CSE 444: Database Internals

Section 3:

Operator Algorithms

Notations

- `B(R)` = # of blocks (i.e. pages) for relation `R`
- `T(R)` = # of tuples in relation `R`
- `V(R,a)` = # of distinct values of attribute a
- Memory `M`

Algorithms for Group By and Aggregate Operators

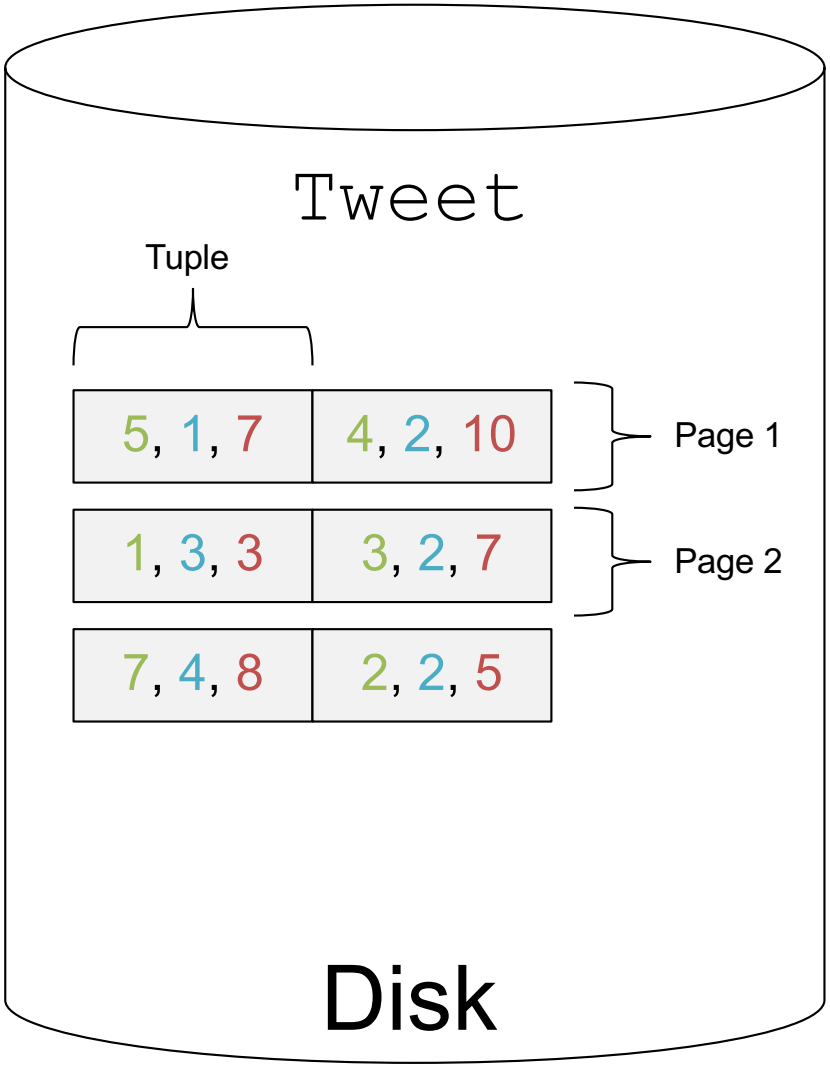
- Tweet Example:

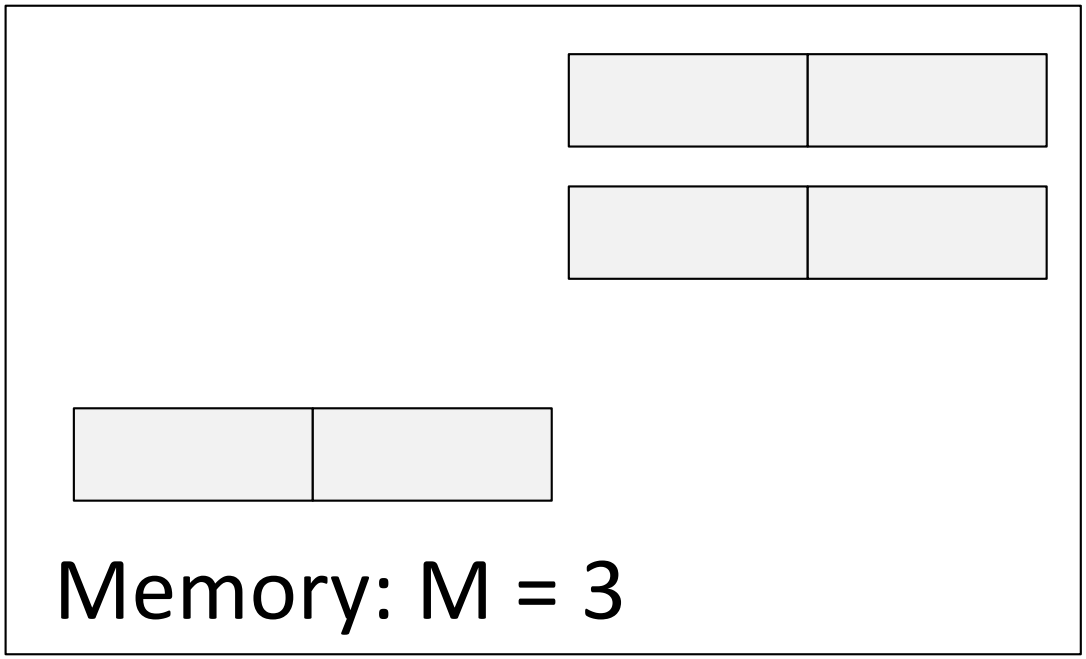
```
Tweet(tweet_id, user_id, tweet_length)

  SELECT user_id, MIN(tweet_len)
  FROM Tweet
  GROUP BY user_id
```

```
Tweet(tweet_id, user_id, tweet_length)
```

```
SELECT user_id, MIN(tweet_length)
FROM Tweet
GROUP BY user_id
```

One pass, hash-based grouping

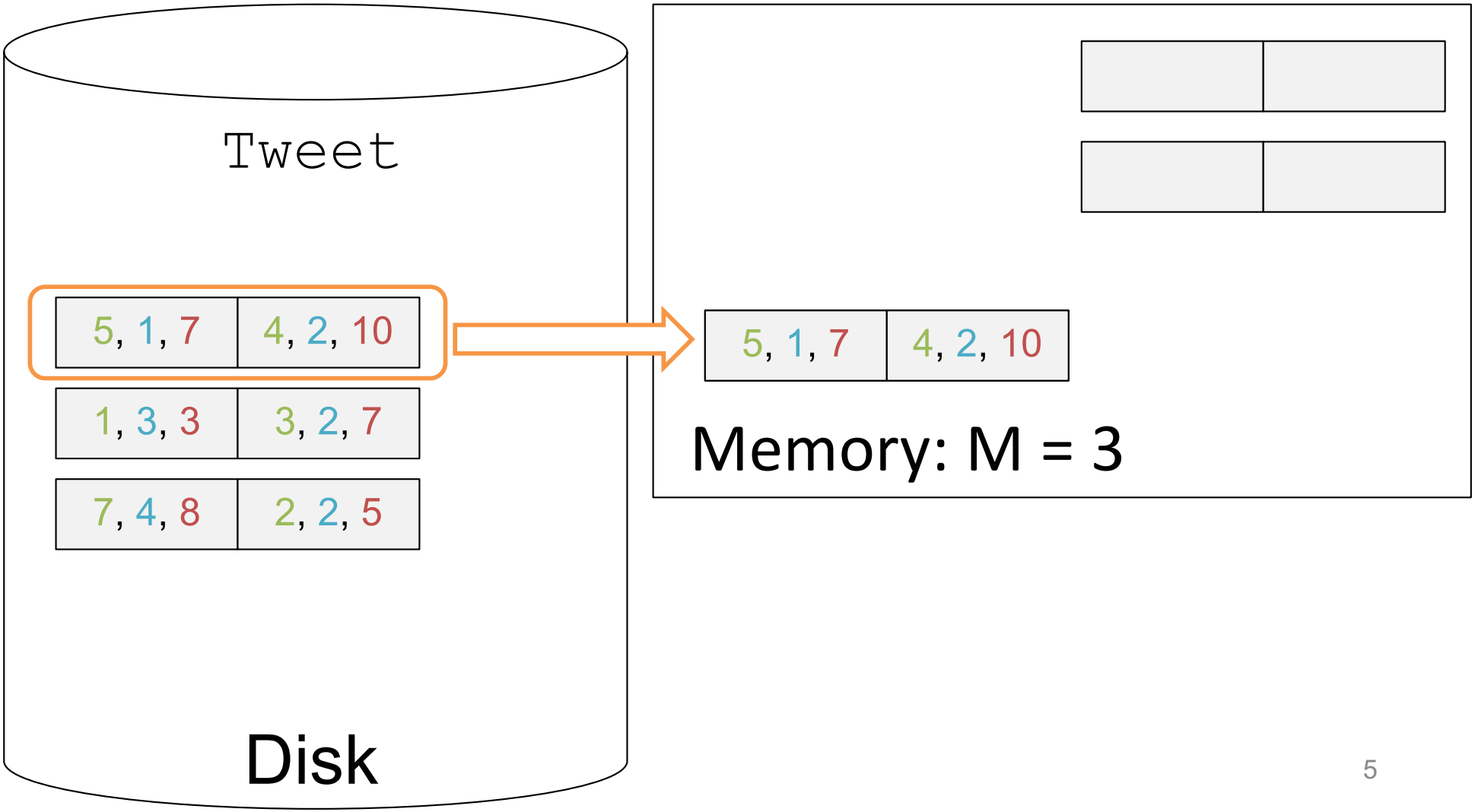




```
Tweet(tweet_id, user_id, tweet_length)
```

```
SELECT user_id, MIN(tweet_length)
FROM Tweet
GROUP BY user_id
```

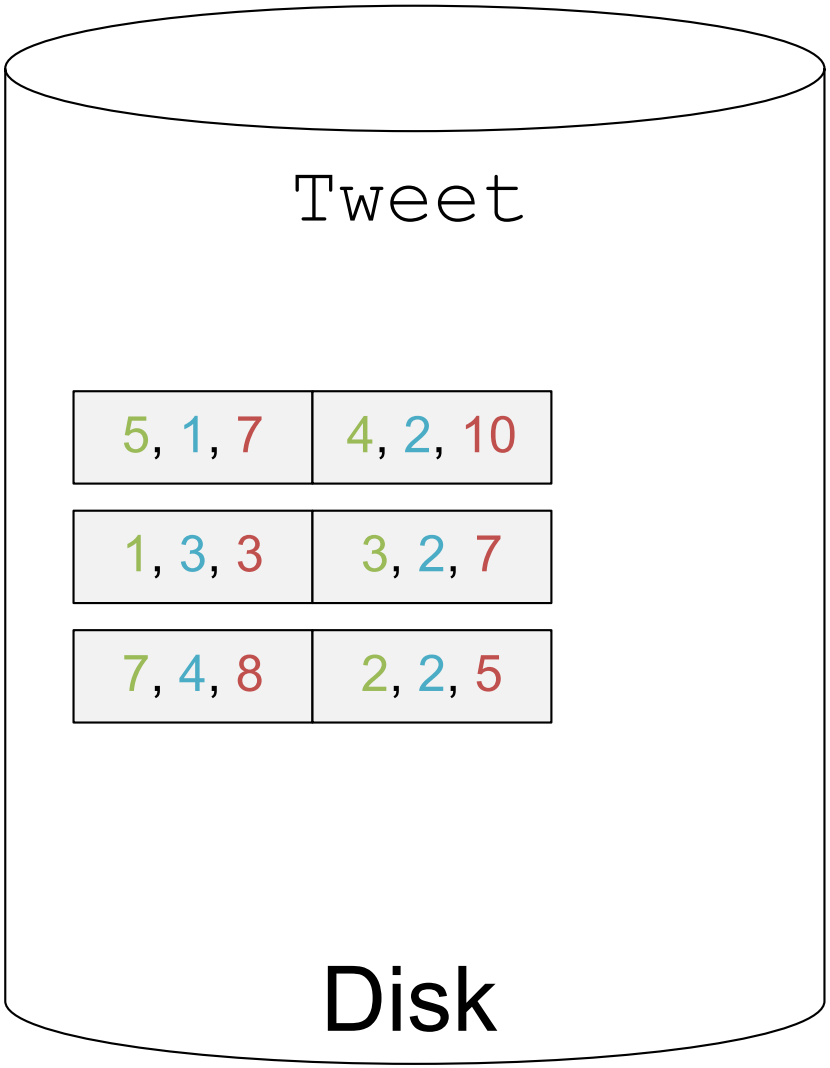
One pass, hash-based grouping

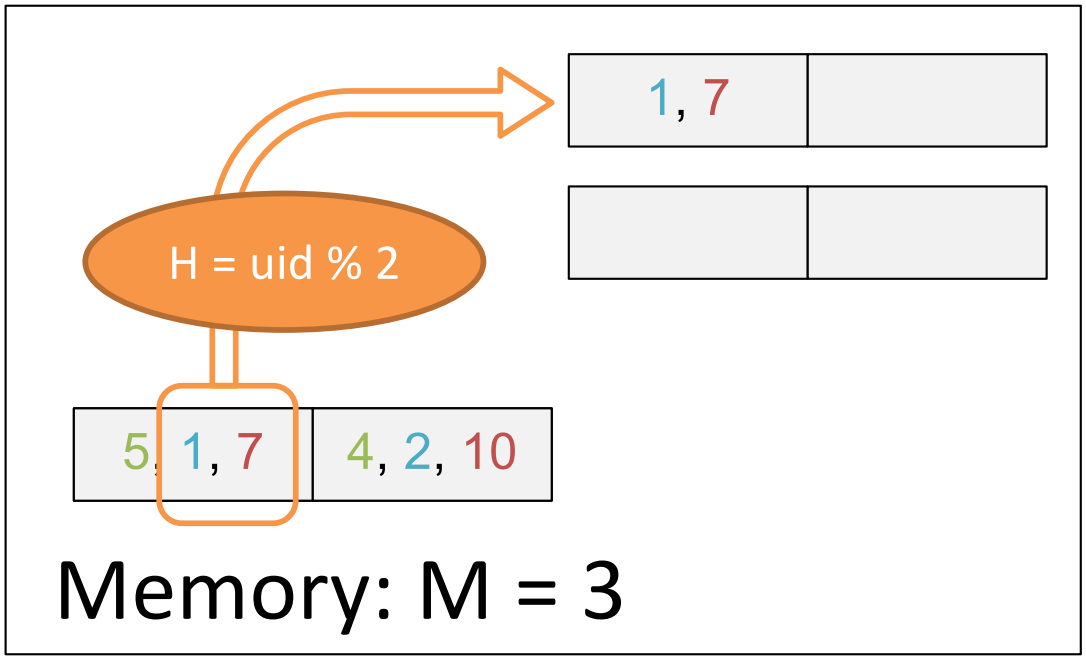


```
Tweet(tweet_id, user_id, tweet_length)
```

```
SELECT user_id, MIN(tweet_length)
FROM Tweet
GROUP BY user_id
```

One pass, hash-based grouping

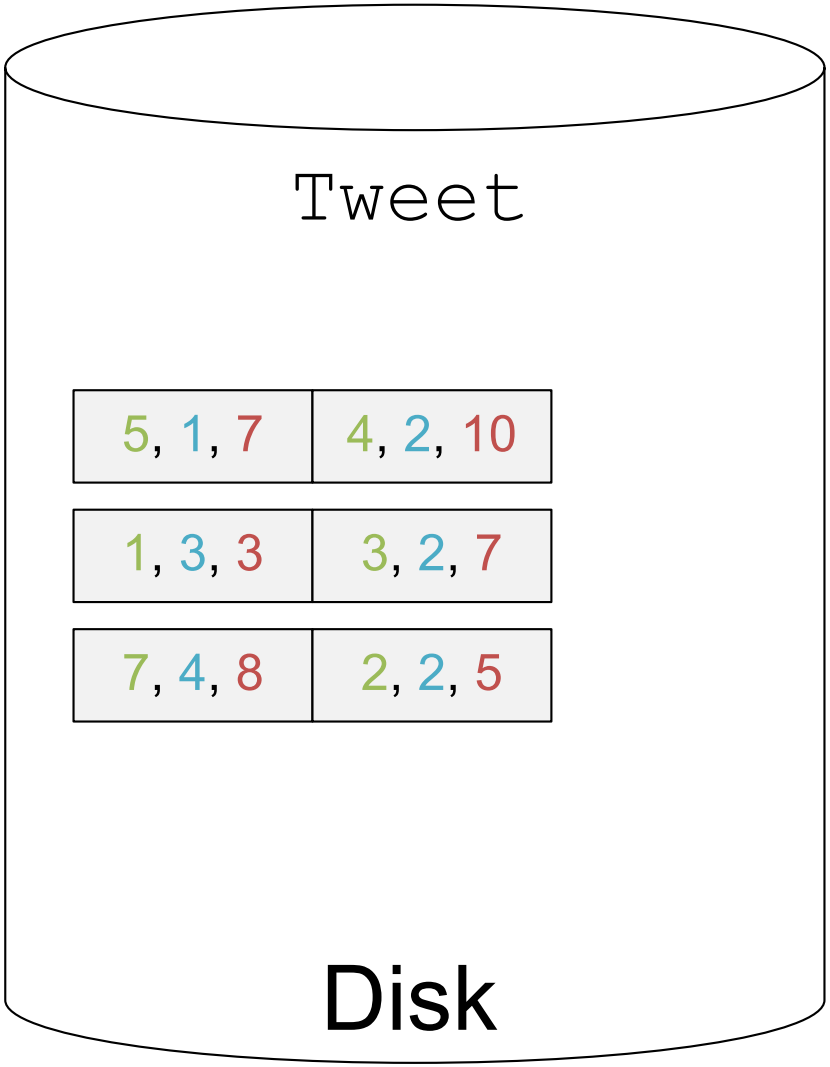


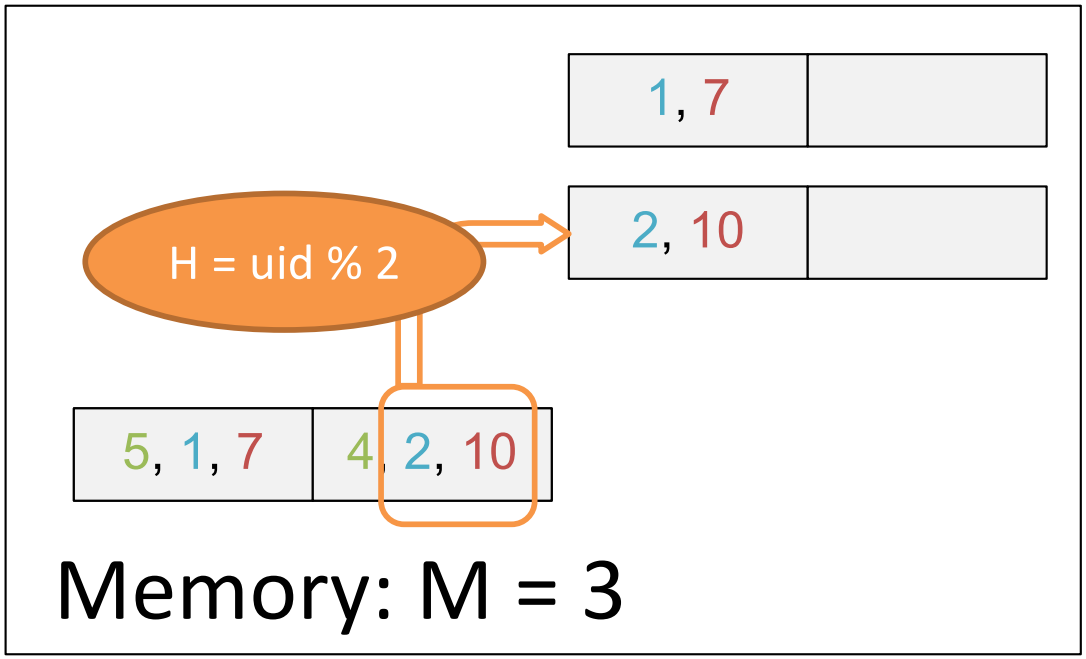


```
Tweet(tweet_id, user_id, tweet_length)
```

```
SELECT user_id, MIN(tweet_length)
FROM Tweet
GROUP BY user_id
```

One pass, hash-based grouping

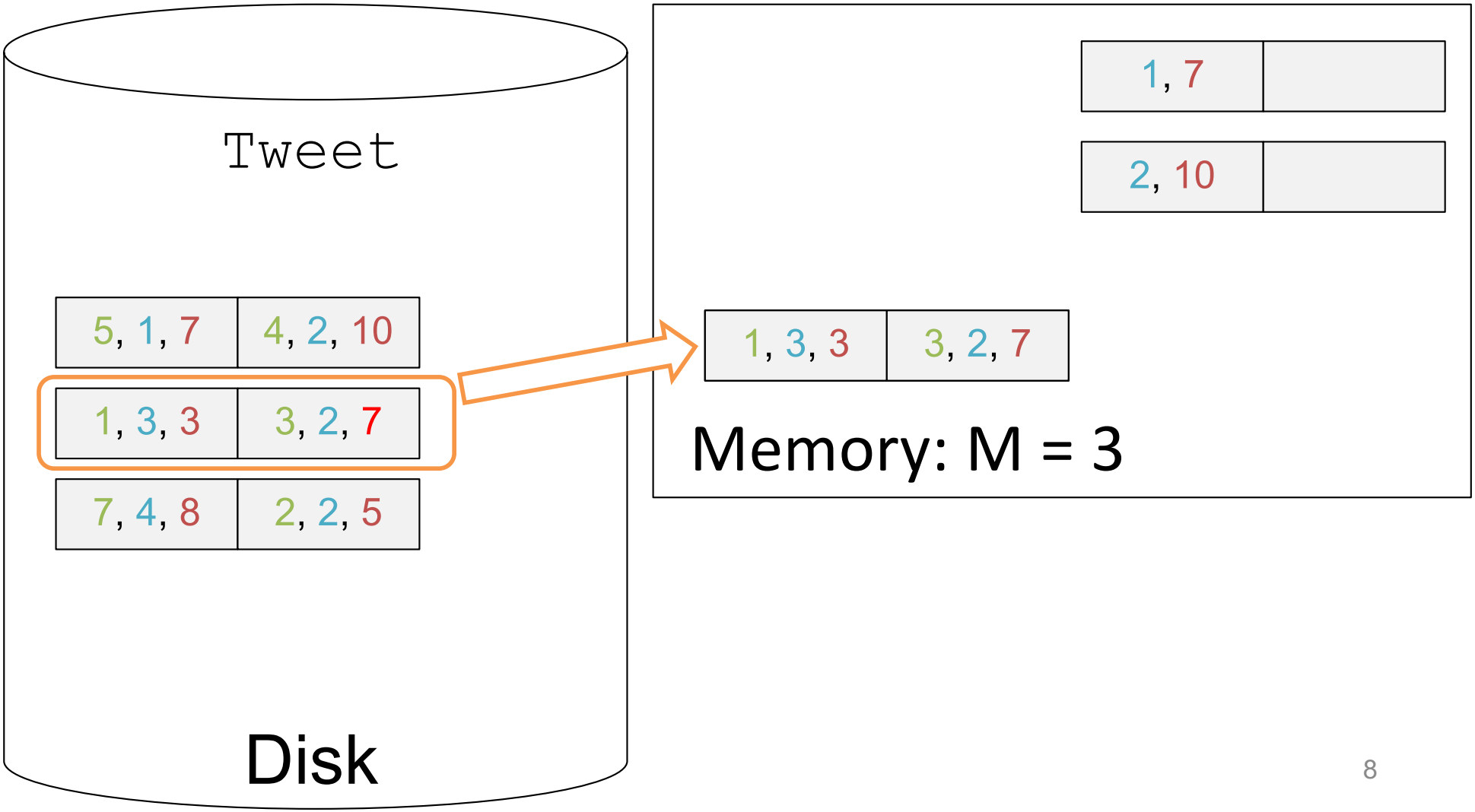




```
Tweet(tweet_id, user_id, tweet_length)
```

```
SELECT user_id, MIN(tweet_length)
FROM Tweet
GROUP BY user_id
```

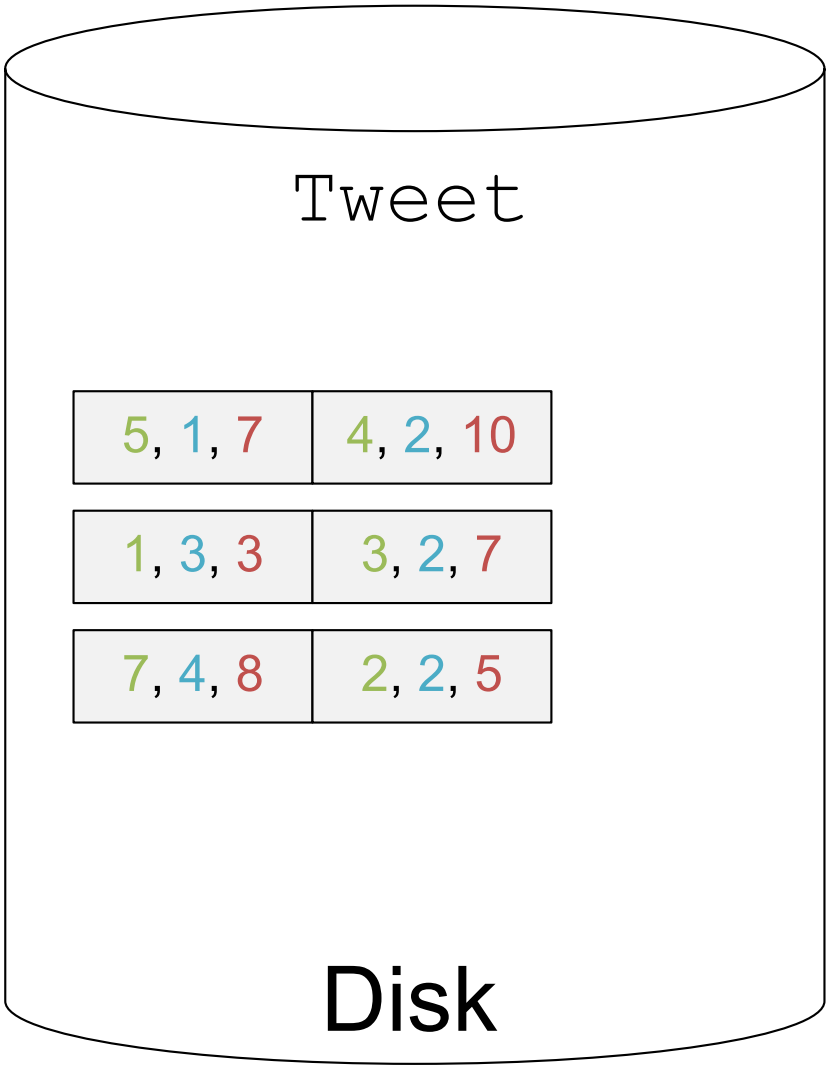
One pass, hash-based grouping

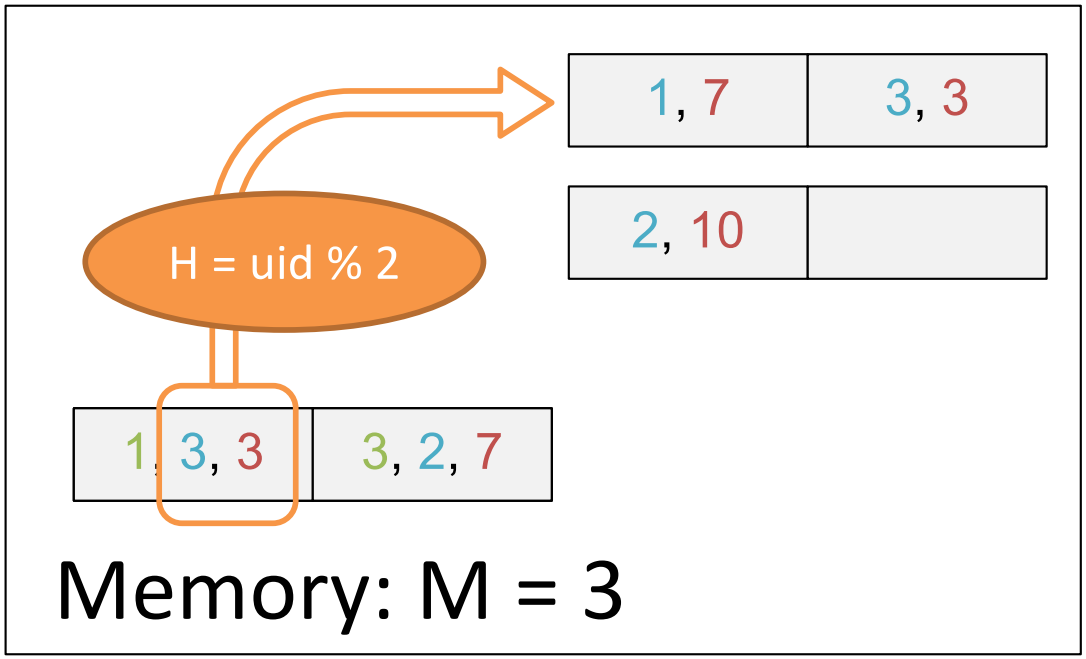


```
Tweet(tweet_id, user_id, tweet_length)
```

```
SELECT user_id, MIN(tweet_length)
FROM Tweet
GROUP BY user_id
```

One pass, hash-based grouping

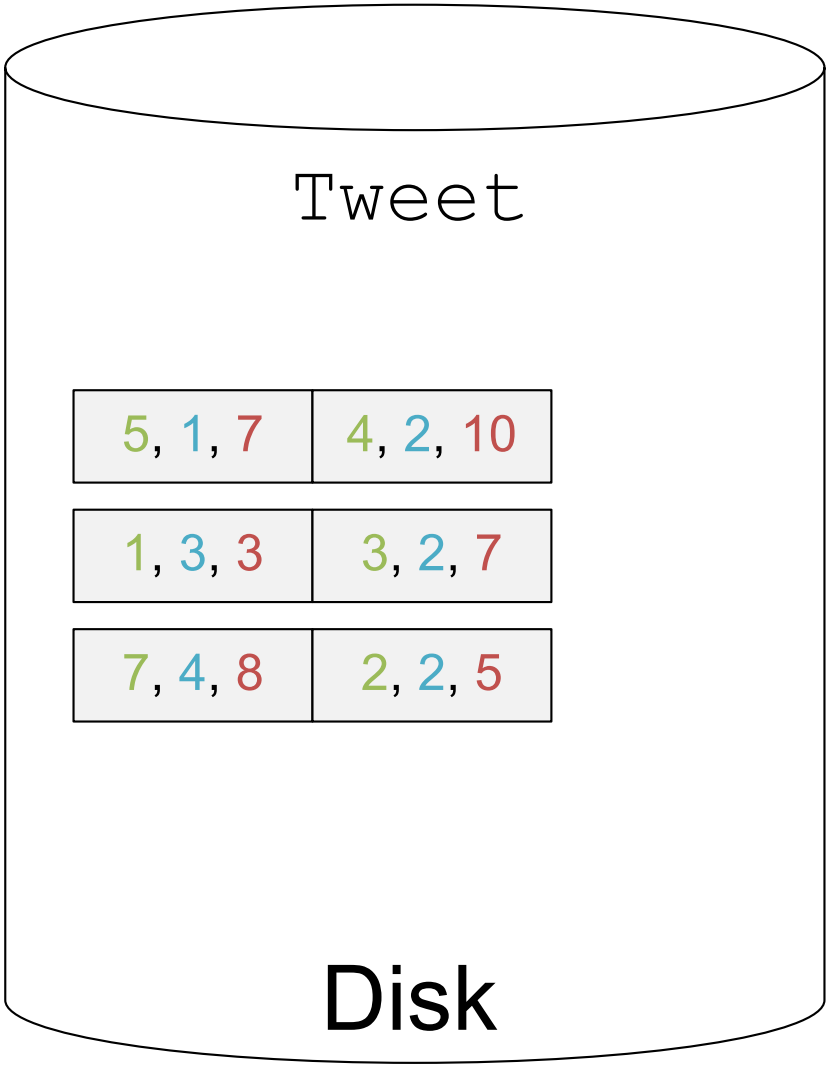


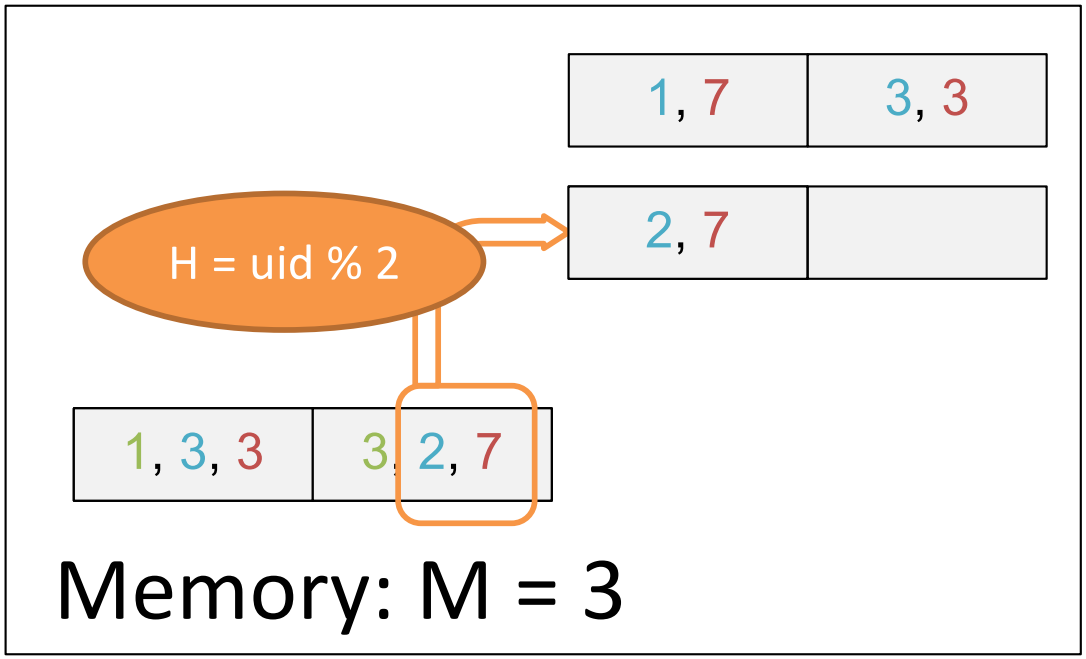


```
Tweet(tweet_id, user_id, tweet_length)
```

```
SELECT user_id, MIN(tweet_length)
FROM Tweet
GROUP BY user_id
```

One pass, hash-based grouping

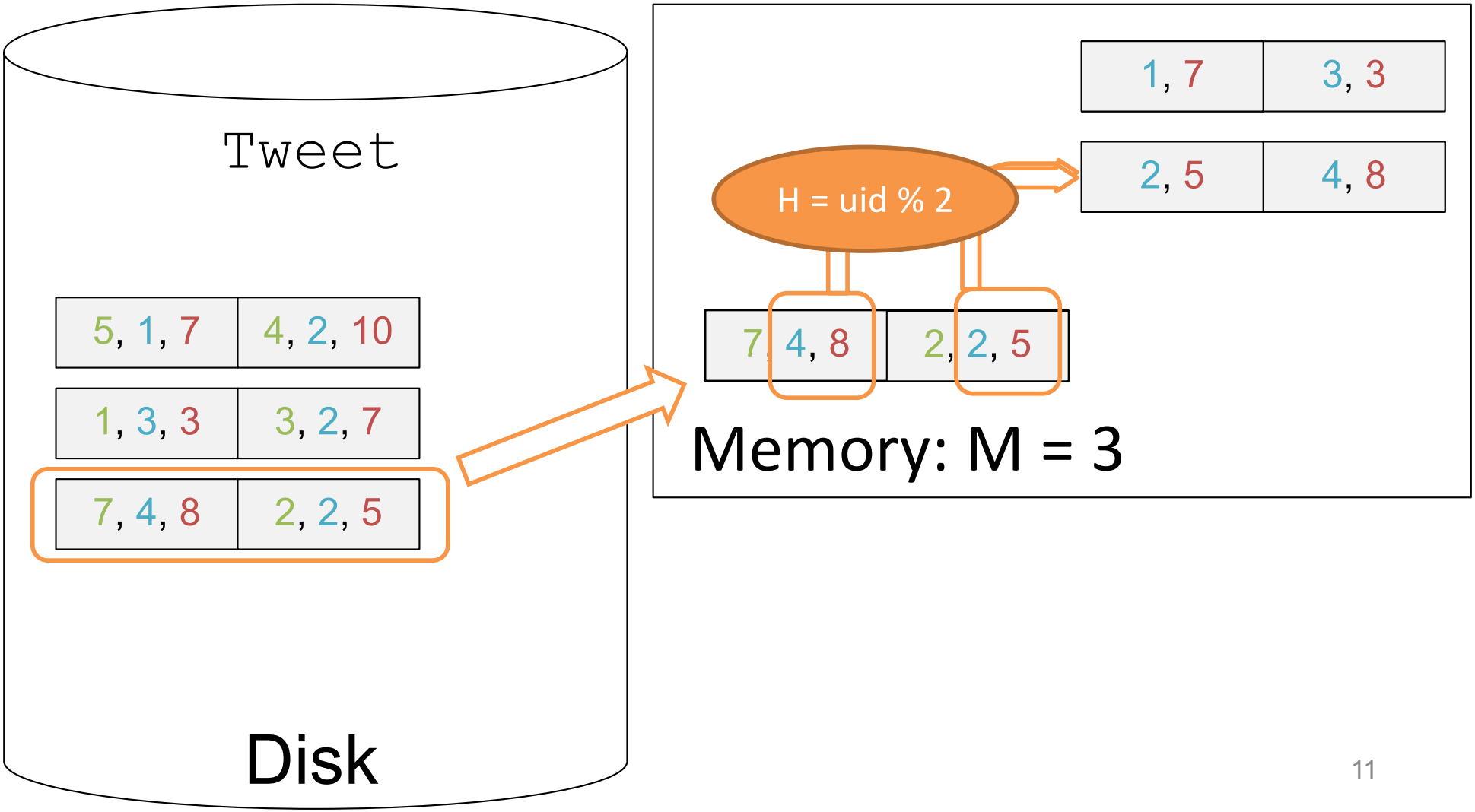




```
Tweet(tweet_id, user_id, tweet_length)
```

```
SELECT user_id, MIN(tweet_length)
FROM Tweet
GROUP BY user_id
```

One pass, hash-based grouping



Discussion

Which operator method does the grouping?

open(), next(), or close()?

What to do for AVG(tlen)?

Cost:

- Clustered?
- Unclustered?

Discussion

Which method does the grouping:

open(), next(), or close()?

- Cannot return anything until the entire data is read. Open() needs to do grouping

What to do for AVG(tlen)?

Keep both SUM(tlen) and COUNT(*) for each group in memory

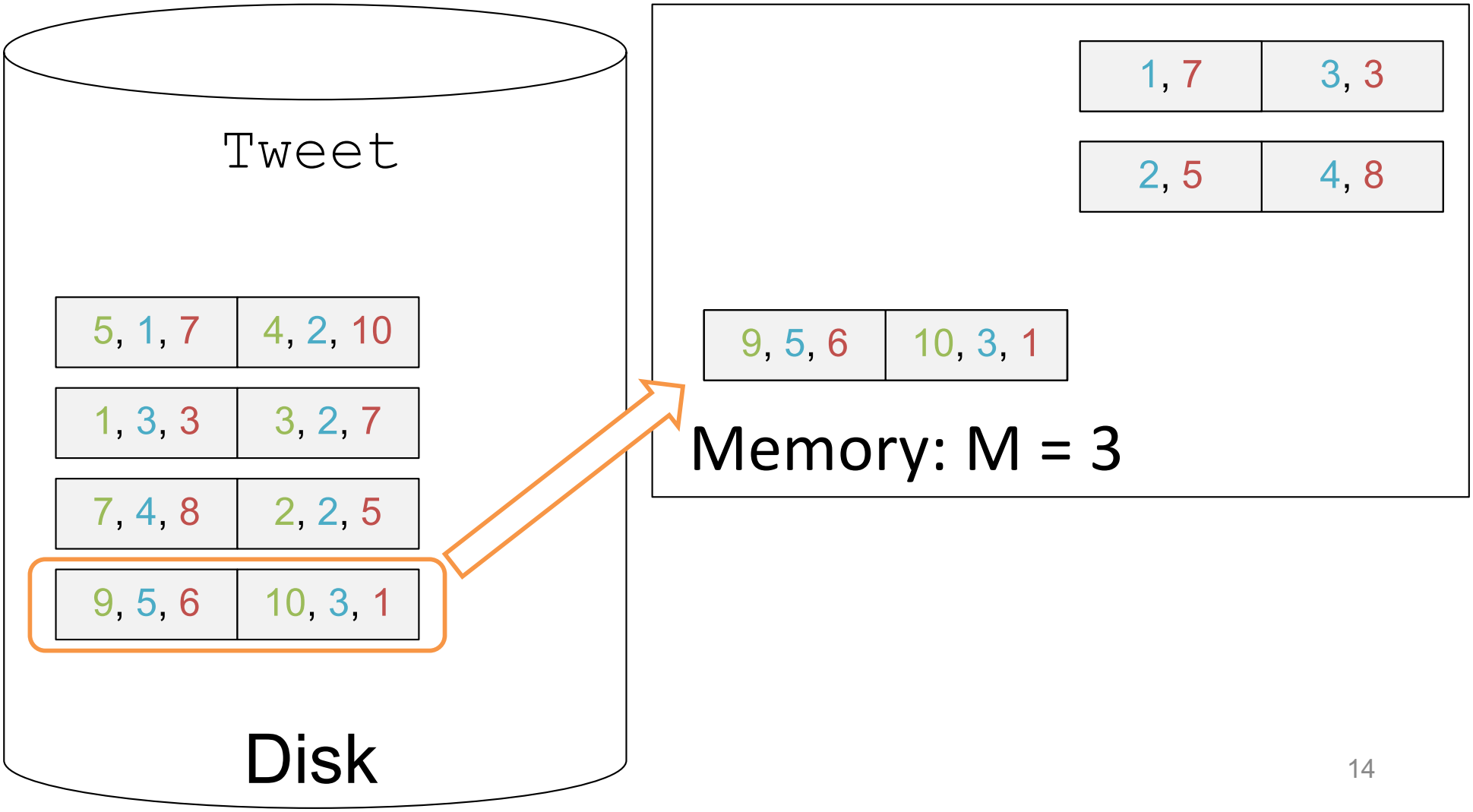
Cost:

- Clustered?
  - B(R): assuming M – 1 pages can hold all groups – tuples for groups can be shorter or larger than original tuples
- Unclustered?
  - Also B(R)

```
Tweet(tweet_id, user_id, tweet_length)
```

```
SELECT user_id, MIN(tweet_length)
FROM Tweet
GROUP BY user_id
```

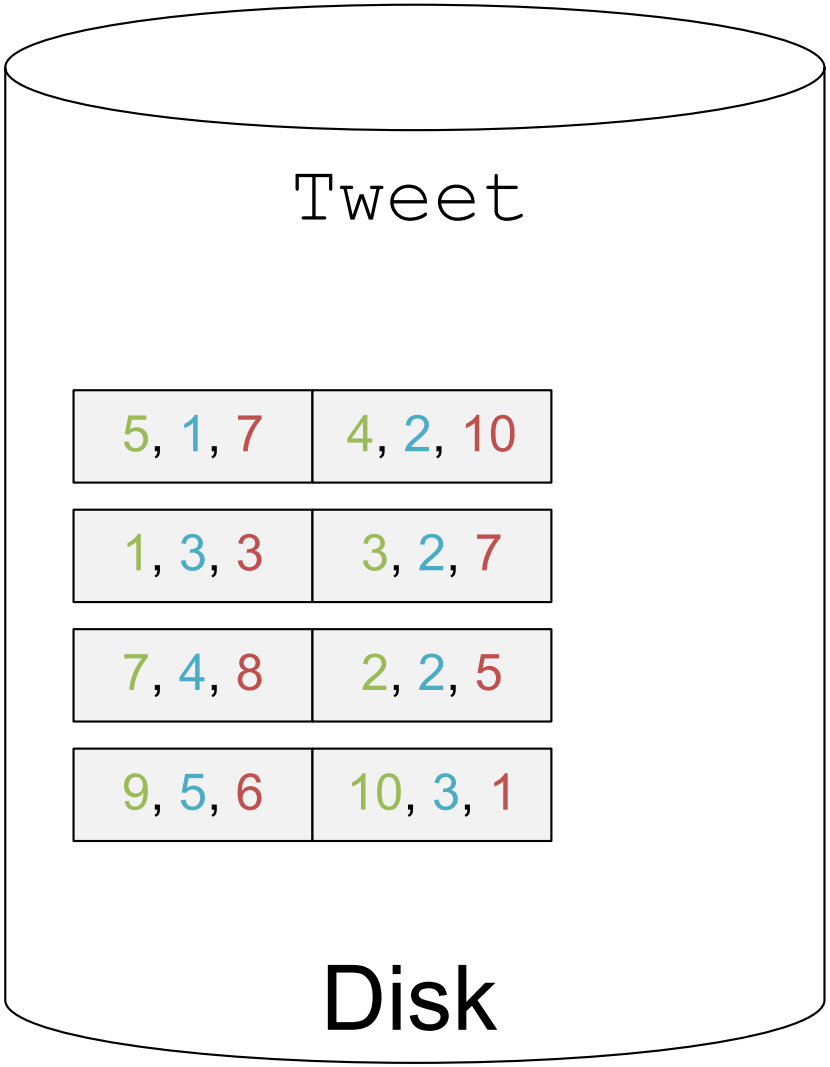
One pass, hash-based grouping, still

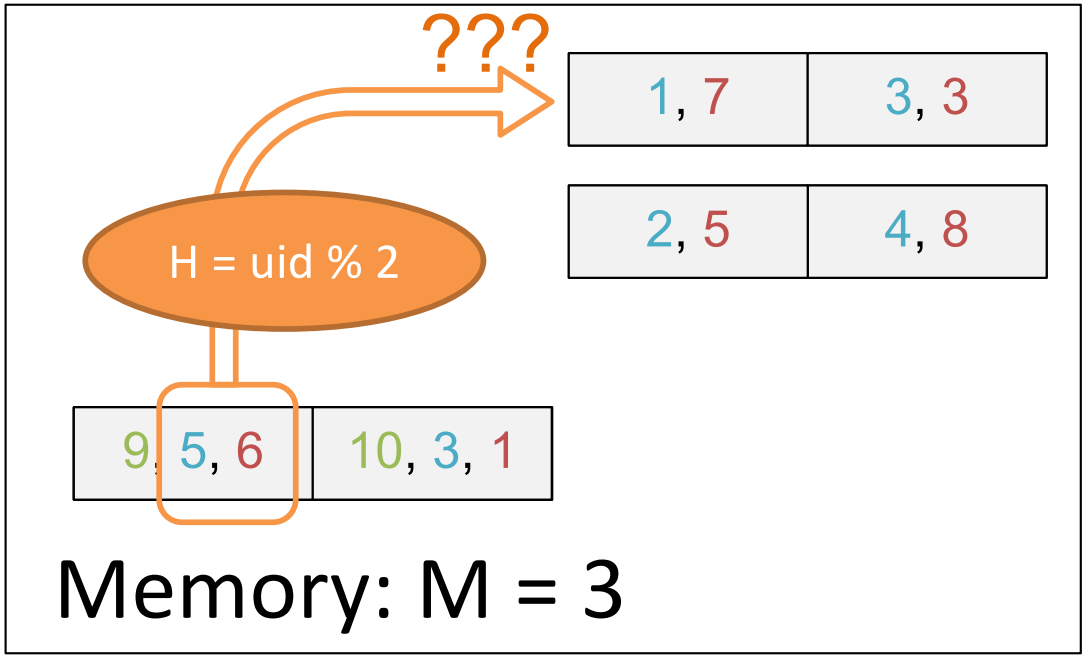


```
Tweet(tweet_id, user_id, tweet_length)
```

```
SELECT user_id, MIN(tweet_length)
FROM Tweet
GROUP BY user_id
```

One pass, hash-based grouping, still

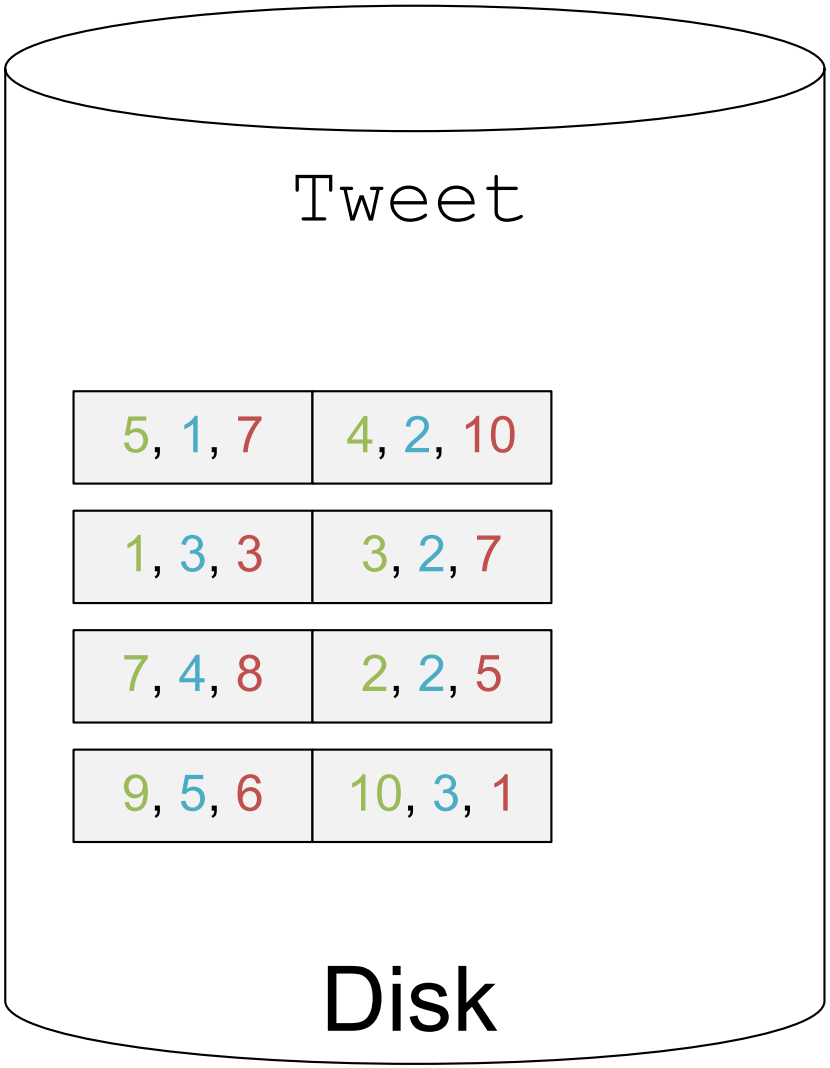


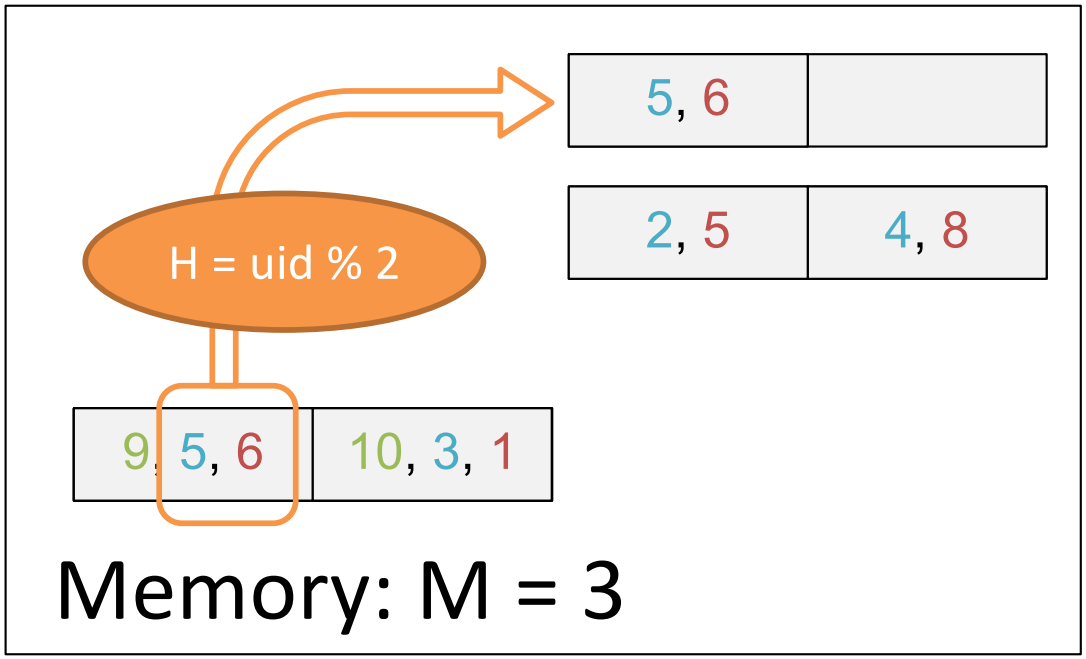


```
Tweet(tweet_id, user_id, tweet_length)
```

```
SELECT user_id, MIN(tweet_length)
FROM Tweet
GROUP BY user_id
```

One pass, hash-based grouping, still

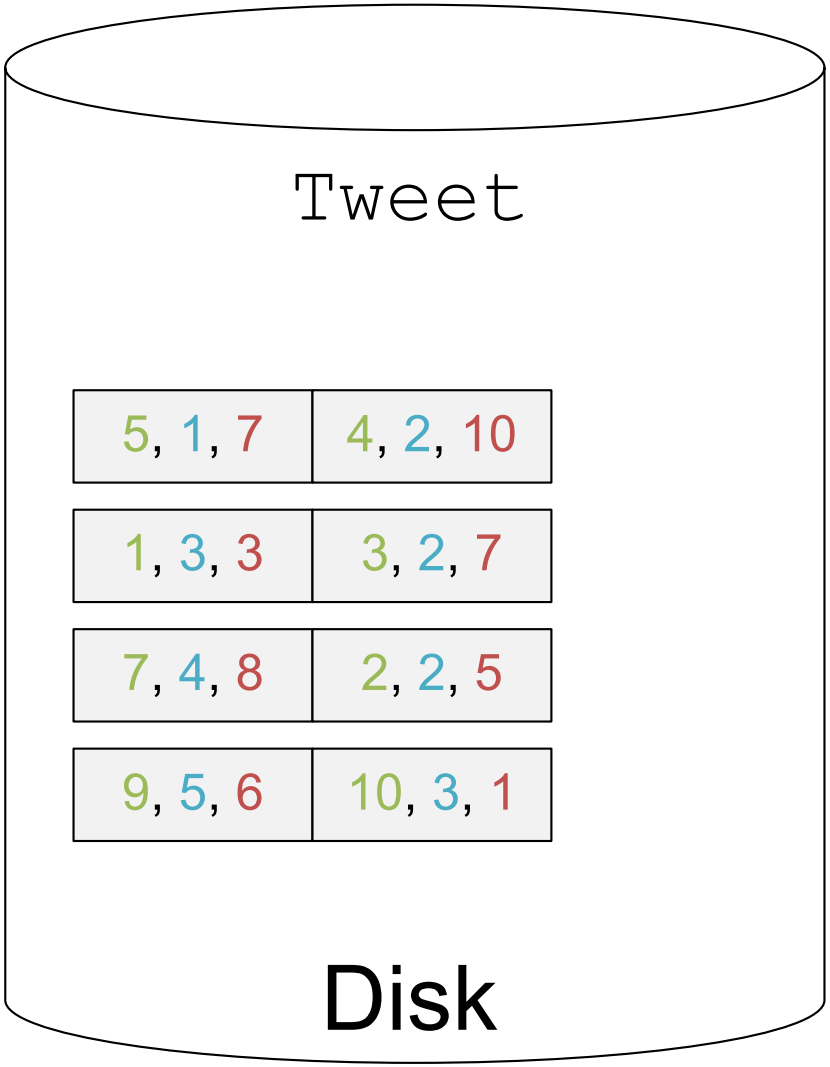


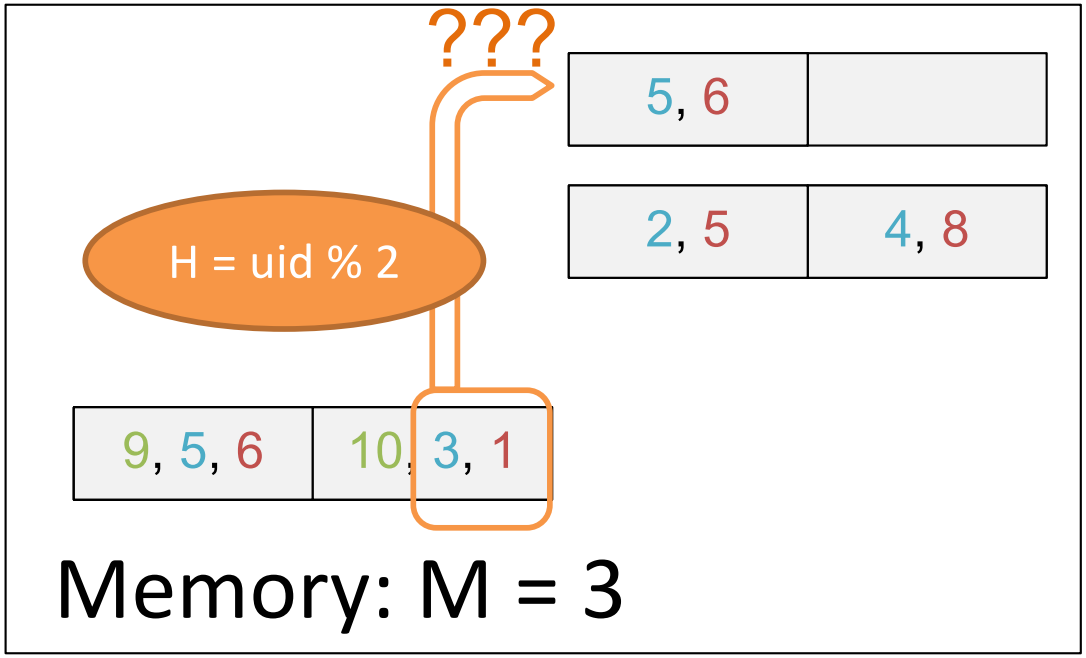


```
Tweet(tweet_id, user_id, tweet_length)
```

```
SELECT user_id, MIN(tweet_length)
FROM Tweet
GROUP BY user_id
```

One pass, hash-based grouping, still





Discussion

One Pass Hash Group:

Cost: B(R): assuming M – 1 pages can hold all groups

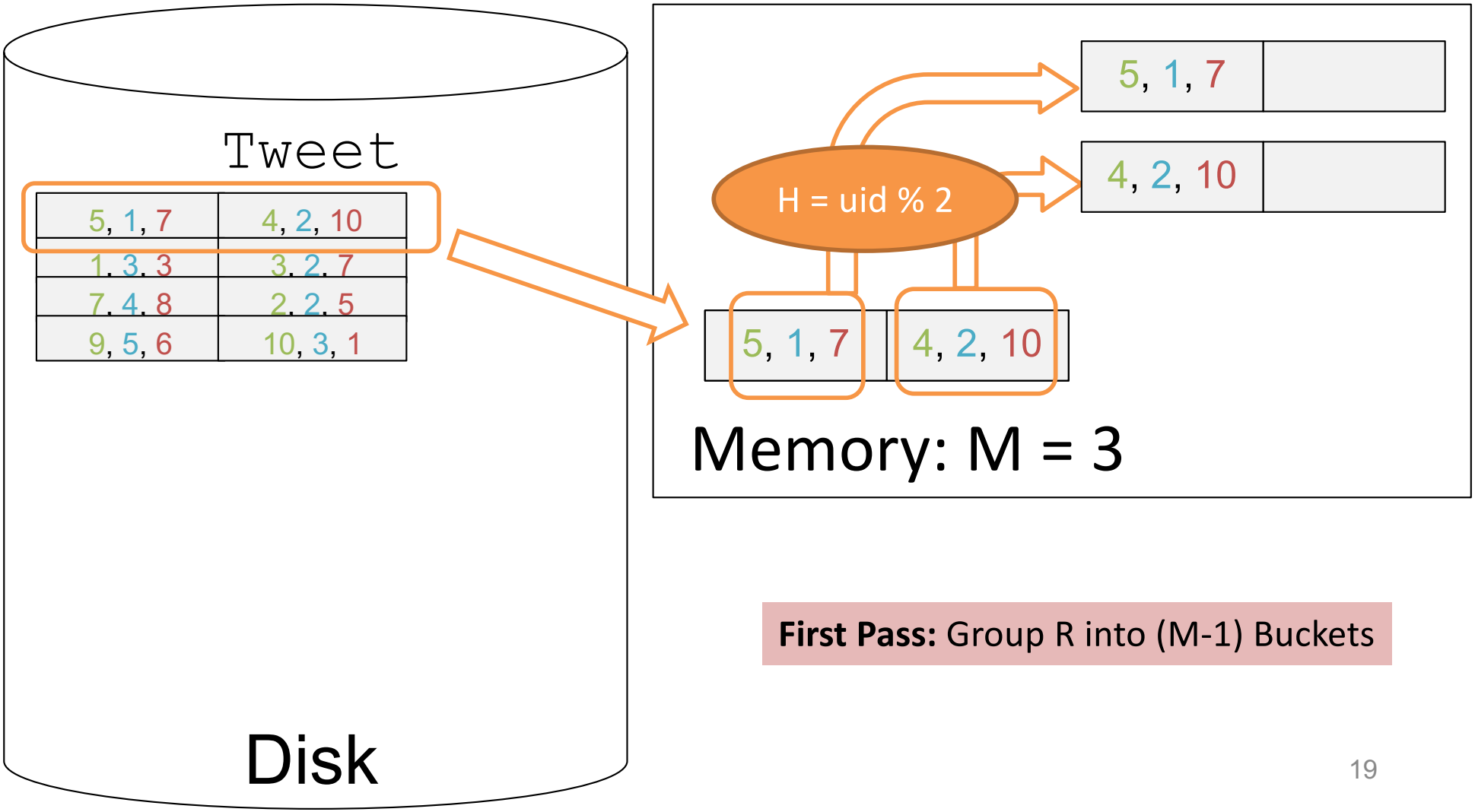
What if M-1 pages can't hold all groups?

Two Pass!

```
Tweet(tweet_id, user_id, tweet_length)
```

```
SELECT user_id, MIN(tweet_length)
FROM Tweet
GROUP BY user_id
```

Two pass, hash-based grouping

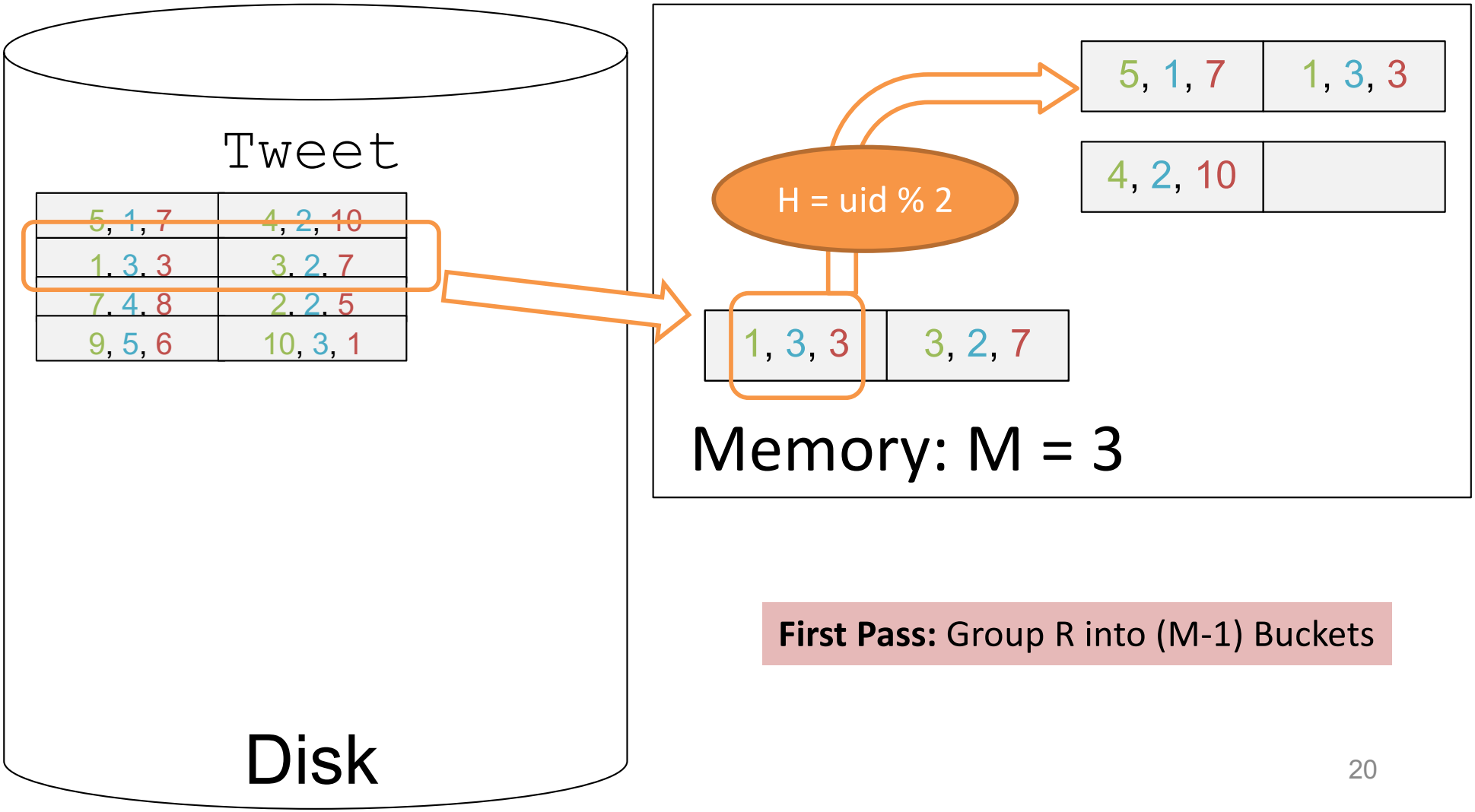


First Pass: Group R into (M-1) Buckets

```
Tweet(tweet_id, user_id, tweet_length)
```

```
SELECT user_id, MIN(tweet_length)
FROM Tweet
GROUP BY user_id
```

Two pass, hash-based grouping

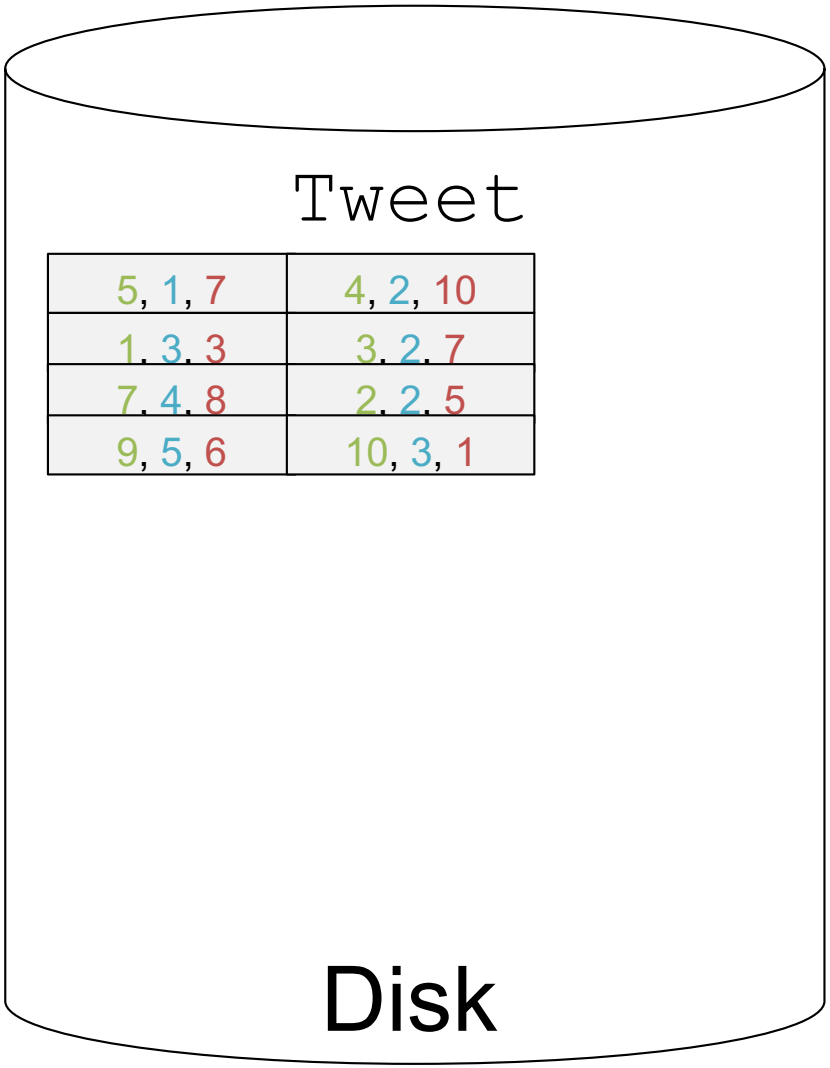


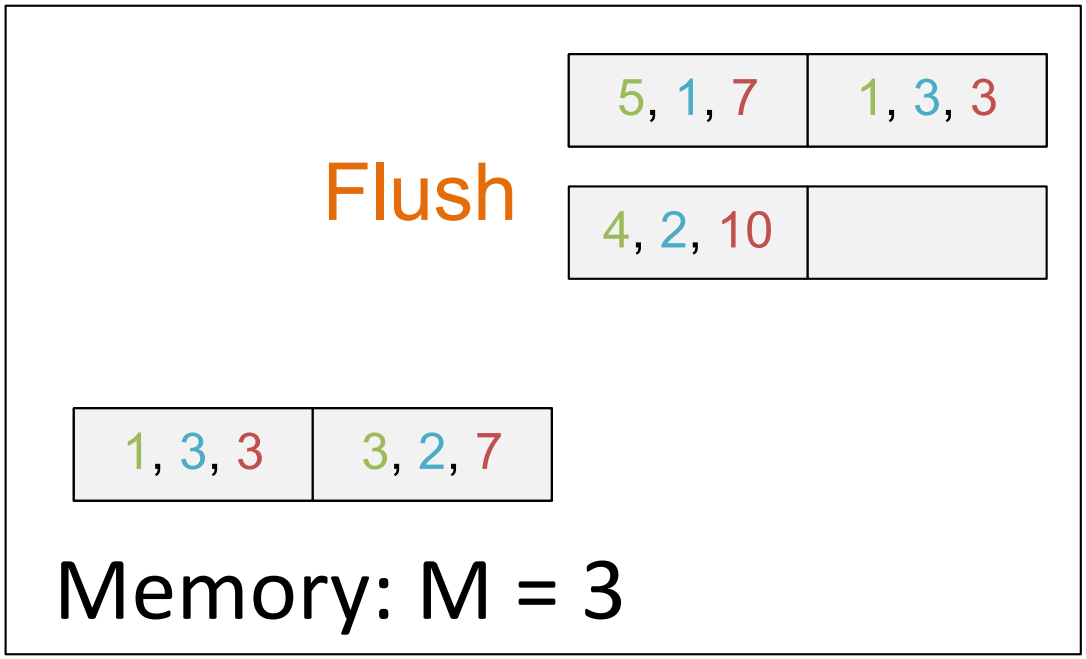
First Pass: Group R into (M-1) Buckets

```
Tweet(tweet_id, user_id, tweet_length)
```

```
SELECT user_id, MIN(tweet_length)
FROM Tweet
GROUP BY user_id
```

Two pass, hash-based grouping



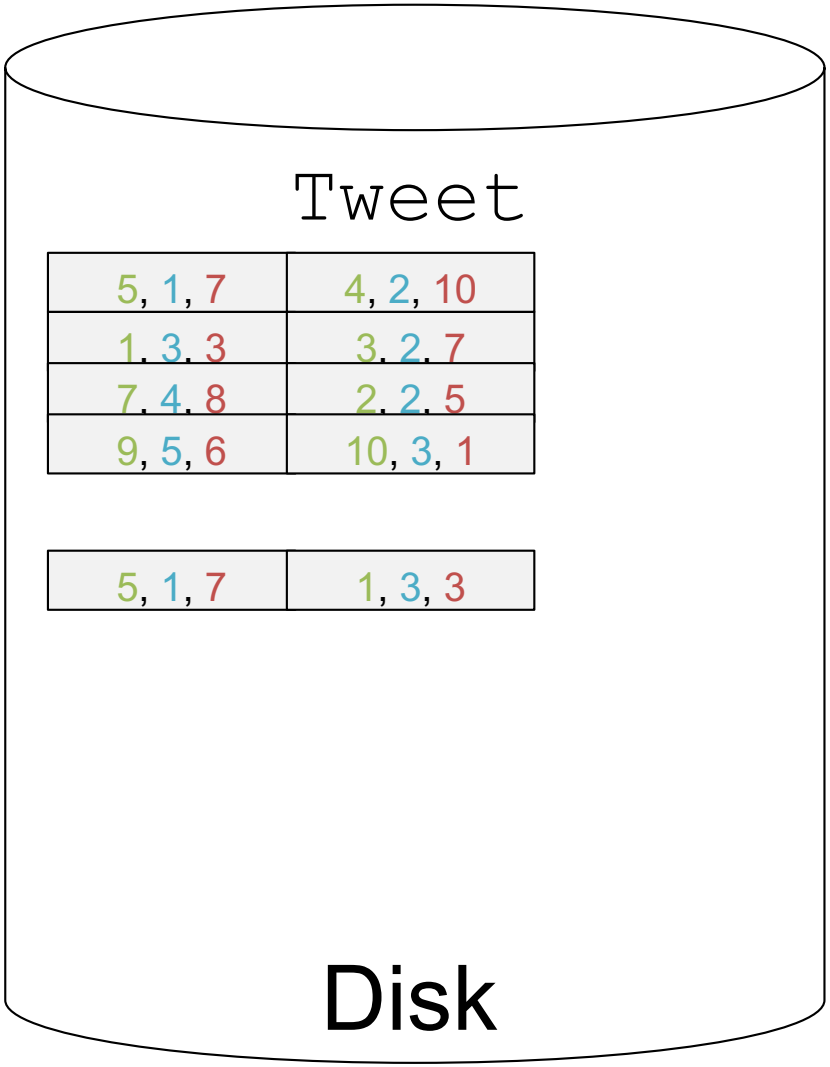


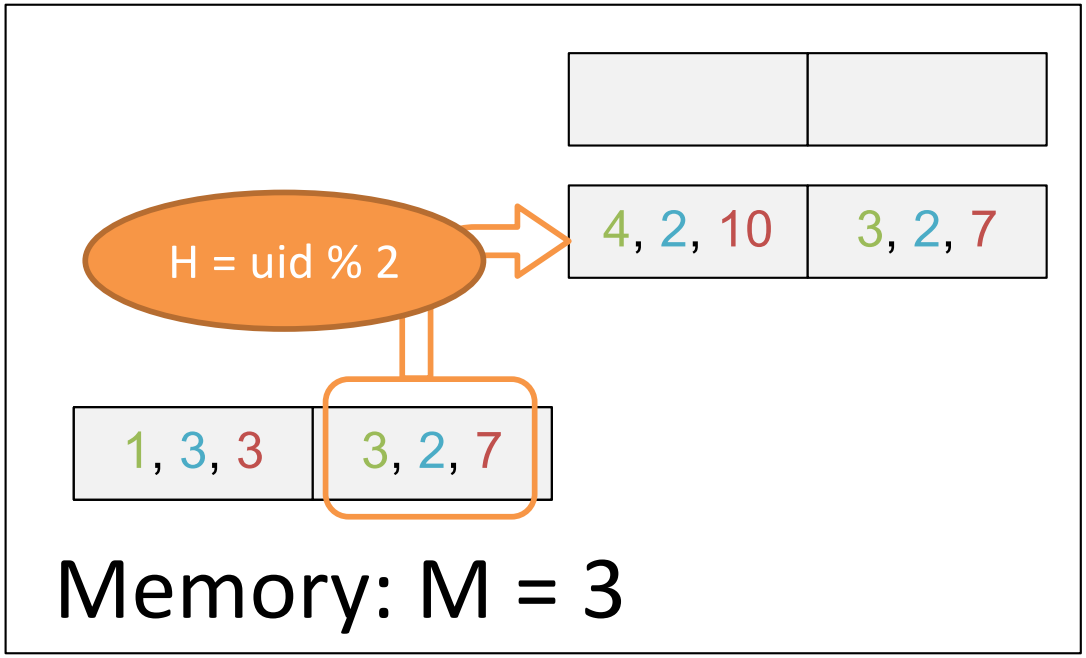
First Pass: Group R into (M-1) Buckets

```
Tweet(tweet_id, user_id, tweet_length)
```

```
SELECT user_id, MIN(tweet_length)
FROM Tweet
GROUP BY user_id
```

Two pass, hash-based grouping



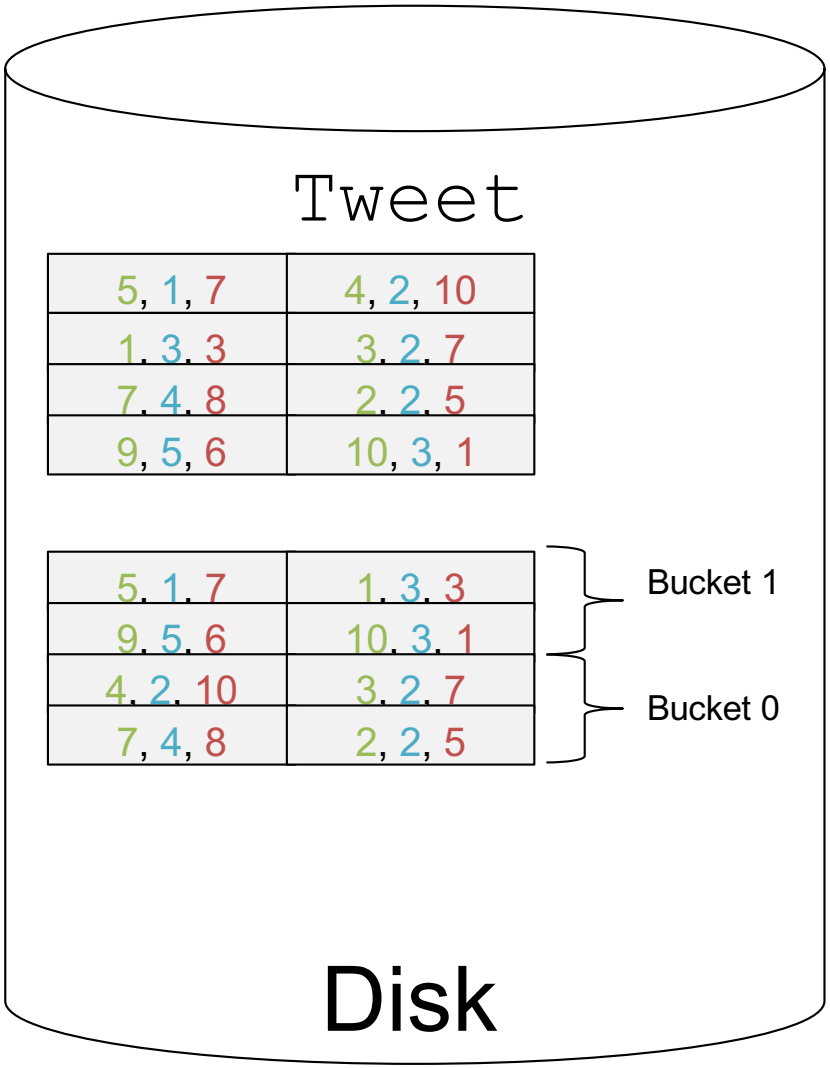


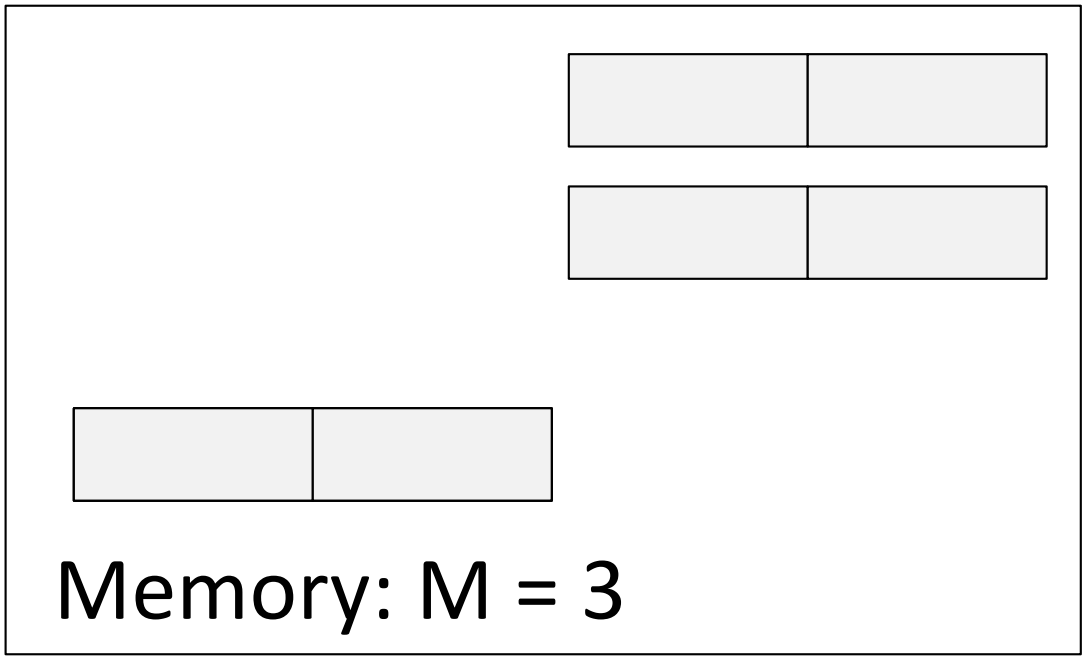
First Pass: Group R into (M-1) Buckets
Flush Full Pages

```
Tweet(tweet_id, user_id, tweet_length)
```

```
SELECT user_id, MIN(tweet_length)
FROM Tweet
GROUP BY user_id
```

Two pass, hash-based grouping

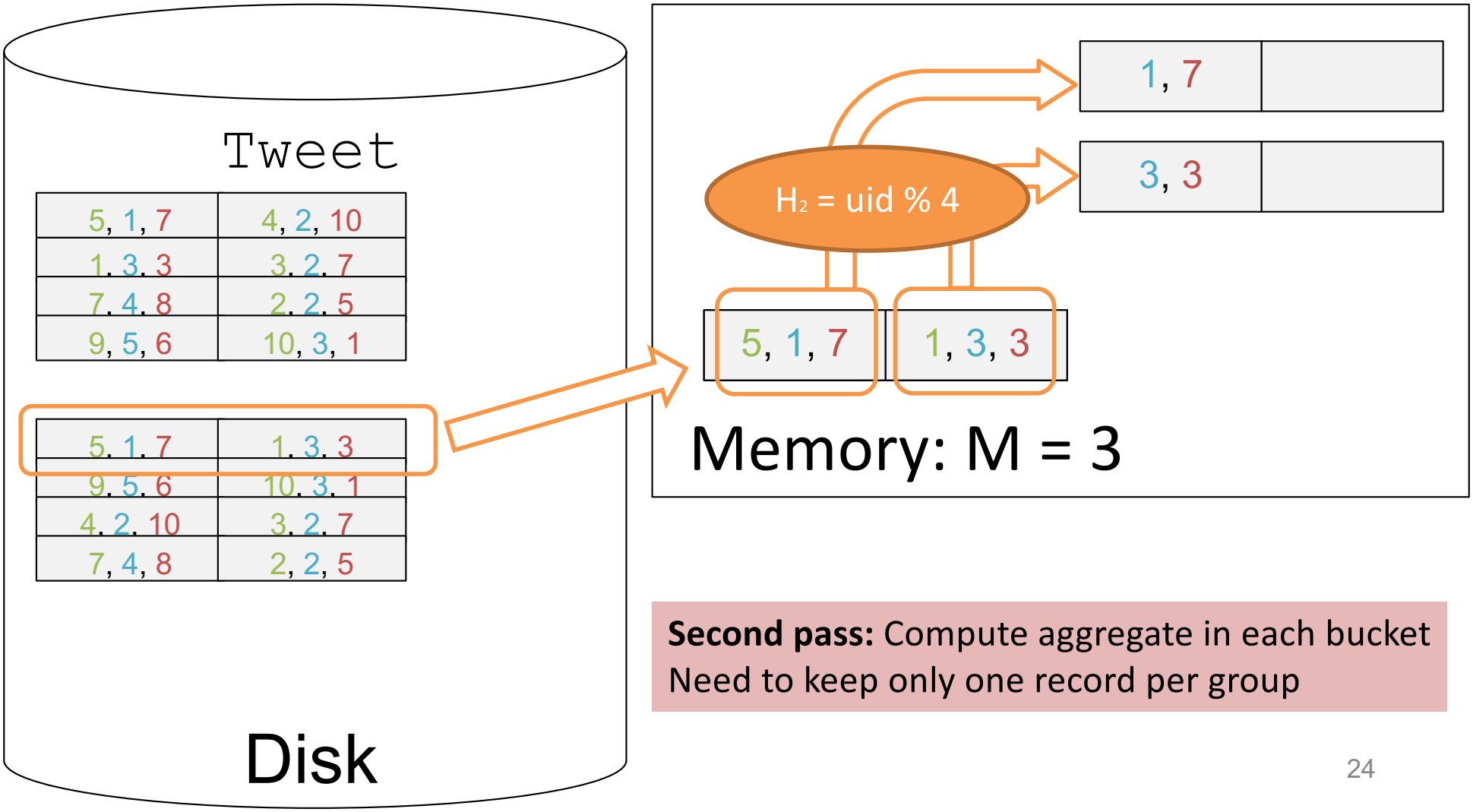




Disk and Memory after **First Pass**

```
Tweet(tweet_id, user_id, tweet_length)

SELECT user_id, MIN(tweet_length)
FROM Tweet
GROUP BY user_id
```

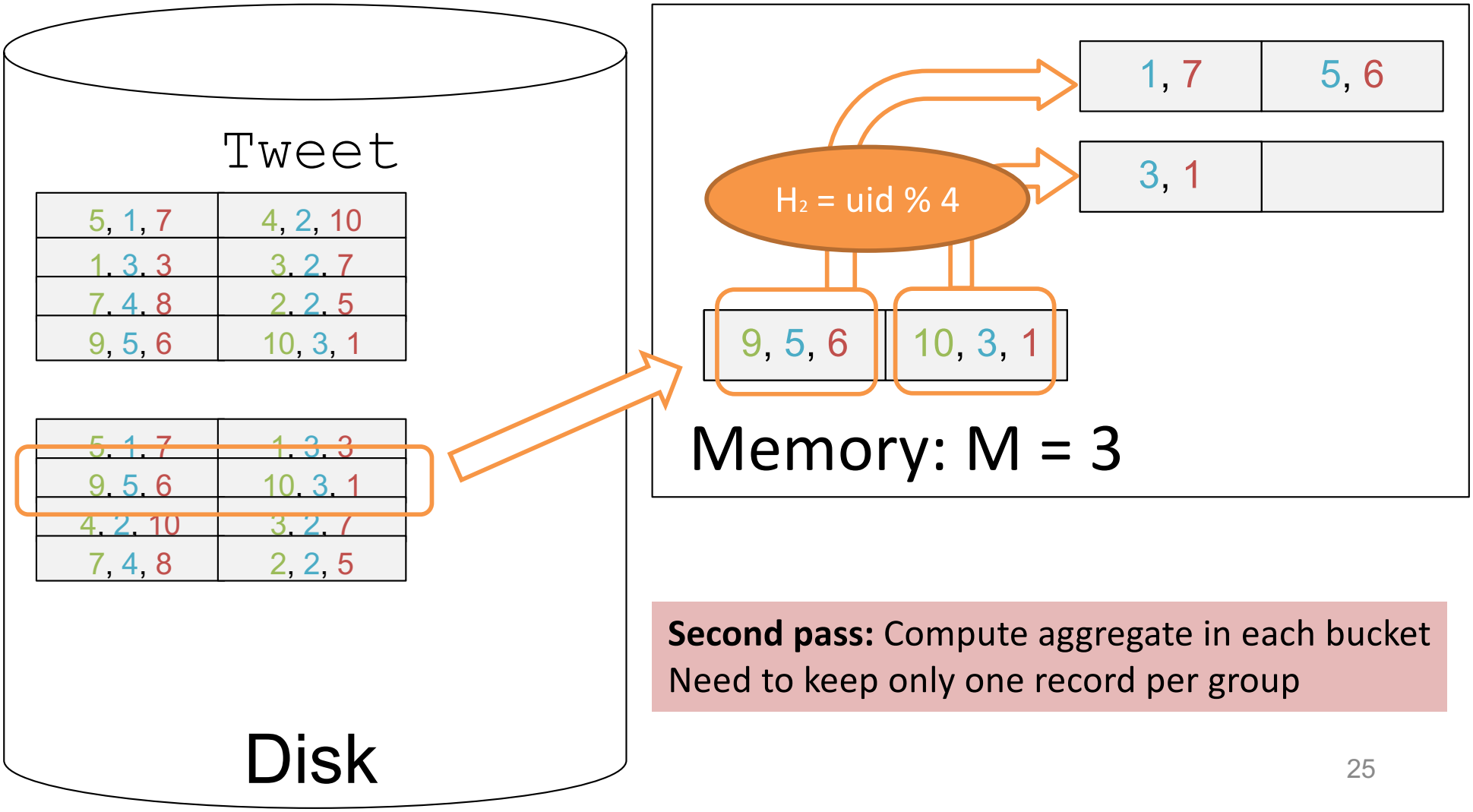
Two pass, hash-based grouping



Second pass: Compute aggregate in each bucket
Need to keep only one record per group

```
Tweet(tweet_id, user_id, tweet_length)

SELECT user_id, MIN(tweet_length)
FROM Tweet
GROUP BY user_id
```

Two pass, hash-based grouping



Second pass: Compute aggregate in each bucket
Need to keep only one record per group

Discussion

Cost?

- 3B(R)

Assumptions?

- Need to hold all distinct values in the same bucket in M-1
- Assuming uniformity, B(R) <= M^2 is safe to assume
  - i.e. B(R)/M <= M

```
Tweet(tweet_id, user_id, tweet_length)
```

```
SELECT user_id, MIN(tweet_length)
FROM Tweet
GROUP BY user_id
```

Two pass, sort-merge based grouping

Memory: M = 3

5, 1, 7	4, 2, 10
1, 3, 3	3, 2, 7
7, 4, 8	2, 2, 5

Step 1:
Divide R into (M-1) partitions
Sort each partition in memory
(on group by attr = user_id)
Write to disk

Tweet

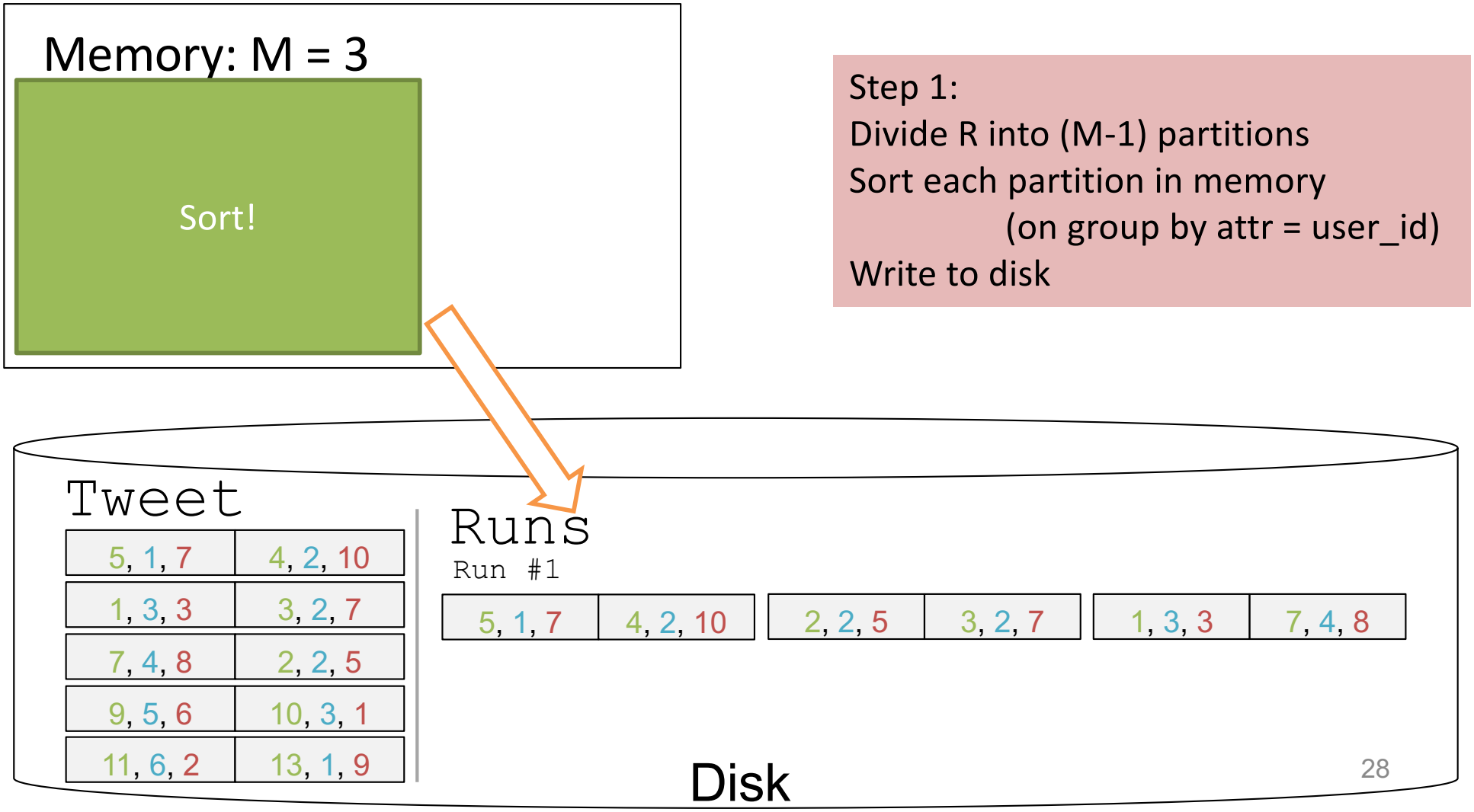
5, 1, 7	4, 2, 10
1, 3, 3	3, 2, 7
7, 4, 8	2, 2, 5
9, 5, 6	10, 3, 1
11, 6, 2	13, 1, 9

Disk

```
Tweet(tweet_id, user_id, tweet_length)
```

```
SELECT user_id, MIN(tweet_length)
FROM Tweet
GROUP BY user_id
```

Two pass, sort-merge based grouping

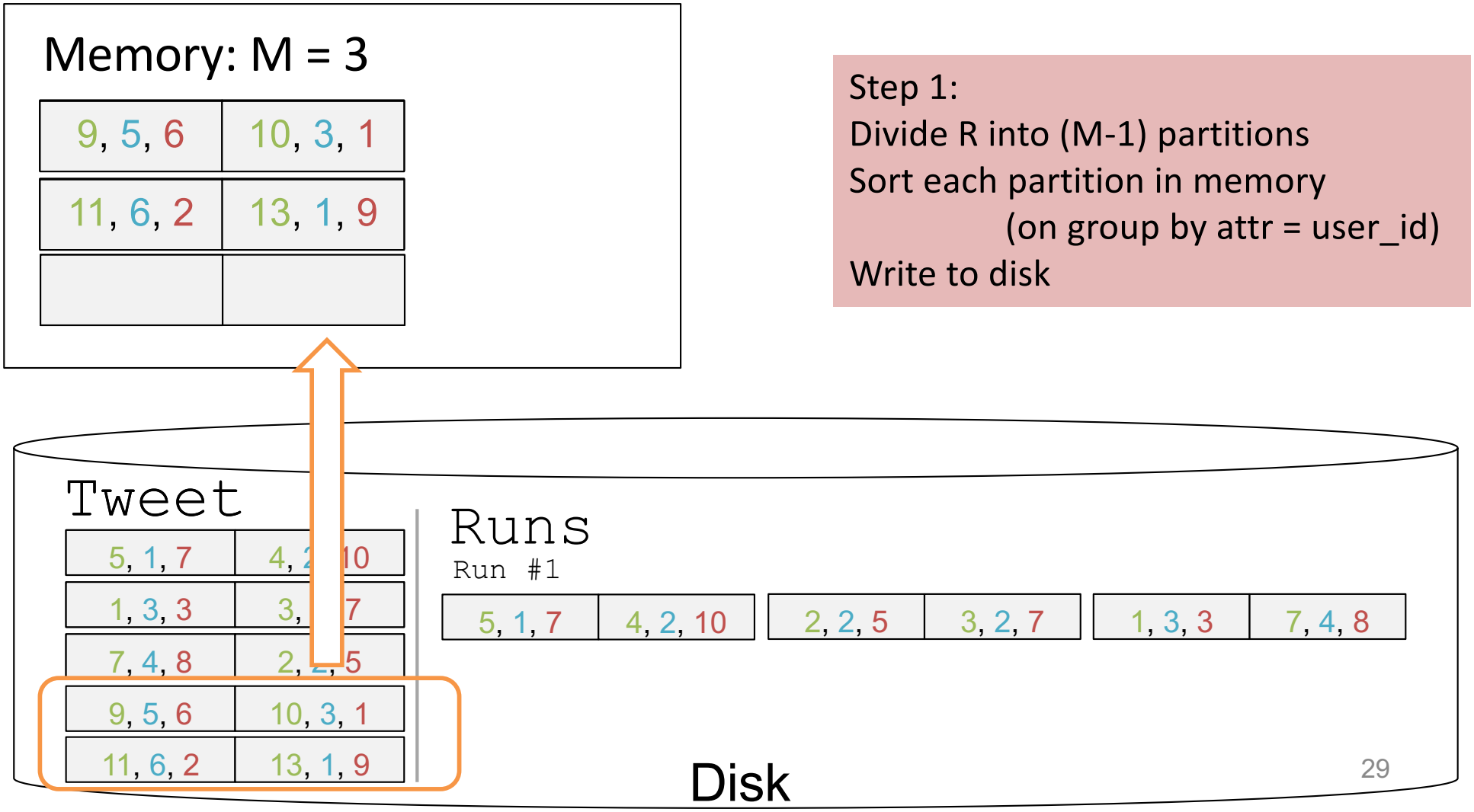


Step 1:
Divide R into (M-1) partitions
Sort each partition in memory
(on group by attr = user_id)
Write to disk

```
Tweet(tweet_id, user_id, tweet_length)
```

```
SELECT user_id, MIN(tweet_length)
FROM Tweet
GROUP BY user_id
```

Two pass, sort-merge based grouping

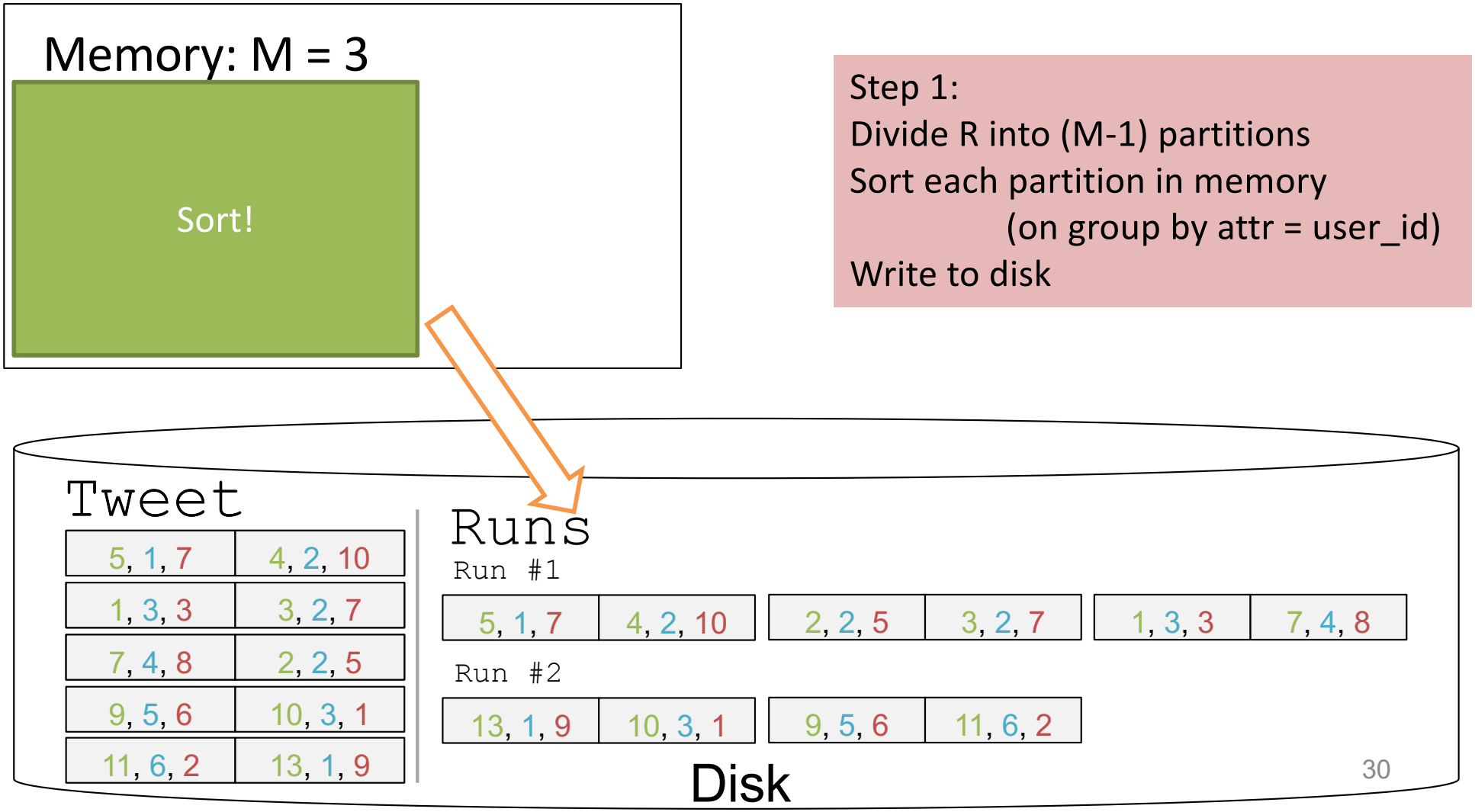


Step 1:
Divide R into (M-1) partitions
Sort each partition in memory
(on group by attr = user_id)
Write to disk

```
Tweet(tweet_id, user_id, tweet_length)
```

```
SELECT user_id, MIN(tweet_length)
FROM Tweet
GROUP BY user_id
```

Two pass, sort-merge based grouping

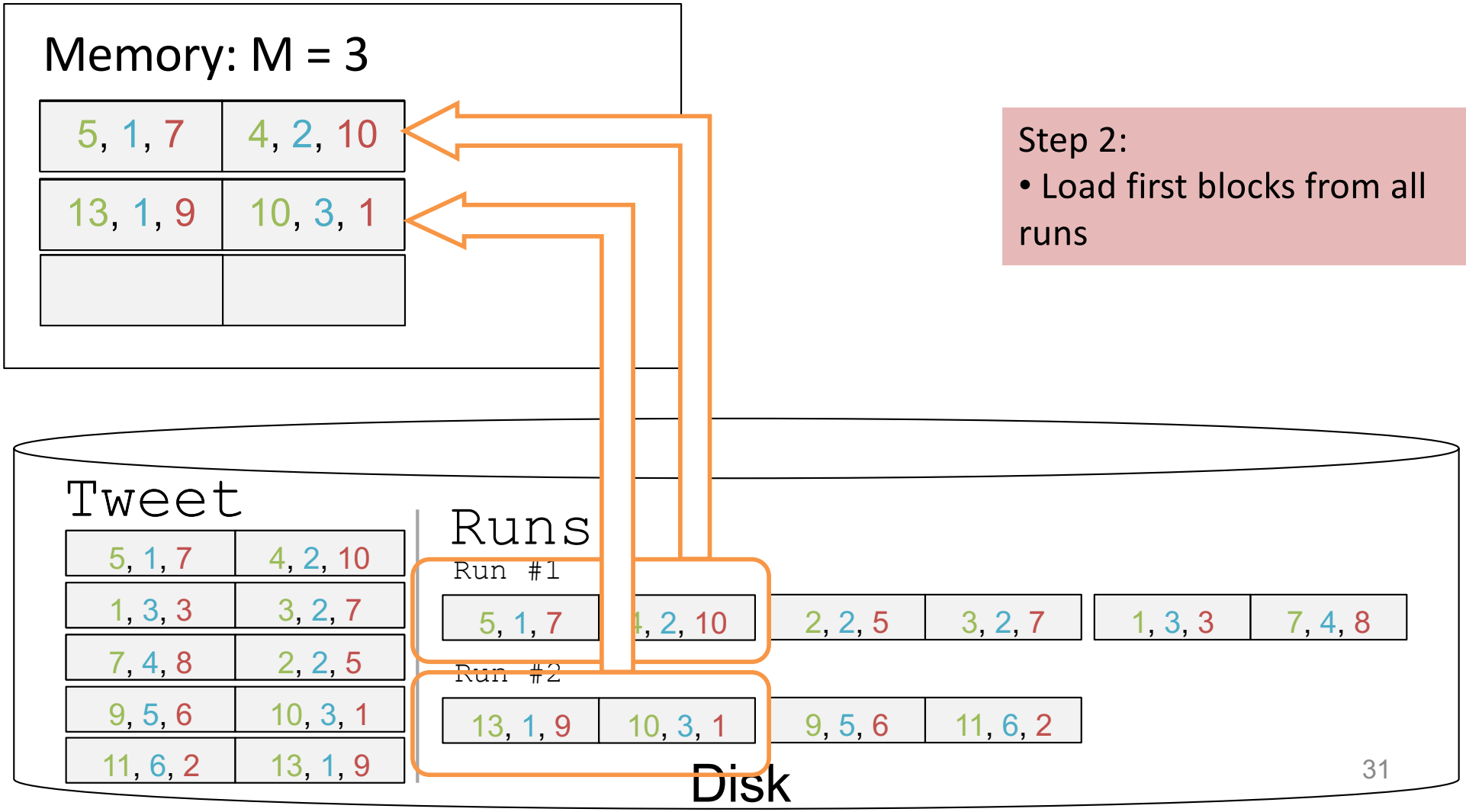


Step 1:
Divide R into (M-1) partitions
Sort each partition in memory
(on group by attr = user_id)
Write to disk

```
Tweet(tweet_id, user_id, tweet_length)
```

```
SELECT user_id, MIN(tweet_length)
FROM Tweet
GROUP BY user_id
```

Two pass, sort-merge based grouping

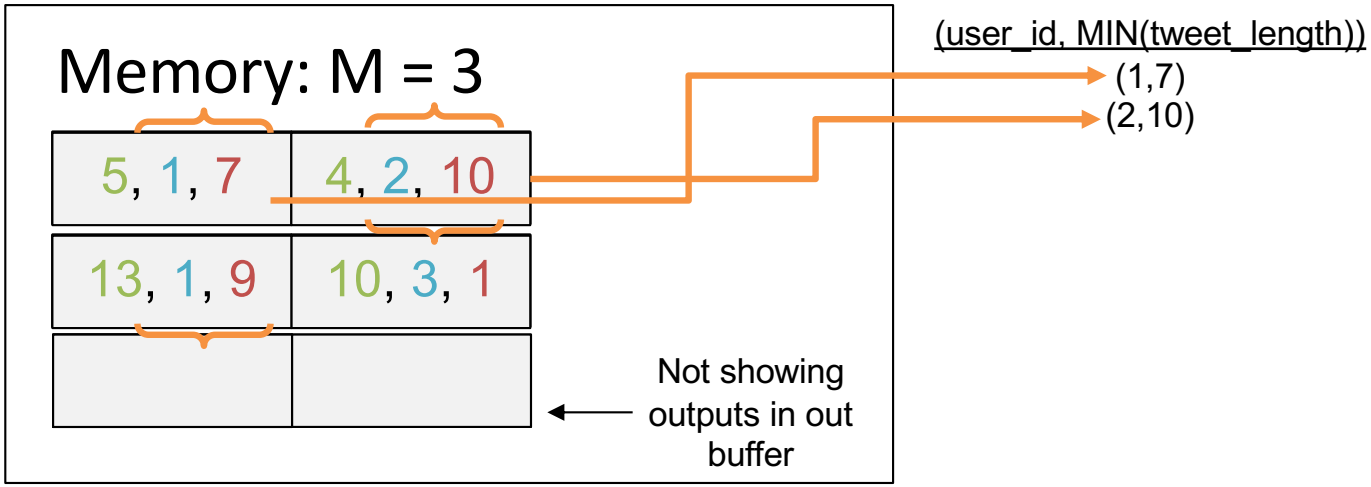


Step 2:
- Load first blocks from all runs

```
Tweet(tweet_id, user_id, tweet_length)
```

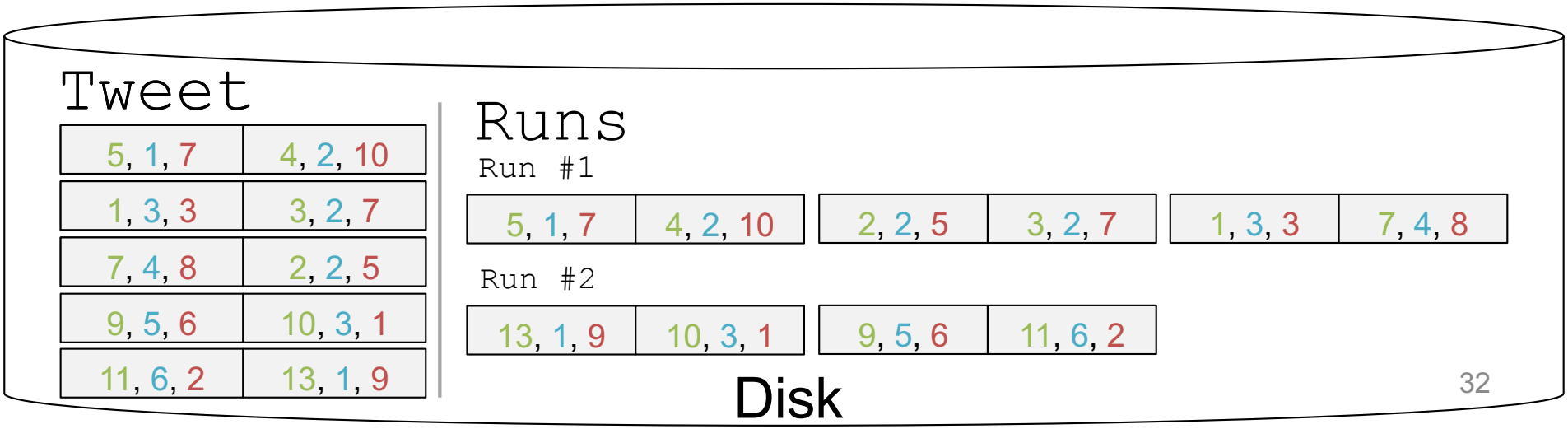
```
SELECT user_id, MIN(tweet_length)
FROM Tweet
GROUP BY user_id
```

Two pass, sort-merge based grouping



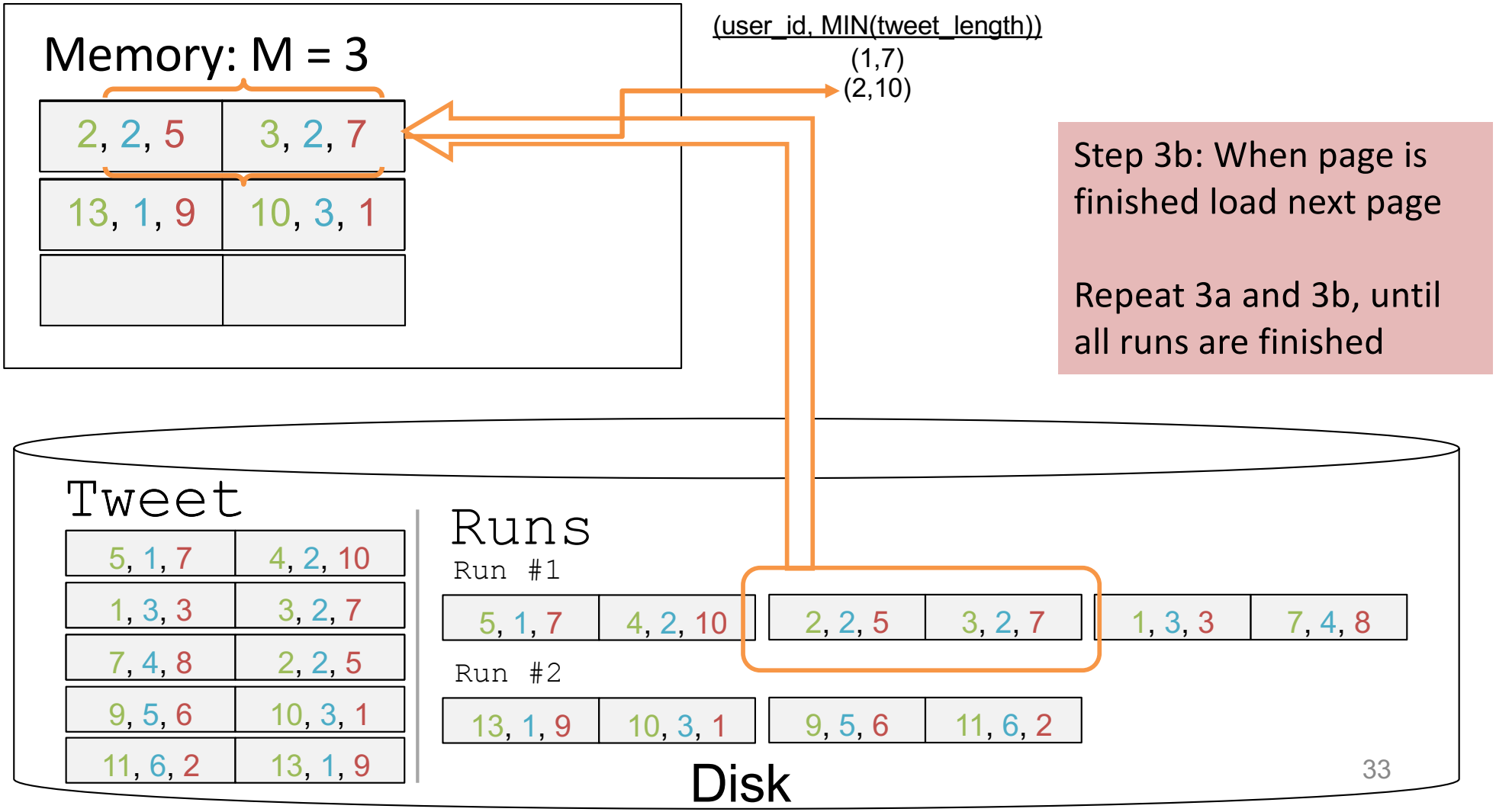
Step 3a: Find minimum of each key by "Combine" approach in merge-sort

Repeatedly find the least value of the sort key



```
Tweet(tweet_id, user_id, tweet_length)

SELECT user_id, MIN(tweet_length)
FROM Tweet
GROUP BY user_id
```

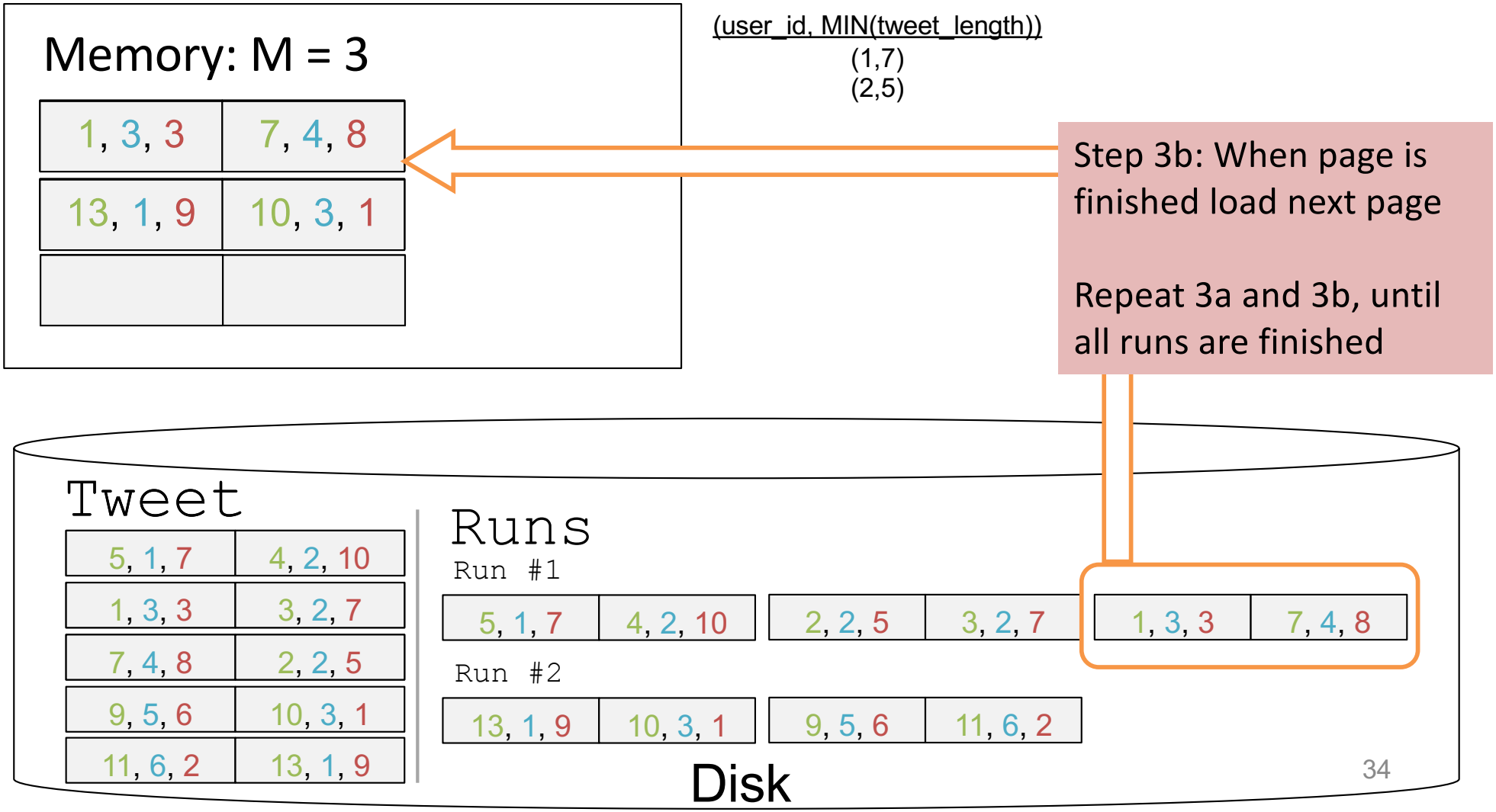
Two pass, sort-merge based grouping



```
Tweet(tweet_id, user_id, tweet_length)
```

```
SELECT user_id, MIN(tweet_length)
FROM Tweet
GROUP BY user_id
```

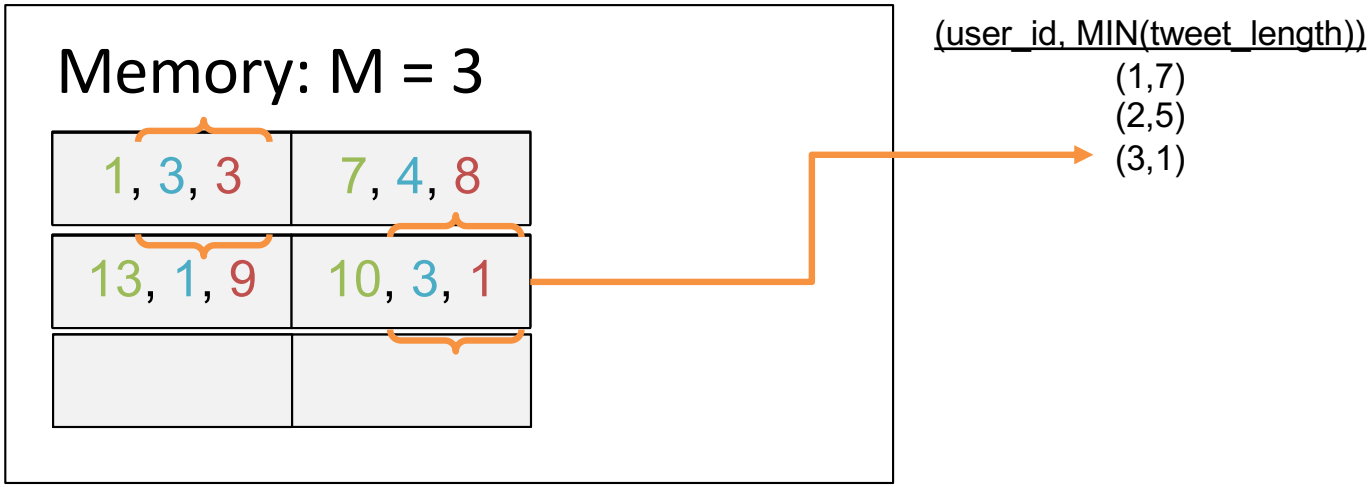
Two pass, sort-merge based grouping



```
Tweet(tweet_id, user_id, tweet_length)
```

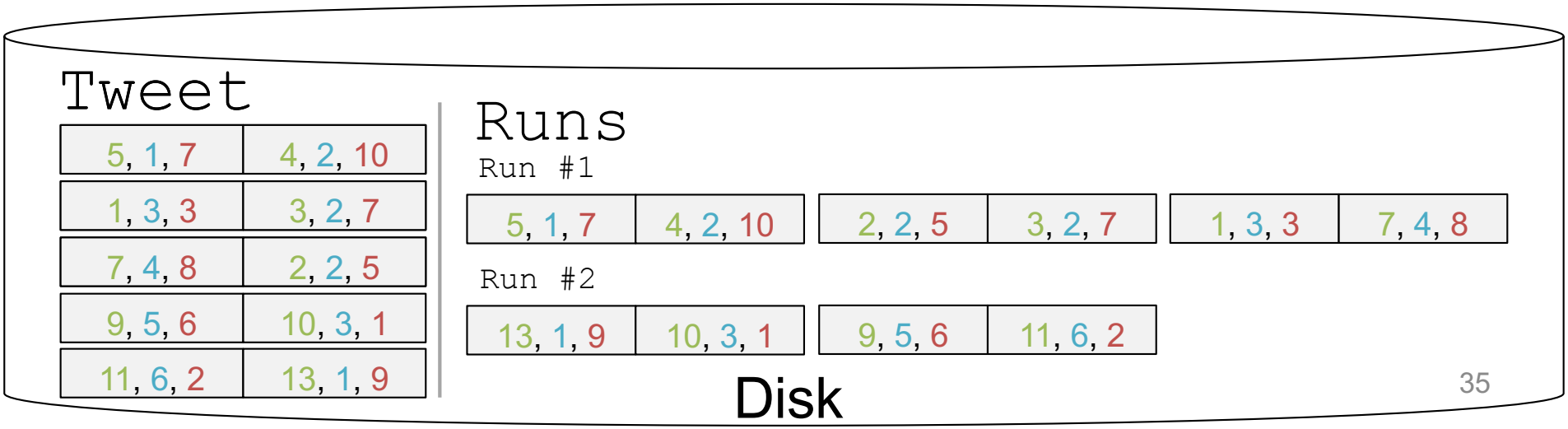
```
SELECT user_id, MIN(tweet_length)
FROM Tweet
GROUP BY user_id
```

Two pass, sort-merge based grouping



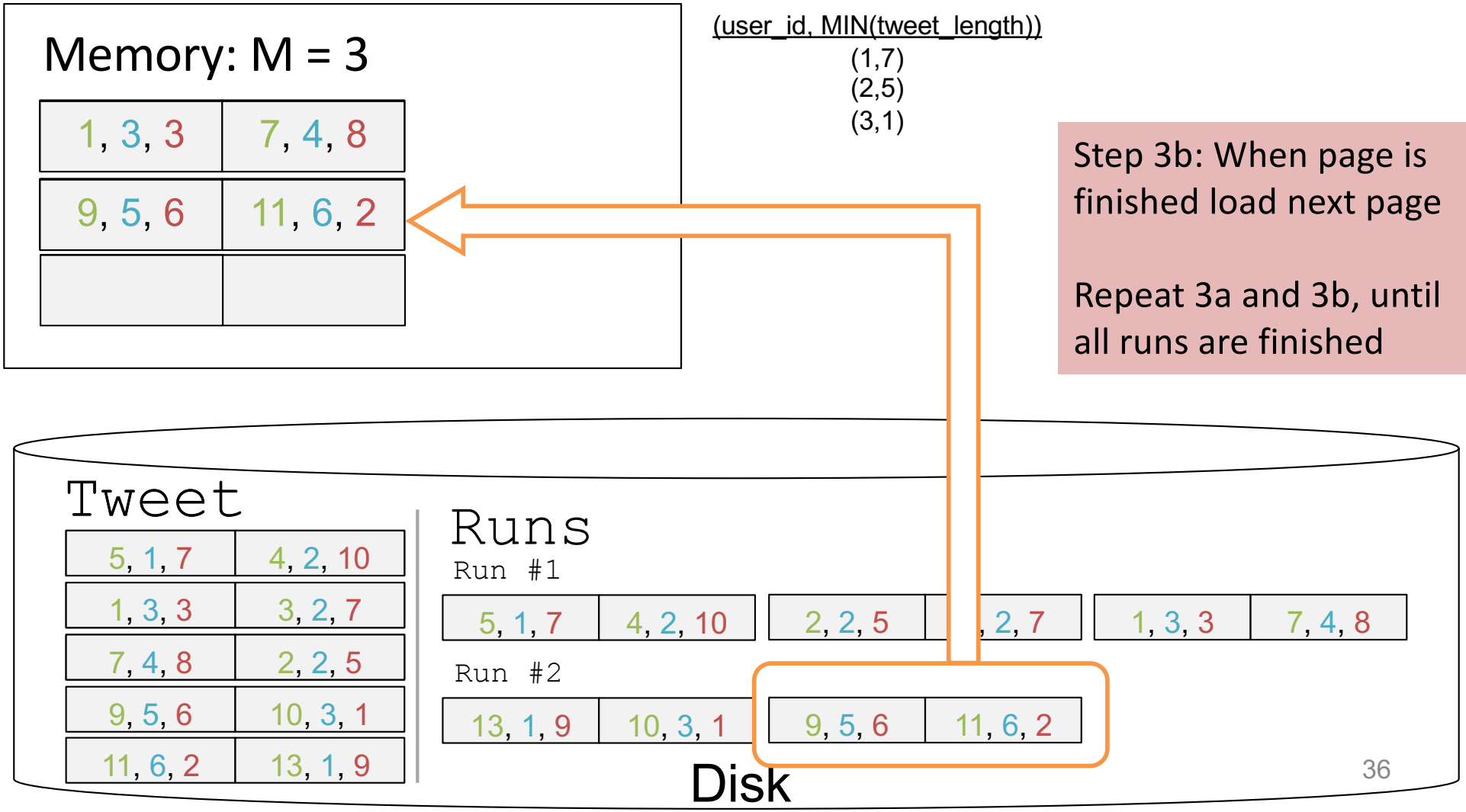
Step 3b: When page is finished load next page

Repeat 3a and 3b, until all runs are finished



```
Tweet(tweet_id, user_id, tweet_length)

SELECT user_id, MIN(tweet_length)
FROM Tweet
GROUP BY user_id
```

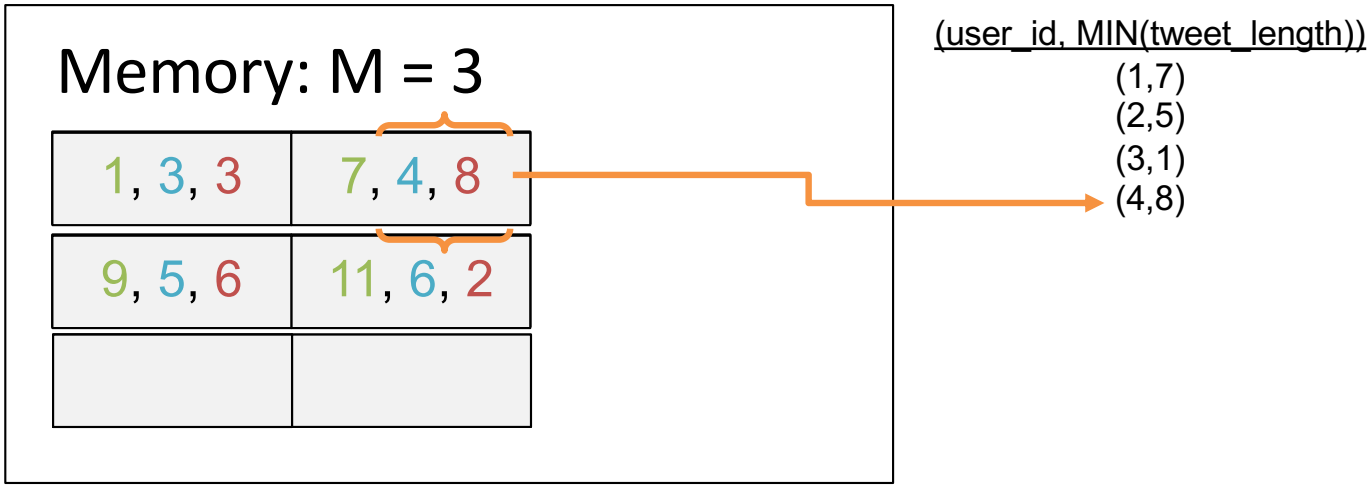
Two pass, sort-merge based grouping



```
Tweet(tweet_id, user_id, tweet_length)
```

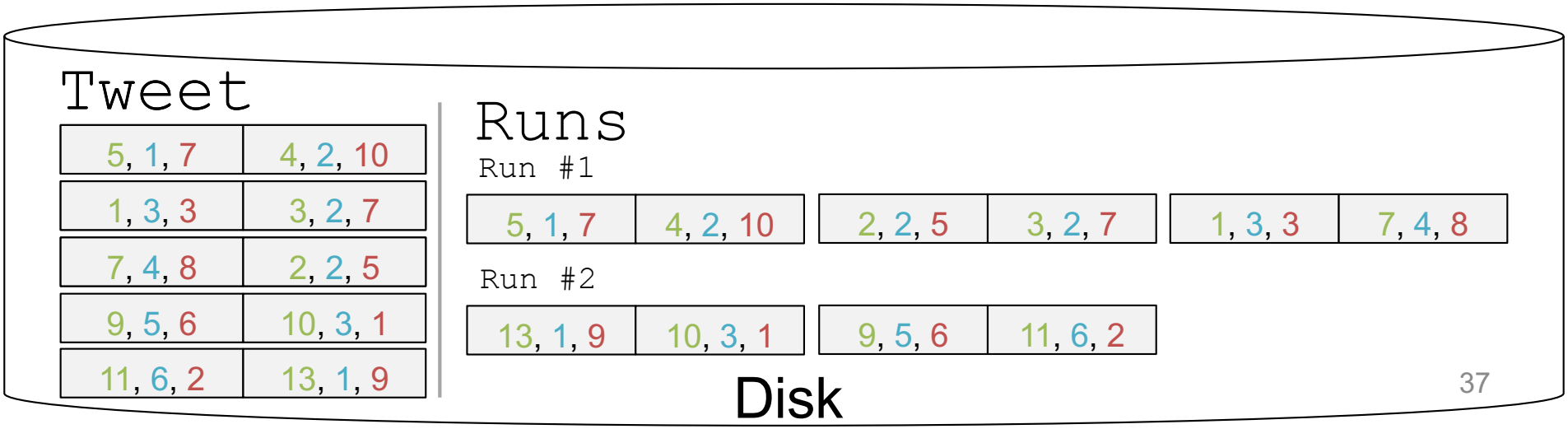
```
SELECT user_id, MIN(tweet_length)
FROM Tweet
GROUP BY user_id
```

Two pass, sort-merge based grouping



Step 3b: When page is finished load next page

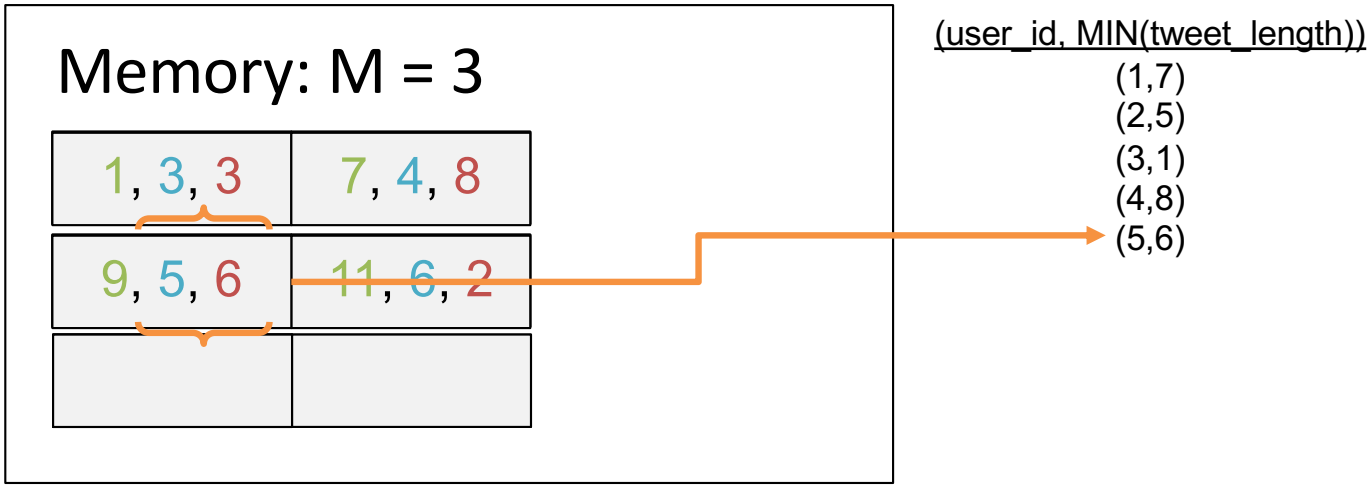
Repeat 3a and 3b, until all runs are finished



```
Tweet(tweet_id, user_id, tweet_length)
```

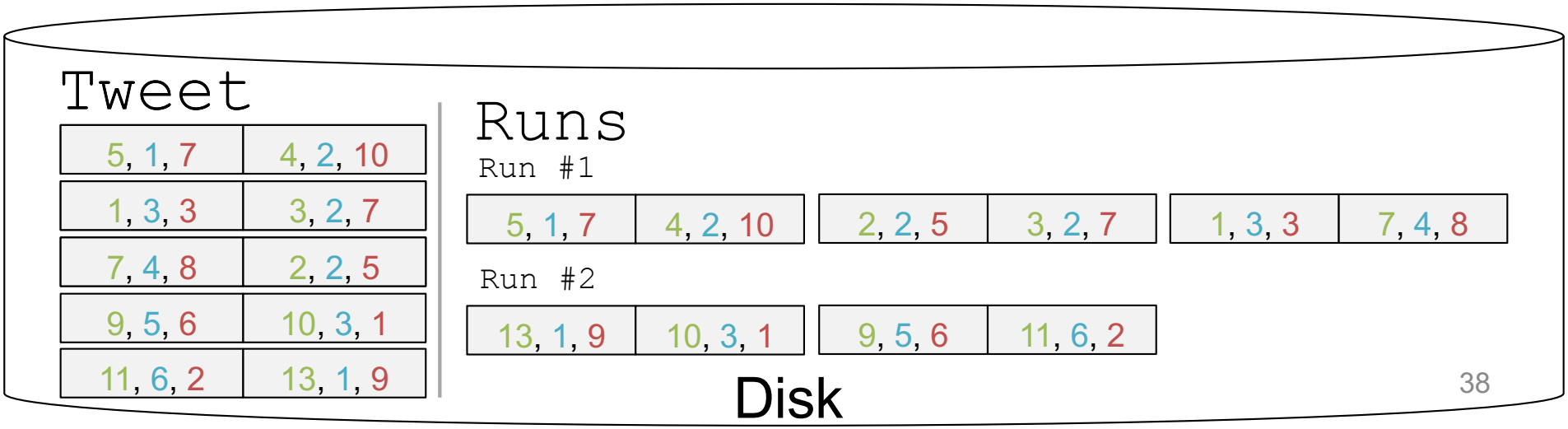
```
SELECT user_id, MIN(tweet_length)
FROM Tweet
GROUP BY user_id
```

Two pass, sort-merge based grouping



Step 3b: When page is finished load next page

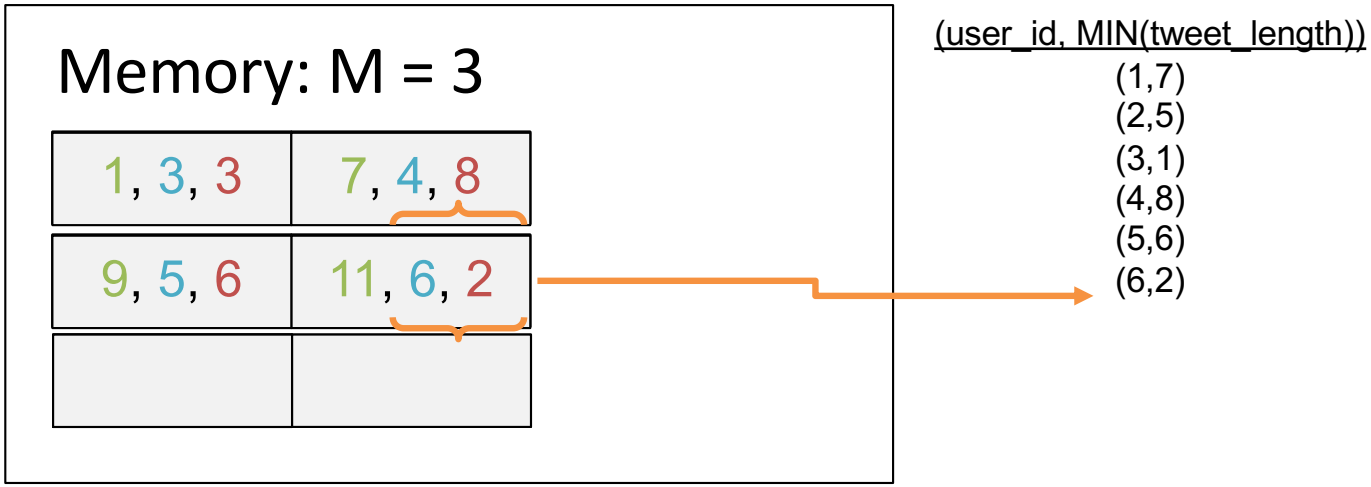
Repeat 3a and 3b, until all runs are finished



```
Tweet(tweet_id, user_id, tweet_length)
```

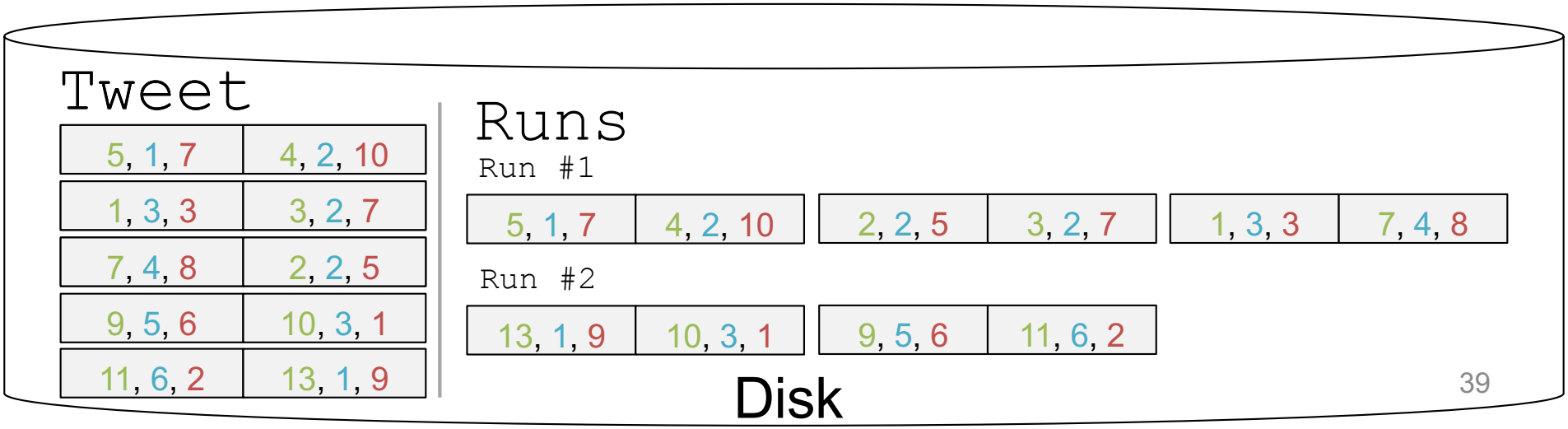
```
SELECT user_id, MIN(tweet_length)
FROM Tweet
GROUP BY user_id
```

Two pass, sort-merge based grouping



Step 3b: When page is finished load next page

Repeat 3a and 3b, until all runs are finished



Discussion

Cost?

- 3B(R)

Assumptions?

- Need to hold one block from each run in M pages
- $B(R) <= M^2$ or $(B(R)/M <= M)$

One pass vs. Two pass

- One pass:
  - smaller disk I/O cost
    - e.g. B(R) for one-pass hash-based aggregation
  - Handles smaller relations
    - e.g. $B(R) < M$
- Two/Multi pass:
  - Larger disk I/O cost
    - e.g. 3B(R) for two-pass hash-based aggregation
  - Can handle larger relations
    - e.g. $B(R) < M^2$

One pass vs. Two pass

What if $B(R) > M^2$?

- One pass: $B(R) < M$ -> $B(R)/1 < M$
- Two pass: $B(R) < M^2$ -> $B(R)/M < M$
- Three pass: $B(R) < M^3$ -> $B(R)/ M^2 < M$

Review for Joins

- Two-pass Hash-based Join
  - Cost: 3B(R) + 3B(S)
  - Assumption: Min(B(R), B(S)) <= M^2
- Two-pass Sort-merge-based Join
  - Implementation:
    - Cost: 3(R) + 3B(S)
      - For R, S: sort into a run (2 I/O, read + write)
      - Join by combining R and S (only read, write not counted - 1 I/O)

Homework 2

- Problem 1
  - B+ Trees (inserting/deleting/lookups)
- Problem 2
  - Operator Algorithms
- Problem 3
  - Multi-Pass Algorithms