PRESENTED BY RED HAT

LEARN. NETWORK.
EXPERIENCE OPEN SOURCE.

www.theredhatsummit.com

Achieving Peak Performance from Red Hat KVM-Based Virtualization

Mark Wagner, Sanjay Rao
Principal SW Engineers, Red Hat
June 23, 2010

Overview

This talk will cover a wide range of topics related to KVM performance

- RHEL5 and RHEL6
  - RHEL6 data is NOT final and subject to change
- Command line vs libvirt
  - Use libvirt where possible, not all features in all releases
- We will not cover the RHEV products
  - Some stuff may apply but...

What we tell You

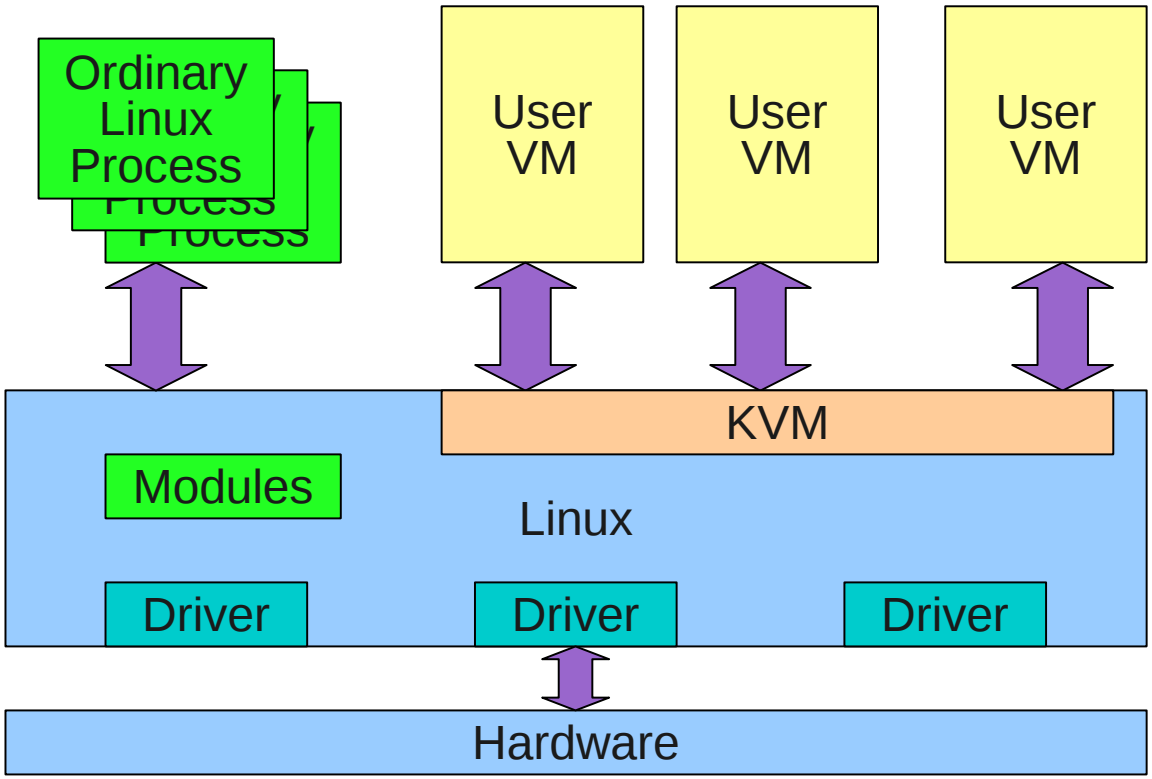
- Some of the Basics
- A quick, high level overview of KVM
- Some block IO basics
  - Some examples using database workloads
- A deeper dive into networking
  - Virtio-net, vhost-net, SR-IOV
- Huge Pages
- Non-uniform Memory Allocation (NUMA) and affinity settings
- Wrap up

A quick KVM primer

Quick Overview – KVM Architecture

- Guests run as a process in userspace on the host
- Guests inherits features from the kernel (NUMA, huge pages, support for new hardware)
- Disk and Network IO through host (most of the time)
  - IO settings in host can make a big difference in guest IO performance
  - Need to understand host buffer caching
    - Proper settings to achieve true direct IO from the guest
    - Deadline scheduler (on host) typically gives best performance
- Network typically goes through a software bridge
- Device assignment can help with network performance

Quick Overview - KVM Architecture



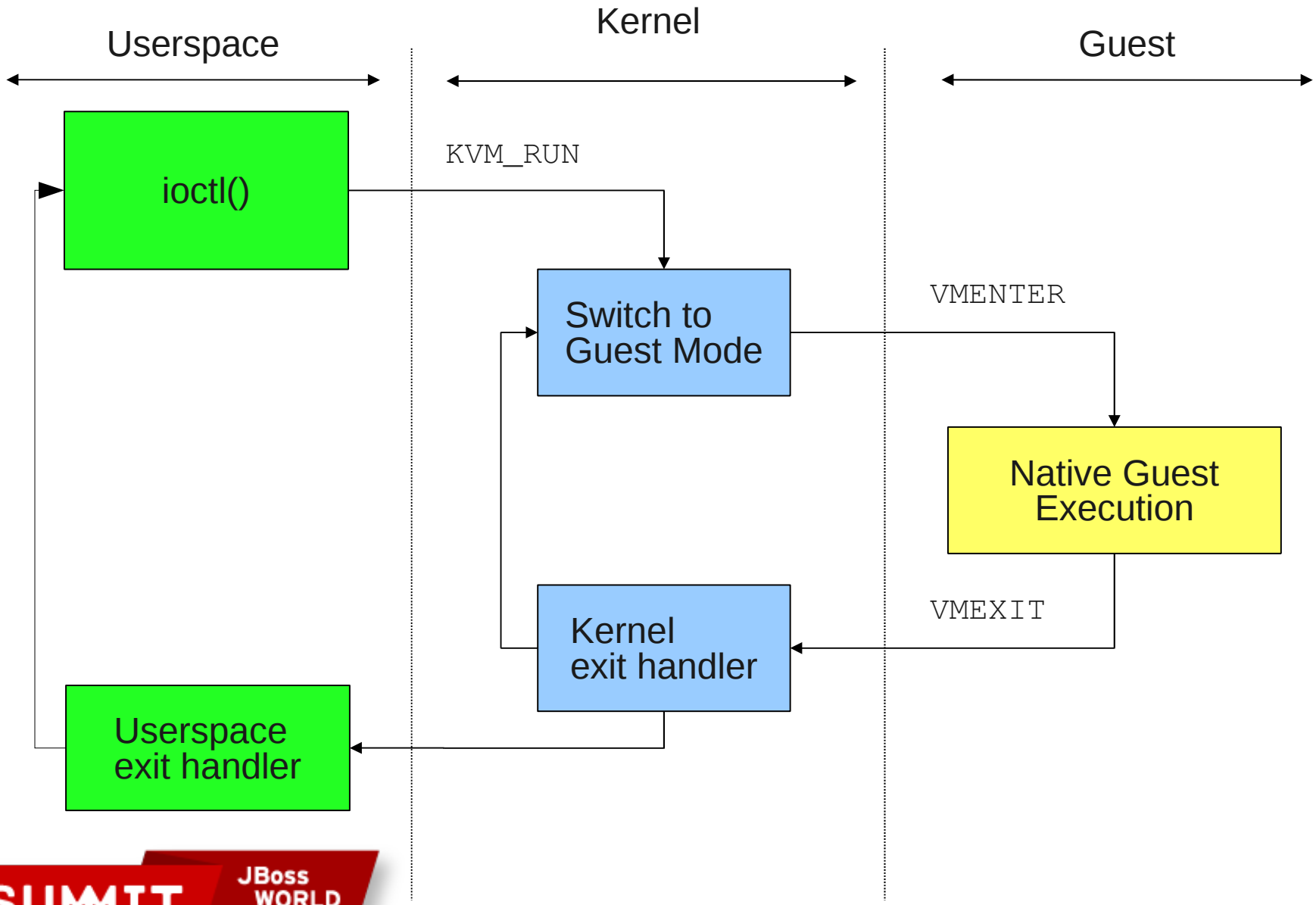
I/O – virtio

- Most devices emulated in userspace
  - With fairly low performance
- Paravirtualized I/O is the traditional way to accelerate I/O
- Virtio is a framework and set of drivers:
  - A hypervisor-independent, domain-independent, bus-independent protocol for transferring buffers
  - A binding layer for attaching virtio to a bus (e.g. pci)
  - Domain specific guest drivers (networking, storage, etc.)
    - RHEL 3/4/5, Windows XP/Server 2003/Server 2008
  - Hypervisor specific host support

KVM Execution model

- Three modes for thread execution instead of the traditional two:
  - User mode
  - Kernel mode
  - Guest mode
- A virtual CPU is implemented using a Linux thread
  - The Linux scheduler is responsible for scheduling a virtual CPU, as it is a normal thread
- Understanding these help when tuning

KVM Execution Model



I/O – virtio

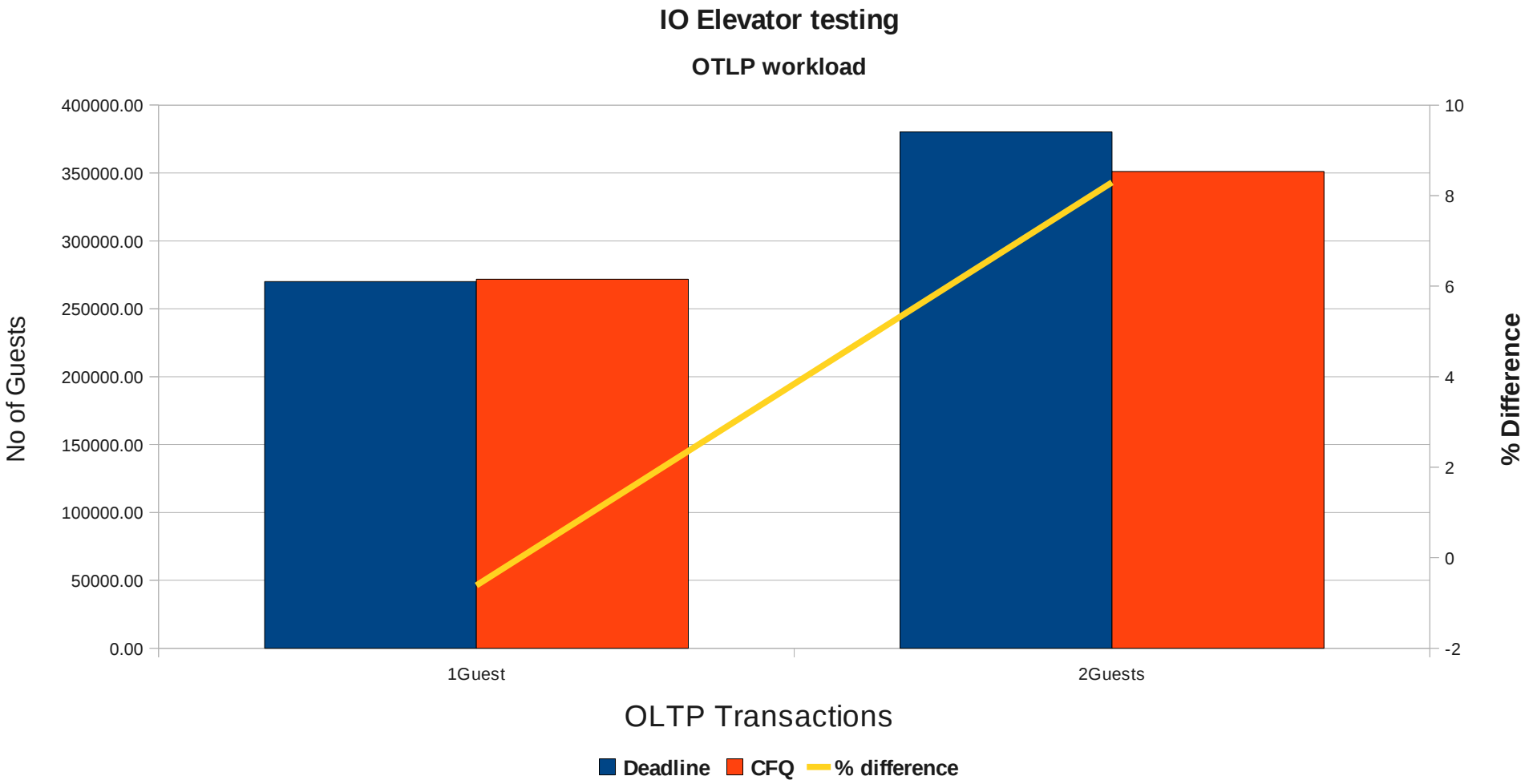
- Most devices emulated in userspace
  - With fairly low performance
- Paravirtualized I/O is the traditional way to accelerate I/O
- Virtio is a framework and set of drivers:
  - A hypervisor-independent, domain-independent, bus-independent protocol for transferring buffers
  - A binding layer for attaching virtio to a bus (e.g. pci)
  - Domain specific guest drivers (networking, storage, etc.)
    - RHEL 3/4/5/6, Windows XP/Server 2003/Server 2008
  - Hypervisor specific host support

Disk IO

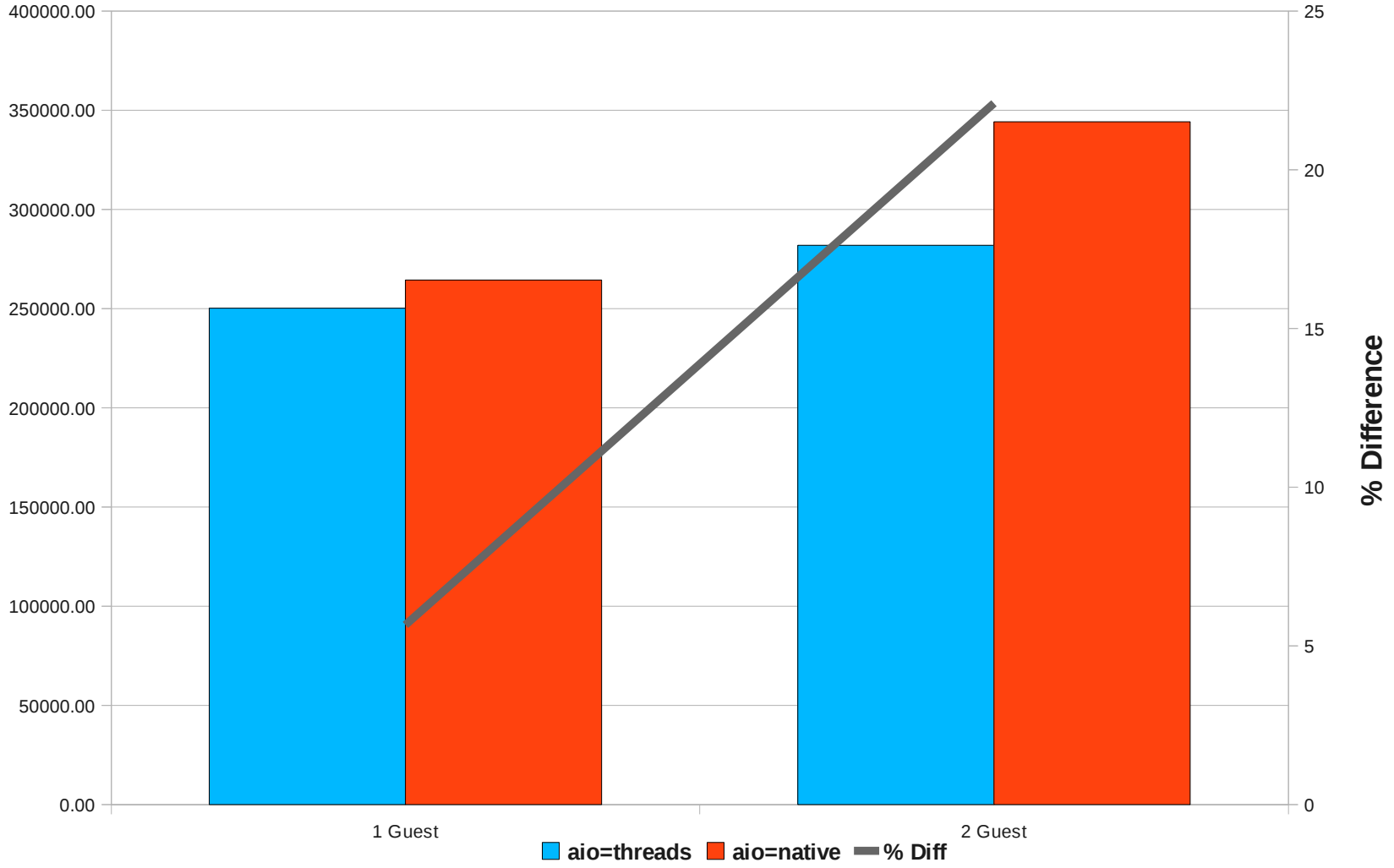
IO Elevators

- Deadline
  - Two queues – one for read and one for write
  - IOs dispatched based on time spent in queue
- CFQ (Completely Fair Queuing)
  - Per process queue
  - Each process queue gets a fixed time slice (based on process priority – to maintain fairness)
- How to configure
  - Boot command line (elevator=deadline/cfq)
  - echo “deadline” > /sys/class/block/sda/queue/scheduler

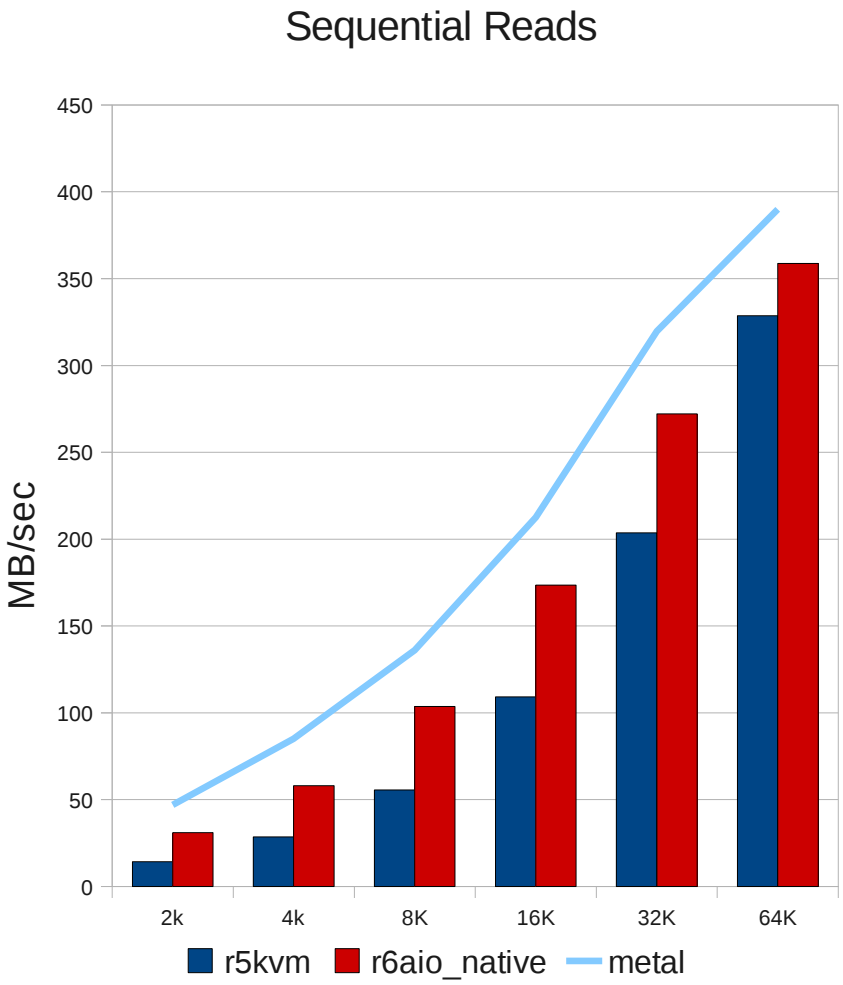
Performance Differences based on IO Elevators

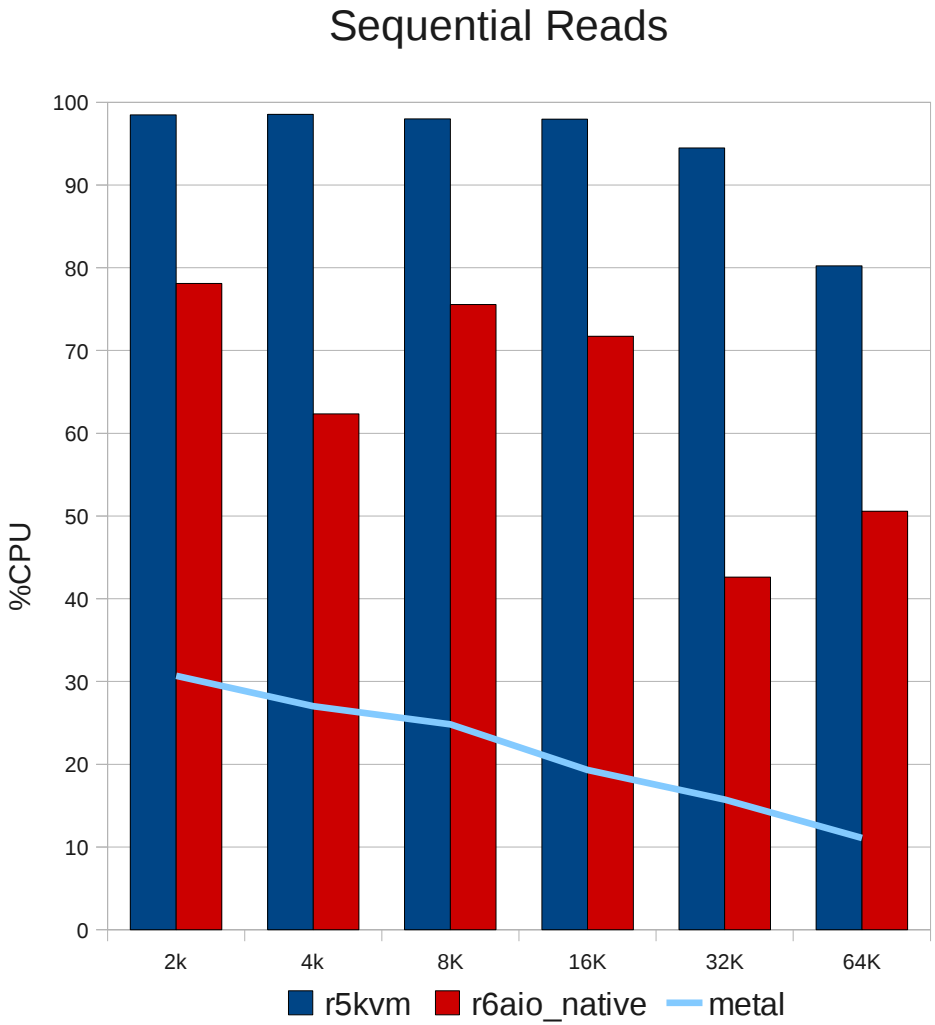


Multi guest database testing with different AIO settings (new with RHEL6)

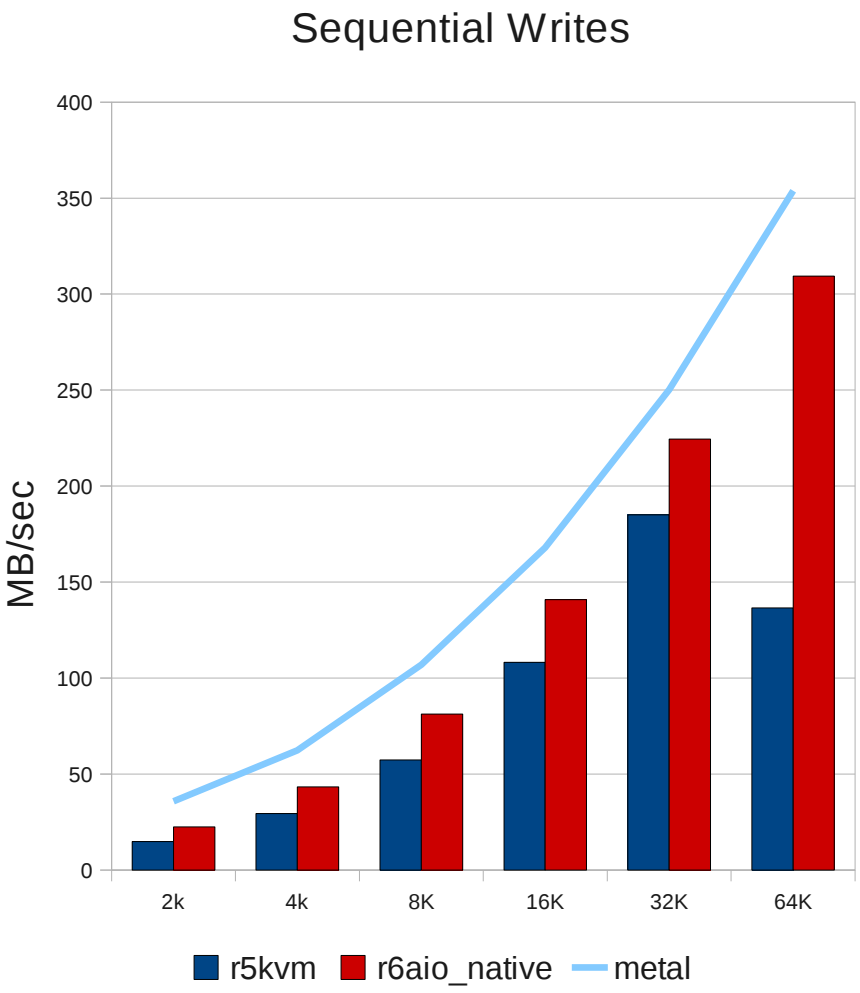


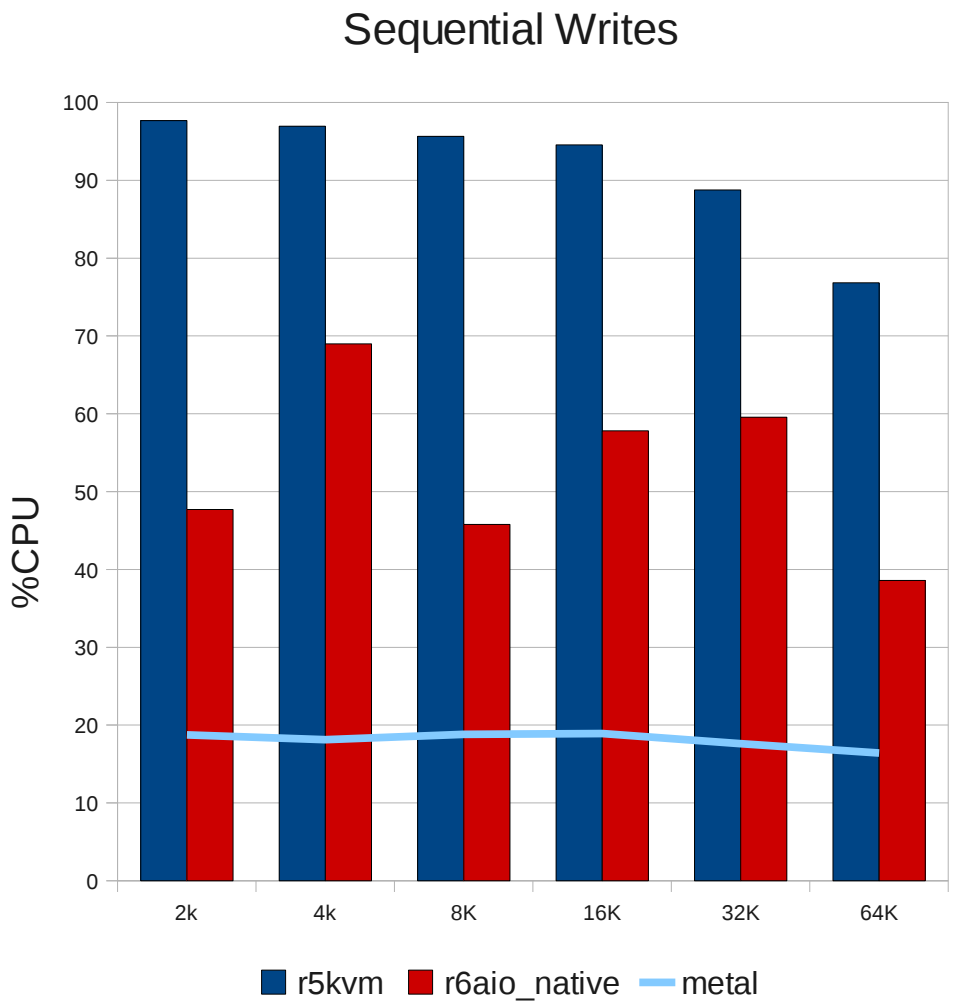
KVM Performance – RHEL6 aio=kernel Win2k8 Intel 24cpu, 64GB, FC IOmeter





KVM Performance – RHEL6 aio=kernel Win2k8 Intel 24cpu, 64GB, FC IOmeter



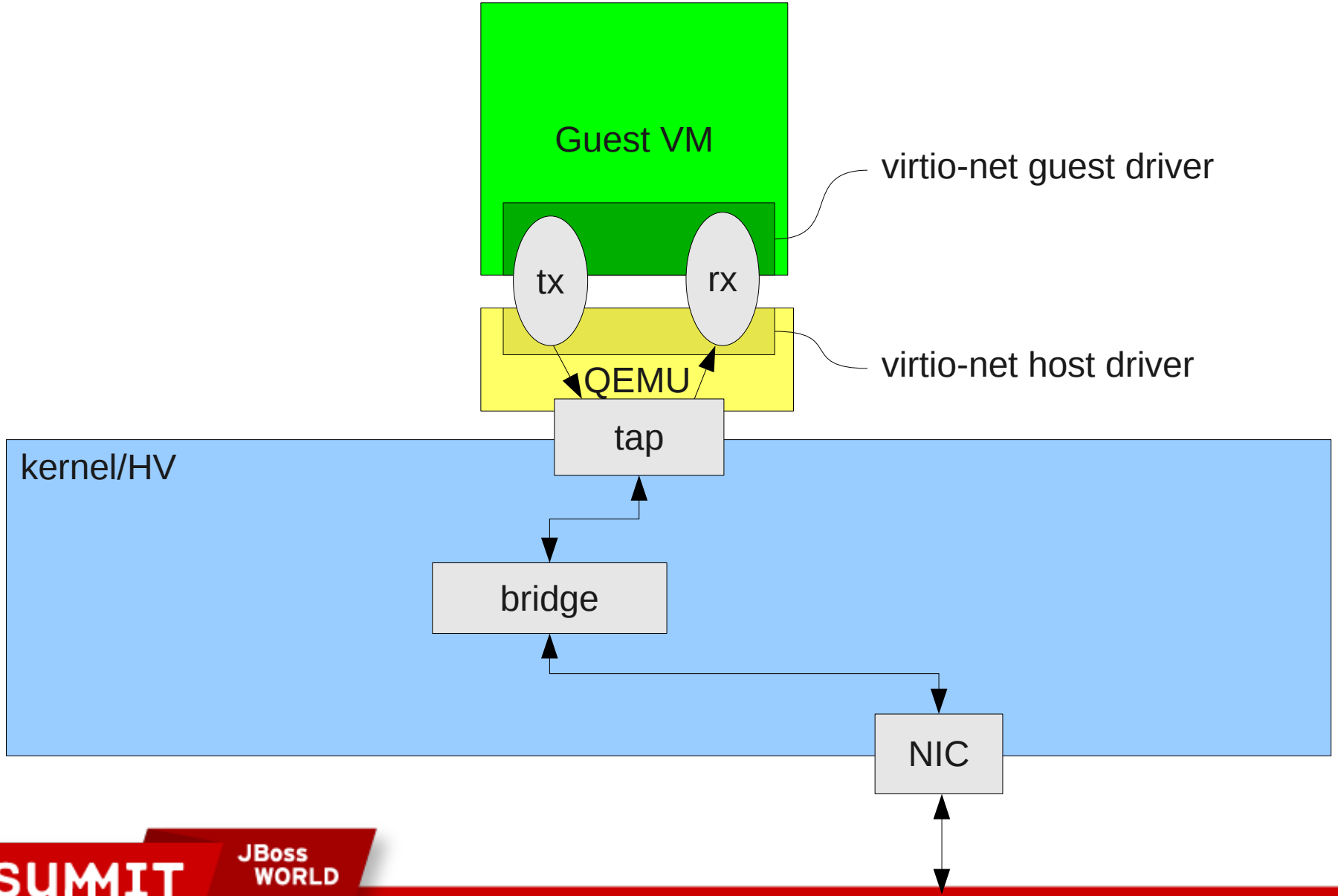


Networking

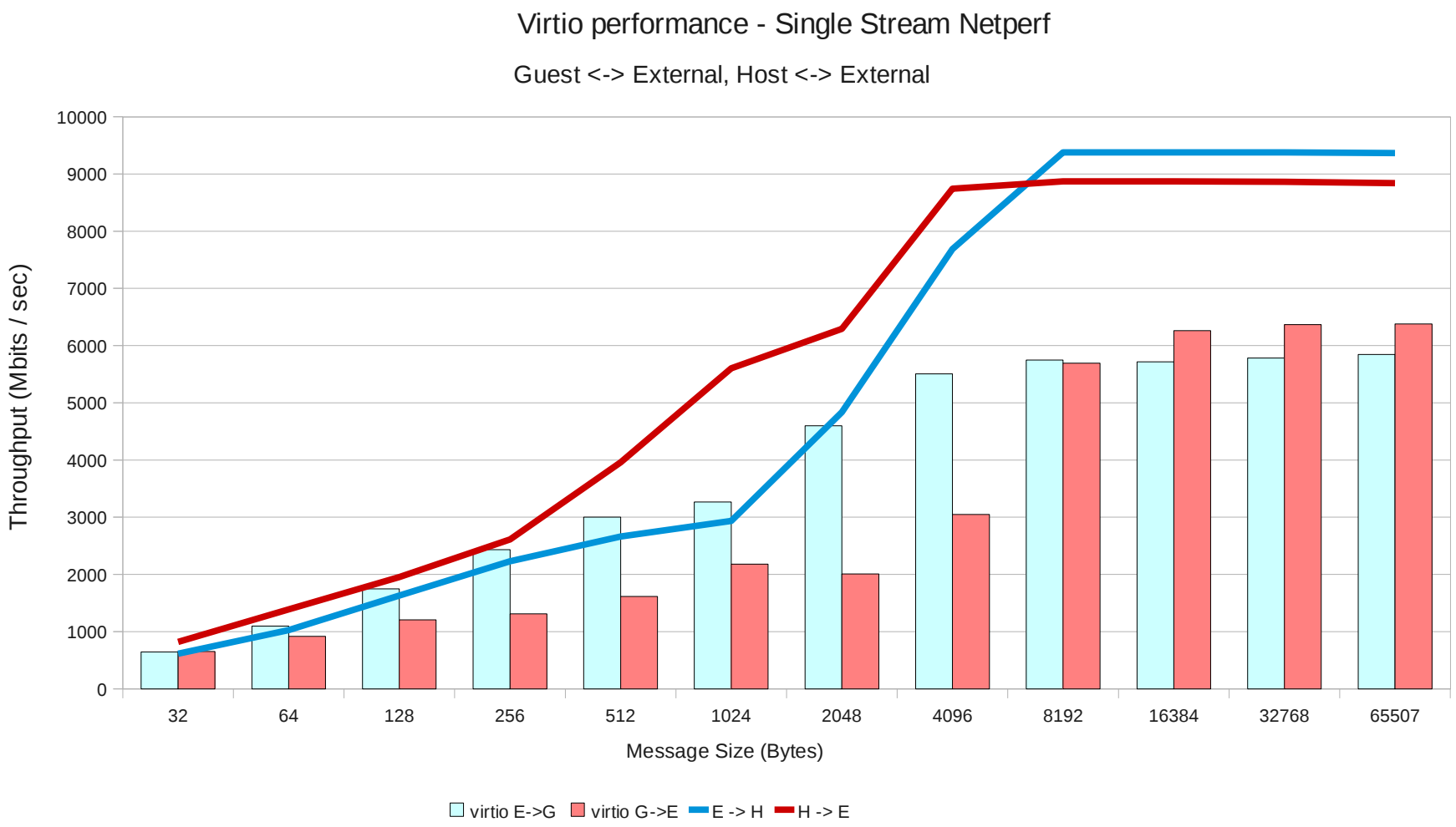
Virtio net

- Provides acceptable performance
- Typically via a bridge / tap device
  - Bridge is shared across multiple guests
  - Throughput is acceptable
  - Latency is not so good
- Changes for RHEL6
  - Moving to vhost-net
  - If you use scripts, you may need to modify them

virtio network architecture



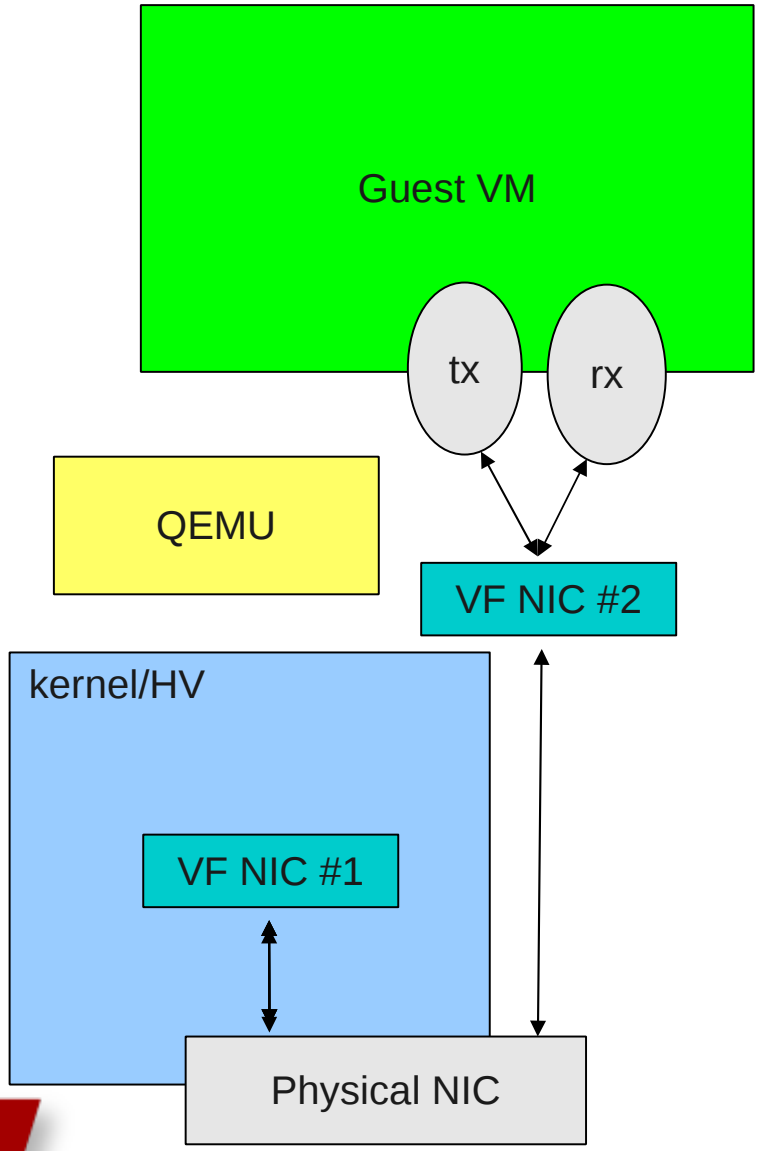
virtio data


Virtio performance - Single Stream Netperf
Guest <-> External, Host <-> External
Throughput (Mbits / sec)
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65507
Message Size (Bytes)
virtio E->G
virtio G->E
E -> H
H -> E

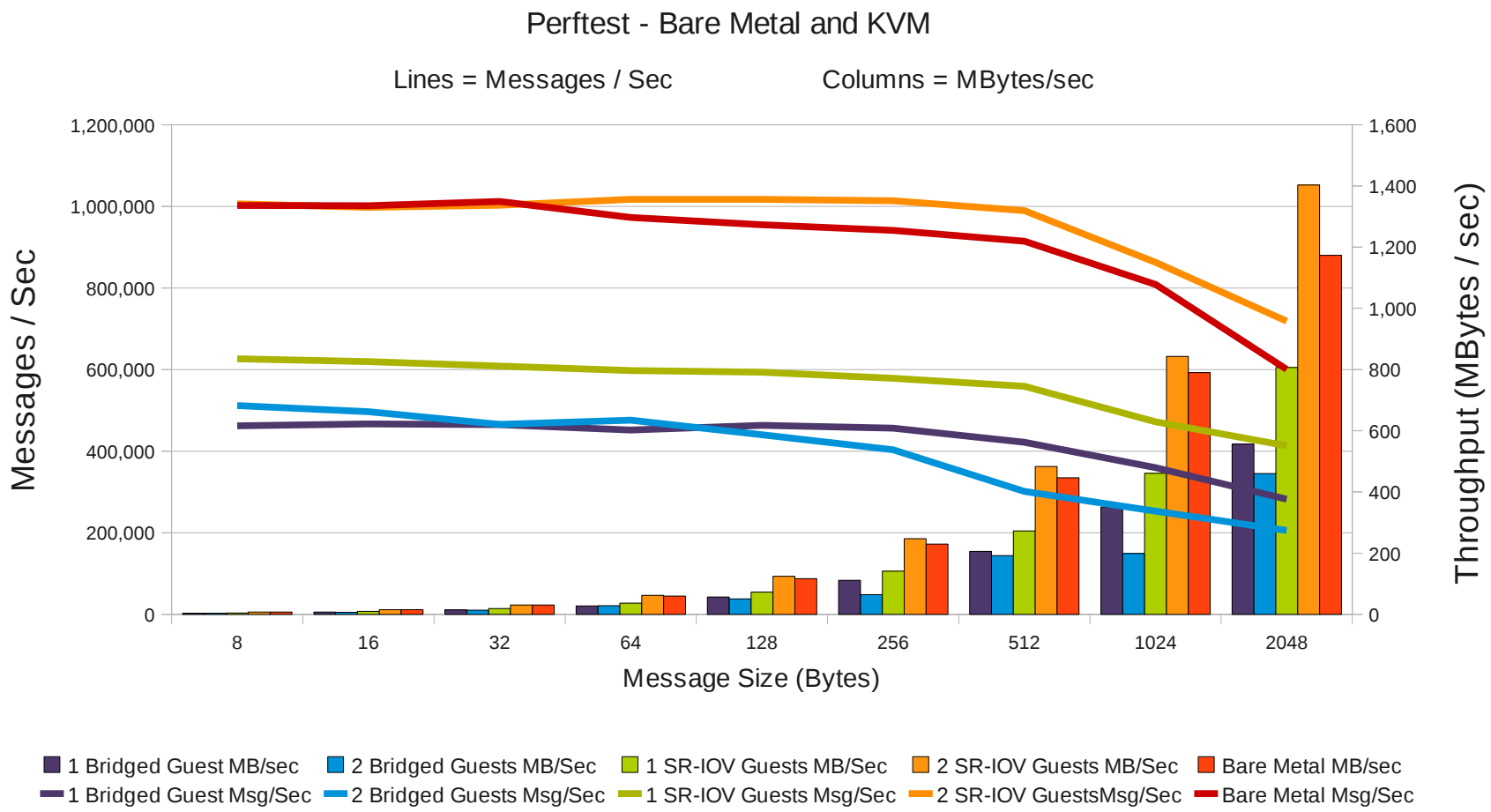
Device Assignment / SR-IOV

- Big win in lowering latency and improving throughput
- Essentially allows device to be accessed from guest
- First vendor to supply this
- Need driver / HW that supports functionality
  - Only a few drivers in RHEL5.5
  - Additional drivers / HW coming in RHEL6

PCI device assignment network (vt-d/SR-IOV)



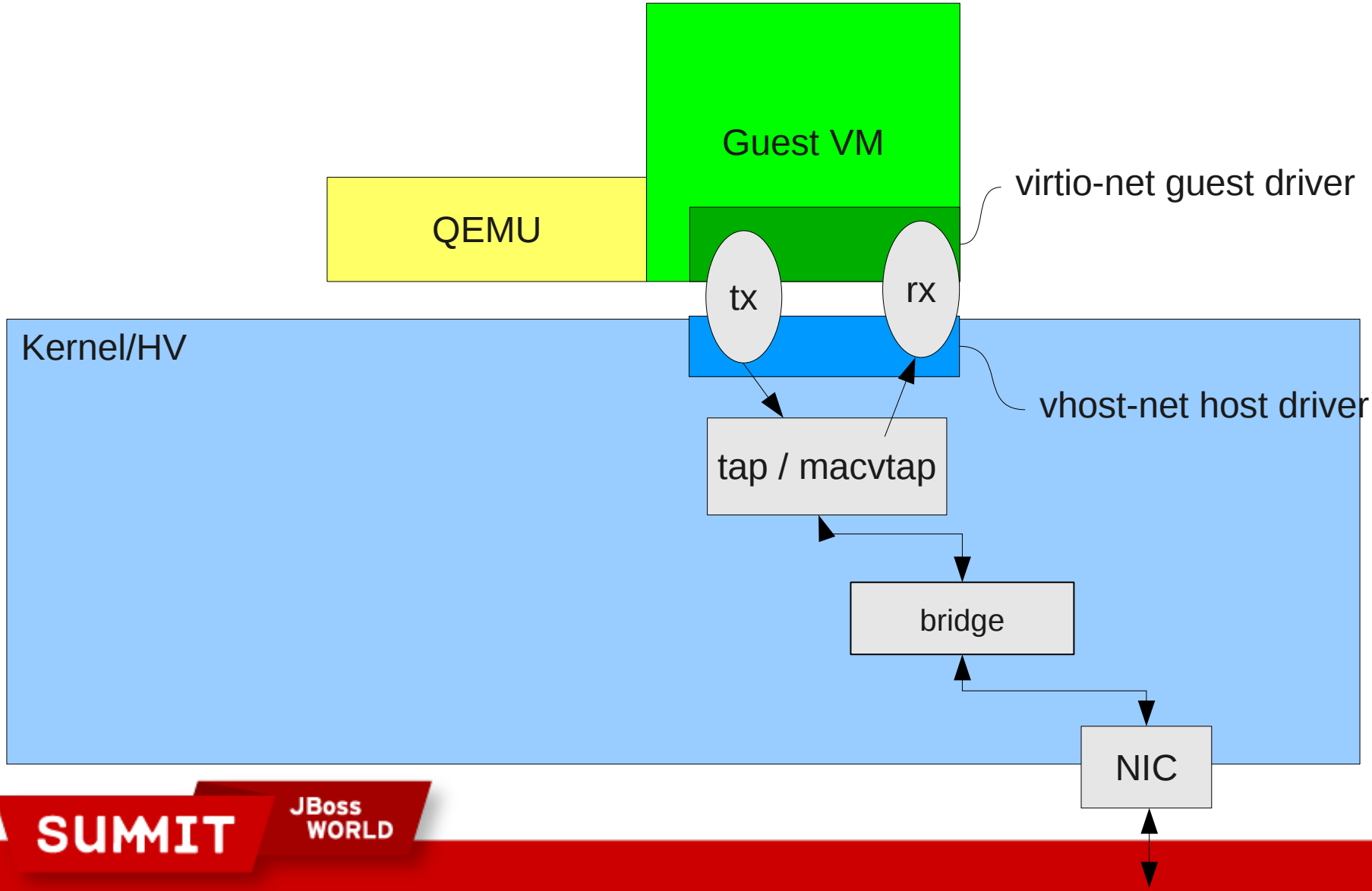
SR-IOV vs Bridged Performance



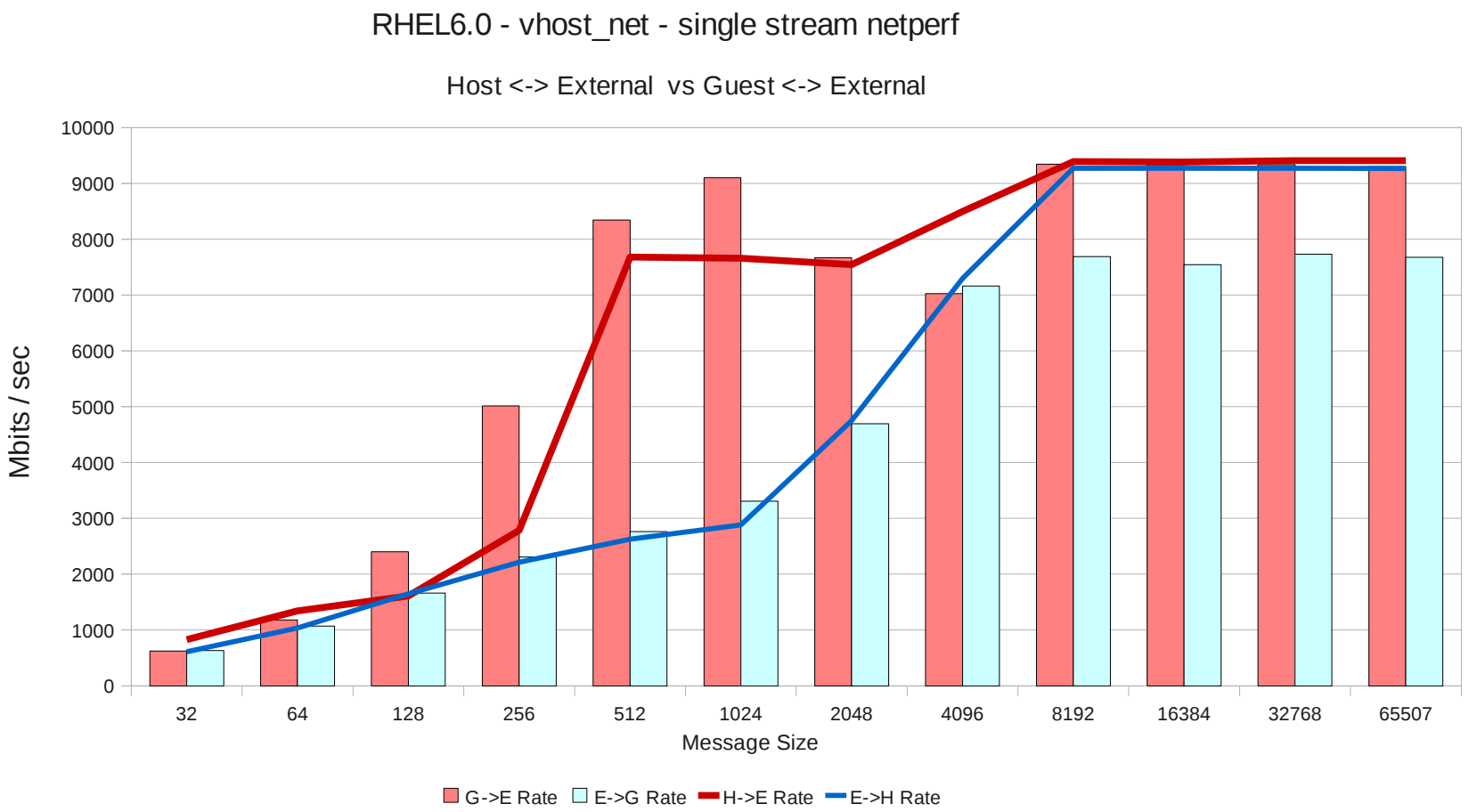
Vhost-net

- Moves host side driver from user space to kernel
  - Less context switching
  - Low latency
  - MSI
  - One less copy

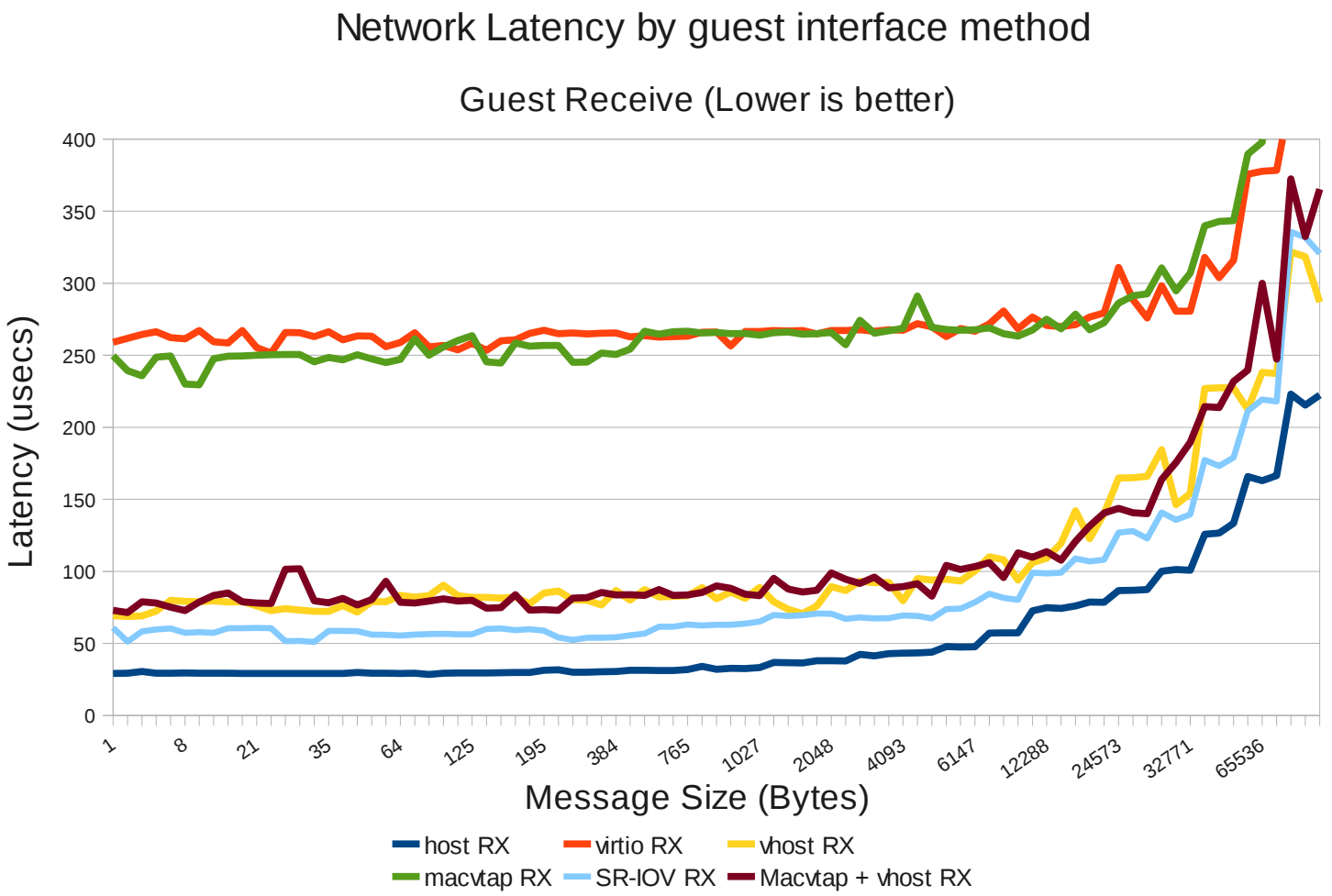
In-Kernel vhost-net architecture (RHEL6)



Vhost vs virtio data



Latency comparison – RHEL 6



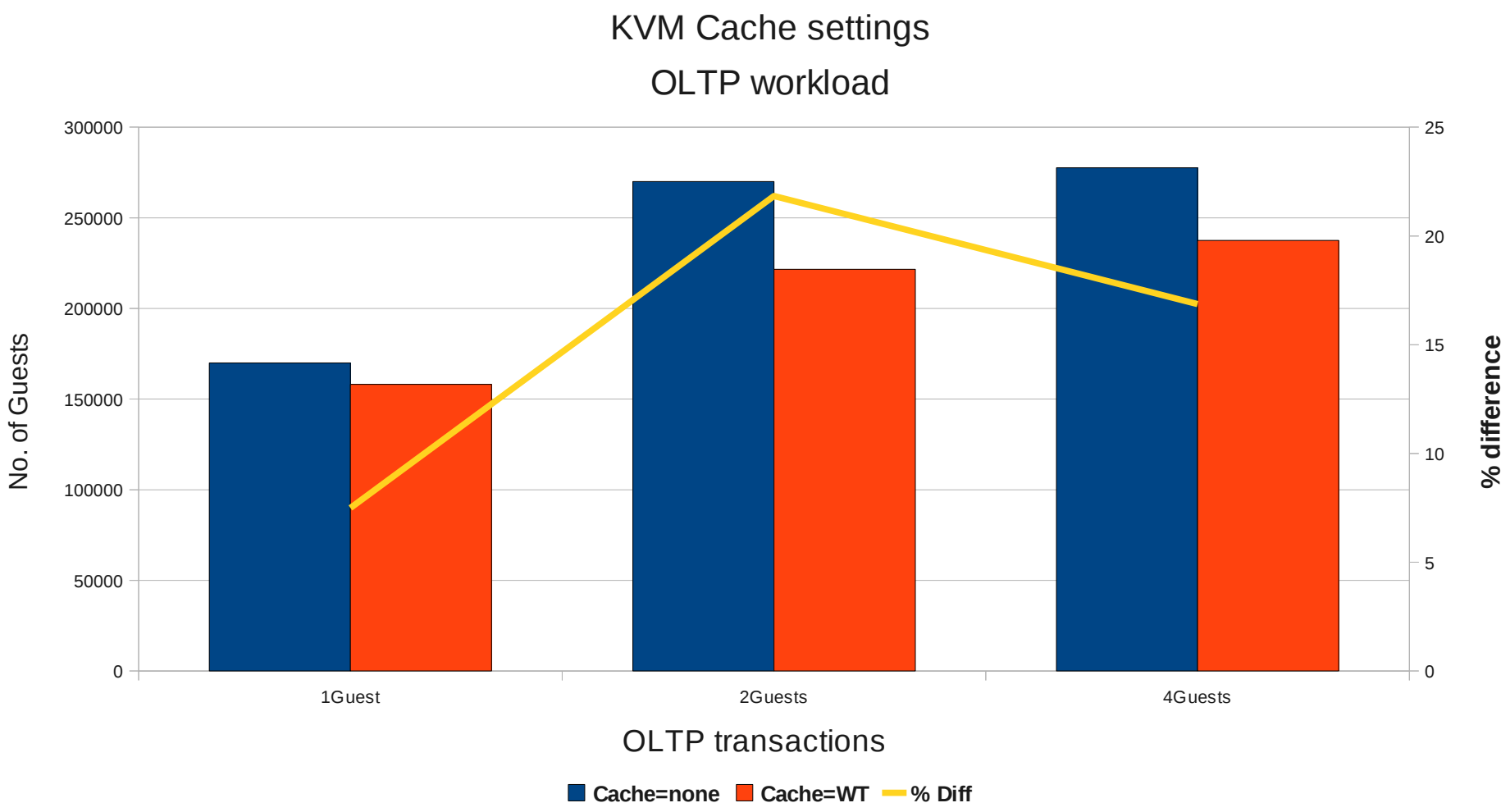
IO-Cache

KVM IO Cache Settings

- Cache=none
  - IO from the guest in not cached
- Cache=writethrough
  - IO from the guest is cached and written through on the host
  - Potential scaling problems with this option with multiple guests (host cpu used to maintain cache)
- Cache=writeback
  - Not supported

- Configure IO-Cache per disk in qemu command line or libvirt

Effect of IO Cache settings on Guest performance

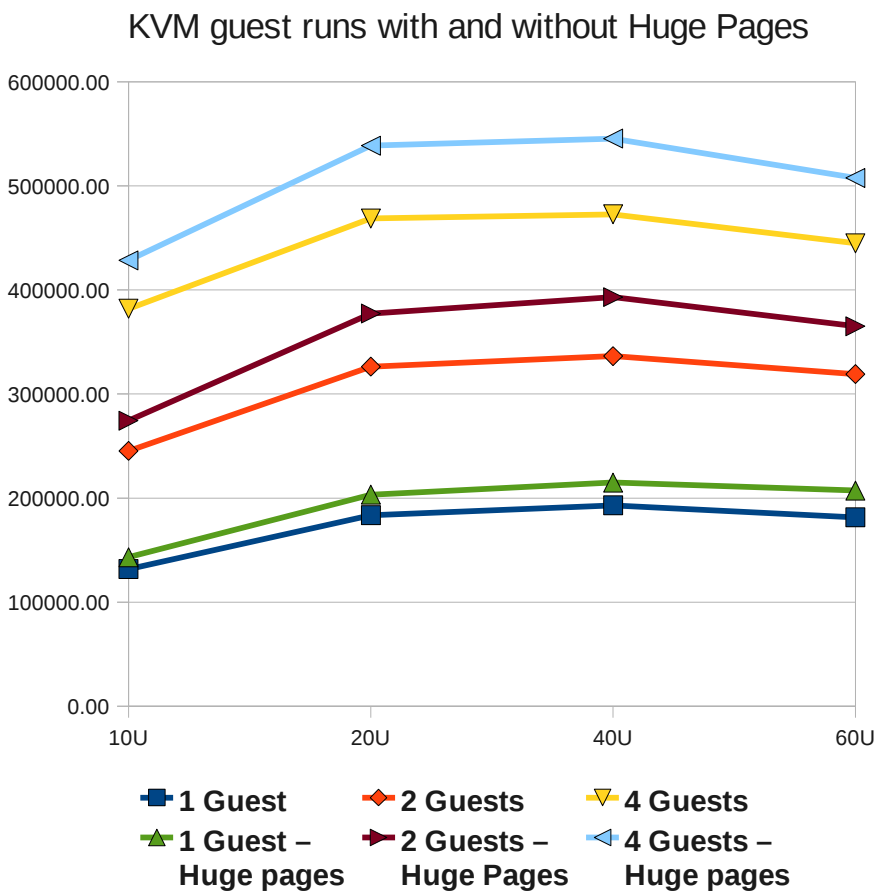


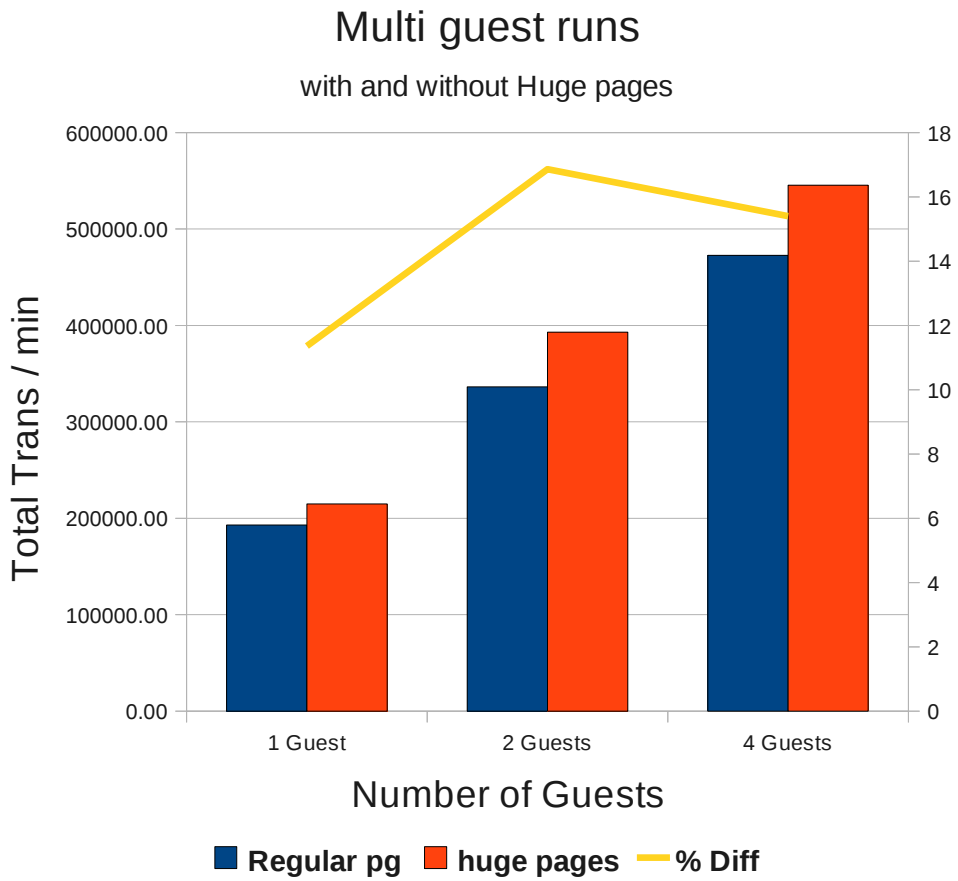
Huge Pages

Understanding Hugepages

- 2M pages vs 4K standard Linux page size
- Virtual to physical page map is 512 times smaller
- TLB cache can map more memory resulting in fewer cache misses
- Huge pages pinned
- Configuring huge pages (4G memory of huge pages)
  - echo 2048 > /proc/sys/vm/nr_hugepages
  - vi /etc/sysctl.conf (vm.nr_hugepages = 2048)

AMD – Magny Cours – RHEL5.5 – KVM





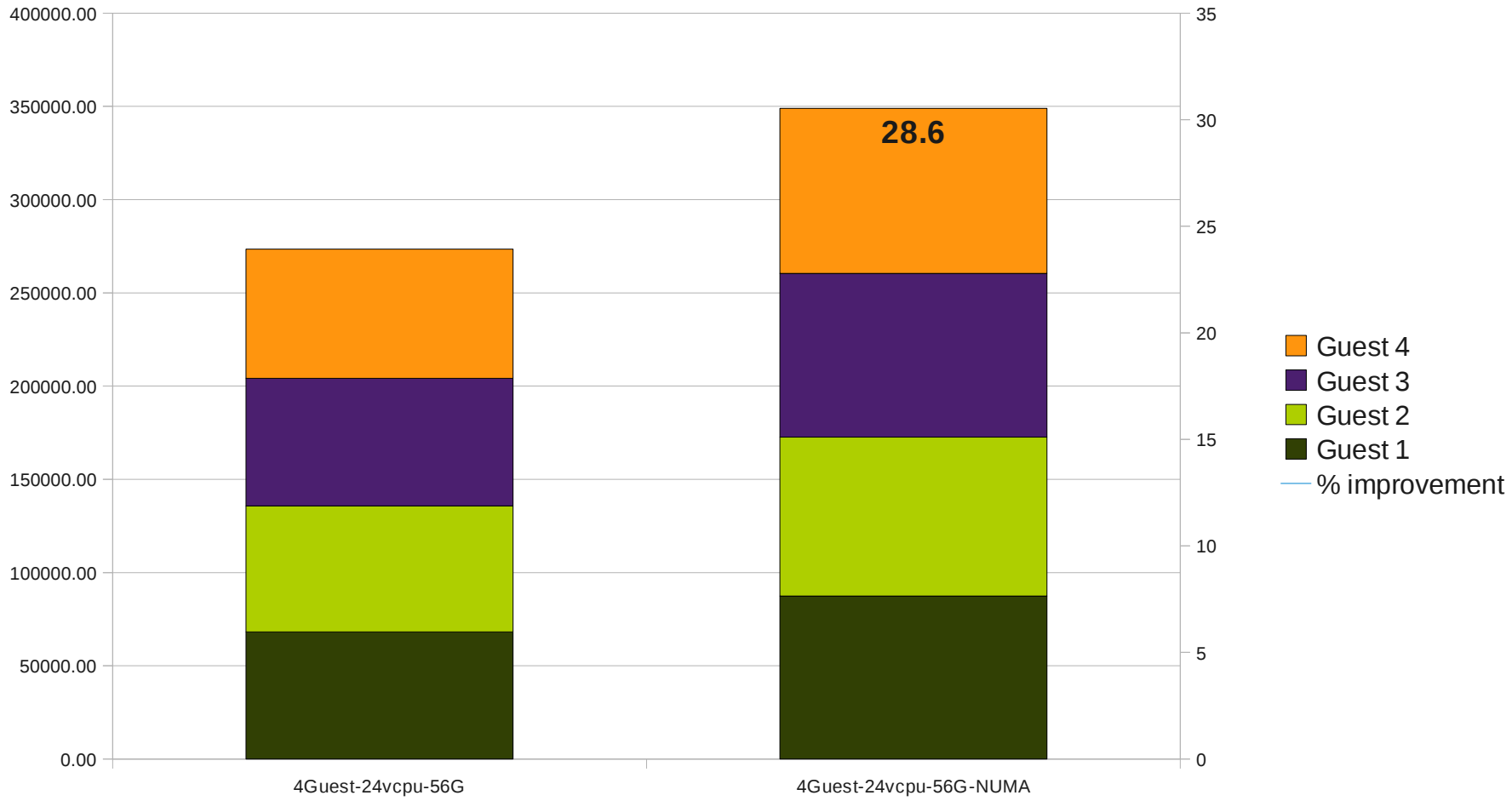
Using huge pages with libvirt, gives a significant performance boost

Using NUMA

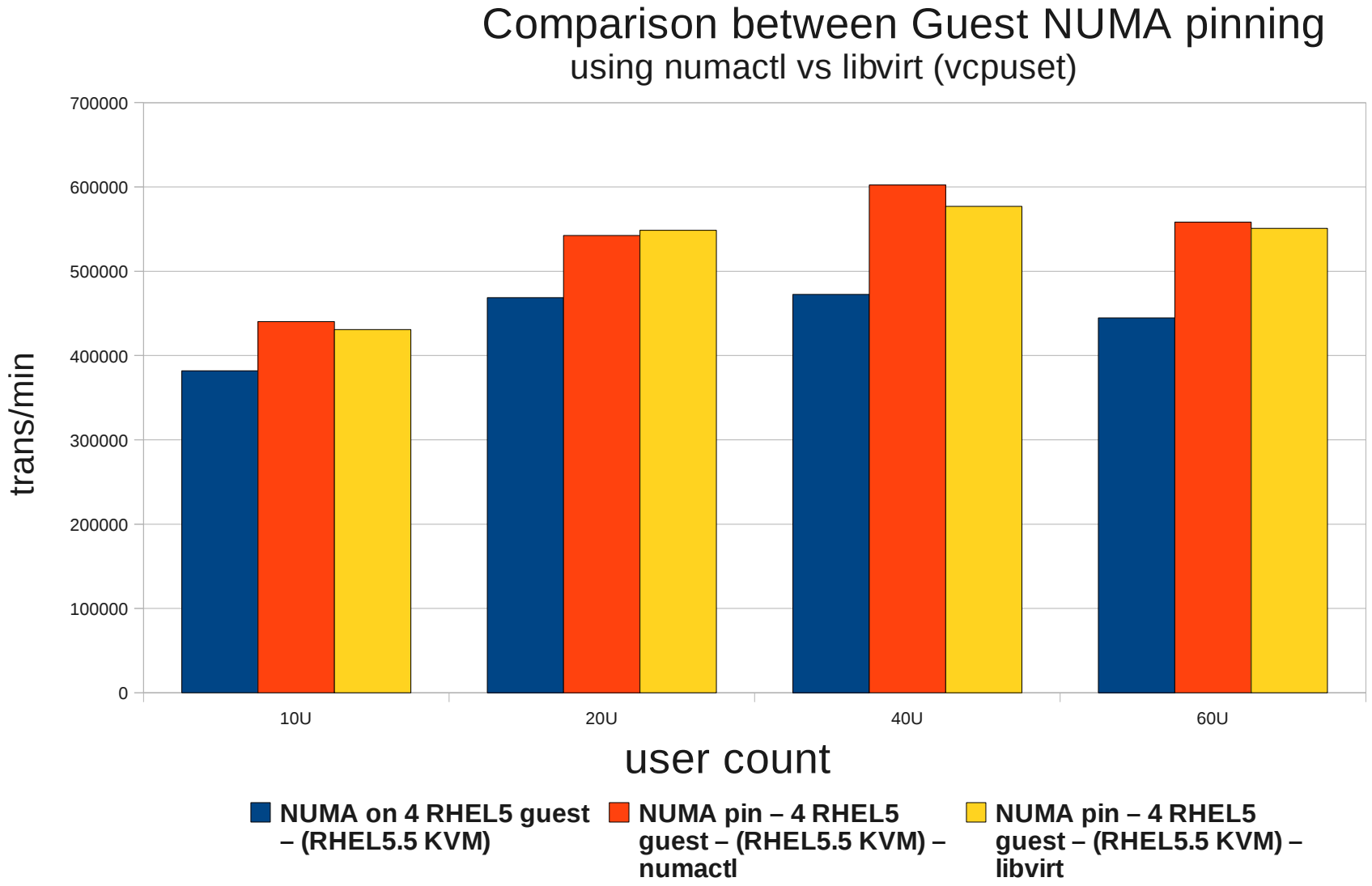
Understanding NonUniform Memory Access (NUMA)

- Multi core – Multi socket architectures
  - NUMA needed for scaling
  - RHEL 5 / 6 completely NUMA aware
  - KVM guests draw benefits of NUMA
    - Additional performance improvements to be gained by enforcing NUMA placement
- How to enforce NUMA placement
  - numactl – cpu and memory pinning
  - taskset – cpu pinning
  - libvirt – cpu pinning in libvirt - “<vcpus cpuset='0-3'>4</vcpus>”

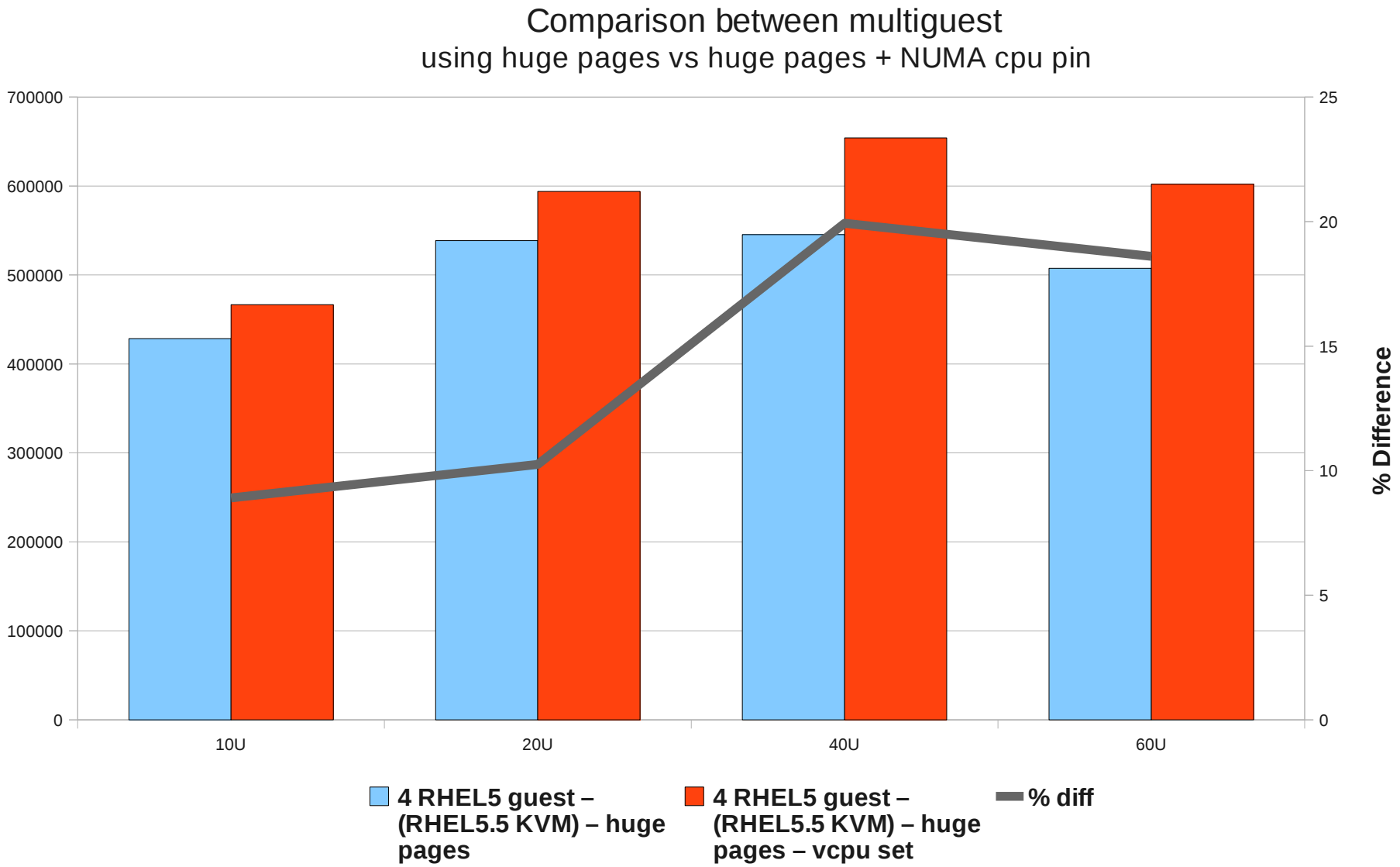
KVM Performance – AMD Istanbul - 24 cpu
Effect of NUMA on multiple guests running OLTP



AMD – Magny Cours – RHEL5.5 – KVM



AMD – Magny Cours – RHEL5.5 – KVM



Wrap it Up

Wrap up

- KVM is a loadable module
- KVM inherits all the kernel features
- KVM can be tuned effectively
  - Make sure you understand what is going on under the covers
  - Are you hitting page cache on the host ?
    - throughput vs latency numbers
- Look at using NUMA
- Huge Pages can help x86_64 hardware TLB
- Choose appropriate elevators (Deadline vs CFQ)

Wrap up (cont)

- Understand the network model
  - Pinning can help
  - Not always easy
- Device Assignment for high throughput / low latency
  - Need specific HW

FOLLOW US ON TWITTER

www.twitter.com/redhatsummit

TWEET ABOUT IT

#summitjbw

READ THE BLOG

http://summitblog.redhat.com/