Cloud Computing

Data Management in the Cloud

Dell Zhang
Birkbeck, University of London
2018/19

Data Management in Today’s Organisations

Big Data Analysis

- Peta-scale datasets are everywhere:
  - Facebook: 2.5PB of user data + 15TB/day (4/2009)
  - eBay: 6.5PB of user data + 50TB/day (5/2009)
  - …
- A lot of these datasets are (mostly) structured
  - Query logs
  - Point-of-sale records
  - User data (e.g., demographics)
  - …

Big Data Analysis

- How do we perform data analysis at scale?
  - Relational databases (RDBMS)
  - MapReduce (Hadoop)

RDBMS vs MapReduce

- Relational databases
  - Multipurpose
    - transactions & analysis
    - batch & 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

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

RDBMS vs MapReduce

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

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

Database Workloads

- Online Transaction Processing (OLTP)
  - 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)

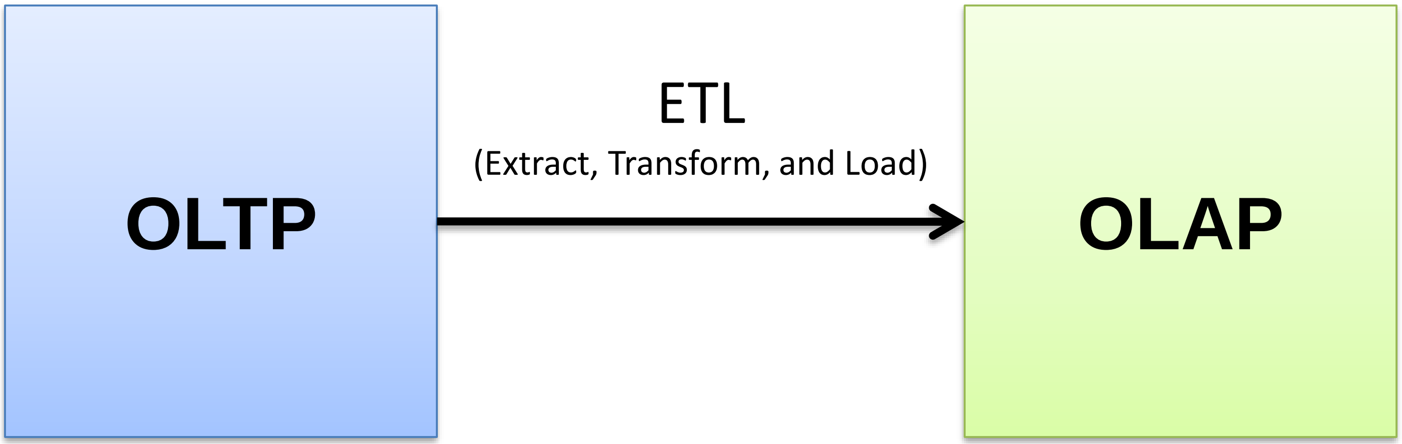
Database Workloads

- Online Analytical Processing (OLAP)
  - 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
  - OLTP database for user-facing transactions
  - OLAP database for data warehousing
- How do we connect the two?

OLTP/OLAP Architecture



OLTP/OLAP Integration

- Extract-Transform-Load (ETL)
  - Extract records from OLTP database
  - Transform records
    - clean data, check integrity, aggregate, etc.
  - Load records into OLAP database

OLTP/OLAP Integration

- OLTP database for user-facing transactions
  - Retain records of all activity
  - Periodic ETL (e.g., nightly)
- OLAP database for data warehousing
  - Business intelligence
    - reporting, ad hoc queries, data mining, etc.
  - Feedback to improve OLTP services

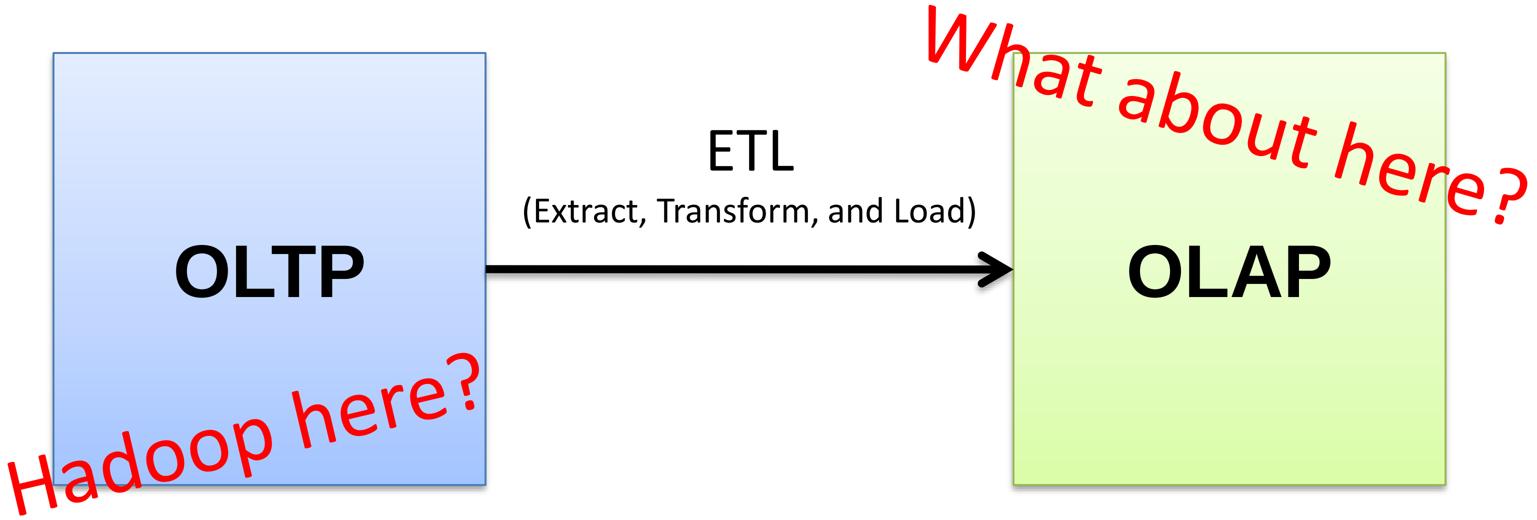
Business Intelligence

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

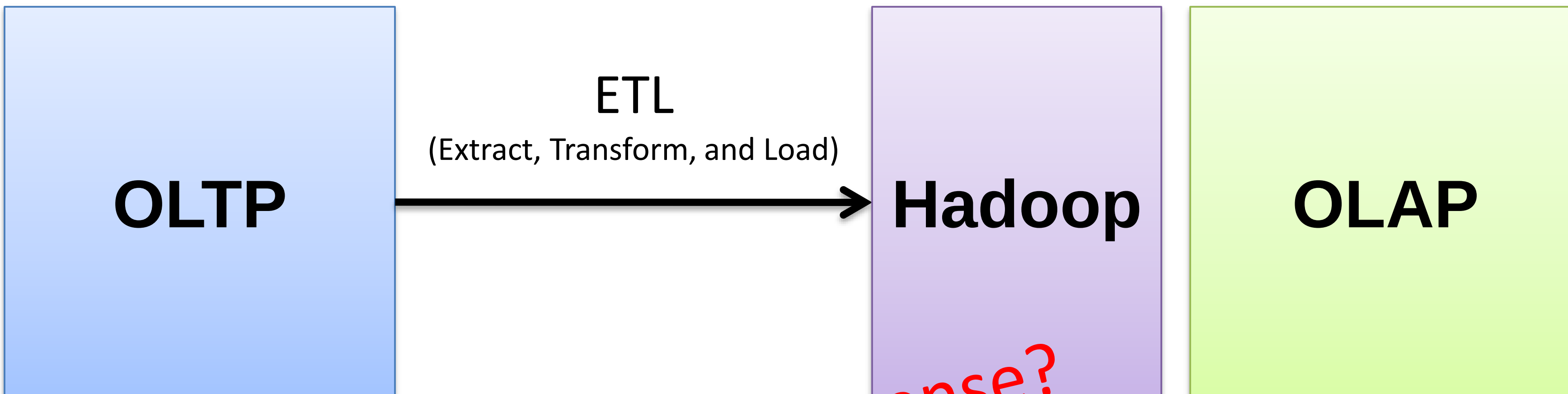
Business Intelligence

- 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?

ETL Bottleneck

- Hadoop is perfect:
  - Most likely, you already have some data warehousing solution
  - Ingestion is limited by the 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 and Matrix Data

Working Scenario

- Two tables:
  - User demographics (gender, age, income, etc.)
  - User page visits (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

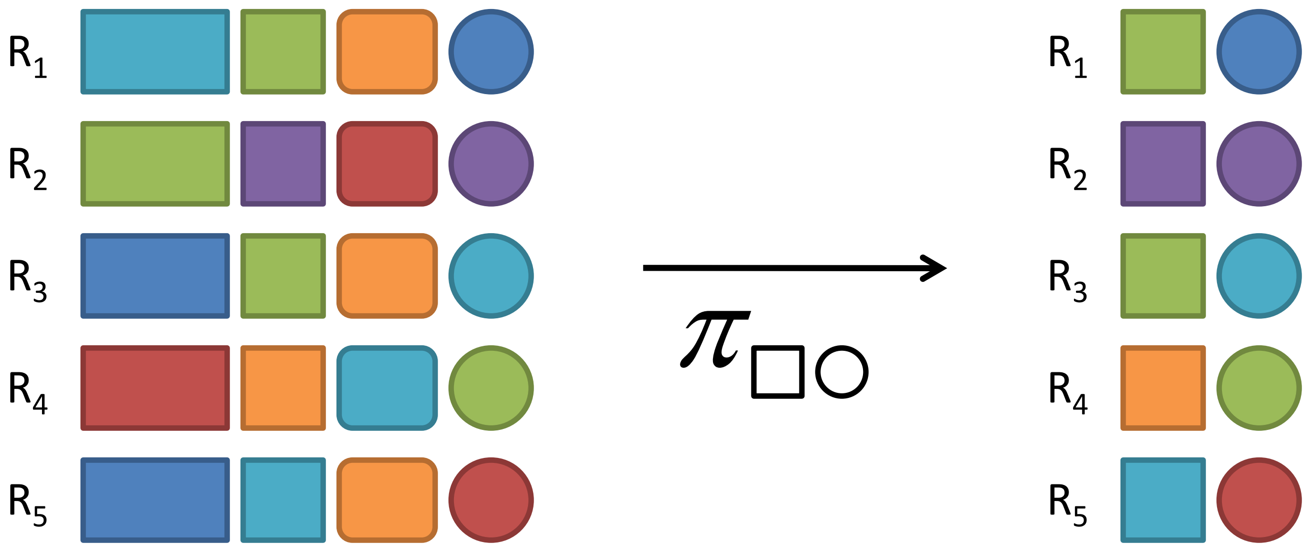
Relational Algebra

- Primitives
  - Projection (π)
  - Selection (σ)
  - Cartesian product ($\times$)
  - Set union ($\cup$)
  - Set difference ($-$)
  - Rename (ρ)
  - …

Relational Algebra

- Other operations
  - Join (⋈)
  - Group by… aggregation
  - …

Projection



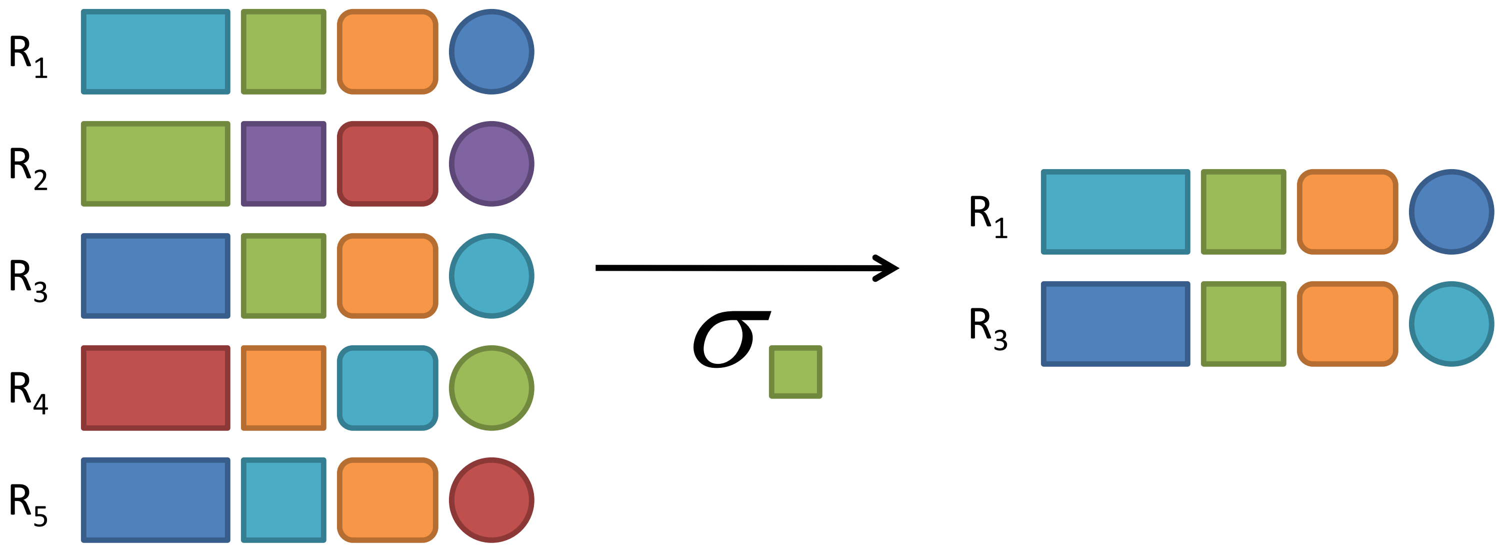
Projection in MapReduce

- 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

Projection in MapReduce

- Basically limited by HDFS streaming speeds
  - Speed of encoding/decoding tuples becomes important
  - Relational databases take advantage of compression
  - Semi-structured 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

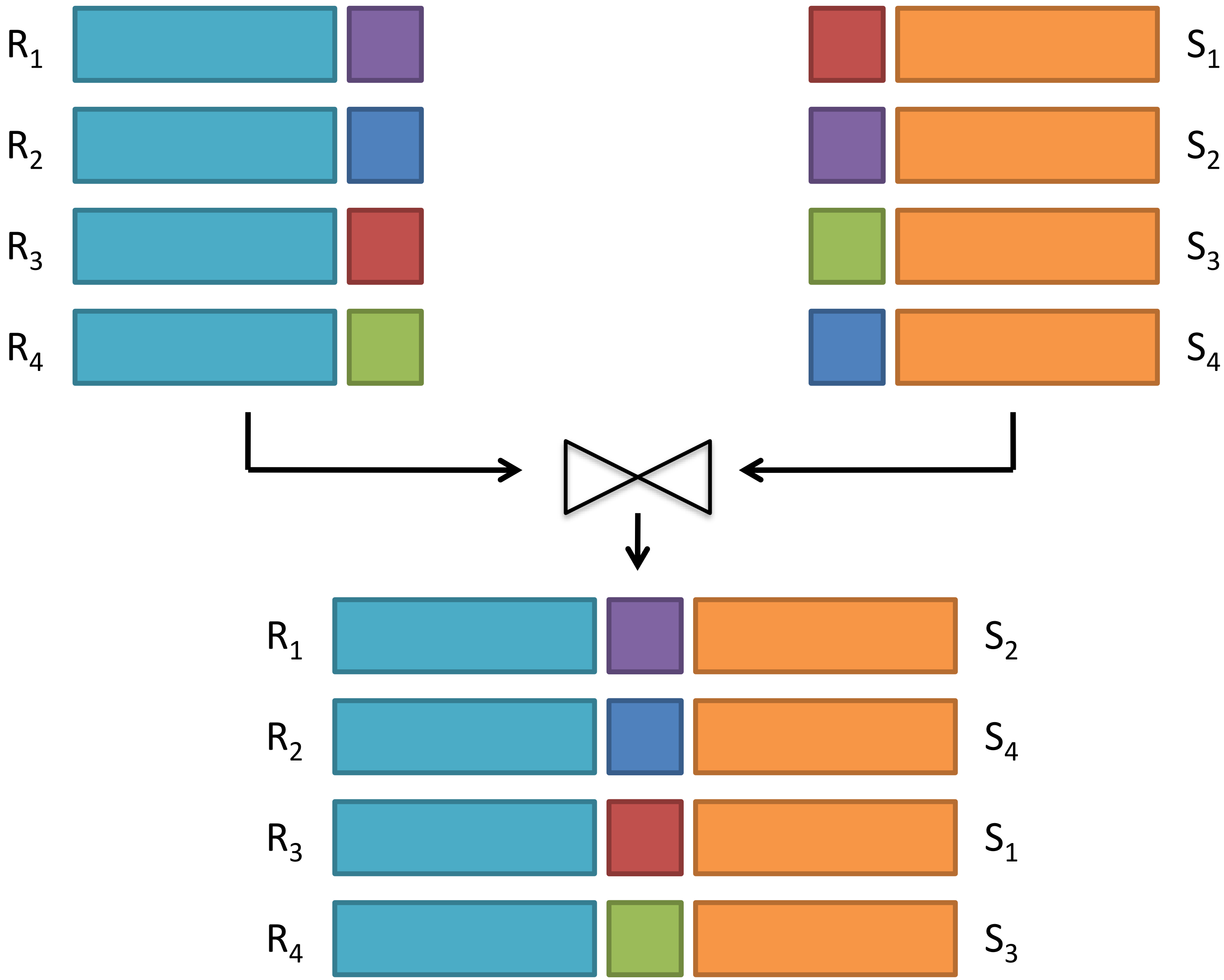
Selection in MapReduce

- Basically limited by HDFS streaming speeds
  - Speed of encoding/decoding tuples becomes important
  - Relational databases take advantage of compression
  - Semi-structured data? No problem!

Group by… Aggregation

- *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



Natural Join: Example

R

sid	bid	day
22	101	10/10/96
58	103	11/12/96

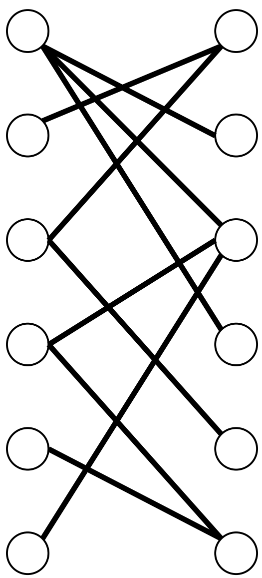
S

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

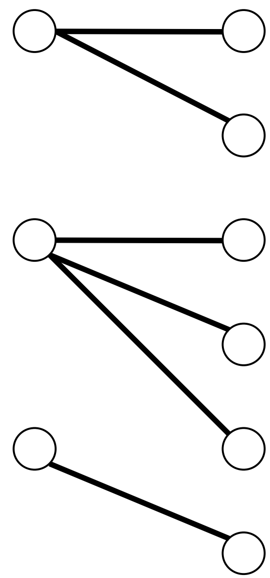
R $\bowtie$ S =

sid	sname	rating	age	bid	day
22	dustin	7	45.0	101	10/10/96
58	rusty	10	35.0	103	11/12/96

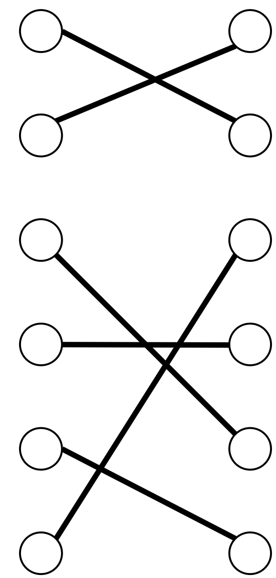
Types of Relationships



Many-to-Many



One-to-Many



One-to-One

Join Algorithms in MapReduce

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

Reduce-side Join

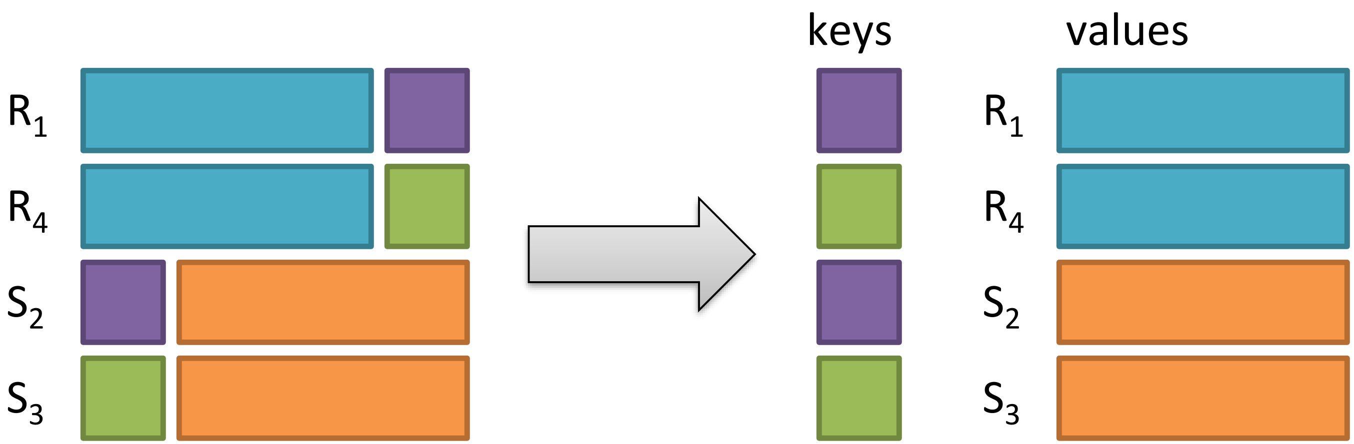
- Basic idea: group by join key
  - Map over both sets of tuples
  - Emit tuple as value with join key as the intermediate key
  - Execution framework brings together tuples sharing the same key
  - Perform actual join in reducer
  - Similar to a “sort-merge join” in database terminology

Reduce-side Join

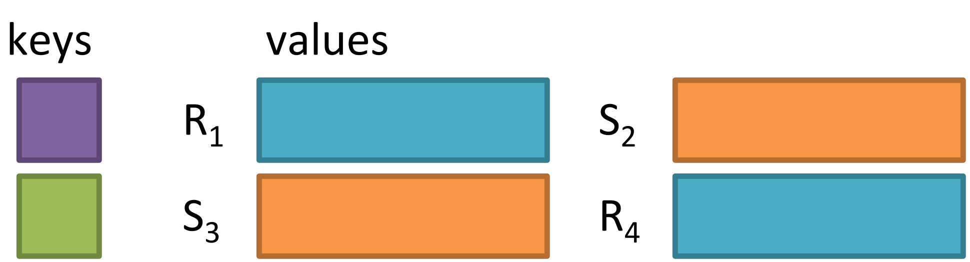
- Two variants
  - 1-to-1 joins
  - 1-to-many and many-to-many joins

Reduce-side Join: 1-to-1

Map



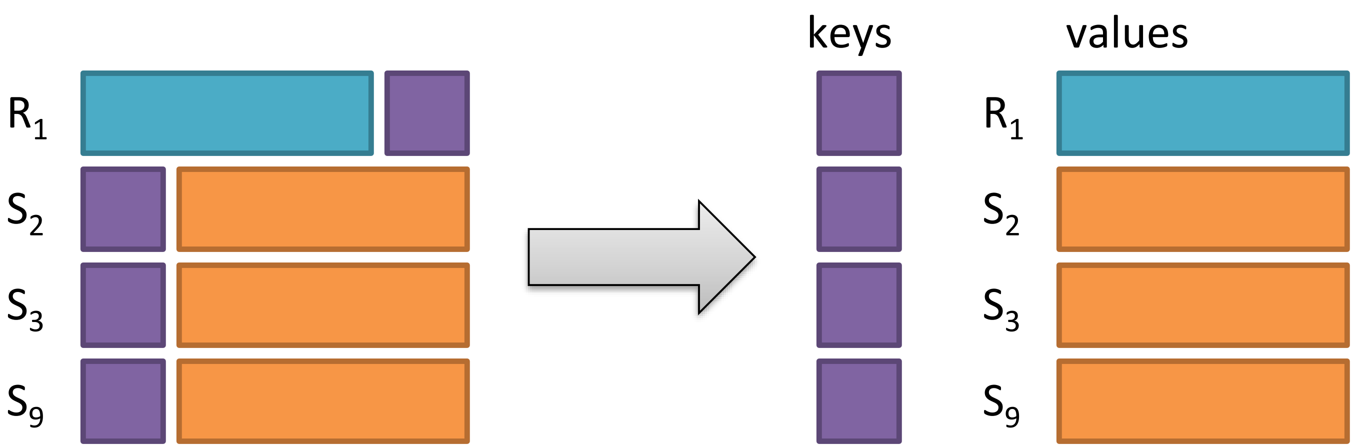
Reduce



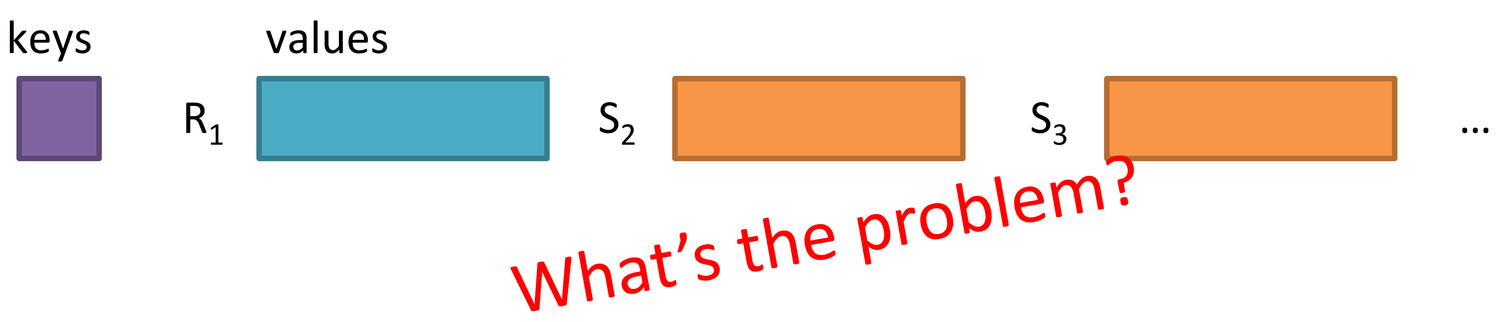
Note: no guarantee if R is going to come first or S

Reduce-side Join: 1-to-many

Map



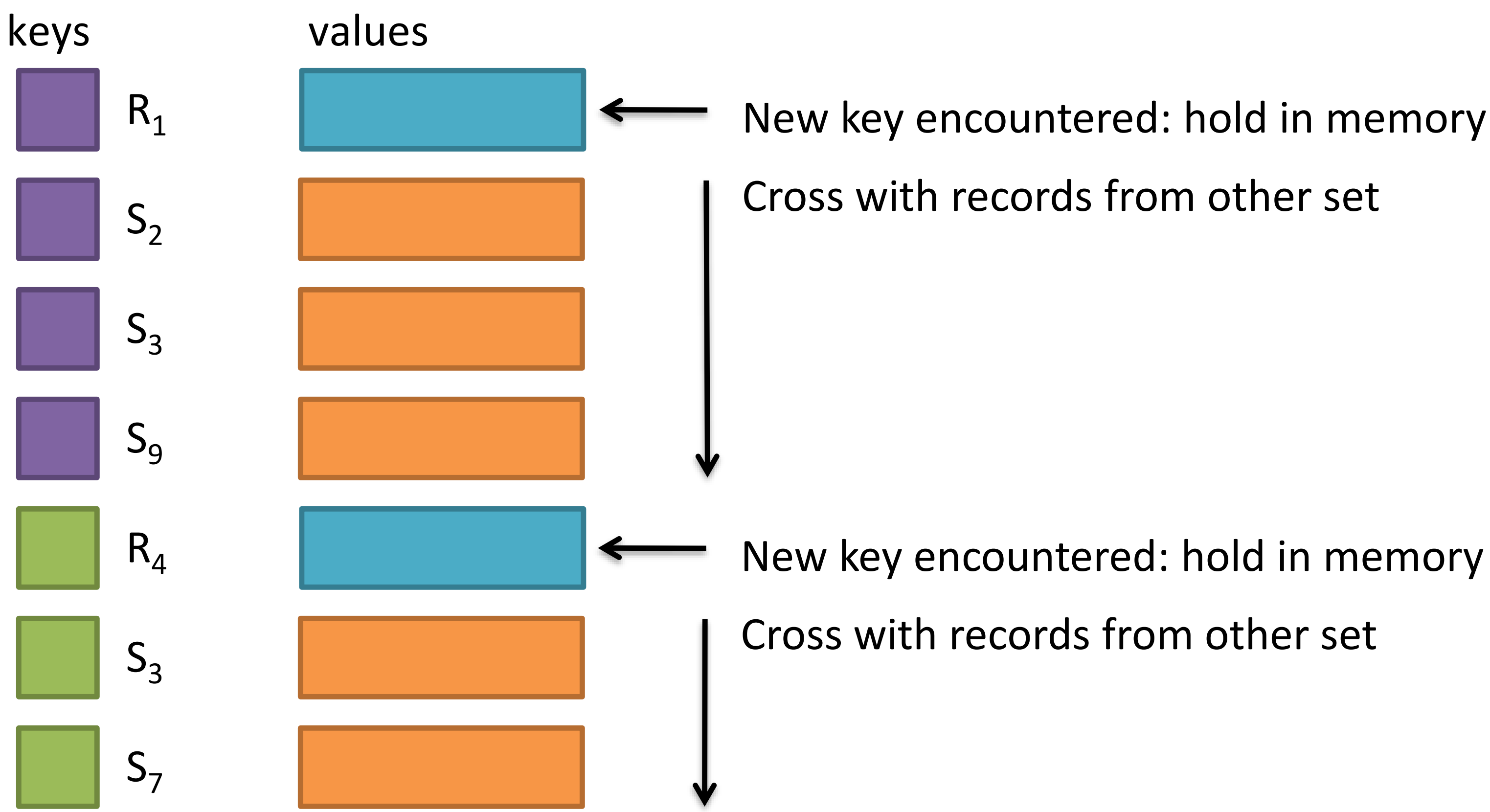
Reduce



Reduce-side Join: 1-to-many

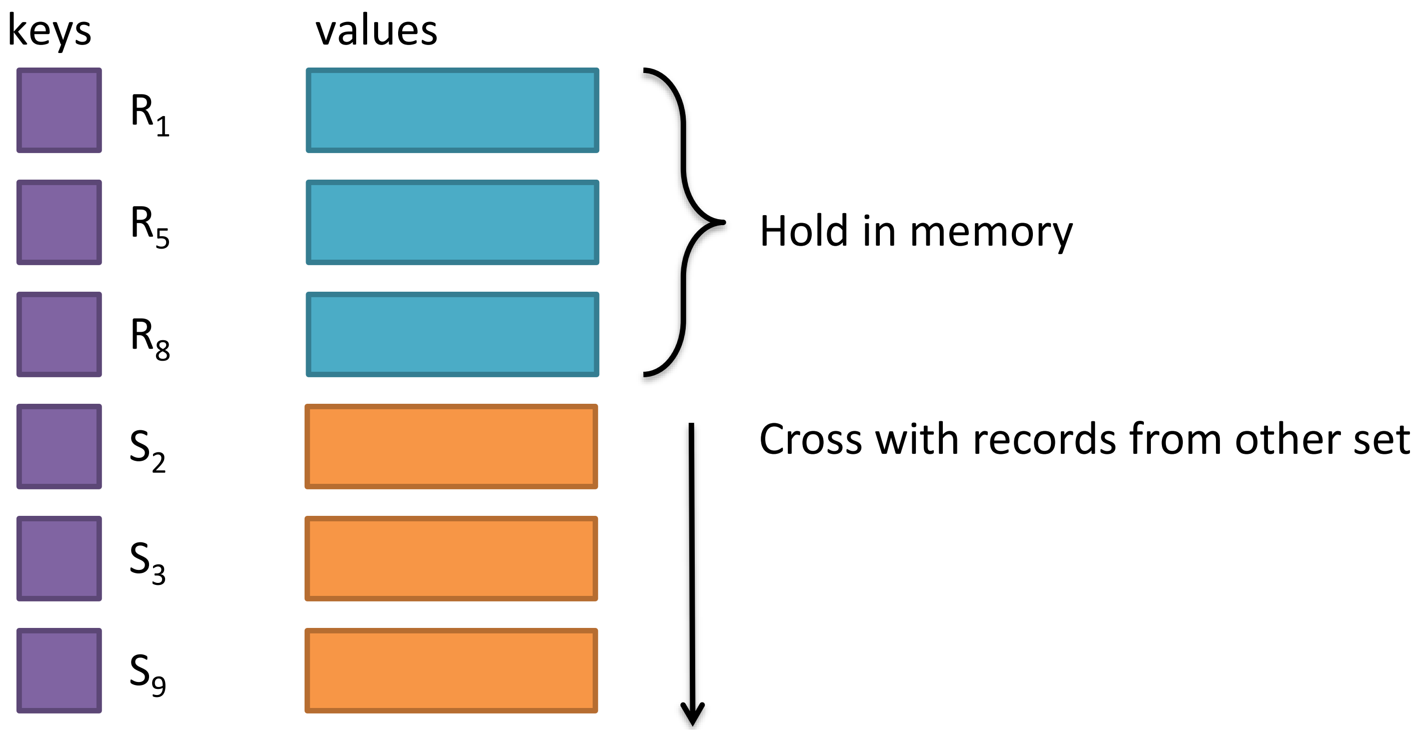
In reducer…

Value-to-Key Conversion



Reduce-side Join: many-to-many

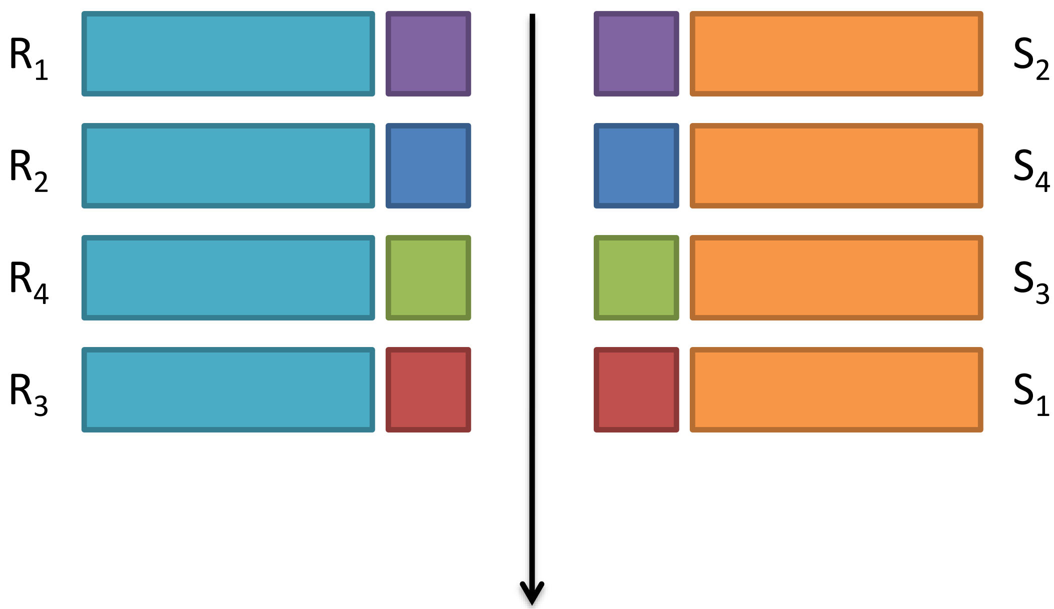
In reducer…



What's the problem?

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

Map-side Join: Parallel Scans

- 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?

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

In-Memory Join

- 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

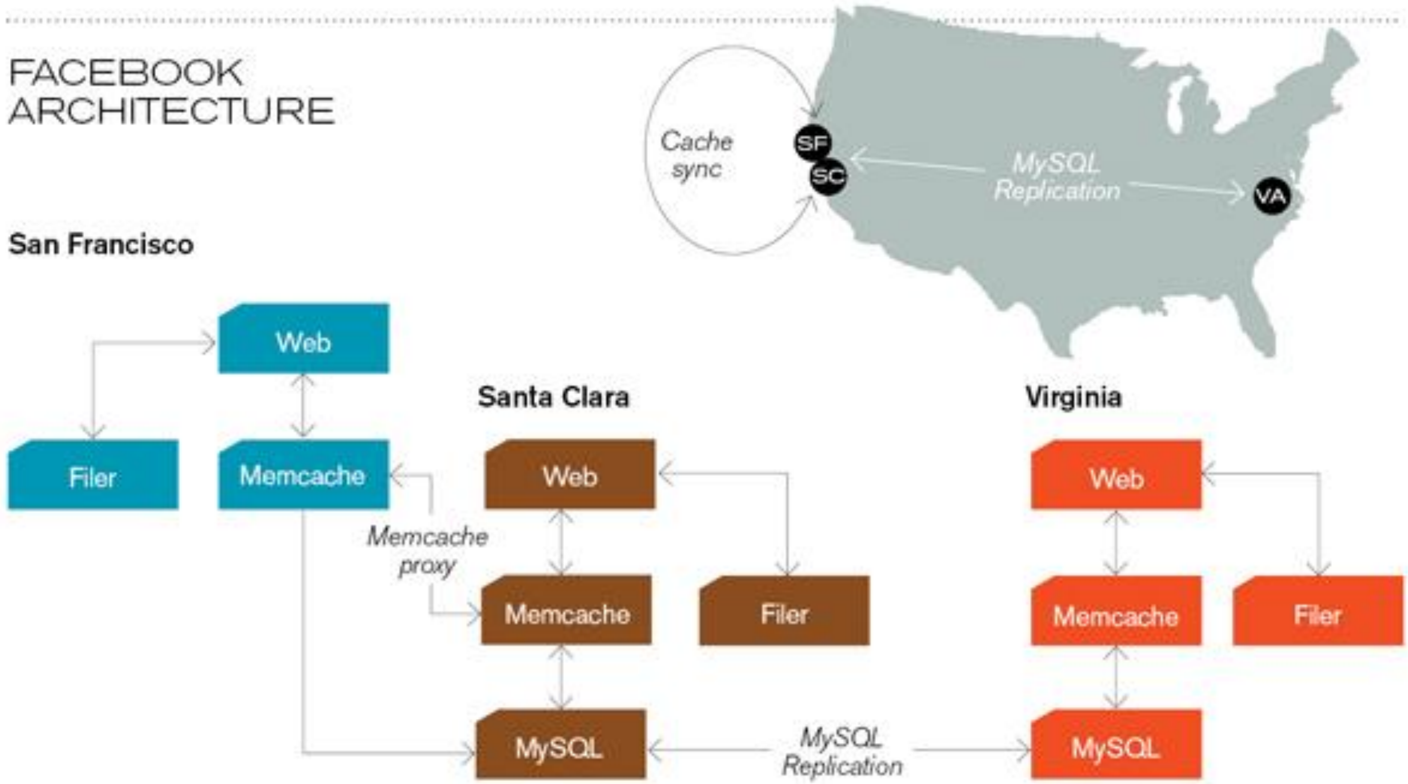
In-Memory Join: Variants

- 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

In-Memory Join: Variants

- Memcached join:
  - Load R into memcached
  - Replace in-memory hash lookup with memcached lookup

Memcached



Caching servers:
15 million requests per second, 95% handled by memcache (15 TB of RAM)

Database layer:
800 eight-core Linux servers running MySQL (40 TB user data)

Source: Technology Review (July/August, 2008)

Memcached Join

- Capacity and Scalability?
  - Memcached capacity >> RAM of individual node
  - Memcached scales out with cluster
- Latency?
  - Memcached is fast (basically, speed of network)
  - Batch requests to amortize latency costs

Source: See tech report by Lin et al. (2009)

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

Processing Relational Data

- Complex operations require multiple MapReduce jobs
  - Example: top 10 URLs in terms of average time spent
  - Opportunities for automatic optimisation

Matrix-Vector Multiplication

- Suppose we have an $n \times n$ matrix M, whose element in row i and column j is denoted m_{ij} .
- Suppose we also have a vector $\mathbf{v}$ of length n, whose jth element is v_j .
- Then the matrix-vector product is the vector $\mathbf{x}$ of length n, whose ith element x_i is given by

$$x_i = \sum_{j=1}^{n} m_{ij} v_j$$

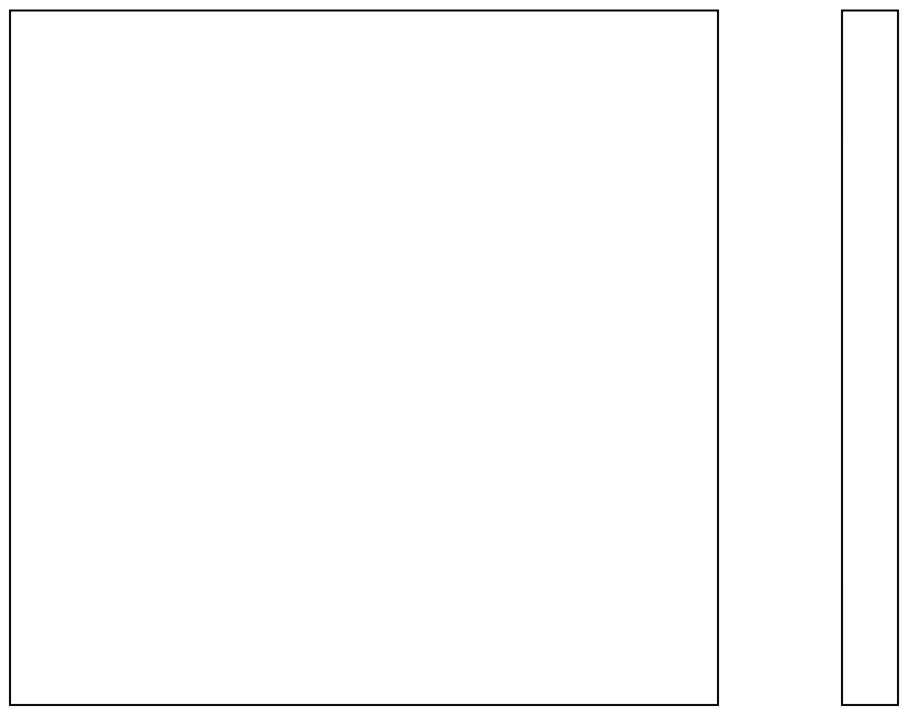
Matrix-Vector Multiplication

$$\underset{M}{\begin{bmatrix} 1 & 4 \\ 0 & 3 \end{bmatrix}} \mathbf{X} \underset{\mathbf{v}}{\begin{bmatrix} 3 \\ 1 \end{bmatrix}} = \begin{bmatrix} 1*3 + 4*1 \\ 0*3 + 3*1 \end{bmatrix}$$

$$= \begin{bmatrix} 7 \\ 3 \end{bmatrix}$$

Matrix-Vector Multiplication

- If $\mathbf{v}$ can fit in main memory



Matrix M Vector $\mathbf{v}$

Matrix-Vector Multiplication

- If $\mathbf{v}$ can fit in main memory:
  - Each Map task will operate on a chunk of the matrix M.
  - At the compute node executing a Map task, $\mathbf{v}$ is first read (in its entirety) into main memory, and subsequently it will be available to all applications of the Map function performed at this Map task.

Matrix-Vector Multiplication

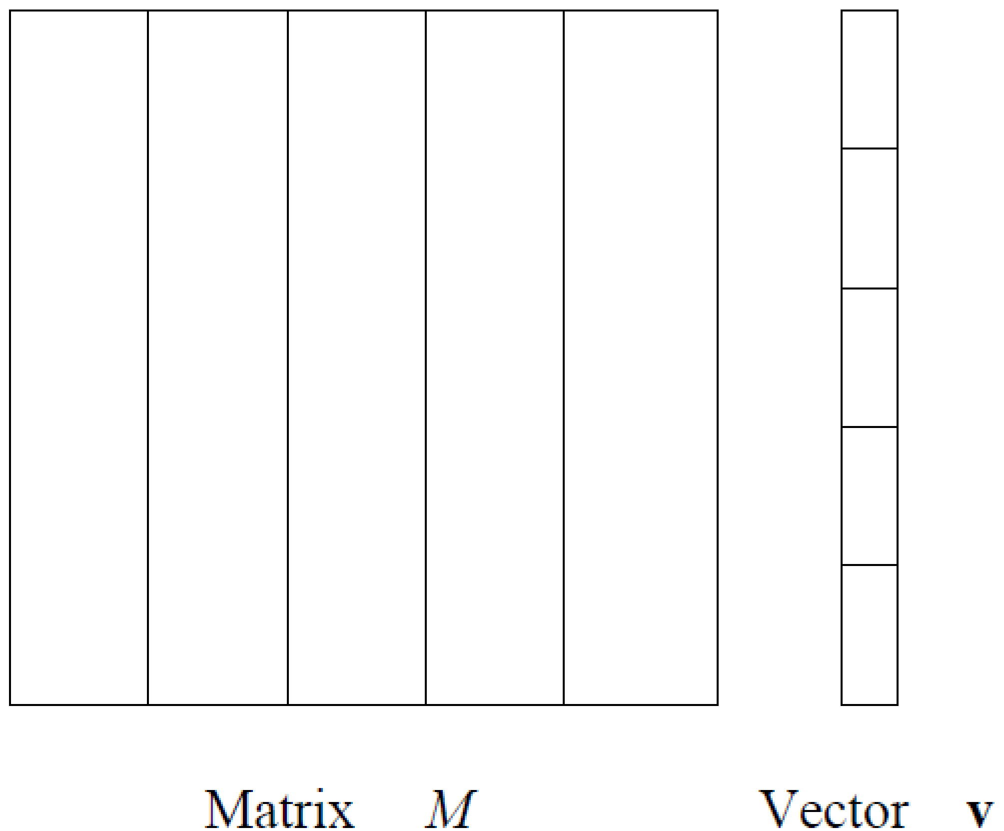
- If $\mathbf{v}$ can fit in main memory
  - **The Map Function**: For each matrix element m_{ij} it produces the key-value pair $(i, m_{ij}v_j)$.
  - **The Reduce Function**: It simply sums all the values associated with a given key i, thus the result will be a pair (i, x_i).

Matrix-Vector Multiplication

- If $\mathbf{v}$ cannot fit in main memory
  - To avoid excessive disk access, we can divide the matrix into *vertical stripes* of equal width and divide the vector into an equal number of *horizontal stripes*, of the same height, so that the portion of the vector in one stripe can fit into main memory at a compute node.

Matrix-Vector Multiplication

- If **v** cannot fit in main memory



Matrix-Vector Multiplication

- If $\mathbf{v}$ cannot fit in main memory
  - The ith stripe of the matrix multiplies only components from the ith stripe of the vector.
  - Thus, we can divide the matrix into one file for each stripe, and do the same for the vector.
  - Each Map task is assigned a chunk from one of the stripes of the matrix and gets the entire corresponding stripe of the vector.

Matrix Multiplication

- If M is a matrix with element m_{ij} in row i and column j,
  and N is a matrix with element n_{jk} in row j and column k,
  then the product $P = MN$ is the matrix P with element p_{ik} in row i and column k, where

$$p_{ik} = \sum_j m_{ij} n_{jk}$$

Matrix Multiplication

$$\underset{M}{\begin{bmatrix} 1 & 4 \\ 0 & 3 \end{bmatrix}} \times \underset{N}{\begin{bmatrix} 0 & 3 & 0 \\ 2 & 1 & 4 \end{bmatrix}} = \begin{bmatrix} 1*0 + 4*2 & 1*3 + 4*1 & 1*0 + 4*4 \\ 0*0 + 3*2 & 0*3 + 3*1 & 0*0 + 3*4 \end{bmatrix}$$

$$= \begin{bmatrix} 8 & 7 & 16 \\ 6 & 3 & 12 \end{bmatrix}$$

Matrix Multiplication

- A matrix = a relation with three attributes: the row number, the column number, and the value at that row and column.
  - M ➔ relation $M(I, J, \mathrm{V})$, with tuples (i, j, m_{ij})
  - N ➔ relation $N(J, K, W)$, with tuples (j, k, n_{jk})
- The product MN is almost a natural join (on attribute J) followed by grouping and aggregation.

Matrix Multiplication

- With two MapReduce steps (1/2)
  - **The Map Function**: For each matrix element m_{ij} , produce the key-value pair (j, (M, i, m_{ij})). Likewise, for each matrix element n_{jk}, produce the key-value pair (j, (N, k, n_{jk})) .
  - **The Reduce Function**: For each key j, examine its list of associated values. For each value from M, say (M, i, m_{ij}), and each value from N, say (N, k, n_{jk}), produce a key-value pair ((i, k), $m_{ij}n_{jk}$).

Matrix Multiplication

- With two MapReduce steps (2/2)
  - **The Map Function**: This function is just the identity.
  - **The Reduce Function**: For each key (i, k), produce the sum of the list of values associated with this key. The result is a pair $((i, k), v)$, where v is the value of the element in row i and column k of the matrix $P = MN$.

Matrix Multiplication

- With one MapReduce step
  - **The Map Function**: For each element m_{ij} of M, produce all the key-value pairs $((i, k), (M, j, m_{ij}))$ for $k = 1, 2, \ldots,$ up to the number of columns of N. Similarly, for each element n_{jk} of N, produce all the key-value pairs $((i, k), (N, j, n_{jk}))$ for $i = 1, 2, \ldots,$ up to the number of rows of M.

Matrix Multiplication

- With one MapReduce step
  - **The Reduce Function**: Each key (i, k) has an associated list with all the values (M, j, m_{ij}) and (N, j, n_{jk}), for all possible values of j. To connect the two values on the list that have the same value of j for each j, we can sort by j the values beginning with M and the values beginning with N, in separate lists. The jth values on each list must have their third components m_{ij} and n_{jk} extracted and multiplied. Then, these products are summed and the paired with (i, k) in the output.

Evolving Roles for Relational Database and MapReduce

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

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

Hive

- 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

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

Hive: Example

```
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)

```
STAGE DEPENDENCIES:
  Stage-1 is a root stage
  Stage-2 depends on stages: Stage-1
  Stage-0 is a root stage

STAGE PLANS:
  Stage: Stage-1
    Map Reduce
      Alias -> Map Operator Tree:
        s
          TableScan
            alias: s
            Filter Operator
              predicate:
                  expr: (freq >= 1)
                  type: boolean
              Reduce Output Operator
                key expressions:
                      expr: word
                      type: string
                sort order: +
                Map-reduce partition columns:
                      expr: word
                      type: string
                tag: 0
                value expressions:
                      expr: freq
                      type: int
                      expr: word
                      type: string
        k
          TableScan
            alias: k
            Filter Operator
              predicate:
                  expr: (freq >= 1)
                  type: boolean
              Reduce Output Operator
                key expressions:
                      expr: word
                      type: string
                sort order: +
                Map-reduce partition columns:
                      expr: word
                      type: string
                tag: 1
                value expressions:
                      expr: freq
                      type: int
      Reduce Operator Tree:
        Join Operator
          condition map:
               Inner Join 0 to 1
          condition expressions:
            0 {VALUE._col0} {VALUE._col1}
            1 {VALUE._col0}
          outputColumnNames: _col0, _col1, _col2
          Filter Operator
            predicate:
                expr: ((_col0 >= 1) and (_col2 >= 1))
                type: boolean
            Select Operator
              expressions:
                    expr: _col1
                    type: string
                    expr: _col0
                    type: int
                    expr: _col2
                    type: int
              outputColumnNames: _col0, _col1, _col2
              File Output Operator
                compressed: false
                GlobalTableId: 0
                table:
                    input format: org.apache.hadoop.mapred.SequenceFileInputFormat
                    output format: org.apache.hadoop.hive.ql.io.HiveSequenceFileOutputFormat

  Stage: Stage-2
    Map Reduce
      Alias -> Map Operator Tree:
        hdfs://localhost:8022/tmp/hive-training/364214370/10002
            Reduce Output Operator
              key expressions:
                    expr: _col1
                    type: int
              sort order: -
              tag: -1
              value expressions:
                    expr: _col0
                    type: string
                    expr: _col1
                    type: int
                    expr: _col2
                    type: int
      Reduce Operator Tree:
        Extract
          Limit
            File Output Operator
              compressed: false
              GlobalTableId: 0
              table:
                  input format: org.apache.hadoop.mapred.TextInputFormat
                  output format:
org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat

  Stage: Stage-0
    Fetch Operator
      limit: 10
```

Pig

- 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

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

⋮

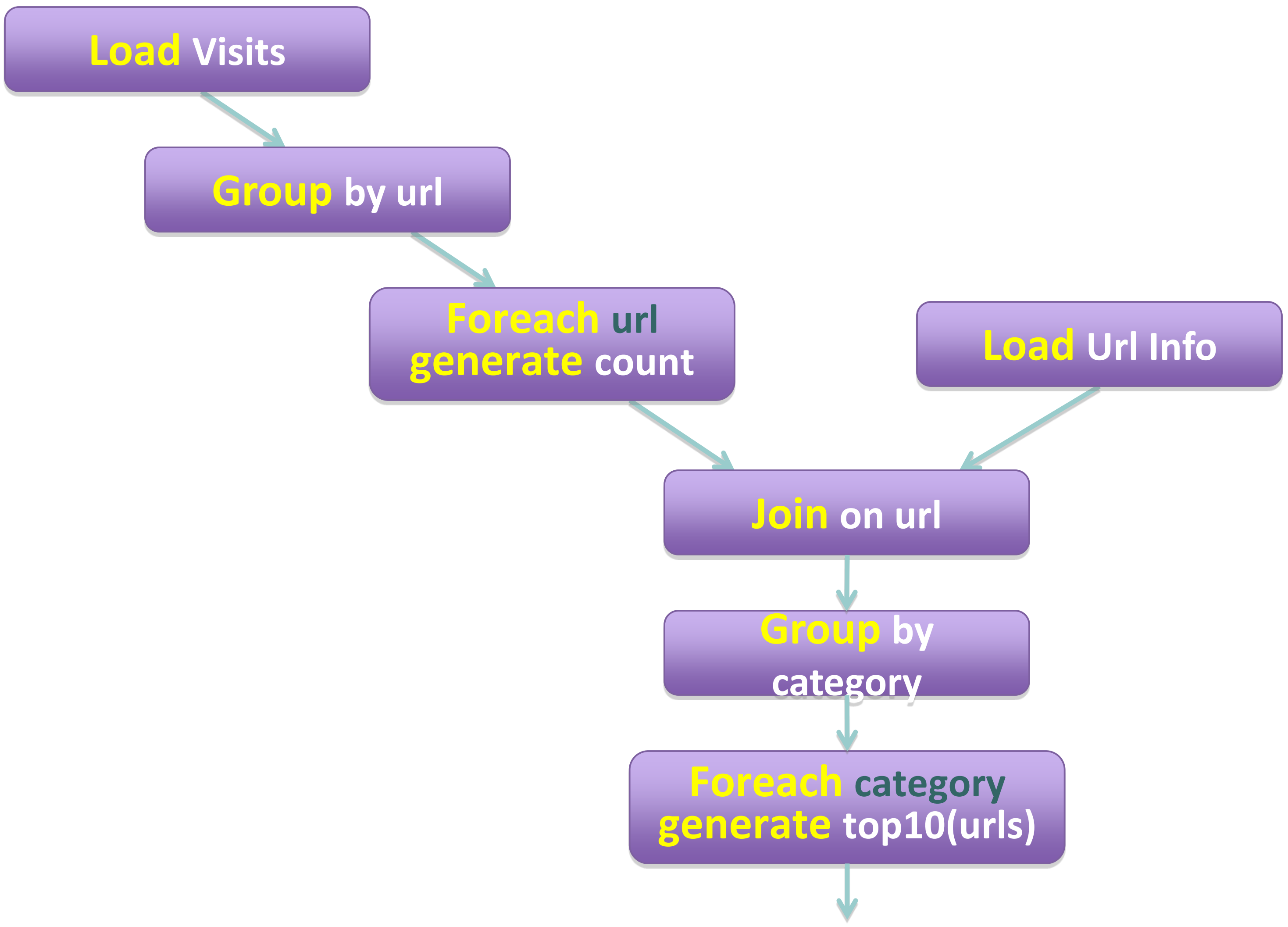
Url Info

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

⋮

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

Pig Query Plan



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

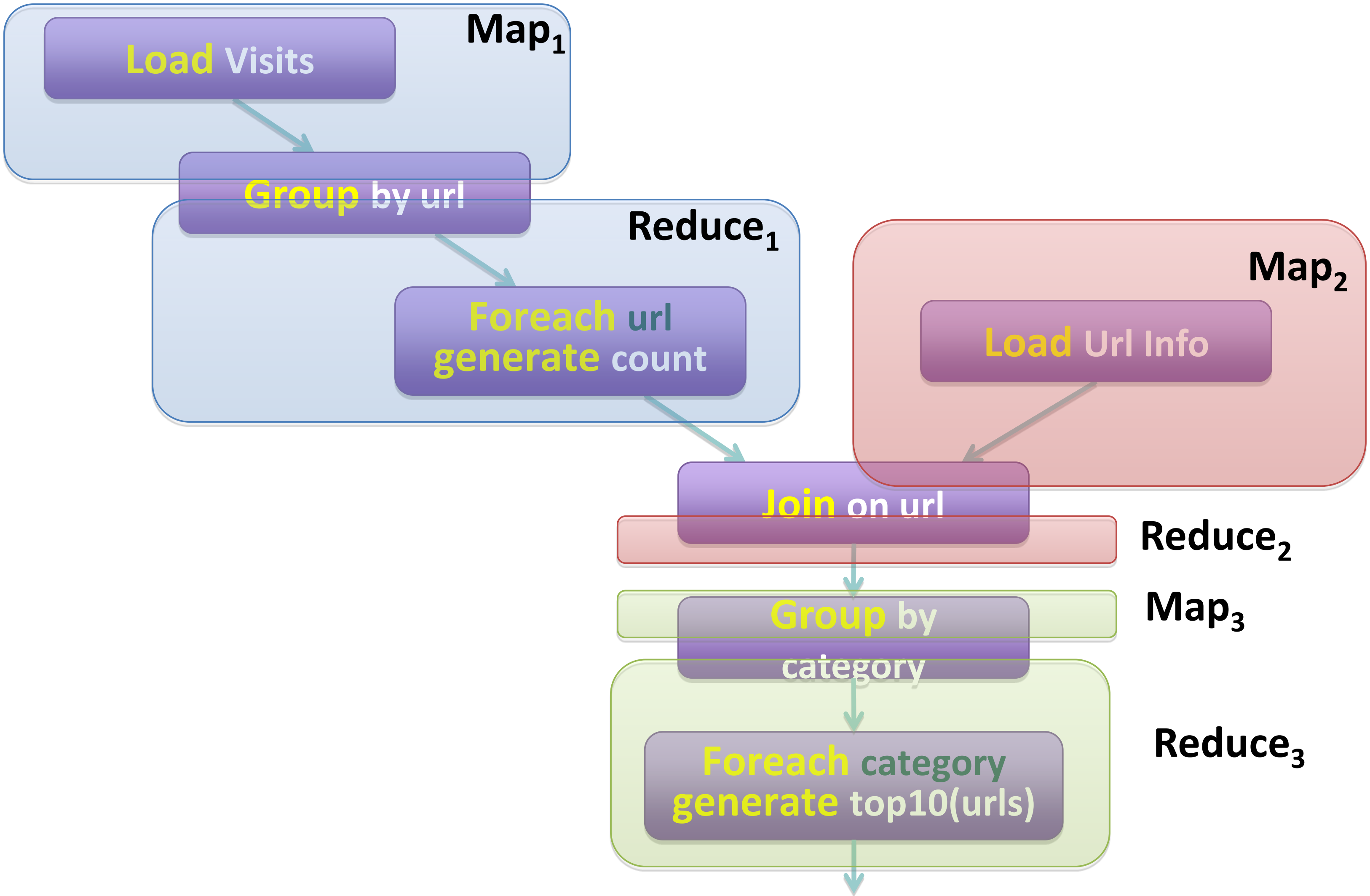
Pig Script

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

Pig Script in Hadoop



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

Parallel Databases $\leftrightarrow$ MapReduce

- Lots of synergy between parallel databases and MapReduce
- Communities have much to learn from each other
- Bottom line: use the right tool for the job!

Take Home Messages

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