Non-Blocking Writes to Files

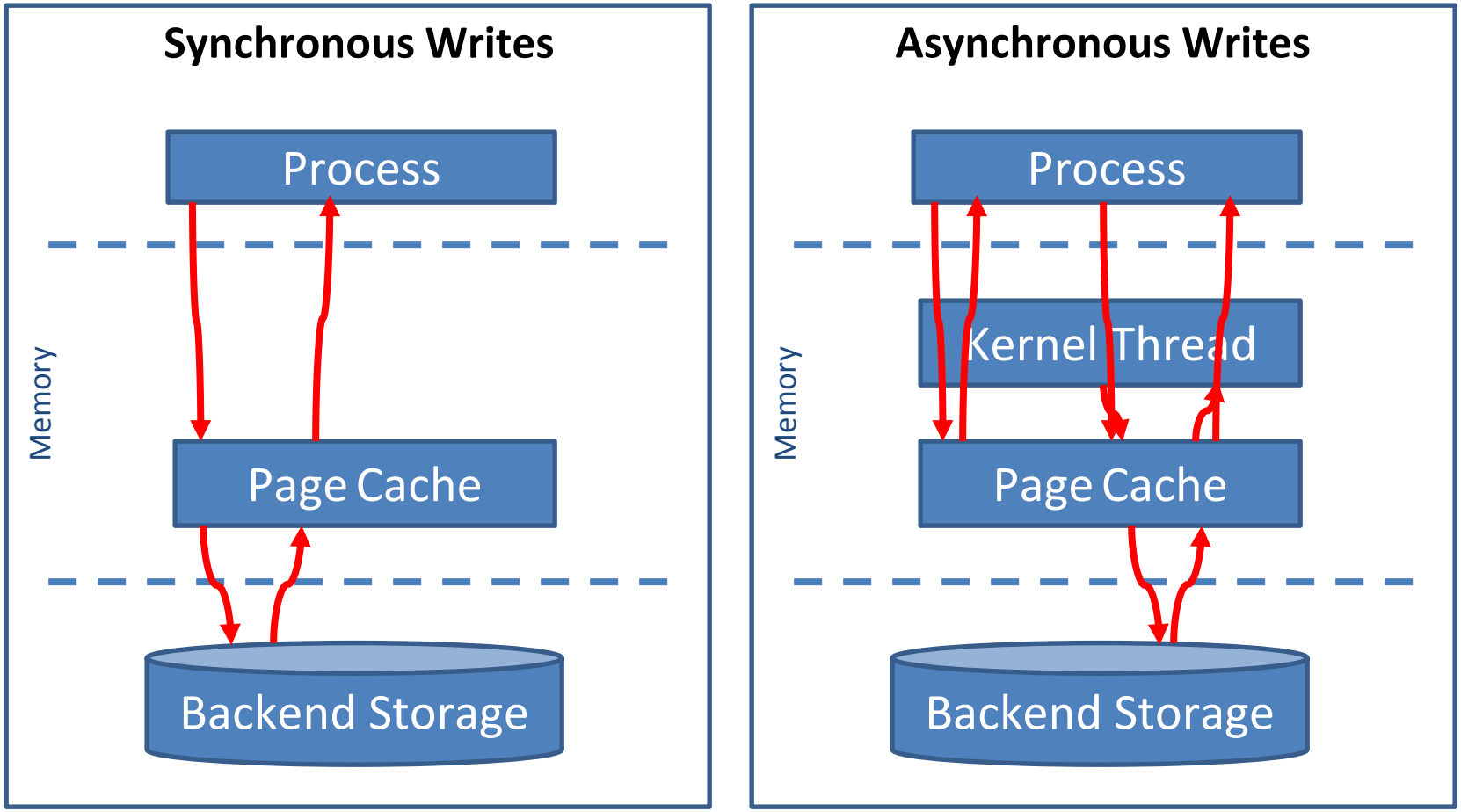
Daniel Campello, Hector Lopez, Luis Useche[1],
Ricardo Koller[2], and Raju Rangaswami

FIU | School of Computing & Information Sciences

[1]Google, Inc. [2]IBM TJ Watson

Synchrony vs Asynchrony

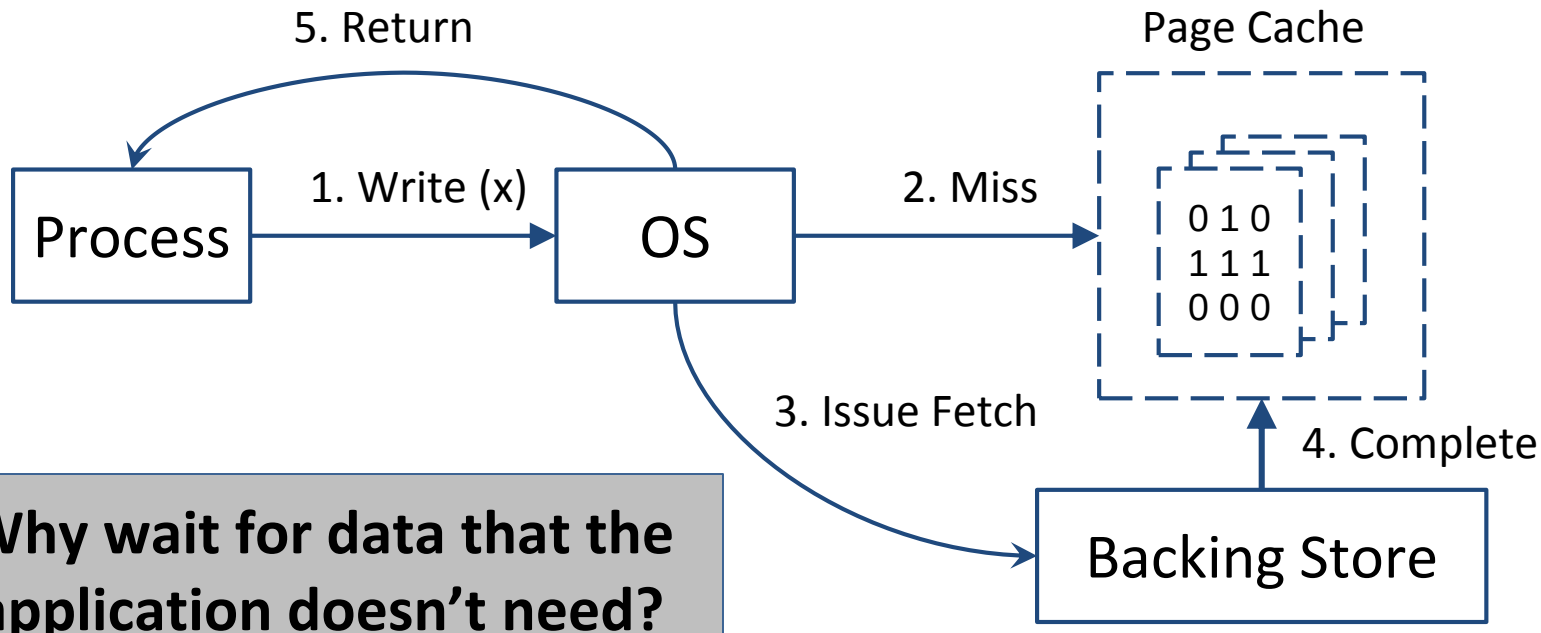
Applications have different alternatives to persist data



Introduction

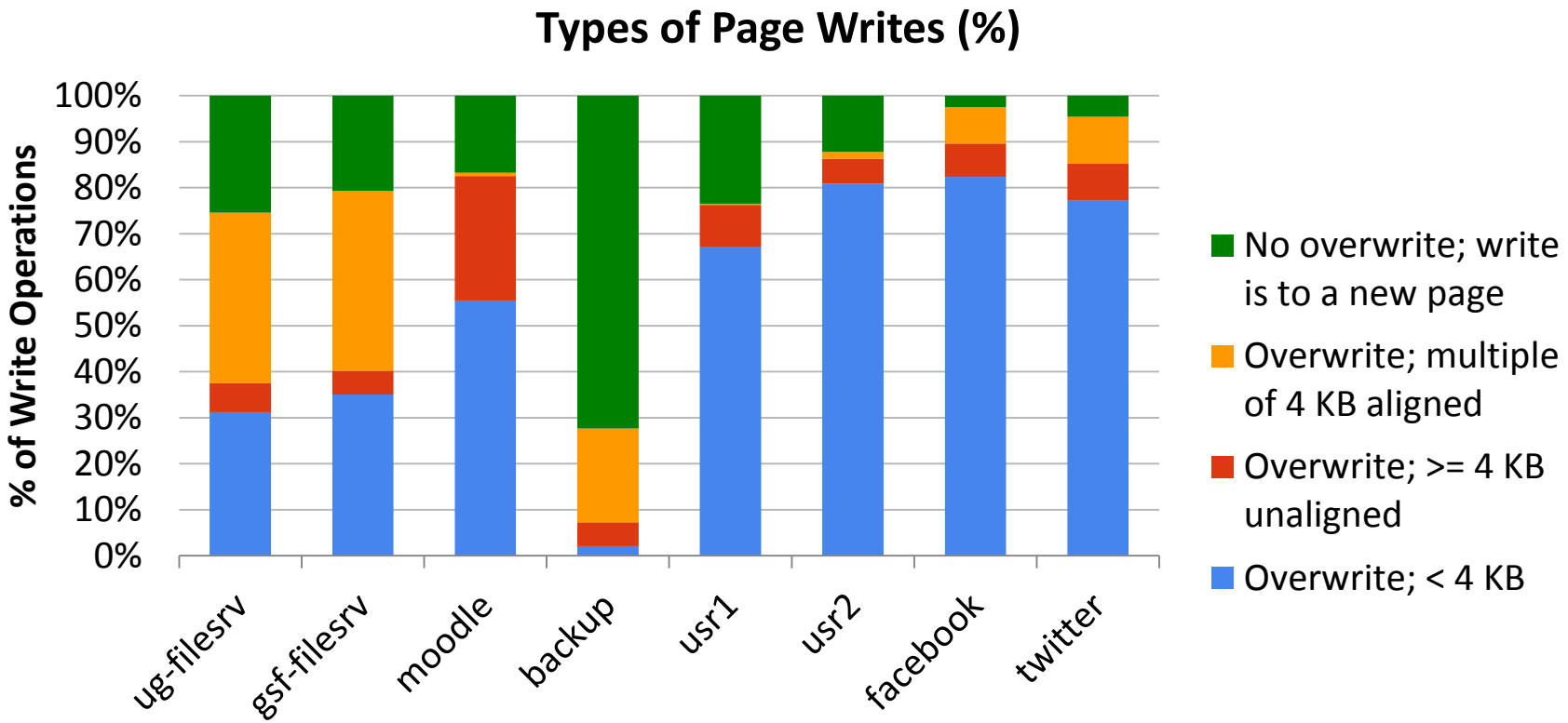
Memory access granularity is smaller than disk's

⇒ Partial writes to pages not present in the cache require page fetch.



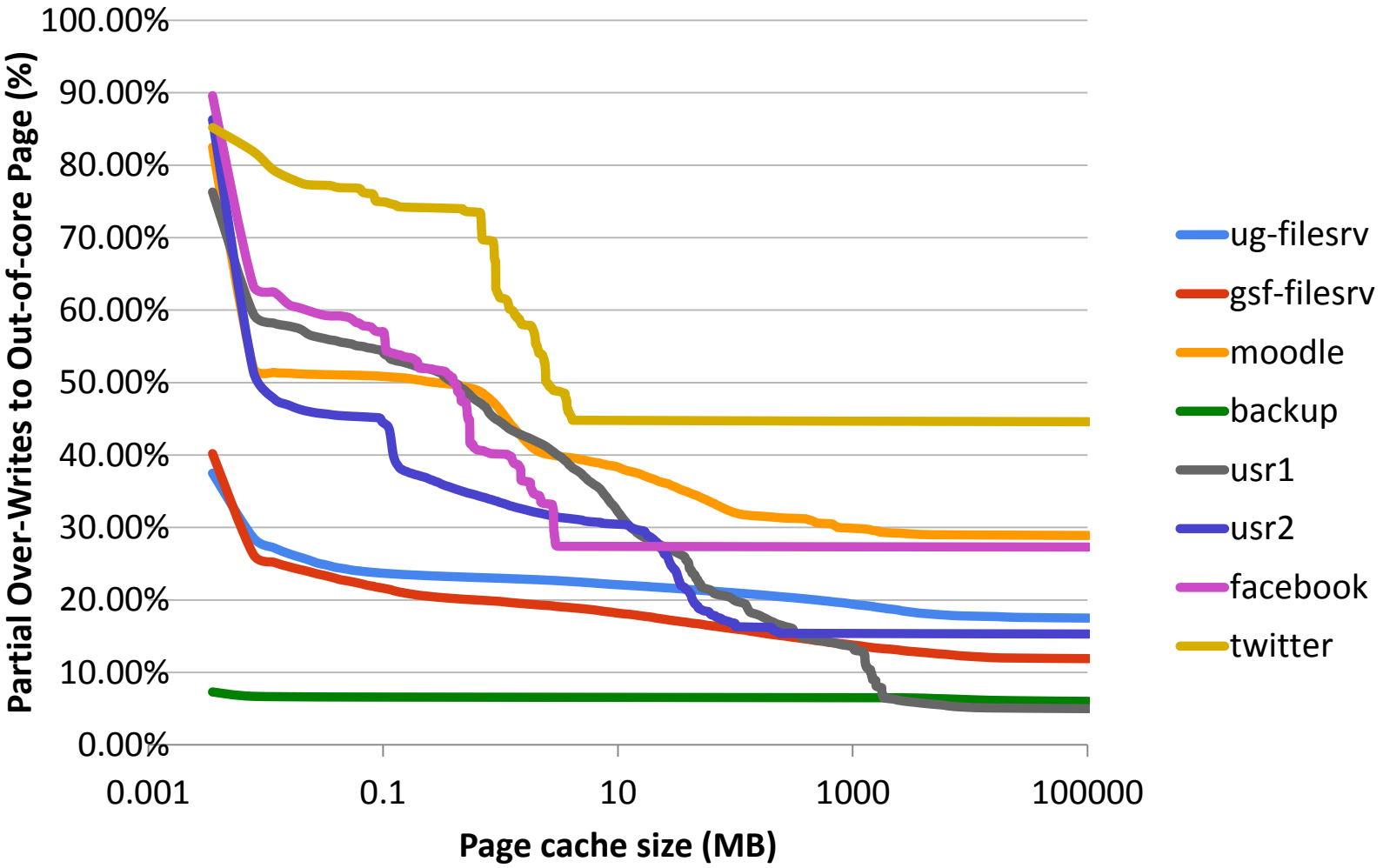
Why wait for data that the application doesn't need?

Motivation: Breakdown of write operations

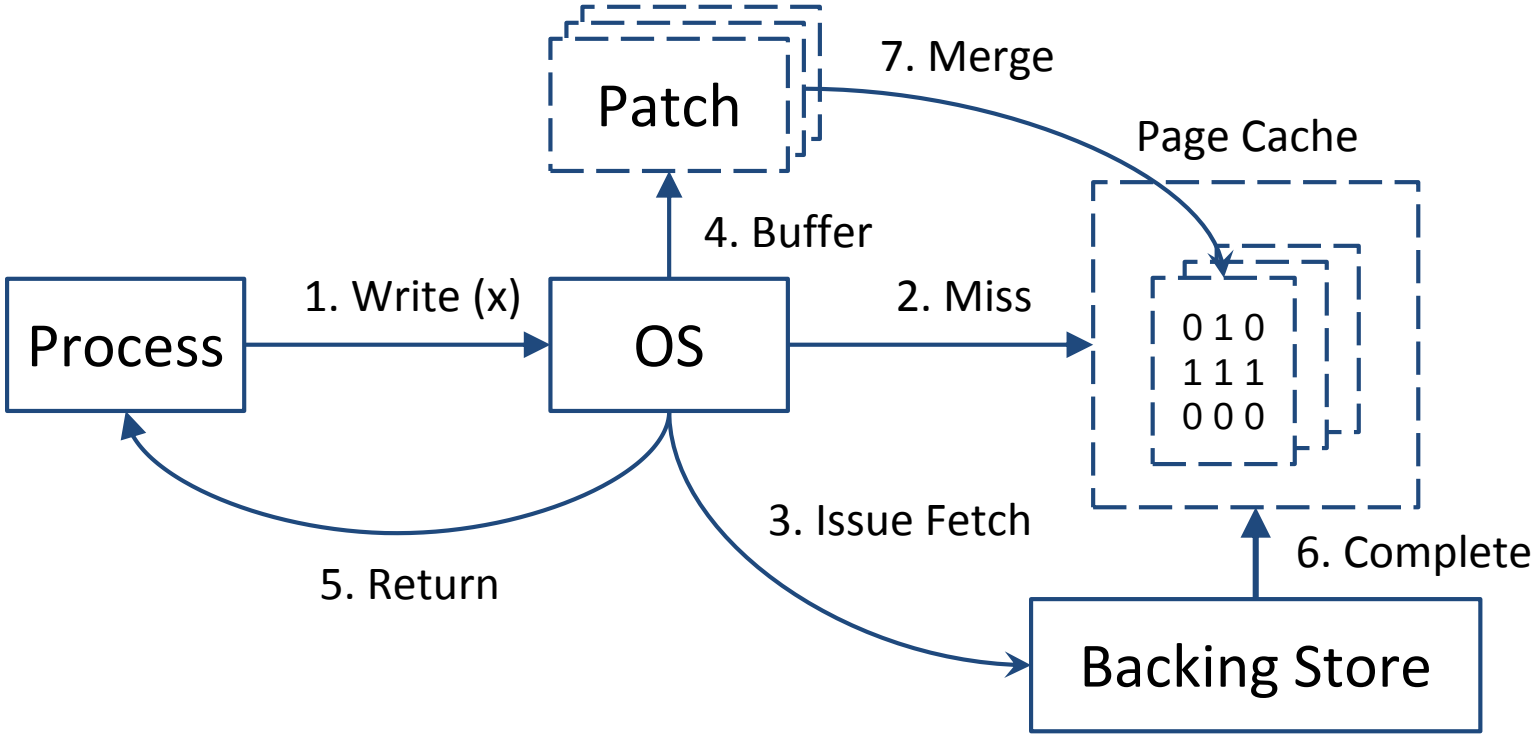


- On an average, 63.12% of the writes involved partial page overwrites.
- Depending of page cache size, these overwrites could result in varying degrees of page fetches.

Motivation: LRU Cache Simulation



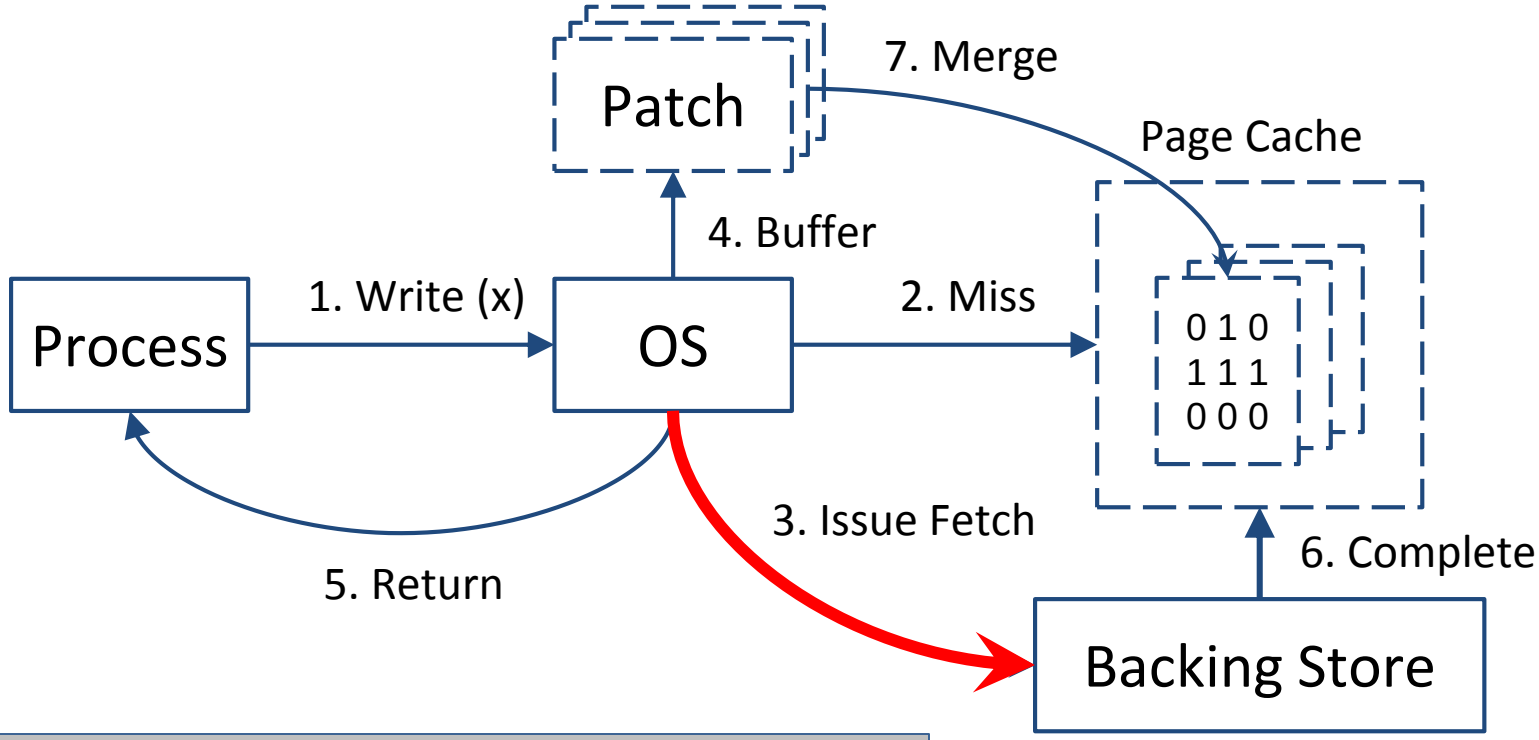
Non-blocking Writes: Asynchronous Fetch



Benefits:

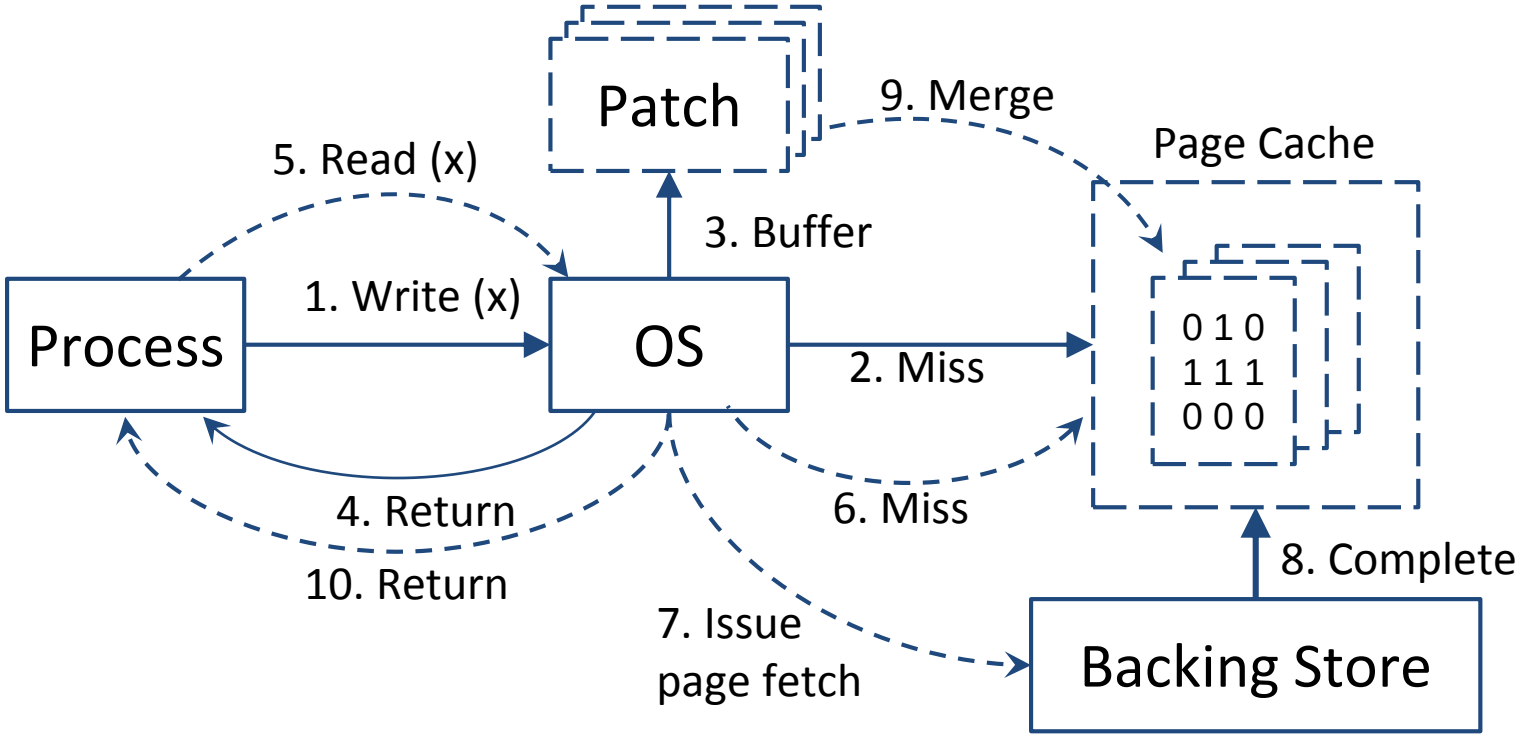
1. Application execution time reduction
2. Increased backing store bandwidth usage

Non-blocking Writes: Asynchronous Fetch



Why issue a page fetch for data that the application doesn't need?

Non-blocking Writes: Lazy Fetch



Benefits and Drawbacks:

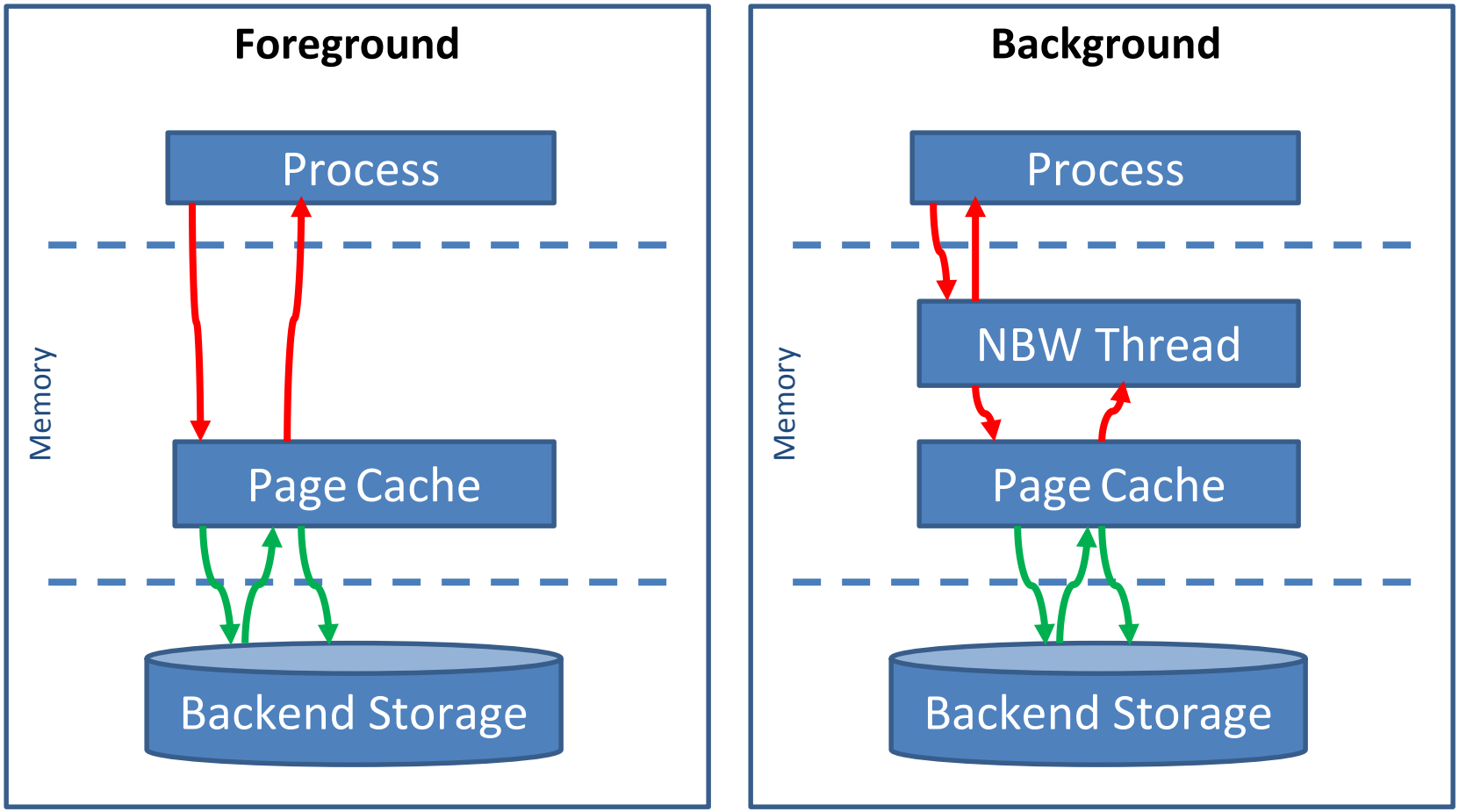
1. Allocating page memory and fetching only if necessary
2. Resource utilization is unpredictable and can be bursty

Managing Page Fetches

Page Fetch Policies	Page Fetch Mechanisms
Asynchronous	Foreground
Lazy	Background

Foreground vs Background Mechanisms

Metadata misses generate extra blocking fetches



Evaluation Specifics

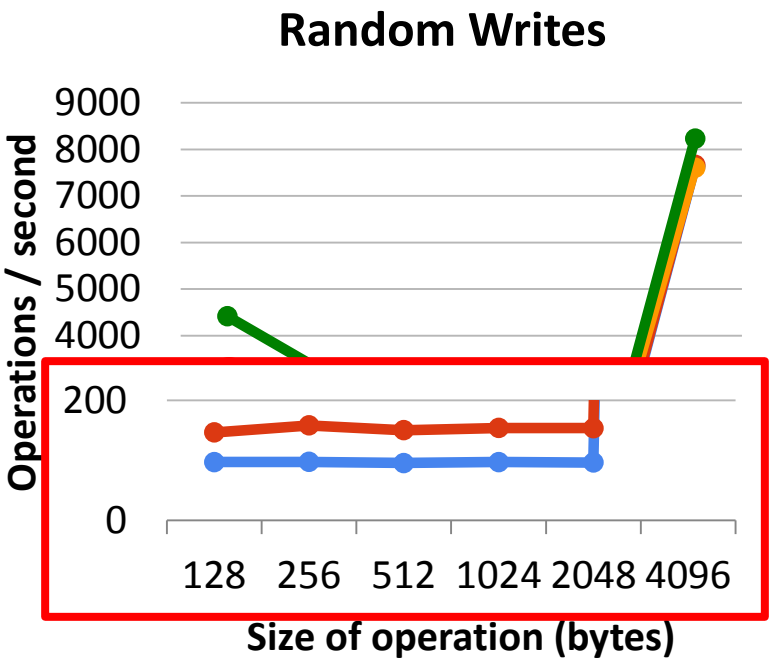
Implementation

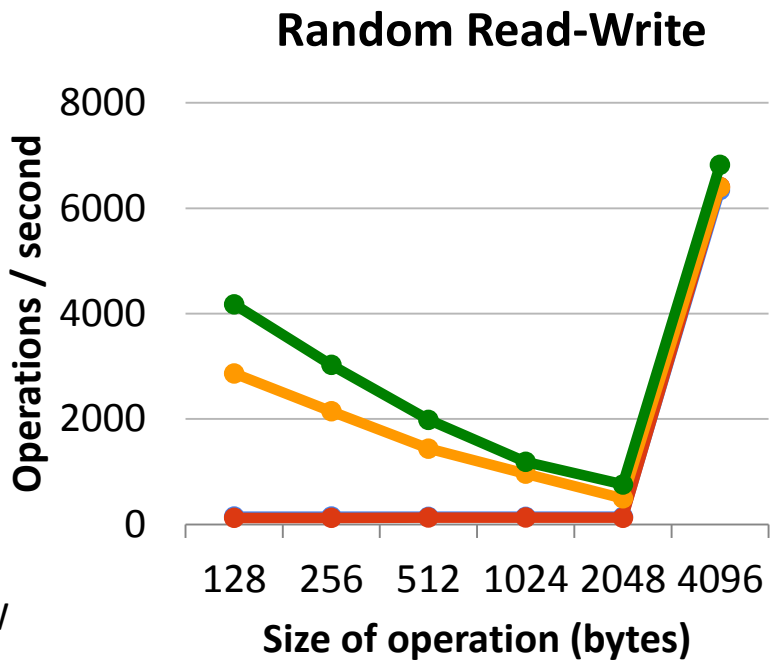
- We found this problem in other OSes (FreeBSD, Xen)
- We implemented non-blocking writes for files in the Linux kernel by modifying the generic virtual file system (VFS) layer

Workloads

- Filebench micro-benchmark
- SPECsfs2008 benchmark
- Mobibench system call trace-replay

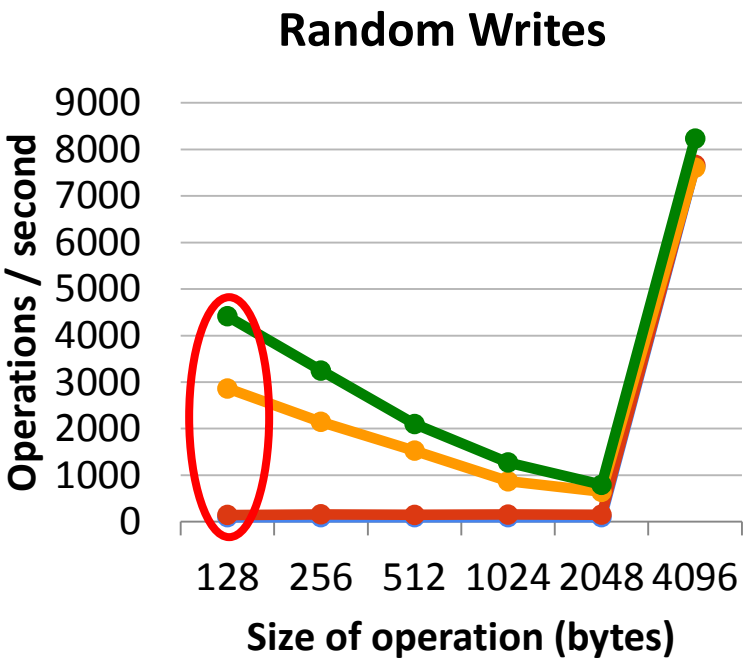
Filebench evaluation: Writes and Read-Writes

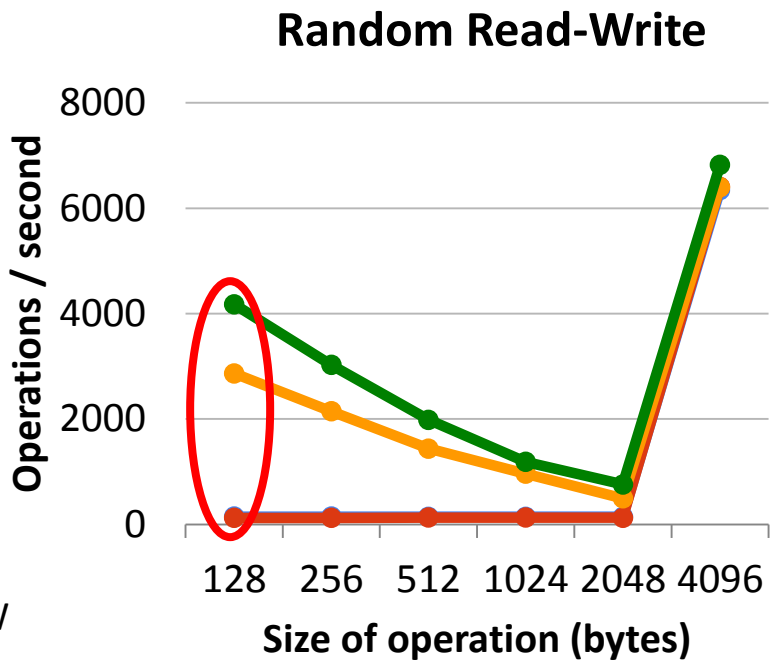




- NBW-Async-FG: 50-60% performance improvement for Random Writes

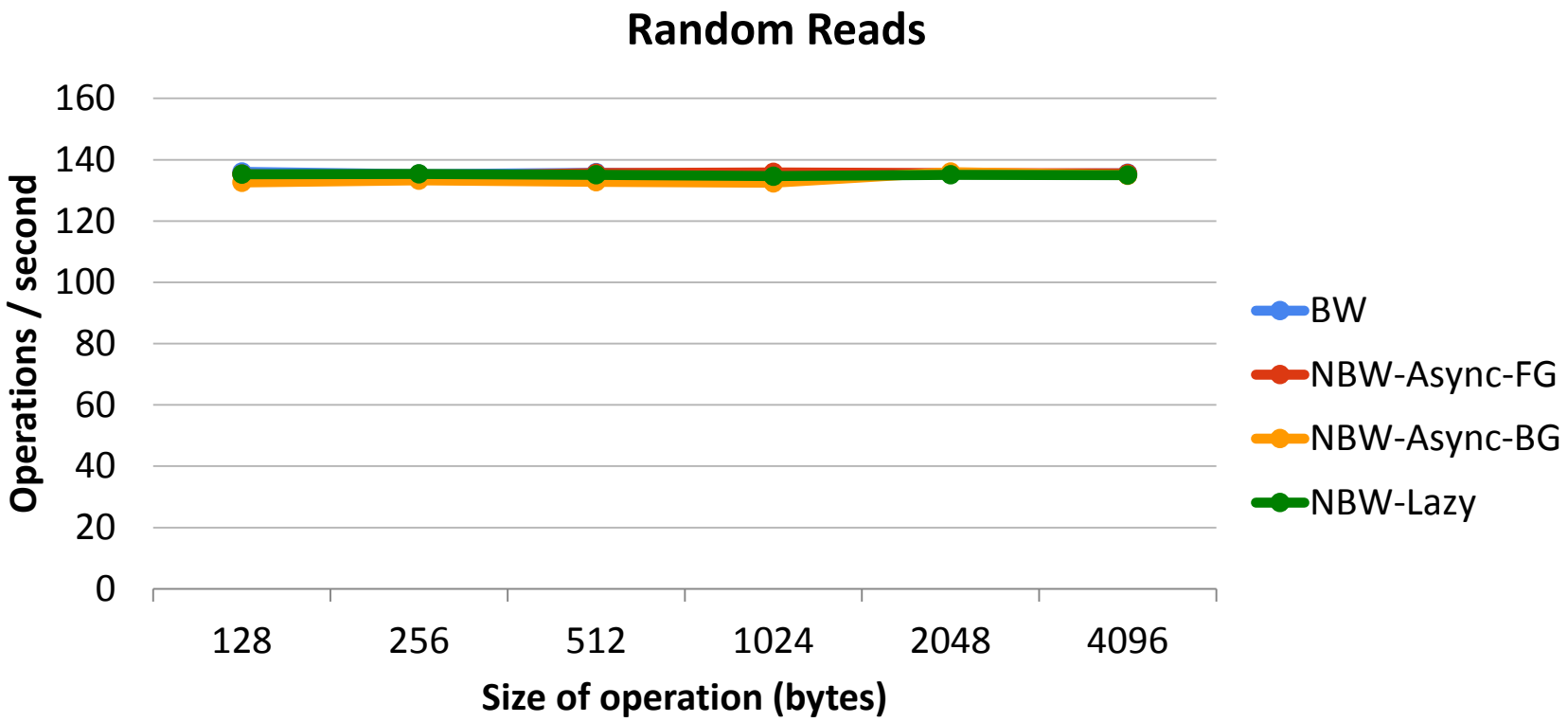
Filebench evaluation: Writes and Read-Writes





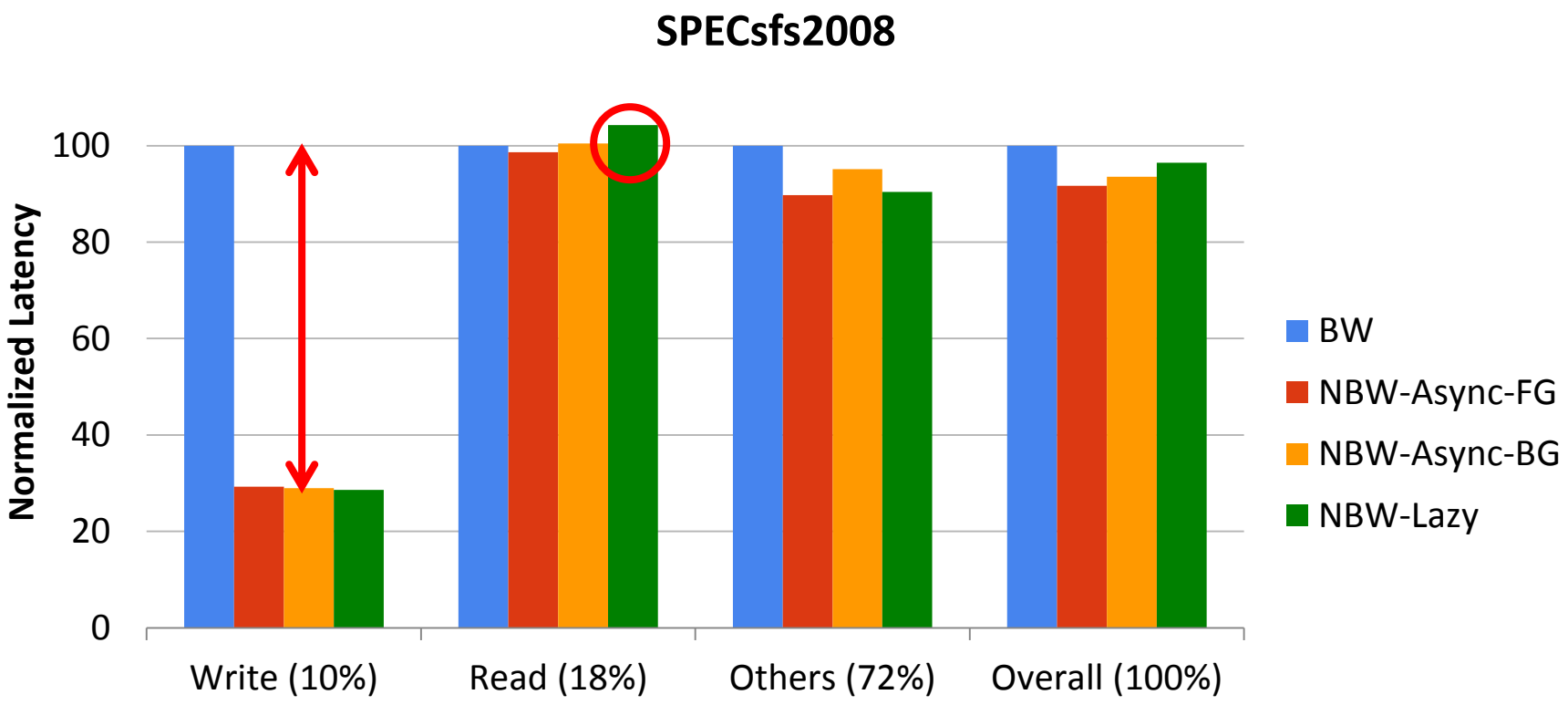
- NBW-Async-FG: 50-60% performance improvement for Random Writes
- NBW-Async-BG: 6.7x-30x (Random Writes) and 3.4x-20x (Random RW)
- NBW-Lazy: Up to 45x performance improvement

Filebench evaluation: Reads



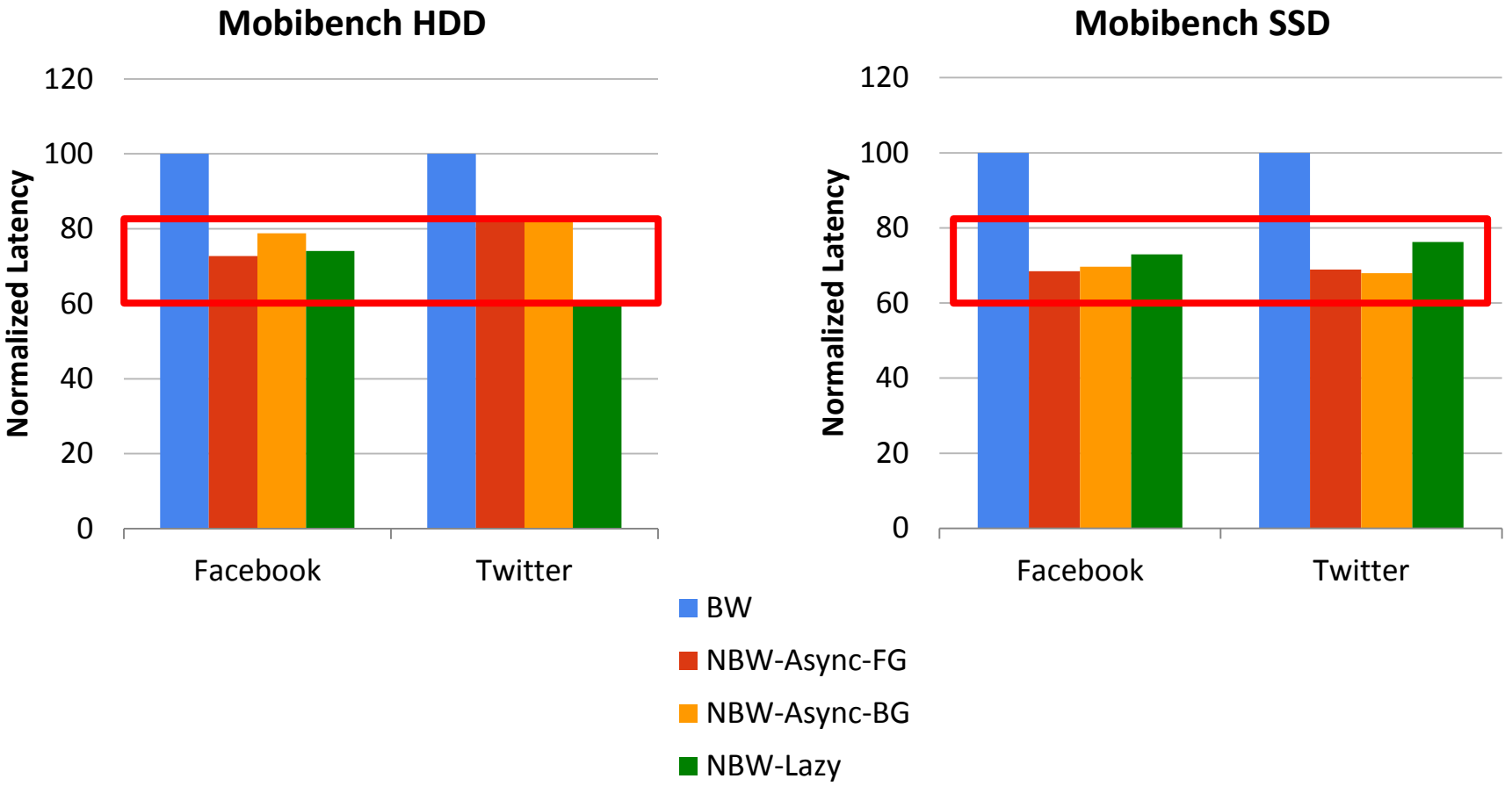
Non-blocking writes do not affect the performance of read-only workloads

SPECsfs2008 evaluation



- Around 70% write latency decrease
- NBW-Lazy read latency slightly affected by delayed fetching
- Overall latency is in direct relation to write and read latency

Mobibench System-call Trace Replay



Average operation latency decreased by 20-40%

Related Work

- Huge body of work on optimizing writes performance
  - O_NONBLOCK flag for OPEN system call
    - If no data can be read or written, returns EAGAIN (or EWOULDBLOCK)
    - Requires application modification
  - Asynchronous I/O Library (POSIX AIO)
    - Multiple threads to perform I/O operations (expensive and scales poorly)
    - Requires application modification

Non-blocking writes do not require application modification

Conclusions

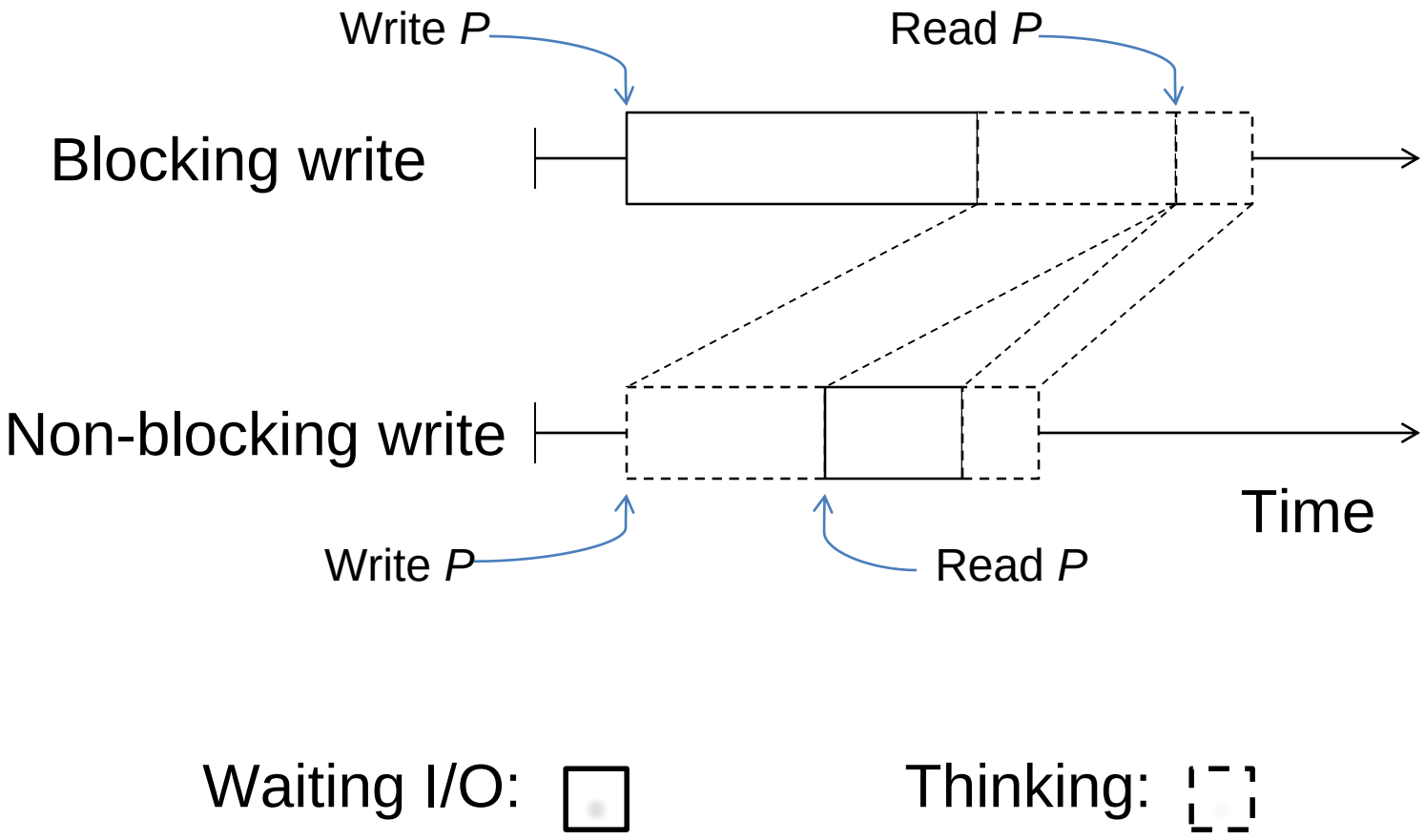
- Operating systems block processes on asynchronous writes in many cases!
- With non-blocking writes, we demonstrate that such blocking is unnecessary and detrimental to performance
- General solution for any kind of modern OS
- Does not require application modification
- Evaluation demonstrated how non-blocking writes effectively reduces write latency across a wide range of workloads

Thank You!

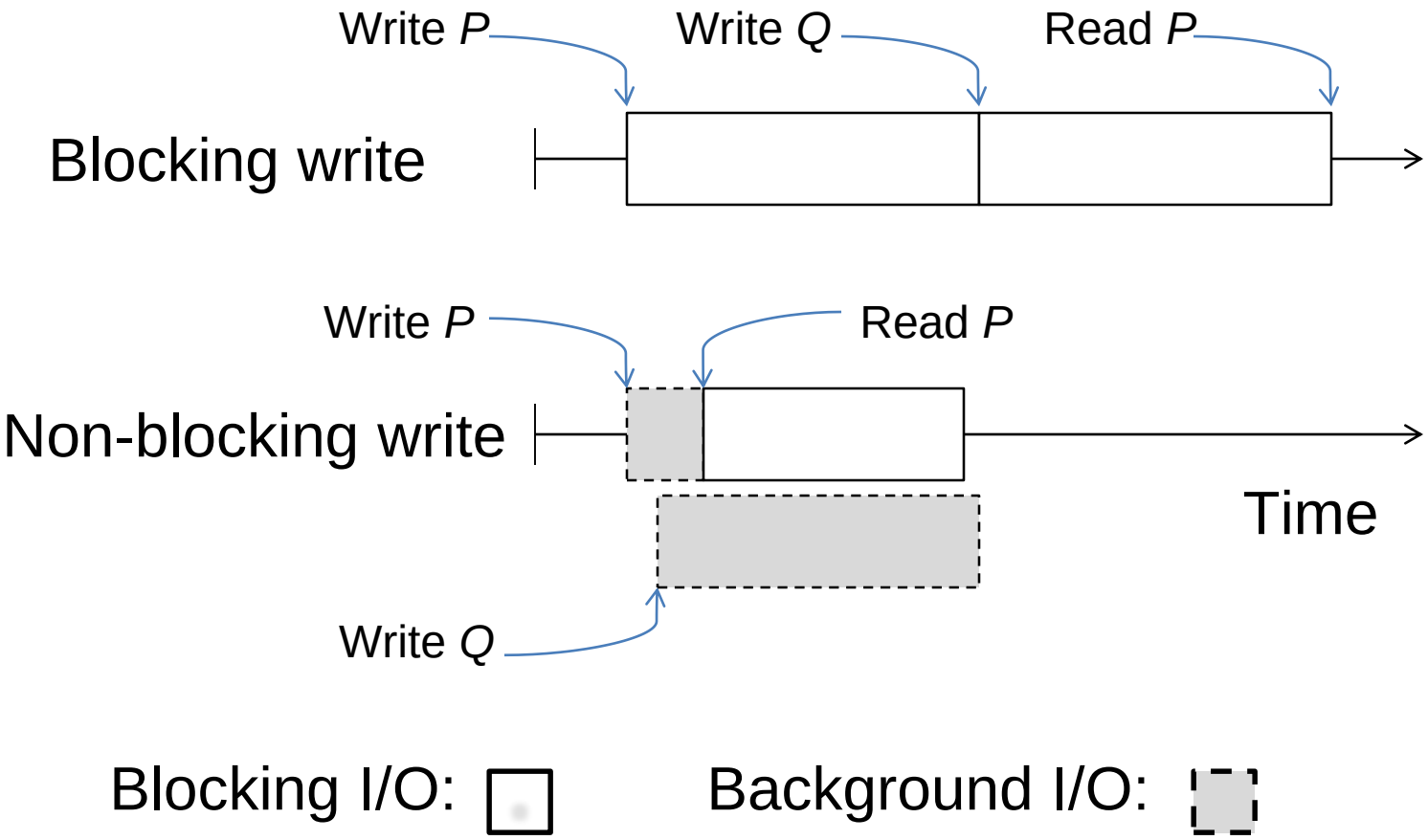
The traces used in this paper are available at

http://sylab-srv.cs.fiu.edu/dokuwiki/doku.php?id=projects:nbw:start

Page Fetch Asynchrony



Page Fetch Parallelism



Motivation: Production workloads

Production system trace descriptions

Workload	Description
ug-filserv	Undergrad NFS/CIFS fileserver
gsf-filesrv	Grad/Staff/Faculty NFS/CIFS fileserver
moodle	Web & DB server for department CMS
backup	Nightly backups of department servers
usr1	Researcher 1 desktop
usr2	Researcher 2 desktop
facebook	Mobibench facebook trace
twitter	Mobibench twitter trace

Highlights On Correctness

- Ordering of page updates
- Handling of disk errors
- Journaling File Systems
- OS-initiated page accesses
- Page persistence and syncs
- Handling read-write dependencies
- Multi-core and kernel preemption