ME-HPTs: Memory-Efficient Hashed Page Tables

HPCA 2023

Jovan Stojkovic, Namrata Mantri, Dimitrios Skarlatos*, Tianyin Xu, Josep Torrellas

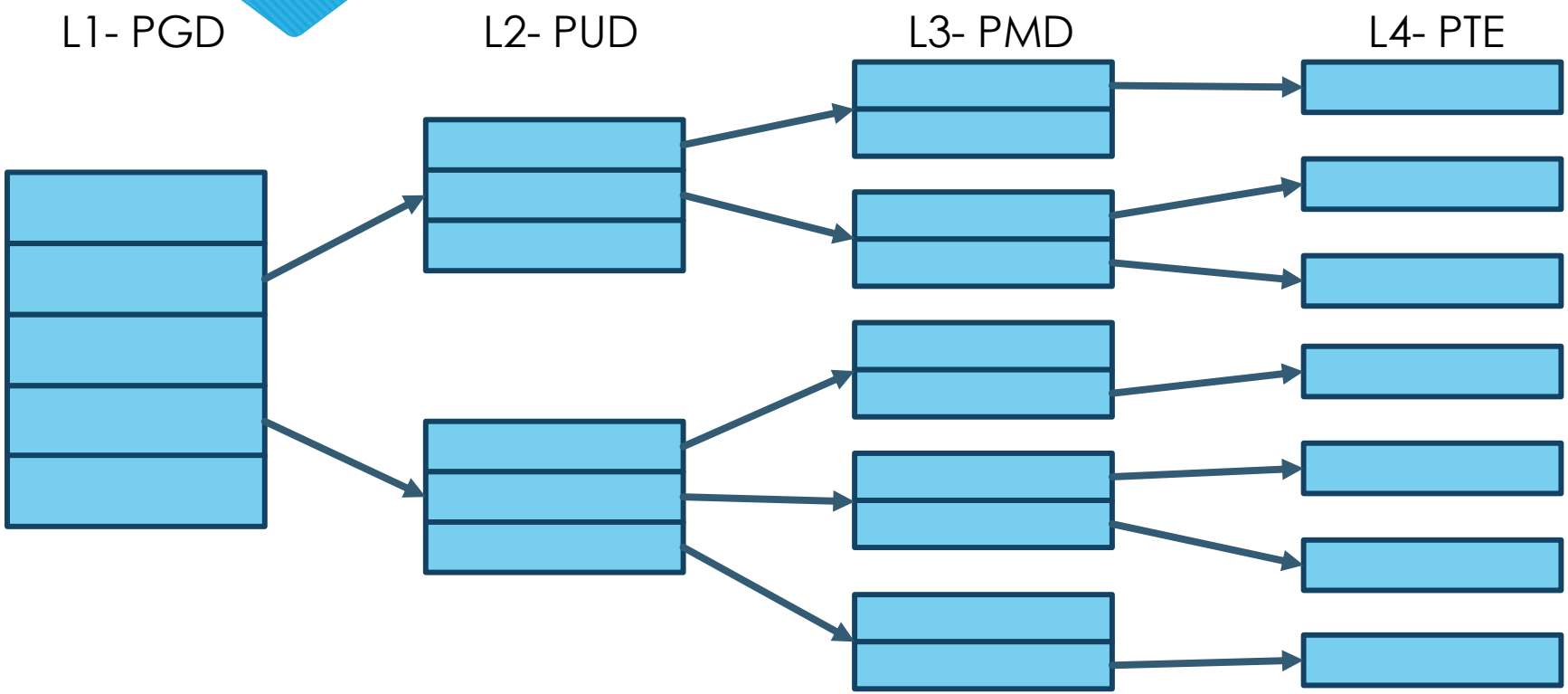
University of Illinois at Urbana-Champaign

*Carnegie Mellon University

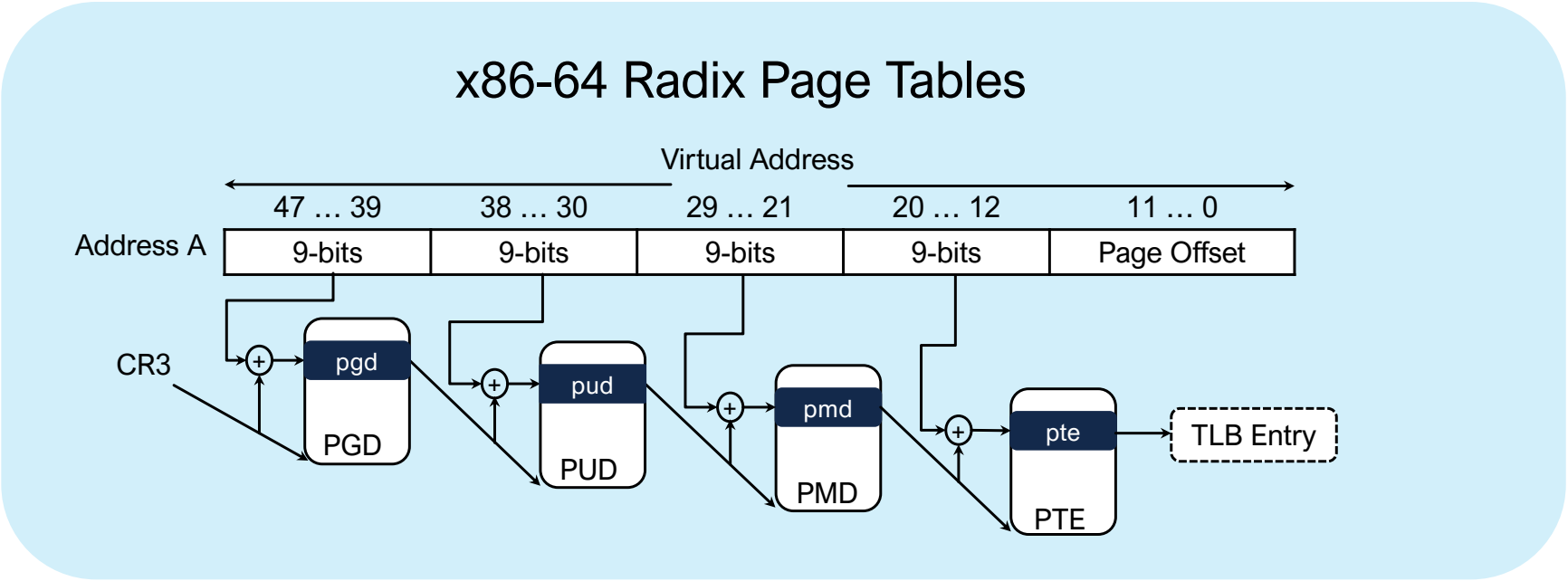
Virtual Memory and Page Tables

- Virtual memory is an essential technique in modern computing systems
  - Memory virtualization
  - Process isolation
- Virtual memory performance depends on the page table organization
  - Radix page tables – slow and not scalable
  - Hashed page tables – memory inefficient

Radix Page Tables: Memory-Efficient Multi-Level Trees

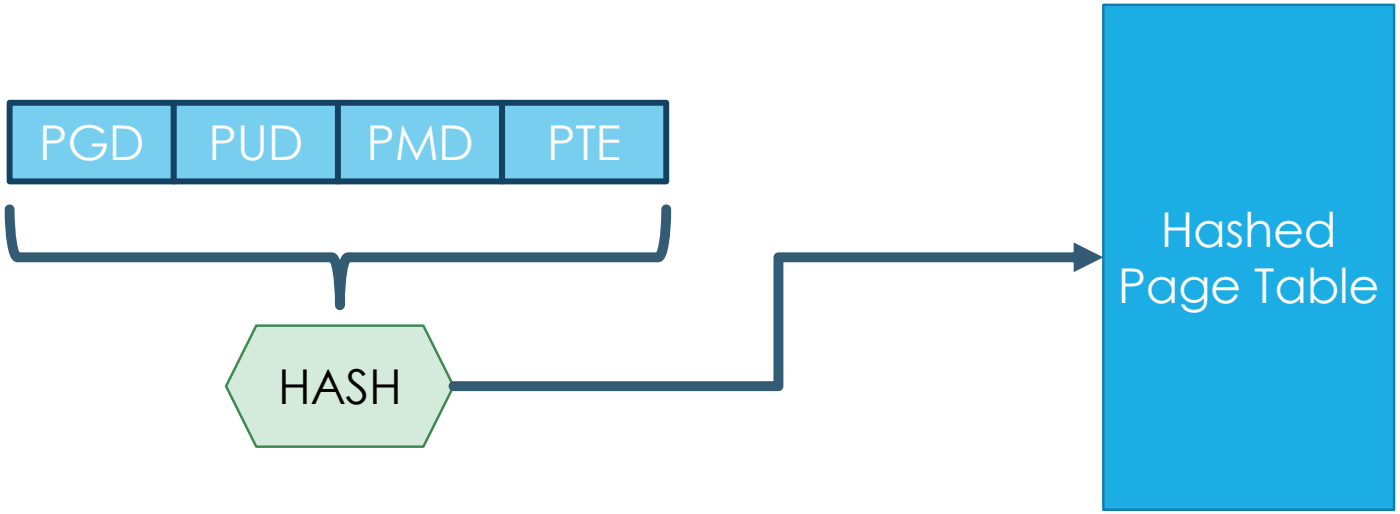


Radix Page Walk: Expensive Pointer Chase



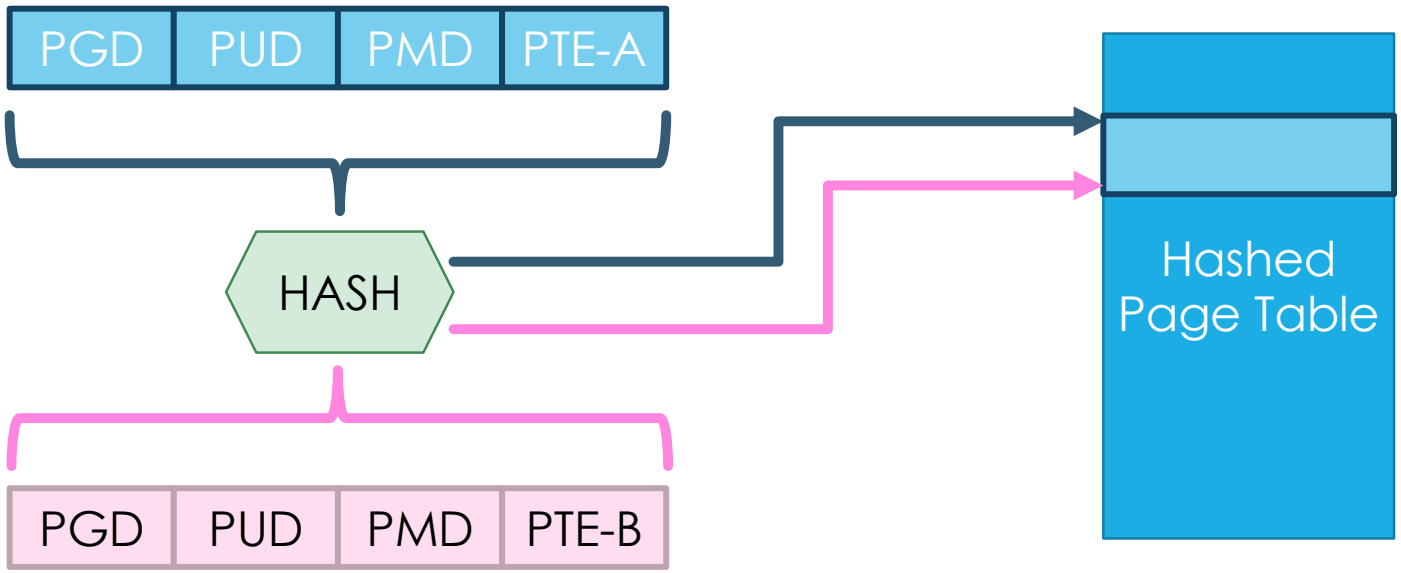
Hashed Page Tables

Page walk requires a single memory access



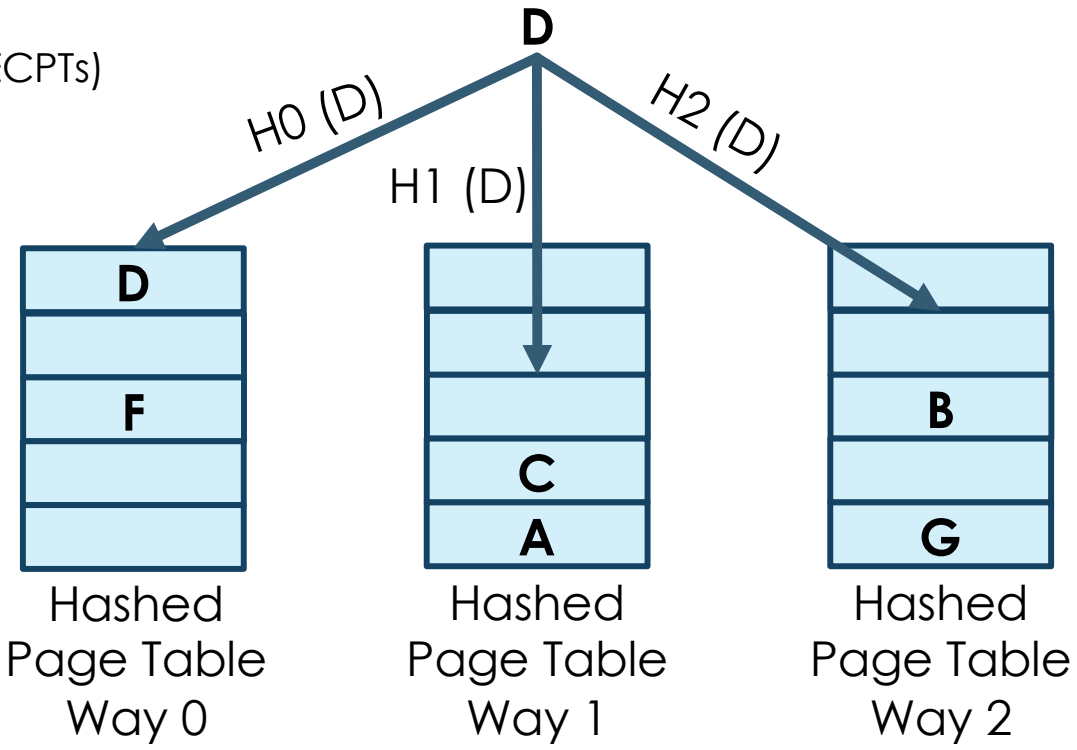
Hashed Page Tables

Hash collisions



Hashed Page Tables: Recent Advances Make Them Compelling

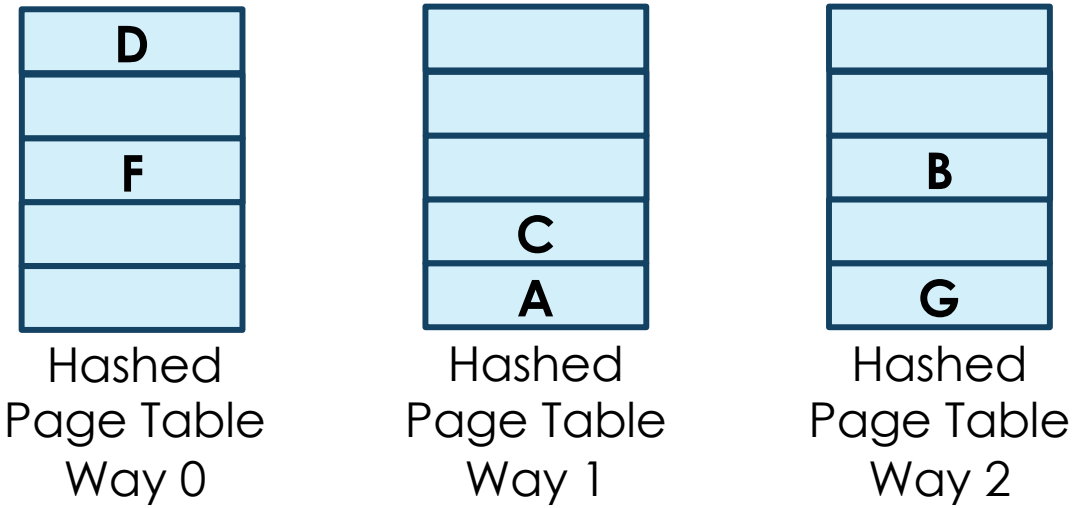
Elastic Cuckoo Page Tables (ECPTs)



Hashed Page Tables: Recent Advances Make Them Compelling

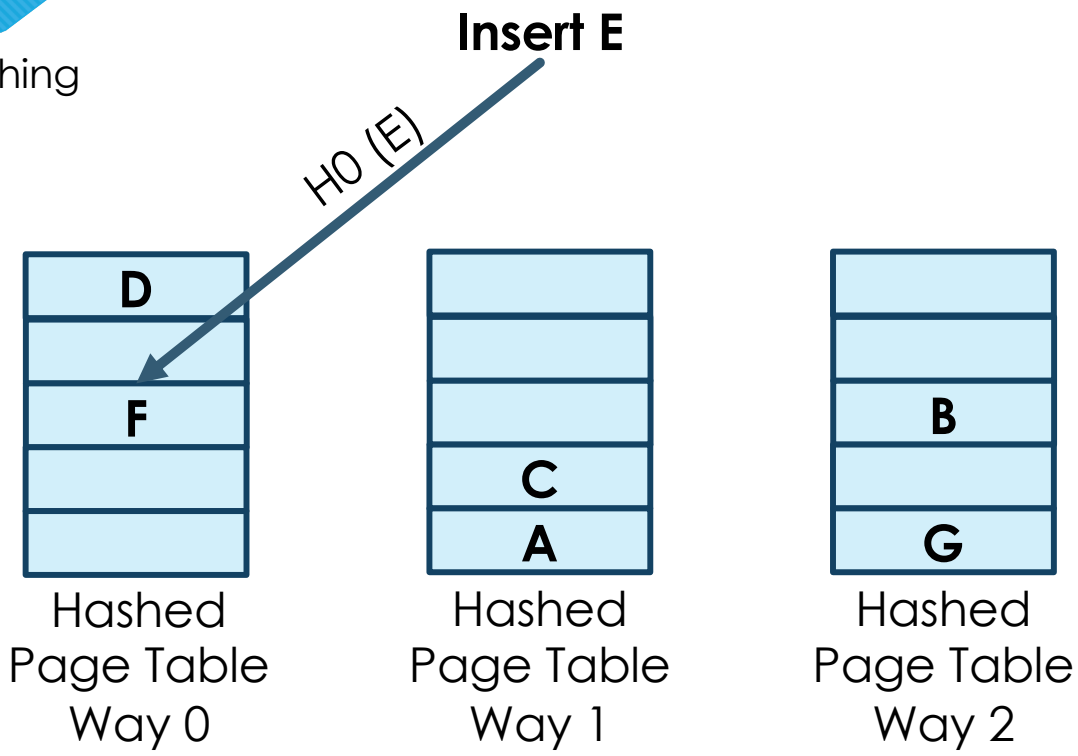
Cuckoo Hashing

Insert E



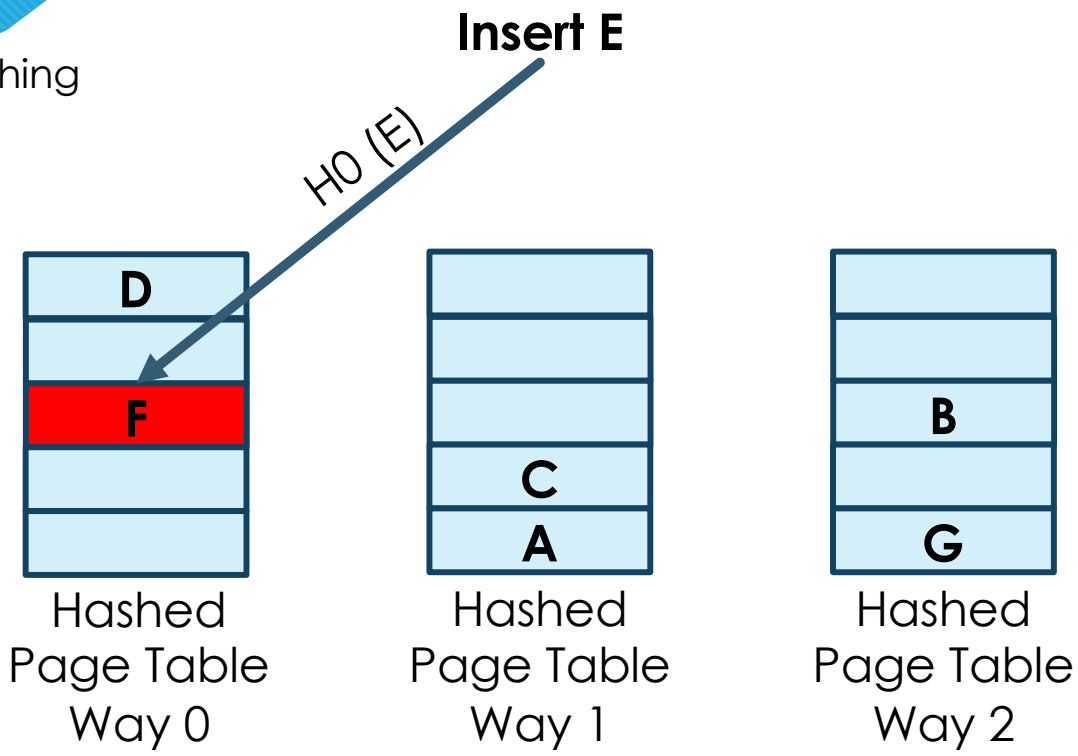
Hashed Page Tables: Recent Advances Make Them Compelling

Cuckoo Hashing



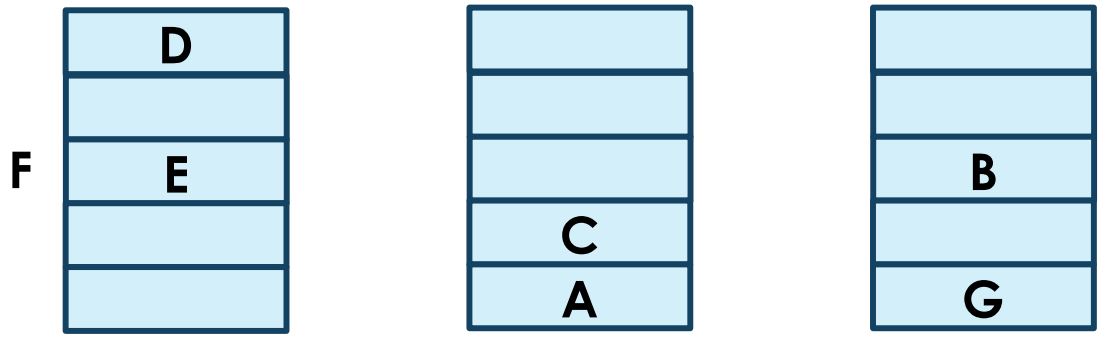
Hashed Page Tables: Recent Advances Make Them Compelling

Cuckoo Hashing



Hashed Page Tables: Recent Advances Make Them Compelling

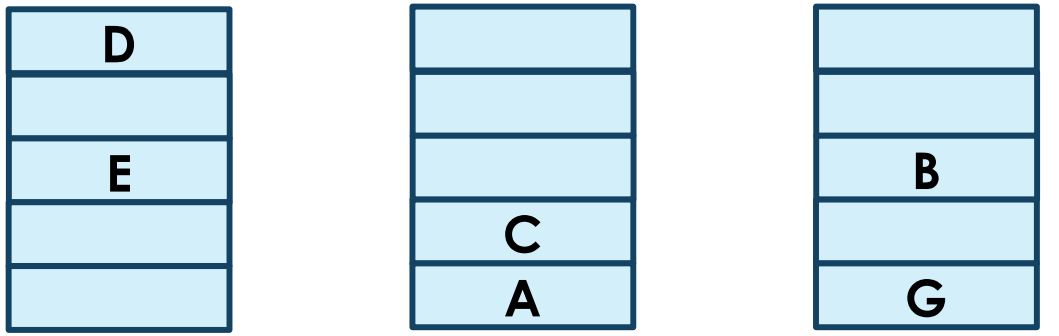
Cuckoo Hashing



Hashed Page Tables: Recent Advances Make Them Compelling

Cuckoo Hashing

Insert F



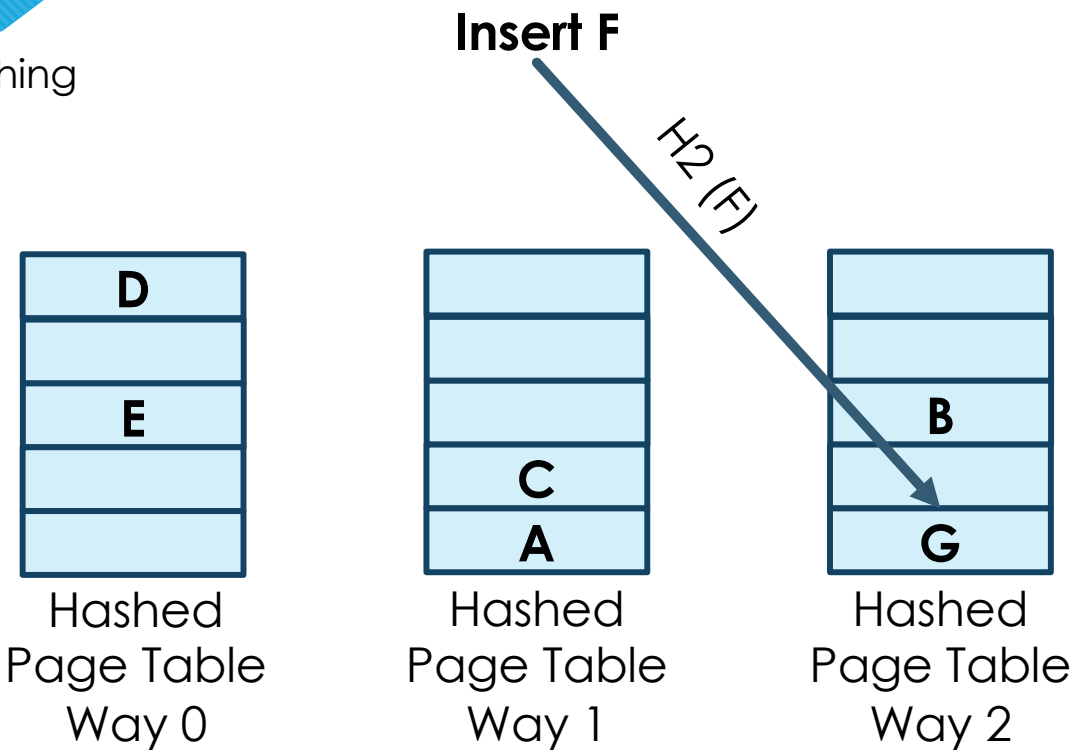
Hashed Page Table Way 0

Hashed Page Table Way 1

Hashed Page Table Way 2

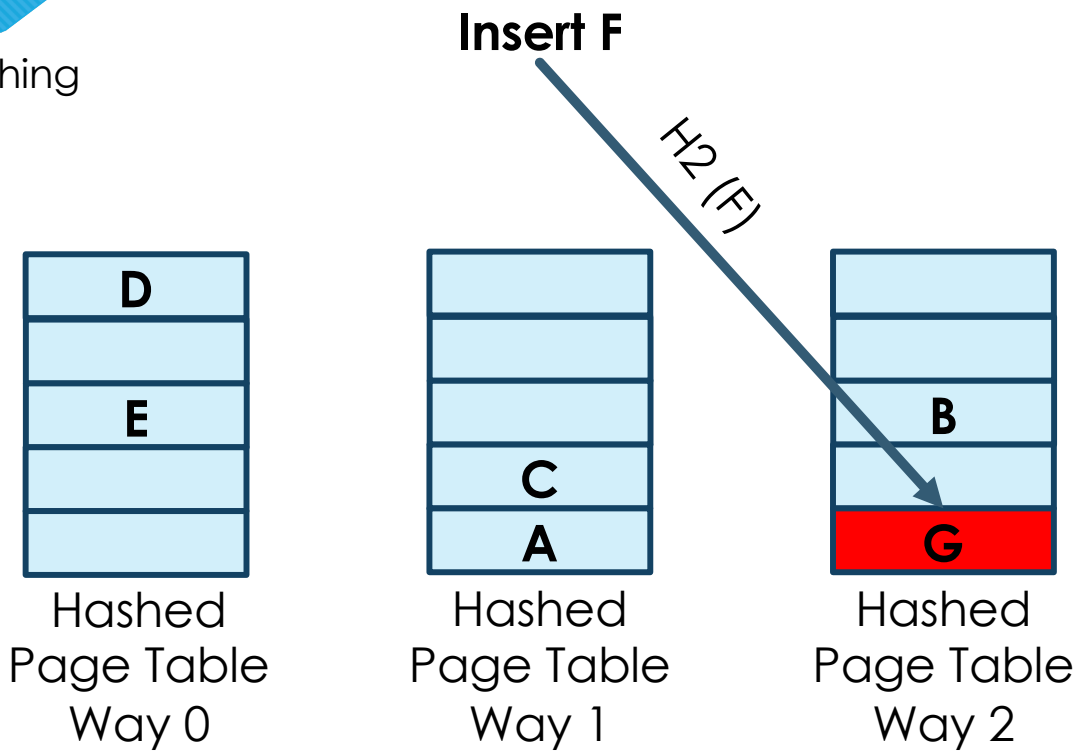
Hashed Page Tables: Recent Advances Make Them Compelling

Cuckoo Hashing



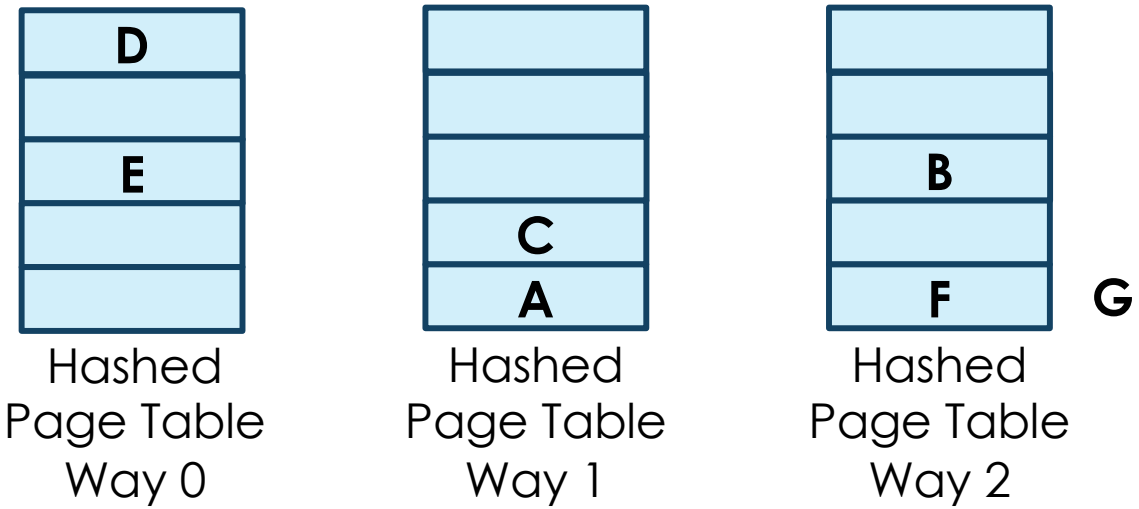
Hashed Page Tables: Recent Advances Make Them Compelling

Cuckoo Hashing



Hashed Page Tables: Recent Advances Make Them Compelling

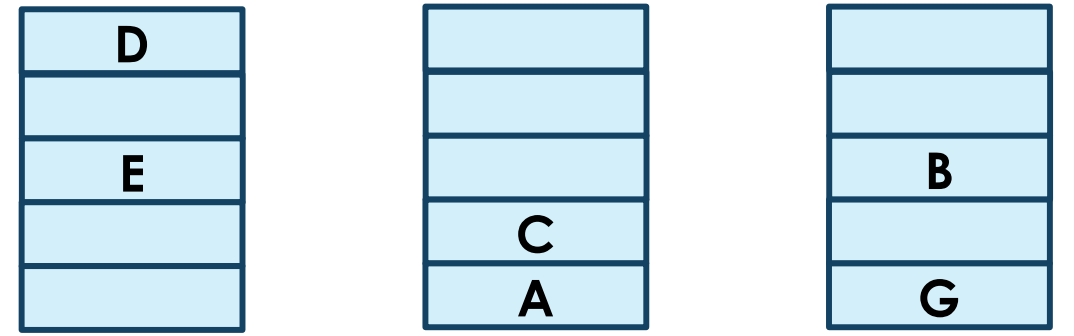
Cuckoo Hashing



Hashed Page Tables: Recent Advances Make Them Compelling

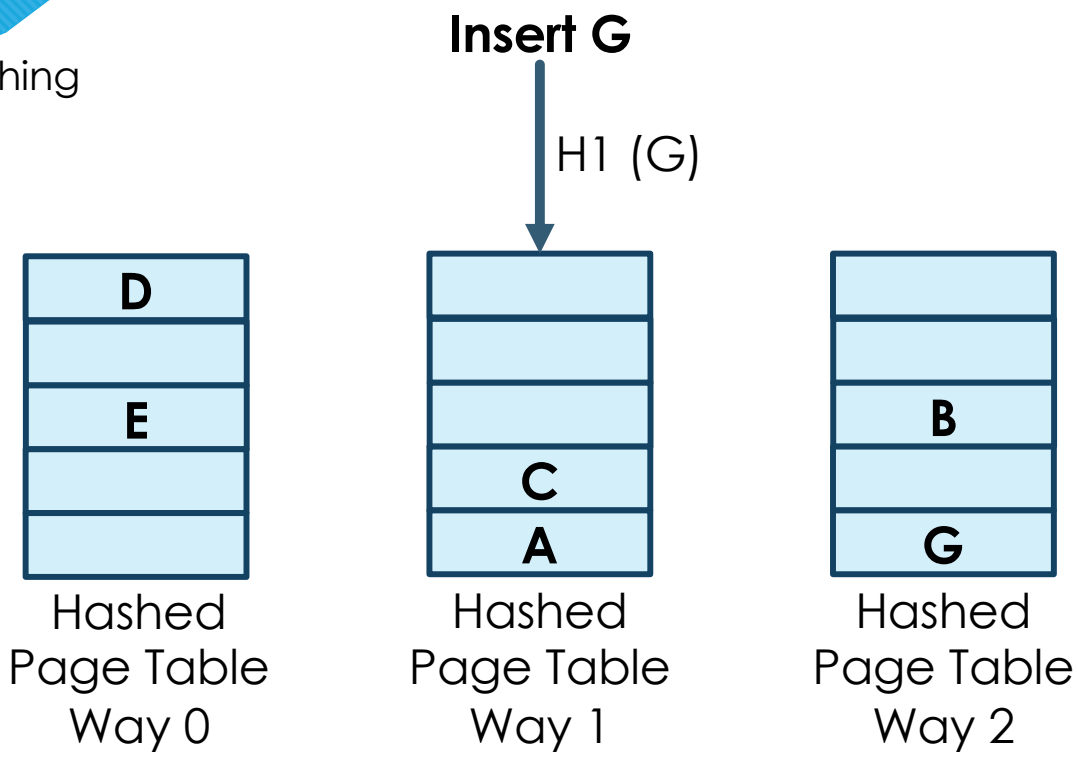
Cuckoo Hashing

Insert G



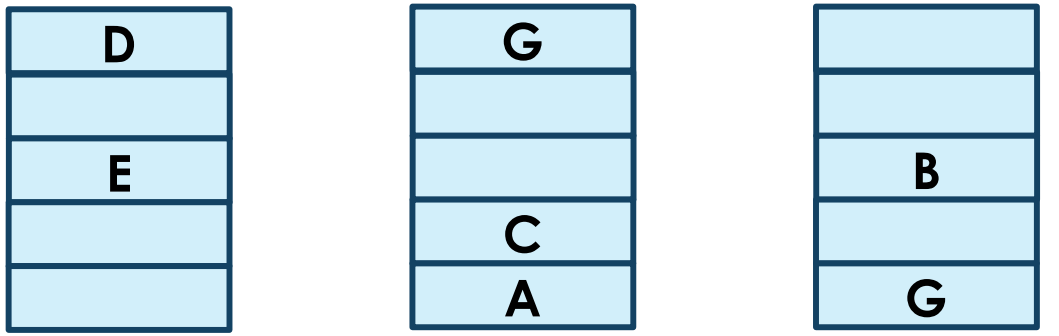
Hashed Page Tables: Recent Advances Make Them Compelling

Cuckoo Hashing



Hashed Page Tables: Recent Advances Make Them Compelling

Cuckoo Hashing

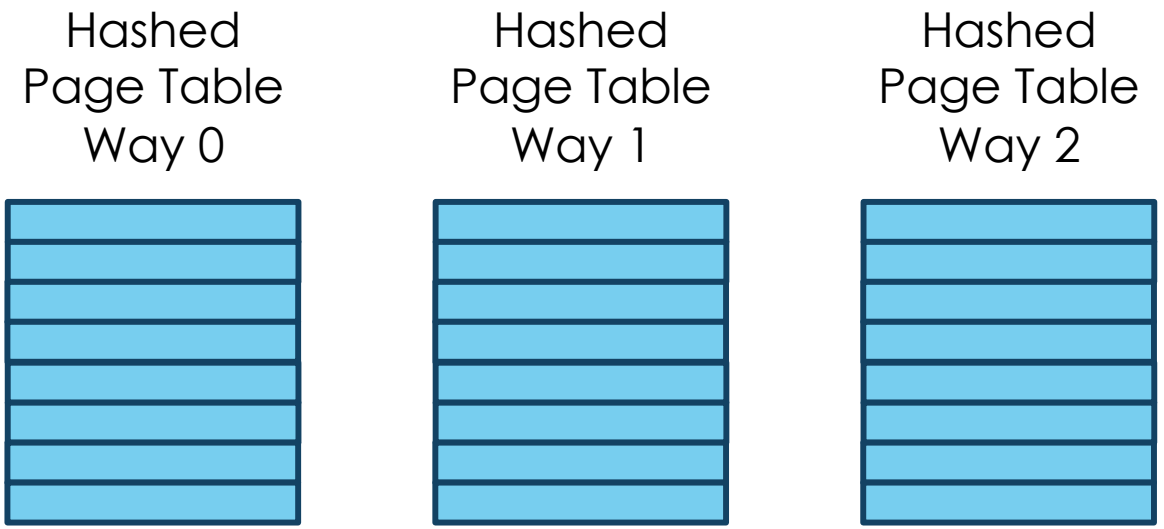


Outline of this talk

- **Problem: Contiguous Memory Requirements of Hashed Page Tables**
- ME-HPTs: Memory-Efficient Hashed Page Tables
  - ME-HPTs Design
  - ME-HPTs Key Results
- Conclusion

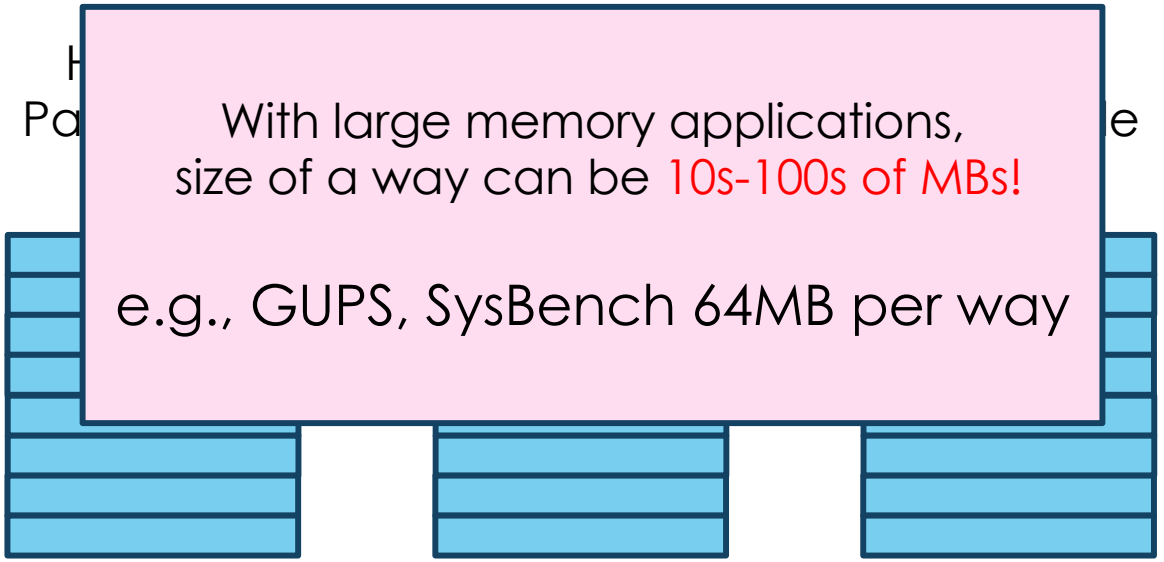
Hashed Page Tables: Large Contiguous Memory Chunks

- With hashed page tables – unity of allocation is one way of the page table



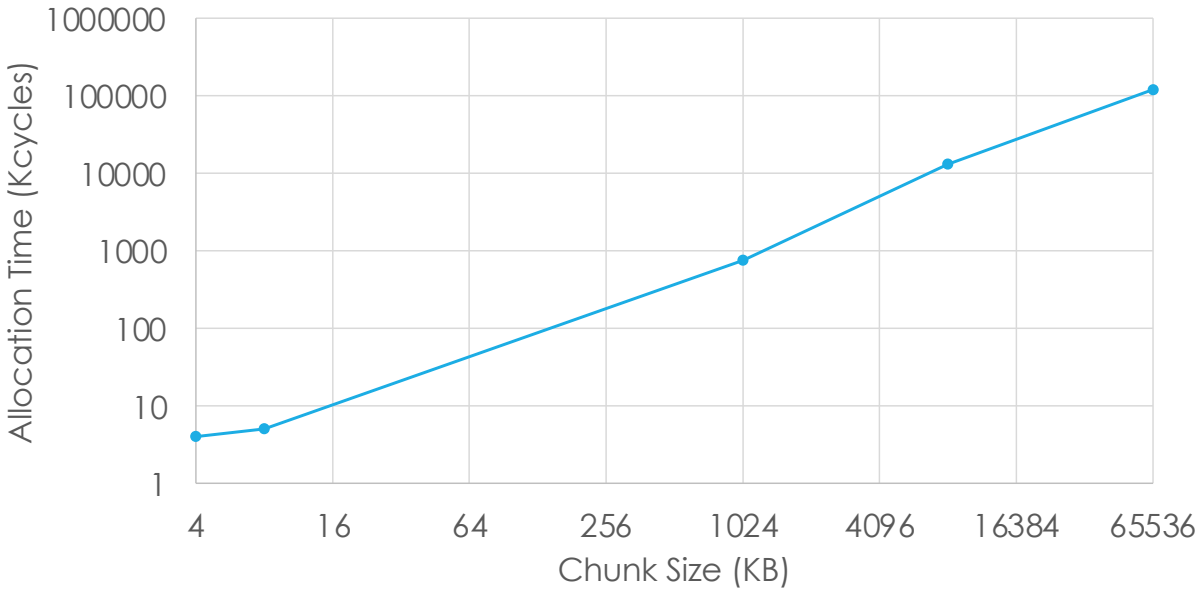
Hashed Page Tables: Large Contiguous Memory Chunks

- With hashed page tables – unity of allocation is one way of the page table



Hashed Page Tables: Contiguity is Expensive!

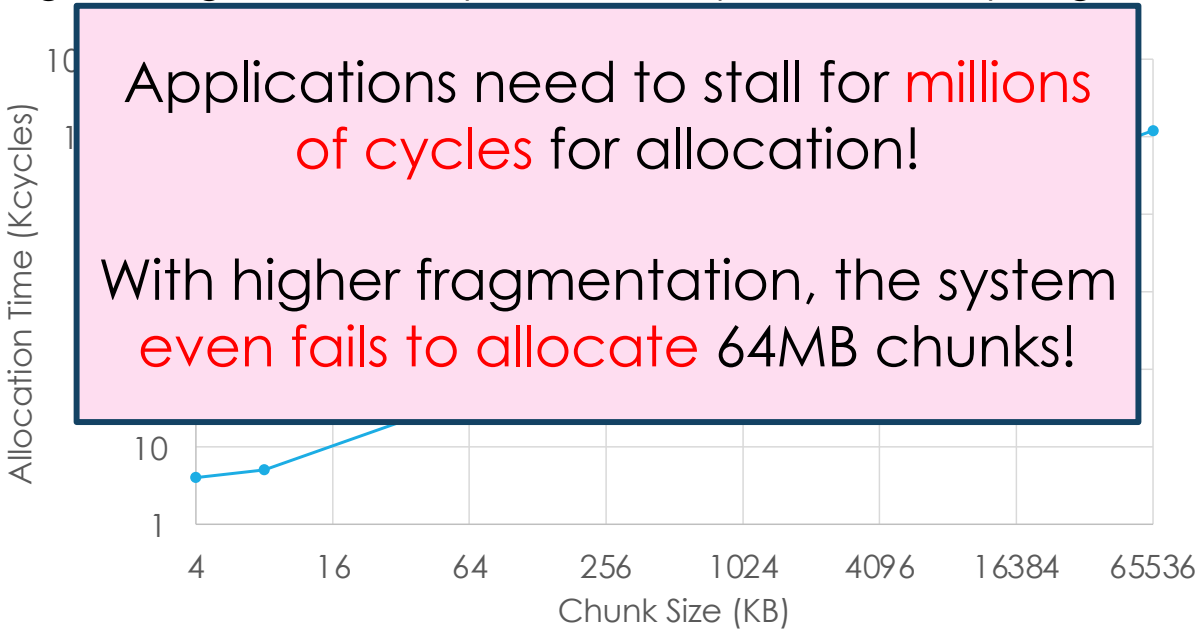
- Finding large contiguous memory chunks is expensive in busy fragmented servers



Linux server
2GHz
0.7 FMFI

Hashed Page Tables: Contiguity is Expensive!

- Finding large contiguous memory chunks is expensive in busy fragmented servers



Linux server
2GHz
0.7 FMFI

Contributions

- Four novel architectural techniques to provide Memory-Efficient Hashed Page Tables (**ME-HPTs**)
- Reduced memory contiguity requirement by 92%
- Sped-up applications by 9% on average
- Allow large-memory applications to run at high performance on highly fragmented servers

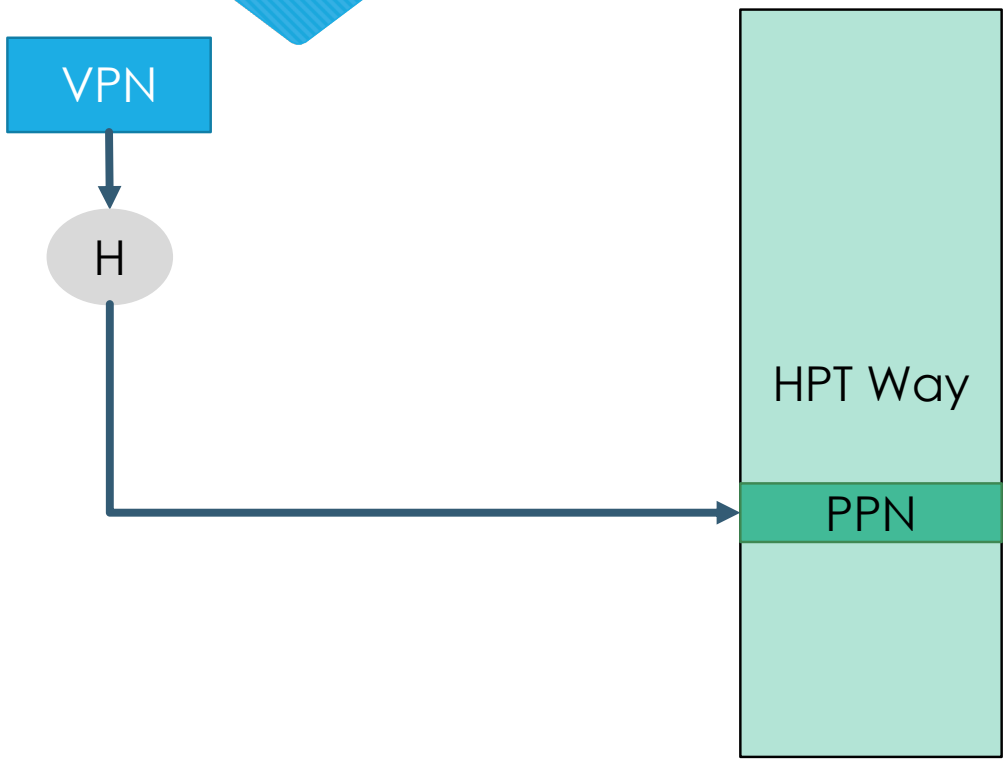
Outline of this talk

- Problem: Contiguous Memory Requirements of Hashed Page Tables
- **ME-HPTs: Memory-Efficient Hashed Page Tables**
  - ME-HPTs Design
  - ME-HPTs Key Results
- Conclusion

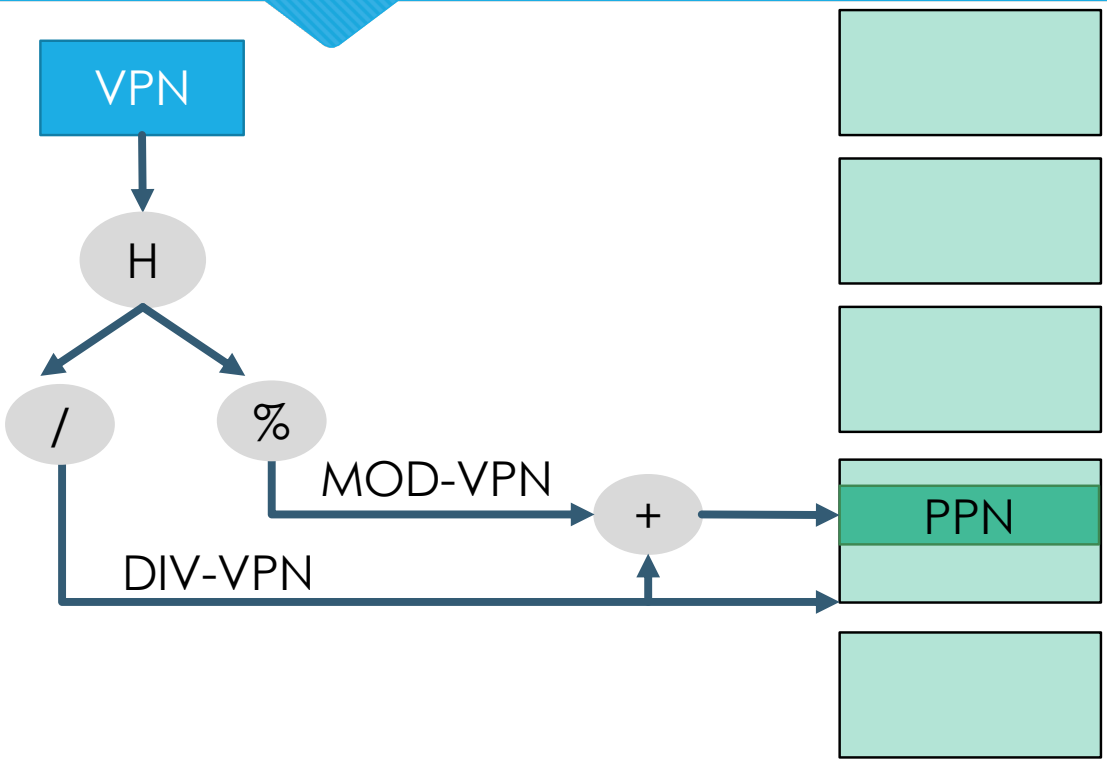
Memory-Efficient Hashed Page Tables: ME-HPTs Design Overview

- Memory-Efficient Hashed Page Tables (ME-HPTs): Four novel architectural techniques
- Directly minimizing contiguity requirements
  - Logical-to-Physical (L2P) Table
  - Dynamically Changing Chunk Size
- Indirectly minimizing contiguity requirements by minimizing memory consumption
  - In-place Page Table Resizing
  - Per-way Page Table Resizing

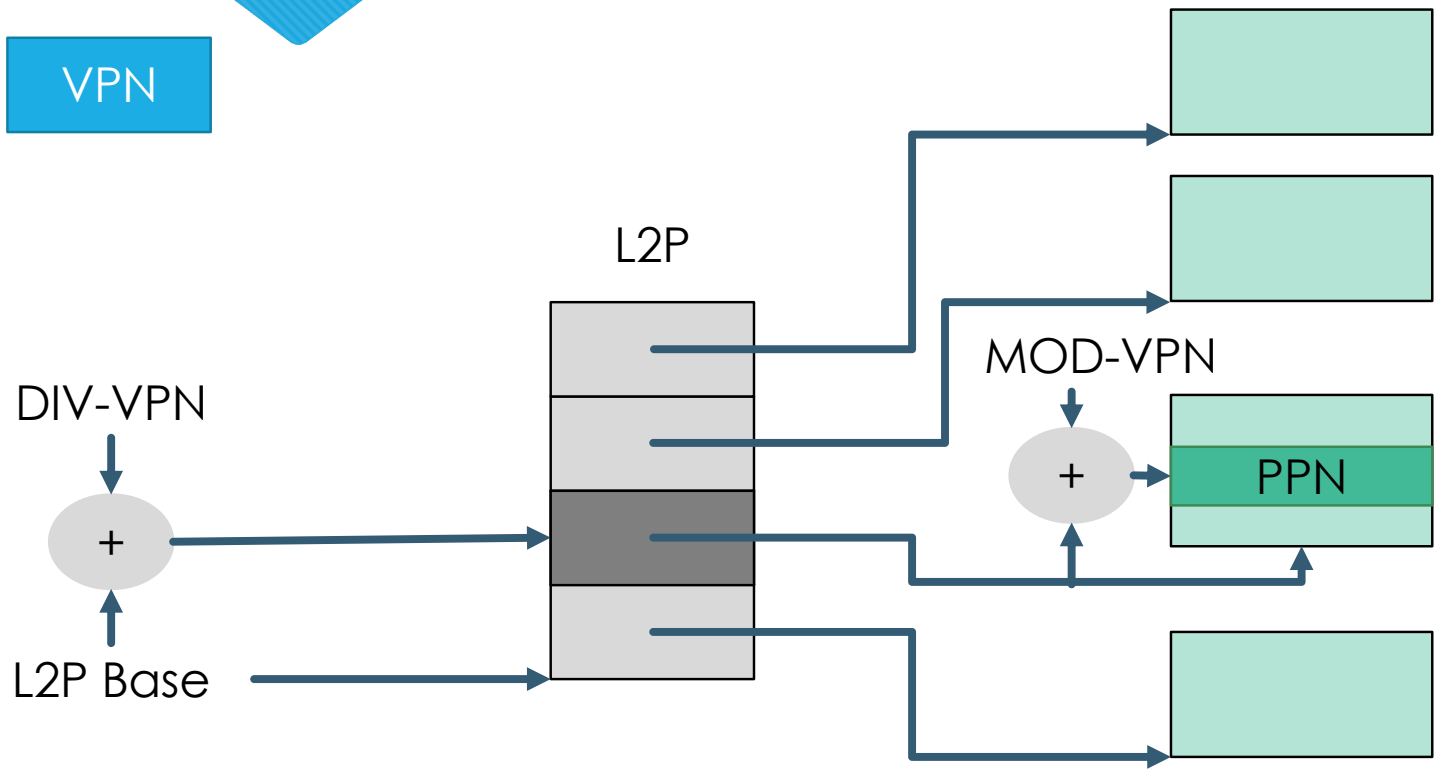
Memory Efficient Hashed Page Tables: Logical-to-Physical (L2P) Table



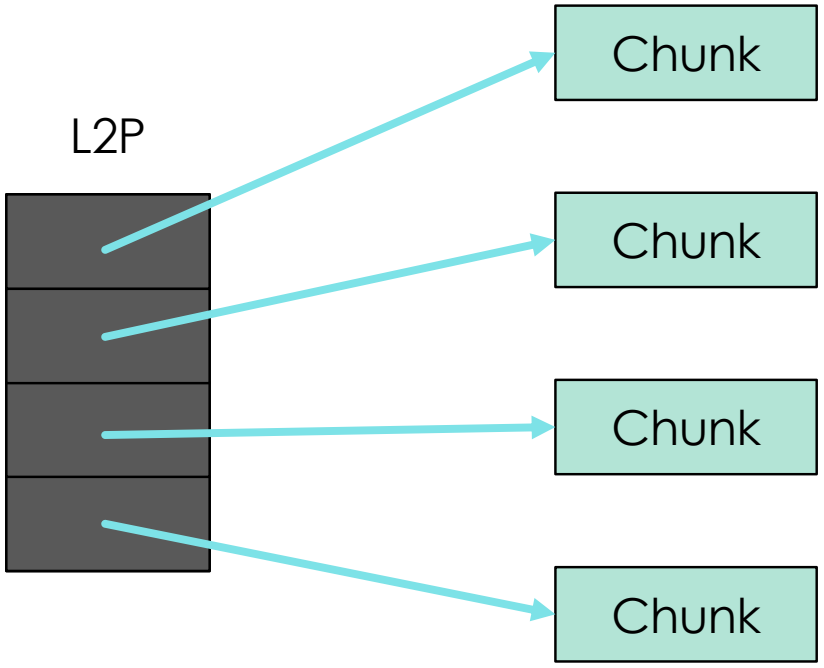
Memory Efficient Hashed Page Tables: Logical-to-Physical (L2P) Table



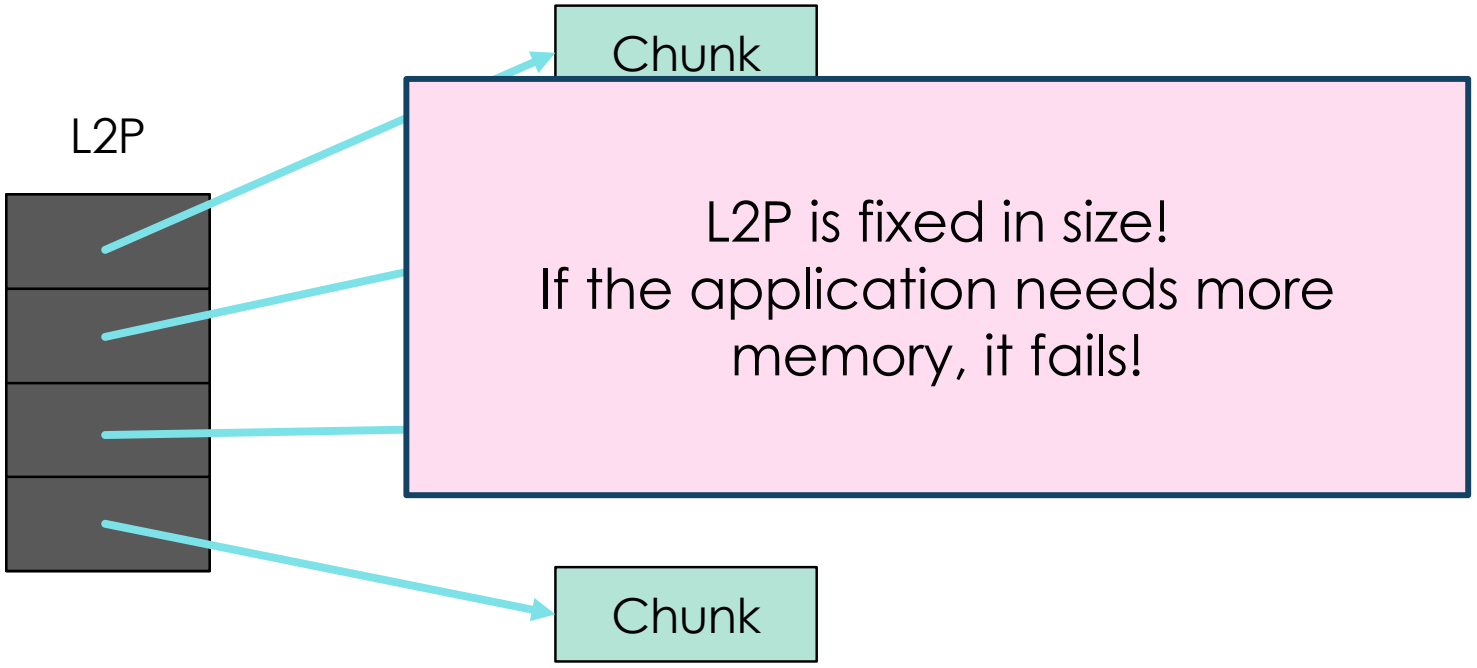
Memory Efficient Hashed Page Tables: Logical-to-Physical (L2P) Table



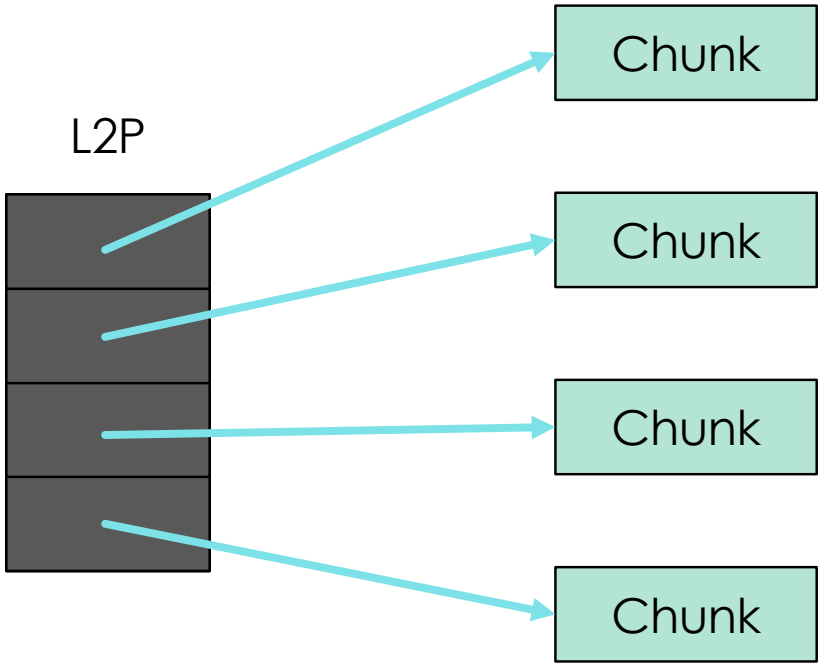
Memory Efficient Hashed Page Tables: Dynamically Changing Chunk Sizes



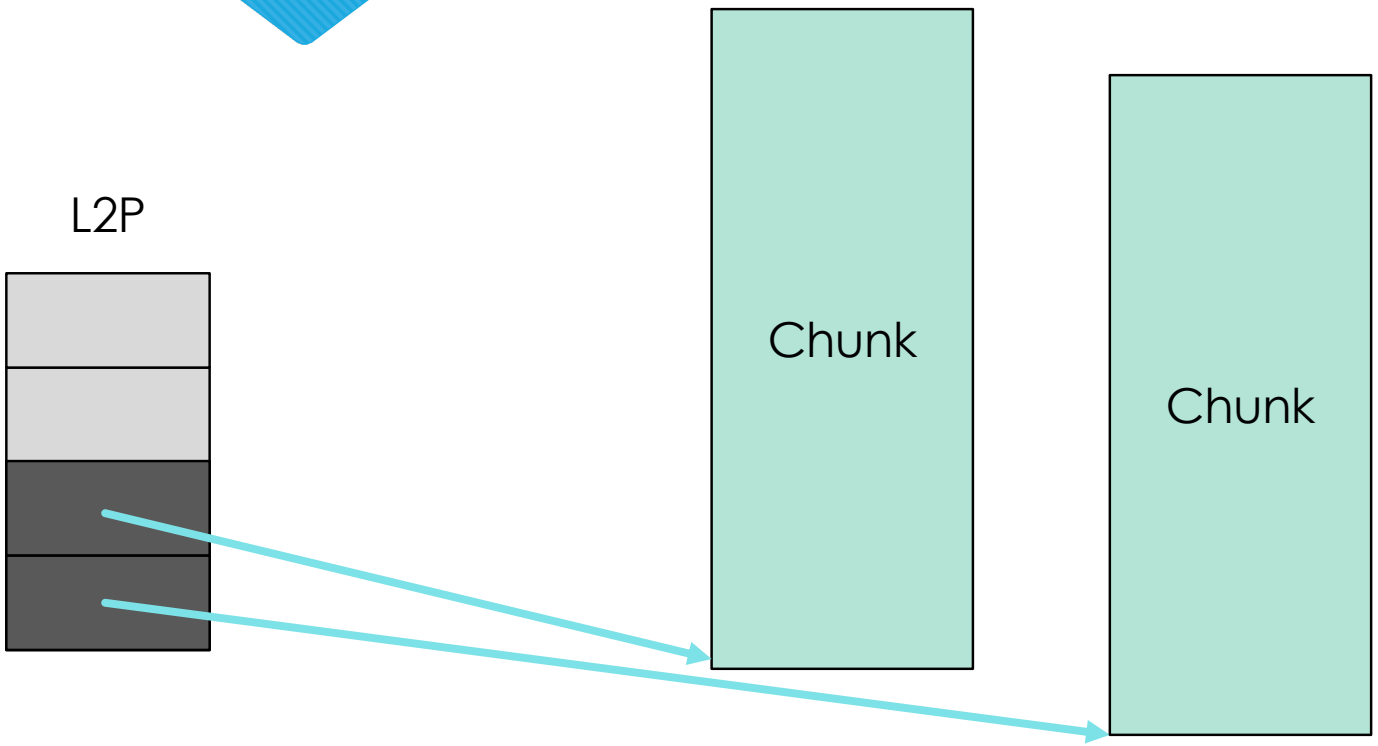
Memory Efficient Hashed Page Tables: Dynamically Changing Chunk Sizes



Memory Efficient Hashed Page Tables: Dynamically Changing Chunk Sizes



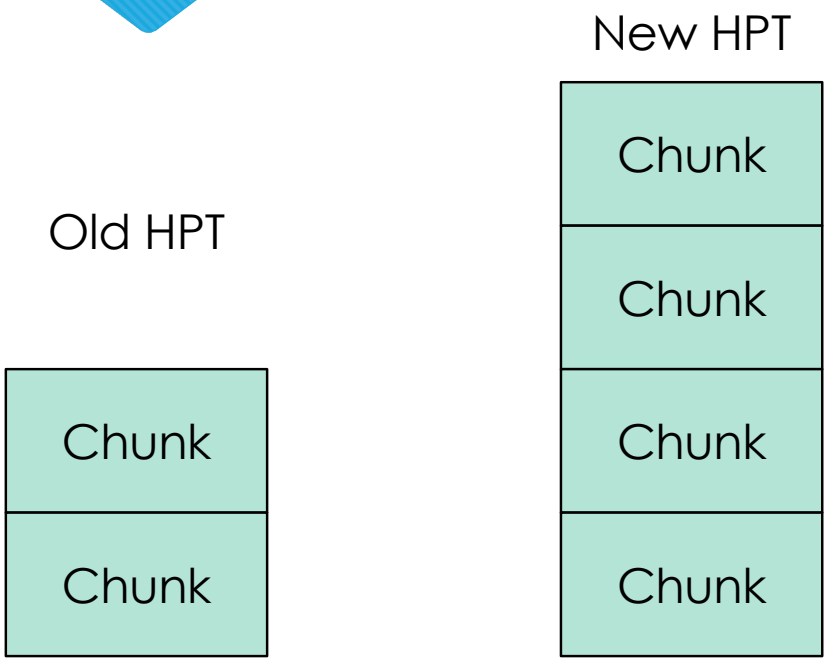
Memory Efficient Hashed Page Tables: Dynamically Changing Chunk Sizes



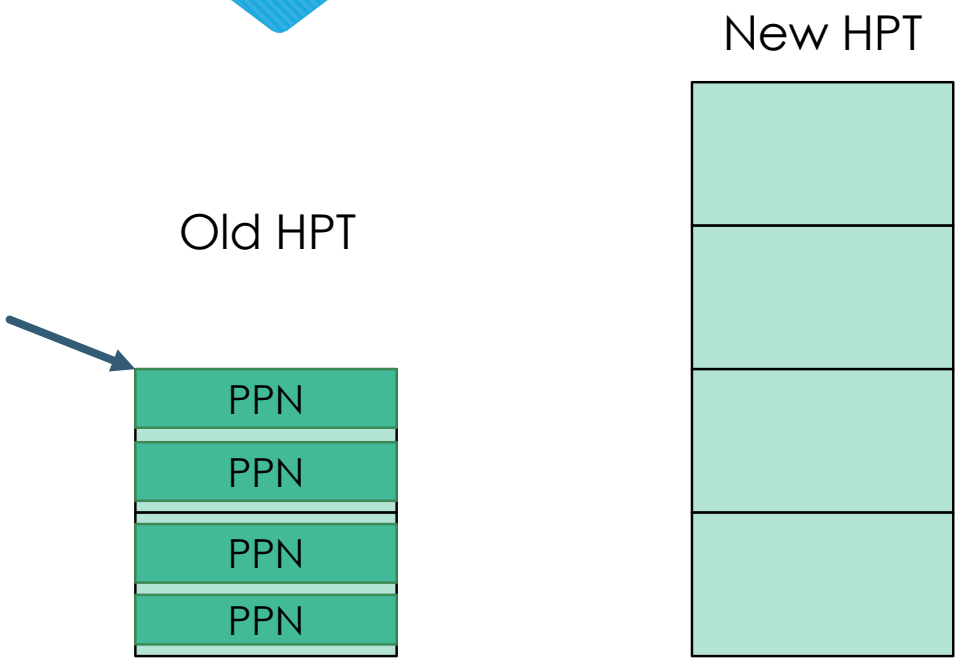
Memory Efficient Hashed Page Tables: Design Overview

- ME-HPTs: Four novel architectural techniques
- Directly minimizing contiguity requirements
  - Logical-to-Physical (L2P) Table
  - Dynamically Changing Chunk Size
- **Indirectly minimizing contiguity requirements by minimizing memory consumption**
  - **In-place Page Table Resizing**
  - **Per-way Page Table Resizing**

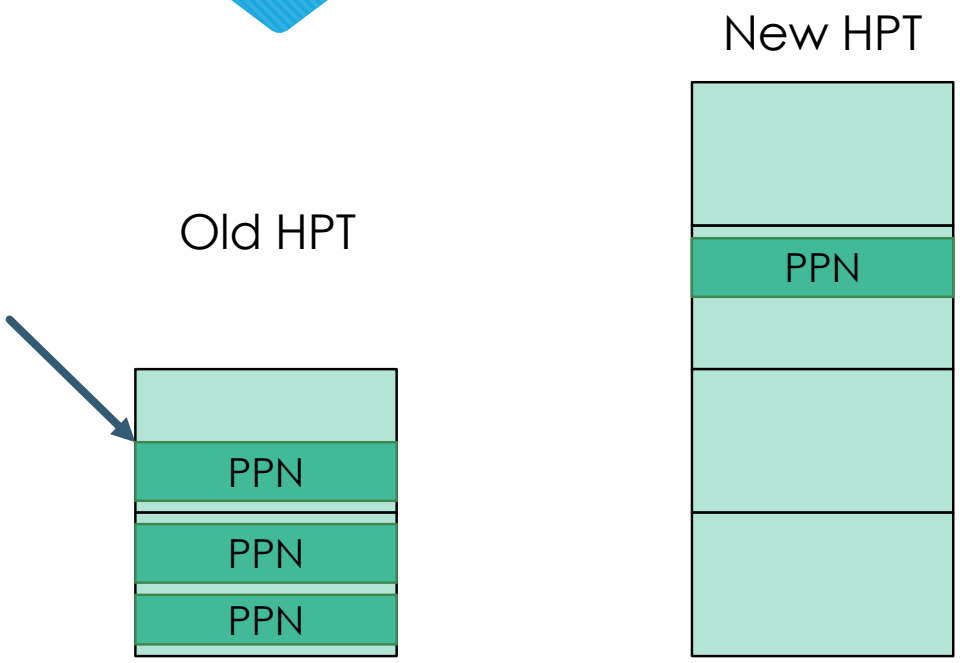
Memory Efficient Hashed Page Tables: In-Place Page Table Resizing



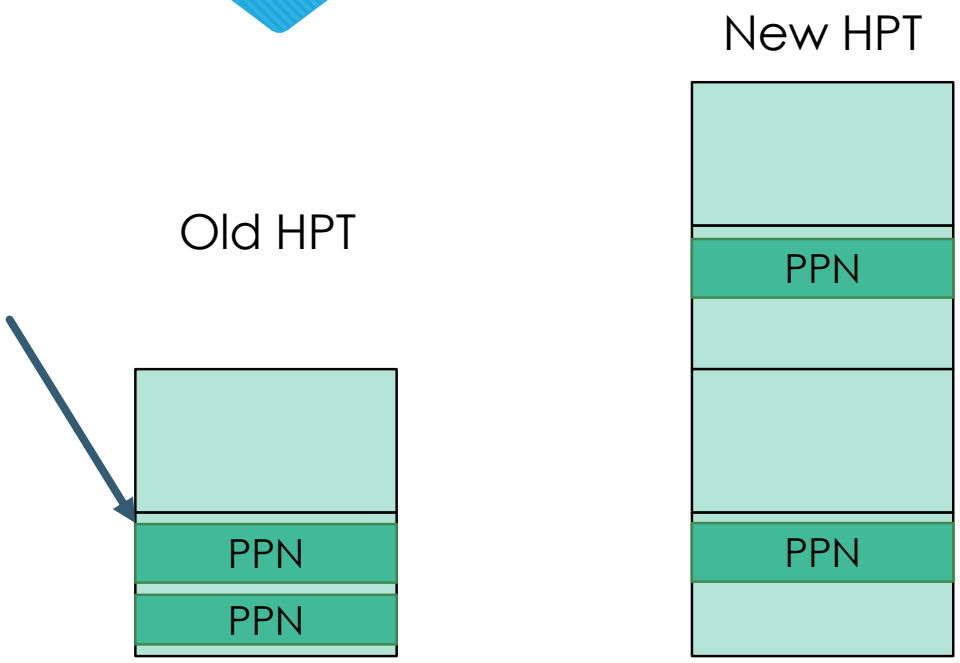
Memory Efficient Hashed Page Tables: In-Place Page Table Resizing



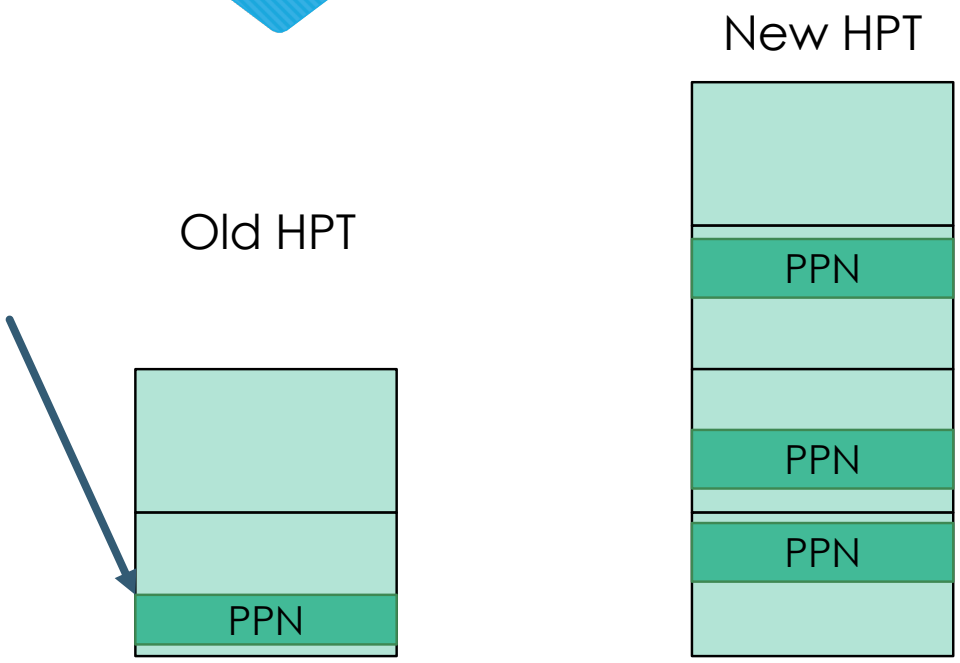
Memory Efficient Hashed Page Tables: In-Place Page Table Resizing



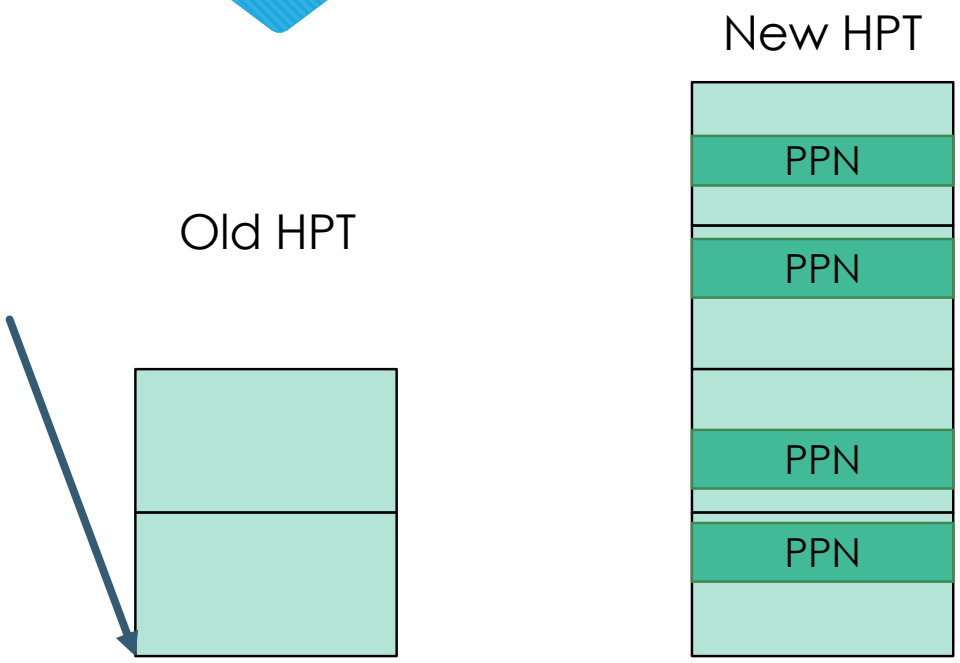
Memory Efficient Hashed Page Tables: In-Place Page Table Resizing



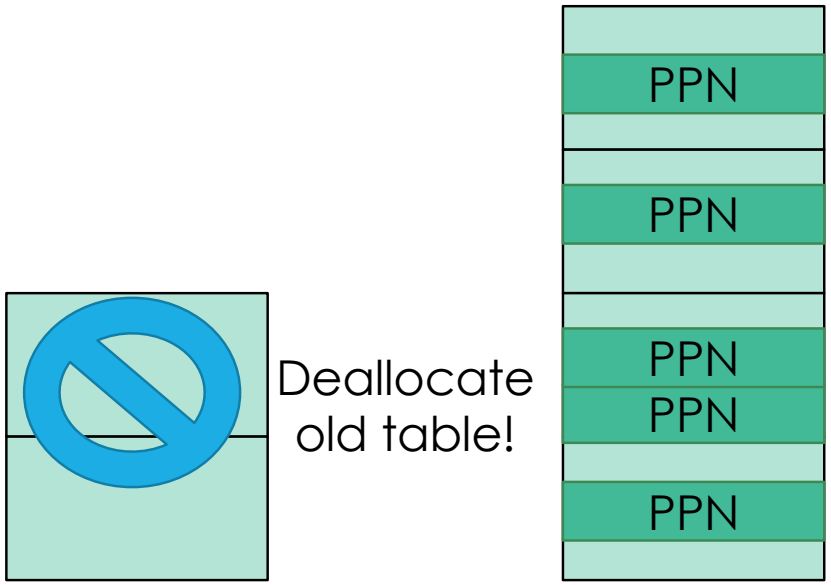
Memory Efficient Hashed Page Tables: In-Place Page Table Resizing



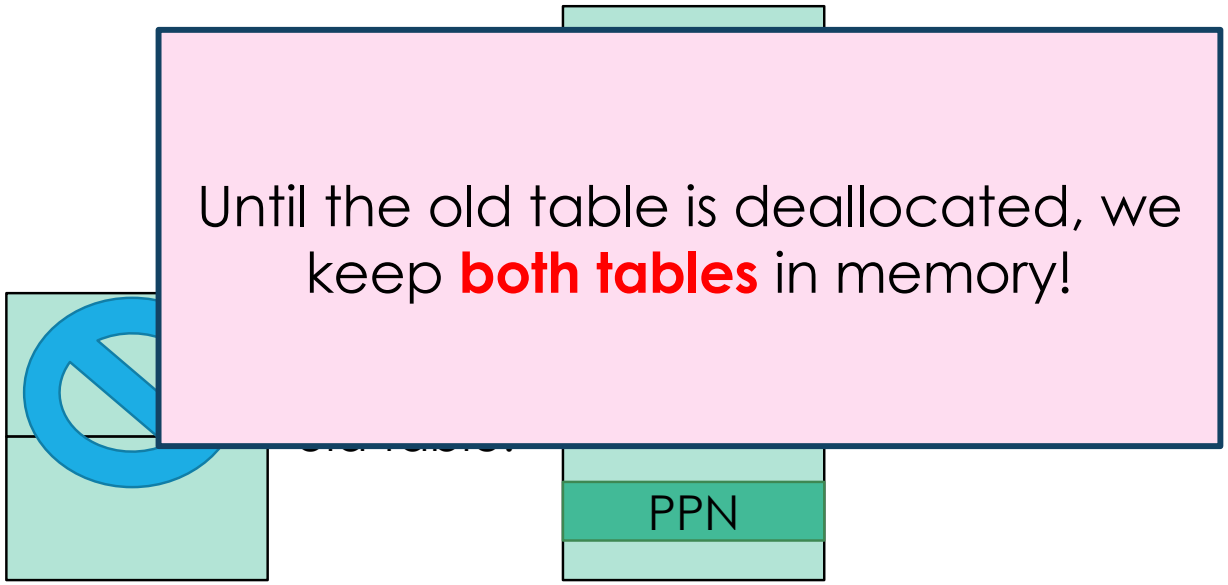
Memory Efficient Hashed Page Tables: In-Place Page Table Resizing



Memory Efficient Hashed Page Tables: In-Place Page Table Resizing

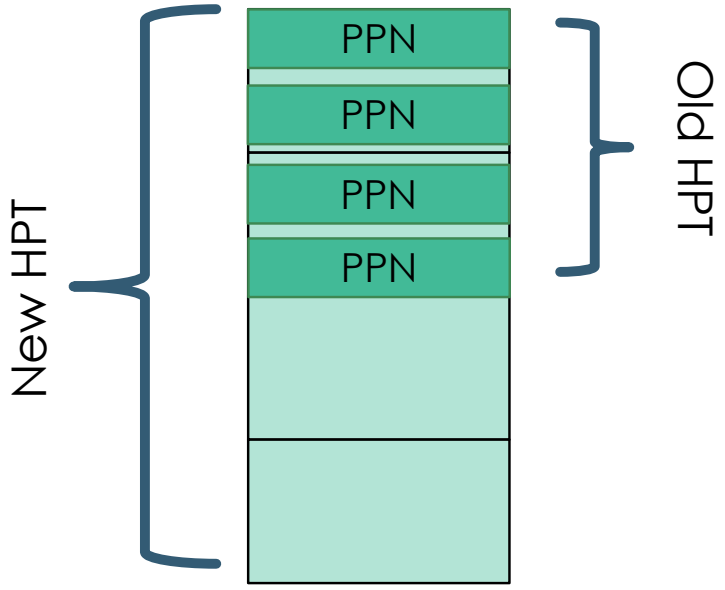


Memory Efficient Hashed Page Tables: In-Place Page Table Resizing



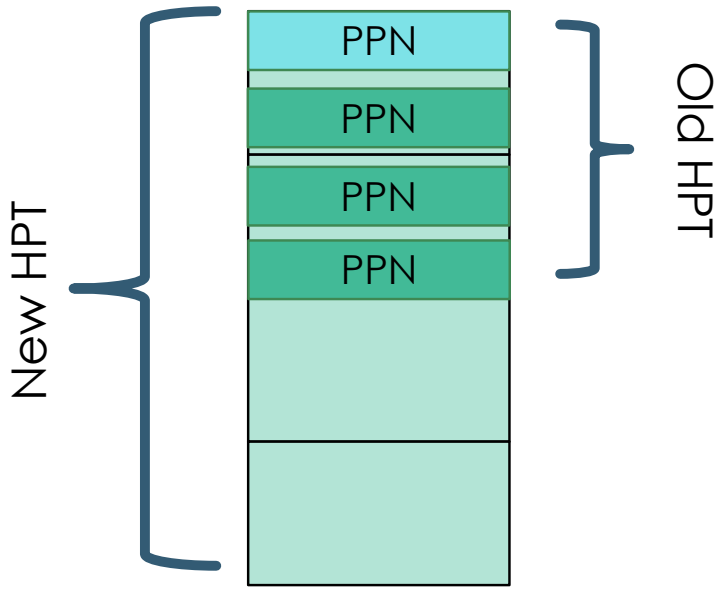
Memory Efficient Hashed Page Tables: In-Place Page Table Resizing

- Keep both tables in shared memory space
- Same hash function for both tables
- On rehash, some entries stay in the same chunk, others move to new chunks



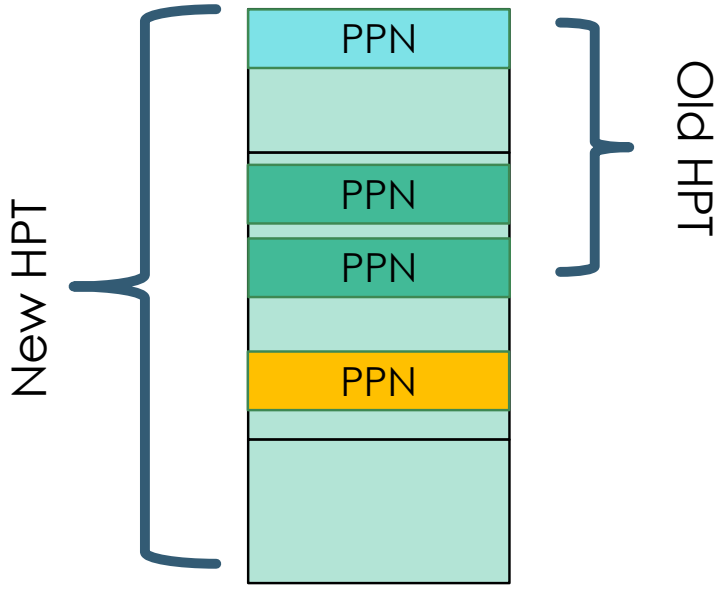
Memory Efficient Hashed Page Tables: In-Place Page Table Resizing

- Keep both tables in shared memory space
- Same hash function for both tables
- On rehash, some entries stay in the same chunk, others move to new chunks



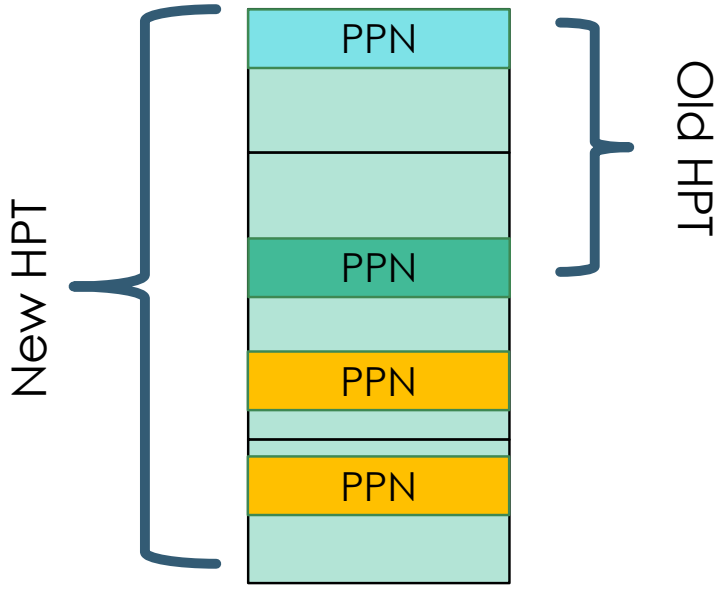
Memory Efficient Hashed Page Tables: In-Place Page Table Resizing

- Keep both tables in shared memory space
- Same hash function for both tables
- On rehash, some entries stay in the same chunk, others move to new chunks



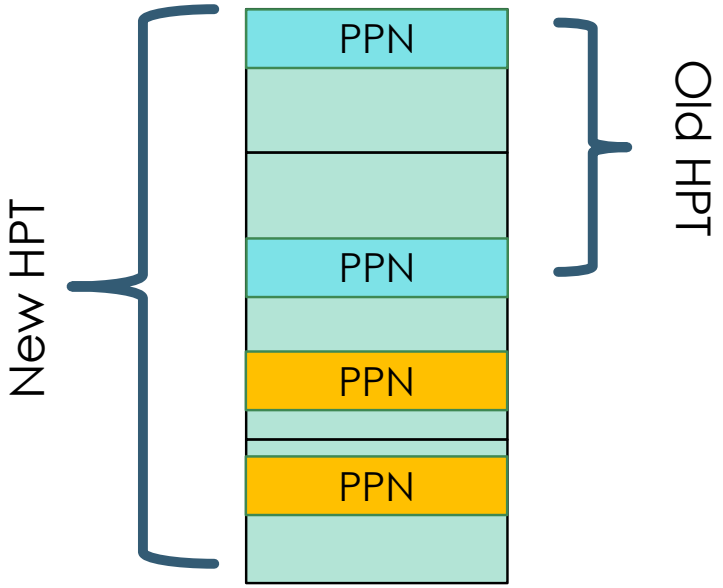
Memory Efficient Hashed Page Tables: In-Place Page Table Resizing

- Keep both tables in shared memory space
- Same hash function for both tables
- On rehash, some entries stay in the same chunk, others move to new chunks

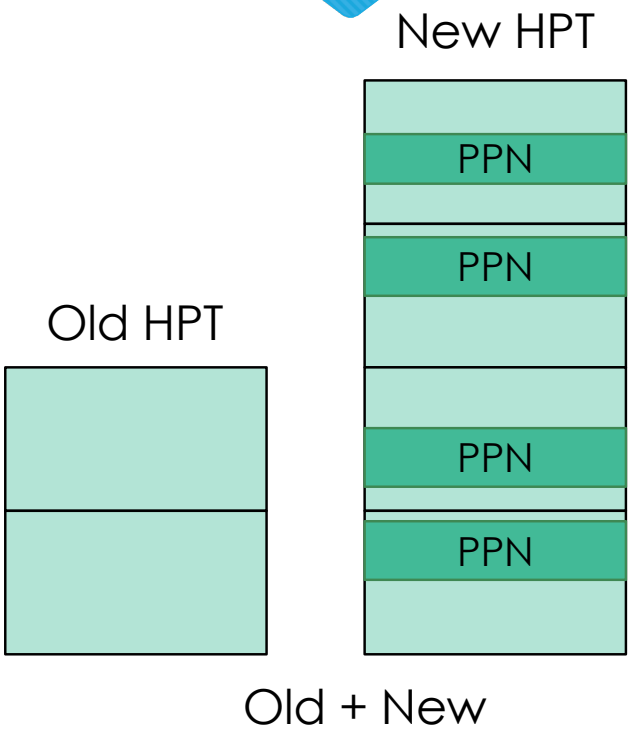


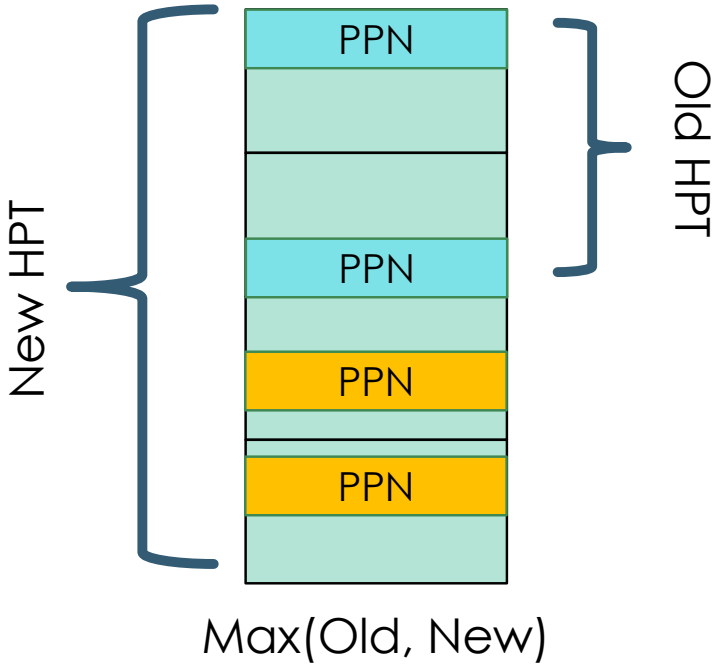
Memory Efficient Hashed Page Tables: In-Place Page Table Resizing

- Keep both tables in shared memory space
- Same hash function for both tables
- On rehash, some entries stay in the same chunk, others move to new chunks

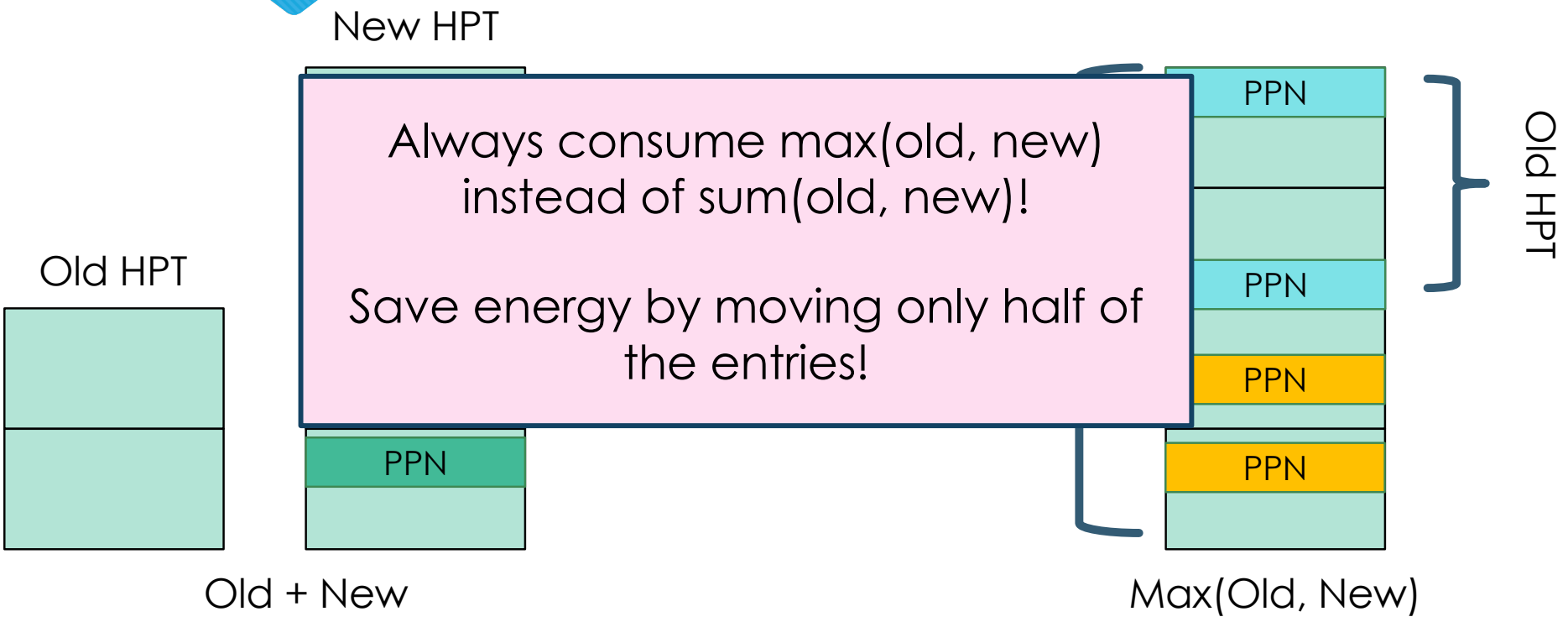


Memory Efficient Hashed Page Tables: In-Place Page Table Resizing

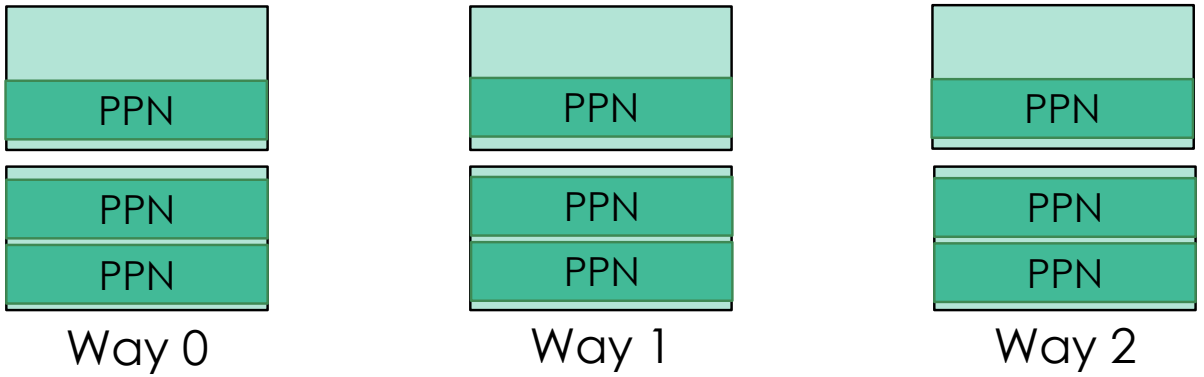




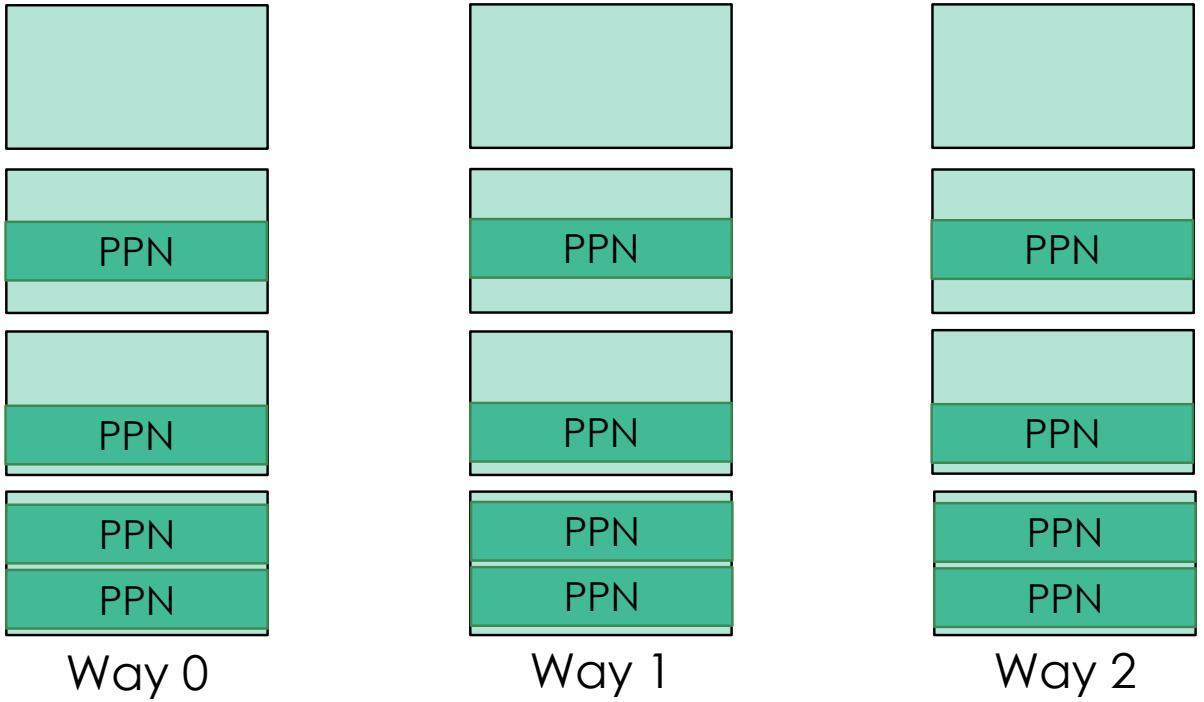
Memory Efficient Hashed Page Tables: In-Place Page Table Resizing



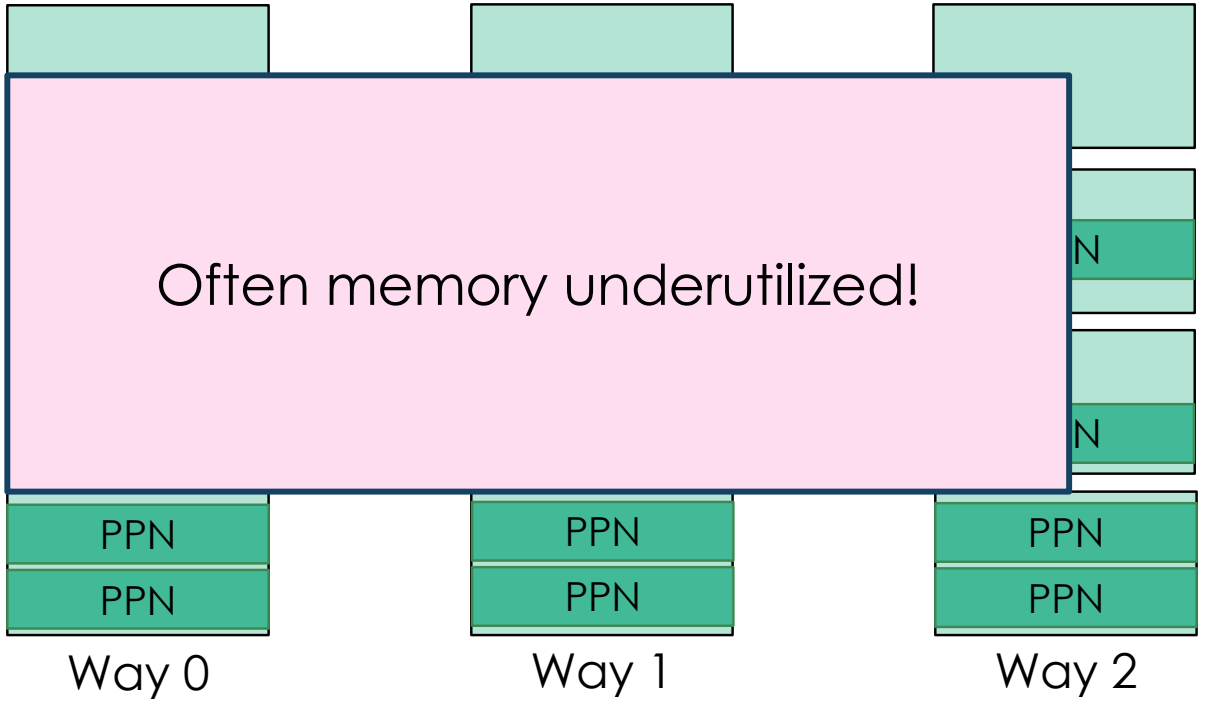
Memory Efficient Hashed Page Tables: Per Way Page Table Resizing



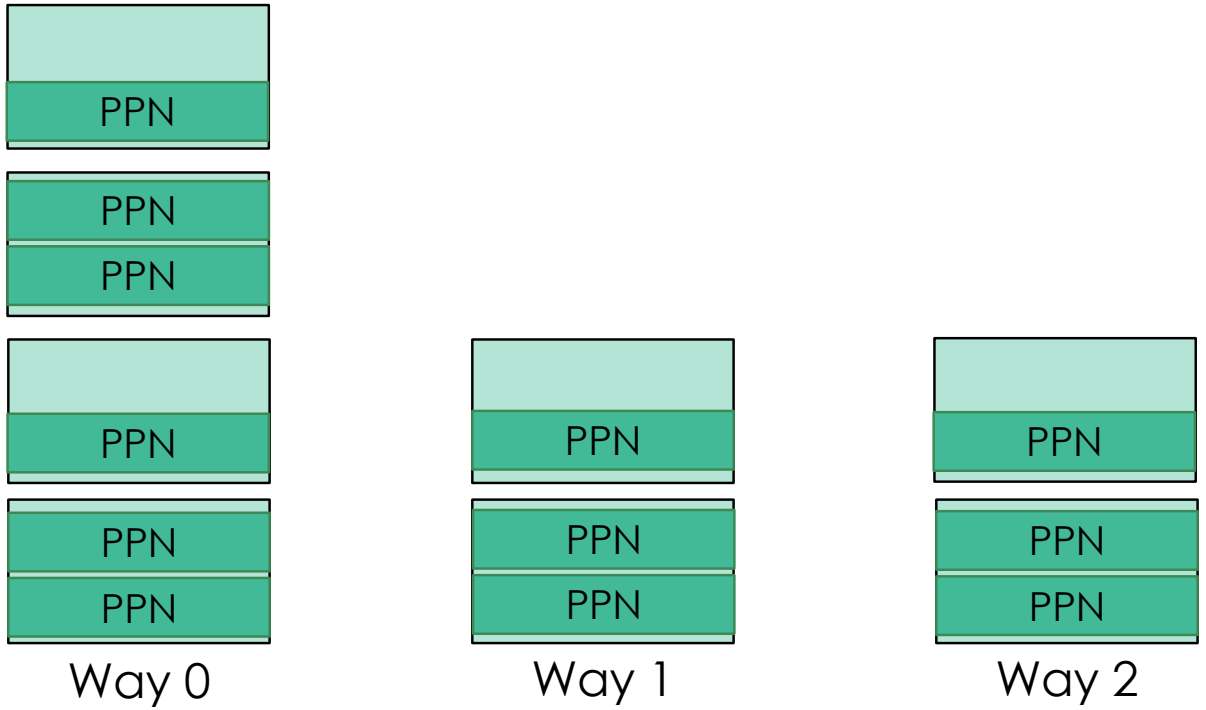
Memory Efficient Hashed Page Tables: Per Way Page Table Resizing



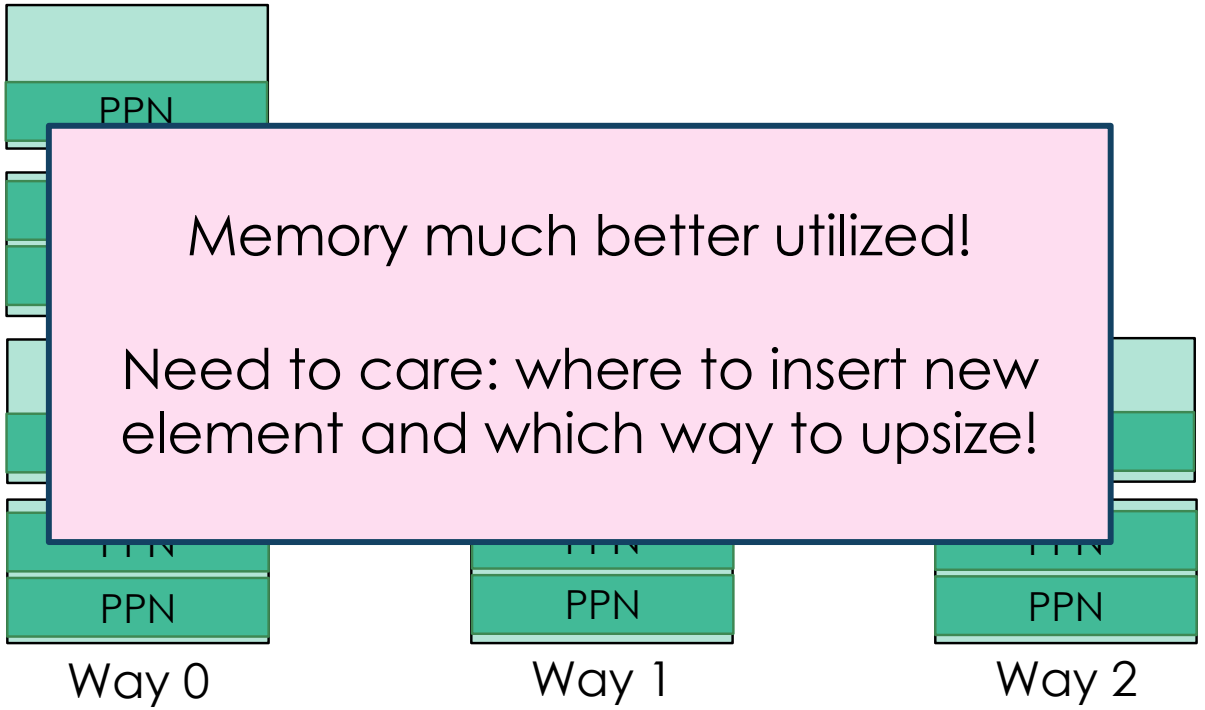
Memory Efficient Hashed Page Tables: Per Way Page Table Resizing



Memory Efficient Hashed Page Tables: Per Way Page Table Resizing



Memory Efficient Hashed Page Tables: Per Way Page Table Resizing

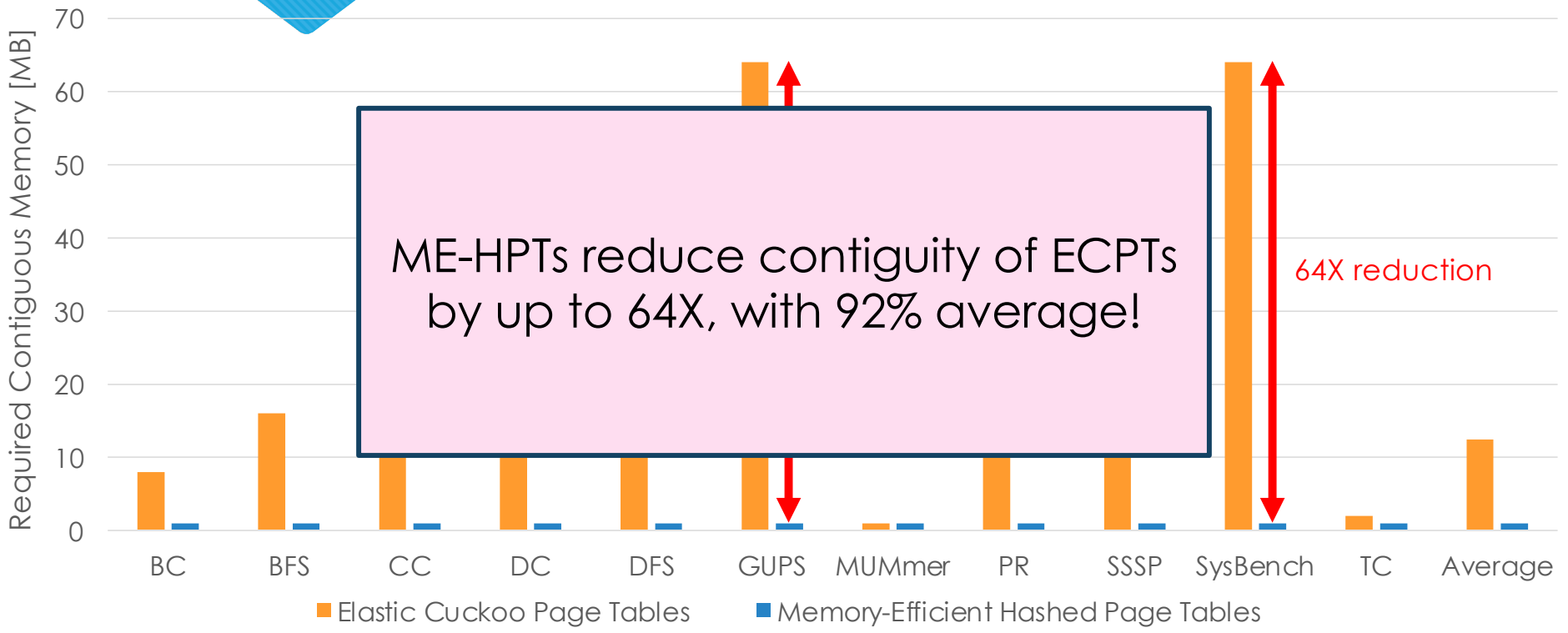


Outline of this talk

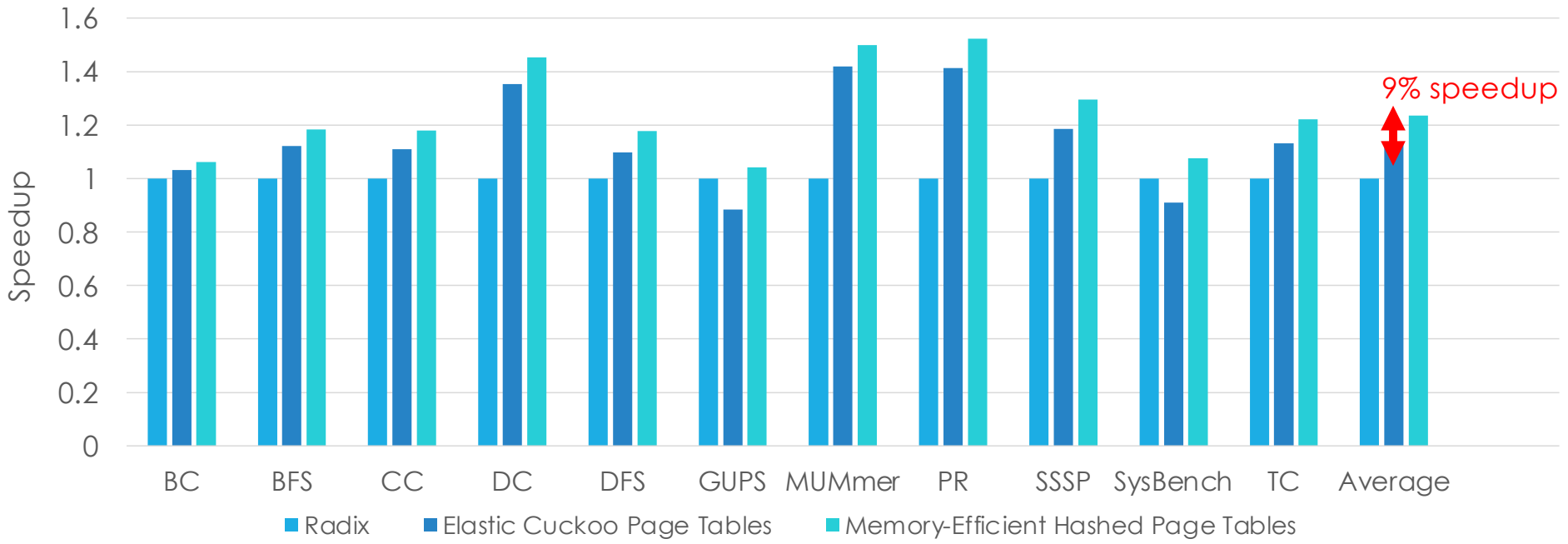
- Page Table Organizations
- Hashed Page Tables Memory Requirements
- **ME-HPTs: Memory-Efficient Hashed Page Tables**
  - ME-HPTs Design
  - ME-HPTs Key Results
- Conclusion

Significant Memory Contiguity Savings

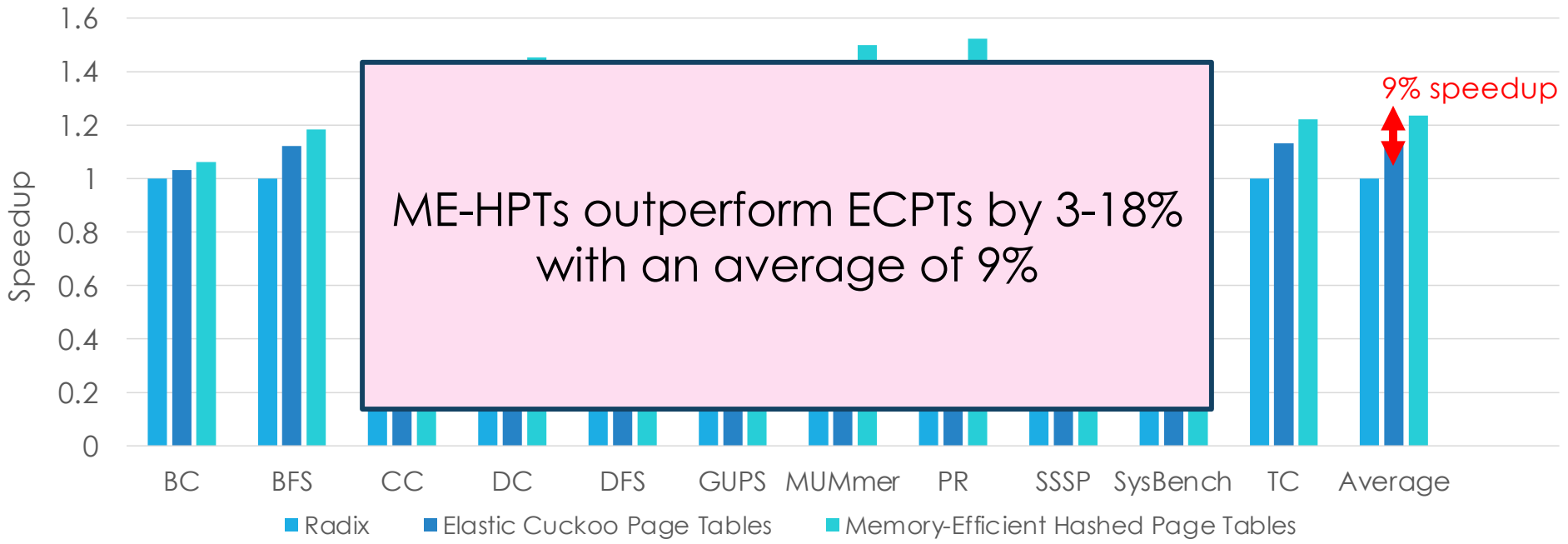
Significant Memory Contiguity Savings



Improved Application Performance



Improved Application Performance



Conclusion

- Four novel architectural techniques to provide Memory-Efficient Hashed Page Tables
  - L2P Table
  - Dynamically Changing Chunk Sizes
  - In-Place Page Table Resizing
  - Per-Way Page Table Resizing
- Reduced memory contiguity requirement by 92%
- Sped-up applications by 9% on average
- Allow large-memory applications to run at high performance on highly fragmented servers

ME-HPTs: Memory-Efficient Hashed Page Tables

HPCA 2023

Jovan Stojkovic, Namrata Mantri, Dimitrios Skarlatos*, Tianyin Xu, Josep Torrellas

University of Illinois at Urbana-Champaign

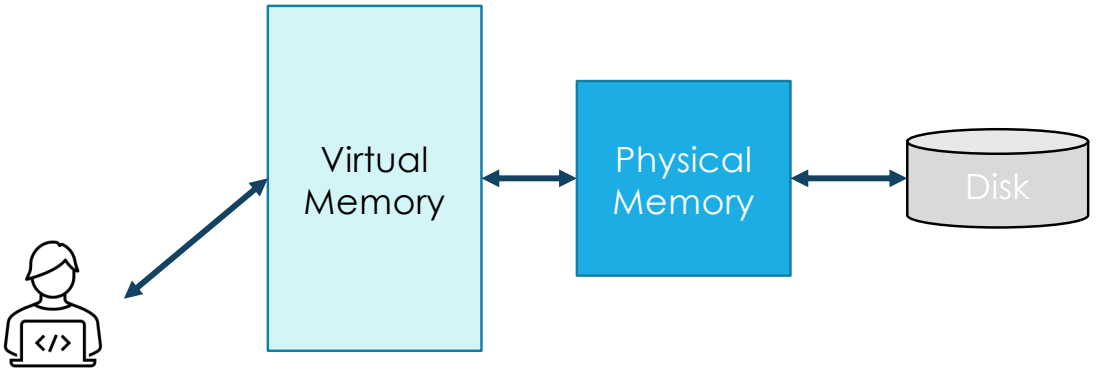
*Carnegie Mellon University

Questions?

Backup Slides

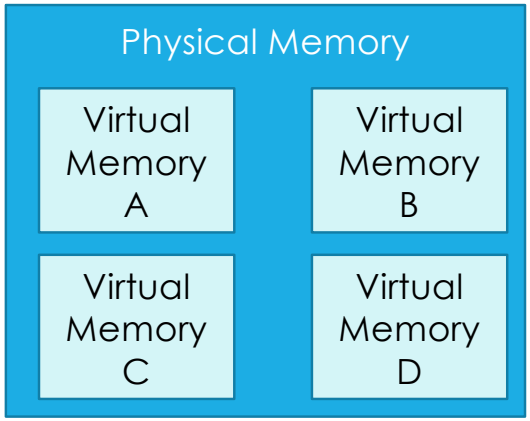
Virtual Memory Needs Memory-Efficient Hashed Page Tables

- Virtual memory is essential technique in modern computing systems
  - Memory virtualization

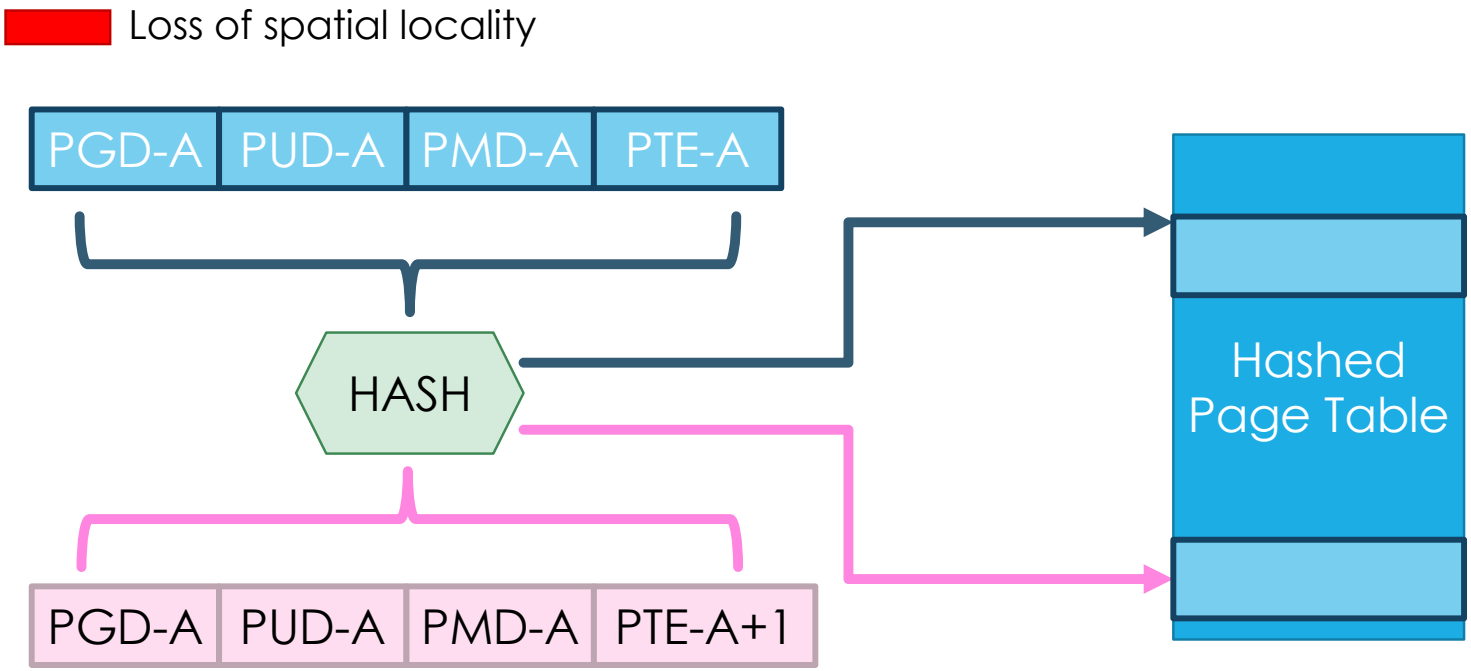


Virtual Memory Needs Memory-Efficient Hashed Page Tables

- Virtual memory is essential technique in modern computing systems
  - Memory virtualization
  - Process isolation

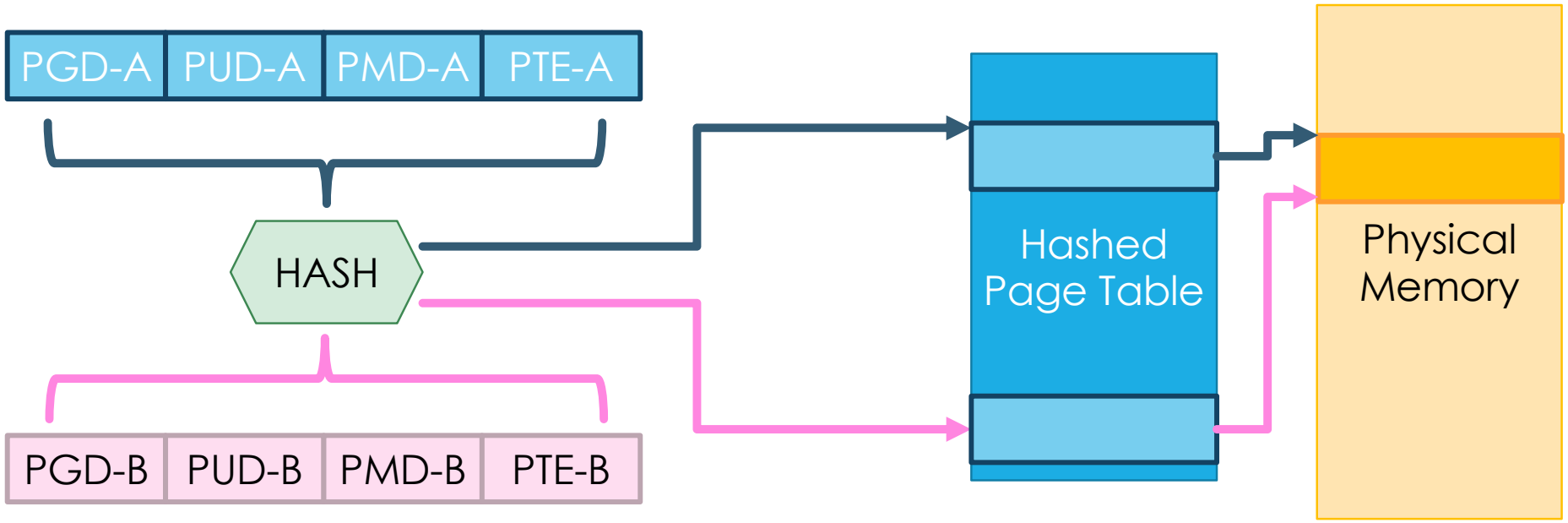


Hashed Page Tables



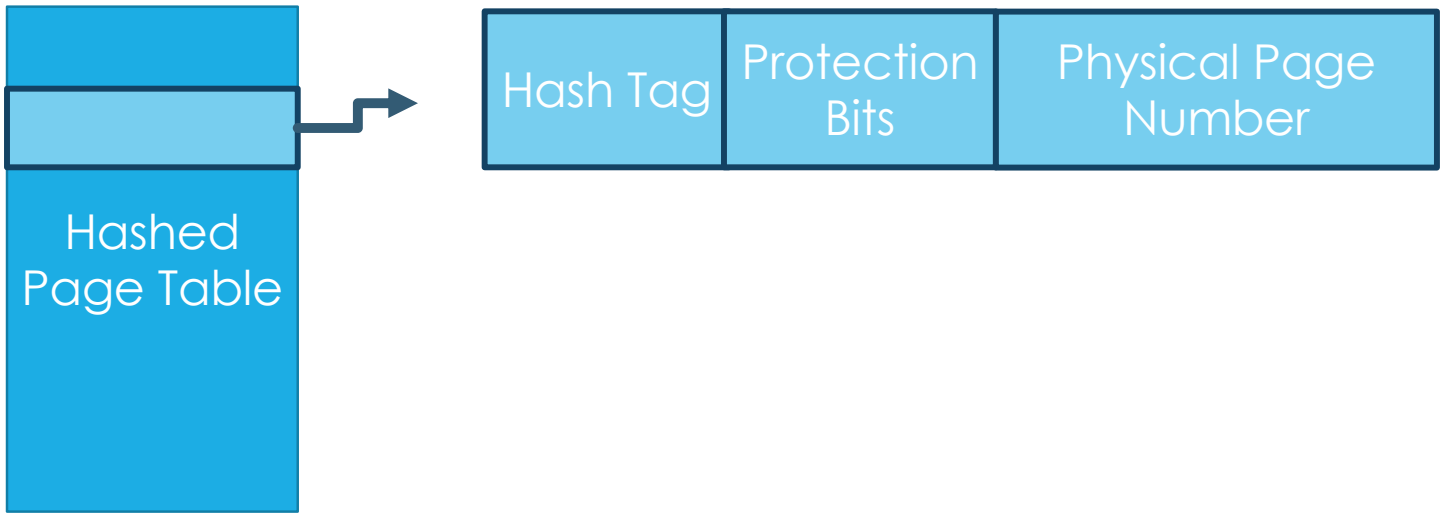
Hashed Page Tables

Page sharing and multiple page sizes not easy to support with global hashed table



Hashed Page Tables: One Step Forward, Two Steps Back

Extra space for hash tags

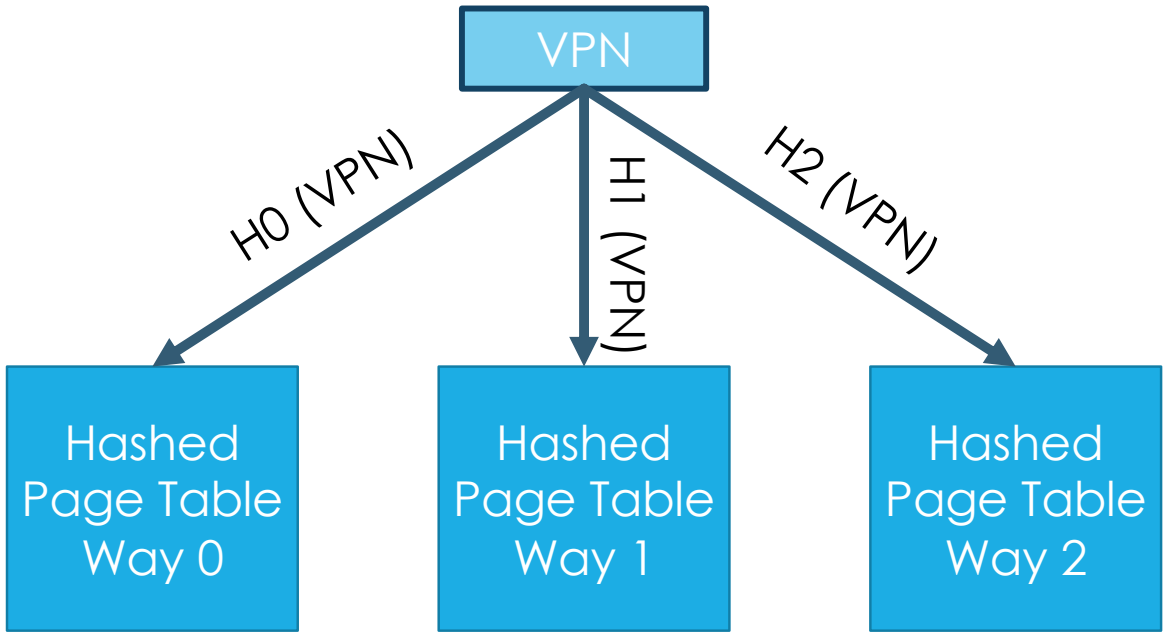


Hashed Page Tables: Recent Advances Make Them Compelling

- PTE Compaction and Clustering

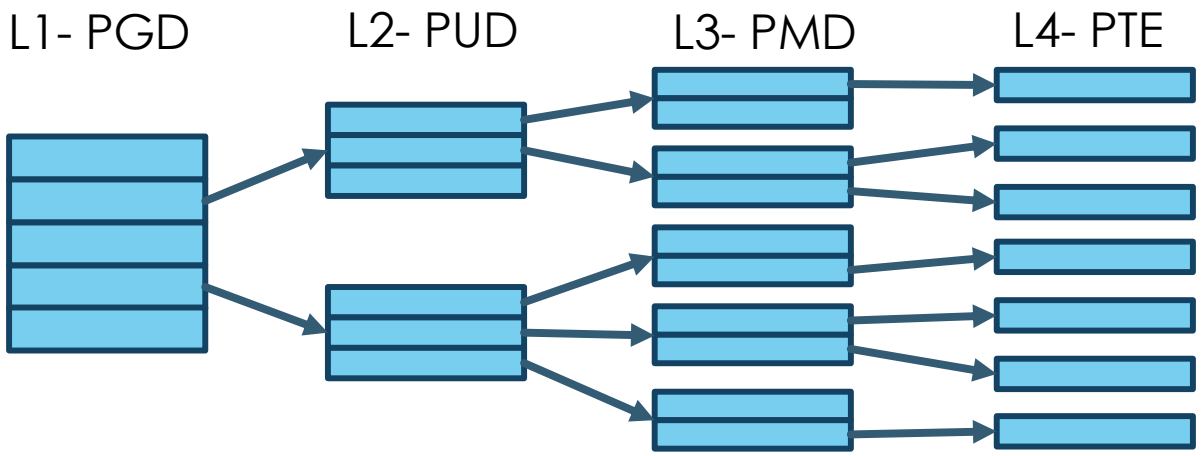
Hashed Page Tables: Recent Advances Make Them Compelling

- Cuckoo Hashing for Collision Handling

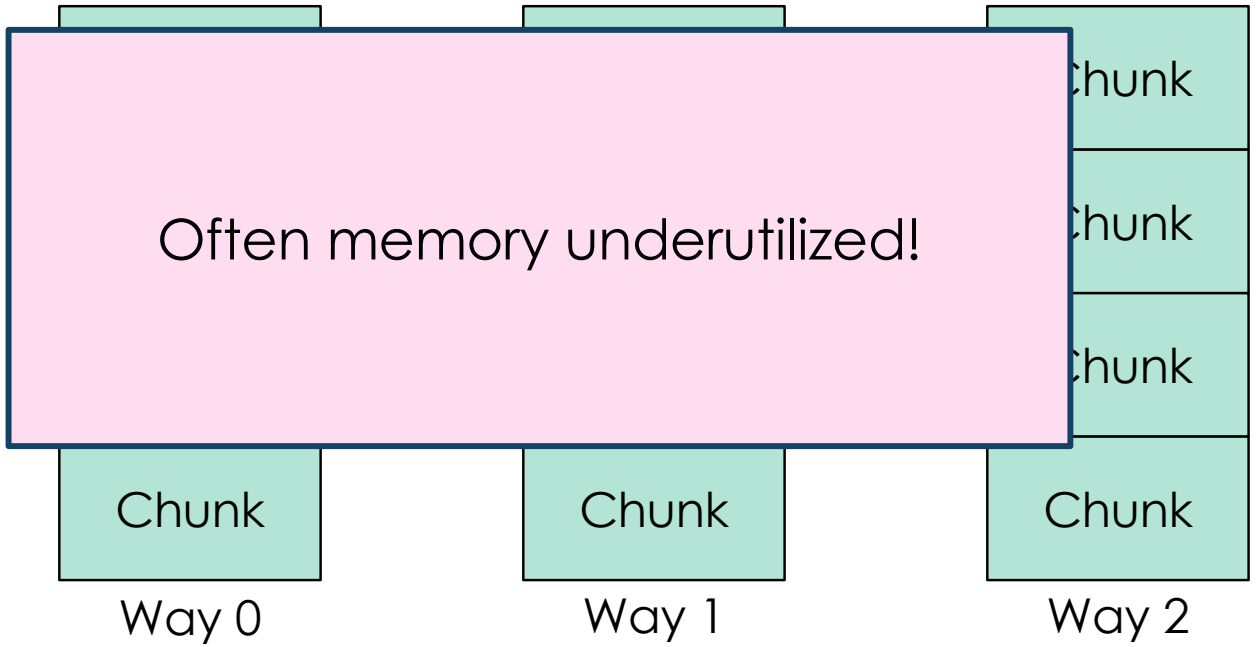


Hashed Page Tables: Large Contiguous Memory Chunks

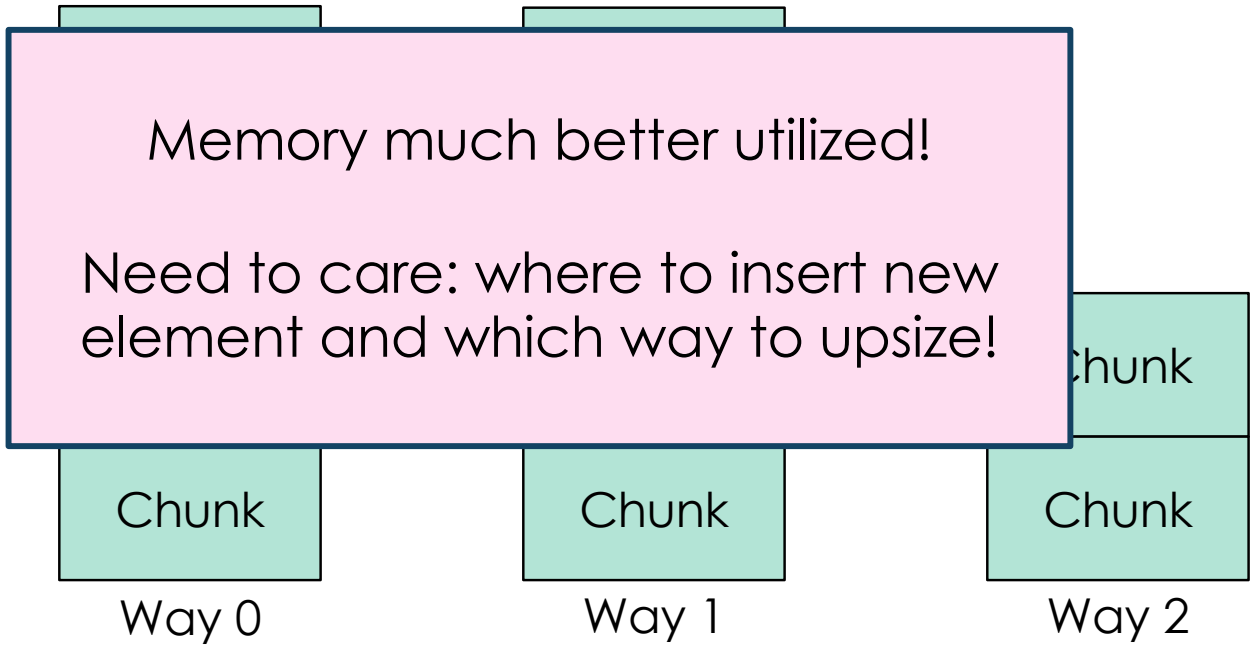
- With radix page tables – unity of allocation is a 4KB page
  - L1 and each L2, L3 and L4 page tables are allocated independently



ME-HPTs:
Per Way Page Table Resizing



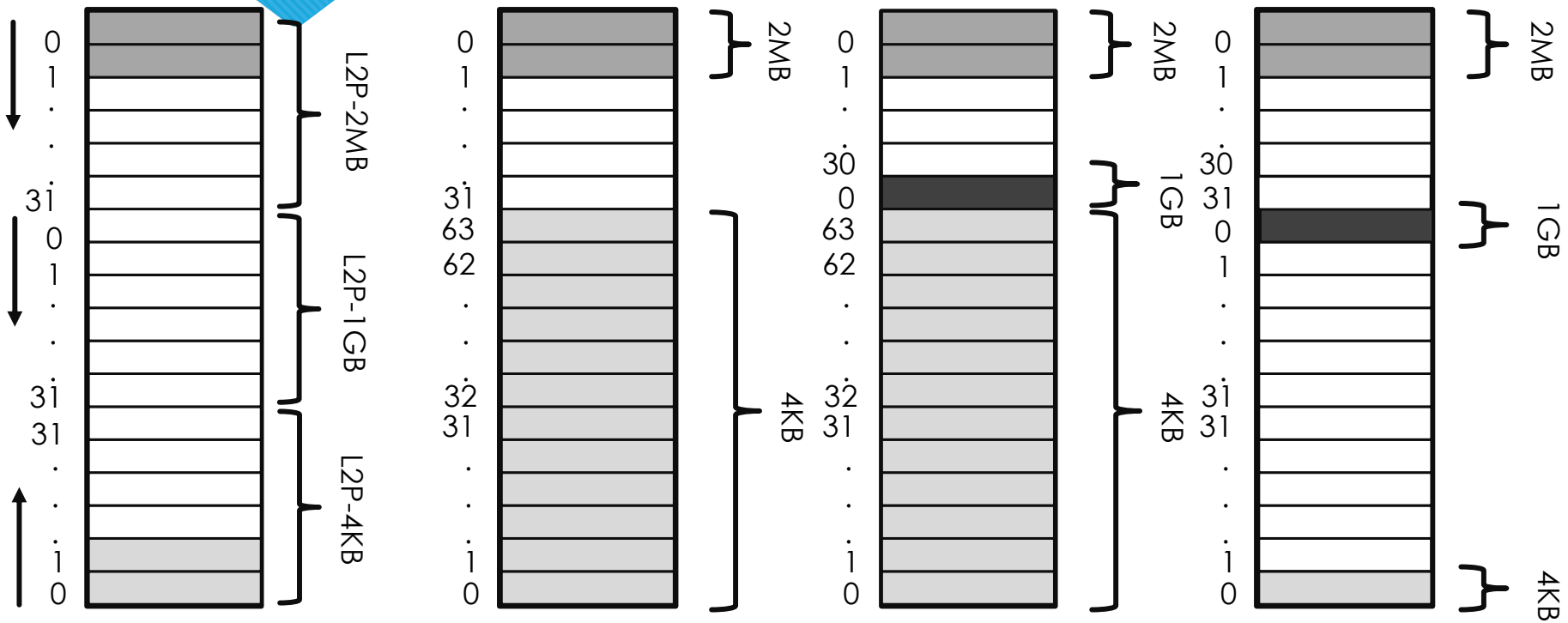
ME-HPTs:
Per Way Page Table Resizing



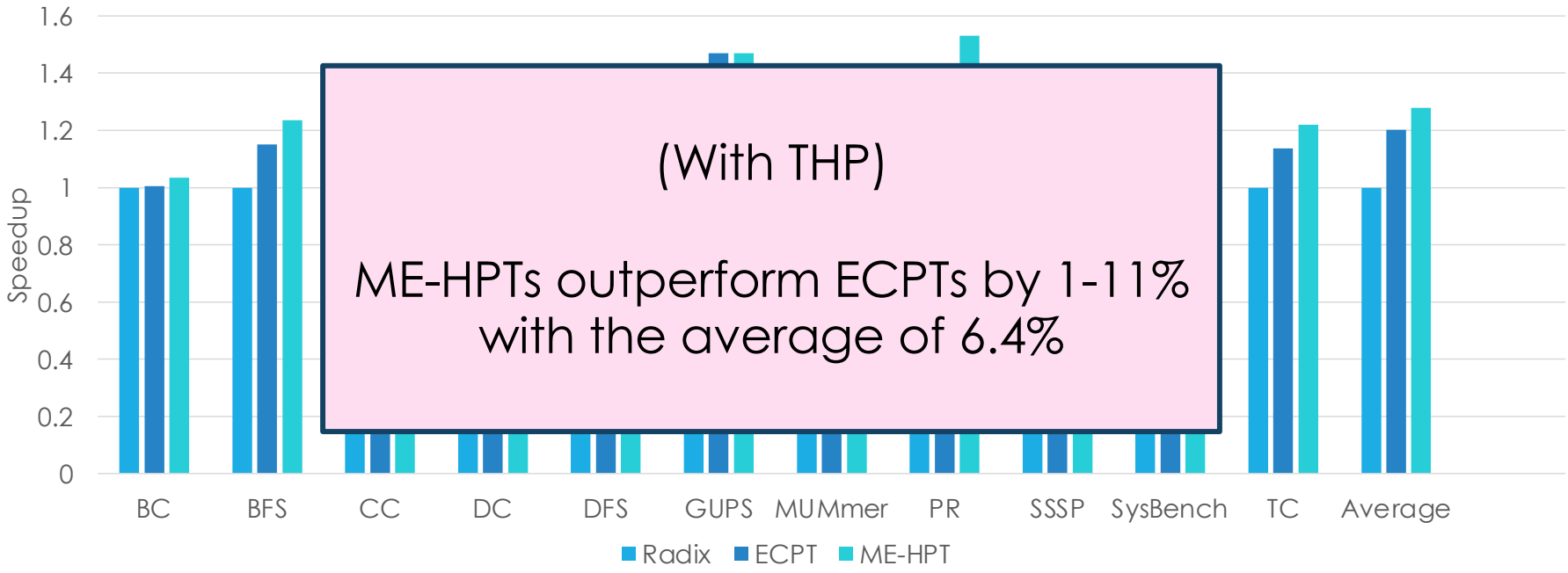
ME-HPTs Implementation: L2P Table Entry Stealing

- L2P Table is per page size of each page table way and its size is fixed
- Applications rarely use all page tables extensively
  - Some L2P tables will be less used than the others
- **Steal L2P entries from one L2P table and give them to another L2P table**

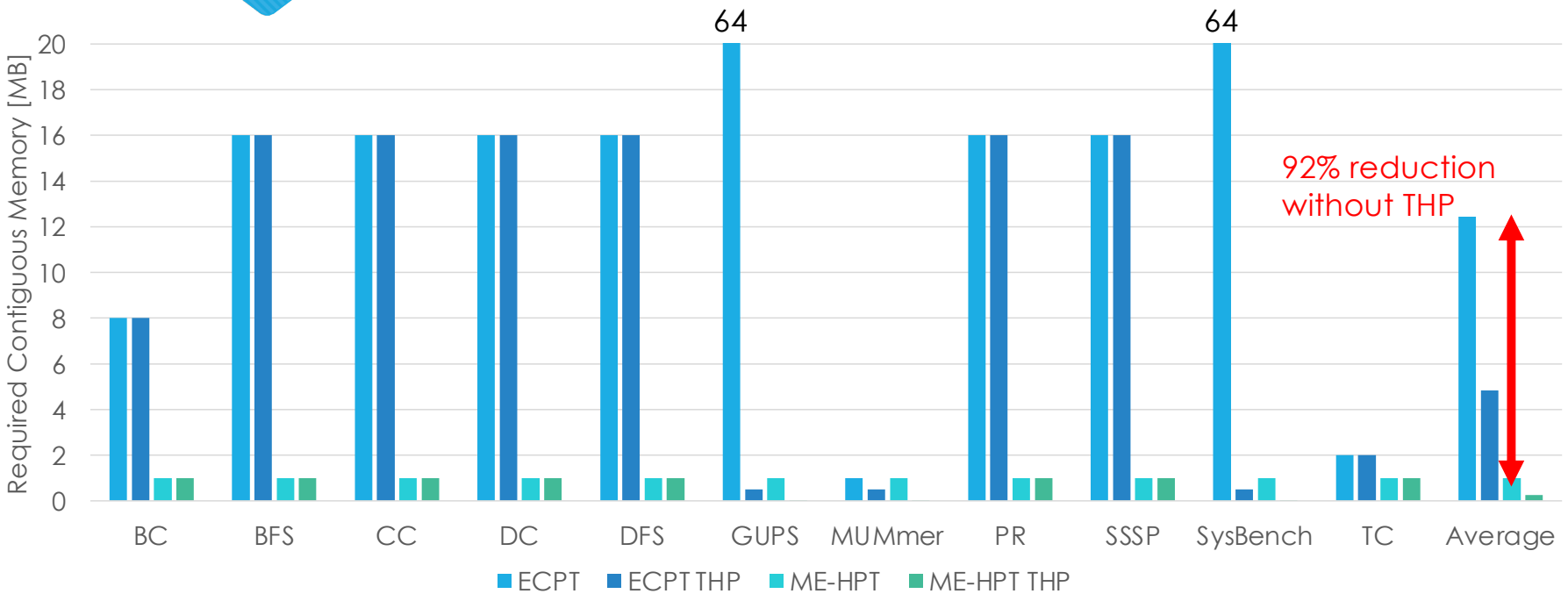
ME-HPTs Implementation: L2P Table Entry Stealing



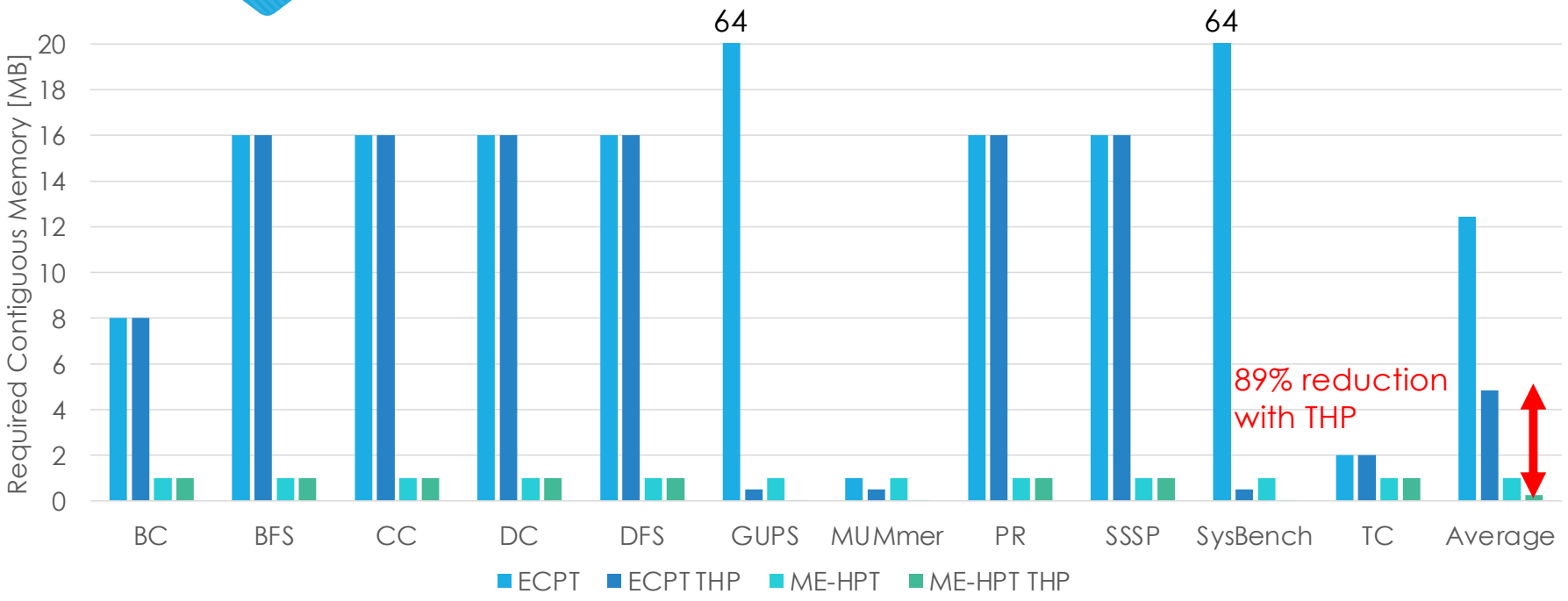
ME-HPTs Key Results: Improved Application Performance



ME-HPTs Key Results: Significant Memory Contiguity Savings



ME-HPTs Key Results: Significant Memory Contiguity Savings



ME-HPTs Key Results: Memory Consumption Reduction

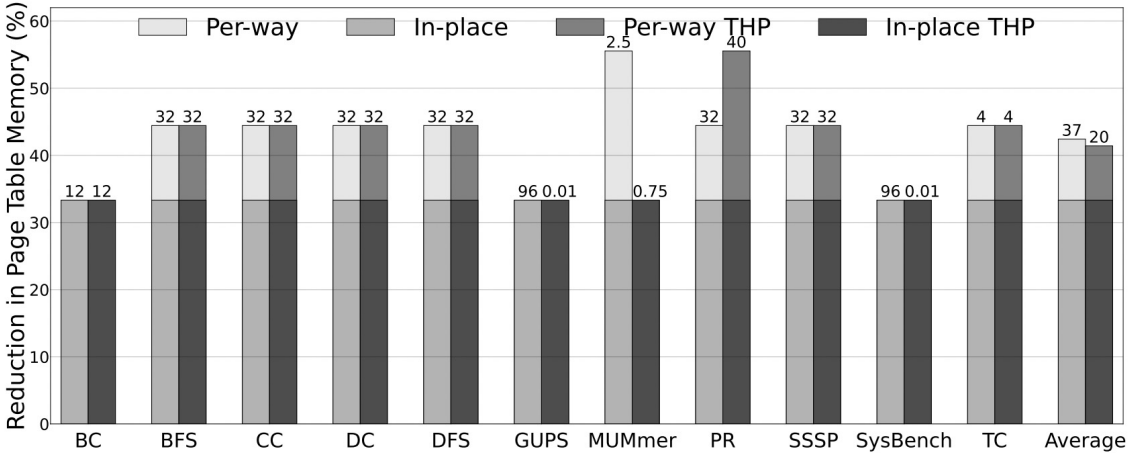


Fig. 10: Reduction in page table memory attained by ME-HPT over the ECPT baseline. The number on top of each bar is the absolute reduction in Mbytes.

ME-HPTs Key Results: Number of L2P Table Entries Used per App

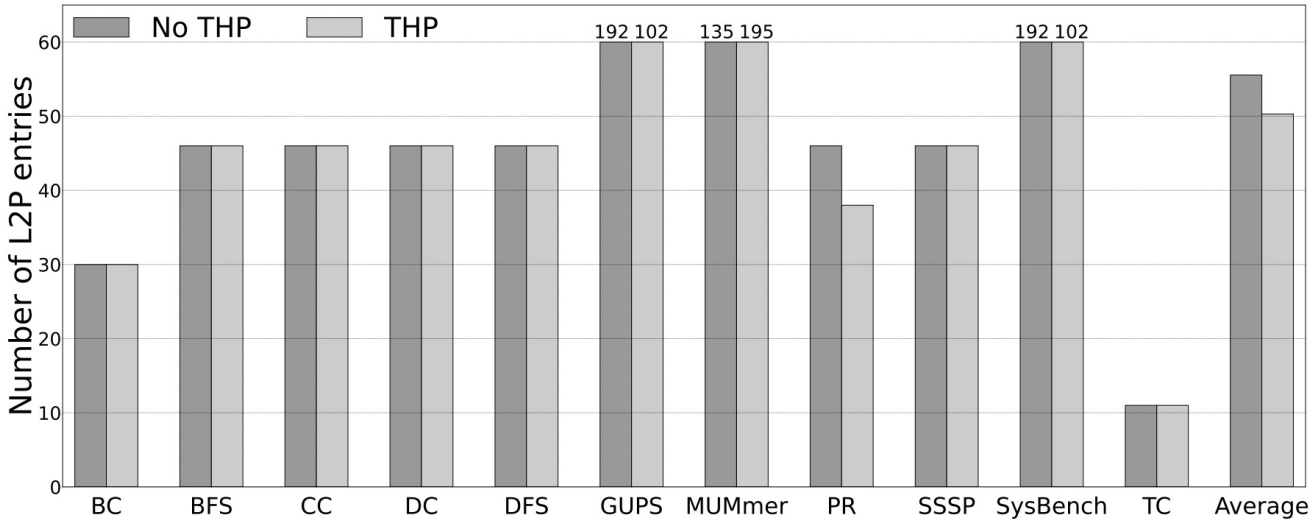


Fig. 14: Number of L2P table entries used per application.

ME-HPTs Other Use Cases

- Techniques applicable to various hash table designs beyond HPTs
- **Scalable Secure Directories**
  - Directories as set-associative structures
  - Efficient resizing required
- **Memory Indexing**
  - Hash tables commonly used to implement memory indices of databases, file systems…
  - Dynamic resizing key operation: in-place resizing useful
- **Key-value Stores**
  - Dynamic structures whose size is unknown ahead of time