CS3223 Tutorial 5

Gary Lim

Welcome Back!

- **Midterms** is on the **4th of March** (Friday) **10 to 11:30am**
  - Venue: MPSH2
  - See course webpage for Seat Allocation
- If tested covid positive
  - Online arrangement: Email prof to attempt the **same paper** online at the **same time** on luminus
  - Make-up test: If unable to sit for the paper (online or onsite), please email Prof to have a separate makeup test (a different paper) arranged for you

Chapter Review

- Access Paths
- Duplicate Removal
  - Sort-based
  - Hash-based

Access Paths

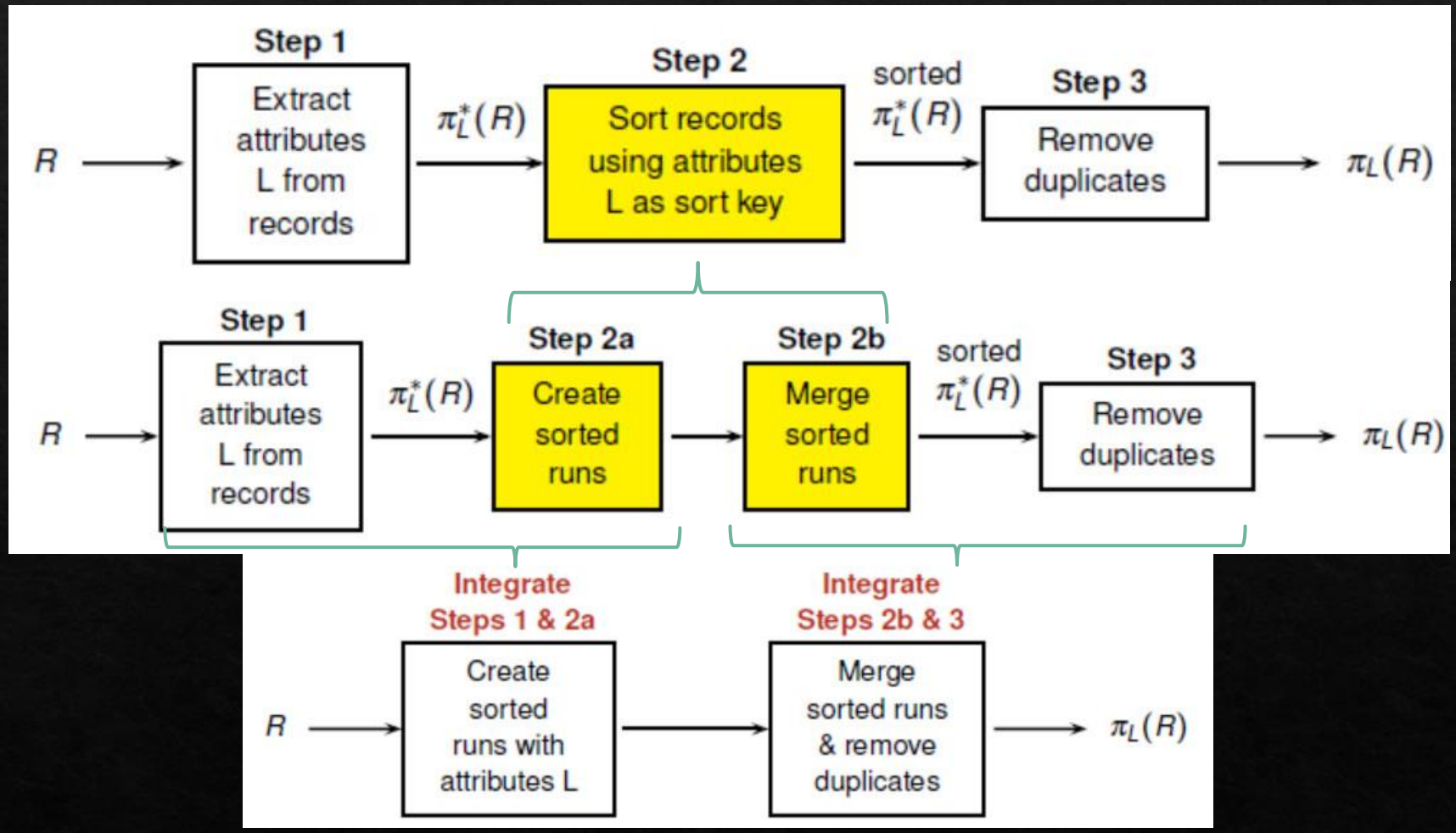
- Table Scan – Scan all pages
  - Projection
  - Selection (no index)
- Index-Only Scan
  - Scan the index and its information (e.g. format 1)
- Index Scan
  - Scan the index and retrieve data pages

Note that cost of reading and writing are usually different as the result would usually be different size (after a projection, selection or a join)

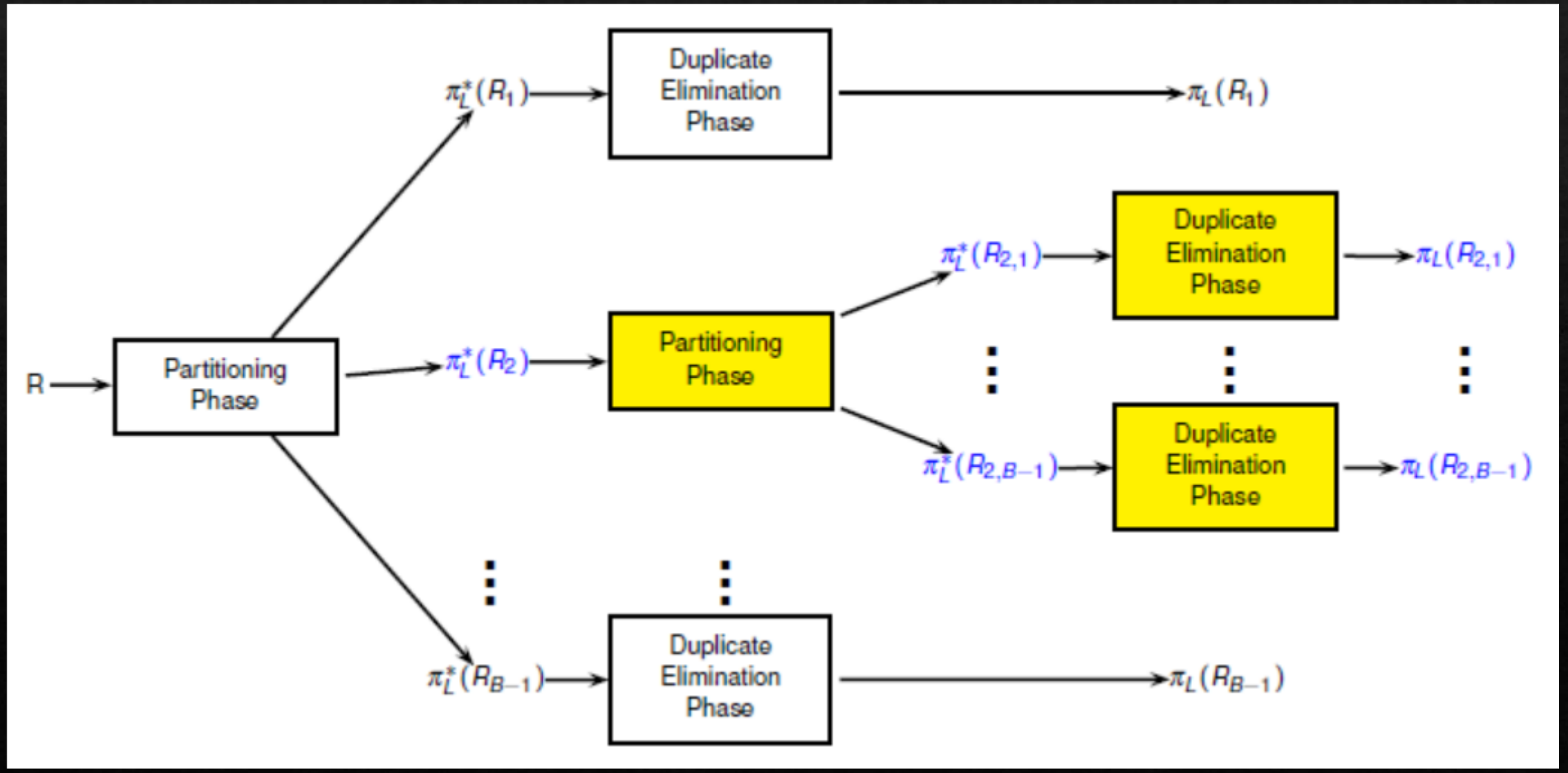
Projection (with duplicate removal)

- Sort-based approach
- Hash-based approach

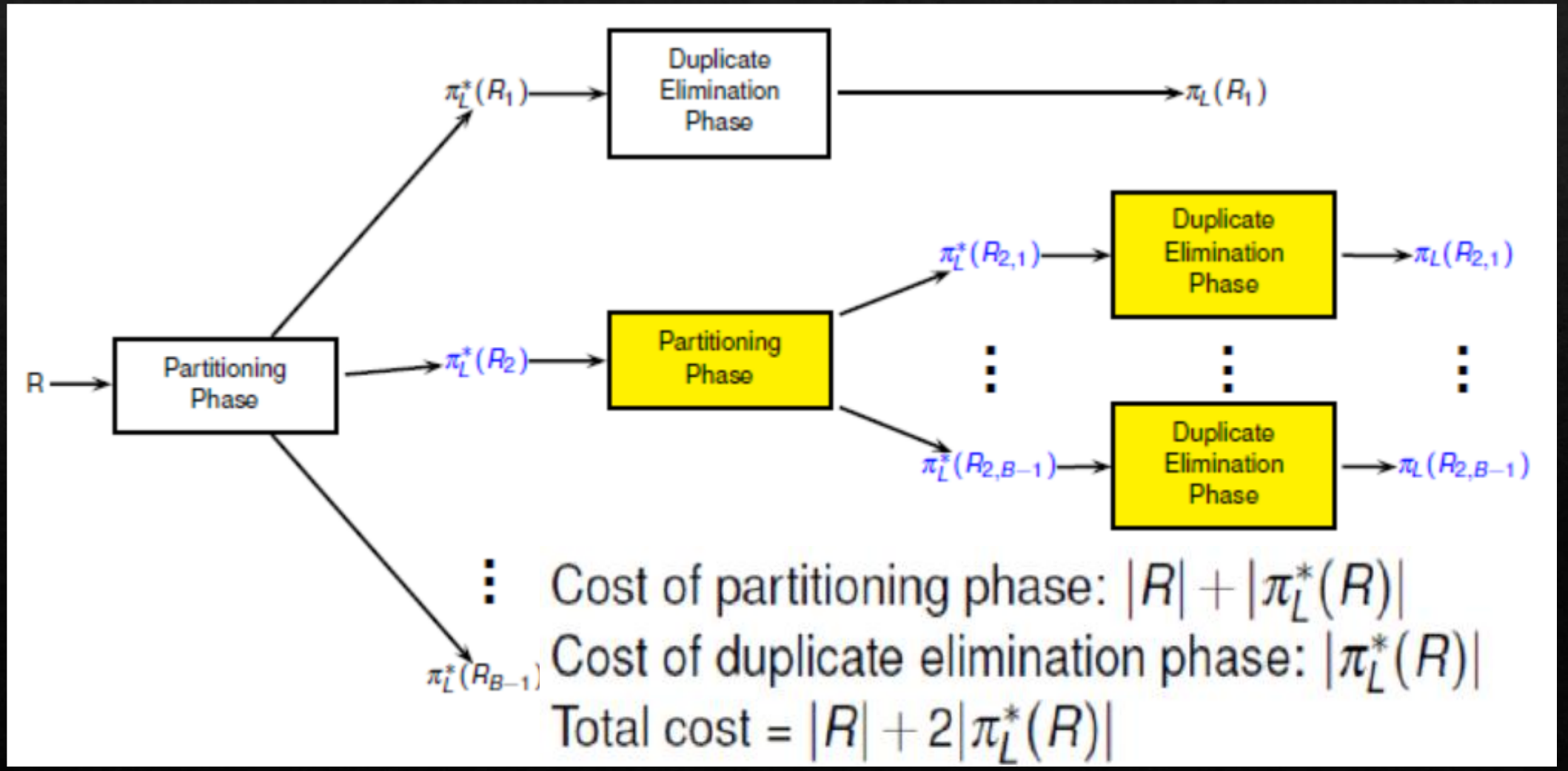
Optimized Sort-Based Approach



Hash-Based Approach



Hash-Based Approach



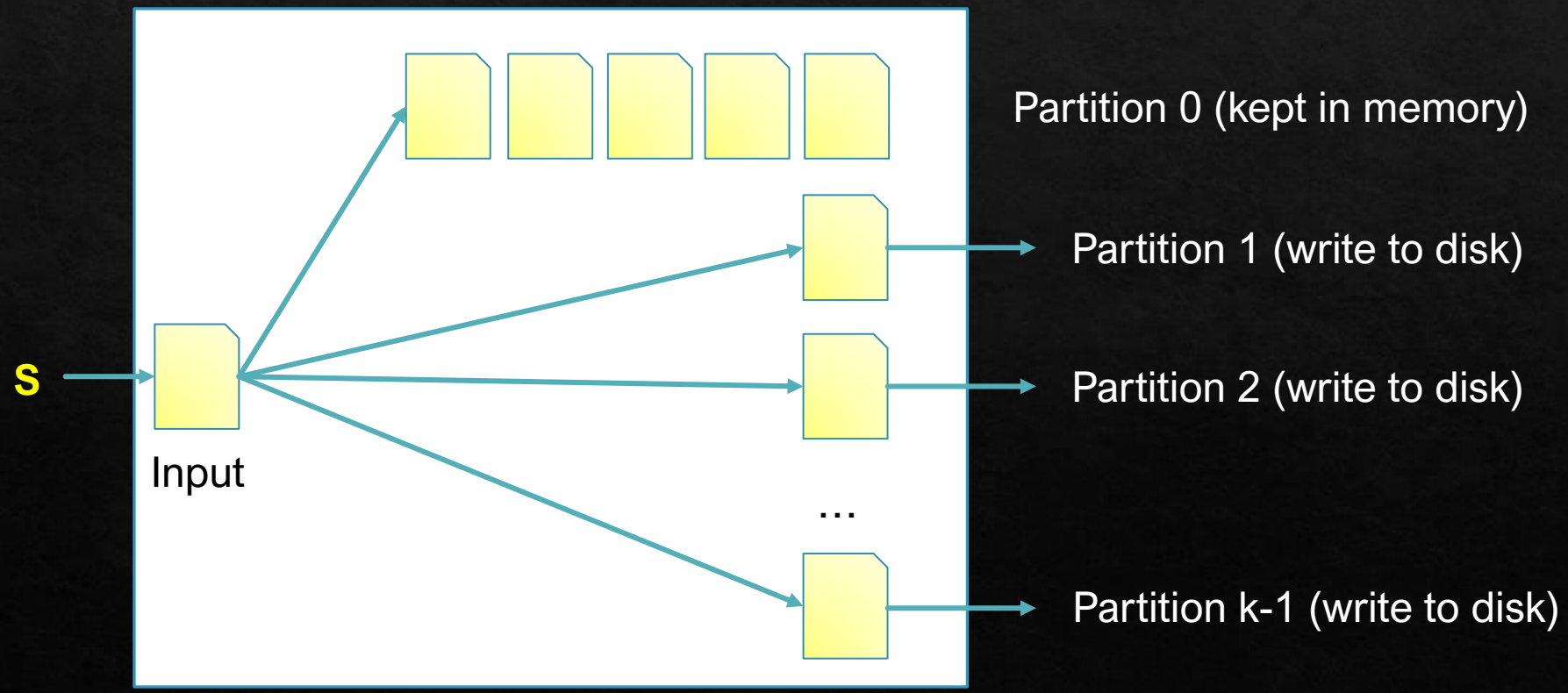
Tutorial Questions

Q1

- Hybrid Hash Join
  - k buckets
  - Keep bucket 0 of relation S in memory and join it on the fly as bucket 0 elements of relation R are being generated
- What is the cost of the algorithm?

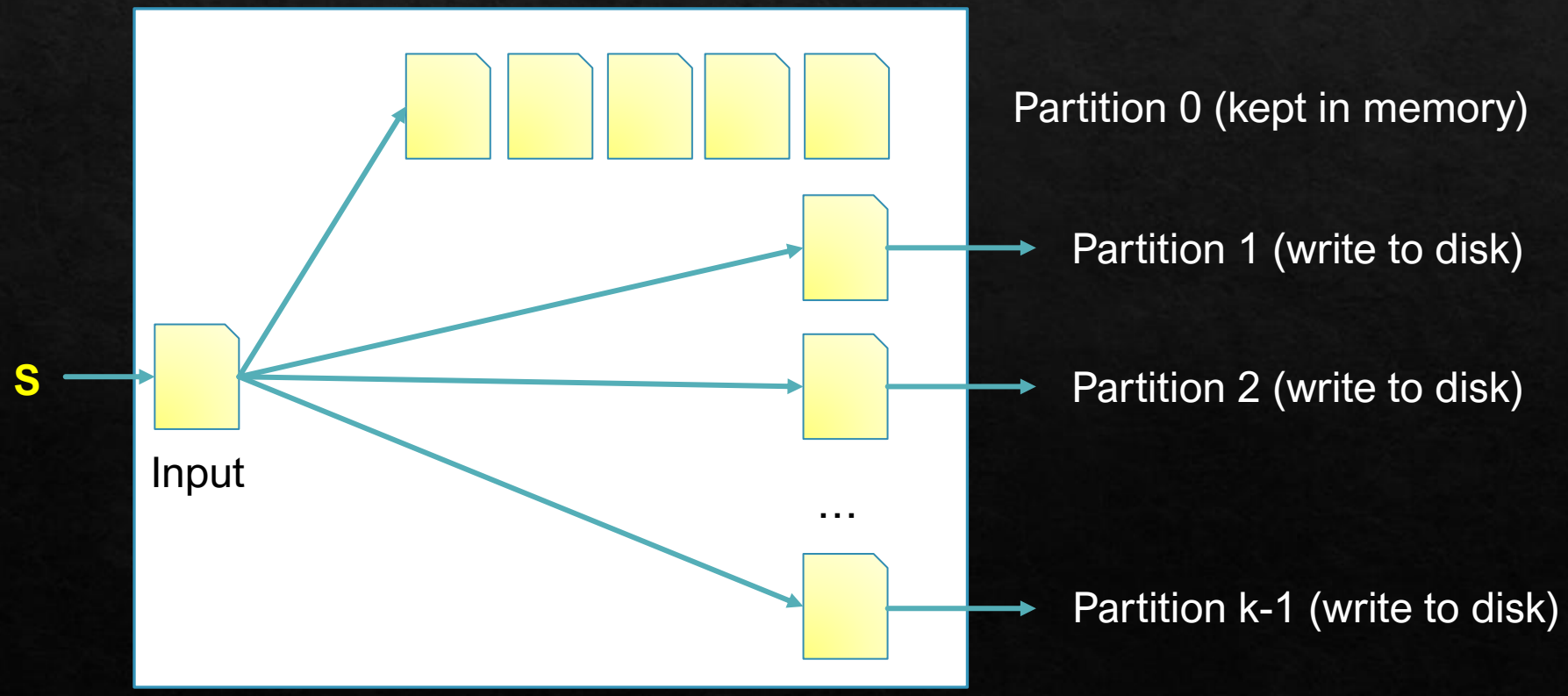
Q1

- Partition Phase: **Partition S**



Q1

- Partition Phase: **Partition S**

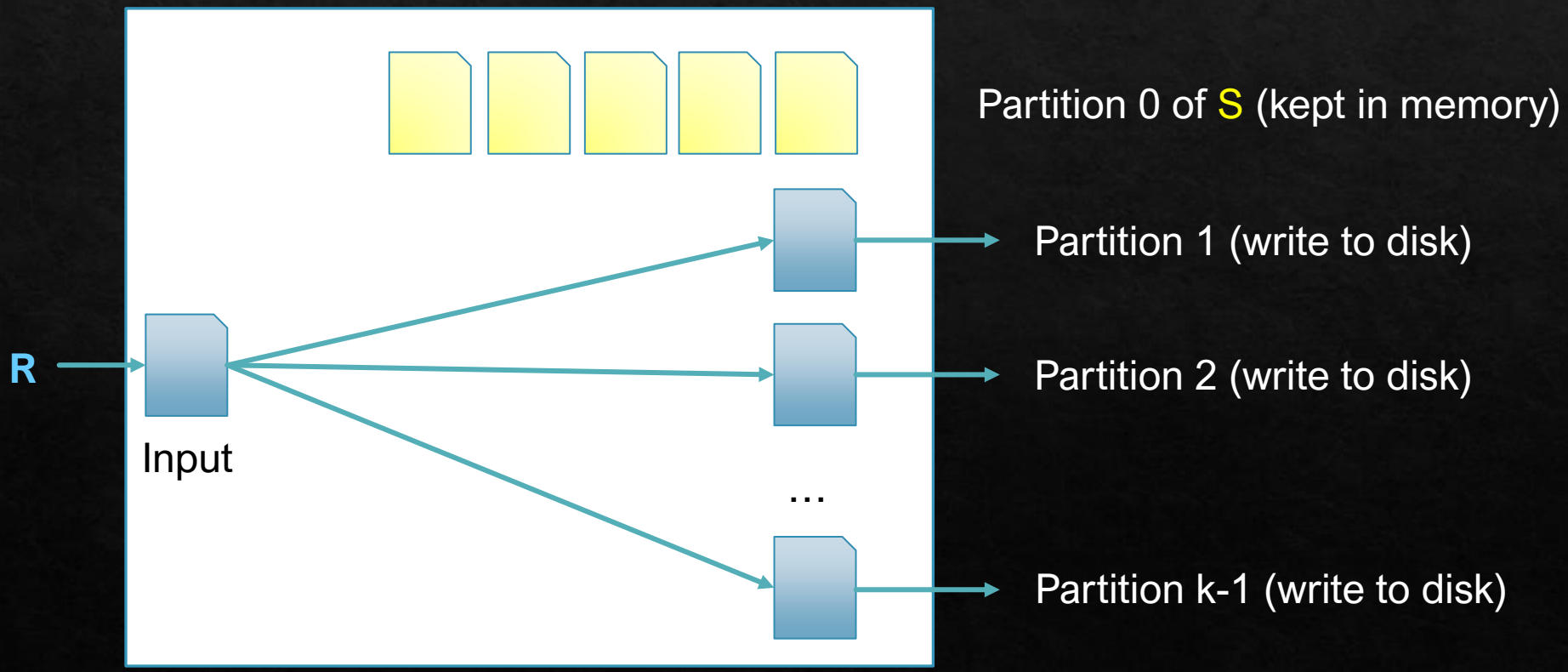


Size of Partition 0:
|S| / k

Cost:
Read + Write
= |S| + |S| - |S|/k
= (2-1/k)|S|

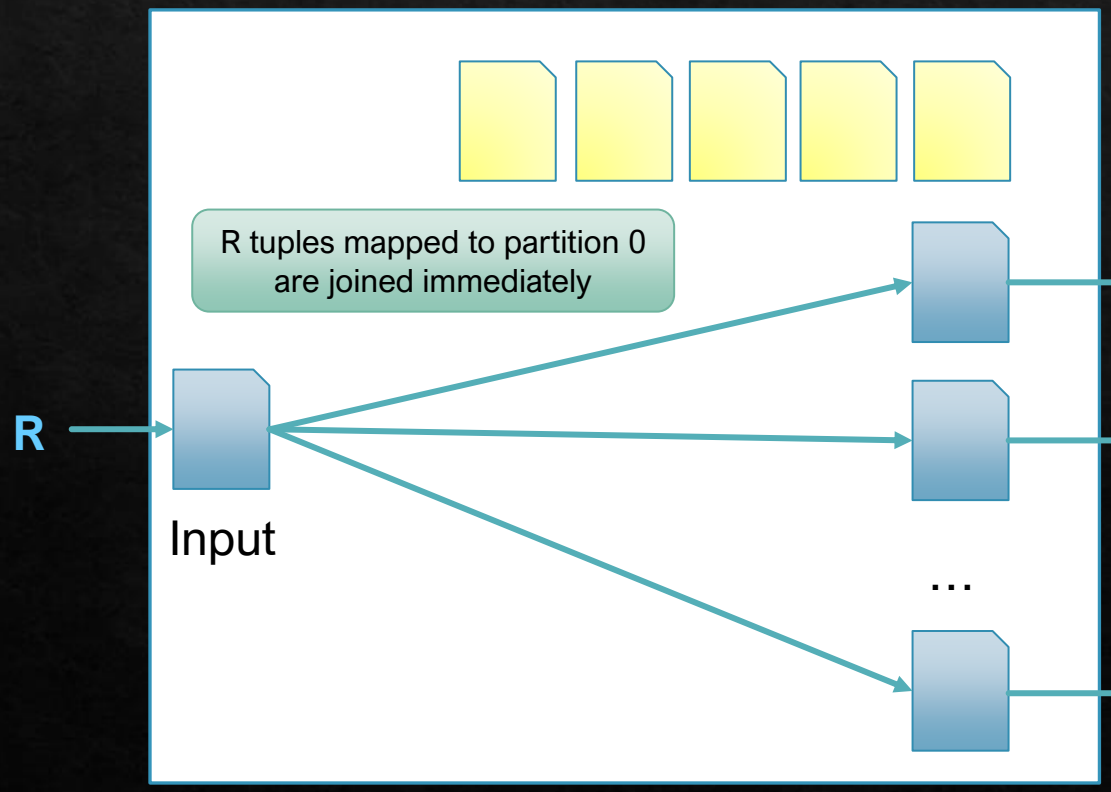
Q1

- Partition Phase: **Partition R**



Q1

- Partition Phase: **Partition R**



Partition 0 of S (kept in memory)

Partition 1 (write to disk)

Partition 2 (write to disk)

Partition k-1 (write to disk)

Size of Partition 0:
|R| / k

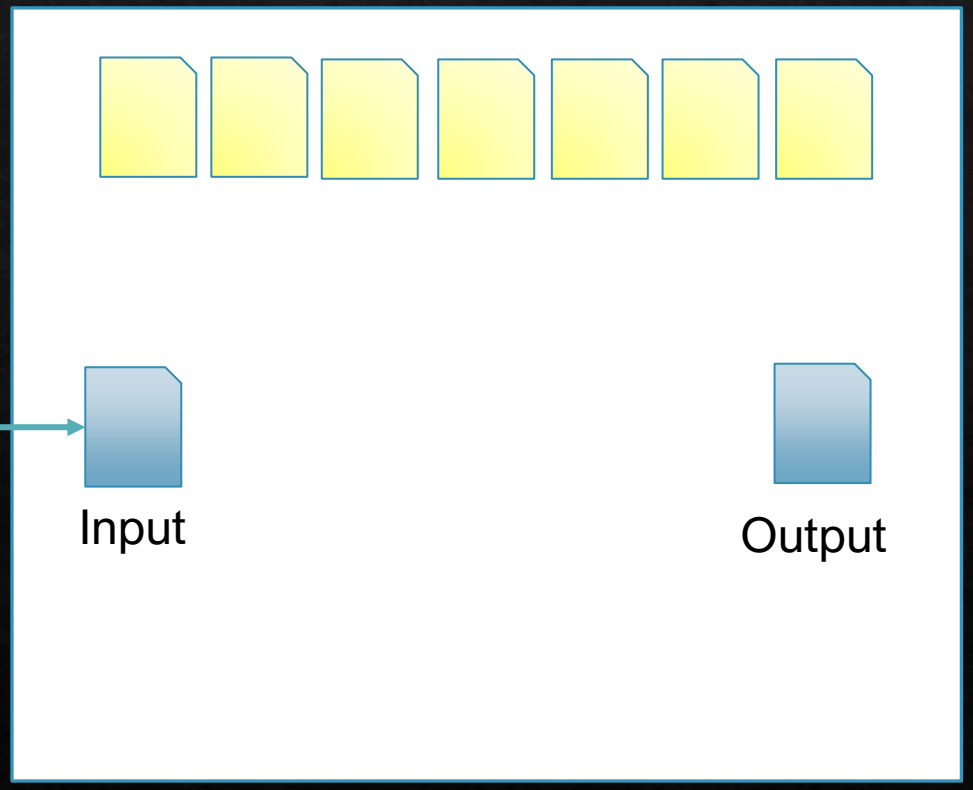
Cost:
Read + Write
= |R| + |R| - |R|/k
= (2-1/k)|R|

Q1

- Join Phase: **Join bucket 1 to (k-1) or S and R**

Load S partitions one by one

Load pages of Partition x of R one by one



Partition x of S (kept in memory)
where x = {1,2,...,k-1}

Same as original

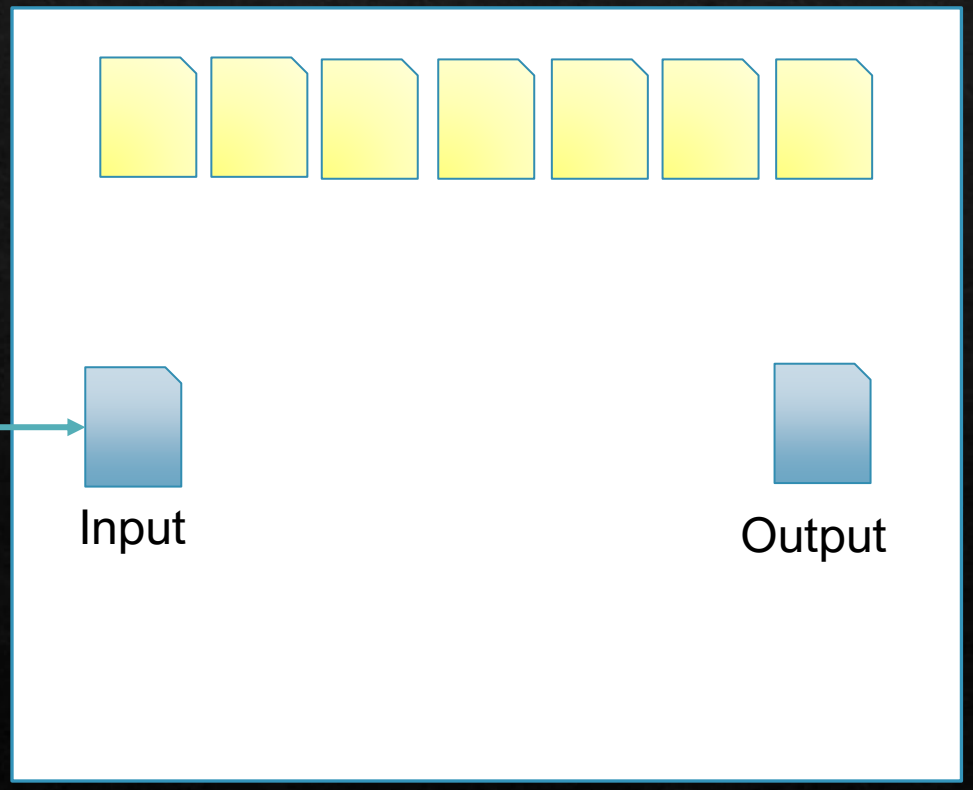
Cost:
Read + Write
= |S| - |S|/k + |R| - |R|/k
= (1-1/k)(|S| + |R|)

Q1

- Join Phase: **Join bucket 1 to (k-1) or S and R**

Load S partitions one by one

Load pages of Partition x of R one by one



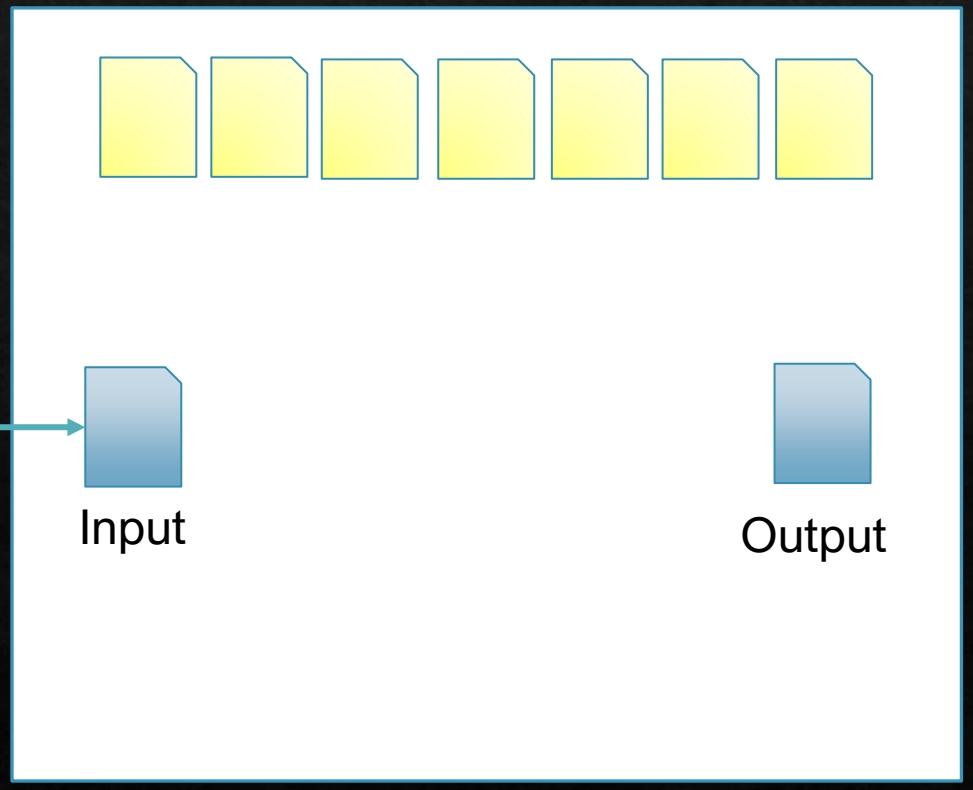
Partition x of S (kept in memory)
where x = {1,2,...,k-1}

Total Cost:
= (2-1/k)|S| + (2-1/k)|R| + (1-1/k)(|S|+|R|)
= (3-2/k)(|S|+|R|)

Q1

- Join Phase: **Join bucket 1 to (k-1) or S and R**

Load S partitions one by one



Load pages of Partition x of R one by one

Partition x of S (kept in memory)
where x = {1,2,...,k-1}

Total Cost:
= (2-1/k)|S| + (2-1/k)|R| + (1-1/k)(|S|+|R|)
= (3-2/k)(|S|+|R|) < 3(|S|+|R|)

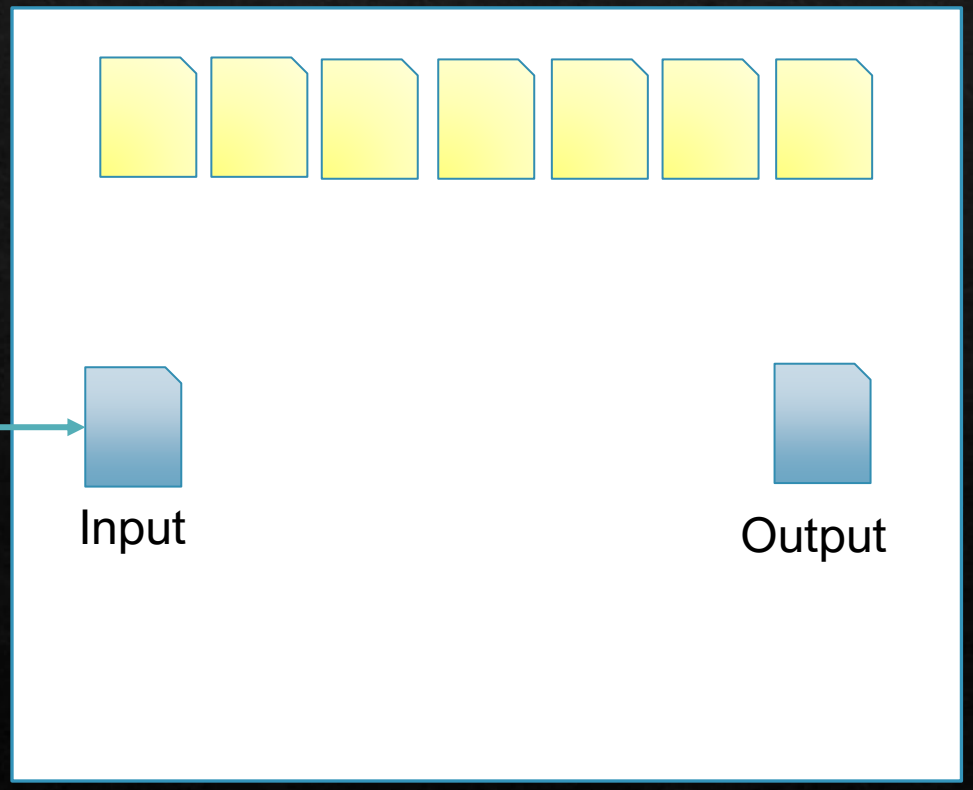
- Always better than original GRACE Hash join
- As k gets smaller, cost savings increase!
- Able to return initial answers faster (faster initial response time)

Q1

- Assumes uniformly distributed buckets (which is not always the case)
- For partition phase of R, we need 1 input buffer, k-1 output buffer for partitions of R, and 1 output buffer for join results of partition 0. Hence in memory partition must not be larger than **B-2-(k-1)**

◈ Join Phase: **Join bucket 1 to (k-1) or S and R**

Load S partitions one by one



Load pages of Partition x of R one by one

Partition x of S (kept in memory)
where x = {1,2,...,k-1}

Total Cost:

$= (2-1/k)|S| + (2-1/k)|R| + (1-1/k)(|S|+|R|)$

$= (3-2/k)(|S|+|R|) < 3(|S|+|R|)$

- Always better than original GRACE Hash join
- As k gets smaller, cost savings increase!
- Able to return initial answers faster (faster initial response time)

Q2

- Advantages of join index (rid, sid)

- Disadvantages

Q2

- Advantages of join index (rid, sid)
  - No need to perform join processing, simply scan index and retrieve relevant data pages
  - Initial response time is good
  - If it is B-tree, results are sorted by join columns
- Disadvantages
  - Updates are costly to maintain. E.g. every insert into relation R require join tuples of S to be computed and added to index. Same applies to deletion
  - Can it perform worse than standard join processing?

Q2

- Advantages of join index (rid, sid)
  - No need to perform join processing, simply scan index and retrieve relevant data pages
  - Initial response time is good
  - If it is B-tree, results are sorted by join columns
- Disadvantages
  - Updates are costly to maintain. E.g. every insert into relation R require join tuples of S to be computed and added to index. Same applies to deletion
  - Can it perform worse than standard join processing? Yes, if the search result is extremely huge wrt to the original relations, probing the index might cost significant overhead

Q3a

- applicant(**pid**, cityid, income, gmat)
- location(**cityid**, country)
- Find applicants that earn a salary greater than 60,000 and have GMAT scores higher than the average score of applicants from USA

Q3a

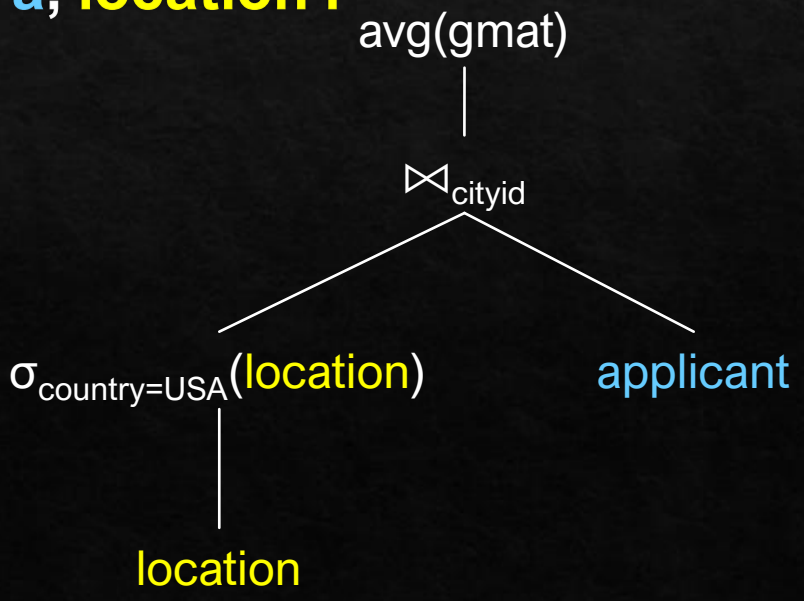
- applicant(**pid**, cityid, income, gmat)
- location(**cityid**, country)
- Find applicants that earn a salary greater than 60,000 and have GMAT scores higher than the average score of applicants from USA
- SELECT pid FROM applicant, location
  WHERE income > 60000
  AND gmat >
  (SELECT AVG(gmat) FROM applicant a, location l
  WHERE a.cityid = l.cityid
  AND l.country = “USA”)

Q3a

- SELECT pid FROM applicant
  WHERE income > 60000
  AND gmat >
  (SELECT AVG(a.gmat) FROM applicant a, location l
  WHERE a.cityid = l.cityid
  AND l.country = “USA”)

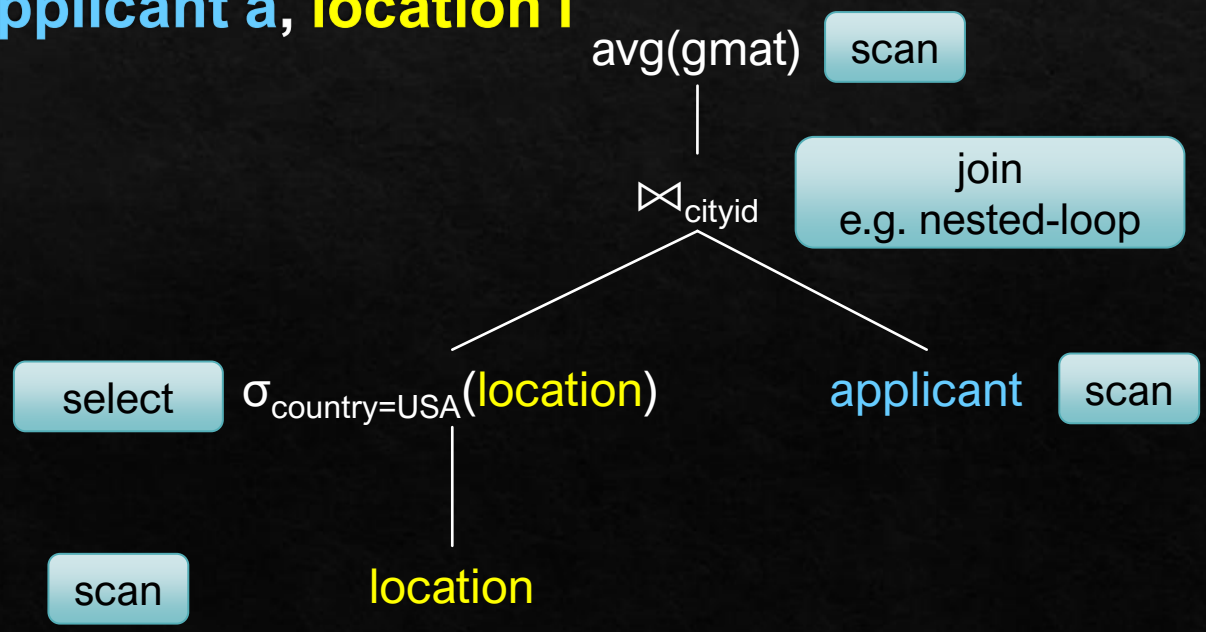
Q3a

- SELECT pid FROM applicant
  WHERE income > 60000
  AND gmat >
  (SELECT AVG(a.gmat) FROM applicant a, location l
  WHERE a.cityid = l.cityid
  AND l.country = “USA”)



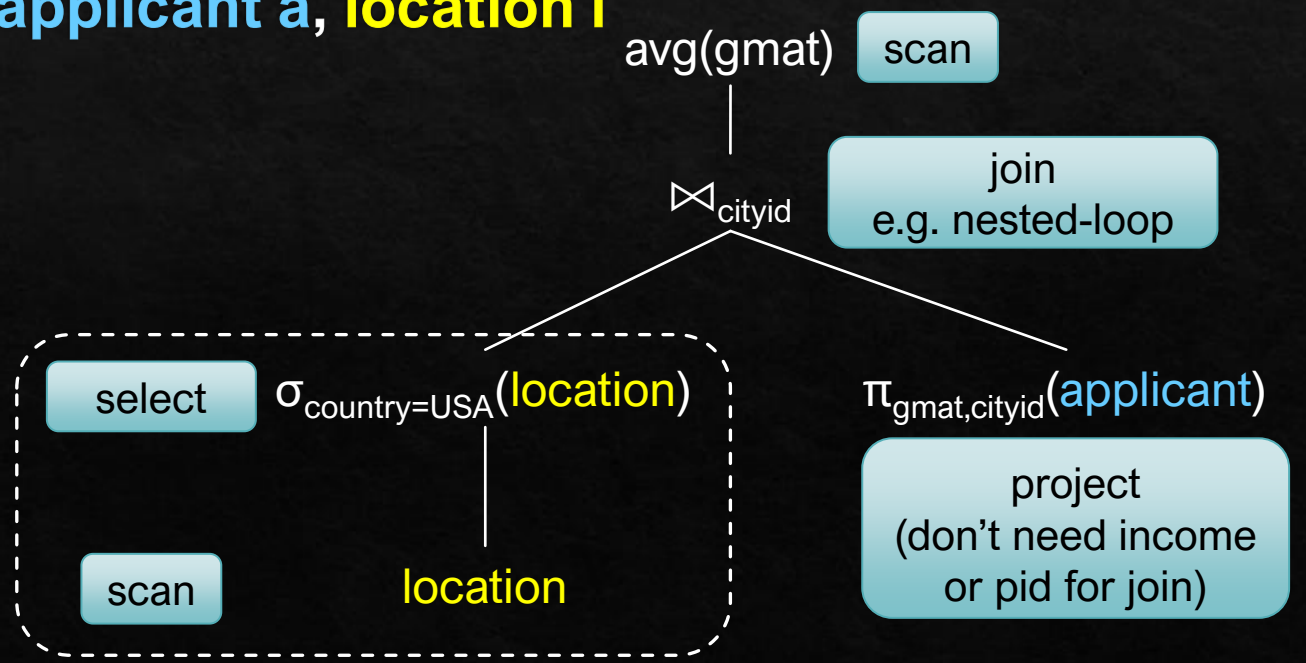
Q3

- SELECT pid FROM applicant
  WHERE income > 60000
  AND gmat >
  (SELECT AVG(a.gmat) FROM applicant a, location l
  WHERE a.cityid = l.cityid
  AND l.country = "USA")



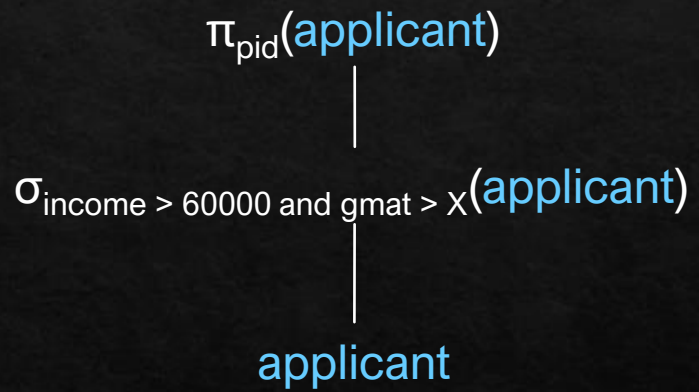
Q3a

- SELECT pid FROM applicant
  WHERE income > 60000
  AND gmat >
  (SELECT AVG(a.gmat) FROM applicant a, location l
  WHERE a.cityid = l.cityid
  AND l.country = "USA")



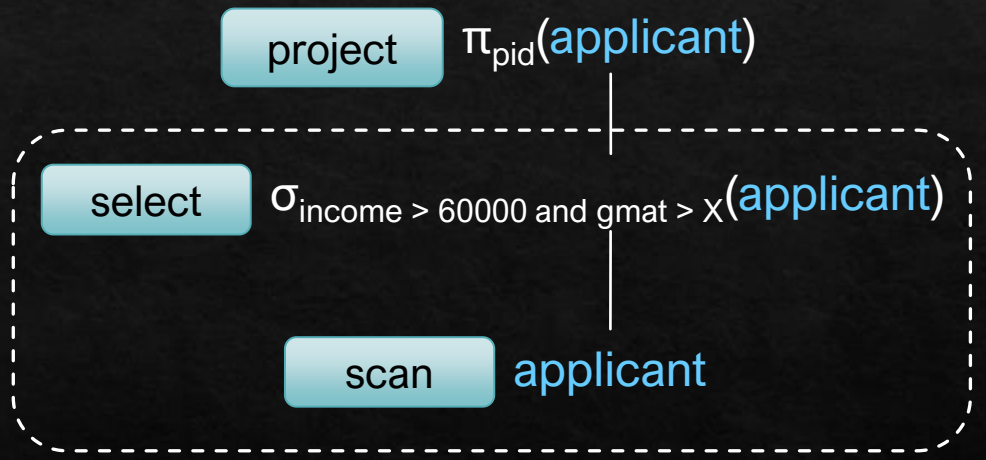
Q3a

- **SELECT pid FROM applicant,**
  WHERE income > 60000
  AND gmat > X



Q3a

- **SELECT pid FROM applicant,**
  WHERE income > 60000
  AND gmat > X



Q3b

- applicant(**pid**, cityid, income, gmat)
- location(**cityid**, country)
- Find applicants that earn a salary greater than 60,000 and have GMAT scores higher than the average score of applicants from the **same US city**”

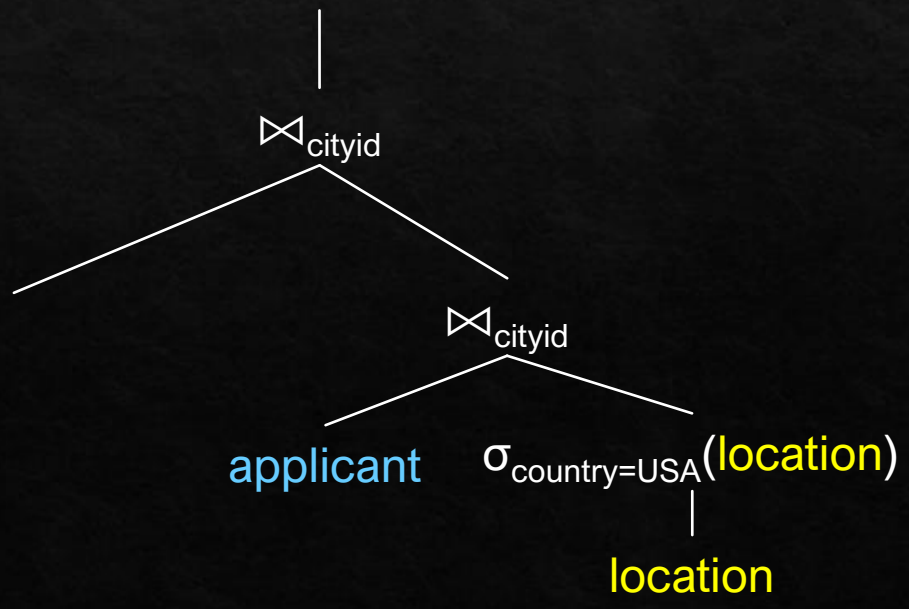
Q3b

- applicant(**pid**, cityid, income, gmat)
- location(**cityid**, country)
- Find applicants that earn a salary greater than 60,000 and have GMAT scores higher than the average score of applicants from the **same US city**"
- SELECT a1.pid, a1.income FROM applicant a1
  WHERE a1.income > 60000
  AND a1.gmat >
  (SELECT avg(a2.gmat)
  FROM applicant a2, location l2
  WHERE a1.cityid = a2.cityid
  AND a2.cityid = l2.cityid
  AND l2.country = "USA");

Q3b

Naive

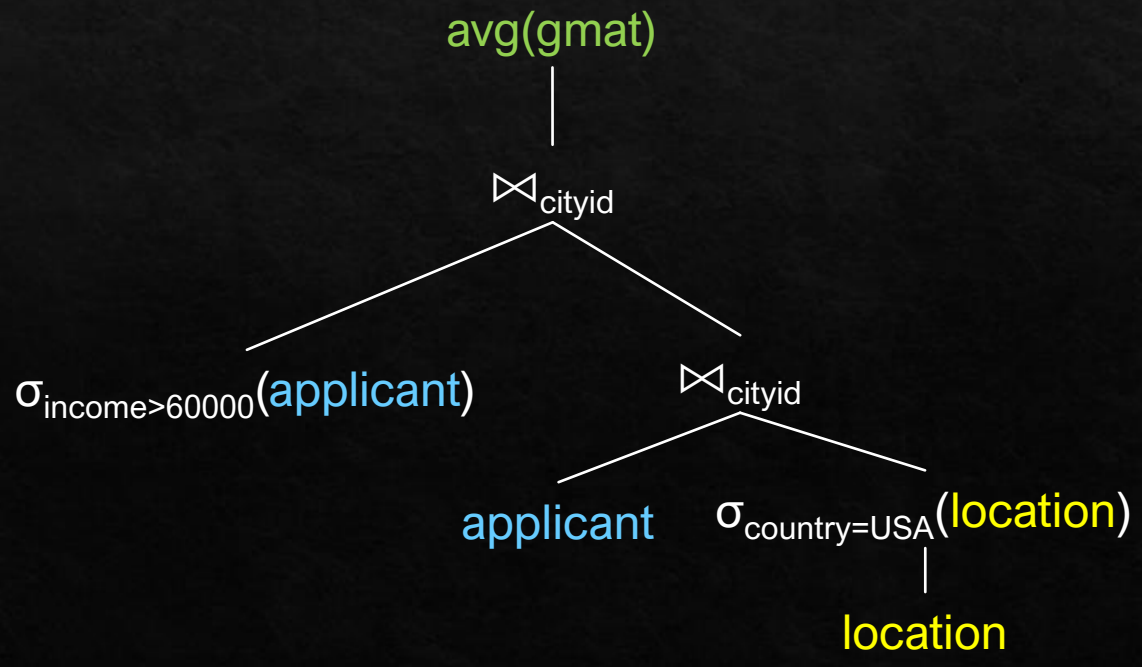
```
SELECT a1.pid, a1.income FROM applicant a1
WHERE a1.income > 60000
AND a1.gmat >
        (SELECT avg(a2.gmat)
        FROM applicant a2, location l2
        WHERE a1.cityid = a2.cityid
        AND a2.cityid = l2.cityid
        AND l2.country = "USA");
```



Q3b

Naive

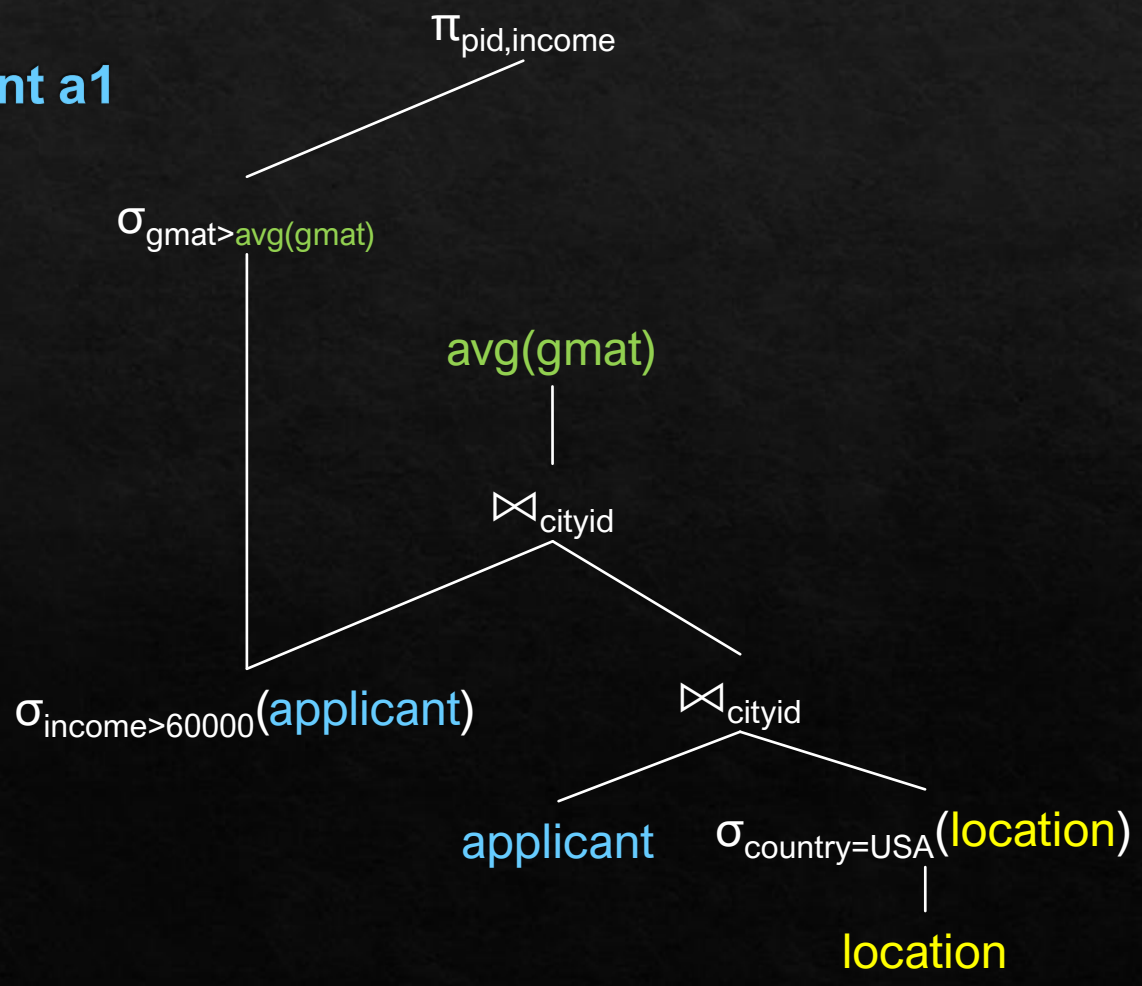
- SELECT a1.pid, a1.income FROM applicant a1
  WHERE a1.income > 60000
  AND a1.gmat >
  (SELECT avg(a2.gmat)
  FROM applicant a2, location l2
  WHERE a1.cityid = a2.cityid
  AND a2.cityid = l2.cityid
  AND l2.country = "USA");



Q3b

Naive

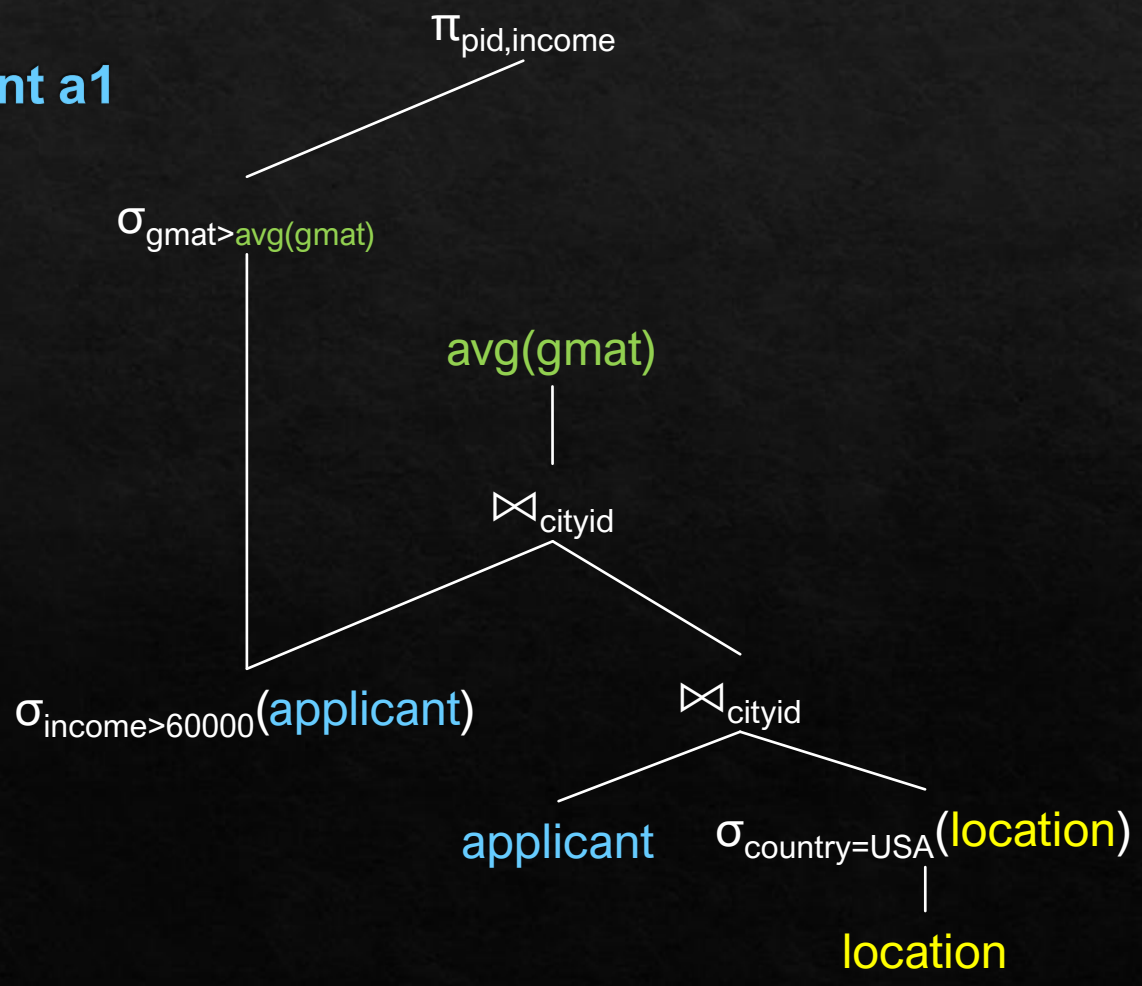
```
SELECT a1.pid, a1.income FROM applicant a1
WHERE a1.income > 60000
AND a1.gmat >
        (SELECT avg(a2.gmat)
        FROM applicant a2, location l2
        WHERE a1.cityid = a2.cityid
        AND a2.cityid = l2.cityid
        AND l2.country = "USA");
```



Q3b

Naive

```
SELECT a1.pid, a1.income FROM applicant a1
WHERE a1.income > 60000
AND a1.gmat >
        (SELECT avg(a2.gmat)
        FROM applicant a2, location l2
        WHERE a1.cityid = a2.cityid
        AND a2.cityid = l2.cityid
        AND l2.country = "USA");
```



Expensive, computing the inner query (SELECT avg…) for every a1 tuple

Q3b

Improved

- **Inner query:**

```
(SELECT cityid, avg(a.gmat)
FROM applicant a, location l
WHERE a.cityid = l.cityid
AND l.country = “USA”
GROUP BY cityid) as t;
```

- Outer query:

```
SELECT a.pid, a.income
FROM applicant a, tmpAgg t
WHERE a.income > 60000
AND a.cityid = t.c1
AND a.gmat > t.c2
```

cityid	avg(gmat)
0	87
1	85
…	…

Q3b

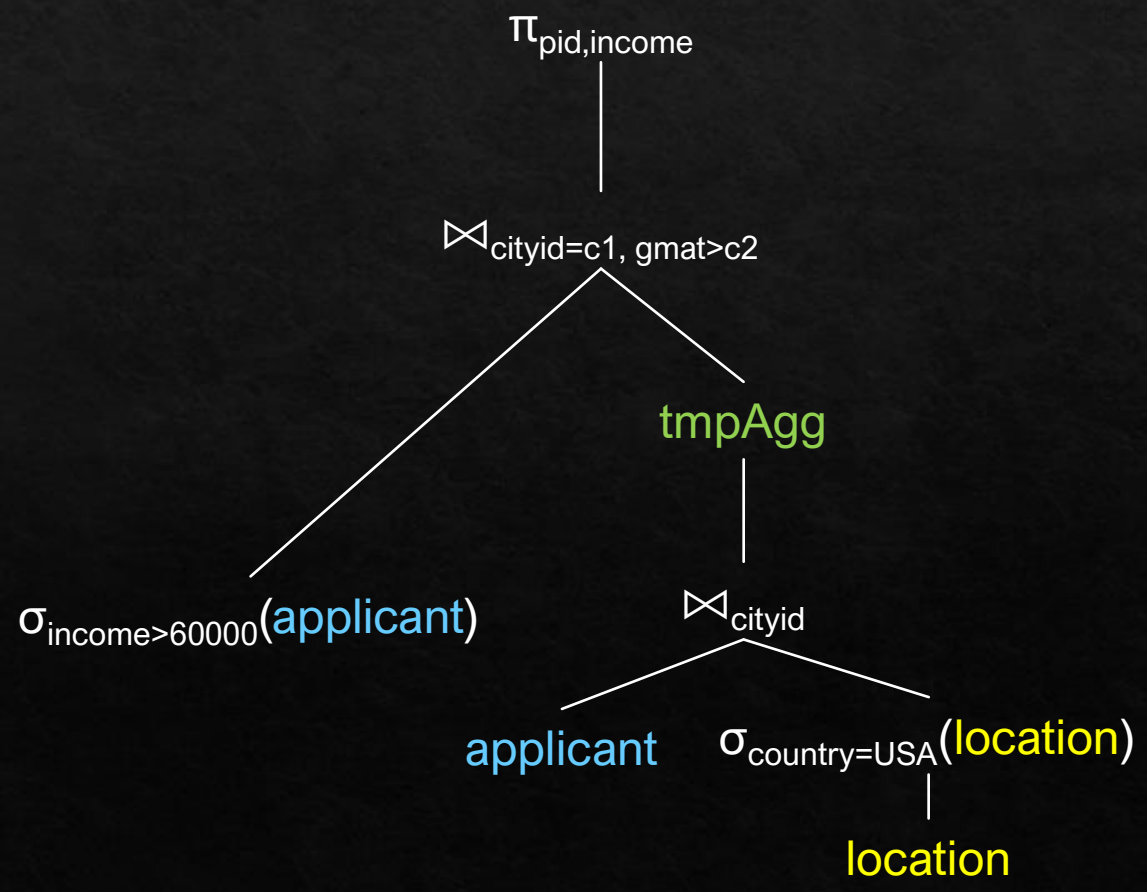
Improved

- **Inner query:**

```
(SELECT cityid, avg(a.gmat)
FROM applicant a, location l
WHERE a.cityid = l.cityid
AND l.country = “USA”
GROUP BY cityid) as t;
```

- **Outer query:**

```
SELECT a.pid, a.income
FROM applicant a, tmpAgg t
WHERE a.income > 60000
AND a.cityid = t.c1
AND a.gmat > t.c2
```



End