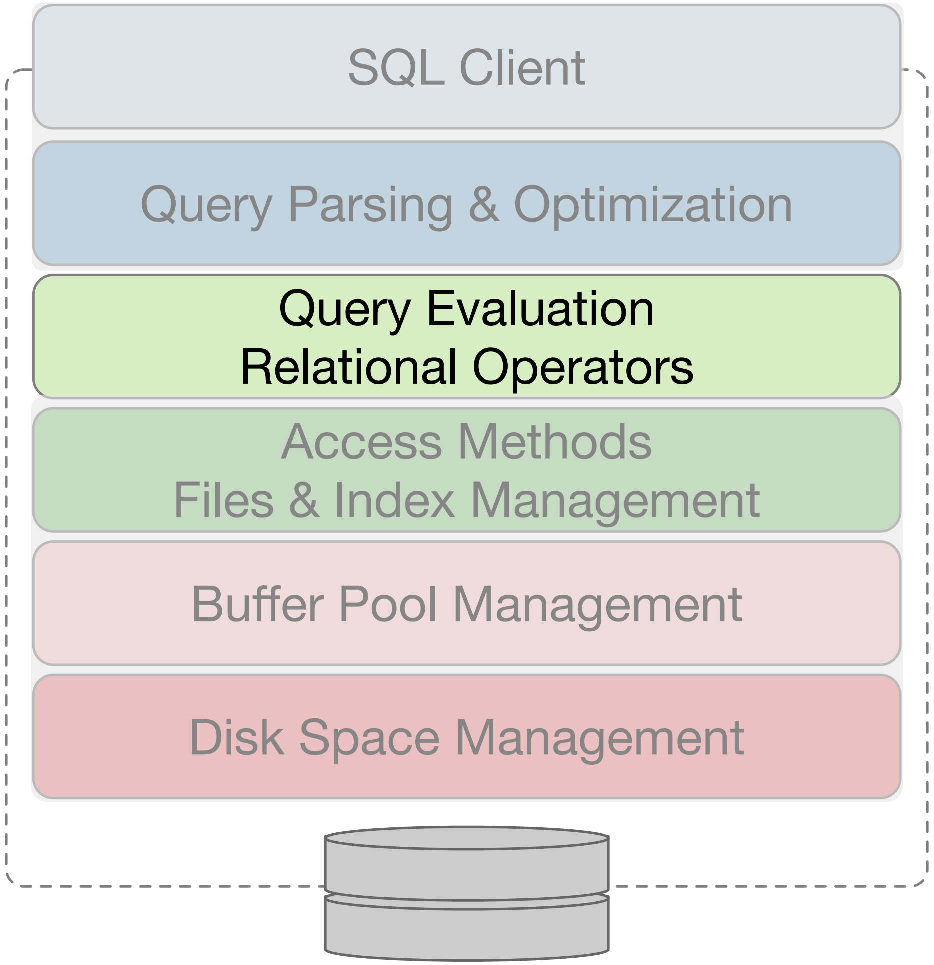
Sorting & Hashing

Where we are right now?



SQL Client

Query Parsing & Optimization

Query Evaluation
Relational Operators

Access Methods
Files & Index Management

Buffer Pool Management

Disk Space Management

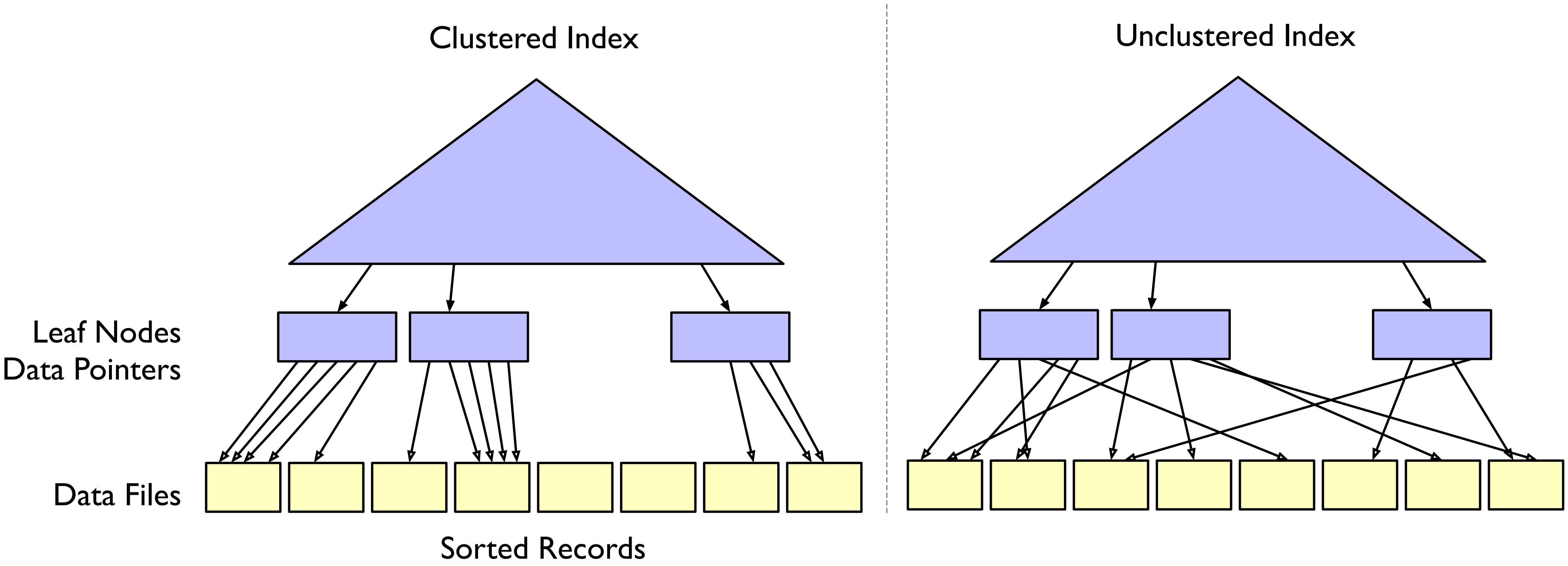
Building blocks: Sorting & Hashing

Why are they building blocks?

- Rendezvous
  - Duplicate elimination:
    ```
    select DISTINCT a
    ```
  - Join processing
    ```
    select R JOIN S on a
    ```
  - Grouping & Aggregations:
    ```
    select SUM(a) from R GROUP BY b
    ```
- Ordering (Sorting)
  - Ordered Result
    ```
    select * from R ORDER BY A
    ```
  - Bulk Loading

Why are they special in a DBMS?

- Well-studied in-memory algorithms!
  - Sorting: Quick-, Merge-, Radix-, …
  - Hashing
- *But … tables don't fit in memory*
  - Can't rely on virtual memory
  - Disk-oriented – minimize IO
  - Prefer sequential IOs

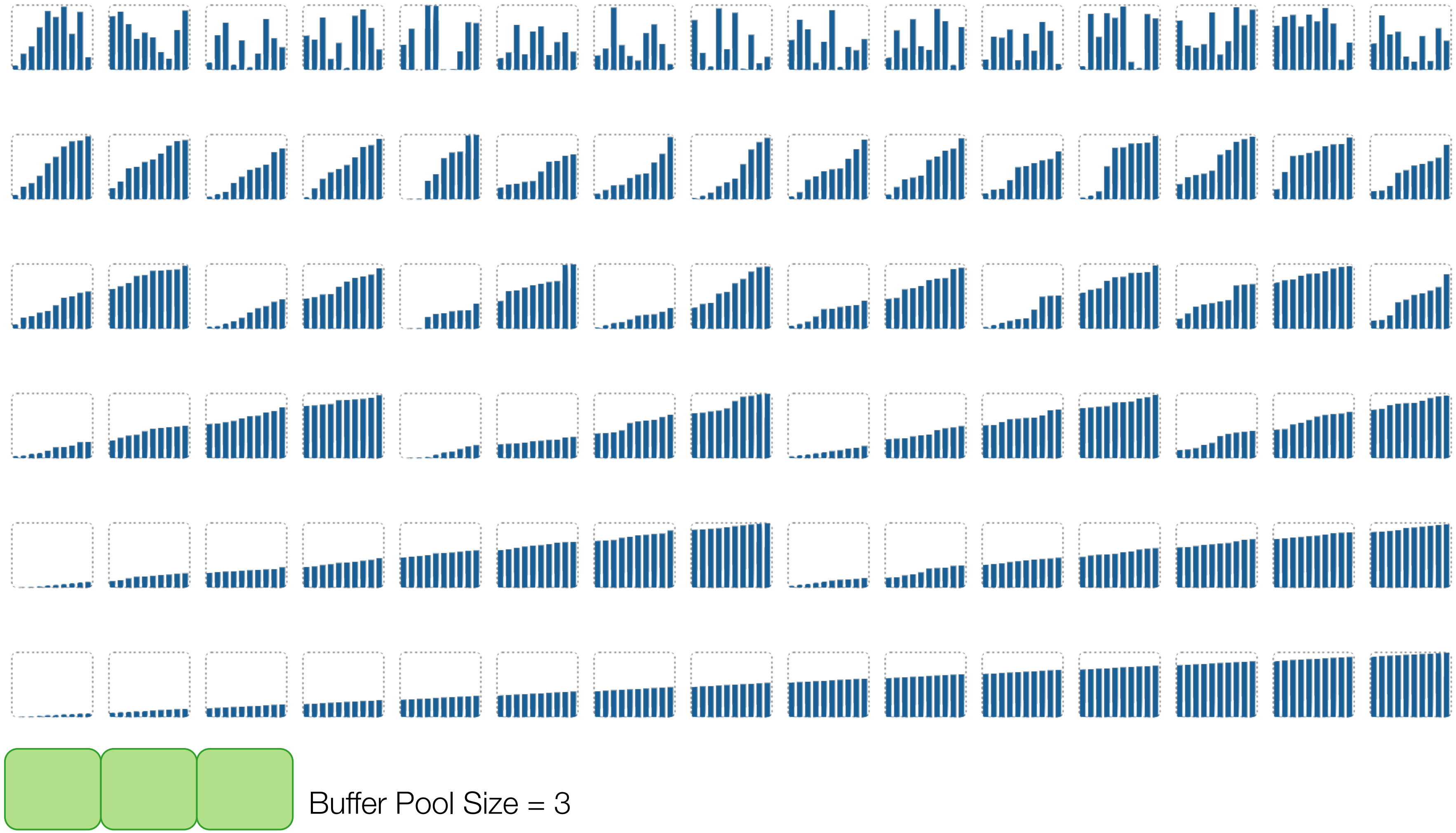


What if we have an index?
Can we use it for sorting?

It depends!

2-Way Merge Sort

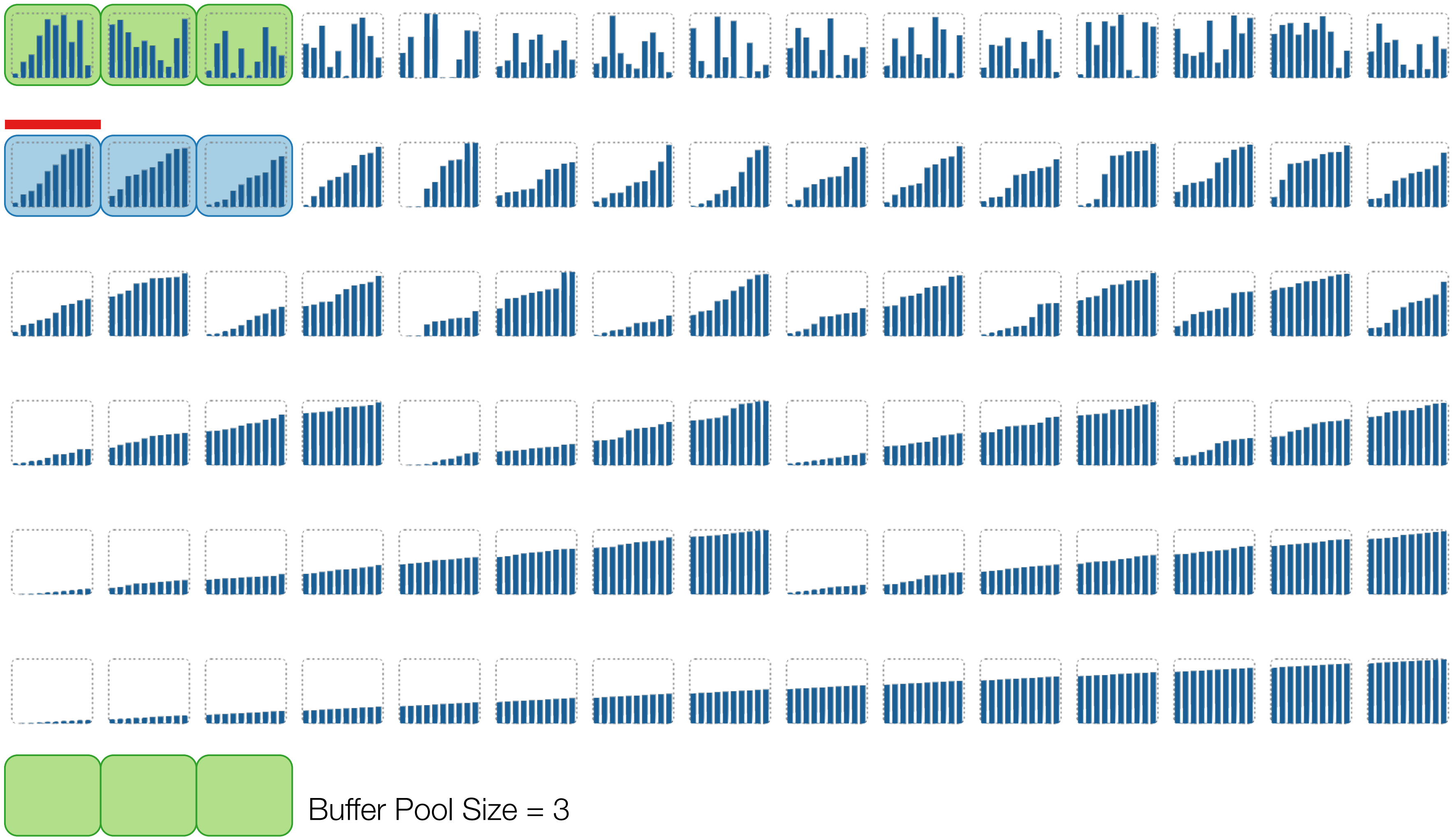
2-Way External Merge Sort Trace



Unsorted file on disk
N = 15 pages

Buffer Pool Size = 3

2-Way External Merge Sort Trace

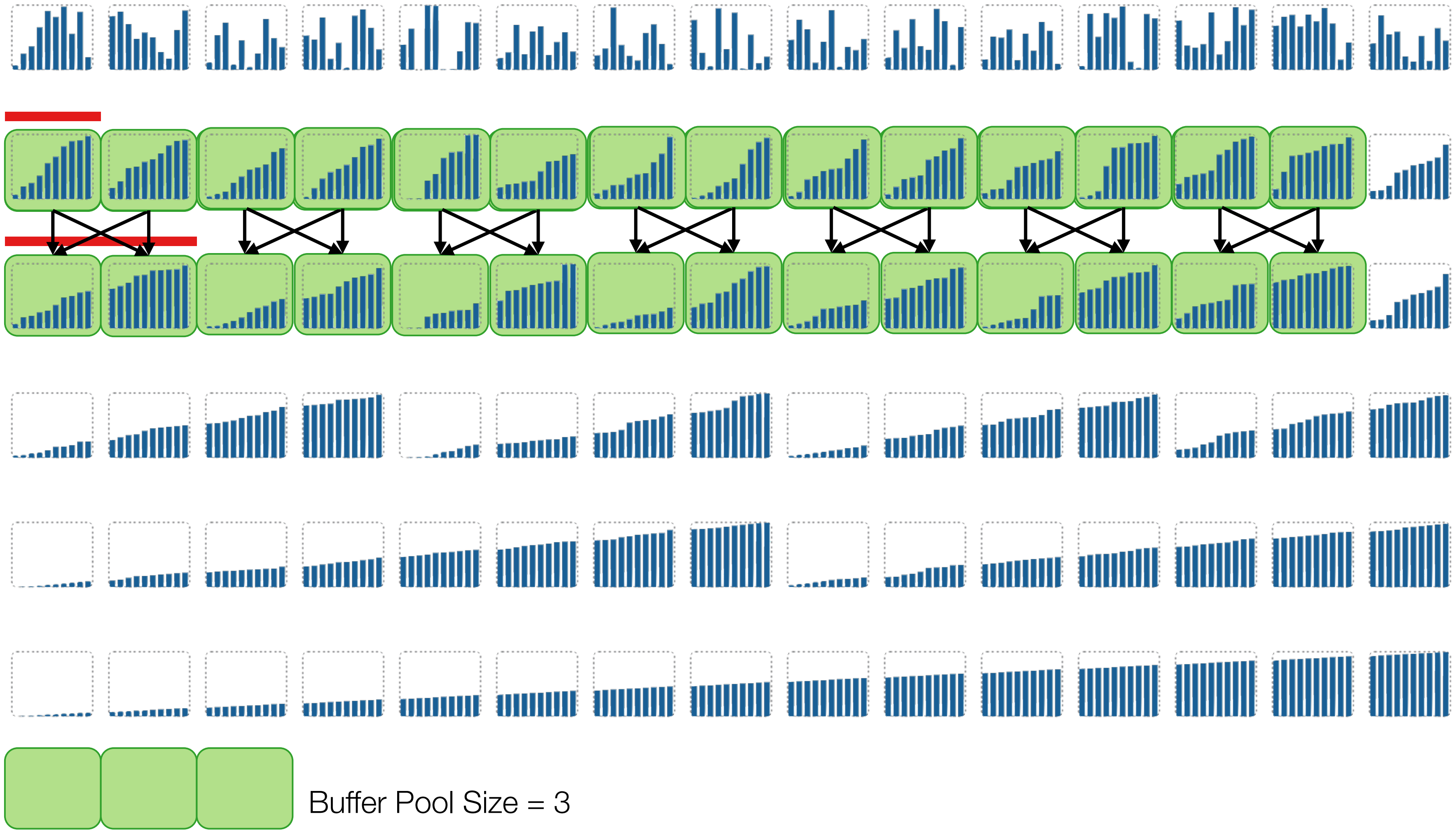


Unsorted file on disk
N = 15 pages

Pass 0 – Streaming Pass
15 sorted runs of length 1

Buffer Pool Size = 3

2-Way External Merge Sort Trace



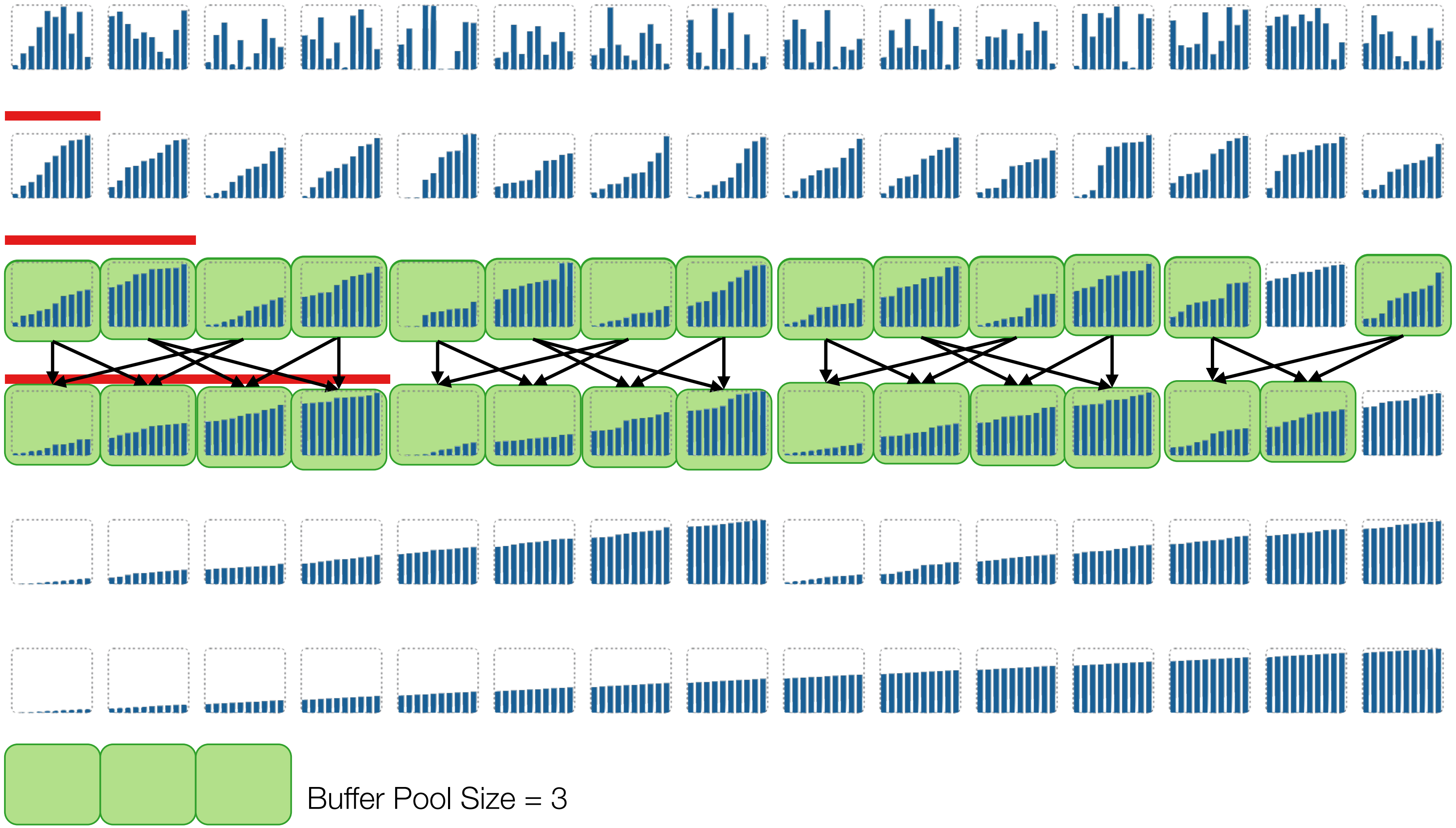
Unsorted file on disk
N = 15 pages

Pass 0 – Streaming Pass
15 sorted runs of length 1

Pass 1 – Merge Pass
8 sorted runs of length 2

Buffer Pool Size = 3

2-Way External Merge Sort Trace



Unsorted file on disk
N = 15 pages

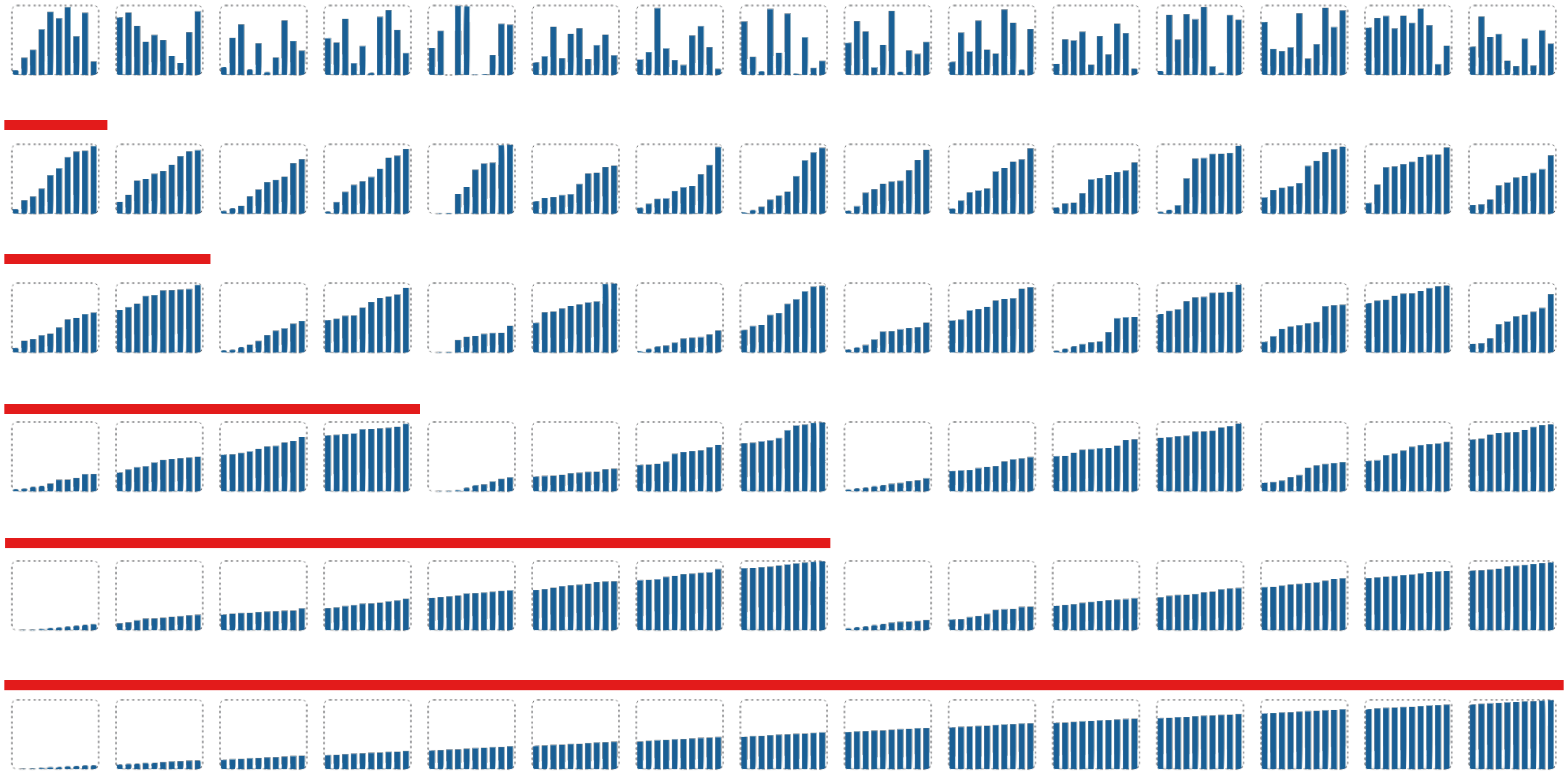
Pass 0 – Streaming Pass
15 sorted runs of length 1

Pass 1 – Merge Pass
8 sorted runs of length 2

Pass 2 – Merge Pass
4 sorted runs of length 4

Buffer Pool Size = 3

2-Way External Merge Sort Trace



Unsorted file on disk
N = 15 pages

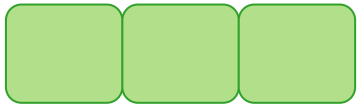
Pass 0 – Streaming Pass
15 sorted runs of length 1

Pass 1 – Merge Pass
8 sorted runs of length 2

Pass 2 – Merge Pass
4 sorted runs of length 4

Pass 3 – Merge Pass
2 sorted runs of length 8

Pass 4 – Merge Pass
1 sorted run of length 15

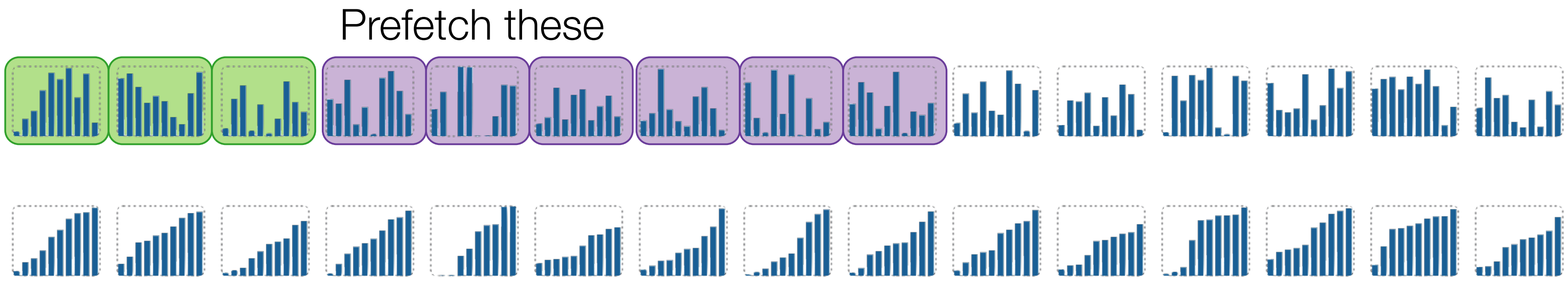


Buffer Pool Size = 3

Each pass costs 2N IOs
Read the file + Sort + Write the file

There are $\lceil \log_2 N \rceil + 1$ passes

Total Cost: $2N \times (\lceil \log_2 N \rceil + 1)$ IOs



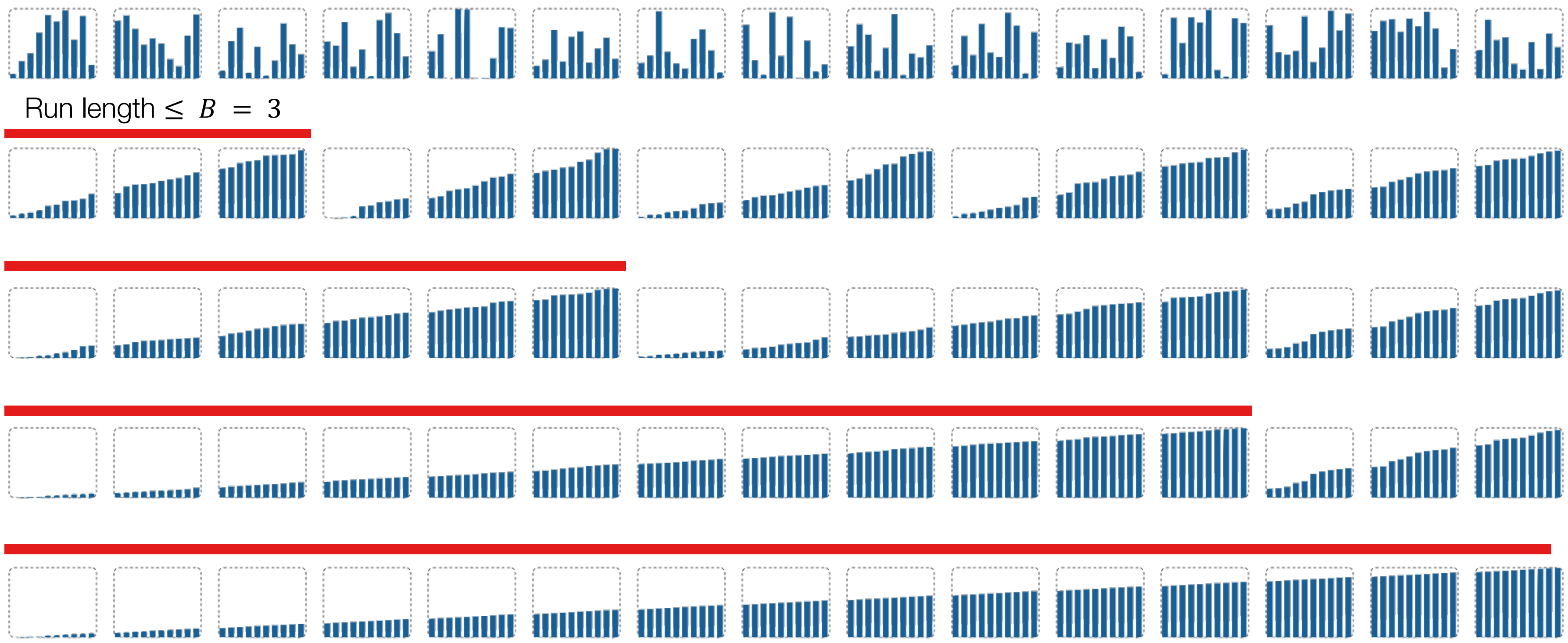
Double Buffering

- Prefetch the next run in the background while the system is processing this run.
- Reduces the wait time for IO requests.
- Requires support for asynchronous IO, multi-threading: the buffer manager brings in the next run while the sorting thread processes the pages currently in the buffer.

What if B > 3 buffer pages?

K-Way Merge Sort

K-Way External Merge Sort Trace: Optimizing the Streaming Phase



Unsorted file on disk
N = 15 pages

Pass 0 – Streaming Pass
5 sorted runs of length 3

Pass 1 – Merge Pass
3 sorted runs of length 6

Pass 2 – Merge Pass
2 sorted runs of length 12

Pass 3 – Merge Pass
1 sorted runs of length 15

Buffer Pool Size = 3

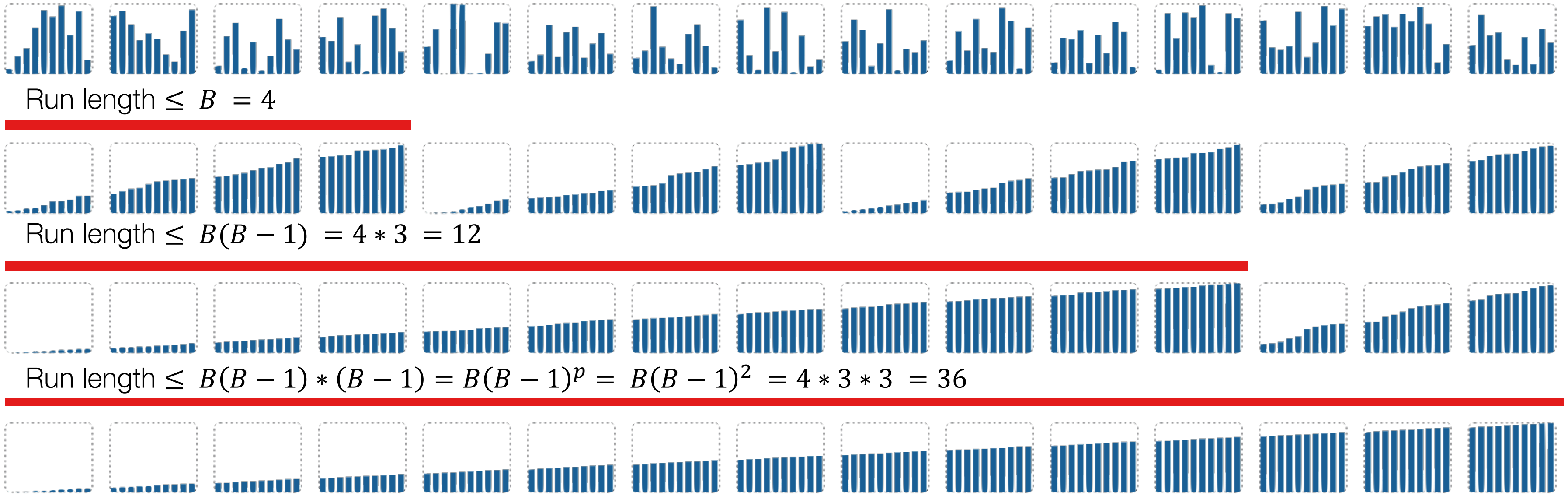
Each pass costs 2N IOs
Read the file + Sort + Write the file

There are $\cancel{\lceil \log_2 N \rceil} + 1$ passes

There are $\lceil \log_2 \lceil N/B \rceil \rceil + 1$ passes

Total Cost: $2N \times (\lceil \log_2 \lceil N/B \rceil \rceil + 1)$ IOs

K-Way External Merge Sort Trace: Making use of more buffers!



Run length ≤ $B = 4$

Run length ≤ $B(B-1) = 4*3 = 12$

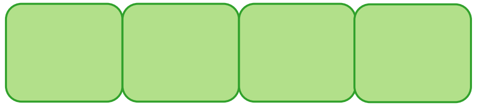
Run length ≤ $B(B-1)*(B-1) = B(B-1)^p = B(B-1)^2 = 4*3*3 = 36$

Unsorted file on disk
N = 15 pages

Pass 0 – Streaming Pass
4 sorted runs of length 4

Pass 1 – Merge Pass
2 sorted runs of length 12

Pass 2 – Merge Pass
1 sorted run of length 15



Buffer Pool Size = 4

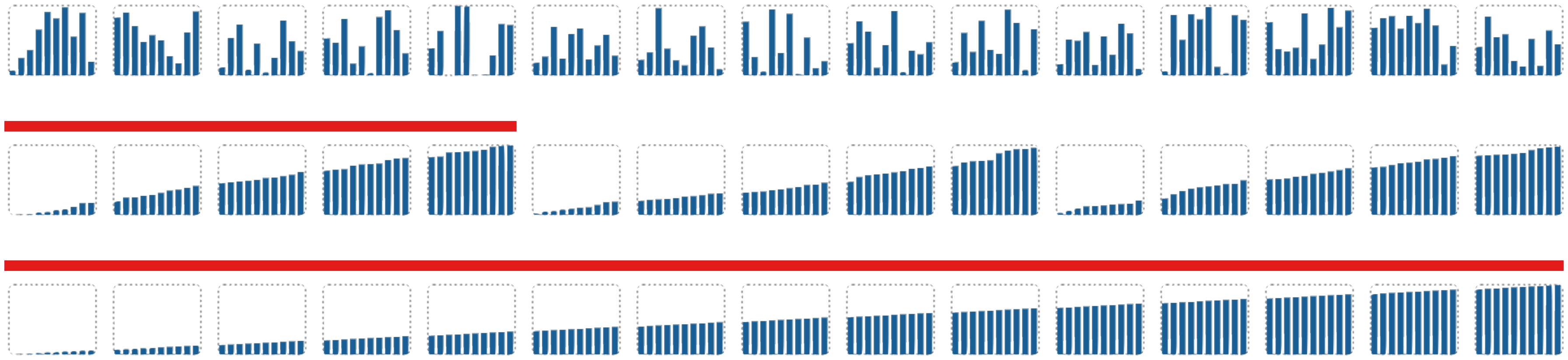
Each pass costs 2N IOs
Read the file + Sort + Write the file

There are ~~$\lceil \log_2 \lceil N/B \rceil \rceil + 1$~~ passes

There are $\lceil \log_{B-1} \lceil N/B \rceil \rceil + 1$ passes

Total Cost: $2N \times (\lceil \log_{B-1} \lceil N/B \rceil \rceil + 1)$ IOs

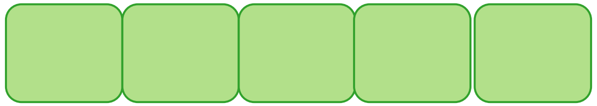
K-Way External Merge Sort Trace: Making use of more buffers!



Unsorted file on disk
N = 15 pages

Pass 0 – Streaming Pass
4 sorted runs of length 5

Pass 1 – Merge Pass
2 sorted runs of length 12



Buffer Pool
Size = 5

Each pass costs 2N IOs
Read the file + Sort + Write the file

There are $\lceil \log_{B-1} \lceil N/B \rceil \rceil + 1$ passes

Total Cost: $2N \times (\lceil \log_{B-1} \lceil N/B \rceil \rceil + 1)$ IOs

# of Passes of External Sort

# of pages in a file to sort (N)	Buffer Size *(B)*					
	3	5	9	17	129	257
100	7	4	3	2	1	1
1,000	10	5	4	3	2	2
10,000	13	7	5	4	2	2
100,000	17	9	6	5	3	3
1,000,000	20	10	7	5	3	3
10,000,000	23	12	8	6	4	3
100,000,000	26	14	9	7	4	4
1,000,000,000	30	15	10	8	5	4

2-Pass External Sort

- After the streaming pass, each sorted run is of length B
- In each subsequent merge pass p, we merge $B - 1$ runs
- Run length $\leq B \times (B - 1)^p$
- In two passes ($p = 1$), we want the run length to be $\geq N$
- or $N \leq B \times (B - 1)^1$

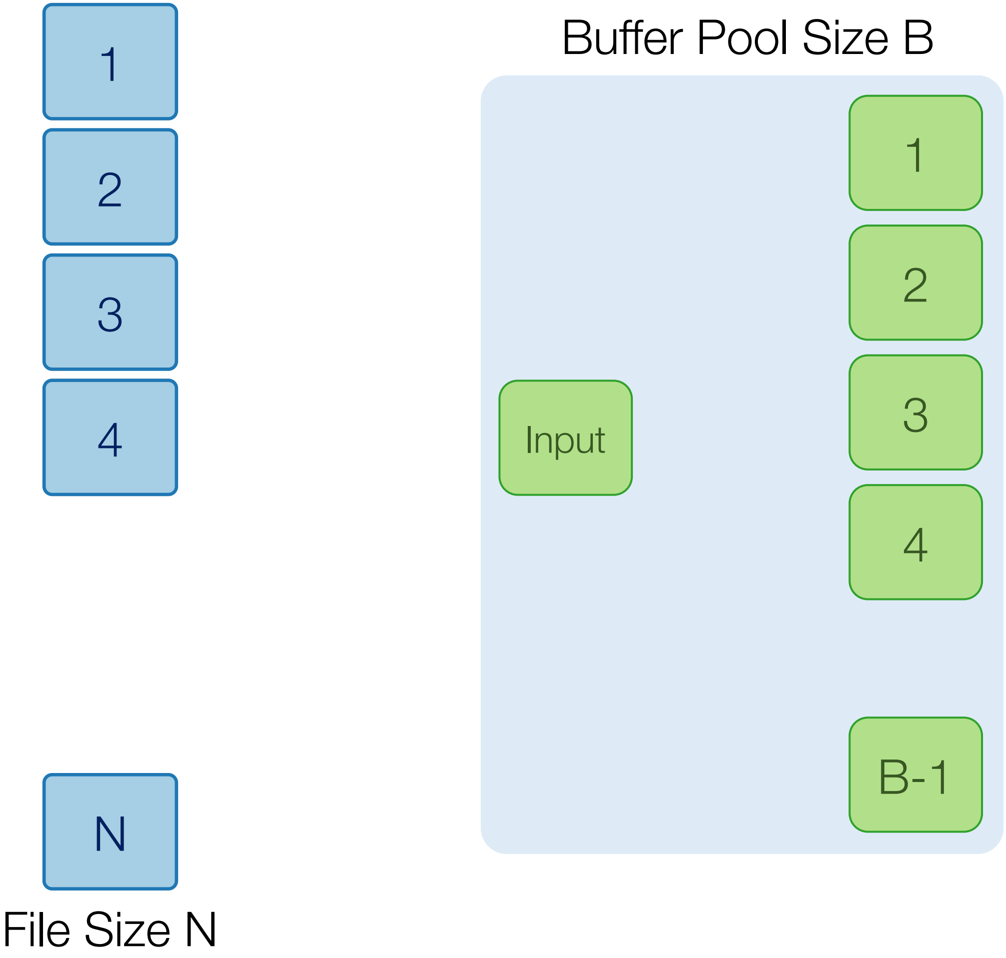
A buffer of size $B = \sqrt{N}$ is needed to sort a table of size N in two passes (1 streaming + 1 merge)

File Size		2- Pass Buffer Size	
Pages (N)	Bytes	Pages (B)	Bytes
100	400 KB	10	40 KB
1,000	4 MB	32	128 KB
10,000	40 MB	100	400KB
100,000	400 MB	317	1.27 MB
1,000,000	4 GB	1000	4 MB
10,000,000	40 GB	3163	12.65 MB
100,000,000	4000 GB	10000	40 MB
1,000,000,000	4 TB	31623	126.5 MB

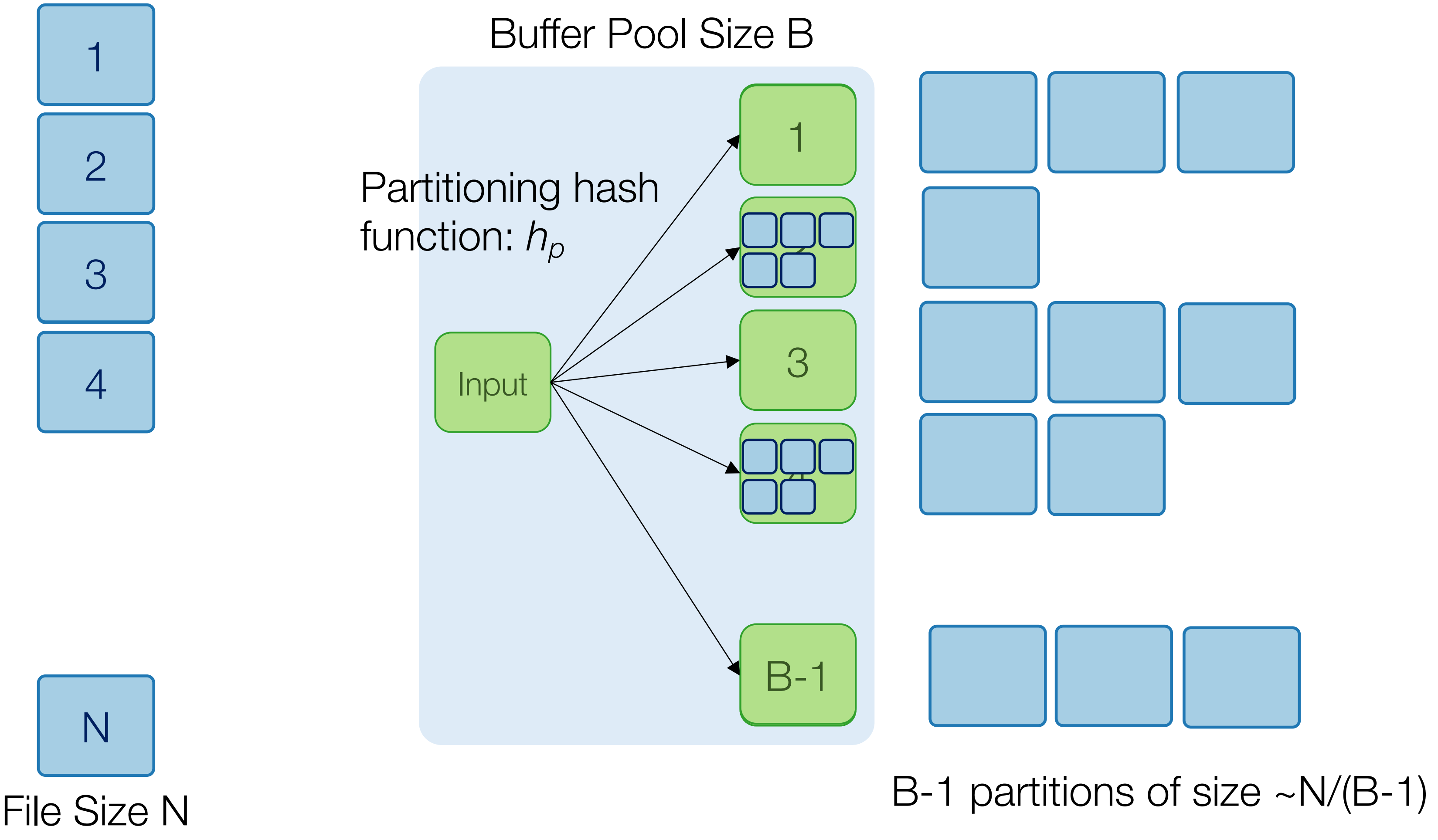
External Hashing

When order isn’t important!

External Hashing: Streaming Phase - Partition



External Hashing: Streaming Phase - Partition

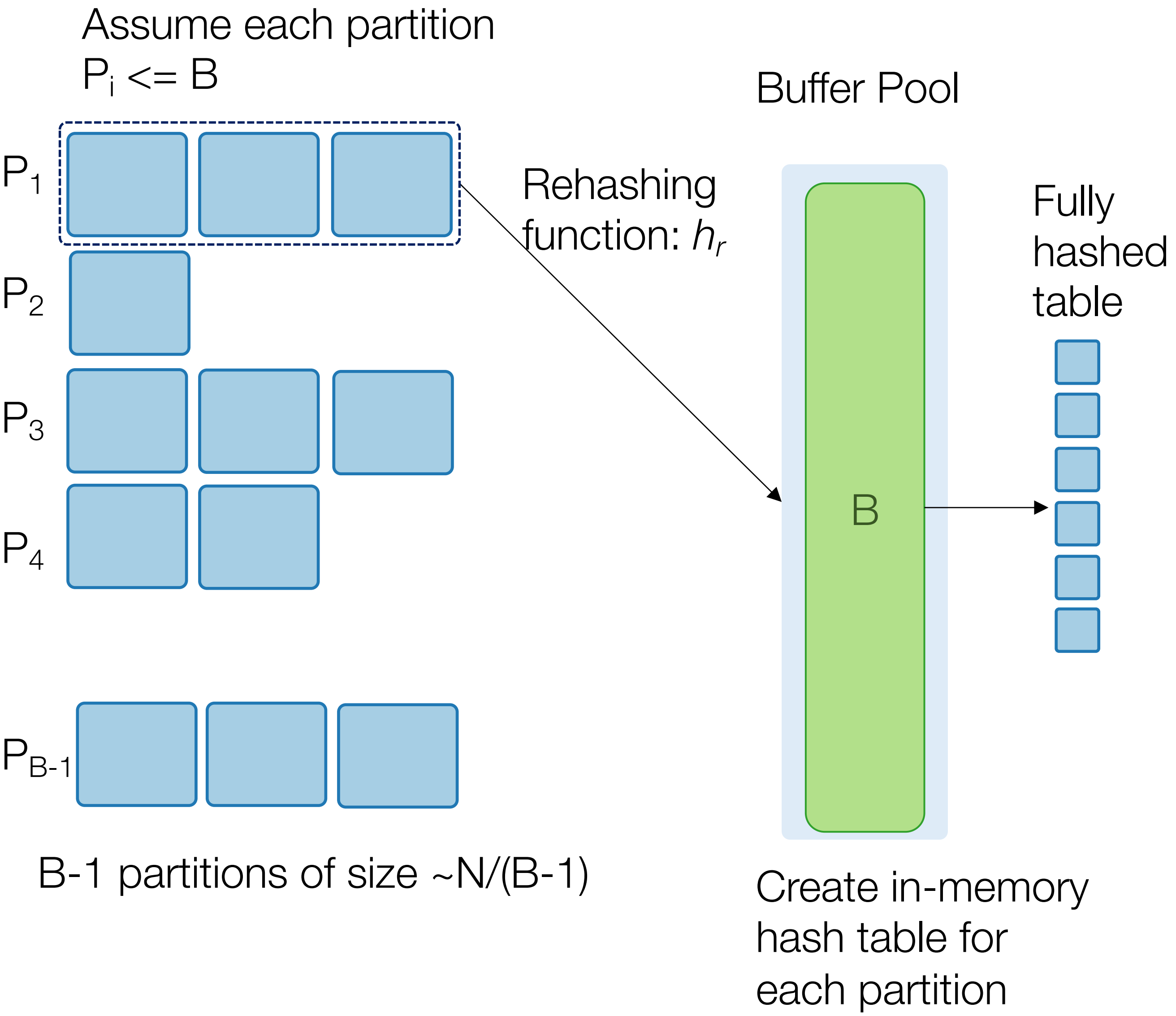

1
2
3
4
N
File Size N
Buffer Pool Size B
Partitioning hash function: h_p
Input
1
3
B-1
B-1 partitions of size ~N/(B-1)

The streaming partition phase produces partitions that have:

1. Many different values
2. Duplicate values that are not contiguous
3. Different sizes!

Cost: $2N$

External Hashing: ReHash


Assume each partition
P_i <= B
P_1
P_2
P_3
P_4
P_{B-1}
B-1 partitions of size ~N/(B-1)
Rehashing function: h_r
Buffer Pool
B
Fully hashed table
Create in-memory hash table for each partition

h_r has to be different from h_p

Each in-memory hash table:

1. Different values in different buckets modulo collisions
2. Duplicate values (same key) stored contiguously in the same bucket

On processing a partition spill out the hash table to disk and process the next partition (note partitions are disjoint!)

Cost: $2N$

2-Pass External Hash

- After the streaming partition pass, we have $B - 1$ partitions
- Each partition must be $\leq\ B$ pages in size to create an in-memory hash table in the ReHash phase
- So the size of the file must be $N \leq (B - 1)\times B$ for a 2-pass external hash!

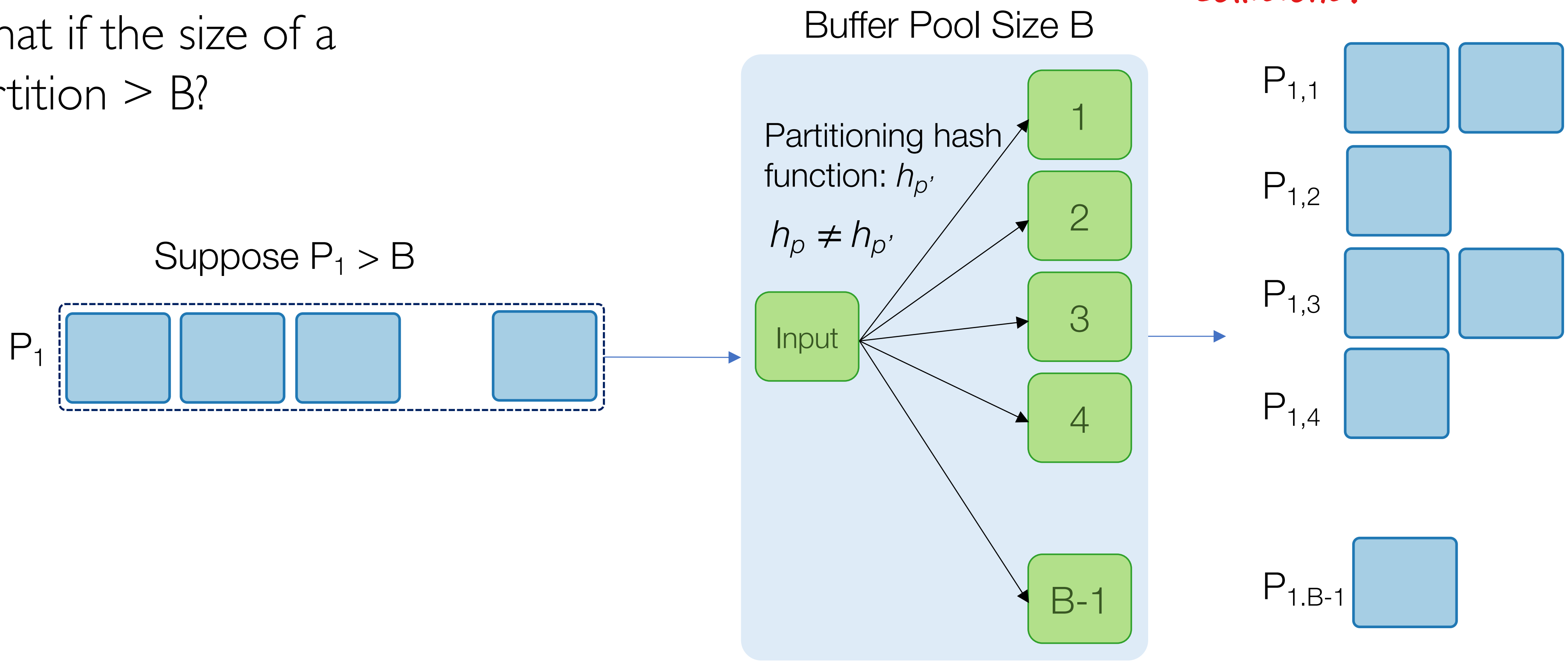
A buffer of size $B = \sqrt{N}$ is needed to hash a table of size N in two passes (1 partitioning + 1 rehash)

Recursive Partitioning

What if the size of a partition > B?

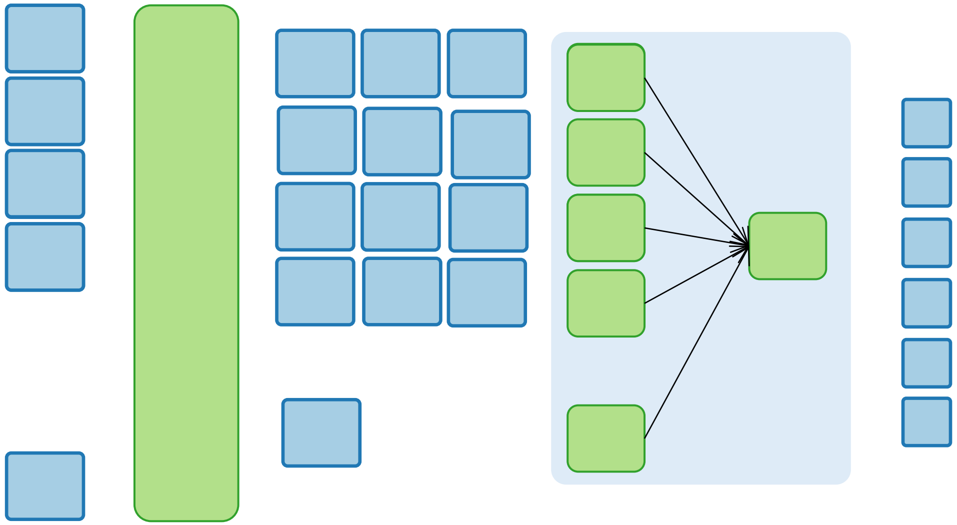
What about too many duplicates?

What about too many collisions?



The Sort-Hash Duality

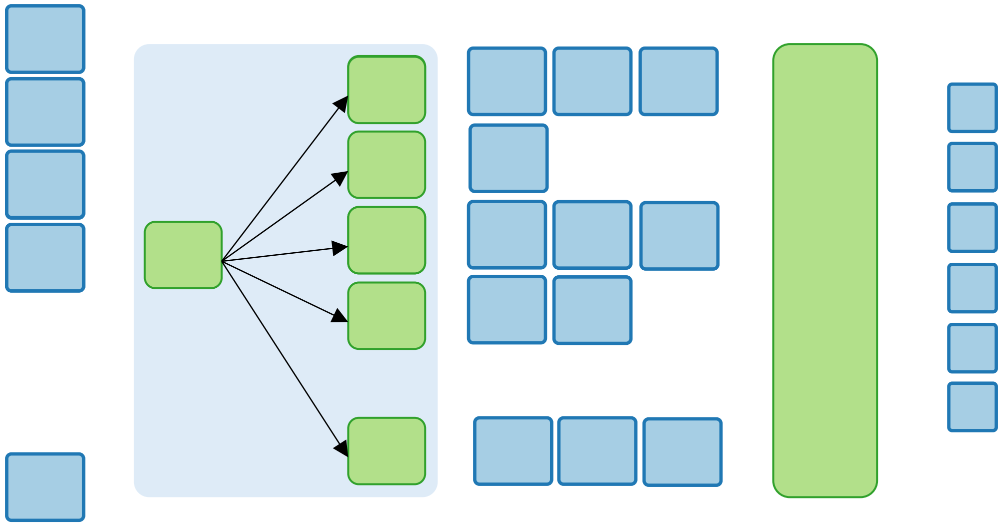
External Sorting



Streaming Sort
(Conquer)

Merge Pass
(Combine)

External Hashing



Streaming Partition
(Divide)

ReHash Pass
(Conquer)

Comparing Sorting with Hashing

	Sorting	*Hashing*
2-Pass Cost	$4N$ IOs ($2N$ for streaming + $2N$ for merge)	$4N$ IOs ($2N$ for streaming partitioning + $2N$ for ReHash)
2-Pass Memory Requirement	$B = \sqrt{N}$	$B = \sqrt{N}$
Duplicate Elimination	Scales with # of items	Scales with # of distinct values
Ordered Results	Supports	Doesn't support
Consistency	Same performance even with duplicates	Sensitive to duplicates & poor hash functions
Computational Cost	More Expensive	Cheaper

Support for higher-order operations

Duplicate Elimination

Streaming Sort Pass – can eliminate some duplicates
Merge Pass – skipping over duplicates

If the file is sorted, scan and skip duplicates

Partitioning Pass – can eliminate some duplicates
ReHash Pass – If entry in hash table, skip, else insert

If the file is hashed, the result is the hash-table

Grouping & Aggregation

- Maintain a running aggregate for each group key
- MIN, MAX, COUNT, SUM → just update the aggregate
- AVG → update two aggregates: SUM, COUNT and then compute the AVG

Joins

- Sort-Merge Join
- Hash Join