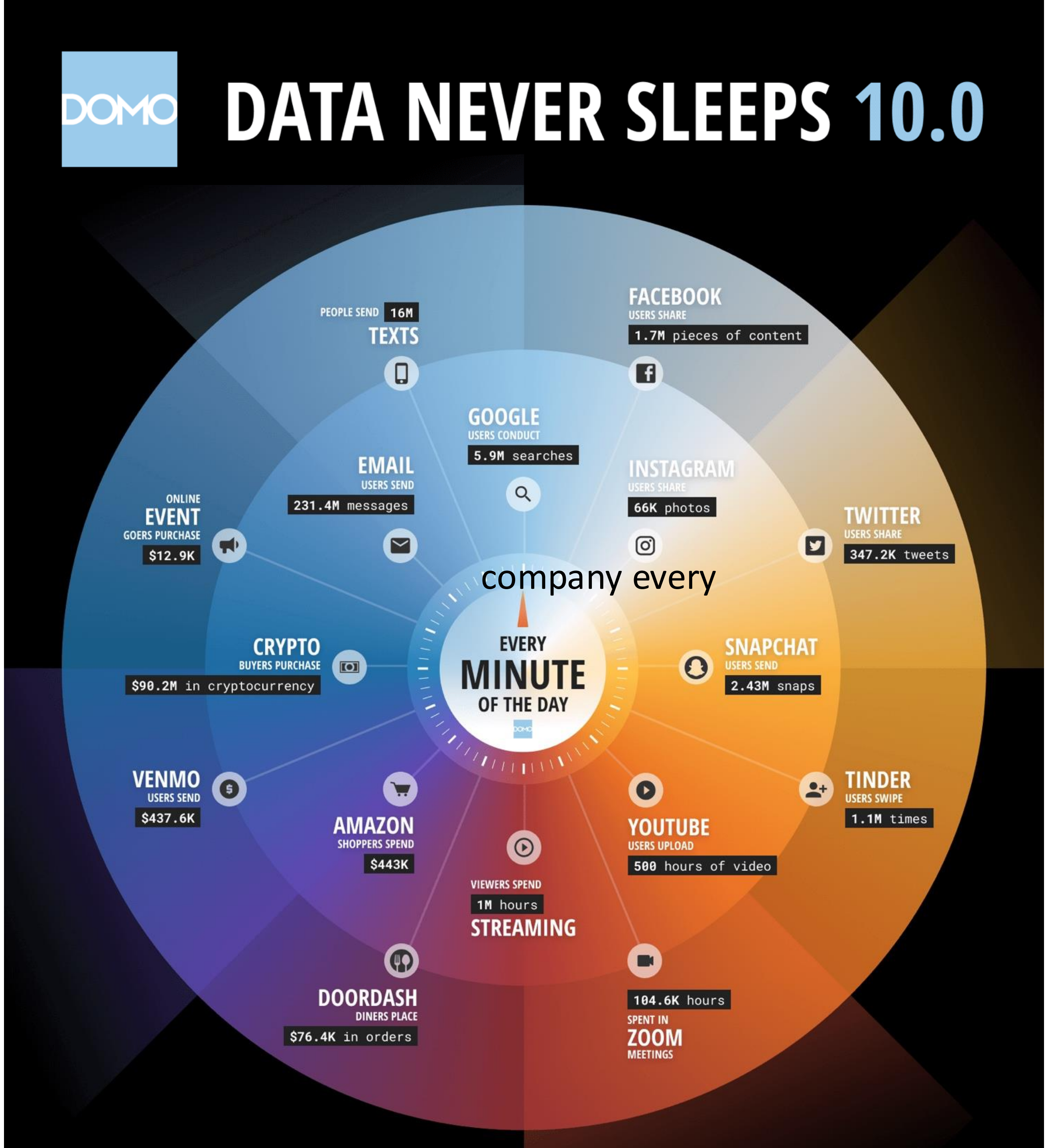
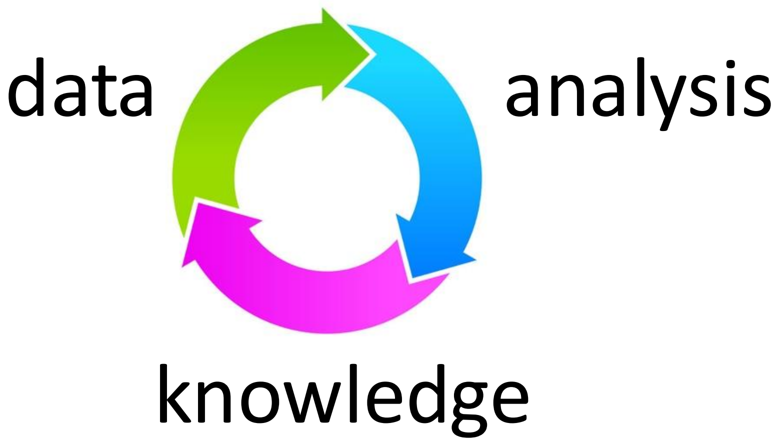
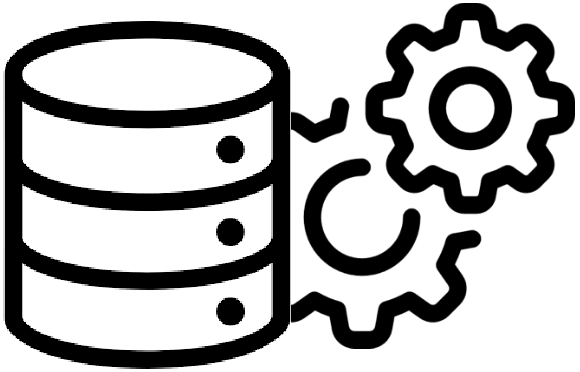
Optimizing Data Systems for Modern Storage Technology

Tarikul Islam Papon
PhD Researcher

BOSTON UNIVERSITY

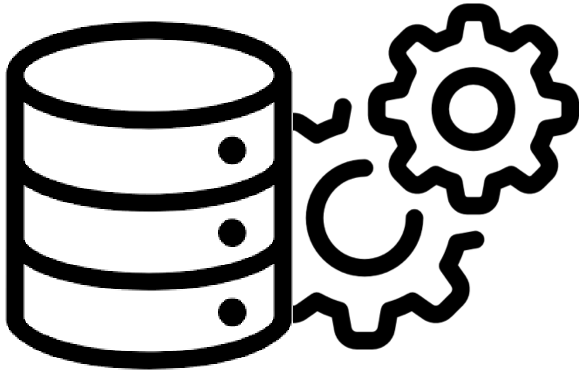




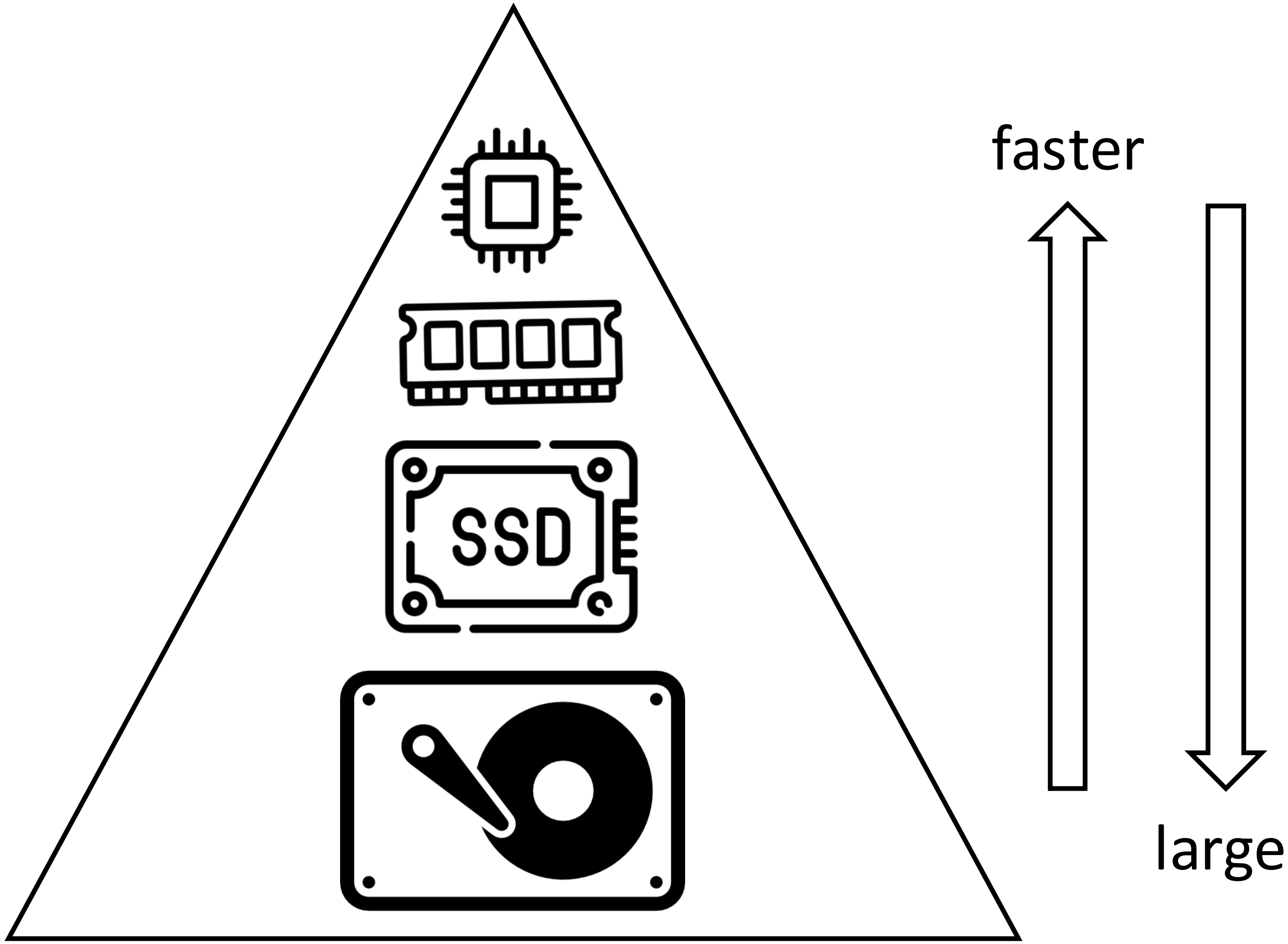


Data Systems

Data Systems & Hardware



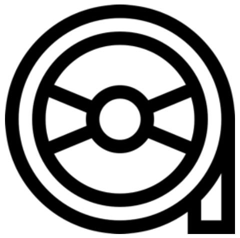
Data Systems



Memory Hierarchy

Hardware Trends

Evolution of Storage Technology

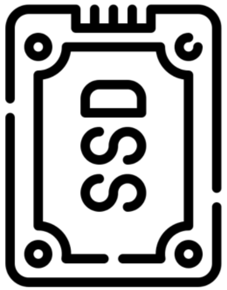


Tape

Floppy

CD

HDD



SSD

“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

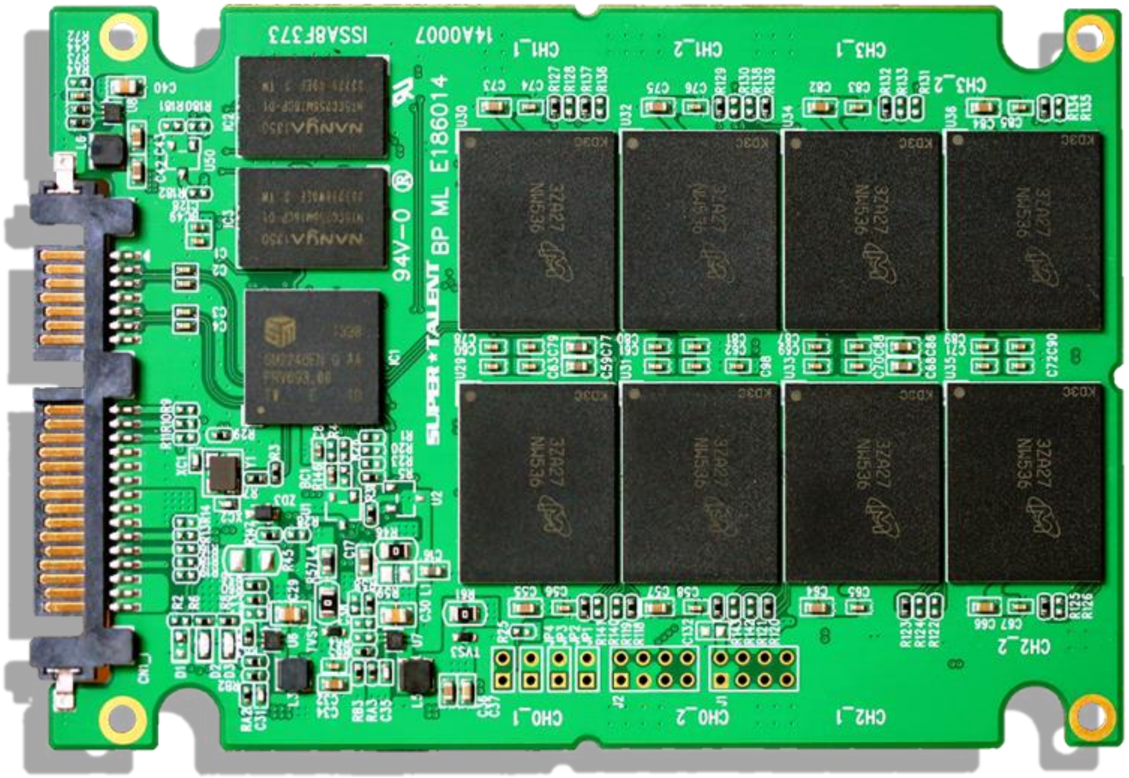
“Tape is Dead. Disk is Tape. Flash is Disk.”

- 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

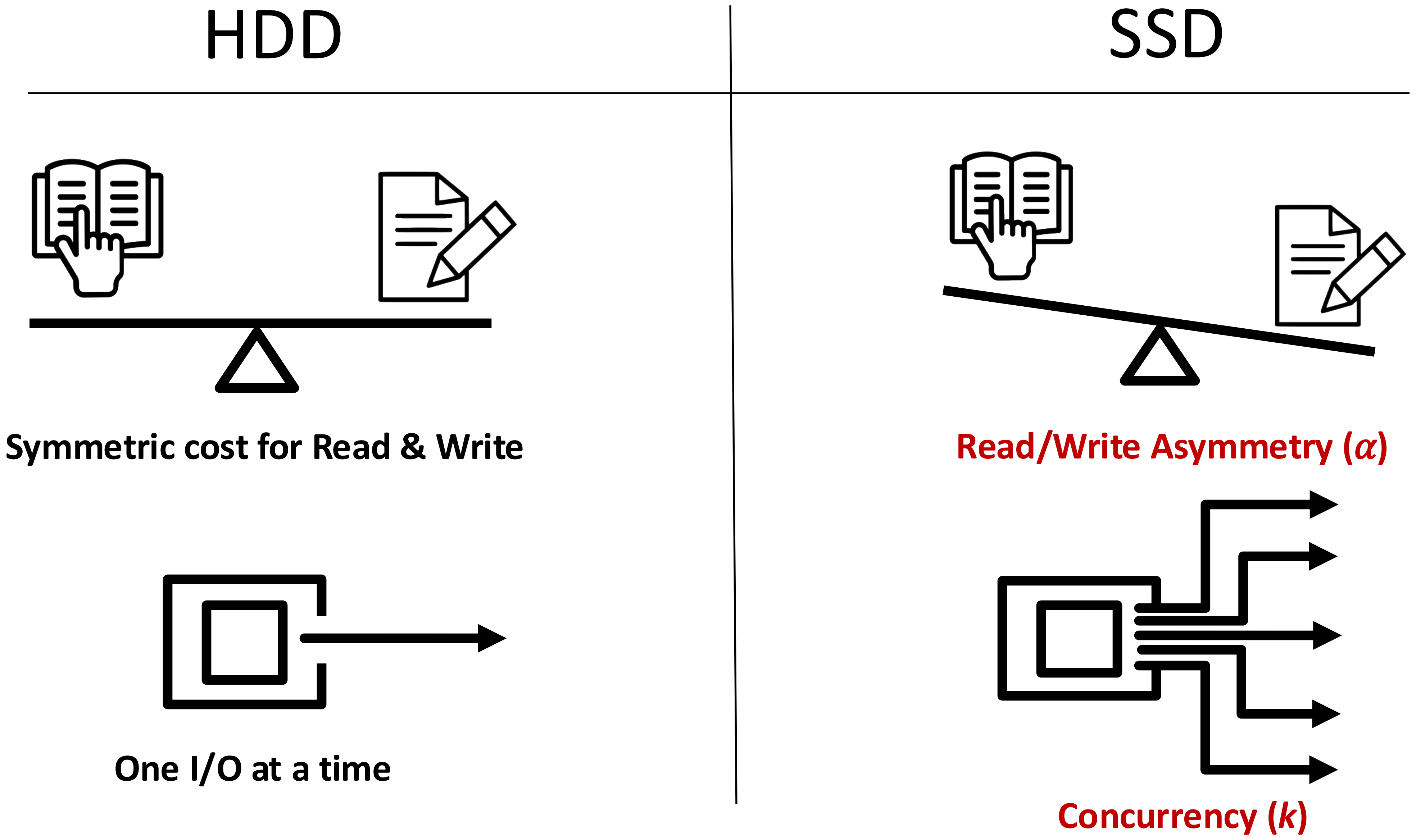


electronic device

fast random access

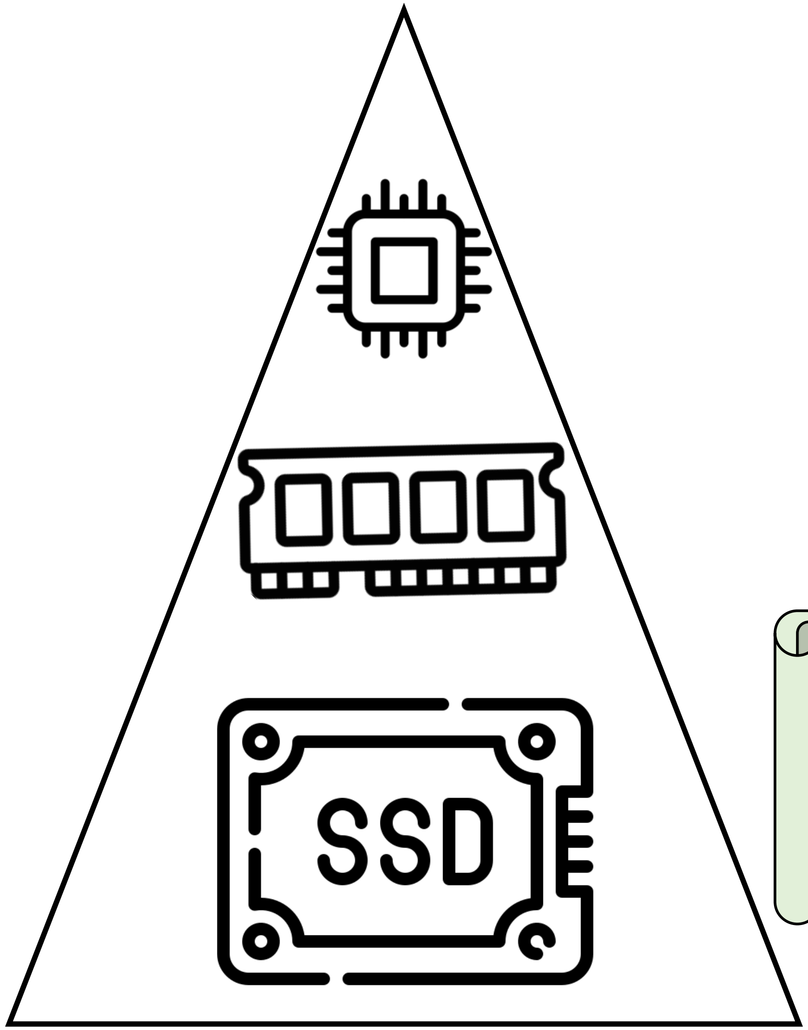
concurrent I/Os

write latency > read latency



HDD
SSD
Symmetric cost for Read & Write
Read/Write Asymmetry (α)
One I/O at a time
Concurrency (k)

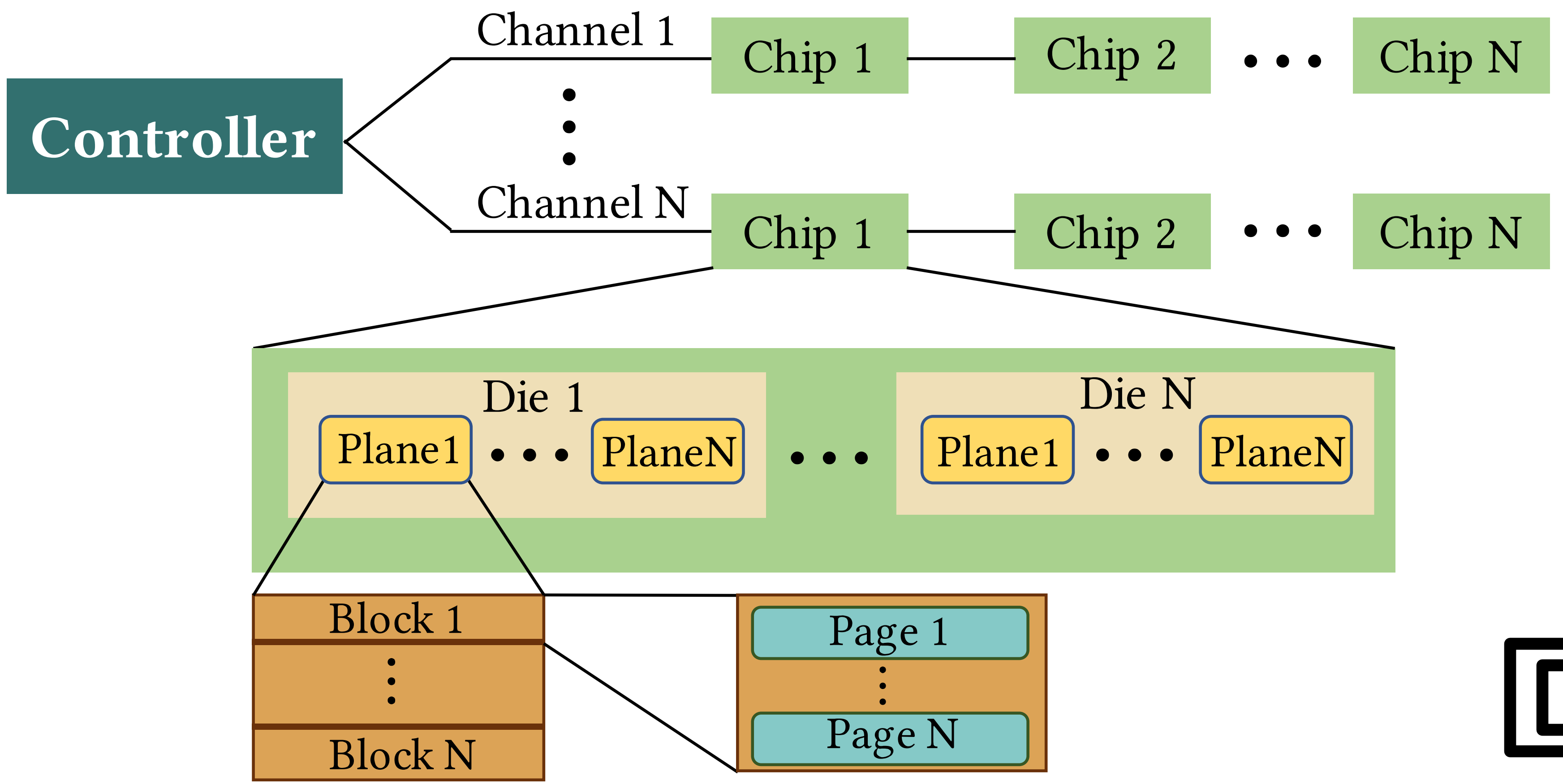
Goal: Developing Storage-Aware Data Systems

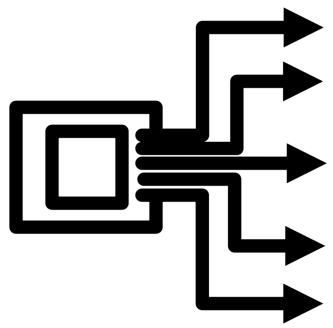


Tailor Data Systems
for **SSD Asymmetry**
& **Concurrency**

PIO [CIDR'21+DaMoN@SIGMOD'21]
ACE Bufferpool [IEEE ICDE '23]
CAVE Graph Engine [SIGMOD '24]
SSD-Aware Systems [IEEE ICDE '24]
ReStore [Under Review]

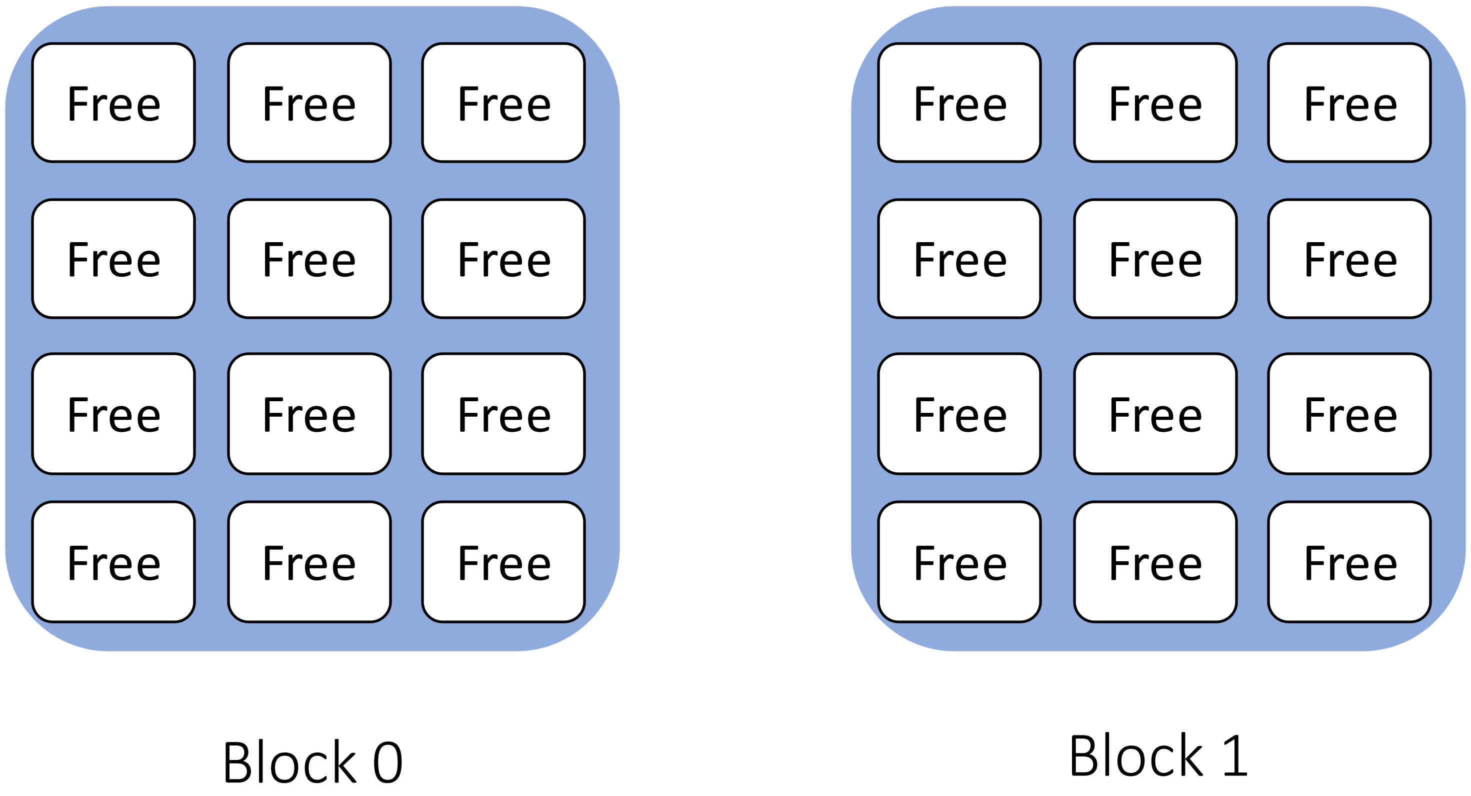
SSD Concurrency



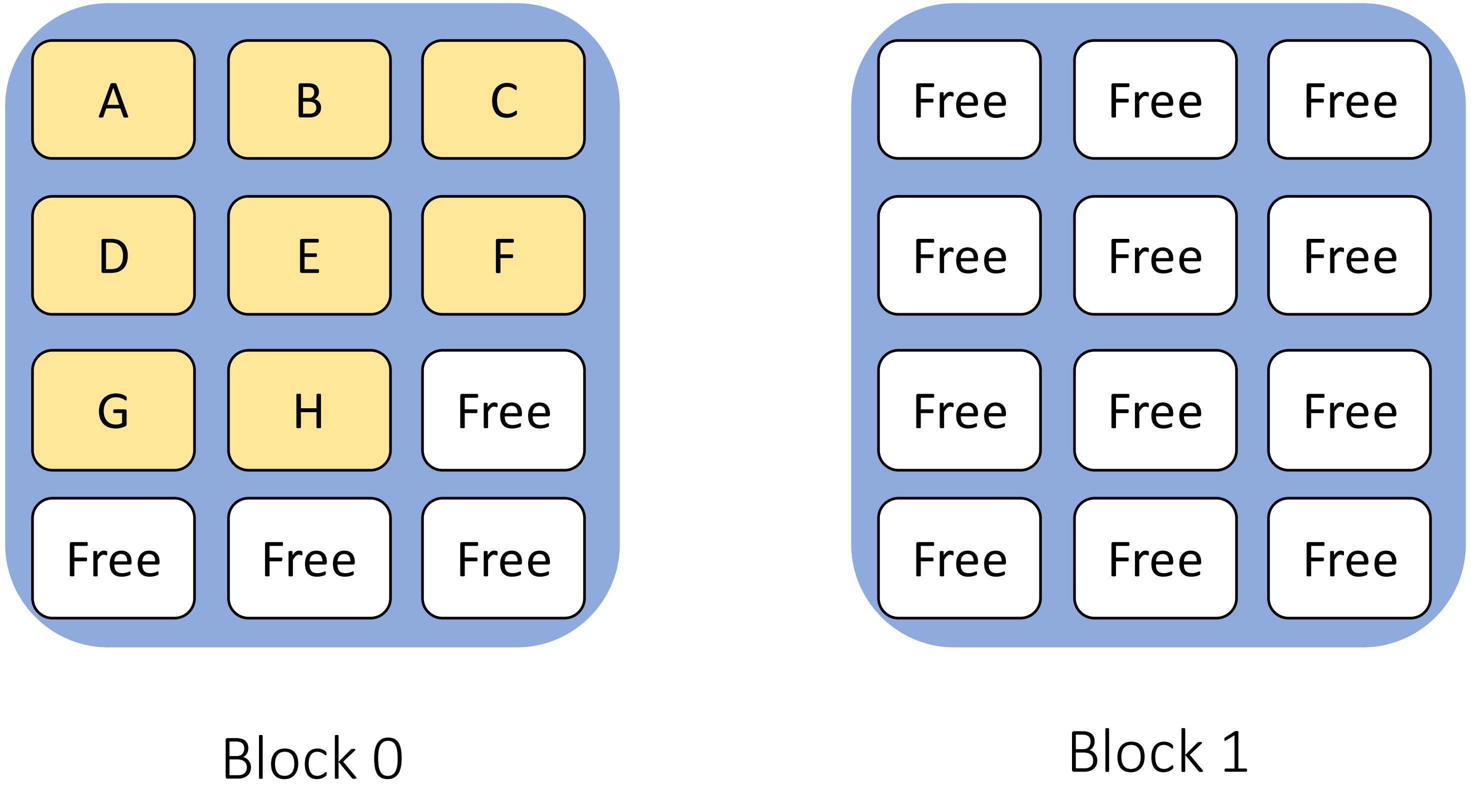


Parallelism at different levels (channel, chip, die, plane block, page)

Writes in SSD



Writes in SSD

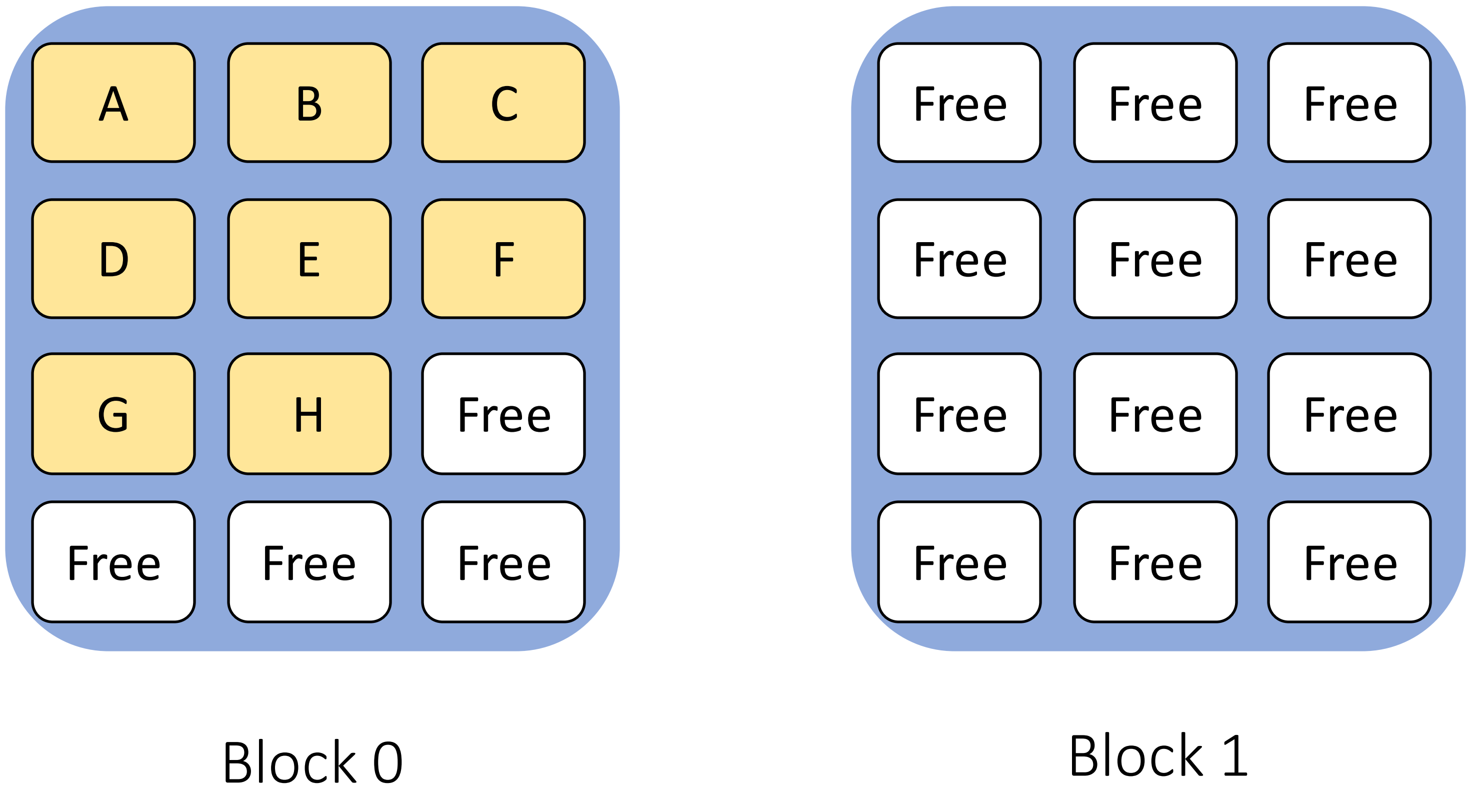


Writing in a free page isn't costly!

Writes in SSD

Update

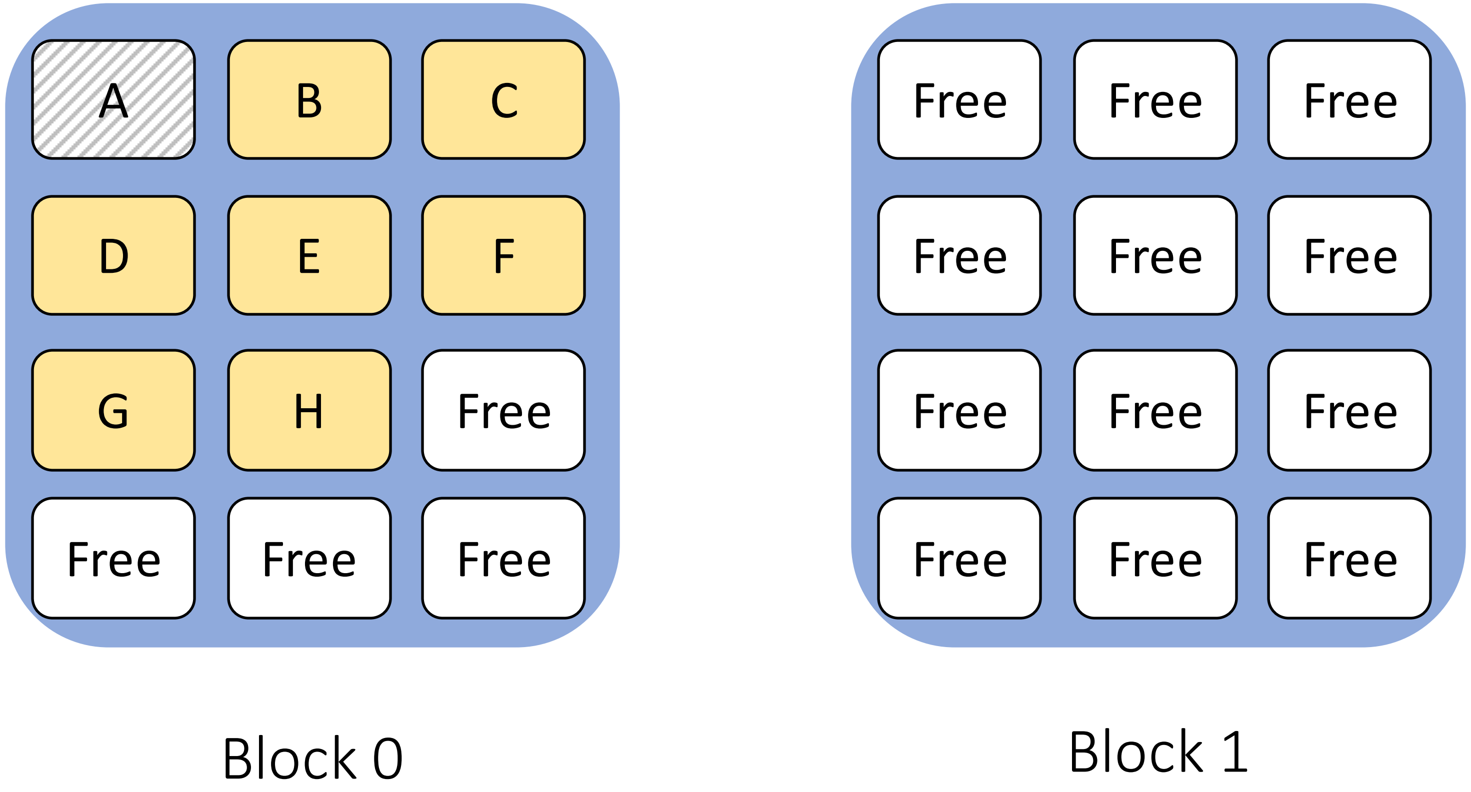
A, B, C, D



Writes in SSD

Update

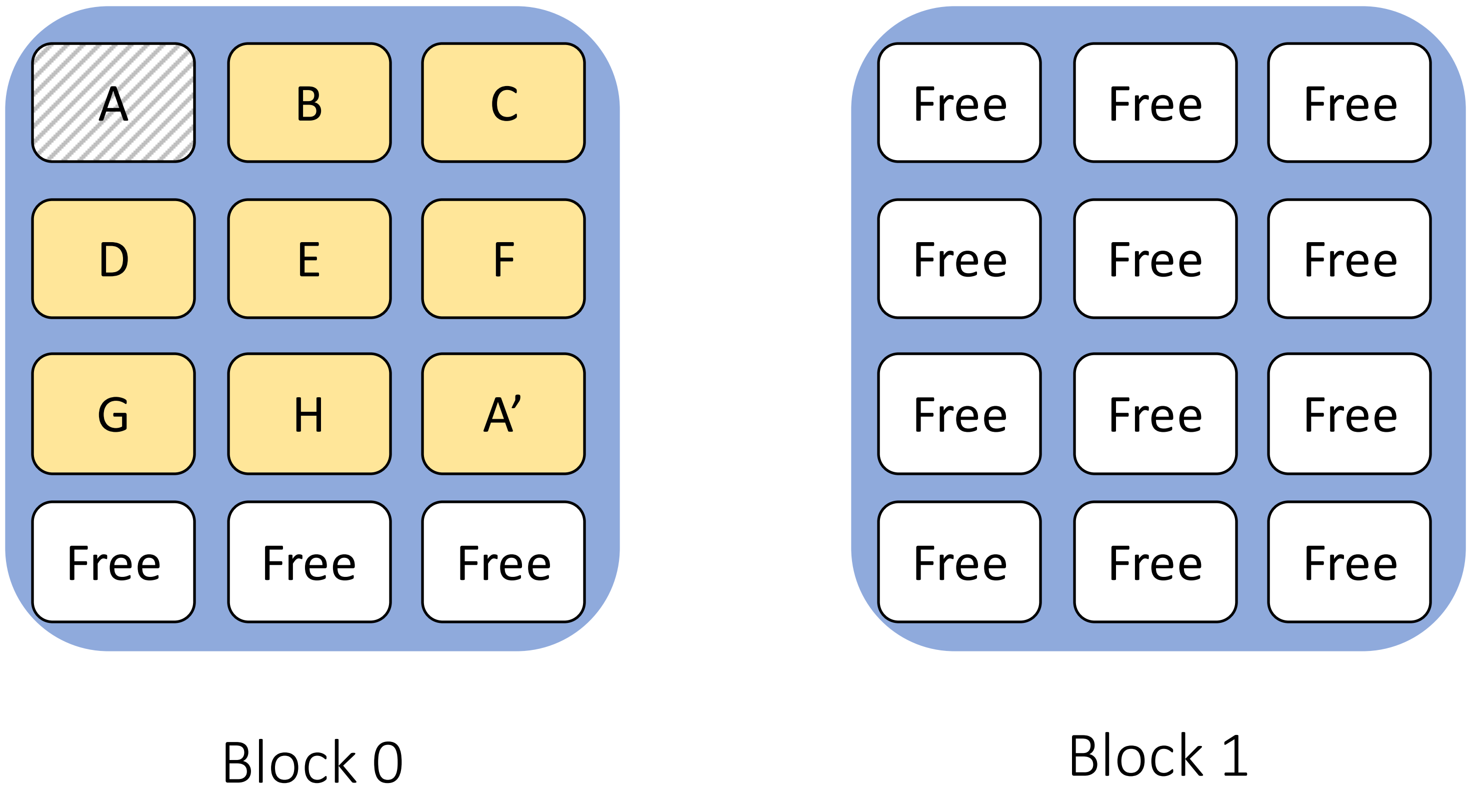
A, B, C, D



Writes in SSD

Update

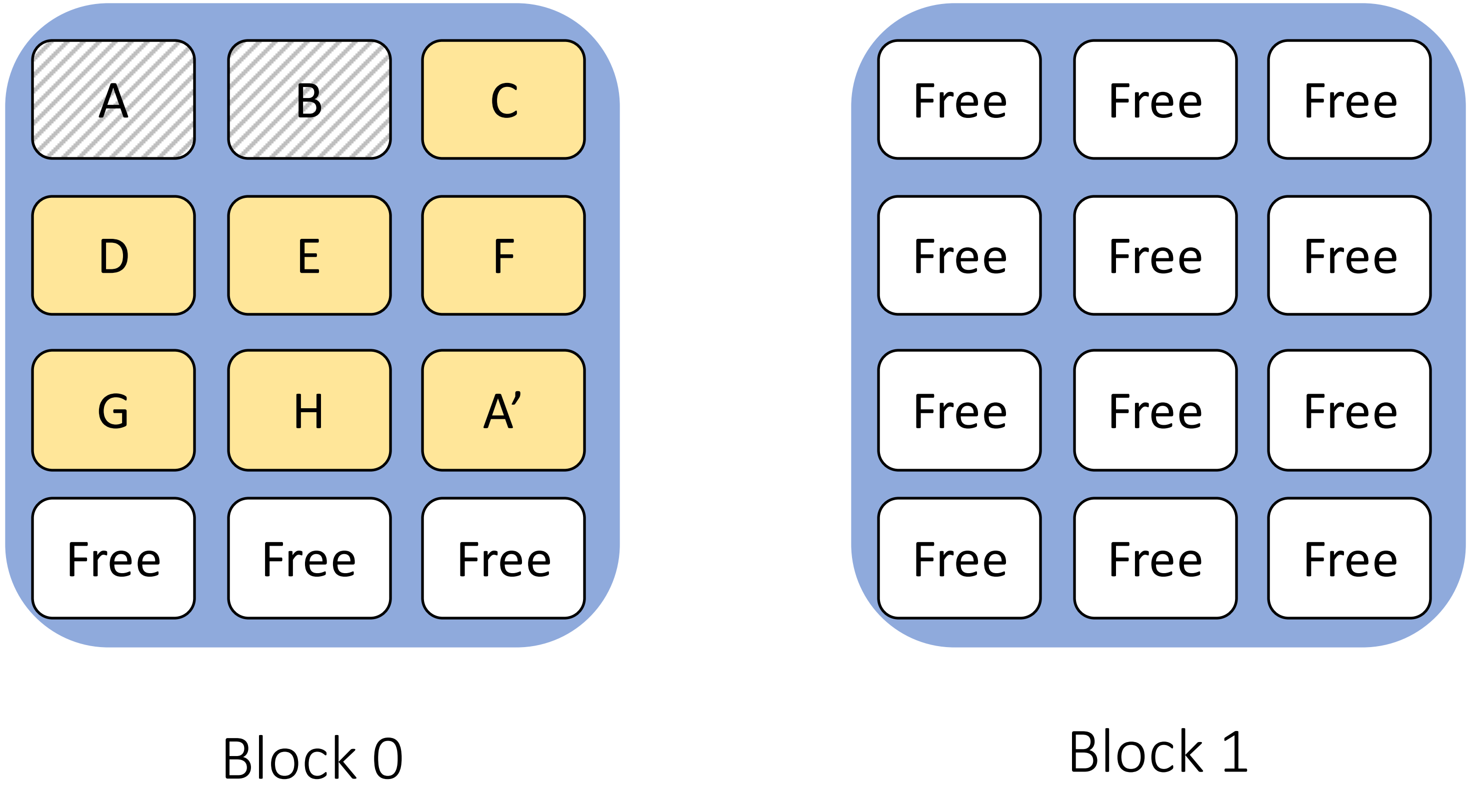
A, B, C, D



Writes in SSD

Update

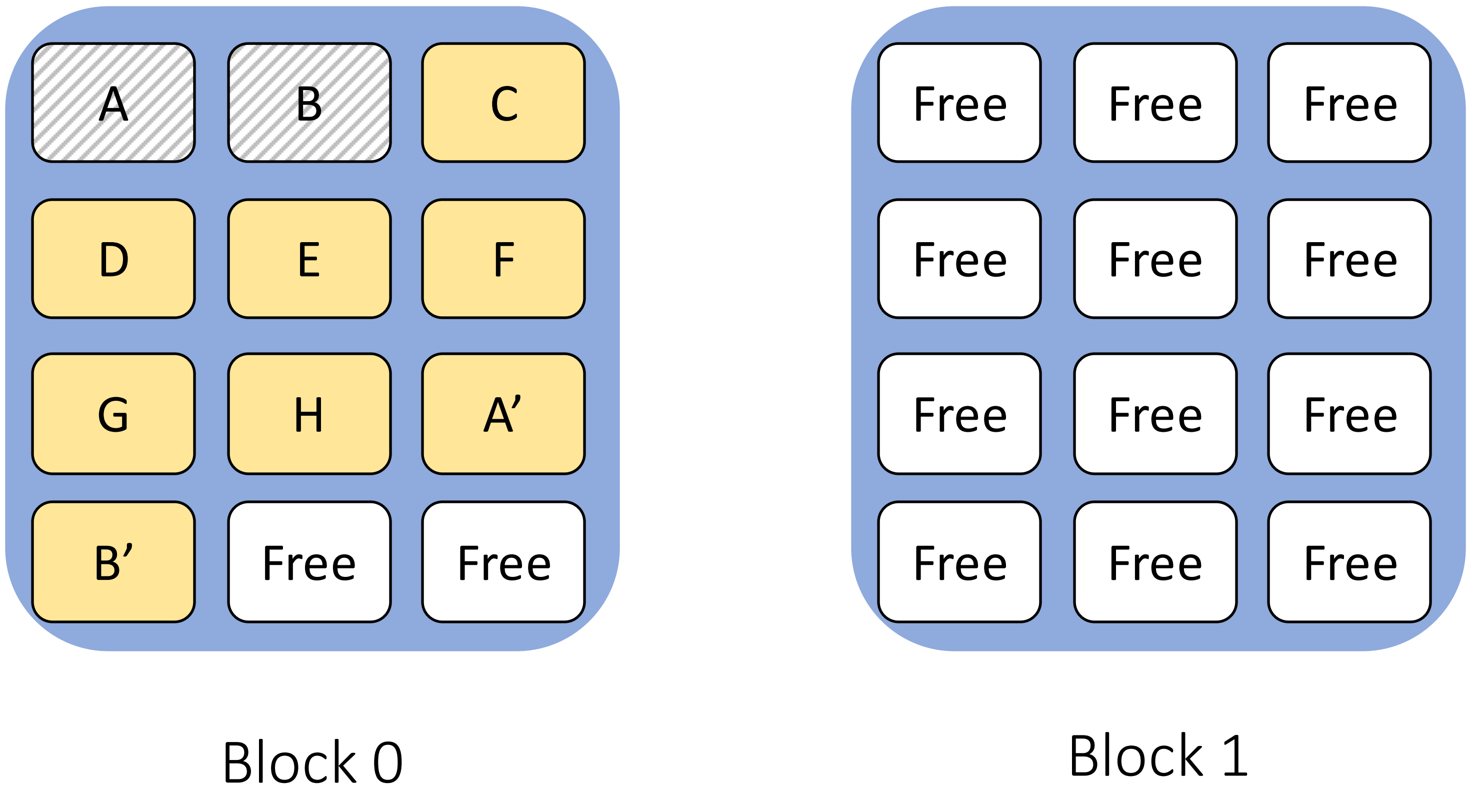
A, **B**, C, D



Writes in SSD

Update

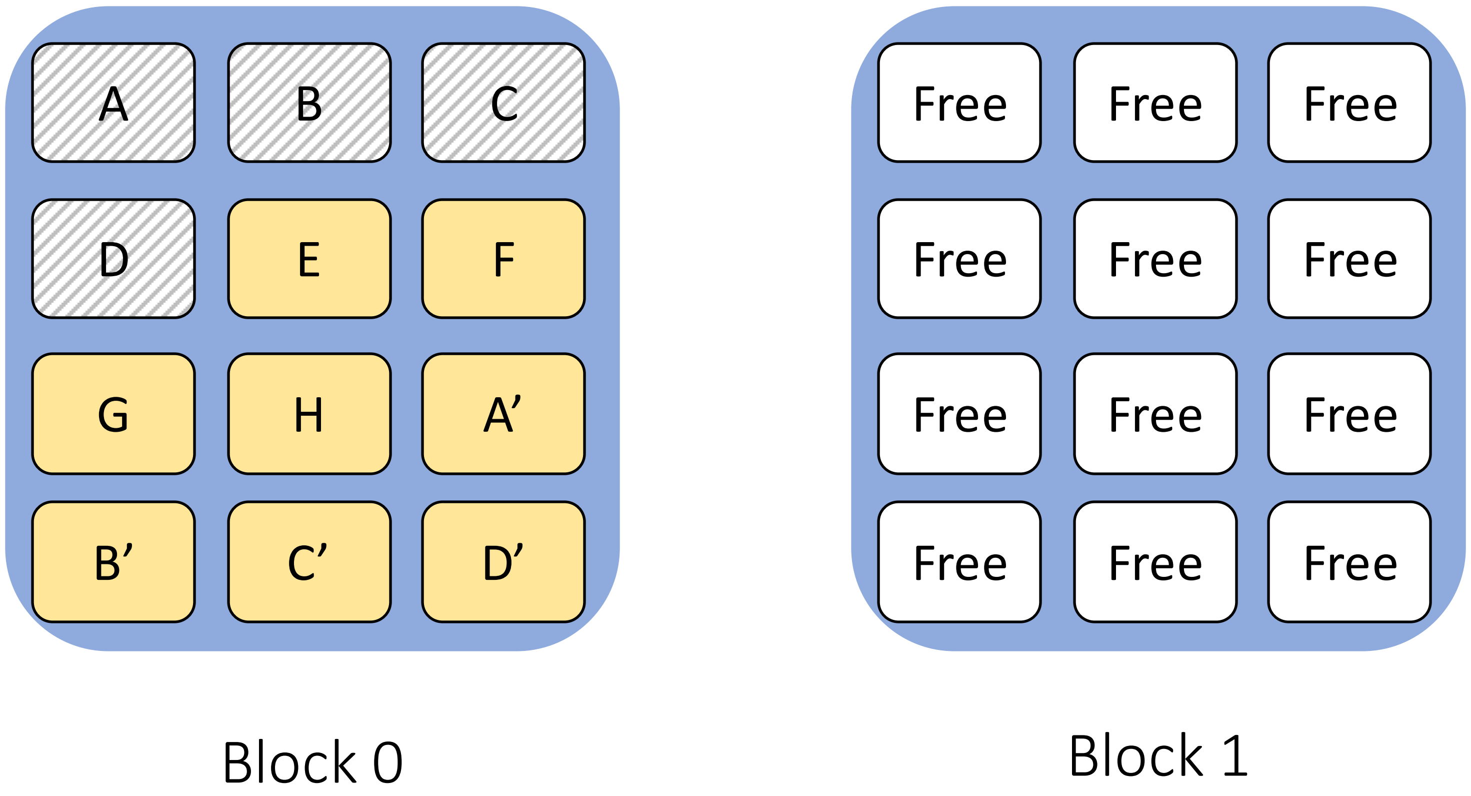
A, **B**, C, D



Writes in SSD

Update

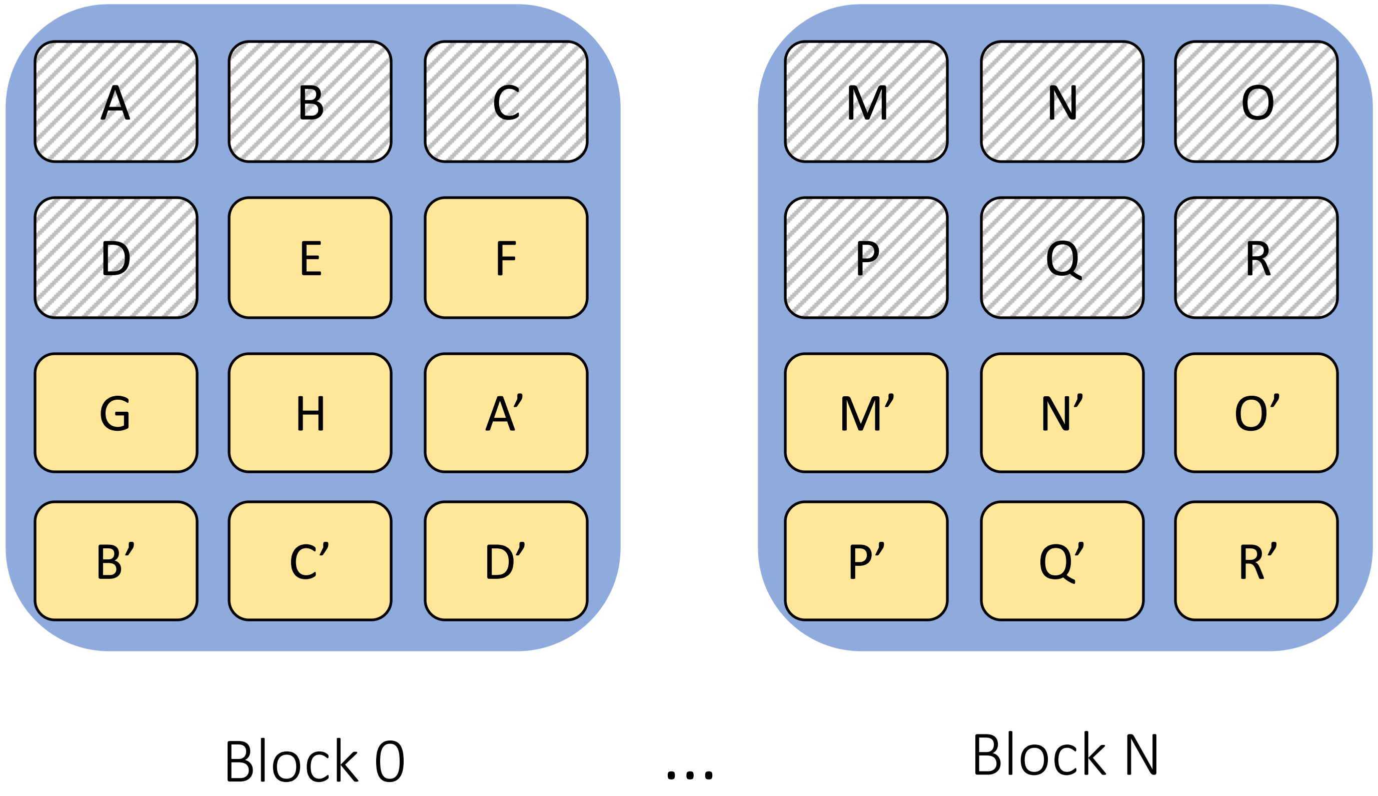
A, B, C, D



Not all updates are costly!

Writes in SSD

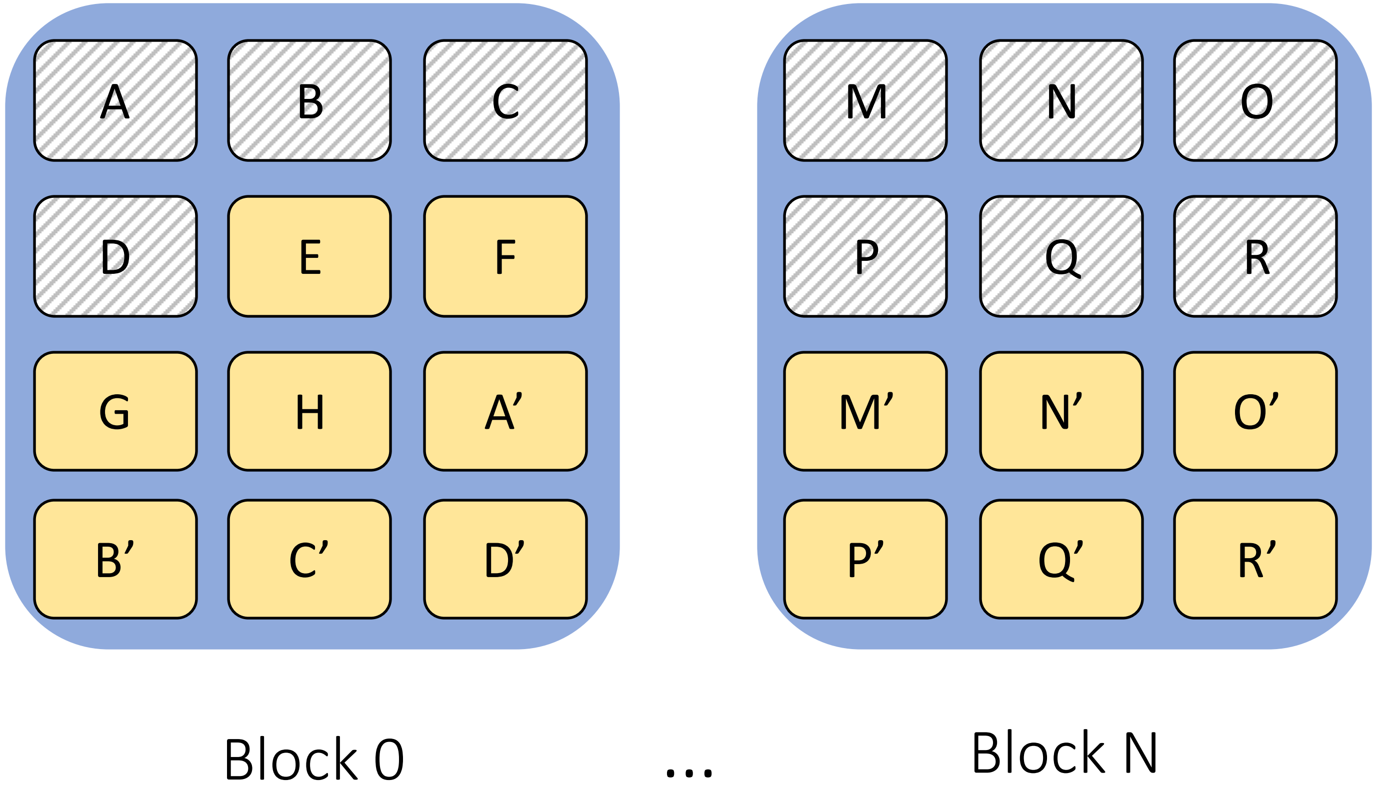
What if there is no space?



Writes in SSD

What if there is no space?

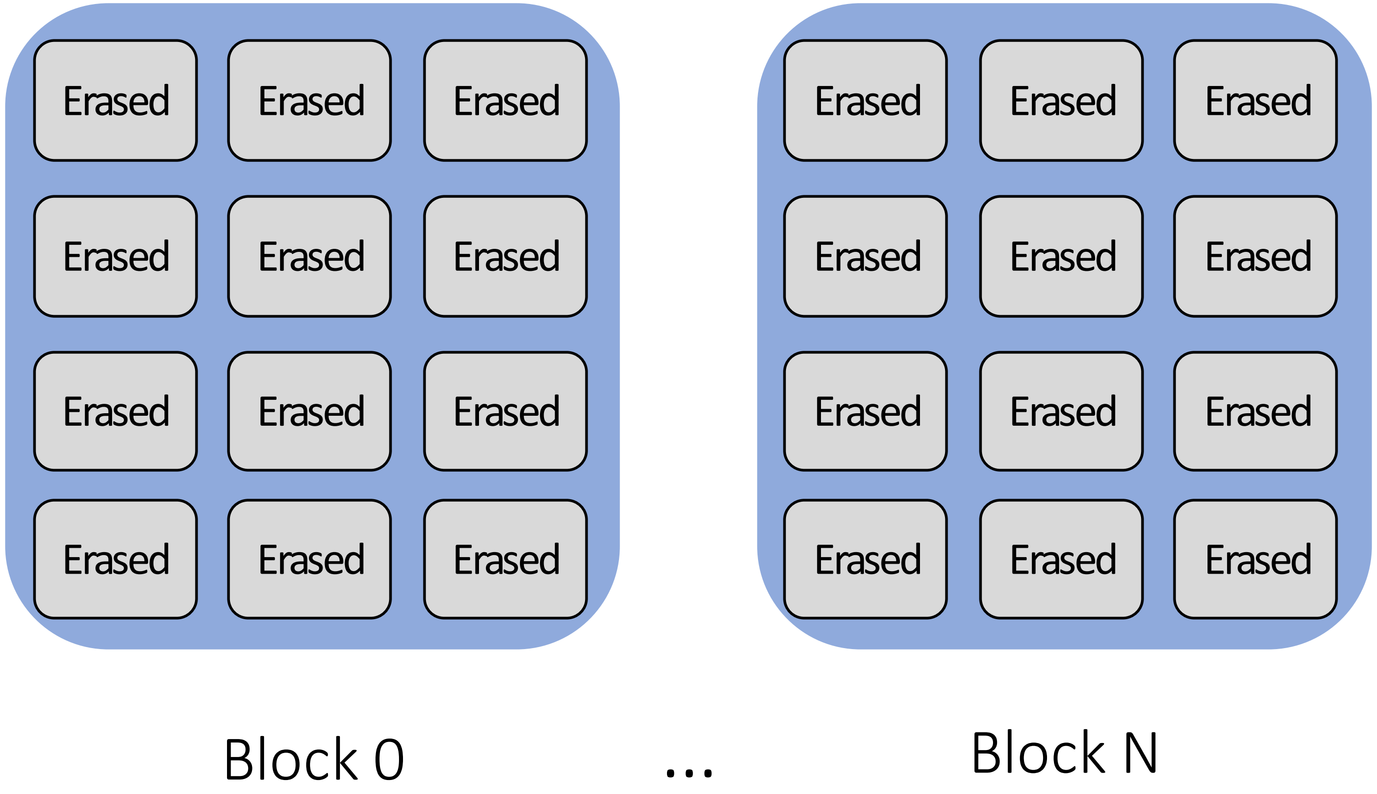
Garbage Collection!



Writes in SSD

What if there is no space?

Garbage Collection!



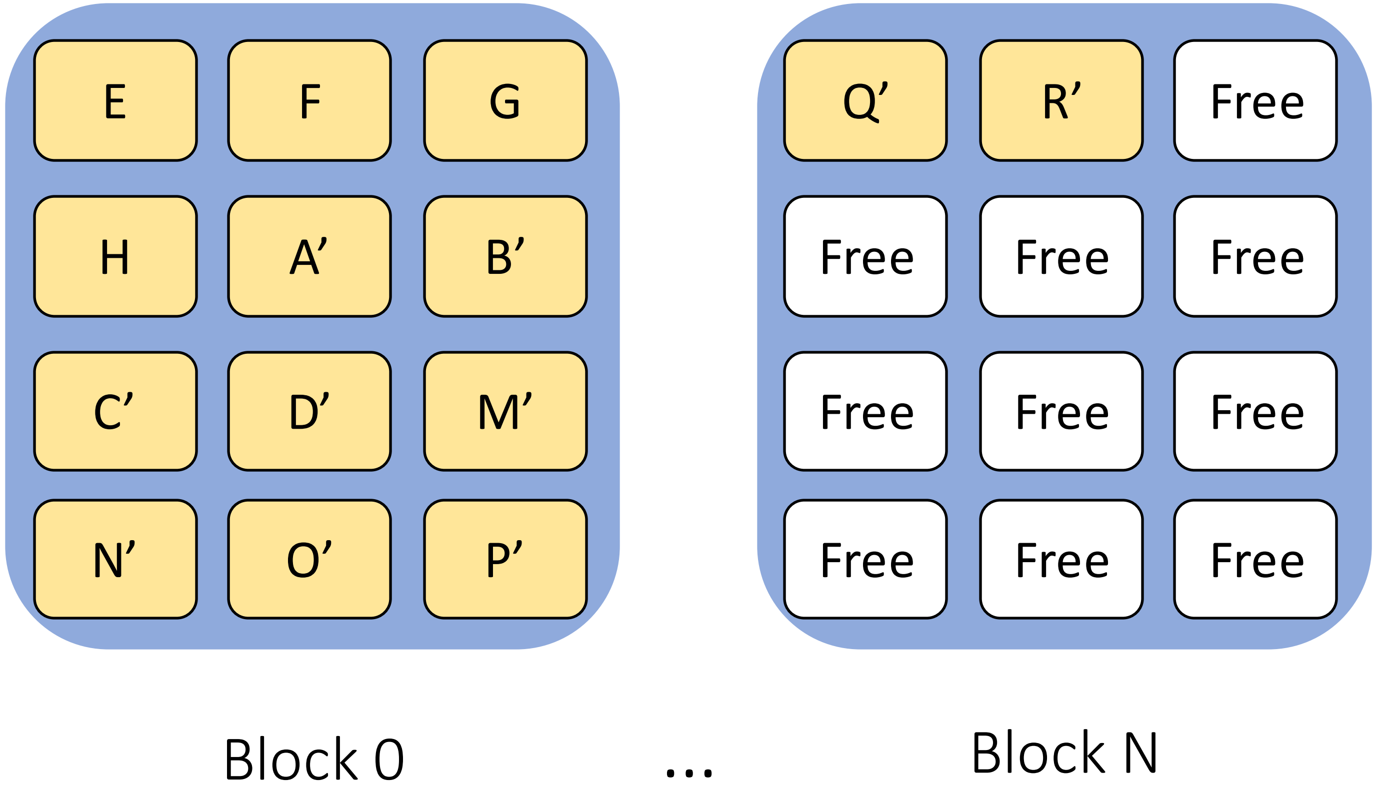
Valid pages:

E	F	G	H	A’	B’	C’	D’	M’	N’	O’	P’	Q’	R’

Writes in SSD

What if there is no space?

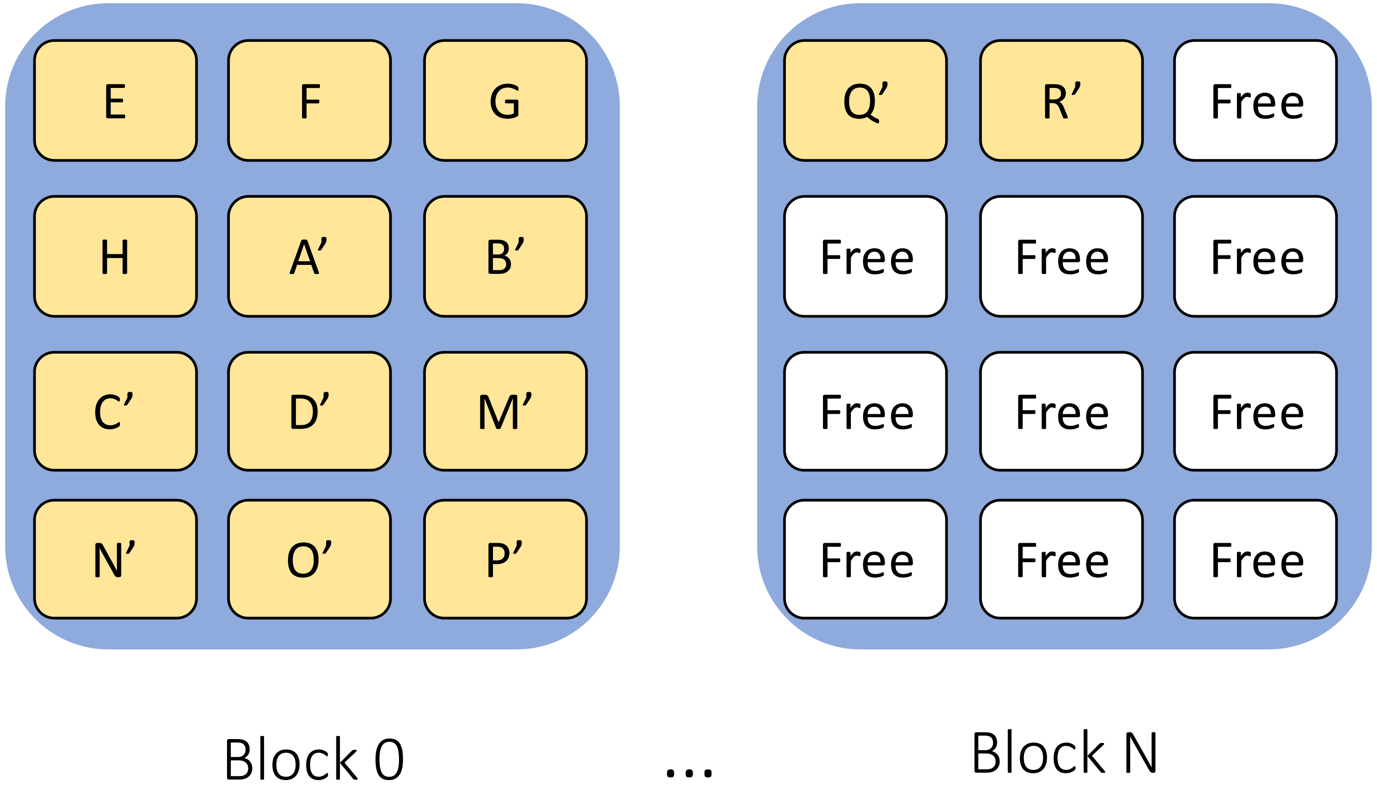
Garbage Collection!



Read/Write Asymmetry in SSD

What if there is no space?

Garbage Collection!



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

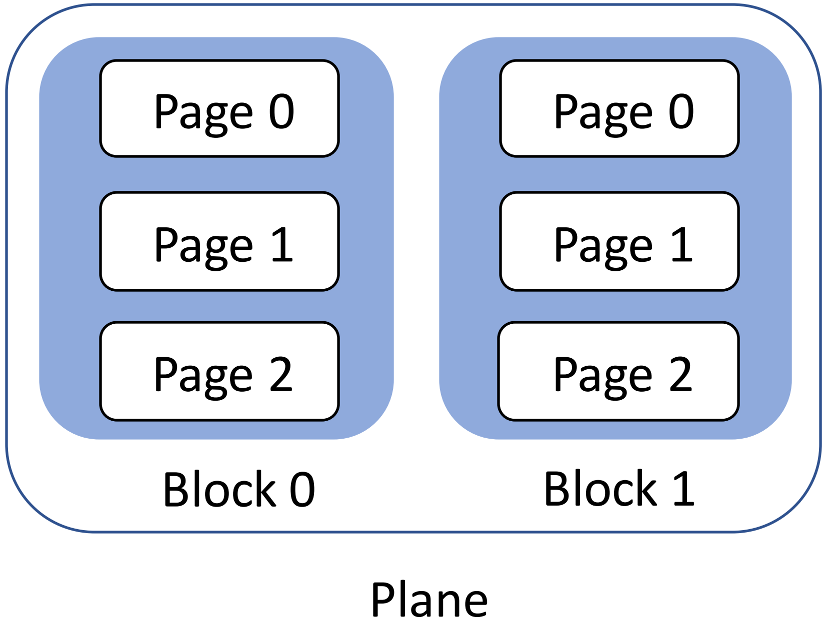
Read/Write Asymmetry

Out-of-place updates cause invalidation

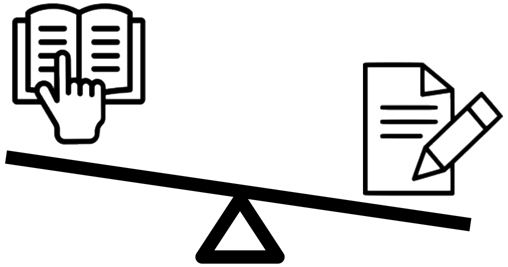
"Erase before write" approach

Garbage Collection

Larger erase granularity

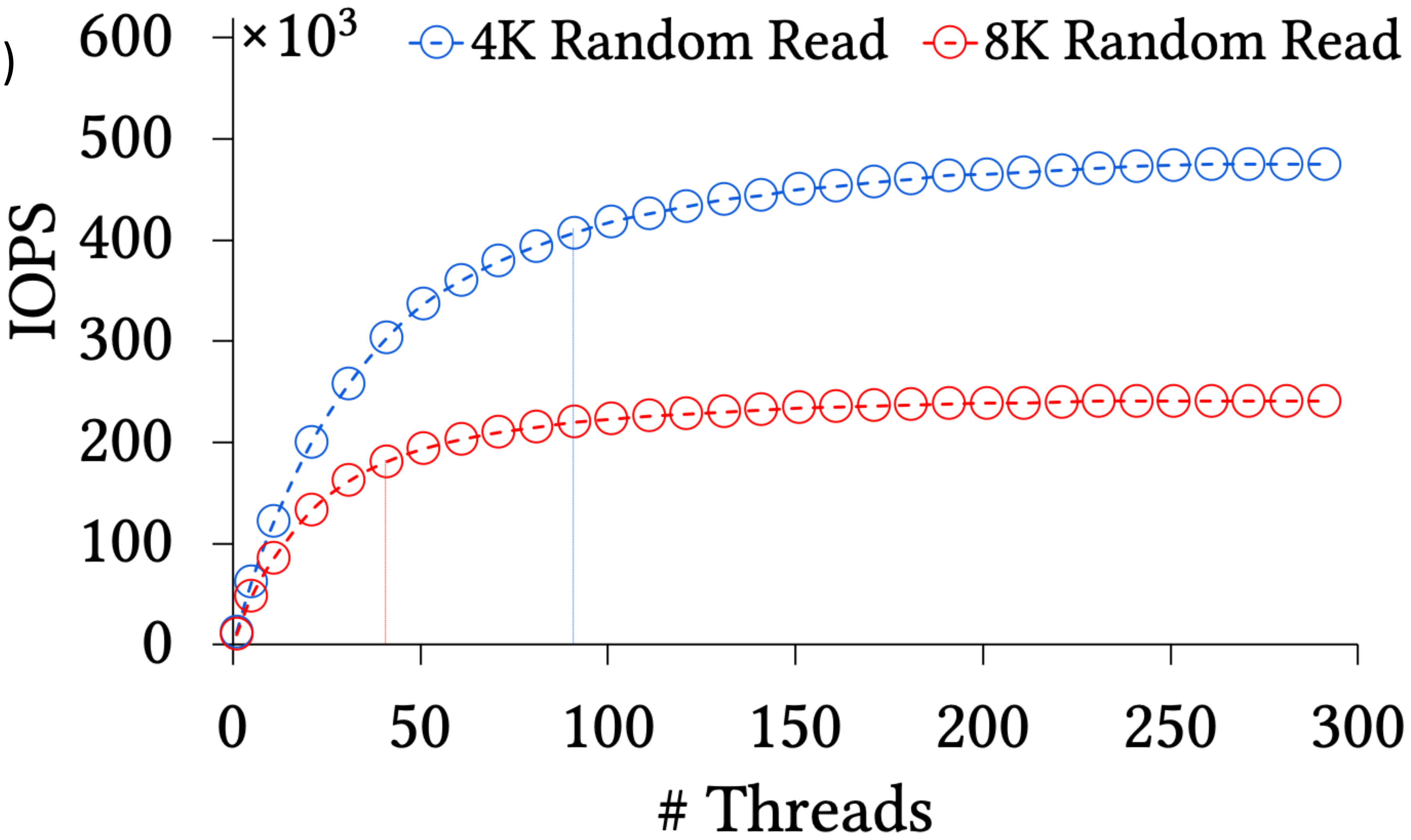


All these results in higher amortized write cost



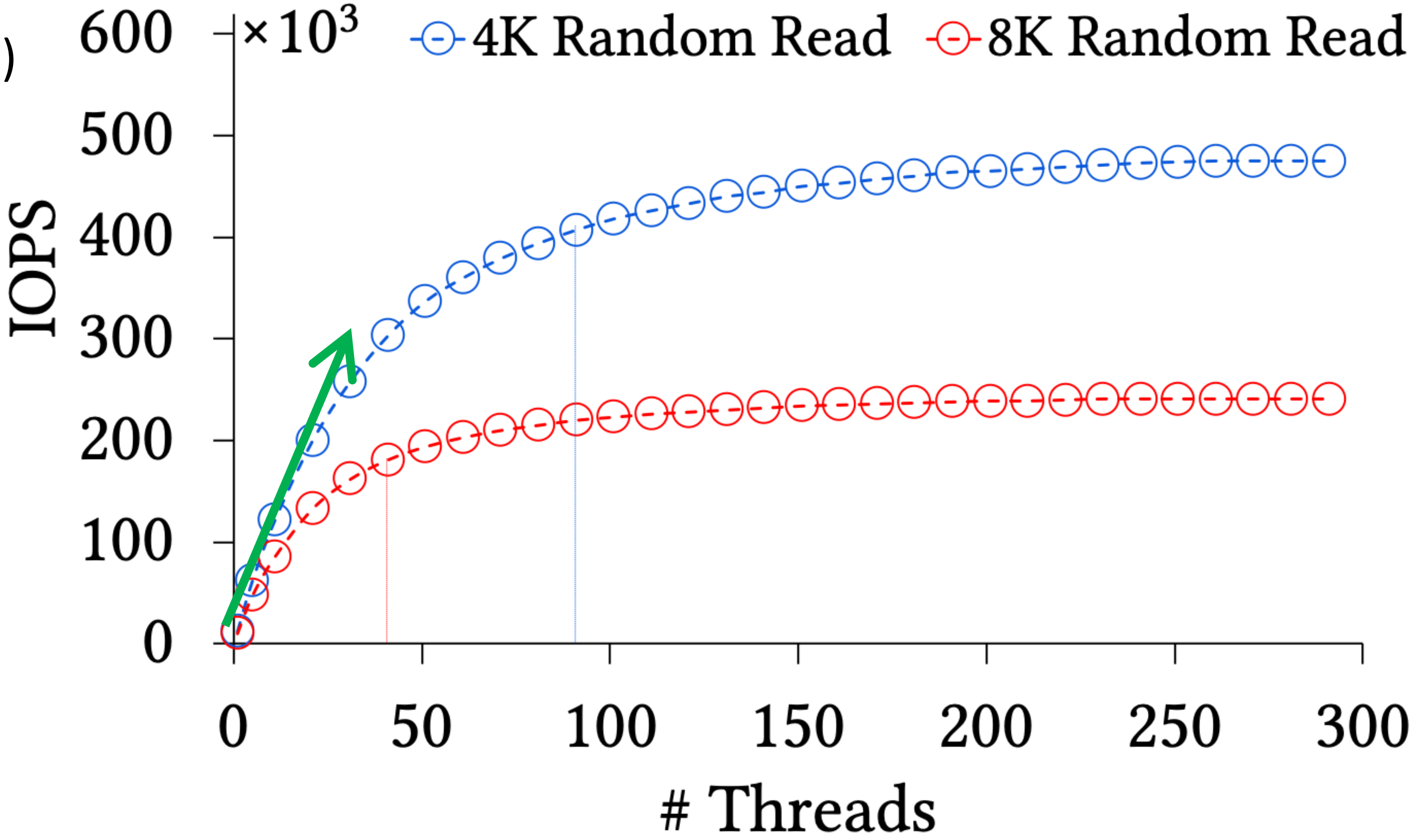
Quantifying Asymmetry & Concurrency

Device
PCIe SSD - P4510 (1TB)



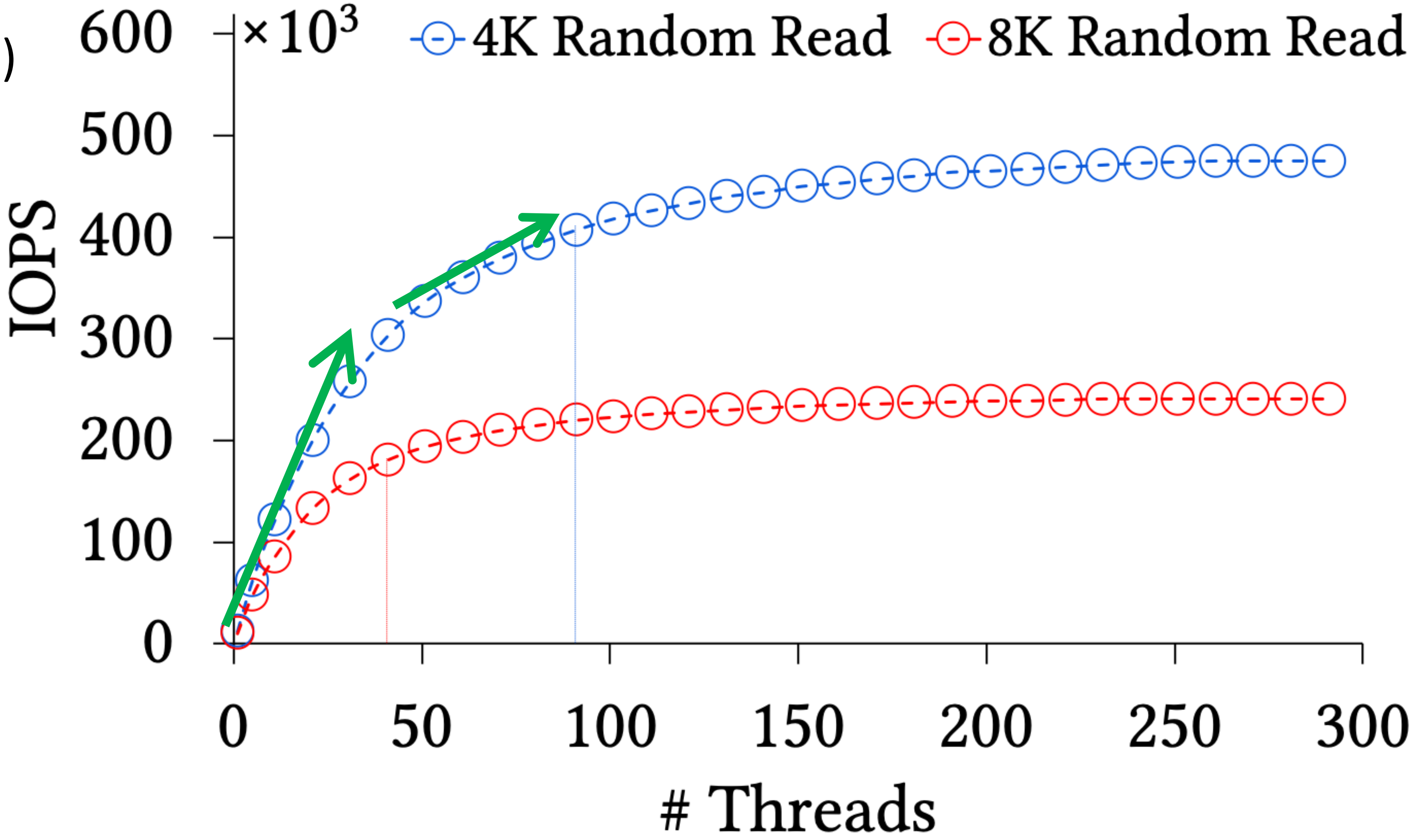
Quantifying Asymmetry & Concurrency

Device
PCIe SSD - P4510 (1TB)



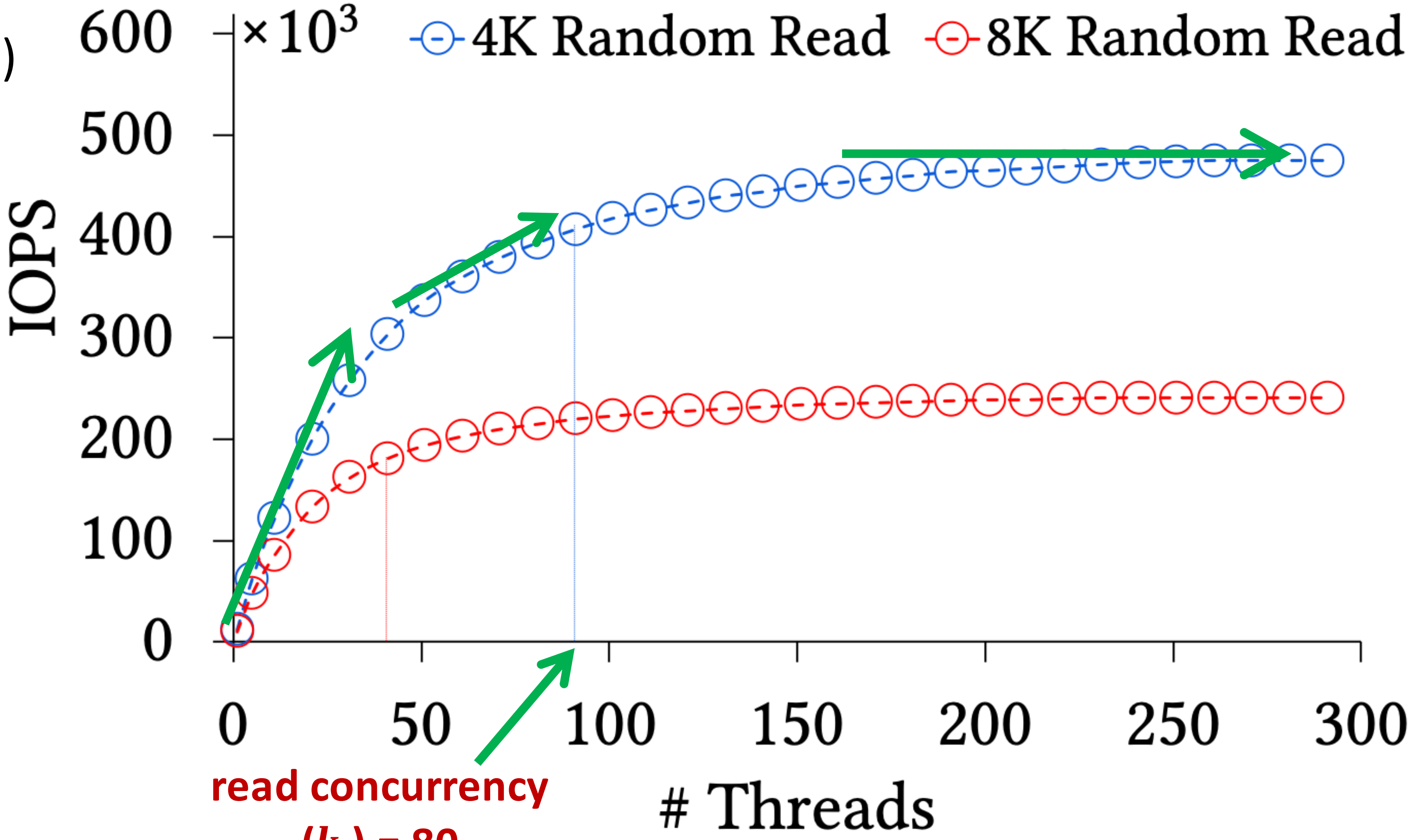
Quantifying Asymmetry & Concurrency

Device
PCIe SSD - P4510 (1TB)



Quantifying Asymmetry & Concurrency

Device
PCIe SSD - P4510 (1TB)

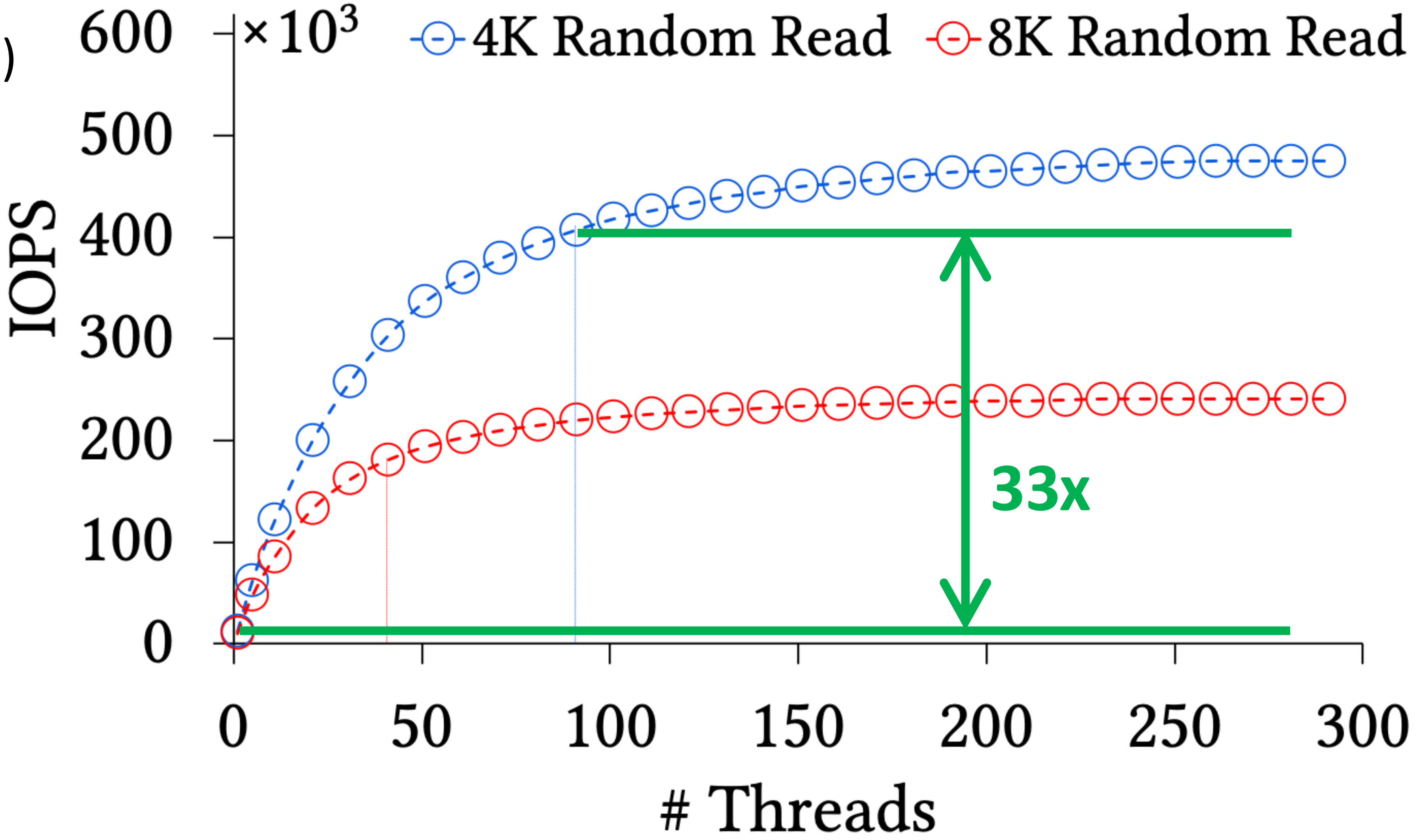


read concurrency (k_r) = 80

Quantifying Asymmetry & Concurrency

Device

PCIe SSD - P4510 (1TB)



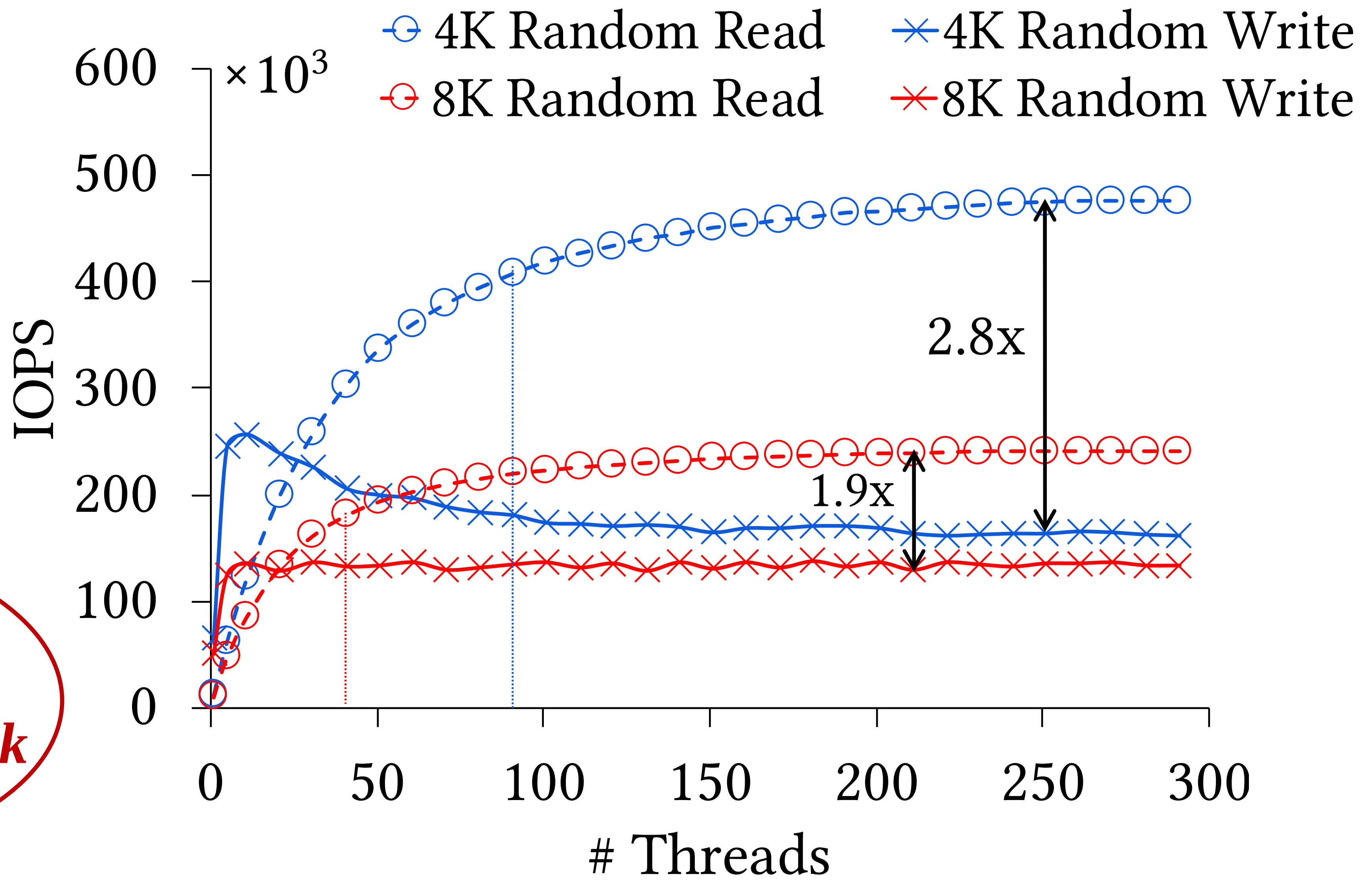
Quantifying Asymmetry & Concurrency

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

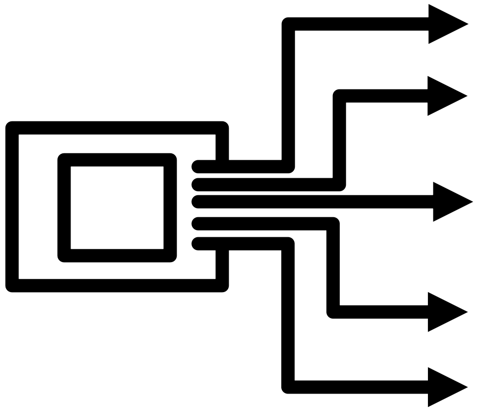
Device	α	k_r	k_w
Optane SSD	1.1	6	5
PCIe SSD	2.8	80	8
SATA SSD	1.5	25	9
Virtual SSD	2.0	11	19

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

disc.bu.edu/papers/damon21-papon

Guidelines for System Design in SSDs

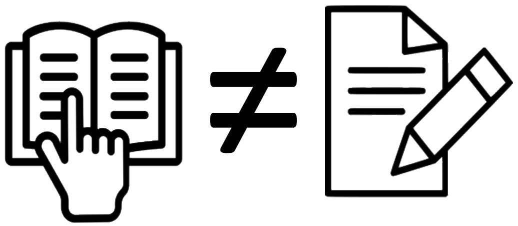
know thy device



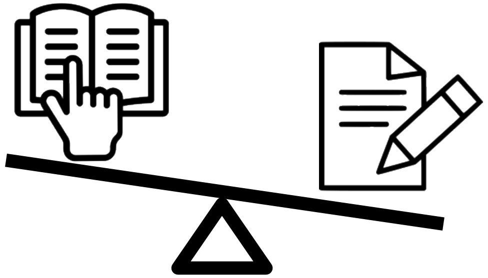
exploit k_r and k_w (with care)

read concurrency

write concurrency

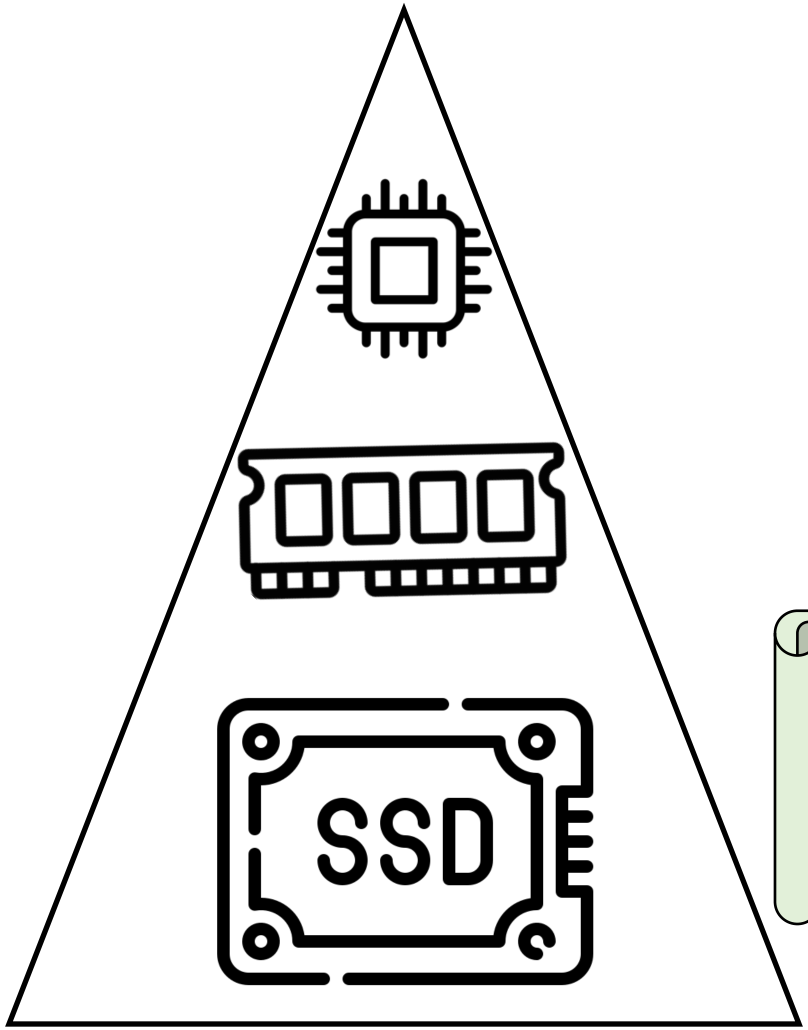


treat read and write differently



asymmetry (α) controls performance

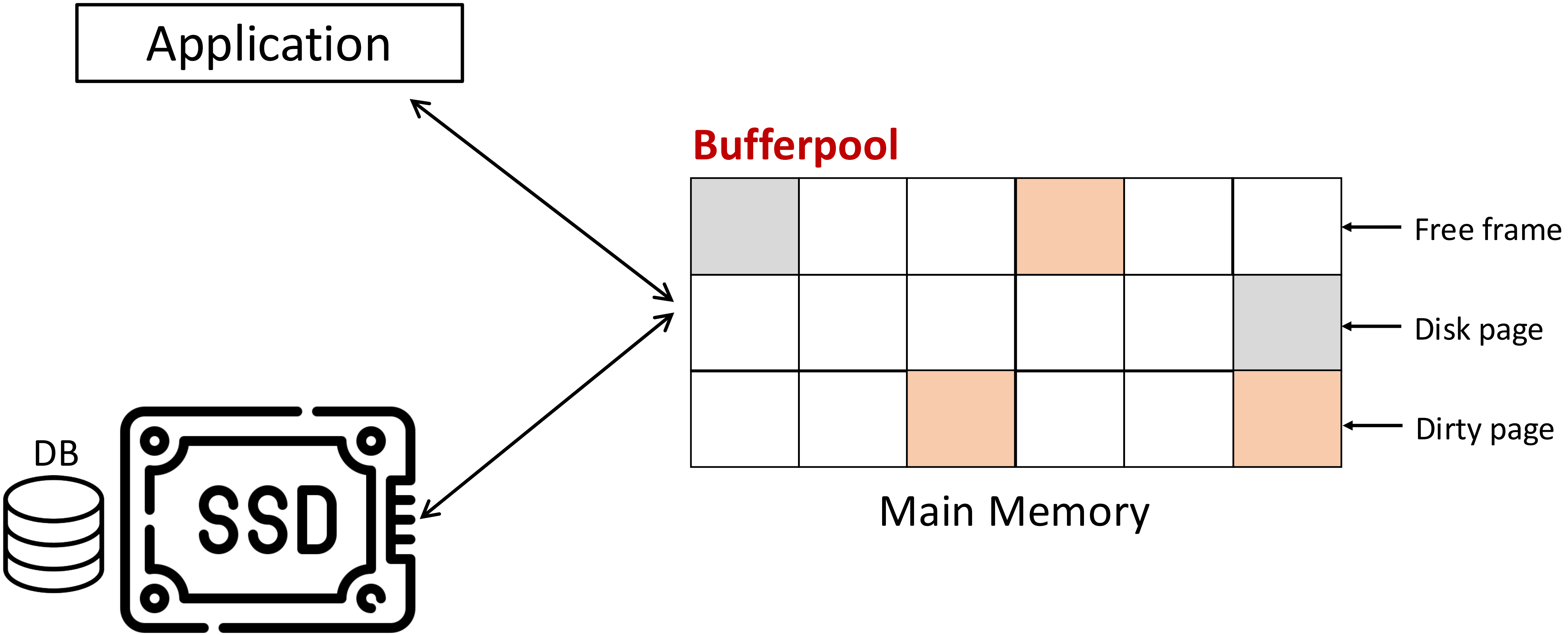
Goal: Developing Storage-Aware Data Systems



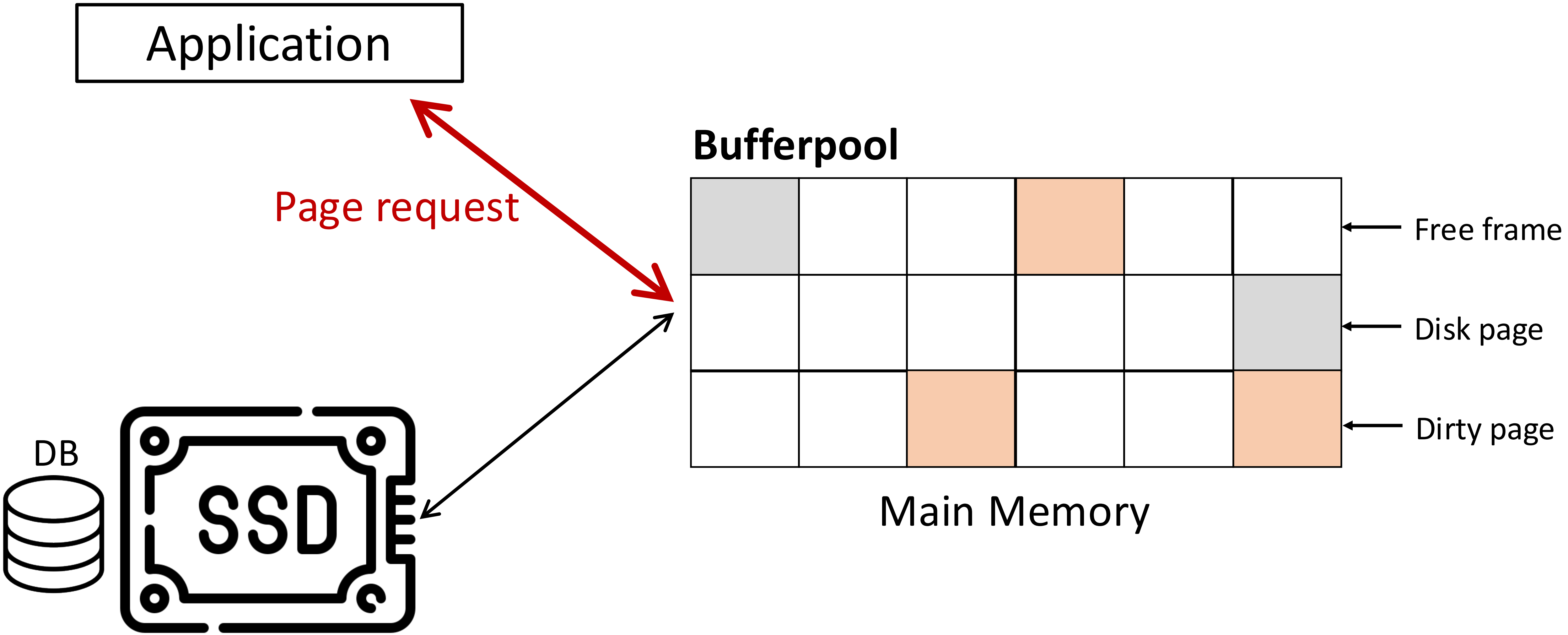
Tailor Data Systems
for **SSD Asymmetry**
& **Concurrency**

PIO [CIDR'21+DaMoN@SIGMOD'21]
ACE Bufferpool [IEEE ICDE '23]
CAVE Graph Engine [SIGMOD '24]
SSD-Aware Systems [IEEE ICDE '24]
ReStore [Under Review]

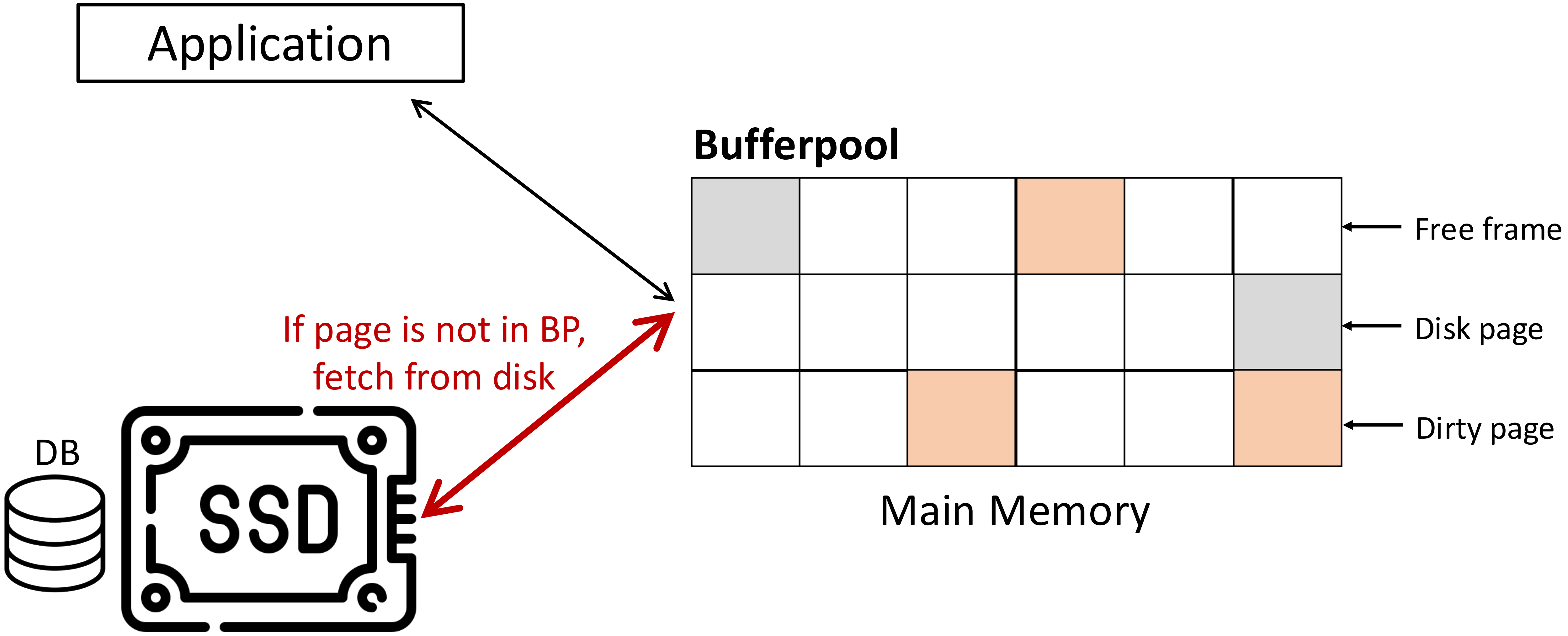
Bufferpool is Tightly Connected to Storage



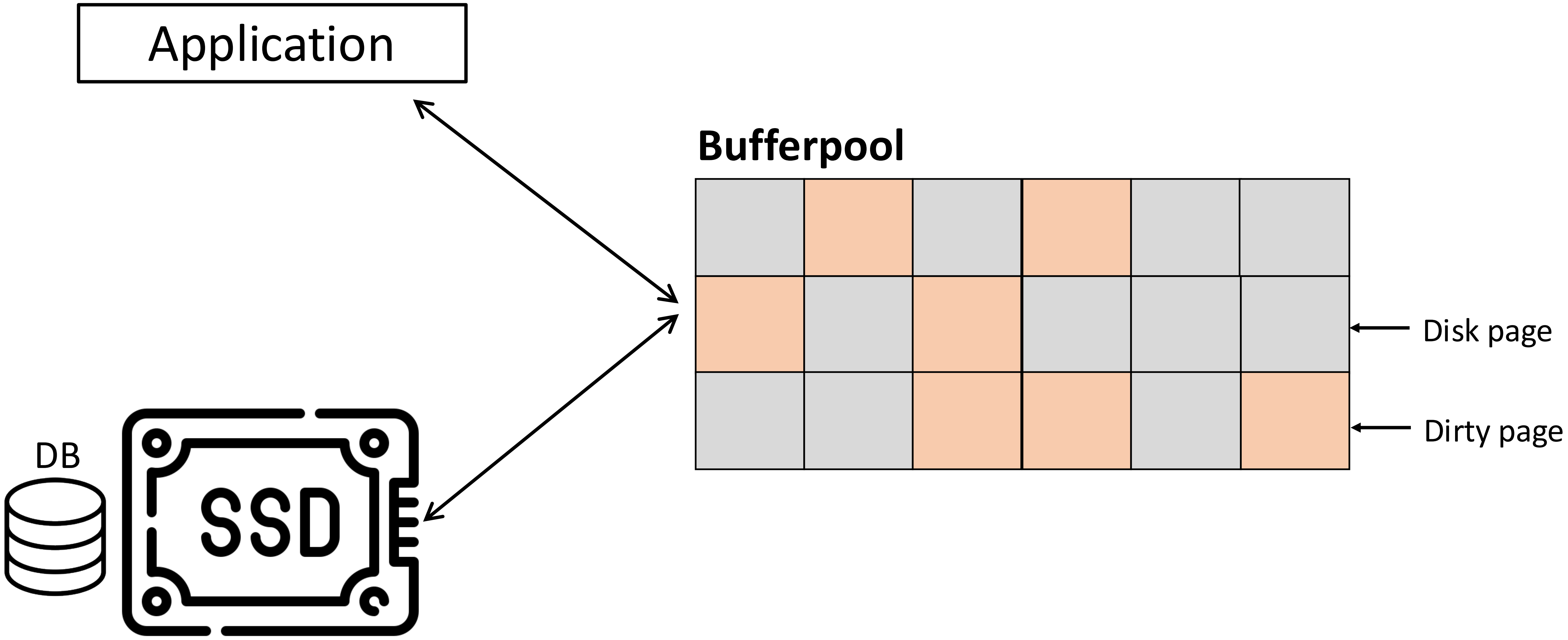
Bufferpool Manager



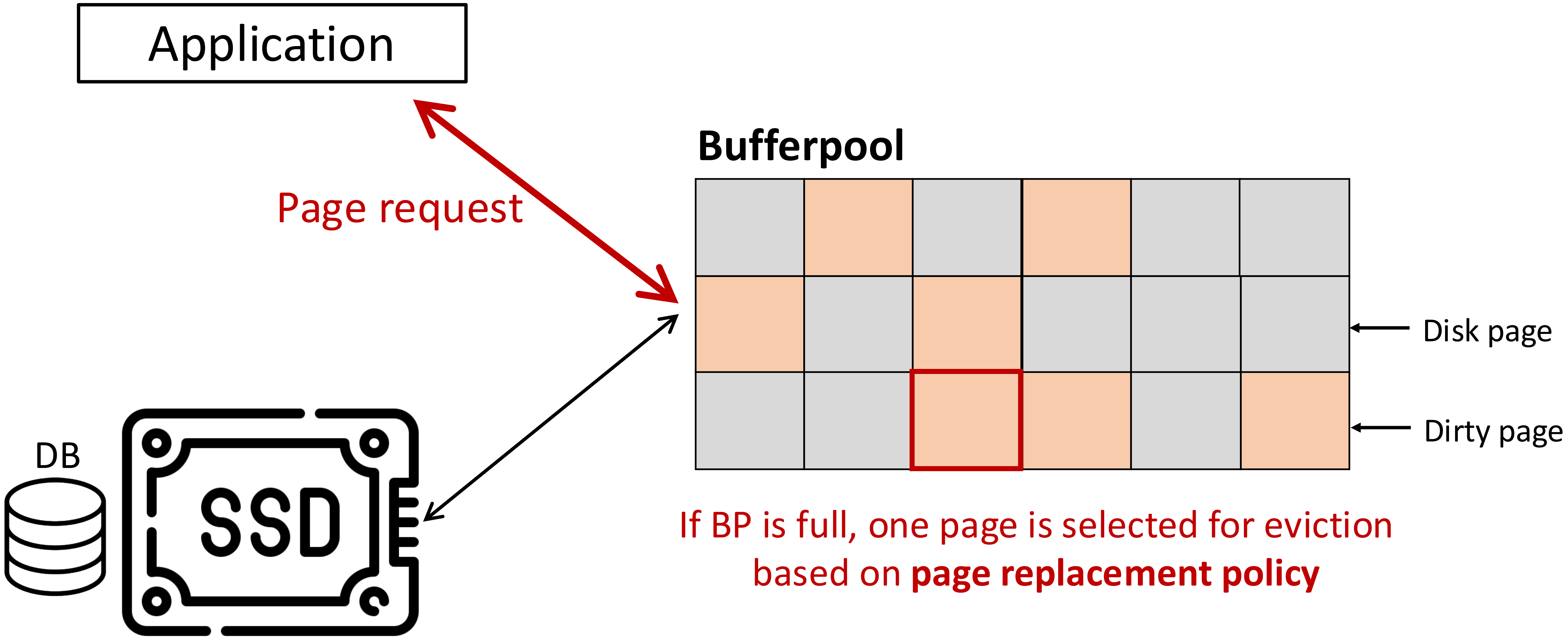
Bufferpool Manager



Bufferpool Manager

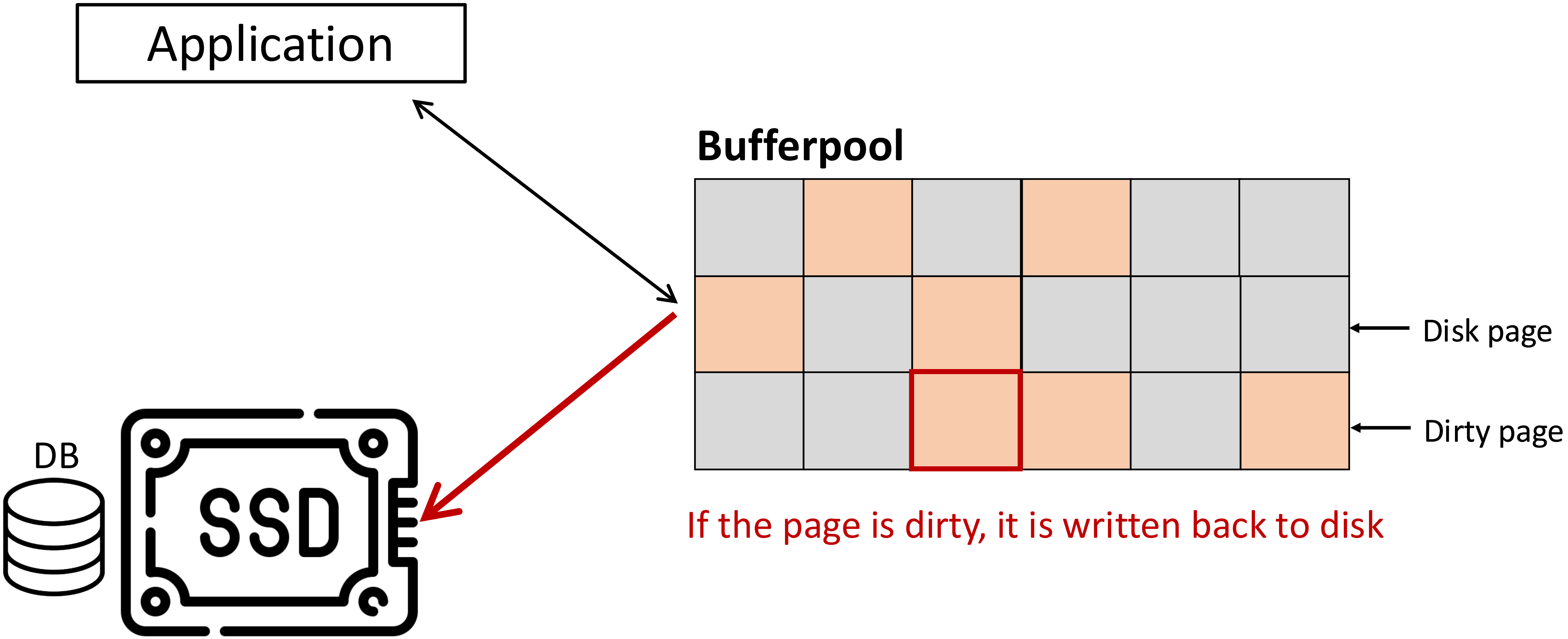


Traditional Bufferpool Manager

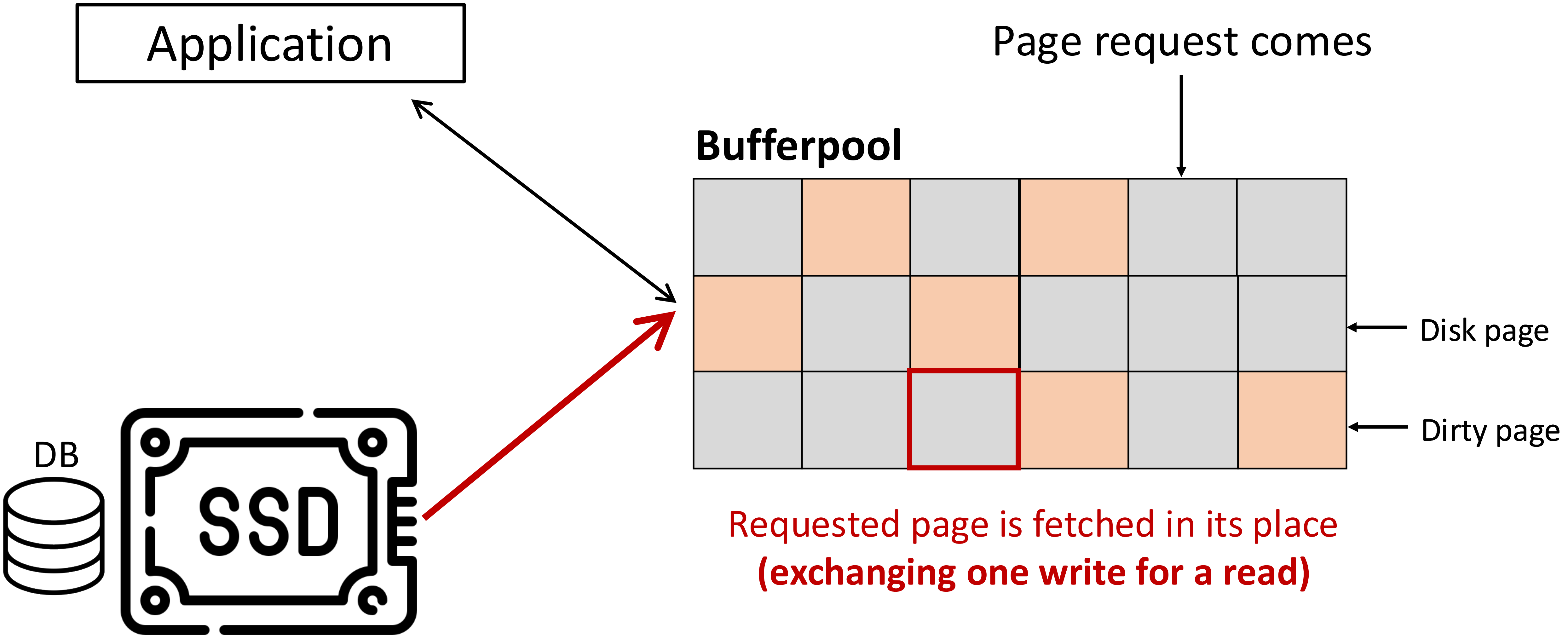


If BP is full, one page is selected for eviction based on **page replacement policy**

Traditional Bufferpool Manager



Traditional Bufferpool Manager



Requested page is fetched in its place
(exchanging one write for a read)

Popular Page Replacement Algorithms

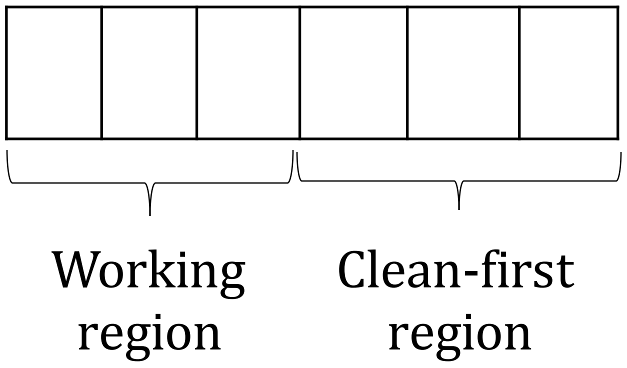
LRU (Most Popular)

LFU, FIFO (Simple)

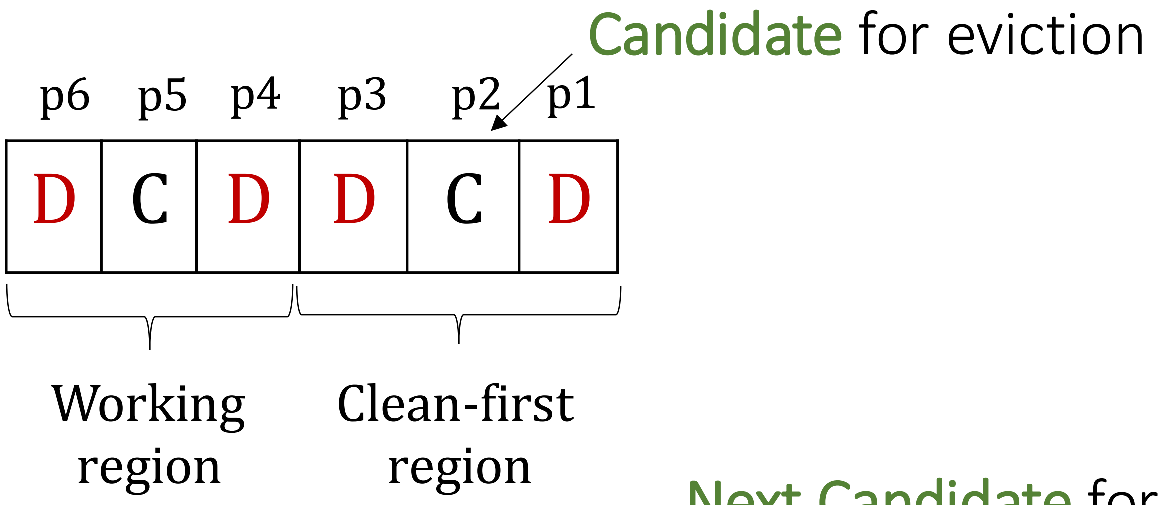
Clock Sweep (Commercial)

CFLRU
LRU-WSR
} Flash-Friendly

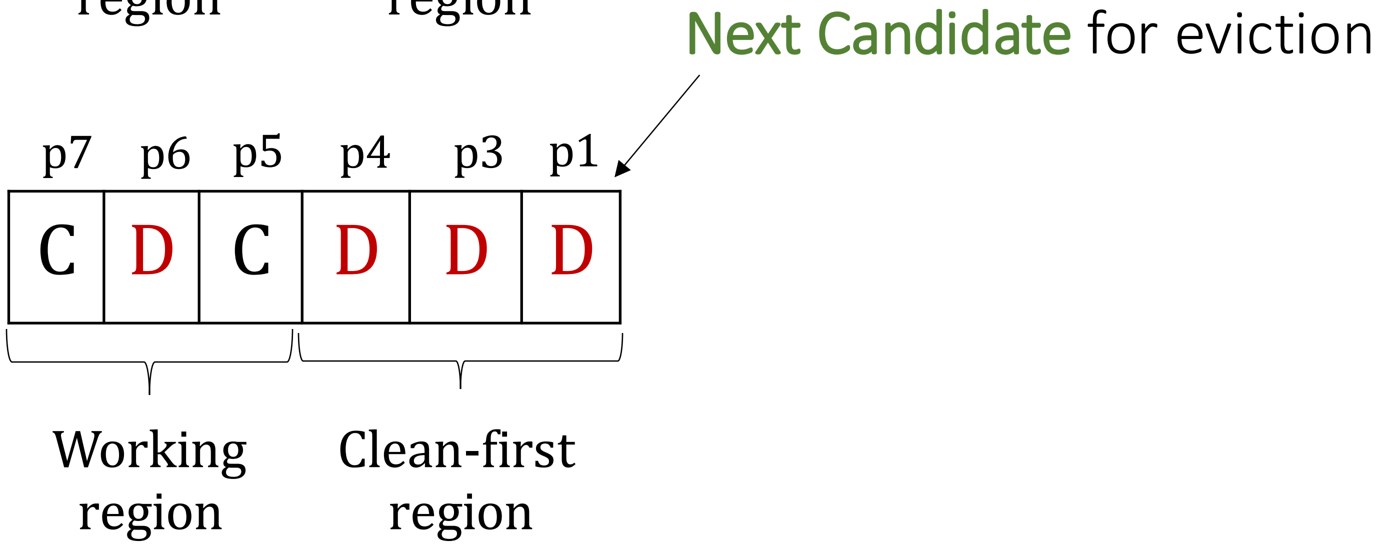
CFLRU



CFLRU



After Eviction:



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

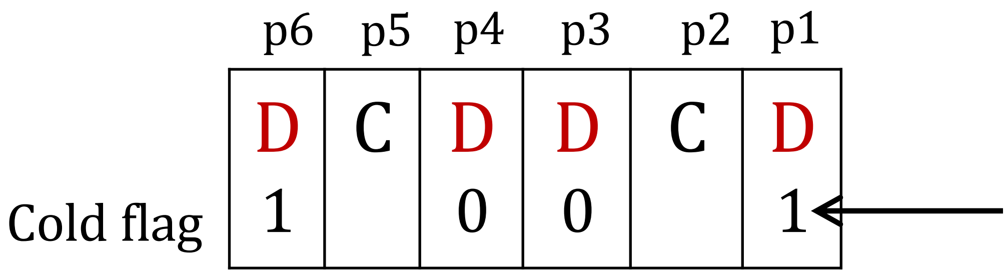
	p1	p6	p5	p4	p3	p2
	D	D	C	D	D	C
Cold flag	1	1		0	0	

Candidate for eviction

After Eviction:

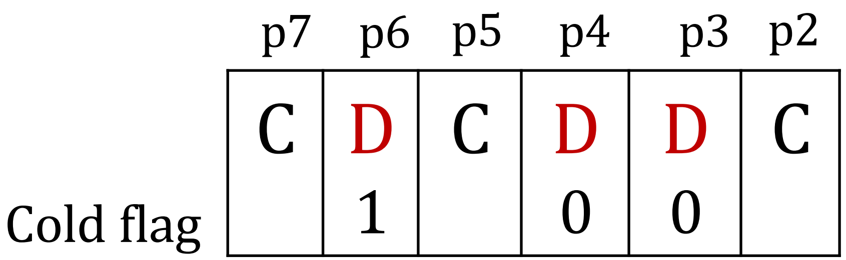
	p7	p1	p6	p5	p4	p3
	C	D	D	C	D	D
Cold flag		1	1		0	0

LRU-WSR



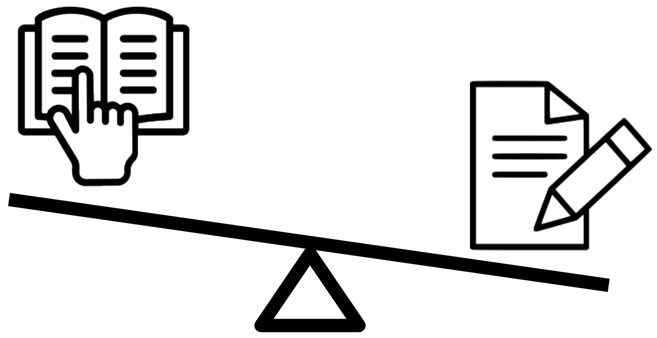
Cold flag set!
Candidate for eviction

After Eviction:

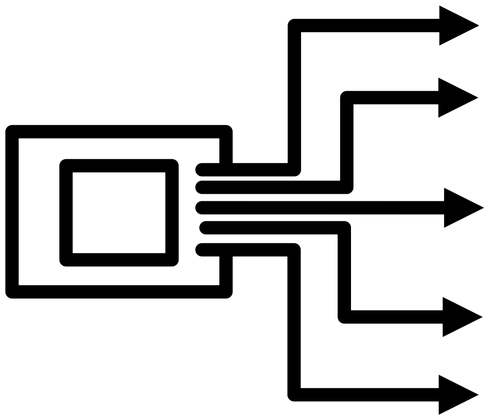


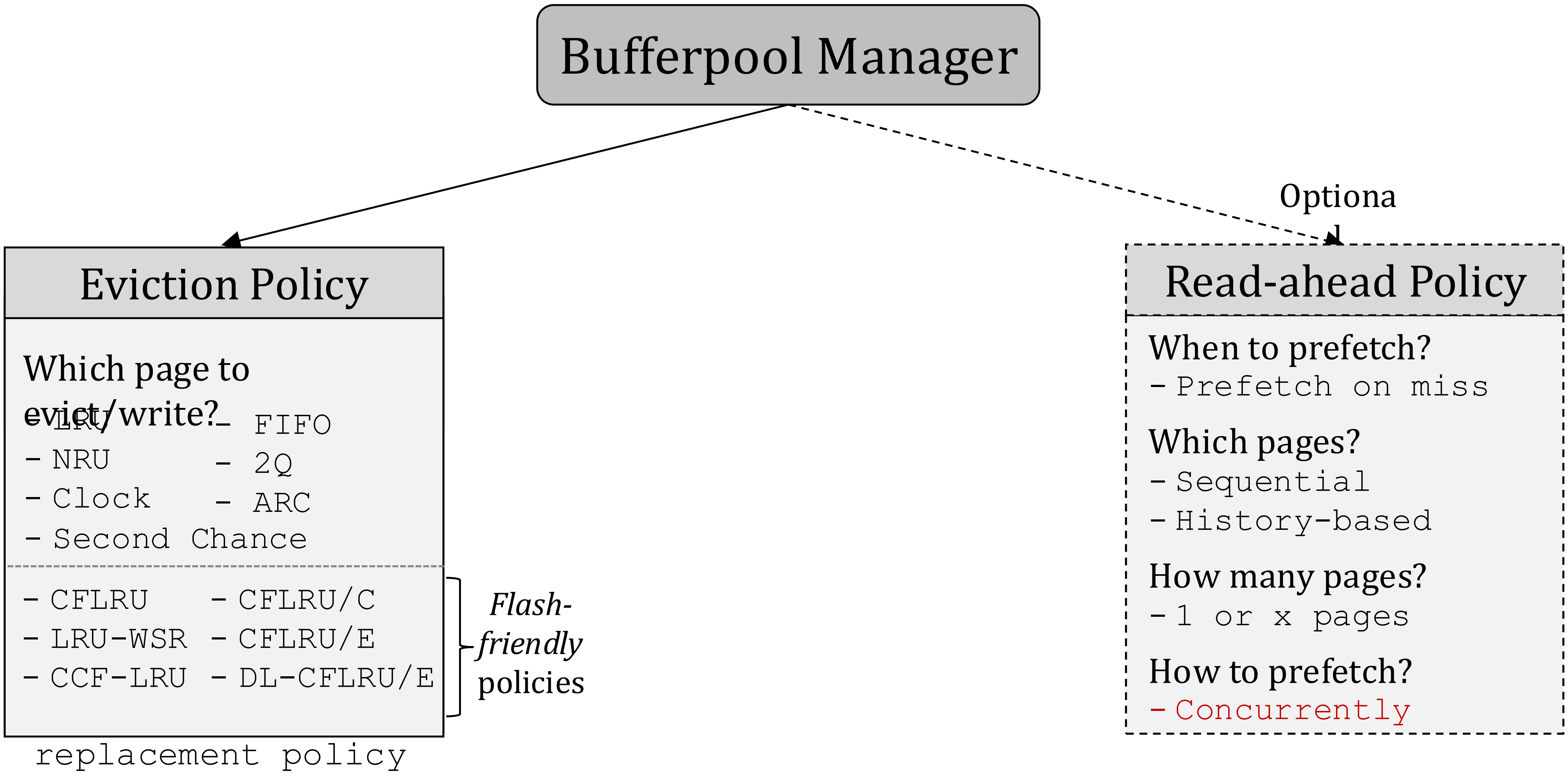
The Challenges

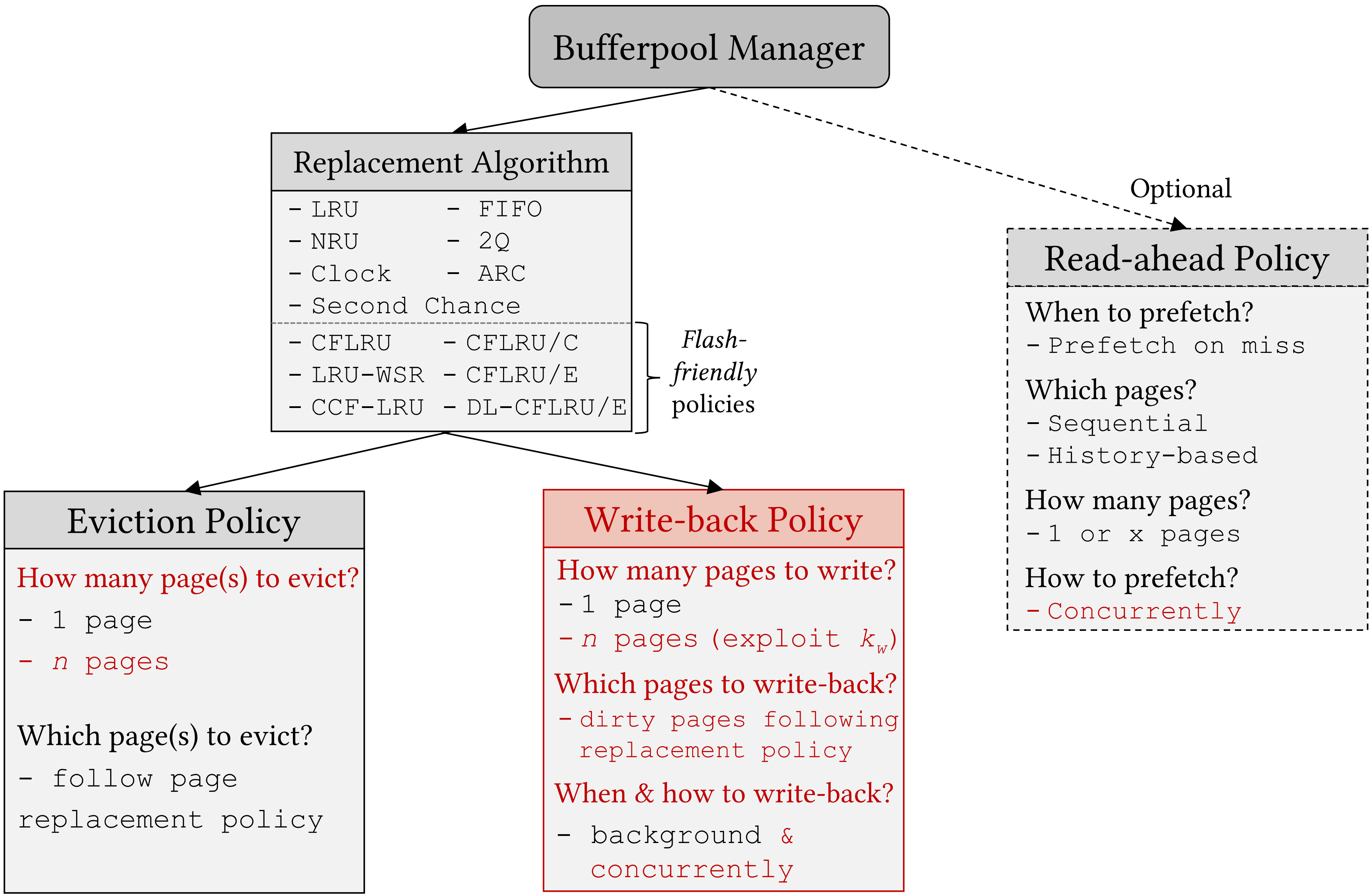
- With write asymmetry, exchanging one write for one read is **NOT ideal**.



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



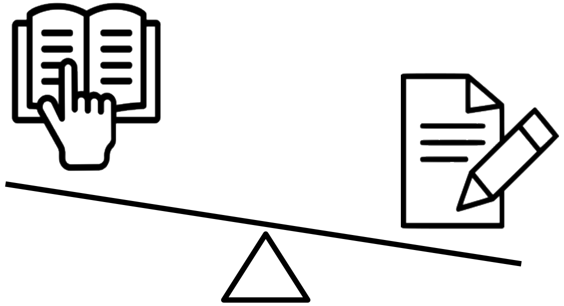

Bufferpool Manager
Optiona
l
Eviction Policy
Which page to evict/write?
- LRU - FIFO
- NRU - 2Q
- Clock - ARC
- Second Chance
- CFLRU - CFLRU/C
- LRU-WSR - CFLRU/E
- CCF-LRU - DL-CFLRU/E
Flash-friendly policies
replacement policy
Read-ahead Policy
When to prefetch?
- Prefetch on miss
Which pages?
- Sequential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

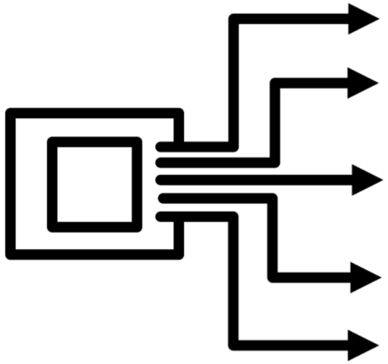

Bufferpool Manager
Replacement Algorithm
- LRU - FIFO
- NRU - 2Q
- Clock - ARC
- Second Chance
- CFLRU - CFLRU/C
- LRU-WSR - CFLRU/E
- CCF-LRU - DL-CFLRU/E
Flash-friendly policies
Optional
Read-ahead Policy
When to prefetch?
- Prefetch on miss
Which pages?
- Sequential
- History-based
How many pages?
- 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

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

ACE Bufferpool Manager

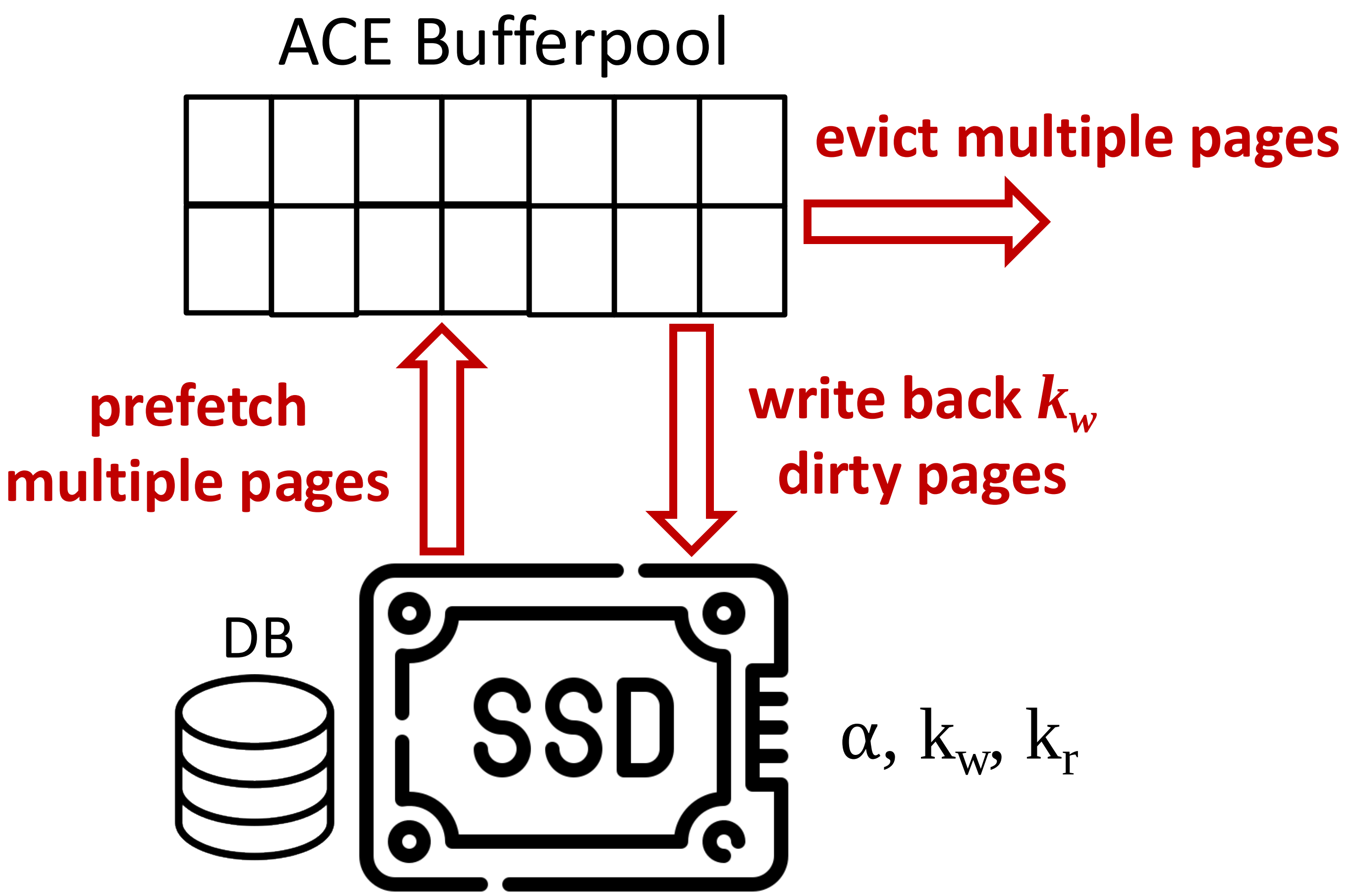
Use device's properties





IEEE ICDE 2023

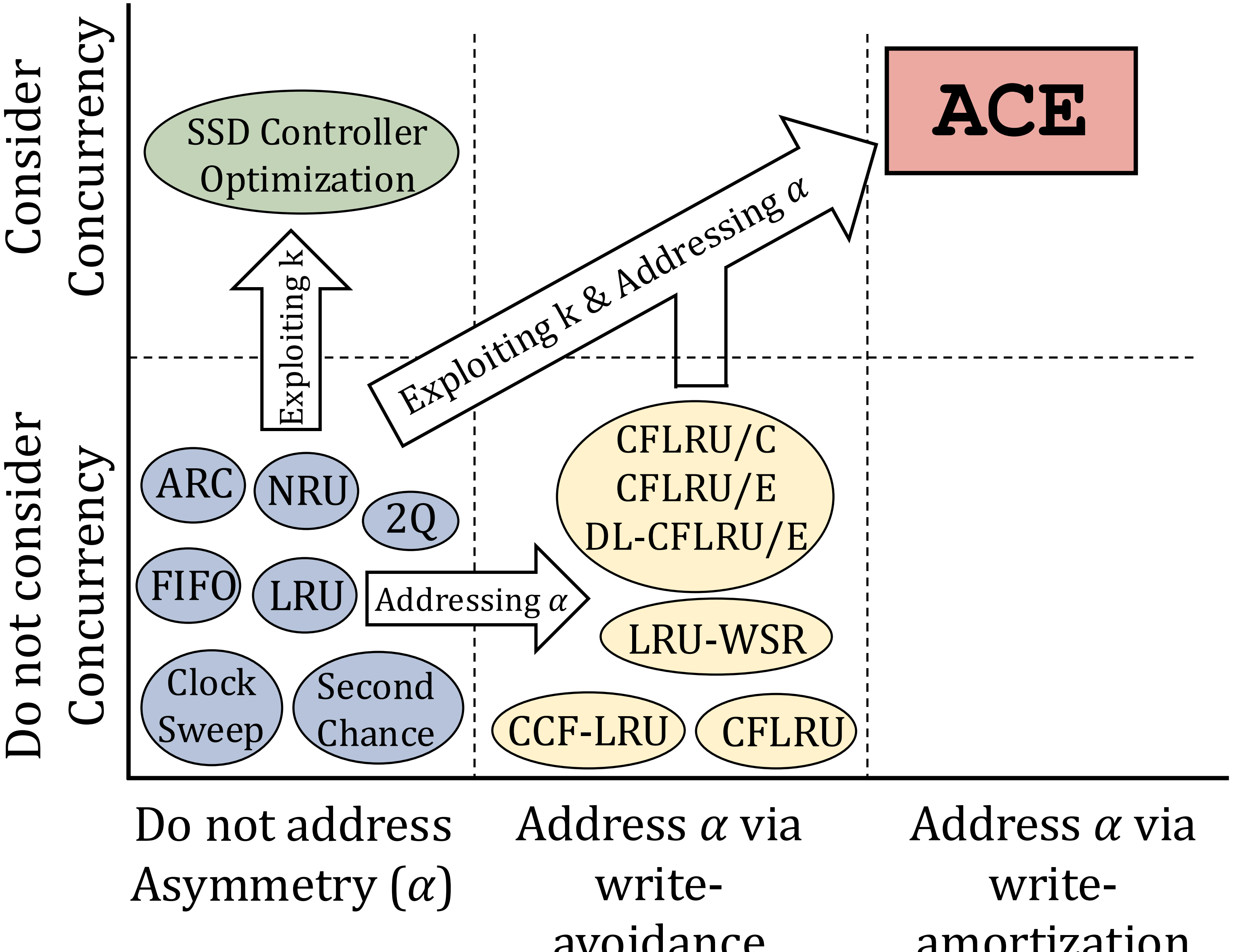
ACE Bufferpool Manager



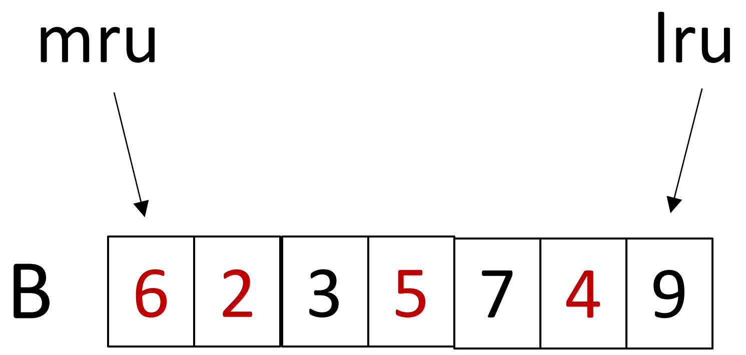
- ✓ Can be integrated with **any** replacement algorithm
- ✓ **Any** prefetching technique can be used

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

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 → 4

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:

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 → 4

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)

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	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

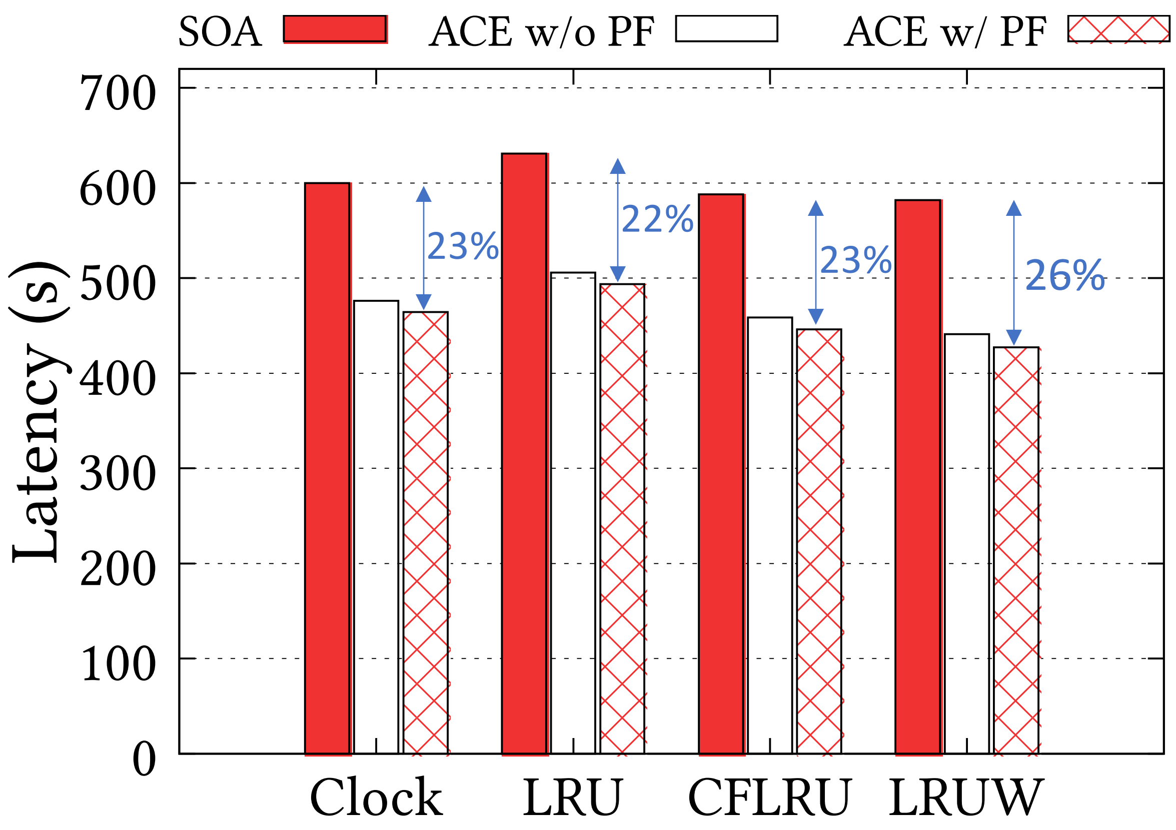
TPC-C benchmark

IEEE ICDE 2023

ACE Improves Runtime

Device: PCIe SSD

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



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

ACE improves runtime by 22-26%

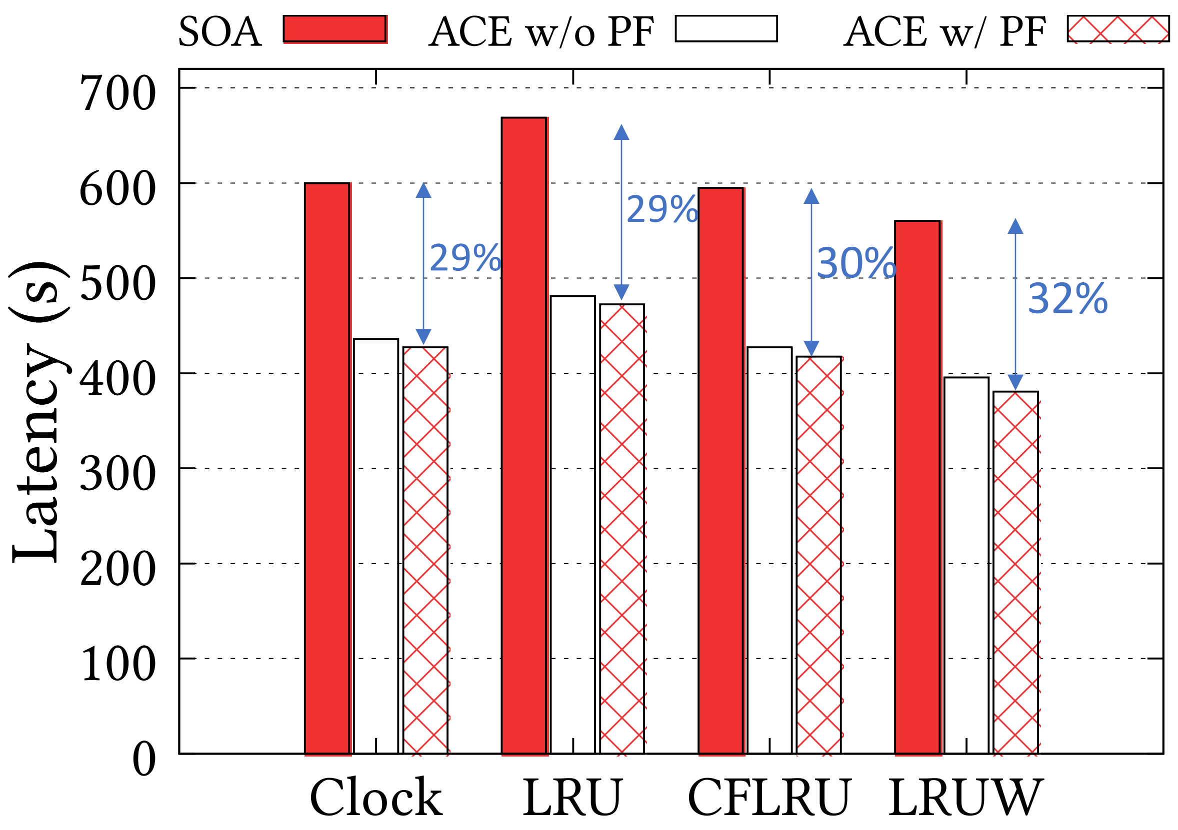
Negligible increase in buffer miss (<0.009%)

Benefit comes at no cost

Higher Gain for Write-Heavy Workload

Device: PCIe SSD

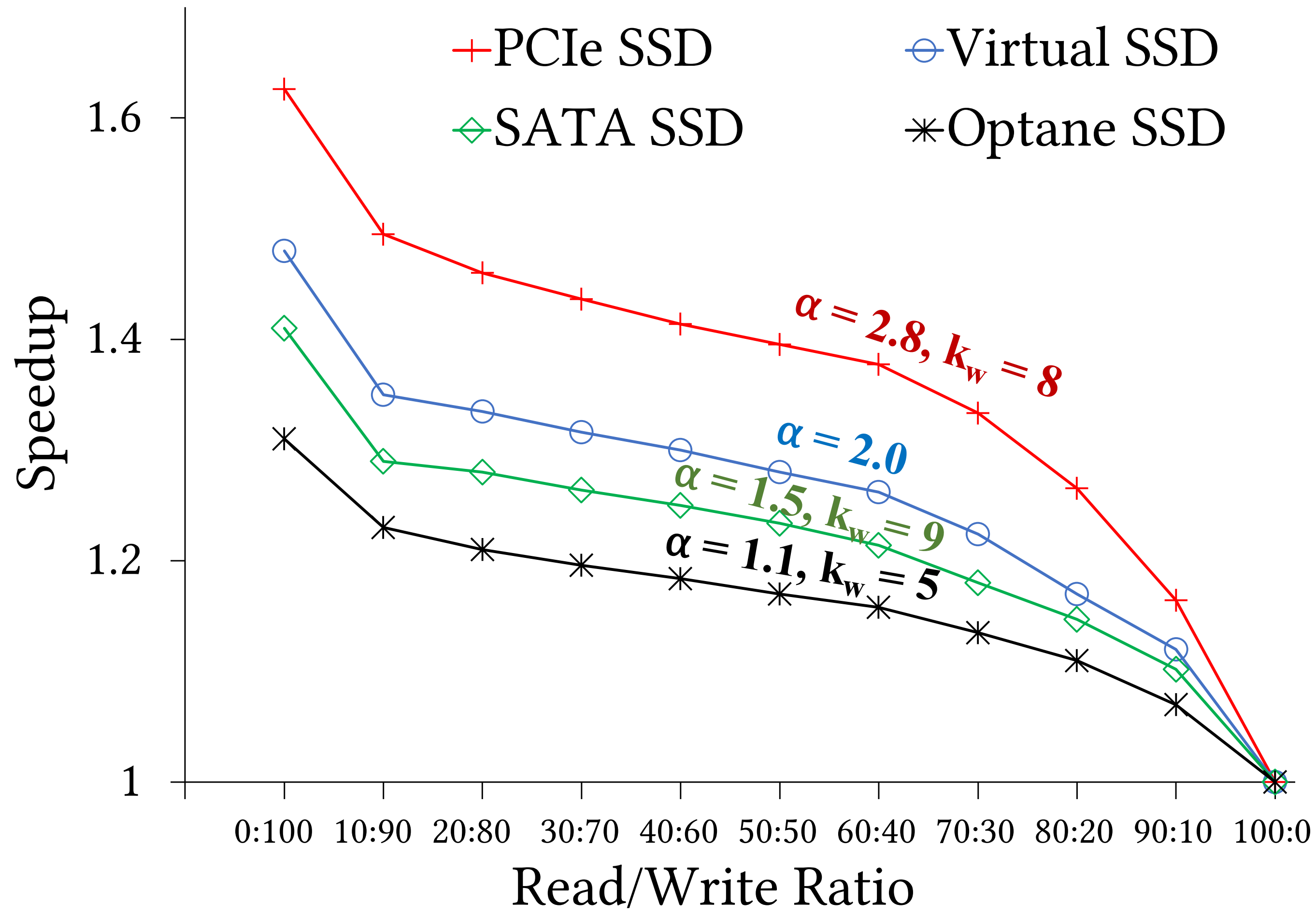
$\alpha = 2.8$, $k_w = 8$



Write-Heavy Skewed Trace
(r/w: 10/90, locality

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

Impact of R/W Ratio & Asymmetry

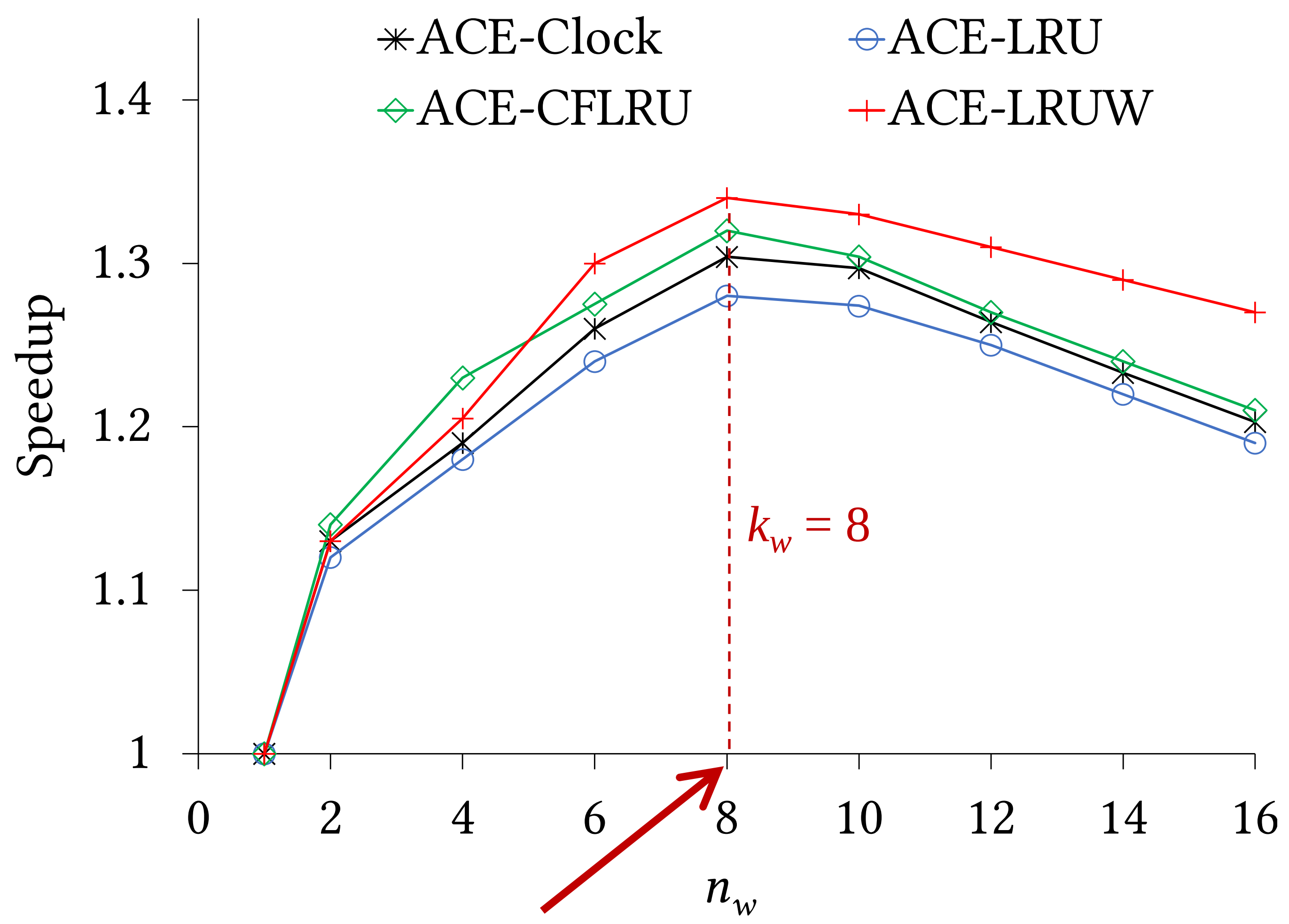


more writes, more speedup

higher asymmetry, higher speedup

good benefit even for low asymmetry

Impact of #Concurrent I/Os



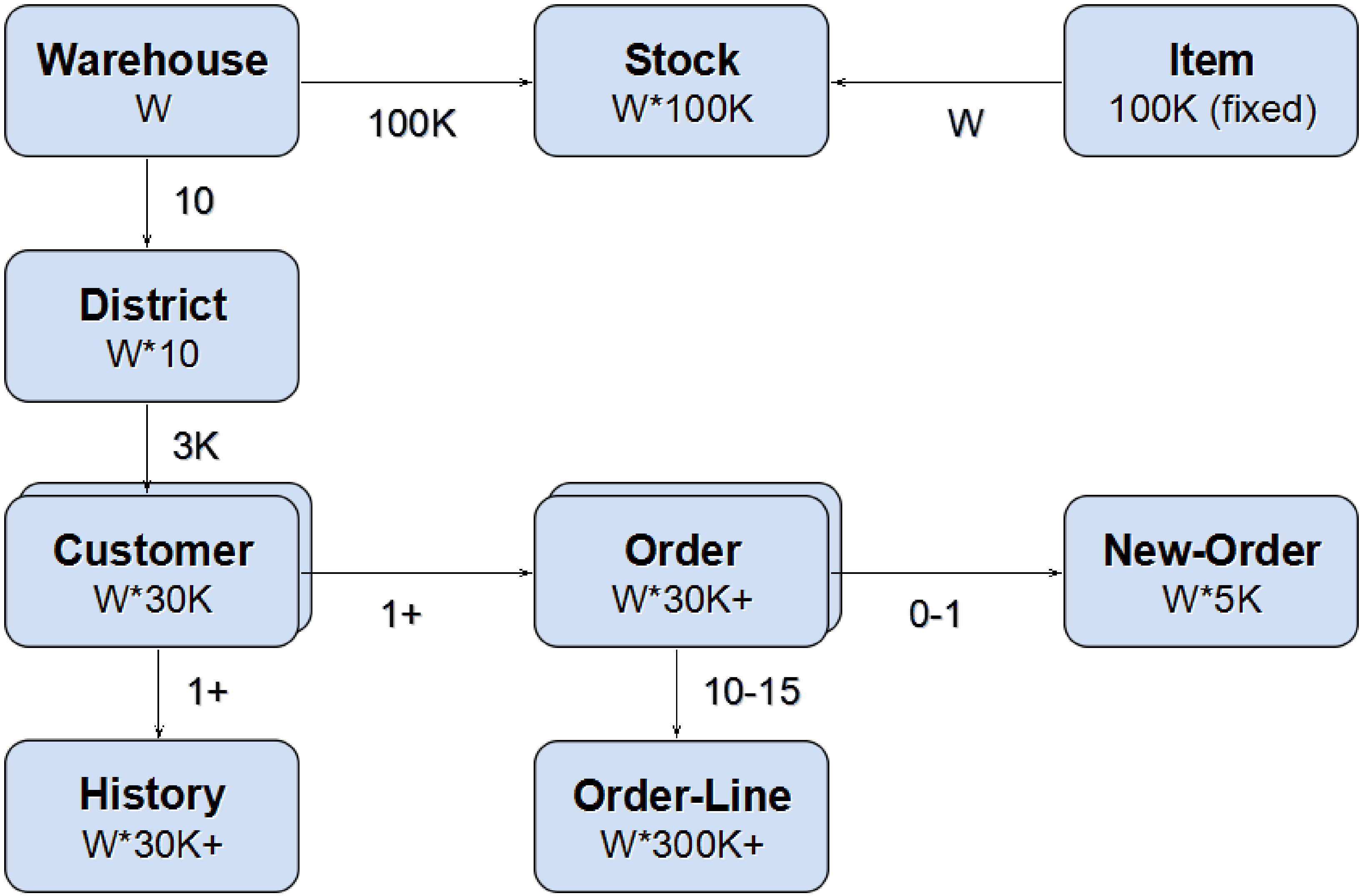
Mixed Skewed Trace
(r/w: 50/50, locality

Device: PCIe SSD

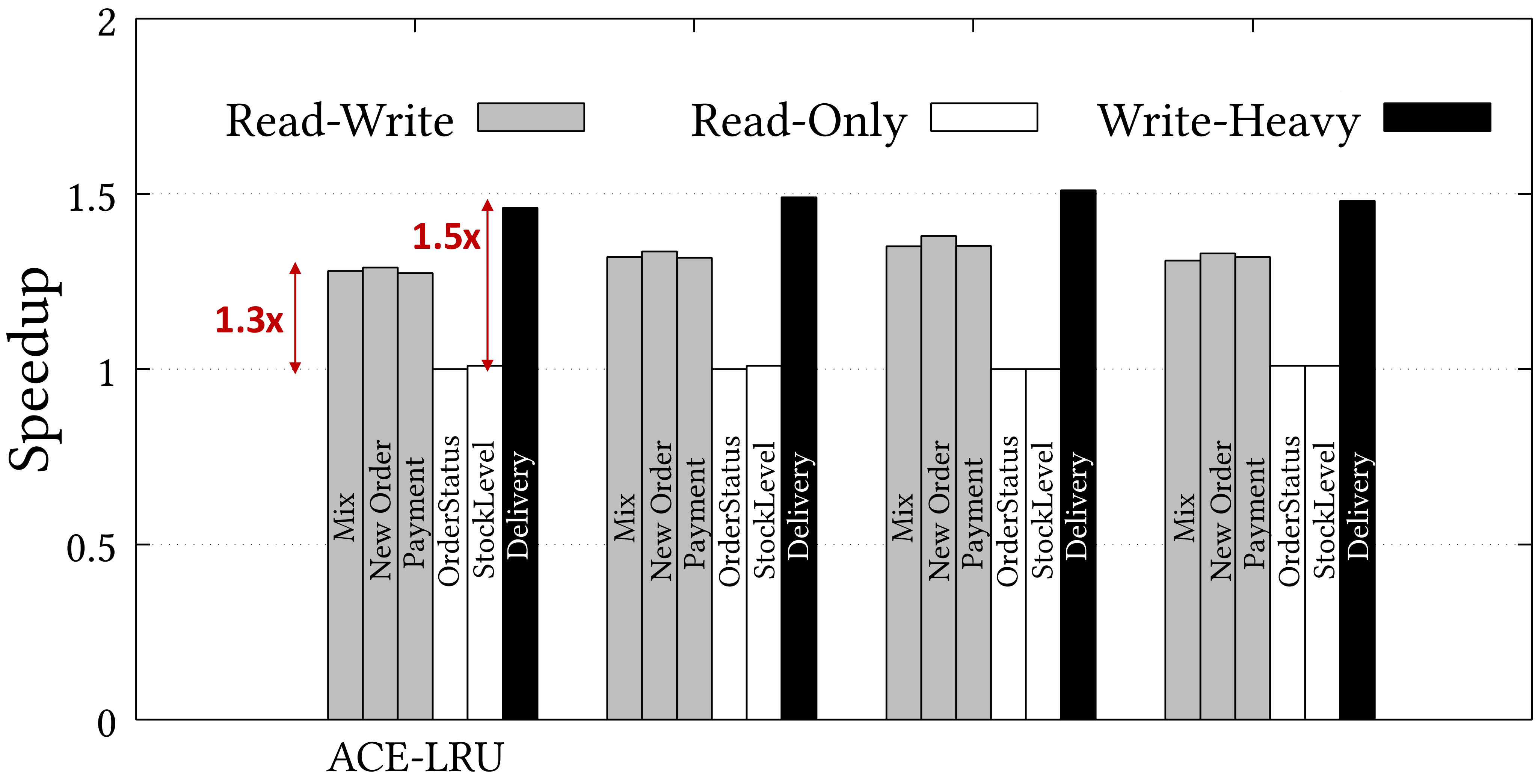
$\alpha = 2.8$, $k_w = 8$

Highest speedup when optimal concurrency is used

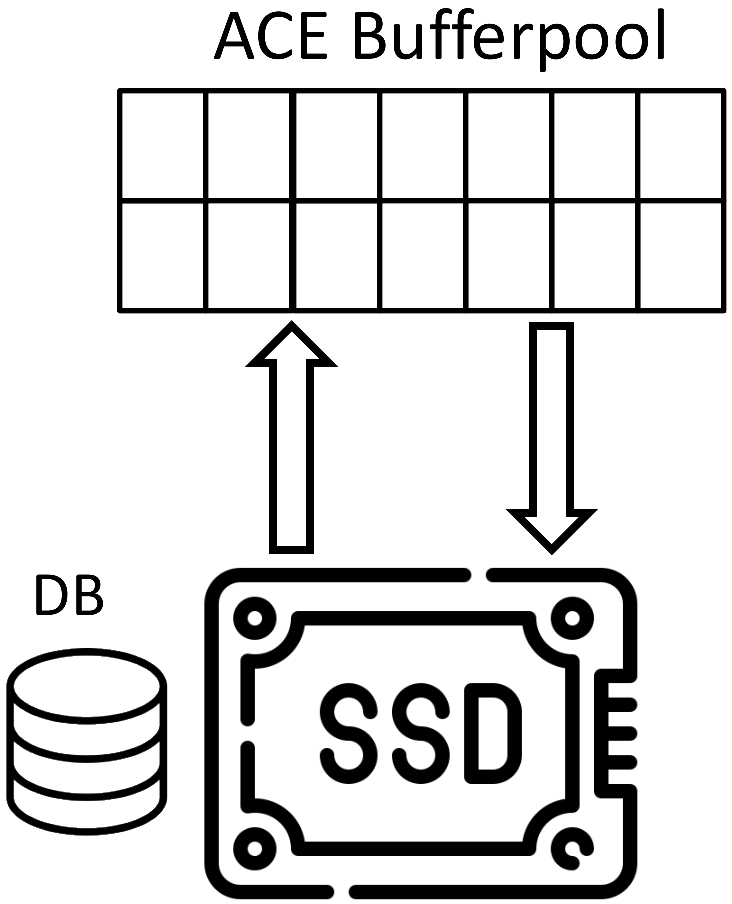
Experimental Evaluation (TPC-C)



Experimental Evaluation (TPC-C)

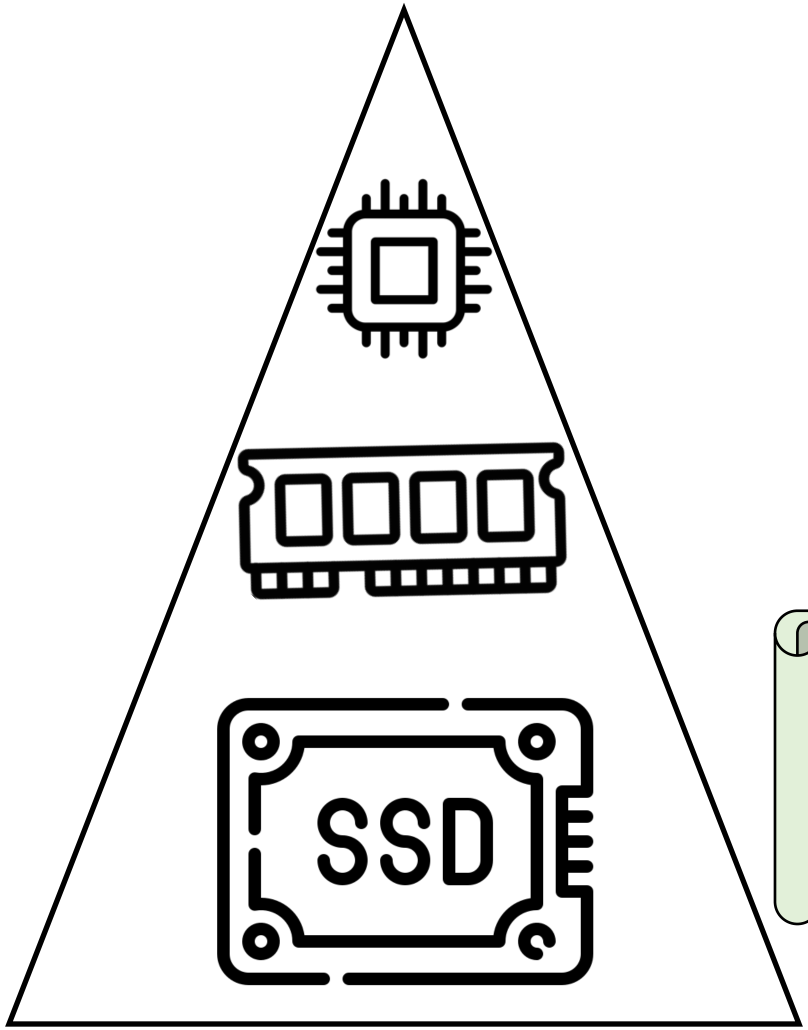


ACE Achieves 1.3x for mixed TPC-C



- ☑ **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 Storage-Aware Data Systems

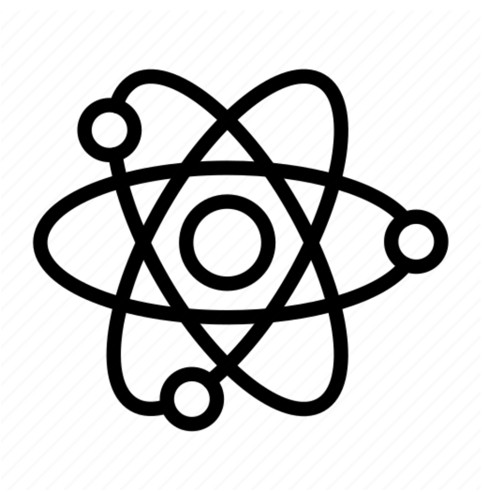


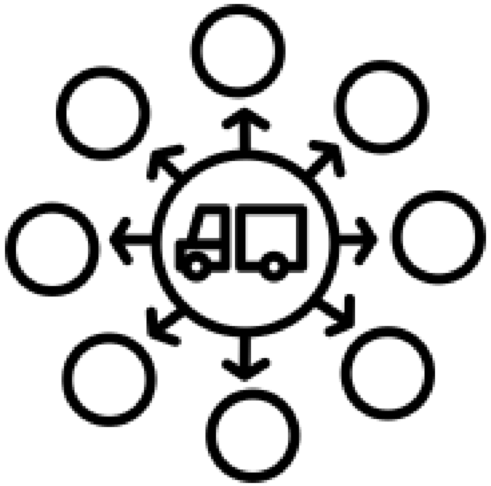
Tailor Data Systems
for **SSD Asymmetry**
& **Concurrency**

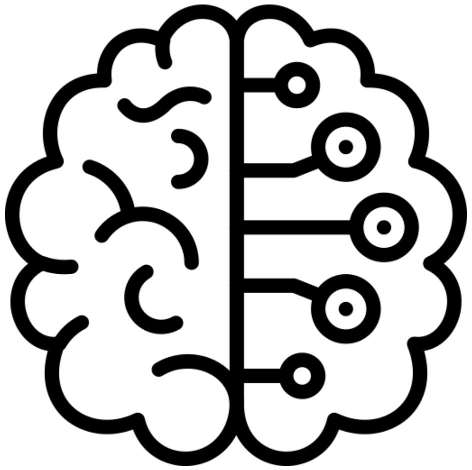
PIO [CIDR'21+DaMoN@SIGMOD'21]
ACE Bufferpool [IEEE ICDE '23]
CAVE Graph Engine [SIGMOD '24]
SSD-Aware Systems [IEEE ICDE '24]
ReStore [Under Review]

Rise of Large Graphs

Graphs are everywhere!







Social Network

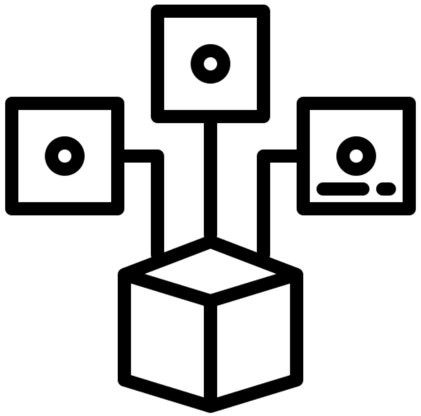
Physical Science

Transportation Network

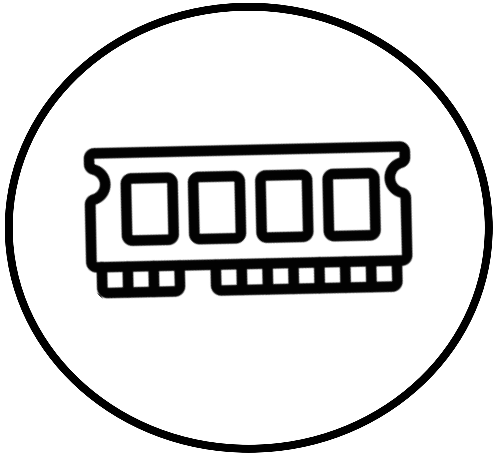
Machine Learning

Real-world graphs often have more than a billion nodes

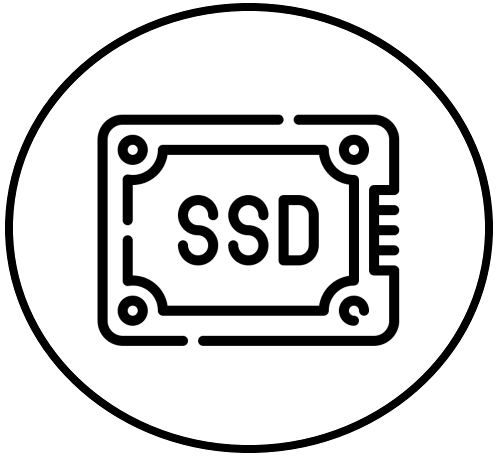
Processing Large Graphs



Distributed Systems



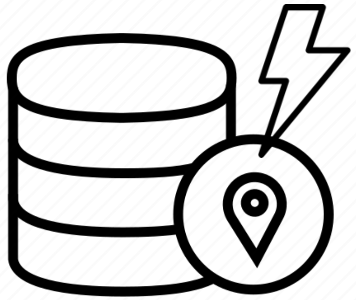
Single-node
in-memory systems



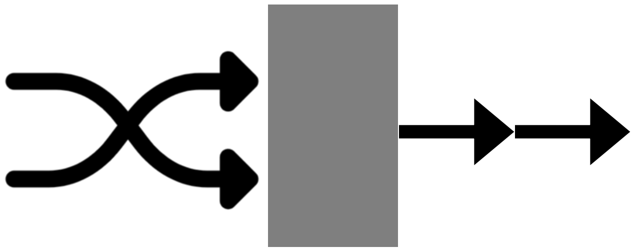
Single-node
out-of-core systems

Out of Core Systems

Data partitioning



Improve memory
& disk locality



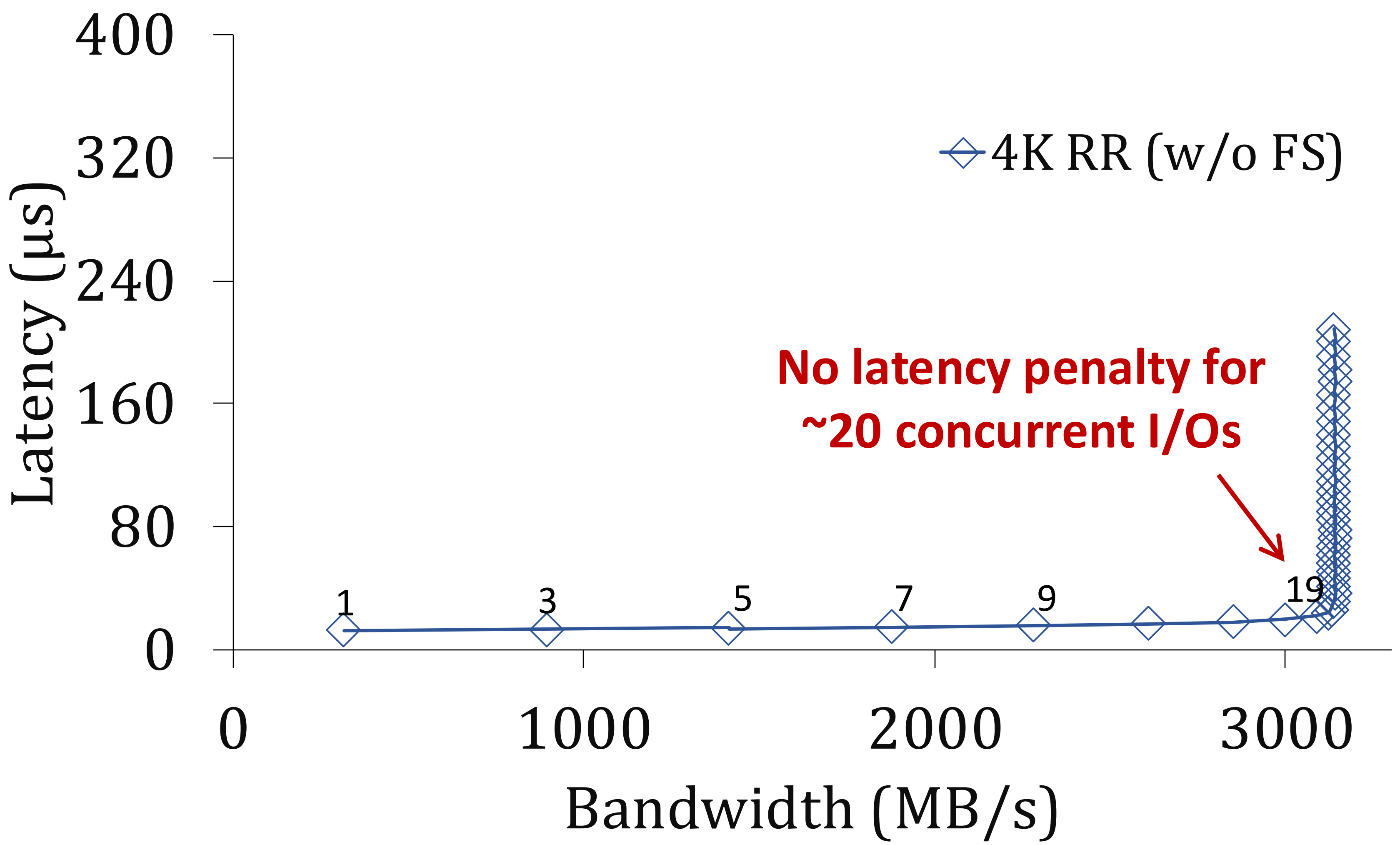
Reduce random I/O

Designed for HDDs

Impact of Concurrency

Device
PCIe SSD - P4510 (1TB)

***Optimal* read concurrency (k_r) = 20**

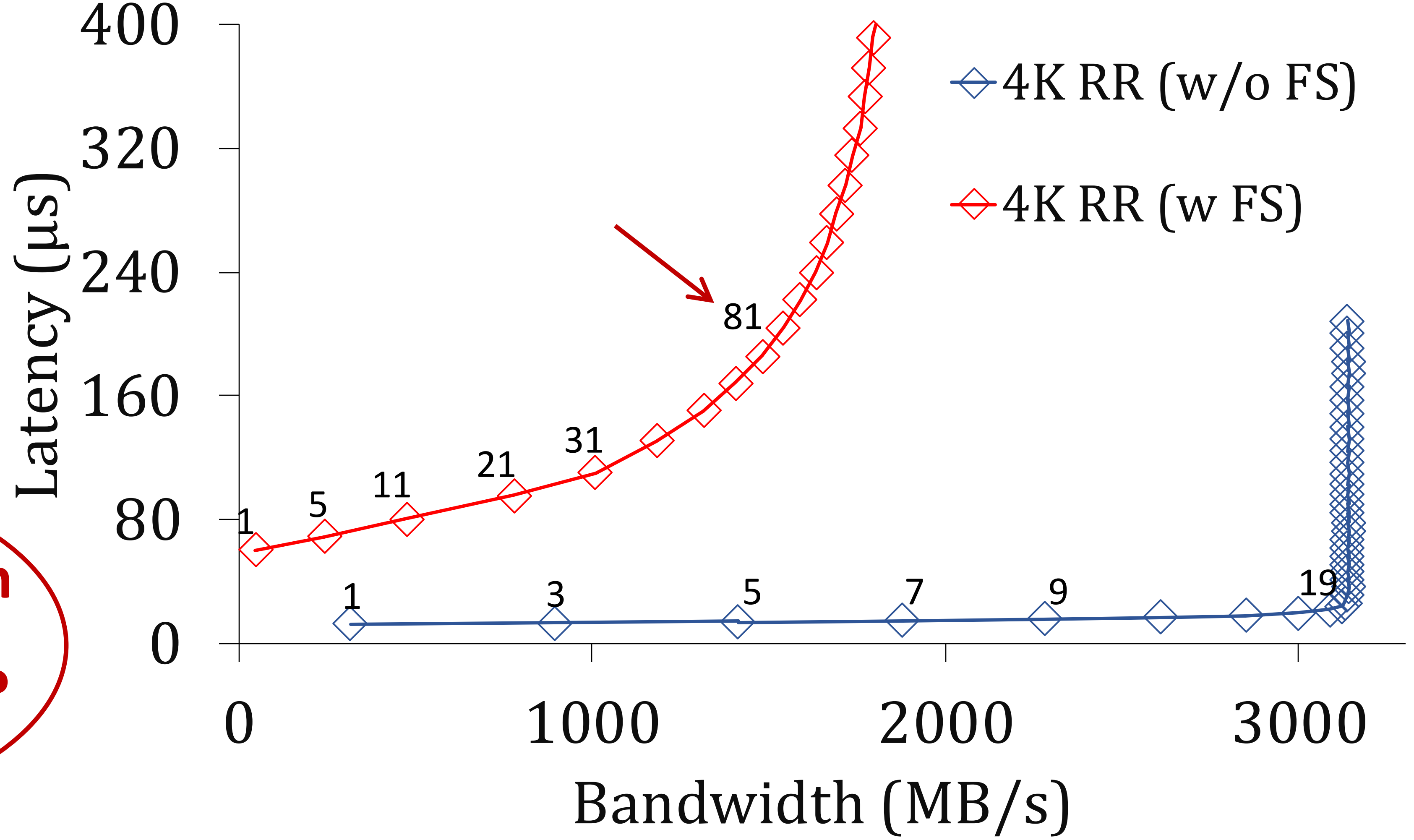


Impact of Concurrency

Device

PCIe SSD - P4510 (1TB)

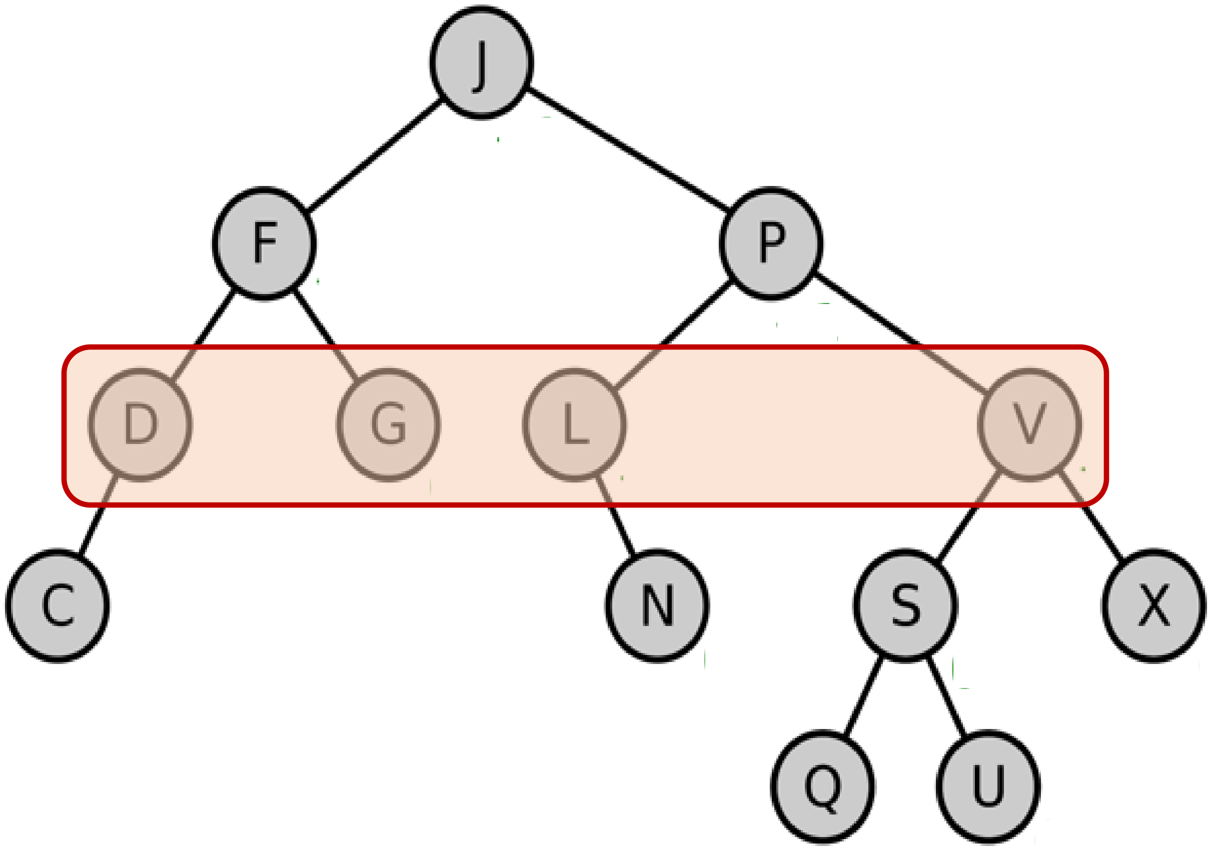
***Optimal* read concurrency (k_r) = 80**



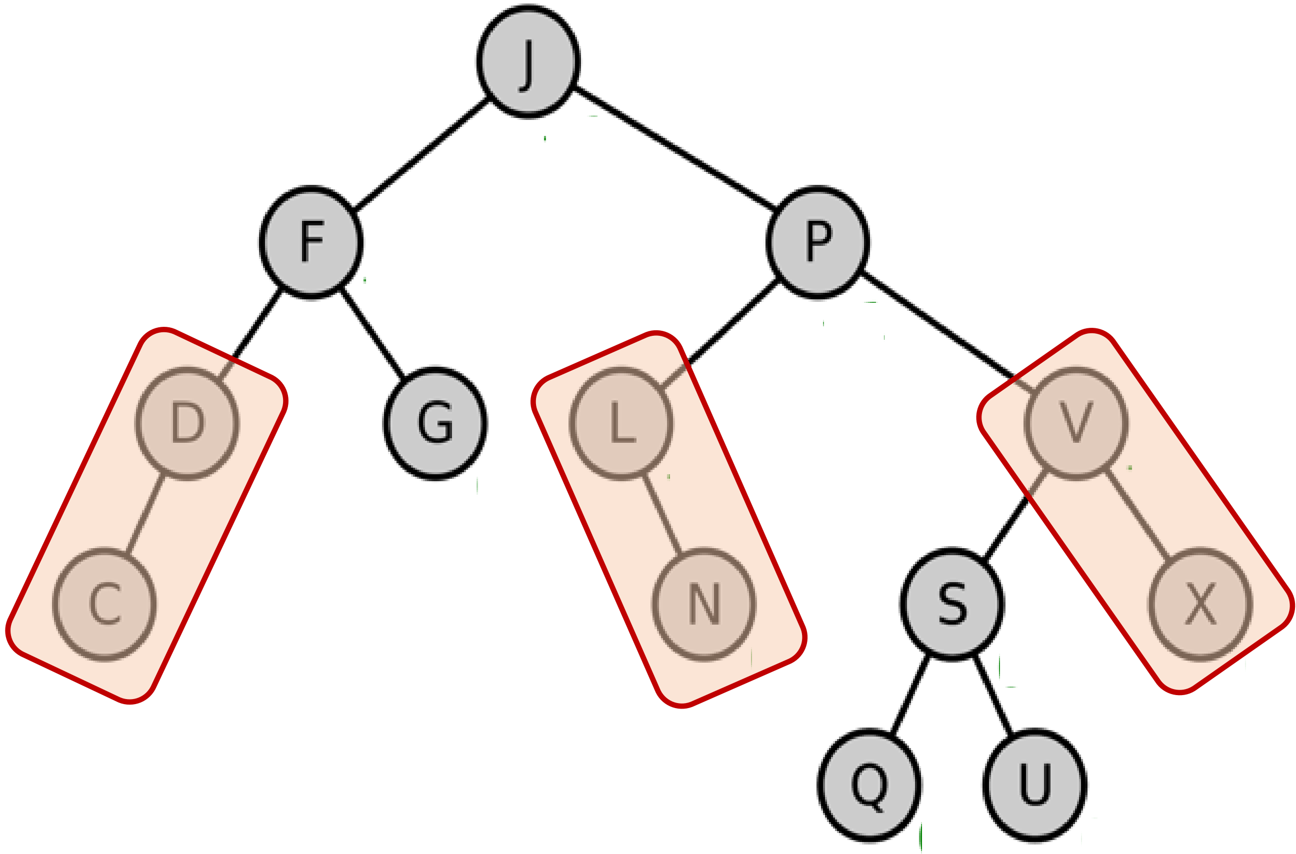
Can SSD-based Graph Systems Exploit This?

Parallelizing Graph Traversal

Intra-Subgraph Parallelization

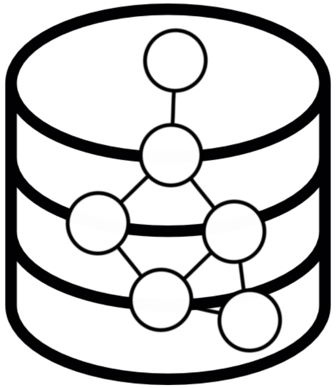


Inter-Subgraph Parallelization

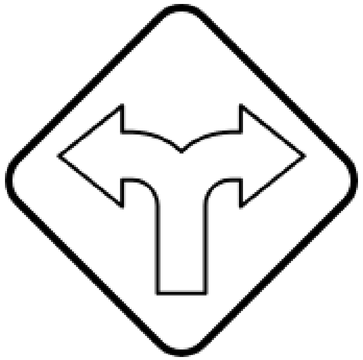


process in parallel up to k_r nodes/subgraphs

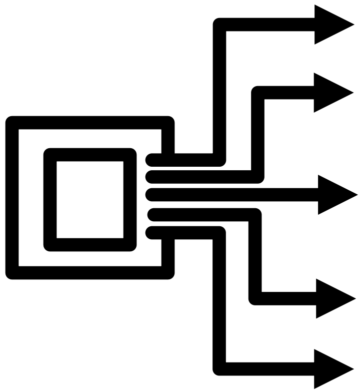
Our Goal



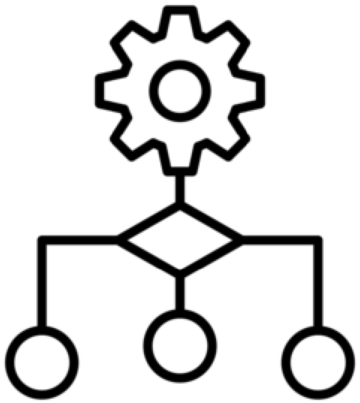
Optimize for **storage-based** graph workloads



Focus on **traversal** operations



Utilize **SSD Concurrency**

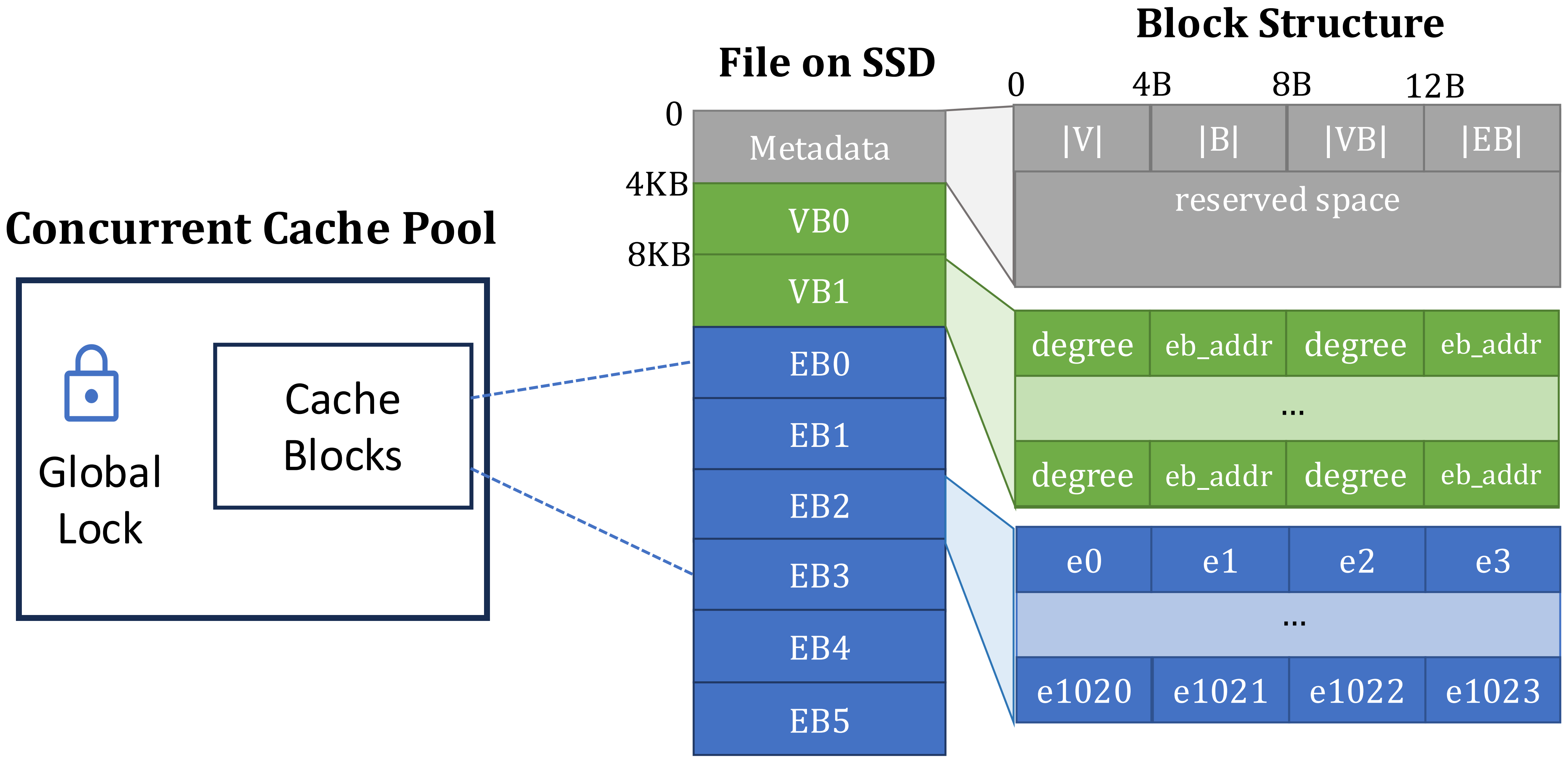


Maintain core **algorithm properties**

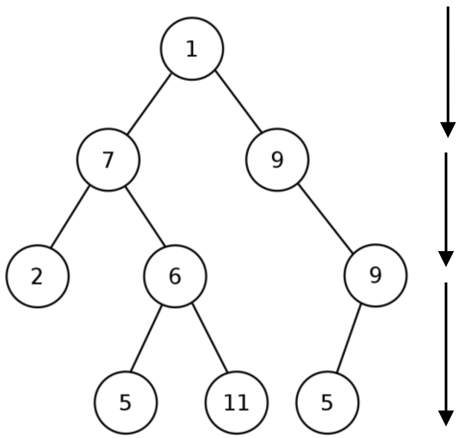
Concurrency-Aware Graph (V, E) Manager

CAVE

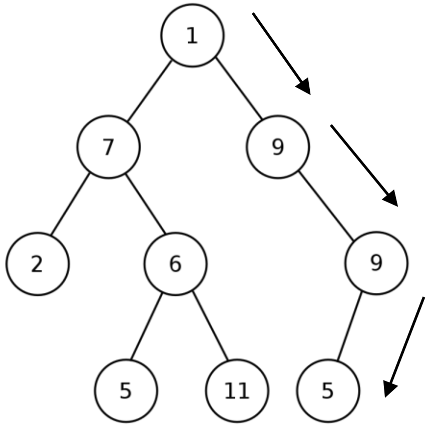
CAVE Architecture



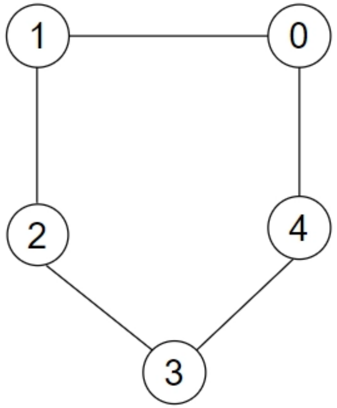
Concurrent Graph Algorithms



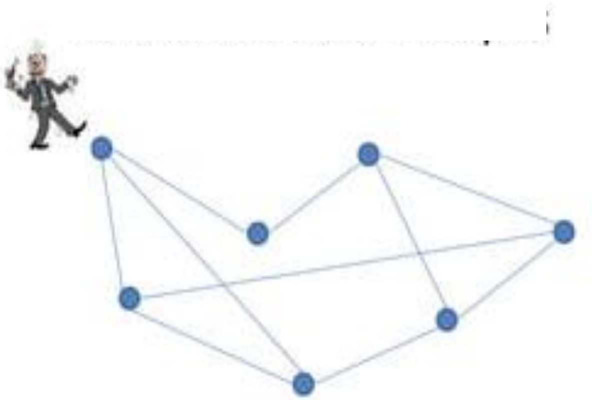
Parallel BFS



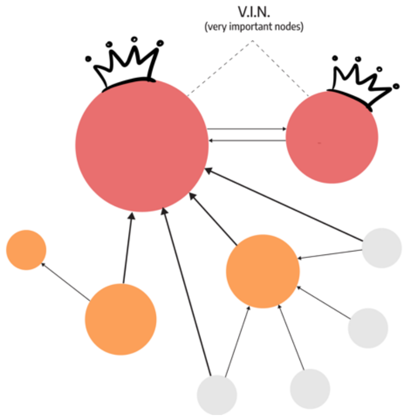
Parallel DFS



Parallel WCC



Parallel
Random Walk

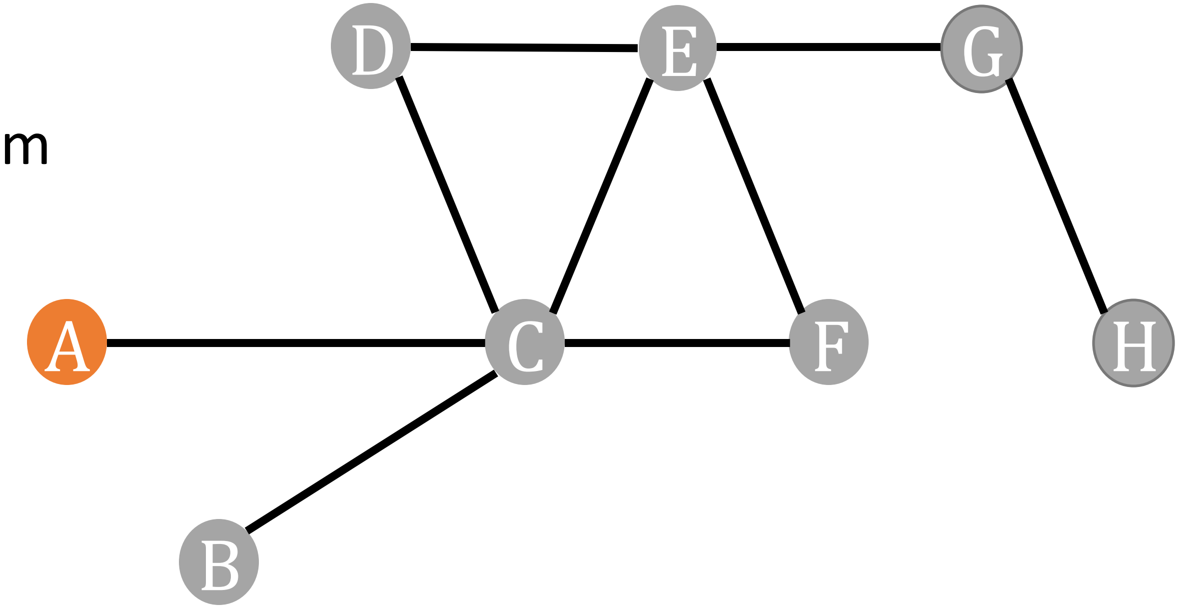


Parallel PageRank

Parallel BFS

Each iteration involves

1. processing k_r vertices concurrently from a list of vertices (**frontier**)
2. accessing neighbors of each vertex
3. updating vertex values
4. determining next frontier

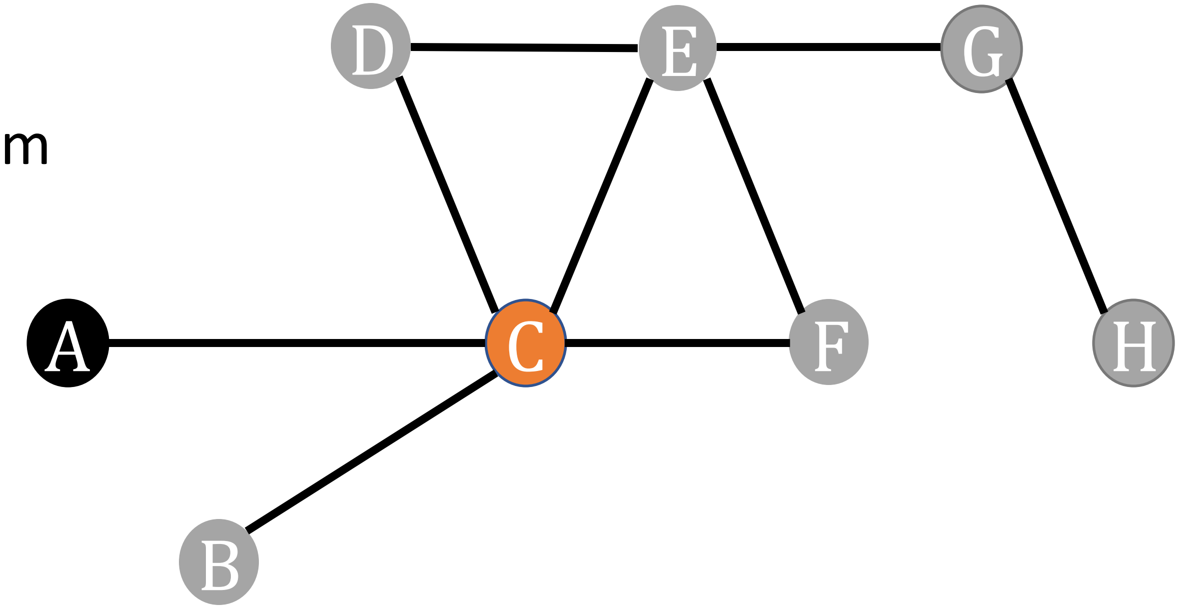


Parallel BFS

● processed nodes ● processing in progress ● yet to be processed

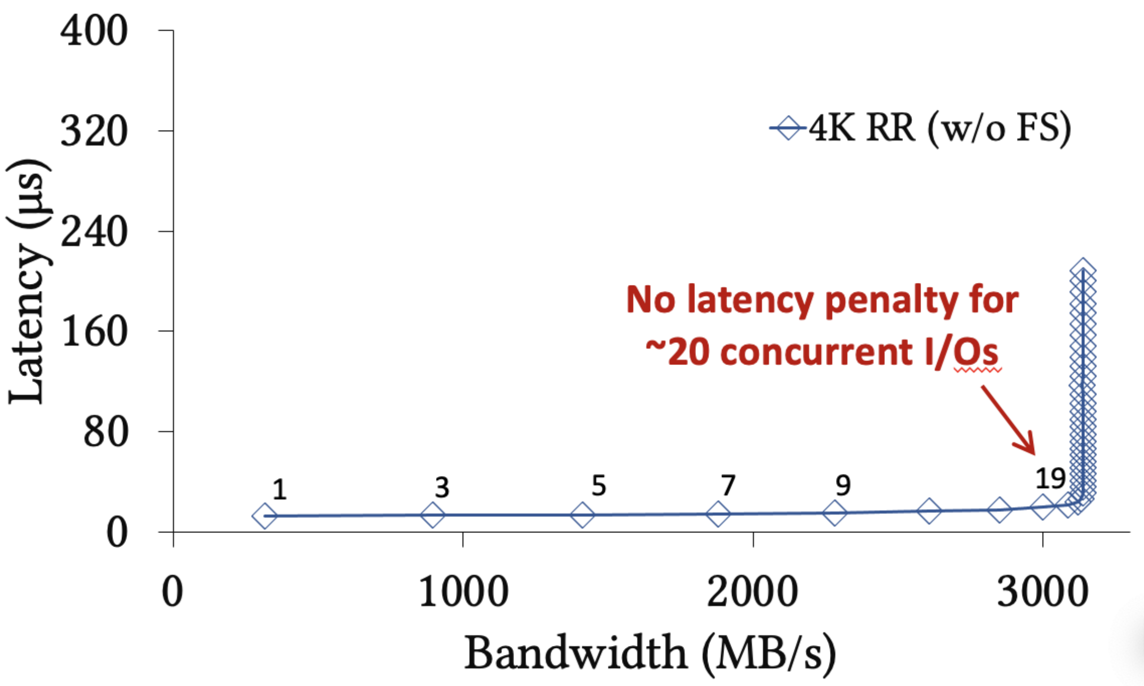
Each iteration involves

1. processing k_r vertices concurrently from a list of vertices (**frontier)**
2. accessing neighbors of each vertex
3. updating vertex values
4. determining next frontier



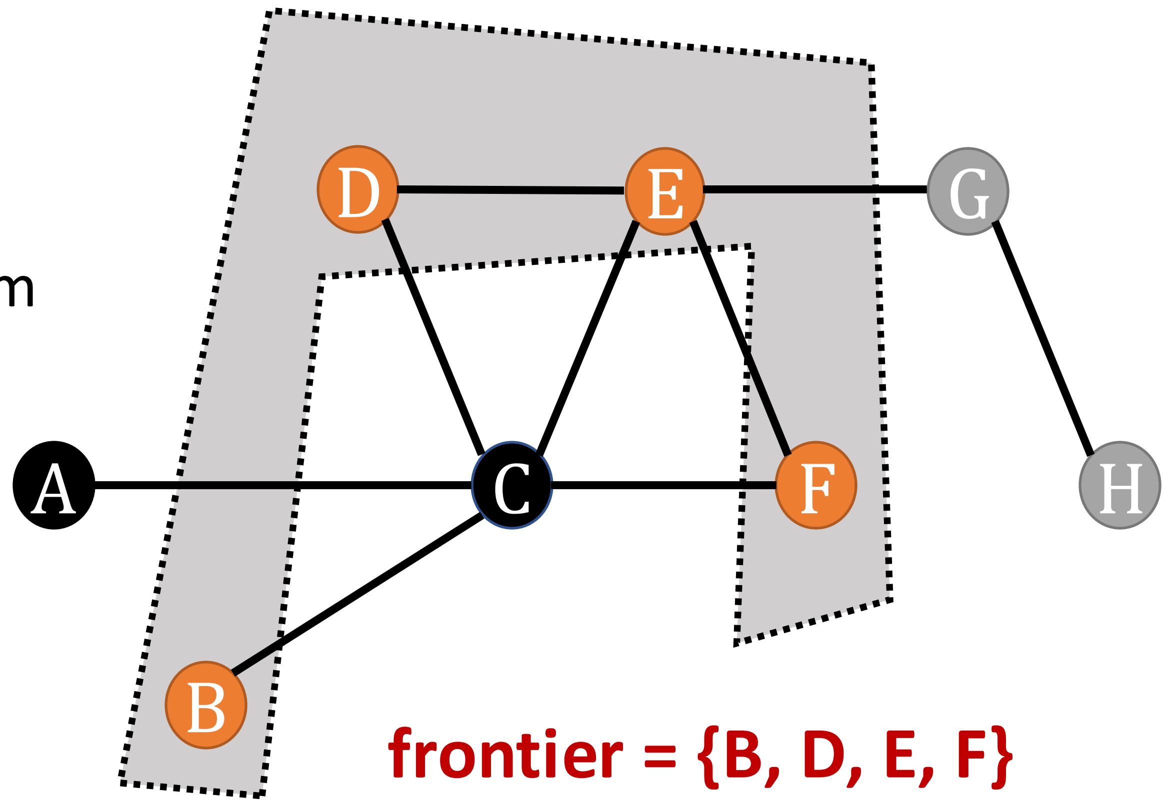
Parallel BFS

● processed nodes ● processing in progress ● yet to be processed



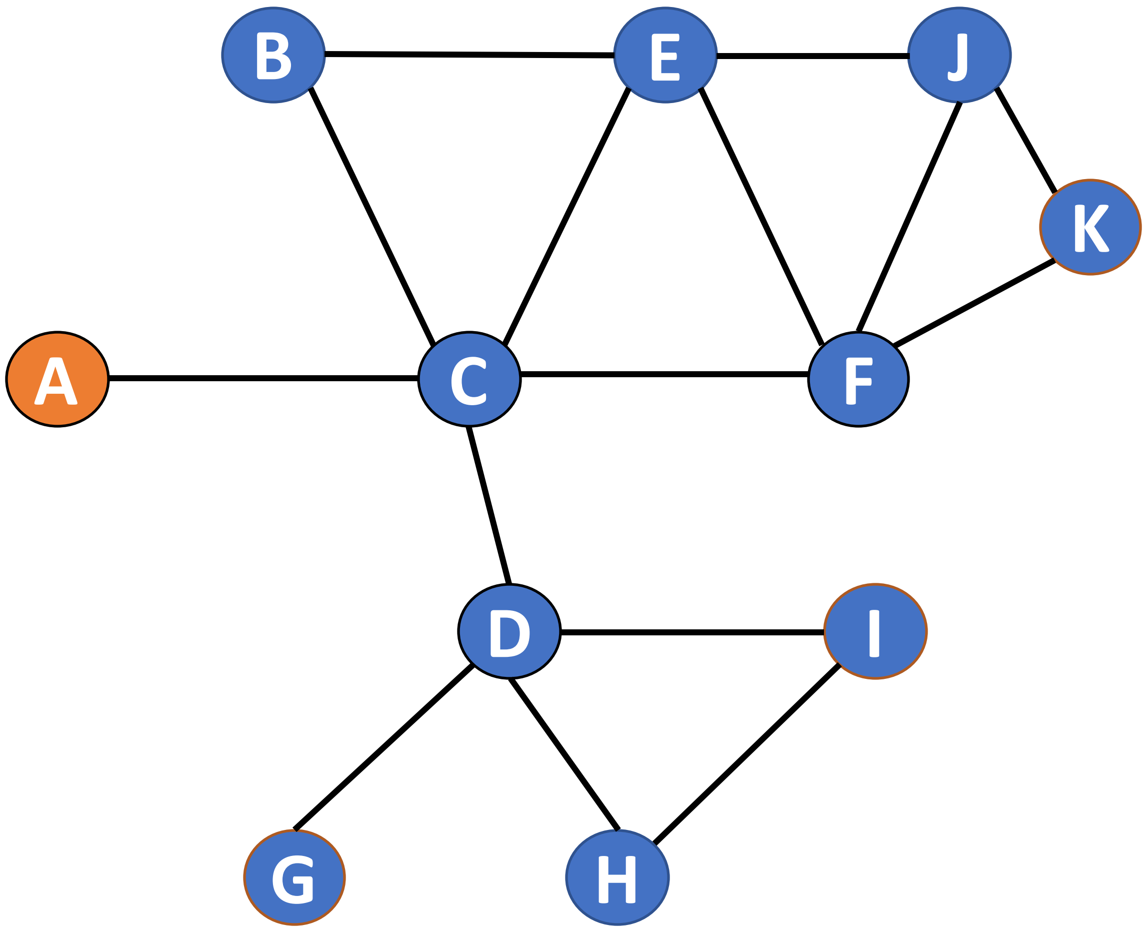
m

4. determining next frontier



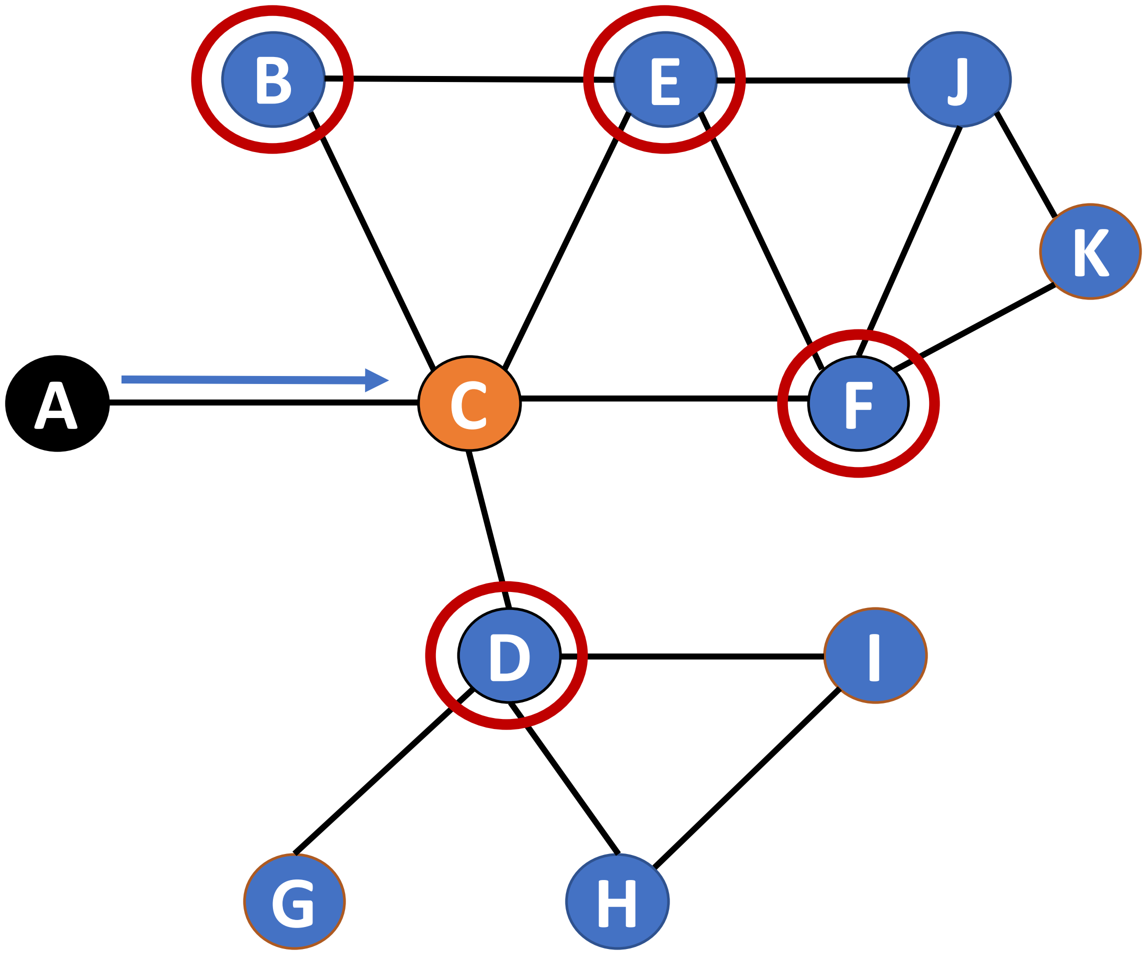
Parallel pseudo DFS

processed nodes   processing in progress   yet to be processed



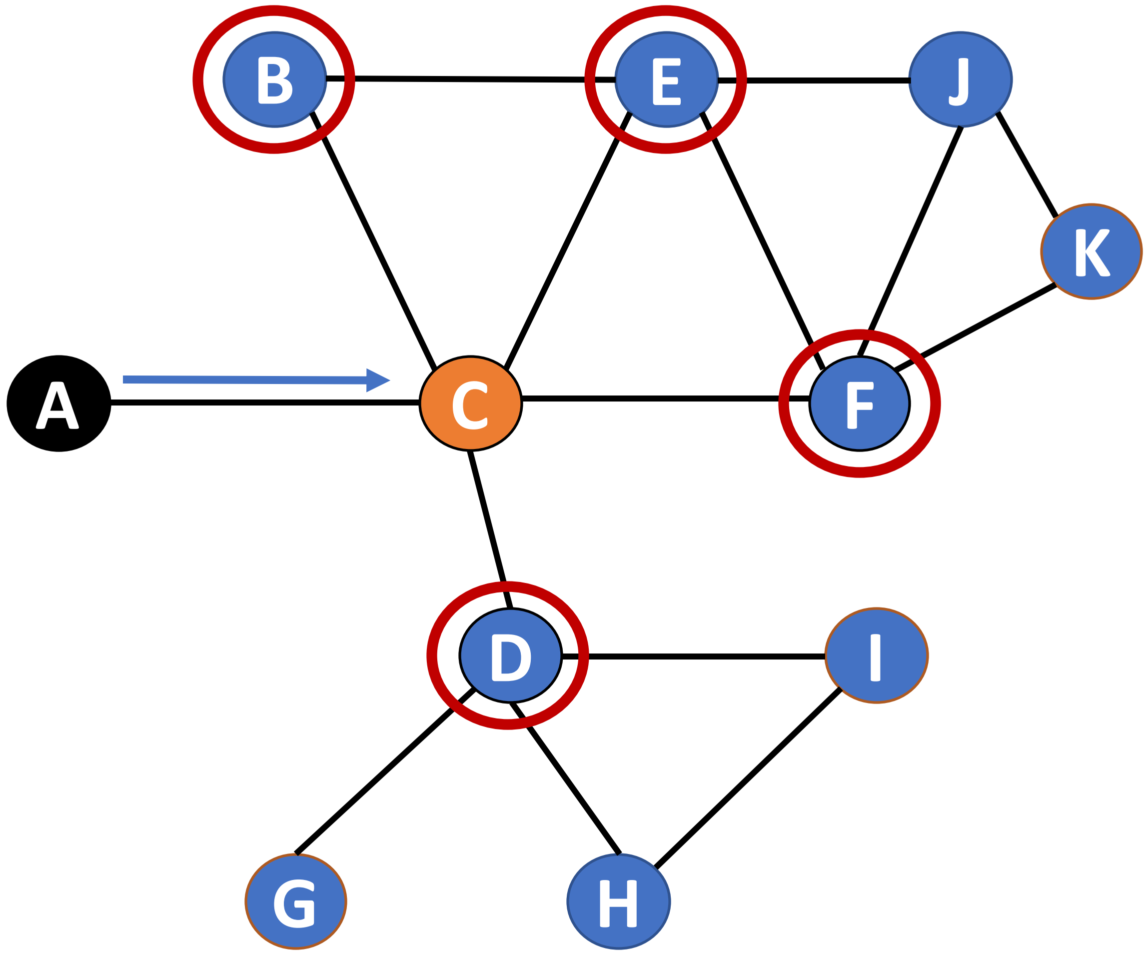
Parallel pseudo DFS

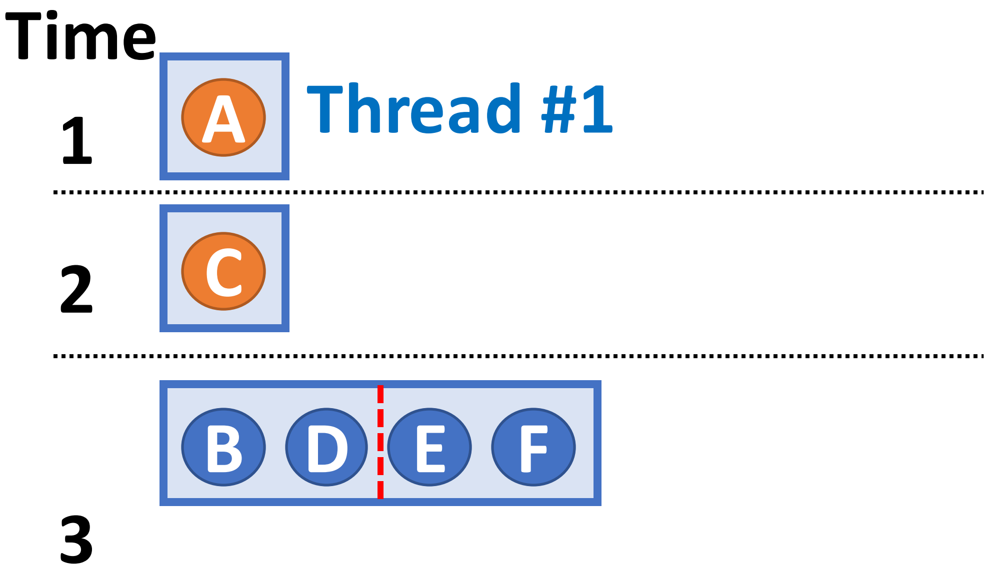
processed nodes | processing in progress | yet to be processed



Parallel pseudo DFS

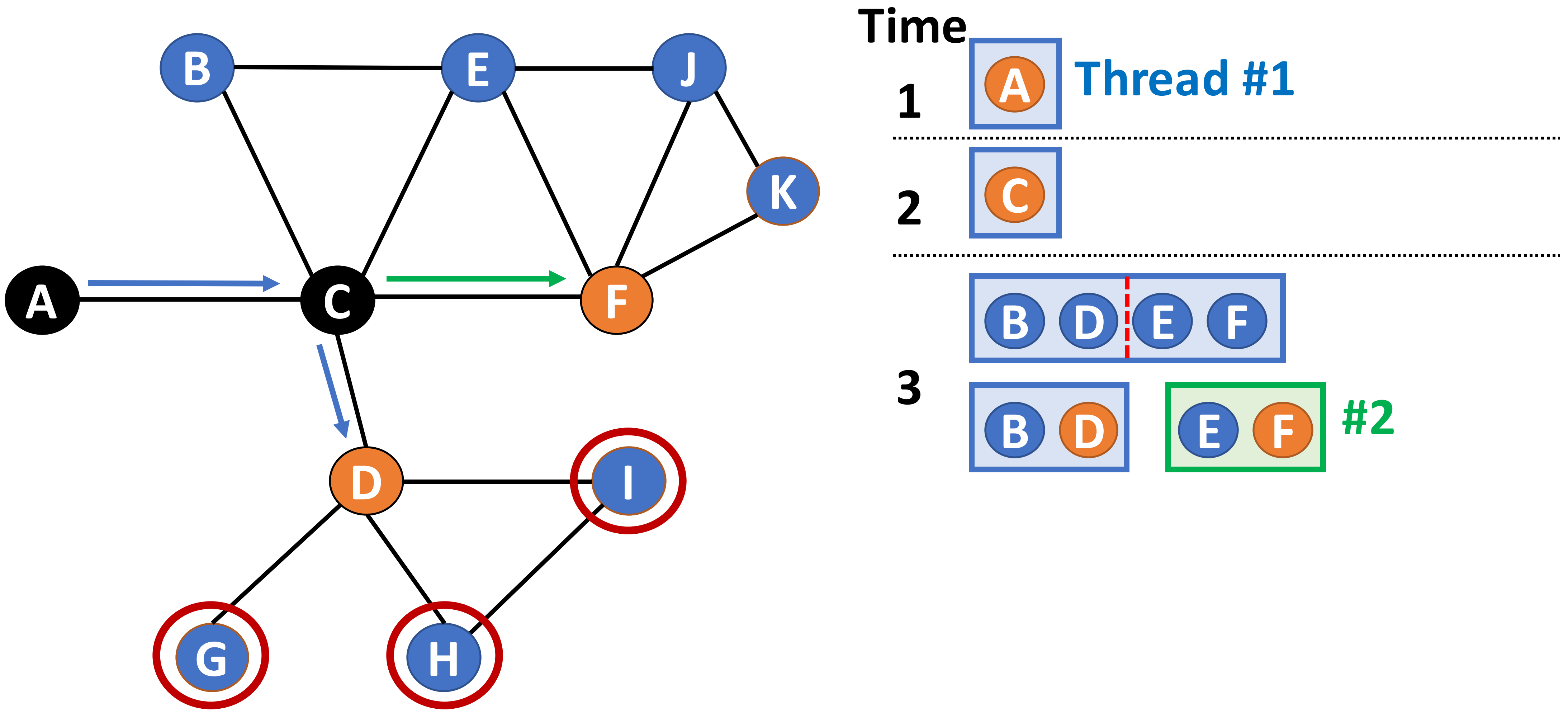
● processed nodes ● processing in progress ● yet to be processed





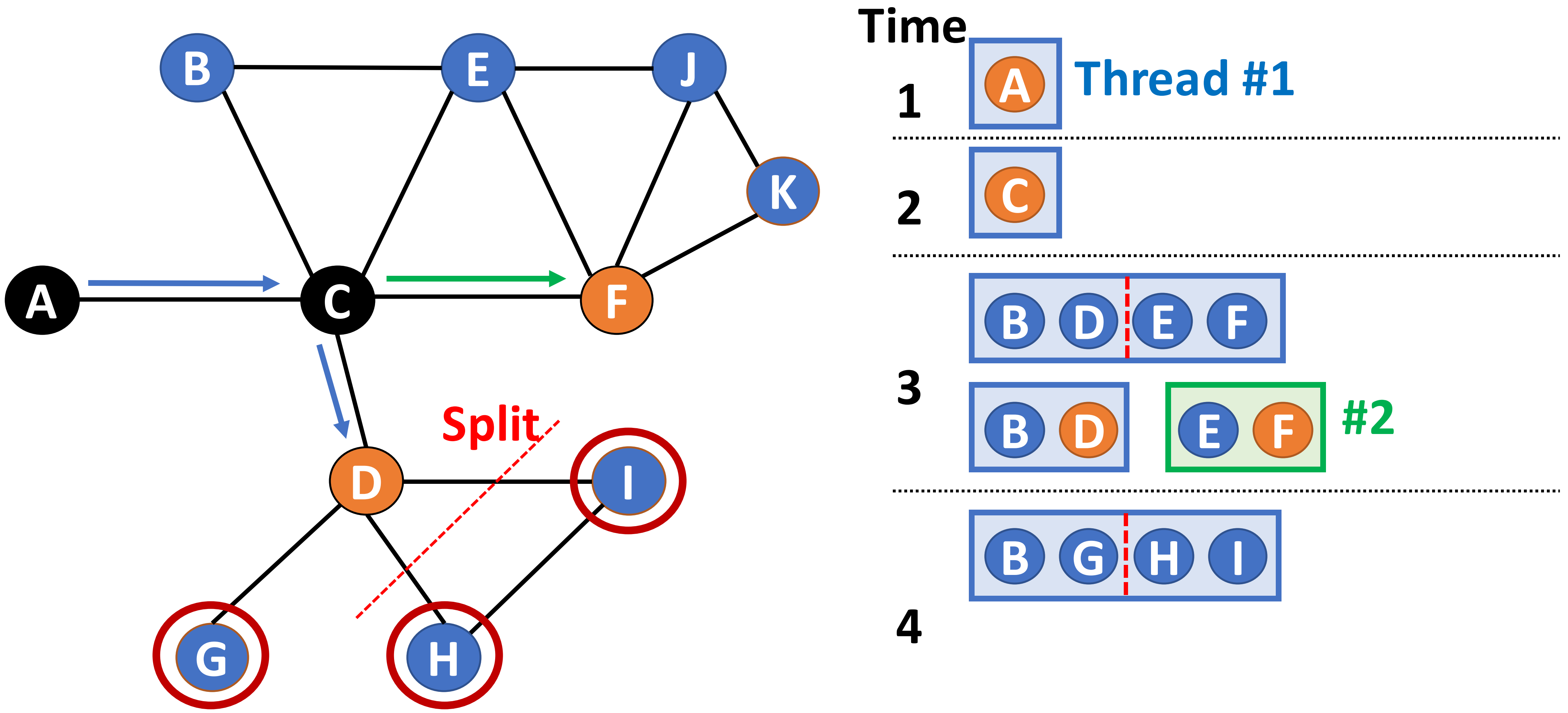
Parallel pseudo DFS

processed nodes — processing in progress — yet to be processed



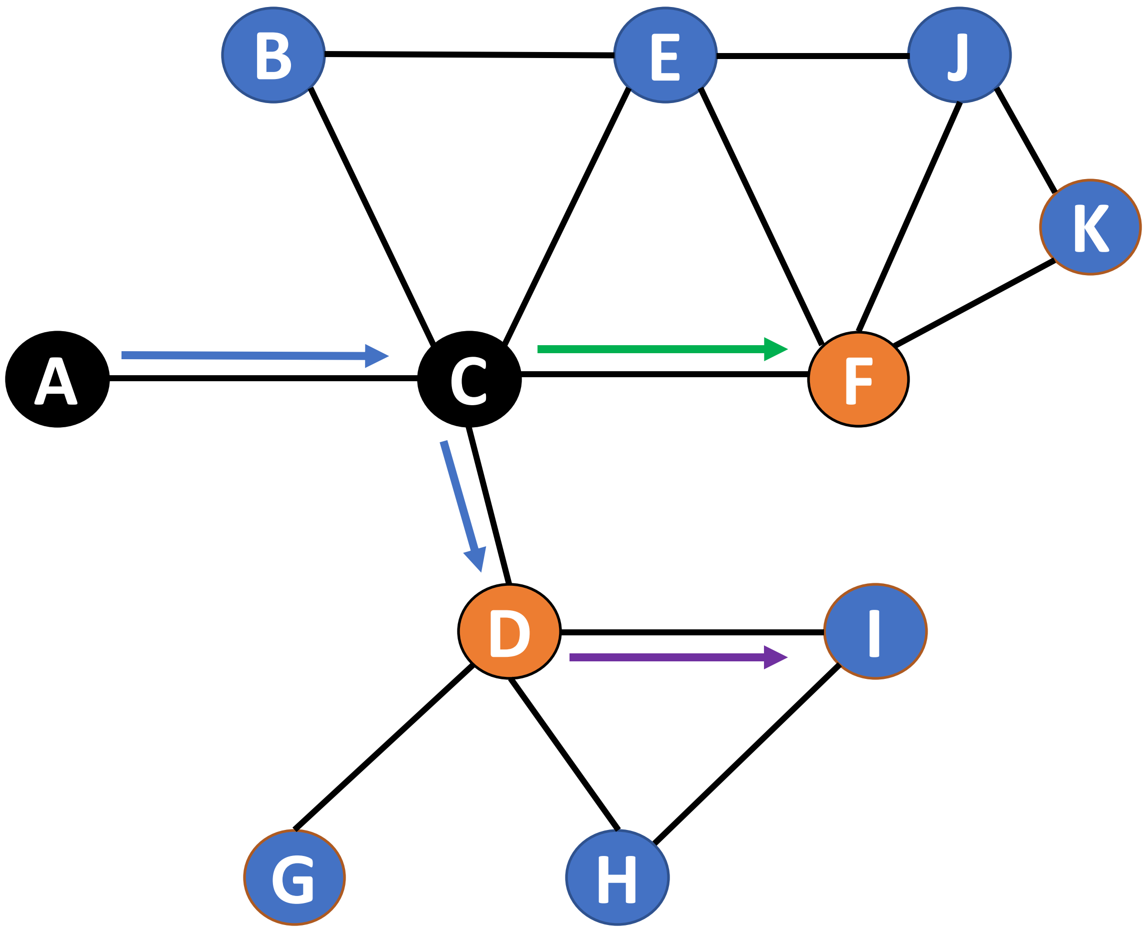
Parallel pseudo DFS

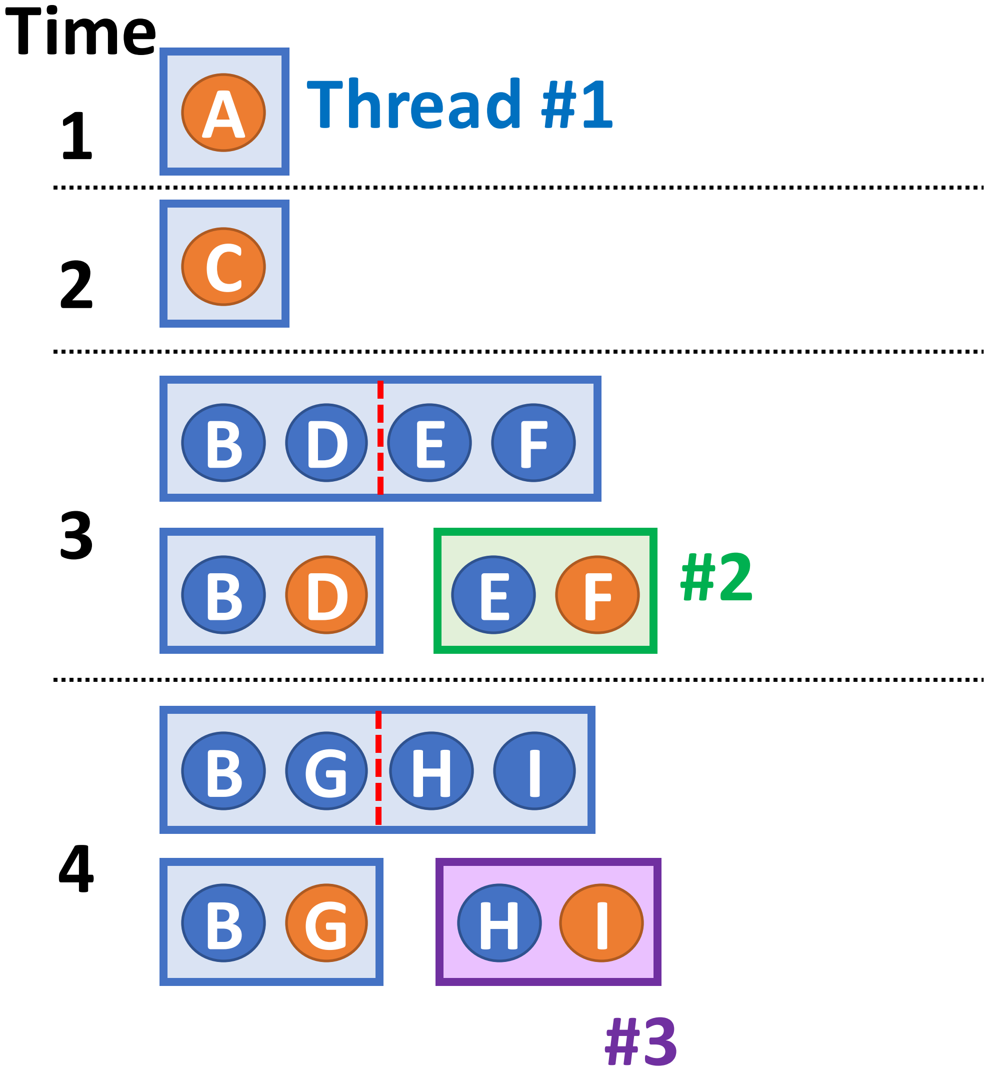
processed nodes   processing in progress   yet to be processed



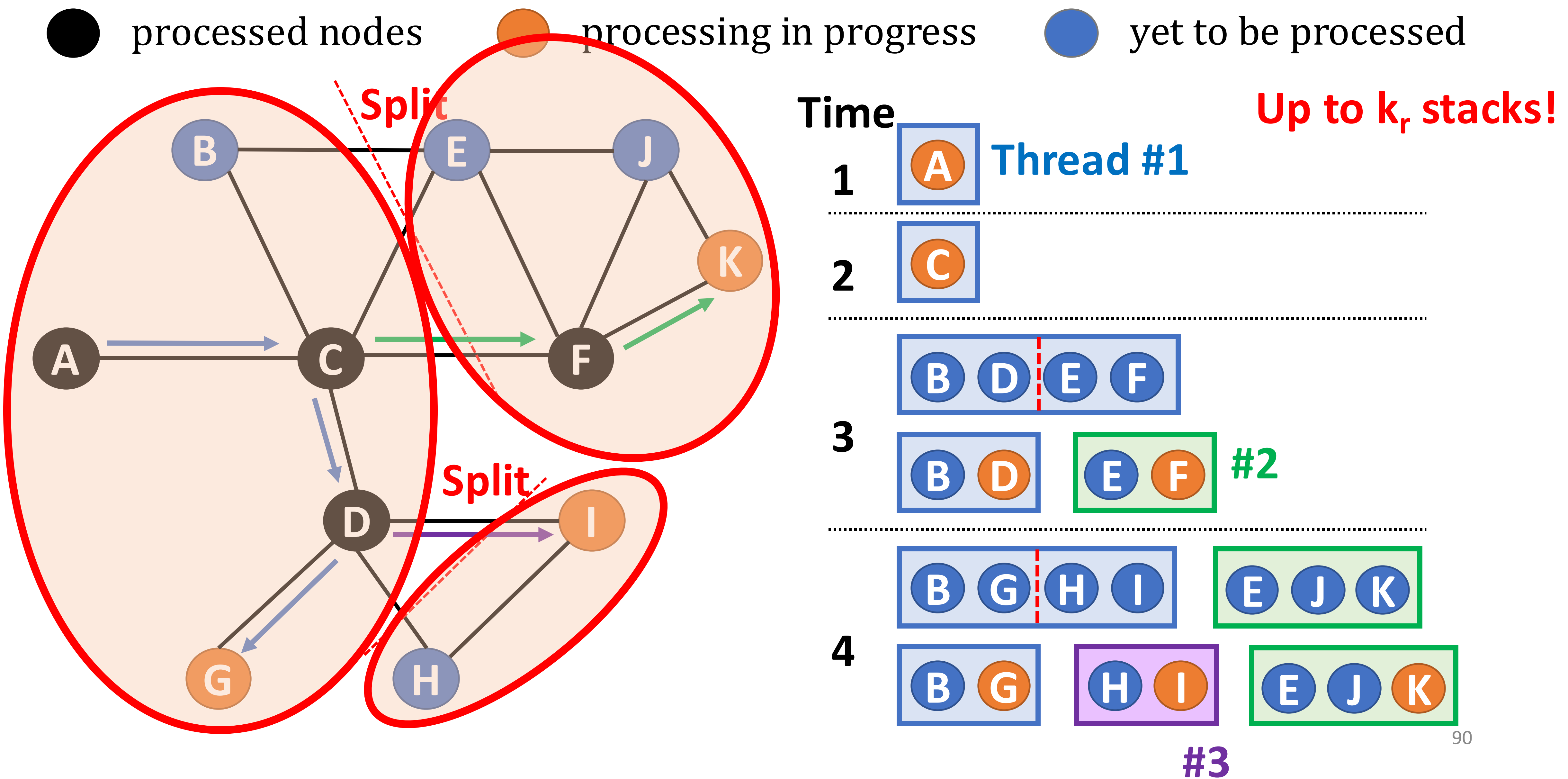
Parallel pseudo DFS

processed nodes   processing in progress   yet to be processed





Parallel pseudo DFS



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

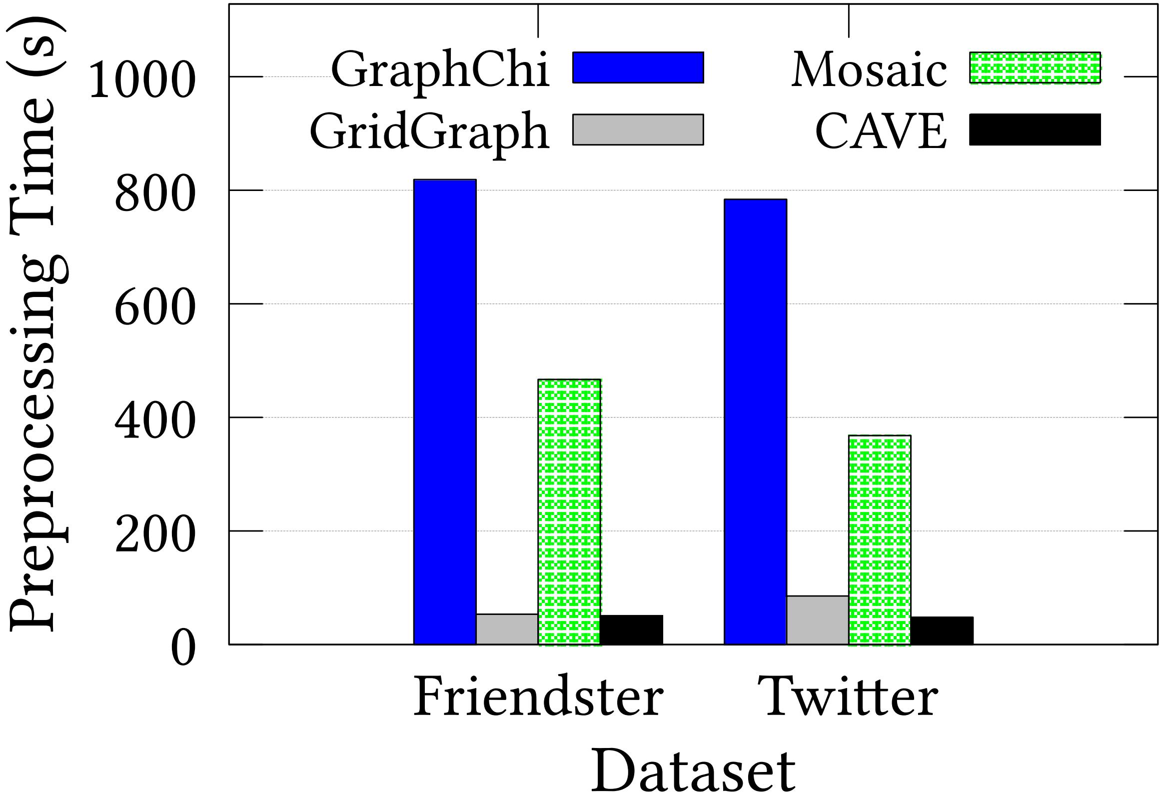
3 devices

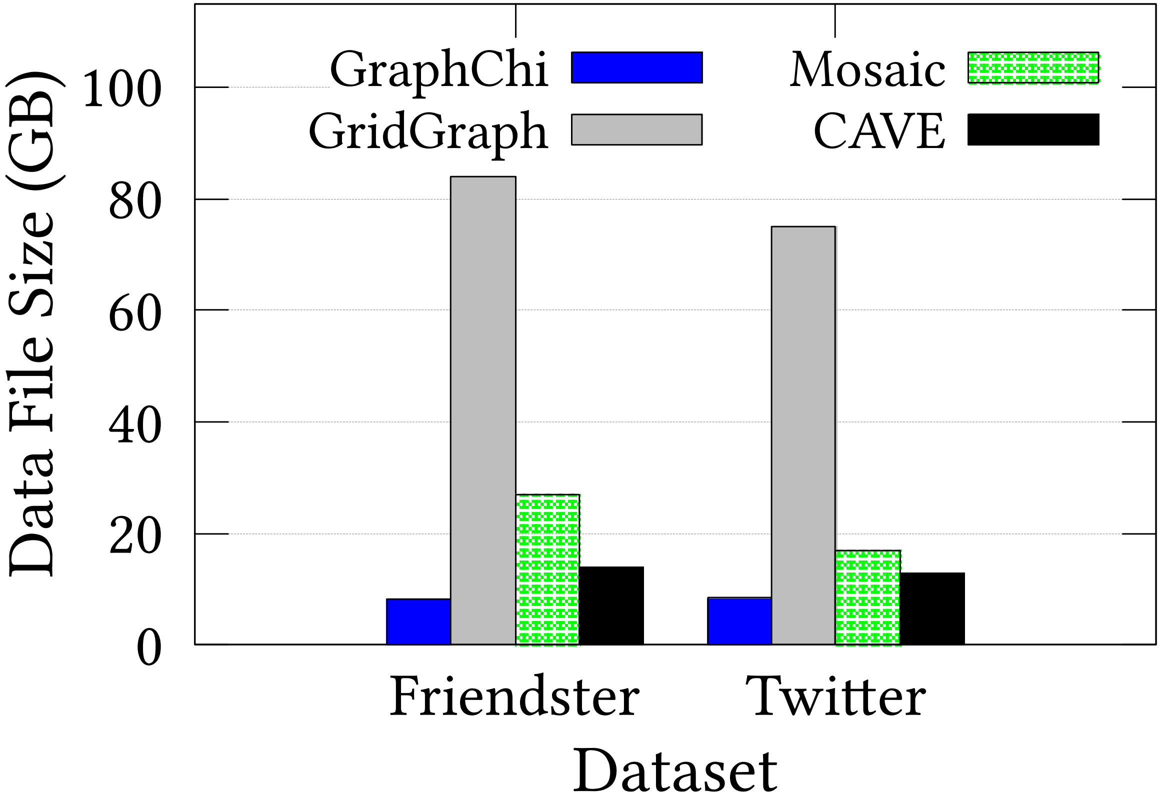
Optane SSD ($k_r = 6$)
PCIe SSD ($k_r = 80$)
SATA SSD ($k_r = 25$)

Approaches Used:

GraphChi, GridGraph, Mosaic, CAVE, CAVE_blocked

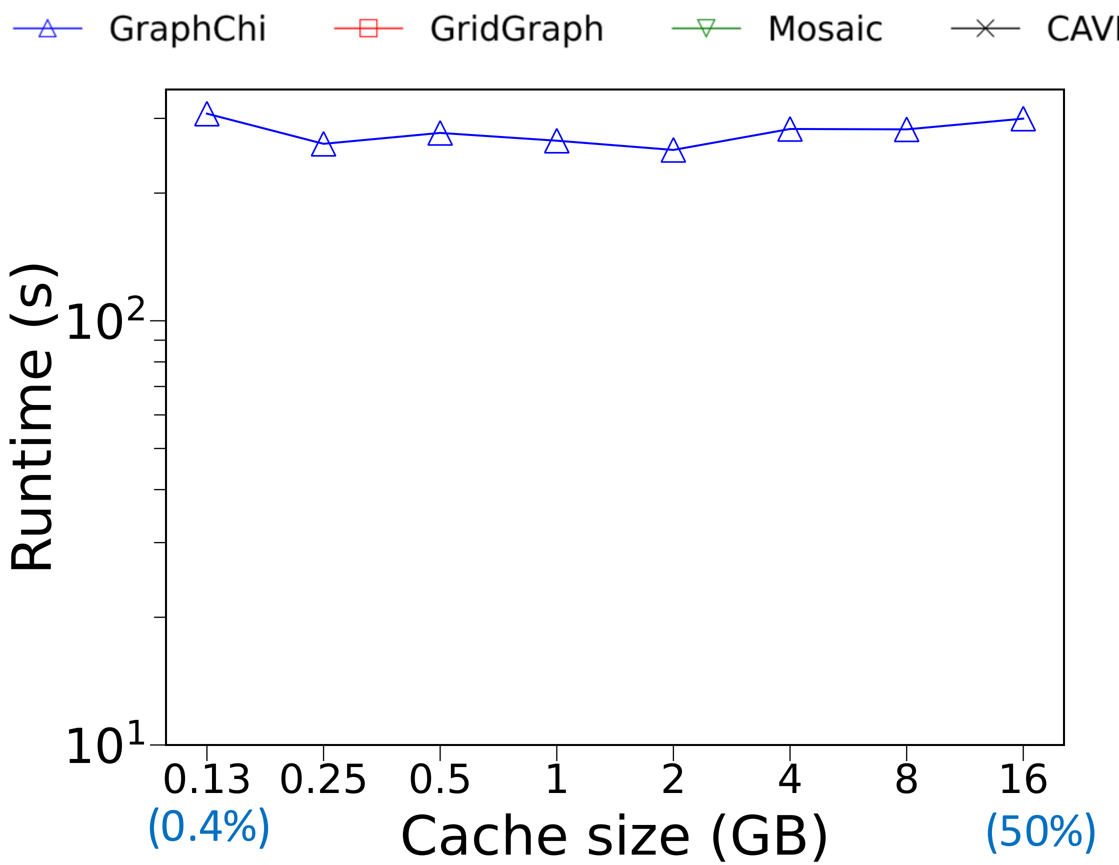
Preprocessing Time and Space Requirement





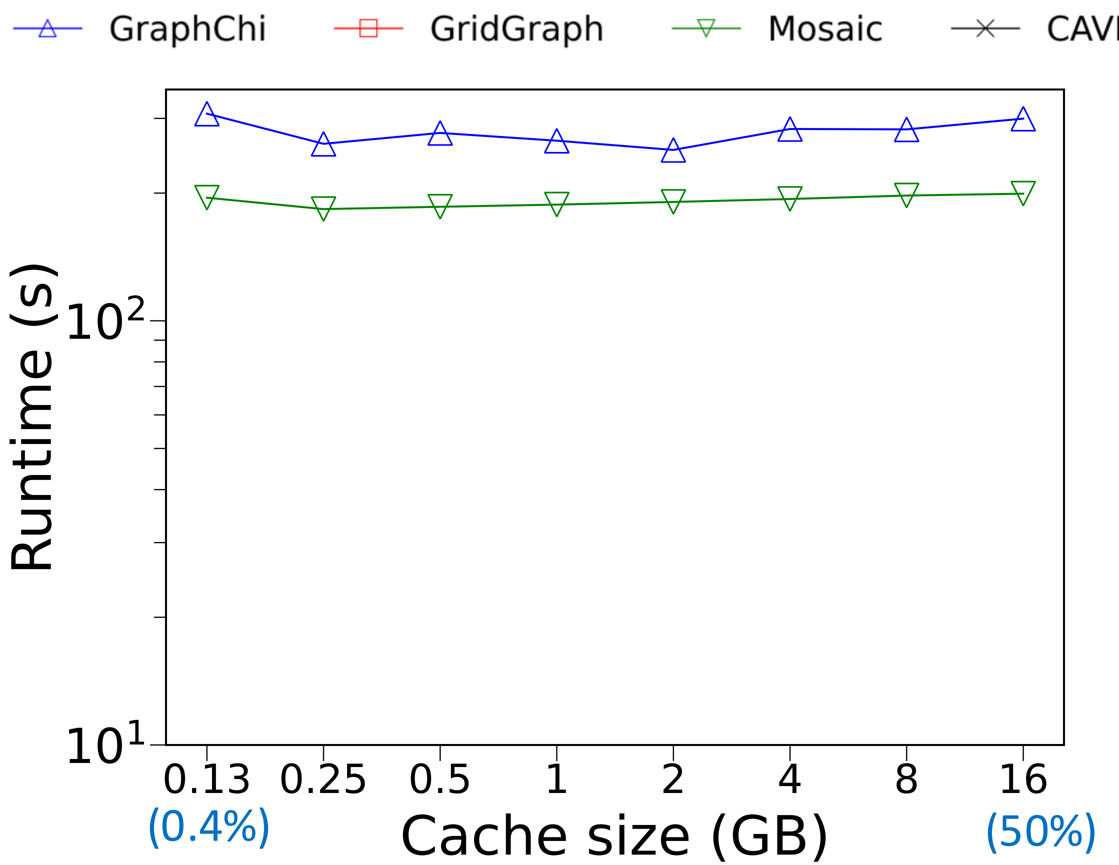
CAVE has **low preprocessing time** and **low file size**

Evaluation: Parallel BFS



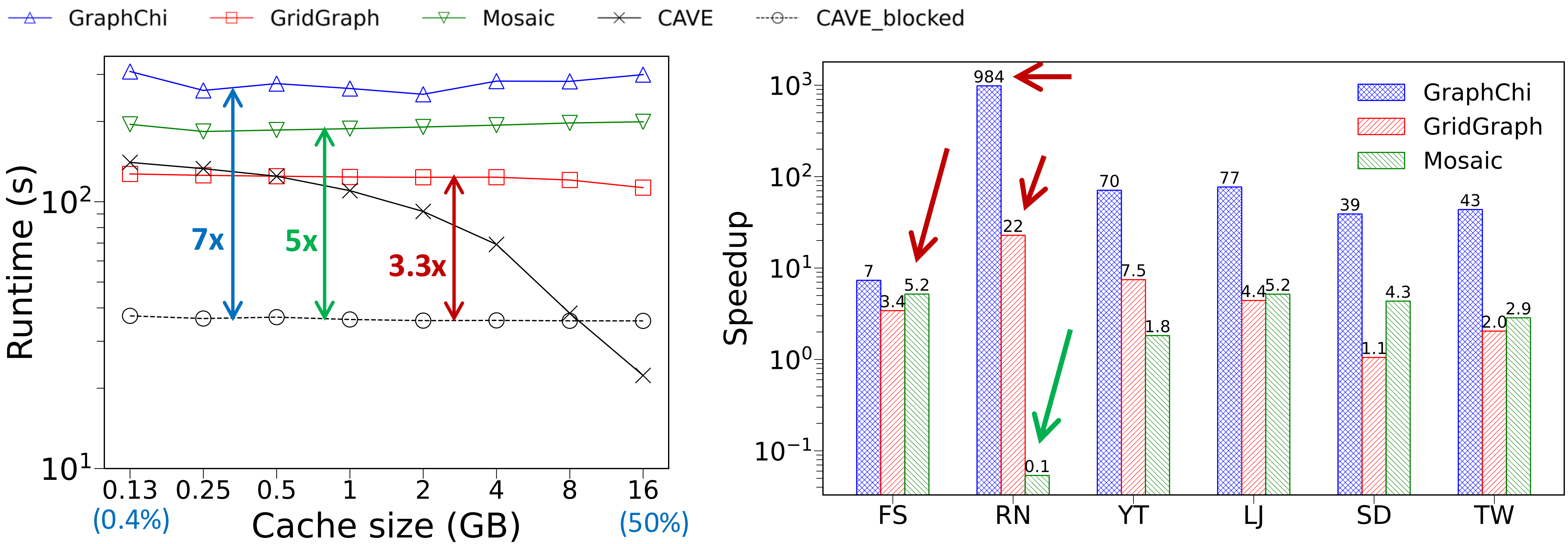
Dataset: FS

Evaluation: Parallel BFS



Dataset: FS

Evaluation: Parallel BFS



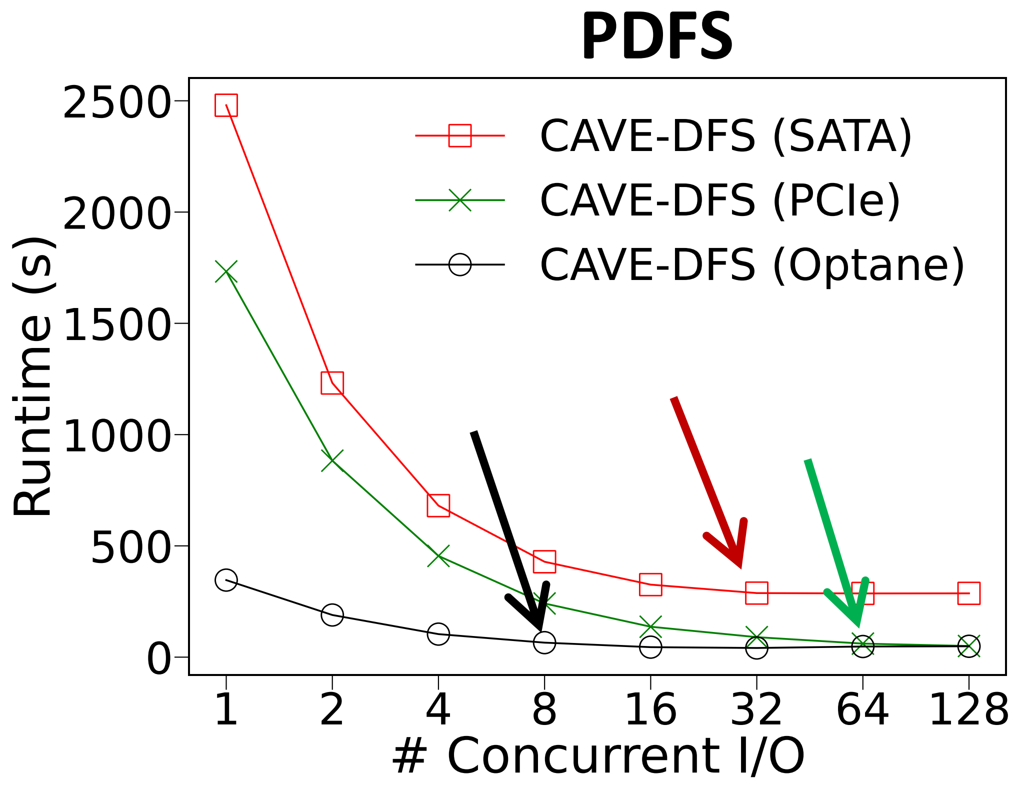
Dataset: FS

CAVE's Speedup

Both CAVE implementations outperforms GridGraph, Mosaic and GraphChi

CAVE Utilizes Concurrent I/O

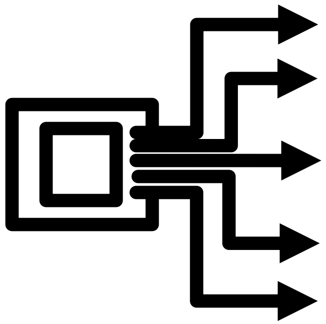
Dataset: FS



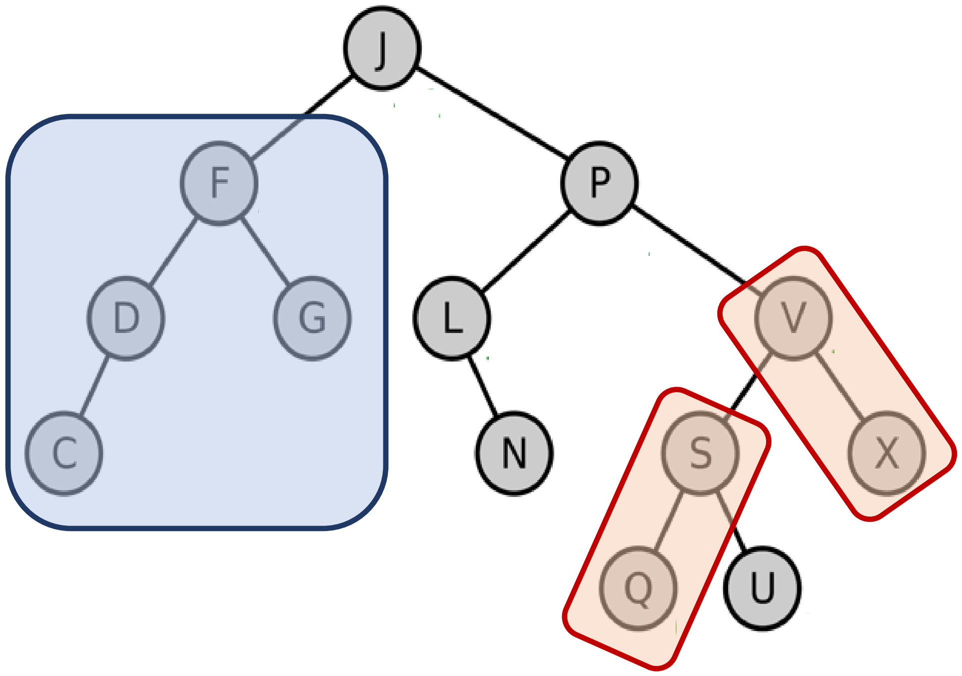
SATA SSD (k_r = *25*)
PCIe SSD (k_r = *80*)
Optane SSD (k_r = *6*)

Device gets saturated at *optimal concurrency*

Summary



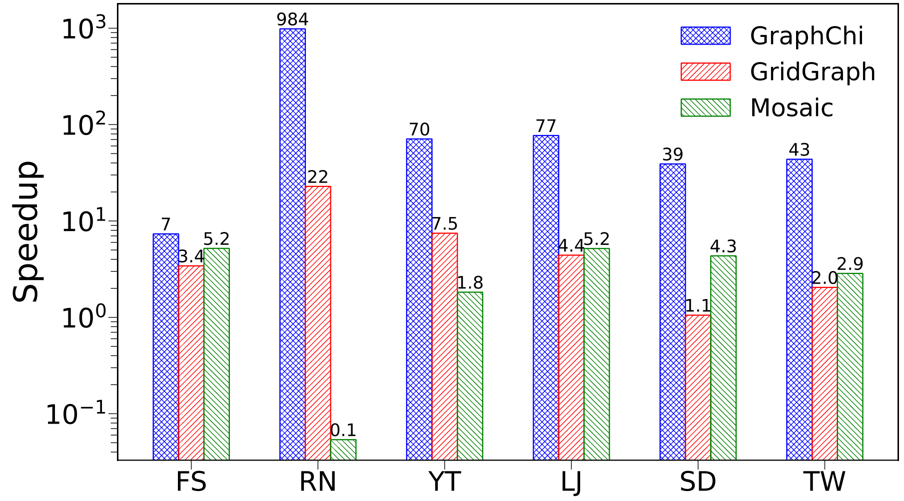
SSD concurrency can accelerate graph traversal



Intra- and inter-subgraph parallelization

Concurrency-Aware Graph (V, E) Manager

CAVE



CAVE implementations outperform SOA systems

Conclusion & Future Work

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

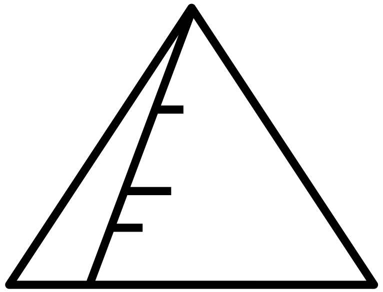
… not simply an engineering optimization

Build algorithms/data structures for storage devices

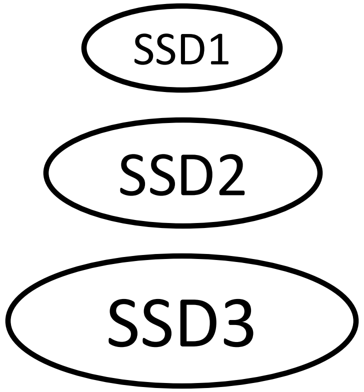
with asymmetry α and concurrency k

Stay Tuned!

index structures



tiered storage



Thank You!

Tarikul Islam Papon

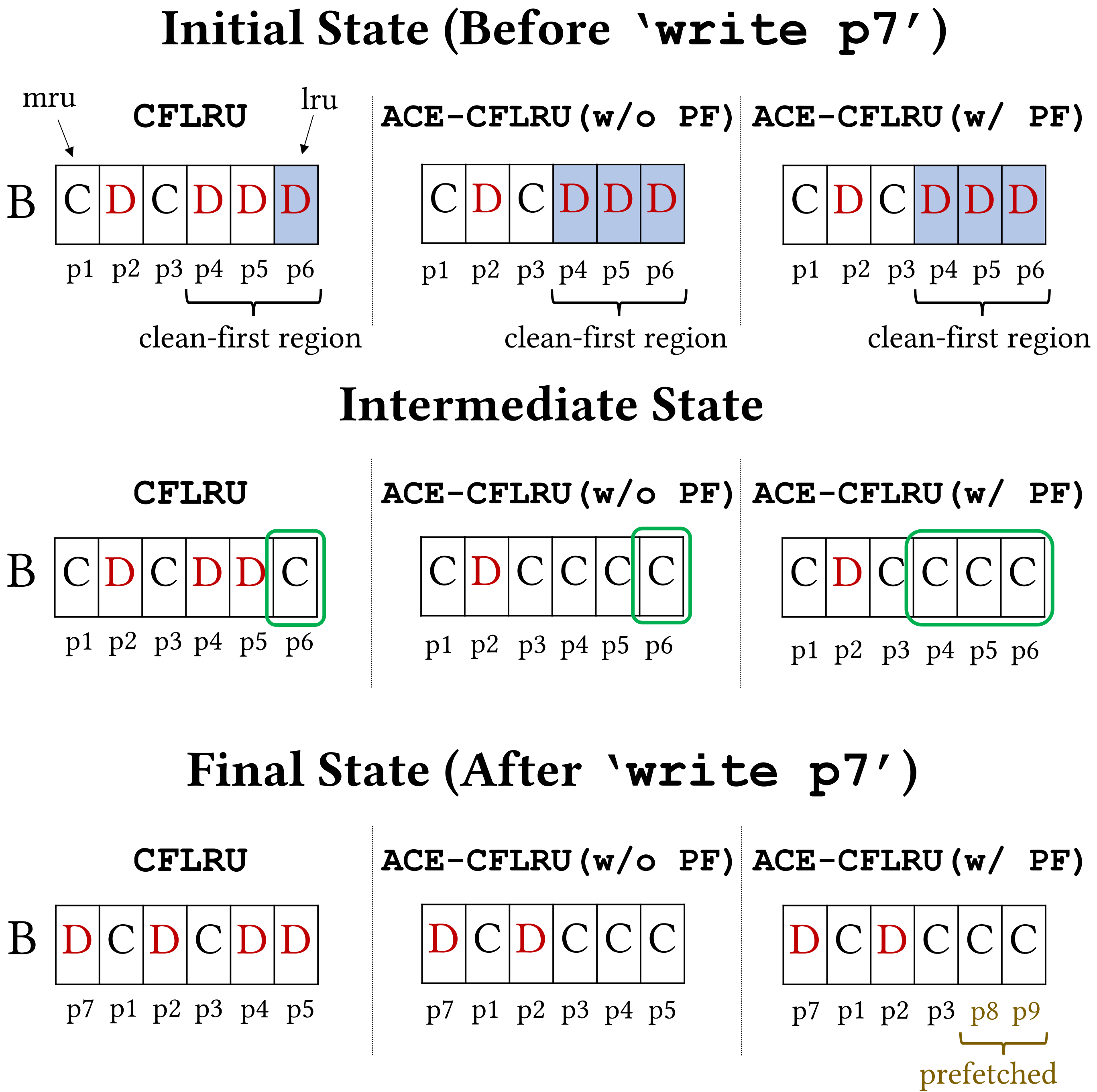
PhD Researcher

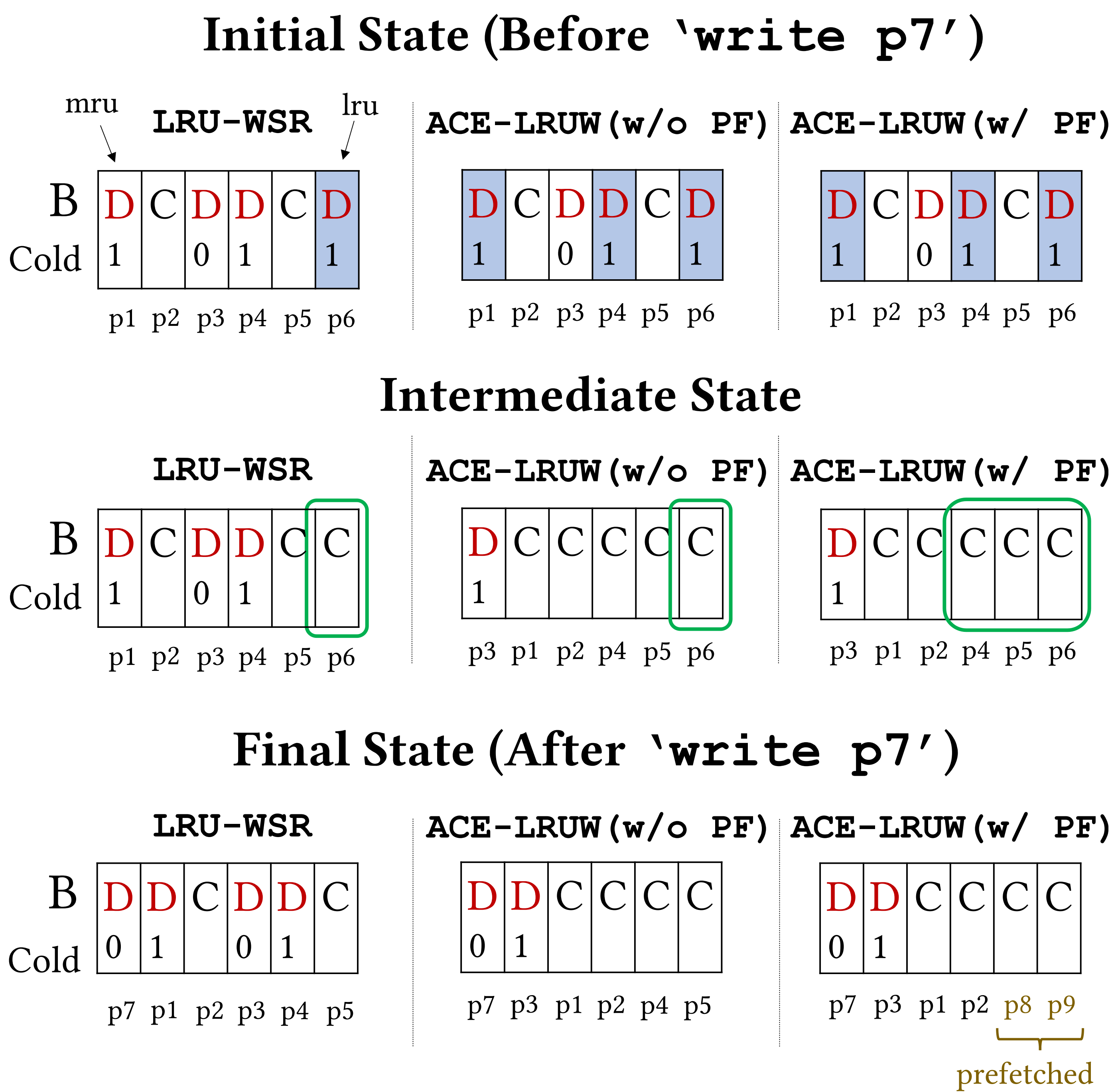
BOSTON UNIVERSITY

cs-people.bu.edu/papon/

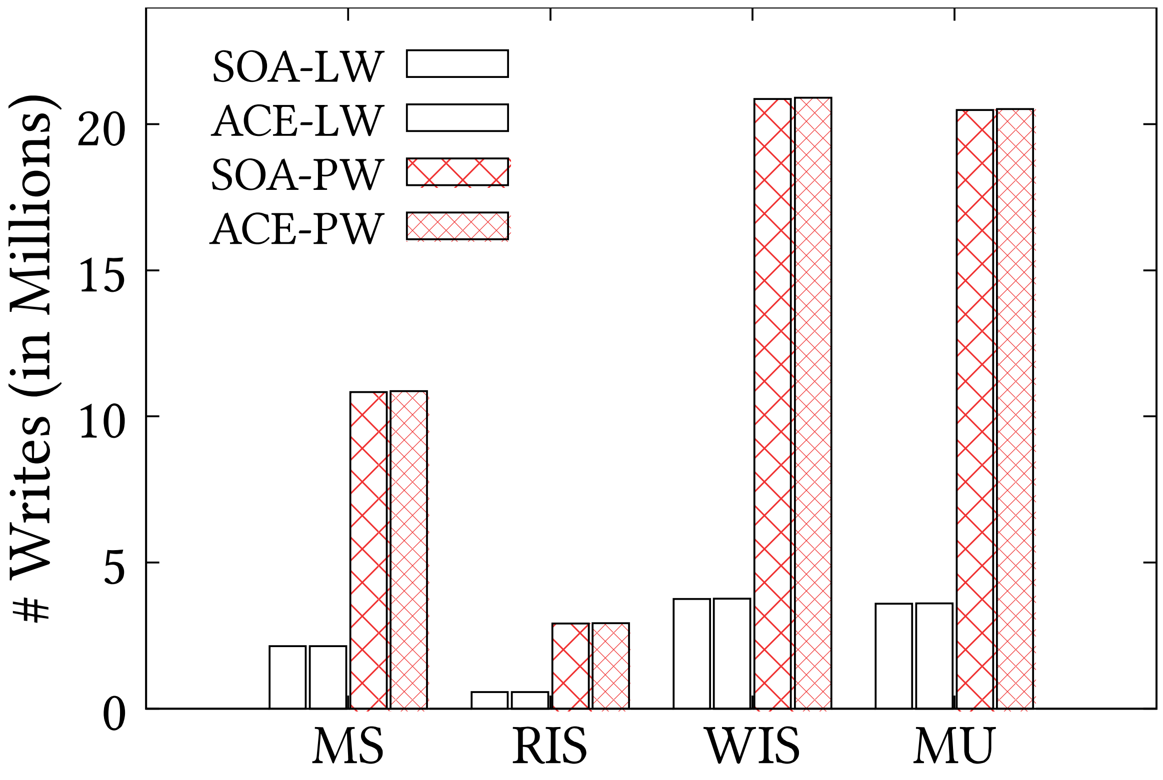
Read/Write Asymmetry - Example

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


Initial State (Before 'write p7')
mru
CFLRU
lru
B C D C D D D
p1 p2 p3 p4 p5 p6
clean-first region
ACE-CFLRU(w/o PF)
C D C D D D
p1 p2 p3 p4 p5 p6
clean-first region
ACE-CFLRU(w/ PF)
C D C D D D
p1 p2 p3 p4 p5 p6
clean-first region
Intermediate State
CFLRU
B C D C D D C
p1 p2 p3 p4 p5 p6
ACE-CFLRU(w/o PF)
C D C C C C
p1 p2 p3 p4 p5 p6
ACE-CFLRU(w/ PF)
C D C C C C
p1 p2 p3 p4 p5 p6
Final State (After 'write p7')
CFLRU
B D C D C D D
p7 p1 p2 p3 p4 p5
ACE-CFLRU(w/o PF)
D C D C C C
p7 p1 p2 p3 p4 p5
ACE-CFLRU(w/ PF)
D C D C C C
p7 p1 p2 p3 p8 p9
prefetched

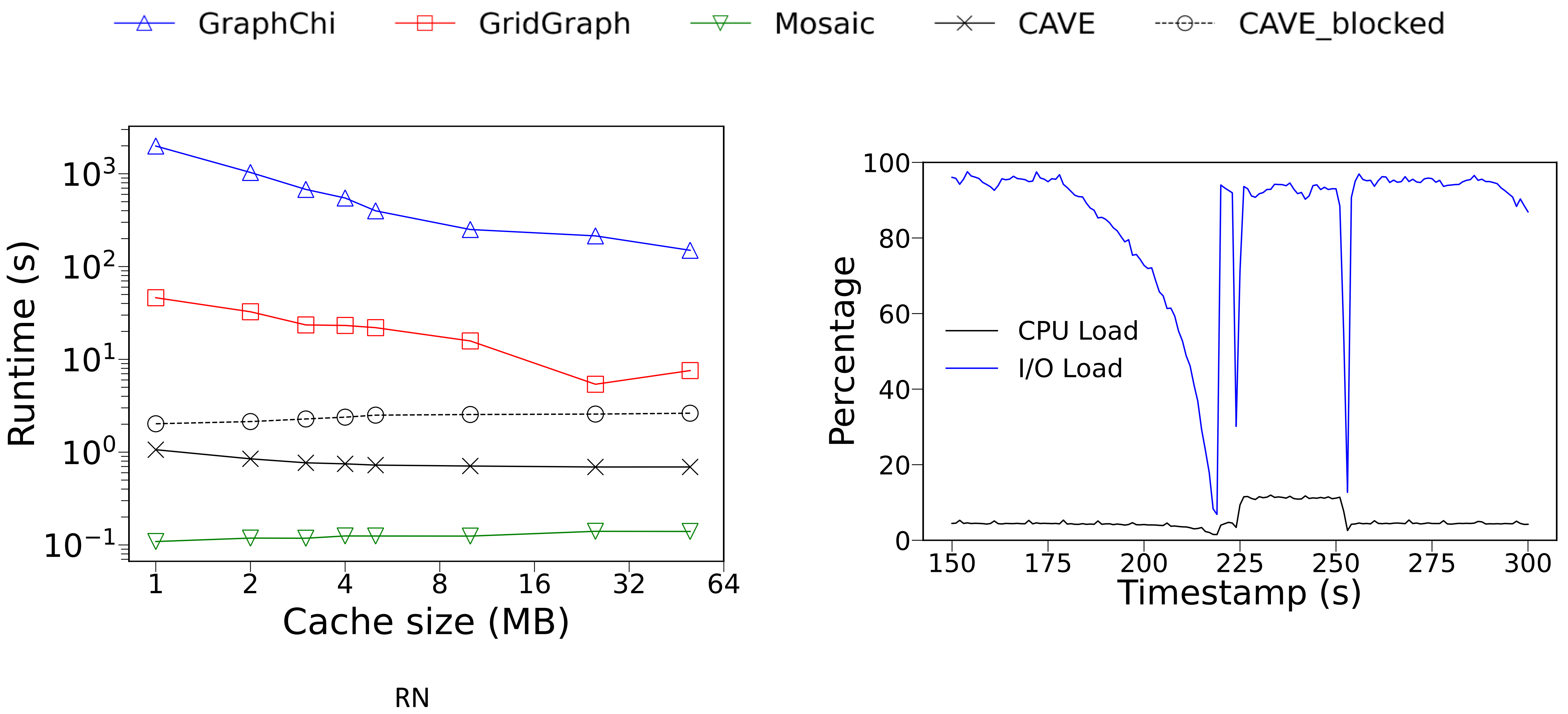

Initial State (Before 'write p7')
mru
lru
LRU-WSR
ACE-LRUW (w/o PF)
ACE-LRUW (w/ PF)
B
Cold
D C D D C D
1 0 1 1
p1 p2 p3 p4 p5 p6
Intermediate State
LRU-WSR
ACE-LRUW (w/o PF)
ACE-LRUW (w/ PF)
D C D D C C
1 0 1
p1 p2 p3 p4 p5 p6
D C C C C C
1
p3 p1 p2 p4 p5 p6
Final State (After 'write p7')
LRU-WSR
ACE-LRUW (w/o PF)
ACE-LRUW (w/ PF)
D D C D D C
0 1 0 1
p7 p1 p2 p3 p4 p5
D D C C C C
0 1
p7 p3 p1 p2 p4 p5
p7 p3 p1 p2 p8 p9
prefetched

Experimental Evaluation



Building Block for Parallelization

- **`ProcessQueue`**
  - processes all vertices of the same level in parallel
  - works on vertex level
  - effective for sparse graph
- **`ProcessQueueBlock`**
  - works on edge level
  - all edge blocks of the vertices of the same level is retrieved and processed
  - effective for dense graph


GraphChi
GridGraph
Mosaic
CAVE
CAVE_blocked
Runtime (s)
10^3
10^2
10^1
10^0
10^{-1}
1
2
4
8
16
32
64
Cache size (MB)
RN
Percentage
100
80
60
40
20
0
CPU Load
I/O Load
150
175
200
225
250
275
300
Timestamp (s)