

# Re-architecting Virtualization in Heterogeneous Multicore Systems

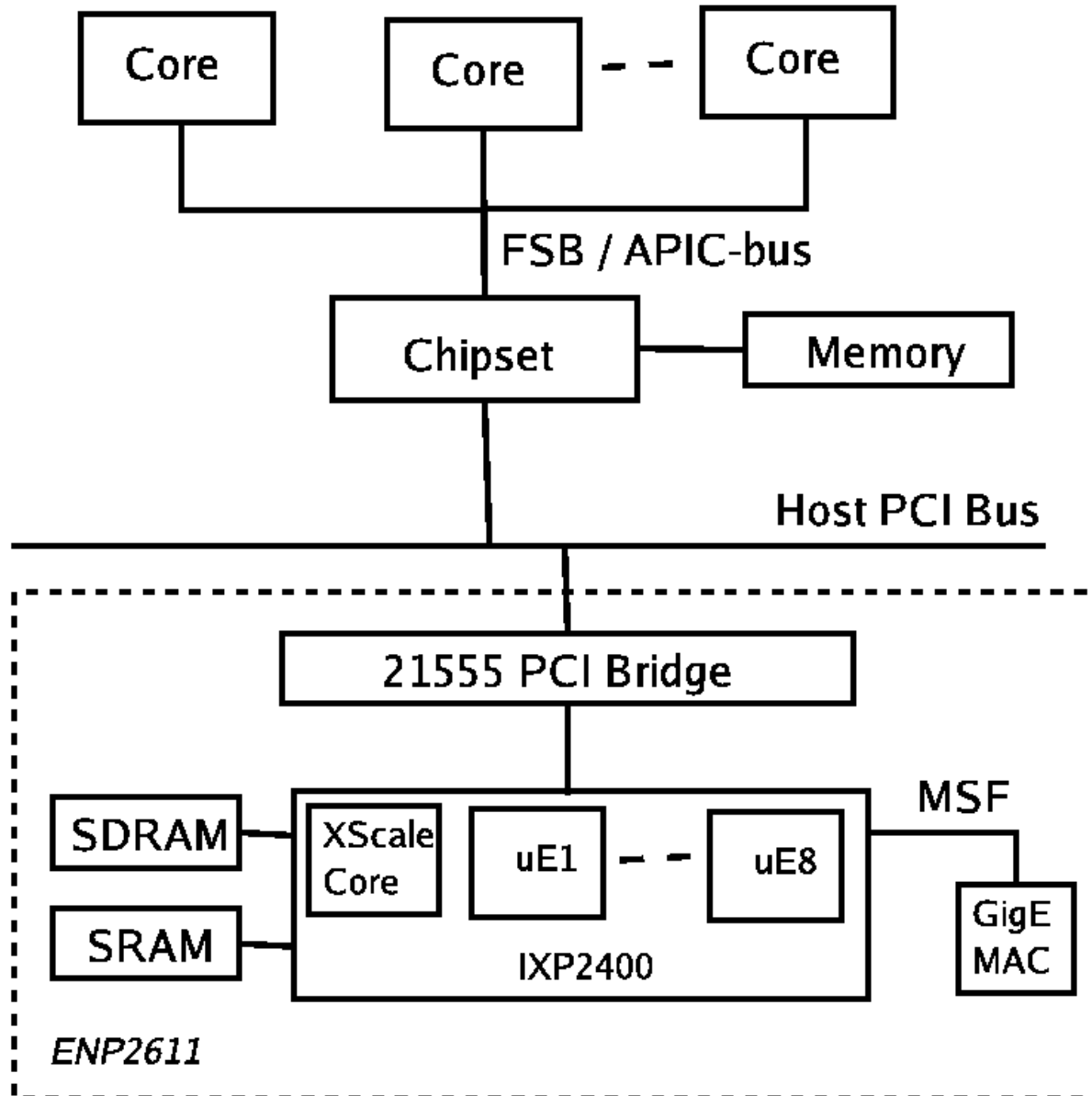
*Himanshu Raj*, Sanjay Kumar, Vishakha Gupta, Gregory  
Diamos, Nawaf Alamoosa, Ada Gavrilovska, Karsten  
Schwan, Sudhakar Yalamanchili  
College of Computing, School of ECE  
Georgia Institute of Technology

- Addressing Virtualization in Heterogeneous Multicores
  - Functional Approach
  - V-Core Approach
- Key Aspects
  - High performance
  - Scalability
  - Enhanced functionality
- Example: Functional Approach
  - Self-Virtualized NIC (SV-NIC) using IXP NP
- Ongoing Work: V-Core
  - Applying V-Core approach to IXP NP, Cell, N-VIDIA GPU and Virtex FPGAs
  - Research Challenges

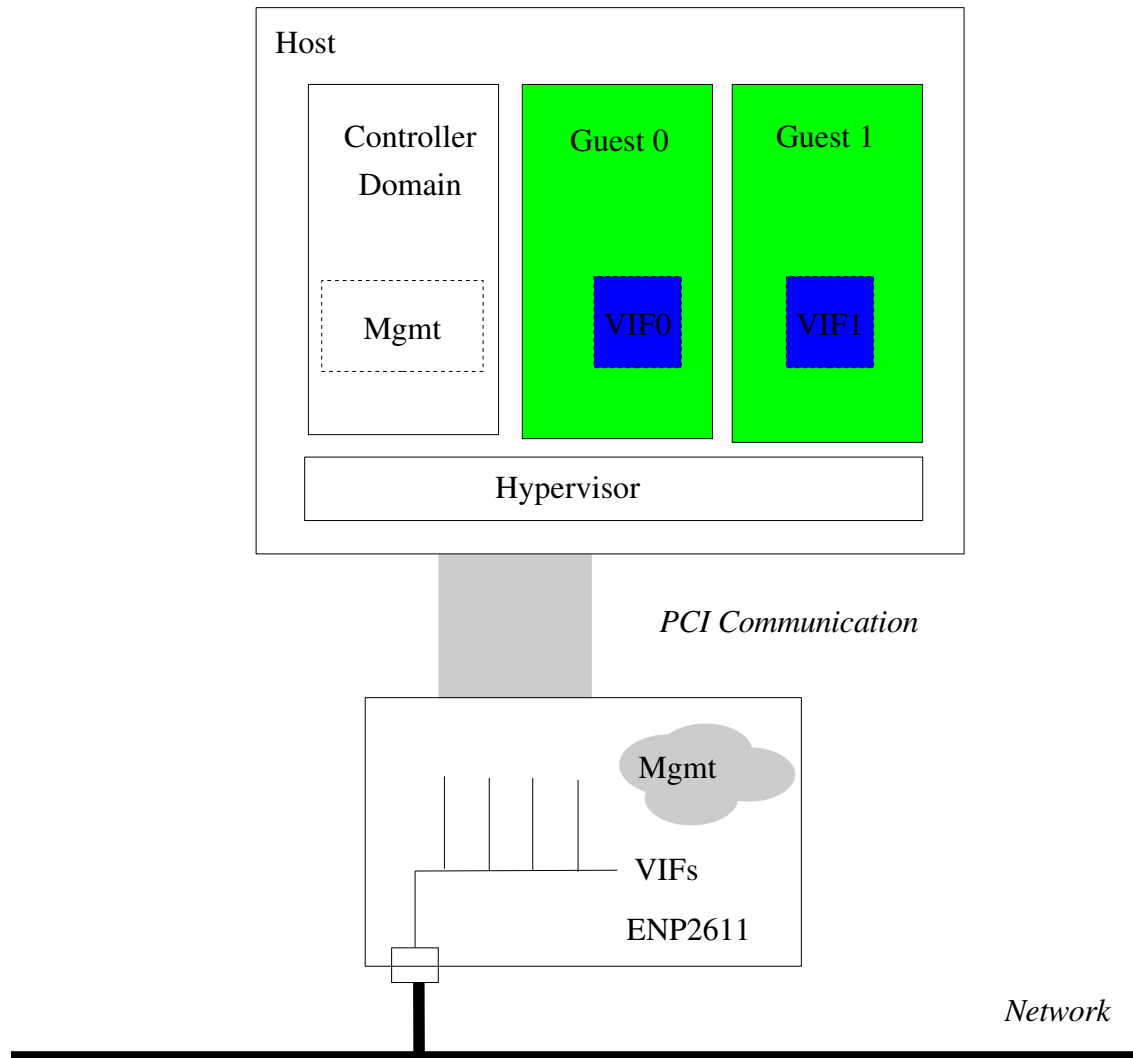
# Functional Approach

- Use Heterogeneous Core to Implement Certain Virtual Platform Related Functionality
- Guest VMs Oblivious to type of Core(s)
- Example: Self-Virtualized I/O Devices
  - Offload I/O virtualization to *smart peripherals*
  - *Componentization, Functional Partitioning of Cores (Utilize processing near physical device for fast communication), Optimal Resource Placement (Efficiently utilize the I/O interconnect between device and host cores), Minimization of Host Interactions in I/O Path, Utilize Specialized Cores*
  - Functionalities
    - Management of virtual devices
      - Creation/Removal/Reconfiguration
    - I/O
      - Between guest VMs and physical device via scalable *multiplexing/demultiplexing* of virtual devices

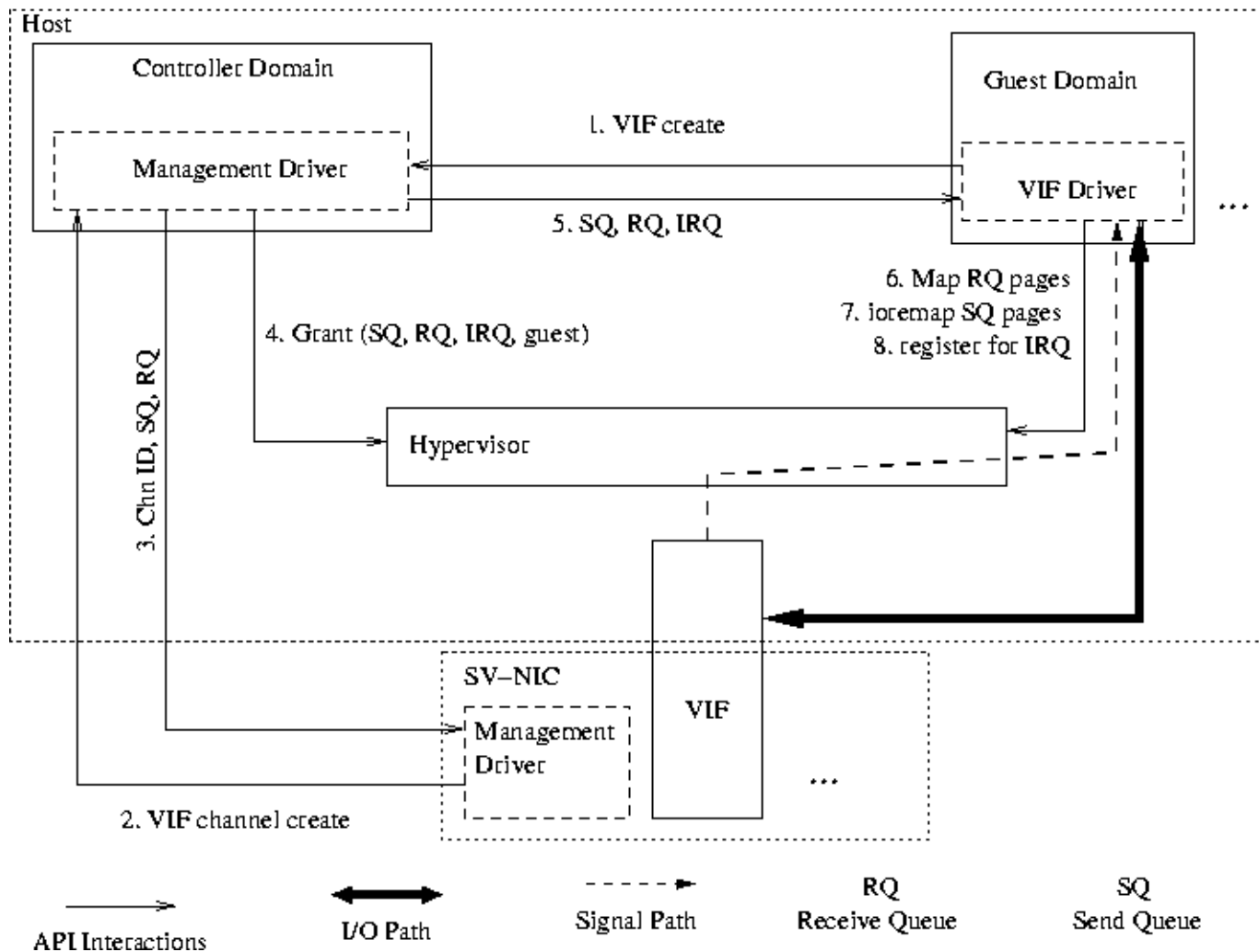
# Smart Device – ENP2611



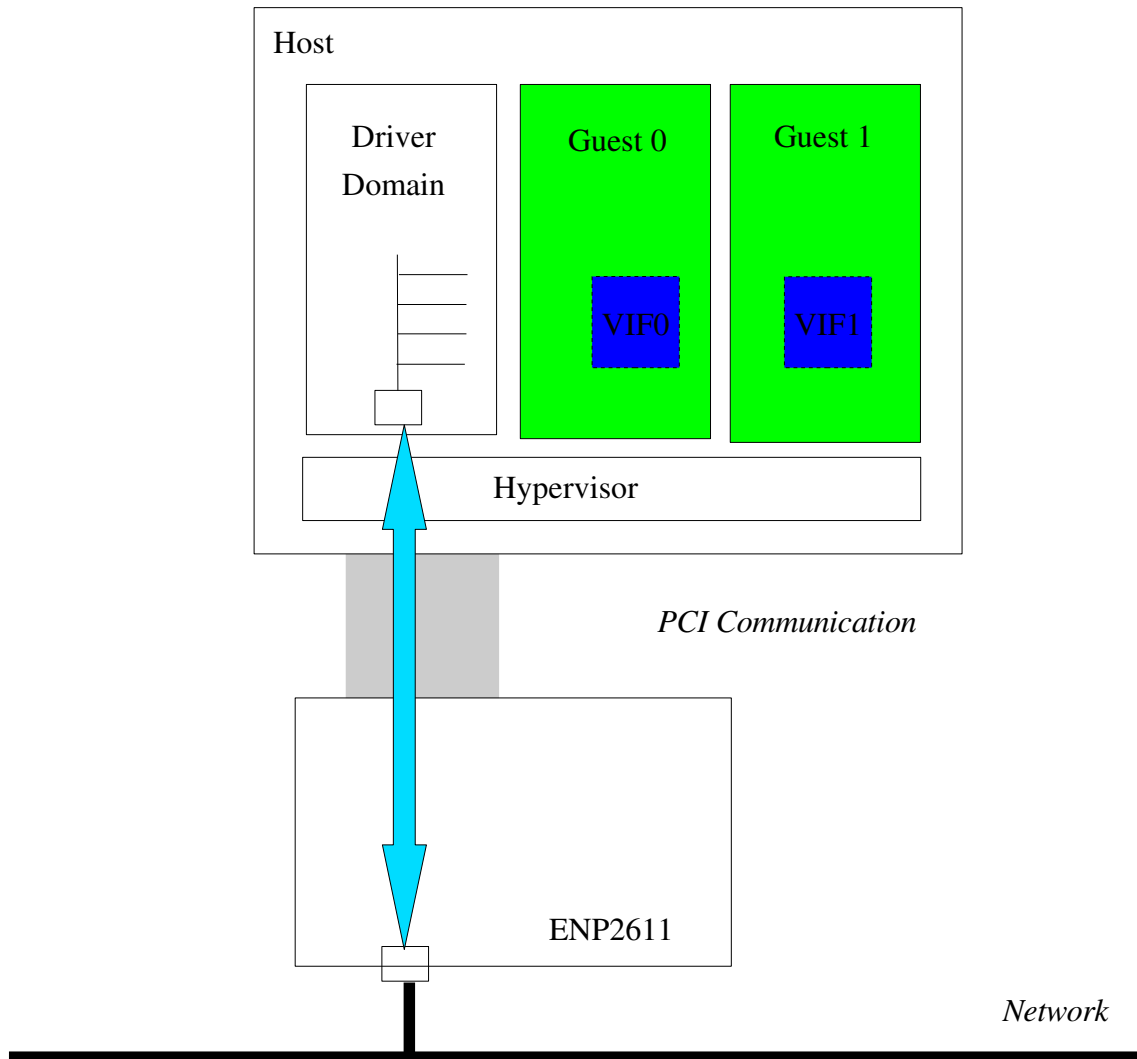
# SV-NIC



# SV-NIC – HV Interaction Example

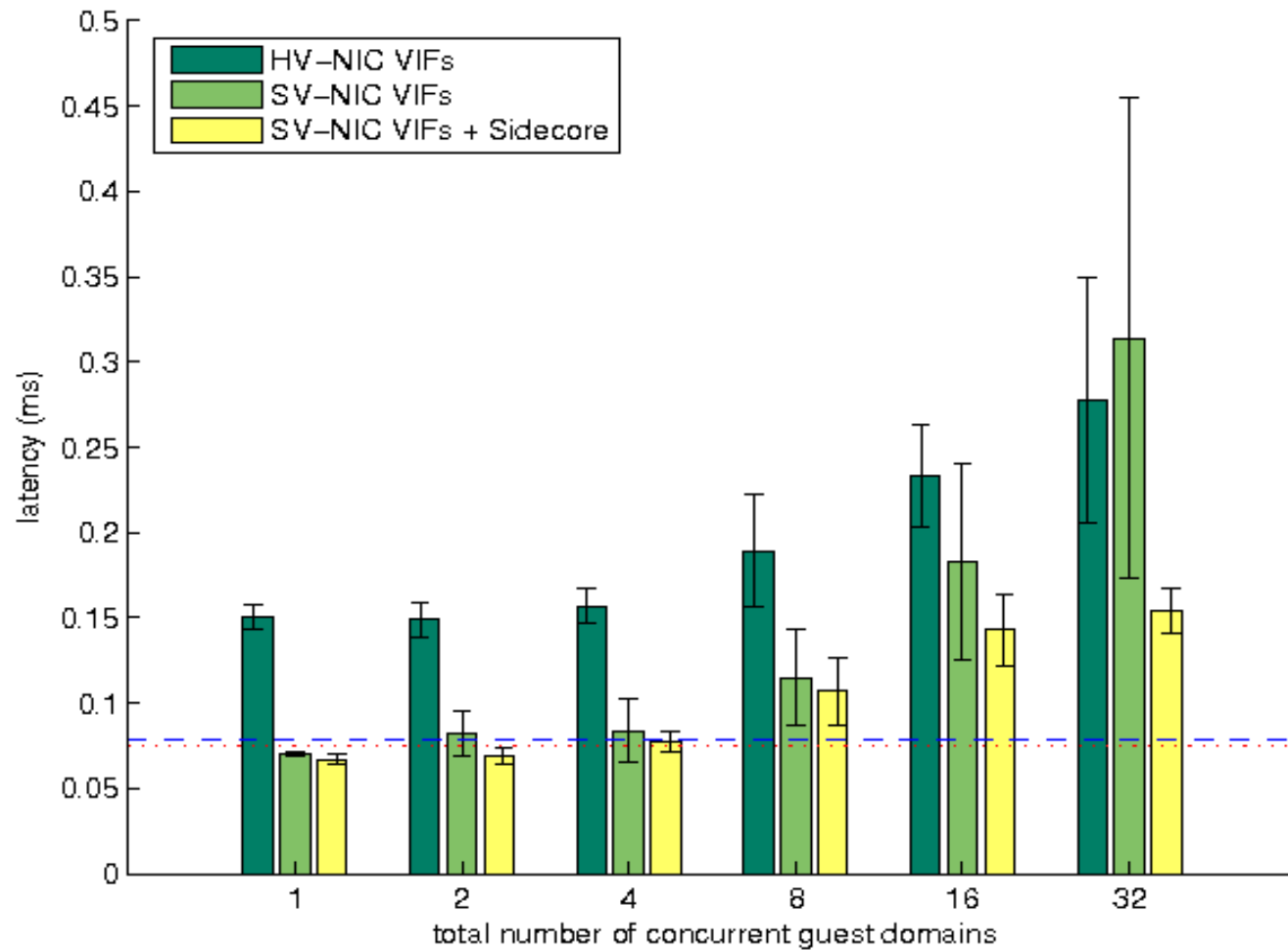


# HV-NIC

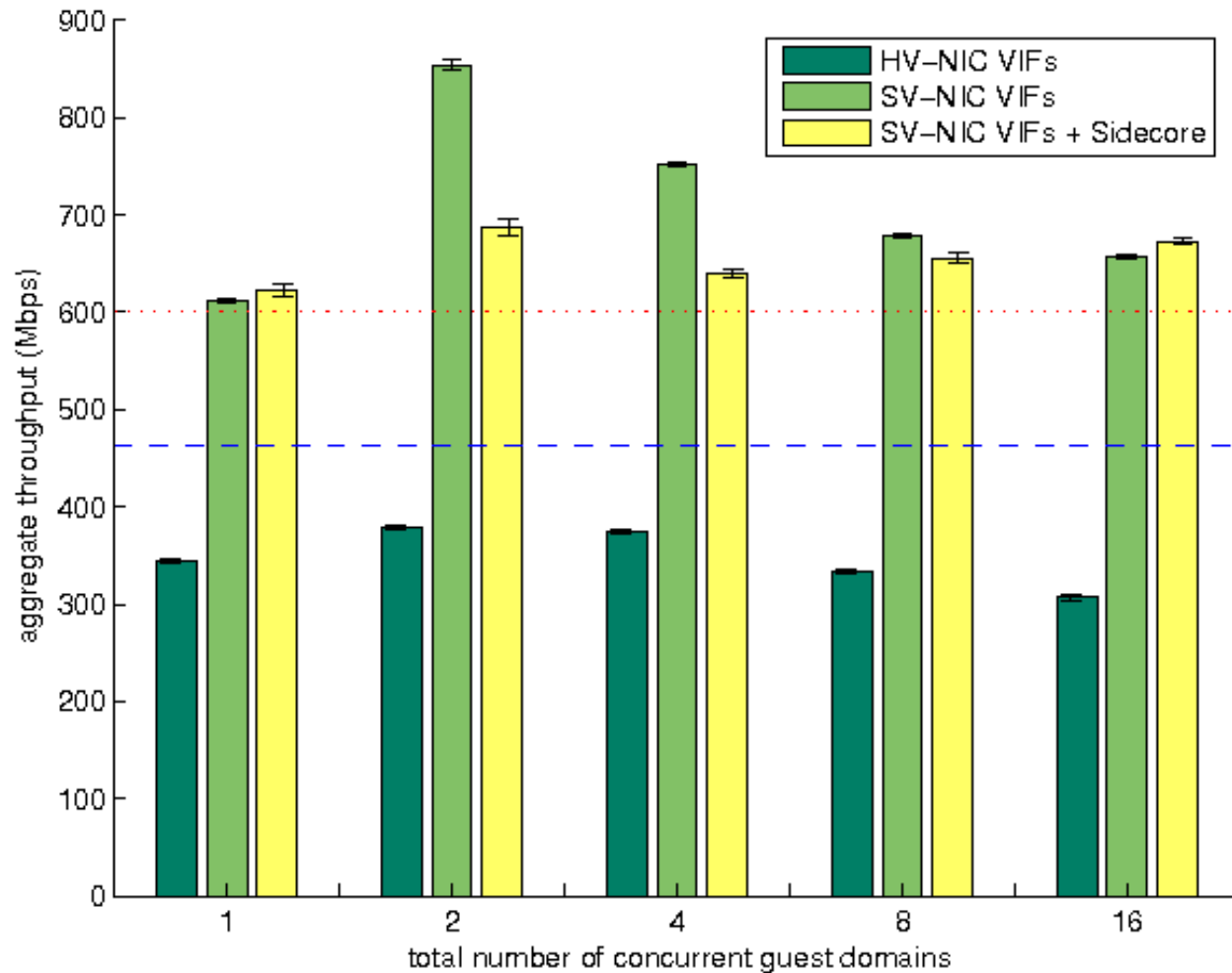


- PCI Implications
  - Asymmetric performance
  - Asymmetric resource access
    - S/W I/O MMU for I/O translation
- Signaling
  - Via PCI interrupt
    - Virtualization via Interrupt Identifier
  - Via asynchronous messaging
    - *Sidcore*

# Experimental Results – Latency



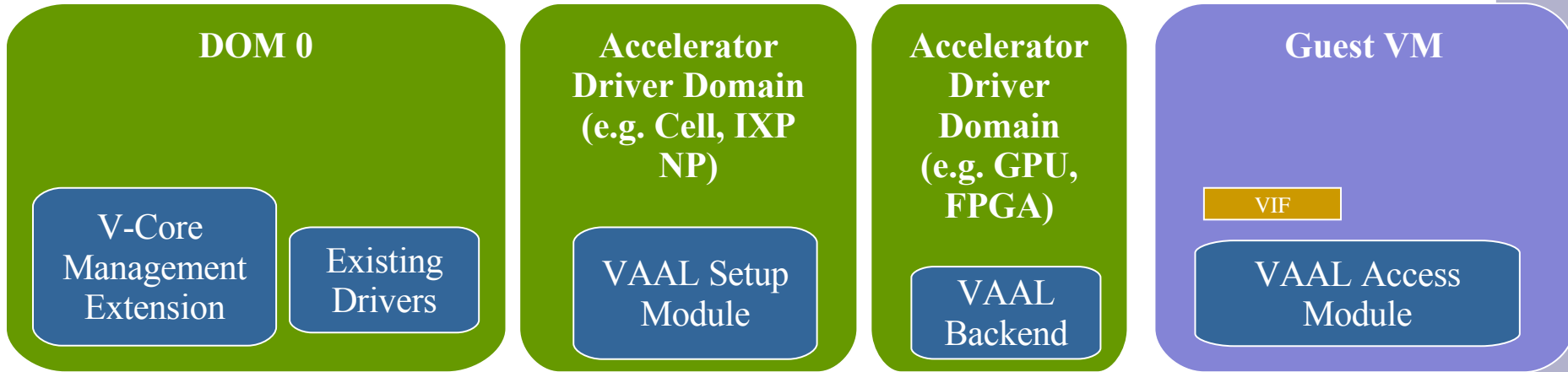
# Experimental Results – Throughput



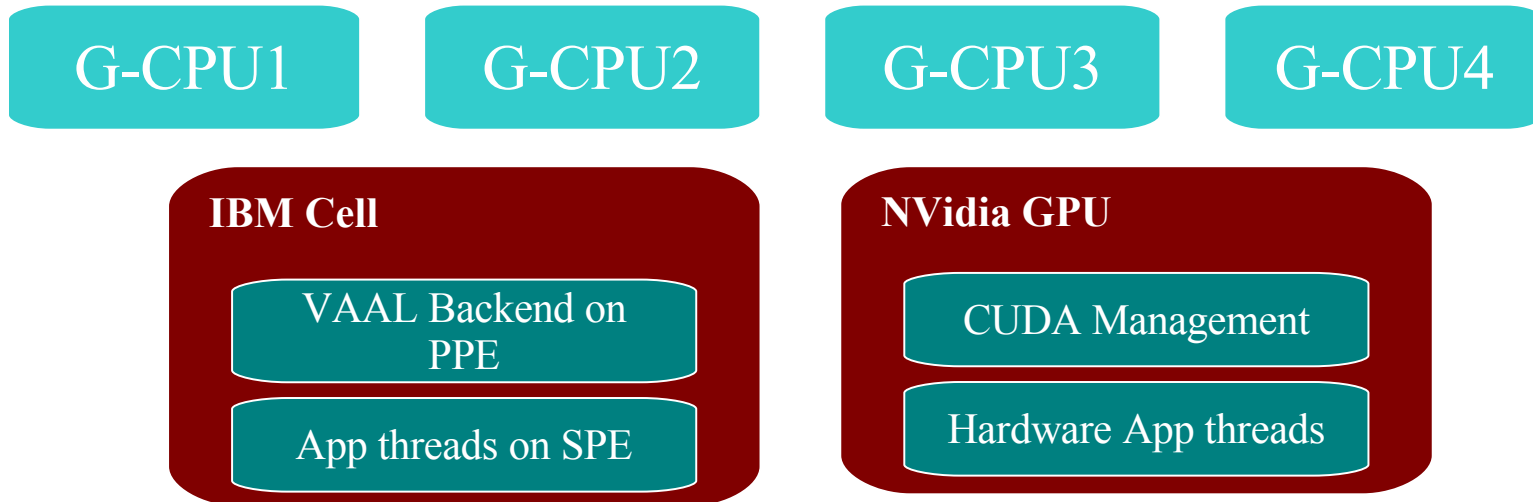
# V-Core Approach

- Provide Virtual Heterogeneous Cores to Guest VMs
- Extensions to Hypervisor and Management Domain
  - Efficient co-scheduling of accelerators at hypervisor
  - Resource allocation and VM creation with admission control at dom0
- Virtualized Accelerator Abstraction Layer (VAAL)
  - Enable interaction between VMs and devices
  - Setup corresponding control and data path

# V-Core Implementation Architecture

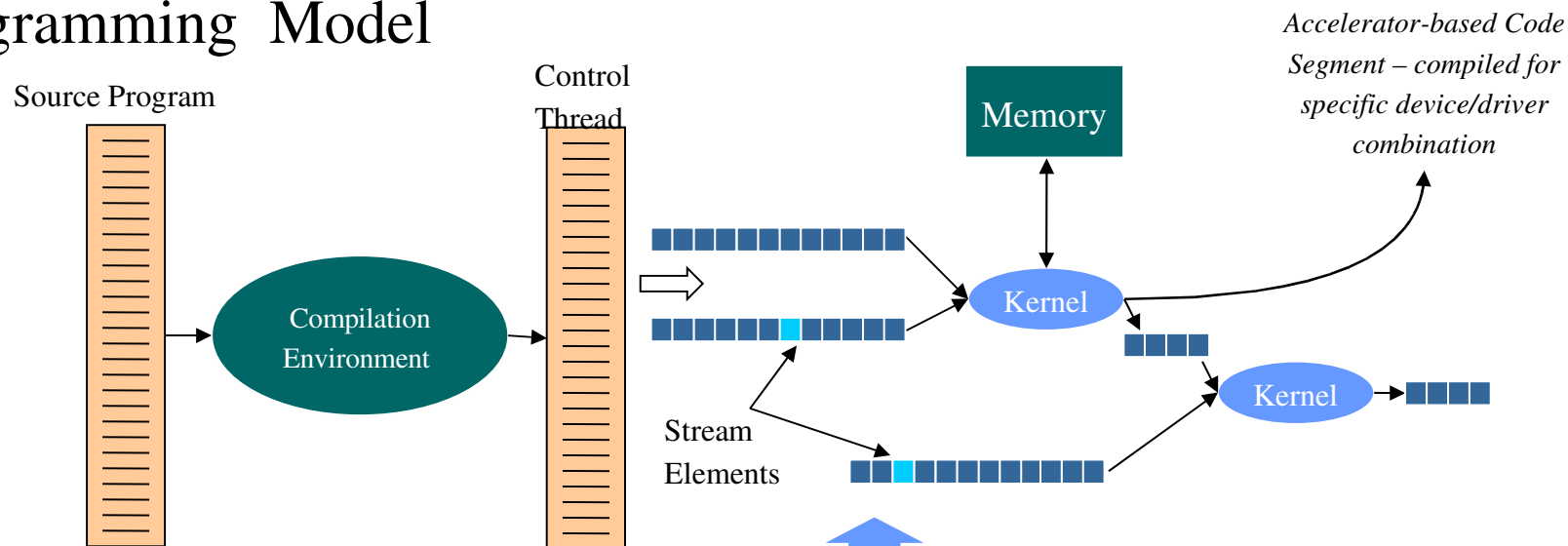


## Xen Hypervisor with V-Core Scheduling Extension

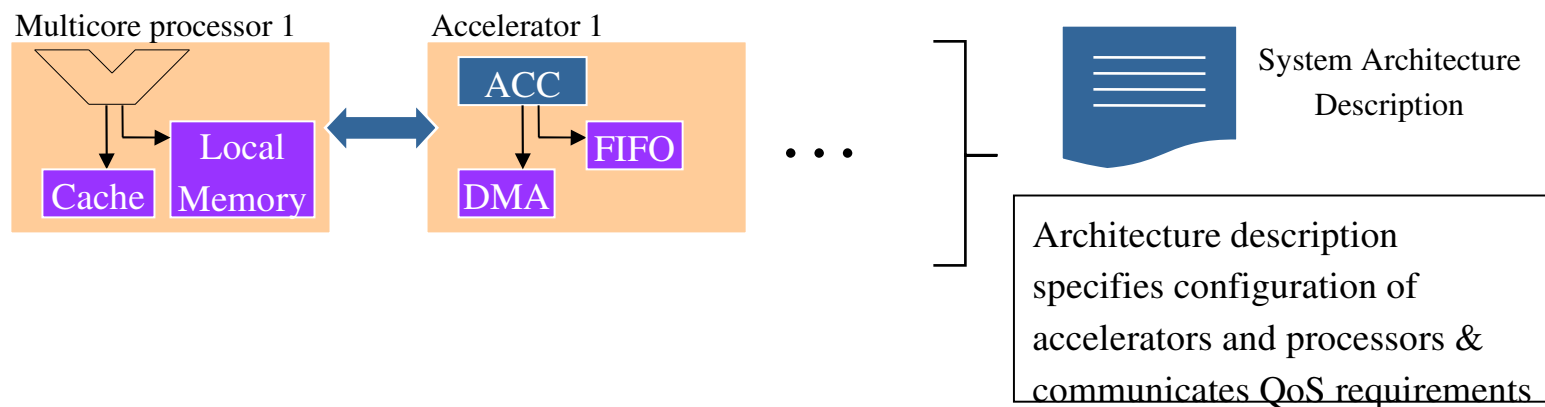


# VM Execution Model

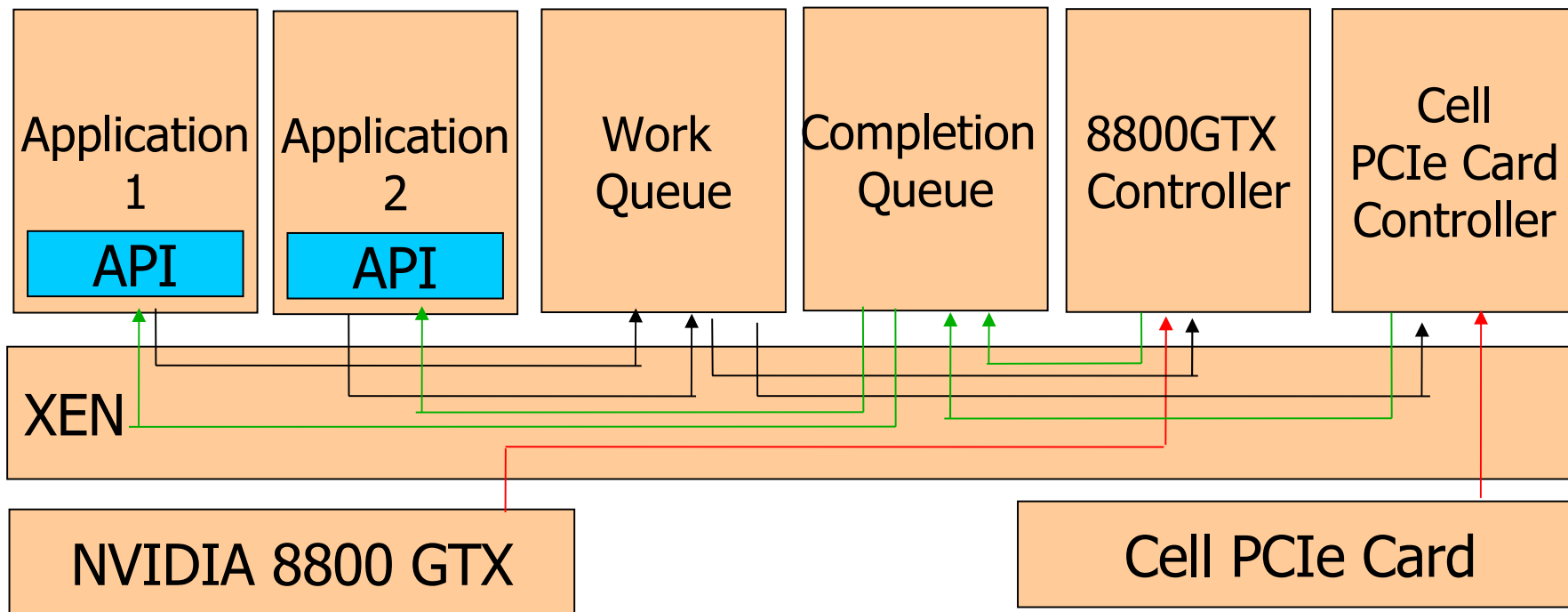
## Programming Model



## Configuration of the Machine Model



- Separate domains for applications, work queues, accelerator controllers
- XEN maps physical accelerator resources to controllers
- XEN provides virtual interfaces for inter-domain communication



# V-Core API

- Guest VM/Application API
  - Work queue
  - Completion queue
  - Dependence checking and synchronization
- Accelerator Controller API
  - NVIDIA, FPGA, Cell, IXP NP

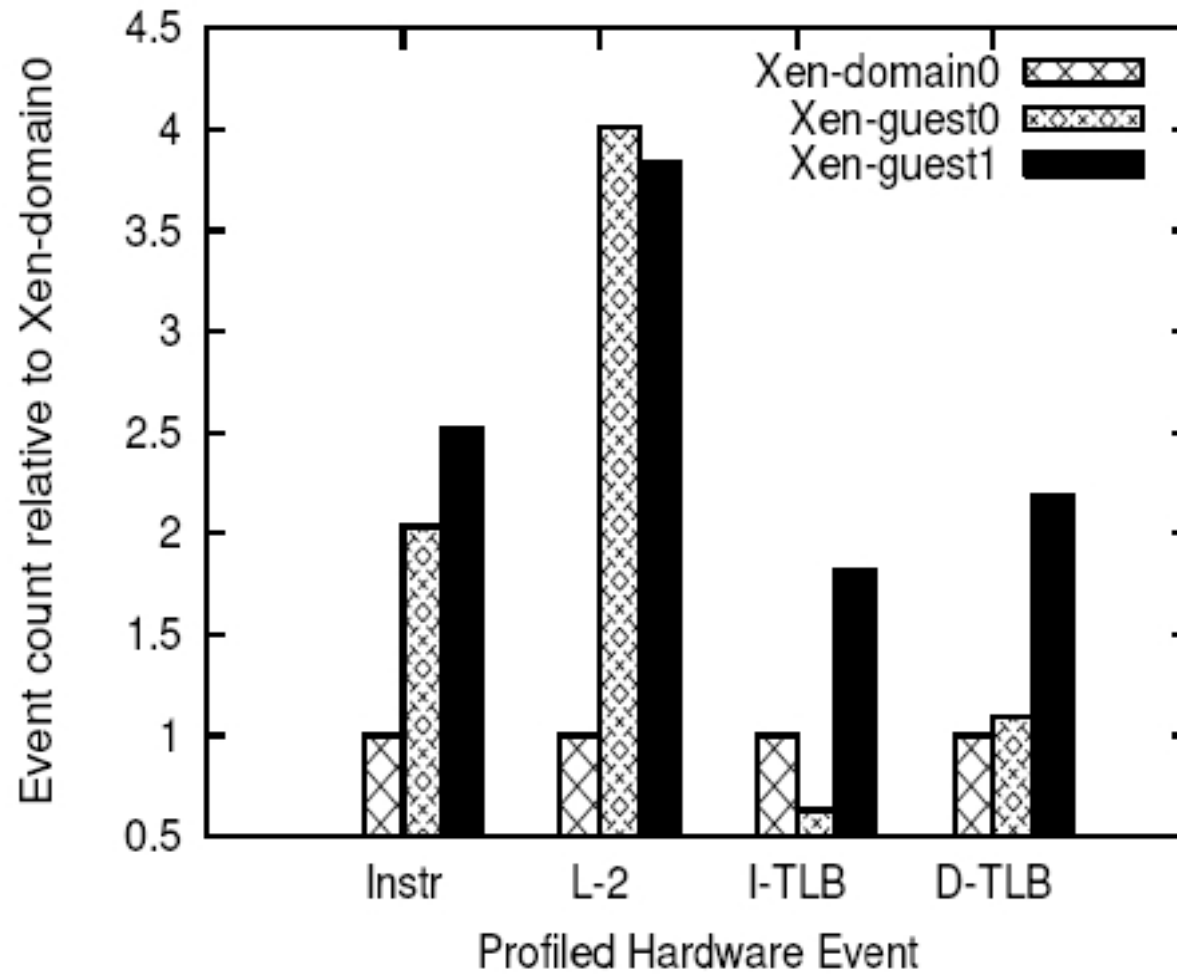
# Research Challenges

- System Definition Language
  - Describe the desired target VM
  - Communicate QoS requirements
- System Partitioning & Allocation
  - Allocate physical resources for a VM
  - Strategies to reduce the complexity of the problem
- Scheduling
  - Co-scheduling accelerators in a VM
  - Static and run-time schedulers: real-time vs. best effort
  - Space vs. time-sharing among VMs, based on capabilities
- Applications

Thank You

## Extra Slides

# Cost of I/O Virtualization



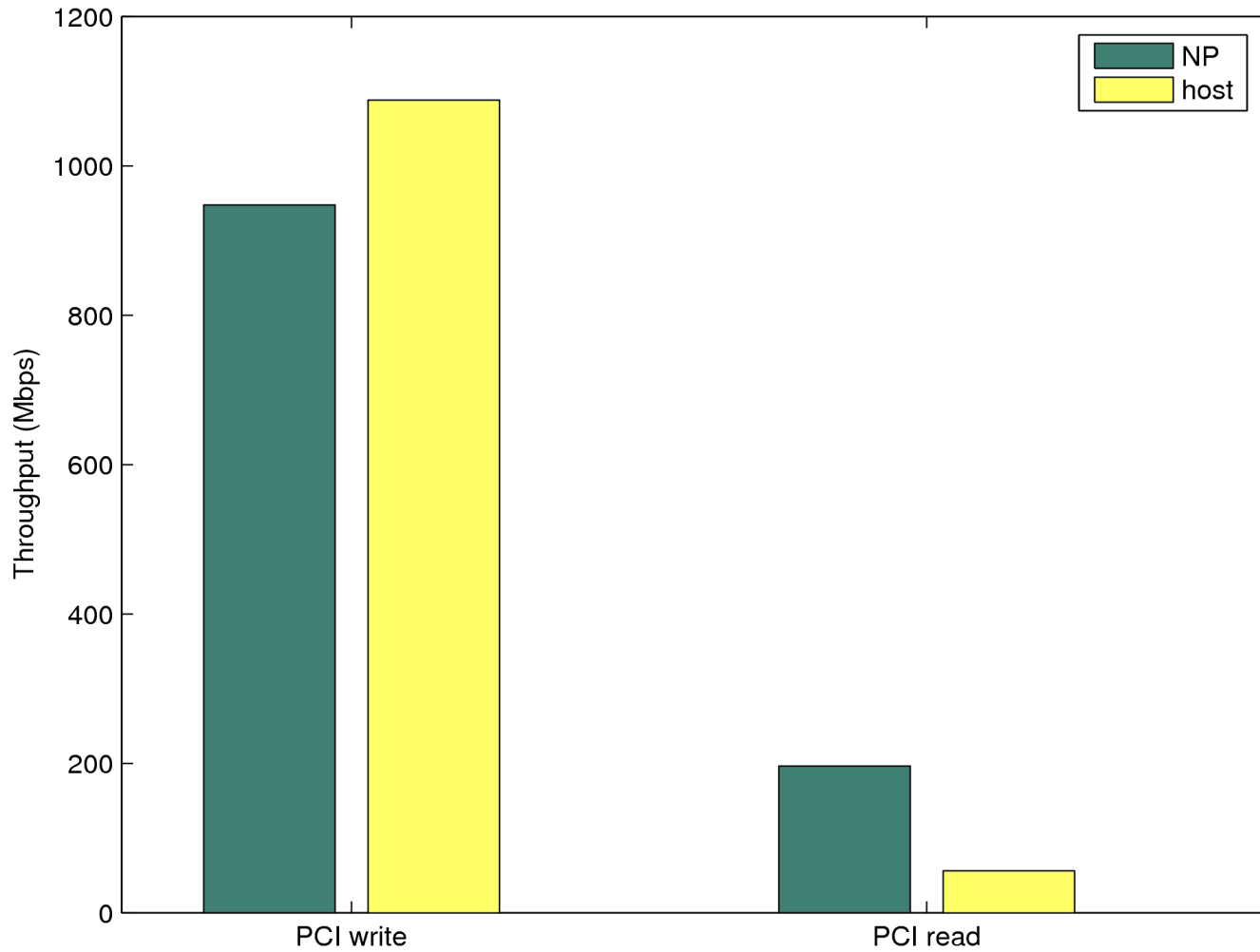
Source: Menon et al, VEE'05

# Functionalities Breakdown

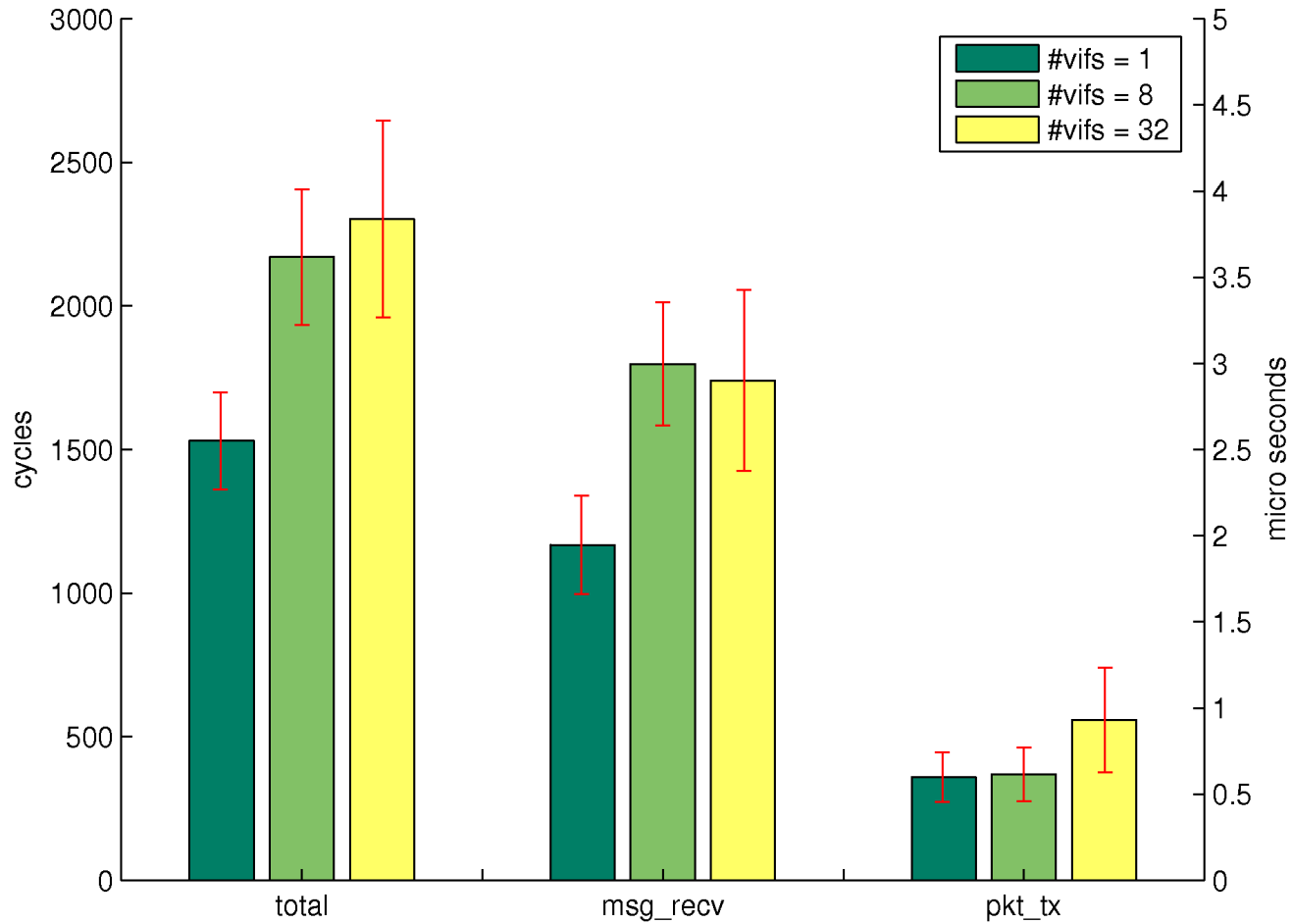
- Virtual Interface Management
  - Jointly done by host CPU and XScale core
  - API for
    - Creating new VIFs
    - Removing existing VIFs
    - Configuration changes
- Network I/O
  - Jointly done by host CPU and IXP micro-engines

- Host Configuration
  - Dell PE2650 server, 2-way HT Xeon 2.8GHz
  - 2GB RAM
  - Xen 3.0-unstable
  - 2.6.16 Linux para-virtualized kernel for Dom0 and DomU
  - BVT scheduling
- ENP2611 Configuration
  - 256 MB SDRAM, 8MB SRAM
  - 2.4.18 Linux kernel

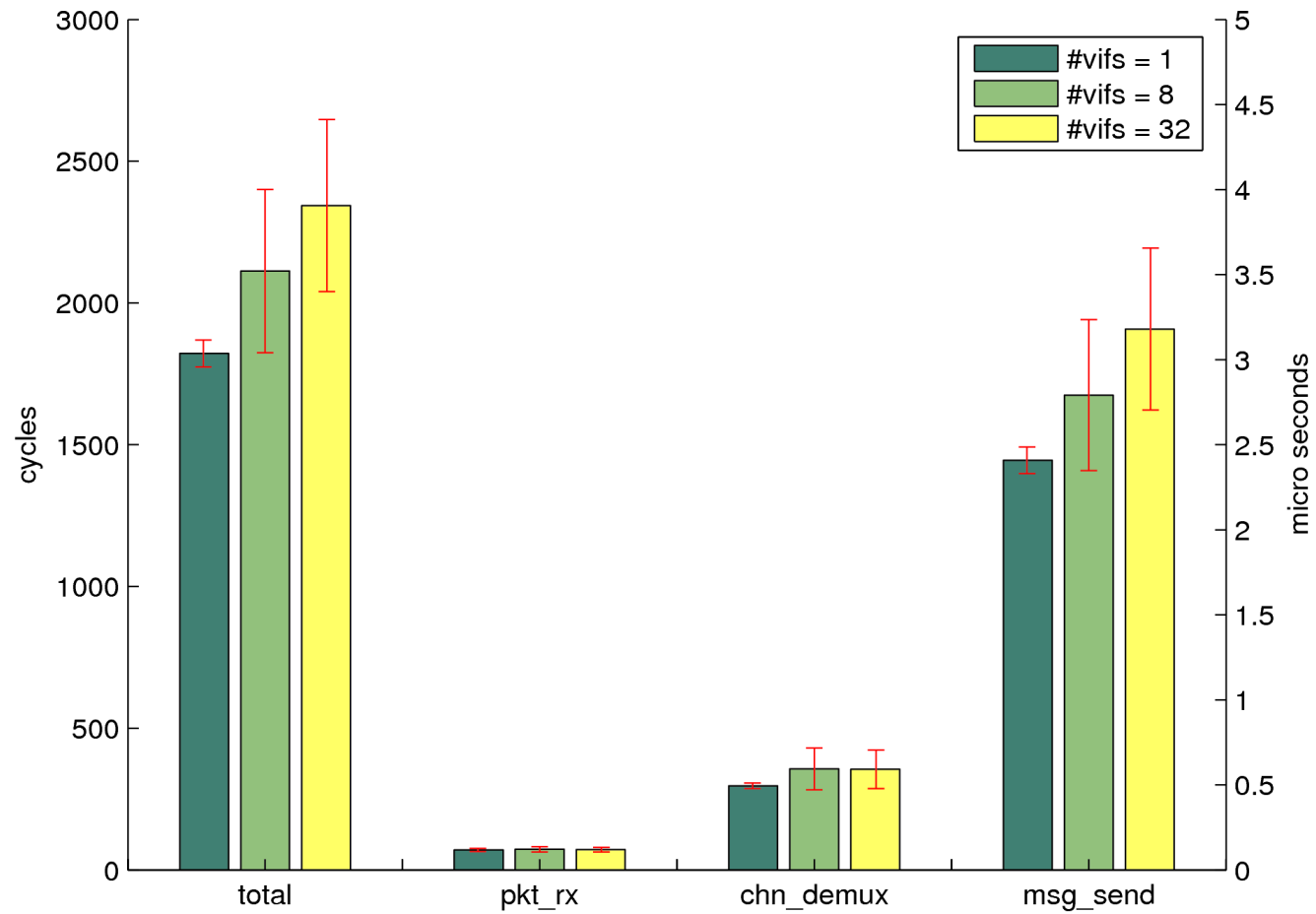
# PCI Asymmetry



# Micro-Benchmarks – Egress (SV-NIC)

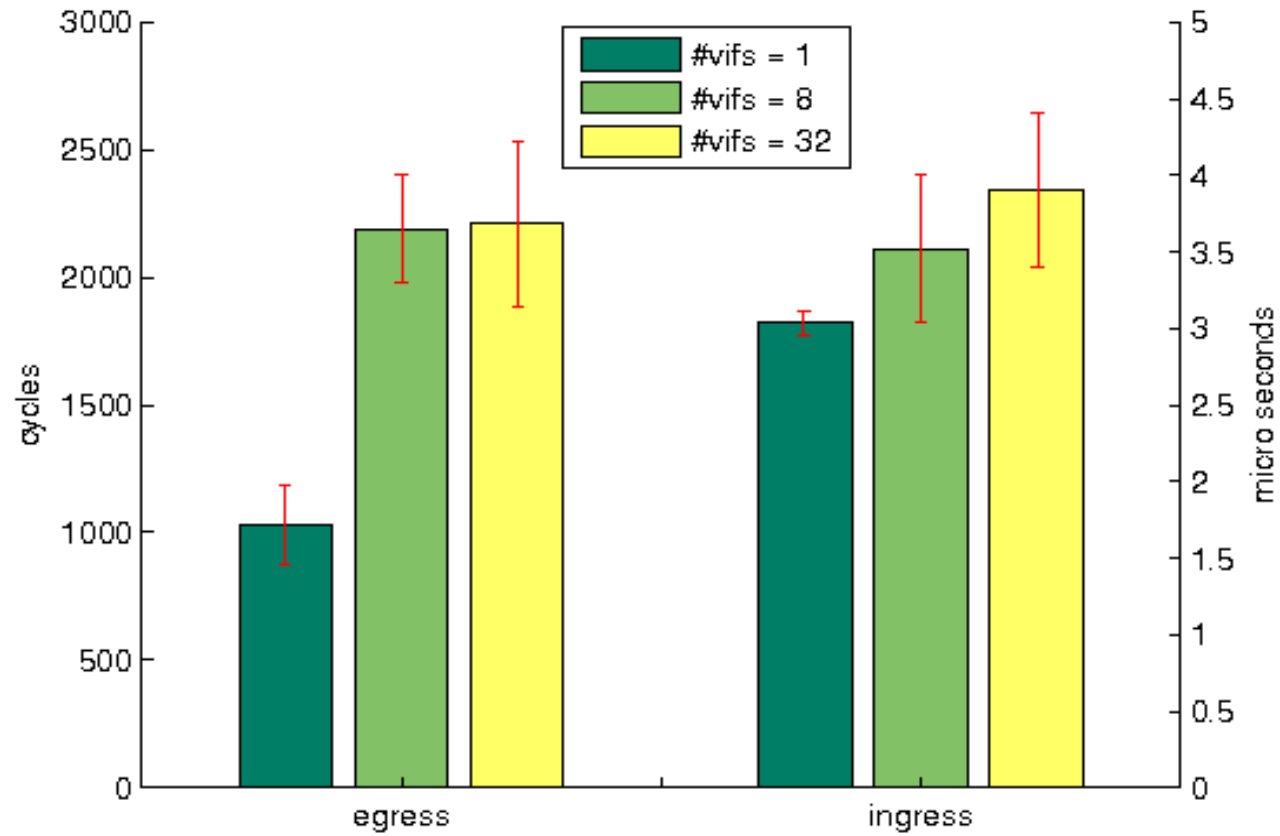


# Micro-Benchmark – Ingress (SV-NIC)



Pkt size = 64 bytes

# Microbenchmarks

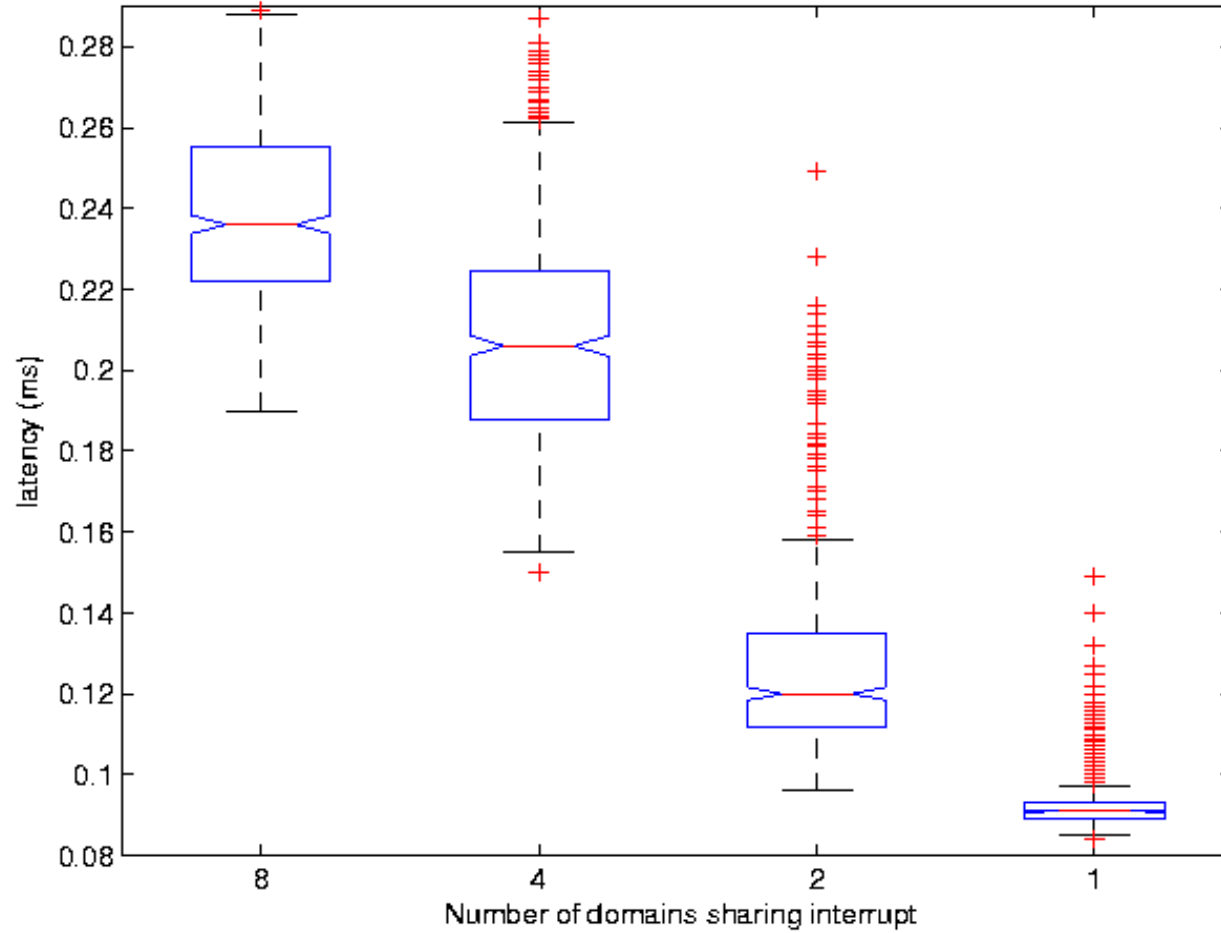




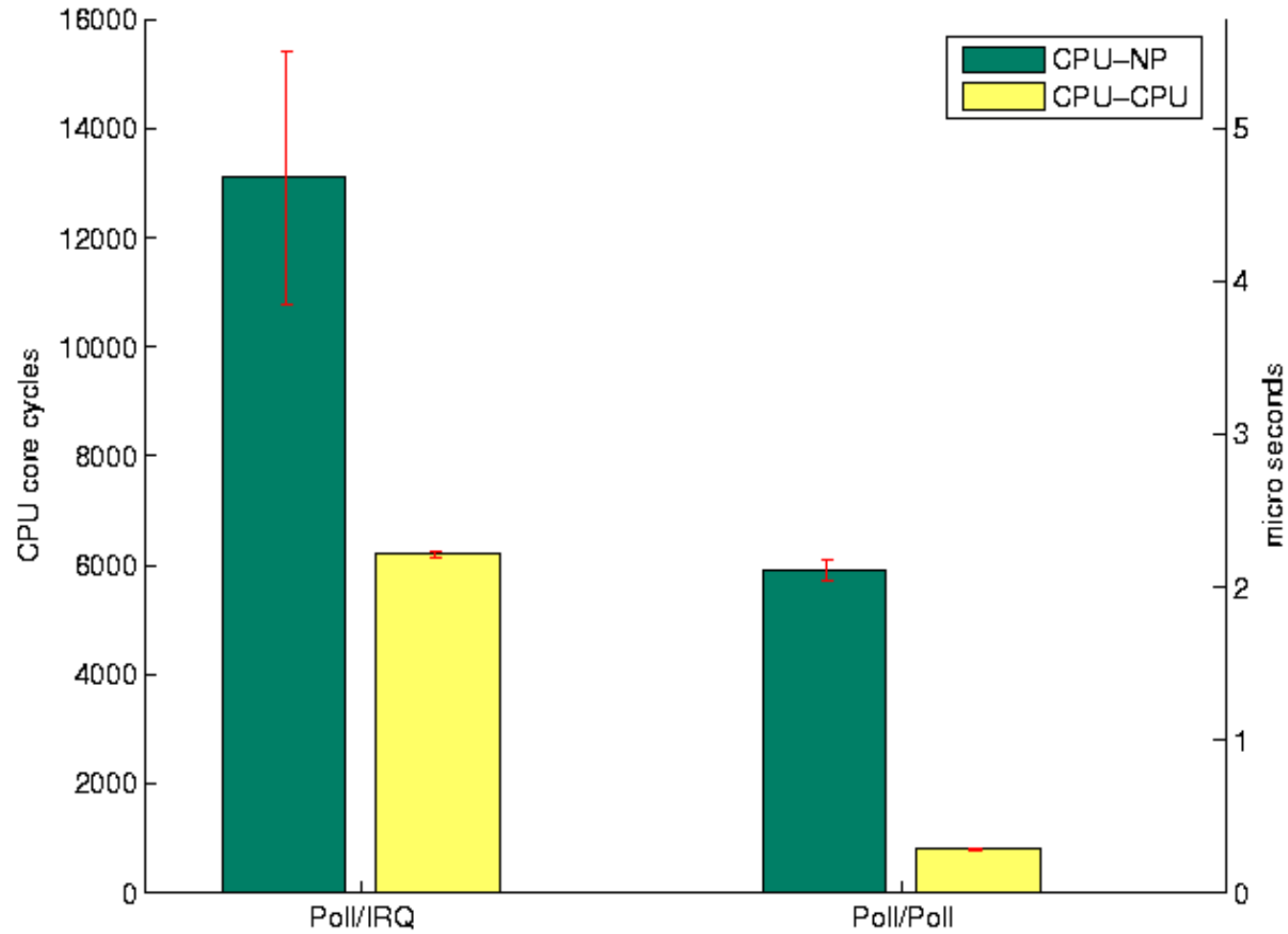
# Micro-Benchmark – Host Latency (SV-NIC)

Num VIFS	Interrupt Virtualization Cost
1	1.99uS
8	3.24uS
32	11.57uS

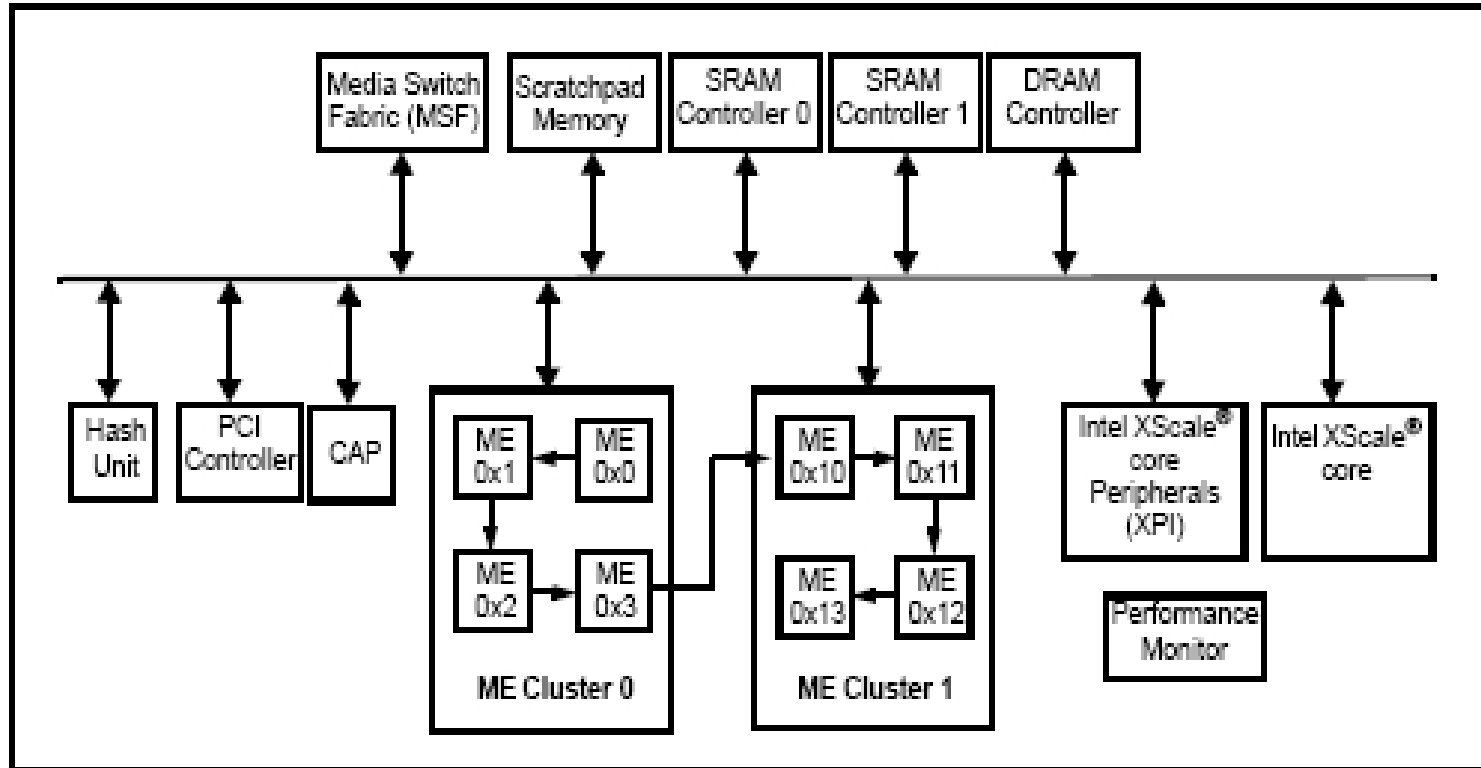
# Virtual Interrupt Identifier



# Core-Core Communication Latency



# IXP2400 Network Processor



Source: IXP2400 H/W Ref. Manual

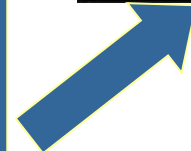
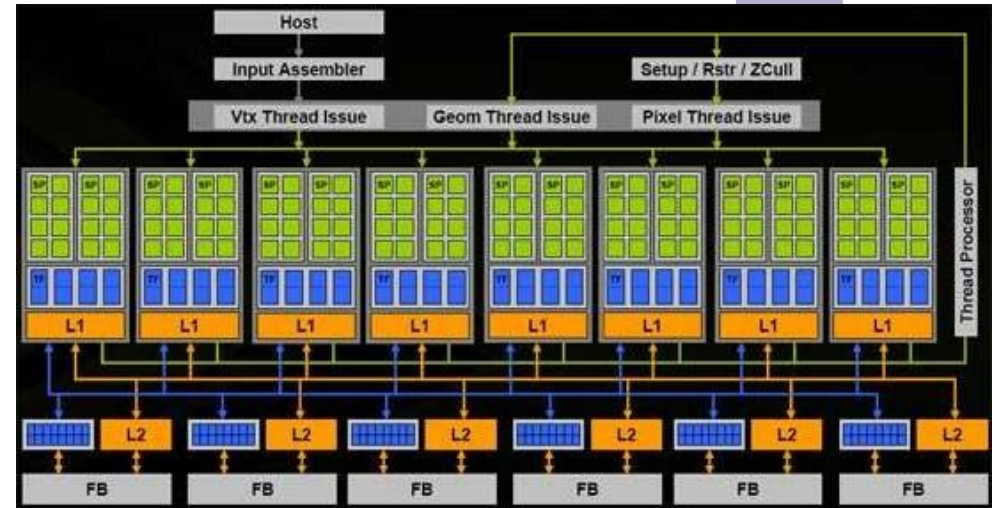
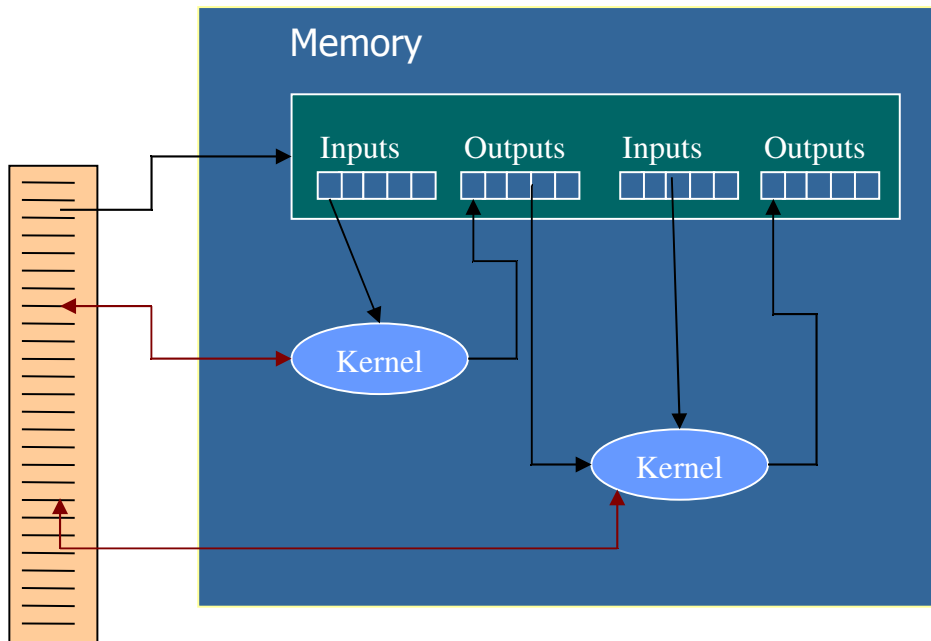
- Utilize a Core in *Appropriate State*
  - Better cache utilization
  - Reduced or no cost for processor state changes
  - Specialized processing
- SV-NIC + Sidecore
  - Limitations of Interrupt Identifier
    - May result in redundant signaling
  - Signal virtualization via dedicated host core
    - No interrupts from device - Polling

# V-Core Issues

- Some accelerators (NVIDIA CUDA) do not support resource sharing
- An accelerators consist of both a device and driver
- Some accelerators do not support sufficiently powerful, programmable back-ends, e.g., some FPGA devices

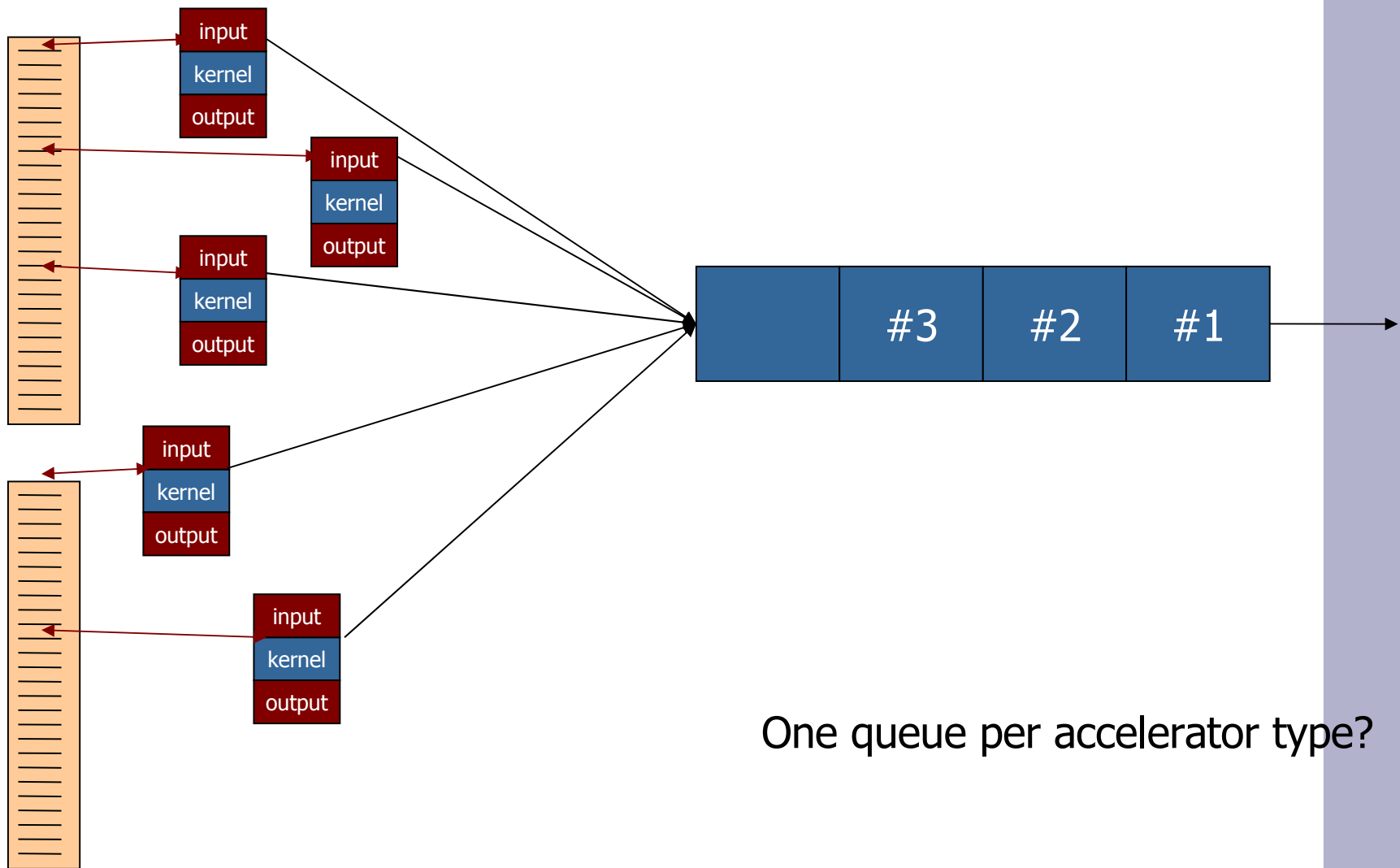
- VAAL Front-End
  - Interface to devices visible at the application layer
  - Enables transmission of executable from the application to the accelerators for execution, e.g., thread binary
- VAAL Back-End
  - Functions needed to virtualize devices, such as memory isolation and kernel to hardware mapping
  - May execute on the device or within an accelerator driver domain depending on strength of the device

# Example: NVIDIA CUDA



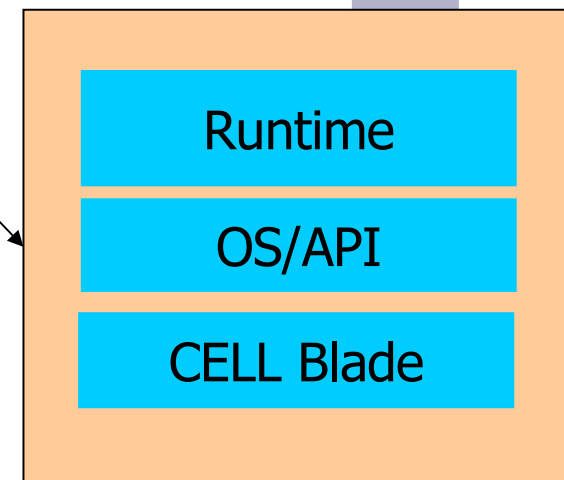
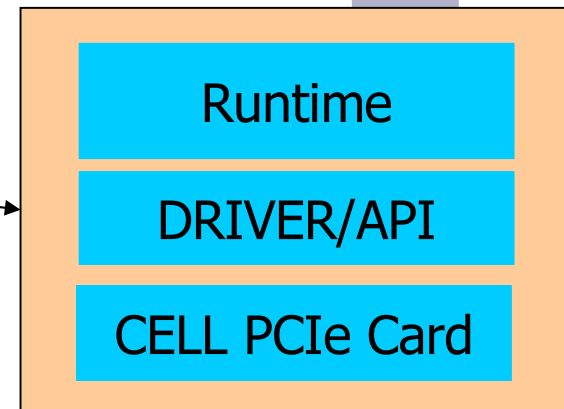
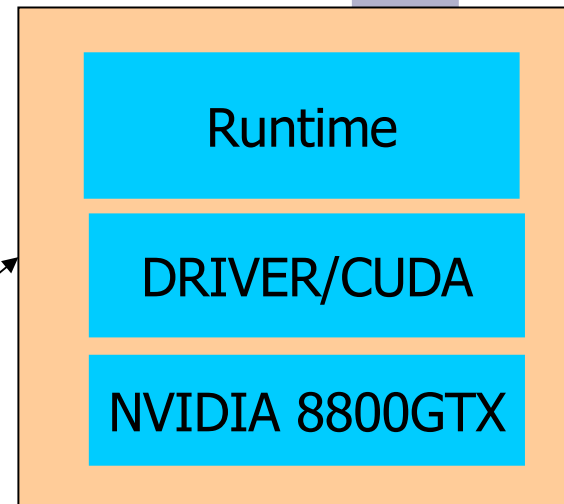
- Describe kernels by function, inputs, and outputs
- Treat kernels as schedulable entities
  - Draw upon insights from OOO superscalar execution

# Run-Time Scheduler: Work Queue

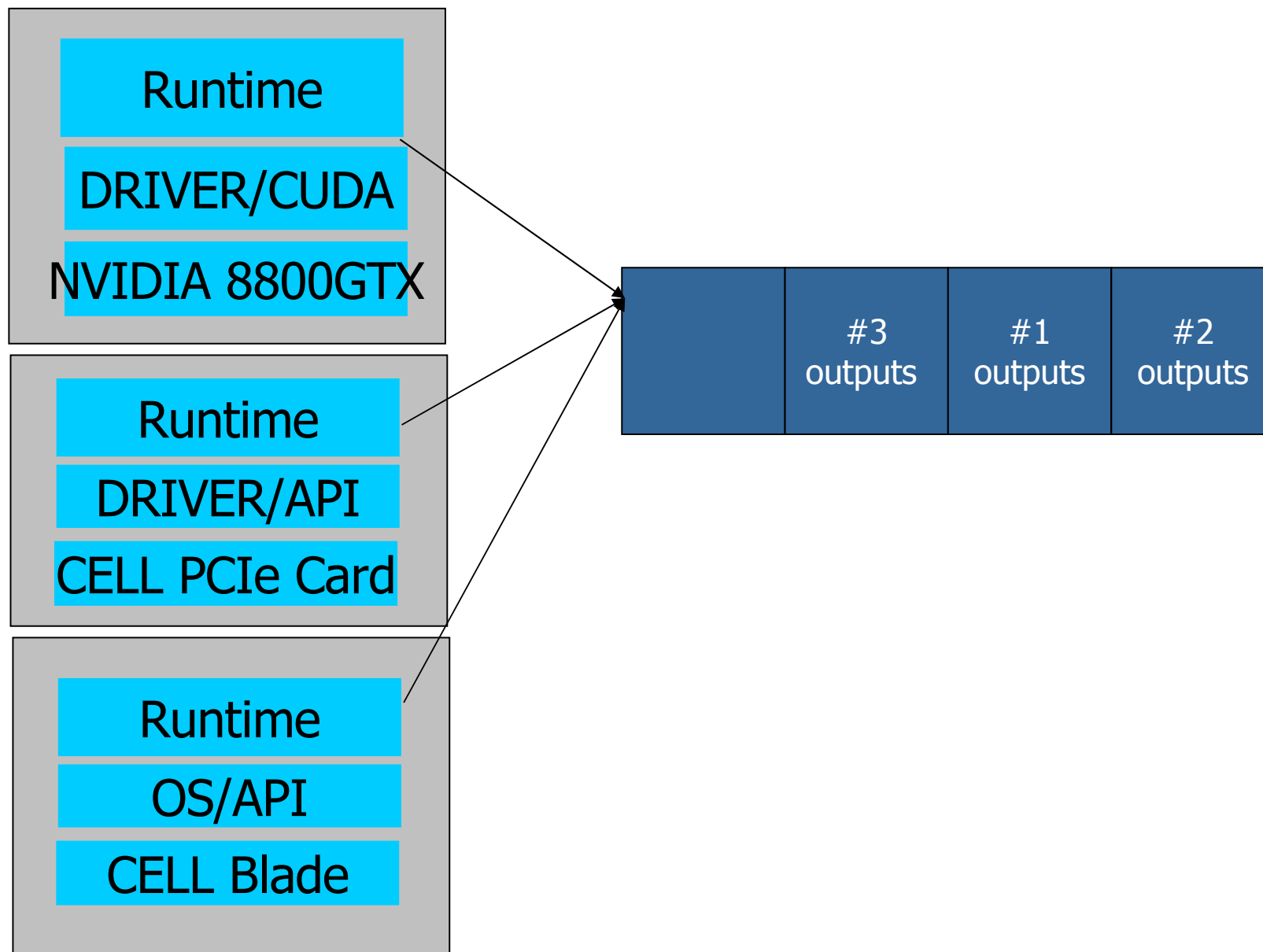


One queue per accelerator type?

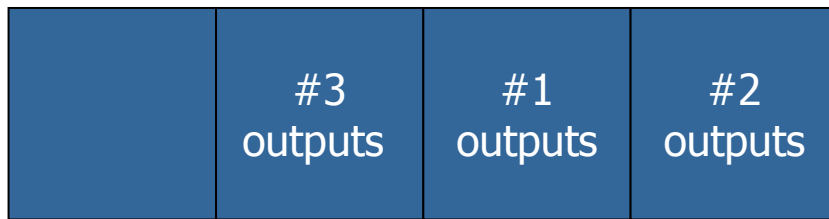
# Accelerator Controllers Pull Kernels From the Work Queue



# Upon Completion, Results Are Pushed Into A Completion Queue



# Reorder Results to Avoid Hazards



Forward outputs from the completion queue to inputs in the work queue as needed

# Applications Register Kernels With Accelerator Controllers At Runtime

