E.ON Energy Research Center

Migration Techniques in HPC Environments

In Conjunction with the FAST Project

08/26/2014

Simon Pickartz[1], Ramy Gad[2], Stefan Lankes[1], Lars Nagel[2], Tim Süß[2], André Brinkmann[2], and Stephan Krempel[3]

[1]RWTH Aachen University [2]Johannes Gutenberg Universität

[3]ParTec Cluster Competence Center GmbH

FAST FOR EXASCALE

RWTH AACHEN UNIVERSITY

Agenda

- The FAST Project
- Migration in HPC Environments
  - Process-level
  - Virtualization
  - Container-based
- Evaluation
  - Overhead
  - Migration Time
- Conclusion

The FAST Project

Exascale Systems

- Characteristics/Assumptions
  - Increasing amount of cores per node
  - Increase of CPU performance will not be matched by I/O performance

→ Most applications cannot exploit this parallelism

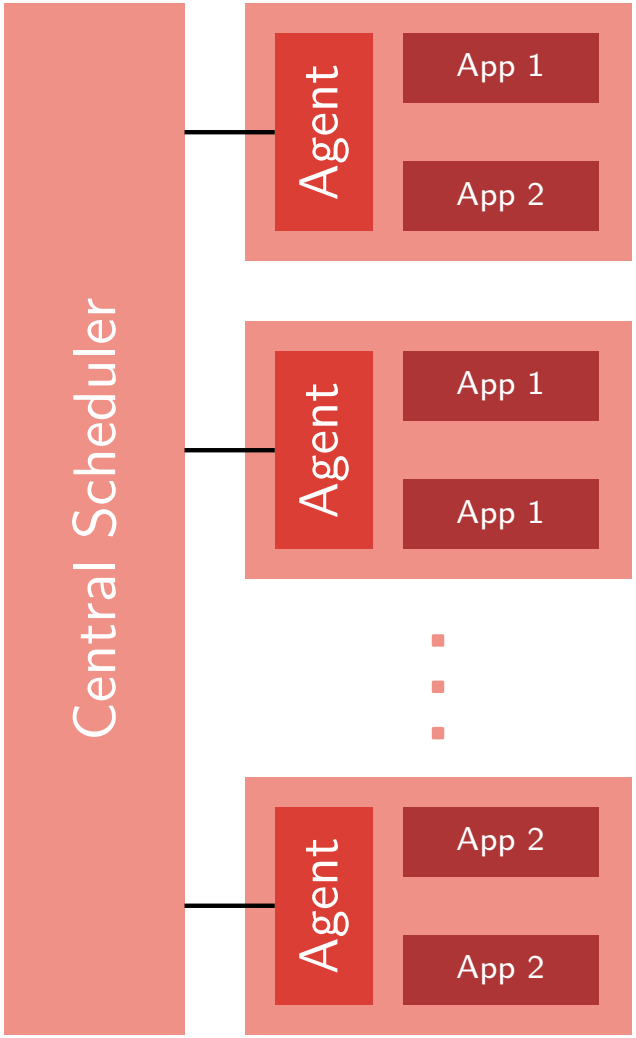
- Consequences
  - Exclusive node assignment has to be revoked
  - Dynamic scheduling during runtime will be necessary

→ Migration of processes between nodes indispensable

Find a Suitable Topology for Exascale Applications

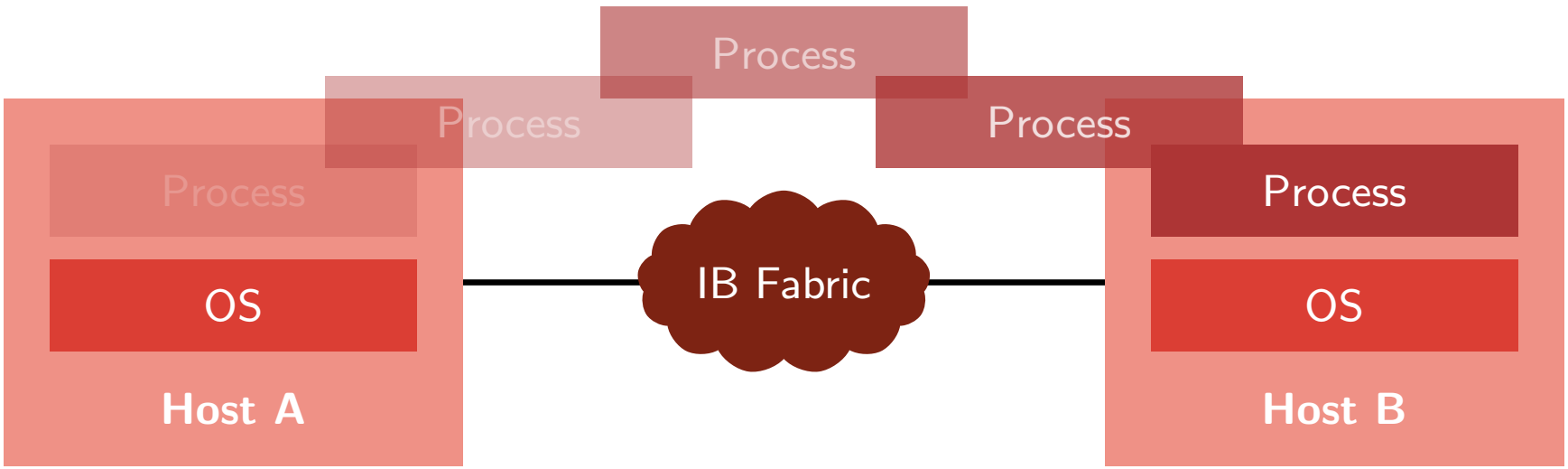
A twofold scheduling approach

1. Initial placement of job by a global scheduler according to KPIs
   - Load of the individual nodes
   - Power consumption
   - Applications' resource requirements
2. Dynamic runtime adjustments
   - Migration of processes
   - Tight coupling with the applications
   - Feedback to the global scheduler



Migration in HPC Environments

Process-level Migration



Process-level Migration

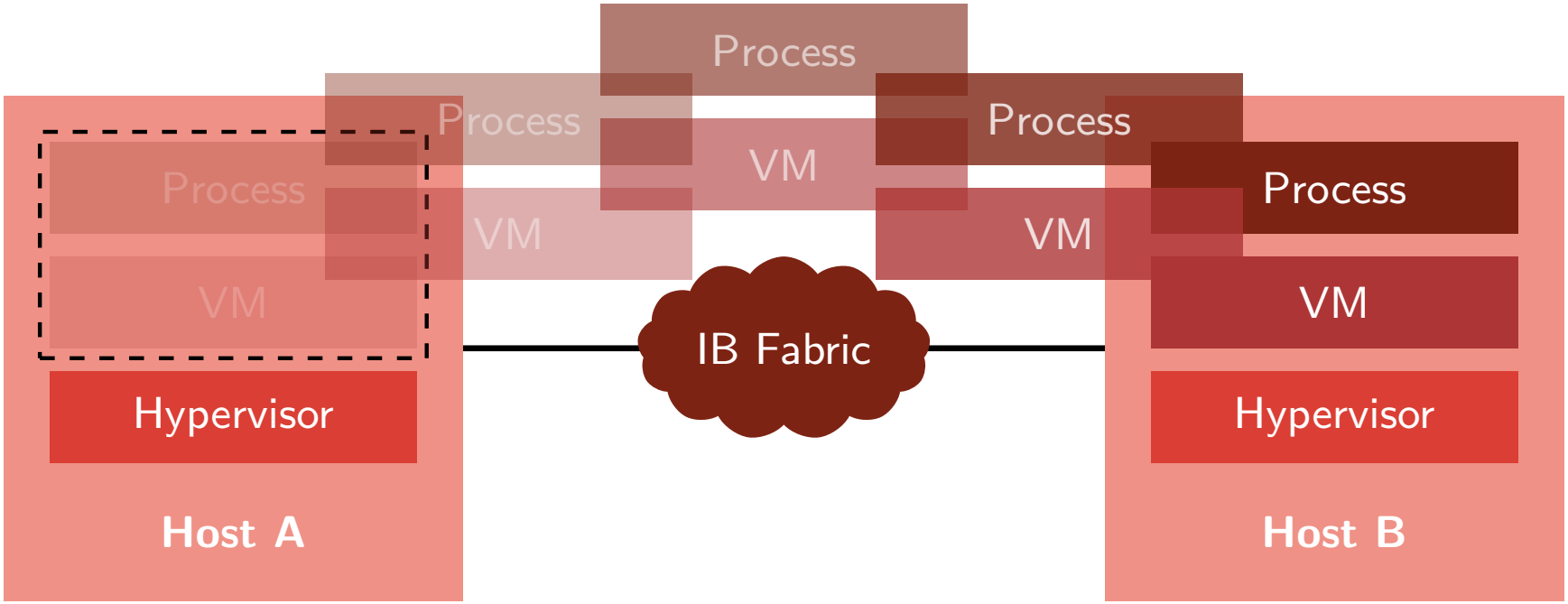
- Move a process from one node to another
  - Including its execution context
    (i. e., register state and physical memory)
- A special kind of Checkpoint/Restart (C/R) operation
- Several frameworks available
  - Condor's checkpoint library
  - libckpt
  - Berkley Lab Checkpoint/Restart (BLCR)

Berkley Lab Checkpoint/Restart

- Specifically designed for HPC applications
- Two components
  - Kernel module for performing the C/R operation
  - Shared-library enabling user-space access
- Cooperation with the C/R procedure via callback interface
  - Close/open file descriptors
  - Tear down/reestablish communication channels
  - ...

→ Residual dependencies that have to be resolved!

Virtual Machine Migration



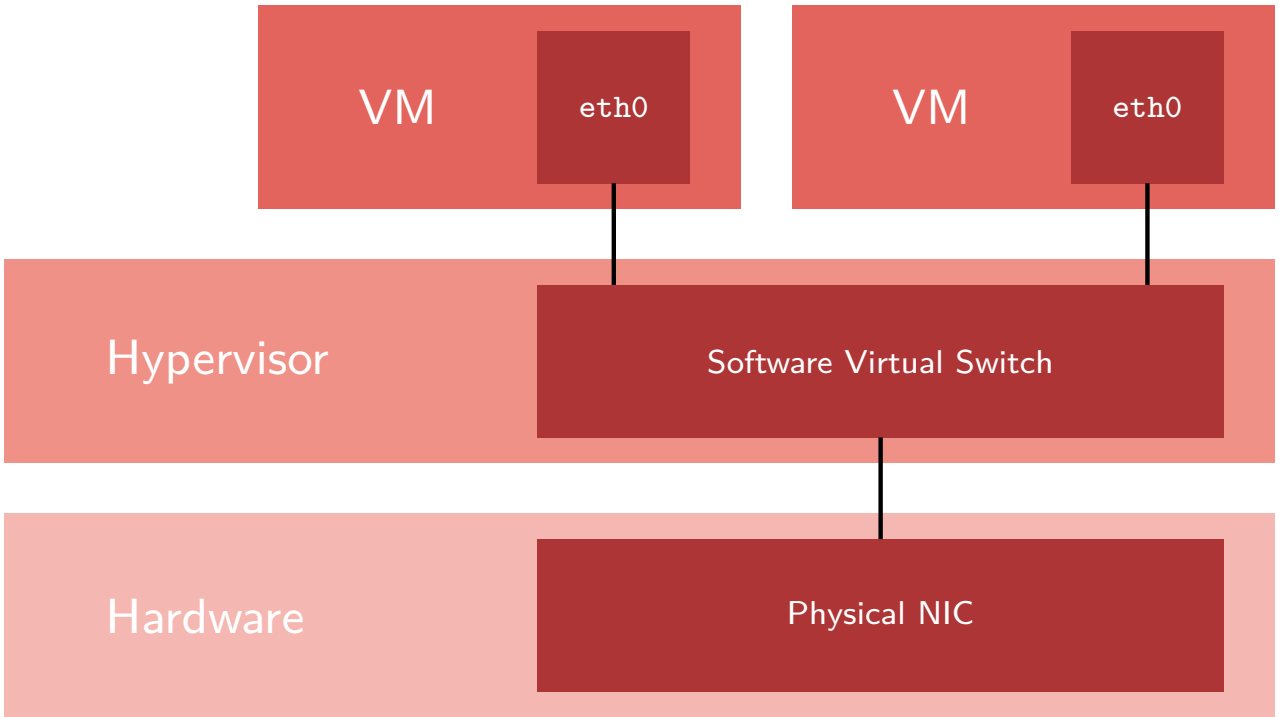
Virtual Machine Migration

- Migration of a virtualized execution environment
- Reduction of the residual dependencies
  - File descriptors still valid after the migration
  - Communication channels are automatically restored (e. g. TCP)
- Performance degradation of I/O devices can be compensated by
  - Pass-through (e. g. Intel VT-d)
  - Single Root I/O virtualization (SR-IOV)
- Various hypervisors available
  - Xen
  - Kernel Based Virtual Machine (KVM)
  - ...

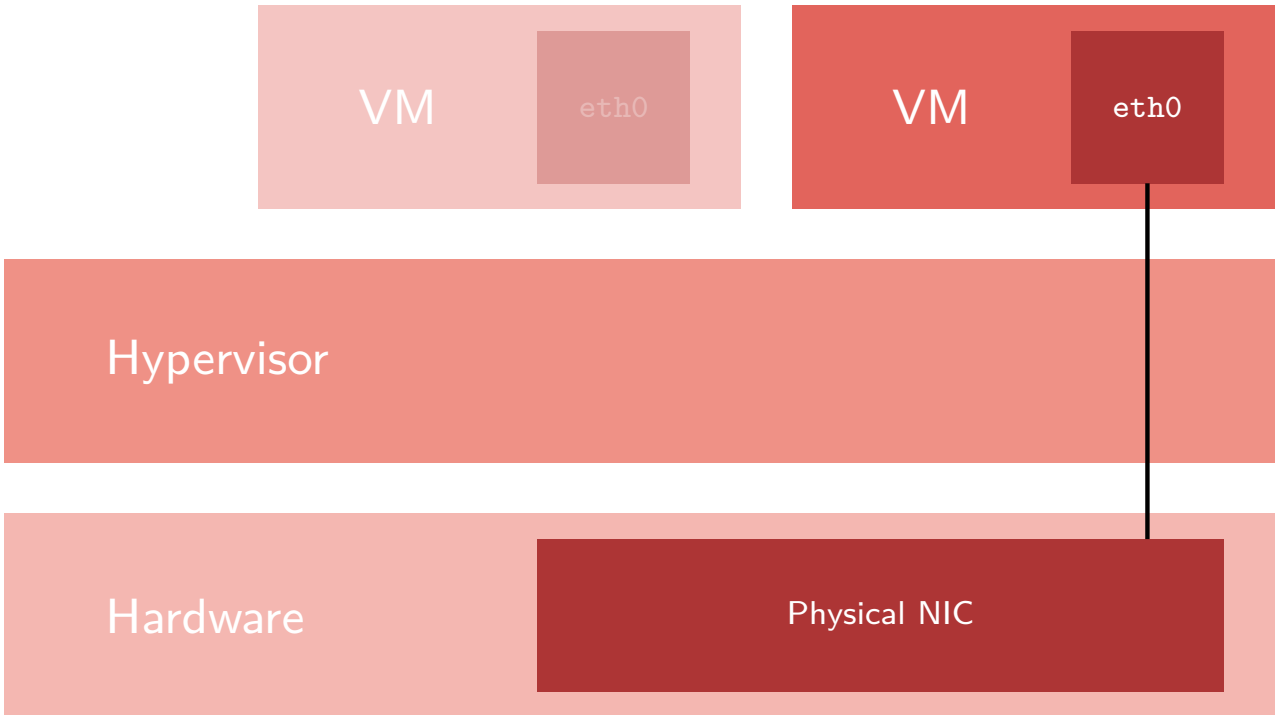
Kernel Based Virtual Machine

- A Linux kernel module that benefits from existing resources
  - Scheduler
  - Memory management
  - ...
- Implements full-virtualization requiring hardware support
- VMs are scheduled by the host like any other process
- Already provides a migration framework
  - Live-migration
  - Cold-migration

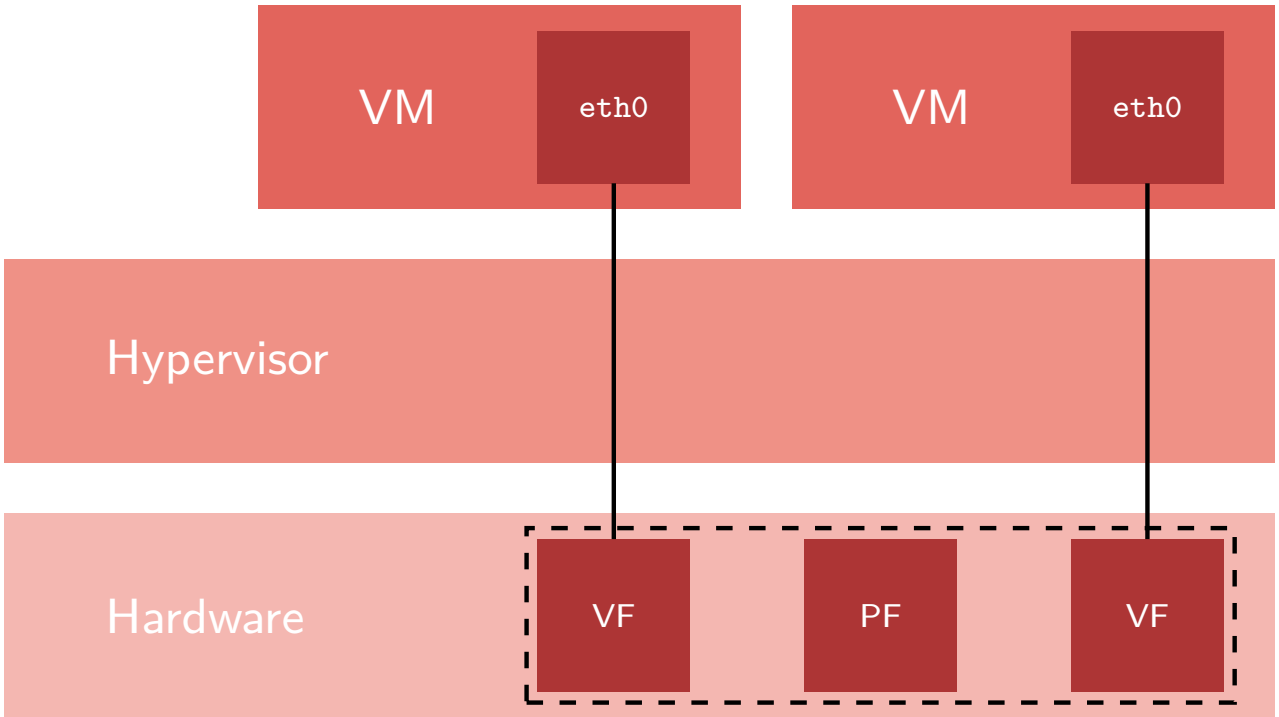
Software-based Sharing



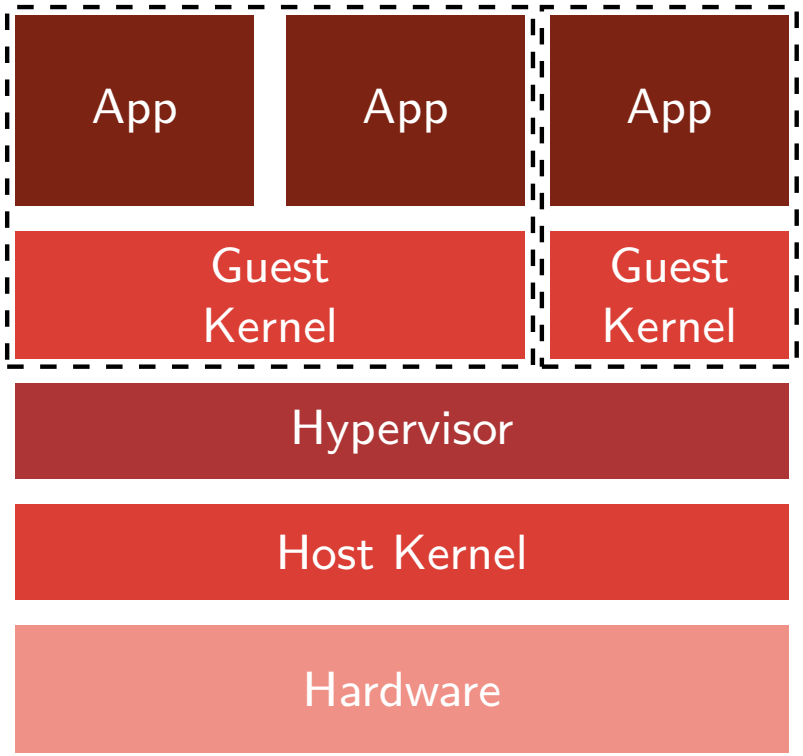
Pass-Through



Single Root I/O Virtualization

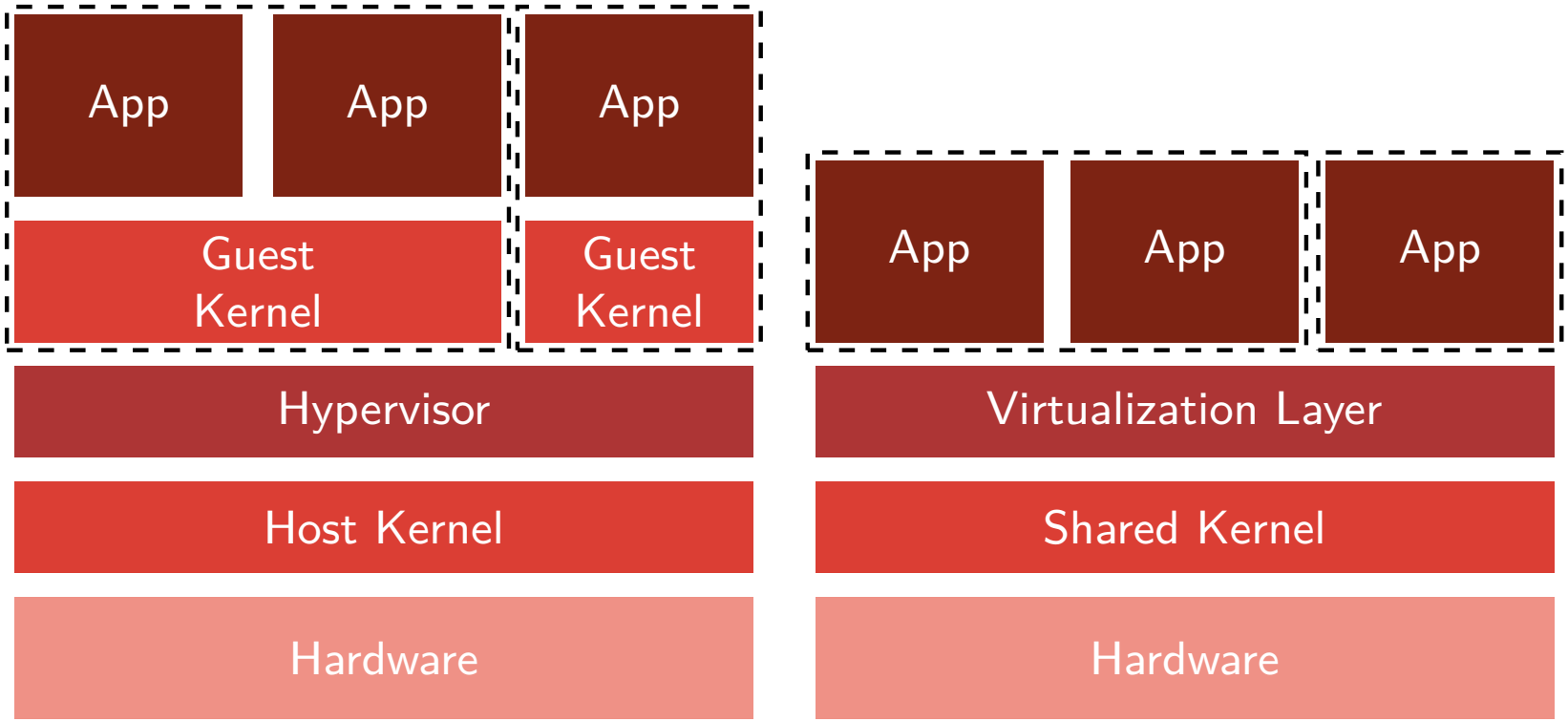


Container-based Virtualization



(a) Virtual Machines

Container-based Virtualization



(a) Virtual Machines

(b) Containers

Container-based Migration

- Full-virtualization results in *multiple* kernels on one node
- Idea of container-based virtualization

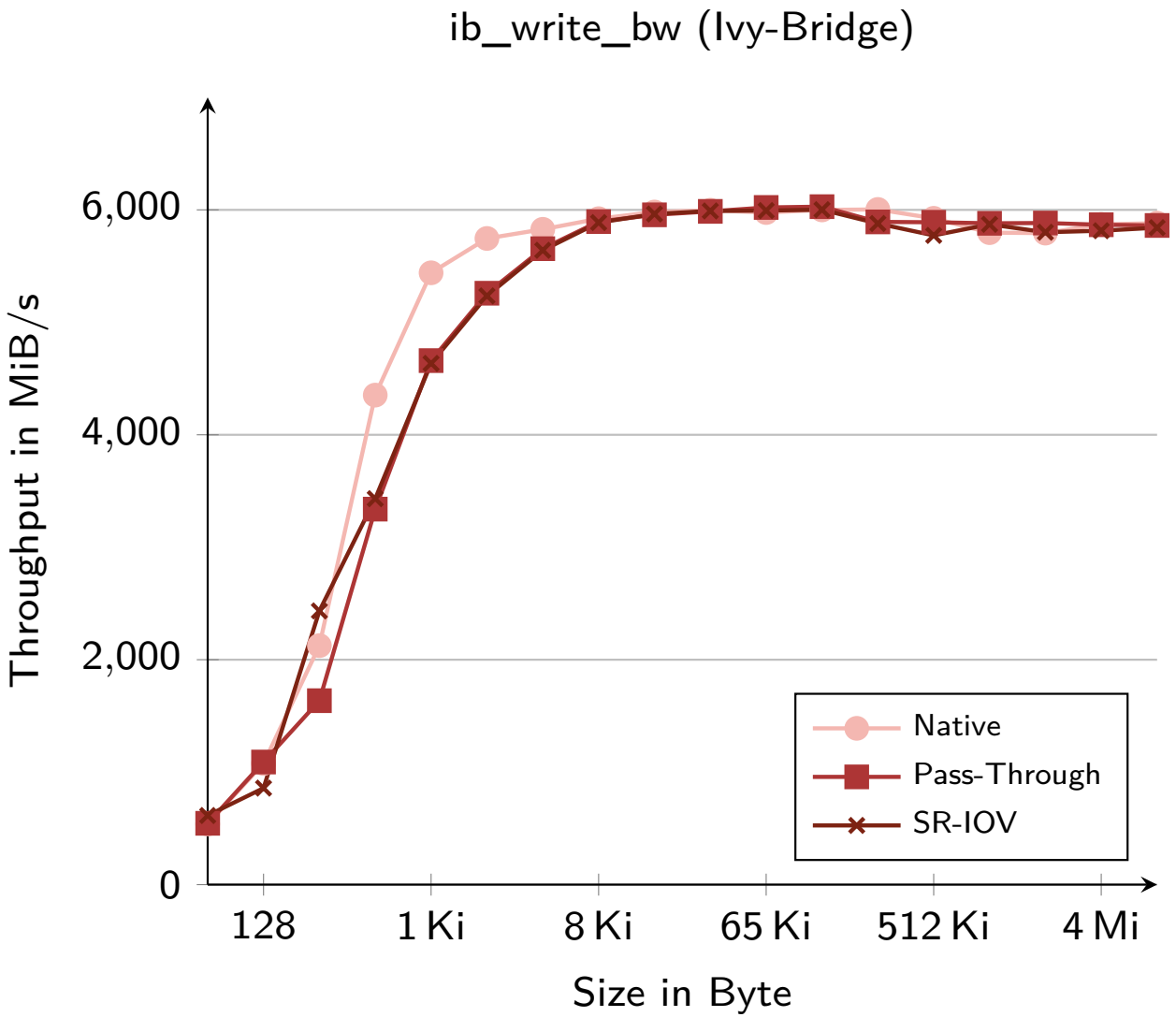
  → Reuse the existing host kernel for the management of multiple user-space instances
- Host and guest have to use the same operating system
- Common representatives
  - OpenVZ
  - LinuX Containers (LXC)

Evaluation

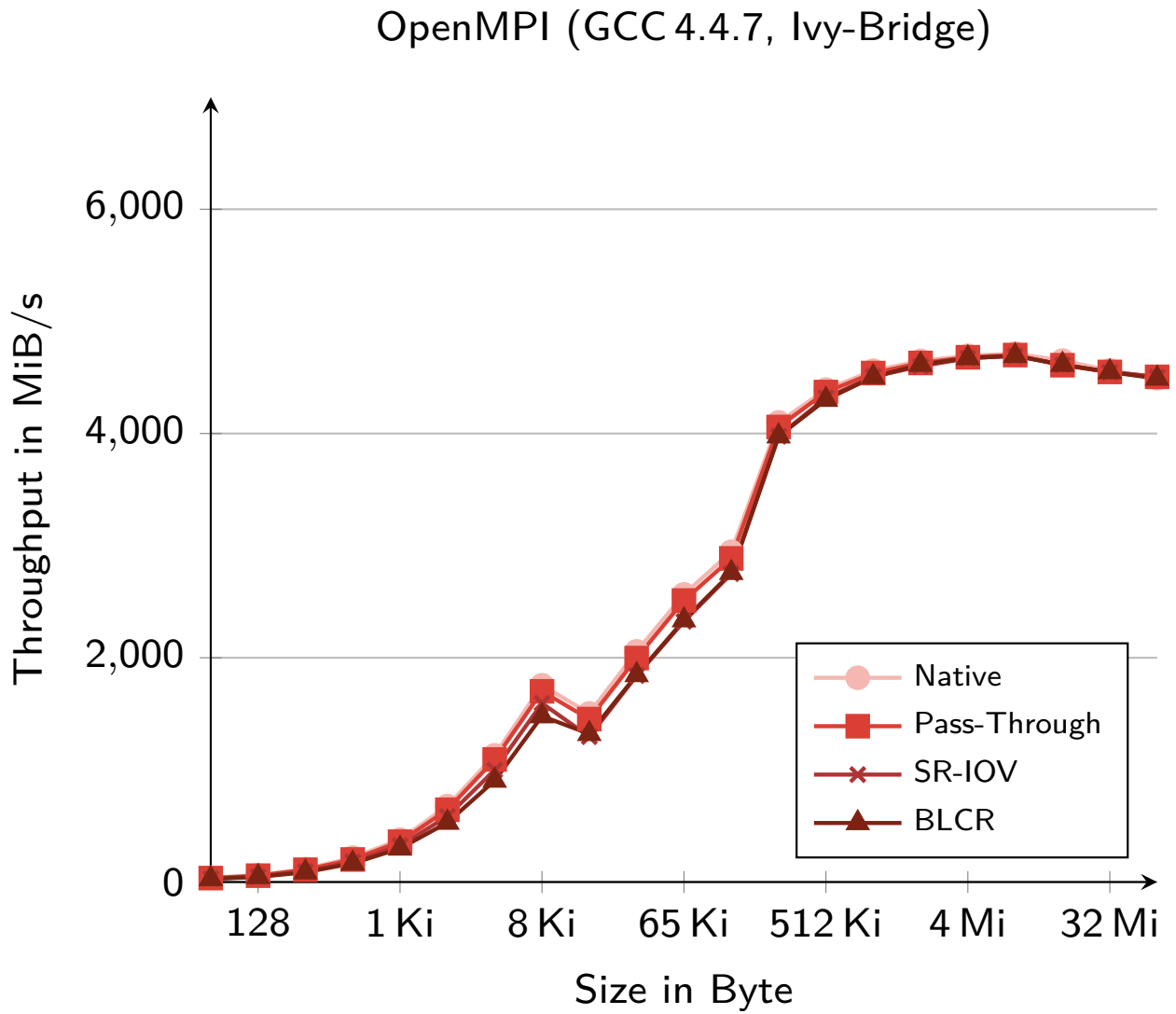
Test Environment

- 4-node Cluster
  - 2 Sandy-bridge Systems
  - 2 Ivy-bridge Systems
- InfiniBand FDR Mellanox Fabric
  - Up to 56 GiB/s
  - Support for SR-IOV
- OpenMPI 1.7 with BLCR support (except for the LXC results)

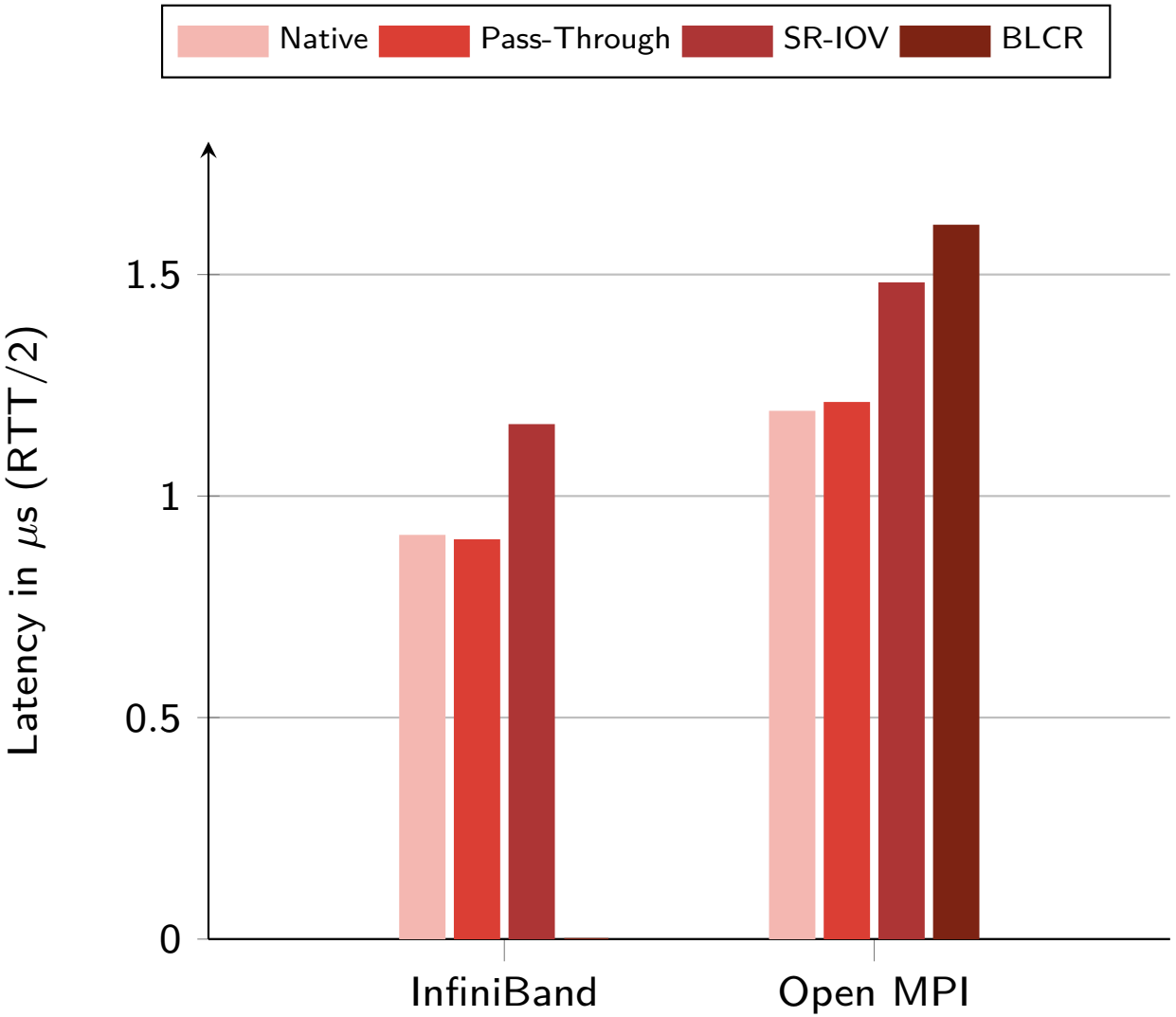
Throughput


ib_write_bw (Ivy-Bridge)
Throughput in MiB/s
6,000
4,000
2,000
0
128
1 Ki
8 Ki
65 Ki
512 Ki
4 Mi
Size in Byte
Native
Pass-Through
SR-IOV

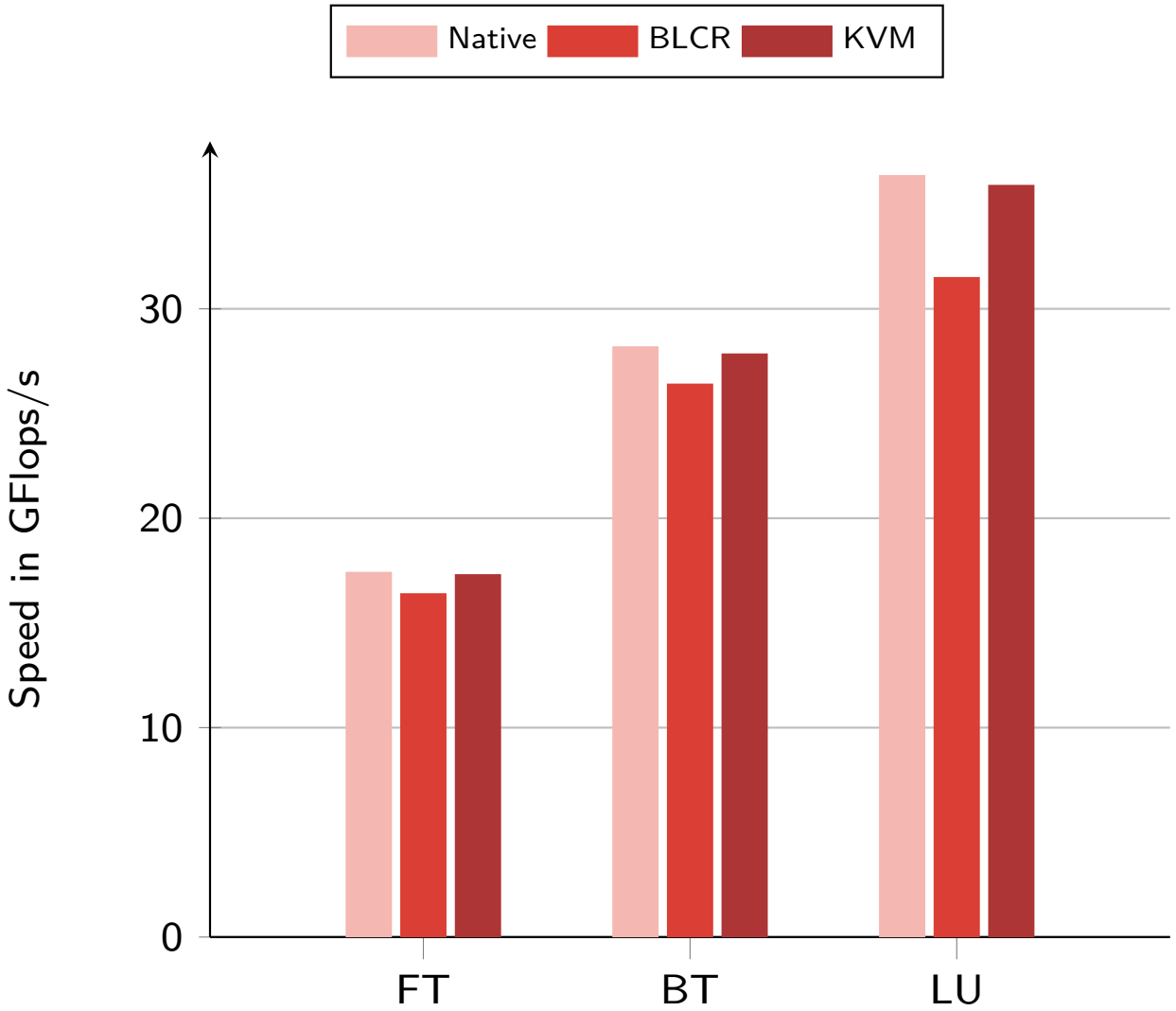
Throughput


OpenMPI (GCC 4.4.7, Ivy-Bridge)
Throughput in MiB/s
6,000
4,000
2,000
0
128
1 Ki
8 Ki
65 Ki
512 Ki
4 Mi
32 Mi
Size in Byte
Native
Pass-Through
SR-IOV
BLCR

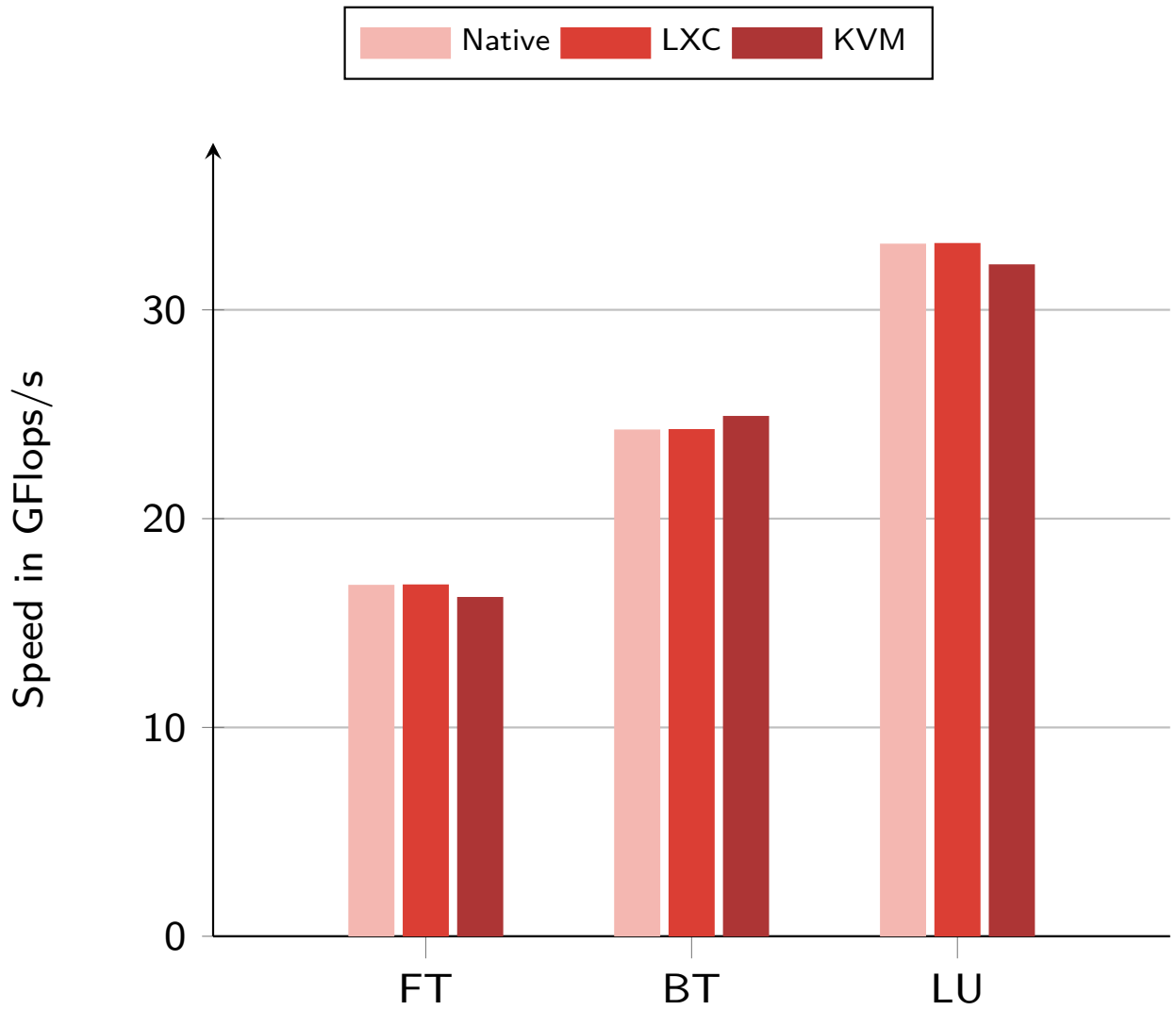
Latency


Native
Pass-Through
SR-IOV
BLCR
Latency in μs (RTT/2)
1.5
1
0.5
0
InfiniBand
Open MPI

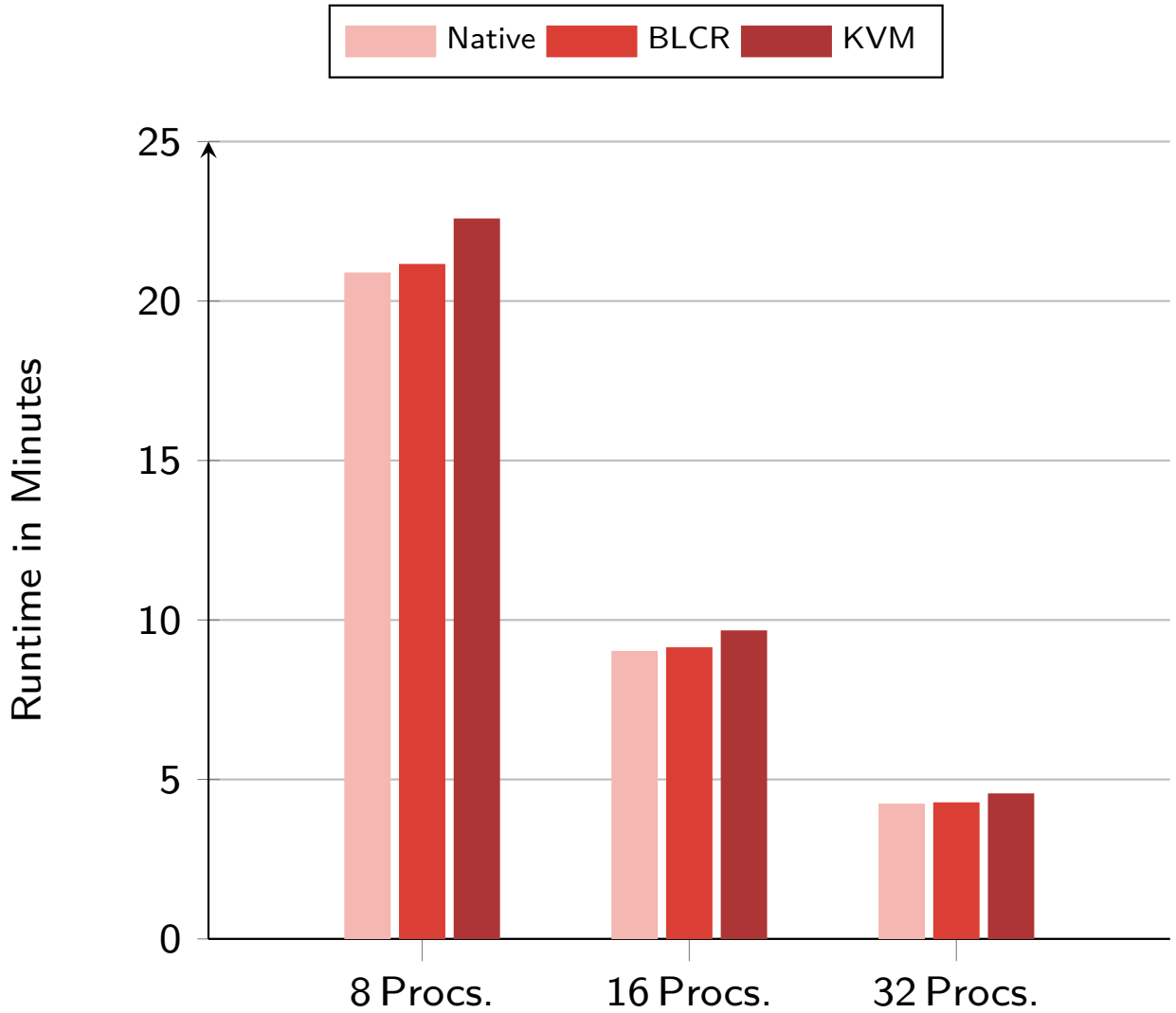
NAS Parallel Benchmarks (Open MPI)



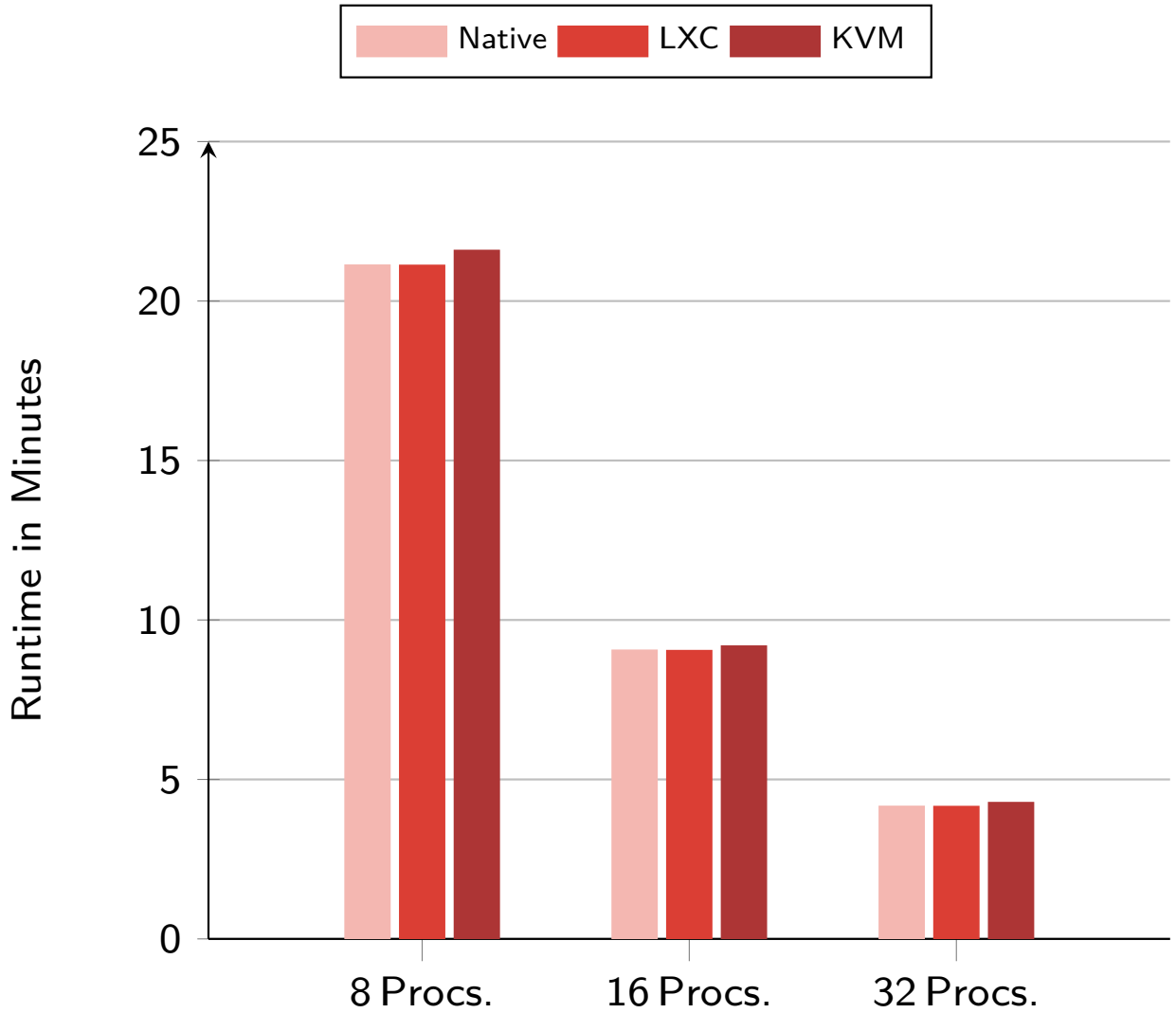
NAS Parallel Benchmarks (Parastation)



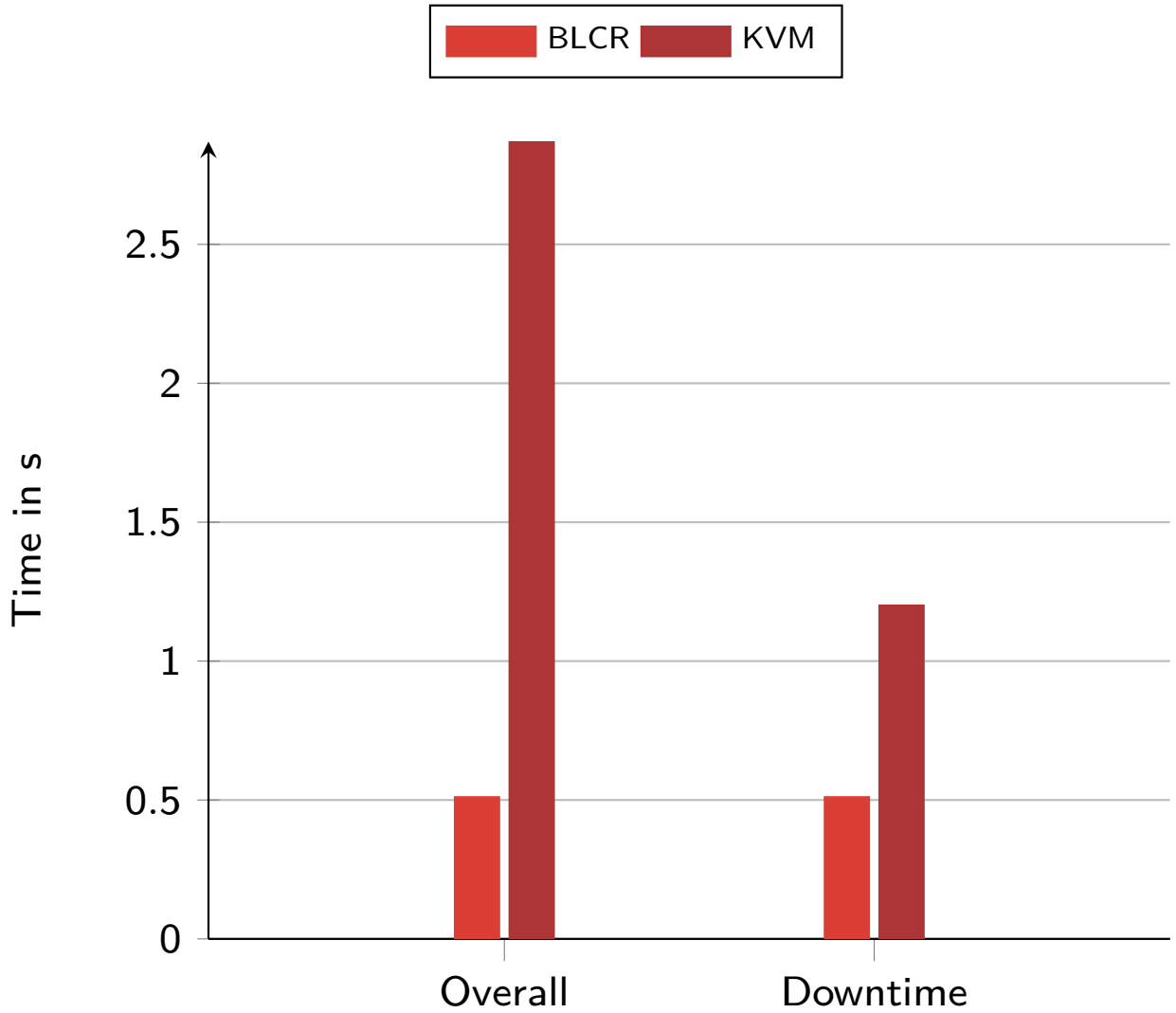
mpiBLAST (Open MPI)



mpiBLAST (Parastation)



Migration Time



Conclusion

Conclusion

- Inconclusive microbenchmarks analysis
- Overhead is highly application dependent
- Migration
  - Significant overhead of KVM concerning overall migration time
  - Qualified by the downtime test
  - KVM already supports live-migration
  - BLCR requires *all* processes to be stopped during migration
- Flexibility
  - Process-level migration generates residual dependencies

    → A non-transparent approach would be required
  - VM/Container-based migration reduces these dependencies

Outlook

- We will focus on VM migration within FAST
- Containers may provide even better performance
  - Not as flexible as VMs (e. g., same OS)
  - More isolation than process-level migration
- Investigation of the application dependency observed during our studies
- Enable VM migration with attached pass-through devices

→ More about FAST on http://www.en.fast-project.de

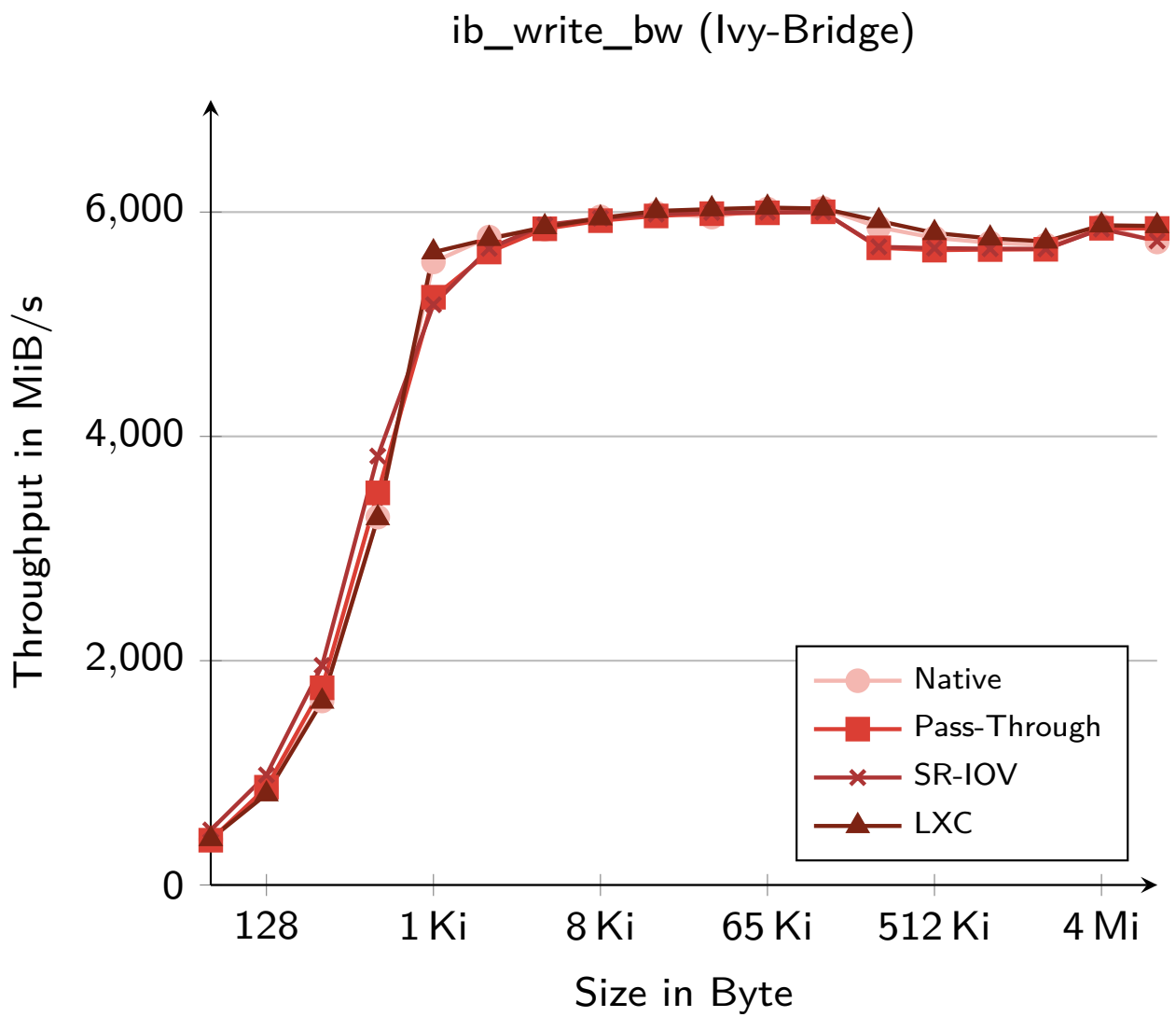
Thank you for your kind attention!

Simon Pickartz[1], **Ramy Gad**[2], **Stefan Lankes**[1], **Lars Nagel**[2],
Tim Süß[2], **André Brinkmann**[2], **and Stephan Krempel**[3]

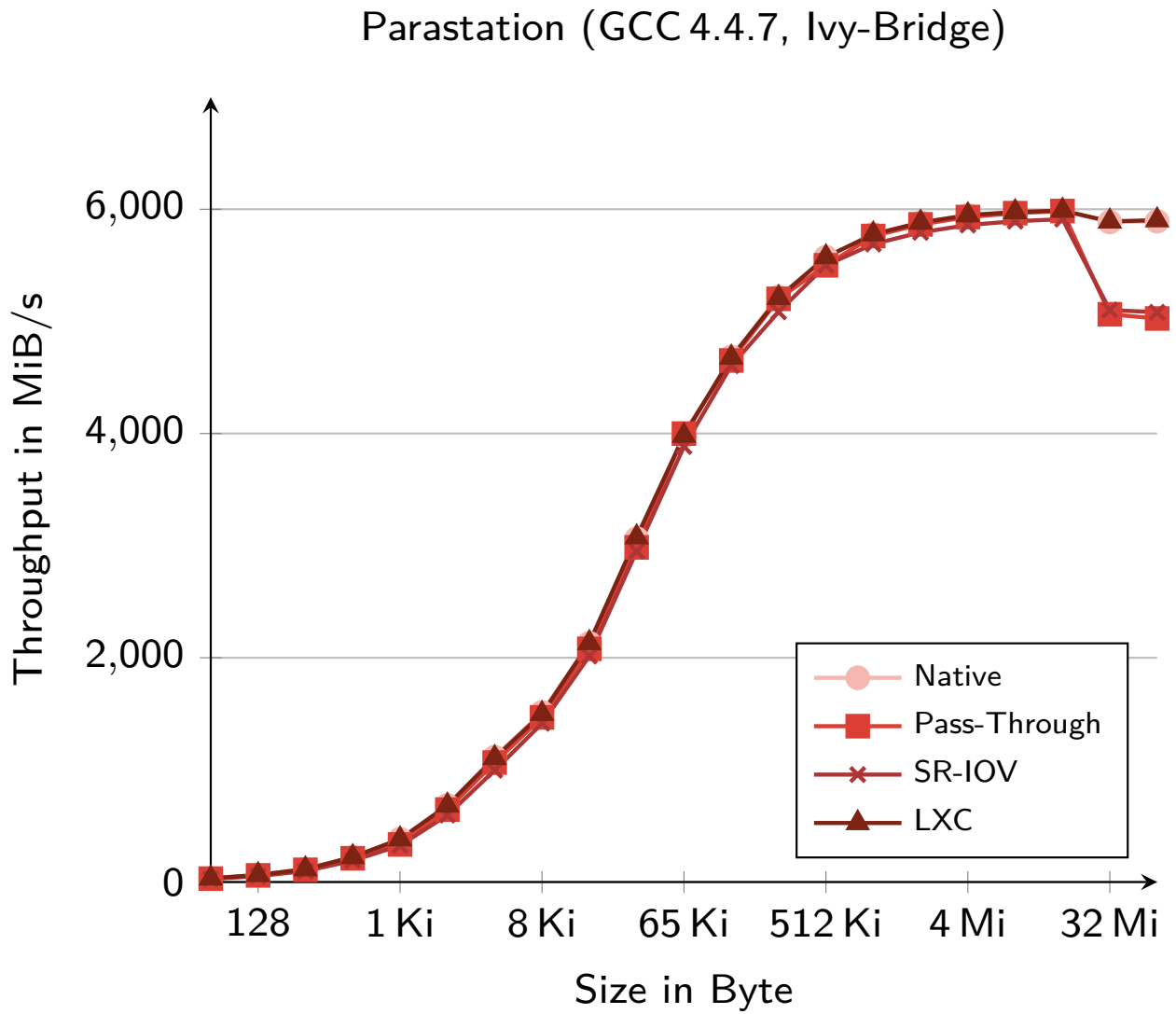
[1]**RWTH Aachen University** [2]**Johannes Gutenberg Universität**
[3]**ParTec Cluster Competence Center GmbH** – spickartz@eonerc.rwth-aachen.de

Institute for Automation of Complex Power Systems
E.ON Energy Research Center, RWTH Aachen University
Mathieustraße 10
52074 Aachen

Throughput (Fedora 20)



Throughput (Fedora 20)



Latency (Fedora 20)