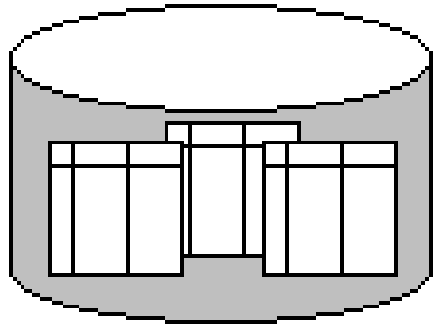
ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ

Συστήματα Διαχείρισης Βάσεων Δεδομένων

Διάλεξη 5η: External sorting

Δημήτρης Πλεξουσάκης
Τμήμα Επιστήμης Υπολογιστών

EXTERNAL SORTING



Sorting

- A classic problem in computer science!
- Data requested in sorted order (sorted output)
  - e.g., find students in increasing grade point average (gpa) order
  - `SELECT A,B, C FROM R ORDER BY A`
- Sorting is first step in bulk loading B+ tree index
- Sorting useful for eliminating duplicates in a collection of records (Why?)
  - `SELECT DISTINCT A,B, C FROM R`
- Some operators rely on their input files being already sorted, or, more often than not, sorted input files boost some operators' performance
  - Sort-merge join algorithm involves sorting

Sorting

- A file of records is sorted with respect to sort key k and ordering θ, if for any two records r1, r2 with r1 preceding r2 in the file, we have that their corresponding keys are in θ-order:
  - `r1 θ r2 <=> r1.k θ r2.k`

- A key may be a single attribute as well as an ordered list of attributes. In the latter case, order is defined lexicographically
  - Example: `k = (A,B), θ = <:`

    `r1 < r2 <=> r1.A < r2.A or`

    `r1.A = r2.A and r1.B < r2.B`

External Sorting

- Definition: Data lives on disk!
  - external vs. internal sort: the collection of data items to be sorted is not stored in main memory
- External Sorting is a challenge even if data << memory
  - The challenge is to overlap disk I/Os with sorting in memory
  - Most benchmarks (see next slide) are of this type, since memory is so cheap
- Examples in textbooks: data >> memory
  - These are classical examples, when memory was expensive, and are still common
  - Why not use virtual memory?

External Sorting Benchmarks

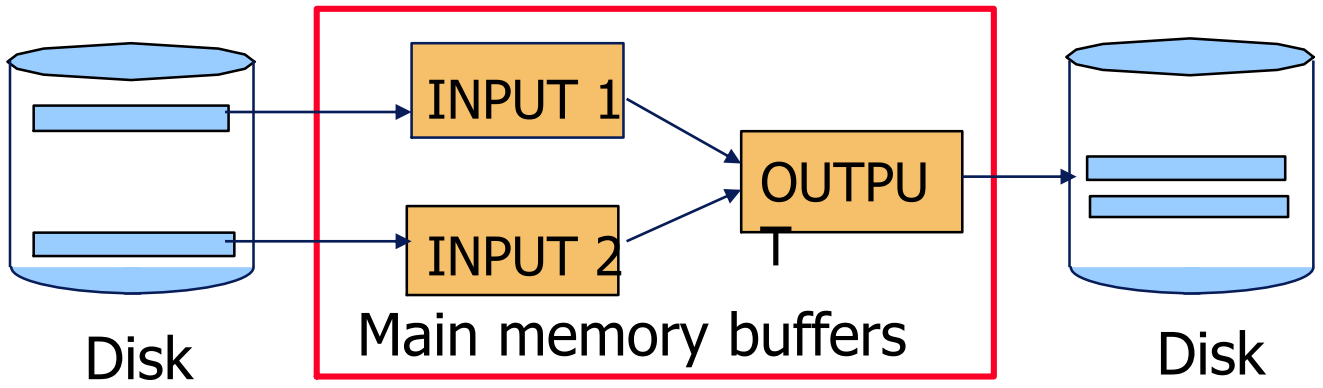
- Sorting has become a blood sport!
  - Sort Benchmarks is the name of the game (http://sortbenchmark.org/)
- How fast we can sort 1M records of size 100 bytes?
  - Typical DBMS: 5 minutes
  - World record: 1second - Deprecated
    - DCOM, cluster of 16 dual 400MHz Pentium II
- New benchmarks proposed:
  - Minute Sort: How many can you sort in 1 minute?
    - Typical DBMS: 10MB (~100,000 records)
    - Current world record: 1.42 TB
      - 2100 nodes, 2 2.3Ghz hexcore Xeon E5-2630, 64 GB memory, 12x3TB disks
  - Penny Sort: How many can you sort for a penny's worth of system time?
    - Current world record: 334 GB
    - 2.7 Ghz AMD Sempron, 4 GB RAM,
    - 5x320 GB 7200 RPM Samsung SpinPoint F4 HD332GJ, Linux

External Sorting Example

- Sort a relation in increasing order of the sort key values (under the assumption that data >> memory)
  - relation R: 10.000.000 tuples
  - one of the fields in each tuple is the sort key (not necessarily a key)
  - Records are of fixed length: 100 bytes; Total size of R: 1GB
  - Available main memory: 50MB
  - Block size: 4.096 (= 2^{12}) bytes
  - 40 records can fit in a block,
    - R occupies 250.000 blocks
  - Main memory can hold 12.800 blocks (= $50*2^{20}/2^{12}$)
- If data were kept in main memory, efficient sorting algorithms (e.g., Quicksort) could be employed to perform sorting on the sort keys
- This approach does not perform well for data in secondary storage:
  - need to move each block between secondary and main memory a number of times, in a regular pattern

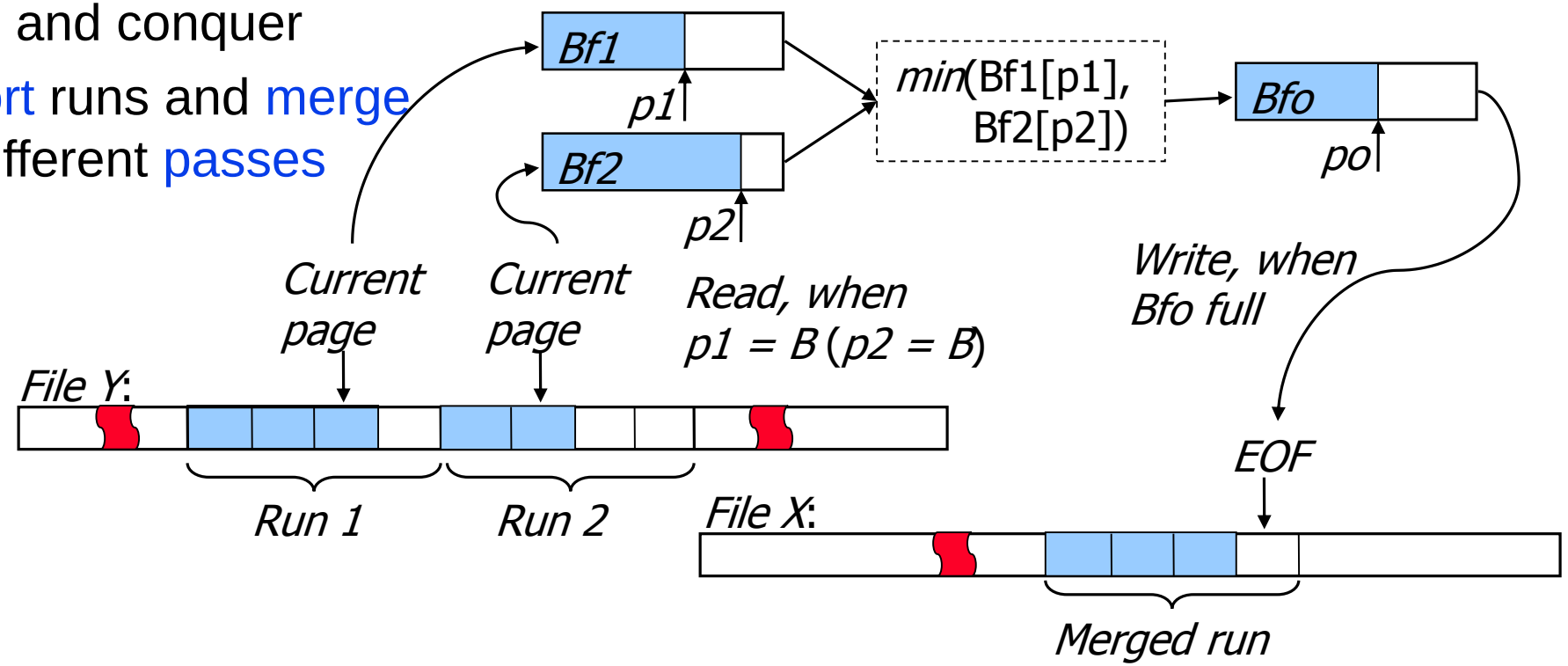
Two-Way Merge Sort

- Goal: even if the entire file does not fit into the available main memory, we can sort it by breaking it into smaller subfiles (called runs), sorting these subfiles and merging them into a larger subfiles using a minimal amount of main memory at any given time
- Idea: to merge sorted runs, repeatedly compare the smallest remaining keys of each run and output the record with the smaller key, until one of the runs is exhausted
- Two-way Merge Sort: requires 3 buffers
  - Pass 0: Read a page (one after the other), sort it, write it
    - only one buffer page is used
  - Pass 1, 2, 3, …, etc.: sort runs and merge
    - three buffer pages are used



Two-Way Merge Sort

- Divide and conquer
  - sort runs and merge in different passes



- Pass 0 writes 2^s sorted runs to disk, only one page of buffer space is used
- Pass 1 writes $2^s/2 = 2^{s-1}$ runs to disk, three pages of buffer space used
- Pass n writes $2^s/2^n = 2^{s-n}$ runs to disk, three pages of buffer space used
- Pass s writes a single sorted run (i.e., the complete sorted file) of size $2^s = N$ to disk

Cost of Two-Way Merge Sort

- In each pass we read all N pages in the file, sort/merge, and write N pages out again
- *N* pages in the file => the number of passes (S)

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

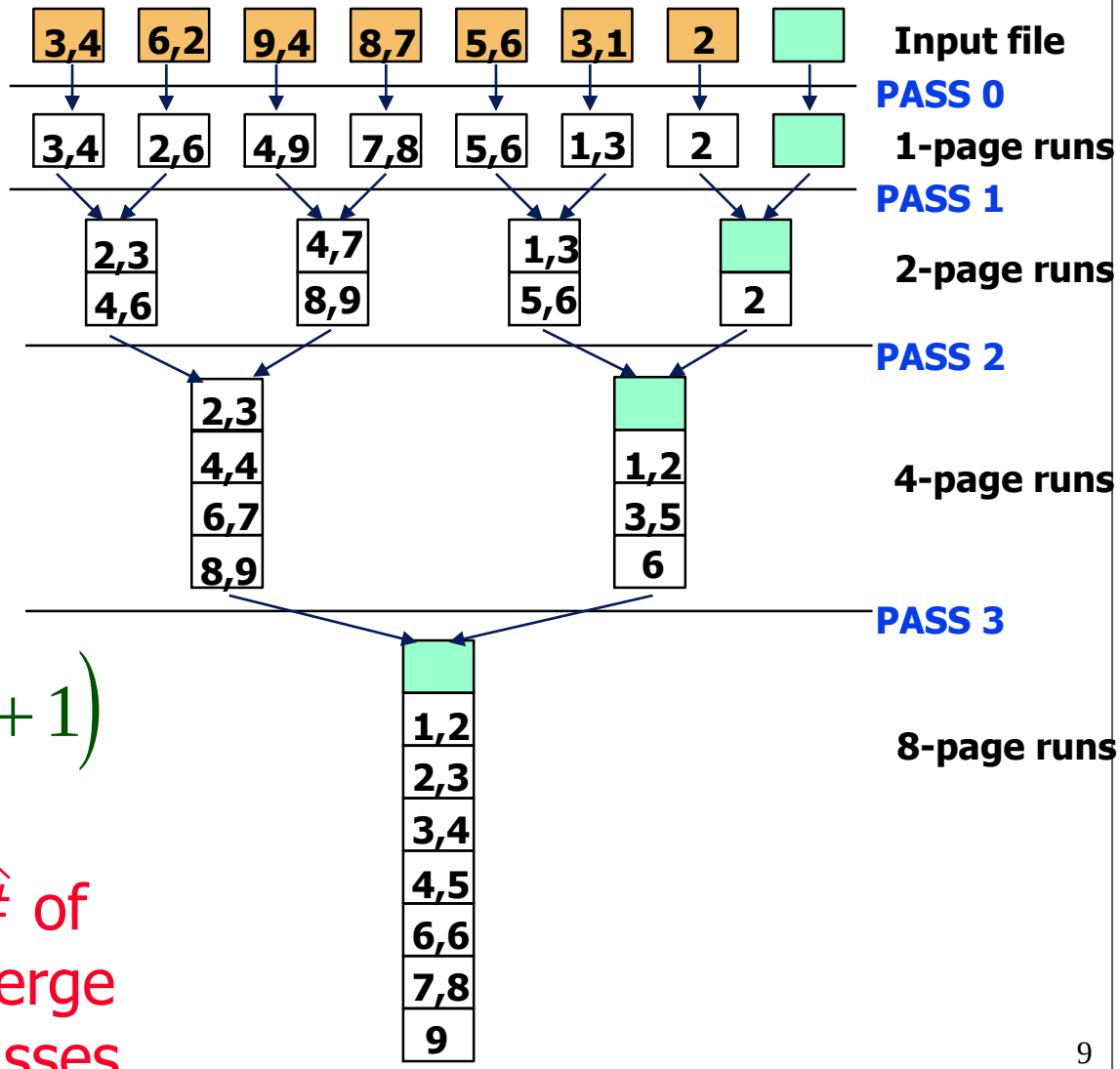
- So total cost is:

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

1 read & 1 write

# of pages

# of merge passes



Multi-way Merge Sort

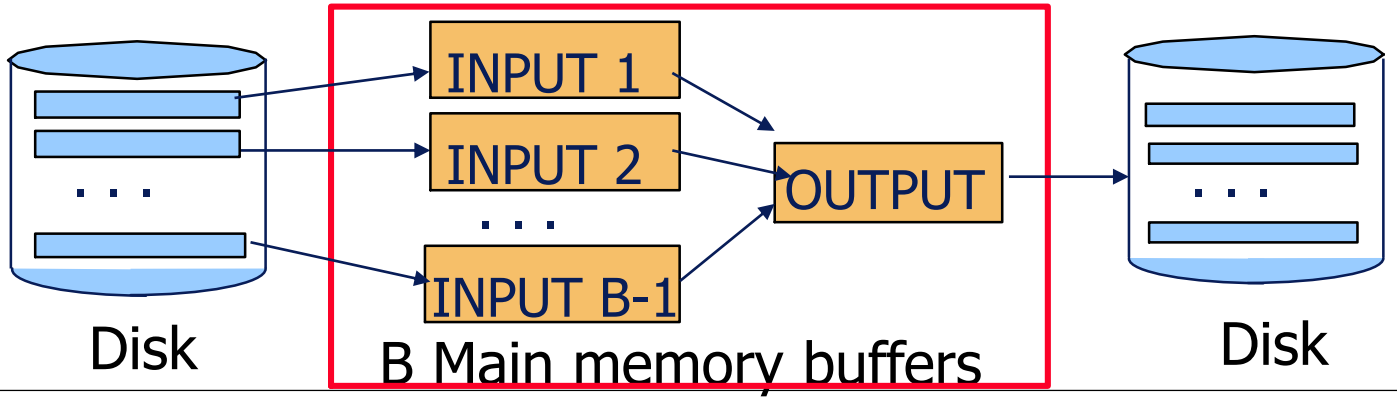
- Plain two-way merge sort algorithm uses no more than three pages of buffer space at any point in time
  - How we can use more than 3 buffer pages?

- (External) Multi-way Merge Sort aims at two improvements:
  - Try to reduce the number of initial runs (avoid creating one-page runs in Pass 0),
  - Try to reduce the number of passes (merge more than 2 runs at a time)

- As before, let
  - N denote the number of pages in the file to be sorted and
  - B buffer pages shall be available for sorting

Multi-way Merge Sort

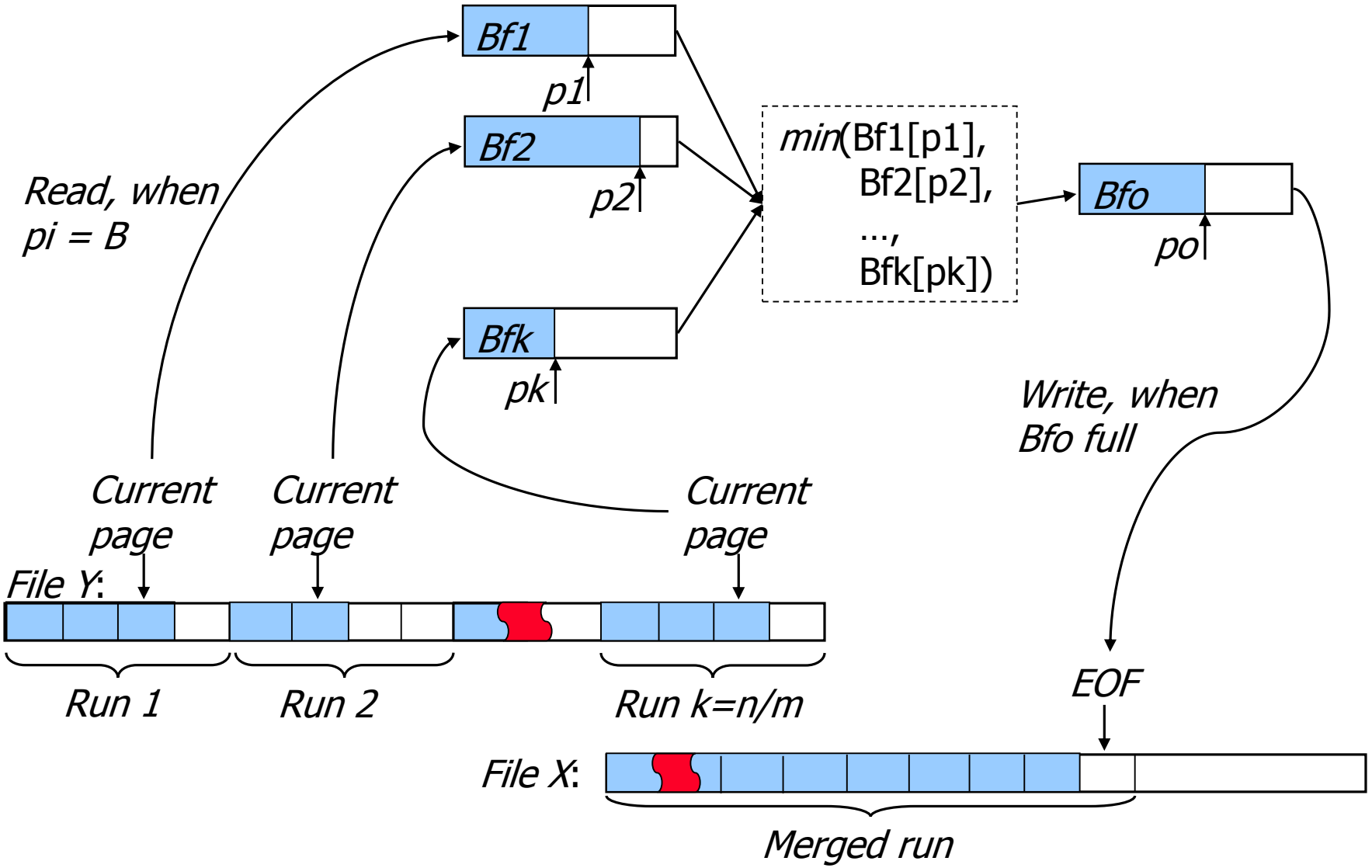
- To sort a file with *N* pages using *B* buffer pages:
  - Pass 0 use B buffers:
    - Read input data in B pages at the time and produce $\lceil N / B \rceil$ sorted runs of B pages each
      - Last run may contain fewer pages
  - Pass 1, 2, …, (until only a single run is left) use B-1 buffers for input and 1 for output:
    - Select B − 1 runs from previous pass
    - (B-1)-way merge in each pass
      - Read each run into an input buffer; one page at a time
      - Merge the runs and write to output buffer
      - Force output buffer to disk one page at the time



Multi-way Merge Sort

- Merging phase
  - Merge groups of B-1 runs at a time to produce longer runs until only one run (containing all records of input file) is left
    - Read the first page of each sorted run into an input buffer
  - Use a buffer for an output page holding as many elements of the first elements in the generated sorted run (at each pass) as it can hold
- Runs are merged into one sorted run as follows:
  - Find the smallest key among the first remaining elements of all the runs
  - Move the smallest element to the first available position in the output buffer
  - If the output buffer is full, write it to disk and empty the buffer to hold the next output page of the generated sorted run (at each pass)
  - If the page from which the smallest element was chosen has no more records, read the next page of the same run into the same input buffer
    - if no blocks remain, leave the run's buffer empty and do not consider this run in further comparisons

Multi-way Merge Sort

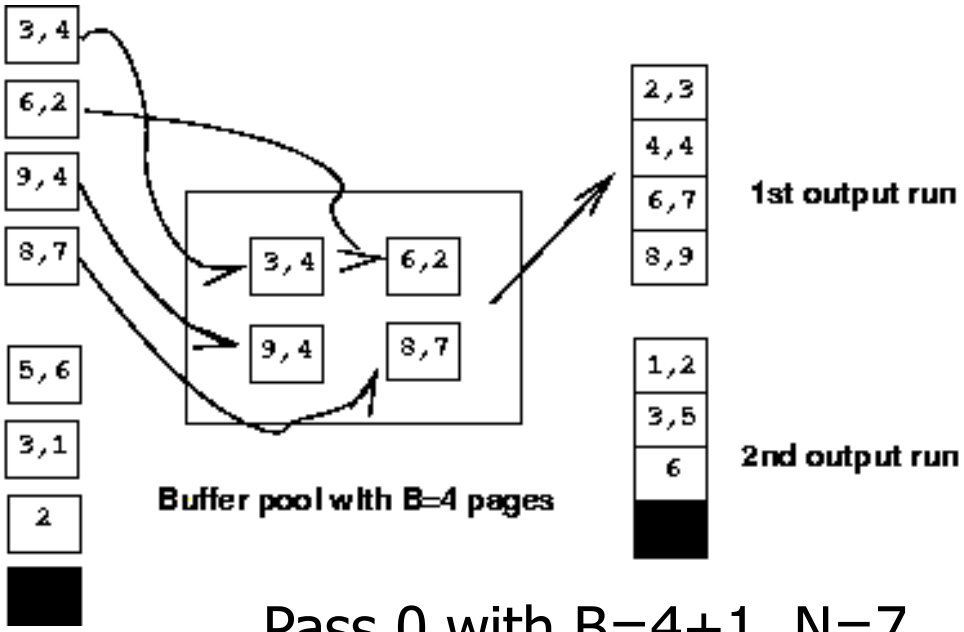


Cost of Multi-way Merge Sort

- 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: 2 sorted runs, 80 pages and 28 pages
  - Pass 3: Sorted file of 108 pages
- Number of passes:

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

- Cost = 2N * (# of passes)
  - per pass = N input + N output = 2 N



Pass 0 with B=4+1, N=7

Example (Cont'd)

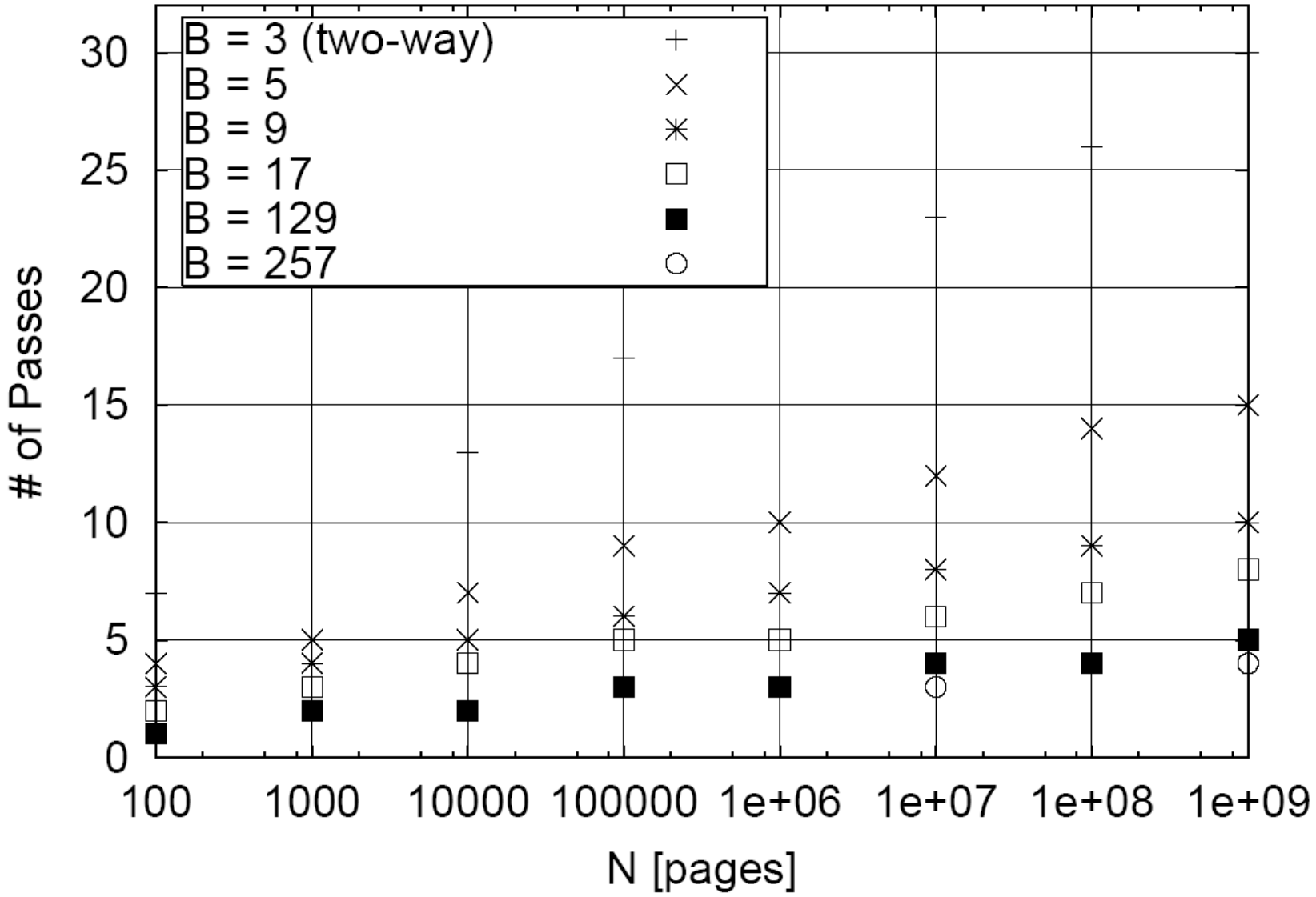
- According to the available number of buffers and the relation size we need 2 passes ($\lceil \log_{12799} \lceil 250.000/12.800 \rceil \rceil + 1 = 2$)
- Pass 0: sort main memory-sized portions of the data so that every record belongs to a run that fits in main memory
  - Need to create 20 runs (19*12.800 pages+ 1*6.800 pages = 250.000)
  - I/O required: 500.000 I/O ops
  - If each I/O op takes 15msec, we need 7.500 secs (125 mins) for phase 1
- Pass 1: merge the sorted runs into a single sorted run
  - run pages are read in an unpredictable order but exactly once
  - hence, 250.000 page reads are needed for phase 2
  - each record is placed only once in the output page
  - hence, 250.000 page writes are needed
  - phase 2 requires 125 mins
- In total, 250 minutes will be required for the entire sort of our relation

Multi-way Merge Sort: Number of Passes

N	B=3	B=5	B=9	B=17	B=129	B=257
100	7	4	3	2	1	1
1000	10	5	4	3	2	2
10000	13	7	5	4	2	2
100000	17	9	6	5	3	3
1000000	20	10	7	5	3	3
10000000	23	12	8	6	4	3
100000000	26	14	9	7	4	4
1000000000	30	15	10	8	5	4

I/O cost is 2N times number of passes

Multi-way Merge Sort I/O Savings



Minimizing the Number of Initial Runs

- Recall that the number of initial runs determines the number of passes we need to make:

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

(i.e., r = 0 . . . $\lceil N/B \rceil - 1$)

- Reducing the number of initial runs is a very desirable optimization for Pass 0
  - consider an alternative of QuickSort that minimizes the required number of passes by generating longer runs

- Replacement (tournament) Sort
  - Assume all tuples are the same size
  - Ignore for simplicity double buffering (more latter)

Replacement (Tournament) External Sort

- Replacement Sort:
  - Produce *runs* of 2*(B-2) pages long on average ("snowplow" analogy)
  - Assume one input and one output buffer; The remaining B − 2 buffer pages are called the current set

- Keep two heaps in memory, `H1` and `H2`

  `Top`: Read in B-2 blocks into `H1`

  `Output`: Move smallest record in `H1` to output buffer

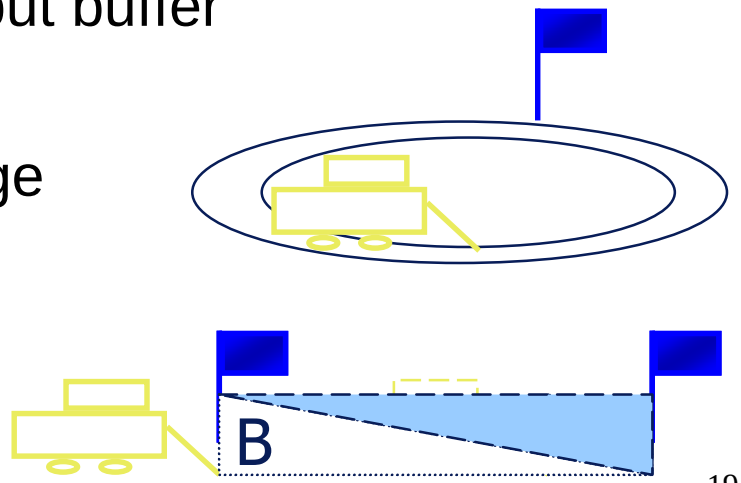
  Read in a new record `r` into `H1`

  If input buffer is empty, read another page

  If `r` not smallest, then GOTO `Output`

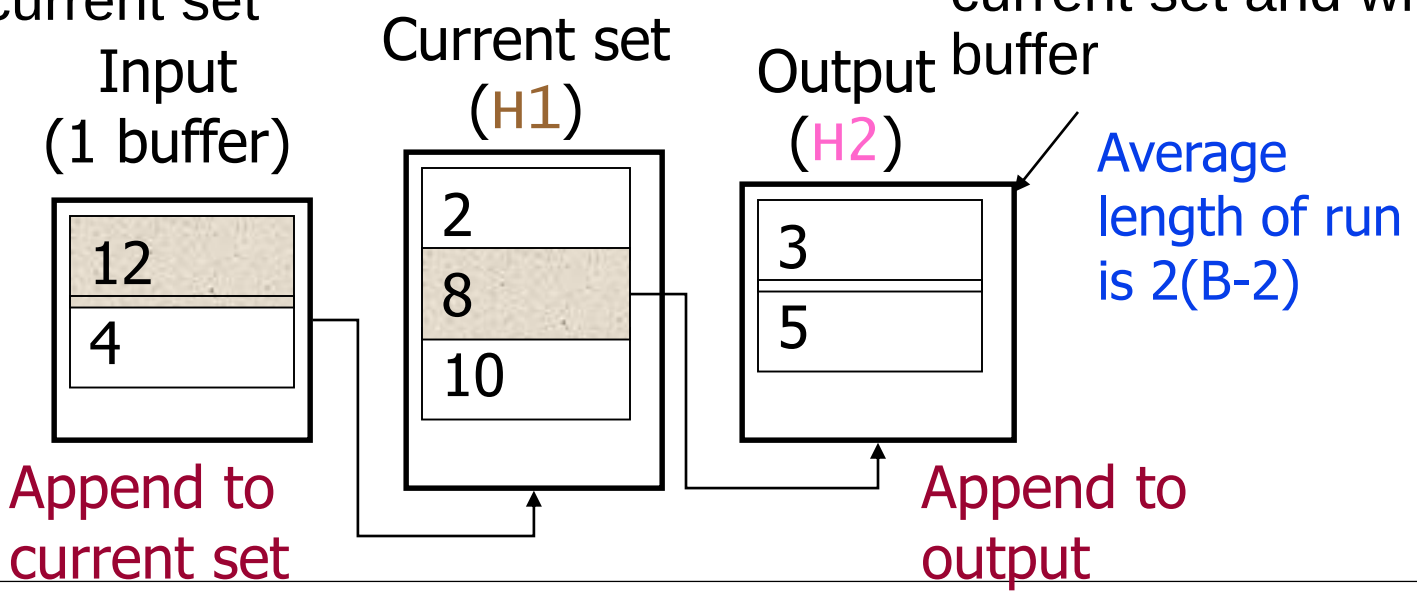
  else remove `r` from "heap"

  Output run `H2`; GOTO `Top`



Replacement (Tournament) External Sort

- Pick tuple in the current set with smallest k value that is still greater than the largest k value in output buffer
  - Append tuple to output buffer
  - Output buffer remain sorted
  - Add tuple from input buffer to current set
- Terminate when all tuples in current set are smaller than the largest tuple in output buffer
  - Write out output buffer page (it becomes the last page of the currently created run)
  - Start a new run by reading tuples from input buffer, moving them to current set and writing to output buffer



Number of Passes of Replacement Sort using Buffer Blocks

N	B=1000	B=5000	B=10000	B=50000
100	1	1	1	1
1000	1	1	1	1
10000	2	2	1	1
100000	3	2	2	2
1000000	3	2	2	2
10000000	4	3	3	2
100000000	5	3	3	2
1000000000	5	4	3	3

- Buffer block = 32 pages and initial pass produces runs of size 2(B-2)

I/O for External Merge Sort

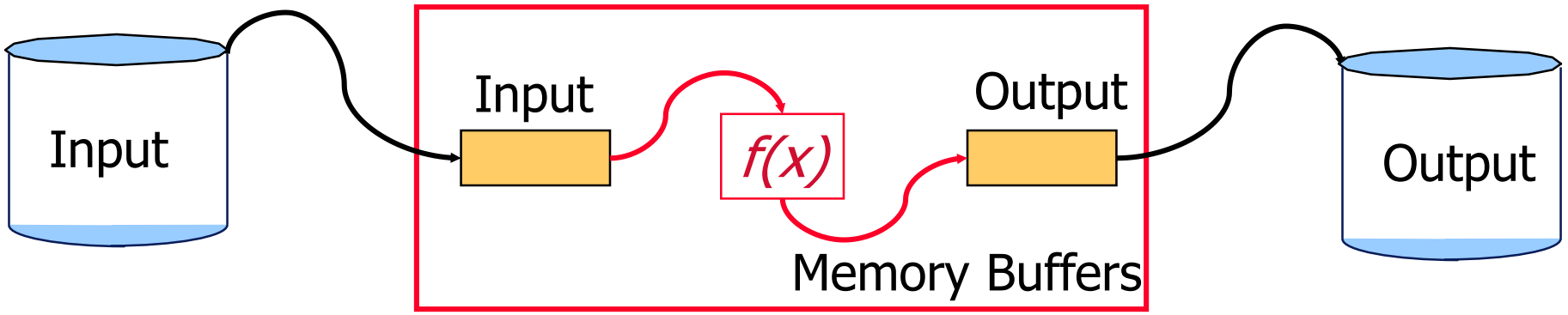
- Actually, in previous algorithms we considered simple page-by-page I/Os
  - Much better than an I/O per record !
- Transfer rate increases 40% per year; seek time and latency time decreases by only 8% per year
  - Is minimizing passes optimal for Pass 1, 2, …?
  - Would merging as many runs as possible the best solution?
- In fact, read a block of pages sequentially!
  - For minimizing seek time and rotational delay
- Suggests we should make each (input/output) buffer be a block of pages
  - But this will reduce fan-out during merge passes!
  - In practice, most files still sorted in 2-3 passes

Sequential vs Random I/Os for External Merge Sort

- Suppose we have 80 runs, each 80 pages long and we have 81 pages of buffer space
- We can merge all 80 runs in a single pass
  - each page requires a seek to access (Why?)
  - there are 80 pages per run, so 80 seeks per run
  - total cost = 80 runs * 80 seeks = 6400 seeks
- We can merge all 80 runs in two steps
  - 5 sets of 16 runs each
    - read 80/16=5 pages of one run
    - 16 runs result in sorted run of 1280 pages (16*80)
    - each merge requires 80/5 * 16 = 256 seeks
    - for 5 sets, we have 5 * 256 = 1280 seeks
  - merge 5 runs of 1280 pages
    - read 80/5=16 pages of one run => 1280/16=80 seeks in total
    - 5 runs => 5*80 = 400 seeks
  - total: 1280+400=1680 seeks!!!
- Number of passes increases, but number of seeks decreases!

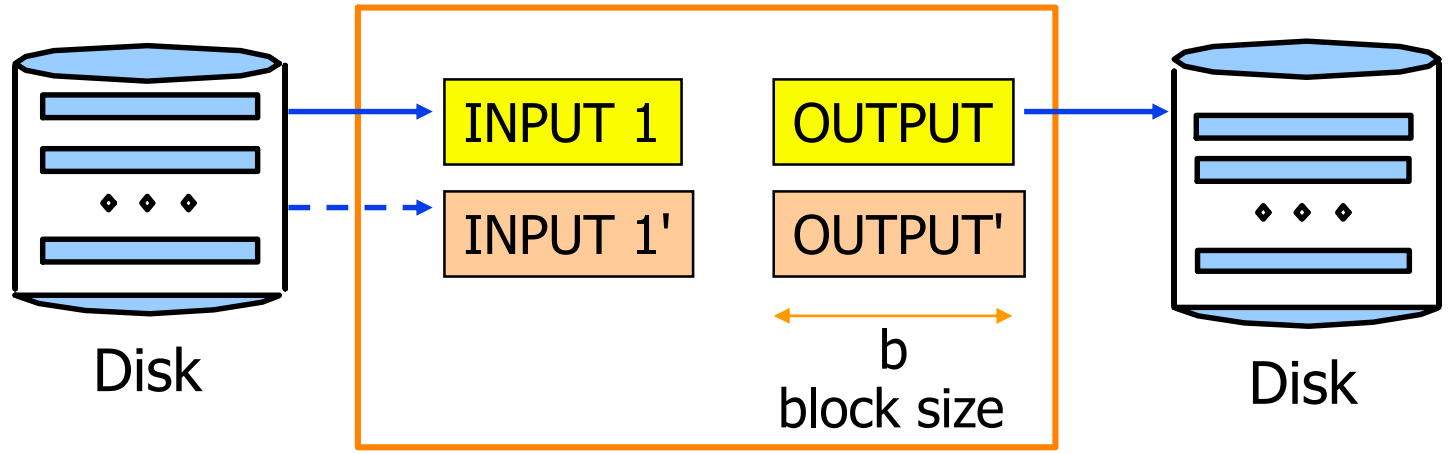
Streaming Data Through Main Memory

- An important detail for sorting & other DB operations
- Simple case: Compute $f(x)$ for each record, write out the result
  - Read a page from INPUT to Input Buffer
  - Write $f(x)$ for each item to Output Buffer
  - When Input Buffer is consumed, read another page
  - When Output Buffer fills, write it to OUTPUT
- Reads and Writes are not coordinated



Double Buffering

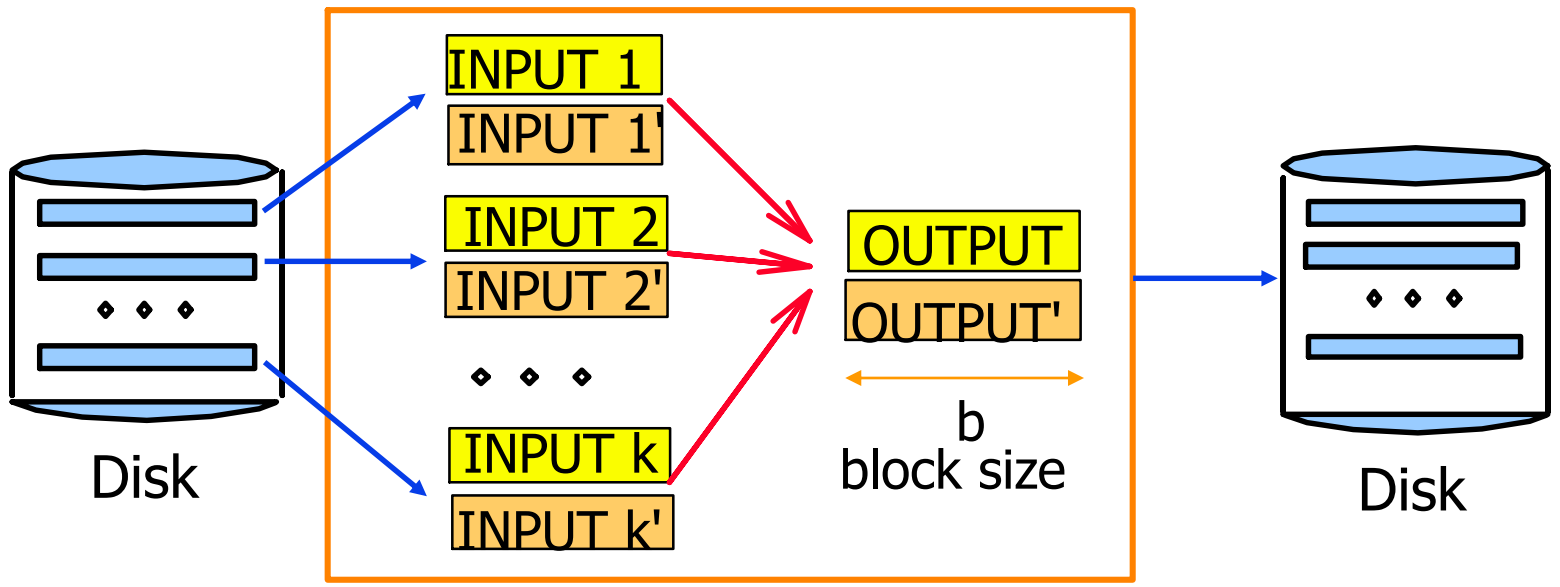
- Issue one read for 1024 bytes instead of 2 reads of 512 bytes (i.e. use a large buffer size)
  - A larger block allows more data to processed with each I/O
- To reduce wait time for I/O request to complete, can prefetch into `shadow block'
- The idea is to avoid waiting for input (or output) buffer while CPU is idle
  - Keep the CPU busy while the input buffer is reloaded (the output buffer is appended to the current run)



B main memory buffers, two-way merge

Double Buffering while Sorting

- Potentially, more passes (because you're effectively using fewer buffers); but, in practice, most files still sorted in 2-3 passes



B main memory buffers, multi-way merge

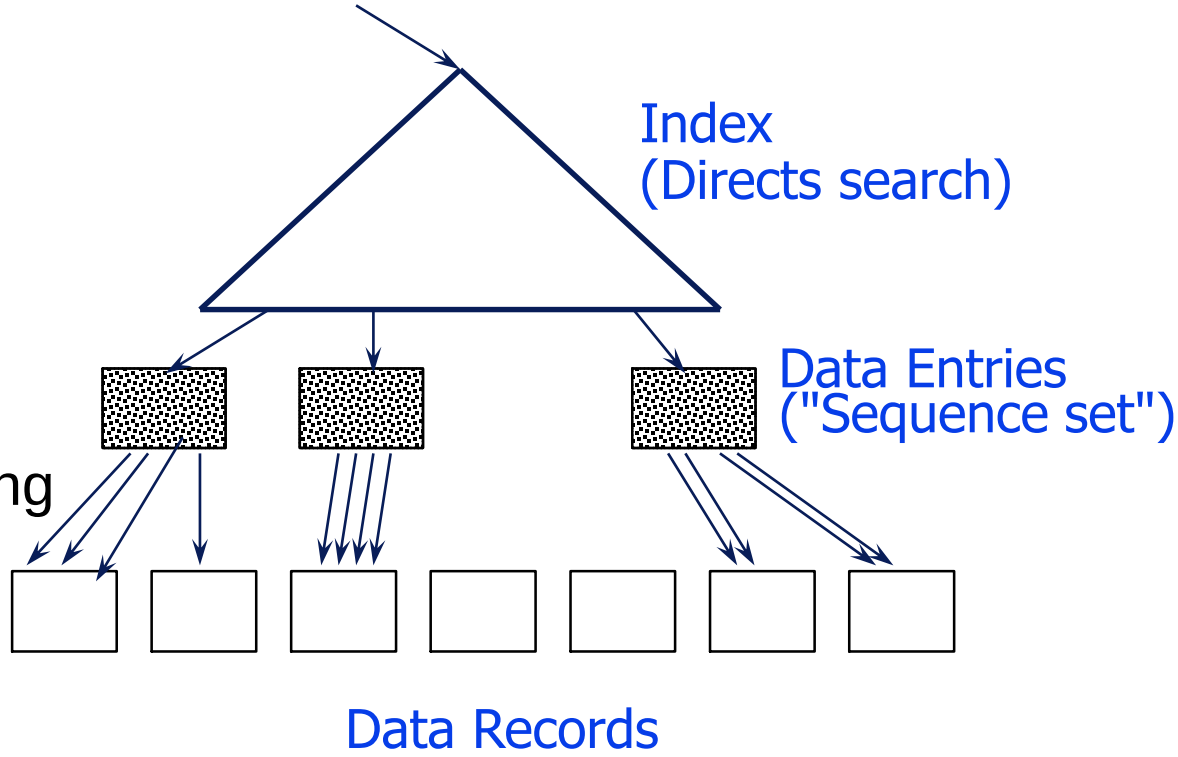
Using B+ Trees for Sorting

- Scenario: Table to be sorted has B^+ tree index on sorting column(s)
  - Idea: Can retrieve records in order by traversing leaf pages
- Is this a good idea?
- Cases to consider:
  - B+ tree is clustered -- Good idea!
  - B+ tree is not clustered -- Could be a very bad idea!

Clustered B+ Tree Used for Sorting

- Cost: root to the left-most leaf, then retrieve all leaf pages (<key,record> pair organization)

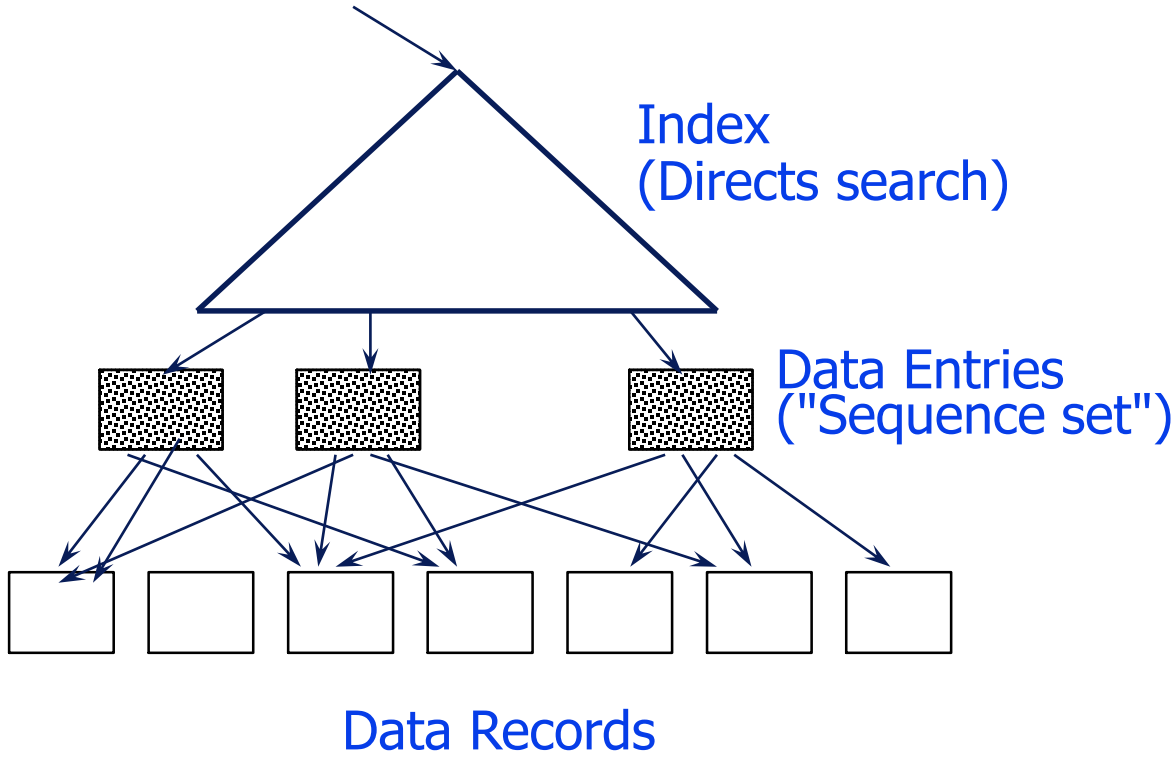
- If <key, rid> pair organization is used?
  - Additional cost of retrieving data records: each page fetched just once



Always better than external sorting!

Unclustered B+ Tree Used for Sorting

- each data entry contains <key, *rid*> of a data record
  - In the worst case, one I/O per data record!

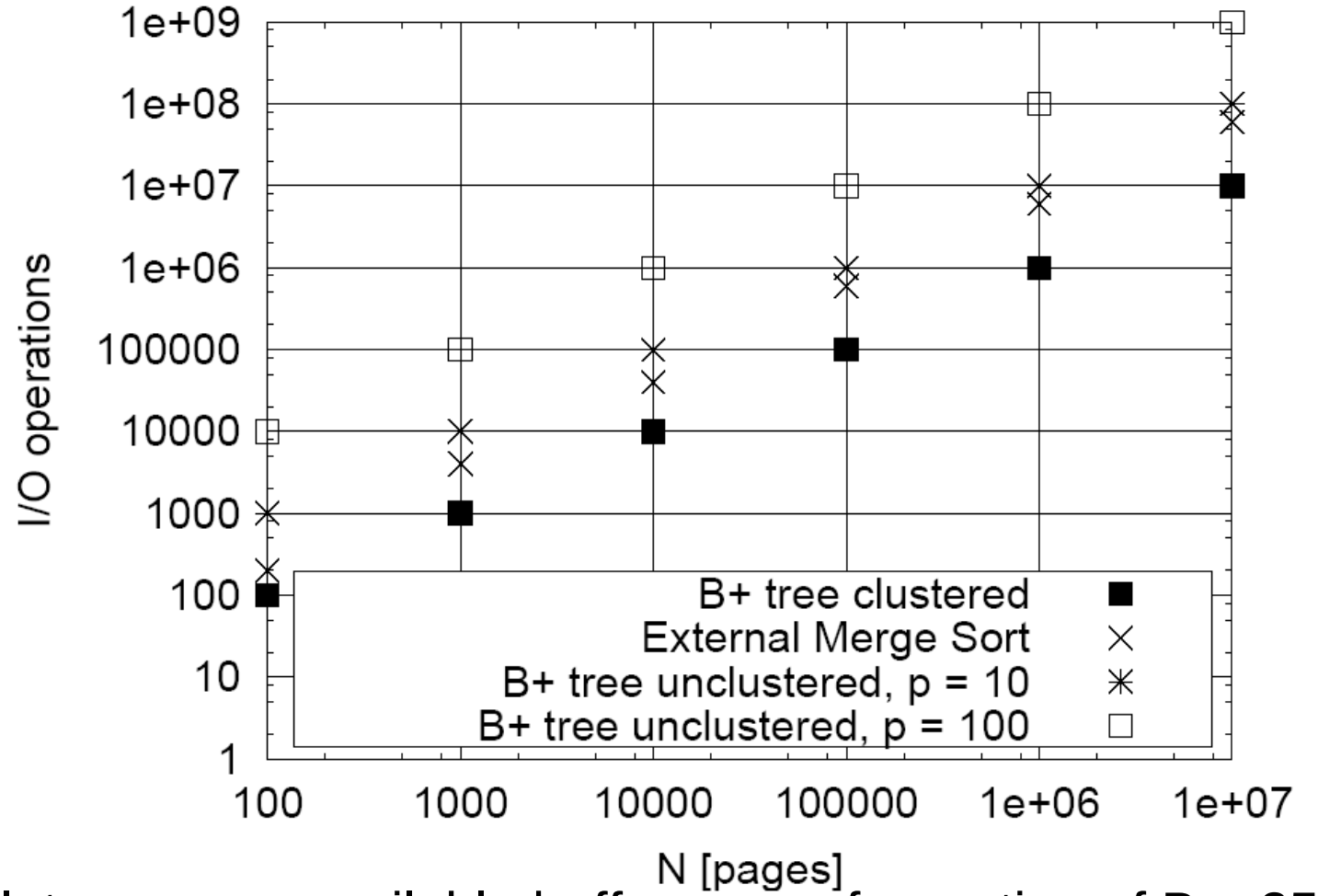


External Sorting vs. Unclustered Index

N	Sorting	p=1	p=10	p=100
100	200	100	1.000	10.000
1.000	2.000	1.000	10.000	100.000
10.000	40.000	10.000	100.000	1.000.000
100.000	600.000	100.000	1.000.000	10.000.000
1.000.000	8.000.000	1.000.000	10.000.000	100.000.000
10.000.000	80.000.000	10.000.000	100.000.000	1.000.000.000

p is the # of records per page (p = 100 is realistic)
B = 1000 and block buffer=32 pages for sorting
Cost : p * N (compared to N when index is clustered)

External Sorting vs. Unclustered Index



- The plot assumes available buffer space for sorting of B = 257 pages
- For even modest file sizes, therefore, sorting by using an unclustered B+ tree index is clearly inferior to external sorting

Summary

- External sorting is important
  - DBMS may dedicate part of buffer pool for sorting!
- External merge sort minimizes disk I/O cost:
  - Pass 0: Produces sorted runs of size **B** (# buffer pages). Later passes: merge runs
  - # of runs merged at a time depends on ***B***, and block size
  - Larger block size means less I/O cost per page
  - Larger block size means smaller # runs merged
  - In practice, # of runs rarely more than 2 or 3
- Choice of internal sort algorithm may matter:
  - Quicksort: Quick!
  - Replacement sort: slower (2x), longer runs
- The best sorts are wildly fast:
  - Despite 40+ years of research, we're still improving!
- Clustered B+ tree is good for sorting
  - unclustered tree is usually very bad

Complexity of Main Memory Sort Algorithms

	stability	space	time		
			best	average	worst
Bubble Sort	stable	little	O(n)	O(n2)	O(n2)
Insertion Sort	stable	little	O(n)	O(n2)	O(n2)
Quick Sort	untable	O(logn)	O(nlogn)	O(nlogn)	O(n2)
Merge Sort	stable	O(n)	O(nlogn)	O(nlogn)	O(nlogn)
Heap Sort	untable	little	O(nlogn)	O(nlogn)	O(nlogn)
Radix Sort	stable	O(np)	O(nlogn)	O(nlogn)	O(nlogn)

References

- Based on slides from:
  - R. Ramakrishnan and J. Gehrke
  - J. Hellerstein
  - M. H. Scholl

Τέλος Ενότητας

BY NC ND

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ & ΘΡΗΣΚΕΥΜΑΤΩΝ, ΠΟΛΙΤΙΣΜΟΥ & ΑΘΛΗΤΙΣΜΟΥ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ

ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Χρηματοδότηση

•Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.

•Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Κρήτης**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.

•Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ

ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Σημειώματα

Σημείωμα αδειοδότησης

•Το παρόν υλικό διατίθεται με τους όρους της άδειας χρήσης Creative Commons Αναφορά, Μη Εμπορική Χρήση, Όχι Παράγωγο Έργο 4.0 [1] ή μεταγενέστερη, Διεθνής Έκδοση. Εξαιρούνται τα αυτοτελή έργα τρίτων π.χ. φωτογραφίες, διαγράμματα κ.λ.π., τα οποία εμπεριέχονται σε αυτό και τα οποία αναφέρονται μαζί με τους όρους χρήσης τους στο «Σημείωμα Χρήσης Έργων Τρίτων».

[1] http://creativecommons.org/licenses/by-nc-nd/4.0/

•Ως **Μη Εμπορική** ορίζεται η χρήση:

–που δεν περιλαμβάνει άμεσο ή έμμεσο οικονομικό όφελος από την χρήση του έργου, για το διανομέα του έργου και αδειοδόχο

–που δεν περιλαμβάνει οικονομική συναλλαγή ως προϋπόθεση για τη χρήση ή πρόσβαση στο έργο

–που δεν προσπορίζει στο διανομέα του έργου και αδειοδόχο έμμεσο οικονομικό όφελος (π.χ. διαφημίσεις) από την προβολή του έργου σε διαδικτυακό τόπο

•Ο δικαιούχος μπορεί να παρέχει στον αδειοδόχο ξεχωριστή άδεια να χρησιμοποιεί το έργο για εμπορική χρήση, εφόσον αυτό του ζητηθεί.

.

Σημείωμα Αναφοράς

Copyright Πανεπιστήμιο Κρήτης, Δημήτρης Πλεξουσάκης. «**Συστήματα Διαχείρισης Βάσεων Δεδομένων. Διάλεξη 5η: External sorting**». Έκδοση: 1.0. Ηράκλειο/Ρέθυμνο 2015. Διαθέσιμο από τη δικτυακή διεύθυνση: http://www.csd.uoc.gr/~hy460/