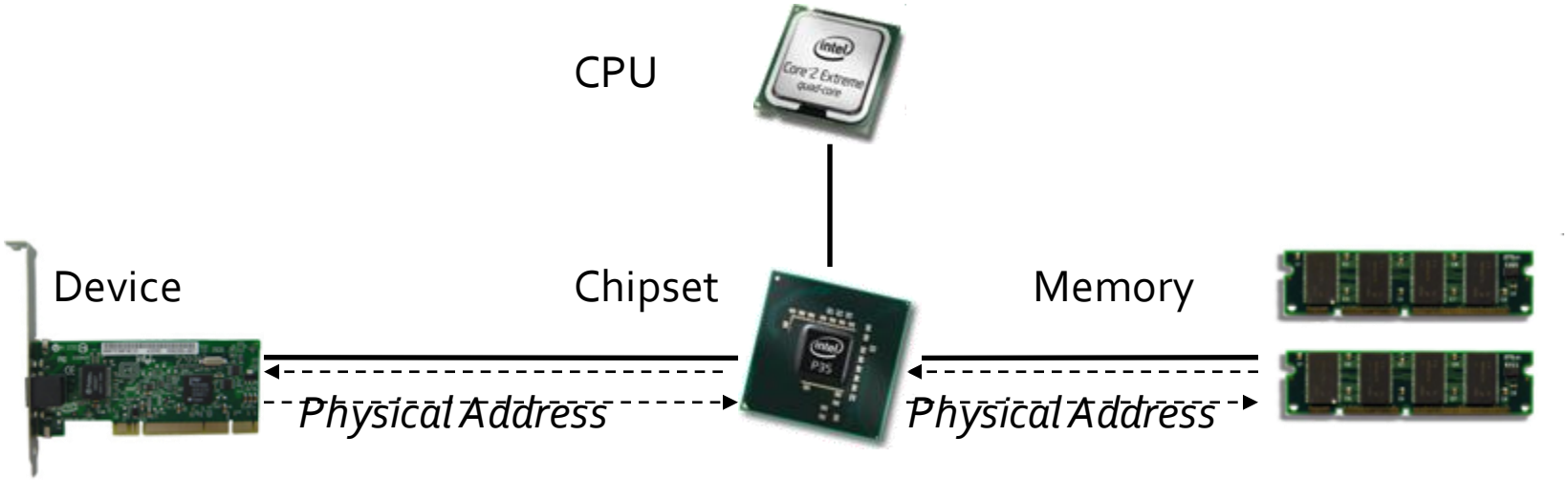
vIOMMU: Efficient IOMMU Emulation

Nadav Amit, Muli Ben-Yehuda, Dan Tsafrir and Assaf Schuster

Technion
IBM Research - Haifa

DMA

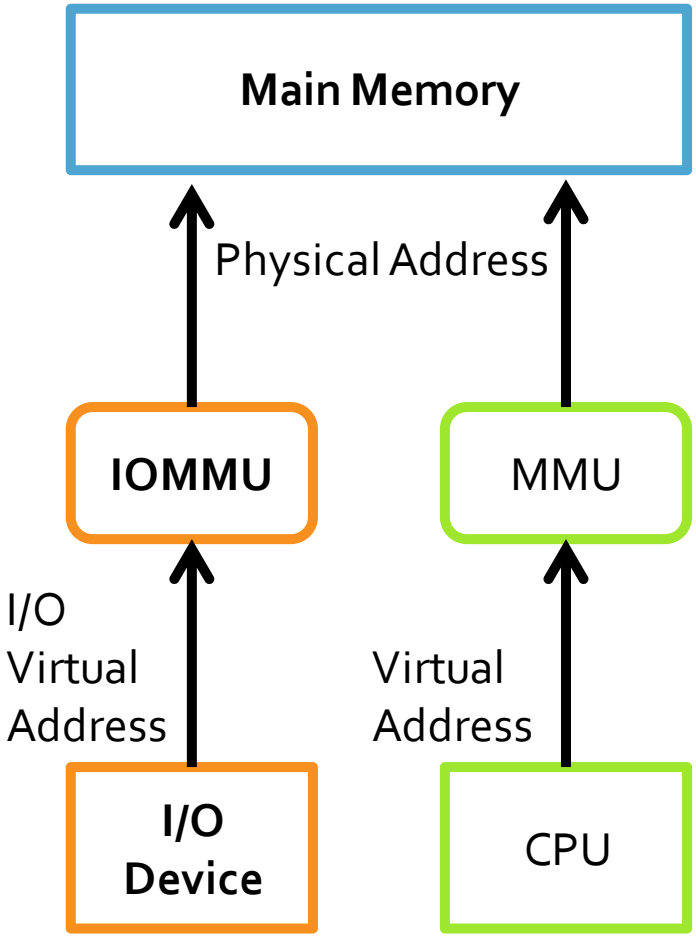
- *DMA* (Direct Memory Access) goal:
  Access system memory independently of the CPU



- Legacy DMA presents several challenges:
  - Cannot be used natively in virtualization
  - Lack of memory protection from device accesses
  - (x86:) Legacy devices cannot access the entire memory

x86 IOMMU

- I/O Memory Management Unit (IOMMU):
  - Indirection level between the address used for DMA and the physical address
  - Similar to MMU, yet translation is performed in the device context
  - Devices are split to protection domains
  - A device can access memory addresses present in its domain



I/O Virtualization

- I/O virtualization through *direct device assignment*:
  - Guest virtual machine interacts with the I/O device directly
  - Unmodified guest OS
  - Best performance
- IOMMU is required for direct device-assignment
  - **Protection** is required to isolate guests (*inter-guest protection*)
  - In virtualization memory accesses of the guest are **redirected** by the hypervisor:
    Guest Physical Address → Host Physical Address
  - Currently, hypervisors use ***direct map*** scheme:
    Mapping GPA→HPA in the IOMMU

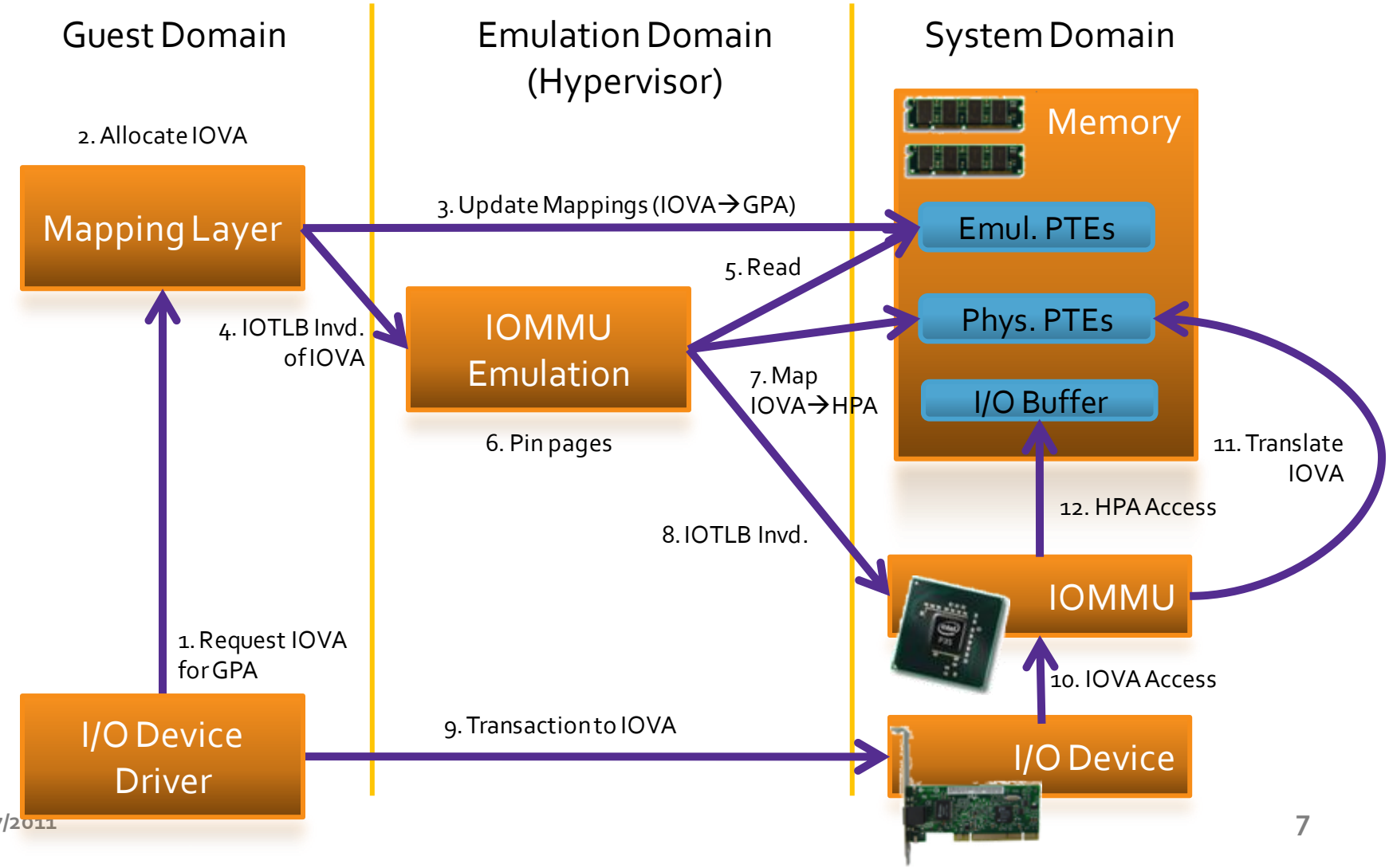
Device-Assignment Shortcomings

- **No memory-overcommitment**
  Memory is the main limiting factor for server consolidation
- **No intra-guest protection**
  Guest is vulnerable to faulty device-drivers and malicious I/O devices that are assigned to it
- **No redirection of DMA**
  Guest cannot use legacy 32-bit I/O devices, nested virtualization or other custom mappings
- These shortcomings will be addressed using IOMMU emulation

IOMMU Emulation

- Use a single physical IOMMU to emulate multiple IOMMUs (for multiple guests)
- The I/O device is still directly assigned to the guest
- First evaluated x86 IOMMU emulation
- The focus of this work is performing IOMMU emulation efficiently

IOMMU Emulation Flow



IOMMU Emulation Uses

- **Memory-overcommitment**

  Pin/unpin during map/unmap of I/O buffers

- **Intra-guest protection & DMA redirection**

  Physical IOMMU is programmed according to emulated

- **Inter-guest protection is not compromised**

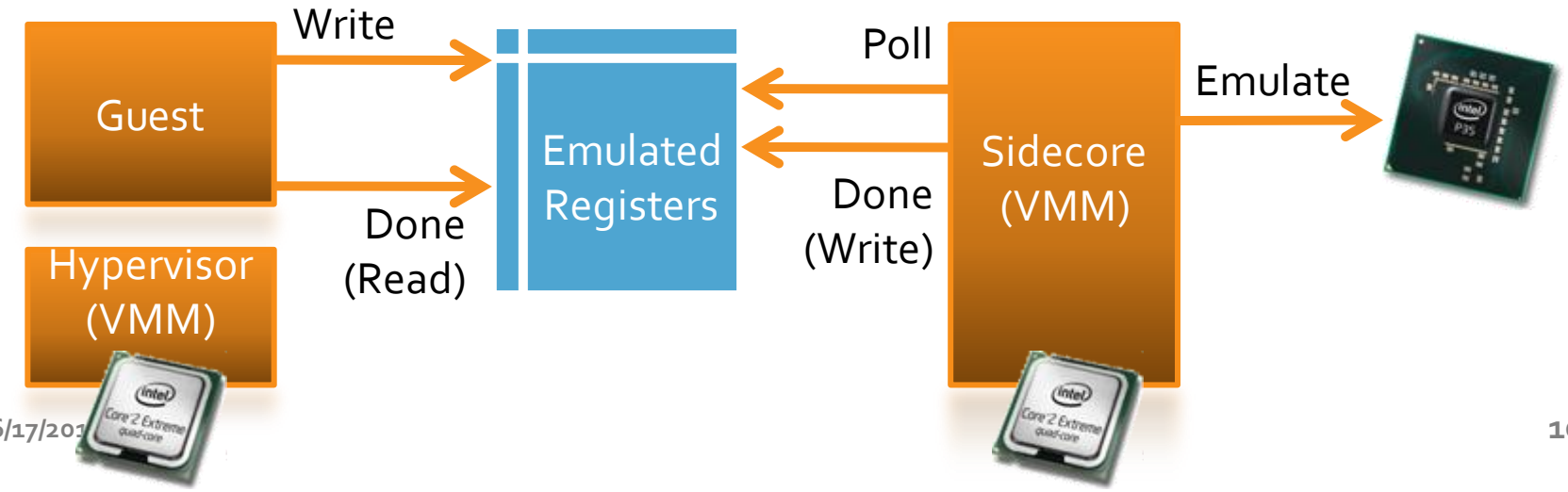
  A page is pinned to physical memory the entire time it is mapped in the physical IOMMU

IOMMU Emulation Enhancements

- **Each mapping change results in VM-exit**
  - Expensive operation (direct cost and indirect cost)
  - Diminishes the advantages of device-assignment
- **Map and unmap in the IOMMU are lengthy and frequent operations**
  - Require allocation of IOVA, MMIO transactions, etc.
  - Even in native-mode can reduce throughput by 57%
- **Our solutions:**
  - Sidecore emulation
  - IOMMU mapping-layer enhancements

Sidecore Emulation

- Additional core (sidecore) can be used to avoid VM-transitions through paravirtualization [Kumar'07, Liu'09]
- Similarly, sidecore emulation can be performed



Sidecore Emulation

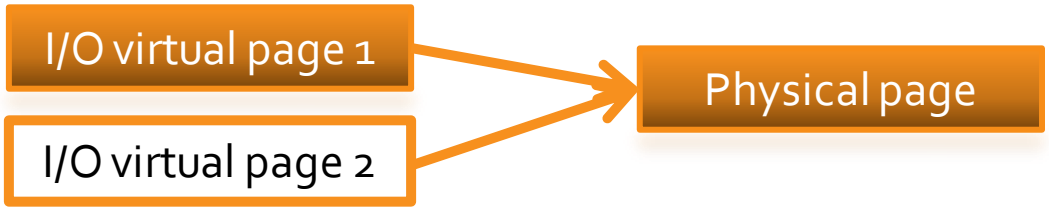
- Required device properties:
  - Synchronous register write protocol
  - A single register holds only read-only or write fields
  - Loose response time requirements
  - Explicit update of memory-resident data structures
- x86 IOMMU:
  - Intel VT-d has these properties
  - AMD IOMMU does not

Mapping Optimizations

- Revised mapping layers are usable and efficient in native-mode
- **IOTLB invalidation scheme**
  - IOTLB invalidation is an expensive operation
  - Linux already coalesces and defers invalidations
  - The paper discusses asynchronous invalidations
- **Mappings reuse**

Mappings Reuse

- **Common IOVA allocation method**
  - *Single-use* – Mapping of I/O virtual address for each given I/O buffer physical address
- **Mappings Reuse (Existing):**



  - *Shared* – If there is an existing mapping of physical, use it [Willmann'08]
  - *Persistent, On-demand* – Never unmap or defer unmappings until a quota of stale mappings is reached [Yassour '10]

Optimistic Teardown

- Optimistic teardown is a mappings reuse scheme
- Deferring unmappings until a quota of stale mappings is reached or time-limit elapses
- Useful since DMA accesses demonstrate both temporal and spatial locality
- Since stale mappings are always reclaimed, it delivers better protection than persistent
- Protection relaxation impact is similar to that of deferring IOTLB invalidations (Linux default)

Evaluation Environment

- **Hardware**
  - *Intel VT-d*
  - IBM x3550 M2 server
  - Intel Xeon X5570 CPUs running at 2.93GHz
  - Emulex 10Gbps NIC
- **Hypervisor**
  - Linux 2.6.35
  - KVM – Kernel-based Virtualization Machine
  - A single-core guest shown
- **Benchmarks**
  - Microbenchmarks: NetPerf TCP Stream and UDP-RR (Latency)
  - Macrobenchmarks: Apache and MySQL
  - The following benchmarks use a single core

Map/Unmap Duration

	CPU Cycles		
	SW/HW Interaction	Logic	Total
Bare-metal	2316	4593	6909
Trap & Emulate	30645	4324	34969
Sidecore	7321	1904	9225

Average breakdown of (un)mapping a single page using the strict invalidation scheme

- Sidecore emulation is over 3 times faster than trap & emulate
- Logic part of the code is performed faster in emulation!

TCP Throughput

Setting	*Secure* (No IOTLB Batching)	Relaxed (Linux Default; IOTLB Batching)	Optimistic (Patched; IOTLB Batching)
Bare-metal	43%	91%	100%
Trap & Emulate	10%	11%	82%
Sidecore	30%	49%	100%

Measuring the Netperf TCP throughput relatively to the maximum attainable (9.3Gbps)

- Optimistic teardown over 7 times faster than relaxed in guest
- Sidecore busy at most 33% of the time

Apache

Setting	Secure (No IOTLB Batching)	Relaxed (Linux Default; IOTLB Batching)	Optimistic (Patched; IOTLB Batching)
Bare-metal	84%	92%	94%
Trap & Emulate	38%	39%	56%
Sidecore	61%	63%	66%

Measuring Apache throughput (25 concurrent requests); the baseline for normalization is 6828 requests per second

- Maximum attainable throughput in guest – 67%

Conclusions

- Using IOMMU emulation, we can enable memory overcommitment, intra-guest protection and redirection of DMA transactions.
- Sidecore emulation of the IOMMU and mapping-layer optimizations can reduce the mapping overhead significantly with modest protection compromise.
- Future hardware support can deliver even better performance:
  - Multiple levels of IOMMU paging hierarchy can deliver intra-guest protection
  - I/O page faults can enable memory overcommitment even for non-cooperative guests