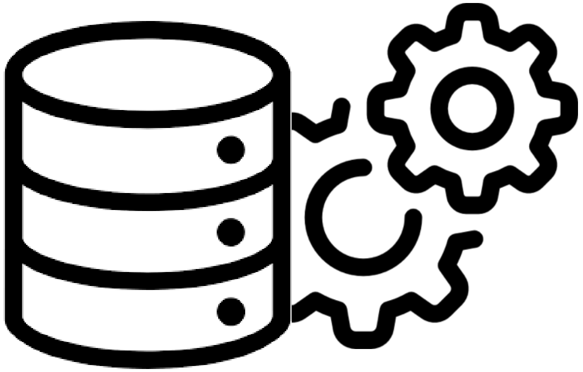
Asymmetry/Concurrency-Aware Bufferpool Manager for Modern Storage Devices

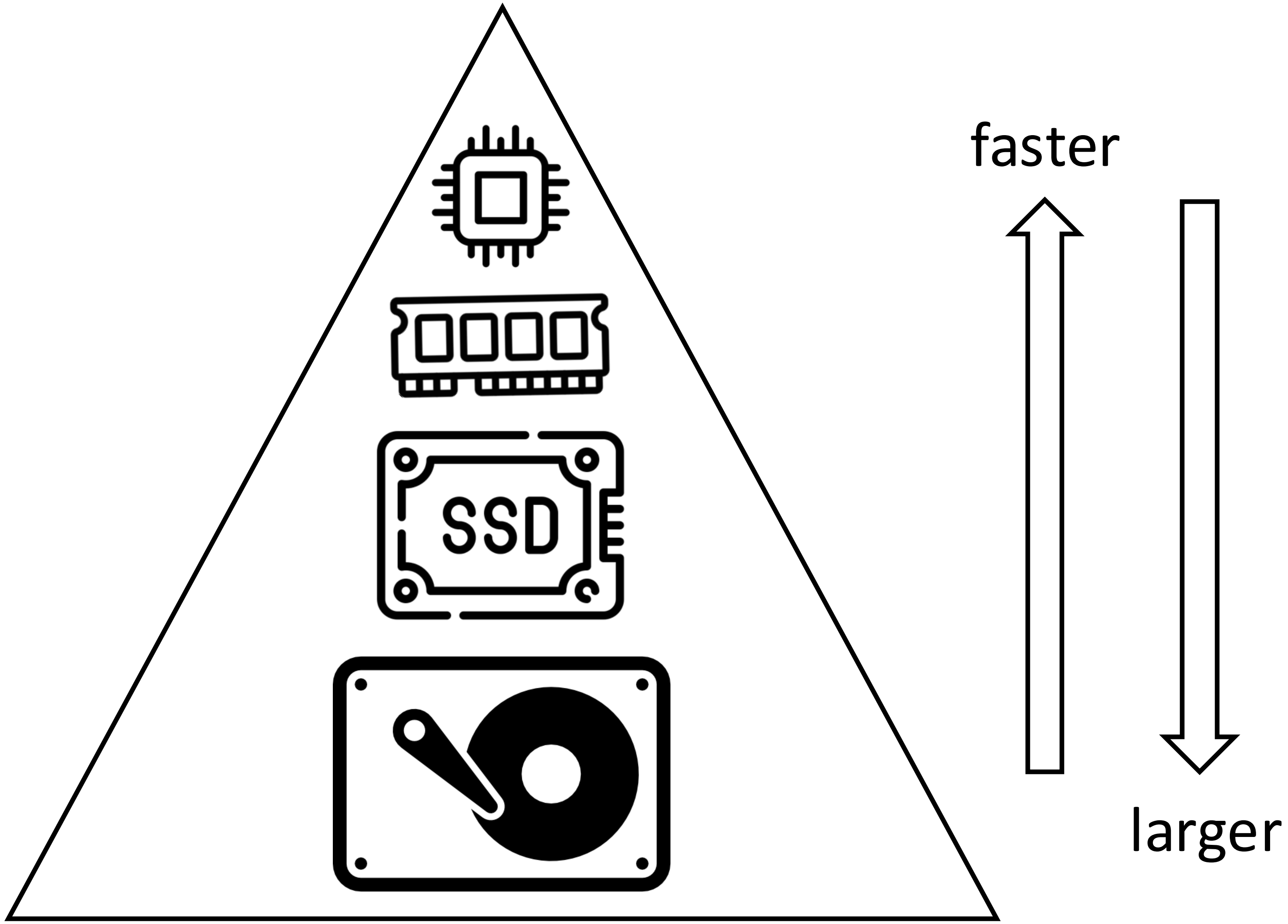
Tarikul Islam Papon
papon@bu.edu

Manos Athanassoulis
mathan@bu.edu

Data Systems & Hardware

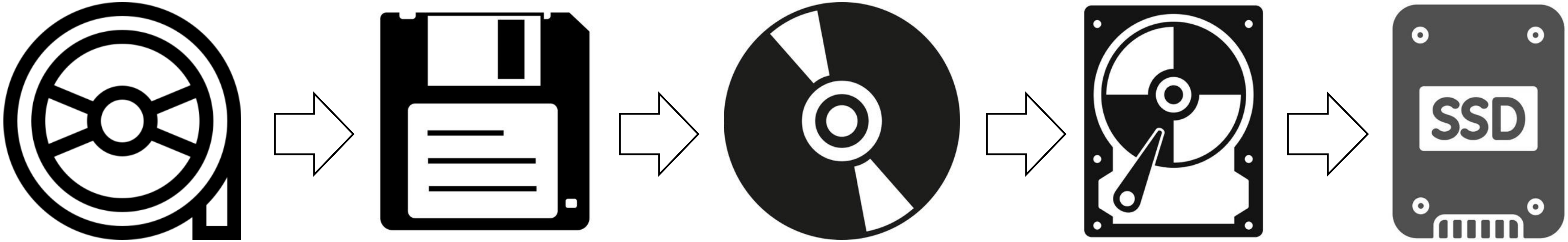


Data Systems



Memory Hierarchy

Evolution of Storage Devices



Tape → Floppy → CD → HDD → SSD

Hard Disk Drives

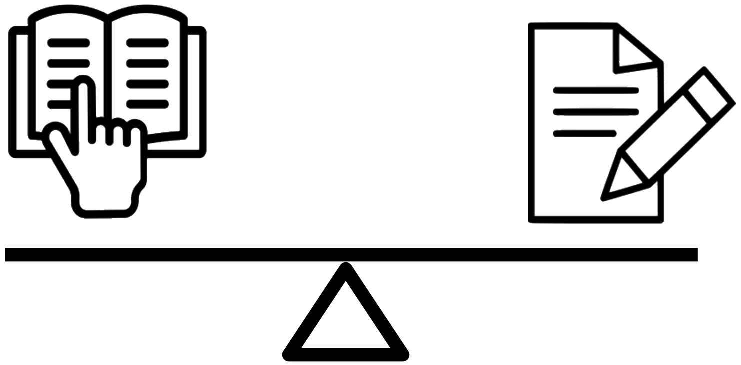
mechanical device

slow random access

one block at a time

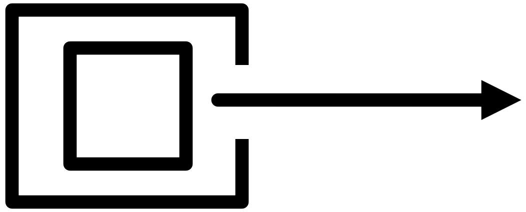
write latency ≈ read latency

Hard Disk Drives



Symmetric cost for **Read & Write** to disk

✓



One I/O at a time

✓

“Tape is Dead. Disk is Tape.

Flash is Disk.”

- Jim Gray

“Tape is Dead. Disk is Tape. Flash is Disk.”

- Jim Gray

Device	Size	Seq B/W	Time to read
HDD 1980	100 MB	1.2 MB/s	~ 1 min
HDD 2022	4 TB	125 MB/s	~ 9 hours

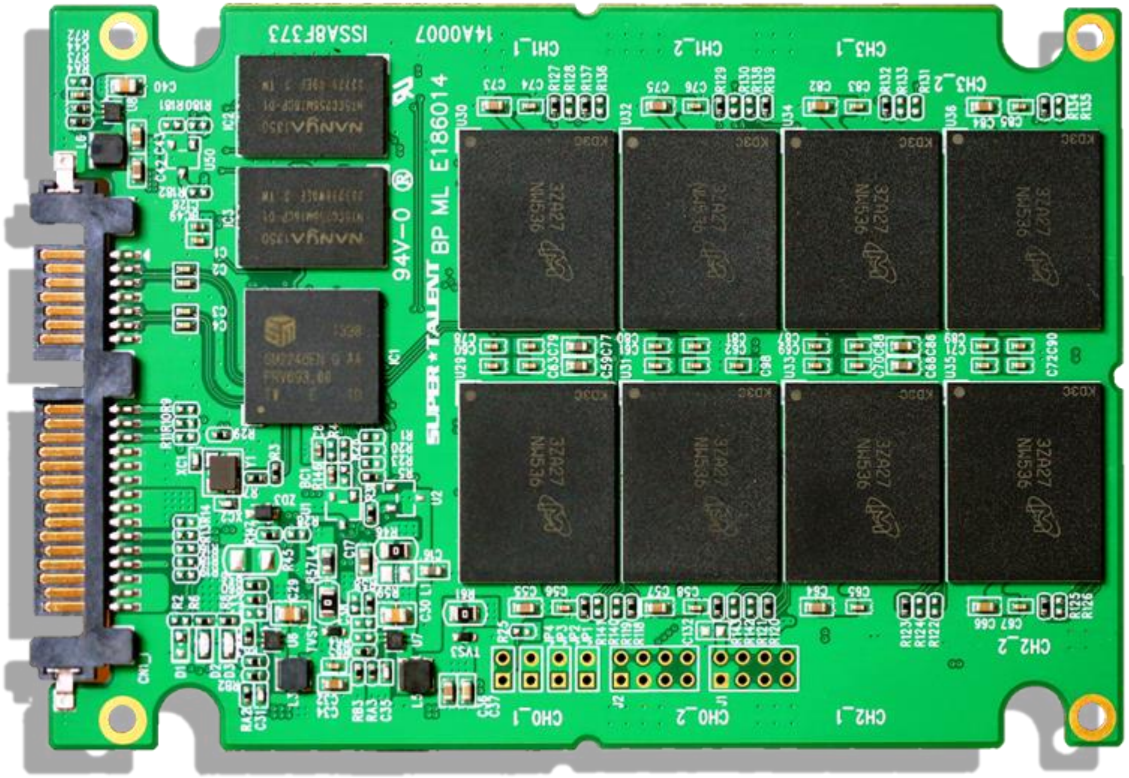
"Tape is Dead. Disk is Tape. Flash is Disk."

- Jim Gray

Device	Size	Seq B/W	Time to read
HDD 1980	100 MB	1.2 MB/s	**~ 1 min**
HDD 2022	4 TB	125 MB/s	**~ 9 hours**

HDDs are moving deeper in the memory hierarchy

Solid State Drives



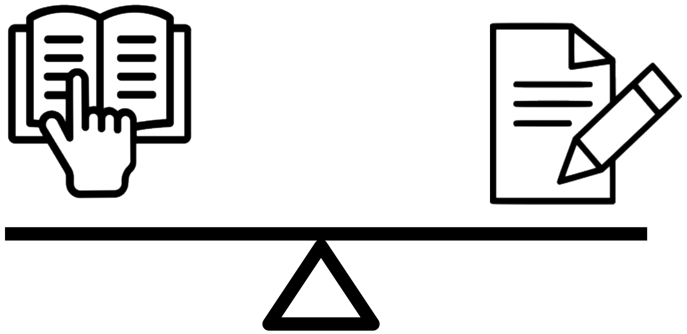
electronic device

fast random access

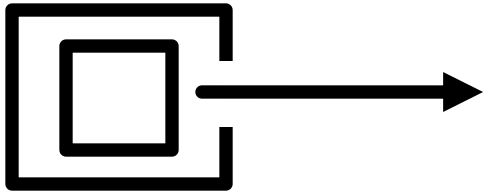
concurrent I/Os

write latency > read latency

HDD

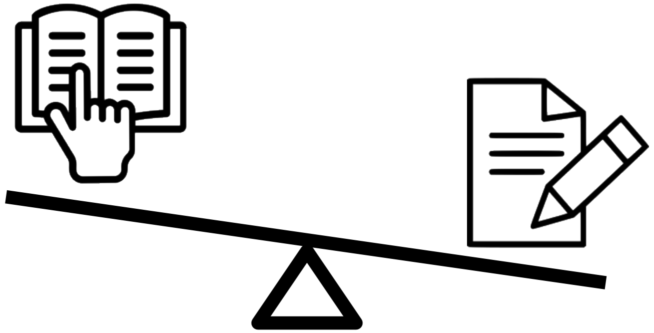


Symmetric cost for Read & Write

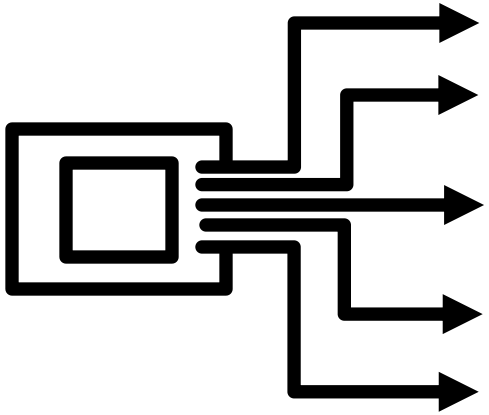


One I/O at a time

SSD

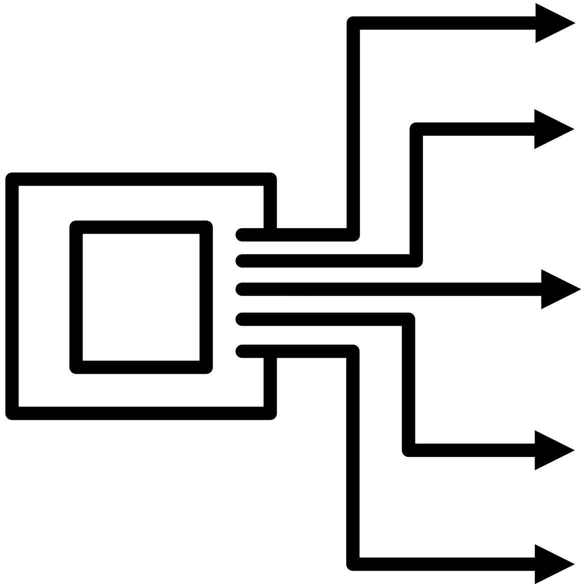


Read/Write Asymmetry (α)

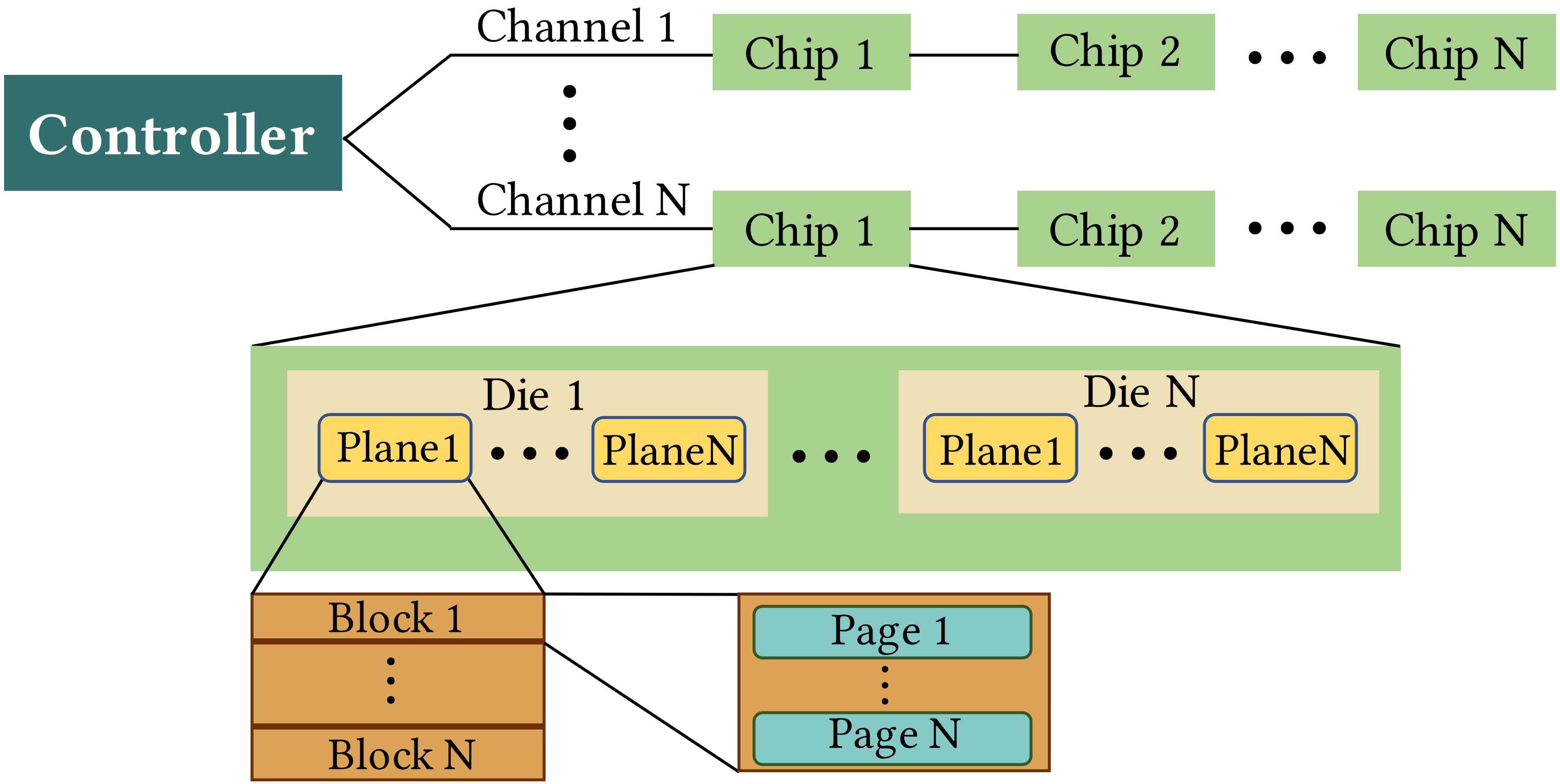


Concurrency (k)

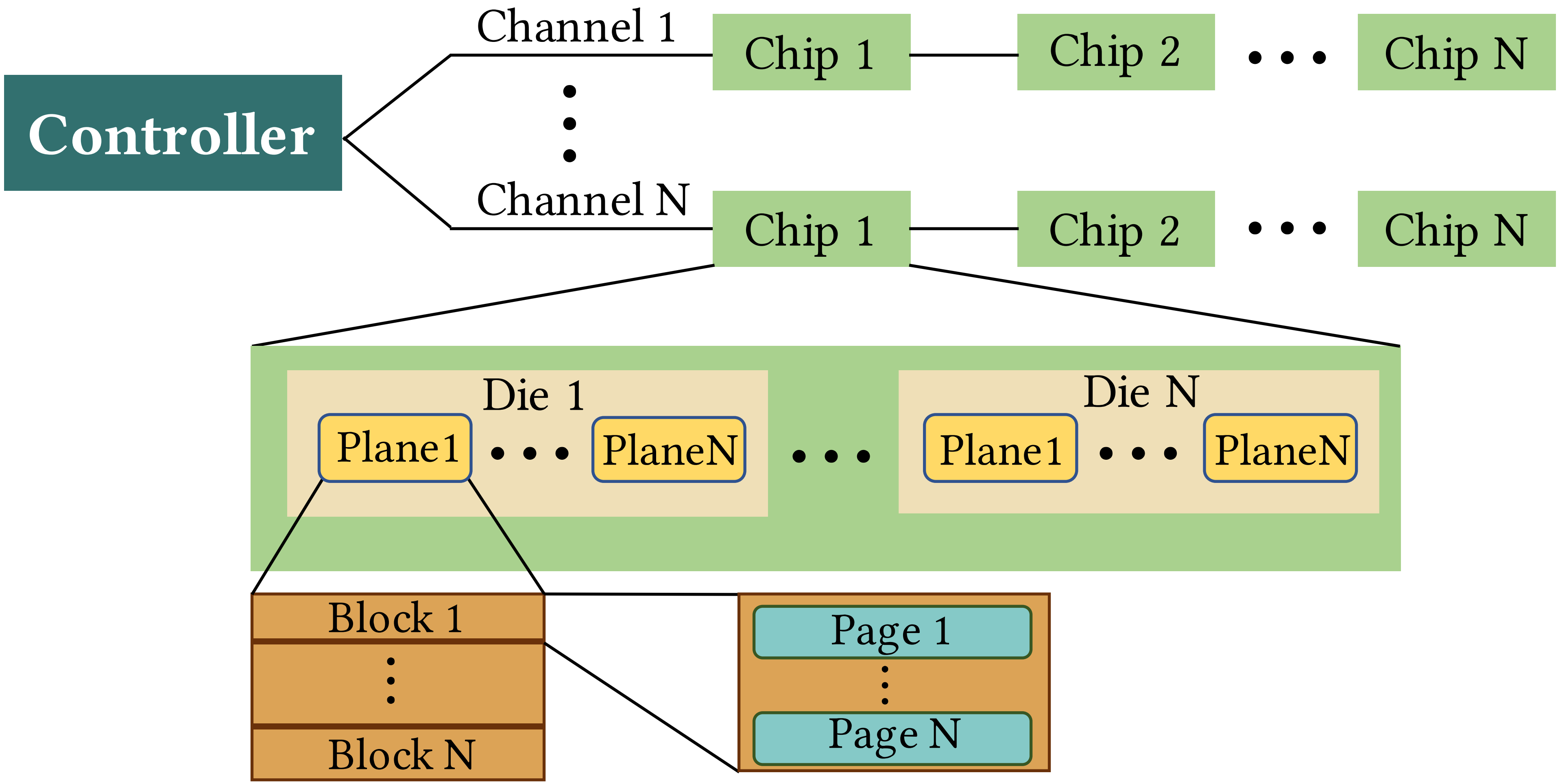
Concurrency



Internals of an SSD

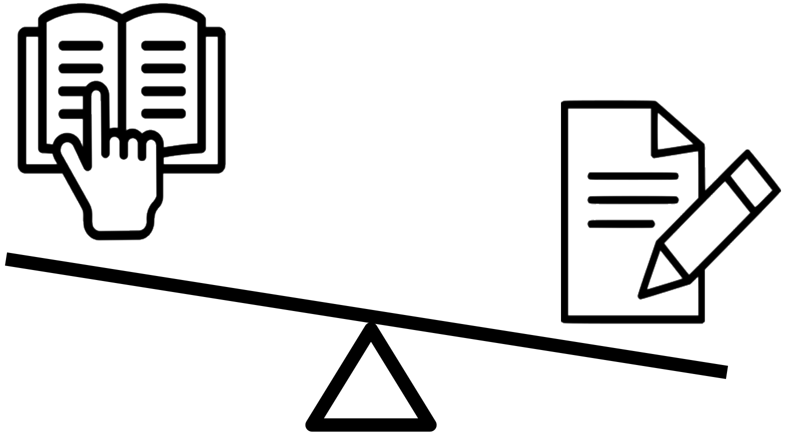


Internals of an SSD



Parallelism at different levels (channel, chip, die, plane block, page)

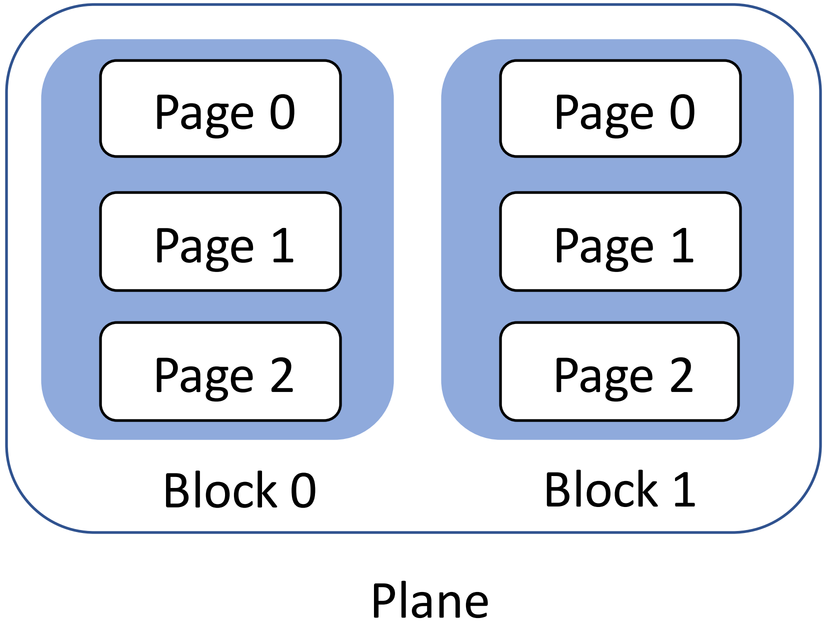
Read/Write Asymmetry



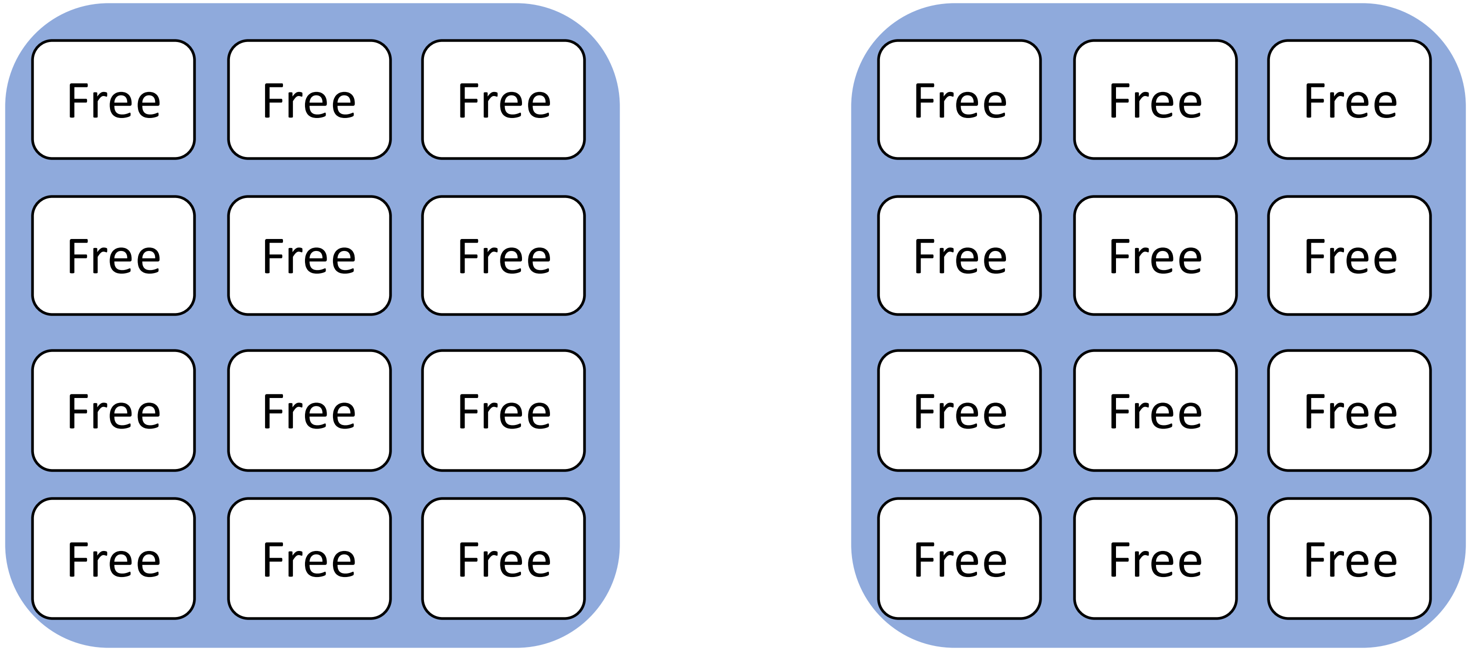
Writes in SSD

Out-of-place updates cause invalidation

"Erase before write" approach



Writes in SSD



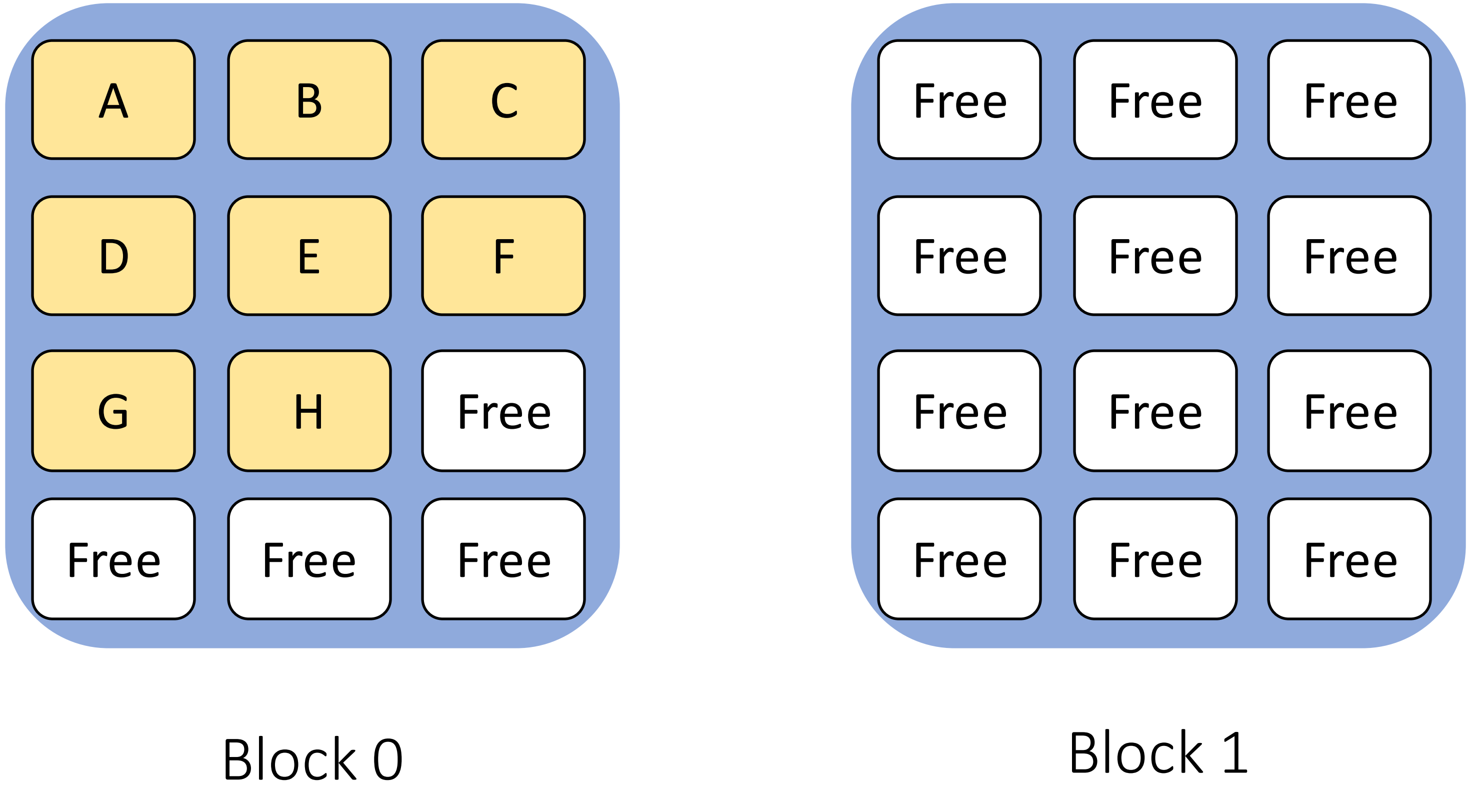
Free	Free	Free
Free	Free	Free
Free	Free	Free
Free	Free	Free

Block 0

Free	Free	Free
Free	Free	Free
Free	Free	Free
Free	Free	Free

Block 1

Writes in SSD



A	B	C
D	E	F
G	H	Free
Free	Free	Free

Block 0

Free	Free	Free
Free	Free	Free
Free	Free	Free
Free	Free	Free

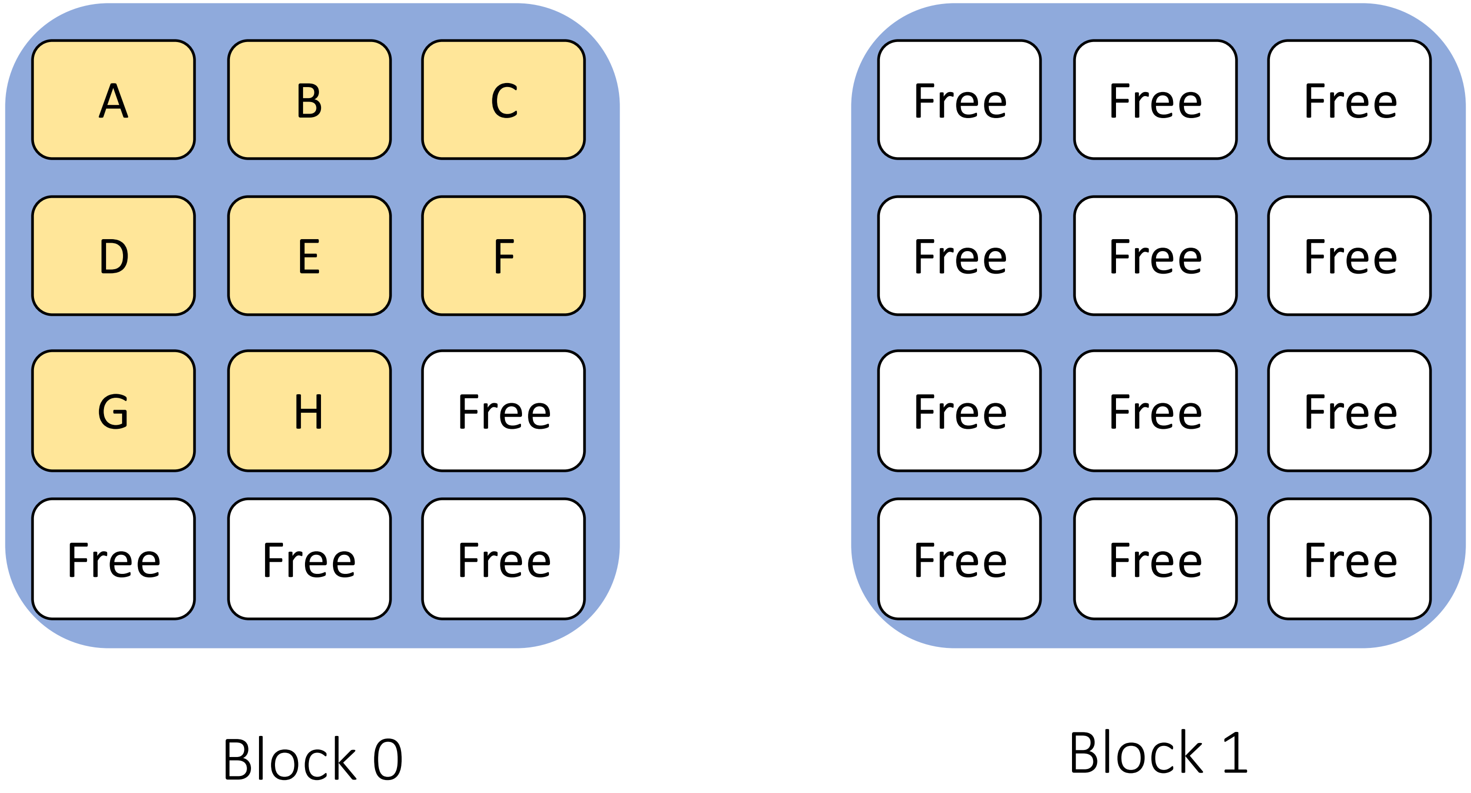
Block 1

Writing in a free page isn’t costly!

Writes in SSD

Update

A, B, C, D



A	B	C
D	E	F
G	H	Free
Free	Free	Free

Block 0

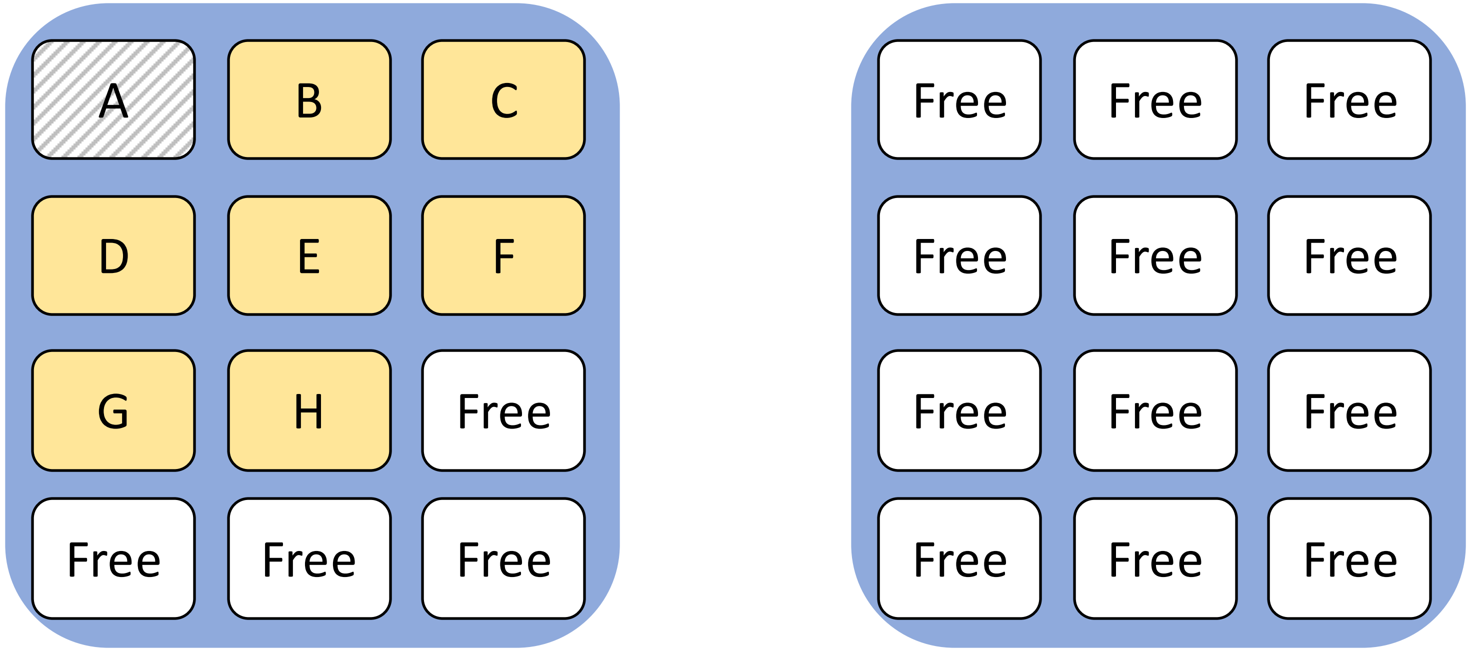
Free	Free	Free
Free	Free	Free
Free	Free	Free
Free	Free	Free

Block 1

Writes in SSD

Update

A, B, C, D



A	B	C
D	E	F
G	H	Free
Free	Free	Free

Block 0

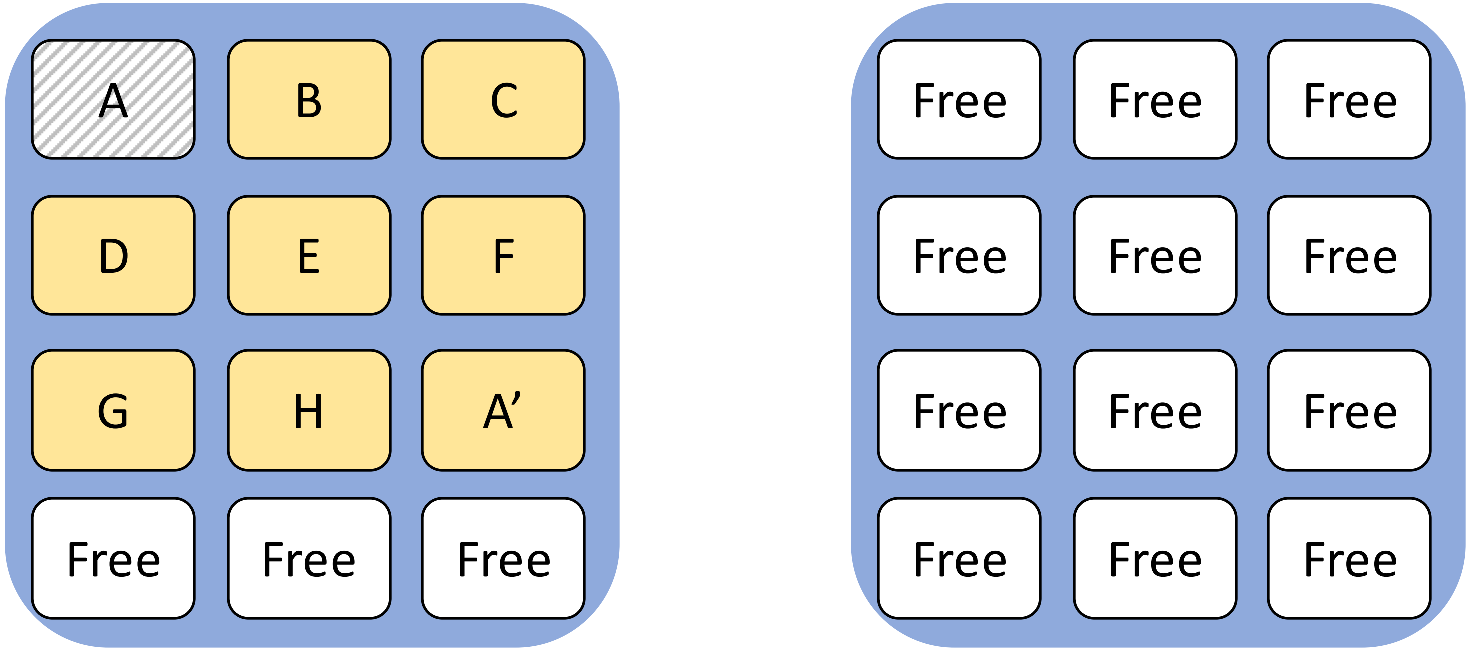
Free	Free	Free
Free	Free	Free
Free	Free	Free
Free	Free	Free

Block 1

Writes in SSD

Update

A, B, C, D



A	B	C
D	E	F
G	H	A’
Free	Free	Free

Block 0

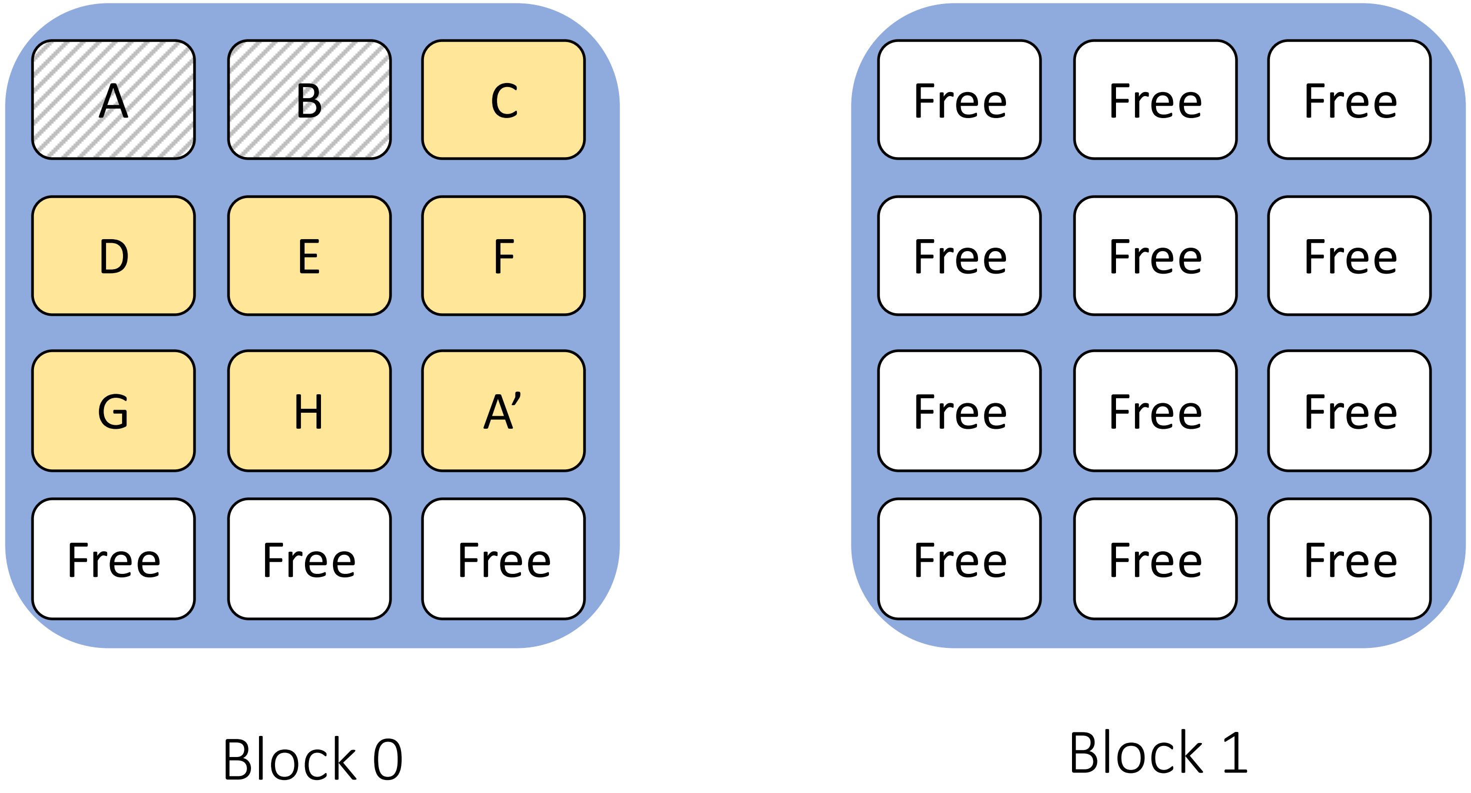
Free	Free	Free
Free	Free	Free
Free	Free	Free
Free	Free	Free

Block 1

Writes in SSD

Update

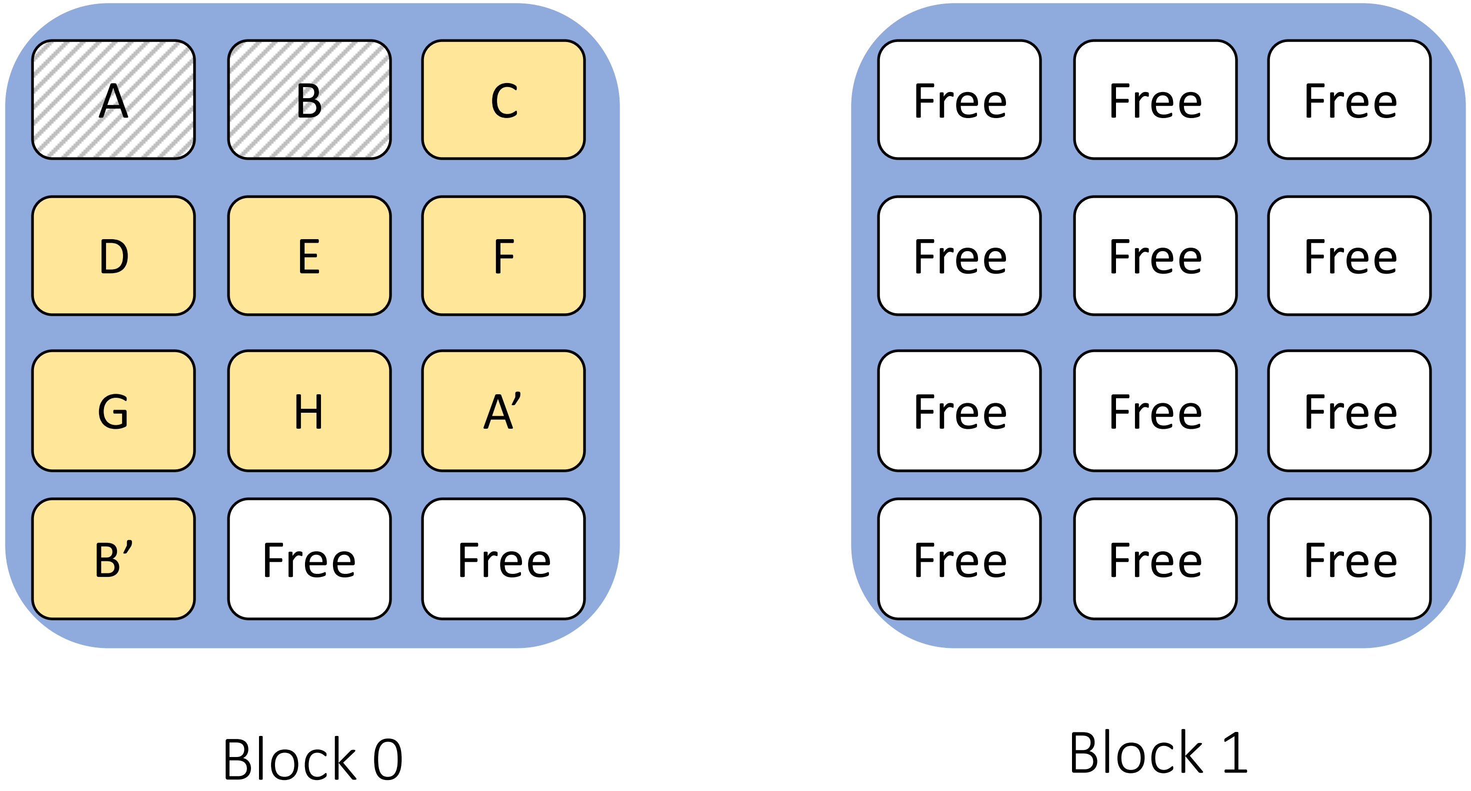
A, **B**, C, D



Writes in SSD

Update

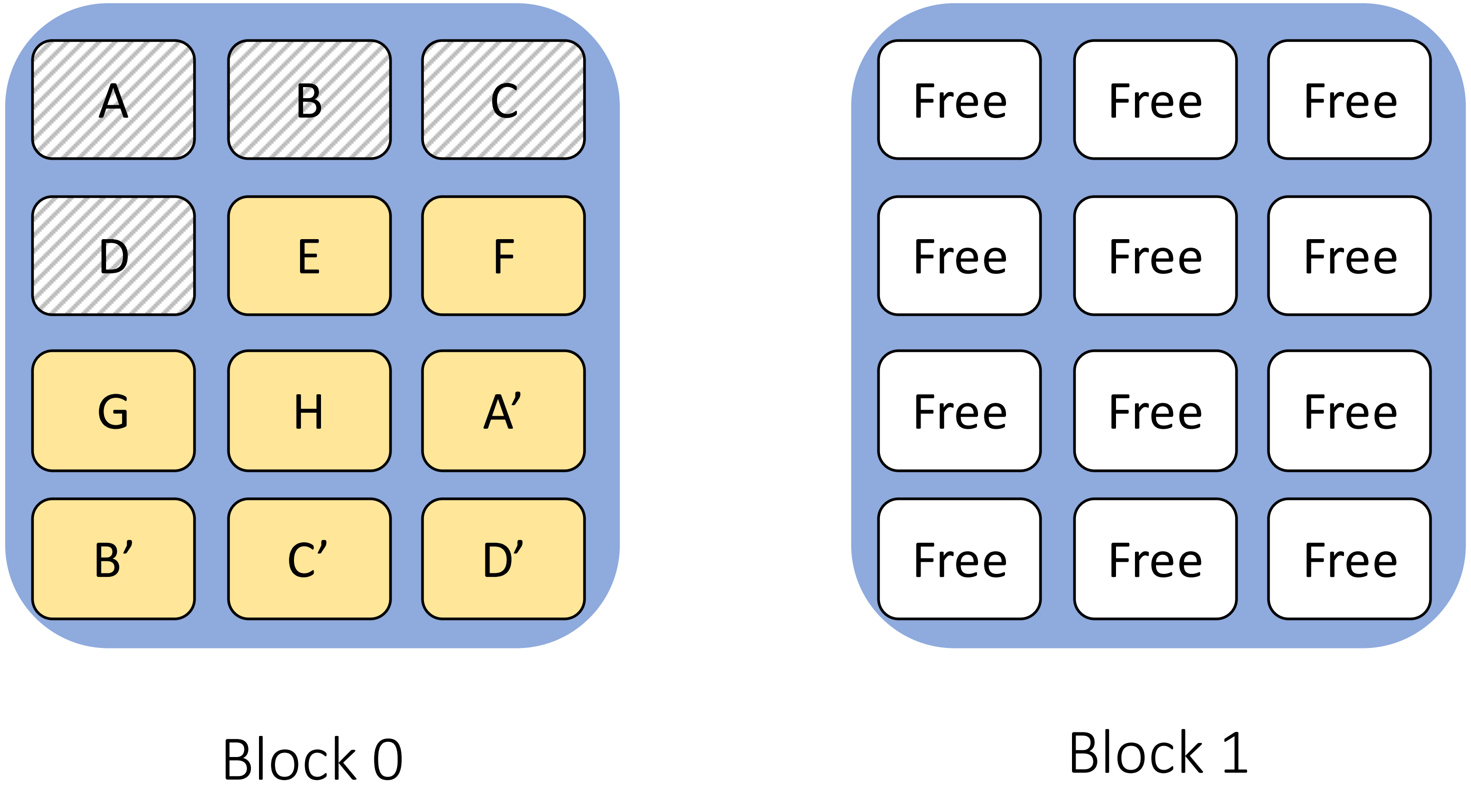
A, **B**, C, D



Writes in SSD

Update

A, B, C, D



A	B	C
D	E	F
G	H	A’
B’	C’	D’

Block 0

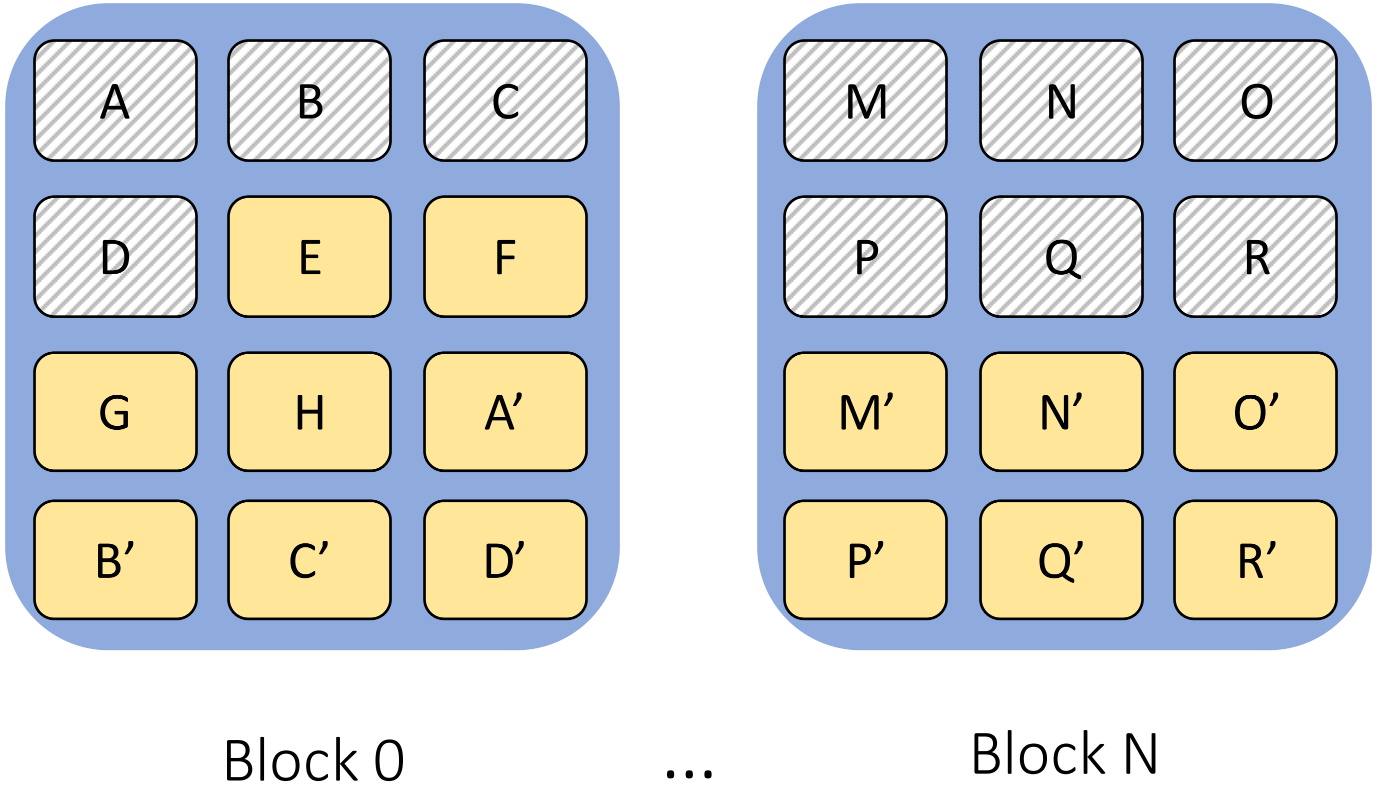
Free	Free	Free
Free	Free	Free
Free	Free	Free
Free	Free	Free

Block 1

Not all updates are costly!

Writes in SSD

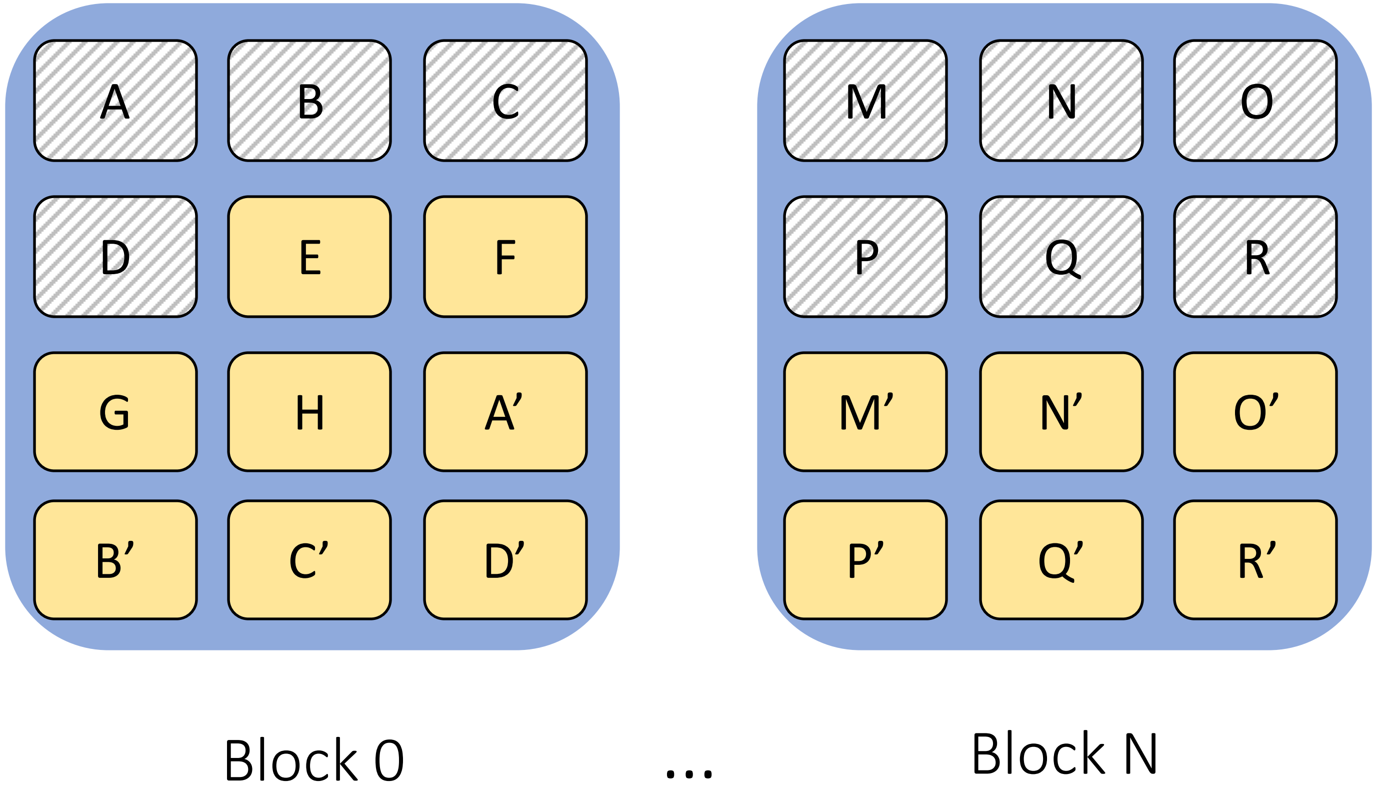
What if there is no space?



Writes in SSD

What if there is no space?

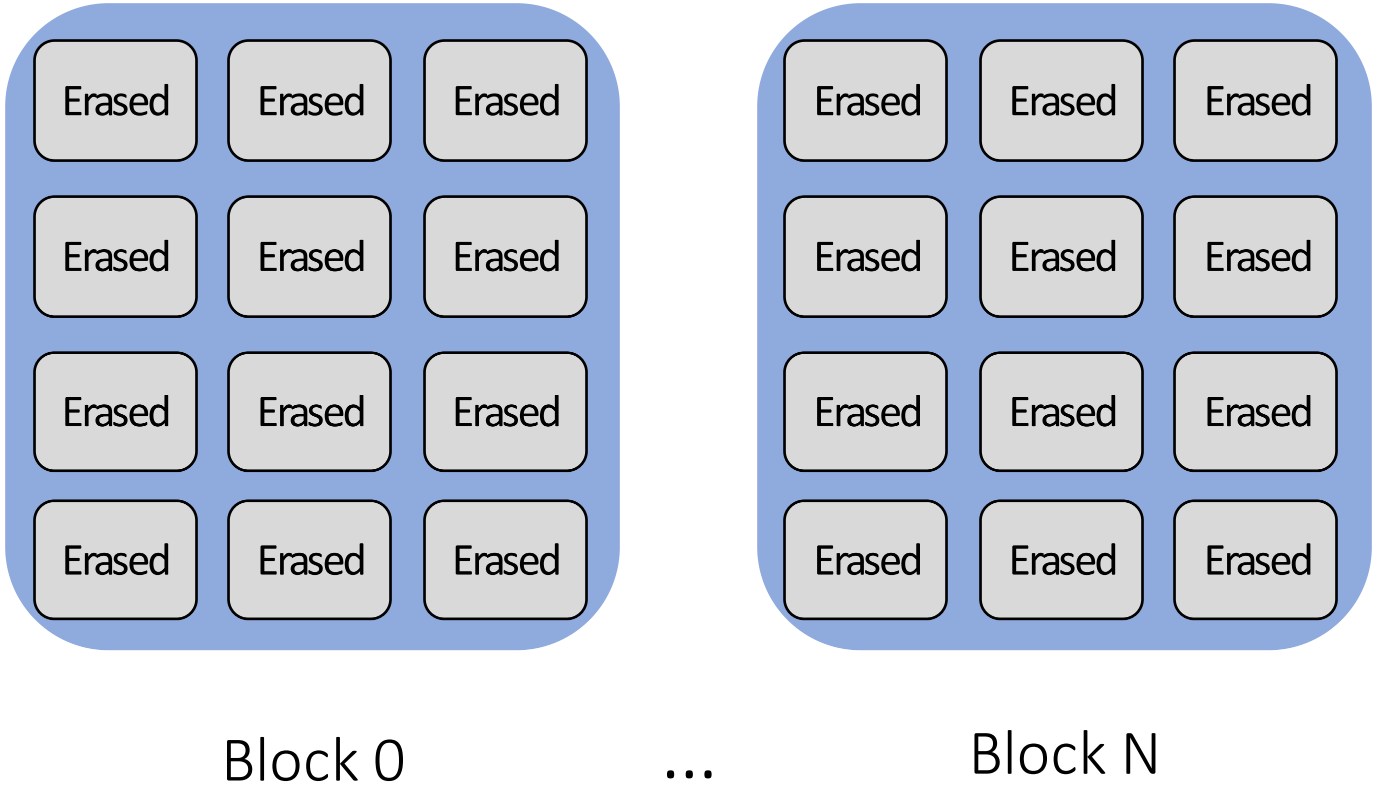
Garbage Collection!



Writes in SSD

What if there is no space?

Garbage Collection!



Block 0

Erased	Erased	Erased
Erased	Erased	Erased
Erased	Erased	Erased
Erased	Erased	Erased

…

Block N

Erased	Erased	Erased
Erased	Erased	Erased
Erased	Erased	Erased
Erased	Erased	Erased

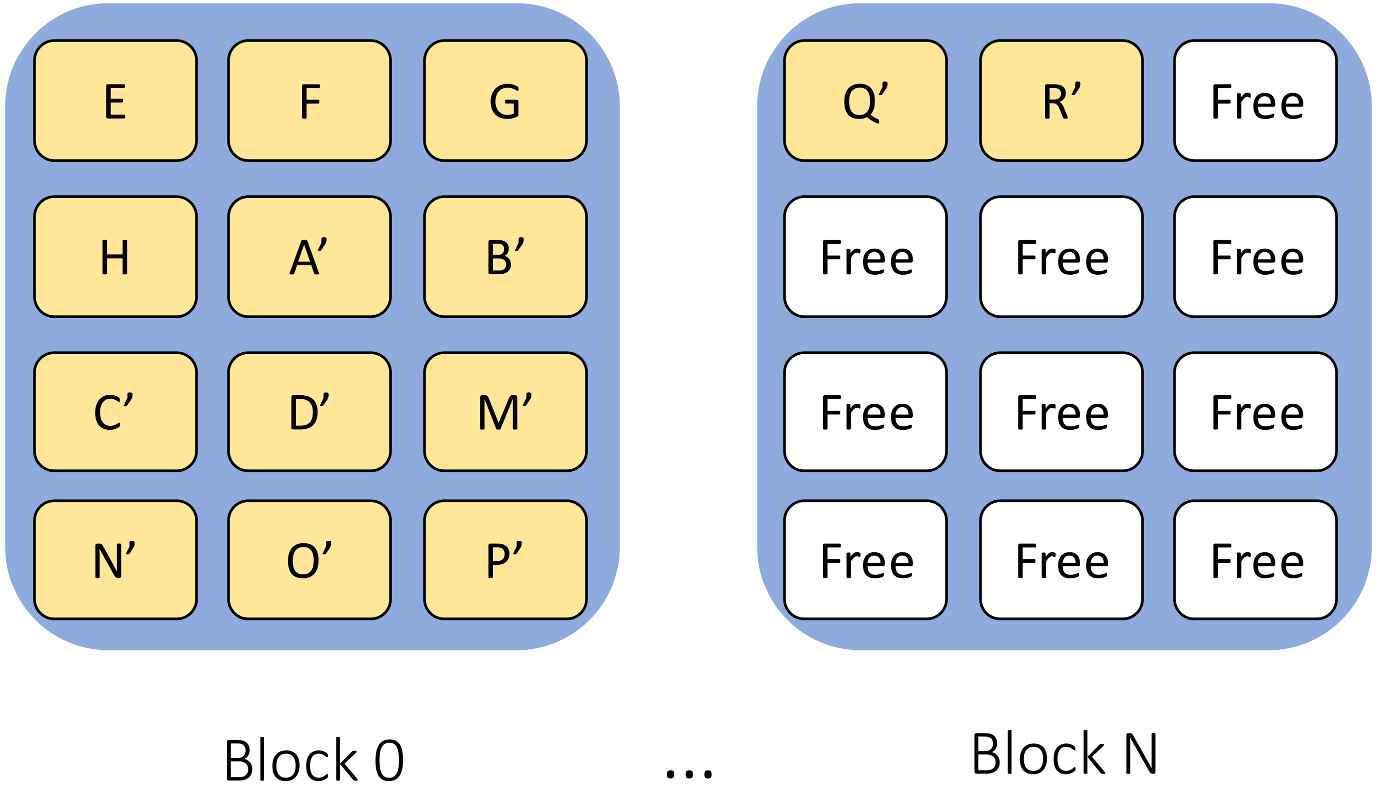
Valid pages:

E	F	G	H	A’	B’	C’	D’	M’	N’	O’	P’	Q’	R’

Writes in SSD

What if there is no space?

Garbage Collection!



E	F	G
H	A’	B’
C’	D’	M’
N’	O’	P’

Block 0

…

Q’	R’	Free
Free	Free	Free
Free	Free	Free
Free	Free	Free

Block N

Higher average update cost (due to GC) → ***Read/Write asymmetry***

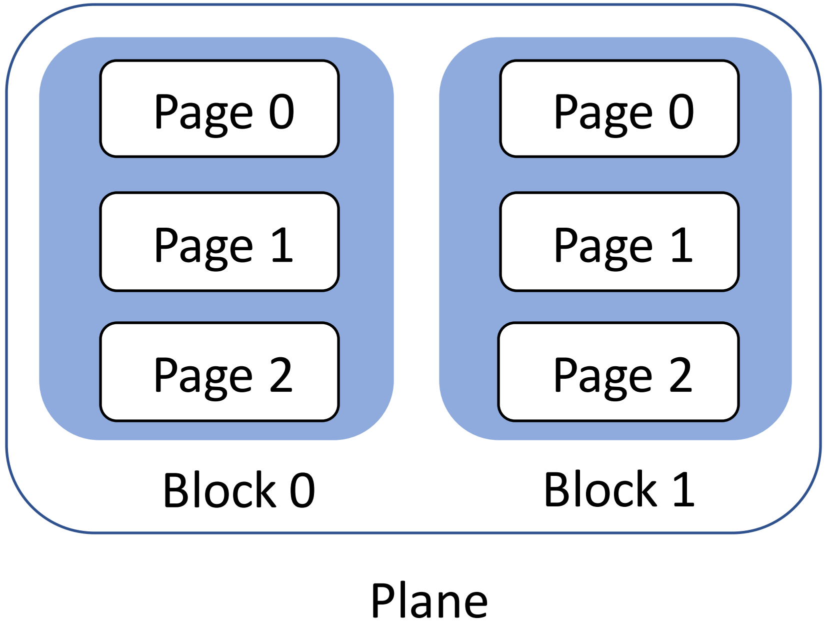
Read/Write Asymmetry

Out-of-place updates cause invalidation

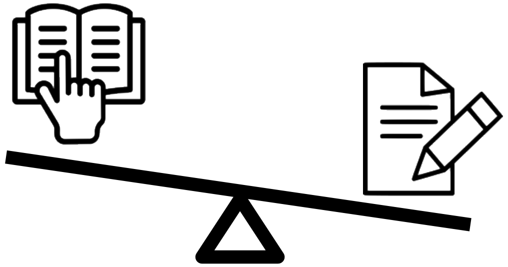
“*Erase before write*” approach

Garbage Collection

Larger erase granularity



All these results in higher amortized write cost



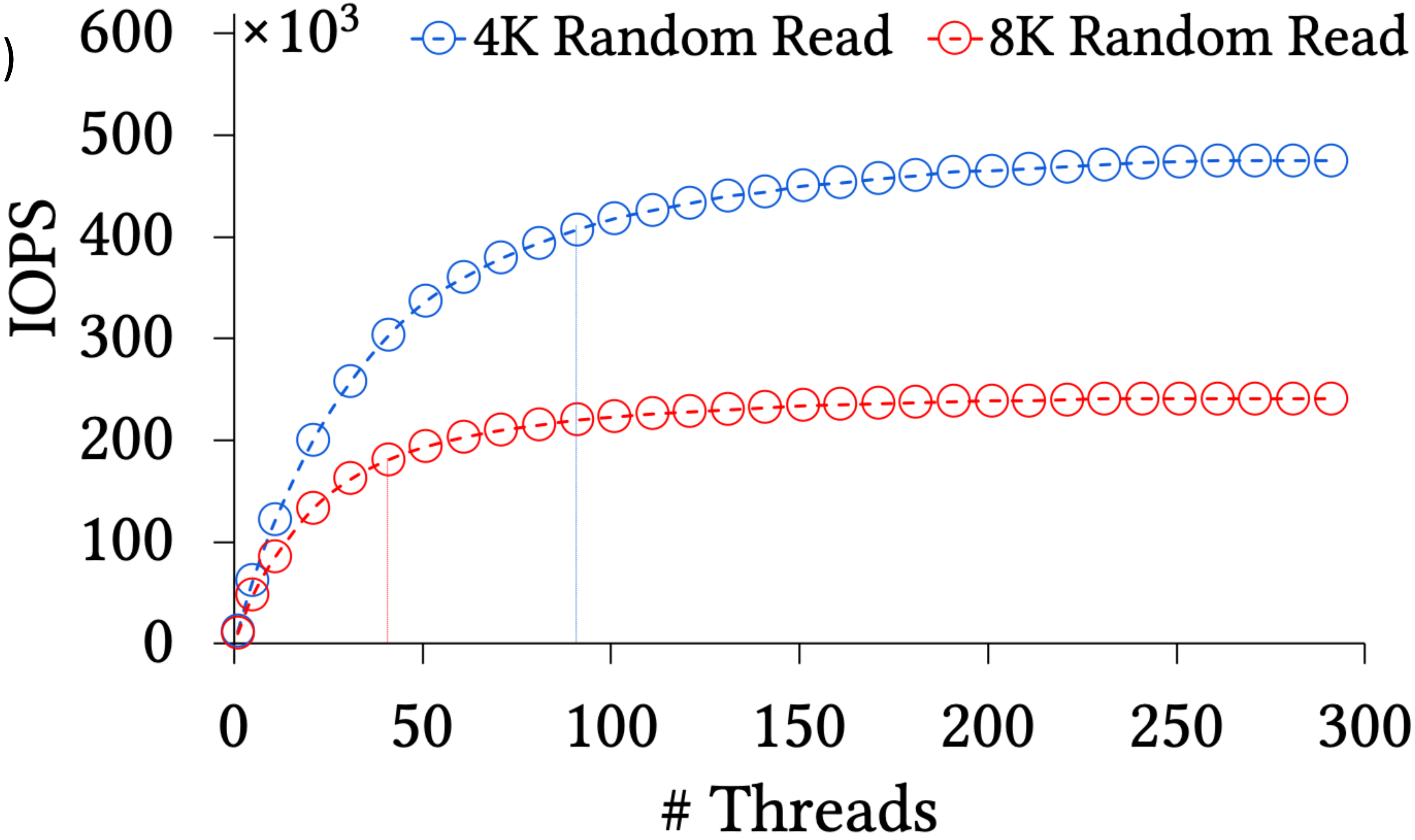
Read/Write Asymmetry - Example

Device	Advertised Rand Read IOPS	Advertised Rand Write IOPS	Advertised Asymmetry
PCIe D5-P4320	427k	36k	11.9
PCIe DC-P4500	626k	51k	12.3
PCIe P4510	465k	145k	3.2
SATA D3-S4610	92k	28k	3.3
Optane P4800X	550k	500k	1.1

Quantifying Asymmetry & Concurrency

Device

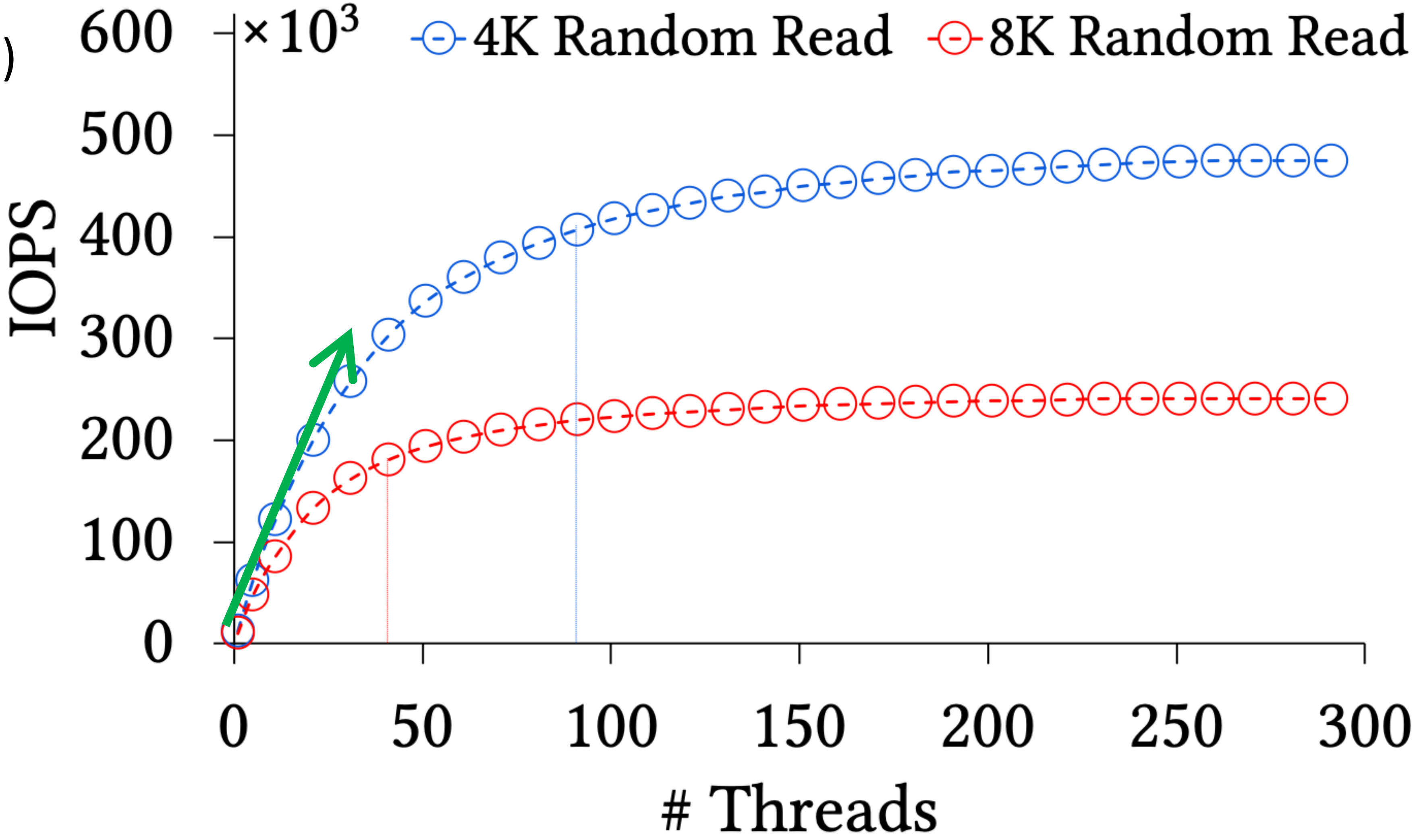
PCIe SSD - P4510 (1TB)



Quantifying Asymmetry & Concurrency

Device

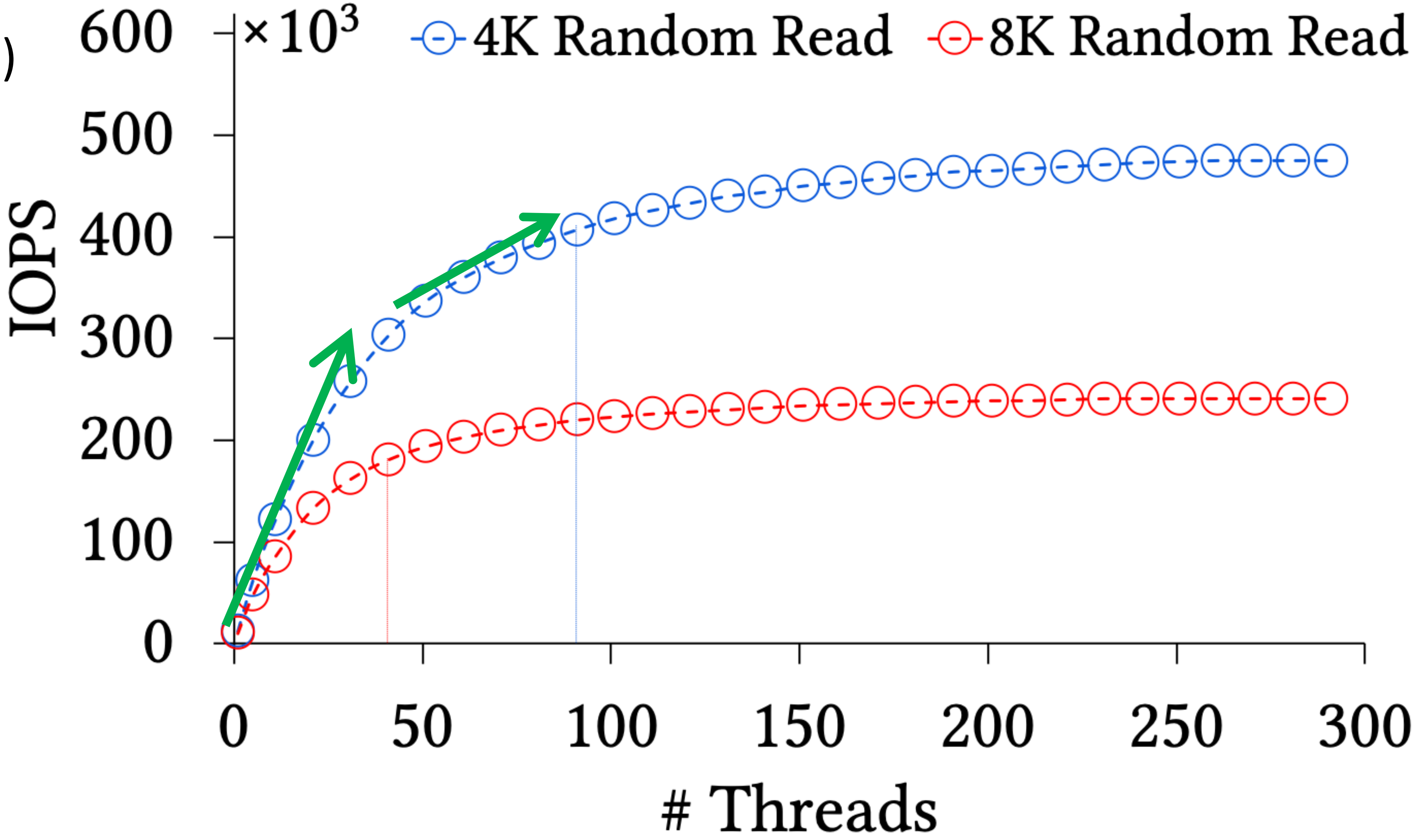
PCIe SSD - P4510 (1TB)



Quantifying Asymmetry & Concurrency

Device

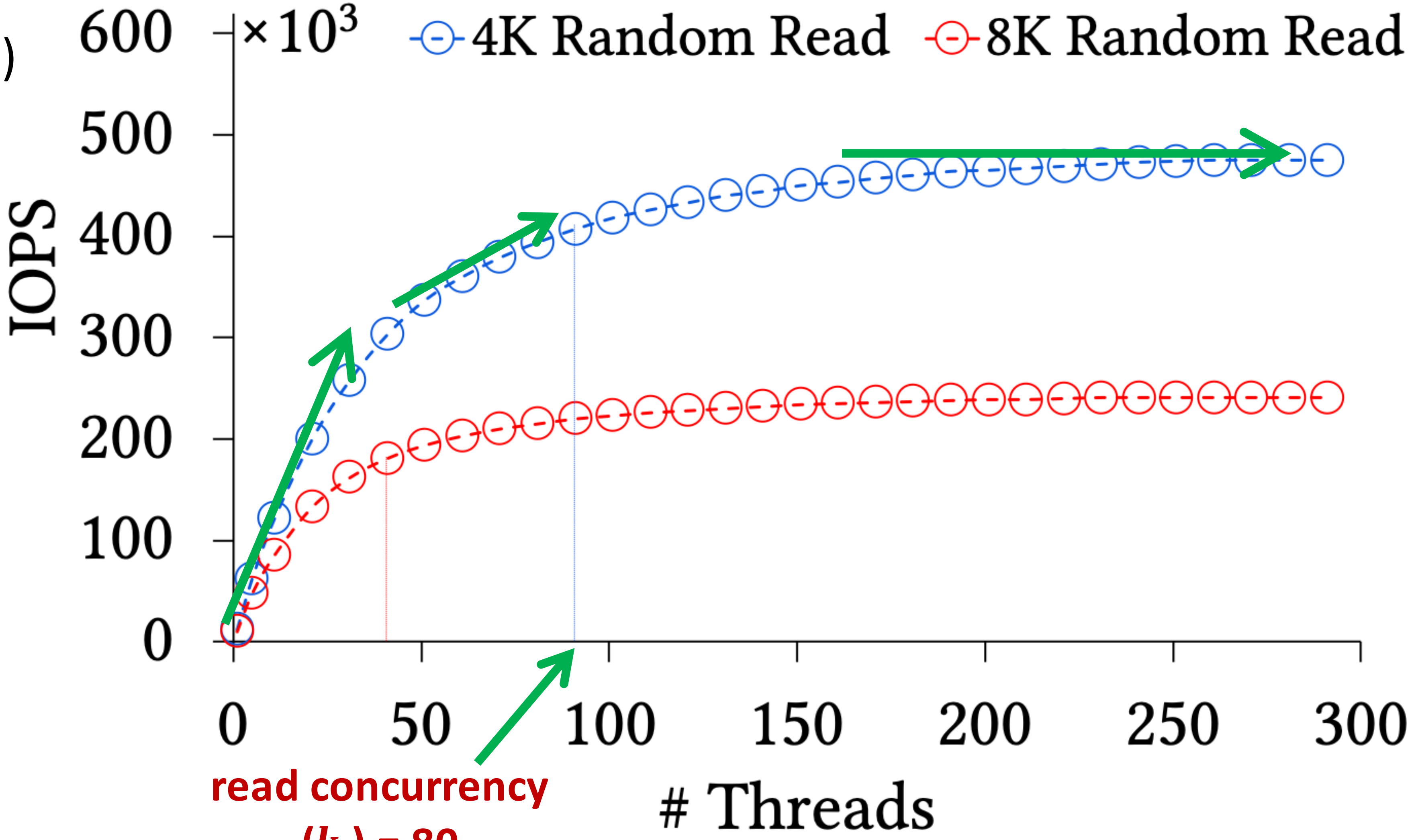
PCIe SSD - P4510 (1TB)



Quantifying Asymmetry & Concurrency

Device

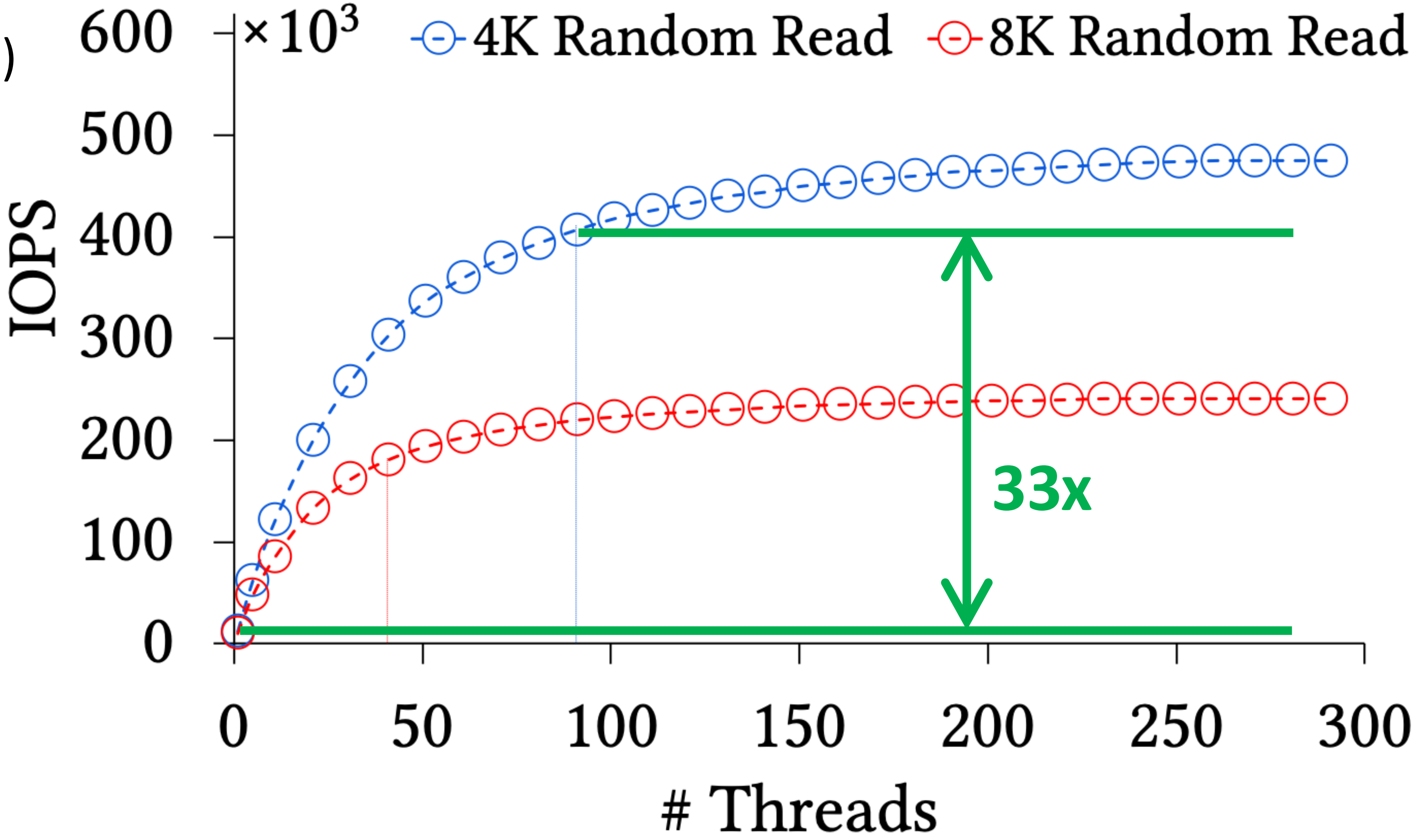
PCIe SSD - P4510 (1TB)



read concurrency (k_r) = 80

Quantifying Asymmetry & Concurrency

Device
PCIe SSD - P4510 (1TB)



Quantifying Asymmetry & Concurrency

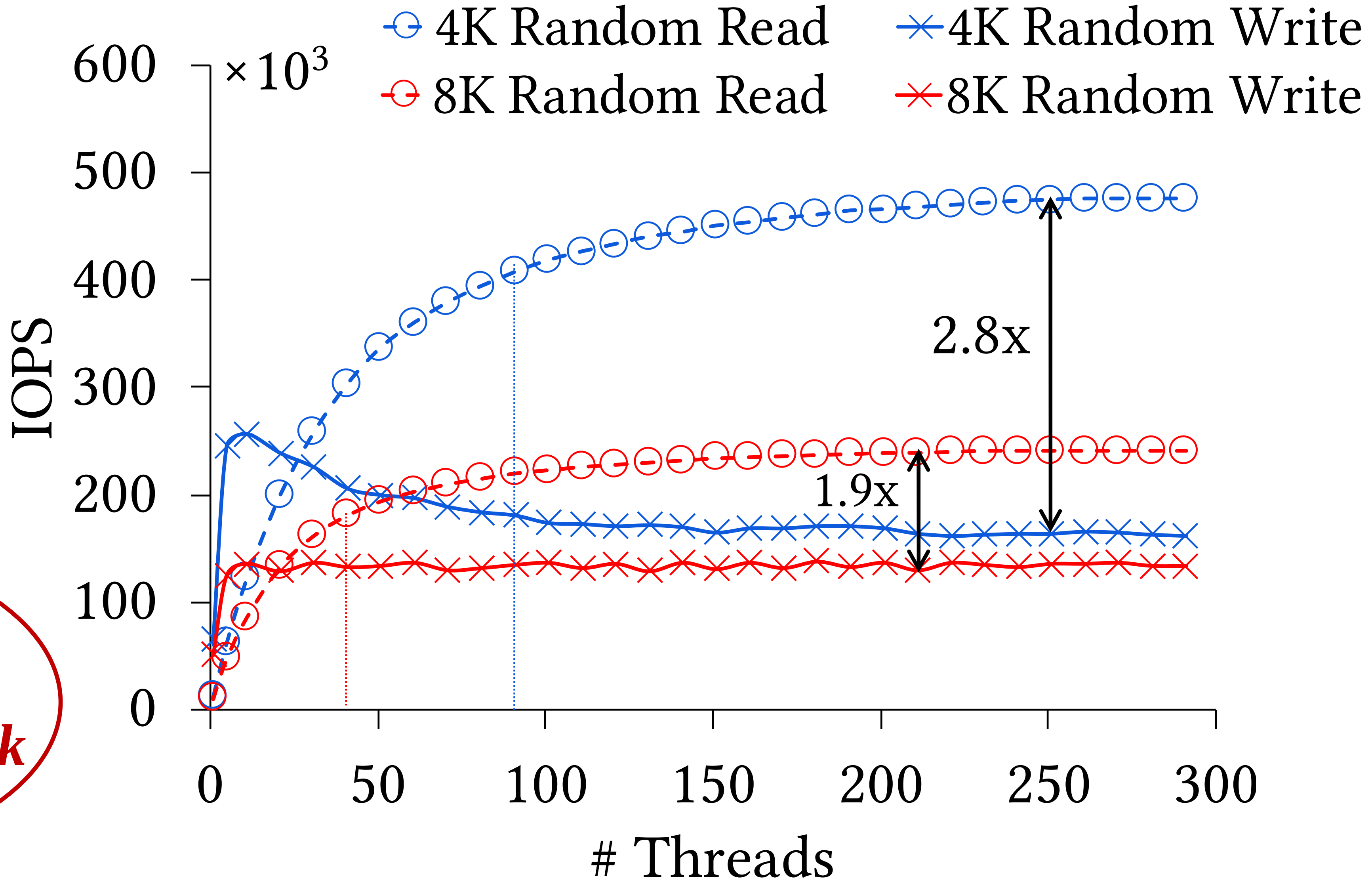
Device
PCIe SSD - P4510 (1TB)

For 4K random read,

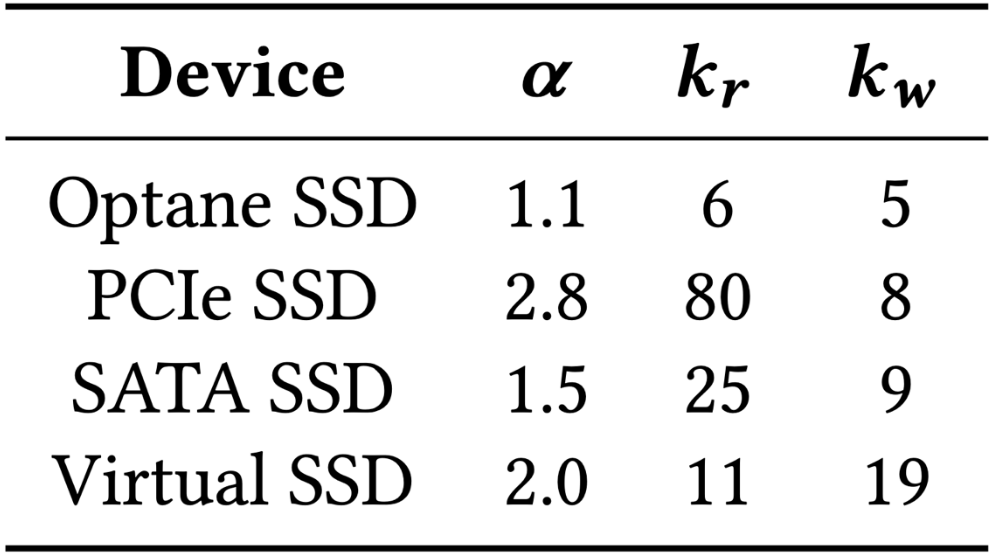
Asymmetry: 2.8

Concurrency: 80

Yet, systems are not always tailored for *α/k*



Empirical Asymmetry and Concurrency



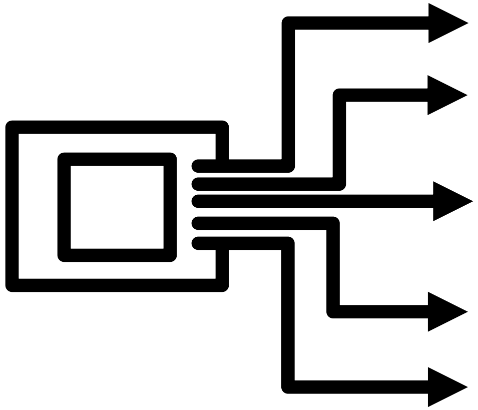
Device	α	k_r	k_w
Optane SSD	1.1	6	5
PCIe SSD	2.8	80	8
SATA SSD	1.5	25	9
Virtual SSD	2.0	11	19

- “A Parametric I/O Model for Modern Storage Devices”, DaMoN 2021

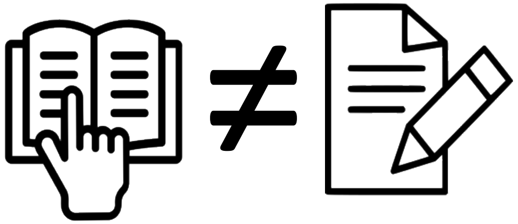
disc.bu.edu/papers/damon21-papon

Guidelines for Algorithm Design

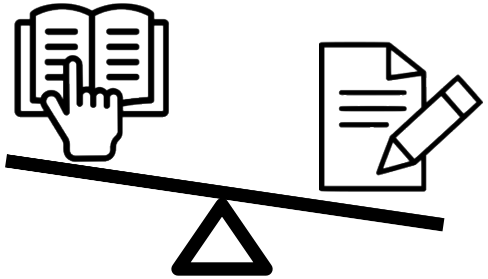
Know Thy Device



Exploit concurrency (with care)



Treat read and write differently.



asymmetry controls performance

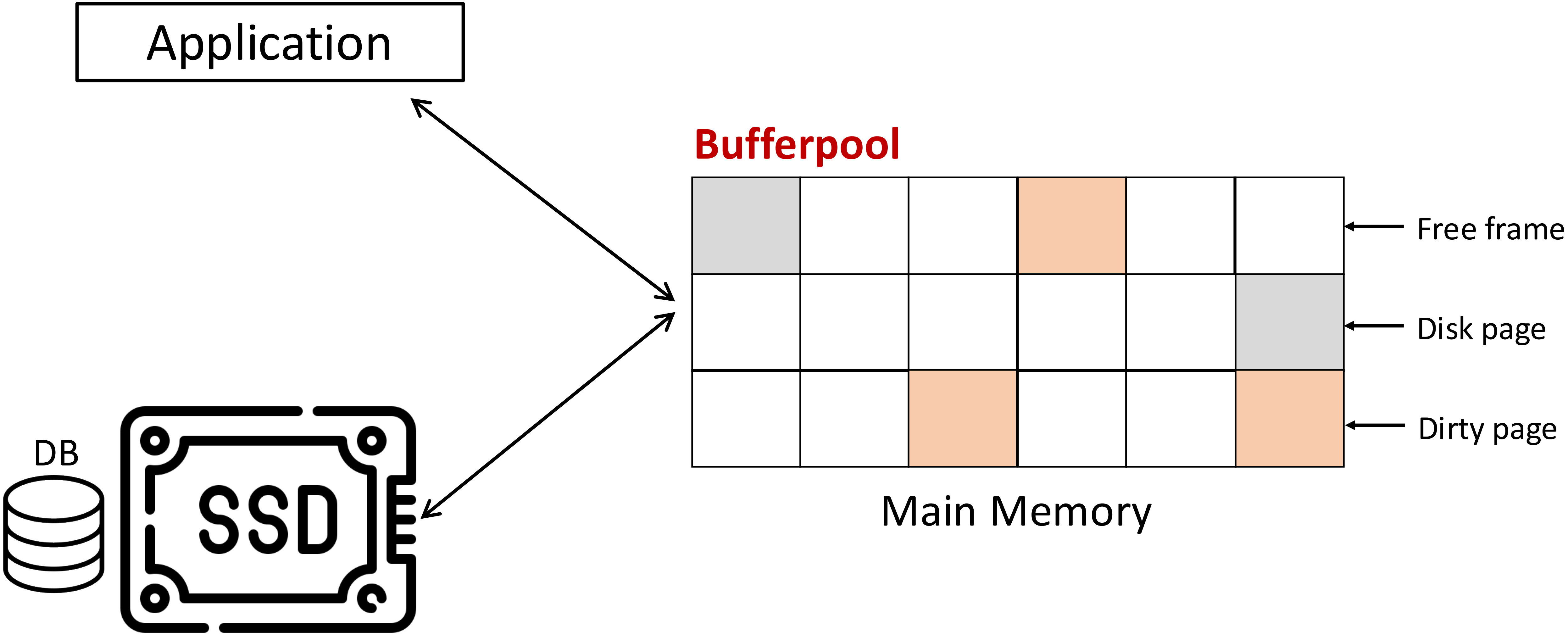
- “A Parametric I/O Model for Modern Storage Devices”, DaMoN 2021

disc.bu.edu/papers/damon21-papon

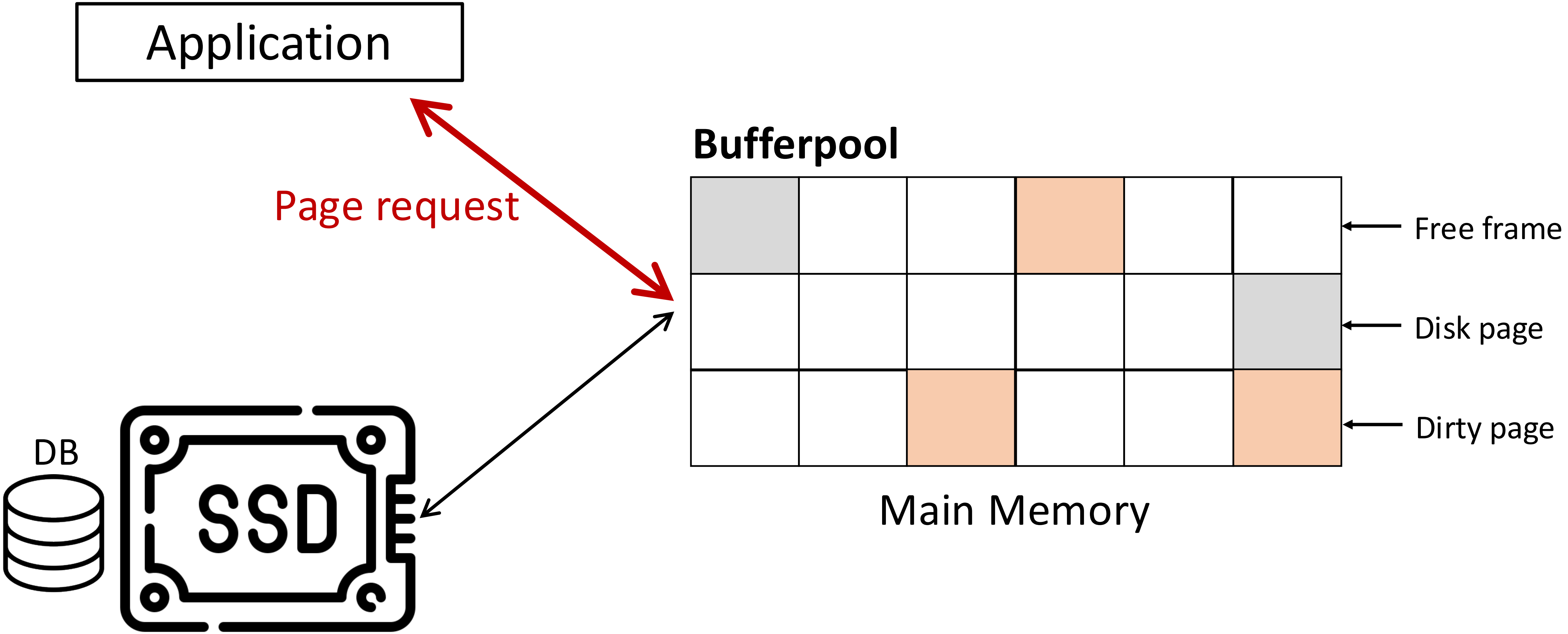
Bufferpool Manager &

The Challenge

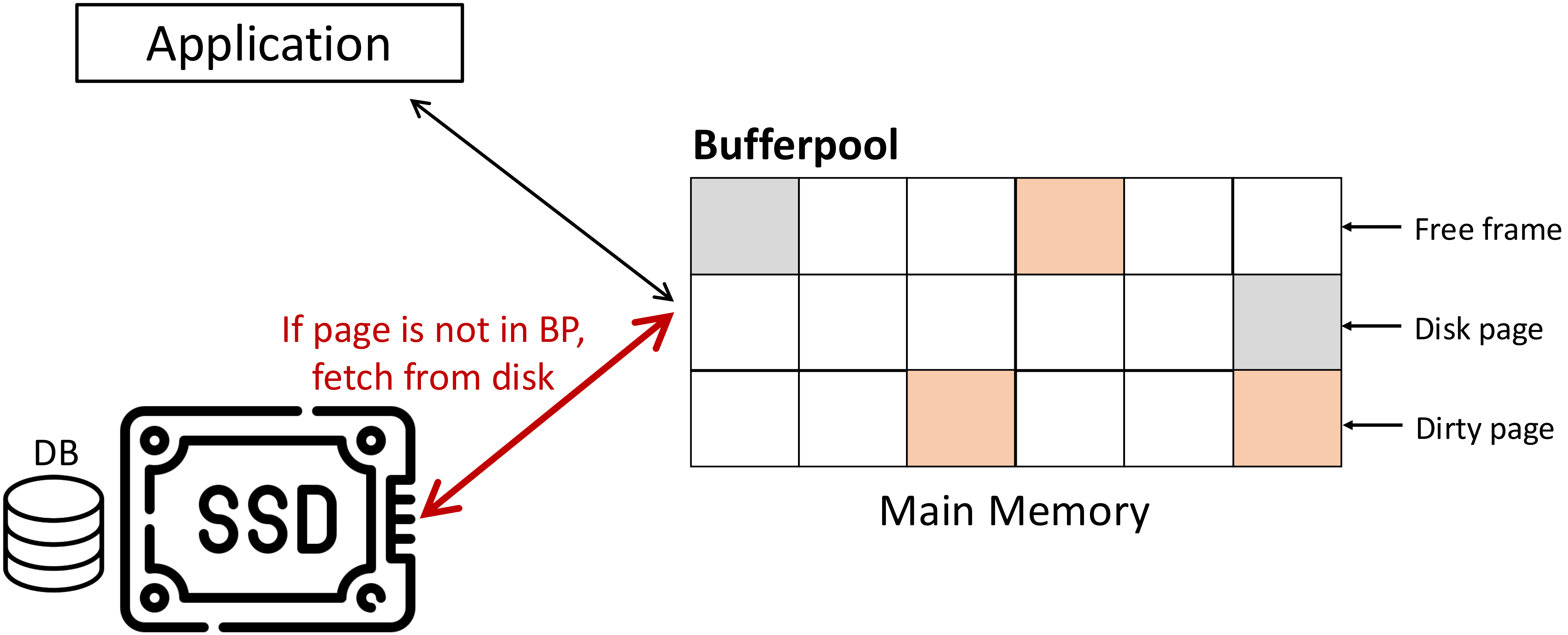
Bufferpool is Tightly Connected to Storage



Bufferpool Manager



Bufferpool Manager



Application

Bufferpool

Free frame

Disk page

Dirty page

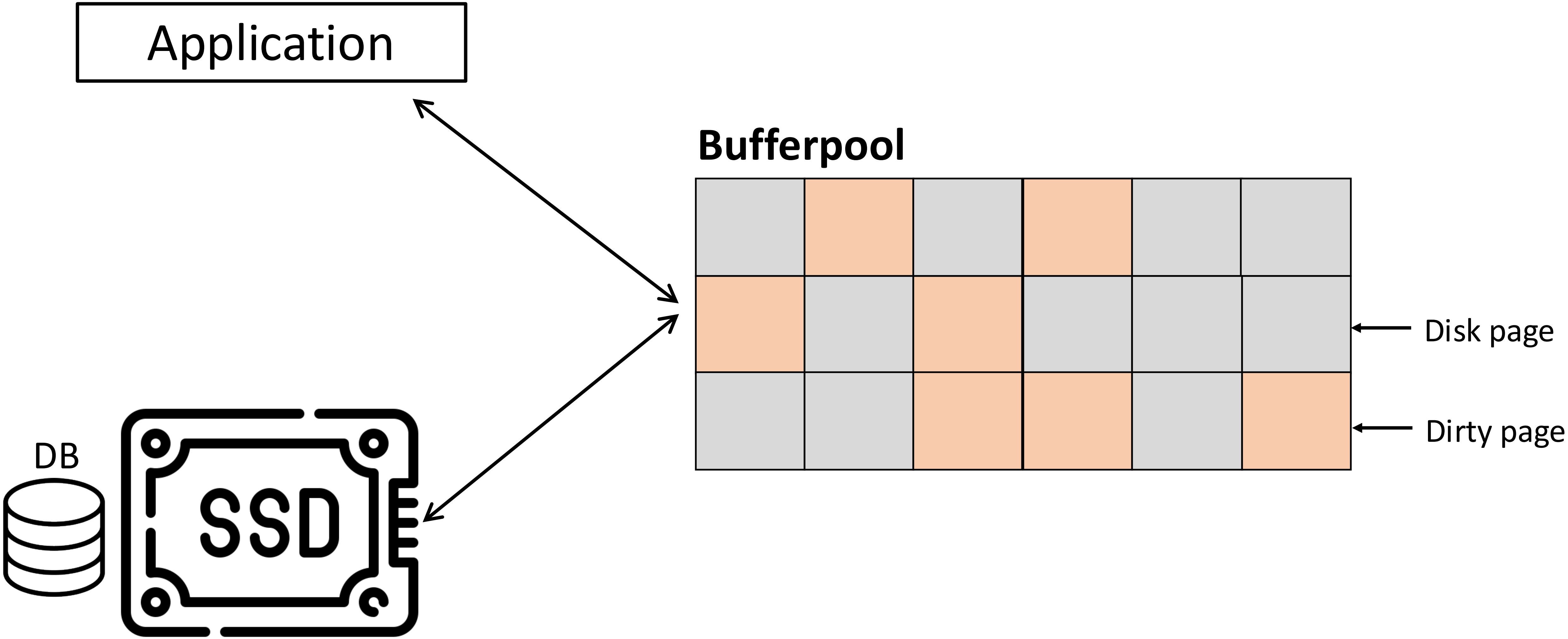
Main Memory

If page is not in BP, fetch from disk

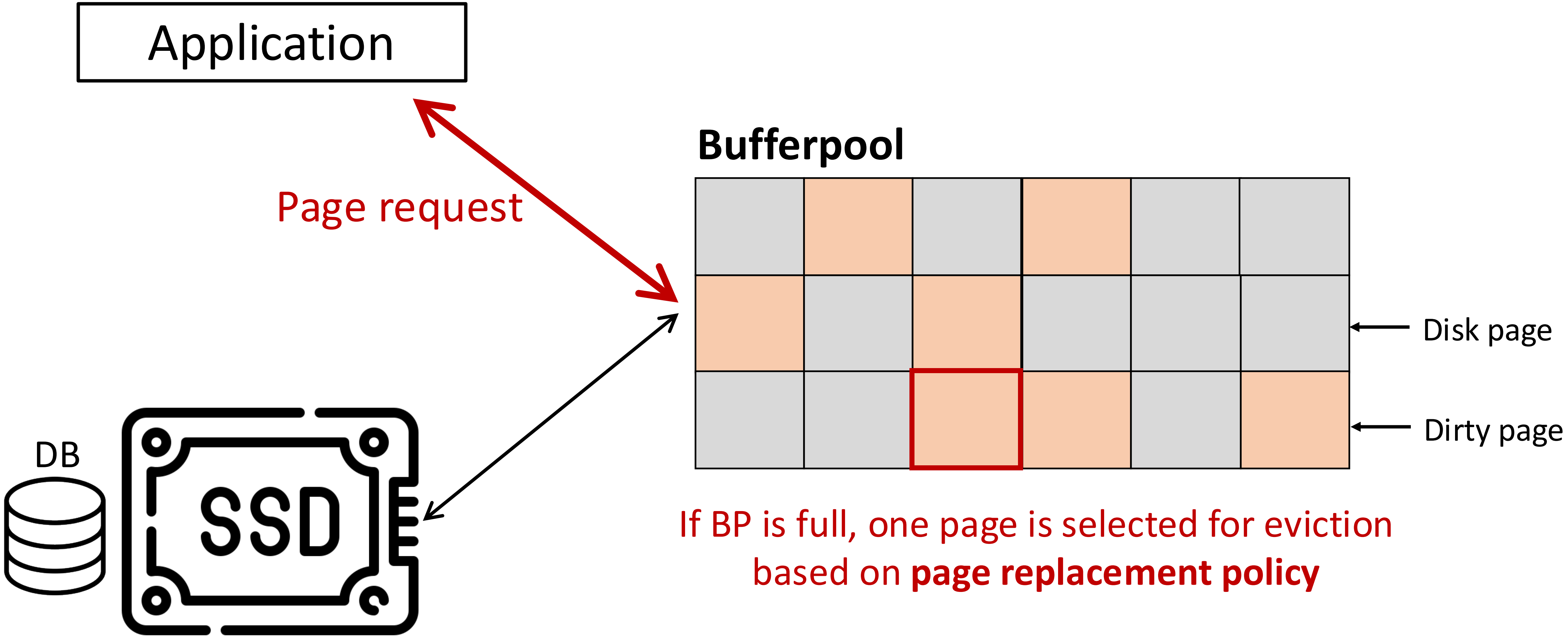
DB

SSD

Bufferpool Manager

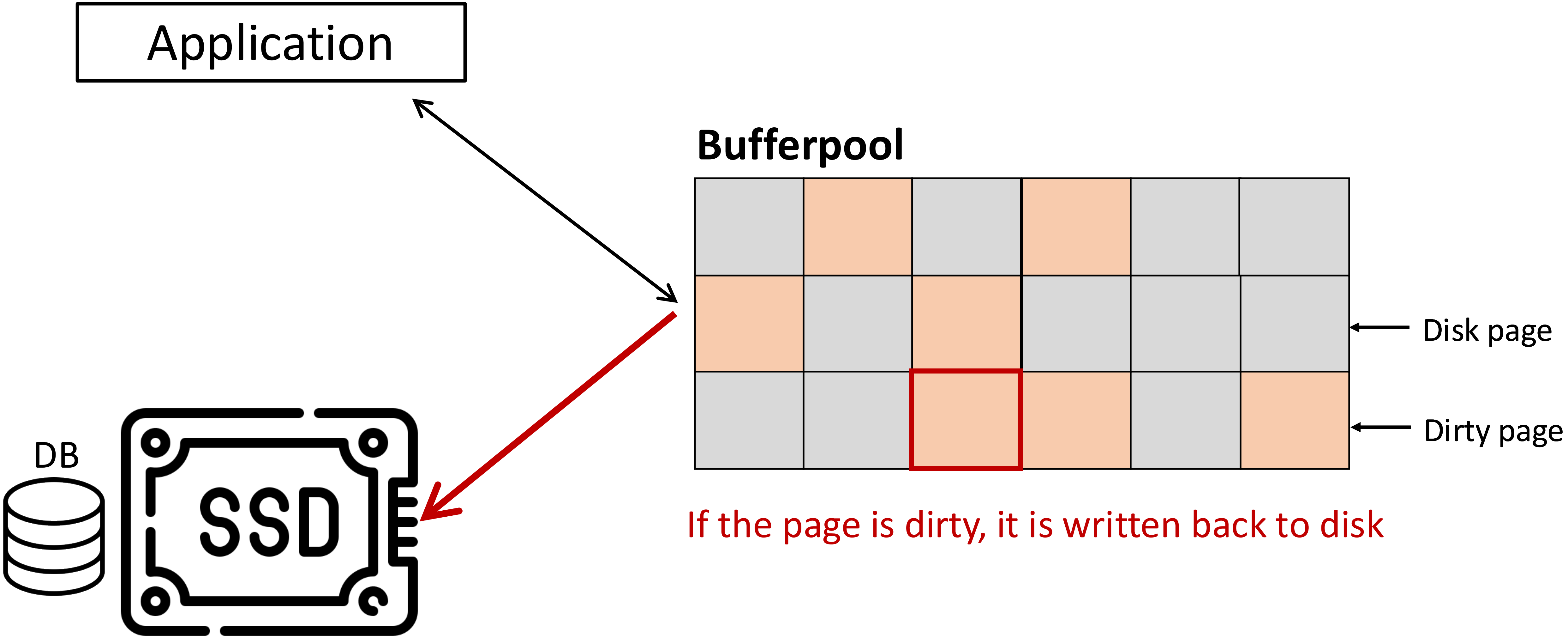


Traditional Bufferpool Manager



If BP is full, one page is selected for eviction based on **page replacement policy**

Traditional Bufferpool Manager



Application

Bufferpool

Disk page

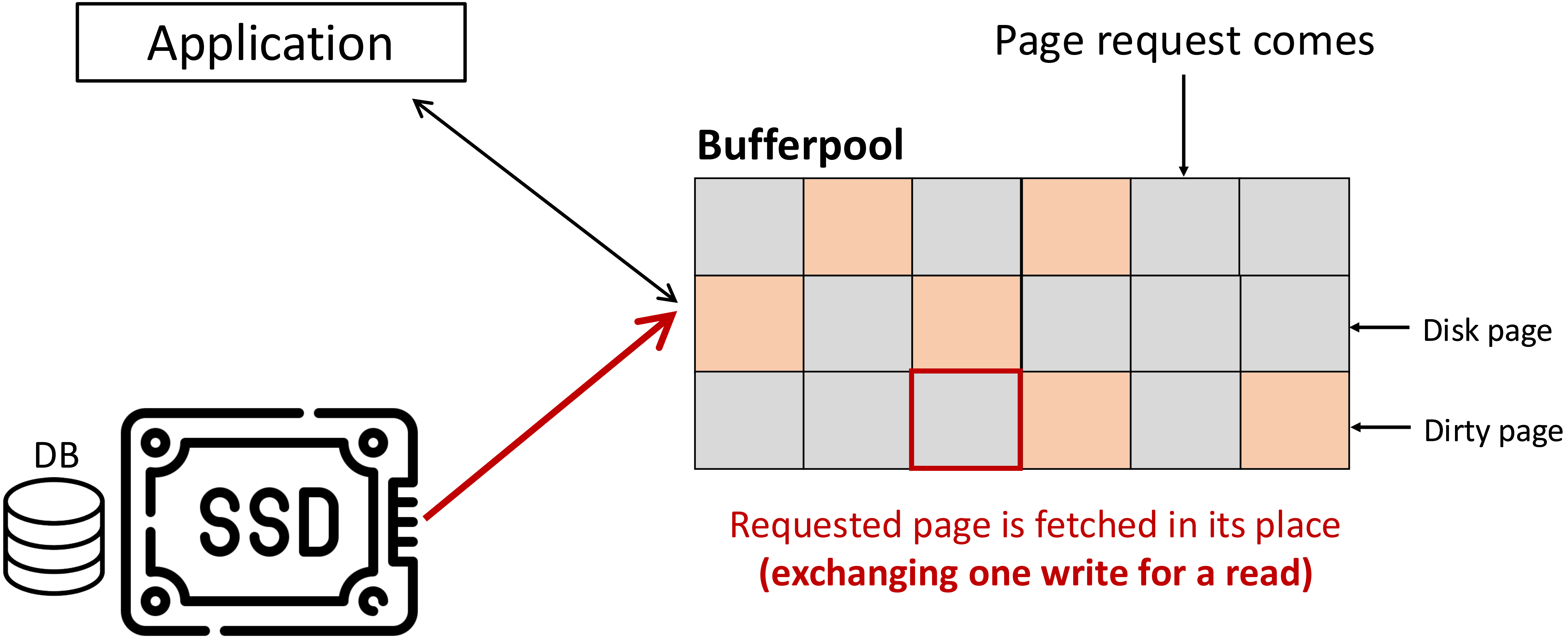
Dirty page

DB

SSD

If the page is dirty, it is written back to disk

Traditional Bufferpool Manager



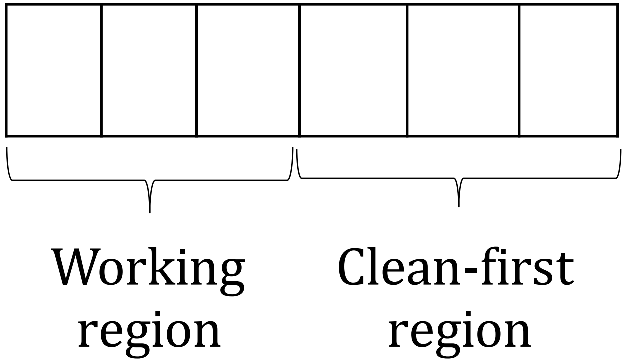
Requested page is fetched in its place

(exchanging one write for a read)

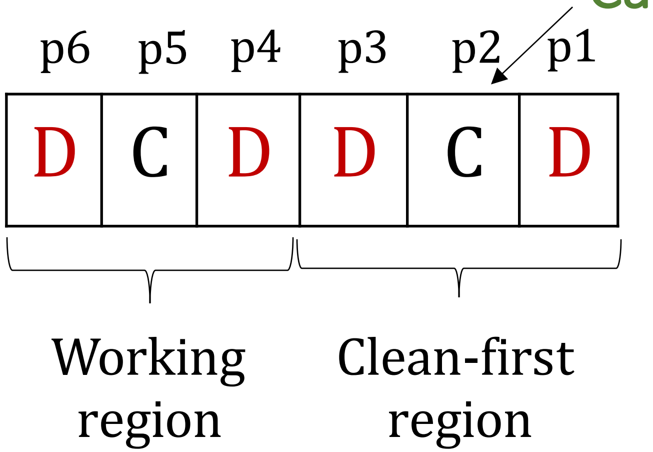
Popular Page Replacement Algorithms

LRU	(Most Popular)
LFU, FIFO	(Simple)
Clock Sweep	(Commercial)
CFLRU	Flash-Friendly
LRU-WSR	

CFLRU



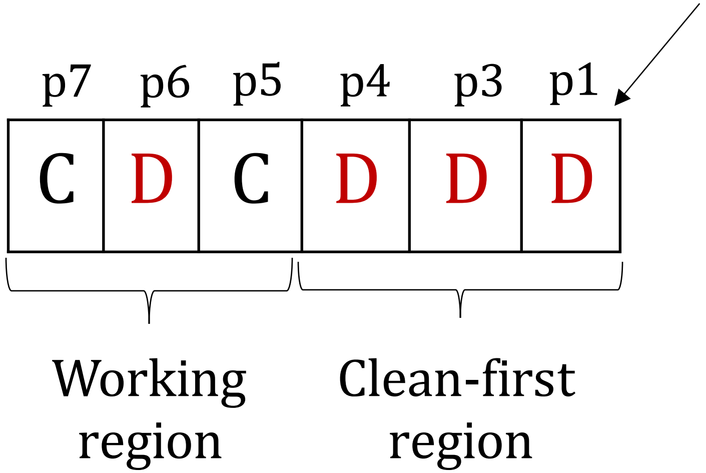
CFLRU



Candidate for eviction

p6	p5	p4	p3	p2	p1
D	C	D	D	C	D

After Eviction:



Next Candidate for eviction

p7	p6	p5	p4	p3	p1
C	D	C	D	D	D

LRU-WSR

	p6	p5	p4	p3	p2	p1
	D	C	D	D	C	D
Cold flag	1		0	0		0

Cold flag NOT set!
This is be moved to front
setting the cold flag

	p1	p6	p5	p4	p3	p2
	D	D	C	D	D	C
Cold flag	1	1		0	0	

Candidate for eviction

After Eviction:

	p7	p1	p6	p5	p4	p3
	C	D	D	C	D	D
Cold flag		1	1		0	0

LRU-WSR

	p6	p5	p4	p3	p2	p1
	D	C	D	D	C	D
Cold flag	1		0	0		1

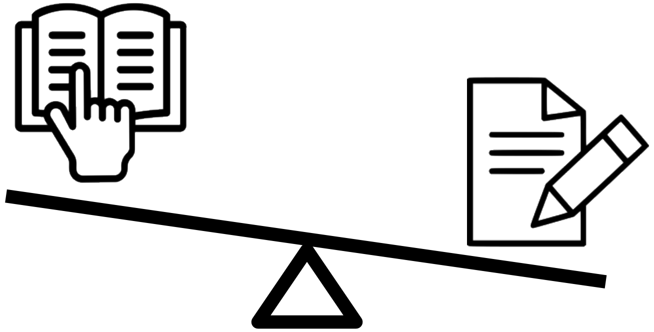
Cold flag set!
Candidate for eviction

After Eviction:

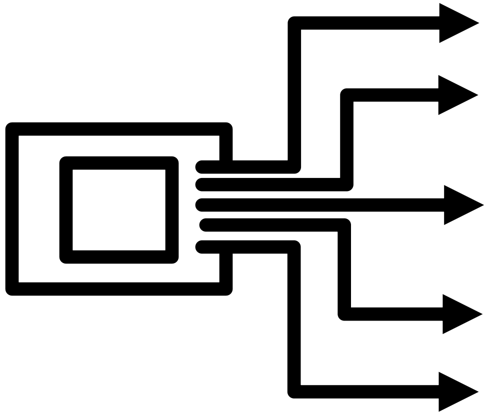
	p7	p6	p5	p4	p3	p2
	C	D	C	D	D	C
Cold flag		1		0	0	

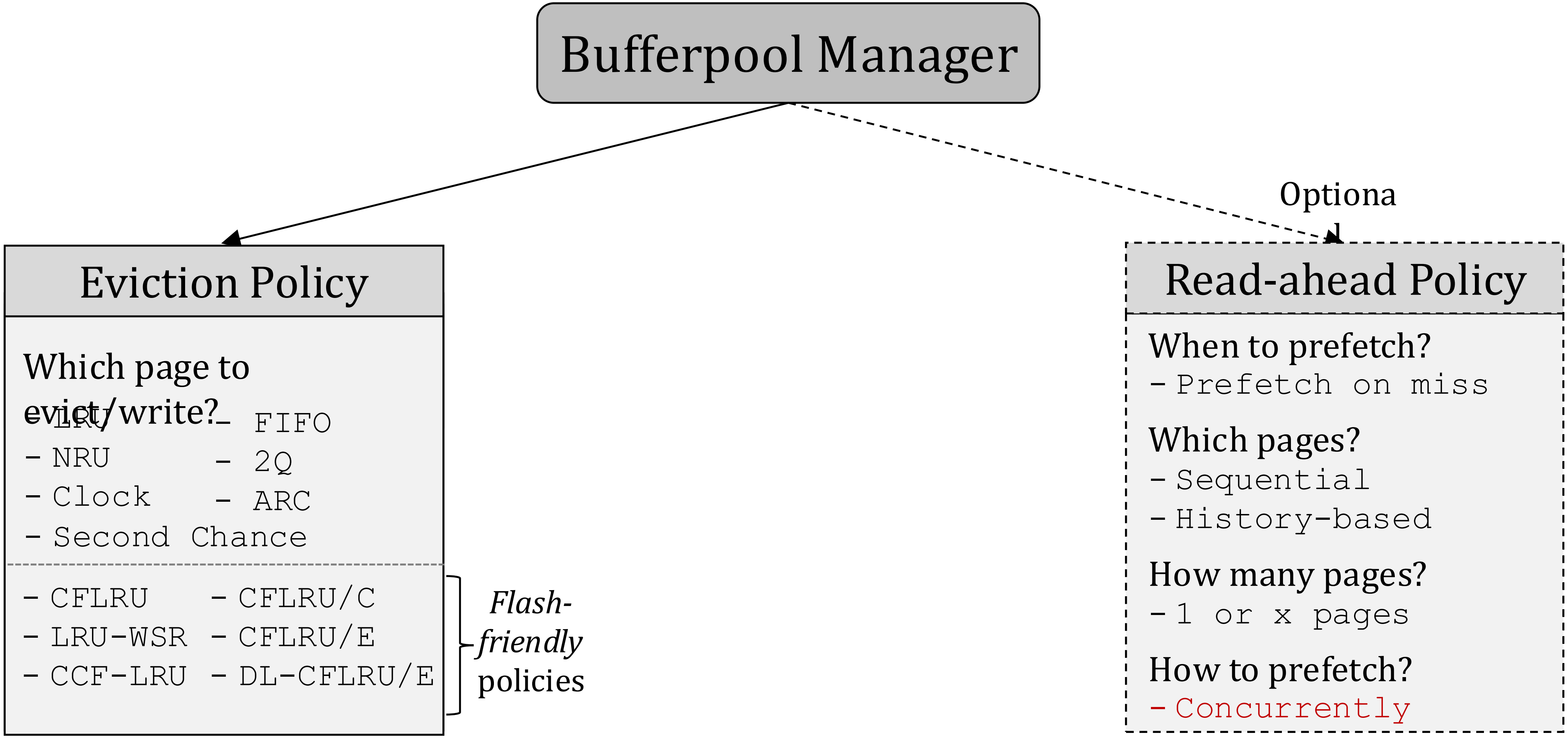
The Challenges

- With write asymmetry, exchanging one write for one read is **NOT ideal**.



- Without exploiting concurrency, device remains vastly **underutilized**.





Bufferpool Manager

Optional

Eviction Policy

Which page to evict/write?

- LRU - FIFO
- NRU - 2Q
- Clock - ARC
- Second Chance

- CFLRU - CFLRU/C
- LRU-WSR - CFLRU/E
- CCF-LRU - DL-CFLRU/E

Flash-friendly policies

replacement policy

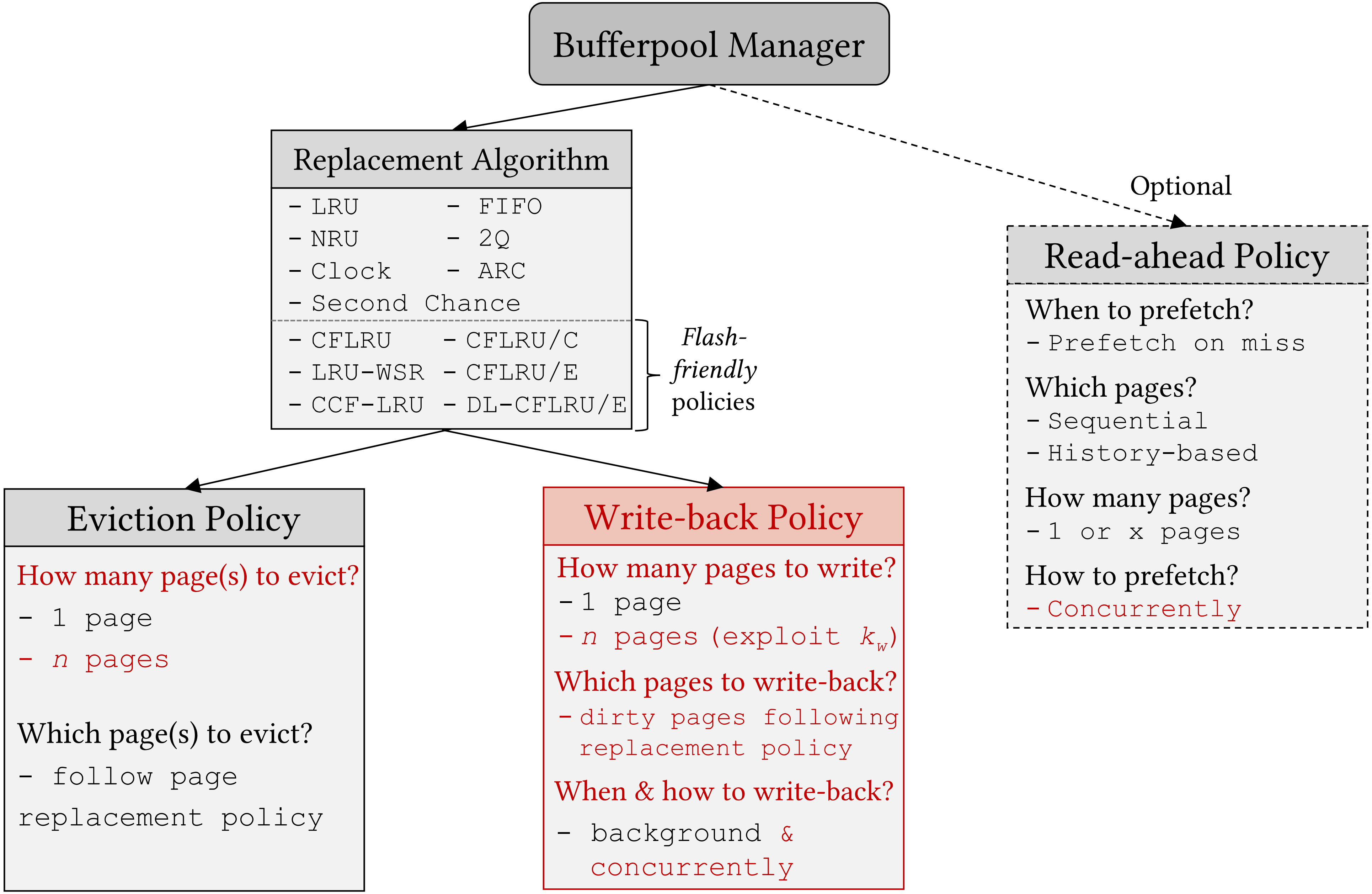
Read-ahead Policy

When to prefetch?
- Prefetch on miss

Which pages?
- Sequential
- History-based

How many pages?
- 1 or x pages

How to prefetch?
- Concurrently



Bufferpool Manager

Replacement Algorithm

- LRU
- NRU
- Clock
- Second Chance
- FIFO
- 2Q
- ARC

- CFLRU
- LRU-WSR
- CCF-LRU
- CFLRU/C
- CFLRU/E
- DL-CFLRU/E

Flash-friendly policies

Optional

Read-ahead Policy

When to prefetch?
- Prefetch on miss

Which pages?
- Sequential
- History-based

How many pages?
- 1 or x pages

How to prefetch?
- Concurrently

Eviction Policy

How many page(s) to evict?
- 1 page
- *n* pages

Which page(s) to evict?
- follow page replacement policy

Write-back Policy

How many pages to write?
- 1 page
- *n* pages (exploit k_w)

Which pages to write-back?
- dirty pages following replacement policy

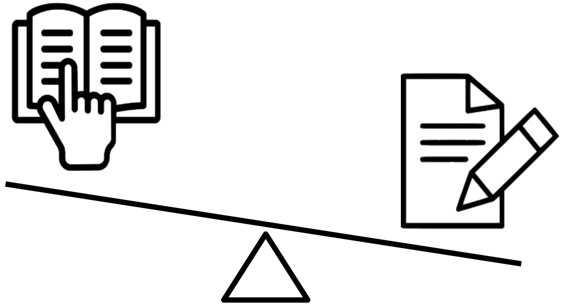
When & how to write-back?
- background & concurrently

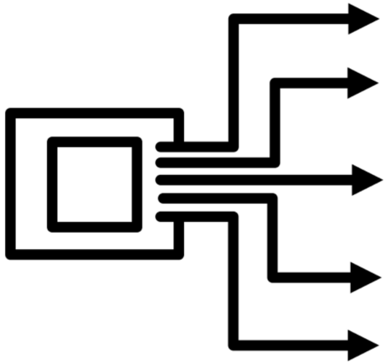
IEEE ICDE 2023

Asymmetry/Concurrency-Aware (**ACE**) Bufferpool Manager

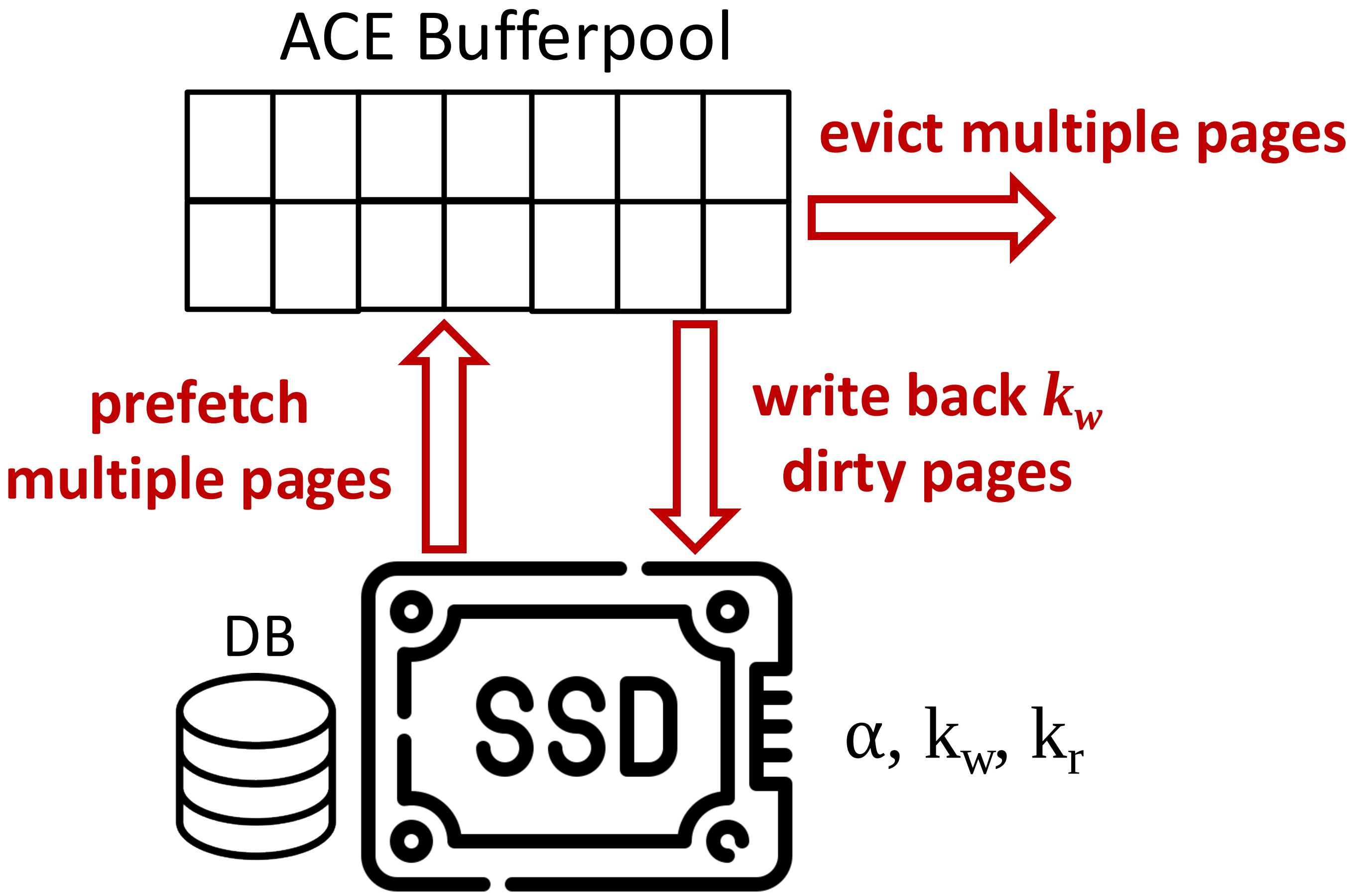
ACE Bufferpool Manager

Use device's properties





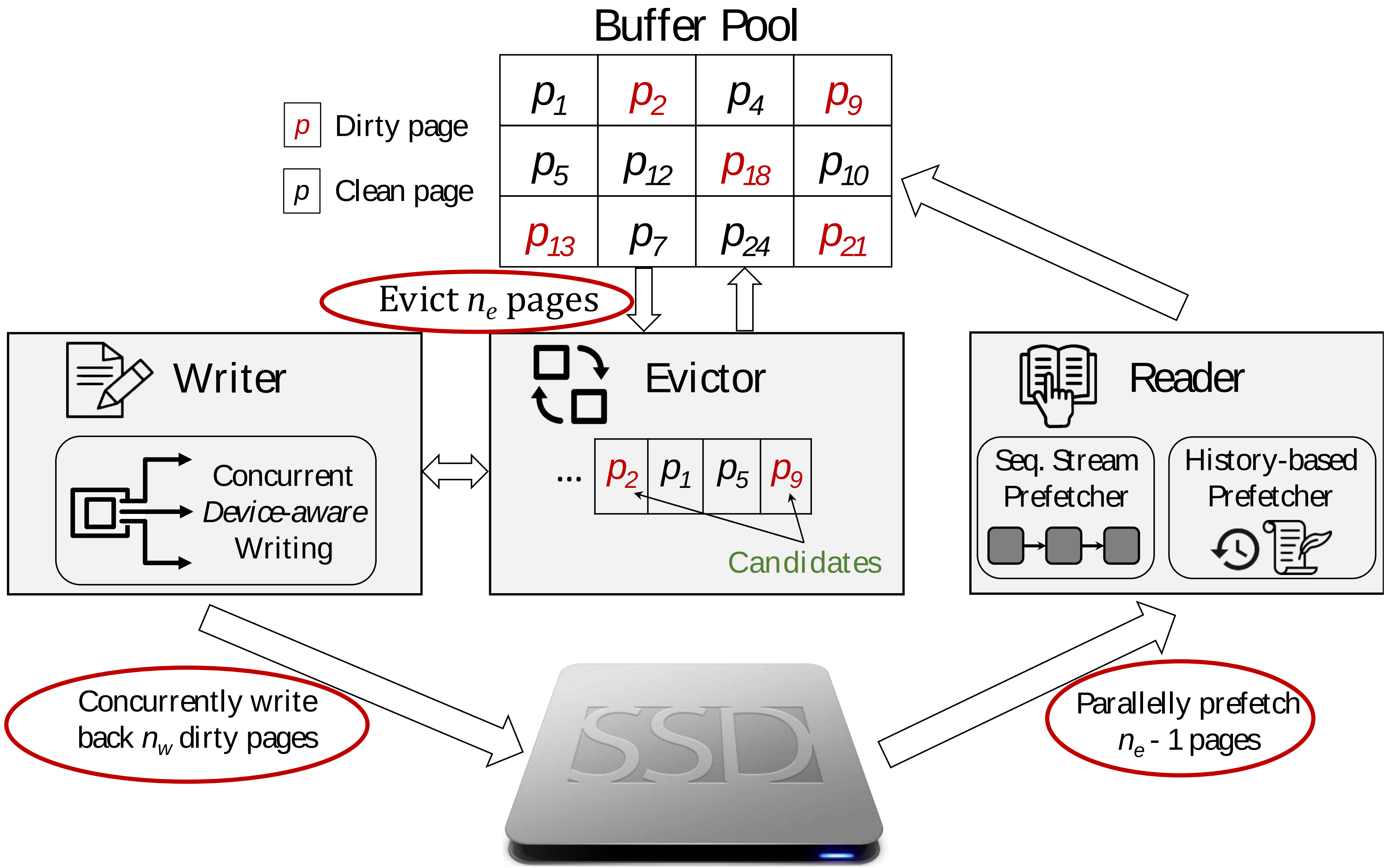
ACE Bufferpool Manager



- ✓ Can be integrated with **any** replacement algorithm
- ✓ **Any** prefetching technique can be used

IEEE ICDE 2023

ACE Bufferpool Manager



Buffer Pool

p_1	p_2	p_4	p_9
p_5	p_{12}	p_{18}	p_{10}
p_{13}	p_7	p_{24}	p_{21}

p Dirty page

p Clean page

Evict n_e pages

Writer

Concurrent *Device-aware* Writing

Evictor

… p_2 p_1 p_5 p_9

Candidates

Reader

Seq. Stream Prefetcher

History-based Prefetcher

SSD

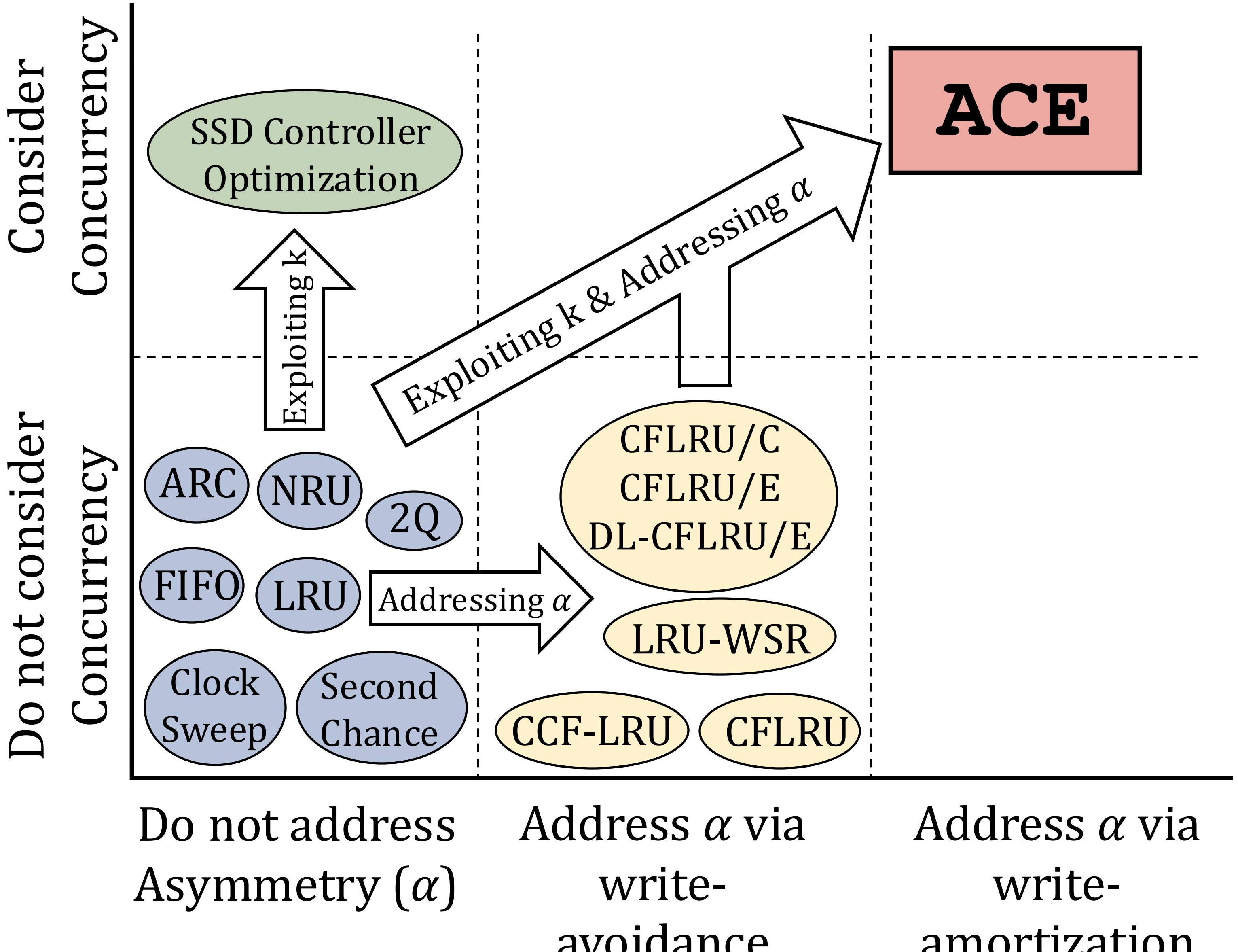
Concurrently write back n_w dirty pages

Parallelly prefetch n_e - 1 pages

ACE Bufferpool Manager

IEEE ICDE 2023

- Better device utilization ✓
- High performance ✓
- Low deployment cost ✓
- **Ease of integration** ✓



Consider Concurrency

Do not consider Concurrency

SSD Controller Optimization

Exploiting k

Exploiting k & Addressing α

ACE

ARC

NRU

2Q

FIFO

LRU

Clock Sweep

Second Chance

Addressing α

CFLRU/C
CFLRU/E
DL-CFLRU/E

LRU-WSR

CCF-LRU

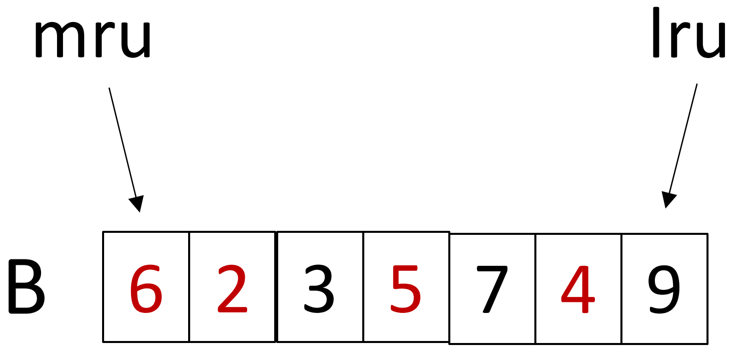
CFLRU

Do not address Asymmetry (α)

Address α via write-avoidance

Address α via write-amortization

An Example ($k_w = 3$)



mru

lru

B

6	2	3	5	7	4	9

Let’s assume: $k_w = 3$, LRU is the baseline replacement policy & red indicates dirty page

Write request of page 8 comes

An Example ($k_w = 3$)

write page 8

Candidate for eviction ↓

B	6	2	3	5	7	4	9

Since candidate page is clean, we simply evict 9

After eviction:

B	8	6	2	3	5	7	4

Write request of page 1 comes

An Example ($k_w = 3$)

write page 1

LRU

Candidate

B	8	6	2	3	5	7	4

After eviction:

B	1						

An Example ($k_w = 3$)

write page 1

LRU

B	8	6	2	3	5	7	4

After eviction:

B	1	8	6	2	3	5	7

LRU+ACE (w/o PF)

Candidate

8	6	2	3	5	7	4

An Example ($k_w = 3$)

write page 1

LRU

B	8	6	2	3	5	7	4

After eviction:

B	1	8	6	2	3	5	7

LRU+ACE (w/o PF)

Candidate

8	6	2	3	5	7	4

SSD

4,5,2 concurrently written

4 evicted

An Example ($k_w = 3$)

write page 1

LRU

B							
	8	6	2	3	5	7	4

After eviction:

B							
	1	8	6	2	3	5	7

LRU+ACE (w/o PF)

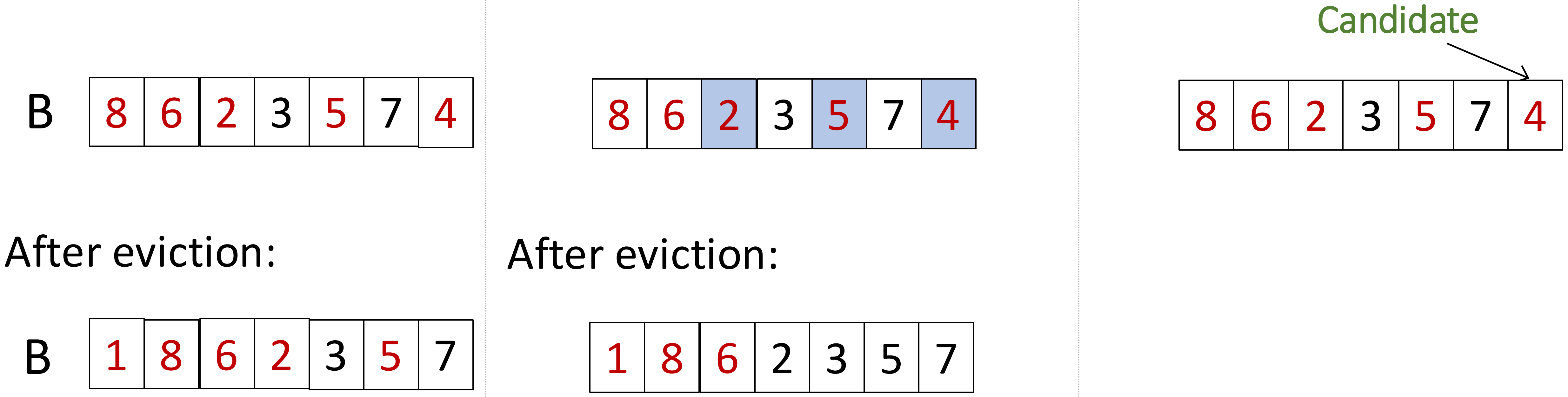
8	6	2	3	5	7	

After eviction:

1	8	6	2	3	5	7

An Example ($k_w = 3, n_e = 2$)

write page 1



LRU

B

8	6	2	3	5	7	4

After eviction:

B

1	8	6	2	3	5	7

LRU+ACE (w/o PF)

8	6	2	3	5	7	4

After eviction:

1	8	6	2	3	5	7

LRU+ACE (w/PF)

Candidate

8	6	2	3	5	7	4

An Example ($k_w = 3, n_e = 2$)

write page 1

LRU

B	8	6	2	3	5	7	4

After eviction:

B	1	8	6	2	3	5	7

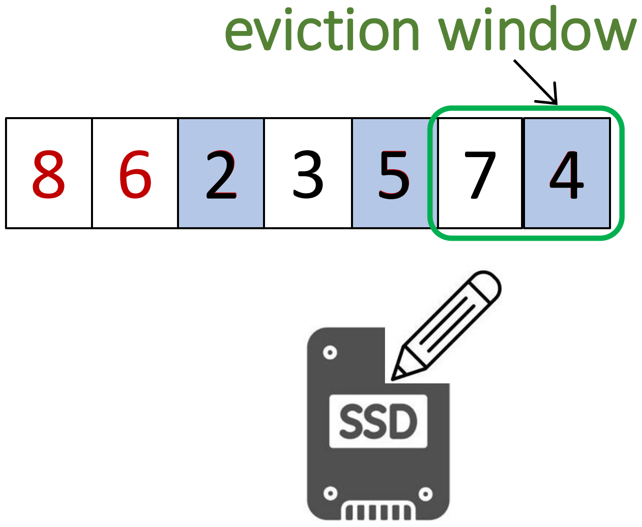
LRU+ACE (w/o PF)

8	6	2	3	5	7	4

After eviction:

1	8	6	2	3	5	7

LRU+ACE (w/PF)



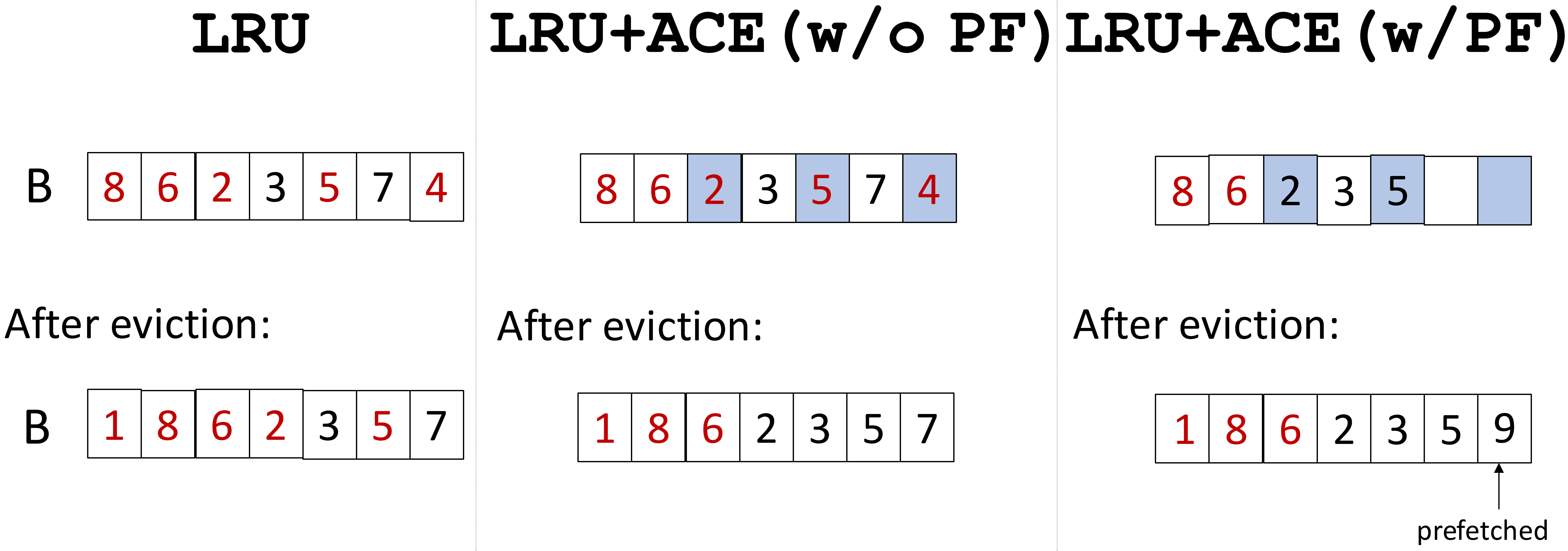
8	6	2	3	5	7	4

4,5,2 concurrently written

4,7 evicted

An Example ($k_w = 3, n_e = 2$)

write page 1



LRU

B						
8	6	2	3	5	7	4

After eviction:

B						
1	8	6	2	3	5	7

LRU+ACE (w/o PF)

8	6	▨ 2	3	▨ 5	7	▨ 4

After eviction:

1	8	6	2	3	5	7

LRU+ACE (w/PF)

8	6	▨ 2	3	▨ 5		▨

After eviction:

1	8	6	2	3	5	9

prefetched

Experimental Evaluation

PostgreSQL

11.5

Clock Sweep
LRU
CFLRU
LRU-WSR

vs their ACE counterparts

Device	α	k_r	k_w
Optane SSD	1.1	6	5
PCIe SSD	2.8	80	8
SATA SSD	1.5	25	9
Virtual SSD	2.0	11	19

Workload:

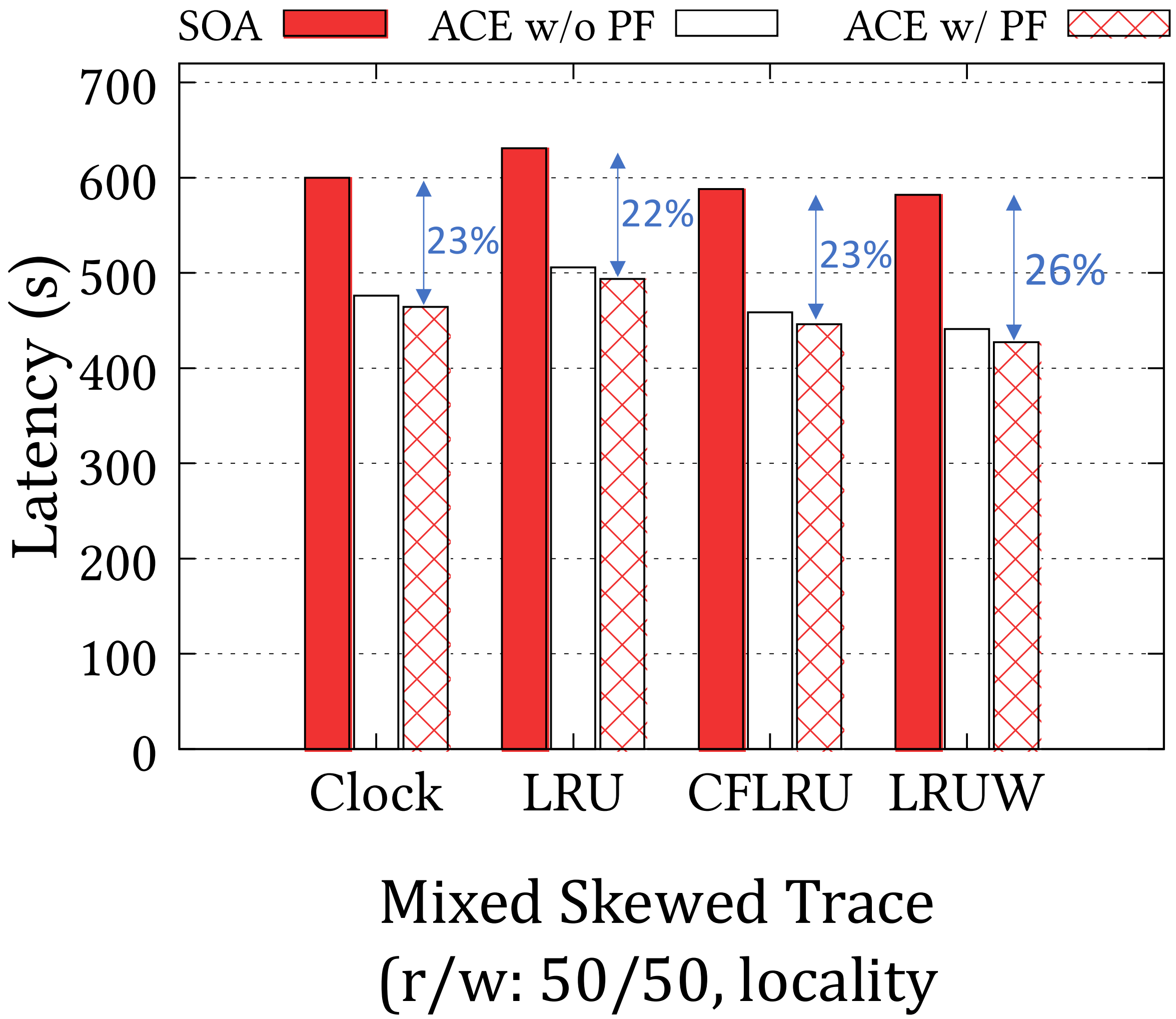
synthesized traces

TPC-C benchmark

ACE Improves Runtime

Device: PCIe SSD

$\alpha = 2.8, k_w = 8$



ACE improves runtime by 22-26%

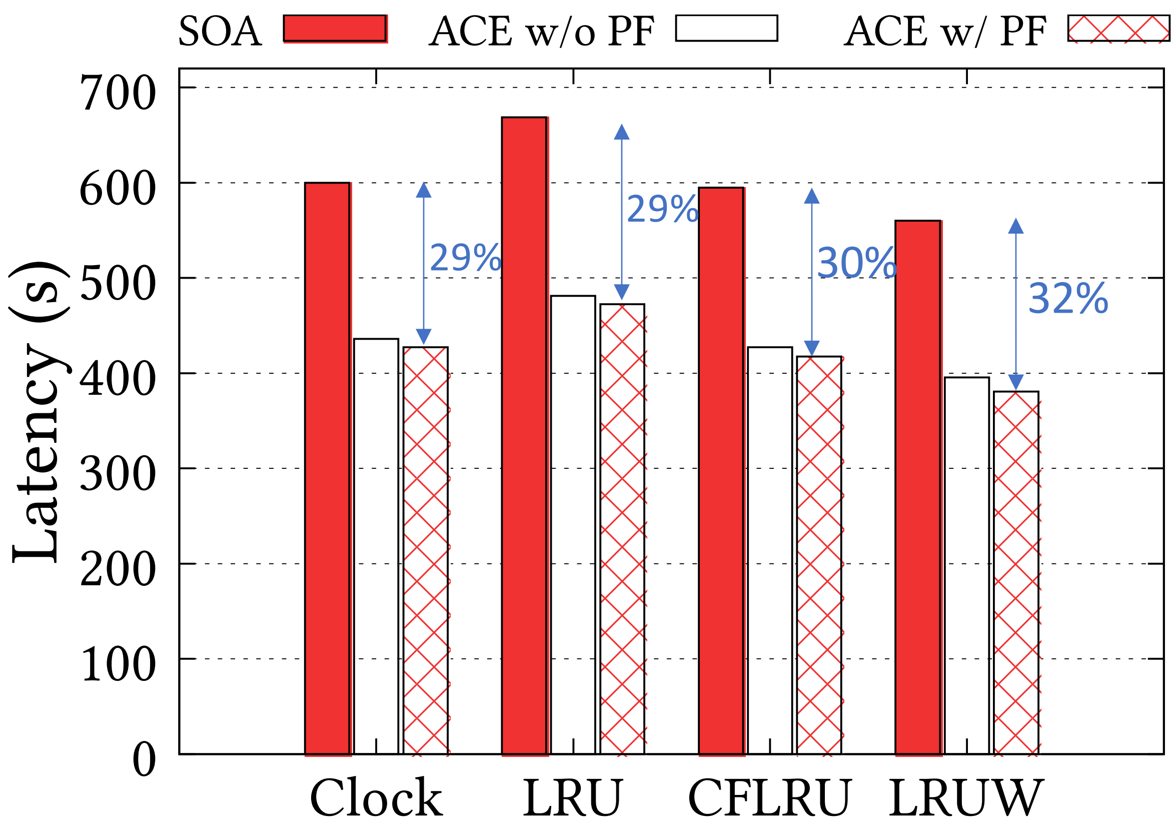
Negligible increase in buffer miss (<0.009%)

Benefit comes at no cost

Higher Gain for Write-Heavy Workload

Device: PCIe SSD

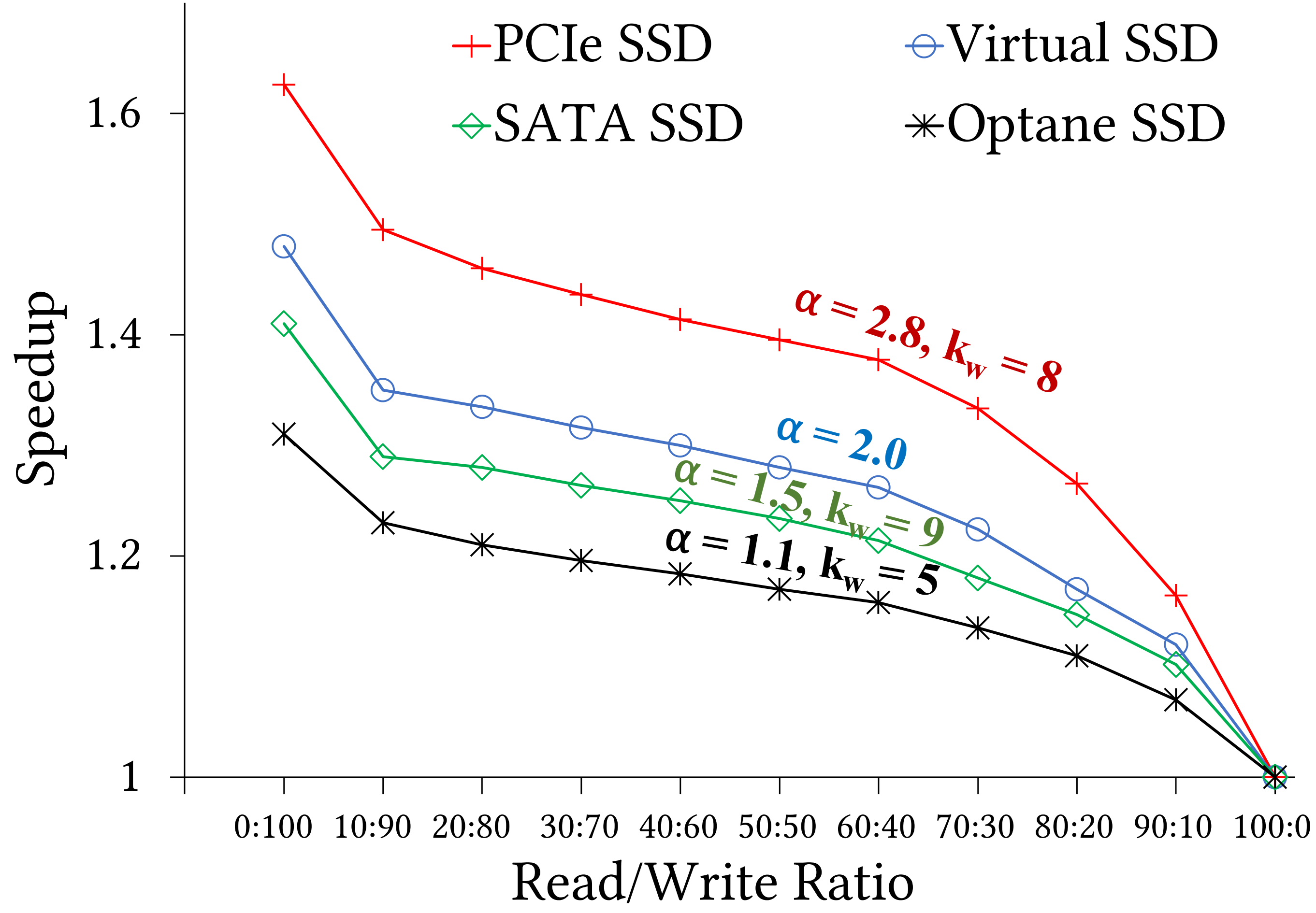
$\alpha = 2.8$, $k_w = 8$



Mixed Skewed Trace
(r/w: 50/50, locality

Write-intensive workloads have higher benefit (up to 32%)

Impact of R/W Ratio & Asymmetry

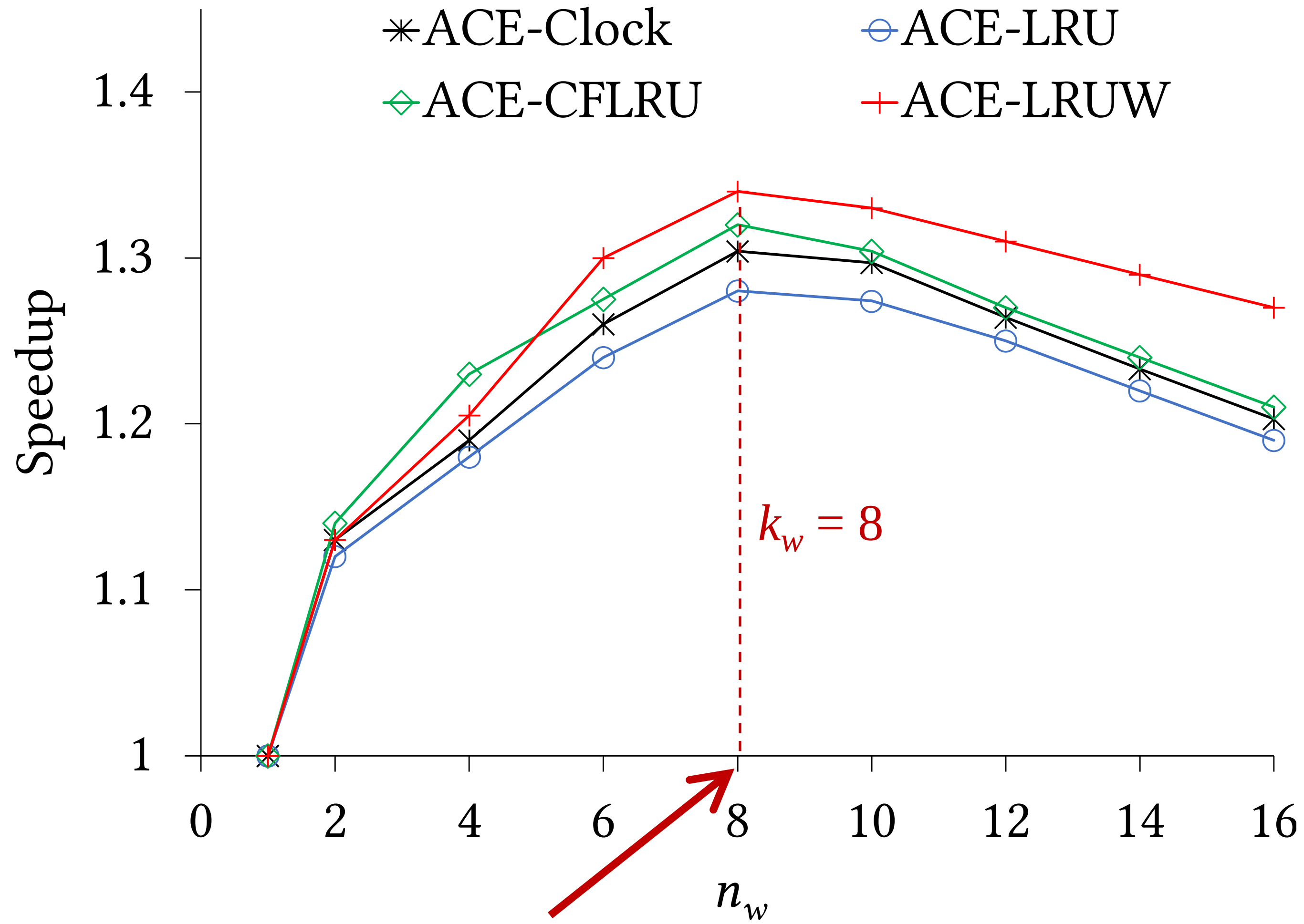


more writes, more speedup

higher asymmetry, higher speedup

good benefit even for low asymmetry

Impact of #Concurrent I/Os



Device: PCIe SSD

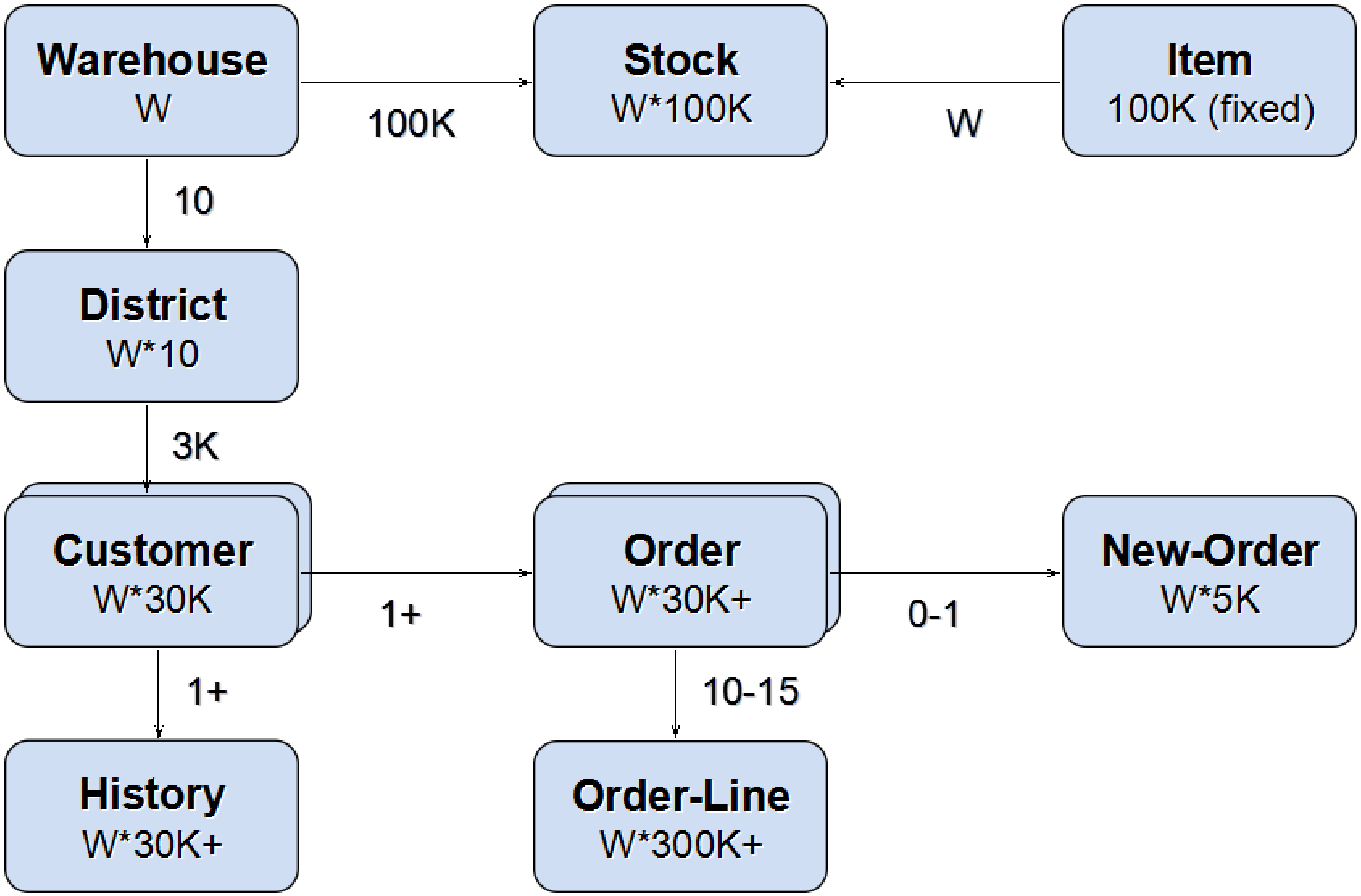
$\alpha = 2.8$, $k_w = 8$

Highest speedup when optimal concurrency is used

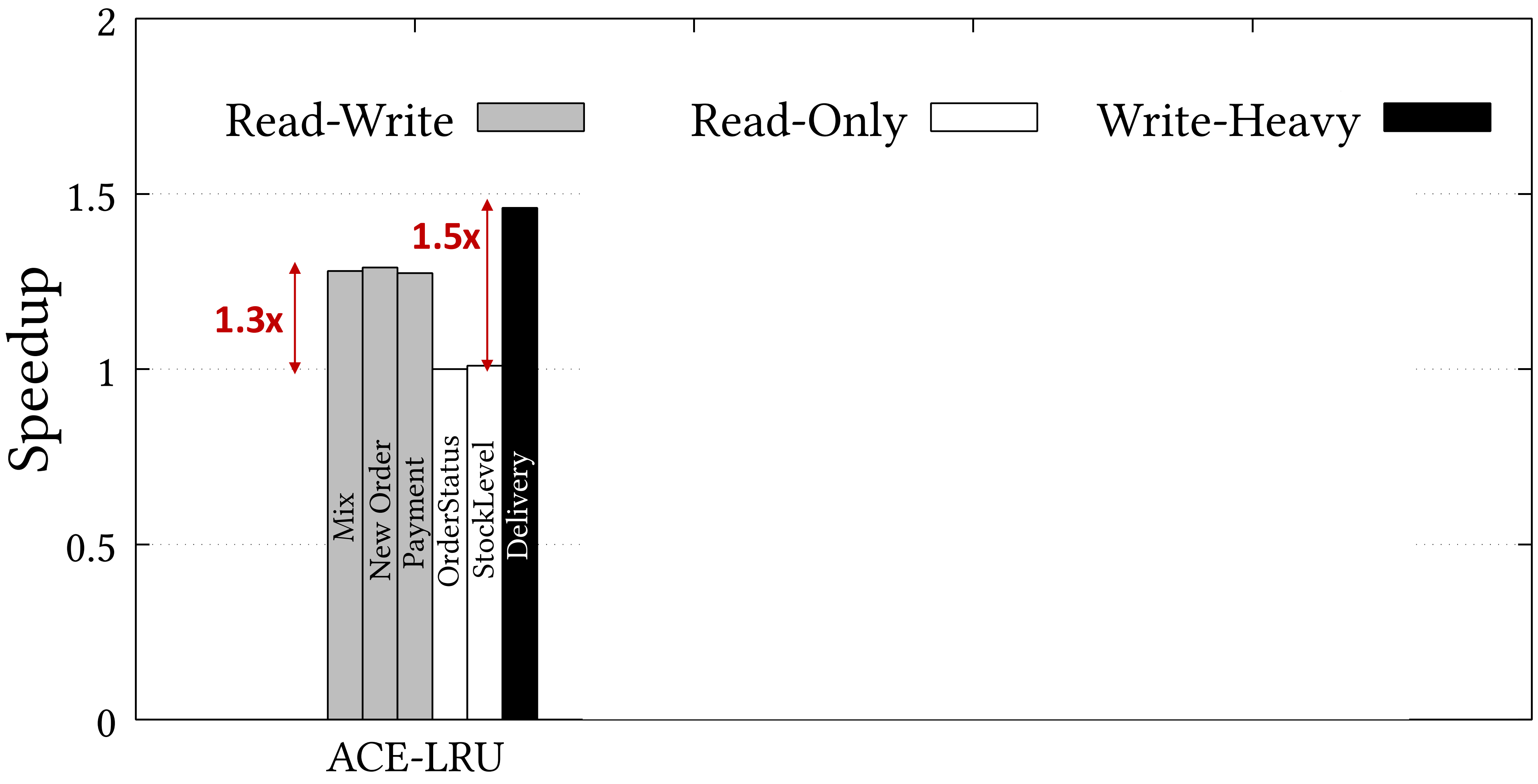
Mixed Skewed Trace

(r/w: 50/50, locality

Experimental Evaluation (TPC-C)

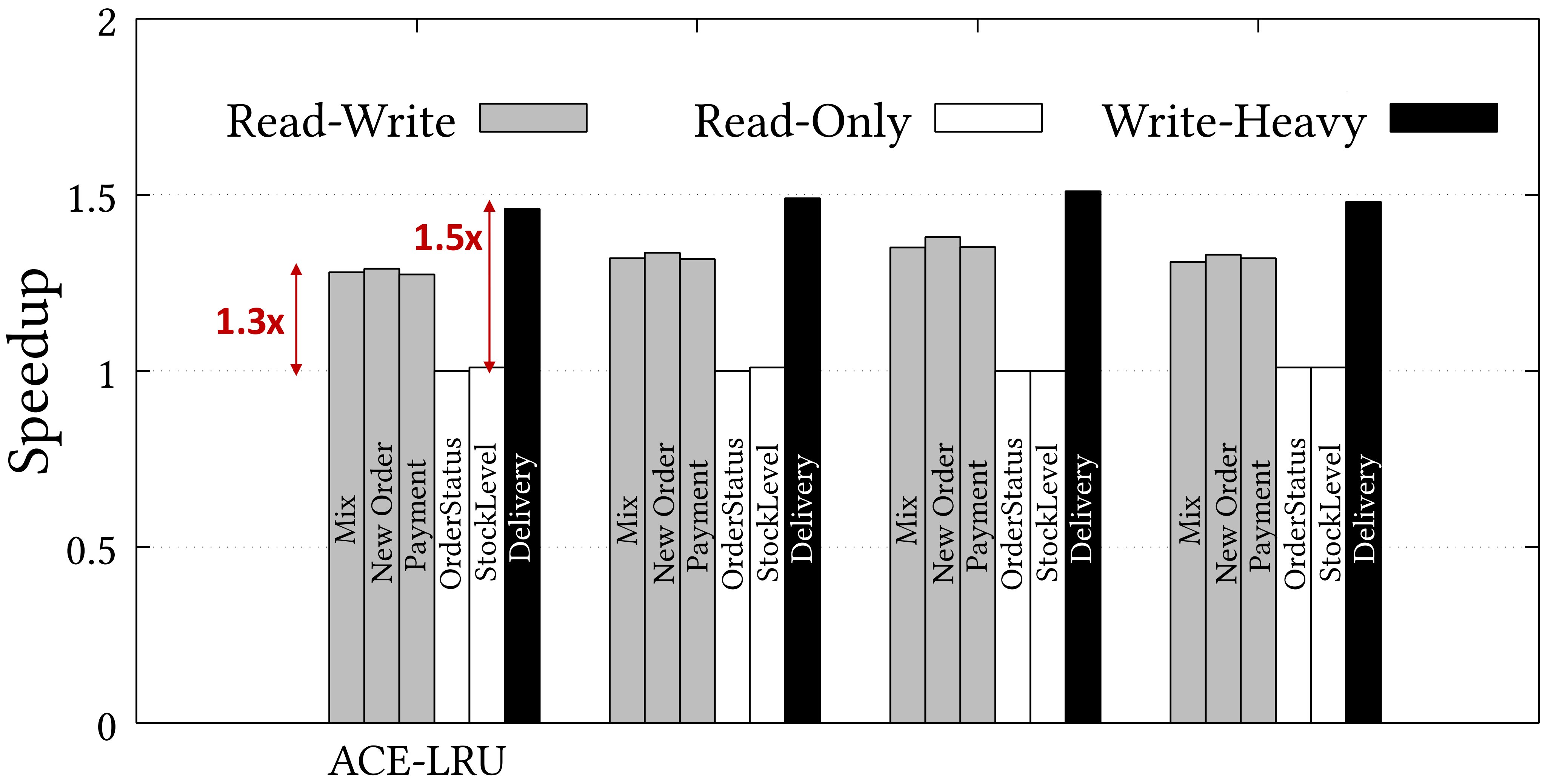


Experimental Evaluation (TPC-C)



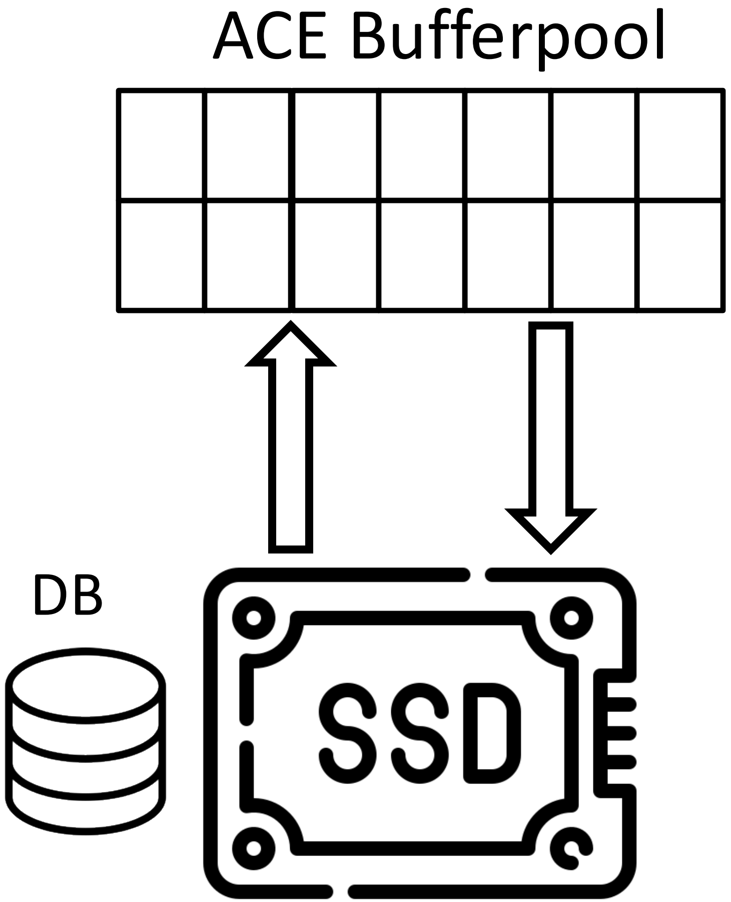
ACE Achieves 1.3x for mixed TPC-C

Experimental Evaluation (TPC-C)



ACE Achieves 1.3x for mixed TPC-C

Summary



☑ Decoupled eviction and write-back mechanism

☑ **ACE** works with **any** page replacement policy

☑ **Any** prefetching technique can be used

☑ With low engineering effort, **any** DBMS bufferpool can benefit from this approach

Conclusion & Future Work

Make *asymmetry and concurrency* part of *algorithm design*

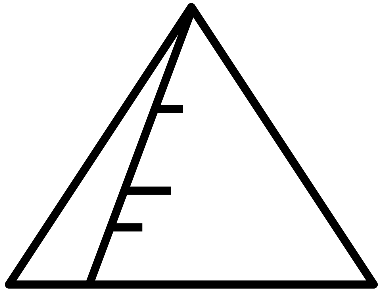
… not simply an engineering optimization

Build algorithms/data structures for storage devices

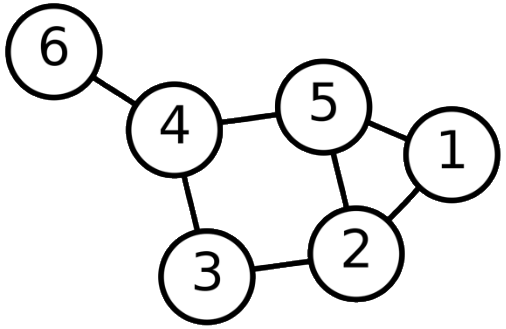
with asymmetry α and concurrency k

Stay Tuned!

index structures



graph traversal



SIGMOD '24

CAVE: Concurrency-Aware Graph Processing on SSDs

disc.bu.edu/papers/sigmod24-cave

Thank You!

disc.bu.edu/papers/icde23-papon