BOSTON UNIVERSITY

CS 561: Data Systems Architectures

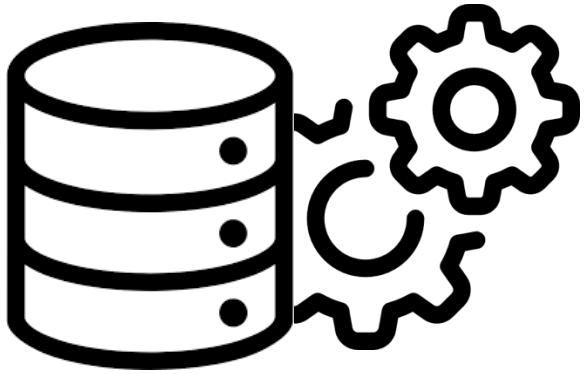
Class 16

Asymmetry and Concurrency Aware Storage Management

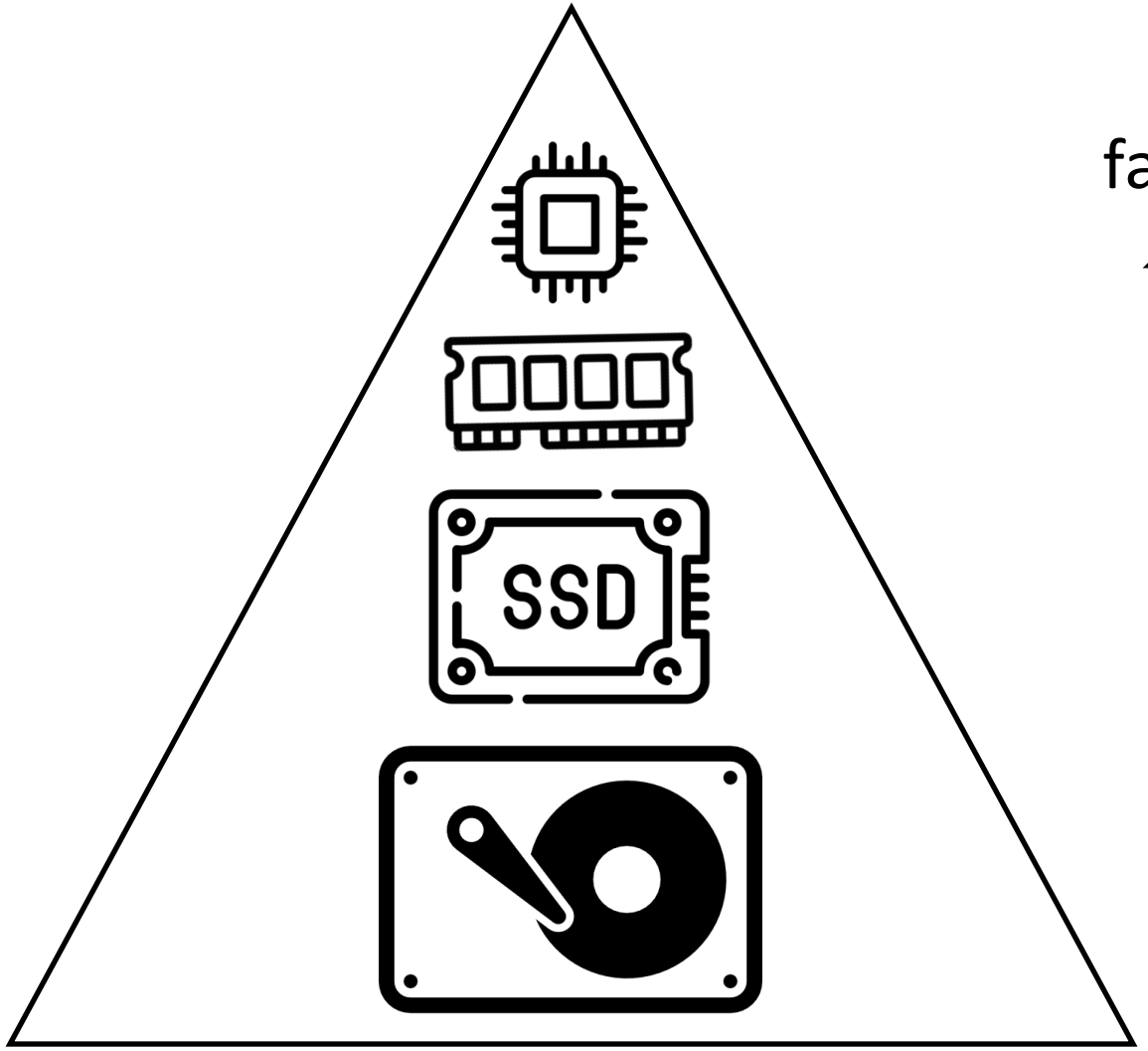
Manos Athanassoulis

https://bu-disc.github.io/CS561/

Data Systems & Hardware



Data Systems



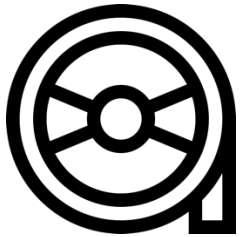
Memory Hierarchy

faster

larger

Hardware Trends

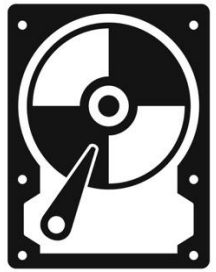
Evolution of Storage Technology



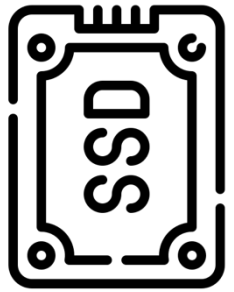
Tape

Floppy

CD

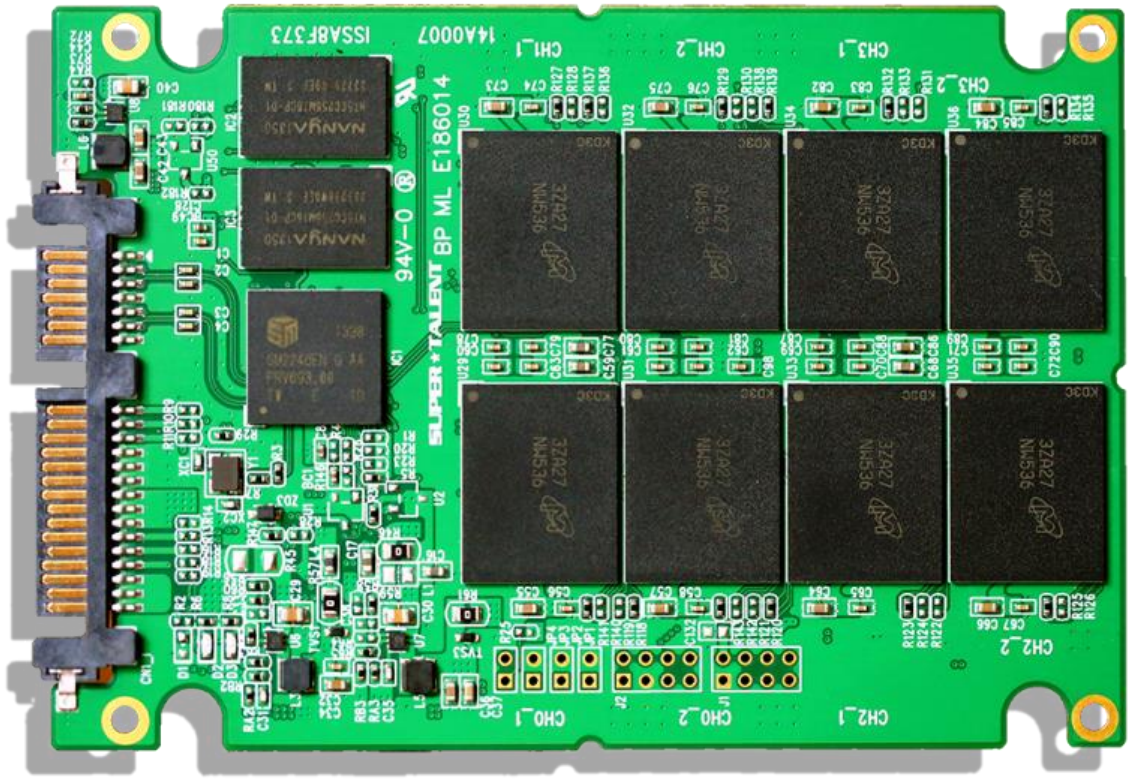


HDD



SSD

Solid State Drives



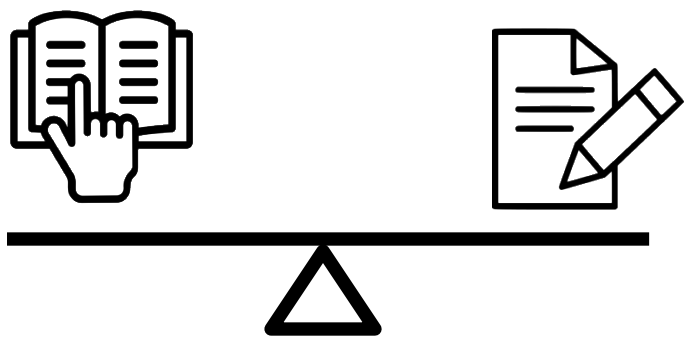
electronic device

fast random access

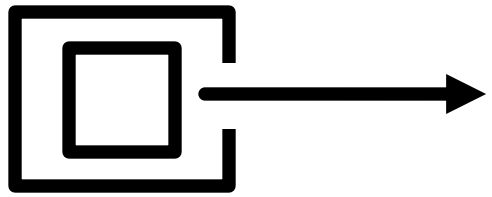
concurrent I/Os

write latency > read latency

HDD

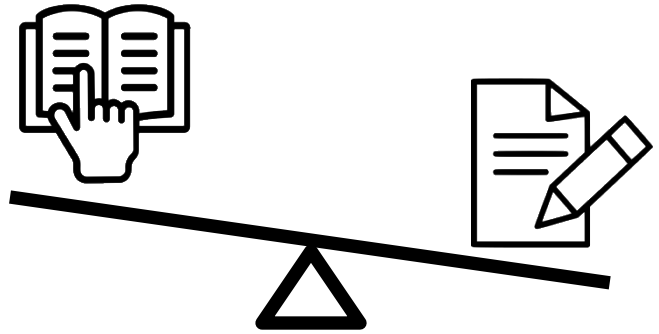


Symmetric cost for Read & Write

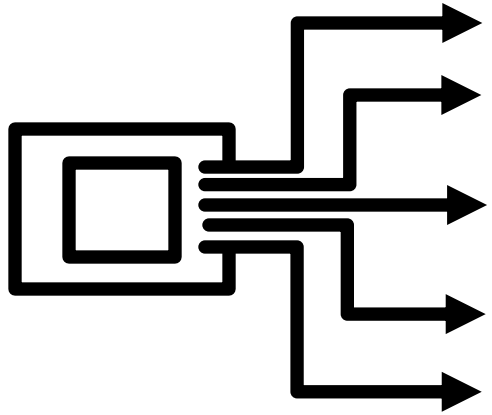


One I/O at a time

SSD

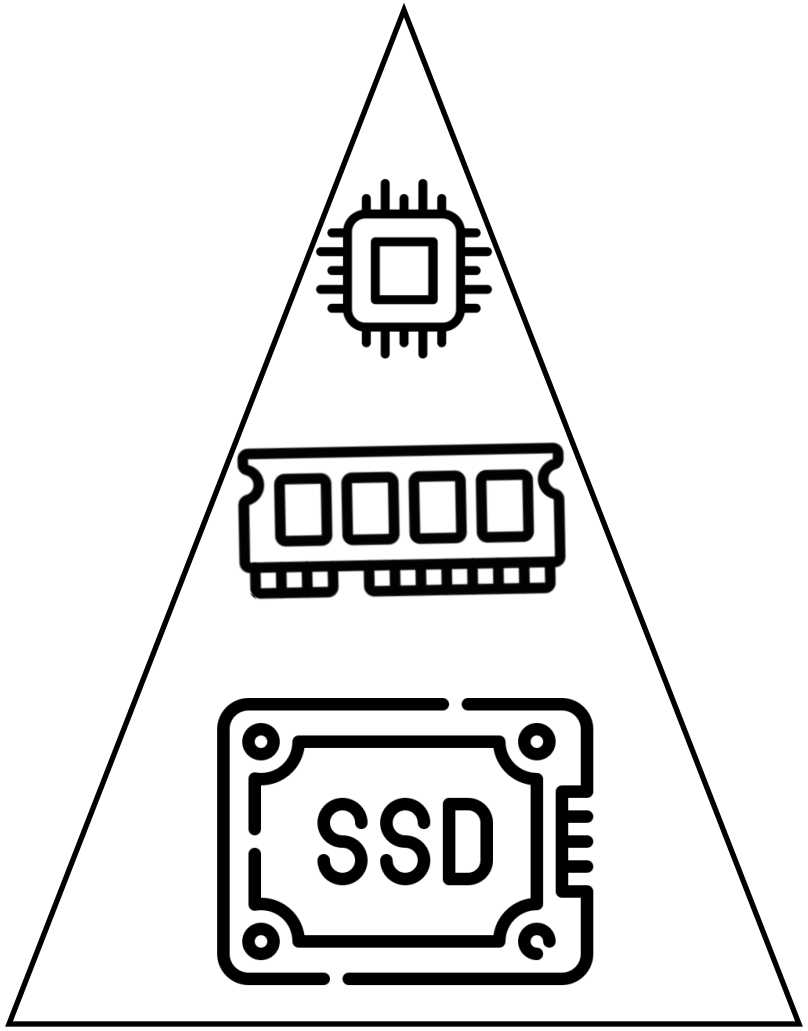


Read/Write Asymmetry (α)



Concurrency (k)

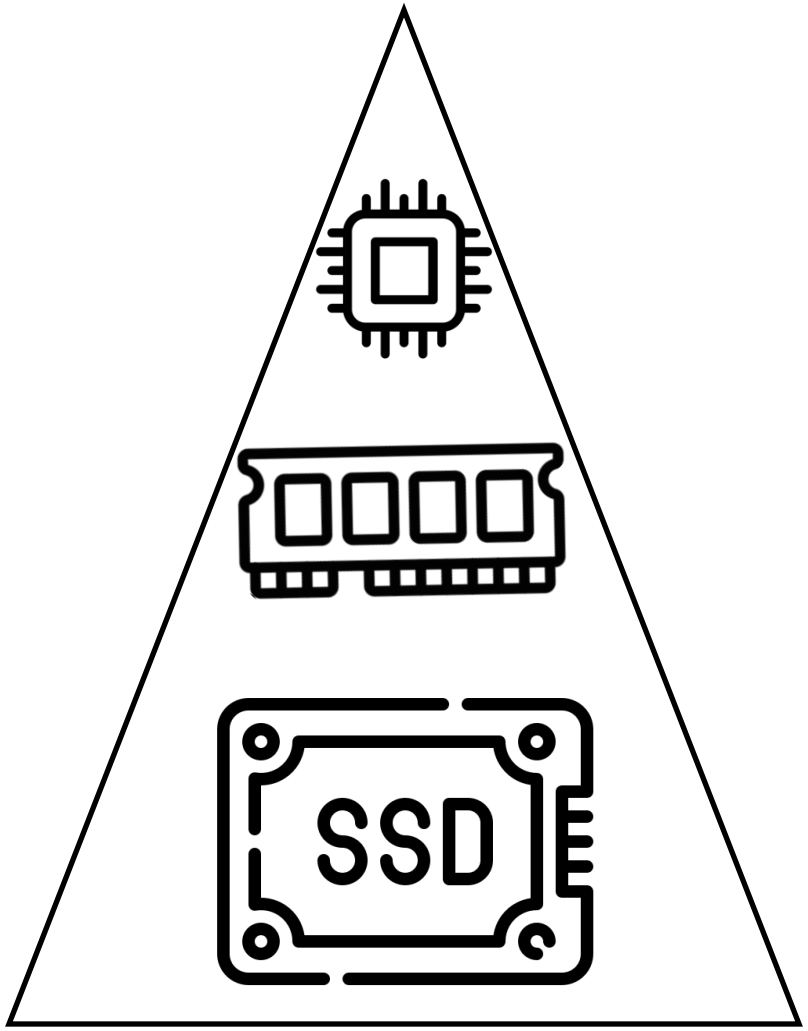
Goal: Developing Hardware-Aware Data Systems



Tailor Data Systems
for **SSD Asymmetry**
& **Concurrency**

Need for an I/O Model [**CIDR '21**]
PIO Model [**DaMoN@SIGMOD '21**]
ACE Bufferpool [**IEEE ICDE '23**]
CAVE Graph Engine [**SIGMOD '24**]
SSD-Aware Systems [**IEEE ICDE '24**]

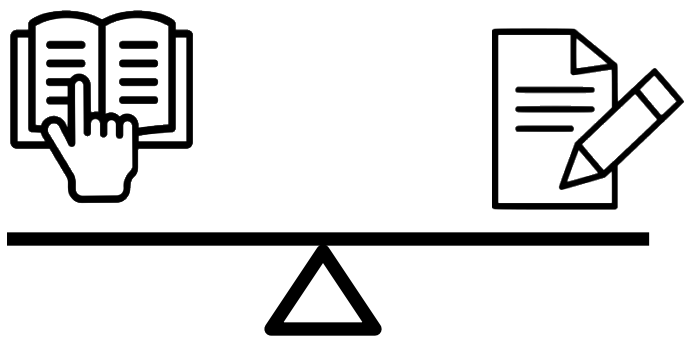
Goal: Developing Hardware-Aware Data Systems



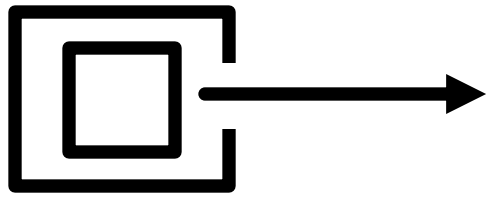
Tailor Data Systems
for **SSD Asymmetry**
& **Concurrency**

Need for an I/O Model [**CIDR '21**]
PIO Model [**DaMoN@SIGMOD '21**]
ACE Bufferpool [**IEEE ICDE '23**]
CAVE Graph Engine [**SIGMOD '24**]
SSD-Aware Systems [**IEEE ICDE '24**]

HDD

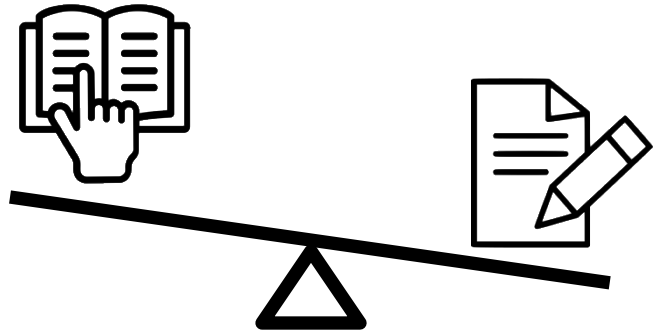


Symmetric cost for Read & Write

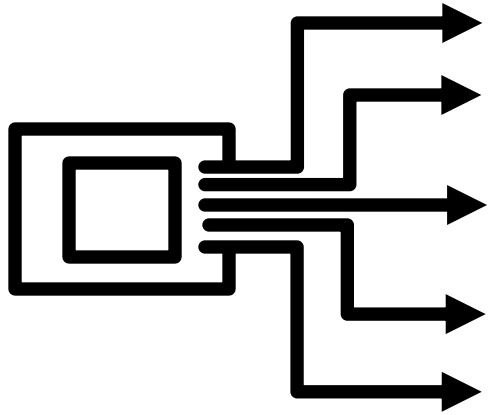


One I/O at a time

SSD

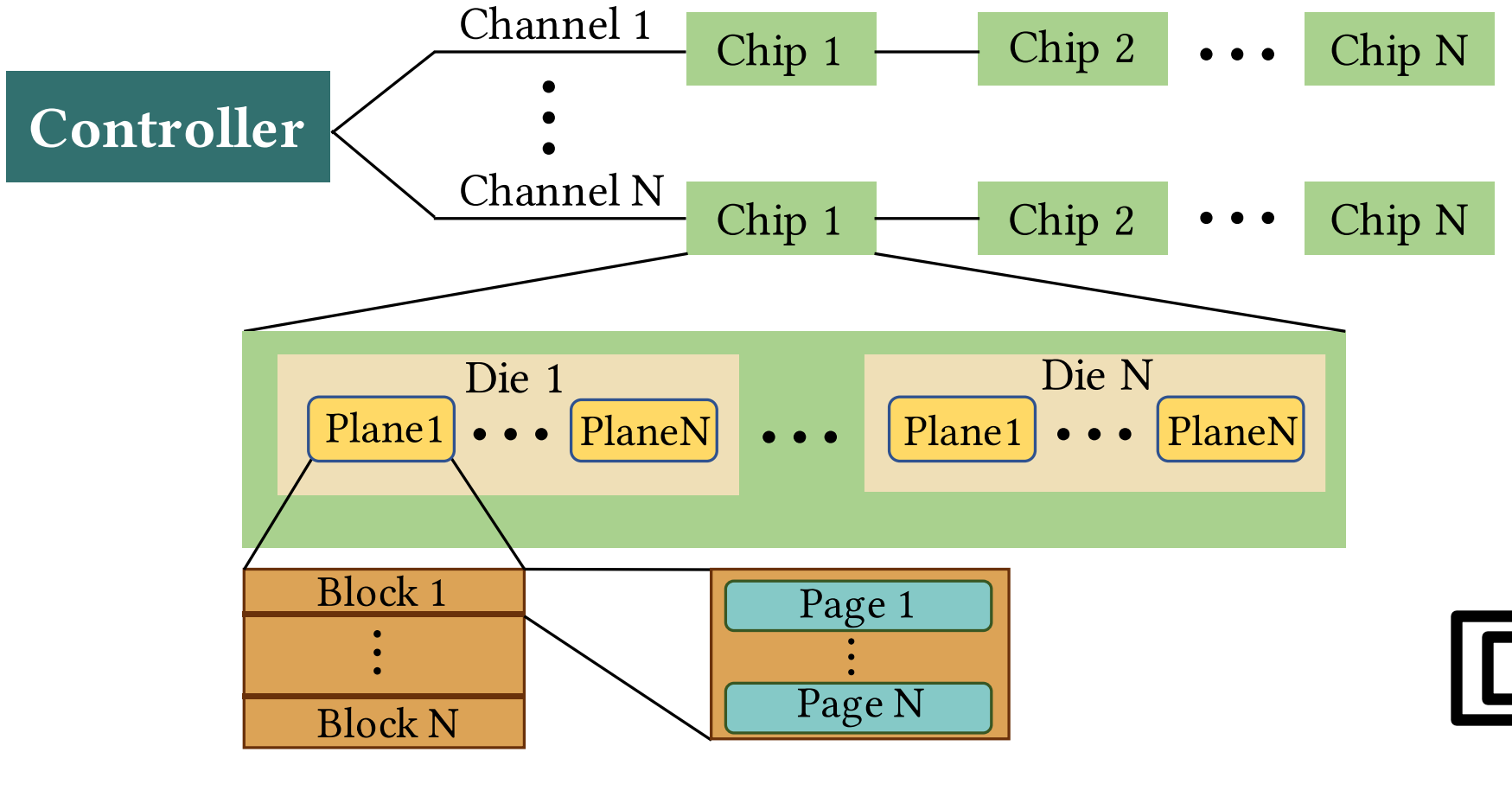


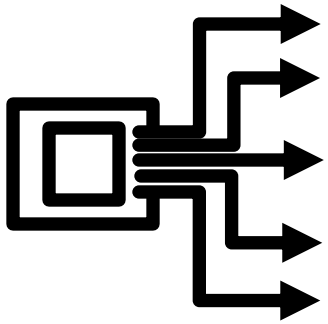
Read/Write Asymmetry (α)



Concurrency (k)

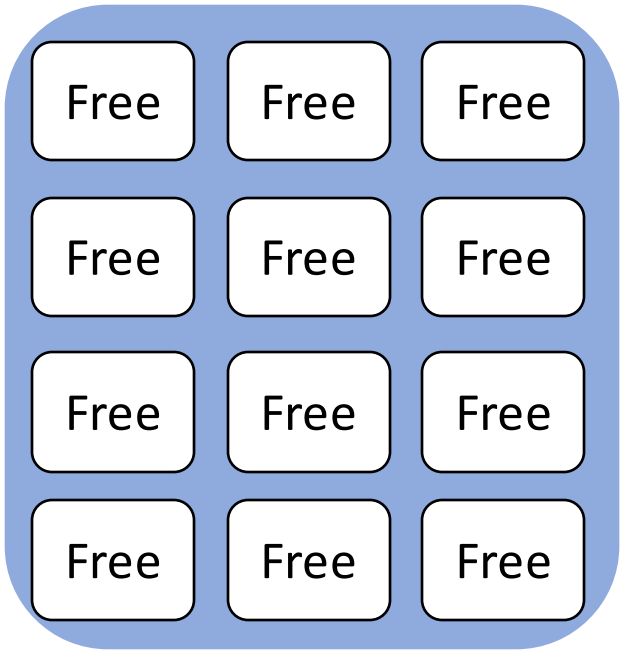
SSD Concurrency



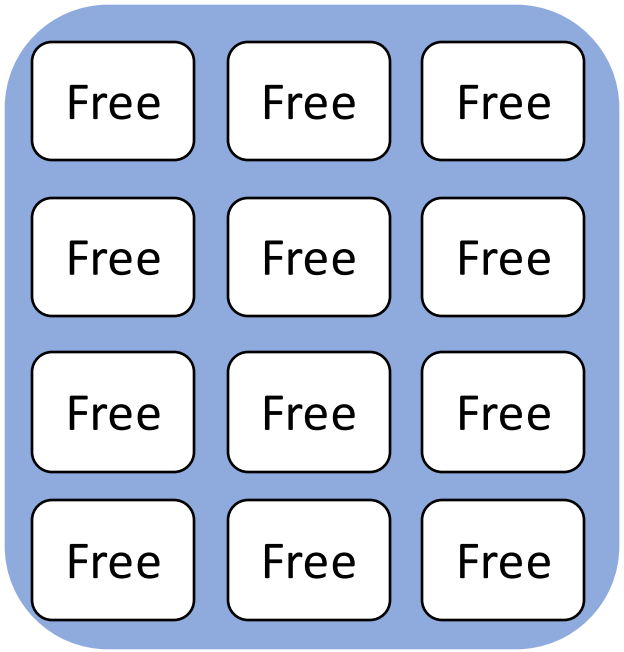


Parallelism at different levels (channel, chip, die, plane block, page)

Writes in SSD

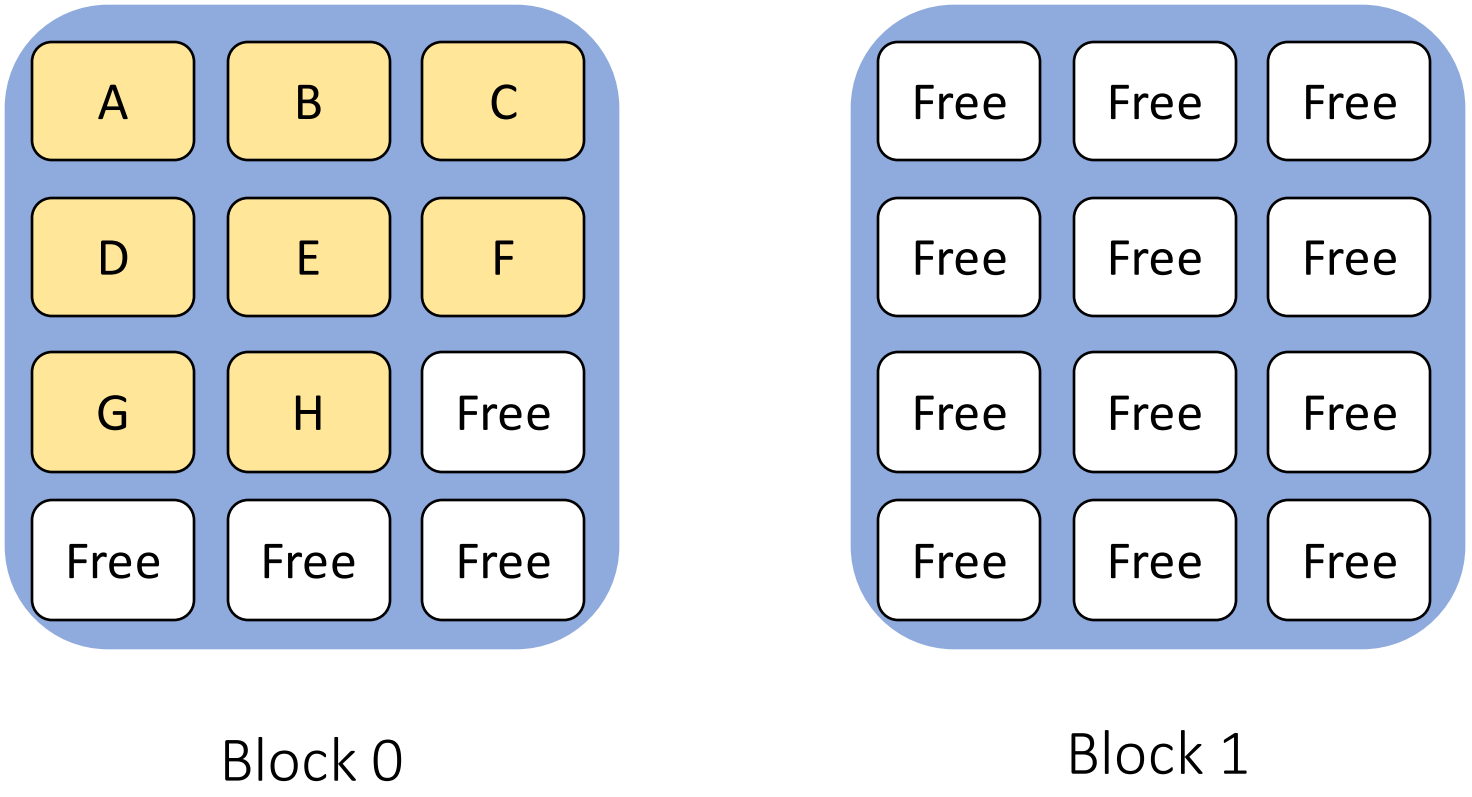


Block 0



Block 1

Writes in SSD



Block 0

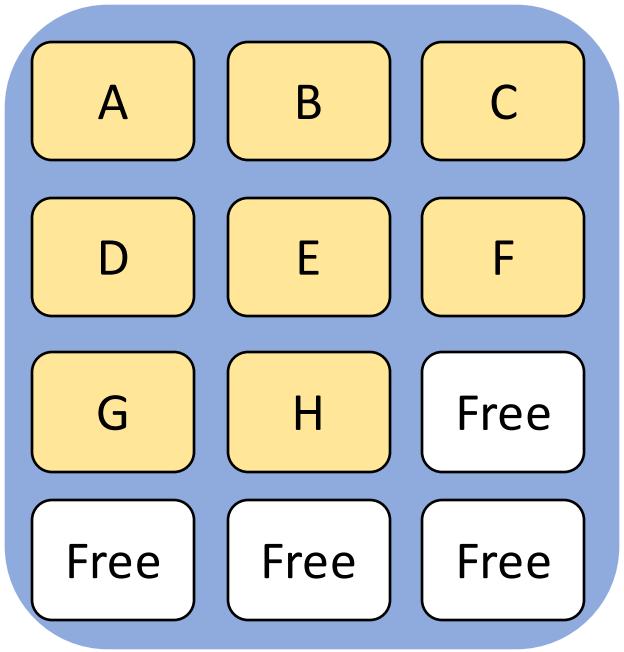
Block 1

Writing in a free page isn’t costly!

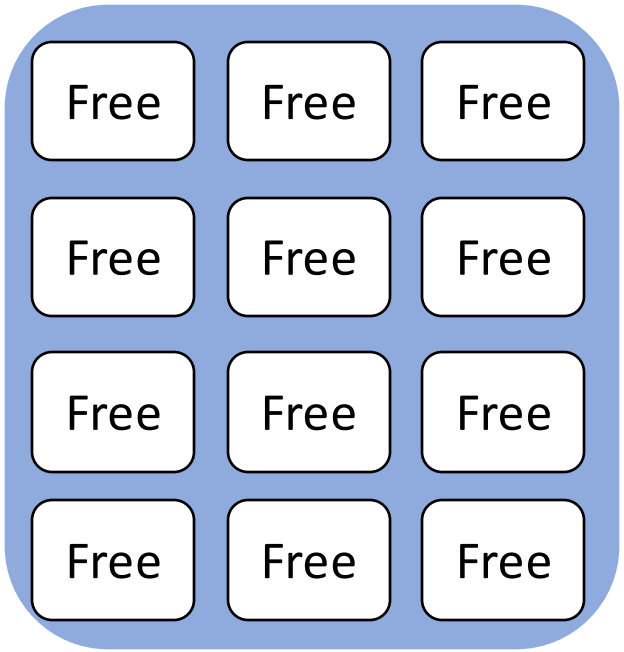
Writes in SSD

Update

A, B, C, D



Block 0

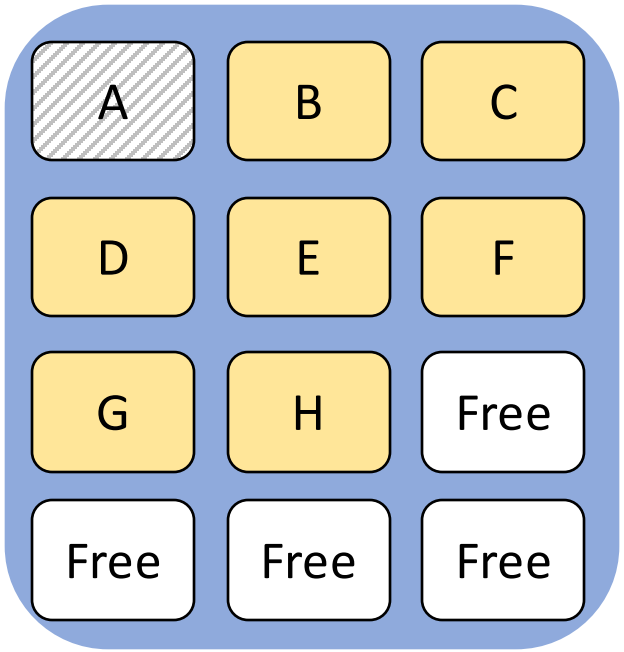


Block 1

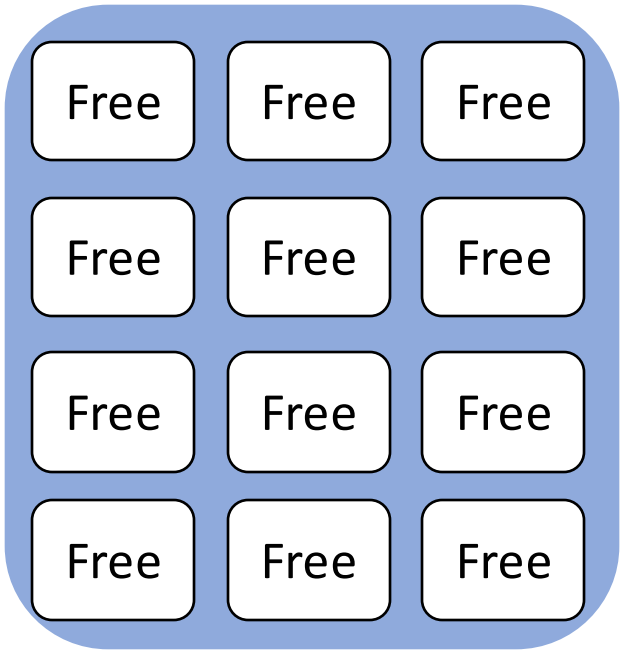
Writes in SSD

Update

A, B, C, D



Block 0

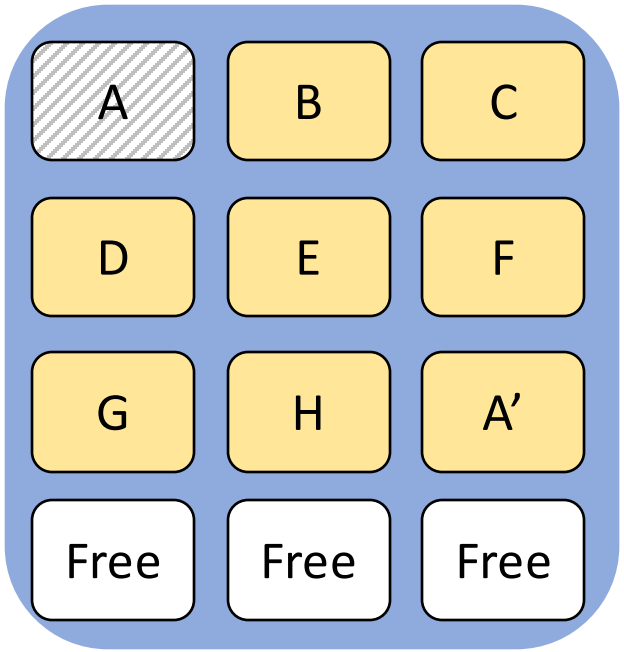


Block 1

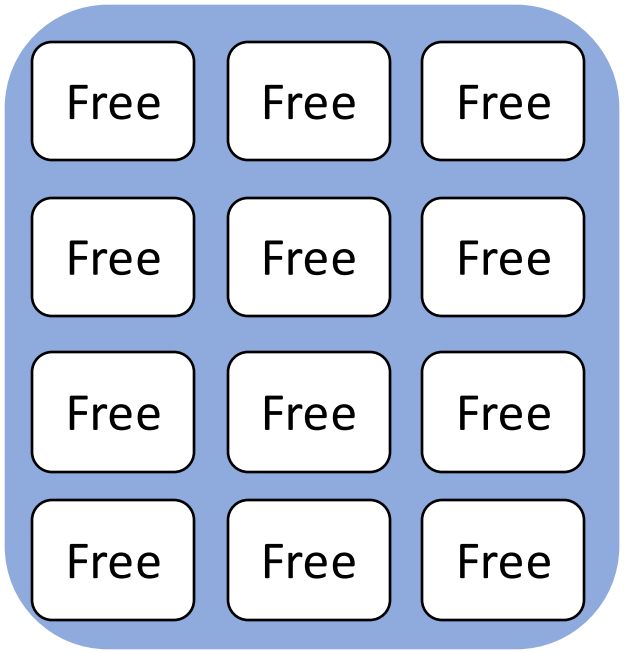
Writes in SSD

Update

A, B, C, D



Block 0

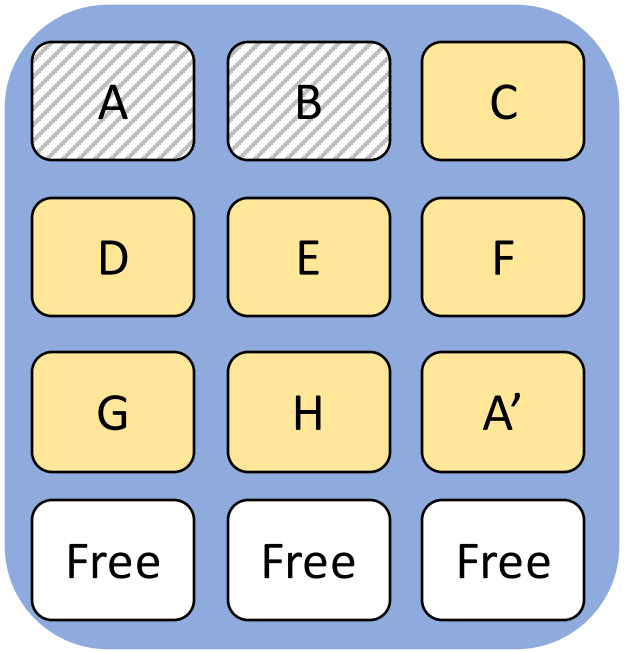


Block 1

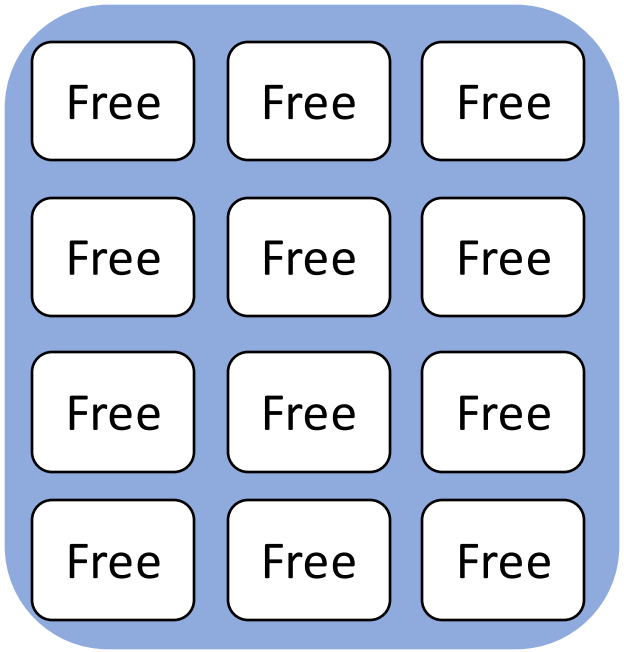
Writes in SSD

Update

A, **B**, C, D



Block 0

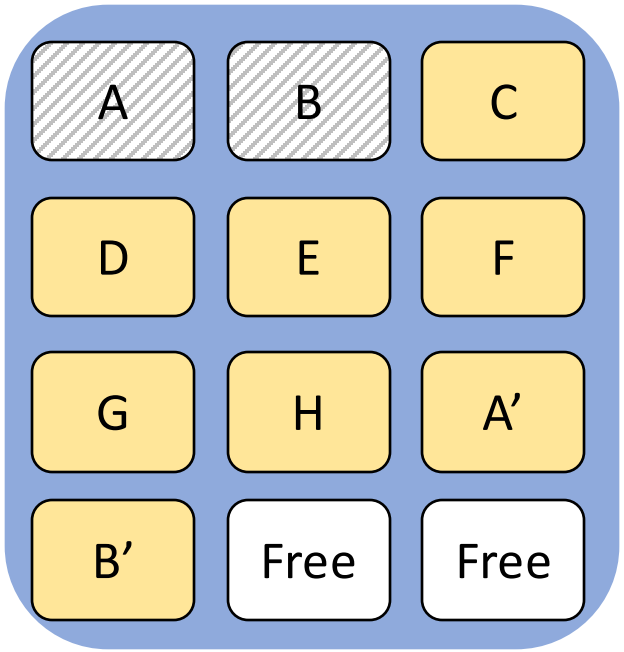


Block 1

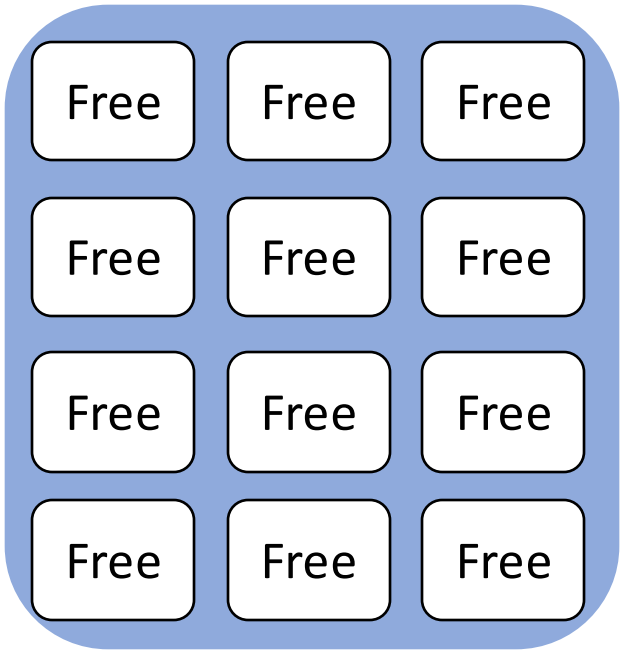
Writes in SSD

Update

A, **B**, C, D



Block 0

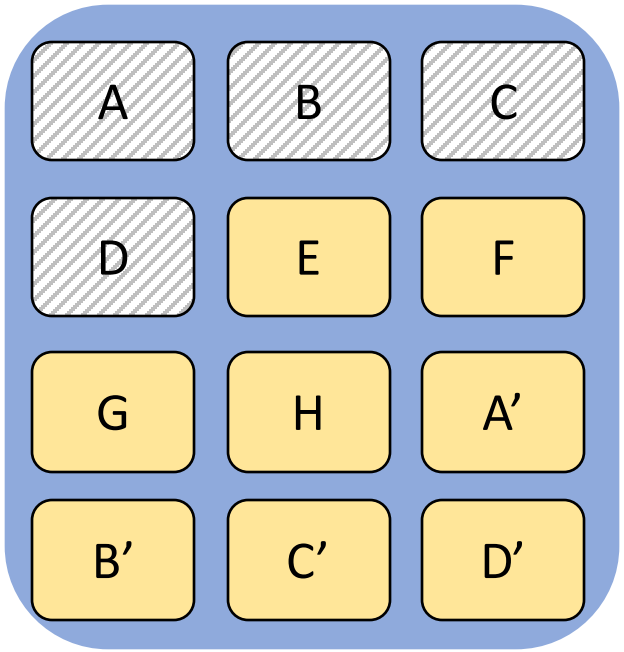


Block 1

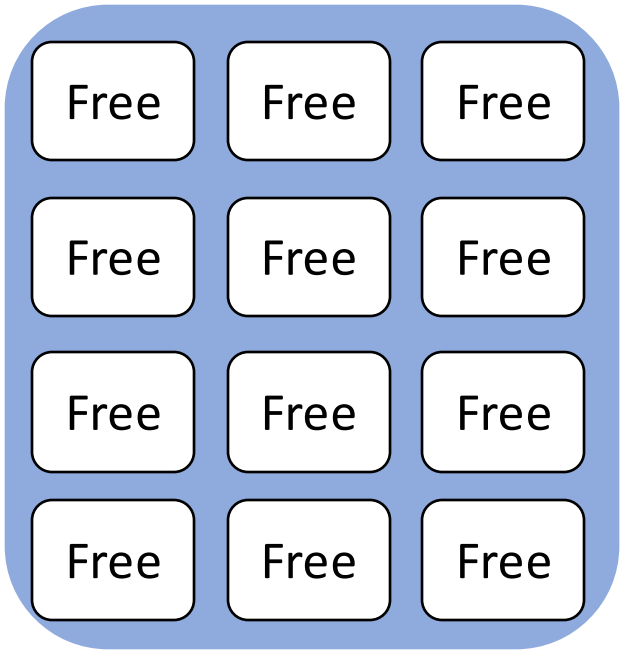
Writes in SSD

Update

A, B, C, D



Block 0

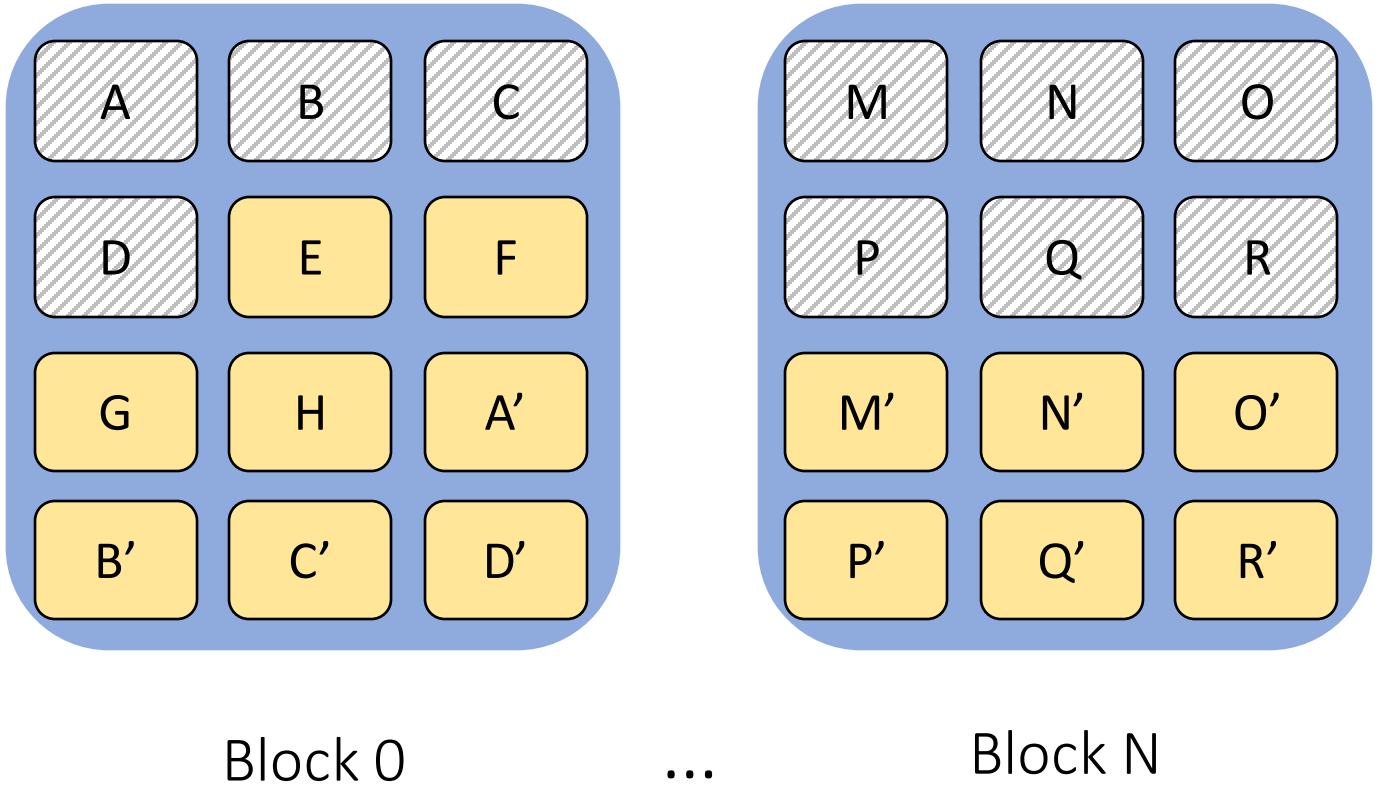


Block 1

Not all updates are costly!

Writes in SSD

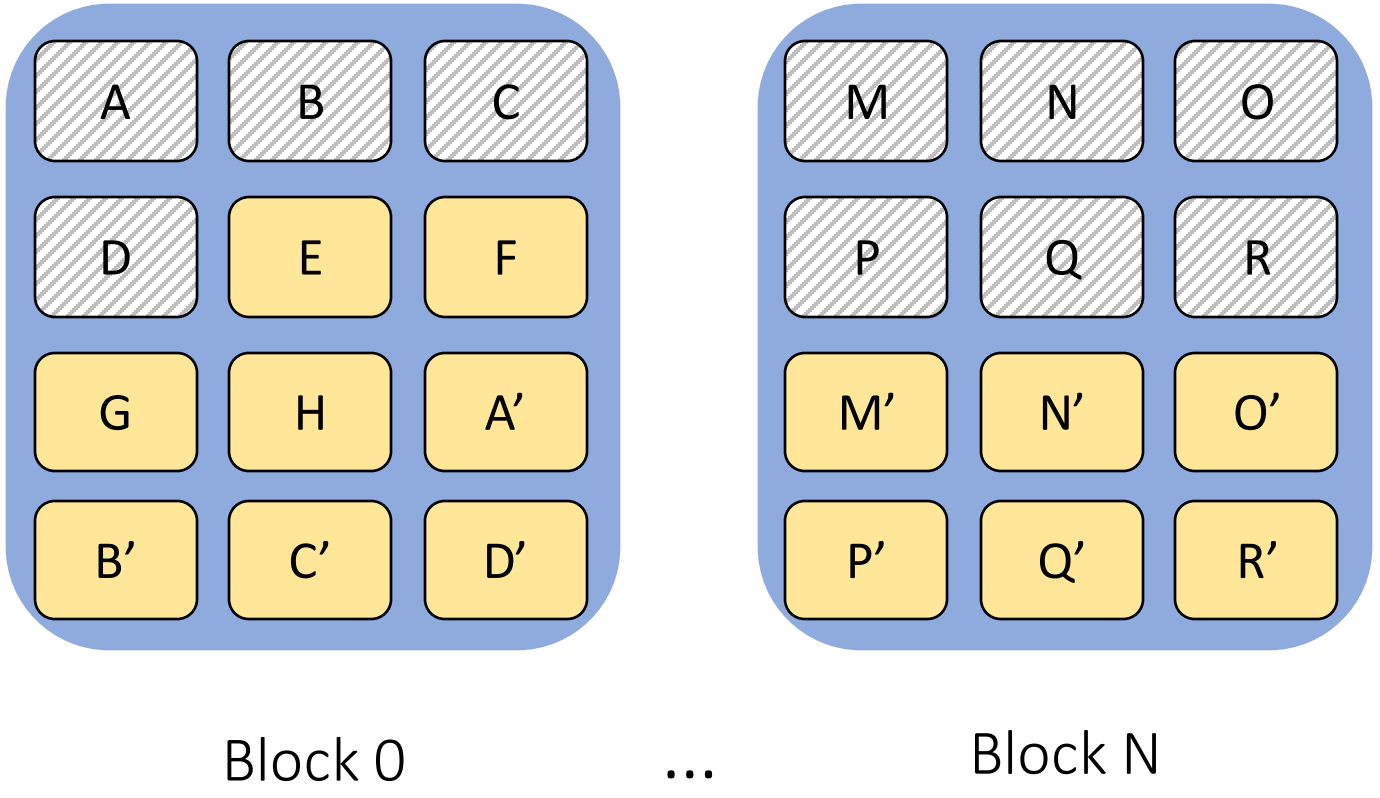
What if there is no space?



Writes in SSD

What if there is no space?

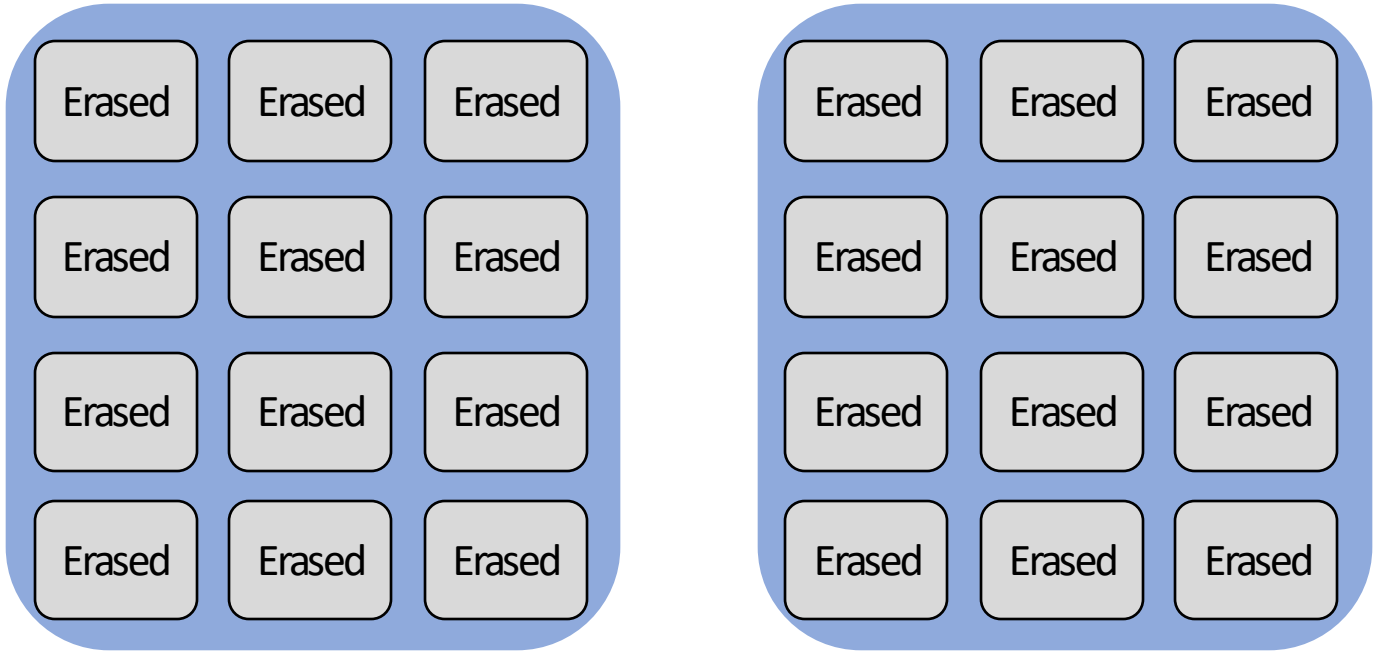
Garbage Collection!



Writes in SSD

What if there is no space?

Garbage Collection!



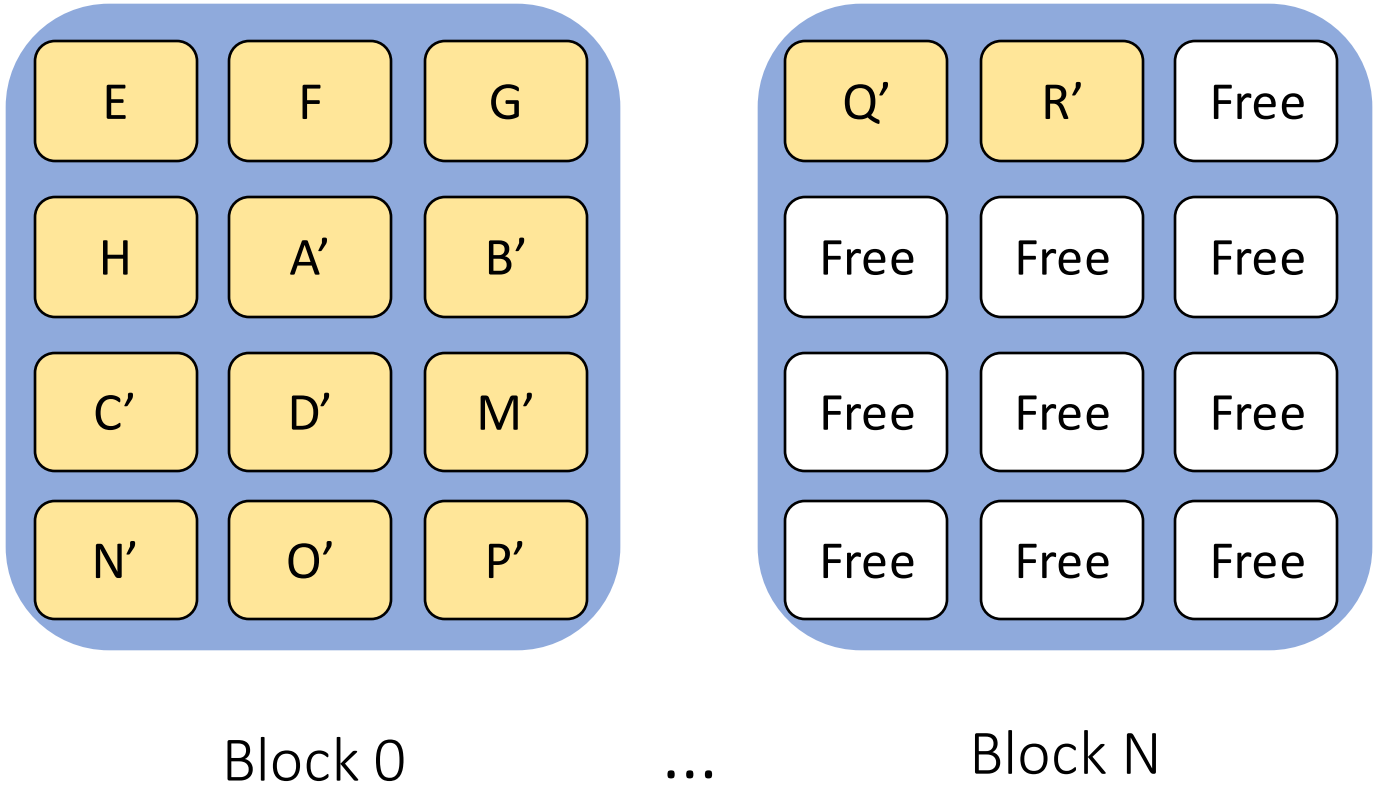
Valid pages:

E	F	G	H	A’	B’	C’	D’	M’	N’	O’	P’	Q’	R’

Writes in SSD

What if there is no space?

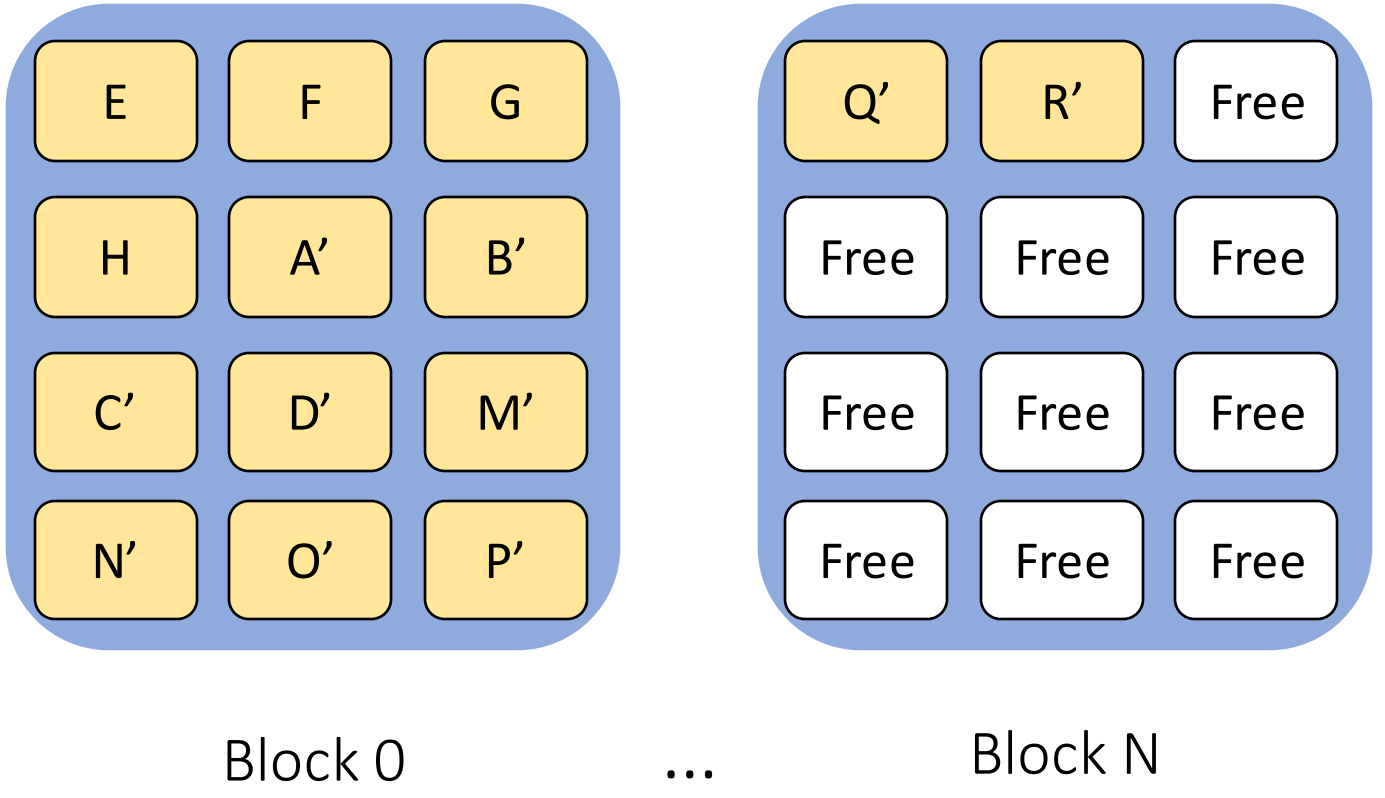
Garbage Collection!



Read/Write Asymmetry in SSD

What if there is no space?

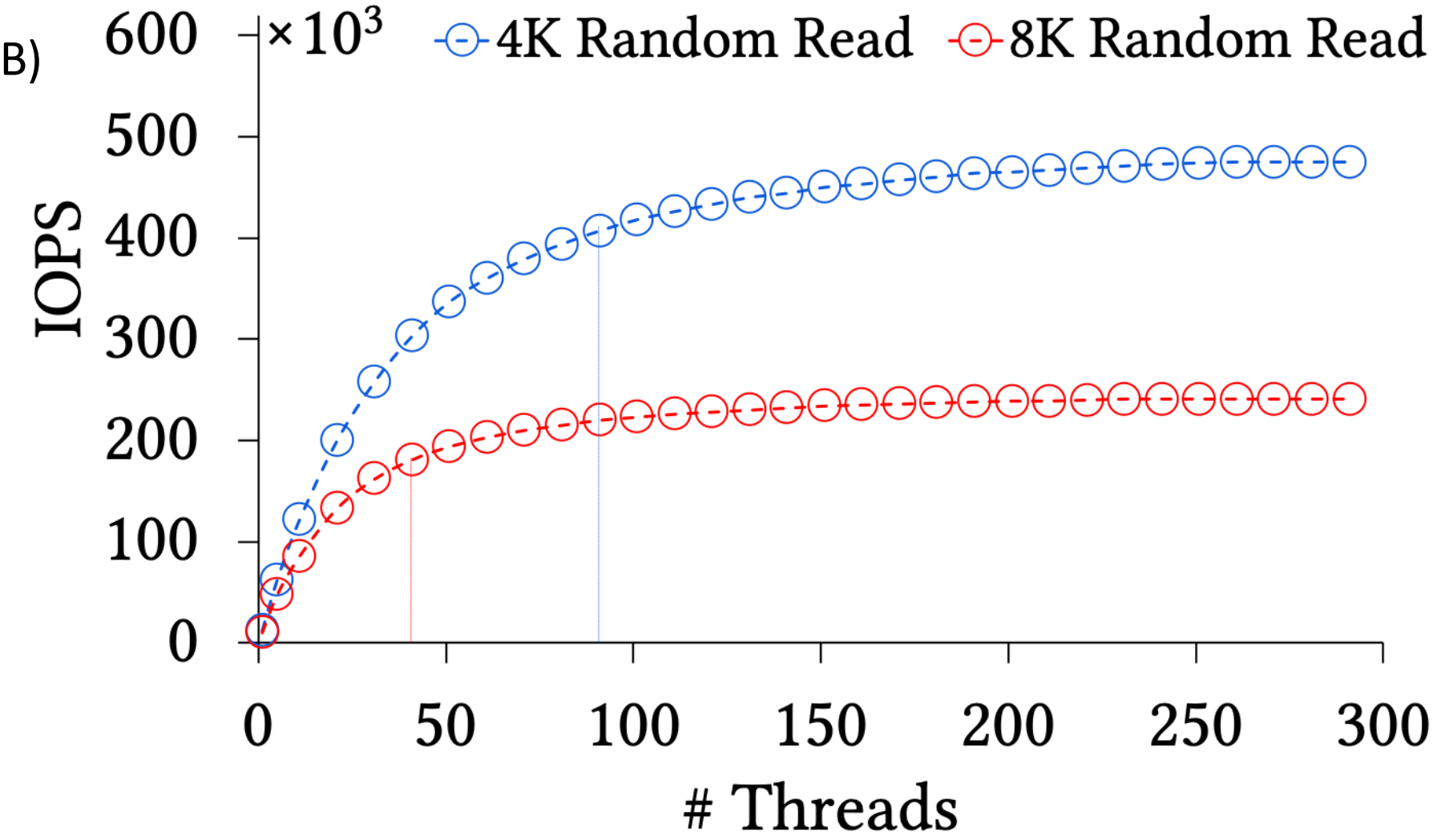
Garbage Collection!



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

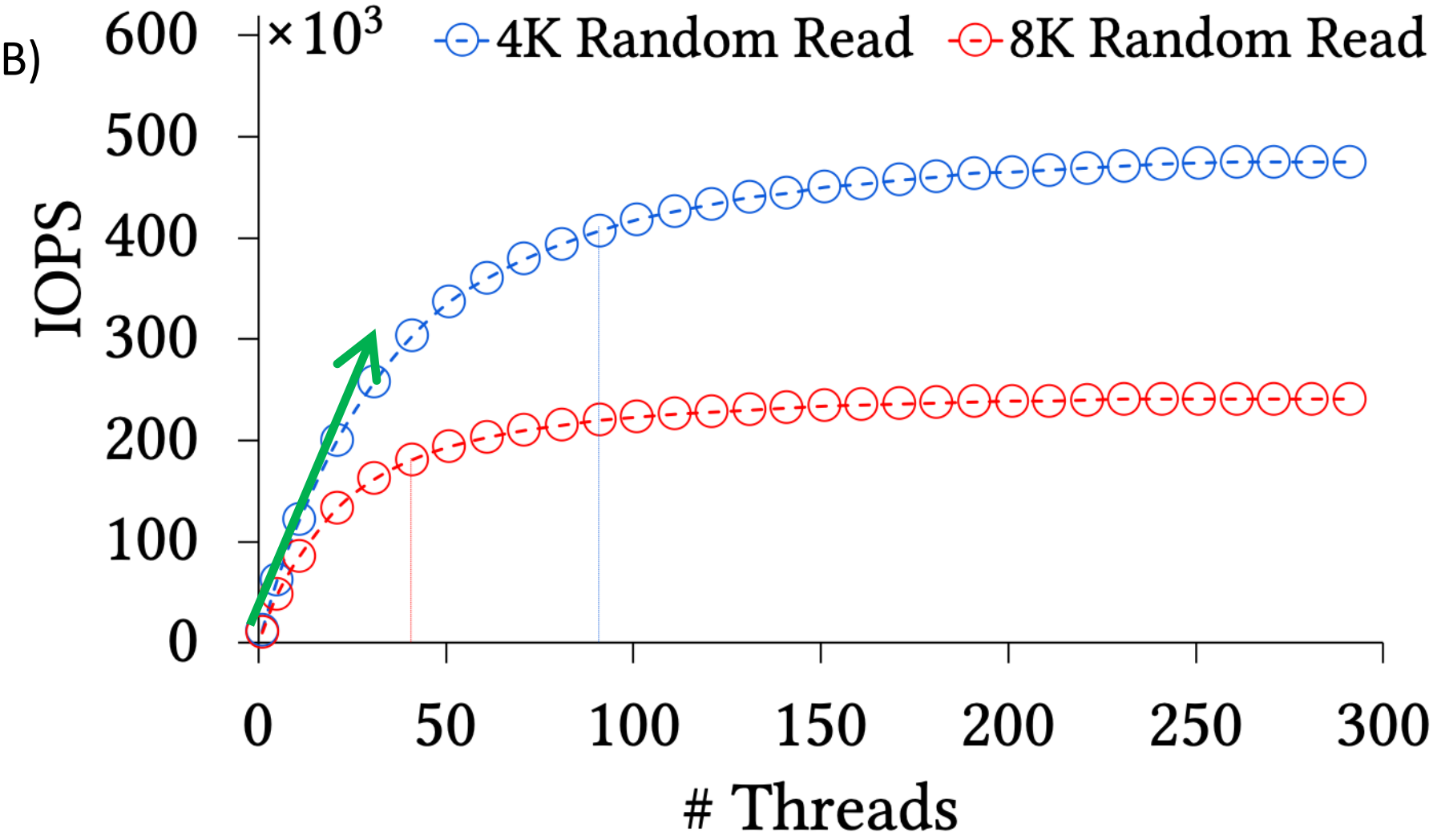
Quantifying Asymmetry & Concurrency

Device
PCIe SSD - P4510 (1TB)



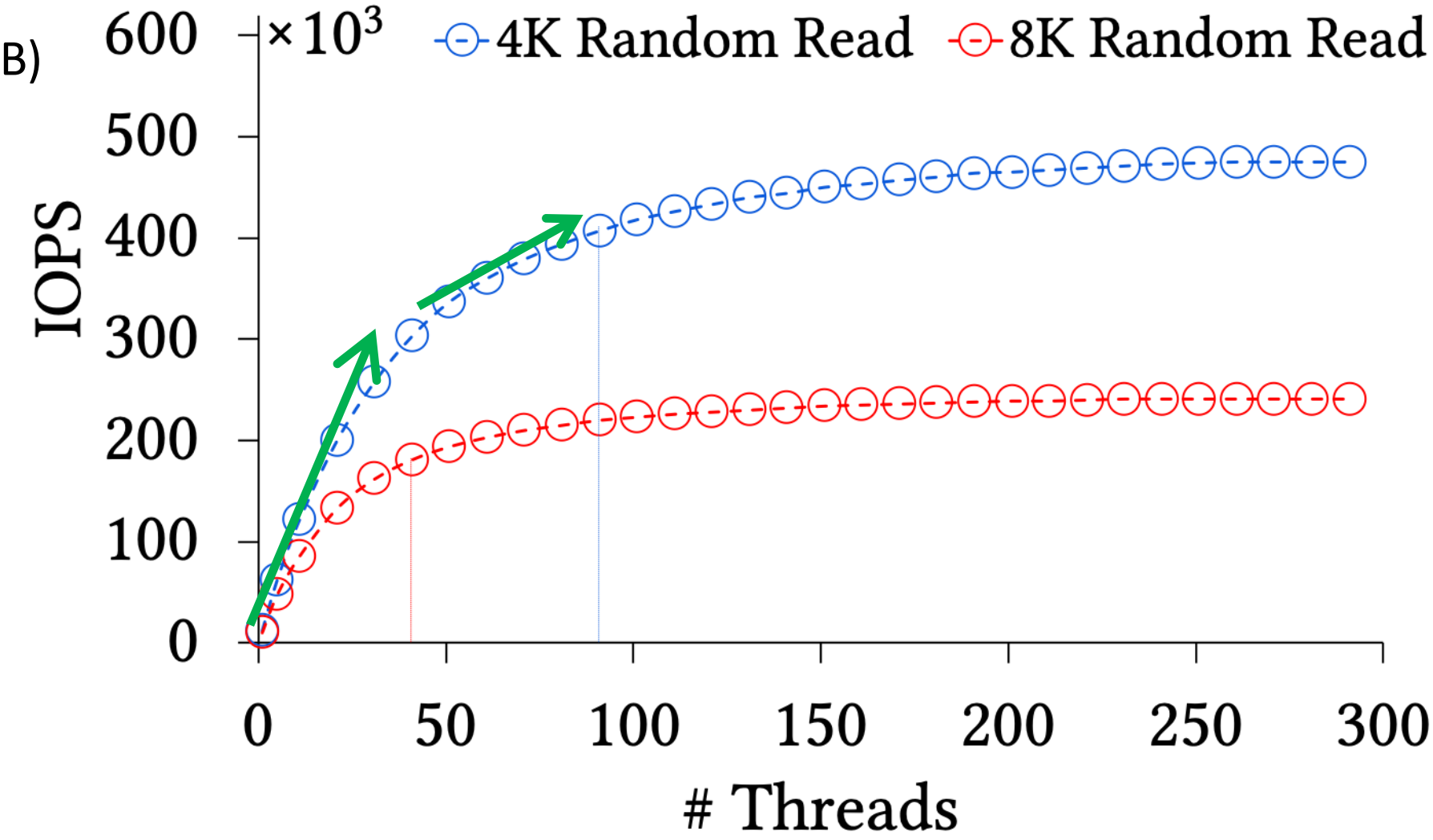
Quantifying Asymmetry & Concurrency

Device
PCIe SSD - P4510 (1TB)



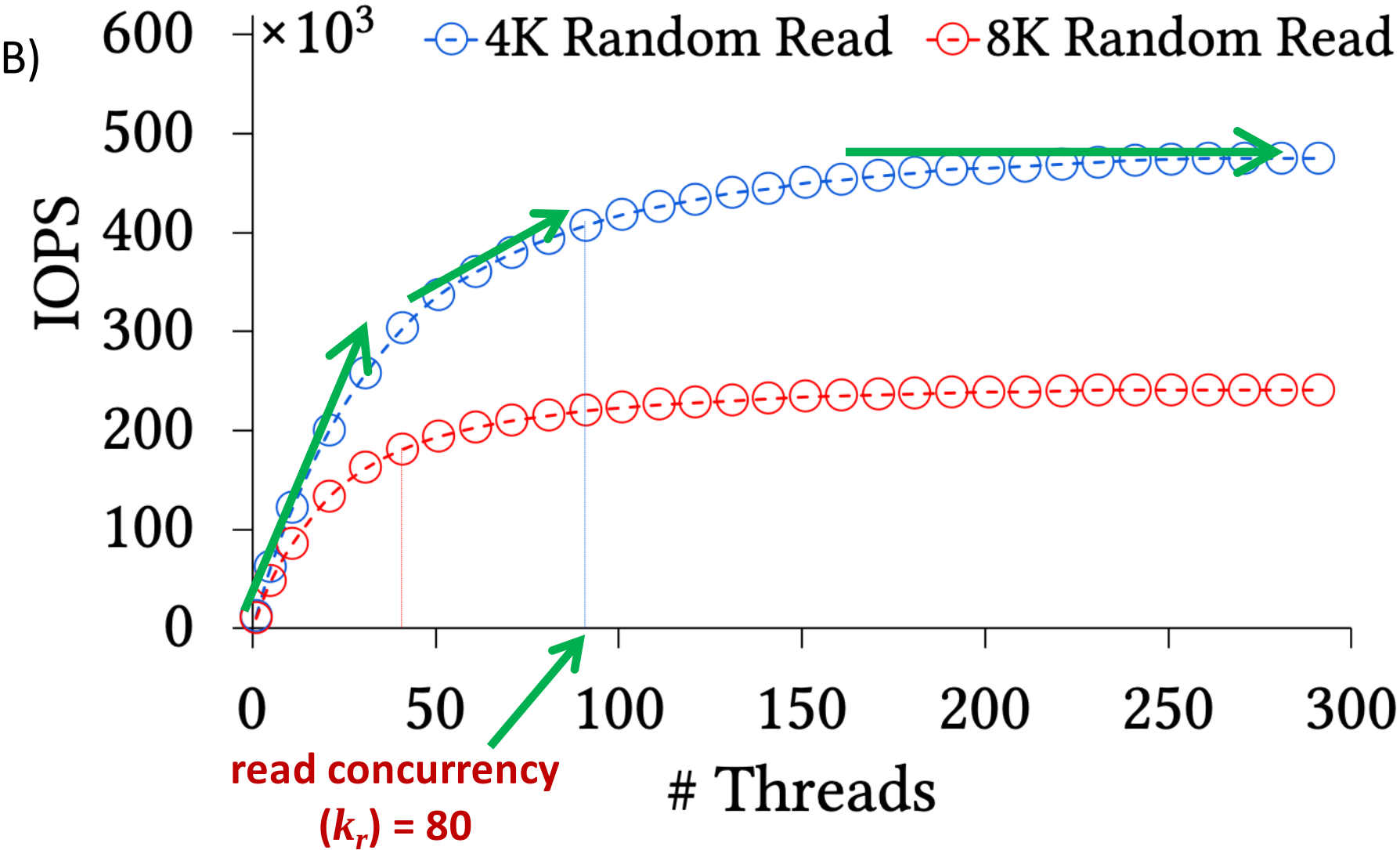
Quantifying Asymmetry & Concurrency

Device
PCIe SSD - P4510 (1TB)



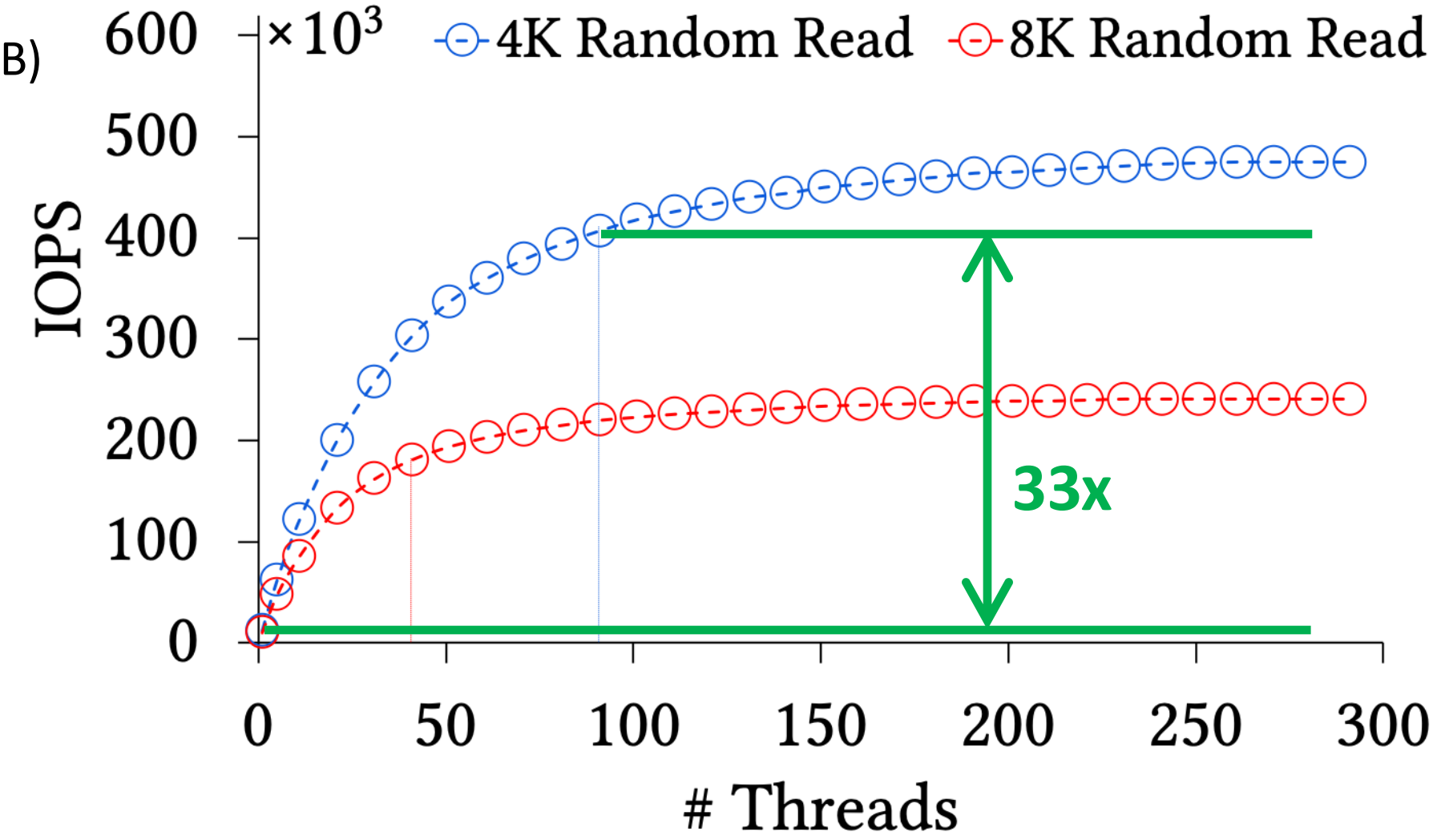
Quantifying Asymmetry & Concurrency

Device
PCIe SSD - P4510 (1TB)



Quantifying Asymmetry & Concurrency

Device
PCIe SSD - P4510 (1TB)



Quantifying Asymmetry & Concurrency

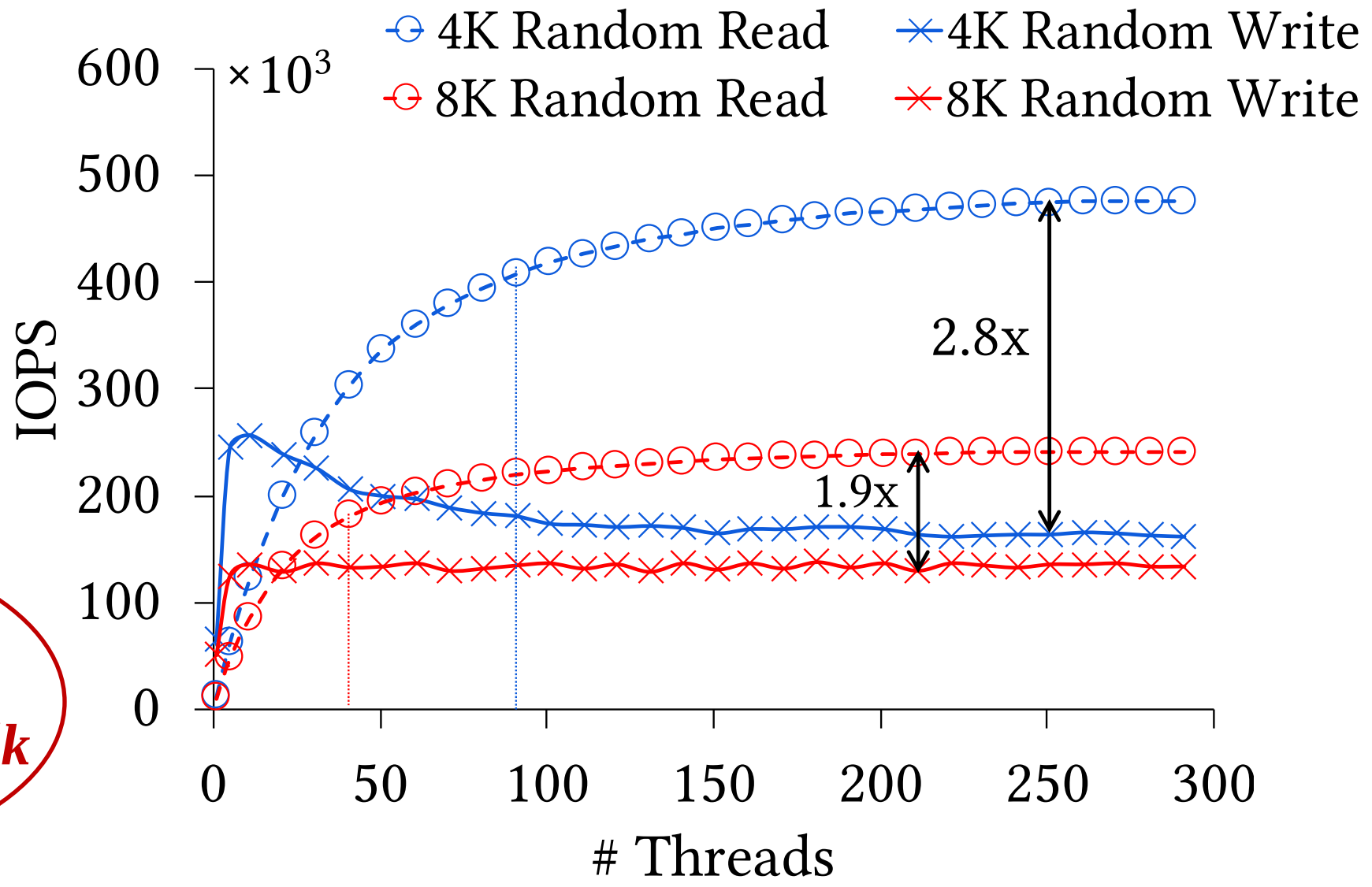
Device
PCIe SSD - P4510 (1TB)

For 4K random read,

Asymmetry: 2.8

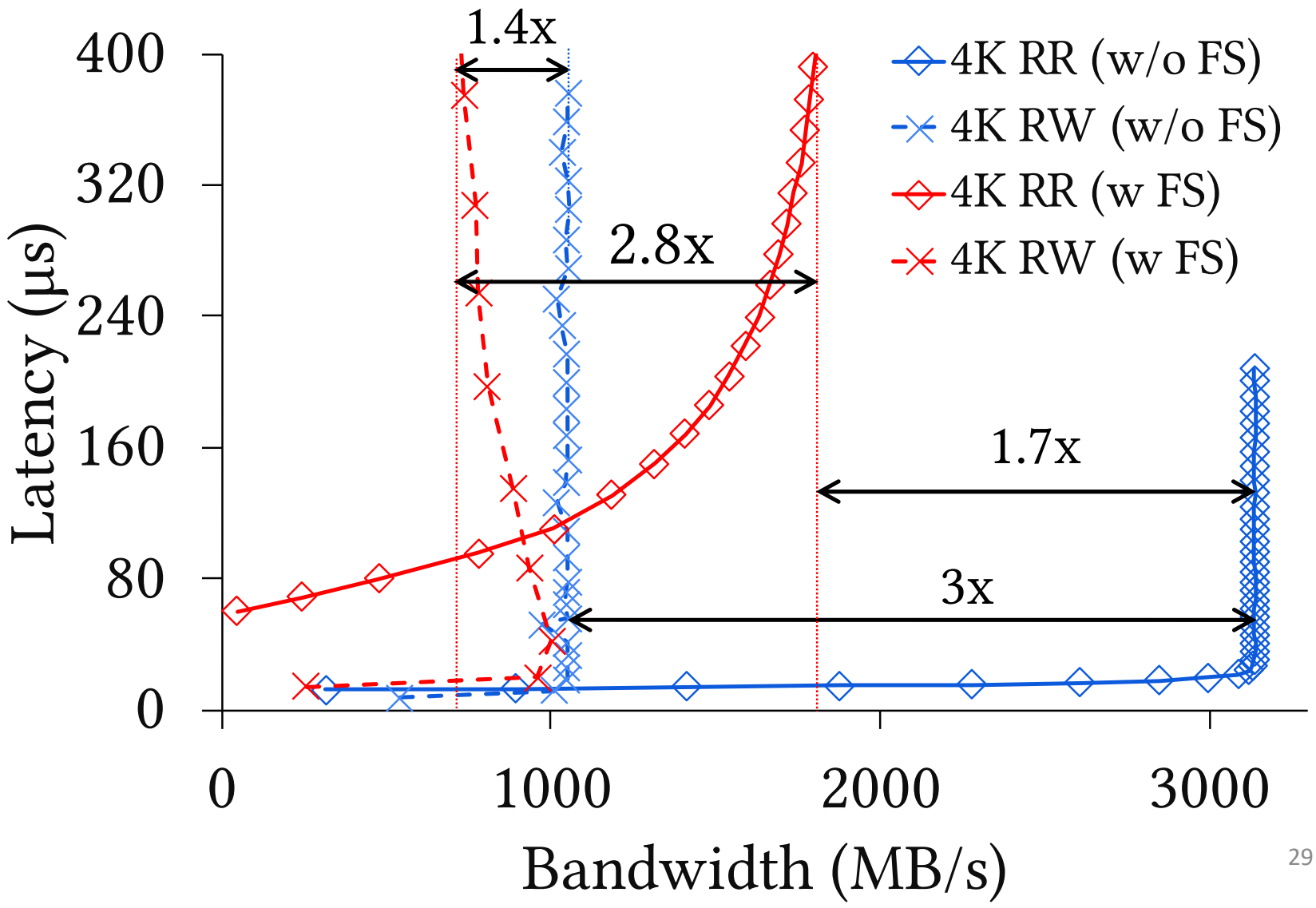
Concurrency: 80

Yet, systems are not always tailored for α/k



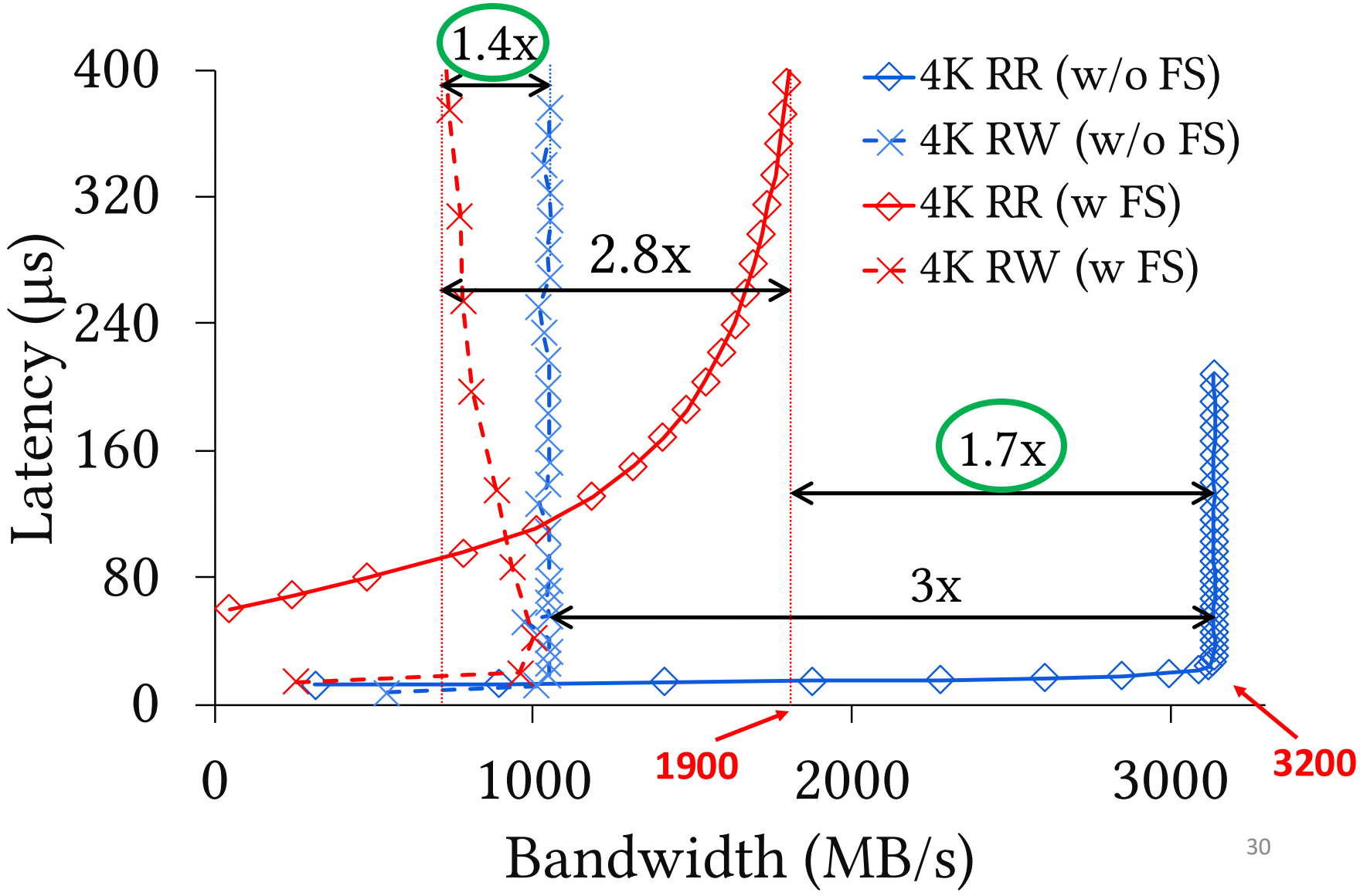
Impact of the File System

Device
PCIe SSD - P4510 (1TB)



Impact of the File System

Device
PCIe SSD - P4510 (1TB)



Impact of the File System

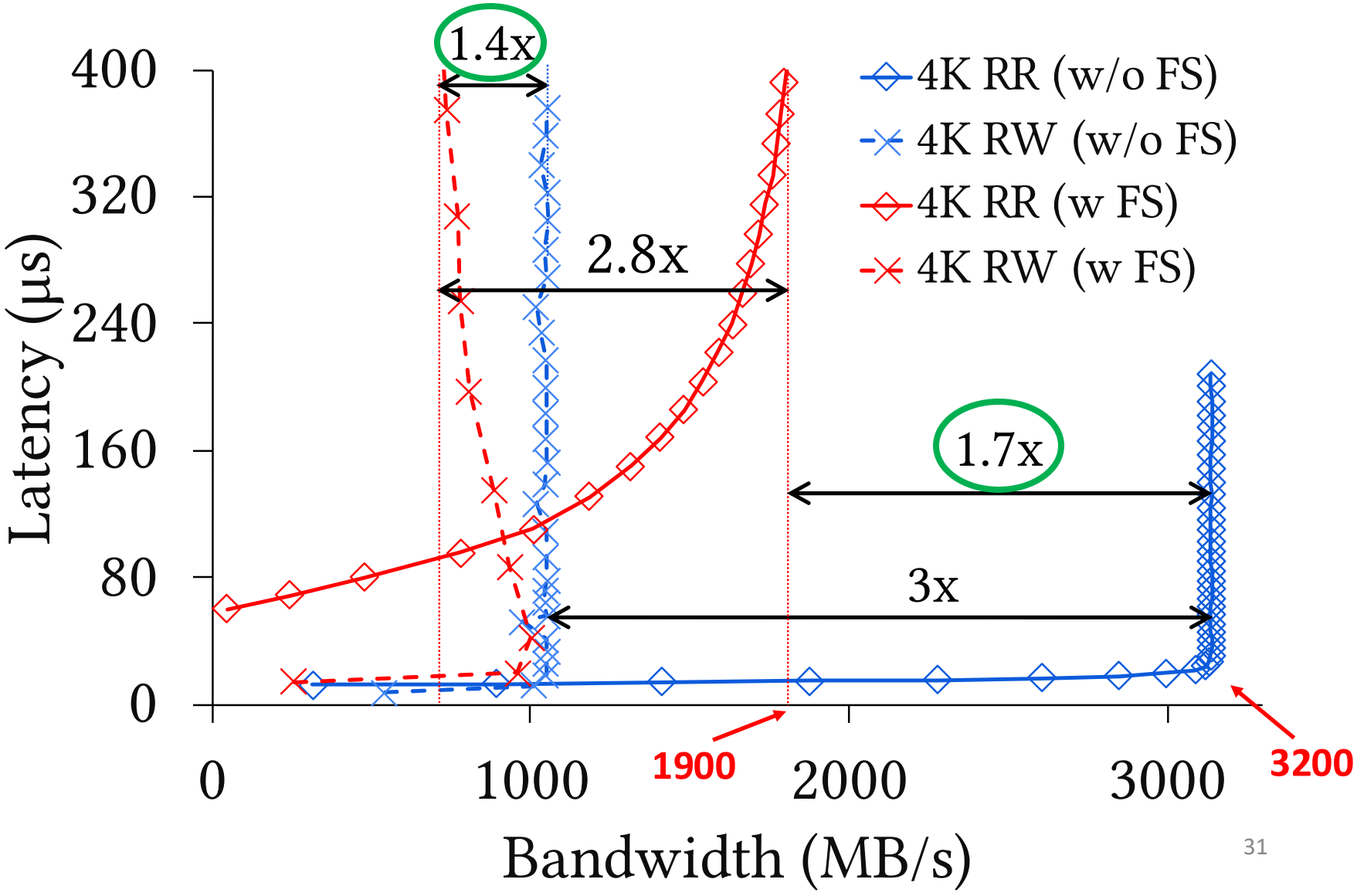
Device
PCIe SSD - P4510 (1TB)

FS affects α and k.

Much stable &

increased bandwidth

w/o the file system.

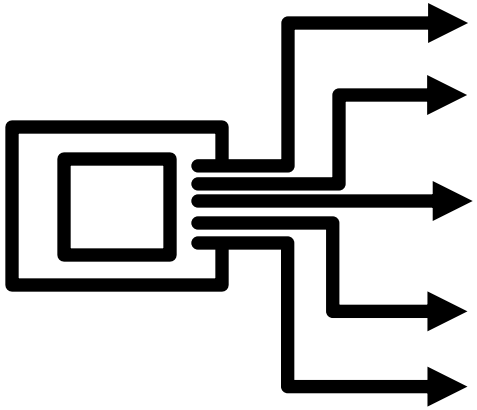


Empirical Asymmetry and Concurrency

Devices	4KB			8KB		
	α	k_r	k_w	α	k_r	k_w
Optane SSD	1.1	6	5	1.0	4	4
PCIe SSD (with FS)	2.8	80	8	1.9	40	7
PCIe SSD (w/o FS)	3.0	16	6	3.0	15	4
SATA SSD	1.5	25	9	1.3	21	5

Guidelines for System Design in SSDs

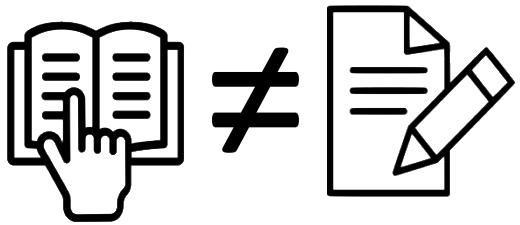
know thy device



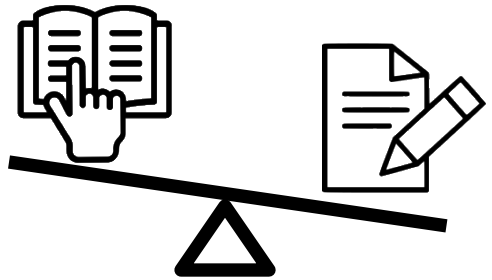
exploit k_r and k_w (with care)

read concurrency

write concurrency

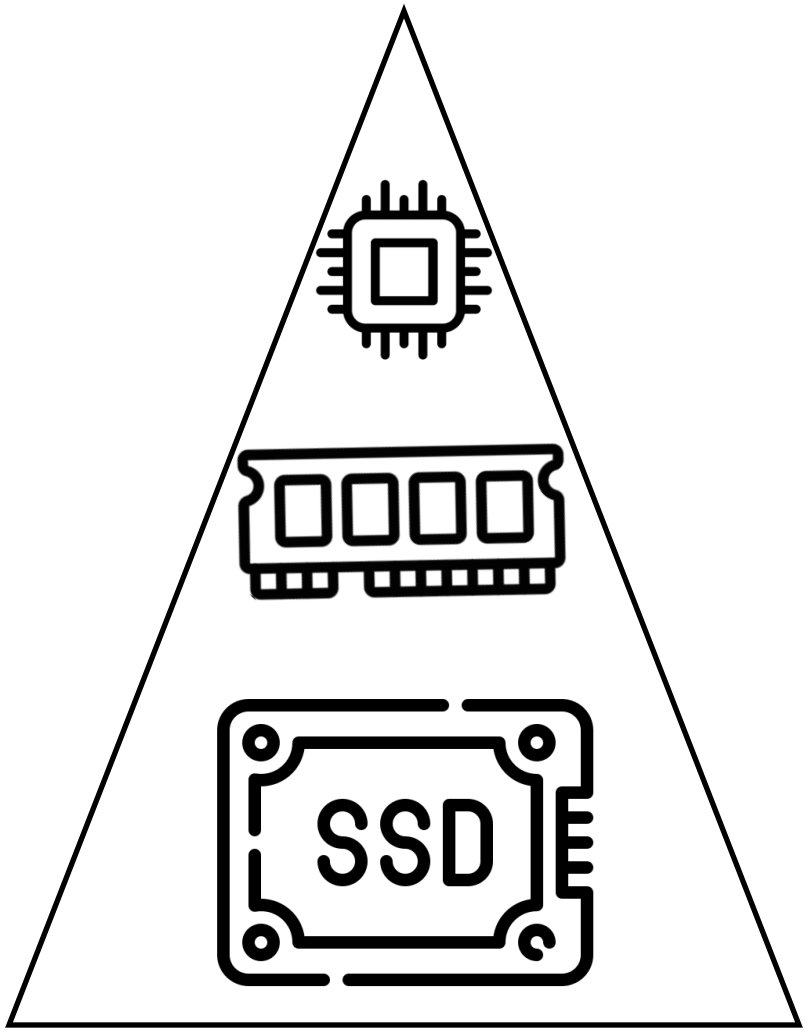


treat read and write differently



asymmetry (α) controls performance

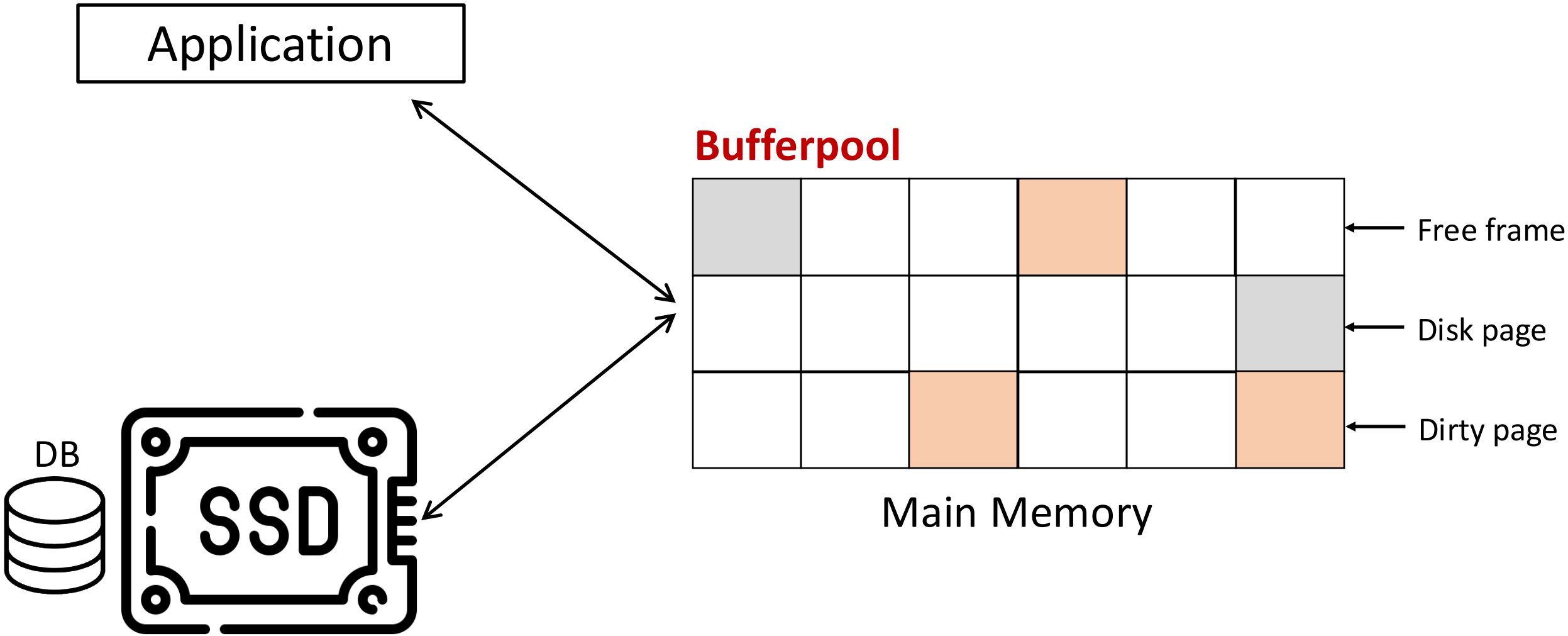
Goal: Developing Hardware-Aware Data Systems



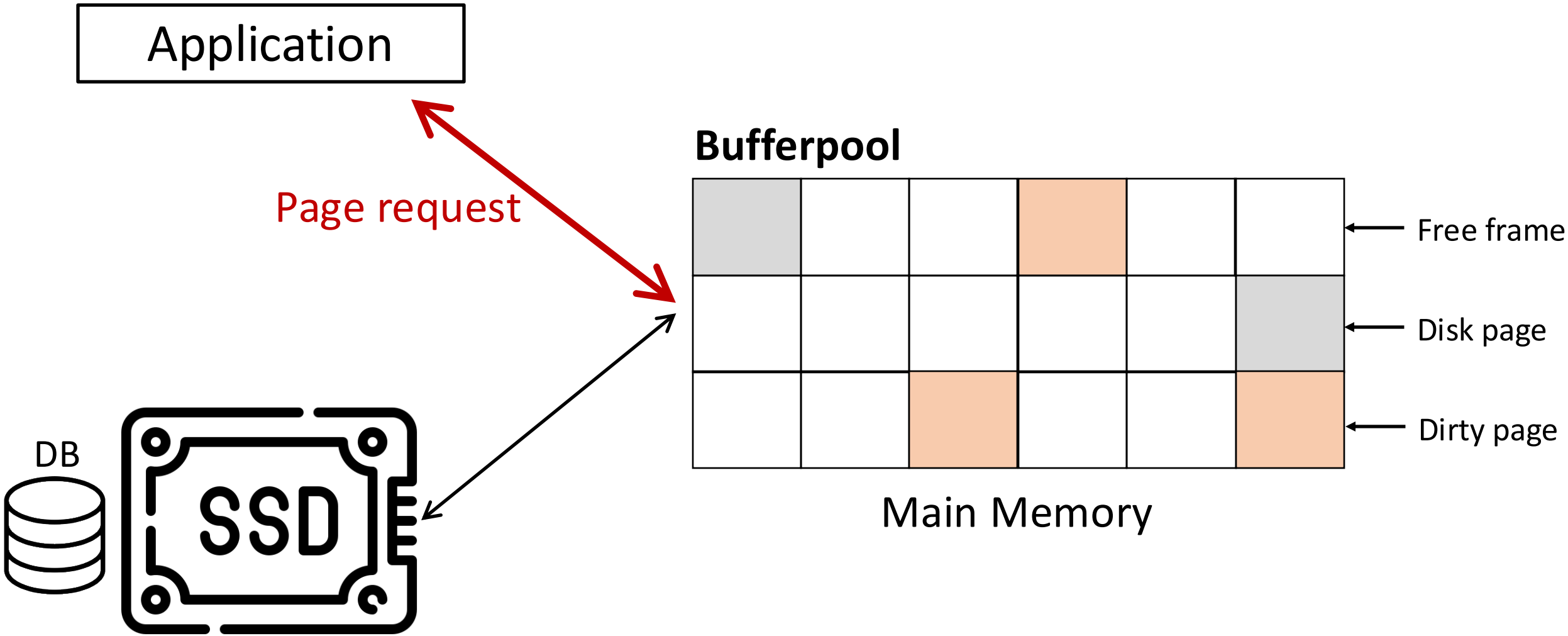
Tailor Data Systems
for **SSD Asymmetry**
& **Concurrency**

Need for an I/O Model [**CIDR '21**]
PIO Model [**DaMoN@SIGMOD '21**]
ACE Bufferpool [IEEE ICDE '23]
CAVE Graph Engine [**SIGMOD '24**]
SSD-Aware Systems [**IEEE ICDE '24**]

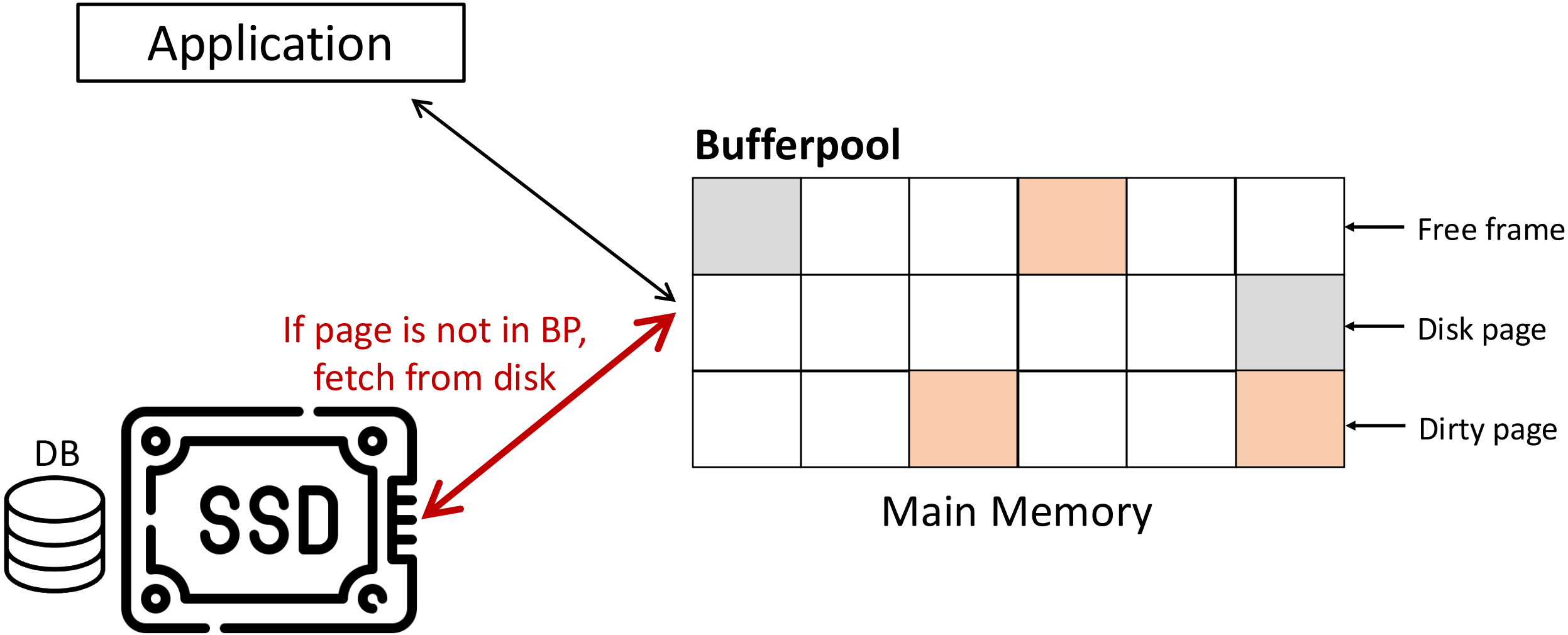
Bufferpool is Tightly Connected to Storage



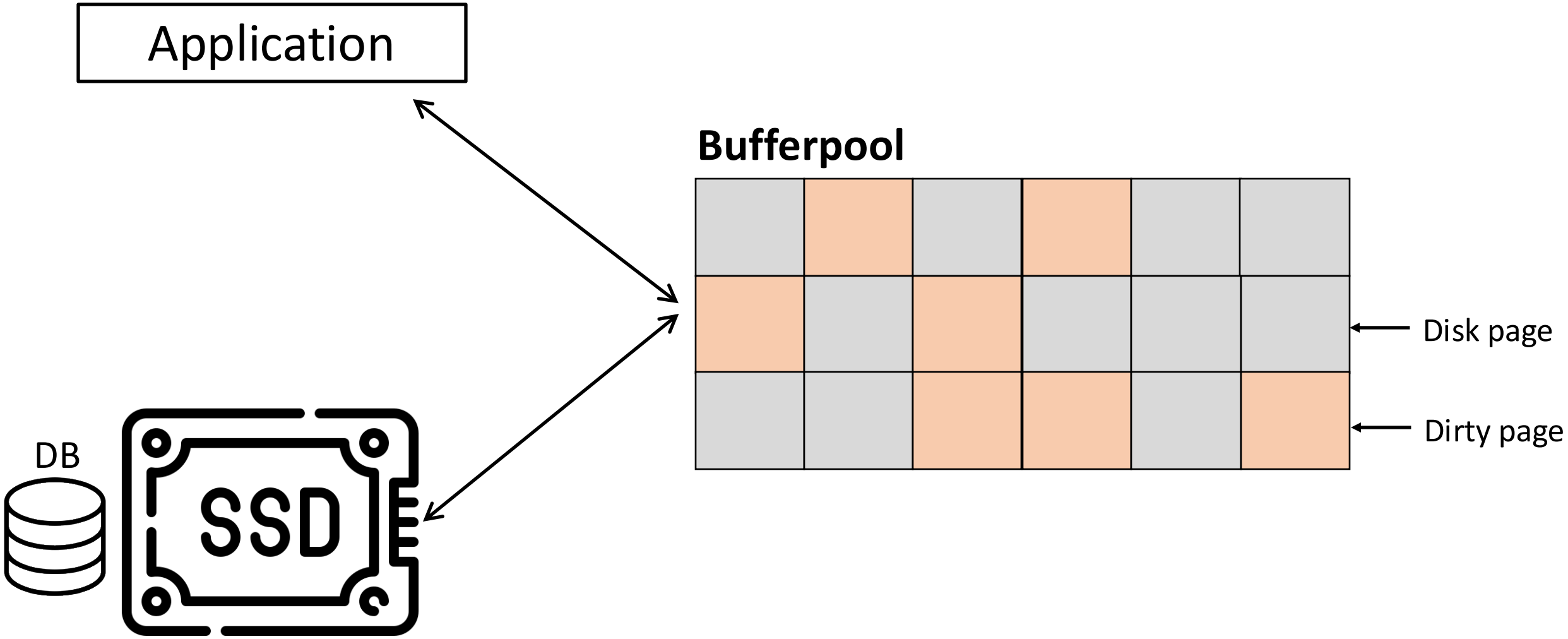
Bufferpool Manager



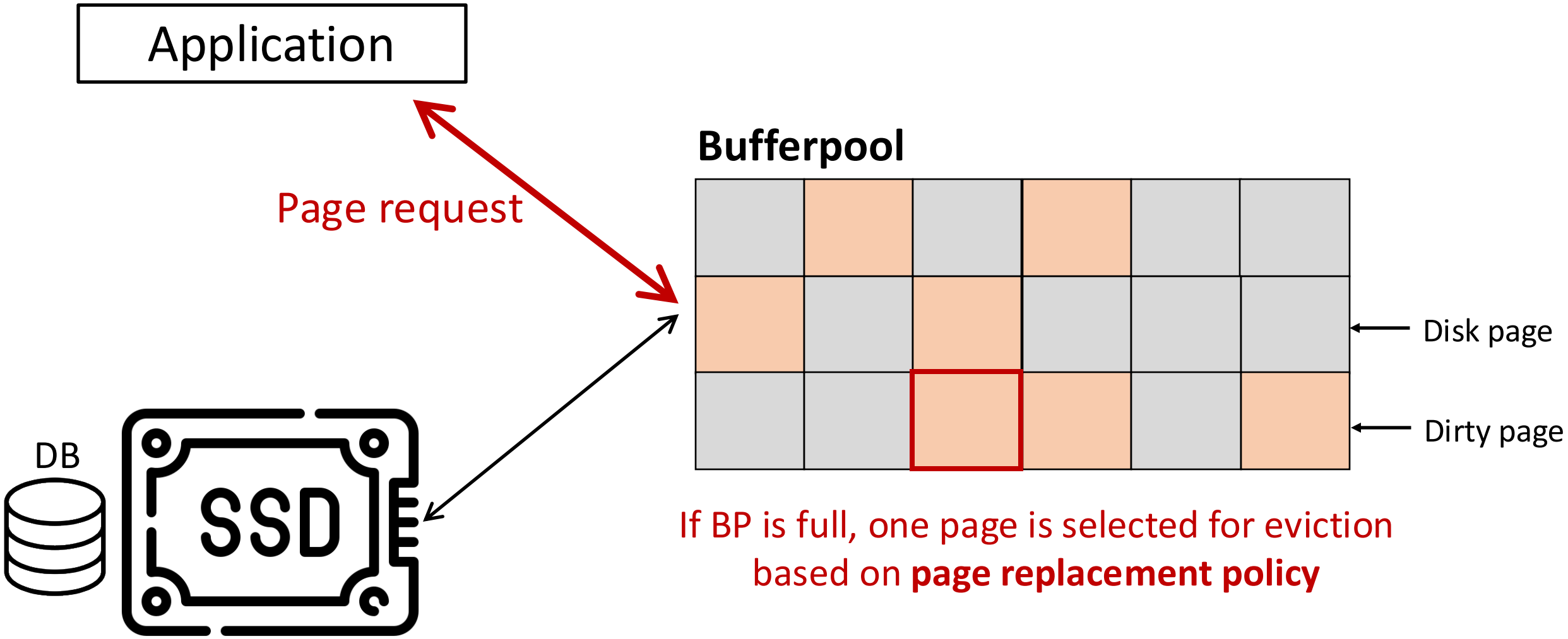
Bufferpool Manager



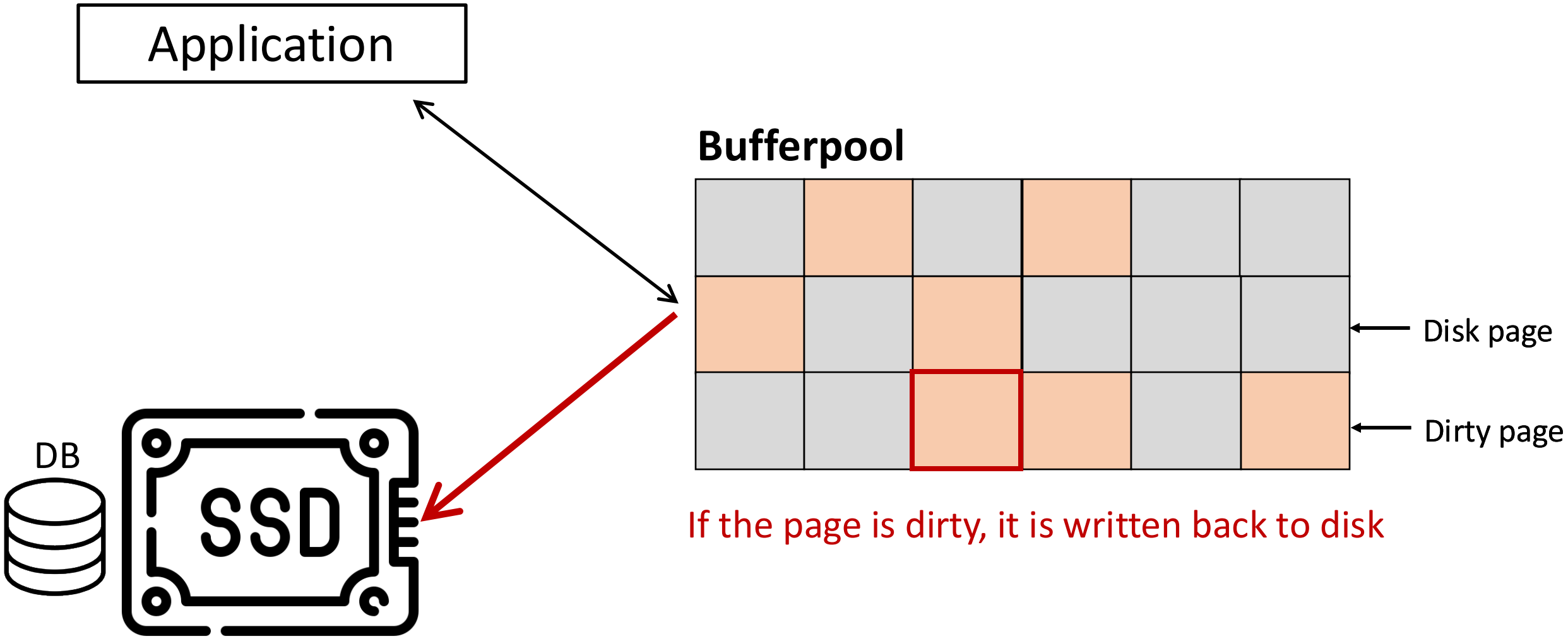
Bufferpool Manager



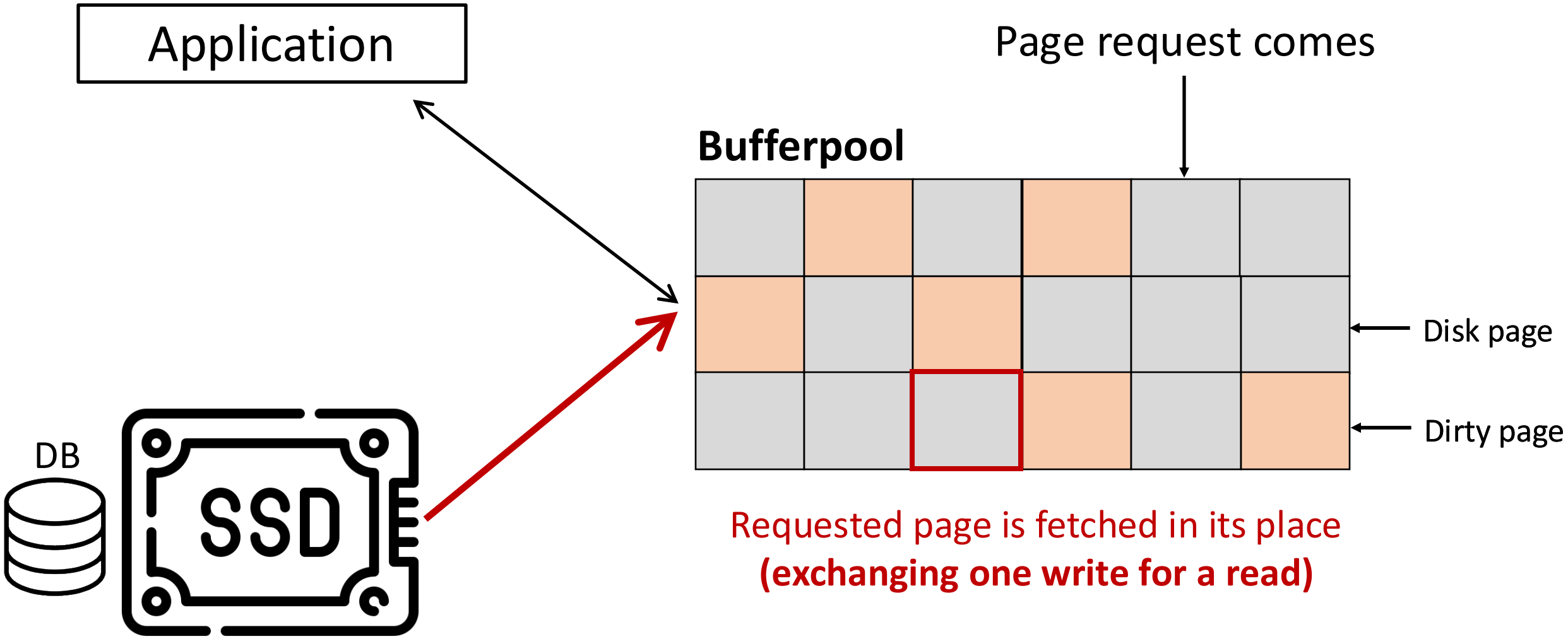
Traditional Bufferpool Manager



Traditional Bufferpool Manager



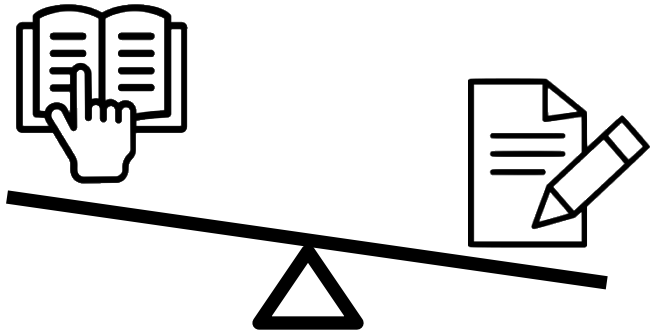
Traditional Bufferpool Manager



The Challenges

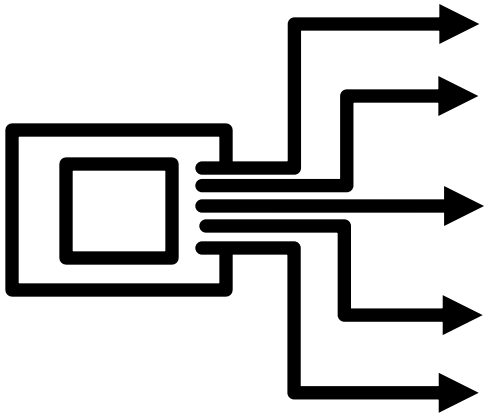
- With write asymmetry, exchanging

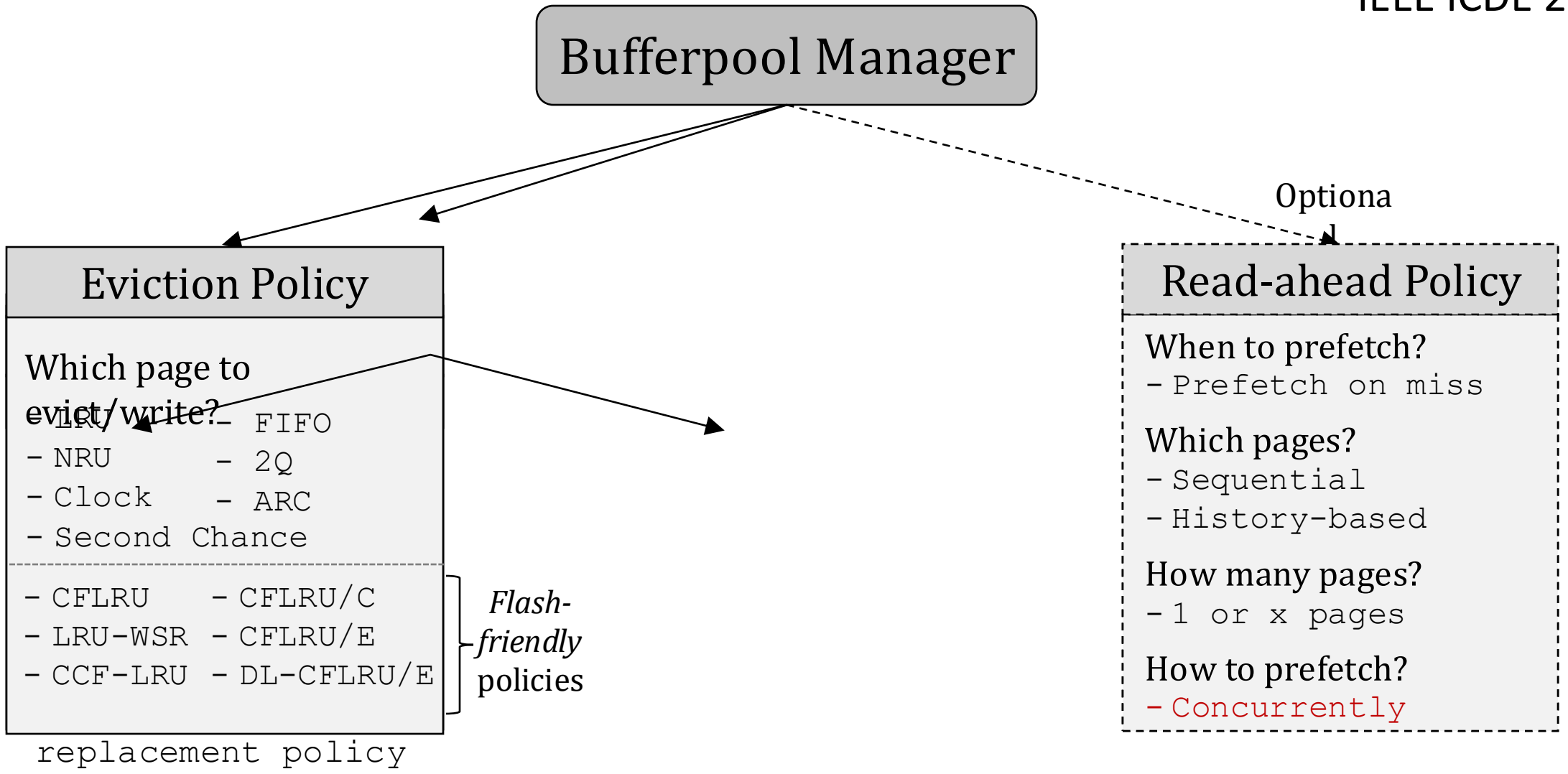
  one write for one read is **NOT ideal**.



- Without exploiting concurrency,

  device remains vastly **underutilized**.

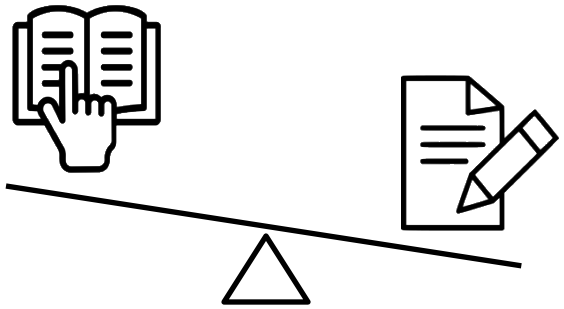


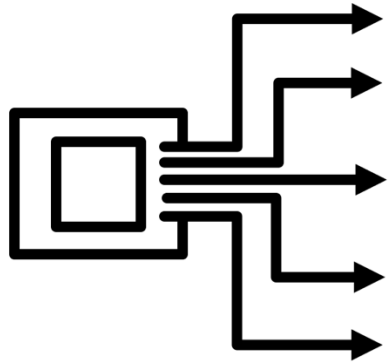

Bufferpool Manager
Optiona
l
Eviction Policy
Which page to evict/write?
- LRU
- FIFO
- NRU
- 2Q
- Clock
- ARC
- Second Chance
- CFLRU
- CFLRU/C
- LRU-WSR
- CFLRU/E
- CCF-LRU
- DL-CFLRU/E
Flash-friendly policies
replacement policy
Read-ahead Policy
When to prefetch?
- Prefetch on miss
Which pages?
- Sequential
- History-based
How many pages?
- 1 or x pages
How to prefetch?
- Concurrently

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

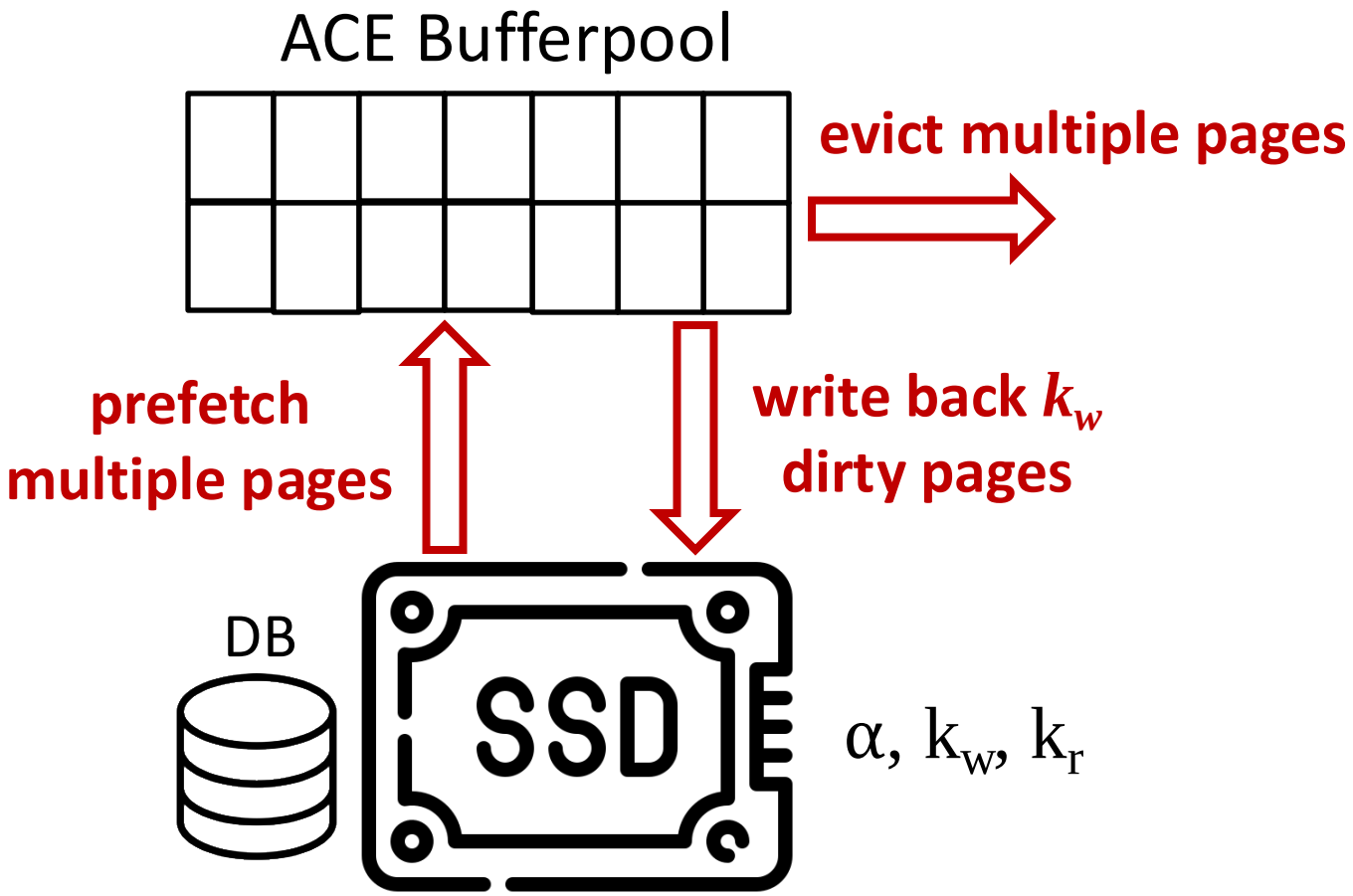
ACE Bufferpool Manager

Use device's properties





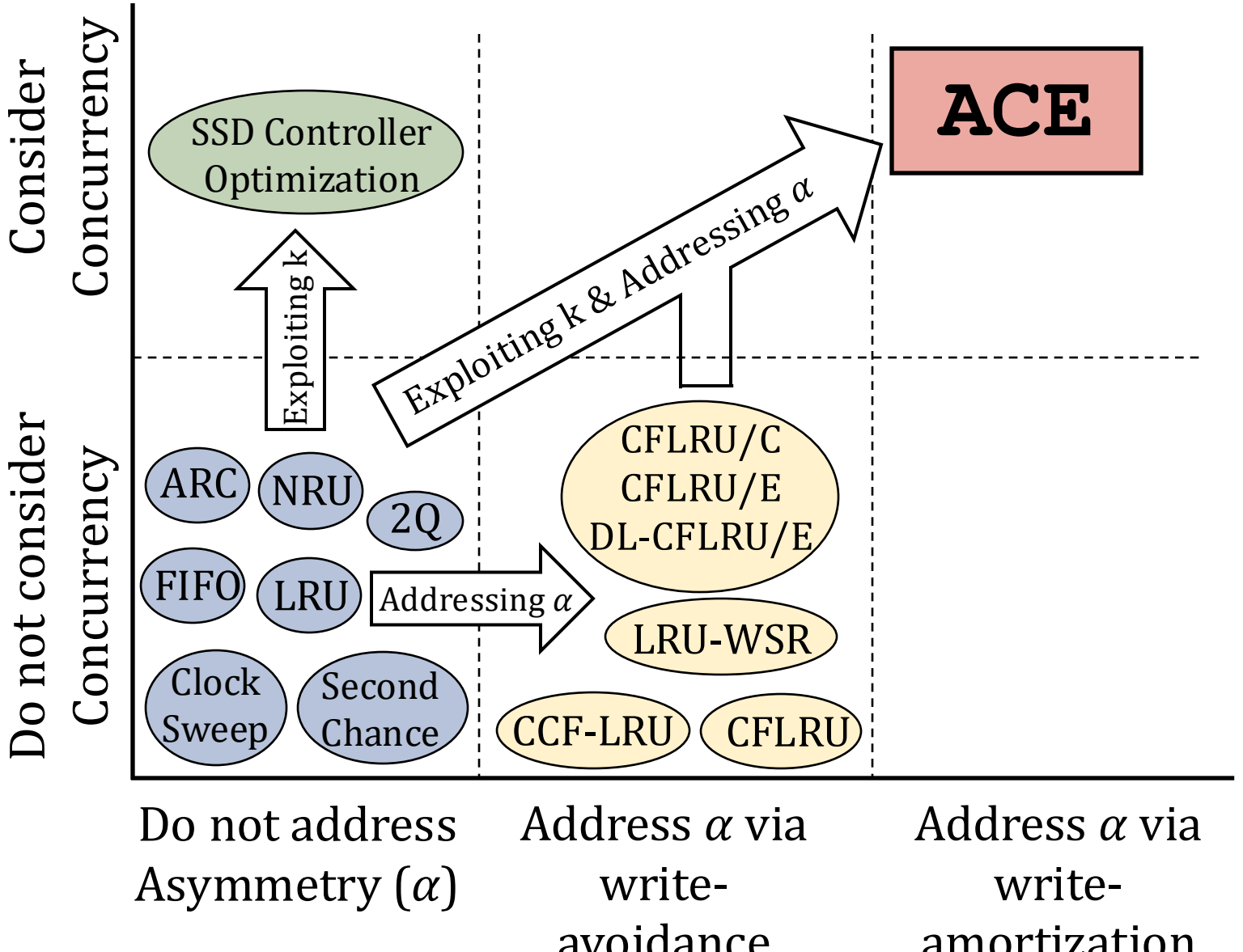
ACE Bufferpool Manager



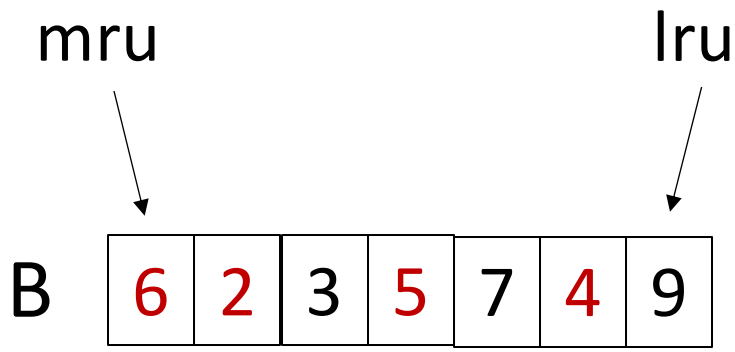
- ✓ Can be integrated with **any** replacement algorithm
- ✓ **Any** prefetching technique can be used

ACE Bufferpool Manager

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



An Example

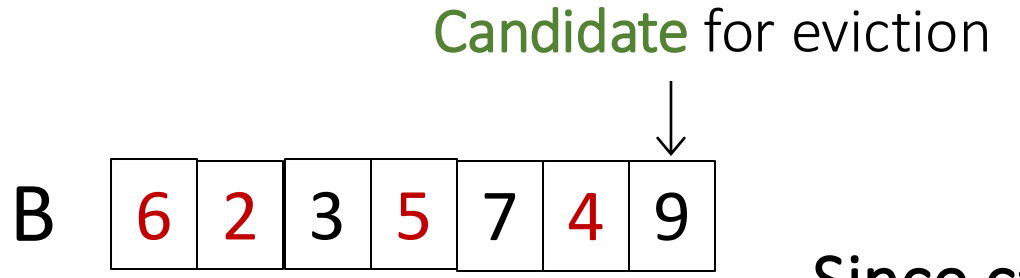


Let's assume: $k_w = 3$, LRU is the replacement policy & red indicates dirty page

Write request of page 8 comes

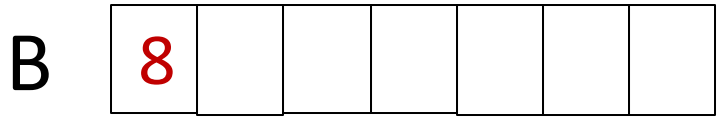
An Example ($k_w = 3$)

`write page 8`



Since candidate page is clean, we simply evict 9

After eviction:



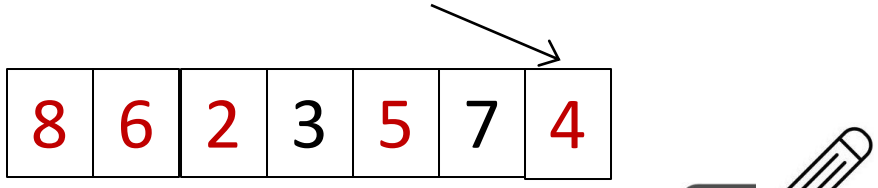
Write request of page 1 comes

An Example ($k_w = 3$)

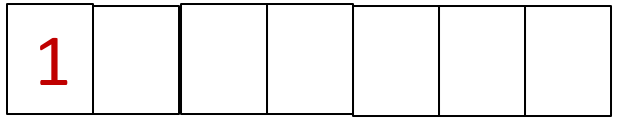
write page 1

LRU

Candidate



After eviction:



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 |

4,5,2 concurrently written

4 evicted

An Example ($k_w = 3$)

write page 1

LRU

B | 8 | 6 | 2 | 3 | 5 | 7 | 4 |

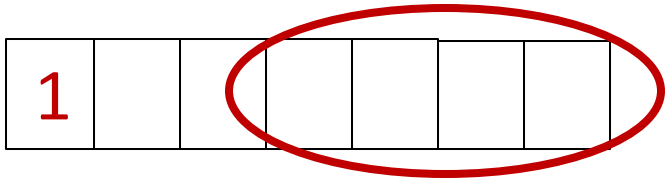
After eviction:

B | 1 | 8 | 6 | 2 | 3 | 5 | 7 |

LRU+ACE (w/o PF)

| 8 | 6 | 2 | 3 | 5 | 7 | |

After eviction:



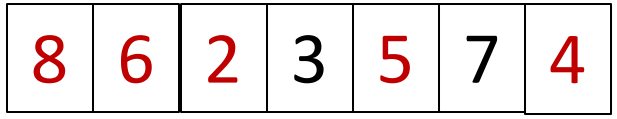
more clean pages

An Example ($k_w = 3$)

write page 1

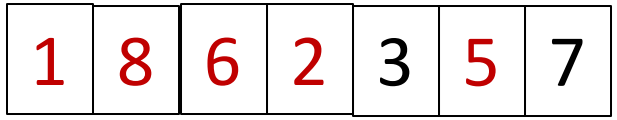
LRU

B

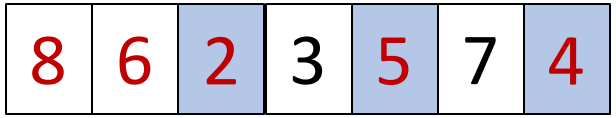


After eviction:

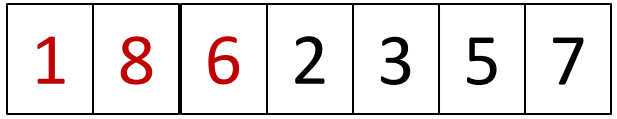
B



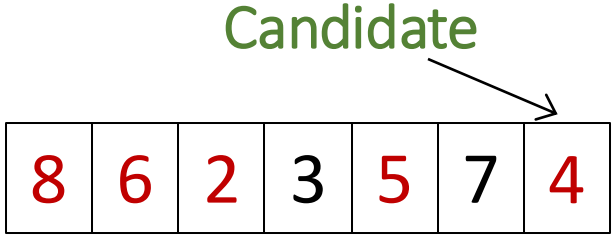
LRU+ACE (w/o PF)



After eviction:



LRU+ACE (w/ PF)



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)

eviction window

| 8 | 6 | 2 | 3 | 5 | 7 | 4 |

4,5,2 concurrently written

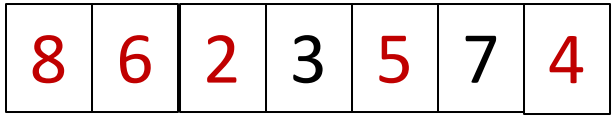
4,7 evicted

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

write page 1

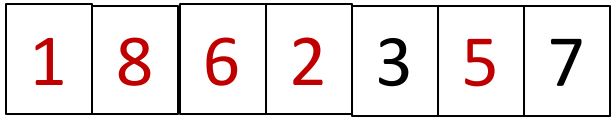
LRU

B

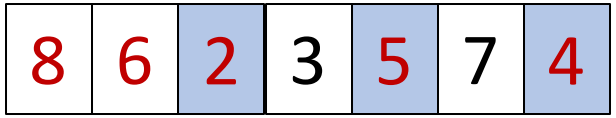


After eviction:

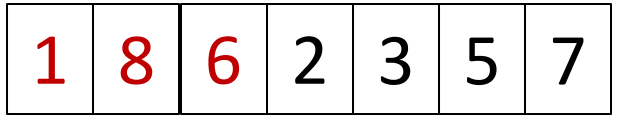
B



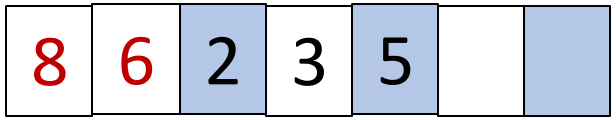
LRU+ACE (w/o PF)



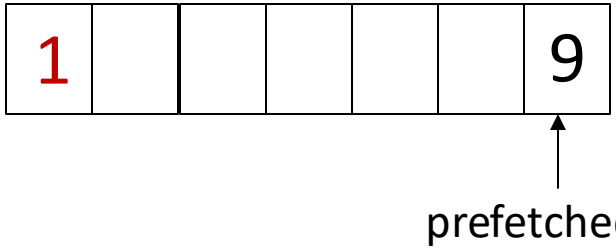
After eviction:



LRU+ACE (w/ PF)



After eviction:



Experimental Evaluation

11.5

Clock Sweep
LRU
CFLRU
LRU-WSR

vs their ACE counterparts

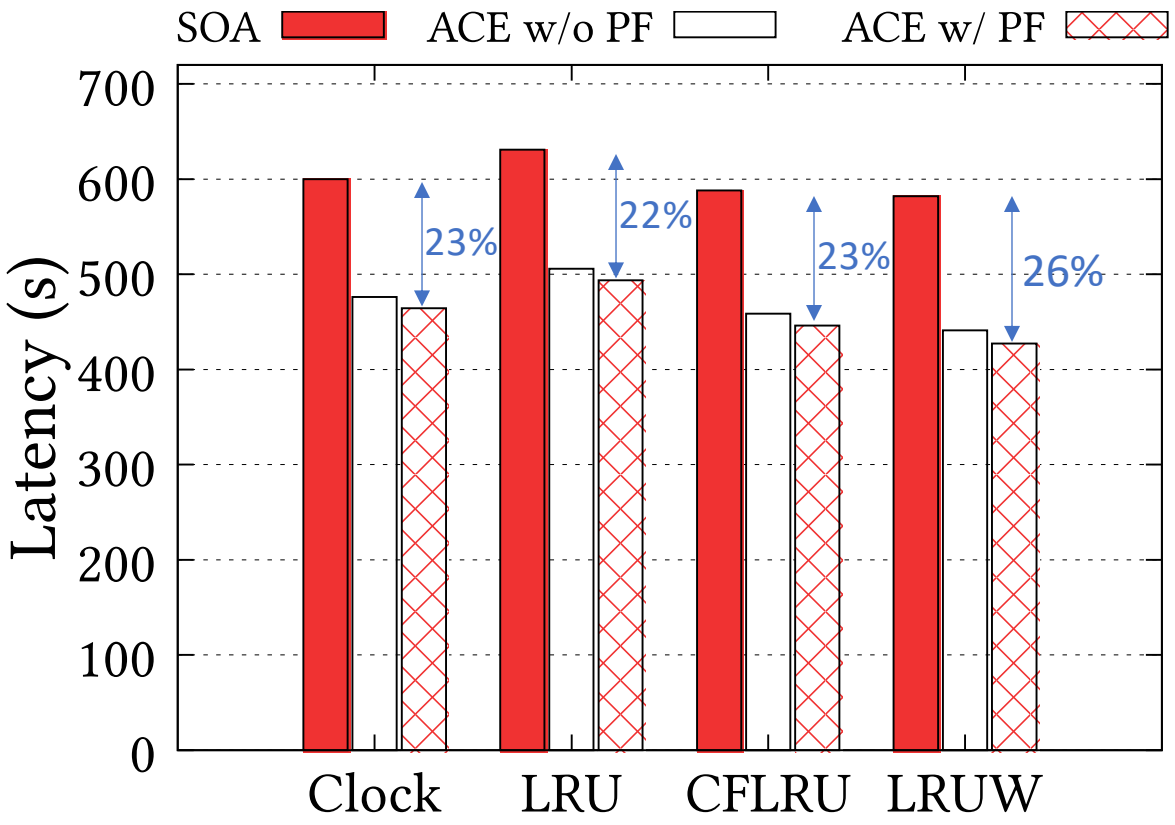
Device	α	k_r	k_w
Optane SSD	1.1	6	5
PCIe SSD	2.8	80	8
SATA SSD	1.5	25	9
Virtual SSD	2.0	11	19

Workload:

synthesized traces

TPC-C benchmark

ACE Improves Runtime



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

Device: PCIe SSD

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

ACE improves runtime by 22-26%

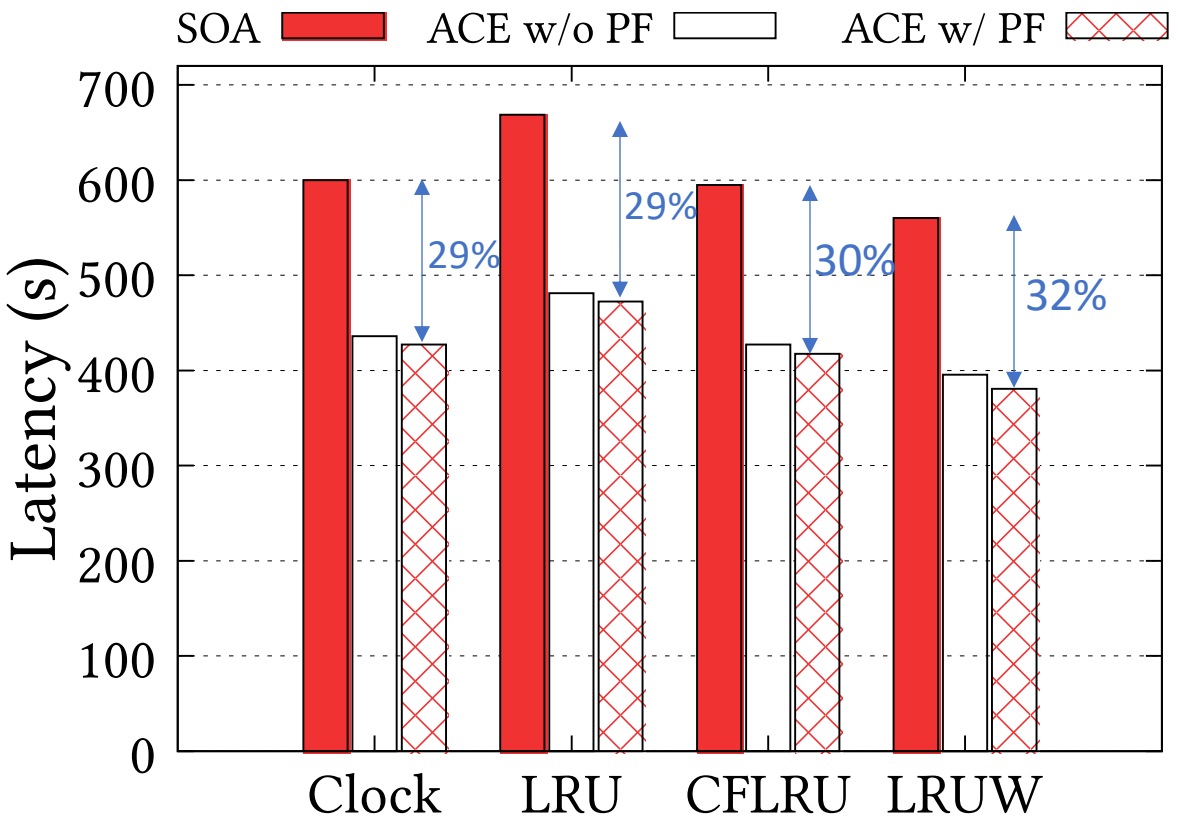
Negligible increase in buffer miss (<0.009%)

Benefit comes at no cost

Higher Gain for Write-Heavy Workload

Device: PCIe SSD

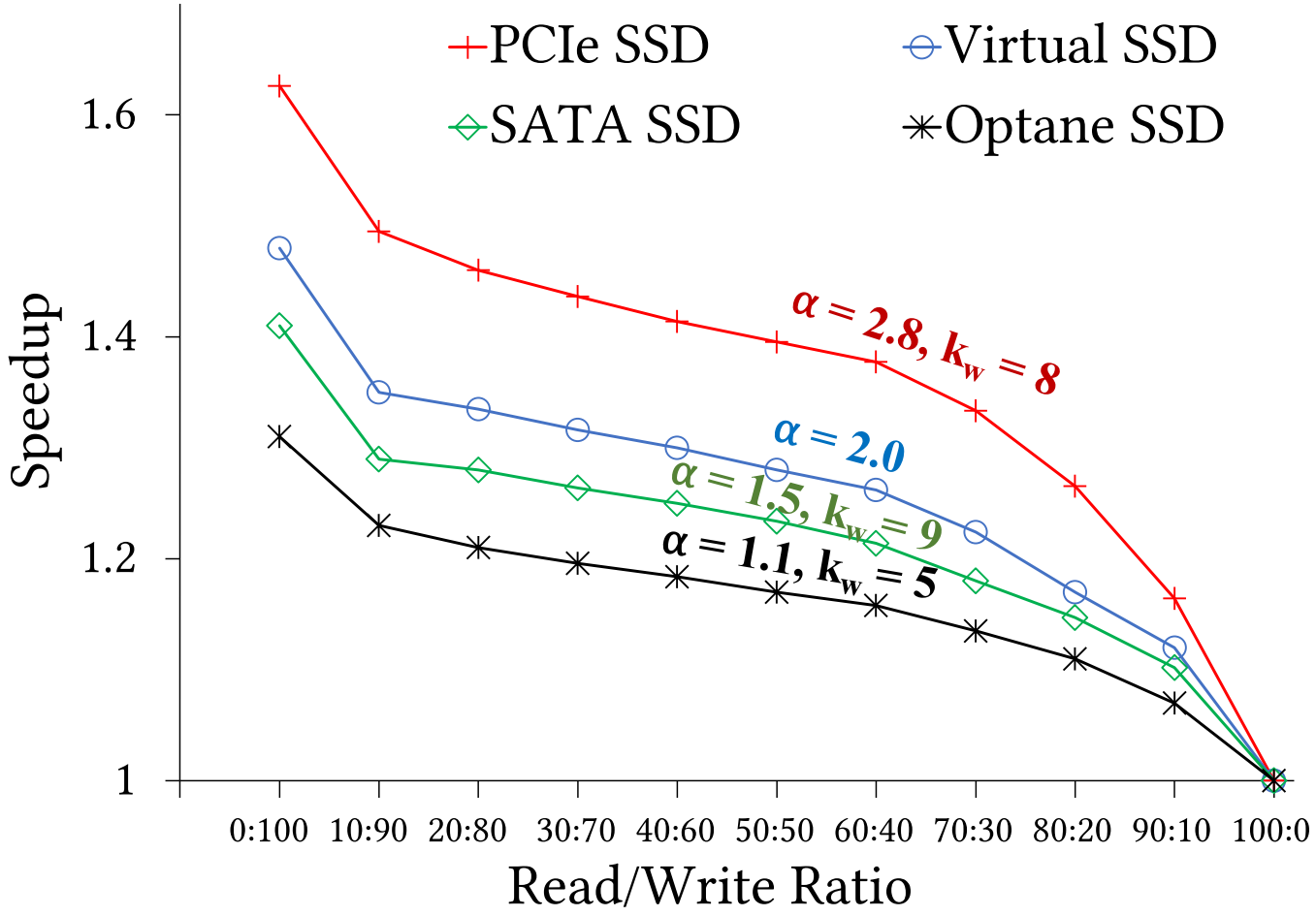
$\alpha = 2.8, k_w = 8$



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

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

Impact of R/W Ratio & Asymmetry

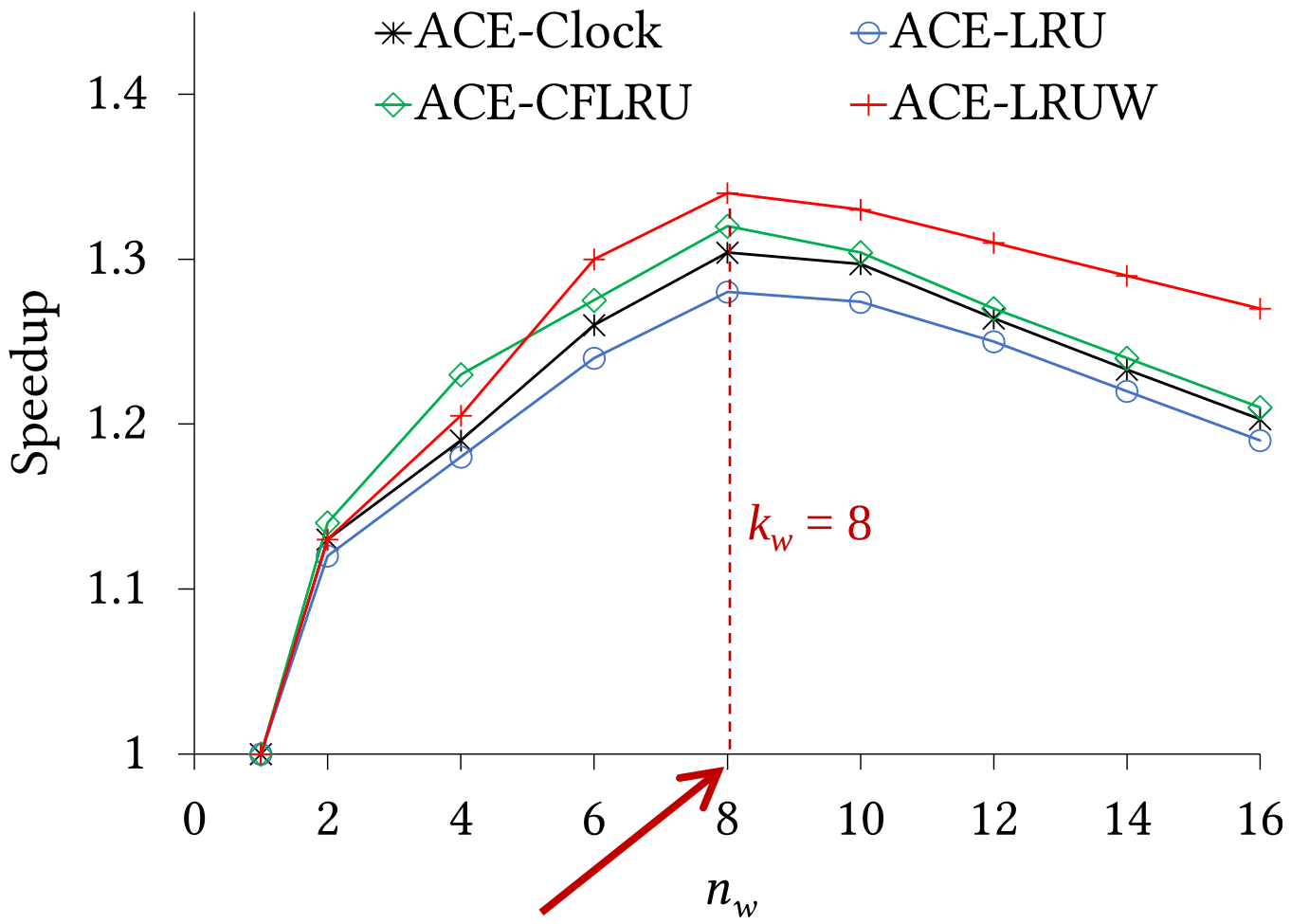


more writes, more speedup

higher asymmetry, higher speedup

good benefit even for low asymmetry

Impact of #Concurrent I/Os



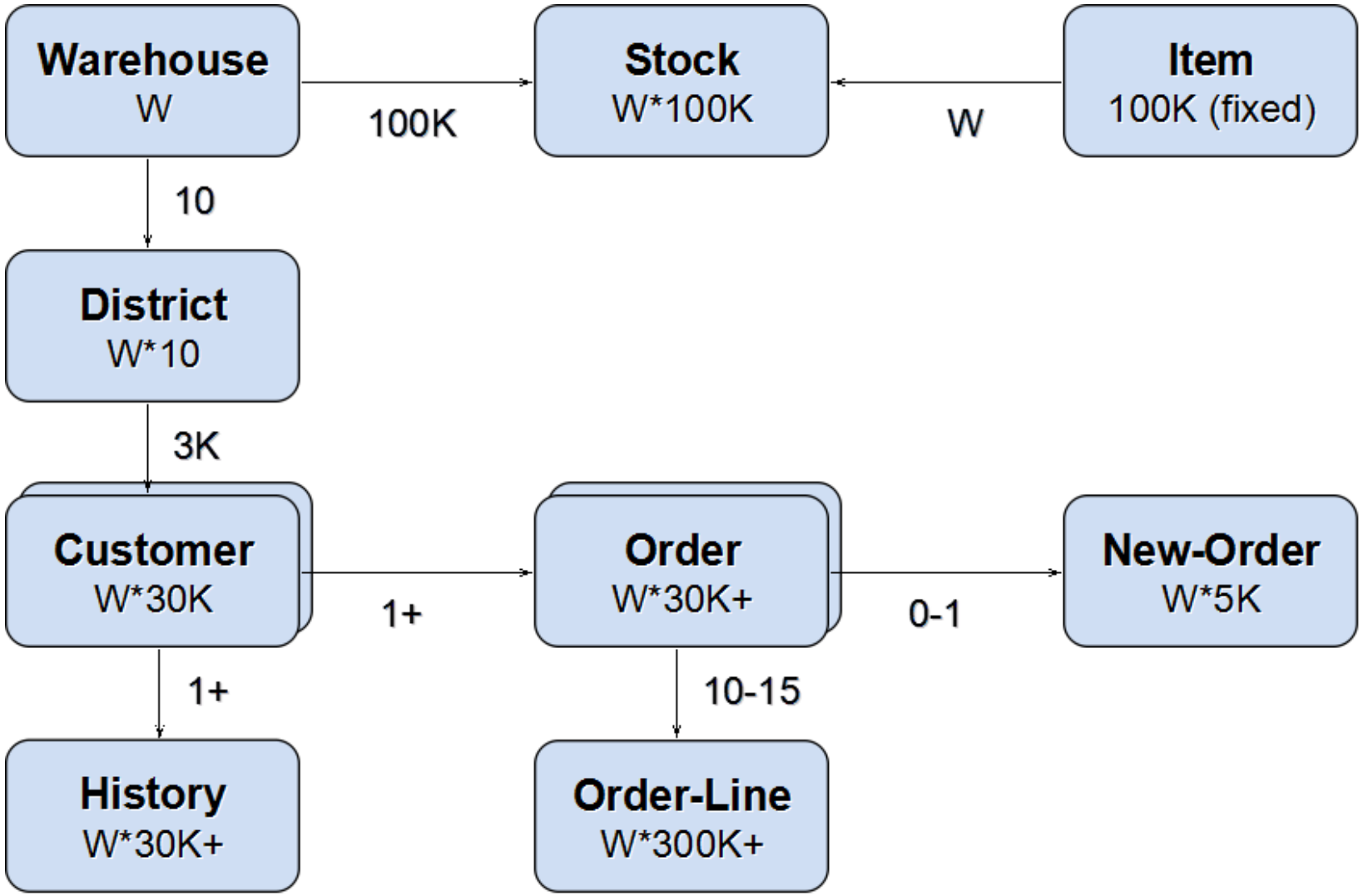
Mixed Skewed Trace
(r/w: 50/50, locality

Device: PCIe SSD

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

Highest speedup when optimal concurrency is used

Experimental Evaluation (TPC-C)

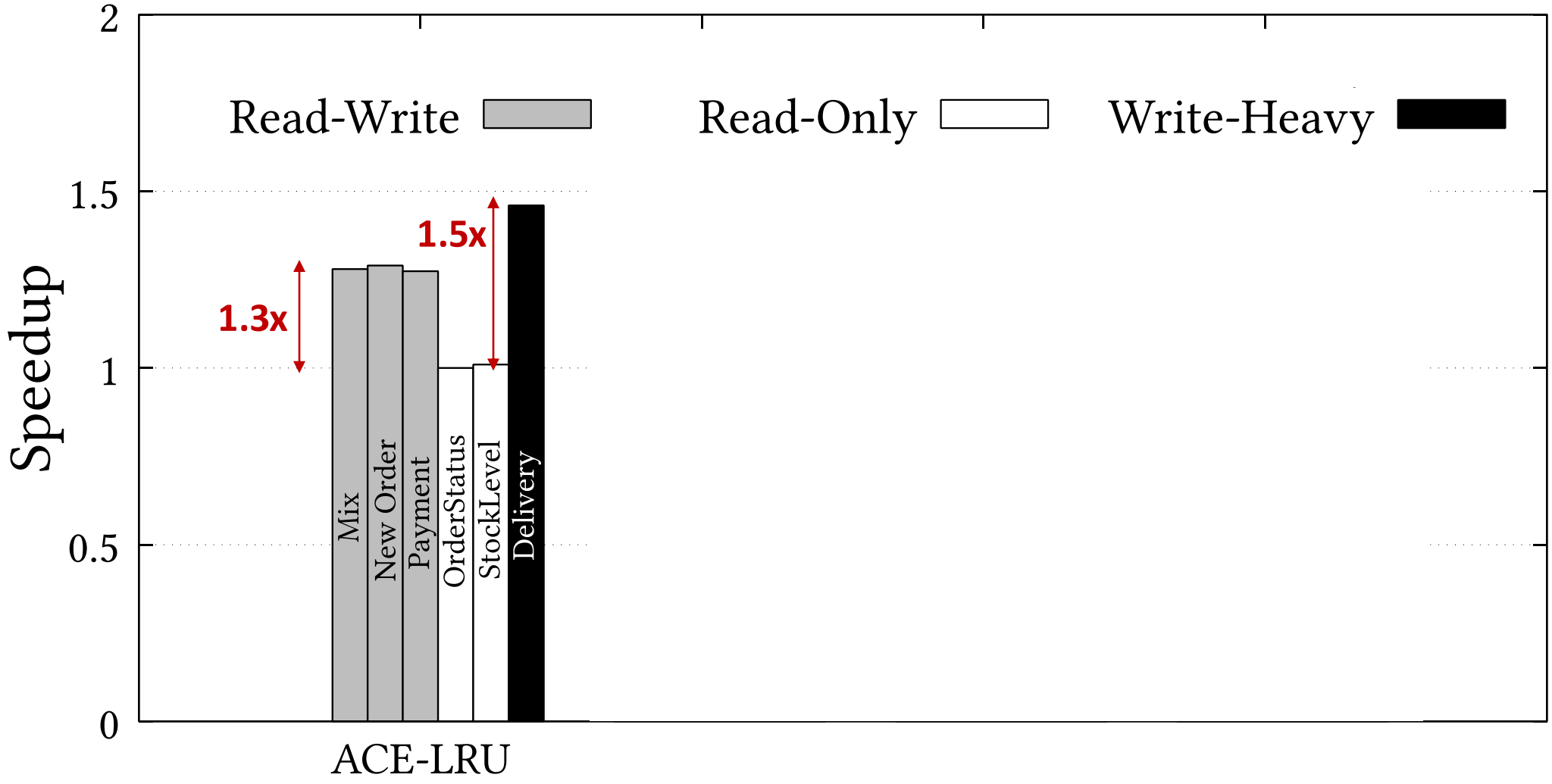


Experimental Evaluation (TPC-C)

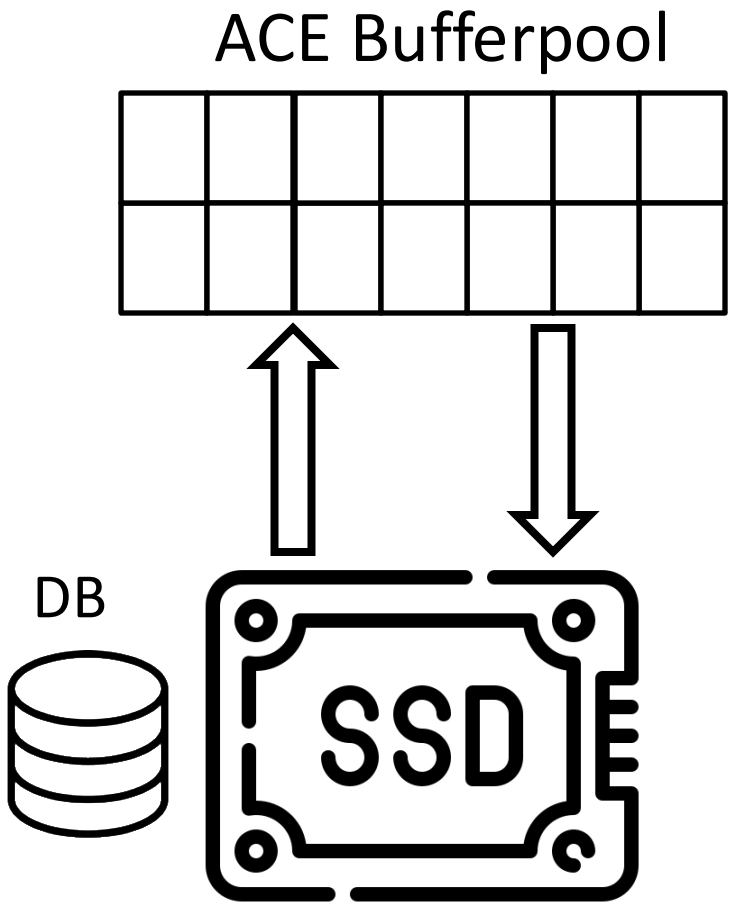
TPC-C consists of 5 transactions

NewOrder (45%)	R/W Mix
Payment (43%)	R/W Mix
OrderStatus (4%)	R-only
StockLevel (4%)	R-only
Delivery (4%)	W-heavy

Experimental Evaluation (TPC-C)

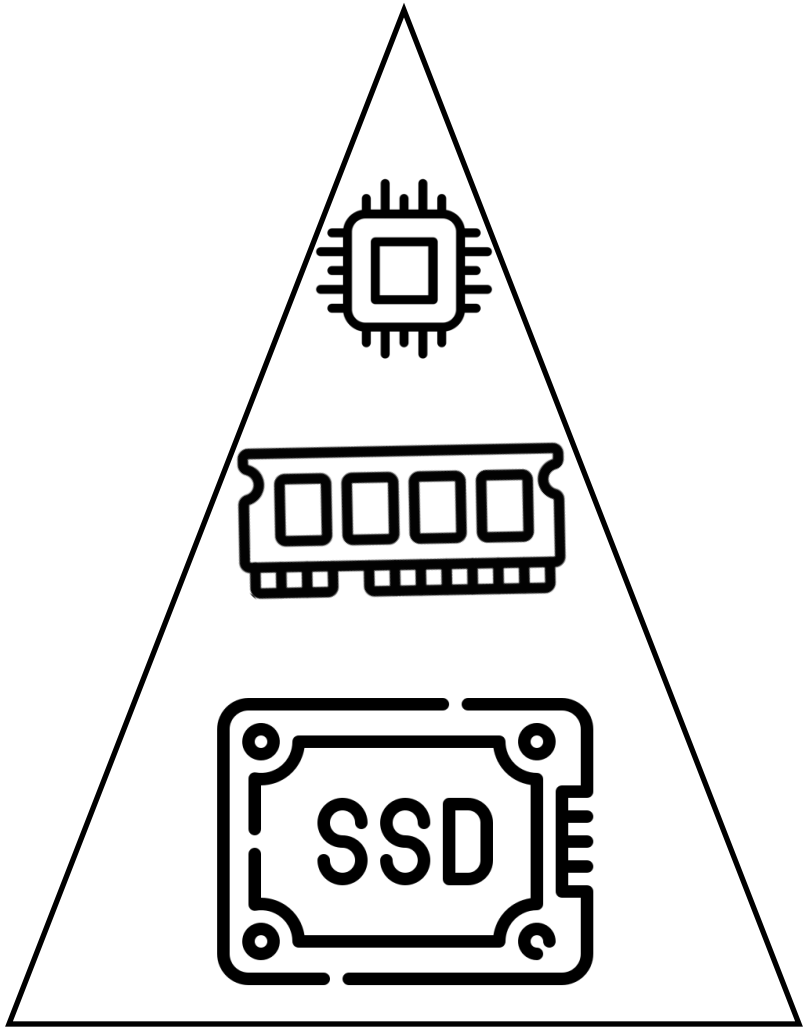


ACE Achieves 1.3x for mixed TPC-C



- ☑ **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

Goal: Developing Hardware-Aware Data Systems



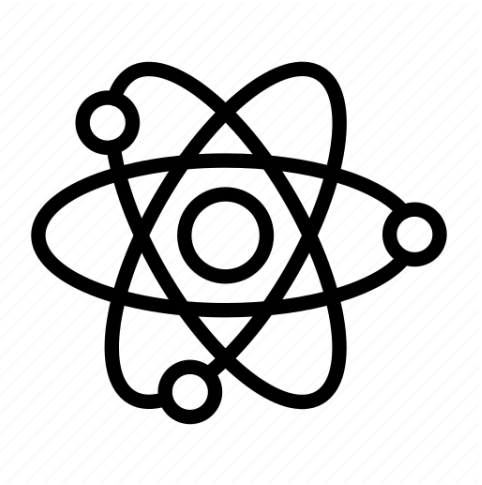
Tailor Data Systems
for **SSD Asymmetry**
& **Concurrency**

Need for an I/O Model [**CIDR '21**]
PIO Model [**DaMoN@SIGMOD '21**]
ACE Bufferpool [**IEEE ICDE '23**]
CAVE Graph Engine [**SIGMOD '24**]
SSD-Aware Systems [**IEEE ICDE '24**]

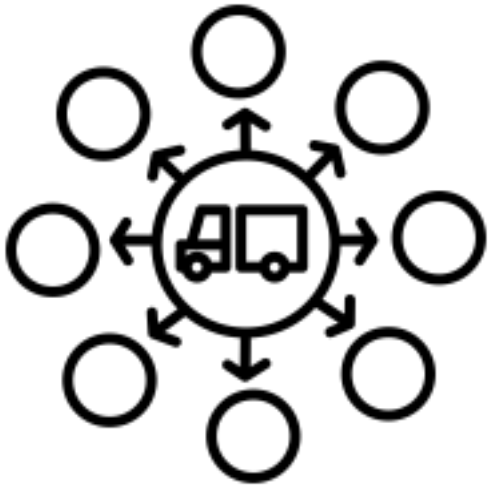
Rise of Large Graphs

Graphs are everywhere!

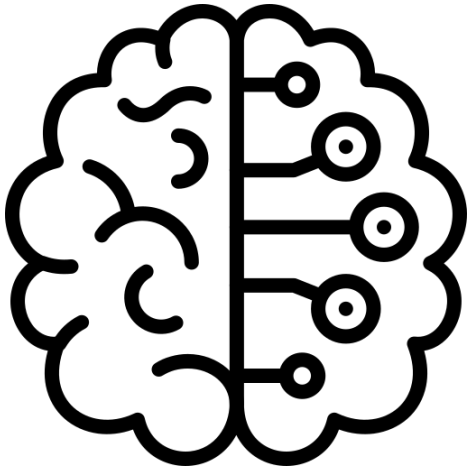
Social Network



Physical Science



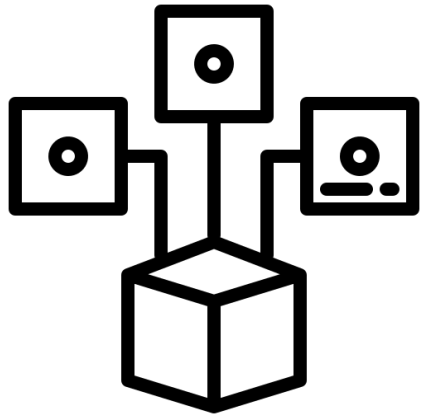
Transportation Network



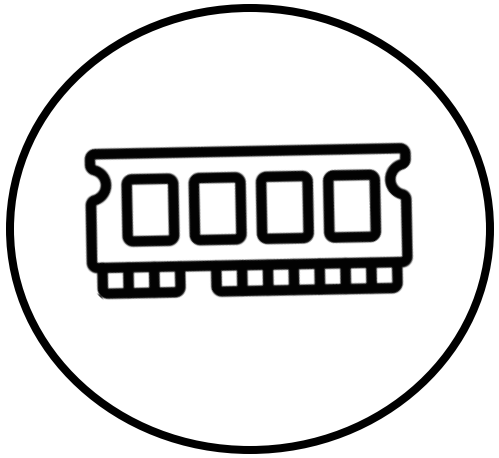
Machine Learning

Real-world graphs often have more than a billion nodes

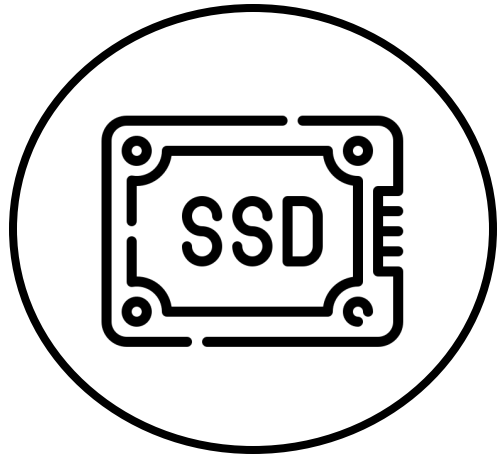
Processing Large Graphs



Distributed Systems

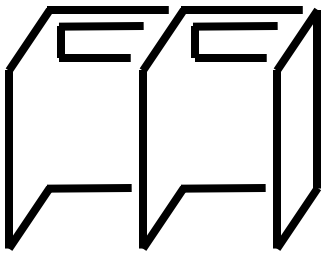


Single-node
in-memory systems



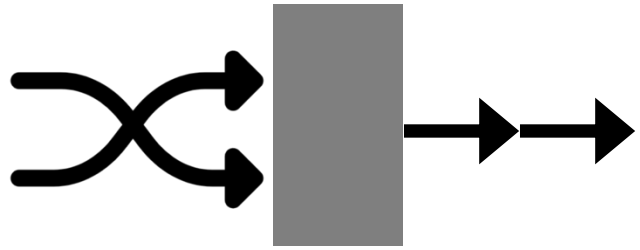
Single-node
out-of-core systems

Out of Core Systems



Data partitioning

Improve memory
& disk locality



Reduce random I/O

Designed for HDDs

Our Goal

- Optimize for **storage-based** workload
- Focus on **traversal** operations
- Utilize efficient SSD **concurrency** by parallelizing independent I/Os
- Maintain **core** algorithm properties

Concurrency-Aware Graph (V, E) Manager

CAVE

Concurrent Graph Algorithms

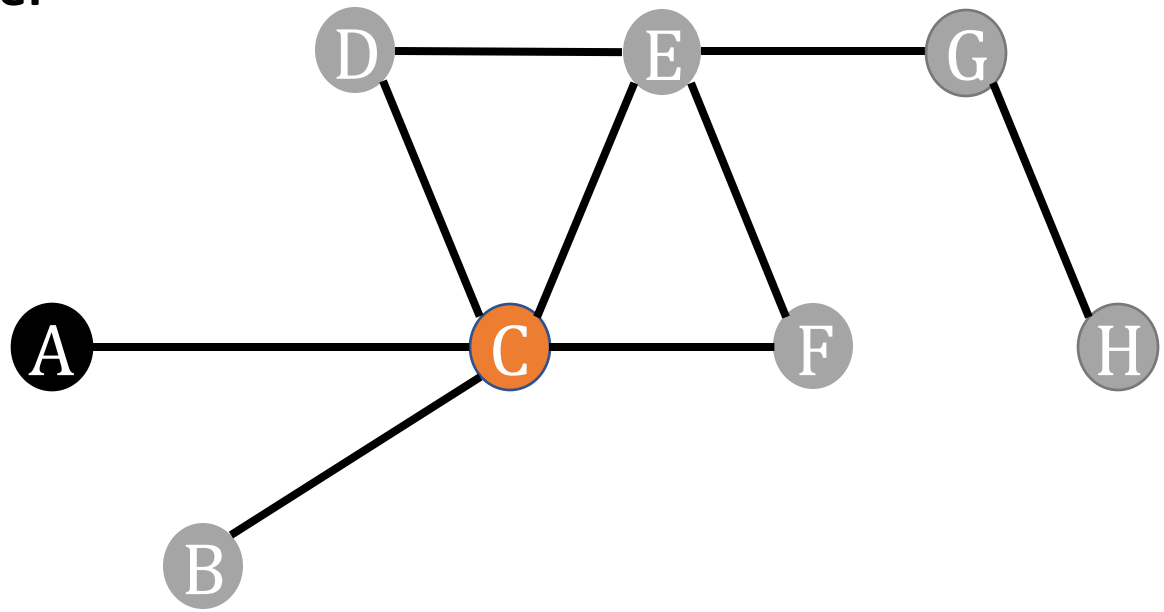
- Parallel Breadth-First Search
- Parallel pseudo Depth-First Search
- Parallel Weakly Connected Components
- Parallel PageRank
- Parallel Random Walk

Parallel BFS

processed nodes processing in progress yet to be processed

Each iteration involves

1. processing a list of vertices aka the **frontier**
2. accessing the neighbors of each vertex
3. updating vertex values
4. determining which vertices should be visited in the next iteration

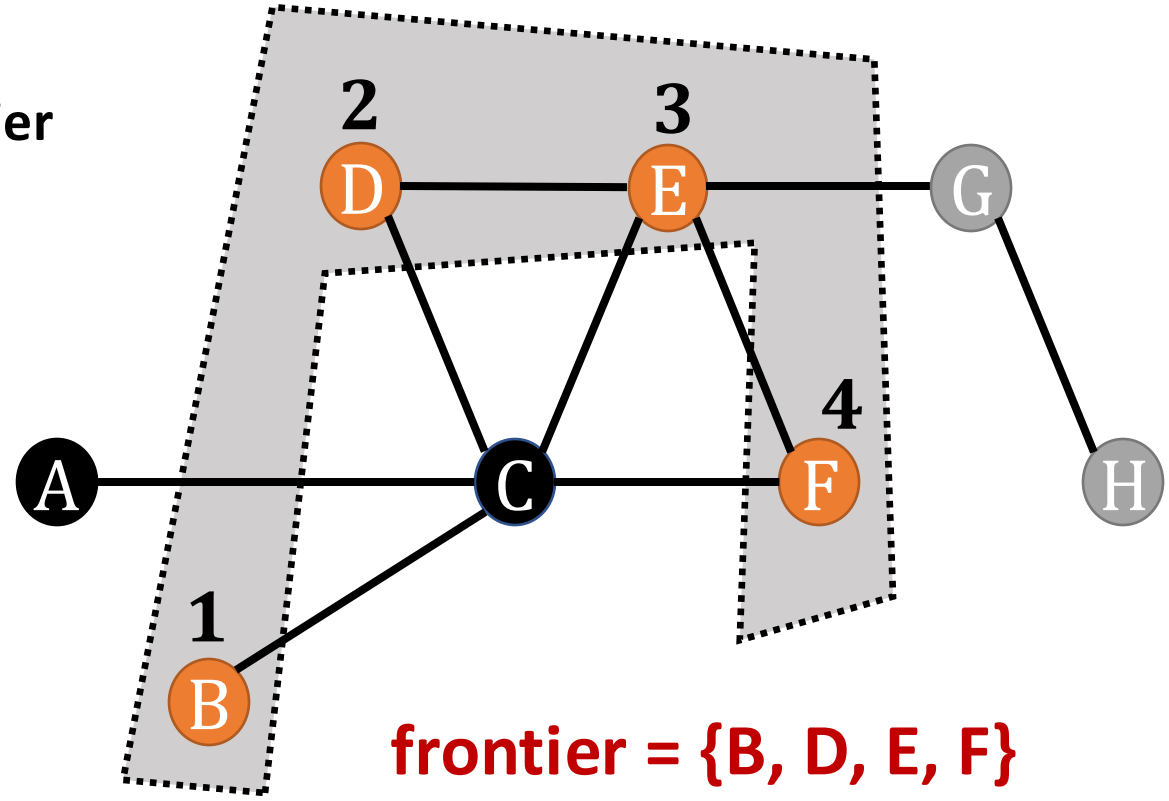


Parallel BFS

processed nodes processing in progress yet to be processed

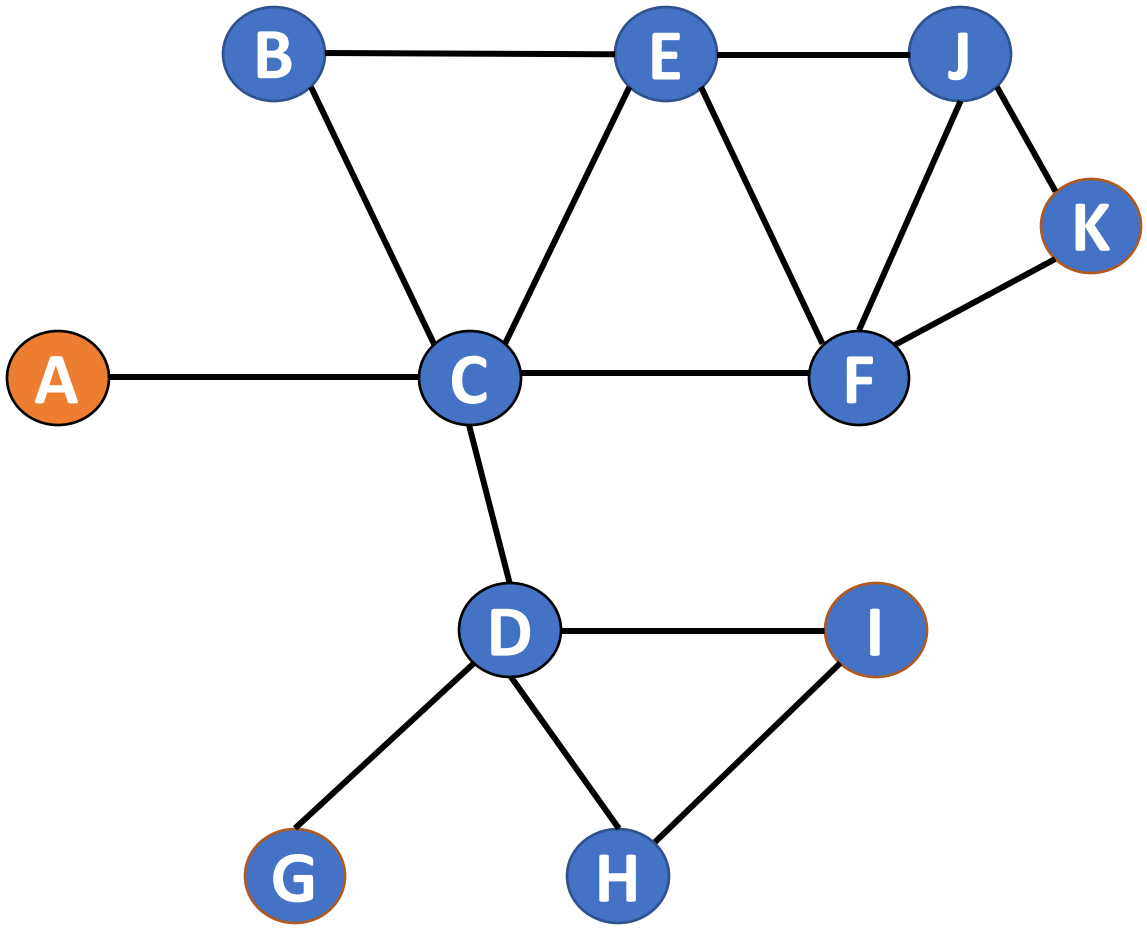
Each iteration involves

1. processing a list of vertices aka the **frontier**
2. accessing the neighbors of each vertex
3. updating vertex values
4. determining which vertices should be visited in the next iteration



Parallel pseudo DFS

processed nodes processing in progress yet to be processed



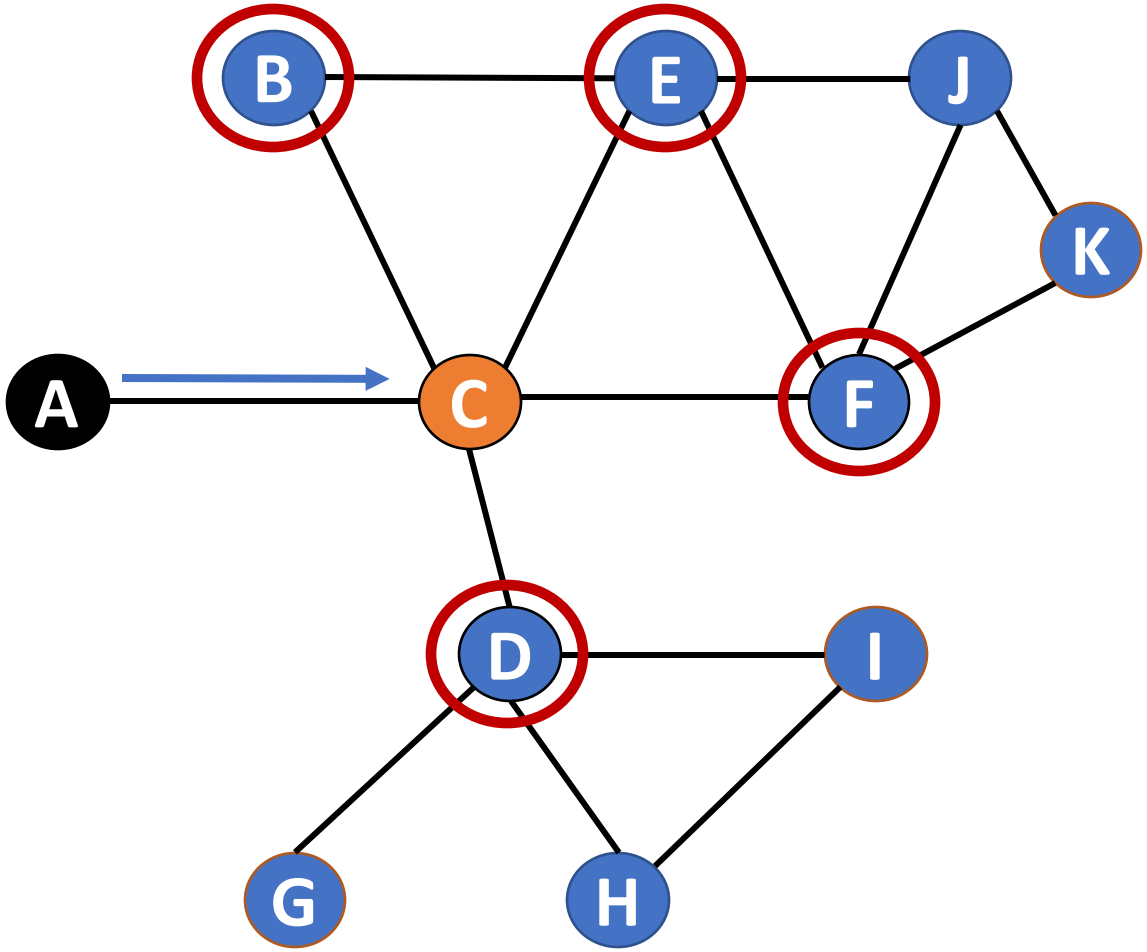
Time

1

A **Thread #1**

Parallel pseudo DFS

processed nodes processing in progress yet to be processed



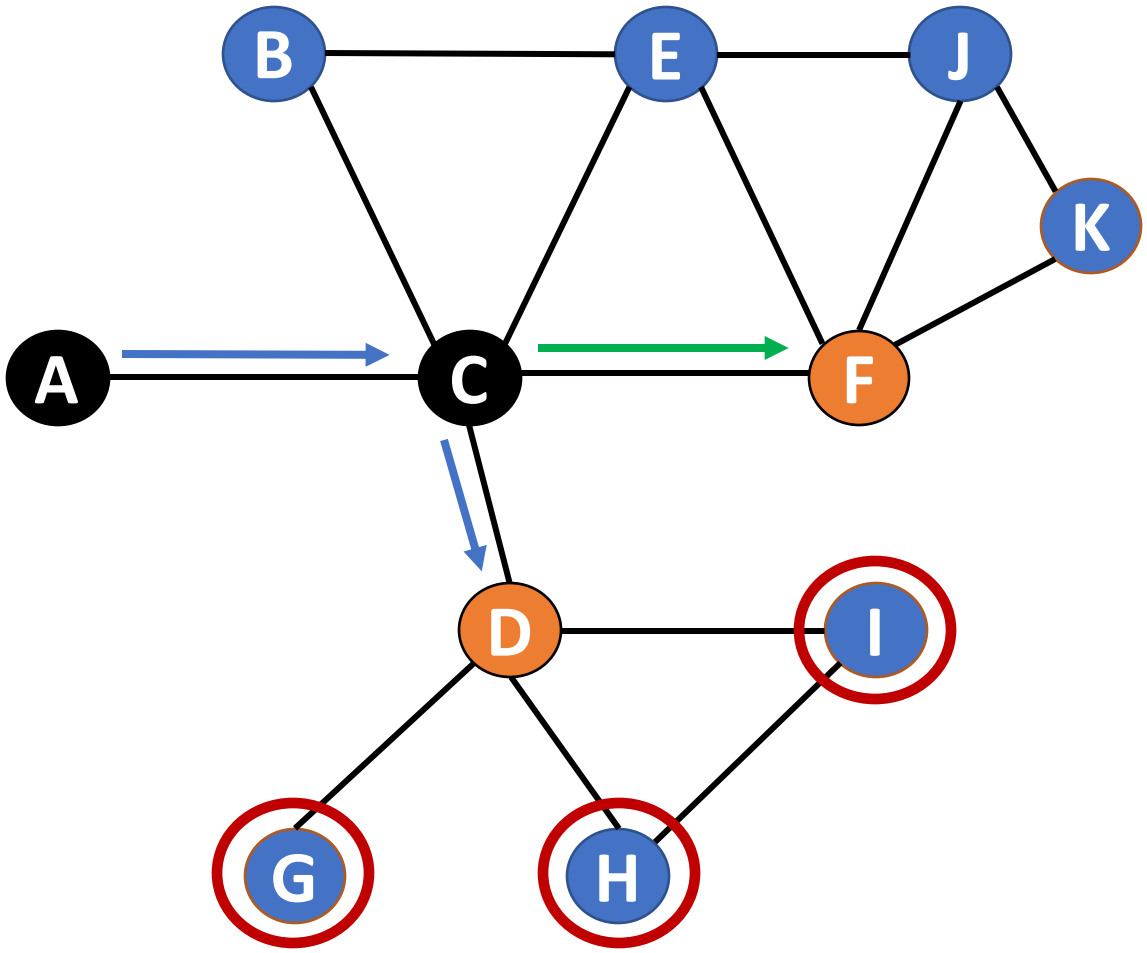
Time

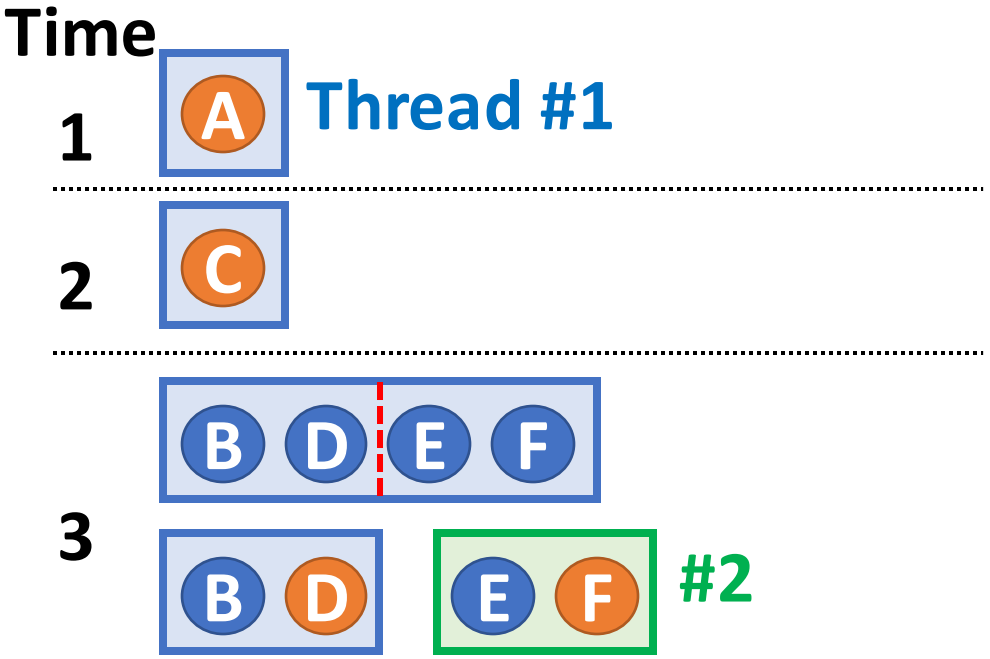
1 A **Thread #1**

2 C

Parallel pseudo DFS

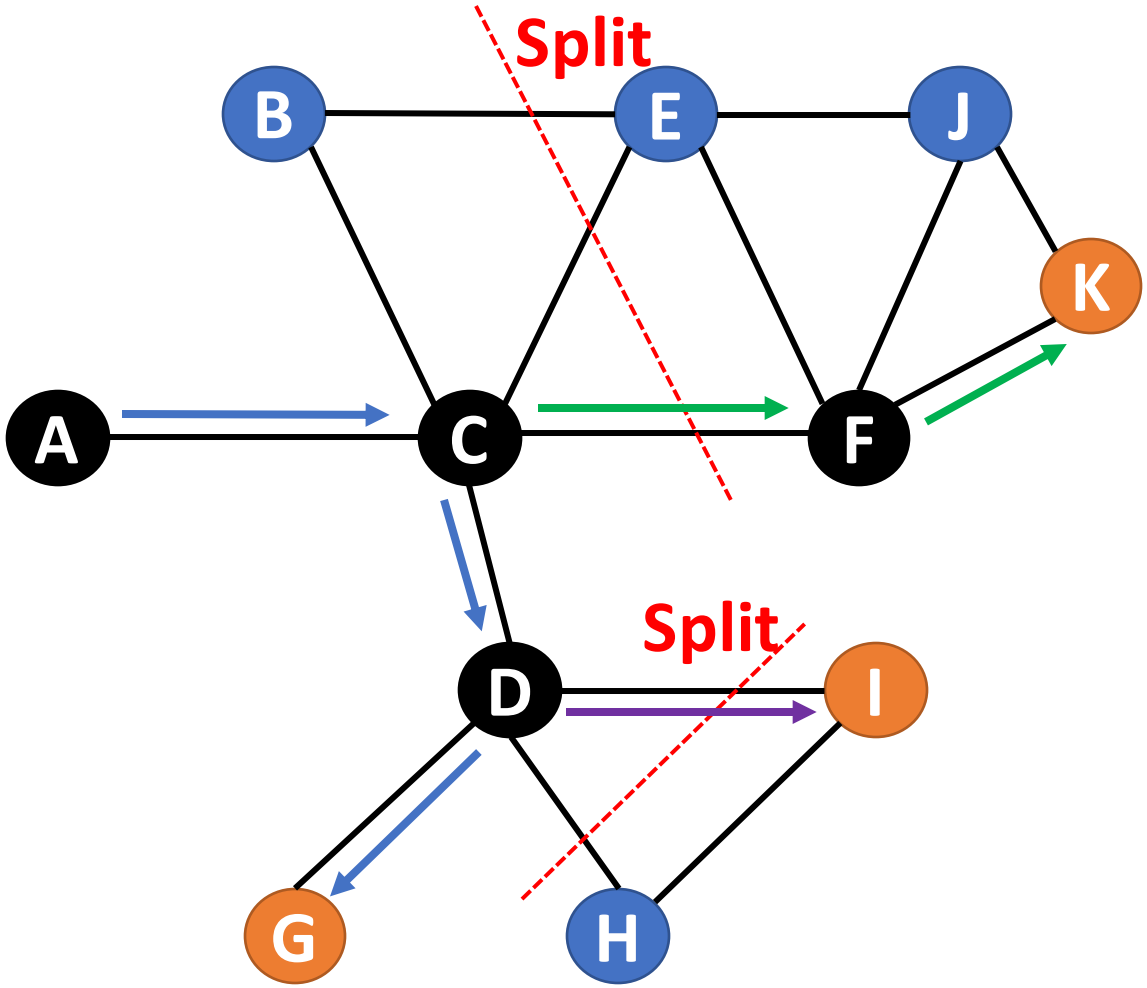
processed nodes processing in progress yet to be processed

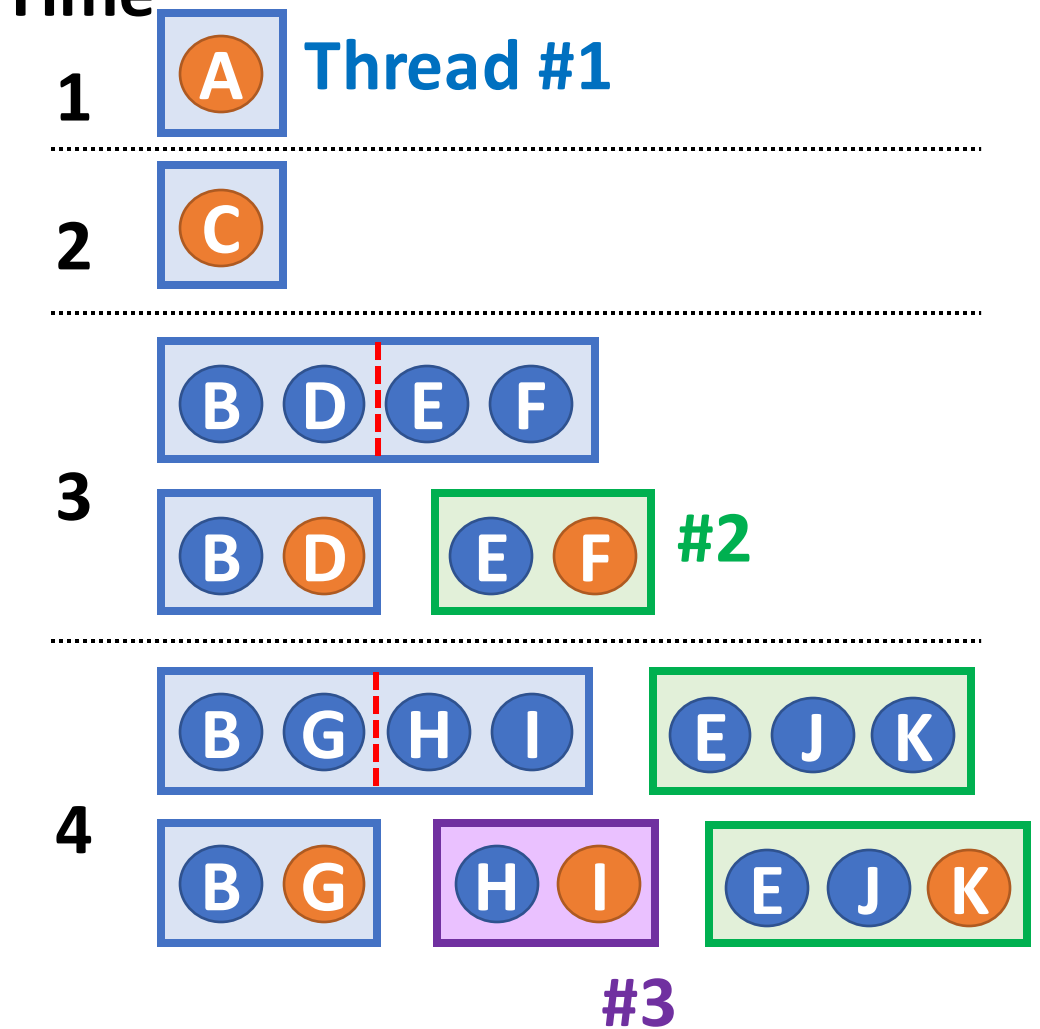




Parallel pseudo DFS

processed nodes processing in progress yet to be processed





Experimental Evaluation

6 datasets

Dataset	Description	#Nodes	#Edges	Diameter	Size
FS	Friendster Social Network	65M	1.8B	32	32 GB
TW	Twitter Social Network	53M	2B	18	28 GB
RN	RoadNet Network of PA	1M	1.5M	786	47 MB
LJ	LiveJournal Social Network	5M	69M	16	1 GB
YT	YouTube Social Network	1.1M	3M	20	39 MB
SD	Synthetic data	50M	1.25B	6	20 GB

3 devices

Optane SSD ($k_r = 6$)

PCIe SSD ($k_r = 80$)

SATA SSD ($k_r = 25$)

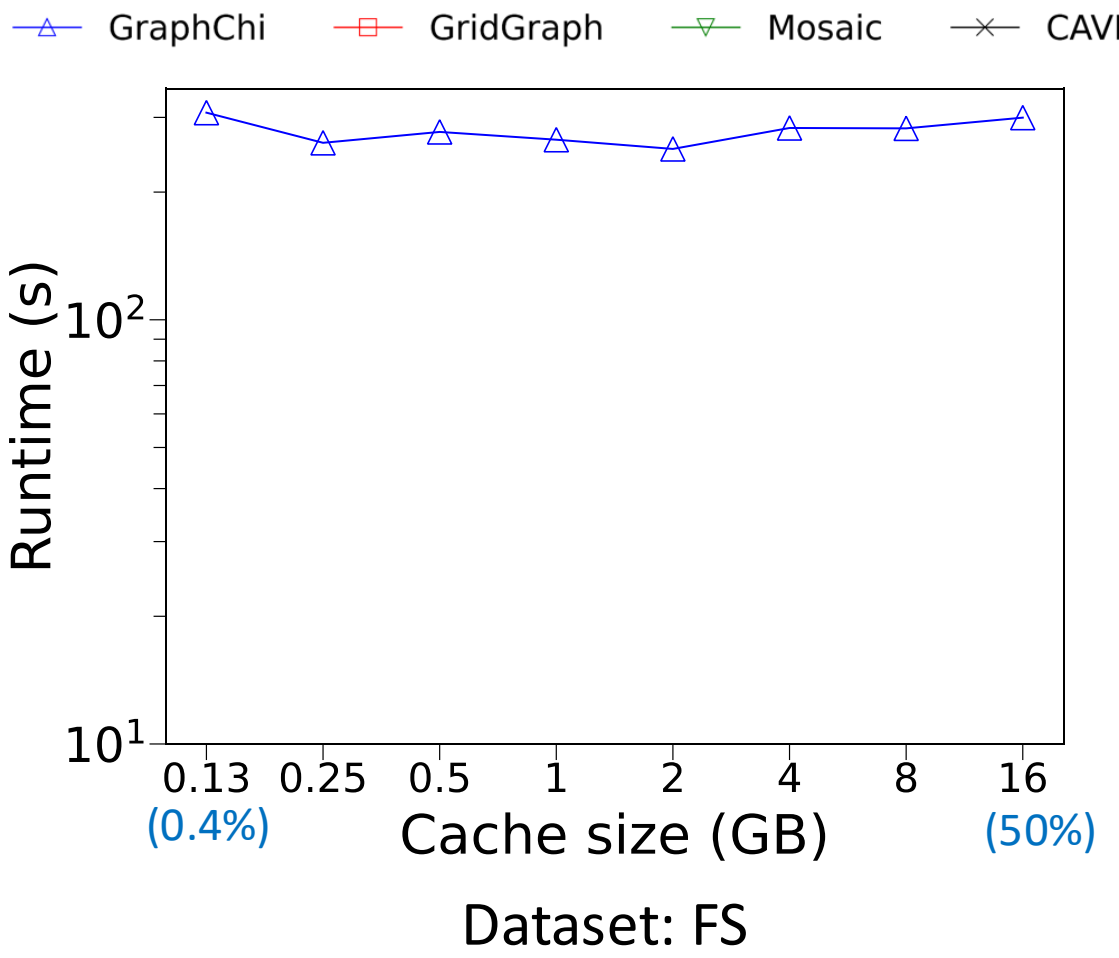
Approaches Used:

GraphChi, GridGraph, Mosaic, CAVE, CAVE_blocked

CAVE's Preprocessing is Efficient

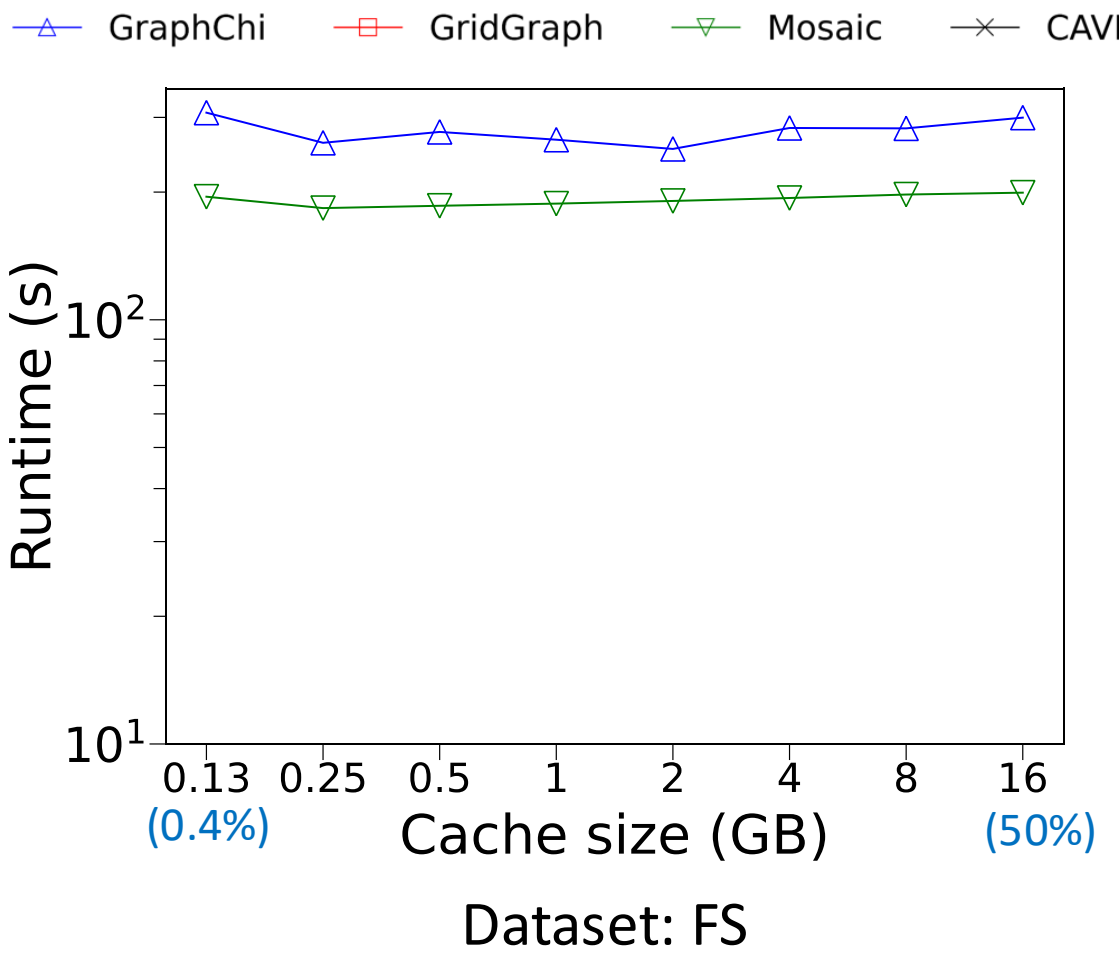
System	Preprocessing Time (s)		Data File Size (GB)	
	Dataset: FS	Dataset: TW	Dataset: FS	Dataset: TW
GraphChi	819	784	8.3	8.4
GridGraph	55	86	84	75
Mosaic	469	370	27	17
CAVE	52	49	14	13

CAVE Performs Efficient PBFS



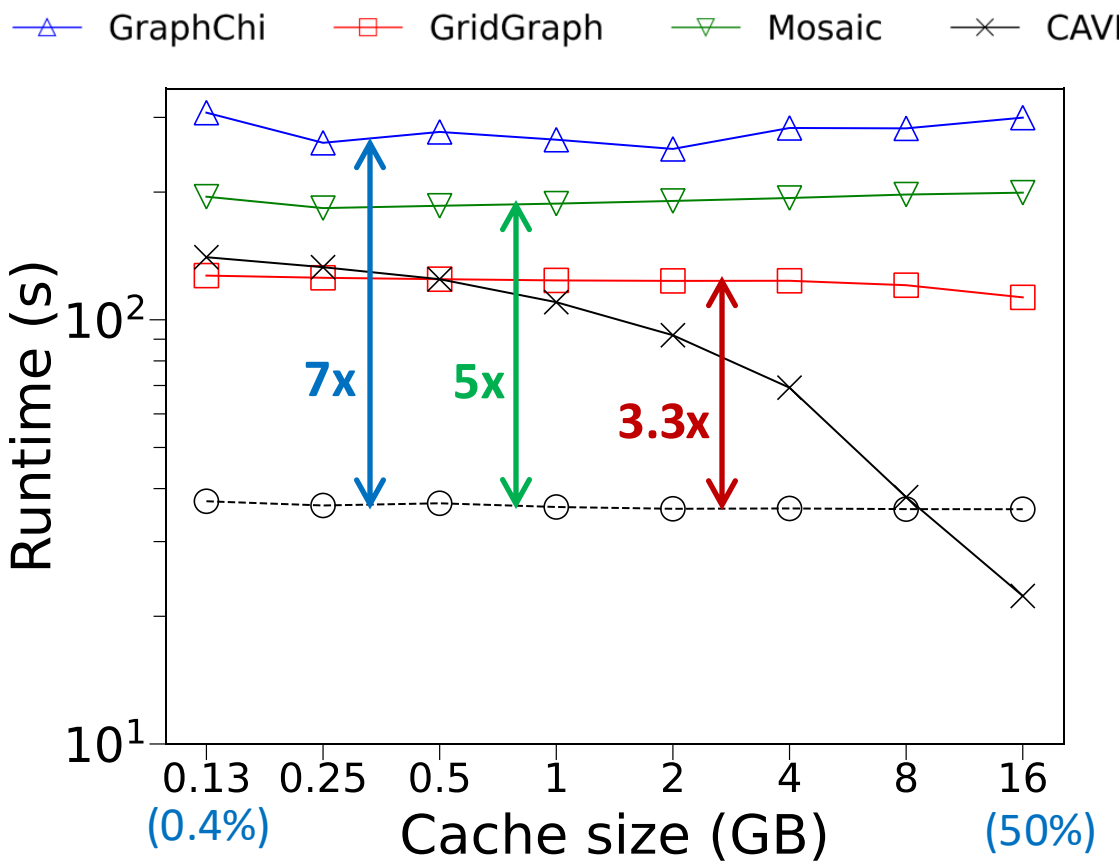
Dataset: FS

CAVE Performs Efficient PBFS

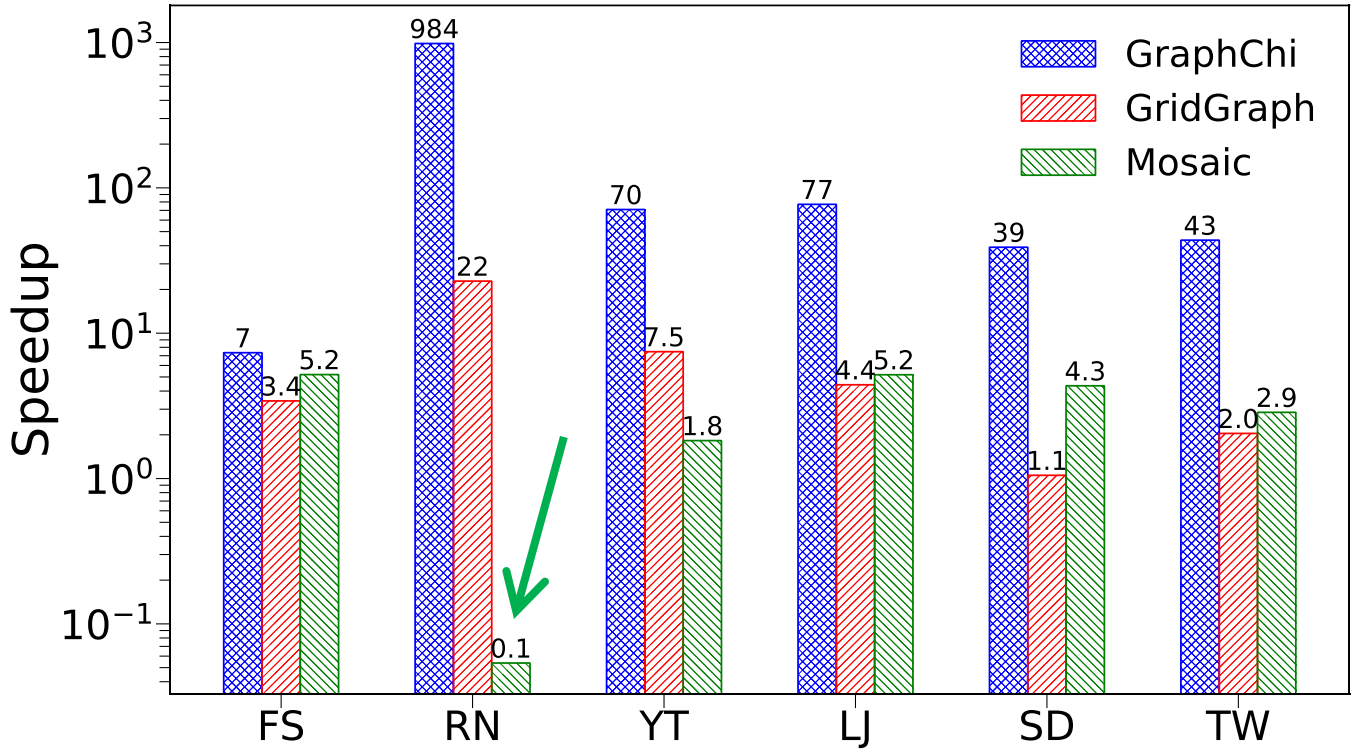


Dataset: FS

CAVE Performs Efficient PBFS



Dataset: FS

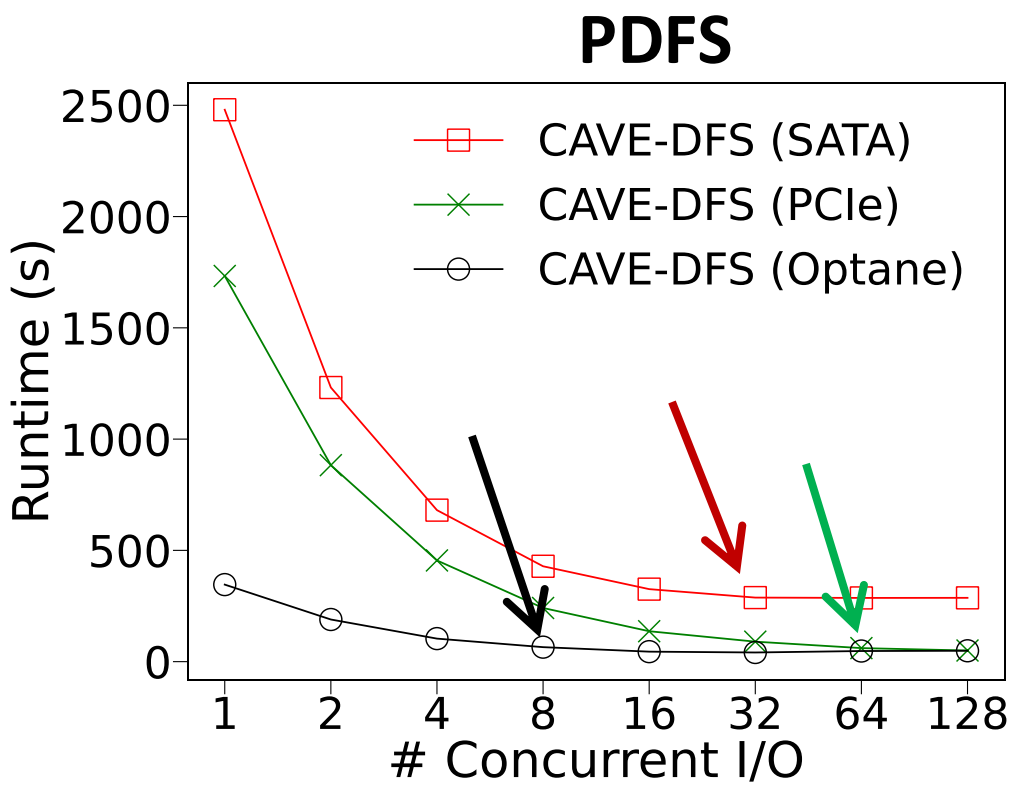


CAVE's Speedup

Both CAVE implementations outperforms GridGraph, Mosaic and GraphChi

CAVE Utilizes Concurrent I/O

Dataset: FS



SATA SSD ($k_r = 25$)
PCIe SSD ($k_r = 80$)
Optane SSD ($k_r = 6$)

Device gets saturated at *optimal concurrency*

Conclusion

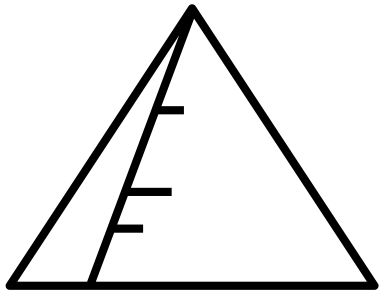
Make *asymmetry and concurrency* part of *algorithm design*

… not simply an engineering optimization

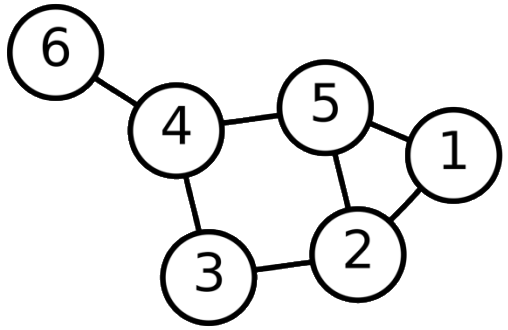
Build algorithms/data structures for storage devices

with asymmetry α and concurrency k

index structures



graph traversal algorithms



bufferpool management