INFO20003 Database Systems

Dr Renata Borovica-Gajic

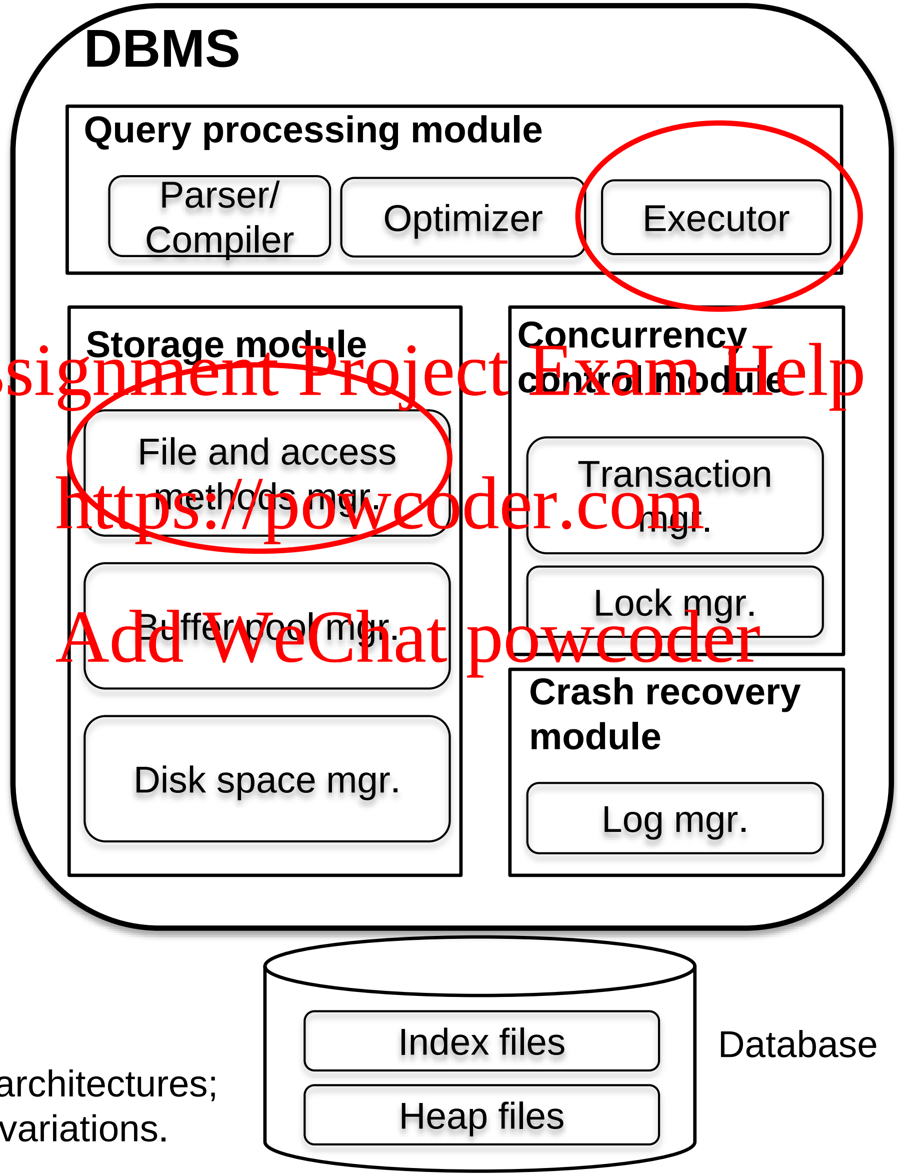
Lecture 11
Query Processing Part I

Week 6

Remember this? Components of a DBMS

TODAY & Next time

Will briefly touch upon …



This is one of several possible architectures; each system has its own slight variations.

Coverage

MELBOURNE

- Query Processing Overview
- Selections
- Projections

Readings: Chapter 12 and 14, Ramakrishnan & Gehrke, Database Systems

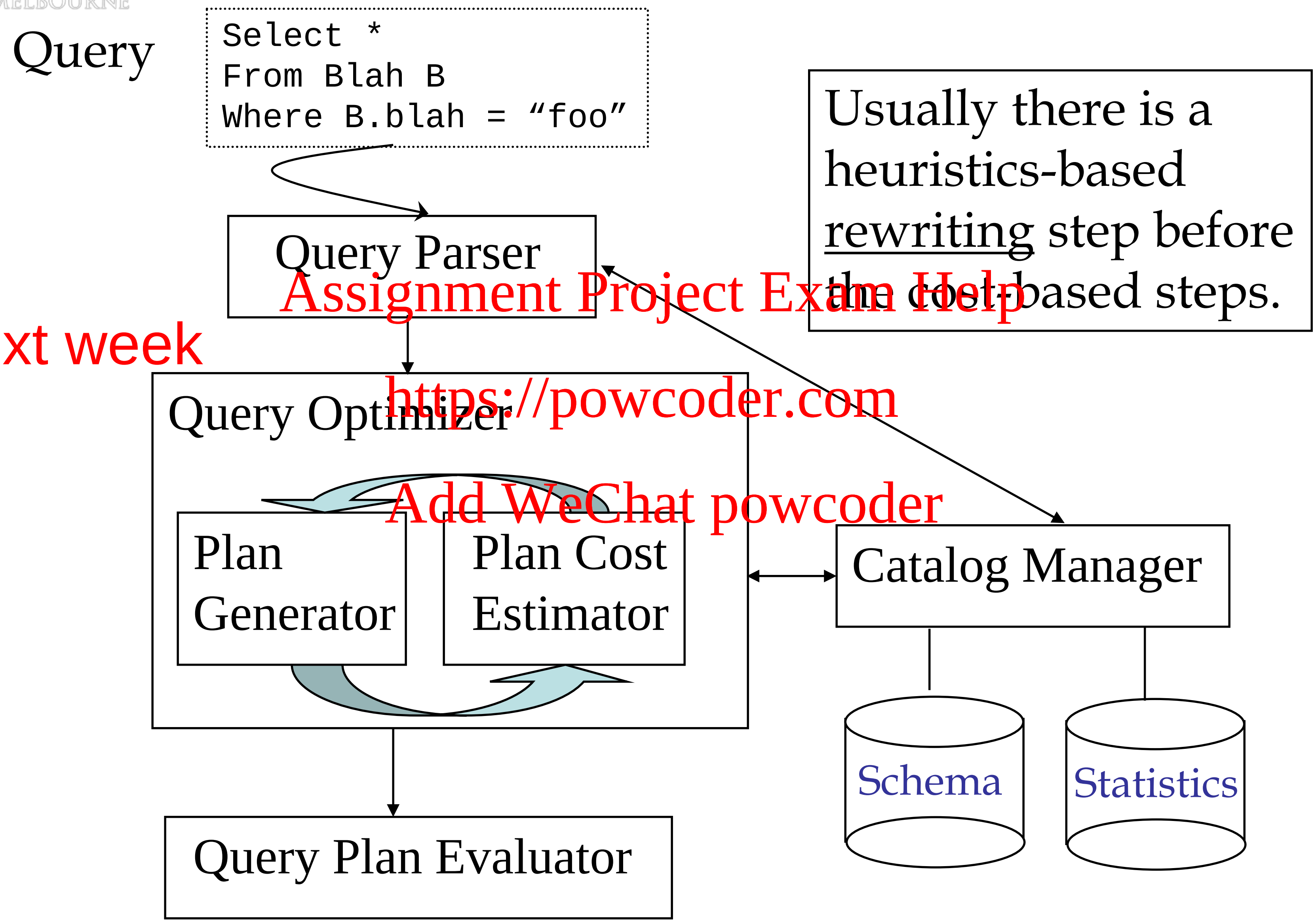
Query processing overview

- Some database operations are EXPENSIVE
- DBMSs can greatly improve performance by being ‘smart’
  – e.g., can speed up 1,000,000x over naïve approach

- Main weapons are:
  1. clever implementation techniques for operators
  2. exploiting ‘equivalencies’ of relational operators
  3. using cost models to choose among alternatives

Query processing workflow

Next week



Relational Operations

- We will consider how to implement:
  - *Selection* (σ) Selects a subset of rows from relation
  - *Projection* (π) Deletes unwanted columns from relation
  - *Join* ($\bowtie$) Allows us to combine two relations

- Operators can be then be *composed* creating *query plans*

Query Processing

MELBOURNE

- Query Processing Overview
- Selections
- Projections

Readings: Chapter 14, Ramakrishnan & Gehrke, Database Systems

Schema for Examples

Sailors (*sid*: integer, *sname*: string, *rating*: integer, *age*: real)
Reserves (*sid*: integer, *bid*: integer, *day*: dates, *rname*: string)

- **Sailors (S)**:
  - Each tuple is 50 bytes long, 80 tuples per page, **500 pages**

  - N = NPages(S) = 500, p_S=NTuplesPerPage(S) = 80
  - NTuples(S) = 500*80 = 40000

- **Reserves (R)**:
  - Each tuple is 40 bytes long, 100 tuples per page, **1000 pages**
  - M= NPages(R) = 1000, p_R=NTuplesPerPage(R) =100
  - NTuples(R) = 100000

THE UNIVERSITY OF MELBOURNE

Simple Selections

MELBOURNE

- Of the form $\sigma_{R.attr\ op\ value}(R)$
- Example:

```
SELECT *
FROM Reserves R
WHERE R.BID > 20;
```


- The best way to perform a selection depends on:
  1. available indexes/access paths
  2. expected **size of the result** (number of tuples and/or number of pages)

Estimate result size (reduction factor)

MELBOURNE

- **Size of result** approximated as:

  *size of relation * ∏ (reduction factors)*

- **Reduction factor** is usually called ***selectivity***. It estimates what portion of the relation will qualify for the given predicate, i.e. satisfy the given condition.
  - This is estimated by the optimizer (will be taught next week)
  - E.g. 30% of records qualify, or 5% of records qualify

Alternatives for Simple Selections

MELBOURNE

1. With no index, unsorted:
   –Must scan the whole relation, i.e. perform Heap Scan
   –**Cost = Number of Pages of Relation, i.e. NPages(R)**
   –**Example**: Reserves cost(R)= 1000 IO (1000 pages)

2. With no index, but file is sorted:

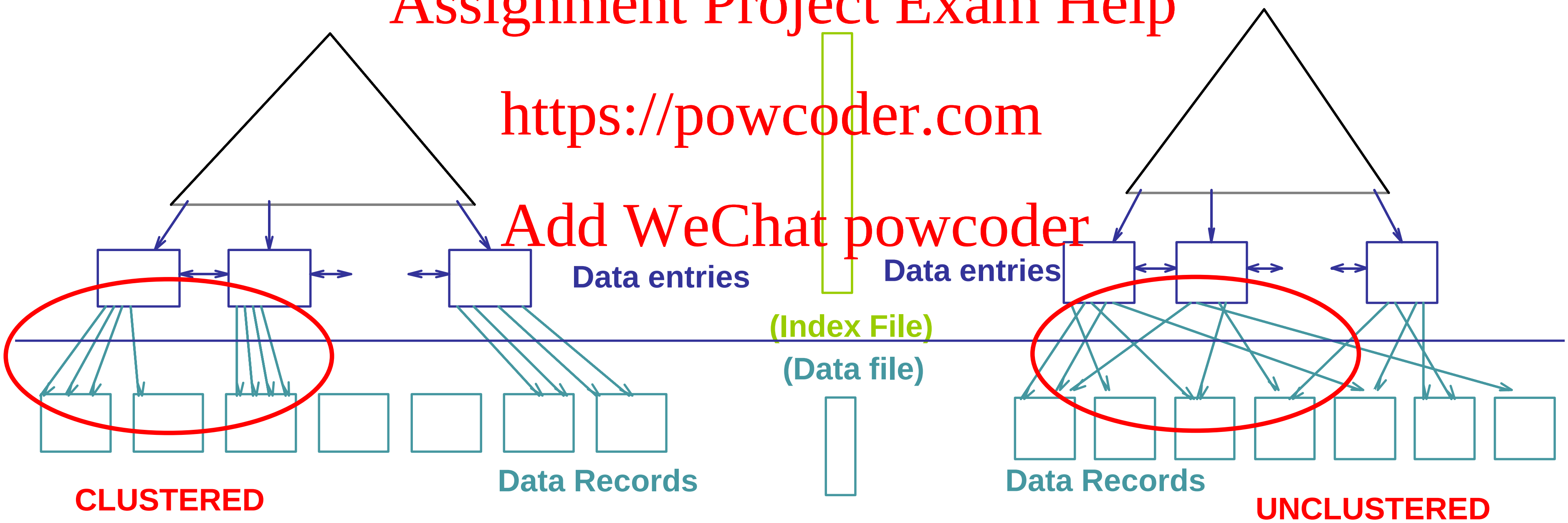
   –cost = **binary search cost + number of pages containing results**
   –**Cost = $\log_2$(NPages(R)) + (**RF*NPages(R)**)**

   –**Example**: Reserves cost(R)= 10 I/O + **(**RF*NPages(R)**)**

3. With an index on selection attribute:
   –Use index to find qualifying data entries,
   –Then retrieve corresponding data records
   –Discussed next….

Index Clustering: Review

Clustered vs. unclustered



Using an Index for Selections

- Cost depends on the number of qualifying tuples
- Clustering is important when calculating the total cost
- Steps to perform:

    1. Find qualifying data entries:

    – Go through the index: height typically small, 2-4 I/O in case of B+tree, 1.2 I/O in case of hash index (*negligible* if many records retrieved)

    – Once data entries are reached, go through data entries one by one and look up corresponding data records (in the data file)

    2. Retrieve data records (in the data file)

- **Cost:**

1. Clustered index**:**

    Cost = (NPages(I) + NPages(R))*RF

2. Unclustered index:

    Cost = (NPages(I) + NTuples(R))*RF

Our example

MELBOURNE

- **Example**: Let's say that 10% of Reserves tuples qualify, and let's say that index occupies 50 pages
- RF = 10% = 0.1, NPages(I) = 50, NPages(R) = 1000, NTuplesPerPage(R) = 100

- **Cost:**

1. Clustered index**:**

   Cost = (NPages(I) + NPages(R))*RF

   Cost = (50+ 1000)*0.1 = 105 (I/O) Cheapest access path

2. Unclustered index:

   Cost = (NPages(I) + NTuples(R))*RF

   Cost = (50+ 100000)*0.1 = 10005 (I/O)

3. Heap Scan:

   Cost = NPages(R) = 1000 (I/O)

General Selection Conditions

- Typically queries have multiple predicates (conditions)
- **Example**: day<8/9/94 AND rname='Paul' AND bid=5 AND sid=3

- A B-tree index matches (a combination of) predicates that involve only attributes in a **prefix of the search key**
  - Index on <a, b, c> matches predicates on: (a,b,c), (a,b) and (a)
  - Index on <*a, b, c*> matches *a=5 AND b=3*, but will not used to answer *b=3*
  - This implies that only reduction factors of predicates that are part of the prefix will be used to determine the cost (they are called matching predicates (or primary conjuncts))

Selections approach

1. Find the **cheapest access path**
   - An index or file scan with the **least estimated page I/O**

2. Retrieve tuples using it
   - Predicates that match this index reduce the number of tuples *retrieved (and impact the cost)*

3. Apply the predicates that don't match the index (if any) later on
   - These predicates are used to discard some retrieved tuples, but do not affect number of tuples/pages fetched (nor the total cost)
   - In this case selection over other predicates is said to be done "on-the-fly"

Cheapest Access Path: Example

- **Example**: day < 8/9/94 AND bid=5 AND sid=3

- A **B+ tree** index **on *day*** can be used;
  - –RF = RF(day)
  - –Then, *bid*=5 and *sid*=3 must be checked for each retrieved tuple *on the fly*
- Similarly, a **hash index on <*bid, sid*>** could be used;
  - –$\prod RF$ = RF(bid)*RF(sid)
  - –Then*, day<8/9/94* must be checked *on the fly*

- How about a B+tree on <rname,day>? (Y/N)
- How about a B+tree on <day, rname>? (Y/N)
- How about a Hash index on <day, rname>? (Y/N)

Query Processing

MELBOURNE

- Overview
- Selections
- Projections

Readings: Chapter 14, Ramakrishnan & Gehrke, Database Systems

The Projection Operation

- Issue with projection is removing duplicates

```
SELECT   DISTINCT   R.sid, R.bid
  FROM   Reserves R
```


- Projection can be done based on **hashing** or **sorting**

The Projection Operation

MELBOURNE

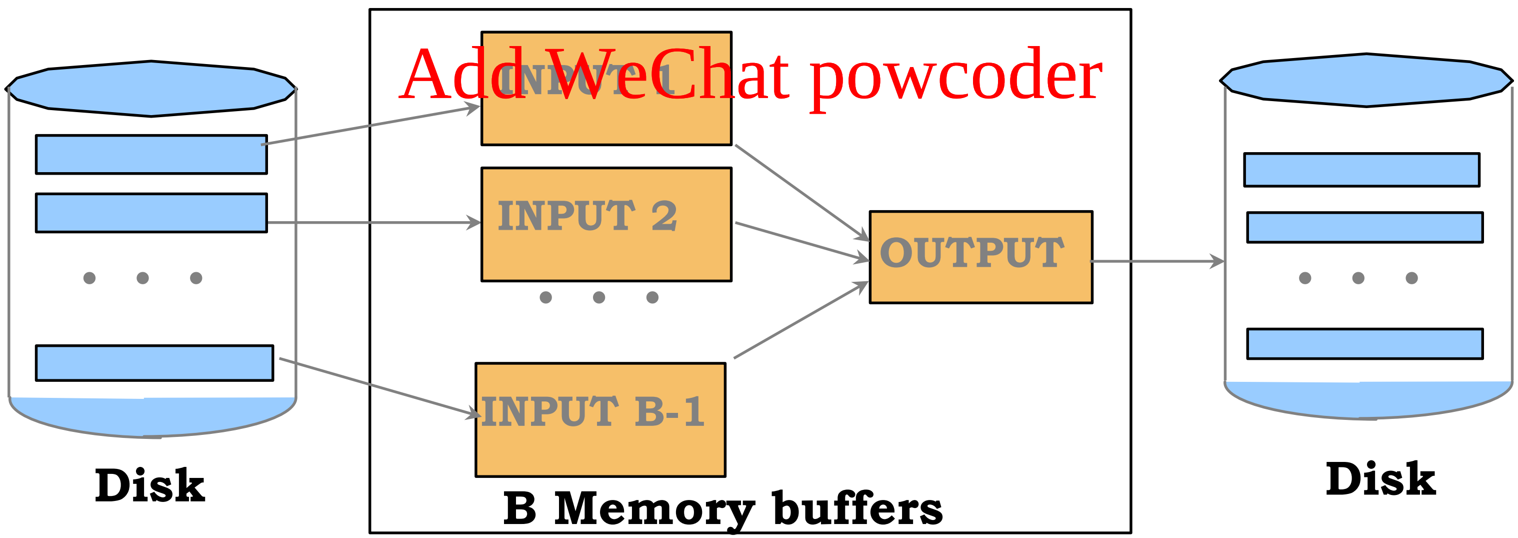
- Basic approach is to use **sorting**
  - –1. Scan R, extract only the **needed** attributes
  - –2. Sort the result set (typically using external merge sort)
  - –3. Remove **adjacent** duplicates

11,80
12,10
12,10
12,75
13,20
13,20
13,75

External Merge Sort

- **If data does not fit in memory do several passes**
- Sort runs: Make each B pages sorted (called runs)
- Merge runs: Make multiple passes to merge runs
  - Pass 2: Produce runs of length B(B-1) pages
  - Pass 3: Produce runs of length $B(B-1)^2$ pages
  - …
  - Pass P: Produce runs of length $B(B-1)^P$ pages

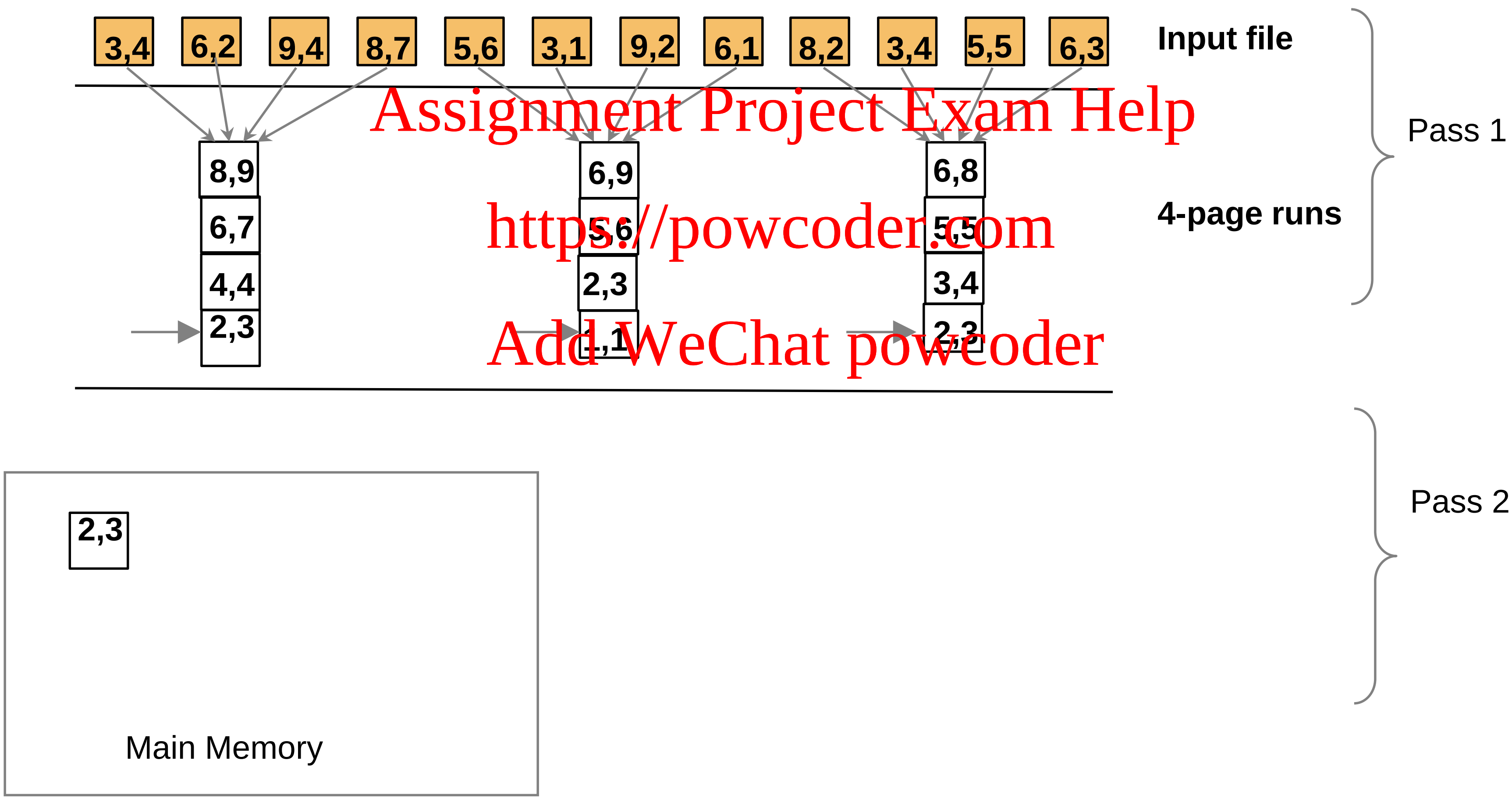
We will let you know how many passes there are



Readings: Chapter 13, Ramakrishnan & Gehrke, Database Systems

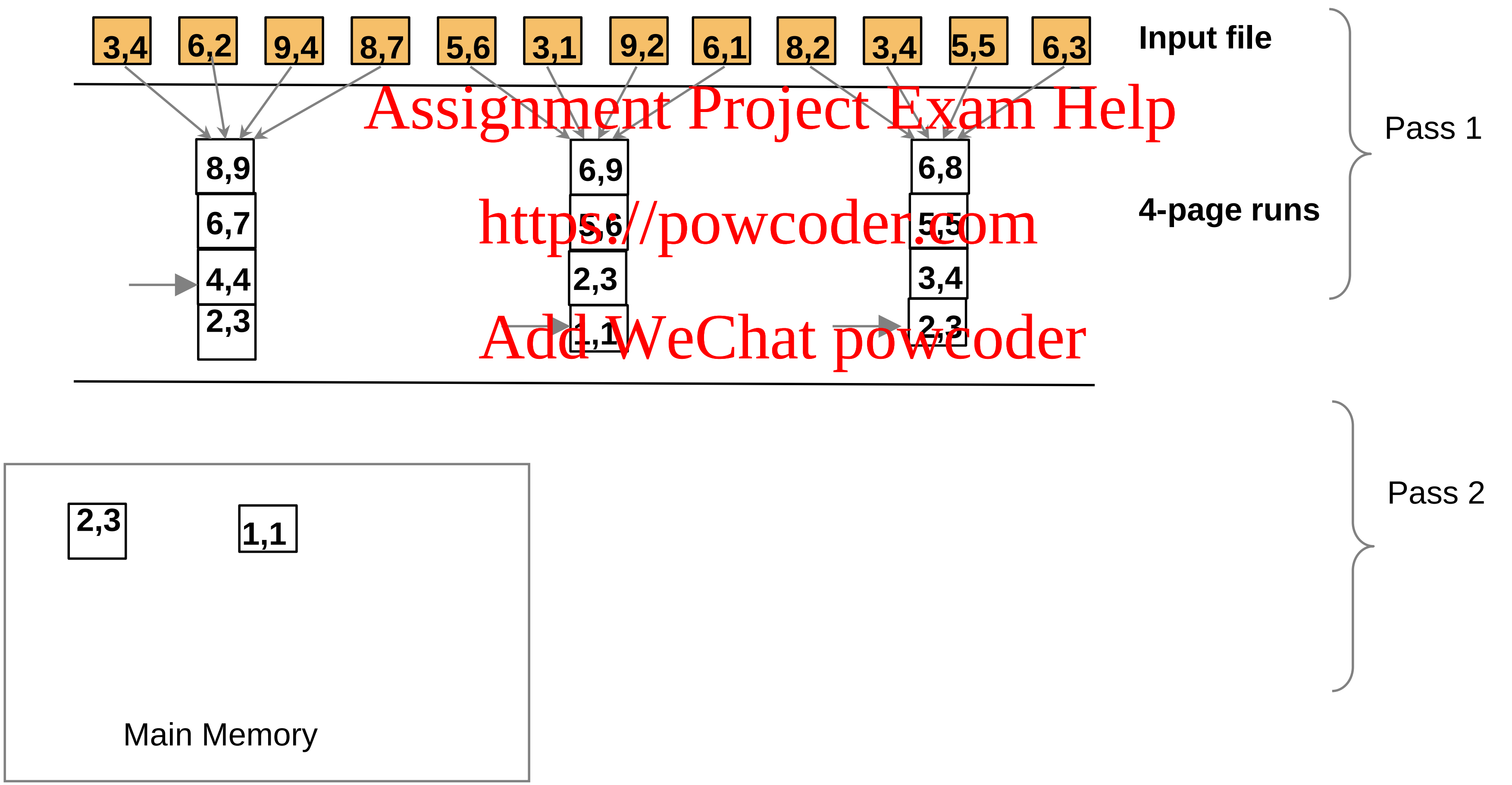
External Merge Sort: Example

buffer pages in memory B = 4, each page 2 records, sorting on a single attribute (just showing the attribute value)



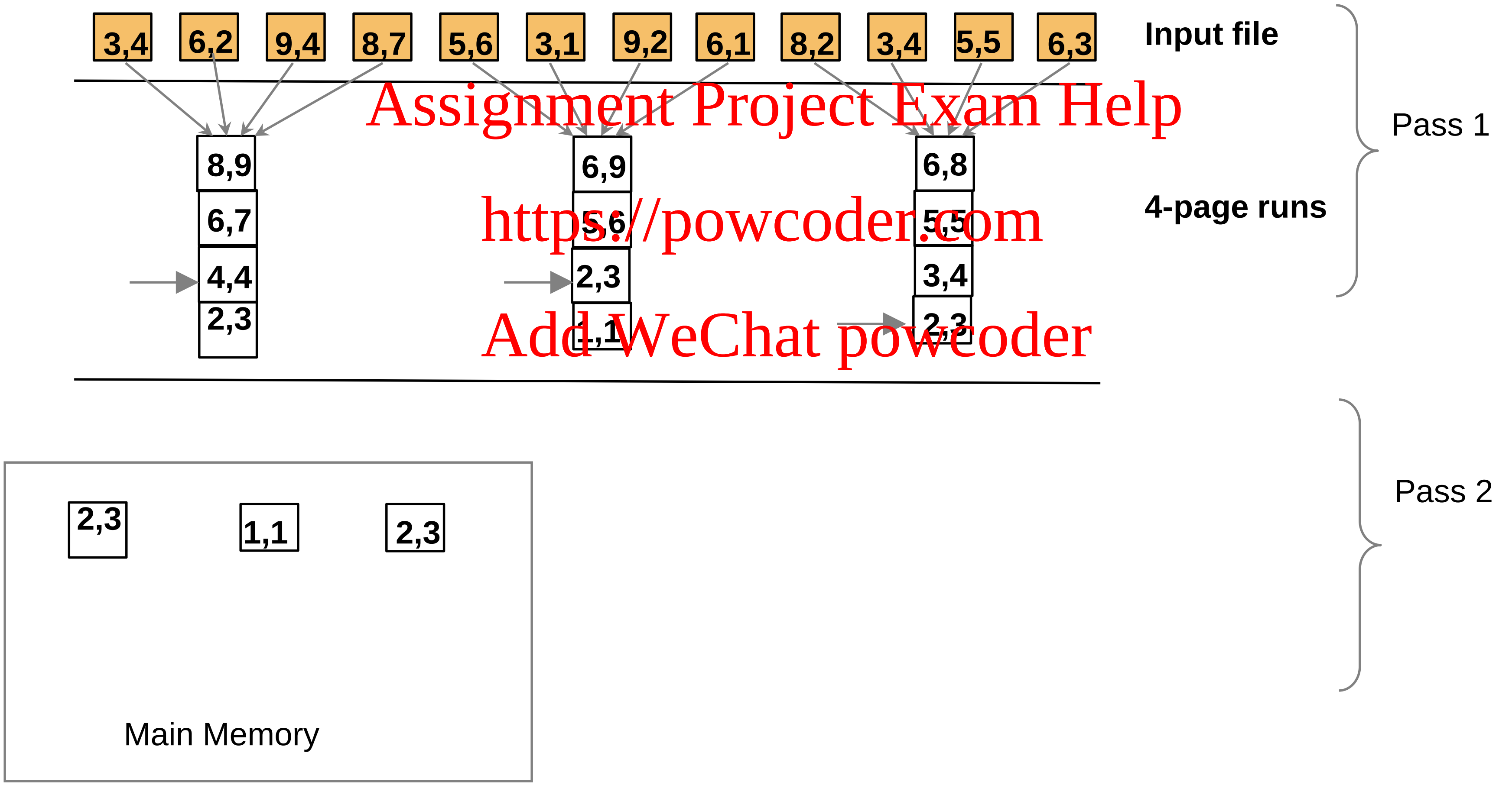
External Merge Sort: Example

buffer pages in memory B = 4, each page 2 records



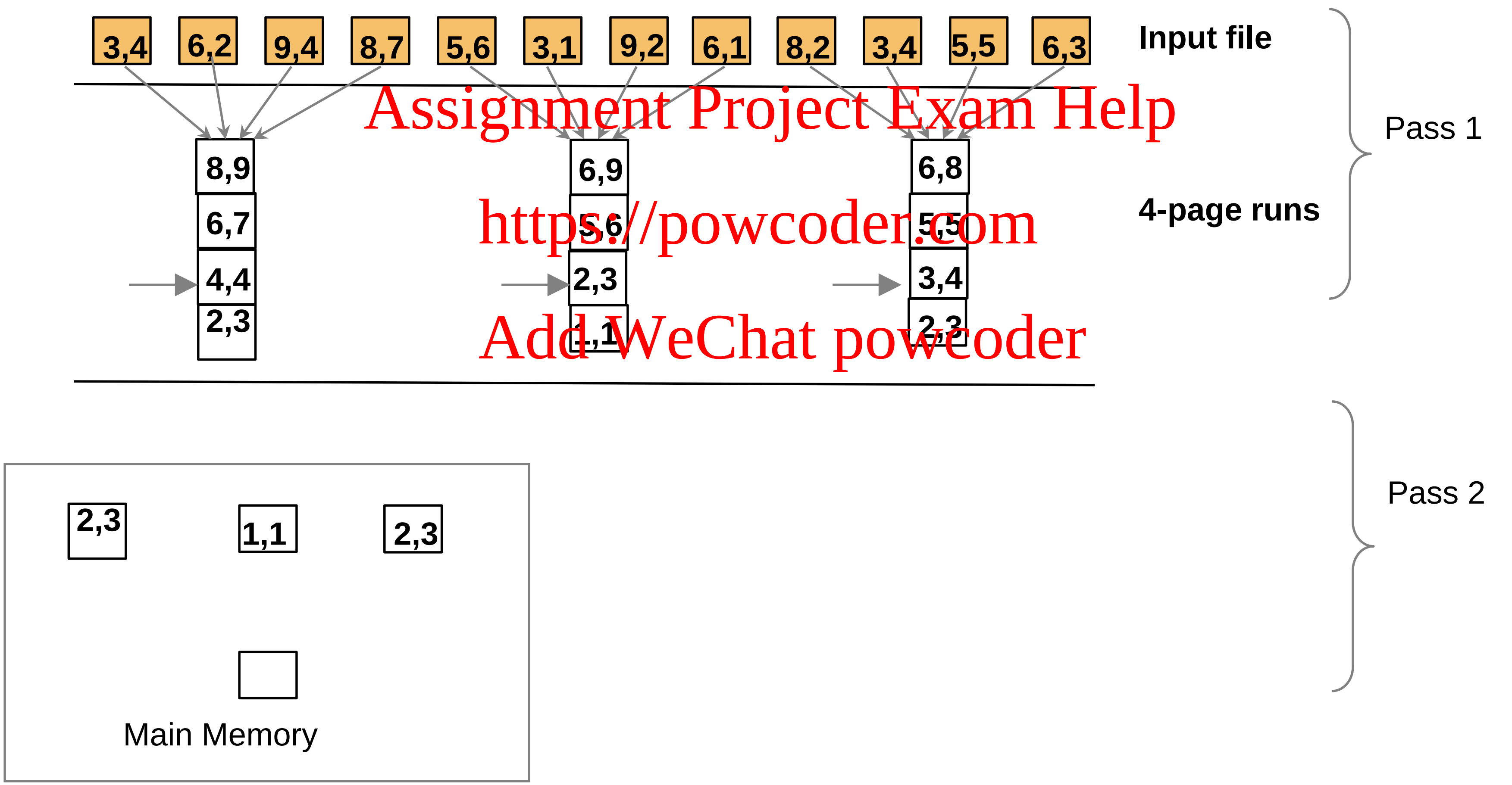
External Merge Sort: Example

buffer pages in memory B = 4, each page 2 records



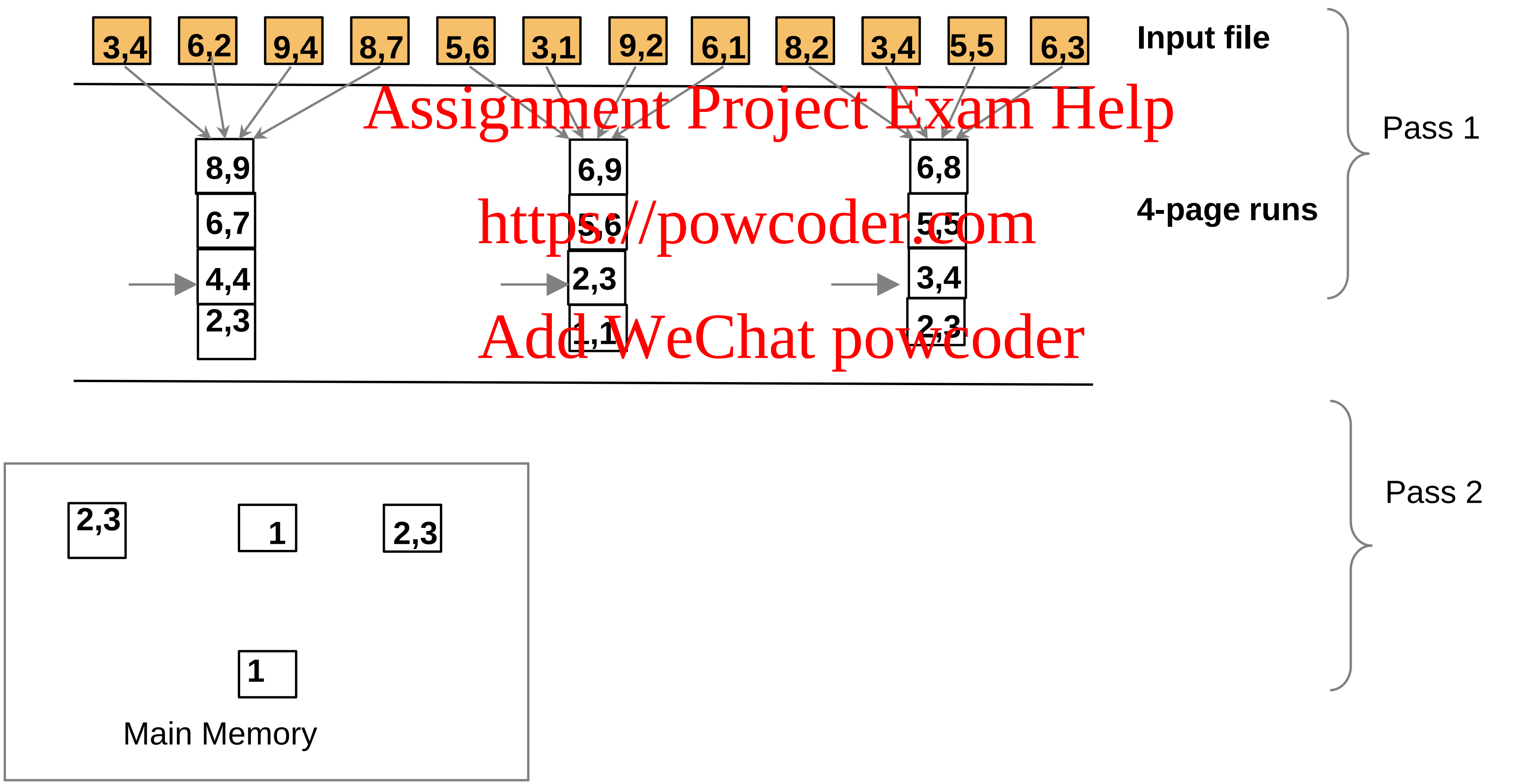
External Merge Sort: Example

buffer pages in memory B = 4, each page 2 records



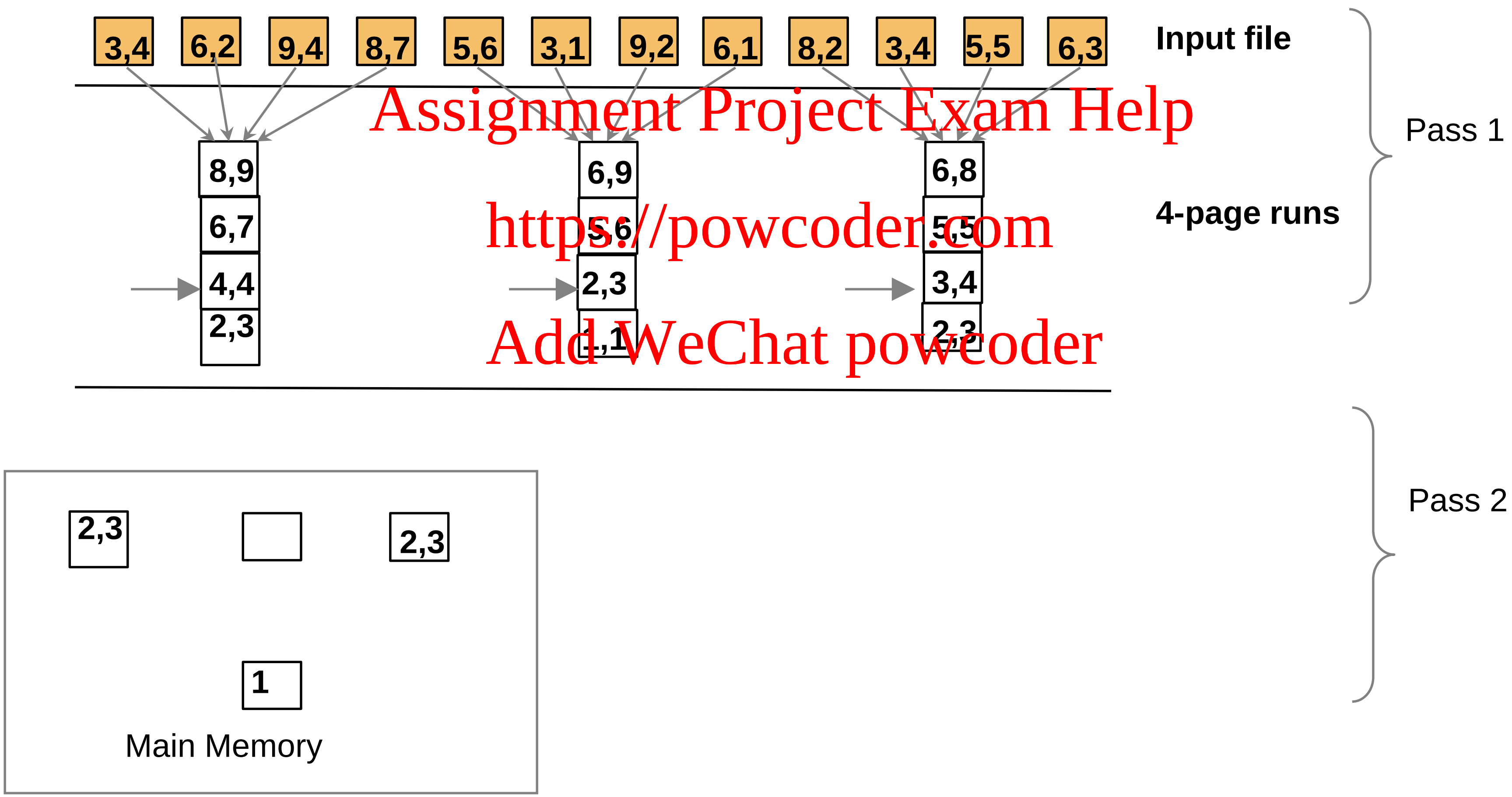
External Merge Sort: Example

buffer pages in memory B = 4, each page 2 records



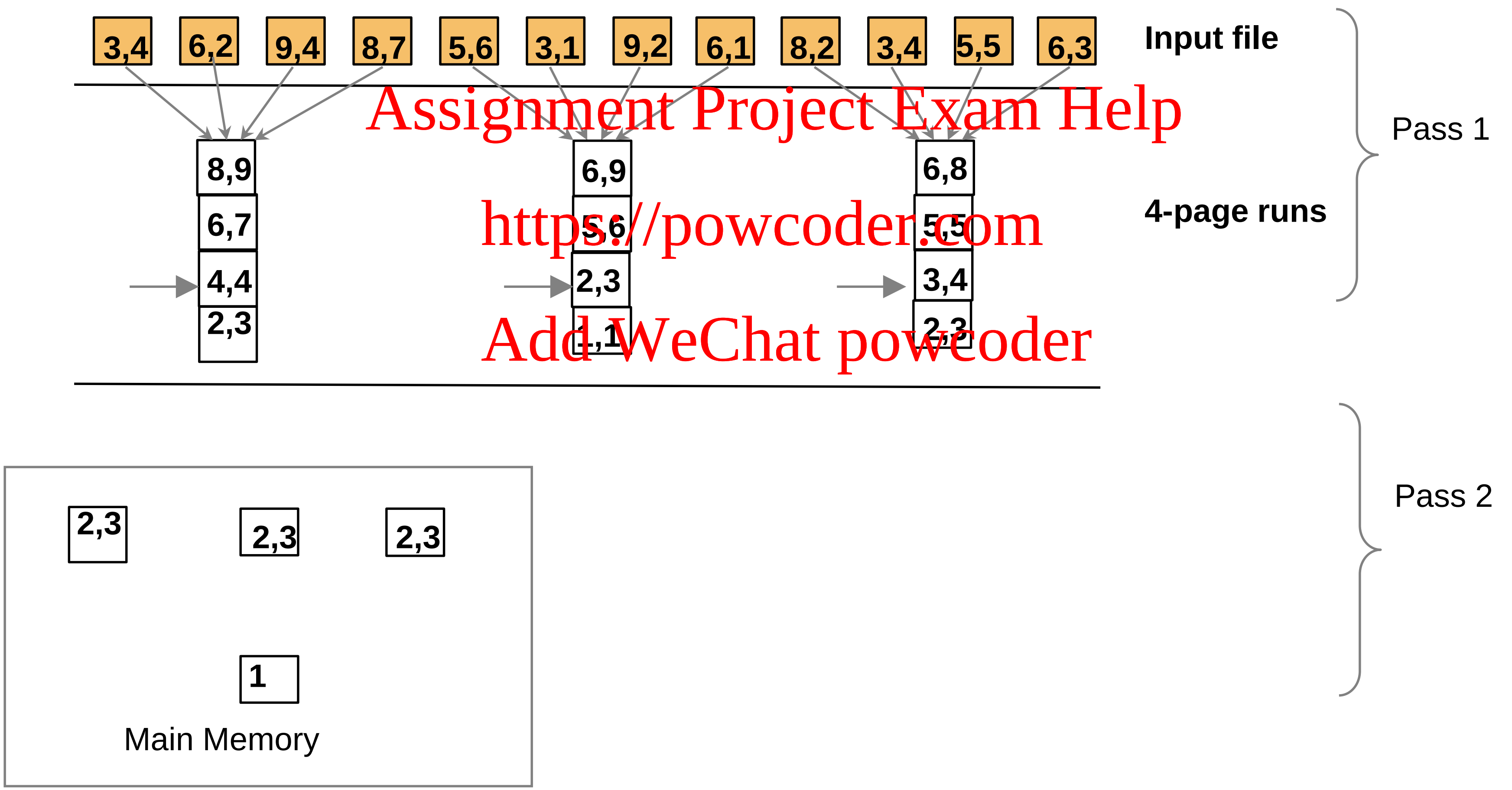
External Merge Sort: Example

buffer pages in memory B = 4, each page 2 records



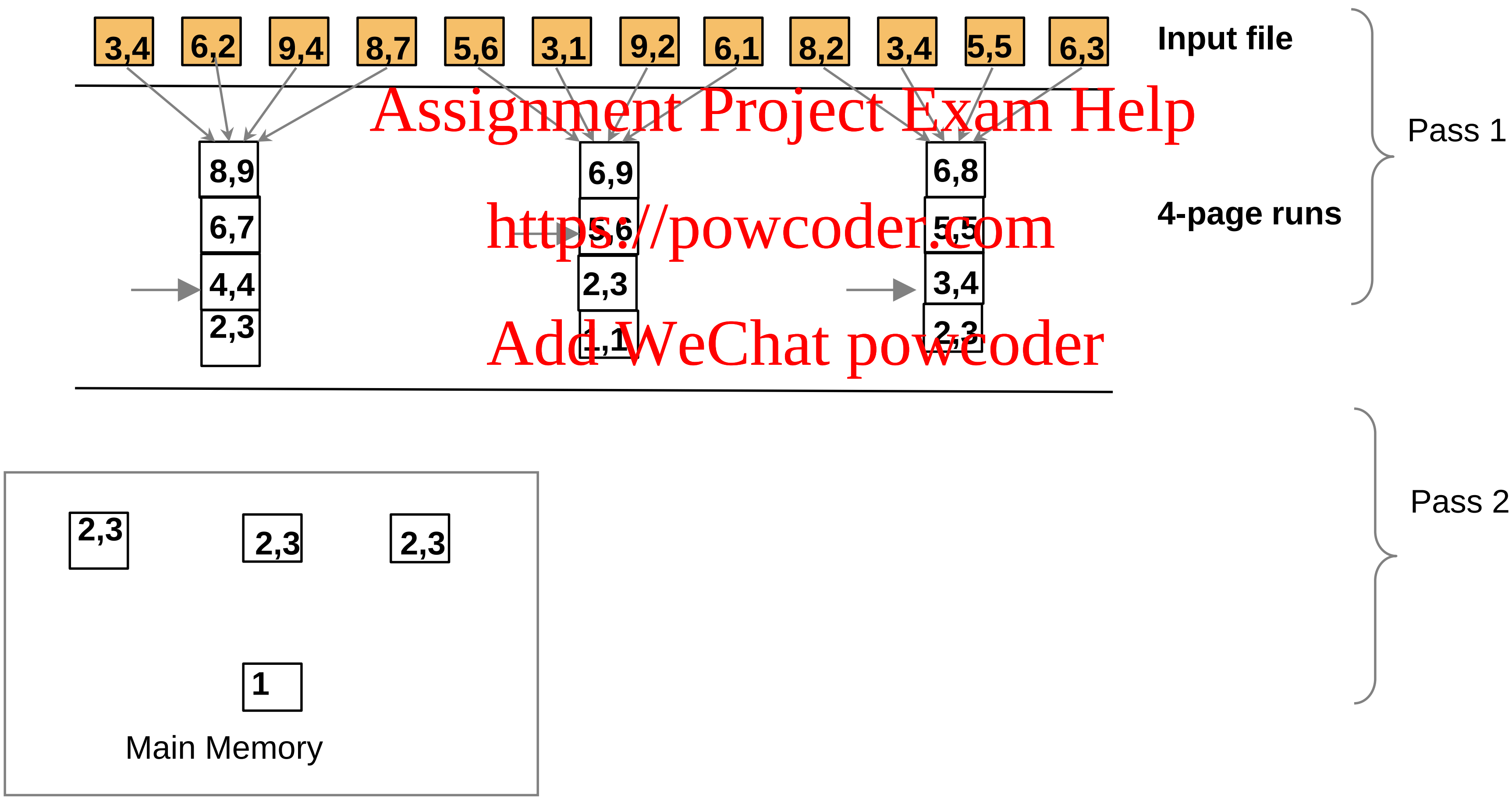
External Merge Sort: Example

buffer pages in memory B = 4, each page 2 records



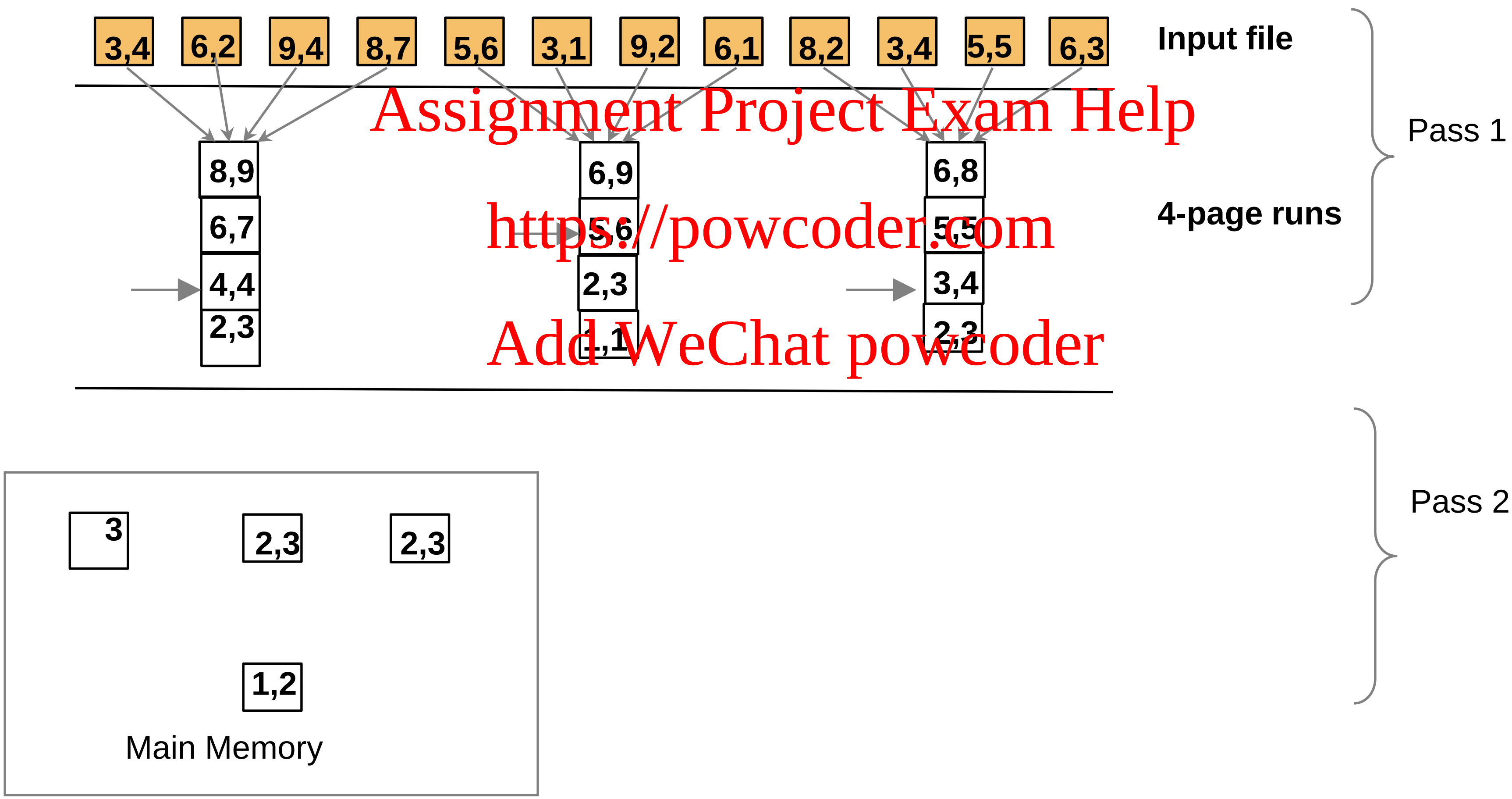
External Merge Sort: Example

buffer pages in memory B = 4, each page 2 records



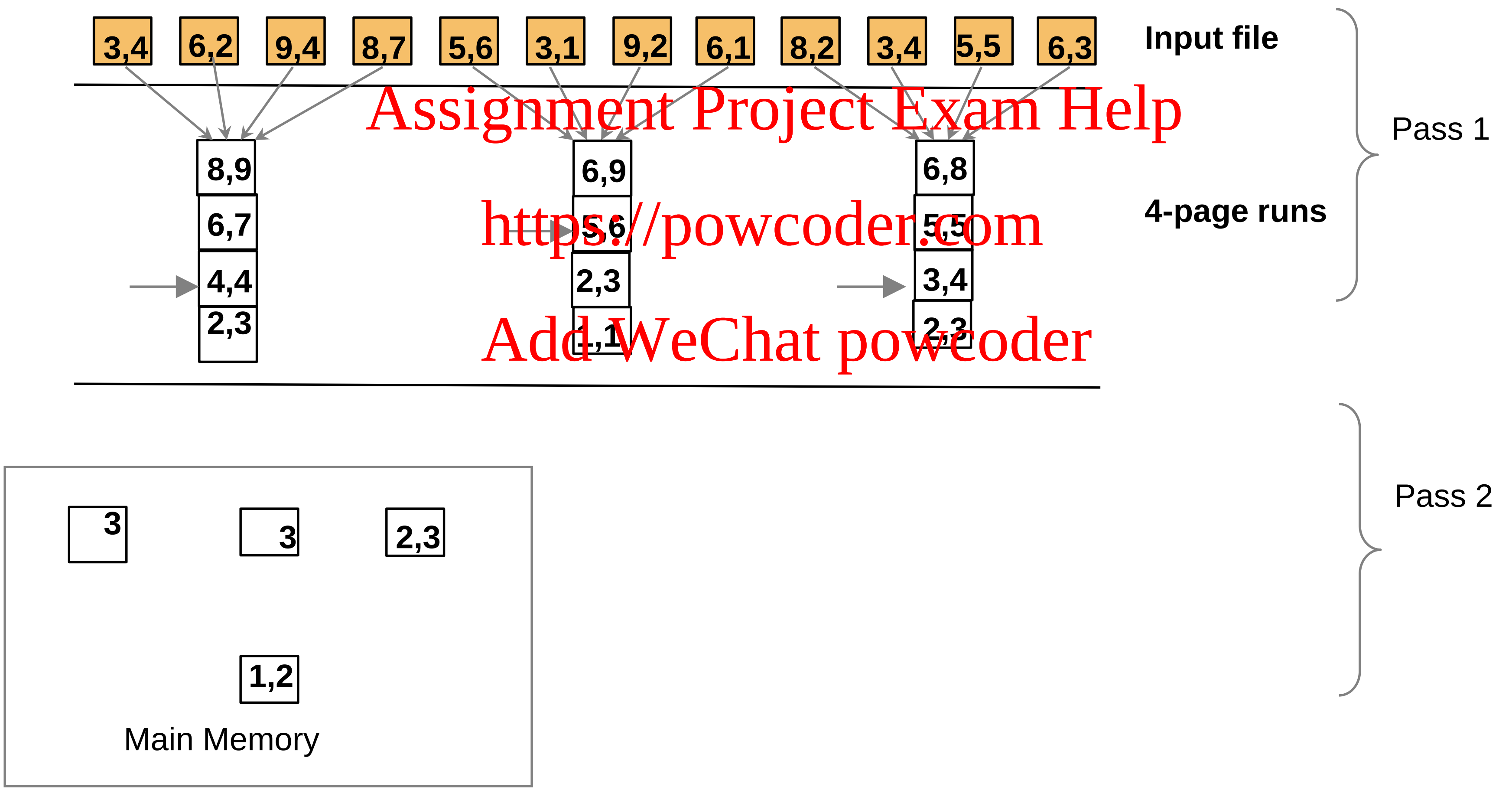
External Merge Sort: Example

buffer pages in memory B = 4, each page 2 records



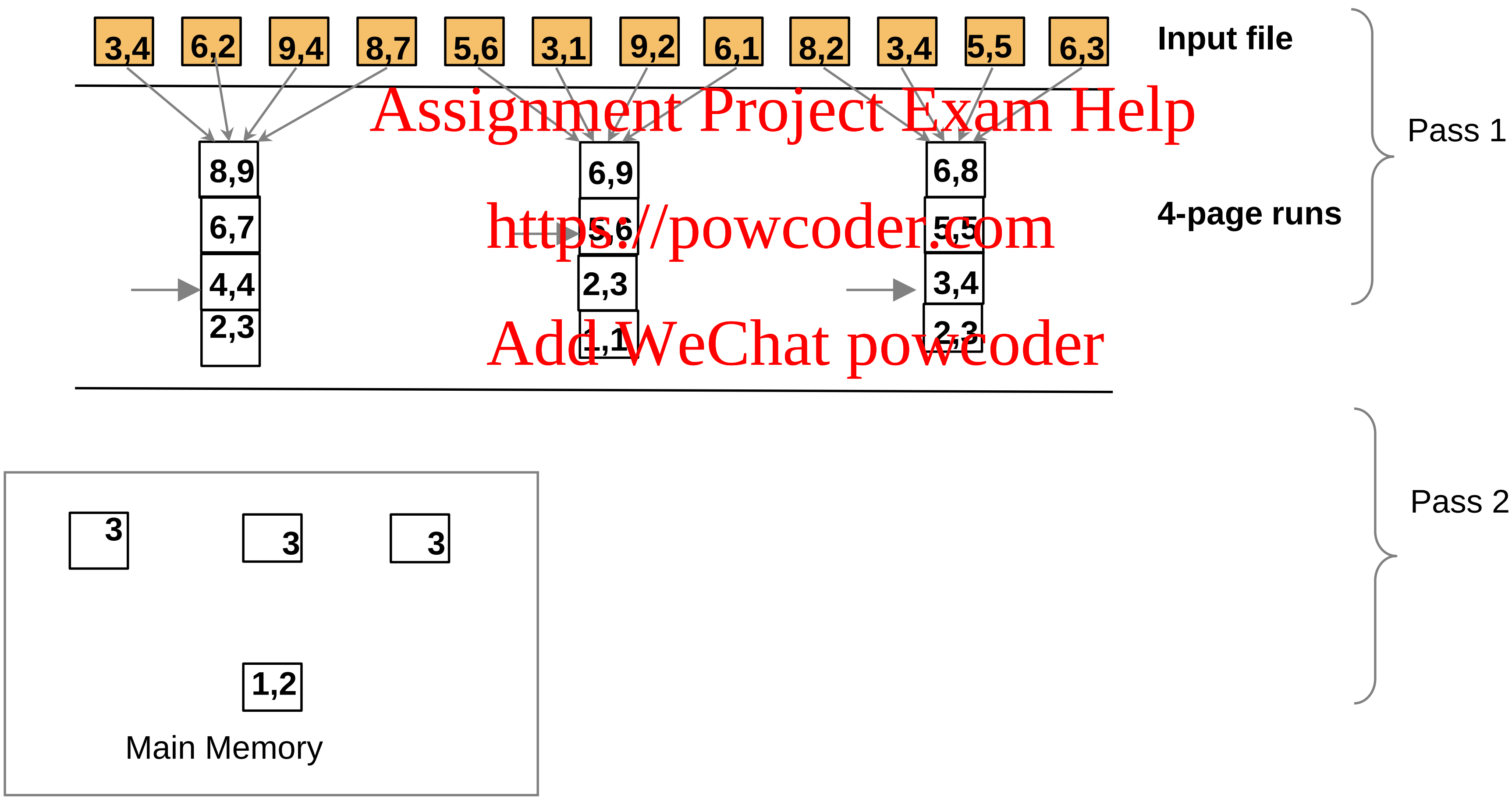
External Merge Sort: Example

buffer pages in memory B = 4, each page 2 records



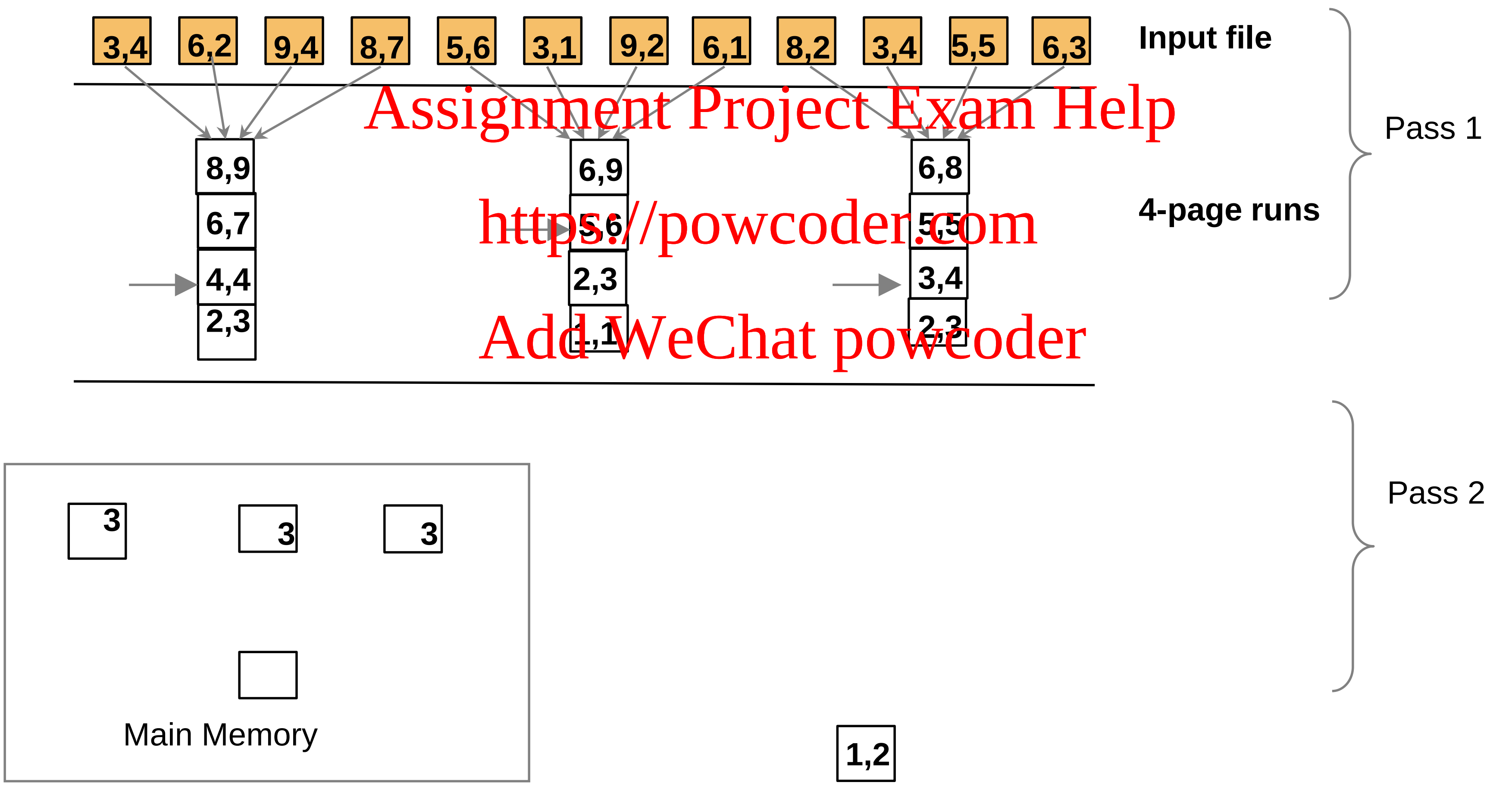
External Merge Sort: Example

buffer pages in memory B = 4, each page 2 records



External Merge Sort: Example

buffer pages in memory B = 4, each page 2 records



The Projection Operation Cost

- Sorting with **external sort**:
  - –1. Scan R, extract only the needed attributes
  - –2. Sort the result set using EXTERNAL SORT
  - –3. Remove adjacent duplicates

Cost = ReadTable + Read the entire table and keep only projected attributes
WriteProjectedPages + Write pages with projected attributes to disk
SortingCost + Sort pages with projected attributes with external sort
ReadProjectedPages Read sorted projected pages to discard adjacent duplicates

WriteProjectedPages = NPages(R)* PF

PF: Projection Factor says how much are we projecting, ratio with respect to all attributes (e.g. keeping ¼ of attributes, or 10% of all attributes)

Every time we read and write

SortingCost = 2*NumPasses*ReadProjectedPages

Our example

- **Example**: Let's say that we project ¼ of all attributes, and let's say that we have 20 pages in memory
- PF = 1/4 = 0.25, NPages(R) = 1000
- With 20 memory pages we can sort in 2 passes

Cost = ReadTable +
WriteProjectedPages +
SortingCost +
ReadProjectedPages

= 1000 + 0.25 * 1000 + 2*2*250 + 250 = 2500 (I/O)

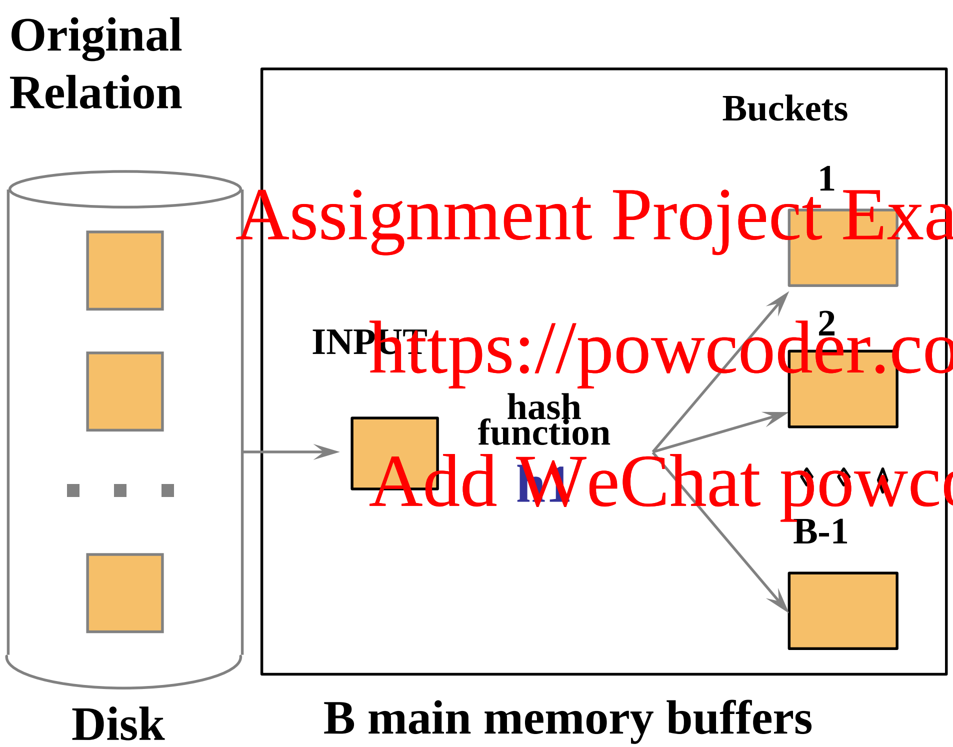
THE UNIVERSITY OF MELBOURNE

Projection based on Hashing

MELBOURNE

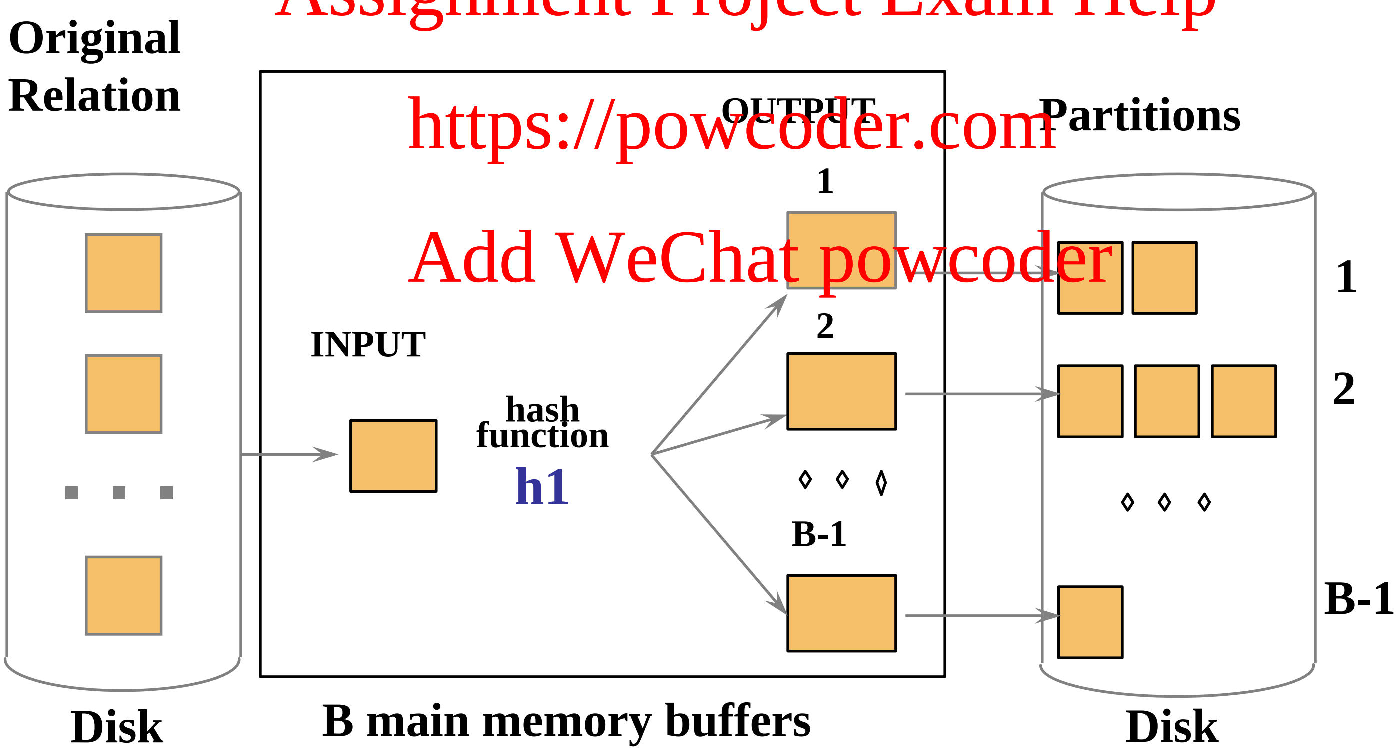
- Hashing-based projection
  - 1. Scan R, extract only the **needed** attributes
  - 2. Hash data into buckets
    - Apply hash function *h1* to choose one of B output buffers
  - 3. Remove **adjacent** duplicates from a bucket
    - 2 tuples from different partitions guaranteed to be distinct

Projection Based on *Hashing*



Projection based on External Hashing

1. **Partition** data into B partitions with h1 hash function
2. Load each partition, hash it with another hash function (h2) and **eliminate duplicates**



Projection based on External Hashing

1. Partitioning phase:
   - –Read R using one input buffer
   - –For each tuple:
     - •Discard unwanted fields
     - •Apply hash function *h1* to choose one of B-1 output buffers
   - –Result is B-1 partitions (of tuples with no unwanted fields)
     - •2 tuples from different partitions guaranteed to be distinct

2. Duplicate elimination phase:
   - –For each partition
     - •Read it and build an in-memory hash table
       - –using hash function *h2* (<> *h1*) on all fields
     - •while discarding duplicates
   - –If partition does not fit in memory
     - •Apply hash-based projection algorithm recursively to this partition (we will not do this…)

Cost of External Hashing

MELBOURNE

Cost = ReadTable + WriteProjectedPages + ReadProjectedPages

Read the entire table and project attributes

Write projected pages into corresponding partitions

Read partitions one by one, create another hash table and discard duplicates within a bucket

Our example:

Cost = ReadTable +

WriteProjectedPages +
ReadProjectedPages

= 1000 + 0.25 * 1000 + 250 = 1500 (I/O)

What's examinable

- Understand the logic behind relational operators
- Learn alternatives for selections and projections (for now)
  - –Be able to calculate the cost of alternatives
- Important for Assignment 3 as well

Next Lecture

- Query Processing Part II
  - Join alternatives