Cloud Programming: Lecture7 – Query Processing on MapReduce

National Tsing-Hua University

2016, Spring Semester

國立清華大學

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 are (mostly) structured
  - 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)

Outline

- Role of relational databases in today's organizations
  - Where does MapReduce fit in?
- MapReduce algorithms for processing relational data
  - How do I perform a query operations?
- Evolving roles of relational databases and MapReduce
  - What's in store for the future?

Outline

- Role of relational databases in today's organizations
  - Where does MapReduce fit in?
- MapReduce algorithms for processing relational data
  - How do I perform a query operations?
- Evolving roles of relational databases and MapReduce
  - What's in store for the future?

Relational Databases vs. MapReduce

	Relational databases	MapReduce (Hadoop)
Workload		
Goal		
Data access		
Support		
Programmability		
Cost		
Performance		

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?

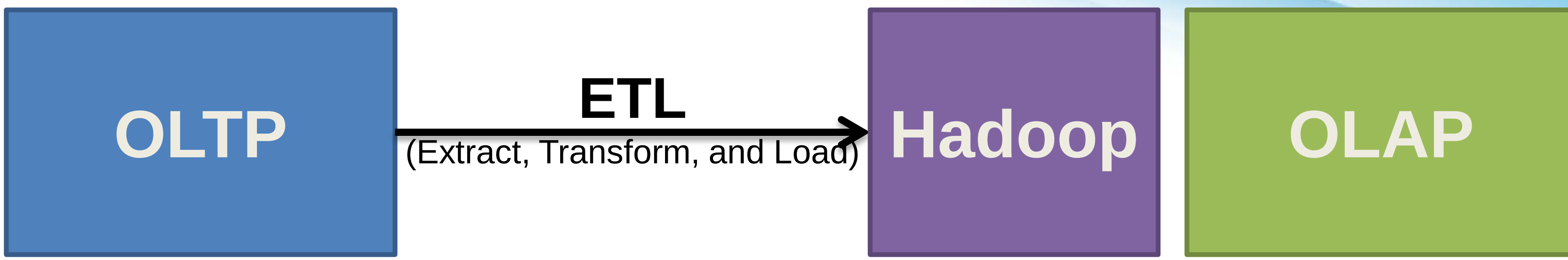
- Advantage of one database/architecture
  - Only need to maintain and learn a single system & programing model
- Downsides of co-existing OLTP and OLAP workloads
  - Poor memory management
  - Conflicting data access patterns
  - Variable latency
- Solution: separate databases
  - User-facing OLTP database for high-volume transactions
  - Data warehouse for OLAP workloads

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:
    - **Periodic reporting** as well as **ad hoc queries**
    - **Analysts, not programmers** (importance of tools and dashboards)
  - Feedback to improve OLTP services

Big Data

OLTP/OLAP/Hadoop Architecture



- Why it make sense?

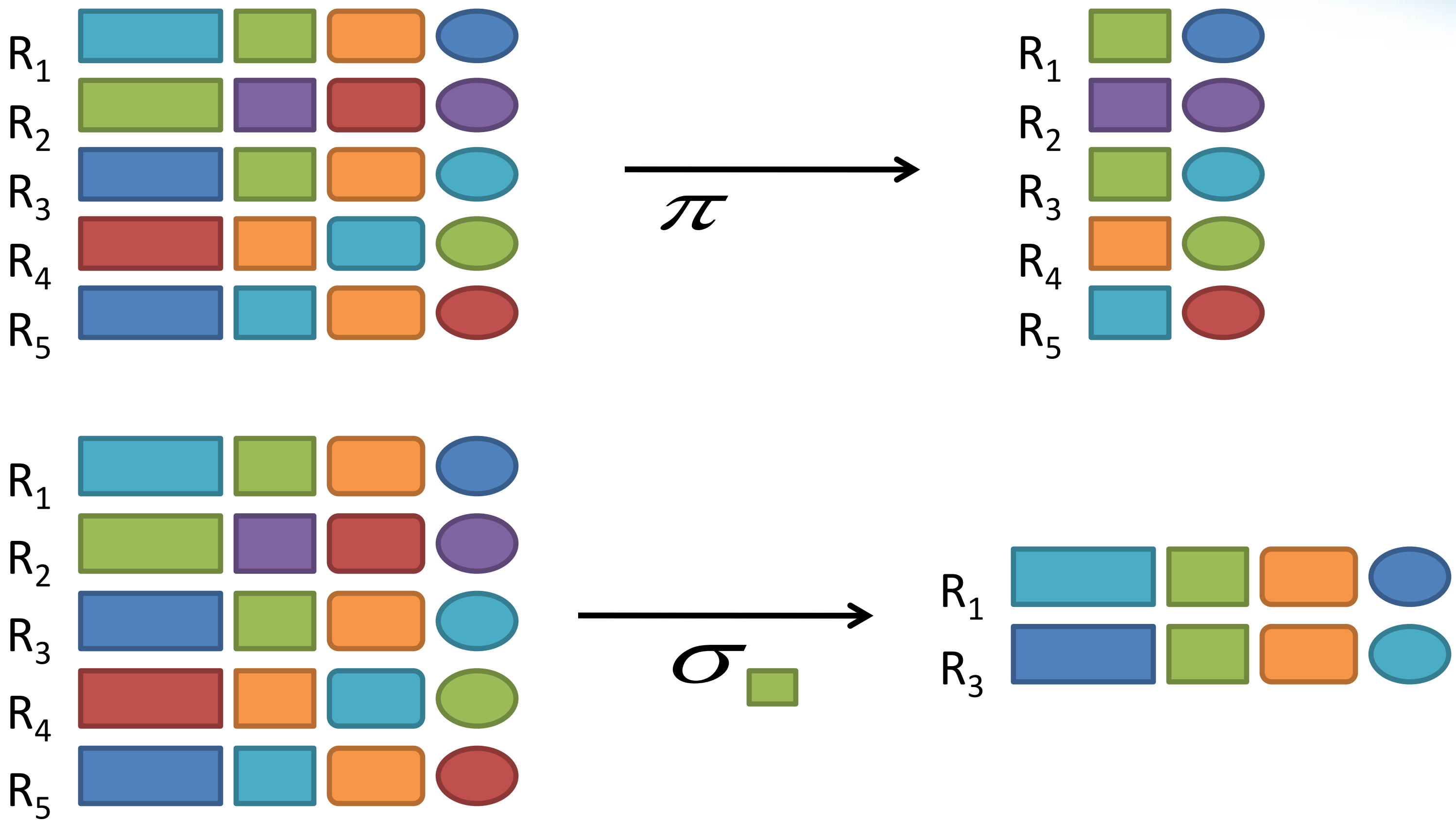
Outline

- Role of relational databases in today's organizations
  - Where does MapReduce fit in?
- MapReduce algorithms for processing relational data
  - How do I perform a join, etc.?
- Evolving roles of relational databases and MapReduce
  - What's in store for the future?

Relational Algebra

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

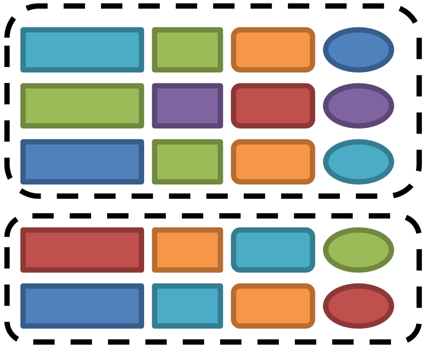
Projection & Selection

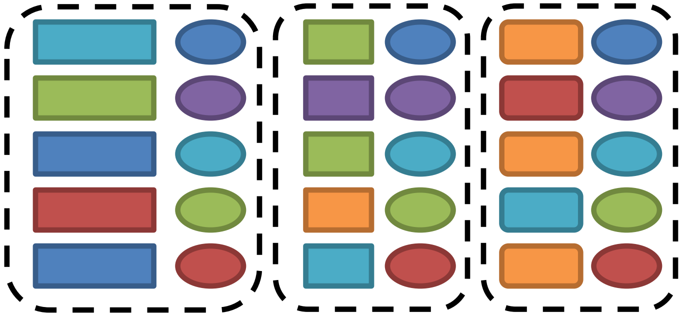


Projection & Section in MapReduce

- MapReduce Implementation

- Basically limited by HDFS streaming speeds
  - Speed of **encoding/decoding tuples** becomes important
    - Columns-based? Row-based? Compression?

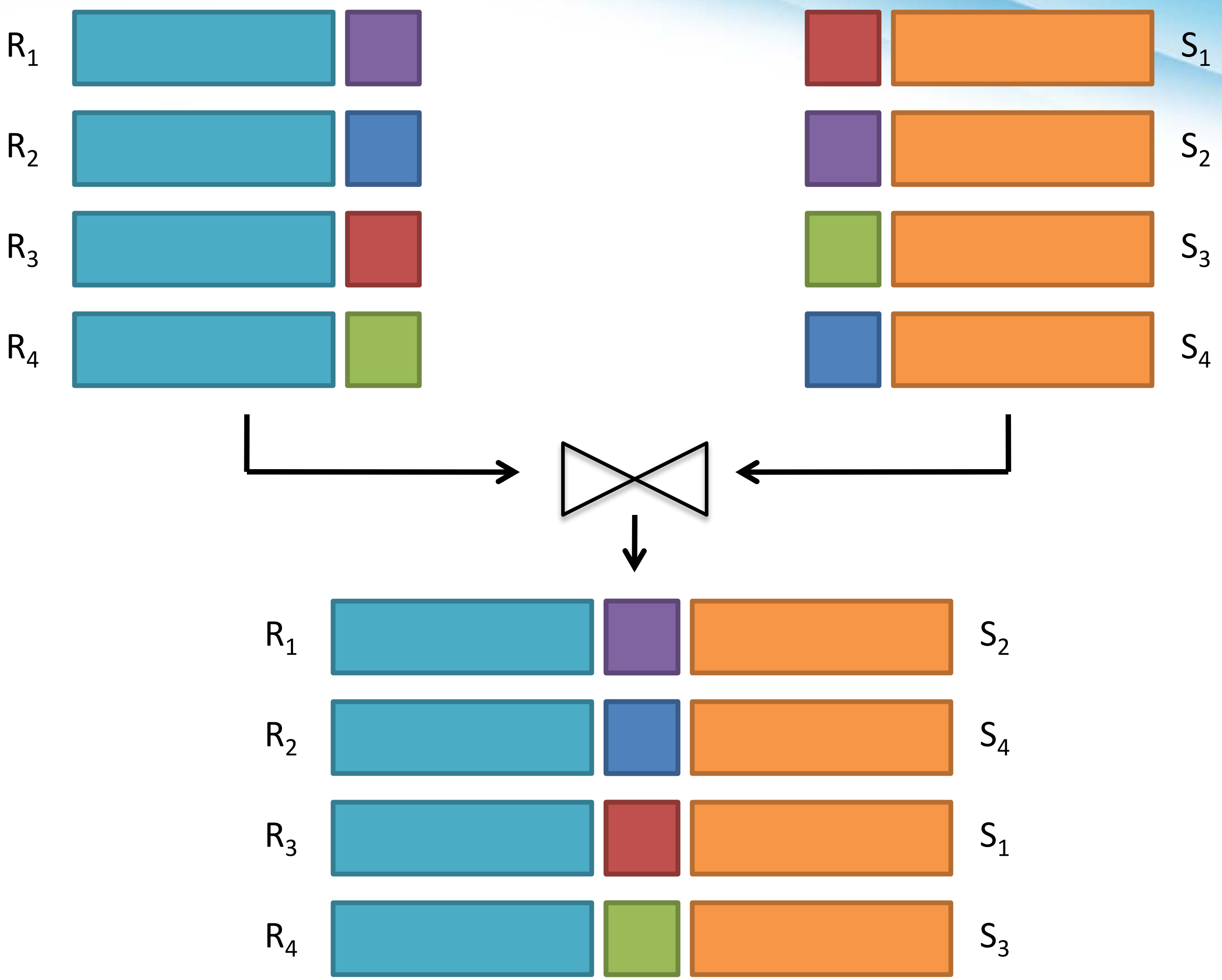




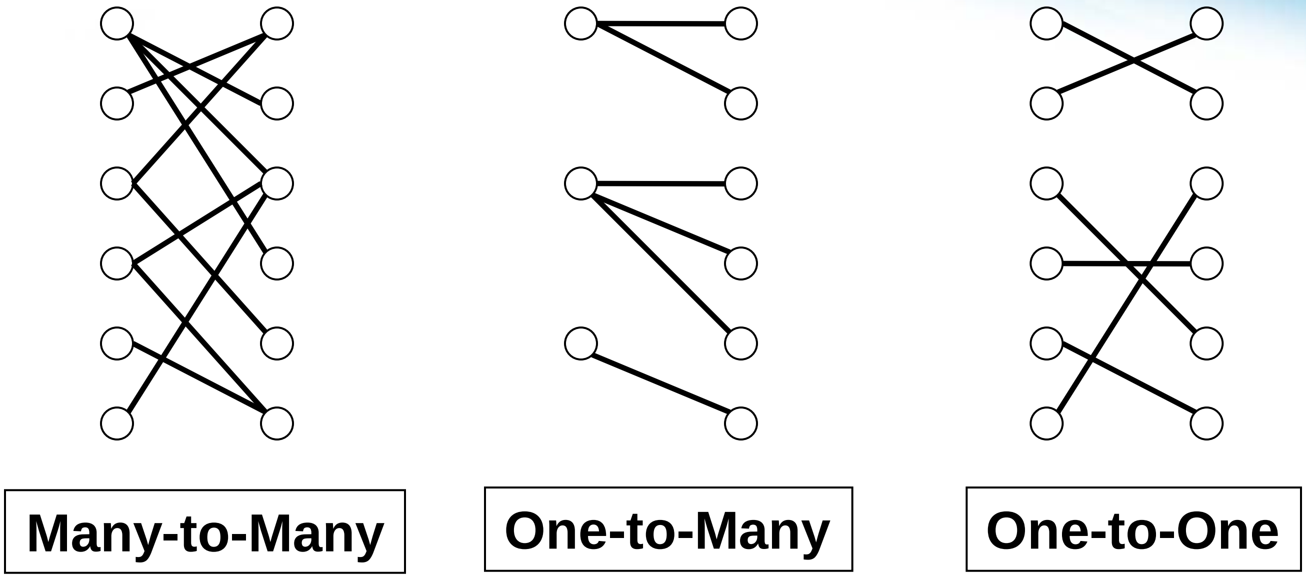
Group by… Aggregation

- 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
  - Optimize with combiners

Relational Joins



Types of Relationships



- Why it is hard in MapReduce?

Join Algorithms in MapReduce

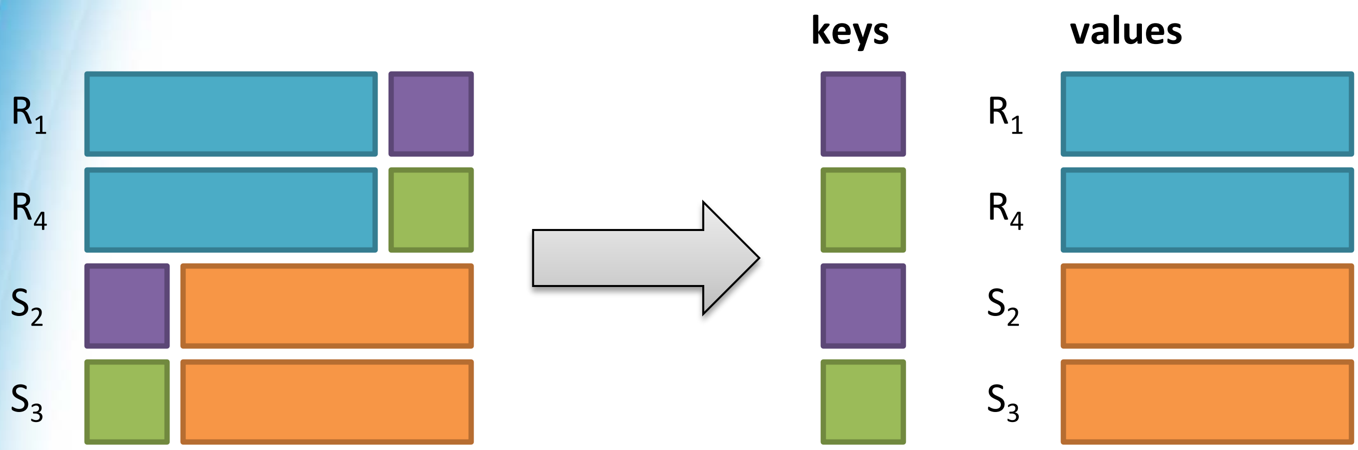
- Reduce-side join
- Map-side join
- In-memory join
  - Striped variant
  - Memcached variant

Reduce-side Join

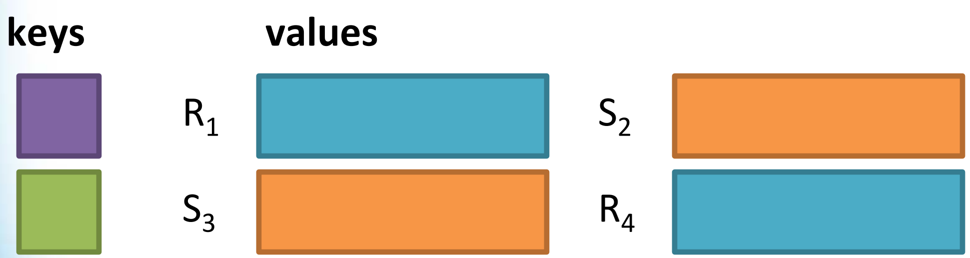
- Basic idea: group by join key
  - Map over both sets of tuples
  - Emit tuple as value with join key as the intermediate key
  - Execution framework brings together tuples sharing the same key
  - Perform actual join in reducer
  - Similar to a “sort-merge join” in database terminology
- Two variants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

Reduce-side Join: 1-to-1

Map



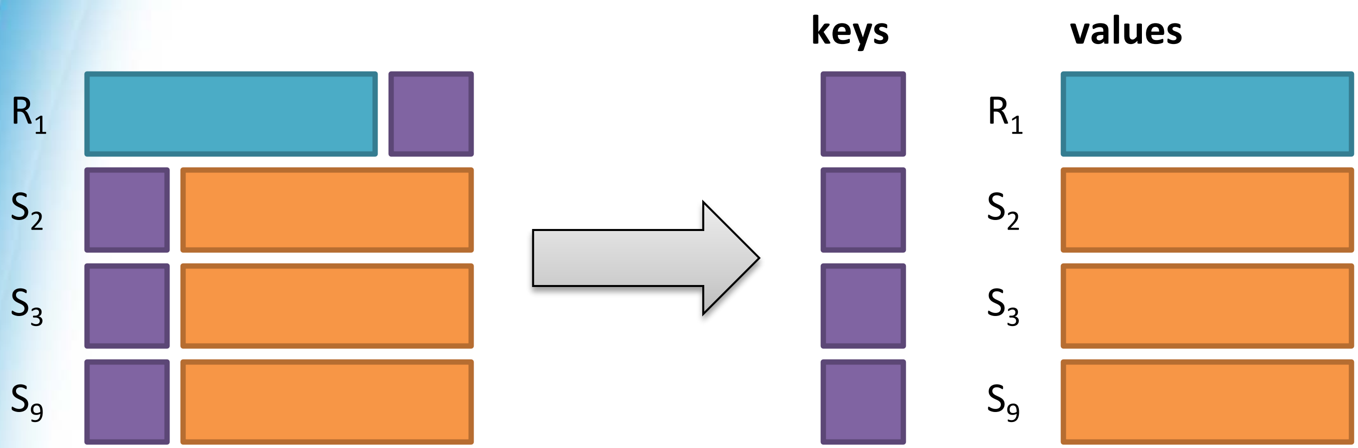
Reduce



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

Reduce-side Join: 1-to-many

Map

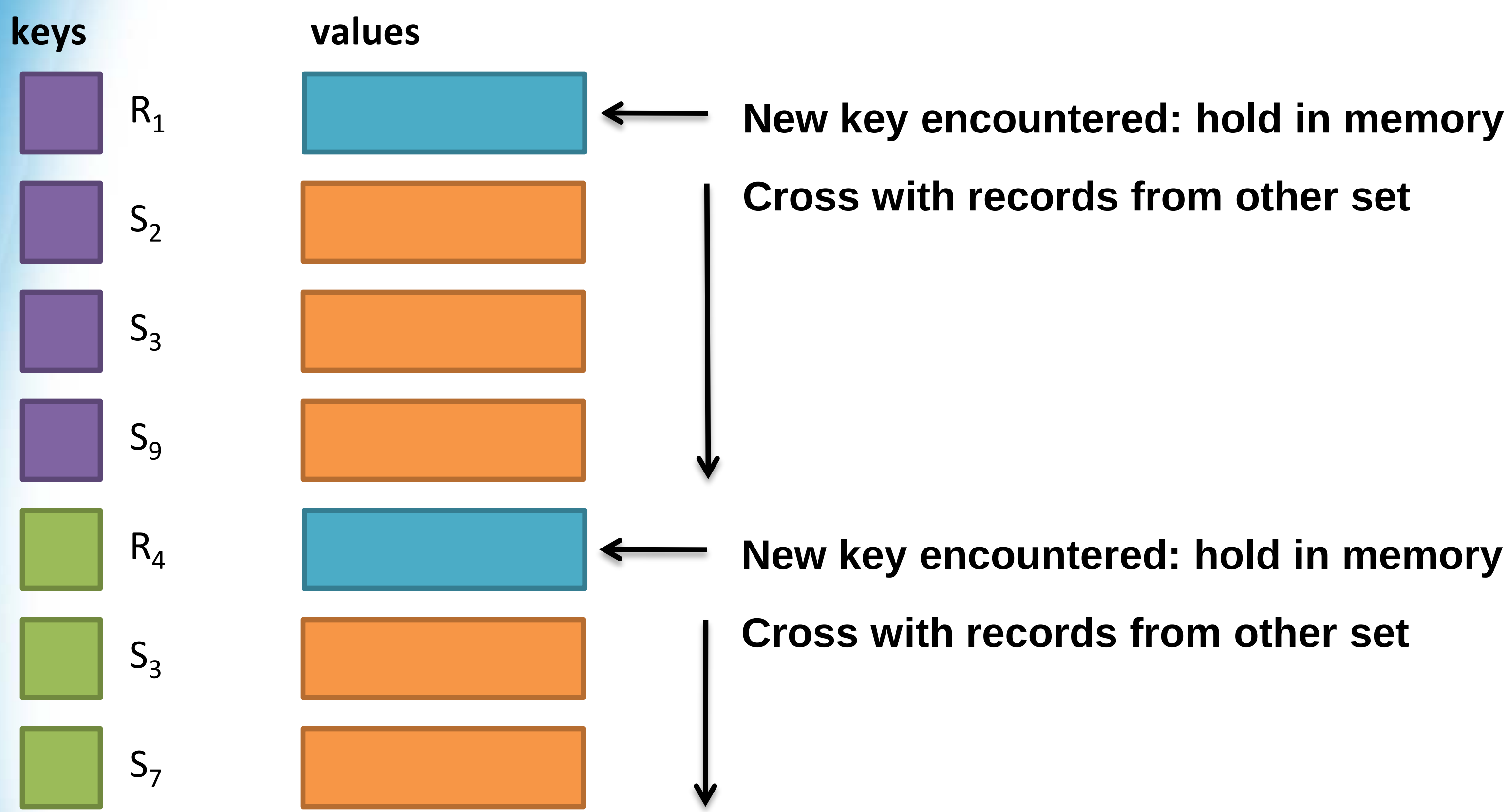


Reduce

What if R1 is not the first record?

Reduce-side Join: V-to-K Conversion

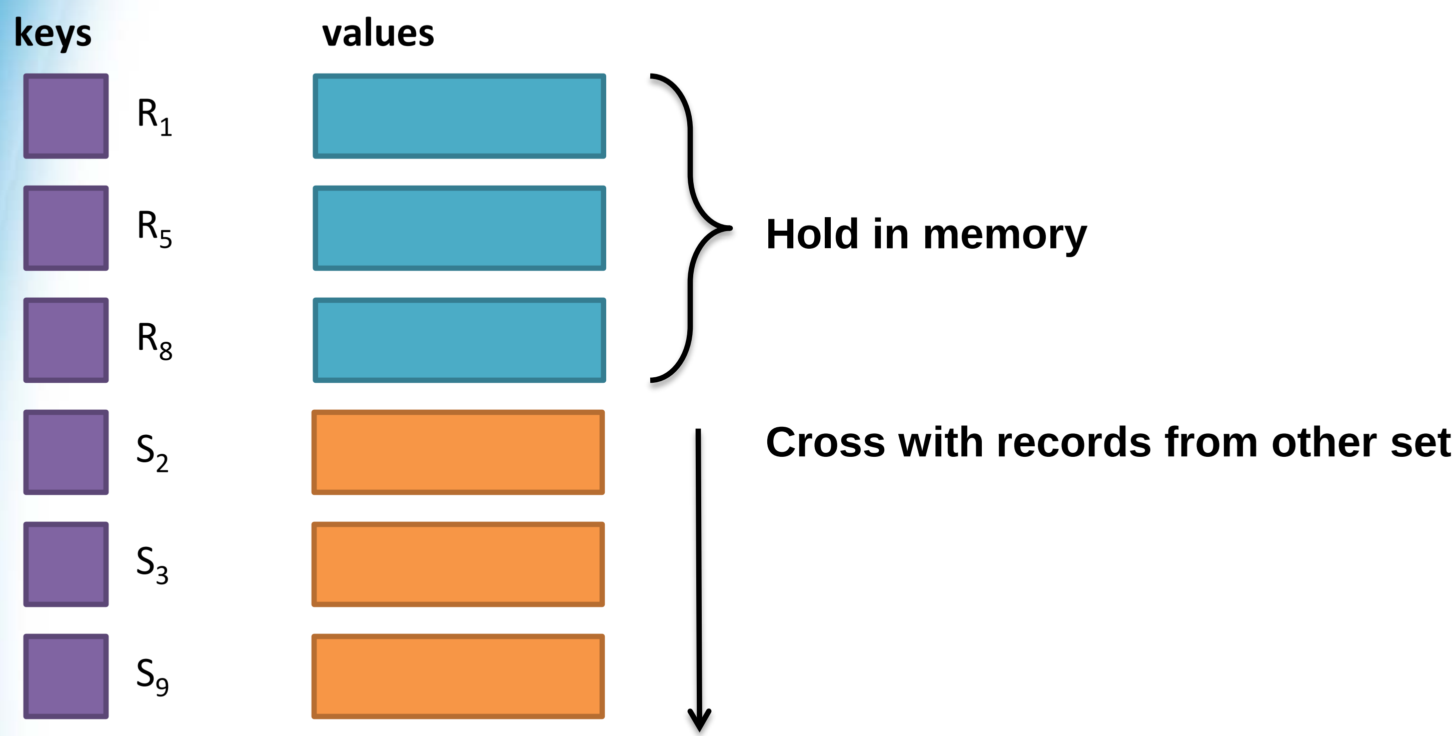
In reducer…



How to make R1 to be first record?

Reduce-side Join: many-to-many

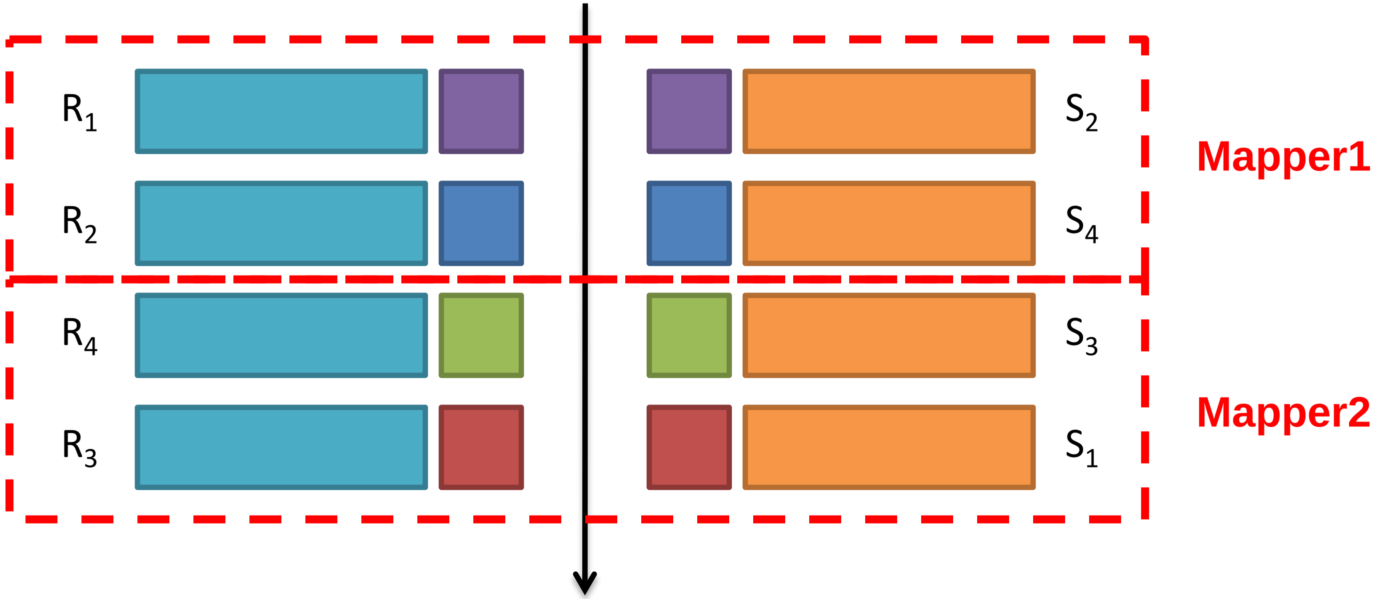
In reducer…



What are the two critical problems for reduce-side join?

Map-side Join: Parallel Scans

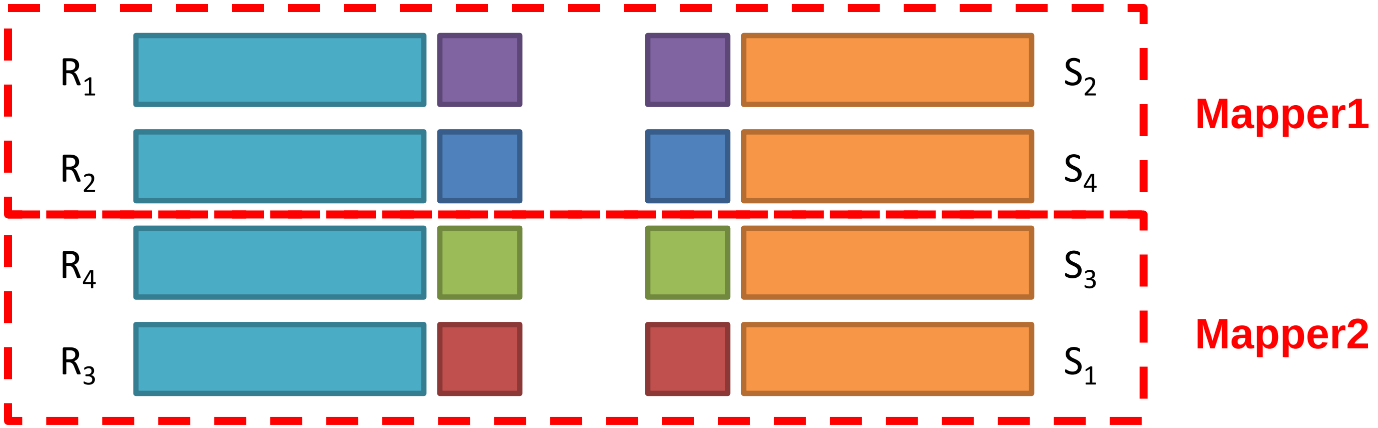
Assume two datasets are sorted by the join key:



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

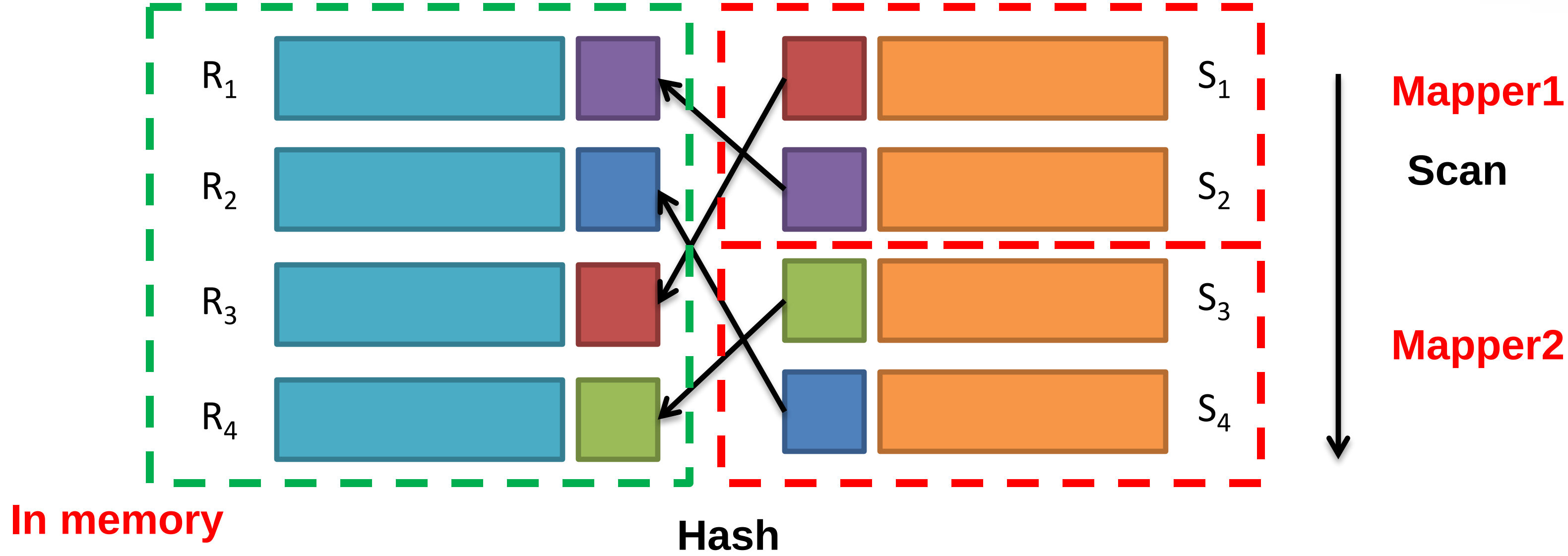
Map-side Join: Parallel Scans

- MapReduce Implementation
  - Map over one dataset, read from the other corresponding partition
  - No reducers necessary (unless to repartition or resort)
- Limitation:



Map-side Join: In-Memory Join

- Assume the smaller dataset can fit into memory



Store R in all mapper memory & build a hashmap with the join key, then sequential scan through S (called a "hash join" in database terminology)

Map-side Join: In-Memory Join

- MapReduce implementation
  - Distribute R to all nodes. **How?**
  - 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
- Limitation:

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
- Memcached join:
  - Memcached is a **distributed memory cache system,** such as REDIS
  - Load R into memcached
  - Replace in-memory hash lookup with memcached lookup

Which join to use?

- Performance?

- Limitations of each?
  - In-memory join:
  - Map-side join:
  - Reduce-side join:

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

Outline

- Role of relational databases in today's organizations
  - Where does MapReduce fit in?
- MapReduce algorithms for processing relational data
  - How do I perform a join, etc.?
- Evolving roles of relational databases and MapReduce
  - What's in store for the future?

Need for High-Level Languages

- Hadoop is great for large-data processing!
  - But writing Java programs for everything is verbose and slow
  - Analysts don't want to (or can't) write Java
- Solution: develop higher-level data processing languages
  - Hive: HQL is like SQL
  - Pig: Pig Latin is a bit like Perl

Hive and Pig

- Hive: data warehousing application in Hadoop
  - Query language is HQL, variant of SQL
  - Tables stored on HDFS as flat files
  - Developed by Facebook, now open source
- Pig: large-scale data processing system
  - Scripts are written in Pig Latin, a dataflow language
  - Developed by Yahoo!, now open source
  - Roughly 1/3 of all Yahoo! internal jobs
  - Similar to the role of SCALE for Spark

- Common idea:
  - Provide higher-level language to facilitate large-data processing
  - Higher-level language "compiles down" to Hadoop jobs

Hive: Example

- Hive looks similar to an SQL database
- Relational join on two tables:
  - Table of word counts from Shakespeare collection
  - Table of word counts from the bible

```
SELECT s.word, s.freq, k.freq FROM shakespeare s
  JOIN bible k ON (s.word = k.word) WHERE s.freq >= 1 AND k.freq >= 1
  ORDER BY s.freq DESC LIMIT 10;
```

the	25848	62394
I	23031	8854
and	19671	38985
to	18038	13526
of	16700	34654
a	14170	8057
you	12702	2720
my	11297	4135
in	10797	12445
is	8882	6884

Source: Material drawn from Cloudera training VM

Hive: Behind the Scenes

```
SELECT s.word, s.freq, k.freq FROM shakespeare s
  JOIN bible k ON (s.word = k.word) WHERE s.freq >= 1 AND k.freq >= 1
  ORDER BY s.freq DESC LIMIT 10;
```

(Abstract Syntax Tree)

```
(TOK_QUERY (TOK_FROM (TOK_JOIN (TOK_TABREF shakespeare s) (TOK_TABREF bible k) (= (.
(TOK_TABLE_OR_COL s) word) (. (TOK_TABLE_OR_COL k) word)))) (TOK_INSERT (TOK_DESTINATION
(TOK_DIR TOK_TMP_FILE)) (TOK_SELECT (TOK_SELEXPR (. (TOK_TABLE_OR_COL s) word)) (TOK_SELEXPR (.
(TOK_TABLE_OR_COL s) freq)) (TOK_SELEXPR (. (TOK_TABLE_OR_COL k) freq))) (TOK_WHERE (AND (>= (.
(TOK_TABLE_OR_COL s) freq) 1) (>= (. (TOK_TABLE_OR_COL k) freq) 1))) (TOK_ORDERBY
(TOK_TABSORTCOLNAMEDESC (. (TOK_TABLE_OR_COL s) freq))) (TOK_LIMIT 10)))
```

(one or more of MapReduce jobs)

Pig: Example

Task: Find the top 10 most visited pages in each category

Visits

User	Url	Time
Amy	cnn.com	8:00
Amy	bbc.com	10:00
Amy	flickr.com	10:05
Fred	cnn.com	12:00

⋮

Url Info

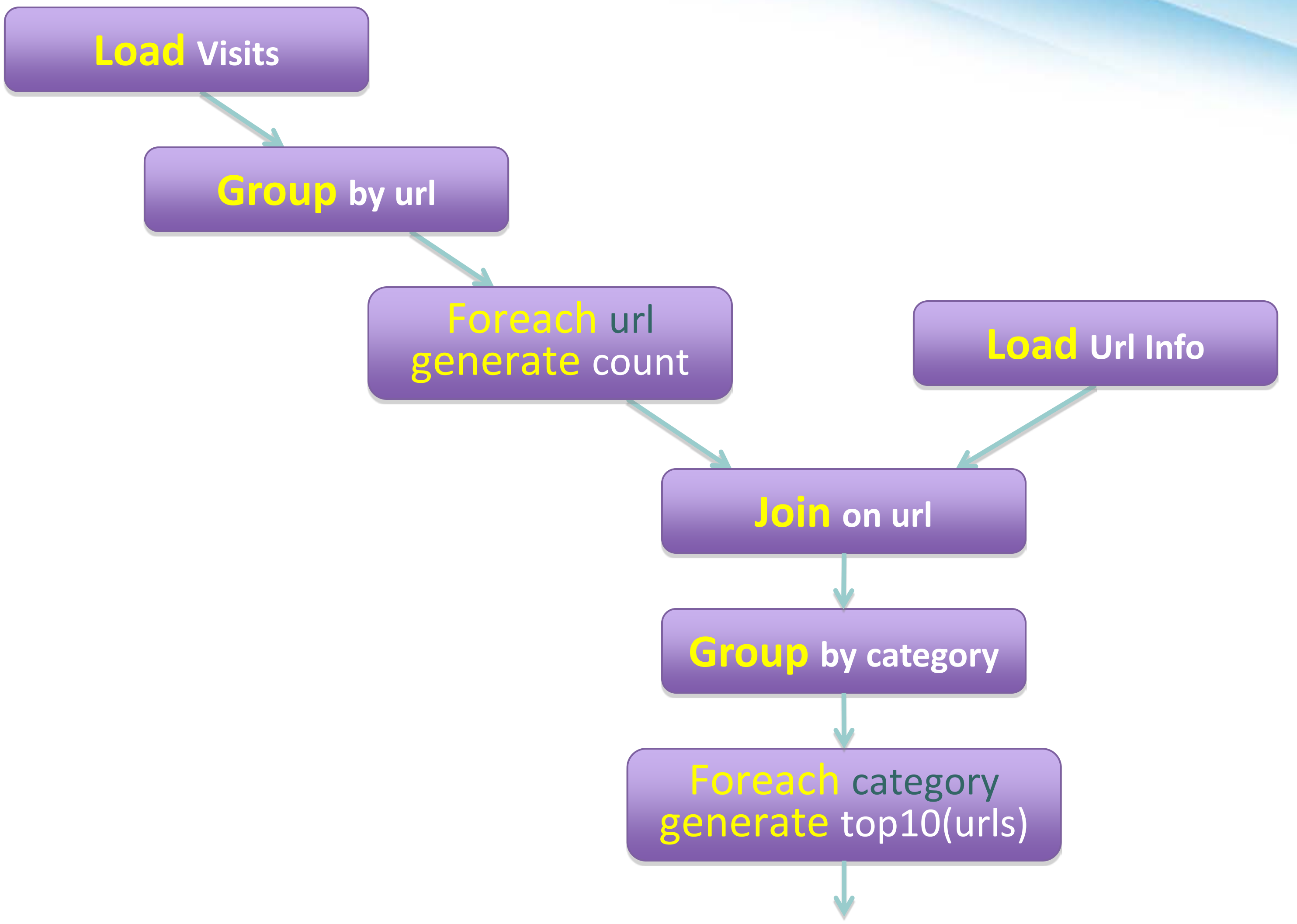
Url	Category	PageRank
cnn.com	News	0.9
bbc.com	News	0.8
flickr.com	Photos	0.7
espn.com	Sports	0.9

⋮

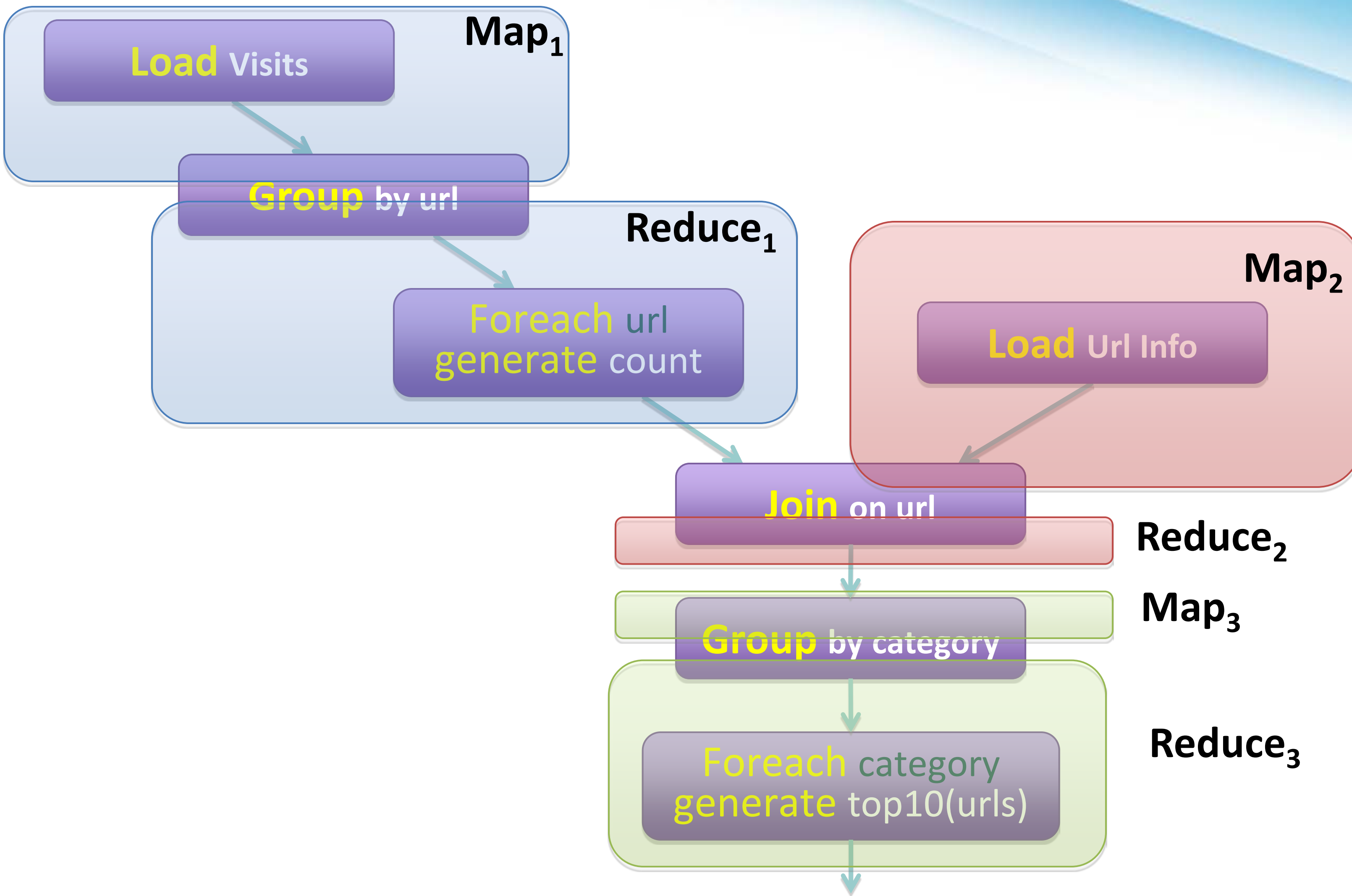
Pig Script

```
visits = load '/data/visits' as (user, url, time);
gVisits = group visits by url;
visitCounts = foreach gVisits generate url, count(visits);
urlInfo = load '/data/urlInfo' as (url, category, pRank);
visitCounts = join visitCounts by url, urlInfo by url;
gCategories = group visitCounts by category;
topUrls = foreach gCategories generate top(visitCounts,10);
store topUrls into '/data/topUrls';
```

Pig Query Plan



Pig Script in Hadoop



Pig Slides adapted from Olston et al. (SIGMOD 2008)

Dataflow Language vs SQL Language

- Dataflow language
  - Uses lazy evaluation
  - Uses extract, transform, load (ETL)
  - It is able to store data at any point during a pipeline
  - Directed acyclic graph (DAG) rather than a pipeline
  - Procedure programming instead of declarative programming
  - More control over execution

Reference

- Slides provided from Jimmy Lin @ University of Maryland
- A. Thusoo *et al*., "Hive - a petabyte scale data warehouse using Hadoop," *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, Long Beach, CA, 2010, pp. 996-1005.
- Apache Hive: https://hive.apache.org/
- Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. 2008. Pig latin: a not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data* (SIGMOD '08). ACM, New York, NY, USA, 1099-1110.
- Apache Pig: https://pig.apache.org/