Virtual Open Systems

Lightweight and Generic RDMA Engine ParaVirtualization for the KVM Hypervisor

Angelos Mouzakitis[1], Christian Pinto[1], Nikolay Nikolaev[1], Alvise Rigo[1], Daniel Raho[1], Babis Aronis[2], Manolis Marazakis[2]

[1] Virtual Open Systems, Grenoble, France
[2] Foundation for Research and Technology – Hellas (FORTH), Heraklion, Greece
(*) The work was supported by the *ExaNeSt* project. (grant agreement No. 671553)

Presentation overview

- Motivation
- The ExaNeSt project
  - Hardware platform
  - User-space API
- RDMA virtualization with API Remoting
  - Split of the RDMA API
  - Shared memory implementation
  - Guest zero-copy access to DMA buffers
  - Completion notification event forwarding
- Experimental results
- Conclusions

Motivation

- Virtualization in HPC
- Network stack in virtualized environments
- State of the Art
  - SR-IOV
  - Direct pass-through
- Why para-virtualization ?
  - Device sharing
  - Fine grain QoS
  - Integrated system architecture
    - Capabilities / limitations

The ExaNeSt project

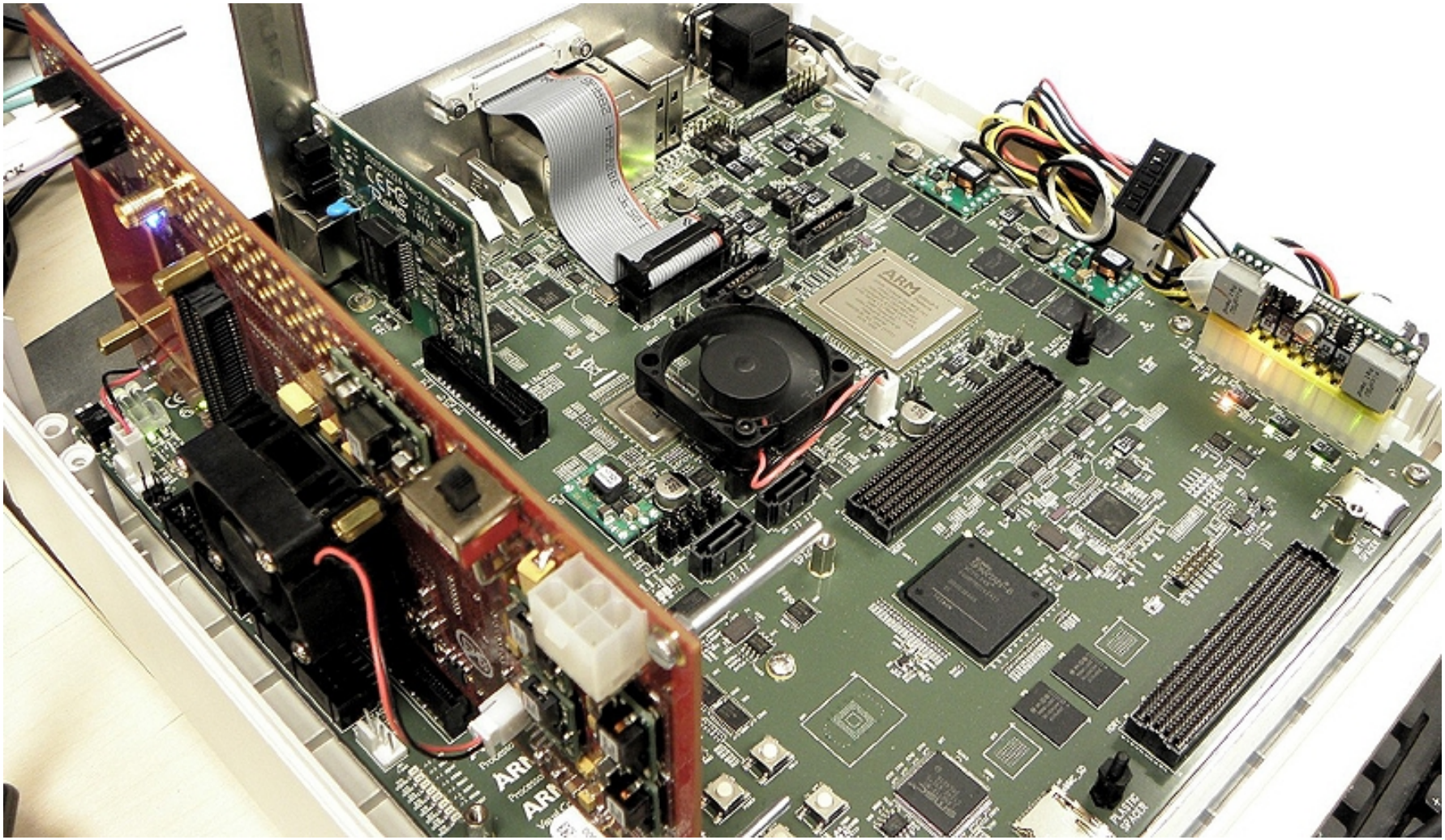
- Hardware description of the prototype
- 6-node cluster with ARMv8 Juno r2 boards
  - 2x ARM Cortex-A72 and 4-core Cortex-A53
  - Linux kernel 4.3.0
  - UNIMEM[1] loaded on the FPGA
- RDMA device
  - Implemented on the FPGA
  - Xilinx CDMA[2] as the core IP
    - Physical contiguous DMA transfers
  - Interconnected through PCI-e over an FPGA switch

The ExaNeSt project

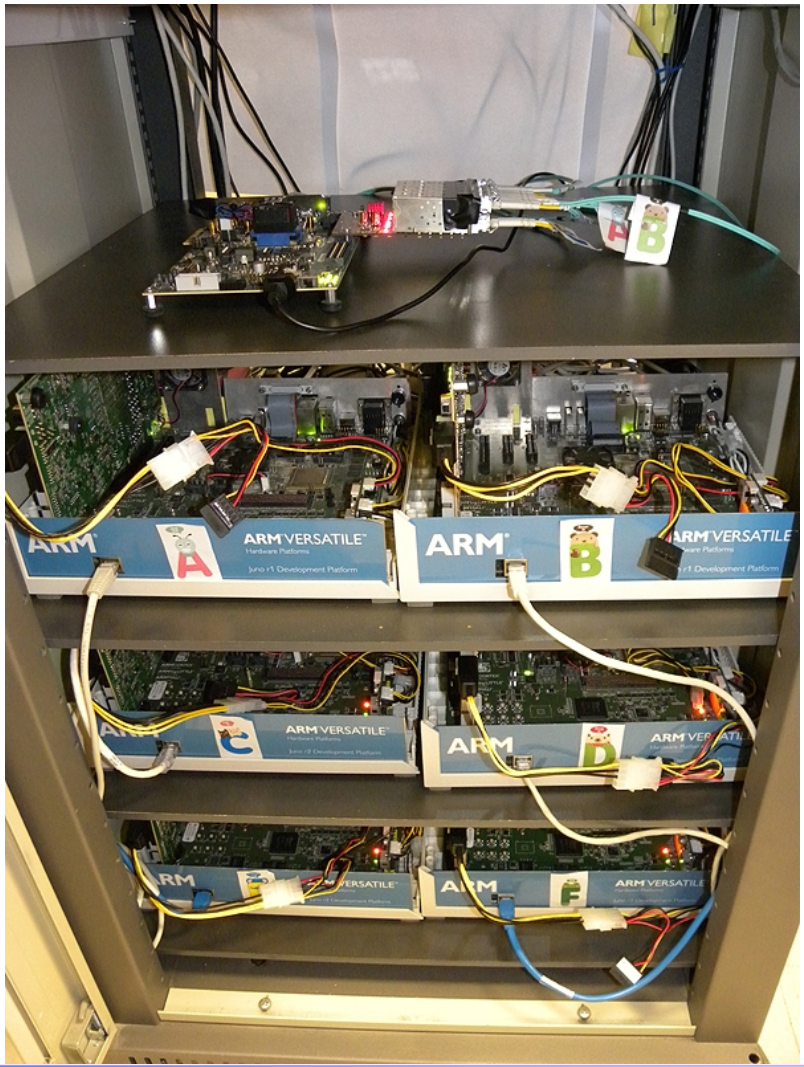
RDMA user-space API

- Custom non-blocking API
  - Not compliant with the OFED-libibverbs API
- User-space API
  - Kernel involvement only for initialization
- No memory-resident pages required
- DMA transfers
  - Polling mode
  - Interrupt mode – asynchronous notification via signal

The ExaNeSt project – ARM Juno r2



The ExaNeSt project - 6 nodes cluster



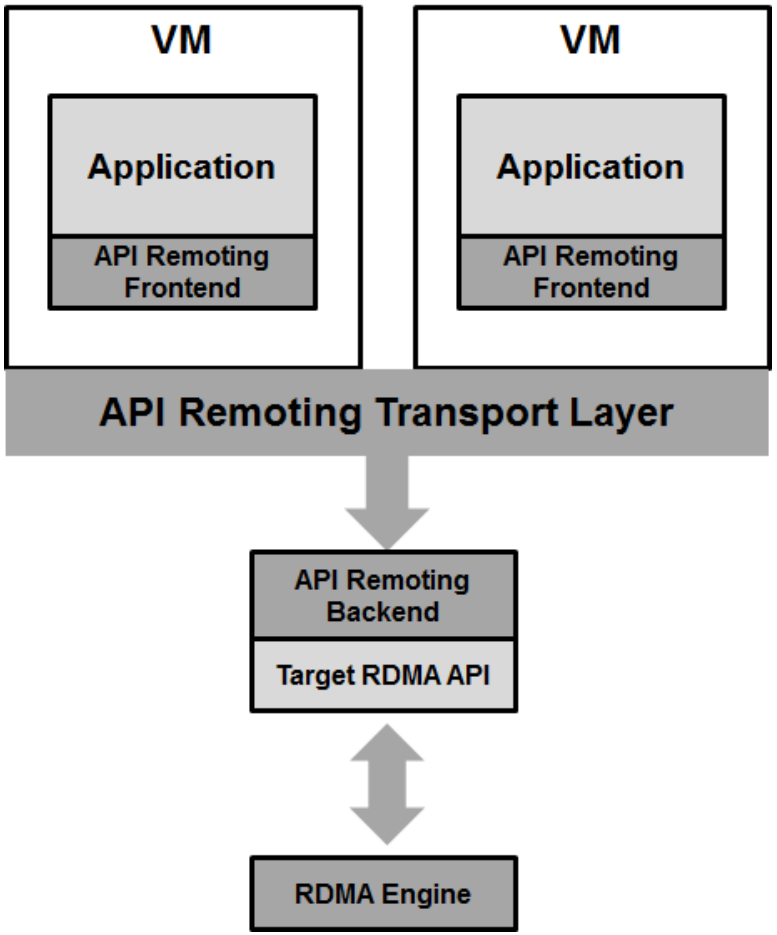
Virtualizing the API

RDMA operations split to Control and Data paths

- Control path
  - Transfer initiation, transfer status, DMA buffer allocation
  - Are simple / cheap to copy
- Data path
  - *The problem:*
    - Moving data to DMA buffers costs in performance
    - The DMA engine requires contiguous physical buffers
  - *Proposed solution:*
    - Re-map host physical memory to guest virtual address space
    - Access data direct to the DMA buffer

API Remoting - Architecture

- Frontend
  - Guest dynamic library used by the application
  - Implements the target API
  - Forwards the requests to the Backend for handling
- Backend
  - Handles guest requests
  - Uses the ‘real’ library to access the hardware
  - Replays return values and errors
- Transport layer
  - Backend and Frontend communication
  - Implementation defined, shared memory in our case

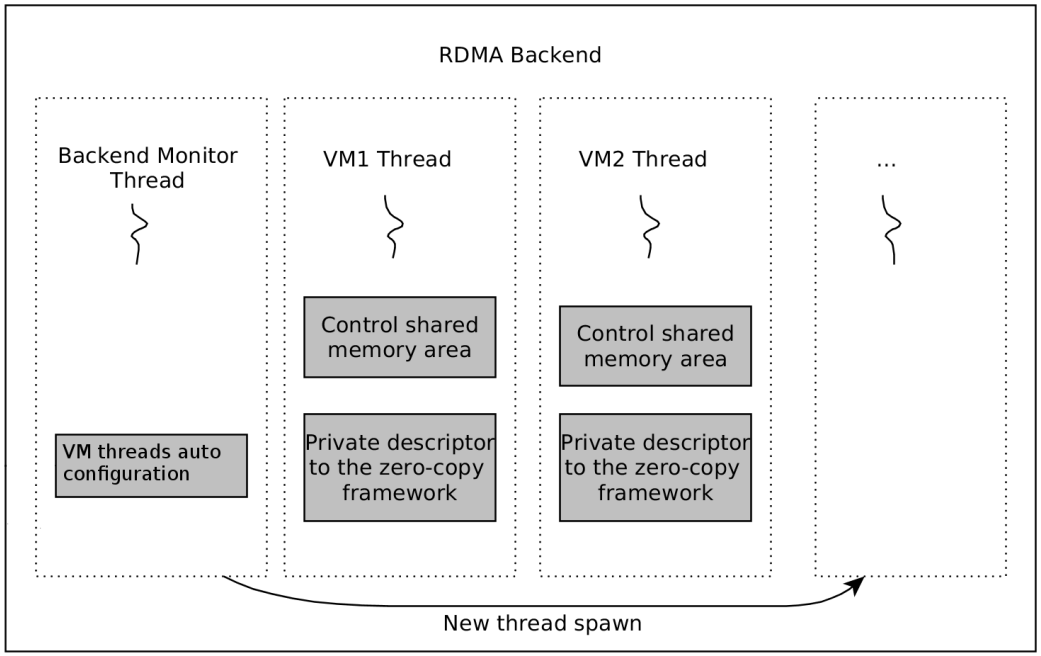


API Remoting – Transport layer

- Implemented on shared memory
  - Enabled by co-operation between host and guest kernels
    - Reconstruction of the hypervisor's page tables
  - Enables zero-copy between the Backend and Frontend
- Guest to Host shared buffers
  - Guest process' memory shared with a host process
  - Used to create the initial shared memory (Backend-Frontend)
- Host to Guest shared buffers
  - Replace guest physical memory with host physical memory
  - Used to map contiguous DMA buffers to guest memory

API Remoting - Backend

- Device sharing
  - multiple VMs
  - multiple guest applications per VM
- Auto-configuration of workers
  - threads are spawned/destroyed on:
    - VM start / shutdown
    - Guest frontend register / unregister
- Each thread has a private:
  - Shared memory with a frontend
  - Descriptor to API Remoting framework



API Remoting - Frontend

- Intercepts the target API
- Forward the requests to the Backend
- Control shared memory area
  - 1 page size length
  - Synchronization primitive - spin-lock
  - Describes the forwarded API functions:
    - An API command identifier
    - Forwarded arguments
    - Stores return values

Control shared memory area	
MEMORY ADDRESS	DATA
0x7ffeffe0	
0x7ffeffe8	
0x7ffefff0	
0x7ffefff8	
CALL_ID: 0x7fff0000	0x00000002
ARGUMENT_1: 0x7fff0008	0x1000a000
ARGUMENT_2: 0x7fff0010	0x00000040
ARGUMENT_3: 0x7fff0018	
ARGUMENT_4: 0x7fff0020	
ARGUMENT_5: 0x7fff0028	
ARGUMENT_6: 0x7fff0030	
ARGUMENT_7: 0x7fff0038	
0x7fff0040	0x00001030
0x7fff0048	0x00000001
0x7fff0050	

API Remoting – Shared memory

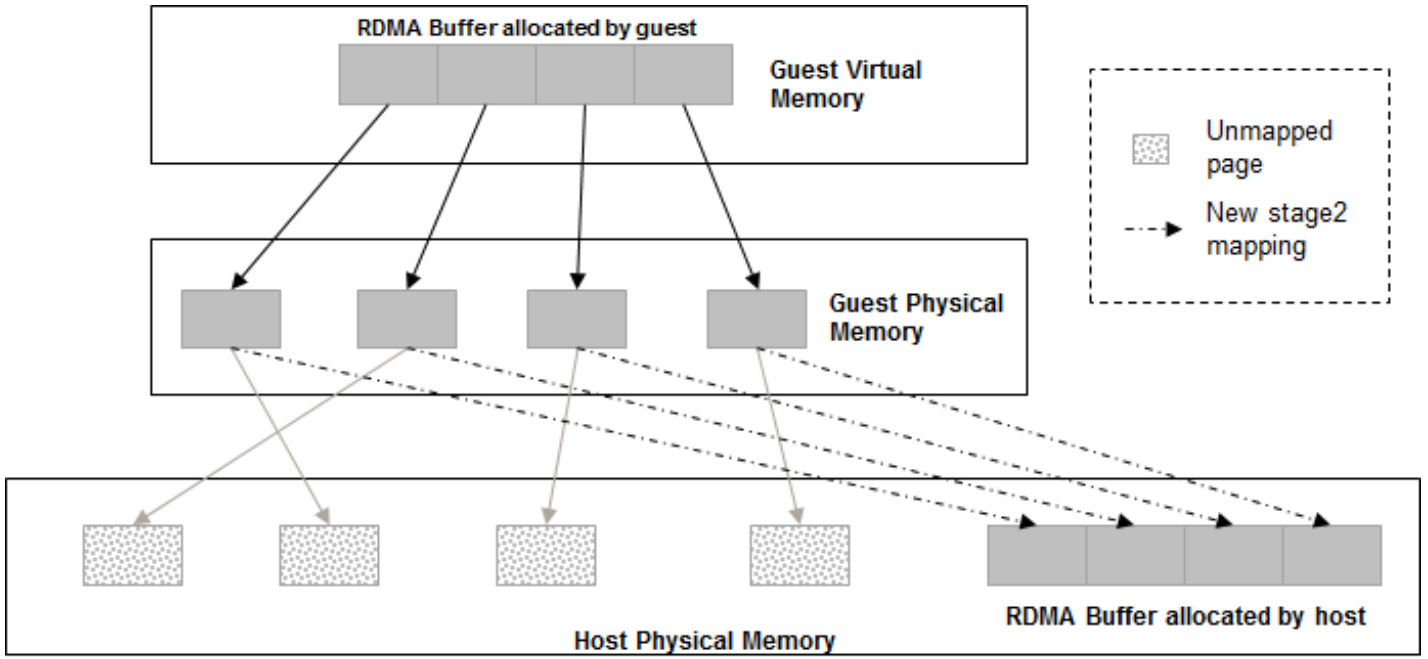
Guest to Host shared memory

- Share a guest allocated buffer with a host process
- After registration, access with load/store instructions
- Zero-copy
  - Eliminates memory copies for guest process data
- Depends on architecture features
  - Memory virtualization
  - On ARM architecture
    - Two memory translation stages (VA → IPA→ PA)

API Remoting – Shared memory

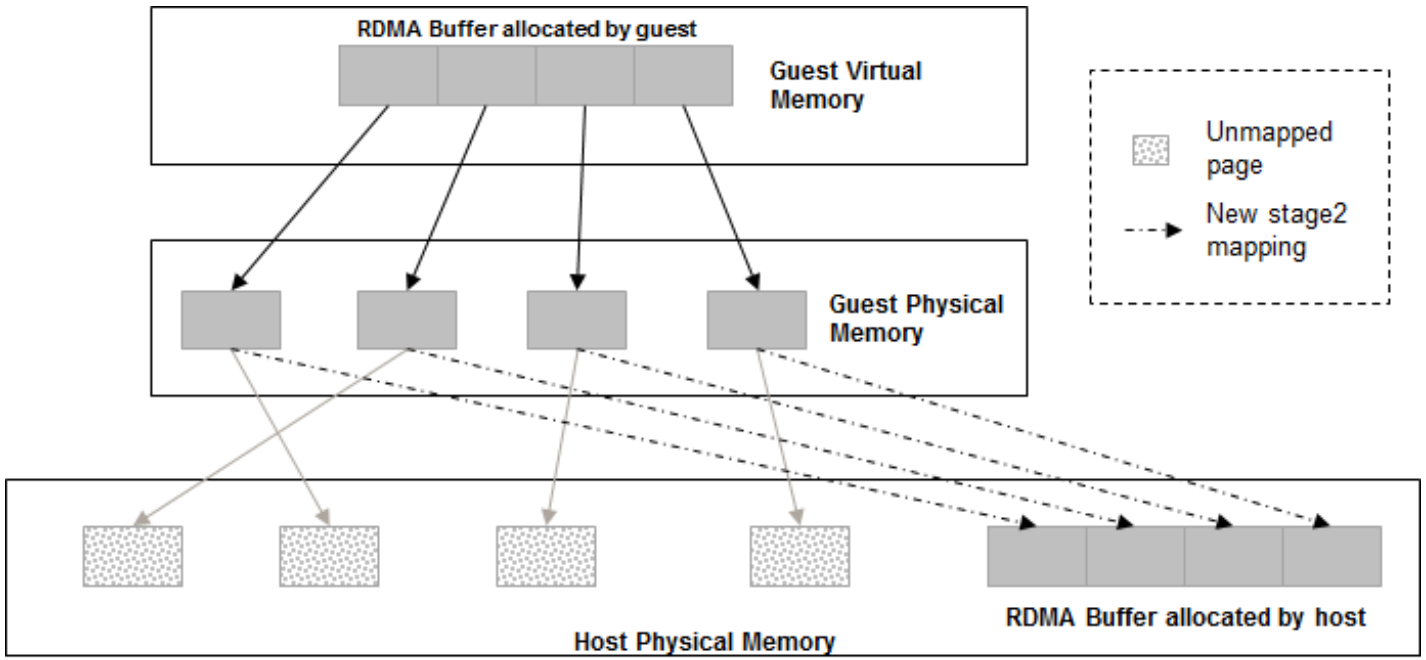
Host to guest shared memory

- An example of a DMA buffer allocation
- Frontend
  - Allocates the destination buffer (mmap)
  - Forwards the call and buffer's VA



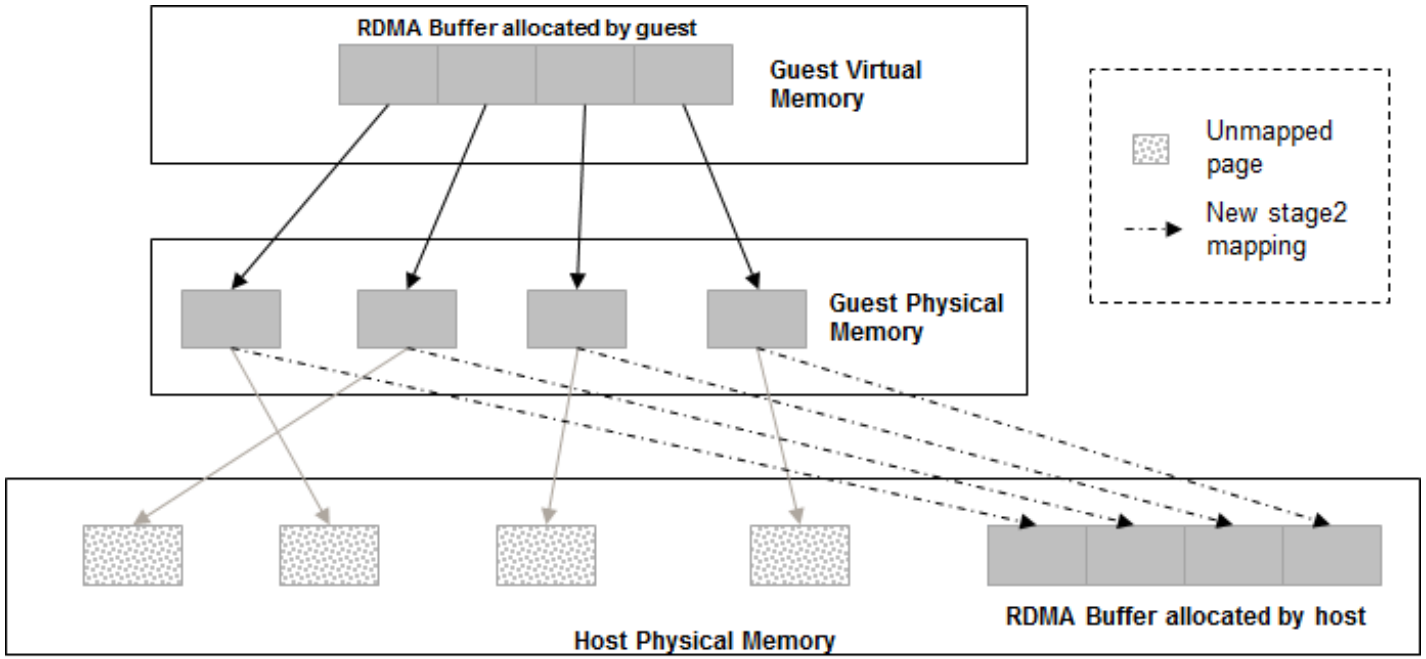
API Remoting – Shared memory

- Backend
  - Allocates a ‘real’ DMA buffer
  - VAs & size are passed to the kernel
  - The driver tracks the page frames:
    - Guest Frames (guest buffer)
    - Page Frames (host buffer)

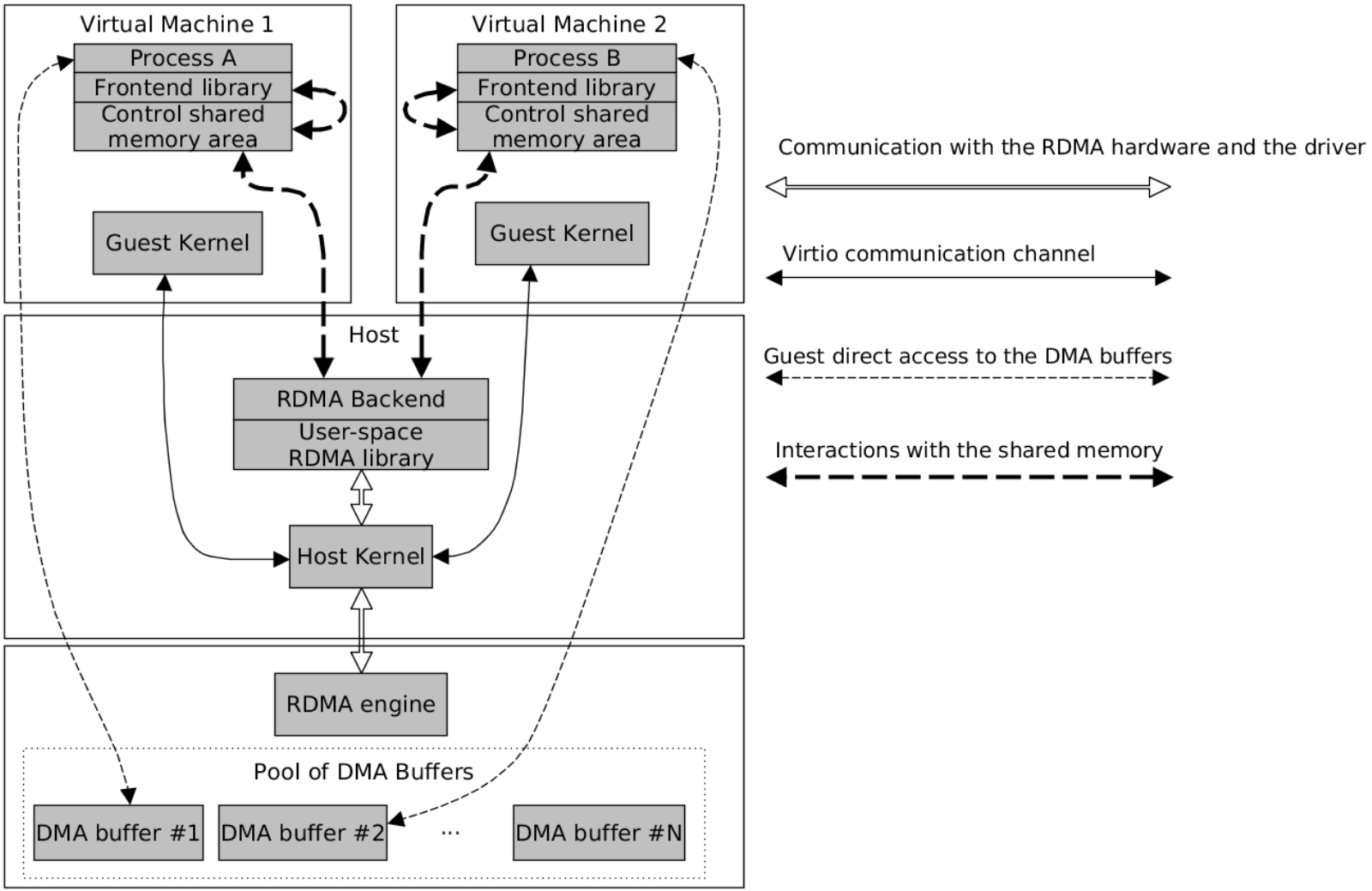


API Remoting – Shared memory

- Backend
  - Pages swap (QEMU address space)
    - Guest pages are removed
    - Host pages inserted in place
- Guest application
  - The mmap()'d buffer points to the DMA buffer
  - Direct access without copies



API Remoting – Overall framework



API Remoting - Synchronization

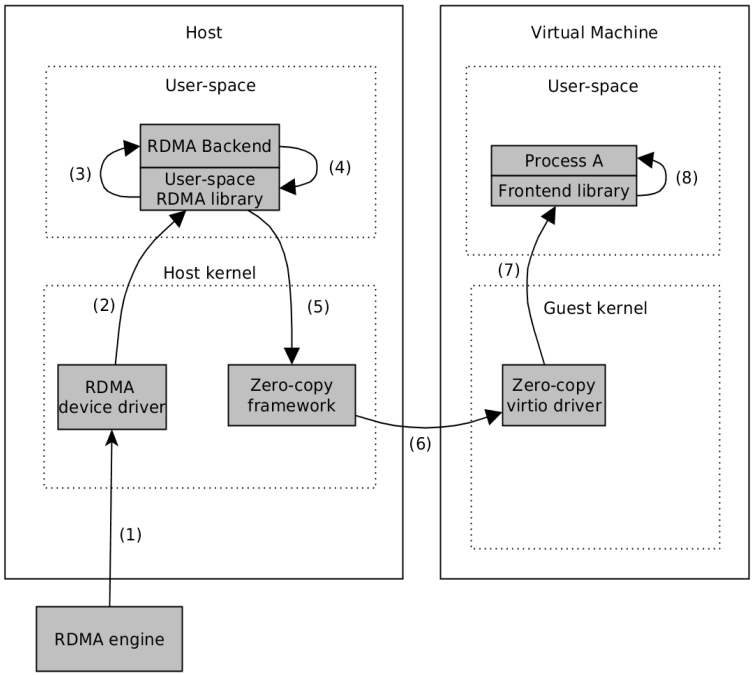
Synchronization primitives

- The Frontend and the Backend thread are synchronized with:
- Spin-lock
  - A spin-lock into the control shared memory area
  - ✔ It is very fast, simple implementation
  - ✖ Will waste CPU cycles in the event of inactivity
- Virtio messages
  - A process can register a synchronization event to the remote kernel
  - The local process may sleep until remote events
  - Efficient management of CPU resources
  - It is slower than spin-lock, due to guest exits

API Remoting – Event forwarding

Forwarding a completion event to the guest process

- Events are handled by the Backend
- Backend locates the initiator
- Using kernel infrastructure backend informs the guest kernel for the completior
- Guest kernel raises a SIGUSR1 to the target process
- The frontend library dispatches the user defined handler



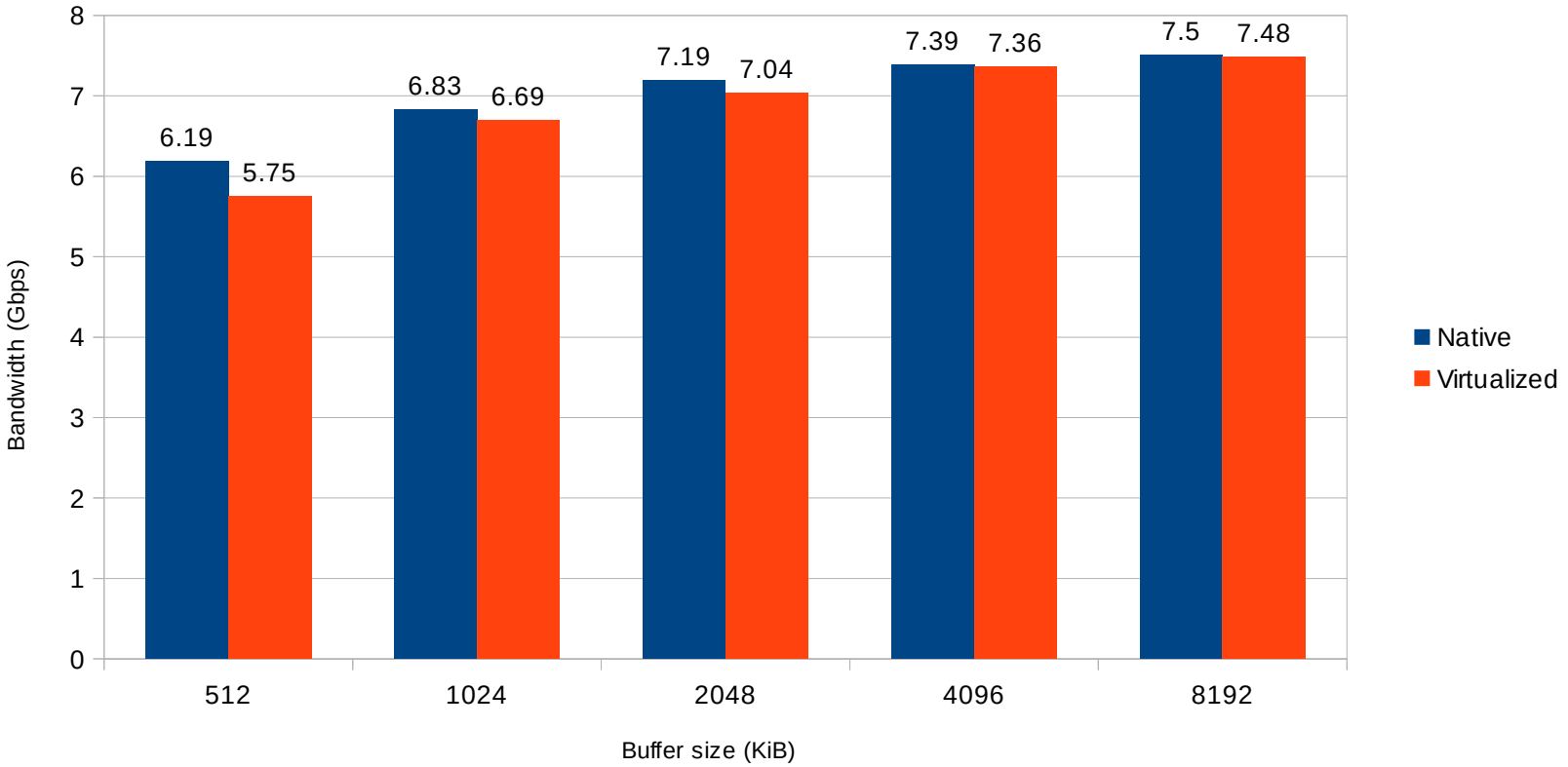
Experimental Results (1/5)

Set of tests performed on ExaNeSt's prototype

- Virtualized vs Native performance
  - Maximum bandwidth for single transfers
  - Maximum bandwidth utilized with variable size burst transfers
  - DMA buffer allocation
- Virtualization overhead
  - Performance vs efficiency of the synchronization primitives

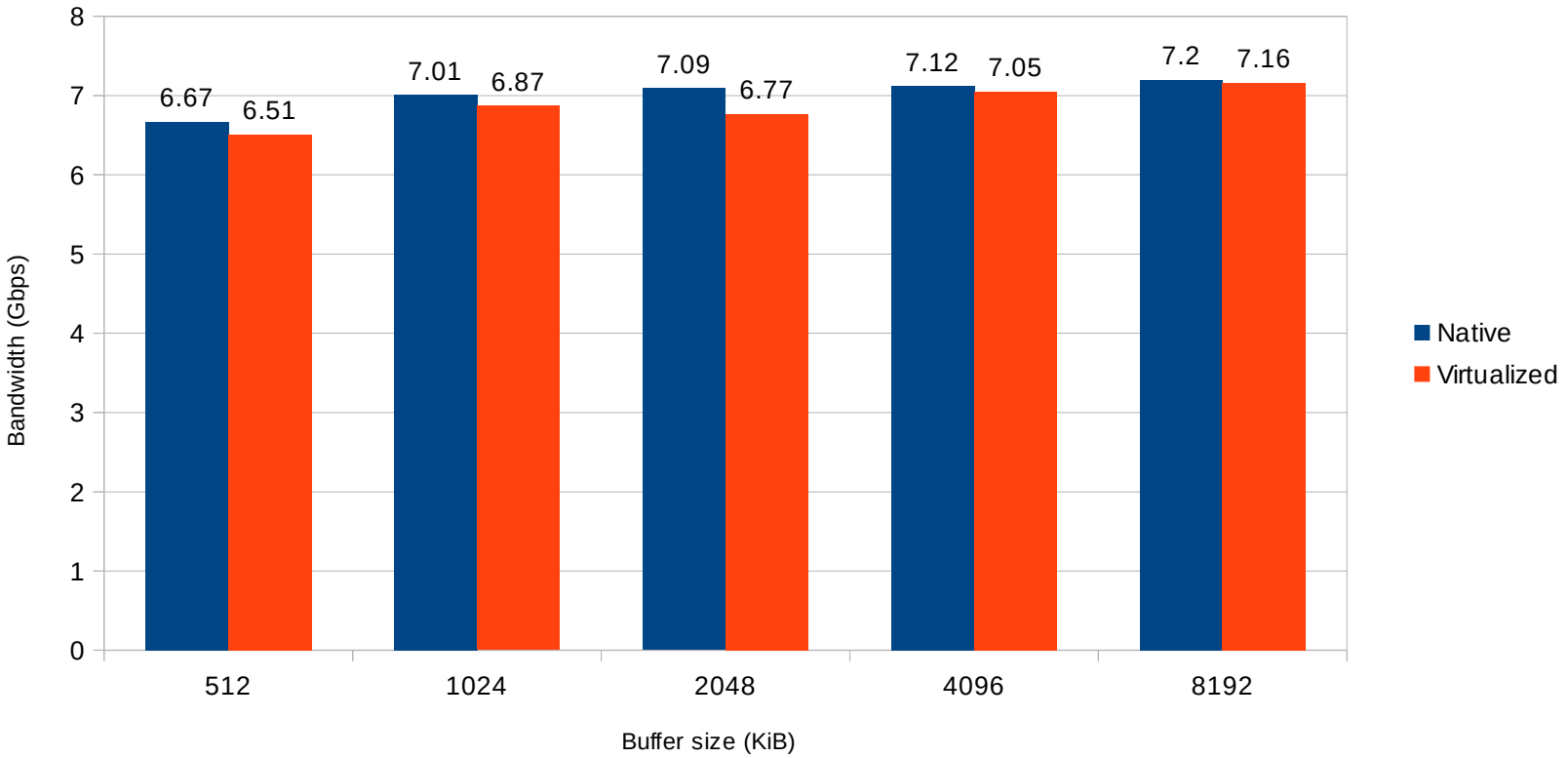
Experimental Results (2/5)

- Maximum transfer rate for a single transfer
  - less than 7% overhead (case of 512KiB buffers)
  - negligible with larger buffers



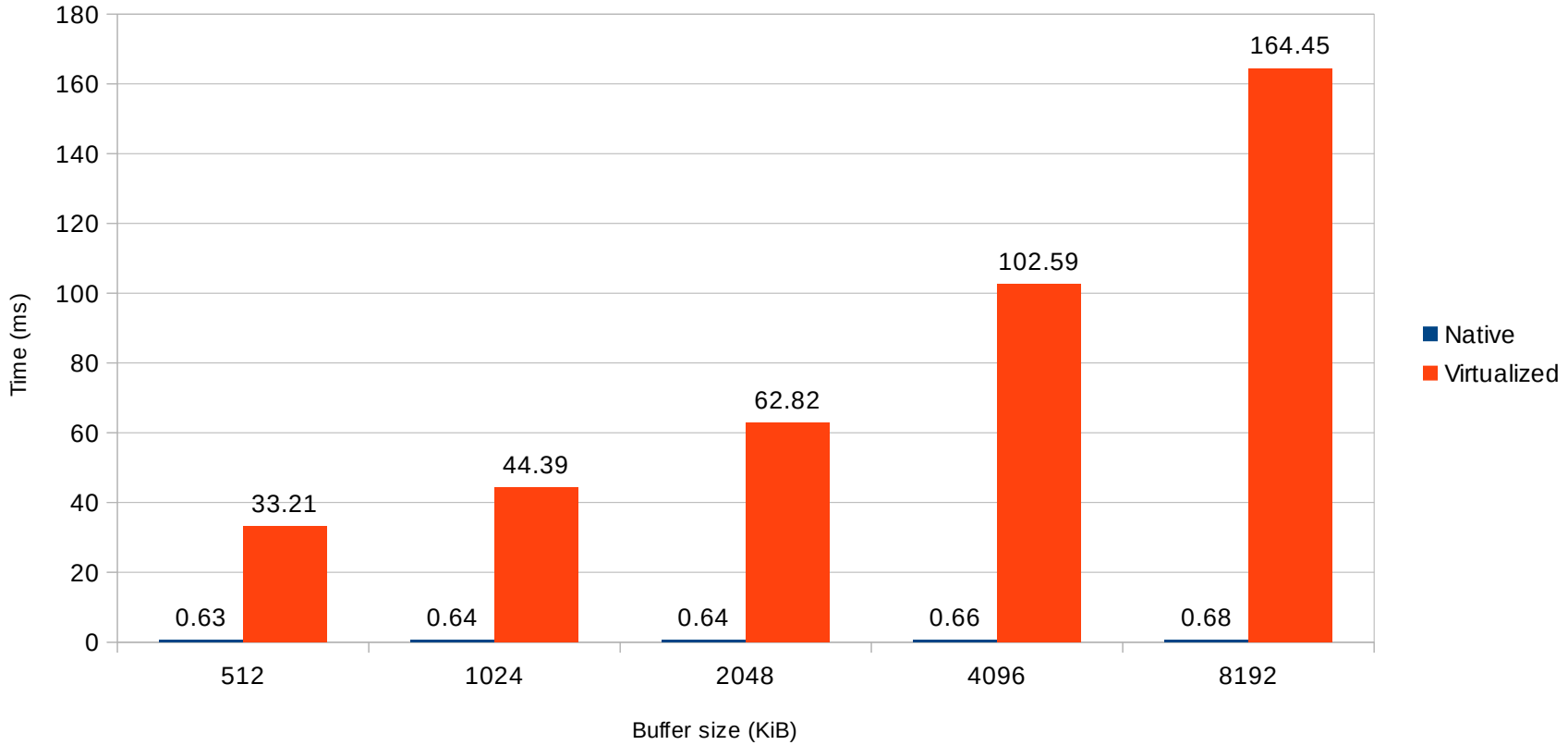
Experimental Results (3/5)

- Maximum transfer rate for a burst transfer of 1GiB of data
  - less than 4% overhead (case of MiBKiB buffers)
  - negligible with larger buffers
  - Parallel burst transfers are more efficient than single transfers



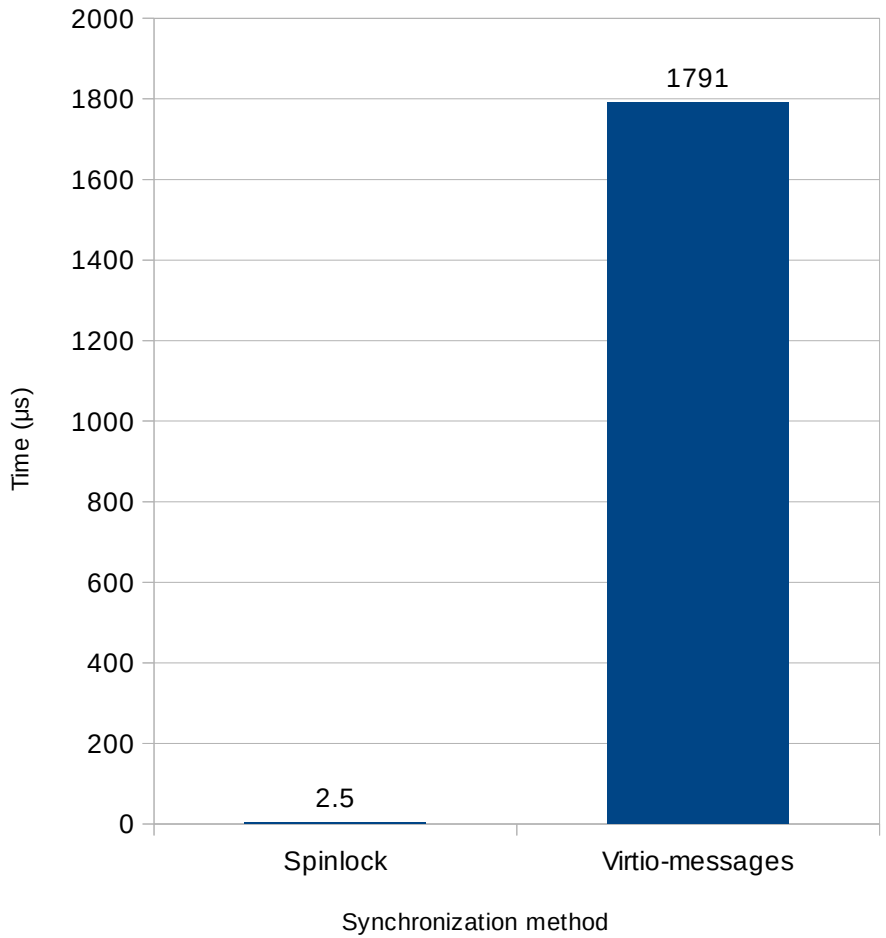
Experimental Results (4/5)

- DMA buffer allocation
  - The virtualization overhead is proportional to buffer's size
    - The overhead is paid only at the allocation time
    - Following accesses to the buffer are served with native performance



Experimental Results (5/5)

- Synchronization primitives benchmarks



Conclusions

- RDMA Virtualization based on API Remoting communication
- Bidirectional shared memory buffers
- Access from guest processes to contiguous physical memory buffers
- Support multiple guest applications from different VM
- Bandwidth utilization close to native
  - Overhead up to 7% for small (512KB) single transfers
- Extensions for the current work
  - Adaptive mixed primitive synchronization policy
  - Extend the target API to support OFED-libibverbs

References

[1] Unimem architecture : Marazakis, M. et al.: EUROSERVER: Share anything scale-out micro-server design. In: 2016 Design, Automation & Test in Europe Conference & Exhibition (DATE). pp. 678–683. IEEE (2016)

[2] Xilinx AXI Central Direct Memory Access IP : https://www.xilinx.com/support/documentation/ip_documentation/axi_cdma/v4_1/pg034-axi-cdma.pdf

Virtual Open Systems

http://exanest.eu