Algoritmos de Junção Estrela em MapReduce

Jaqueline Joice Brito

09 de junho de 2015

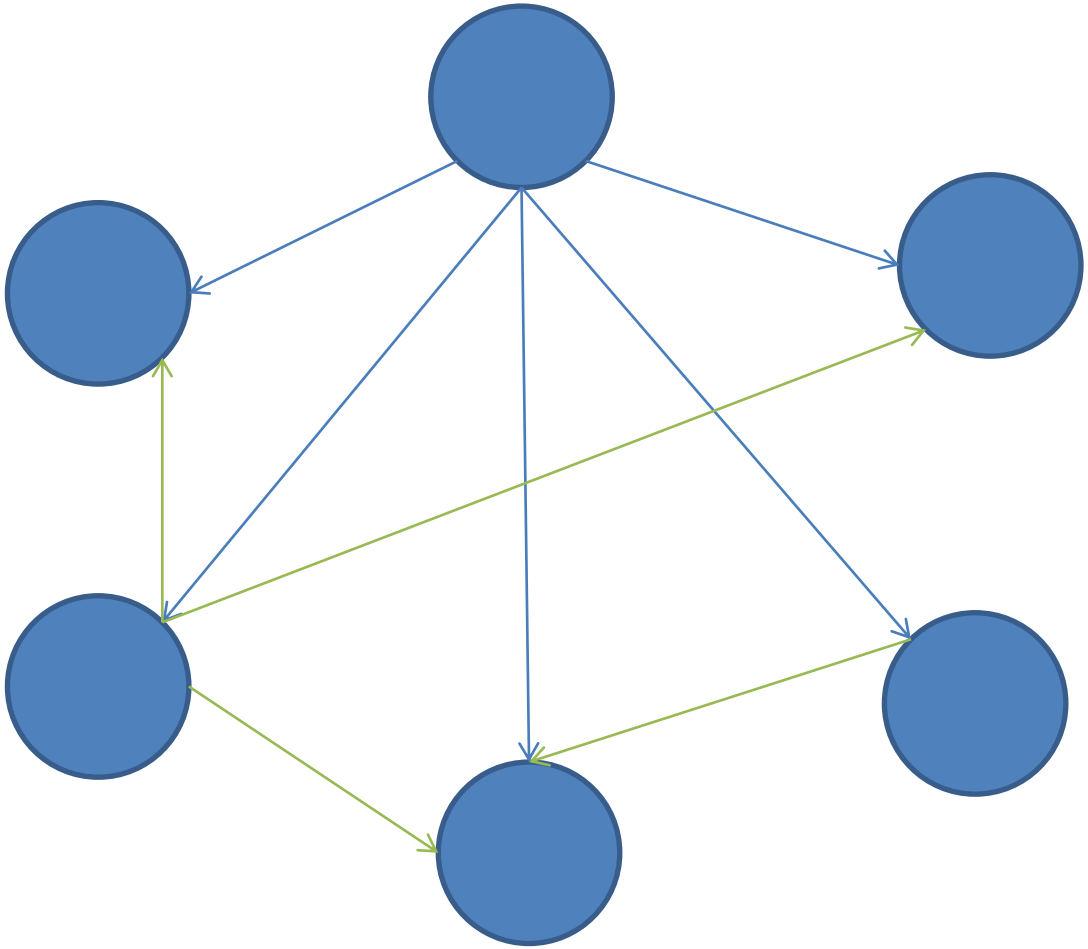
Modelo Relacional

- Dados armazenados em um conjunto de tabelas
- Amplamente utilizado
- Junção
  - Recuperação de dados de duas ou mais tabelas baseada em relações lógicas
  - Operação custosa

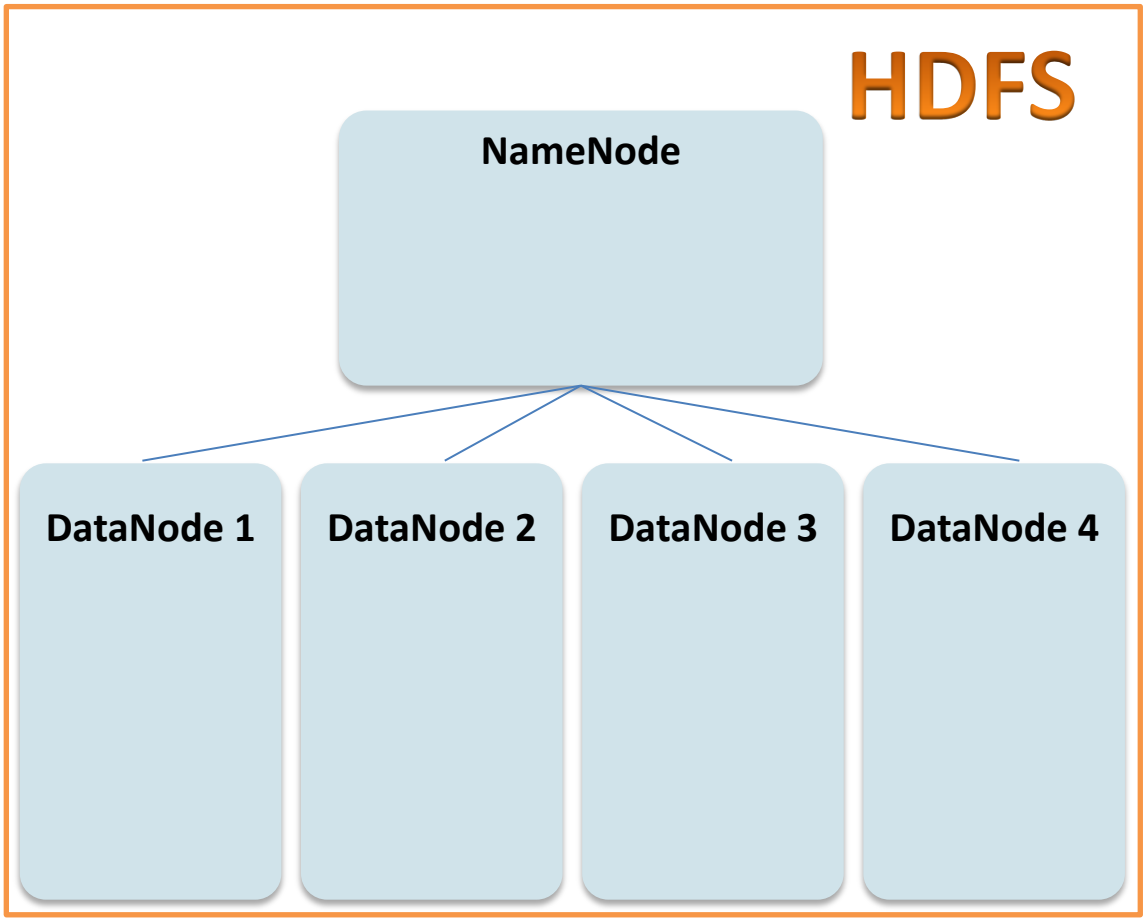
```
SELECT nome, cargo
FROM Cliente C, Profissao P
WHERE  C.cod_profissao = P.cod_profissao
```

Processamento Distribuído

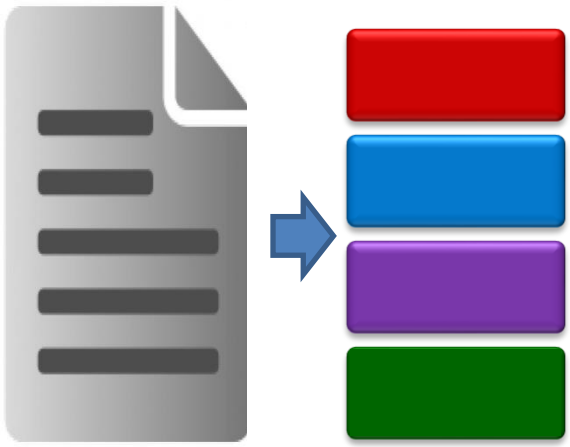
- Otimização
  - Minimização da comunicação entre os nós

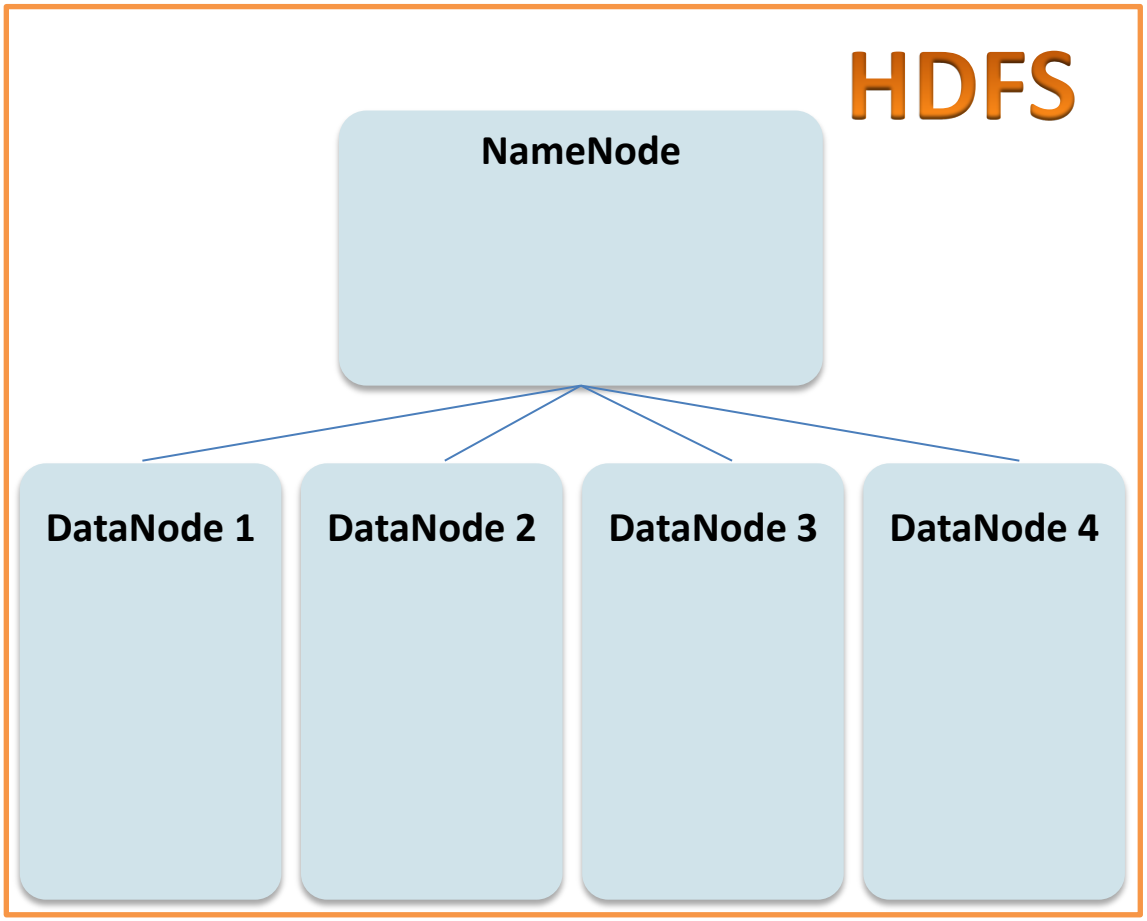


HDFS - Hadoop Distributed File System

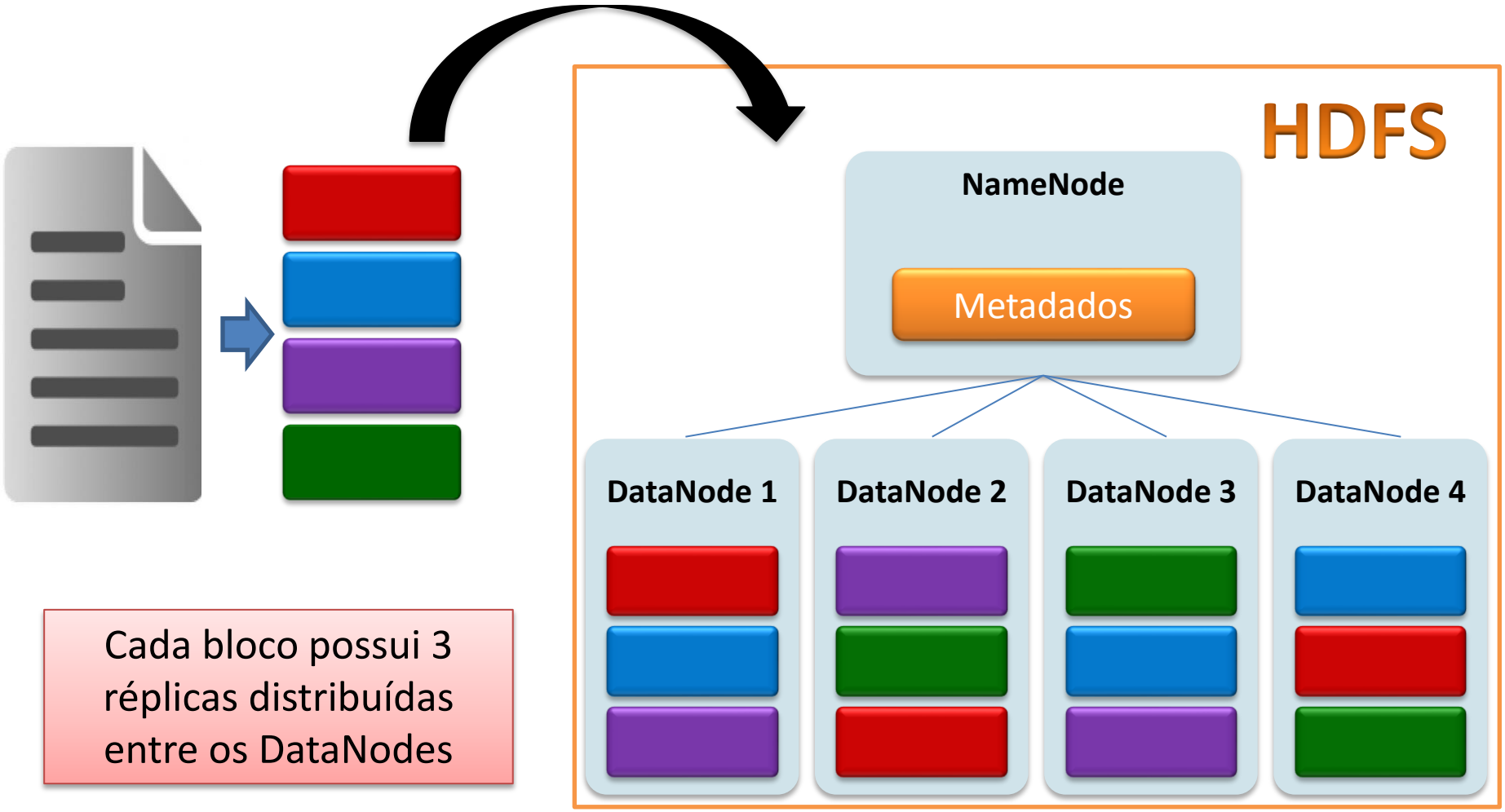


HDFS - Hadoop Distributed File System





HDFS - Hadoop Distributed File System

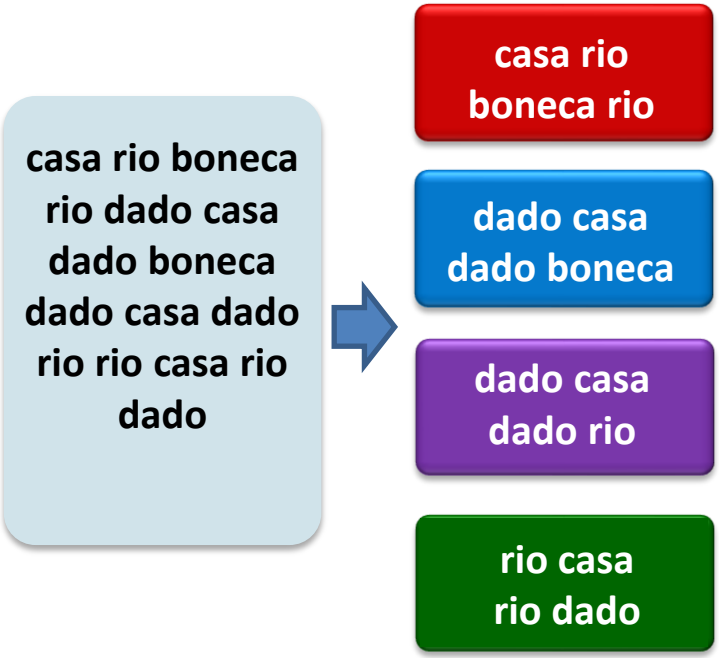


Cada bloco possui 3 réplicas distribuídas entre os DataNodes

MapReduce

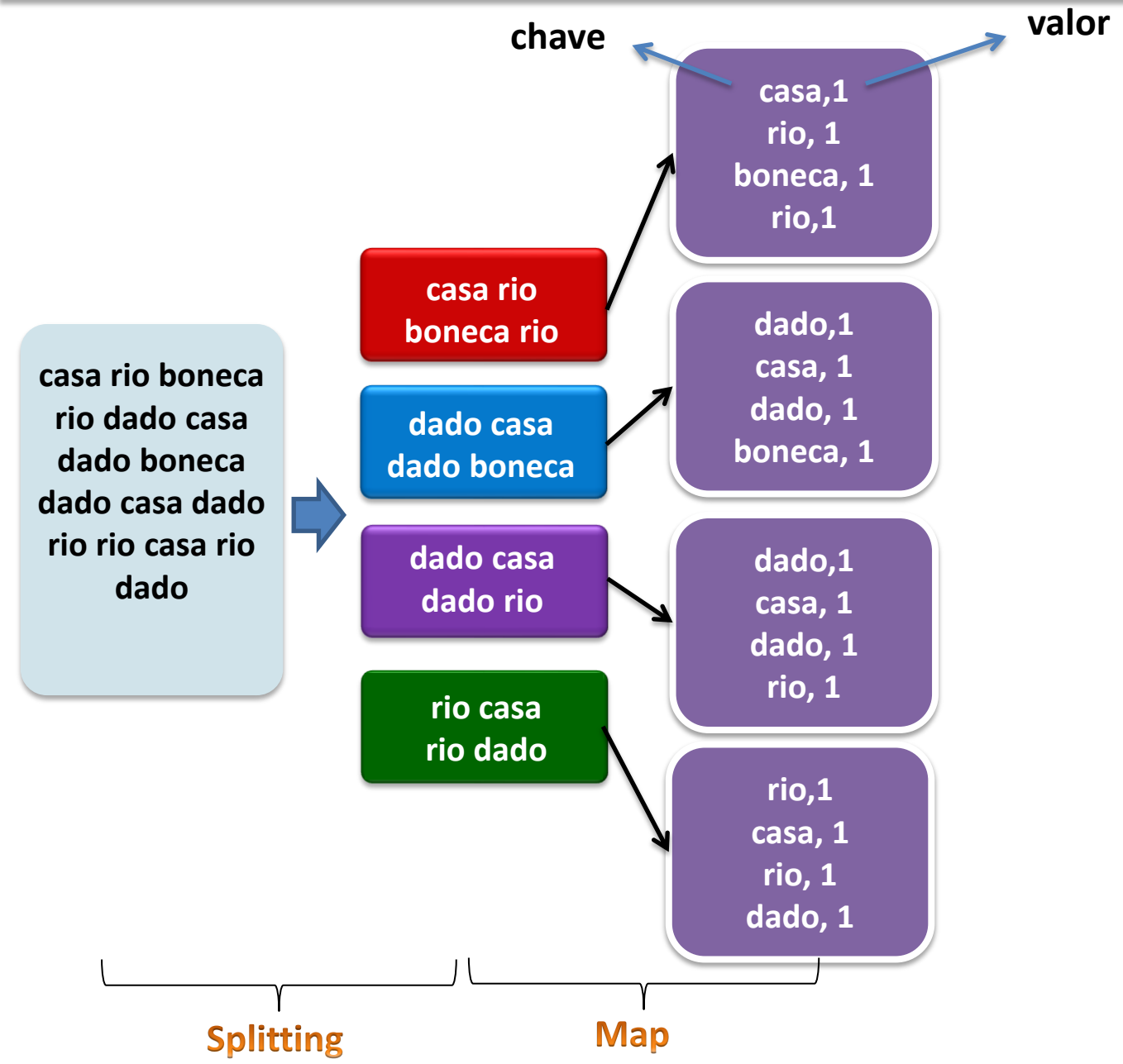
casa rio
boneca rio
dado casa
dado boneca
dado casa
boneca rio rio
casa boneca
dado

MapReduce

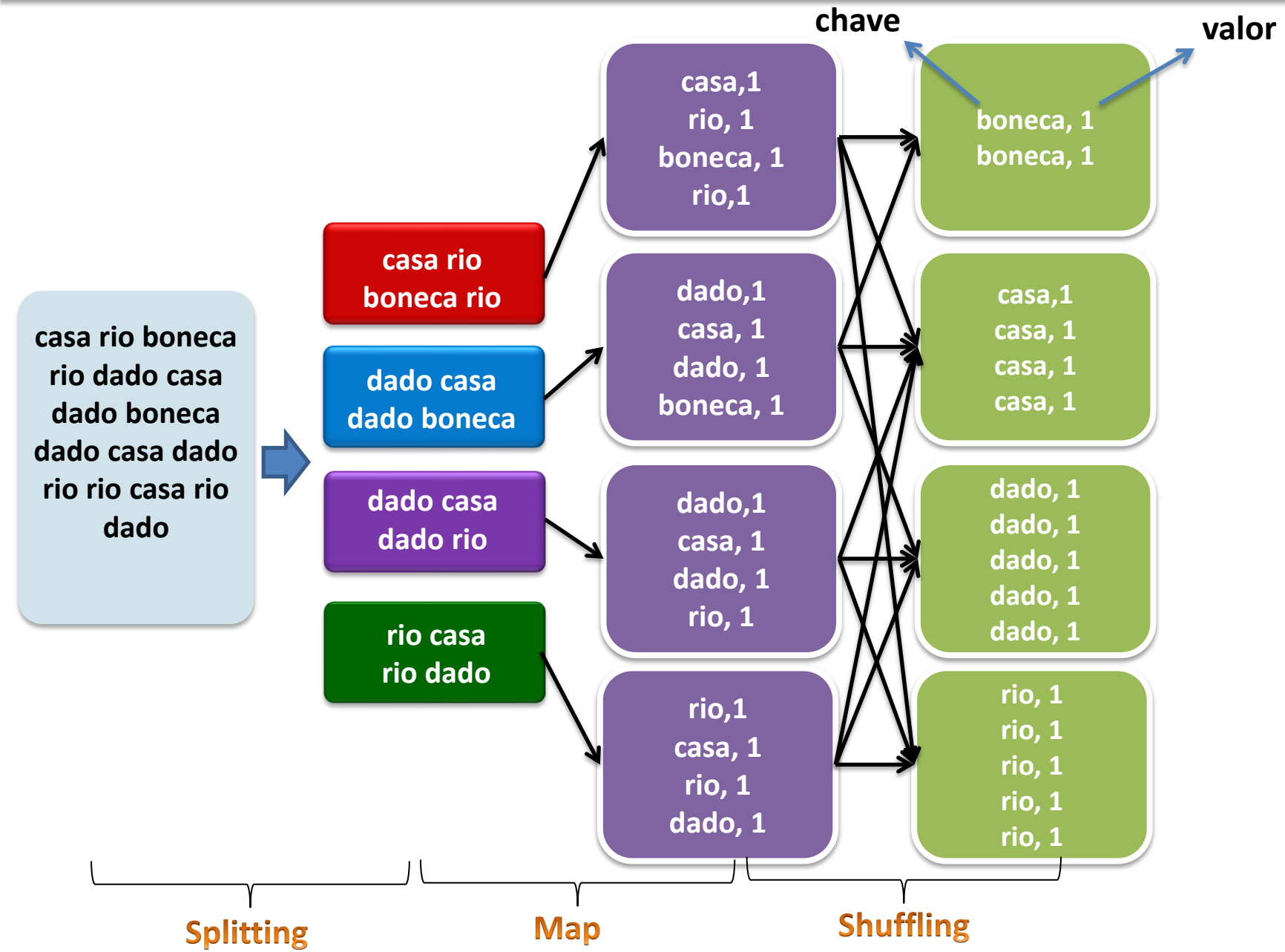


Splitting

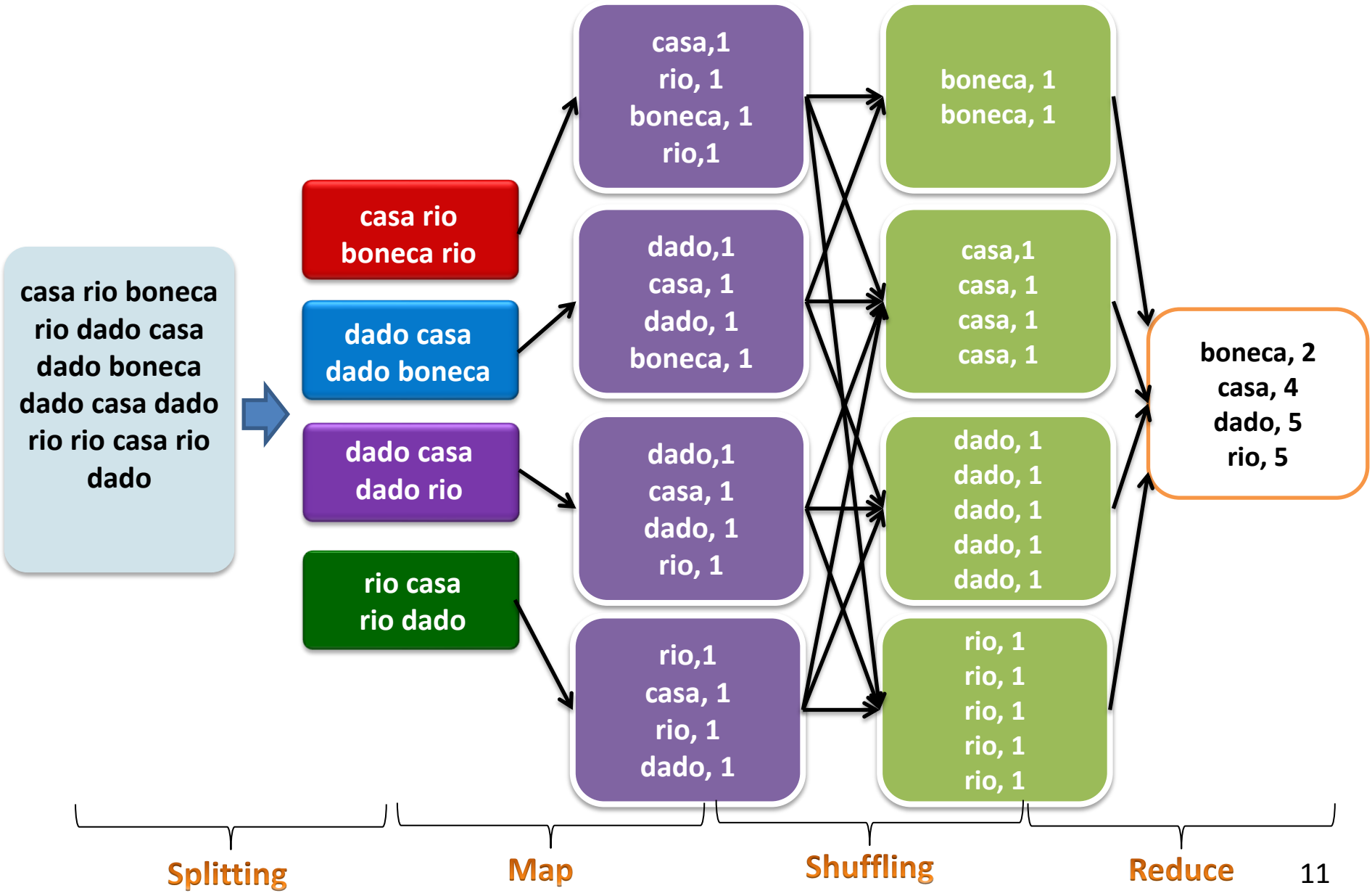
MapReduce


chave
valor
casa rio boneca rio dado casa dado boneca dado casa dado rio rio casa rio dado
casa rio boneca rio
dado casa dado boneca
dado casa dado rio
rio casa rio dado
casa,1 rio, 1 boneca, 1 rio,1
dado,1 casa, 1 dado, 1 boneca, 1
dado,1 casa, 1 dado, 1 rio, 1
rio,1 casa, 1 rio, 1 dado, 1
Splitting
Map

MapReduce

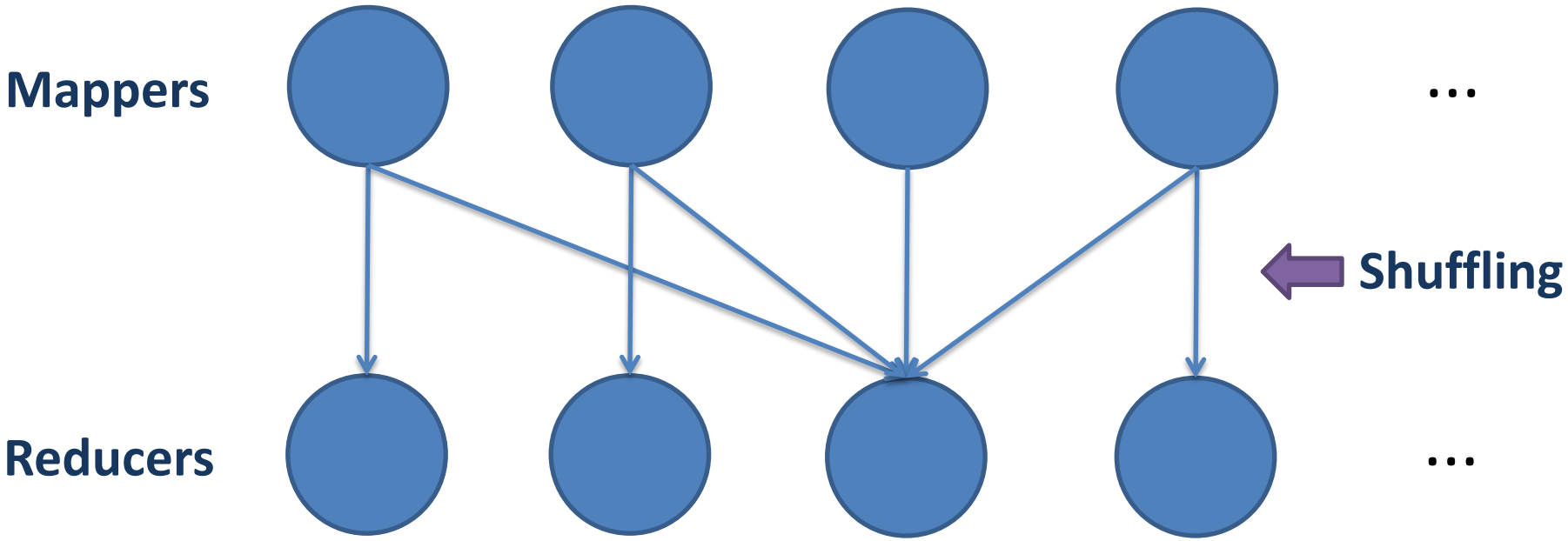

chave
valor
casa rio boneca rio dado casa dado boneca dado casa dado rio rio casa rio dado
casa rio boneca rio
dado casa dado boneca
dado casa dado rio
rio casa rio dado
casa,1 rio, 1 boneca, 1 rio,1
dado,1 casa, 1 dado, 1 boneca, 1
dado,1 casa, 1 dado, 1 rio, 1
rio,1 casa, 1 rio, 1 dado, 1
boneca, 1 boneca, 1
casa,1 casa, 1 casa, 1 casa, 1
dado, 1 dado, 1 dado, 1 dado, 1 dado, 1
rio, 1 rio, 1 rio, 1 rio, 1 rio, 1
Splitting
Map
Shuffling

MapReduce


casa rio boneca rio dado casa dado boneca dado casa dado rio rio casa rio dado
casa rio boneca rio
dado casa dado boneca
dado casa dado rio
rio casa rio dado
casa,1 rio, 1 boneca, 1 rio,1
dado,1 casa, 1 dado, 1 boneca, 1
dado,1 casa, 1 dado, 1 rio, 1
rio,1 casa, 1 rio, 1 dado, 1
boneca, 1 boneca, 1
casa,1 casa, 1 casa, 1 casa, 1
dado, 1 dado, 1 dado, 1 dado, 1 dado, 1
rio, 1 rio, 1 rio, 1 rio, 1 rio, 1
boneca, 2 casa, 4 dado, 5 rio, 5
Splitting
Map
Shuffling
Reduce

Modelo MapReduce

- Otimização
  - Redução da comunicação (fase de shuffling)
  - Minimização do número de jobs MapReduce



Junção em MapReduce

- Map-side Join
  - Junção na função Map
- Reduce-side Join
  - Junção na função Reduce

Map-side Join

- Junção é realizada na função Map

```
SELECT a, b
FROM S, T
WHERE  S.c = T.c
```

S

s	c	a
1	5	8
2	6	7
3	6	4
4	4	7

T

t	c	b
5	6	4
7	4	2
8	4	1
10	5	4

Map-side Join

- Junção é realizada na função Map

```
SELECT a, b
FROM S, T
WHERE S.c = T.c
```

S

s	c	a
1	5	8
2	6	7
3	6	4
4	4	7

S_1

s	c	a
4	4	7
1	5	8

S_2

s	c	a
2	6	7
3	6	4

T

t	c	b
5	6	4
7	4	2
8	4	1
10	5	4

T_1

t	c	b
7	4	2
8	4	1
10	5	4

T_2

t	c	b
5	6	4

Map-side Join

Mapper 1

S_1

s	c	a
4	4	7
1	5	8

T_1

t	c	b
7	4	2
8	4	1
10	5	4

Mapper 2

S_2

s	c	a
2	6	7
3	6	4

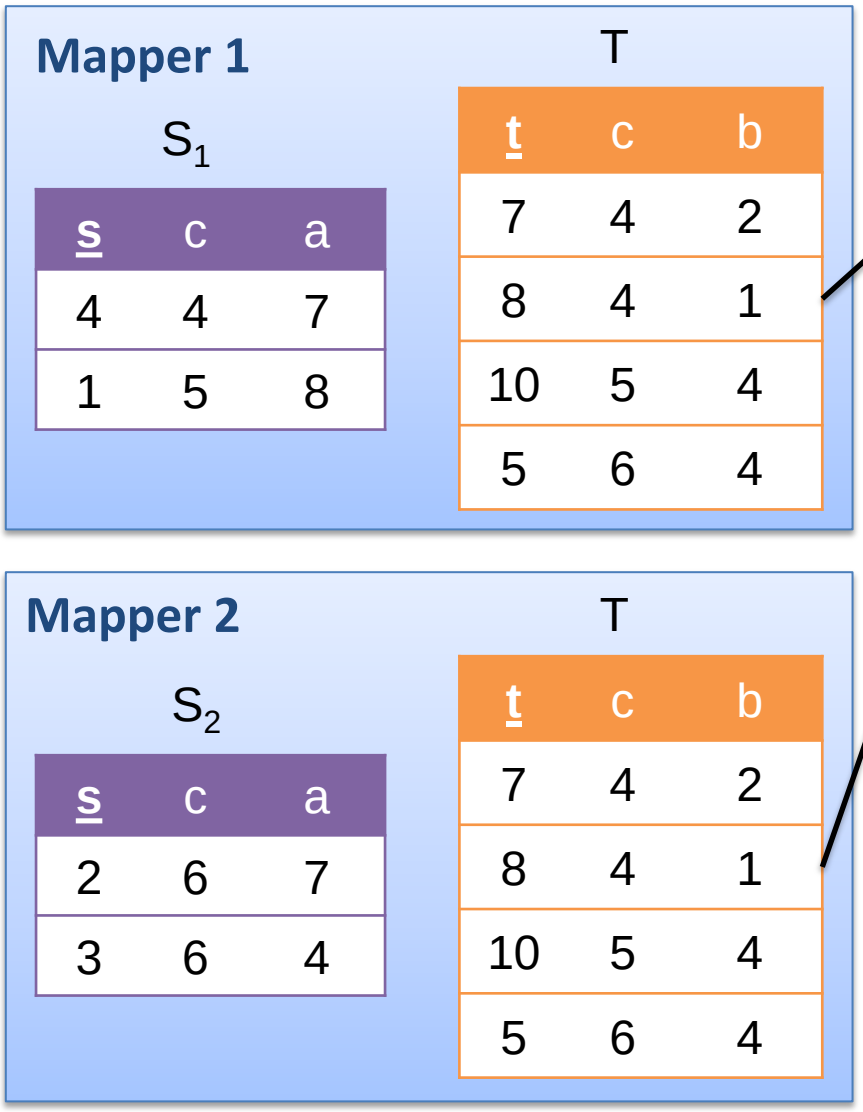
T_2

t	c	b
5	6	4

- Dados são particionados e ordenados pela chave de junção (atributo c)
- Blocos correspondentes de cada arquivo são processados por uma única tarefa Mapper
- A função Map é aplicada sobre um dos blocos (ex: S_1), enquanto o outro bloco correspondente (ex: T_1) é lido dentro da tarefa Mapper
- Cada Mapper possui os dados necessários para realizar a junção de seus blocos

Map-side Join

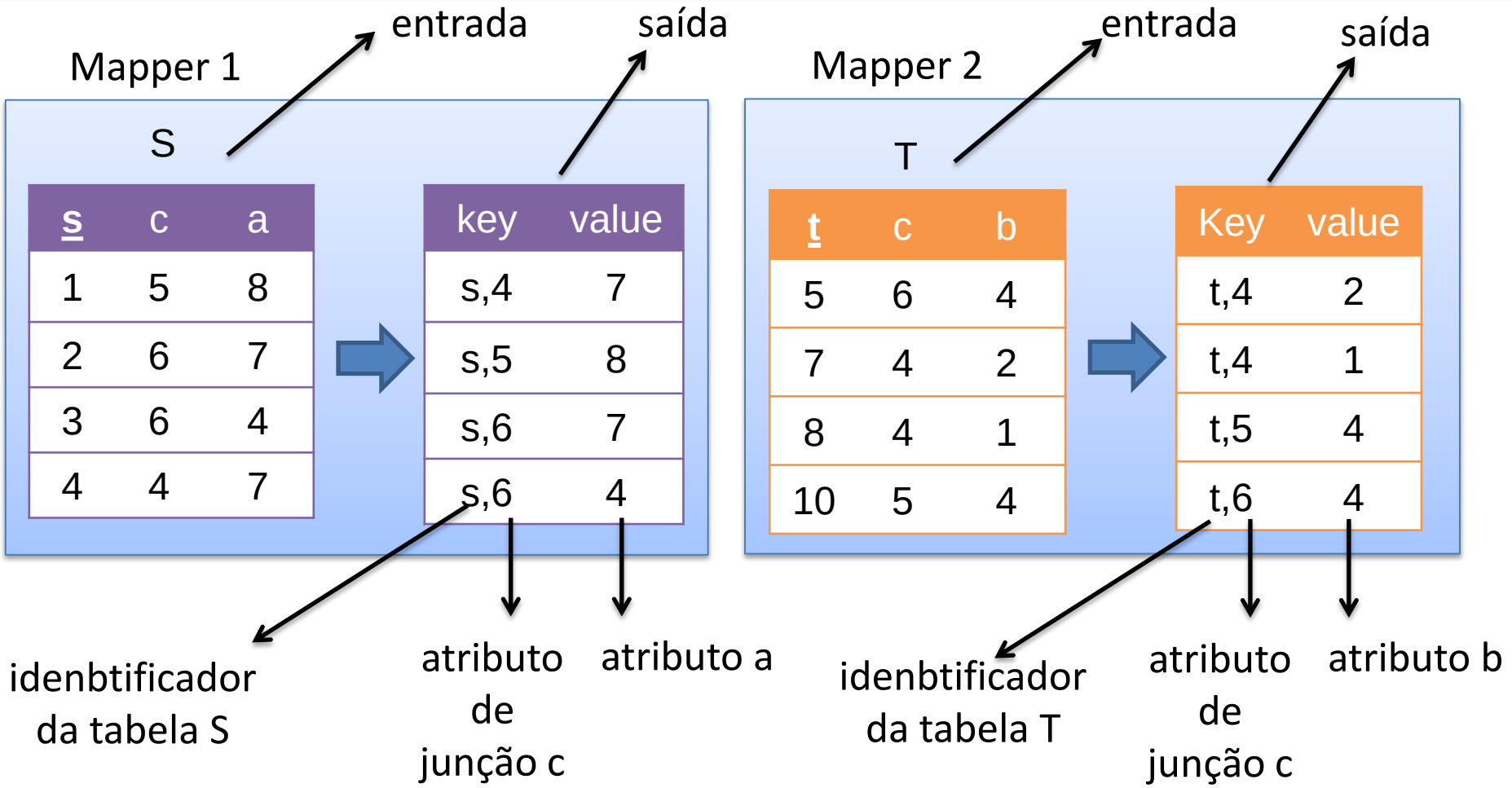
Memory-backed Join



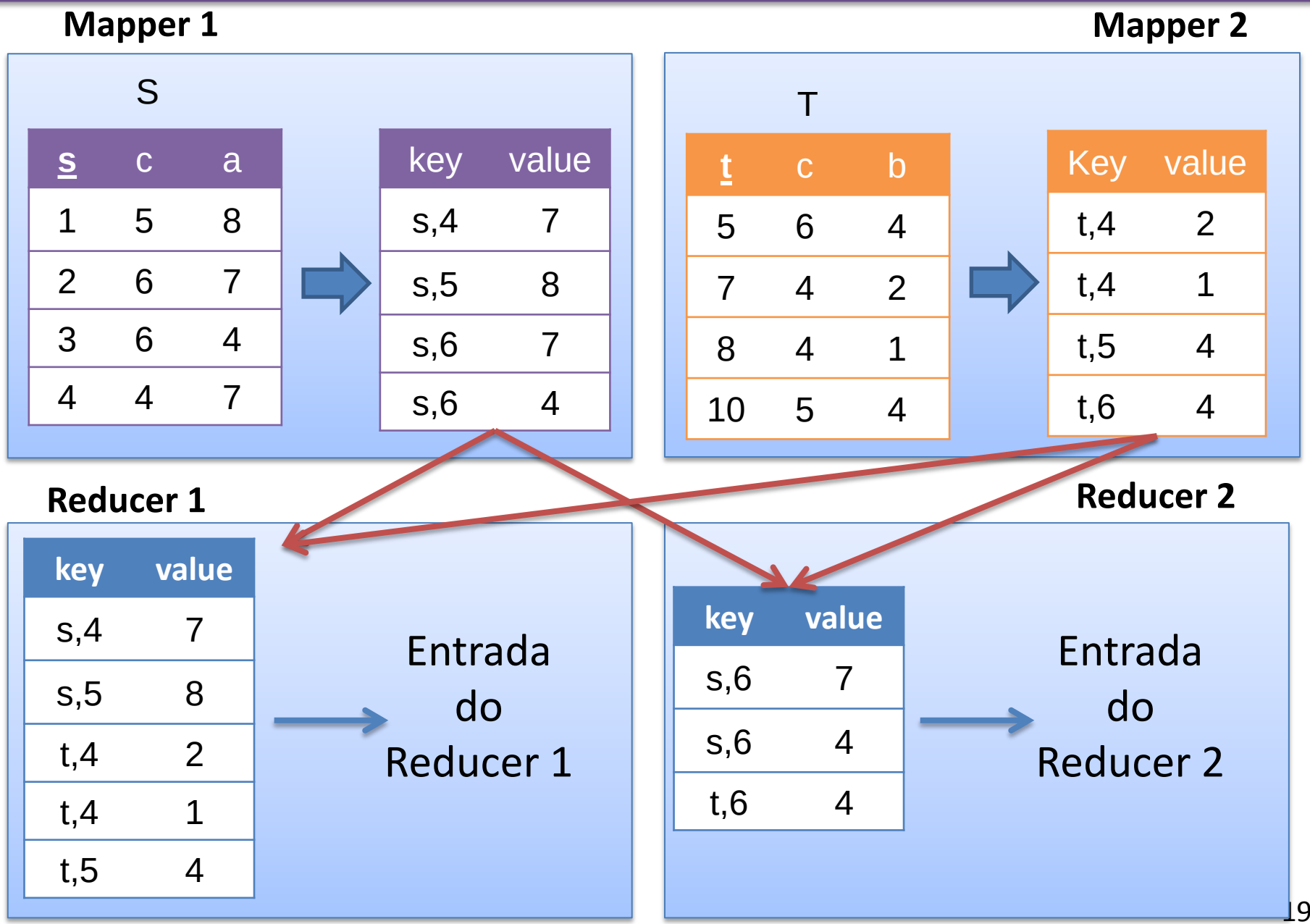
Dados em memória primária local

- Tabela menor (ex: tabela T) é armazenada na memória primária local de cada nó
- Blocos da tabela maior (ex: S) são processados nos diferentes mappers
- Cada mappers tem acesso a todos os dados da tabela menor (ex: tabela T)

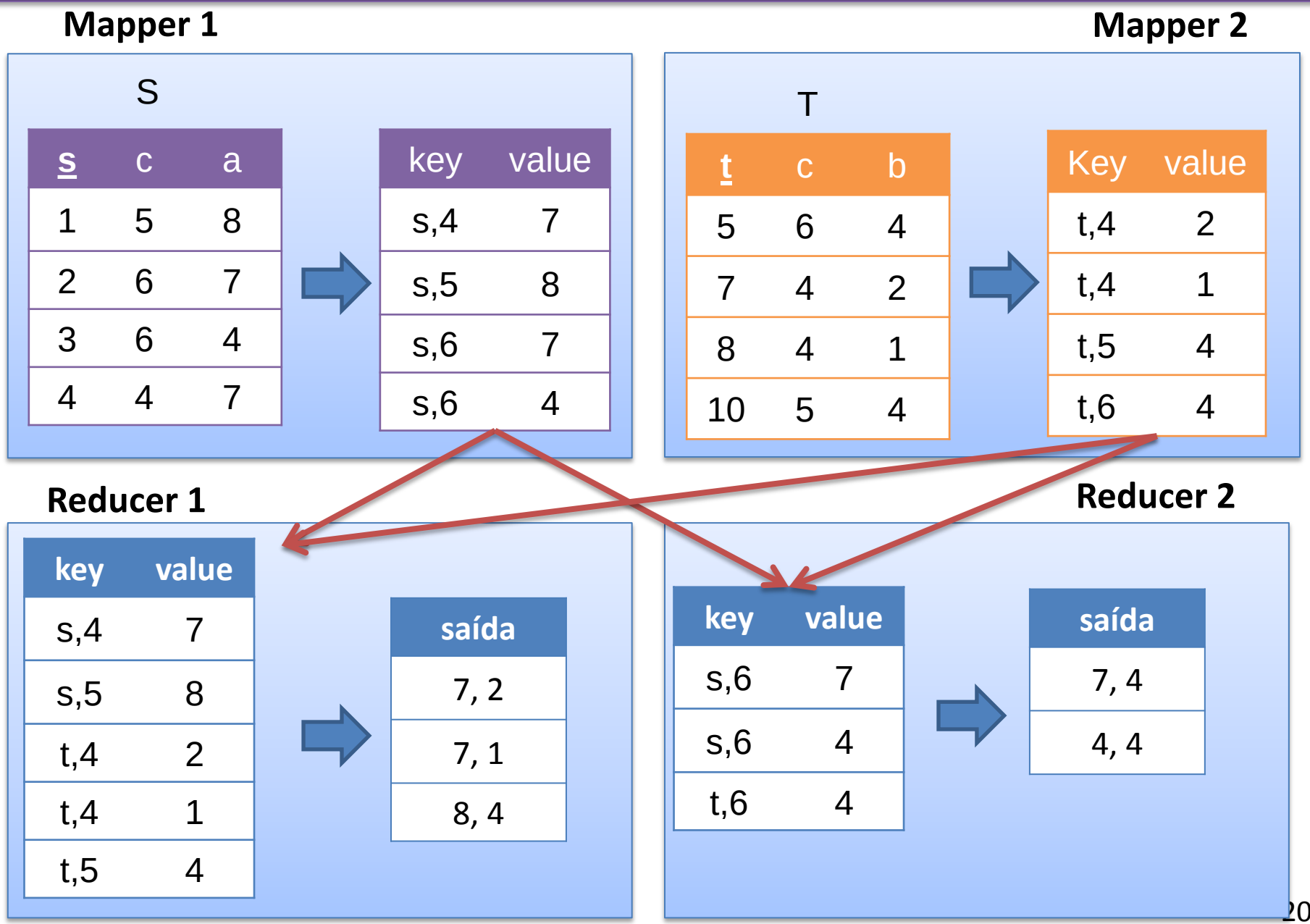
Reduce-side Join



Reduce-side Join



Reduce-side Join



Junção em MapReduce

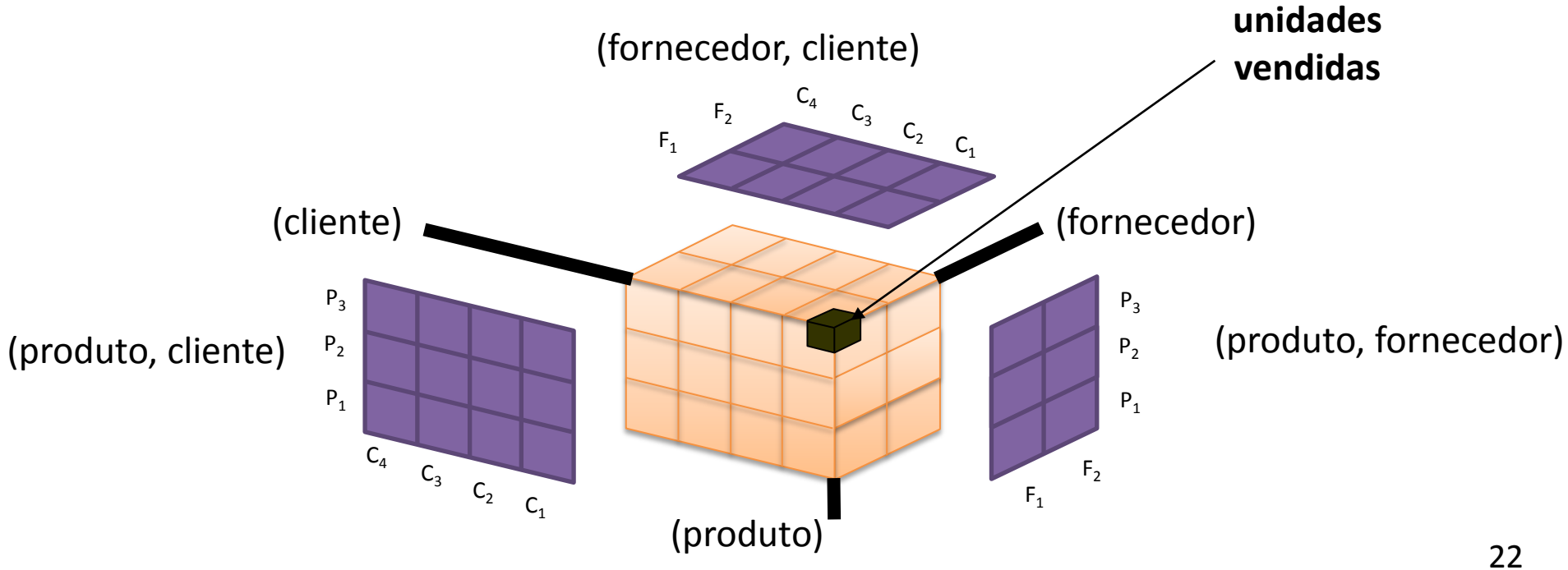
- **Map-side Join**
  - **Desvantagem:** aplicável quando o conjunto pode ser ordenado e particionado pelo atributo de junção
  - Memory-backed Join
    - **Desvantagem:** aplicável quando uma das tabelas é pequena e cabe na memória primária de cada nó
  - **Vantagem:** processamento local, dispensando a fase de shuffling

- **Reduce-side Join**
  - **Vantagem:** aplicável em qualquer conjunto de tabelas
  - **Desvantagem:** necessidade da fase de shuffling

Data Warehouse

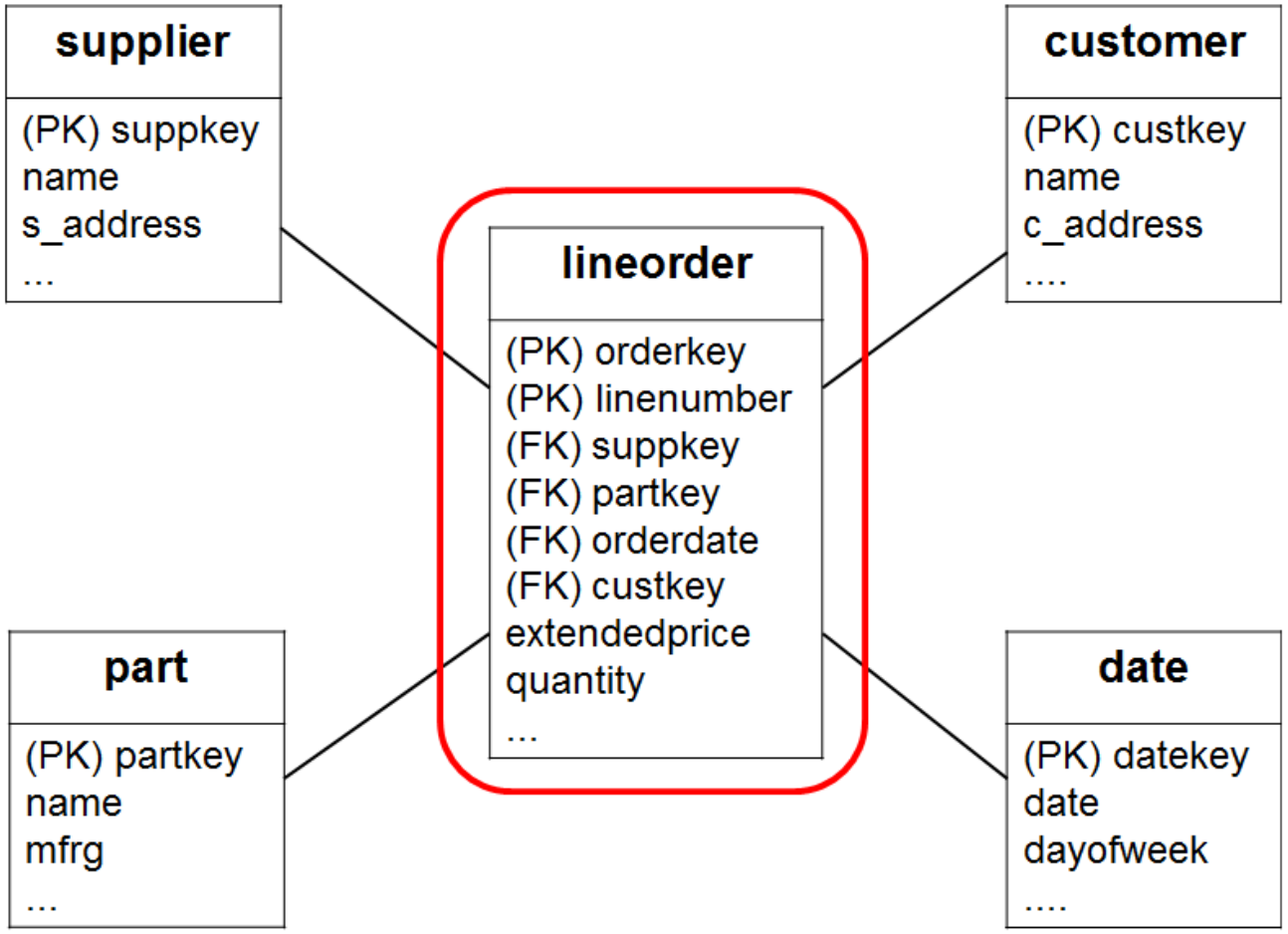
- Banco de dados voltado ao processamento analítico para a tomada de decisão

- Modelagem multidimensional
  - Medidas numéricas: objetos de análise
  - Dimensões: perspectiva/contexto para as análises



Esquema Estrela

- *Star Schema Benchmark* (SSB)



Consulta de Junção Estrela

- Consulta Q3.2 do SSB

```
SELECT  c_city, s_city, d_year, SUM(lo_revenue) as revenue
FROM    Lineorder, Supplier,  Customer, Date
WHERE   lo_custkey = c_custkey
    AND   lo_suppkey = s_suppkey
    AND   lo_orderdate = d_datekey
    AND   c_nation = 'UNITED STATES'
    AND   s_nation = 'UNITED STATES'
    AND   d_year >= 1992 AND d_year <= 1997
GROUP BY c_city, s_city, d_year
ORDER BY c_city, s_city, d_year
```

Consulta de Junção Estrela

- Consulta Q3.2 do SSB

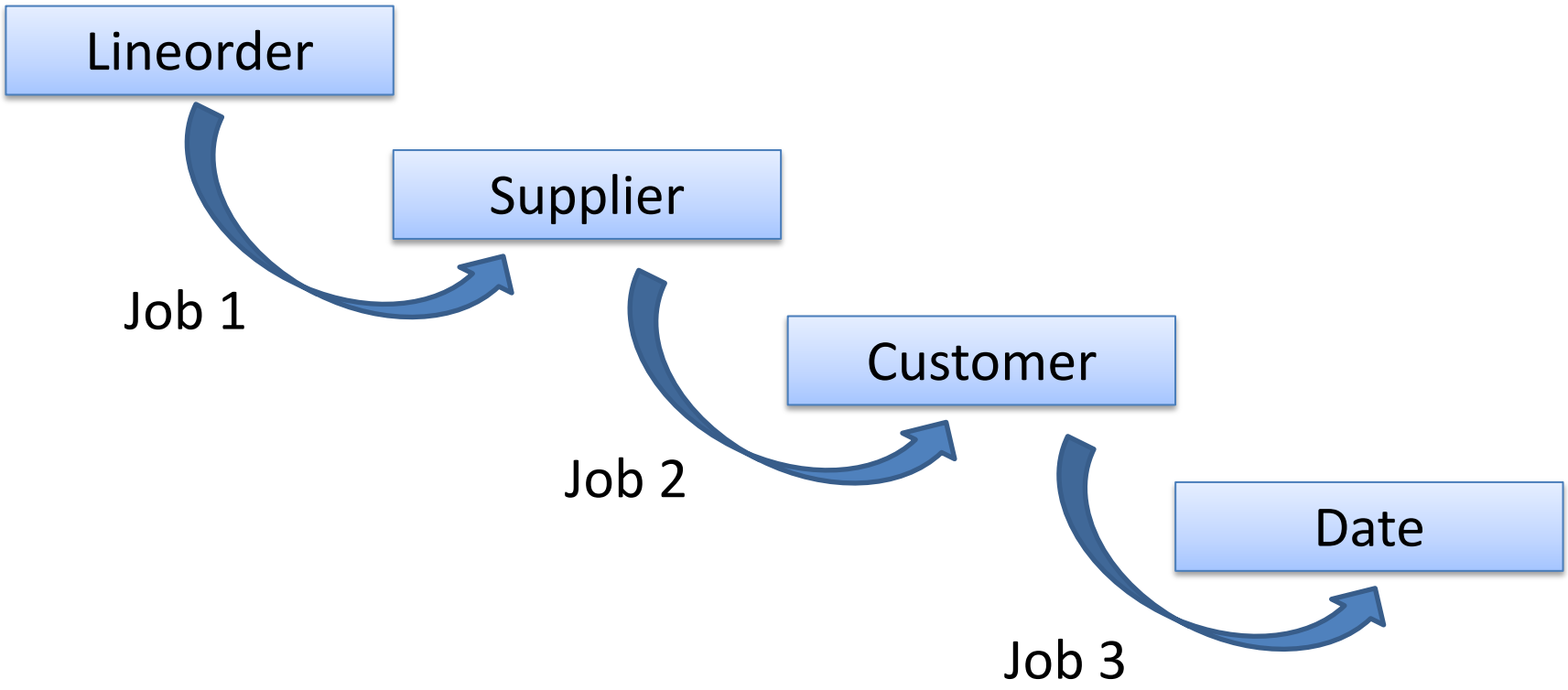
```
SELECT  c_city, s_city, d_year, SUM(lo_revenue) as revenue
FROM    Lineorder, Supplier,  Customer, Date
WHERE   lo_custkey = c_custkey
    AND lo_suppkey = s_suppkey
    AND lo_orderdate = d_datekey
    AND c_nation = 'UNITED STATES'
    AND s_nation = 'UNITED STATES'
    AND d_year >= 1992 AND d_year <= 1997
GROUP BY c_city, s_city, d_year
ORDER BY c_city, s_city, d_year
```

cláusulas de junção

cláusulas de filtragem

Sequência de Junções Binárias em MapReduce

- Um job MapReduce para cada junção

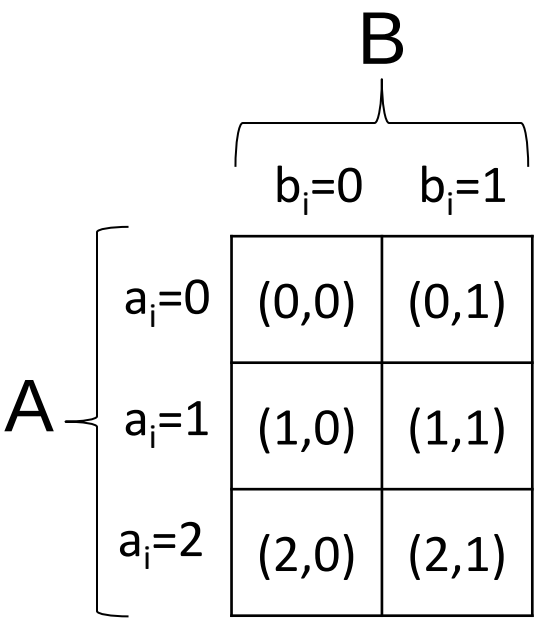


Algoritmo de Afrati e Ullman (2010)

- Proposta: realizar todas as junções em apenas um job

$$S(s) \bowtie U(s, t) \bowtie T(t)$$

- O domínio do atributo s é dividido em A blocos, enquanto que o domínio de t é dividido em B blocos
- O número de processos reducers é dado por AB



Supondo que A=3 e B=2, temos um total de 6 reducers

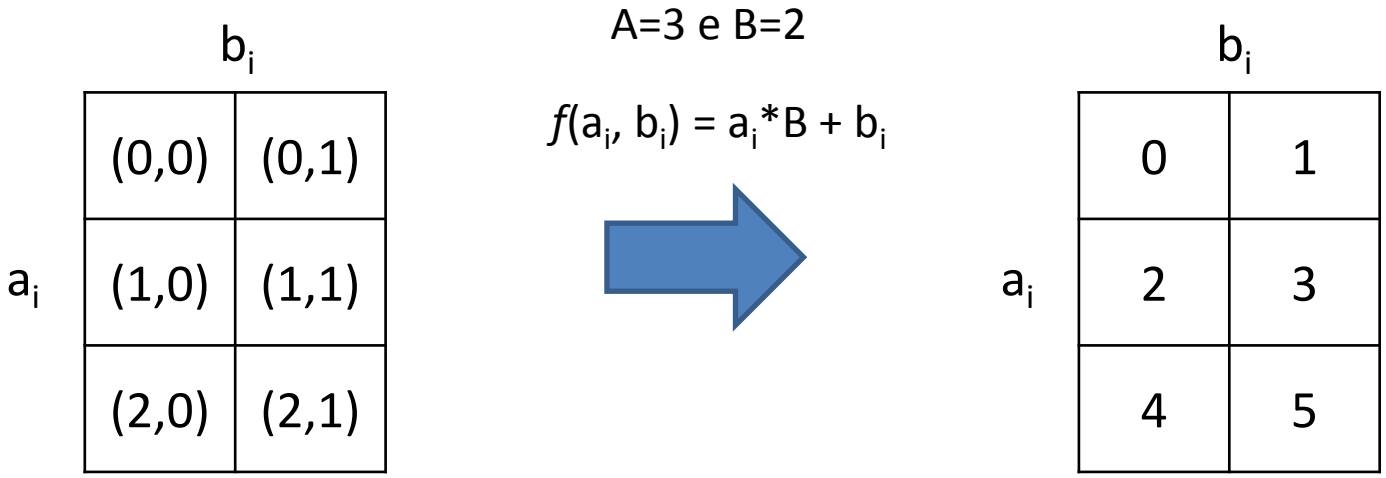
Cada processo reduce é identificado por um par (a_i, b_i)

Algoritmo de Afrati e Ullman (2010)

- Proposta: realizar todas as junções em apenas um job

$$S(s) \bowtie U(s, t) \bowtie T(t)$$

- O processo reduce para o qual uma tupla deve ser enviada é identificado por dois valores, a e b, determinados a partir dos atributos s e t (atributos de junção)



Algoritmo de Afrati e Ullman (2010)

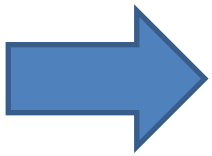
- Proposta: realizar todas as junções em apenas um job

$$S(s) \bowtie U(s, t) \bowtie T(t)$$

- Para cada valor s_i do atributo s, $a_i = mod(s_i, A)$
- Para cada valor t_i do atributo t, $b_i = mod(t_i, B)$
- Reducer identificado por uma função $f(a_i, b_i) = a_i * B + b_i$

a_i \ b_i		
	(0,0)	(0,1)
	(1,0)	(1,1)
	(2,0)	(2,1)

A=3 e B=2



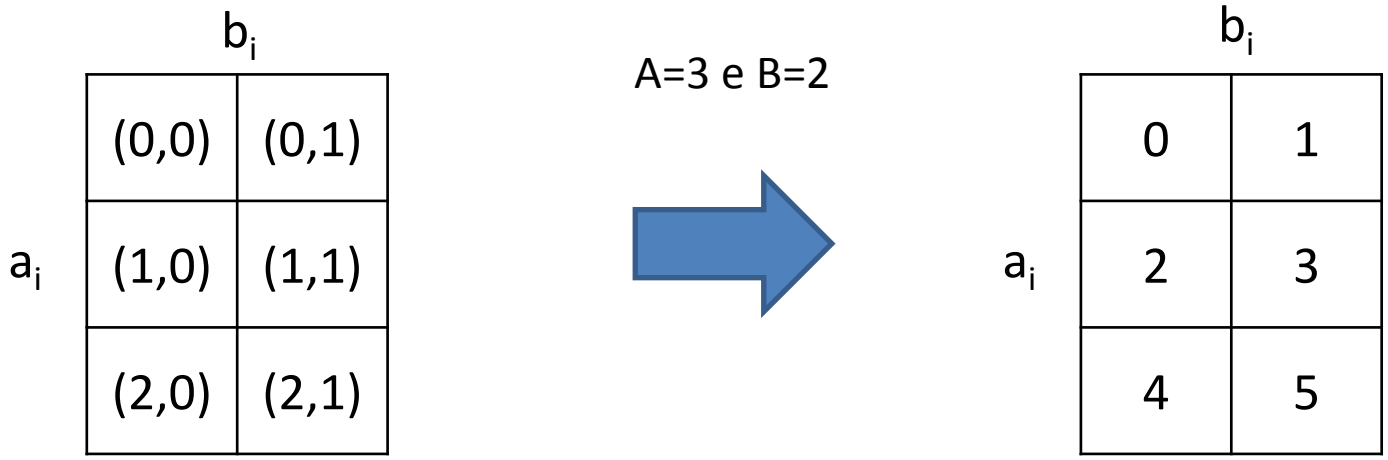
a_i \ b_i		
	0	1
	2	3
	4	5

Algoritmo de Afrati e Ullman (2010)

- Proposta: realizar todas as junções em apenas um job

$$S(s) \bowtie U(s, t) \bowtie T(t)$$

- Cada tupla de S precisa ser enviada para todos os reducers identificados por um determinado valor a_i
- Cada tupla de T precisa ser enviada para todos os reducers identificados por um um determinado valor b_i



Algoritmo de Afrati e Ullman (2010)

➢ Exemplo: $S(s) \bowtie U(s, t) \bowtie T(t)$

Id do processo reduce

$f(a_i, b_i) = a_i * B + b_i$

A=3 e B=2

S

s
1
2
3
4

U

s	t
2	5
3	7
1	8
4	10

T

t
5
7
8
10

Tuplas da tabela U são enviadas para um único reduce

S

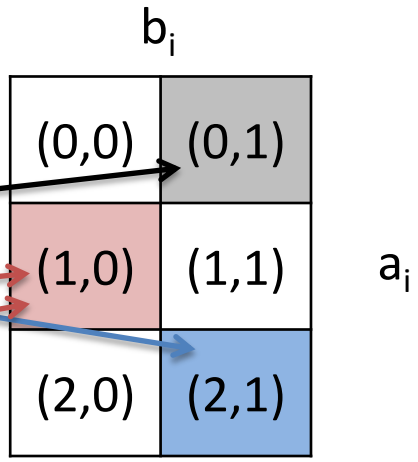
s
$a_i = 1$
$a_i = 2$
$a_i = 0$
$a_i = 1$

U

s	t
$a_i = 2$	$b_i = 1$
$a_i = 0$	$b_i = 1$
$a_i = 1$	$b_i = 0$
$a_i = 1$	$b_i = 0$

T

t
$b_i = 1$
$b_i = 1$
$b_i = 0$
$b_i = 0$



Algoritmo de Afrati e Ullman (2010)

➢ Exemplo: $S(s) \bowtie U(s, t) \bowtie T(t)$

Id do processo reduce

$f(a_i, b_i) = a_i * B + b_i$

A=3 e B=2

S

s
1
2
3
4

U

s	t
2	5
3	7
1	8
4	10

T

t
5
7
8
10

Cada tupla da tabela S é enviada para B reducers (todos reducers de uma mesma linha)

S

s
$a_i = 1$
$a_i = 2$
$a_i = 0$
$a_i = 1$

U

s	t
$a_i = 2$	$b_i = 1$
$a_i = 0$	$b_i = 1$
$a_i = 1$	$b_i = 0$
$a_i = 1$	$b_i = 0$

T

t
$b_i = 1$
$b_i = 1$
$b_i = 0$
$b_i = 0$

(0,0)	(0,1)
(1,0)	(1,1)
(2,0)	(2,1)

a_i

b_i

Algoritmo de Afrati e Ullman (2010)

➢ Exemplo: $S(s) \bowtie U(s, t) \bowtie T(t)$

Id do processo reduce

$f(a_i, b_i) = a_i * B + b_i$

A=3 e B=2

S

s
1
2
3
4

U

s	t
2	5
3	7
1	8
4	10

T

t
5
7
8
10

Cada tupla da tabela T é enviada para A reducers (todos reducers de uma mesma coluna)

S

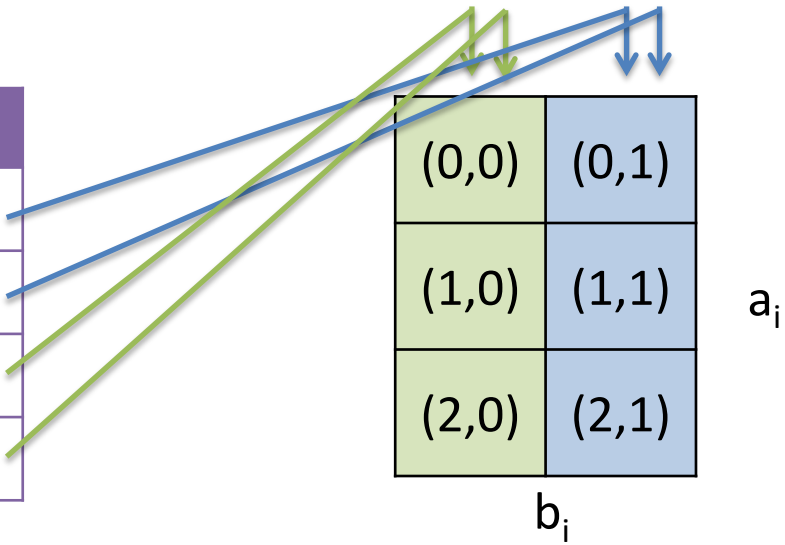
s
$a_i = 1$
$a_i = 2$
$a_i = 0$
$a_i = 1$

U

s	t
$a_i = 2$	$b_i = 1$
$a_i = 0$	$b_i = 1$
$a_i = 1$	$b_i = 0$
$a_i = 1$	$b_i = 0$

T

t
$b_i = 1$
$b_i = 1$
$b_i = 0$
$b_i = 0$



Algoritmo de Afrati e Ullman (2010)

Reducer 0

Chave	Valor
S|3	null
T|8	null
T|10	null

Reducer 1

Chave	Valor
S|3	null
T|5	null
T|7	null
U|3,7	null

Reducer 2

Chave	Valor
S|1	null
S|4	null
T|8	null
T|10	null
U|1,8	null
U|4,10	null

Reducer 3

Chave	Valor
S|1	null
S|4	null
T|5	null
T|7	null

Reducer 4

Chave	Valor
S|2	null
T|8	null
T|10	null

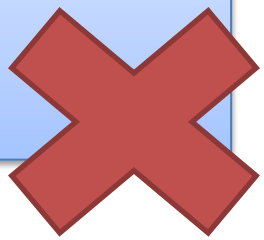
Reducer 5

Chave	Valor
S|2	null
T|5	null
T|7	null
U|2,5	null

Algoritmo de Afrati e Ullman (2010)

Reducer 0

Chave	Valor
S|3	null
T|8	null
T|10	null



Reducer 1

Chave	Valor
S|3	null
T|5	null
T|7	null
U|3,7	null

Reducer 2

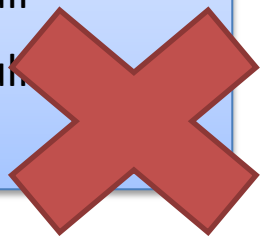
Chave	Valor
S|1	null
S|4	null
T|8	null
T|10	null
U|1,8	null
U|4,10	null

Reducer 3

Chave	Valor
S|1	null
S|4	null
T|5	null
T|7	null

Reducer 4

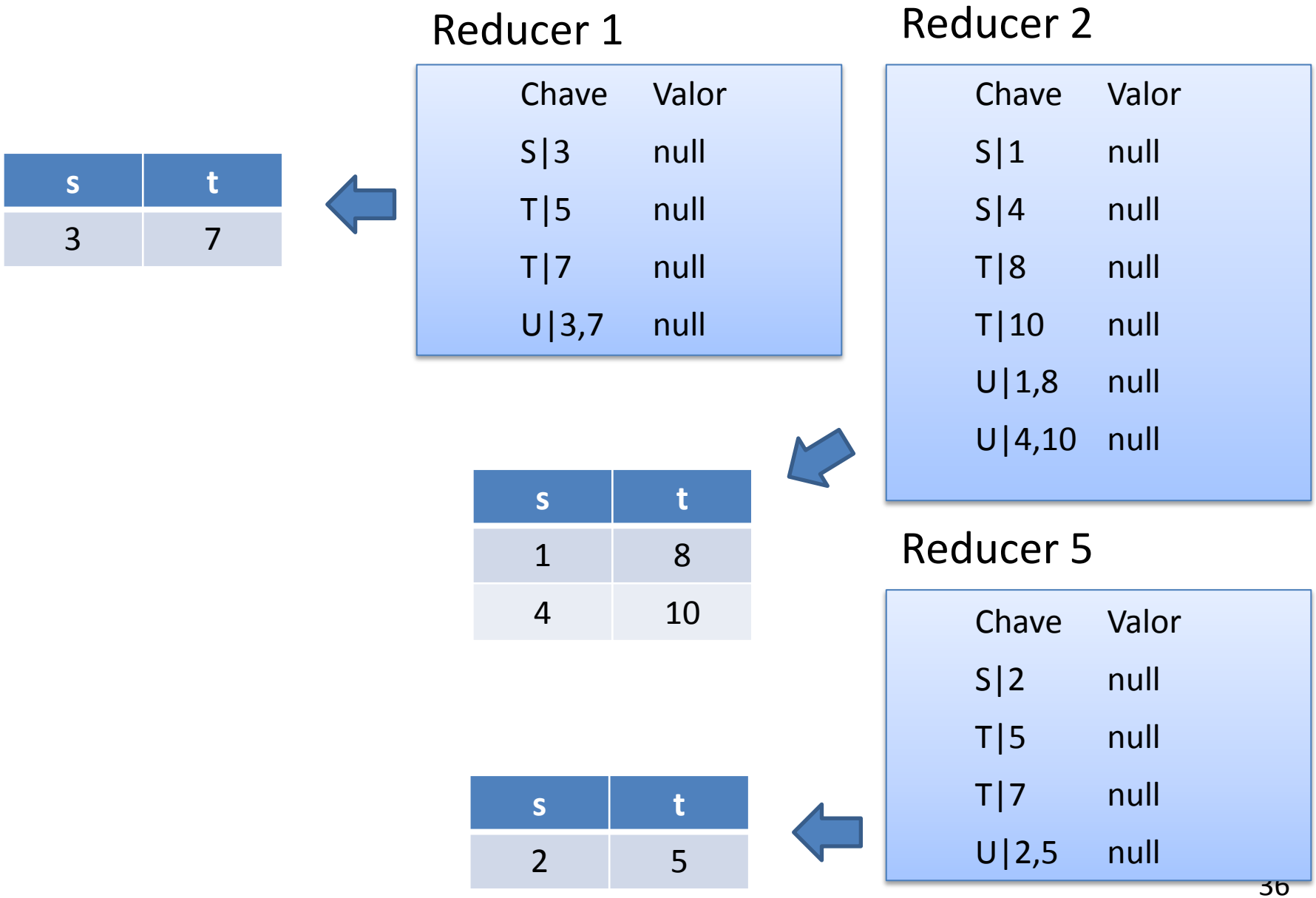
Chave	Valor
S|2	null
T|8	null
T|10	null



Reducer 5

Chave	Valor
S|2	null
T|5	null
T|7	null
U|2,5	null

Algoritmo de Afrati e Ullman (2010)



Algoritmo de Afrati e Ullman (2010)

Vantagem

- realiza todas as junções em apenas um job MapReduce

Desvantagem

- Replicação de dados das tabelas de dimensão (S e T no exemplo)
- Caso existam filtros nas tabelas de dimensão, tuplas da tabela de fatos (U no exemplo) são enviadas para os reducers desnecessariamente

Referências

- Han, H.; Jung, H.; Eom, H.; Yeom, H. Y. Scatter-gather-merge: An efficient star-join query processing algorithm for data-parallel frameworks. Cluster Computing, v. 14, n. 2, p. 183–197, 2011.
- Afrati, F. N.; Ullman, J. D. Optimizing joins in a map-reduce environment. In: Proceedings of the 13th International Conference on Extending Database Technology (EDBT 2010), 2010. p. 99–110.
- Tao, Y., Zhou, M., Shi, L., Wei, L., Cao, Y.: Optimizing multi-join in cloud environment. In: Proceedings of the IEEE International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing. pp. 956–963 (2013).
- Thusoo, A., Sarma, J.S., Jain, N., Shao, Z., Chakka, P., Zhang, N., Anthony, S., Liu, H., Murthy, R.: Hive - a petabyte scale data warehouse using hadoop. In: ICDE. pp. 996–1005 (2010)5.
- Zhang, C., Wu, L., Li, J.: Efficient processing distributed joins with bloomfilter using mapreduce. Int J Grid Ditrib Comput 6(3), 45-58 (2013).

Obrigada