External Algorithms
and
Query Evaluation

(Continued)

R&G Chapter 12,**13**,14

(slides adapted from content by J.Gehrke, J.Shanmugasundaram, and/or C.Koch)

Project 1

- Project 1 will be posted later today.
  - Due Mon, Feb 18 (2 weekends)
  - 2-3 Person Groups
  - 2 parts in Java
    - 1-Answer queries posed in RA
    - 2-Generate RA from SQL
- In-depth project discussion on Friday.

Review

- Nested-Loop Join (Cartesian Cross-Product)
  - For Each(A) { For Each(B) { emit(A, B); }}
  - Join Predicate implemented though Selection
- High Cost
  - O(|A| * |B|) operations
  - If |B| doesn't fit in memory, it must be fully re-read |A| times.

Review

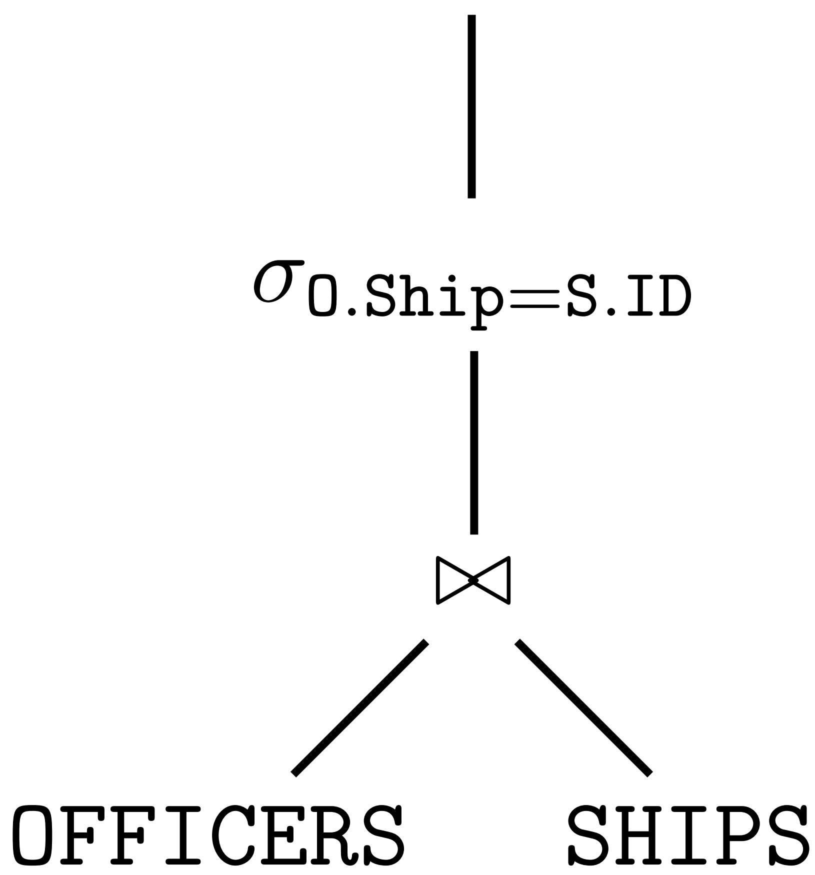
- Nested-Loop Join (Cartesian Cross-Product)
  - For Each(A) { For Each(B) { emit(A, B); }}
  - Join Predicate implemented though Selection
- High Cost
  - O(|A| * |B|) operations
  - If |B| doesn't fit in memory, it must be fully re-read |A| times.

How do we reduce the cost?

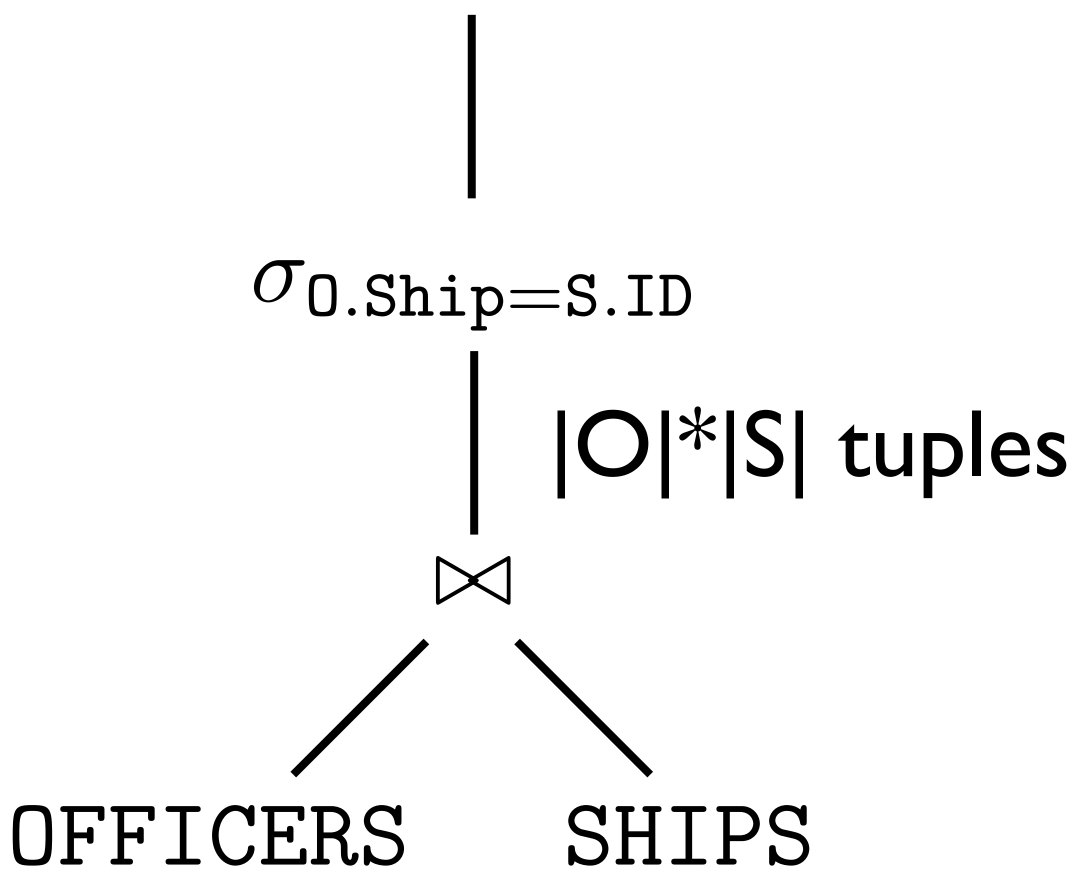
Review

- Block Nested Loop Join
  - Minimize IO cost if inner relation doesn't fit in memory.
  - Divide each relation into chunks.
  - Load pairs of chunks into memory.
  - Do a NLJ on tuples in each chunk pair.

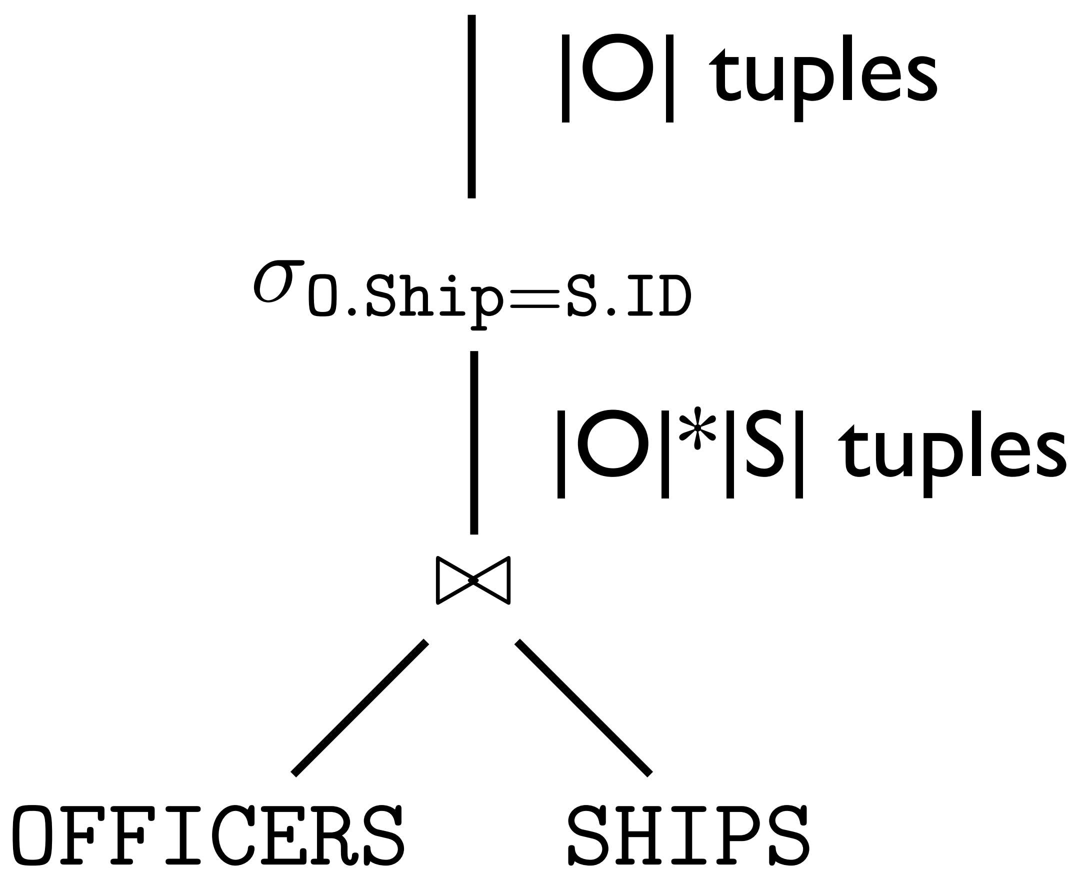
Review



Review

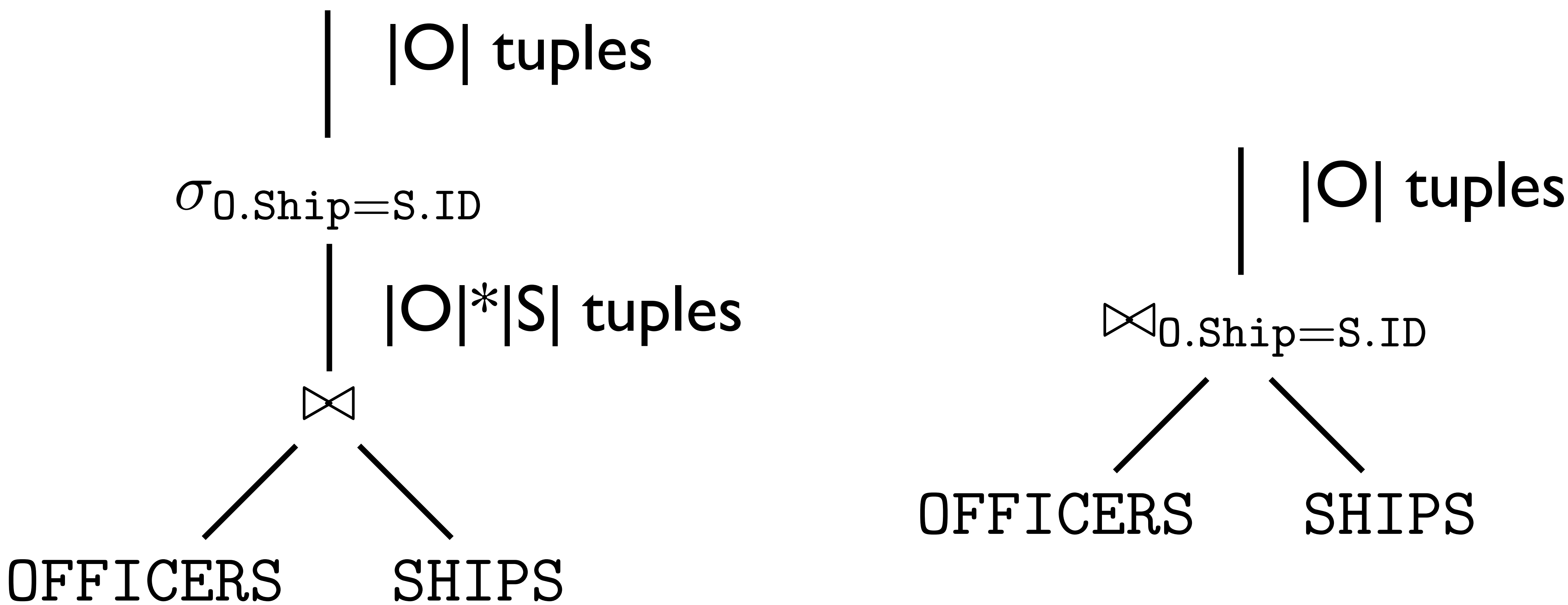


Review


|O| tuples
$\sigma_{\texttt{O.Ship=S.ID}}$
|O|*|S| tuples
$\bowtie$
OFFICERS
SHIPS

Review

Equi-joins exploit highly-selective equality join predicates

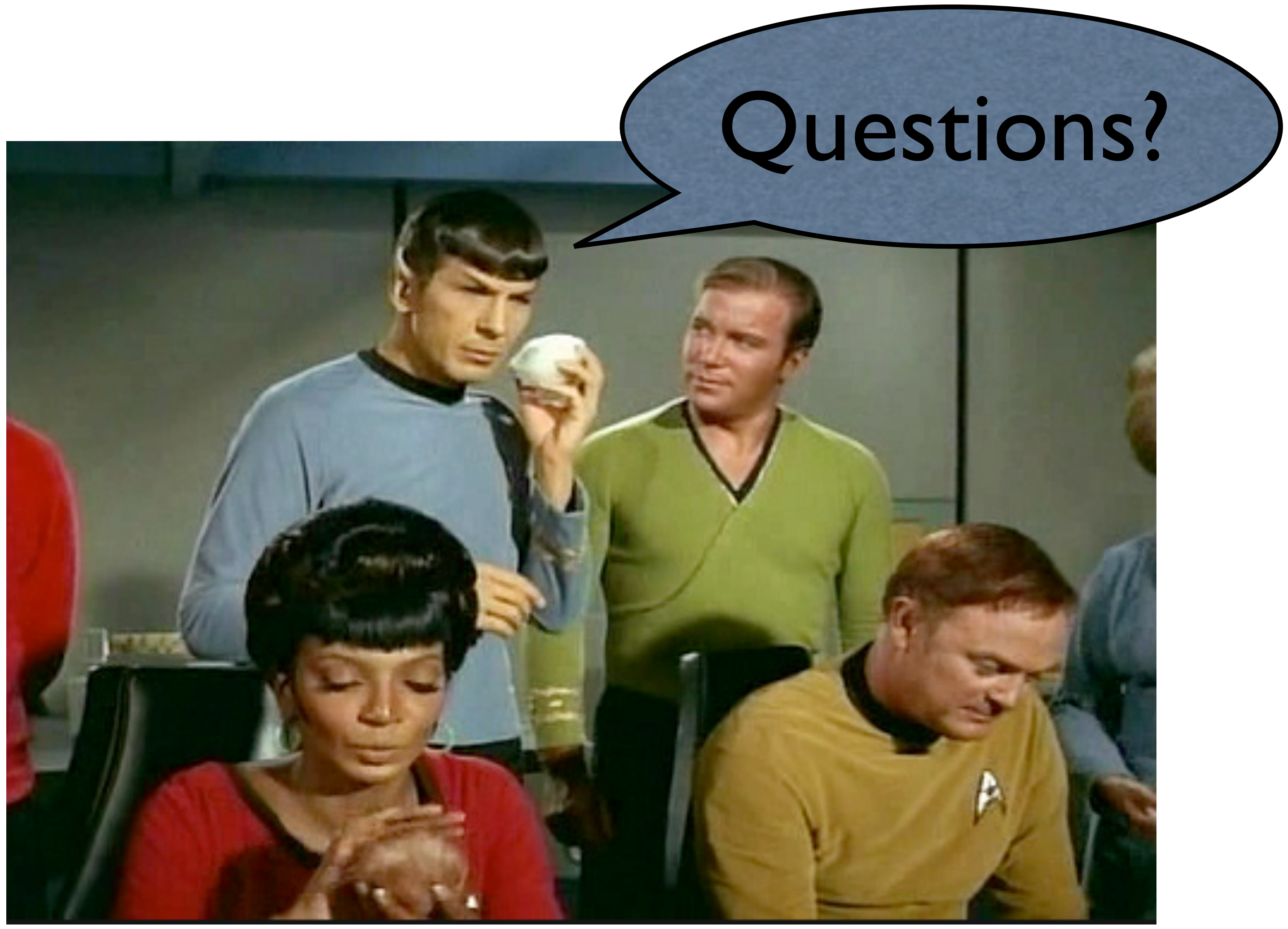


Review

- How do we avoid wasting effort on irrelevant tuple pairs.
  - Organize both relations to put joinable tuples together
    - Sort on the equality attribute (sort-merge join)
    - Hash on the equality attribute (hash join)
  - Organize one relation, so you can find joinable tuples more easily.
    - Hash table keyed on the equality attribute (hybrid hash join)
    - Build a tree on the attribute (index join)

Review

- Pipelining (aka Streaming)
    - Some (non-blocking) operators can operate on individual data values (e.g., select, bag-project)
    - Other (blocking) operators need the entire relation (e.g., distinct, sort, aggregate)
- Several join algorithms only block on one of the two input relations. (e.g., Nested Loop, Hybrid Hash, Index)



Tribbles are much like tuples created by a nested loop join. They may be fun and easygoing for a while, but they can easily overwhelm your system.

Implementing: Aggregates

General Solution: Iterators

	SUM()	AVG()
Intermediate State (Running Information)	`Total`	`< Total,Count >`
`Update(Value,IS)`	`Value+IS`	`< IS.Total+Value, IS.Count+1 >`
`Finalize(IS)`	`IS`	`IS.Total/IS.Count`

Implementing: Aggregates

General Solution: Iterators

Classes of Aggregate Functions [1]

Distributive: F(A, B, C, D) = F(F(A, B), F(C,D))

e.g., Sum, Count, Min/Max

Algebraic: F(A, B, C, D) = G(H(A, B), H(C,D))

e.g., Avg, Std Dev

Holistic: Unbounded Intermediate State

e.g., Median, Mode

[1] Grey et al. "Data Cube: A Relational Aggregation Operator..."

For a function to be Algebraic, the output of H must be of constant (or at least bounded) size.
Distributive functions are a special case of algebraic functions where G = H = F
Holistic functions are hard to implement "efficiently", but can be approximated. Time permitting, we will return to this idea later in the term. If you're interested, check out "online aggregation" and "data sketching" on Google Scholar.

Implementing: Grouping

Solution 1 (Hash)

- In-Memory:
  - Keep a hash table from group-keys to the intermediate state for the group
- On-disk
  - Partition the data into buckets (one scan)
  - In-memory grouping for each bucket

Implementing: Grouping

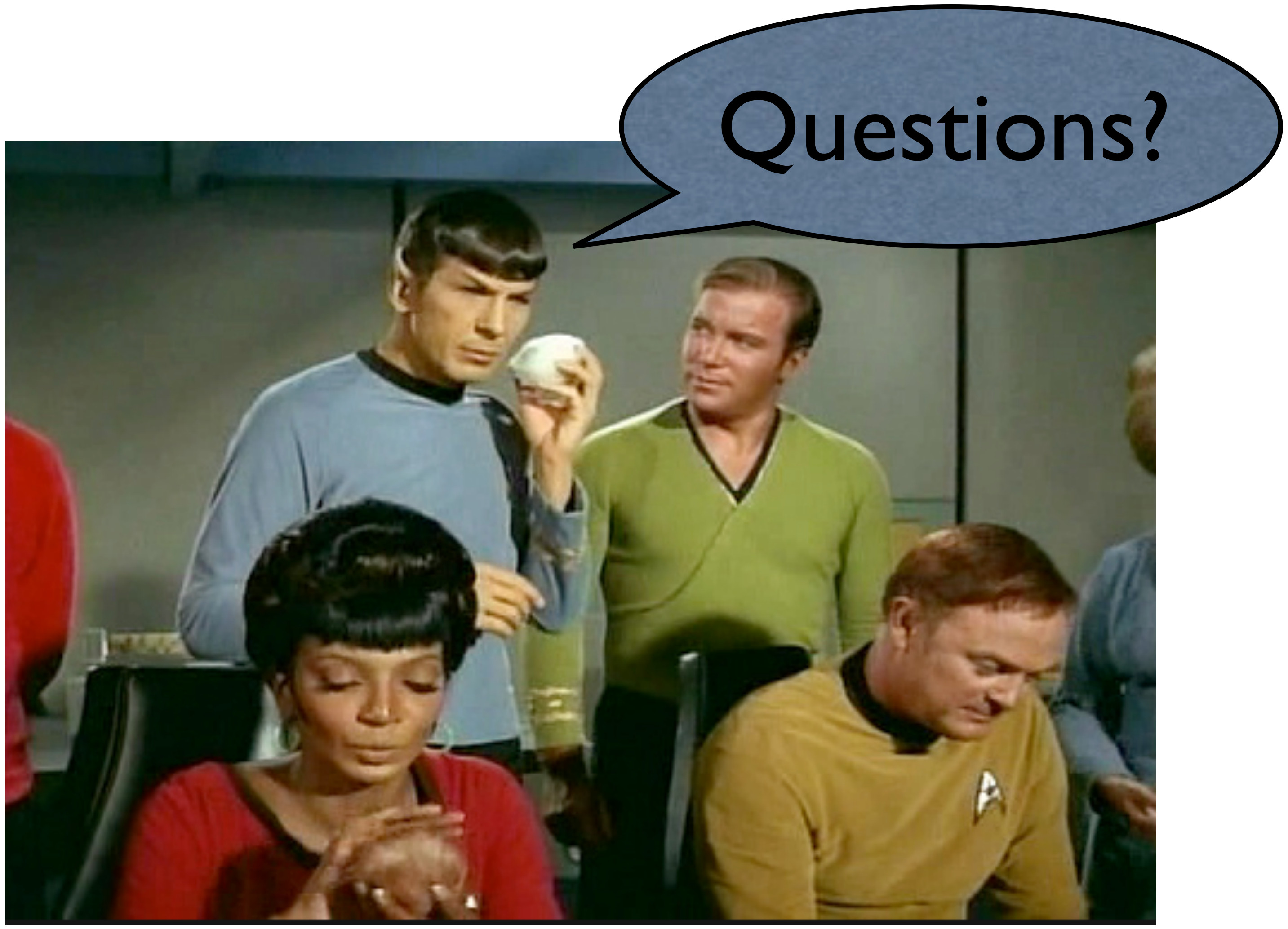
Solution 2 (Sort-Iterate)

- Sort the input data by group-key (if unsorted)
  - Does this remind you of anything?
- Scan the sorted data in order
  - When we encounter a new group-key:
    - Finalize/Output the last group
    - Start a new group

Grouping is analogous to duplicate elimination. Deduplication (i.e., DISTINCT) is effectively just a group-by query with no aggregate value.

Summary

- Query plans are trees of relational operators.
- Relational operators (or subtrees) of can be implemented using different algorithms.
- Different algorithms have different costs/requirements.
  - e.g., data in sorted order
- Cost-based optimization used to select which algorithm to use.


Questions?

External Algorithms

- How do we process data that doesn't fit in memory?
  - One-pass algorithms
  - Split up the data into smaller chunks.
- Why not use Virtual Memory?
- These algorithms can be adapted to other levels of the memory hierarchy.
  - e.g., Cache-conscious algorithms

Virtual Memory attempts to automatically infer access patterns and adapt itself dynamically to the user's needs. We have this information available already!

External Algorithms

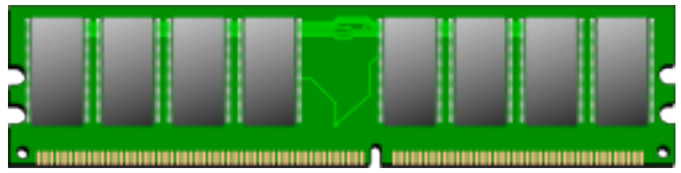
- Streaming: Do everything in a single scan.
    - Can be combined with data partitioning
- Data Partitioning
    - Arbitrary Binning
    - Partitioning by Sorting
    - Partitioning by Hashing

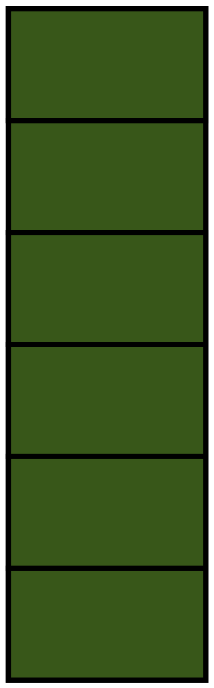
External Algorithms

- Streaming: Do everything in a single scan.
  - Can be combined with data partitioning
- Data Partitioning
  - Arbitrary Binning
  - Partitioning by Sorting
  - Partitioning by Hashing

bin = f(value)

Partitioning





Allocate pages for each bin

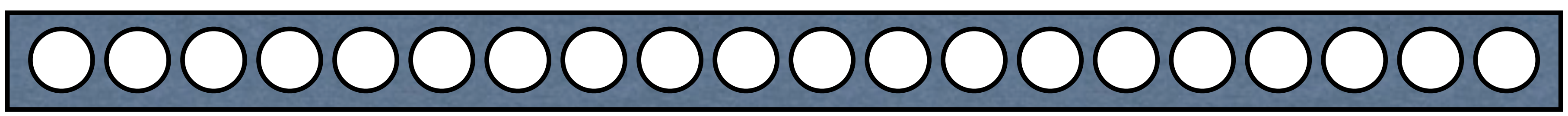
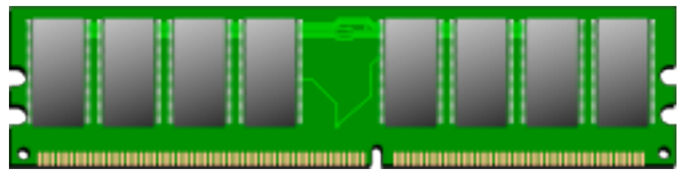
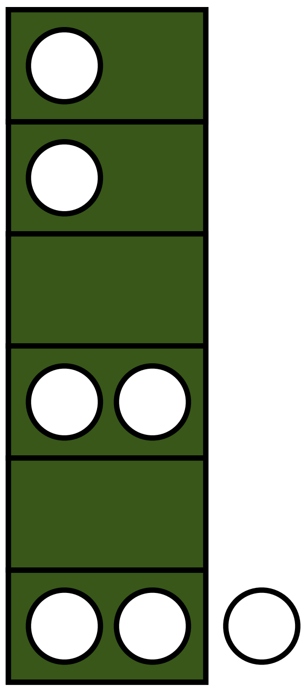


image credit: openclipart.org

Partitioning





Allocate pages for each bin

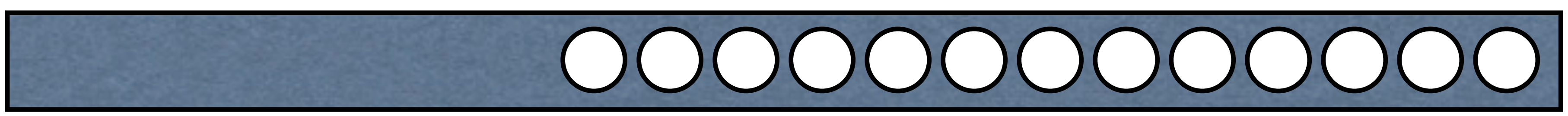
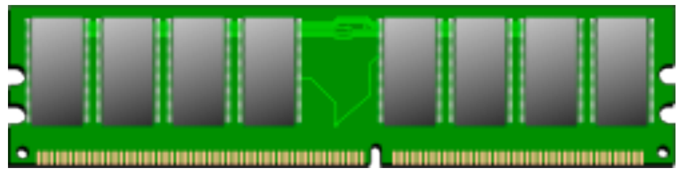
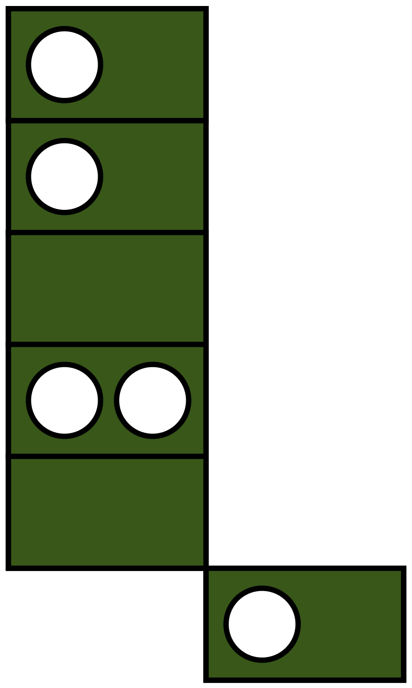


image credit: openclipart.org

Partitioning





Allocate pages for each bin

Only the last page needs to be in memory

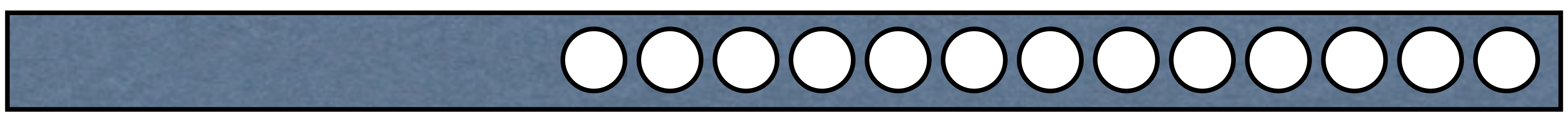
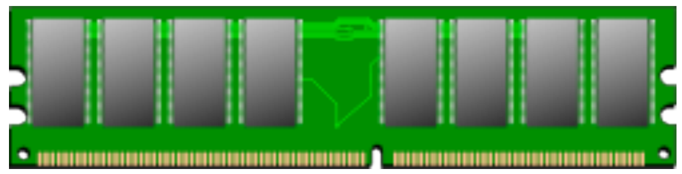
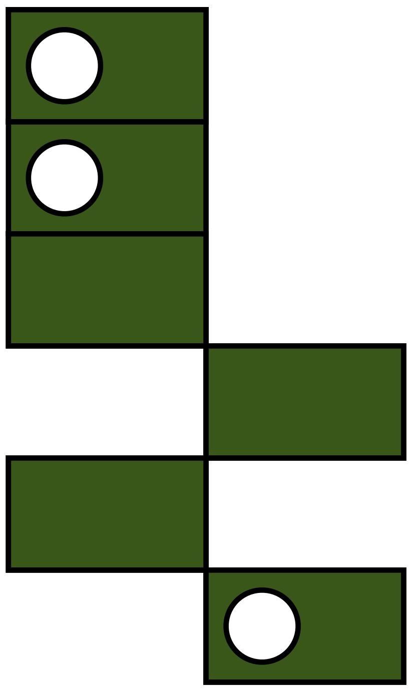


image credit: openclipart.org

Partitioning





Allocate pages for each bin

Only the last page needs to be in memory

Flush pages to disk as soon as they are full (buffering)

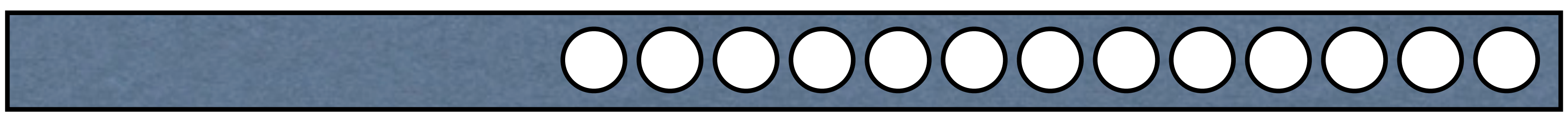
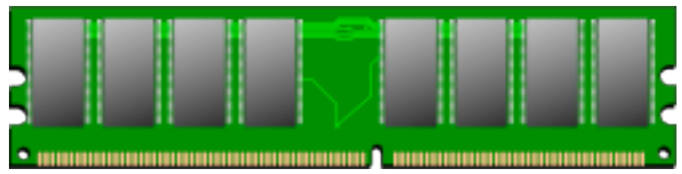
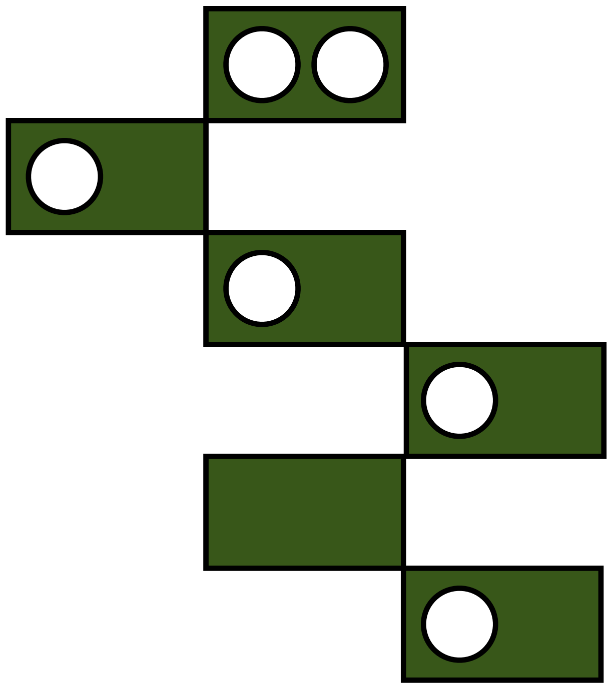


image credit: openclipart.org

Partitioning





Allocate pages for each bin

Only the last page needs to be in memory

Flush pages to disk as soon as they are full (buffering)

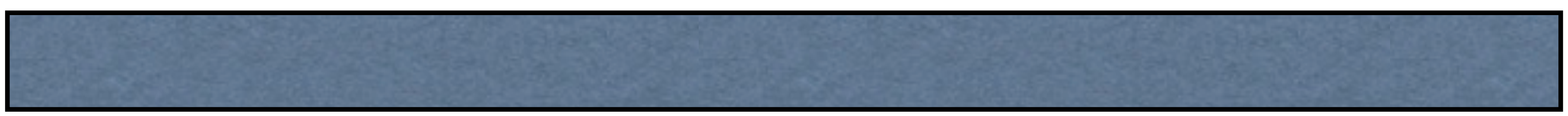
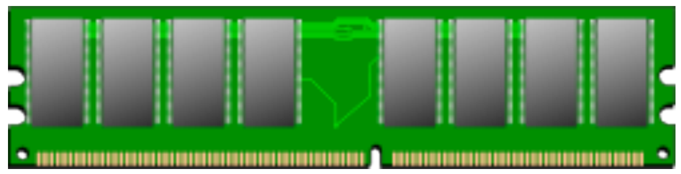


image credit: openclipart.org

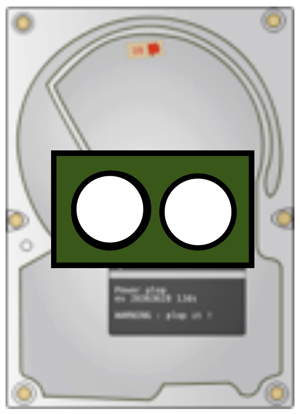
Partitioning



Allocate pages for each bin

Only the last page needs to be in memory

Flush pages to disk as soon as they are full (buffering)



Finally flush all remaining pages to disk

Buffering (for scans)

- Input Buffers (for scans)
  - Keep multiple pages loaded in memory.
  - When a page is fully scanned, start to read another page.
- Output Buffers (for streaming output)
  - When a page is full, start to write it to disk.

Streaming

- Read each value (or block) exactly once.
  - Does the intermediate data use a fixed amount of space? (examples?)
  - Are there properties of the data stream that can be exploited? (examples?)
  - Can outputs be generated inline as inputs arrive? (examples?)

Intermediate data example: Non-holistic Aggregate functions
Properties: Sorted order in a sort/merge join
Inline generation: Selection, Projection, Half of the Hybrid-Hash Join

Example: Sort

- Why sorting?
    - A classic problem in computer science
    - Data in sorted order required by several relational query algorithms.
- **Problem**: Sort 10 TB of data with 10 GB RAM

2-Way Sort

Pass 1

image credit: openclipart.org

2-Way Sort

Pass 1

Load a Page

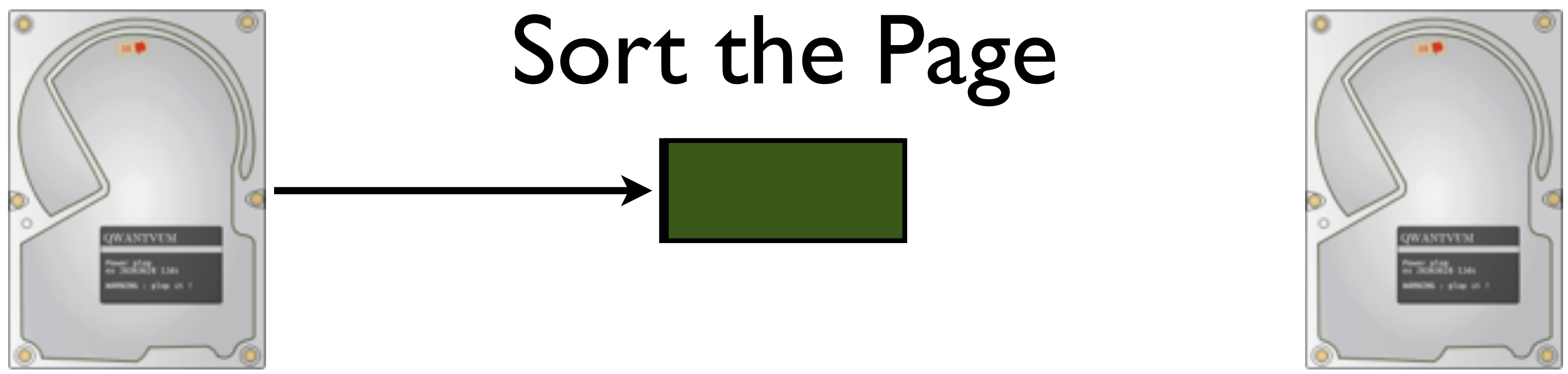
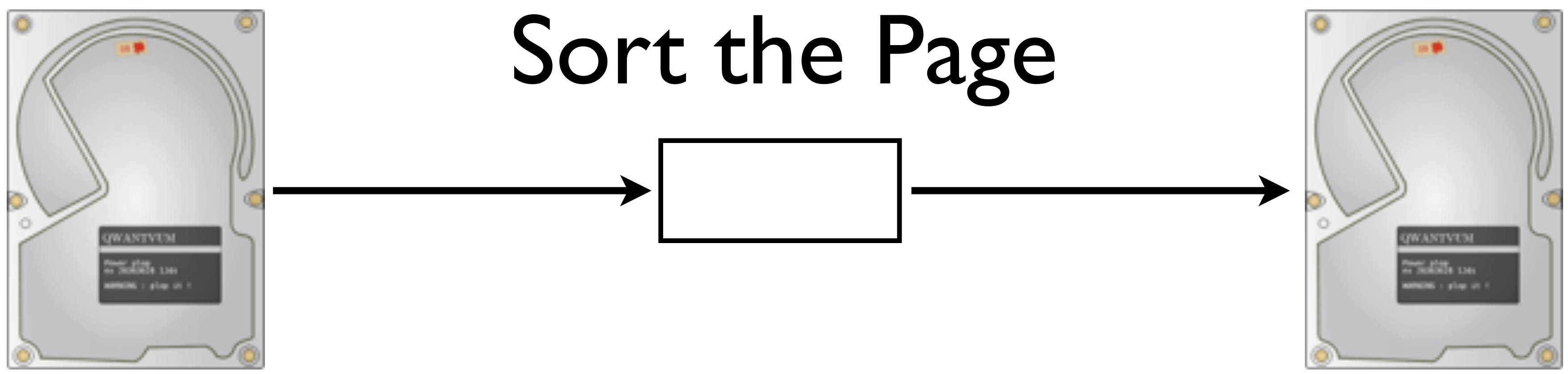


image credit: openclipart.org

2-Way Sort

Pass 1

Load a Page



Flush the Page

image credit: openclipart.org

2-Way Sort

Pass 2 and beyond

image credit: openclipart.org

Log_2 N steps
How much memory is required? (3 pages + additional buffer space)

2-Way Sort

Pass 2 and beyond

Read from 2 (sorted) buffers of size K

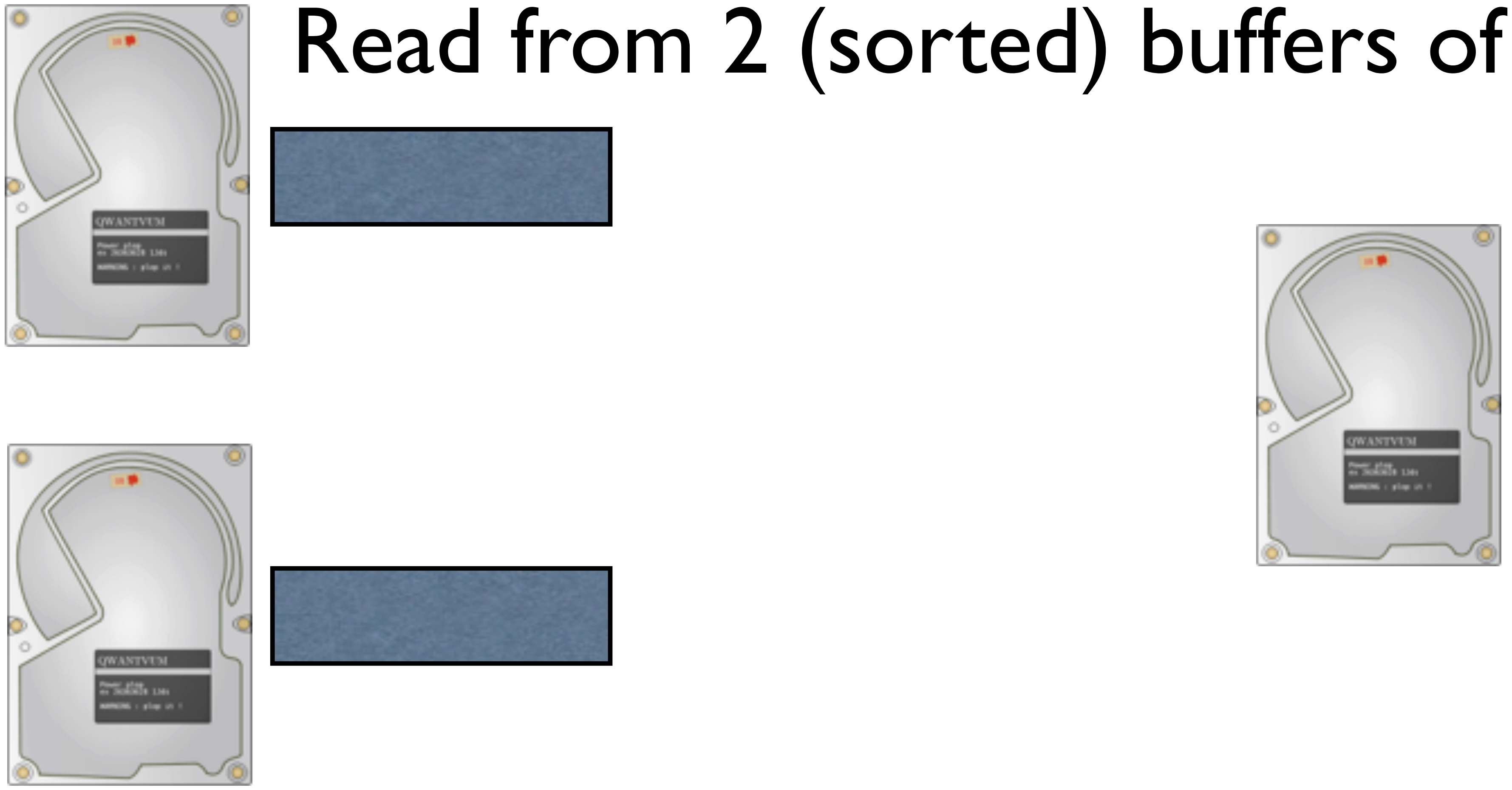


image credit: openclipart.org

Wednesday, January 30, 13

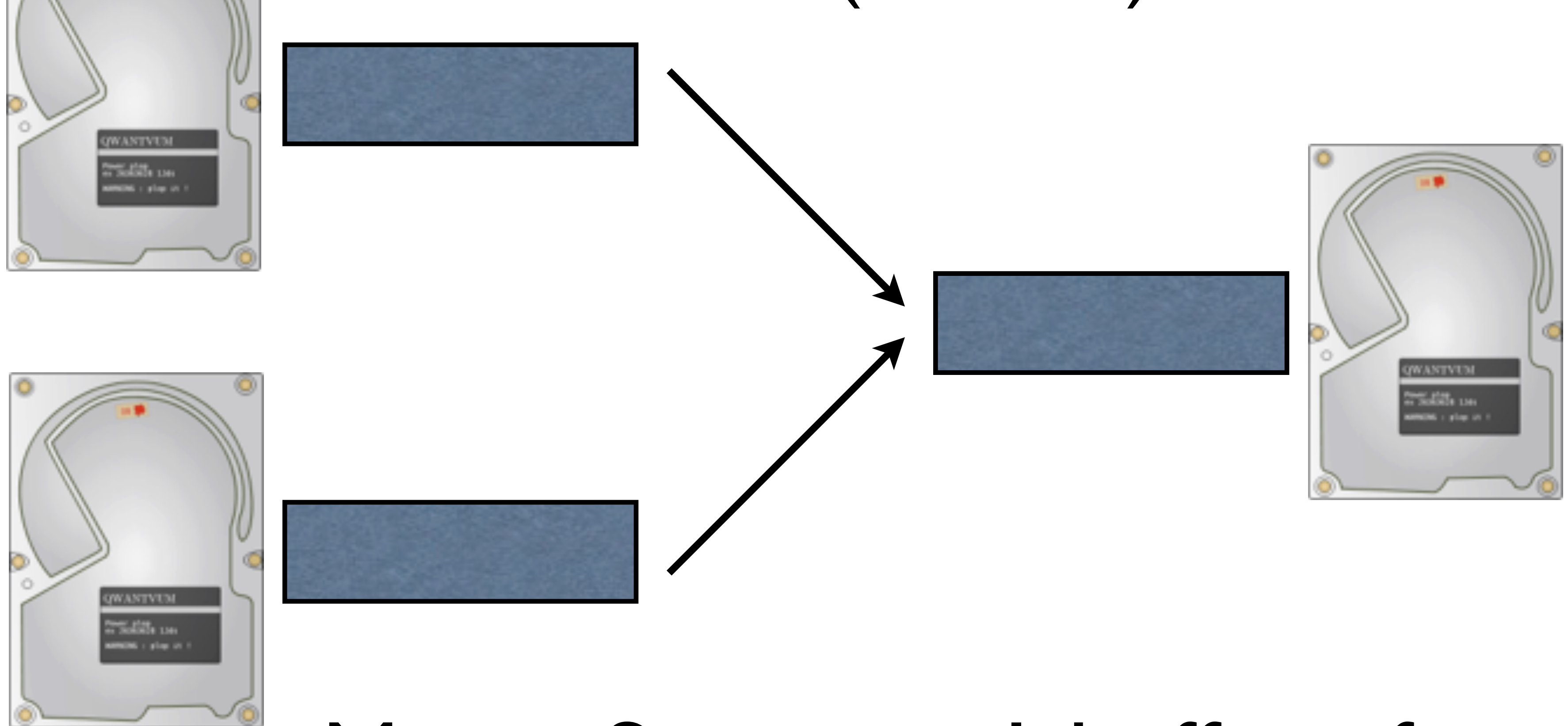
Log_2 N steps

How much memory is required? (3 pages + additional buffer space)

2-Way Sort

Pass 2 and beyond

Read from 2 (sorted) buffers of size K



Merge Sort into 1 buffer of size 2K

image credit: openclipart.org

Wednesday, January 30, 13

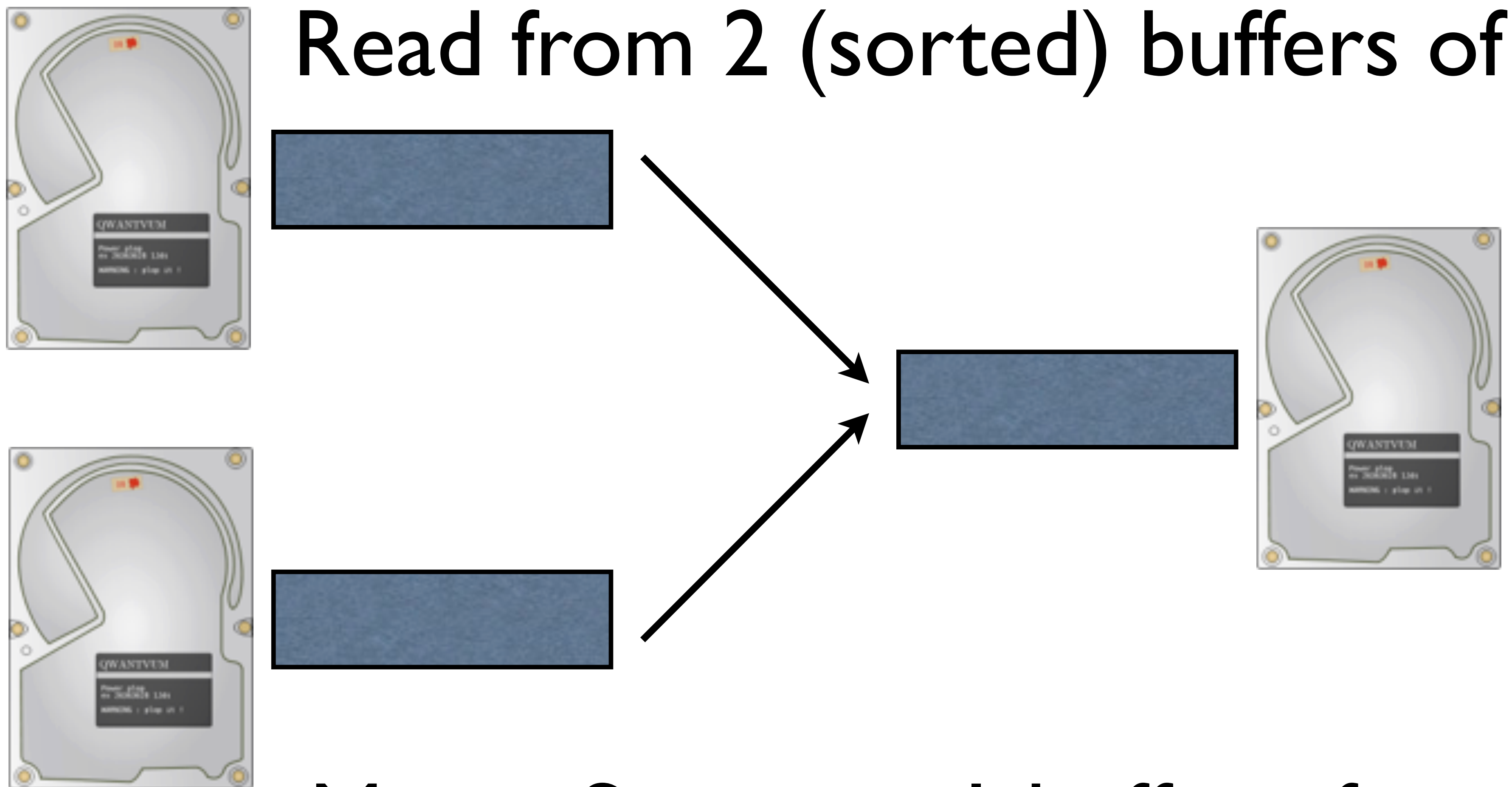
Log_2 N steps

How much memory is required? (3 pages + additional buffer space)

2-Way Sort

Pass 2 and beyond

Read from 2 (sorted) buffers of size K



Merge Sort into 1 buffer of size 2K

Repeat (how many times?)

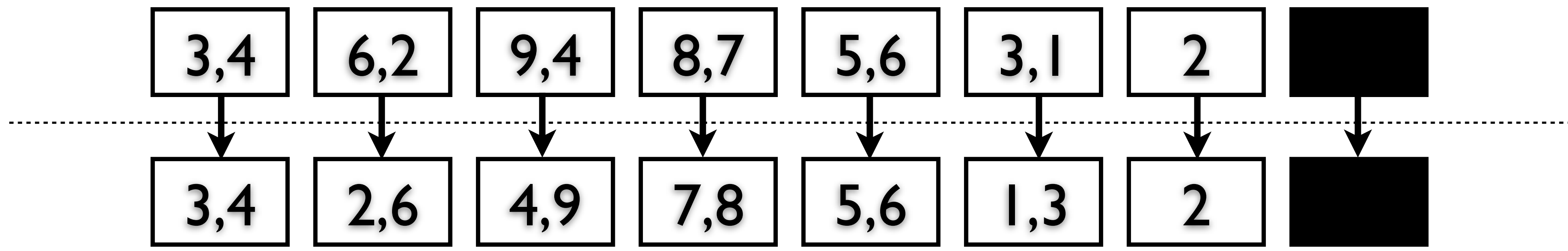
image credit: openclipart.org

Log_2 N steps
How much memory is required? (3 pages + additional buffer space)

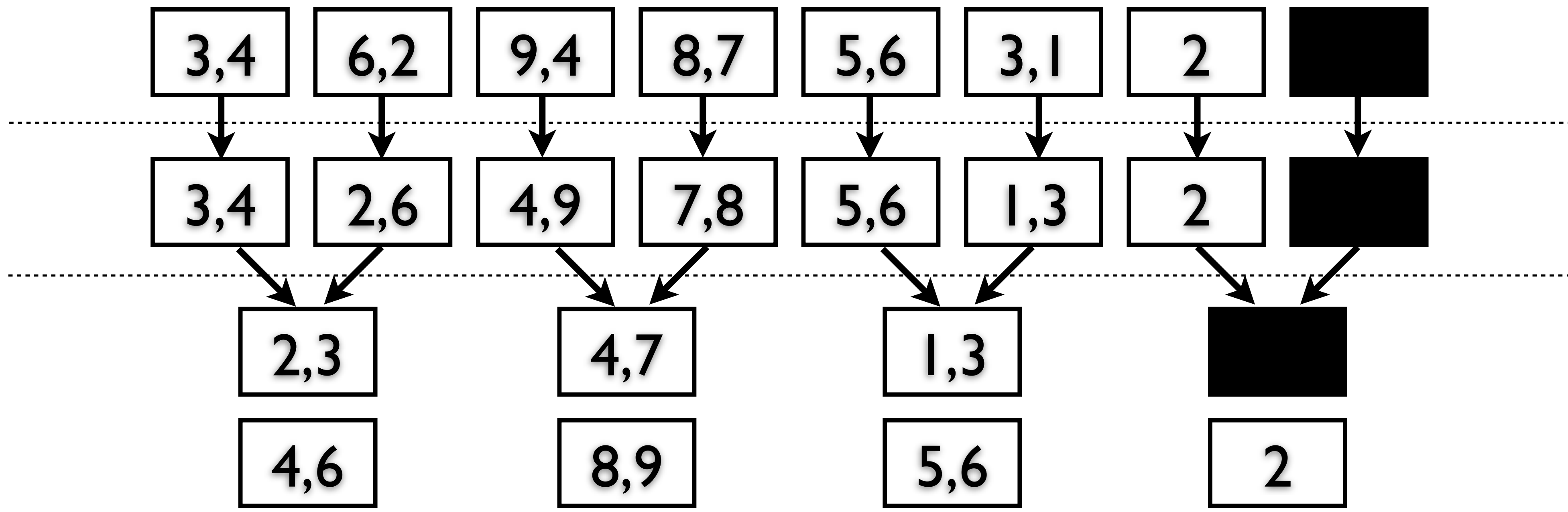
2-Way Sort

3,4	6,2	9,4	8,7	5,6	3,1	2	

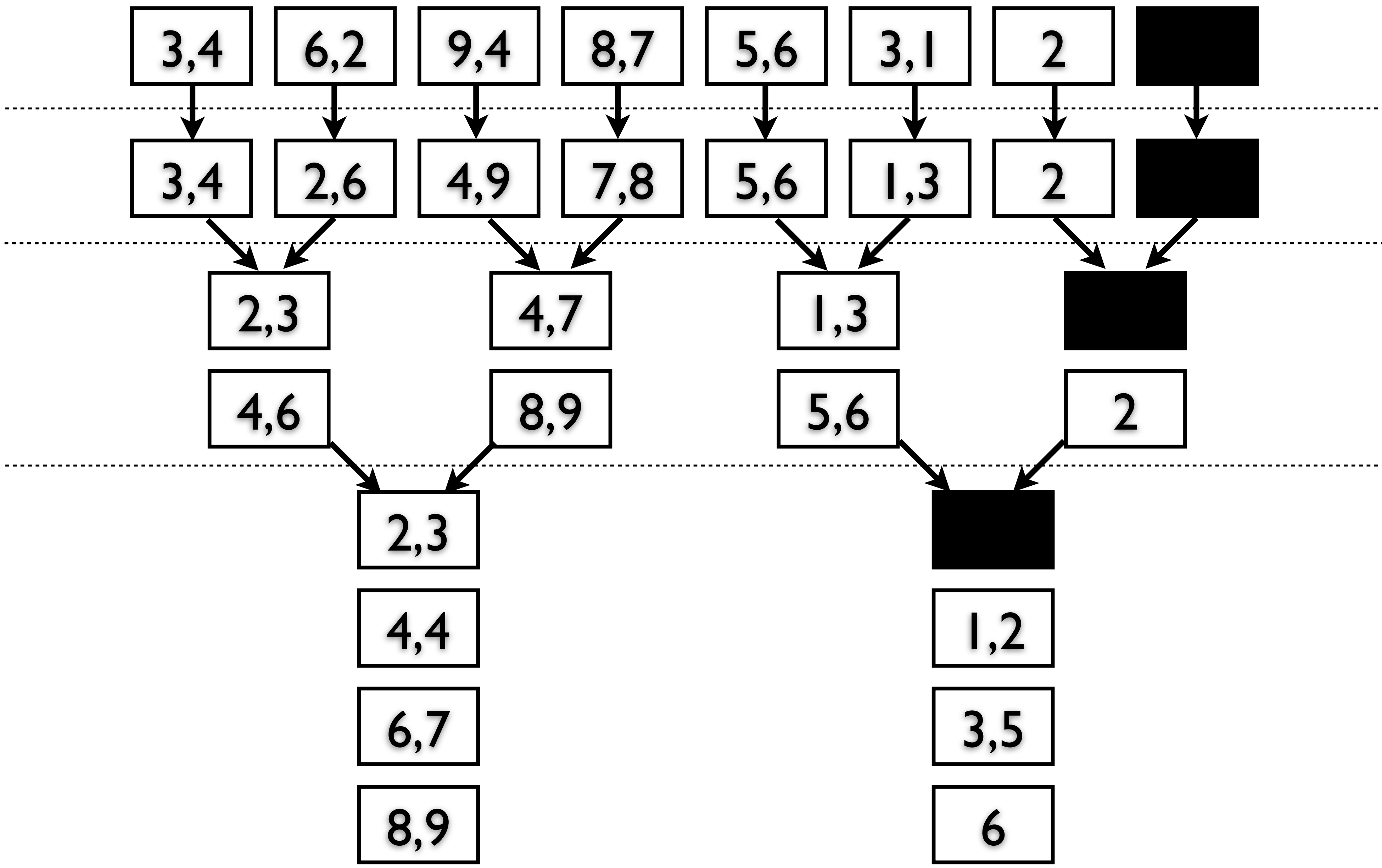
2-Way Sort



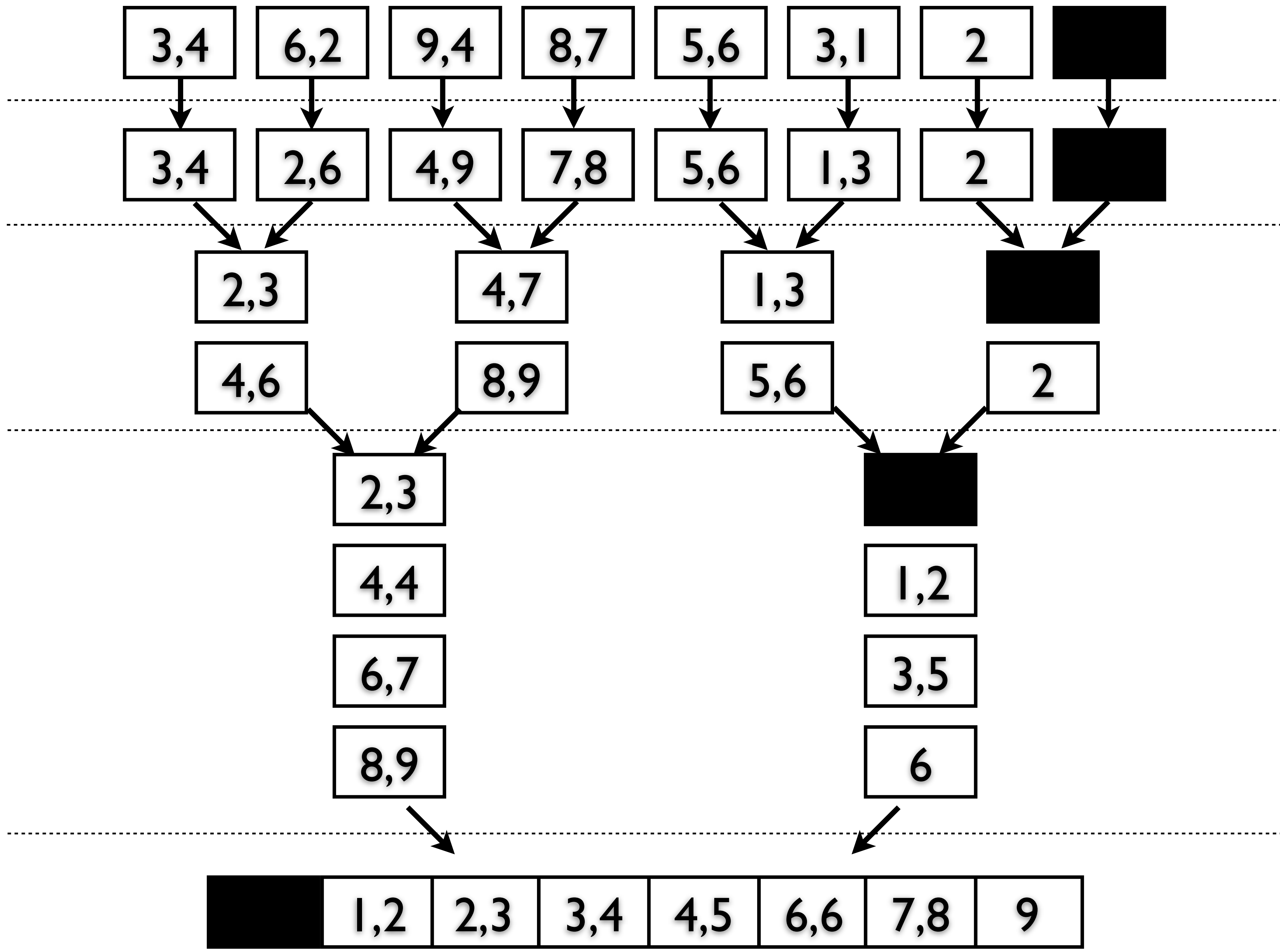
2-Way Sort

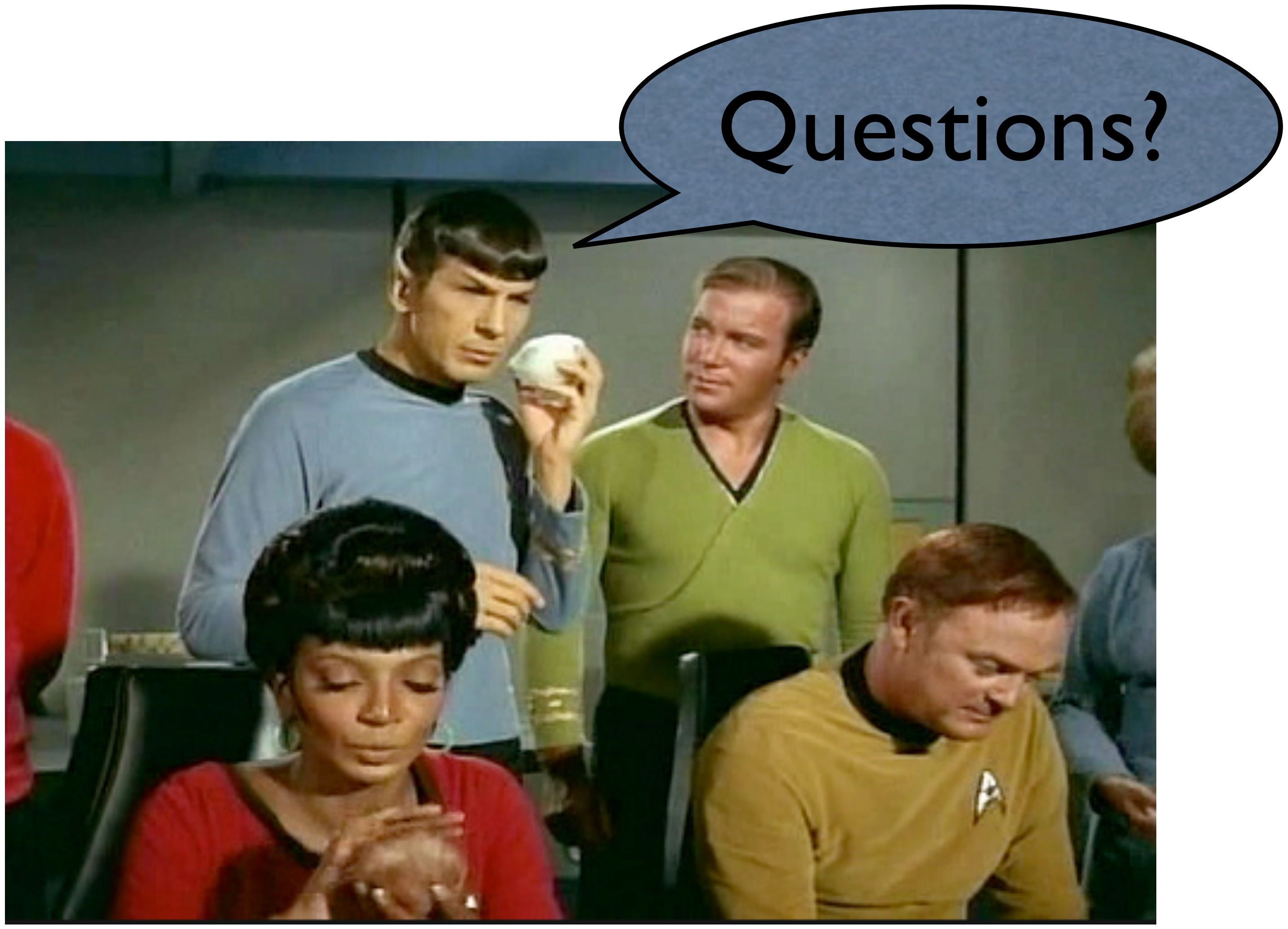


2-Way Sort


3,4
6,2
9,4
8,7
5,6
3,1
2
3,4
2,6
4,9
7,8
5,6
1,3
2
2,3
4,6
4,7
8,9
1,3
5,6
2
2,3
4,4
6,7
8,9
1,2
3,5
6

2-Way Sort


3,4
6,2
9,4
8,7
5,6
3,1
2
3,4
2,6
4,9
7,8
5,6
1,3
2
2,3
4,7
1,3
4,6
8,9
5,6
2
2,3
4,4
6,7
8,9
1,2
3,5
6
1,2
2,3
3,4
4,5
6,6
7,8
9


Questions?

Generalized External Sort

How can we use N buffer frames?

For Pass 1?

For Pass 2 onwards?

Generalized External Sort

How can we use N buffer frames?

For Pass 1? Sort Bigger Initial Buffers

For Pass 2 onwards?

Generalized External Sort

How can we use N buffer frames?

For Pass 1? Sort Bigger Initial Buffers

For Pass 2 onwards? Merge-sort Multiple Streams

How many passes do we make over the full data?

Generalized External Sort

How can we use N buffer frames?

For Pass 1? Sort Bigger Initial Buffers

For Pass 2 onwards? Merge-sort Multiple Streams

How many passes do we make over the full data?

For data of size N, a K-way sort requires $\lceil log_K(N) \rceil + 1$ passes

How many IOs do we use?

Generalized External Sort

How can we use N buffer frames?

For Pass 1? Sort Bigger Initial Buffers

For Pass 2 onwards? Merge-sort Multiple Streams

How many passes do we make over the full data?

For data of size N, a K-way sort requires $\lceil log_K(N) \rceil + 1$ passes

How many IOs do we use?

$$2 \cdot \#\text{pages} \cdot \#\text{passes}$$

Pass 1 is memory-limited

If we have N pages of memory,
can we create more than N pages of sorted data?

Replacement Sort

- General idea: Create "runs" of sorted data
- Keep a very large "working set" of data.
- Keep appending data in ascending order to an output buffer.
- As you flush sorted data to the output, keep loading new tuples into the working set.
    - If you get new tuples useful for the current buffer, great!
    - Otherwise, they'll go into the next run
- When you run out of valid tuples to append, start a new run!

Replacement Sort

Step 0: k is the last value that was appended to the output buffer

Input Buffer

Working Set

2
8
10
...

k=5

5 3

Output Buffer

Replacement Sort

Input Buffer

2
8
10
...

Working Set

Step 1: Find the lowest value in the working set greater than k

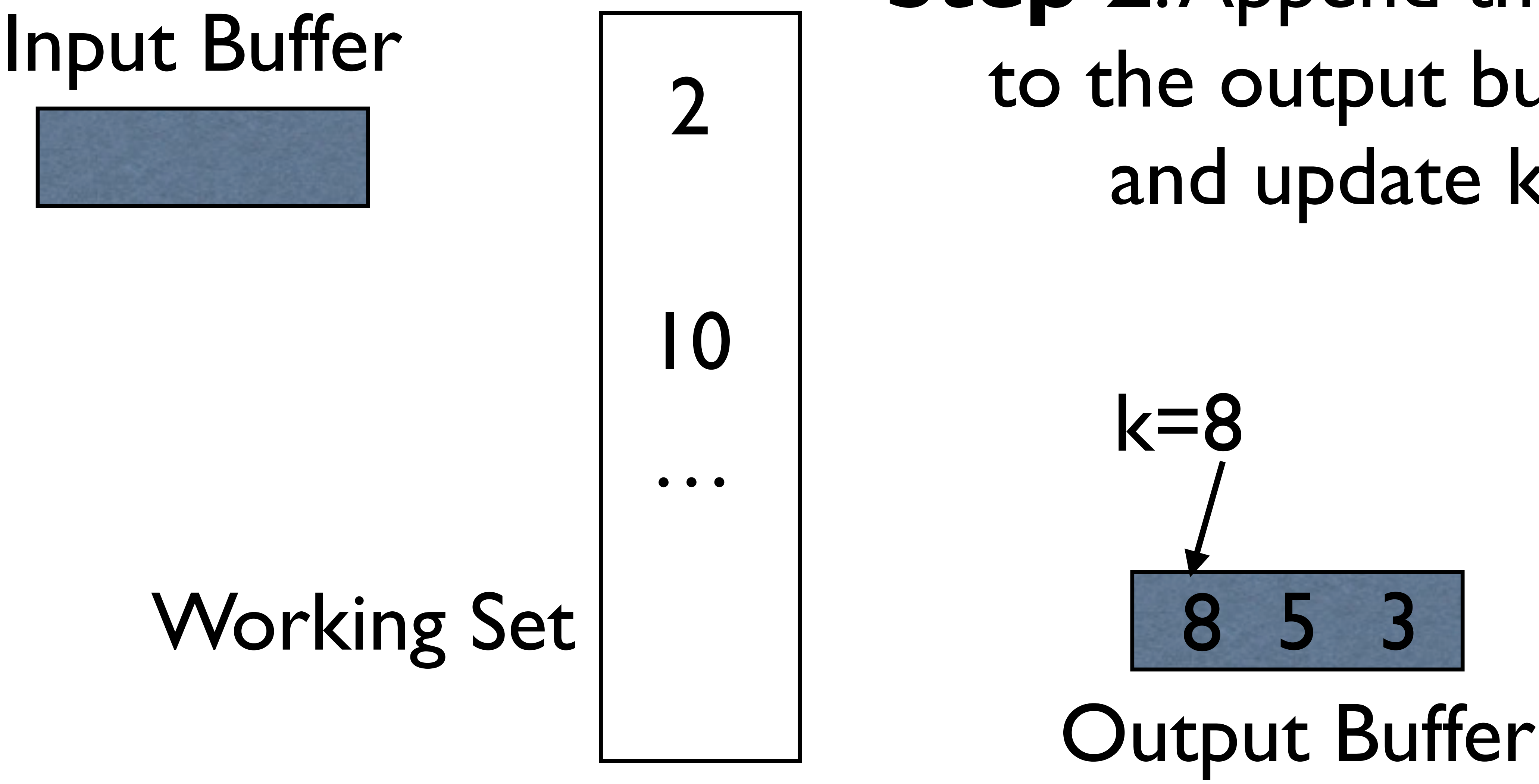
k=5

5 3

Output Buffer

Replacement Sort

Step 2: Append the value to the output buffer and update k



Replacement Sort

Step 3: Insert a tuple from the input buffer and re-sort the working set

Input Buffer

Working Set

2
10
12
…

k=8

8 5 3

Output Buffer

Replacement Sort

Repeat until k is bigger than all values in the working set

Input Buffer

2
10
12
...

Working Set

Finish the "run" and start a new one

k=8

8 5 3

Output Buffer

Replacement Sort

$E[k] = avg(k)$

On average, half of the tuples you read in will be useful for the current stream.

If you have N pages of memory, how many pages of sorted data will you make?

You'll generate 2N pages of sorted data.
Think of Xeno's paradox. Every time you generate K pages of sorted output, you get (on average) K/2 more pages of data that can still be appended to the sorted output.

Summary

- Dealing with the memory hierarchy requires understanding…
  - … how to stream data effectively (nonblocking ops)
  - … how to organize/partition data effectively
- These ideas are applicable to other layers too!
  - Network is another layer of the memory hierarchy.
  - Cache is another layer of the memory hierarchy.

Questions?