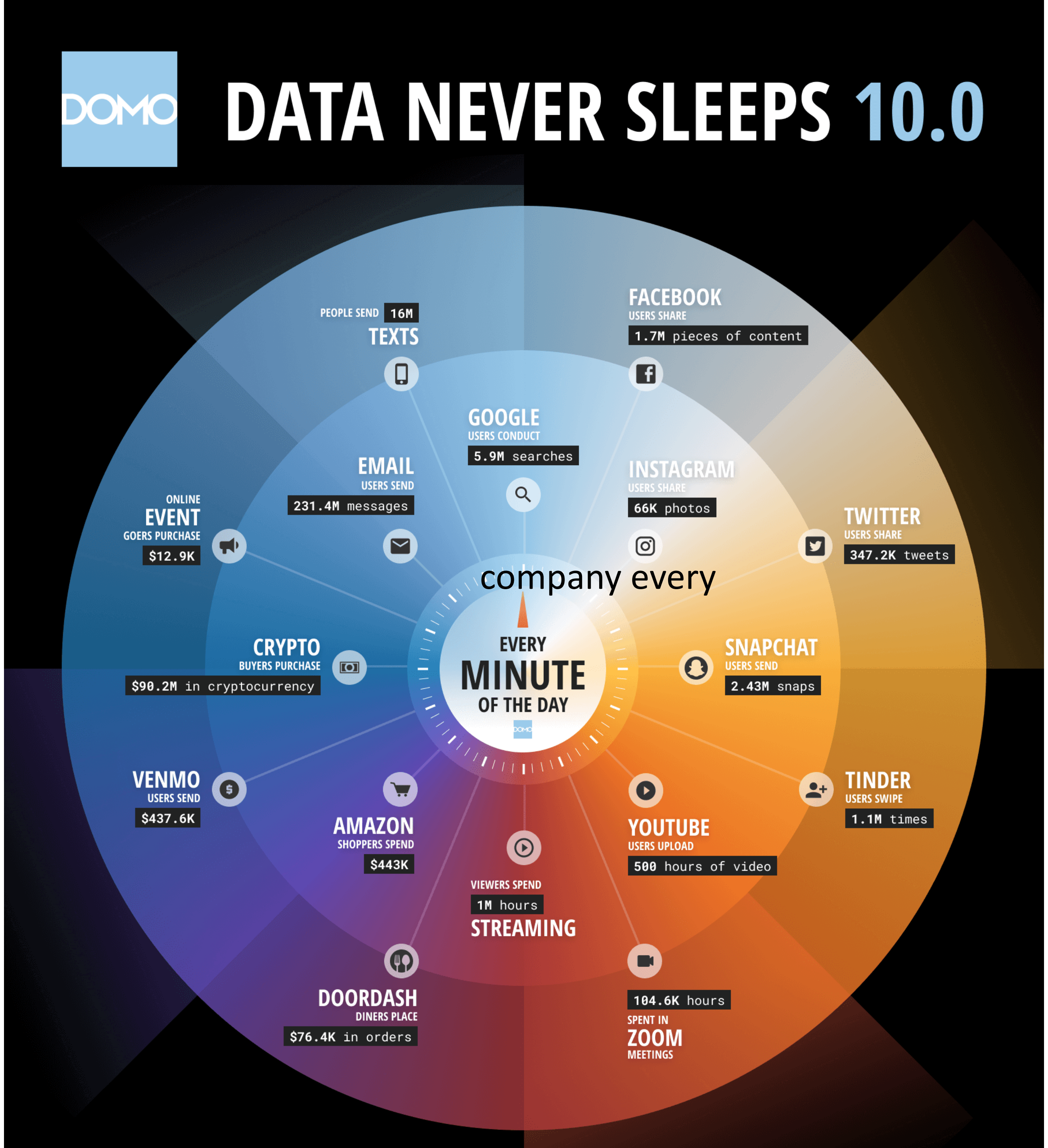
Optimizing Data Systems for Modern Storage Technology

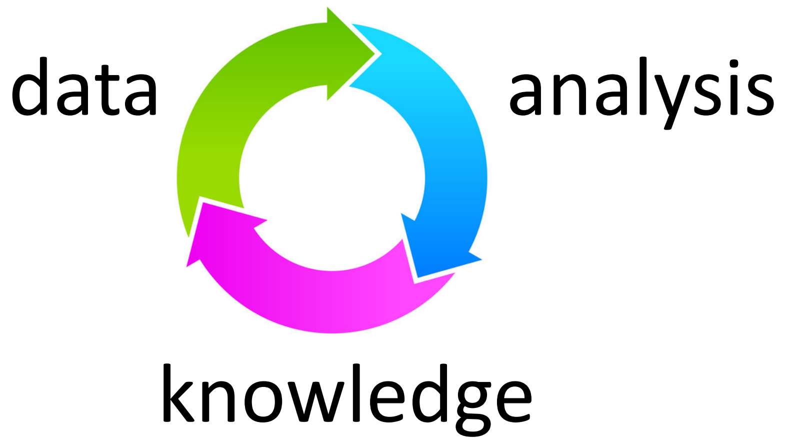
Tarikul Islam Papon

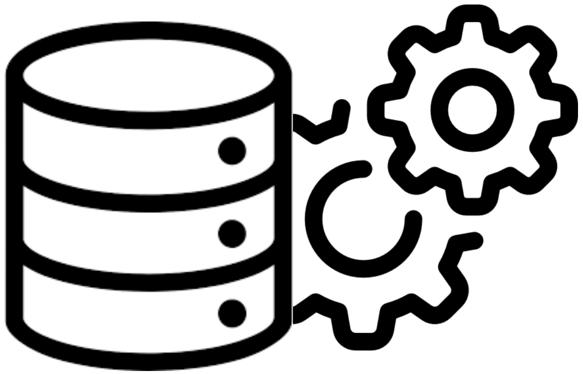
PhD Researcher

BOSTON UNIVERSITY



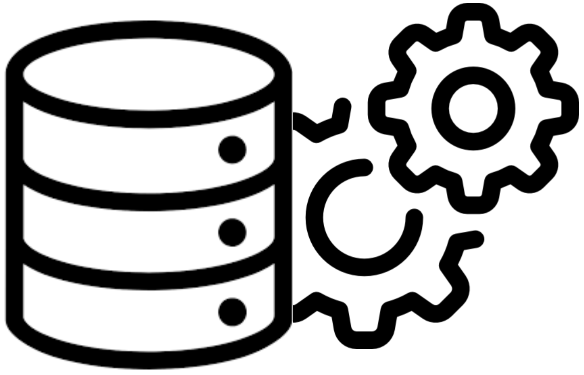
company every



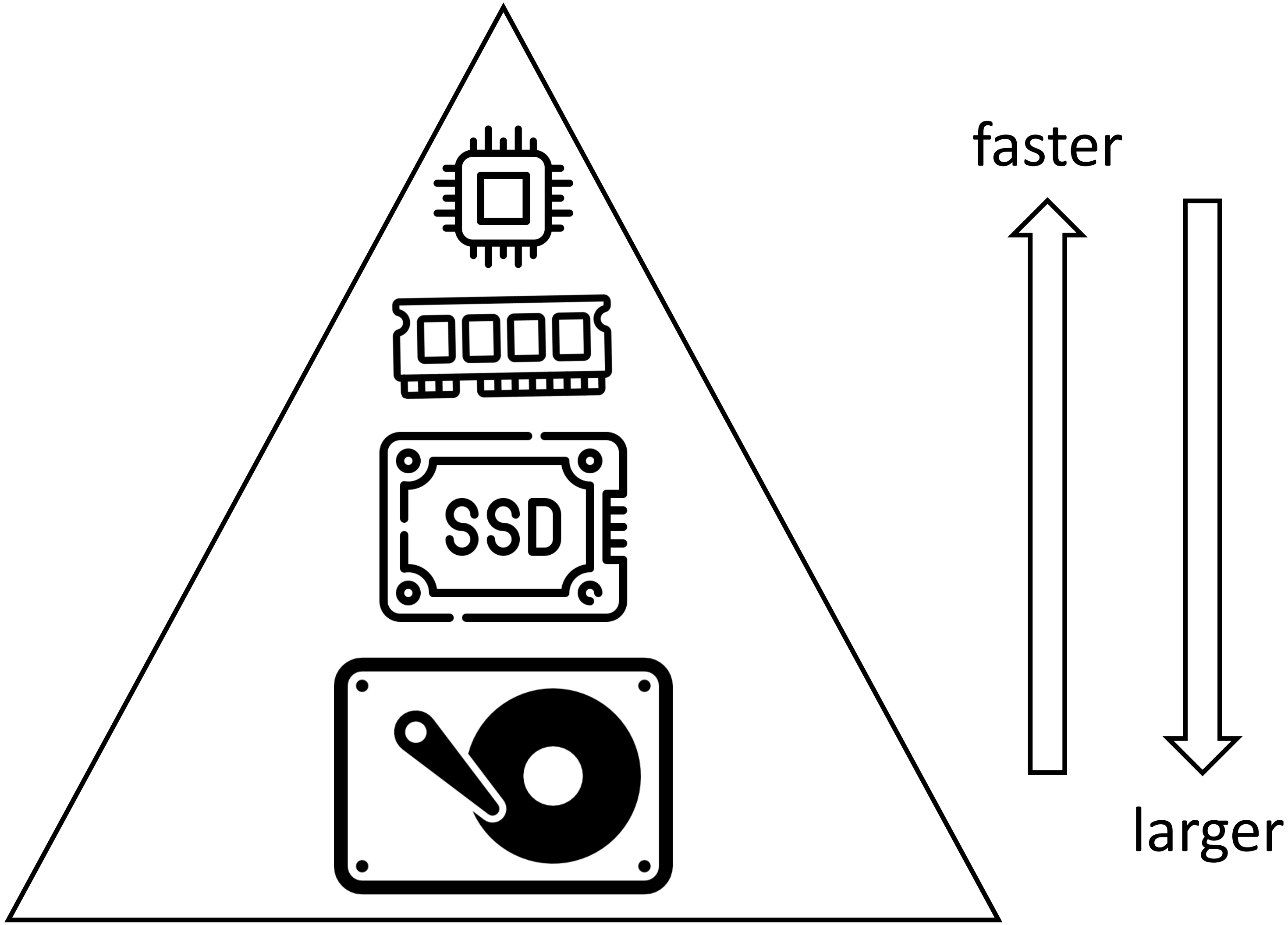


Data Systems

Data Systems & Hardware



Data Systems



Memory Hierarchy

Hardware Trends

Evolution of Storage Technology

Tape

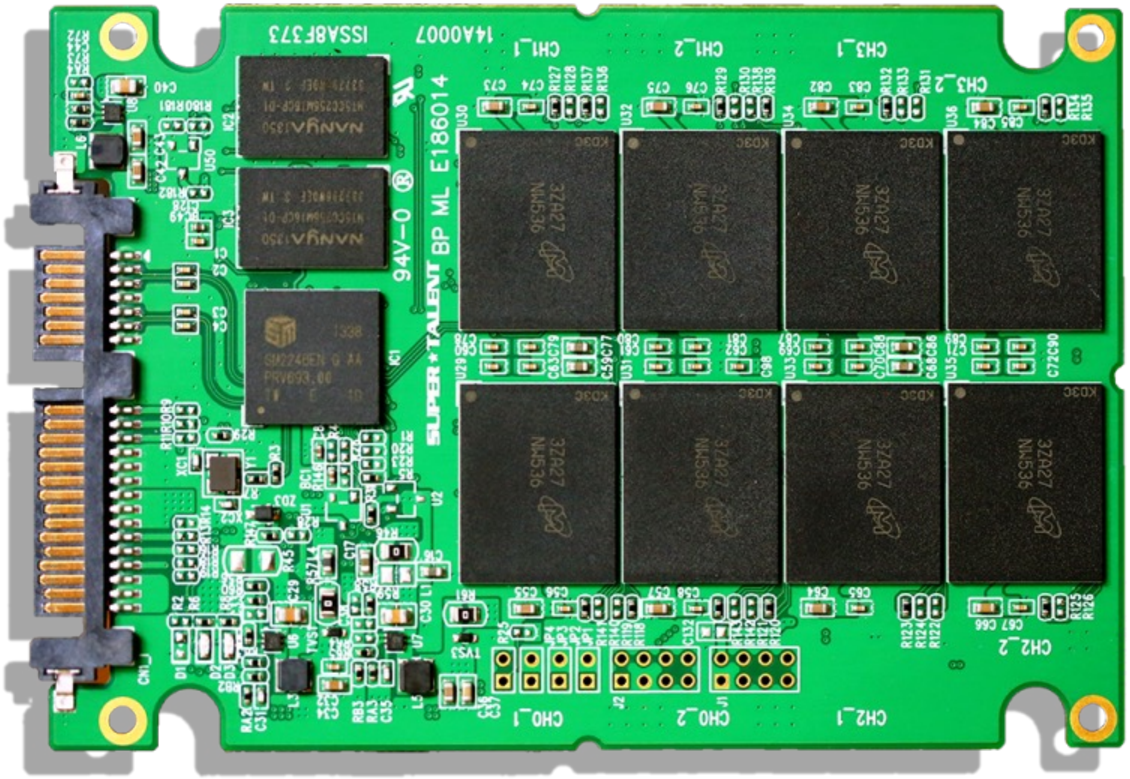
Floppy

CD

HDD

SSD

Solid State Drives



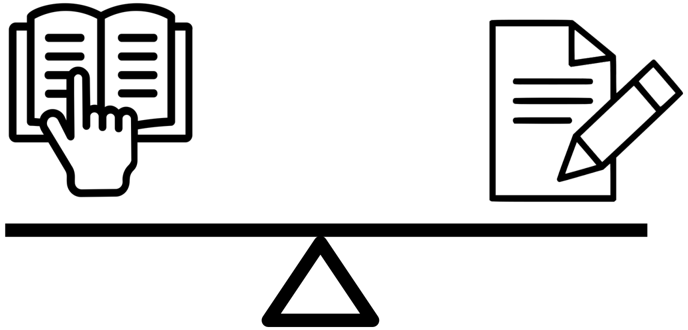
electronic device

fast random access

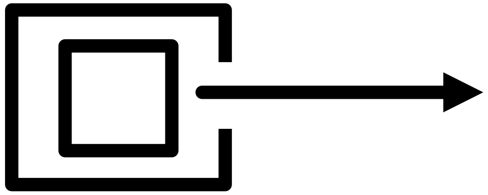
concurrent I/Os

write latency > read latency

HDD

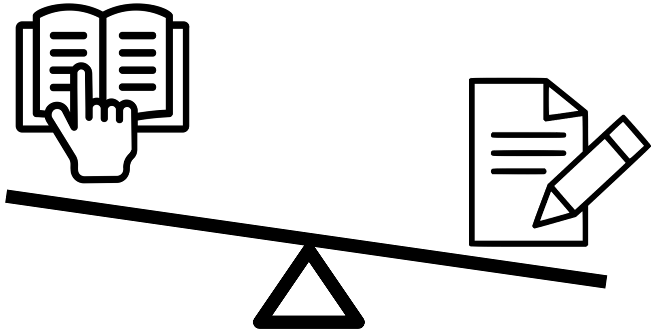


Symmetric cost for Read & Write

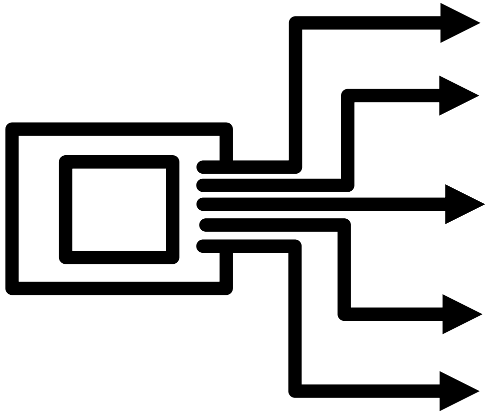


One I/O at a time

SSD

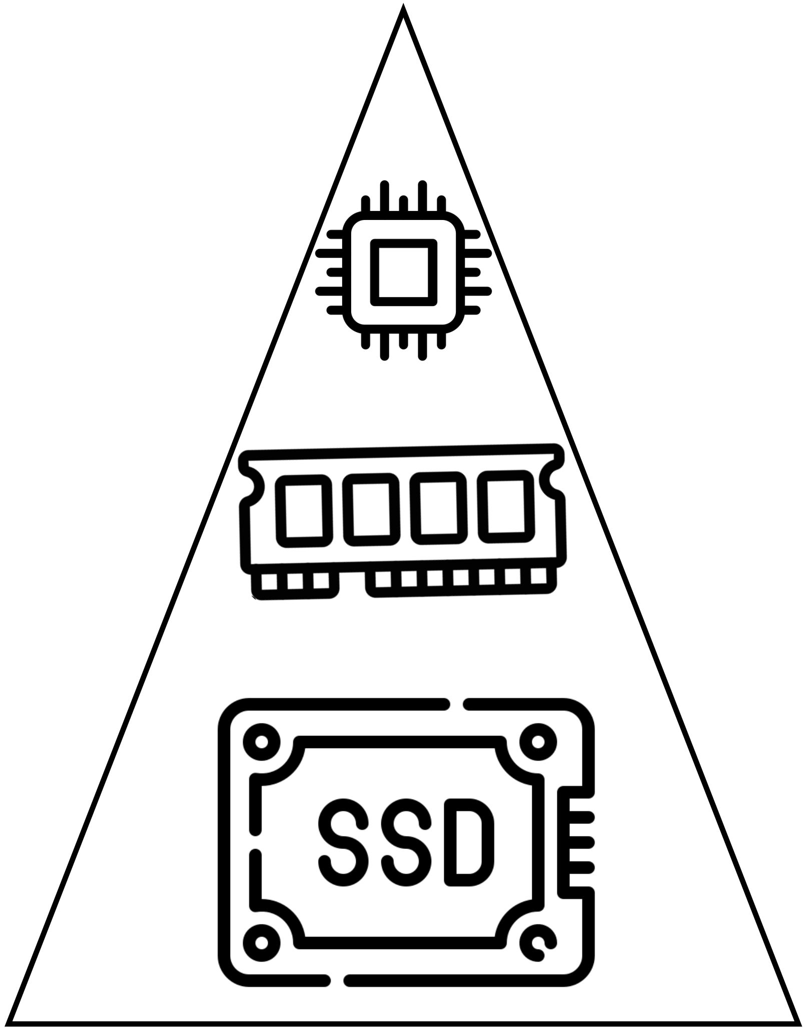


Read/Write Asymmetry (α)



Concurrency (k)

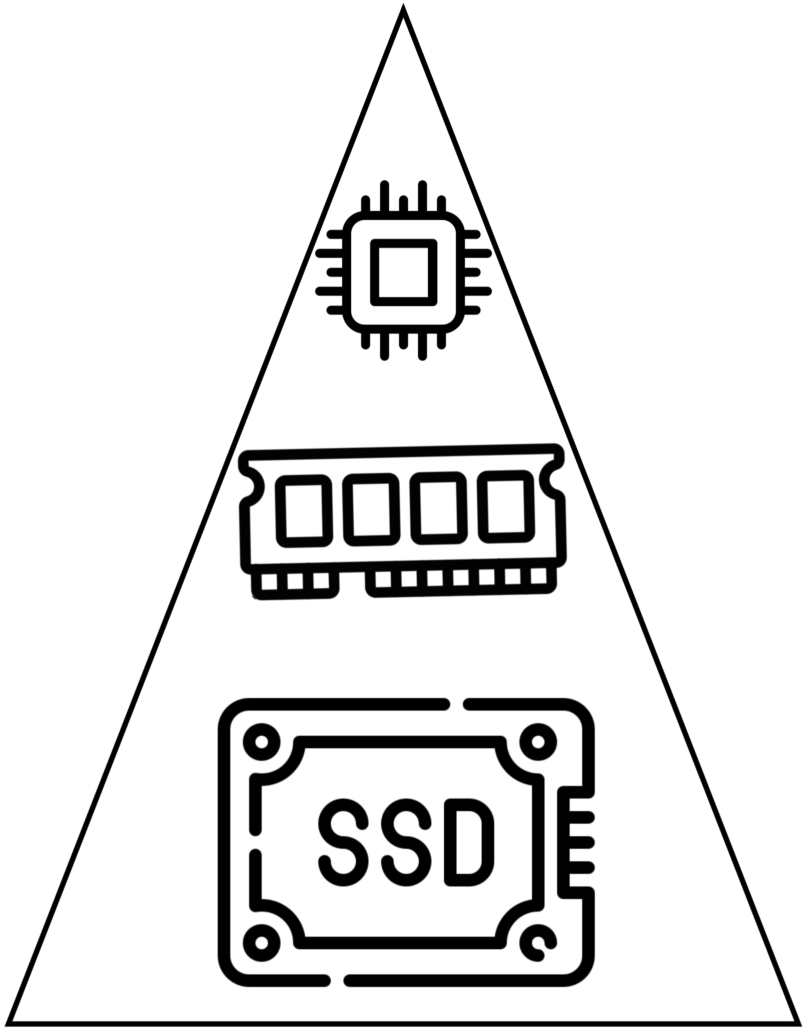
Goal: Developing Hardware-Aware Data Systems



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**]

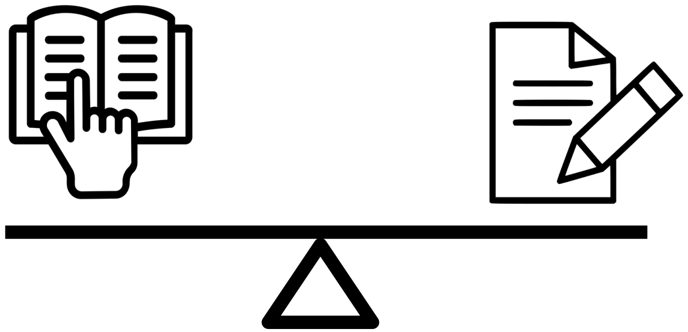
PIO Model [**DaMoN@SIGMOD ’21**]

ACE Bufferpool [**IEEE ICDE ’23**]

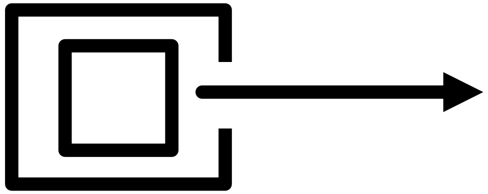
CAVE Graph Engine [**SIGMOD ’24**]

SSD-Aware Systems [**IEEE ICDE ’24**]

HDD

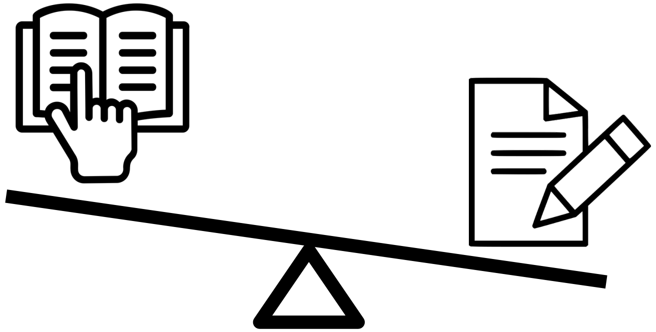


Symmetric cost for Read & Write

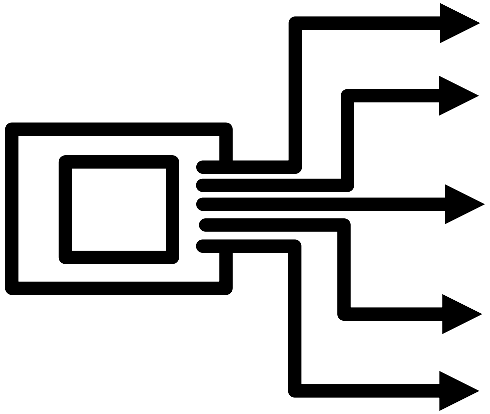


One I/O at a time

SSD

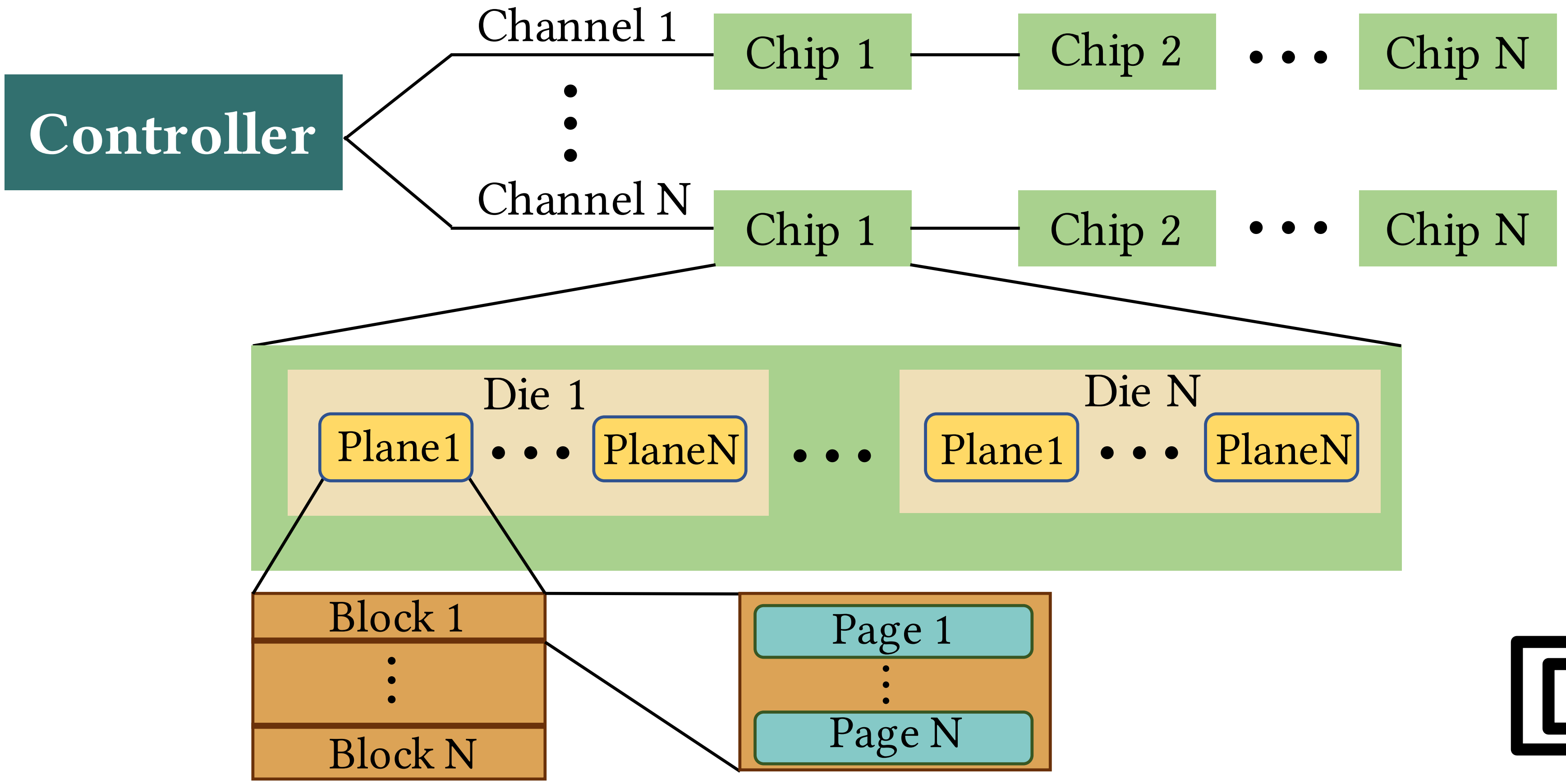


Read/Write Asymmetry (α)



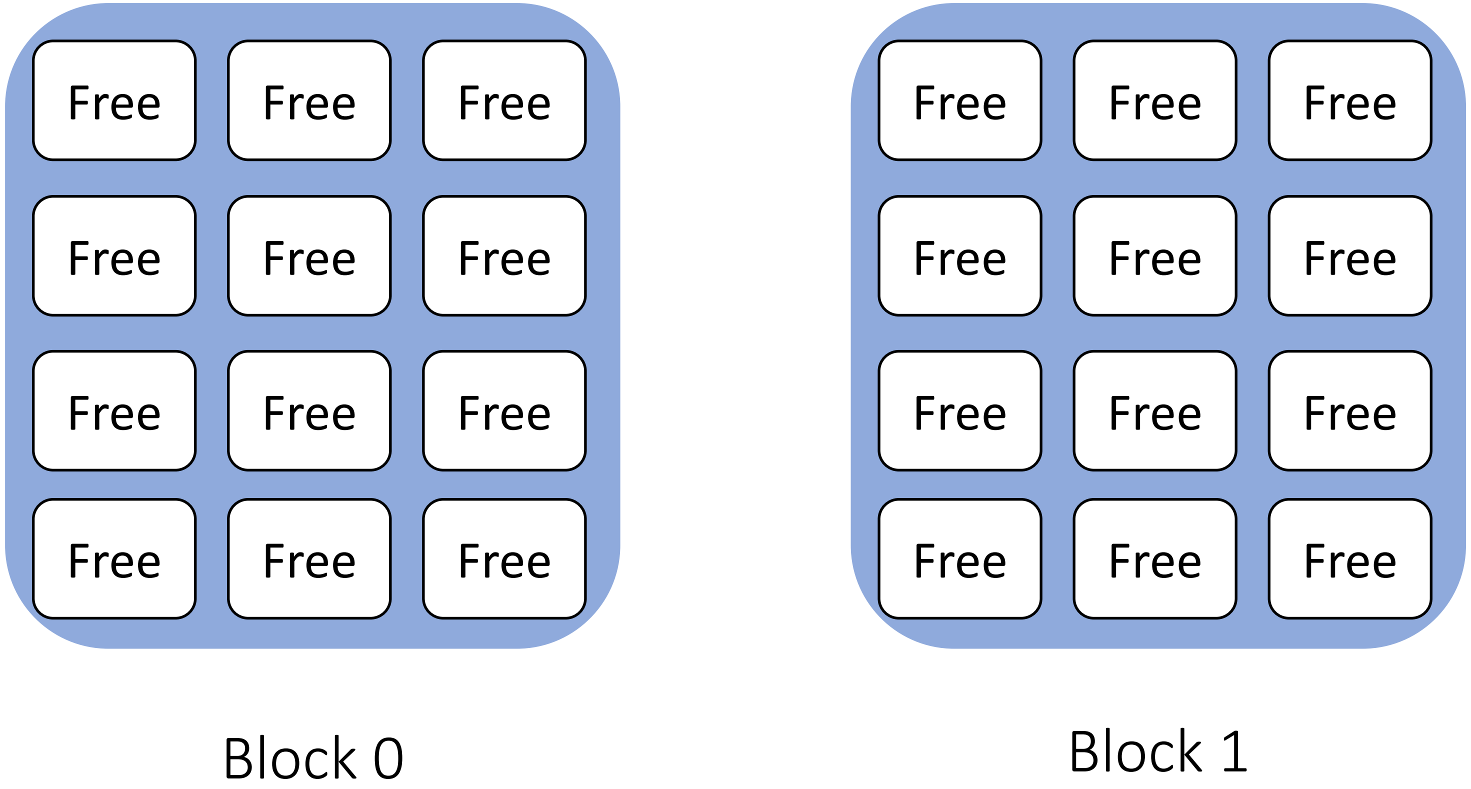
Concurrency (k)

SSD Concurrency



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

Writes in SSD



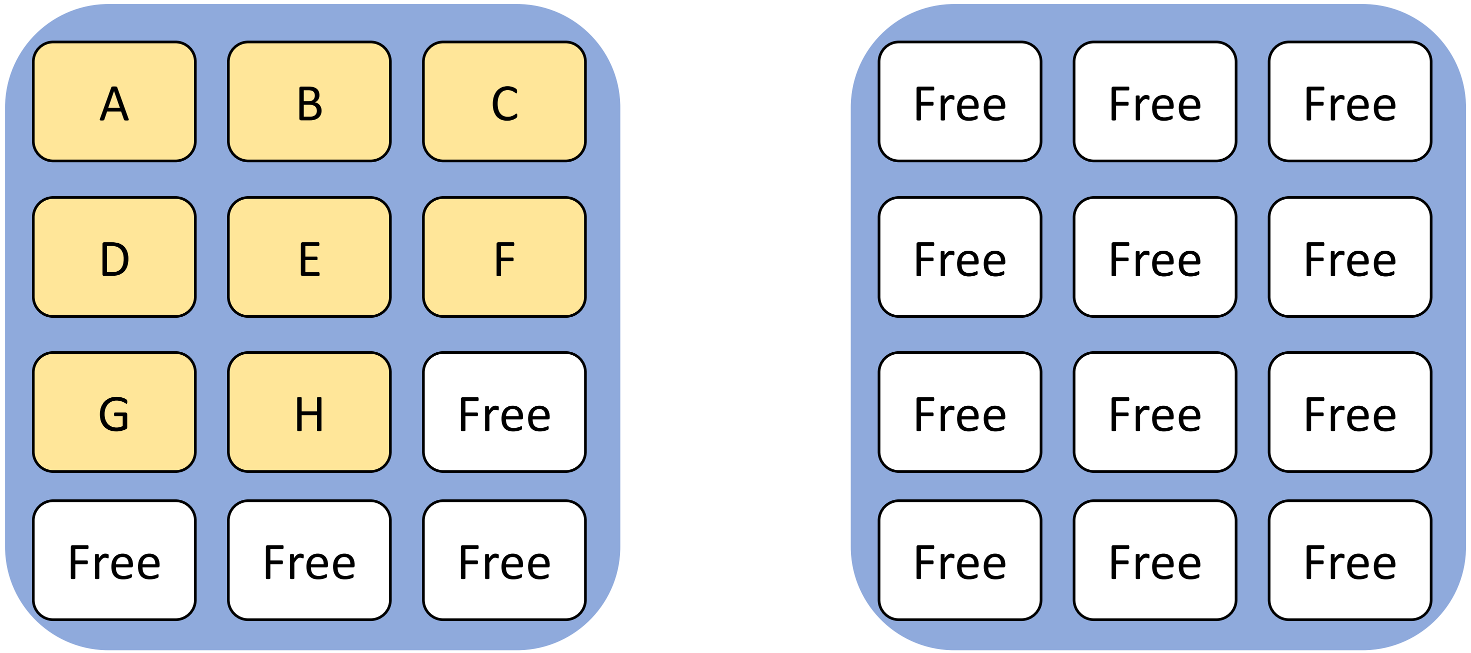
Free Free Free
Free Free Free
Free Free Free
Free Free Free

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

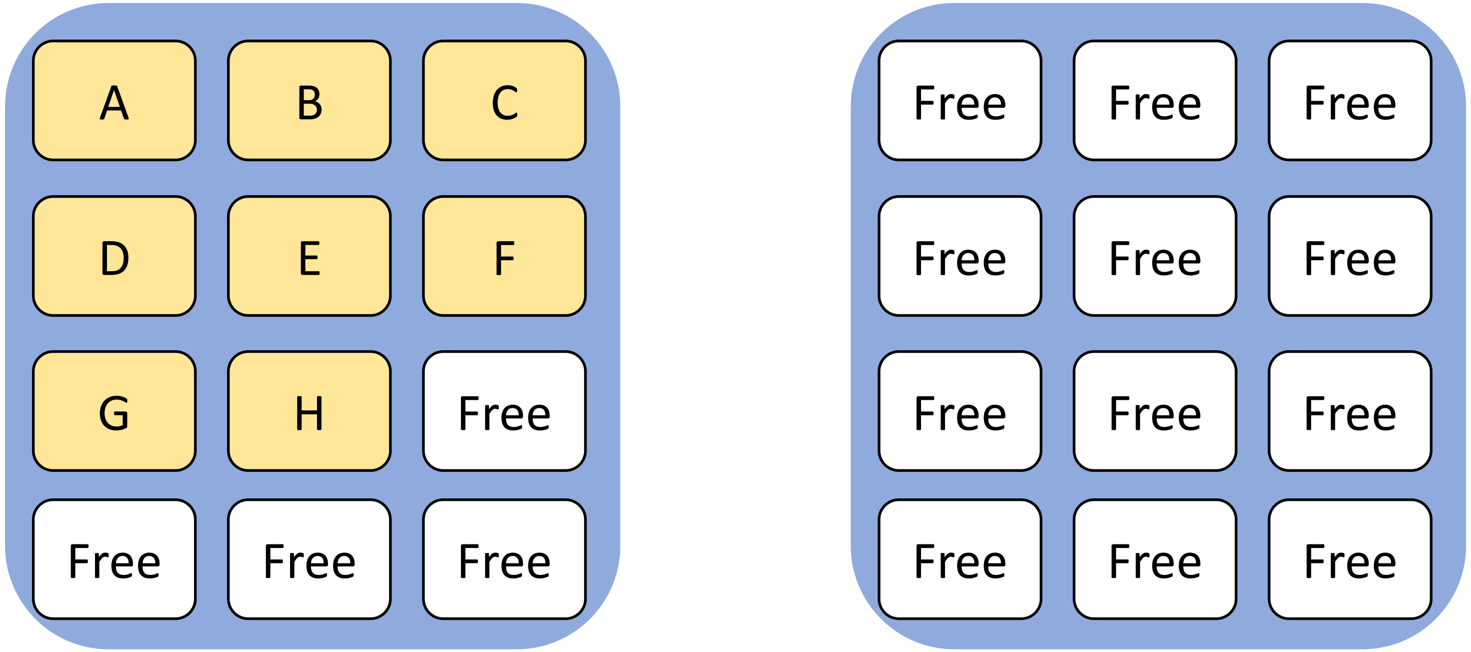
Block 1

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

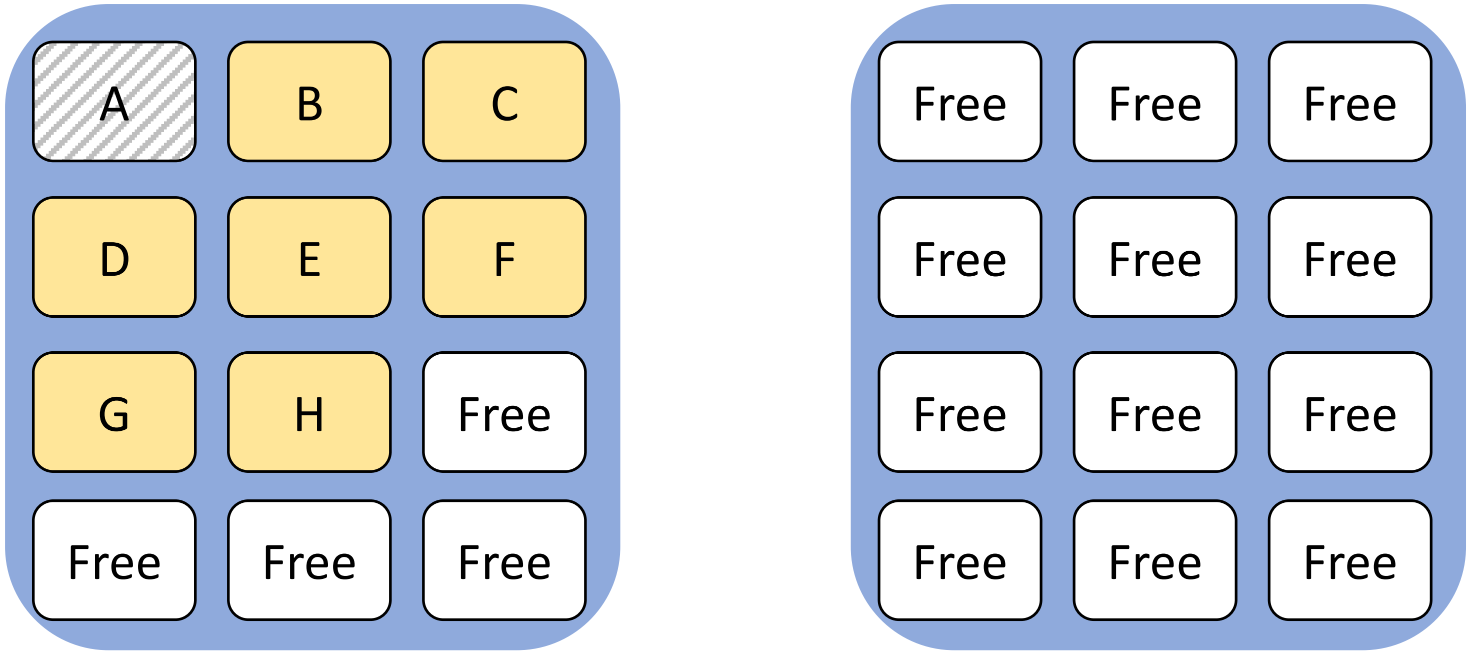
Free	Free	Free
Free	Free	Free
Free	Free	Free
Free	Free	Free

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

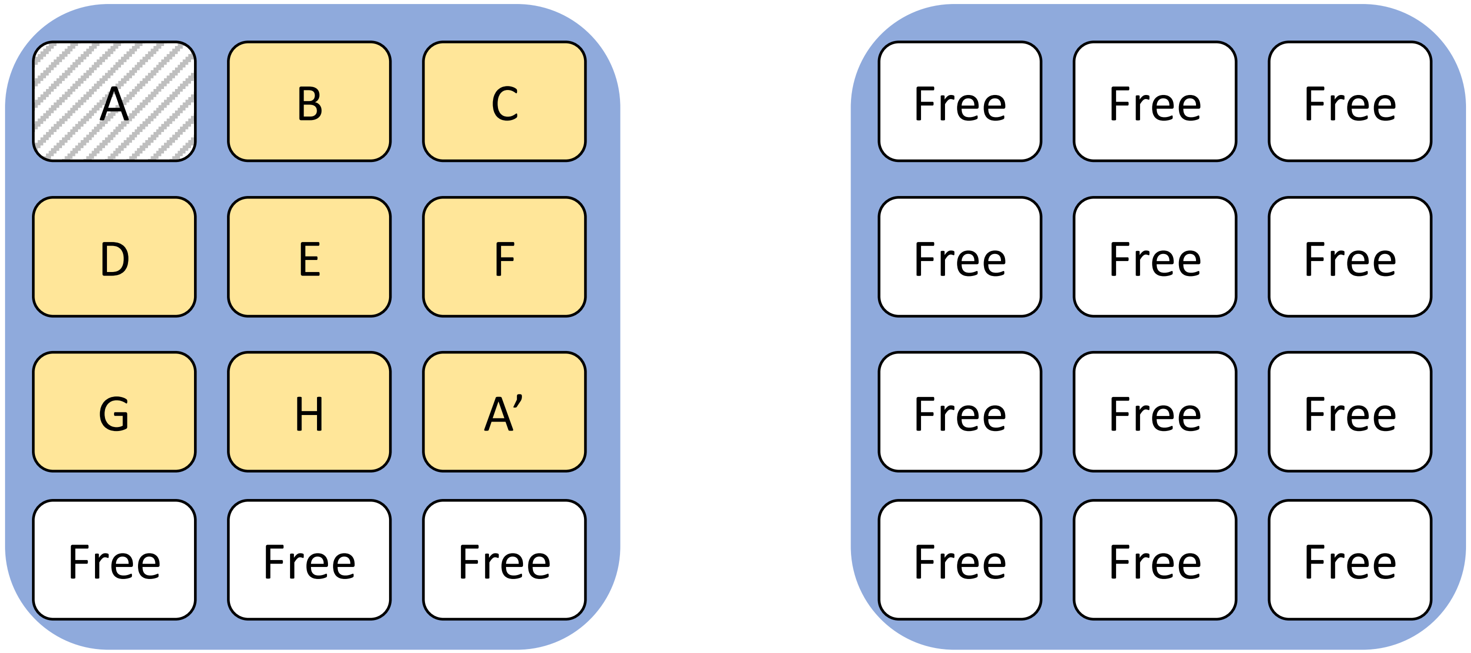
Free Free Free

Block 1

Writes in SSD

Update

A, B, C, D



A	B	C
D	E	F
G	H	A’
Free	Free	Free

Block 0

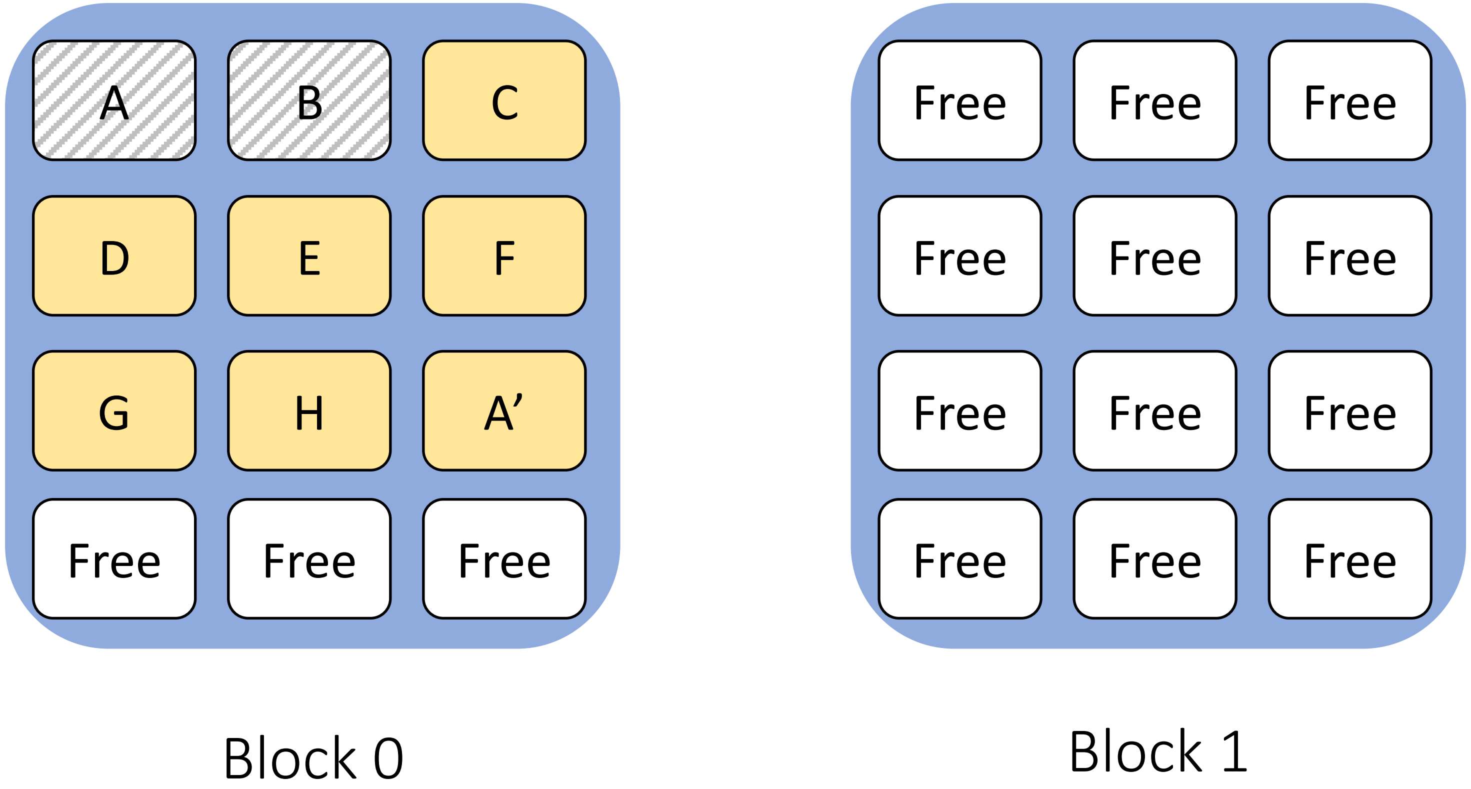
Free	Free	Free
Free	Free	Free
Free	Free	Free
Free	Free	Free

Block 1

Writes in SSD

Update

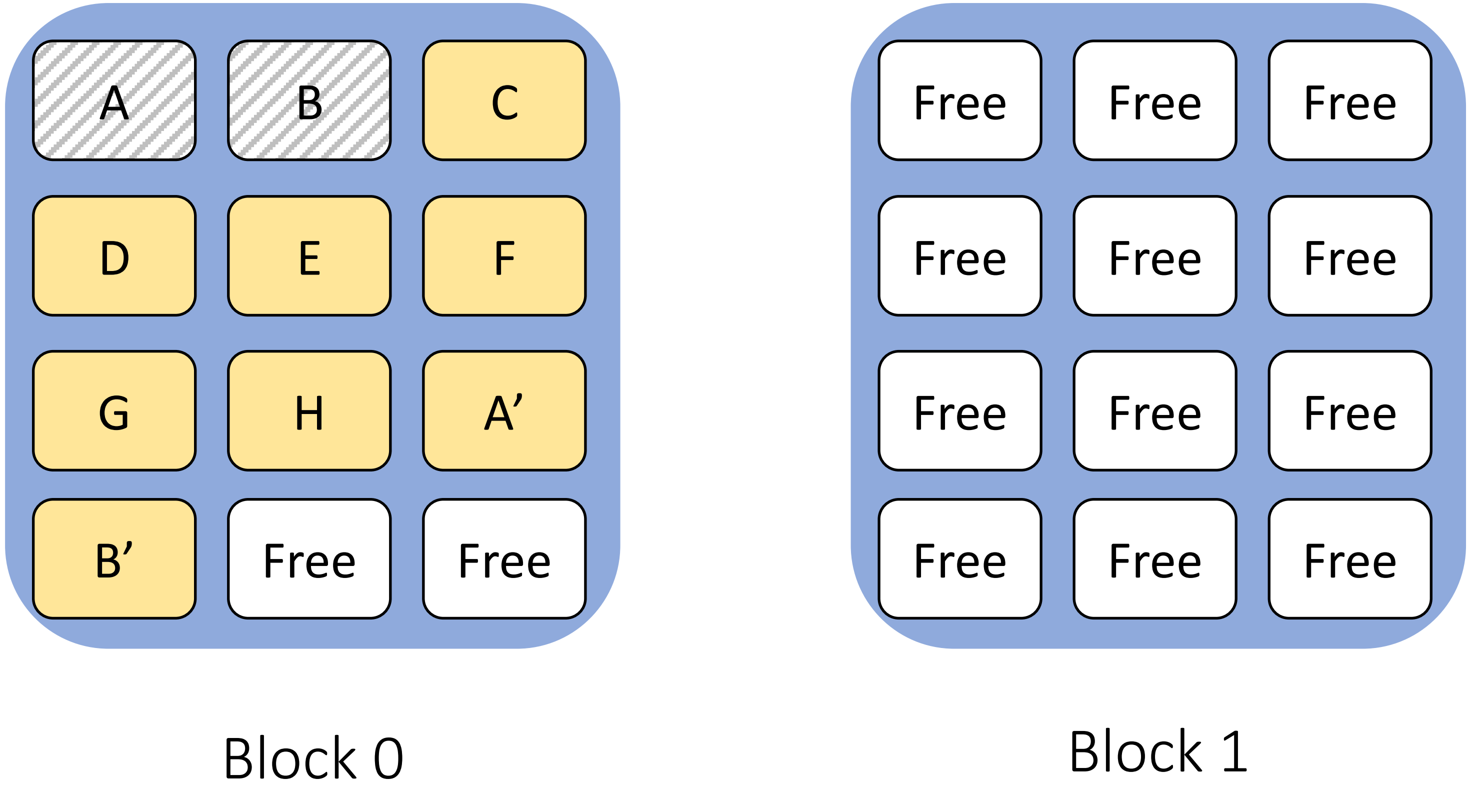
A, **B**, C, D



Writes in SSD

Update

A, **B**, C, D



A	B	C
D	E	F
G	H	A’
B’	Free	Free

Block 0

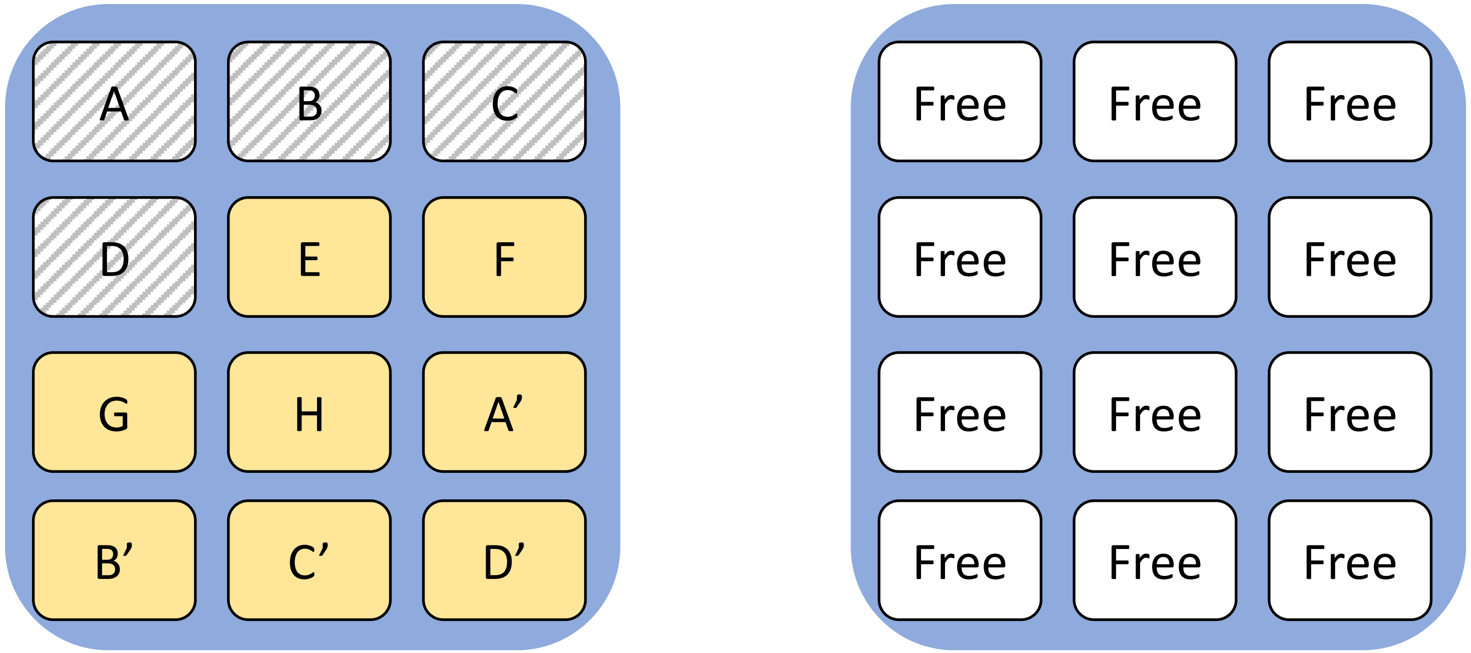
Free	Free	Free
Free	Free	Free
Free	Free	Free
Free	Free	Free

Block 1

Writes in SSD

Update

A, B, C, D



A	B	C
D	E	F
G	H	A’
B’	C’	D’

Block 0

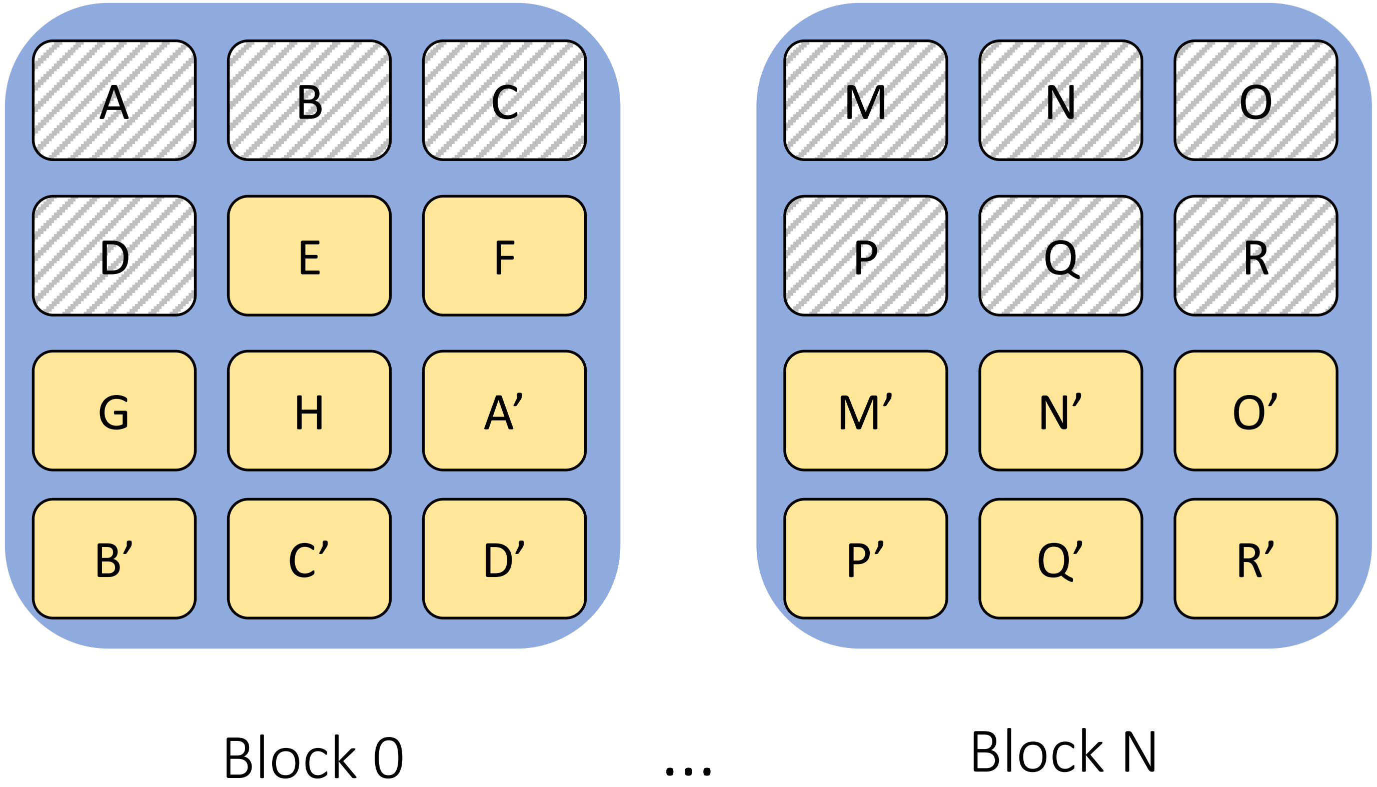
Free	Free	Free
Free	Free	Free
Free	Free	Free
Free	Free	Free

Block 1

Not all updates are costly!

Writes in SSD

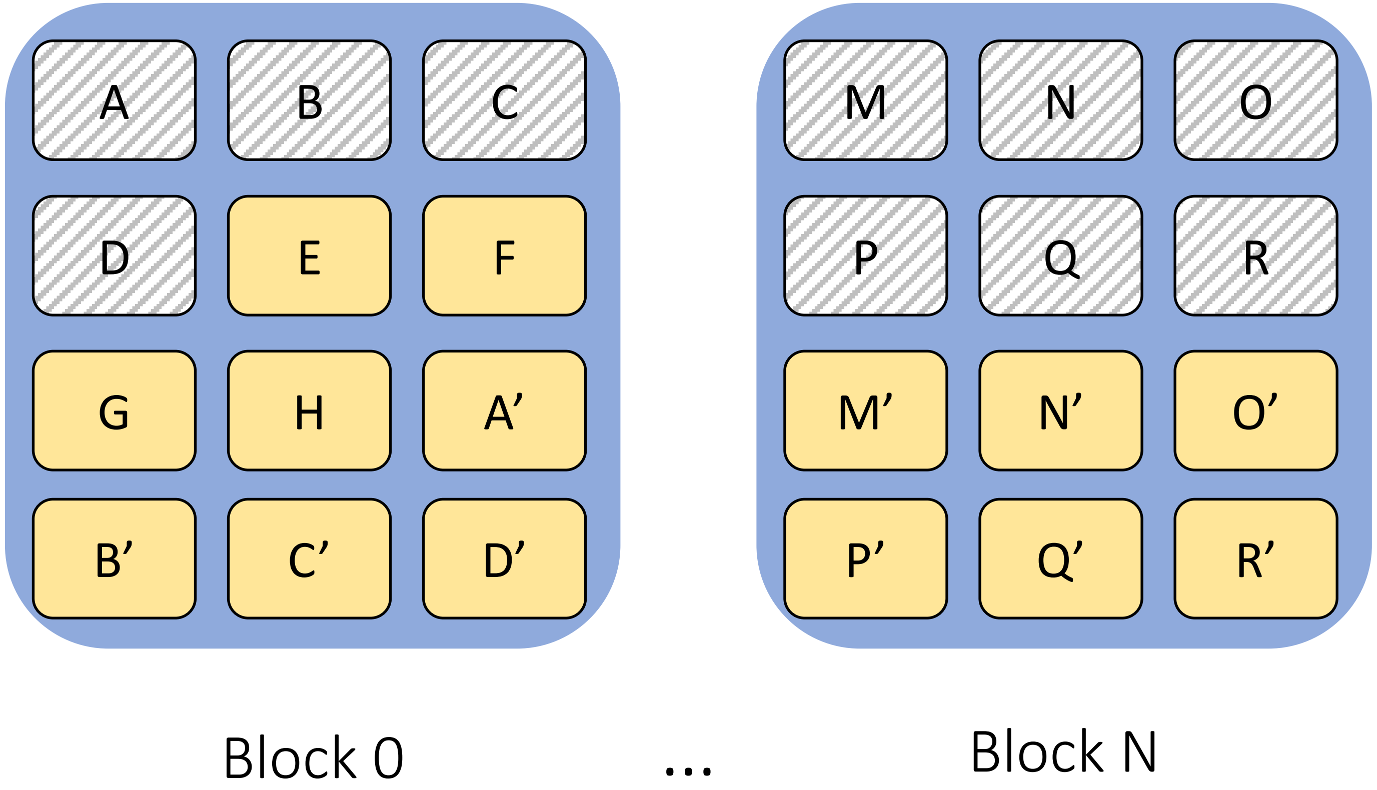
What if there is no space?



Writes in SSD

What if there is no space?

Garbage Collection!



A B C
D E F
G H A’
B’ C’ D’

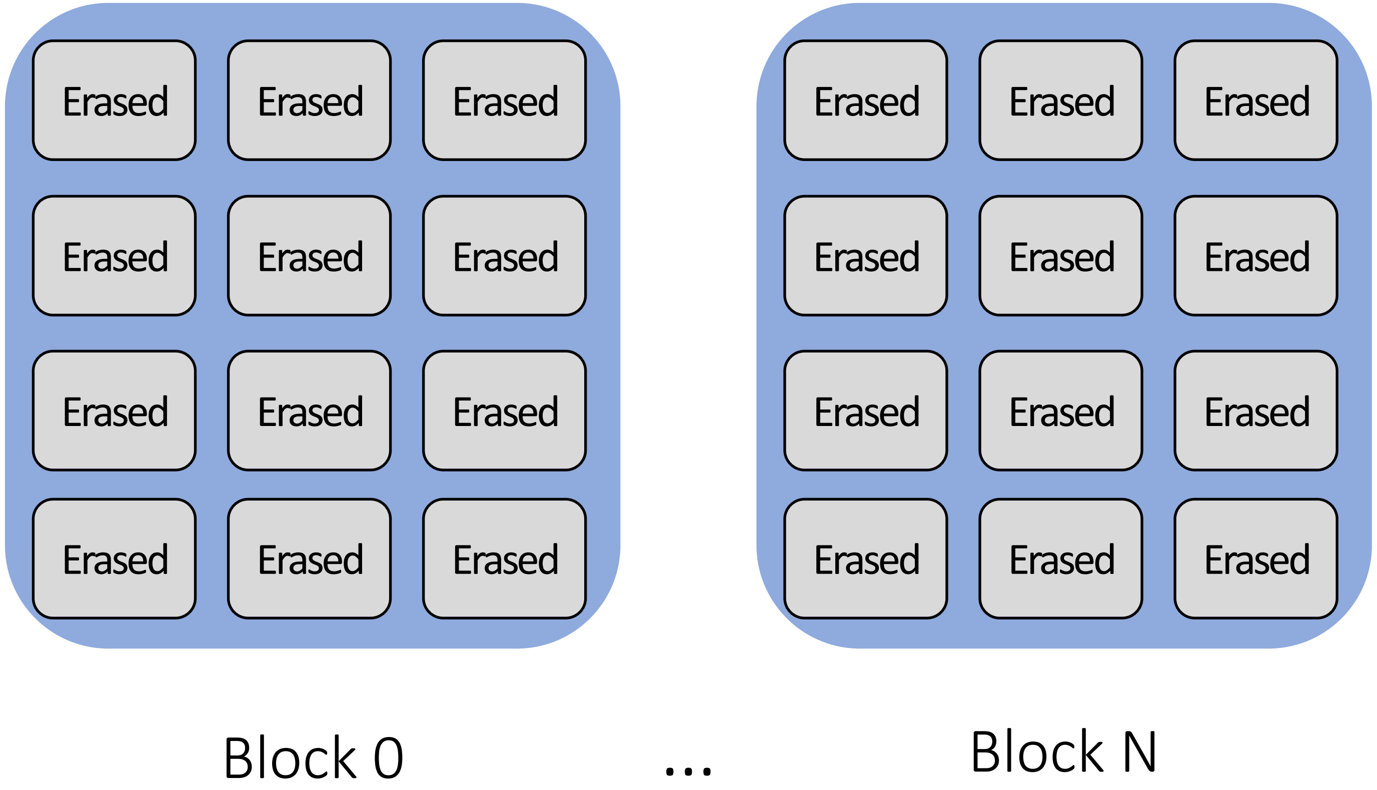
M N O
P Q R
M’ N’ O’
P’ Q’ R’

Block 0 … Block N

Writes in SSD

What if there is no space?

Garbage Collection!



Erased Erased Erased

Erased Erased Erased

Erased Erased Erased

Erased Erased Erased

Block 0

...

Erased Erased Erased

Erased Erased Erased

Erased Erased Erased

Erased Erased Erased

Block N

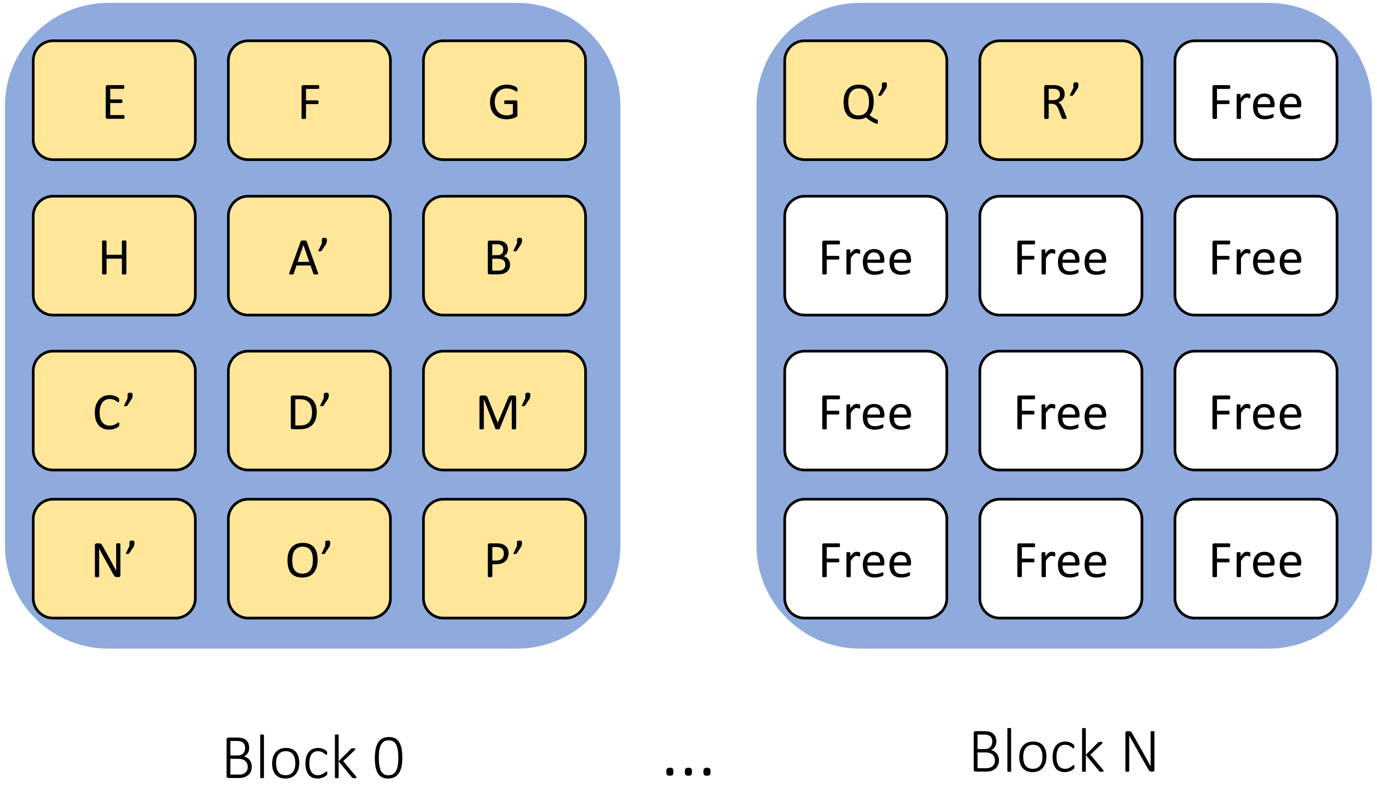
Valid pages:

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

...

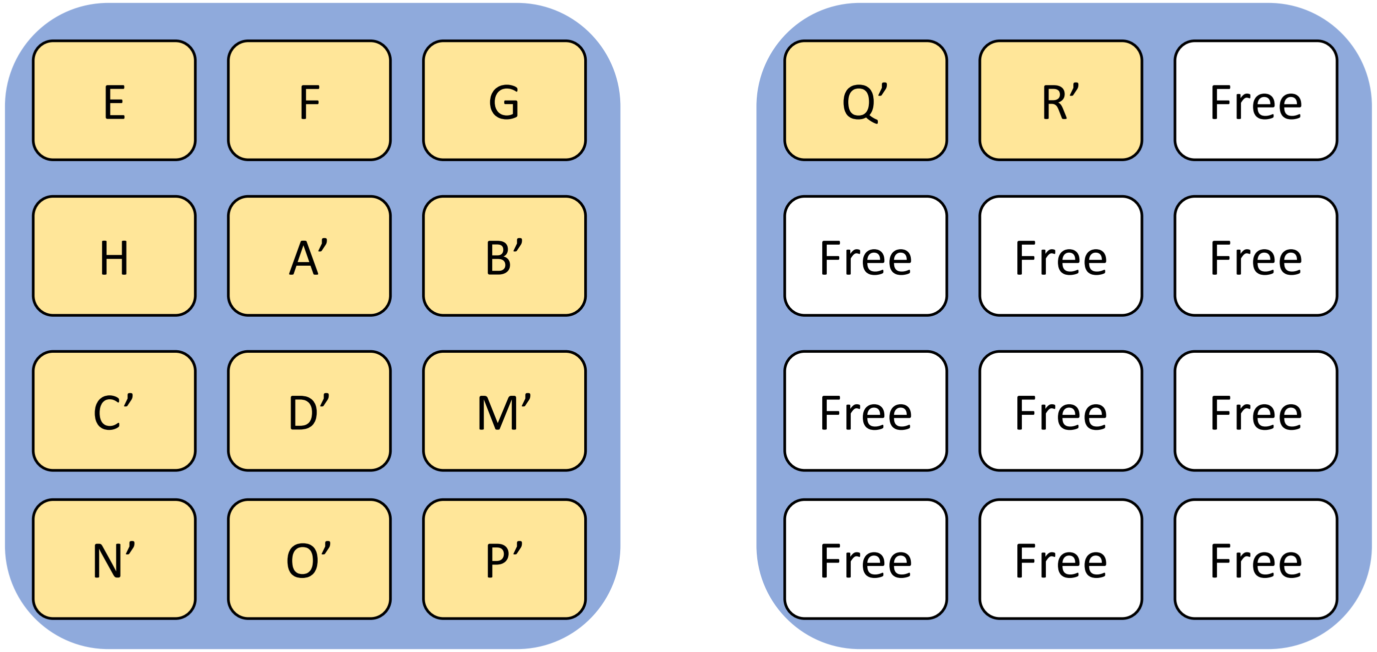
Q’	R’	Free
Free	Free	Free
Free	Free	Free
Free	Free	Free

Block N

Read/Write Asymmetry 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

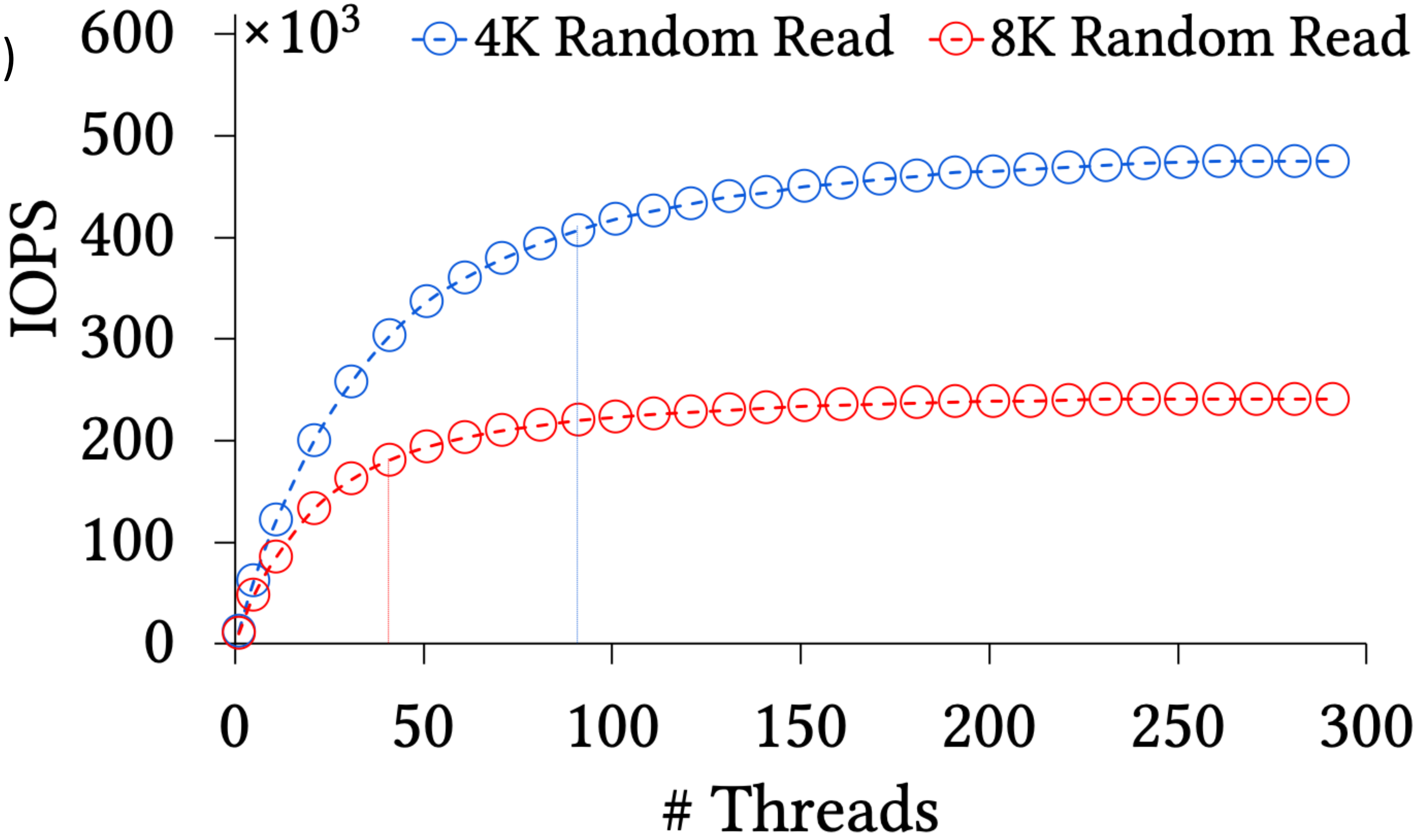
Block N

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

Quantifying Asymmetry & Concurrency

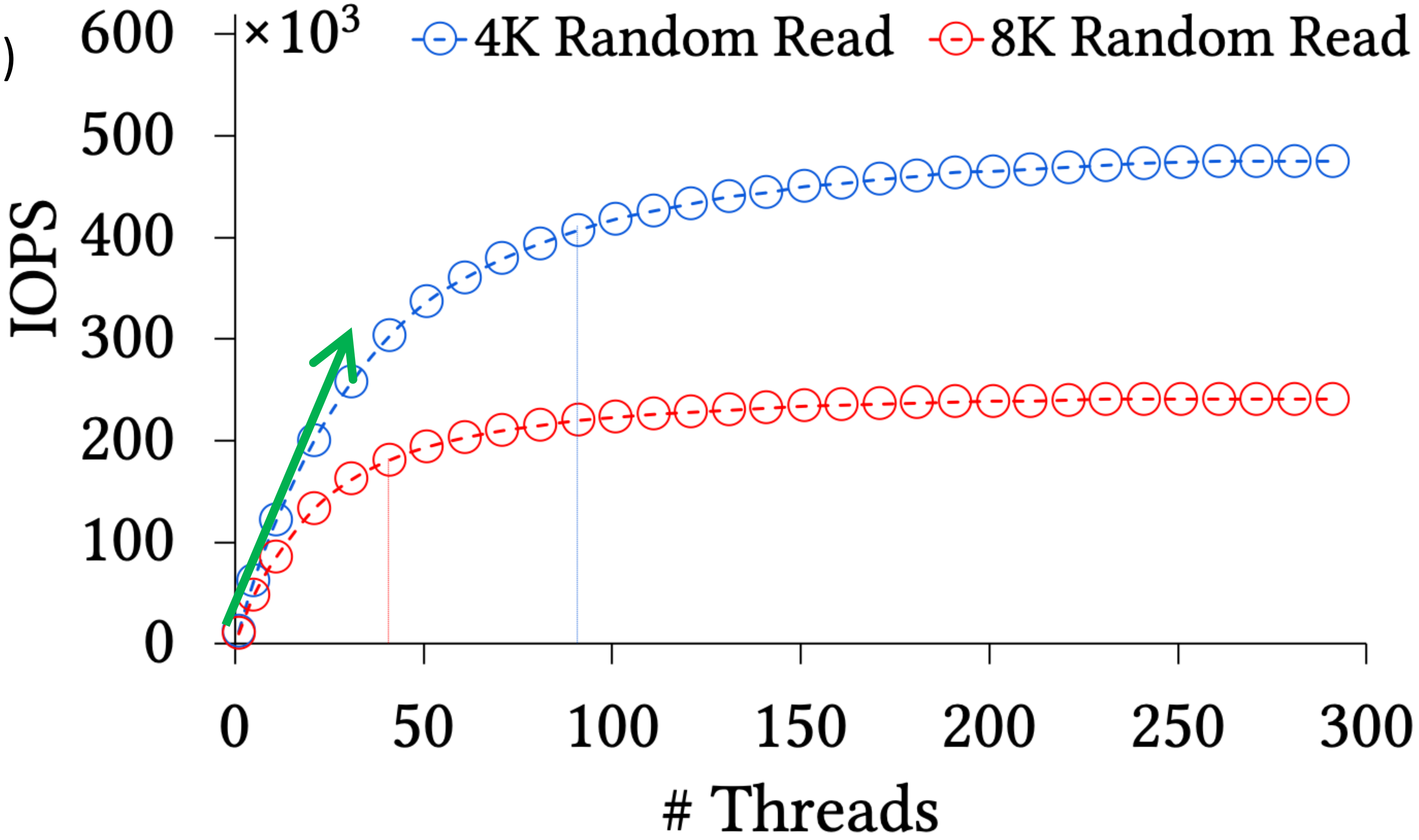
Device

PCIe SSD - P4510 (1TB)



Quantifying Asymmetry & Concurrency

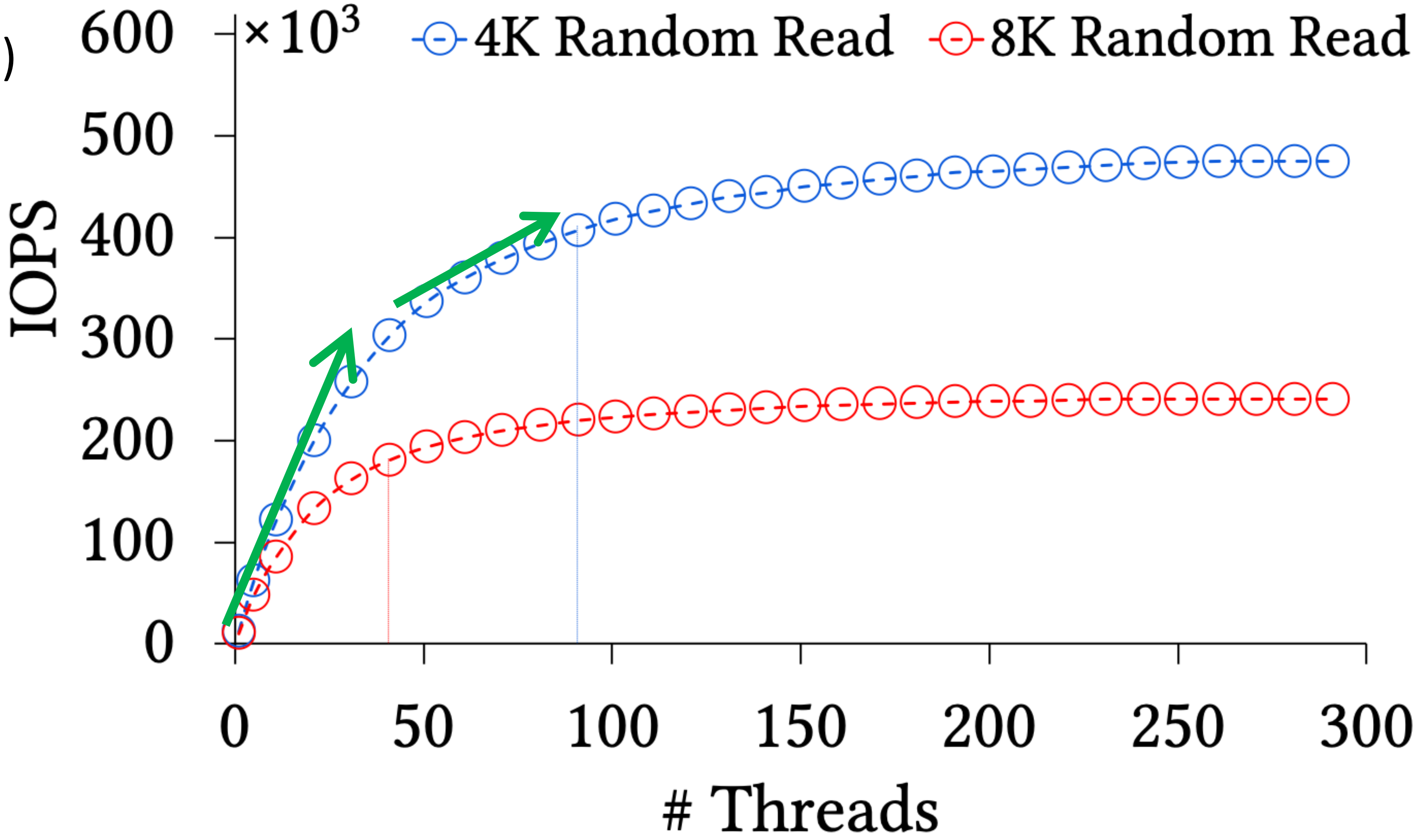
Device
PCIe SSD - P4510 (1TB)



Quantifying Asymmetry & Concurrency

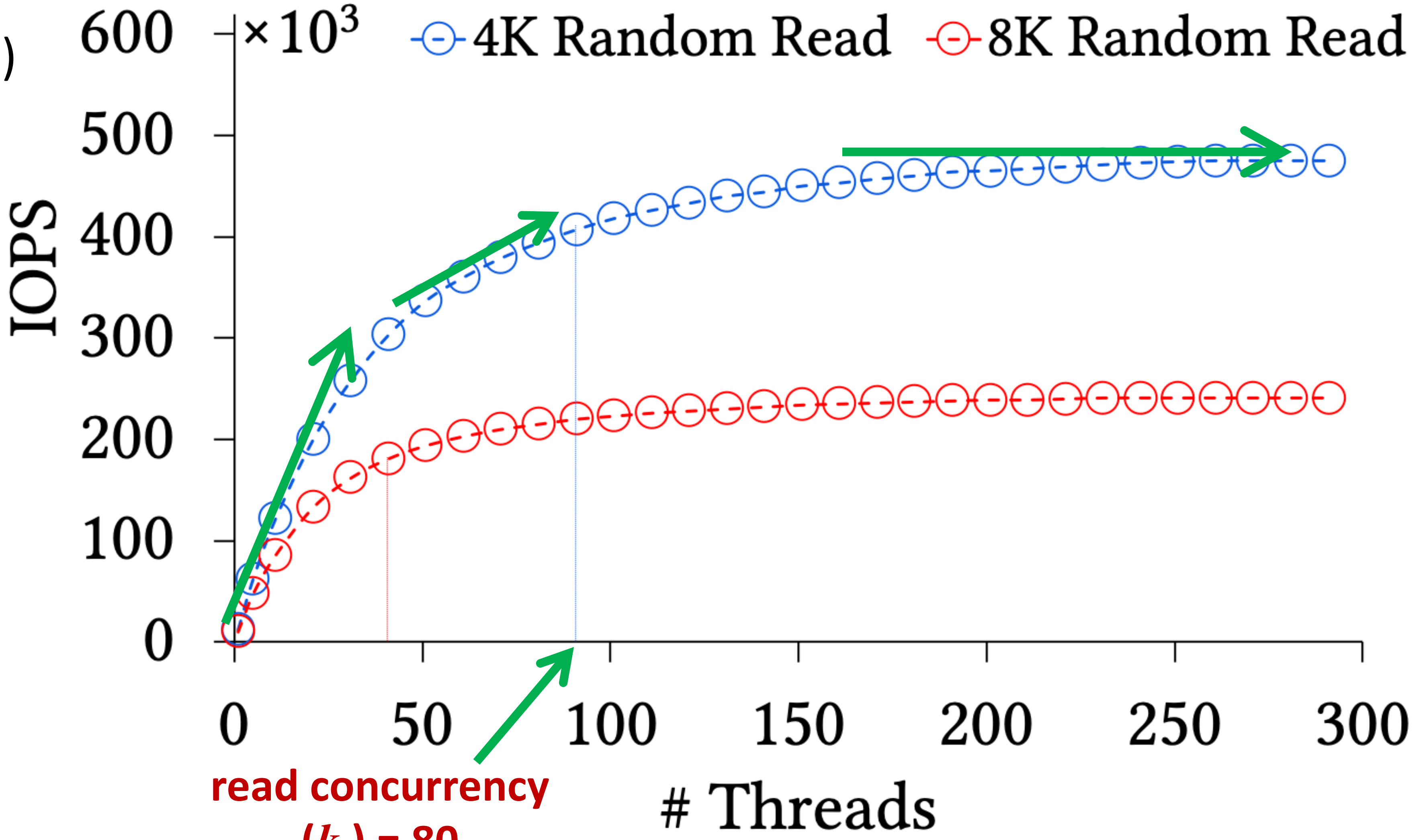
Device

PCIe SSD - P4510 (1TB)



Quantifying Asymmetry & Concurrency

Device
PCIe SSD - P4510 (1TB)

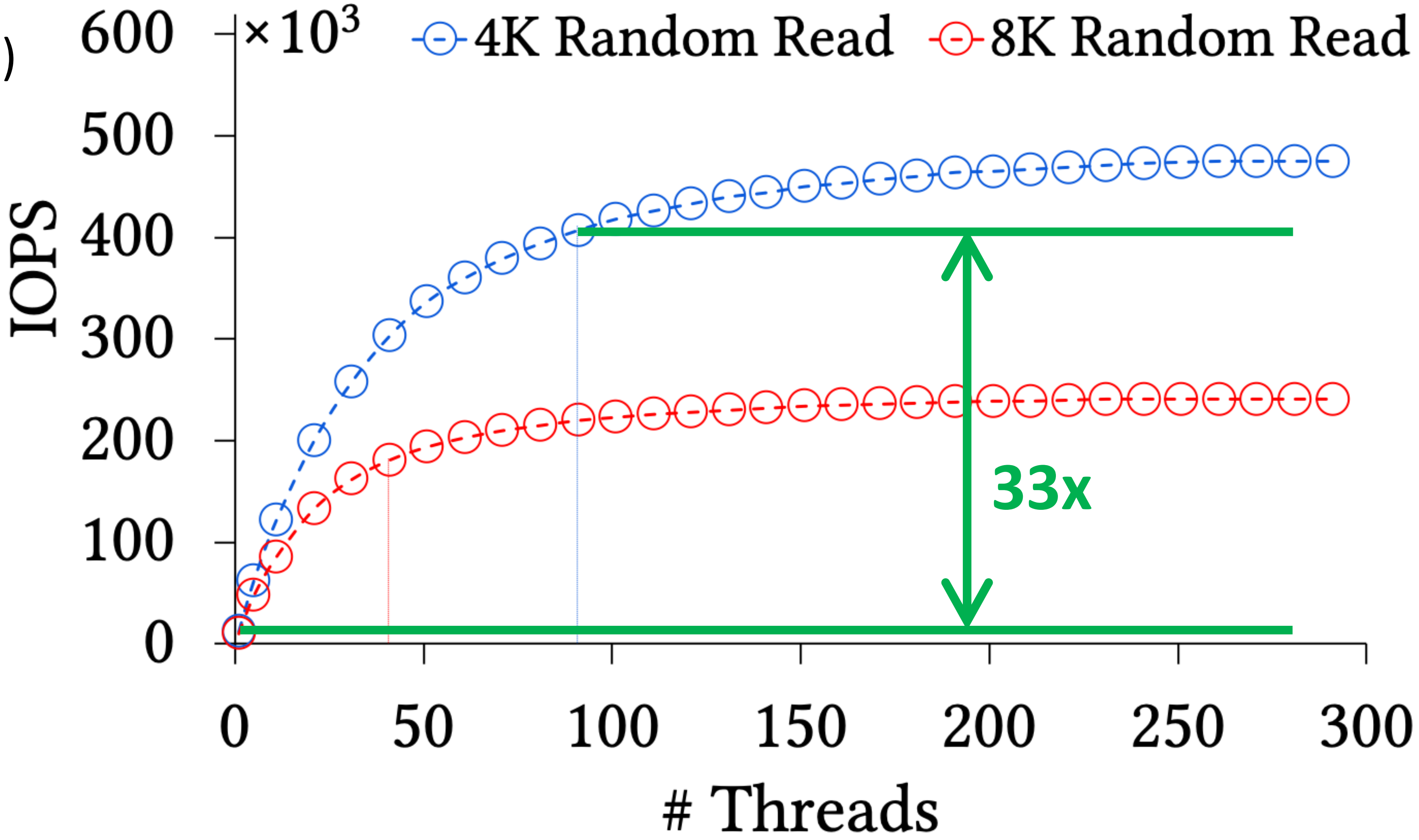


read concurrency (k_r) = 80

Quantifying Asymmetry & Concurrency

Device

PCIe SSD - P4510 (1TB)



Quantifying Asymmetry & Concurrency

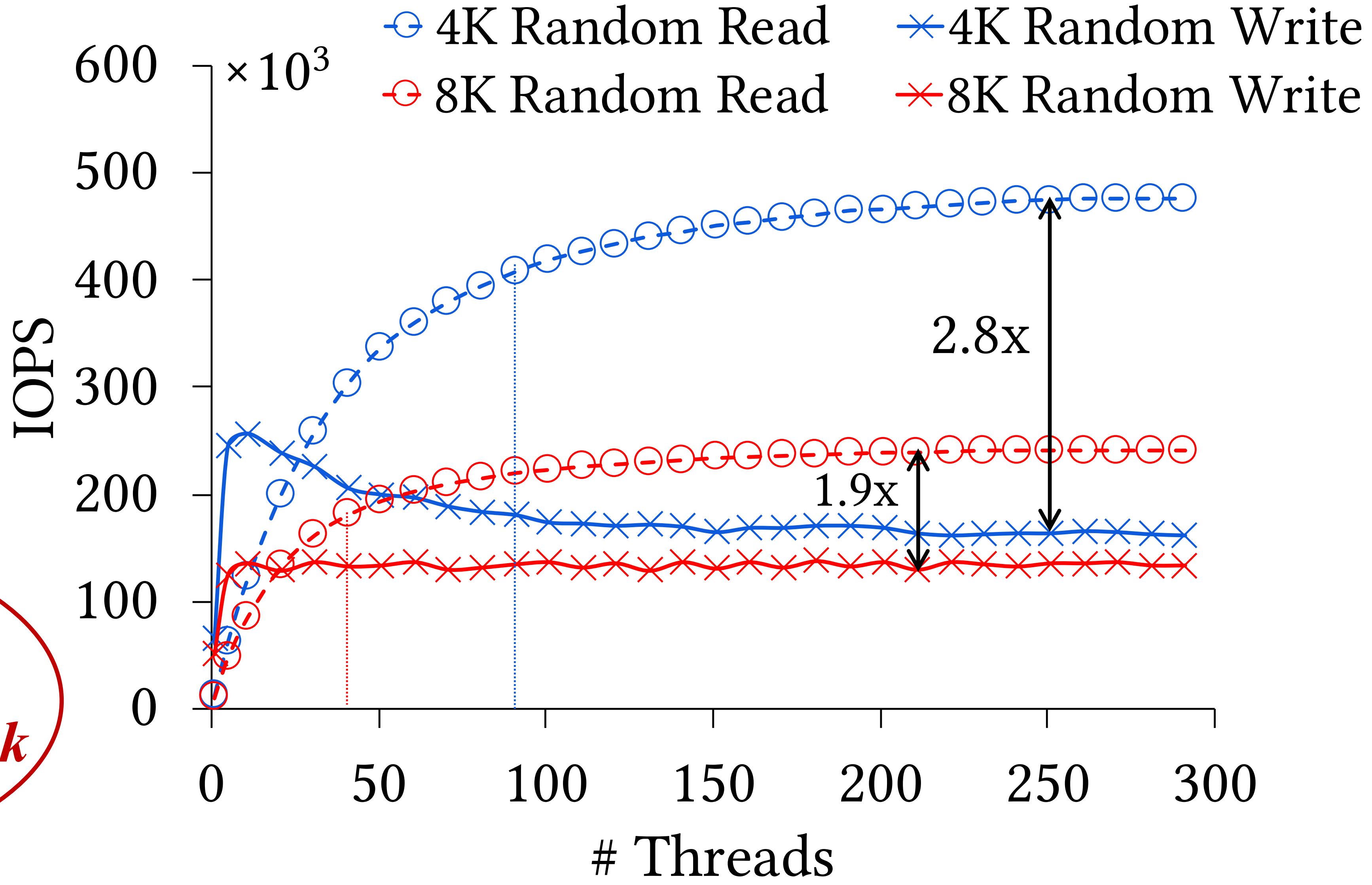
Device
PCIe SSD - P4510 (1TB)

For 4K random read,

Asymmetry: 2.8

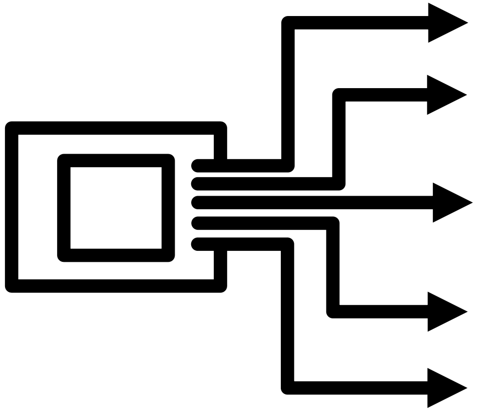
Concurrency: 80

Yet, systems are not always tailored for α/k



Guidelines for System Design in SSDs

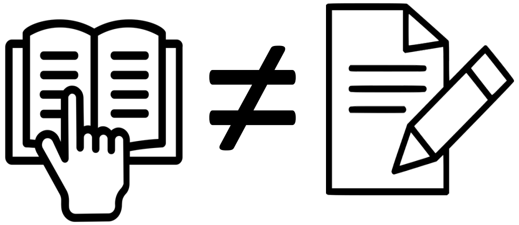
know thy device



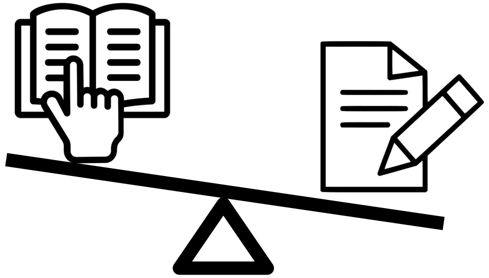
exploit k_r and k_w (with care)

read concurrency

write concurrency

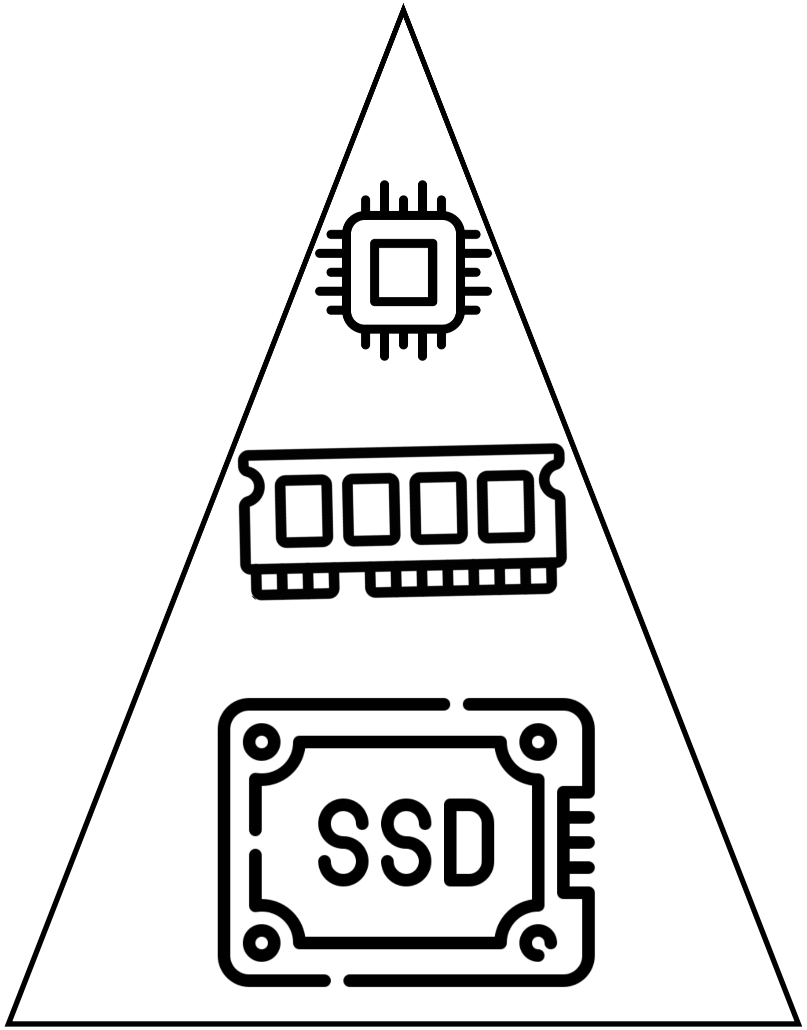


treat read and write differently



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**]

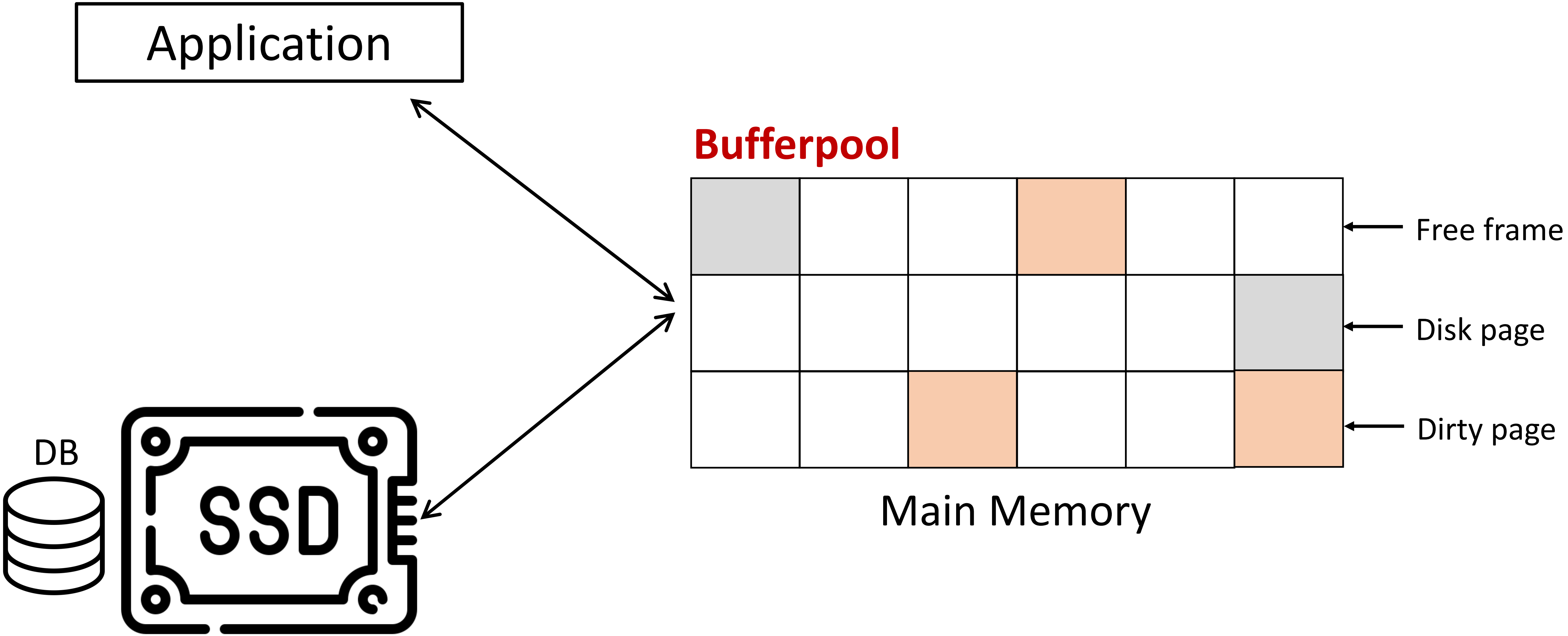
PIO Model [**DaMoN@SIGMOD ’21**]

ACE Bufferpool [IEEE ICDE ’23]

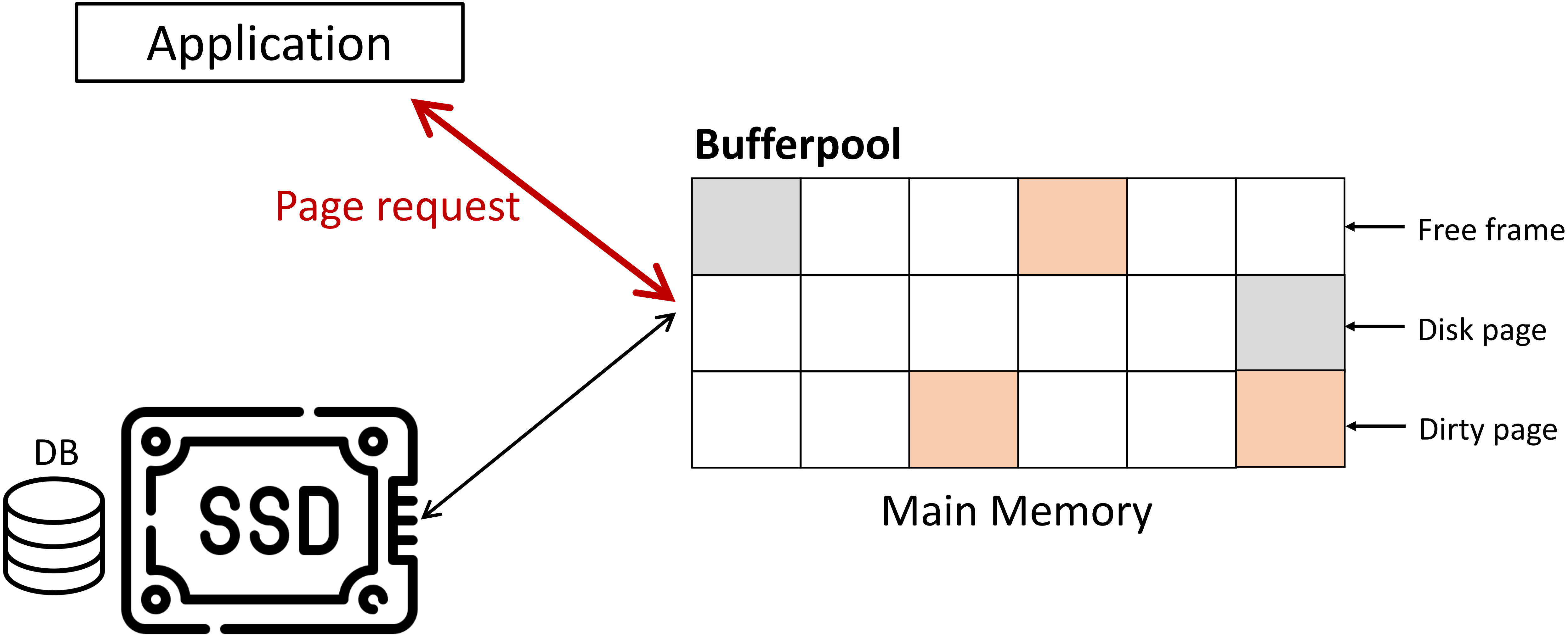
CAVE Graph Engine [**SIGMOD ’24**]

SSD-Aware Systems [**IEEE ICDE ’24**]

Bufferpool is Tightly Connected to Storage



Bufferpool Manager



Application

Page request

Bufferpool

Free frame

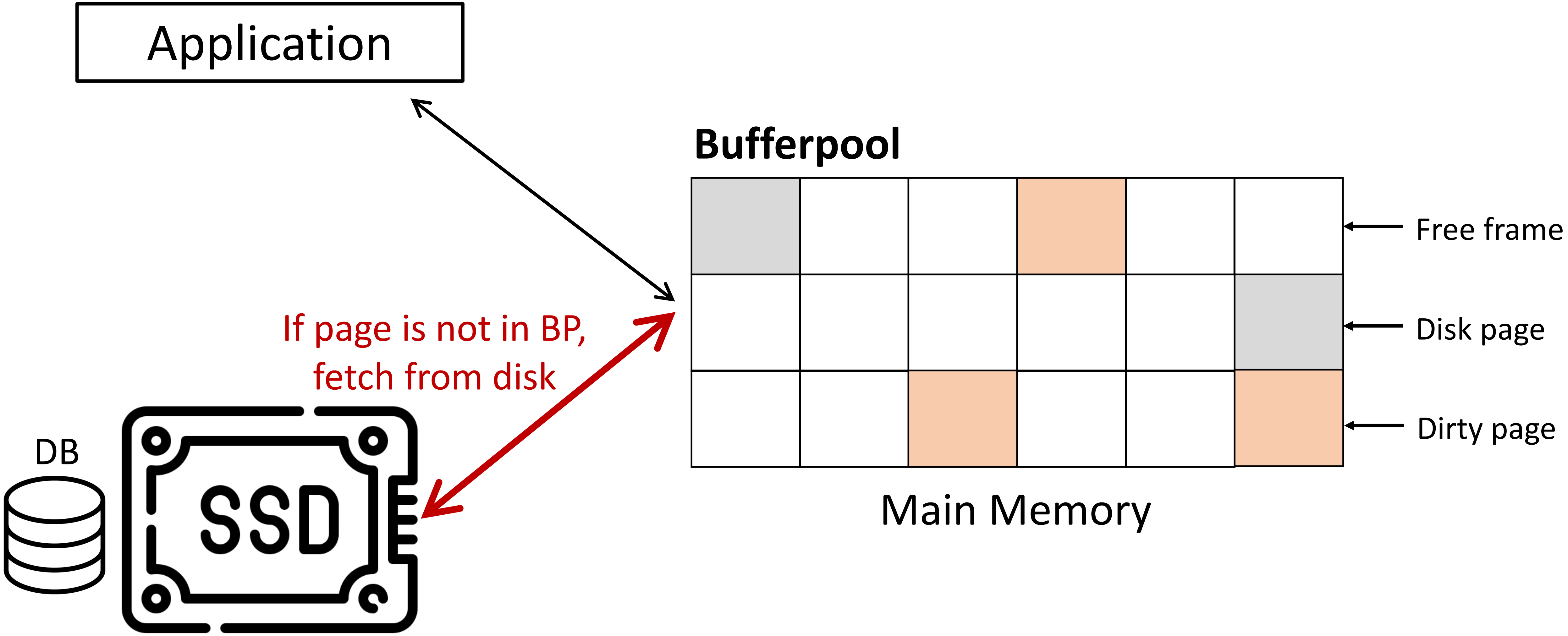
Disk page

Dirty page

Main Memory

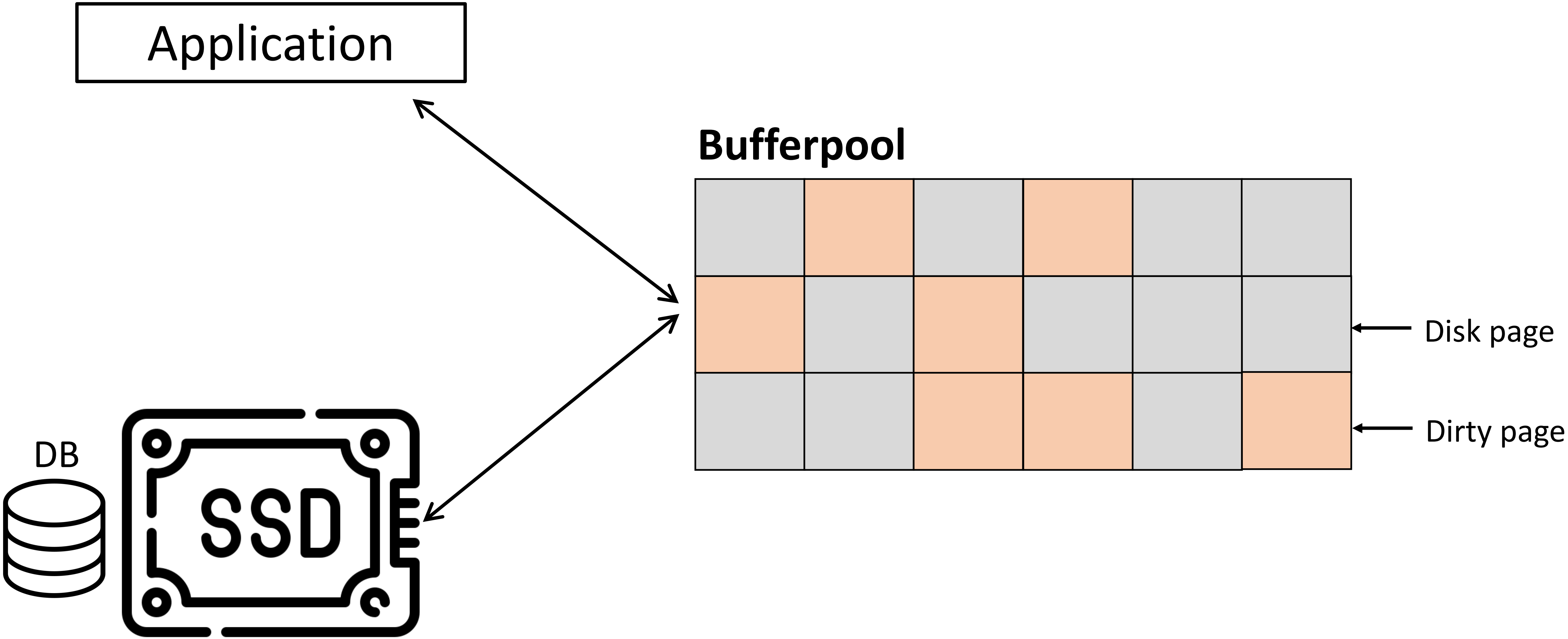
DB

Bufferpool Manager

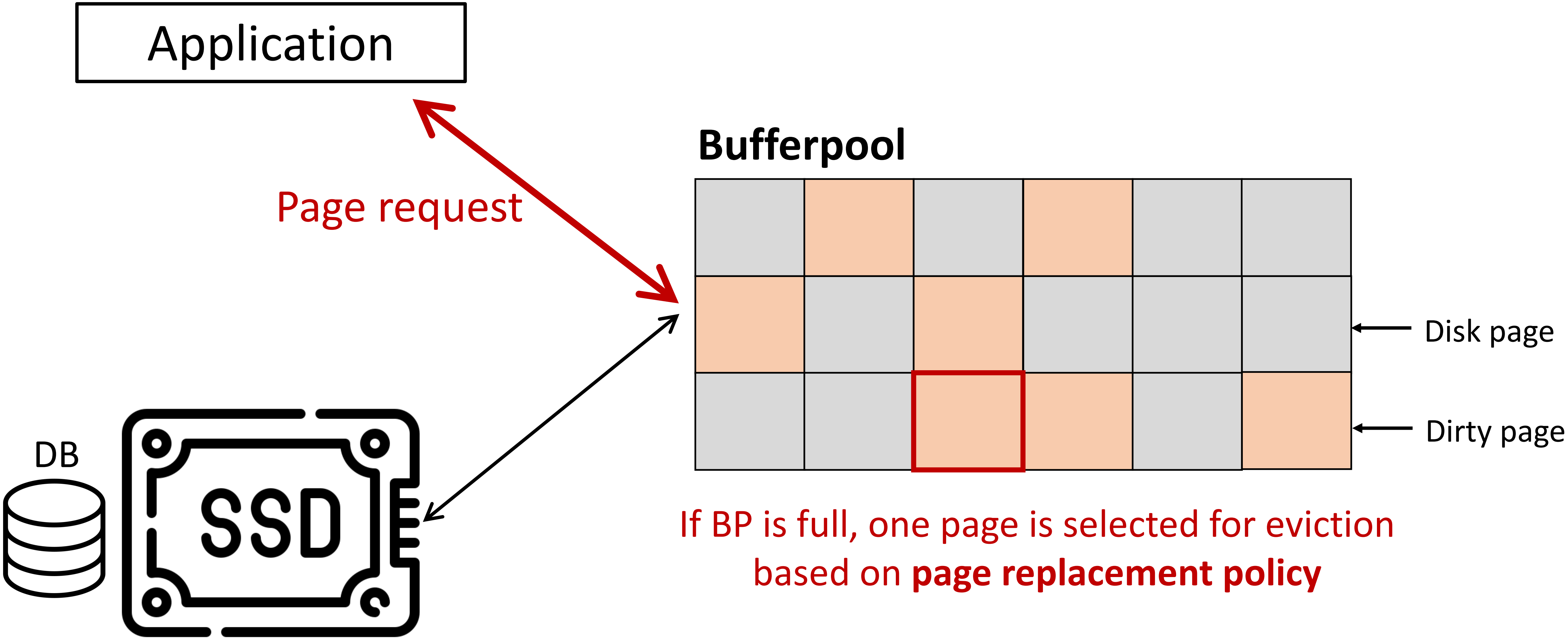


If page is not in BP, fetch from disk

Bufferpool Manager

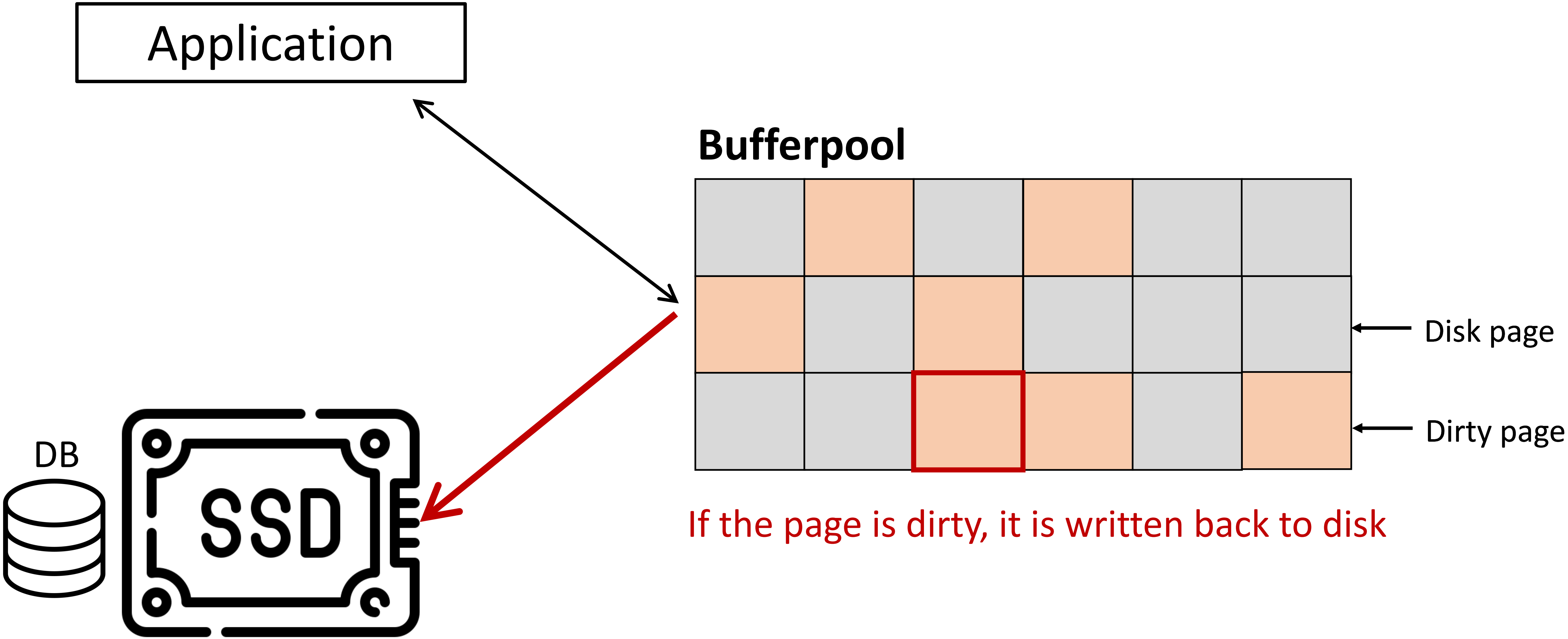


Traditional Bufferpool Manager



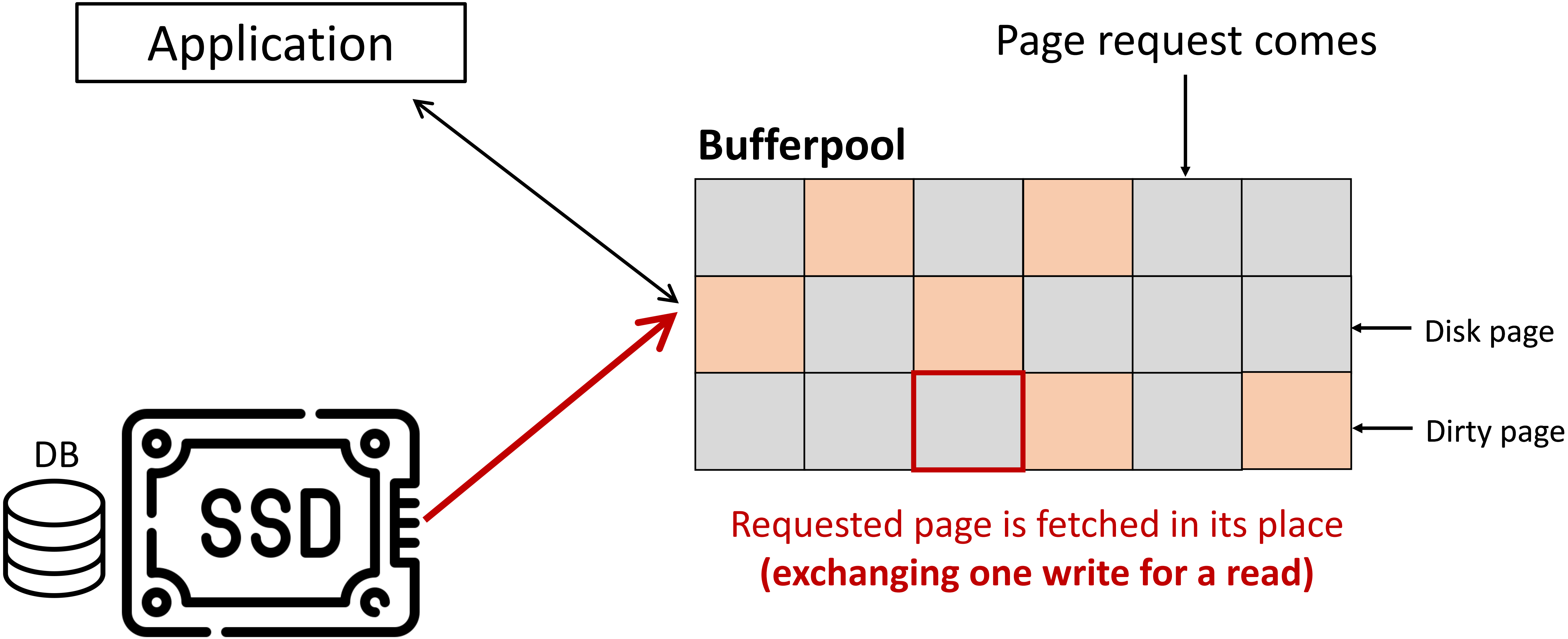
If BP is full, one page is selected for eviction based on **page replacement policy**

Traditional Bufferpool Manager



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

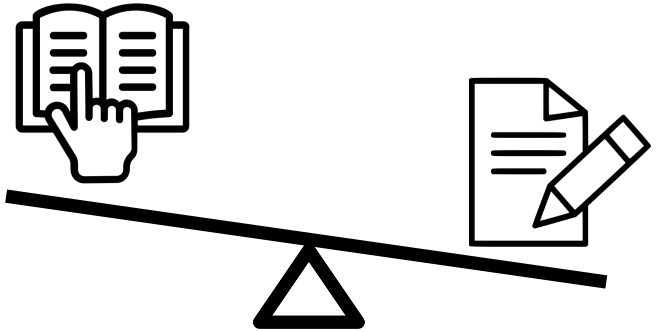
Traditional Bufferpool Manager



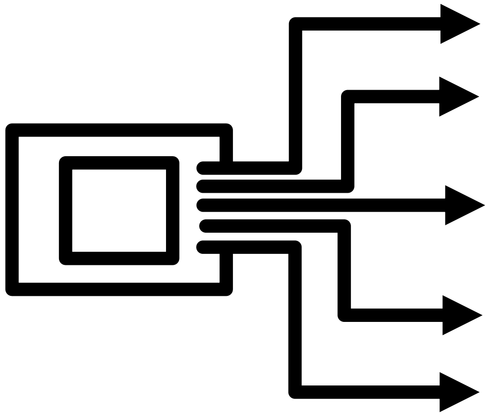
Requested page is fetched in its place
(exchanging one write for a read)

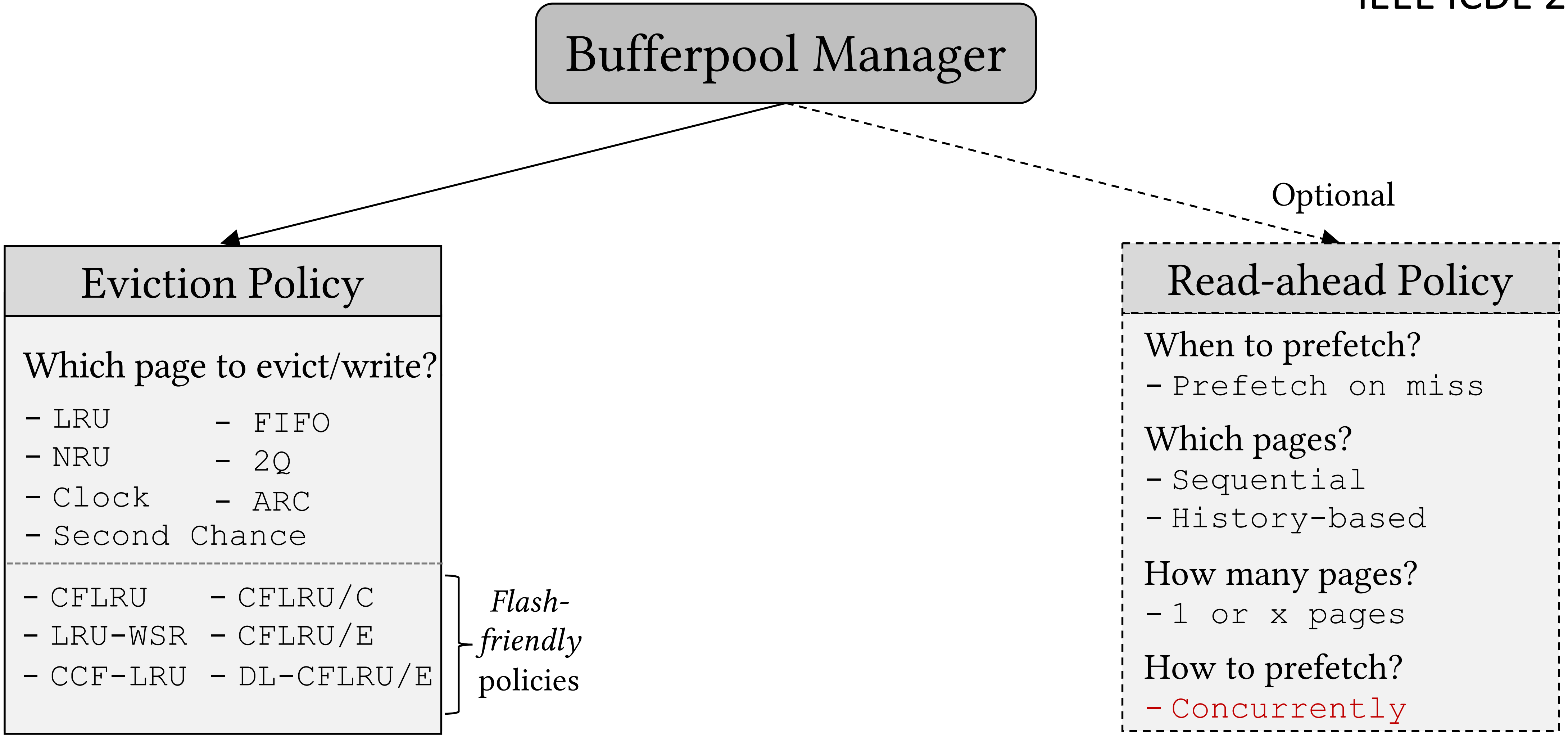
The Challenges

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

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

Flash-friendly policies

Read-ahead Policy

When to prefetch?

- Prefetch on miss

Which pages?

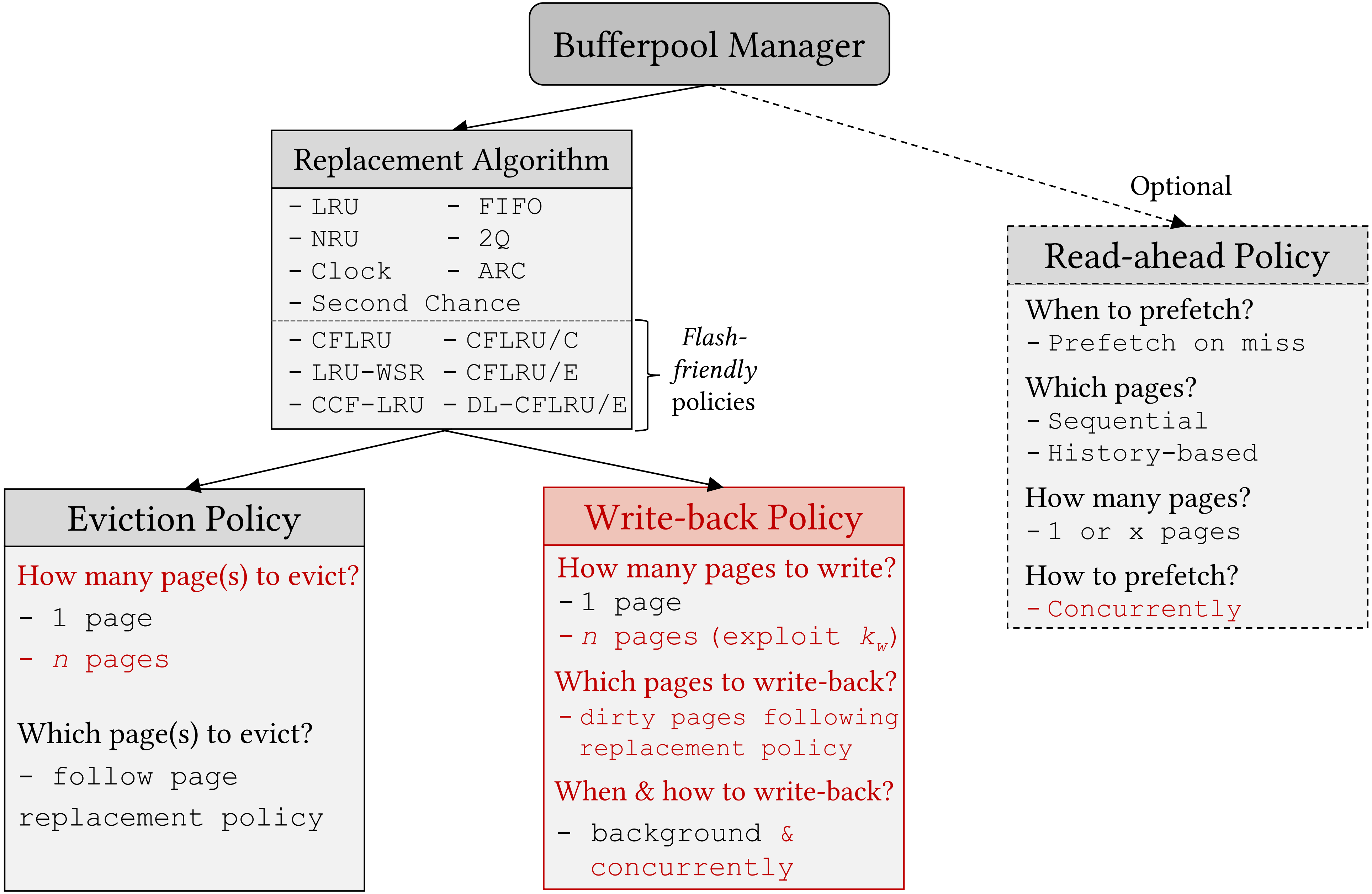
- Sequential
- History-based

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

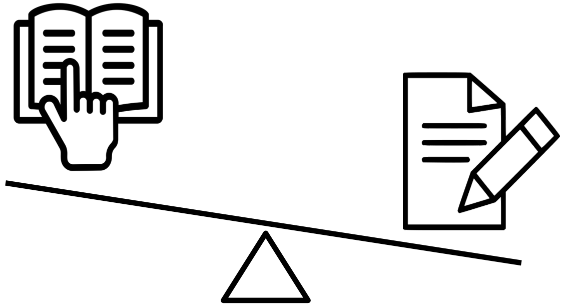
When & how to write-back?
- background & concurrently

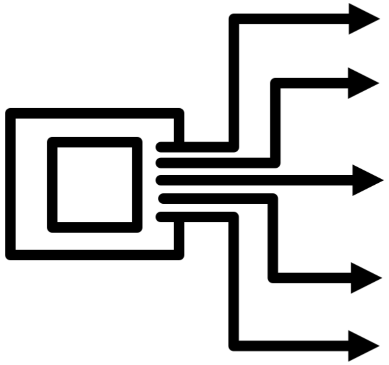
IEEE ICDE 2023

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

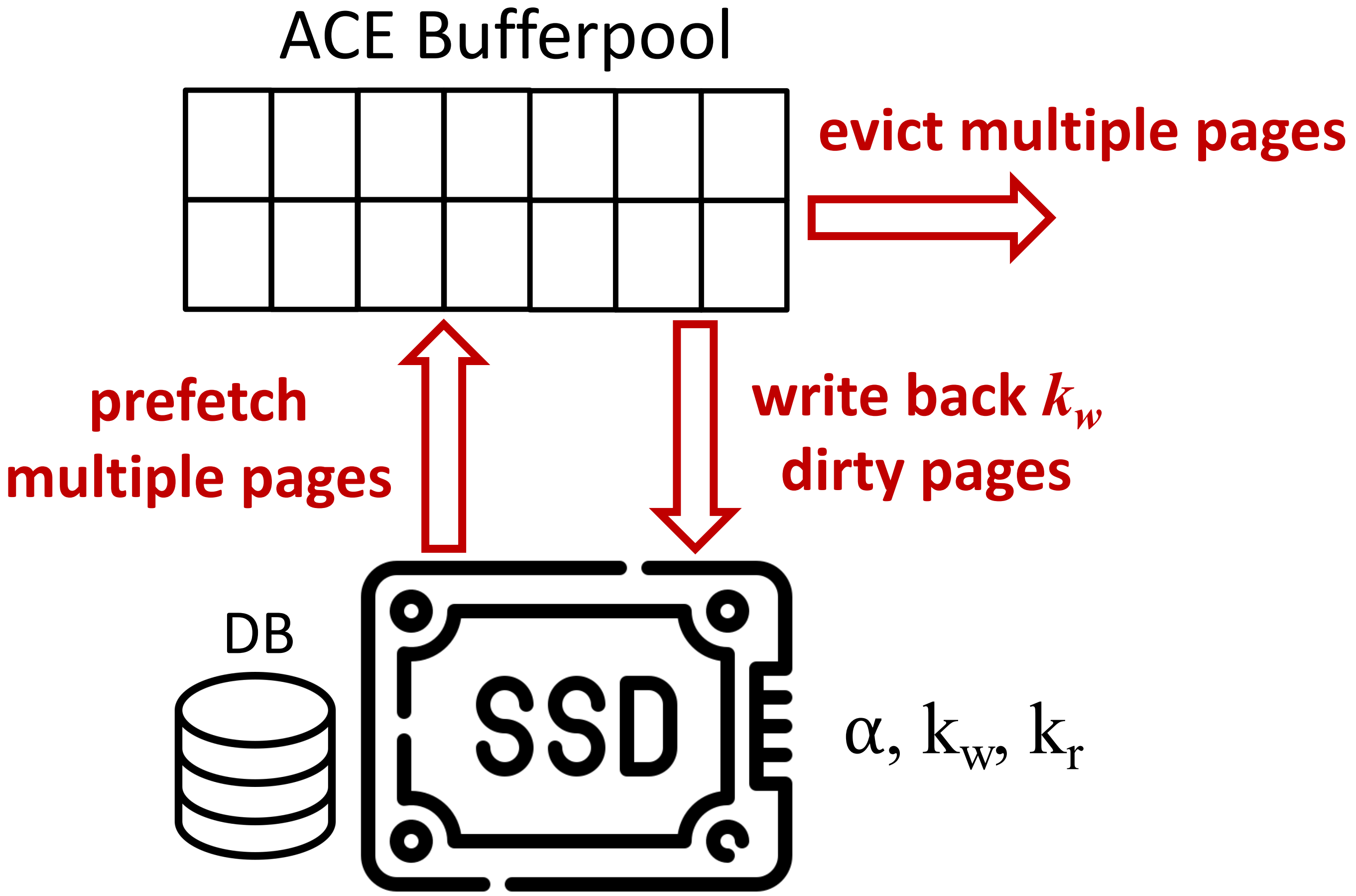
ACE Bufferpool Manager

Use device's properties





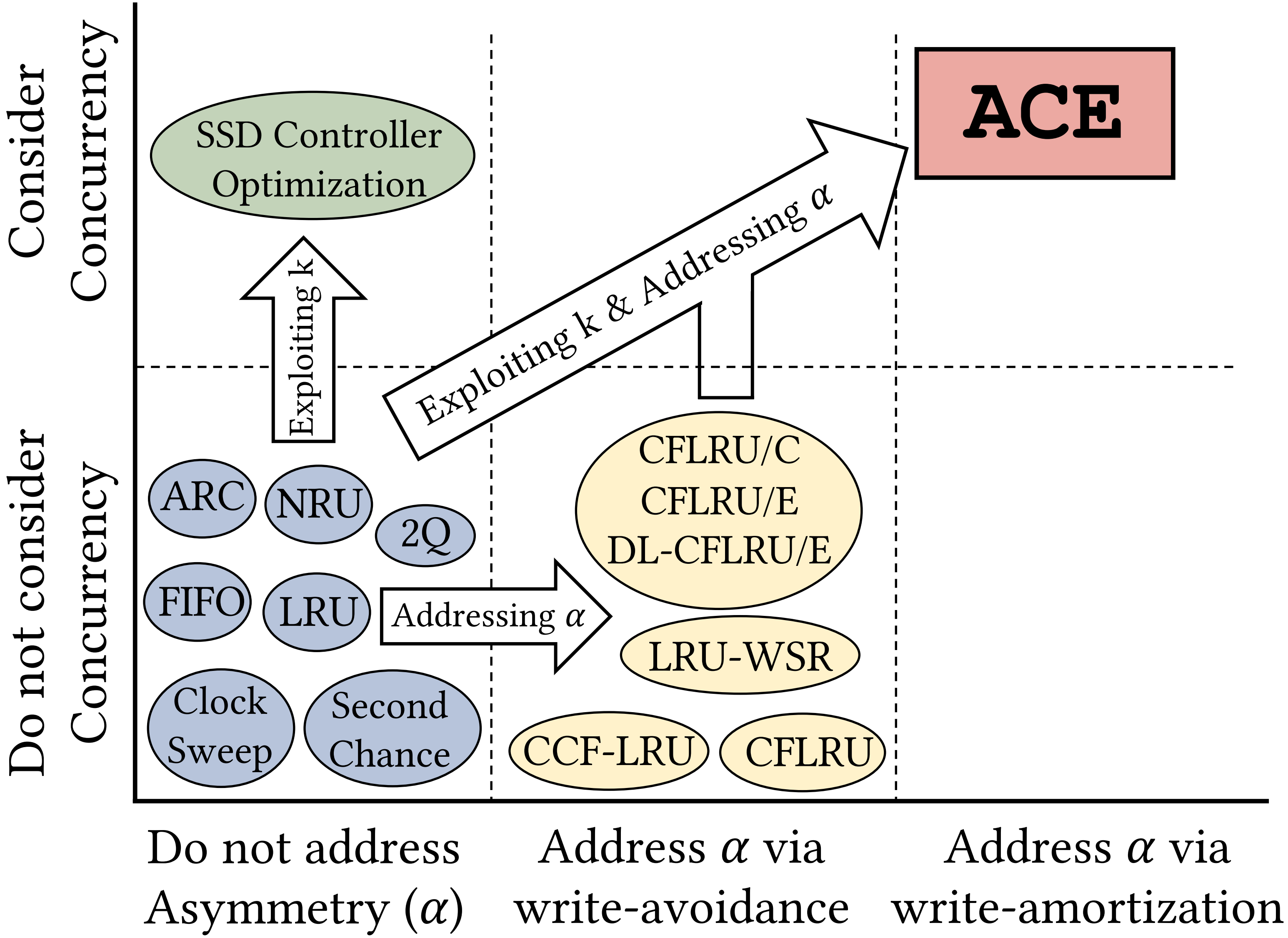
ACE Bufferpool Manager



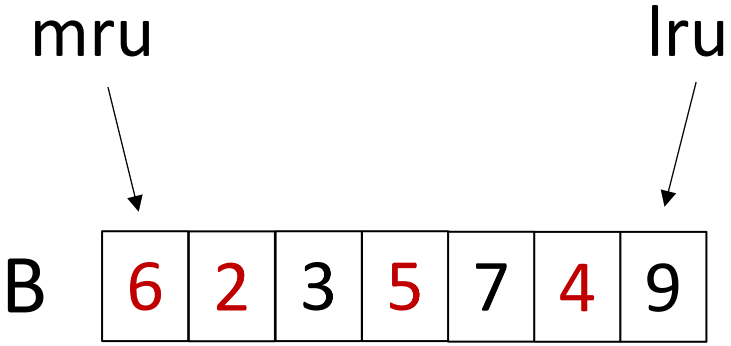
- ✓ Can be integrated with **any** replacement algorithm
- ✓ **Any** prefetching technique can be used

ACE Bufferpool Manager

- 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
Addressing α
Clock Sweep
Second Chance
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



mru

lru

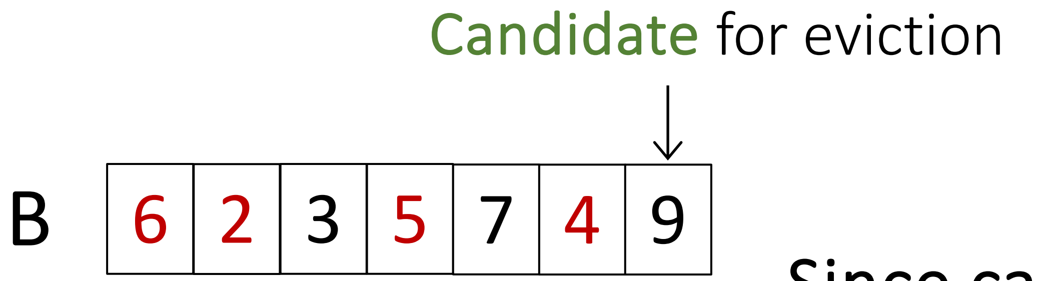
B	6	2	3	5	7	4	9

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

Write request of page 8 comes

An Example ($k_w = 3$)

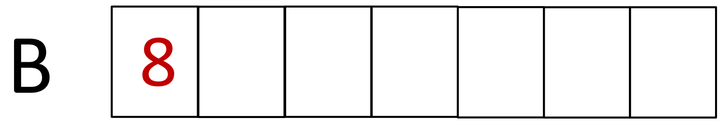
write page 8


Candidate for eviction

B	6	2	3	5	7	4	9

Since candidate page is clean, we simply evict 9

After eviction:

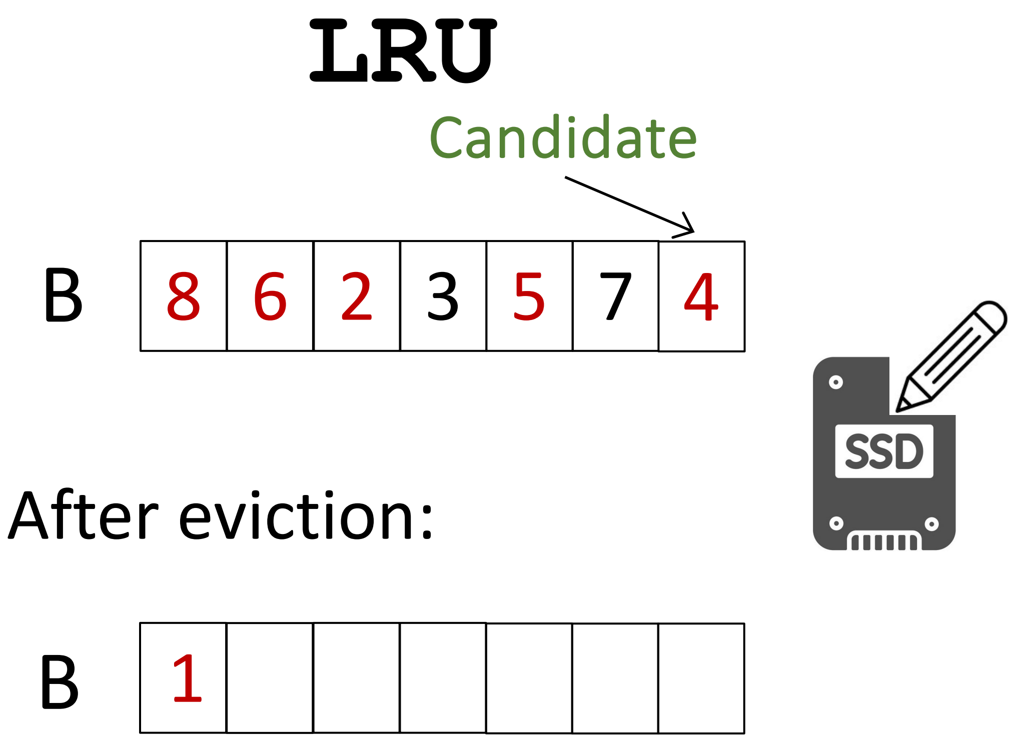


B	8						

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

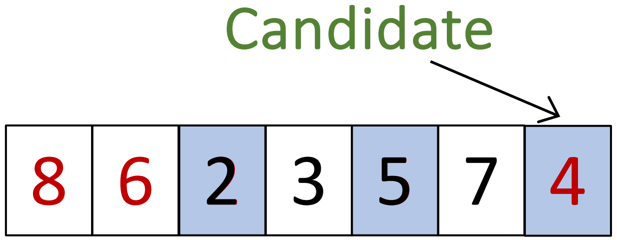
LRU

B							
	8	6	2	3	5	7	4

After eviction:

B							
	1	8	6	2	3	5	7

LRU+ACE (w/o PF)



8	6	2	3	5	7	4

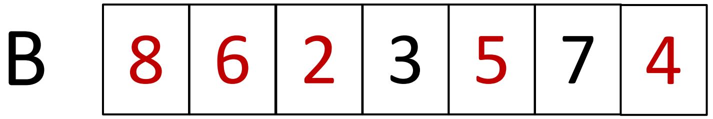
4,5,2 concurrently written

4 evicted

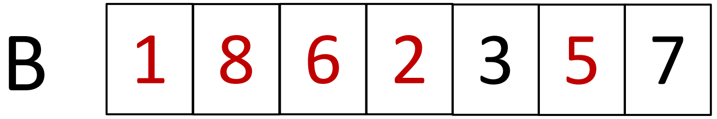
An Example ($k_w = 3$)

write page 1

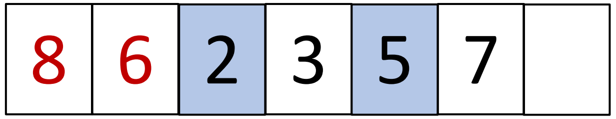
LRU



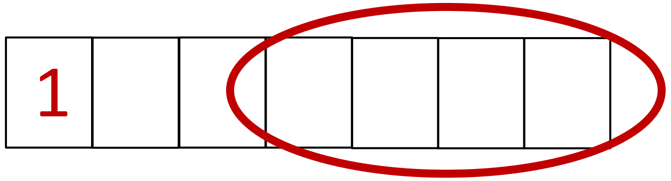
After eviction:



LRU+ACE (w/o PF)



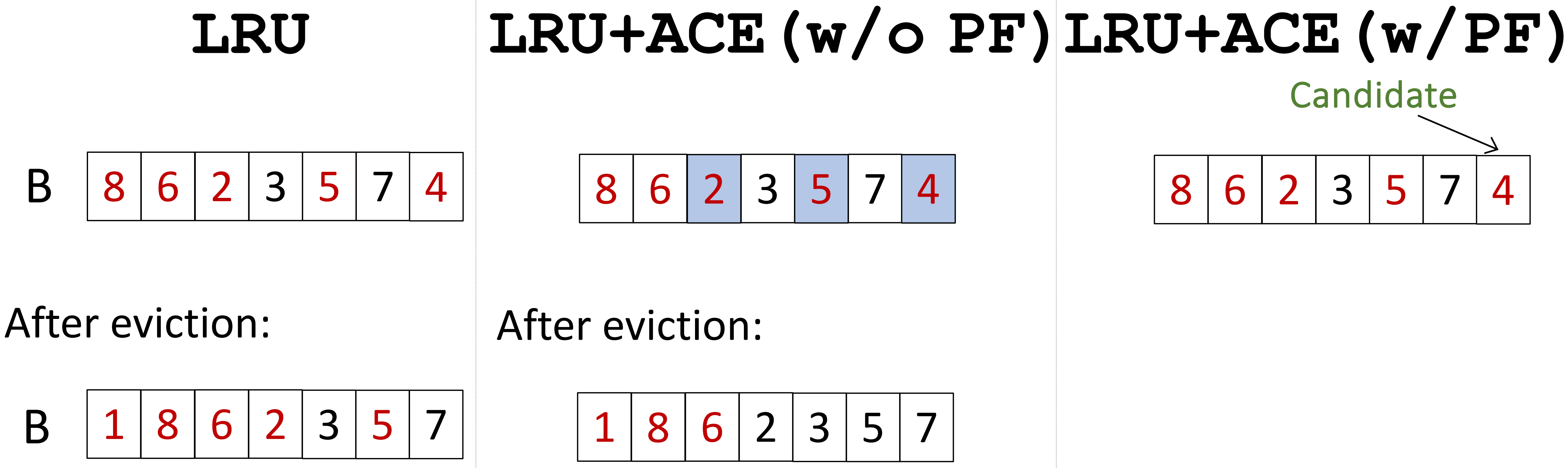
After eviction:



more clean pages

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	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)

eviction window

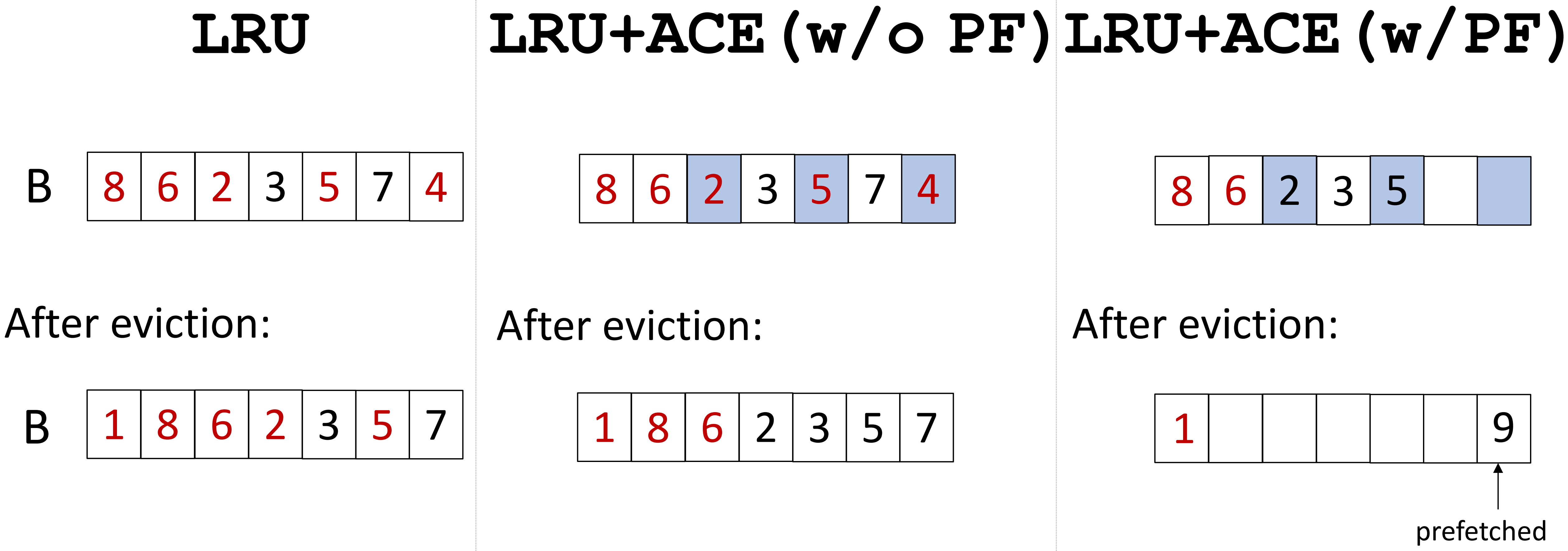
8	6	2	3	5	7	4

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 | | | | | | 9

prefetched

Experimental Evaluation

PostgreSQL

11.5

Clock Sweep
LRU
CFLRU
LRU-WSR

vs their ACE counterparts

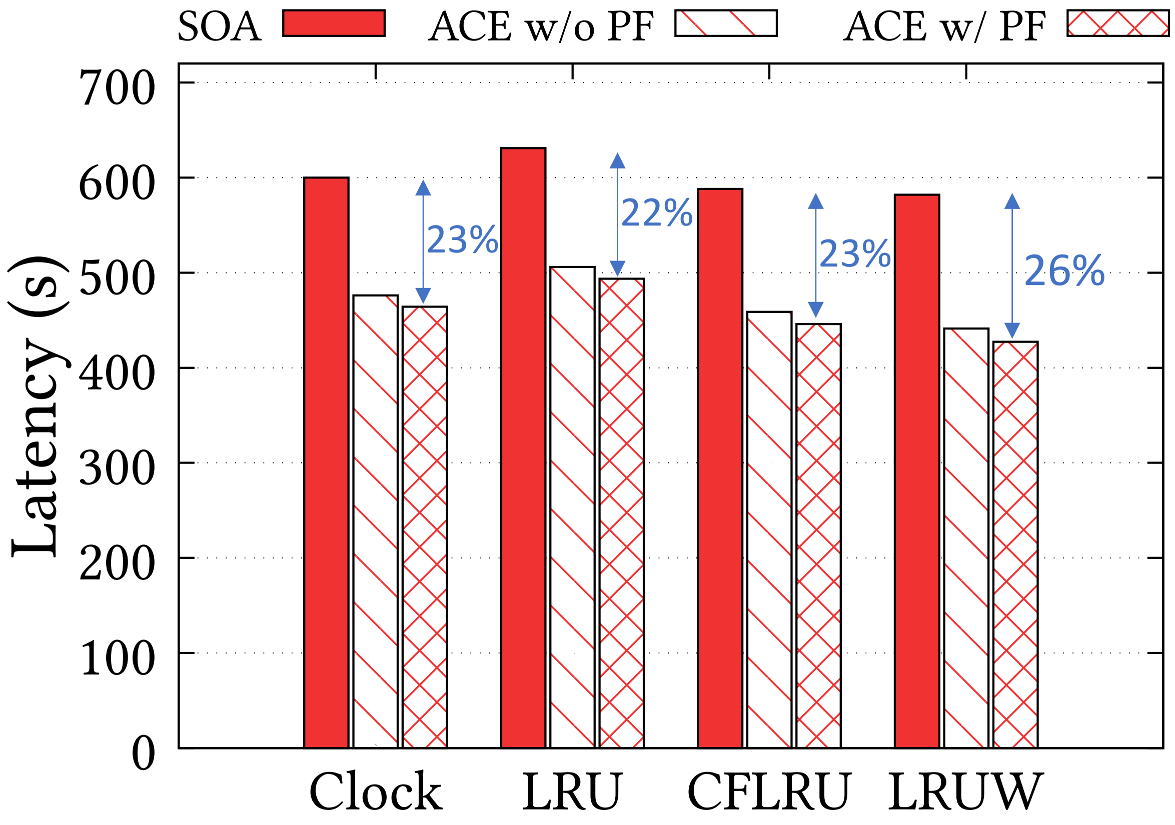
Device	α	k_r	k_w
Optane SSD	1.1	6	5
PCIe SSD	2.8	80	8
SATA SSD	1.5	25	9
Virtual SSD	2.0	11	19

Workload:

synthesized traces

TPC-C benchmark

ACE Improves Runtime



Mixed Skewed Trace
(r/w: 50/50, locality 90/10)

Device: PCIe SSD

$\alpha = 2.8, k_w = 8$

ACE improves runtime by 22-26%

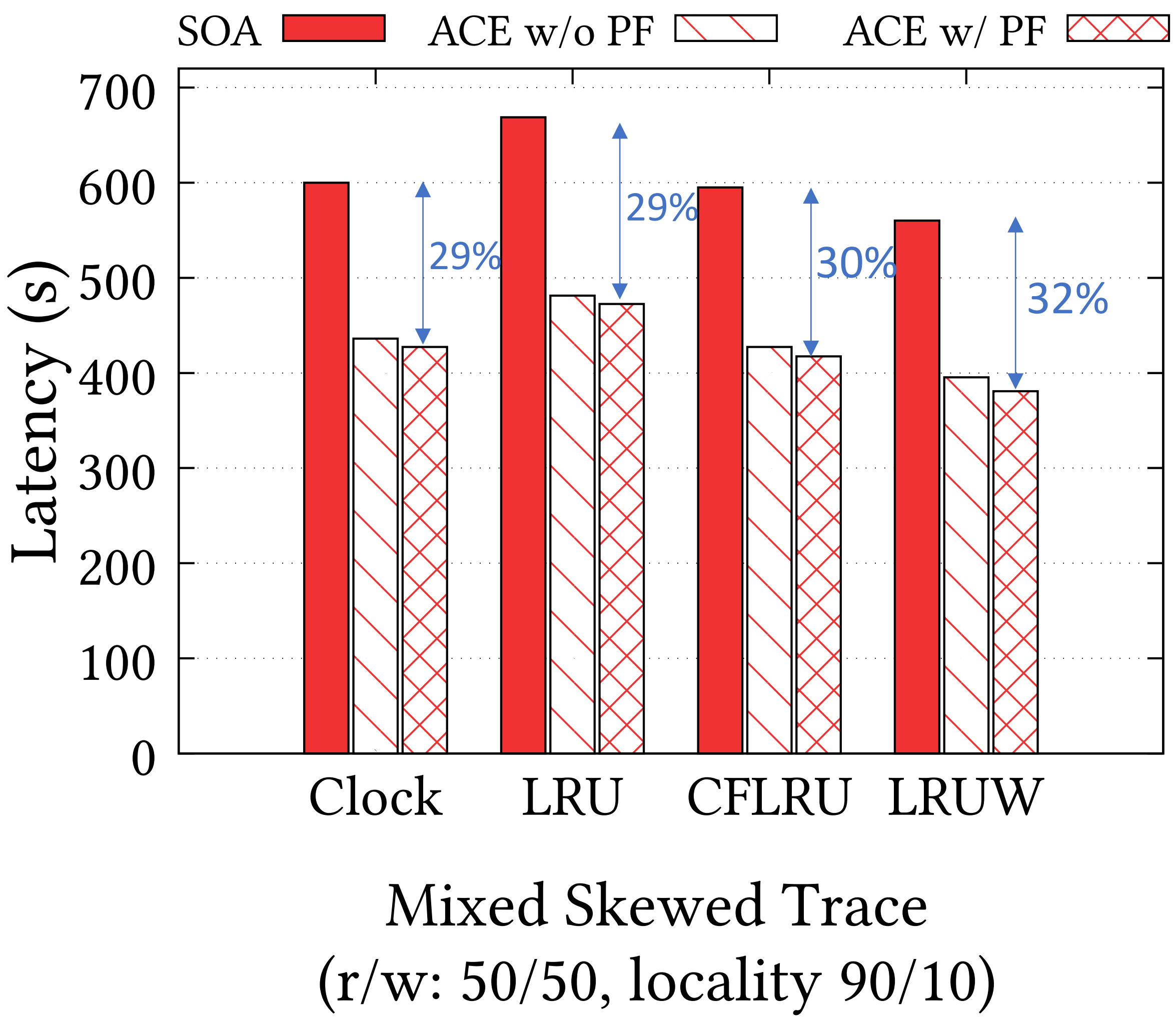
Negligible increase in buffer miss (<0.009%)

Benefit comes at no cost

Higher Gain for Write-Heavy Workload

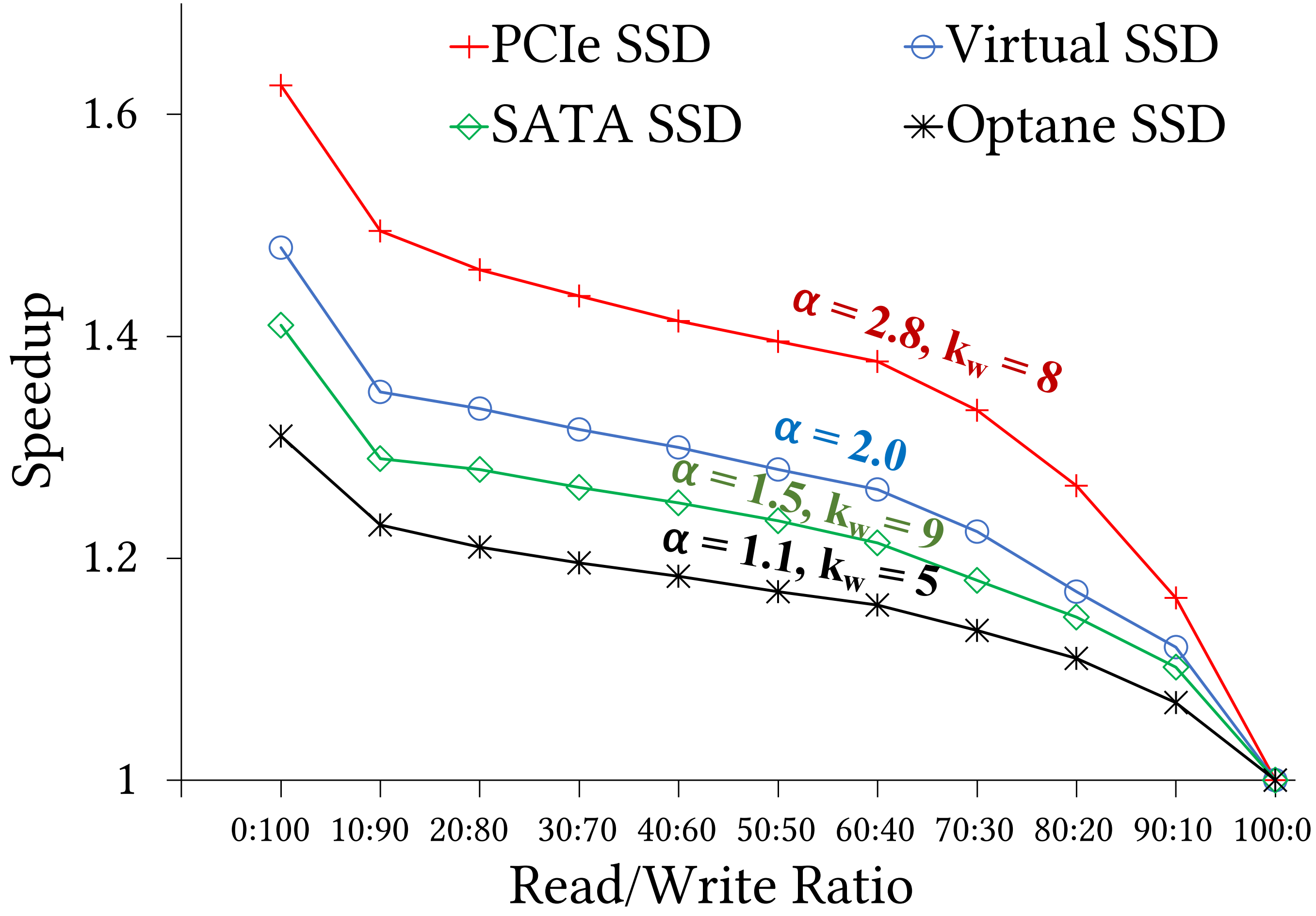
Device: PCIe SSD

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



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

Impact of R/W Ratio & Asymmetry

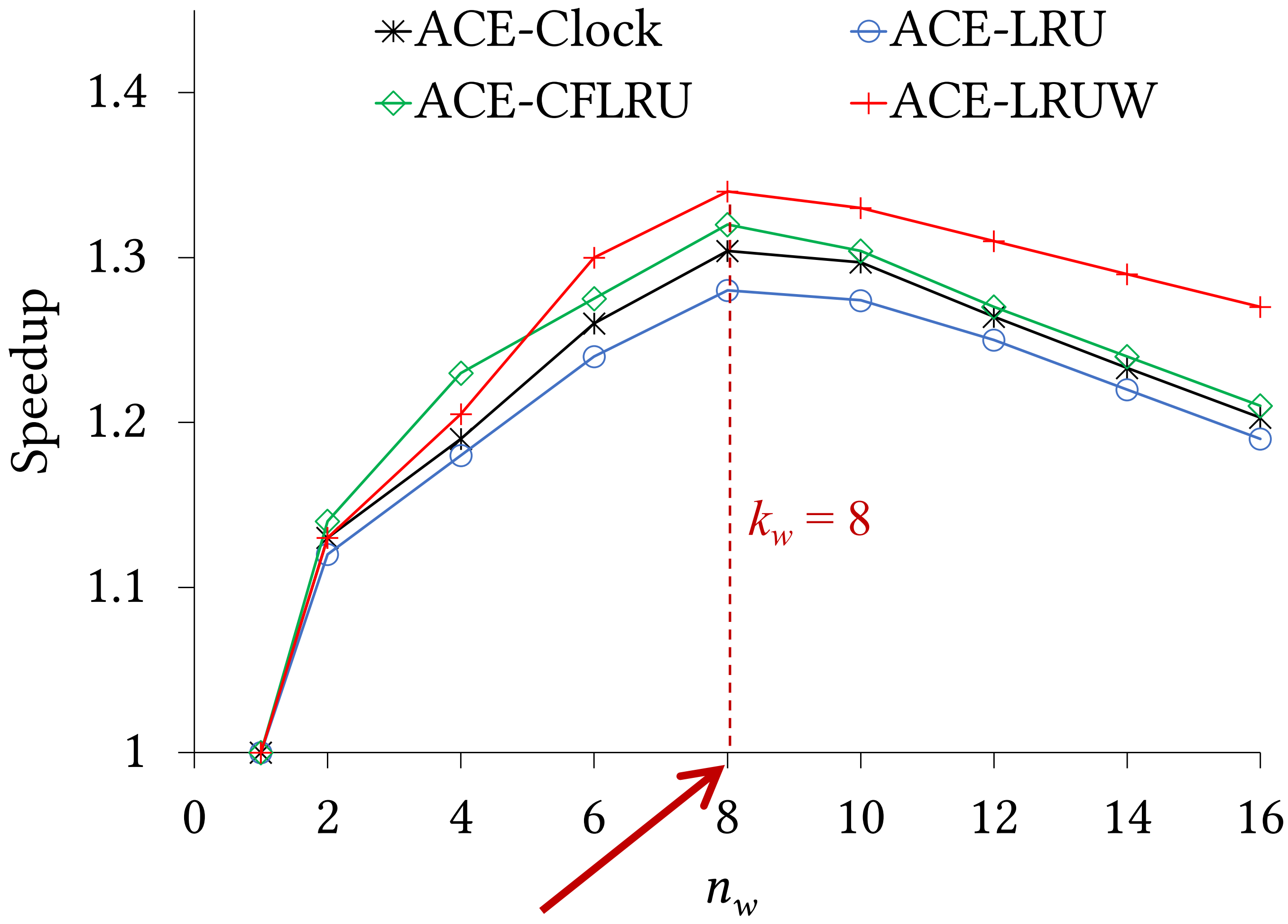


more writes, more speedup

higher asymmetry, higher speedup

good benefit even for low asymmetry

Impact of #Concurrent I/Os



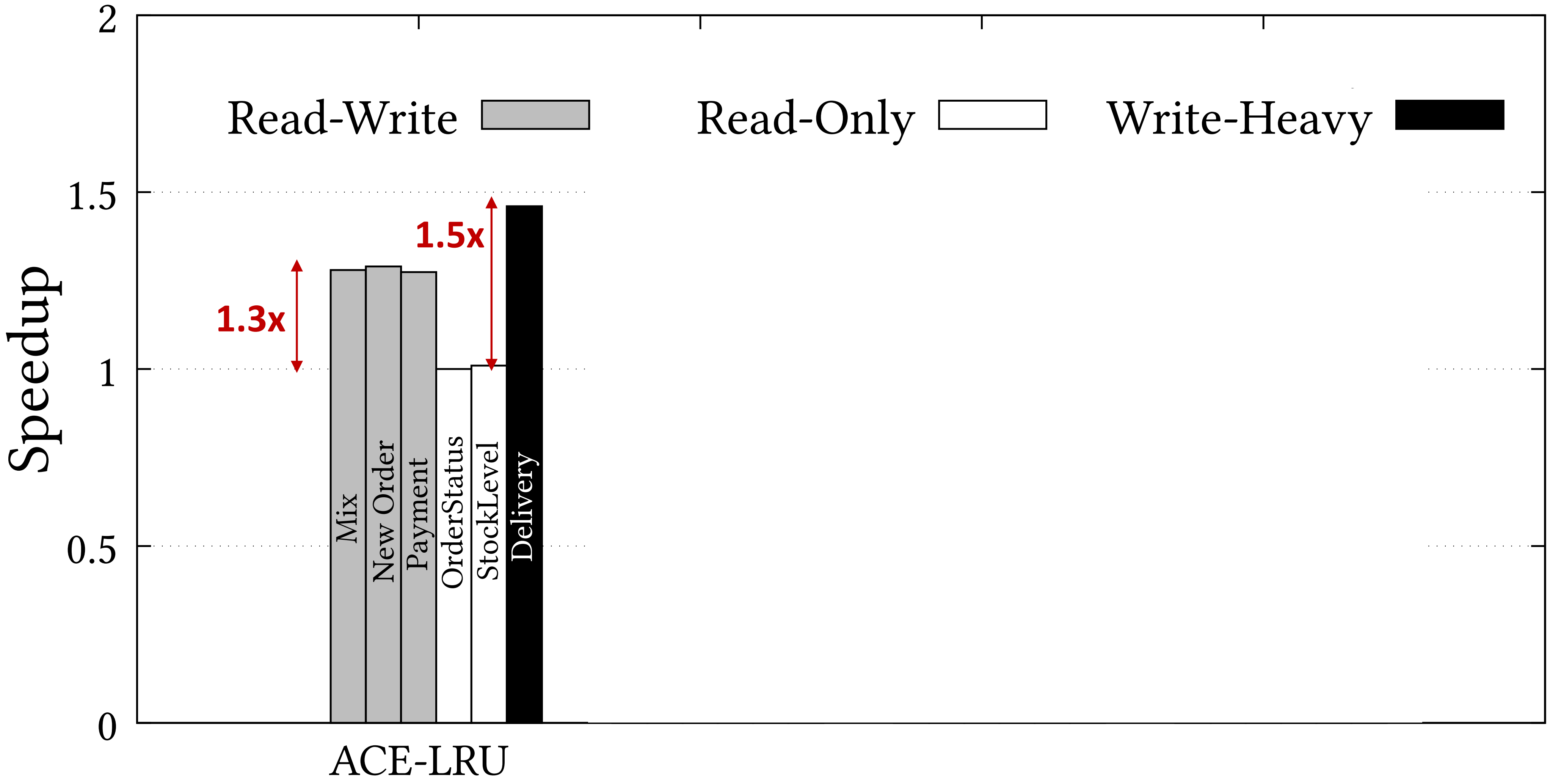
Mixed Skewed Trace
(r/w: 50/50, locality 90/10)

Device: PCIe SSD

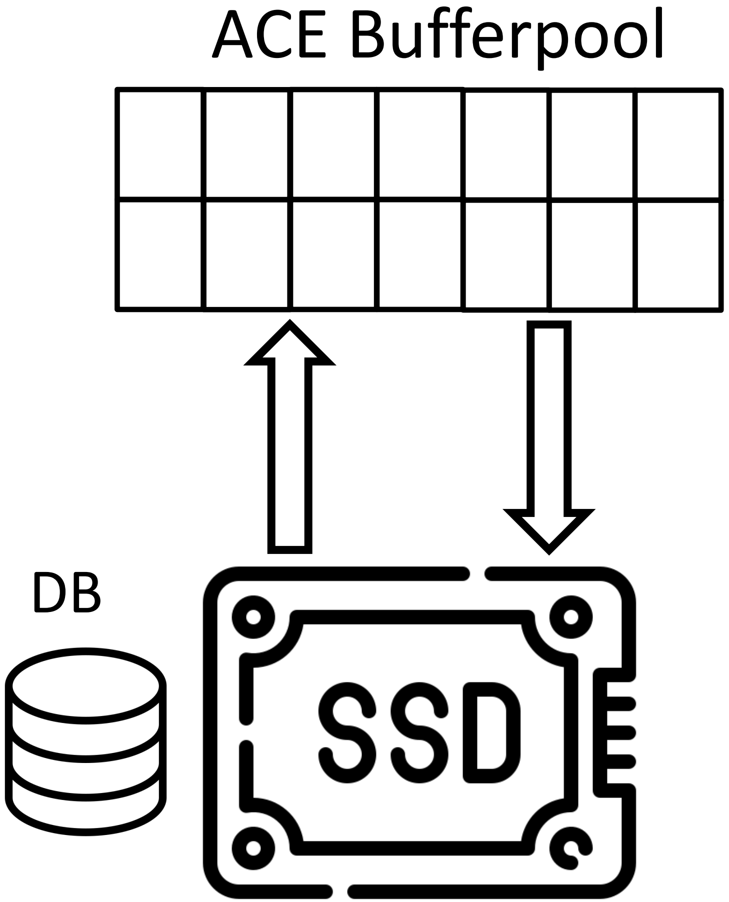
$\alpha = 2.8$, $k_w = 8$

Highest speedup when optimal concurrency is used

Experimental Evaluation (TPC-C)



ACE Achieves 1.3x for mixed TPC-C



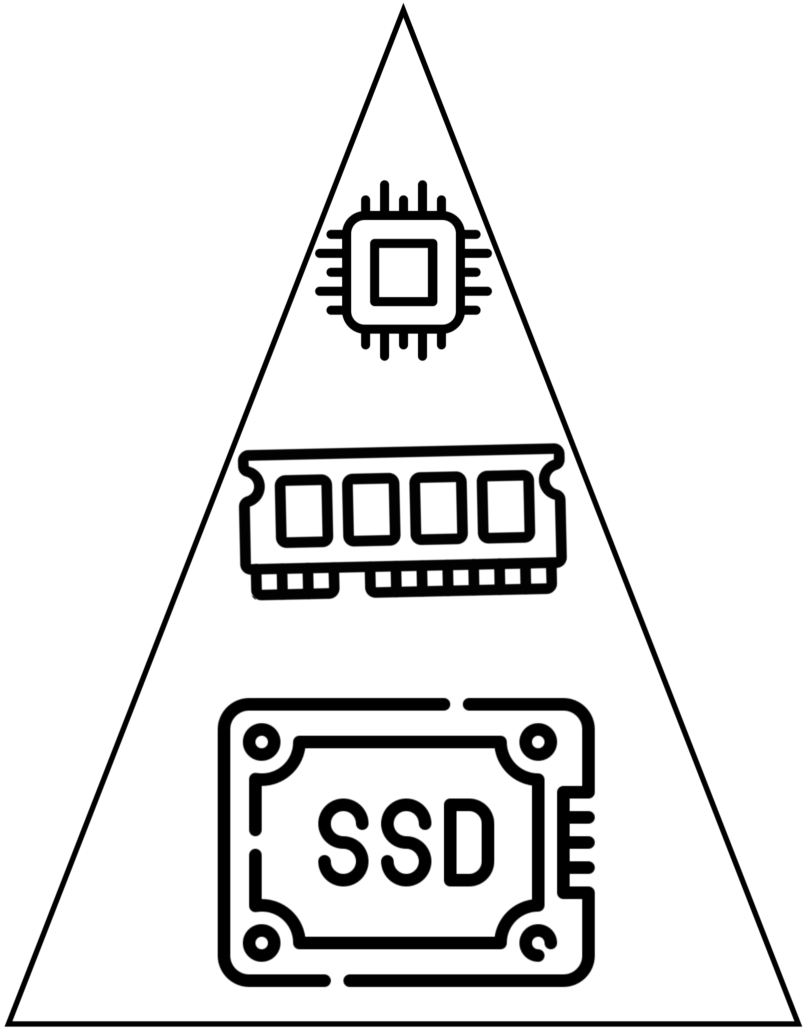
ACE works with **any** page replacement policy

Any prefetching technique can be used

With low engineering effort, **any** DBMS

bufferpool can benefit from this approach

Goal: Developing Hardware-Aware Data Systems



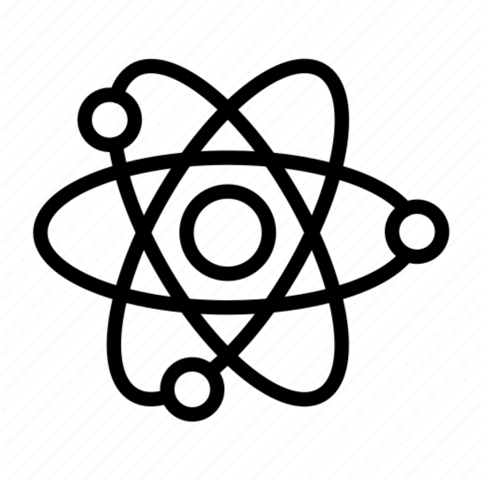
Tailor Data Systems
for **SSD Asymmetry**
& **Concurrency**

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**]

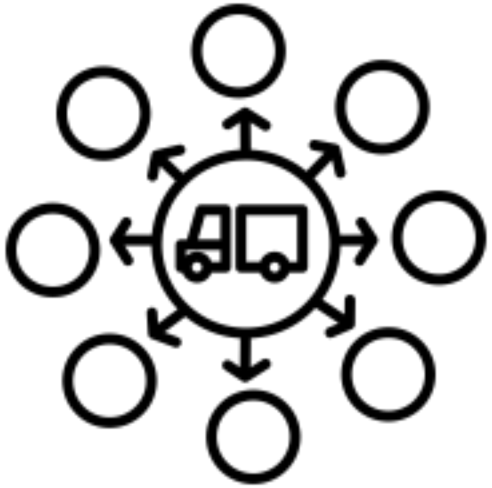
Rise of Large Graphs

Graphs are everywhere!

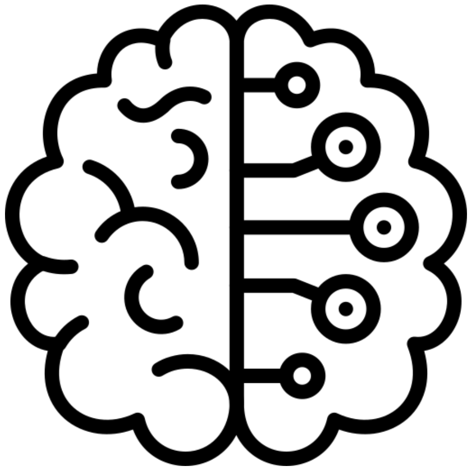
Social Network



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

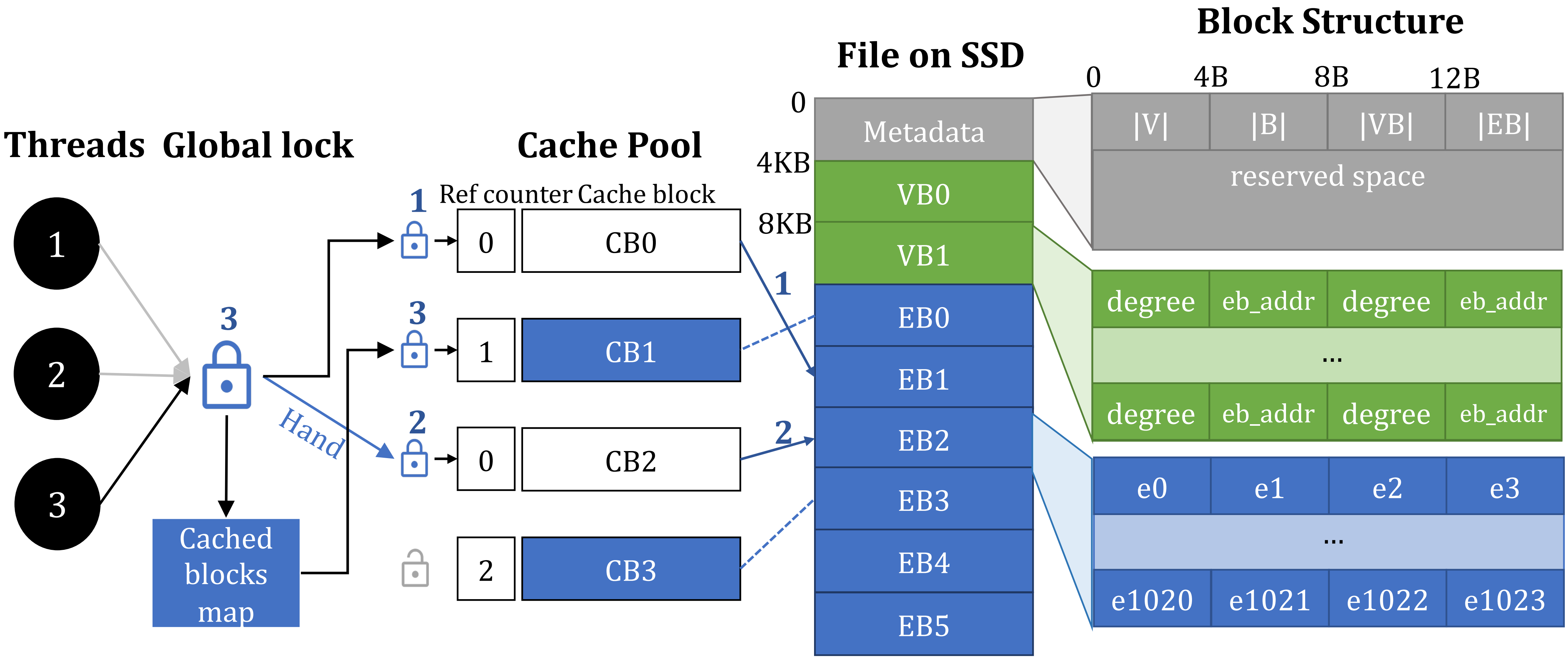
Our Goal

- 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

CAVE Architecture



Concurrent Graph Algorithms

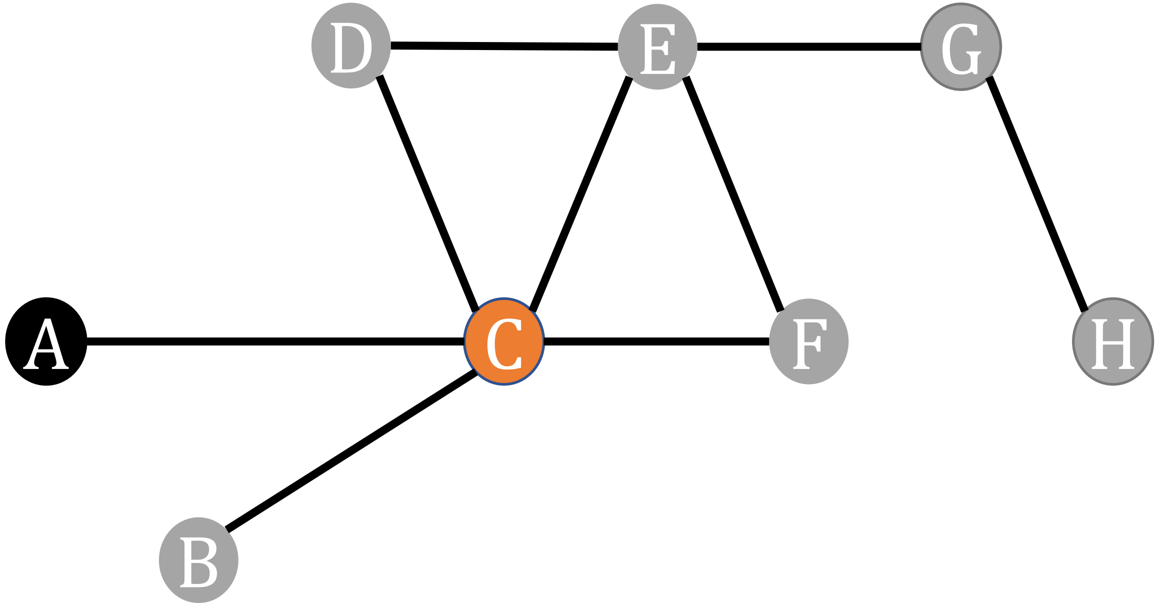
- Parallel Breadth-First Search
- Parallel pseudo Depth-First Search
- Parallel Weakly Connected Components
- Parallel PageRank
- Parallel Random Walk

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

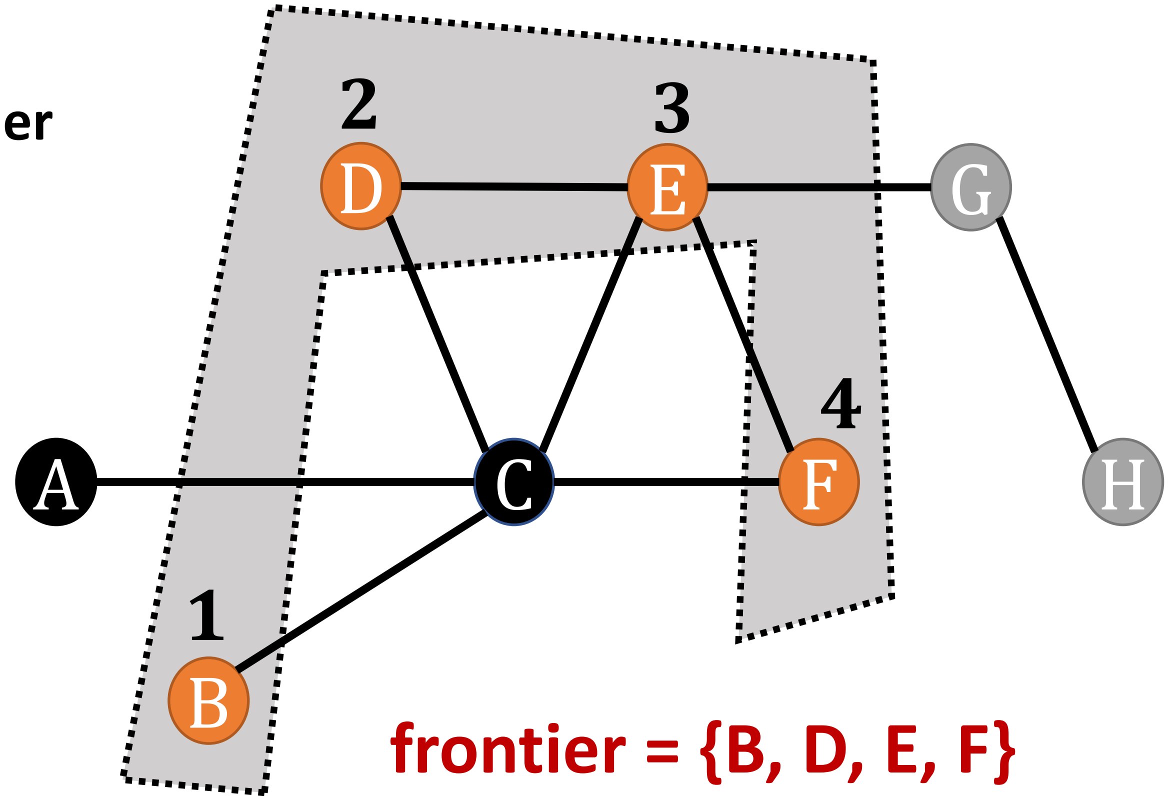


Parallel BFS

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

Each iteration involves

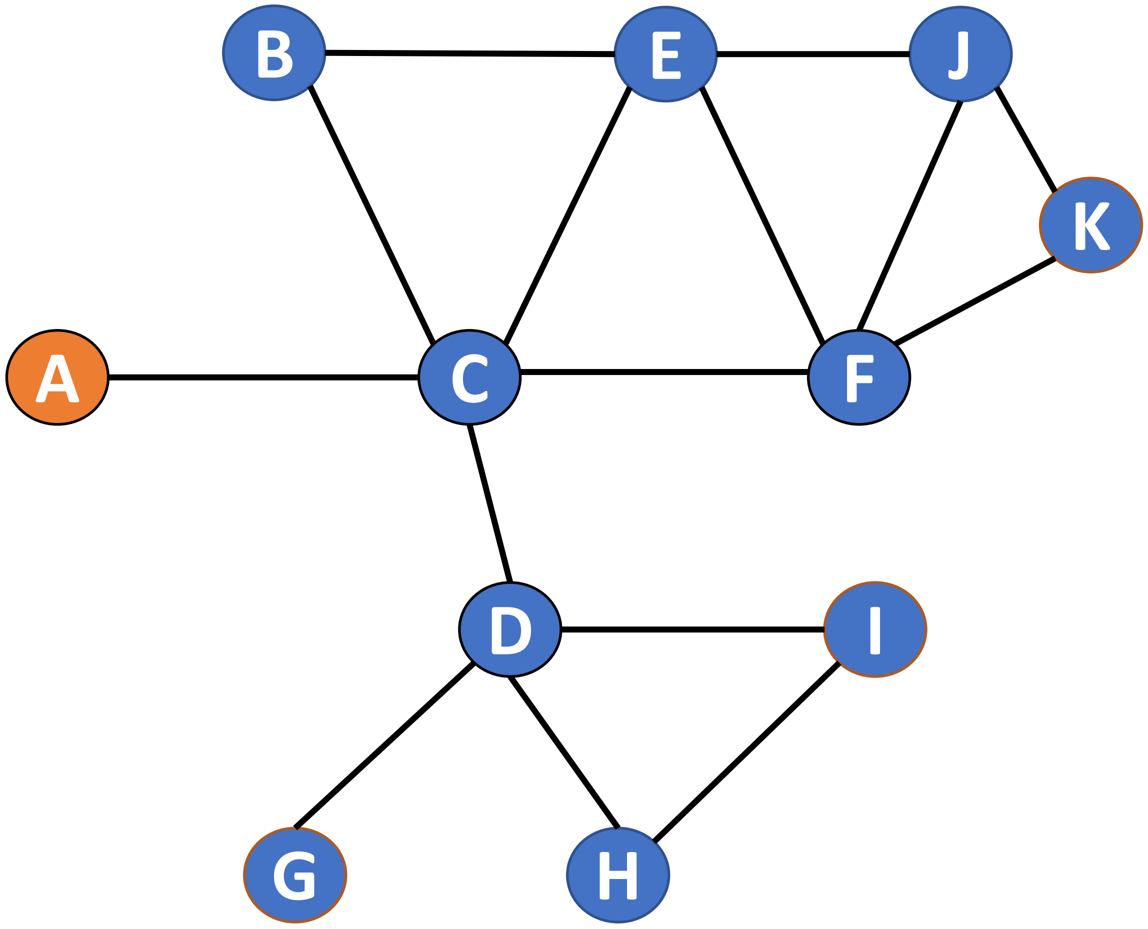
1. processing a list of vertices aka the **frontier**
2. accessing the neighbors of each vertex
3. updating vertex values
4. determining which vertices should be visited in the next iteration



frontier = {B, D, E, F}

Parallel pseudo DFS

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

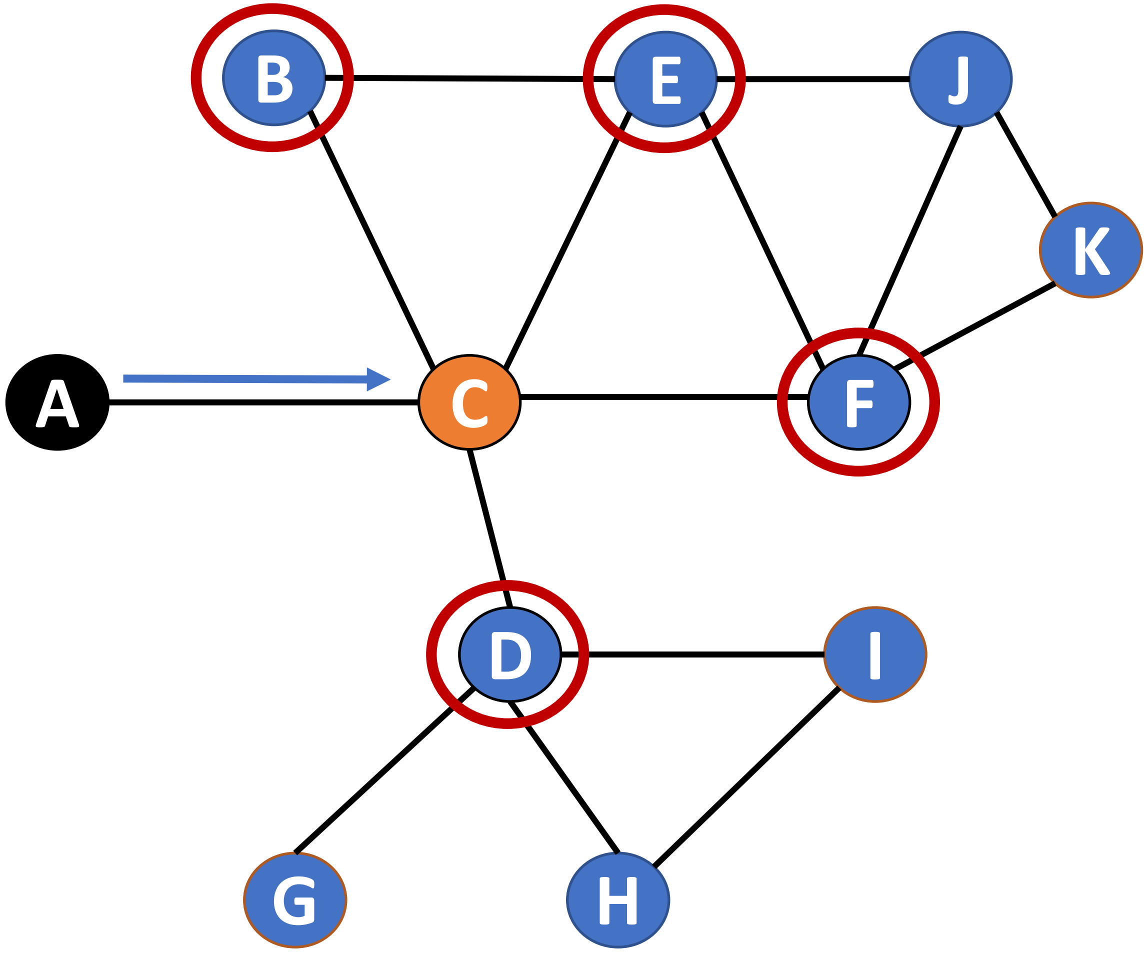


Time

1 A **Thread #1**

Parallel pseudo DFS

⚫ processed nodes 🟠 processing in progress 🔵 yet to be processed



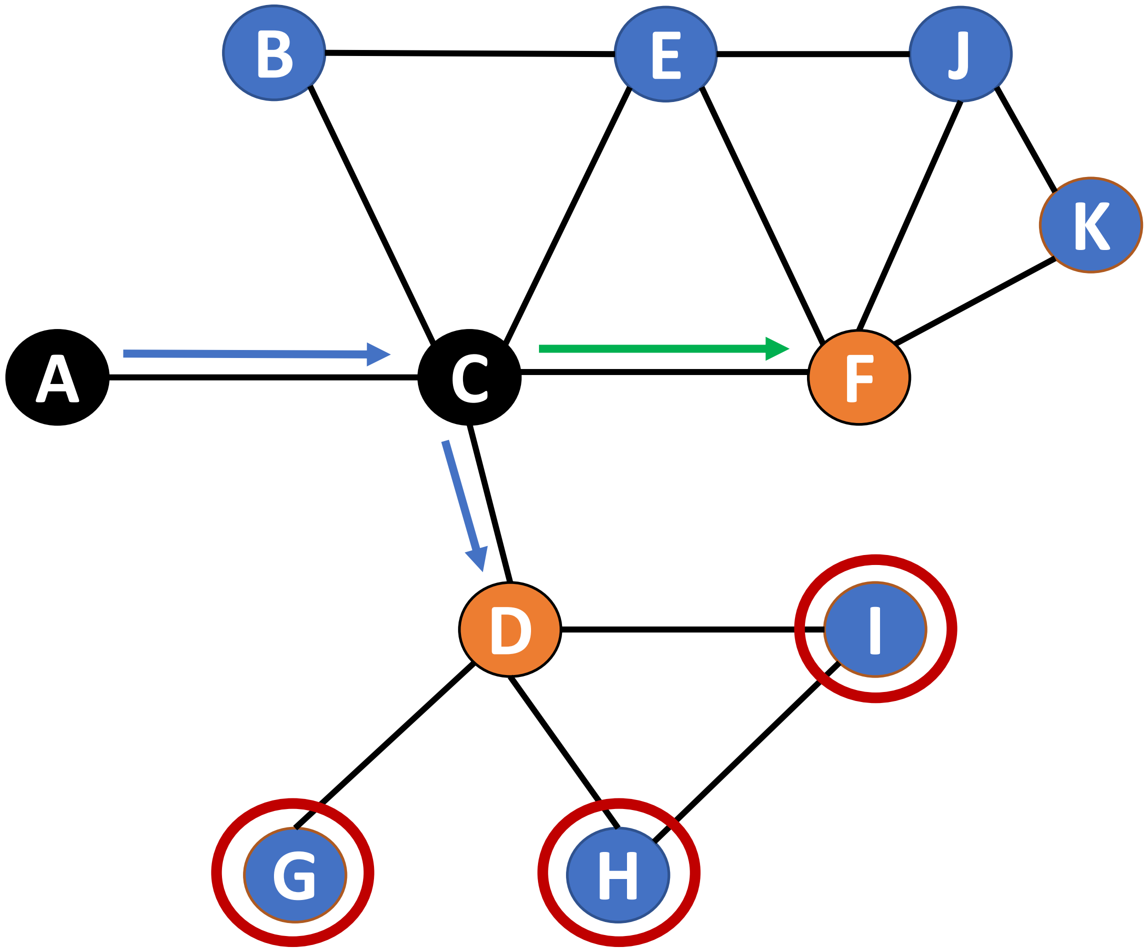
Time

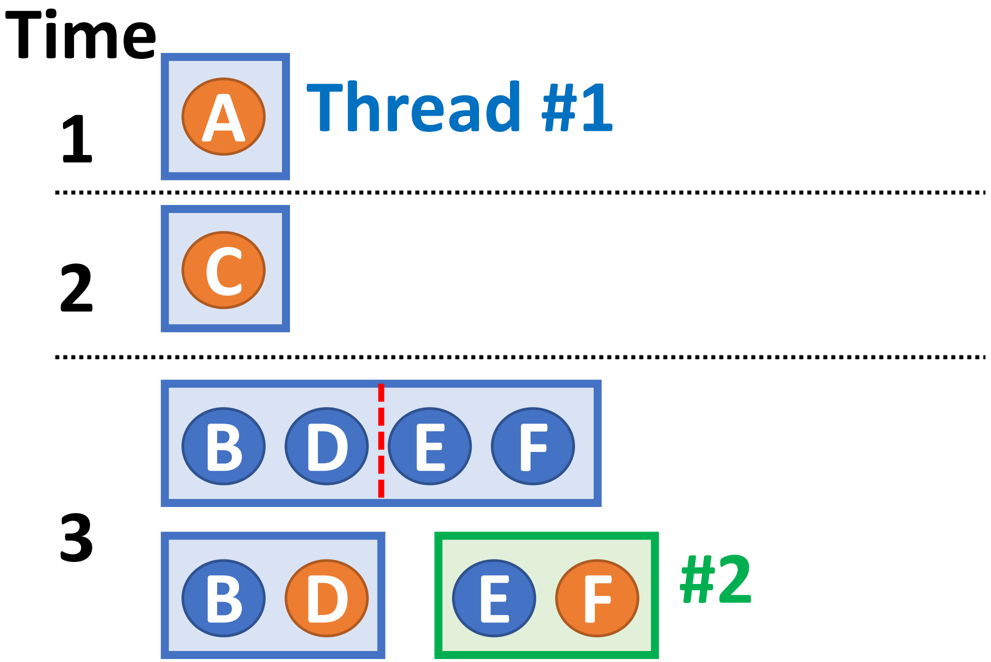
1 A **Thread #1**

2 C

Parallel pseudo DFS

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




Time

1 A Thread #1

2 C

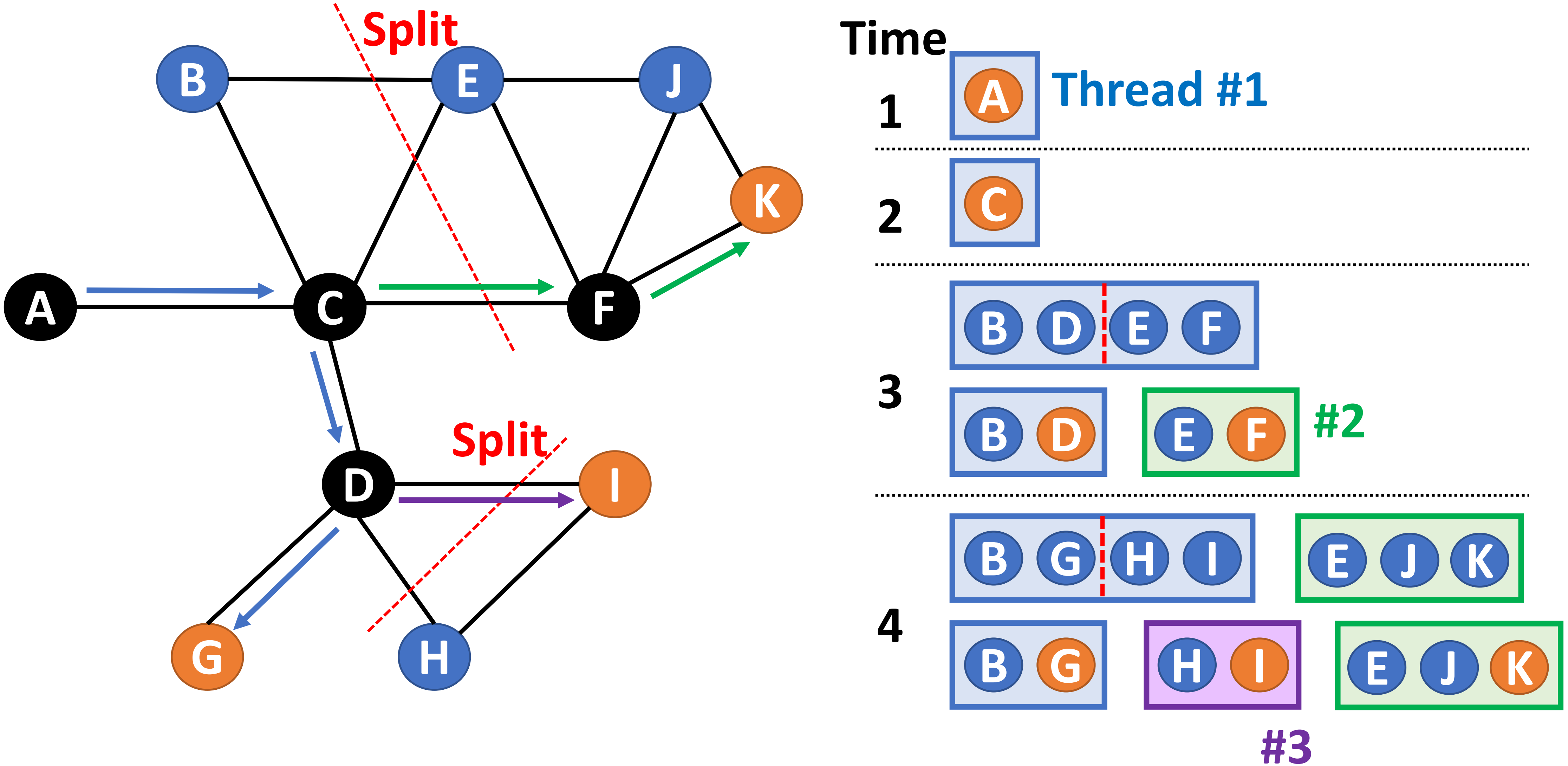
3 B D E F

B D

E F #2

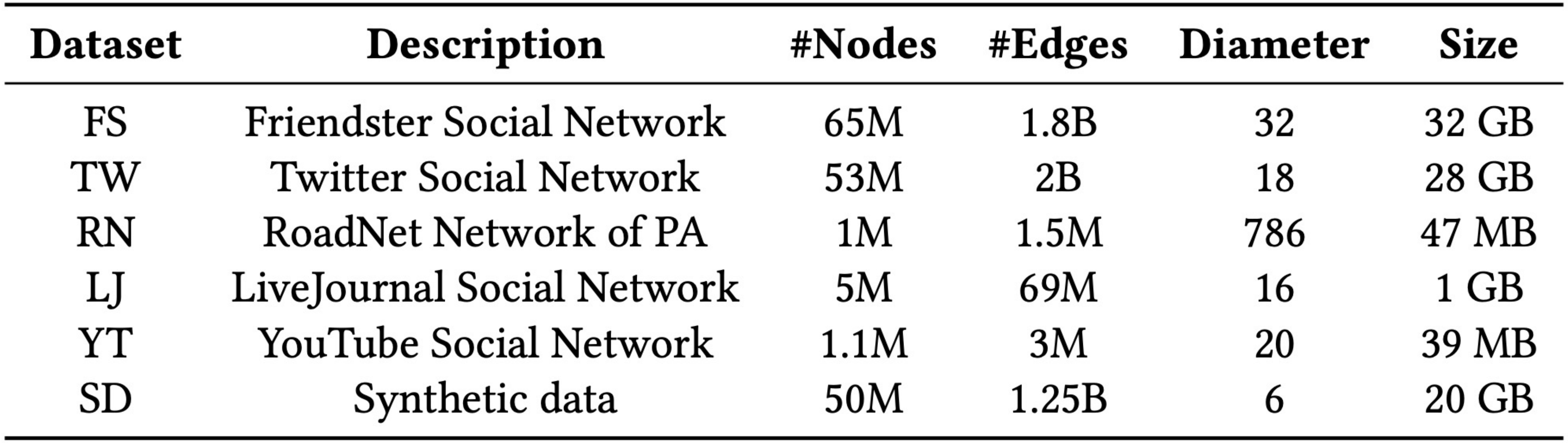
Parallel pseudo DFS

● 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

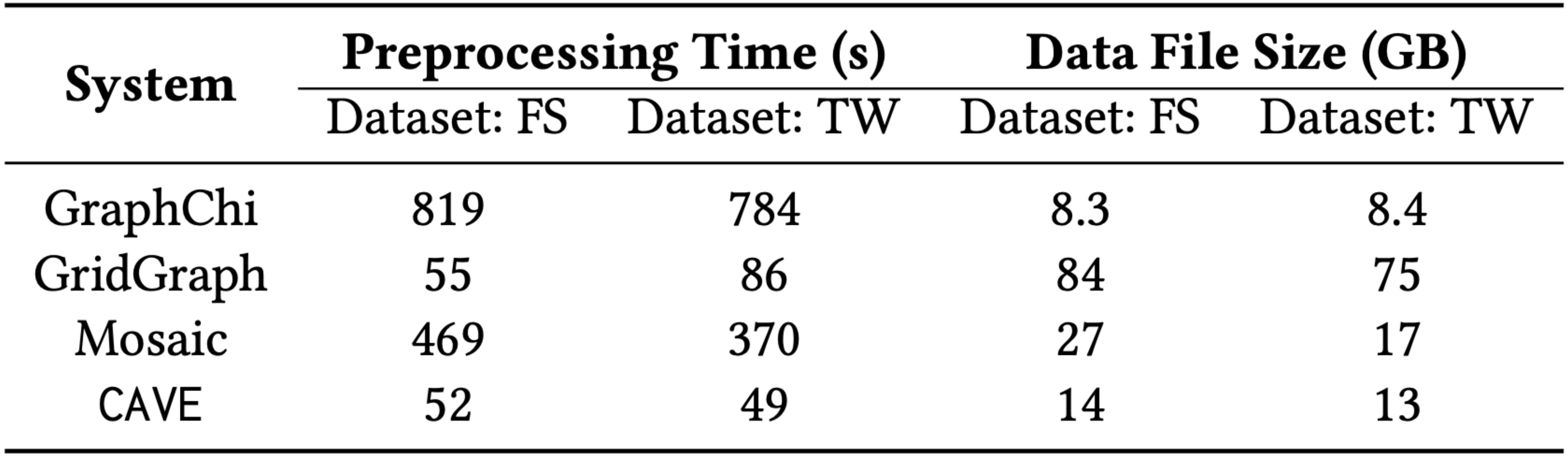
3 devices

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

Approaches Used:

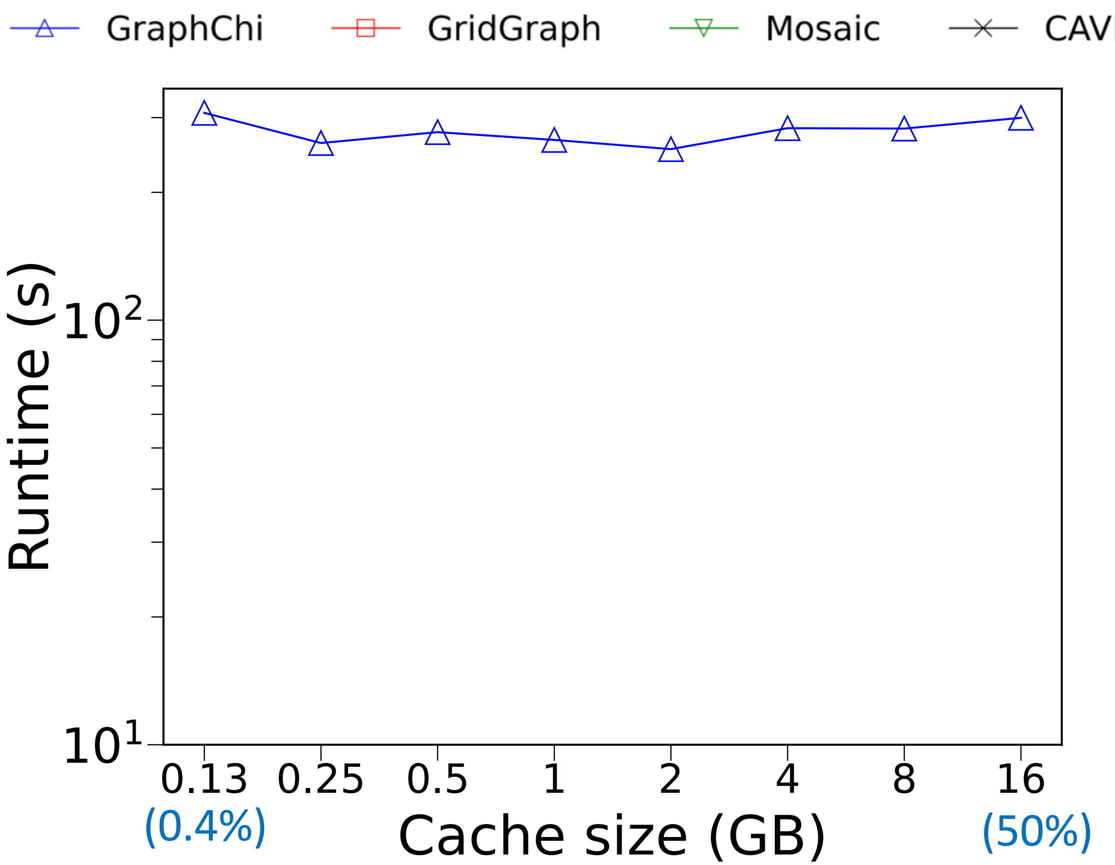
GraphChi, GridGraph, Mosaic, CAVE, CAVE_blocked

CAVE's Preprocessing is Efficient



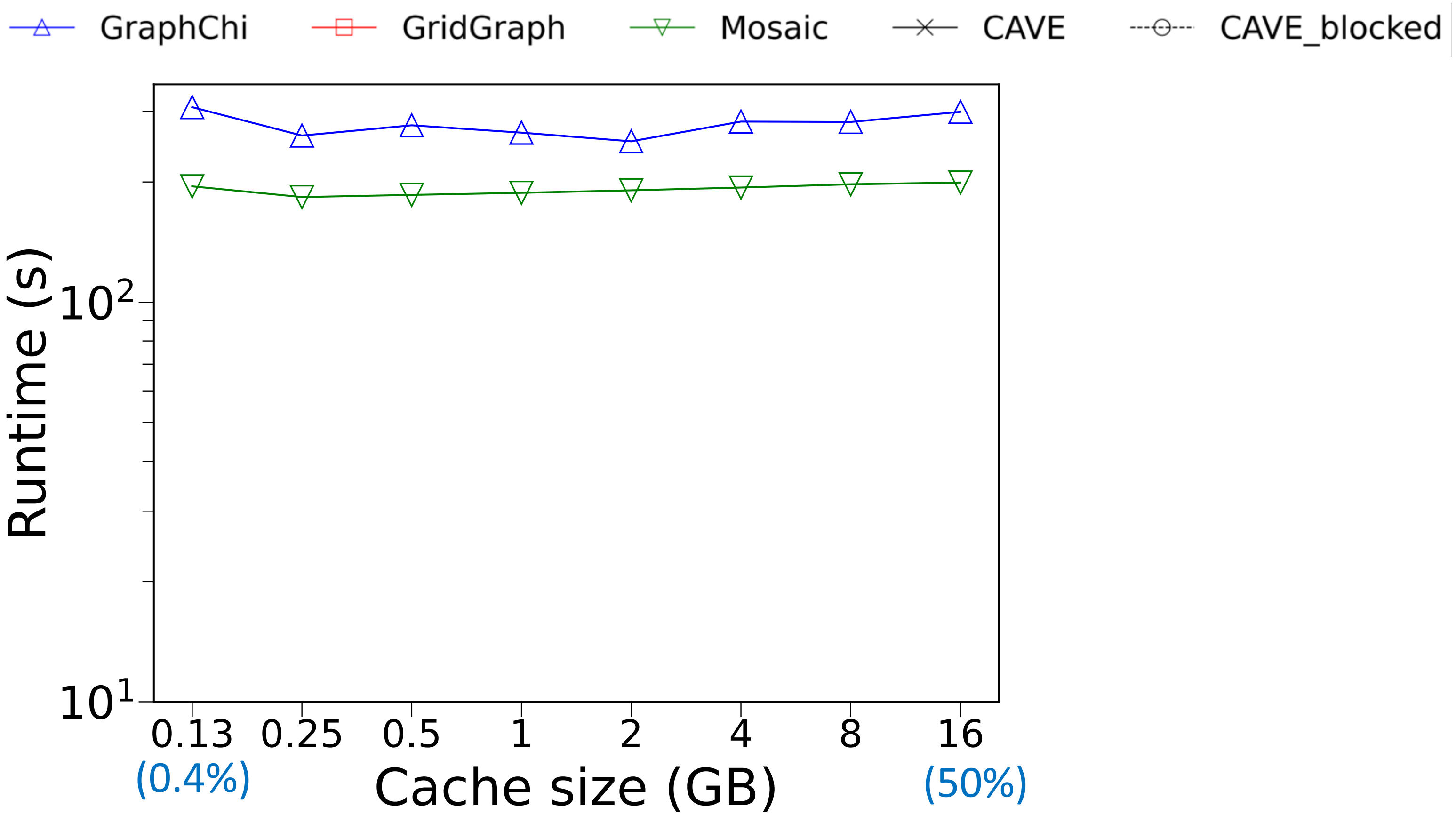
System	Preprocessing Time (s)		Data File Size (GB)	
	Dataset: FS	Dataset: TW	Dataset: FS	Dataset: TW
GraphChi	819	784	8.3	8.4
GridGraph	55	86	84	75
Mosaic	469	370	27	17
CAVE	52	49	14	13

CAVE Performs Efficient PBFS



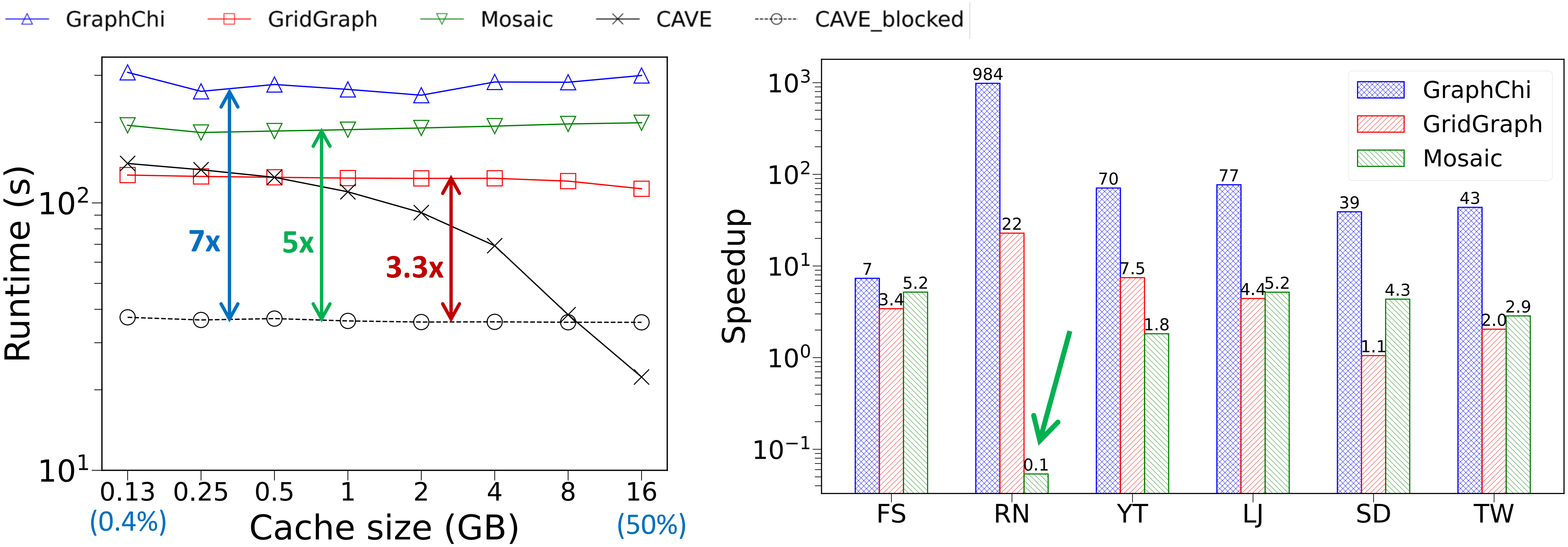
Dataset: FS

CAVE Performs Efficient PBFS



Dataset: FS

CAVE Performs Efficient PBFS



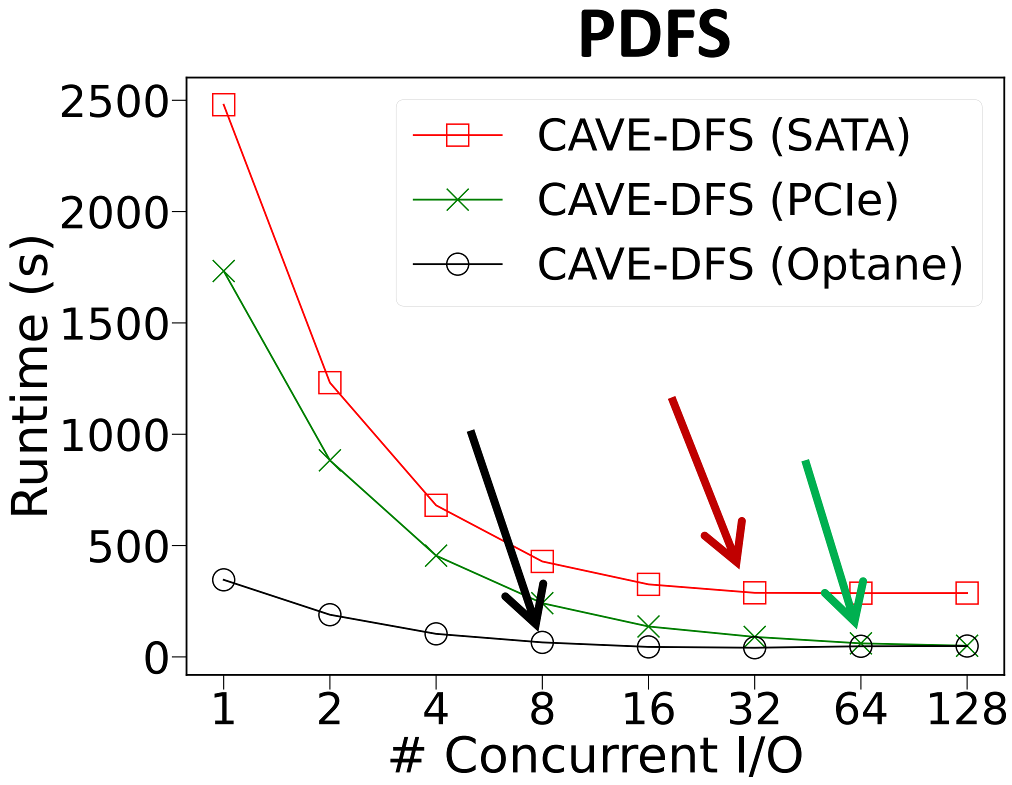
Dataset: FS

CAVE’s Speedup

Both CAVE implementations outperforms GridGraph, Mosaic and GraphChi

CAVE Utilizes Concurrent I/O

Dataset: FS



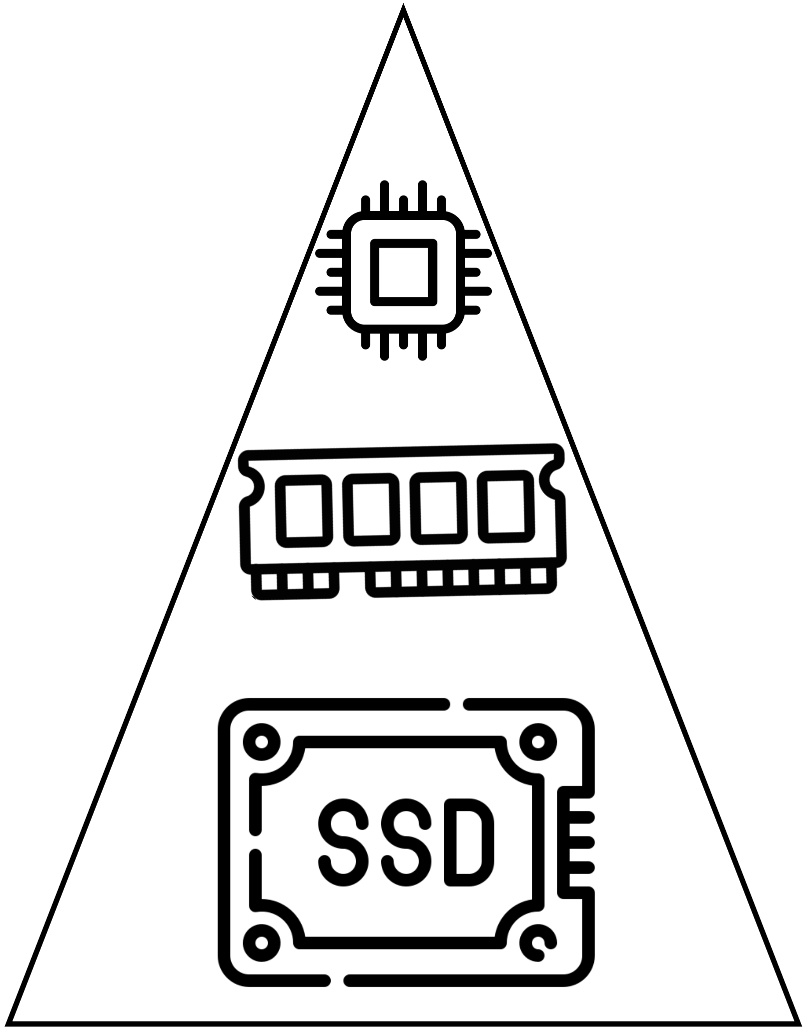
SATA SSD ($k_r = 25$)

PCIe SSD ($k_r = 80$)

Optane SSD ($k_r = 6$)

Device gets saturated at *optimal concurrency*

Goal: Developing Hardware-Aware Data Systems



Learned CPU Embedding

Silhouette [**mlforsys@NeuRIPS** ’23]

Minimize Data Movement through Memory Hierarchy

Relational Memory [**EDBT** ’23 +
Demo: **VLDB** ’23] ❖ Best Demo
Relational Fabric [**IEEE ICDE** ’23]
RelFeb ext. [Under Revision@**TKDE**]

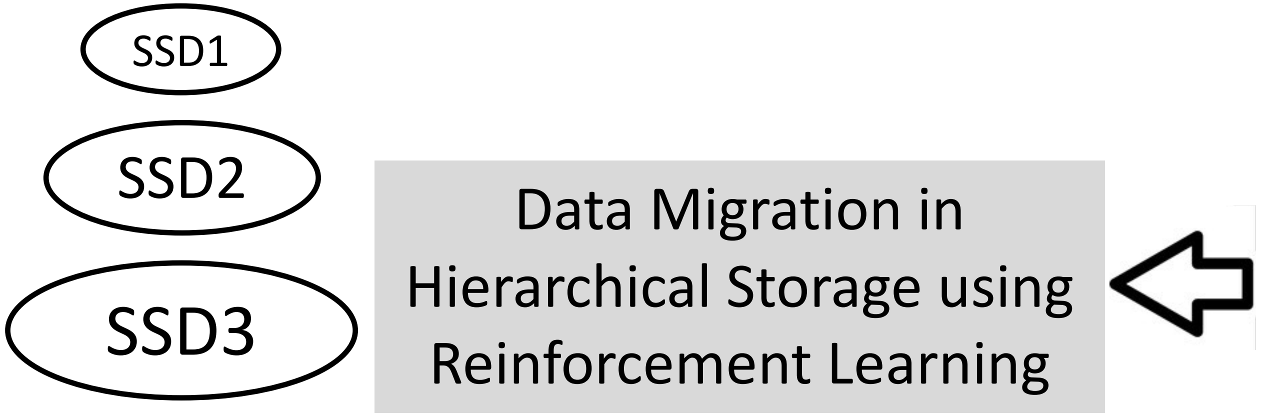
Timely Deletion in LSM Storage Layout

LETHE [**SIGMOD** ’20 + **ACM TODS** ’23]
Delete-Compliance [**IEEE DEBULL** ’22]

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]

Future Work



SSD1

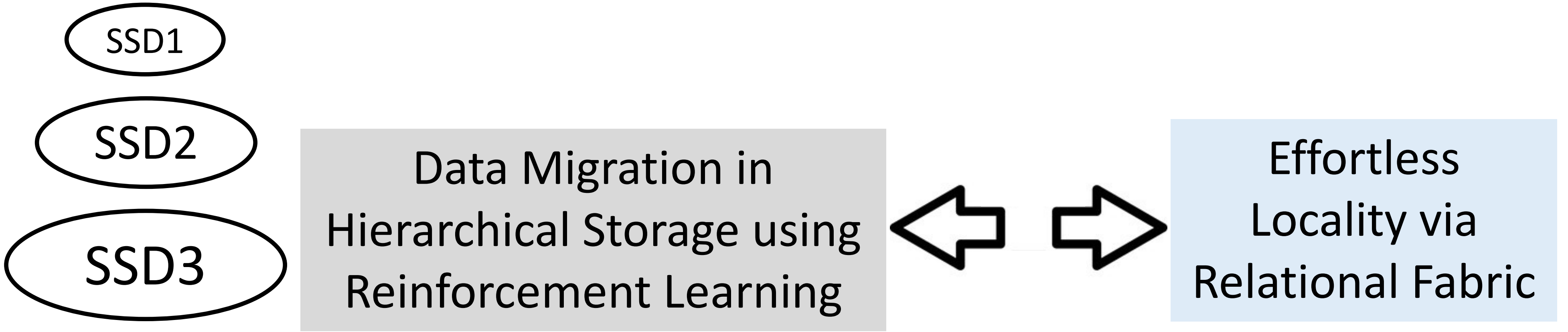
SSD2

SSD3

Data Migration in Hierarchical Storage using Reinforcement Learning

- How can $\boldsymbol{\alpha/k}$ help the agent?
- How to handle **deduplication**?
- How to ensure **data consistency**?

Future Work



SSD1

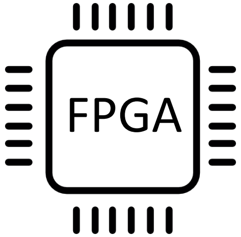
SSD2

SSD3

Data Migration in Hierarchical Storage using Reinforcement Learning

Effortless Locality via Relational Fabric

- How can $\boldsymbol{\alpha/k}$ help the agent?
- How to handle **deduplication**?
- How to ensure **data consistency**?

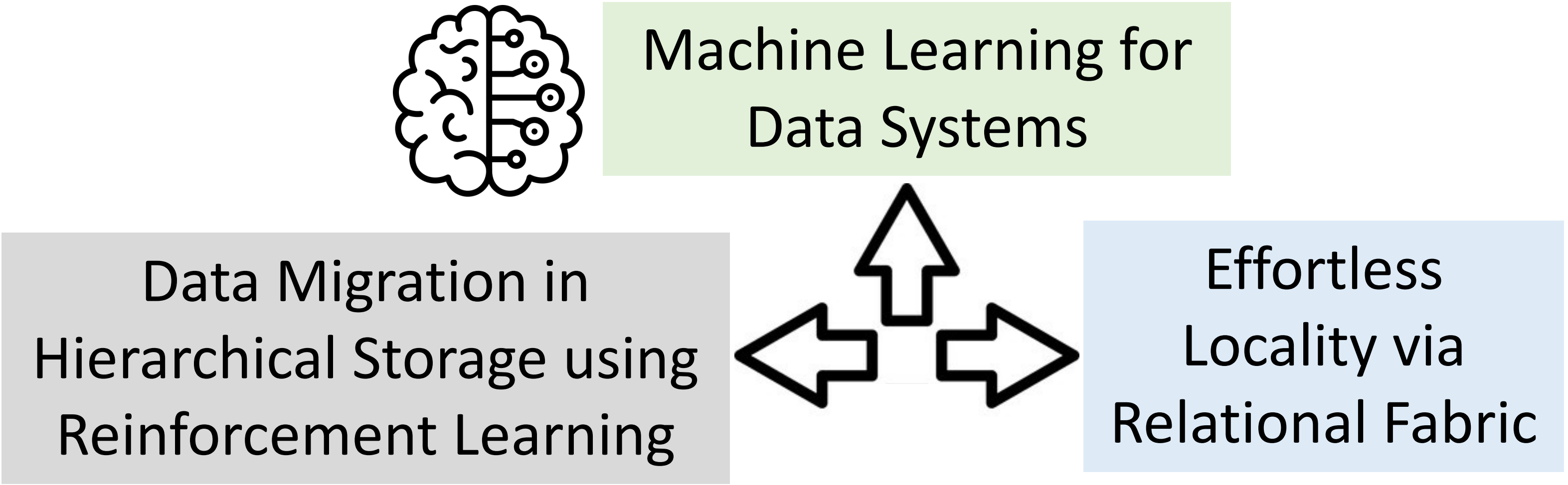


FPGA

- How to handle **updates**?
- How to do **compression**?
- **Impact** on DB architecture
- Leverage computational SSDs to build '**Relational Storage**'

Future Work

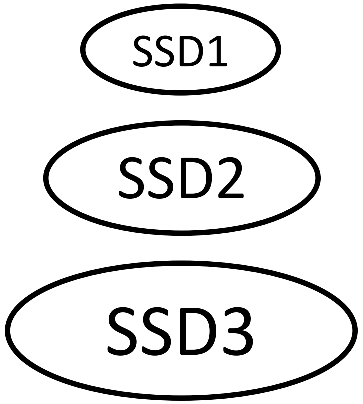
- How can ML models contribute to query optimization?
- Learned indexes & Learned database tuning
- ML techniques to optimize energy consumption in data centers



Machine Learning for Data Systems

Data Migration in Hierarchical Storage using Reinforcement Learning

Effortless Locality via Relational Fabric



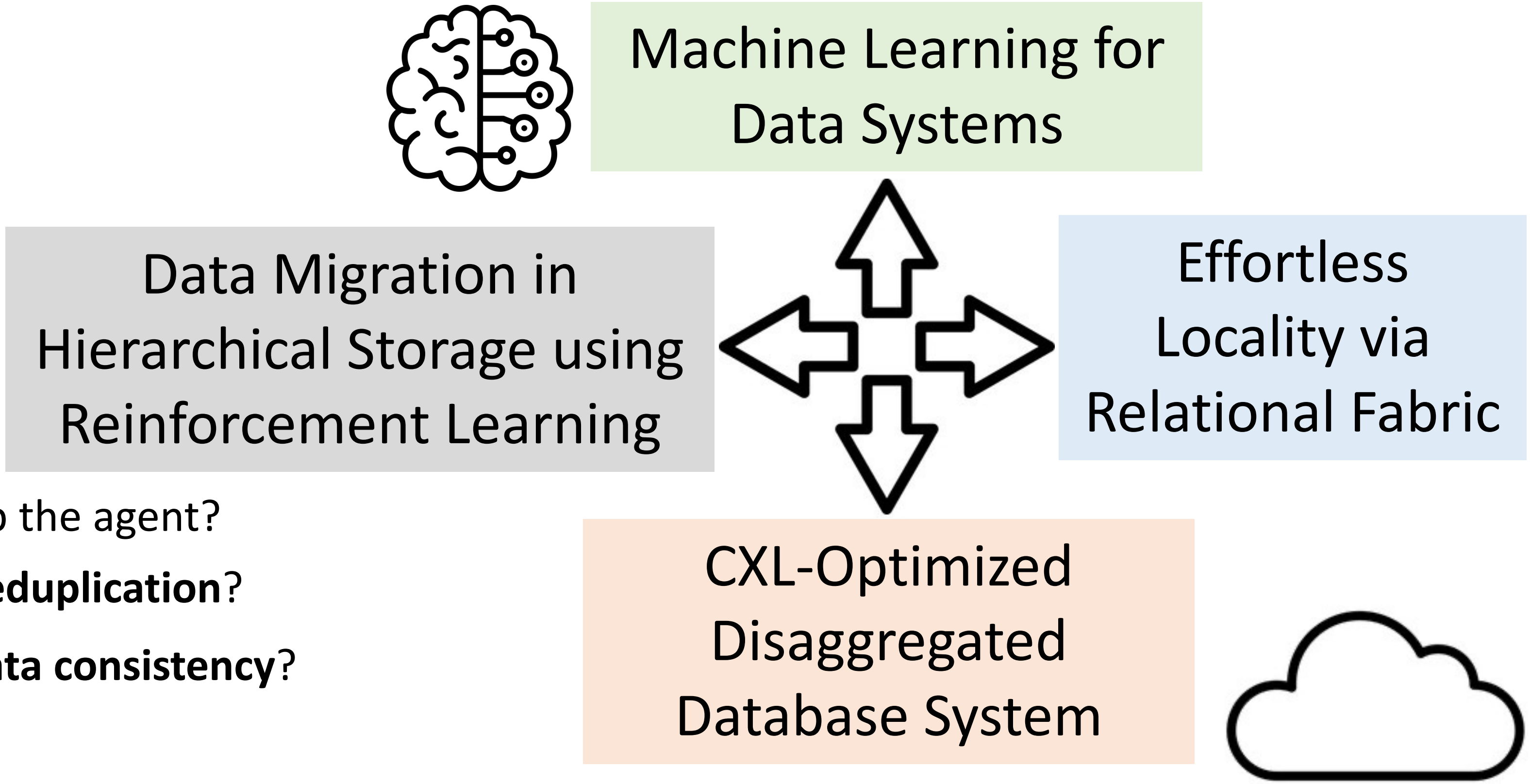
- How can α/k help the agent?
- How to handle **deduplication**?
- How to ensure **data consistency**?

FPGA

- How to handle **updates**?
- How to do **compression**?
- **Impact** on DB architecture
- Leverage computational SSDs to build '**Relational Storage**'

Future Work

- How can ML models contribute to query optimization?
- Learned indexes & Learned database tuning
- ML techniques to optimize energy consumption in data centers



Machine Learning for Data Systems

Data Migration in Hierarchical Storage using Reinforcement Learning

Effortless Locality via Relational Fabric

CXL-Optimized Disaggregated Database System

FPGA

- How to handle **updates**?
- How to do **compression**?
- **Impact** on DB architecture
- Leverage computational SSDs to build ‘**Relational Storage**’

SSD1

SSD2

SSD3

- How can $\boldsymbol{\alpha/k}$ help the agent?
- How to handle **deduplication**?
- How to ensure **data consistency**?

- Can we ensure **scalable transactions & reliability** in disaggregated databases?
- How to manage storage and memory efficiently via **automatic resource provisioning**?
- How to ensure **compatibility** across generations?

Thank You!

Tarikul Islam Papon

PhD Researcher

BOSTON
UNIVERSITY

cs-people.bu.edu/papon/