Analytics Queries for Large Data Volumes

Gianluca Demartini

DATA7201 Data Analytics at Scale

Week 4

Week	Date	Lecture	Prac	Assessment
1	21-Feb	Introduction to DATA7201 - Data Analytics at Scale	-	
2	28-Feb	Supporting Infrastructures and Use Cases	-	
3	6-Mar	Storage Infrastructures for Large Data Volumes	Intro to Cluster and HDFS	
4	13-Mar	Analytics Queries for Large Data Volumes	PIG(1)	
5	20-Mar	Distributed Data Processing	PIG (2)	
6	27-Mar	Processing Large Data Streams	PySpark (1)	**Quiz 1 Due (5)**
Semester Break				
7	10-Apr	Processing Large Graph Data (1) + use cases	PySpark (2)	
8	17-Apr	Processing Large Graph Data (2) + use cases	Project support	
9	24-Apr	Recommender Systems	Project support	**Quiz 2 Due (5)**
10	1-May	Opinion Mining + use cases	Project support	
11	8-May	Health Data Analytics (guest speaker)	Project support	
12	15-May	Large Language Models?	Project support	**Report Due (45)**
13	22-May	Course Revision	-	**Quiz 3 Due (5)**

Last Week

- Distributed Infrastructures
  - CAP Theorem
- Google File System
- HDFS
- Spark RDDs
- Column-stores / noSQL

Lecture Outline

- OLAP vs OLTP
- Architectures for Distributed Databases
- Big Table
- HBase and Hive
- PIG

OLAP / OLTP

- **O**n**l**ine **T**ransaction **P**rocessing
- **O**n**l**ine **A**nalytical **P**rocessing
  - OLTP-style: create sales order, invoice, accounting documents, display customer master data or sales order
  - OLAP-style: dunning, available-to-promise, cross selling, operational reporting (list open sales orders)
- Modern enterprise resource planning (ERP) systems are challenged by **mixed workloads**

OLAP / OLTP

- Systems are optimized **either** for daily **transactional** or **analytical** workloads

- OLAP systems do not have the **latest** data
- OLAP systems only have **predefined subset** of the data
- Separation introduces data **redundancy**

Structured Query Language: SQL

- Declarative query language
- Multiple aspects of the language
  - Data definition language
    - Statements to create, modify tables and views
  - Data manipulation language
    - Statements to issue queries, insert, delete data
- More

SQL Example

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

```
SELECT  PName, Price, Manufacturer
FROM    Product
WHERE   Price > 100
```

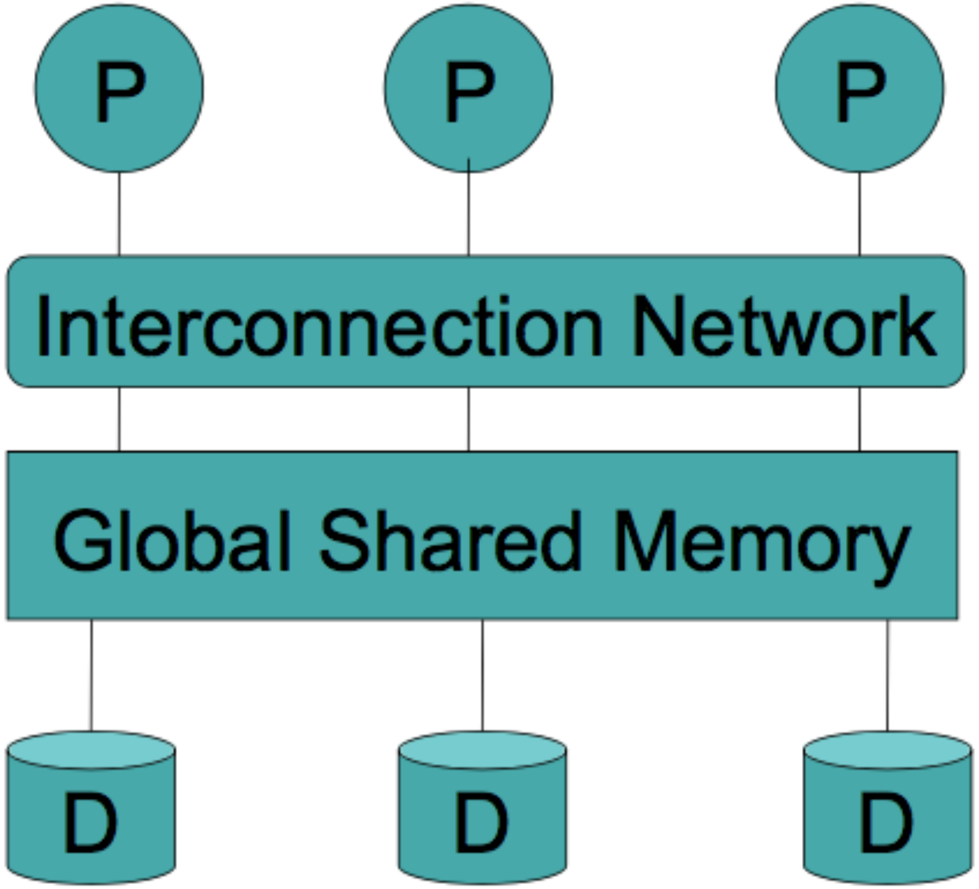
“selection” and “projection”

PName	Price	Manufacturer
SingleTouch	$149.99	Canon
MultiTouch	$203.99	Hitachi

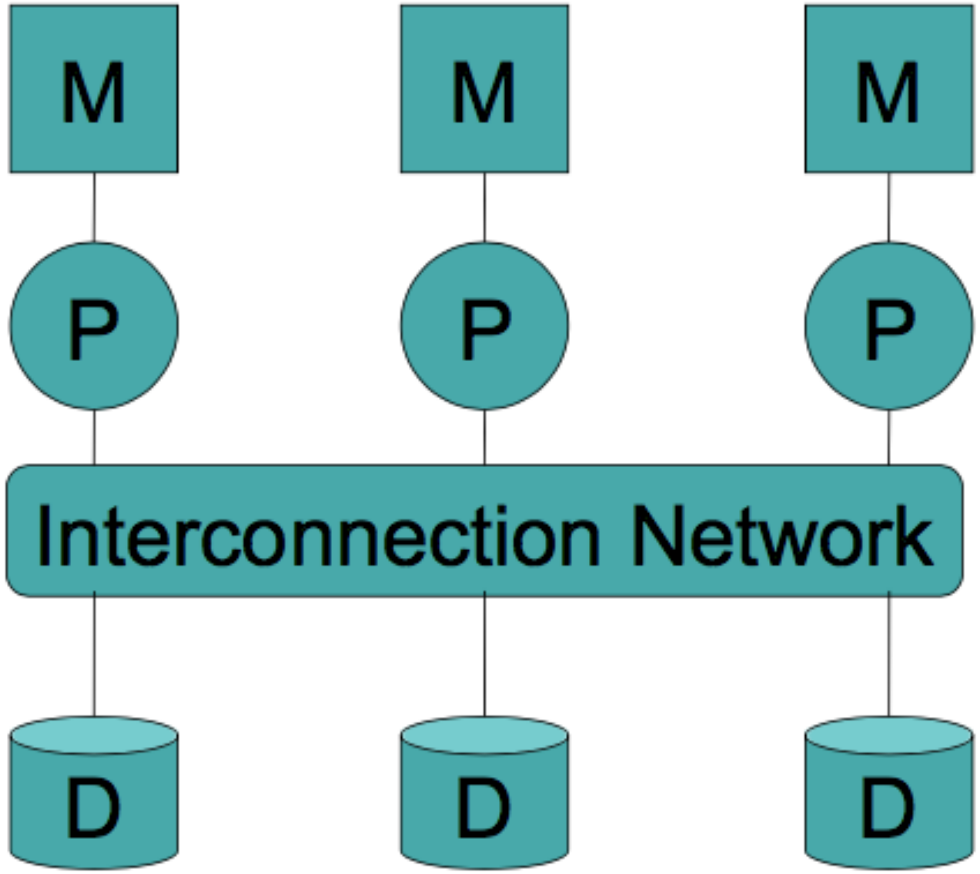
Distributed Databases

- Databases having storage devices distributed over a network of connected computers
  - **Replication** (keep synchronized versions of the data)
  - **Duplication** (periodically backup of the entire database and keep a copy; only modify the master copy)
- Distributed query of the database

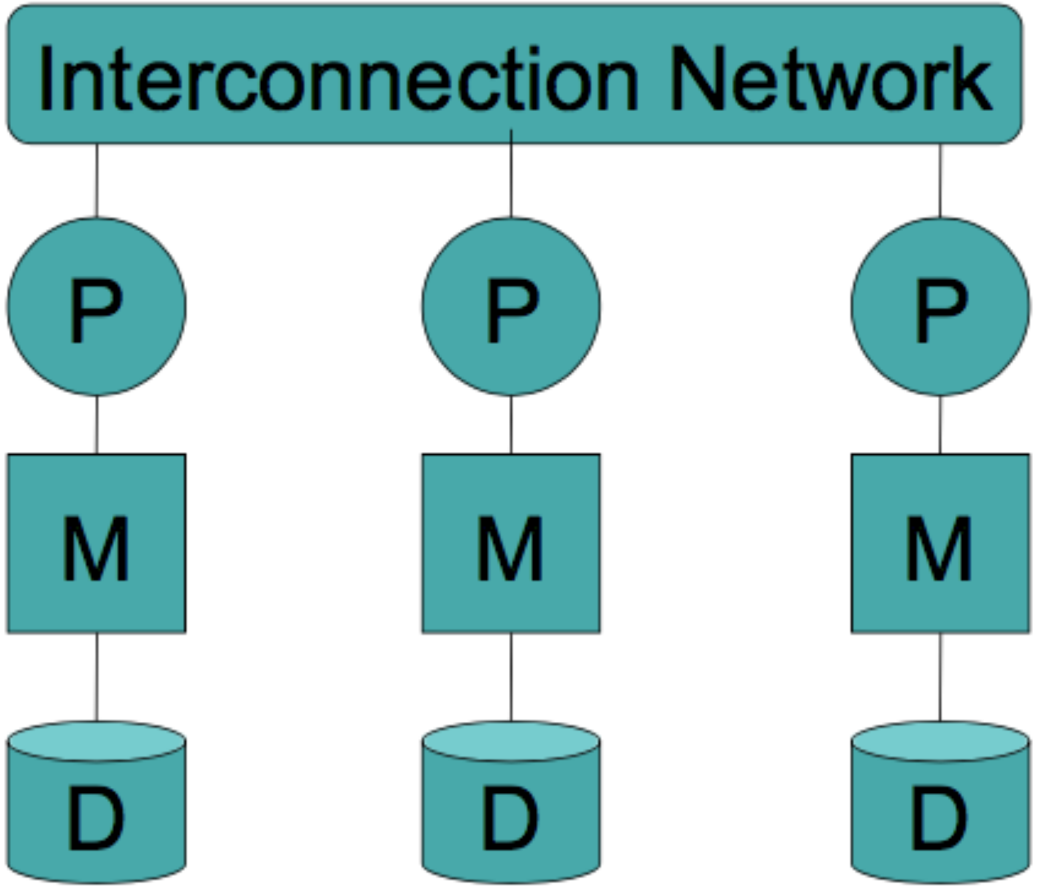
Shared Memory



Shared Disk



Shared Nothing



Shared Nothing

- Most scalable architecture
  - Minimizes interference by minimizing resource sharing
  - Can use commodity hardware
- Also most difficult to program and manage
  - Processor=server=node
  - P=number of nodes

Parallel Query Evaluation

- Inter-operator parallelism (most scalable)
  - A query runs on multiple processors
  - An operator runs on one processor

Horizontal Data Partitioning (Sharding)

- Typical shared-nothing parallelization

- Relation (i.e., table) R split into P chunks $R_0,\ldots,R_{P-1}$, stored at the P nodes
  - **Round robin**: tuple t_i to chunk (i mod P)
  - **Hash based** partitioning on attribute A:
    - Tuple t to chunk h(t.A) mod P
  - **Range based** partitioning on attribute A:
    - Tuple t to chunk i If $v_{i-1} < t.A < v_i$

Data Partitioning Revisited

- What are the pros and cons ?
- Round robin
  - Good load balance but always needs to read all the data
- Hash based partitioning
  - Good load balance but works only for equality predicates and full scans
- Range based partitioning
  - Works well for range predicates but can suffer from data skew

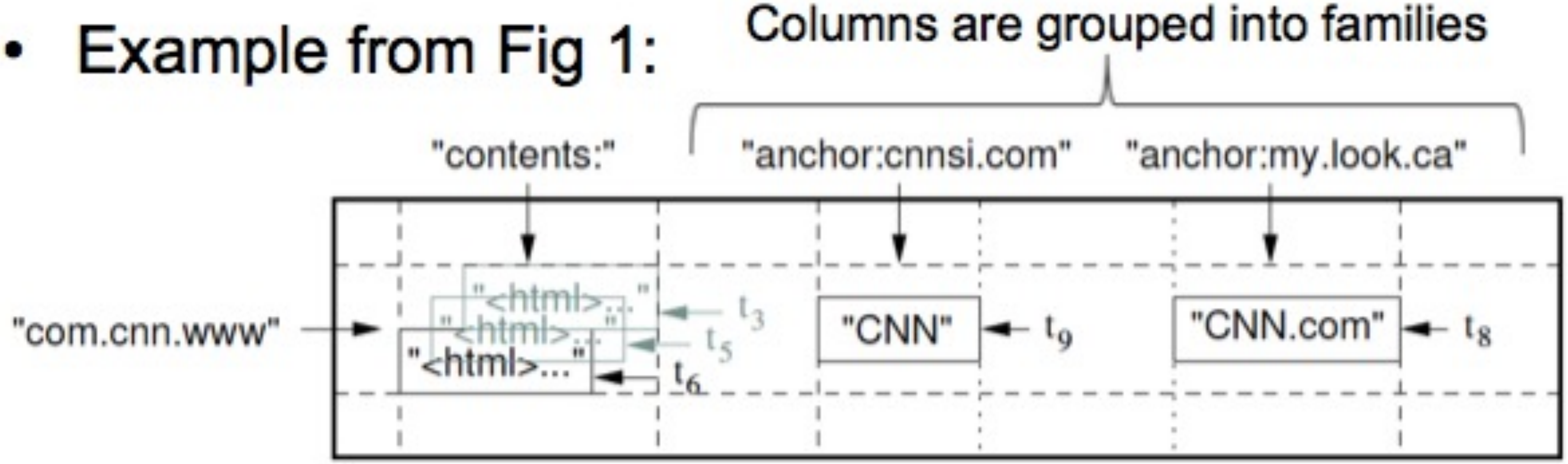
Google Big Table

- Distributed storage system
- Designed to
  - Hold (semi) structured data
  - Scale to thousands of servers
  - Store up to several hundred terabytes (maybe even petabytes)
  - Perform backend bulk processing
  - Perform real-time data serving
- To scale, Big Table has a limited set of features

Big Table

- Sparse, multidimensional sorted map
  (row:string, column:string, time:int64) -> string
  - Notice how everything but time is a string

- Example from Fig 1:



Big Table - Features

- Read/writes of data under single row key is atomic
  - Only single-row transactions!
- Data is stored in lexicographical order
  - Improves data access locality
- Column families (i.e., tables) are unit of access control
- Data is versioned (old versions can be garbage-collected)
  - Example: most recent three crawls of each page, with times

Apache HBase

Apache HBase

- "Apache HBase is the Hadoop database, a distributed, scalable, big data store."
- On top of HDFS
- Billions of rows * millions of columns
- Non-relational DB / NoSQL
- Column store: columns instead of tables
- Key-value cells

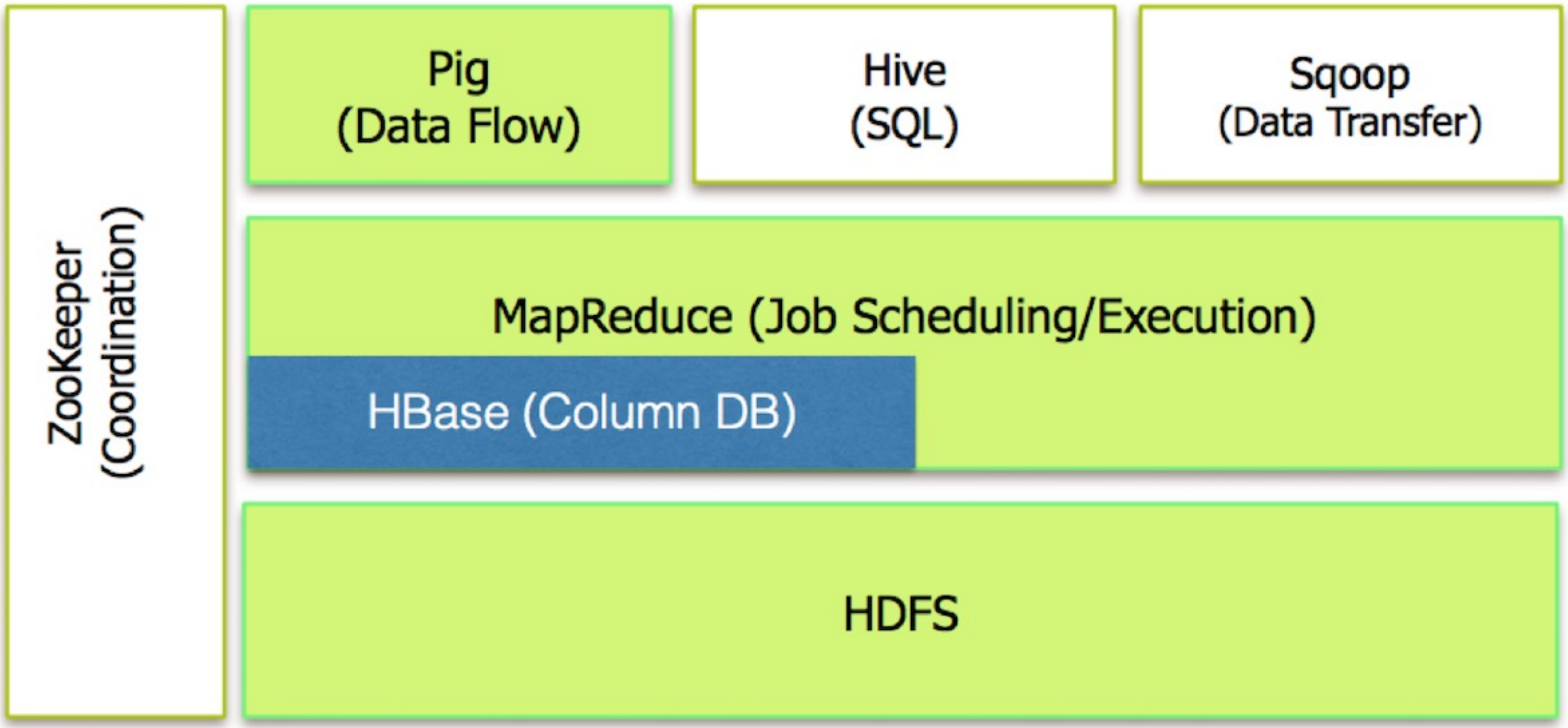
Apache HBase

HBase is not ACID compliant

"HBase is a **distributed column-oriented** *database* built on top of HDFS. HBase is the Hadoop application to use when you require **real-time** read/write **random** access to very **large** datasets." (Tom White)

"HBase tables are like those in an RDBMS, only cells are **versioned**, rows are **sorted**, and columns can be **added on the fly**…" (Tom White)

HBase in the Hadoop ecosystem



History of HBase

- Started at the end of **2006**
- Modelled after **Google's Bigtable paper** (2006)
- **January 2008:** Hadoop becomes Apache top level project, HBase becomes subproject
- **May 2010:** HBase becomes an Apache top level project
- Contributors from Cloudera, Facebook, Intel, Hortonworks, etc.

Demands for HBase

- **Structured data**, scaling to petabytes
- **Efficient** handling of **diverse** data
  - Wrt data size (URLs, web pages, satellite images)
  - Wrt latency (backend bulk processing vs. real-time data serving)
- **Efficient** read and write of **individual records**

HBase vs. Hadoop

- Hadoop’s use case is **batch processing**
  - Not suitable for a single record lookup
  - Not suitable for adding **small amounts** of data at all times
  - Not suitable for making **updates** to existing records
- HBase addresses Hadoop’s weaknesses
  - Provides fast lookup of **individual** records
  - Supports **insertion** of **single** records
  - Supports record **updating**
  - Not all columns are of interest to everyone; each client only wants a particular subset of columns (**column-based storage**)

HBase vs. Hadoop

HBase is built on top of HDFS!

	Hadoop	HBase
writing	file append only, no updates	random write, updating
reading	sequential	random read, small range scan, full scan
structured storage	up to the user	sparse column family data model

HBase vs. RDBMS

	RDBMS	HBase
	small to medium-volume applications	use when scaling up in terms of dataset size, read/write concurrency
schema	fixed	random write, updating
orientation	row-oriented	column-oriented
query language	SQL	simple data access model
size	terabytes (at most)	billions of rows, millions of columns
scaling up	difficult (workarounds)	add nodes to a cluster

Apache Hive

- Data warehousing infrastructure on top of Hadoop
  - Not OLTP!
- SQL-like interface to Map/Reduce jobs

- Data organisation
  - Databases: access control
  - Tables: data with the same schema
  - Partitions: logical partitioning based on expected queries (e.g., US sales during 2010-2015)

- More on Hive: https://vimeo.com/29732341

PIG

Pig vs. Pig Latin

- Pig: an **engine** for executing **data flows** in parallel on Hadoop
- **Pig Latin**: the high-level (SQL-like) language for expressing data flows
- Pig Latin contains common data processing operators (**join**, **sort**, **filter**, ...)
- **User defined functions** (UDFs): developers can write their own functions to read/process/store the data

O'Reilly® Programming Pig, by Alan Gates

Pig on Hadoop

- Makes use of **HDFS** and the **MapReduce core** of Hadoop
  - By default, reads input from & writes output to HDFS
- Pig Latin scripts are **compiled** into **one or more** Hadoop jobs which are executed in order
- Pig Latin users need **not** to be aware of the algorithmic details in the map/reduce phases

Pig Latin

- A parallel **dataflow language**: users describe **how** data is read, processed and stored
- Dataflows can be simple (e.g. “counting words”) or complex (multiple inputs are joined, data is split up into streams and processed separately)

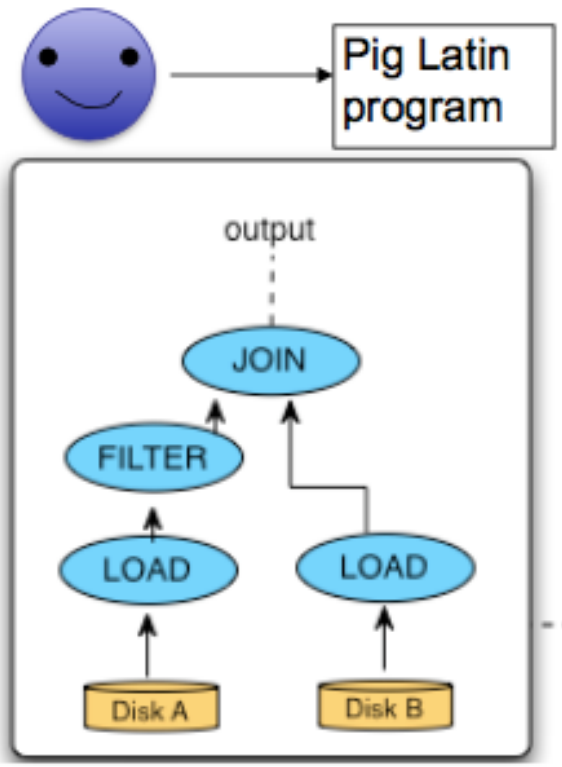
Pig vs SQL

Pig	SQL
Procedural: script describes **how** to process the data	Descriptive: query describes **what** the output should be
Workflows can contain many data processing operations	One query answers one question (*subqueries)
Schemas may be unknown or inconsistent	RDBMSs have defined schemas
Reads files from HDFS (and other sources)	Data is read from database tables

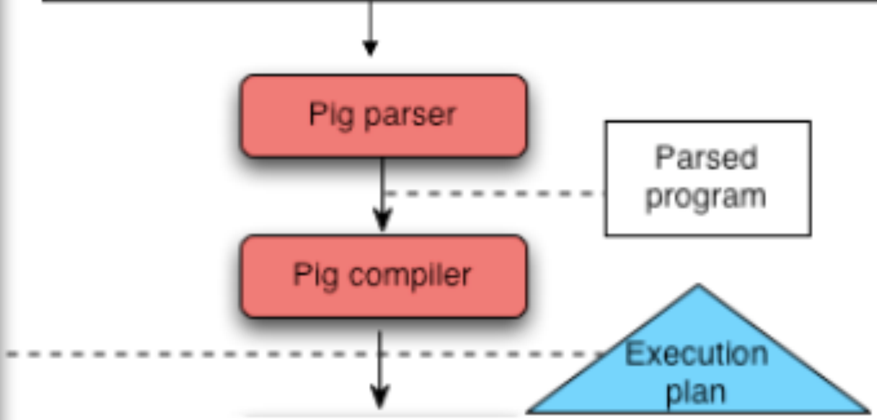
Pig - Word Count Example

```
-- read the file pg46.txt line by line, call each record line
cur = load 'pg46.txt' as (line);

-- tokenize each line, each term is now a record called word
words = foreach cur generate flatten(TOKENIZE(line)) as word;

-- group all words together by word
grpd  = group words by word;

-- count the words
cntd  = foreach grpd generate group, COUNT(words);

/*
 * start the Hadoop job and print results
 */
dump cntd;
```

Pig




```
A = LOAD 'file1' AS (sid,pid,mass,px:double);
B = LOAD 'file2' AS (sid,pid,mass,px:double);
C = FILTER A BY px < 1.0;
D = JOIN C BY sid,
         B BY sid;
  STORE D INTO 'output.txt';
```



PIG - Features

- PIG **handles erroneous**/corrupt **data** entries gracefully
  - Schema can be **inconsistent** or missing
  - (cleaning step can be skipped)
  - Exploratory analysis can be performed **quickly**
- PIG operates on any data (schema or not, files or not, nested or not)
- Parallel data processing language; implemented on Hadoop but not tied to it
- Easily controlled and modified
- Fast processing

Summary

- OLAP vs OLTP
- Architectures for Distributed Databases
- Big Table
- Hbase and Hive
- PIG

References

- **Bigtable: A Distributed Storage System for Structured Data.** Fay Chang et. al. OSDI 2006

- **Pig Latin: A Not-So-Foreign Language for Data Processing.** C. Olston, B. Reed, U. Srivastava, R. Kumar and A. Tomkins. SIGMOD 2008.

- O'Reilly® Programming Pig, by Alan Gates

Scenarios

- UQ Poll: apps.elearning.uq.edu.au/poll/55029

- Scenario Assumptions
  - What is the data we expect?
  - Who are the users? What do they know?
  - What are the queries we expect?
  - What is the timeline?

Scenarios - Which tool is best for which use case?

1. Data generated by the Large Hadron Collider is stored and analysed by researchers
2. All pages crawled from the Web are stored by a search engine and served to clients via its search interface
3. Data generated by the Australian taxation office is used to send warning letters to tax payers (“you are late with your taxes”)

- Options
  - A. Relational DBMS
  - B. Hadoop (HDFS, M/R, PIG, HBase etc.)
  - C. Spark (HDFS, RDDs, MLLib, etc.)
  - D. Streaming solution (Storm, Kafka, SparkStreaming, etc.)
  - E. Graph data solution (Pregel, Giraph, Spark GraphX, etc.)

What’s next

- Zookeeper
- Map/Reduce framework
  - Scheduling of jobs
- Apache Hadoop and Apache Spark
- Python, R, Notebooks