Discussion 5

Sorting and Hashing

Binbin Chen

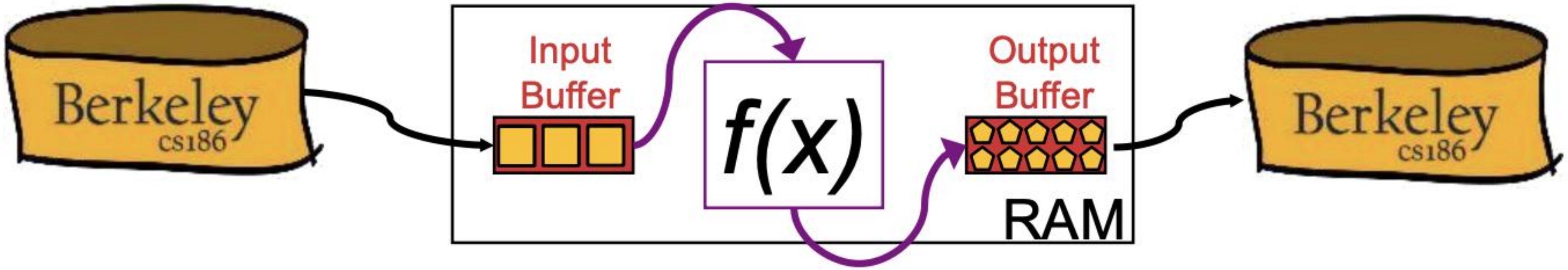
chenbb@shanghaitech.edu.cn

Purpose of sorting and hashing

- sorting
  - rendezvous matches (getting together tuples processed at the same time in the same place)
    - like eliminating duplicates, grouping
  - ordering
    - like step in B+ trees
  - infeasibility of virtue memory
    - sort is incompatible with random disk IOs
    - slow with big buffer
- hashing
  - often just rendezvous matches
  - different with traditional hashing with hash tables
    - divide and conquer for out-of-core algorithms

Out-of-Core (RAM) Algorithms

- Two themes
  - Single-pass streaming data from disk through RAM back to disk
    - Read a chunk from INPUT to an Input Buffer
    - Write f(x) to map each item to an Output Buffer
    - When Input Buffer is consumed, read another chunk
    - When Output Buffer fills, write it to OUTPUT
  - Divide (into RAM-sized chunks) and Conquer

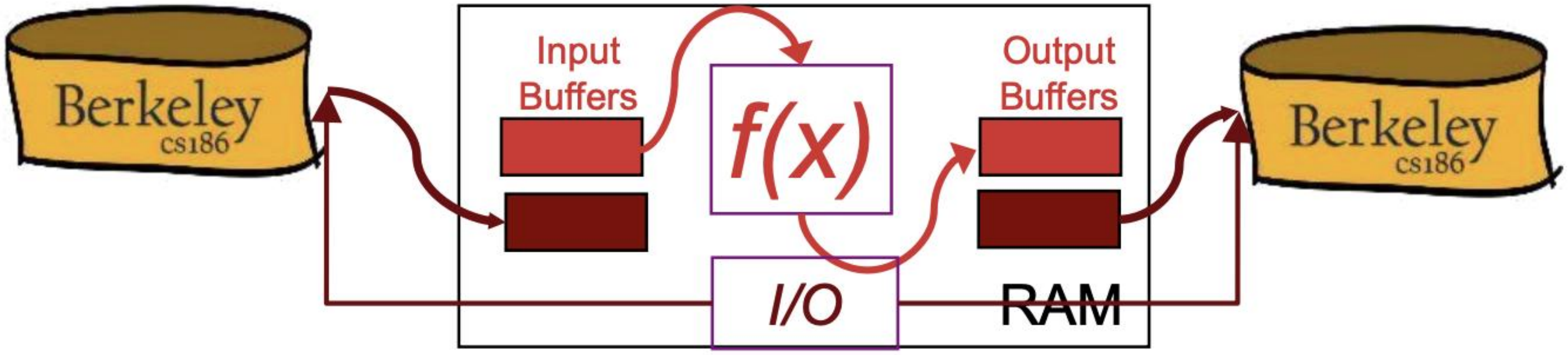


Note: the shape in the output buffer is smaller because f(x) maps the item to smaller bytes, so input and output are not synchronize

Optimization for Streaming - Double Buffering

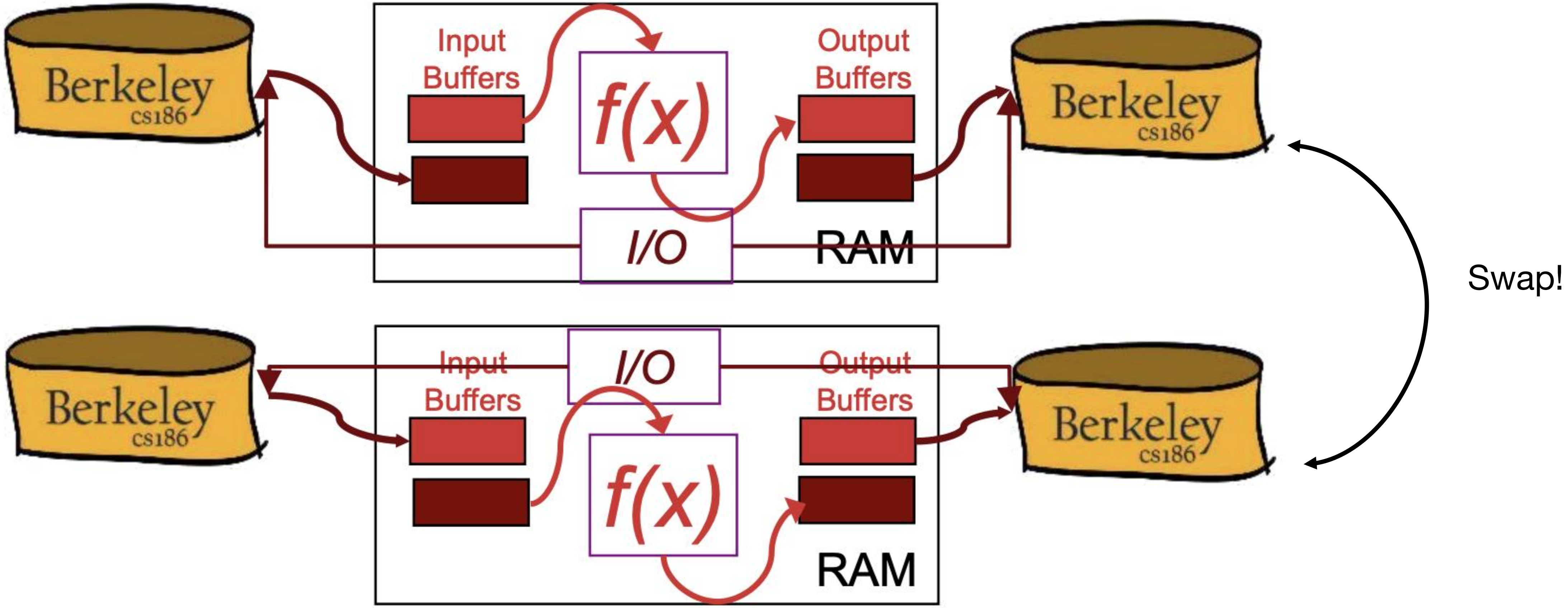
- Main thread runs f(x) on one pair I/O bufs
- 2nd I/O thread drains/fills unused I/O bufs in parallel

- so the IOs and f(x) computation are in parallel
- why available: the parallelism of disk



Optimization for Streaming - Double Buffering pt 2

- approach: each thread do their job independently, when they finish their current job, they swap for new buffers.



Sorting: 2-Way

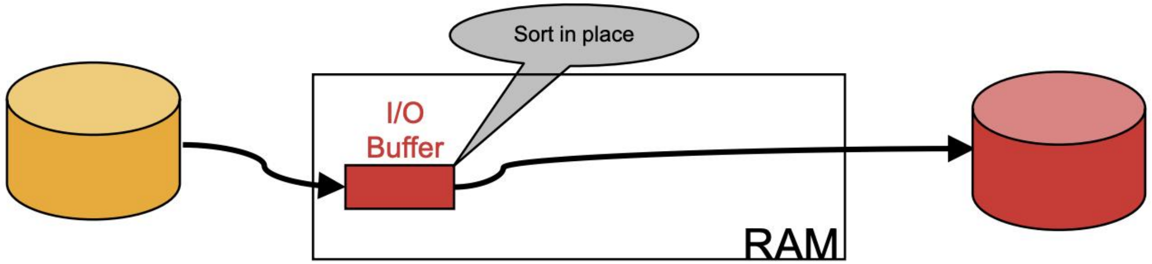
- Pass 0 (conquer a batch):
  - read a page from disk, sort it in RAM, write it to disk.
  - only one buffer page is used in memory
  - a repeated “batch job”
- Pass 1, 2, 3, …, etc. (merge via streaming):
  - repeatedly pick min item of buffer 1,2 to buffer 3 until full
  - merge pairs of runs into runs twice as long
    - pass1 generates 2-batch-long runs
    - pass2 generates 4-batch-long runs, etc.
  - a streaming algorithm, no double buffering
  - Drain/fill buffers as the data streams through them

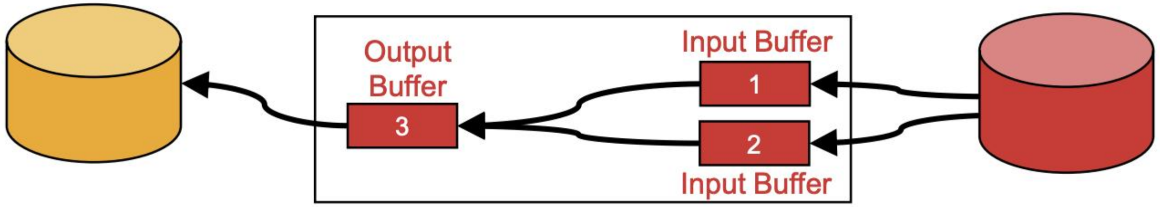
#passes:

$$= \lceil \log_2 N \rceil + 1$$

Total IOs:

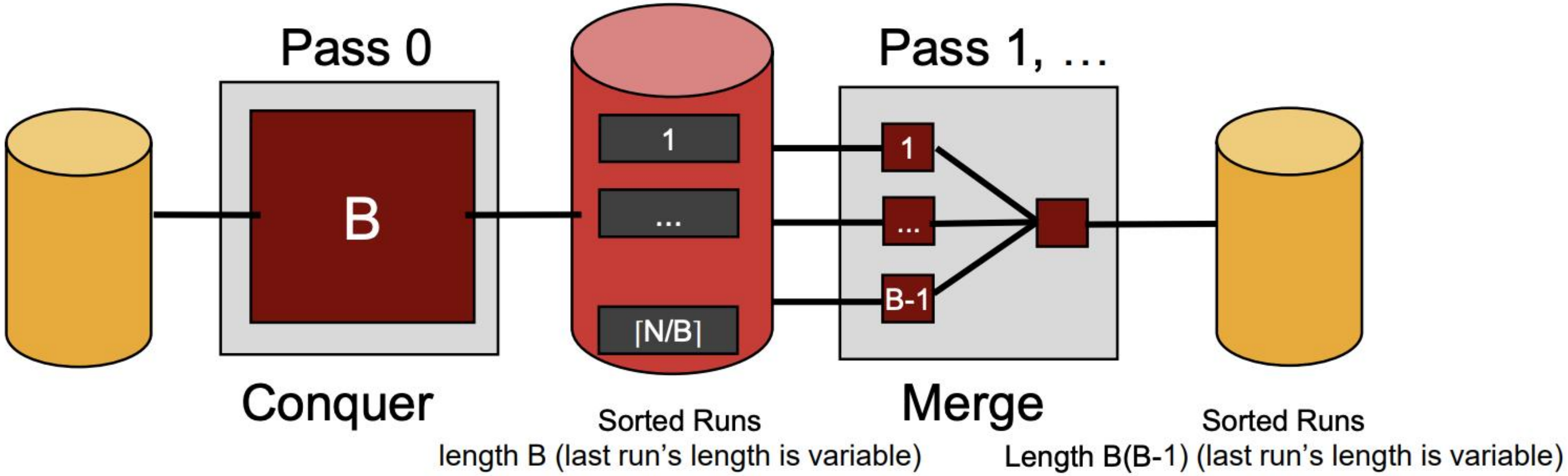
$$2N(\lceil \log_2 N \rceil + 1)$$





General External Merge Sort

- B buffer pages in total
  - Pass 0: use B buffer pages. Produce $\lceil N / B \rceil$ sorted runs of B pages each.
  - Pass 1, 2, …, etc.:
    - repeatedly pick min item of buffer 1, 2, ..., B-1 to buffer B untill full
    - merge B-1 runs at a time.
    - pass1 generates B(B-1)-batch-long runs, etc.


Pass 0
B
Conquer
1
...
⌈N/B⌉
Sorted Runs
length B (last run's length is variable)
Pass 1, …
1
...
B-1
Merge
Sorted Runs
Length B(B-1) (last run's length is variable)

#passes:

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

Total IOs:

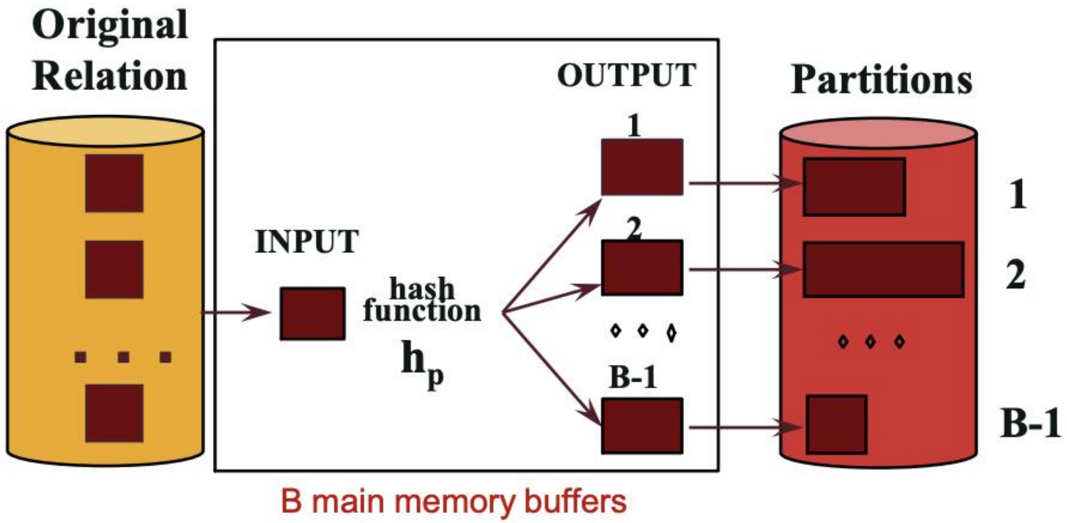
$$2*N * (1 + \lceil \log_{B-1} \lceil N / B \rceil \rceil)$$

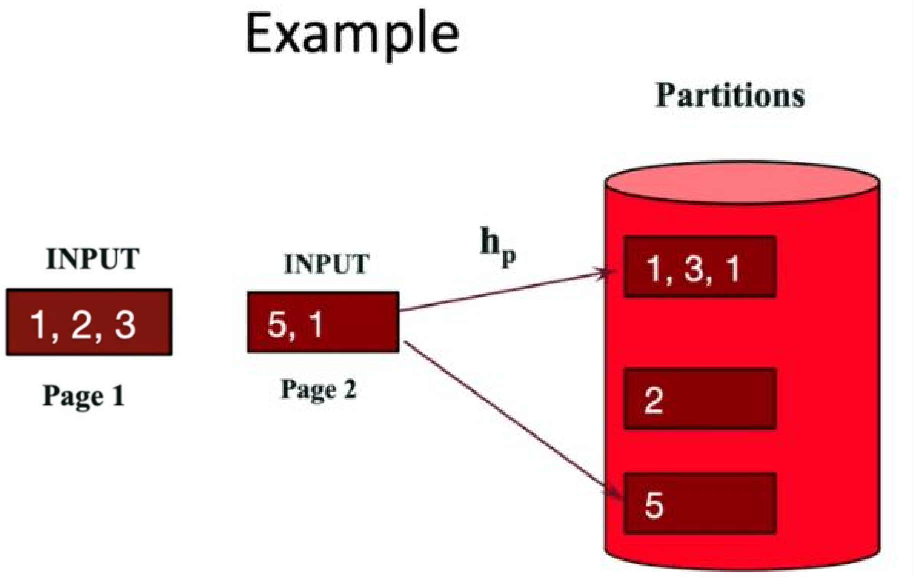
Memory Requirement: B(B-1).

Divide and Conquer in Hashing

- Streaming Partition (divide):
- Use a hash function h_p to stream records to disk partitions
  - All matches rendezvous in the same partition.
  - Each partition a mix of values
  - Streaming alg to create partitions on disk:
    - “Spill” partitions to disk via output buffers
  - Note:
    - same value item may not be contiguous (see later conquer)
    - pages from output after partition can be more than pages from input

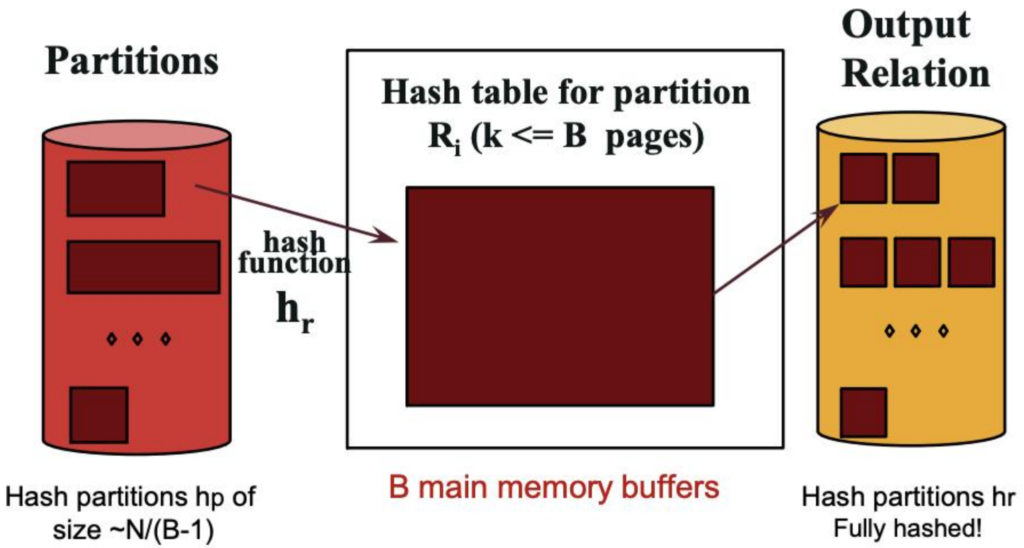
Divide and Conquer in Hashing pt 2



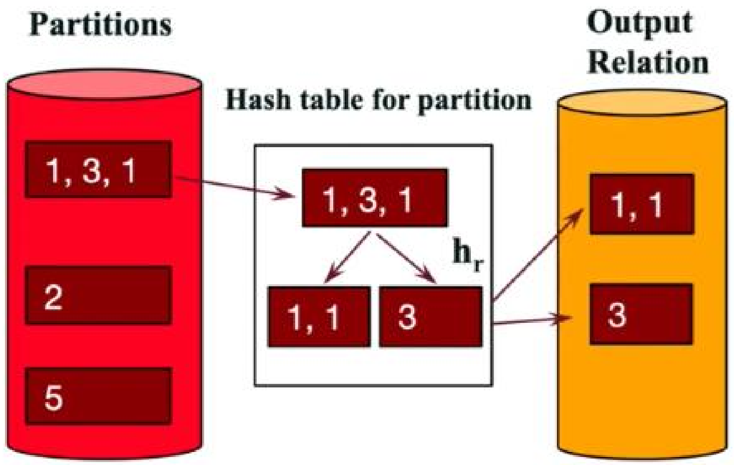


Divide and Conquer in Hashing pt 3

- conquer - rehash:
    - Read partitions into RAM hash table one at a time, using different hash function h_r
        - Each bucket contains a small number of distinct values
    - Then read out the RAM hash table buckets and write to disk
        - Ensuring that duplicate values are contiguous (same value same bucket)



Example

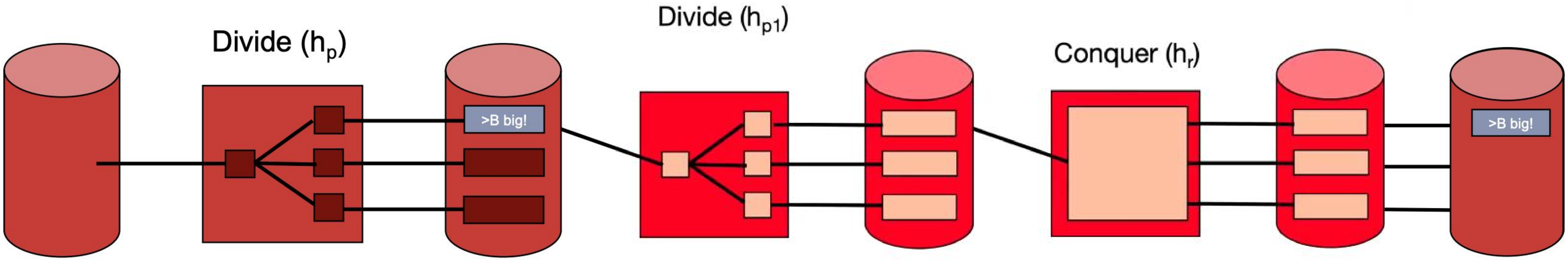


Cost of 2-pass external hashing

- cost ~ 2*N*(#passes) = 4*N IOs
- memory requirement: B(B-1)

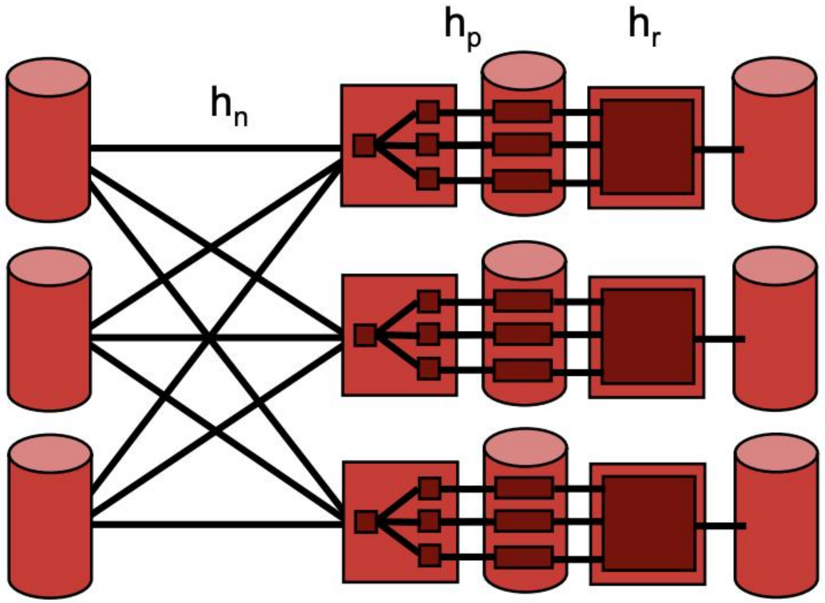
- Duality: the cost of 2-pass external hashing equals to the cost of 2-pass external sorting
  - Hashing: Divide & Conquer
  - Sorting: Conquer & Merge

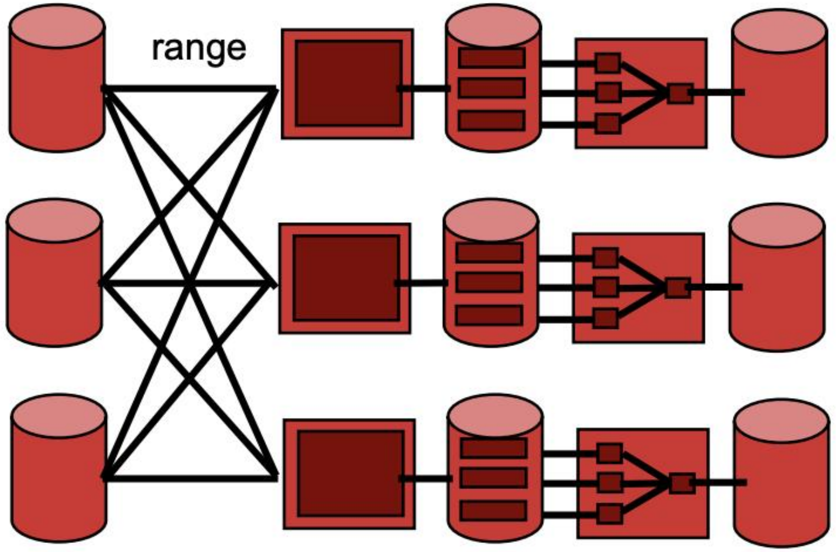
Recursive Partitioning



Parallelism of Hashing and Sorting

- 1 scan of entire data set in parallel across all machines
- cost of 2-pass with n machines: 4N/n IOs
- range seperation for sorting in parallel
    - avoid skew: estimate the distribution





- The question form may appear in exams
- Quiz Overview