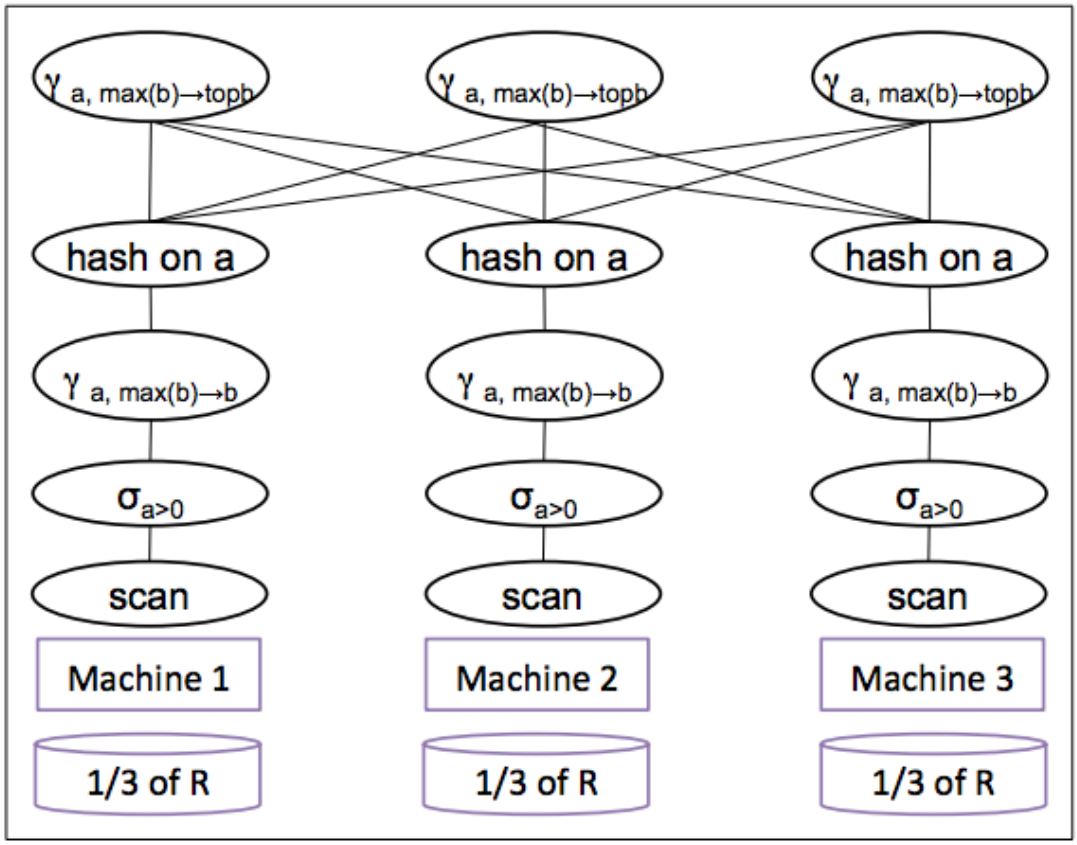
Section 10 – Big Data

CSE 344

Parallel Data Processing

Given the following query, show a (parallel) relational algebra plan for this query. There are 3 machines and the data is block-partitioned evenly across each machine.

```
SELECT a, max(b) as topb
FROM R
WHERE a > 0
GROUP BY a;
```



MapReduce

Suppose you have two relations: R(a,b) and S(b,c)
MapReduce needs (key, value) pairs as input, so we parse the above relations into such pairs.
R.a will be the key for each tuple in R. Imagine it as a map: { R.a → (R.a, R.b, tag="R") }
S.b will be the key for each tuple in S. Imagine it as a map: { S.b → (S.b, S.c, tag="S") }
For each relational plan below, write pseudocode for the Map and Reduce functions.

Select tuples from *R*: $\sigma_{a<10}R$

In this simple example, all the work is done in the map function when we read R, where we copy the input to the intermediate data, but only for tuples that meet the selection condition:

```
map(inkey, invalue):
  if inkey < 10
    emit_intermediate(inkey, invalue)
```

Reduce then simply outputs all the values it is given:

```
reduce(hkey, hvalues[]):
  for each t in hvalues:
    emit(t)
```

Eliminate duplicates from R: $\delta(R)$

For this problem we will use a simple trick, described in your textbook — we'll use the fact that duplicate elimination in the bag relational algebra is equivalent to grouping on all attributes of the relation. MapReduce does grouping for us, so all we need is to make the entire tuple the intermediate key.

```
map(inkey, invalue):
  emit_intermediate(invalue, 'abc')  // won't use intermediate value
```

Once we do that we just output the intermediate key as the final value:

```
reduce(hkey, hvalues[]):
  emit(hkey)
```

Natural join of *R* and *S*: $R \mid\!><\!\mid_{R.b=S.b} S$

The map function outputs the same value as its input, but changes the key to always be the join attribute

```
map(inkey, invalue):
 emit_intermediate(invalue.B, invalue)
```

After the MapReduce system groups together the intermediate data by the intermediate key, we use the reduce function to do a nested loop join over each group. Because all the values from each group have the same join attribute, we don't check the join attribute in the nested loop. We do need to check which relation each tuple comes from, so that (for example) we don't join a tuple from *R* with itself, or with another *R* tuple.

```
reduce(hkey, hvalues[]):
  for each r in hvalues:
    for each s in hvalues:
      if r.tag = 'R' and s.tag = 'S':
        emit(r.a, r.b, s.c)
```

Note that this is actually a very inefficient way to compute a join because we are essentially doing a Cartesian product!