Transparent OS Support for Variable Translation Sizes

Stratos Psomadakis, Georgios Goumas
National Technical University of Athens

National Technical University of Athens CSLab AERO Funded by the European Union

Summary

Problem: Address translation overheads exacerbated by 5-level and nested paging.

2MB pages (THP): Diminishing returns as:

- the working sets continue to grow.
- Fragmentation limits effective coverage.

1GB pages: Not as widely supported, harder to use, harder to allocate when memory gets fragmented.

Our proposal

- Exploit the **intermediate translation sizes** available on ARMv8-A and RISC-V.
- Enhance the OS memory manager to **transparently support** these variable translation sizes.

ARMv8-A Intermediate Translation Sizes

Contig Bit: L1 (PTE) and L2 (PMD) paging structures include a contig bit, which when set in 16 consecutive suitably aligned entries, allows the TLB to cache them as a single translation entry.

Supported Intermediate Sizes:

- 16x L1 (PTE) 4KB entries, coalesced to a single 64KB translation,
- 16x L2 (PMD) 2MB entries, coalesced to a single 32MB translation.

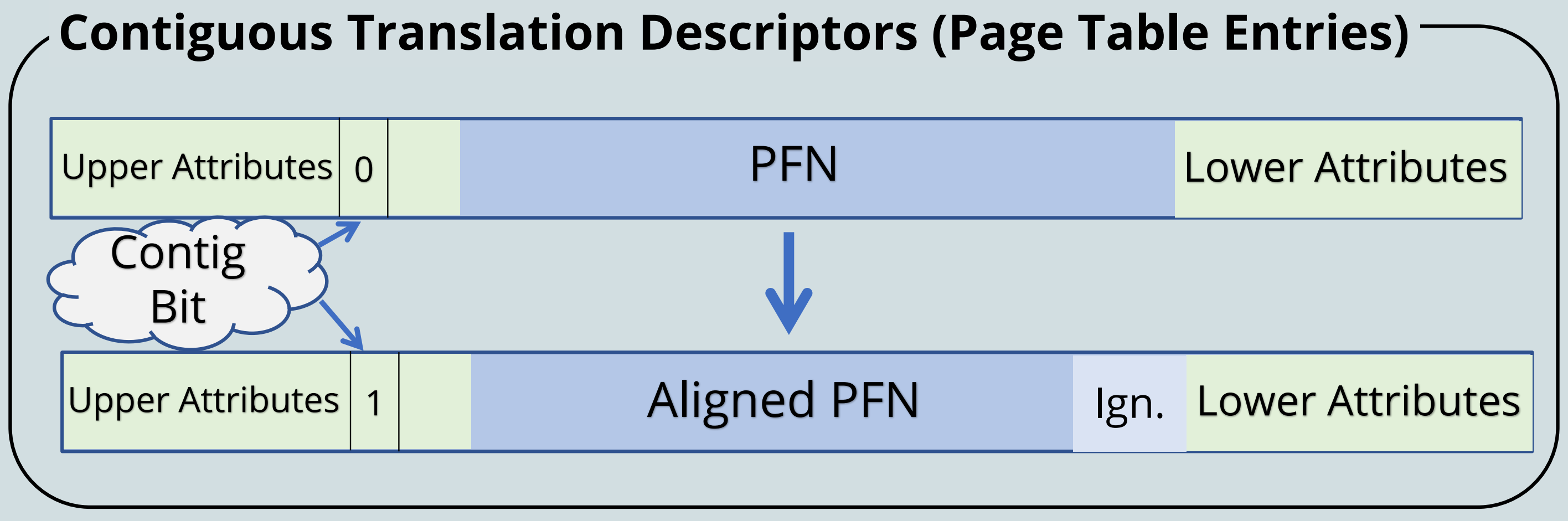
HW-supported Translations Sizes

Base	Large	Intermediate
4KB	2MB, 1GB	64KB, 32MB

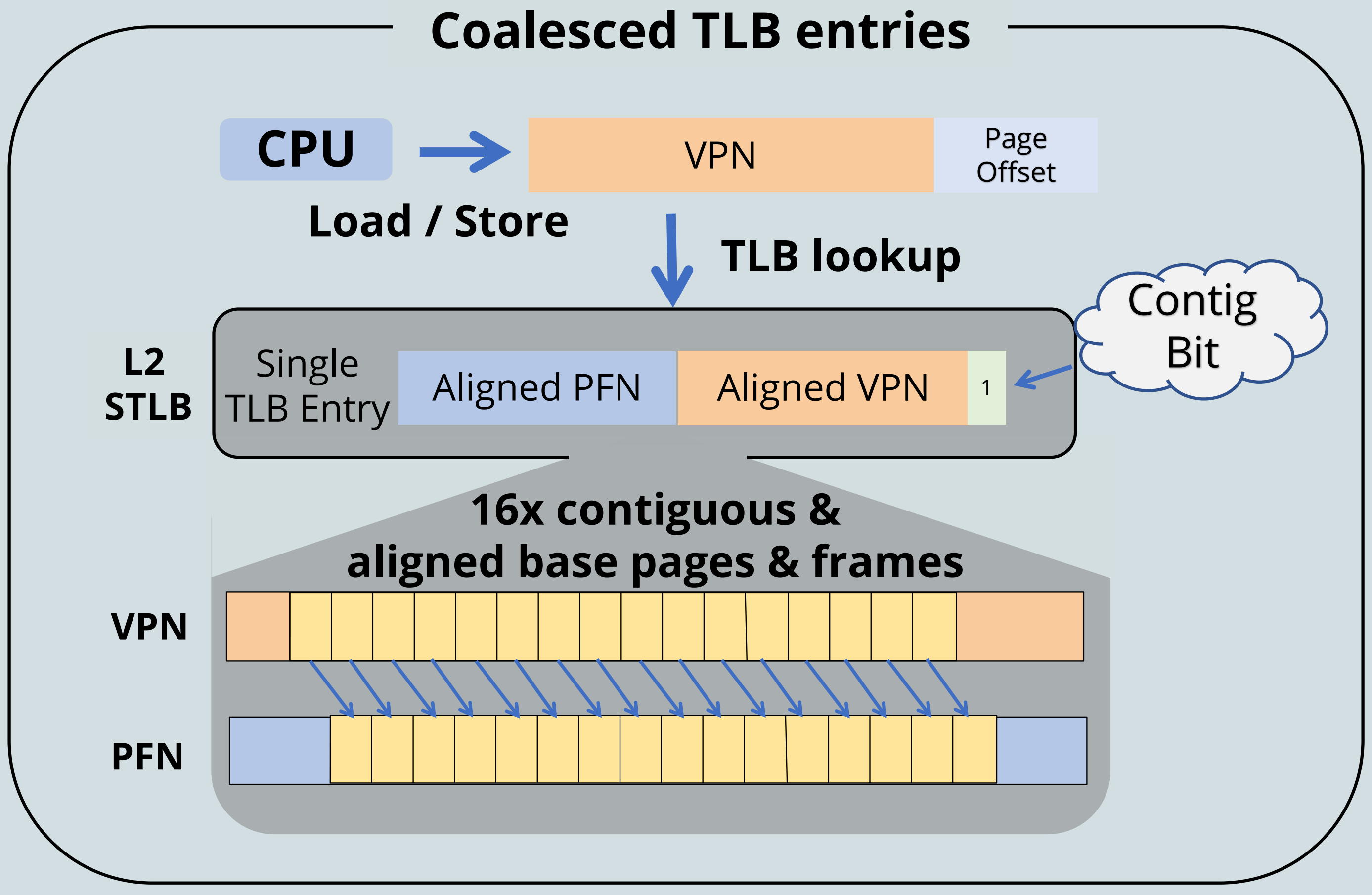
✓ Transparent OS Support (THP)

✗ Requires pre-allocation (Hugetlbfs)

Contiguous Translation Descriptors (Page Table Entries)



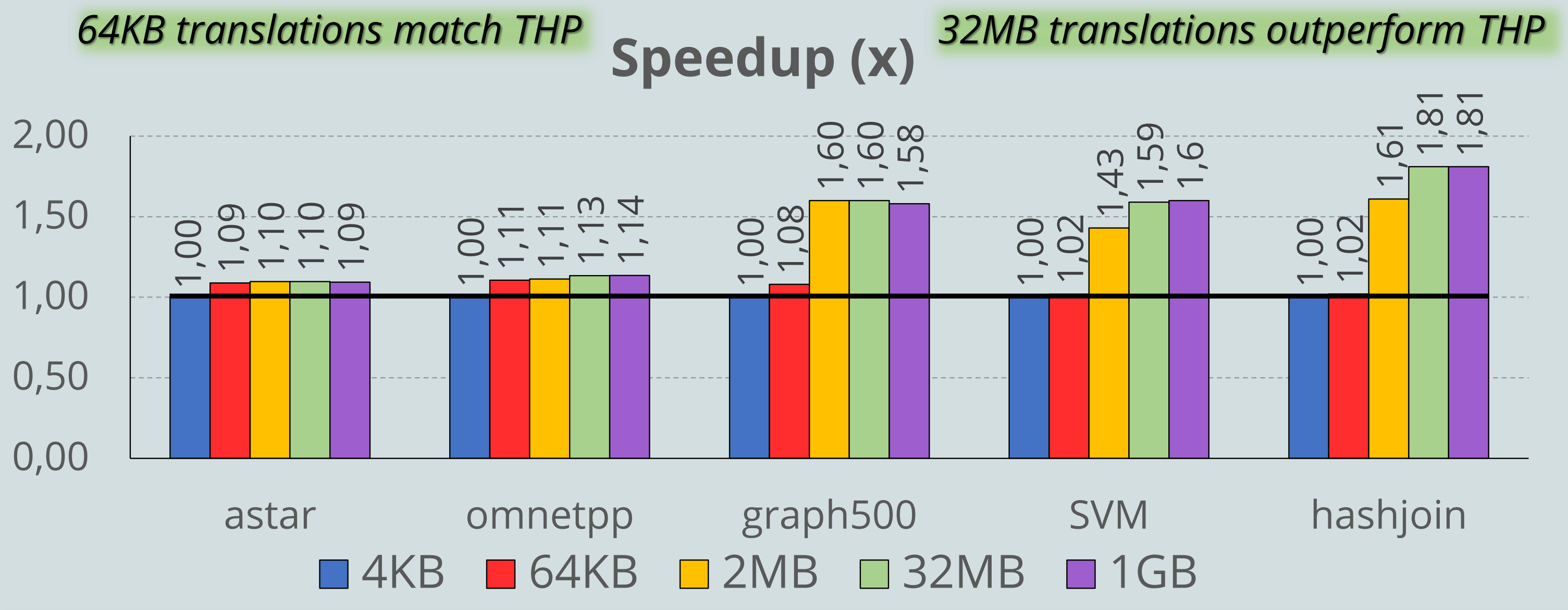
Coalesced TLB entries



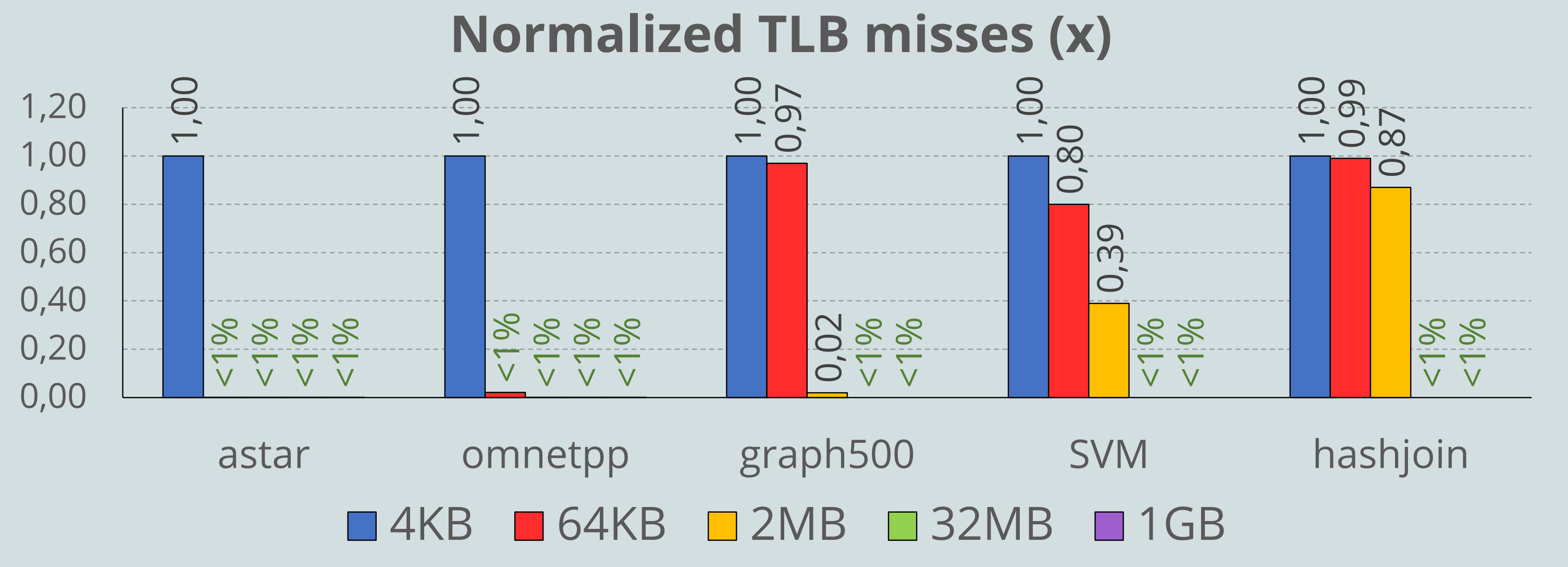
Intermediate Translation Sizes: Potential and Challenges

64KB translations match THP

32MB translations outperform THP



Hugetlbfs performance normalized to 4KB,running on Ampere Altra (ARMv8.2-A)



Observations

On-par performance to larger sizes with *less strict alignment and fragmentation requirements*.

- ✓ *64KB translations* eliminate the AT overhead for irregular workloads with smaller footprints (*astar* and *omnetp*).
- ✓ *32MB translations* improve performance by up to *20% vs THP*, match the performance of 1GB, for big memory workloads. For *SVM*, they use *16% less memory vs 1GB*.

Combining 64KB with larger sizes could better utilize fragmented memory.

Limitations

Intermediate sizes are *supported on Linux only via Hugetlbfs*, which:

1. requires *memory pre-allocation*,
   a. incurring significant *run-time overheads*, e.g. *50%* for *hashjoin*, with 32MB huge pages,
   b. making reserved *memory non-reclaimable* by the OS.
2. must be *manually configured* by the application.

Intermediate-sized on-demand faults could *increase tail latency* and lead to *memory bloat*.

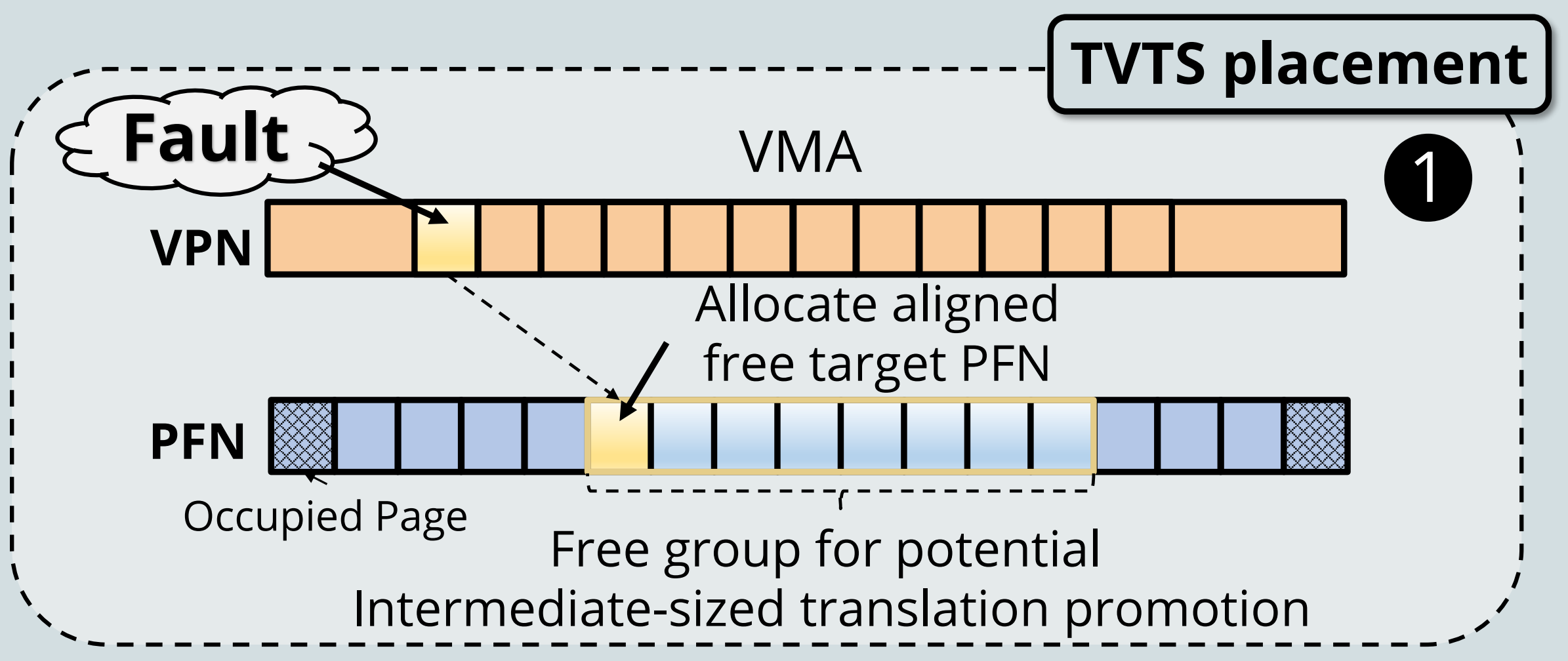
Transparent Variable Translation Sizes (TVTS)

Our Proposal

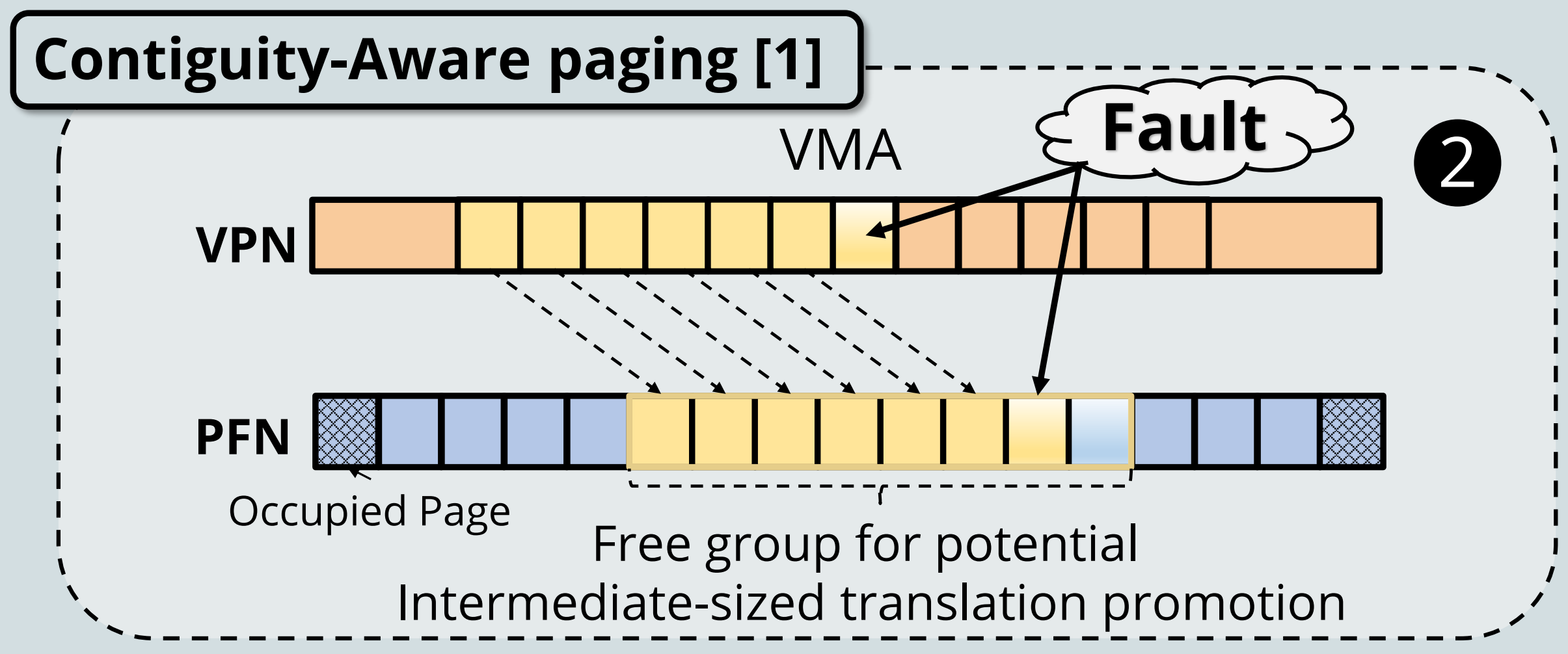
- Enhance CA-Paging [1] to create on demand suitably-aligned contiguous groups of pages.
- Transparently map them to the corresponding HW-supported intermediate translation sizes.

1. First fault in the VMA. TVTS *selects a intermediate-size aligned target PFN*.
2. CA-Paging [1] directs subsequent base page size faults to their corresponding aligned PFNs.
3. All base pages have been allocated. TVTS promotes the group to an intermediate-sized translation, by setting the contig bit in the base pages' descriptors.

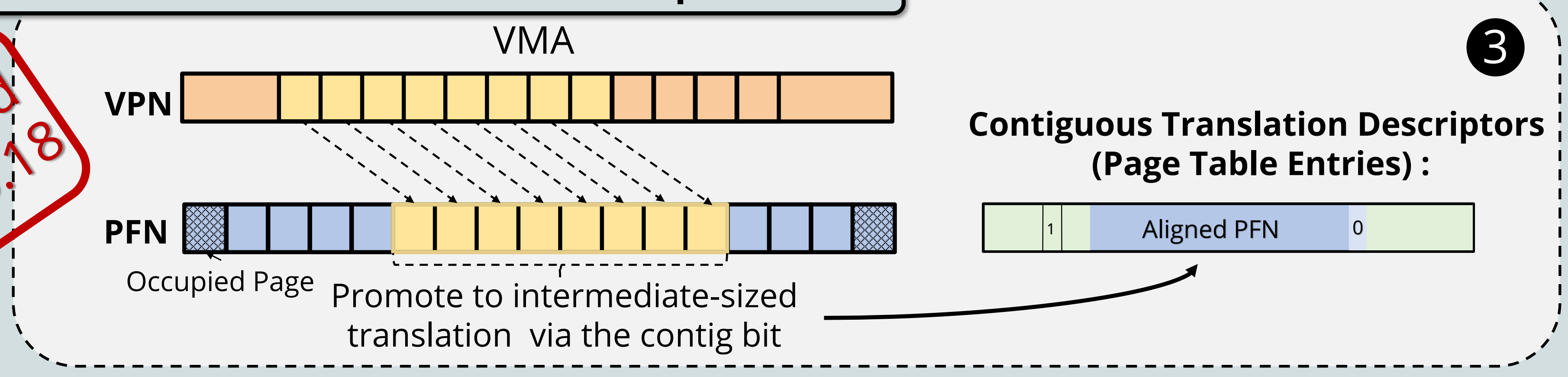
TVTS placement


Fault
VMA
1
VPN
Allocate aligned free target PFN
PFN
Occupied Page
Free group for potential Intermediate-sized translation promotion

Contiguity-Aware paging [1]


VMA
Fault
2
VPN
PFN
Occupied Page
Free group for potential Intermediate-sized translation promotion

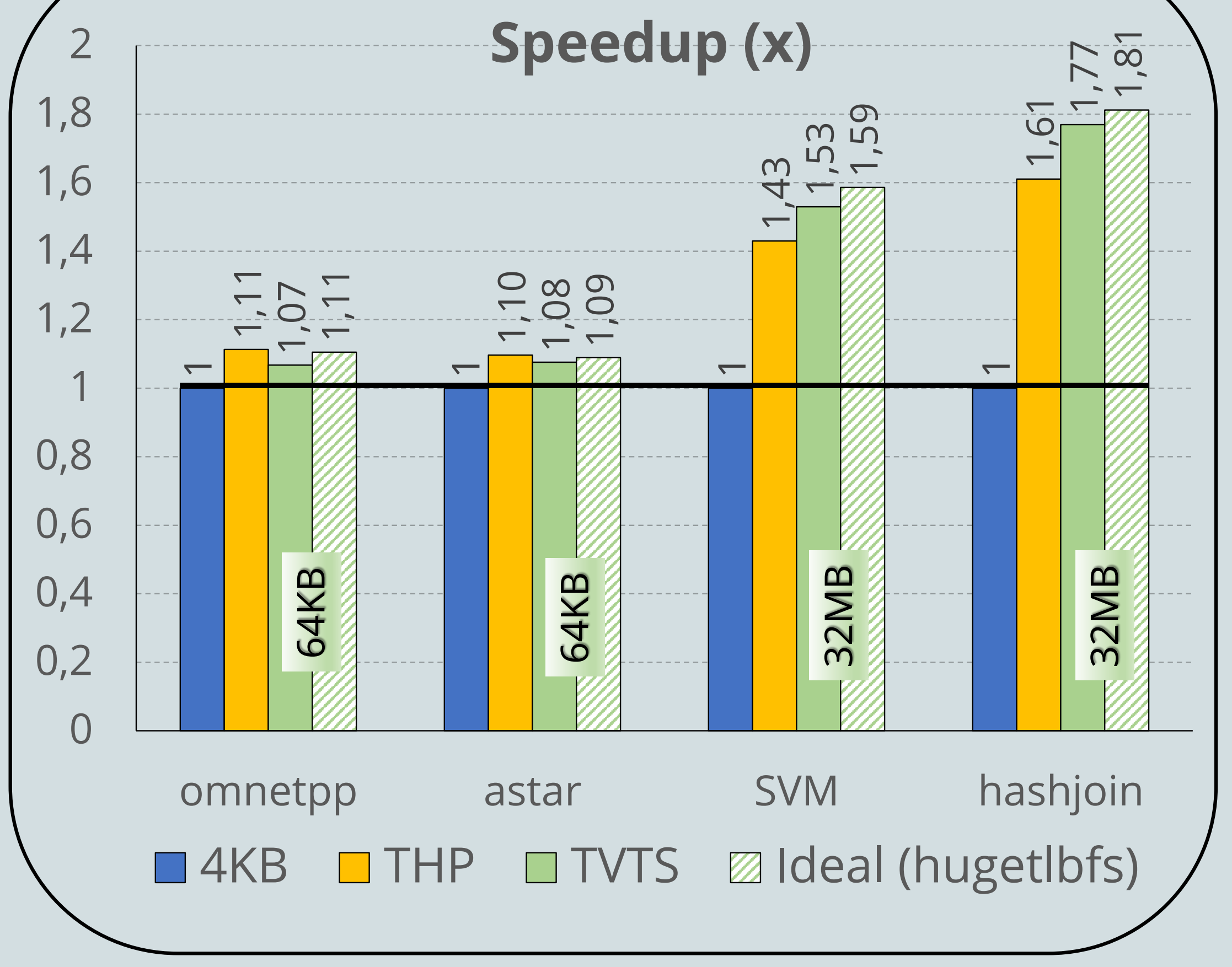
TVTS intermediate-sized translation promotion


VMA
3
VPN
PFN
Occupied Page
Promote to intermediate-sized translation via the contig bit
Contiguous Translation Descriptors (Page Table Entries) :
1
Aligned PFN
0

Implemented in Linux v5.18

Preliminary Results

Up to *16%* speedup on a real-world setup. Close to the ideal, without memory pre-allocation.



Work-in-progress

1. *KVM support* for contiguous translation descriptors.
2. Extend TVPS for *virtualized execution*.
3. Design an *online page-size selection* mechanism.
4. Study TVPS *performance under fragmentation*.
5.Add *support for the RISC-V Svnapot* extension.

[1] Chloe Alverti et al. "Enhancing and Exploiting Contiguity for Fast Memory Virtualization", ISCA'20