ELASTIC CUCKOO PAGE TABLES

Parallelism for Minimizing Virtual Memory Translation Overhead

Nick Hollon & Rachel Nguyen

01

Background

02

Elastic Cuckoo Hashing

03

Elastic Cuckoo Page Table Design

04

Elastic Cuckoo Page Table Performance

Based on **Elastic Cuckoo Page Tables: Rethinking Virtual Memory Translation for Parallelism** by

Dimitrios Skarlatos, Apostolos Kokolis, Tianyin Xu, and Josep Torrellas

BACKGROUND

Radix Page Tables

- Standard implementation of a page table as a multi-level tree
- Walk the page table to find a page table entry
- Address translation with new emerging workloads has become a major performance bottleneck
    - Can account for 20-50 percent of application execution time
    - Page table walks may account for 20-40 percent of main memory access
- Need to find a scalable approach to this problem

Hashed Page Tables

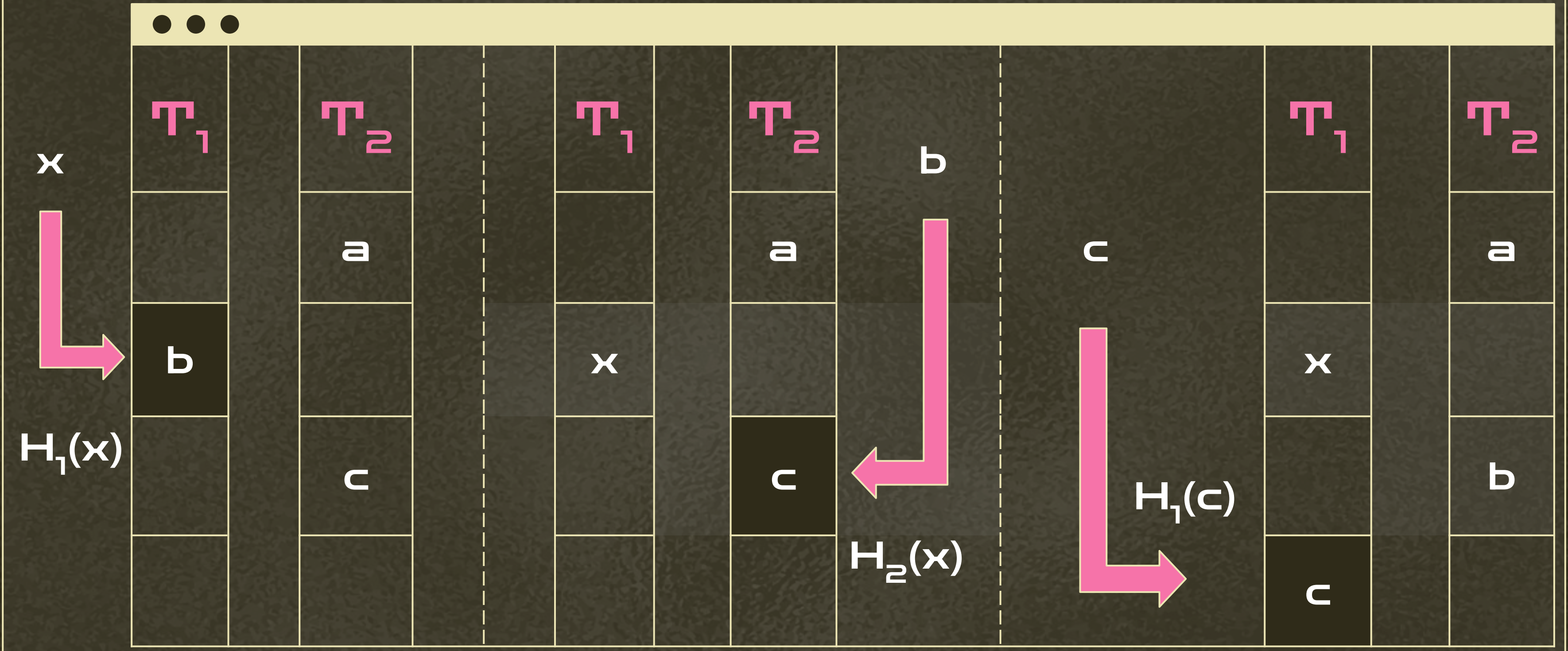
- Address translation involves hashing the virtual page number and using the hash key to index the page table
- Three limitations:
  - Loss of spatial locality in page table accesses
  - PTEs consume more space since each requires a hash tag
  - Need to handle hash collisions
- Hash collisions are the most significant of these concerns and remain unsolved
  - Main solution is resizing but is expensive
  - Global hash table
    - Cannot have multiple page sizes or page sharing
    - Linear scan of hash table to delete

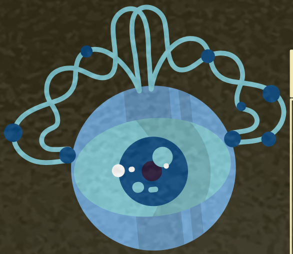
CUCKOO HASHING

Cuckoo Hashing

- Resolve collision by allowing an element to have multiple possible hashing locations while being stored in at most one of these locations at a time
- A *d*-ary cuckoo hash table consists of:
    - *d* ways $T_D = \{T_i : i \in 1..d\}$
    - *d* independent hash functions $H_D = \{H_i : i \in 1..d\}$
- Insertion, look-up, and deletion all take *O(d)* time

2-ary Cuckoo Hash Table

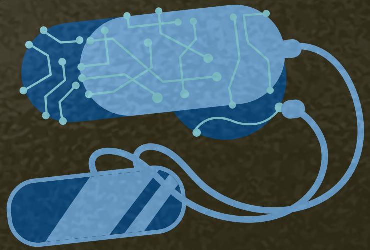




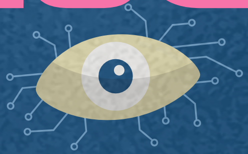
Resizing Cuckoo Hash Table is Expensive!

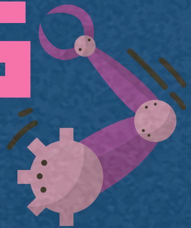
"During resizing, a look-up needs to perform $2 \times d$ accesses."

- When the occupancy of T_D reaches a Rehashing Threshold, a bigger *d*-ary cuckoo hash table (T'_D, H'_D) is allocated.
- Then, after each insert, a rehash occurs: removing an element from (T_D, H_D) and inserting it into (T'_D, H'_D).



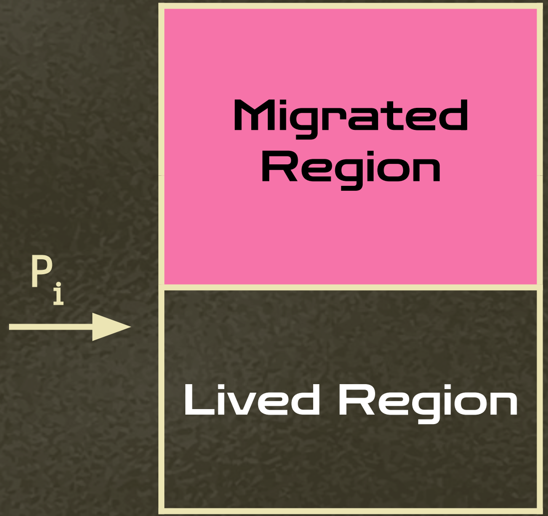
ELASTIC CUCKOO HASHING





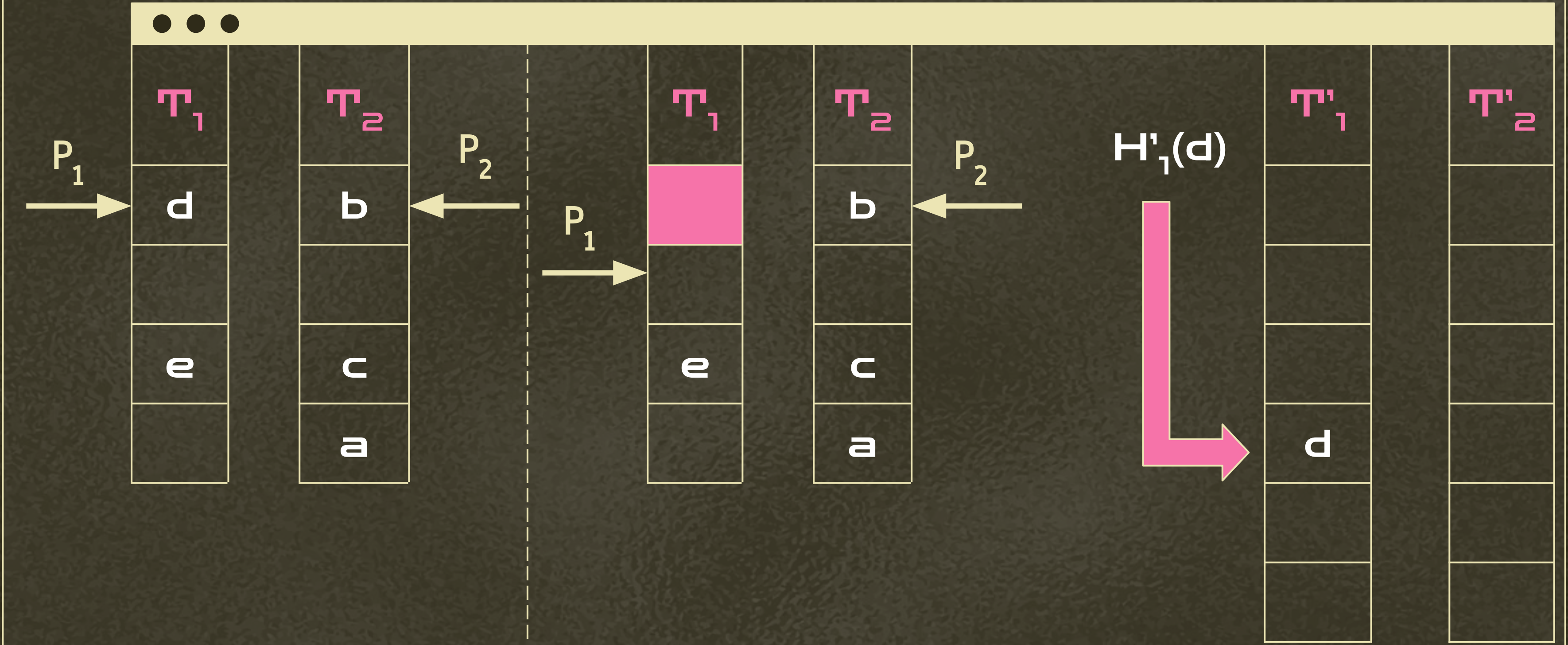
Elastic Cuckoo Hashing

- A *d-ary elastic* cuckoo hash table consists of:
  - *d* ways $T_D = \{T_i: i \in 1..d\}$; each T_i now has a Rehashing Pointer P_i which is initialized to 0
  - *d* rehashing pointers $P_D = \{P_i: i \in 1..d\}$
  - *d* independent hash functions $H_D = \{H_i: i \in 1..d\}$



DETAILED ALGORITHMS

Rehash



Look-up

function LOOK-UP(x):

for each way i in (T_D, H_D, P_D) **do:**

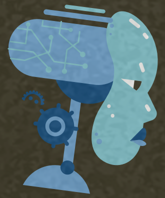
  for $H_i(x) < P_i$ **then:**

    if $T'_i[H'_i(x)] == x$ **then return true;**

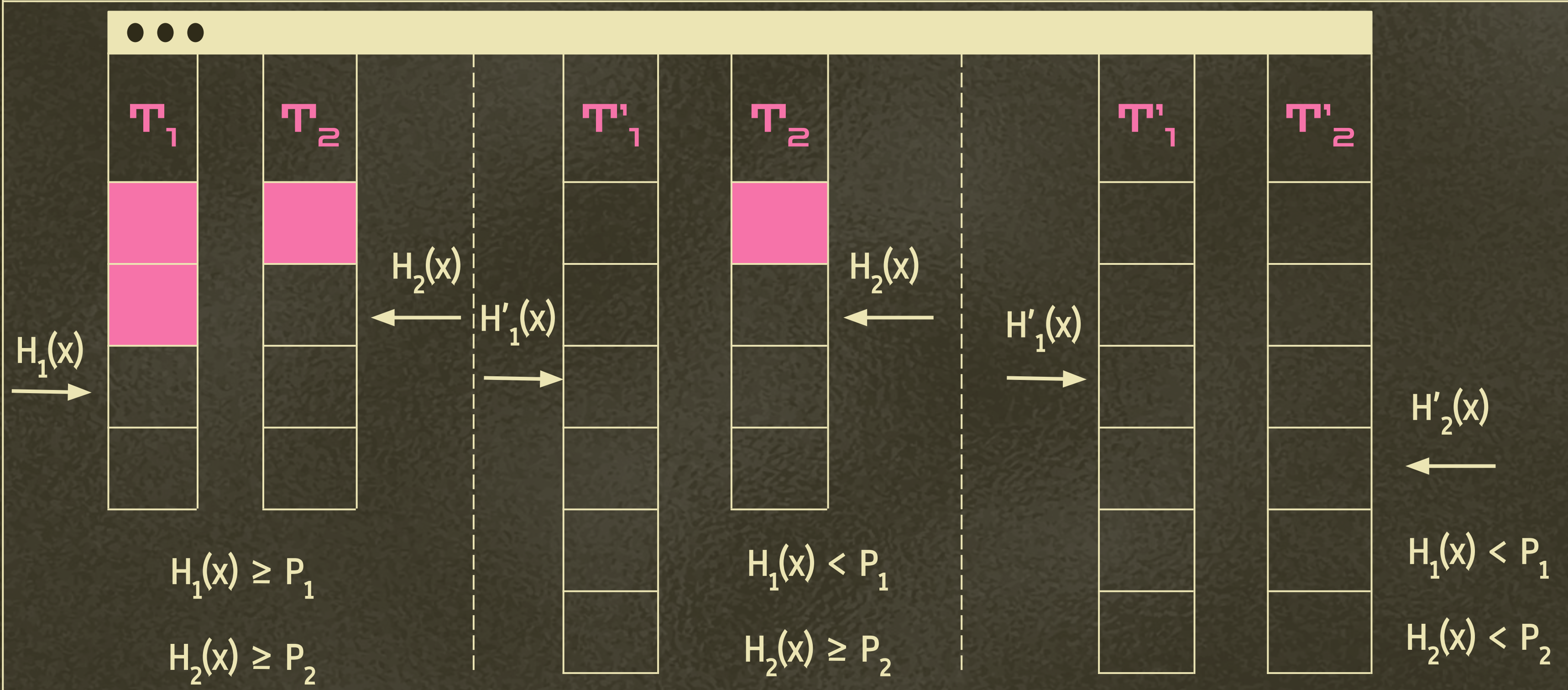
  else:

    if $T_i[H_i(x)] == x$ **then return true;**

return false



Look-up



T1
T2
H2(x)
H'1(x)
T'1
T2
H2(x)
H'1(x)
T'1
T'2
H1(x)
H'2(x)
H1(x) ≥ P1
H2(x) ≥ P2
H1(x) < P1
H2(x) ≥ P2
H1(x) < P1
H2(x) < P2

Insert

function INSERT(x):

$i \leftarrow$ Rand_Pick({1, ..., d})

for loop=1 to MAX_ATTEMPTS do:

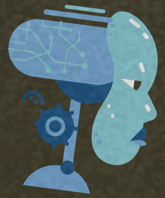
  if $H_i(x) < P_i$ then: $x \leftrightarrow T'_i[H'_i(x)]$

  else: $x \leftrightarrow T_i[H_i(x)]$

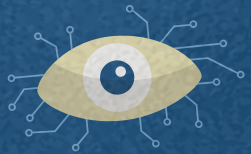
  if $x == \varnothing$ then return true;

  $i \leftarrow$ Rand_Pick({1, ..., d} - {i})

return false

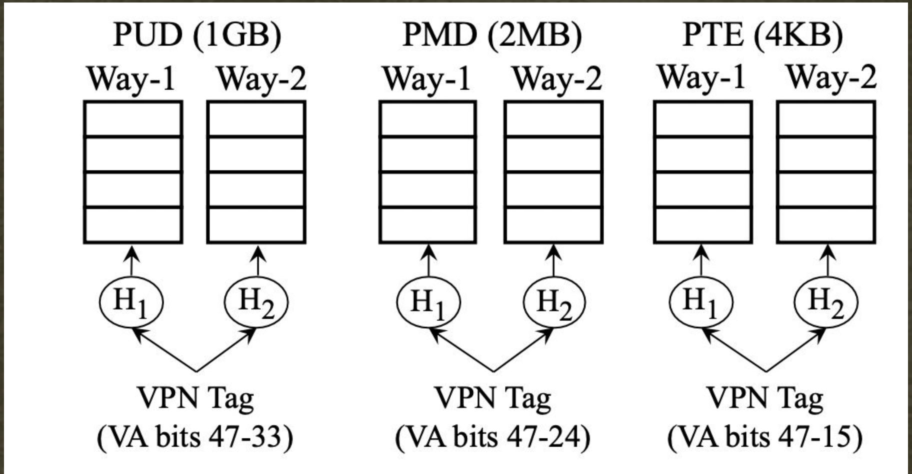


Design



Organization

- *d*-ary elastic cuckoo hash table indexed by hashing a VPN tag
- Each process has one elastic cuckoo page table for each page size
- Exploits 2 levels of parallelism

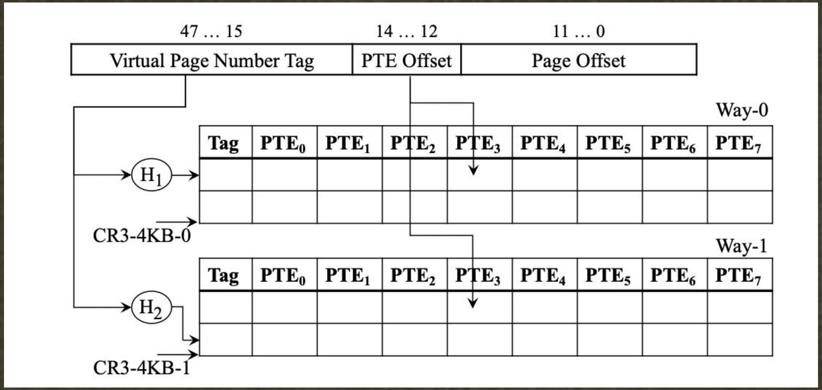


Page Table Entry

- Want entries in an elastic cuckoo page table to support clustering and compaction
  - Improves spatial locality
- A single hash table entry contains a VPN tag and multiple consecutive physical page translation entries packed together
  - Clustering factor depends on the size of a cache line

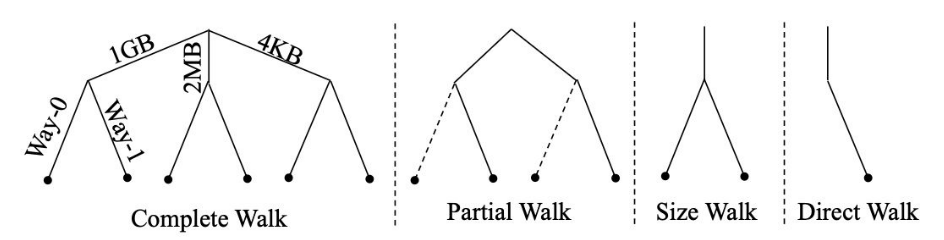
Cuckoo Walk

- Method for address translation in an elastic cuckoo page table
  - Parallel walk that may look up multiple hash tables in parallel
- Hardware page table walker takes a VPN tag and hashes it using the hash functions of different hash tables
  - Uses the resulting keys to index multiple hash tables in parallel



Cuckoo Walk Tables

- A cuckoo walk may have to lookup the d ways of S elastic cuckoo page tables
    - S x d parallel lookups
- To reduce the number of lookups, use cuckoo walk tables
- Contain information about which way of which elastic cuckoo page table should be accessed
- 4 types of walks depending on the information contained in the CWTs
    - Complete walk: no information
    - Partial walk: page is not of a given size
    - Size walk: page is of a given size
    - Direct walk: page is of a given size and way of cuckoo hash is known



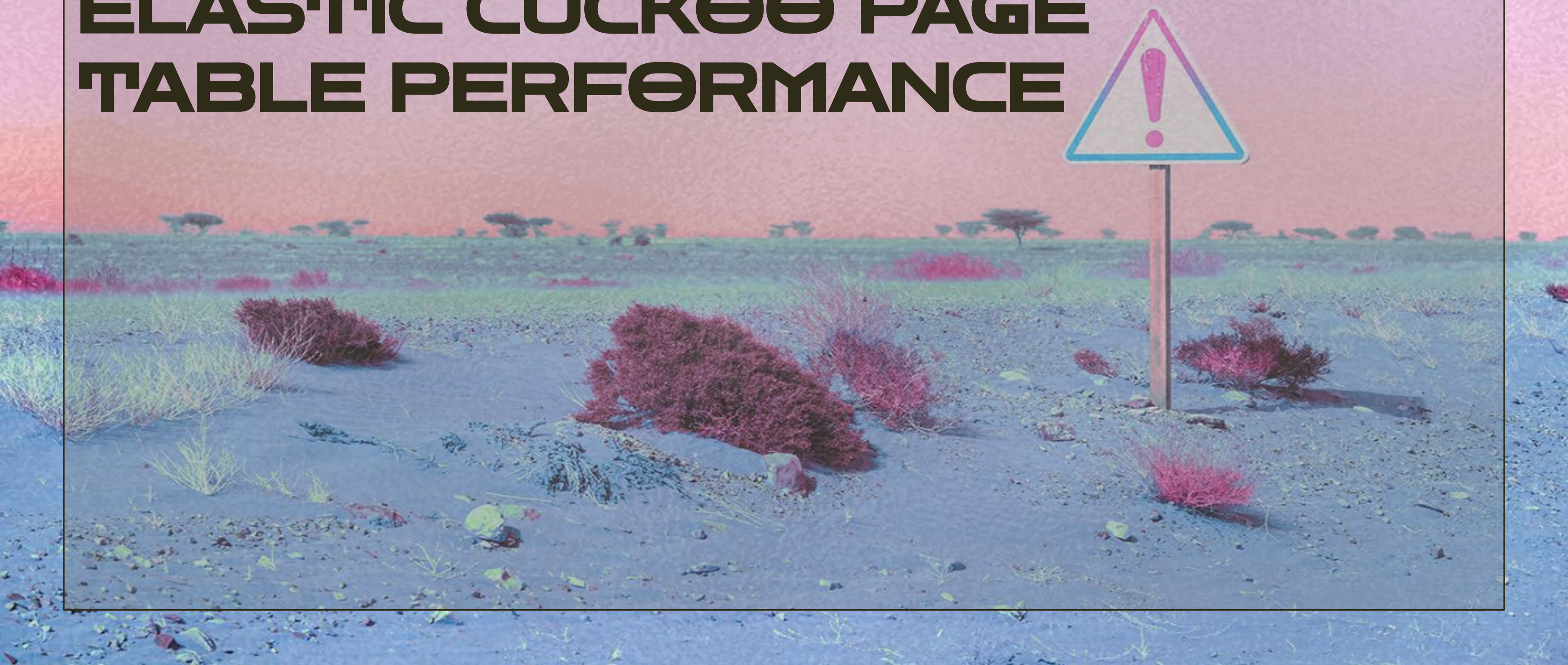
Cuckoo Walk Table Entries

- VPN tag and consecutive section headers
  - Section headers provide information about a given virtual memory section
  - Section is a range of virtual memory address space translated by one entry in the corresponding elastic cuckoo page table

Cuckoo Walk Caches

- Cuckoo walk tables reside in memory
- To speed up walking, cache some of the entries in cuckoo walk caches
- These caches are different from the caches in radix page tables since they do not store PTEs
  - Instead they store page size and way information
  - Two implications:
    - On a CWC miss, page walker can proceed to access the target page table entry right away
    - CWC entry is small so that CWCs have small size and high hit rate
    - PMD-CWC section header covers a 16MB region with 4 bits while the traditional PMD cache covers 2MB with 64 bits

ELASTIC CUCKOO PAGE TABLE PERFORMANCE



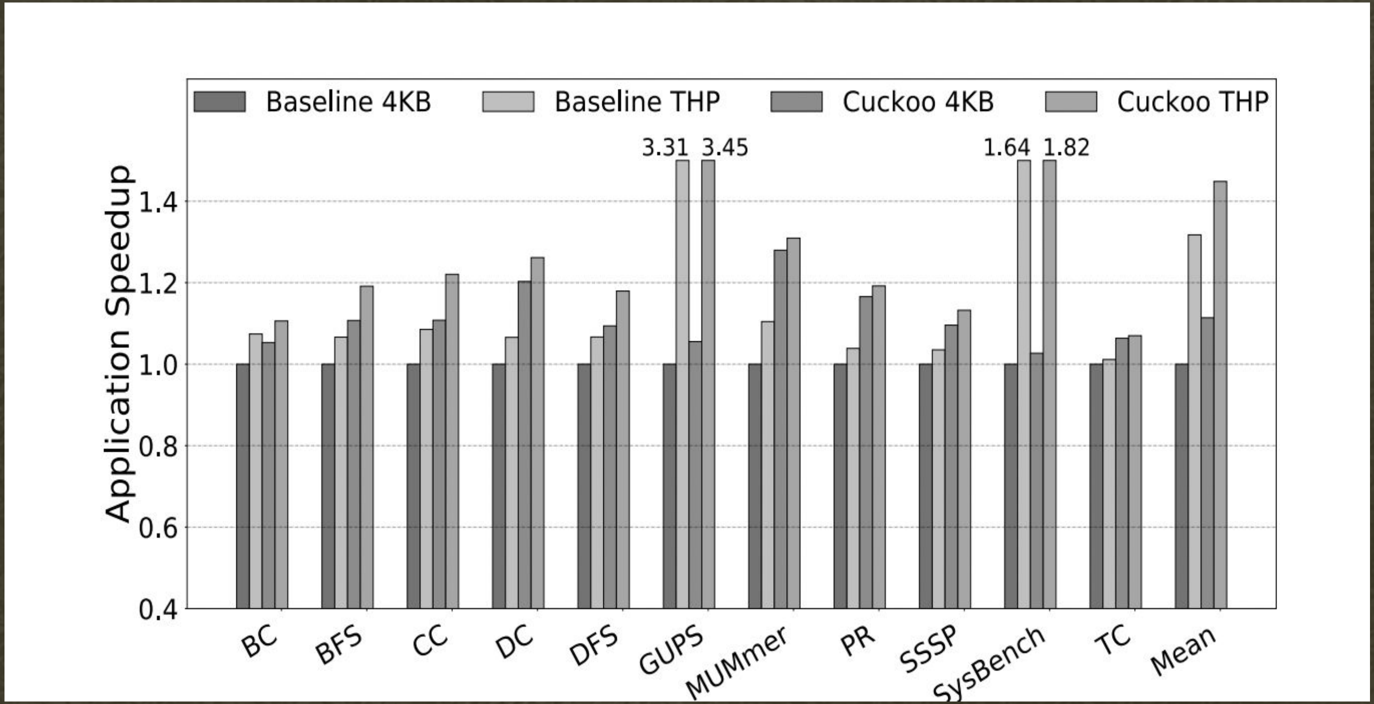
Elastic Cuckoo Page Table Performance

- Evaluate performances of 4 systems across various workloads in graph analytics, bioinformatics, high performance computing, and system domains:
  - Baseline 4KB and Cuckoo 4KB: with only 4KB pages
  - Baseline THP and Cuckoo THP: with multiple page sizes by enabling Transparent Huge Pages in Linux

Application Speedup

Cuckoo 4KB results in application speedup of **3-28%** over **Baseline 4KB**.

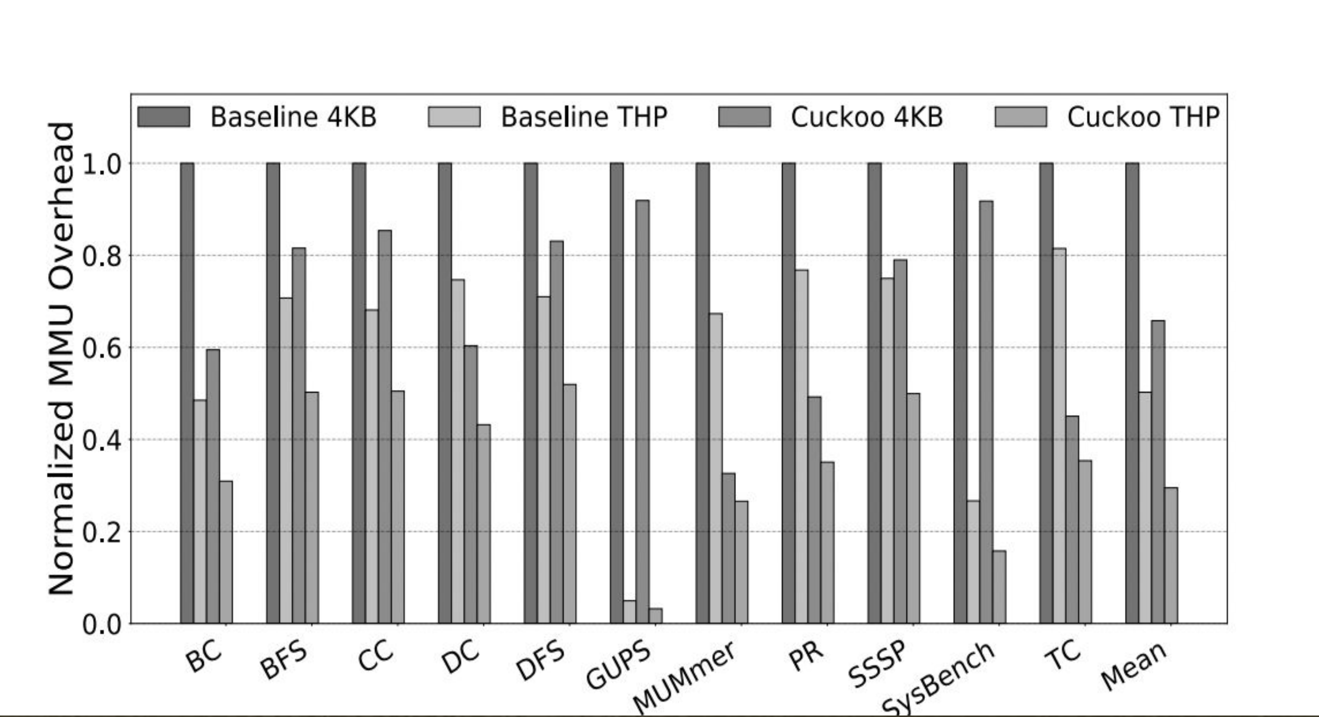
The speedup of **Cuckoo THP** over **Baseline THP** is **3-18%**.



MMU Overhead Reduction

Cuckoo 4KB reduces MMU overhead of **Baseline 4KB** by **34%**.

Cuckoo THP's overhead is **41%** lower than **Baseline THP**.



THANKS!

Do you have any questions?