Translation Caching: Skip, Don’t Walk (The Page Table)

Thomas W. Barr, Alan L. Cox, Scott Rixner
Rice Computer Architecture Group, Rice University
International Symposium on Computer Architecture, June 2010

Virtual Memory: An increasing challenge

- **Virtual memory**
  - Performance overhead 5-14% for “typical” applications. [Bhargava08]
  - 89% under virtualization! [Bhargava08]
    - Overhead comes primarily from referencing the in-memory page table
- **MMU Cache**
  - Dedicated cache to speed access to parts of page table

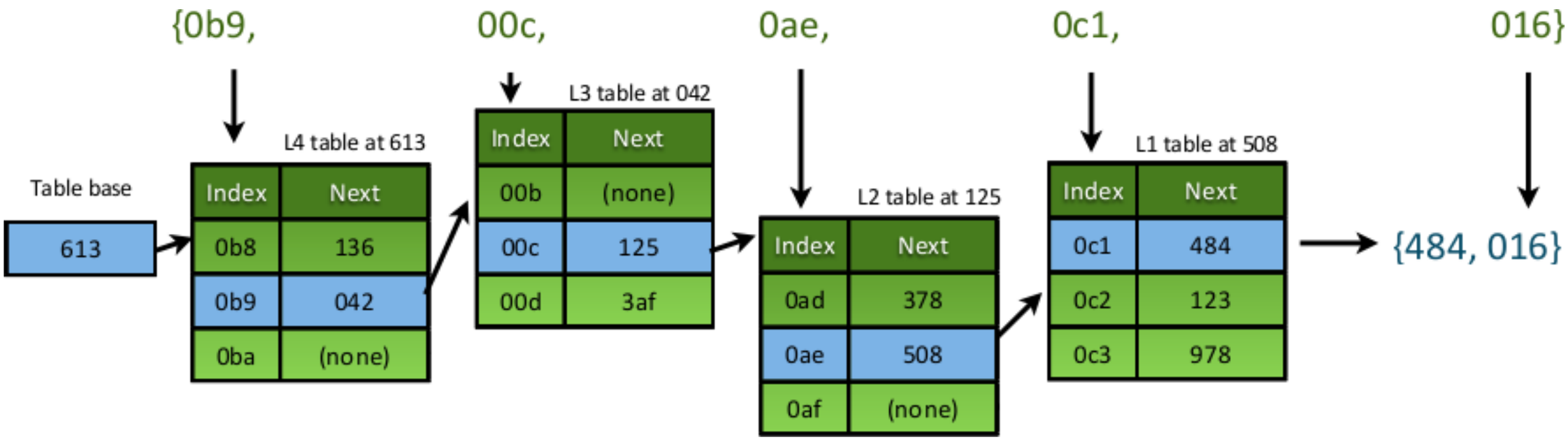
Overview

- **Background**
  - Why is address translation slow?
  - How MMU Caching can help
- **Design and comparison of MMU Caches**
  - Systematic exploration of design space
    - Previous designs
    - New, superior point in space
    - Novel replacement scheme
- **Revisiting previous work**
  - Comparison to Inverted Page Table

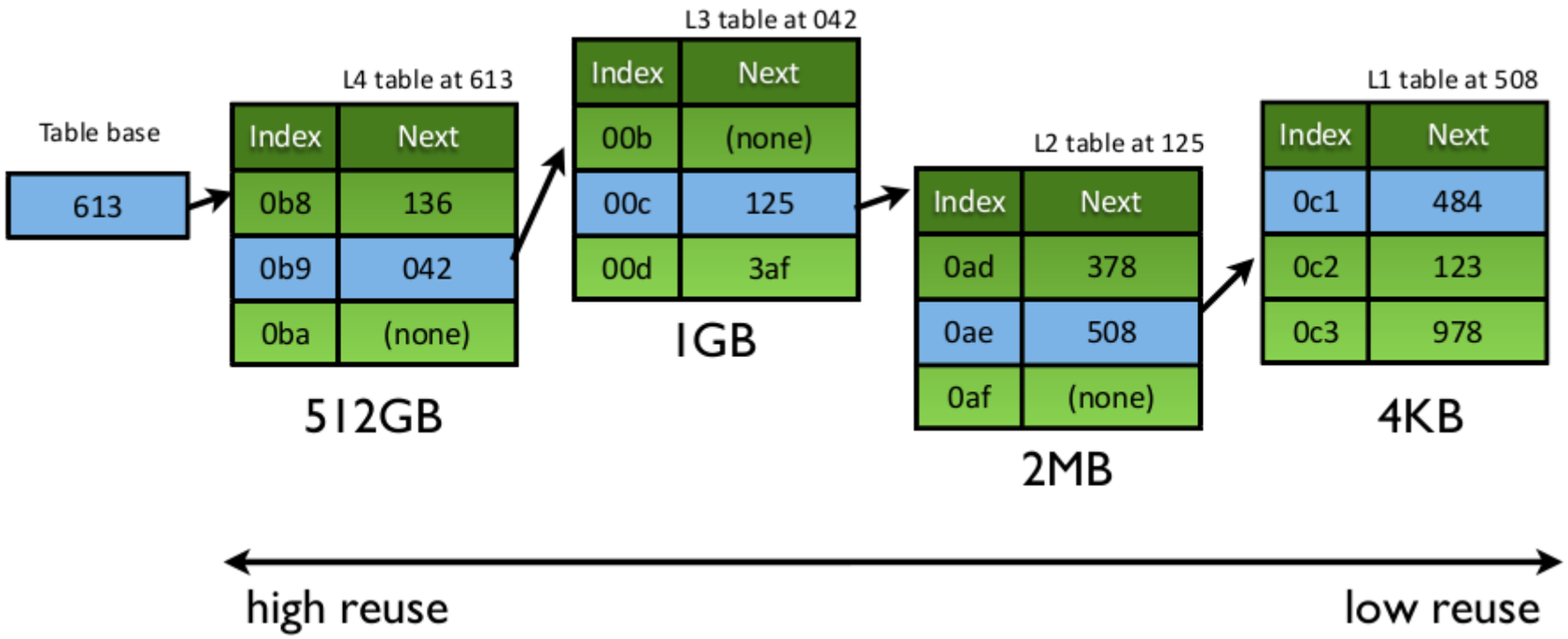
Why is Address Translation Slow?

- **Four-level Page Table**

0x5c8315cc1016



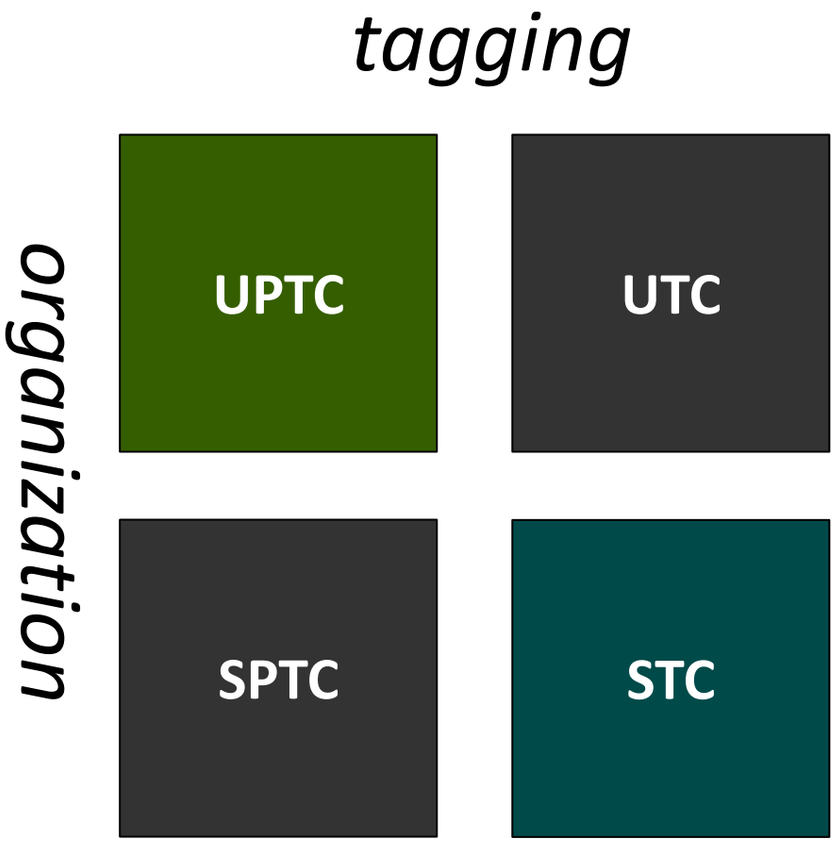
MMU Caching



- **Upper levels of page table correspond to large regions of virtual memory**
  - Should be easily cached
  - MMU does not have access to L1 cache
  - MMU Cache: Caches upper level entries (L4, L3, L2)

MMU Caching

- **In production**
  - AMD and Intel
- **Design space**
  - Tagging
    - *Page table/Translation*
  - Organization
    - *Split/Unified*
- **Previous designs not optimal**
  - Unified translation cache (with modified replacement scheme) outperforms existing devices



Page table caches

{0b9, 00c, 0ae, 0c1, 016}

PTE address	pointer
0x23410	0xabcde
0x55320	0x23144
0x23144	0x55320
…	…
…	…
…	…

- **Simple design**
  - Data cache
  - Entries tagged by physical address of page table entry
- **Page walk unchanged**
  - Replace memory accesses with MMU cache accesses
  - Three accesses/walk

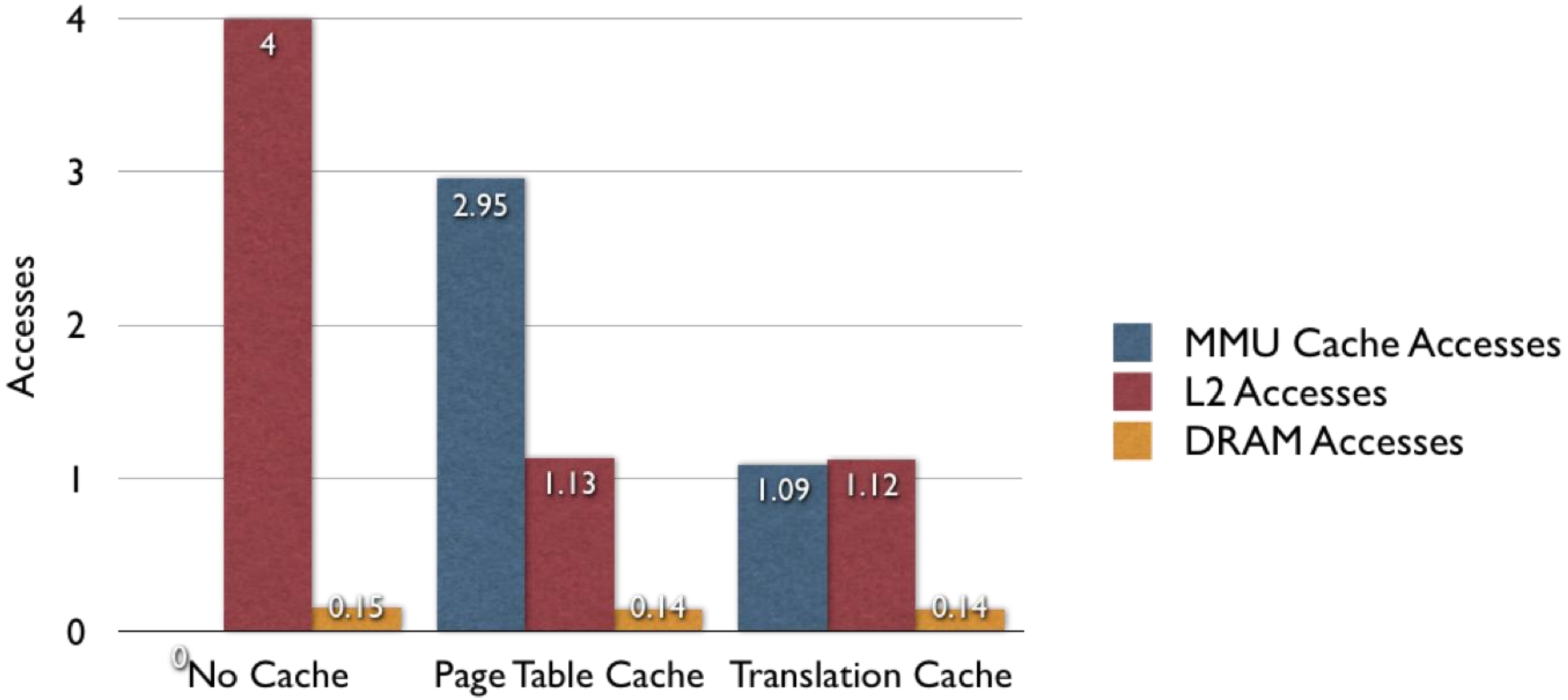
Translation caches

{0b9, 00c, 0ae, 0c1, 016}

(L4, L3, L2 indices)	pointer
(0b9, 00c, 0ae)	0xabcde
(0b9, 00c, xxx)	0x23410
(0b9, xxx, xxx)	0x55320
…	…
…	…
…	…

- **Alternate tag**
  - Tag by virtual address fragment
- **Smaller**
  - 27 bits vs. 49 bits
- **Skip parts of page walk**
  - Skip to bottom of tree

Cache tagging comparison

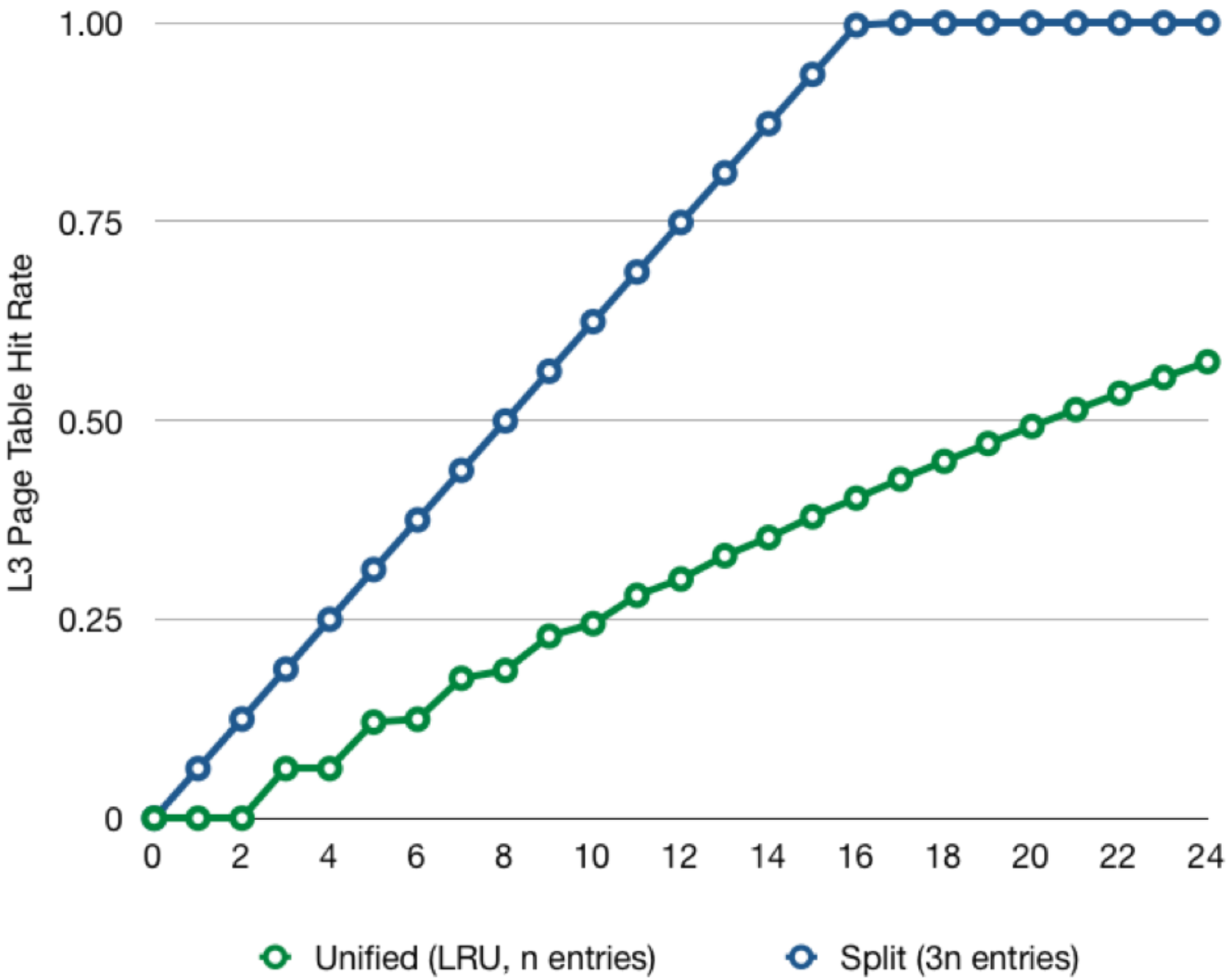


SPEC CPU2006 Floating Point Suite

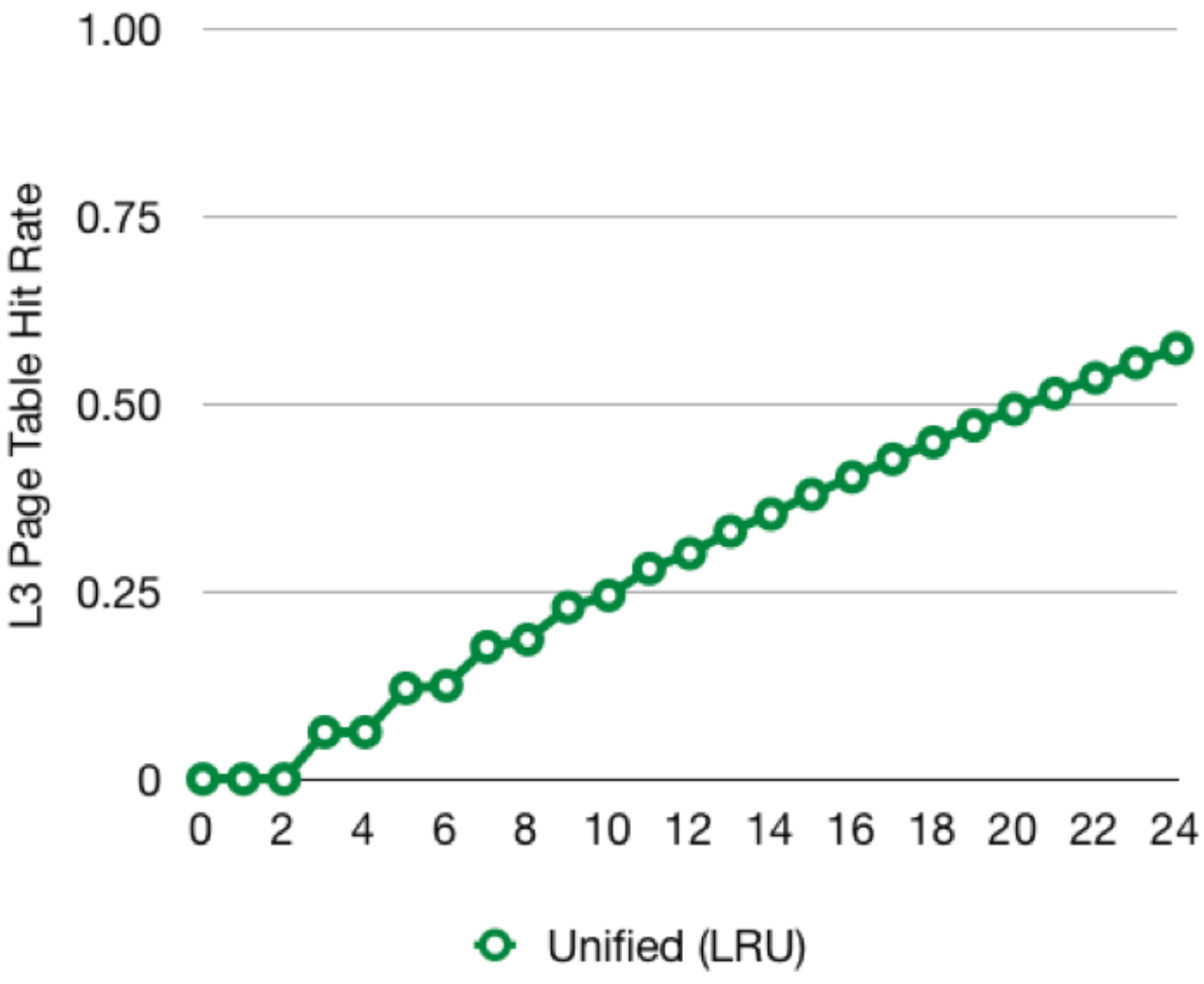
Split vs. Unified Caches

- **Hash joins**
  - Reads to many gigabyte table nearly completely random
  - Vital to overall DMBS performance [Aliamaki99]
  - Simulate with synthetic trace generator
- **MMU cache performance**
  - 16 gigabyte hash table
    - 1 L4 entry
    - 16 L3 entries
    - 8,192 L2 entries
  - Low L2 hit rate leads to “level conflict” in unified caches
    - Solve by splitting caches or using a smarter replacement scheme

Split vs. Unified Caches



Level Conflict in Unified Caches

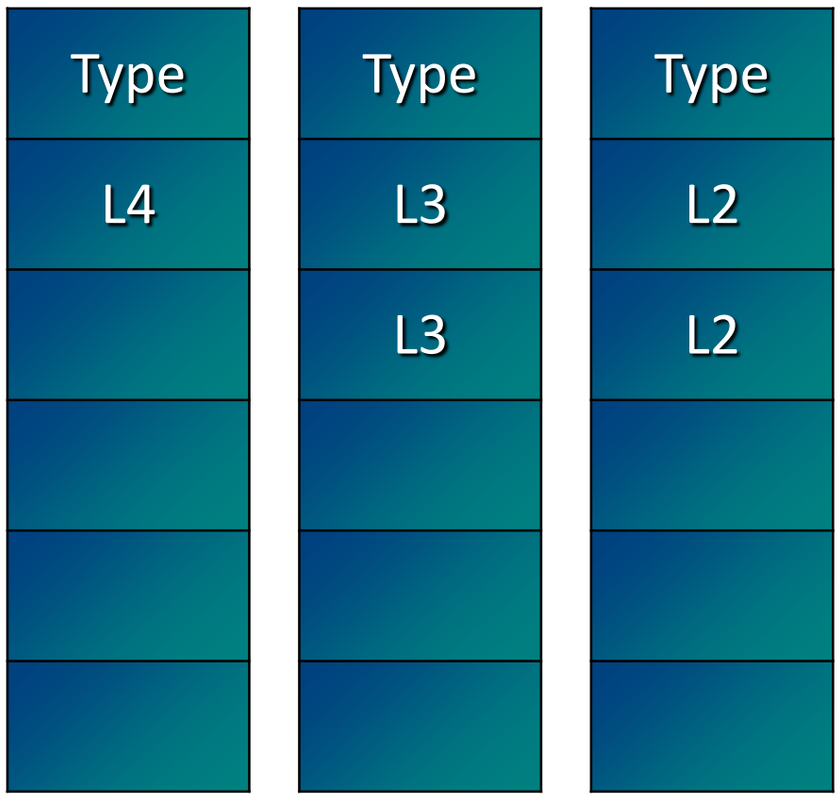


- **LRU replacement**
  - Important for high-locality applications
    - Avoid replacing upper level entries
  - After every L3 access, must be one L2 access
  - Each L3 entry pollutes the cache with at least one unique L2 entry

Split vs. Unified Caches

- **Split caches**
  - Split caches have one cache per level
  - Protects entries from upper levels
  - Intel's *Paging Structure Cache*

Split vs. Unified Caches

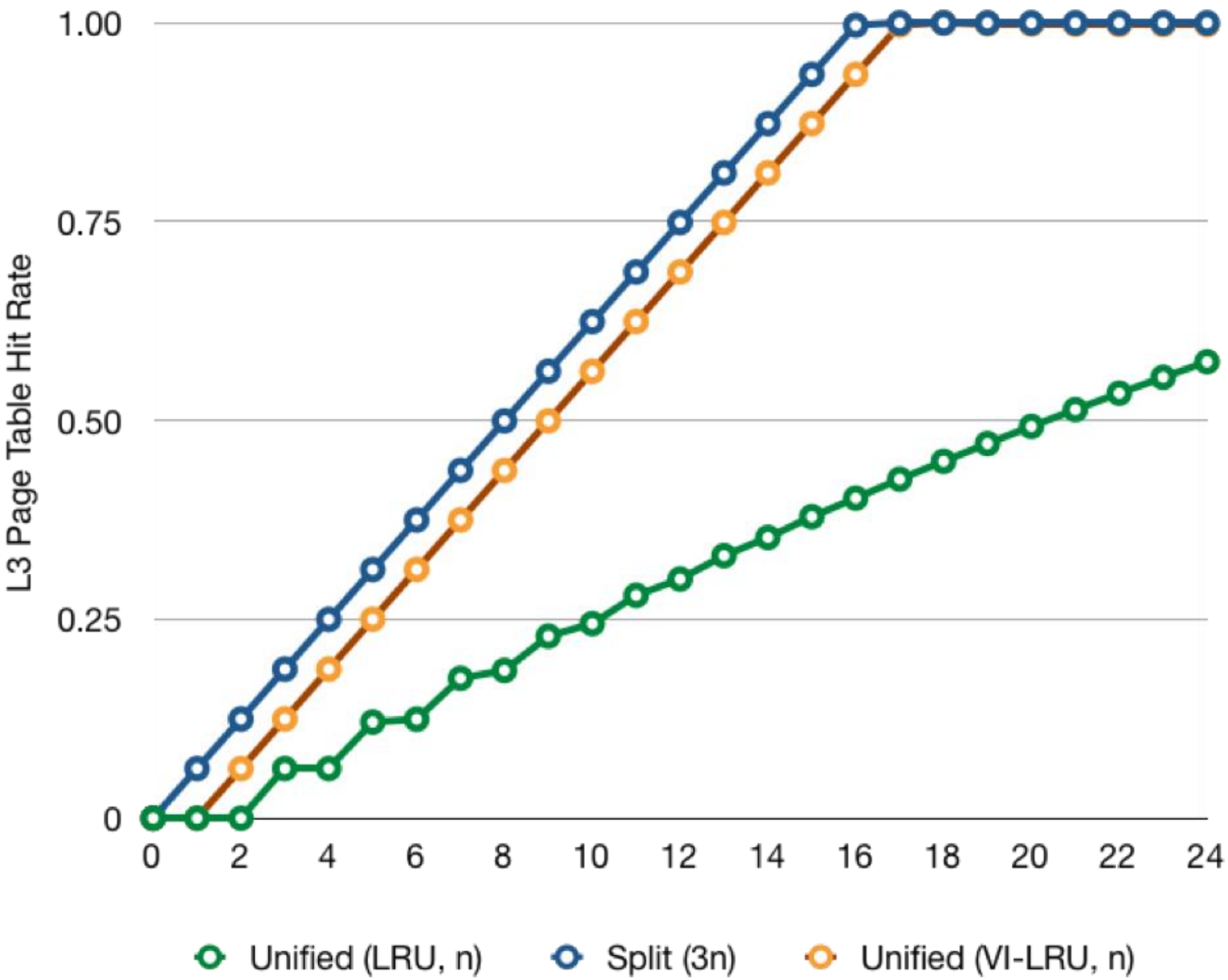


- **Problem: Size allocation**
  - Each level large?
    - Die area
  - Each level small?
    - Hurts performance for all applications
- Unequal distribution?
  - Hurts performance for particular applications

Variable insertion point LRU replacement

- **Modified LRU**
  - Preserve entries with low reuse for less time
  - Insert them below the MRU slot
- **VI-LRU**
  - Novel scheme
  - Vary insertion point based on content of cache
  - If L3 entries have high reuse, give L2 entries less time

Variable insertion point LRU replacement



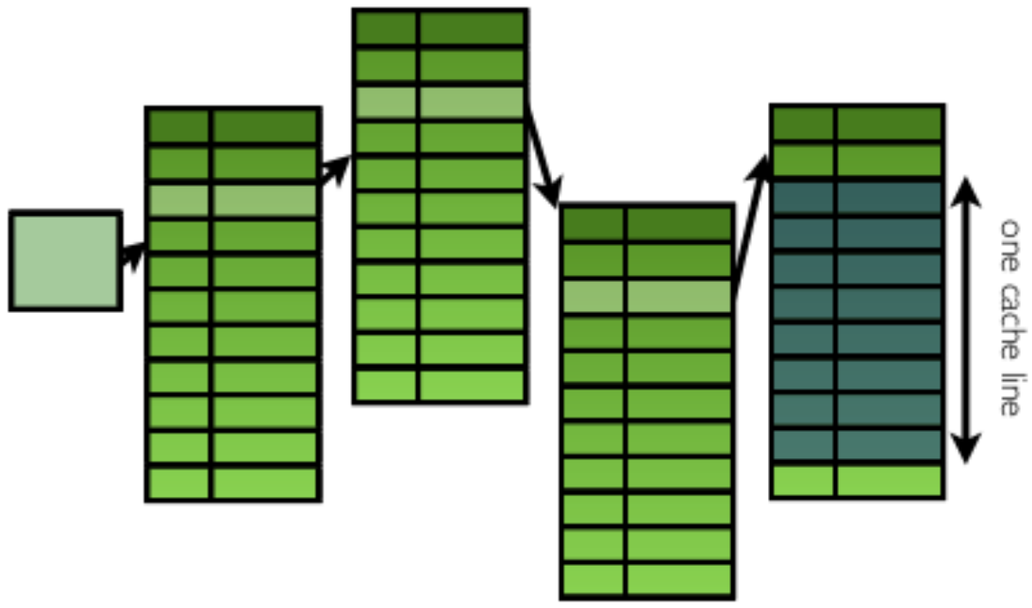
Page Table Formats

- **In the past, radix table implementations required four memory references per TLB miss**
  - Many proposed data structure solutions to replace format
    - Reduces memory references/miss
- **This situation has changed**
  - MMU cache is a hardware solution
    - Also reduces memory references
  - Revisit previous work
    - Competing formats are not as attractive now

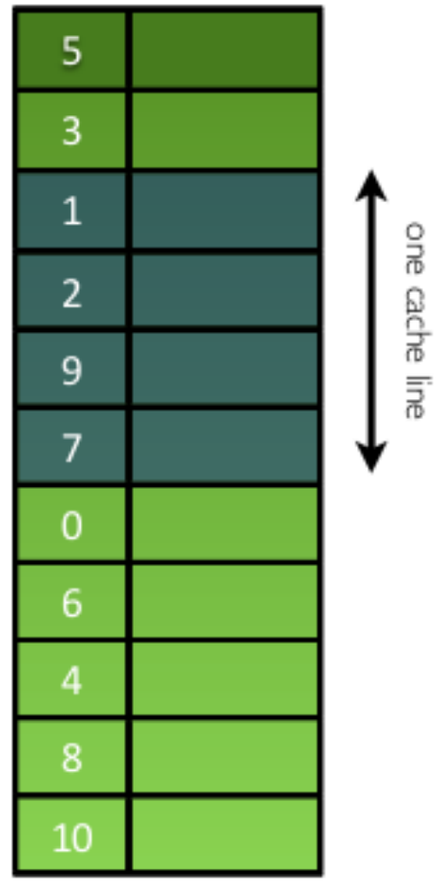
Inverted page table

- **Inverted (hashed) page table**
  - Flat table, regardless of key (virtual address) size
  - Best case lookup is one
    - Average increases as hash collisions occur
      - 1.2 accesses / lookup for half full table [Knuth98]
- **Radix vs. inverted page table**
  - IPT poorly exploits spatial locality in processor data cache
  - Increases DRAM accesses/walk by *400%* for SPEC in simulation

Inverted page table



radix table



hashed page table

Inverted page table

- **IPT compared to *cached* radix table**
  - Number of memory accesses similar (≈1.2)
  - Number of DRAM accesses increased 4x
- **SPARC TSB, Clustered Page Table, etc.**
  - Similar results
    - Caching makes performance proportional to size
      - Translations / L2 cache
      - Consecutive translations / cache line
- **New hardware changes old “truths”**
  - Replace complex data structures with simple hardware

Conclusions

- **Address translation will continue to be a problem**
  - Up to 89% performance overhead
- **First design space taxonomy and evaluation of MMU caches**
  - Two-dimension space
    - Translation/Page Table Cache
    - Split/Unified Cache
  - 4.0 → 1.13 L2 accesses/TLB miss for current design
- **Existing designs are not ideal**
  - Tagging
    - Translation caches can skip levels, use smaller tags
  - Partitioning
    - Novel VI-LRU allows partitioning to adapt to workload