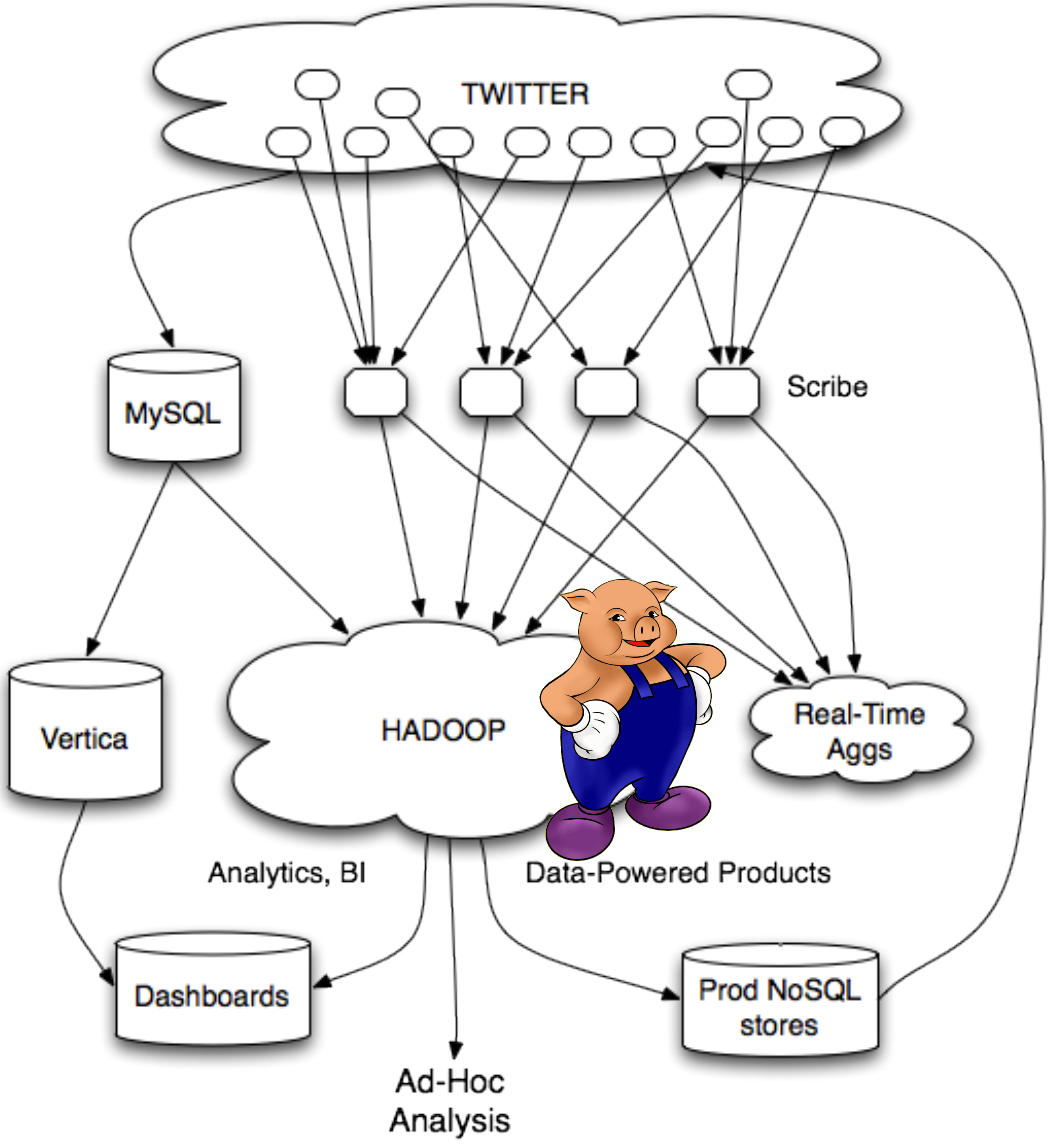
THE UNIVERSITY *of* EDINBURGH
informatics

Extreme Computing

Data warehousing in the Cloud era


TWITTER
Scribe
MySQL
Vertica
HADOOP
Real-Time Aggs
Analytics, BI
Data-Powered Products
Dashboards
Ad-Hoc Analysis
Prod NoSQL stores

Google

processes 20 PB a day (2008)
crawls 20B web pages a day (2012)

JPMorganChase

150 PB on 50k+ servers
running 15k apps (6/2011)

ebay

>10 PB data, 75B DB calls per day (6/2012)

>100 PB of user data
+
500 TB/day (8/2012)

facebook

Today's agenda

- How we got here: the historical perspective
- MapReduce algorithms for processing relational data
- Evolving roles of relational databases and MapReduce

HISTORY

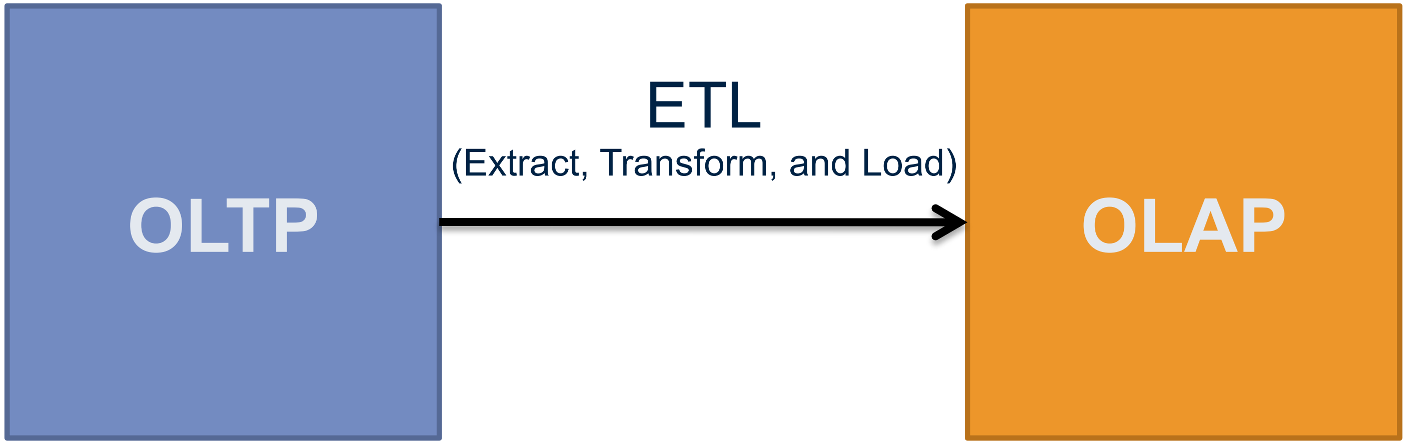
Database workloads

- 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 per query

If one is good, two is better

- 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

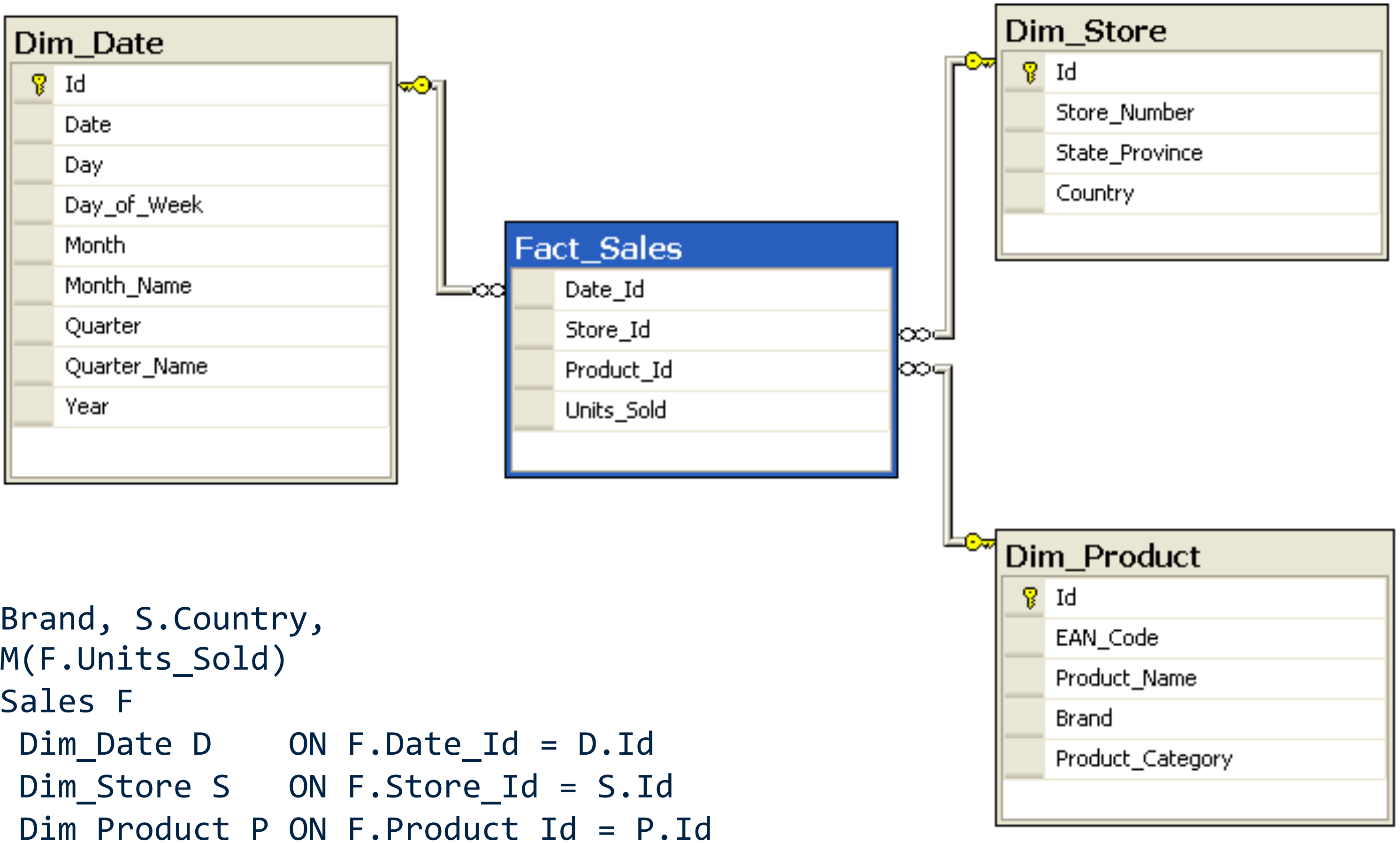
ETL
(Extract, Transform, and Load)

OLAP

OLTP/OLAP integration

- OLTP database for user-facing transactions
- Extract-Transform-Load (ETL)
  - Extract records from source
  - Transform: clean data, check integrity, aggregate, etc.
  - Load into OLAP database
- OLAP database for data warehousing

Structure of data warehouses



Dim_Date
- Id
- Date
- Day
- Day_of_Week
- Month
- Month_Name
- Quarter
- Quarter_Name
- Year

Fact_Sales
- Date_Id
- Store_Id
- Product_Id
- Units_Sold

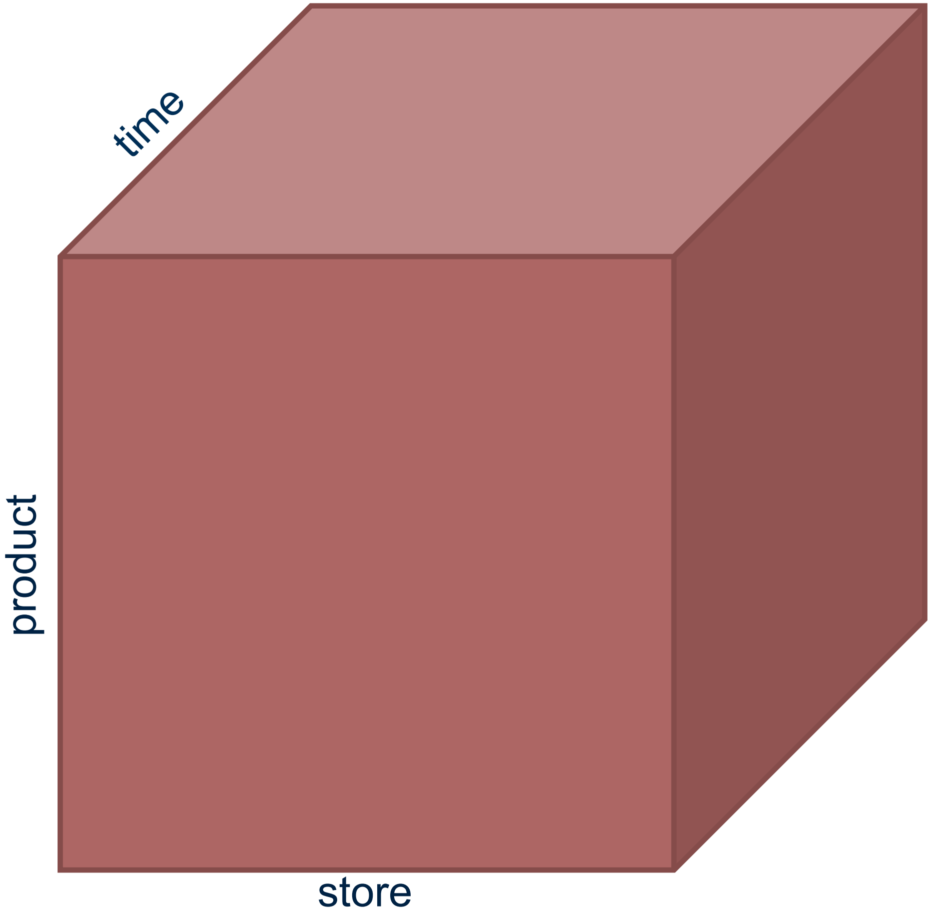
Dim_Store
- Id
- Store_Number
- State_Province
- Country

Dim_Product
- Id
- EAN_Code
- Product_Name
- Brand
- Product_Category

```
SELECT  P.Brand, S.Country,
        SUM(F.Units_Sold)
FROM Fact_Sales F
INNER JOIN Dim_Date D    ON F.Date_Id = D.Id
INNER JOIN Dim_Store S   ON F.Store_Id = S.Id
INNER JOIN Dim_Product P ON F.Product_Id = P.Id
WHERE D.YEAR = 1997 AND P.Product_Category = 'tv'
GROUP BY P.Brand, S.Country;
```

Source: Wikipedia (Star Schema)

OLAP cubes



Common operations

- slice and dice
- roll up/drill down
- pivot

OLAP cubes: research challenges

- Fundamentally, lots of group-bys and aggregations
  - How to take advantage of schema structure to avoid repeated work?
- Cube materialization
  - Realistic to materialize the entire cube?
  - If not, how/when/what to materialize?

facebook®

Jeff Hammerbacher, Information Platforms and the Rise of the Data Scientist.
In, *Beautiful Data*, O'Reilly, 2009.

> "On the first day of logging the Facebook clickstream, more than 400 gigabytes of data was collected. The load, index, and aggregation processes for this data set really taxed the Oracle data warehouse. Even after significant tuning, we were unable to aggregate a day of clickstream data in less than 24 hours."

RELATIONAL PROCESSING USING MAPREDUCE

What's changed?

- Dropping cost of disks
  - Cheaper to store everything than to figure out what to throw away
- Types of data collected
  - From data that's obviously valuable to data whose value is less apparent
- Rise of social media and user-generated content
  - Large increase in data volume
- Growing maturity of data mining techniques
  - Demonstrates value of data analytics
- Virtuous product growth cycle

ETL bottleneck

- ETL is typically a nightly task:
  - What happens if processing 24 hours of data takes longer than 24 hours?
- Hadoop is perfect:
  - 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!

We need algorithms for ETL processing using MapReduce

Design pattern: secondary sorting

- MapReduce sorts input to reducers by key
  - Values are arbitrarily ordered
- What if want to sort value also?
  - E.g., k → $(v_1, r), (v_3, r), (v_4, r), (v_8, r)\ldots$

Secondary sorting: solutions

- Solution 1:
  - Buffer values in memory, then sort
  - Why is this a bad idea?
- Solution 2:
  - "Value-to-key conversion" design pattern: form composite intermediate key, (k, v_1)
  - Let execution framework do the sorting
  - Preserve state across multiple key-value pairs to handle processing
  - Anything else we need to do?

Value-to-key conversion

Before

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

Values arrive in arbitrary order…

After

$(k, v_1) \longrightarrow (v_1, r)$

$(k, v_3) \longrightarrow (v_3, r)$

$(k, v_4) \longrightarrow (v_4, r)$

$(k, v_8) \longrightarrow (v_8, r)$

…

Values arrive in sorted order…

Process by preserving state across multiple keys

Remember to partition correctly!

Relational databases

- A relational database is comprised of tables
- Each table represents a relation = collection of tuples (rows)
- Each tuple consists of multiple fields

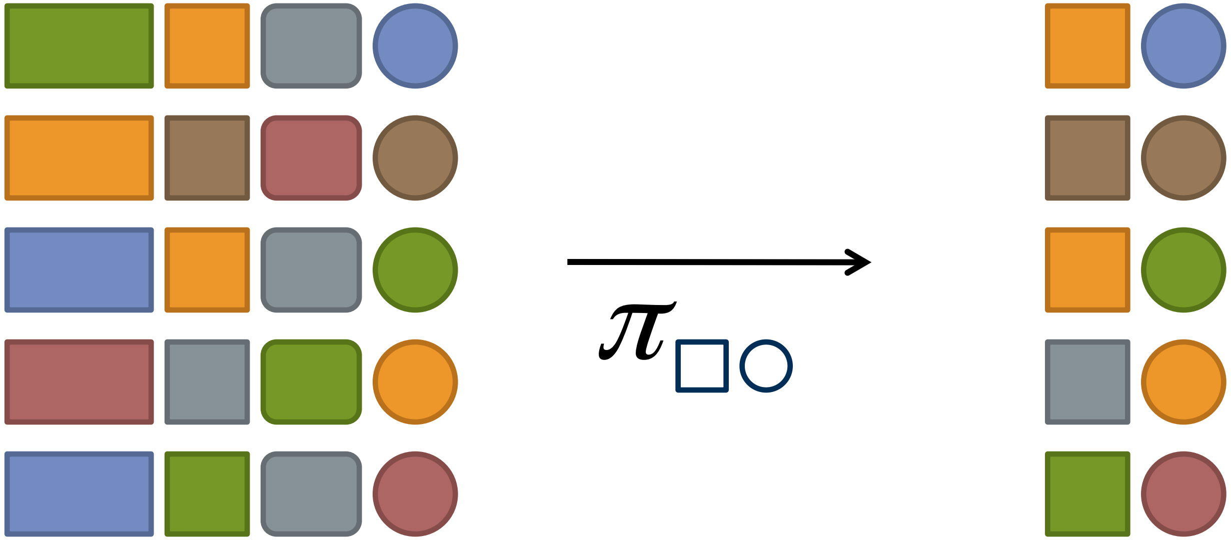
Working scenario

- 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)
- Other operations
  - Join ($\bowtie$)
  - Group by… aggregation
  - …

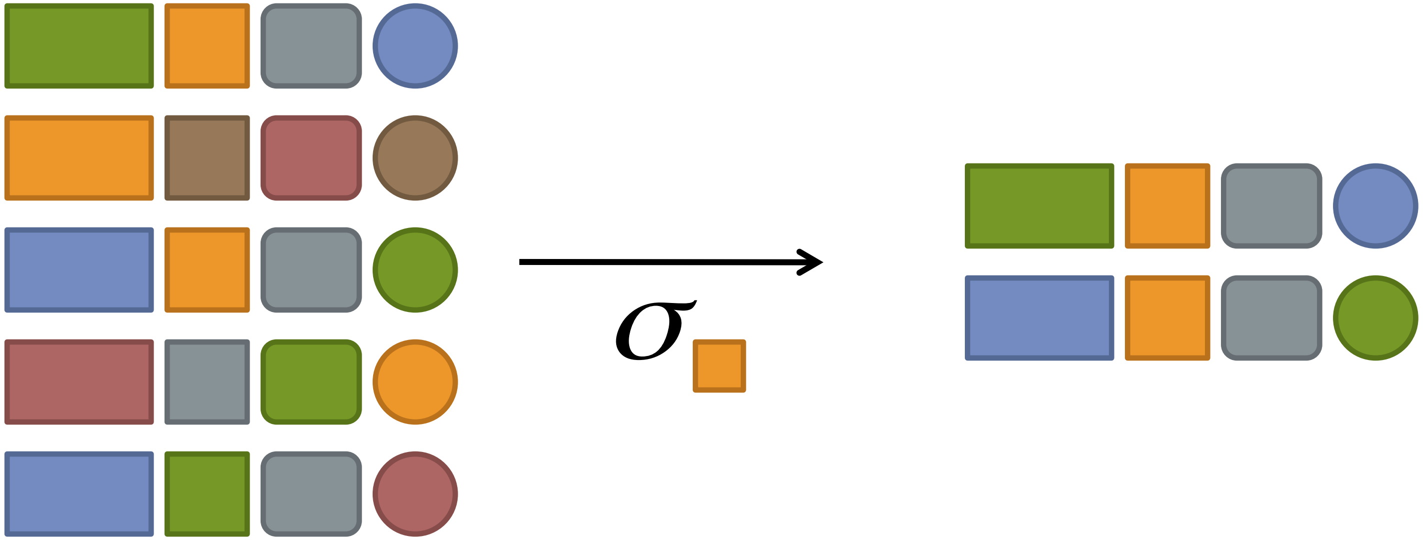
Projection



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
  - Take advantage of compression when available
  - Semistructured data? No problem!

Selection



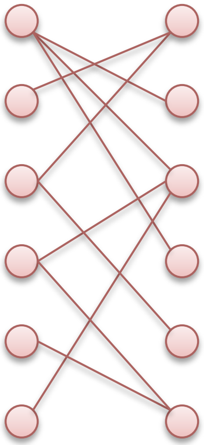
Selection in MapReduce

- 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
  - Take advantage of compression when available
  - Semistructured data? No problem!

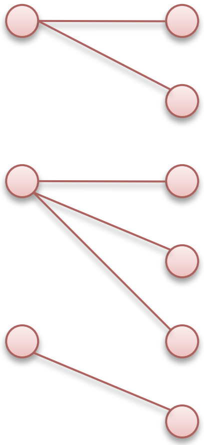
Group by and aggregation

- 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

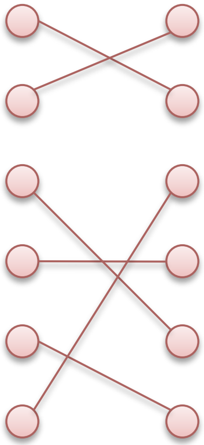
Types of join relationships



Many-to-Many



One-to-Many



One-to-One

Join algorithms in MapReduce

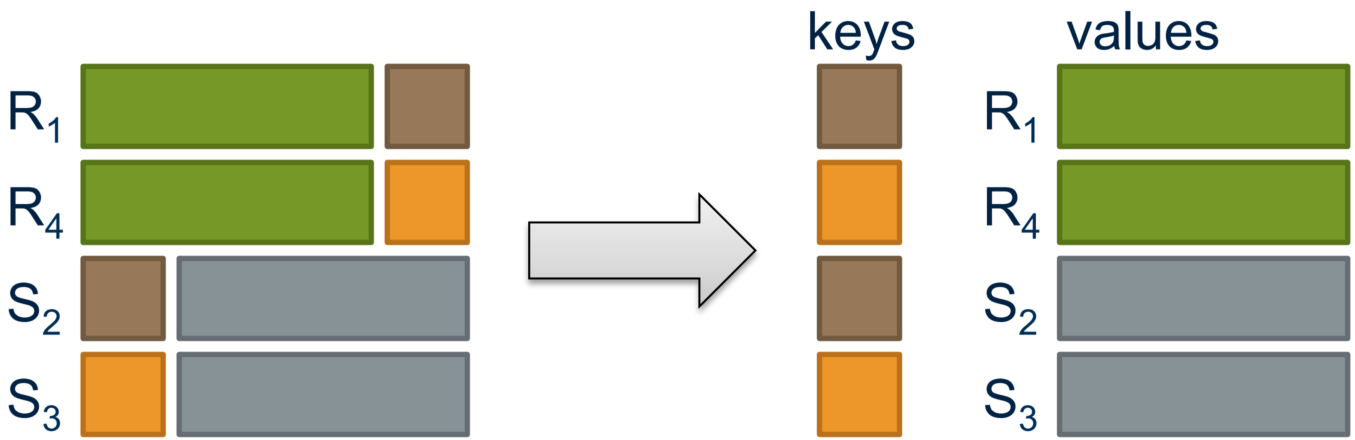
- Reduce-side join
- Map-side join
- In-memory join
  - Striped variant
  - Memcached variant

Reduce-side join

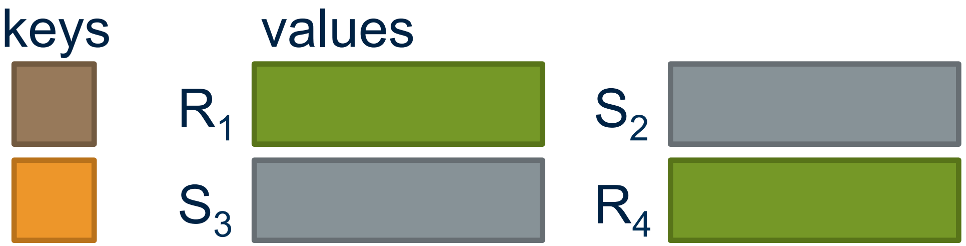
- Basic idea: group by join key
  - Map over both sets of tuples
  - Emit tuple as value with join key as the intermediate key
  - Execution framework brings together tuples sharing the same key
  - Perform actual join in reducer
  - Similar to a “sort-merge join” in database terminology
- Two variants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

Reduce-side join: 1-to-1

Map



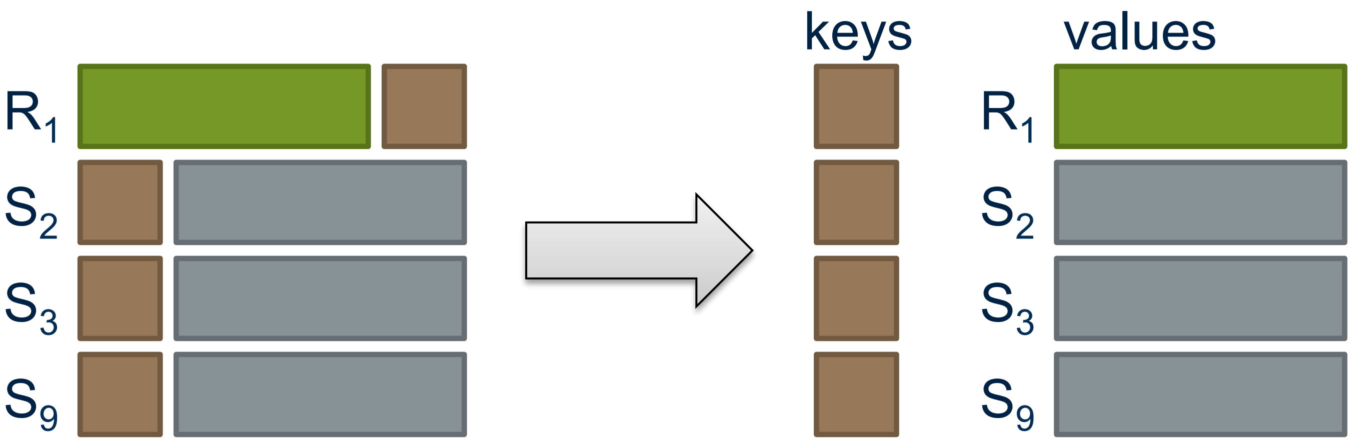
Reduce



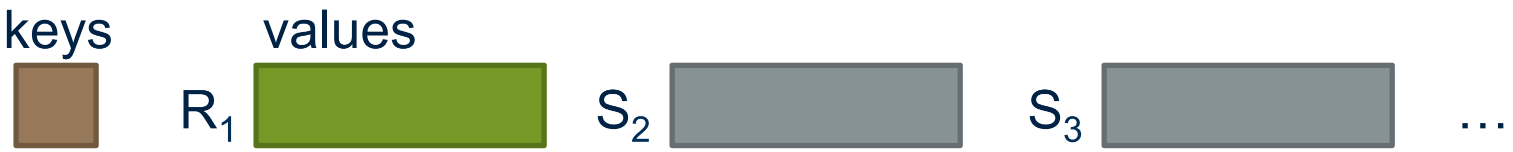
Note: no guarantee if R is going to come first or S

Reduce-side join: 1-to-many

Map



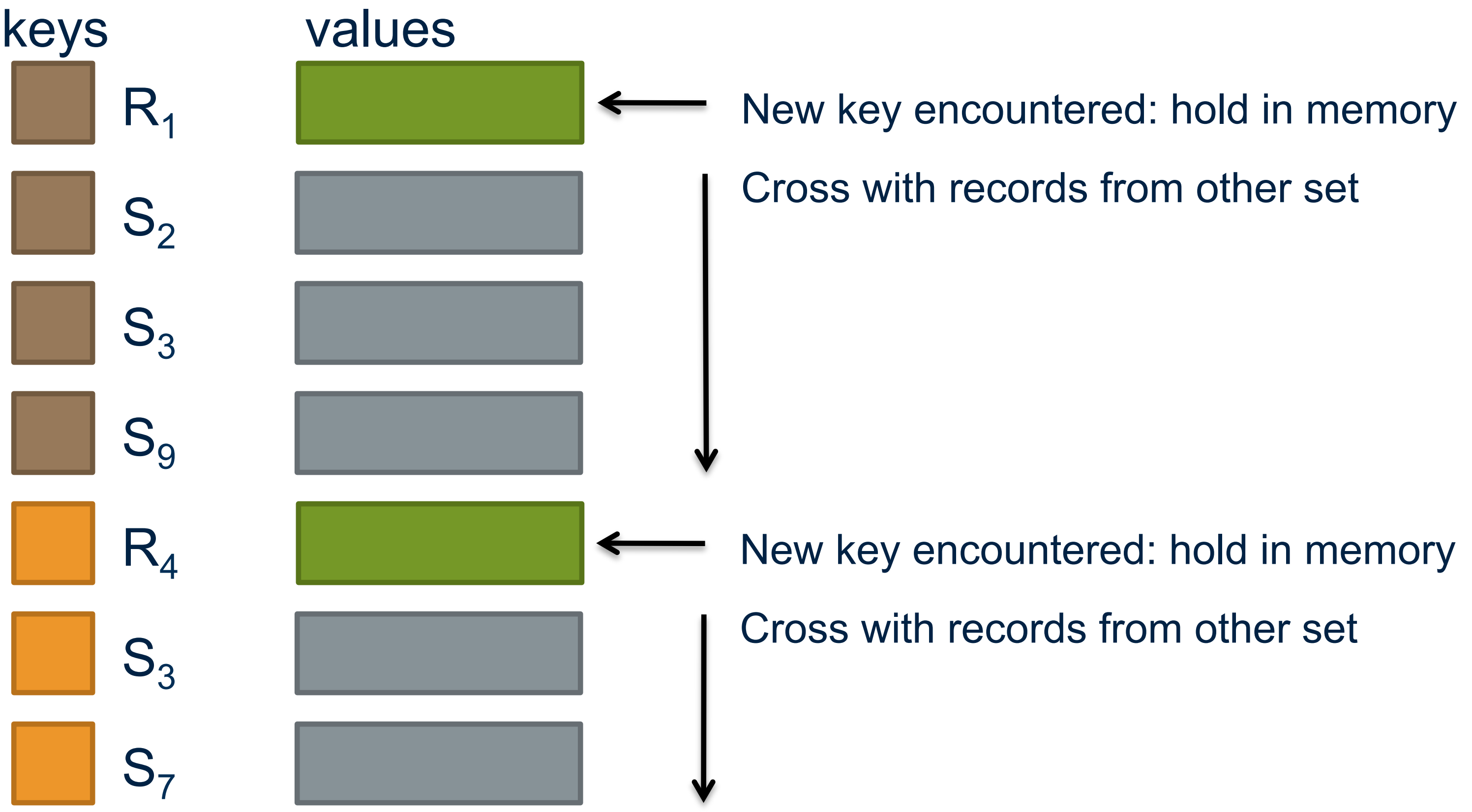
Reduce



What's the problem?

Reduce-side join: value-to-key conversion

In reducer…



New key encountered: hold in memory

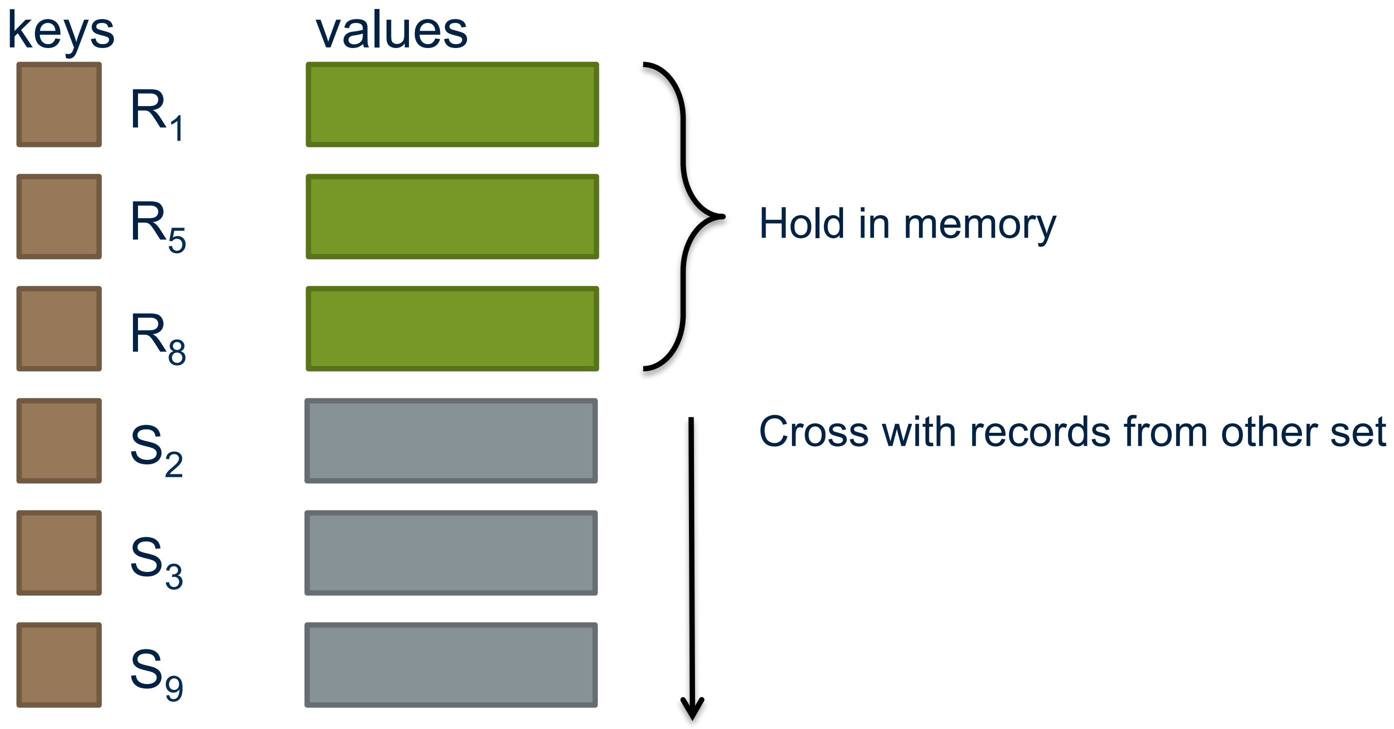
Cross with records from other set

New key encountered: hold in memory

Cross with records from other set

Reduce-side join: many-to-many

In reducer…



keys

values

R_1

R_5

R_8

S_2

S_3

S_9

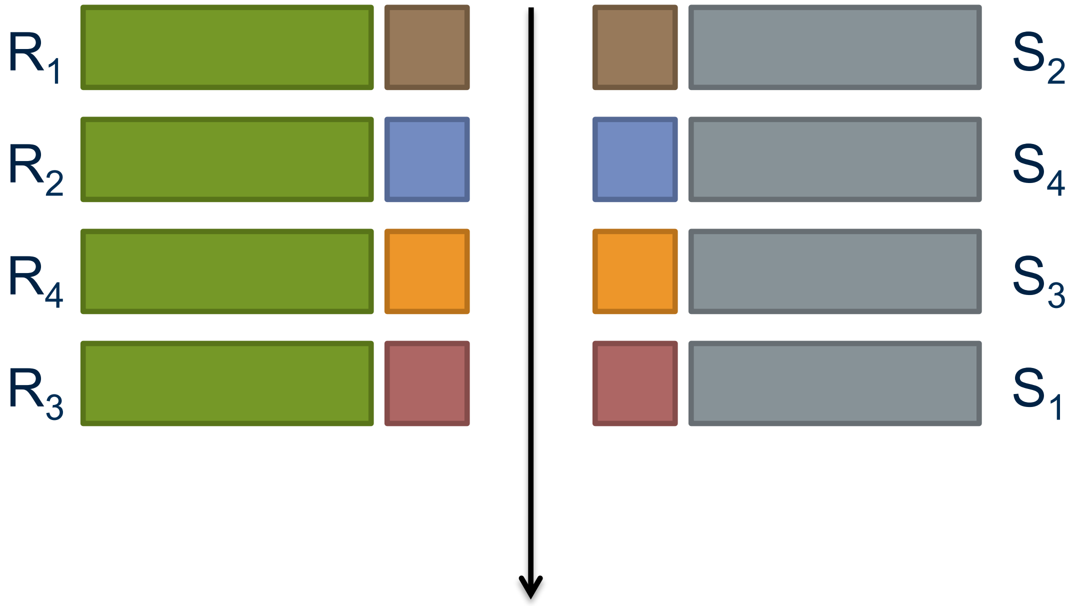
Hold in memory

Cross with records from other set

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
- 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
- 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

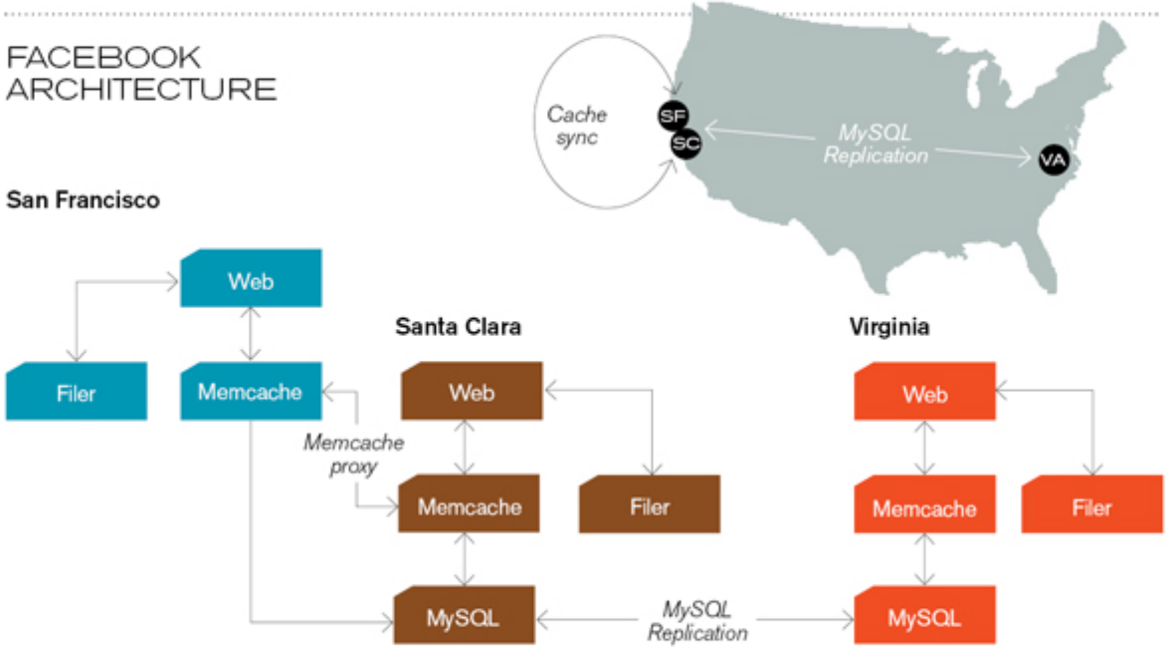
In-memory join: variants

- Striped variant:
  - R too big to fit into memory?
  - Divide R into R_1, R_2, R_3, … such that 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:
  - Load R into memcached
  - Replace in-memory hash lookup with memcached lookup

THE UNIVERSITY *of* EDINBURGH
informatics

Memcached

Circa 2008 Architecture



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

- Memcached join:
  - Load R into memcached
  - Replace in-memory hash lookup with memcached lookup
- 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
- Complex operations require multiple MapReduce jobs
  - Example: top ten URLs in terms of average time spent
  - Opportunities for automatic optimization

HIGH-LEVEL WORKFLOWS

Need for high-level languages

- Hadoop is great for large-data processing!
  - But writing Java programs for everything is verbose and slow
  - Data scientists don’t want to 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 with different encodings
  - Developed by Facebook, now open source
- Pig: large-scale data processing system
  - Scripts are written in Pig Latin, a dataflow language
  - Programmer focuses on data transformations
  - Developed by Yahoo!, now open source
- 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

⋮

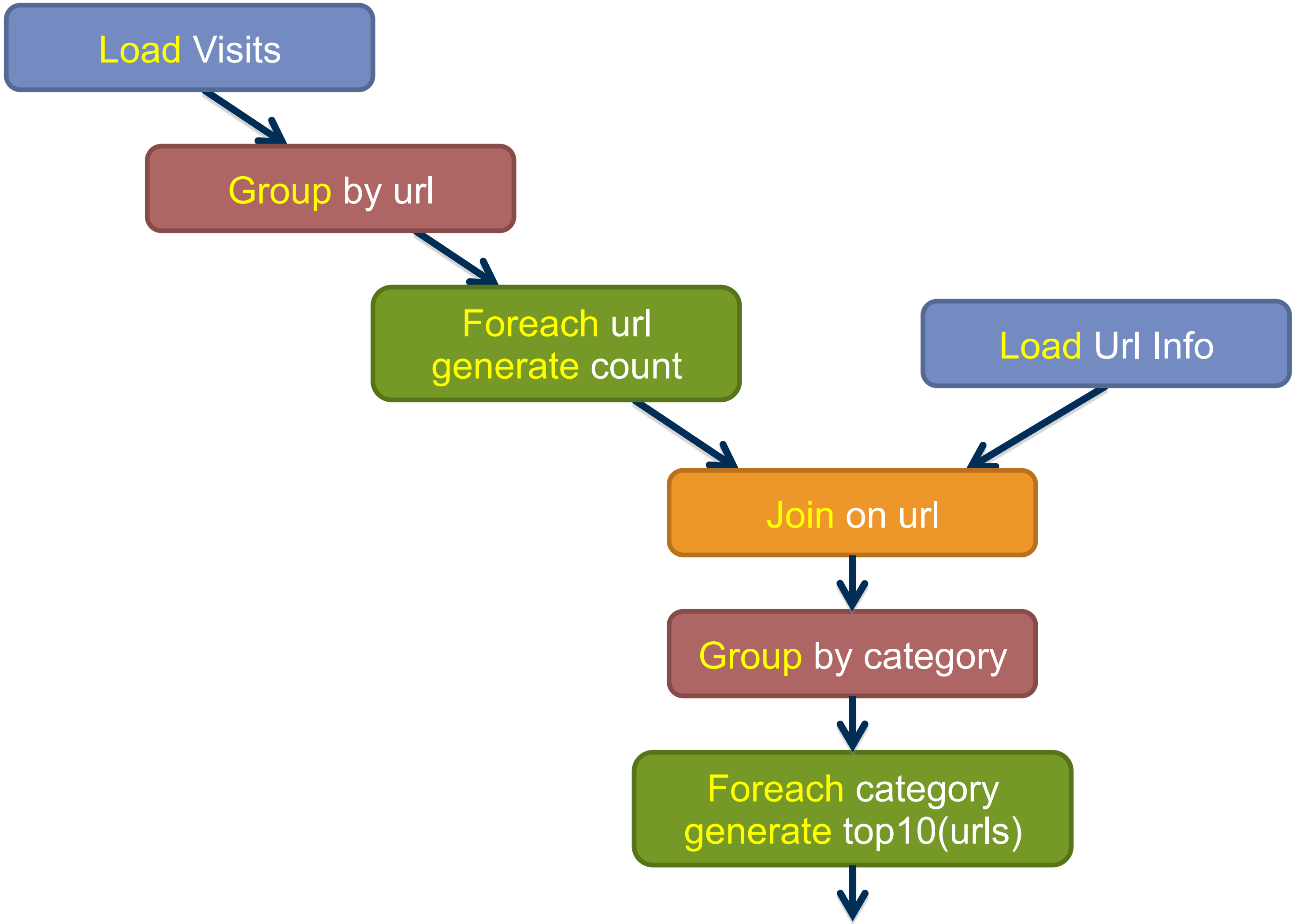
Url Info

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

⋮

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

Pig query plan



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

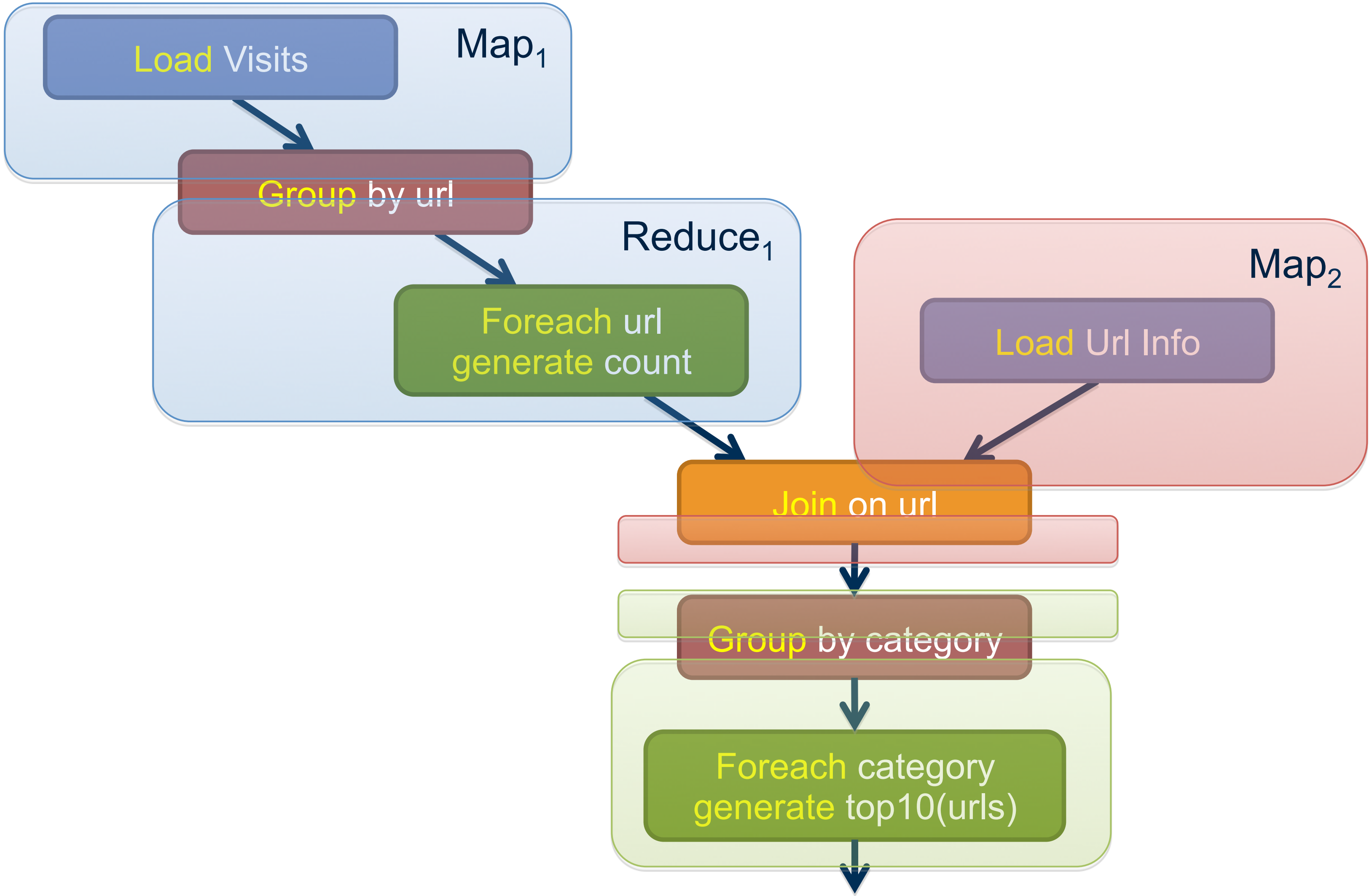
Pig script

```
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’;
```

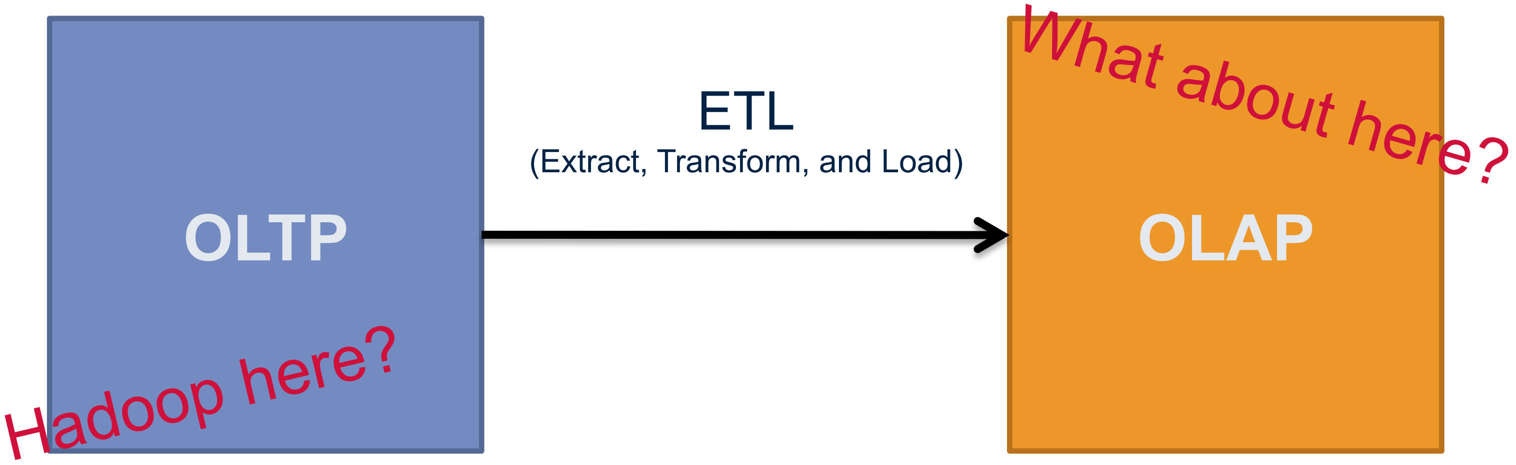
Pig Slides adapted from Olston et al. (SIGMOD 2008)

Pig query plan



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

Where does Hadoop go?



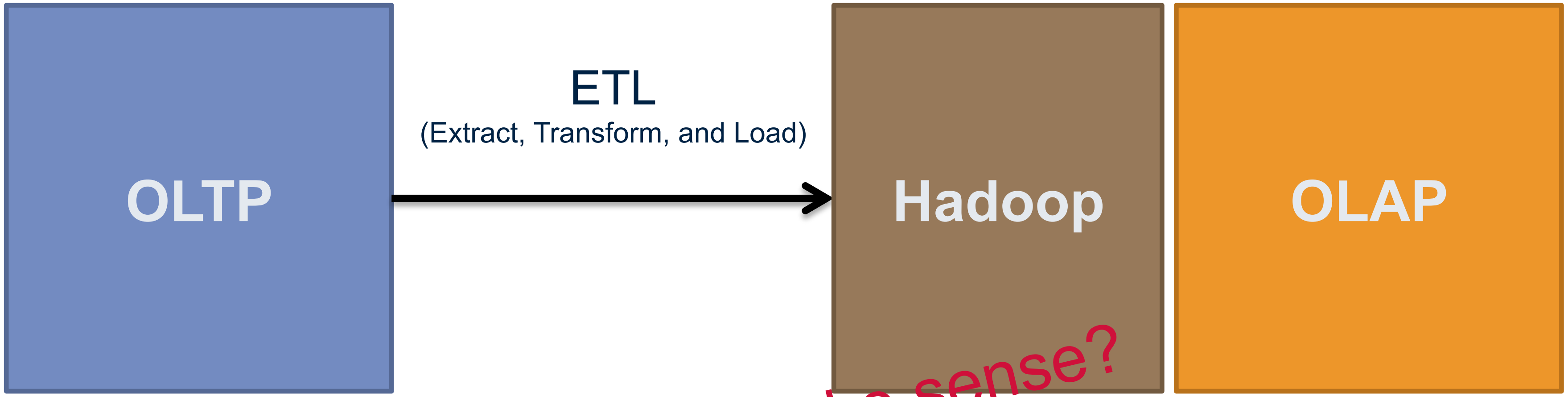
A major step backwards?

- MapReduce is a step backward in database access
  - Schemas are good
  - Separation of the schema from the application is good
  - High-level access languages are good
- MapReduce is poor implementation
  - Brute force and only brute force (no indexes, for example)
- MapReduce is not novel
- MapReduce is missing features
  - Bulk loader, indexing, updates, transactions…
- MapReduce is incompatible with DMBS tools

Known and unknown unknowns

- Databases only help if you know what questions to ask
  - “Known unknowns”
- What’s if you don’t know what you’re looking for?
  - “Unknown unknowns”

OLTP/OLAP/Hadoop Architecture



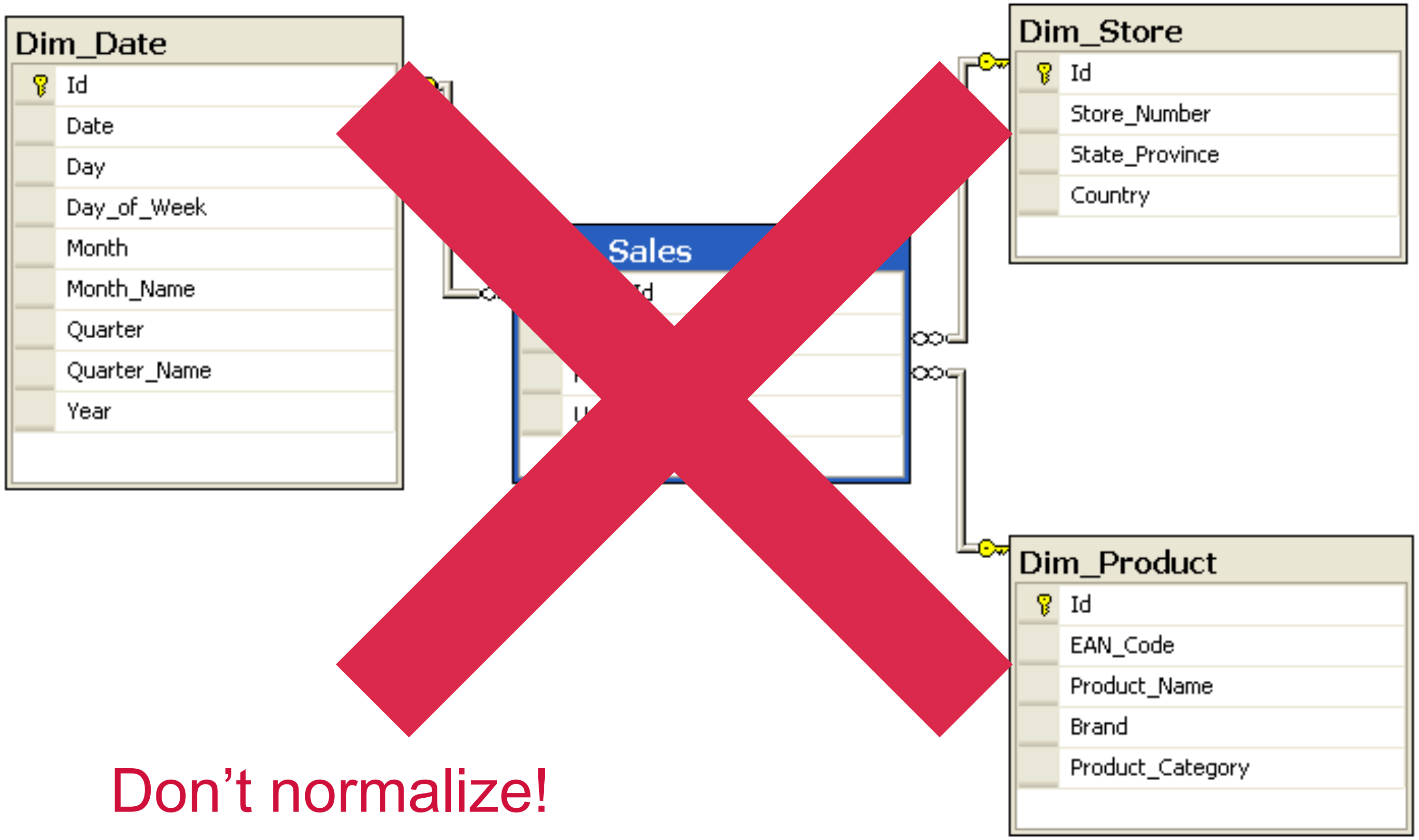
Why does this make sense?

ETL: redux

- Often, with noisy datasets, ETL *is* the analysis!
- Note that ETL necessarily involves brute force data scans
- L, then E and T?

THE UNIVERSITY *of* EDINBURGH
informatics

Structure of Hadoop warehouses



Don’t normalize!

Source: Wikipedia (Star Schema)

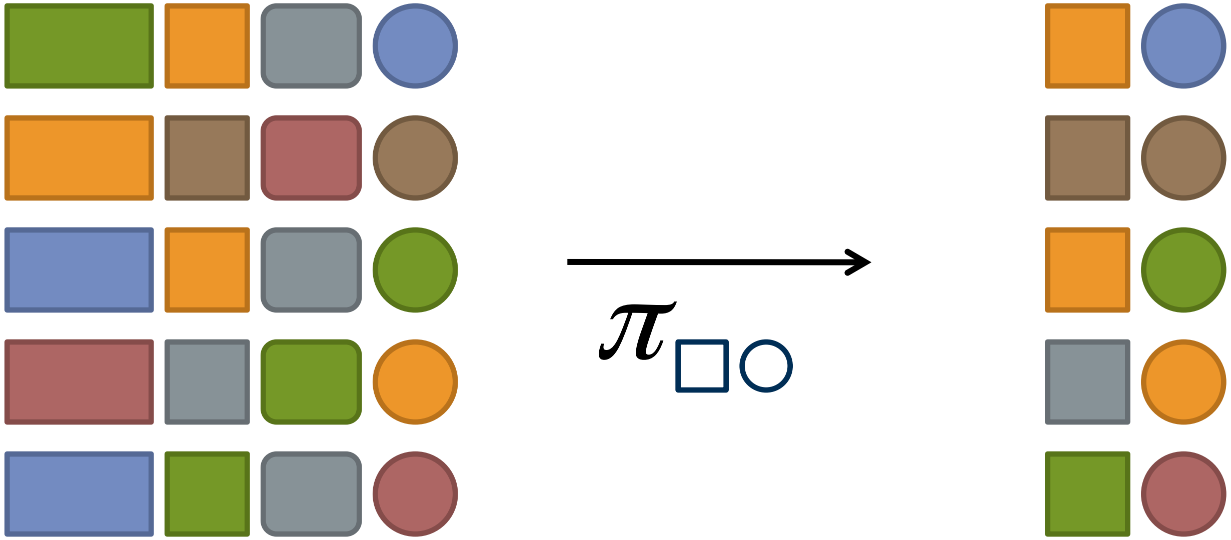
A major step backwards?

- MapReduce is a step backward in database access:
  - Schemas are good
  - Separation of the schema from the application is good
  - High-level access languages are good
- MapReduce is poor implementation
  - Brute force and only brute force (no indexes, for example)
- MapReduce is not novel
- MapReduce is missing features
  - Bulk loader, indexing, updates, transactions…
- MapReduce is incompatible with DMBS tools

Bottom line: issue of maturity, not fundamental capability!

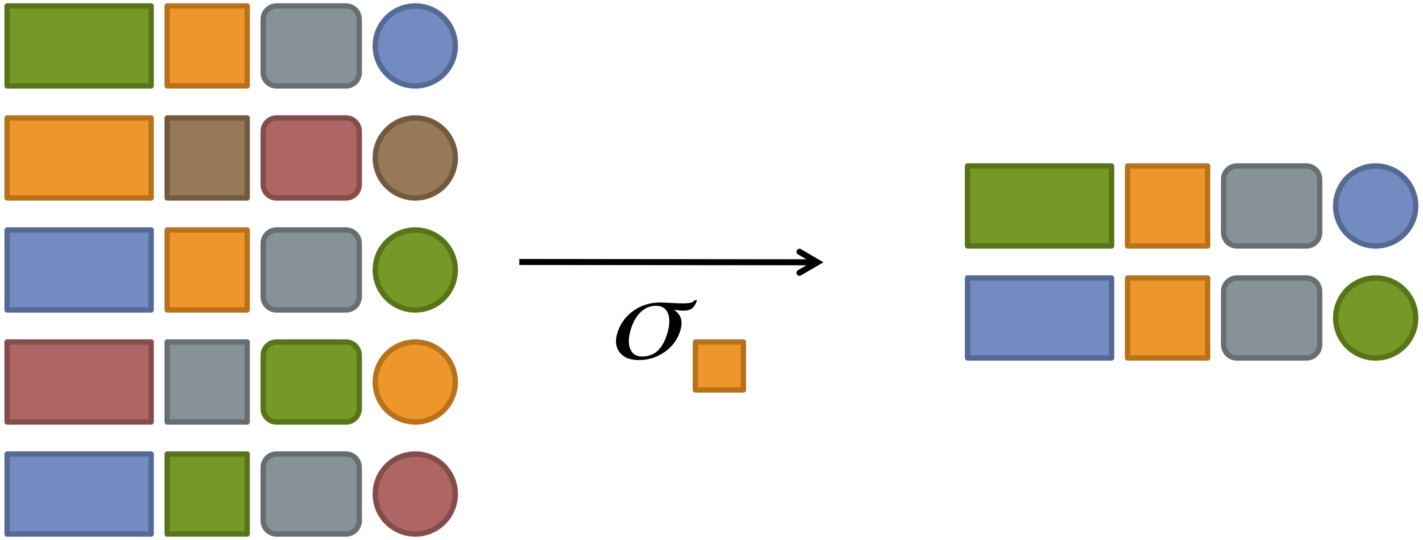
Source: Blog post by DeWitt and Stonebraker

Projection in MapReduce



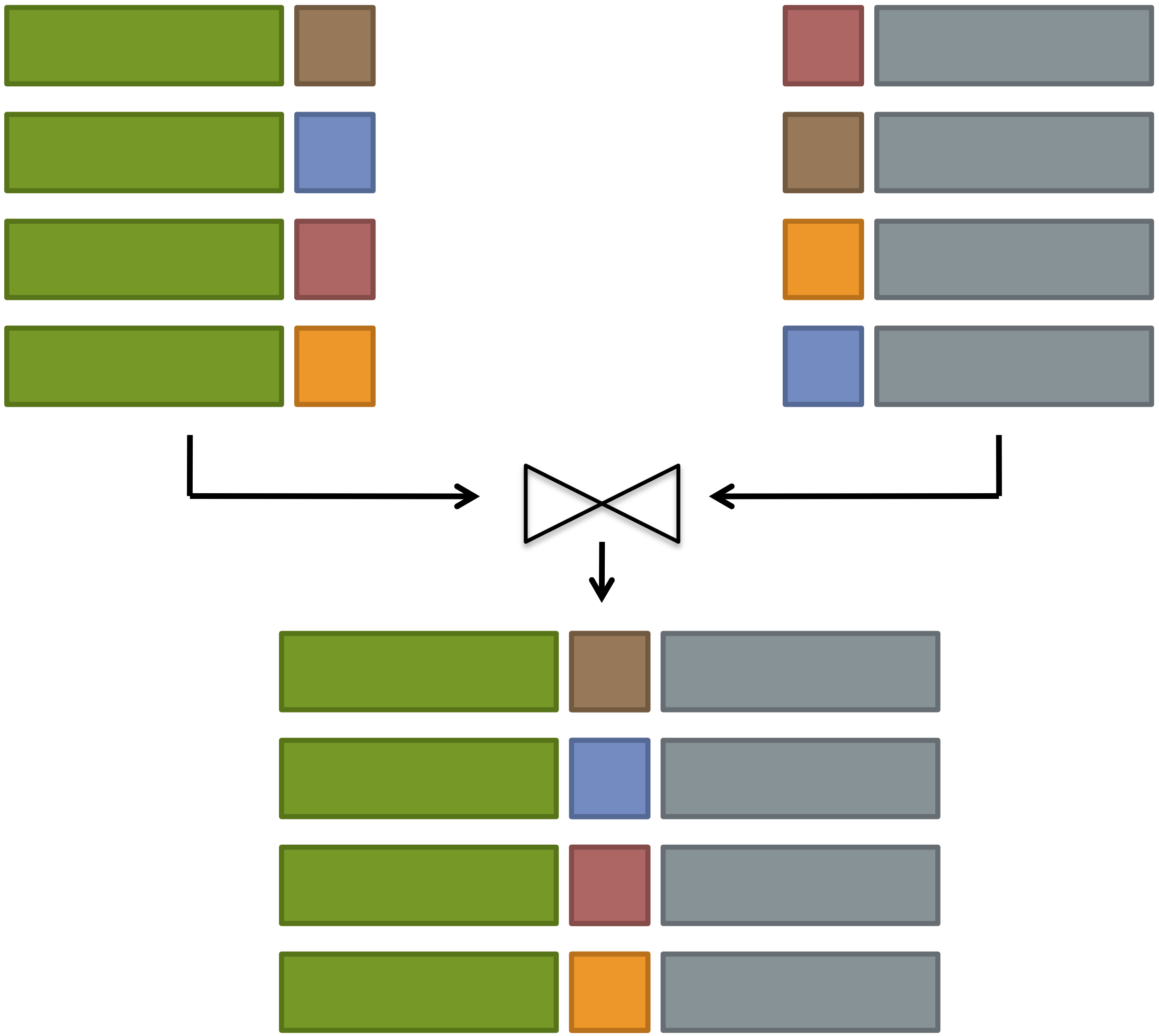
Can we do better than brute force?

Selection in MapReduce



Can we do better than brute force?

Relational joins in MapReduce



Query optimizers to the rescue!

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 query languages
  - Programmers retain control over performance
  - Open source

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

Philosophical differences

- Parallel relational databases
  - Schema on write
  - Failures are relatively infrequent
  - “Possessive” of data
  - Mostly proprietary
- MapReduce
  - Schema on read
  - Failures are relatively common
  - In situ data processing
  - Open source

Summary

- How we got here: the historical perspective
- MapReduce algorithms for processing relational data
- Evolving roles of relational databases and MapReduce