

Sorting and Hashing

Algorithms and Costs

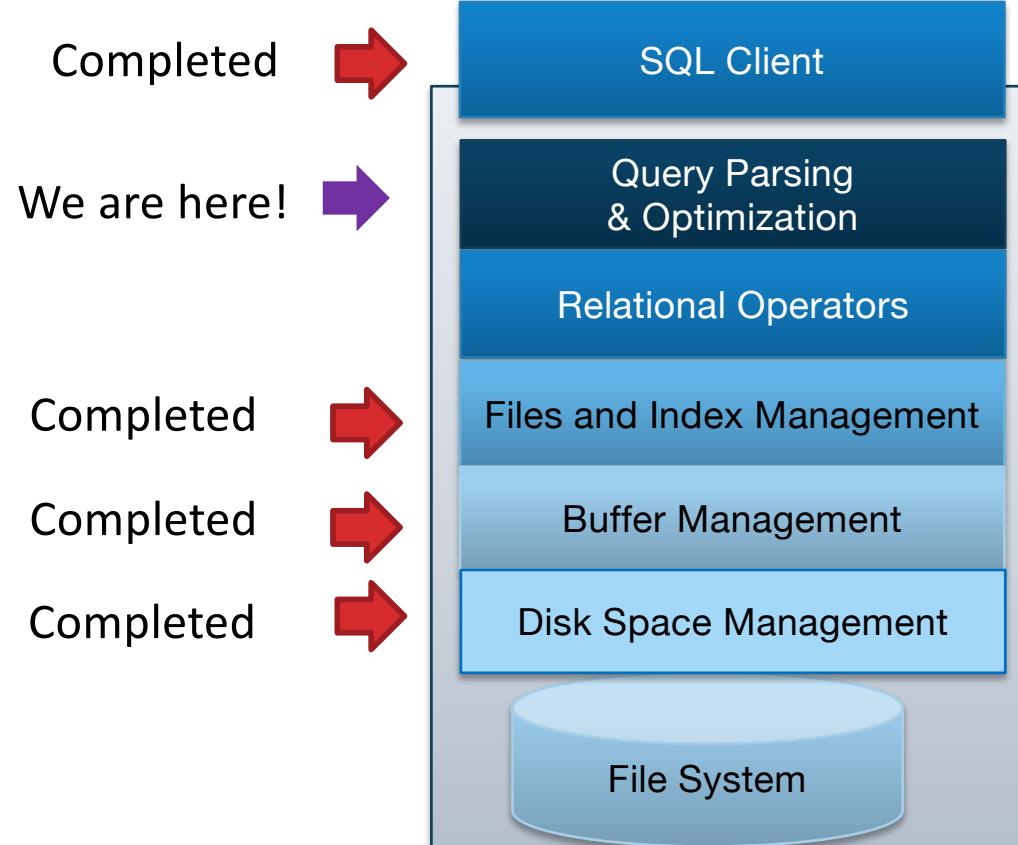
Alvin Cheung

Fall 2022

Reading: R & G Chapter 9.1,
13.1-3, 13.4.2



Architecture of a DBMS: What we've learned



Why Sort?



- “Rendezvous”
 - Eliminating duplicates (DISTINCT)
 - Grouping for summarization (GROUP BY)
 - Upcoming sort-merge join algorithm
- Explicitly requested: ordering
 - For ordered outputs (ORDER BY)
 - First step in bulk-loading tree indexes
- Problem: sort 100GB of data with 1GB of RAM.
 - why not virtual memory?

Out-of-Core Algorithms



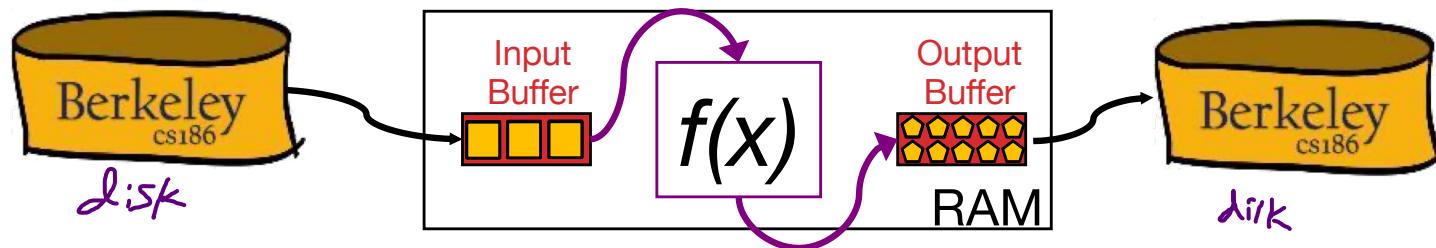
- Two themes
 1. Single-pass streaming data through RAM
 2. Divide (into RAM-sized chunks) and Conquer

Single-pass Streaming

(for creating
map)



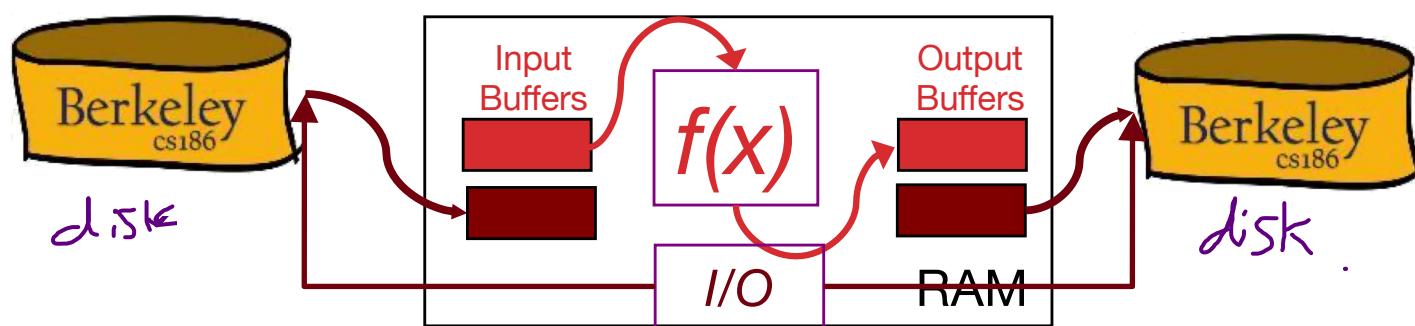
- Simple case: “Map”
 - Goal: Compute $f(x)$ for each record, write out the result
 - Challenge: minimize RAM, call read/write rarely
- Approach
 - Read a chunk from INPUT to an Input Buffer
 - Write $f(x)$ for each item to an Output Buffer
 - When Input Buffer is consumed, read another chunk
 - When Output Buffer fills, write it to OUTPUT



Better: Double Buffering pt 1



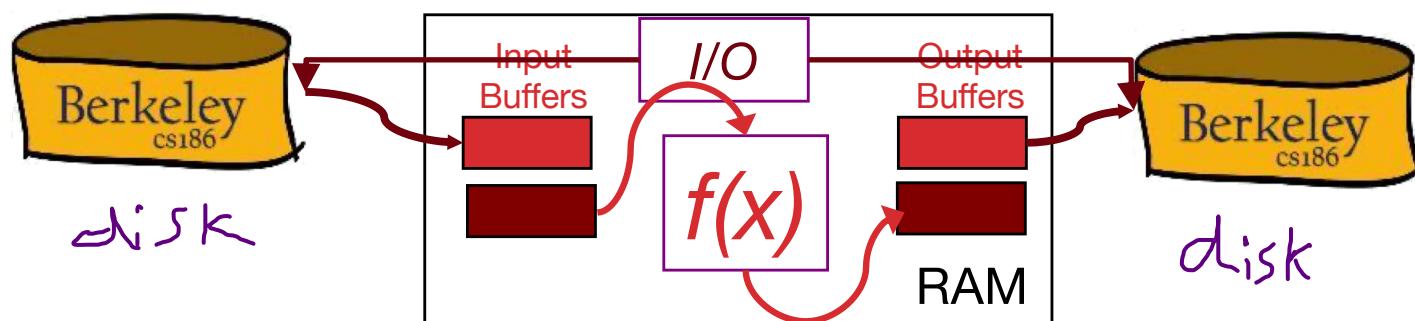
- **Main thread** runs $f(x)$ on one pair I/O buffers
- Second **I/O thread** drains/fills unused I/O buffers in parallel
 - Why is parallelism available?
 - Theme: I/O handling usually deserves its own thread
- Main thread ready for a new buffer? Swap!



Better: Double Buffering pt 2



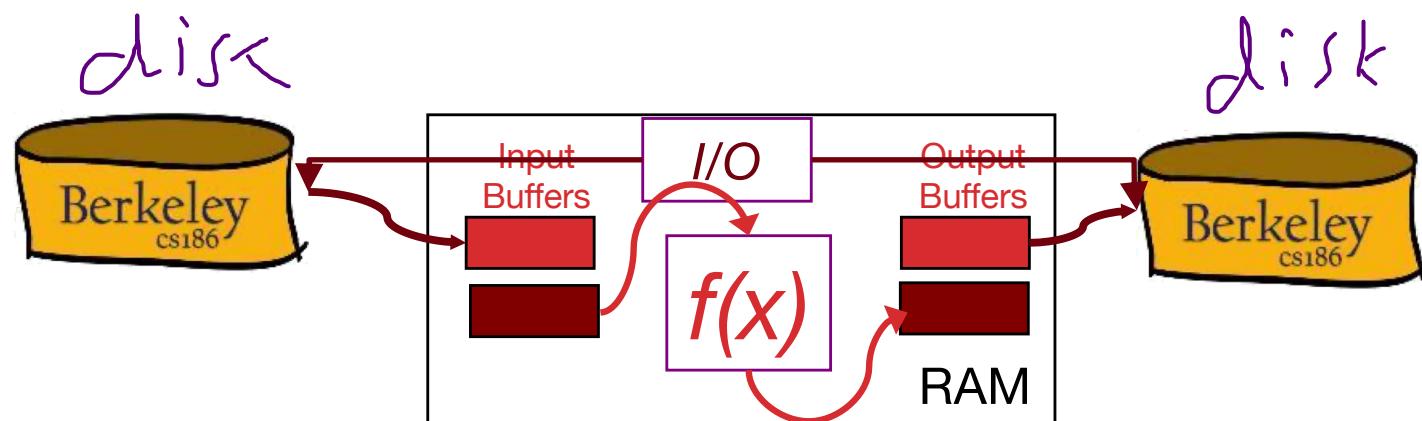
- **Main thread** runs $f(x)$ on one pair I/O buffers
- Second **I/O thread** drains/fills unused I/O buffers in parallel
 - Why is parallelism available?
 - Theme: I/O handling usually deserves its own thread
- Main thread ready for a new buffer? Swap!



Double Buffering applies to all streams



- Usable in any of the subsequent discussion
 - Assuming you have RAM buffers to spare!
 - But for simplicity we won't bring this up again.



Sorting & Hashing: Formal Specs



Sorting

- Produce an output file F_S
 - with contents R stored in order by a given sorting criterion

Hashing

- Produce an output file F_H
 - with contents R , arranged on disk so that no two records that have the same value are separated by a record with a different value.
 - I.e. matching records are always “stored consecutively” in F_H .
 - Recall our goal of using this to eliminate duplicates using only sequential scans

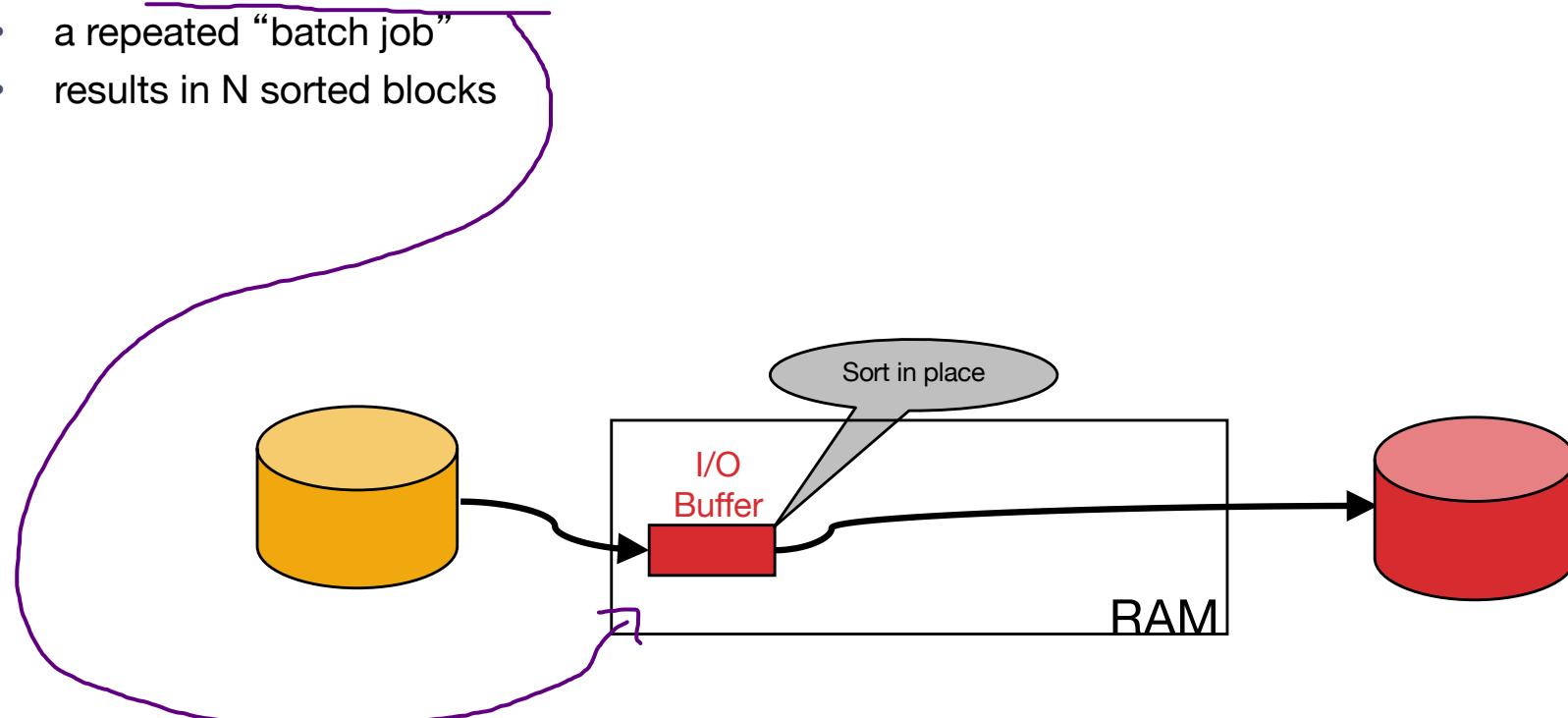
Given:

- A file F :
 - containing a multiset of records R
 - consuming \mathbf{N} blocks of storage
- Two “scratch” disks
 - each with $\gg N$ blocks of free storage
- A fixed amount of space in RAM
 - memory capacity equivalent to \mathbf{B} blocks of disk

Sorting: 2-Way (a strawman)



- **Pass 0 (conquer a batch):**
 - read a page, sort it, write it.
 - only one buffer page is used
 - a repeated “batch job”
 - results in N sorted blocks

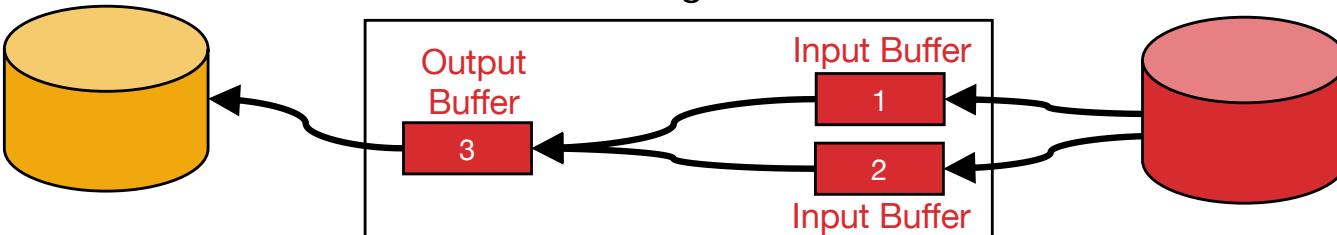


Sorting: 2-Way (a strawman), cont



- Pass 0 (conquer a batch):
 - read a page, sort it, write it.
 - only one buffer page is used
 - a repeated “batch job”
 - results in N sorted blocks
- Pass 1, 2, 3, ..., etc. (merge via streaming):
 - requires 3 buffer pages
 - note: this has nothing to do with double buffering!
 - merge pairs of runs into runs twice as long
 - a streaming algorithm, as in the previous slide!
 - Drain/fill buffers as the data streams through them

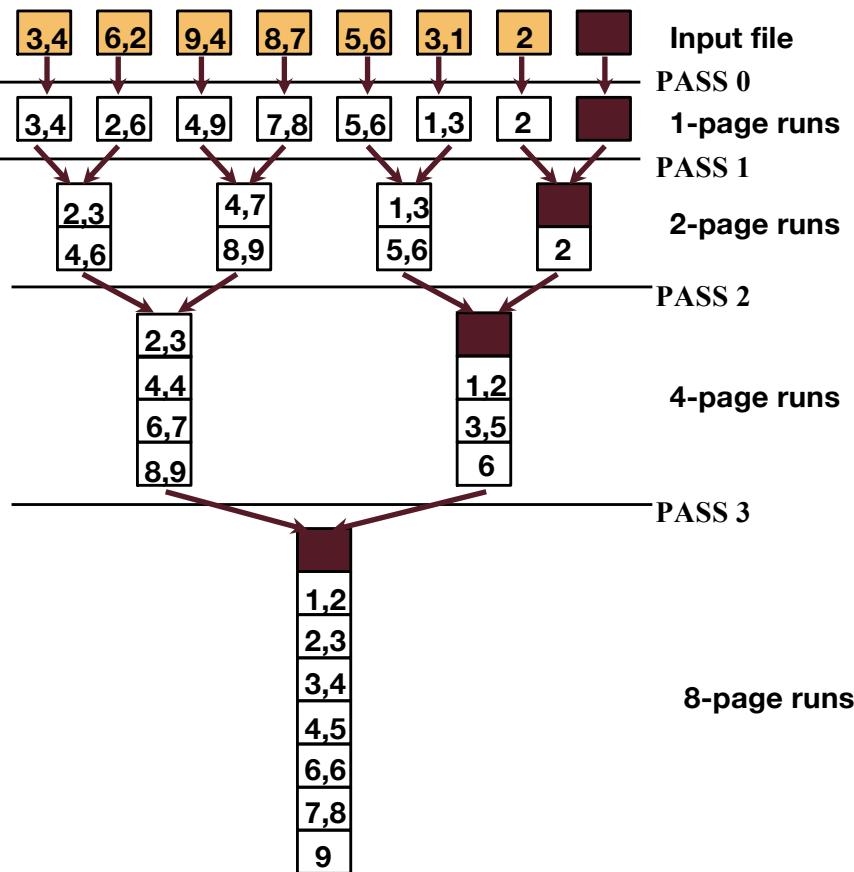
edge part of
Mergesort



when only 3 buffer pages

$$N = \# \text{ pages/blocks}$$

Two-Way External Merge Sort

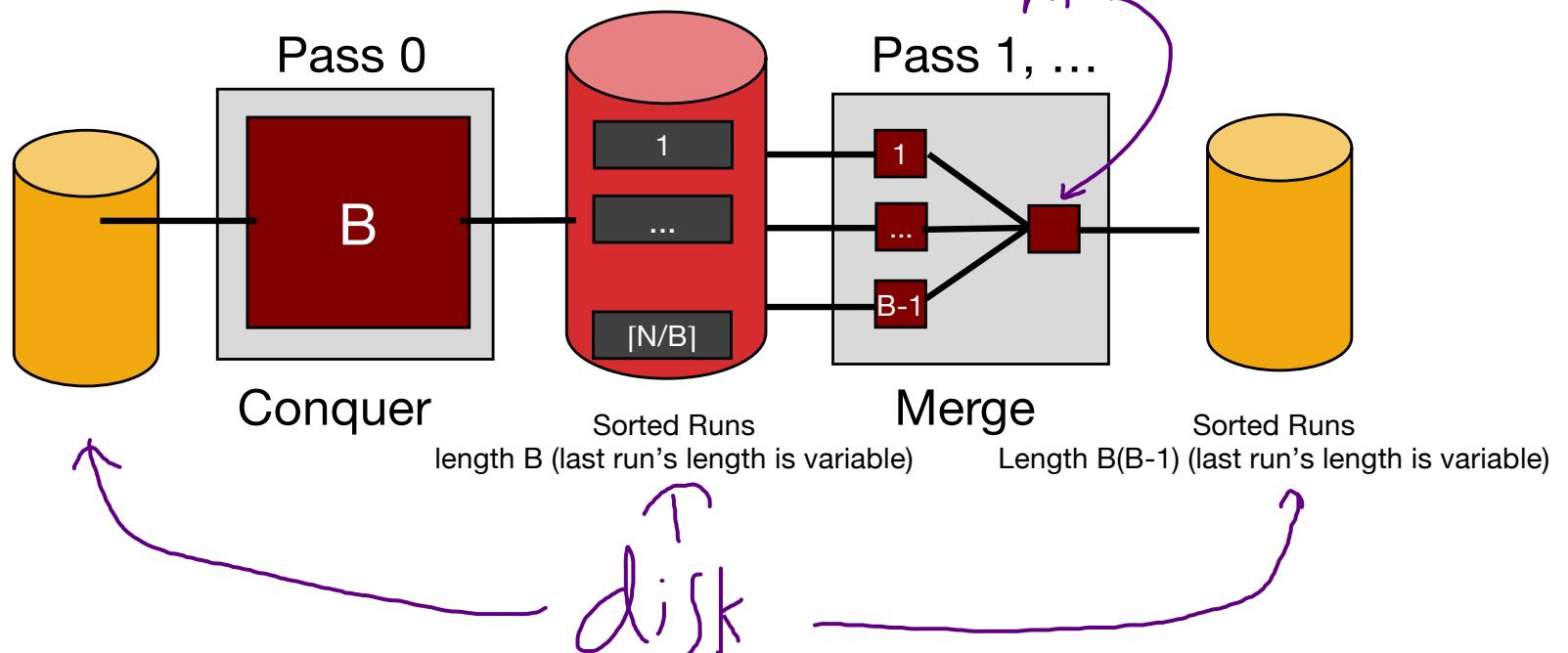


when B buffer pages

General External Merge Sort



- We got more than 3 buffer pages. How can we utilize them?
 - Big batches in pass 0, many streams in merge passes
- To sort a file with N pages using B buffer pages:
 - Pass 0: use B buffer pages. Produce $\lceil N/B \rceil$ sorted runs of B pages each.
 - Pass 1, 2, ..., etc.: merge $B-1$ runs at a time. ← have one buffer page as output



two-way: $B = 2$

Whole #

Cost of External Merge Sort



- Number of passes: $1 + \lceil \log_{B-1} \lceil N / B \rceil \rceil$
- Total I/Os = (I/Os per pass) * (# of passes) = $2^*N * (1 + \lceil \log_{B-1} \lceil N / B \rceil \rceil)$
- E.g., with 5 buffer pages, to sort 108 page file:
 - Pass 0: $\lceil 108 / 5 \rceil = 22$ sorted runs of 5 pages each
 - last run is only 3 pages
 - Pass 1: $\lceil 22 / 4 \rceil = 6$ sorted runs of 20 pages each
 - last run is only 8 pages
 - Pass 2: $\lceil 6 / 4 \rceil = 2$ sorted runs, 80 pages and 28 pages
 - Pass 3: Sorted file of 108 pages

Formula check: $1 + \lceil \log_4 22 \rceil = 1+3 \rightarrow 4 \text{ passes } \checkmark$

of Passes of External Sort

(Total I/O is $2N * \# \text{ of passes}$)



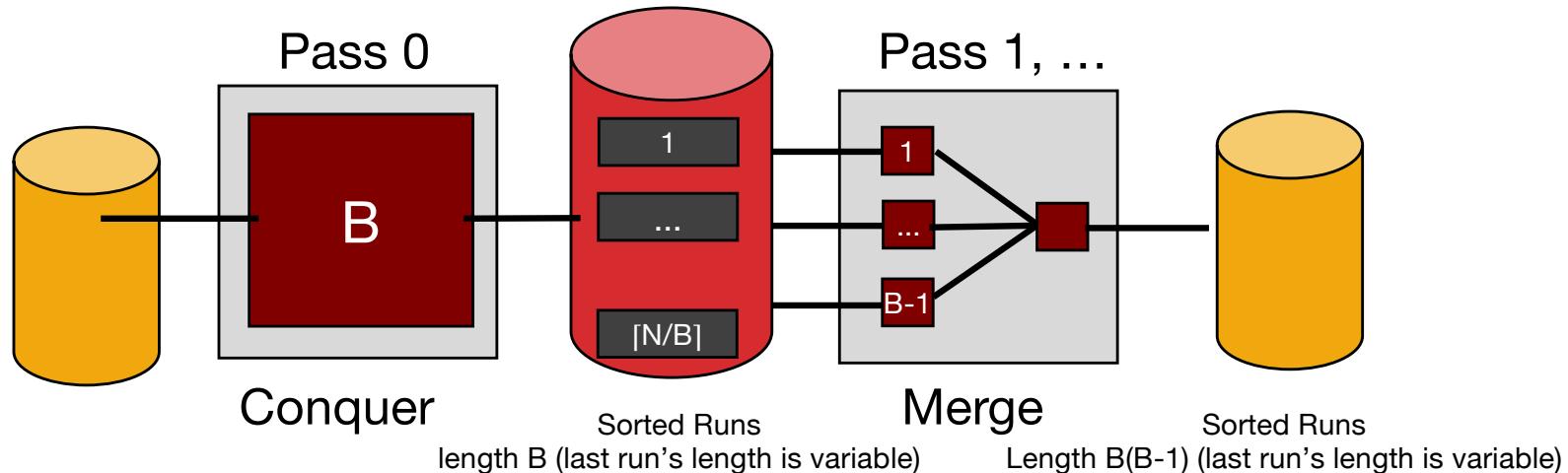
N	B=3	B=5	B=9	B=17	B=129	B=257
100	7	4	3	2	1	1
1,000	10	5	4	3	2	2
10,000	13	7	5	4	2	2
100,000	17	9	6	5	3	3
1,000,000	20	10	7	5	3	3
10,000,000	23	12	8	6	4	3
100,000,000	26	14	9	7	4	4
1,000,000,000	30	15	10	8	5	4

Few runs can already sort large amounts of data!

Memory Requirement for External Sorting



- How big of a table can we sort in exactly two passes?
 - Each “sorted run” after Phase 0 is of size B
 - Can merge up to B-1 sorted runs in Phase 1
- Answer: $B(B-1) \sim B^2$ data in two passes, using size B space
 - Sort X amount of data in about $B = \sqrt{X}$ space (if we run only 2 passes)



Alternative: Hashing



- Many times we don't require order
 - E.g., remove duplicates, form groups
- Often just need to group matches together
- Hashing does this
 - But how to do it out-of-core??

Streaming Partition (Divide)



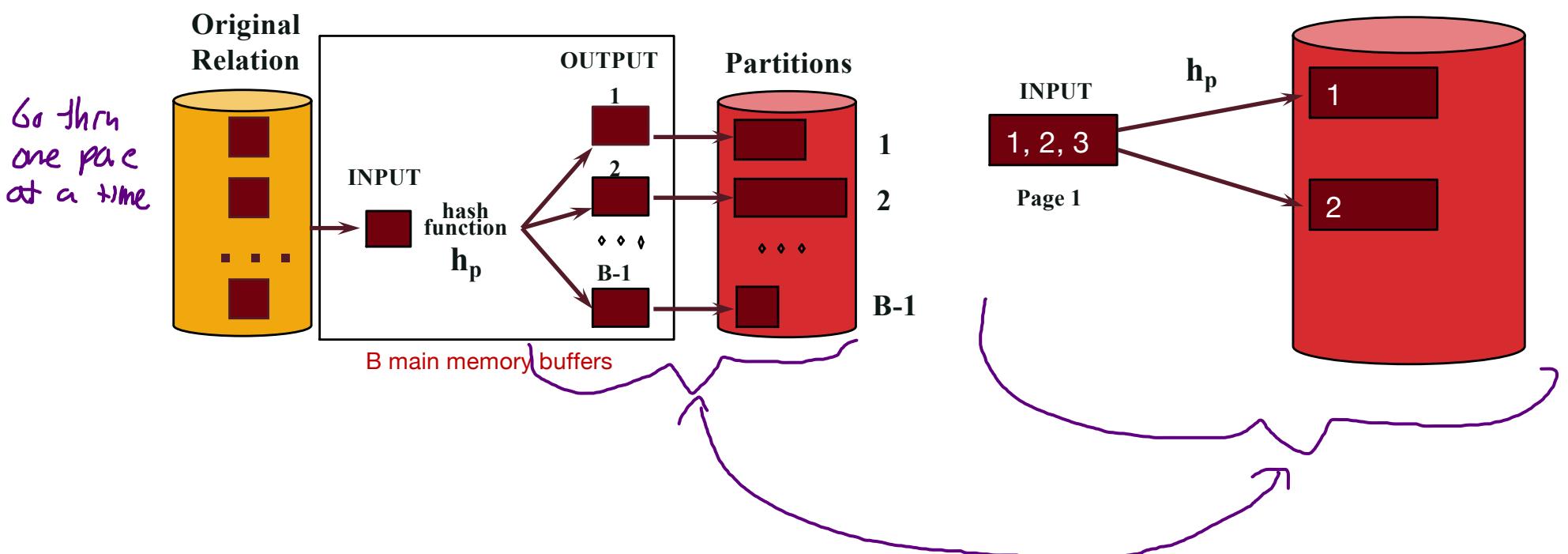
- Use a hash function h_p to stream records and create partitions
 - All matches grouped together in the same partition
 - Each partition can have a mix of values based on h_p
 - Partitions are written to disk
 - Each partition can take up multiple pages

Two Phases: Divide



- Partition:
(Divide)

Example

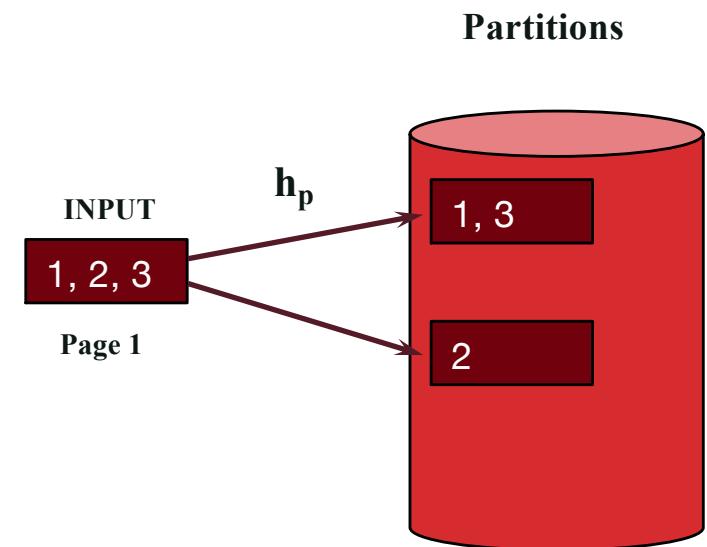
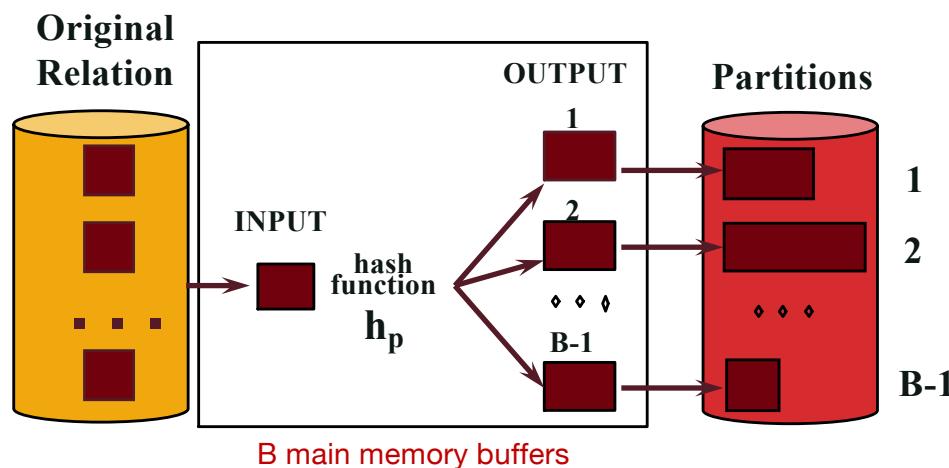


Two Phases: Divide



- Partition:
(Divide)

Example

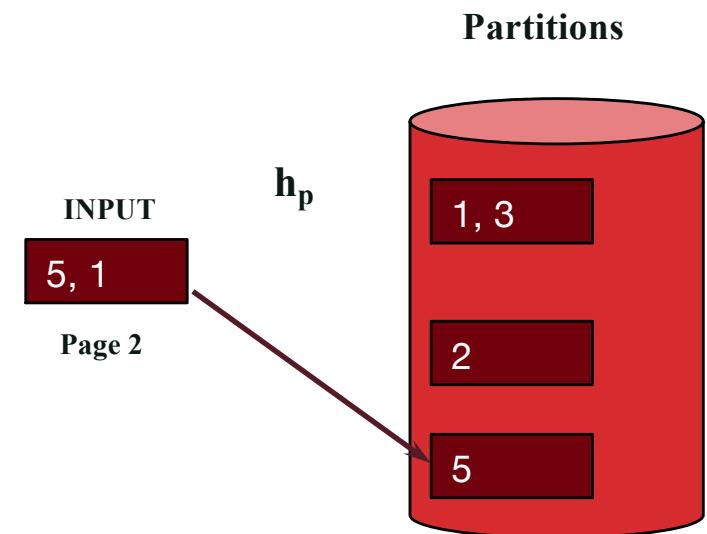
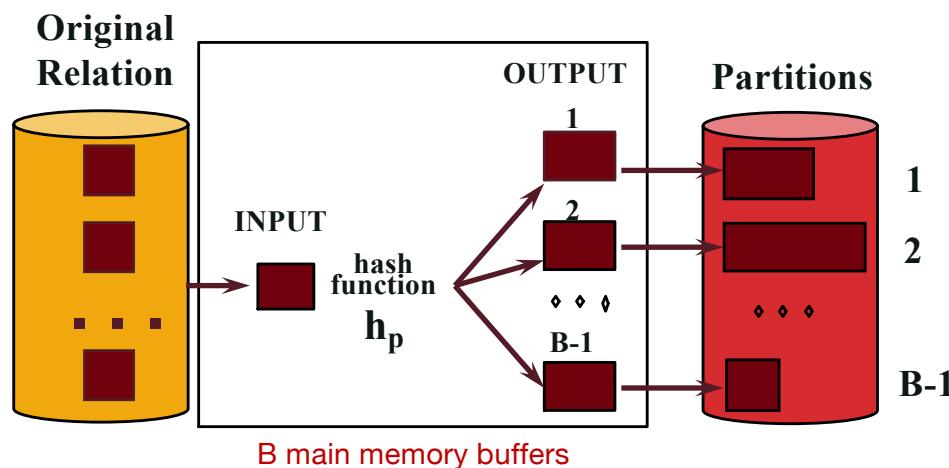


Two Phases: Divide



- Partition:
(Divide)

Example

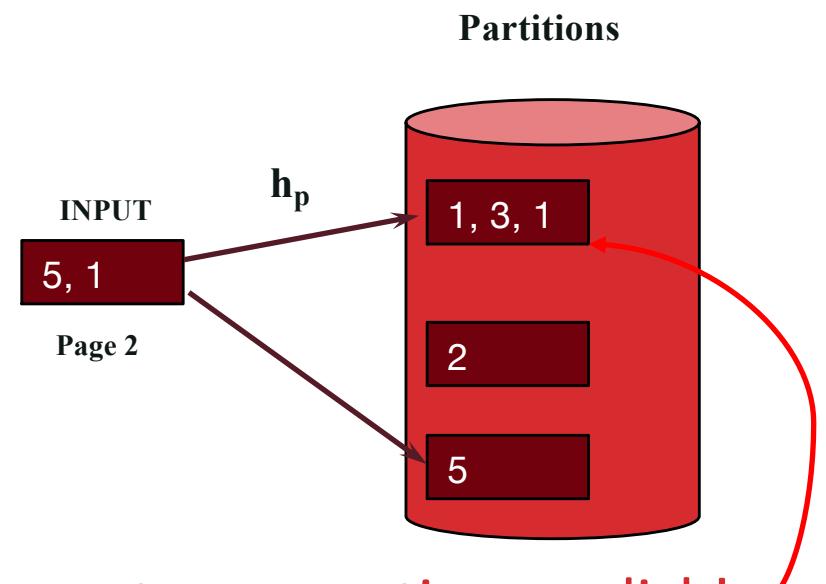
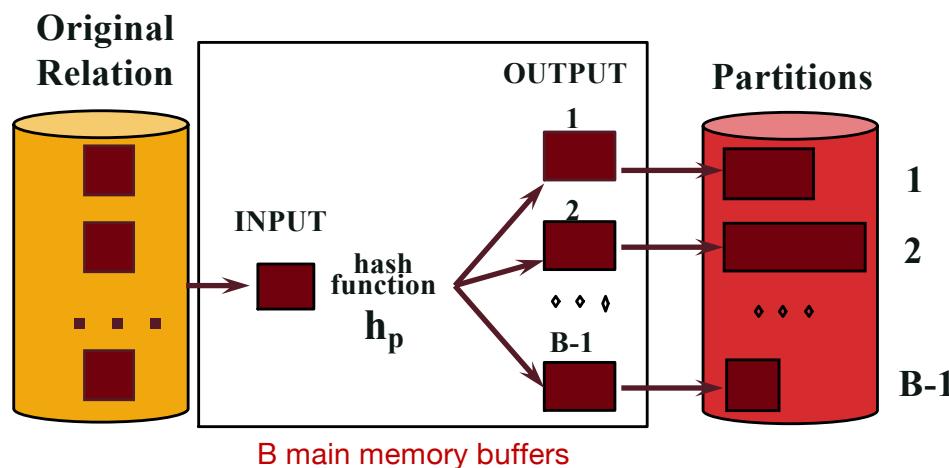


Two Phases: Divide



- Partition:
(Divide)

Example



The 1's are not consecutive on disk!

Hashing rule: duplicates next to each other

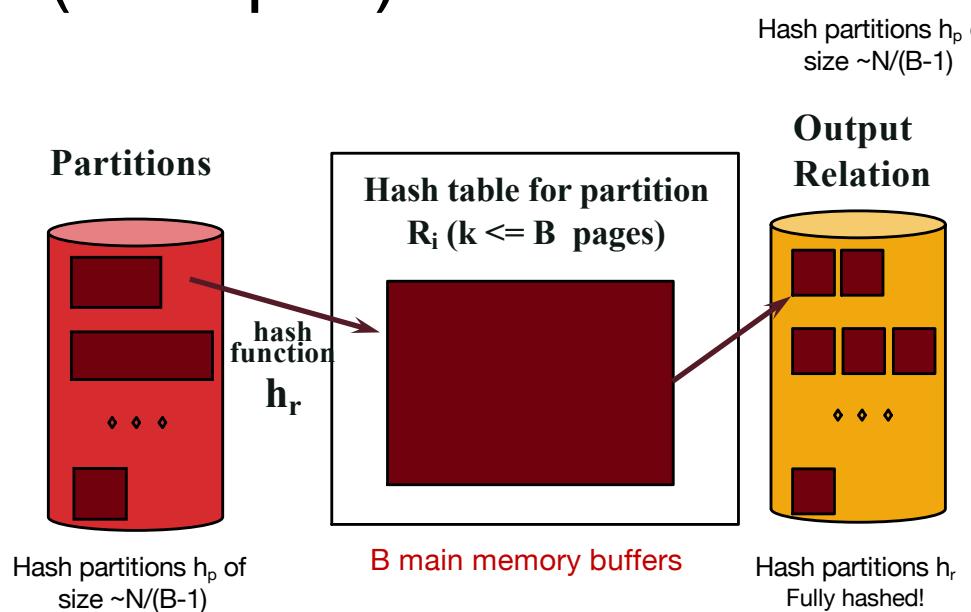
ReHash (Conquer)



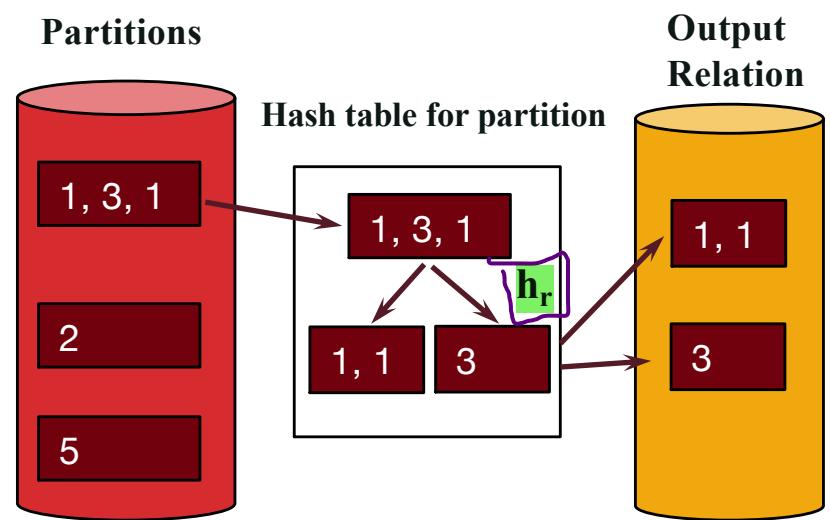
- Read each partition into RAM hash table one at a time, using different hash function h_r
 - Each hash table bucket contains a small number of distinct values
- Read out the RAM hash table buckets and write to disk
 - Duplicate values are now contiguous on disk

Two Phases: Conquer

- Rehash:
(Conquer)



Example

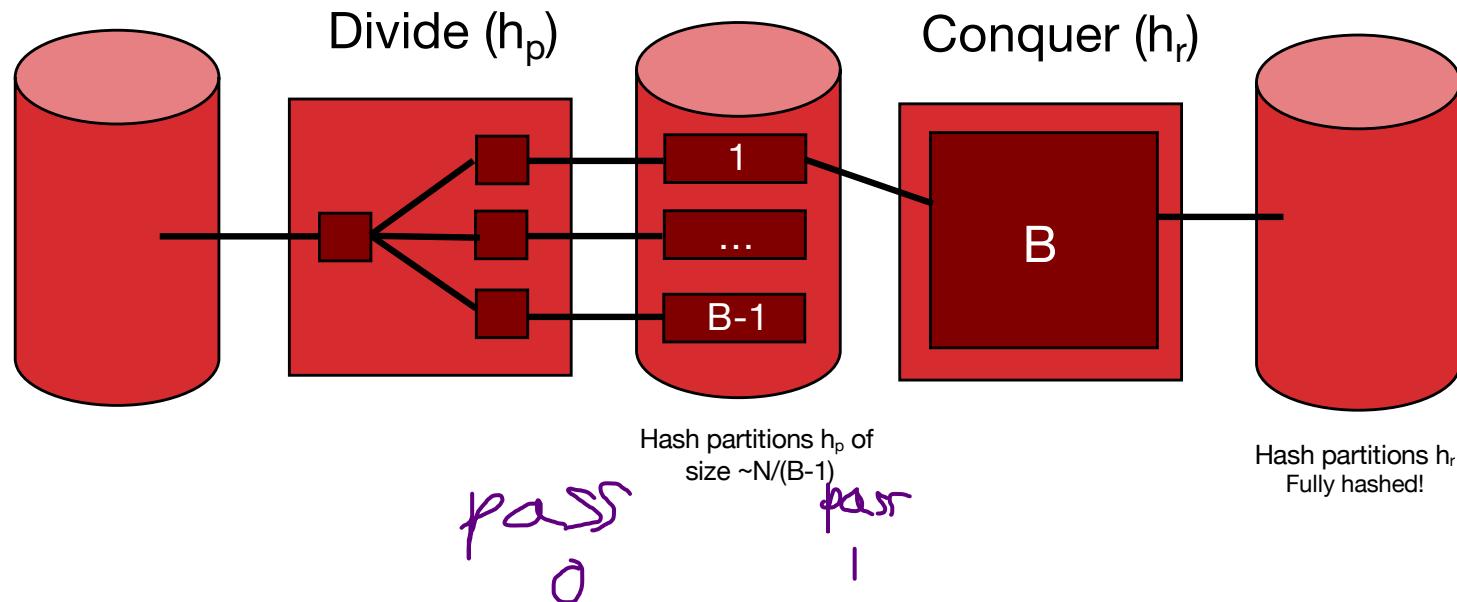


Cost of External Hashing



Total I/Os $\sim 2^*N^*(\# \text{ passes}) = 4^*N$

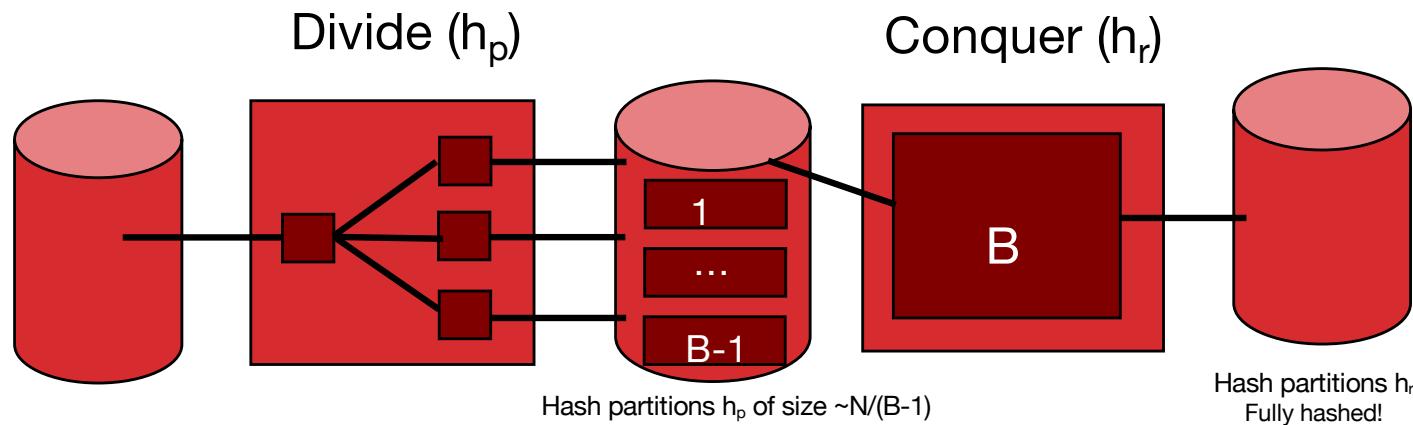
(includes initial read, final write)



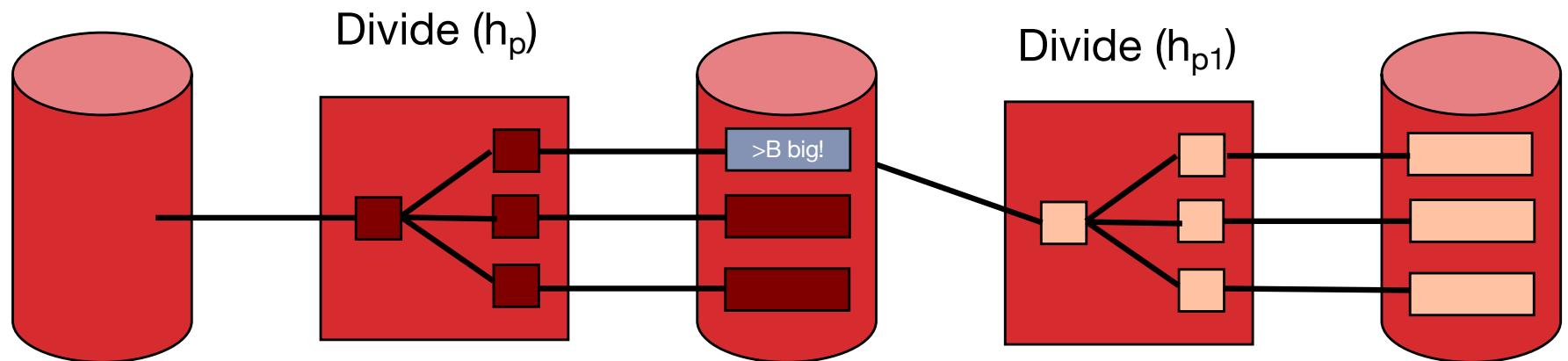
Memory Requirement



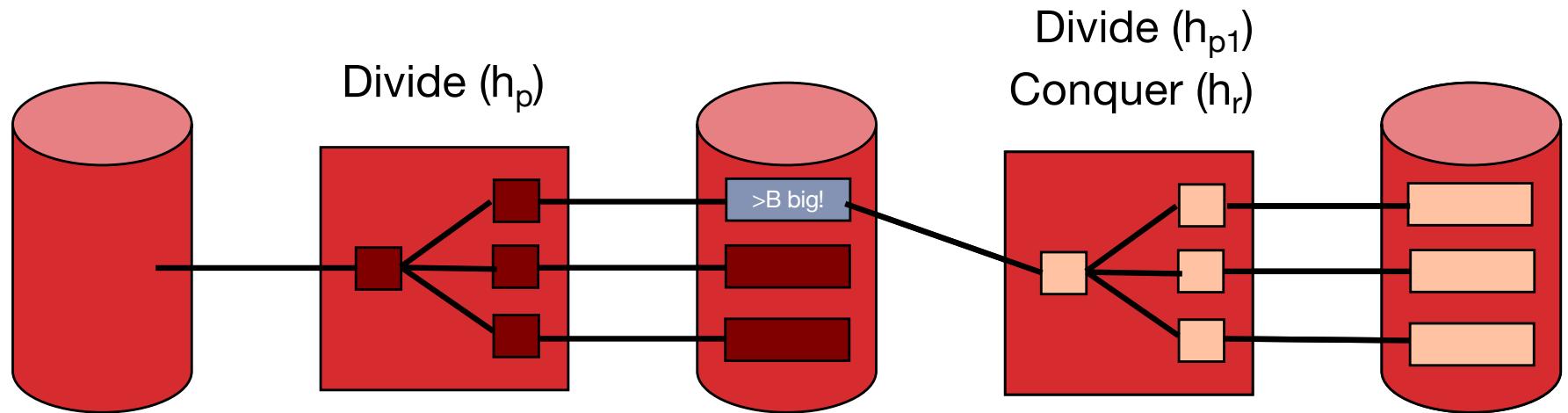
- How big of a table can we hash in exactly two passes?
 - $B-1$ “partitions” result from Pass 1
 - Each should be no more than B pages in size
 - Answer: $B(B-1) \sim B^2$
 - We can hash a table of size X in about $B = \sqrt{X}$ space (if we run only 2 passes)
 - Note: assumes hash function distributes records evenly!
- Have a bigger table? Recursive partitioning!



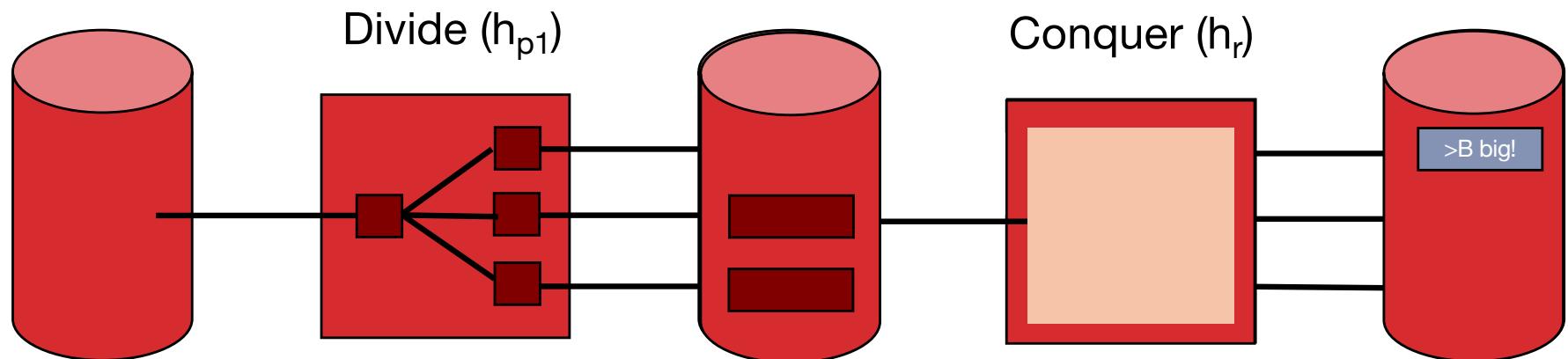
Recursive Partitioning, Pt 1



Recursive Partitioning, Pt 2



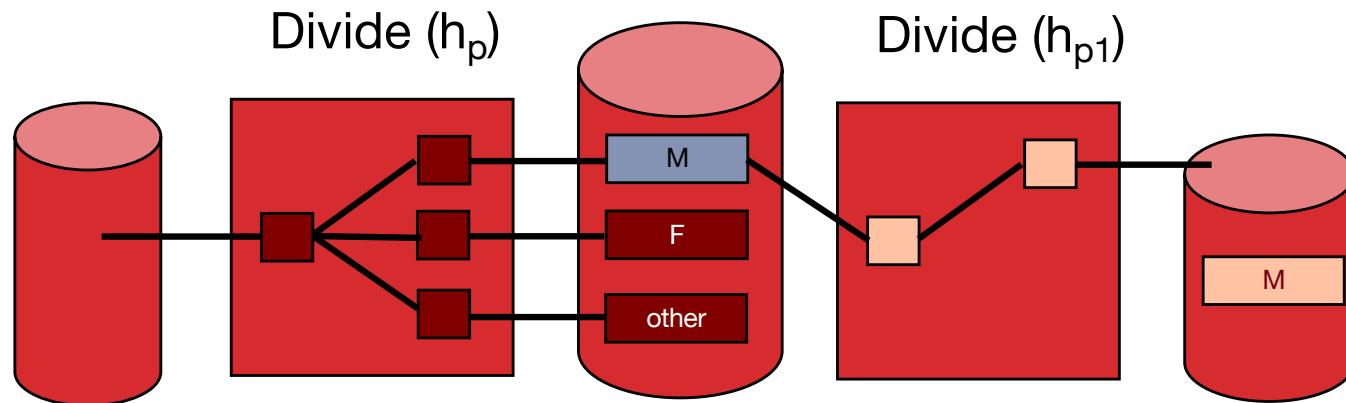
Recursive Partitioning, Pt 3



A Wrinkle: Duplicates



- Consider a dataset with a *very* frequent key
 - E.g. in a big table, consider the *gender* column
- What happens during recursive partitioning?

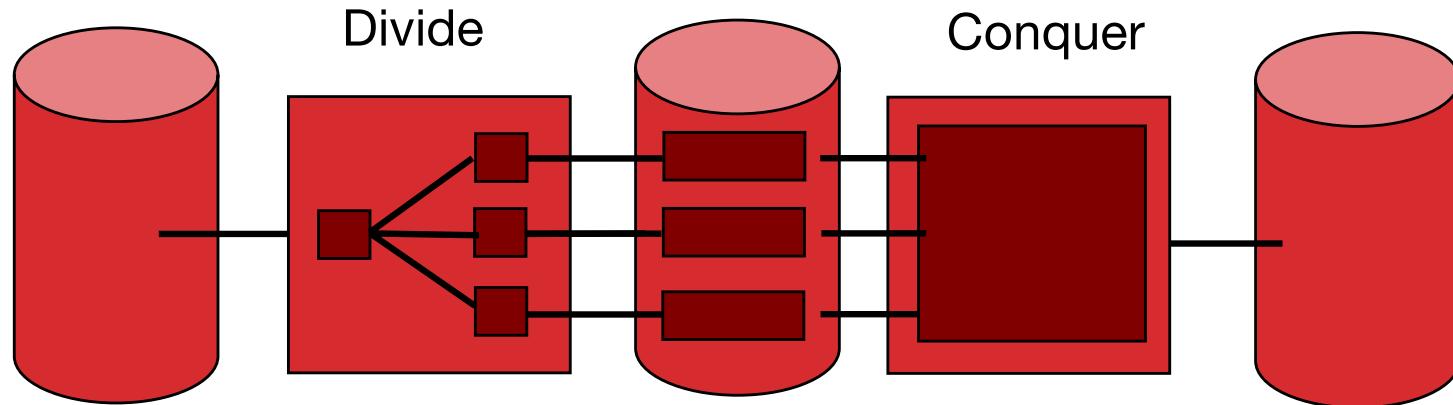


Question...



How does external hashing compare
with external sorting?

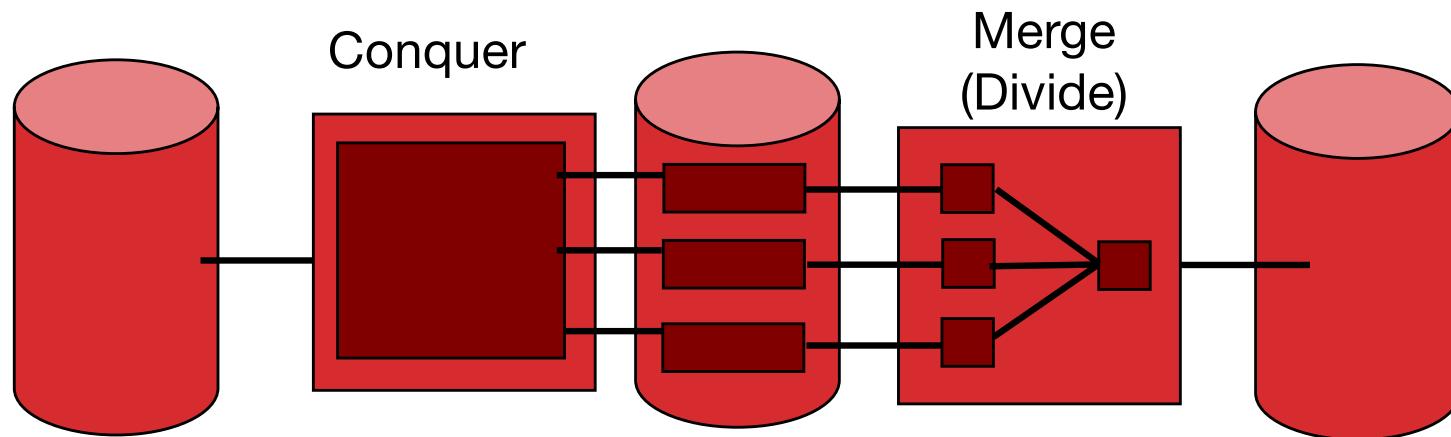
Cost of External Hashing: 1-Pass Case



**cost ~ $4 \cdot N$ I/Os
(including initial read, final write)**

(case for 1 divide pass and 1 conquer pass only. Not the general formula!)

Cost of External Sorting



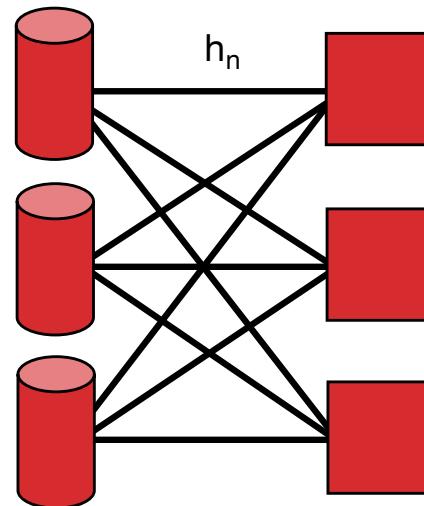
cost = 4*N I/Os
(including initial read, final write)

(case for 1 divide pass and 1 conquer pass only. Not the general formula!)

Parallelize me! Hashing Phase 1



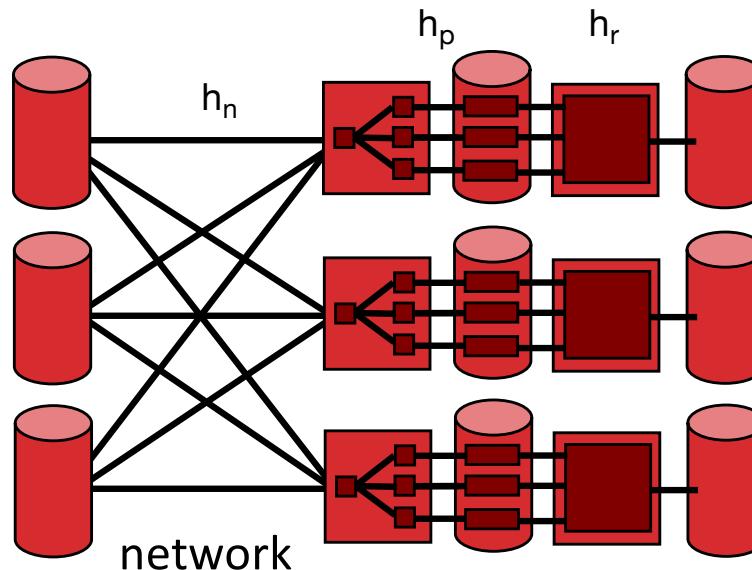
- Phase 1: shuffle data across machines (h_n)
 - stream out to network as it is scanned
 - which machine for this record?
use (yet another) independent hash function h_n



Parallelize me! Hashing Phase 2

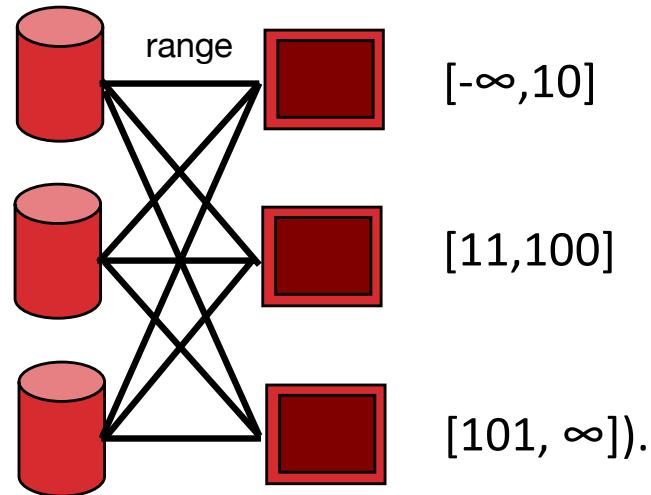


- Phase 1: shuffle data across machines (h_n)
- Receivers proceed with phase 1 as data streams in
 - from local disk and network



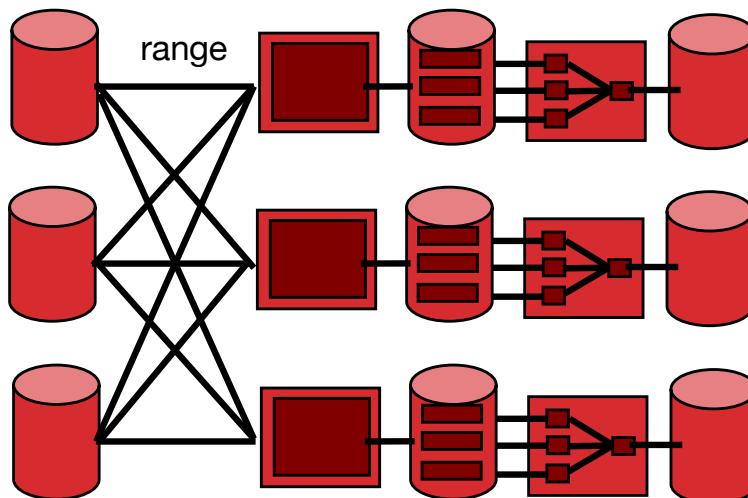
Parallelize me! Sorting

- Pass 0: shuffle data across machines
 - stream out to network as it is scanned
 - which machine for this record?
Split on value range (e.g. $[-\infty, 10]$, $[11, 100]$, $[101, \infty]$).



Parallelize me! Sorting, cont

- Pass 0: shuffle data across machines
- Receivers proceed with pass 0 as the data streams in
- A Wrinkle: How to ensure ranges are the same #pages?!
 - i.e. avoid data skew?



Summary

- Sort/Hash Duality
 - Hashing is Divide & Conquer
 - Sorting is Conquer & Merge
- Sorting is overkill if we just want to group tuples together
 - But sometimes a win anyhow
- Don't forget one pass streaming and double buffering
 - Can “hide” the latency of I/O behind CPU work
- Don't we have TB of RAM these days?
 - We still have a memory hierarchy!