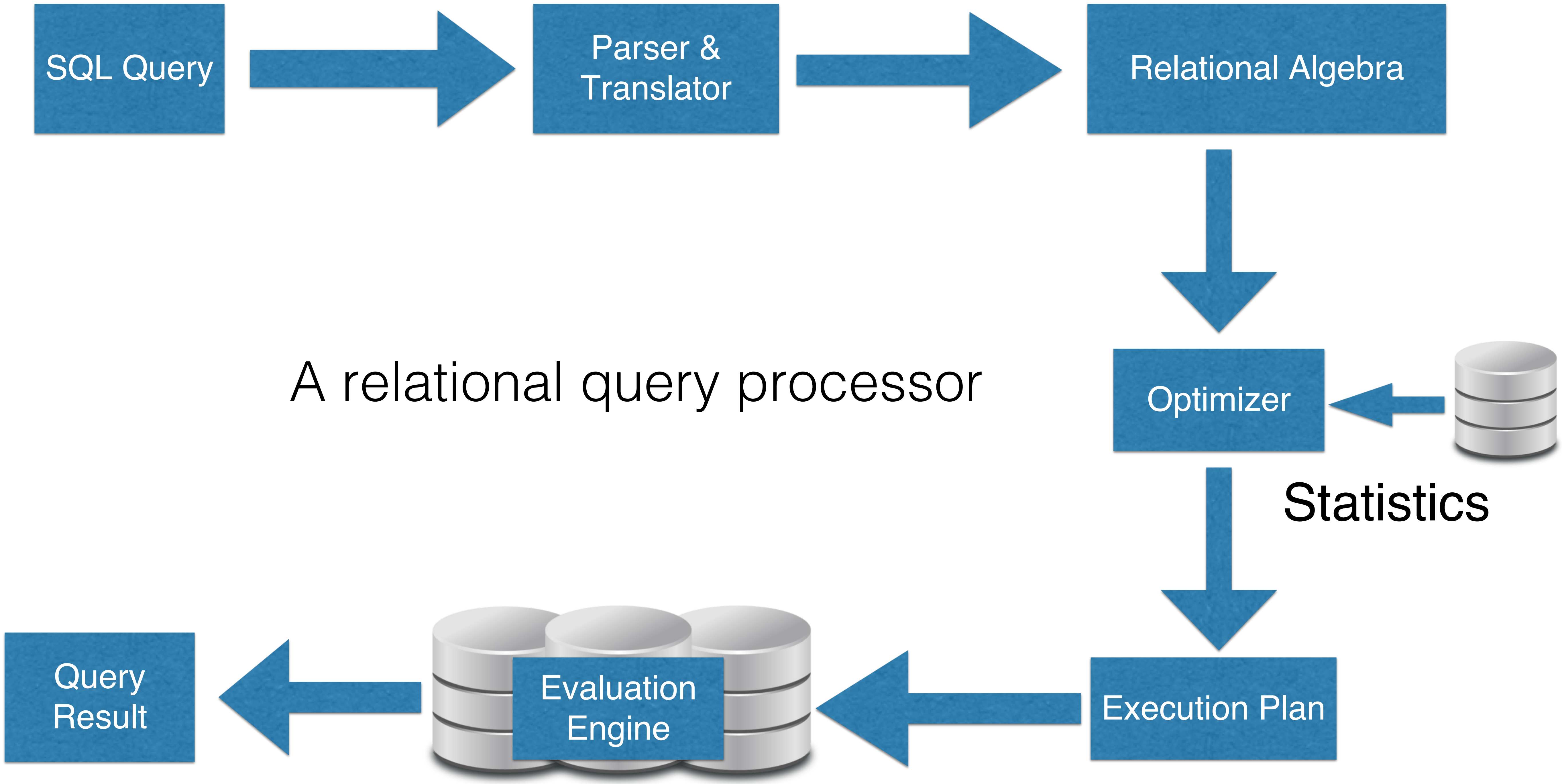
CSE 4/562
Database Systems

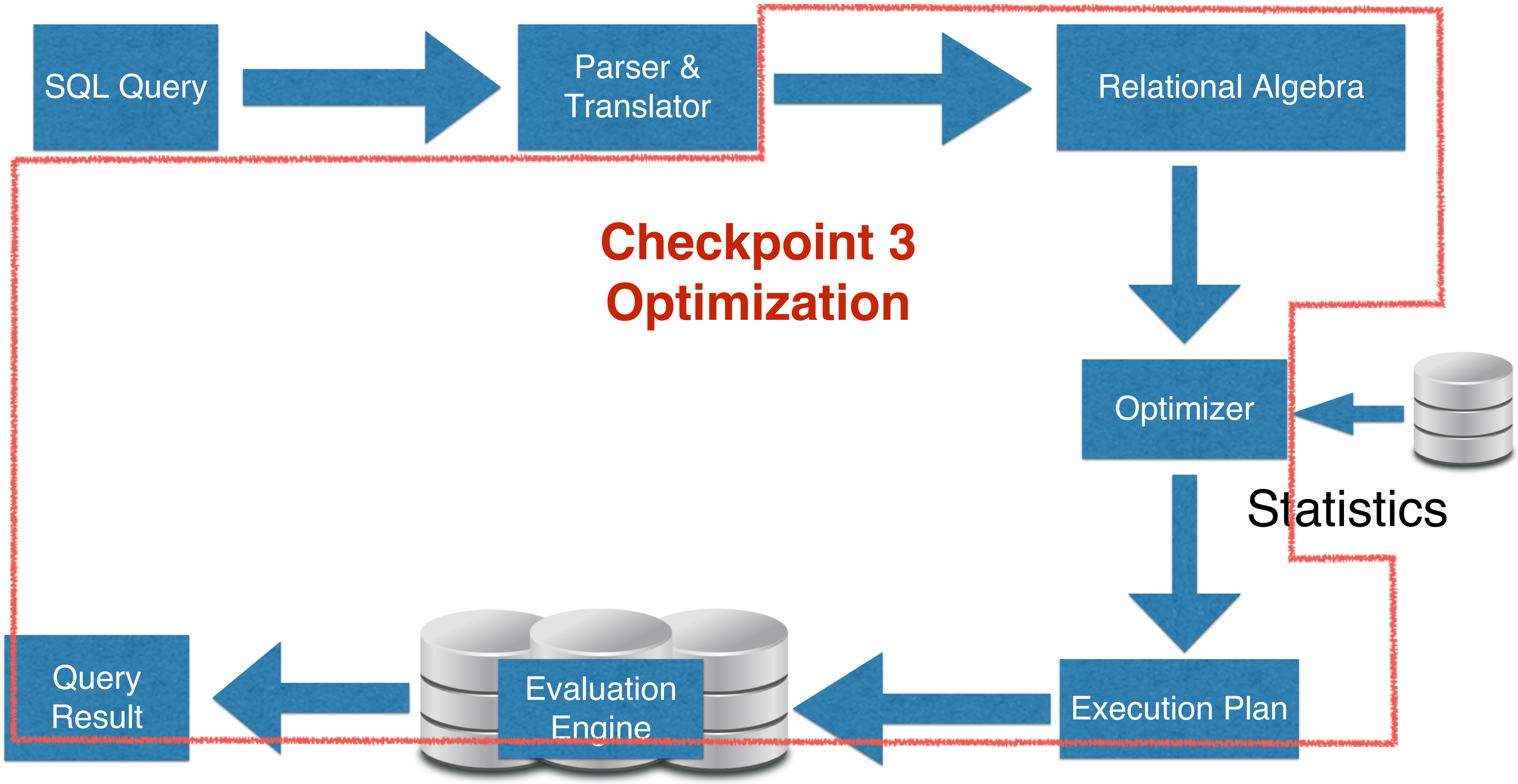
Practicum Component

Project Outline



A relational query processor

Project Outline



Checkpoint 3

- How do make your system faster?
  - Programming efficiency?
  - Choosing a strategy?
  - More efficient operators?
- How can you deal with aggregation?

Checkpoint 3

- 3 Main Components
    - New functions (Distinct, Group-By, and functions)
    - Optimization (Projection/Selection Pushdown)
    - Join Algorithms

New Functions

- GROUP BY – HAVING
- COUNT()
- AVG()
- SUM()
- MIN() – MAX()
- DISTINCT

Aggregations

- Hash aggregation algorithm
  - Requires more memory
- Stream aggregation algorithm
  - Requires data to be sorted first

Hash Aggregation

```
for each input row
    begin
        calculate hash value on group by column(s)
        check for a matching row in the hash table
        if we find a match
            update the matching row with the input row
        else
            insert a new row into the hash table
    end
output all rows in the hash table
```

Stream Aggregation

```
For each input row
begin
    if the input row does not match the current columns
        begin
            output the aggregate results
            clear the current aggregate results
            set the current group-by columns to the input row
        end
    update the aggregate results with the input row
end
```

Optimization

- You already implemented Selection pushdown
- You need to implement Projection pushdown

Selection Pushdown

- Helps you filter out unnecessary tuples early on
- Provides both memory and CPU time profits
    - Saves you memory because join and cross product algorithms use memory based on their input size
    - Saves you CPU time because other operators do not need to deal with unnecessary tuples

Projection Pushdown

- Helps you filter out unnecessary attributes early on
- Provides both memory and CPU time profits
    - Saves memory because you don’t carry unnecessary data between operators
    - Saves CPU time because you don’t copy unnecessary data when you modify the tuple schema and need to copy data to the new tuple

Join Algorithms

- Nested-Loop-Join and Block-Nested-Loop-Join are slow…
- Try other join algorithms

Classic Hash Join

Works when the smaller relation *R* fits in memory.

1. Build a in-memory hash table for the smaller relation;

2. For each record in the larger relation, probe the hash table.

If the smaller relation does not fit in memory, partition into smaller buckets!

Simple Hash Join

```
1. for each logical bucket $j$
2.     for each record $r$ in $R$
3.         if $r$ is in bucket $j$ then
4.             insert $r$ into the hash table;
5.     for each record $s$ in $S$
6.         if $s$ is in bucket $j$ then
7.             probe the hash table;
```

- Classic hash join is a special case, with one bucket;
- Optimization: write the tuples not in bucket *j* to disk;
- Works good when memory is large (nearly as large as $|R|$).

GRACE Hash Join

```
1. partition $R$ into $n$ buckets so that each bucket fits
   in memory;
2. partition $S$ into $n$ buckets;
3. for each bucket $j$ do
4.     for each record $r$ in $Rj$ do
5.         insert into a hash table;
6.     for each record $s$ in $Sj$ do
7.         probe the hash table.
```

- Works good when memory is small.

Hybrid Hash Join

- Hybrid of simple hash join and GRACE;
- When partitioning *R*, keep the records of the first bucket in memory as a hash table;
  - Typically this means that the first bucket uses more pages in memory (all other partitions are 1 page each)
- When partitioning *S*, for records of the first bucket, probe the hash table directly;
- Saving: no need to write R_1 and S_1 to disk or read back to memory.

- Works good for large and small memory.

Handle Partition Overflow

- Case 1, overflow on disk: an *R* partition is larger than memory size (note: don't care about the size of *S* partitions).
  - Solution (a) small partitions first and combine before join;
  - Solution (b) recursive partition.
- Case 2, overflow in memory: the in-memory hash table of *R* becomes too large.
  - Solution: revise the partitioning scheme and keep a smaller partition in memory.

Questions?