X86-64: MMU Virtualization With EPT

Hardware and Software Support For Virtualization

Chapter 5

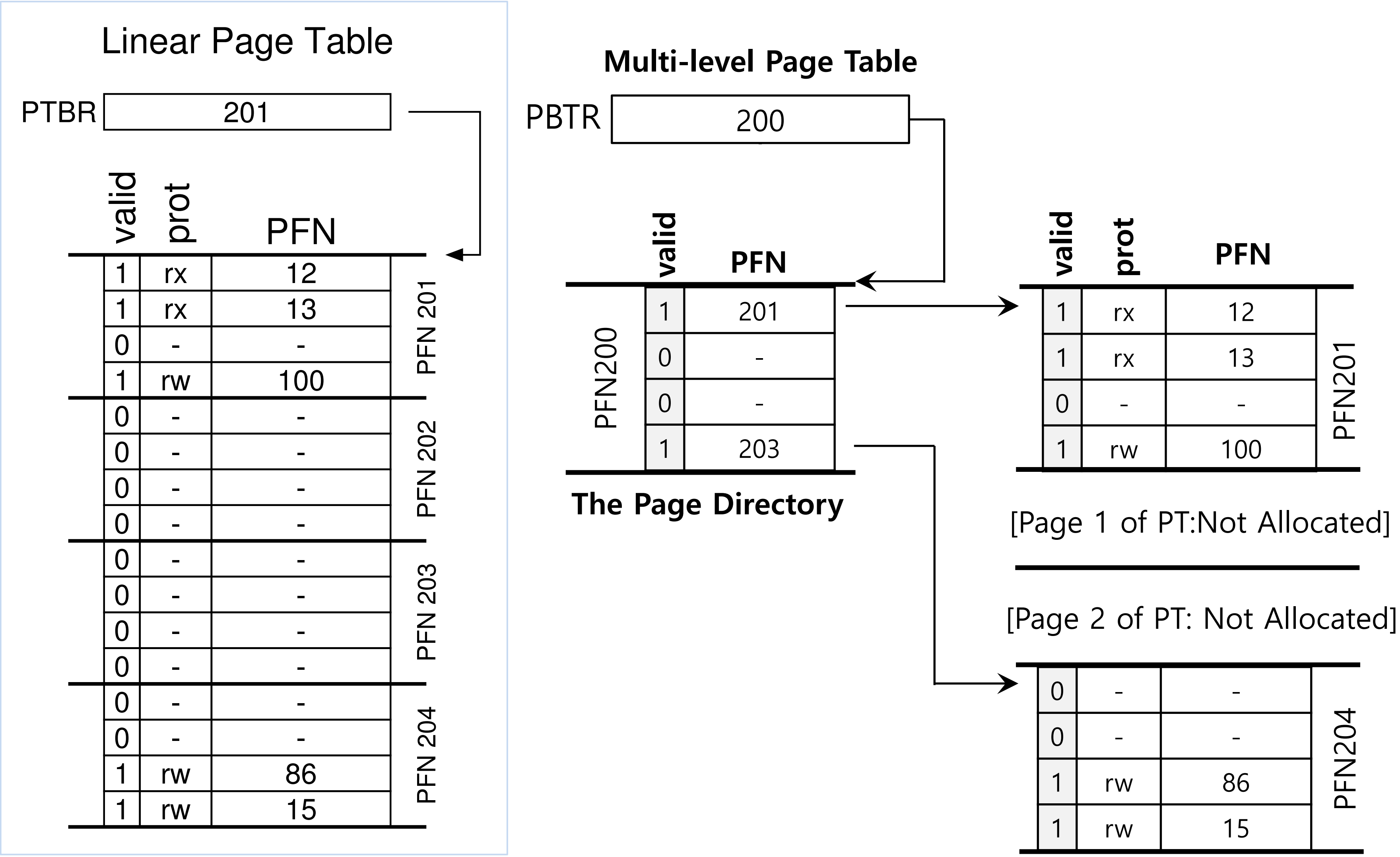
Memory is a Two-dimensional problem

- **Problem**
  - Each VM is provided with the **guest-physical** memory abstraction
  - Hypervisor handles **host-physical** memory (i.e., the physical resource)

- Memory translation is no longer a linear problem: a unidimensional approach (i.e., *linear* page table) **won't suffice**

- Approaches
  - **Pure Software solutions** on HW conceived for two privilege levels (OS) are required (e.g. Shadow Page Tables). Arguably **the most complex** part of the Hypervisor. [Memory paravirtualization restricted by guest-OS/VMM]
  - **Hardware support** for bi-dimensional page tables, (a.k.a. Extended Page Tables or Nested Page Tables)

Review: Multi Level Page Tables I



Linear Page Table

PTBR 201

valid	prot	PFN	
1	rx	12	PFN 201
1	rx	13	
0	-	-	
1	rw	100	
0	-	-	PFN 202
0	-	-	
0	-	-	
0	-	-	
0	-	-	PFN 203
0	-	-	
0	-	-	
0	-	-	
0	-	-	PFN 204
0	-	-	
1	rw	86	
1	rw	15	

Multi-level Page Table

PBTR 200

	valid	PFN
PFN200	1	201
	0	-
	0	-
	1	203

The Page Directory

valid	prot	PFN	
1	rx	12	PFN201
1	rx	13	
0	-	-	
1	rw	100	

[Page 1 of PT:Not Allocated]

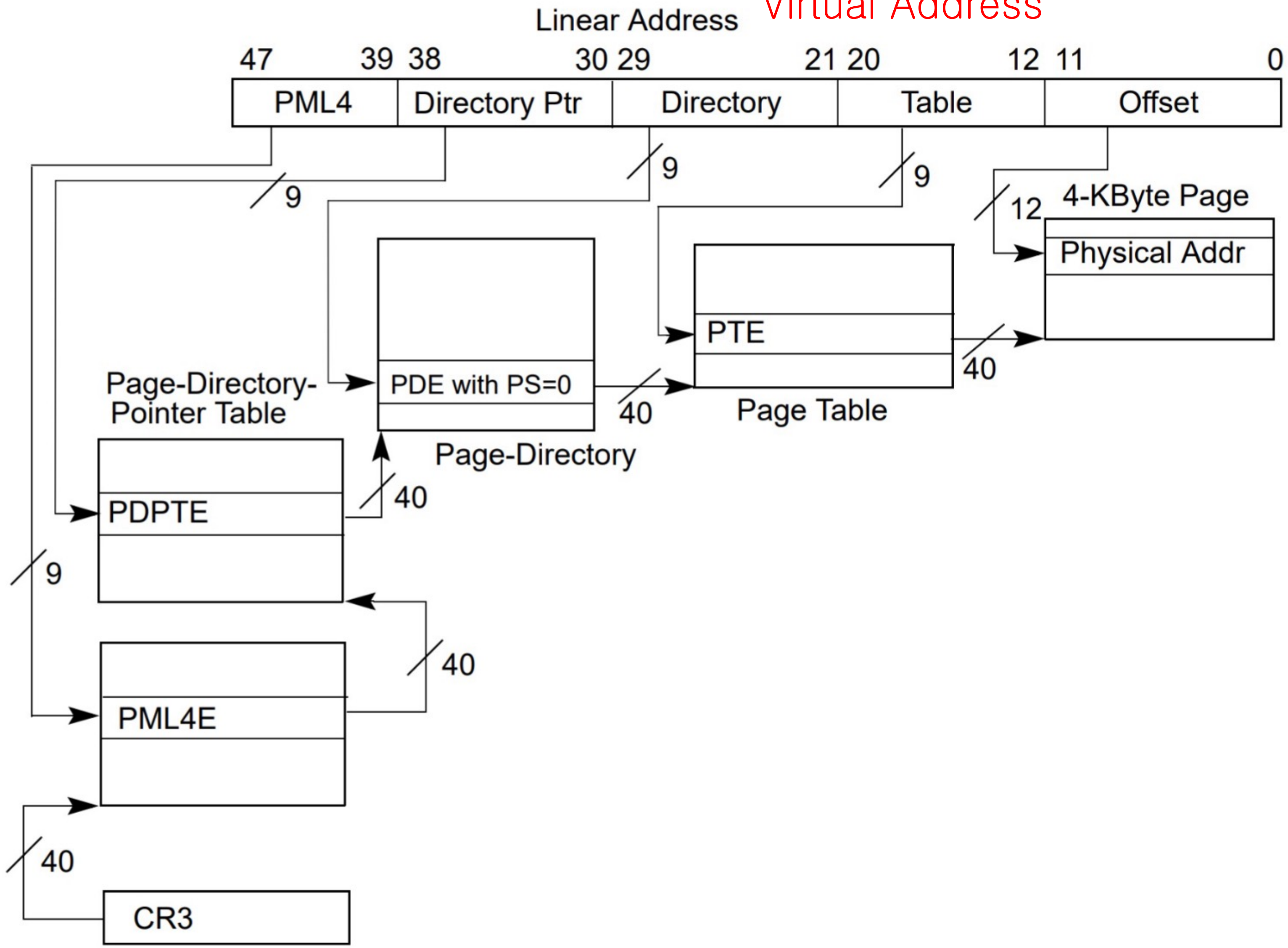
[Page 2 of PT: Not Allocated]

0	-	-	PFN204
0	-	-	
1	rw	86	
1	rw	15	

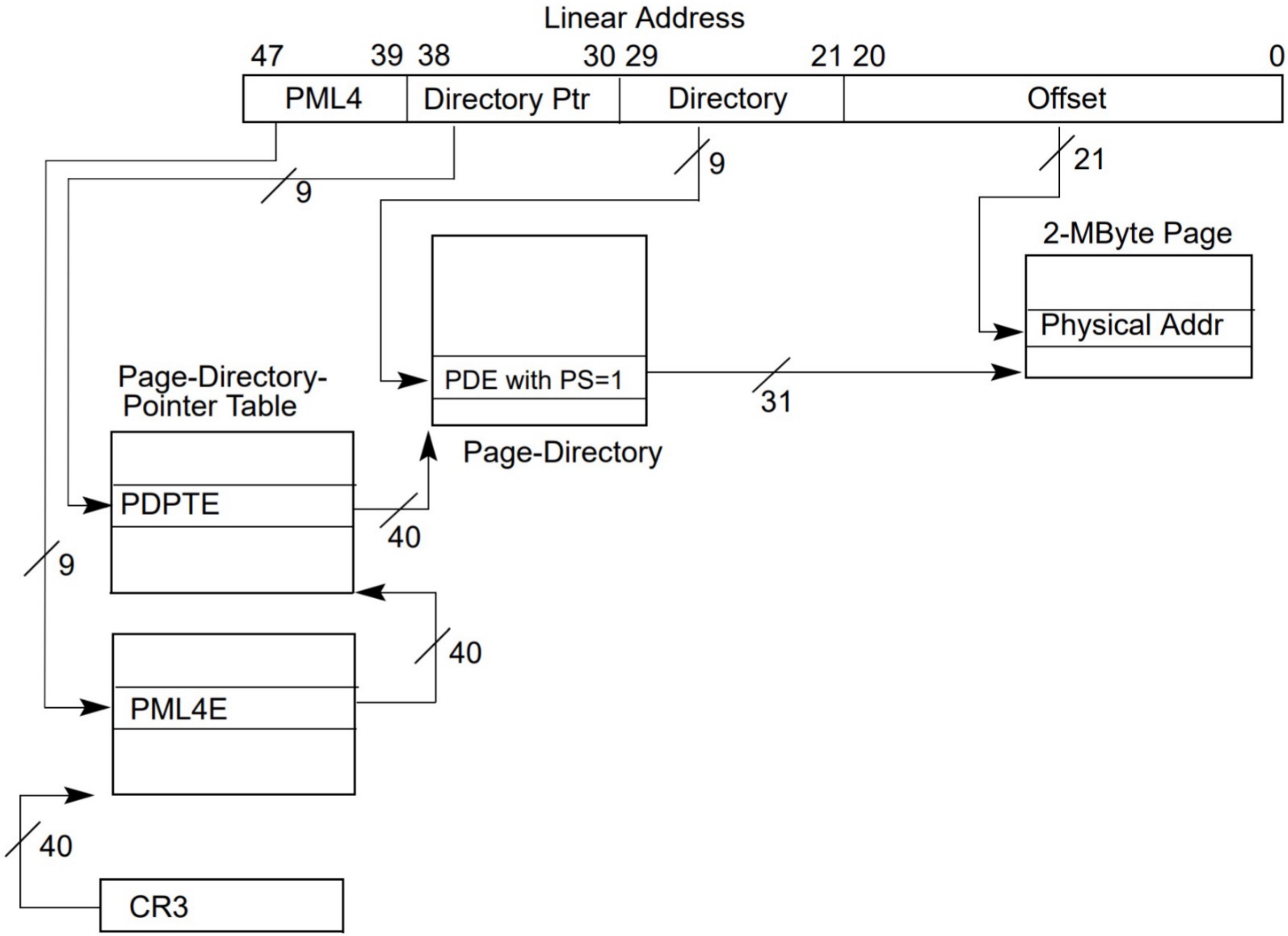
Linear (Left) And Multi-Level (Right) Page Tables

Review: Multilevel Page Tables II (2^{48}B=256TB)

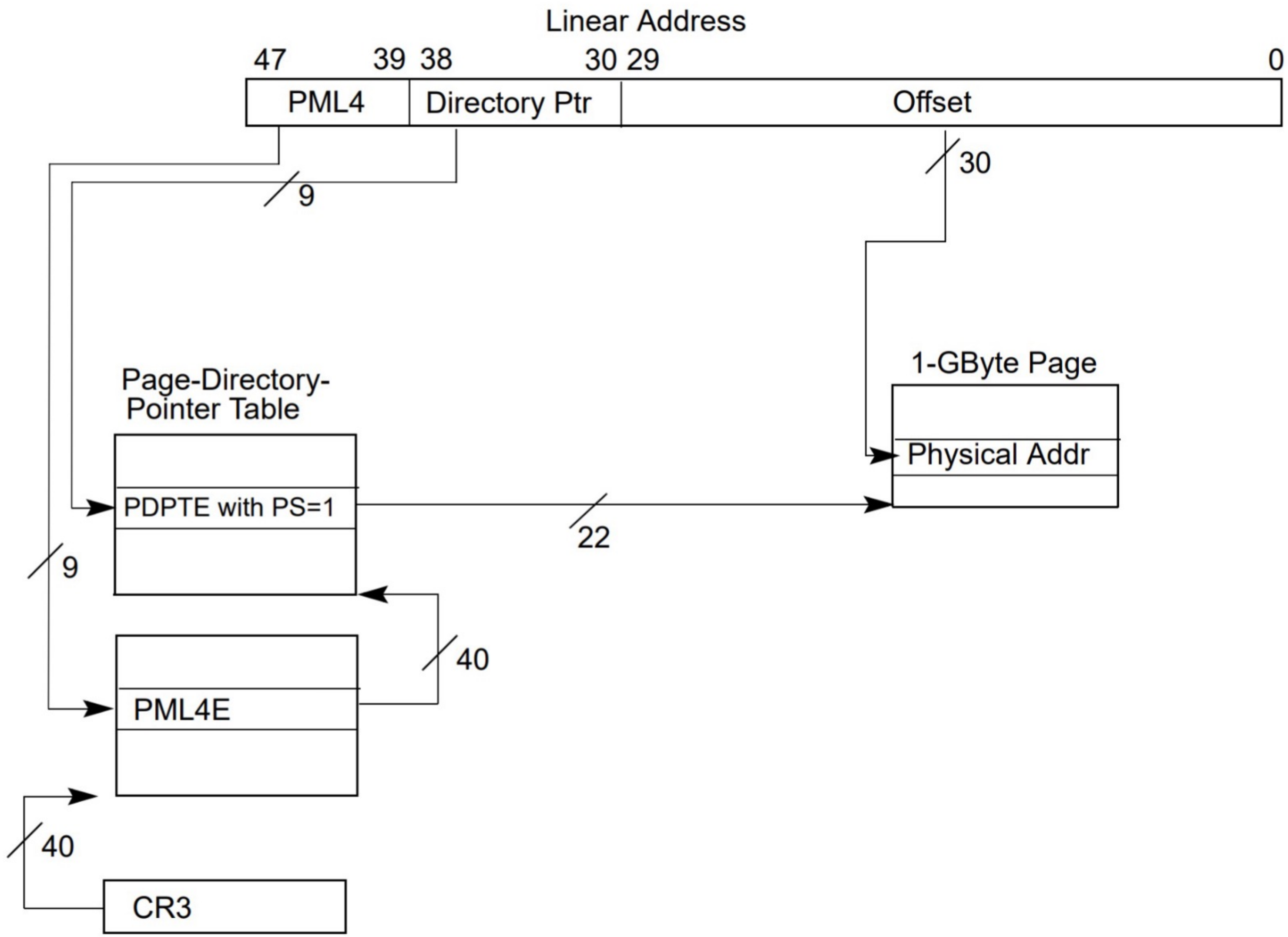
Virtual Address



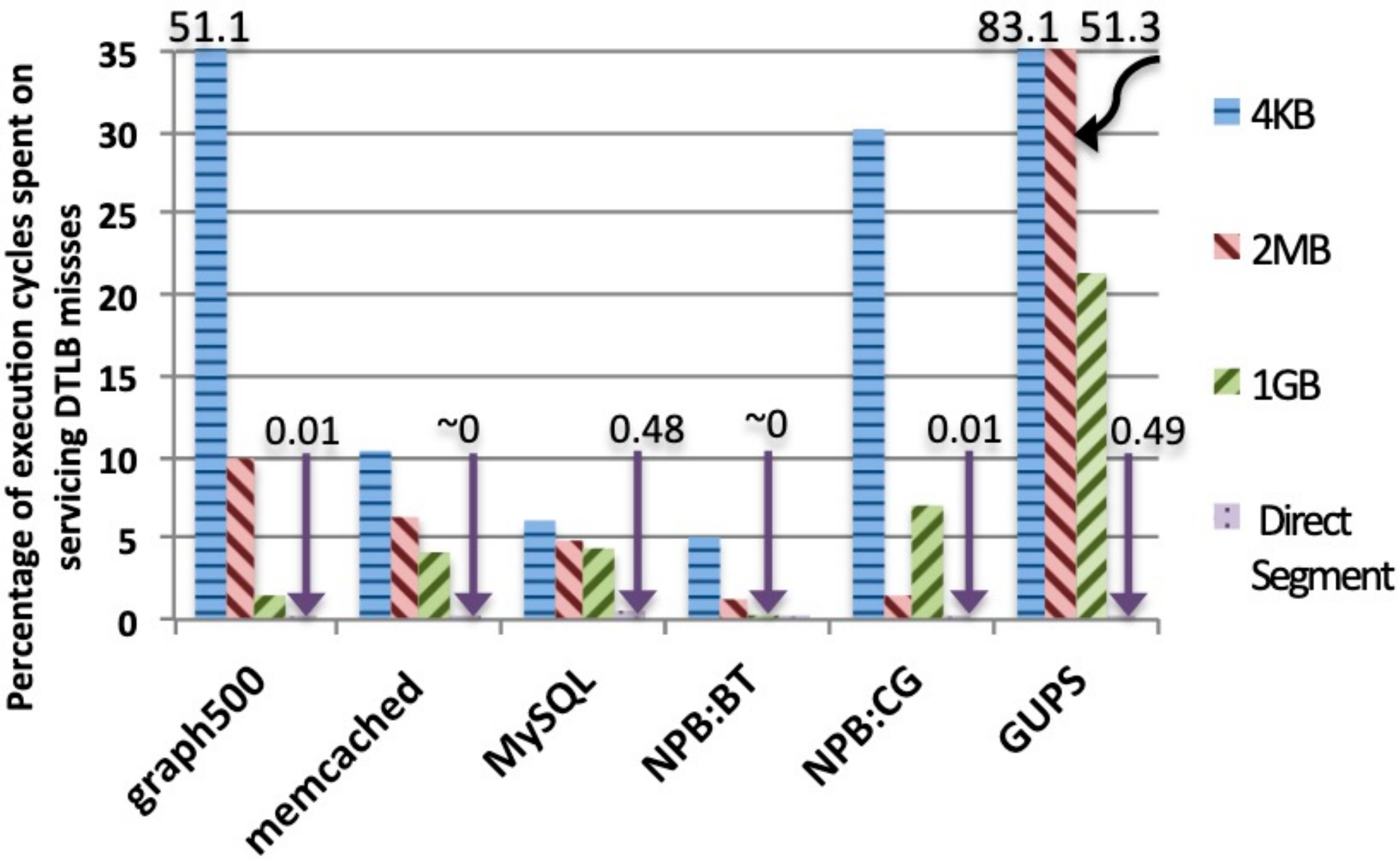
Mega pages (2MB pages)



Giga Pages (1GB Pages)

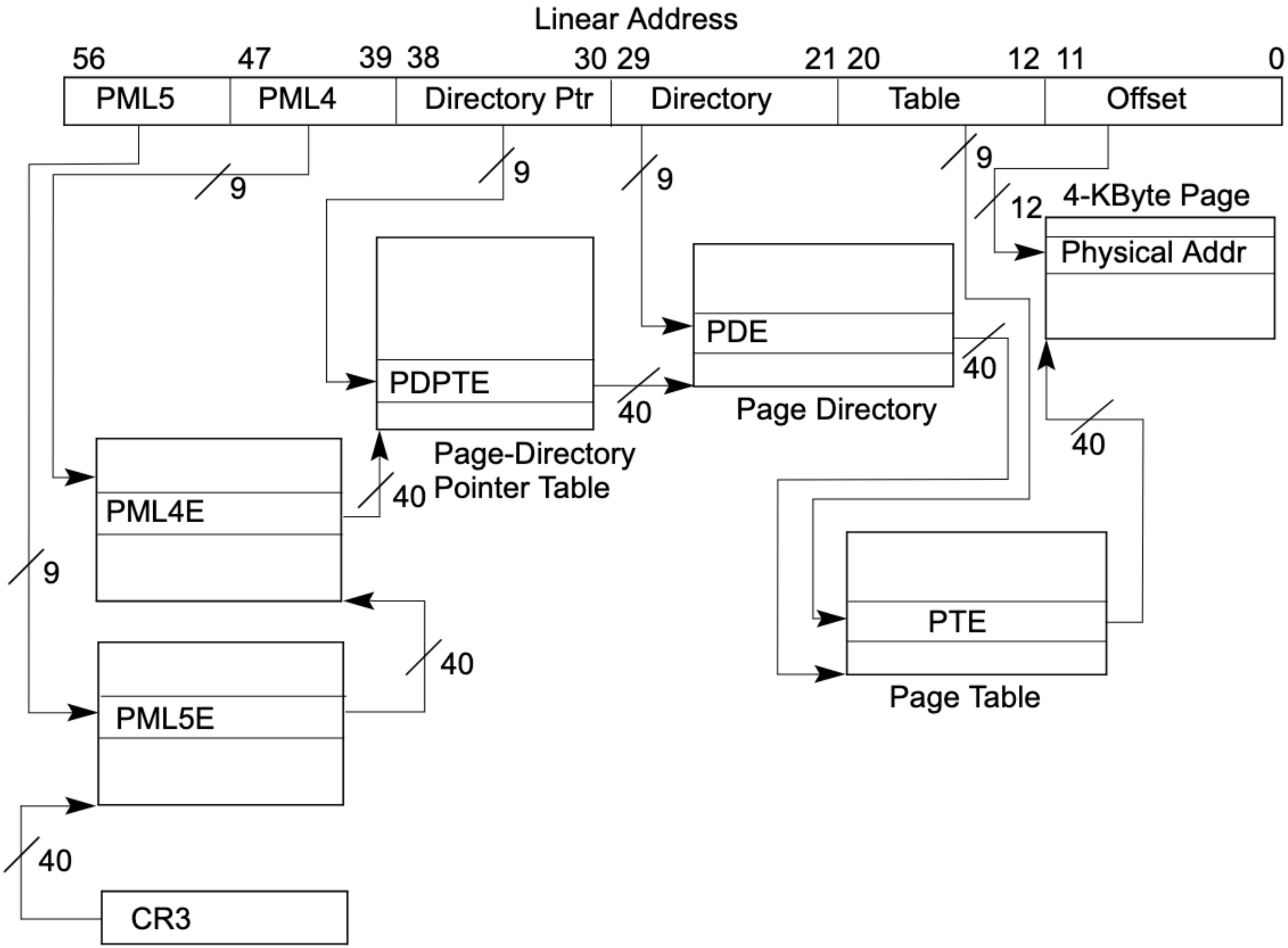


It Matters?



[1] A. Basu, J. Gandhi, J. Chang, M. D. Hill, and M. M. Swift, “Efficient Virtual Memory for Big Memory Servers”, CAL 2013.

Review: Since 2021 (2^{56}B=128PB)

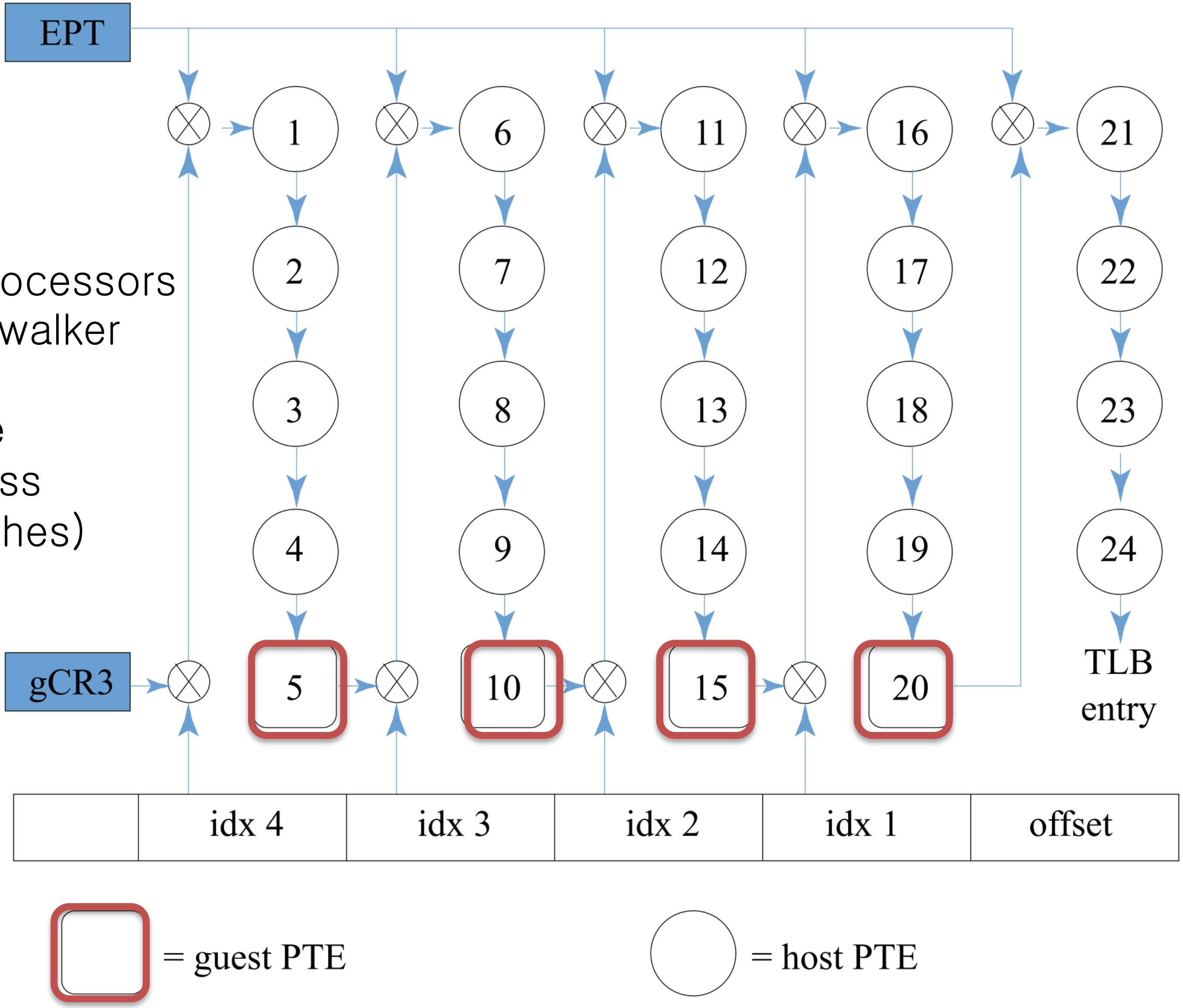


Extended Page Tables

- Circa 2008 Intel (Extended Page Tables or **EPT**) and AMD (Nested Page Tables or **NPT**)
- Add hardware support to complement the classic hardware-defined page table structures (maintained guest-OS) with a **second page table** structure (maintained by the **Hypervisor**)
- TLB miss handling logic is deeply integrated with it (i.e., when we assign the `%cr3`, TLB faults are resolved by TLB walker under the table)
- Memory accesses after TLB miss
  - In root mode EPT is disabled tree levels n=4 (kB pages), n=3, (2MB pages), n=2 (1GB pages), will be *n* access
  - In non-root mode, another table (host-to-physical) with *m* levels will be *n x m + 1 + m*

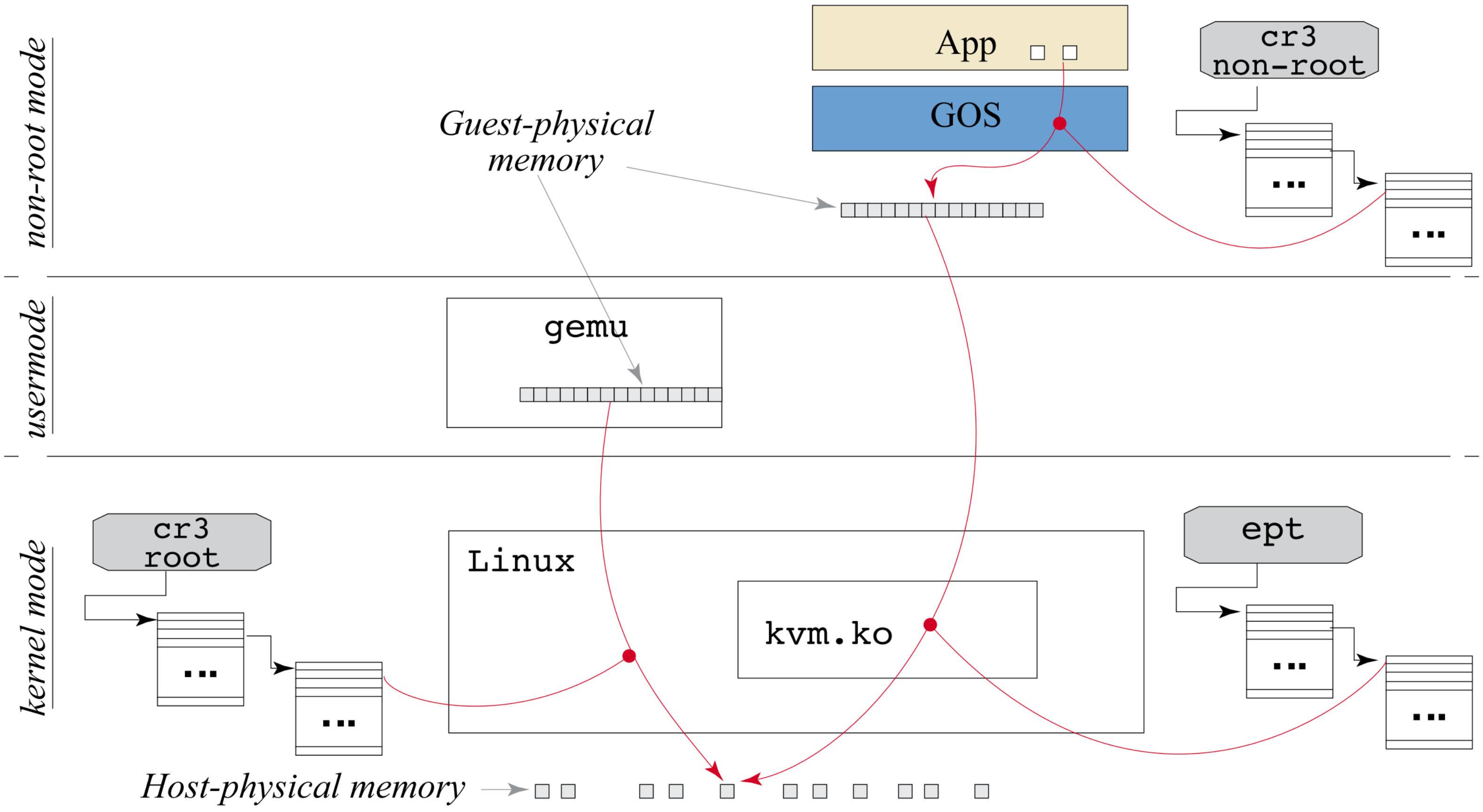
Example: m=4, n=4 (4KB EPT Pages, 4KB Mem Page)

Current processors uses TLB walker caches to accelerate this process (MMU caches)



EPT in KVM

- KVM was conceived when EPT wasn't available: now the feature can be used or not. Today non using it is mostly restricted to nested virtualization

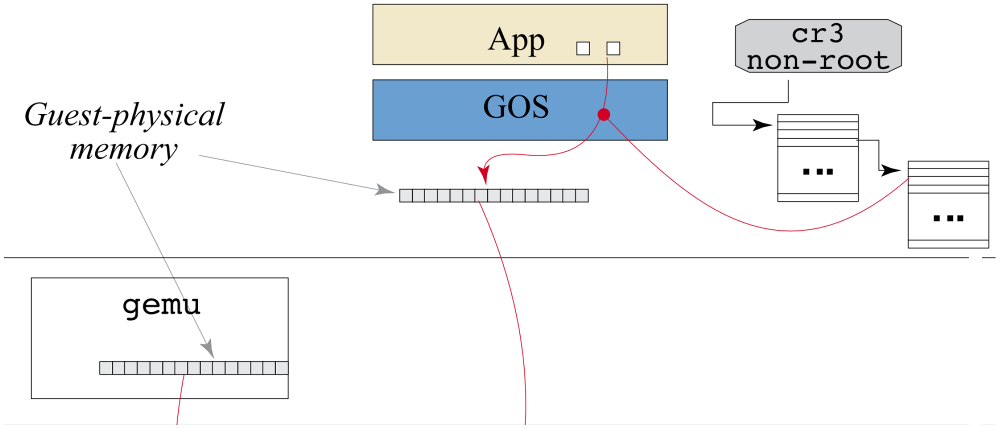


Actors in the memory access pass

- QEMU: Allocates guest-physical **contiguous memory** in his own virtual address space
  - The details about memory management is deferred to the host-OS (type-2)
  - Provides a convenient access from user space to the guest-physical space (crucial for DMA device emulation)
  - (as any other process) will have a %cr3 in the root mode
- Virtual machine manages its own page tables. `Non-root %cr3` points to the page table of the process running (1cpu)
  - Assignments (e.g. context switch) of `non-root %cr3` do not cause #`vmexit`
- In root mode there is special register (`eptp`) pointing to he the EPT. `kvm.ko` **manages** the content of it (one per VM, i.e., accessible via VMCS)
  - One `eptp` per VM

Collateral complications

- `%eip` in App uses virtual addresses: hypervisor needs to decode the mapping before emulate the instruction
  - It’s trivial to access to the guest-physical address space (add a constant)
  - **Its very hard to access** (to emulate) the virtual address space of the App (translations are only at the non-root MMU but non on root [**hypervisor**])

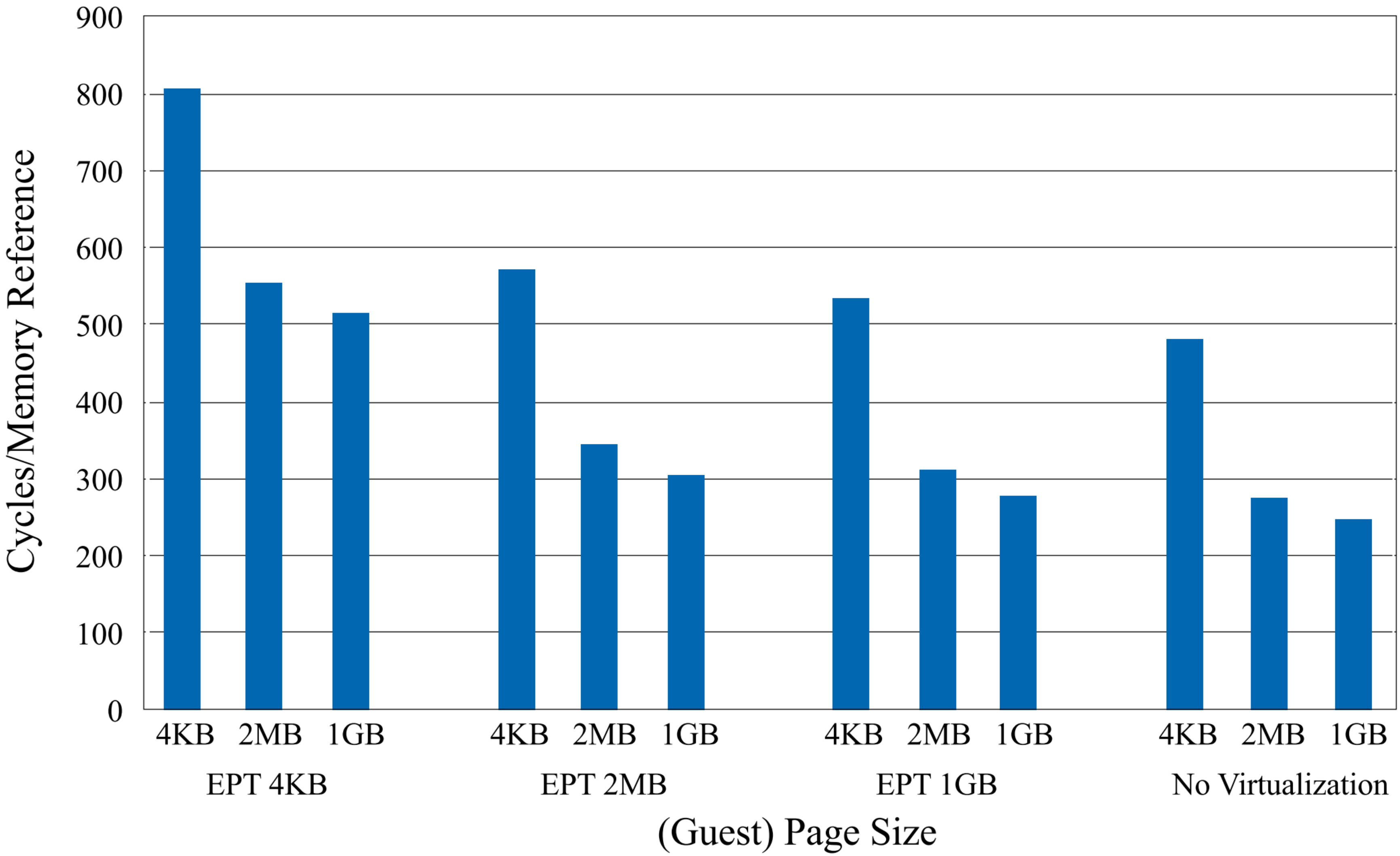


- Type-2 KVM design required a constant synchronization between QEMU and guest-OS mapping
  - E.g. If a page in QEMU is swapped out, affected guest-OS page table should be updated
- **KSM** mechanism (allows to **share read-only pages** within processes) can be used (i.e multiple VM can share host physical pages)

Performance Considerations

- EPT avoids 80% `vmexit`, but TLB misses are much more costly
    - In 64-bit address space, with 4KB (m=n=4), a single TLB miss requires 24 access to memory
    - Page walker caches reduce it (hardware)
- In spite of the efforts, some pointer-chasing benchmarks showed 17% and 39% **performance degradation** on Intel and AMD in early designs
- Next, similar pointer-chasing with Intel **Sandy Bridge** Architectures
    - List of objects that randomly links to a heap of 32GB
    - Measure average access time to each object

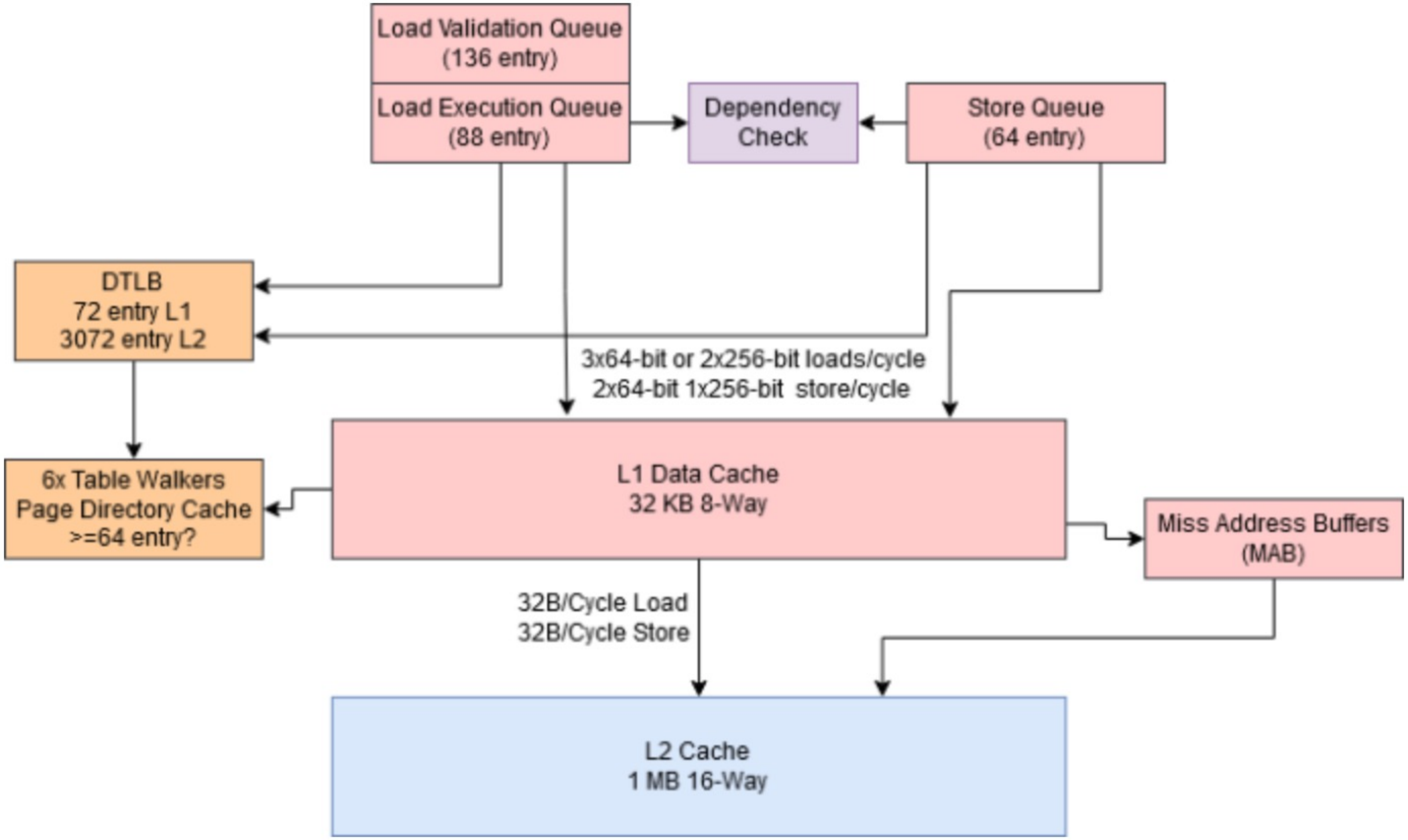
EPT Configuration Effect on Sandy Bridge (non NUMA effects)



Tradeoffs in EPT

- Although prev. benchmark is somewhat unrealistic, **TLB might have** a significant effect on performance
- **Carefully choose EPT** configuration: most Hypervisors uses a 2MB page EPT (internal fragmentation is better than memory access time degradation)
- Also **fine-tuning applications** require in many cases to use 2MB super-pages (or huge)
- **Constant improvements** in hardware generation to generation (and software, e.g. DBT, Unikernels)
- It's needed memory virtualization in **multi terabyte** RAM sytems? (e.g. GPT-3 requires ~1TB)

How to addressed TLB pressure: Zen 4


Load Validation Queue (136 entry)
Load Execution Queue (88 entry)
Dependency Check
Store Queue (64 entry)
DTLB
72 entry L1
3072 entry L2
3x64-bit or 2x256-bit loads/cycle
2x64-bit 1x256-bit store/cycle
6x Table Walkers
Page Directory Cache
>=64 entry?
L1 Data Cache
32 KB 8-Way
Miss Address Buffers (MAB)
32B/Cycle Load
32B/Cycle Store
L2 Cache
1 MB 16-Way