creativecommons.org/licenses/by-nc-nd/4.0/legalcode

Jens Dittrich

Duplicates in both Columns!

Customers		
name	**street**	**cityID**
frank	minstreet	0
peter	minstreet	0
stefan	unistreet	0
jens	shortstreet	1
steve	macstreet	1
felix	macstreet	5
hans	msstreet	5
alekh	unistreet	7
jorge	minstreet	9
mike	longstreet	9
olaf	macstreet	9
tim	unistreet	9

Cities_Dictionary	
cityNo	**city**
1	new york
1	cuppertino
1	paris
5	berlin
5	london
9	saarbruecken

Joins? CoGroups!

Customers		
name	**street**	**cityID**
frank	minstreet	0
peter	minstreet	
stefan	unistreet	
jens	shortstreet	1
steve	macstreet	
felix	macstreet	5
hans	msstreet	
alekh	unistreet	7
jorge	minstreet	9
mike	longstreet	
olaf	macstreet	
tim	unistreet	

Cities_Dictionary	
cityNo	**city**
1	new york
	cuppertino
	paris
5	berlin
	london
9	saarbruecken

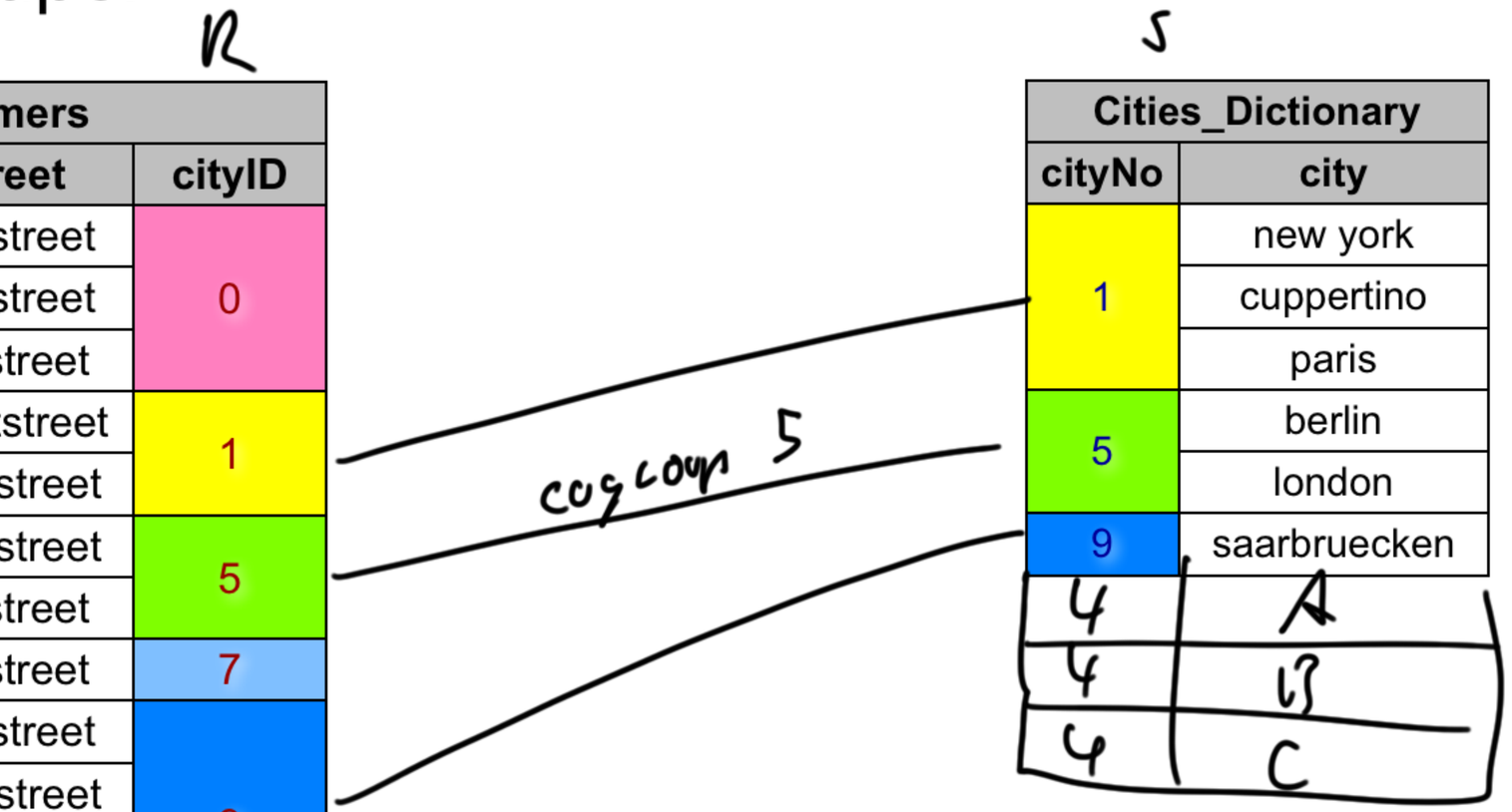
Joins? CoGroups!

R

Customers		
name	**street**	**cityID**
frank	minstreet	0
peter	minstreet	
stefan	unistreet	
jens	shortstreet	1
steve	macstreet	
felix	macstreet	5
hans	msstreet	
alekh	unistreet	7
jorge	minstreet	9
mike	longstreet	
olaf	macstreet	
tim	unistreet	

S

Cities_Dictionary	
cityNo	**city**
1	new york
	cuppertino
	paris
5	berlin
	london
9	saarbruecken



Co-Grouping without Sorting

Customers		
name	**street**	**cityID**
peter	minstreet	0
steve	macstreet	1
mike	longstreet	9
tim	unistreet	9
hans	msstreet	5
jens	shortstreet	1
frank	minstreet	0
olaf	macstreet	9
stefan	unistreet	0
alekh	unistreet	7
felix	macstreet	5
jorge	minstreet	9

Cities_Dictionary	
cityNo	**city**
5	berlin
1	cuppertino
9	saarbruecken
1	paris
1	new york
5	london

Co-Grouping without Sorting

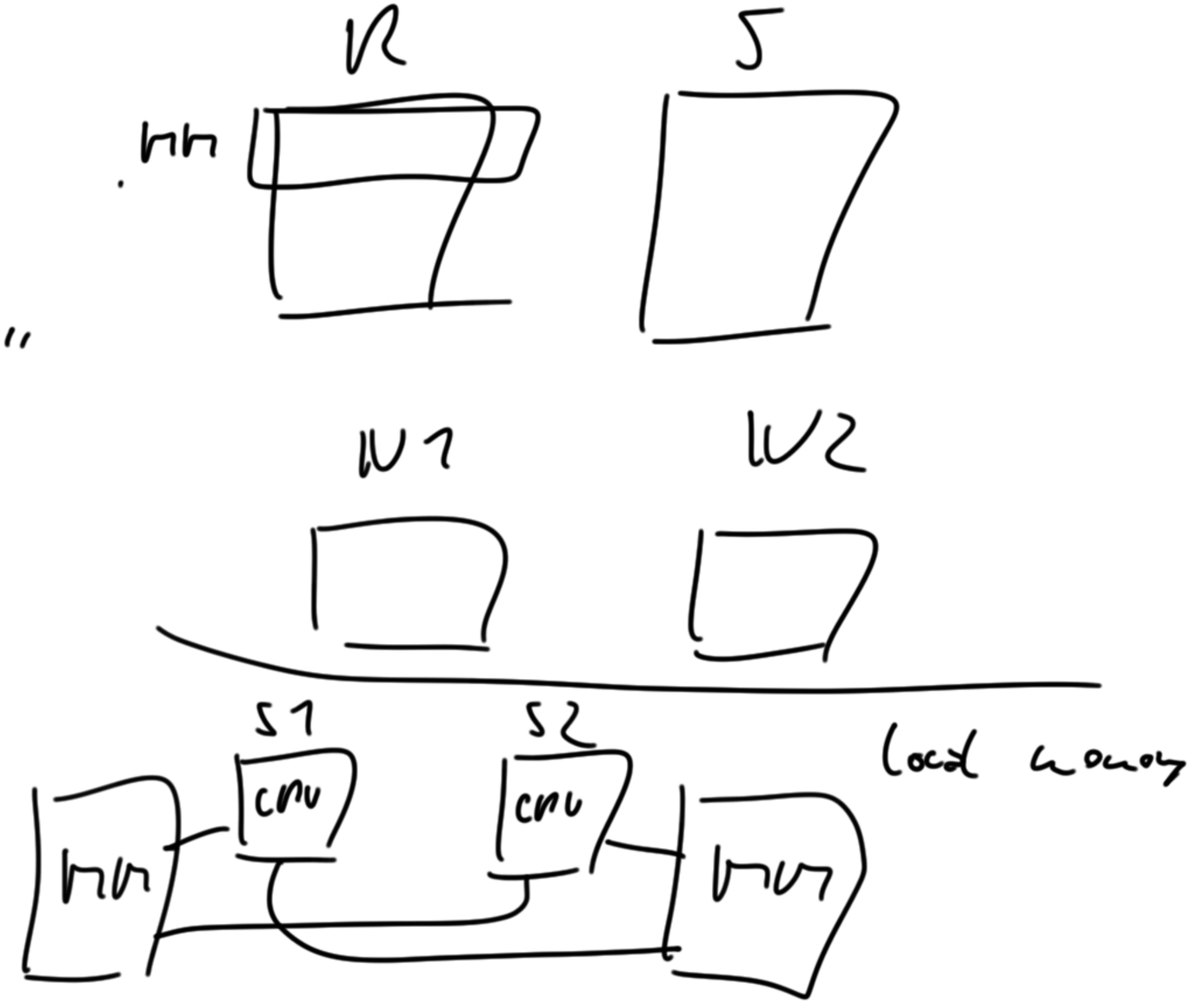
Customers		
name	**street**	**cityID**
peter	minstreet	0
steve	macstreet	1
mike	longstreet	9
tim	unistreet	9
hans	msstreet	5
jens	shortstreet	1
frank	minstreet	0
olaf	macstreet	9
stefan	unistreet	0
alekh	unistreet	7
felix	macstreet	5
jorge	minstreet	9

Cities_Dictionary	
cityNo	**city**
5	berlin
1	cuppertino
9	saarbruecken
1	paris
1	new york
5	london

Generalized Co-Grouped Join

special cases:

1. Grace Hash Join
2. Partitioned Joins
   "Repartitioning Joins"
3. NUMA Join



Generalized Co-Grouped Join

JP(r,s) := r.x == s.x //definition of the join predicate

group(Tuple **):** Tuple ↦ [0, ..., k-1] //generalized grouping function

Labels tuple

$r_{42} \rightarrow 5$

$r_{13} \rightarrow 7$

$s_3 \rightarrow 2$

$s_4 \rightarrow 7$

Generalized Co-Grouped Join

JP(r,s) := r.x == s.x //definition of the join predicate

group(Tuple **):** Tuple ↦ [0, ..., k-1] //generalized grouping function

partition(Set, group() **):** (Set, group()) ↦ Set of Pair<Set, groupID> //generalized partitioning function

Generalized Co-Grouped Join

JP(r,s) := r.x == s.x //definition of the join predicate

group(Tuple **):** Tuple ↦ [0, ..., k-1] //generalized grouping function

partition(Set, group() **):** (Set, group()) ↦ Set of Pair<Set, groupID> //generalized partitioning function

CoGroupedJoin(R, S, JP(r,s), group(), partition()):

Set of Pair<Set, groupID> build := partition(R, group()): //partition R into subsets (aka groups)

Generalized Co-Grouped Join

JP(r,s) := r.x == s.x //definition of the join predicate

group(Tuple **):** Tuple ↦ [0, ..., k-1] //generalized grouping function

partition(Set, group() **):** (Set, group()) ↦ Set of Pair<Set, groupID> //generalized partitioning function

CoGroupedJoin(R, S, JP(r,s), group(), partition()):

Set of Pair<Set, groupID> build := partition(R, group()): //partition R into subsets (aka groups)

Set of Pair<Set, groupID> probe := partition(S, group()); //partition S into subsets (aka groups)

co-grouping: use the same group()-function

Generalized Co-Grouped Join

JP(r,s) := r.x == s.x //definition of the join predicate

group(Tuple **):** Tuple ↦ [0, ..., k-1] //generalized grouping function

partition(Set, group() **):** (Set, group()) ↦ Set of Pair<Set, groupID> //generalized partitioning function

CoGroupedJoin(R, S, JP(r,s), group(), partition()):

```
Set of Pair<Set, groupID> build  := partition( R, group() ):   //partition R into subsets (aka groups)
Set of Pair<Set, groupID> probe := partition( S, group() );   //partition S into subsets (aka groups)

ForEach groupID in [0 to k-1]:                                //foreach existing unique groupID
    leftInput = build.getSet( groupID );                      //retrieve corresponding subset from R
    rightInput =probe.getSet( groupID );                      //retrieve corresponding subset from S
```

= co-group

Generalized Co-Grouped Join

JP(r,s) := r.x == s.x //definition of the join predicate

group(Tuple **):** Tuple ↦ [0, ..., k-1] //generalized grouping function

partition(Set, group() **):** (Set, group()) ↦ Set of Pair<Set, groupID> //generalized partitioning function

CoGroupedJoin(R, S, JP(r,s), group(), partition()):

```
Set of Pair<Set, groupID> build  := partition( R, group() ):   //partition R into subsets (aka groups)
Set of Pair<Set, groupID> probe := partition( S, group() );   //partition S into subsets (aka groups)

ForEach groupID in [0 to k-1]:                                //foreach existing unique groupID
    leftInput = build.getSet( groupID );                      //retrieve corresponding subset from R
    rightInput =probe.getSet( groupID );                      //retrieve corresponding subset from S
    If NOT leftInput.isEmpty() AND NOT rightInput.isEmpty():  //only if both inputs have some data
```

Generalized Co-Grouped Join

R S

JP(r,s) := r.x == s.x //definition of the join predicate

group(Tuple **):** Tuple ↦ [0, ..., k-1] //generalized grouping function

partition(Set, group() **):** (Set, group()) ↦ Set of Pair<Set, groupID> //generalized partitioning function

CoGroupedJoin(R, S, JP(r,s), group(), partition()):

```
    Set of Pair<Set, groupID> build  := partition( R, group() ):      //partition R into subsets (aka groups)
    Set of Pair<Set, groupID> probe := partition( S, group() );      //partition S into subsets (aka groups)

    ForEach groupID in [0 to k-1]:                                    //foreach existing unique groupID
        leftInput = build.getSet( groupID );                          //retrieve corresponding subset from R
        rightInput =probe.getSet( groupID );                          //retrieve corresponding subset from S
        If NOT leftInput.isEmpty() AND NOT rightInput.isEmpty():      //only if both inputs have some data
            WhateverJoin( leftInput, rightInput, JP(r,s) );           //call whatever join algorithm
```

only above: cross product why? groupID := r.x = s.x

Note: does not have to be like this in all special cases