Part 2: Aggregation Algorithms

Basics

- For Each grouping columns, aggregate the other columns as requested
- Two kinds - Algebraic, Holistic
- Algebraic : $SUM(A \cup B) = SUM(SUM(A), SUM(B))$
- Holistic: $MEDIAN(A \cup B) \neq MEDIAN(MEDIAN(A), MEDIAN(B))$

Algebraic

- Functions supported:
  - SUM, COUNT, MIN, MAX, (PRODUCT)
  - Others, e.g. CORR, can be composed of these
- Steps: Collect all the identical groups together
  - Hash
  - Sort

Sort Based Aggregation

- Sort the table on the group by values
- While scanning the sorted table, for each row
  - Assume we are computing for each g, SUM(a), COUNT(b)
  - if the row's group by value matches current group by i.e. current.g = row.g
    - update the aggregated values: current.sum += row.a, current.count++
  - else /* there is a mismatch */
    - write out the final value for the group by value:
      (current.g, current.sum, current.count)
    - initialize the new group by entry and aggregate values
      - current.g = row.g; current.sum = row.a; current.count = 1

Original Data

supp	item	price
S6	P3	
S2	P3	100
S2	P2	20
	P1	
S7	P4	1100
S7	P1	11
S7	P2	22
S1	P1	10
S7	P3	110
S1	P2	20
S6	P1	12
S3	P4	1000
S5		
S4	P1	9
S4	P3	

Project the needed columns

supp	item	price
S6	P3	
S2	P3	100
S2	P2	20
	P1	
S7	P4	1100
S7	P1	11
S7	P2	22
S1	P1	10
S7	P3	110
S1	P2	20
S6	P1	12
S3	P4	1000
S5		
S4	P1	9
S4	P3	

item	price
P3	
P3	100
P2	20
P1	
P4	1100
P1	11
P2	22
P1	10
P3	110
P2	20
P1	12
P4	1000
P1	9
P3	

Project the needed columns

supp	item	price
S6	P3	
S2	P3	100
S2	P2	20
	P1	
S7	P4	1100
S7	P1	11
S7	P2	22
S1	P1	10
S7	P3	110
S1	P2	20
S6	P1	12
S3	P4	1000
S5		
S4	P1	9
S4	P3	

item	price
P3	
P3	100
P2	20
P1	
P4	1100
P1	11
P2	22
P1	10
P3	110
P2	20
P1	12
P4	1000
P1	9
P3	

Sort on the GROUP BY

item	price
P3	
P3	100
P2	20
P1	
P4	1100
P1	11
P2	22
P1	10
P3	110
P2	20
P1	12
P4	1000
P1	9
P3	

item	price
P1	9
P1	10
P1	11
P1	12
P1	
P2	20
P2	20
P2	22
P3	100
P3	110
P3	
P3	
P3	
P4	1000
P4	1100

Scan and Aggregate

item	price
P1	12
P1	10
P1	9
P1	11
P2	20
P2	20
P2	22
P3	100
P3	110
P3	
P3	
P4	1100
P4	1000

GROUP BY value,SUM,COUNT

Accumulator

P1,12,1

Scan and Aggregate

item	price
P1	12
P1	**10**
P1	9
P1	11
P2	20
P2	20
P2	22
P3	100
P3	110
P3	
P3	
P4	1100
P4	1000

GROUP BY value,SUM,COUNT

Accumulator

P1,22,2

Scan and Aggregate

item	price
P1	12
P1	10
P1	9
P1	11
P2	20
P2	20
P2	22
P3	100
P3	110
P3	
P3	
P4	1100
P4	1000

GROUP BY value,SUM,COUNT

Accumulator

P1,31,3

Scan and Aggregate

item	price
P1	12
P1	10
P1	9
P1	
P1	11
P2	20
P2	20
P2	22
P3	100
P3	110
P3	
P3	
P4	1100
P4	1000

GROUP BY value,SUM,COUNT

Accumulator

P1,31,3

Scan and Aggregate

item	price
P1	12
P1	10
P1	9
P1	
P1	**11**
P2	20
P2	20
P2	22
P3	100
P3	110
P3	
P3	
P4	1100
P4	1000

GROUP BY value,SUM,COUNT

Accumulator

P1,42,4

Scan and Aggregate

```
 item | price
------+-------
 P1   |    12
 P1   |    10
 P1   |     9
 P1   |
 P1   |    11
 P2   |    20
 P2   |    20
 P2   |    22
 P3   |   100
 P3   |   110
 P3   |
 P3   |
 P4   |  1100
 P4   |  1000
      |
```

GROUP BY value,SUM,COUNT

Accumulator P1,42,4

- new GROUP BY value is different from Accumulator
- Output result (for current GROUP BY)
- Accumulator reset using new row value

```
Output:    P1    42/4=10.50
```

Accumulator P2,20,1

Scan and Aggregate

item	price
P1	12
P1	10
P1	9
P1	11
P2	20
P2	**20**
P2	22
P3	100
P3	110
P3	
P3	
P4	1100
P4	1000

GROUP BY value,SUM,COUNT

Accumulator

P2,40,2

Output: P1 42/4=10.50

Scan and Aggregate

```
 item | price
------+-------
 P1   |    12
 P1   |    10
 P1   |     9
 P1   |
 P1   |    11
 P2   |    20
 P2   |    20
 P2   |    22
 P3   |   100
 P3   |   110
 P3   |
 P3   |
 P4   |  1100
 P4   |  1000
      |
```

GROUP BY value,SUM,COUNT

Accumulator

P2,62,3

Output: P1 42/4=10.50

Scan and Aggregate

item	price
P1	12
P1	10
P1	9
P1	11
P2	20
P2	20
P2	22
P3	**100**
P3	110
P3	
P3	
P4	1100
P4	1000

GROUP BY value,SUM,COUNT

Accumulator P2,62,3

- new GROUP BY value is different from Accumulator
- Output result (for current GROUP BY)
- Accumulator reset using new row value

```
Output:   P1    42/4=10.50
          P2    62/3=20.67
```

Accumulator P3,100,1

Scan and Aggregate

item	price
P1	12
P1	10
P1	9
P1	
P1	11
P2	20
P2	20
P2	22
P3	100
P3	**110**
P3	
P3	
P4	1100
P4	1000

GROUP BY value,SUM,COUNT

Accumulator

P3,210,2

Output:

P1 42/4=10.50
P2 62/3=20.67

Scan and Aggregate

item	price
P1	12
P1	10
P1	9
P1	
P1	11
P2	20
P2	20
P2	22
P3	100
P3	**110**
P3	
P3	
P4	1100
P4	1000

GROUP BY value,SUM,COUNT

Accumulator

P3,210,2

Output:

P1 42/4=10.50
P2 62/3=20.67

Scan and Aggregate

```
 item | price
------+-------
 P1   |    12
 P1   |    10
 P1   |     9
 P1   |
 P1   |    11
 P2   |    20
 P2   |    20
 P2   |    22
 P3   |   100
 P3   |   110
 P3   |
 P3   |
 P4   |  1100
 P4   |  1000
      |
```

GROUP BY value,SUM,COUNT

Accumulator

P3,210,2

Output:

```
P1    42/4=10.50
P2    62/3=20.67
```

Scan and Aggregate

```
 item | price
------+-------
 P1   |    12
 P1   |    10
 P1   |     9
 P1   |
 P1   |    11
 P2   |    20
 P2   |    20
 P2   |    22
 P3   |   100
 P3   |   110
 P3   |
 P3   |
 P4   |  1100
 P4   |  1000
      |
```

GROUP BY value,SUM,COUNT

Accumulator

P3,210,2

Output:

```
P1    42/4=10.50
P2    62/3=20.67
```

Scan and Aggregate

```
 item | price
------+-------
 P1   |    12
 P1   |    10
 P1   |     9
 P1   |
 P1   |    11
 P2   |    20
 P2   |    20
 P2   |    22
 P3   |   100
 P3   |   110
 P3   |
 P3   |
 P3   |
 P4   |  1100
 P4   |  1000
      |
```

GROUP BY value,SUM,COUNT

Accumulator P3,210,2

- new GROUP BY value is different from Accumulator
- Output result (for current GROUP BY)
- Accumulator reset using new row value

Output:

```
P1    42/4=10.50
P2    62/3=20.67
P3   210/2=100.50
```

Accumulator P4,1100,1

Scan and Aggregate

```
 item | price
------+-------
 P1   |    12
 P1   |    10
 P1   |     9
 P1   |
 P1   |    11
 P2   |    20
 P2   |    20
 P2   |    22
 P3   |   100
 P3   |   110
 P3   |
 P3   |
 P4   |  1100
 P4   |  1000
      |
```

GROUP BY value,SUM,COUNT

Accumulator

P4,2100,2

Output:

```
P1    42/4=10.50
P2    62/3=20.67
P3   210/2=100.50
```

Scan and Aggregate

item	price
P1	12
P1	10
P1	9
P1	
P1	11
P2	20
P2	20
P2	22
P3	100
P3	110
P3	
P3	
P4	1100
P4	1000

GROUP BY value,SUM,COUNT

Accumulator P4,2100,2

- new GROUP BY value is different from Accumulator
- Output result (for current GROUP BY)
- Accumulator reset using new row value

Output:

```
P1    42/4=10.50
P2    62/3=20.67
P3   210/2=100.50
P4  2100/2=1050.0
```

Accumulator Null,Null,Null

Scan and Aggregate

```
 item | price
------+-------
 P1   |    12
 P1   |    10
 P1   |     9
 P1   |
 P1   |    11
 P2   |    20
 P2   |    20
 P2   |    22
 P3   |   100
 P3   |   110
 P3   |
 P3   |
 P4   |  1100
 P4   |  1000
      |
```

GROUP BY value,SUM,COUNT

Accumulator Null,Null,Null

- End of Data
- Output result (for current GROUP BY)
- END

Final Output:

```
P1    42/4=10.50
P2    62/3=20.67
P3   210/2=100.50
P4  2100/2=1050.0
Null      =Null
```

Sort Based Aggregation

- Fairly straightforward
- Have to handle NULL values
  - ignore NULL row value, for all aggregates except COUNT(*)
- Special SQL case: empty table
  - For GROUP BY aggregate, no rows returned
  - If no GROUP BY, return 1 row with NULLs for all except COUNT(*) and 0 for COUNT(*)
- O(N log N) sort cost + Scan
- Can we do better?

Hash Based Aggregation

- Create a hash table to hold the aggregate values per unique group by value
  - How big?
- While scanning the table, for each row
  - if there is already an entry for the group by value
    - update the aggregated values
  - else
    - add a new group by value entry to the hash table

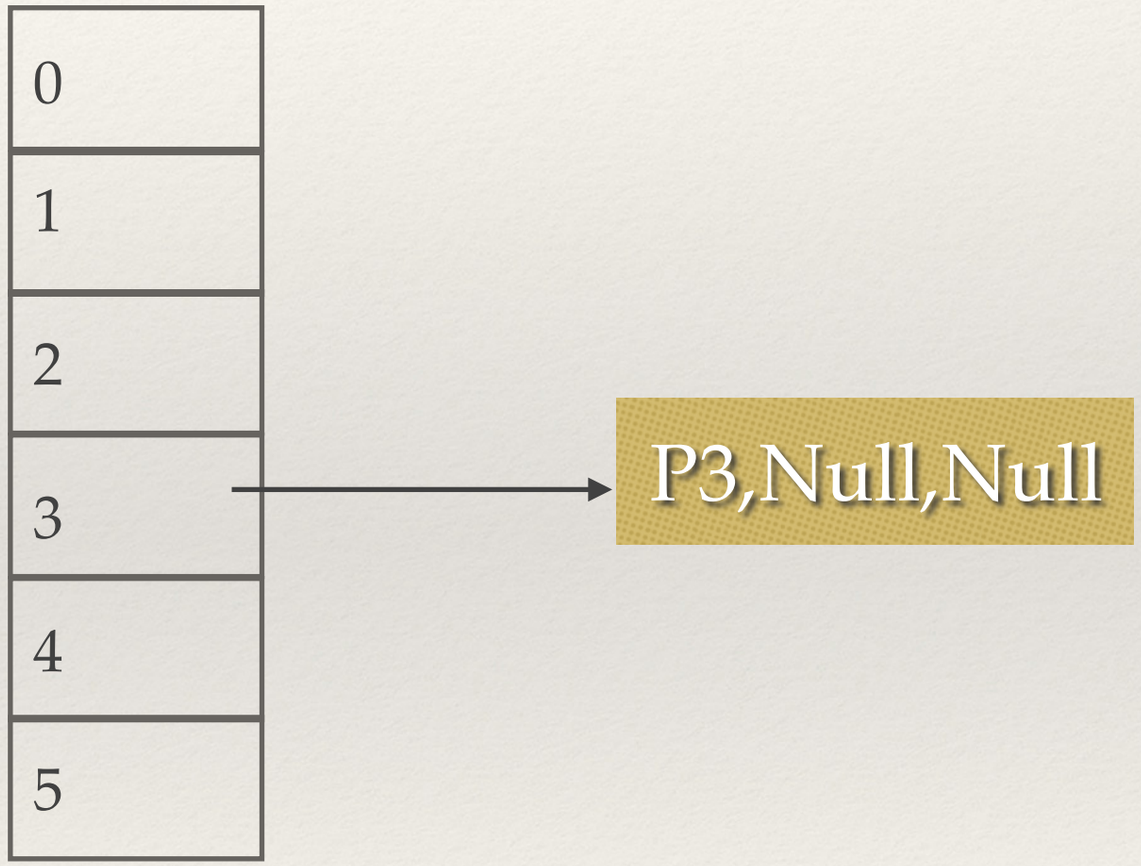
Original Data

supp	item	price
S6	P3	
S2	P3	100
S2	P2	20
	P1	
S7	P4	1100
S7	P1	11
S7	P2	22
S1	P1	10
S7	P3	110
S1	P2	20
S6	P1	12
S3	P4	1000
S5		
S4	P1	9
S4	P3	

AVG(Price) GROUP BY Item

supp	item	price
S6	**P3**	
S2	P3	100
S2	P2	20
	P1	
S7	P4	1100
S7	P1	11
S7	P2	22
S1	P1	10
S7	P3	110
S1	P2	20
S6	P1	12
S3	P4	1000
S5		
S4	P1	9
S4	P3	

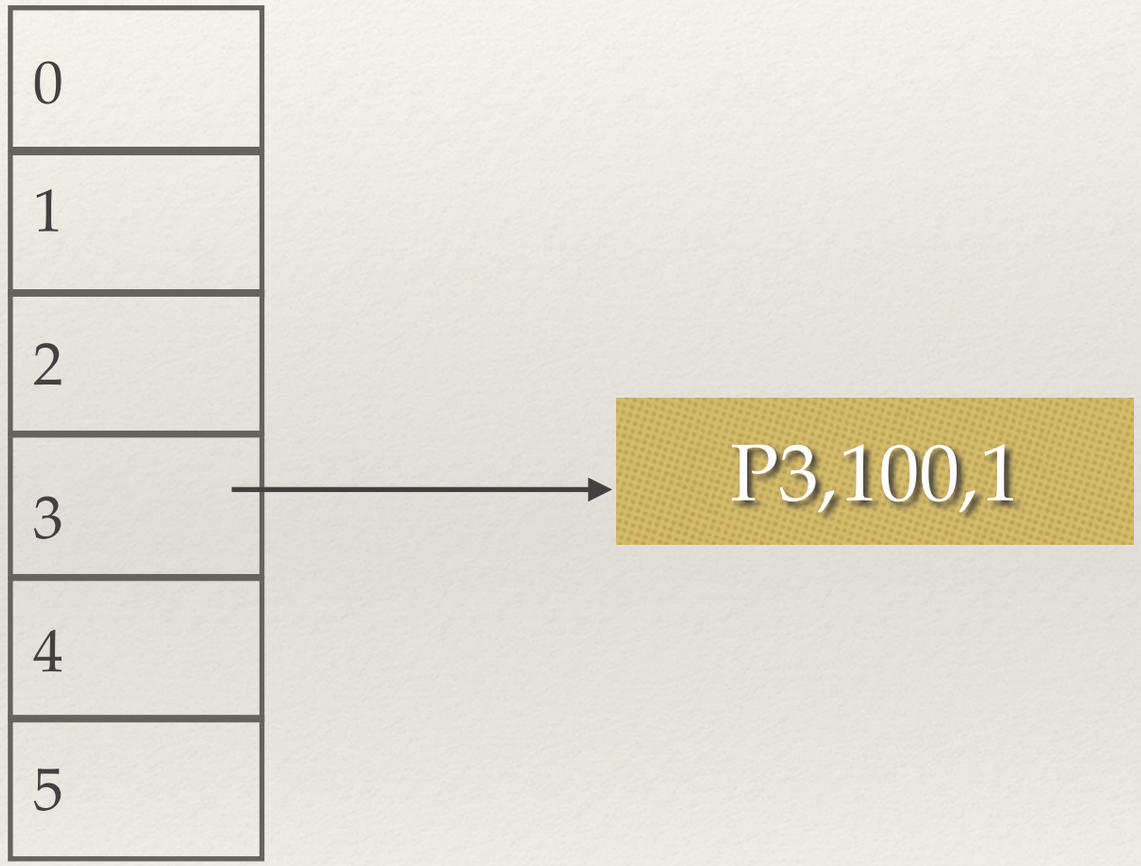
GROUP BY value,SUM,COUNT



AVG(Price) GROUP BY Item

supp	item	price
S6	P3	
S2	**P3**	**100**
S2	P2	20
	P1	
S7	P4	1100
S7	P1	11
S7	P1	11
S7	P2	22
S1	P1	10
S7	P3	110
S1	P2	20
S6	P1	12
S3	P4	1000
S5		
S4	P1	9
S4	P3	

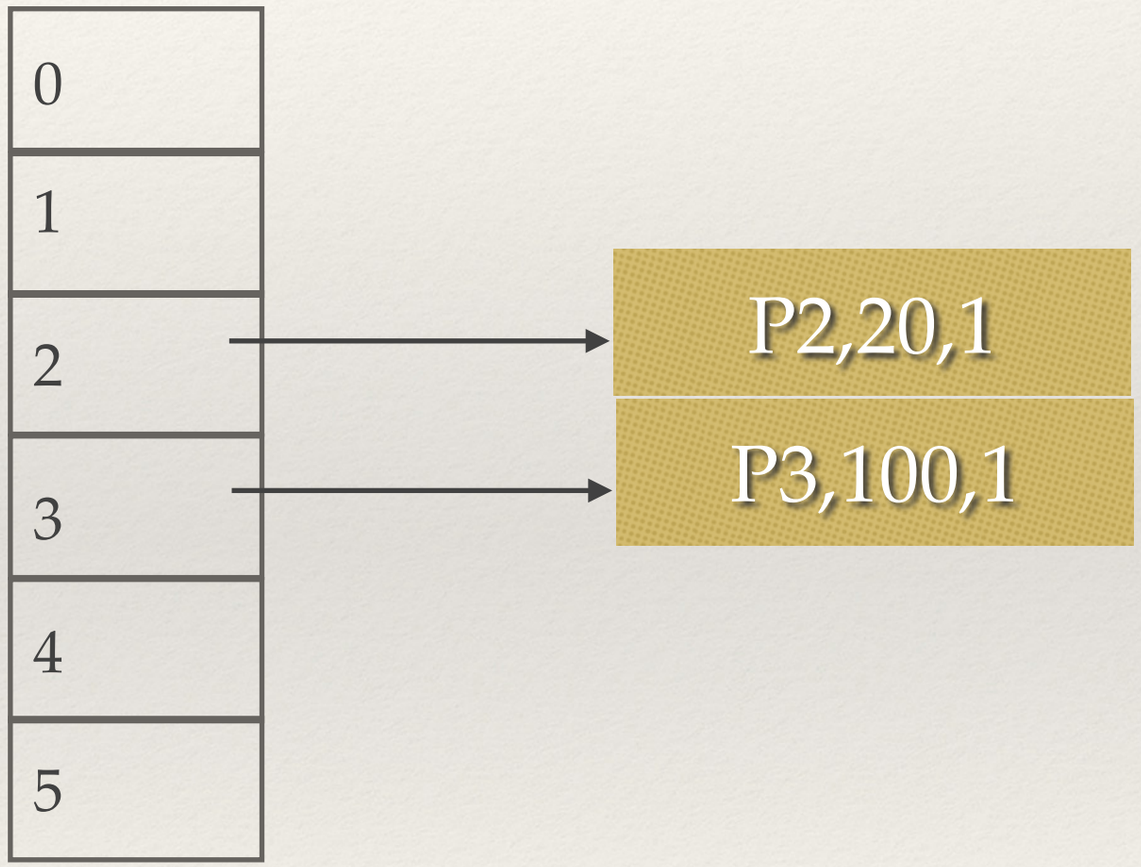
GROUP BY value,SUM,COUNT



AVG(Price) GROUP BY Item

supp	item	price
S6	P3	
S2	P3	100
S2	**P2**	**20**
	P1	
S7	P4	1100
S7	P1	11
S7	P2	22
S1	P1	10
S7	P3	110
S1	P2	20
S6	P1	12
S3	P4	1000
S5		
S4	P1	9
S4	P3	

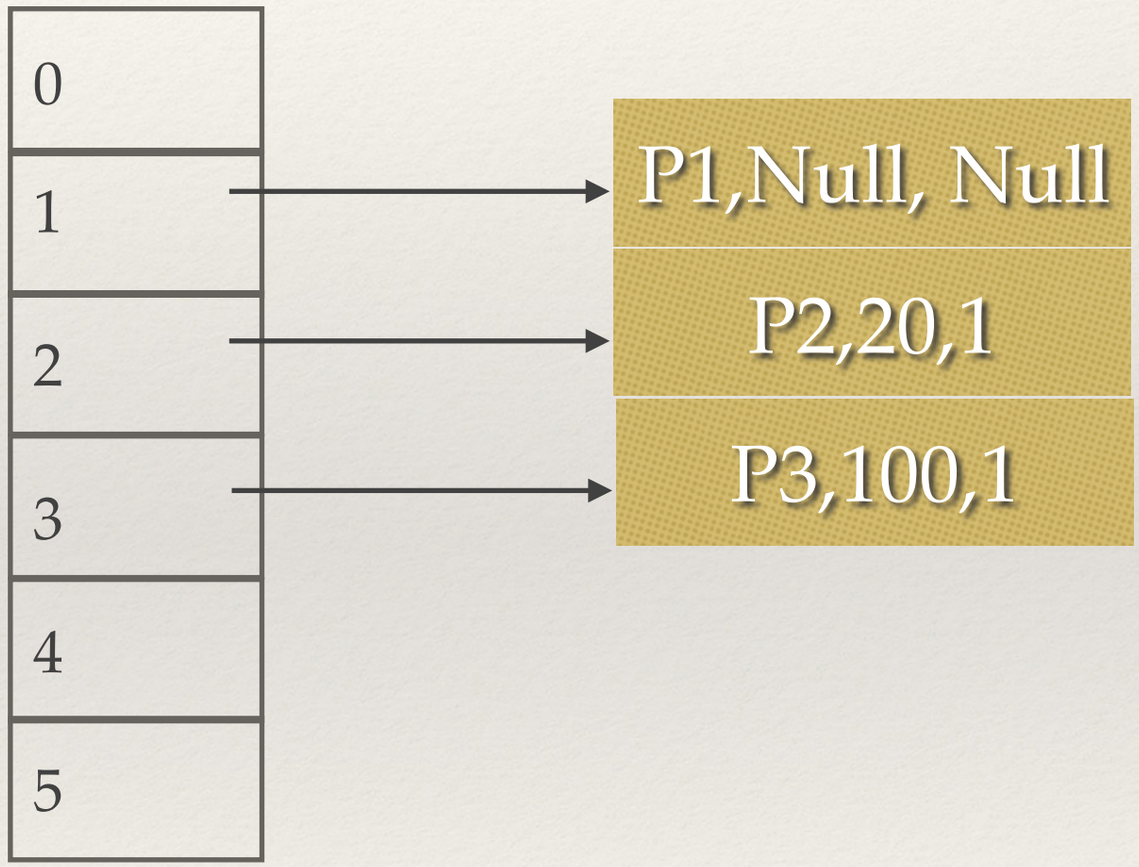
GROUP BY value,SUM,COUNT



AVG(Price) GROUP BY Item

supp	item	price
S6	P3	
S2	P3	100
S2	P2	20
	P1	
S7	P4	1100
S7	P1	11
S7	P2	22
S1	P1	10
S7	P3	110
S1	P2	20
S6	P1	12
S3	P4	1000
S5		
S4	P1	9
S4	P3	

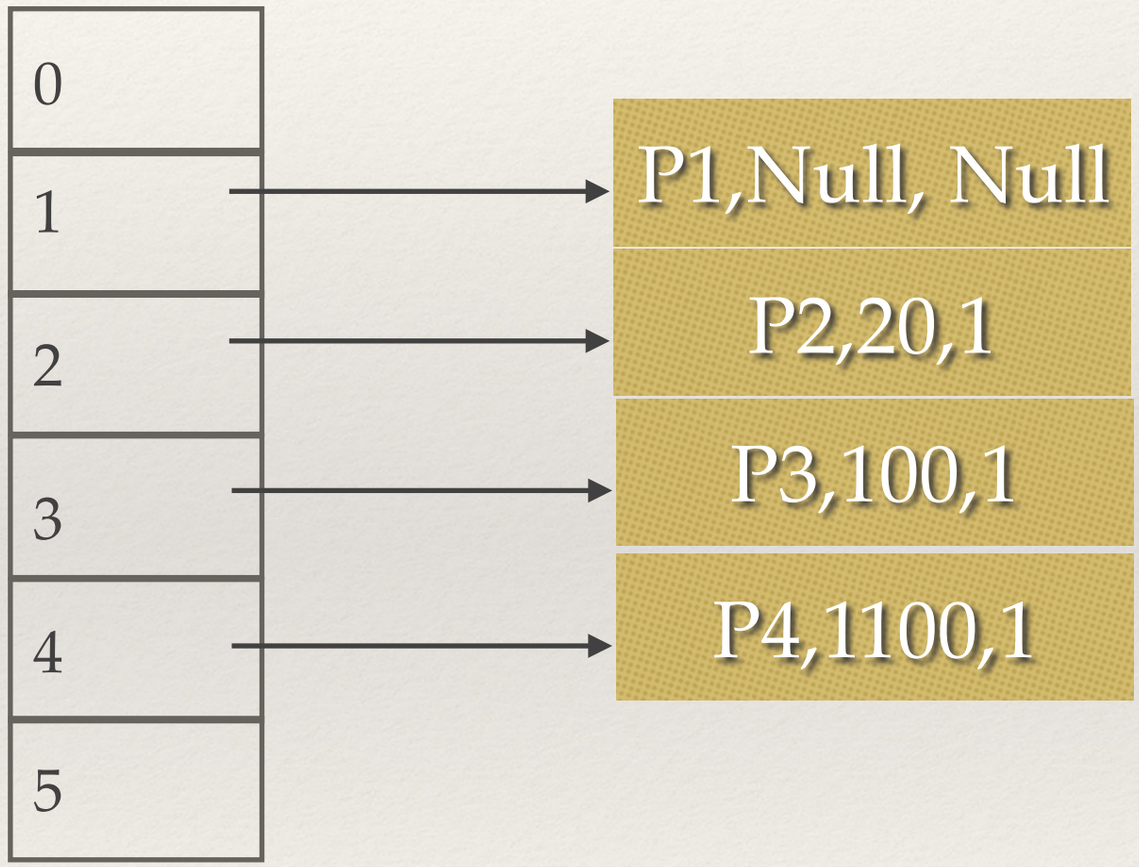
GROUP BY value,SUM,COUNT



AVG(Price) GROUP BY Item

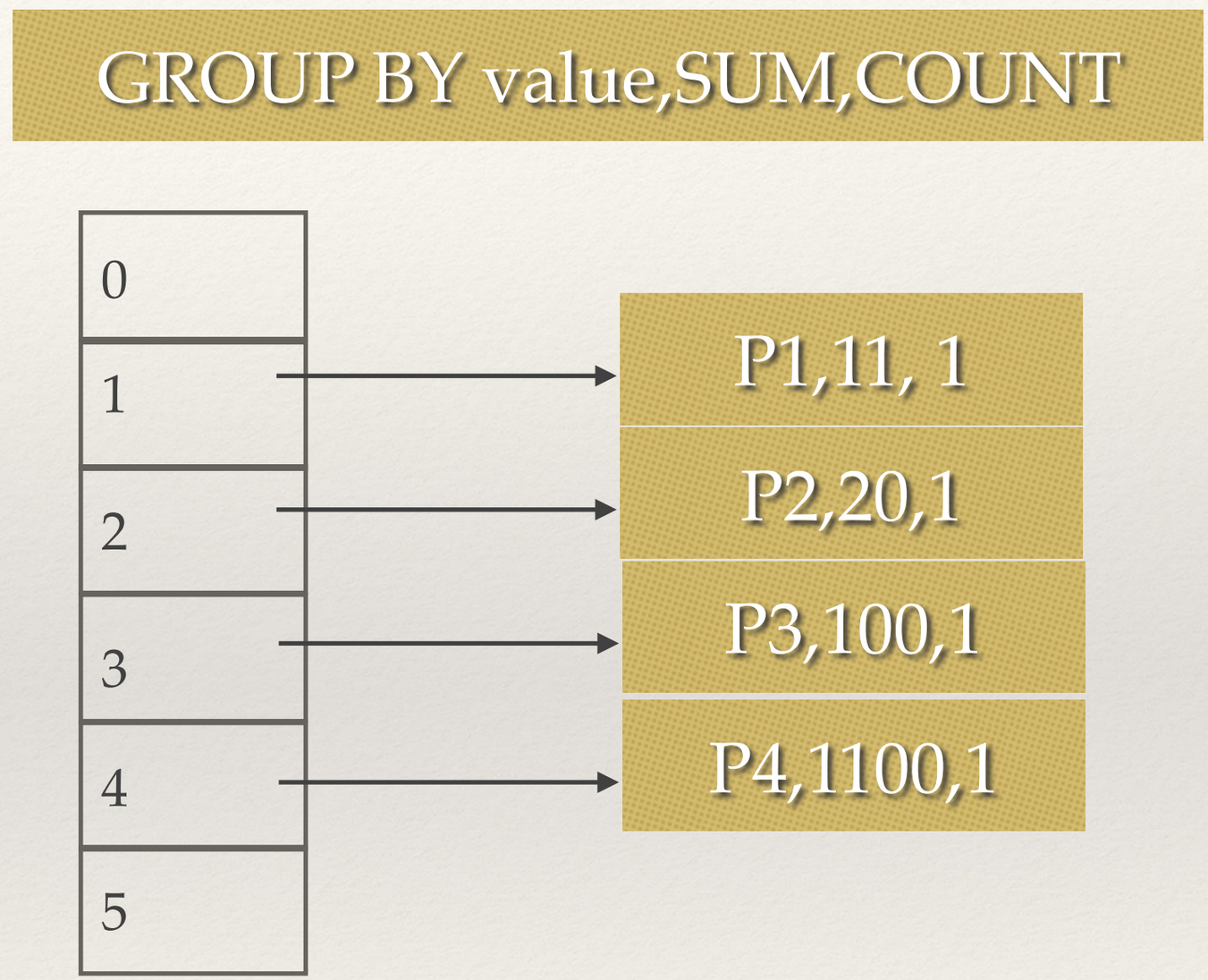
supp	item	price
S6	P3	
S2	P3	100
S2	P2	20
	P1	
S7	**P4**	**1100**
S7	P1	11
S7	P2	22
S1	P1	10
S7	P3	110
S1	P2	20
S6	P1	12
S3	P4	1000
S5		
S4	P1	9
S4	P3	

GROUP BY value,SUM,COUNT



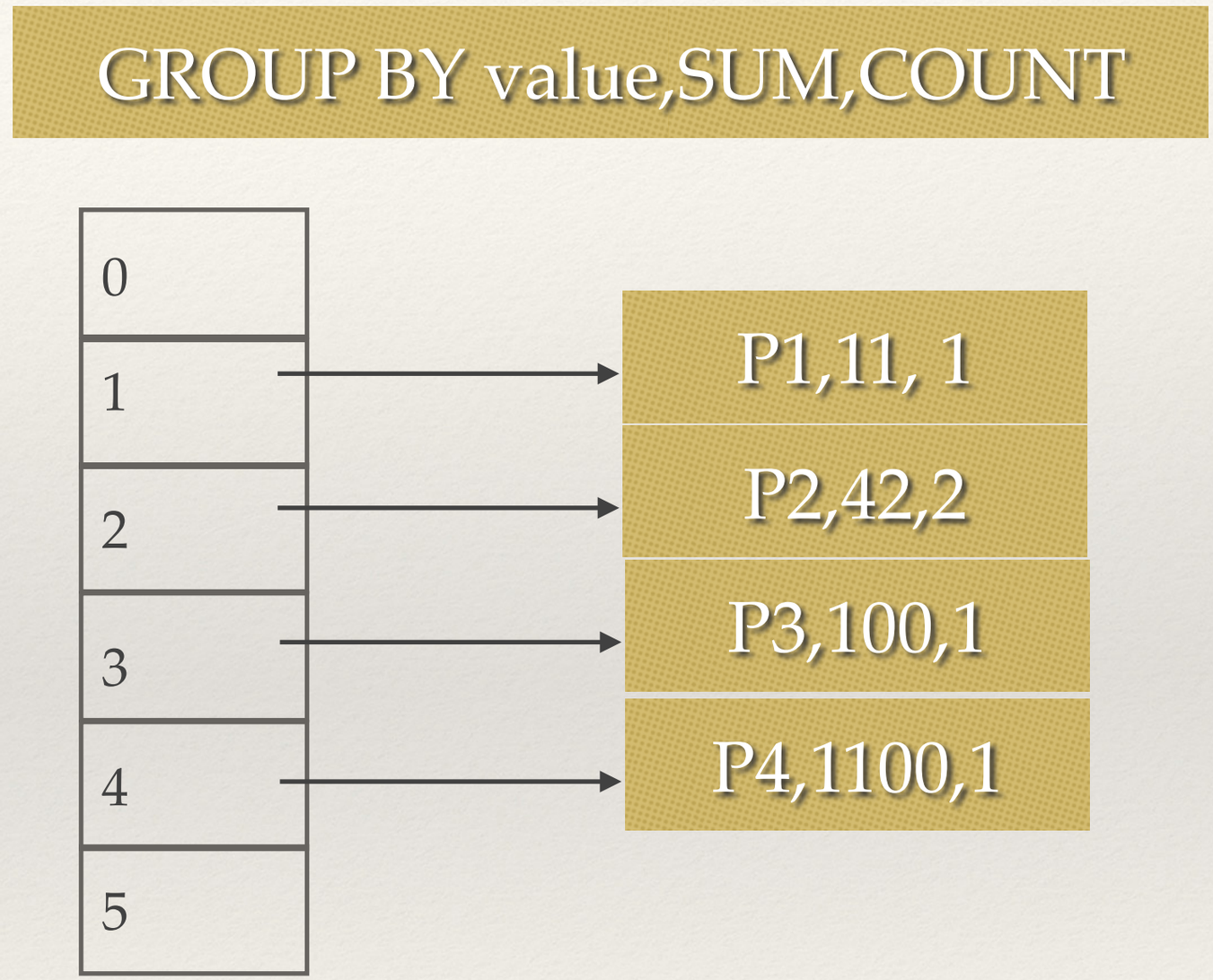
AVG(Price) GROUP BY Item

supp	item	price
S6	P3	
S2	P3	100
S2	P2	20
	P1	
S7	P4	1100
S7	**P1**	**11**
S7	P2	22
S1	P1	10
S7	P3	110
S1	P2	20
S6	P1	12
S3	P4	1000
S5		
S4	P1	9
S4	P3	



AVG(Price) GROUP BY Item

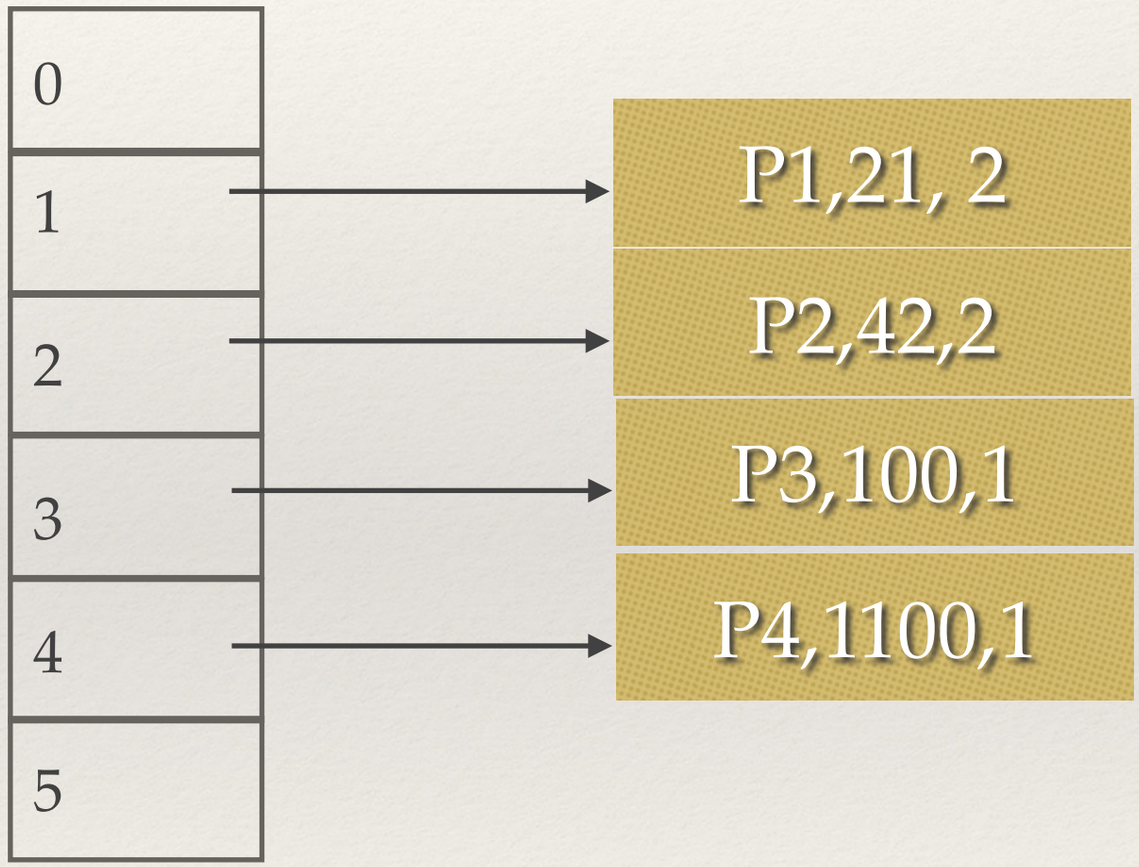
supp	item	price
S6	P3	
S2	P3	100
S2	P2	20
	P1	
S7	P4	1100
S7	P1	11
S7	**P2**	**22**
S1	P1	10
S7	P3	110
S1	P2	20
S6	P1	12
S3	P4	1000
S5		
S4	P1	9
S4	P3	



AVG(Price) GROUP BY Item

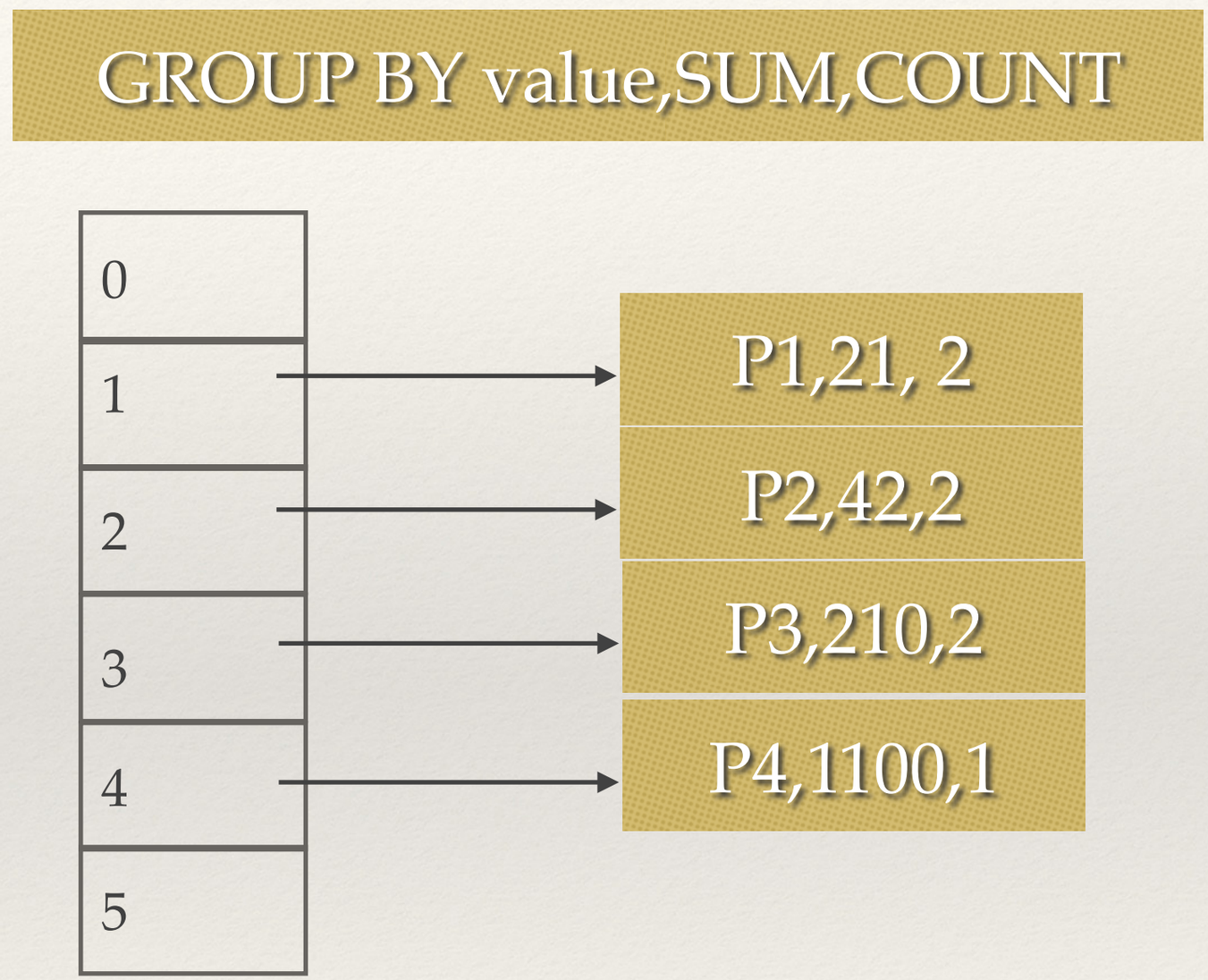
supp	item	price
S6	P3	
S2	P3	100
S2	P2	20
	P1	
S7	P4	1100
S7	P1	11
S7	P2	22
S1	**P1**	**10**
S7	P3	110
S1	P2	20
S6	P1	12
S3	P4	1000
S5		
S4	P1	9
S4	P3	

GROUP BY value,SUM,COUNT



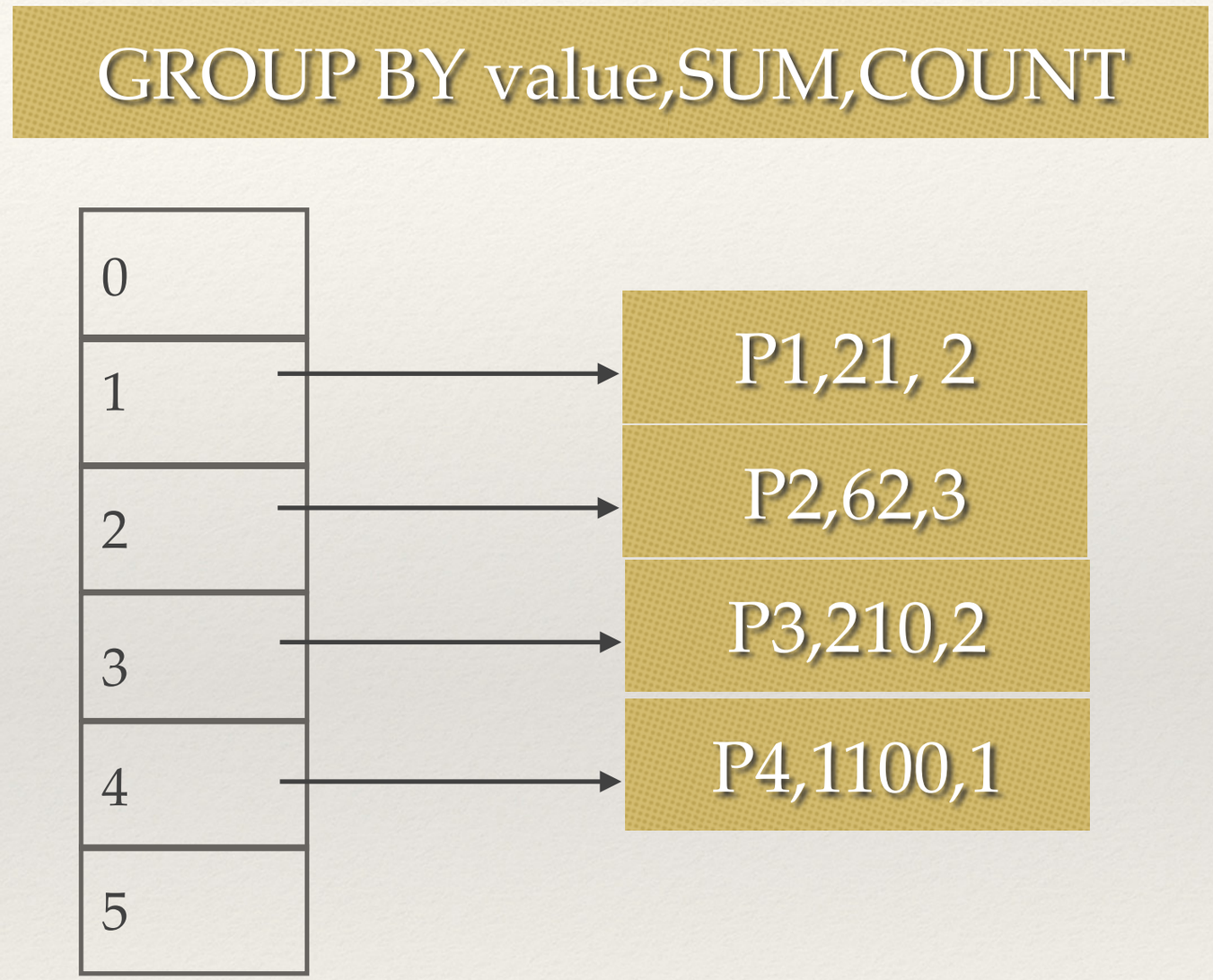
AVG(Price) GROUP BY Item

supp	item	price
S6	P3	
S2	P3	100
S2	P2	20
	P1	
S7	P4	1100
S7	P1	11
S7	P2	22
S1	P1	10
S7	**P3**	**110**
S1	P2	20
S6	P1	12
S3	P4	1000
S5		
S4	P1	9
S4	P3	



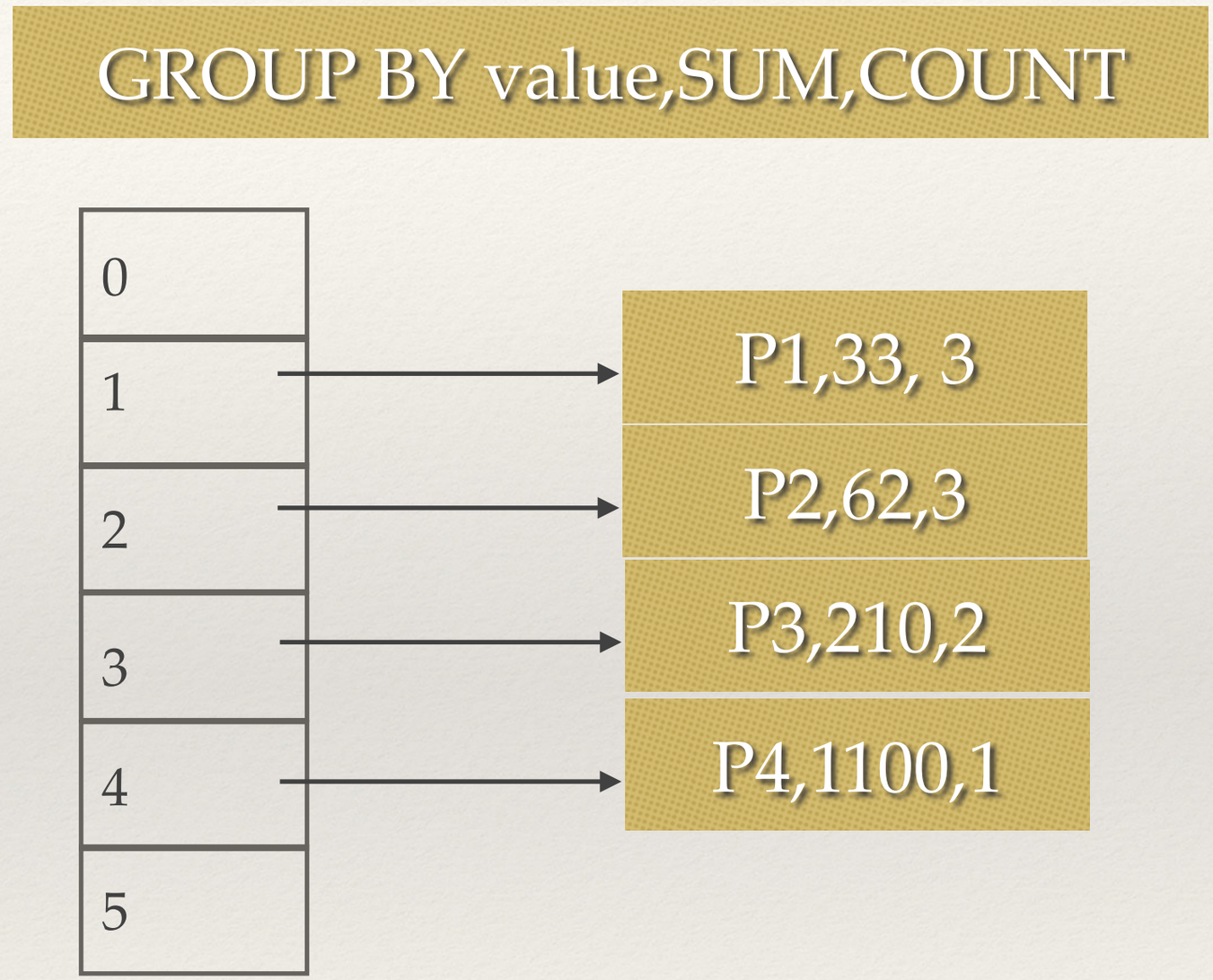
AVG(Price) GROUP BY Item

supp	item	price
S6	P3	
S2	P3	100
S2	P2	20
	P1	
S7	P4	1100
S7	P1	11
S7	P2	22
S1	P1	10
S7	P3	110
S1	**P2**	**20**
S6	P1	12
S3	P4	1000
S5		
S4	P1	9
S4	P3	



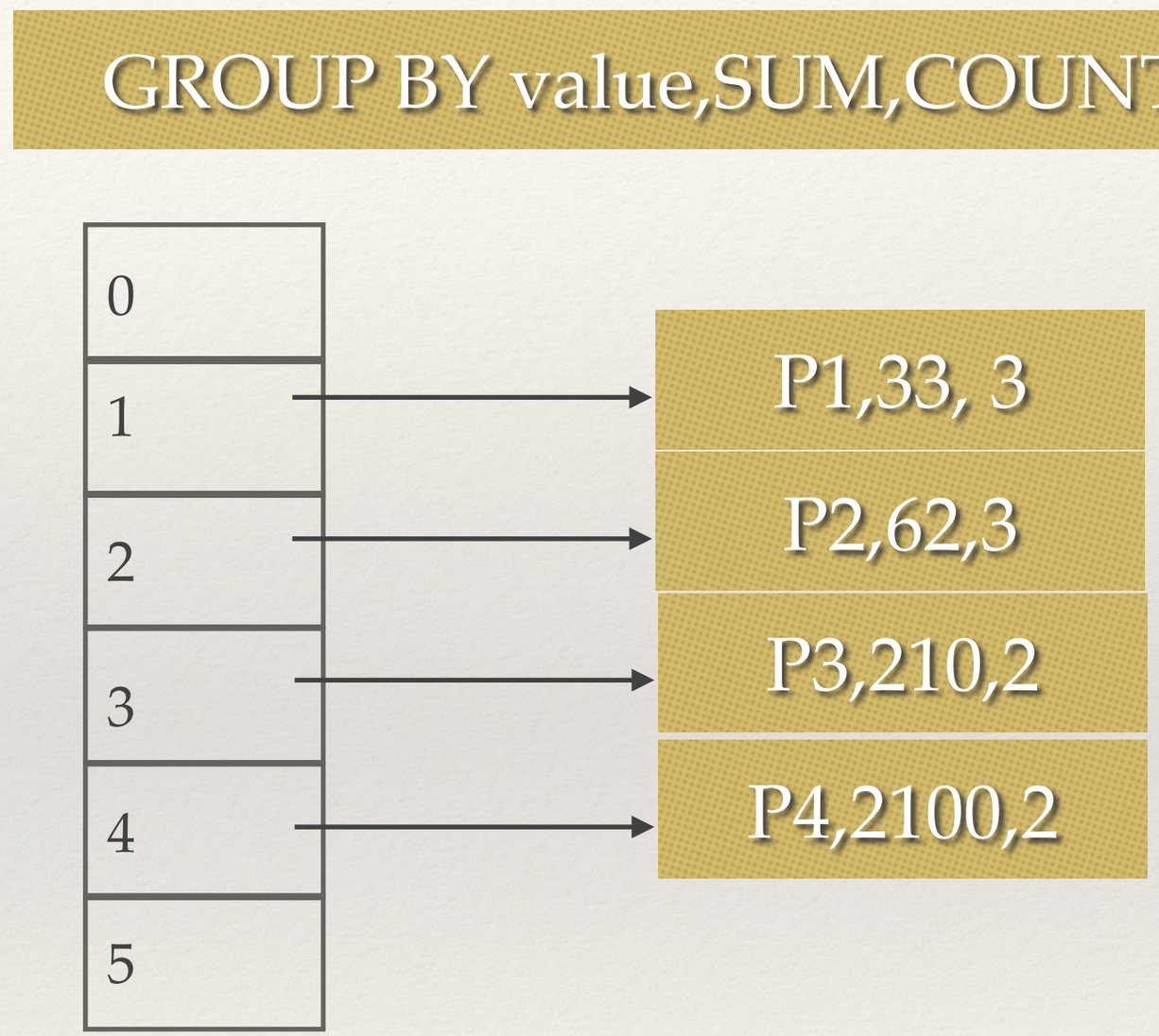
AVG(Price) GROUP BY Item

supp	item	price
S6	P3	
S2	P3	100
S2	P2	20
	P1	
S7	P4	1100
S7	P1	11
S7	P2	22
S1	P1	10
S7	P3	110
S1	P2	20
S6	**P1**	**12**
S3	P4	1000
S5		
S4	P1	9
S4	P3	



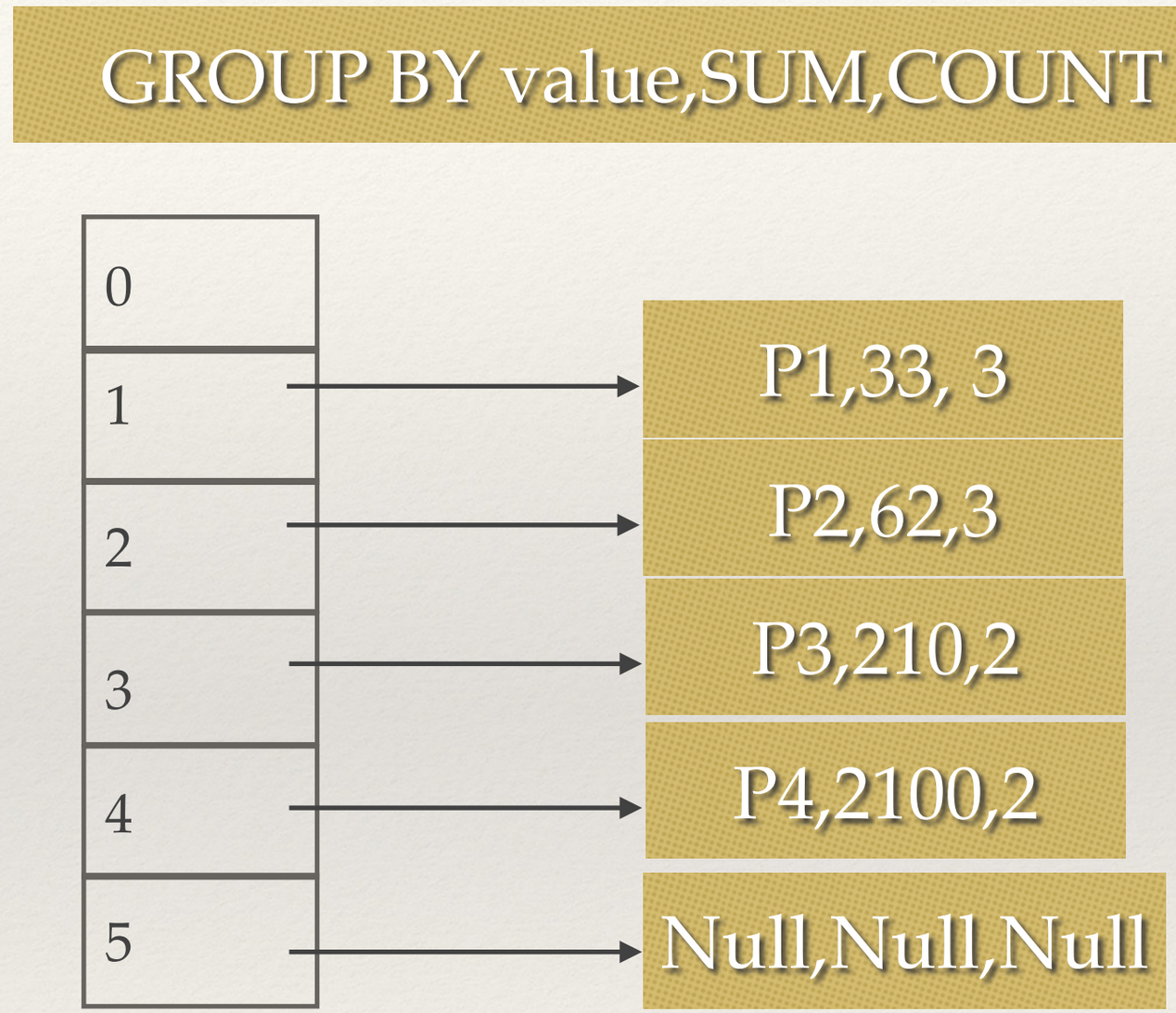
AVG(Price) GROUP BY Item

supp	item	price
S6	P3	
S2	P3	100
S2	P2	20
	P1	
S7	P4	1100
S7	P1	11
S7	P2	22
S1	P1	10
S7	P3	110
S1	P2	20
S6	P1	12
S3	**P4**	**1000**
S5		
S4	P1	9
S4	P3	



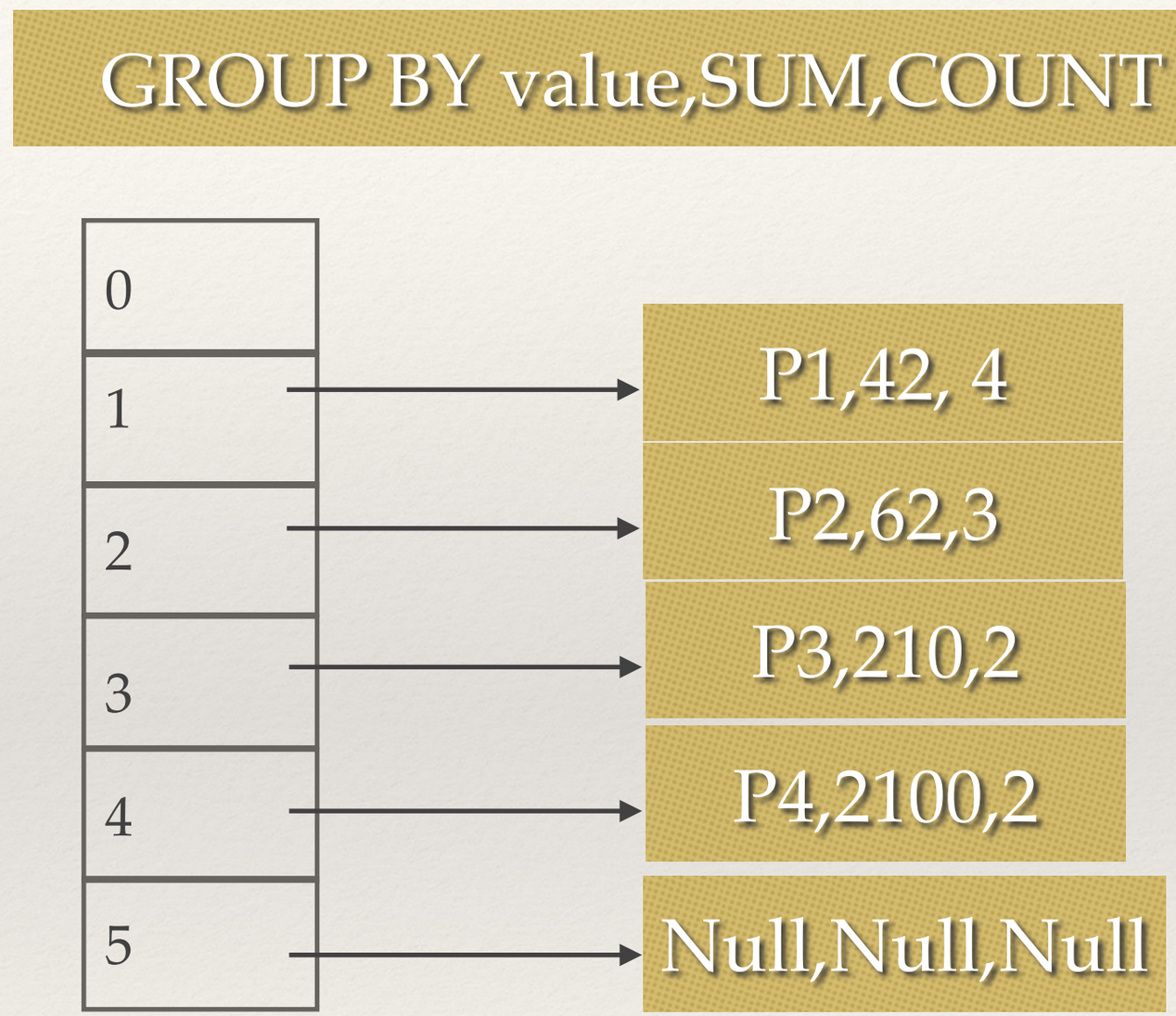
AVG(Price) GROUP BY Item

supp	item	price
S6	P3	
S2	P3	100
S2	P2	20
	P1	
S7	P4	1100
S7	P1	11
S7	P2	22
S1	P1	10
S7	P3	110
S1	P2	20
S6	P1	12
S3	P4	1000
S5		
S4	P1	9
S4	P3	



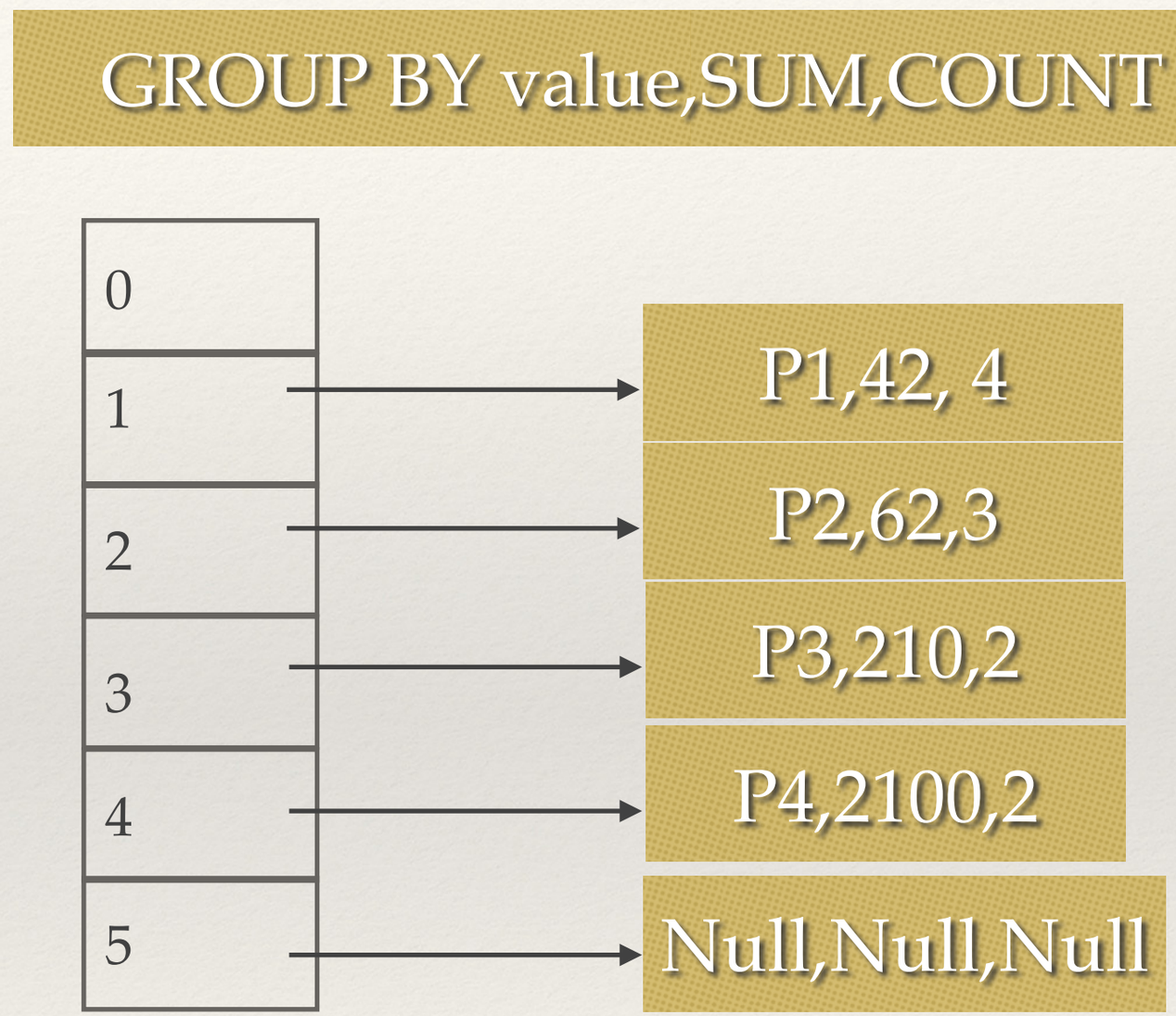
AVG(Price) GROUP BY Item

supp	item	price
S6	P3	
S2	P3	100
S2	P2	20
	P1	
S7	P4	1100
S7	P1	11
S7	P2	22
S1	P1	10
S7	P3	110
S1	P2	20
S6	P1	12
S3	P4	1000
S5		
S4	**P1**	**9**
S4	P3	



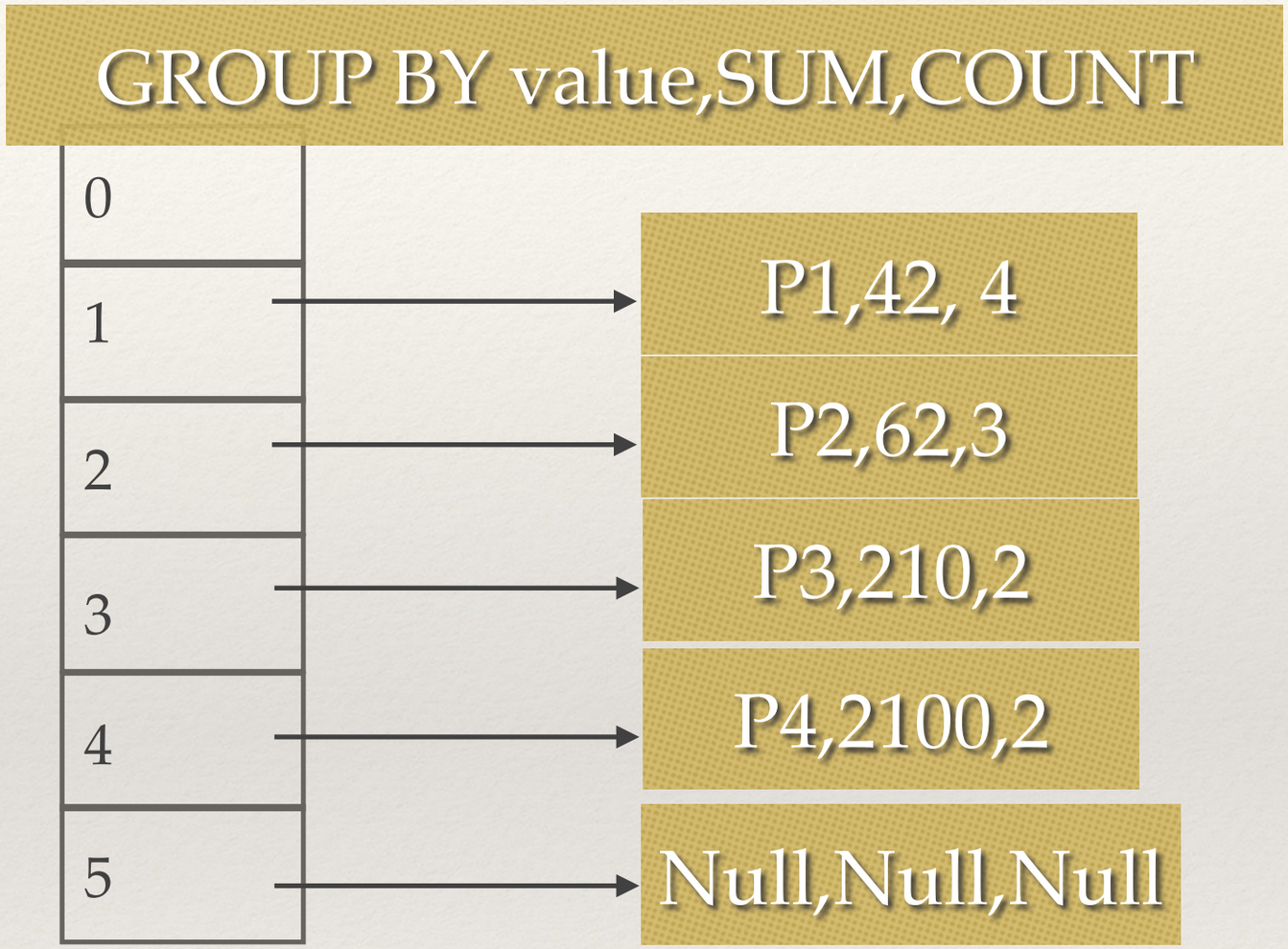
AVG(Price) GROUP BY Item

supp	item	price
S6	P3	
S2	P3	100
S2	P2	20
	P1	
S7	P4	1100
S7	P1	11
S7	P2	22
S1	P1	10
S7	P3	110
S1	P2	20
S6	P1	12
S3	P4	1000
S5		
S4	P1	9
S4	**P3**	



AVG(Price) GROUP BY Item

supp	item	price
S6	P3	
S2	P3	100
S2	P2	20
	P1	
S7	P4	1100
S7	P1	11
S7	P2	22
S1	P1	10
S7	P3	110
S1	P2	20
S6	P1	12
S3	P4	1000
S5		
S4	P1	9
S4	P3	




```
P1     42/4=10.50
P2     62/3=20.67
P3    210/2=105.0
P4   2100/2=1050.0
Null       =Null
```

Hash Based Aggregation

- If no memory issues
  - O(N): scan the table once
  - Much faster than sort based aggregation
- Unfortunately, not always the case

Handling Memory Overflow

- Hash Table (cache) full or out of memory because number of unique group by values is too high
- Option 1
  - Flush the current partially aggregated results and continue
  - Aggregate the partially aggregated results using sort

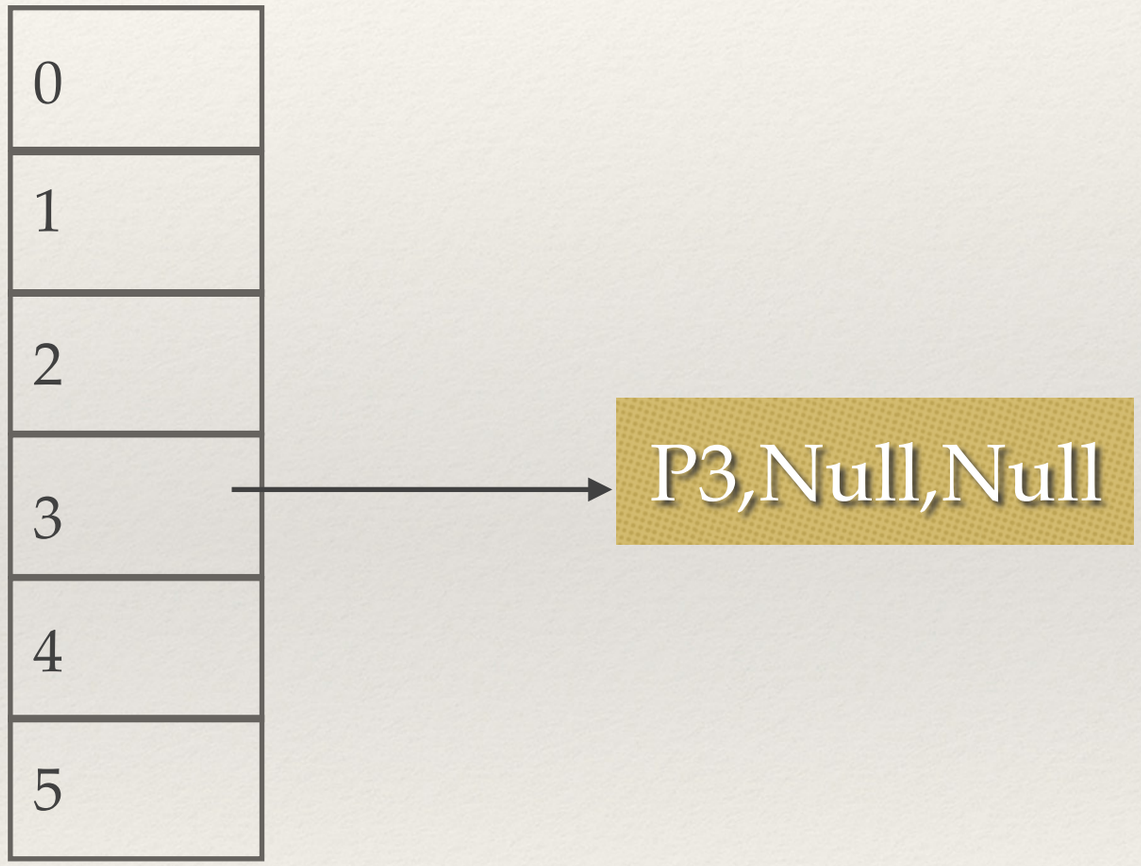
Original Data

supp	item	price
S6	P3	
S2	P3	100
S2	P2	20
	P1	
S7	P4	1100
S7	P1	11
S7	P2	22
S1	P1	10
S7	P3	110
S1	P2	20
S6	P1	12
S3	P4	1000
S5		
S4	P1	9
S4	P3	

AVG(Price) GROUP BY Item

supp	item	price
S6	**P3**	
S2	P3	100
S2	P2	20
	P1	
S7	P4	1100
S7	P1	11
S7	P2	22
S1	P1	10
S7	P3	110
S1	P2	20
S6	P1	12
S3	P4	1000
S5		
S4	P1	9
S4	P3	

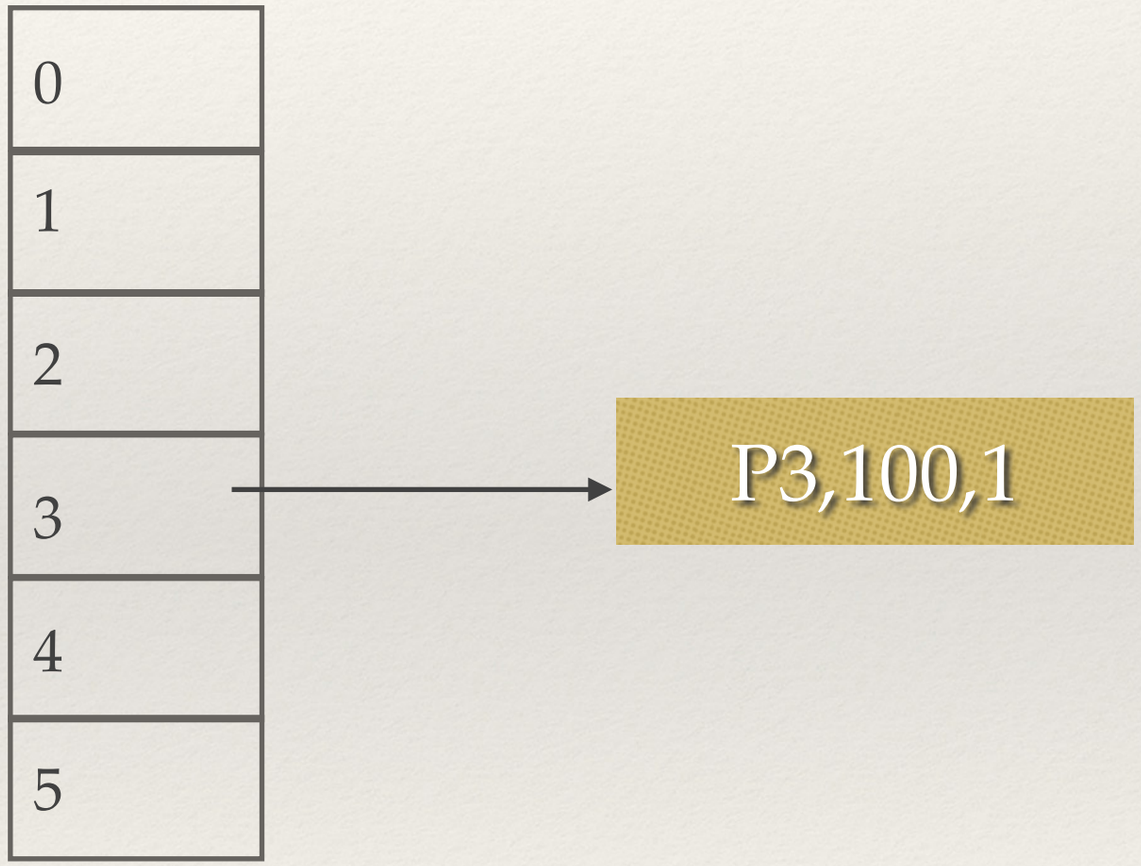
GROUP BY value,SUM,COUNT



AVG(Price) GROUP BY Item

supp	item	price
S6	P3	
S2	**P3**	**100**
S2	P2	20
	P1	
S7	P4	1100
S7	P1	11
S7	P1	11
S7	P2	22
S1	P1	10
S7	P3	110
S1	P2	20
S6	P1	12
S3	P4	1000
S5		
S4	P1	9
S4	P3	

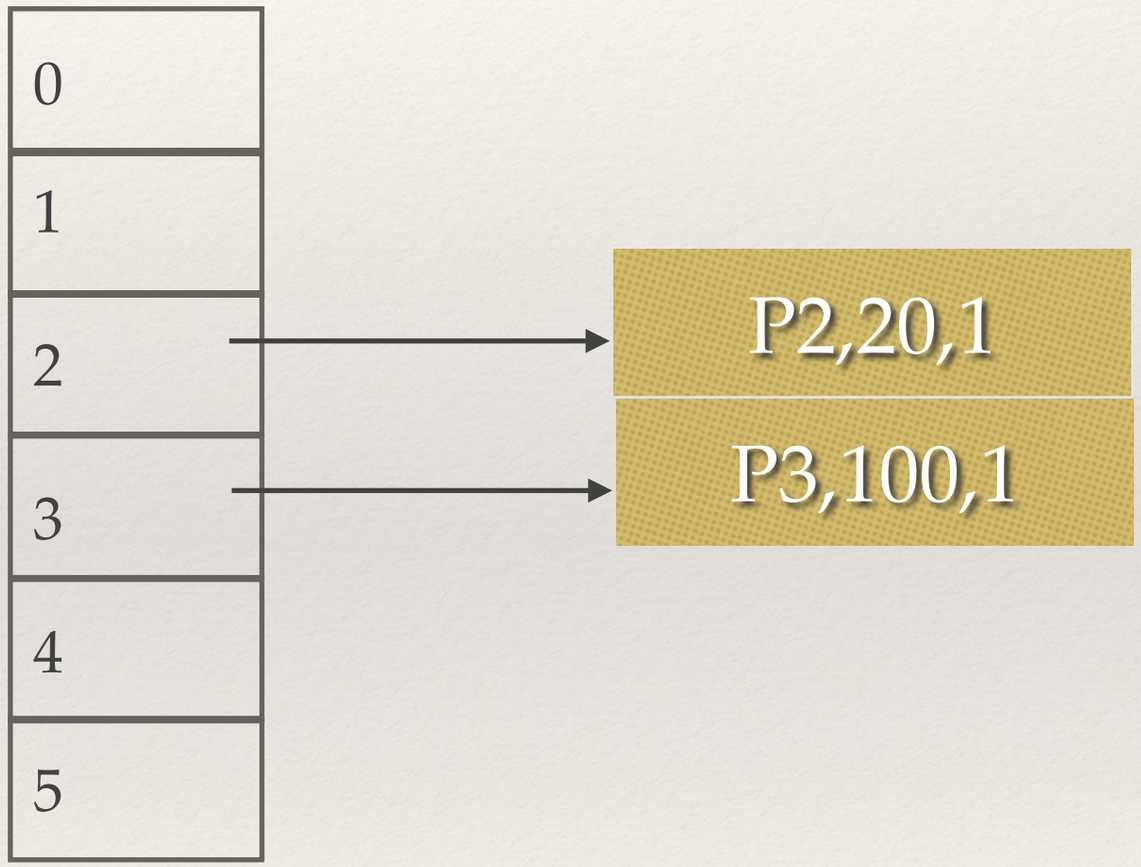
GROUP BY value,SUM,COUNT



AVG(Price) GROUP BY Item

supp	item	price
S6	P3	
S2	P3	100
S2	**P2**	**20**
	P1	
S7	P4	1100
S7	P1	11
S7	P2	22
S1	P1	10
S7	P3	110
S1	P2	20
S6	P1	12
S3	P4	1000
S5		
S4	P1	9
S4	P3	

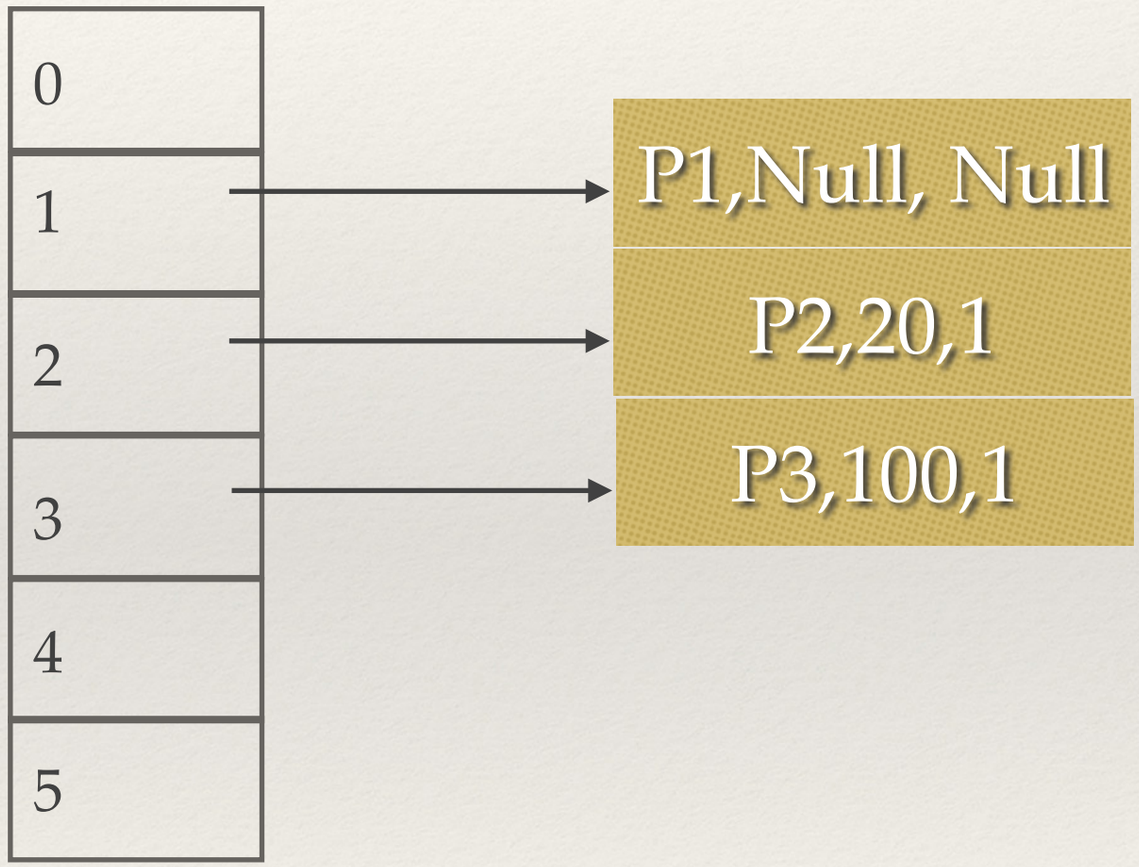
GROUP BY value,SUM,COUNT



AVG(Price) GROUP BY Item

supp	item	price
S6	P3	
S2	P3	100
S2	P2	20
	P1	
S7	P4	1100
S7	P1	11
S7	P2	22
S1	P1	10
S7	P3	110
S1	P2	20
S6	P1	12
S3	P4	1000
S5		
S4	P1	9
S4	P3	

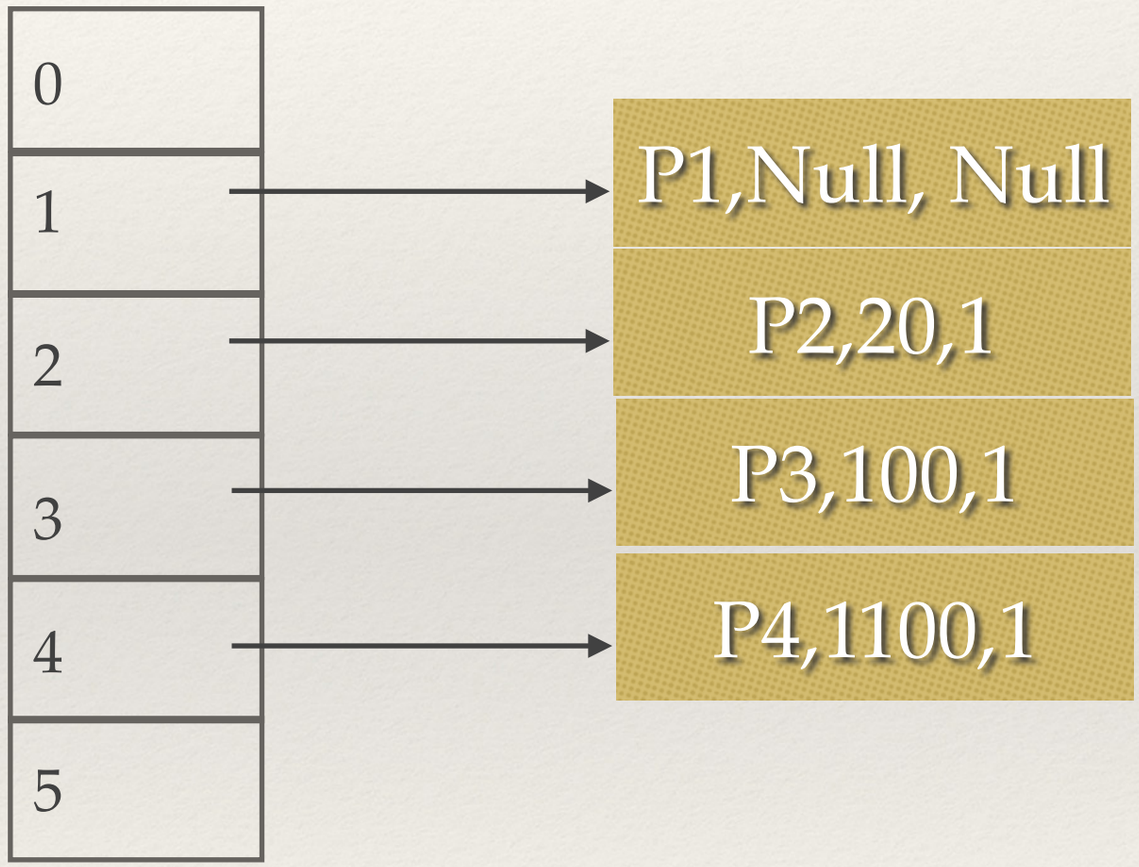
GROUP BY value,SUM,COUNT



AVG(Price) GROUP BY Item

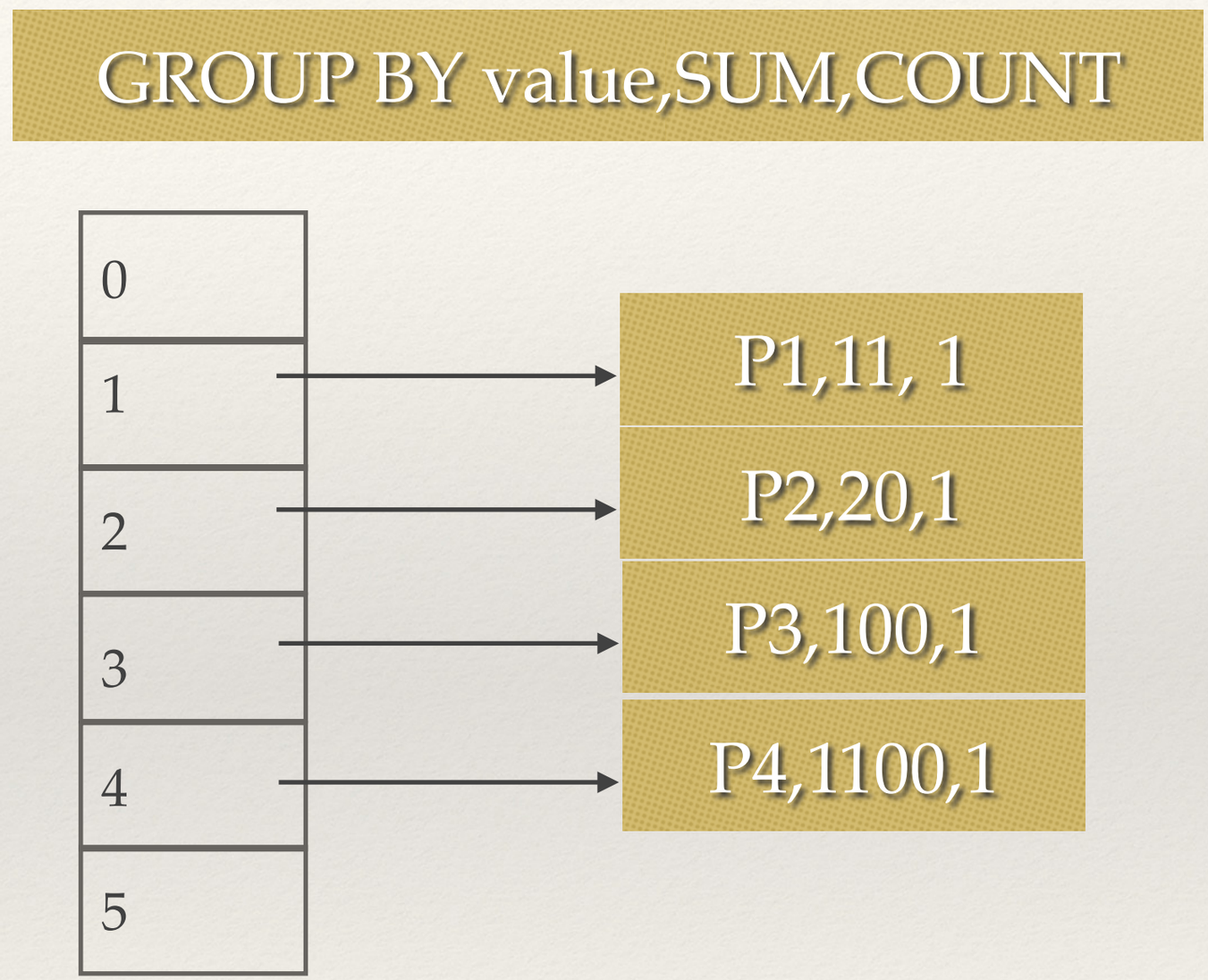
supp	item	price
S6	P3	
S2	P3	100
S2	P2	20
	P1	
S7	**P4**	**1100**
S7	P1	11
S7	P2	22
S1	P1	10
S7	P3	110
S1	P2	20
S6	P1	12
S3	P4	1000
S5		
S4	P1	9
S4	P3	

GROUP BY value,SUM,COUNT



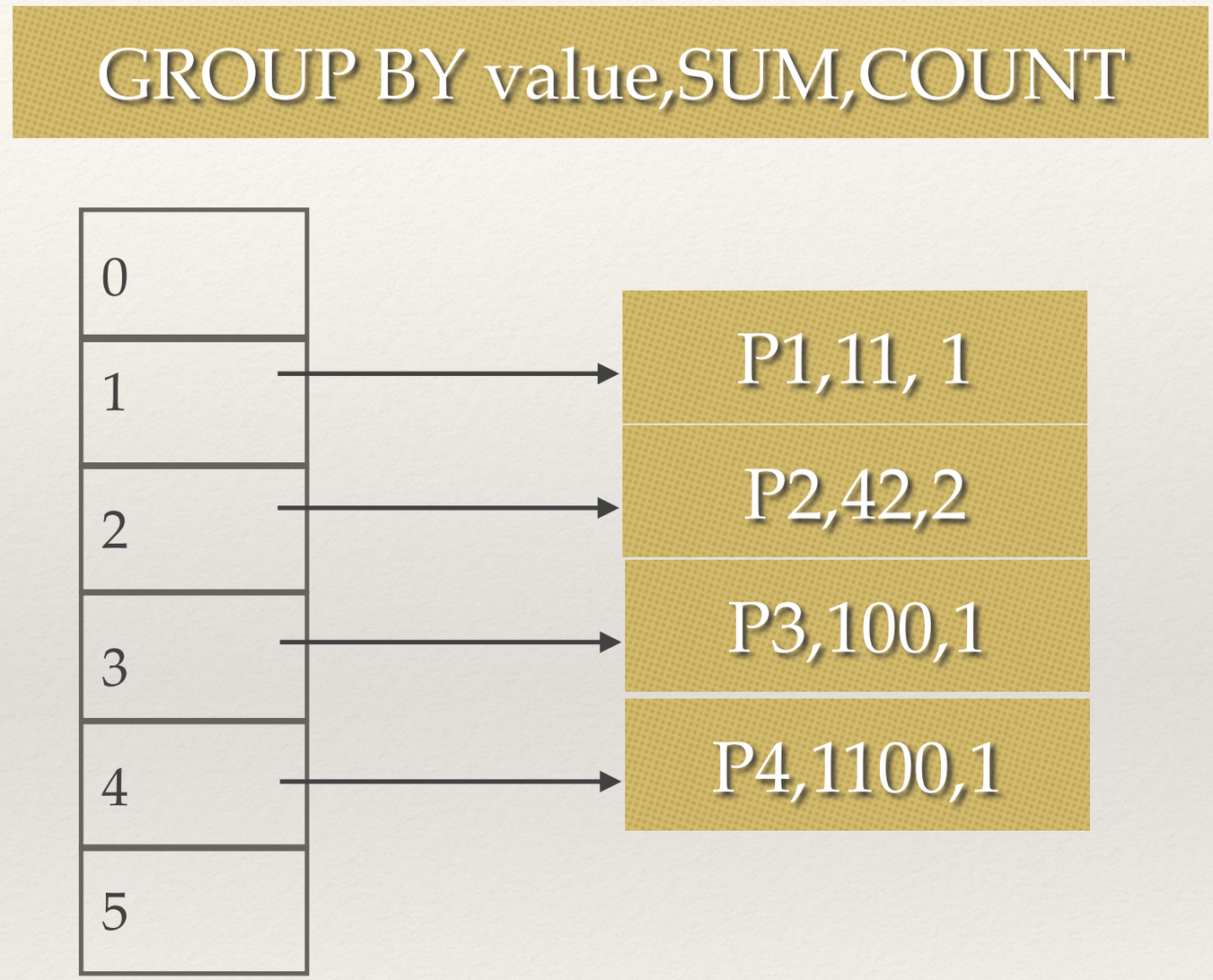
AVG(Price) GROUP BY Item

supp	item	price
S6	P3	
S2	P3	100
S2	P2	20
	P1	
S7	P4	1100
S7	**P1**	**11**
S7	P2	22
S1	P1	10
S7	P3	110
S1	P2	20
S6	P1	12
S3	P4	1000
S5		
S4	P1	9
S4	P3	



AVG(Price) GROUP BY Item

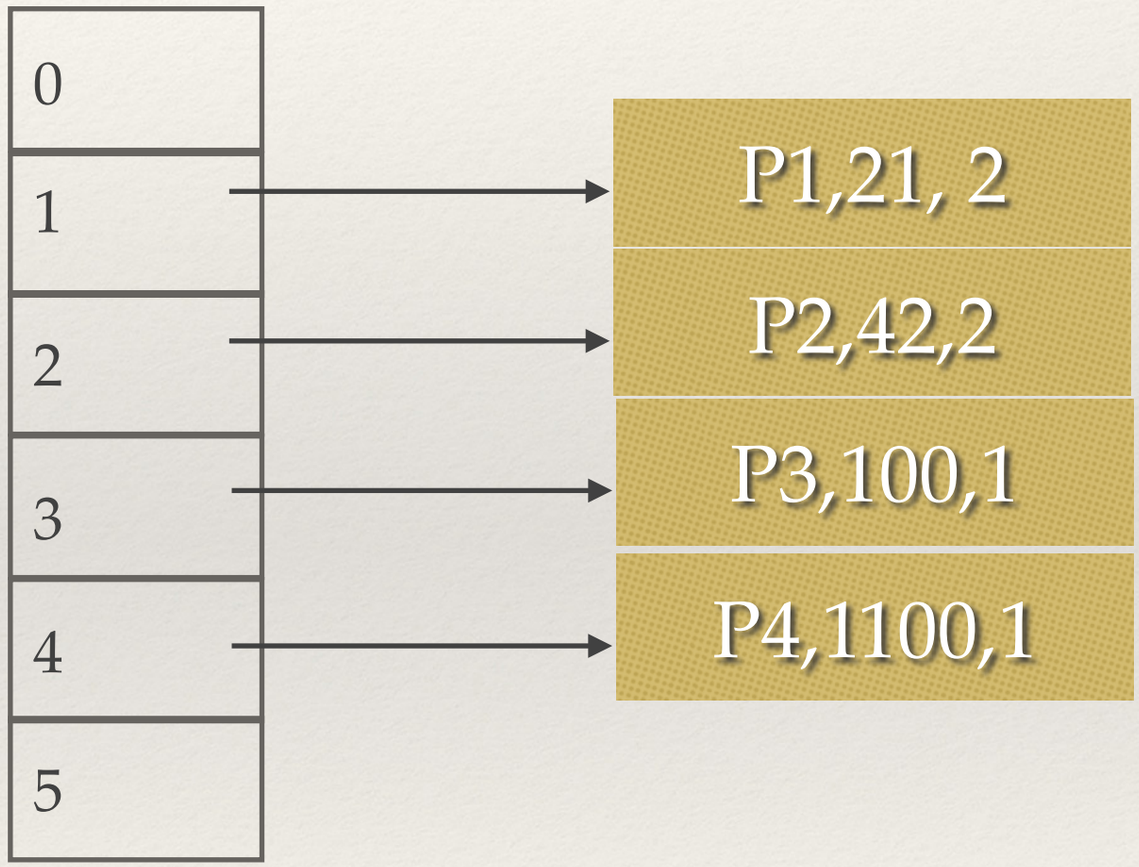
supp	item	price
S6	P3	
S2	P3	100
S2	P2	20
	P1	
S7	P4	1100
S7	P1	11
S7	**P2**	**22**
S1	P1	10
S7	P3	110
S1	P2	20
S6	P1	12
S3	P4	1000
S5		
S4	P1	9
S4	P3	



AVG(Price) GROUP BY Item

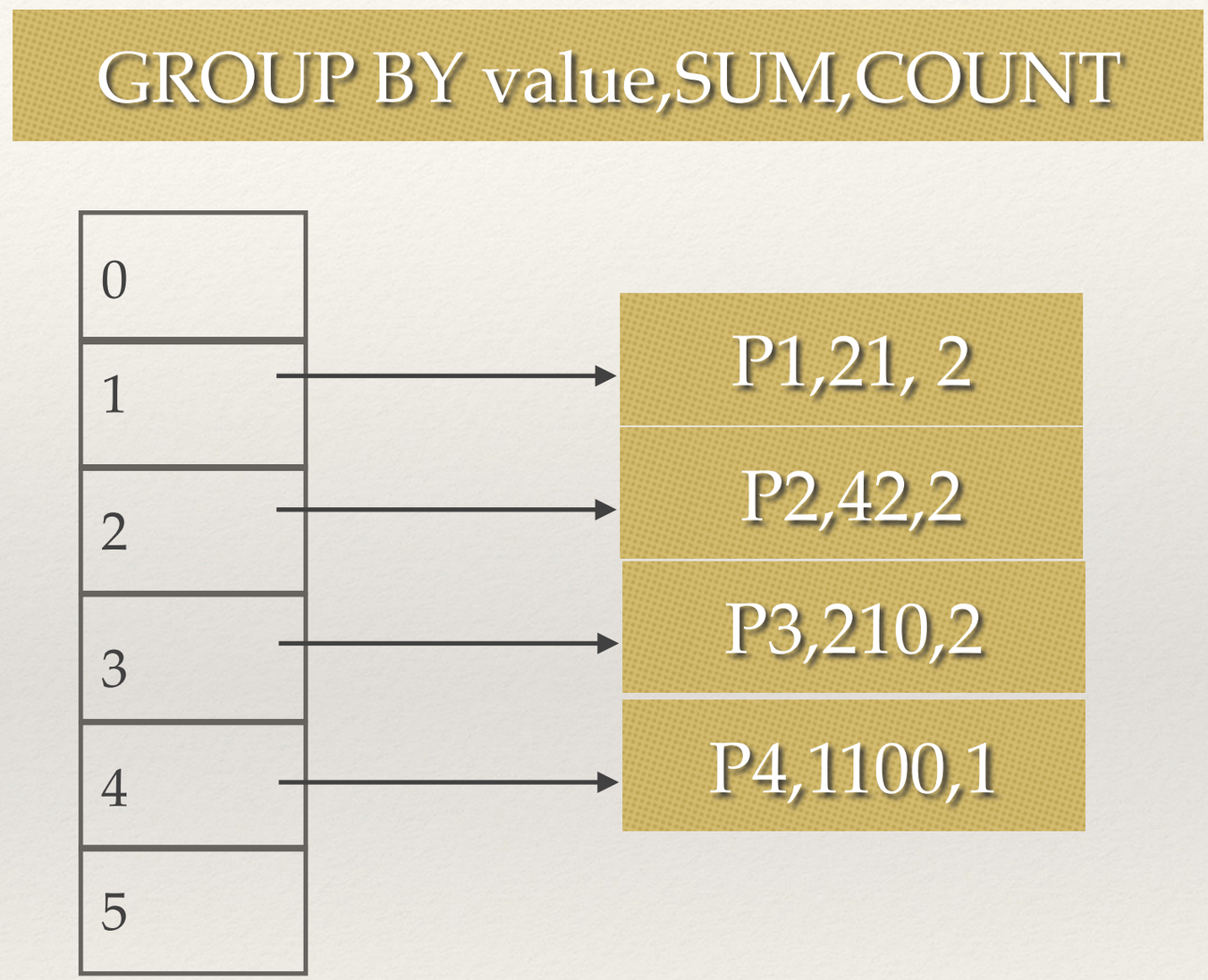
supp	item	price
S6	P3	
S2	P3	100
S2	P2	20
	P1	
S7	P4	1100
S7	P1	11
S7	P2	22
S1	**P1**	**10**
S7	P3	110
S1	P2	20
S6	P1	12
S3	P4	1000
S5		
S4	P1	9
S4	P3	

GROUP BY value,SUM,COUNT



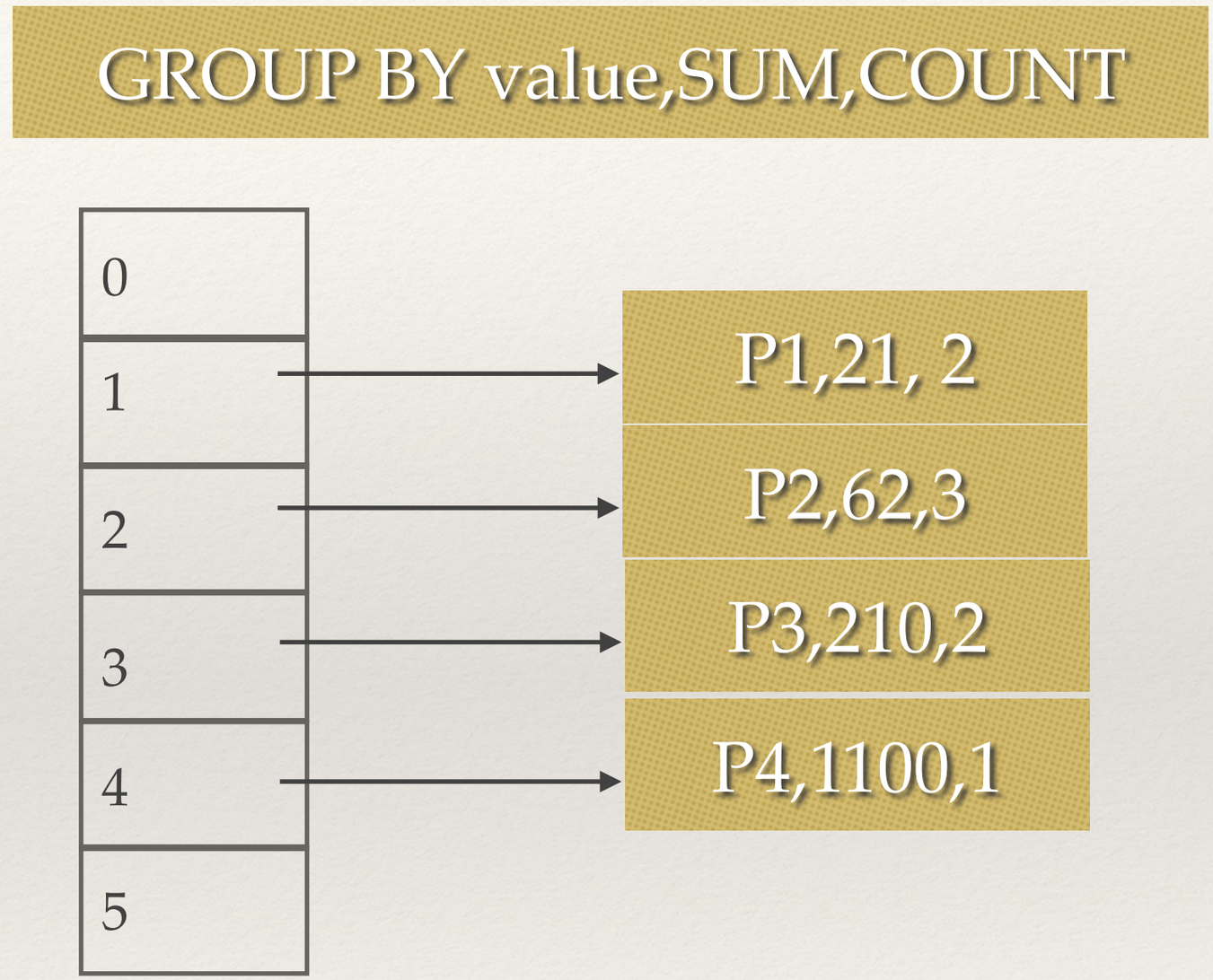
AVG(Price) GROUP BY Item

supp	item	price
S6	P3	
S2	P3	100
S2	P2	20
	P1	
S7	P4	1100
S7	P1	11
S7	P2	22
S1	P1	10
S7	**P3**	**110**
S1	P2	20
S6	P1	12
S3	P4	1000
S5		
S4	P1	9
S4	P3	



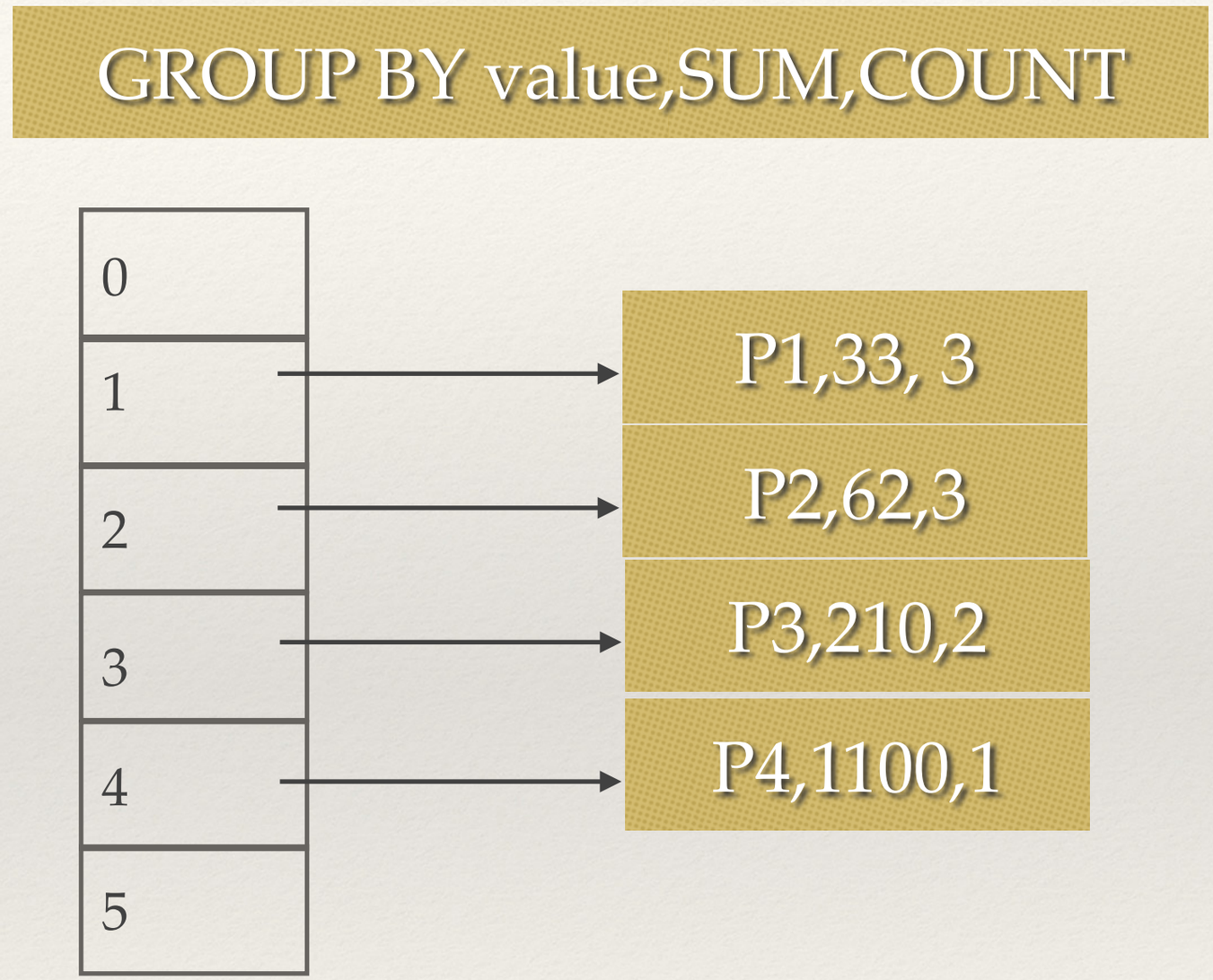
AVG(Price) GROUP BY Item

supp	item	price
S6	P3	
S2	P3	100
S2	P2	20
	P1	
S7	P4	1100
S7	P1	11
S7	P2	22
S1	P1	10
S7	P3	110
S1	**P2**	**20**
S6	P1	12
S3	P4	1000
S5		
S4	P1	9
S4	P3	



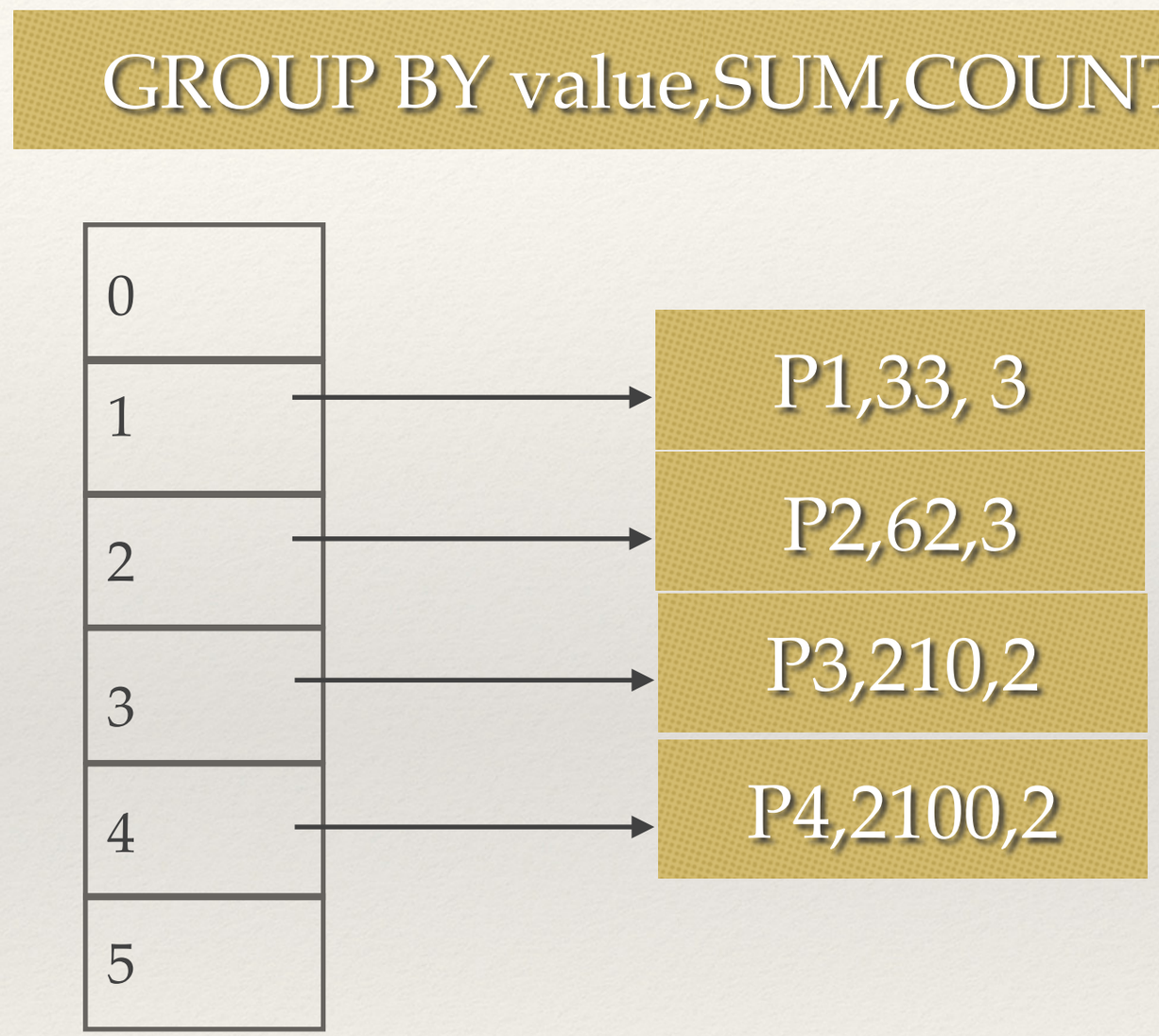
AVG(Price) GROUP BY Item

supp	item	price
S6	P3	
S2	P3	100
S2	P2	20
	P1	
S7	P4	1100
S7	P1	11
S7	P2	22
S1	P1	10
S7	P3	110
S1	P2	20
S6	**P1**	**12**
S3	P4	1000
S5		
S4	P1	9
S4	P3	



AVG(Price) GROUP BY Item

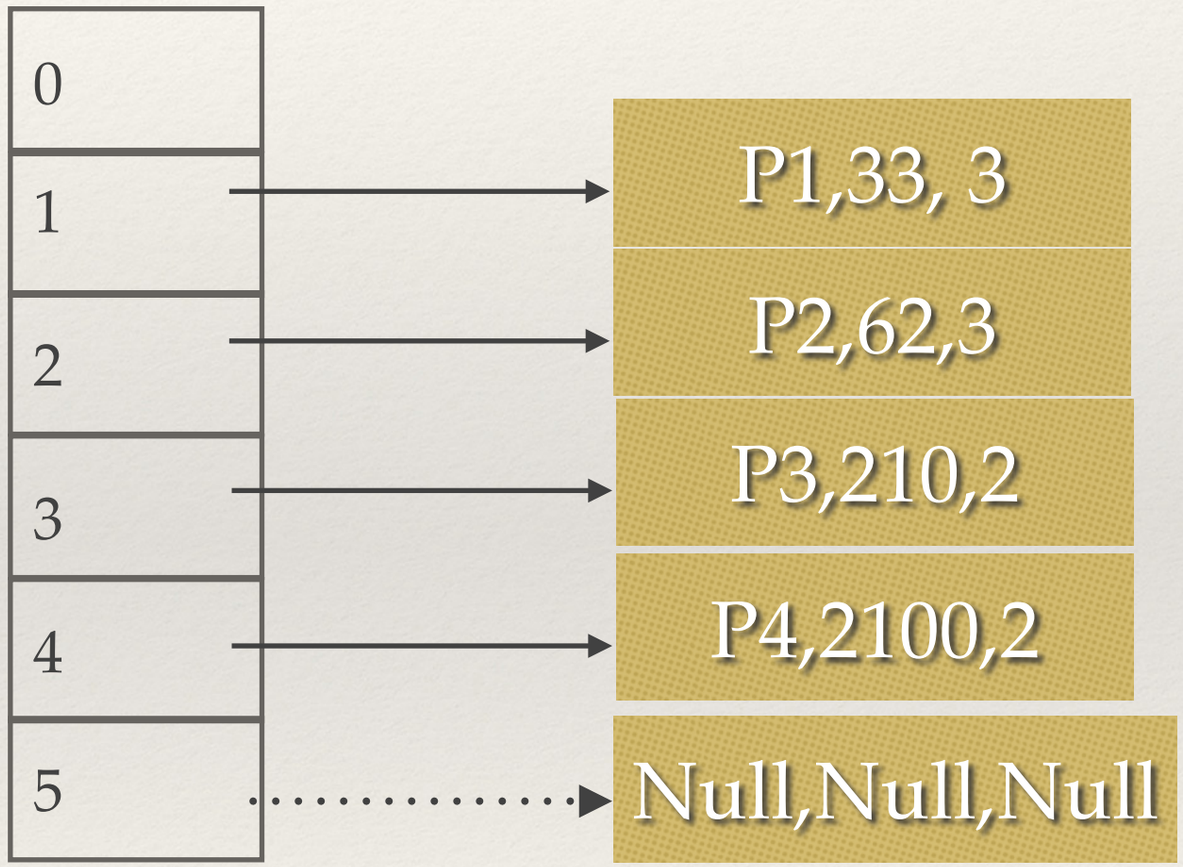
supp	item	price
S6	P3	
S2	P3	100
S2	P2	20
	P1	
S7	P4	1100
S7	P1	11
S7	P2	22
S1	P1	10
S7	P3	110
S1	P2	20
S6	P1	12
S3	**P4**	**1000**
S5		
S4	P1	9
S4	P3	



No Room for new hash value: Flush

supp	item	price
S6	P3	
S2	P3	100
S2	P2	20
	P1	
S7	P4	1100
S7	P1	11
S7	P2	22
S1	P1	10
S7	P3	110
S1	P2	20
S6	P1	12
S3	P4	1000
S5		
S4	P1	9
S4	P3	

GROUP BY value,SUM,COUNT

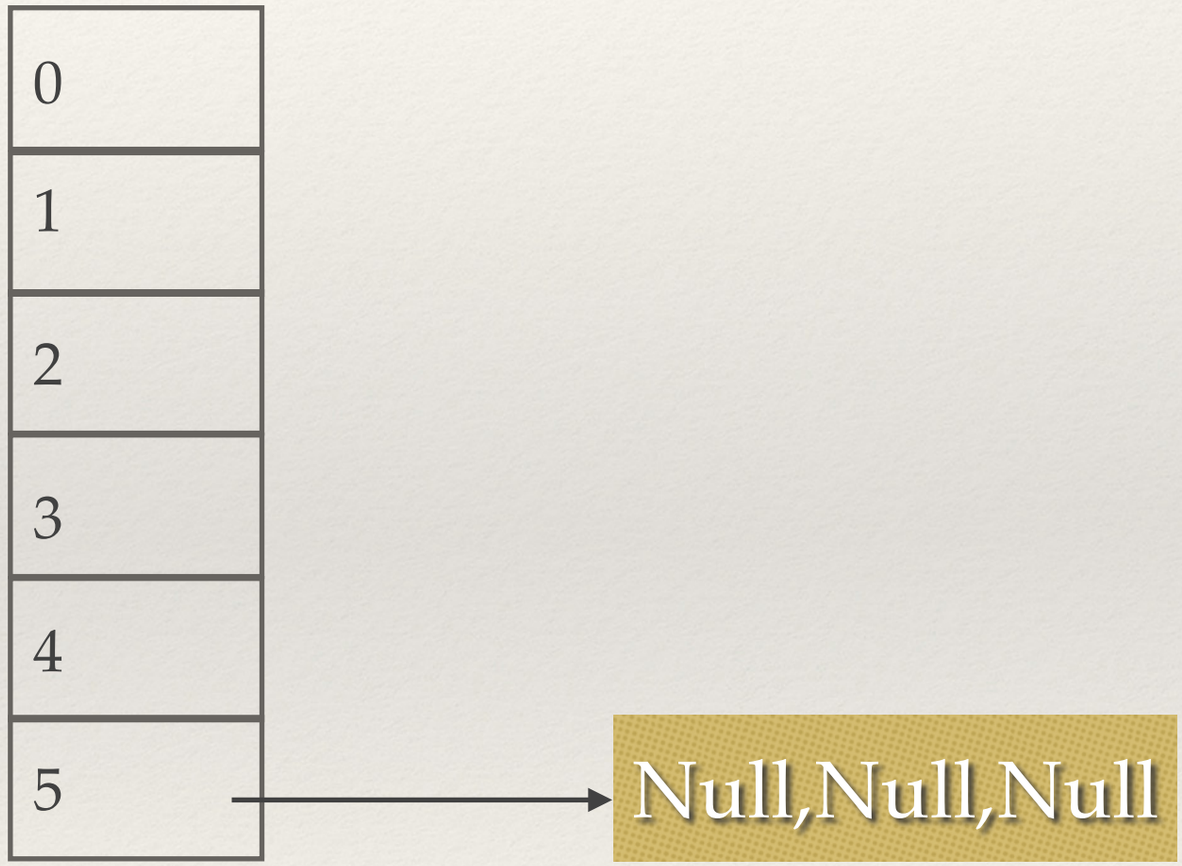



```
P1,   33, 3
P2,   62, 3
P3,  210, 2
P4, 2100, 2
```

No Room for new hash value: Flush

supp	item	price
S6	P3	
S2	P3	100
S2	P2	20
	P1	
S7	P4	1100
S7	P1	11
S7	P2	22
S1	P1	10
S7	P3	110
S1	P2	20
S6	P1	12
S3	P4	1000
S5		
S4	P1	9
S4	P3	

GROUP BY value,SUM,COUNT

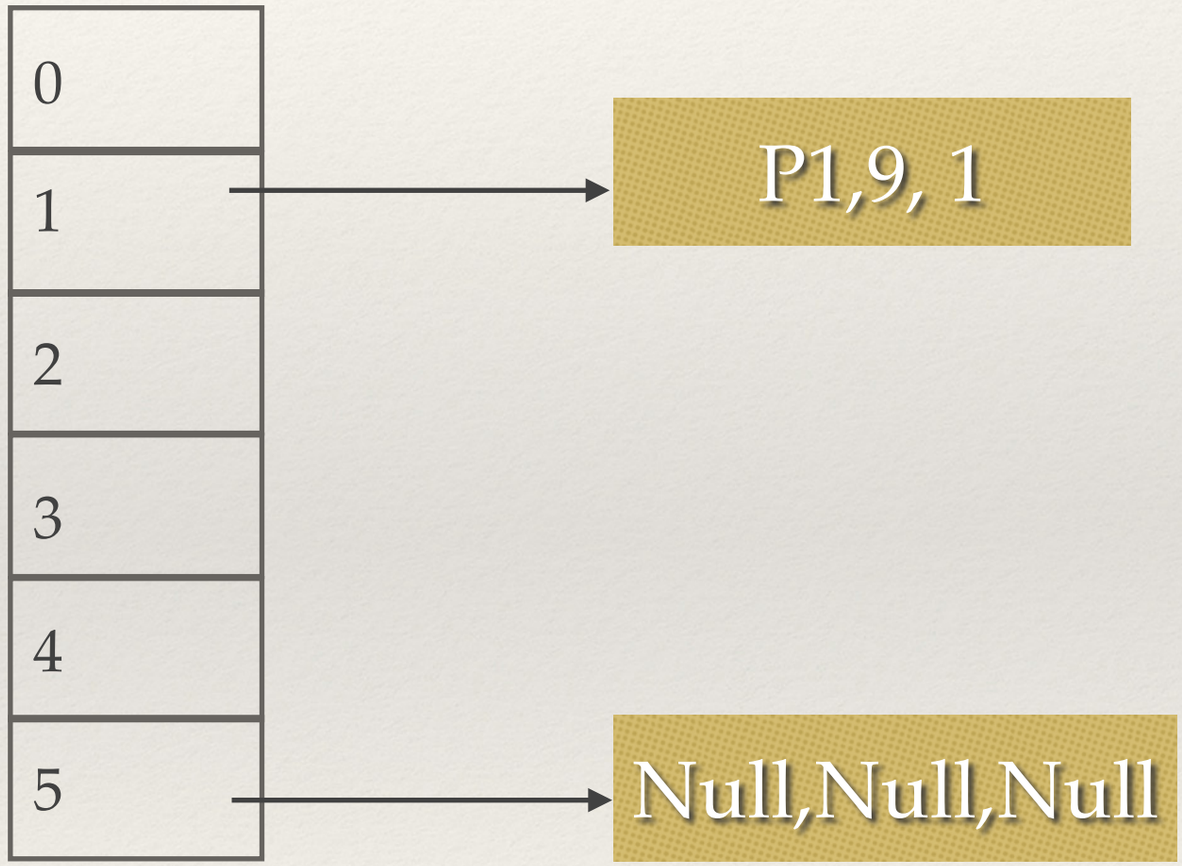


P1, 33, 3
P2, 62, 3
P3, 210, 2
P4, 2100, 2

AVG(Price) GROUP BY Item

supp	item	price
S6	P3	
S2	P3	100
S2	P2	20
	P1	
S7	P4	1100
S7	P1	11
S7	P2	22
S1	P1	10
S7	P3	110
S1	P2	20
S6	P1	12
S3	P4	1000
S5		
S4	**P1**	**9**
S4	P3	

GROUP BY value,SUM,COUNT

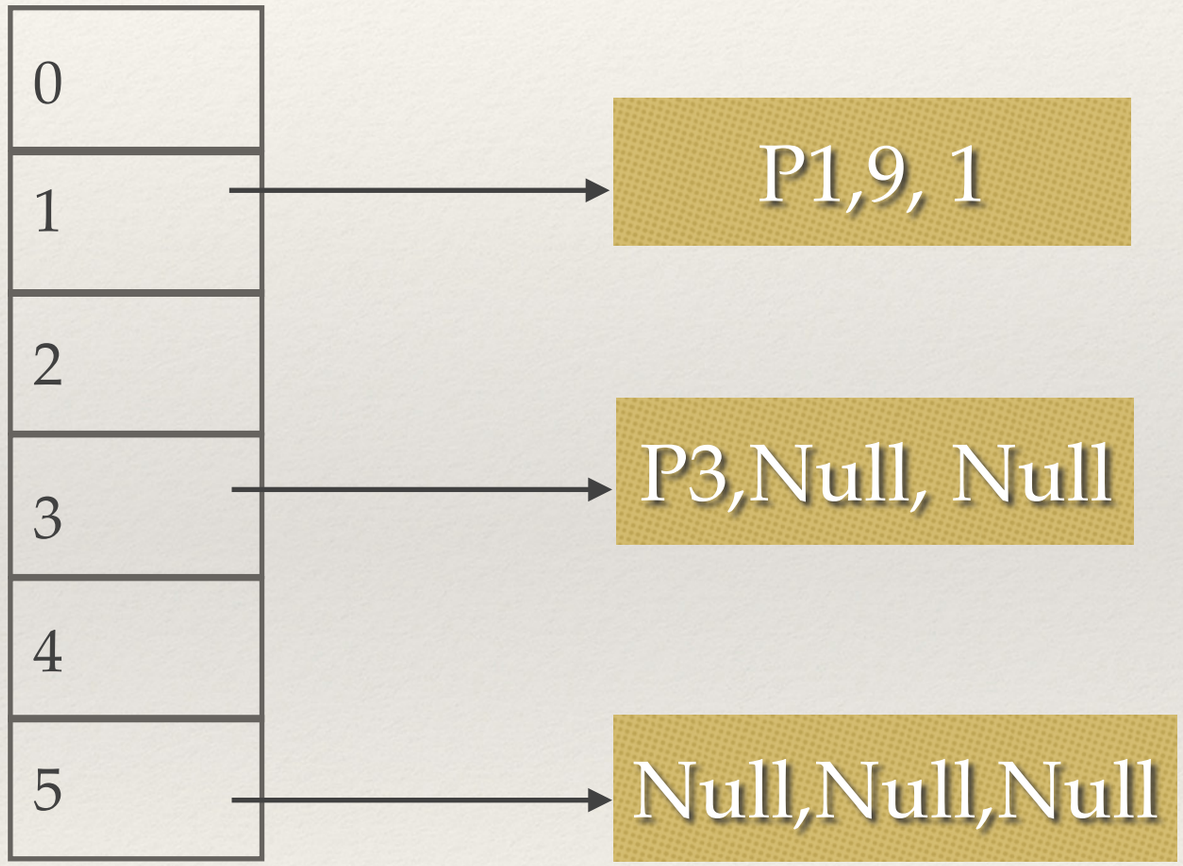



```
P1,    33, 3
P2,    62, 3
P3,   210, 2
P4, 2100, 2
```

AVG(Price) GROUP BY Item

supp	item	price
S6	P3	
S2	P3	100
S2	P2	20
	P1	
S7	P4	1100
S7	P1	11
S7	P2	22
S1	P1	10
S7	P3	110
S1	P2	20
S6	P1	12
S3	P4	1000
S5		
S4	P1	9
S4	**P3**	

GROUP BY value,SUM,COUNT

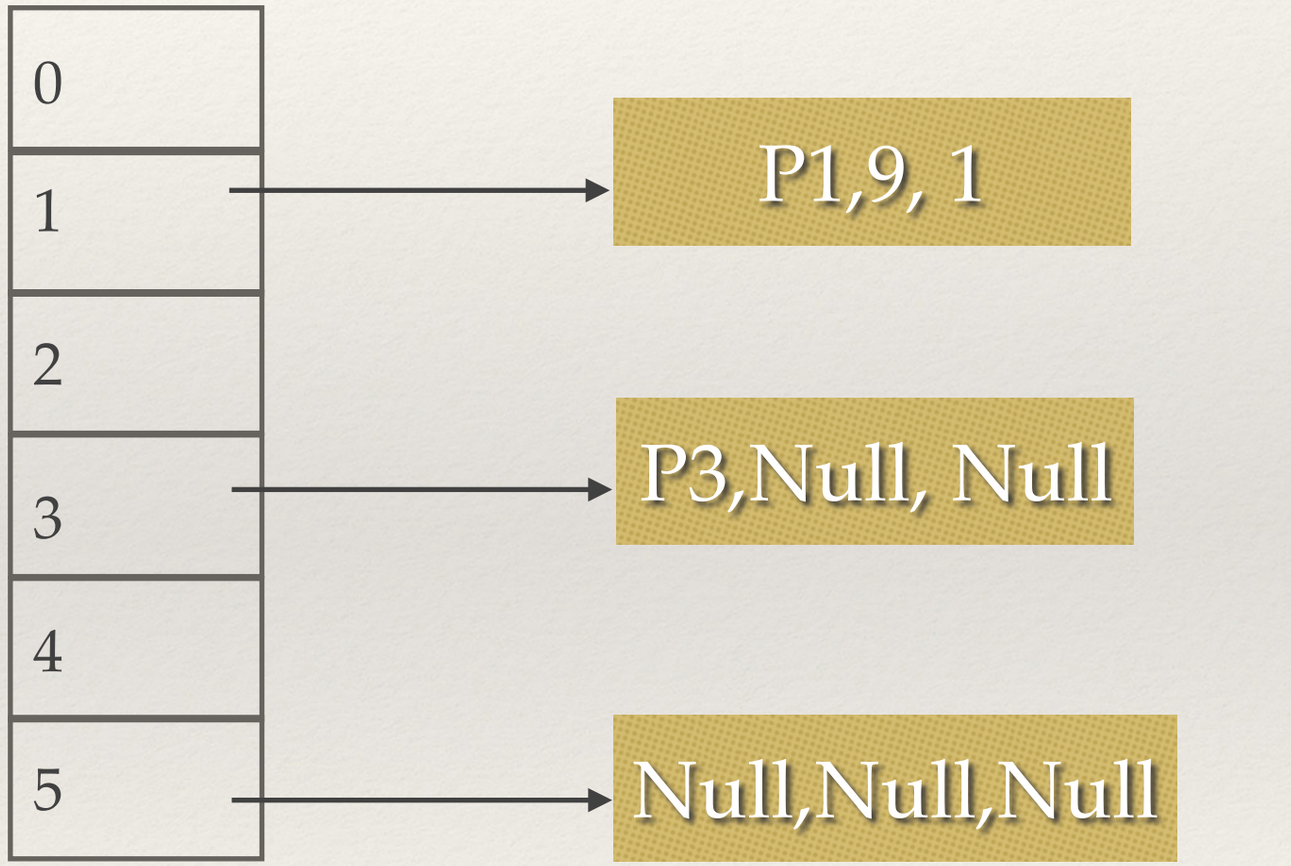



```
P1,   33, 3
P2,   62, 3
P3,  210, 2
P4, 2100, 2
```

AVG(Price) GROUP BY Item

supp	item	price
S6	P3	
S2	P3	100
S2	P2	20
	P1	
S7	P4	1100
S7	P1	11
S7	P2	22
S1	P1	10
S7	P3	110
S1	P2	20
S6	P1	12
S3	P4	1000
S5		
S4	P1	9
S4	**P3**	

GROUP BY value,SUM,COUNT




```
P1,    33, 3
P2,    62, 3
P3,   210, 2
P4, 2100, 2
```

```
     P1,    9,    1
+    P3, Null, Null
  Null, Null, Null
```

Aggregate Using Sort

```
P1,    33, 3
P2,    62, 3
P3,   210, 2
P4, 2100, 2
```

```
      P1,    9,    1
+     P3, Null, Null
   Null, Null, Null
```

```
P1,      33, 3
P1,       9, 1
P2,      62, 3
P3,     210, 2
P3,    Null, Null
P4,    2100, 2
Null, Null, Null
```

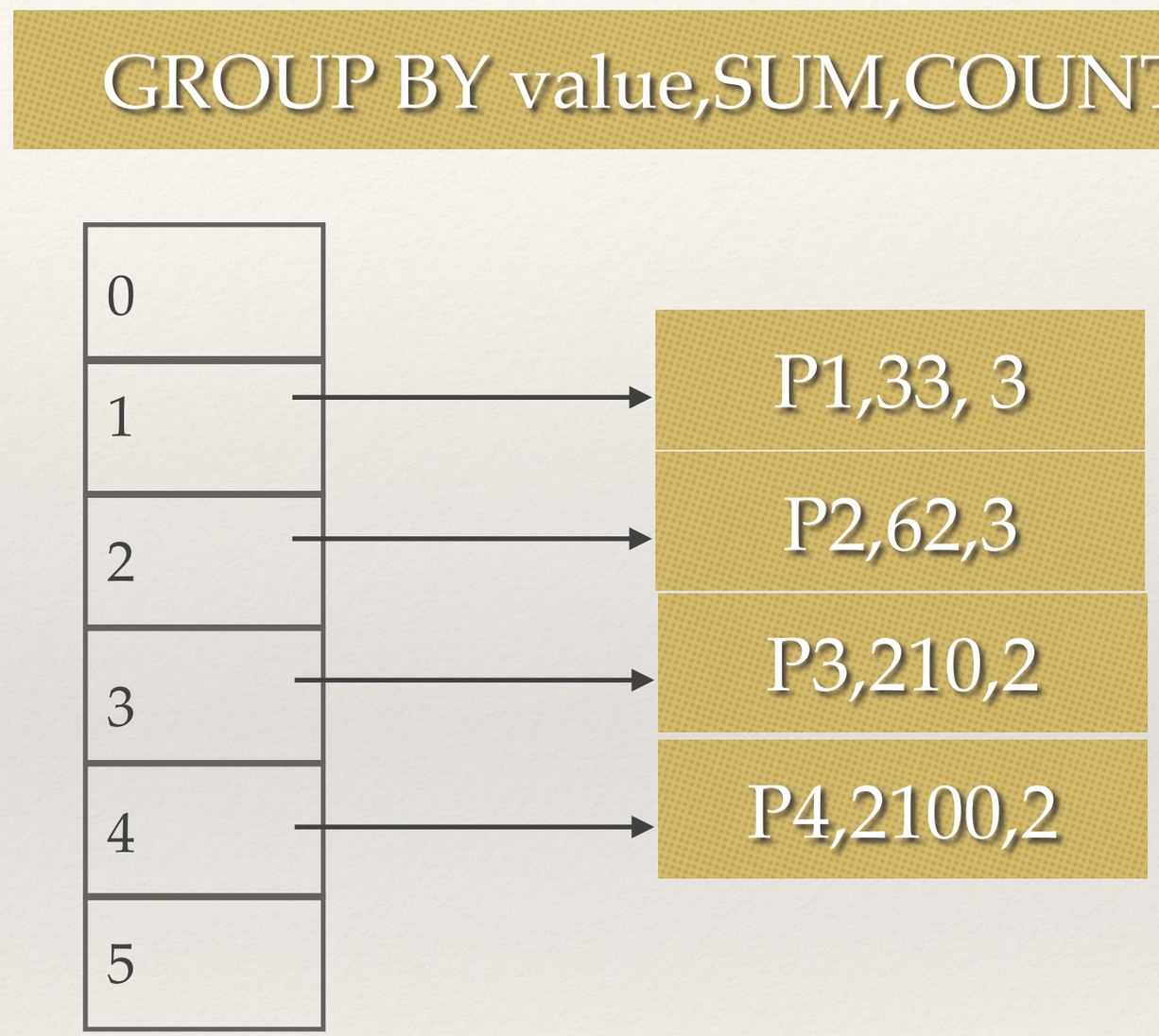
```
P1,      42, 4
P2,      62, 3
P3,     210, 2
P4,    2100, 2
Null, Null, Null
```

Handling Memory Overflow

- Option 2
  - Flush part of the current "cache" to make room
  - Scan the rest of rows
    - if in the current hash table, aggregate
    - else "divide" them up into hash partitions that would fit in memory
  - For each hash partition, aggregate as usual
- O(N) still but with up to roughly 2 scans over the data

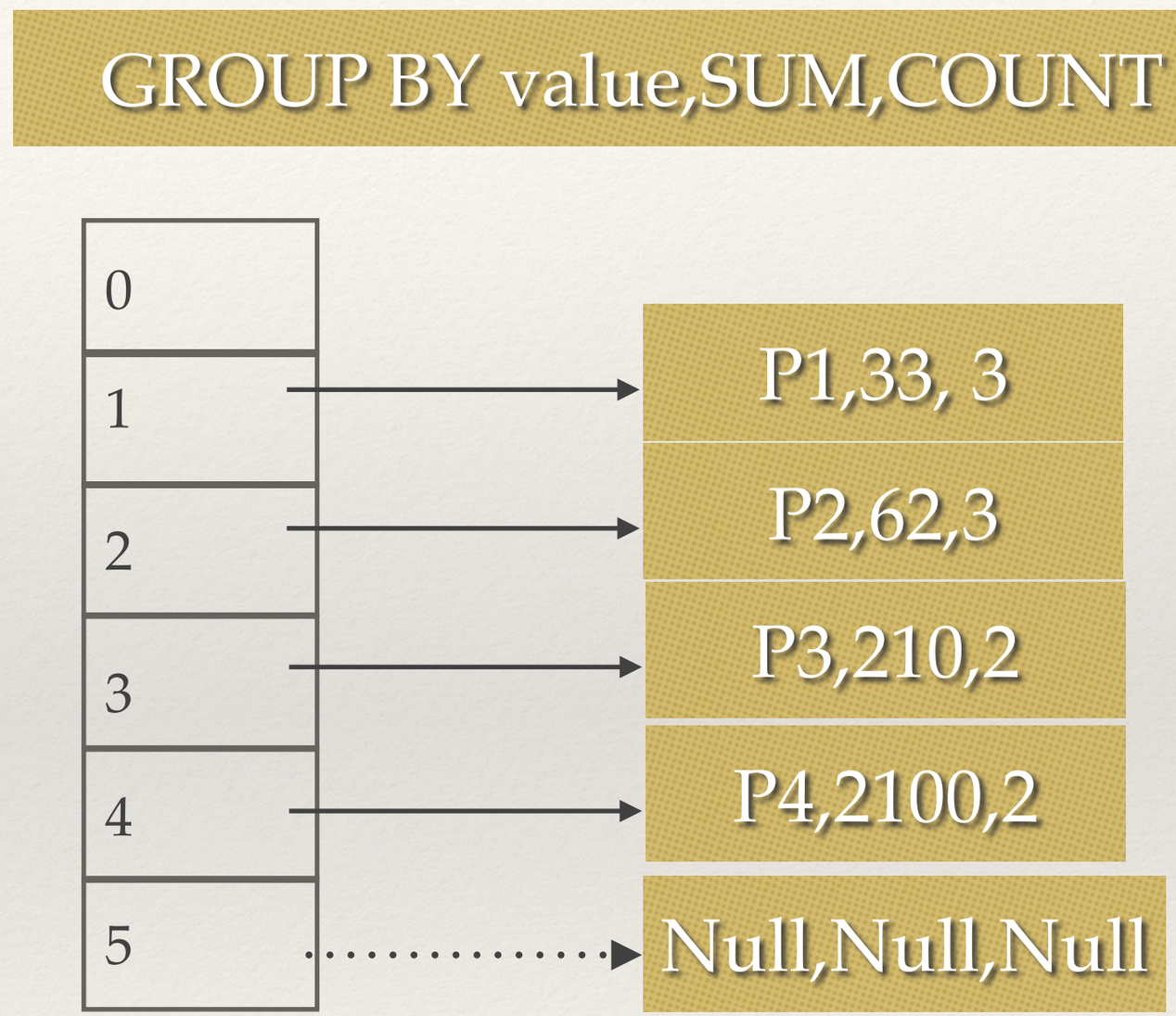
AVG(Price) GROUP BY Item

supp	item	price
S6	P3	
S2	P3	100
S2	P2	20
	P1	
S7	P4	1100
S7	P1	11
S7	P2	22
S1	P1	10
S7	P3	110
S1	P2	20
S6	P1	12
S3	**P4**	**1000**
S5		
S4	P1	9
S4	P3	



No Room for new hash value: Flush

supp	item	price
S6	P3	
S2	P3	100
S2	P2	20
	P1	
S7	P4	1100
S7	P1	11
S7	P2	22
S1	P1	10
S7	P3	110
S1	P2	20
S6	P1	12
S3	P4	1000
S5		
S4	P1	9
S4	P3	

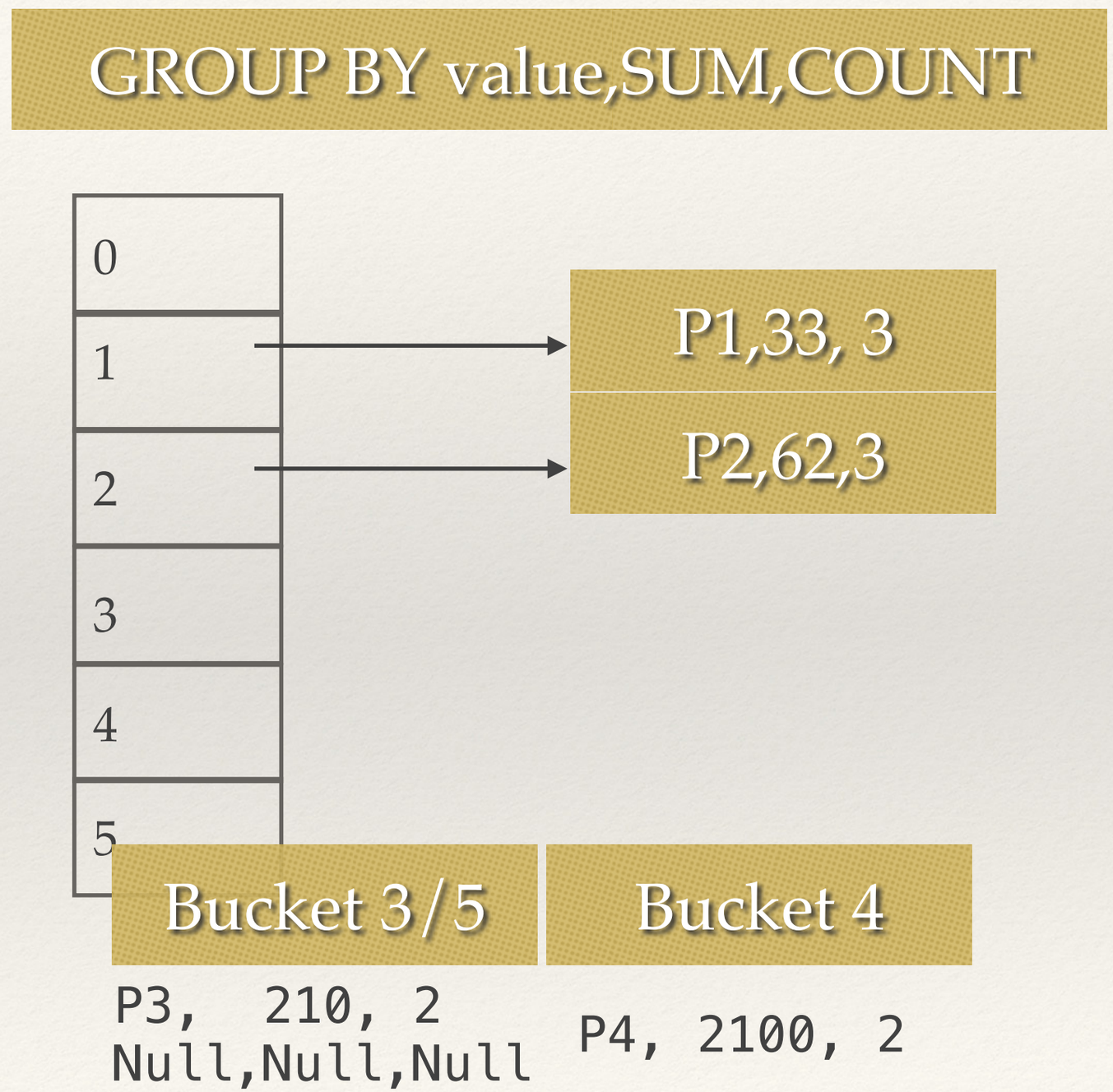


P3, 210, 2
Null,Null,Null

P4, 2100, 2

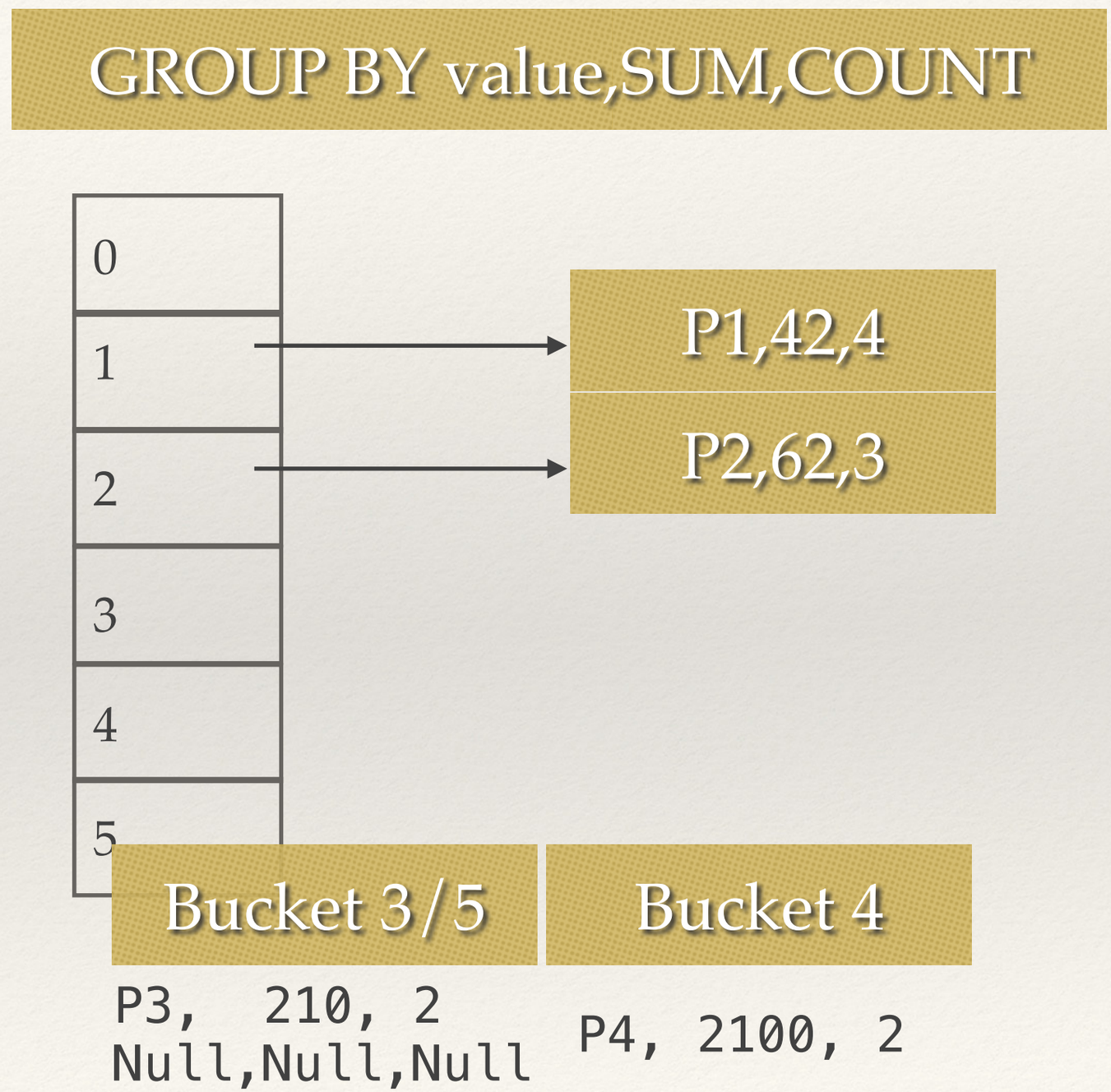
No Room for new hash value: Flush

supp	item	price
S6	P3	
S2	P3	100
S2	P2	20
	P1	
S7	P4	1100
S7	P1	11
S7	P2	22
S1	P1	10
S7	P3	110
S1	P2	20
S6	P1	12
S3	P4	1000
S5		
S4	P1	9
S4	P3	



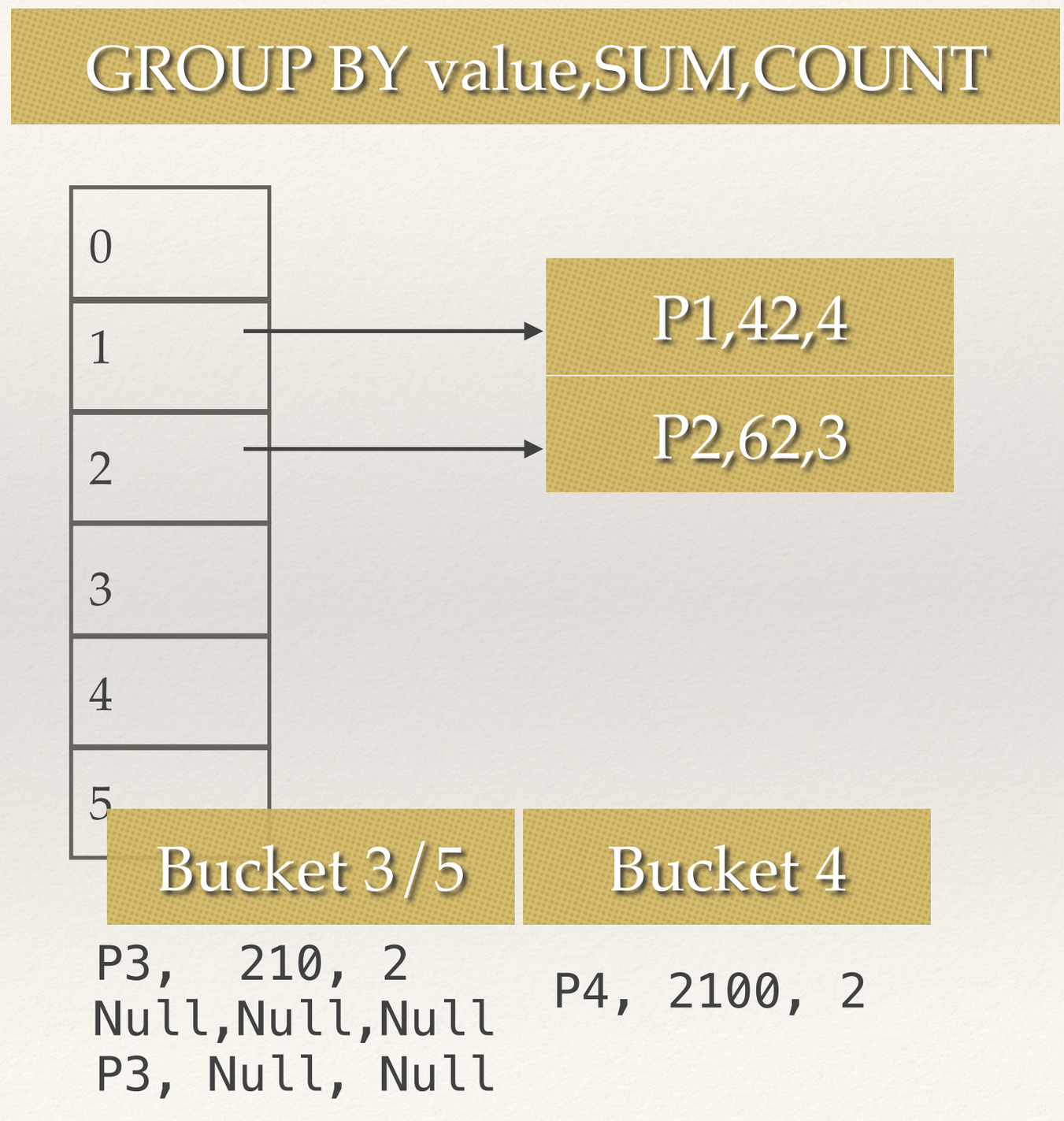
No Room for new hash value: Flush

supp	item	price
S6	P3	
S2	P3	100
S2	P2	20
	P1	
S7	P4	1100
S7	P1	11
S7	P2	22
S1	P1	10
S7	P3	110
S1	P2	20
S6	P1	12
S3	P4	1000
S5		
S4	**P1**	**9**
S4	P3	



No Room for new hash value: Flush

supp	item	price
S6	P3	
S2	P3	100
S2	P2	20
	P1	
S7	P4	1100
S7	P1	11
S7	P2	22
S1	P1	10
S7	P3	110
S1	P2	20
S6	P1	12
S3	P4	1000
S5		
S4	P1	9
S4	**P3**	



Handling Memory Overflow

- If you already "knew" that there is not enough memory
- Hash Partition the table into buckets that would fit in memory
- Then process each partition independently
- O(N) - 2 scans of data

Holistic Functions

- Technically don't need sort to compute MEDIAN
- But sorting is the most practical way given GROUP BY
- Example: g, SUM(a), COUNT(a), MEDIAN(a)
- Sort on g, a (normally we sort just on g)
- MEDIAN(a) = the middle value of the group in the sort order
- Can it be done in one scan?
- What if there was another MEDIAN(b)?

Special Cases

- If the data is already sorted as needed (e.g. due *group by* being a prefix of the primary index)
  - skip the sort
- If the needed columns are available in a Secondary Index, scan the index as it's likely to be much smaller.
  - Must account for *rid's* in this case
  - Can skip the sort if the prefix condition met
- Assumes WHERE conditions can be evaluated using the index