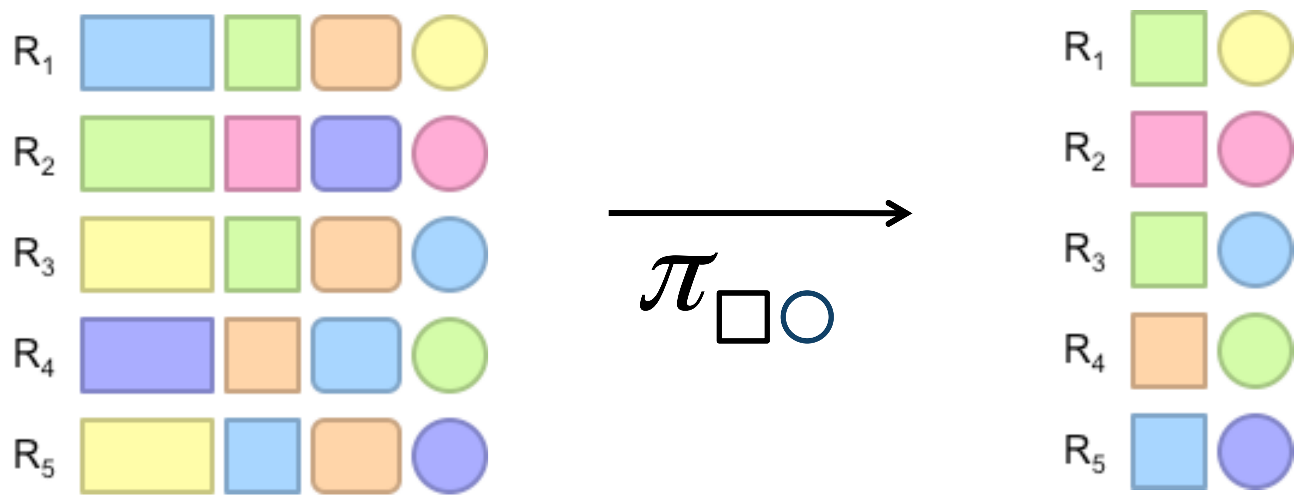
MapReduce: Algorithm Design for Relational Operations

Juliana Freire & Cláudio Silva

Some slides borrowed from Jimmy Lin, Jeff Ullman, Jerome Simeon, and Jure Leskovec

Projection

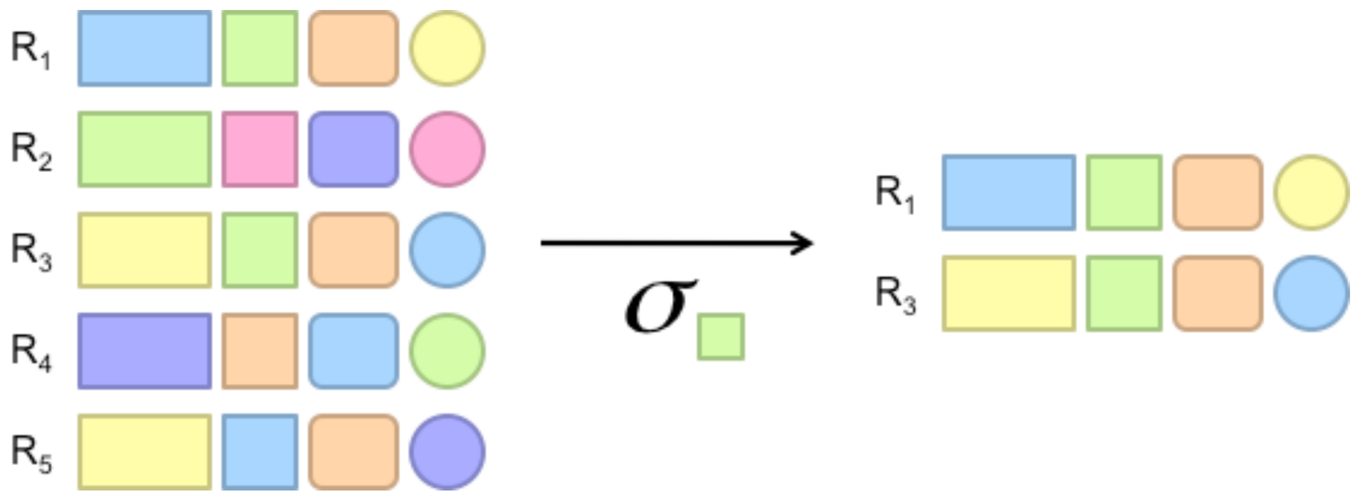


Projection in MapReduce

- Easy
  - Map over tuples, emit new tuples with appropriate attributes
  - No reducers, unless for regrouping or re-sorting tuples
  - Alternative: do projection in reducer, after some other processing

- Limited by HDFS streaming speeds
  - Speed of encoding/decoding tuples becomes important
  - Semi-structured (XML) data? No problem!

Selection



Selection in MapReduce

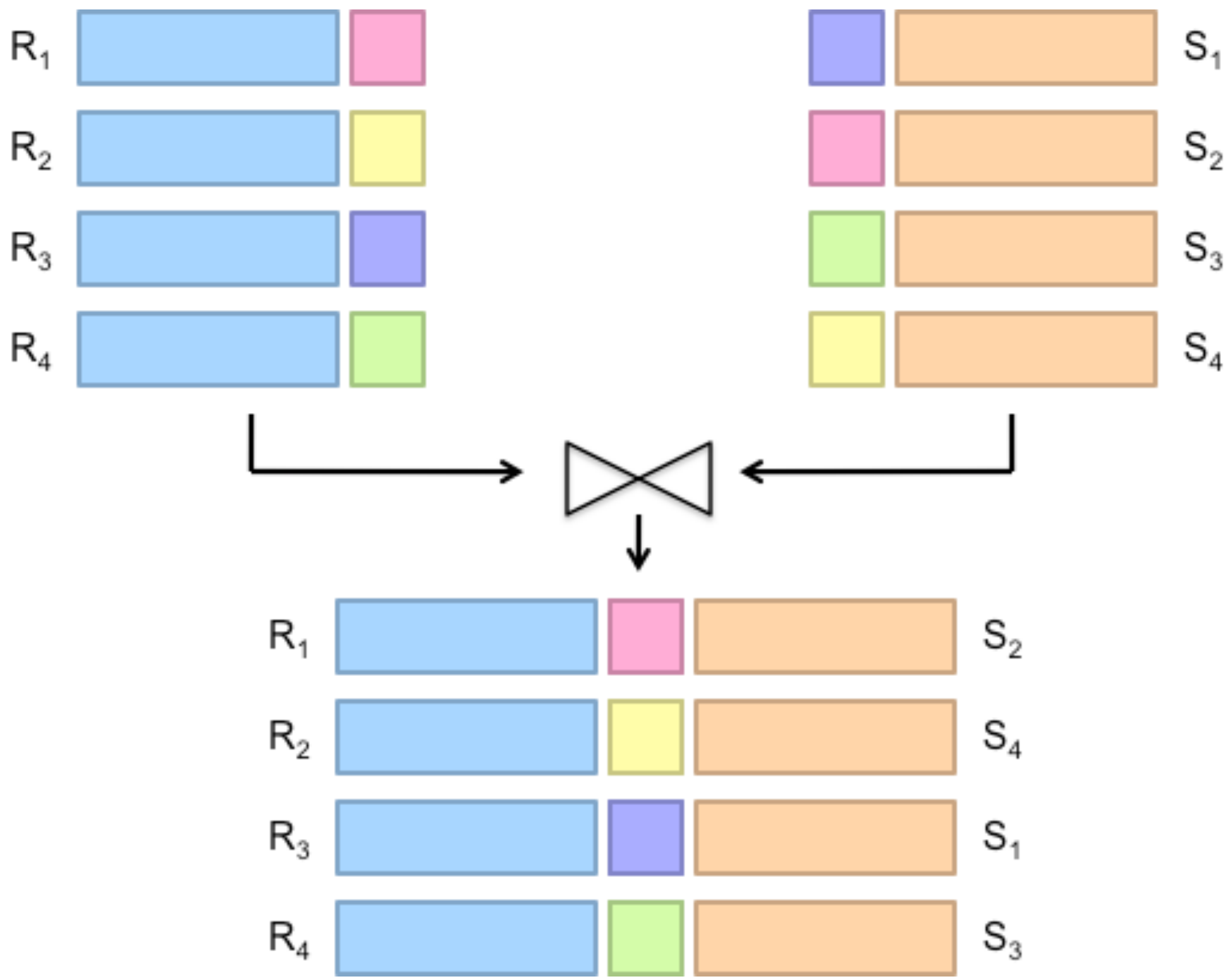
- Easy
  - Map over tuples, emit new tuples with appropriate attributes
  - No reducers, unless for regrouping or re-sorting tuples
  - Alternative: do projection in reducer, after some other processing

- Limited by HDFS streaming speeds
  - Speed of encoding/decoding tuples becomes important
  - Semi-structured (XML) data? No problem!

Aggregation in MapReduce

- Example – Log analysis: What is the average time spent per URL?
- In SQL:
  - SELECT url, AVG(time) FROM visits GROUP BY url
- In MapReduce:
  - Map over tuples, emit time, keyed by url
  - Framework *automatically* groups values by keys
  - Compute average in reducer
  - How can you make this more efficient?

Relational Joins



Running Example

- Two tables:
  - S: User demographics (user_id, gender, age, income, etc.)
  - R: User page visits (user_id, URL, time spent, etc.)

- Analyses we might want to perform:
  - Statistics on demographic characteristics
  - Statistics on page visits
  - Statistics on page visits by URL
  - Statistics on page visits by demographic characteristic

- Need to join S and R!

Join Algorithms in MapReduce

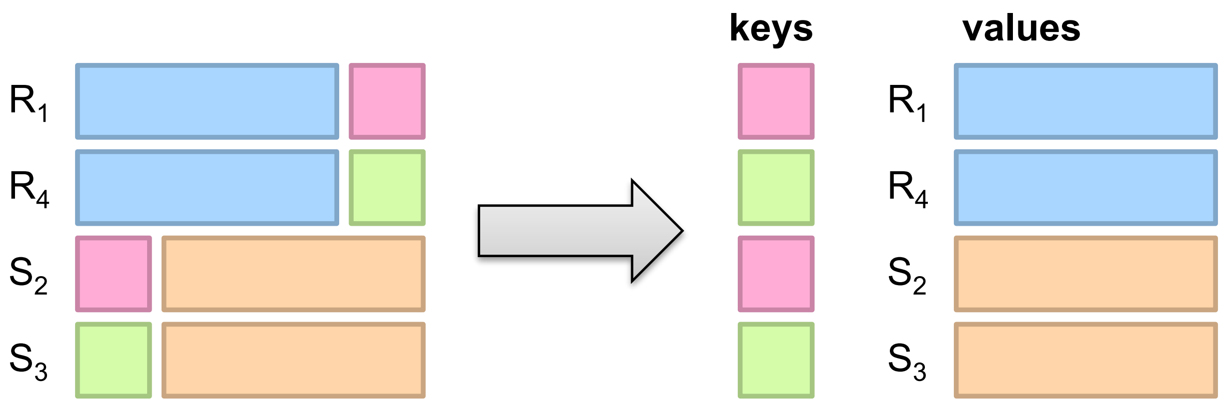
- Reduce-side join
- Map-side join
- In-memory join

Reduce-Side Join

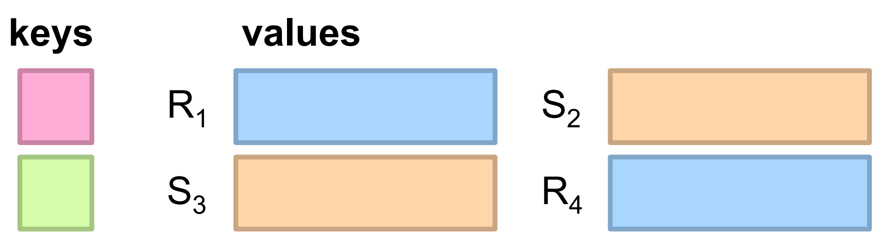
- Basic idea: *group by join key*
  - Map over both sets of tuples
  - Emit tuple as value with **join key as the intermediate key**
  - Execution framework brings together tuples sharing the same key
  - Perform actual join in reducer
  - Similar to a “sort-merge join” in database terminology
- Two variants
  - 1-to-1 joins
  - 1-to-many and many-to-many joins

Reduce-Side Join: 1-to-1

Map



Reduce



Note: no guarantee if R is going to come first or S

Reduce-Side Join: 1-to-1

- At most one tuple from R and one tuple from S share the same join key
- Reducer will receive keys in the following format

$k23 \rightarrow [(r64, R_{64}), (s84, S_{84})]$

$k37 \rightarrow [(r68, R_{68})]$

$k59 \rightarrow [(s97, S_{97}), (r81, R_{81})]$

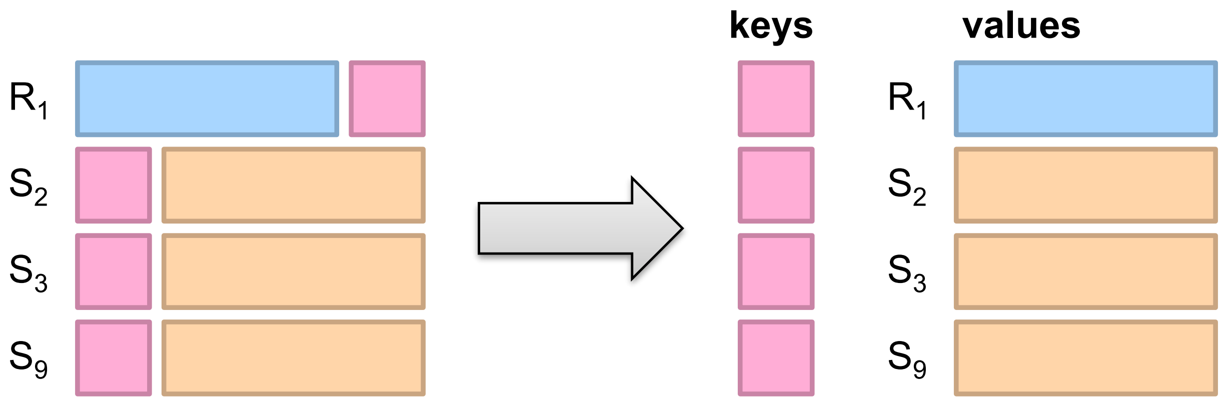
$k61 \rightarrow [(s99, S_{99})]$

S_i → other attr of S
R_i → other attr of R
si,ri → tuple id

- If there are two values associated with a key, one must be from R and the other from S
- What should the reduced do for k_{37}?

Reduce-Side Join: 1-to-many

Map



Reduce

R is the one side, S is the many

Reduce-Side Join: 1-to-many

- Reducer will receive keys in the following format

k23 $\rightarrow$[(r64,R_{64}),(s84,S_{84}),(r65, R_{65})]

k37 $\rightarrow$ [(r68, R_{68})]

k59 $\rightarrow$[(s97,S_{97}),(s98,S_{98}), (s99,S_{99}),
(s100,S_{100}), (r81,R_{81})]

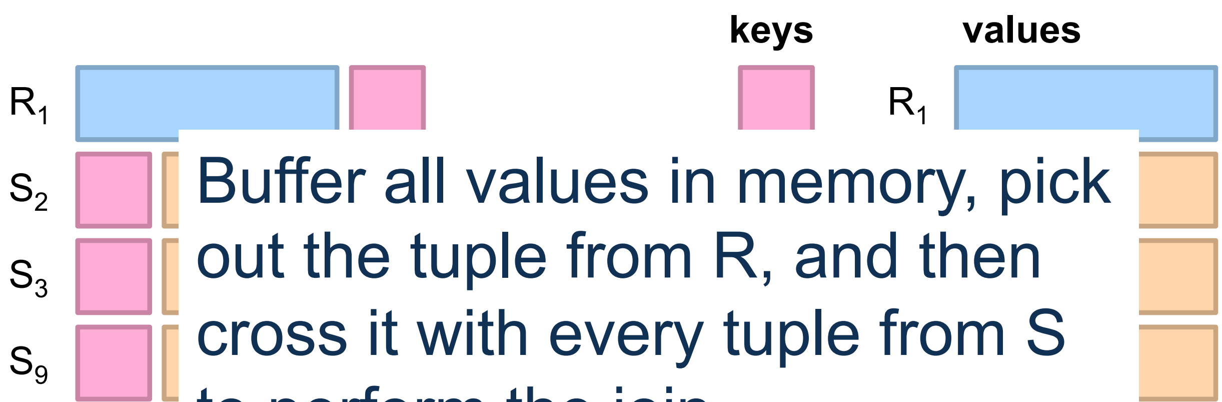
k61 $\rightarrow$ [(s99, S_{99})]

S_i → other attr of S
R_i → other attr of R
si,ri→ tuple id

- There can be many values from S for a given key, we do not know when the value for R will be encountered

Reduce-Side Join: 1-to-many

Map



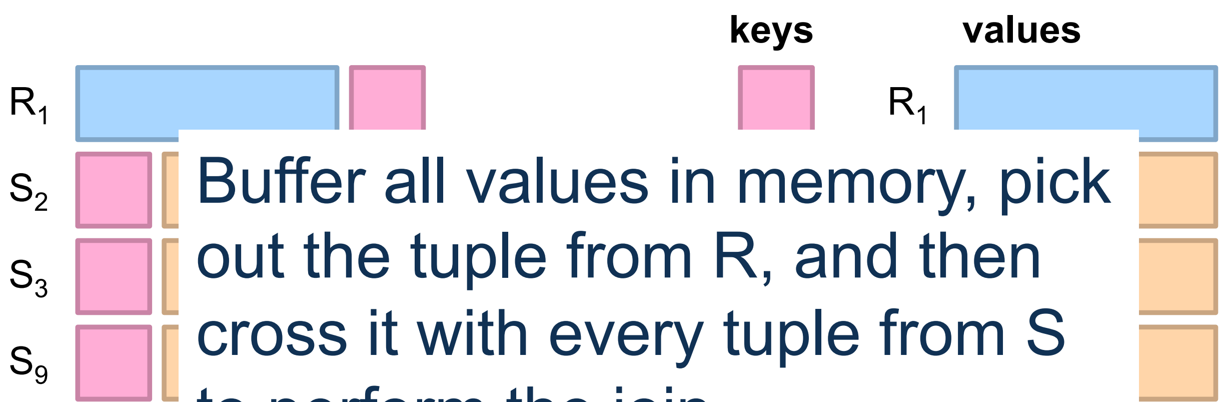
Buffer all values in memory, pick out the tuple from R, and then cross it with every tuple from S to perform the join

Reduce

R is the one side, S is the many

Reduce-Side Join: 1-to-many

Map



Buffer all values in memory, pick out the tuple from R, and then cross it with every tuple from S to perform the join

Reduce

What's the problem?

R is the one side, S is the many

Sorting: Use Values

- "Value-to-key conversion" design pattern: form composite intermediate key, (k, v_1)
- Let execution framework do the sorting: Sort by join attribute, then sort all tuple ids from R before S

Before $k \rightarrow (s97, S_{97}), (s98, S_{98}), (s99, S_{99}), (s100, S_{100}), (r81, R_{81})\ldots$ Values arrive in arbitrary order…

After $(k, r81) \rightarrow (r81, R_{81})$ Values arrive in sorted order of tuple id

$(k, s97) \rightarrow (s97, S_{97})$

$(k, s98) \rightarrow (s98, S_{98})$

$(k, s99) \rightarrow (s99, S_{99})$

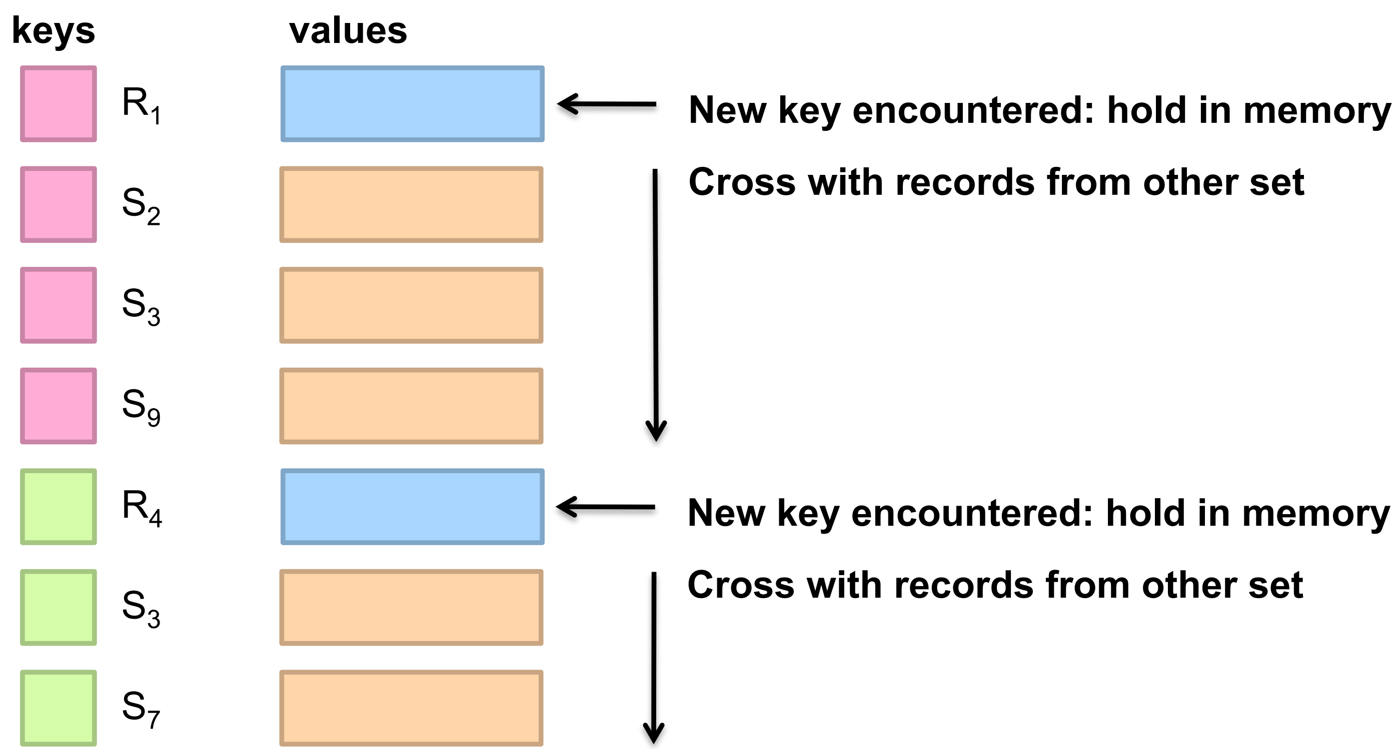
…

Process by preserving state across multiple keys

Remember to partition correctly!

Reduce-Side Join: Value-to-Key Conversion

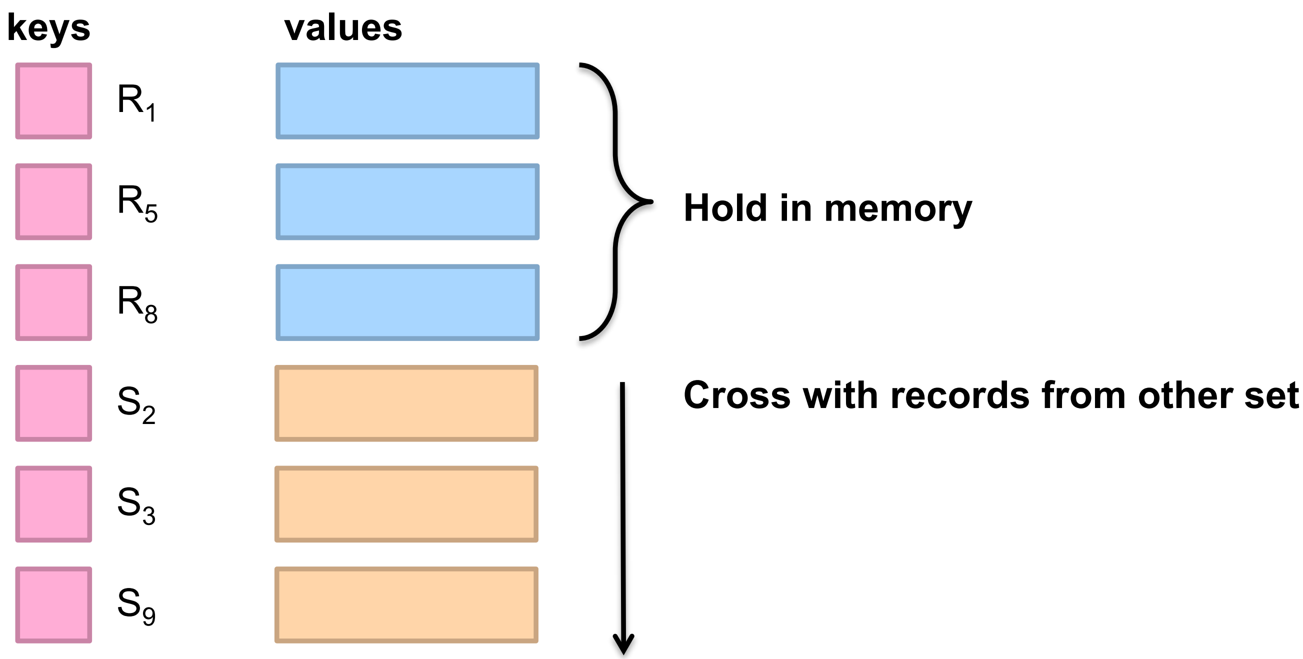
In reducer…



https://www.inkling.com/read/hadoop-definitive-guide-tom-white-3rd/chapter-8/example-8-9

Reduce-Side Join Many-to-Many

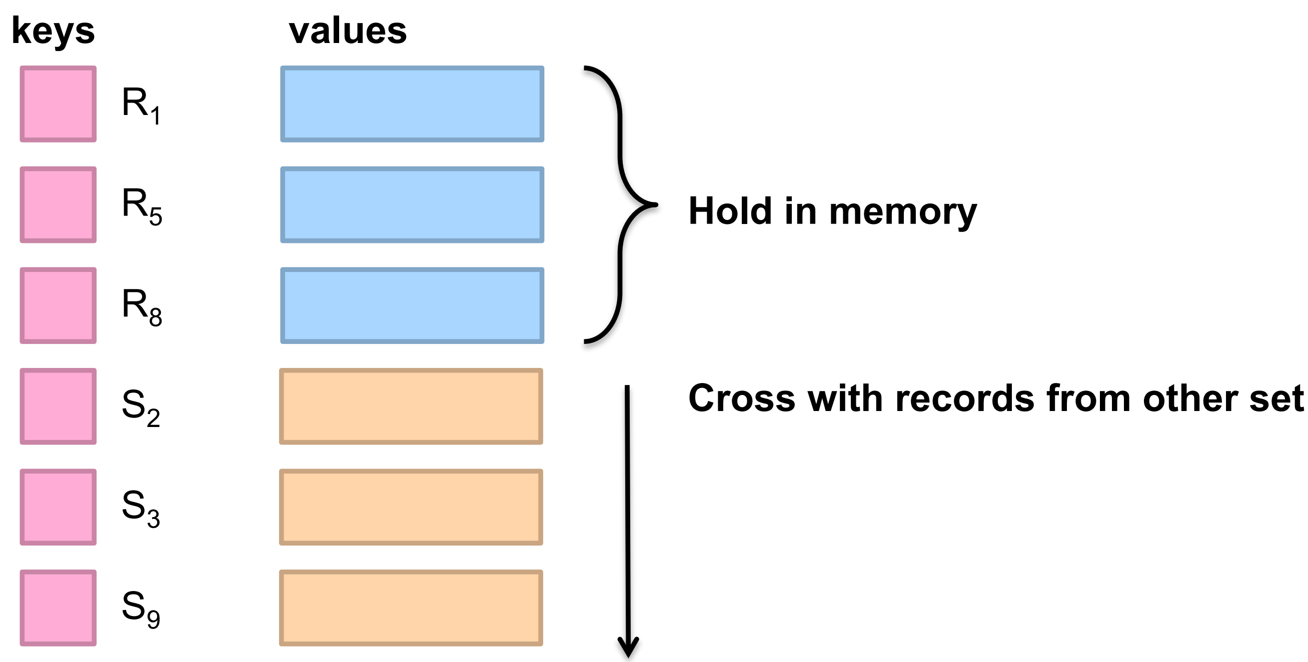
In reducer…



R is the smaller dataset

Reduce-Side Join Many-to-Many

In reducer…



What's the problem?

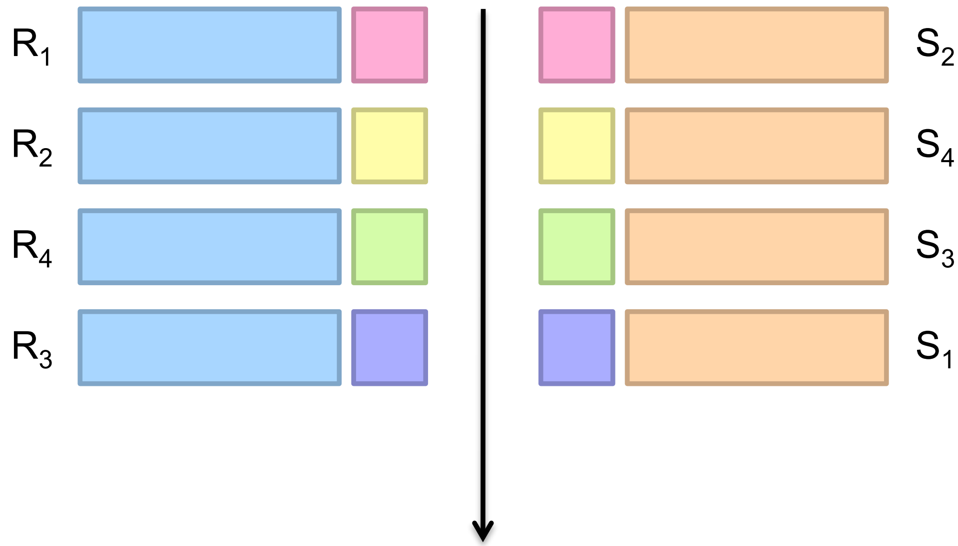
R is the smaller dataset

Reduce-Side Join

- What are the limitations?
  - Both datasets are transferred over the network!

Map-Side Join: Basic Idea

Assume two datasets are sorted by the join key:



A sequential scan through both datasets to join
(called a "merge join" in database terminology)

Map-Side Join: Parallel Scans

- If datasets are *sorted by join key*, join can be accomplished by a scan over both datasets
- How can we accomplish this in parallel?
  - Partition and sort both datasets in the same manner
- In MapReduce:
  - Map over one dataset, read from other corresponding partition
  - No reducers necessary (unless to repartition or resort)
- Consistently partitioned datasets: realistic to expect?
  - Depends on the workflow
  - For ad hoc data analysis, reduce-side are more general, although less efficient

References

- Data Intensive Text Processing with MapReduce, Lin and Dyer (Chapter 3)
- Mining of Massive Data Sets, Rajaraman et al. (Chapter 2)
- Hadoop tutorial: https://developer.yahoo.com/hadoop/tutorial/module4.html
- Jeffrey Dean and Sanjay Ghemawat, MapReduce: Simplified Data Processing on Large Clusters

http://labs.google.com/papers/mapreduce.html

- Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, The Google File System

http://labs.google.com/papers/gfs.html