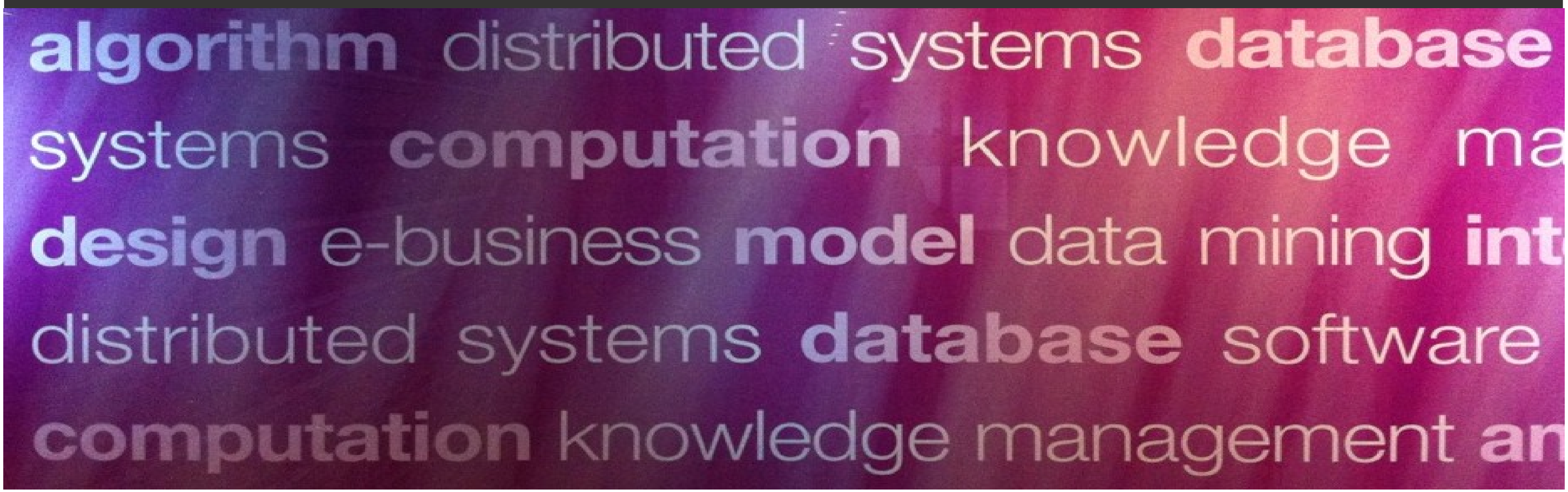
MONASH University

Information Technology

FIT5202 (Volume III - Join)

Week 3b – Parallel Outer Join



Join Queries

- Two types of Join Queries
  - **Inner Join**

```
Select R.x, R.a, S.y, S.b
From R, S
Where R.a = S.b;
```

  - **Outer Join**

```
Select R.x, R.a, S.y, S.b
From R left outer join S on R.a = S.b;
```

Join Queries

R

x	a
0	1
1	2
2	3

S

y	b
0	6
0	4
3	1
6	2
1	6
4	2
7	2
7	1
2	1
5	5
5	6
8	9

Results

x	a	y	b
0	1	3	1
0	1	7	1
0	1	2	1
1	2	6	2
1	2	4	2
1	2	7	2

- **Inner Join**

  Select R.x, R.a, S.y, S.b

  From R, S

  Where R.a = S.b;

Join Queries

- **Outer Join**

  Select R.x, R.a, S.y, S.b
  From R left outer join S
  On R.a = S.b;

R

x	a
0	1
1	2
2	3

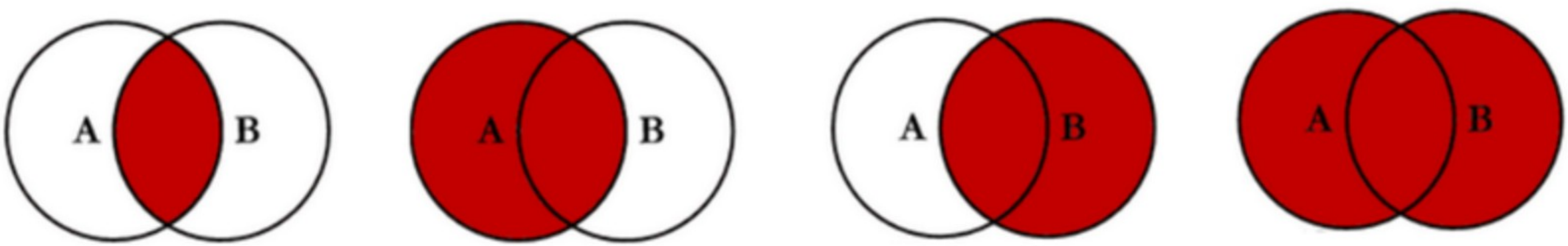
S

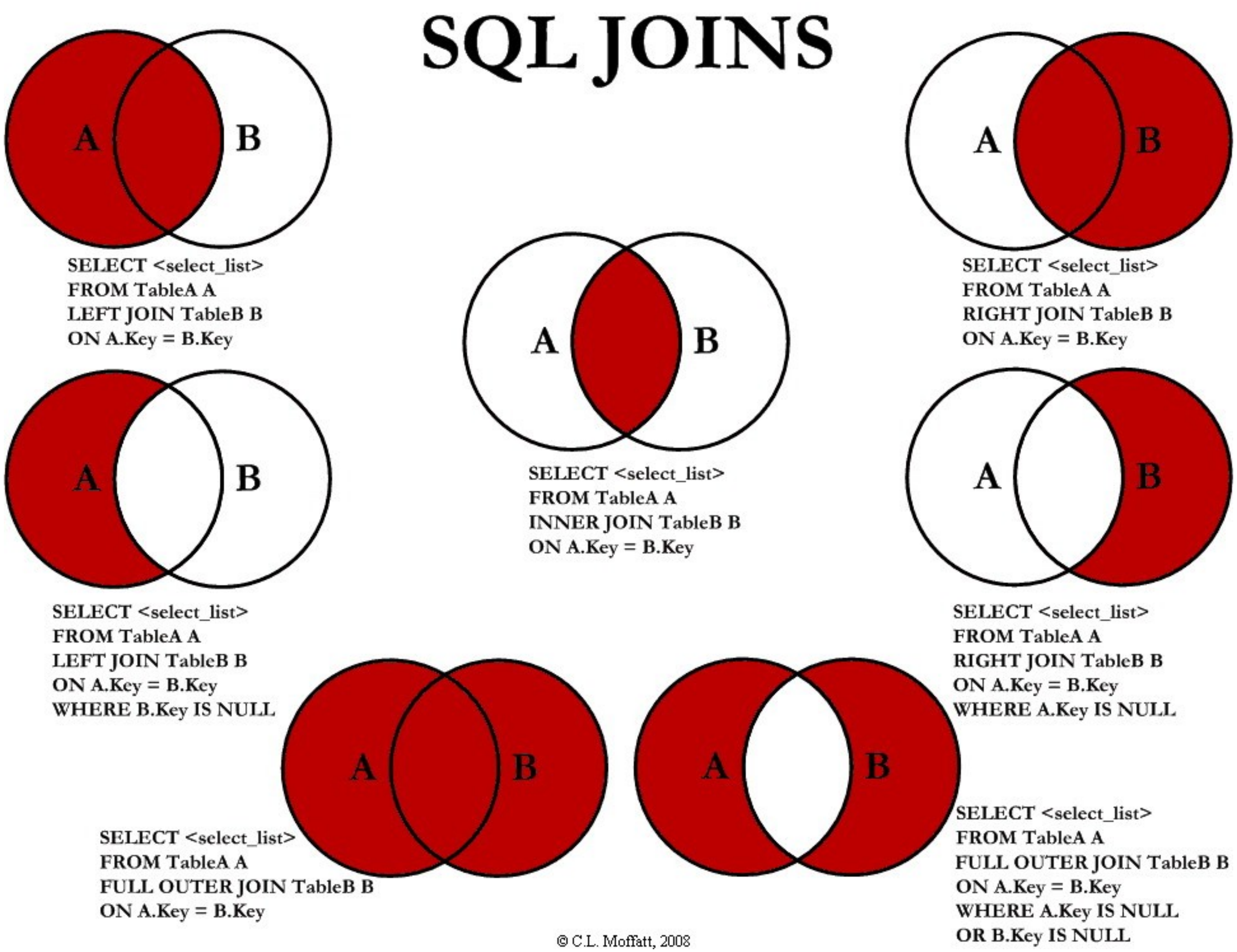
y	b
0	6
0	4
3	1
6	2
1	6
4	2
7	2
7	1
2	1
5	5
5	6
8	9

Results

x	a	y	b
0	1	3	1
0	1	7	1
0	1	2	1
1	2	6	2
1	2	4	2
1	2	7	2
2	3	Null	Null

- **Exercise 1**
  - Identify the LEFT OUTER JOIN?





https://www.codeproject.com/Articles/33052/Visual-Representation-of-SQL-Joins

Parallel Join Query Processing

- Parallel Inner Join components
  - **Data Partitioning**
    - Divide and Broadcast
    - Disjoint Partitioning
  - **Local Join**
    - Nested-Loop Join
    - Sort-Merge Join
    - Hash Join

- Example of a Parallel Inner Join Algorithm
  - **Divide and Broadcast, plus Hash Join**

Parallel Join Query Processing

- Parallel Outer Join processing methods
  - **ROJA** (Redistribution Outer Join Algorithm)
  - **DOJA** (Duplication Outer Join Algorithm)
  - **DER** (Duplication & Efficient Redistribution)

- Load Balancing
  - **OJSO** (Outer Join Skew Optimization)

1. ROJA

Step 1: Distribute or reshuffle data based on join attribute.
Step 2: Each processor performs local outer Join.

SELECT R.x, R.a, S.y, S.b
FROM R left outer join S on R.a = S.b

Eg. Using hash func h(i) = i mod 3 + 1

h(i)= 1

R: x	R: a	S: y	S: b
0	1	0	6
		0	4
		3	1
		6	2

Processor 1

h(i)= 2

R: x	R: a	S: y	S: b
1	2	1	6
		4	2
		7	2
		7	1

Processor 2

h(i)= 3

R: x	R: a	S: y	S: b
2	3	2	1
		5	5
		5	6
		8	9

Processor 3

2. DOJA

Step 1: Replicate small table.
Step 2: Local Inner Join
Step 3: Hash redistribute inner join result based on attribute x.

SELECT R.x, R.a, S.y, S.b
FROM R left outer join S on R.a = S.b

Processor 1

R: x	R: a	S: y	S: b
0	1	0	6
1	2	0	4
2	3	3	1
		6	2

Processor 2

R: x	R: a	S: y	S: b
1	2	1	6
0	1	4	2
2 / 4	3 / 2	7	2
		7	1

Processor 3

R: x	R: a	S: y	S: b
2	3	2	1
1	2	5	5
0	1	5	6
		8	9

2. DOJA

Step 4: Local outer join

SELECT R.x, R.a, S.y, S.b
FROM R left outer join S on R.a = S.b

Processor 1

R		J			
x	a	x	a	y	b
0	1	0	1	3	1
		0	1	7	1
		0	1	2	1

Processor 2

R		J			
x	a	x	a	y	b
1	2	1	2	6	2
		1	2	4	2
		1	2	7	2

Processor 3

R		J			
x	a	x	a	y	b
2	3			N	N

3. DER

SELECT R.x, R.a, S.y, S.b
FROM R left outer join S on R.a = S.b

Step 1: Replicate small table (left)
Step 2: Local Inner Join
Step 3: Select ROW ID of left table with no matches.
Step 4: Redistribute the ROW ID.
Step 5: Store the ROW ID that appears as many times as the number of processors

Processor 1

Row ID	R: x	R: a	S: y	S: b
0	0	1	0	6
1	1	2	0	4
2	2	3	3	1
			6	2

Processor 2

Row ID	R: x	R: a	S: y	S: b
0	0	1	1	6
1	1 1	2 2	4	2
2	2	3	7	2
			7	1

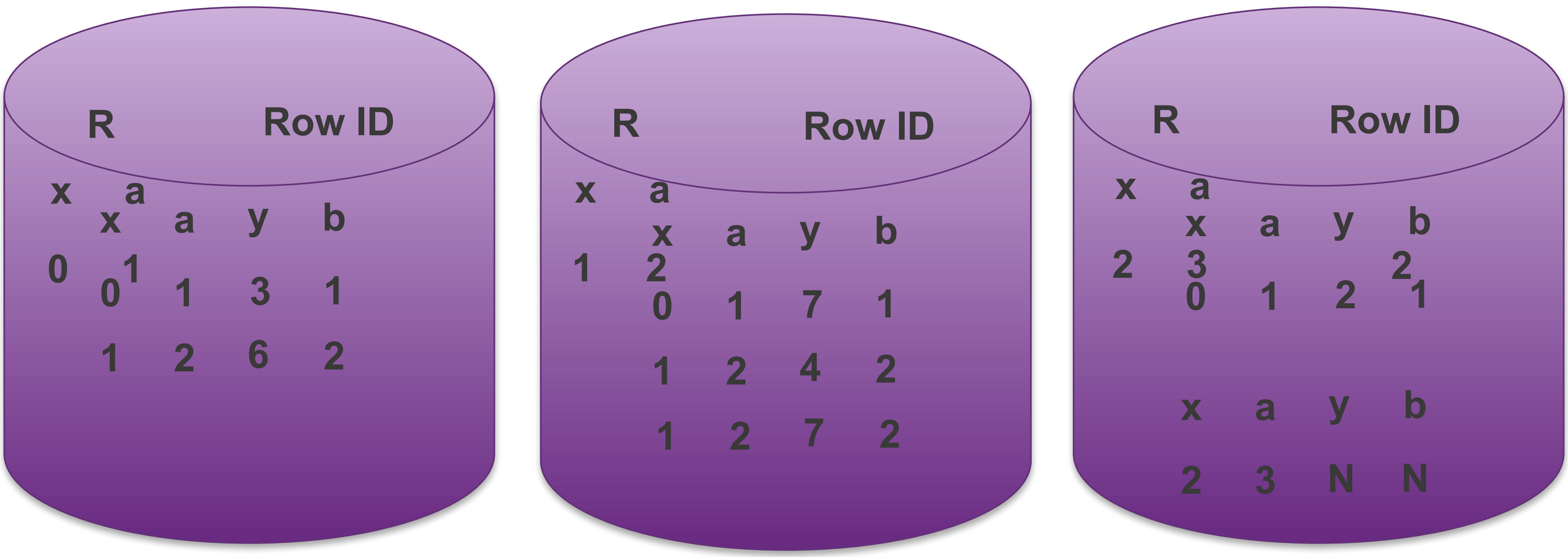
Processor 3

Row ID	R: x	R: a	S: y	S: b
0	0	1	2	1
1	1	2	5	5
2	2	3	5	6
			8	9

3. DER

Step 6: Inner join

SELECT R.x, R.a, S.y, S.b
FROM R left outer join S on R.a = S.b



Processor 1 **Processor 2** **Processor 3**

Parallel Join Query Processing

- Parallel Outer Join processing methods
  - **ROJA** (Redistribution Outer Join Algorithm)
  - **DOJA** (Duplication Outer Join Algorithm)
  - **DER** (Duplication & Efficient Redistribution)

- Load Balancing
  - **OJSO** (Outer Join Skew Optimization)

	ROJA	DOJA	DER
Steps	**Step 1:** Distribute or reshuffle the data based on the join attribute. **Step 2:** Each processor performs the Local outer Join.	**Step 1:** Replication. We duplicate the small table. **Step 2:** Local Inner Join **Step 3:** Hash redistribute the inner join result based on attribute X. **Step 4:** Local outer join	**Step 1:** Replication. We broadcast the left table. **Step 2:** Local Inner Join **Step 3:** Select the ROW ID of left table with no matches. **Step 4:** Redistribute the ROW ID. **Step 5:** Store the ROW ID that appears as many times as the number of processors. **Step 6:** Inner join
Pros	fast performance, only two steps	None. ROJA is faster than DOJA.	Redistributes dangling row IDs instead of actual records.
Cons	redistribution of data -> data skew, communication cost	In the replication step, if the table is large, the replication cost is expensive. In the distribution step, data skew and communication cost similar to ROJA	In the replication step, if the table is large, the replication cost is expensive.

Another example…

```
Select x, y, z, a, c
From R left outer join S on R.a=S.b
       left outer join T on S.c=T.d;
```

- Initial Data Placement

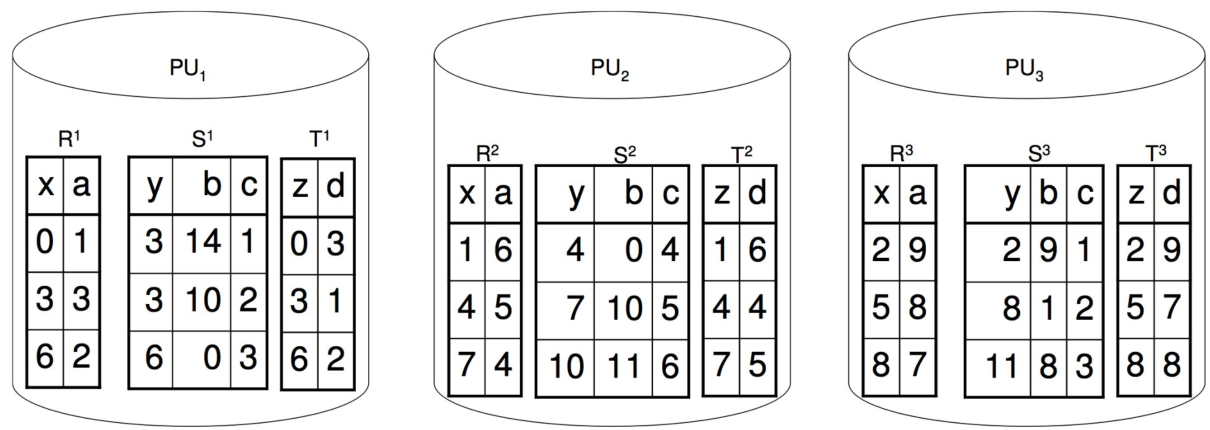
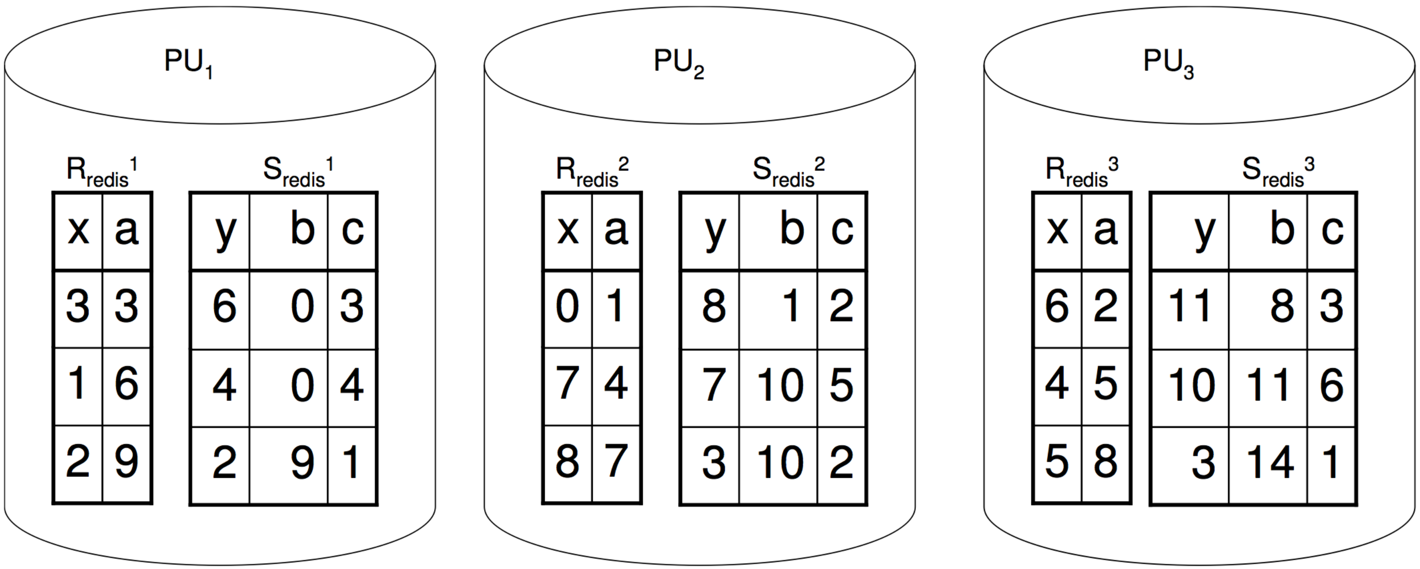


Figure 1: Three relations R, S and T are hash partitioned on a three parallel-unit system. The partitioning columns are $R.x$, $S.y$ and $T.z$ respectively. The hash function, $h(i) = i \ mod \ 3 + 1$, places a tuple with value i in the partitioning column on the $h(i)$-th PU.

Another example…

- Step 1: Redistribution of R and S (**why do we need to redistribute?**)



PU1

R_{redis}^{1}

x	a
3	3
1	6
2	9

S_{redis}^{1}

y	b	c
6	0	3
4	0	4
2	9	1

PU2

R_{redis}^{2}

x	a
0	1
7	4
8	7

S_{redis}^{2}

y	b	c
8	1	2
7	10	5
3	10	2

PU3

R_{redis}^{3}

x	a
6	2
4	5
5	8

S_{redis}^{3}

y	b	c
11	8	3
10	11	6
3	14	1

Figure 2: The result of hash redistributing R and S on their join attributes ($R.a$ and $S.b$) to two temporary tables R_{redis} and S_{redis}.

Another example…

- Step 2: (a) Outer Join R and S, and store in J

PU$_1$

J^1

x	a	y	c
3	3	$\perp$	$\perp$
1	6	$\perp$	$\perp$
2	9	2	1

PU$_2$

J^2

x	a	y	c
0	1	8	2
7	4	$\perp$	$\perp$
8	7	$\perp$	$\perp$

PU$_3$

J^3

x	a	y	c
6	2	$\perp$	$\perp$
4	5	$\perp$	$\perp$
5	8	11	3

Figure 3: The results of left outer joining R_{redis} and S_{redis} (R_{redis} and S_{redis} are shown in Figure 2) are stored in a temporary table J.

Another example…

- Step 2: (b) Redistribute J and T

PU_1

J_{redis}^1

x	a	y	c
5	8	11	3
3	3	⊥	⊥
1	6	⊥	⊥
7	4	⊥	⊥
8	7	⊥	⊥
6	2	⊥	⊥
4	5	⊥	⊥

T_{redis}^1

z	d
0	3
1	6
2	9

PU_2

J_{redis}^2

x	a	y	c
2	9	2	1

T_{redis}^2

z	d
3	1
4	4
5	7

PU_3

J_{redis}^3

x	a	y	c
0	1	8	2

T_{redis}^3

z	d
6	2
7	5
8	8

Figure 4: The result of hash redistributing J (shown in Figure 3) and T (shown in Figure 1) on their join attributes ($J.c$ and $T.d$) to two temporary tables J_{redis} and T_{redis}.

Another example…

- Step 3: Outer Join J and T ➔ Final Results

PU_1

F^1

x	a	y	c	z
5	8	11	3	0
3	3	$\perp$	$\perp$	$\perp$
1	6	$\perp$	$\perp$	$\perp$
7	4	$\perp$	$\perp$	$\perp$
8	7	$\perp$	$\perp$	$\perp$
6	2	$\perp$	$\perp$	$\perp$
4	5	$\perp$	$\perp$	$\perp$

PU_2

F^2

x	a	y	c	z
2	9	2	1	3

PU_3

F^3

x	a	y	c	z
0	1	8	2	6

Figure 5: The final results of the two outer joins in Query 1 are stored in a temporary table F.

Conclusion…

- Skew can easily happen easily in Outer Join queries

- **Exercise 1 (FLUX Quiz)**
  - In the previous example (R outer join S outer join T), which parallel outer join method was used?
  - A. ROJA
  - B. DOJA
  - C. DER
  - D. None of the above

A better solution… OJSO

- Step 1: Redistribute R and S (same as the previous example)

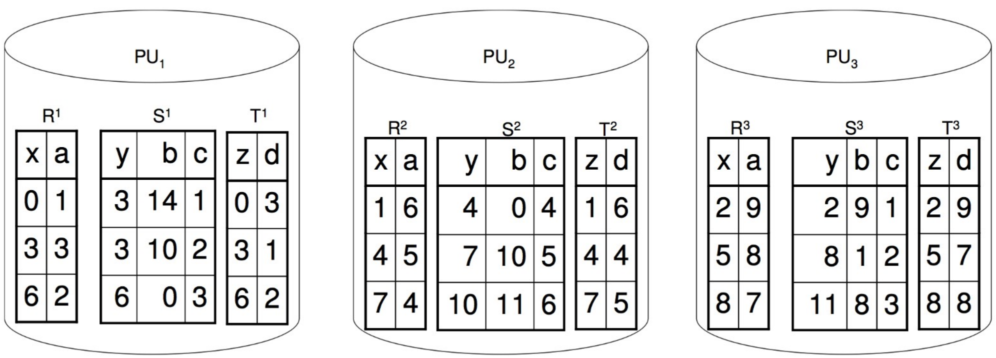


Figure 1: Three relations R, S and T are hash partitioned on a three parallel-unit system. The partitioning columns are $R.x$, $S.y$ and $T.z$ respectively. The hash function, $h(i) = i \ mod\ 3 + 1$, places a tuple with value i in the partitioning column on the $h(i)$-th PU.

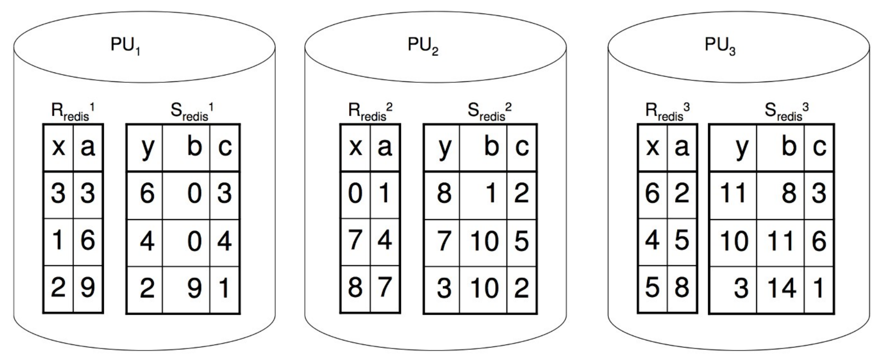


Figure 2: The result of hash redistributing R and S on their join attributes ($R.a$ and $S.b$) to two temporary tables R_{redis} and S_{redis}.

A better solution… OJSO

- Step 2: (a) Outer Join R and S, but the results are divided into J_{2redis} and J_{local}

PU_1

J_{2redis}^{1}

x	a	y	c
2	9	2	1

J_{local}^{1}

x	a	y	c
3	3	⊥	⊥
1	6	⊥	⊥

PU_2

J_{2redis}^{2}

x	a	y	C
0	1	8	2

J_{local}^{2}

x	a	y	c
7	4	⊥	⊥
8	7	⊥	⊥

PU_3

J_{2redis}^{3}

x	a	y	c
5	8	11	3

J_{local}^{3}

x	a	y	c
6	2	⊥	⊥
4	5	⊥	⊥

Figure 9: The results of left outer joining R_{redis} and S_{redis} are split into two temporary tables J_{2redis} and J_{local}.

A better solution… OJSO

- Step 2: (b) Redistribute J_{2redis} and T; and do an outer join

PU$_1$

J_{redis}^1

x	a	y	c
5	8	11	3

T_{redis}^1

z	d
0	3
1	6
2	9

$J_{locpadding}^1$

x	a	y	c	z
3	3	⊥	⊥	⊥
1	6	⊥	⊥	⊥

PU$_2$

J_{redis}^2

x	a	y	c
2	9	2	1

T_{redis}^2

z	d
3	1
4	4
5	7

$J_{locpadding}^2$

x	a	y	c	z
7	4	⊥	⊥	⊥
8	7	⊥	⊥	⊥

PU$_3$

J_{redis}^3

x	a	y	c
0	1	8	2

T_{redis}^3

z	d
6	2
7	5
8	8

$J_{locpadding}^3$

x	a	y	c	z
6	2	⊥	⊥	⊥
4	5	⊥	⊥	⊥

Figure 10: The result of hash redistributing J_{2redis} (shown in Figure 9) and T (shown in Figure 1) on their join attributes to two temporary tables J_{redis} and T_{redis}. $J_{locpadding}$ is created from J_{local} (shown in Figure 9) with padded nulls.

A better solution… OJSO

- Step 3: Union the final results in each processor

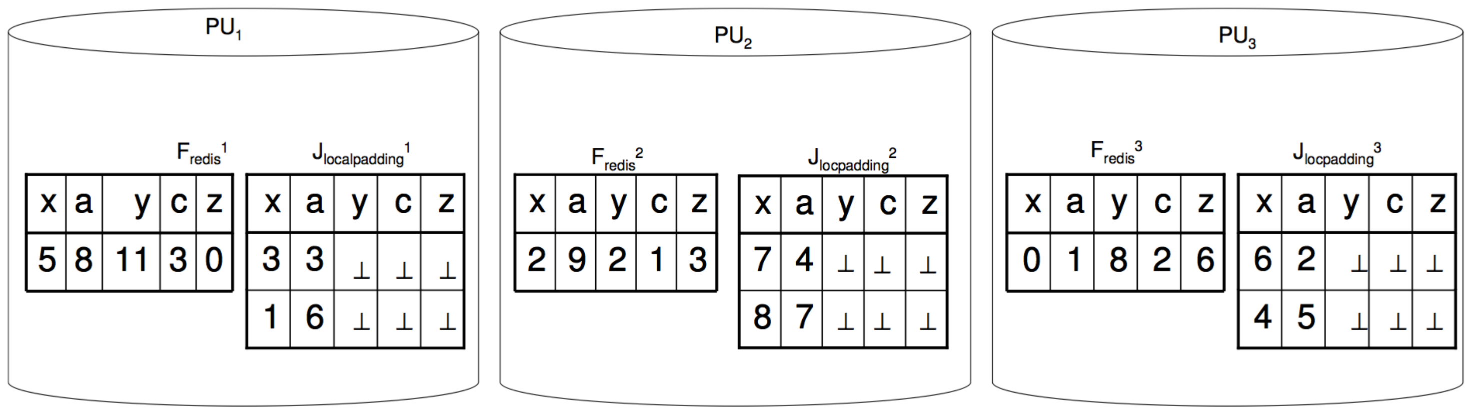


Figure 11: The results of the second outer join in Query 1 are stored in a temporary table F. The final result for Query 1 is the union of F and $J_{locpadding}$.

OJSO Conclusion…

- Do not redistribute the dangling records from the previous outer join

Summary…

- Parallel Outer Join processing methods
  - **ROJA** (Redistribution Outer Join Algorithm)
  - **DOJA** (Duplication Outer Join Algorithm)
  - **DER** (Duplication & Efficient Redistribution)

- Load Balancing
  - **OJSO** (Outer Join Skew Optimization)

References

- Xu, Y. & Kostamaa, P. (2010). A new algorithm for small-large table outer joins in parallel DBMS. In *Proceedings of the 26th Intl Conference on Data Engineering (ICDE'2010)* (pp. 1018-1024), IEEE Comp Society Press.
- Xu,Y. & Kostamaa, P. (2009). Efficient Outer Join Data Skew Handling in Parallel DBMS. In *Proceedings of the 35th International Conference on Very Large Data Bases (VLDB'2009)* (pp. 1390-1396), VLDB Endowment.