Hash, Don’t Cache (the Page Table)

Idan Yaniv, Dan Tsafrir

TECHNION
Israel Institute
of Technology

SIGMETRICS ’16
June 14 - 18, 2016
Antibes Juan-Les-Pins, France

Presentation by Nicolas Winkler

Outline

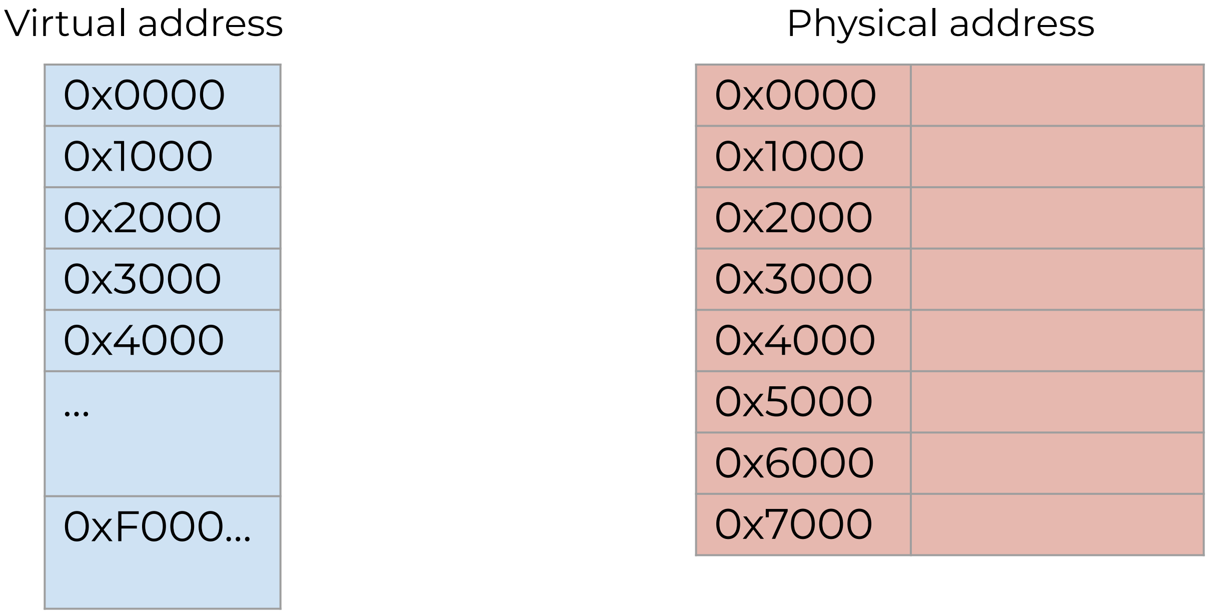
- Background
- Skip, Don’t Walk (the Page Table)
- Motivation
- Rethinking Hash-based Page Tables
- Evaluation
- Strengths & Weaknesses
- Discussion

Outline

- **Background**
  - **Virtual Memory**
  - **Page Table Design**
  - **Memory Management Unit**
- Skip, Don’t Walk (the Page Table)
- Motivation
- Rethinking Hash-based Page Tables
- Evaluation
- Strengths & Weaknesses
- Discussion

Virtual Memory Basics

- Virtual memory introduces indirect addressing to:
  - Provide the impression of infinite memory
  - Enable application-transparent memory management



Virtual address

Virtual address
0x0000
0x1000
0x2000
0x3000
0x4000
…
0xF000…

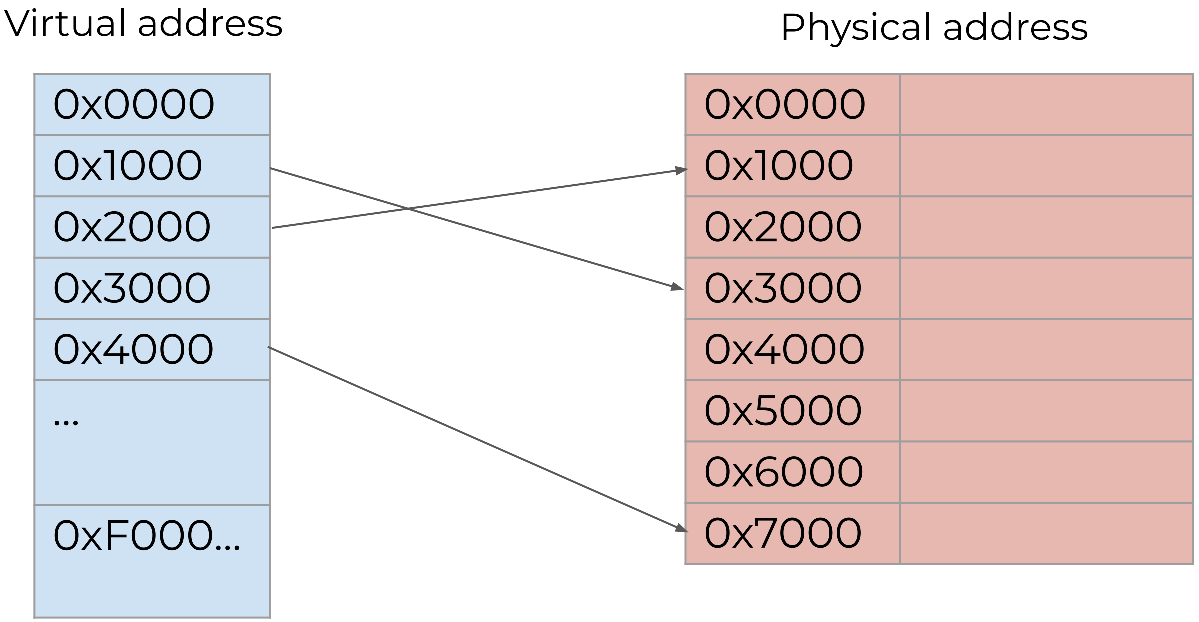
Physical address

Physical address	
0x0000	
0x1000	
0x2000	
0x3000	
0x4000	
0x5000	
0x6000	
0x7000	

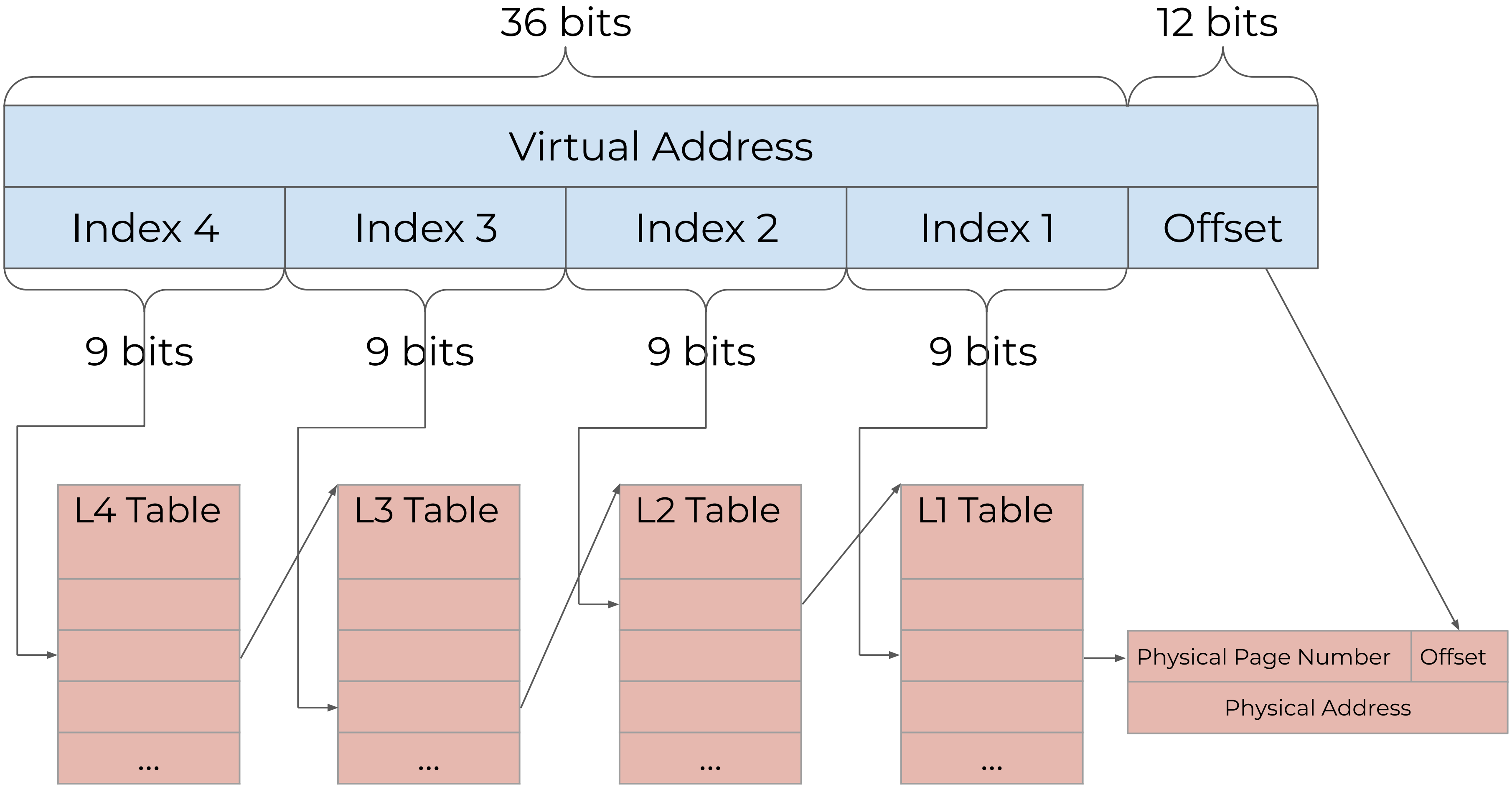
- Virtual memory is used in many modern computing systems like CPUs and GPUs

Virtual Memory Basics

- OS creates virtual-to-physical mapping for each process
- The virtual and physical address space is split into chunks which are called pages or frames
- VM frameworks use a data structure called page table to store the virtual to physical page mapping

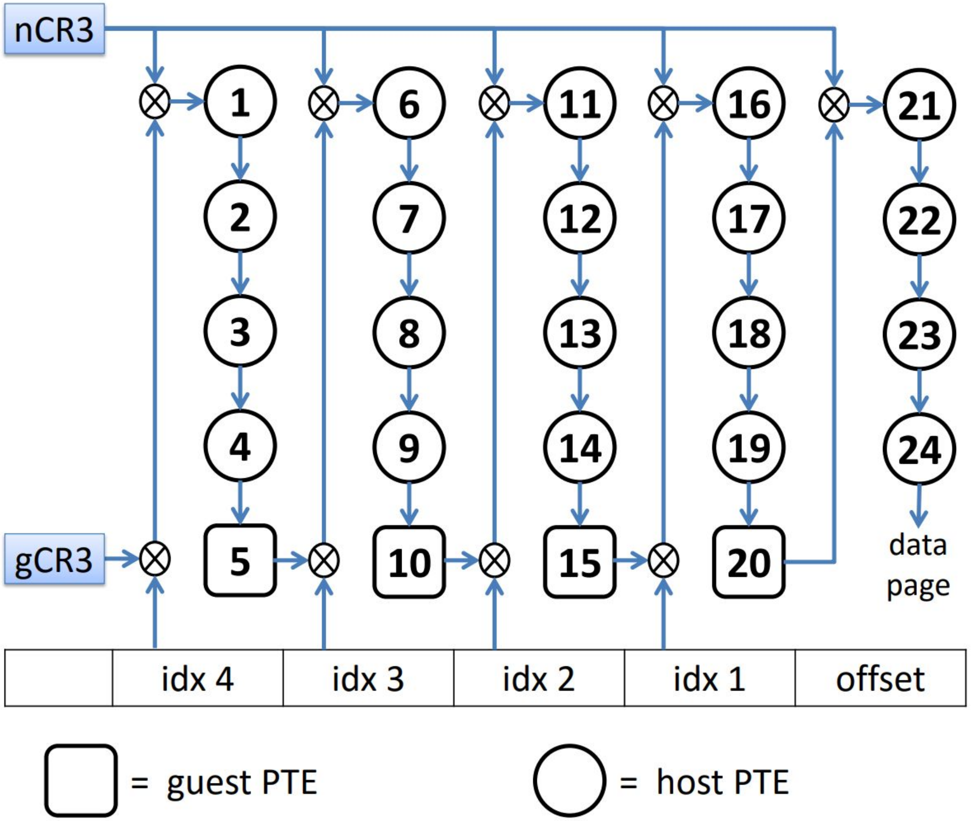


x86-64 Page Table Walk



Radix-based page tables are storage-efficient and easily modifiable but require four sequential memory accesses to perform address translation

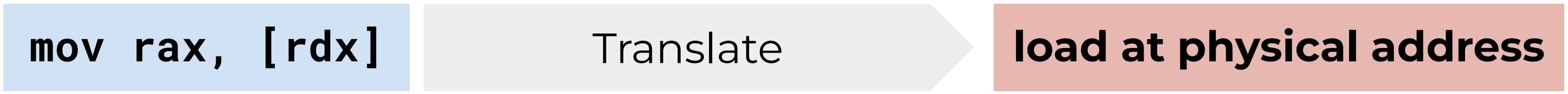
x86-64 Nested Page Table Walk



In the case of Nested Page Walks even 24 memory accesses are needed

Virtual Memory - Address Translation

- For each memory access a program does, the memory address has to be translated



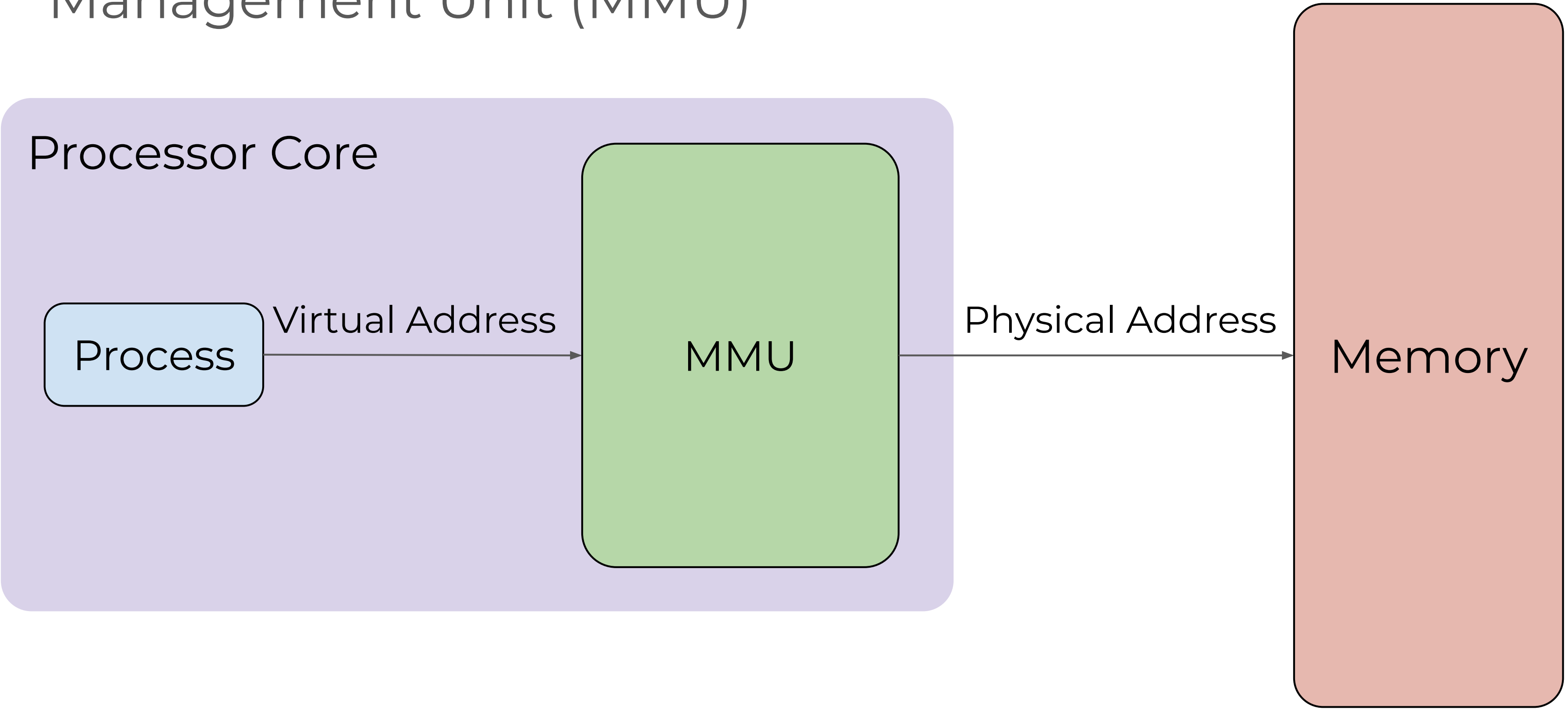
```
mov rax, [rdx]
```

Translate

load at physical address

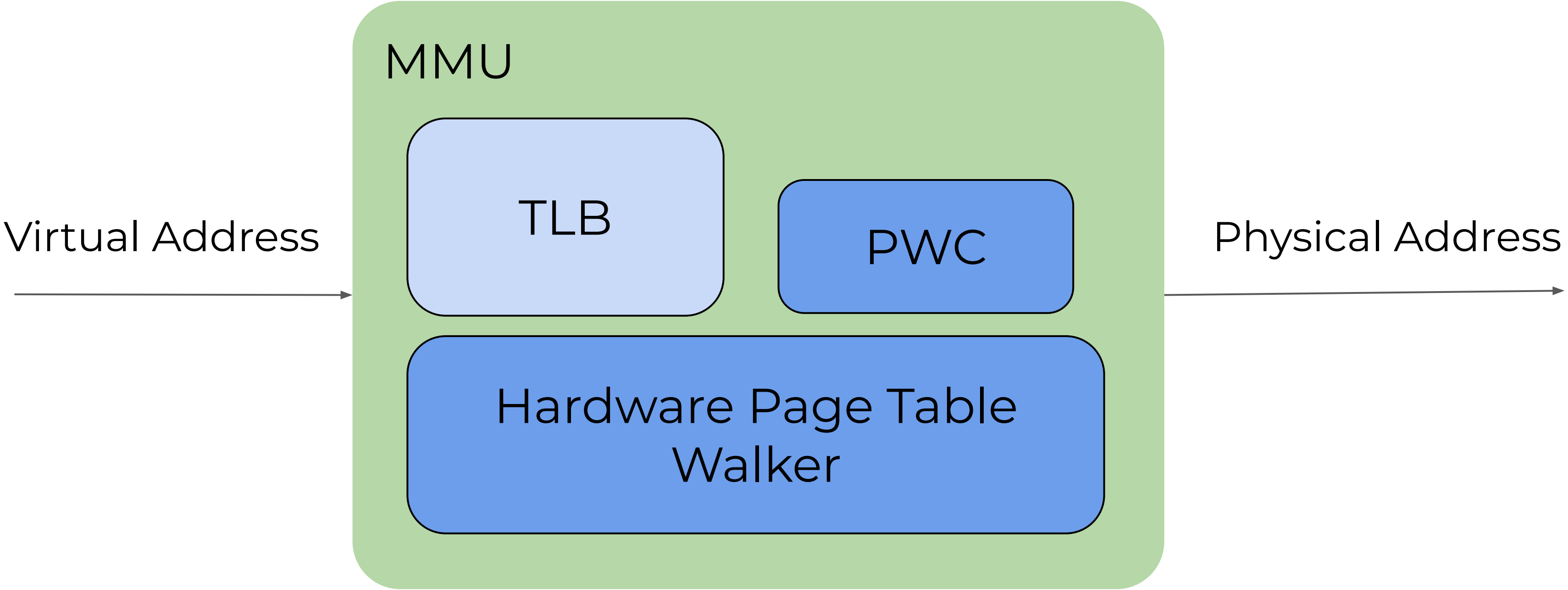
Virtual Memory - Hardware Support

- In order to speed up address translation, current systems have hardware support – called the Memory Management Unit (MMU)



Translation Lookaside Buffer

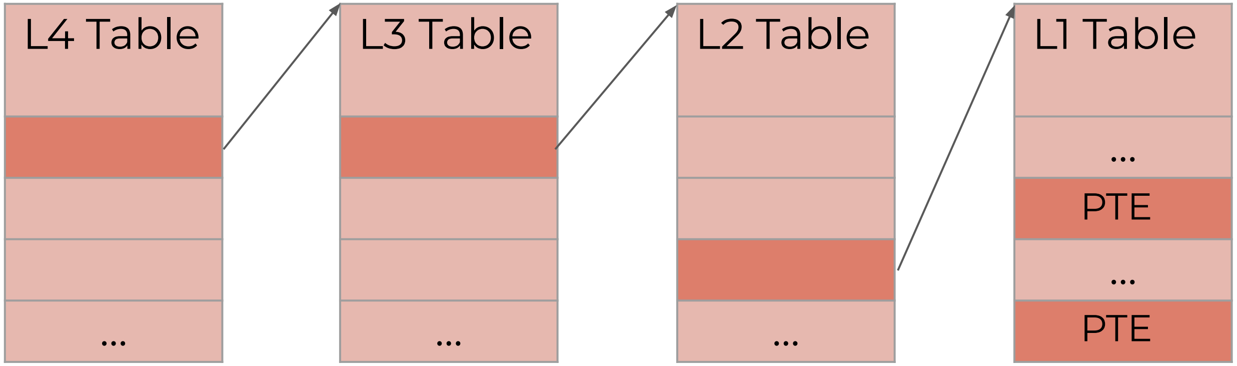
- TLB keeps recently used translations cached
  - VPN → PPN
- Hardware Page Table Walker
- Page Walk Caches are used to accelerate PTWs



Page Walk Caches

- Two consecutive memory accesses often share part of the page walk

Virtual Address	Idx 4	Idx 3	Idx 2	Idx 1	Offset
0x41fe1c	000	000	002	01f	e1c
0x5df2a4	000	000	002	1df	2a4



Page Walk Caches

- Two consecutive memory accesses often share part of the page walk

Virtual Address	Idx 4	Idx 3	Idx 2	Idx 1	Offset
0x41fe1c	000	000	002	01f	e1c
0x5df2a4	000	000	002	1df	2a4

Tag	Value
000	L3 Table
---	-
---	-

Tag	Value
000/000	L2 Table
---/---	-
---/---	-

Tag	Value
000/000/002	L1 Table
---/---/---	-
---/---/---	-

X86-64 Radix-Based Page Table

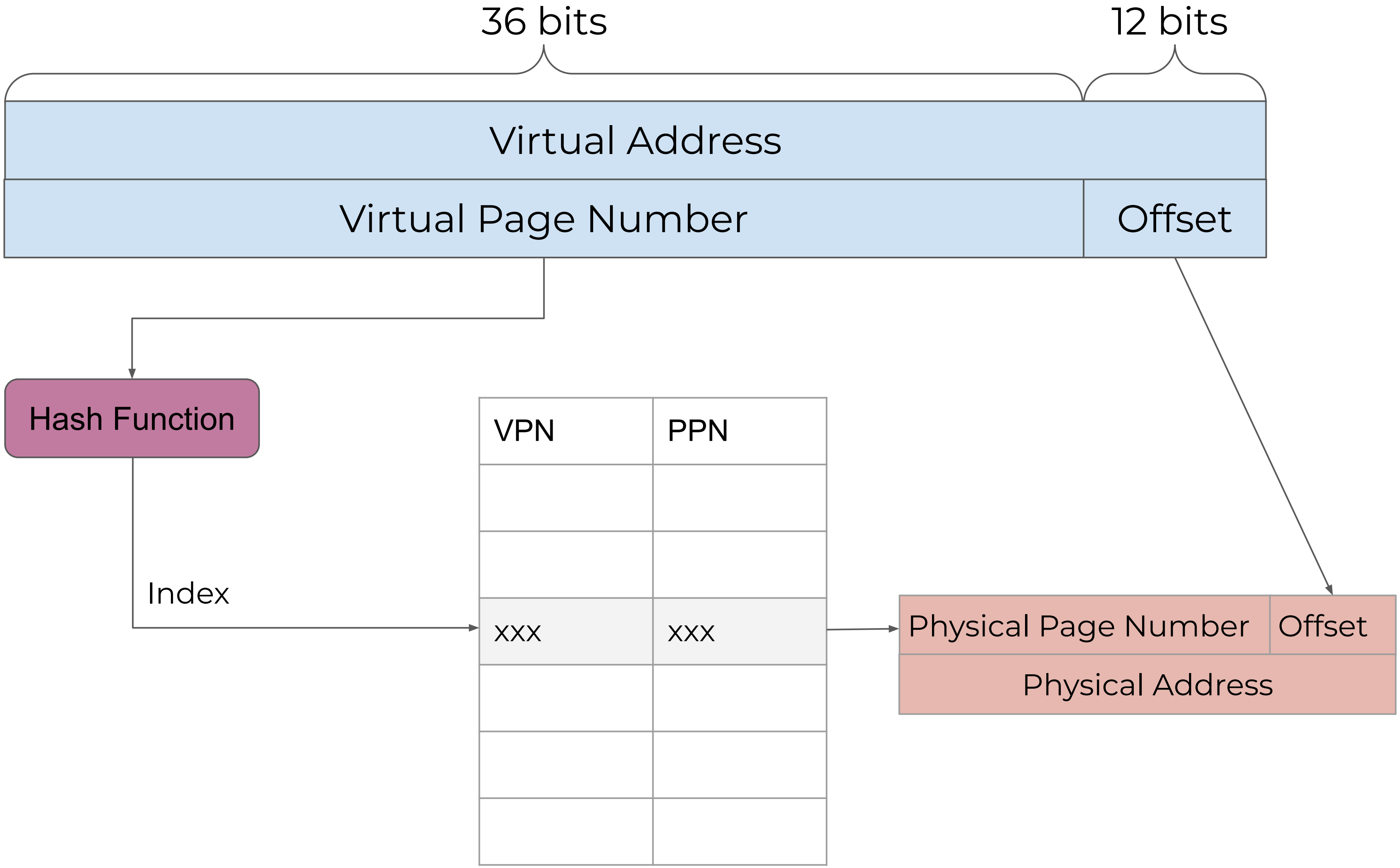
Advantages

- Easily growable/shrinkable according to needs
- Very fast using TLB and PWCs

Disadvantages

- Relies on locality of accesses (although over wide memory ranges), slow if a full page walk happens
- Requires a lot of “helper hardware”

Hash-based Page Table



36 bits

12 bits

Virtual Address

Virtual Page Number

Offset

Hash Function

Index

VPN	PPN
xxx	xxx

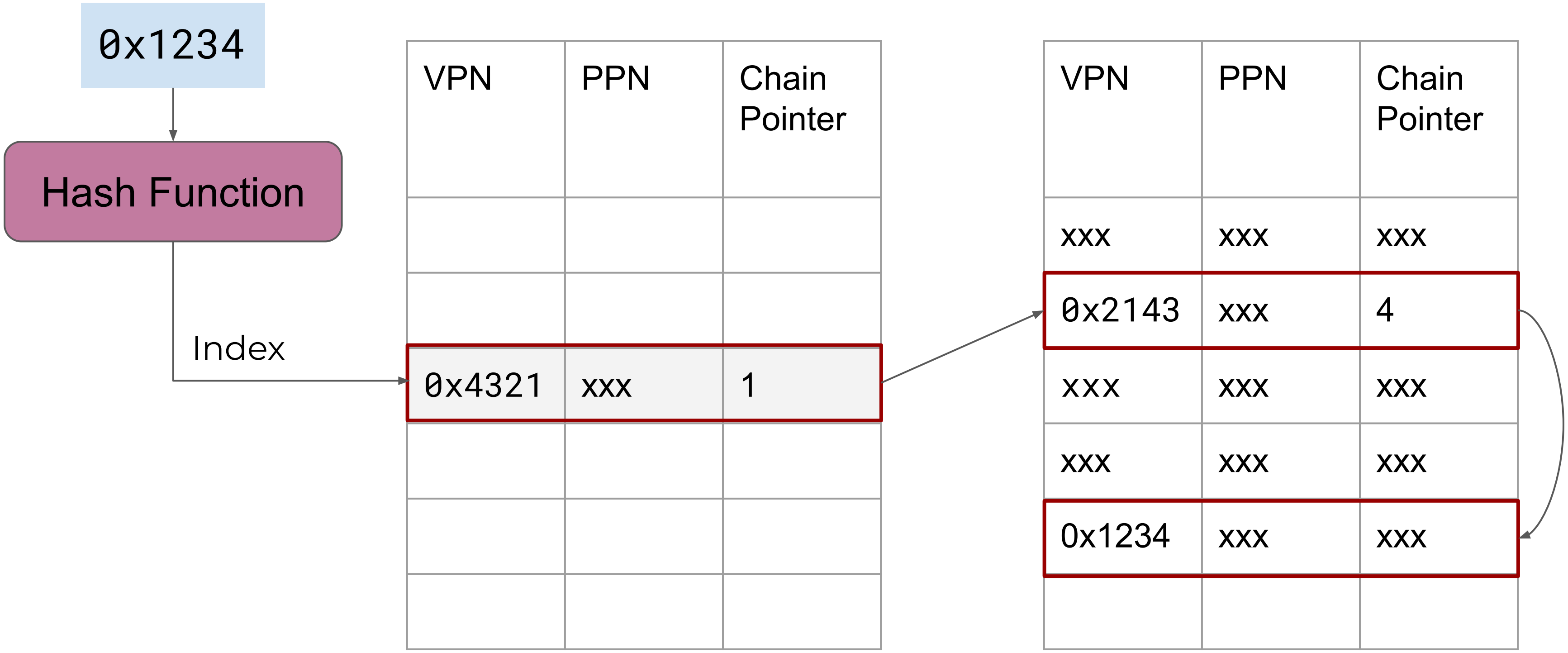
Physical Page Number

Offset

Physical Address

Hashed Page Table

- Need collision resolution
  - One possible solution is to use a chain table



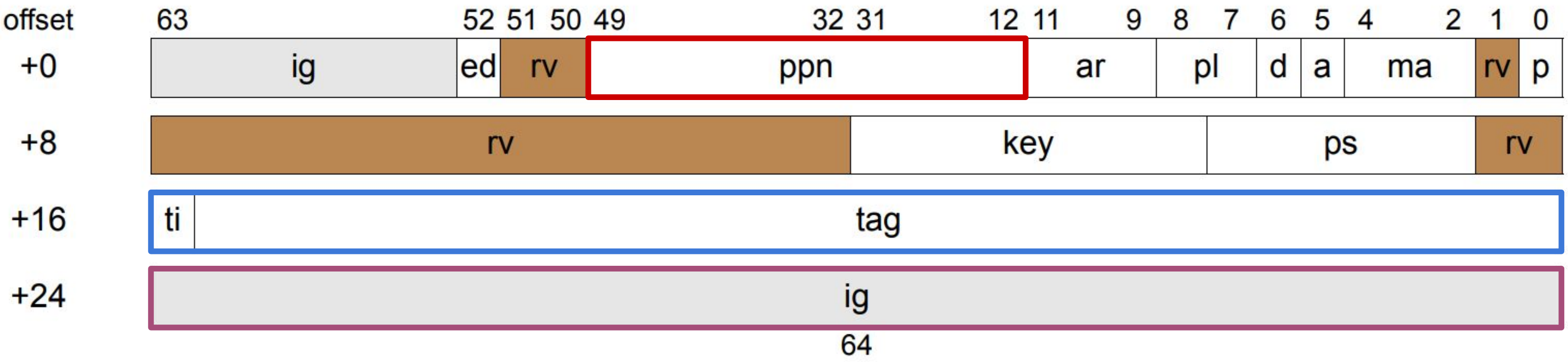
VPN	PPN	Chain Pointer
0x4321	xxx	1

VPN	PPN	Chain Pointer
xxx	xxx	xxx
0x2143	xxx	4
xxx	xxx	xxx
xxx	xxx	xxx
0x1234	xxx	xxx

Many collisions means we have to traverse a long linked list

Example: Itanium Hash Table Design

Figure 4-12. VHPT Long Format



Intel® Itanium® Architecture Software Developer's Manual, Volume 2

Hashed Page Tables

Advantages

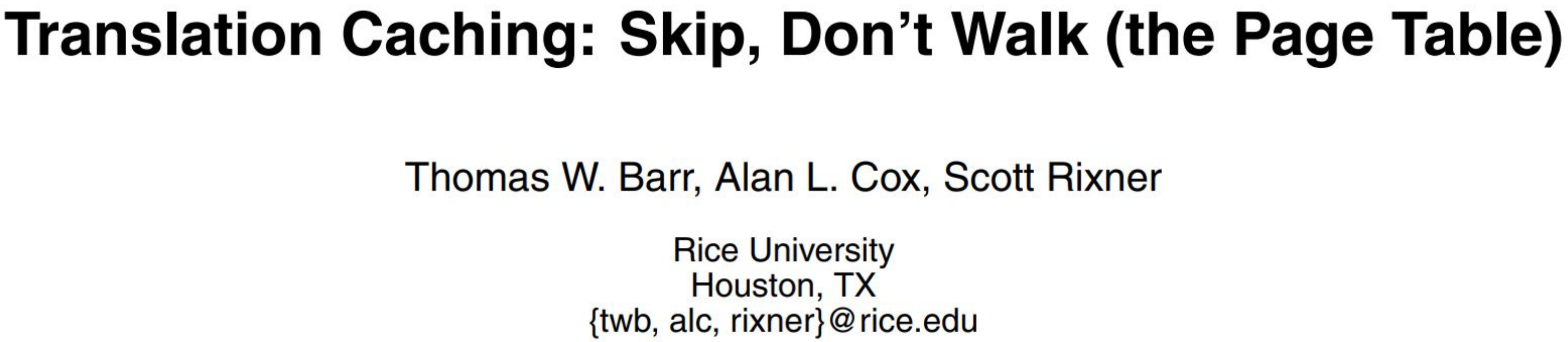
- Only one memory access (if no collision, on average slightly higher)
- Doesn’t require PWCs to be fast

Disadvantages

- Can’t be easily extended/shrunk
- Underutilization
- Potentially high number of collisions

Outline

- Background
- **Skip, Don’t Walk (the Page Table)**
- Motivation
- Rethinking Hash-based Page Tables
- Evaluation
- Strengths & Weaknesses
- Discussion



Translation Caching: Skip, Don't Walk (the Page Table)

Thomas W. Barr, Alan L. Cox, Scott Rixner

Rice University
Houston, TX
{twb, alc, rixner}@rice.edu

ISCA, 2010

- Analyzes effect of page walk caches
- Compares them against hashed page tables similar to the implementation in Itanium processors

Conclusion: Radix Page Tables are superior to Hashed Page Tables by a large margin.

- Hashed Page Tables cause about 1.2 memory accesses per lookup, only 44% of which L2 hits
- over 400% more DRAM accesses than radix page tables.

Motivation

“In all affairs it’s a healthy thing now and then to hang a question mark on the things you have long taken for granted.” (B. Russell)

We show that, when carefully optimized, hashed page tables in fact outperform existing PWC-aided x86-64 hardware, shortening benchmark runtimes by 1%–27% […].

- Hash, Don’t Cache (the Page Table)

Goals of this work

- Optimize the Itanium hash-based page table implementation

- Compare the optimized hash-based page table to Radix Page Tables with PWCs

- Demonstrate that optimizing hash-based page table leads to highly efficient address translation

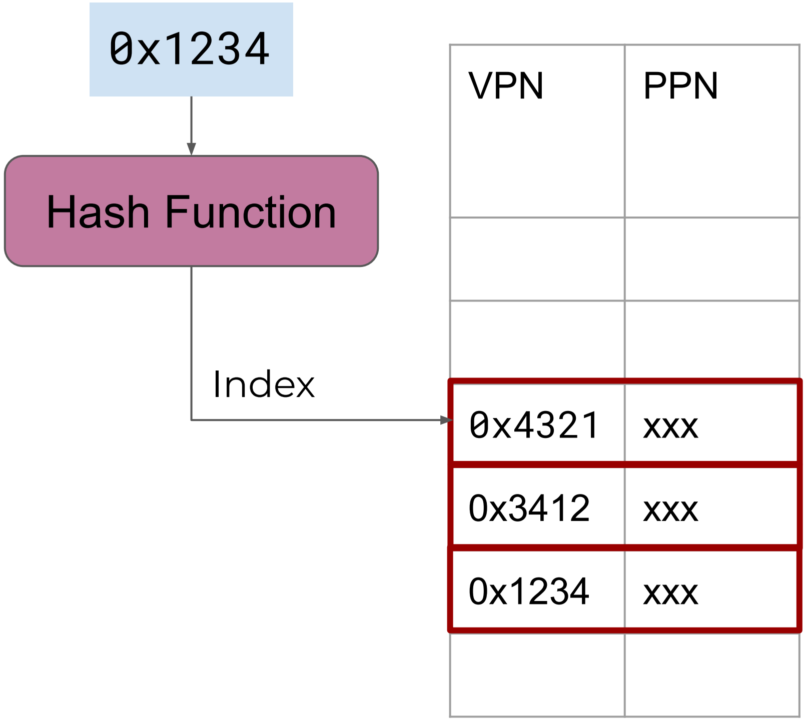
Outline

- Background
- Skip, Don’t Walk (the Page Table)
- Contradiction
- **Rethinking Hash-based Page Tables**
    - **Optimization #1: Open Addressing**
    - **Optimization #2: Clustering**
    - **Optimization #3: Compaction**
- Evaluation
- Strengths & Weaknesses
- Discussion

Rethinking Hash-based Page Tables

Open Addressing

- Linear search
- Allows us to get rid of the chain table
- Shrinks Hash Table Entries to 16 bytes

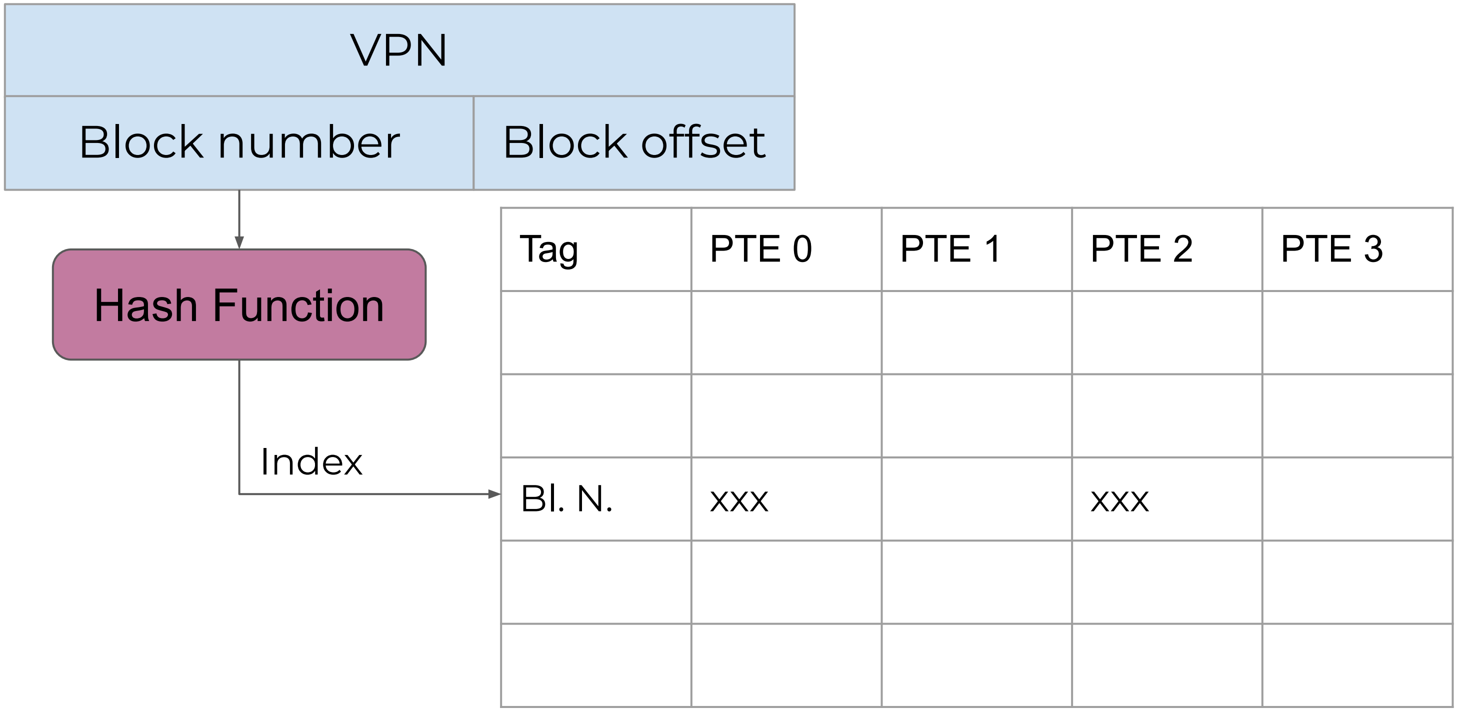


VPN	PPN
0x4321	xxx
0x3412	xxx
0x1234	xxx

Rethinking Hash-based Page Tables

Clustering

- Cluster entries to cache line size
- Better cache locality - leverage spatial locality

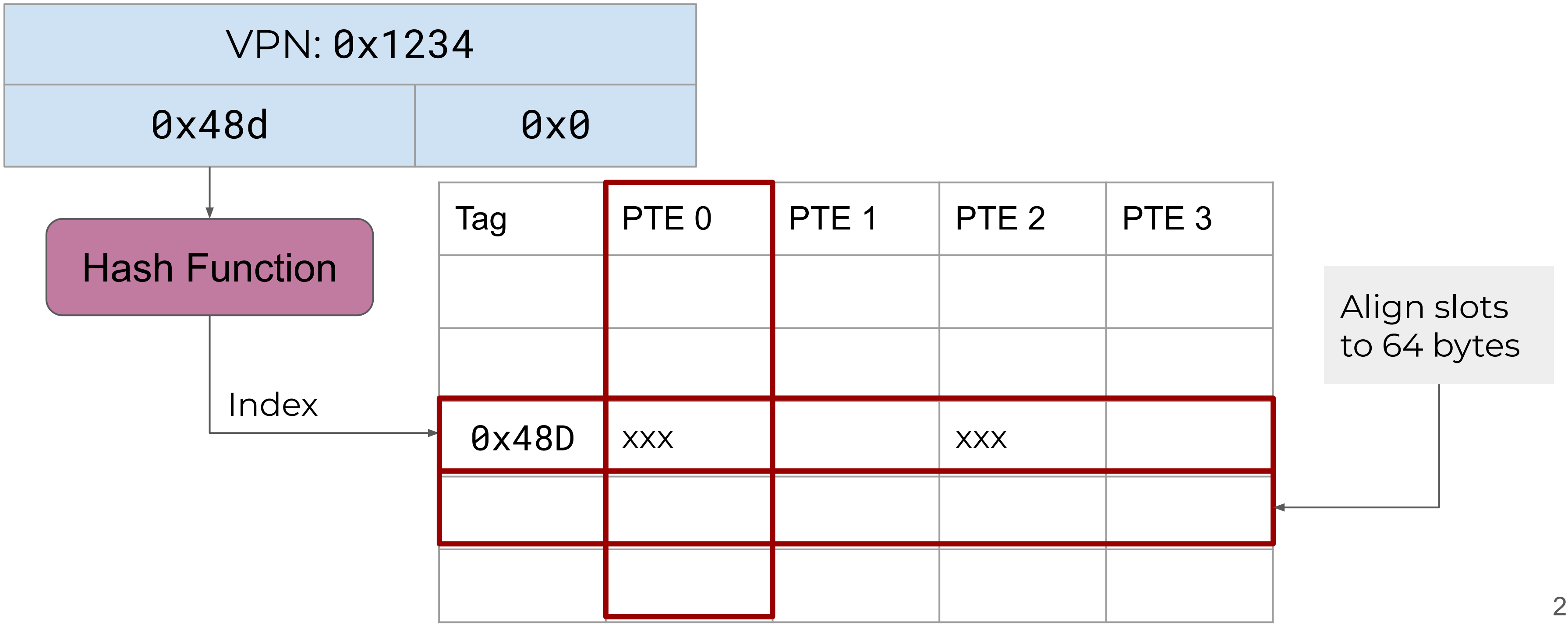


Tag	PTE 0	PTE 1	PTE 2	PTE 3
Bl. N.	xxx		xxx	

Rethinking Hash-based Page Tables

Clustering

- Cluster entries to cache line size
- Better cache locality - leverage spatial locality



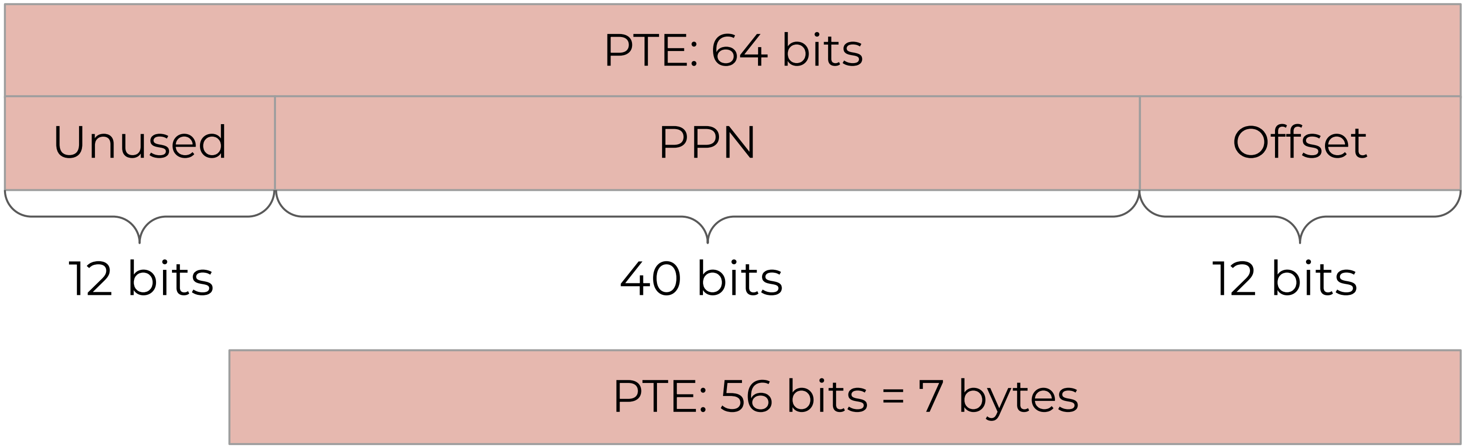
Tag	PTE 0	PTE 1	PTE 2	PTE 3
0x48D	xxx		xxx	

Align slots to 64 bytes

Rethinking Hash-based Page Tables

Compaction

- Discard unneeded bits



PTE: 64 bits

Unused | PPN | Offset

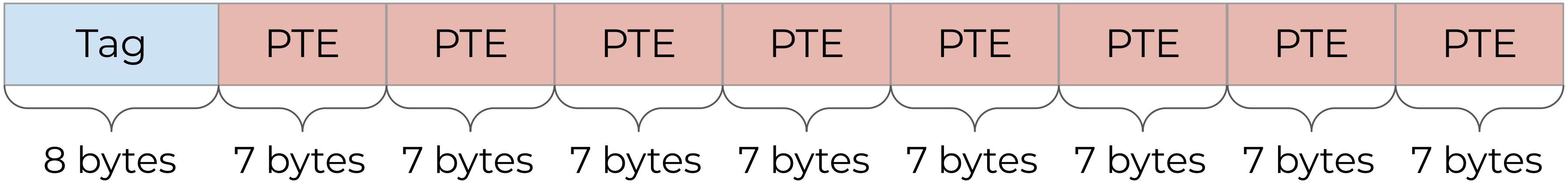
12 bits | 40 bits | 12 bits

PTE: 56 bits = 7 bytes

Rethinking Hash-based Page Tables

Compaction

- Discard unneeded bits
- Allows storing 8 PTEs in one cache line like for radix page tables

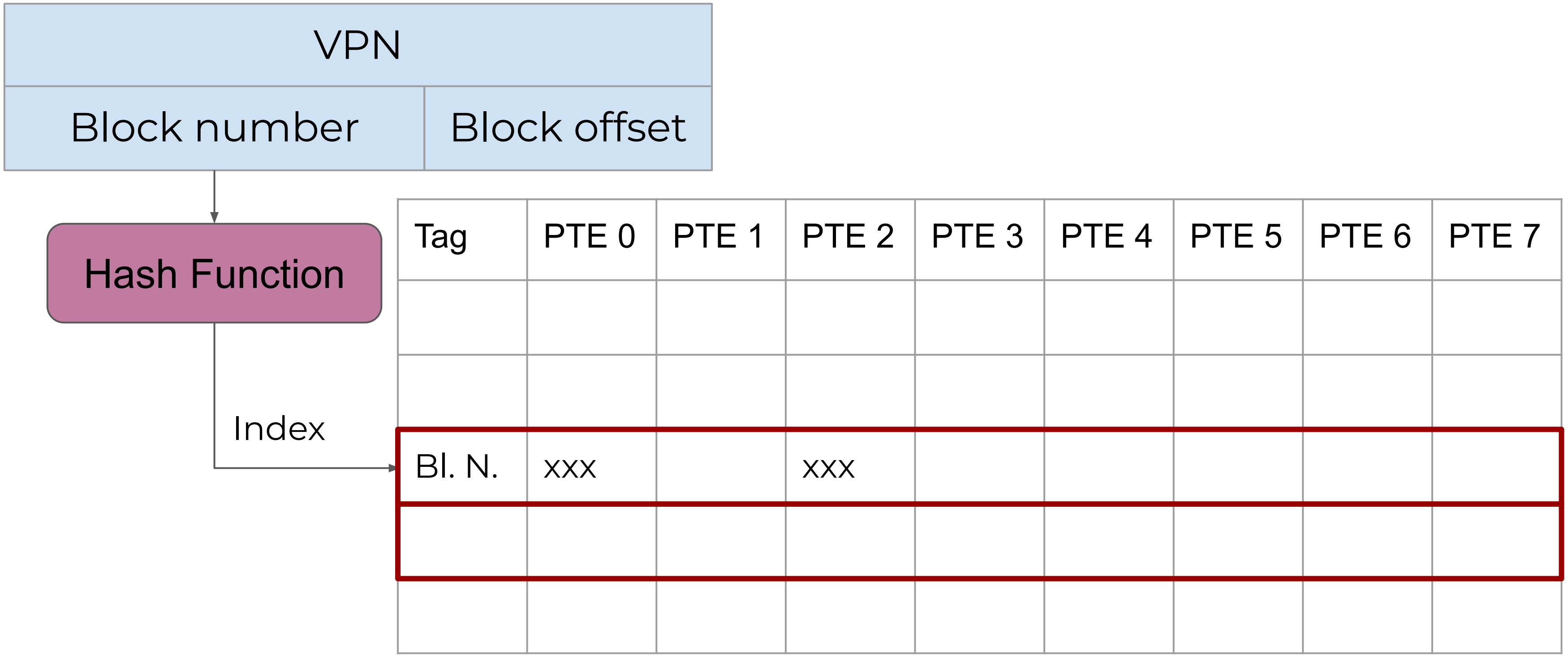


Tag	PTE	PTE	PTE	PTE	PTE	PTE	PTE	PTE
8 bytes	7 bytes	7 bytes	7 bytes	7 bytes	7 bytes	7 bytes	7 bytes	7 bytes

8 bytes + 8 * 7 bytes = 64 bytes = 1 Cache Line

Putting it All Together

- Open Addressing
- Clustering
- Compaction

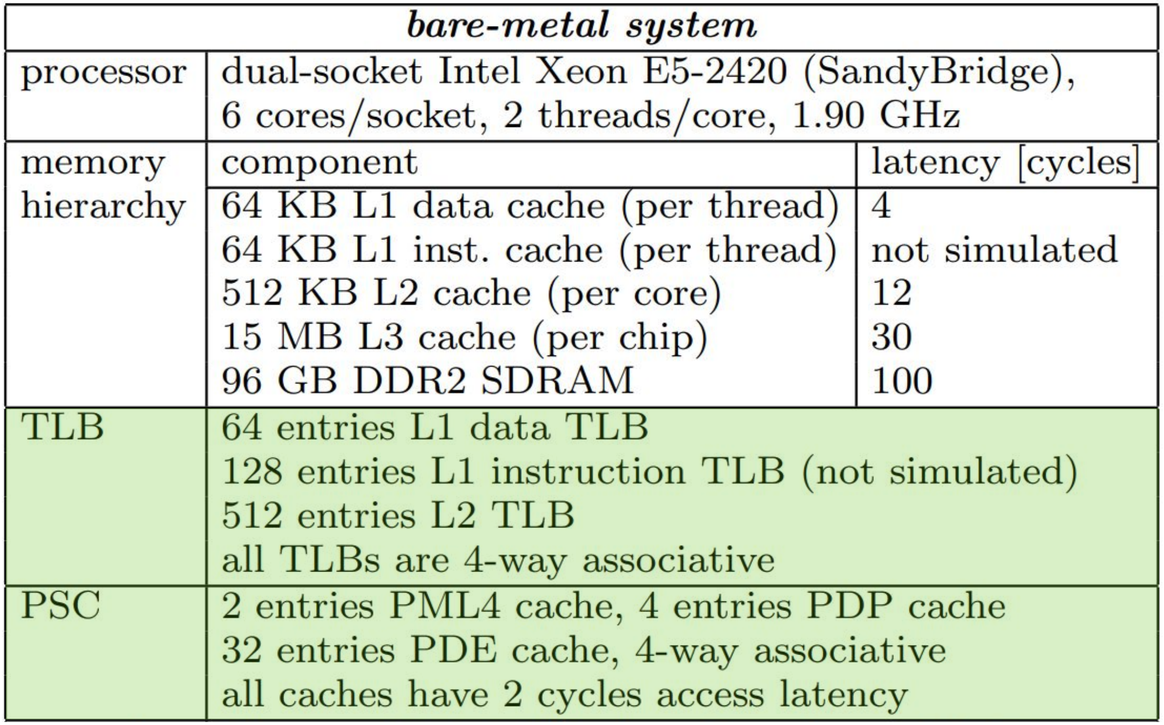


Tag	PTE 0	PTE 1	PTE 2	PTE 3	PTE 4	PTE 5	PTE 6	PTE 7
Bl. N.	xxx		xxx					

Outline

- Background
- Skip, Don’t Walk (the Page Table)
- Motivation
- Rethinking Hash-based Page Tables
- **Evaluation**
- Strengths & Weaknesses
- Discussion

System Configuration + Evaluated Workloads



bare-metal system		
processor	dual-socket Intel Xeon E5-2420 (SandyBridge), 6 cores/socket, 2 threads/core, 1.90 GHz	
memory hierarchy	component	latency [cycles]
	64 KB L1 data cache (per thread)	4
	64 KB L1 inst. cache (per thread)	not simulated
	512 KB L2 cache (per core)	12
	15 MB L3 cache (per chip)	30
	96 GB DDR2 SDRAM	100
TLB	64 entries L1 data TLB 128 entries L1 instruction TLB (not simulated) 512 entries L2 TLB all TLBs are 4-way associative	
PSC	2 entries PML4 cache, 4 entries PDP cache 32 entries PDE cache, 4-way associative all caches have 2 cycles access latency	

Spec cpu2006	**graph500**	**gups**
● mcf	● 4GB	● 2GB
● cactusADM	● 8GB	● 8GB
● xalacbmk	● 16GB	● 32GB

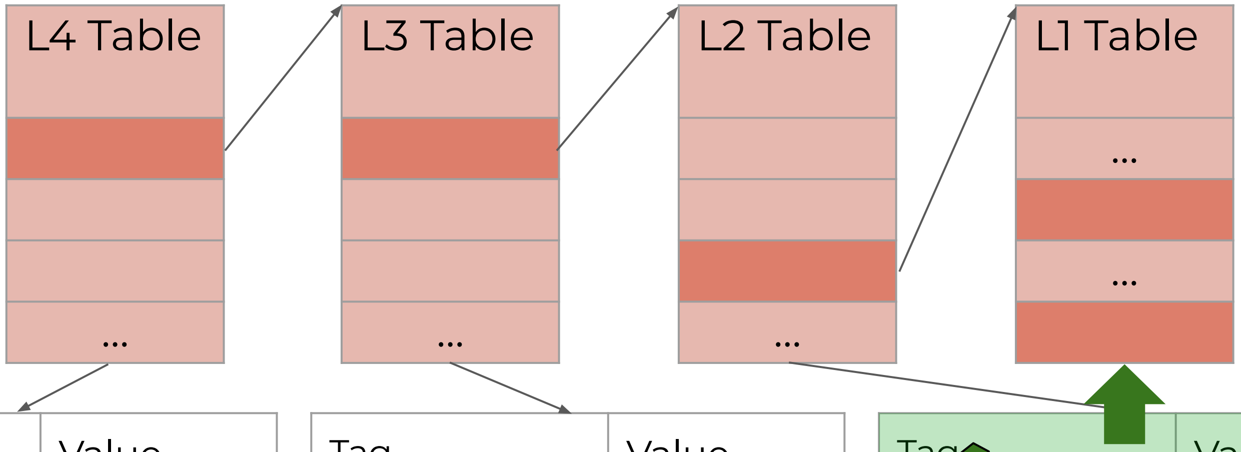
Evaluated Configurations

- Study the performance of memory-intensive programs when using the improved Hash-based Page Table
- Compare the optimized Hash-based Page table against:
  - Current Intel Radix Page Table design with PWCs.
  - A system with an MMU that uses “perfect PWCs”
- Evaluate all designs in both native and virtualized environments

Real x86-64 System vs. Hash Page Table vs. Perfect PWCs

Perfect PWCs

Virtual Address	Idx 4	Idx 3	Idx 2	Idx 1	Offset
`0x41fe1c`	`000`	`000`	`002`	`01f`	`e1c`
`0x5df2a4`	`000`	`000`	`002`	`1df`	`2a4`



Tag	Value
`000`	L3 Table
`---`	-
`---`	-

Tag	Value
`000/000`	L2 Table
`---/---`	-
`---/---`	-

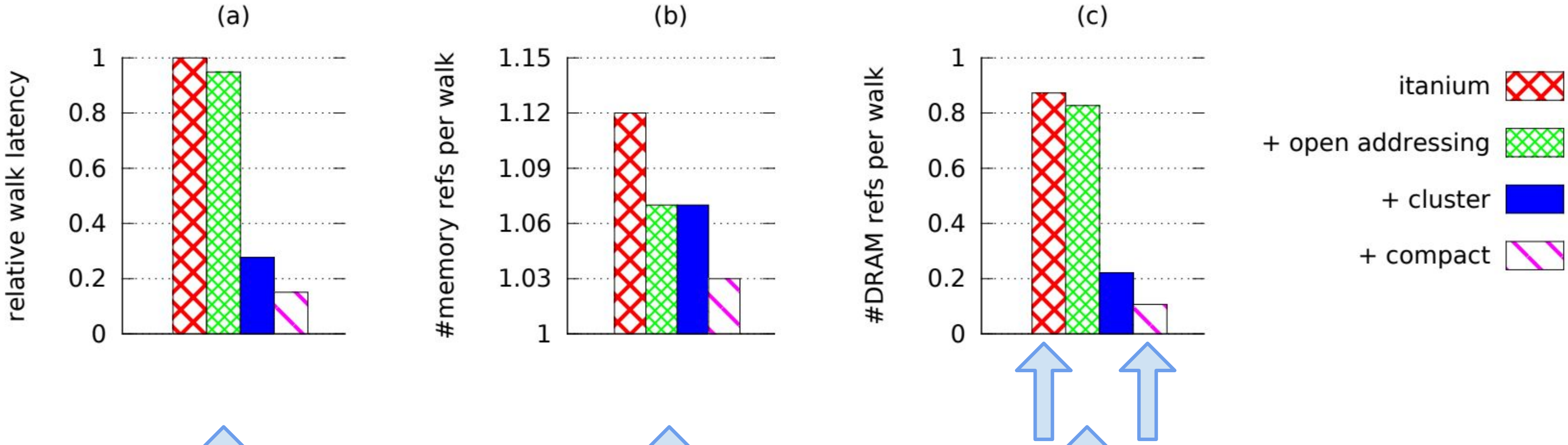
Tag	Value
`000/000/002`	L1 Table
`---/---/---`	-
`---/---/---`	-

Always Hit

Measurements

- Trace all memory accesses of a run using Pin
- Sample a set of these accesses and simulate them in the proposed Architectures
  - optimized Hash Table
  - perfect PWCs
- Add up simulated latencies and calculate runtime

Optimizing Hash-based Page Tables



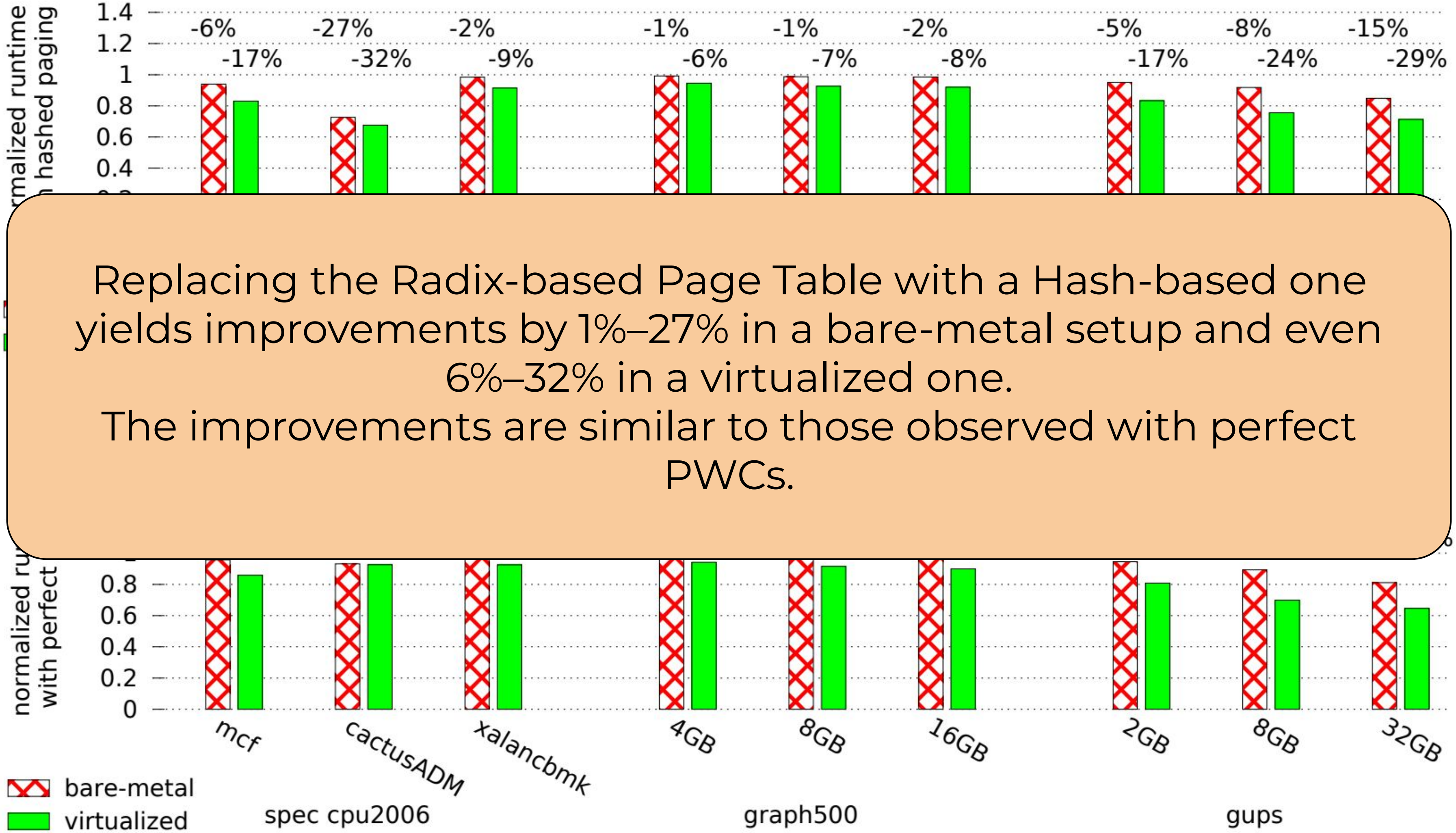
“Overall, the inverted page table increases

By implementing the proposed three optimizations, DRAM references and therefore the walk latency can be reduced by almost a factor of 10

(the Page Table)

Evaluation Results

Hashed Page Tables



Replacing the Radix-based Page Table with a Hash-based one yields improvements by 1%–27% in a bare-metal setup and even 6%–32% in a virtualized one.
The improvements are similar to those observed with perfect PWCs.

Strengths & Weaknesses

Strengths

- Questions an established state-of-the-art solution and brings new insight about the performance of hash based page tables

Weaknesses

- Many unsolved Problems left for future work
  - no support for multiple page sizes
  - how to share pages between processes
  - how to resize the page table
- Optimizes only a small set of programs
- The optimizations seem like small tweaks coming from existing literature
- Only analyzes single program performance
  - No multi-programmed results

Questions

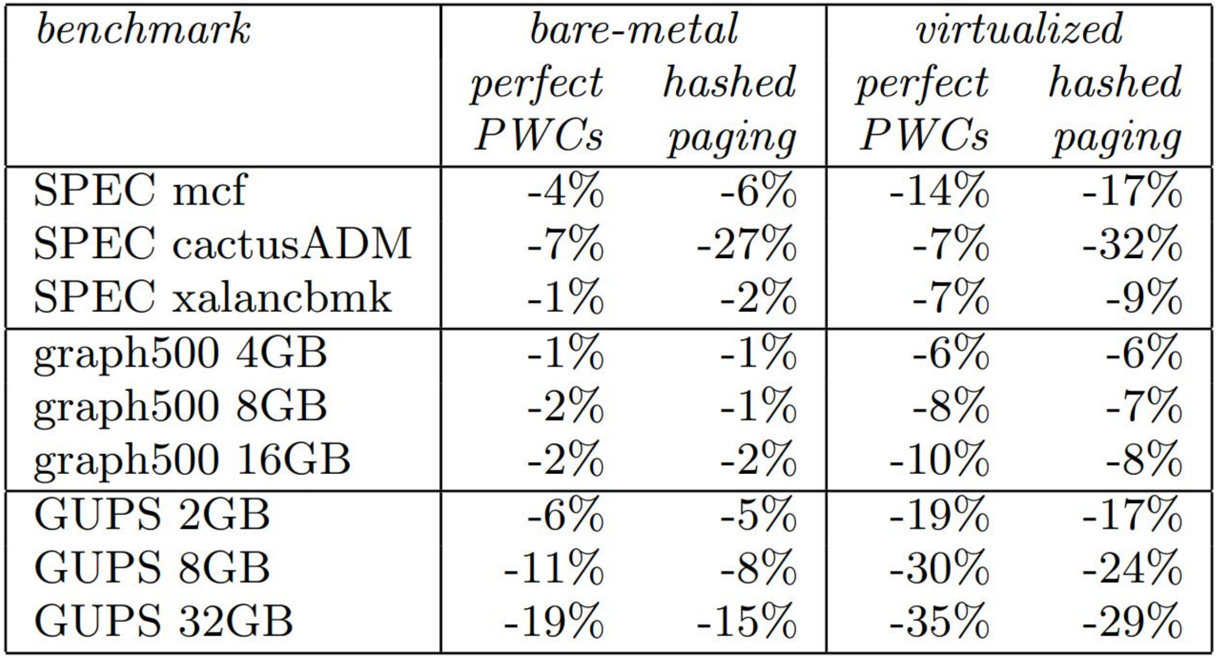
Discussion

- What is more important: Efficient memory usage or performance?
  - Is designing a system that is trading a big chunk of memory usage for some better performance a worthy tradeoff?
- Is it better to use one global hash table or have one per process?
  - What size should they be?
- Can we grow and shrink hash tables, depending on the memory usage?
  - allow the process to control its size?

Thanks!

- Also thanks a lot to my Mentors
  - Konstantinos Kanellopoulos

Measurements



benchmark	*bare-metal*		*virtualized*	
	perfect PWCs	*hashed paging*	*perfect PWCs*	*hashed paging*
SPEC mcf	-4%	-6%	-14%	-17%
SPEC cactusADM	-7%	-27%	-7%	-32%
SPEC xalancbmk	-1%	-2%	-7%	-9%
graph500 4GB	-1%	-1%	-6%	-6%
graph500 8GB	-2%	-1%	-8%	-7%
graph500 16GB	-2%	-2%	-10%	-8%
GUPS 2GB	-6%	-5%	-19%	-17%
GUPS 8GB	-11%	-8%	-30%	-24%
GUPS 32GB	-19%	-15%	-35%	-29%

Virtual Memory Basics

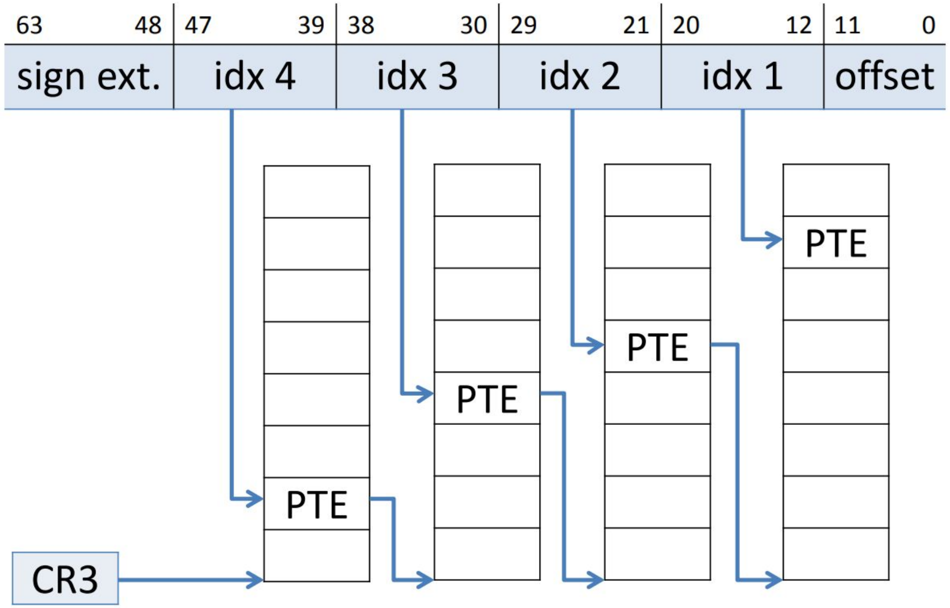


Figure 1: Bare-metal radix page table walk.

Page Table Design - Hash-based

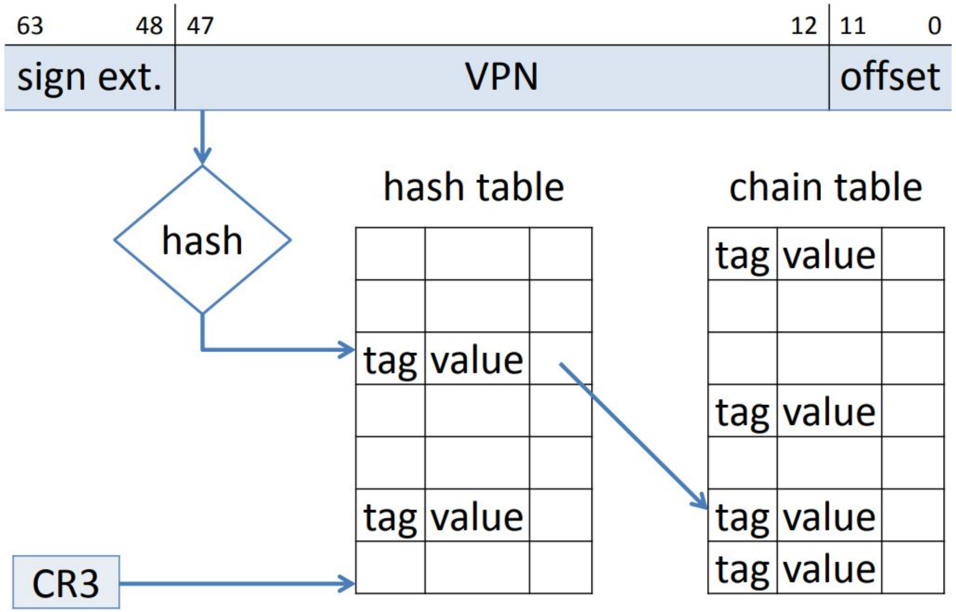


Figure 4: Hashed page table utilizing closed addressing.

Page Table Design - Radix-based

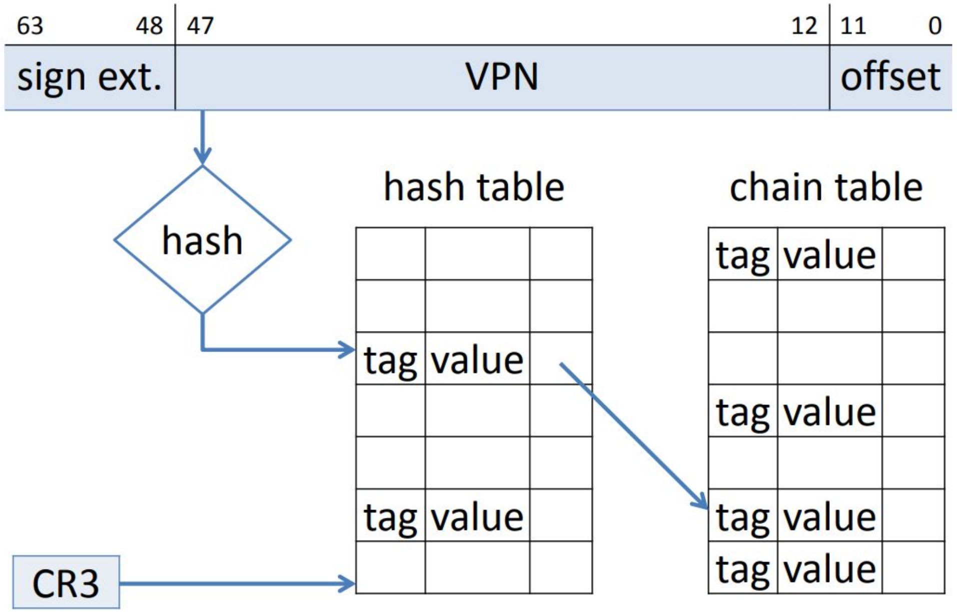


Figure 4: Hashed page table utilizing closed addressing.

Address Translation in Virtualized Environments

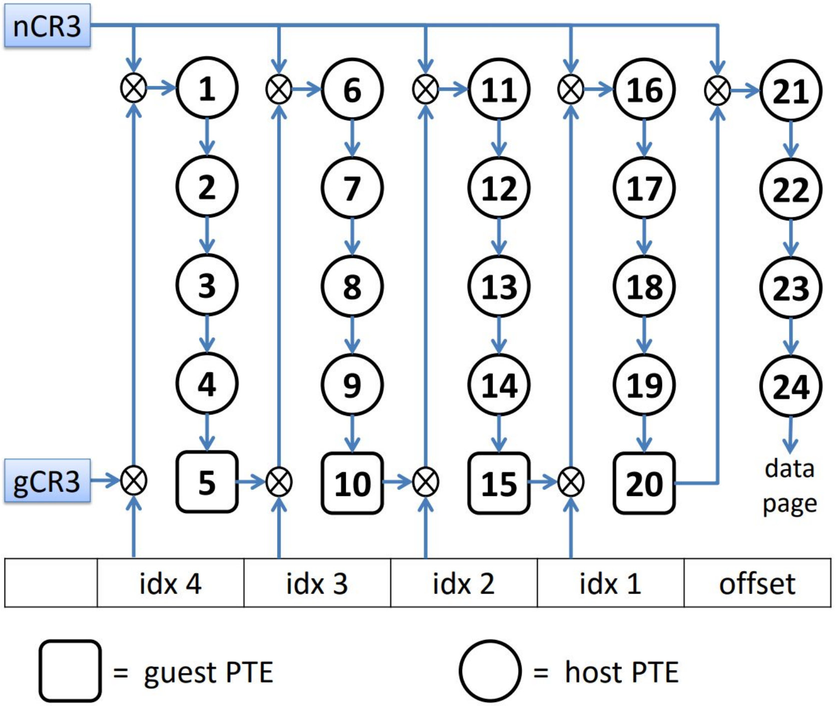


Figure 2: 2D radix page table walk.

Background – Virtual Memory & Page Tables

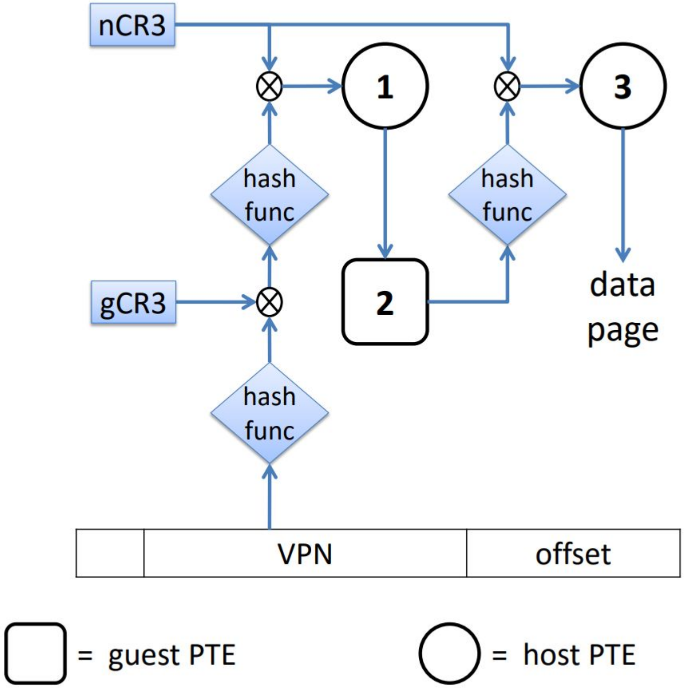


Figure 8: 2D hashed page table walk.

Optimizations

2. Clustering

- cluster entries to cache line size
- better cache locality

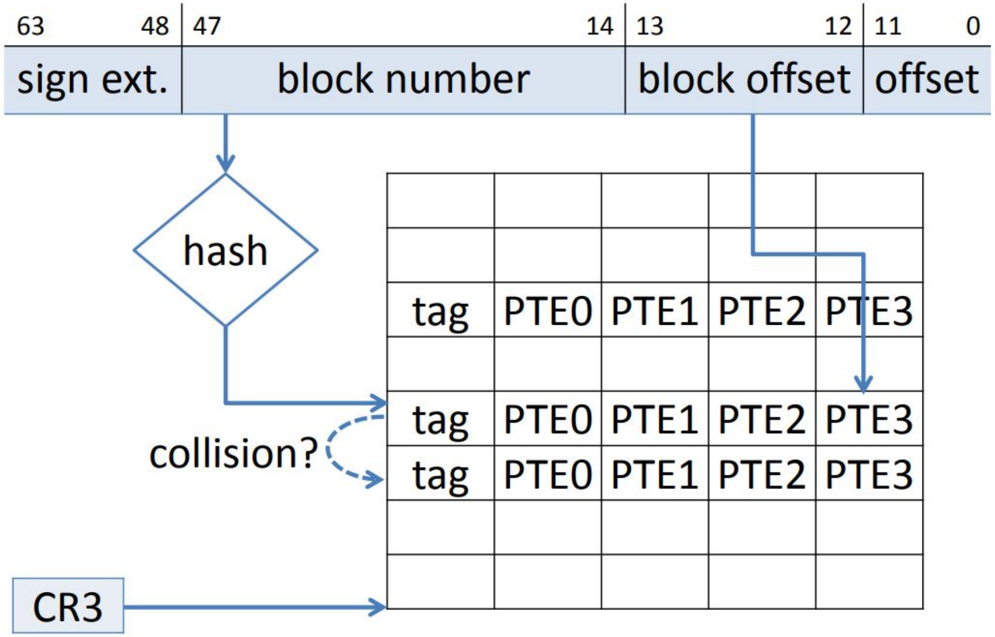
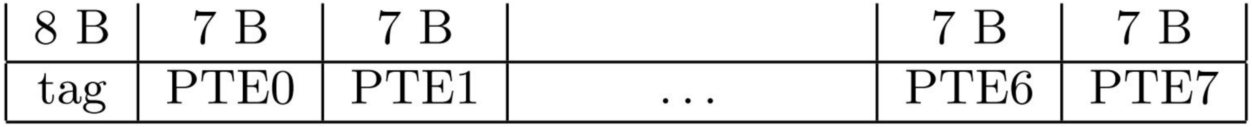


Figure 7: Clustered page table walk.

Optimizations

2. Compaction

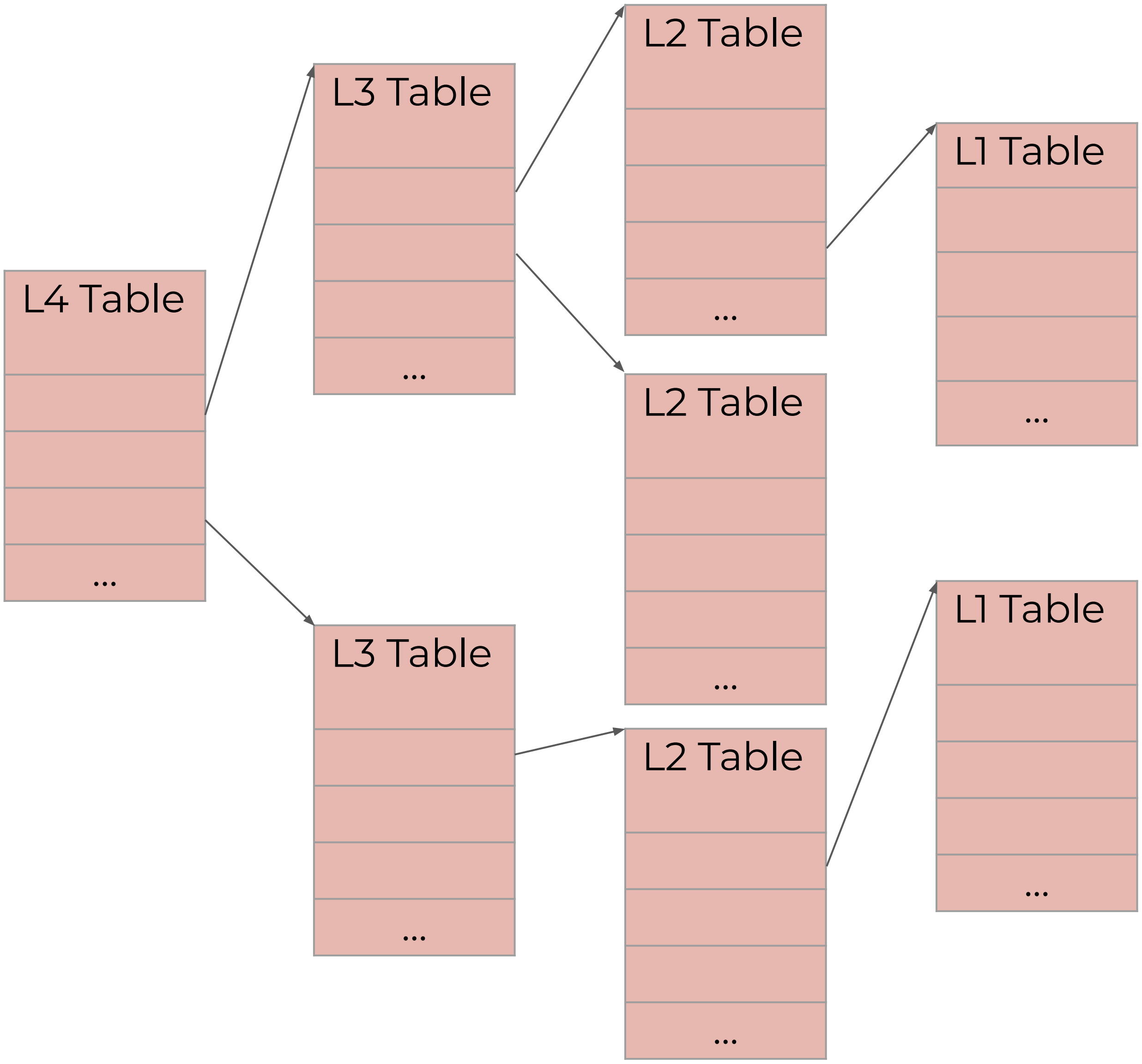


8 B	7 B	7 B		7 B	7 B
tag	PTE0	PTE1	. . .	PTE6	PTE7

Table 2: Compact cluster of PTEs.

- allows storing 8 PTEs in one cache line like for radix page tables

x86-64 Radix-based Page Table

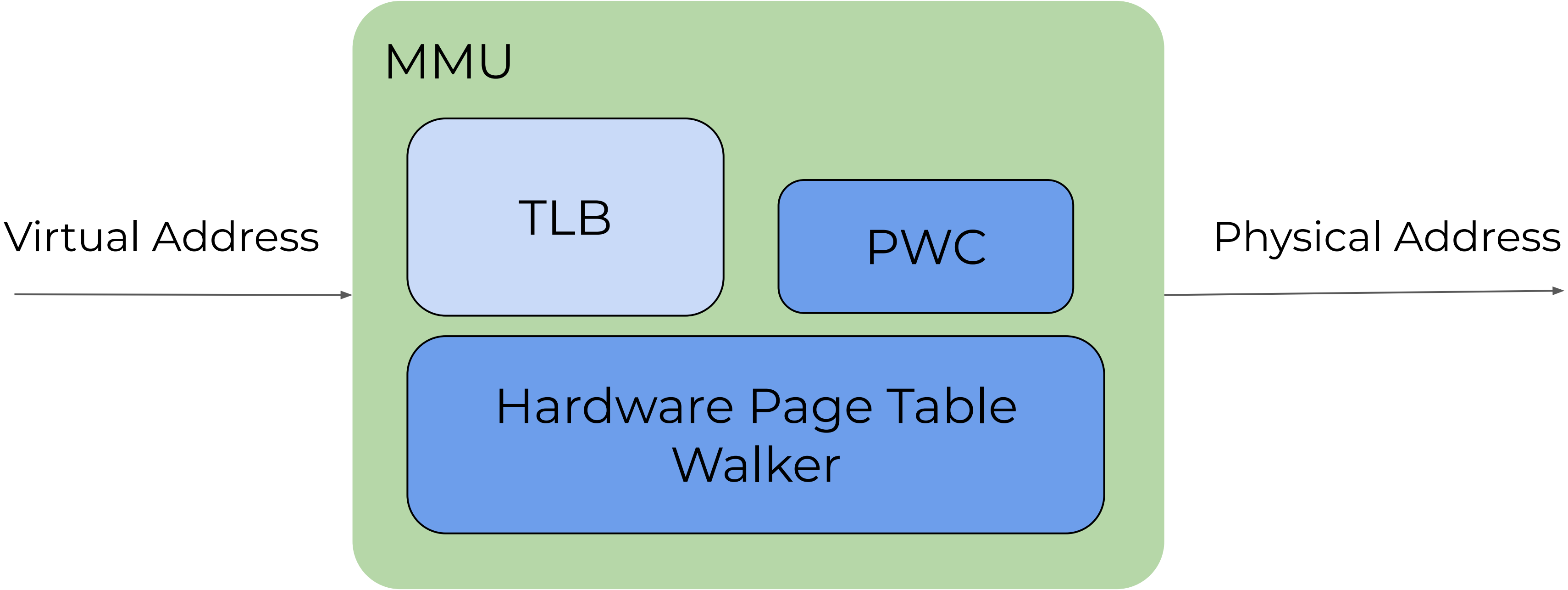


Measurements

- Model the runtime of a benchmark
- *`runtime = A * walk_cycles + B`*
- determine parameters A and B by running the program twice with different page granule size
    - measure time spent during page walks

Virtual Memory Basics

- In order to speed up the translation, current systems have hardware support – called the Memory Management Unit (MMU)


MMU
Virtual Address
TLB
PWC
Physical Address
Hardware Page Table Walker