Perforated Page: Supporting Fragmented Memory Allocation for Large Pages

Chang Hyun Park, Sanghoon Cha, Bokyeong Kim,
Youngjin Kwon, David Black-Schaffer, and Jaehyuk Huh

UPPSALA
UNIVERSITET

KAIST

Benefits and Challenges of Large Pages

Performance

Difficult to make

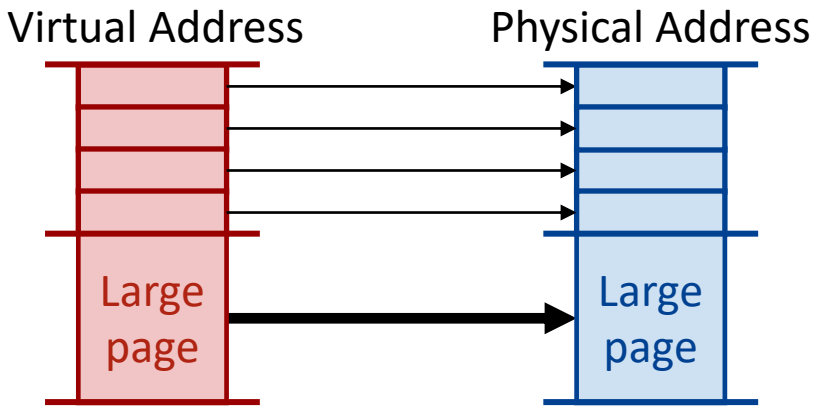
Overheads

UPPSALA
UNIVERSITET

KAIST

Large Pages for Performance

- 2MB large page → 512x TLB coverage
- 2MB large page → ~68% faster execution [1]



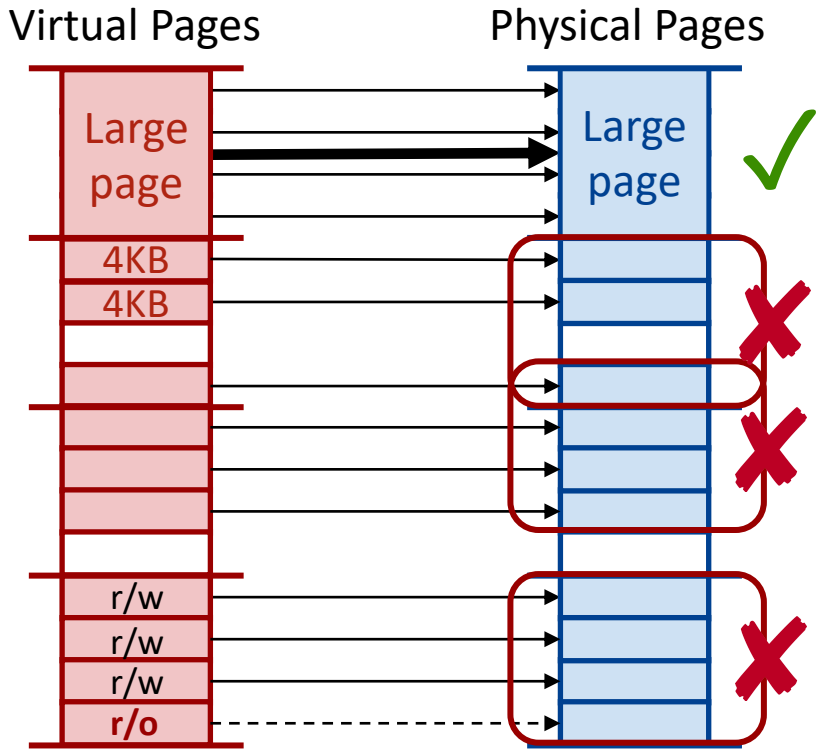
UPPSALA UNIVERSITET

[1] Margaritov et al., MICRO '19

KAIST

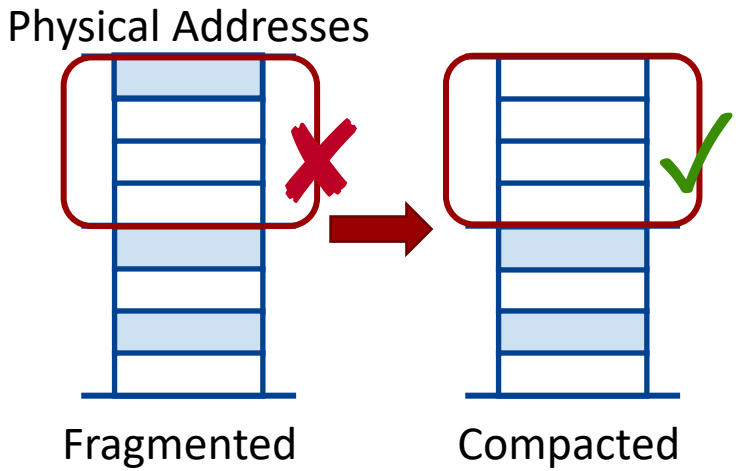
Large Pages are Difficult to Make

- Contiguous: 512 4KB pages
- Aligned: 2MB boundary
- Homogeneous: permissions

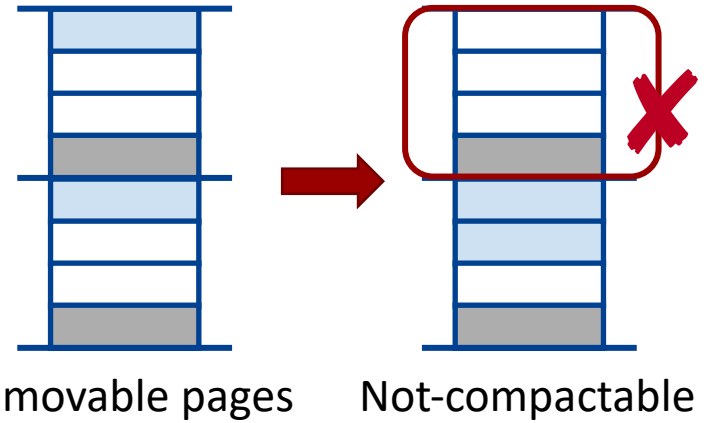


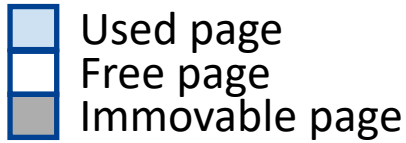
Overheads: Compaction

- Memory compaction to create large pages
  - Compaction takes up to 35% of time [1]



- Immovable pages prevent compaction
  - E.g., kernel allocations, I/O buffers, etc.
  - After Linux kernel build, 50% of 2MB regions had immovable pages





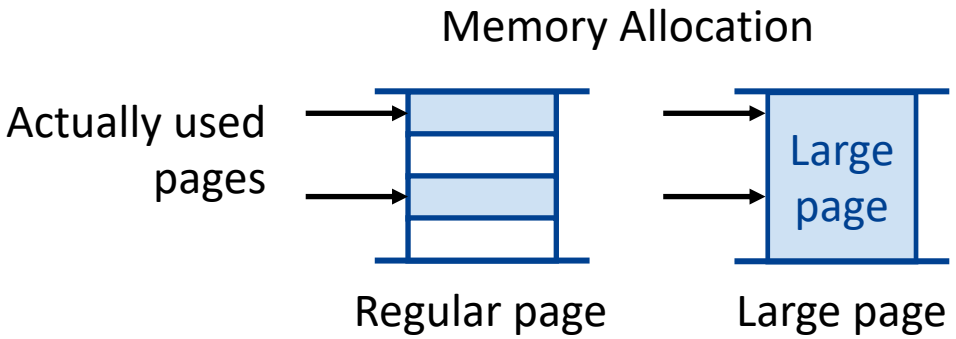
UPPSALA UNIVERSITET

[1] Parwar et al. ASPLOS '18

KAIST

Overheads: Memory Bloating

- Sparse access/use in large pages → wasted physical space



- Redis with large pages (4M keys, 16KB values)
  - **20% more** memory consumption (78GB → 93GB)
  - **45% fewer** TLB misses (1.8 MPKI → 1.1 MPKI)

UPPSALA UNIVERSITET

KAIST

Summary

Large pages for **Performance**

- Better TLB Coverage → Better Performance

Large pages are **Difficult** to make

- Require **contiguous** regions
- Require **homogeneous** permissions

Large pages come with **Overheads**

- **Costly compacting** to create contiguous regions
- **Immovable** pages common
- **Memory bloating** due to sparse access

UPPSALA UNIVERSITET

KAIST

Summary

Large pages for **Performance**

- Better TLB Coverage → Better Performance

Large p… **Difficult** to make

- Require **homogeneous** permissions

Can we get the **benefits** of large pages without the **difficulties** and **overheads**?

Large pages come with **Overheads**

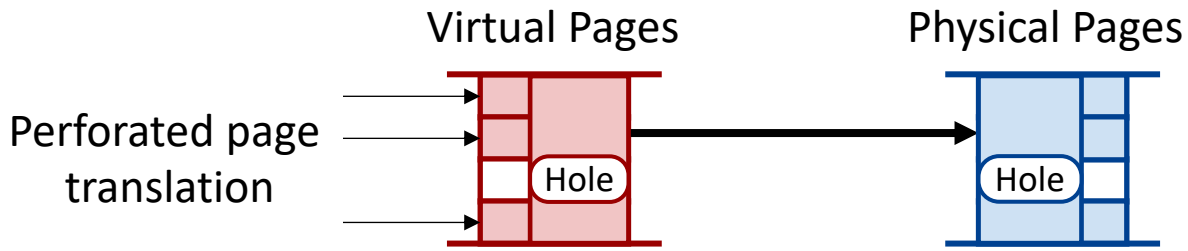
- **Costly compacting** to create contiguous regions
- **Immovable** pages common
- **Memory bloating** due to sparse access

UPPSALA UNIVERSITET

KAIST

Perforated Pages

- *Perforated pages* for majority of 2MB region
- *Hole* pages for flexible fine-grained mappings

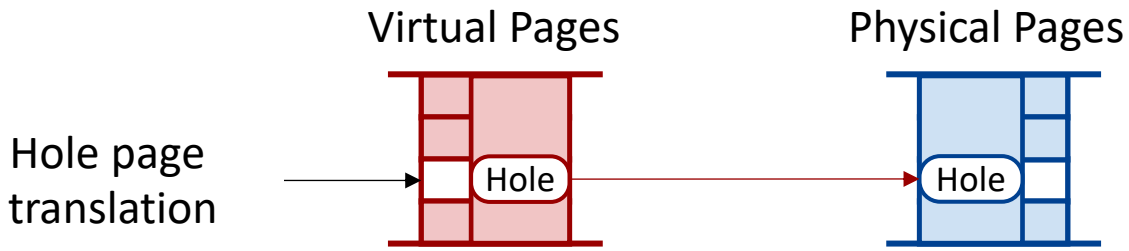


UPPSALA UNIVERSITET

KAIST

Perforated Pages

- *Perforated pages* for majority of 2MB region
- *Hole* pages for flexible fine-grained mappings

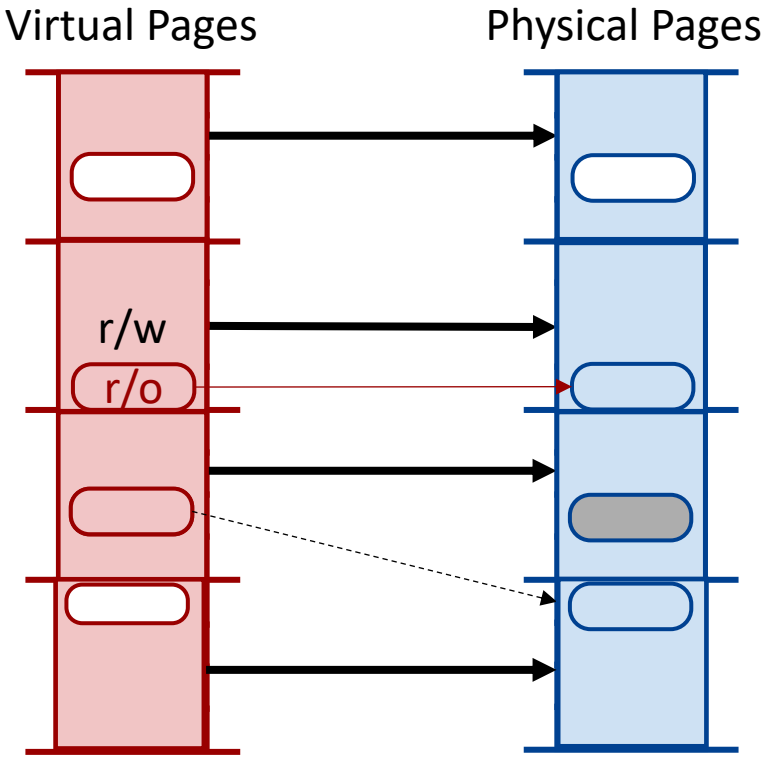


UPPSALA UNIVERSITET

KAIST

Perforated Pages

- **Large pages with holes**
  - Efficient translation *for most of the page*
  - Relaxed contiguity constraint *where needed*
  - Relaxed permission constraint *where needed*

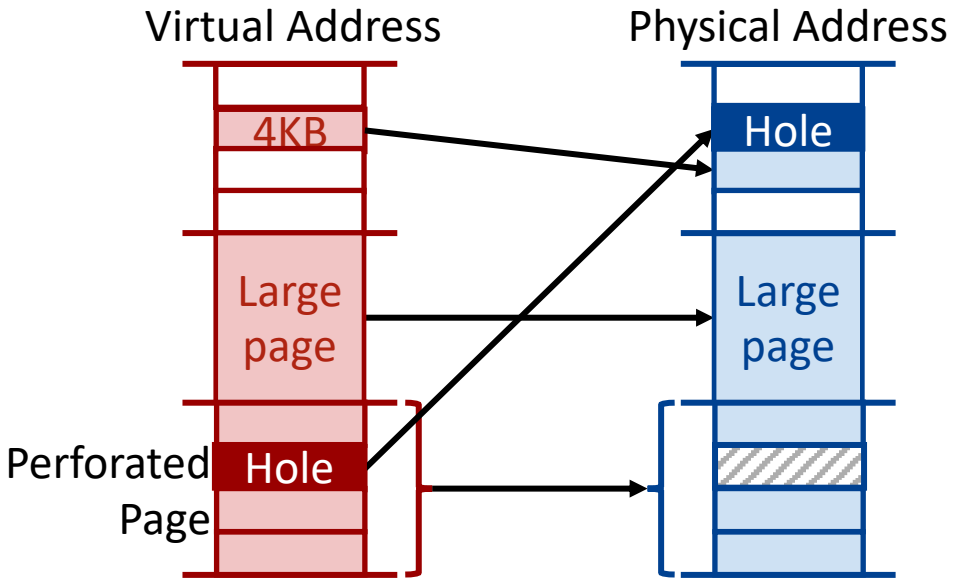


- Tolerate immovable pages
  - Overcome by re-mapping holes
  - 50% of 2MB + 50% of perforated pages
- Avoid bloating
  - Conserve untouched pages via holes
  - 0% bloating, 17% TLB MPKI reduction (2MB: 45% TLB MPKI reduction)

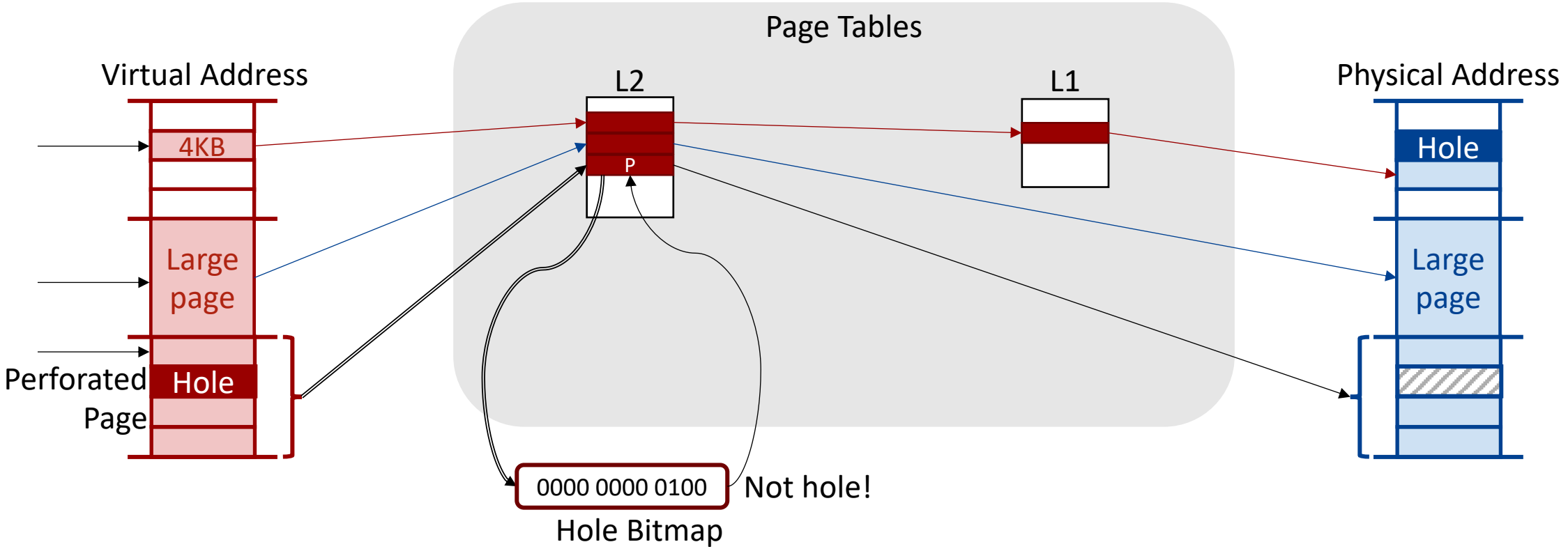
UPPSALA UNIVERSITET

KAIST

How it works


Virtual Address
Physical Address
4KB
Hole
Large page
Large page
Perforated Page
Hole

How it works: Basics



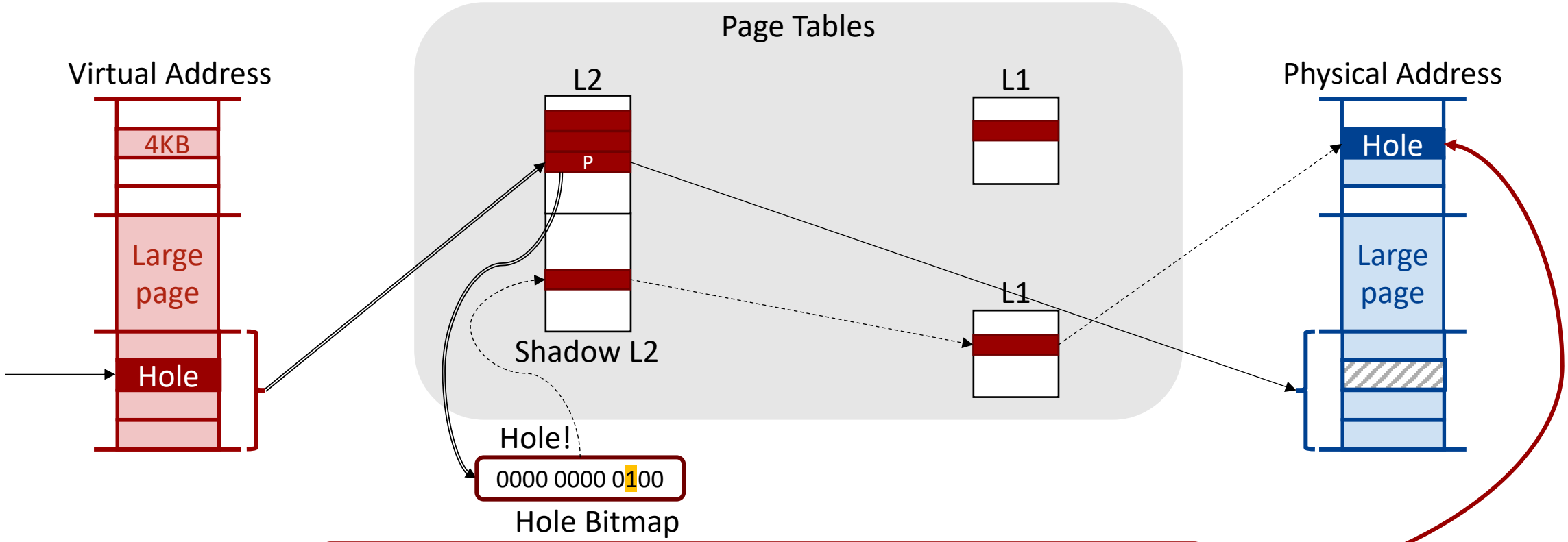
How to distinguish holes/perforated page?

Per-page bitmap to mark holes
0.003% capacity overhead

UPPSALA UNIVERSITET

KAIST

How it works: Basics



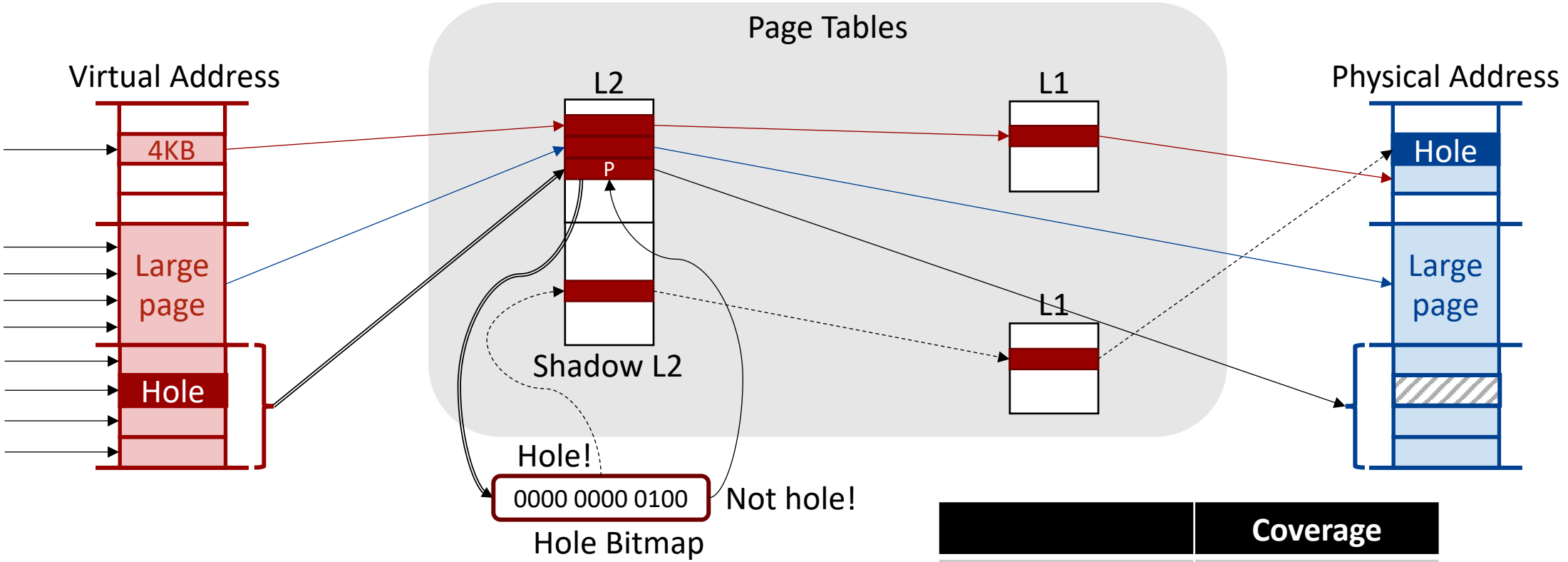
Where do we keep addresses for Hole pages?

Shadow L2 and Regular L1 Page Table!

UPPSALA UNIVERSITET

KAIST

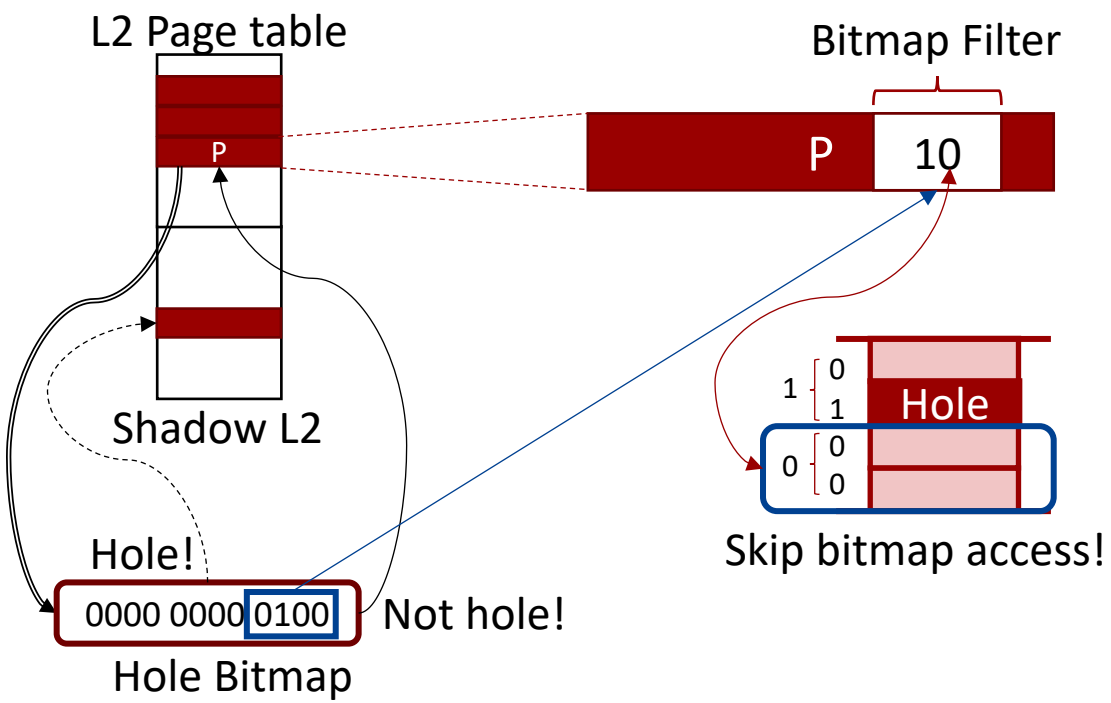
How it works: Summary



Additional accesses	Hole bitmap	Shadow L2
Perforated page walk	✓	

	Coverage
Large page	**512x**
Perforated page	**(512 - # holes)x**

How it works: Optimization



1. Coarse grain bitmap filter
   - Skip bitmap if definitely not hole
2. Hole bitmaps cached in TLB
   - 16 TLB entries per perforated page
   - Only insert accessed bitmap entries
3. Shadow L2 entries cached in Page table walker cache

UPPSALA UNIVERSITET

KAIST

Evaluation Methodology

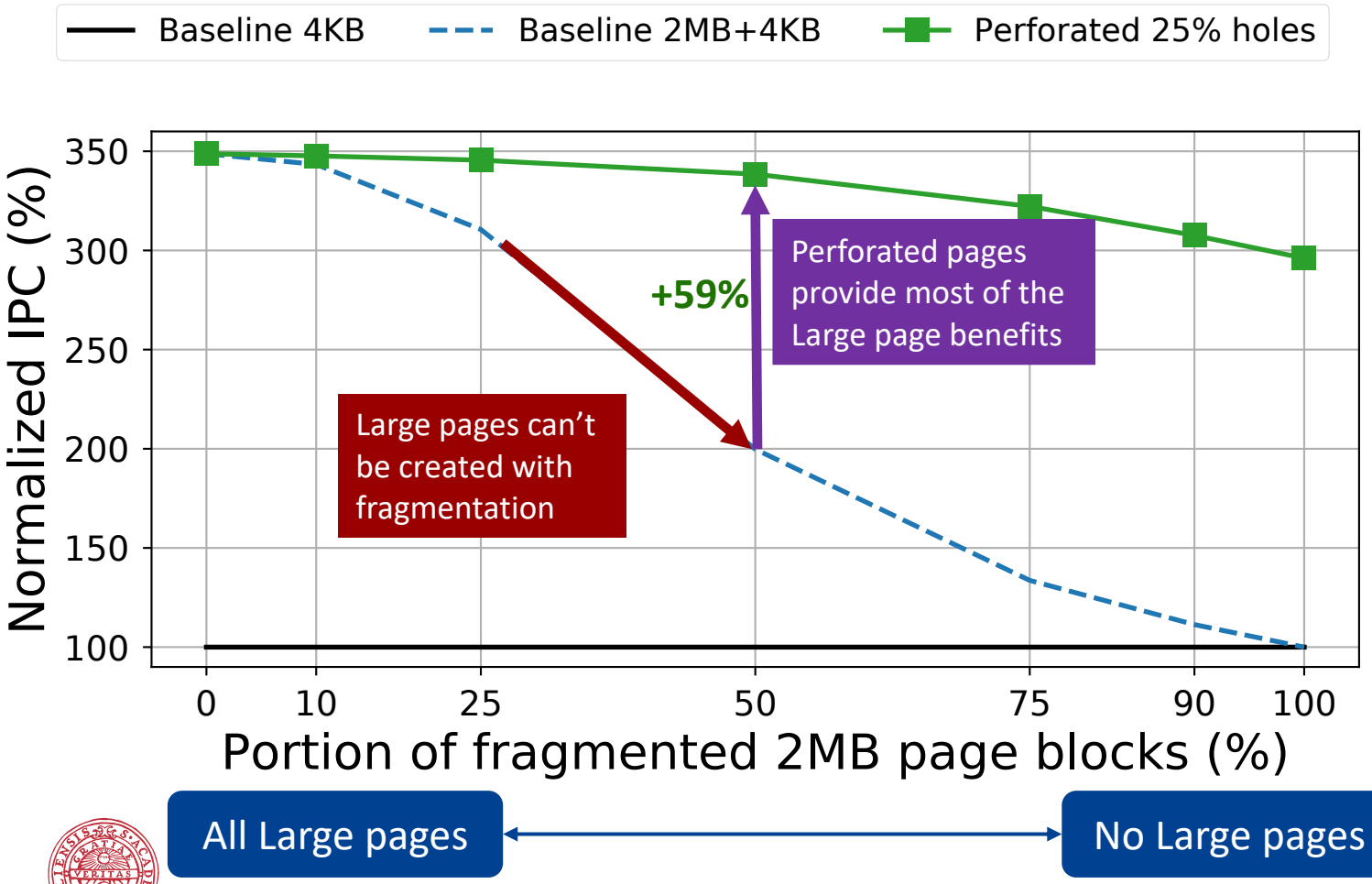
- Simulation configuration
  - Gem5
  - System-call emulation mode
- Microbenchmark
  - Random access (worst case)
- Real world benchmarks
  - SPECCPU
  - Biobench

Component	Configuration
Processor	2GHz, OoO x86
Caches	32KB L1 I/D 2MB L2
Memory	DDR4-2400, 4 channels

UPPSALA
UNIVERSITET

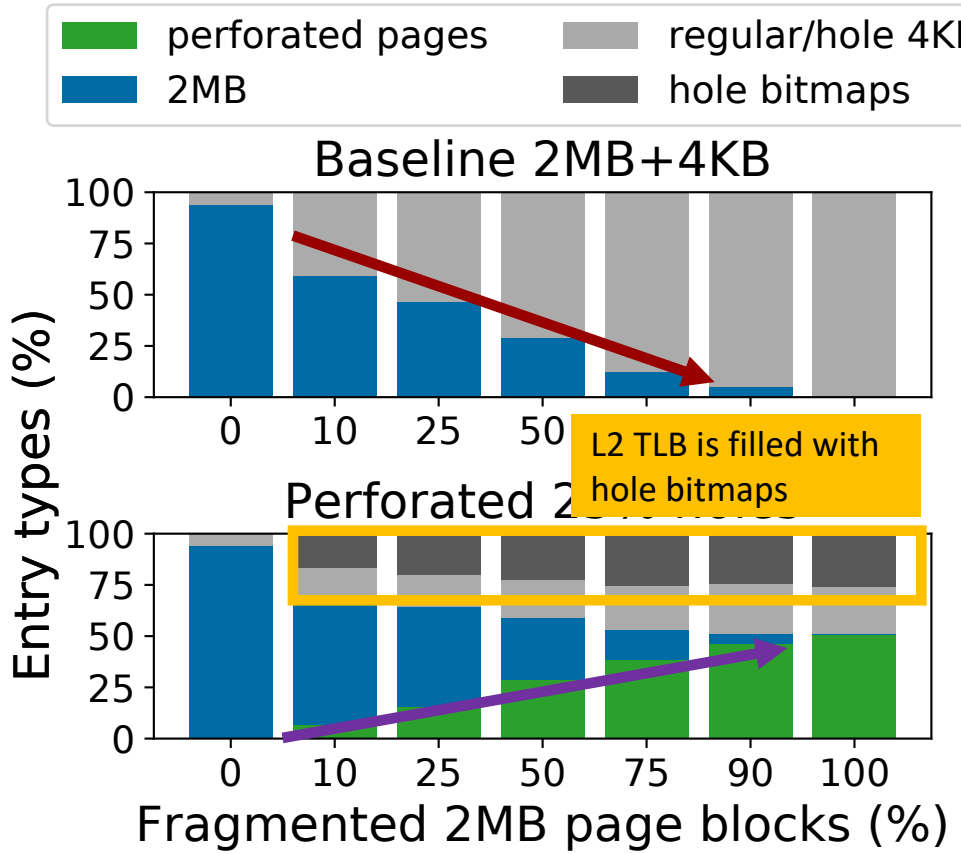
KAIST

Microbenchmark



All Large pages ⟷ No Large pages

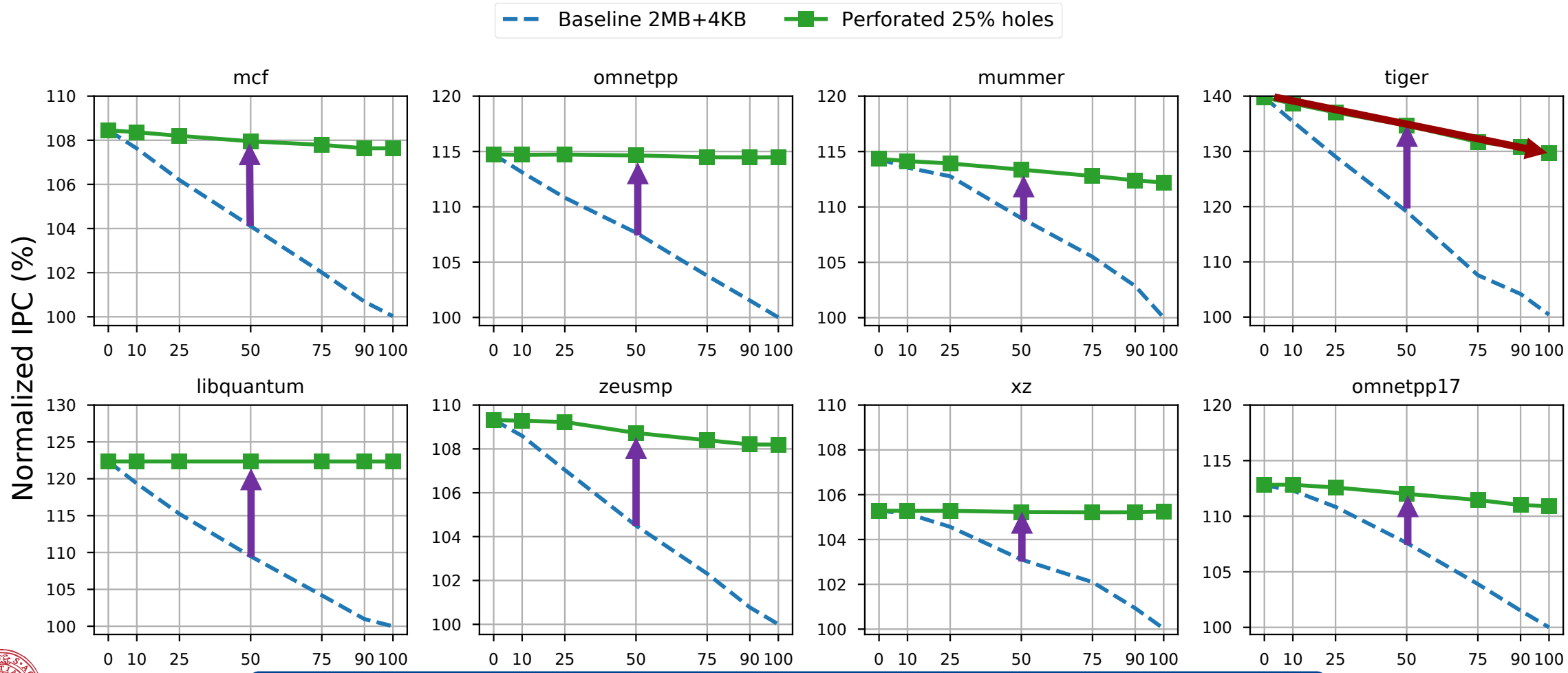
Breakdown of entries in L2 TLB



UPPSALA UNIVERSITET

KAIST

Benchmarks



2% - 11% performance improvement over Large page TLB

93% - 99% performance of ideal 2MB mapping (x=0)

UPPSALA UNIVERSITET

KAIST

More in the paper

- Details of TLB
- OS issues:
  - Advise for OS
  - TLB Shootdowns
- Virtualization Support
- More evaluation:
  - More hole % scenarios
  - Dispersed holes scenario
  - Virtualization
  - Comparison to prior work
- And more…

UPPSALA
UNIVERSITET

KAIST

Conclusion

- Large pages deliver performance, but has challenges:
  - Contiguous, homogenous
  - Compaction, bloating
  - Immovable pages

- Perforated page provides flexible large page
  - Large-page translations for most of the data
  - Holes to handle pages that differ
- Minimal changes to existing translation HW and data structure
- Performance similar to ideal large page mappings
  - Retains 93-99% of performance

UPPSALA
UNIVERSITET

KAIST

Perforated Page: Supporting Fragmented Memory Allocation for Large Pages

Chang Hyun Park, Sanghoon Cha, Bokyeong Kim,
Youngjin Kwon, David Black-Schaffer, and Jaehyuk Huh

UPPSALA
UNIVERSITET

KAIST