SEARIS Workshop

Reducing Application-Stage Latencies For Real-Time Interactive Systems

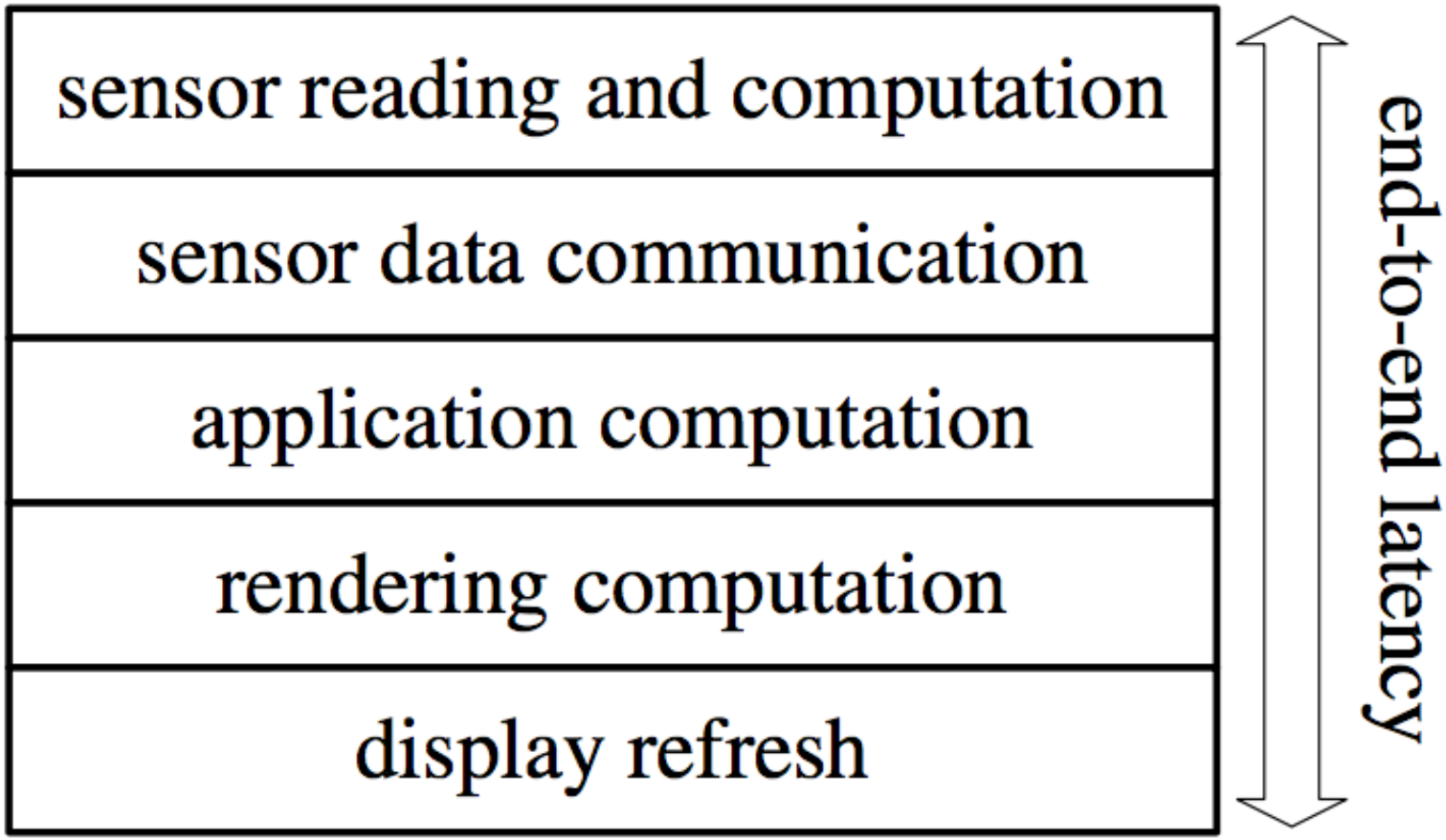
Jan-Philipp Stauffert, Florian Niebling, Marc Erich Latoschik

Informatik IX, Mensch-Computer-Interaktion
Universität Würzburg

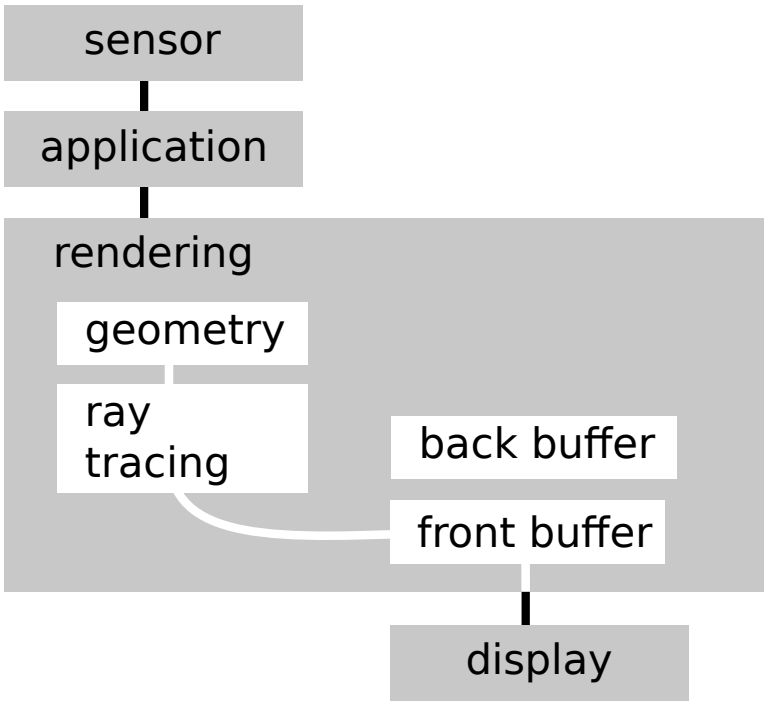
April 11, 2016

Overview

Latency Sources [Mine, 1993]



Motion-To-Photon Latency Reduction - Frameless Rendering



Frameless Rendering [Bishop et al., 1994, Dayal et al., 2005]

- Avoids double buffering
- Updates pixels randomly in the front buffer

Motion-To-Photon Latency Reduction - Image Warping

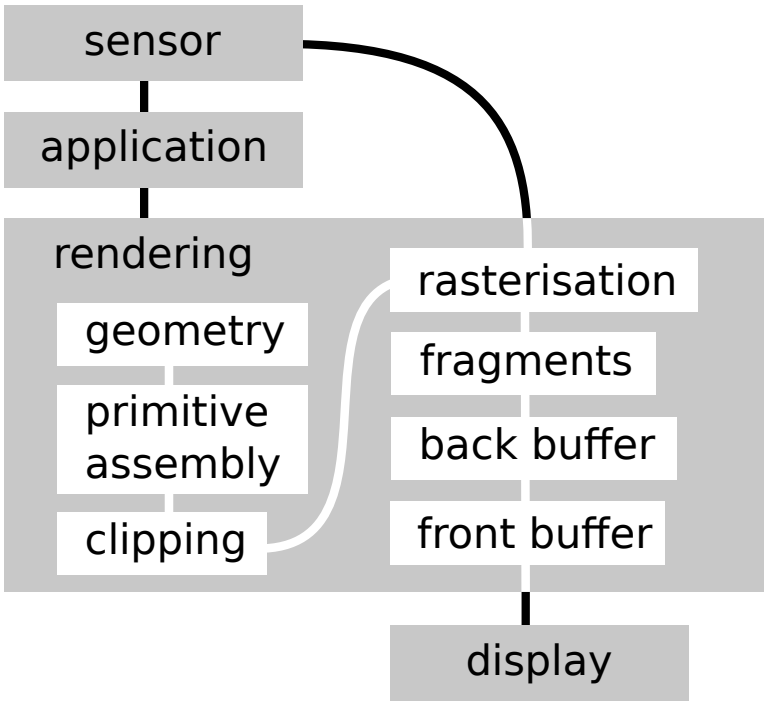
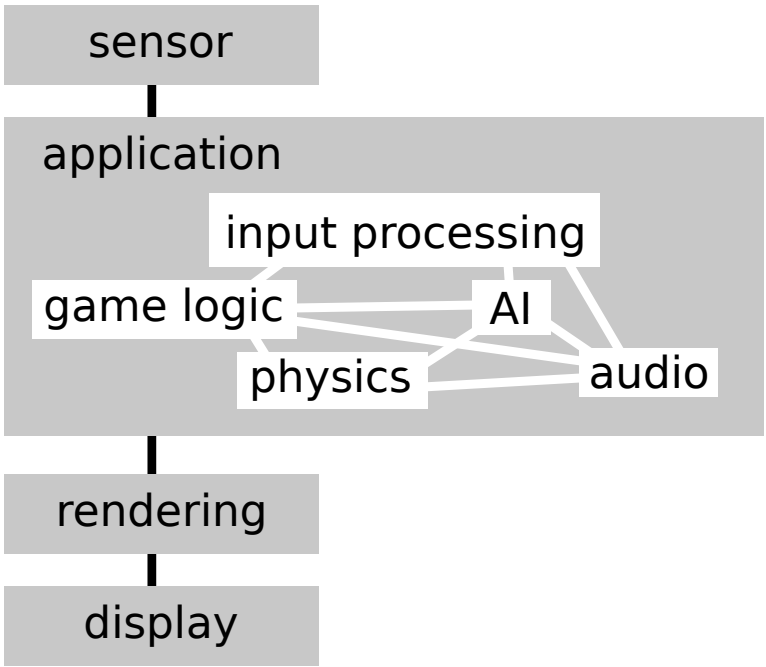


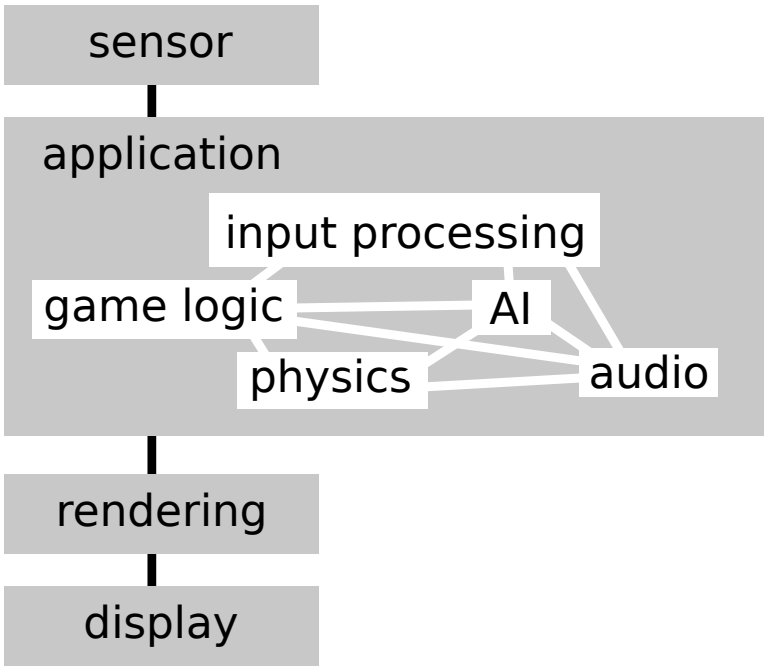
Image Warping [McMillan and Bishop, 1995]

- Warps image at the end of the rendering
- Uses most recent sensor data for warping

Application-stage latency

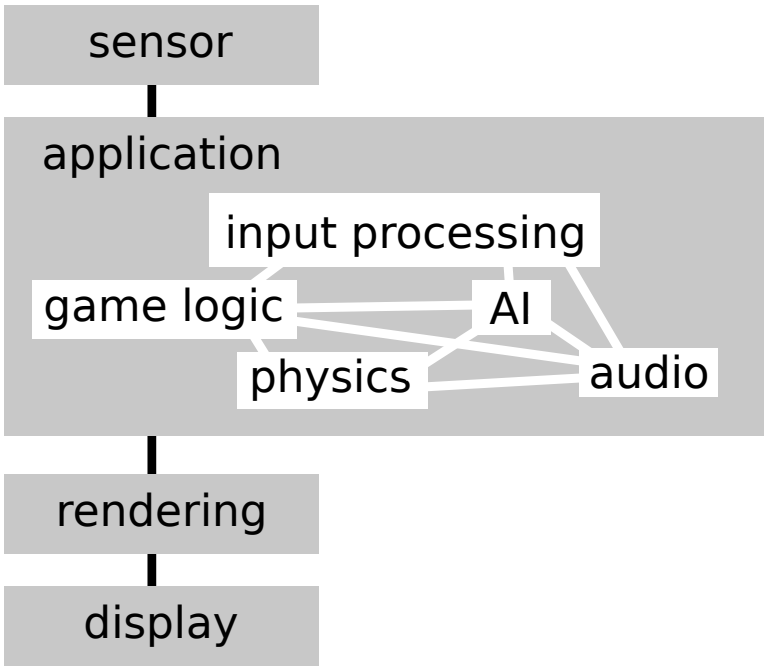


Application-stage latency



- VR applications consist of several components

Application-stage latency



- VR applications consist of several components
- Parts have to communicate to synchronise world state

Latency

Observation

1. Effects of time invariant latency is well researched
   - [Frank et al., 1988]
   - [Ivkovic et al., 2015]

Latency

Observation

1. Effects of time invariant latency is well researched
   - [Frank et al., 1988]
   - [Ivkovic et al., 2015]
2. Less research about latency jitter
   - Test subject performance degredation due to latency spikes [Teather et al., 2009]

Latency

Observation

1. Effects of time invariant latency is well researched
   - [Frank et al., 1988]
   - [Ivkovic et al., 2015]
2. Less research about latency jitter
   - Test subject performance degredation due to latency spikes [Teather et al., 2009]
3. Most research in latency is for the sensor and rendering stage

Latency

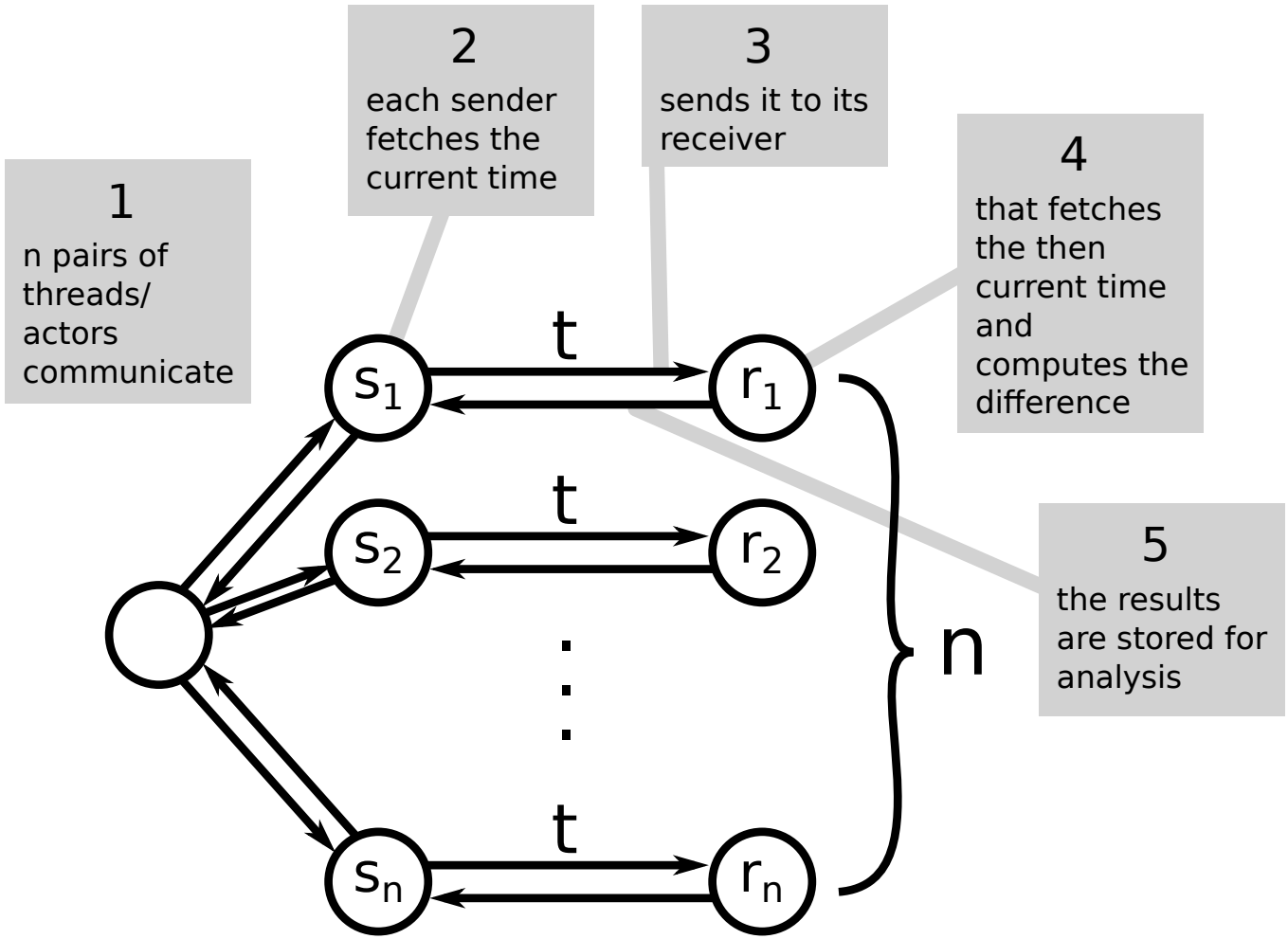
Observation

1. Effects of time invariant latency is well researched
   - [Frank et al., 1988]
   - [Ivkovic et al., 2015]
2. Less research about latency jitter
   - Test subject performance degredation due to latency spikes [Teather et al., 2009]
3. Most research in latency is for the sensor and rendering stage

Question

How can latency spikes in the application stage be measured and how to alleviate them?

Our test approach



Schedulers as Sources of Latency

Observation

OS schedulers give different promises about real-time behaviour

`SCHED_OTHER` - default scheduler

Best effort round robin.

`SCHED_FIFO` - Real-Time scheduler

Runs the first task of the list until it yields the CPU.

`SCHED_RR` - Real-Time scheduler

Promises each task the same share of CPU time in round robin.

Schedulers as Sources of Latency

`SCHED_DEADLINE` - Real-Time Scheduler

Each task under this policy is assigned a deadline, and the earliest-deadline task is executed.
Earliest deadline first.

Tests

Concurrency Comparison Chart

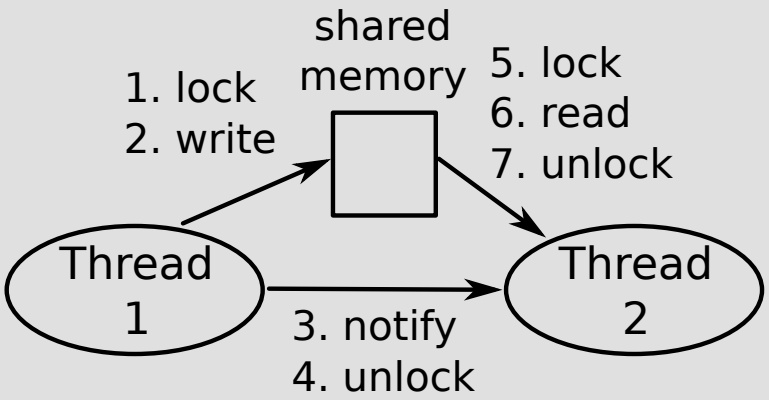
Platform	Parallelisation: Threads	Parallelisation: Actors
JVM (Bytecode)	Java Threads	Scala / Akka[1]
Linux (native binaries)	C++ pthreads	C++ Actor Framework (CAF)[2]

[1] http://akka.io/

[2] [Charousset et al., 2014]

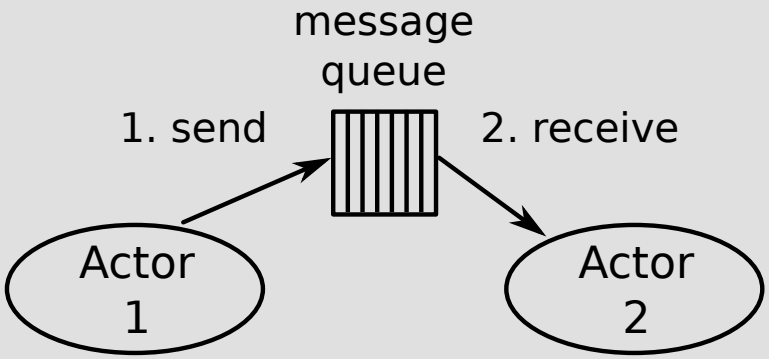
Difference of used Parallelisation Methods

Threads



- use shared memory
- lock and unlock
- use OS scheduler

Actors



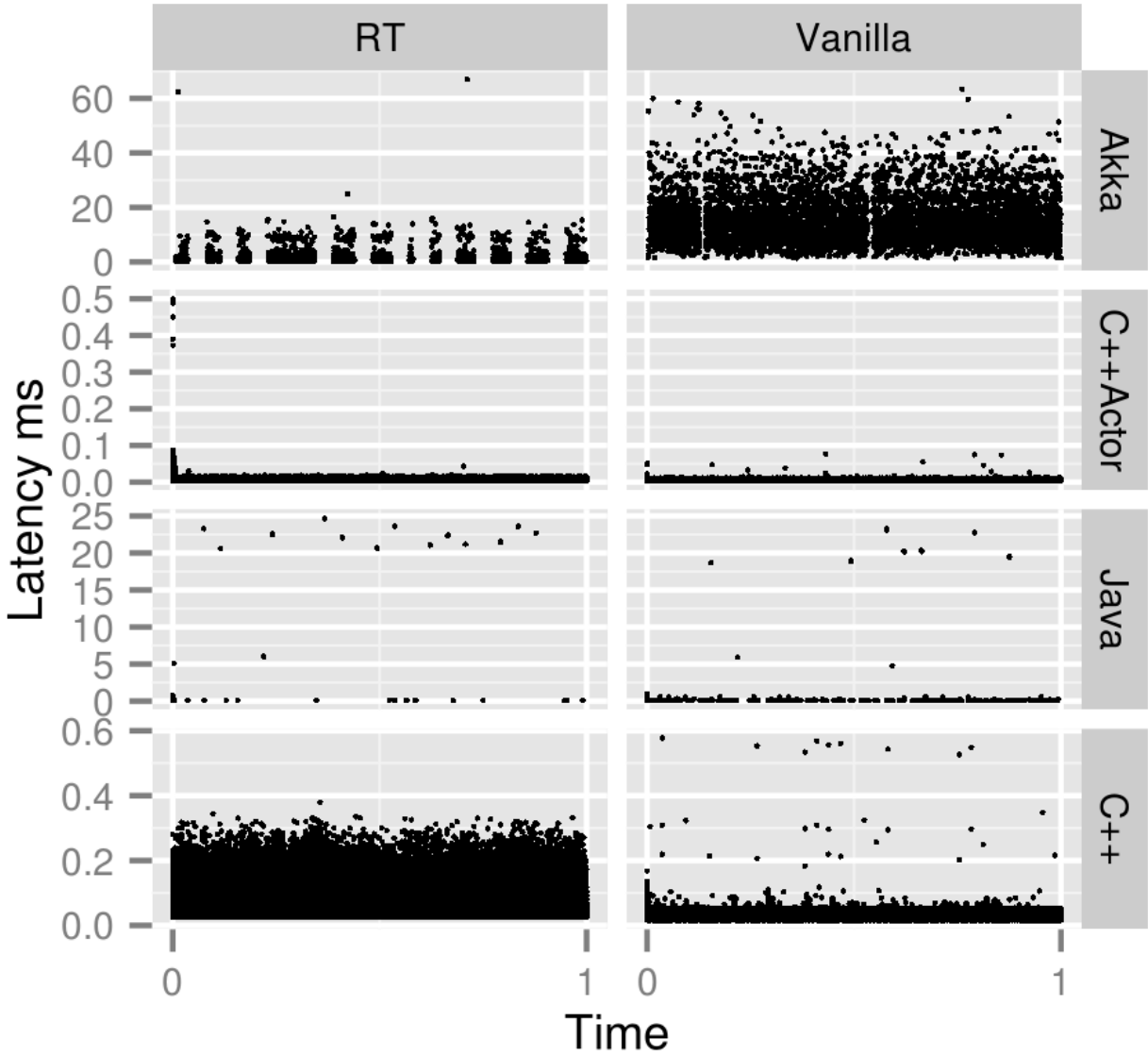
- send immutable messages
- queues usually implemented lockfree
- run on top of thread pool
- use own scheduler on top of OS scheduler

Used Benchmarking Conditions

1. 10 000 000 samples per test run
2. Plots presented use 16 thread/actor pairs
3. Only samples with latency $>$ median $+ 2 \cdot$ standard deviation were plotted
4. Time was normalised for better display:
   $[0; 1] \equiv [\text{test begin}; \text{test end}]$
5. Ubuntu 14.04.3 with a Linux Kernel version of 3.14.57
6. Intel© Core™ i7-2700K with 4 cores and hyperthreading

Results of the OS Scheduling on Latency

Latency Jitter Results for Different Operating Systems



Max, Mean, and Median of Latency Values

	Max		Mean		Median	
	RT	Vanilla	RT	Vanilla	RT	Vanilla
Akka	67ms	63.4ms	1.9ms	15.9ms	915.8μs	14.5ms
Java	24.7ms	23.3ms	684.2μs	106.2μs	144.3μs	67.5μs
C++	380.2μs	577.9μs	48μs	34.8μs	43.1μs	35.3μs
C++Actor	130.8μs	72.5μs	4.1μs	3.7μs	3.5μs	3.2μs

(1) Comparison values for latencies with and without RT patch with 16 thread/actor pairs

Findings in Comparison of Operating Systems

1. The C++ implementation shows the expected reaction:
   - Outliers on the vanilla Linux due to scheduler latency
   - Reduced outliers on the Linux RT
   - More mean latency on Linux RT due to scheduling overhead

Findings in Comparison of Operating Systems

1. The C++ implementation shows the expected reaction:
   - Outliers on the vanilla Linux due to scheduler latency
   - Reduced outliers on the Linux RT
   - More mean latency on Linux RT due to scheduling overhead
2. The C++ Actor implementation differs by:
   - Having outliers in the beginning of the Linux RT test probably due to initialisation of the actors
   - Not suffering from the increased latency due to using own scheduling on top of the OS scheduling

Findings in Comparison of Operating Systems

1. The C++ implementation shows the expected reaction:
   - Outliers on the vanilla Linux due to scheduler latency
   - Reduced outliers on the Linux RT
   - More mean latency on Linux RT due to scheduling overhead
2. The C++ Actor implementation differs by:
   - Having outliers in the beginning of the Linux RT test probably due to initialisation of the actors
   - Not suffering from the increased latency due to using own scheduling on top of the OS scheduling
3. Regular patterns for the JVM implementations hint to the Garbage Collector as additional source

JVM Garbage Collection

List of evaluated Garbage Collectors (GC)

Question

How do different Garbage Collectors impact latency?

GCs used for comparison

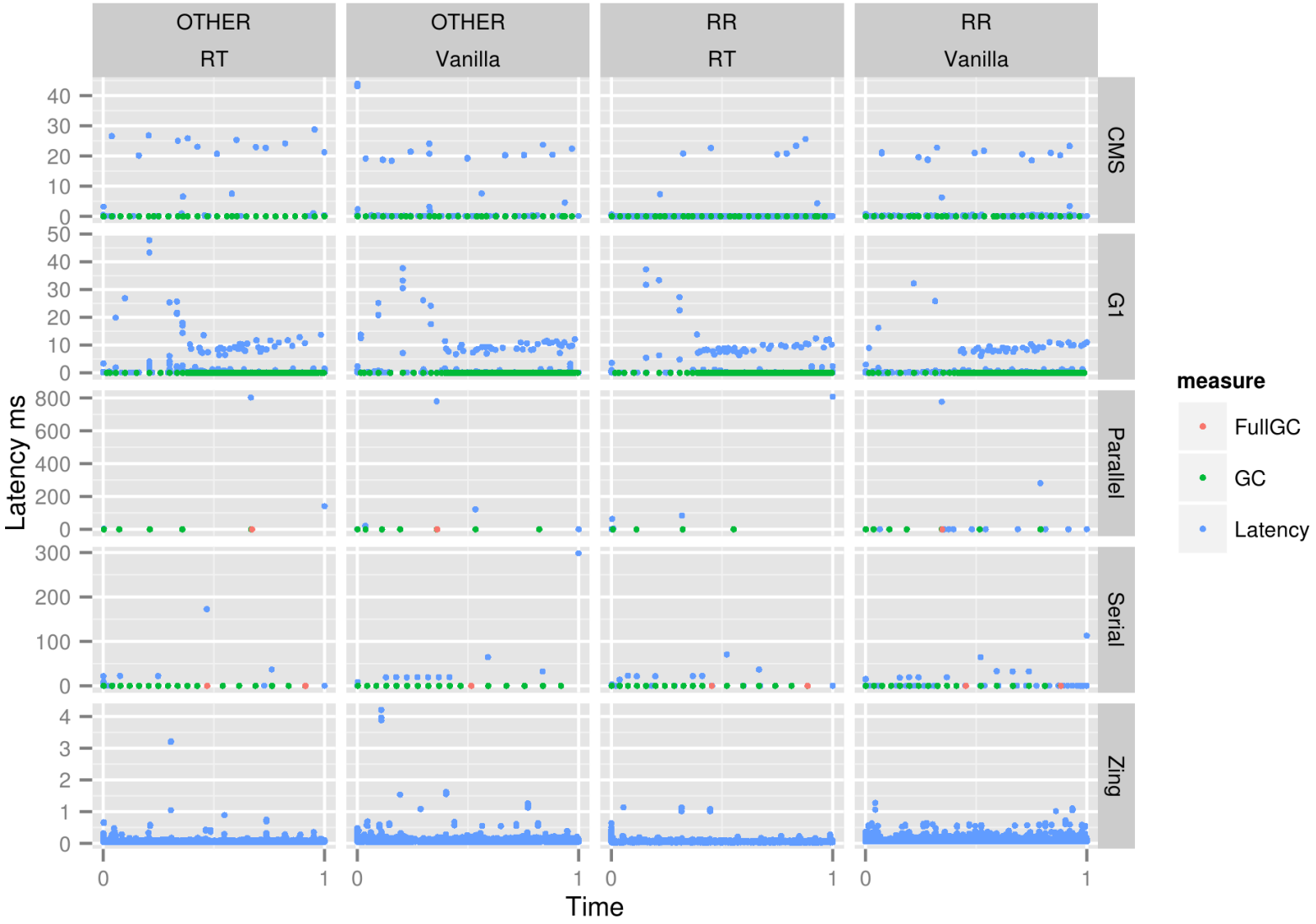
1. Concurrent Mark Sweep (CMS) GC
2. G1 GC
3. Parallel GC
4. Serial GC
5. C4 GC (Zing)
   commercial virtual machine that promises uninterrupted GC

Additional Conditions for JVM GC Benchmarks

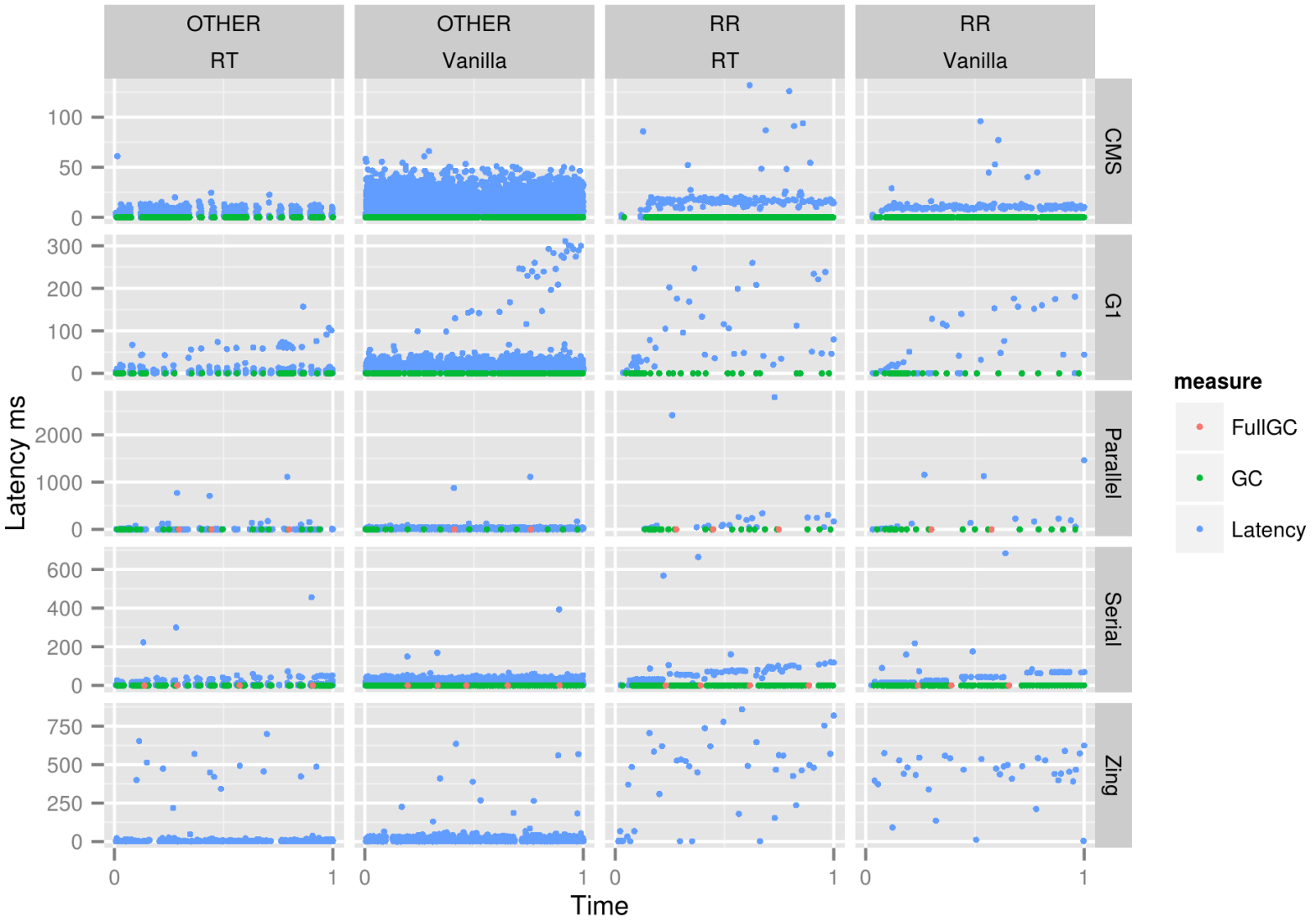
1. For Hotspot's GCs, the times of a GC run are added
   - green $\equiv$ young generation GC
   - red $\equiv$ stop-the-world full GC
2. GC time of Zing VM not measured due to different approach.
3. Measurements were done with `SCHED_OTHER` and `SCHED_RR`.
4. No attempts to reuse objects or circumvent the GC were made.

Results of the GC Tests on Latency

Java Thread Measures



Akka Actor Measures



Max, Mean, and Median of GCs

GC	Scheduler	Max		Mean		Median	
		RT	Vanilla	RT	Vanilla	RT	Vanilla
CMS	SCHED_OTHER	67ms	63.4ms	1.9ms	15.9ms	915.8μs	14.5ms
	SCHED_RR	191ms	212ms	18.3ms	12ms	14.9ms	9.5ms
G1	SCHED_OTHER	241ms	216.8ms	3.4ms	15.3ms	1.1ms	14.1ms
	SCHED_RR	235.9ms	210.6ms	45.1ms	8.8ms	28.9ms	454.8μs
Parallel	SCHED_OTHER	911ms	1110.2ms	3.1ms	17.1ms	1.8ms	14.8ms
	SCHED_RR	3756.6ms	192ms	212.7ms	46.3ms	62.3ms	13.8ms
Serial	SCHED_OTHER	631.8ms	385.1ms	4.2ms	16.4ms	1.9ms	14.9ms
	SCHED_RR	1423.5ms	717.4ms	90.8ms	42.6ms	58.6ms	24.4ms
Zing	SCHED_OTHER	744.3ms	387.5ms	11.2ms	13.7ms	2.7ms	11.4ms
	SCHED_RR	720.1ms	655.6ms	137.7ms	126.6ms	4.8ms	4ms

(2) Comparison values for latencies for the Scala/Akka implementation running with different GCs. The tests were conducted with and without RT patch with 16 actor pairs

Findings in Comparison of GCs

1. Latency spikes due to the JVM and its GC are much higher than OS scheduler induced latency.
2. GCs show unique patterns of latency spikes.
3. Linux RT influenced the Scala/Akka implementation with the default scheduler but not the Real-Time Scheduler.

Findings in Comparison of GCs

1. Latency spikes due to the JVM and its GC are much higher than OS scheduler induced latency.
2. GCs show unique patterns of latency spikes.
3. Linux RT influenced the Scala/Akka implementation with the default scheduler but not the Real-Time Scheduler.

Note

The Scala/Akka test had implementation issues and is therefore only comparable to itself but not to the Java version.

Conclusion

Conclusion

- The JVM shows high latency spikes while having a good mean latency

Conclusion

- The JVM shows high latency spikes while having a good mean latency
- Garbage Collection is not the only source but a dominant one

Conclusion

- The JVM shows high latency spikes while having a good mean latency
- Garbage Collection is not the only source but a dominant one
- Garbage Collection has to be tuned for each application

Conclusion

- The JVM shows high latency spikes while having a good mean latency
- Garbage Collection is not the only source but a dominant one
- Garbage Collection has to be tuned for each application
- Each software layer introduces sources for latency

Conclusion

- The JVM shows high latency spikes while having a good mean latency
- Garbage Collection is not the only source but a dominant one
- Garbage Collection has to be tuned for each application
- Each software layer introduces sources for latency
- Latency behaviour is application dependent

Conclusion

- The JVM shows high latency spikes while having a good mean latency
- Garbage Collection is not the only source but a dominant one
- Garbage Collection has to be tuned for each application
- Each software layer introduces sources for latency
- Latency behaviour is application dependent
- Actor systems scale better with increased parallelism

Conclusion

- The JVM shows high latency spikes while having a good mean latency
- Garbage Collection is not the only source but a dominant one
- Garbage Collection has to be tuned for each application
- Each software layer introduces sources for latency
- Latency behaviour is application dependent
- Actor systems scale better with increased parallelism
- Linux RT reduces latency outliers

Conclusion

- The JVM shows high latency spikes while having a good mean latency
- Garbage Collection is not the only source but a dominant one
- Garbage Collection has to be tuned for each application
- Each software layer introduces sources for latency
- Latency behaviour is application dependent
- Actor systems scale better with increased parallelism
- Linux RT reduces latency outliers
- Linux RT increases mean latency

Future work

- Evaluate different communication patterns.
- Evaluate more sources for application stage latency jitter.
- Evaluate effects of latency jitter on humans using VR w.r.t. simulator sickness.
- Evaluate effects of longer test runs.

SEARIS Workshop

Reducing Application-Stage Latencies For Real-Time Interactive Systems

Jan-Philipp Stauffert, Florian Niebling, Marc Erich Latoschik

Informatik IX, Mensch-Computer-Interaktion
Universität Würzburg

April 11, 2016

Bishop, G., Fuchs, H., McMillan, L., and Zagier, E. J. S. (1994).
Frameless rendering: Double buffering considered harmful.
In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 175–176. ACM.

Charousset, D., Hiesgen, R., and Schmidt, T. C. (2014).
CAF - the C++ Actor Framework for Scalable and Resource-Efficient Applications.
pages 15–28. ACM Press.

Dayal, A., Woolley, C., Watson, B., and Luebke, D. (2005).
Adaptive frameless rendering.
In *ACM SIGGRAPH 2005 Courses*, page 24. ACM.

Frank, L. H., Casali, J. G., and Wierwille, W. W. (1988).
Effects of visual display and motion system delays on operator performance and uneasiness in a driving simulator.
Human Factors: The Journal of the Human Factors and Ergonomics Society, 30(2):201–217.

Ivkovic, Z., Stavness, I., Gutwin, C., and Sutcliffe, S. (2015).
Quantifying and Mitigating the Negative Effects of Local Latencies on Aiming in 3d Shooter Games.
pages 135–144. ACM Press.

McMillan, L. and Bishop, G. (1995).
Head-tracked stereoscopic display using image warping.
In *IS&T/SPIE's Symposium on Electronic Imaging: Science & Technology*, pages 21–30. International Society for Optics and Photonics.

Mine, M. (1993).
Characterization of end-to-end delays in head-mounted display systems.
The University of North Carolina at Chapel Hill, TR93-001.

Teather, R. J., Pavlovych, A., Stuerzlinger, W., and MacKenzie, S. I. (2009).

Effects of tracking technology, latency, and spatial jitter on object movement.
In *3D User Interfaces, 2009. 3DUI 2009. IEEE Symposium on*, pages 43–50. IEEE.