HawkEye: Efficient Fine-grained OS Support for Huge Pages

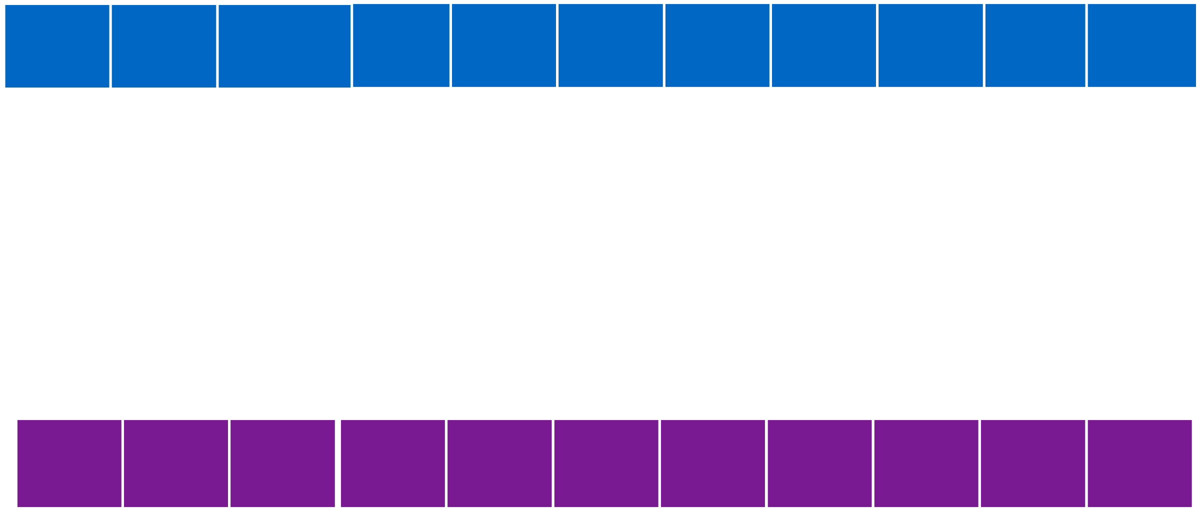
Ashish Panwar[1], Sorav Bansal[2], K. Gopinath[1]

Indian Institute of Science (IISc), Bangalore[1]

Indian Institute of Technology, Delhi [2]

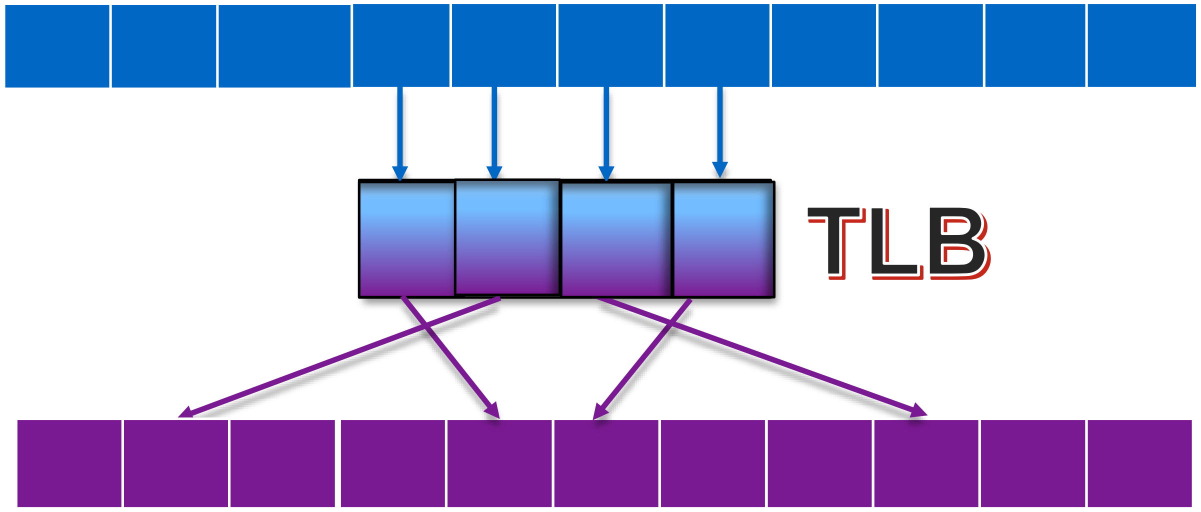
Virtual address space

Virtual address space

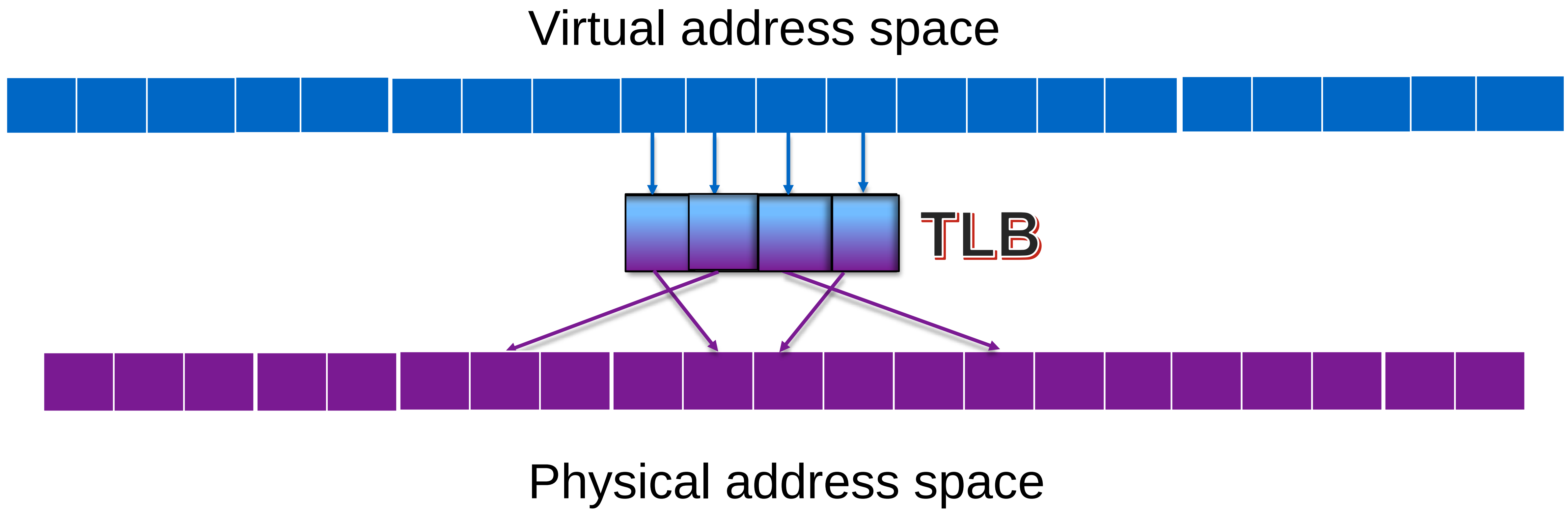


Physical address space

Virtual address space

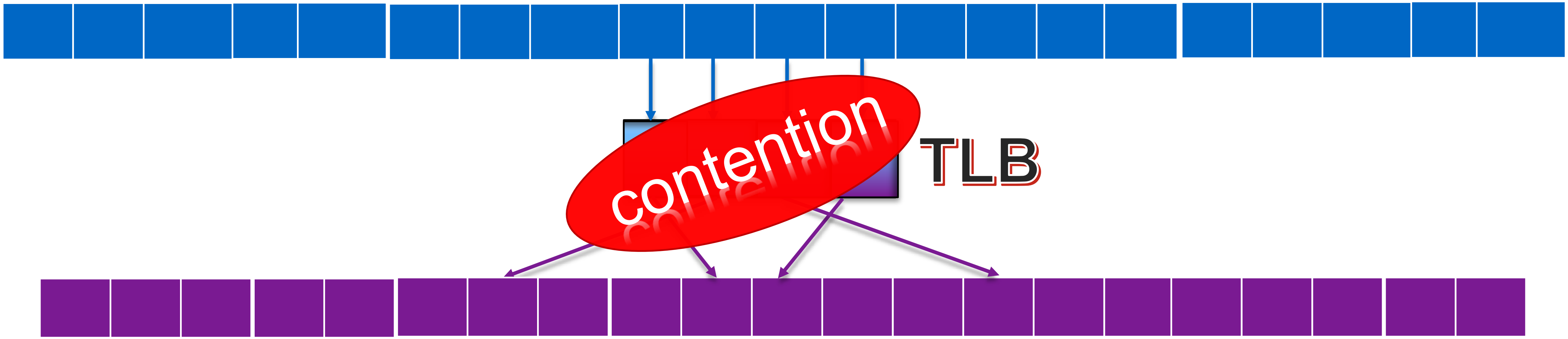


Physical address space

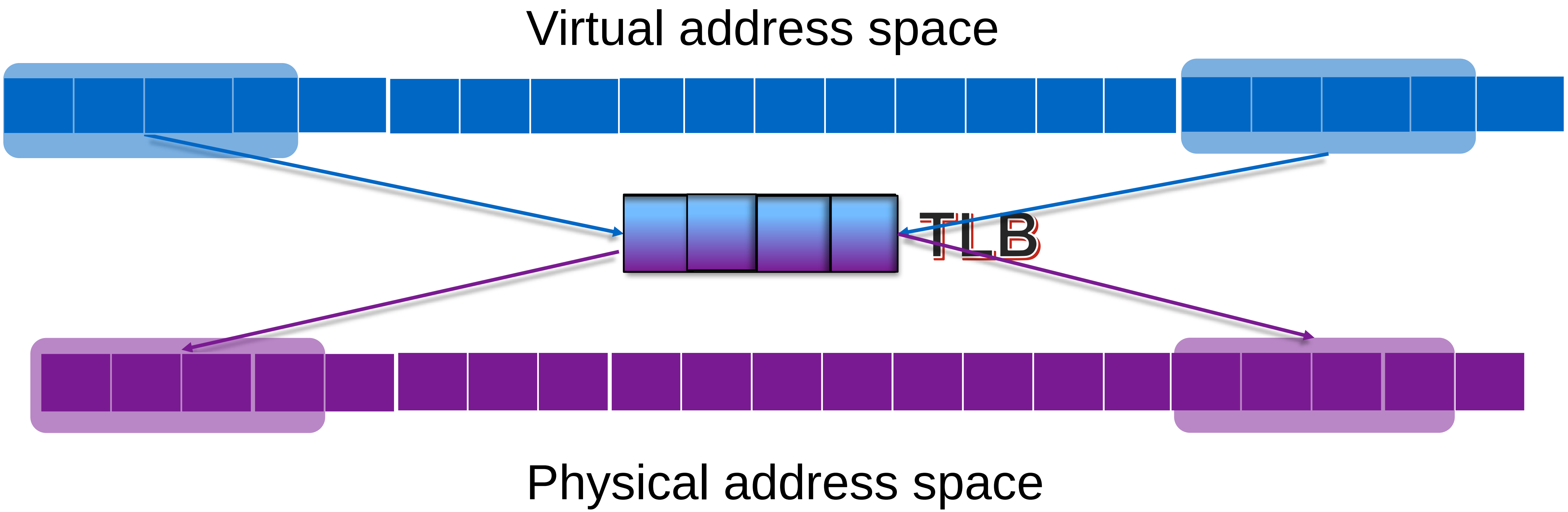

Virtual address space
TLB
Physical address space

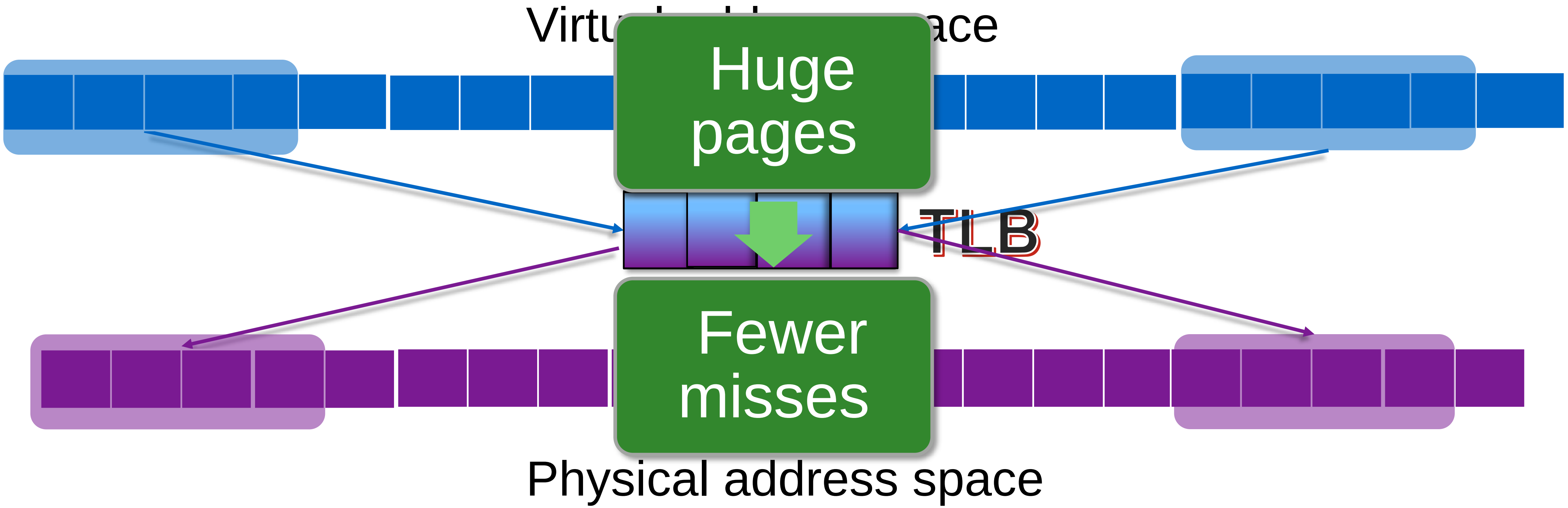
Too much TLB pressure!

Virtual address space



Physical address space


Virtual address space
TLB
Physical address space


Virtu
ace
Huge pages
TLB
Fewer misses
Physical address space

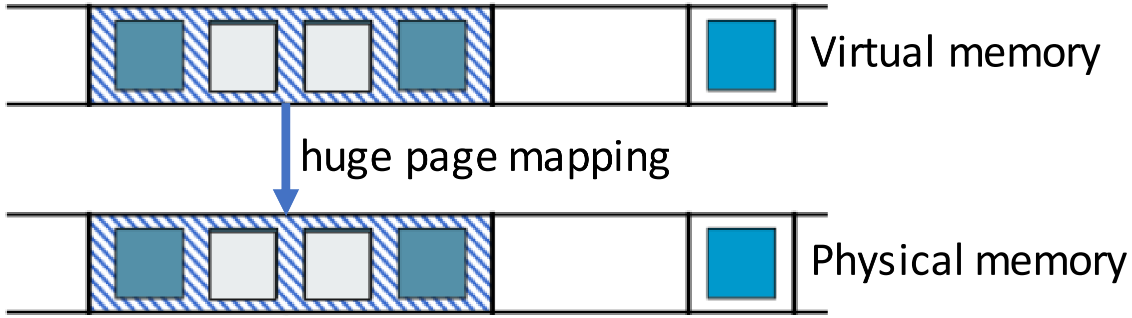
OS Challenges

- ❑ Complex trade-offs
  - Memory bloat vs. performance
  - Page fault latency vs. the number of page faults
- ❑ Challenges due to (external) fragmentation
  - How to leverage limited memory contiguity
  - Fairness in huge page allocation

Memory bloat
vs.
performance

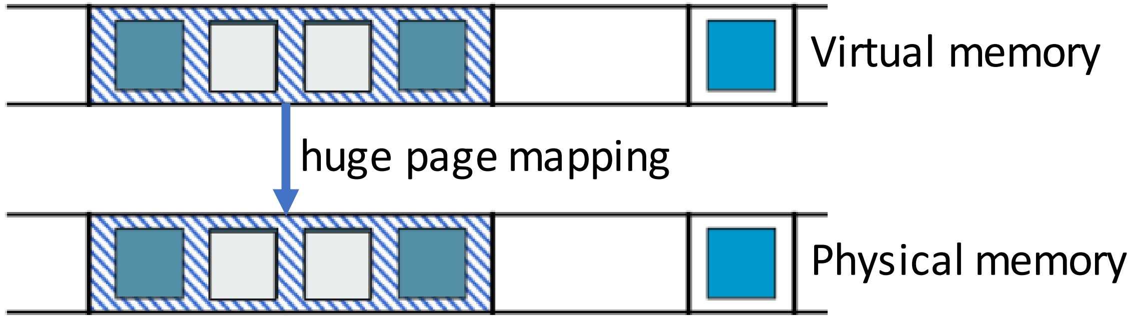
Internal fragmentation

aggressive allocation

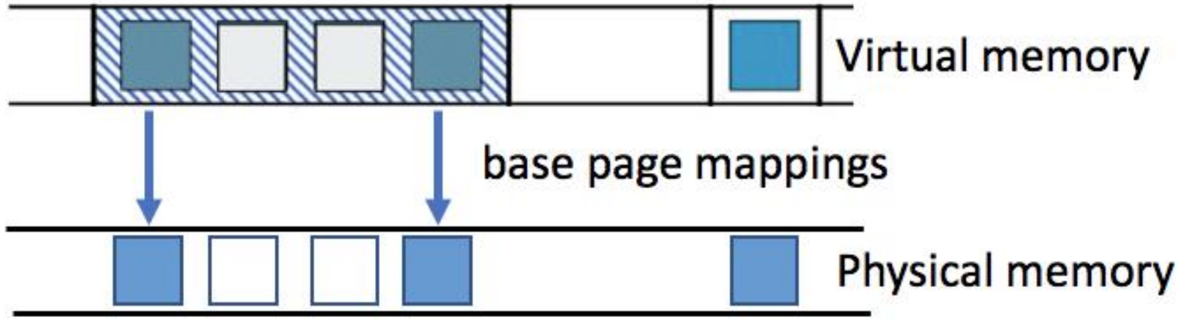


Internal fragmentation

aggressive allocation

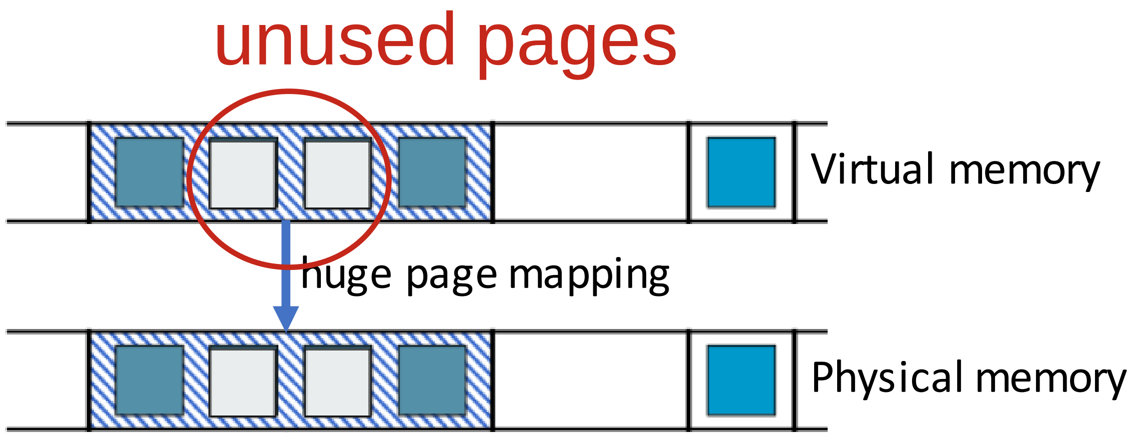


conservative allocation

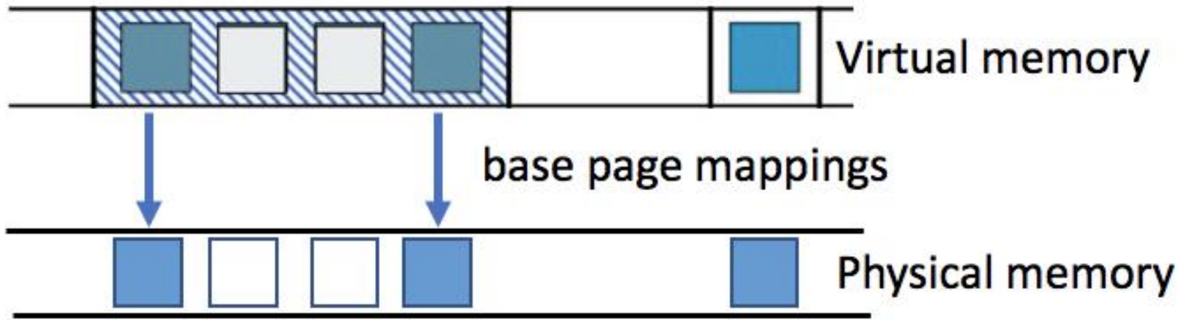


Internal fragmentation

aggressive allocation

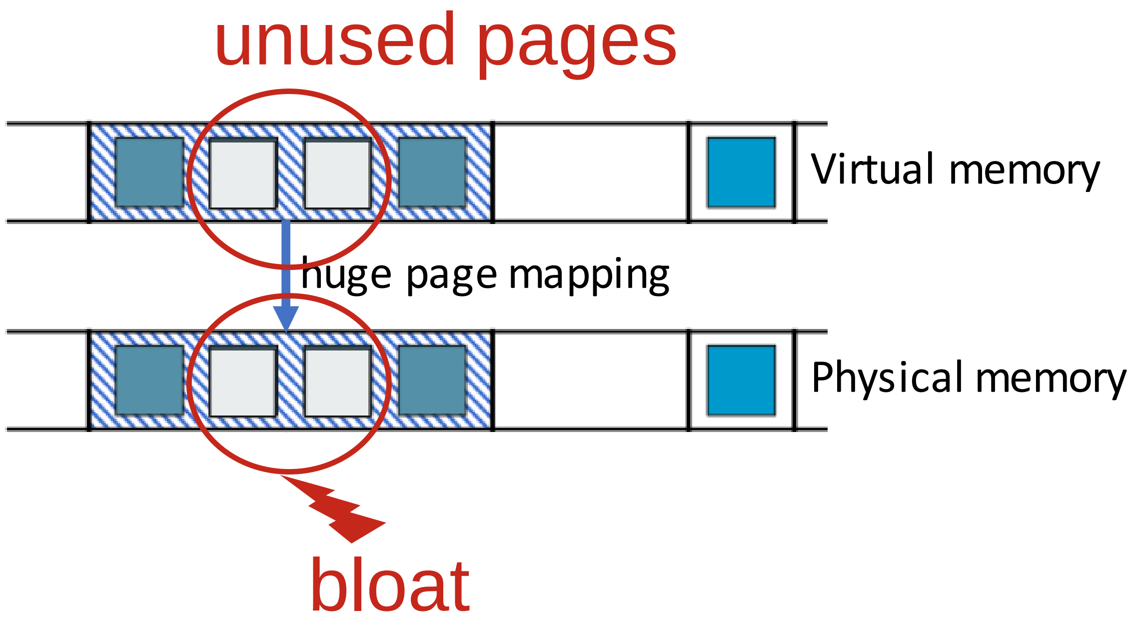


conservative allocation

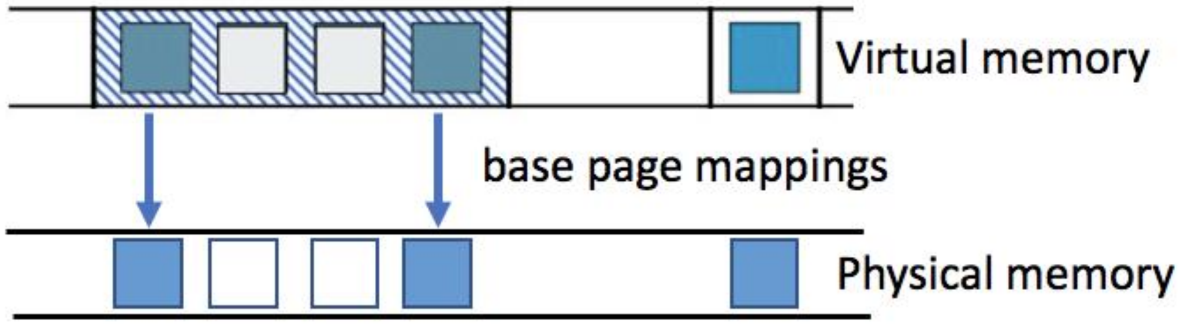


Internal fragmentation

aggressive allocation



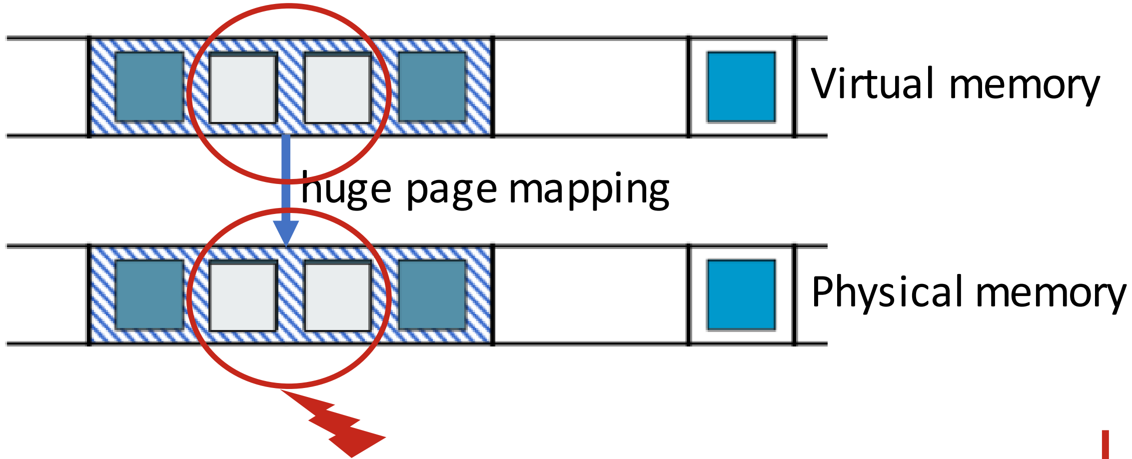
conservative allocation



Internal fragmentation

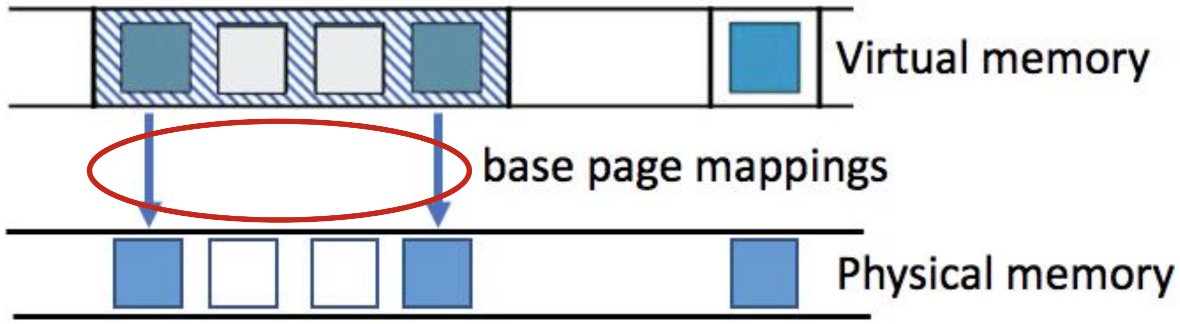
aggressive allocation

unused pages



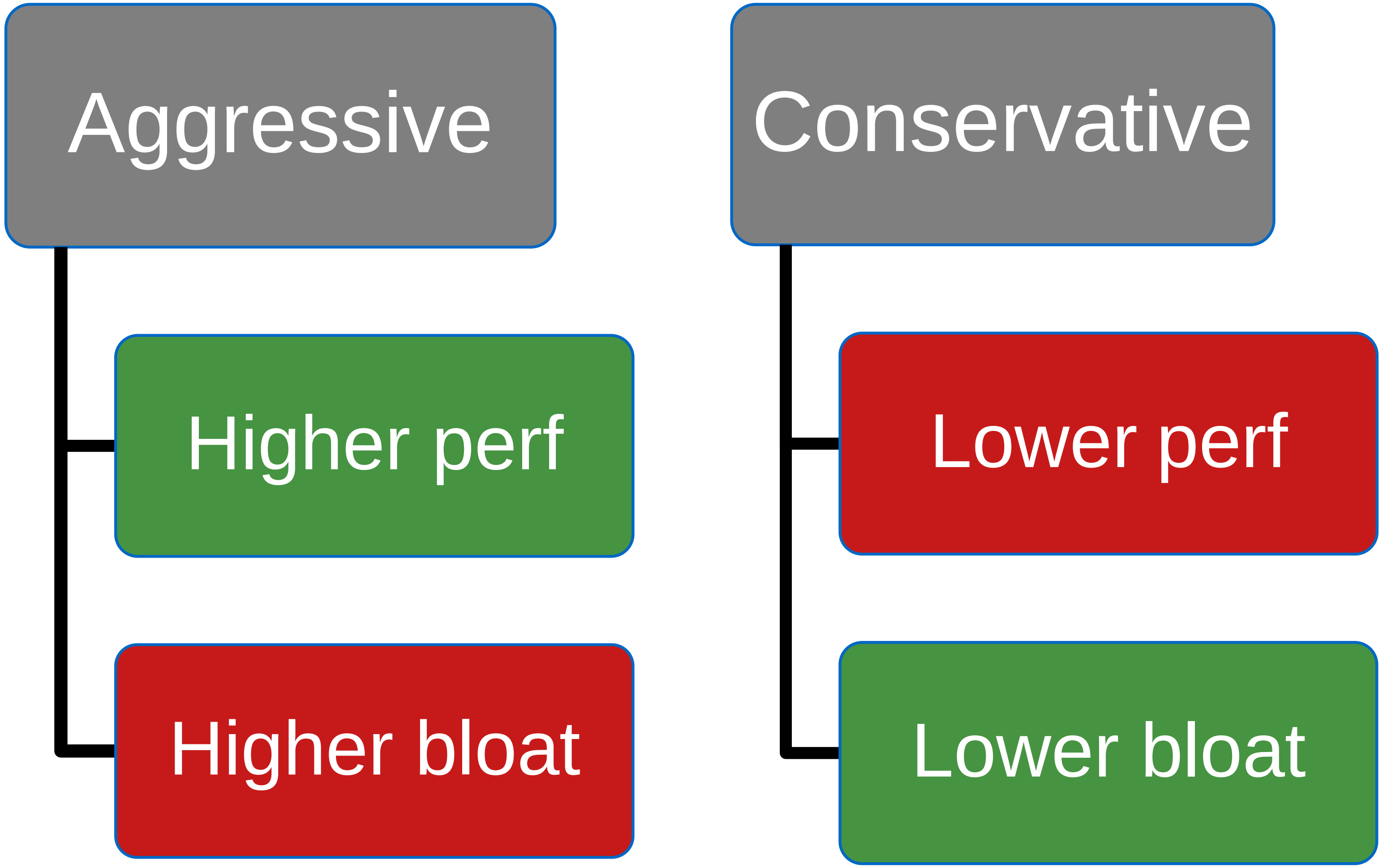
bloat

conservative allocation



Lower TLB reach (impacts performance)

Bloat vs. performance



Latency
vs.
page faults

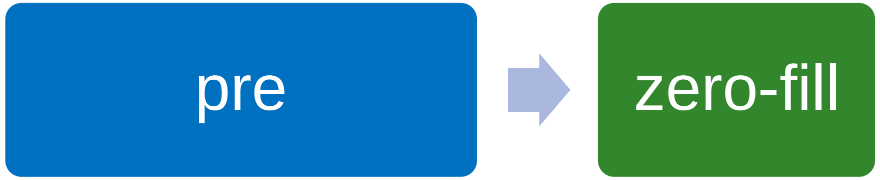
- Find a page

4-KB

pre

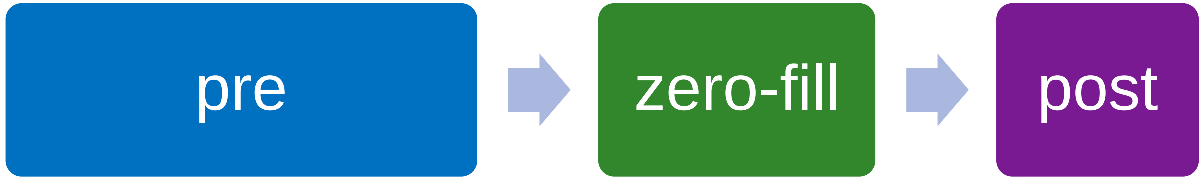
- Find a page, zero-fill

4-KB



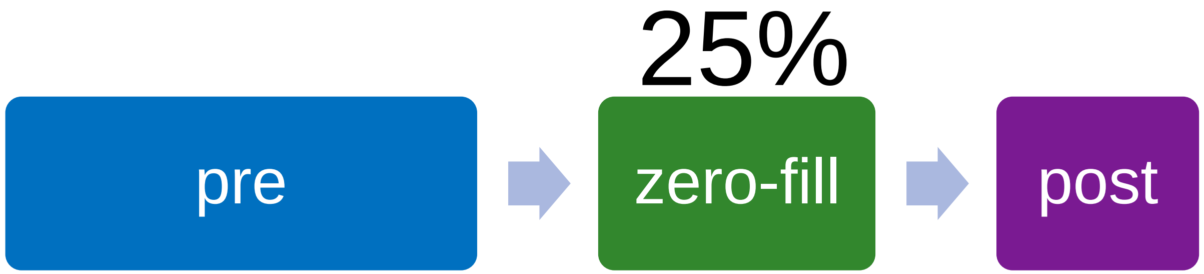
- Find a page, zero-fill, map

4-KB

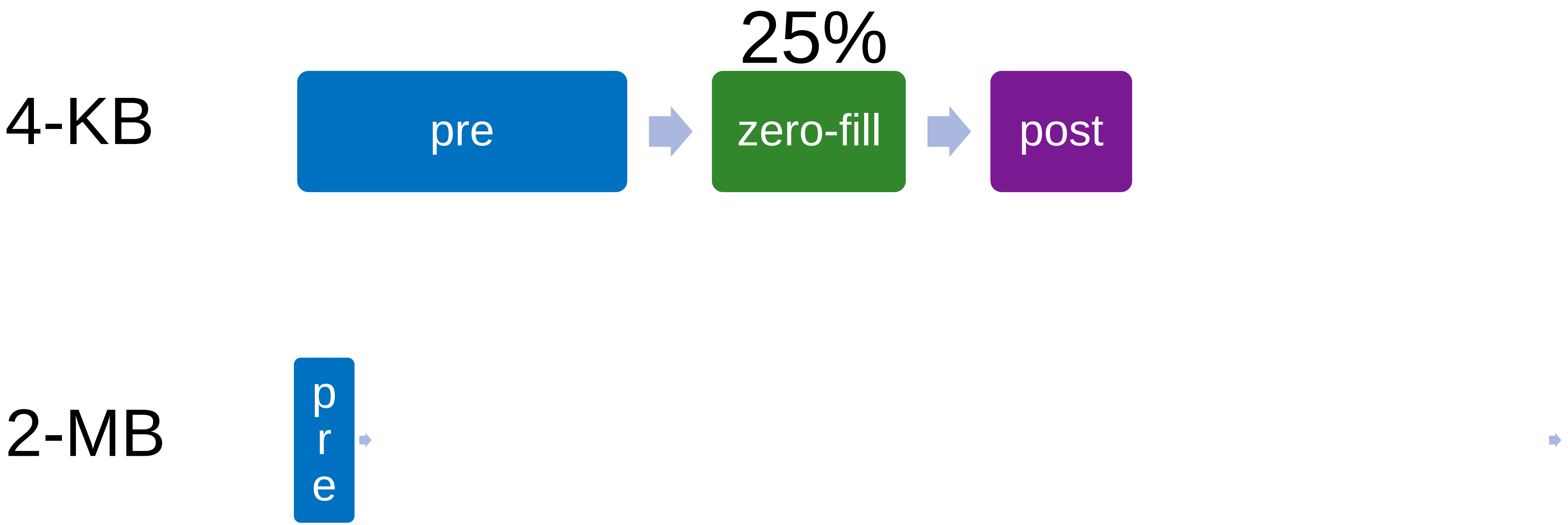


- Find a page, zero-fill, map

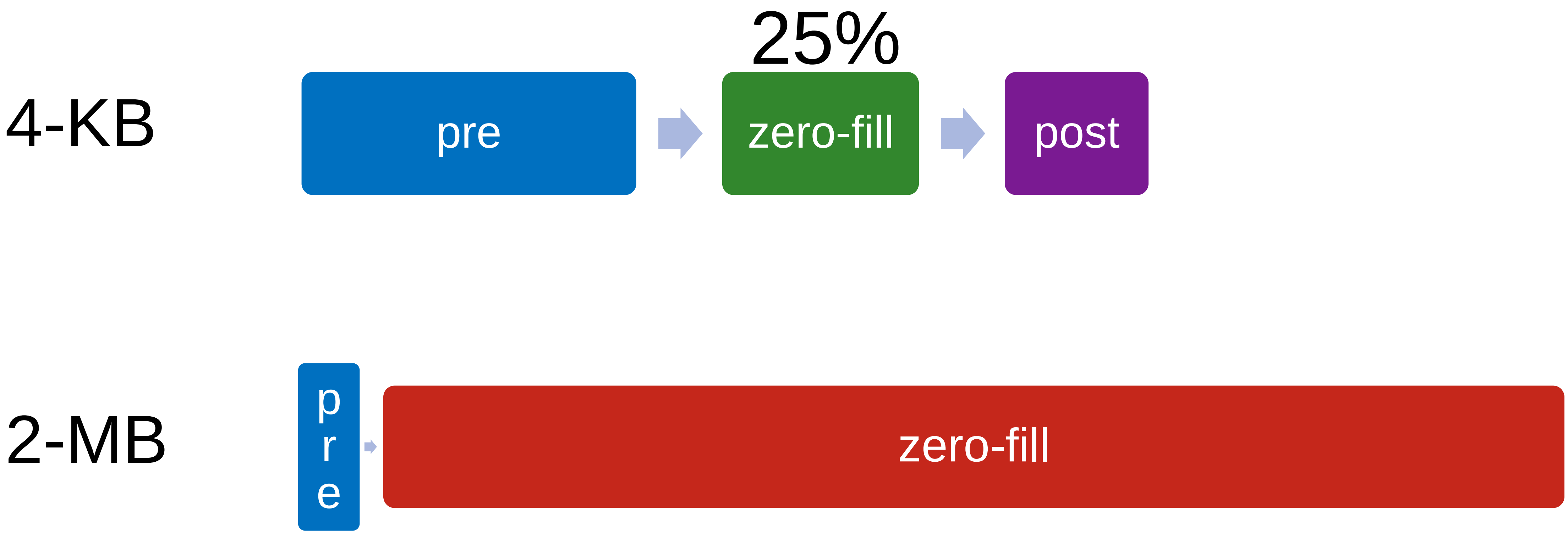
4-KB



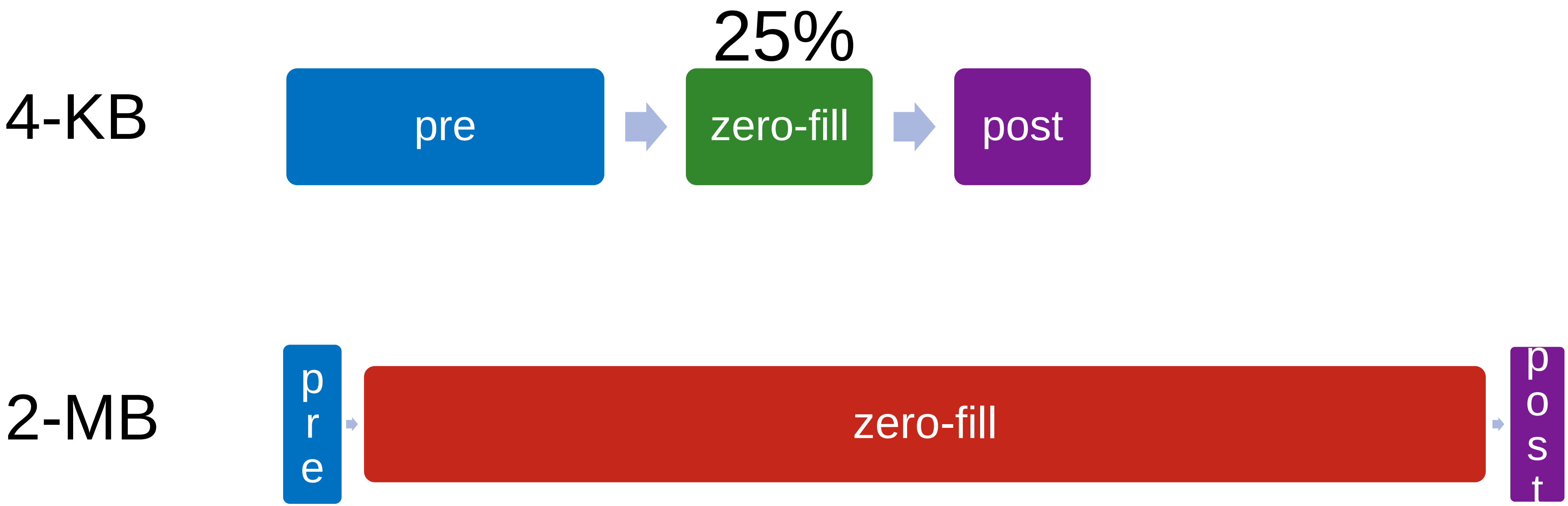
- Find a page, zero-fill, map



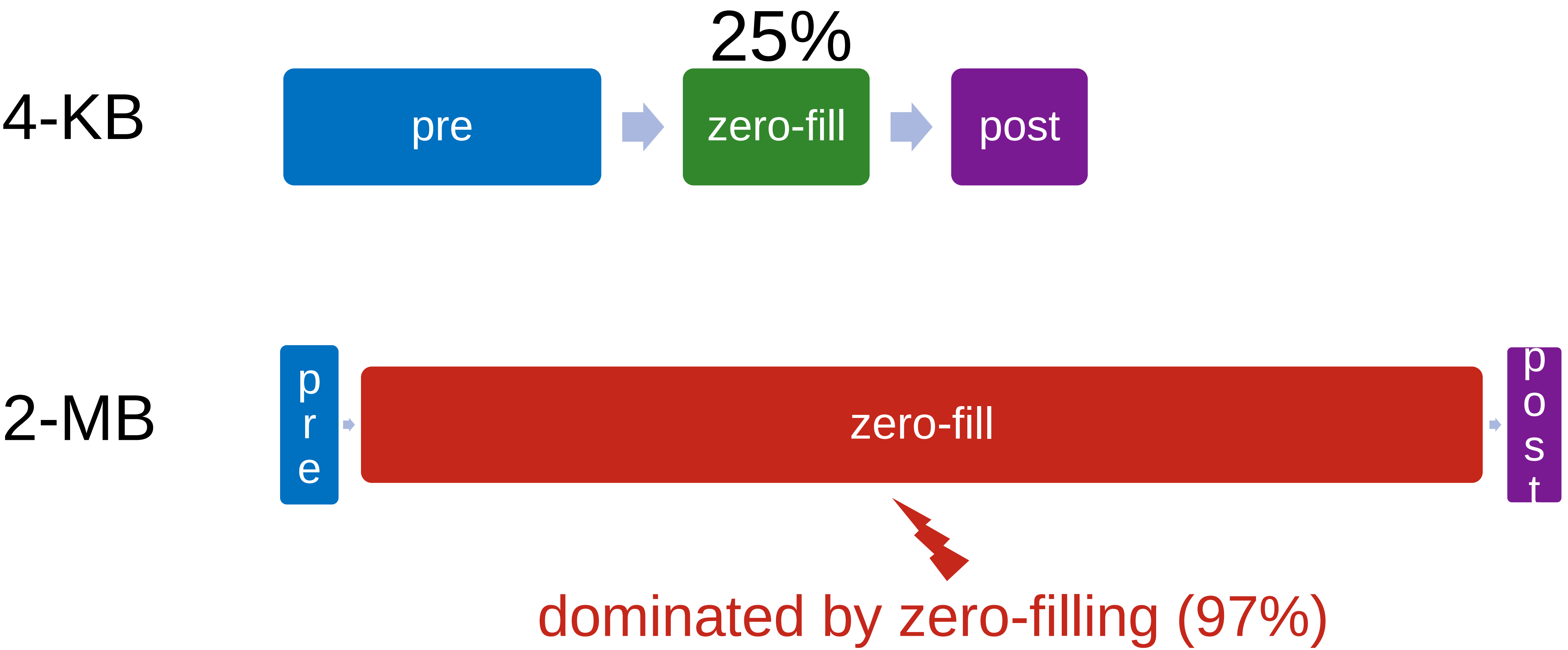
- Find a page, zero-fill, map



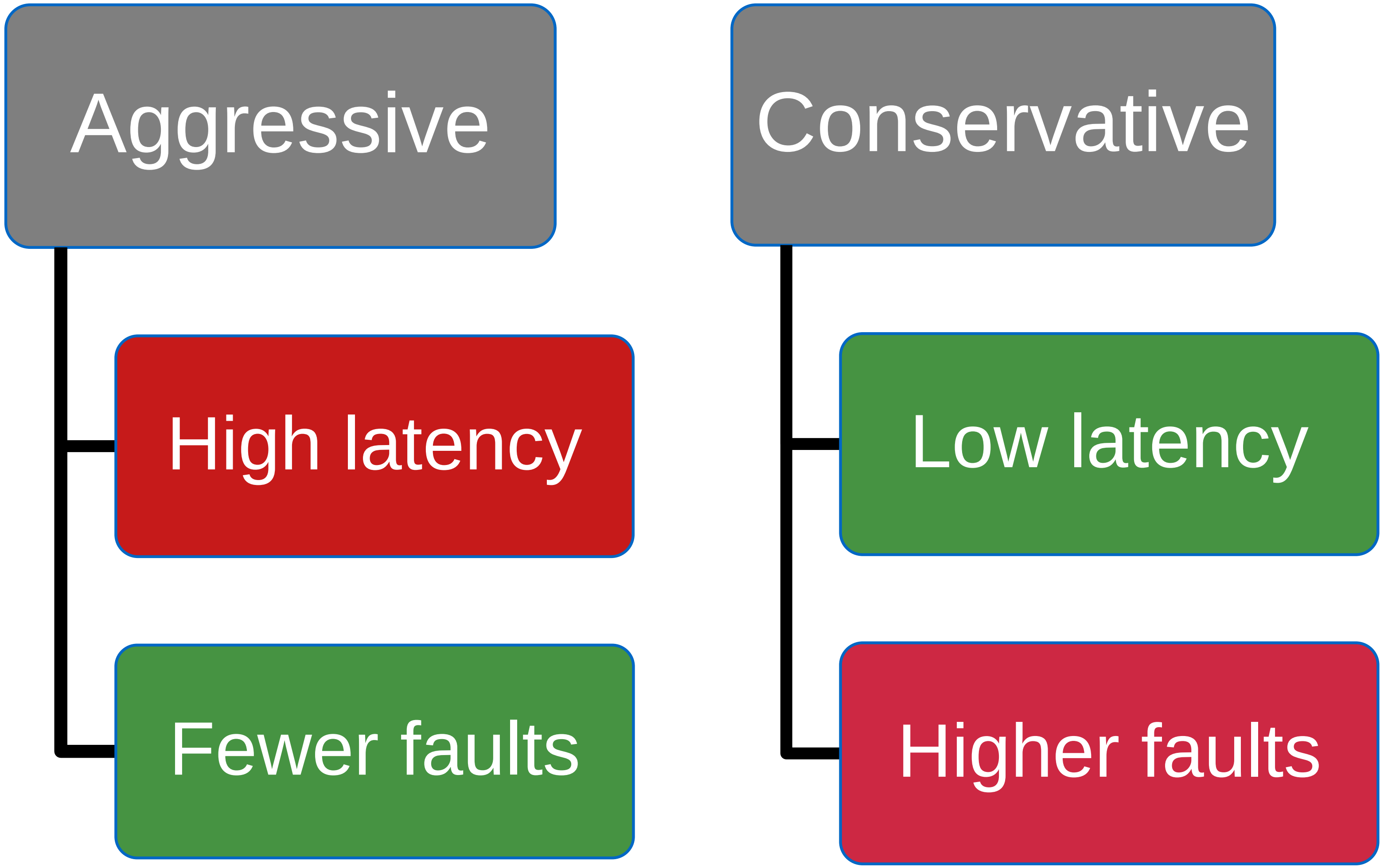
- Find a page, zero-fill, map



- Find a page, zero-fill, map



Latency vs. # page faults



Current systems favor opposite ends of the design spectrum

- FreeBSD is conservative (compromise on performance)
- Linux is throughput-oriented (compromise on latency and bloat)

conservative vs. aggressive

		FreeBSD	Linux
Tradeoff-1:	Memory bloat	Low	High
	Performance	Low	High
Tradeoff-2:	Allocation latency	Low	High
	# page faults	High	Low

Ingens (OSDI’16)

- **Asynchronous** allocation
  - Huge pages allocated in the background
- **Utilization-threshold** based allocation
  - Tunable bloat vs. performance
  - Adaptive based on memory pressure
- Fairness driven by **per-process fairness metric**
  - Heuristic based on past behavior

Ingens (OSDI'16)

- **Asynchronous** allocation
  - Huge pages allocated in the background
- **Utilization-threshold** based allocation
  - Tunable bloat vs. performance
  - Adaptive based on memory pressure
- Fairness driven by **per-process fairness metric**
  - Heuristic based on past behavior

low latency

too many page faults

Ingens (OSDI'16)

- **Asynchronous** allocation
  - Huge pages allocated in the background
- **Utilization-threshold** based allocation
  - Tunable bloat vs. performance
  - Adaptive based on memory pressure
- Fairness driven by **per-process fairness metric**
  - Heuristic based on past behavior

low latency

too many page faults

manual

Ingens (OSDI'16)

- **Asynchronous** allocation
  - Huge pages allocated in the background
- **Utilization-threshold** based allocation
  - Tunable bloat vs. performance
  - Adaptive based on memory pressure
- Fairness driven by **per-process fairness metric**
  - Heuristic based on past behavior

low latency

too many page faults

manual

weak correlation with
page walk overhead

Current state-of-the-art

		FreeBSD	Linux	Ingens
Tradeoff-1:	Memory bloat	Low	High	Tunable
	Performance	Low	High	Tunable
Tradeoff-2:	Allocation latency	Low	High	Low
	# page faults	High	Low	High

- Hard to find the sweet-spot for utilization-threshold in Ingens
  - Application dependent, phase dependent

HawkEye

Key Optimizations

- Asynchronous page pre-zeroing[1]
- Content deduplication based bloat mitigation
- Fine-grained intra-process allocation
- Fairness driven by hardware performance counters

[1] Optimizing the Idle Task and Other MMU Tricks, OSDI'99

Asynchronous page pre-zeroing

- Pages zero-filled in the background
- Potential issues:
  - Cache pollution – leverage non-temporal writes
  - DRAM bandwidth consumption – rate-limited
    - Limit CPU utilization (e.g., 5%)

Asynchronous page pre-zeroing

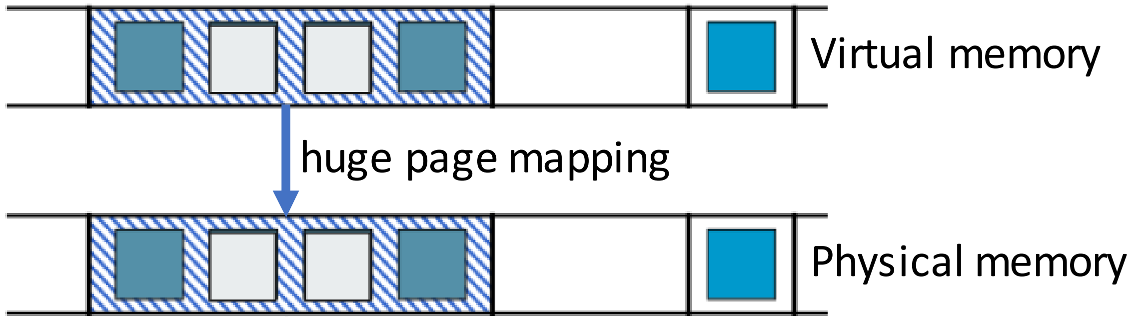
Enables aggressive allocation with low latency

✓ 13.8x faster VM spin-up

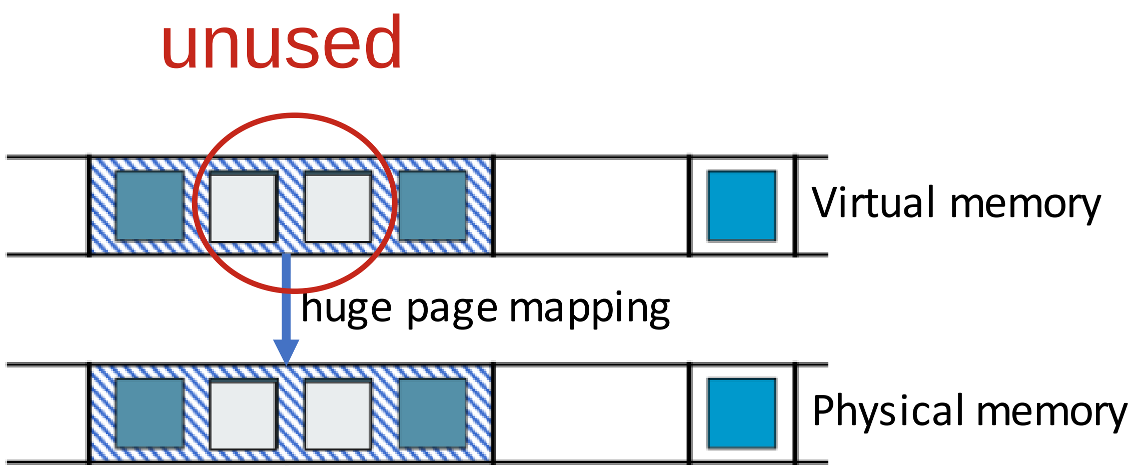
✓ 1.26x higher throughput (Redis)

Mitigating bloat

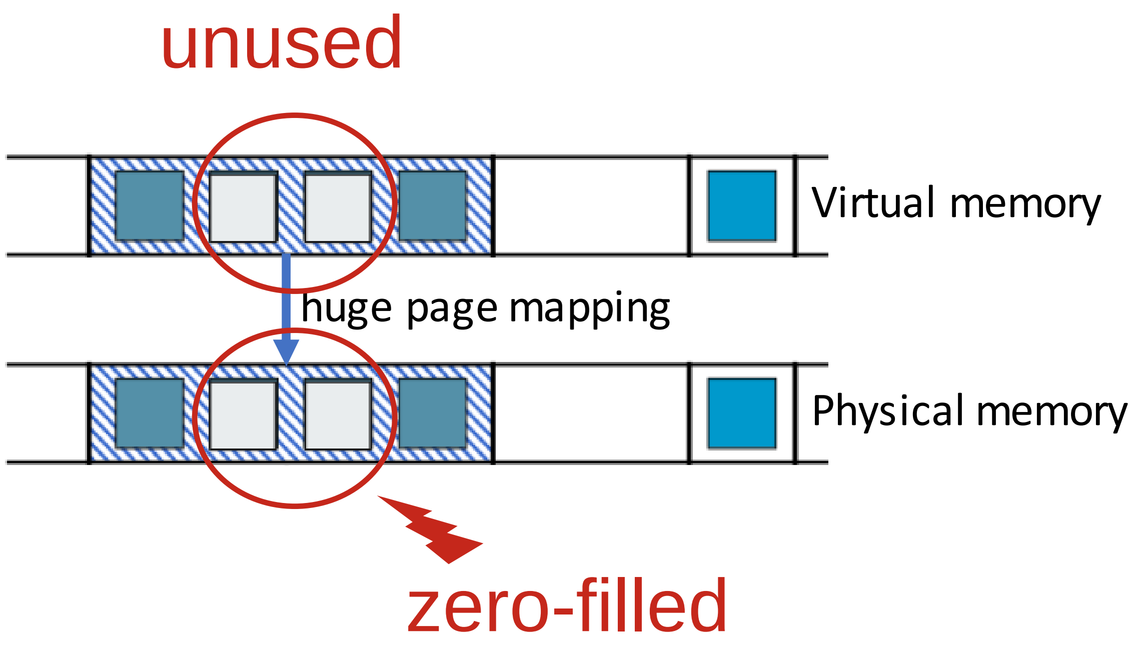
Mitigating bloat



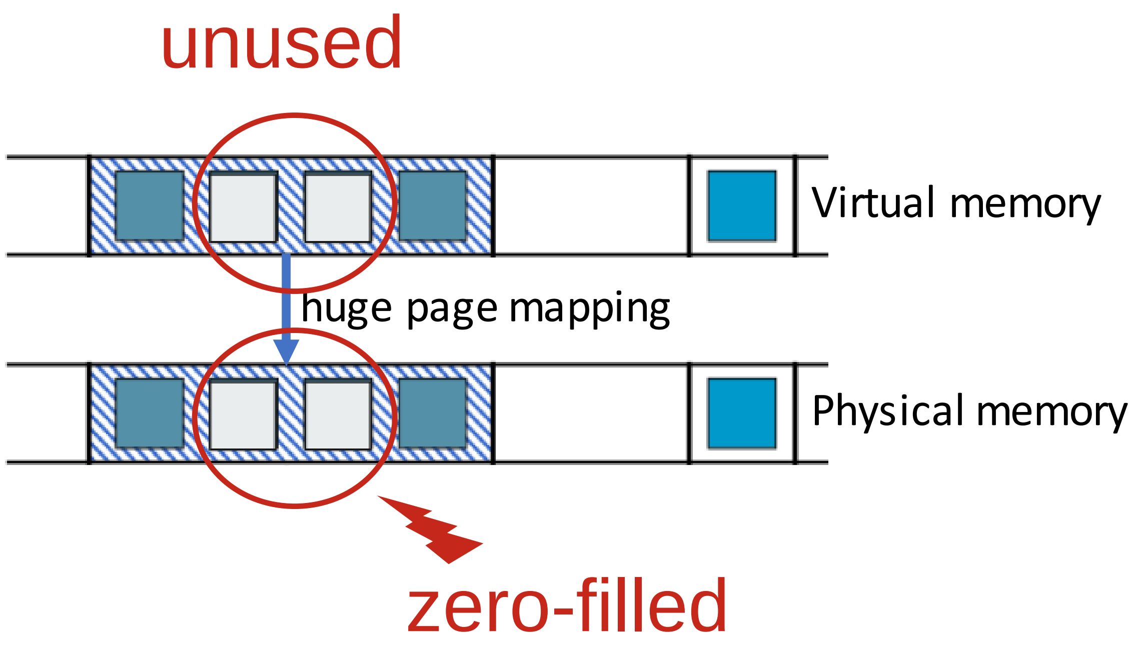
Mitigating bloat



Mitigating bloat



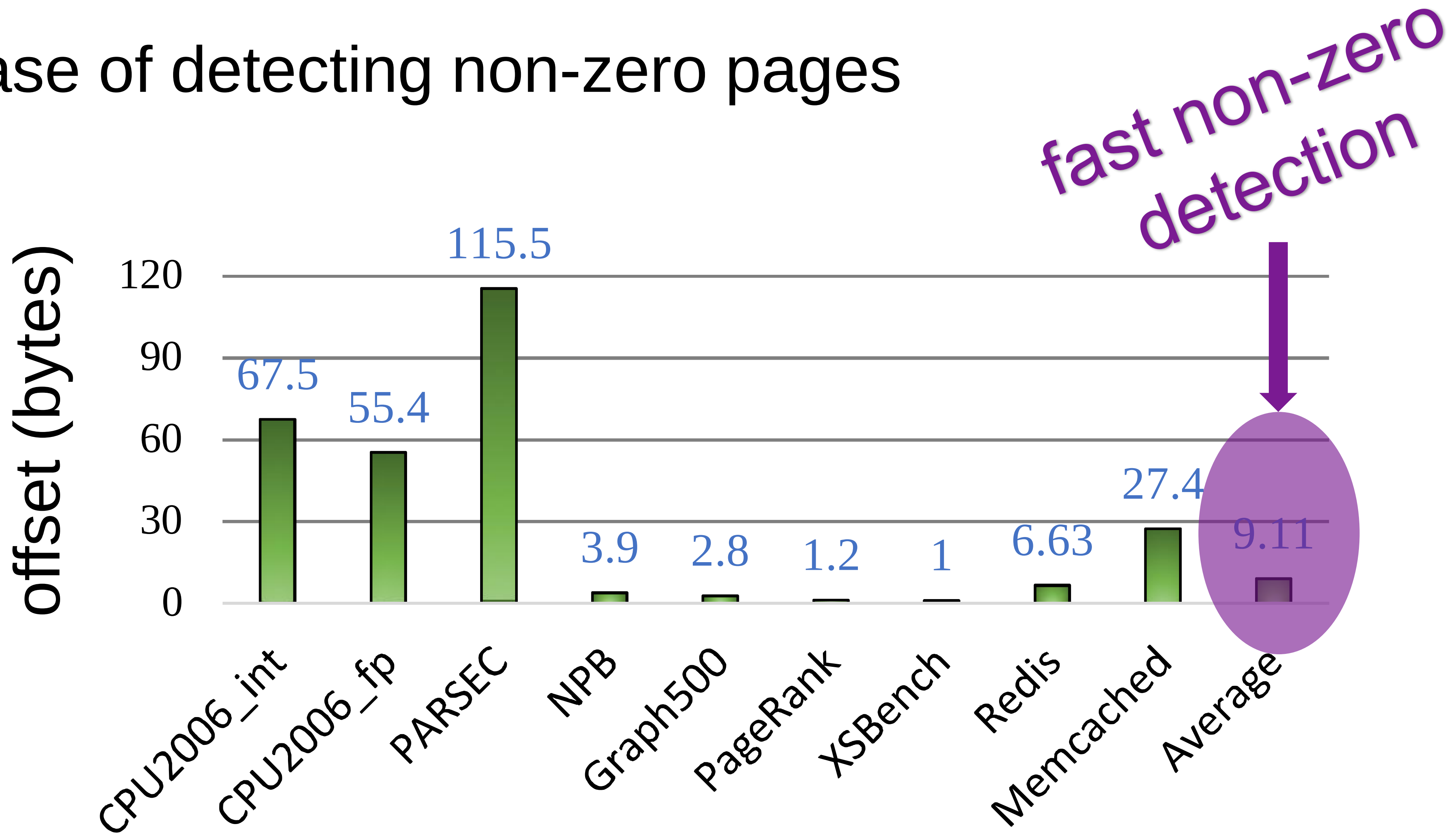
Mitigating bloat



- Observation: Unused base pages remain zero-filled
- Identify bloat by scanning memory
- Dedup zero-filled base pages to remove bloat

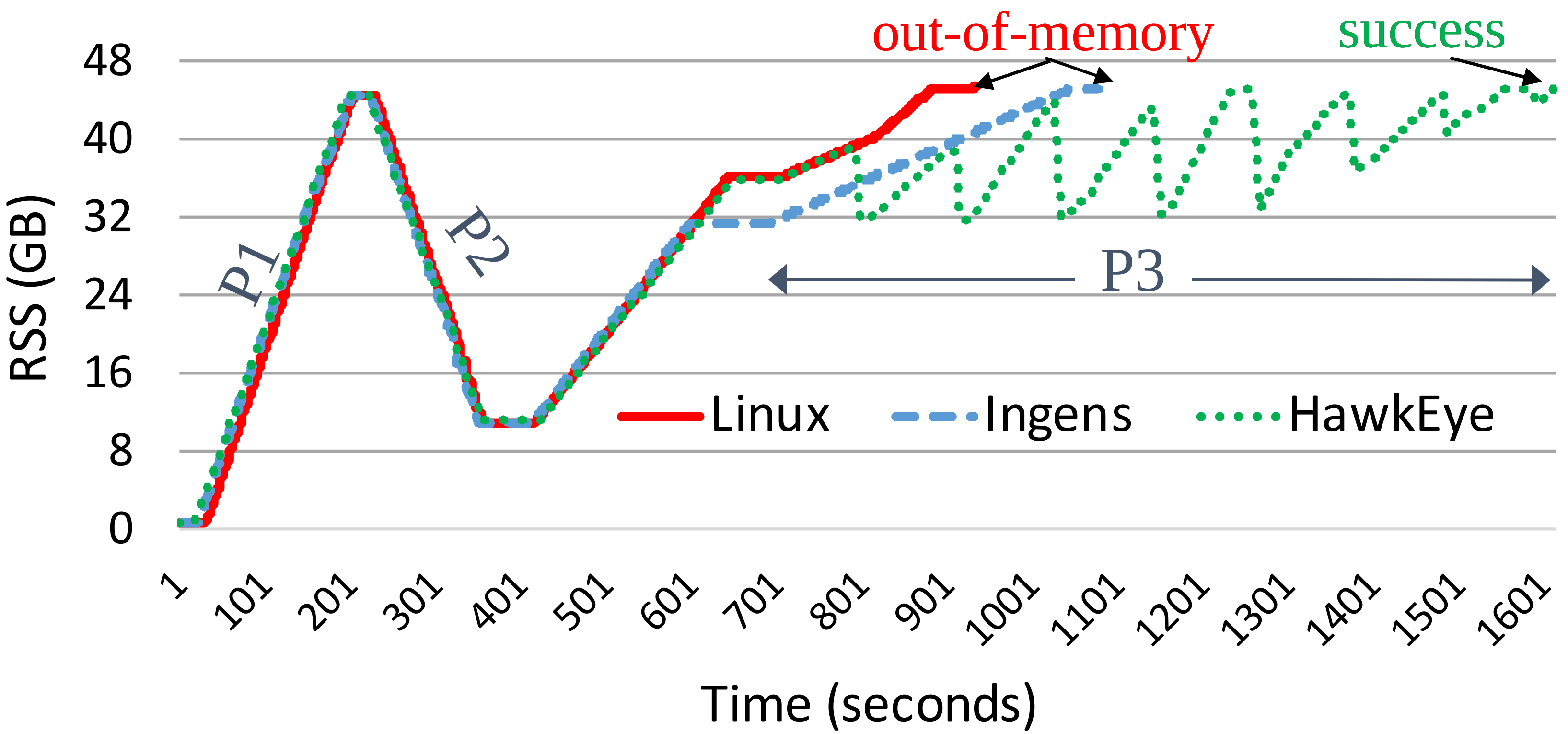
Mitigating bloat

- Ease of detecting non-zero pages



Mitigating bloat

✓ Automated "**bloat vs. performance**" management



Redis

P1: insert

P2: delete

P3: insert

	FreeBSD	Linux	Ingens	HawkEye
Memory bloat	Low	High	Tunable	Automated
Performance	Low	High	Tunable	Automated
Allocation latency	Low	High	Low	Low
# page faults	High	Low	High	Low

Tradeoff-1: Memory bloat ↔ Performance

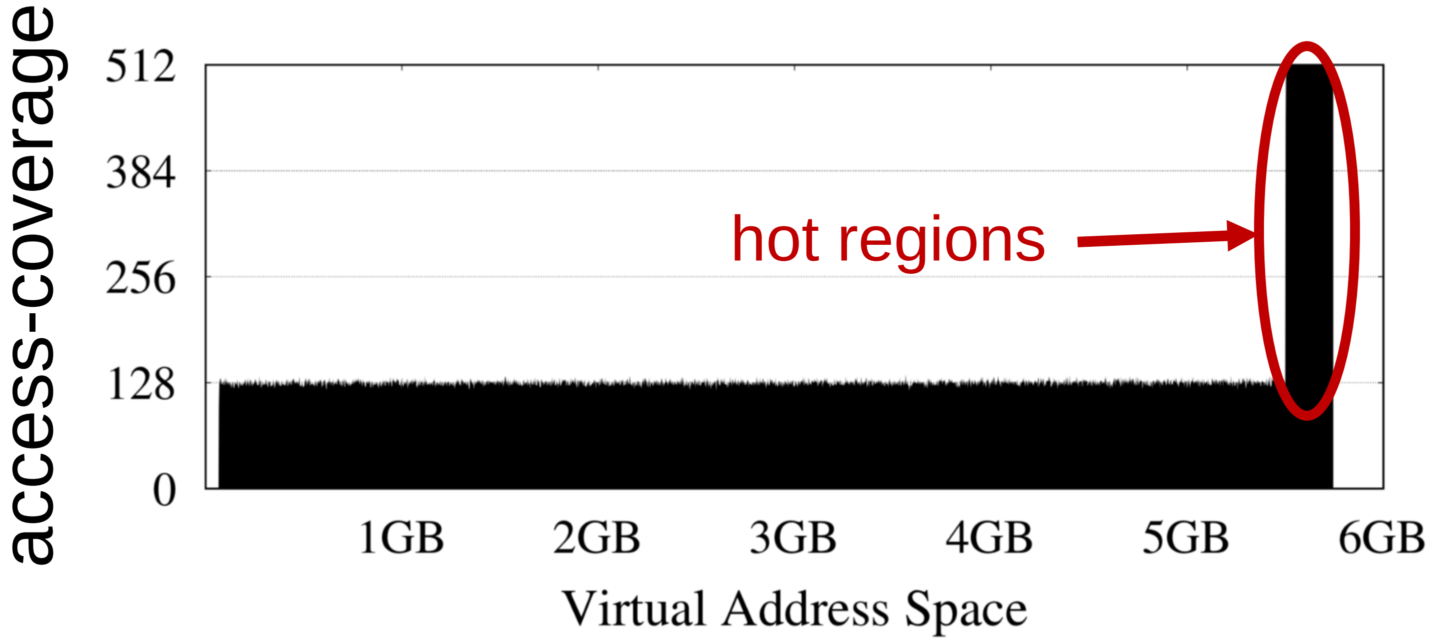
Tradeoff-2: Allocation latency ↔ # page faults

Fine-grained (intra-process) allocation

- Maximizing performance with limited contiguity

Fine-grained (intra-process) allocation

- Maximizing performance with limited contiguity



XSBench

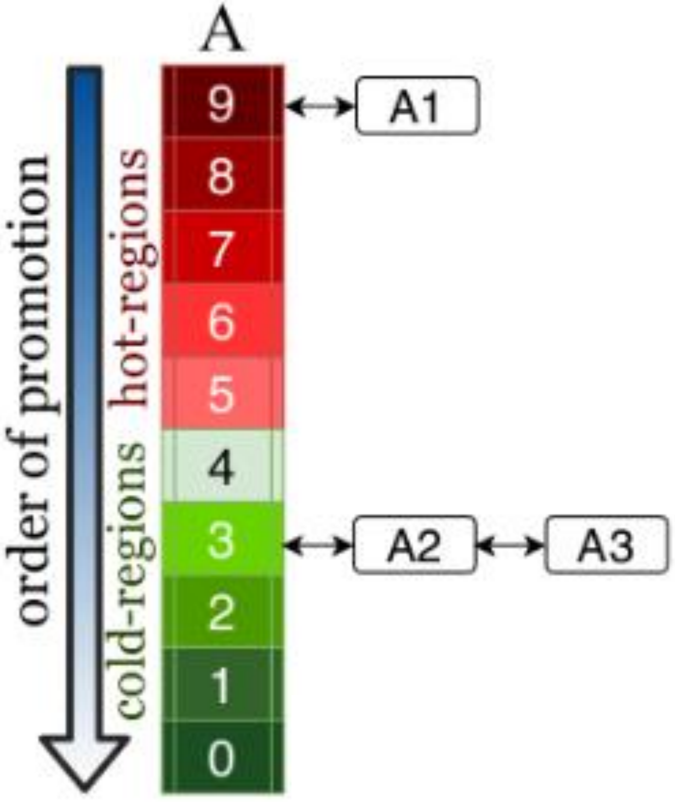
access-coverage: # base pages accessed per second

❖ A good indicator of TLB-contention due to a region

Fine-grained (intra-process) allocation

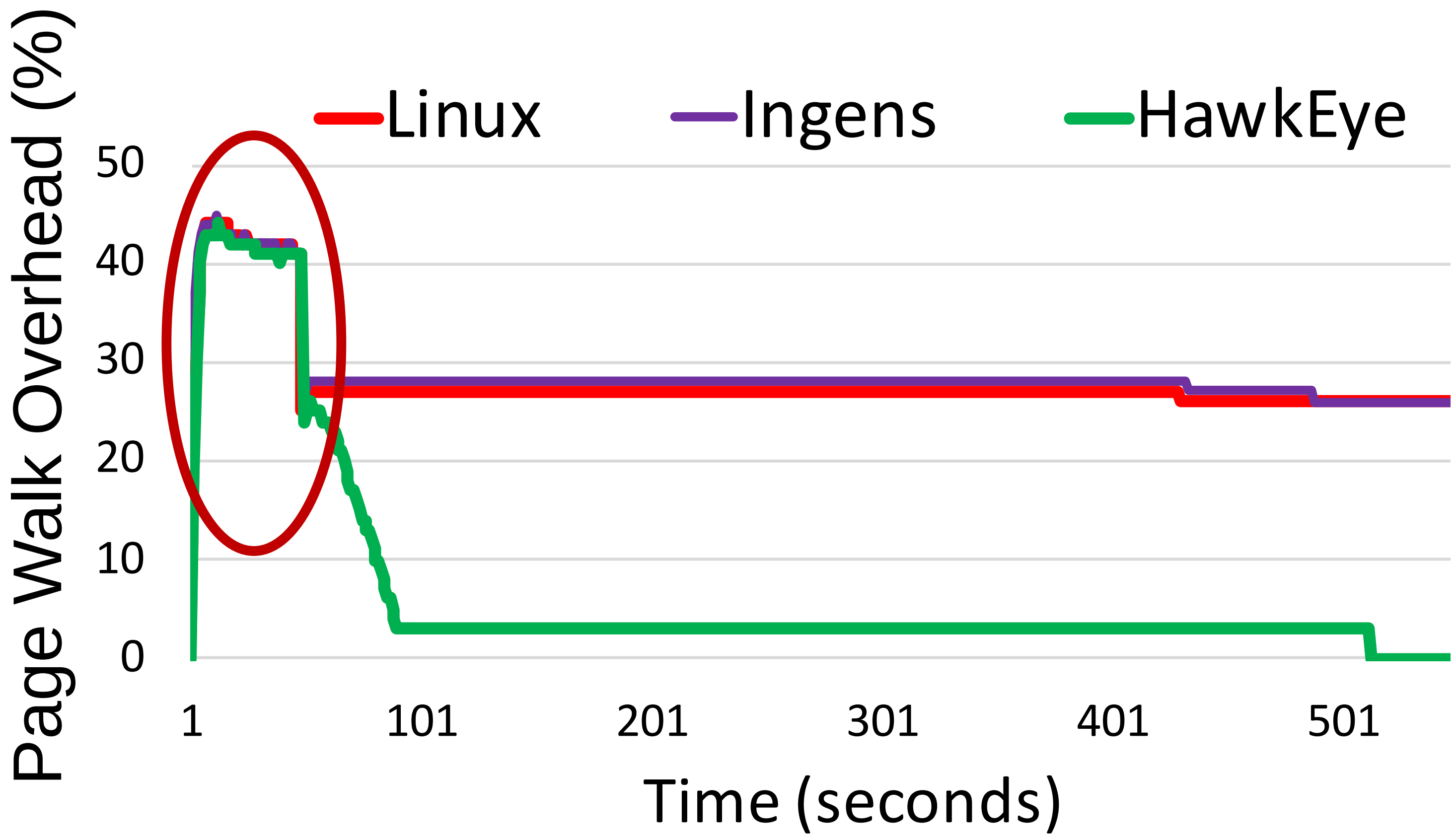
- Track access-coverage (**access_map)**
- Allocate in the sorted order
  (top to bottom)

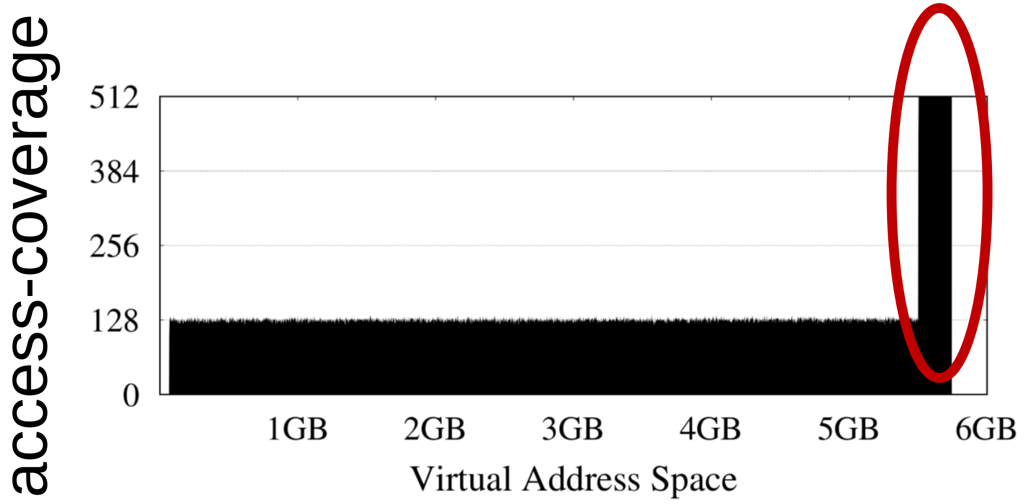
✓ Yields higher profit per allocation



access_map

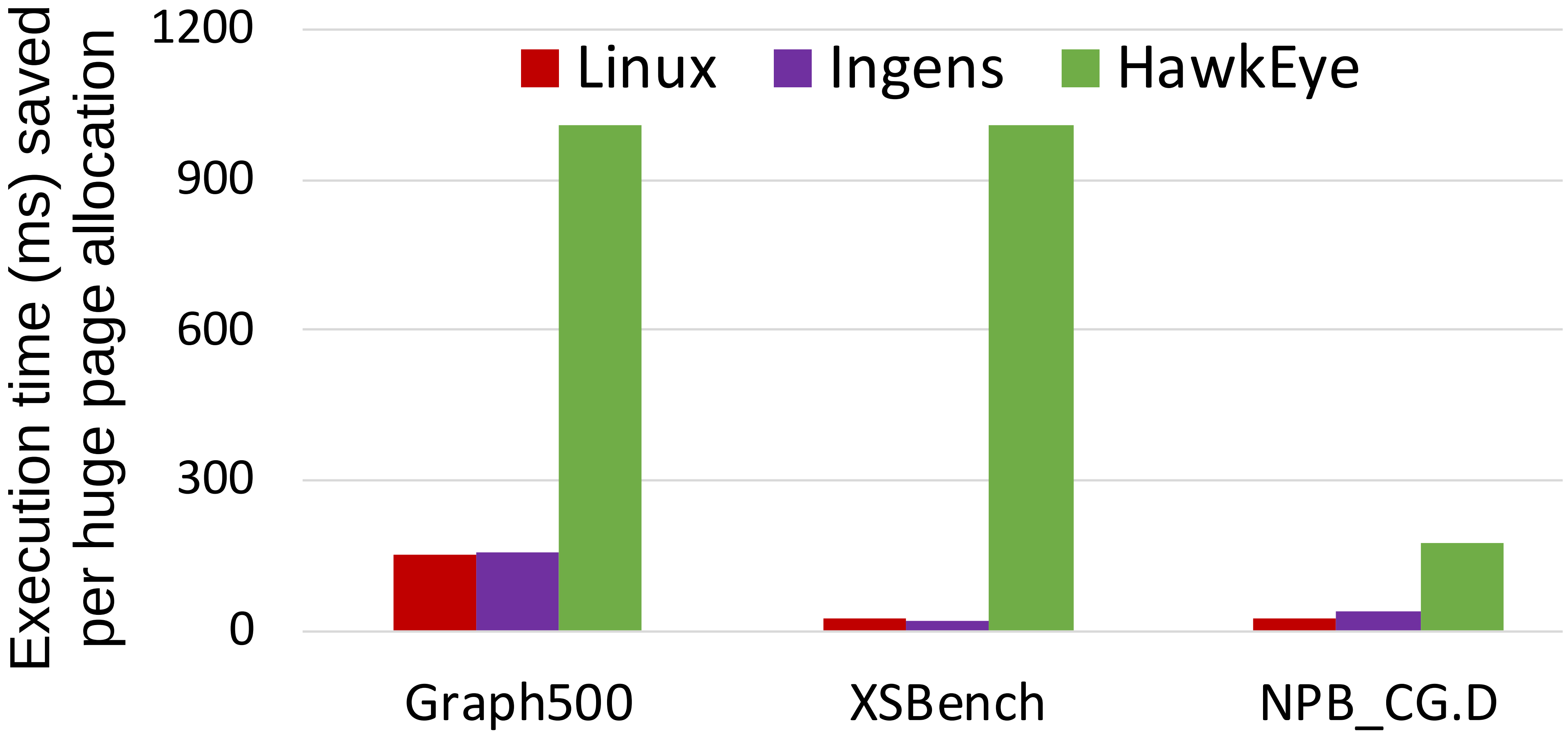
Fine-grained (intra-process) allocation





Workload: XSBench

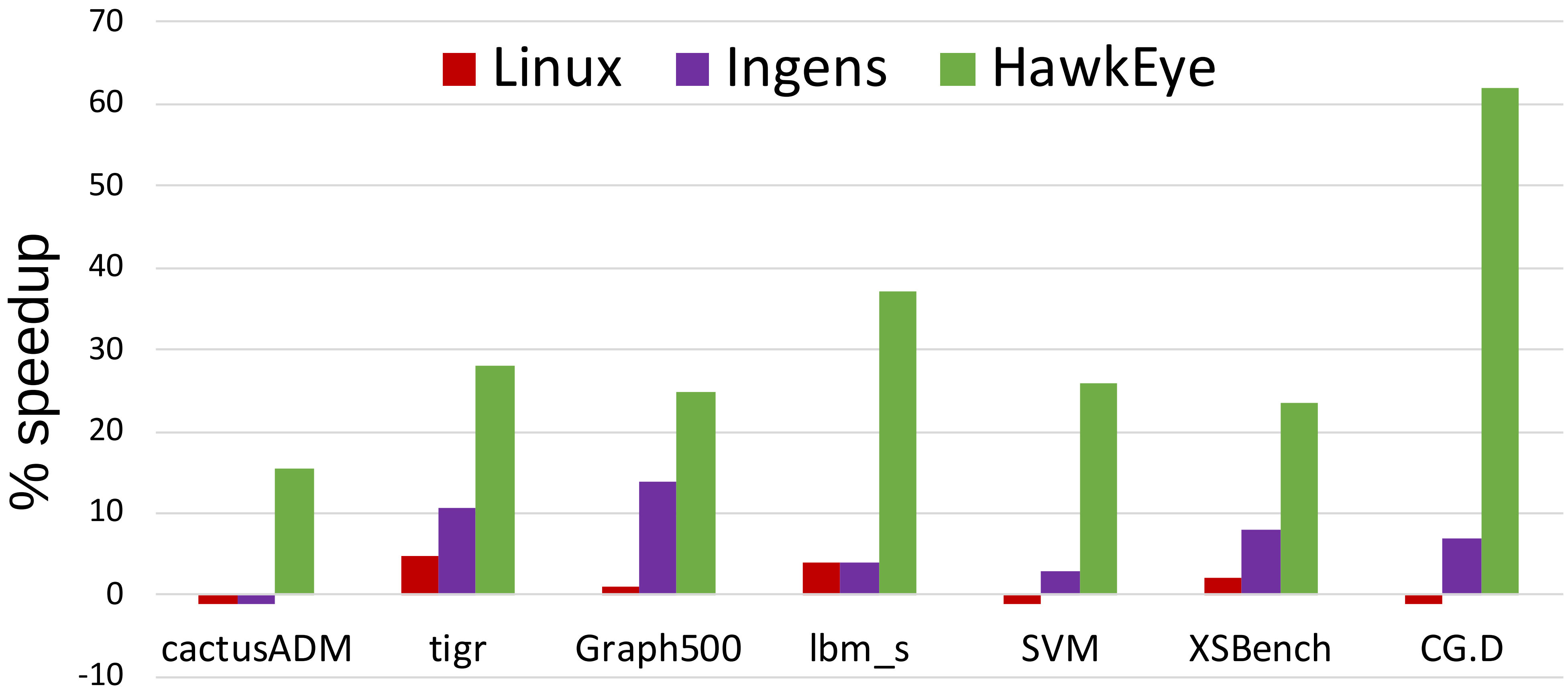
Fine-grained (intra-process) allocation



Fair (inter-process) allocation

- Prioritize allocation to the process with highest expected improvement
- How to estimate page walk overhead
  - Profile hardware performance counters
  - Low cost, accurate!

Fair (inter-process) allocation



Workloads running alongside a TLB-insensitive process

Summary

- OS support for huge pages involves complex tradeoffs
- Balancing fine-grained control with high performance
- Dealing with fragmentation for efficiency and fairness

Summary

- OS support for huge pages involves complex tradeoffs
- Balancing fine-grained control with high performance
- Dealing with fragmentation for efficiency and fairness

HawkEye: Resolving fundamental conflicts for huge page optimizations
https://github.com/apanwariisc/HawkEye

Thank You