Virtualization

Clothing the Wolf in Wool

Virtual Machines

- Began in 1960s with IBM and MIT Project MAC
- Also called “open shop” operating systems
- Present user with the view of a bare machine
- Execute most instructions directly, but trap operations that would reveal the fiction, and emulate
- Hot topic until Unix, then architectural support drops
    - Example of M68010

Advantages

- Greater security -- more isolation of tasks
- Ability to manage QoS for tasks
- Easy to profile tasks
- Can run multiple operating systems
- Can test and debug new OS code directly

Disadvantages

- More complex OS/VMM
- Takes more resources than kernel OS
- Requires hardware support
- Can be slow if hardware support is weak

Gerald Popek CACM 1974

Formal Requirements for Virtualizable Third Generation Architectures

Requirements

- Equivalence: Virtual machine must look like bare HW (but smaller)
- Resource control: All resources are virtualized and managed by the VMM
- Efficiency: Has to be nearly as fast as running natively

Instruction Types

- Privileged: Implies supervisor state with special instr.
- Control sensitive: Changes processor mode, or memory map
- Behavior sensitive: Behavior depends on mode or on location (anything that can reveal state of other tasks)
- A VMM requires that the sensitive instructions be a subset of the privileged instructions

Recursive Virtualizability

- If there is proper HW support, a VM can run recursively within itself
- Allows nested operating systems, layers of control
- Rarely supported

Paravirtualization

- Can fake with JIT or direct binary translation (DBT)
- XEN approach when no HW virtualization mode
- Can’t be completely hidden from adversary

Sensitive, Non-privileged x86 Instructions

- SGDT – Store Global Descriptor Table register
- SIDT – Store Interrupt Descriptor Table register
- SLDT – Store Local Descriptor Table register
- SMSW – Store Machine Status Word
- PUSHF(D) – Push EFLAGS register on stack (16 and 32-bit versions)
- POPF(D) – Pop EFLAGS register from stack, with some privilege levels (16 and 32-bit versions)

Sensitive, Non-privileged x86 Instructions

- LAR – Load access rights into GP register
- LSL – Load segment address limit into GP register
- VERR – Verify if code/data segment is readable based on current protection level
- VERW – Verify is code/data segment is writeable based on current protection level
- POP – Can raise general protection exception depending on target register and protection level
- PUSH – Can push protection status onto the stack

Sensitive, Non-privileged x86 Instructions

- CALL – Can call to same or a different privilege level, saving return info
- JMP – Like CALL, but without saving return info
- INT n – Like a far CALL to a different level, but also pushes EFLAGS on stack
- RET – Can return between privilege levels
- STR – Store segment selector (including privilege bits)
- MOV – Can be used to load or store control register set

Virtualization Extensions

- VT-x introduced 2005 by Intel
- AMD-V introduced 2006 by AMD
- AMD Rapid Virtualization Indexing (nested page tables) adds hardware to MMU
- Intel adds Extended Page Tables
- VT-D provides virtualization of directed I/O, trapping DMA, etc.
- Typically not enabled in BIOS
- VirtualBox claims to be faster without VT-x

Itanium Example

- Trap to a higher level, setting violation status
- Can only return to a lower level resetting status
- Can’t forward the violation to a guest OS to handle

Beyond the ISA

- Areas that are hard to virtualize
  - Complex virtual memory
  - I/O and network devices
  - Graphics, GPUs
  - Multithreading
  - Cache coherence
  - Multicore
  - TLB, branch predictor -- clever optimizations can backfire when they are virtual

Dall ISCA16

ARM Virtualization: Performance and Architectural Implications

Hypervisor Types

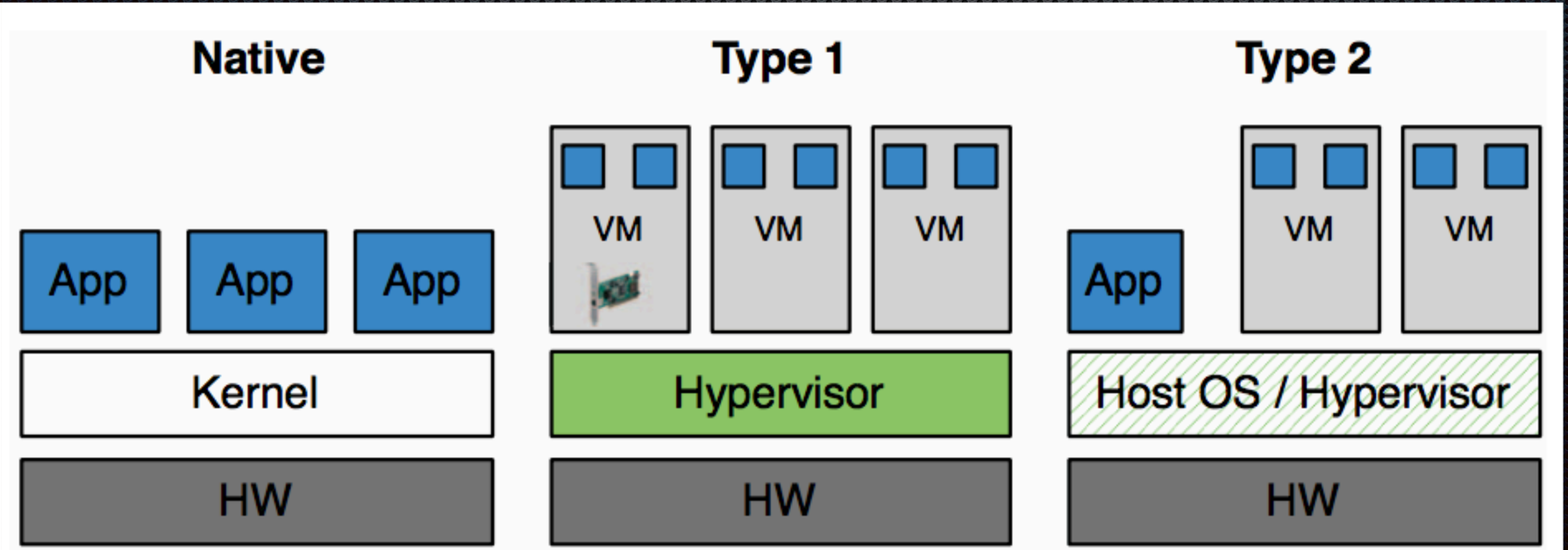


Figure 1: Hypervisor Design

KVM

- Type 2 hypervisor
- Integrates with Linux
- Reuses existing device drivers
- Higher cost to switch from VM to Host
- Paravirtualizes I/O devices with Virtio
  - Has direct access to hardware resources

Xen

- Type 1 hypervisor
- Runs under multiple host operating systems
- Has a privileged VM, Dom0, that runs an existing OS and funnels I/O through its drivers (except for some basic operations), giving greater isolation
- Guest VMs known as DomU
- Paravirtualizes I/O devices with Xen PV

ARM Virtualization Extension

- Additional privilege level: EL2, for hypervisor
- EL2 has three types of addresses: Virtual, Intermediate, Physical
- Interrupt controller supports virtual interrupting of VMs
- Physical interrupts go to EL2
- VM can set timer, but it traps to EL2, which fires a virtual timer interrupt to the VM
- State saving is flexible compared to x86 VMCS table

Xen on ARM

- Runs hypervisor in EL2
- VM kernel in EL1, VM users in EL0

Xen on ARM

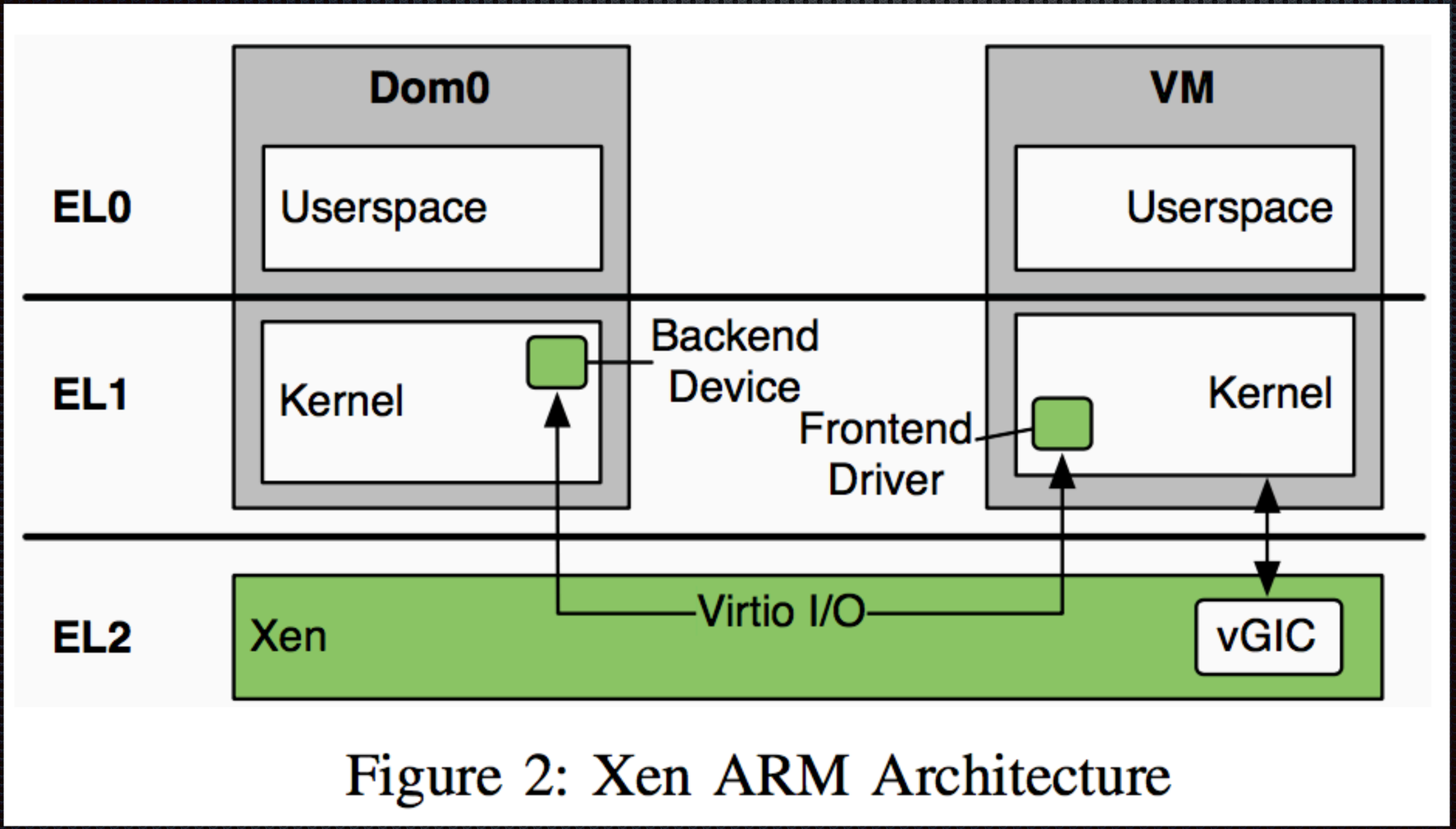


Figure 2: Xen ARM Architecture

KVM on ARM

- Host OS can’t run in EL2
- Runs host and VMs in EL1
- Puts minimal hypervisor functionality in EL2
  - Reverts to EL1 host/hypervisor for rest
- Enables virtualization only when in EL2, since host needs full hardware access
- Hypervisor does more context switch state saving because host, hypervisor, and VMs share EL1

Microbenchmarks

	ARM		x86	
Microbenchmark	**KVM**	**Xen**	**KVM**	**Xen**
Hypercall	6,500	376	1,300	1,228
Interrupt Controller Trap	7,370	1,356	2,384	1,734
Virtual IPI	11,557	5,978	5,230	5,562
Virtual IRQ Completion	71	71	1,556	1,464
VM Switch	10,387	8,799	4,812	10,534
I/O Latency Out	6,024	16,491	560	11,262
I/O Latency In	13,872	15,650	18,923	10,050

Table II: Microbenchmark Measurements (cycle counts)

Hypervisor calls are expensive for KVM because of the EL2 to EL1 transition. I/O is expensive on Xen because of having to send it via the hypervisor to Dom0

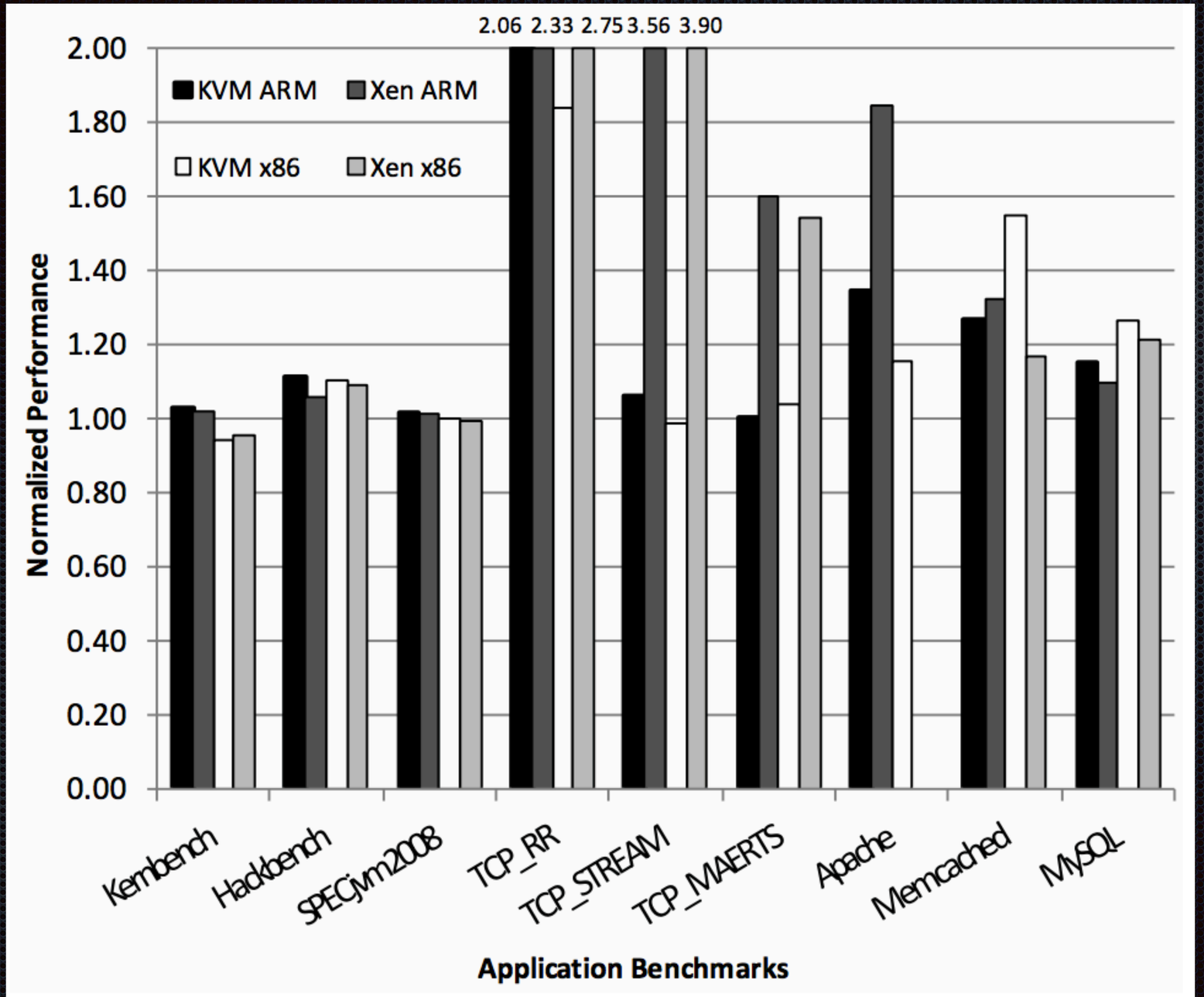


Figure 4: Application Benchmark Performance

Netperf Detail

	Native	KVM	Xen
Trans/s	23,911	11,591	10,253
Time/trans (μs)	41.8	86.3	97.5
Overhead (μs)	-	44.5	55.7
send to recv (μs)	29.7	29.8	33.9
recv to send (μs)	14.5	53.0	64.6
recv to VM recv (μs)	-	21.1	25.9
VM recv to VM send (μs)	-	16.9	17.4
VM send to send (μs)	-	15.0	21.4

Table V: Netperf TCP_RR Analysis on ARM

Packet processing cost is high, especially for Xen, due to a lack of zero-copy I/O (done for more isolation)

Virtualization Host Extension

- Change EL2 so host OS can run in it
- Provides equivalent registers (especially page table)
- Instructions to access EL1 registers from EL2
- Make EL2 page tables compatible with EL1

VHE for Type 2, not Type 1

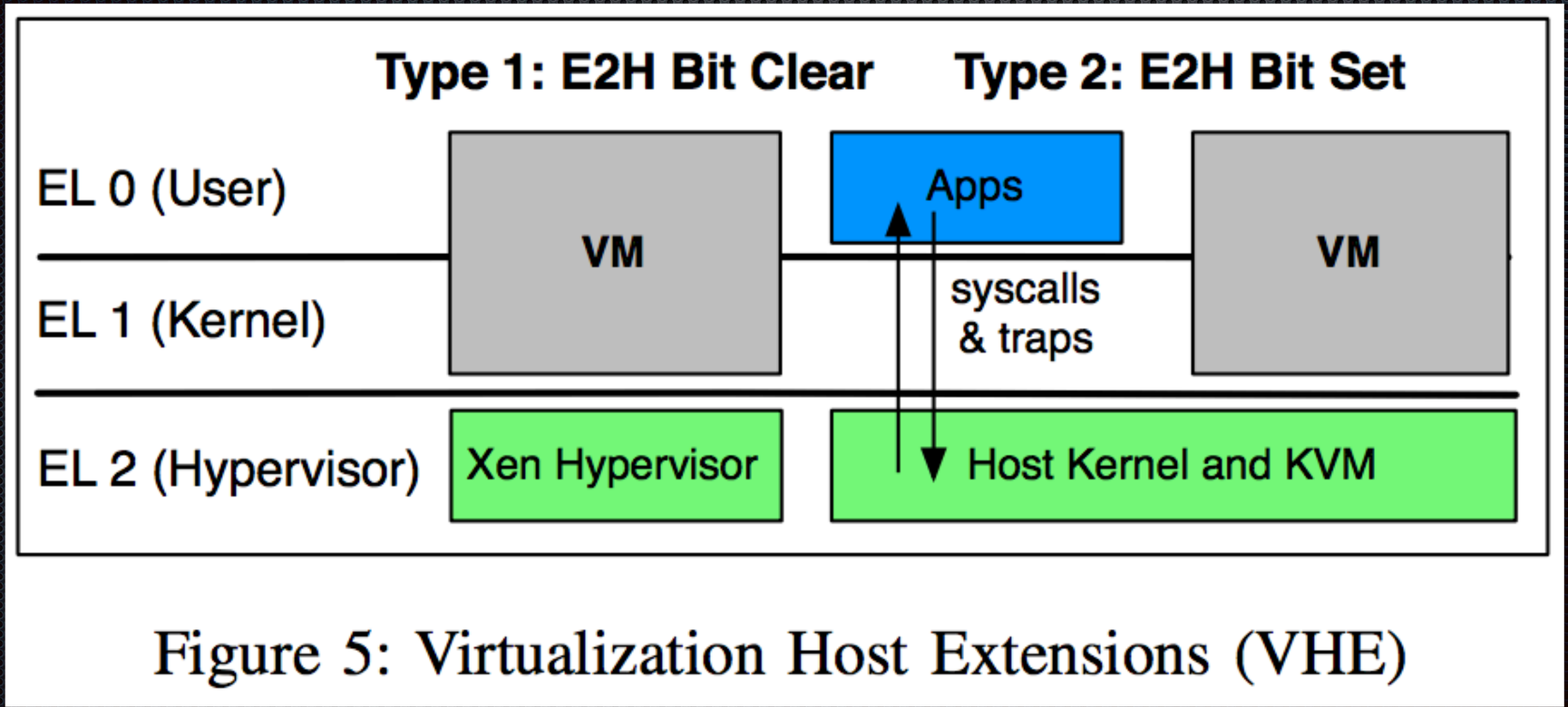


Figure 5: Virtualization Host Extensions (VHE)

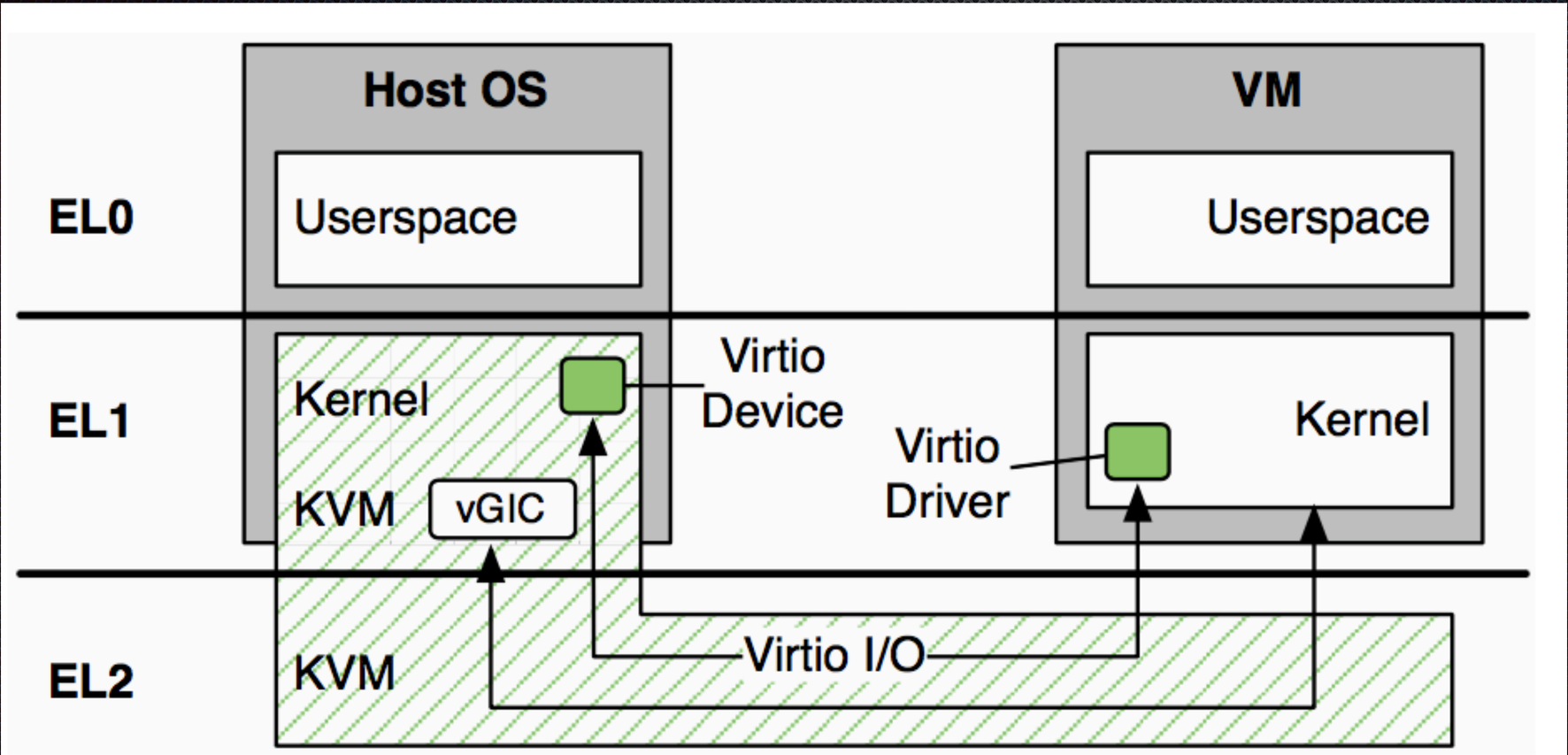


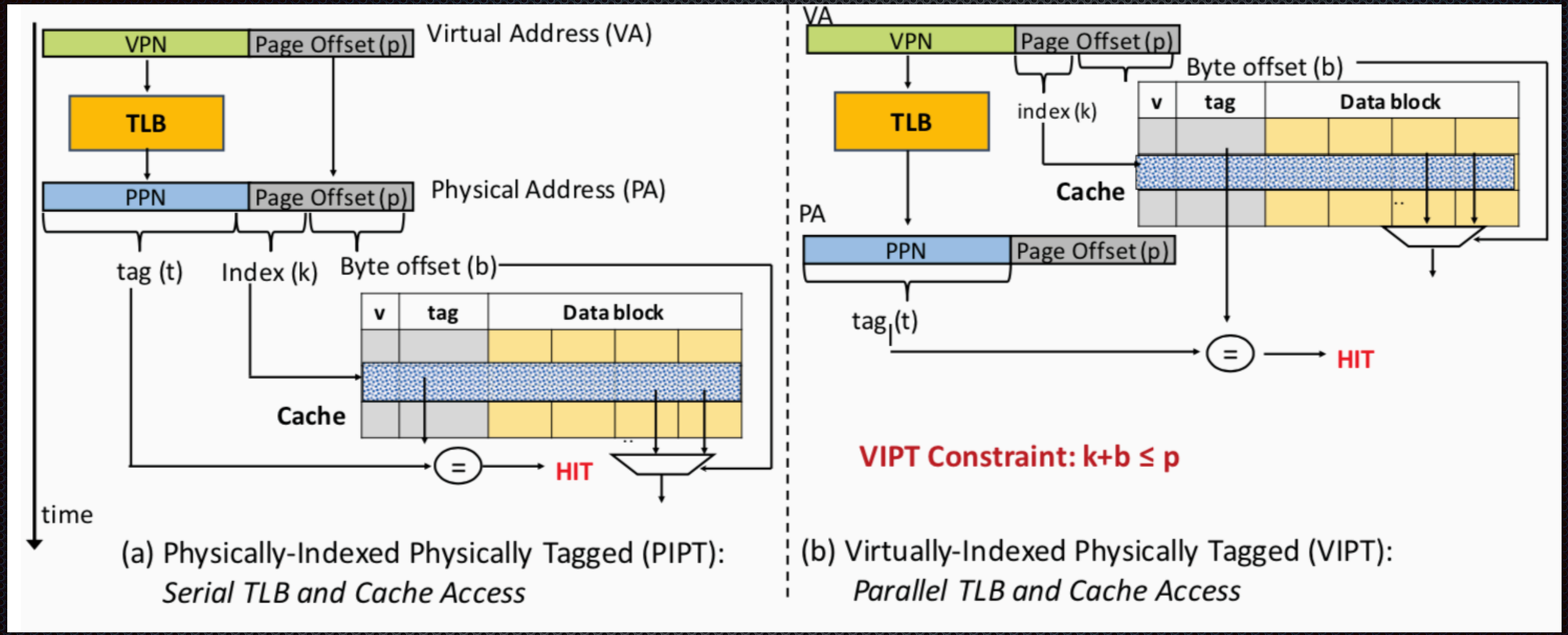
Figure 3: KVM ARM Architecture

Discussion

Parasar ISCA 18

SEESAW: Using Superpages to Improve VIPT Caches

Caches and Virtual Memory



- Another option for TLB placement — beside L1D

Previously, and a Different Way

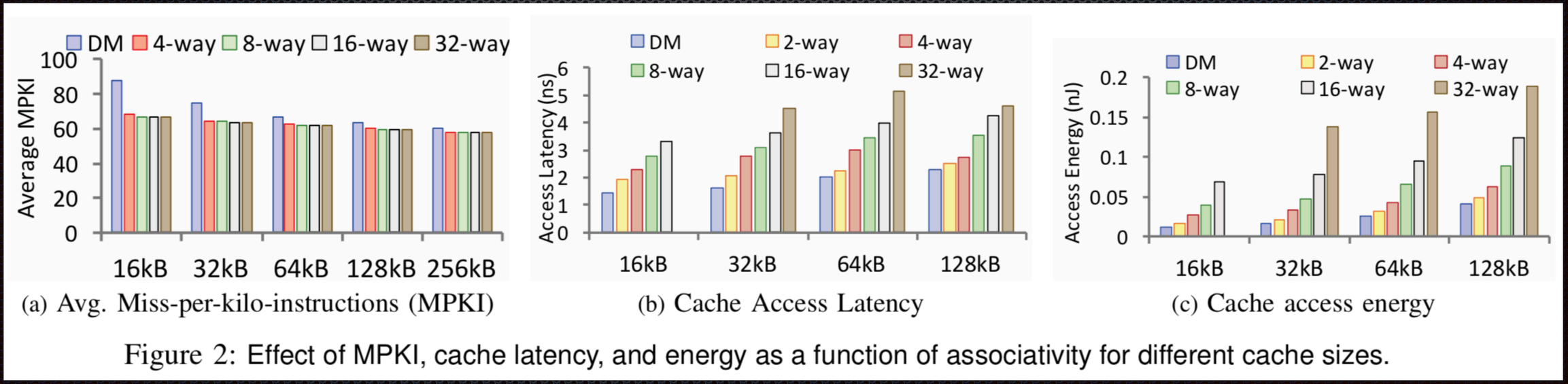
- We looked at TLB above L1 (Physically indexed, physically tagged)
- Or TLB below L1 (Virtually indexed, virtually tagged)
- Can also use just a portion of the page offset as an index (virtual, but within a page, so aligned) and access the TLB with the VPN to get a translation that is compared to the physical address stored in the indexed line's tag
- As cache lines grow long, less of the page offset is available as an index because more bits are taken for the line offset
- If the index is limited, caches can only grow via higher associativity

Obvious Question

- Why not just increase the page size, e.g., to 64KB?
  - That only gives 4 more bits of index (comparable to 16 ways of associativity)
  - Bigger would result in excessive waste of memory (RAM and disk) space
  - It would require a major OS rewrite
- We already have superpages in the hardware and OS

More Ways Can be Worse

- As we know, latency and power increase
- Beyond 4 ways, hit rate isn't significantly better, so performance loss isn't hidden



(a) Avg. Miss-per-kilo-instructions (MPKI) (b) Cache Access Latency (c) Cache access energy

Figure 2: Effect of MPKI, cache latency, and energy as a function of associativity for different cache sizes.

Proposed System

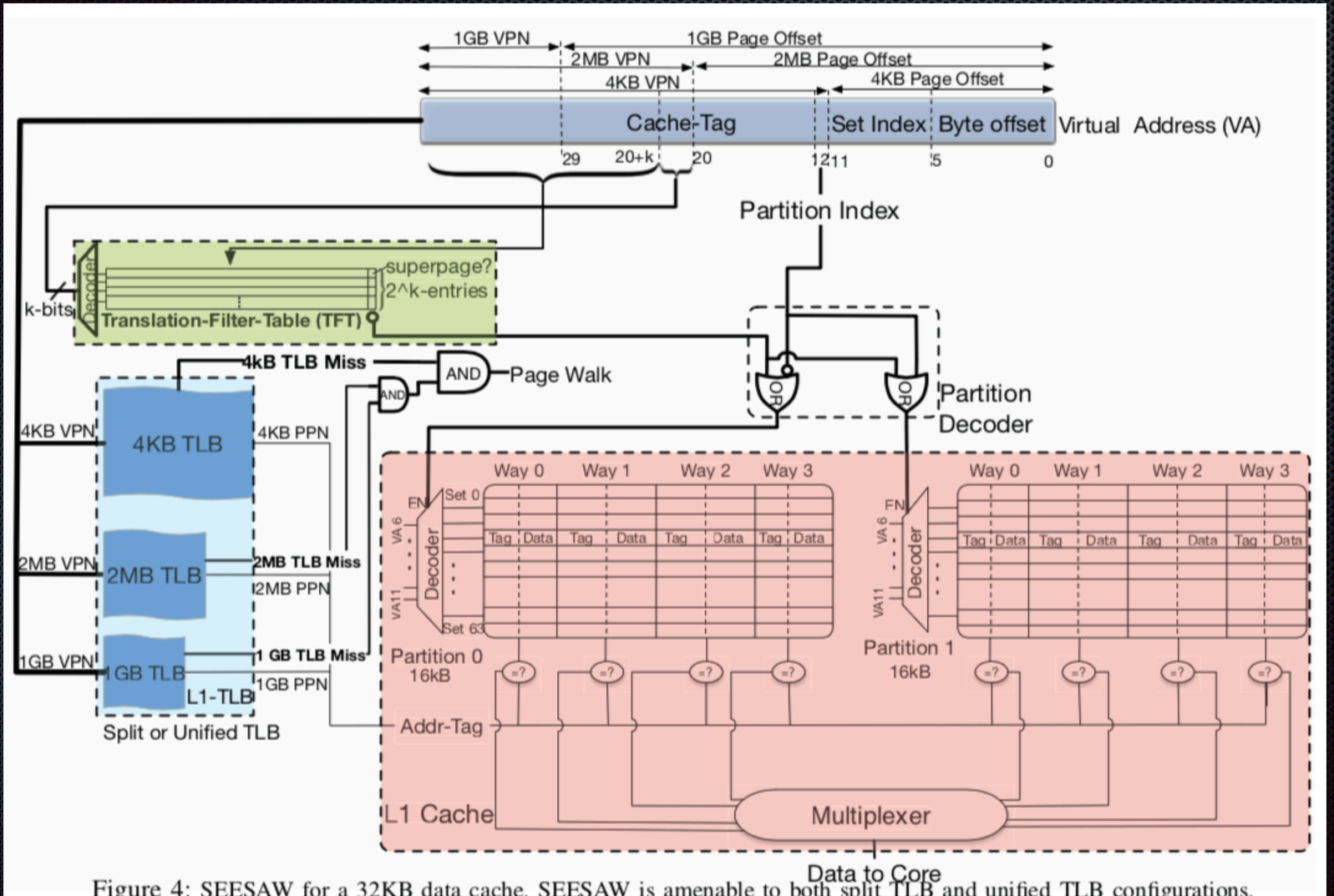


Figure 4: SEESAW for a 32KB data cache. SEESAW is amenable to both split TLB and unified TLB configurations.

Translation Filter Table

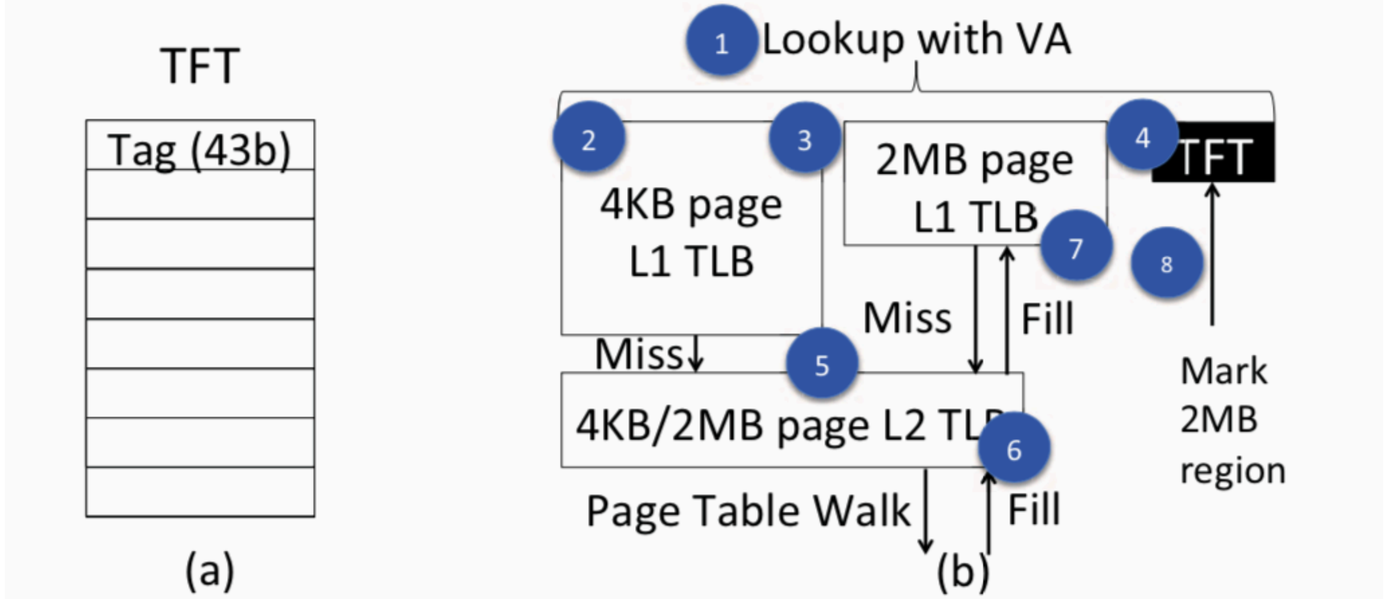


Figure 5: (a) The TFT consists of a list of 2MB virtual address regions that backed by 2MB superpages; and (b) the TFT is looked up in parallel with the L1 TLBs and is updated (or marked) whenever an entry is filled into the 2MB L1 TLB.

Results - 1.33 GHz

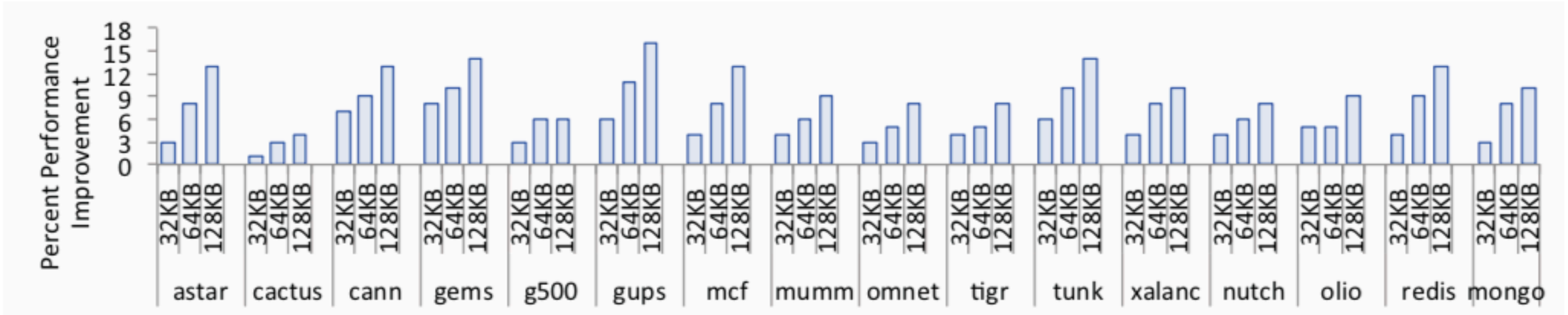


Figure 7: Percent improvement in runtime using SEESAW on an OoO processor versus baseline VIPT. (Freq = 1.33GHz, L1 cache: 32KB to 128KB.

Results — Faster Clock

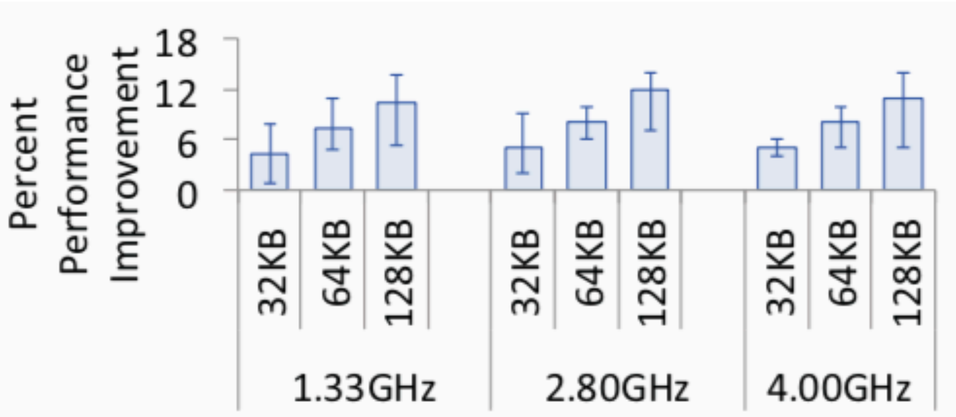


Figure 8: Percent improvement in runtime using SEESAW on an out-of-order processor versus baseline VIPT. We show the average, minimum, and maximum improvements across all workloads for all cache sizes, across 1.33GHz, 2.80GHz, and 4.00GHz operating frequency.

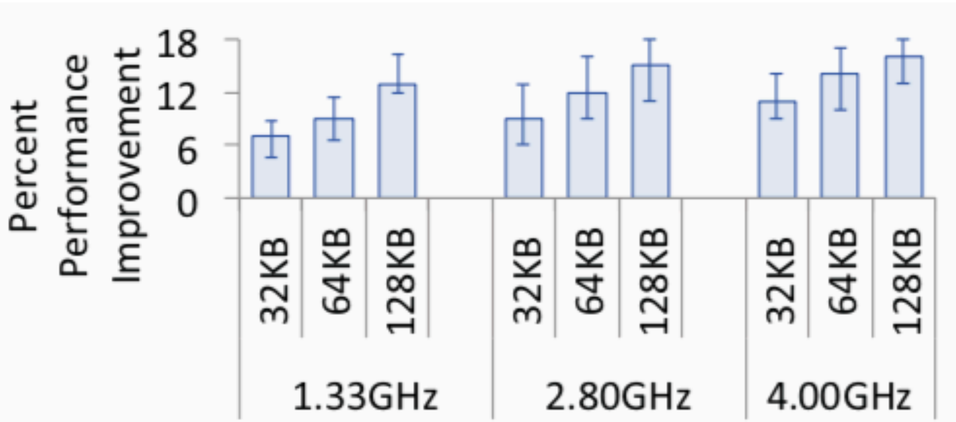


Figure 9: Runtime improvement using SEESAW on an in-order processor versus baseline VIPT. We show the average, minimum, and maximum improvements across all workloads for all cache sizes, across 1.33GHz, 2.80GHz, and 4.00GHz operating frequency. Benefits are higher on an in-order versus out-of-order processor.

Results — Energy

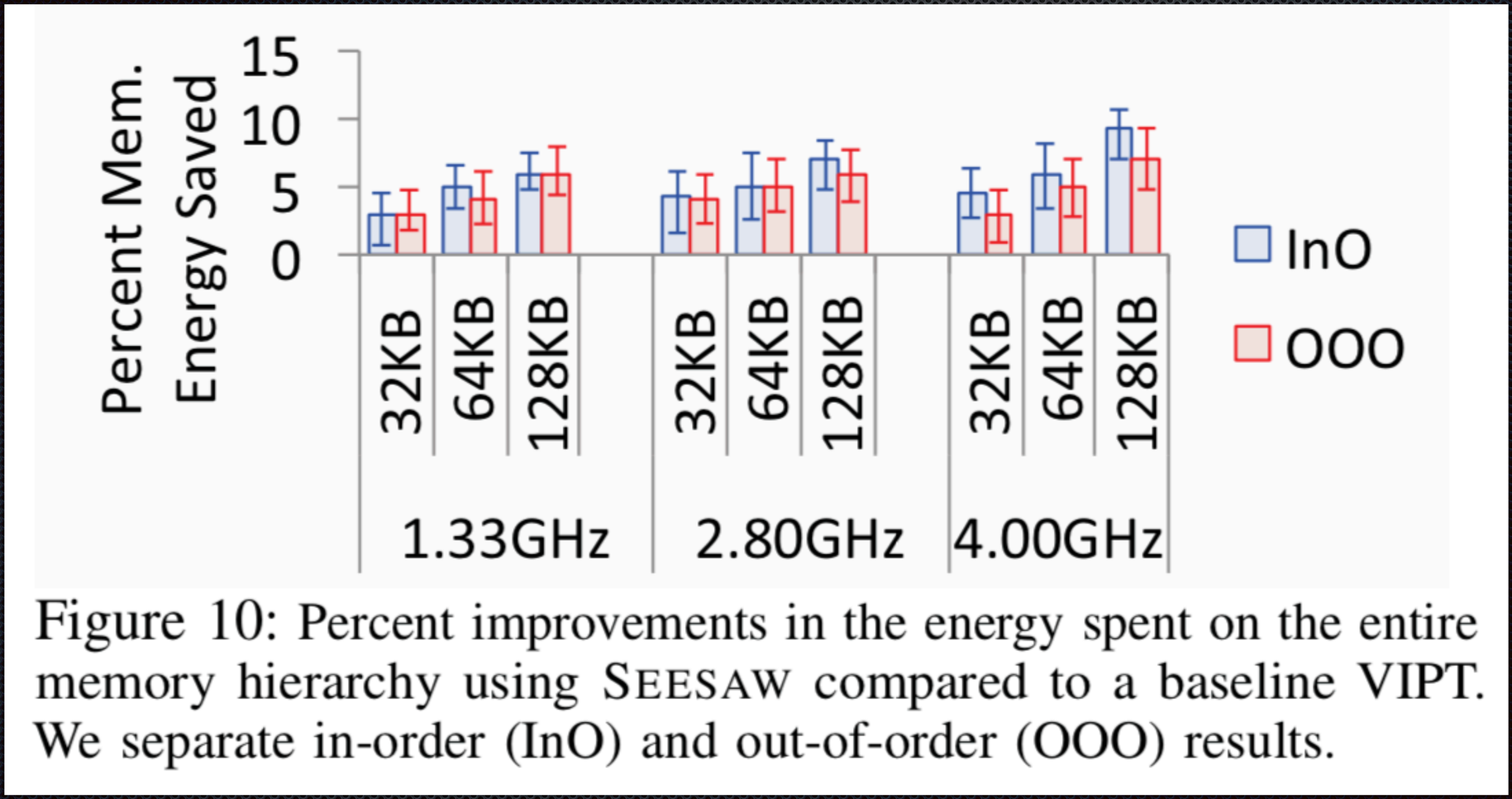


Figure 10: Percent improvements in the energy spent on the entire memory hierarchy using SEESAW compared to a baseline VIPT. We separate in-order (InO) and out-of-order (OOO) results.

Discussion