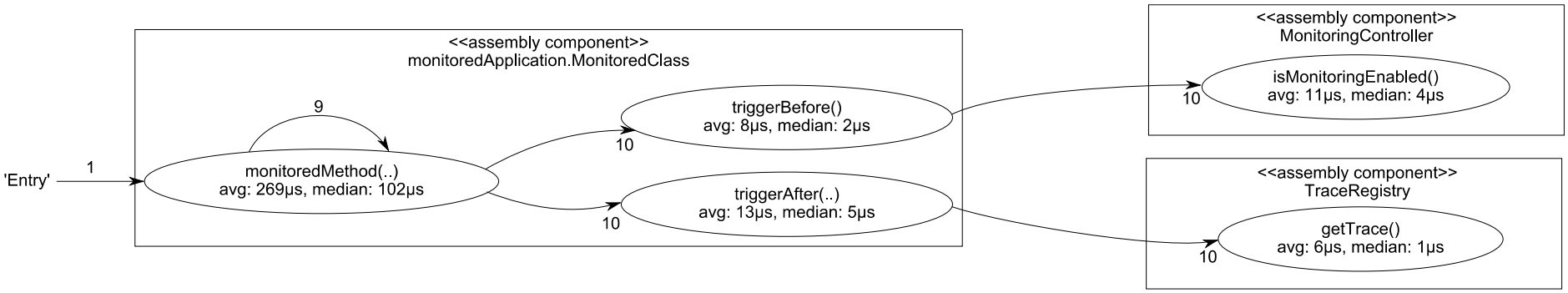
Exp	Writer	Cores	Notes
A1	AsyncFS	1	single physical core
A2	AsyncFS	2	two logical cores on the same physical core
A3	AsyncFS	2	two physical cores on the same processor
A4	AsyncFS	2	two physical cores on different processors
A5	AsyncFS	16	whole system is available

X6270 Blade Server
2x Intel Xeon 2.53 GHz E5540 Quadcore / 24 GB RAM
Solaris 10 / Oracle Java x64 Server VM 1.6.0_26 (1 GB heap)

Meta-Monitoring (no writing)



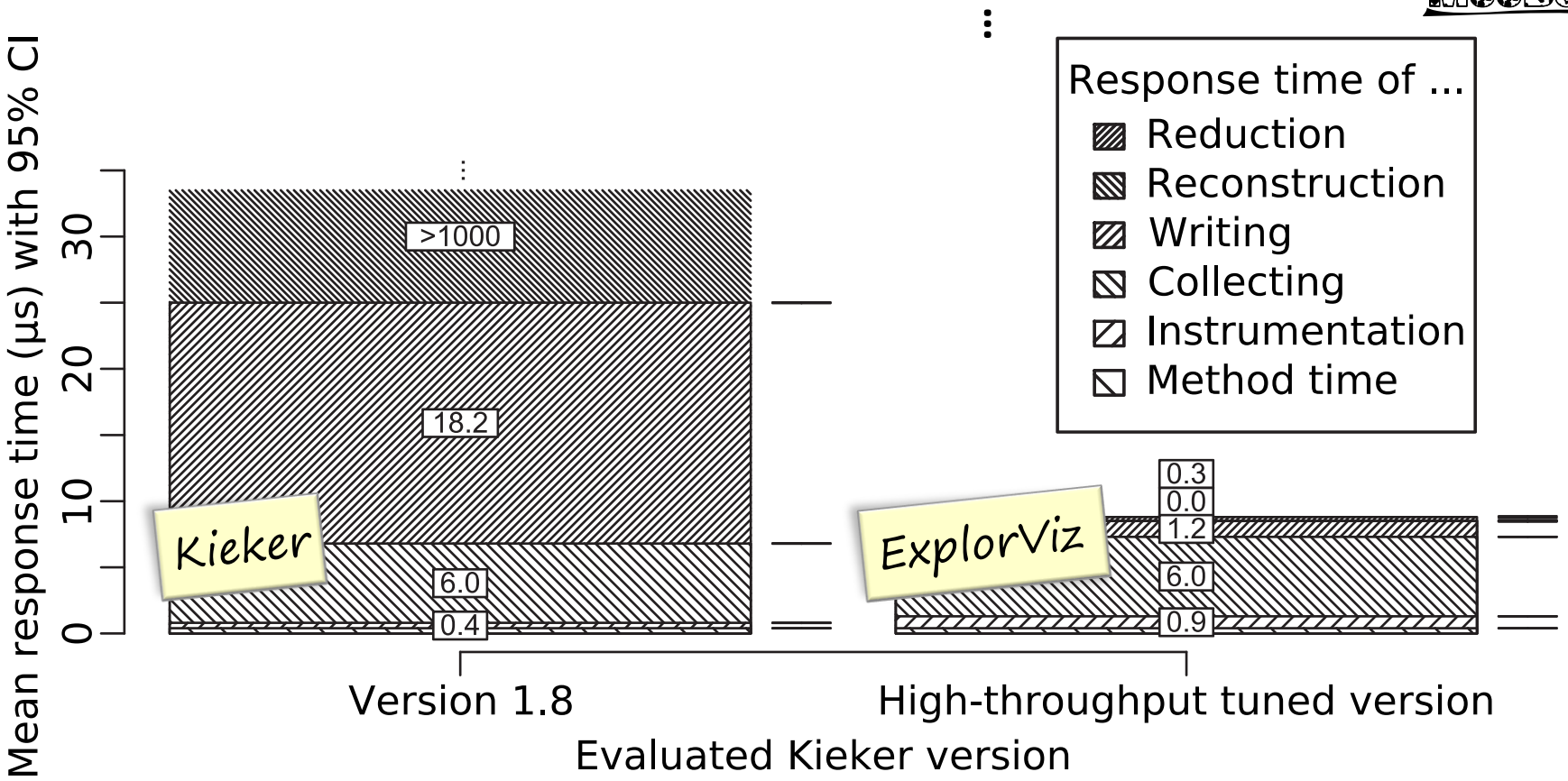
Meta-Monitoring (deactivated probe)



ExplorViz Visualizations

Benchmarking the Analysis

MooBench



Benchmark capabilities:

- ✓ Additional **benchmark scenarios** (online analysis)
- ✓ Additional **environments** (private cloud)
- ✓ Additional **monitoring tools** (ExplorViz monitoring)

Further reading: Chap. 11 and [Fittkau et al. 2013b]