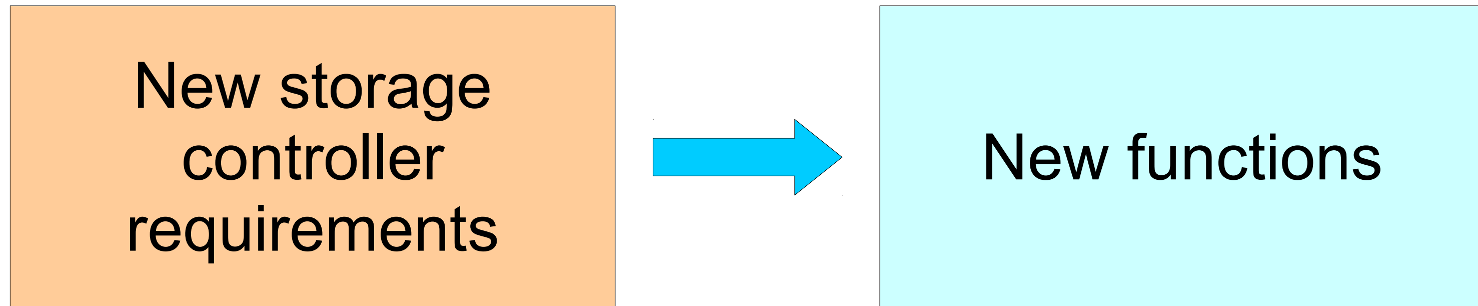
Adding Advanced Storage Controller Functionality via Low-Overhead Virtualization

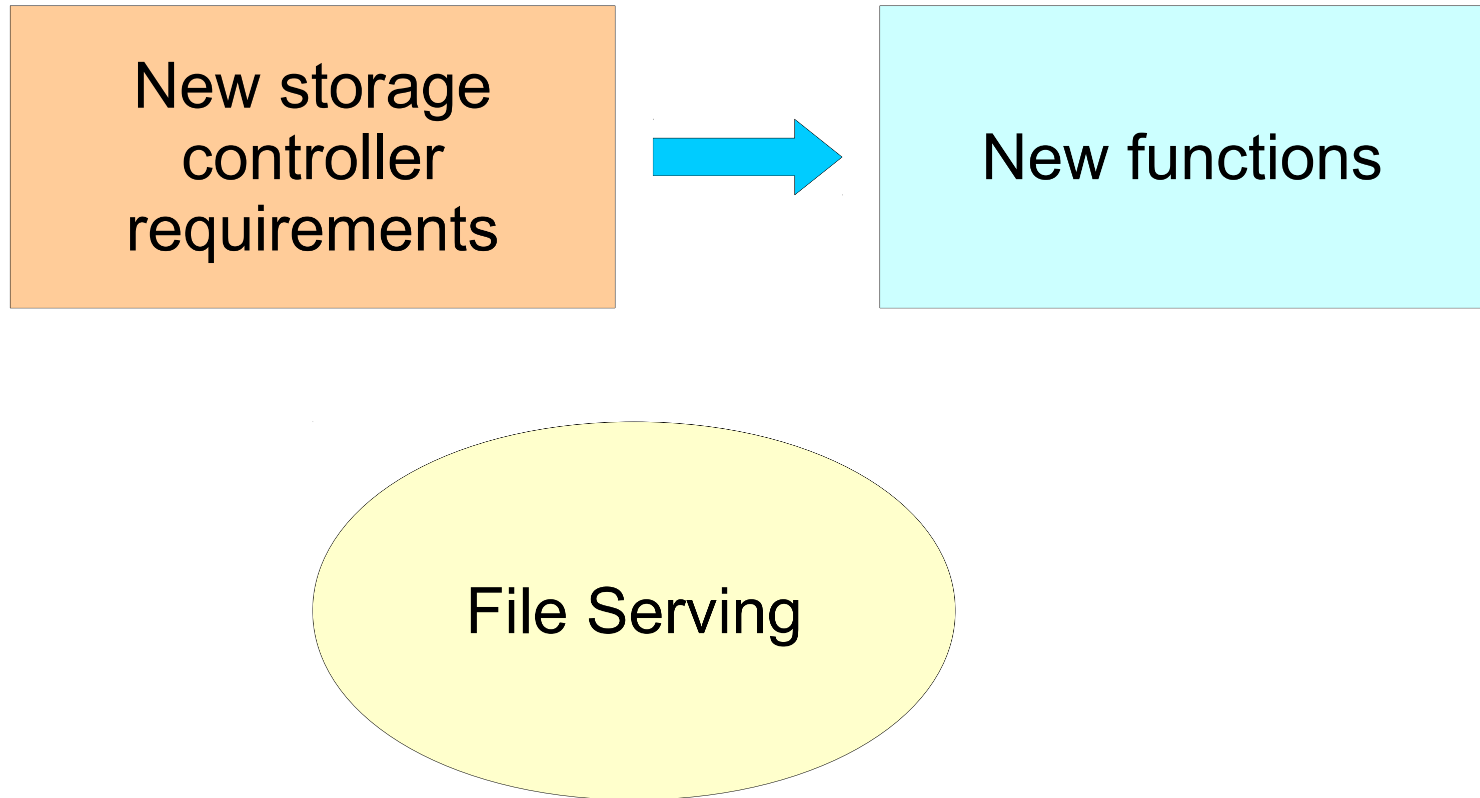
Muli Ben-Yehuda, Eran Borovik, Michael Factor, Eran Rom,
Avishay Traeger, Ben-Ami Yassour

Motivation



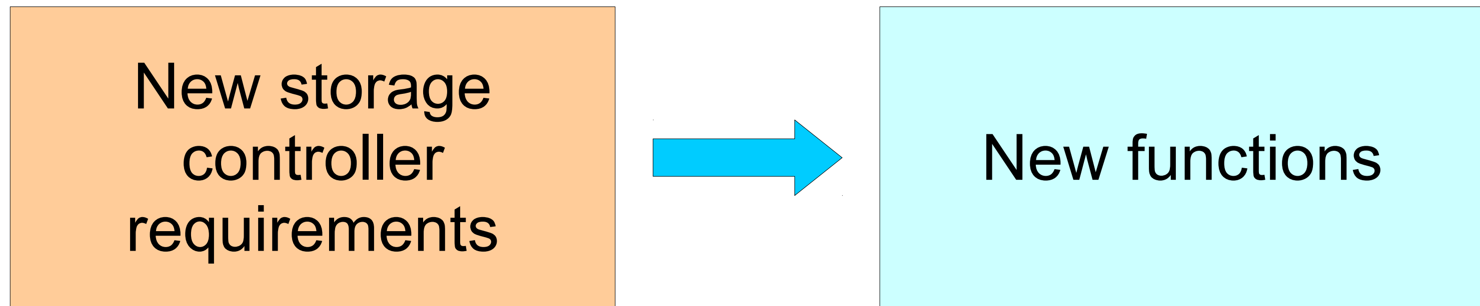
© 2012 IBM Corporation

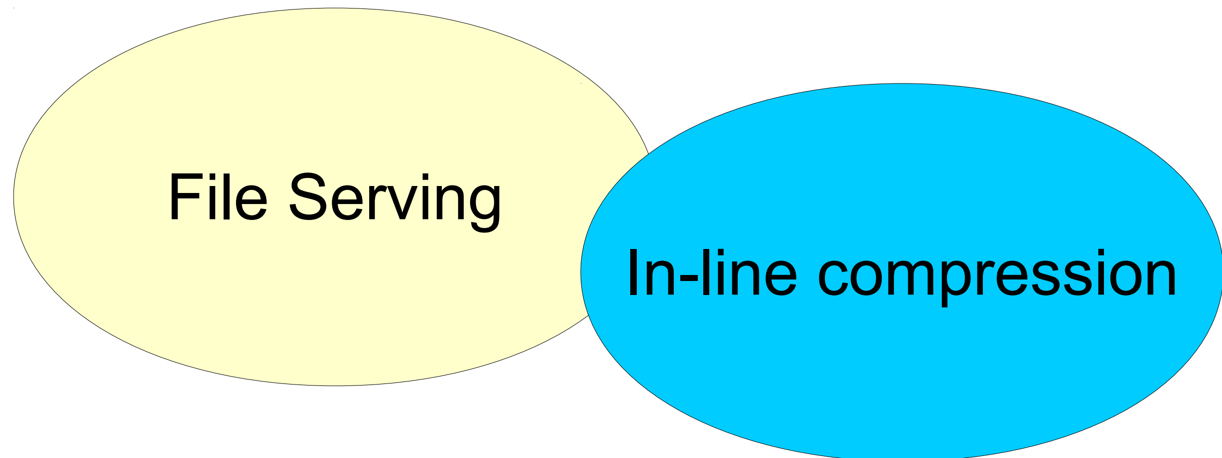
Motivation



© 2012 IBM Corporation

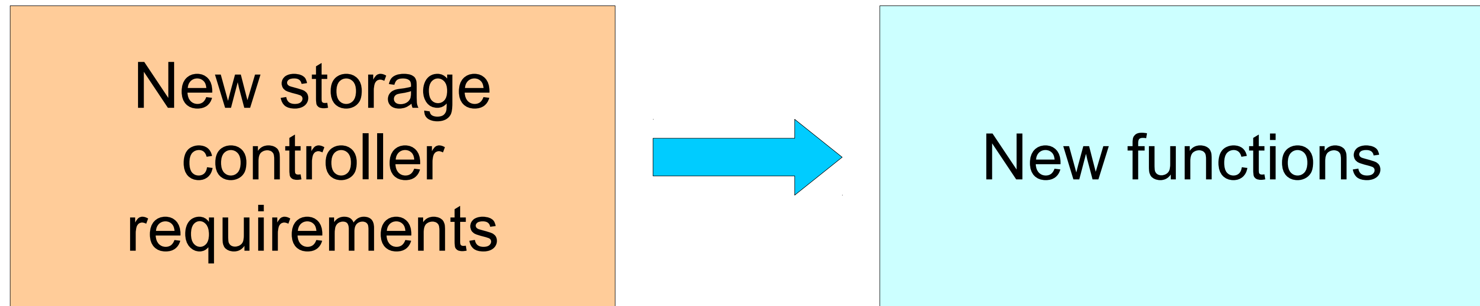
Motivation

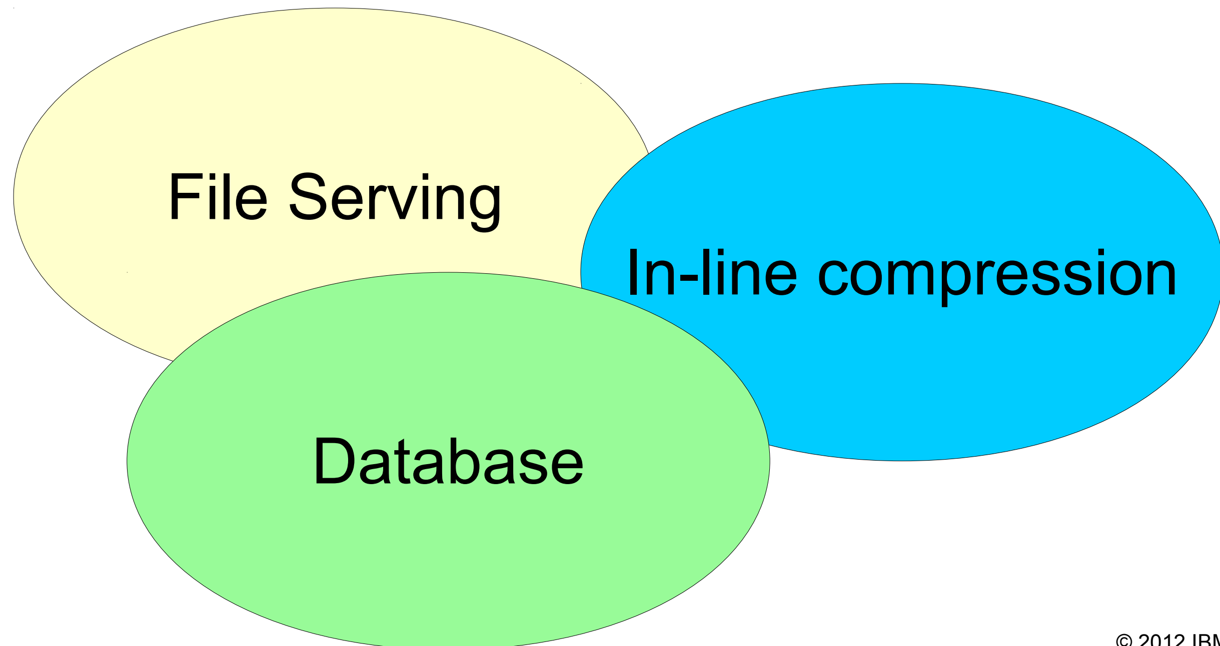




© 2012 IBM Corporation

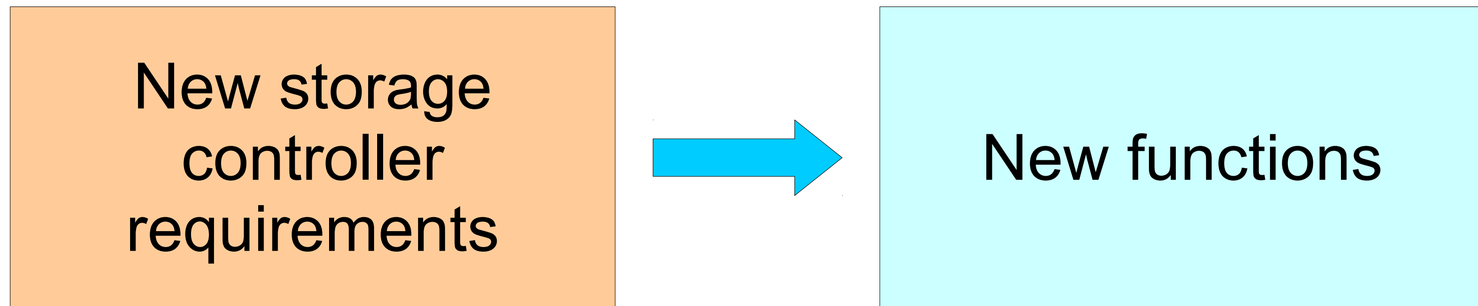
Motivation





© 2012 IBM Corporation

Motivation



File Serving

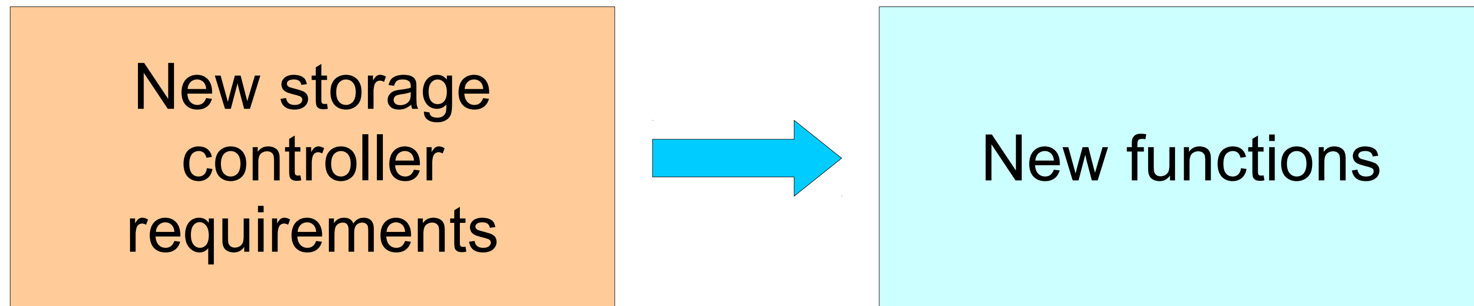
In-line compression

In-line dedup

Database

© 2012 IBM Corporation

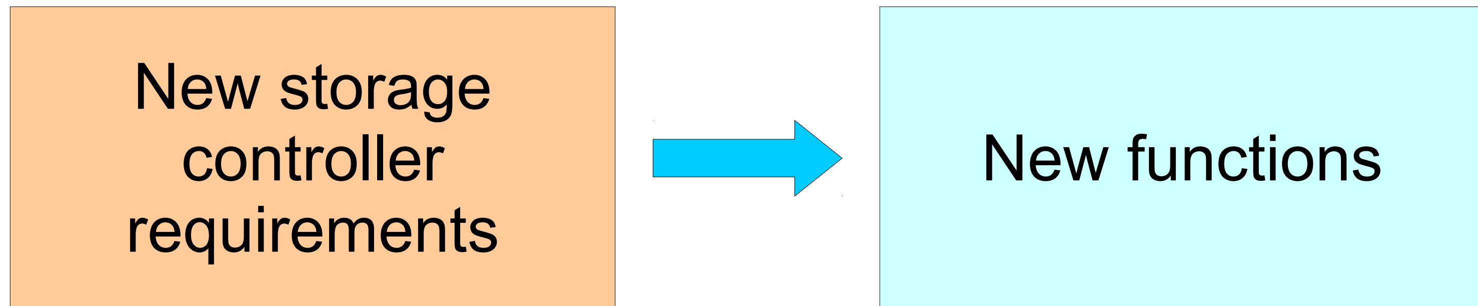
Motivation



Often, these functions already exist

© 2012 IBM Corporation

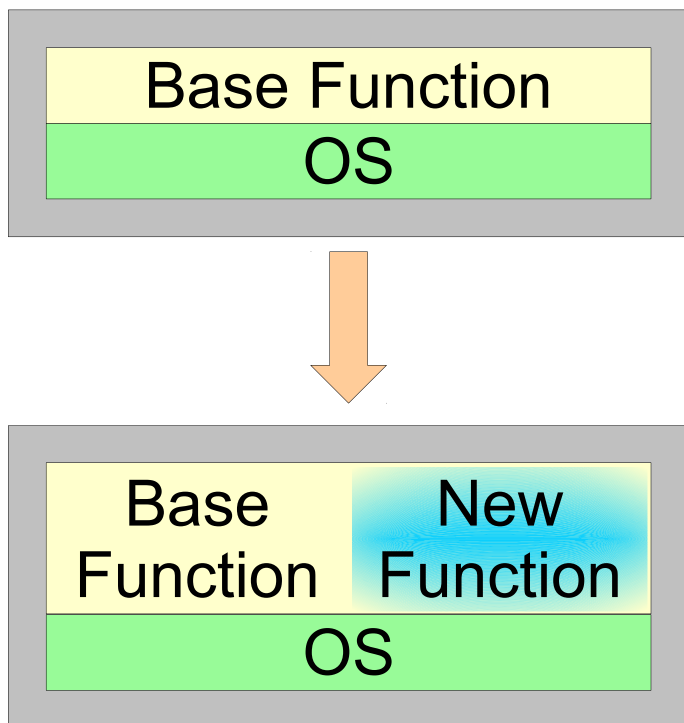
Motivation



What is the best method for adding the new function?

© 2012 IBM Corporation

First Method: Deep Integration



Pros

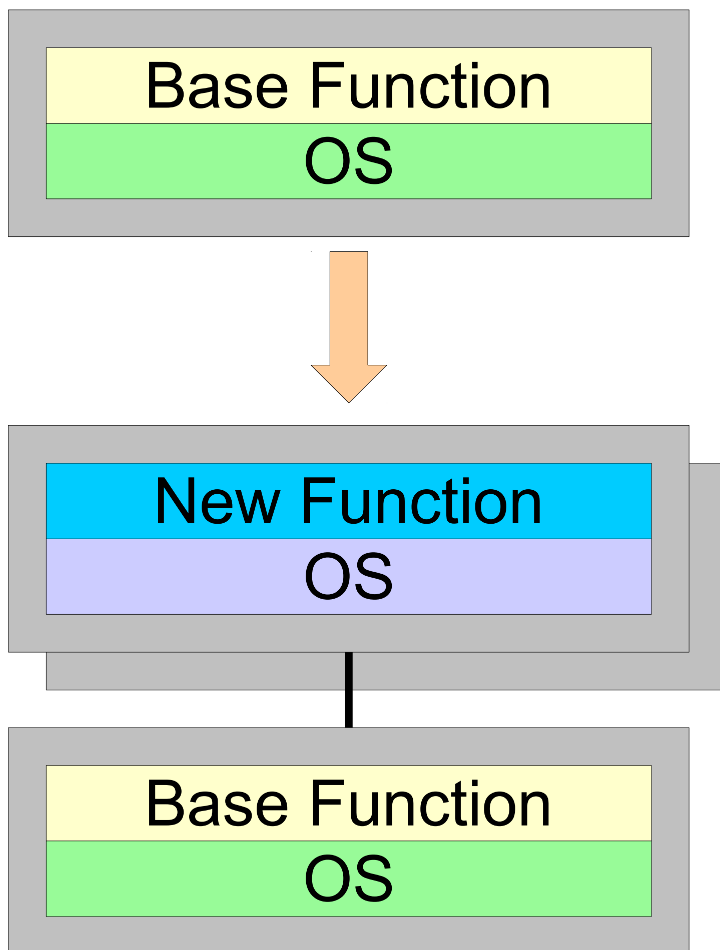
- Good performance
- Little hardware overhead

Cons

- Time-consuming integration
- Varying OS versions
- Difficult resource management – base function assumes a dedicated system
- Core function vulnerable to bugs
- Dual maintenance

© 2012 IBM Corporation

Second Method: External Gateway



Pros

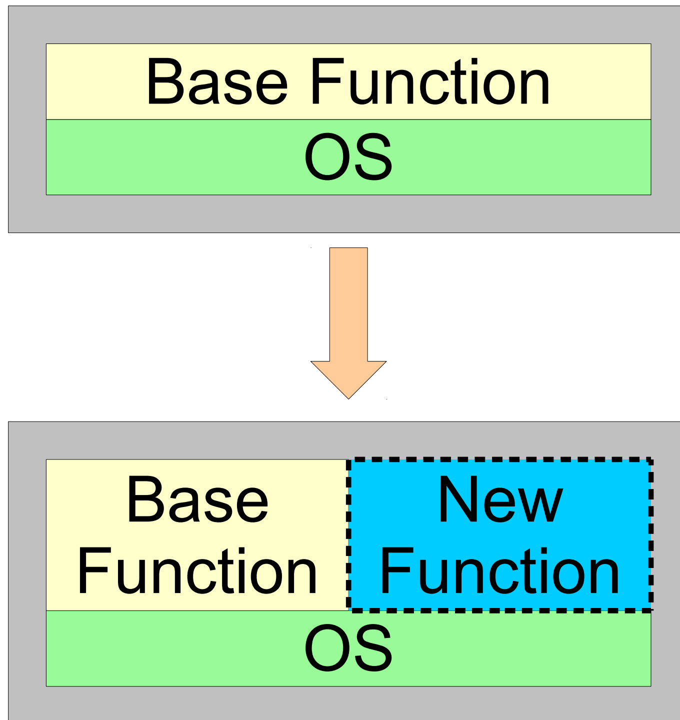
- Quick integration
- Protection of core function

Cons

- Higher communication overheads
- Expensive hardware costs (CAPEX+OPEX)

© 2012 IBM Corporation

Our Method: VM on the storage controller



Pros

- Quick integration
- Little hardware overhead
- Protection of core function

Cons

- “VMs have high overhead for I/O – I don't want that on my critical path!”

© 2012 IBM Corporation

Observation

A storage controller is a special-purpose machine with finely tuned resource control

VMs provide all the features we need and some that we don't
Need: fault isolation, resource isolation, dual environments
Don't need: resource sharing, over-commit, migration

Conclusion: customize the VM behavior for our needs

© 2012 IBM Corporation

IBM

External and Internal Communication

External

Clients ↔ New function (VM)

➢ Device assignment + SR-IOV

✔ Fast - bypass the host

✗ Exits for interrupts

✗ Exits for IOCs

Internal

New function (VM) ↔ Controller

➢ virtio block device

✔ Fast - shared memory

✗ Exits for submitting I/Os

✗ Exits for interrupts

✗ Exits for IOCs

© 2012 IBM Corporation

I/O Path

Guest
(KVM Process)

Core 1: File Server

3 File Server

4 Put block request in shared memory

Core 2: Guest Polling Thread

8 Poll block completions

2 Poll NIC driver

Host
(Controller Process)

Core 3: Controller

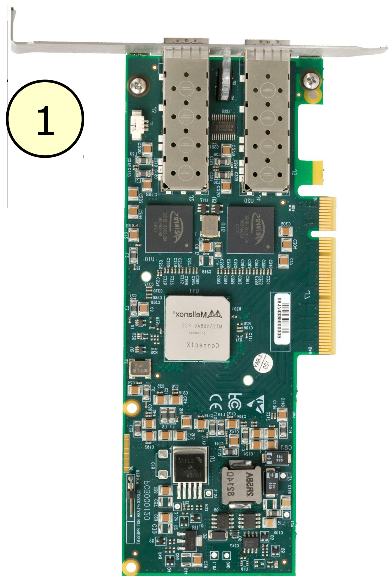
6 Storage Controller

7 Put completion in shared memory

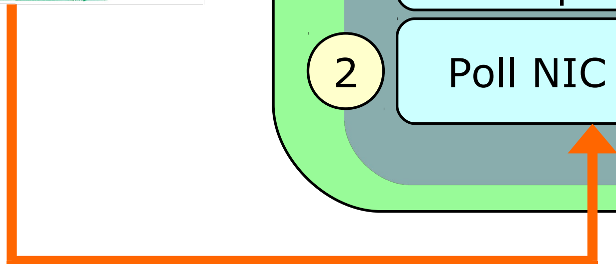
Core 4: Host Polling Thread

5 Poll file server block requests

(Poll other sources)



1



Device assignment

© 2012 IBM Corporation

CPU and Memory

Statically allocate CPU cores and memory

Boot guest kernel with `idle=poll`

Use *HugePages* for backing the guest's memory

Modify thread priorities and affinities

© 2012 IBM Corporation

IBM

Experimental Setup

- Used two servers, each with
  - CPU: 2 quad-core 2.93GHz Intel Xeon 5500 (w/ EPT)
  - RAM: 16GB
  - Ethernet adapter: Emulex OneConnect 10GbE
- Servers directly connected with 10GbE
- One server was the load generator, other was our (emulated) storage controller
- Controller server used 4 cores, unless otherwise specified
  - VM tests: guest got 2 cores and 2GB RAM
  - Bare metal tests: host got all cores and RAM
- Storage back-end: 8GB ramdisk via loopback
  - Physical disk doesn't become the bottleneck
  - Assignment of I/O to specific cores (similar to real controller)

© 2012 IBM Corporation

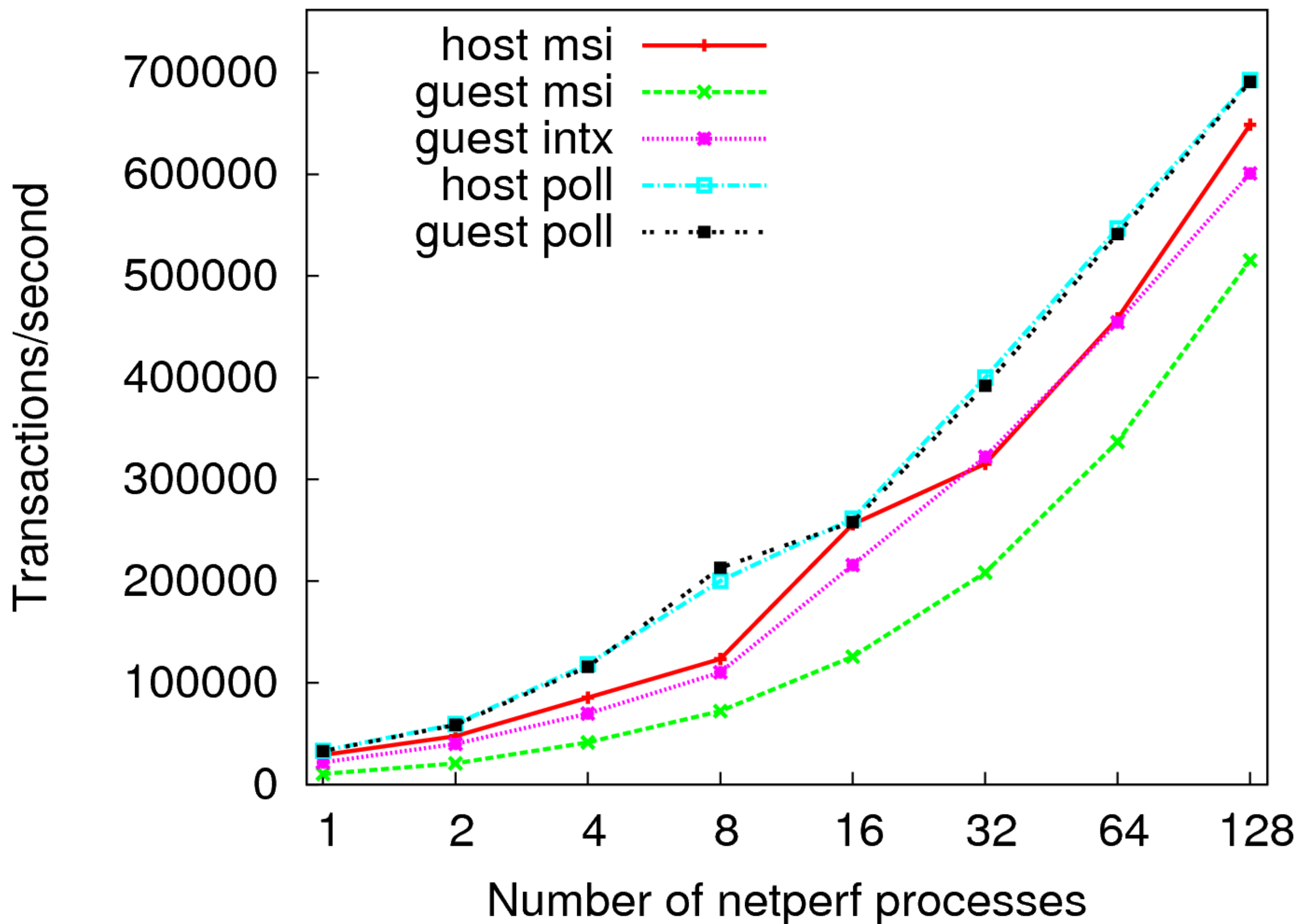
Network Latency: Ping Flood

	Bare metal	Guest (halt)
No polling	24 µs	89 µs
Polling	**21 µs**	**21 µs**

© 2012 IBM Corporation

IBM

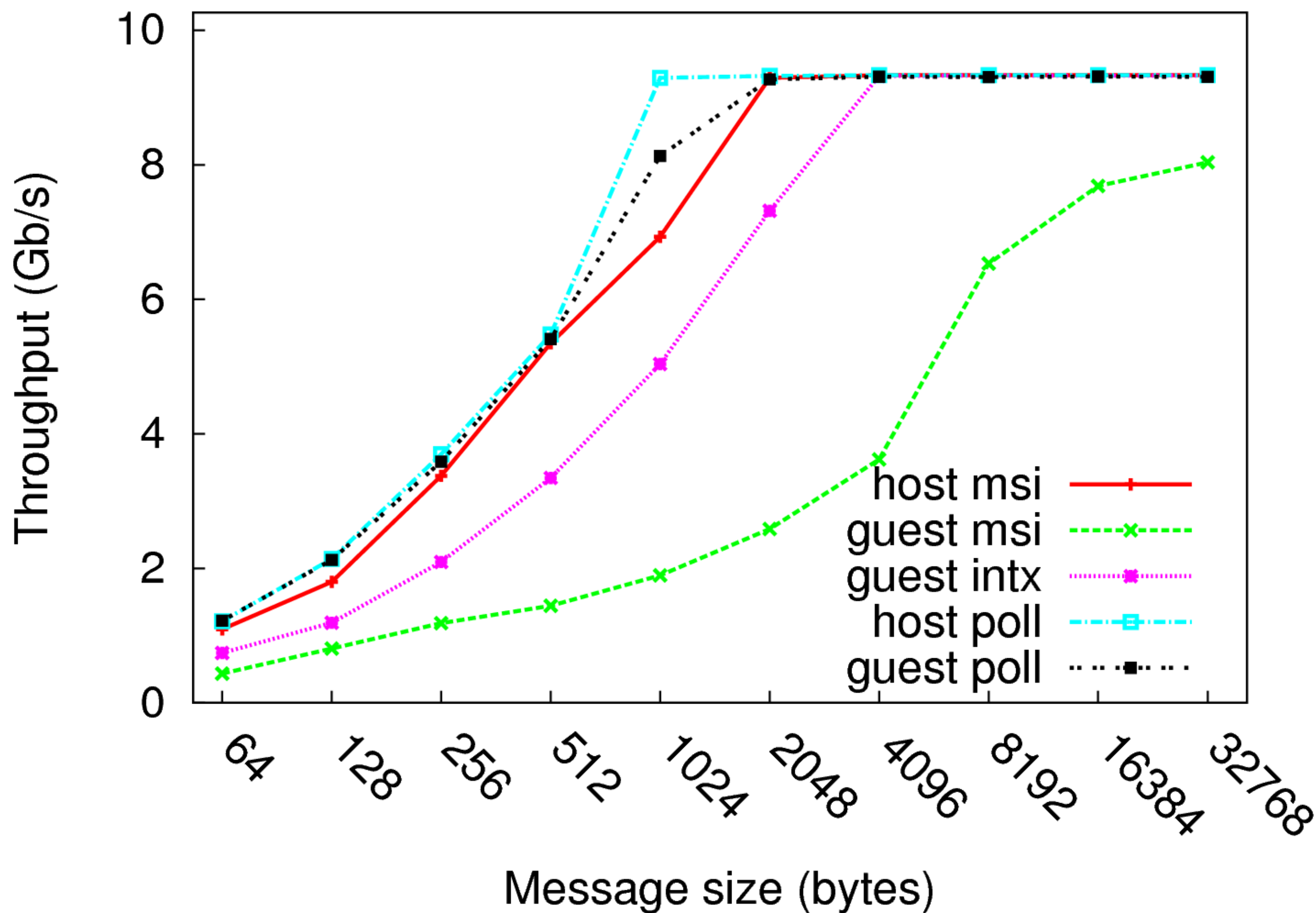
Netperf: Request-Response Throughput



© 2012 IBM Corporation

IBM

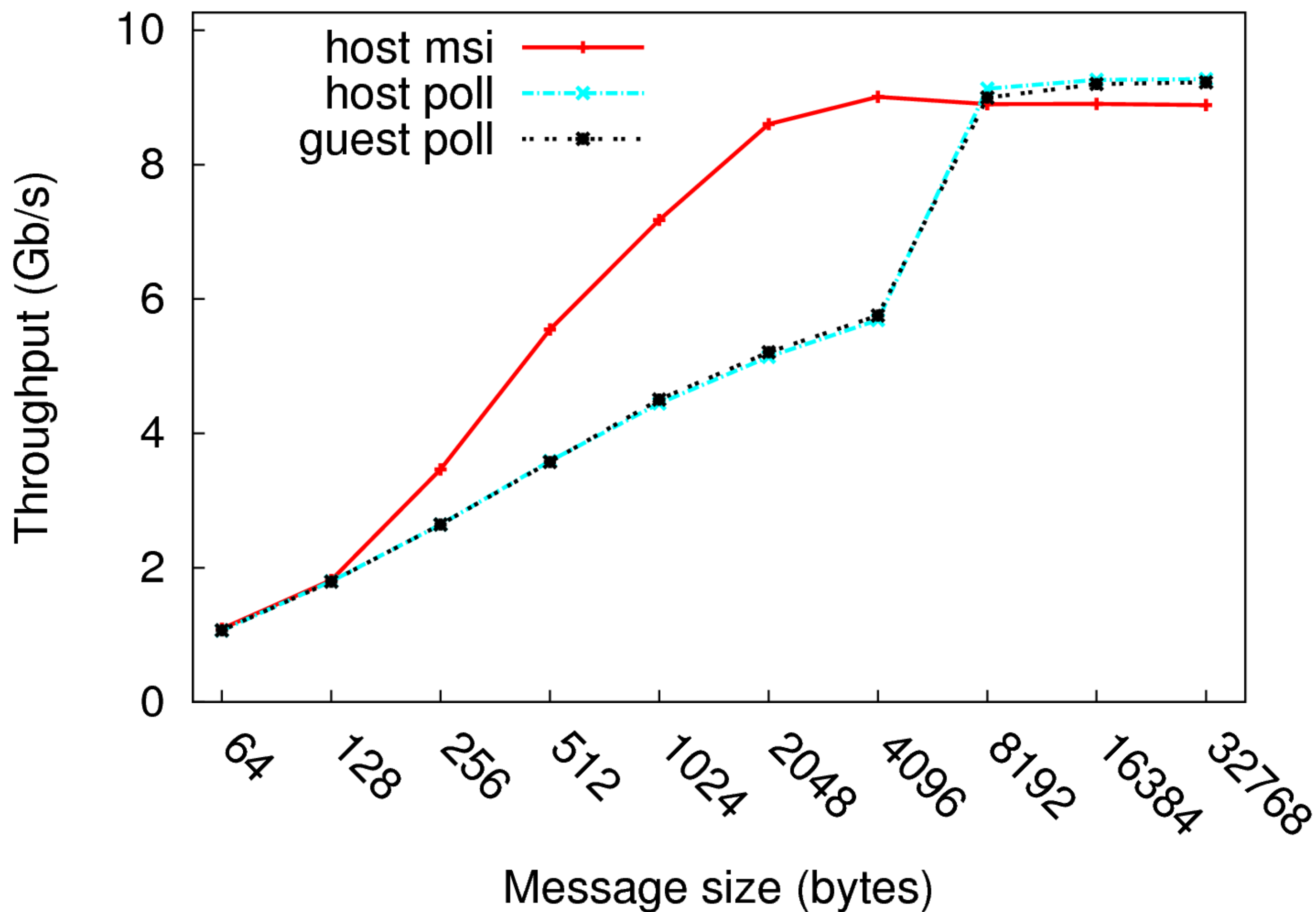
Netperf: TCP Send Throughput



© 2012 IBM Corporation

IBM

Netperf: TCP Receive Throughput



© 2012 IBM Corporation

IBM

Block Latency: 4KB Sync Writes

	Initial	Optimized
Total Latency	50 µs	15.9 µs
Added Latency	**49 µs**	**6.6 µs**

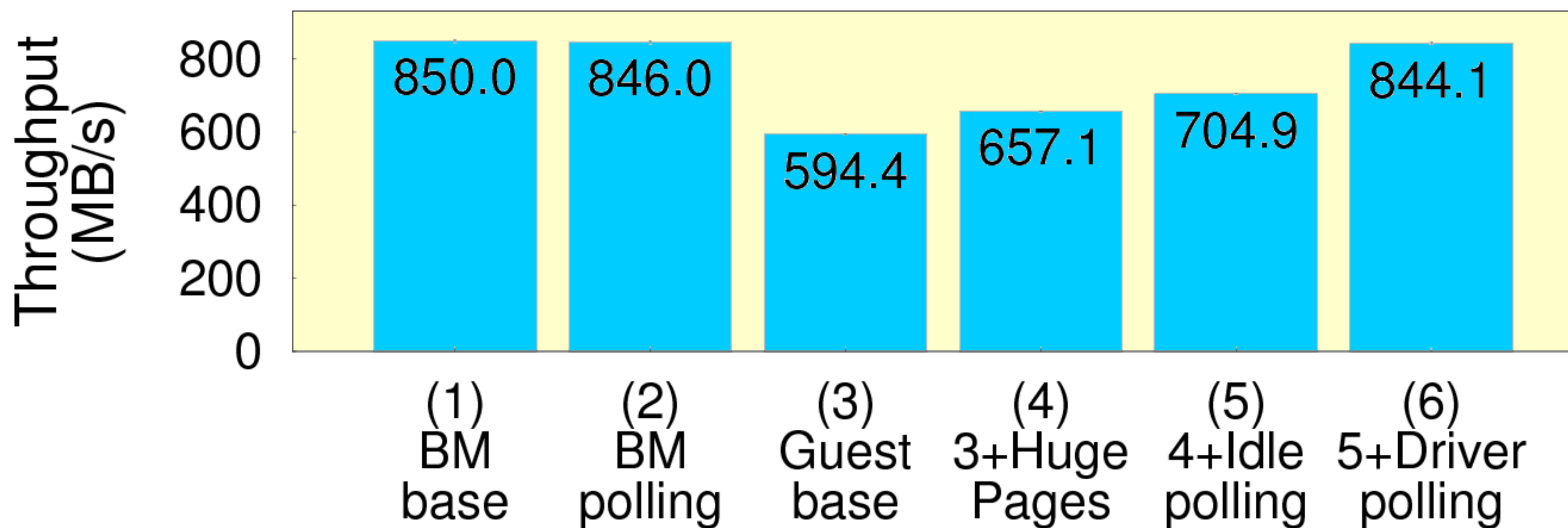
© 2012 IBM Corporation

Block Throughput: 4KB Sync I/Os

	Throughput	Improvement
Read	350K	**7.3x**
Write	284K	**6.5x**

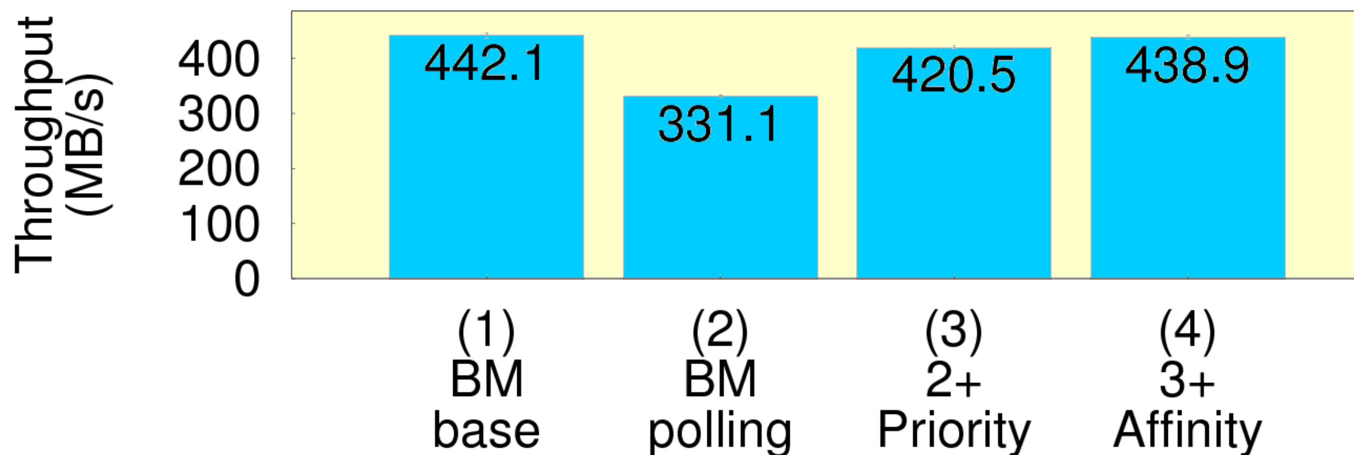
© 2012 IBM Corporation

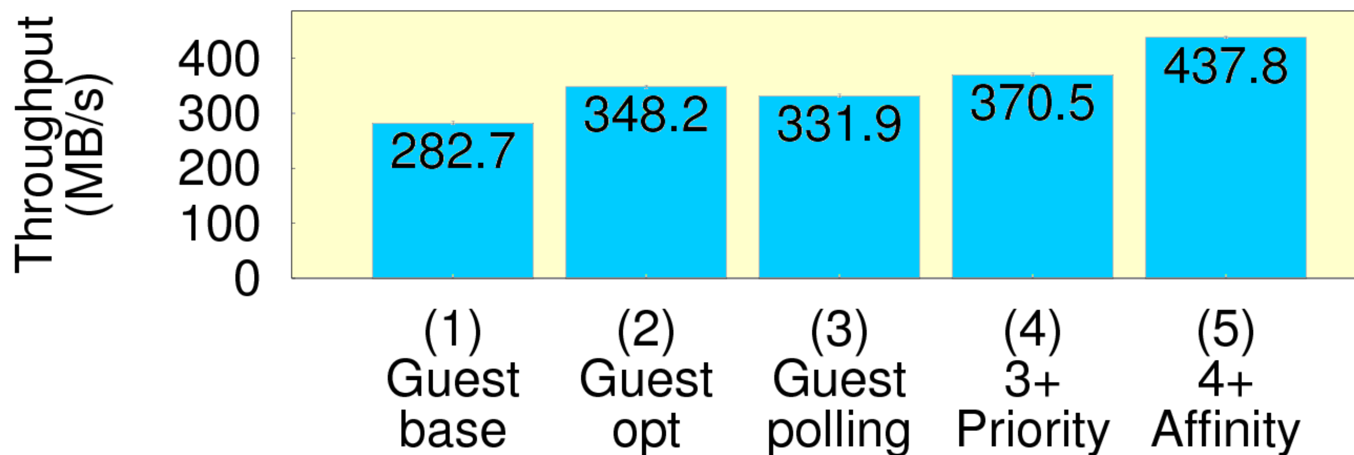
File Server: 4KB Read Cache Miss (6 cores)



© 2012 IBM Corporation

File Server: 4KB Read Cache Miss (4 cores)





© 2012 IBM Corporation

Conclusions

- It is feasible to use a virtual infrastructure to integrate new functions into a storage controller
- We demonstrated a set of mechanisms and techniques that achieve near zero performance overhead
- Benefit from performance and hardware cost of deep integration
- Benefit from shorter time to market, isolation, and simpler development model of the gateway approach

Future work:

1. Guest polling thread optional while keeping overheads near zero (ELI – ASPLOS 2012)
2. Benchmark multiple VMs on one host

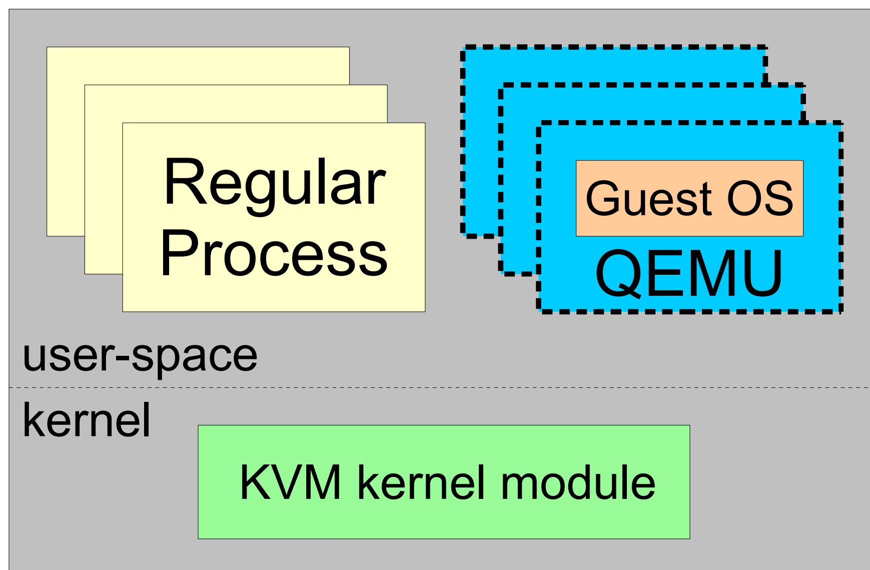
© 2012 IBM Corporation

Backup

© 2012 IBM Corporation

Background: KVM

- Open source, Linux-based hypervisor
- Leverages Intel VT-X or AMD-V features to virtualize the CPU
- Minimalistic hypervisor
  - VMs look like regular processes
  - Uses Linux's existing infrastructure (memory manager, scheduler, etc.)
- Asymmetric model



© 2012 IBM Corporation

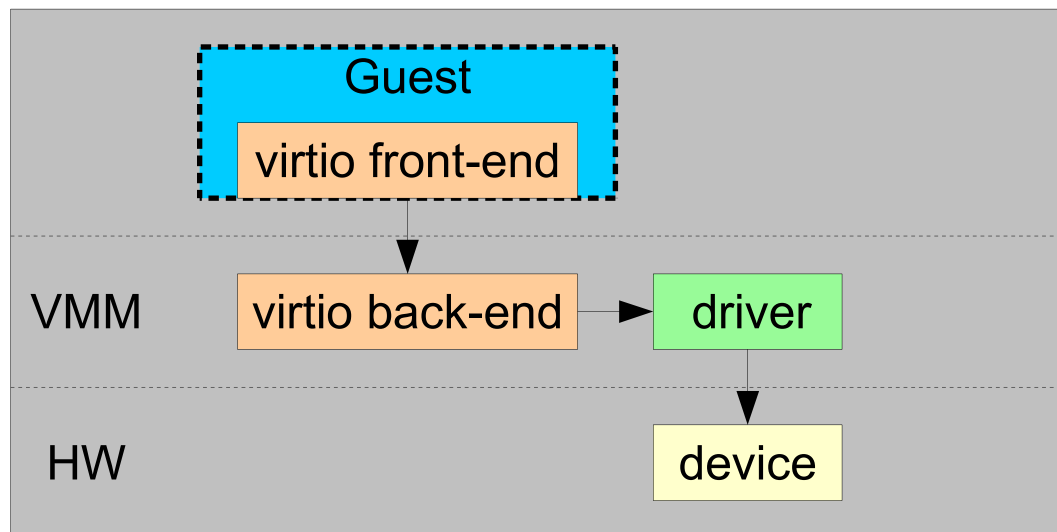
Background: Emulated I/O

- QEMU may emulate in software
  - BIOS
  - PCI bus
  - USB bus
  - Standard set of devices (IDE, SCSI, network)
- Guest OS uses its native drivers to access these devices
- Guest OS not aware of emulation – no guest changes required!
- Poor performance – each access to the device must be trapped and emulated → "world switches" AKA "exits"

© 2012 IBM Corporation

Background: virtio

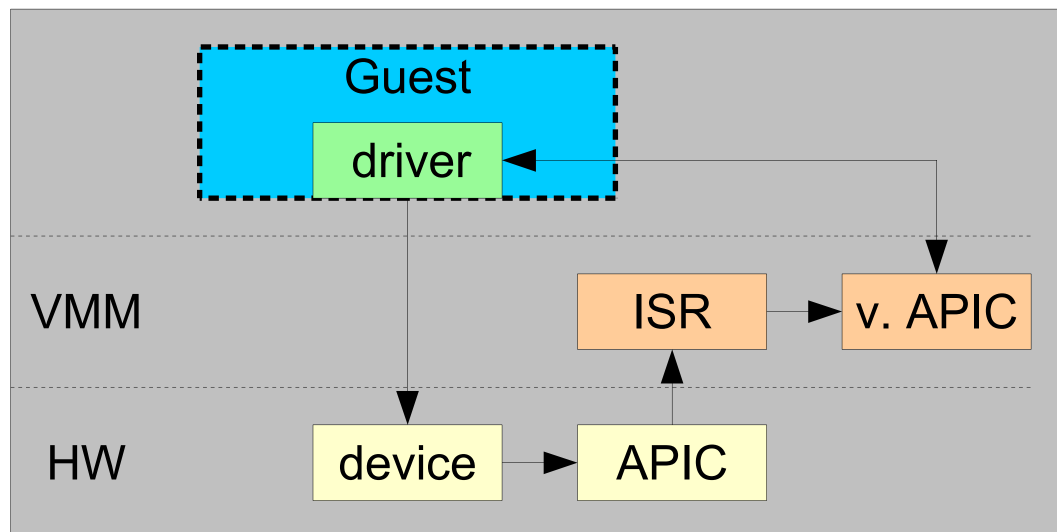
- Uses para-virtualized drivers
  - Guest OS uses drivers that are “aware” that the OS is virtualized and cooperate with the hypervisor to improve performance
- Up to 3 exits per I/O
  - PIO for I/O submission (may be batched)
  - Interrupts
  - IOCs



© 2012 IBM Corporation

Background: Device Assignment

- "Assigns" device to guest
  - Requires either dedicated device or SR-IOV
- Guest driver submits I/O directly to device
- Interrupts delivered by hypervisor – two exits

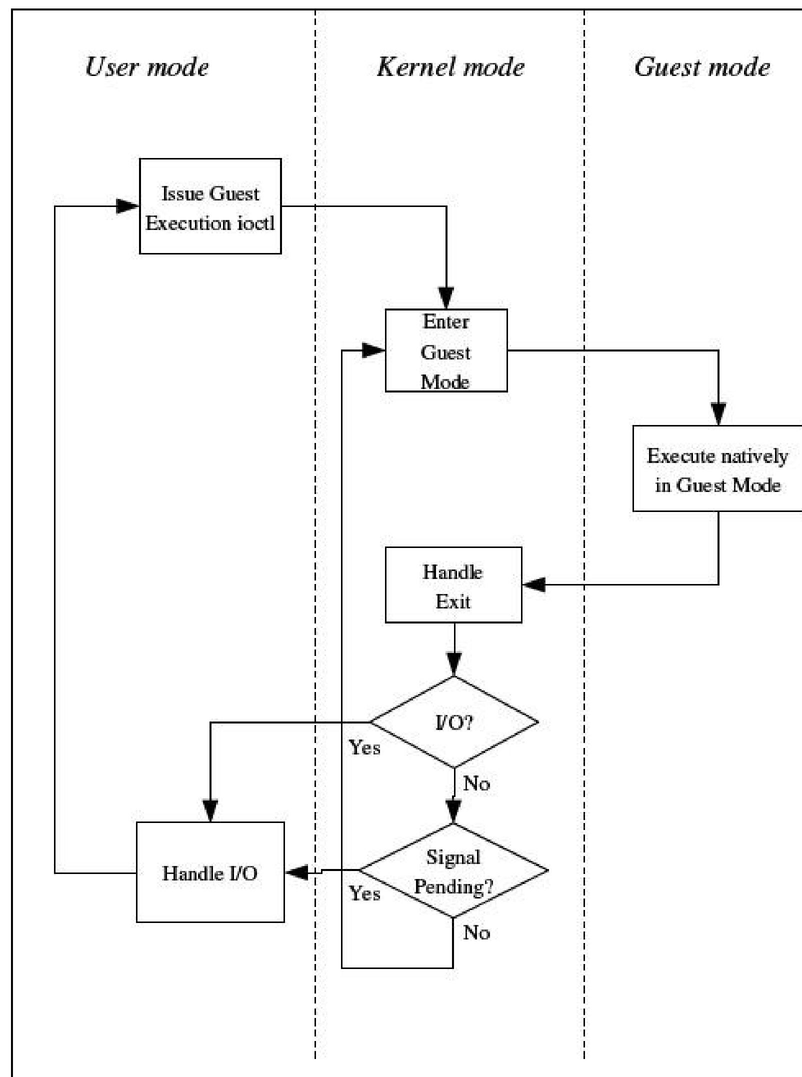


© 2012 IBM Corporation

IBM

Background: Guest Execution

- As long as there are no exits, stay in guest mode and run with minimal overhead
- Exit handling can be relatively fast if not I/O
  - e.g., interrupts delivery in device assignment
- Large overhead if VMM must handle I/O during exit
  - virtio
  - Emulation



© 2012 IBM Corporation