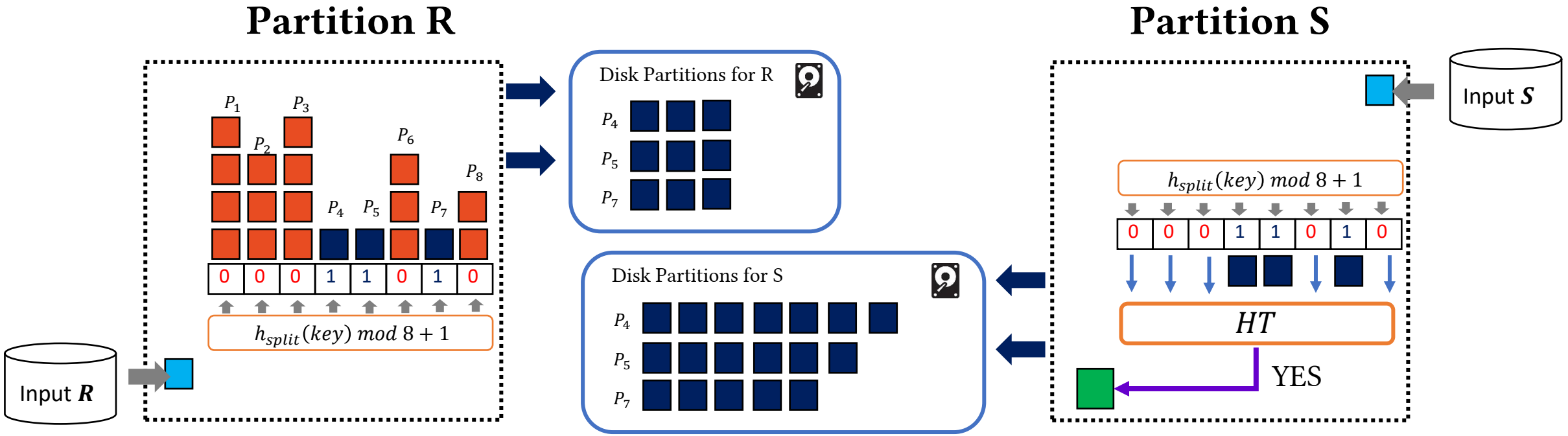
NOCAP: Near-Optimal Correlation-Aware Partitioning for Joins

Zichen Zhu Xiao Hu Manos Athanassoulis

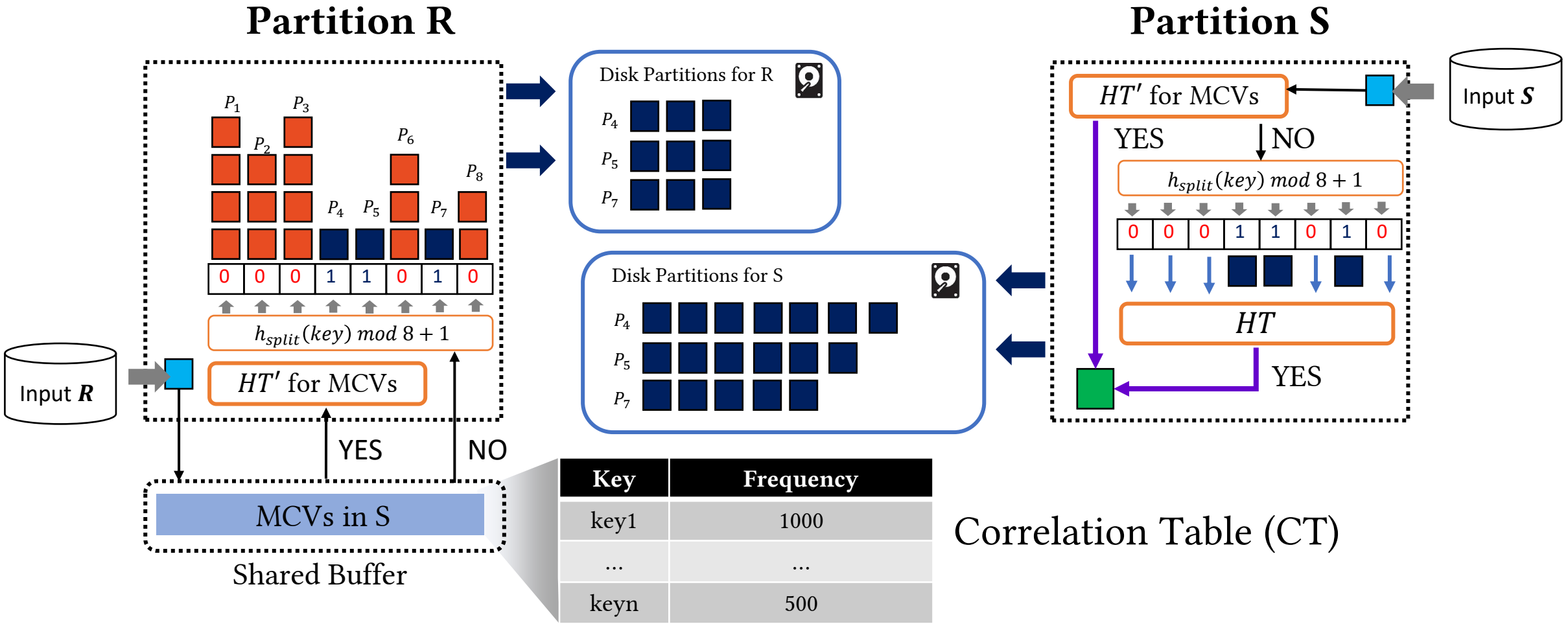
Dynamic Hybrid Hash Join (DHH)

State of the art DBs (e.g., PostgreSQL and AsterixDB) employ DHH to implement storage-based equii-joins.



Dynamic Hybrid Hash Join (DHH)

State of the art DBs (e.g., PostgreSQL and AsterixDB) employ DHH to implement storage-based equii-joins.

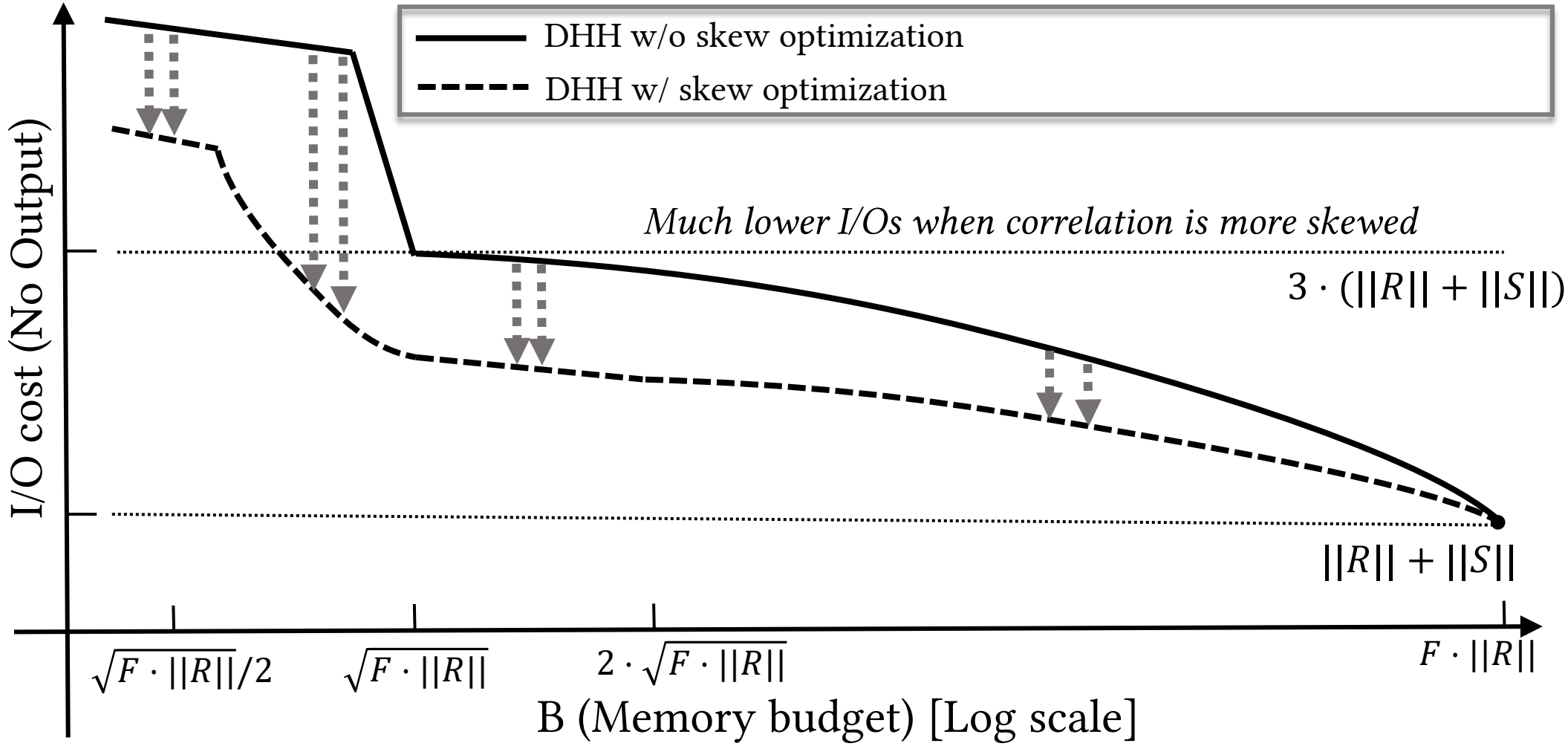


Key	Frequency
key1	1000
...	...
keyn	500

Correlation Table (CT)

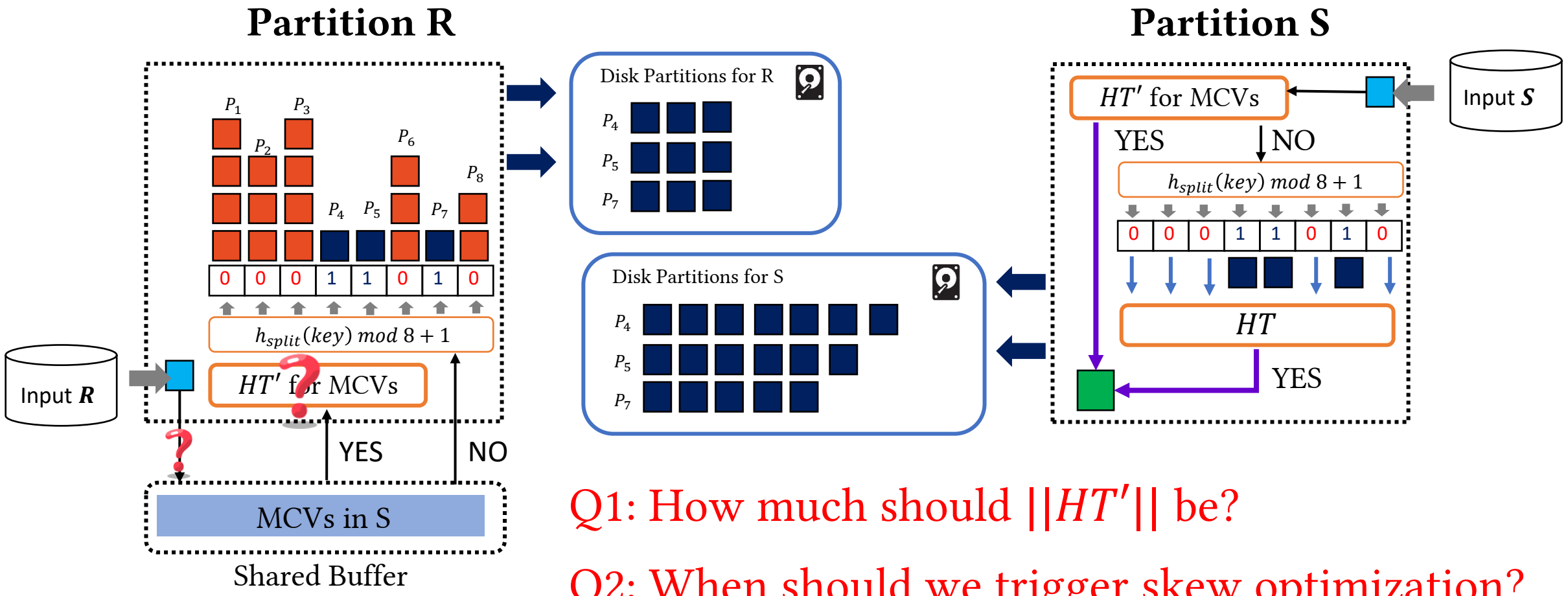
Skew Optimization

Skew Optimization in DHH



Skew optimization reduces the number of I/Os when the matching exhibits skew

Problems



Q1: How much should $||HT'||$ be?

Q2: When should we trigger skew optimization?

OCAP (Optimal Correlation-Aware Partitioning)

$$arg \min_{P,m} \sum_{j=2}^{m+1} Cost(||R_j||, ||S_j||)$$

s.t. $\forall i \in [n],\ \sum_{j=1}^{m+1} P_{i,j} = 1$

$$||R_1|| + m + 2 \leq B$$

$$P_{i,j} \in \{0,1\}, \forall i \in [n], \forall j \in [m+1]$$

Define an $n \times (m+1)$ Boolean matrix P to represent the partitioning assignment

Notation	Meaning
$n\ (n_R)$	The number of tuples in relation R
m	The number of partitions on disk
$P = \begin{bmatrix} 0 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 0 \end{bmatrix}_{n \times (m+1)}$	A Boolean matrix P where $P_{i,j} = 1$ represents the i^{th} record belongs to the j^{th} partition
R_1	A partition cached in memory
$Cost(\|\|R_j\|\|, \|\|S_j\|\|)$	The write cost of R_j, S_j plus the join cost between R_j, S_j

$$||R_j|| = \sum_{i=1}^{n} P_{i,j} / b_R$$

$$||S_j|| = \sum_{i=1}^{n} P_{i,j} \cdot CT[i] / b_S$$

Correlation Table (CT)

Index i	Frequency in S
1	1
…	…
n-1	77
n	100

Exponential searching space !

$O(n^2 \cdot \log B / B)$ *for PK-FK joins*

Practical Challenges for OCAP

1. *We cannot have the whole CT in practice*

Index i	Frequency in S
1	1
…	…
10M	1000

2. *Partitioning assignment also occupies memory*

$$P = \begin{bmatrix} 0 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 0 \end{bmatrix}_{n \times (m+1)}$$

Observations from OCAP

O1: MCVs can be prioritized in *two* ways: build an in-memory hash table (if B is large) or **assign them into a small partition on disk (if B is small)**

O2: We should ensure on-disk partitions fully fill $z \cdot (B-2)$ pages ($z \in Z^+$)

Divisibility when Partitioning Data on Disk

uniform

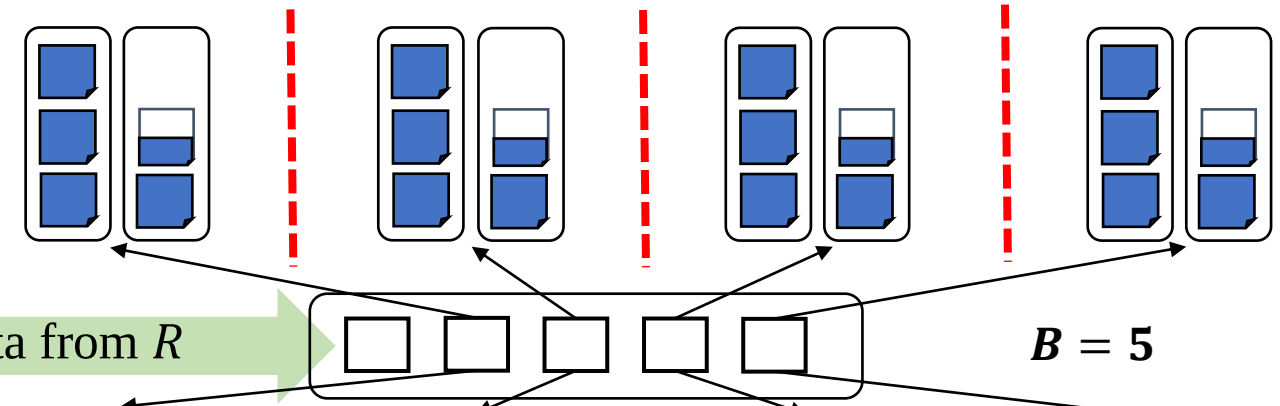
$PartID = h(k)\ mod\ 4$

$||S_1|| = 9$ $||S_2|| = 10$ $||S_3|| = 10$ $||S_4|| = 8$

Partition 1 Partition 2 Partition 3 Partition 4

$(9 + 10 + 10 + 8) \cdot 2$

$= \mathbf{74}$



18-page input data from R

$\boldsymbol{B = 5}$

$PartID = (h(k)\ mod\ 6)\ mod\ 4$

non-uniform

Partition 1 Partition 2 Partition 3 Partition 4

$6 = ||R|| / (B - 2)$

$||S_1|| = 12$ $||S_2|| = 13$ $||S_3|| = 7$ $||S_4|| = 5$

$(12 + 13) \cdot 2 + 7 + 5$

$= \mathbf{62}$

Prioritizing MCVs with Constrained Memory

Index i	Frequency in S
1	1
...	...
n-k+1	73
n-k+2	73
...	...
n-1	100
n	100

We only have top-k frequent keys (MCVs)

Run augmented DHH with memory budget m_{rest}

Run OCAP for top-k' MCVs with memory budget $m = B - 2 - m_{rest}$

A hash set for keys that are staged in memory — HS

A hash table to store the whole record of the hash set — HT

A hash map for keys to be partitioned — f

Several write-buffer pages for partitioning — m_{disk}

```
minCost ← ∞, HS ← ∅, f ← ∅
for k' ∈ [0, k]
    for m ∈ [0, B − 2]
        HS, f ← OCAP(k', m, MCV)
        if the estimated I/O cost is smaller
            update minCost, HS, f
return HS, f
```

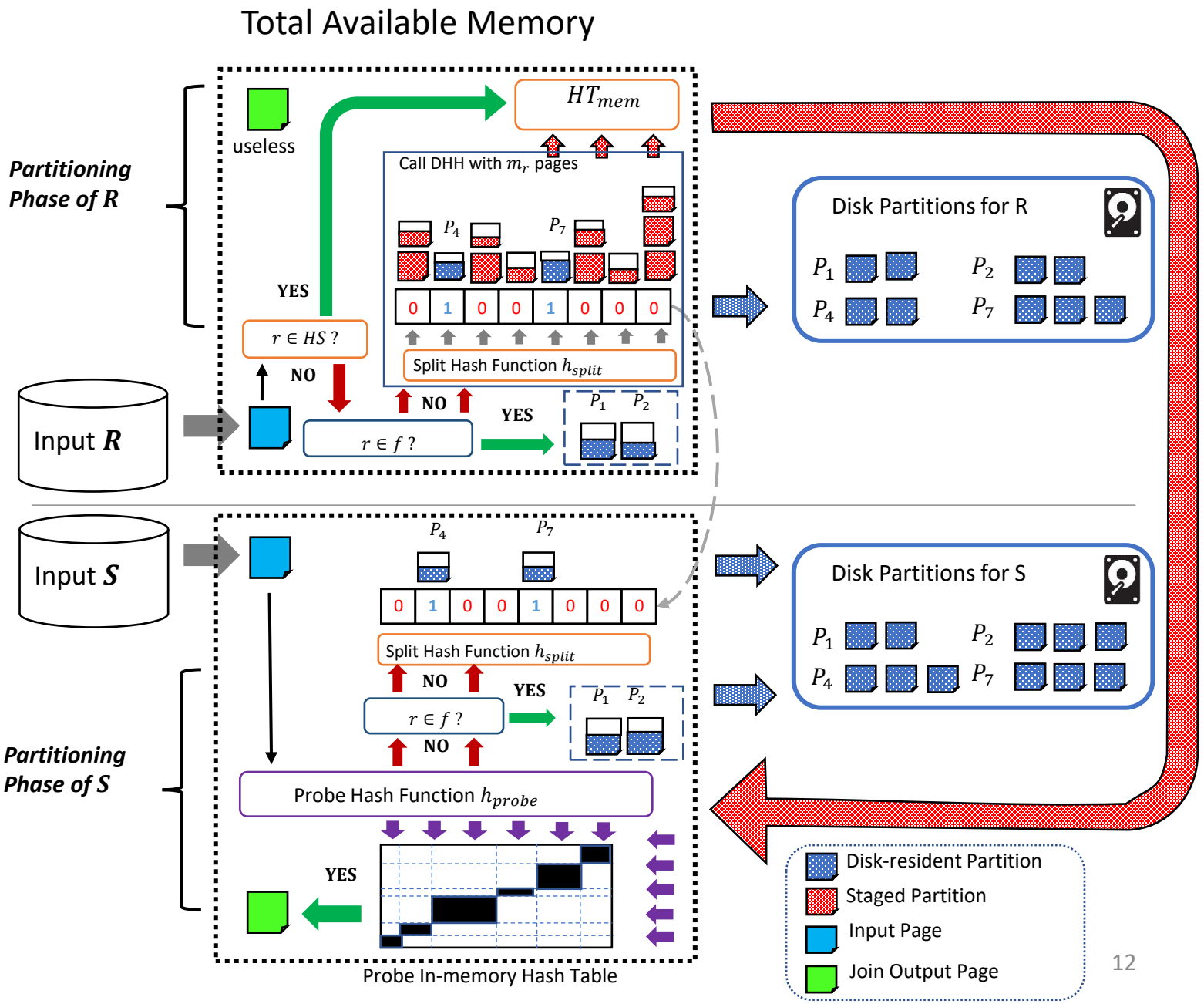
Only takes ≈ 1 second for 50K MCVs

NOCAP

Partitioning Workflow:

OCAP *for top-k′ frequent keys*

DHH *to partition the rest*



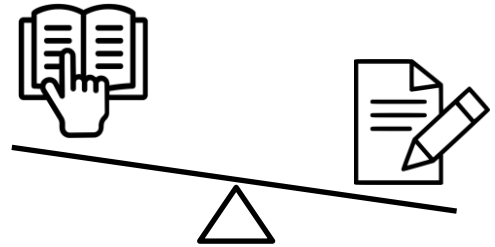
Experiment Setup

Storage: PCIe NVM SSD (15 μs for reading a 4KB page)

Measured read/write symmetry:

random_write_latency/sequential_read_latency = 3.3

sequential_write_latency/sequential_read_latency = 3.2

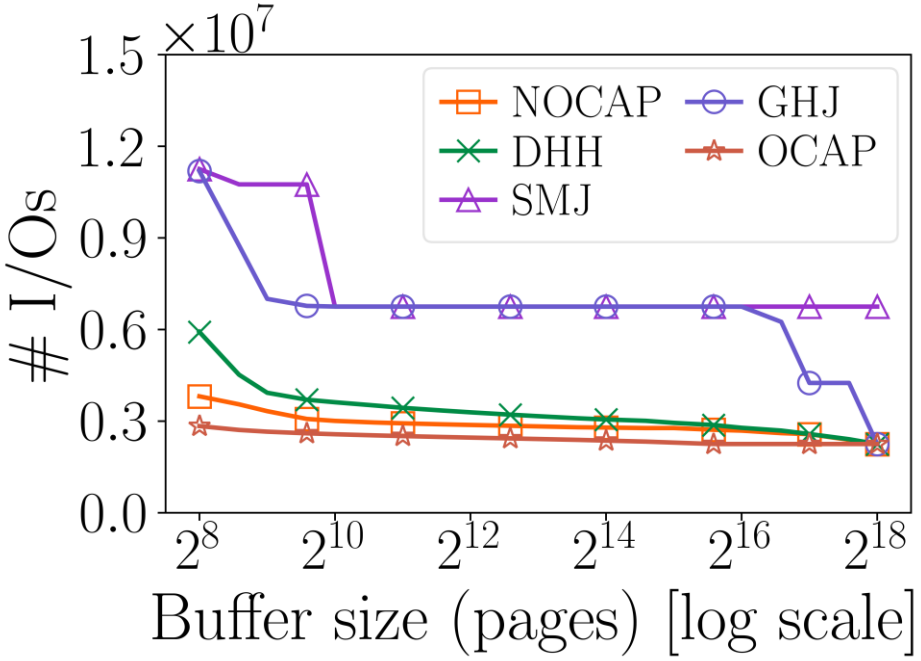


PK-FK join input size: 1M #records join with 8M #records

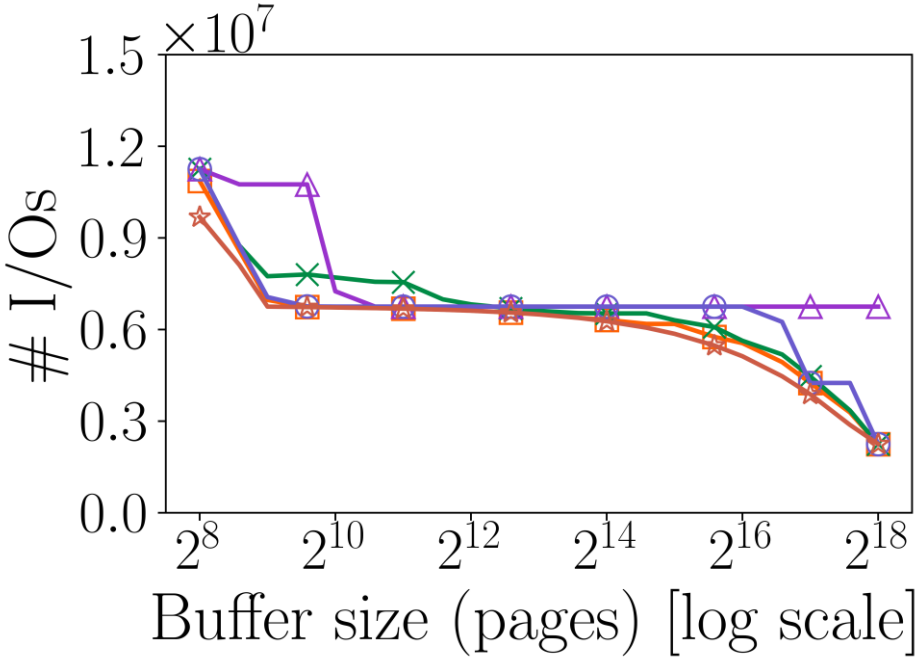
Record size: 1KB per record

Page size: 4KB

Selected Experimental Results



Zipfian ($\alpha = 1.3$)

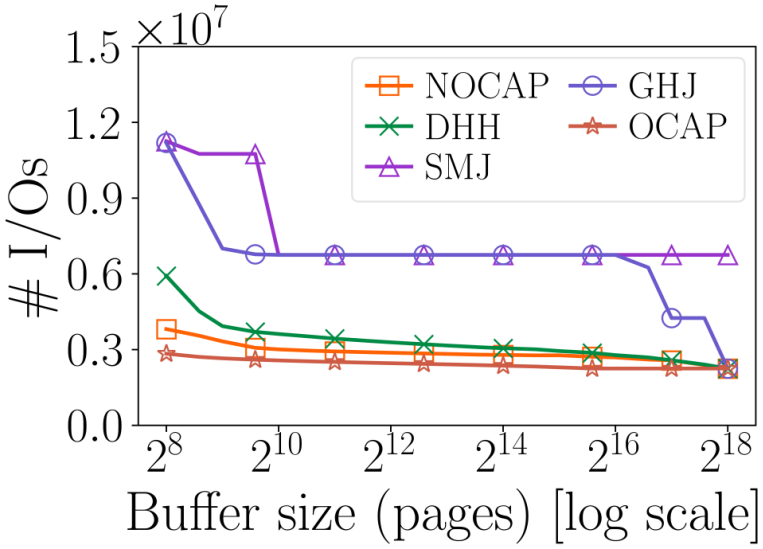


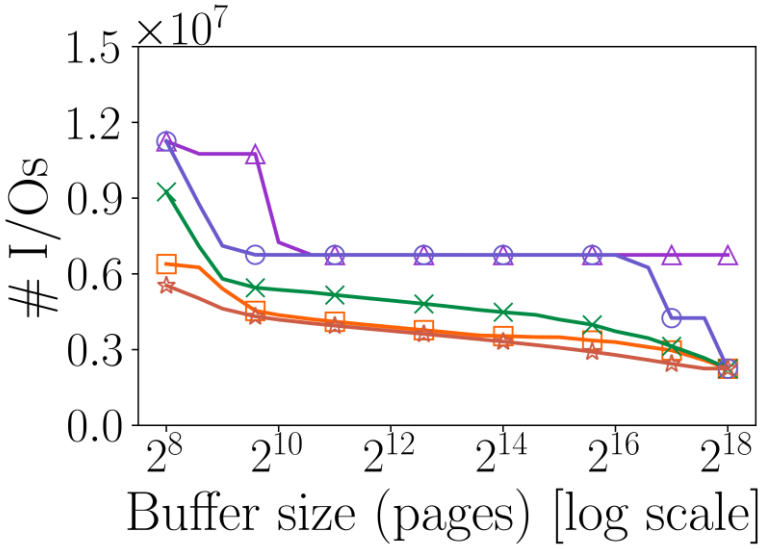
Uniform

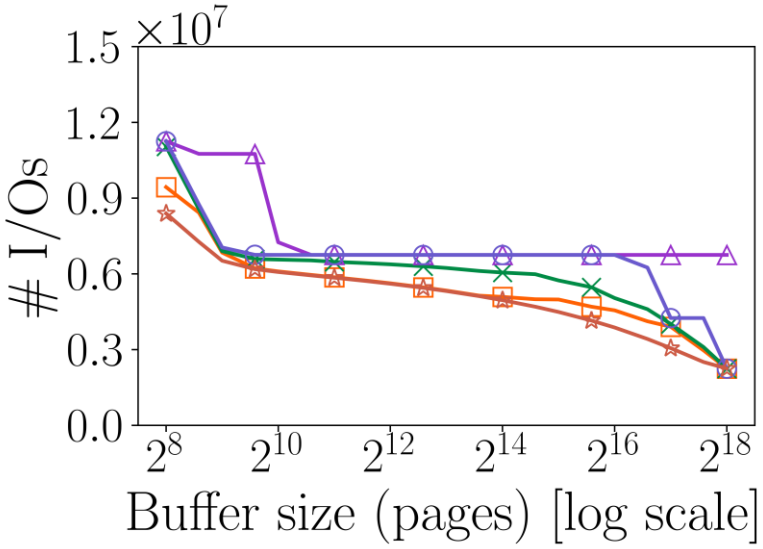
Correlation-aware joins (**DHH and NOCAP**) can **adaptively** reduce I/O cost when it comes to a **skew** distribution.

Note: OCAP only represents a lower bound, not a practical algorithm

Varying skew







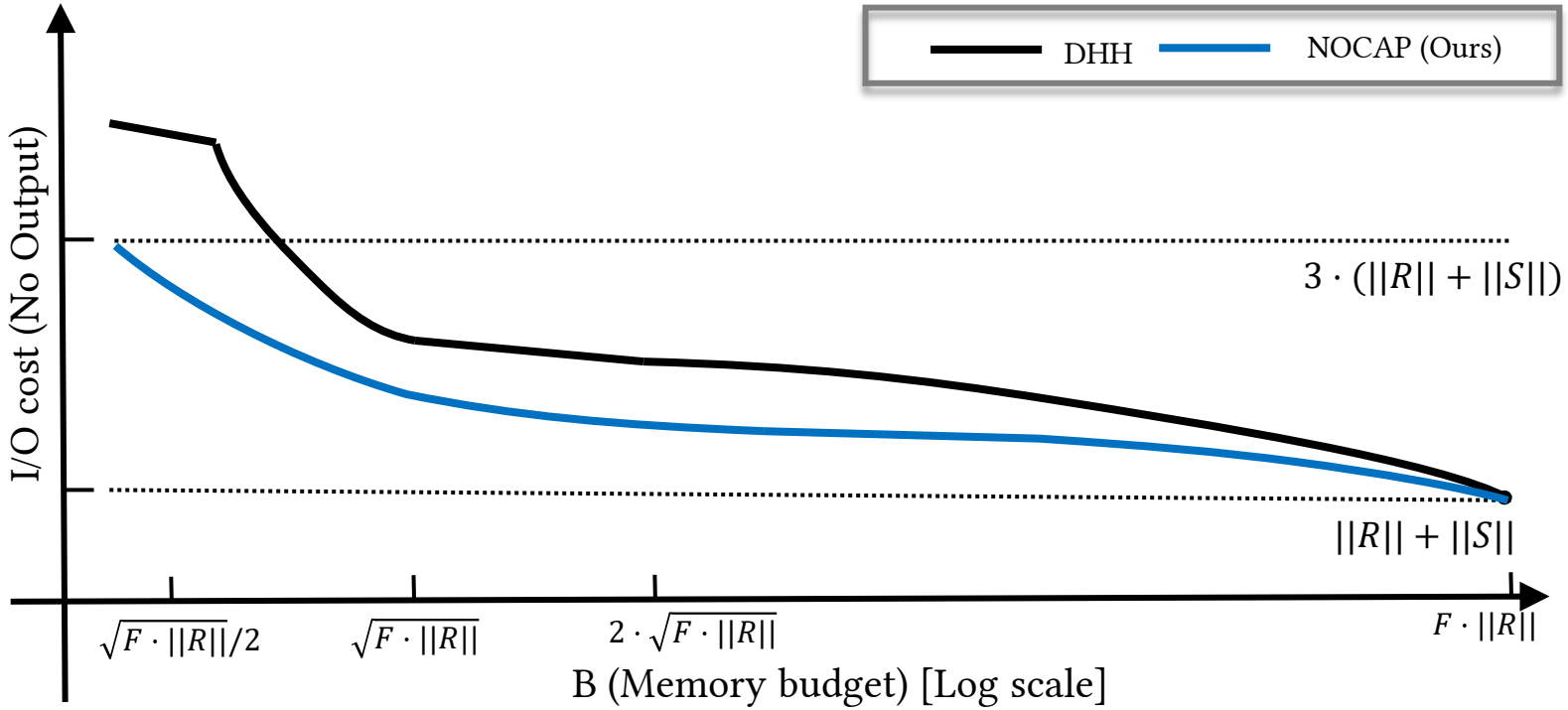
Zipfian ($\alpha = 1.3$) Zipfian ($\alpha = 1.0$) Zipfian ($\alpha = 0.7$)

High skew *Low skew*

While DHH helps reduce #I/Os, **NOCAP** can better exploit the correlation skew to **achieve even lower I/O cost**.

Summary of NOCAP

NOCAP join outperforms DHH by up to 30%, and the textbook GHJ by up to 4X. Even for uniform distribution, NOCAP outperforms DHH by up to 10%!



Thanks

Q&A