Every Walk's a Hit: Making Page Walks Single-Access Cache Hits

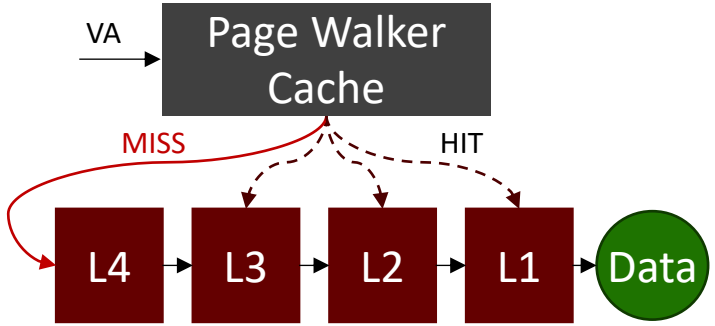
Chang Hyun Park, Ilias Vougioukas,
Andreas Sandberg, and David Black-Schaffer

UPPSALA
UNIVERSITET

arm Research

Today's Virtual Memory Translations don't scale

- TLB reach is not scaling with DRAM
  - 8 MB/4 GB when using 4 KB/2 MB pages[1]
- TLB misses resolved with Page Table Walks
  - Four serial memory accesses required
- Page Walk Caches help by skipping levels
- 1.1 ~ 2.5 (avg 1.5) memory accesses per TLB miss
  - Workloads with memory sizes up to 8GB



Coverage[2]	# mem access per walk
48 MB	1
4 GB	2
2 TB	3
> 2 TB	4

Page Walker Cache coverages don't scale!
E.g. Larger memory sizes, 5-level page tables

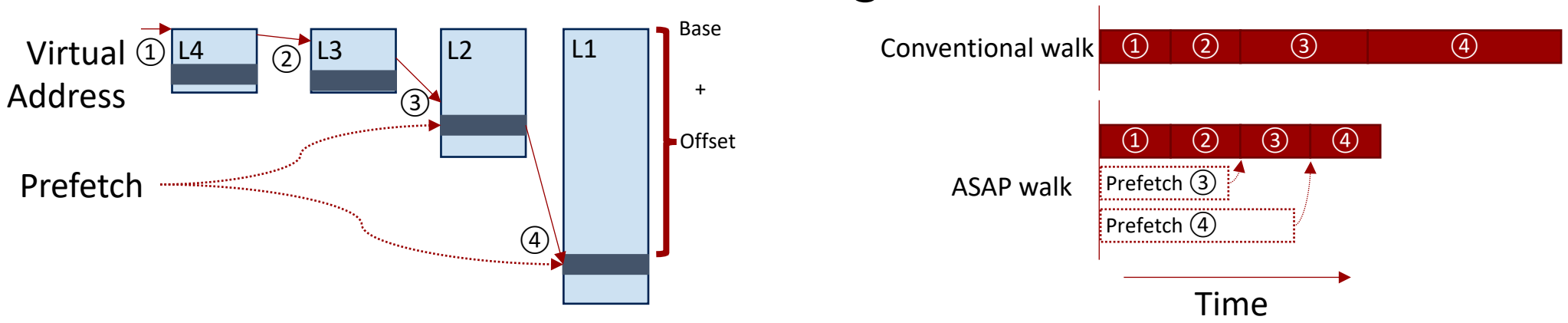
UPPSALA UNIVERSITET

[1] AMD Zen3 uArch
[2] Van Schaik et al., Vrije Universiteit Tech report 2017. Coverage based on Intel Skylake uArch

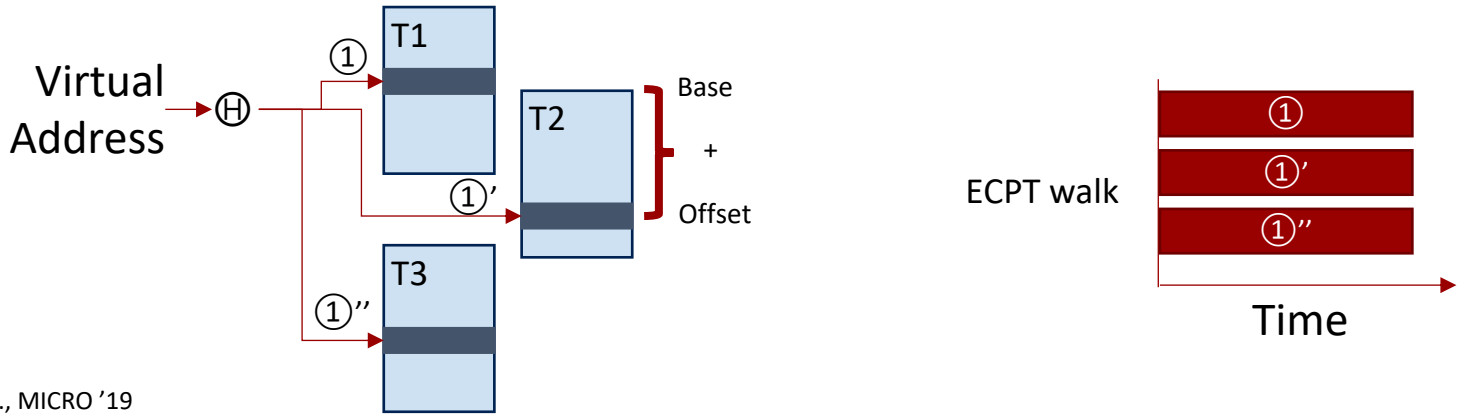
arm Research

Past Proposals: Require large contiguous physical memory

- Address Translation with Prefetching (ASAP)[1] - Prefetch leaf nodes



- Elastic Cuckoo Page Tables (ECPT)[2] – multi-way hash PTs



UPPSALA UNIVERSITET

[1] Margaritov et al., MICRO '19
[2] Skarlatos et al., ASPLOS '20

arm Research

Challenge: Contiguous Physical Memory

- Page table allocation is a critical task
  - Cannot fail, cannot take too long (e.g. tail latency!)
- Difficult to get large physically contiguous memory allocations
  - Availability of contiguous memory is not guaranteed
- Implications: guarantee contiguous allocations
  or suffer latency spikes or PT allocation failures

Can we have a scalable page walk that does not need contiguity guarantees?

UPPSALA UNIVERSITET

arm Research

Our Work

Problem: Page table walks → 1.1-2.5 accesses (incl. DRAM), Large physically contiguous memory

Solution: Making Page Table Walks Single-access Cache Hits

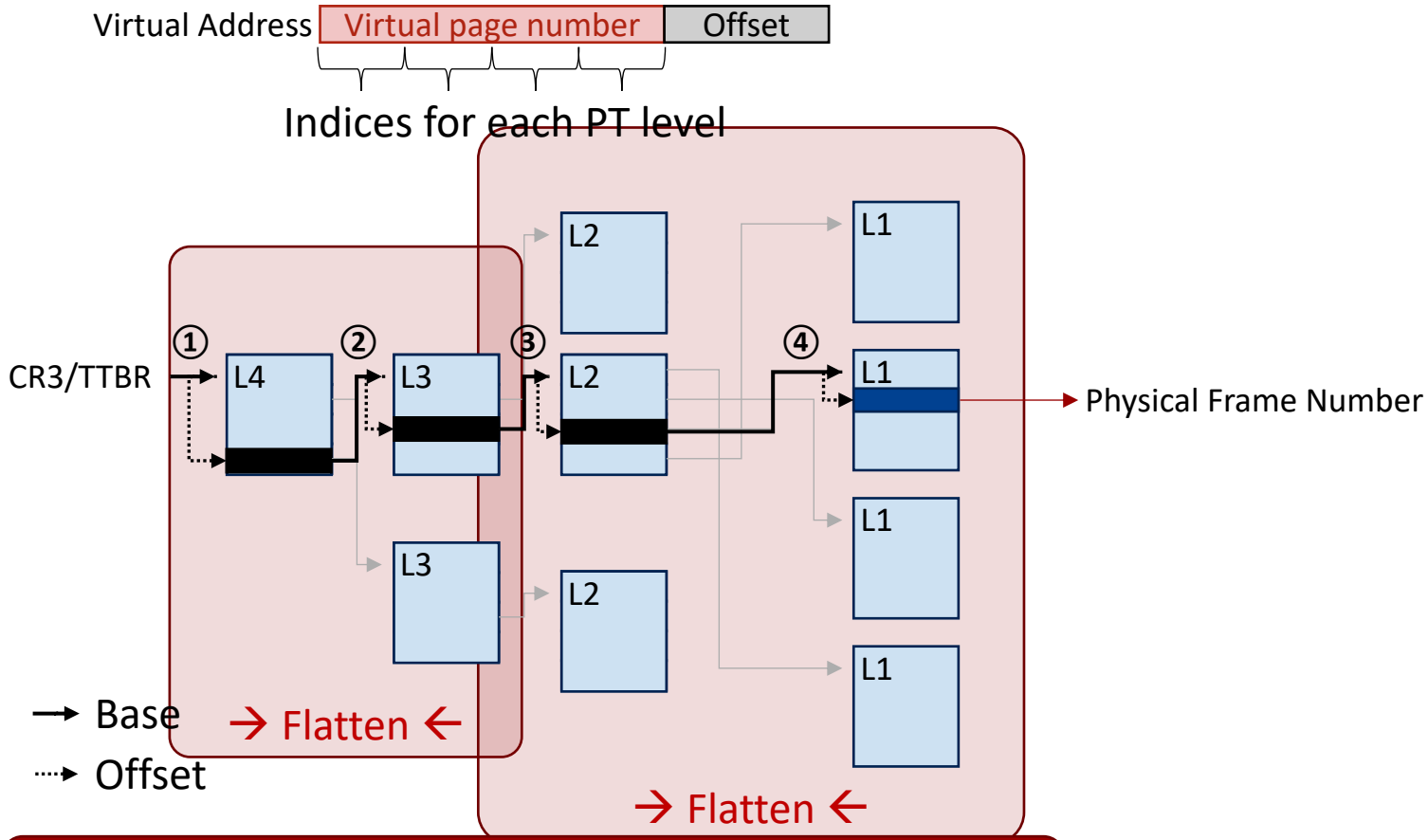
Flatten the page table

Prioritize the page table in the cache

UPPSALA UNIVERSITET

arm Research

Conventional (4-level tree) Page Tables



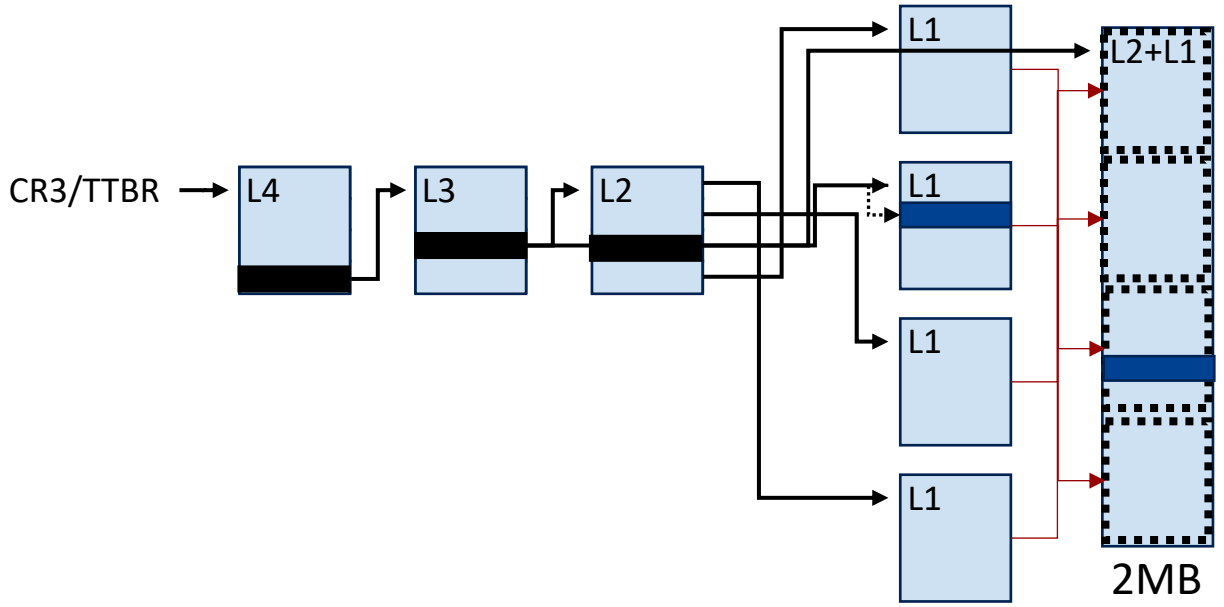
Height of tree == Number of serial memory accesses

① + ② + ③ + ④ → 4 serial memory accesses*

Flatten the tree → fatter nodes, shorter tree

* Worst case: Page wal

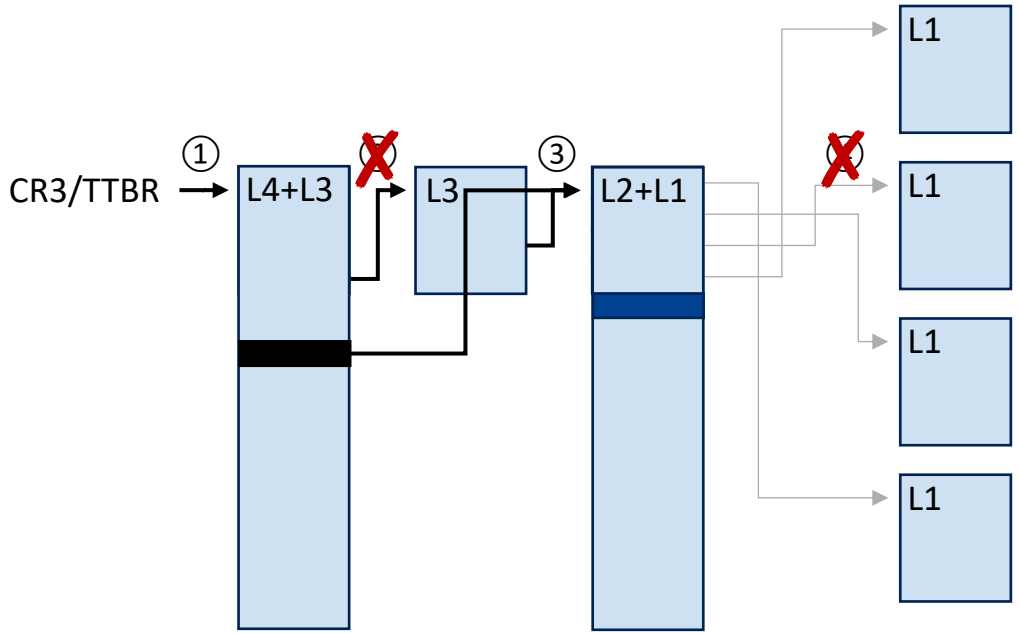
Flattening two levels



Gather the scattered L1 Page tables of a L2 into one L2+L1 flattened page table

Flattened nodes stored in 2MB pages: leverages existing OS 2MB support

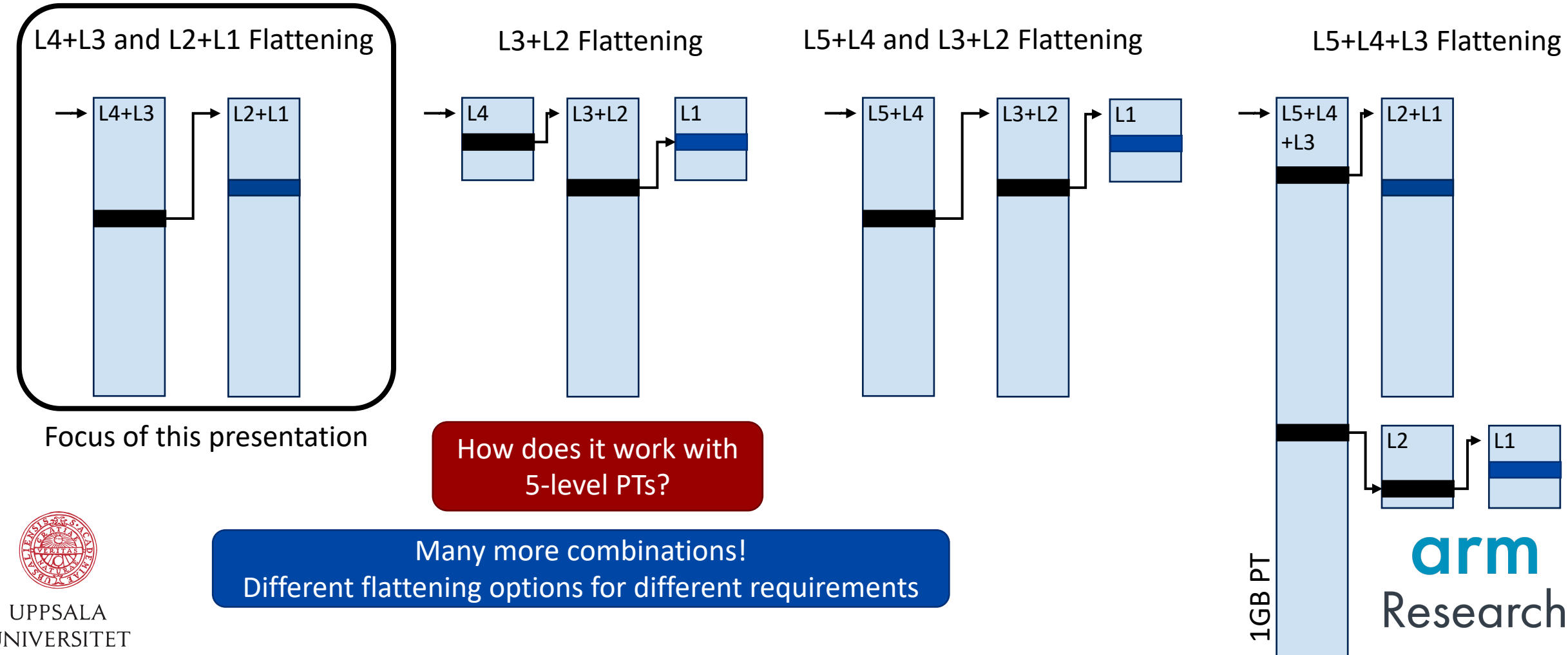
Flattening the Page Table



Flattening reduces serial memory accesses 4 → 2

Trade-off memory space for shorter page table walks

Various Combinations of Flattening



Focus of this presentation

How does it work with 5-level PTs?

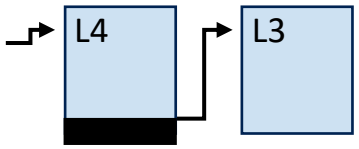
Many more combinations!
Different flattening options for different requirements

UPPSALA UNIVERSITET

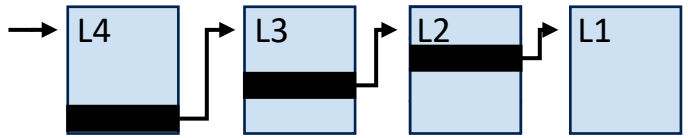
arm Research

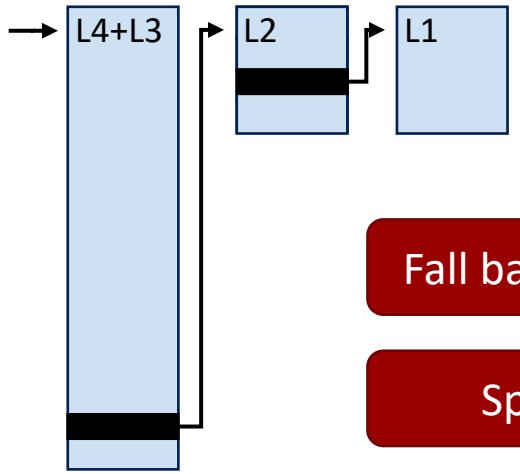
What if a 2MB Allocation Fails?

1. Allocating root node



2. Allocating L2+L1

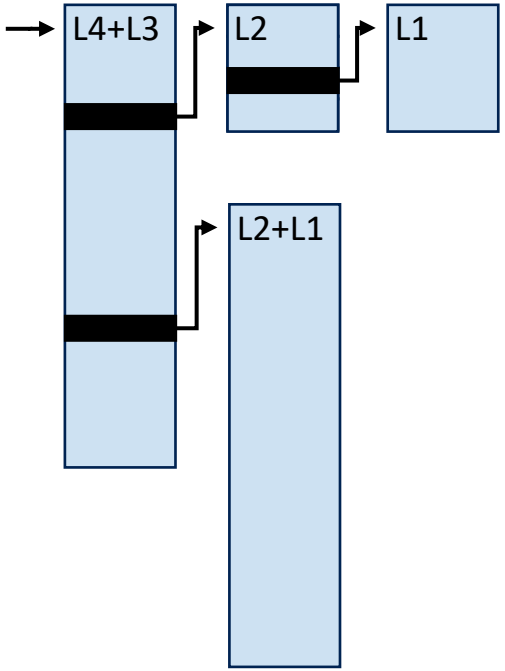




Fall back to allocating 4KB PTs

Specify next PT is 4KB

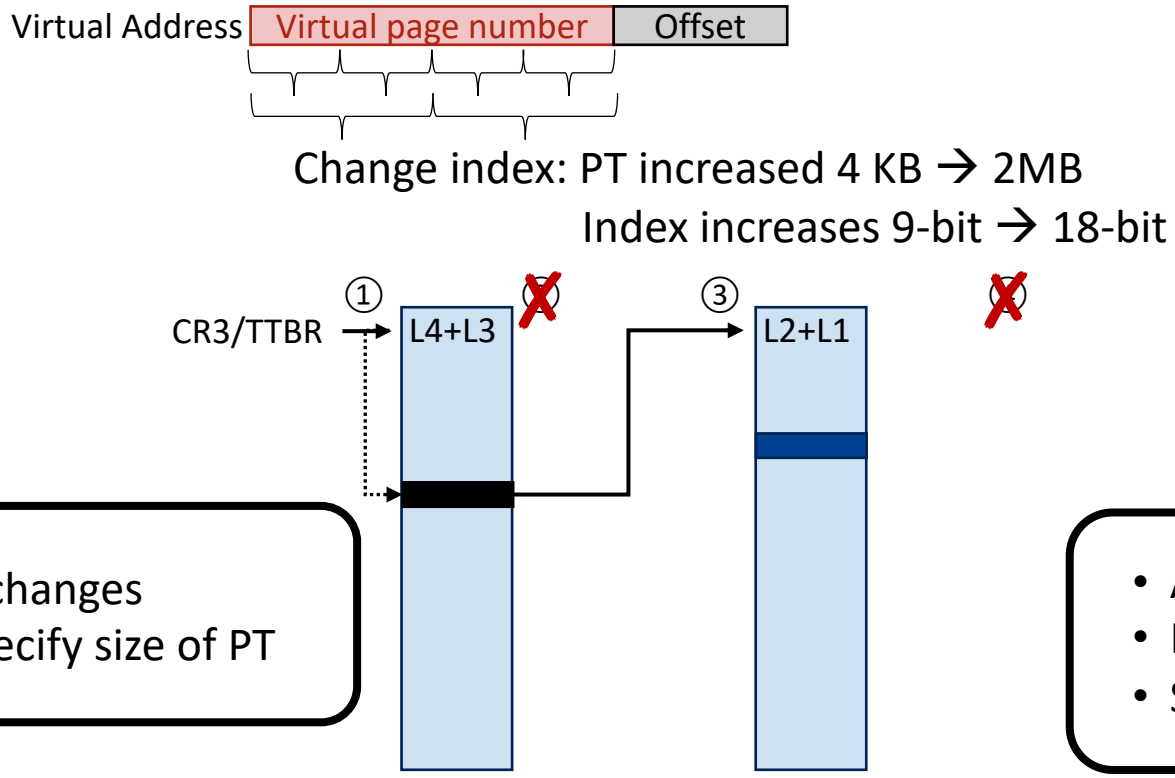
Note: Flattened L2+L1 may coexist with unflattened L2 and L1 PTs



UPPSALA UNIVERSITET

arm Research

HW and SW Changes



HW

- Virtual address indexing changes
- Pointers to PT need to specify size of PT

SW

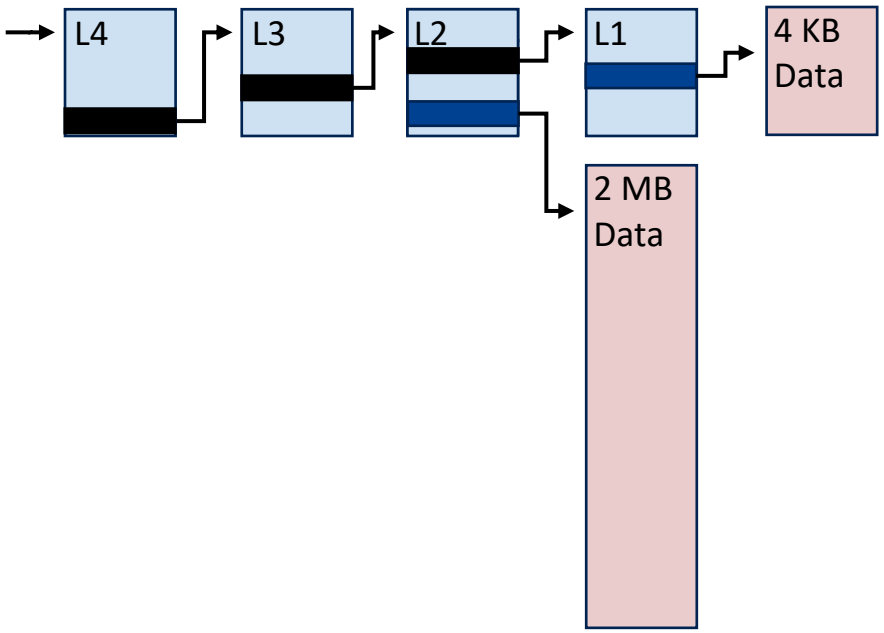
- Allocate 2MB PTs
- Mark pointers with appropriate size
- Small changes (+614/-109 LOC)

UPPSALA UNIVERSITET

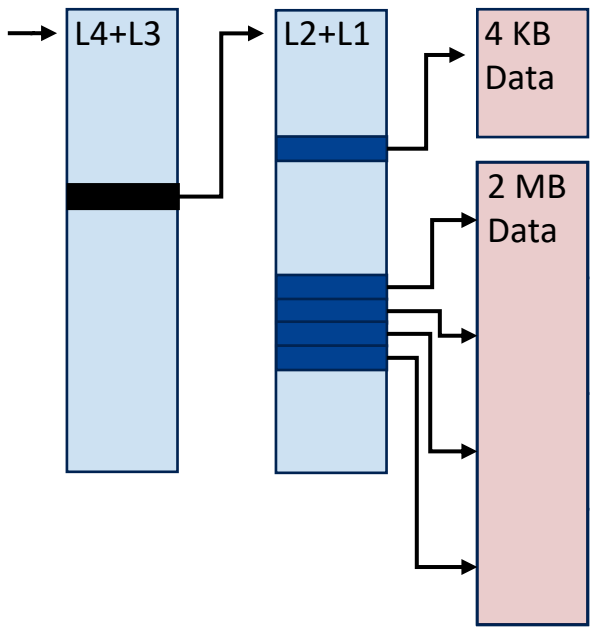
arm Research

Flattening and 2 MB data pages

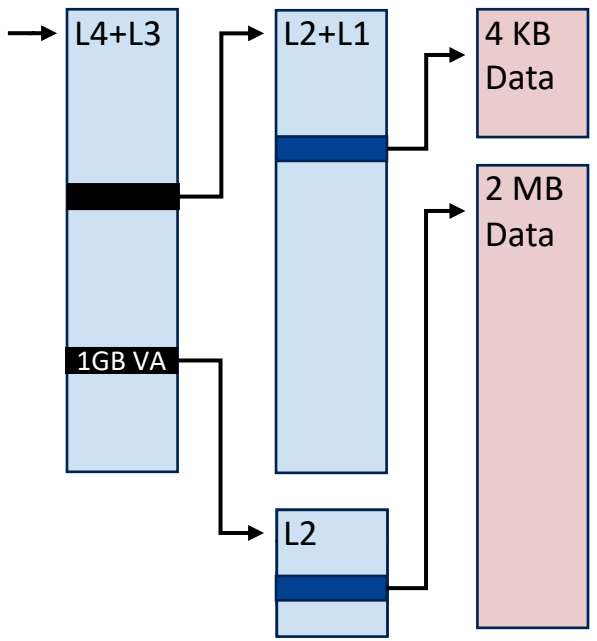
Conventional PT and 2 MB Data



Replicated 2MB entries



Flattening + non-flattening



2 MB Data pointed by L2 PT entries

Flattened Entries can point to each sub-4 KB data page of 2 MB page

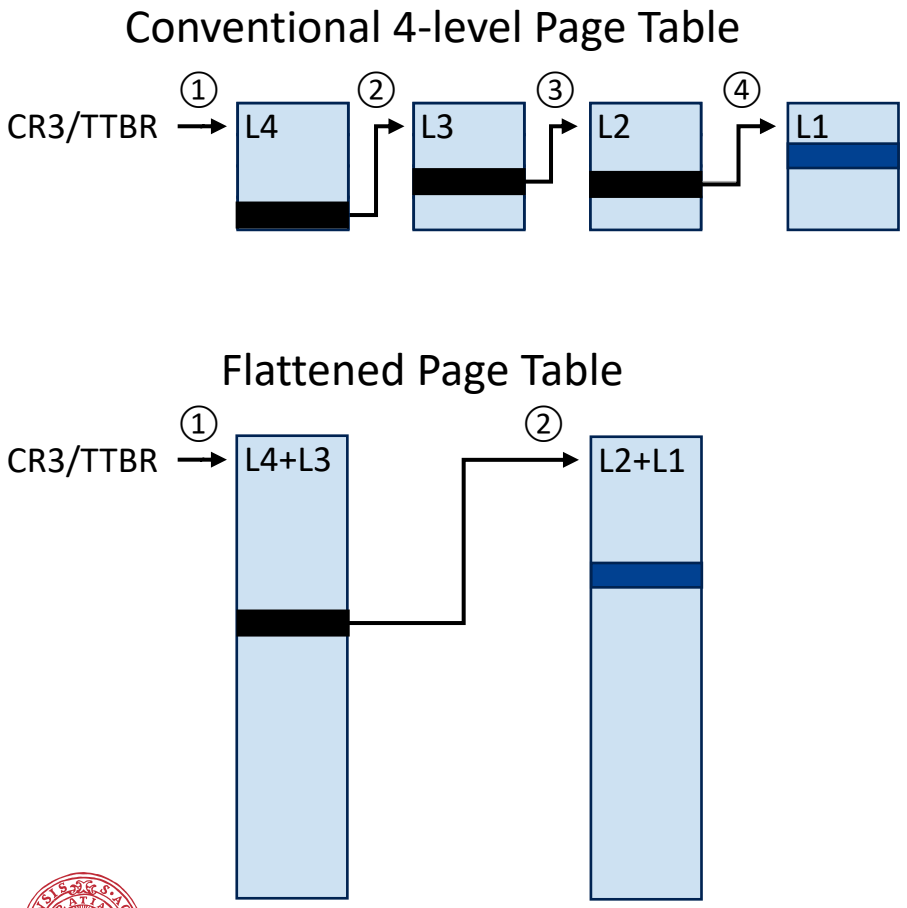
Requires more PT entries, bad cache performance

Don’t flatten 1 GB regions with many large pages!

UPPSALA UNIVERSITET

arm Research

Summary: Flattening



- Flattening reduces serial memory accesses
  - 4 → 2 memory accesses
  - In practice: 1.5 → 1 memory accesses (PWC)
- Memory access are still missing the cache
  - Long latency accesses to the DRAM

Can we do something about the DRAM accesses?

UPPSALA UNIVERSITET

arm Research

Prioritize Keeping Page Table in the Cache

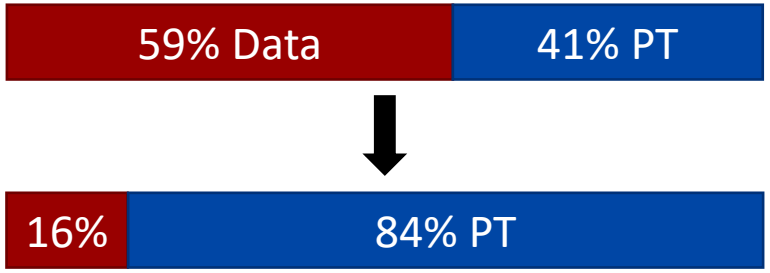
- Workloads with high TLB miss show high data cache miss ratio
  - L2: 95% miss ratio, L3: 80% miss ratio

Data caching not effective
→ Cache page table instead!

Insight: PTE covers 64 cachelines
→ 64x as likely to see reuse

- Simple solution: bias replacement to keep page table entries
  - 99x more likely to evict data

GUPS* L3 Occupancy



L3 Miss ratio	Data	PT Walk
Normal	99.1%	54.5%
PT Prioritization	99.5%	**14.7%**

Data miss ratio hardly changed,
walker miss rate improved by 3.7x

UPPSALA UNIVERSITET

* Workload that shows significant impact. Cherry-picked to emphasize behavior of prioritization

arm Research

Evaluation Methodology

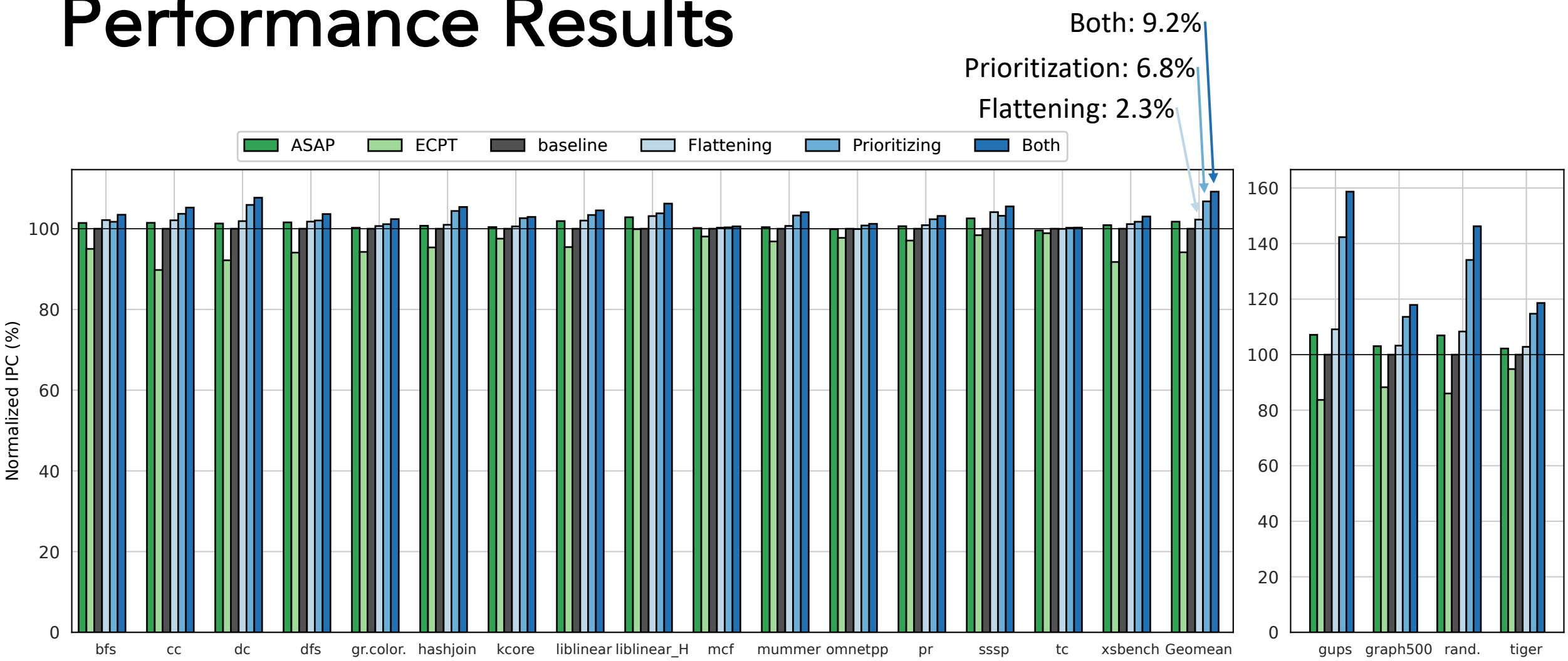
- Simulation infrastructure
  - Gem5
  - System-call emulation mode
- Graph, bioinformatics, SPEC CPU, linear classifiers, microbenchmarks

Component	Configuration
Processor	2GHz, OoO x86
Caches	L1 I/D: 32KB, 8-way L2: 256KB, 8-way L3: 16MB, 8-way
Memory	DDR4-2400, 4 channels
L1 TLB	64-entry, 4-way, 4KB 32-entry, 4-way, 2MB
L2 TLB	1,536-entry, 12 way (Shared by 4KB and 2MB)
Page Structure Cache	4-entry L4 cache entries 4-entry L3 cache entries 24-entry L2 cache entries

UPPSALA UNIVERSITET

arm Research

Performance Results

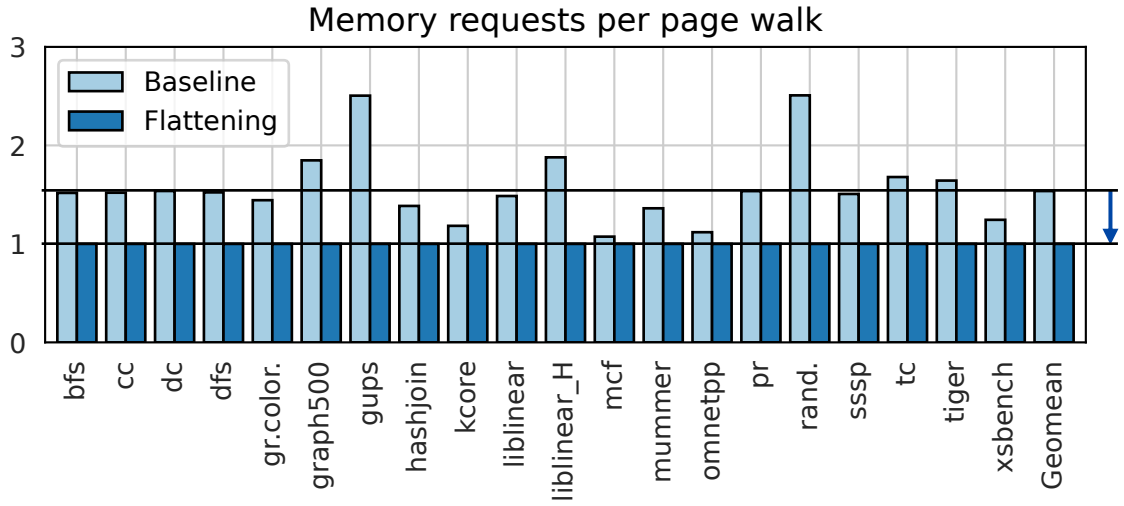


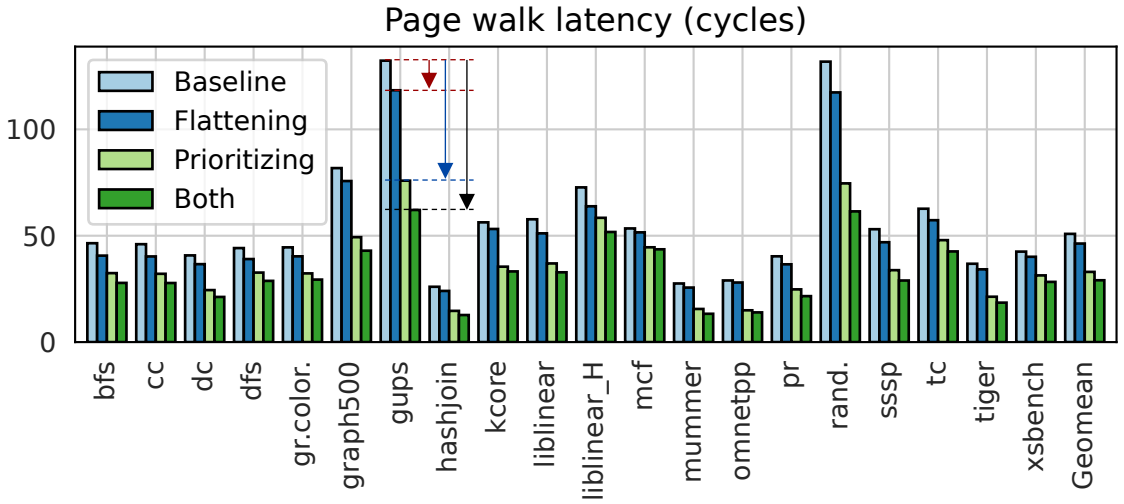
Outperforms prior work using
smaller chunks (2MB) of contiguous memory

UPPSALA
UNIVERSITET

arm
Research

Looking further into the results





1.5 → 1.0 (avg)

Flattening alone (little benefit)
50.9 → 46.3 cycles (avg)

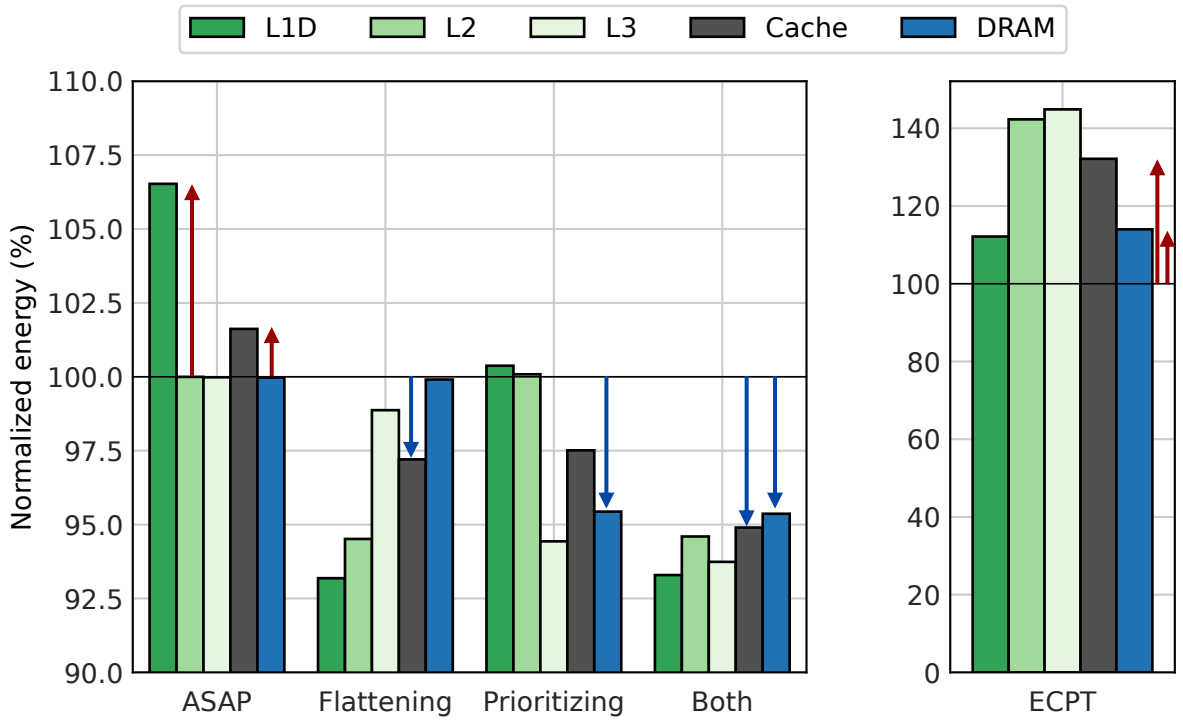
Prioritization
50.9 → 33.0 cycles (avg)

Both
50.9 → 29.1 cycles (avg)

UPPSALA
UNIVERSITET

arm
Research

Energy implications



ASAP issues prefetches for leaf page table entries
→ Increased cache accesses

ECPT issues multiple concurrent memory accesses,
→ Increased cache and memory accesses

Flattening reduces number of accesses per walk
→ Reduced cache accesses

Prioritization increases cache hits for walks
→ Reduced memory accesses

Flattening and Prioritization reduces
both memory and cache accesses

UPPSALA
UNIVERSITET

arm
Research

More in the Paper

- Flattening
  - How we support large page data
  - Various flattening options
  - Flattening for virtualization
  - **Supporting recursive PTs (Windows)**
- Results
  - Multi-core results
  - Virtualization results
  - Performance results with different cache size
  - **Case study: flattening for mobile systems**

UPPSALA
UNIVERSITET

arm
Research

Conclusion

- Page table walks need to be short
  - Current page walk caching does not scale
  - Prior work require large contiguous physical memory and has no fallback path

- Flattening and Prioritizing → Single-access cache hitting page walks
  - **Simple:** Uses existing large pages for the page table
  - **Practical:** Provides graceful fallback when large pages are not available
  - **Results:** Performance gains 9.2% (14.0% in virtualized systems)
    Energy reductions (5.1% cache, 4.7% memory)

UPPSALA
UNIVERSITET

arm
Research

Every Walk's a Hit: Making Page Walks Single-Access Cache Hits

Chang.Hyun.Park@it.uu.se, Ilias.Vougioukas@arm.com,
Andreas.Sandberg@arm.com, and David.Black-Schaffer@it.uu.se

UPPSALA
UNIVERSITET

arm Research