VAMOS: Virtualization Aware Middleware

Abel Gordon[1] Muli Ben-Yehuda[1,2] Dennis Filimonov[2] Mahor Dahan[2]

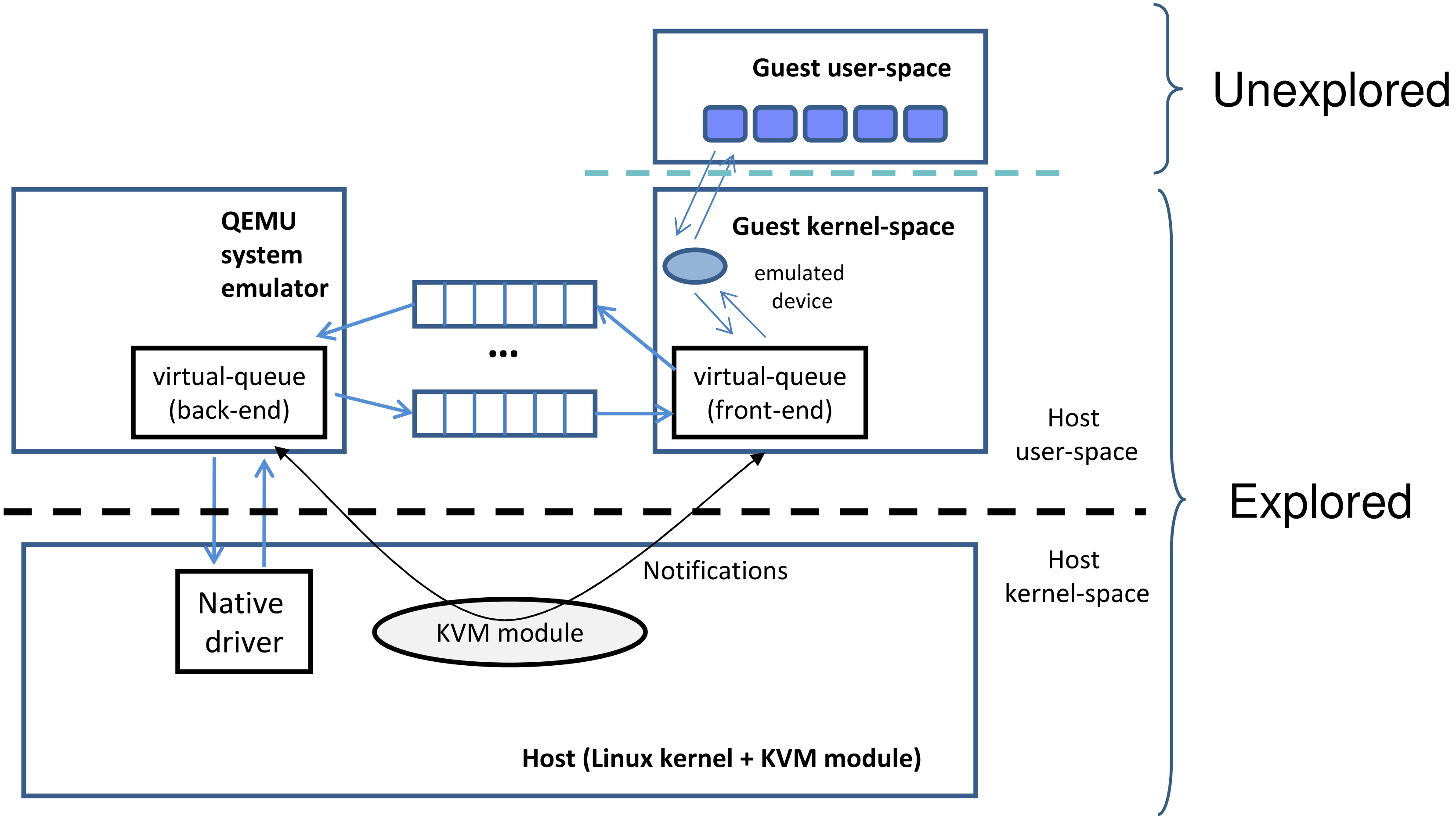
[1] IBM Research – Haifa
[2] Technion – Israel Institute of Technology

WIOV '11 June 14, 2011 • Portland, OR
3rd Workshop on I/O Virtualization
usenix

VAMOS: Virtualization Aware Middleware

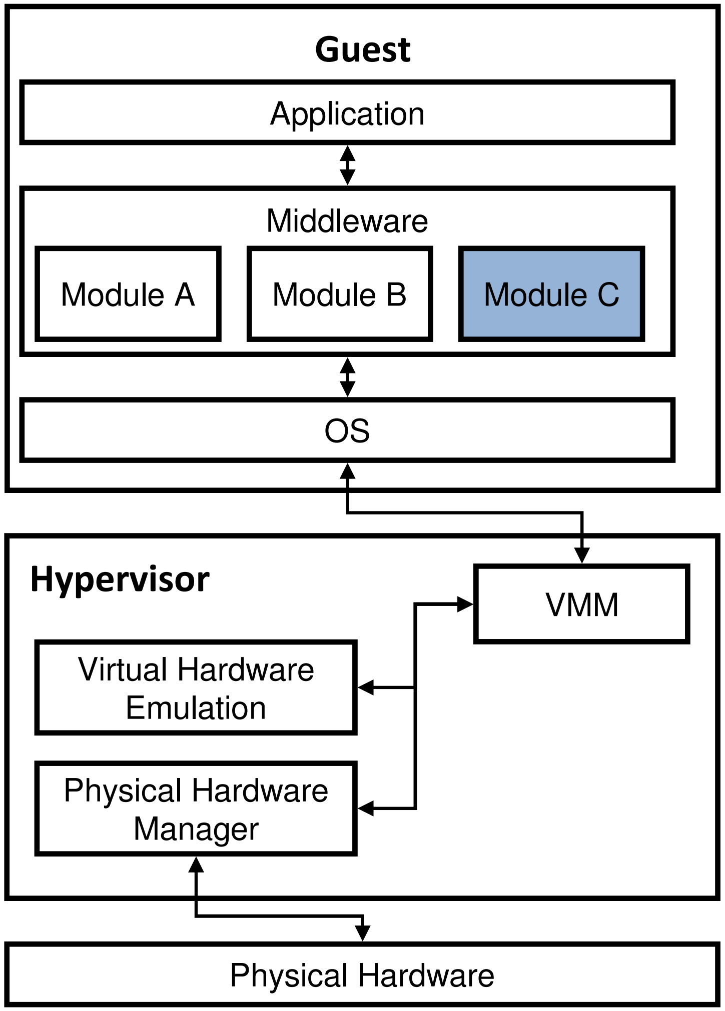
- Virtualization overhead is still high due to the transitions between the guest and the hypervisor
- Motivation: potential optimizations at the application layer have been ignored
  - Software is still built based on models that apply to non-virtual systems
  - Applications are not being adapted for the underlying “virtual” platform
  - No cooperation between the application and the hypervisor
  - Adapting the middleware such as Databases, Web Servers, Application Servers to virtualized platforms we can indirectly adapt many applications and regain lost performance

I/O virtualization with KVM, a long way down….

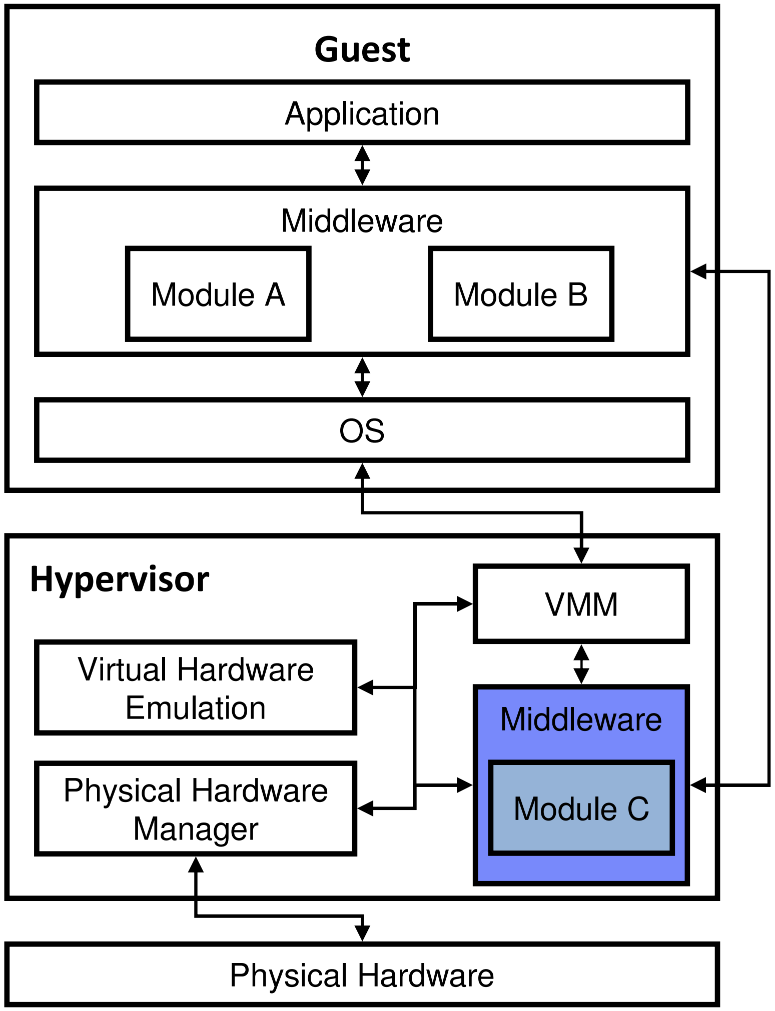


VAMOS Goals

- Reduce virtualization overhead by adding virtualization awareness to the middleware
- Avoid changes in the guest operating system
- Re-use/re-factor existent code by exploiting modularity
- Avoid software re-write/re-design

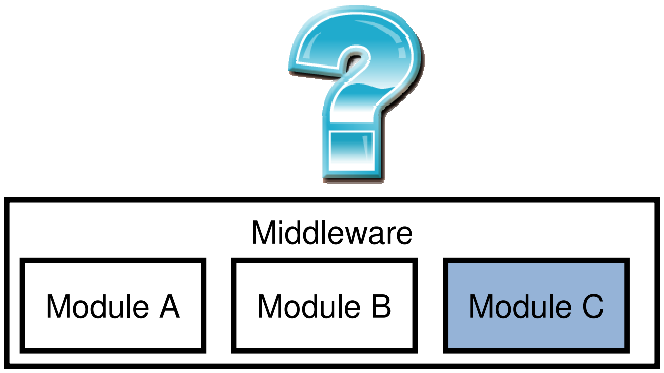


Traditional Architecture



VAMOS Architecture

Middleware Adaptation



1. Modules which interact directly with system resources, generating many transitions to the hypervisor context and requiring emulation of virtual hardware

2. Modules which can be easily re-factored into a client side running in the guest and a server side running in the host

3. Modules which do not share state with other components, avoiding data sharing and synchronization between the guest and the hypervisor.

4. Modules that do not require persisted state at the hypervisor level (do not require special handling for live-migration, checkpoint/restore)

VAMOS Requirements

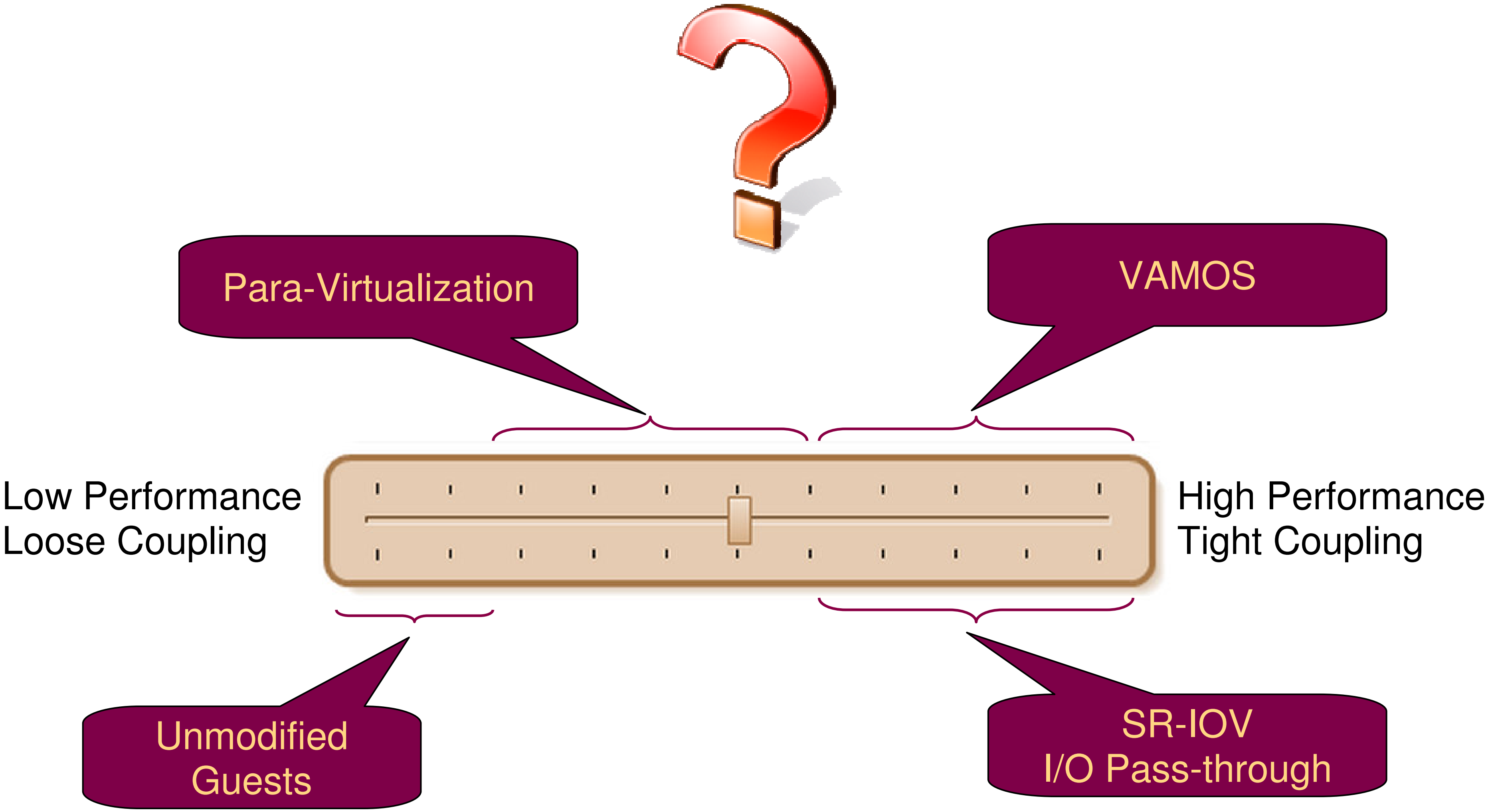
1. A runtime environment:
   - Isolated
   - At the hypervisor level
   - Executing middleware code
   - With access to physical resources, such as network devices or disk drives

   – Most hypervisors have a general purpose OS

2. A communication channel between:
   - the middleware running in the guest
   - the middleware running in the host
   - the hypervisor

   – Para-virtualization channels are commonly used

Looking for the right balance



Para-Virtualization

VAMOS

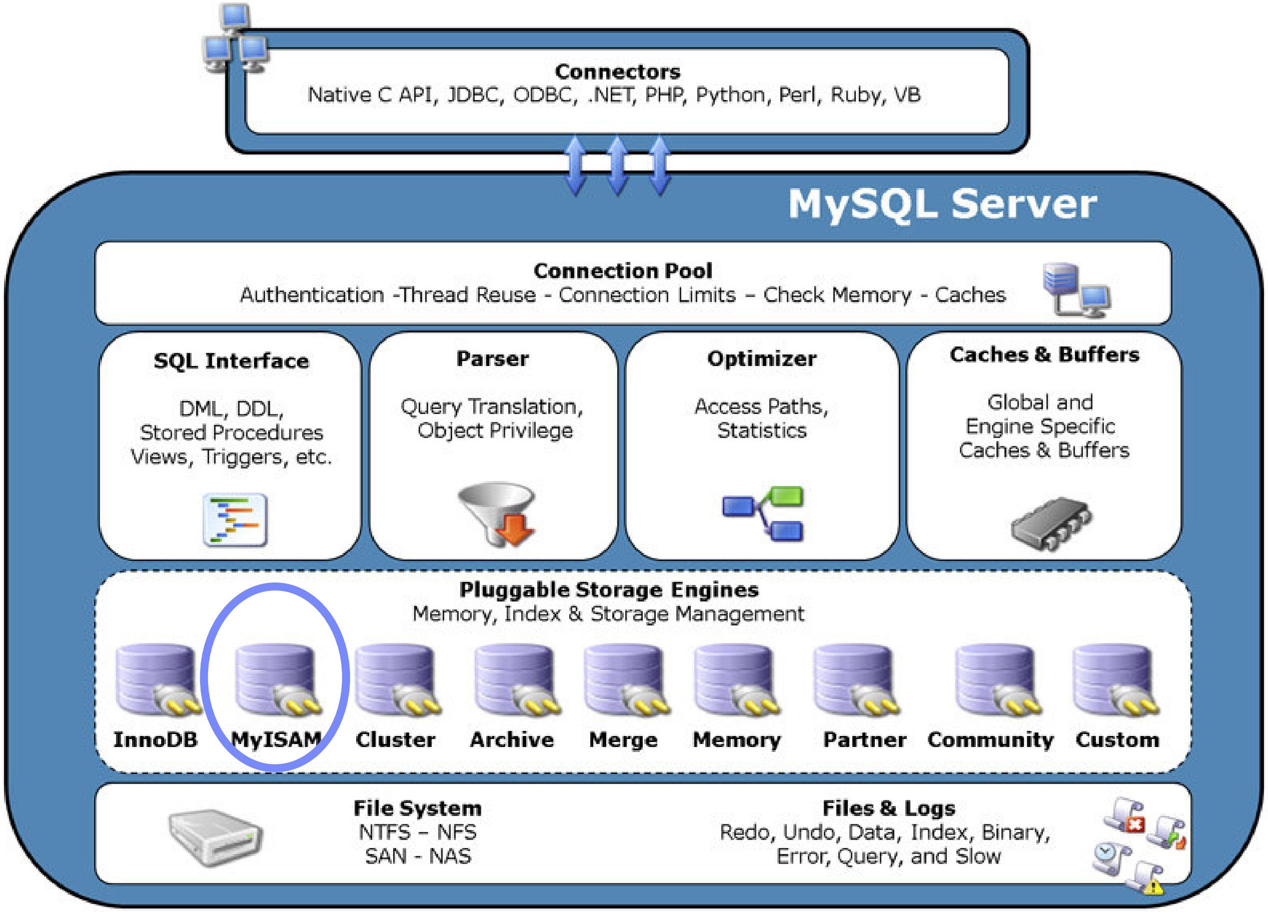
Low Performance
Loose Coupling

High Performance
Tight Coupling

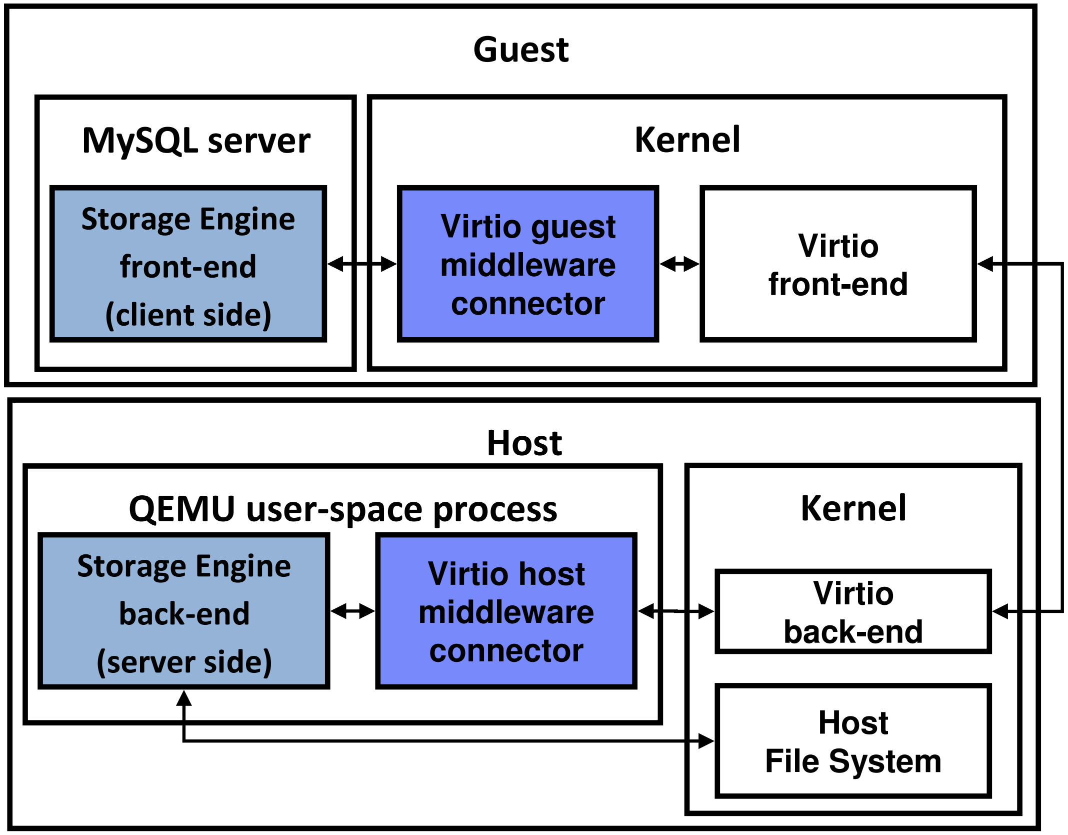
Unmodified
Guests

SR-IOV
I/O Pass-through

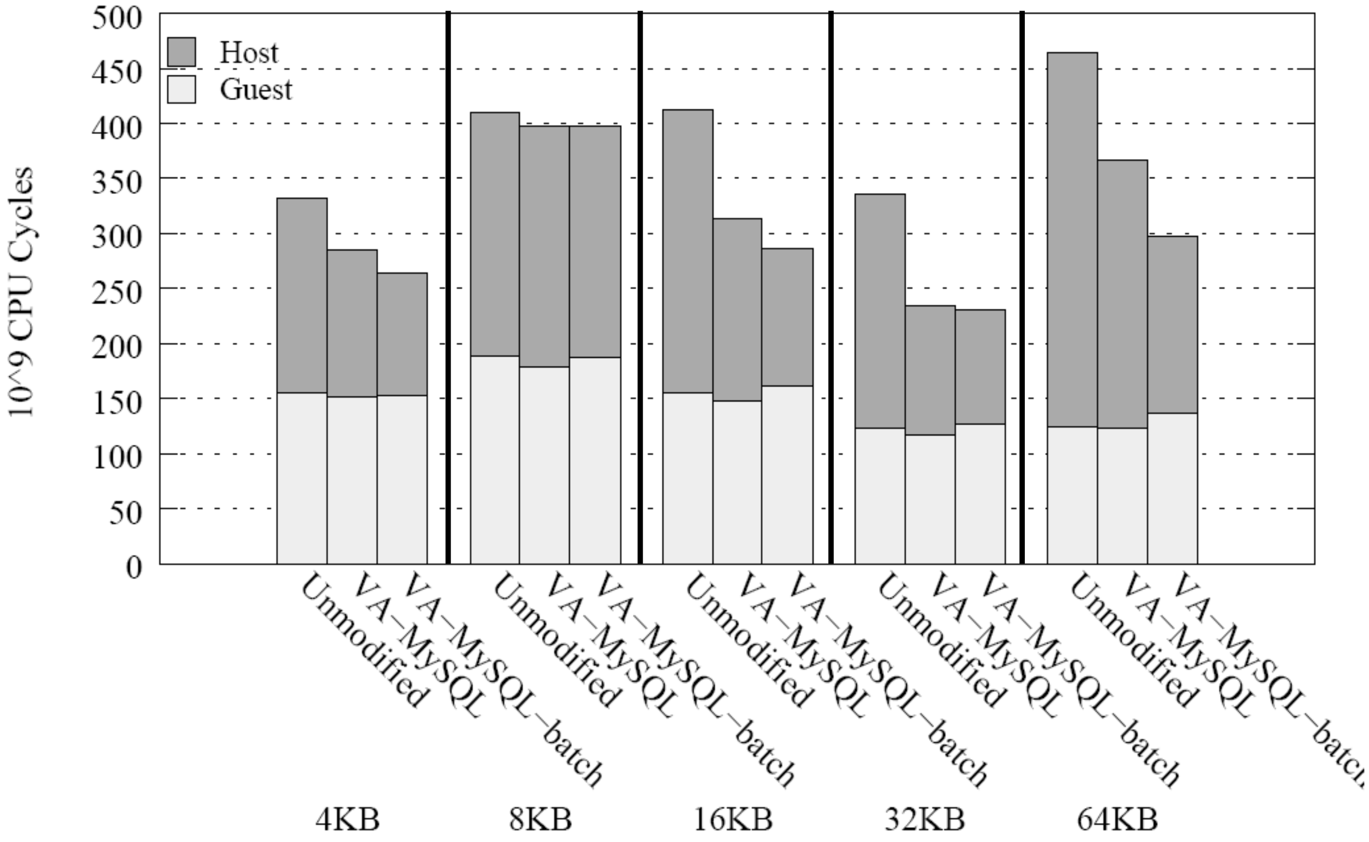
MySQL Software Architecture


Connectors
Native C API, JDBC, ODBC, .NET, PHP, Python, Perl, Ruby, VB
MySQL Server
Connection Pool
Authentication -Thread Reuse - Connection Limits – Check Memory - Caches
SQL Interface
DML, DDL, Stored Procedures Views, Triggers, etc.
Parser
Query Translation, Object Privilege
Optimizer
Access Paths, Statistics
Caches & Buffers
Global and Engine Specific Caches & Buffers
Pluggable Storage Engines
Memory, Index & Storage Management
InnoDB
MyISAM
Cluster
Archive
Merge
Memory
Partner
Community
Custom
File System
NTFS – NFS
SAN - NAS
Files & Logs
Redo, Undo, Data, Index, Binary, Error, Query, and Slow

VAMOS for MySQL


Guest
MySQL server
Storage Engine front-end (client side)
Kernel
Virtio guest middleware connector
Virtio front-end
Host
QEMU user-space process
Storage Engine back-end (server side)
Virtio host middleware connector
Kernel
Virtio back-end
Host File System

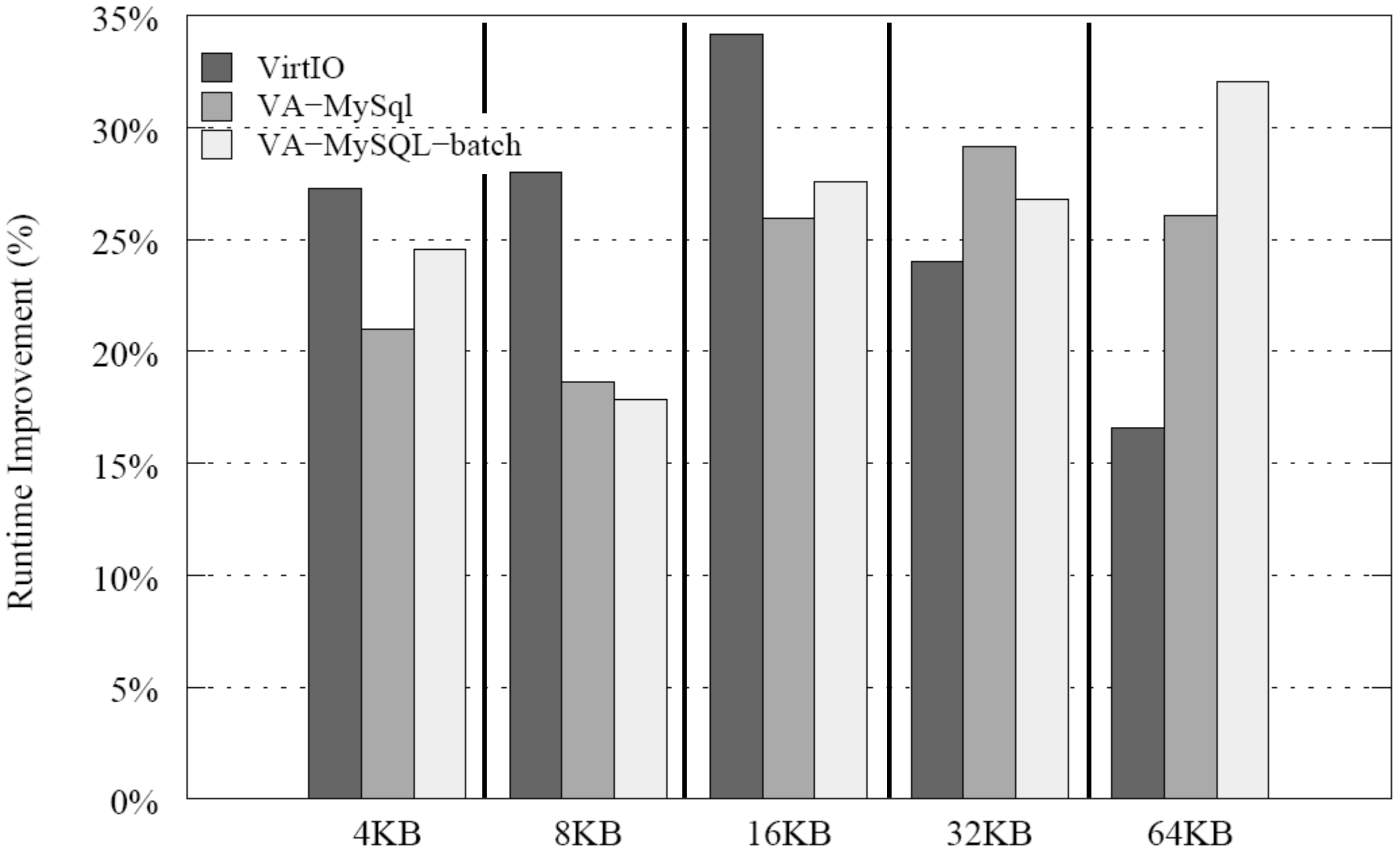
Guest/Host – Cycles Distribution



Experimental setup for different row sizes

- Guest cycles: still the same
- Host cycles: significantly reduced

Runtime Improvement



Tradeoff between amount of data and number of switches:

- VAMOS: number of switches depends on the request type
- Virtio: number of switches depends on the amount of data

Related Work

- Virtual Interface
  - Xen [Barham03]
  - HPC [Gavrilovska08]
  - Virtio [Rusell08]
- OS Interface
  - VirtFS [Jujjuri10]
  - Libra [Ammons07]
- Hardware Interface
  - SR-IOV [Dong08, Liu10]
- VAMOS takes virtualization awareness up into userspace (Middleware)

Conclusions & Future Work

- Virtualization overhead is still high due to the transitions between the guest and the hypervisor
- Running part of the middleware at the hypervisor level, VAMOS reduces the overall number of guest/hypervisor switches and improves I/O performance
- Exploiting existing modular designs and abstraction layers, middleware can be adapted to run at the hypervisor level with **modest cost**
- VAMOS presents a new design point to be considered in the [transparency vs. performance] trade-off spectrum
- Next Steps:
  - Apply VAMOS to other middleware
  - Explore additional areas such as memory over-commit
  - Analyze feasibility of building a common infrastructure shared across different middleware
  - Improve middleware isolation and security
  - Guest/Host communication optimizations
  - What can we do if we re-think the middleware from scratch ?

Questions ?