Making Huge Pages *Actually* Useful

Ashish Panwar[1,2], Aravinda Prasad[1], K. Gopinath[1]

[1]Indian Institute of Science, [2]NetApp Inc.

Introduction

Huge pages can effectively alleviate the ever-increasing overheads of virtual-to-physical address translation. However, OSs have not been able to use them well because of the memory fragmentation problem despite hardware support for huge pages being available for nearly two decades.

- Hardware support demands contiguous memory for huge pages (limited by fragmentation).
- **Observation:** Current fragmentation handling policies induce poor decision making and unnecessary work in critical code paths while allocating huge pages.
- **The root cause:** Unmovable (kernel) pages—most kernel pages are directly mapped into physical memory.

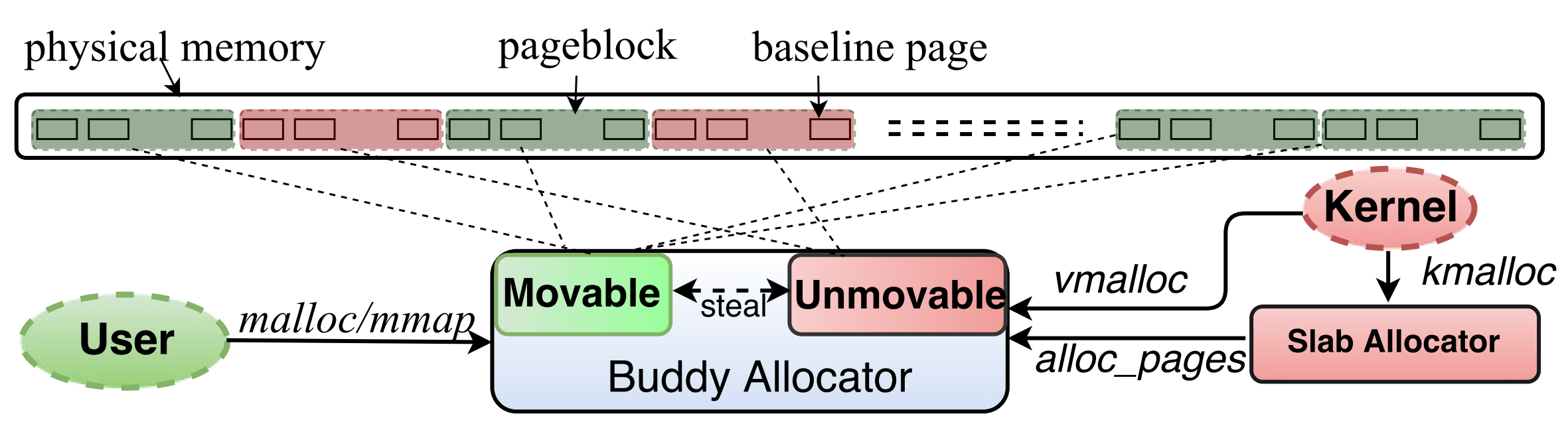

Figure 1: Physical memory management in Linux.

Memory Management in Linux

Anti-fragmentation:

- Partitions memory at the granularity of huge page sized regions (pageblocks) and uses dedicated regions for allocating kernel and user pages.
- Prevents fragmentation occurrence by clustering alike allocations together (less pollution).

Memory Compaction:

- Migrate pages to restore memory contiguity.
- Skip kernel regions (colored red).

Issues with Linux Memory Manager

- The two-way partitioning based anti-fragmentation makes hybrid pageblocks (i.e., pageblocks that contain both movable and unmovable pages) invisible.
- **Fragmentation-via-pollution:** Memory is polluted unnecessarily due to poor decision making in the page allocation path.

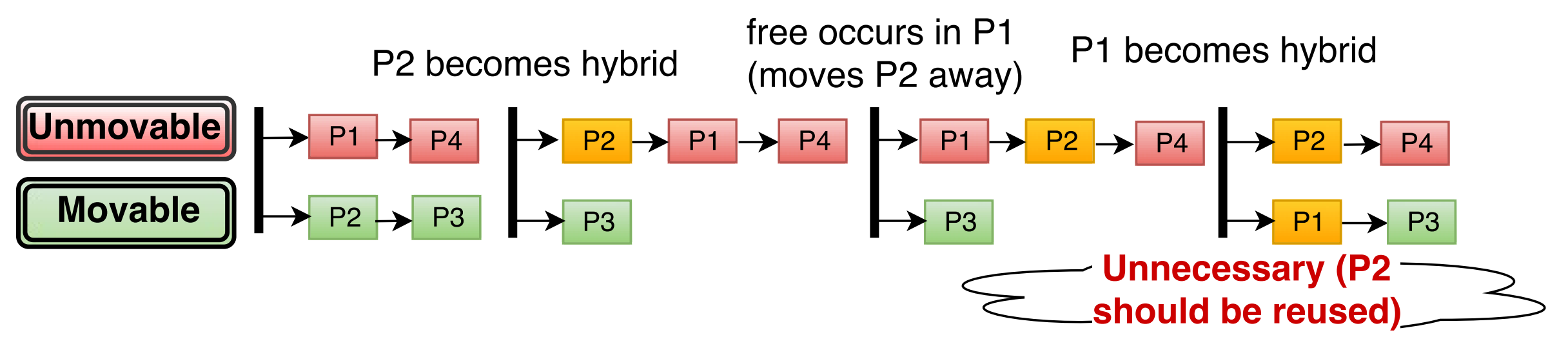
Unmovable
Movable
P2 becomes hybrid
free occurs in P1 (moves P2 away)
P1 becomes hybrid
P1
P2
P3
P4
Unnecessary (P2 should be reused)

Figure 2: A simple example showing how fragmentation-via-pollution occurs due to the invisibility of hybrid (yellow) pageblocks.

- **LIU (latency-inducing unsuccessful) migration:** The kernel migrates pages from hybrid pageblocks while attempting to allocate huge pages due to poor decision making in memory compaction routine.
- Unfortunately, the (f)utility of LIU migration is never realized—in pathological cases, the kernel can get into a compaction loop.

Performance Implications

- High rate of pollution limits huge page allocations—performance loss due to increased TLB pressure.
- LIU migration increases memory traffic, TLB shootdowns, cache pollution and eventually leads to high-latency huge page allocations and higher system time.
- Performance with huge pages can be even worse than base pages in a fragmented system!
- Performance isolation—a workload can exploit LIU migration to impact other applications.
- Large memory large problems—all memory sizes eventually get fragmented.

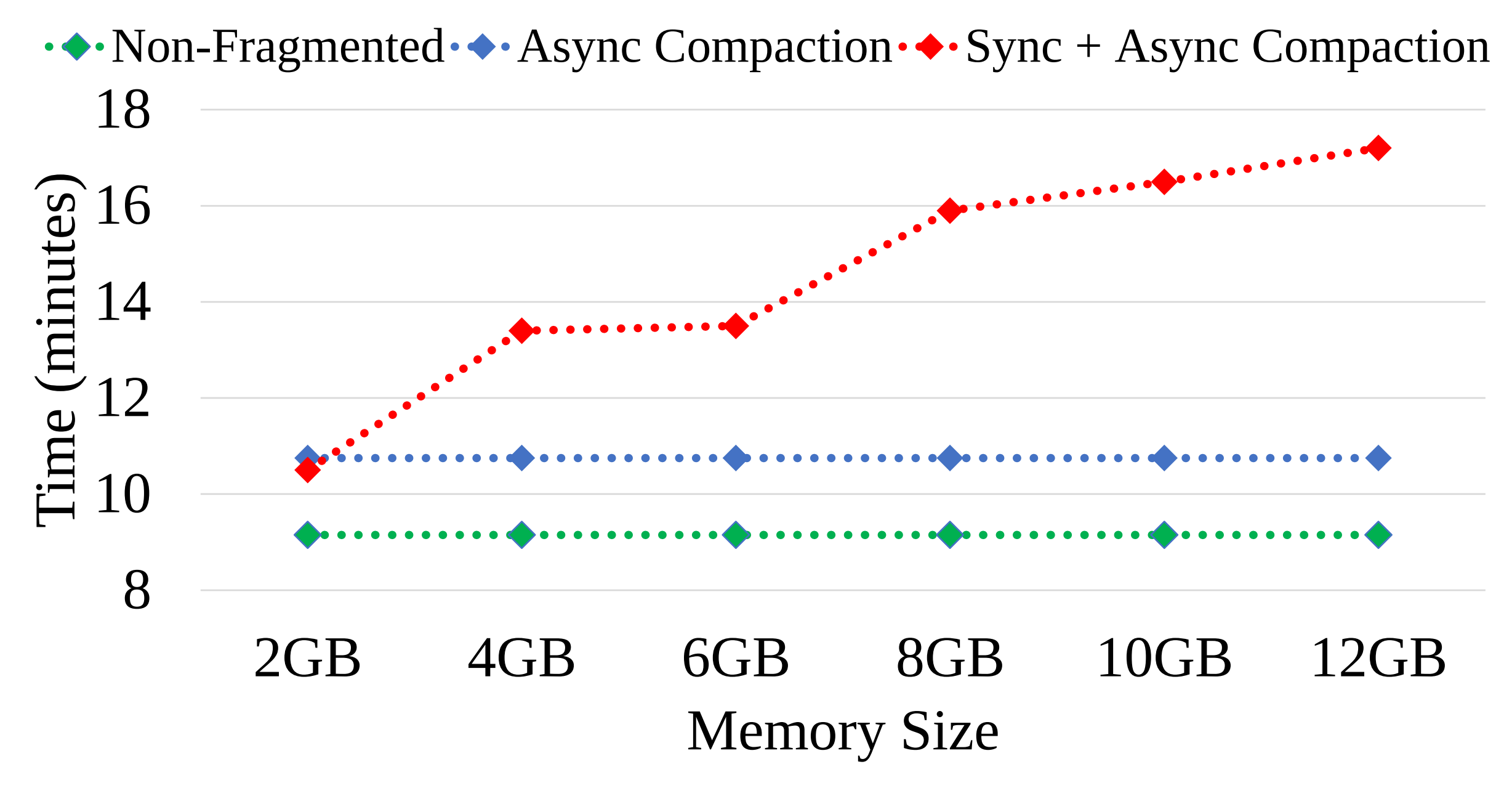

Figure 3: Execution time of `milc` (from SPEC CPU2006).

Huge Pages and Virtualization

Virtualization can exacerbate fragmentation related performance anomalies with huge pages as both guest and host OSs may perform unnecessary work.

Illuminator

Illuminator explicitly manages hybrid pageblocks to present a precise view of each pageblock's mobility thus enabling informed decision making across subsystems while dealing with fragmentation.

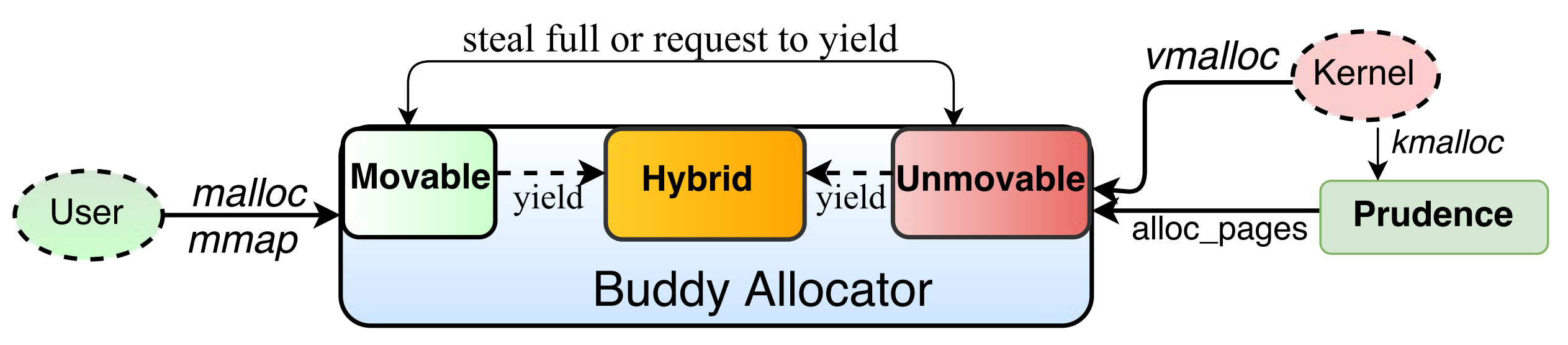

Figure 4: Physical memory management in Illuminator.

- Provides better control over the rate of pollution.

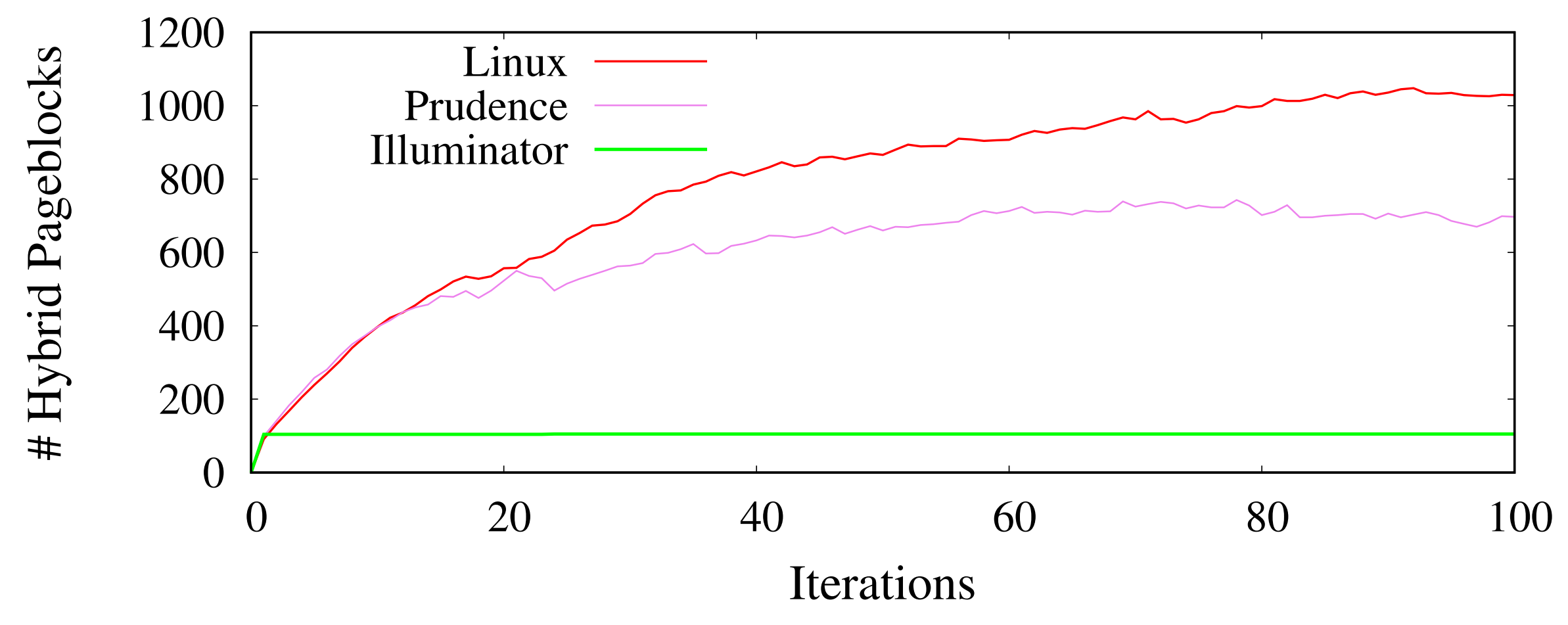

Figure 5: Rate of pollution with different memory allocators.

- Eliminates LIU migration—not subjected to pathological performance anomalies.
- Kernel memory pressure is reduced by replacing the slab allocator with Prudence [1].

Results

- 8-core Intel Ivy-Bridge server running with Linux kernel v4.5. KVM hypervisor is used for experiments in a virtualized system.
- Page sizes: Base—4KB, Huge—2MB.

	At rest		Under stress	
	Linux	**Illuminator**	**Linux**	**Illuminator**
Min	67%	68%	3%	12%
Max	72%	72%	7%	28%
Avg	69%	69%	5%	17%

Table 1: Huge page allocation success rate.

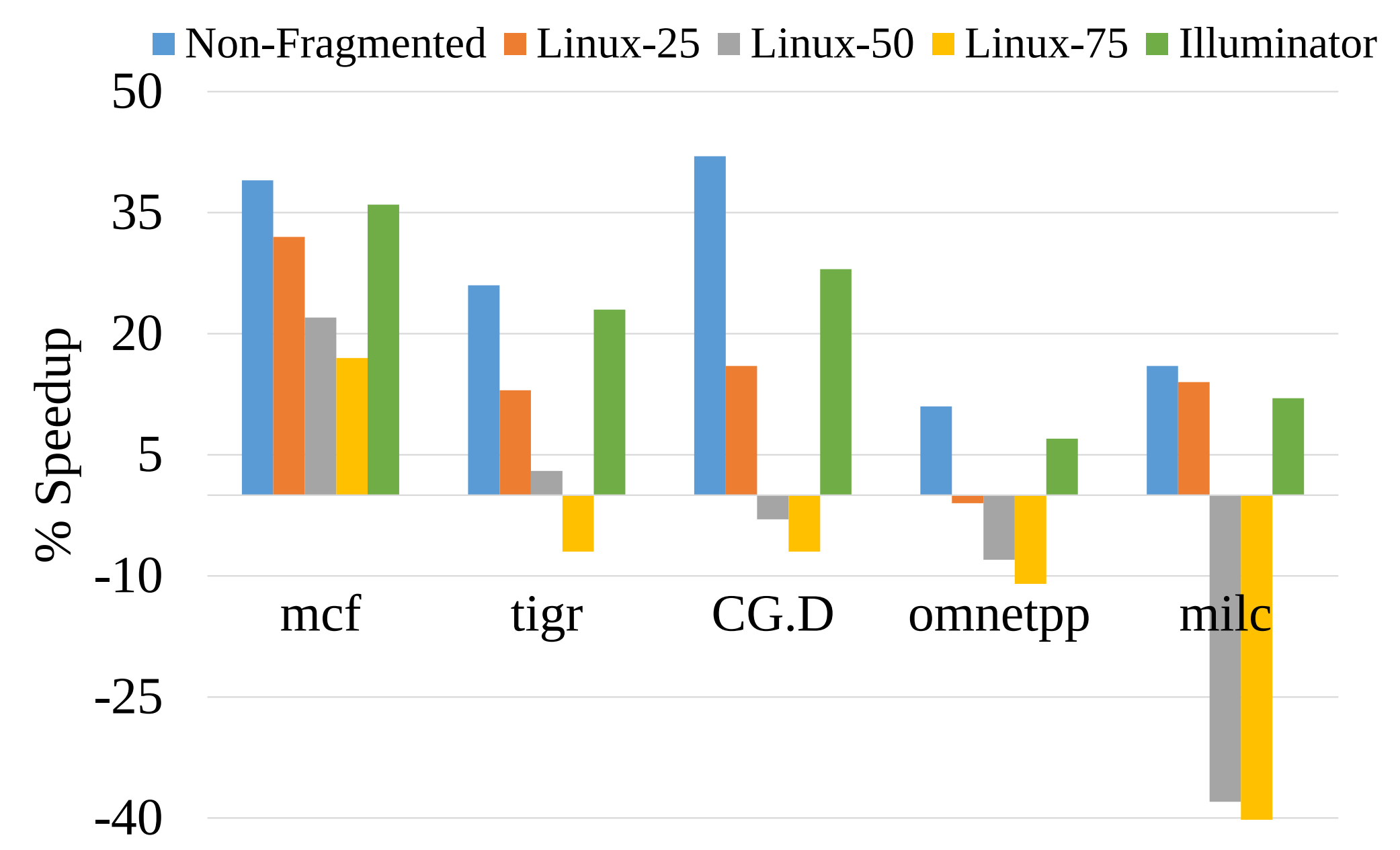

Figure 6: Speedup over base pages (numbers in the legend represent the percentage of pageblocks polluted by unmovable pages).

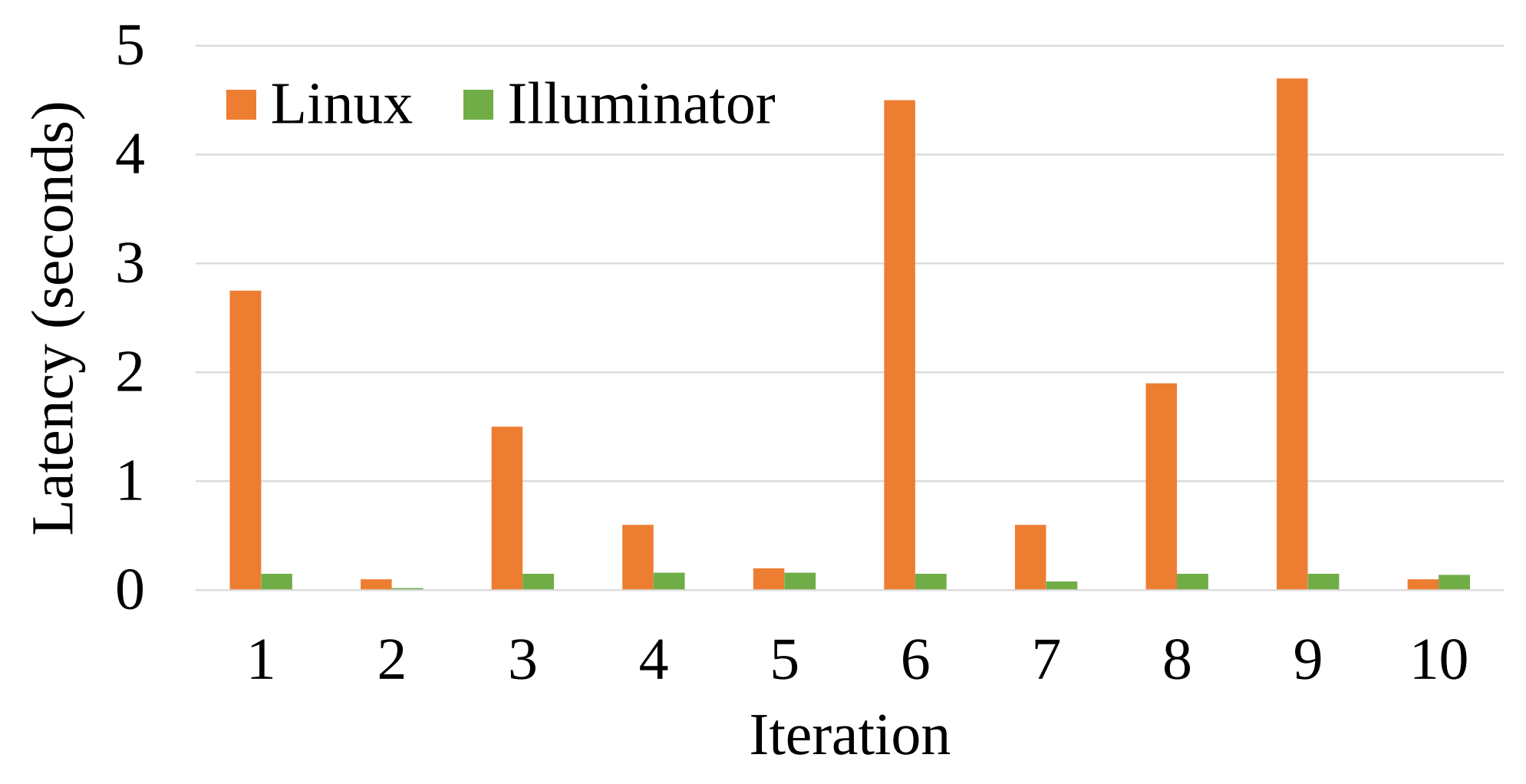

Figure 7: Max latency for MySQL database server.

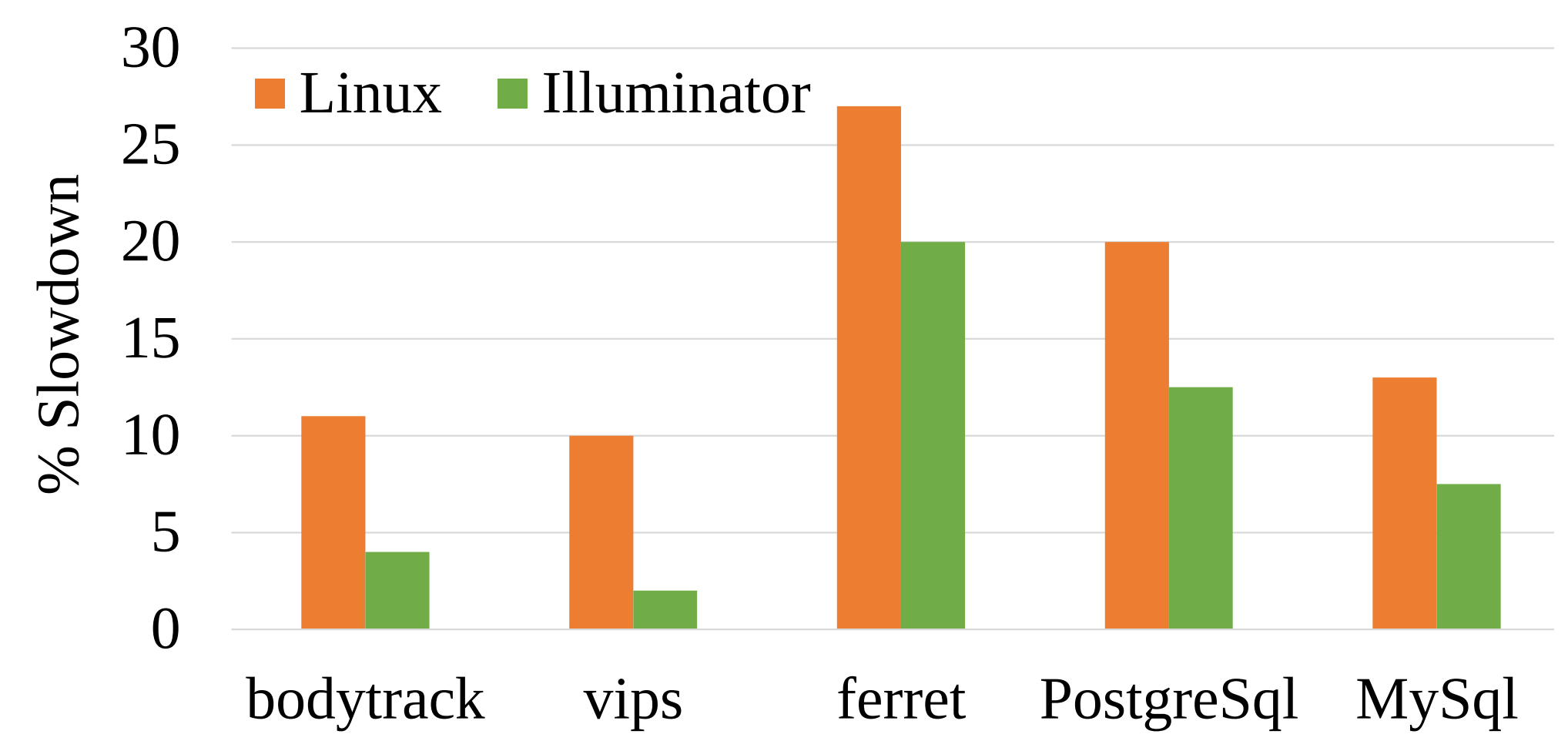

Figure 8: Illuminator provides better performance isolation.

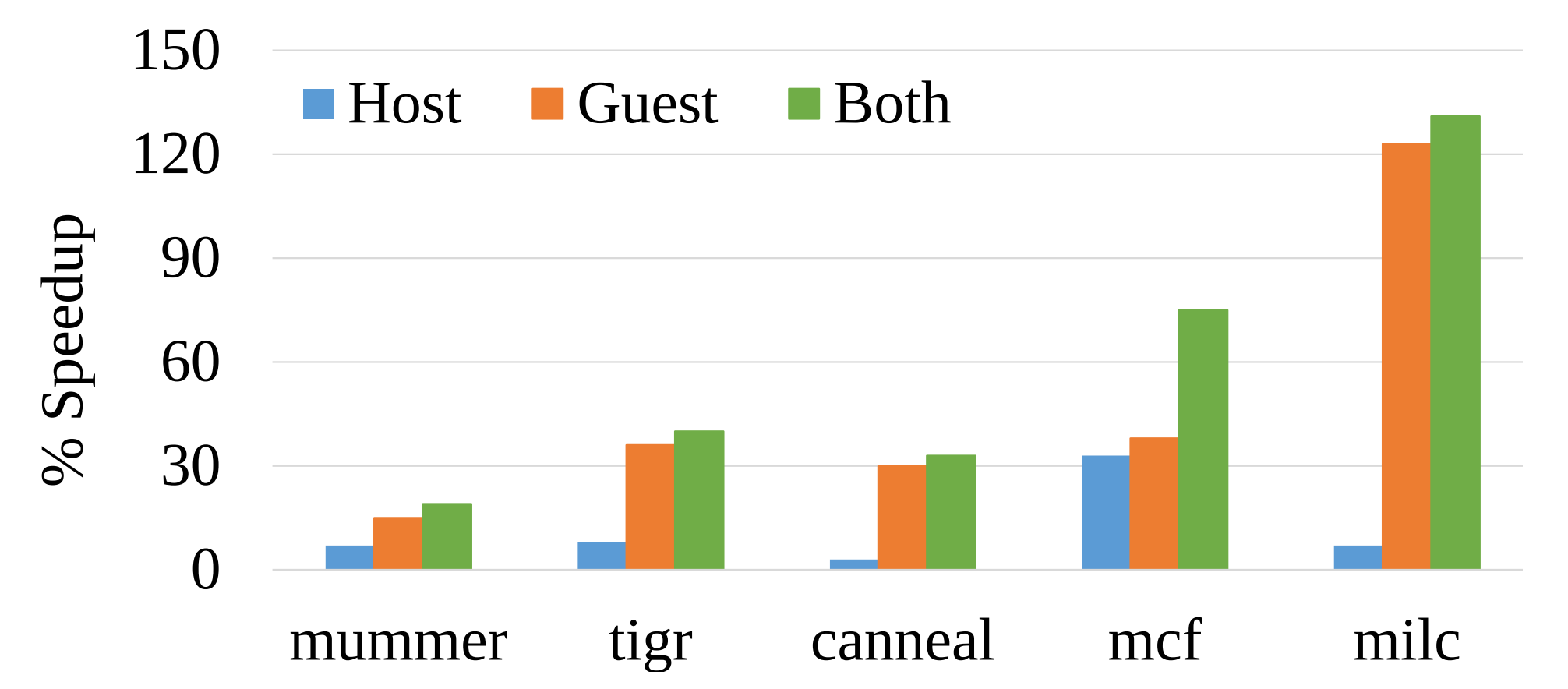

Figure 9: Performance improvement over Linux in a virtualized system when Illuminator is applied at different layers.

References

[1] "Prudent Memory Reclamation in Procrastination-Based Synchronization", Aravinda Prasad, K. Gopinath, ASPLOS 2016.

[2] Illuminator source is available at: `https://github.com/apanwariisc/Illuminator`