Distributed OLAP Systems

Lecture #24

Database Systems
15-445/15-645
Fall 2017

Andy Pavlo
Computer Science Dept.
Carnegie Mellon Univ.

ADMINISTRIVIA

Monday Dec 4th – NuoDB

→ Barry Morris (Co-Founder, Exec. Chairman)

Wednesday Dec 6th – Potpourri + Final Review

→ Vote for what system you want me to talk about.

→ http://cmudb.io/f17-systems

Wednesday Dec 6th – Project #4

OLTP VS. OLAP

On-line Transaction Processing (OLTP):

→ Short-lived txns.
→ Small footprint.
→ Repetitive operations.

On-line Analytical Processing (OLAP):

→ Long running queries.
→ Complex joins.
→ Exploratory queries.

TODAY'S AGENDA

Partitioning

Distributed Join Algorithms

DATABASE PARTITIONING

Split database across multiple resources:

→ Disks, nodes, processors.

→ Sometimes called "sharding"

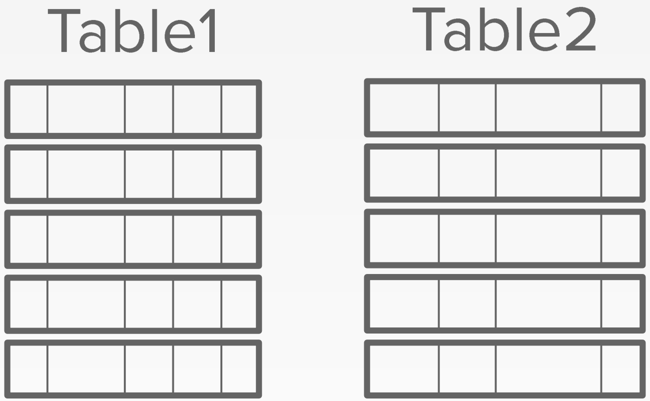
The DBMS executes query fragments on each partition and then combines the results to produce a single answer.

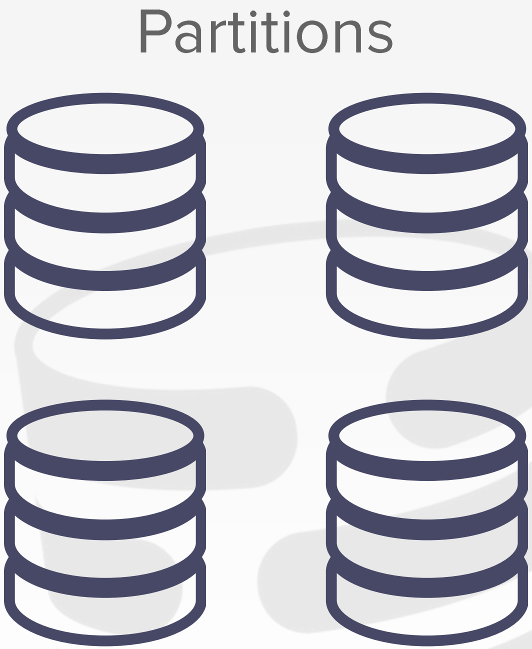
NAÏVE TABLE PARTITIONING

Each node stores one and only table.

Assumes that each node has enough storage space for a table.

NAÏVE TABLE PARTITIONING

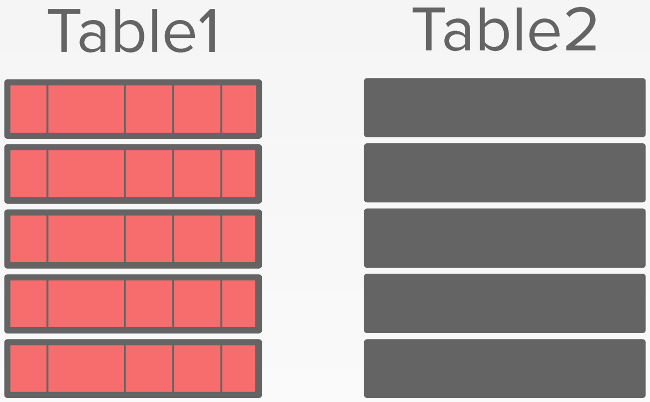


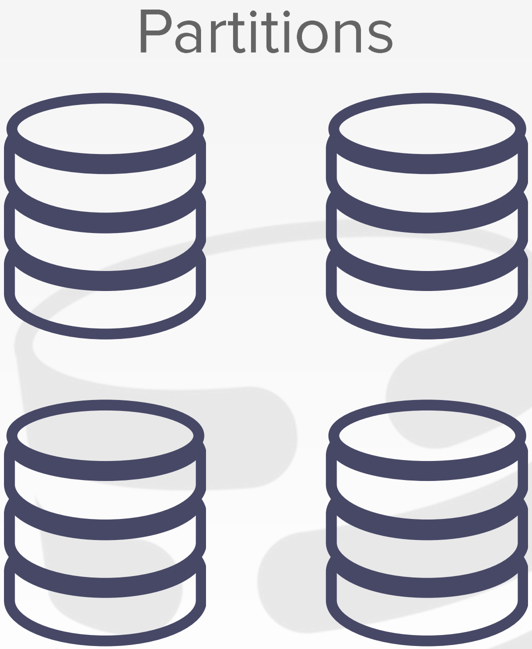


Ideal Query:

```
SELECT * FROM table
```

NAÏVE TABLE PARTITIONING

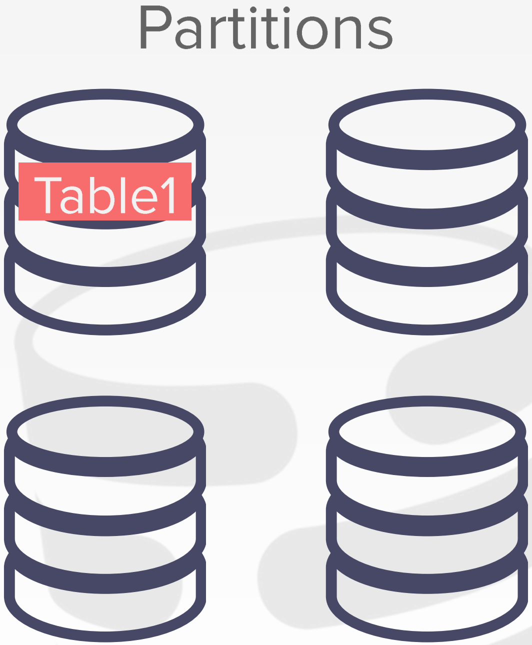




Ideal Query:

```
SELECT * FROM table
```

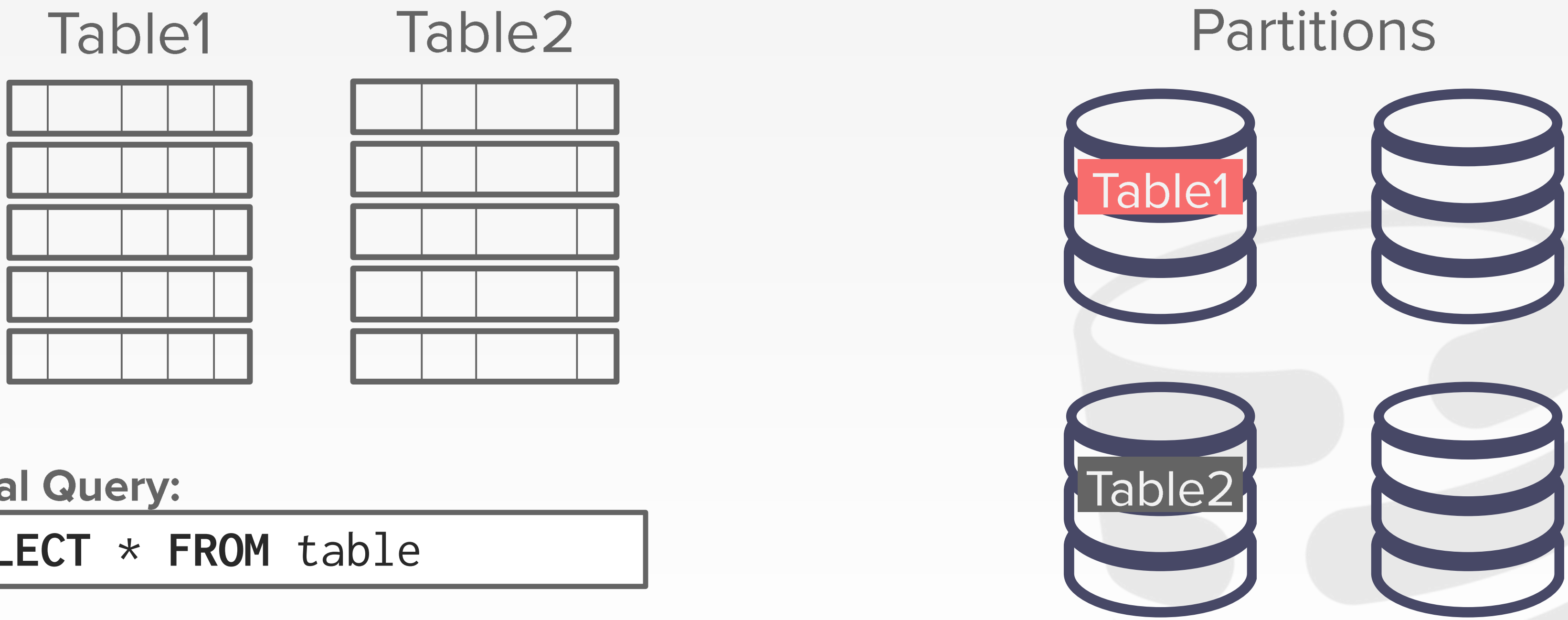
NAÏVE TABLE PARTITIONING



Ideal Query:

```
SELECT * FROM table
```

NAÏVE TABLE PARTITIONING



Ideal Query:

```
SELECT * FROM table
```

HORIZONTAL PARTITIONING

Split a table's tuples into disjoint subsets.

→ Choose column(s) that divides the database equally in terms of size, load, or usage.

→ Each tuple contains all of its columns.

→ Hash Partitioning, Range Partitioning

The DBMS can partition a database **physical** (shared nothing) or **logically** (shared disk).

HORIZONTAL PARTITIONING

Partitioning Key

Table1

101	a	XXX	2017-11-29
102	b	XXY	2017-11-28
103	c	XYZ	2017-11-29
104	d	XYX	2017-11-27
105	e	XYY	2017-11-29

Ideal Query:

```
SELECT * FROM table
 WHERE partitionKey = ?
```

Partitions

HORIZONTAL PARTITIONING

Partitioning Key

Table1

101	a	XXX	2017-11-29	hash(a)%4 = P2
102	b	XXY	2017-11-28	hash(b)%4 = P4
103	c	XYZ	2017-11-29	hash(c)%4 = P3
104	d	XYX	2017-11-27	hash(d)%4 = P2
105	e	XYY	2017-11-29	hash(e)%4 = P1

Partitions

Ideal Query:

```
SELECT * FROM table
 WHERE partitionKey = ?
```

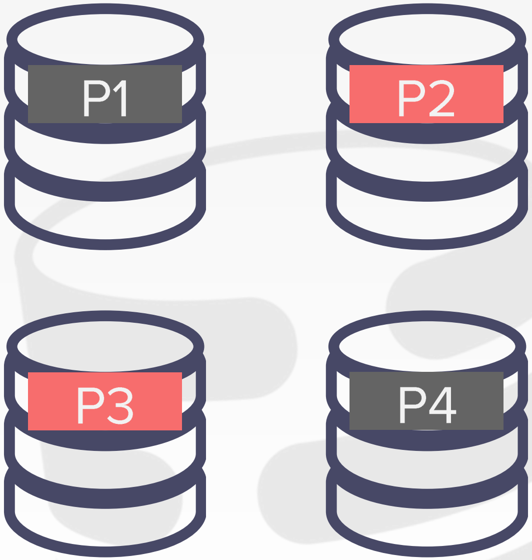
HORIZONTAL PARTITIONING

Partitioning Key

Table1

101	a	XXX	2017-11-29	hash(a)%4 = P2
102	b	XXY	2017-11-28	hash(b)%4 = P4
103	c	XYZ	2017-11-29	hash(c)%4 = P3
104	d	XYX	2017-11-27	hash(d)%4 = P2
105	e	XYY	2017-11-29	hash(e)%4 = P1

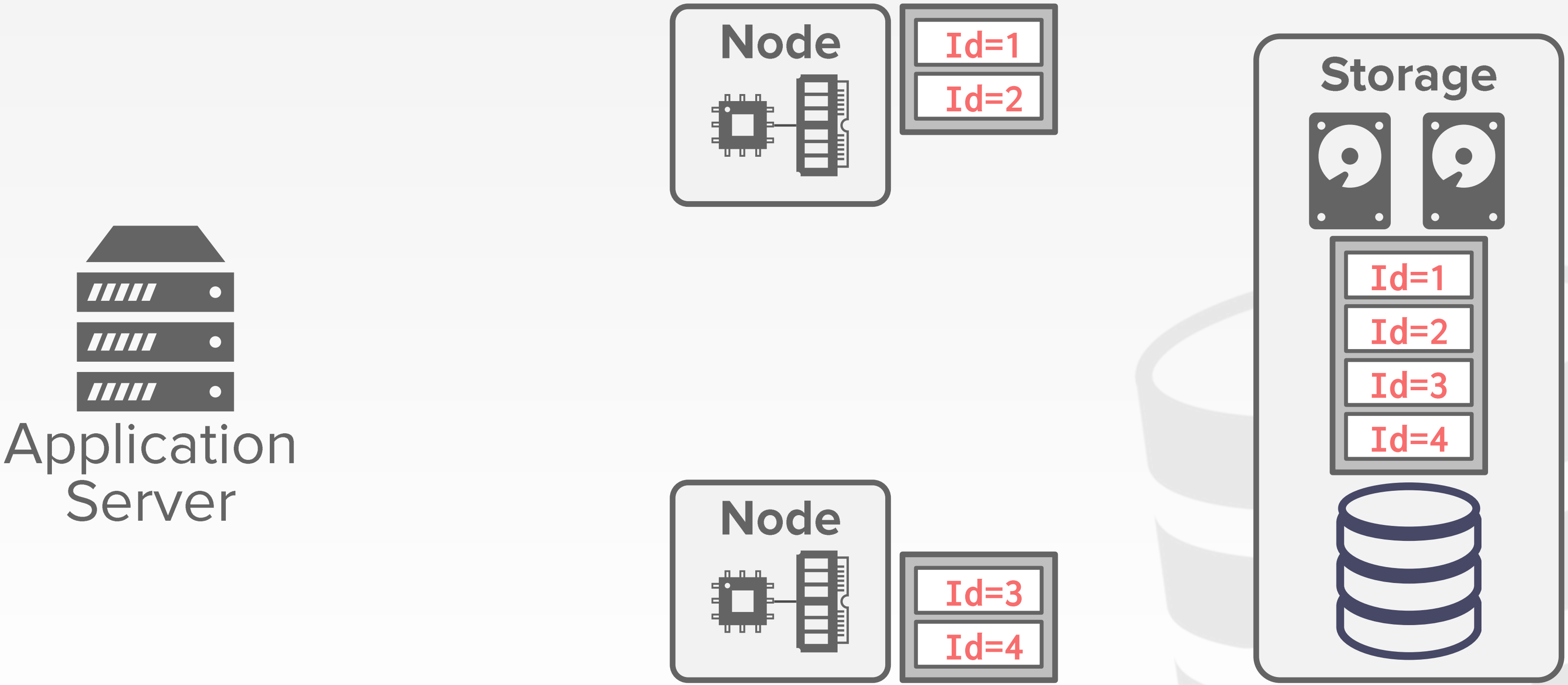
Partitions



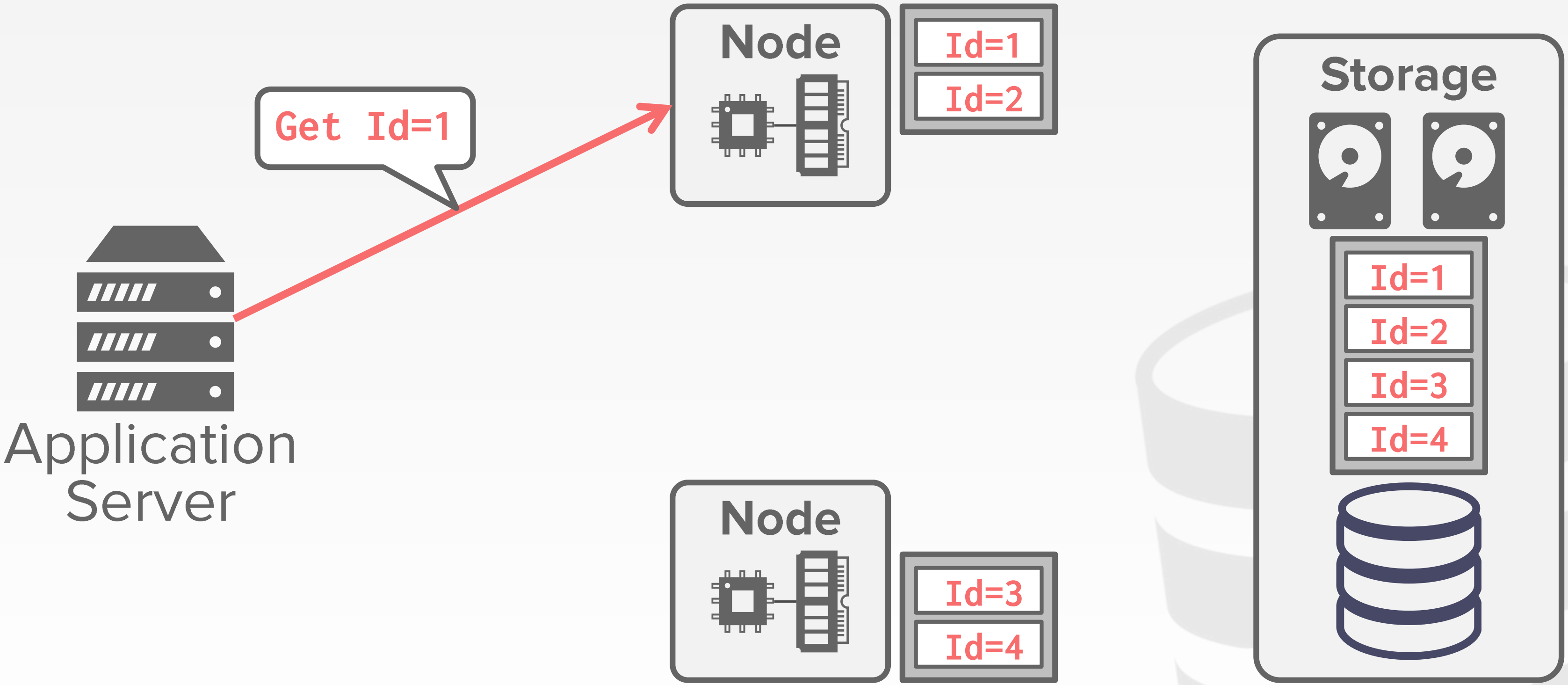
Ideal Query:

```
SELECT * FROM table
 WHERE partitionKey = ?
```

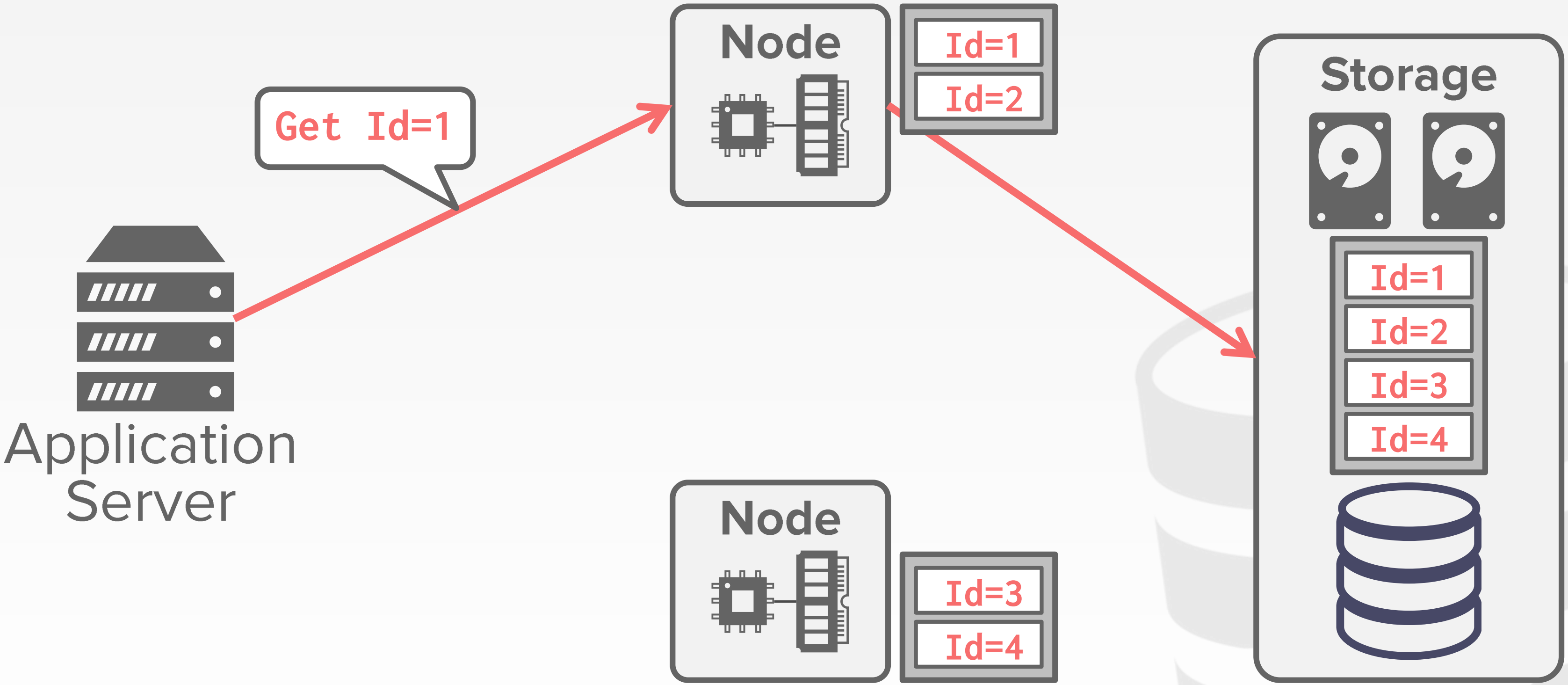
LOGICAL PARTITIONING



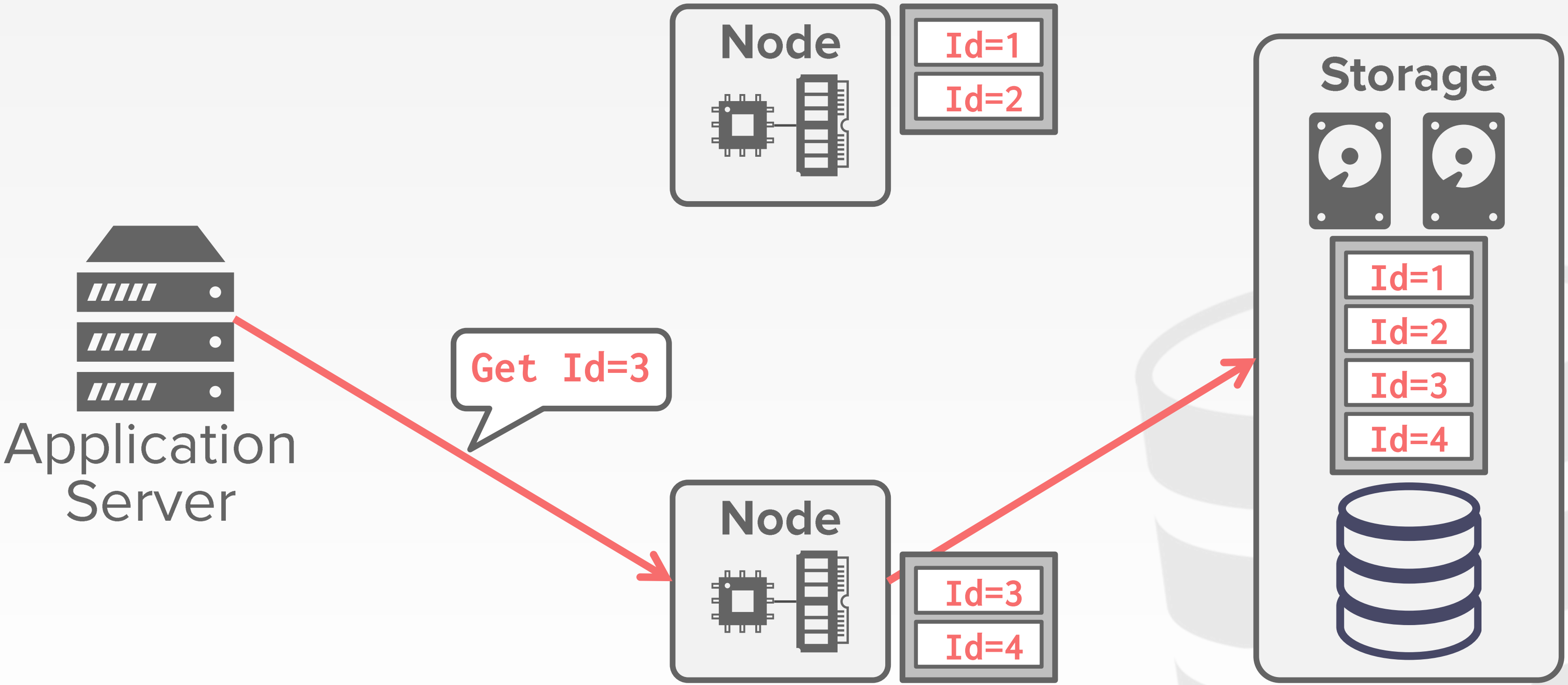
LOGICAL PARTITIONING



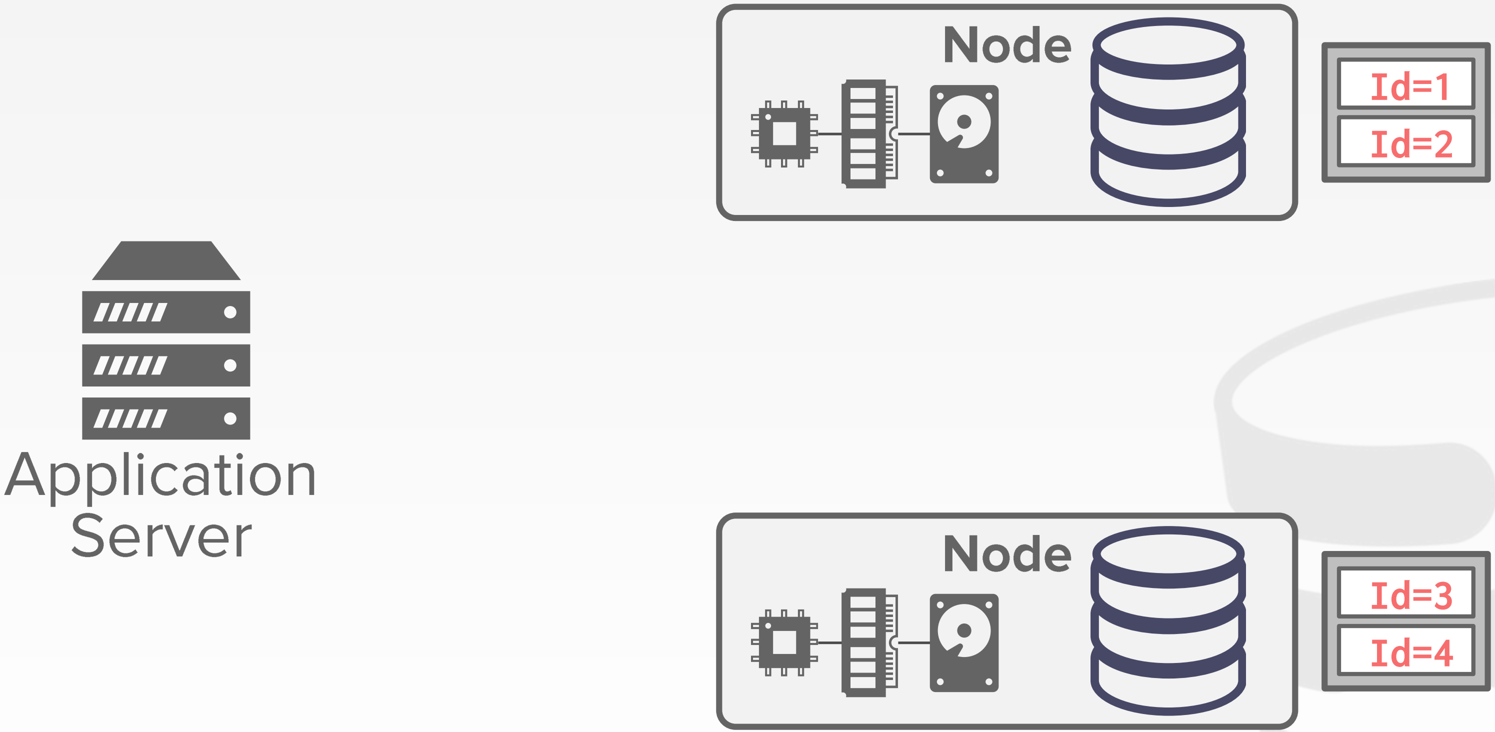
LOGICAL PARTITIONING



LOGICAL PARTITIONING

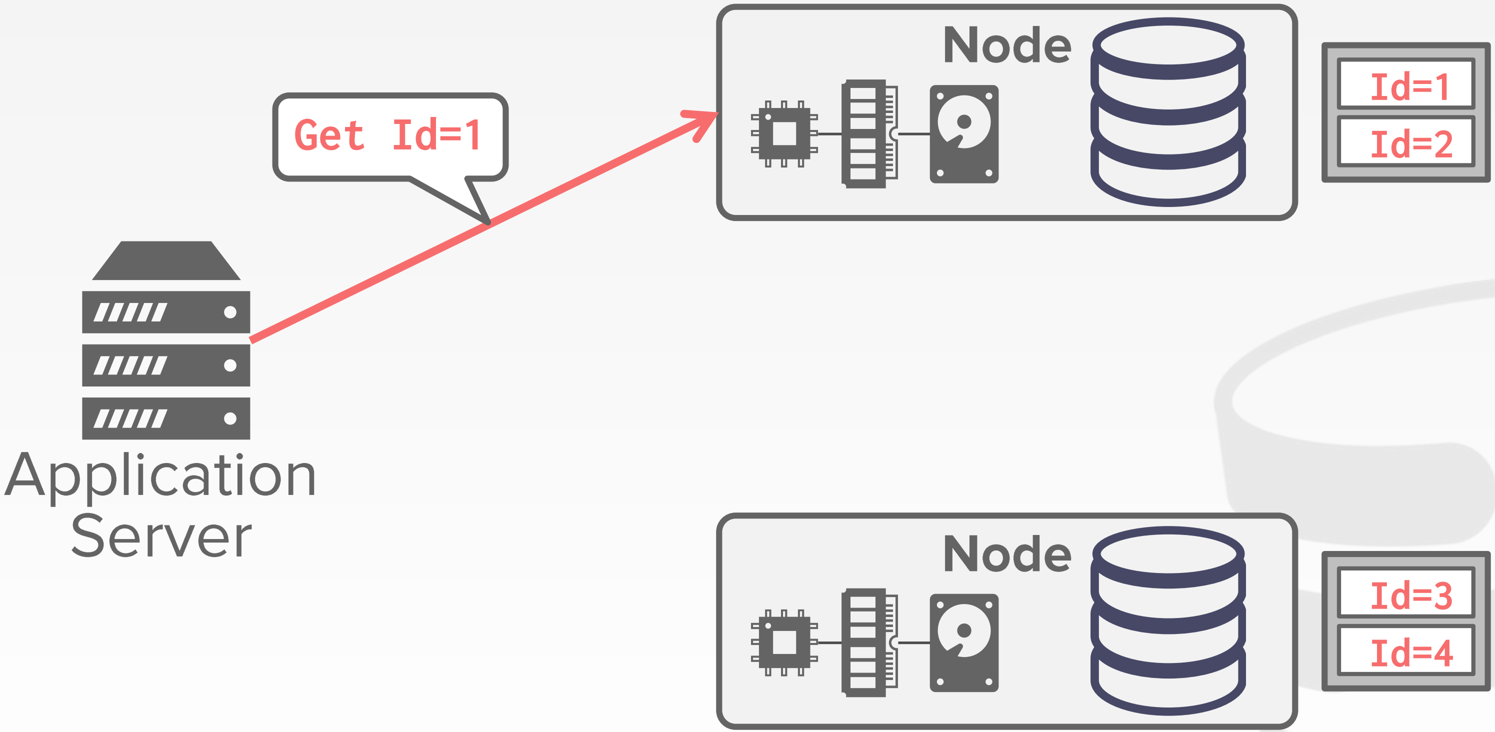


PHYSICAL PARTITIONING

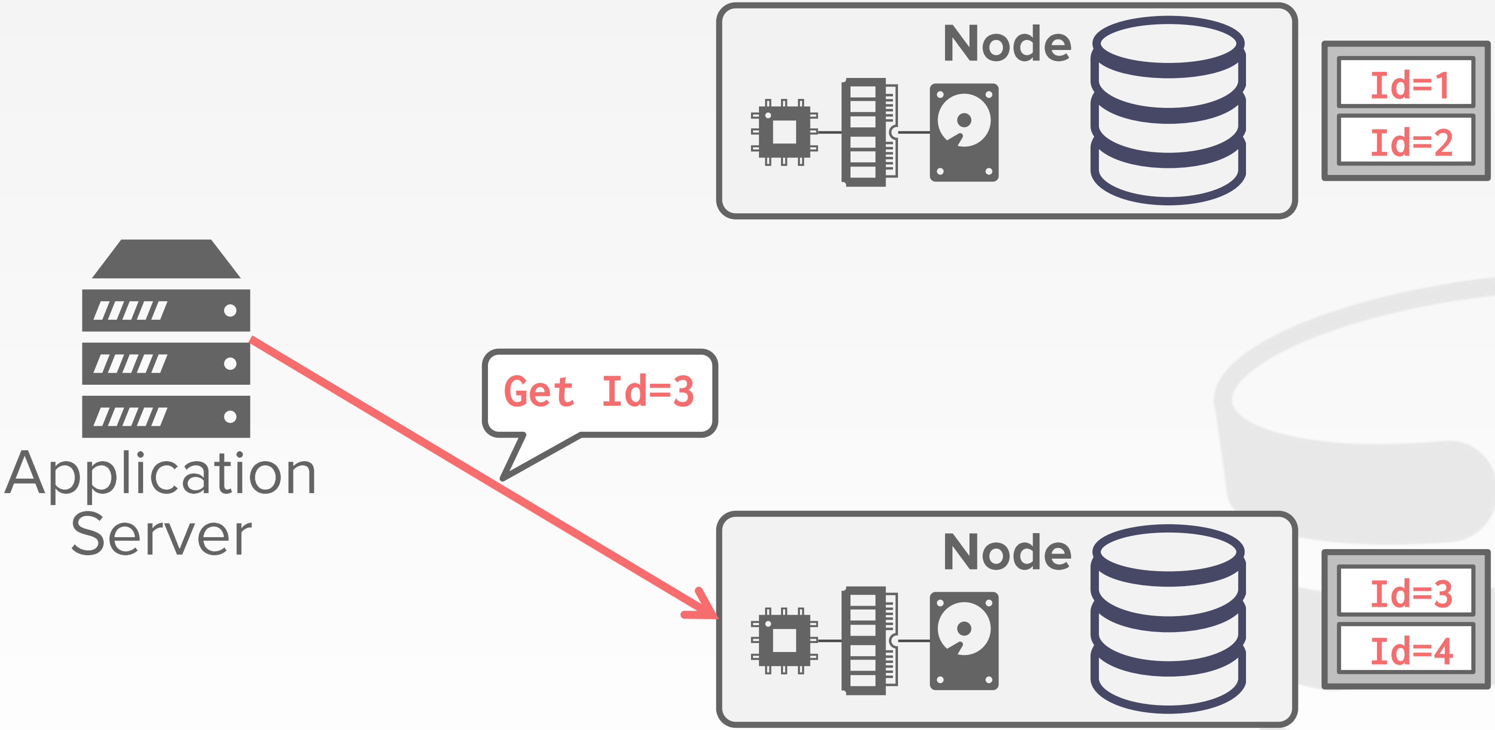


CARNEGIE MELLON
DATABASE GROUP

PHYSICAL PARTITIONING



PHYSICAL PARTITIONING



OBSERVATION

The efficiency of a distributed join depends on the target tables' partitioning schemes.

One approach is to put entire tables on a single node and then perform the join.
→ You lose the parallelism of a distributed DBMS.
→ Costly data transfer over the network.

DISTRIBUTED JOIN ALGORITHMS

To join tables A and B, the DBMS needs to get the proper tuples on the same node.

Once there, it then executes the same join algorithms that we discussed earlier in the semester.

SCENARIO #1

One table is replicated at every node.
Each node joins its local data and then sends their results to a coordinating node.

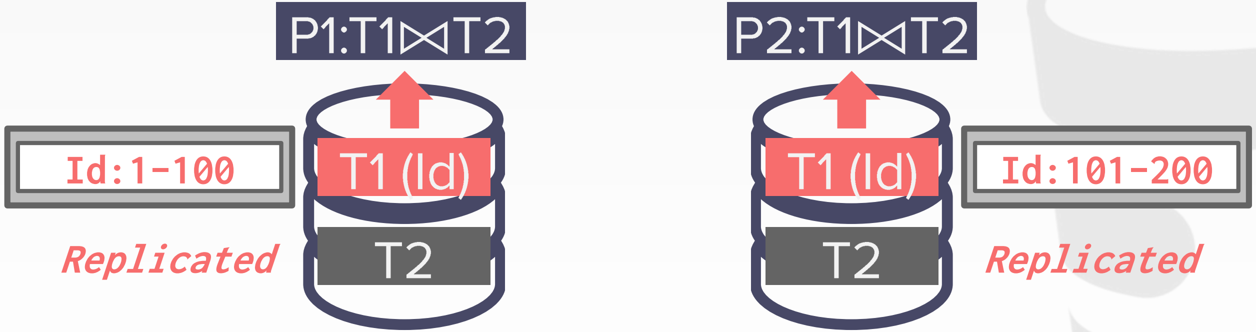
```
SELECT * FROM T1, T2
 WHERE T1.id = T2.id
```

CARNEGIE MELLON
DATABASE GROUP

SCENARIO #1

One table is replicated at every node. Each node joins its local data and then sends their results to a coordinating node.

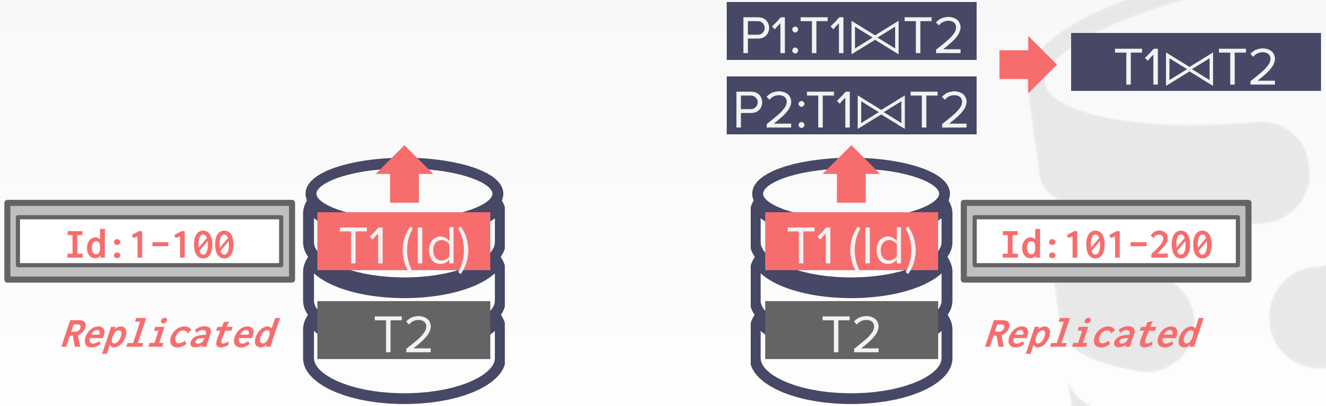
```
SELECT * FROM T1, T2
 WHERE T1.id = T2.id
```



SCENARIO #1

One table is replicated at every node. Each node joins its local data and then sends their results to a coordinating node.

```
SELECT * FROM T1, T2
 WHERE T1.id = T2.id
```



SCENARIO #2

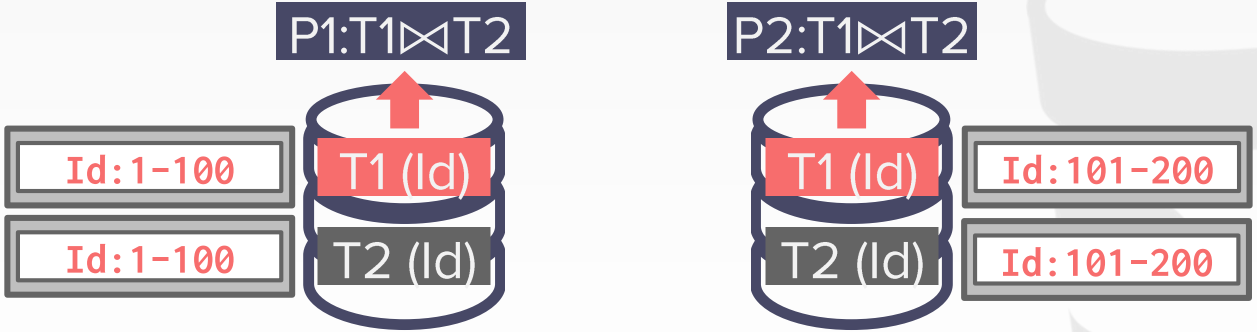
Tables are partitioned on the join attribute. Each node performs the join on local data and then sends to a node for coalescing.

```
SELECT * FROM T1, T2
 WHERE T1.id = T2.id
```

SCENARIO #2

Tables are partitioned on the join attribute. Each node performs the join on local data and then sends to a node for coalescing.

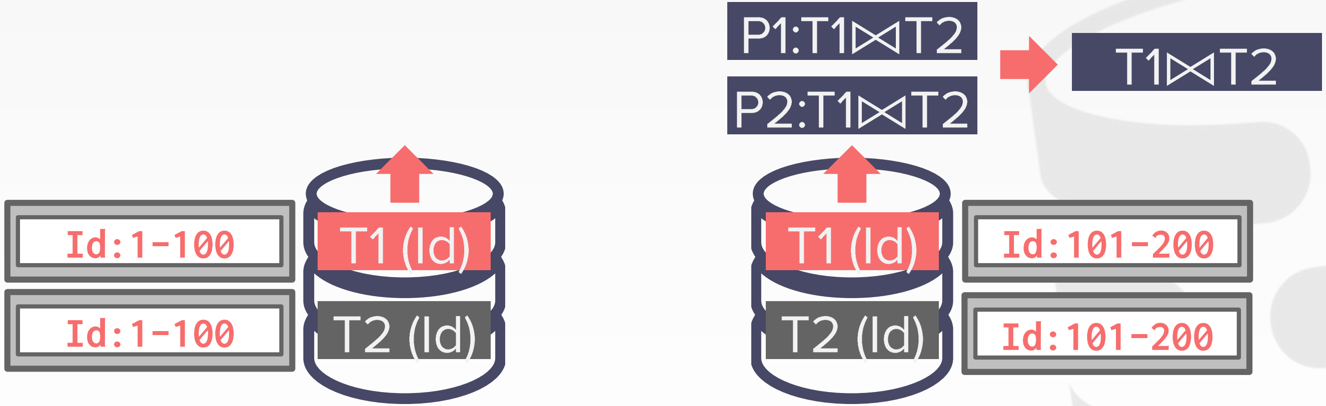
```
SELECT * FROM T1, T2
 WHERE T1.id = T2.id
```



SCENARIO #2

Tables are partitioned on the join attribute. Each node performs the join on local data and then sends to a node for coalescing.

```
SELECT * FROM T1, T2
 WHERE T1.id = T2.id
```



SCENARIO #3

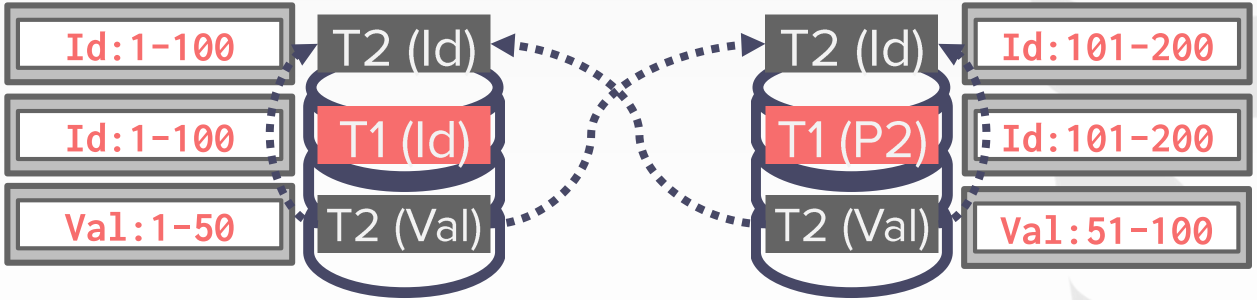
Both tables are partitioned on different keys. If one of the tables is small, then the DBMS **broadcasts** that table to all nodes.

```
SELECT * FROM T1, T2
 WHERE T1.id = T2.id
```

SCENARIO #3

Both tables are partitioned on different keys. If one of the tables is small, then the DBMS **broadcasts** that table to all nodes.

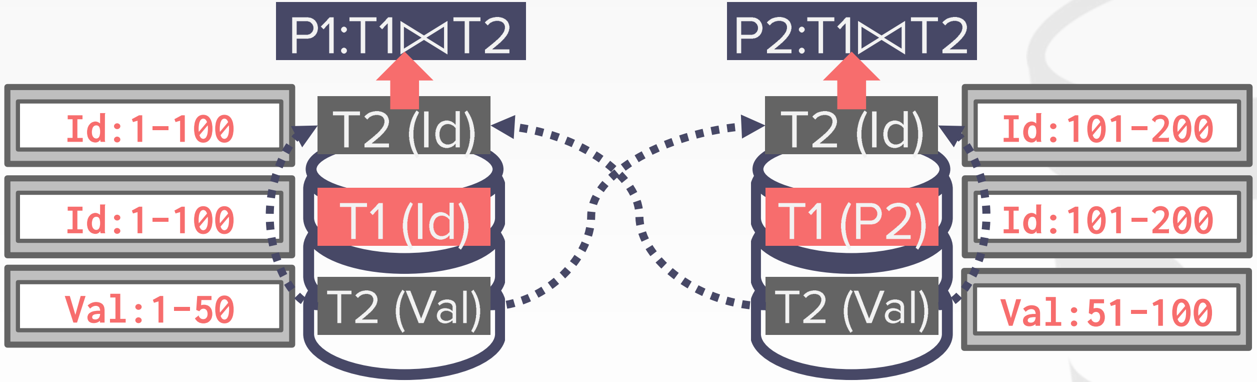
```
SELECT * FROM T1, T2
 WHERE T1.id = T2.id
```



SCENARIO #3

Both tables are partitioned on different keys. If one of the tables is small, then the DBMS **broadcasts** that table to all nodes.

```
SELECT * FROM T1, T2
 WHERE T1.id = T2.id
```

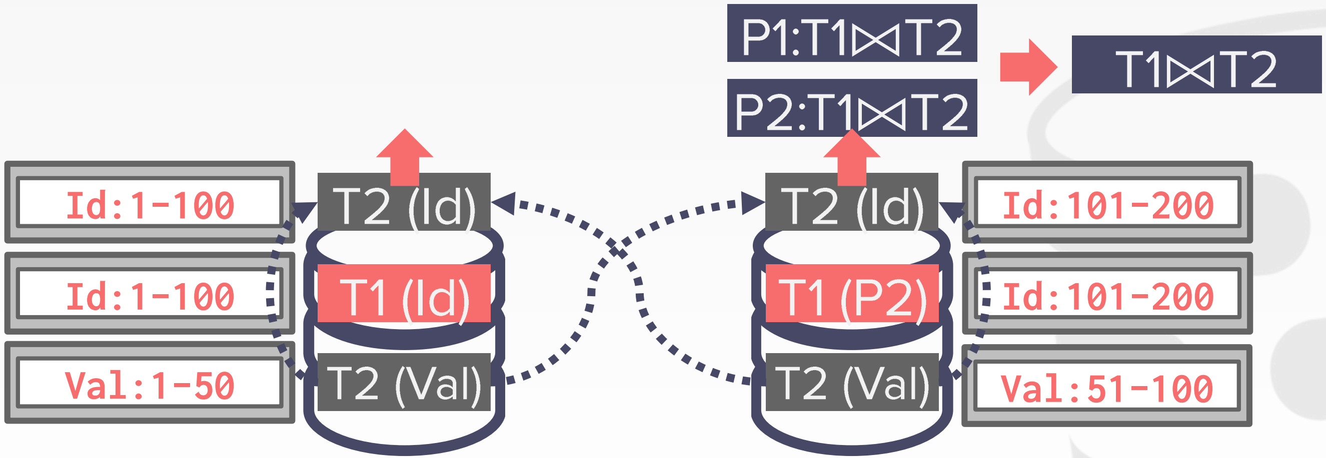


CARNEGIE MELLON DATABASE GROUP

SCENARIO #3

Both tables are partitioned on different keys. If one of the tables is small, then the DBMS **broadcasts** that table to all nodes.

```
SELECT * FROM T1, T2
 WHERE T1.id = T2.id
```



SCENARIO #4

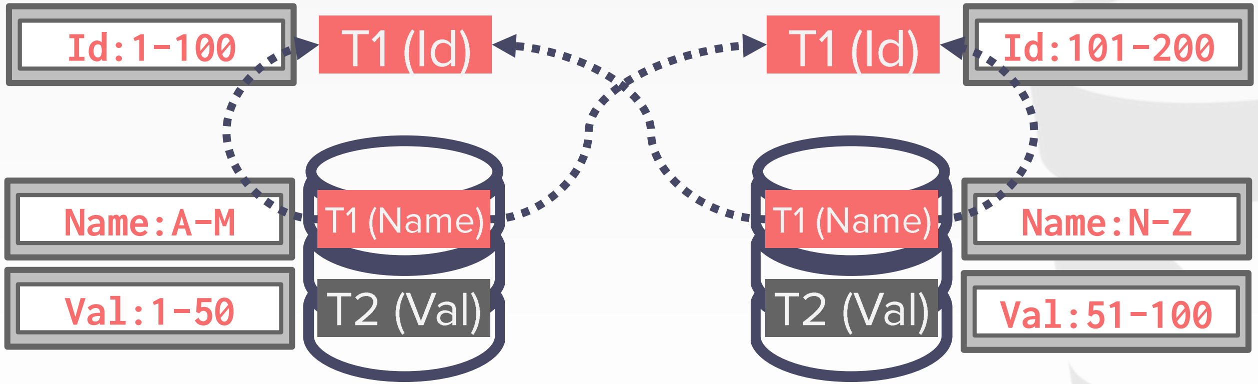
Both tables are not partitioned on the join key. The DBMS copies the tables by **reshuffling** them across nodes.

```
SELECT * FROM T1, T2
 WHERE T1.id = T2.id
```

SCENARIO #4

Both tables are not partitioned on the join key. The DBMS copies the tables by **reshuffling** them across nodes.

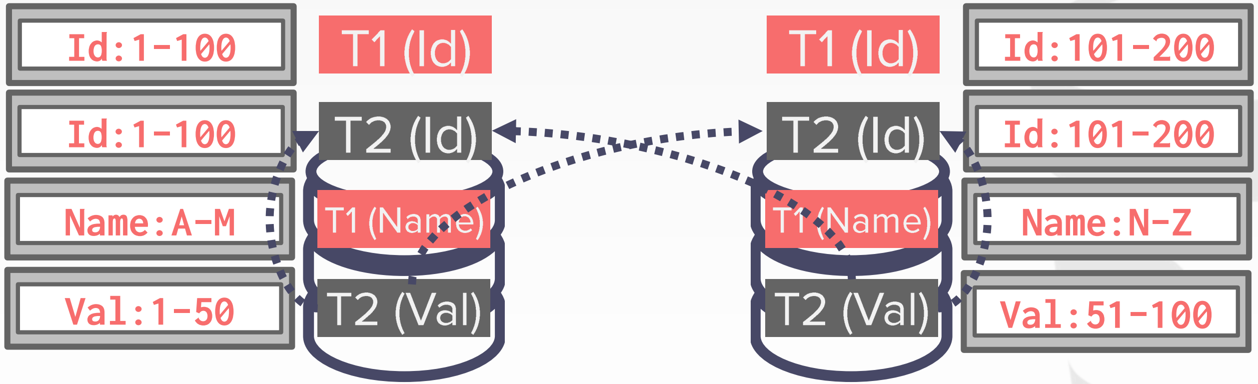
```
SELECT * FROM T1, T2
 WHERE T1.id = T2.id
```



SCENARIO #4

Both tables are not partitioned on the join key. The DBMS copies the tables by **reshuffling** them across nodes.

```
SELECT * FROM T1, T2
 WHERE T1.id = T2.id
```

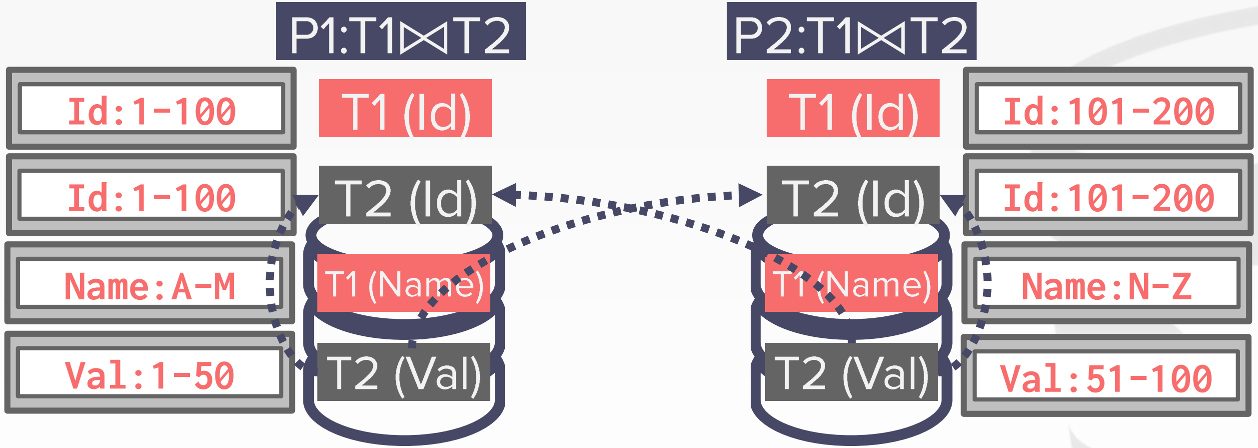


CARNEGIE MELLON DATABASE GROUP

SCENARIO #4

Both tables are not partitioned on the join key. The DBMS copies the tables by **reshuffling** them across nodes.

```
SELECT * FROM T1, T2
 WHERE T1.id = T2.id
```

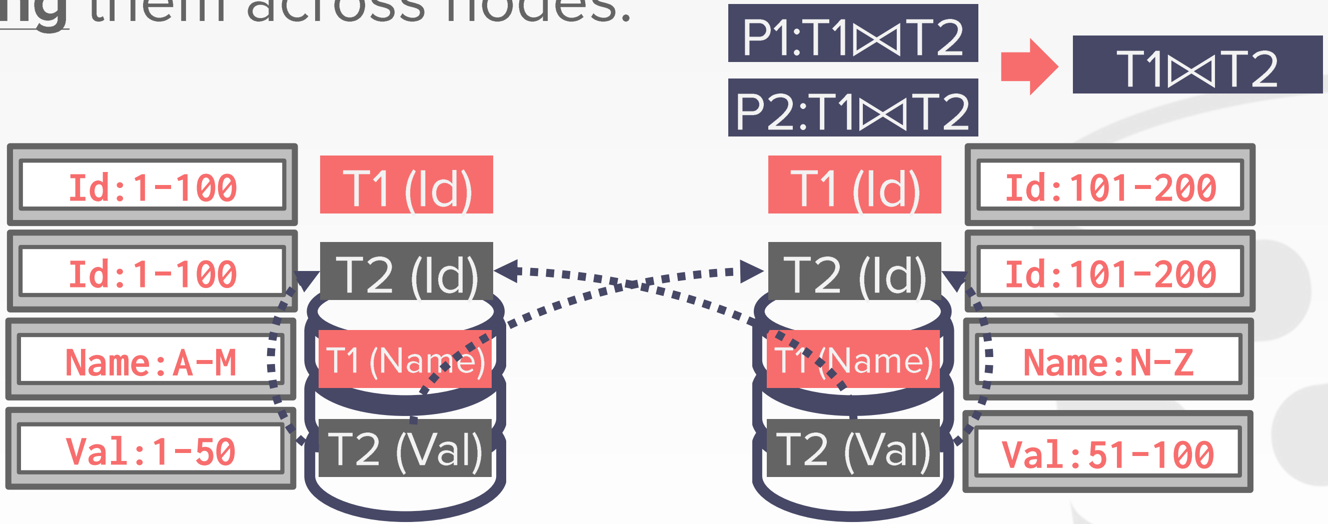


CARNEGIE MELLON DATABASE GROUP

SCENARIO #4

Both tables are not partitioned on the join key. The DBMS copies the tables by <u>**reshuffling**</u> them across nodes.

```
SELECT * FROM T1, T2
 WHERE T1.id = T2.id
```



CONCLUSION

Again, efficient distributed OLAP systems are difficult to implement.

Whenever possible, you want to push the query to the data rather than pull the data to the query.

NEXT CLASS

NuoDB!