

How does Hash Join work in PostgreSQL and its derivatives

Yandong Yao

Pivotal Greenplum team

yyao@pivotal.io



Hash join in PostgreSQL

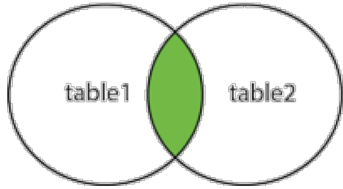
What is JOIN

An SQL join clause - corresponding to a join operation in relational algebra - combines columns from one or more tables in a relational database.

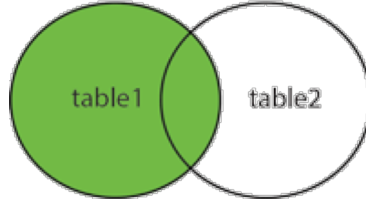
-- Wikipedia

JOIN Types

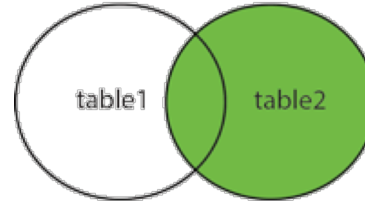
INNER JOIN



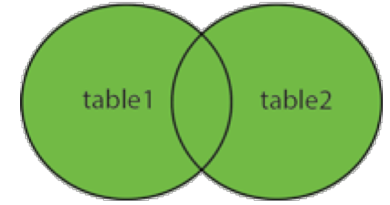
LEFT JOIN



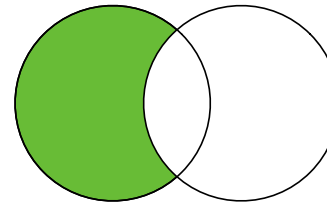
RIGHT JOIN



FULL OUTER JOIN



ANTI JOIN



SEMI JOIN

Examples

student table

id	name	age
10	Jack	25
12	Tom	26
13	Ariel	25

score table

id	stu_id	subject	score
1	10	math	95
2	10	history	98
3	12	math	97
4	15	history	92

```
CREATE TABLE student(id int, name text, age int);  
CREATE TABLE score(id int, stu_id int, subject text, score int);  
INSERT INTO student VALUES (10, 'Jack', 25), (12, 'Tom', 26), (13, 'Ariel', 25);  
INSERT INTO score VALUES (1, 10, 'math', 95), (2, 10, 'history', 98), (3, 12, 'math', 97), (4, 15, 'history', 92);
```

JOIN Examples

set enable_mergejoin to off;
set enable_hashagg to off;

JOIN Types	Examples
Inner JOIN	SELECT name, score FROM student st INNER JOIN score s ON st.id = s.stu_id
Left JOIN	SELECT name, score FROM student st LEFT JOIN score s ON st.id = s.stu_id
Right JOIN	SELECT name, score FROM student st RIGHT JOIN score s ON st.id = s.stu_id
Full JOIN	SELECT name, score FROM student st FULL JOIN score s ON st.id = s.stu_id
Semi JOIN	SELECT id, name FROM student st WHERE EXISTS (SELECT id FROM score WHERE stu_id = st.id)
Anti JOIN	SELECT name FROM student st WHERE NOT EXISTS (SELECT stu_id FROM score where score.stu_id = st.id)

Explain shows join type

JOIN Examples

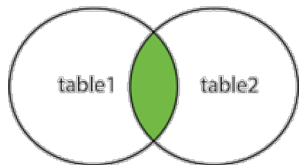
```
# SELECT * FROM student;
```

id	name	age
10	Jack	25
13	Ariel	25
12	Tom	26

```
# SELECT * FROM score;
```

id	stu_id	subject	score
1	10	math	95
2	10	history	98
3	12	math	97
4	15	history	92

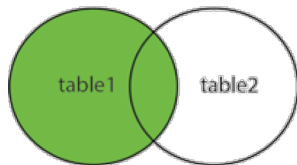
INNER JOIN



name	score
Tom	97
Jack	95
Jack	98

(3 rows)

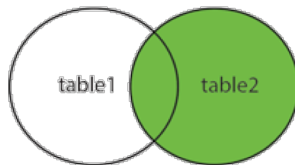
LEFT JOIN



name	score
Tom	97
Jack	95
Jack	98
Ariel	

(4 rows)

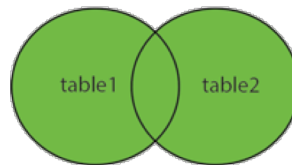
RIGHT JOIN



name	score
Tom	97
Jack	92
Jack	95
Jack	98

(4 rows)

FULL OUTER JOIN



name	score
Tom	97
Jack	92
Jack	95
Jack	98
Ariel	

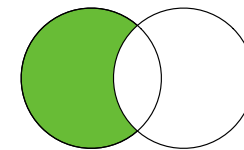
(5 rows)

Semi JOIN

id	name
10	Jack
12	Tom

(2 rows)

ANTI JOIN



name
Ariel

(1 row)

JOIN implementation algorithms

- Nested Loop
- Merge Join
- Hash Join

Hash join has two phases

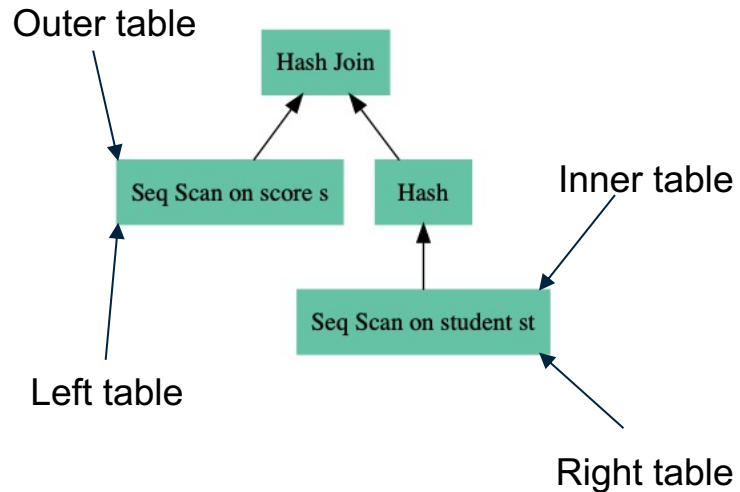
- **Build phase:** build hash table on the smaller table after applying possible local predicates, which is called **inner** table.
- **Probe phase:** scan tuple from another table and probe hash table for matches based on criteria, this 'another' table is called **outer** table

Let us start from inner join

```
SELECT name, subject, score FROM student st INNER JOIN score s ON st.id
= s.stu_id
```

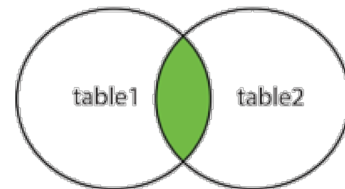
QUERY PLAN

```
-----
Hash Join (cost=35.42..297.73 ...)
  Hash Cond: (st.id = s.stu_id)
    -> Seq Scan on student st (cost=0.00..22.00)
    -> Hash (cost=21.30..21.30 rows=1130 width=8)
          -> Seq Scan on score s (cost=0.00..21.30)
(5 rows)
```



Inner join: build phase

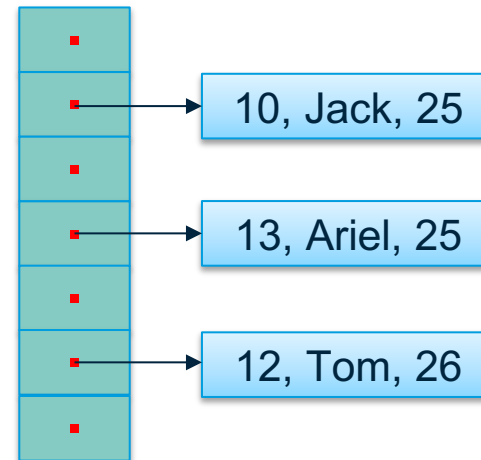
INNER JOIN



```
SELECT * FROM student;  
id | name | age  
---+-----+---  
10 | Jack | 25  
13 | Ariel | 25  
12 | Tom | 26
```



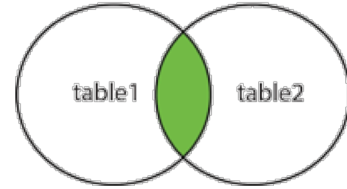
nbucket



```
SELECT name, subject, score FROM student st INNER JOIN score s ON st.id = s.stu_id
```

Inner join: probe phase

INNER JOIN



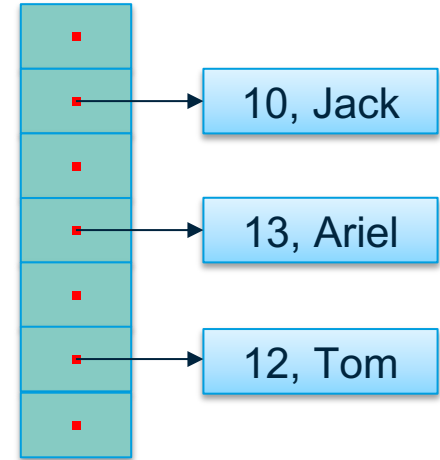
Score table

id	stu_id	subject	score
1	10	math	95
2	10	history	98
3	12	math	97
4	15	history	92



Jack	math	95
Jack	hist	98
Tom	math	97

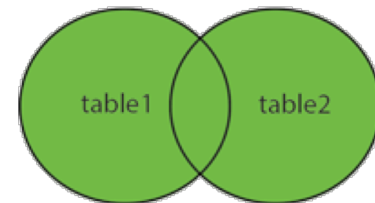
Hash table for student



```
SELECT name, subject, score FROM student st INNER JOIN score s ON st.id = s.stu_id
```


Full outer join

FULL OUTER JOIN

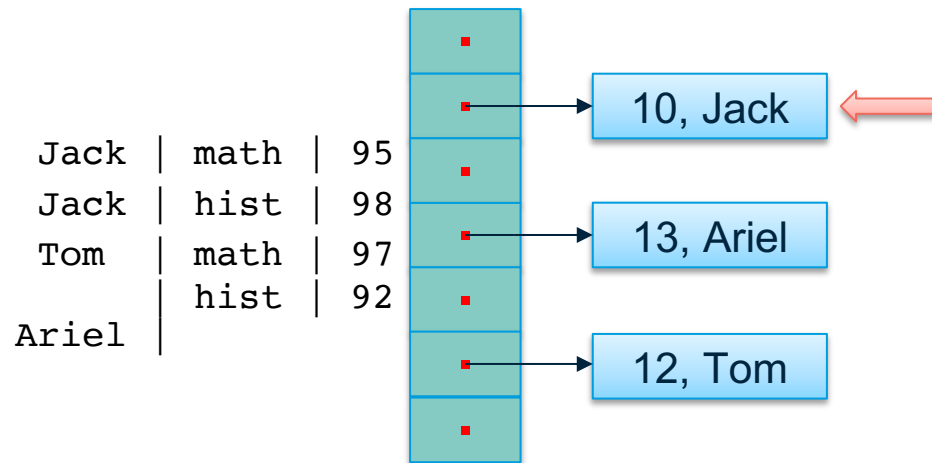


Score table

id	stu_id	subject	score
1	10	math	95
2	10	history	98
3	12	math	97
4	15	history	92



Hash table for student



Jack	math	95
Jack	hist	98
Tom	math	97
Ariel	hist	92

```
SELECT name, subject, score FROM student st INNER JOIN score s ON st.id = s.stu_id
```

JOIN SQL semantics vs. JOIN imp types

```
explain SELECT * FROM bigger_table LEFT JOIN smaller_table ON ...
```

Hash Left Join

```
Hash Cond: (b.id = s.id)
```

```
-> Seq Scan on bigger_table b
```

```
-> Hash
```

```
    -> Seq Scan on smaller_table s
```

```
(5 rows)
```

```
explain SELECT * FROM smaller_table LEFT JOIN bigger_table ON ...
```

Hash Right Join

```
Hash Cond: (s.id = b.id)
```

```
-> Seq Scan on bigger_table s
```

```
-> Hash
```

```
    -> Seq Scan on smaller_table b
```

```
(5 rows)
```

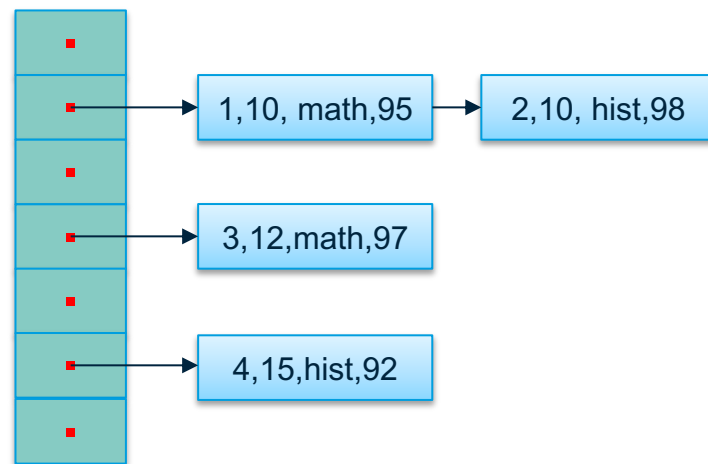
Semi join: stop with first match

student table

id	name	age
10	Jack	25
13	Ariel	25
12	Tom	26

10 | Jack
12 | Tom

Hash table for score on stu_id



```

SELECT id, name FROM student st
WHERE EXISTS (SELECT id FROM score WHERE stu_id = st.id)
  
```

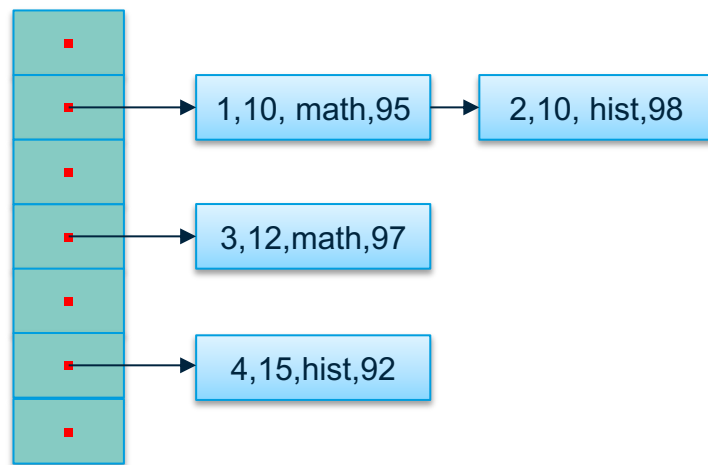
Anti join: emit tuple when there is no match

student table

id	name	age
10	Jack	25
13	Ariel	25
12	Tom	26

13 | Ariel

Hash table for score on stu_id

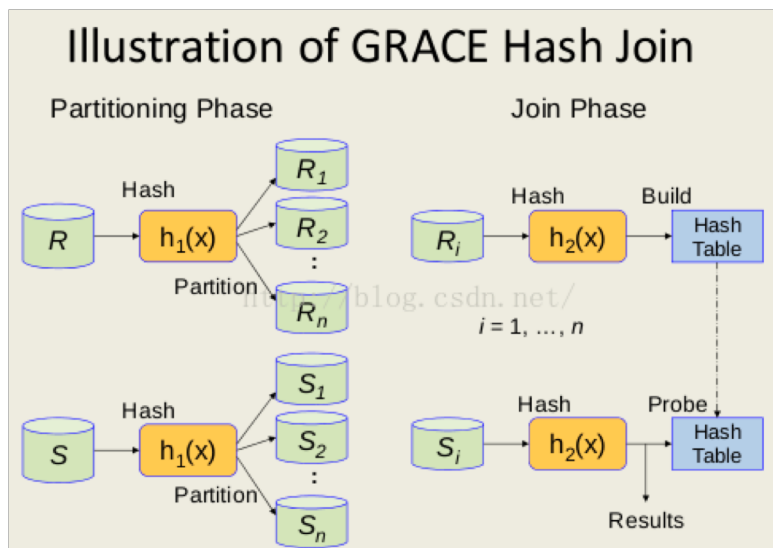


```

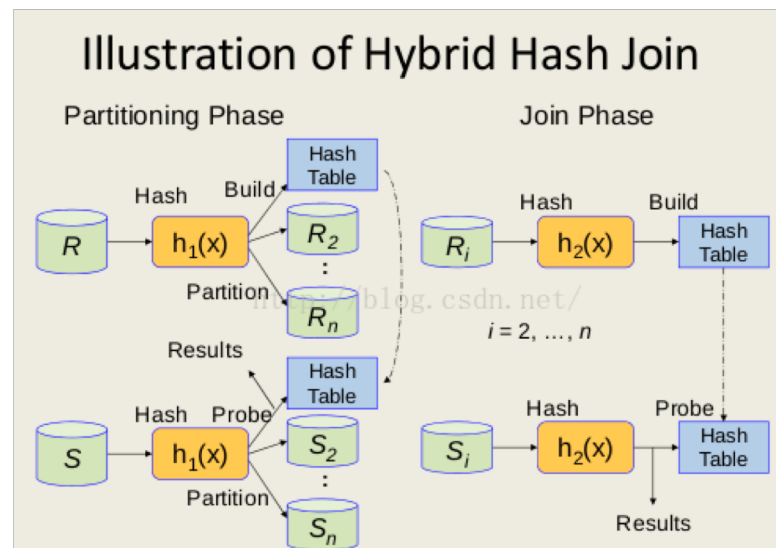
SELECT id, name FROM student st
WHERE id NOT EXISTS (SELECT stu_id FROM score)
  
```

What about if inner table is too big to fit into memory?

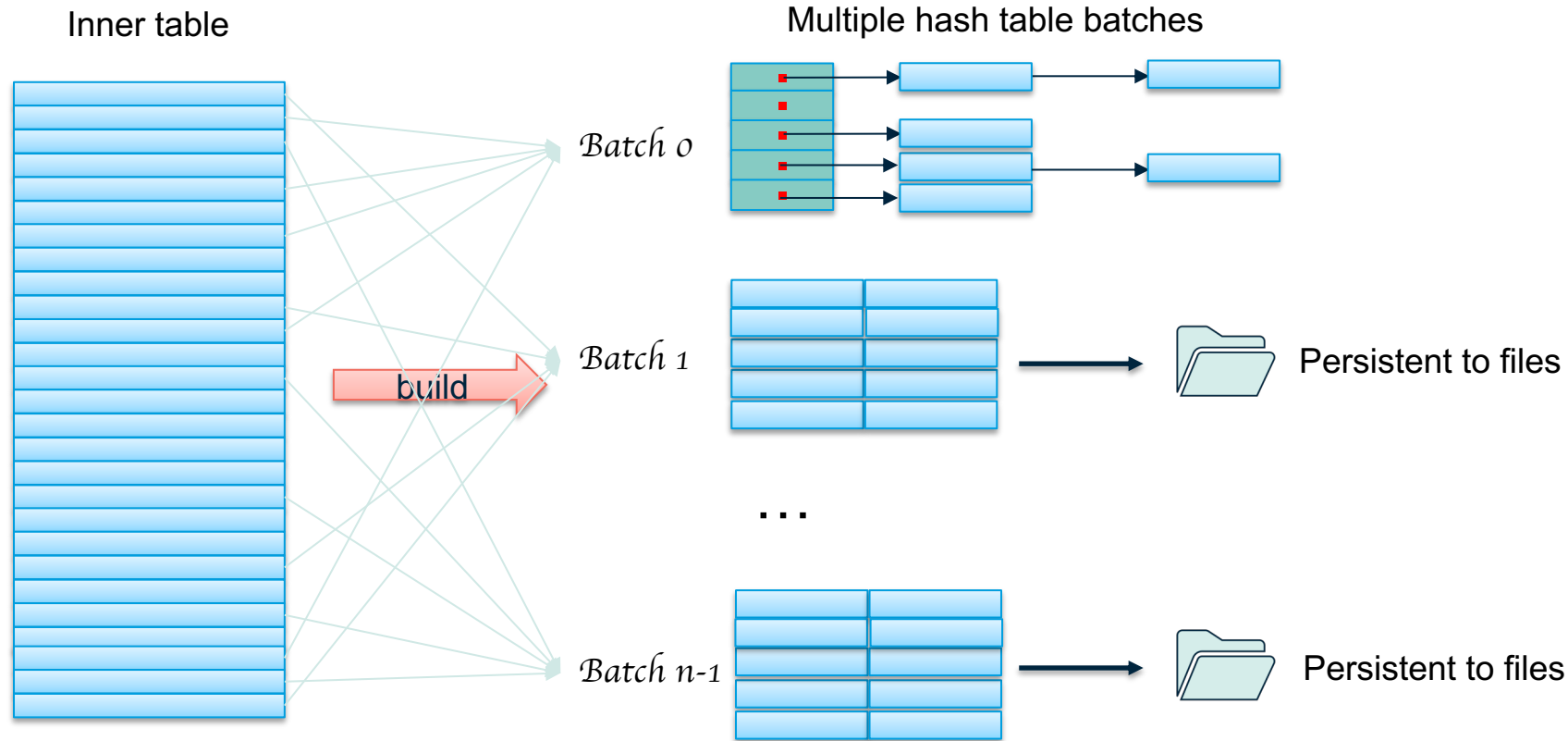
Grace Hash Join



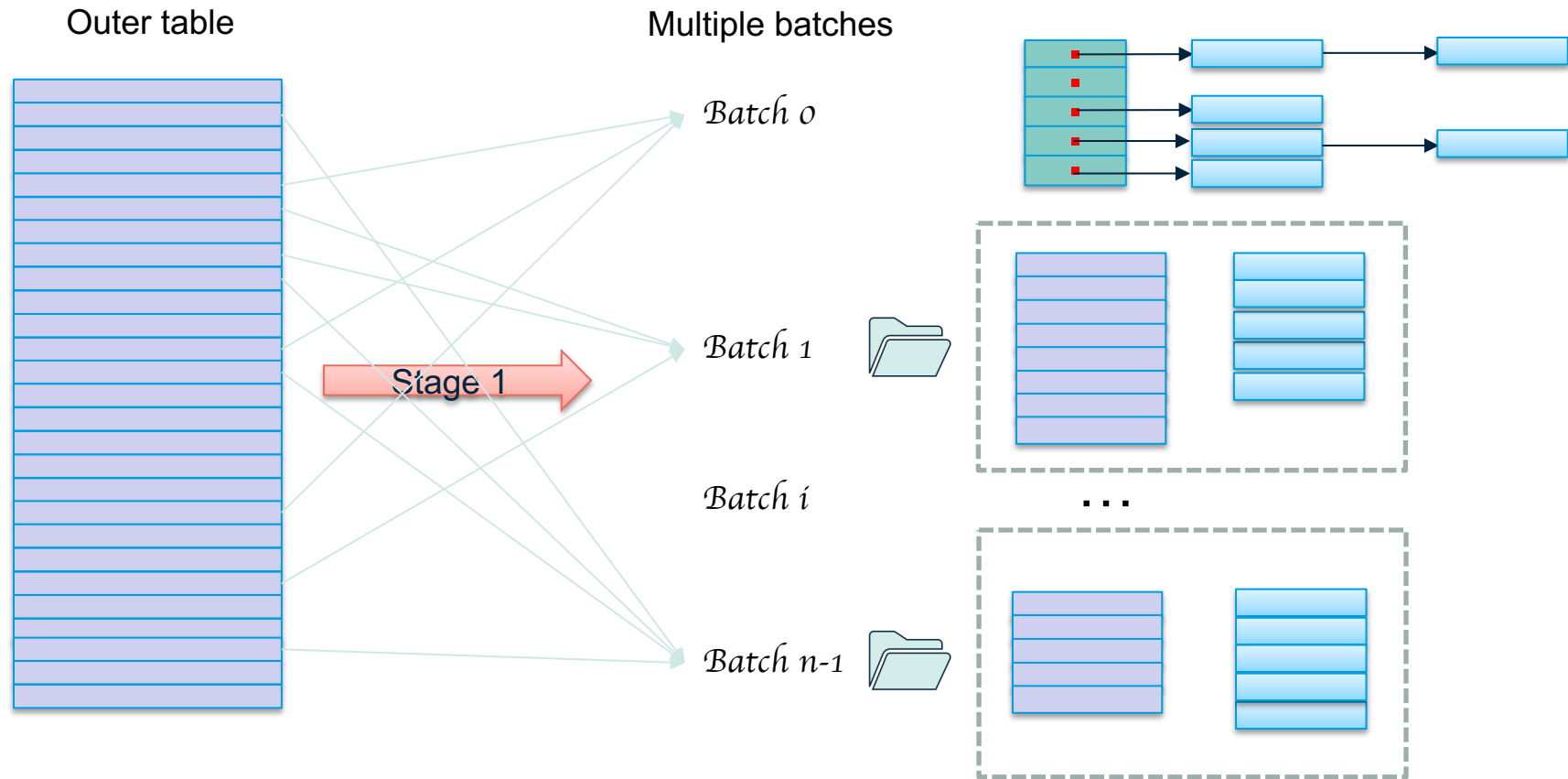
Hybrid Hash Join



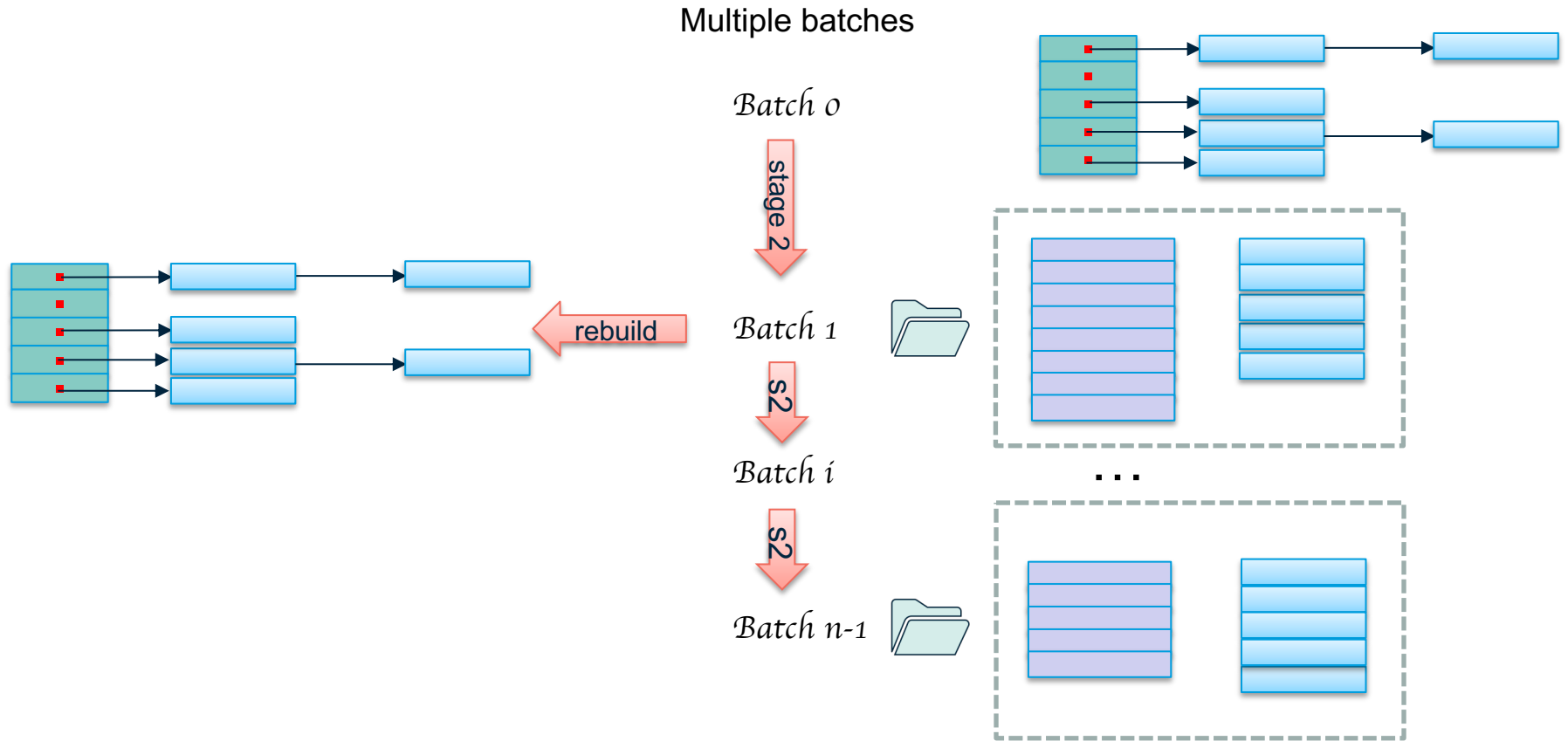
Partition phase for inner table of hybrid hash join



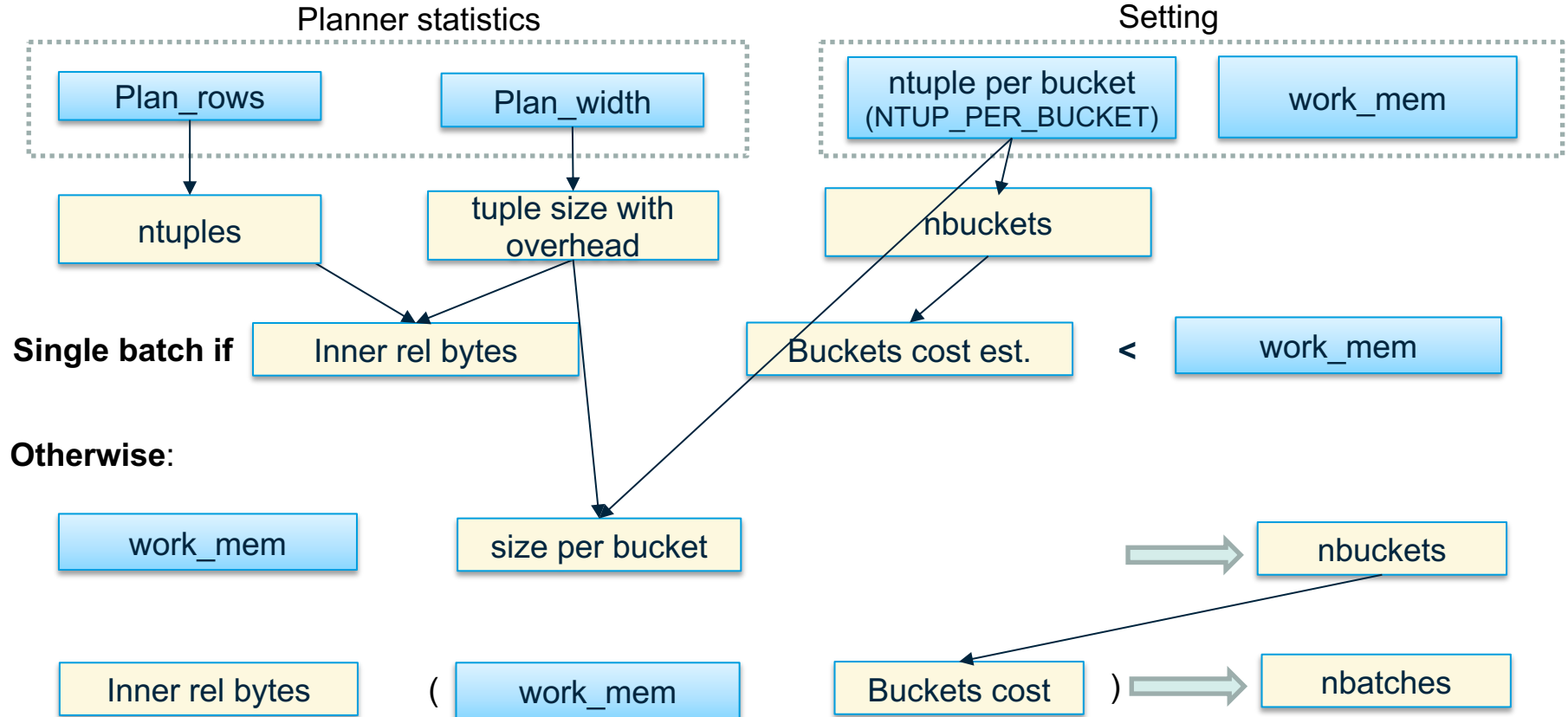
Partition phase for outer table of Hybrid hash join



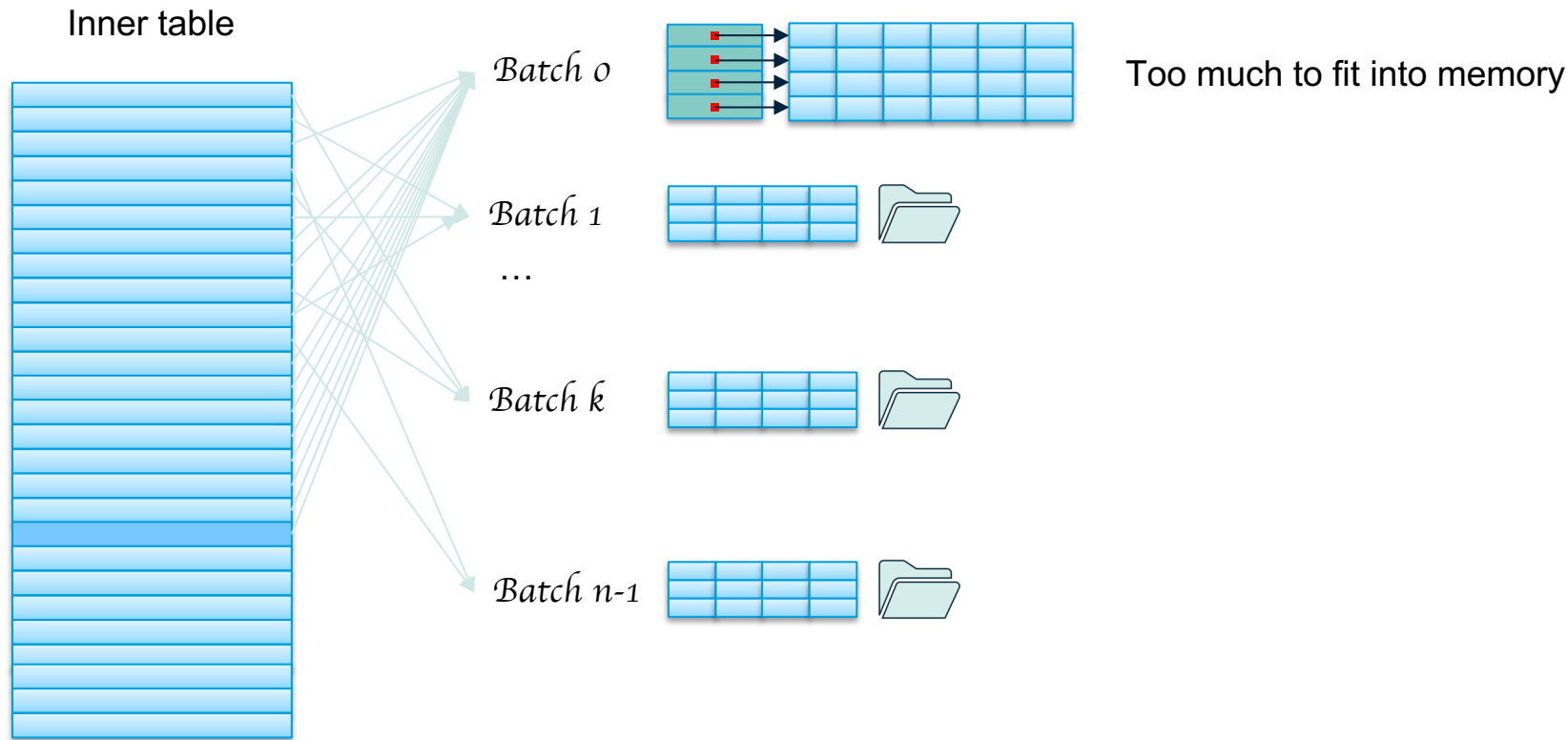
Join phase of Hybrid hash join: for later batches



How to determine number of buckets & batches

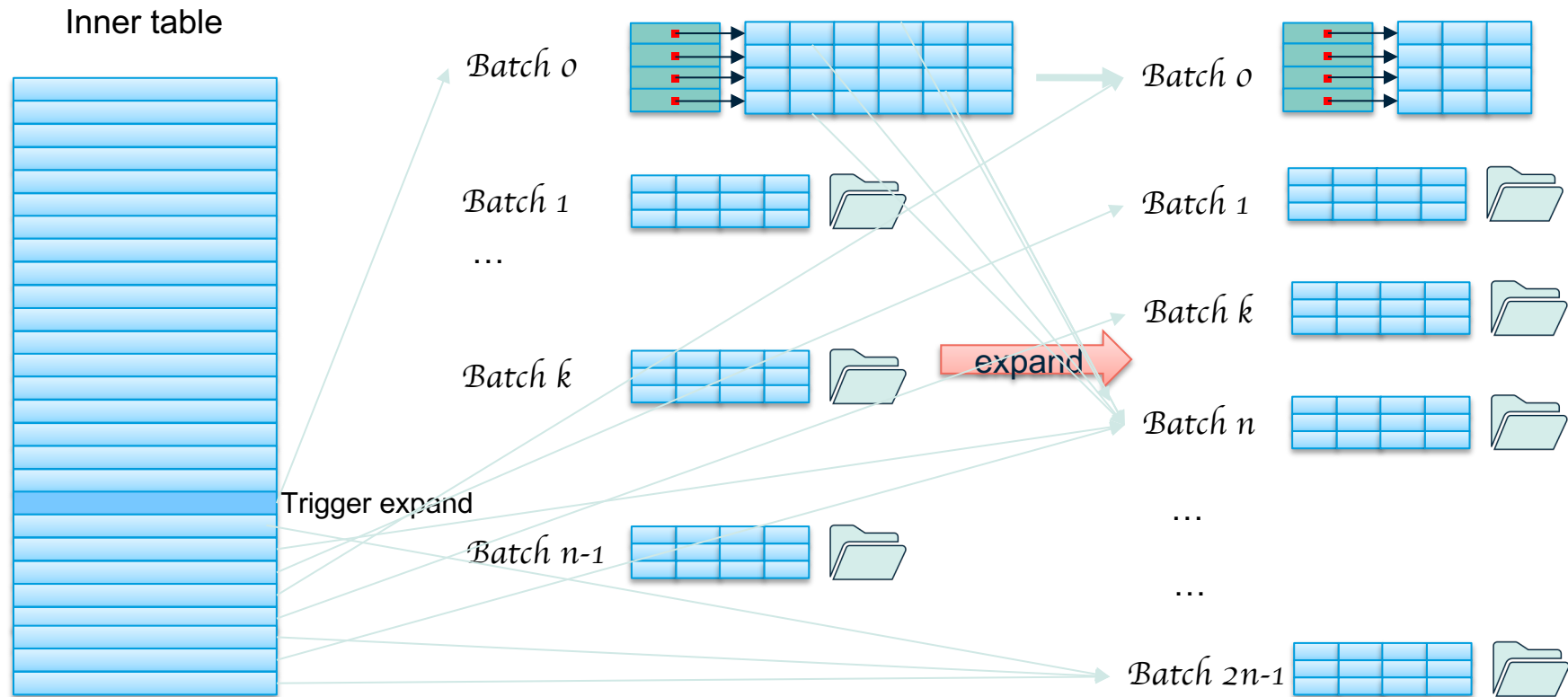


How about if batch 0 is too big?

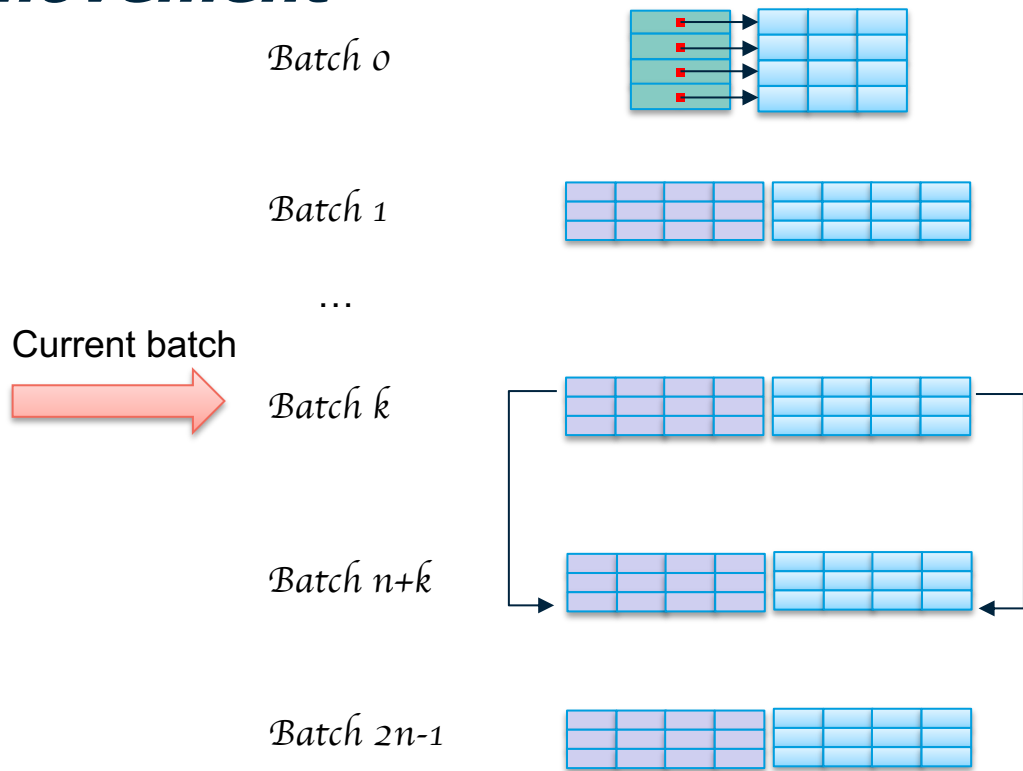


Double batches: $n \rightarrow 2n$

Re-calc batches according to $2n$

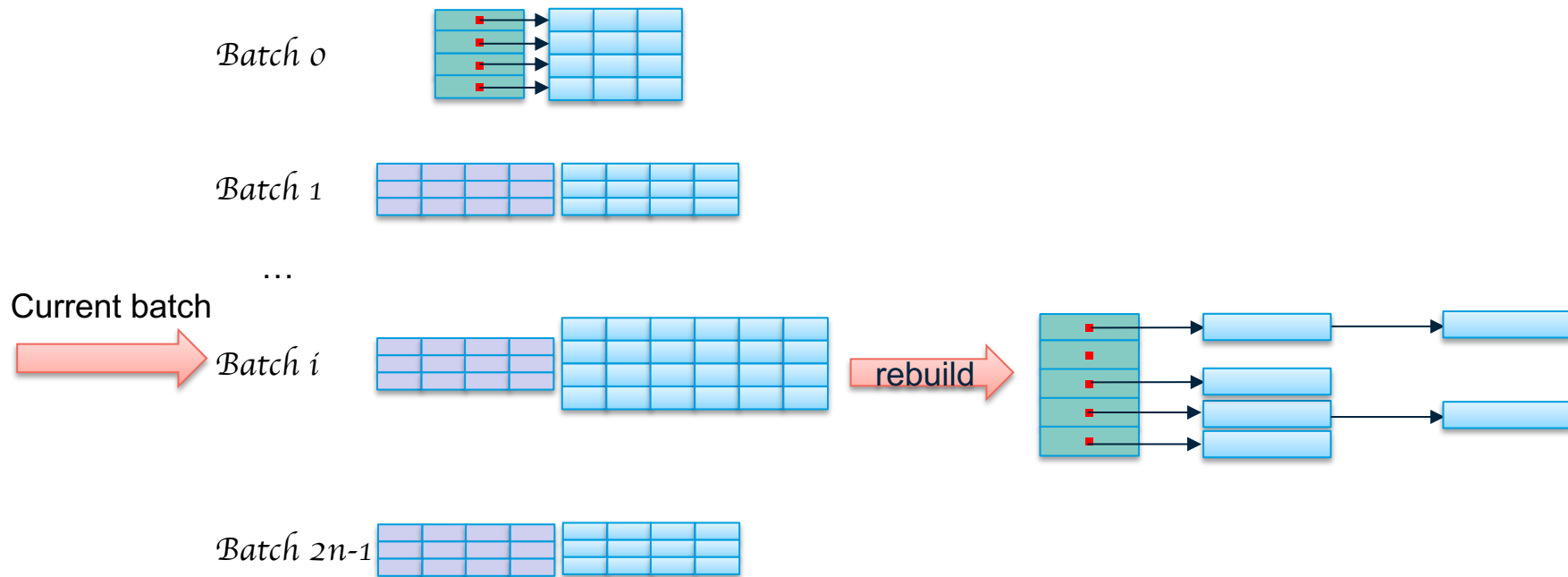


Observation: batch expands result in tuple movement

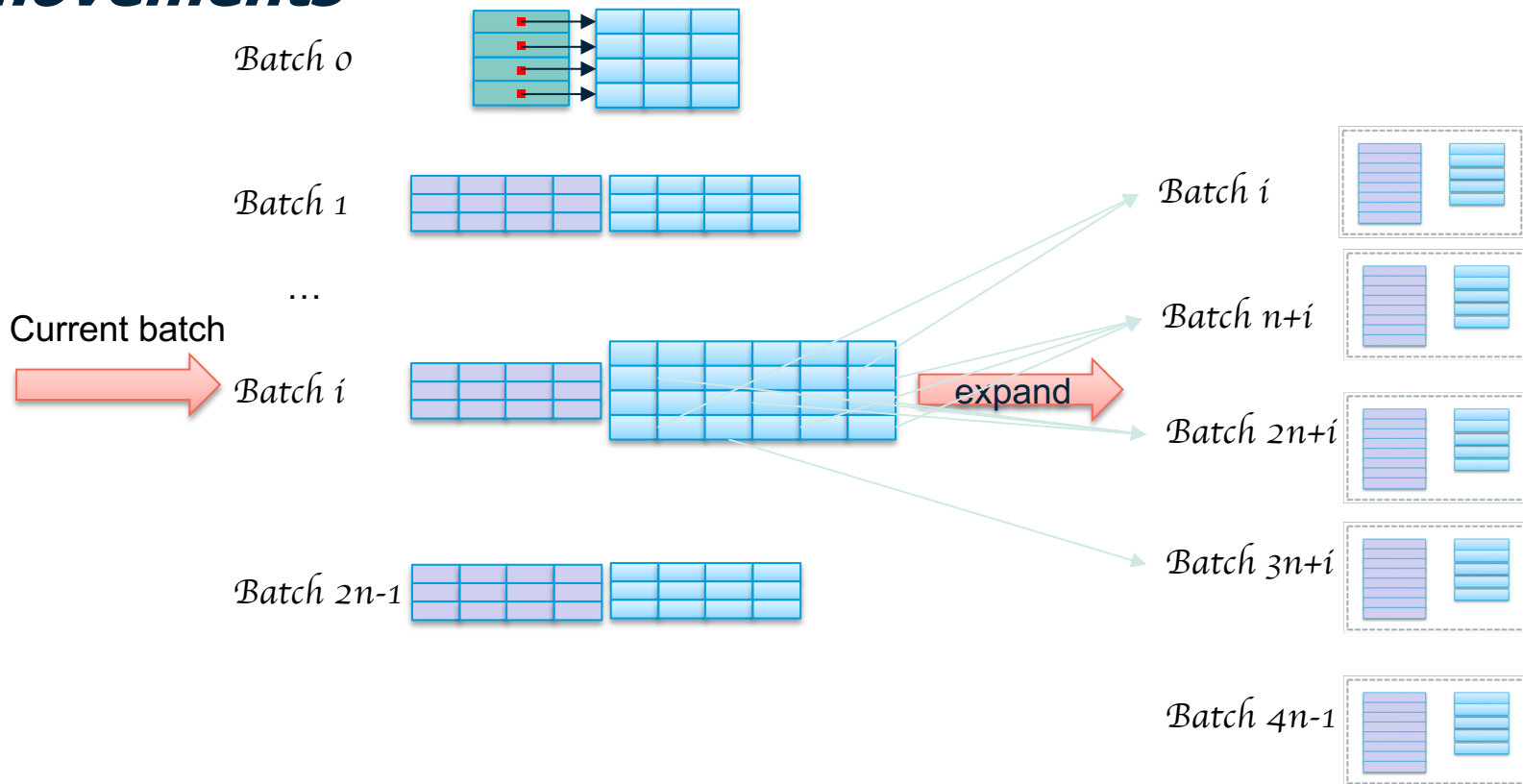


So tuples might move to later batches, as we build hash table from inner batch file, and scan outer batch file.

Batch i might result in batch expand again



Then expand batches again, and more tuple movements



Skew optimization overview

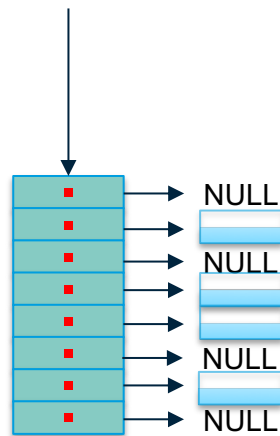
- Optimization if outer table has ***non uniform distribution***, so that most common values (MCV) will be processed in batch 0

Skew optimization: prepare skew hash table

$$\frac{work_mem * 0.02}{tupsize + extra_cost}$$

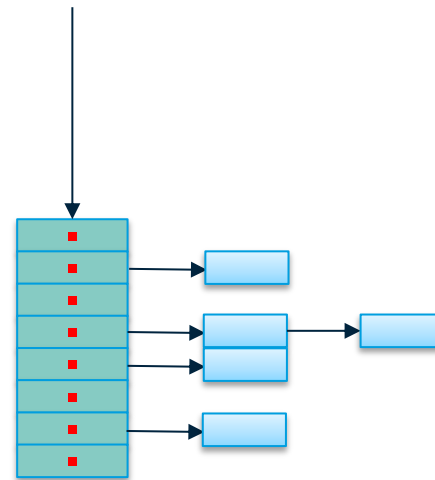
Number of MCVs
memory allowed

pg_statistic of outer table
MVC stats



Skew hash table

Inner table



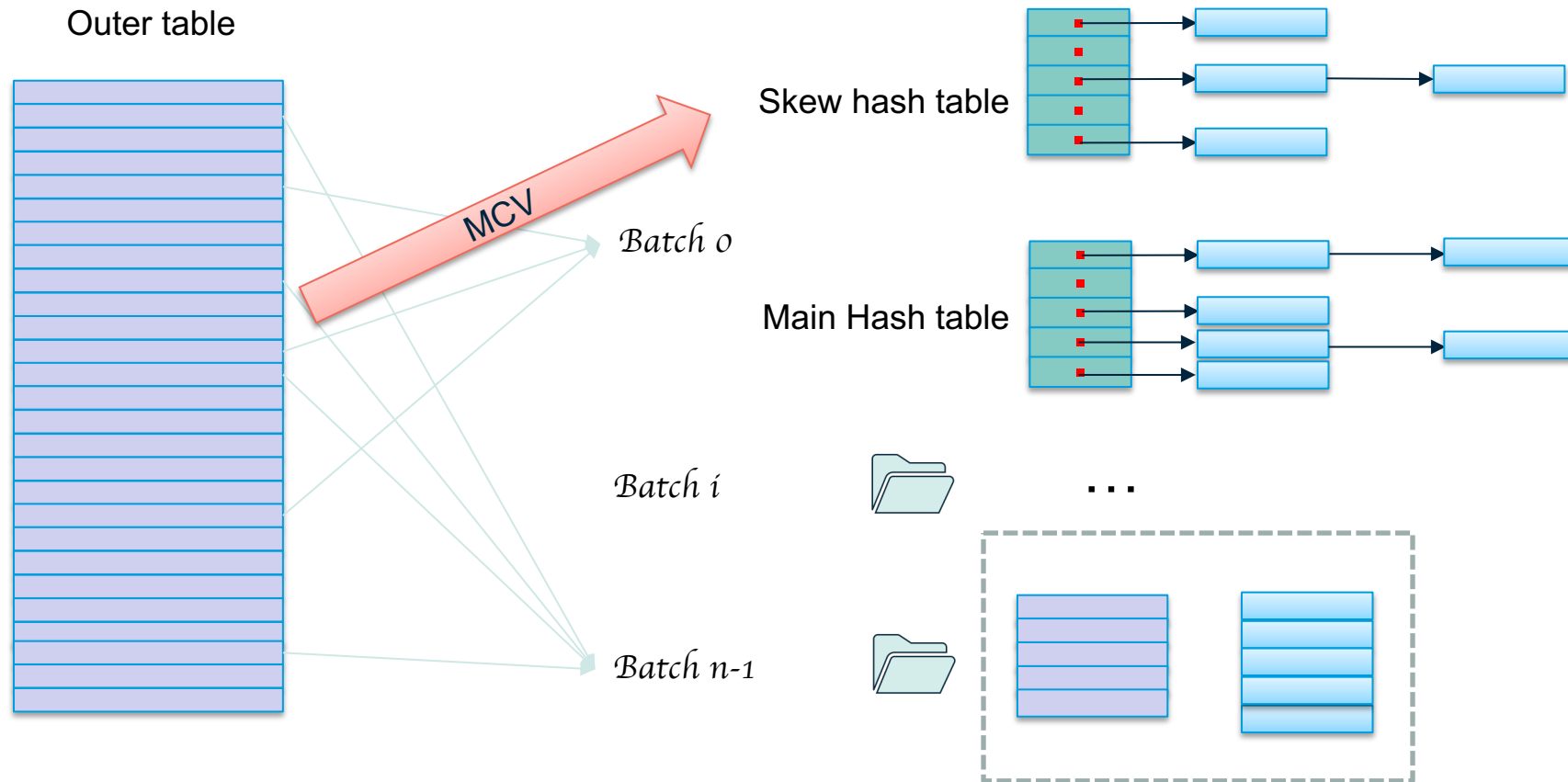
Skew hash table

1. Determine size

2. Create empty table with hash value

3. Populate skew hash table

Skew optimization: probe skew hash table



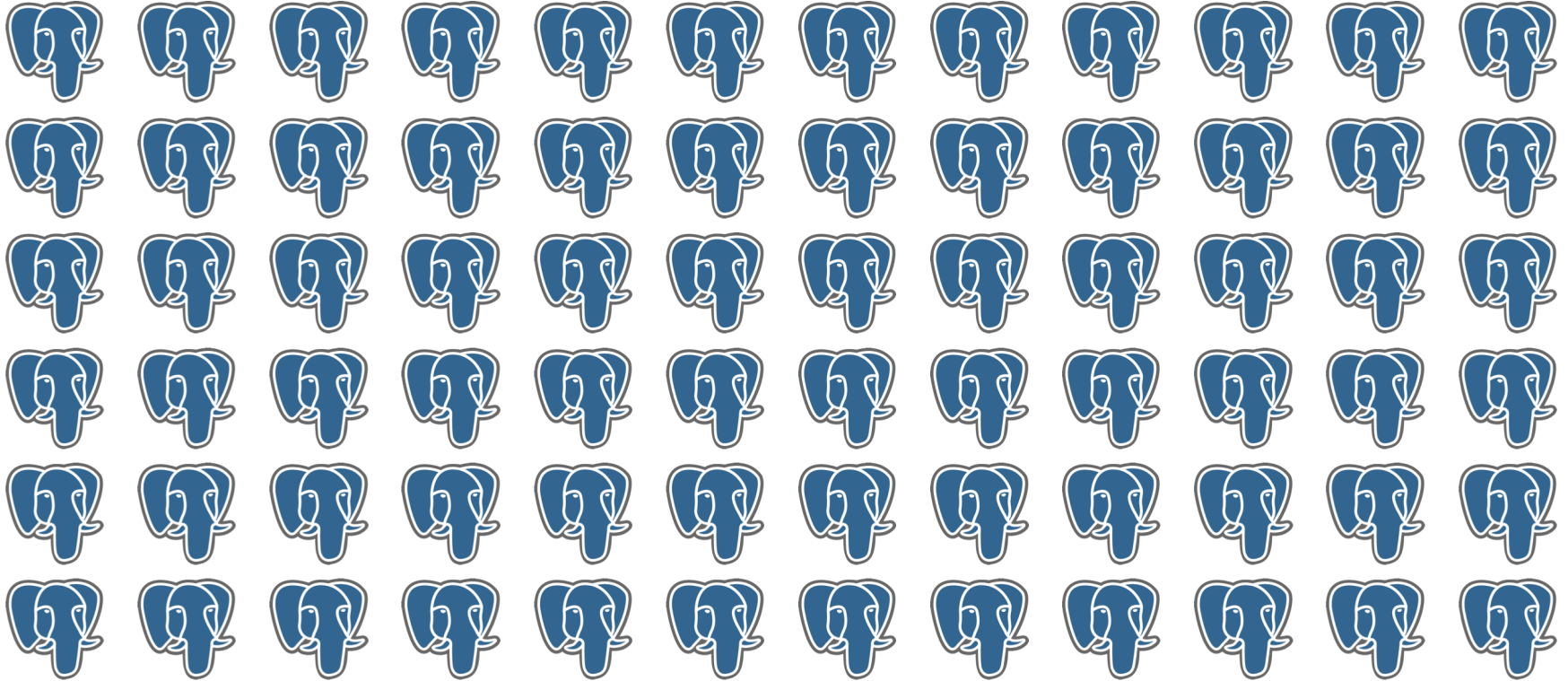
Parallel Join



A group of people in a meeting room, with a teal square highlighting a group of four people in the center. The scene is dimly lit with a blue tint. A man on the left is pointing at a whiteboard. A man on the right is standing and talking to a group of people seated at a table. A teal square highlights a group of four people in the center.

Hash join in Greenplum

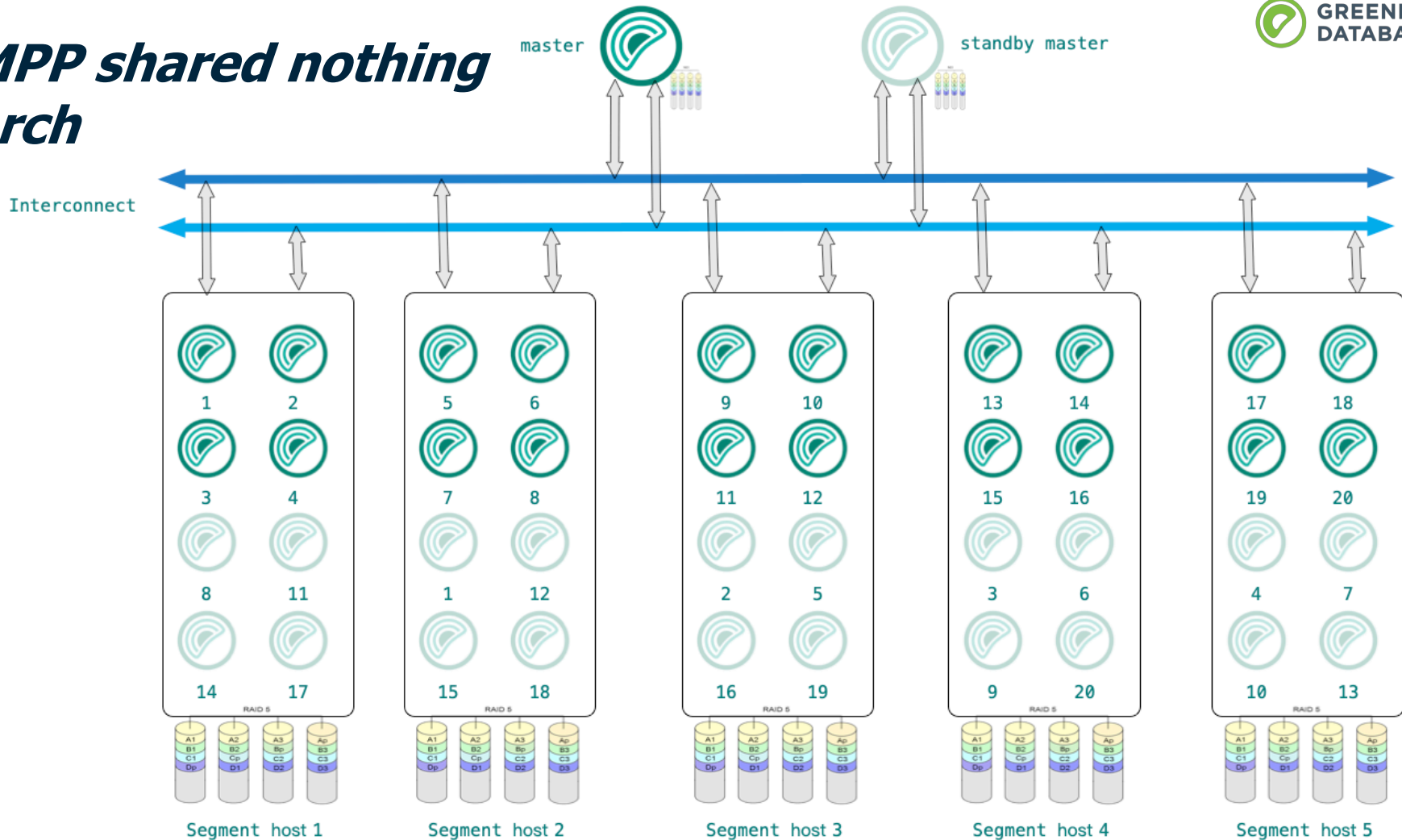
Basically many PostgreSQL nodes



A transparent distributed database with ACID



MPP shared nothing arch



Key Greenplum concepts

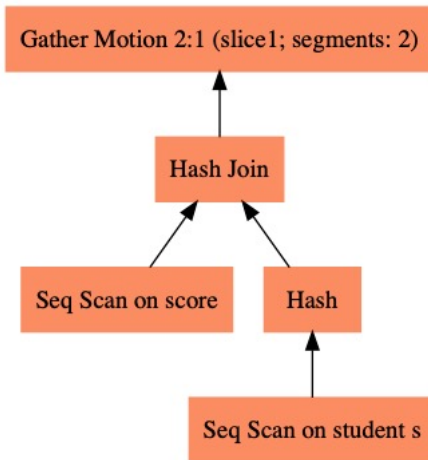
- **Distribution** policy: controls how to distribute tuples to each segments
 - Hash distribution
 - Random distribution
 - Replicated tables
 - Customized hash function
- **Motion**: transfer data between different segments
 - Gather
 - Redistribution
 - Broadcast

Hash join for tables with 'same' distribution

CREATE TABLE student(id int, name text, age int) **distributed by (id);**

CREATE TABLE score(id int, stu_id int, subject text, score int) **distributed by (stu_id);**

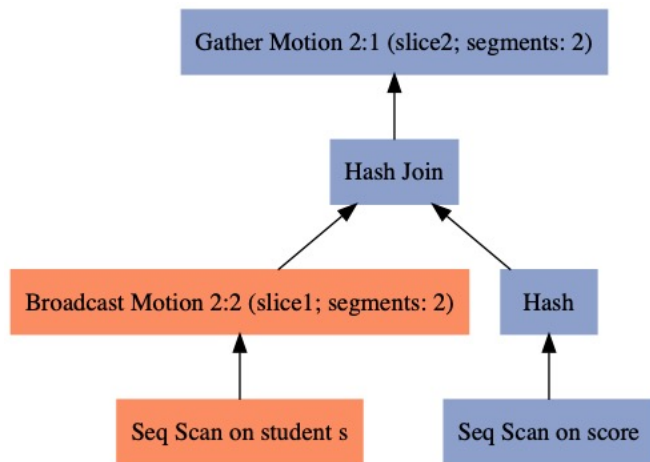
SELECT name, subject, score FROM student s INNER JOIN score ON **s.id = score.stu_id;**

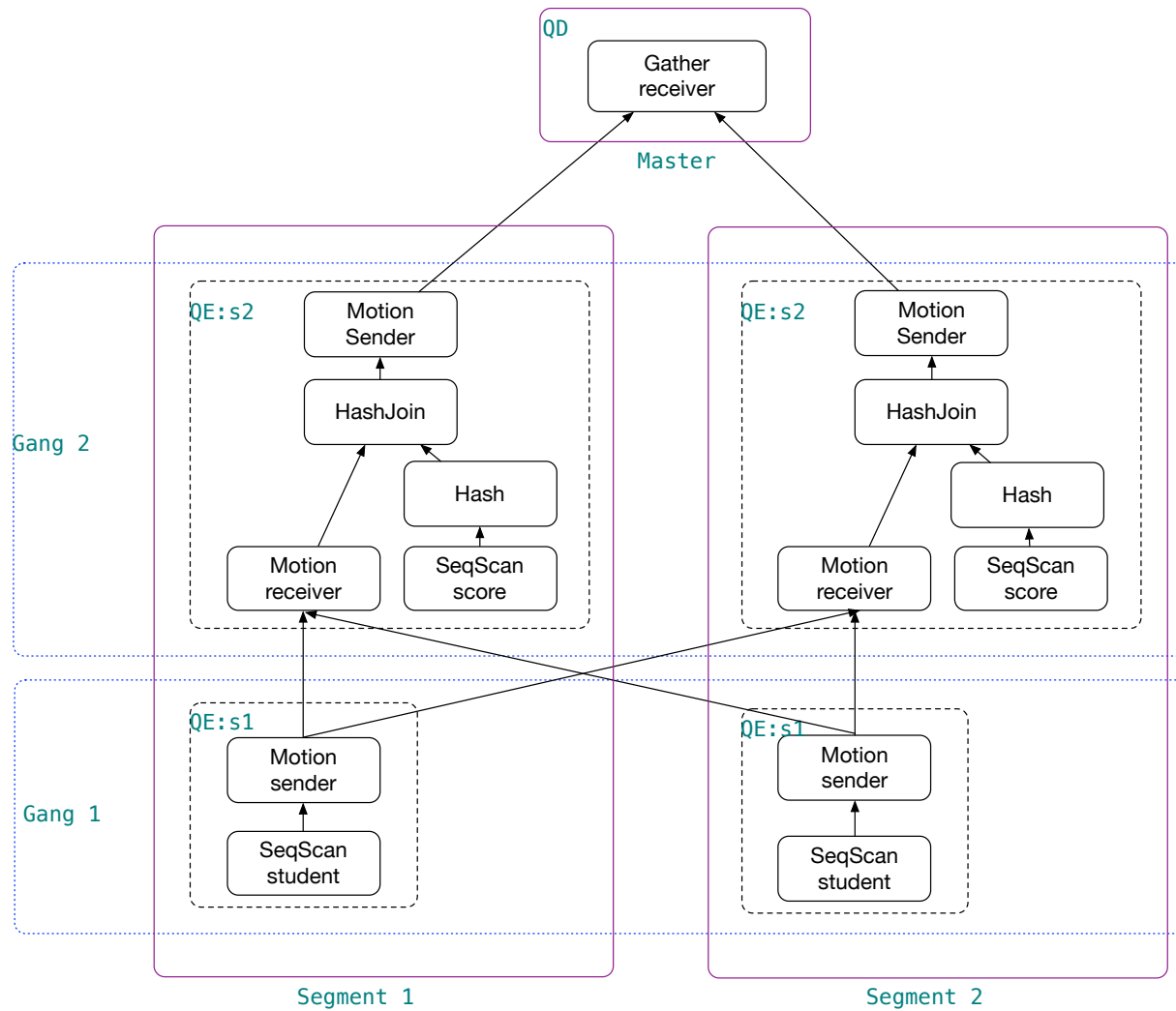


Hash join for tables with different distribution

```
CREATE TABLE student(id int, name text, age int) distributed by (id);
CREATE TABLE score(id int, stu_id int, subject text, score int) distributed by (id);
```

```
SELECT name, subject, score FROM student s INNER JOIN score ON s.id = score.stu_id;
```





Greenplum enhancements

- Batch file compression using zstd
- *Left anti semi join to optimize 'NOT IN'*

```
# explain SELECT name FROM student st WHERE id NOT IN (SELECT stu_id FROM score);  
QUERY PLAN
```

```
-----  
Gather Motion 2:1 (slice2; segments: 2) (cost=2.26..4.35 rows=4 width=5)
```

```
-> Hash Left Anti Semi (Not-In) Join (cost=2.26..4.35 rows=2 width=5)
```

```
Hash Cond: (st.id = score.stu_id)
```

```
-> Seq Scan on student st (cost=0.00..2.03 rows=2 width=9)
```

```
-> Hash (cost=2.16..2.16 rows=4 width=4)
```

```
-> Broadcast Motion 2:2 (slice1; segments: 2) (cost=0.00..2.16 rows=4 width=4)
```

```
-> Seq Scan on score (cost=0.00..2.04 rows=2 width=4)
```

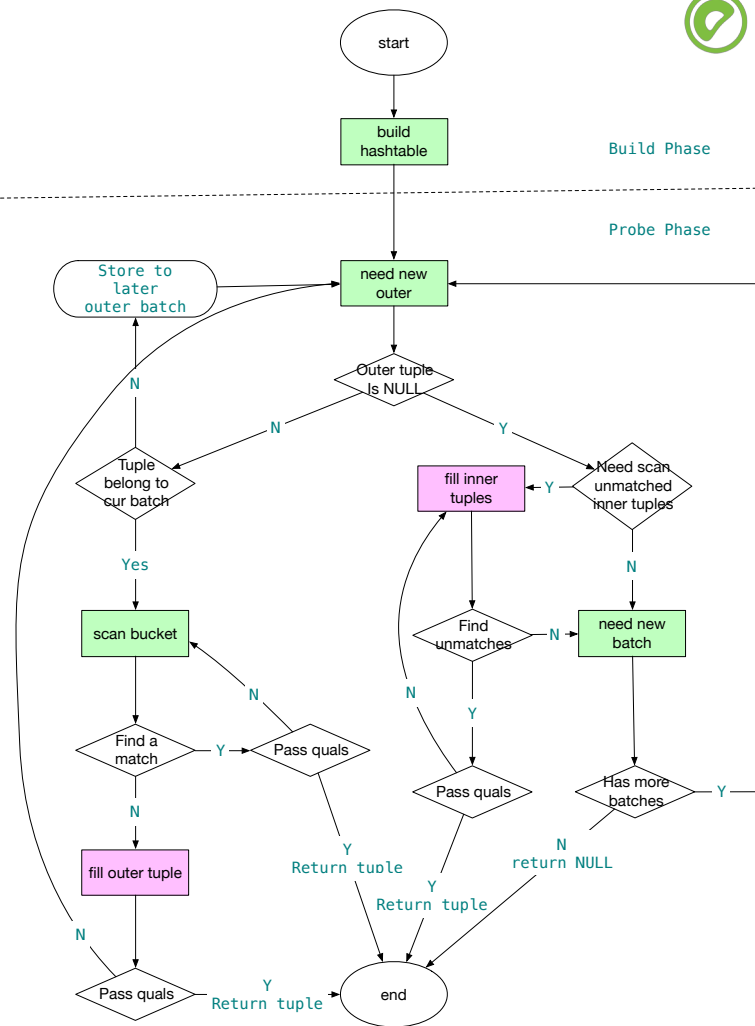
```
Optimizer: Postgres query optimizer
```



Hash join state machine

Hash join state machine

- HJ_BUILD_HASHTABLE
- HJ_NEED_NEW_OUTER
- HJ_SCAN_BUCKET
- HJ_FILL_OUTER_TUPLE
- HJ_FILL_INNER_TUPLES
- HJ_NEED_NEW_BATCH



9th PostgreSQL China Conference at Beijing



- 11/29 – 11/30 2019, Beijing
- ~500 attendees
- CFP is open
 - Send email to press@postgres.cn with your topic title and abstraction
 - Email to yyao@pivotal.io if you have any question