sslab
system software

GPUvm: Why Not Virtualizing GPUs at the Hypervisor?

Yusuke Suzuki*
in collaboration with
Shinpei Kato**, Hiroshi Yamada***, Kenji Kono*

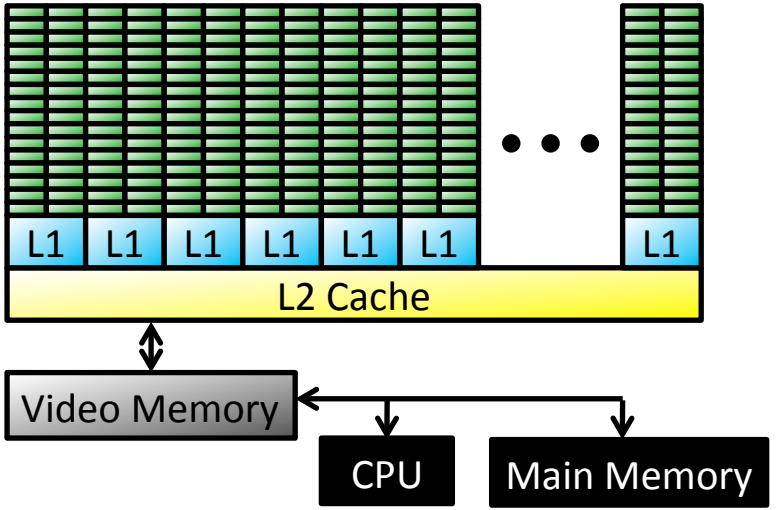
* Keio University
** Nagoya University
*** Tokyo University of Agriculture and Technology

Graphic Processing Unit (GPU)

sslab system software

- GPUs are used for data-parallel computations
  - Composed of thousands of cores
  - Peak double-precision performance exceeds 1 TFLOPS
  - Performance-per-watt of GPUs outperforms CPUs
- GPGPU is widely accepted for various uses
  - Network Systems **[Jang et al. '11]**, FS **[Silberstein et al. '13]** **[Sun et al. '12]**, DBMS **[He et al. '08]** etc.

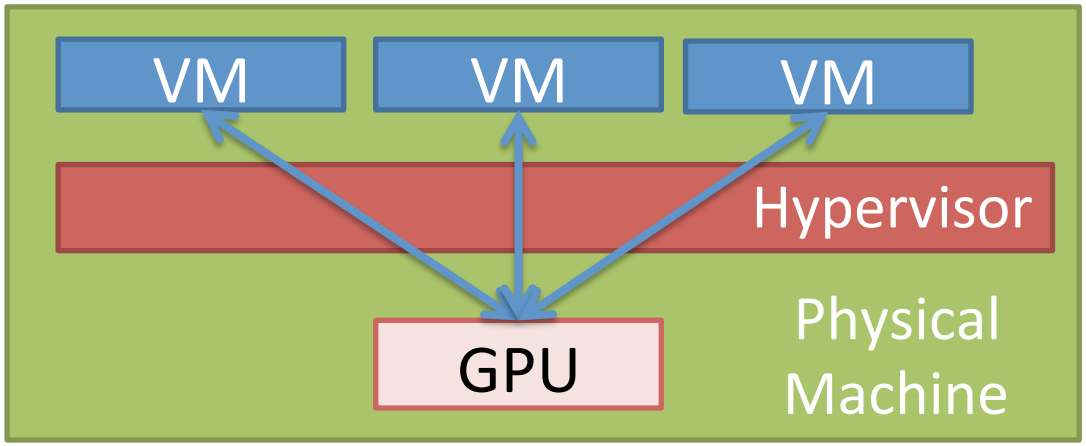
NVIDIA/GPU



Motivation

- GPU is not the first-class citizen of cloud computing environment
  - Can not multiplex GPGPU among virtual machines (VM)
  - Can not consolidate VMs that run GPGPU applications
- GPU virtualization is necessary
  - Virtualization is the norms in the clouds

Share a single GPU among VMs



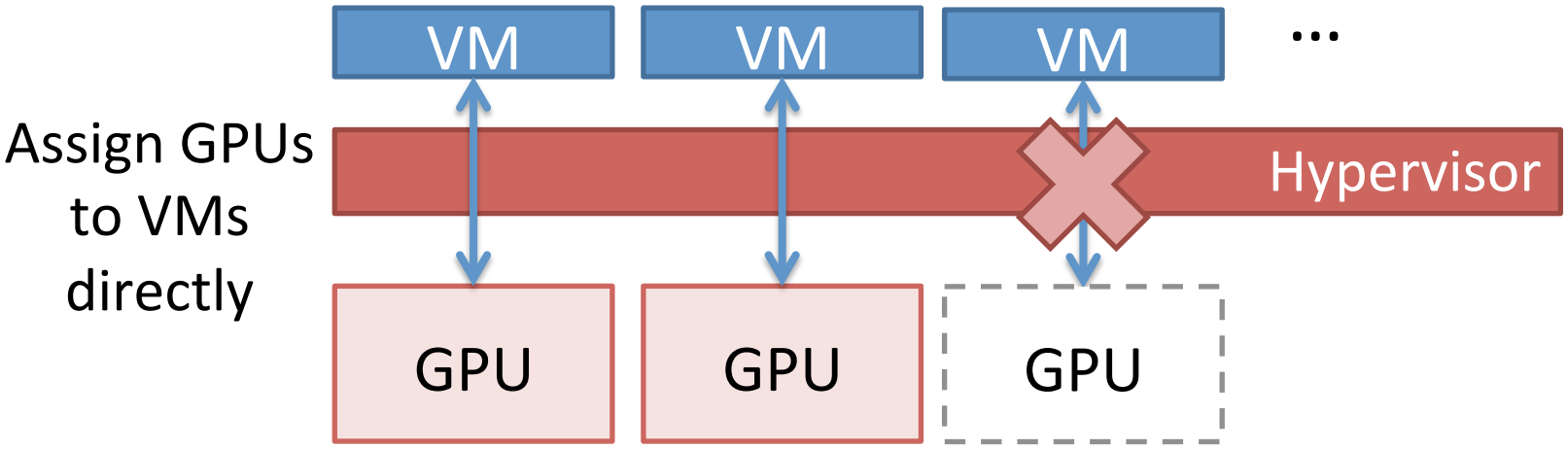
Virtualization Approaches

sslab
system software

- Categorized into three approaches
  1. I/O pass-through
  2. API remoting
  3. Para-virtualization

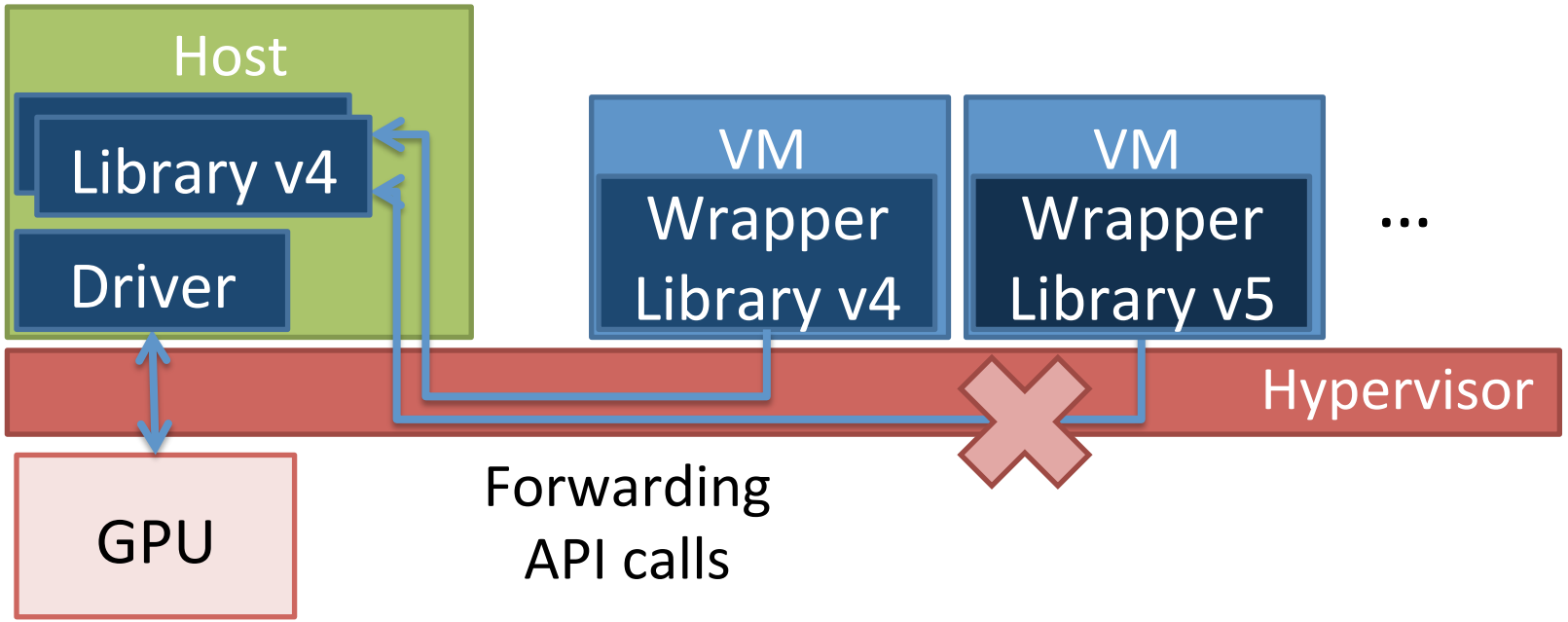
I/O pass-through

- Amazon EC2 GPU instance, Intel VT-d
  - Assign physical GPUs to VMs directly
  - Multiplexing is impossible



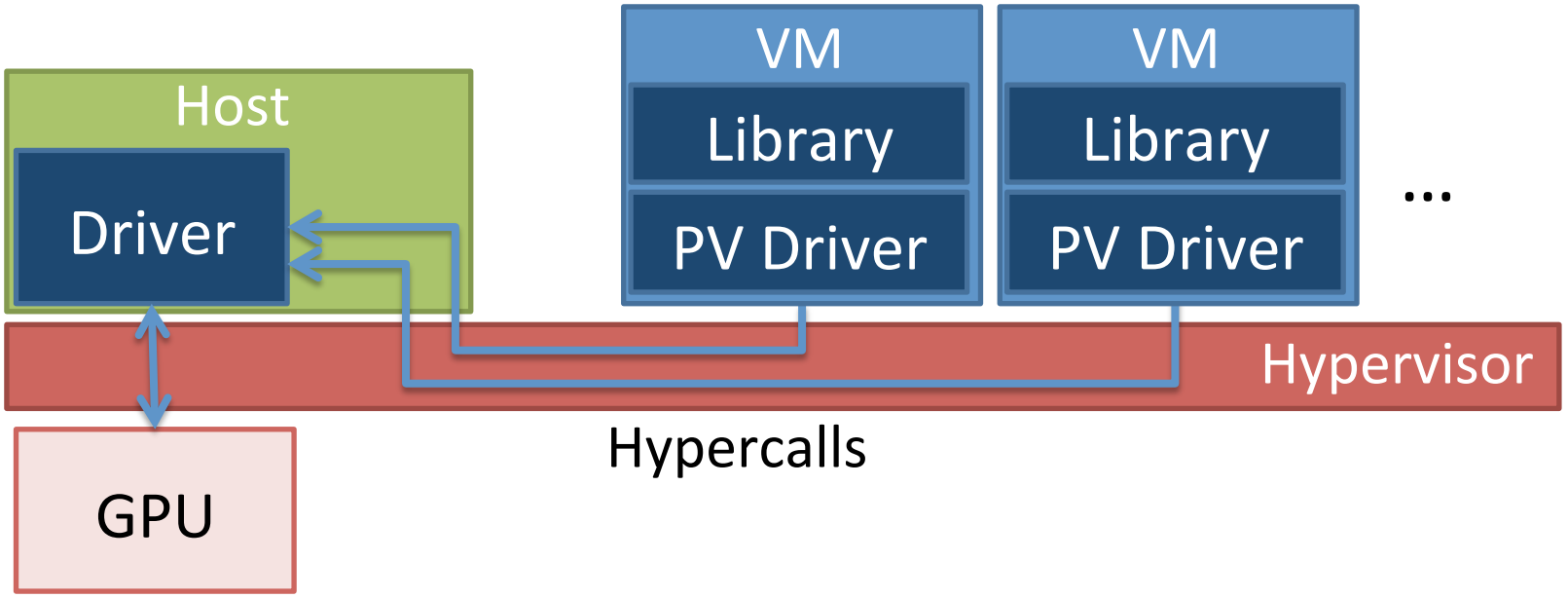
API remoting

- GViM [Gupta et al. '09], rCUDA [Duato et al '10], VMGL [Largar-Cavilla et al. '07] etc.
  - Forward API calls from VMs to the host's GPUs
  - API and its version compatibility problem
  - Enlarge the trusted computing base (TCB)



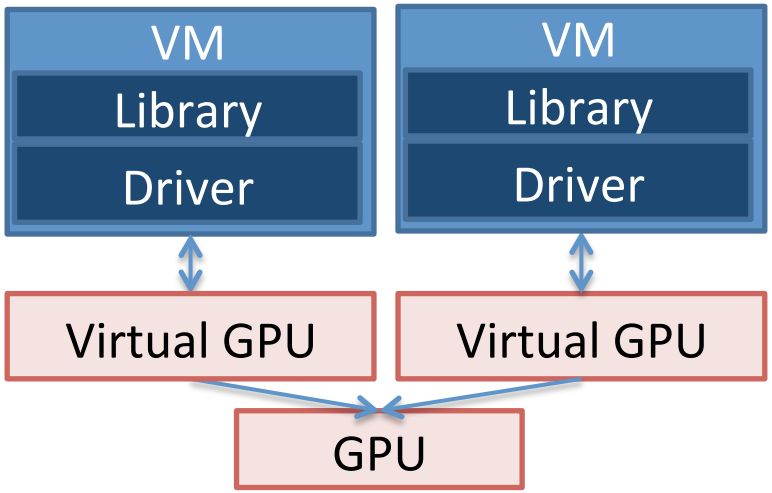
Para-virtualization

- VMWare SVGA2 **[Dowty '09]** LoGV **[Gottschalk et al. '10]**
  - Expose an ideal GPU device model to VMs
  - Guest device driver must be modified or rewritten



Goals

- Fully virtualize GPUs
  - **allow multiple VMs to share a single GPU**
  - **without any driver modification**
    - Vanilla driver can be used “as is” in VMs
    - GPU runtime can be used “as is” in VMs
- Identify performance bottlenecks of full virtualization
  - GPU details are not open…



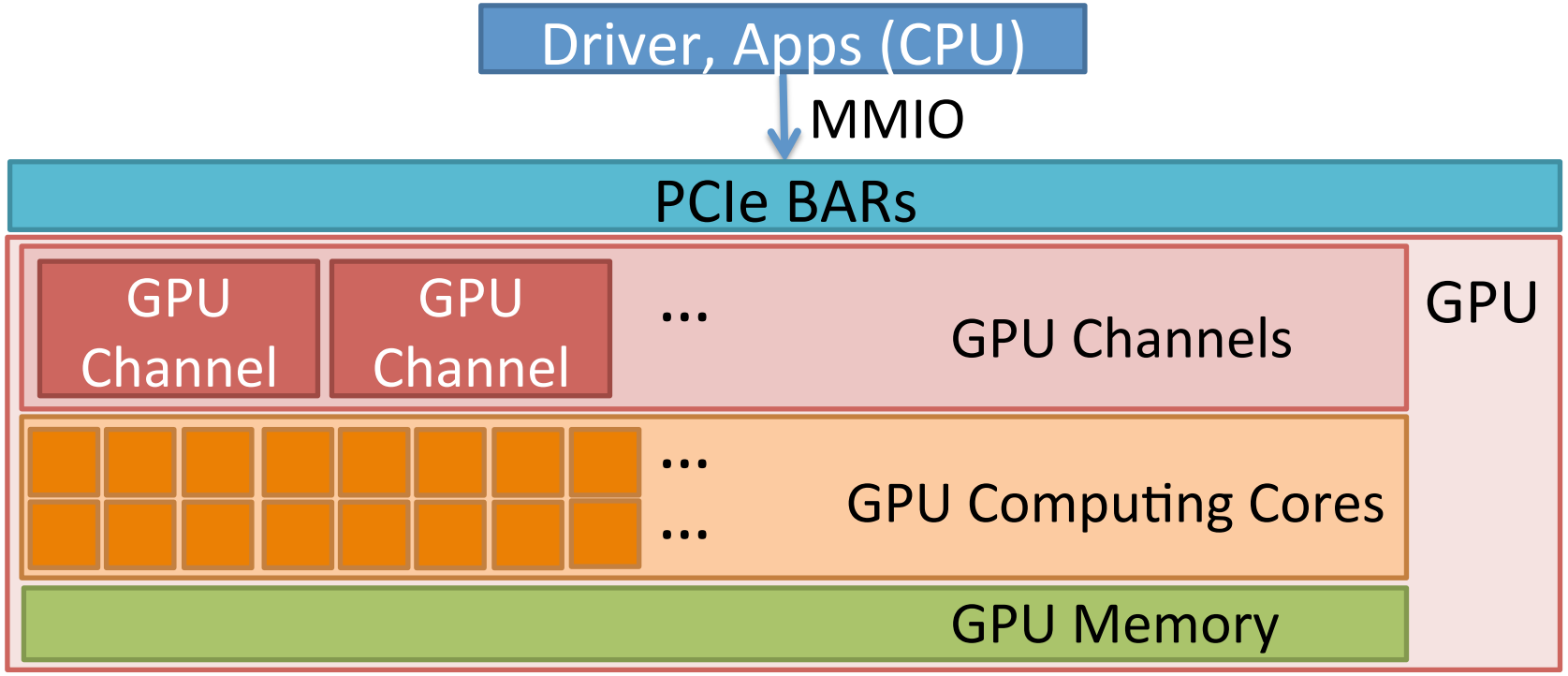
Outline

sslab system software

- Motivation & Goals
- GPU Internals
- Proposal: GPUvm
- Experiments
- Related Work
- Conclusion

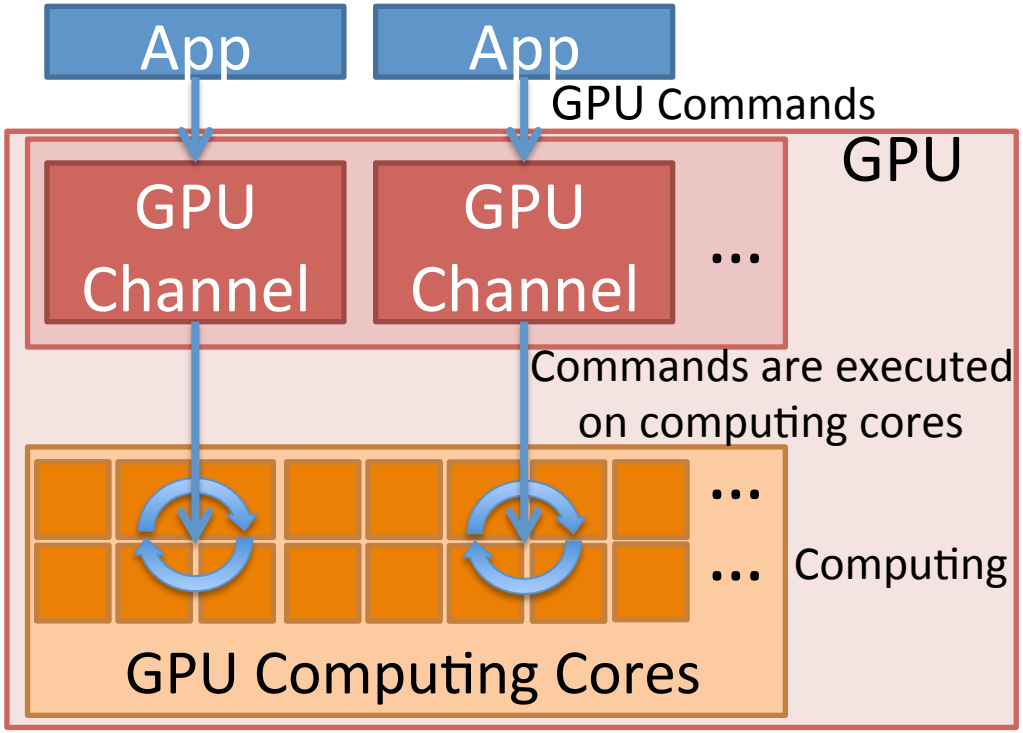
GPU Internals

- PCIe connected discrete GPU (NVIDIA, AMD GPU)
- Driver accesses to GPU w/ MMIO through PCIe BARs
- Three major components
  - **GPU computing cores**, **GPU channel** and **GPU memory**



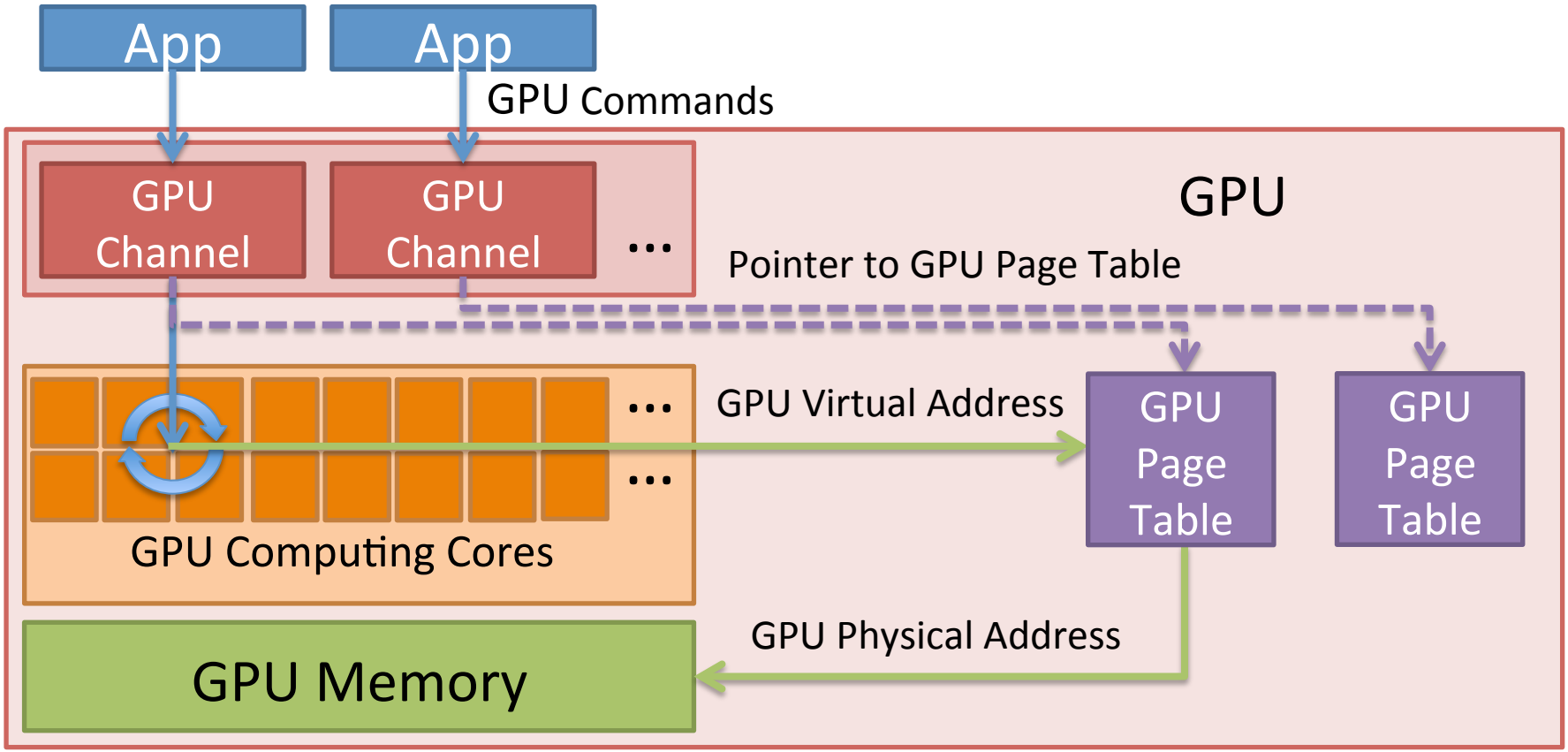
GPU Channel & Computing Cores

- GPU channel is a hardware unit to submit commands to GPU computing cores
- The number of GPU channels is fixed
- Multiple channels can be active at a time



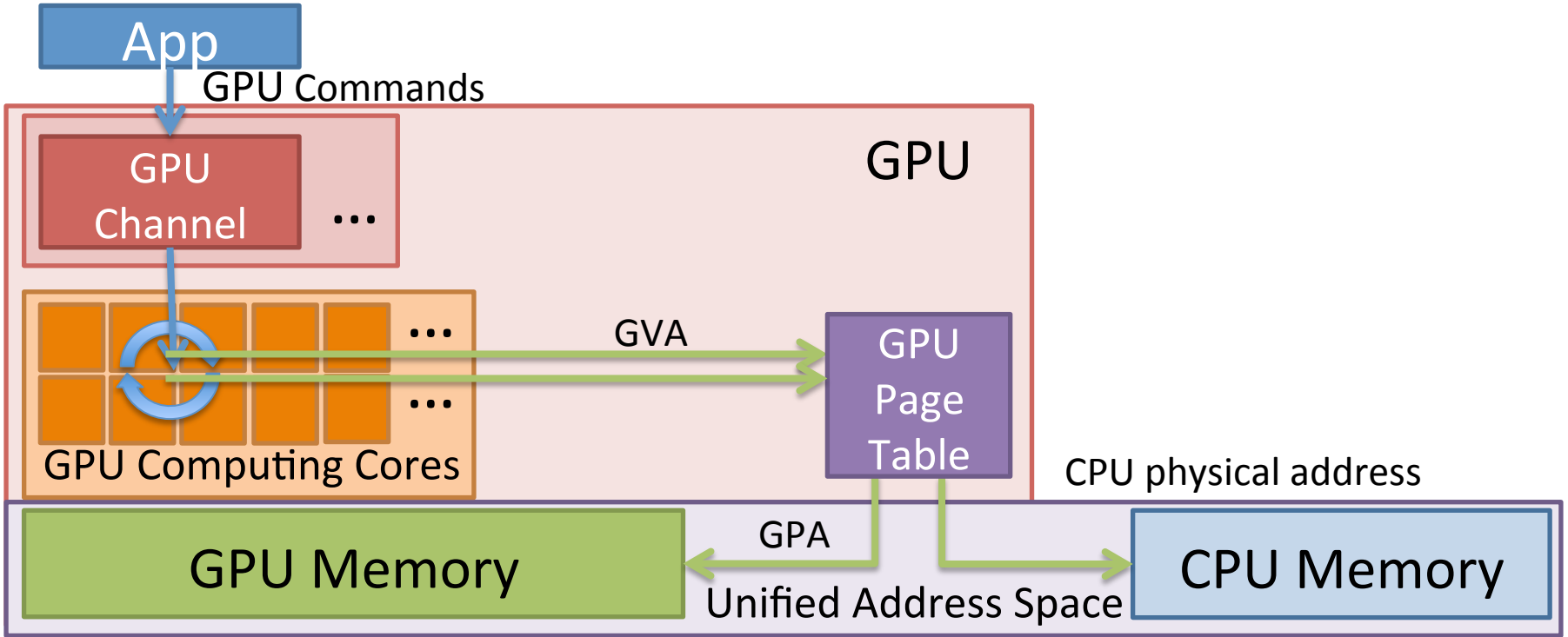
GPU Memory

- Memory accesses from computing cores are confined by GPU page tables



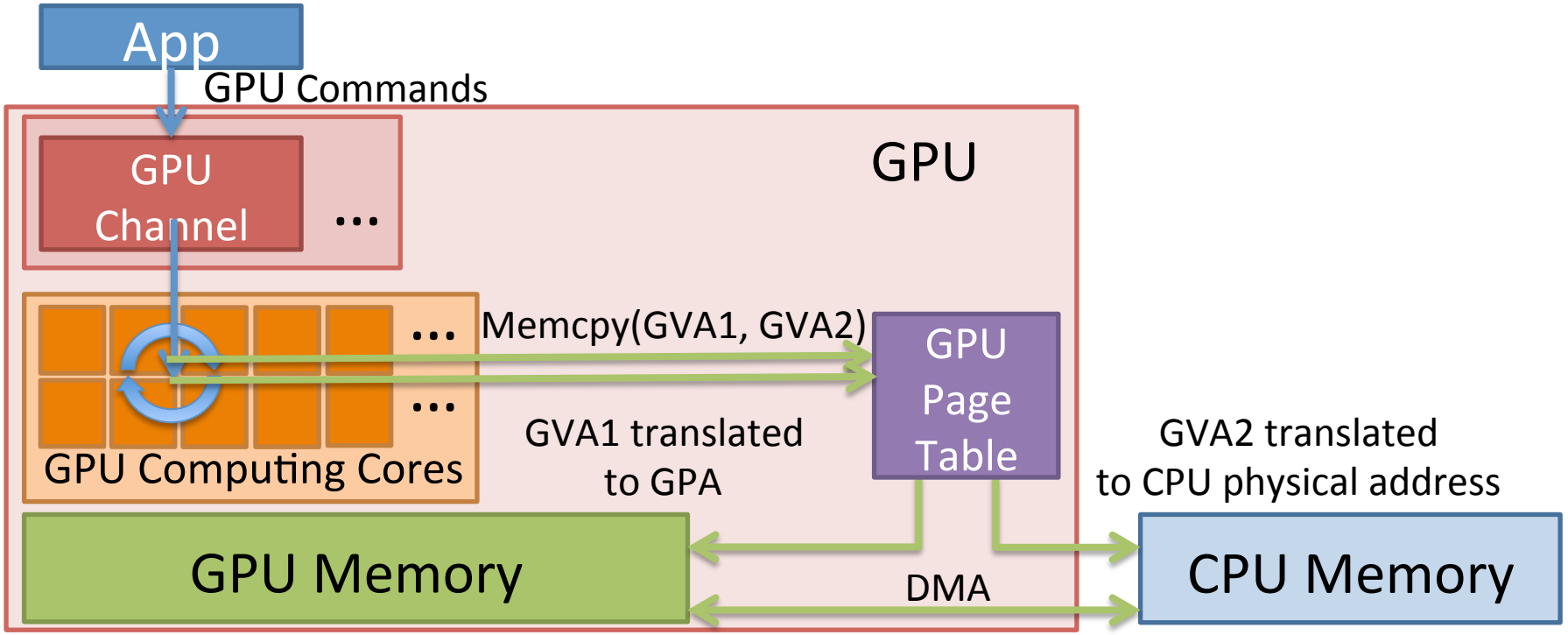
Unified Address Space

- GPU and CPU memory spaces are unified
  - GPU virtual address (GVA) is translated CPU physical addresses as well as GPU physical addresses (GPA)



DMA handling in GPU

- DMAs from computing cores are issued with GVA
  - Confined by GPU Page Tables
- DMAs must be isolated between VMs

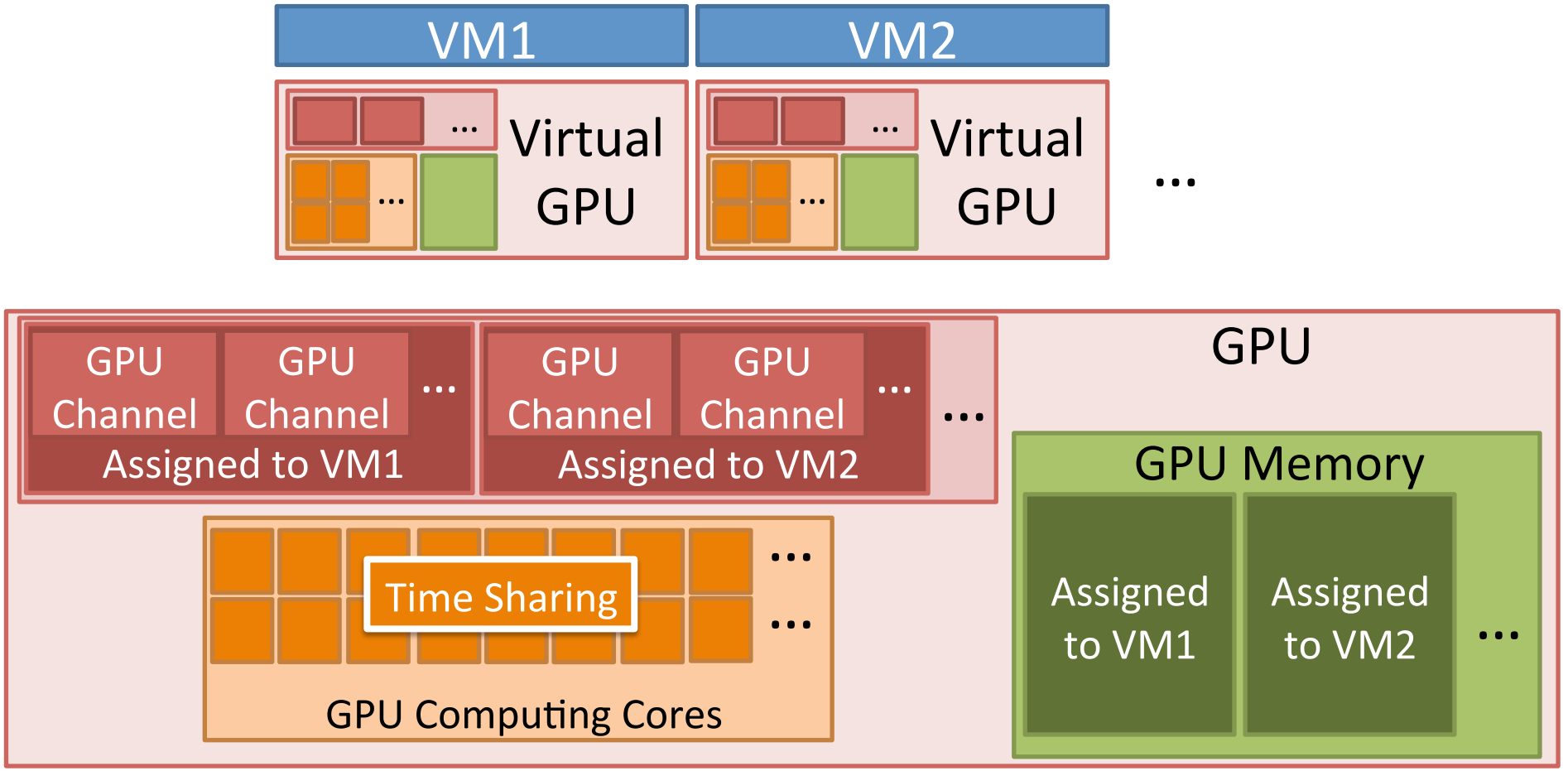


Outline

- Motivation & Goals
- GPU Internals
- Proposal: GPUvm
- Experiments
- Related Work
- Conclusion

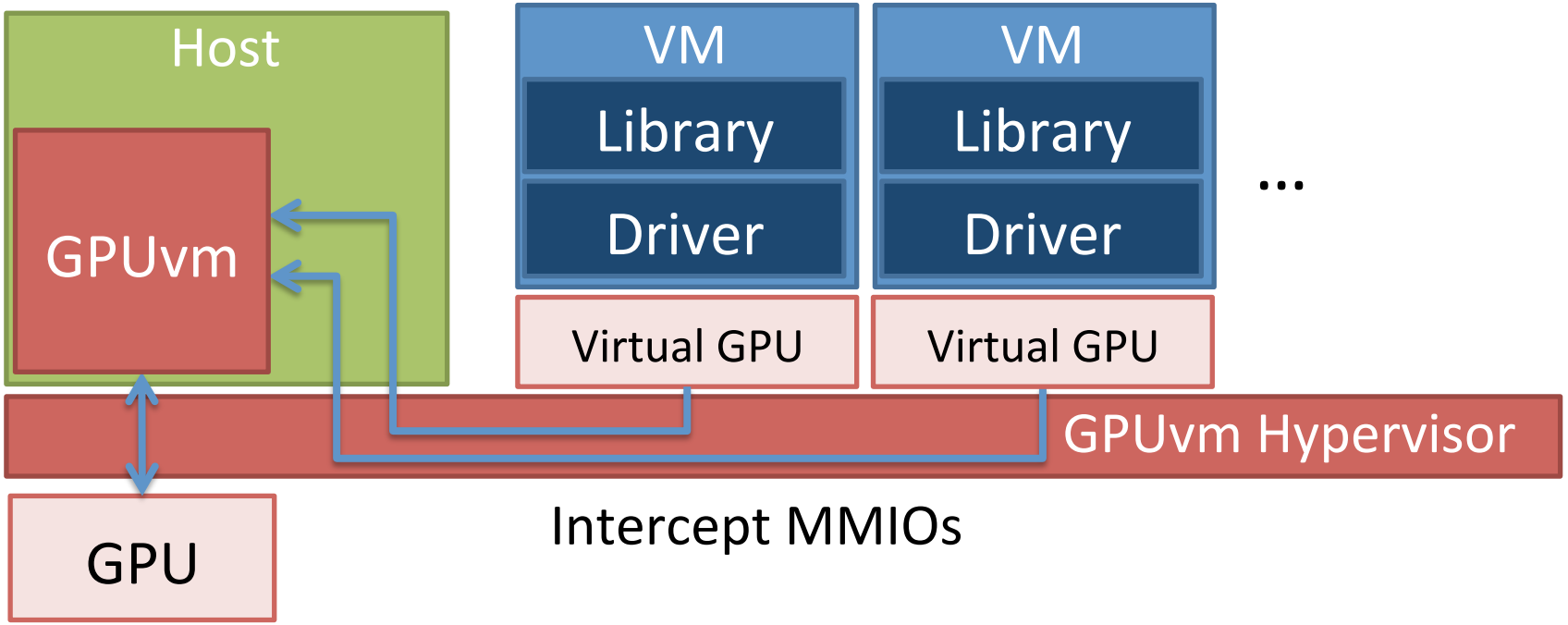
GPUvm overview

- Isolate GPU channel, computing cores & memory



GPUvm Architecture

- Expose the Virtual GPU to each VM and intercept & aggregate MMIO to them
- Maintain Virtual GPU views and arbitrate accesses to physical GPU



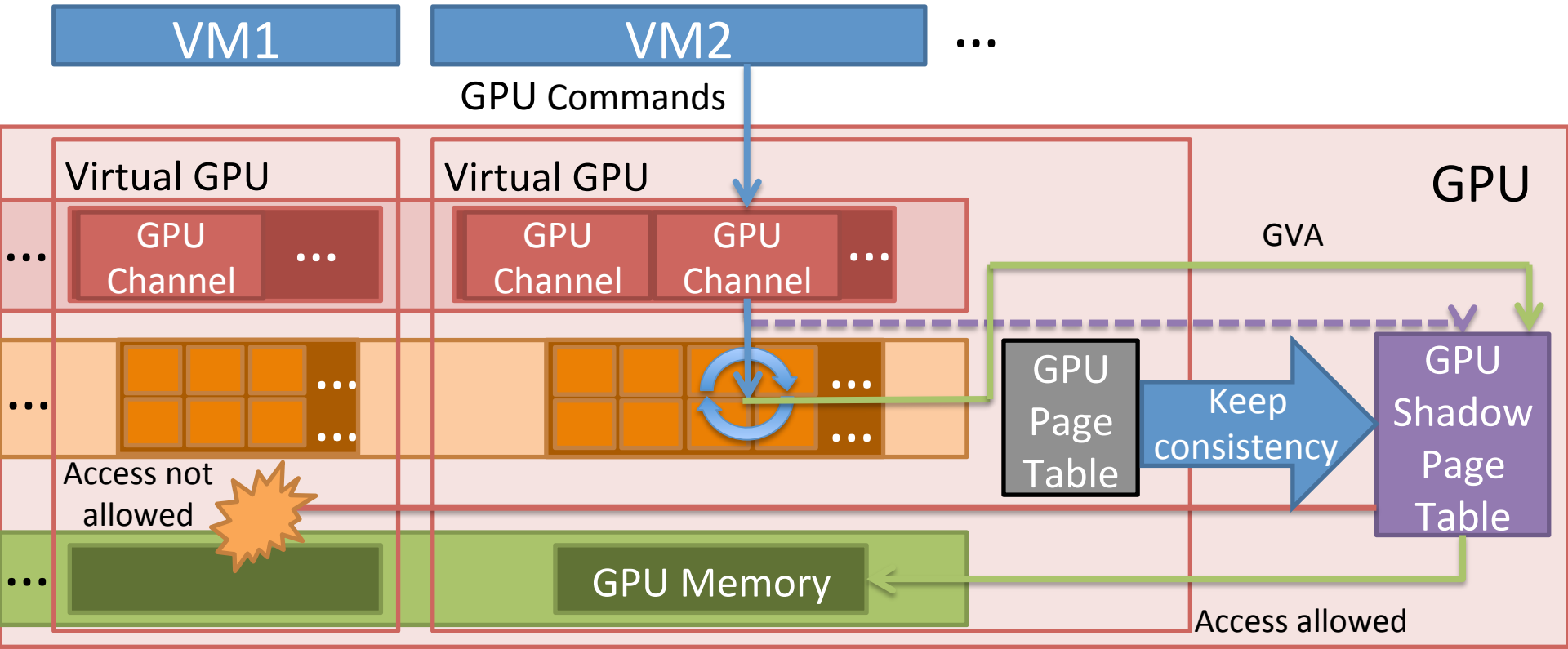
GPUvm components

sslab system software

1. GPU shadow page table
   - Isolate GPU memory
2. GPU shadow channel
   - Isolate GPU channels
3. GPU fair-share scheduler
   - Isolate GPU time using GPU computing cores

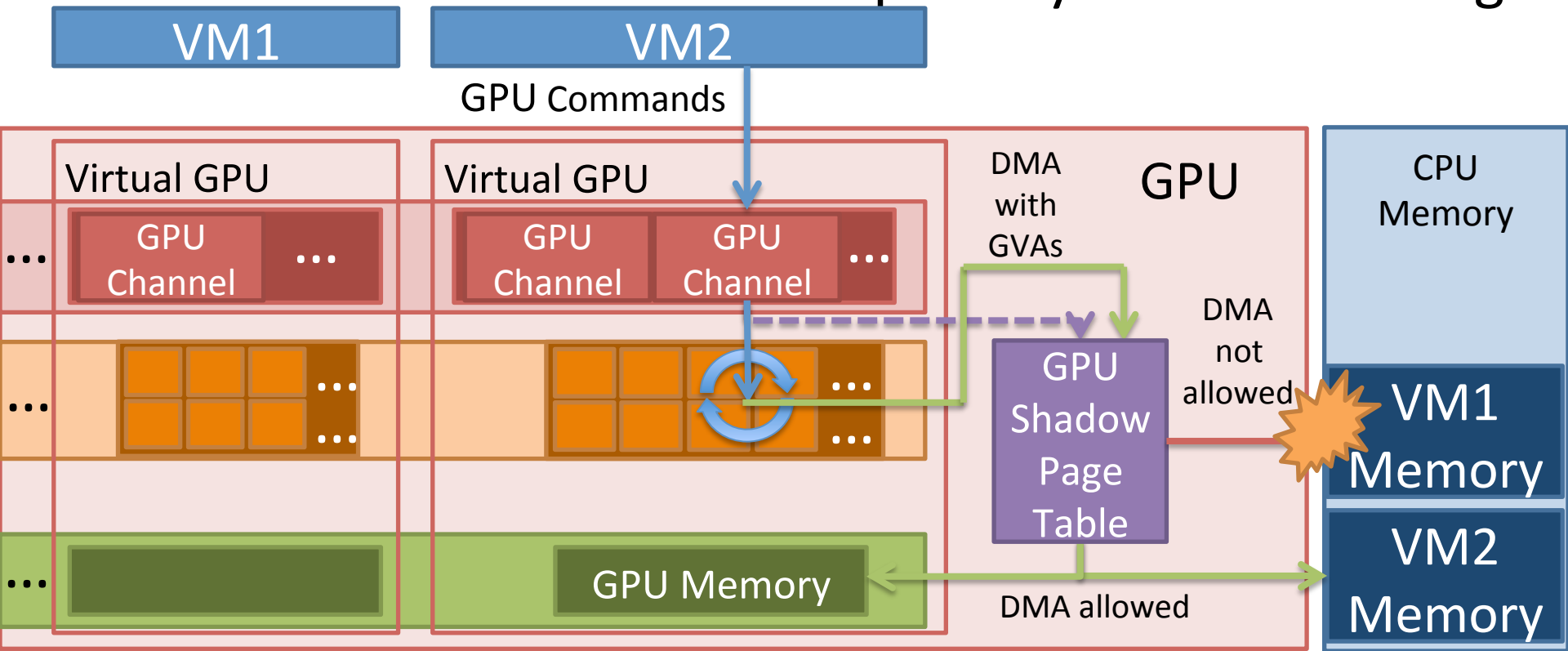
GPU Shadow Page Table

- Create GPU shadow page tables
  - Memory accesses from GPU computing cores are confined by GPU shadow page tables



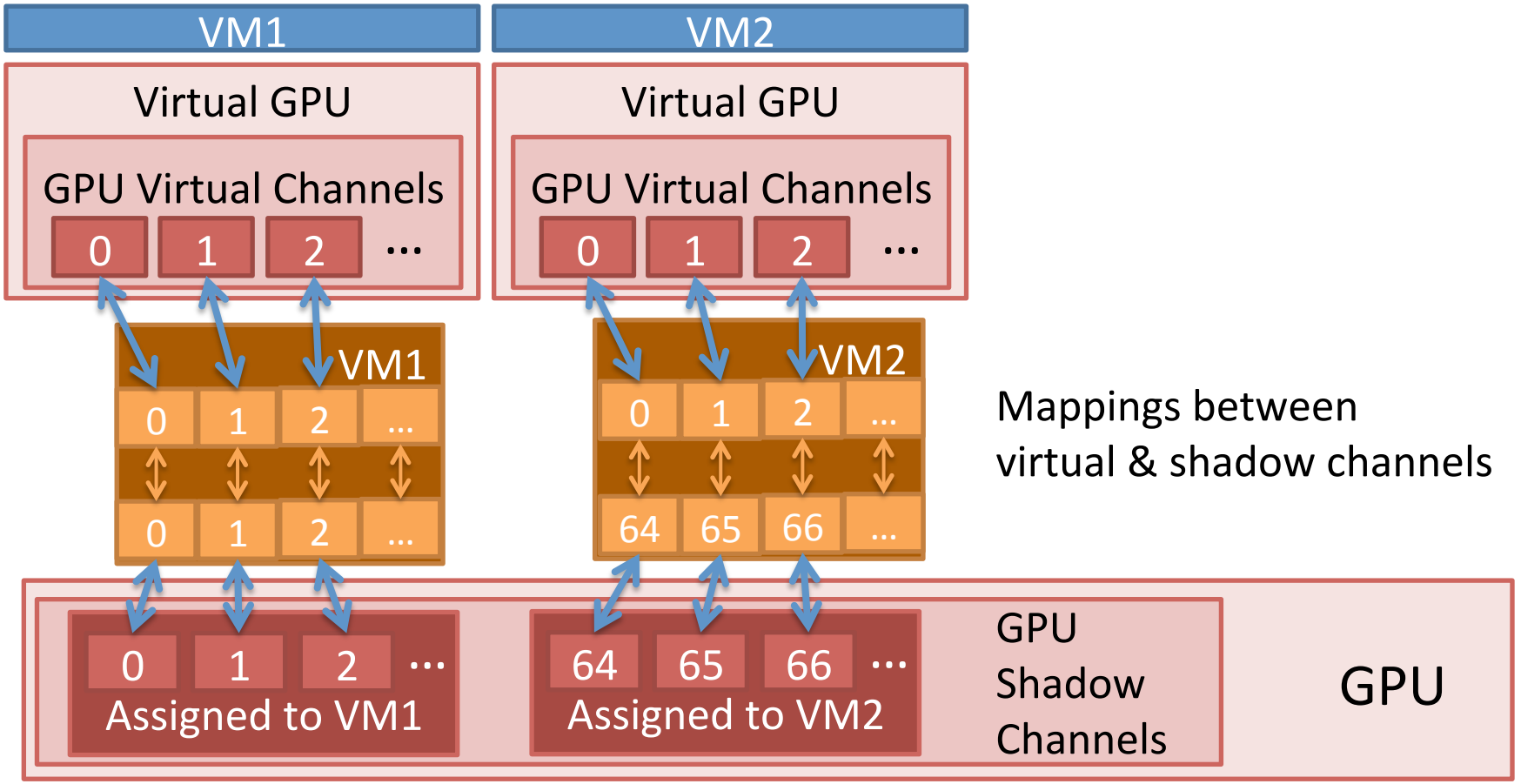
GPU Shadow Page Table & DMA

- DMA is also confined by GPU shadow page tables
  - Since DMA is issued with the GVA
- Other DMAs can be intercepted by MMIO handling



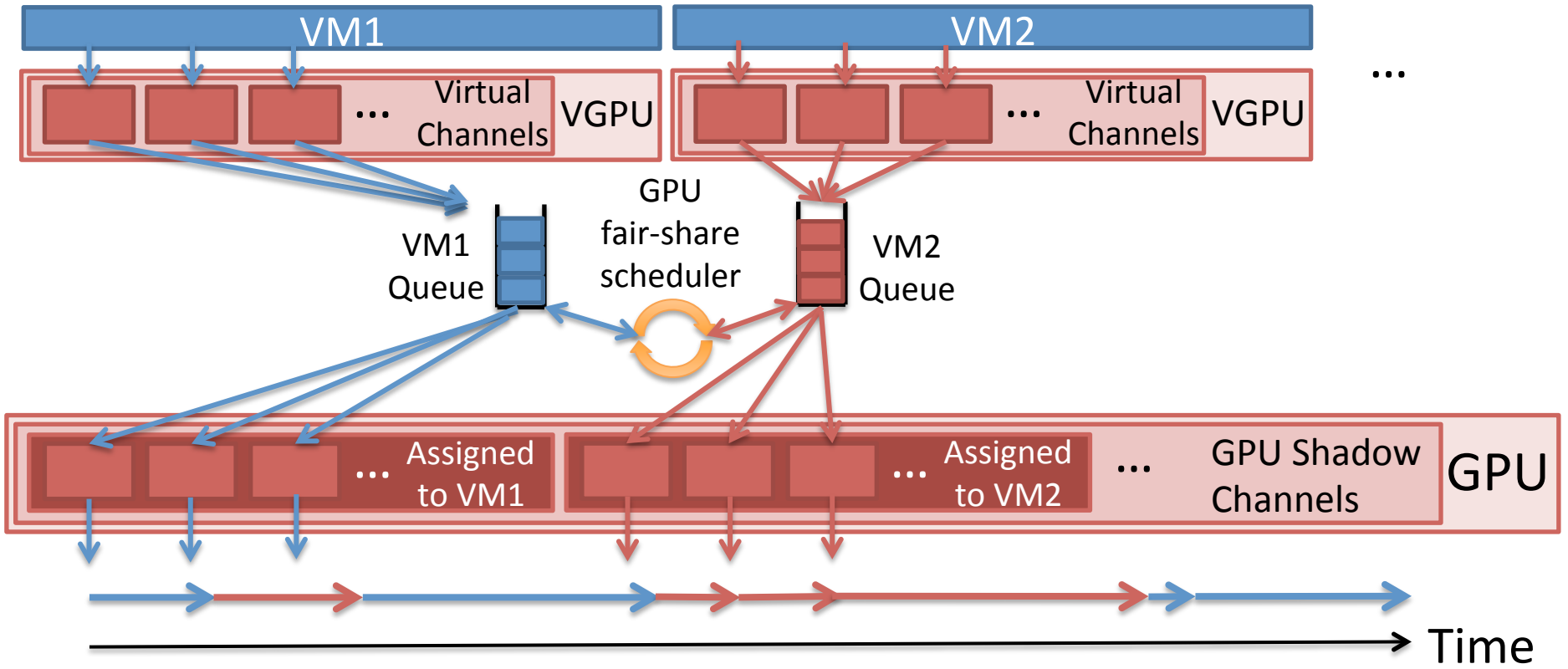
GPU Shadow Channel

- Channels are logically partitioned for VMs
- Maintain mappings between virtual & shadow channels



GPU Fair-Share Scheduler

- Schedules non-preemptive command executions
- Employs BAND scheduling algorithm **[Kato et al. '12]**
- GPUvm can employ existing algorithms
  - VGRIS **[Yu et al. '13]**, Pegasus **[Gupta et al. '12]**, TimeGraph **[Kato et al. '11]**, Disengaged Scheduling **[Menychtas et al. '14]**



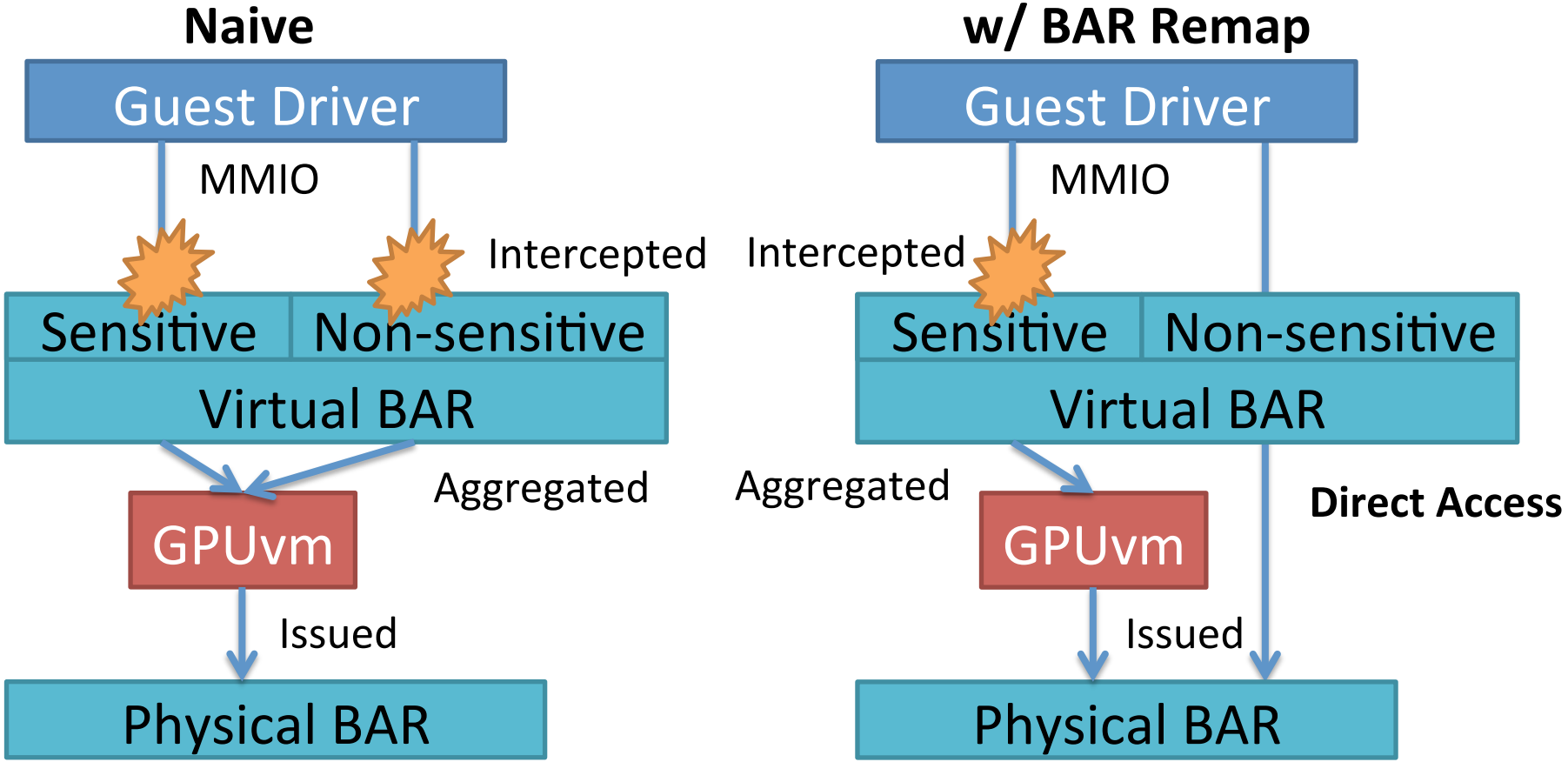
Optimization Techniques

sslab
system software

- Introduce several optimization techniques to reduce overhead caused by GPUvm
  1. BAR Remap
  2. Lazy Shadowing
  3. Para-virtualization

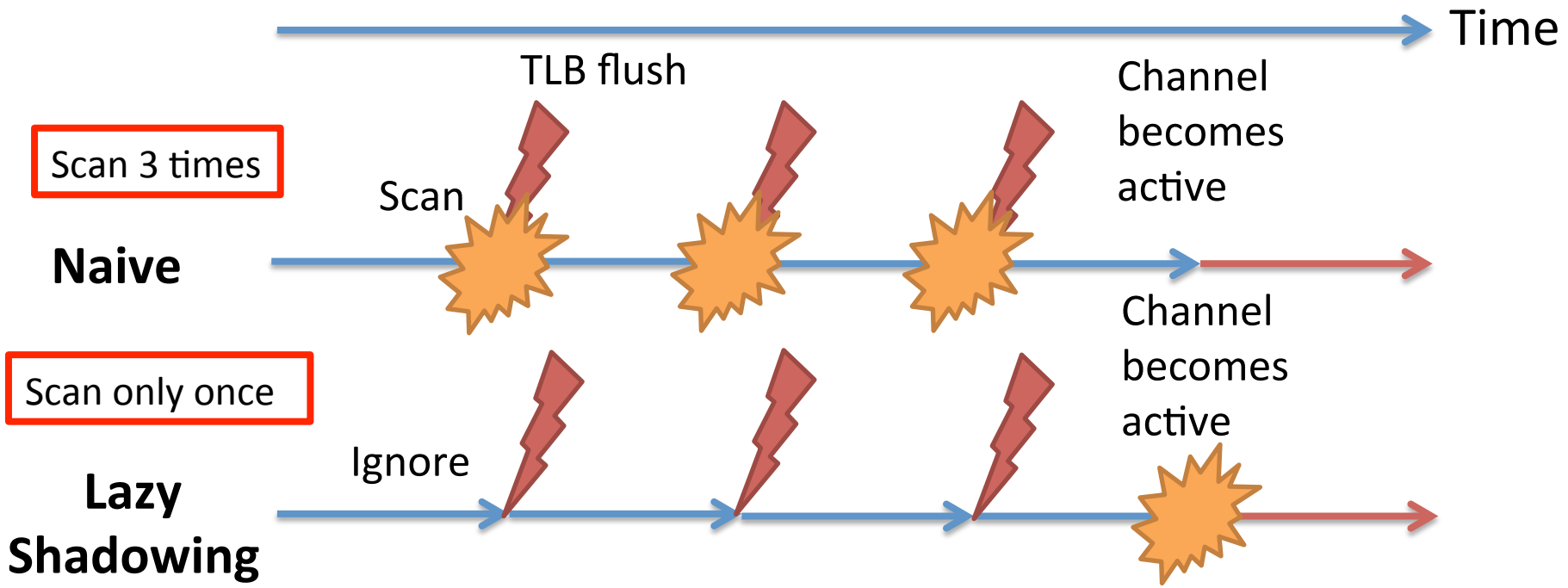
BAR Remap

- MMIO through PCIe BARs is intercepted by GPUvm
- Allow direct BAR accesses to the non-virtualization-sensitive areas



Lazy Shadowing

- Page-fault-driven shadowing cannot be applied
  - When fault occurs, computation cannot be resumed
- Scanning entire page tables incurs high overhead
- Delay the reflection to the shadow page tables until the channel is used



Para-virtualization

sslab system software

- Shadowing is still a major source of overhead
- Provide para-virtualized driver
  - Manipulate page table entries through hypercalls (similar to Xen direct-paging)
  - Provide a multicall interface that can batch several hypercalls into one (borrowed from Xen)
- Eliminate cost of scanning entire page tables

Outline

- Motivation & Goals
- GPU Internals
- Proposal: GPUvm
- Experiments
- Related Work
- Conclusion

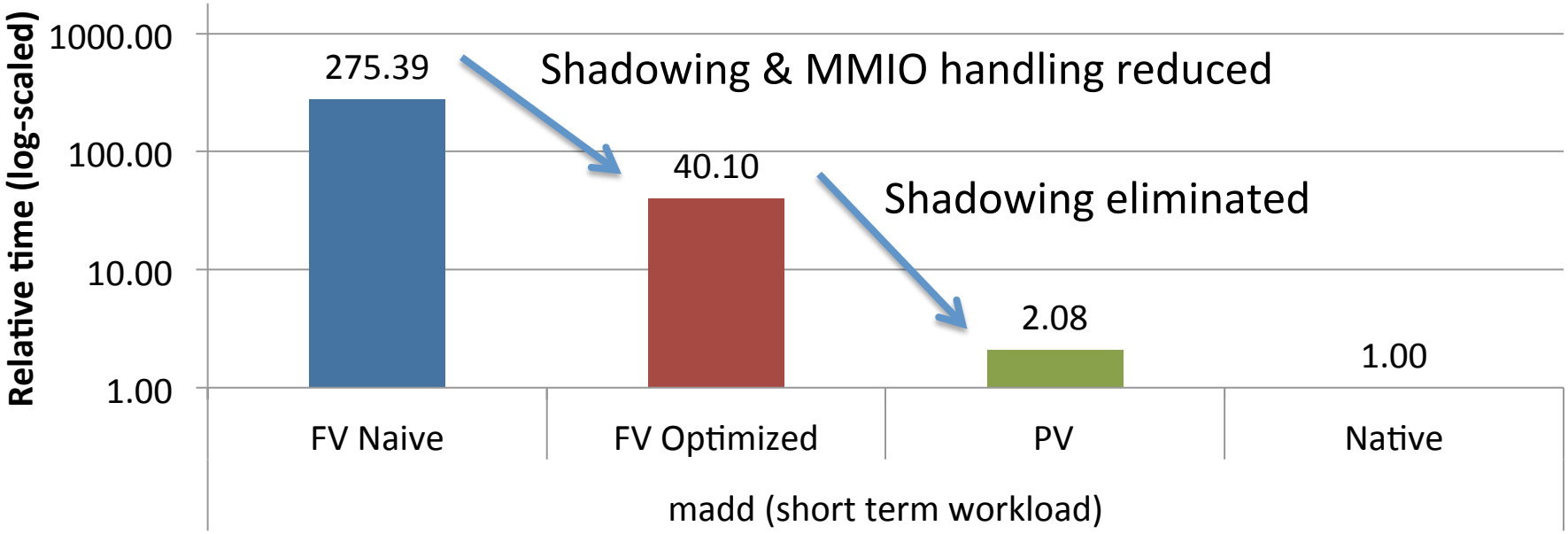
Evaluation Setup

sslab system software

- Implementation
  - Xen 4.2.0, Linux 3.6.5
  - Nouveau **[http://nouveau.freedesktop.org/]**
    - Open-source device driver for NVIDIA GPUs
  - Gdev **[Kato et al. '12]**
    - Open-source CUDA runtime
- Xeon E5-24700, NVIDIA Quadro6000 GPU
- Schemes
  - **Native**: non-virtualized
  - **FV Naive**: Full-virtualization w/o optimizations
  - **FV Optimized**: FV w/ optimizations
  - **PV**: Para-virtualization

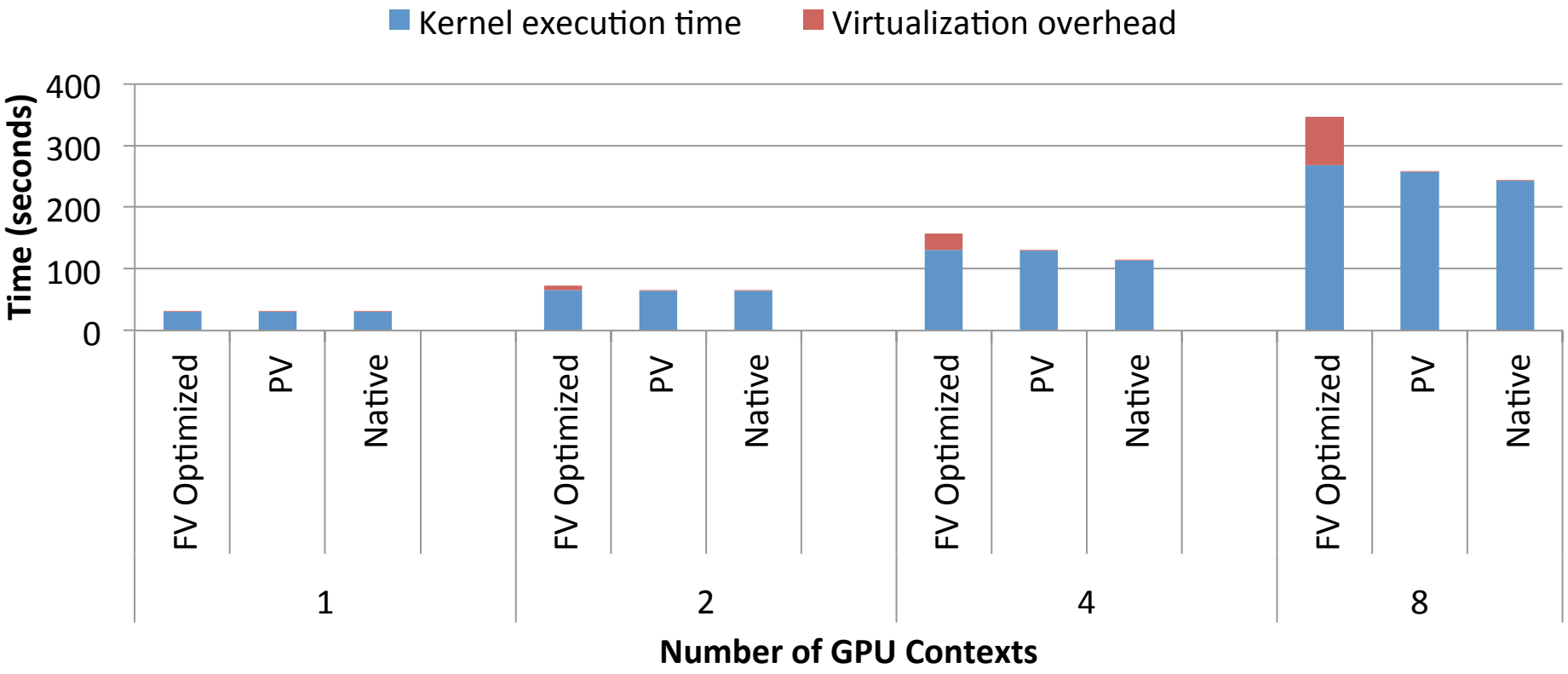
Overhead

- Significant overhead over Native
  - Can be mitigated by optimization techniques
  - PV is faster than FV since it eliminates shadowing
  - PV is still 2-3x slower than Native
    - Hypercalls, MMIO interception



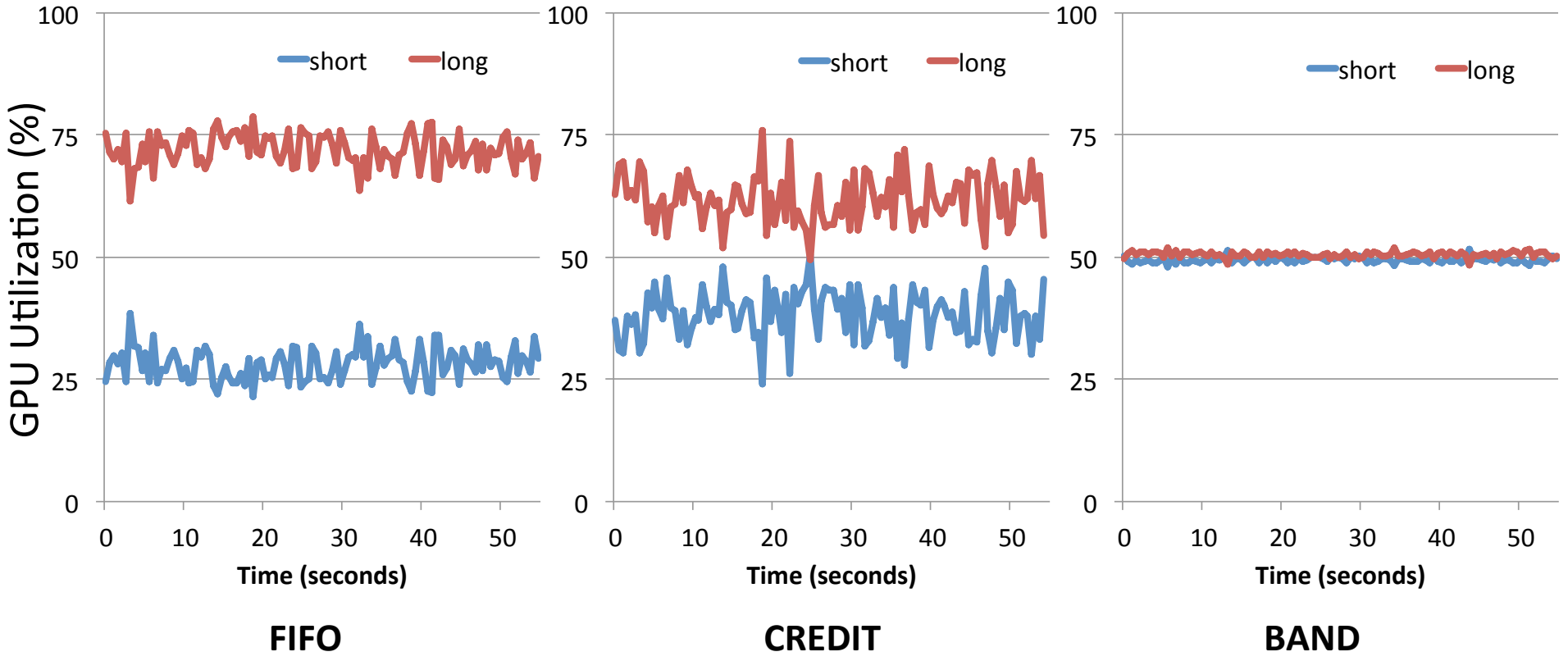
Performance at Scale

- FV incurs large overhead in 4- and 8- VM case
  - Since page shadowing locks GPU resources



Performance Isolation

- In FIFO and CREDIT a long-running task occupies GPU
- BAND achieves fair-share



Outline

- Motivation & Goals
- GPU Internals
- Proposal: GPUvm
- Experiments
- Related Work
- Conclusion

Related Work

sslab system software

- I/O pass-through
  - Amazon EC2
- API remoting
  - GViM **[Gupta et al. '09]**, vCUDA **[Shi et al. '12]**, rCUDA **[Duato et al '10]**, VMGL **[Largar-Cavilla et al. '07]**, gVirtuS **[Giunta et al. '10]**
- Para-virtualization
  - VMware SVGA2 **[Dowty et al. '09]**, LoGV **[Gottschalk et al. '10]**
- Full-virtualization
  - XenGT **[Tian et al. '14]**
  - GPU Architecture is different (Integrated Intel GPU)

Conclusion

sslab system software

- GPUvm shows the design of full GPU virtualization
  - GPU shadow page table
  - GPU shadow channel
  - GPU fair-share scheduler
- Full-virtualization exhibits non-trivial overhead
  - MMIO handling
    - Intercept TLB flush and scan page table
  - Optimizations and para-virtualization reduce this overhead
  - However still 2-3 times slower