MapReduce and Data Analysis

Juliana Freire
New York University

Some slides from J. Lin, J. Simeon

NYU | POLYTECHNIC SCHOOL OF ENGINEERING

CENTER FOR URBAN SCIENCE+PROGRESS

Big Data Analysis

- Peta-scale datasets are everywhere:
  - Facebook has 2.5 PB of user data + 15 TB/day (4/2009)
  - eBay has 6.5 PB of user data + 50 TB/day (5/2009)
  - …
- A lot of these datasets have some structure
  - Query logs
  - Point-of-sale records
  - User data (e.g., demographics)
  - …
- How do we perform data analysis at scale?
  - Relational databases and SQL
  - MapReduce (Hadoop)

Relational Databases vs. MapReduce

- Relational databases:
  - Multipurpose: analysis and transactions; batch and interactive
  - Data integrity via ACID transactions
  - Lots of tools in software ecosystem (for ingesting, reporting, etc.)
  - Supports SQL (and SQL integration, e.g., JDBC)
  - Automatic SQL query optimization

- MapReduce (Hadoop):
  - Designed for large clusters, fault tolerant
  - Data is accessed in “native format”
  - Supports many programming and query languages
  - Programmers retain control over performance
  - Open source

Source: O’Reilly Blog post by Joseph Hellerstein (11/19/2008)

Database Workloads

- OLTP (online transaction processing)
  - Typical applications: e-commerce, banking, airline reservations
  - User facing: real-time, low latency, highly-concurrent
  - Tasks: relatively small set of "standard" transactional queries
  - Data access pattern: random reads, updates, writes (involving relatively small amounts of data)

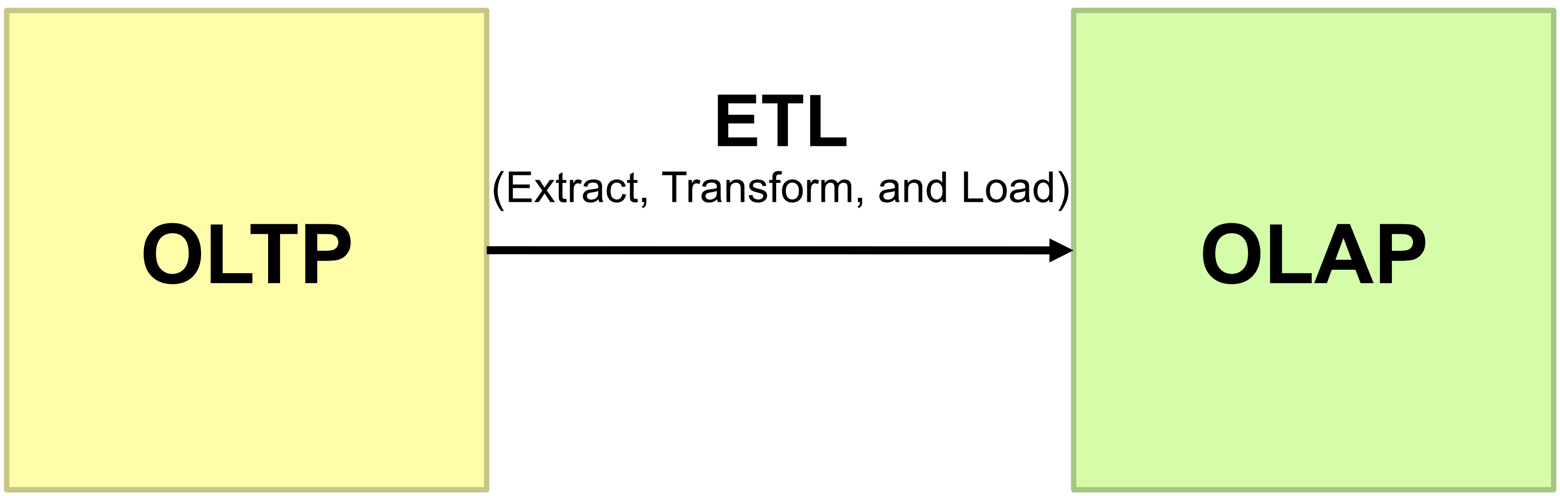
- OLAP (online analytical processing)
  - Typical applications: business intelligence, data mining
  - Back-end processing: batch workloads, less concurrency
  - Tasks: complex analytical queries, often ad hoc
  - Data access pattern: table scans, large amounts of data involved per query

One Database or Two?

- Downsides of co-existing OLTP and OLAP workloads
  - Poor memory management
  - Conflicting data access patterns
  - Variable latency

- Solution: separate databases
  - User-facing OLTP database for high-volume transactions
  - Data warehouse for OLAP workloads
  - How do we connect the two?

OLTP/OLAP Architecture



OLTP/OLAP Integration

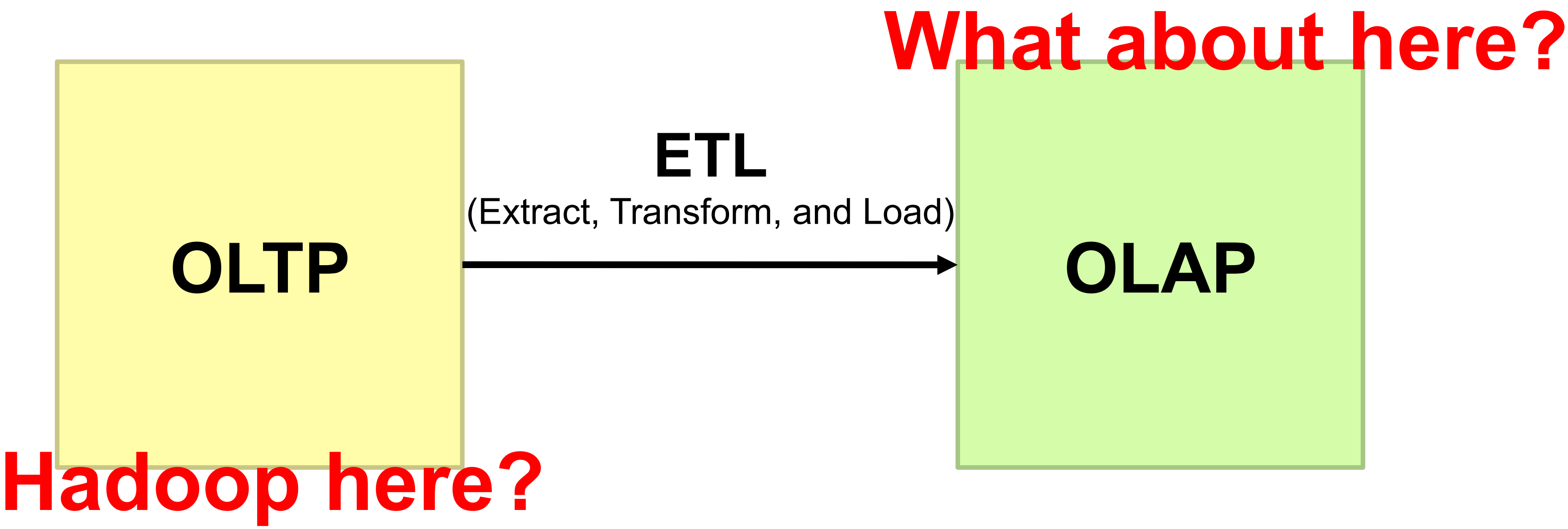
- OLTP database for user-facing transactions
  - Retain records of all activity
  - Periodic ETL (e.g., nightly)
- Extract-Transform-Load (ETL)
  - Extract records from source
  - Transform: clean data, check integrity, aggregate, etc.
  - Load into OLAP database
- OLAP database for data warehousing
  - Business intelligence: reporting, ad hoc queries, data mining, etc.
  - Feedback to improve OLTP services

Business Intelligence

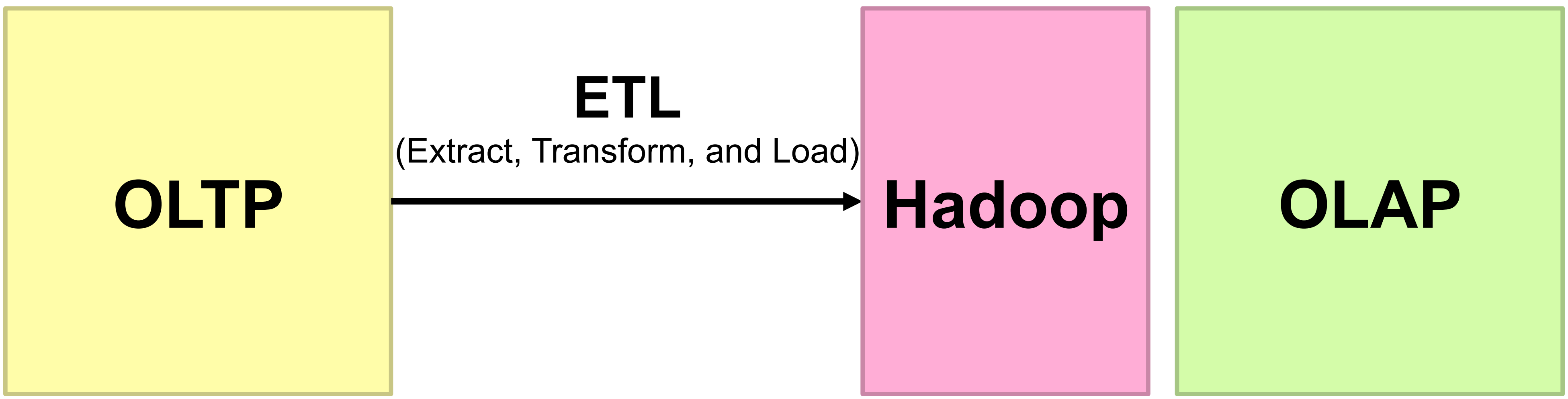
- Premise: more data leads to better business decisions
  - Periodic reporting as well as ad hoc queries
  - Analysts, not programmers (importance of tools and dashboards)

- Examples:
  - Slicing-and-dicing activity by different dimensions to better understand the marketplace
  - Analyzing log data to improve OLTP experience
  - Analyzing log data to better optimize ad placement
  - Analyzing purchasing trends for better supply-chain management
  - Mining for correlations between otherwise unrelated activities

OLTP/OLAP Architecture: Hadoop?



OLTP/OLAP/Hadoop Architecture



Why does this make sense?

ETL Bottleneck

- Reporting is often a nightly task:
  - ETL is often slow: why?
  - What happens if processing 24 hours of data takes longer than 24 hours?
- Hadoop is perfect:
  - Most likely, you already have some data warehousing solution
  - Ingest is limited by speed of HDFS
  - Scales out with more nodes
  - Massively parallel
  - Ability to use any processing tool
  - Much cheaper than parallel databases
  - ETL is a batch process anyway!

MapReduce Algorithms for Processing Relational Data

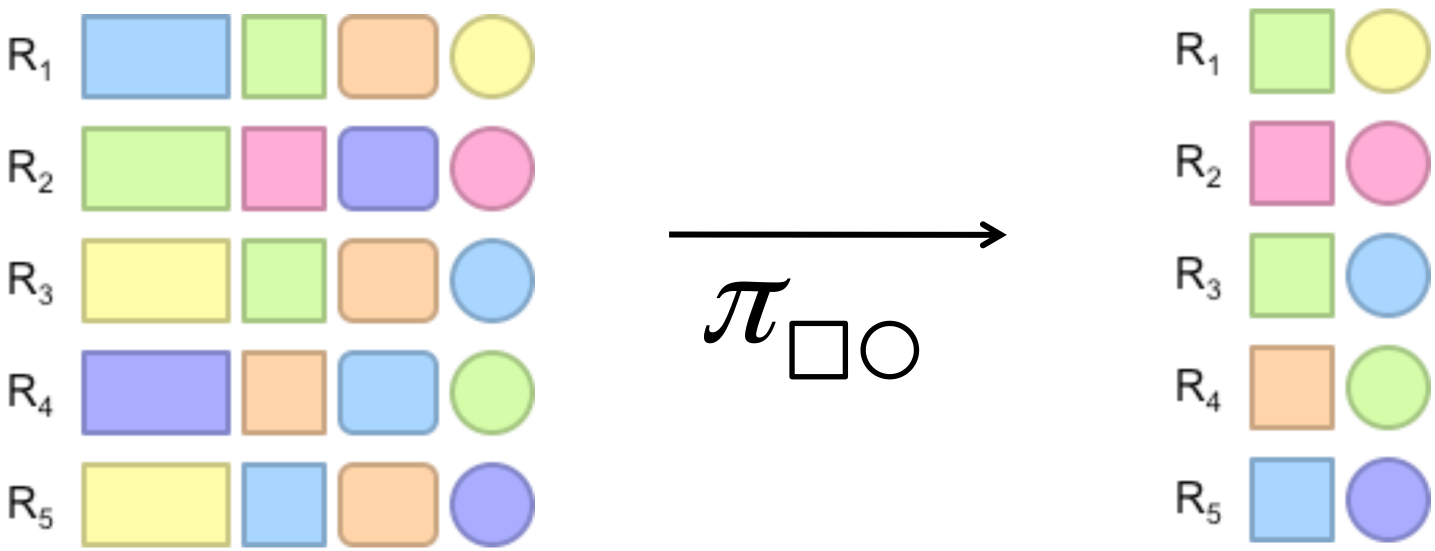
NYU | POLYTECHNIC SCHOOL OF ENGINEERING

CENTER FOR URBAN SCIENCE+PROGRESS

Relational Algebra

- Primitives
  - Projection (π)
  - Selection (σ)
  - Cartesian product ($\times$)
  - Set union ($\cup$)
  - Set difference (-)
  - Rename (ρ)
- Other operations
  - Join ($\bowtie$)
  - Group by… aggregation
  - …

Projection

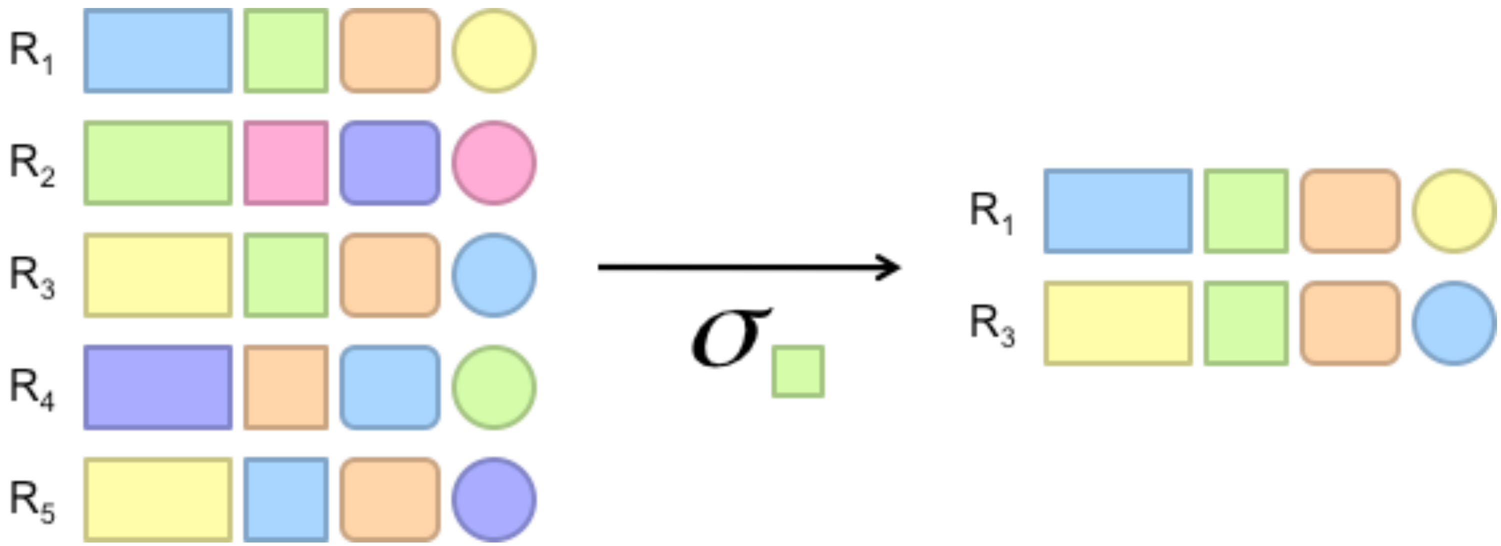


Projection in MapReduce

- Easy
  - Map over tuples, emit new tuples with appropriate attributes
  - No reducers, unless for regrouping or resorting tuples
  - Alternatively: perform in reducer, after some other processing

- Basically limited by HDFS streaming speeds
  - Speed of encoding/decoding tuples becomes important
  - Semistructured data? No problem!

Selection



Selection in MapReduce

- Easy
  - Map over tuples, emit only tuples that meet criteria
  - No reducers, unless for regrouping or resorting tuples
  - Alternatively: perform in reducer, after some other processing

- Basically limited by HDFS streaming speeds
  - Speed of encoding/decoding tuples becomes important
  - Semistructured data? No problem!

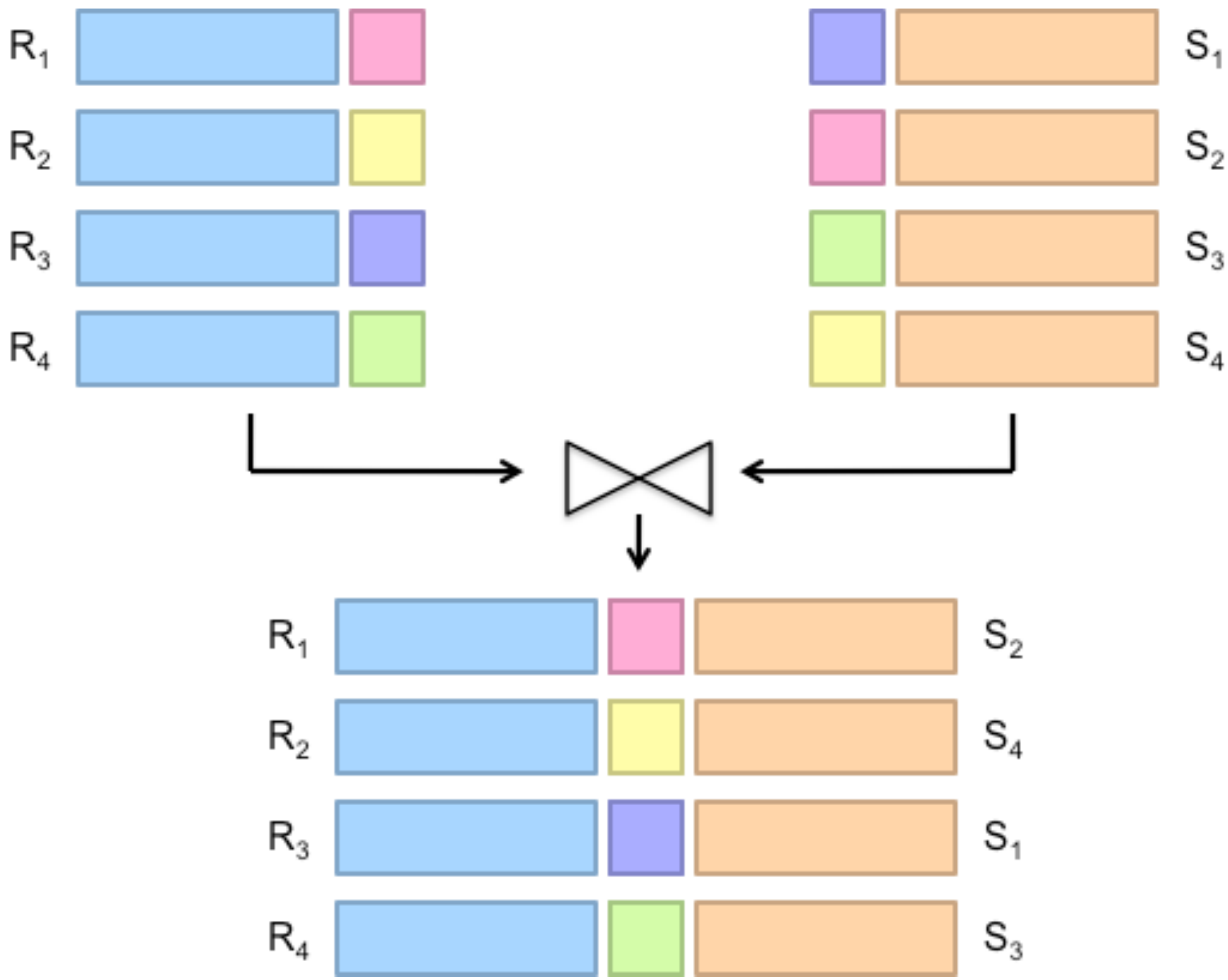
Group by... Aggregation

- Example: What is the average time spent per URL?

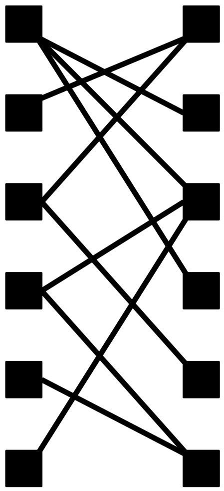
- In SQL:
  - SELECT url, AVG(time) FROM visits GROUP BY url

- 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?

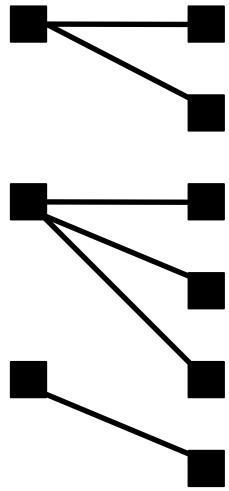
Relational Joins



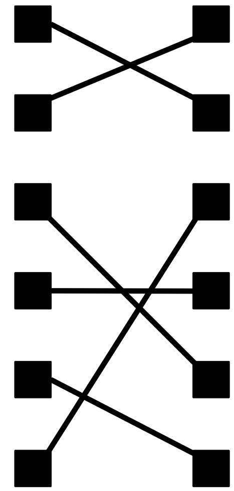
Types of Relationships



Many-to-Many



One-to-Many



One-to-One

Working Scenario

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

- 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
  - …

Join Algorithms in MapReduce

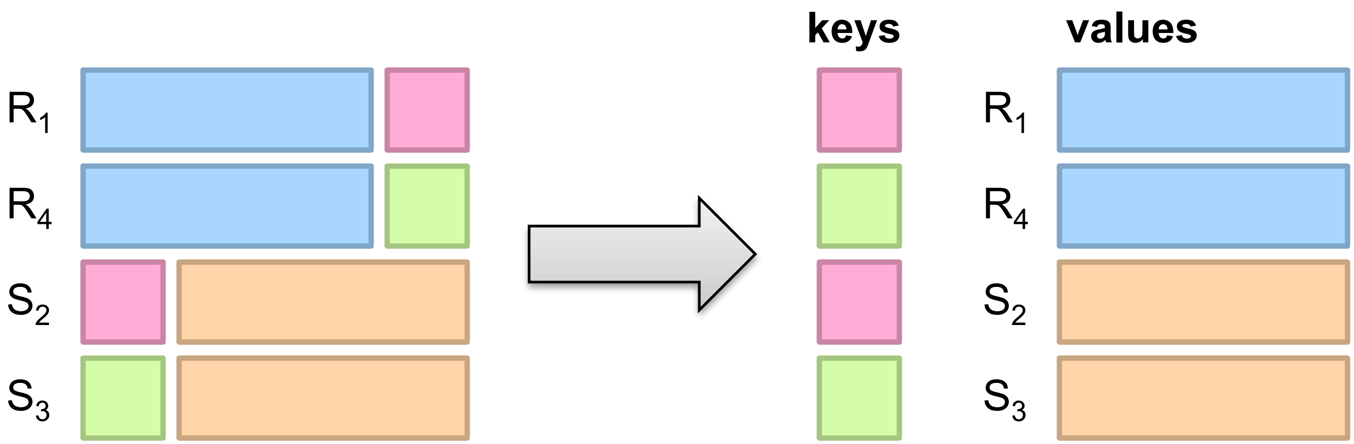
- Reduce-side join
- Map-side join
- In-memory join

Reduce-Side Join

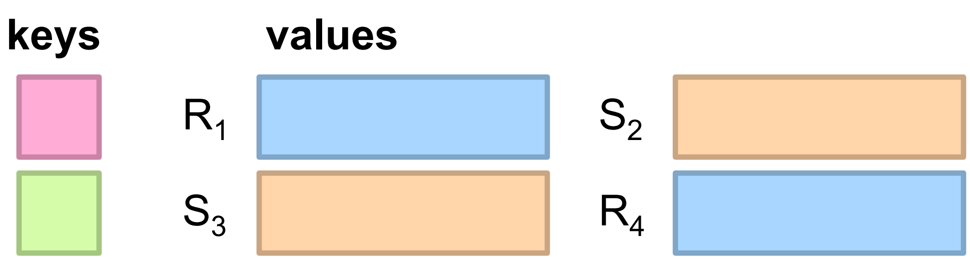
- Basic idea: group by join key
  - Map over both sets of tuples
  - Emit tuple as value with **join key as the intermediate key**
  - Execution framework brings together tuples sharing the same key
  - Perform actual join in reducer
  - Similar to a “sort-merge join” in database terminology
- Two variants
  - 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 →[(r64,R64),(s84,S84)]

  k37 → [(r68, R68)]

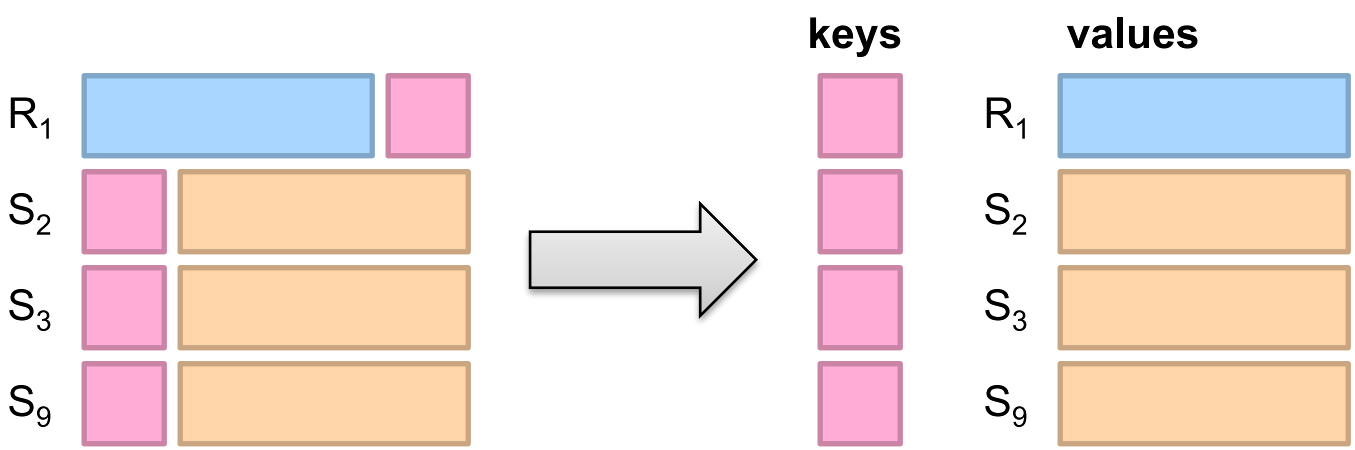
  k59 →[(s97,S97),(r81,R81)]

  k61 → [(s99, S99)]

- 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

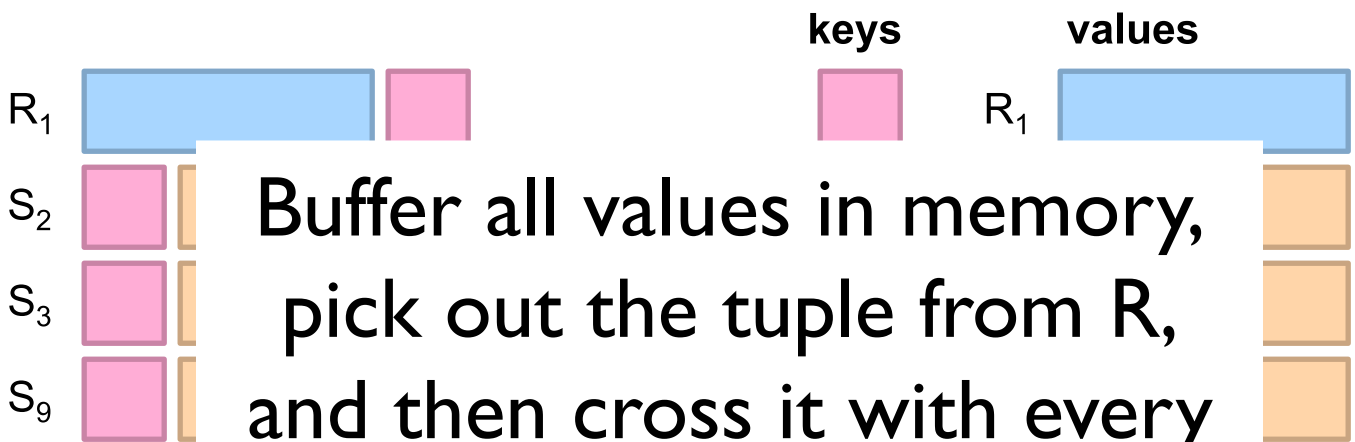


Reduce

R is the one side, S is the many

Reduce-Side Join: 1-to-many

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

Before

$$k \rightarrow (v_1, r), (v_4, r), (v_8, r), (v_3, r)\ldots$$

Values arrive in arbitrary order…

After

$$(k, v_1) \rightarrow (v_1, r)$$
$$(k, v_3) \rightarrow (v_3, r)$$
$$(k, v_4) \rightarrow (v_4, r)$$
$$(k, v_8) \rightarrow (v_8, r)$$
$$\ldots$$

Values arrive in sorted order…

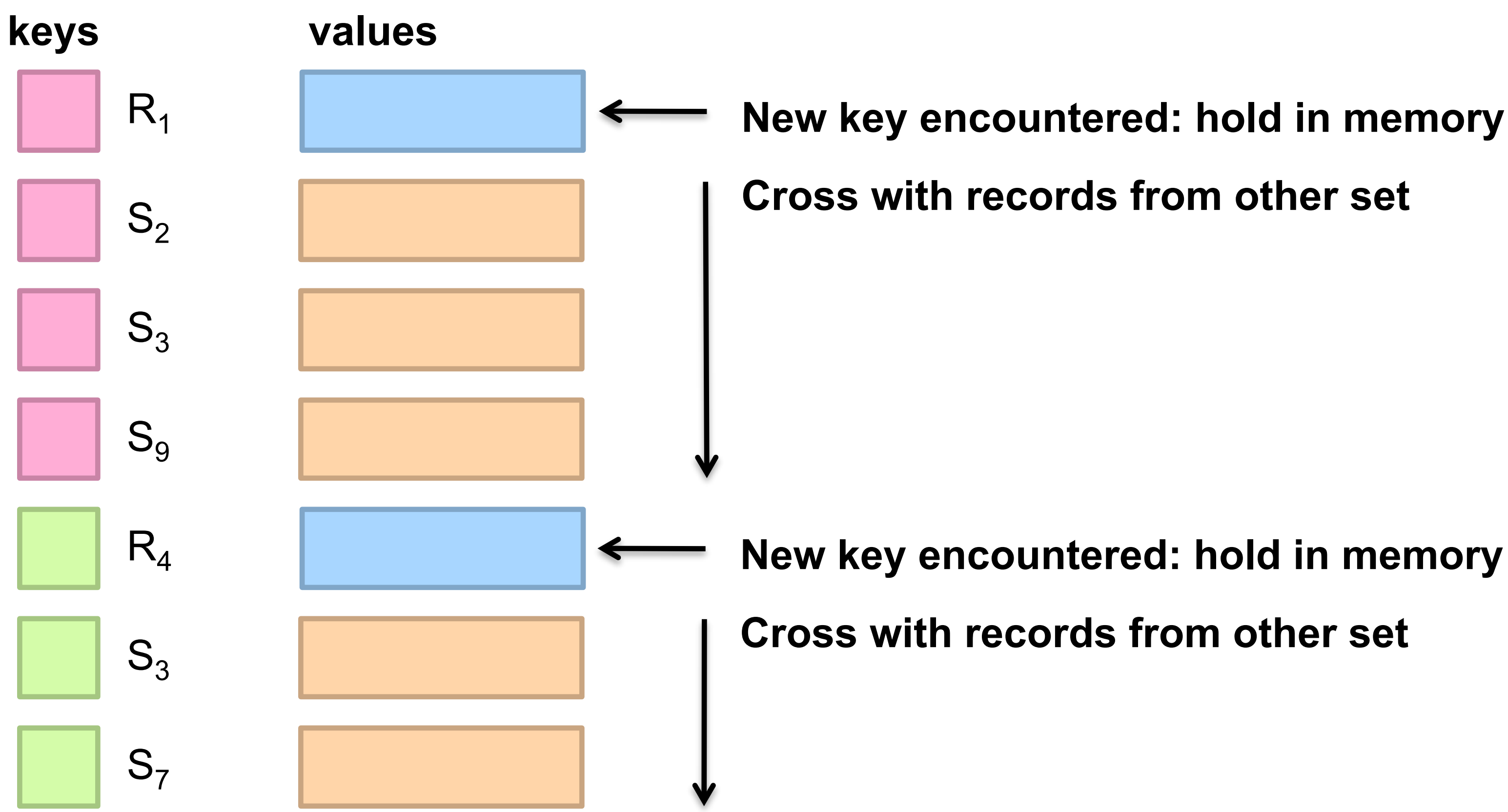
Process by preserving state across multiple keys

Remember to partition correctly!

- Anything else we need to do?

Reduce-Side Join: V-to-K Conversion

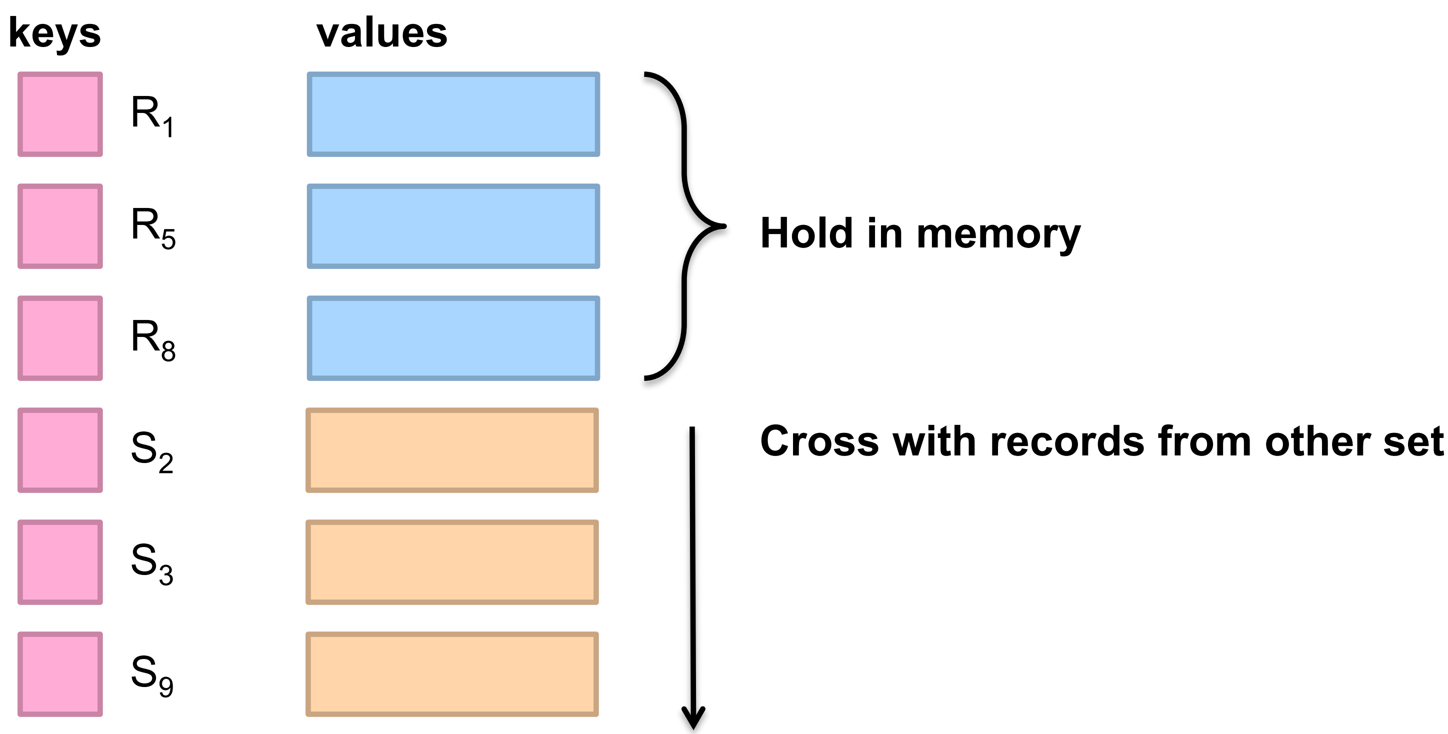
In reducer…



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

Reduce-side Join: many-to-many

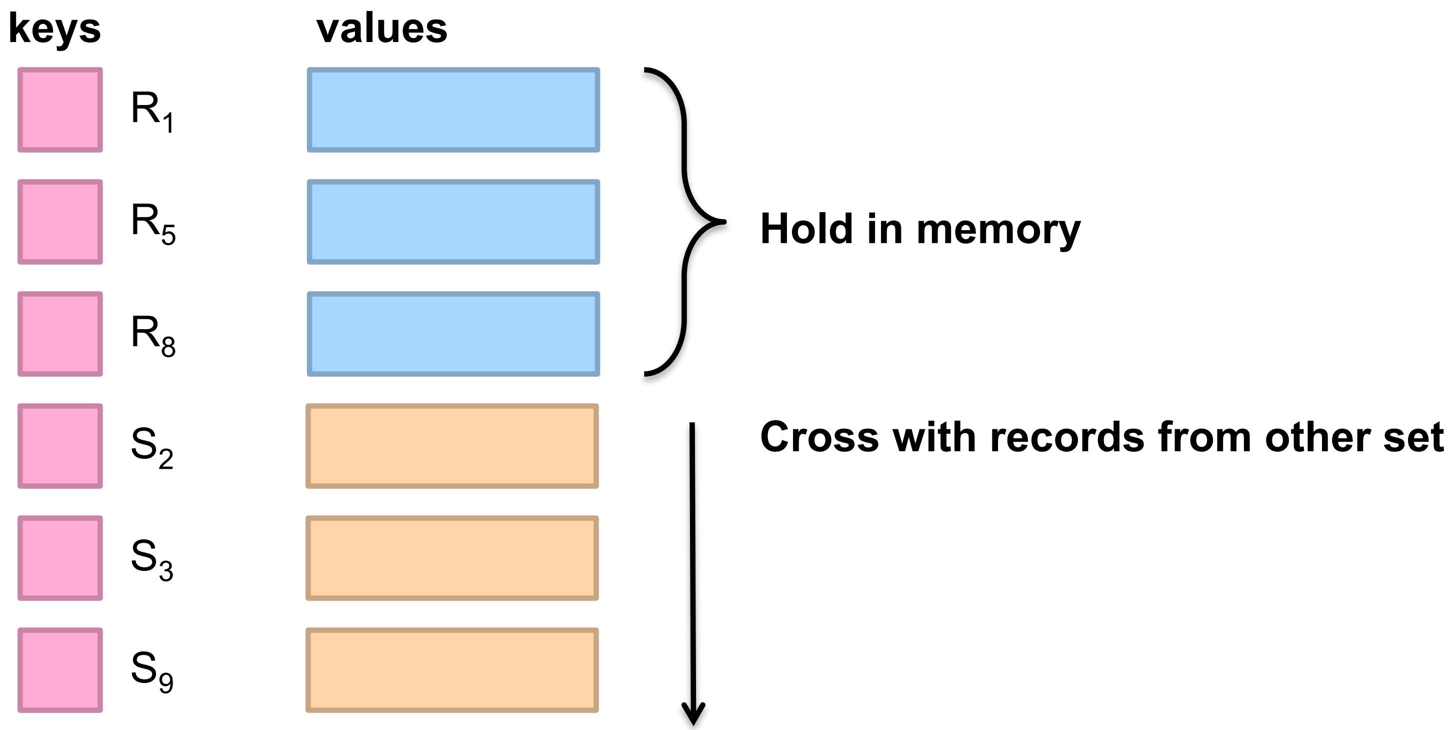
In reducer…



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

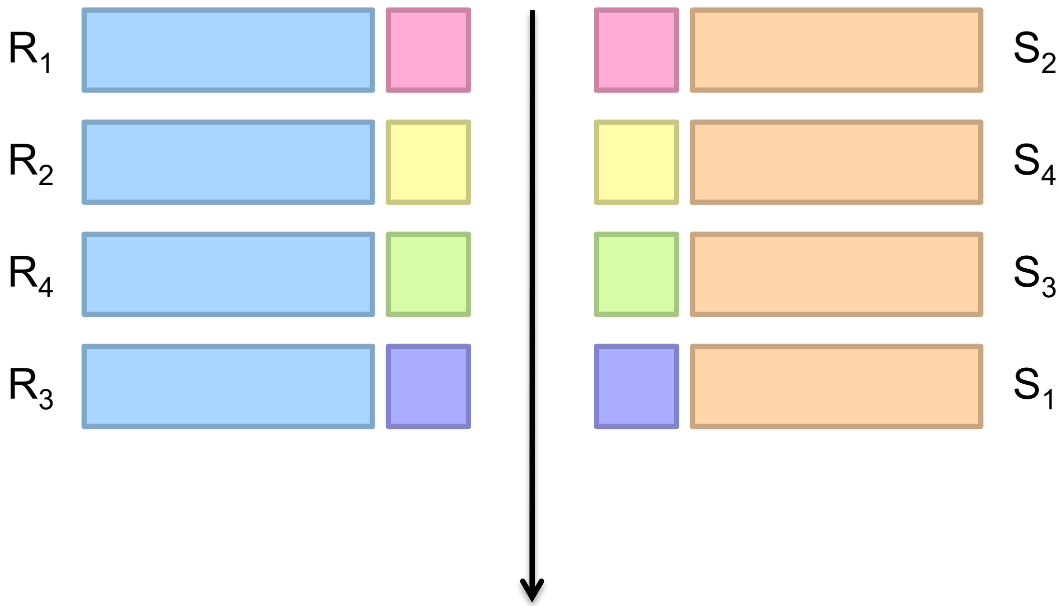
- What are the limitations?

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

In-Memory Join

- Basic idea: load one dataset into memory, stream over other dataset
  - Works if R << S and R fits into memory
  - Called a “hash join” in database terminology
- MapReduce implementation
  - Distribute R to all nodes
  - Map over S, each mapper loads R in memory, hashed by join key
  - For every tuple in S, look up join key in R
  - No reducers, unless for regrouping or resorting tuples

In-Memory Join: Variants

- Striped variant:
  - R too big to fit into memory?
  - Divide R into R_1, R_2, R_3, … s.t. each R_n fits into memory
  - Perform in-memory join: $\forall n$, $R_n \bowtie S$
  - Take the union of all join results

Which join to use?

- In-memory join > map-side join > reduce-side join
  - Why?
- Limitations of each?
  - In-memory join: memory
  - Map-side join: sort order and partitioning
  - Reduce-side join: general purpose

Processing Relational Data: Summary

- MapReduce algorithms for processing relational data:
  - Group by, sorting, partitioning are handled automatically by shuffle/sort in MapReduce
  - Selection, projection, and other computations (e.g., aggregation), are performed either in mapper or reducer
  - Multiple strategies for relational joins
- Complex operations require multiple MapReduce jobs
  - Example: top ten URLs in terms of average time spent
  - Opportunities for automatic optimization

Questions?

NYU | POLYTECHNIC SCHOOL OF ENGINEERING

CENTER FOR URBAN SCIENCE+PROGRESS