Efficient and sca**L**able para**V**irtual **I**/o **S**ystem (**ELVIS**)

Nadav Har'El[×] Abel Gordon[×] Alex Landau[×]

Muli Ben-Yehuda[×,¤] Avishay Traeger[×] Razya Ladelsky[×]

[×] IBM Research – Haifa
[¤] Technion and Hypervisor Consulting

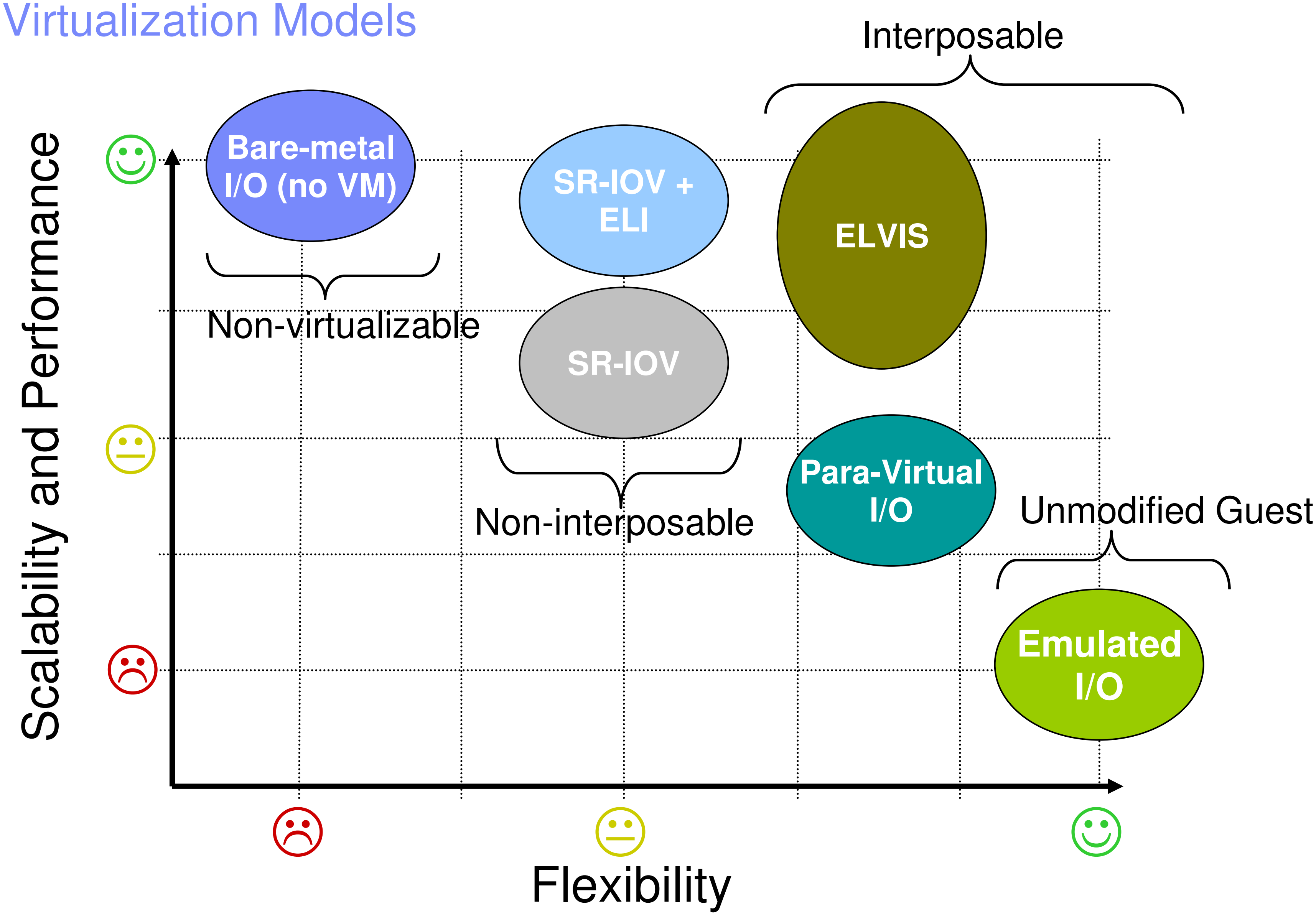
Why (not) software-based I/O interposition in virtual environments?

- Pros
  - Software Defined Networking
  - File based images
  - Live Migration
  - Fault Tolerance
  - Security
  - ….

- Cons
  - Scalability Limitations
  - Performance Degradation

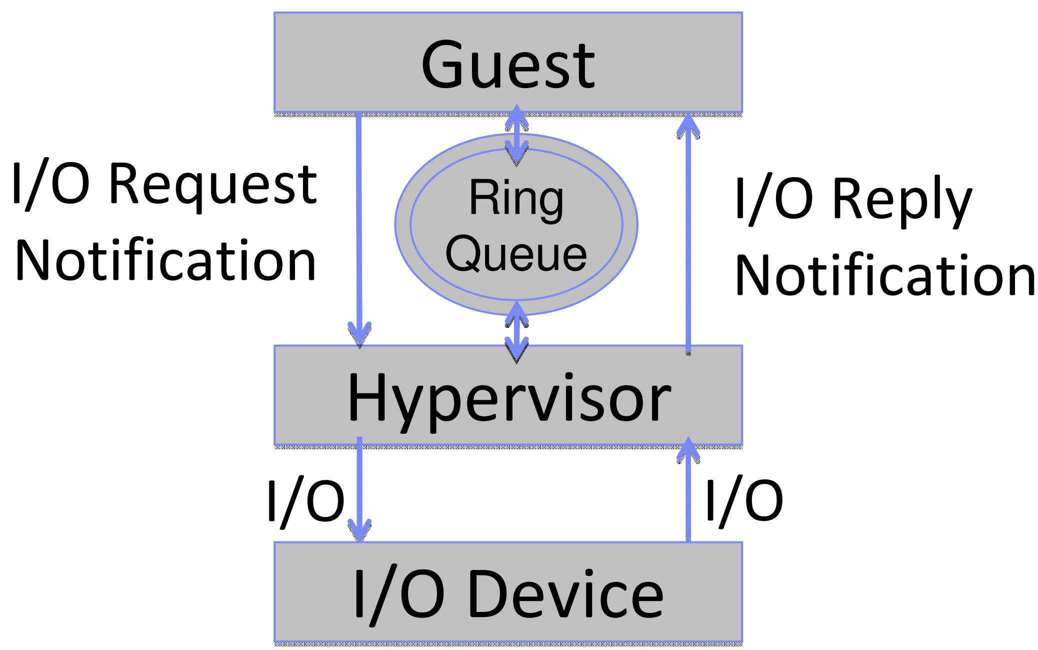
  - Scalability Limitations
  - Performance Degradation

I/O Virtualization Models



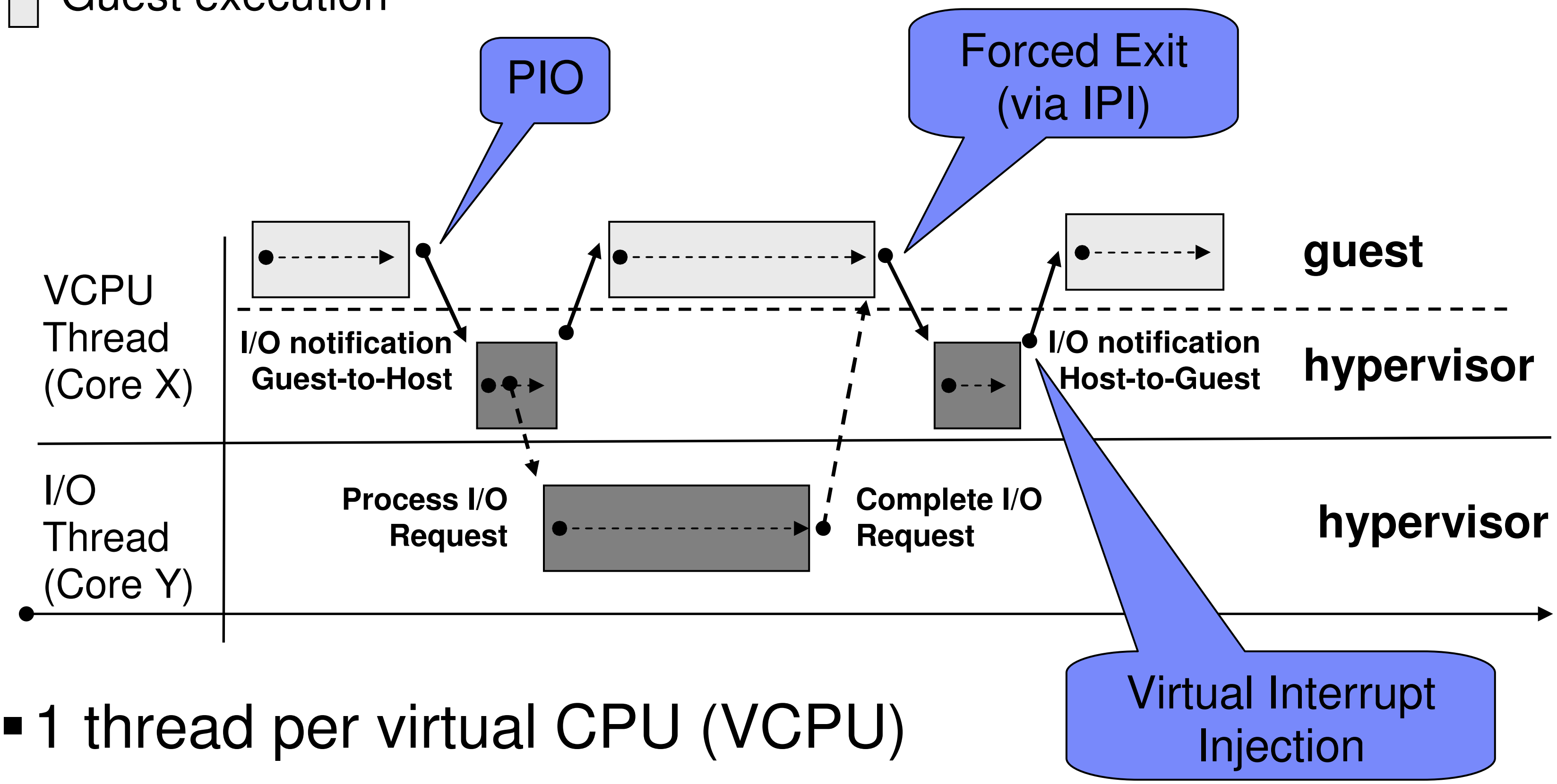
How Paravirtual I/O works today

- The guest posts I/O requests in ring-queue (shared with the hypervisor) and sends a request notification
- The hypervisor processes the requests and sends a reply notification



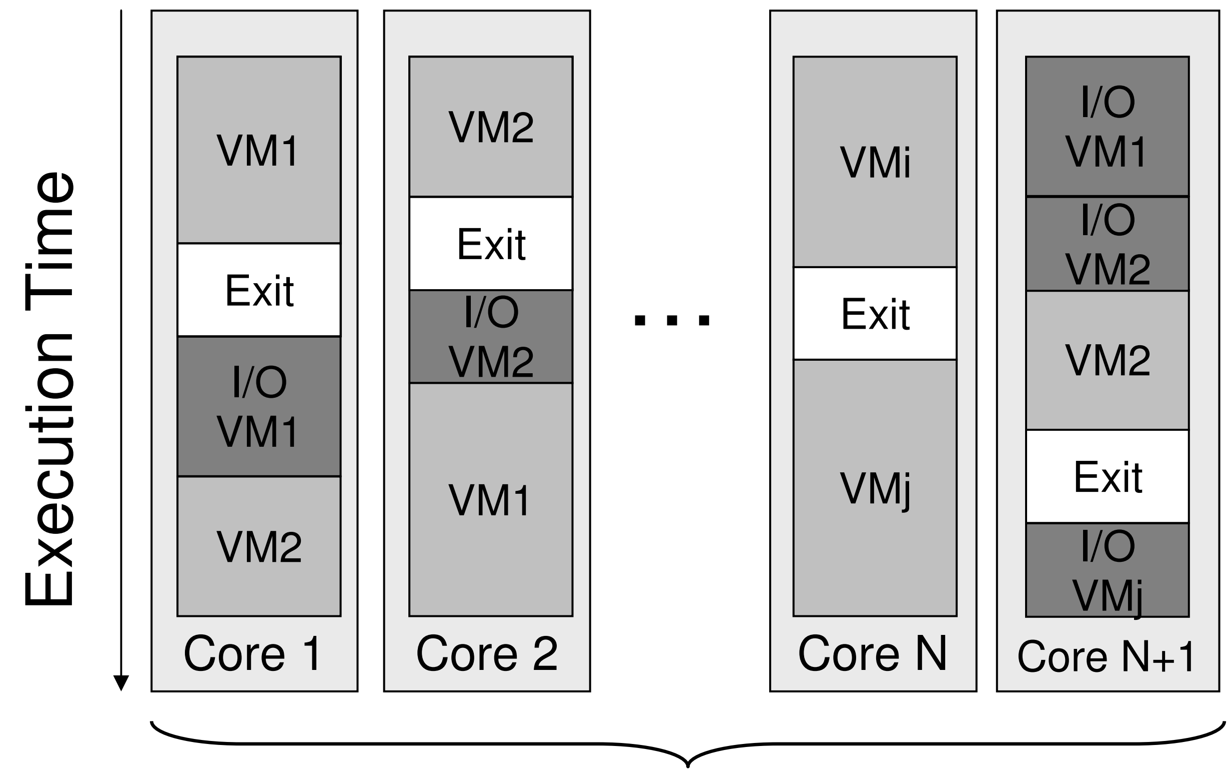
How I/O notifications are sent/received

- CPU context switch (exits and entries)
- I/O processing
- Guest execution



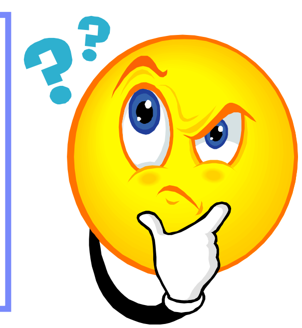
- 1 thread per virtual CPU (VCPU)
- 1 thread per virtual I/O device

Is this model scalable with the number of guests and I/O bandwidth ?



VCPU and I/O thread-based scheduling for all cores

Depends on the host thread scheduler but the scheduler has no information about the I/O activity of the virtual devices….

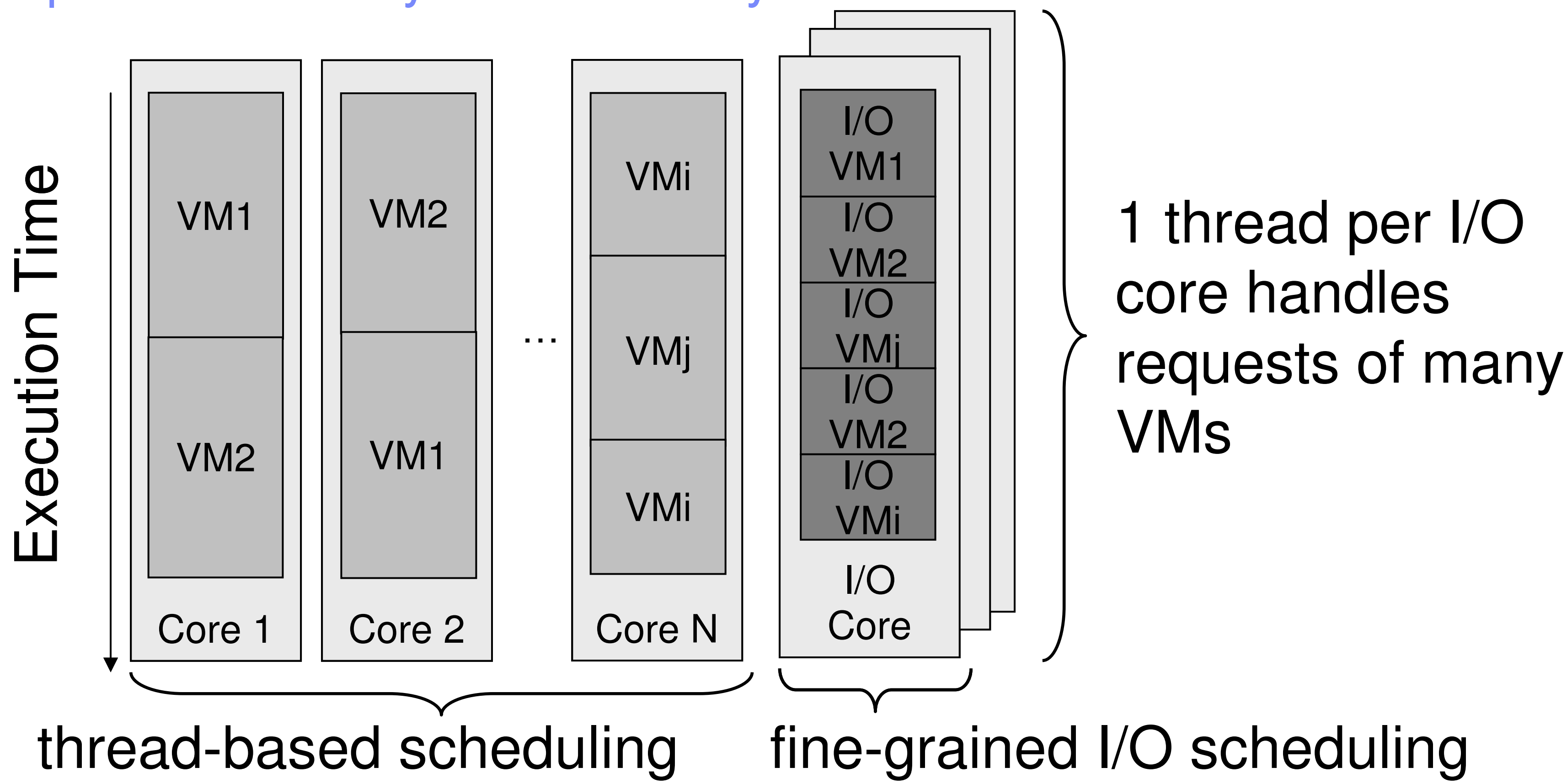


Facts and Trends

- Notifications cause exits (context switches) == overhead!
- Current trend is:
  – Towards multi-core systems with an increasing numbers of cores per socket (4->6->**8**->16**->**32) and guests per host
  – Faster networks with expectation of lower latency and higher bandwidth (1GbE->10GbE->**40GbE**->100GbE)
- I/O virtualization is a CPU intensive task, and may require more cycles than the available in a single core

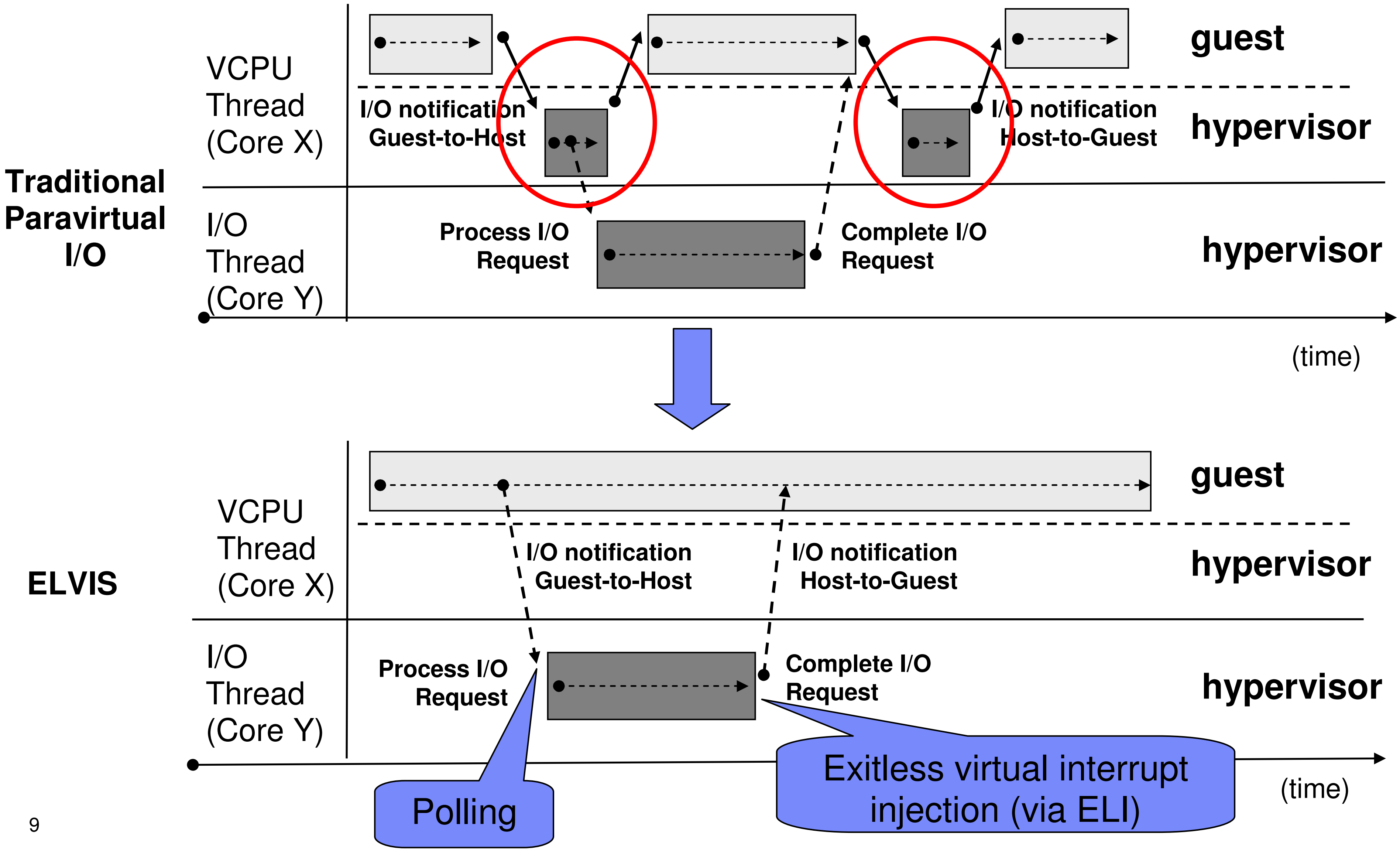
We need a new "efficient" and "scalable" Paravirtual I/O model!

ELVIS: use fine-grained I/O scheduling and dedicate cores to improve scalability and efficiency



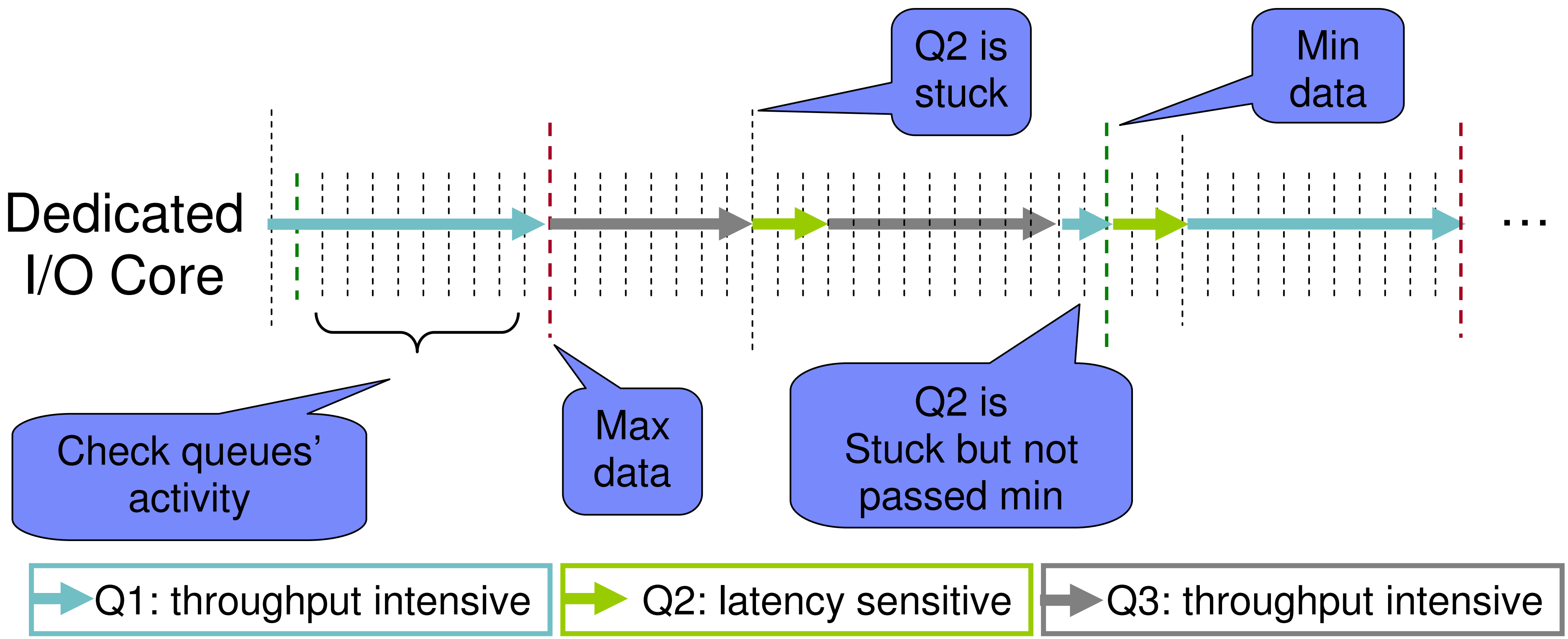
- Process queues based on the I/O activity
- Balance between throughput and latency
- No process/thread context switches for I/O
- Exitless communication (next slide)

ELVIS: remove notifications overhead to further improve efficiency



ELVIS: Fine-grained I/O scheduling in a nutshell

- Single thread in a dedicated core monitors the activity of each queue (VMs I/O)
- Decide which queue should be processed and for how long



ELVIS: Placement of threads, memory and interrupts

- Dedicate 1 I/O core per CPU socket
  - Cores per socket continue to increase year by year
  - More cores are required to virtualize more bandwidth at lower latencies (network links continue to be improved)
  - NUMA awareness: shared LLC cache and memory controller, DDIO technology
- Deliver interrupts to the “corresponding” I/O core
  - Interrupts are processed by I/O cores and do not disturb the running the guests
  - Improve locality
  - Multi-port and SR-IOV adapters can dedicate interrupts per port or virtual function

Implementation and Experimental Setup

- Implementation
  - Based on KVM Hypervisor (Linux Kernel 3.1 / QEMU 0.14)
  - With VHOST, in-kernel paravirtual I/O framework
  - Use ELI patches to enable exitless replies and to improve hardware-assisted non-interposable I/O (SR-IOV)
- Experimental Setup
  - IBM System x3550 M4, dual socket 8 cores per socket Intel Xeon E2660 2.2GHz (SandyBridge)
  - Dual port 10GbE Intel x520 SRIOV NIC
  - 2 identical servers: one used to host the VMs and the other used to generate load on bare-metal

Methodology

- Repeated experiments using 1 to 14 UP VMs
  - 1x10GbE when running up-to 7 VMs
  - 2x10GbE when running more than 7 VMs

- Compared ELVIS against 3 other configurations

- **No interposition**
  - Each VM runs on a dedicated core and has a SR-IOV VF assigned using ELI
  - The closer ELVIS is to this configuration, the smaller the overhead is (used to evaluate ELVIS efficiency)

Methodology (cont.)

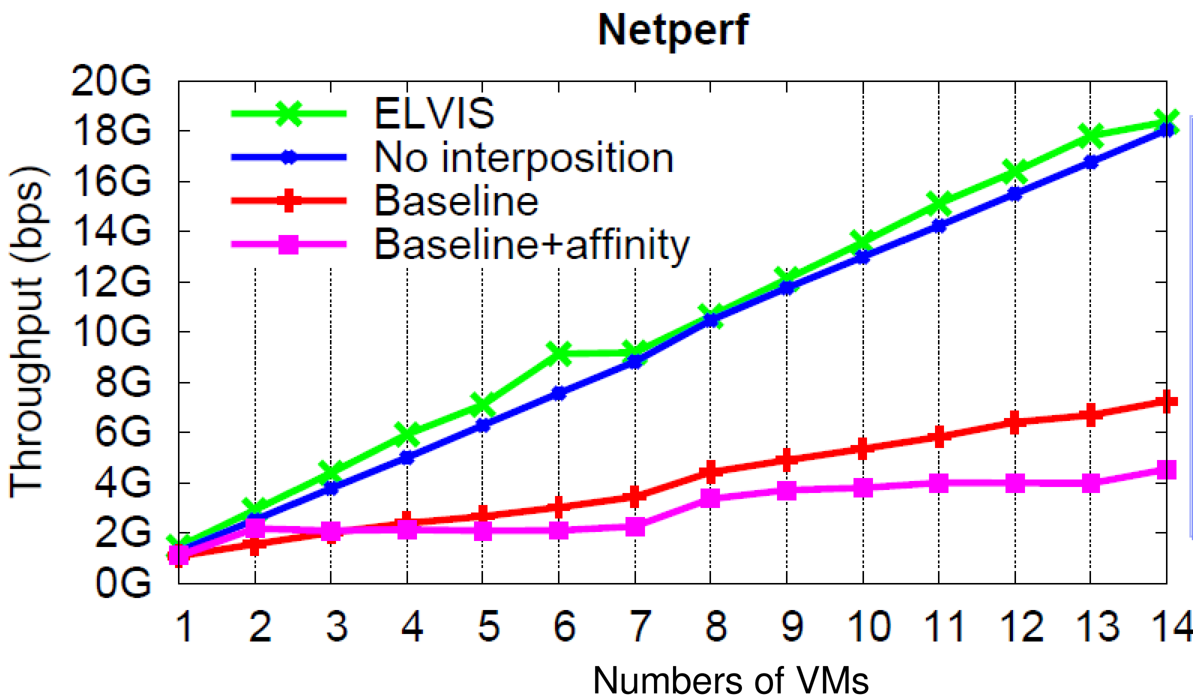
- N=number of VMs (1 to 14)
- Used N+1 cores (N≤ 7) or N+2 cores (N>7)
  - This is the resource overhead for I/O interposition

- **ELVIS**
  - 1 dedicated core per VCPU (VM)
  - 1 core (N<=7) or (N>7) 2 cores dedicated for I/O

- **Baseline**
  - N+1 cores (N ≤ 7) or N+2 cores (N>7) to run VCPU and I/O threads (no thread affinity)

- **Baseline+Affinity**
  - Baseline but dedicate 1 core per VCPU and pin I/O threads to dedicated I/O cores

Netperf – TCP Stream 64Bytes (throughput intensive)

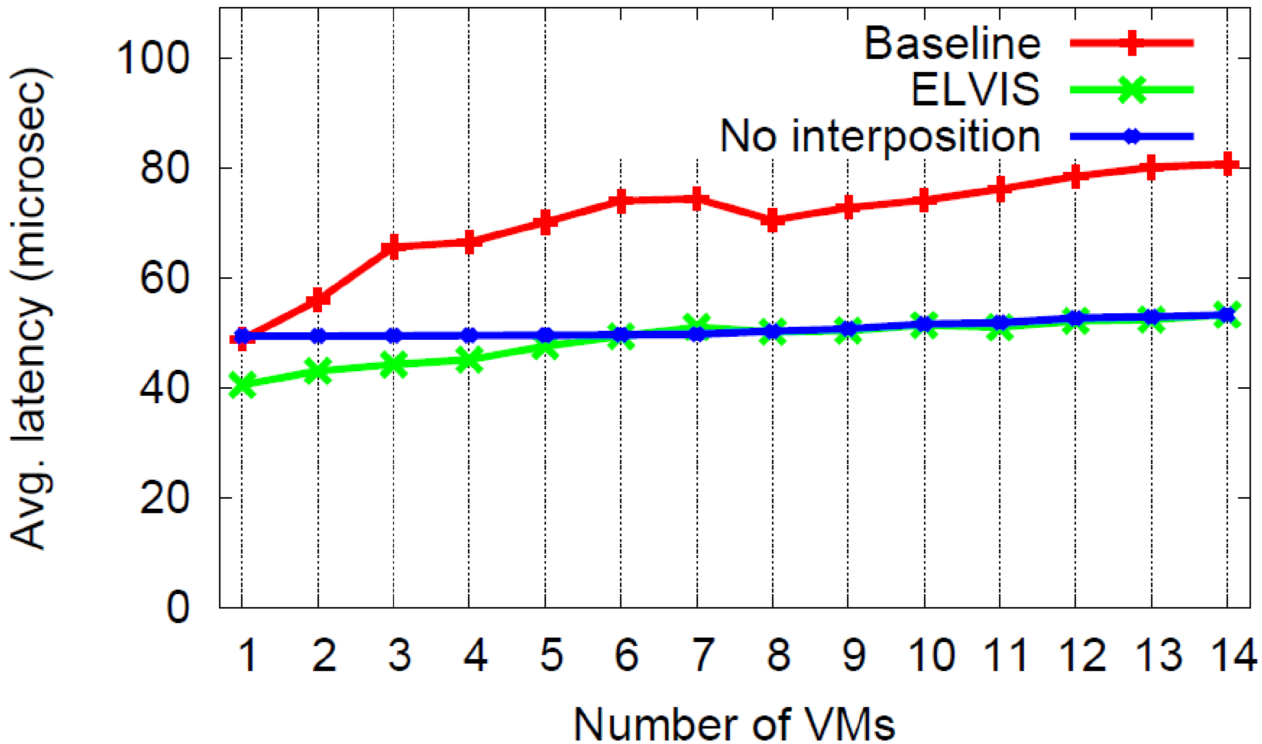


- Scaled perfectly
- 1 core managed to handle I/O for 7 VMs (cores)
- Maximum throughput
- Coalesced more interrupts than the SR-IOV device (4K-11K vs. 30K ints/sec)

1x10Gb port
ELVIS: 1 core dedicated for I/O and 1 dedicated core per VM (N+1 total)
Baseline: N+1 cores (to handle I/O and to run the VMs)
No Interposition: N cores to run the VMs

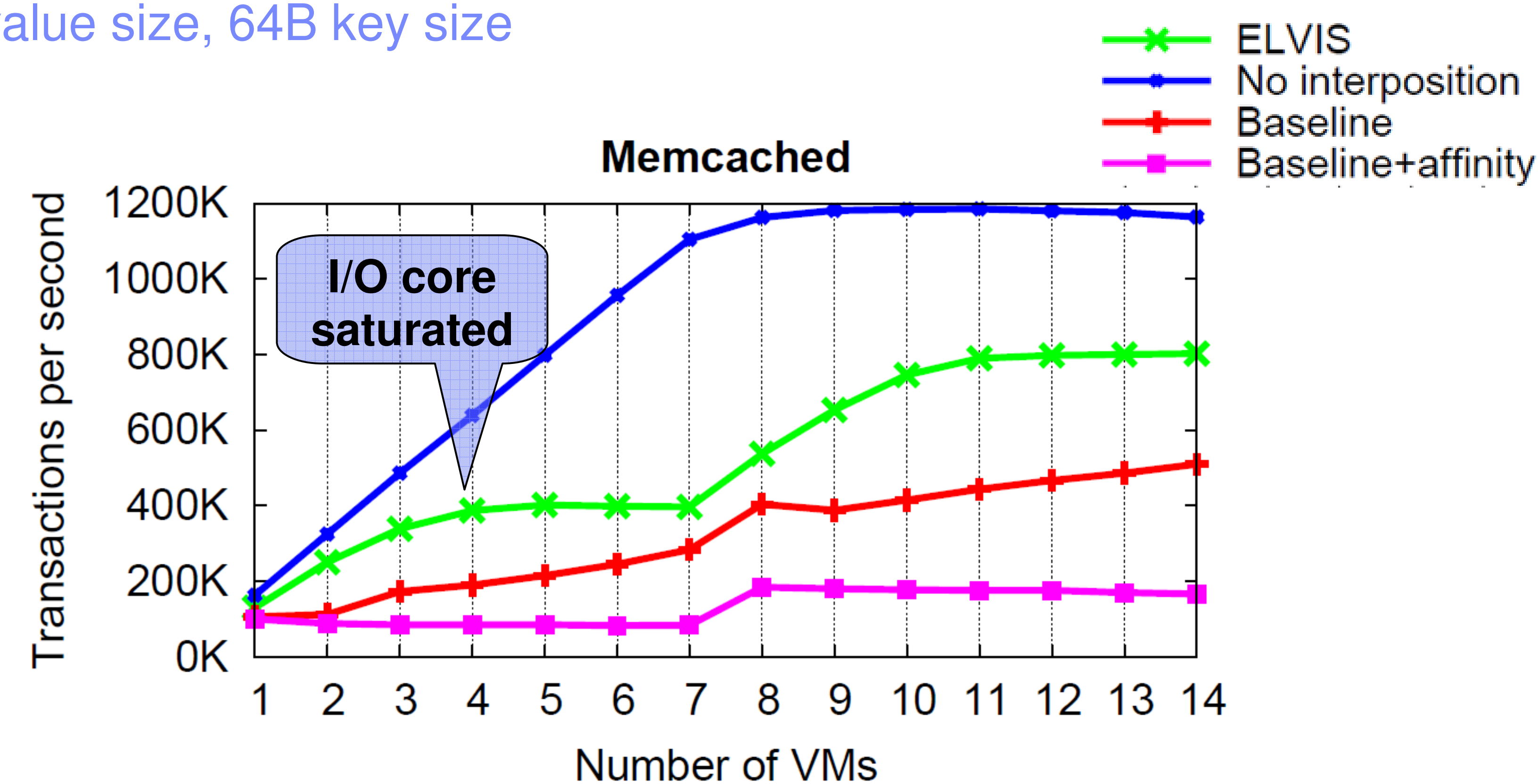
2x10Gb port
ELVIS: 2 cores dedicated for I/O and 1 dedicated core per VM (N+2 total)
Baseline: N+2 cores (to handle I/O and to run the VMs)
No Interposition: N cores to run the VMs

Netperf – UDP Request Response (latency sensitive)



- Latency slightly increased with more VMs
- Better than No Interposition in some cases because enabling SR-IOV in the NIC increases latency by 22% (ELVIS disables SR-IOV)

Memcached - 90% get, 10% set, 32 concurrent requests per VM
1KB value size, 64B key size



- I/O core saturated after 3 VMs
- ELVIS was up to 30% slower than No interposition when the I/O core was not saturated, but was always 30%-115% better than Baseline

Improving I/O Virtualization - Related Work

- Paravirtual I/O
- Polling
- Spatial division of cores / core dedication
- Exitless Interrupts

We extended many of these ideas and integrated them with a fine-grained I/O scheduling to build a new **Efficient** and **Scalable** paravirtual I/O System (ELVIS)

Conclusions and Future Work

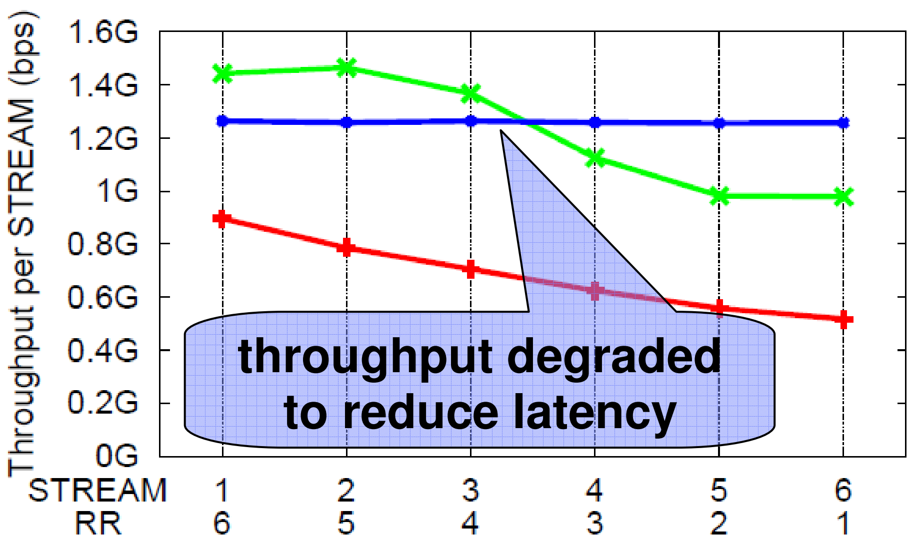
- Most data centers and cloud providers use paravirtual I/O (required to enable many useful virtualization features)
- Current trend towards multi-core systems and towards faster networks makes paravirtual I/O **inefficient** and **not scalable**
- ELVIS presents a new **efficient and scalable** I/O virtualization system that removes paravirtual I/O deficiencies
- Future Work
  - Improve fine-grained I/O scheduling to consider VM's SLAs
  - Dynamically allocate or release I/O cores based on the system load and guest's workloads
  - Core Specialization: I/O core <> VCPU cores

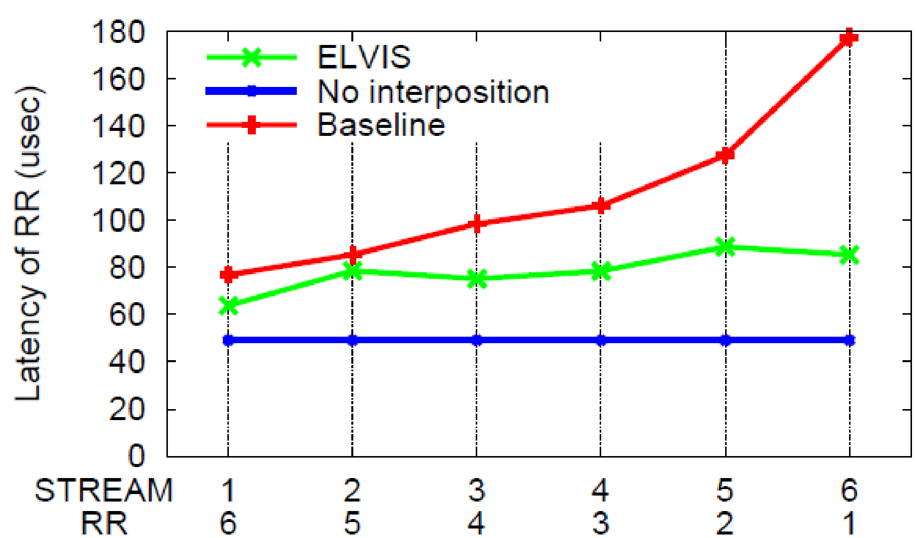
Questions ?

Backup

Mix of throughput intensive and latency sensitive VMs

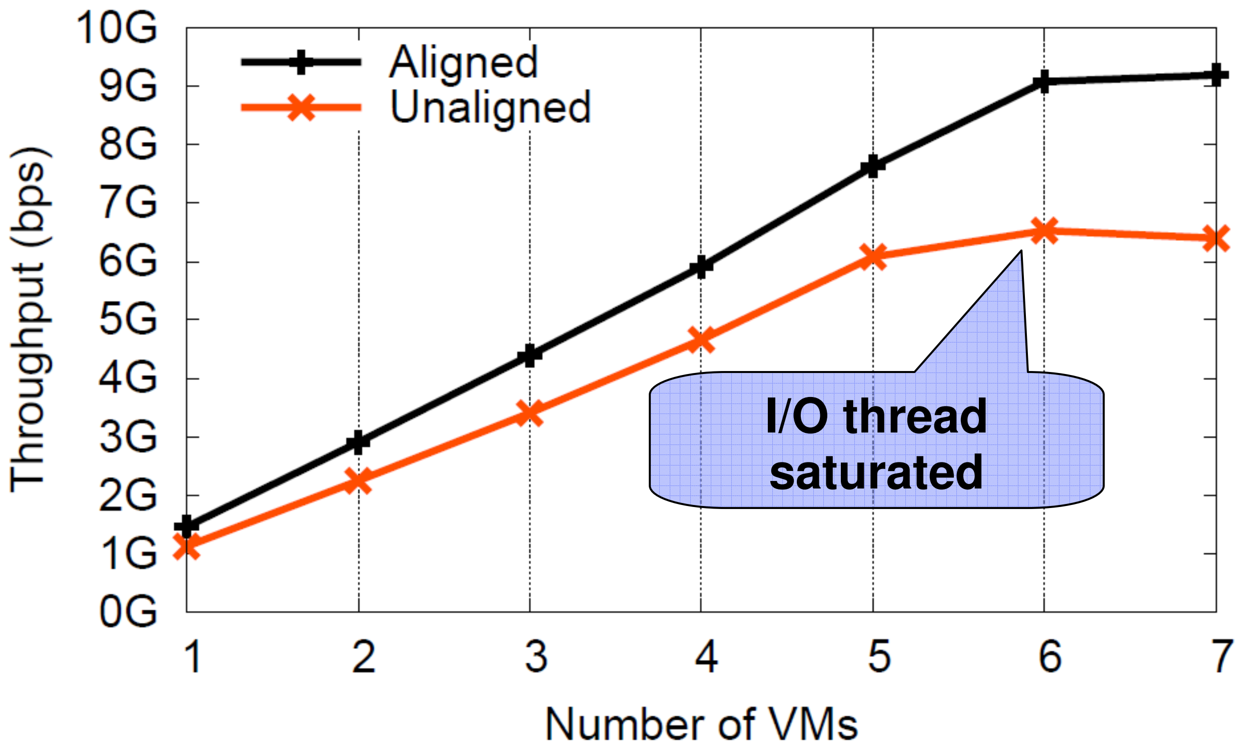
- Throughput intensive: N VMs run Netperf TCP Stream 64Bytes (STREAM)
- Latency sensitive: 7-N VMs run Netperf UDP Request Response (RR)
- N = 1 to 6





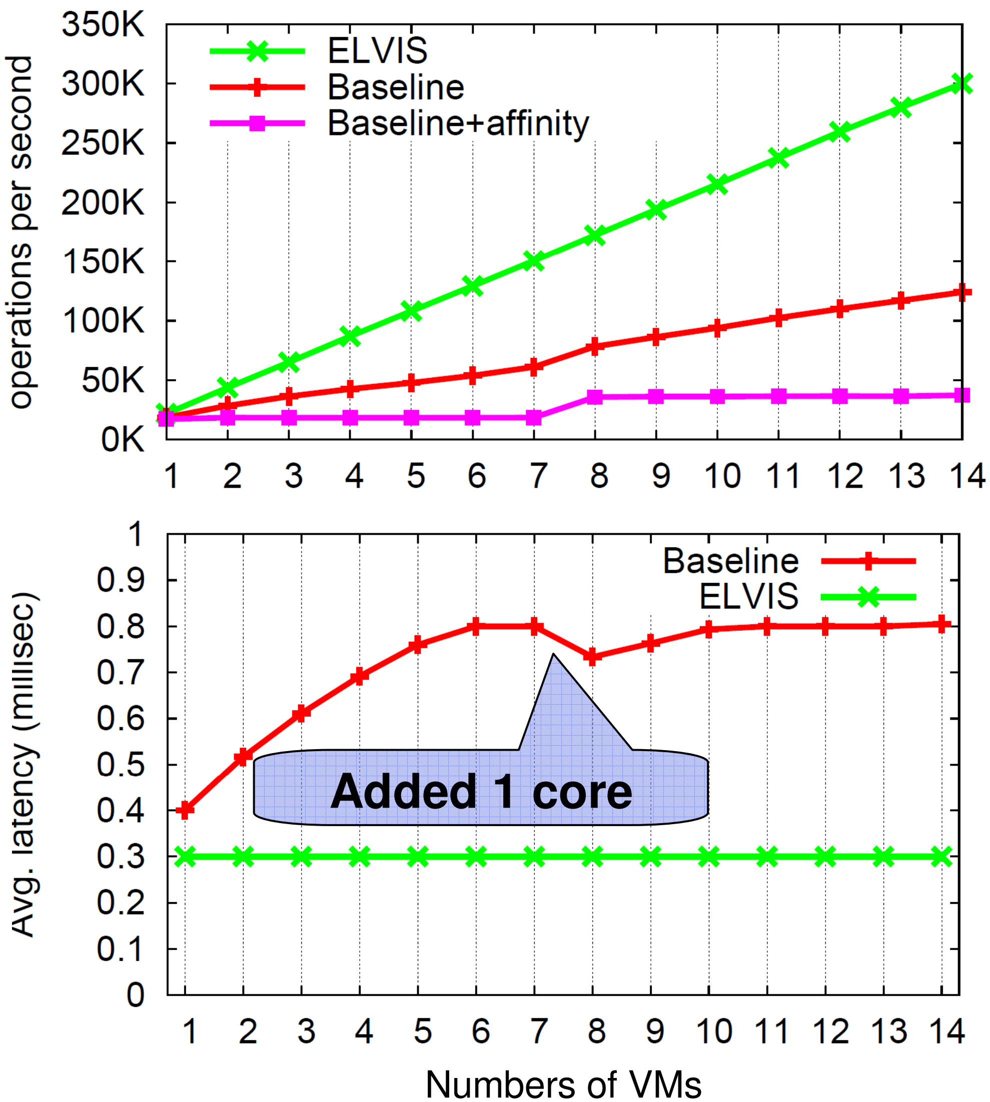
- Managed to balance between throughput intensive and latency sensitive workloads

NUMA awareness
Netperf – TCP Stream 64Bytes



- Aligned: improves performance by 30%-40% (I/O thread runs in the same socket)
- Unaligned: saturated after 5-6VMs (I/O thread runs in a different socket)

Filebench – block I/O interposition based on host RAM disk
4x4KB random writes, 4x4KB random reads per VM



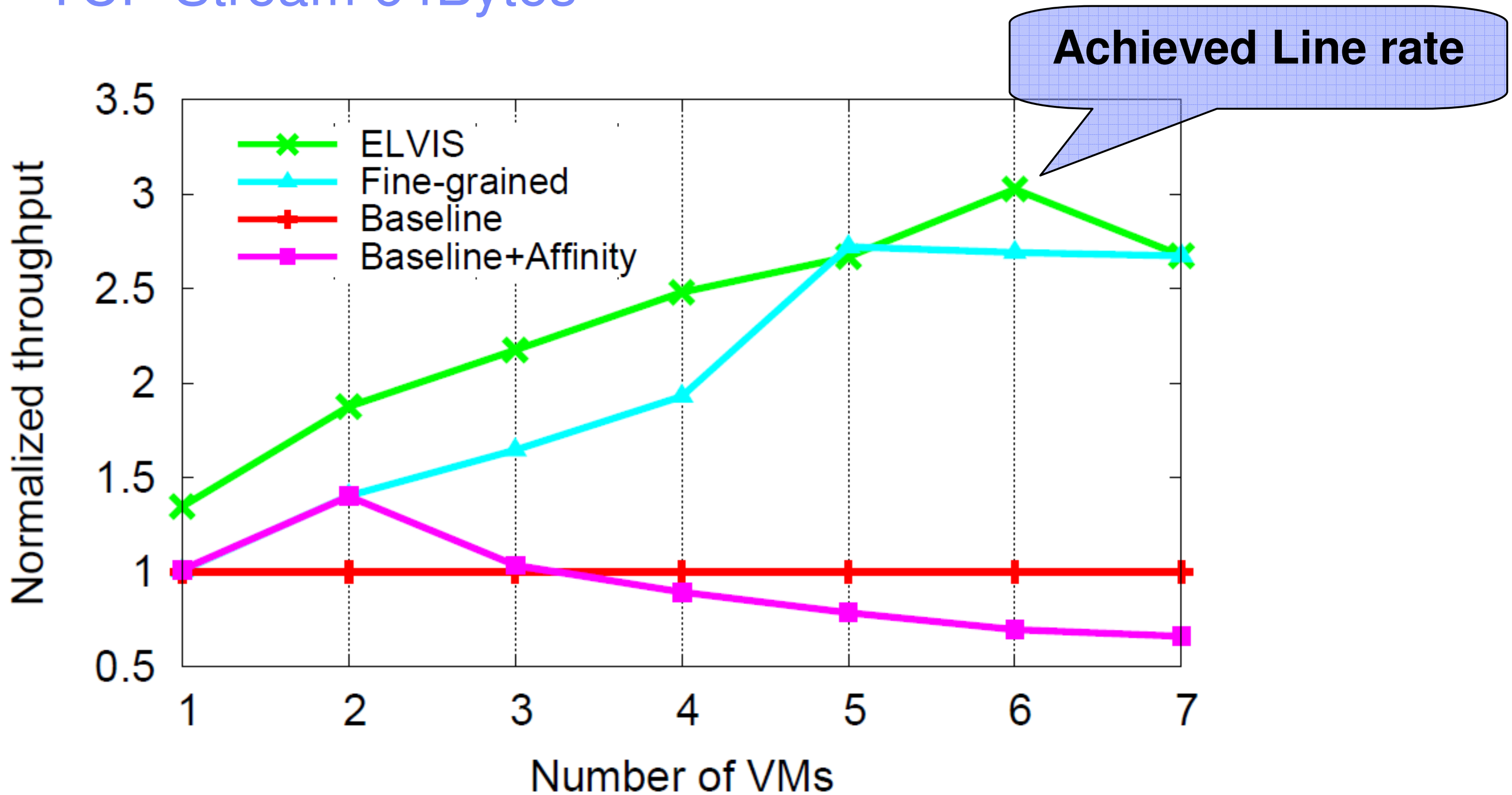
- Latency remains constant
- Throughput increases linearly

I/O becomes Exitless

	Baseline	ELVIS
NetPerf TCP stream		
Exits/s 1 VM	**142K**	**<800**
Exits/s per VM (7 VMs)	**53K**	**<800**
Apache		
Exits/s 1 VM	**109K**	**<800**
Exits/s per VM (7 VMs)	**39K**	**<800**
Memcached		
Exits/s 1 VM	**146K**	**<800**
Exits/s per VM (7 VMs)	**60K**	**<800**
Filebench		
Exits/s 1 VM	**56K**	**<800**
Exits/s per VM (7 VMs)	**35K**	**<800**

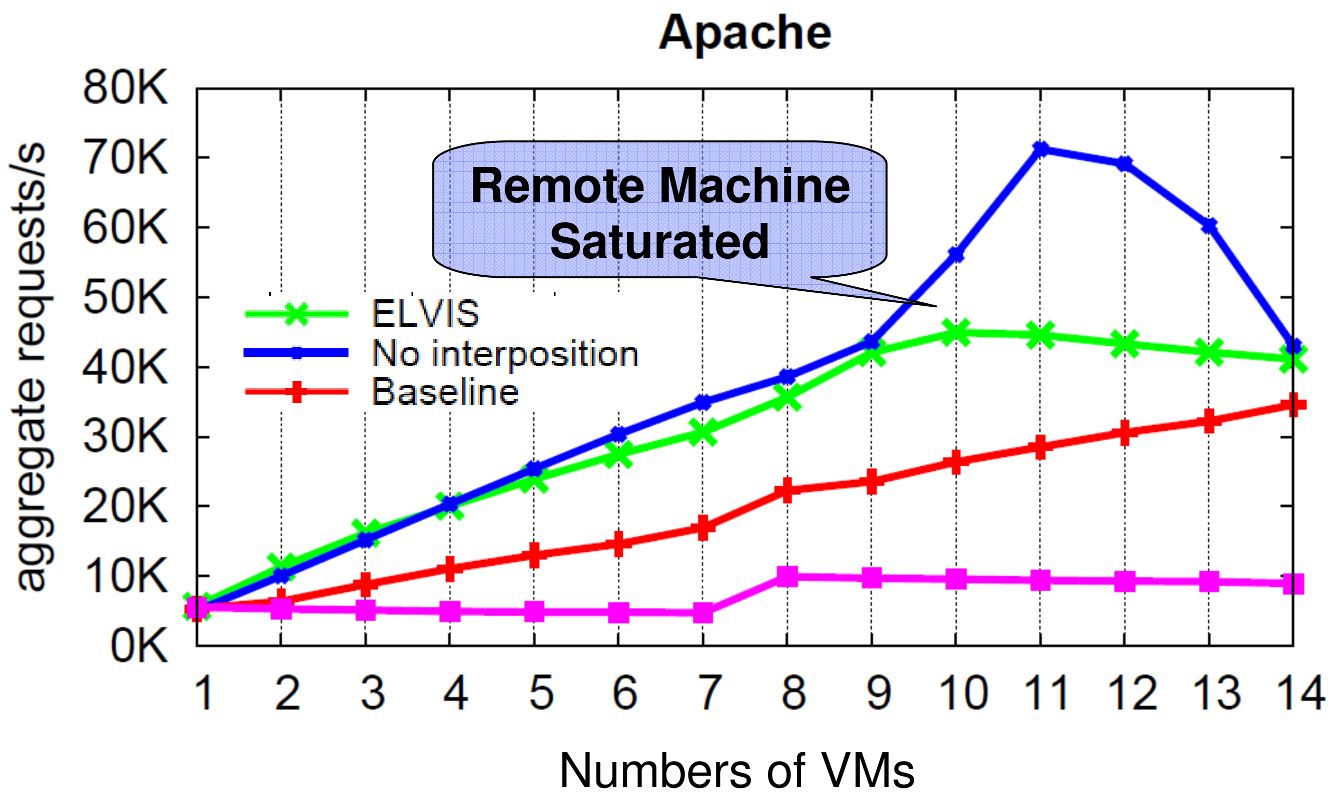
- Baseline: exits/VM decreased as the number of VMs increased (batching/coalescing effect)
- ELVIS: removed most of the exits!

Fine-grained I/O scheduling and Exitless requests/replies
Netperf – TCP Stream 64Bytes



- Fine-grained I/O scheduling is required to improve scalability
- Exitless notifications are required to improve per VM performance

Apache serving 4KB static pages



- Scaled perfectly while the remove machine was not saturated
- 1 core managed to handle I/O for 7 VMs (cores)
- Maximum throughput