# Introduction to Data Science
# Lecture 3
# Manipulating Tabular Data

# File Formats

- Considerations for a file format
  - Data model: tabular, hierarchical, array
  - Physical layout
  - Field units and validation
  - Metadata: header, side file, specification, other?
  - Plain text or binary
  - Encoding: ASCII, UTF-8, other?
  - Delimiters and escaping
  - Compression, encryption, checksums?
  - Schema evolution

# File Performance

## Read/Write time (626 MB tabular file)

| | Read Time (Text) | Write Time (Text) | Read Time (Binary) | Write Time (Binary) |
|---|---|---|---|---|
| Pandas (Python) | 36 secs | 45 secs | ** | ** |
| Scala/Java | 18 secs | 21 secs | 1-6* secs | 1-6* secs |

**Read-Write Times Comparable**

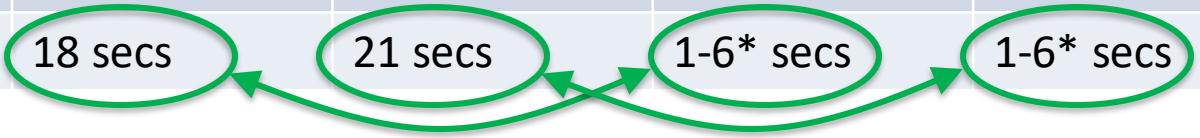
** Pandas doesn't have a default binary file I/O library –
you can use Python, but performance depends on what you pick.

* 6 seconds is the time for sustainable read/write.
Often faster due to caching

# File Performance

## Read/Write time (626 MB tabular file)

| | Read Time (Text) | Write Time (Text) | Read Time (Binary) | Write Time (Binary) |
|---|---|---|---|---|
| Pandas (Python) | 36 secs | 45 secs | ** | ** |
| Scala/Java | 18 secs | 21 secs | 1-6* secs | 1-6* secs |



**Binary I/O much faster than text**

** Pandas doesn't have a default binary file I/O library –
you can use Python, but performance depends on what you pick.

* 6 seconds is the time for sustainable read/write.
Often faster due to caching

# File Performance - Compression

## Read/Write time (626 MB tabular file, Scala/Java)

| Binary File | Read Time | Write Time | File Size |
|---|---|---|---|
| Gzip level 6 (Java default) | 4 secs | 75 secs | 286 MB |
| Gzip level 3 | 4 secs | 20 secs | 313 MB |
| Gzip level 1 | 4 secs | 14 secs | 328 MB |
| LZ4 fast | 2 secs | 4 secs | 423 MB |
| Raw binary file | 1-6 secs | 1-6 secs | 787 MB |

Write times much larger than read

| Text File | Read Time | Write Time | File Size |
|---|---|---|---|
| Gzip level 6 (default) | 26 secs | 98 secs | 243 MB |
| Gzip level 3 | 25 secs | 46 secs | 259 MB |
| Gzip level 1 | 25 secs | 33 secs | 281 MB |
| LZ4 fast | 22 secs | 24 secs | 423 MB |
| Raw text file | 18 secs | 21 secs | 626 MB |

# File Performance - Compression

## Read/Write time (626 MB tabular file, Scala/Java)

| Binary File | Read Time | Write Time | File Size |
|---|---|---|---|
| Gzip level 6 (Java default) | 4 secs | 75 secs | 286 MB |
| Gzip level 3 | 4 secs | 20 secs | 313 MB |
| Gzip level 1 | 4 secs | 14 secs | 328 MB |
| LZ4 fast | 2 secs | 4 secs | 423 MB |
| Raw binary file | 1-6 secs | 1-6 secs | 787 MB |

Large range of compression times

| Text File | Read Time | Write Time | File Size |
|---|---|---|---|
| Gzip level 6 (default) | 26 secs | 98 secs | 243 MB |
| Gzip level 3 | 25 secs | 46 secs | 259 MB |
| Gzip level 1 | 25 secs | 33 secs | 281 MB |
| LZ4 fast | 22 secs | 24 secs | 423 MB |
| Raw text file | 18 secs | 21 secs | 626 MB |

# File Performance - Compression

## Read/Write time (626 MB tabular file, Scala/Java)

| Binary File | Read Time | Write Time | File Size |
|---|---|---|---|
| Gzip level 6 (Java default) | 4 secs | 75 secs | 286 MB |
| Gzip level 3 | 4 secs | 20 secs | 313 MB |
| Gzip level 1 | 4 secs | 14 secs | 328 MB |
| LZ4 fast | 2 secs | 4 secs | 423 MB |
| Raw binary file | 1-6 secs | 1-6 secs | 787 MB |

Large range of compression times

| Text File | Read Time | Write Time | File Size |
|---|---|---|---|
| Gzip level 6 (default) | 26 secs | 98 secs | 243 MB |
| Gzip level 3 | 25 secs | 46 secs | 259 MB |
| Gzip level 1 | 25 secs | 33 secs | 281 MB |
| LZ4 fast | 22 secs | 24 secs | 423 MB |
| Raw text file | 18 secs | 21 secs | 626 MB |

# File Performance - Compression

## Read/Write time (626 MB tabular file, Scala/Java)

| Binary File | Read Time | Write Time | File Size |
|---|---|---|---|
| Gzip level 6 (Java default) | 4 secs | 75 secs | 286 MB |
| Gzip level 3 | 4 secs | 20 secs | 313 MB |
| Gzip level 1 | 4 secs | 14 secs | 328 MB |
| LZ4 fast | 2 secs | 4 secs | 423 MB |
| Raw binary file | 1-6 secs | 1-6 secs | 787 MB |

**Small range (15%) of compressed file sizes**

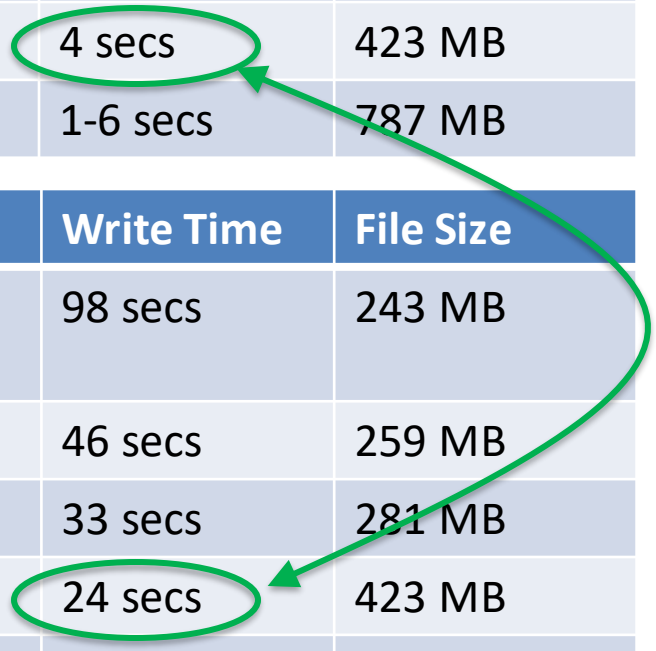
| Text File | Read Time | Write Time | File Size |
|---|---|---|---|
| Gzip level 6 (default) | 26 secs | 98 secs | 243 MB |
| Gzip level 3 | 25 secs | 46 secs | 259 MB |
| Gzip level 1 | 25 secs | 33 secs | 281 MB |
| LZ4 fast | 22 secs | 24 secs | 423 MB |
| Raw text file | 18 secs | 21 secs | 626 MB |

# File Performance - Compression

## Read/Write time (626 MB tabular file, Scala/Java)

| Binary File | Read Time | Write Time | File Size |
|---|---|---|---|
| Gzip level 6 (Java default) | 4 secs | 75 secs | 286 MB |
| Gzip level 3 | 4 secs | 20 secs | 313 MB |
| Gzip level 1 | 4 secs | 14 secs | 328 MB |
| LZ4 fast | 2 secs | 4 secs | 423 MB |
| Raw binary file | 1-6 secs | 1-6 secs | 787 MB |

| Text File | Read Time | Write Time | File Size |
|---|---|---|---|
| Gzip level 6 (default) | 26 secs | 98 secs | 243 MB |
| Gzip level 3 | 25 secs | 46 secs | 259 MB |
| Gzip level 1 | 25 secs | 33 secs | 281 MB |
| LZ4 fast | 22 secs | 24 secs | 423 MB |
| Raw text file | 18 secs | 21 secs | 626 MB |



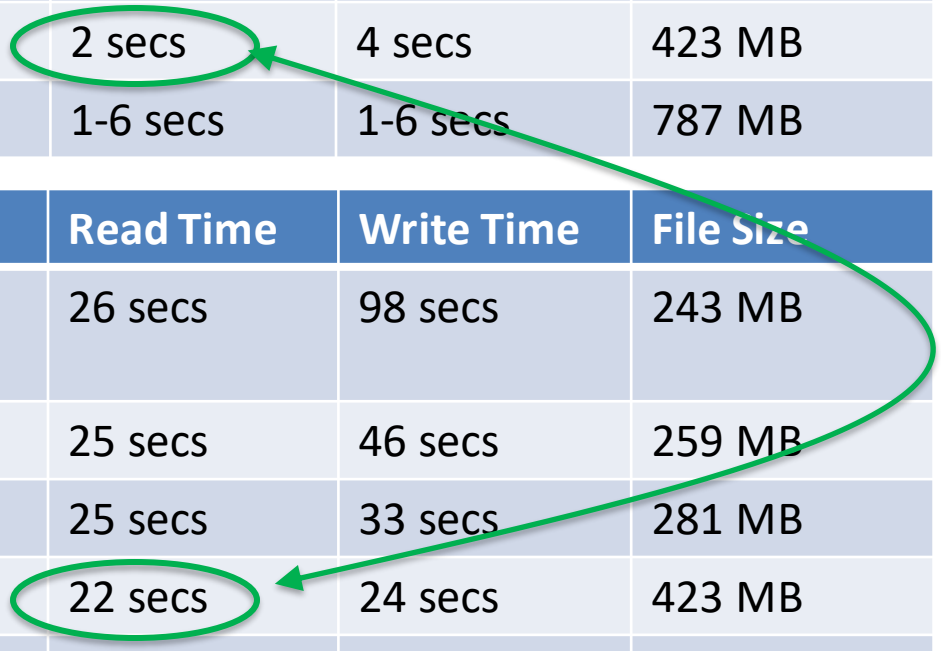
**Binary I/O still much faster than text**

# File Performance - Compression

## Read/Write time (626 MB tabular file, Scala/Java)

| Binary File | Read Time | Write Time | File Size |
|---|---|---|---|
| Gzip level 6 (Java default) | 4 secs | 75 secs | 286 MB |
| Gzip level 3 | 4 secs | 20 secs | 313 MB |
| Gzip level 1 | 4 secs | 14 secs | 328 MB |
| LZ4 fast | 2 secs | 4 secs | 423 MB |
| Raw binary file | 1-6 secs | 1-6 secs | 787 MB |

| Text File | Read Time | Write Time | File Size |
|---|---|---|---|
| Gzip level 6 (default) | 26 secs | 98 secs | 243 MB |
| Gzip level 3 | 25 secs | 46 secs | 259 MB |
| Gzip level 1 | 25 secs | 33 secs | 281 MB |
| LZ4 fast | 22 secs | 24 secs | 423 MB |
| Raw text file | 18 secs | 21 secs | 626 MB |



**Binary I/O still much faster than text**

# File Performance - Compression

## Read/Write time (626 MB tabular file, Scala/Java)

| Binary File | Read Time | Write Time | File Size |
|---|---|---|---|
| Gzip level 6 (Java default) | 4 secs | 75 secs | 286 MB |
| Gzip level 3 | 4 secs | 20 secs | 313 MB |
| Gzip level 1 | 4 secs | 14 secs | 328 MB |
| LZ4 fast | 2 secs | 4 secs | 423 MB |
| Raw binary file | 1-6 secs | 1-6 secs | 787 MB |

**LZ4 compression ≈ raw I/O speed**

| Text File | Read Time | Write Time | File Size |
|---|---|---|---|
| Gzip level 6 (default) | 26 secs | 98 secs | 243 MB |
| Gzip level 3 | 25 secs | 46 secs | 259 MB |
| Gzip level 1 | 25 secs | 33 secs | 281 MB |
| LZ4 fast | 22 secs | 24 secs | 423 MB |
| Raw text file | 18 secs | 21 secs | 626 MB |

# File Performance - Compression

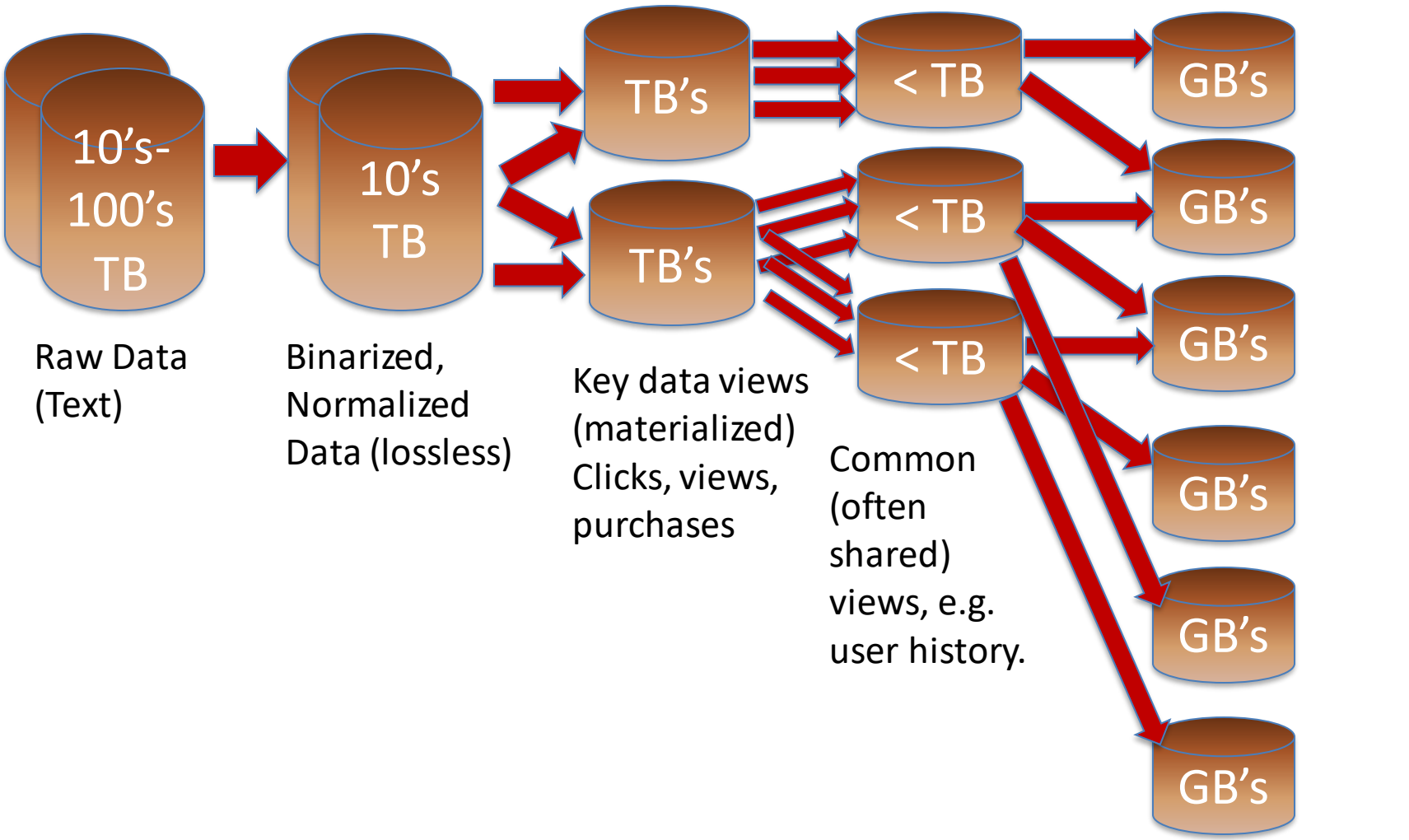
## Read/Write time (626 MB tabular file, Scala/Java)

| Binary File | Read Time | Write Time | File Size |
|---|---|---|---|
| Gzip level 6 (Java default) | 4 secs | 75 secs | 286 MB |
| Gzip level 3 | 4 secs | 20 secs | 313 MB |
| Gzip level 1 | 4 secs | 14 secs | 328 MB |
| LZ4 fast | 2 secs | 4 secs | 423 MB |
| Raw binary file | 1-6 secs | 1-6 secs | 787 MB |

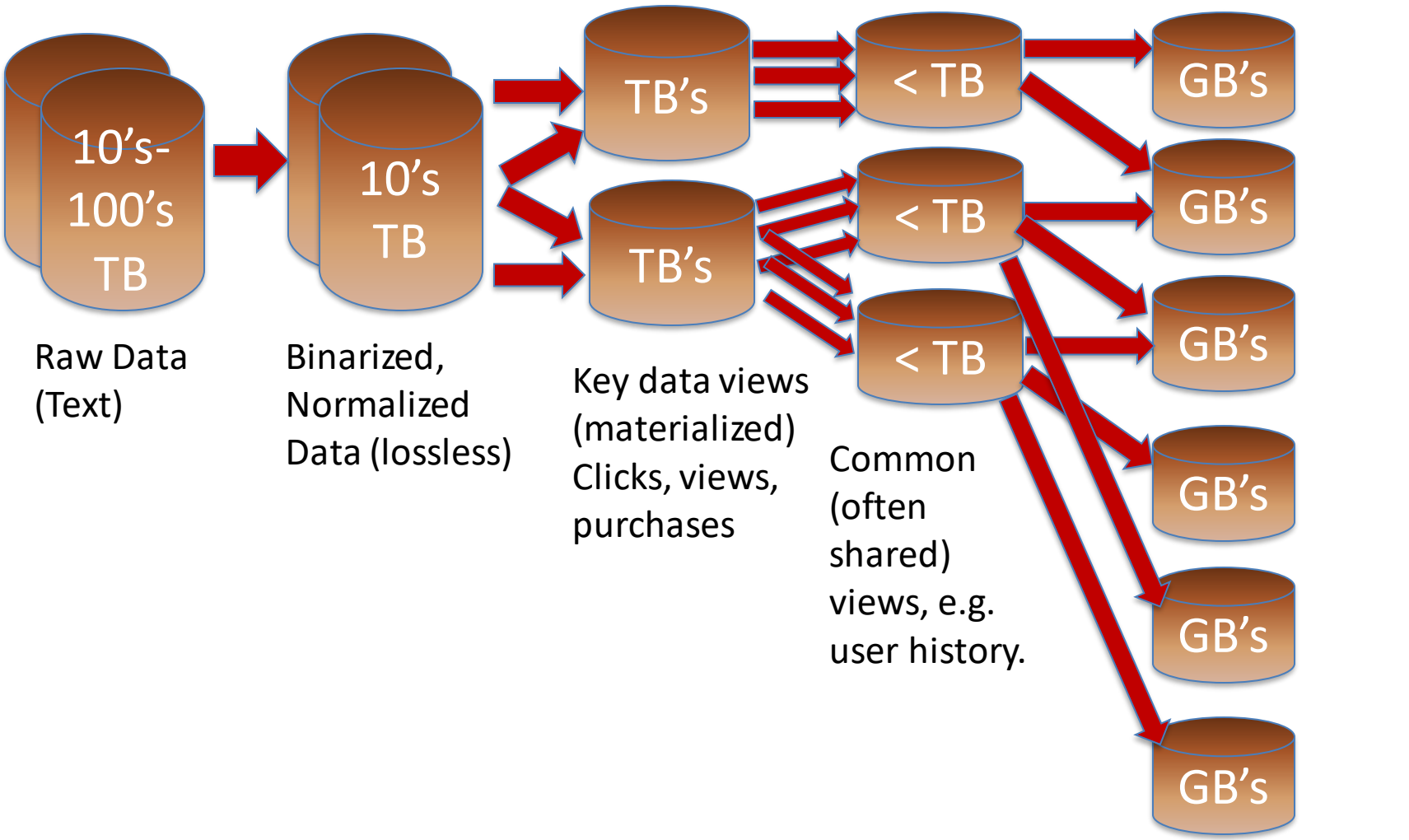
LZ4 compression ≈ raw I/O speed

| Text File | Read Time | Write Time | File Size |
|---|---|---|---|
| Gzip level 6 (default) | 26 secs | 98 secs | 243 MB |
| Gzip level 3 | 25 secs | 46 secs | 259 MB |
| Gzip level 1 | 25 secs | 33 secs | 281 MB |
| LZ4 fast | 22 secs | 24 secs | 423 MB |
| Raw text file | 18 secs | 21 secs | 626 MB |

# Data Pipeline Design

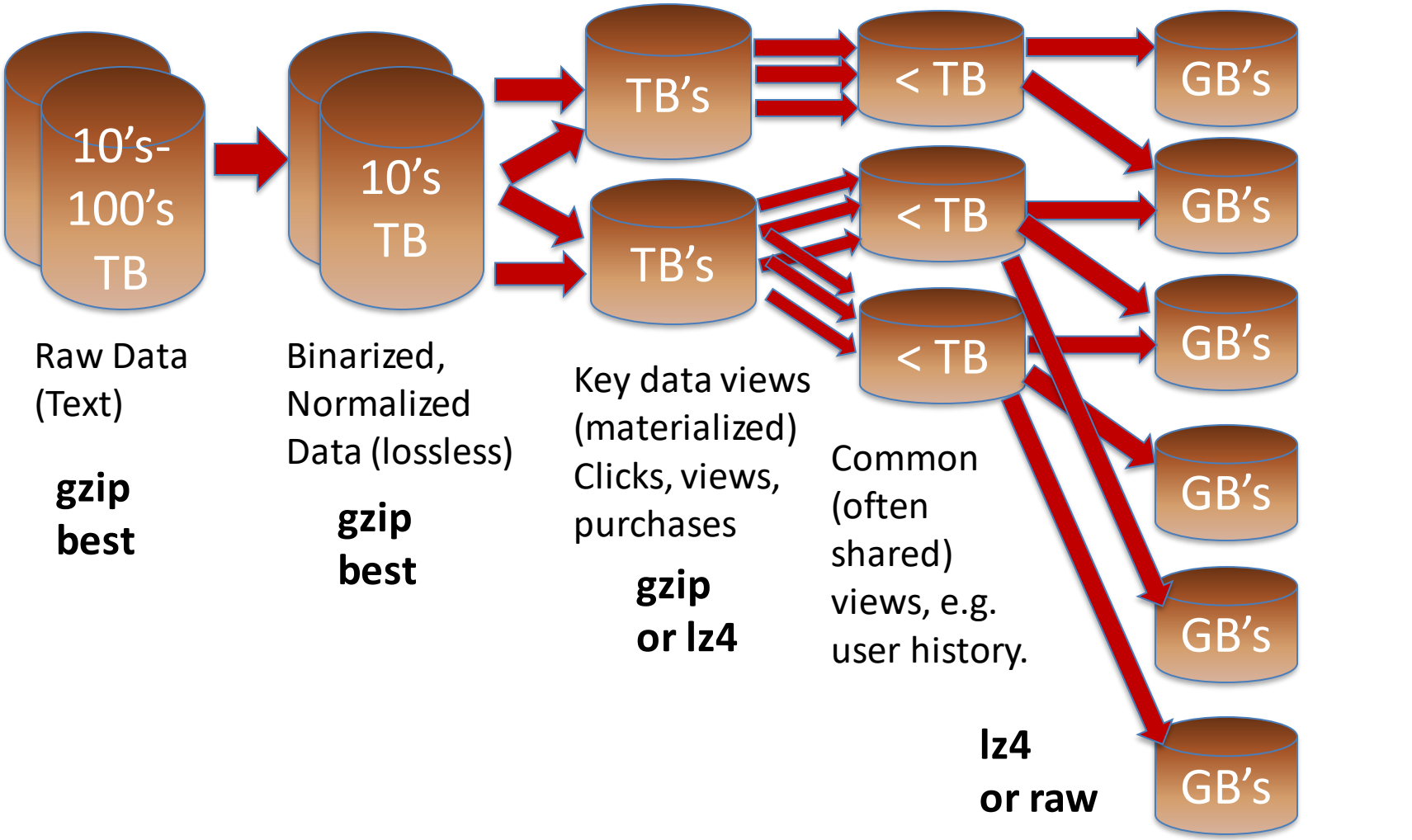


# Data Pipeline Design




Number of queries Increasing → Updates Increasing →

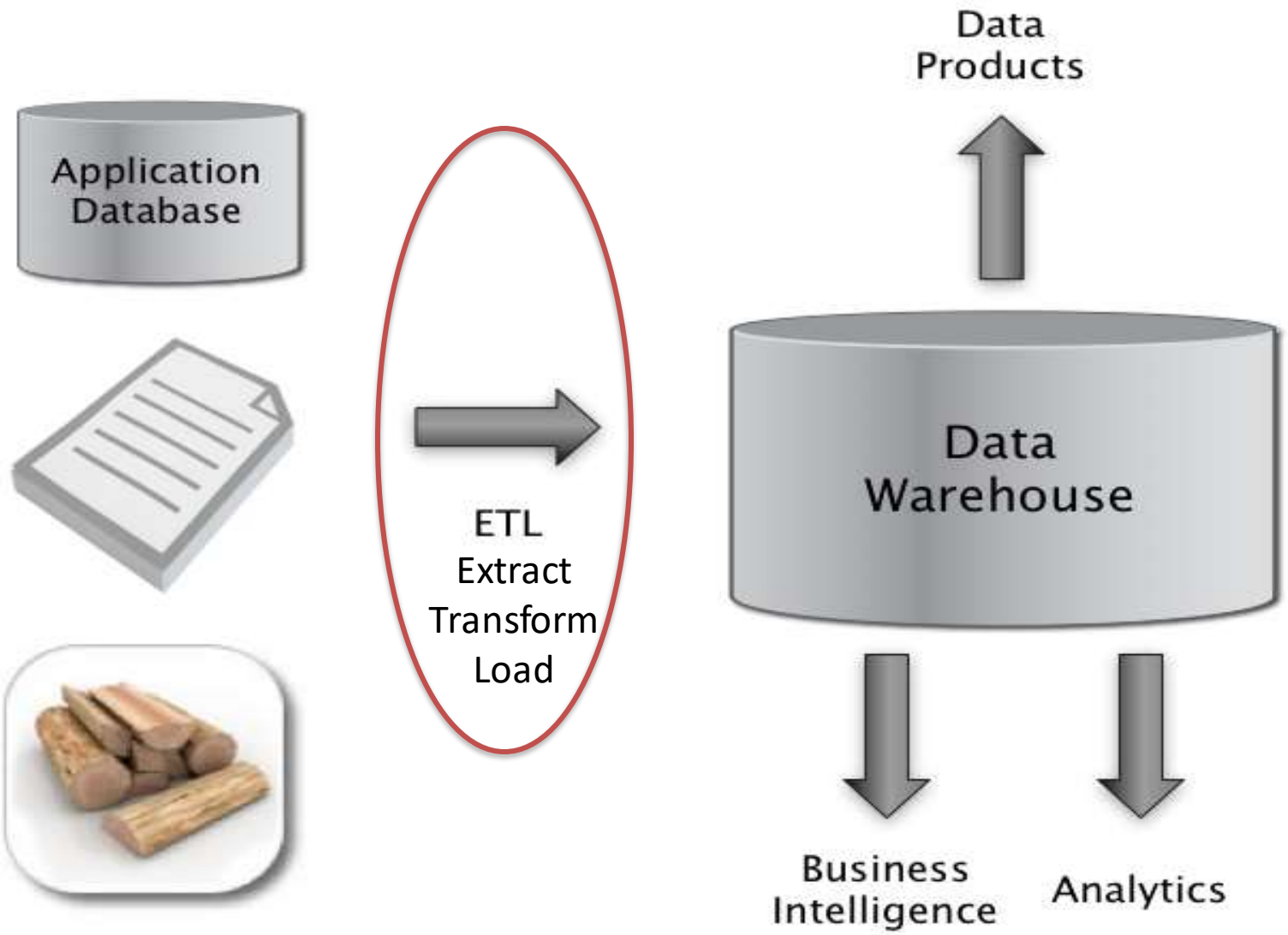
# Data Pipeline Design



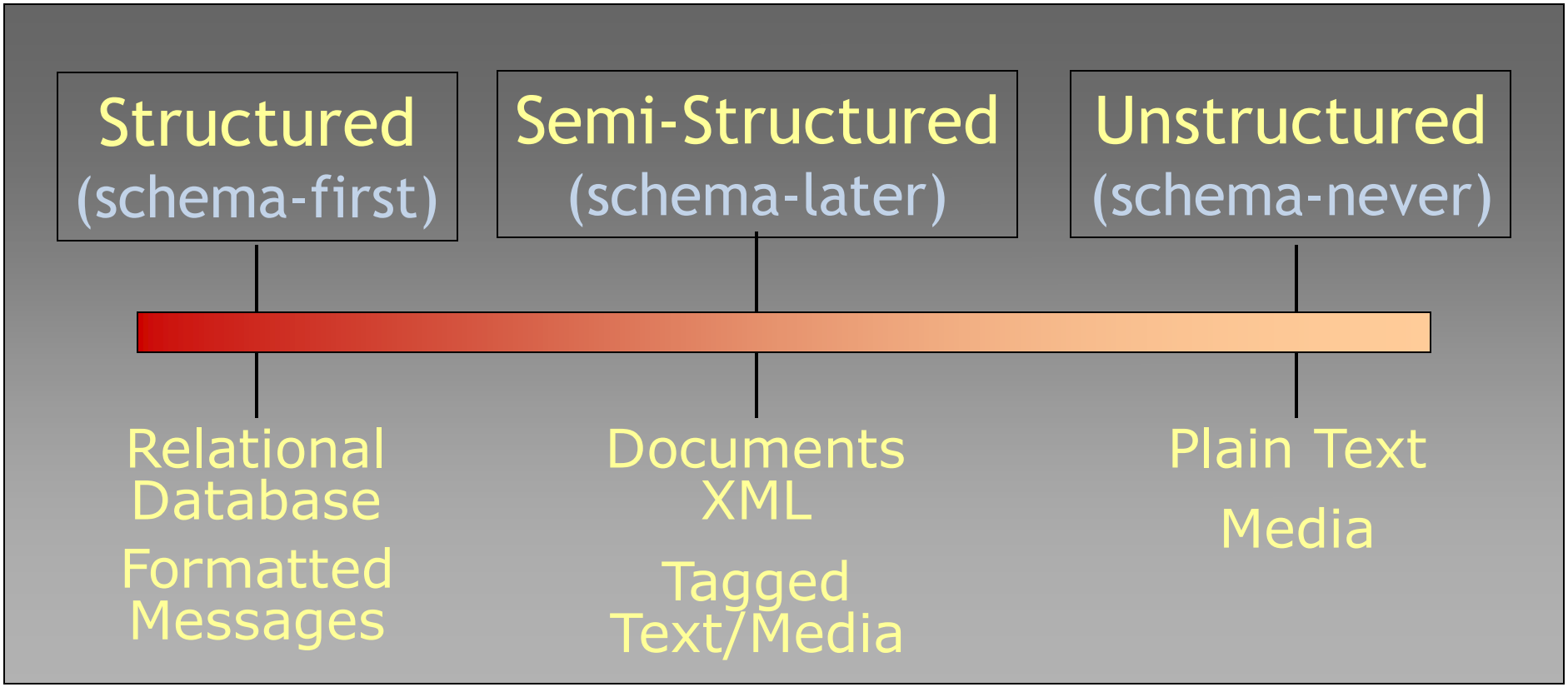
# Outline for this Evening

- Data Models, Tables, Structure, etc.
  - SQL
  - NoSQL
  - Schema on Read vs. Schema on Write

# The Big Picture



# The Structure Spectrum



# Key Concept: Structured Data

A *data model* is a collection of concepts for describing data.

A *schema* is a description of a particular collection of data, using a given data model.

# The Relational Model*

- The Relational Model is Ubiquitous:
  - MySQL, PostgreSQL, Oracle, DB2, SQLServer, …
  - Foundational work done at
    - IBM - System R
    - UC Berkeley - Ingres

E. F., “Ted” Codd
Turing Award 1981

- Object-oriented concepts have been merged in
  - Early work: POSTGRES research project at Berkeley
  - Informix, IBM DB2, Oracle 8i

- Also has support for XML (semi-structured data)

**Codd, E. F. (1970). "A relational model of data for large shared data banks". Communications of the ACM 13 (6): 37*

# Relational Database: Definitions

- *Relational database:* a set of *relations*
- *Relation:* made up of 2 parts:

  *Schema* : specifies name of relation, plus name and type of each column

  **Students(*sid*: string, *name*: string, *login*: string, *age*: integer, *gpa*: real)**

  *Instance* : the actual data at a given time
  - #rows = *cardinality*
  - #fields = *degree / arity*
- A relation is a mathematical object (from set theory) which is true for certain arguments.
- An instance defines the set of arguments for which the relation is true.

# Ex: Instance of Students Relation

| sid | name | login | age | gpa |
|---|---|---|---|---|
| 53666 | Jones | jones@cs | 18 | 3.4 |
| 53688 | Smith | smith@eecs | 18 | 3.2 |
| 53650 | Smith | smith @math | 19 | 3.8 |

- Cardinality = 3, arity = 5 , all rows distinct
- The relation is true for these tuples and false for others

# SQL - A language for Relational DBs*

- SQL = Structured Query Language
- Data Definition Language (DDL)
  - create, modify, delete relations
  - specify constraints
  - administer users, security, etc.
- Data Manipulation Language (DML)
  - Specify queries to find tuples that satisfy criteria
  - add, modify, remove tuples
- The DBMS is responsible for efficient evaluation.

* Developed at IBM by Donald D. Chamberlin and Raymond F. Boyce in the 1970s. Used to be *SEQUEL* (*Structured English QUEry Language*)

# Creating Relations in SQL

- Create the Students relation.
  - Note: the type (domain) of each field is specified, and enforced by the DBMS whenever tuples are added or modified.

```
CREATE TABLE Students
  (sid CHAR(20),
   name CHAR(20),
   login CHAR(10),
   age INTEGER,
   gpa FLOAT)
```

# Table Creation (continued)

- Another example: the Enrolled table holds information about courses students take.

```
CREATE TABLE Enrolled
   (sid CHAR(20),
    cid CHAR(20),
    grade CHAR(2))
```

# Adding and Deleting Tuples

- Can insert a single tuple using:

```
INSERT INTO Students (sid, name, login, age, gpa)
 VALUES ('53688', 'Smith', 'smith@ee', 18, 3.2)
```

- Can delete all tuples satisfying some condition (e.g., name = Smith):

```
DELETE
  FROM Students S
 WHERE S.name = 'Smith'
```

# Queries in SQL

- Single-table queries are straightforward.

- To find all 18 year old students, we can write:

```
SELECT *
  FROM Students S
 WHERE S.age=18
```

- To find just names and logins, replace the first line:

```
SELECT S.name, S.login
```

# Querying Multiple Relations

- Can specify a join over two tables as follows:

```
SELECT S.name, E.cid
 FROM Students S, Enrolled E
 WHERE S.sid=E.sid AND E.grade='B'
```

**S**

| sid | cid | grade |
|---|---|---|
| 53831 | Carnatic101 | C |
| 53831 | Reggae203 | B |
| 53650 | Topology112 | A |
| 53666 | History105 | B |

**E**

| sid | name | login | age | gpa |
|---|---|---|---|---|
| 53831 | Jones | jones@cs | 18 | 3.4 |
| 53831 | Smith | smith@ee | 18 | 3.2 |

result =

| S.name | E.cid |
|---|---|
| Jones | History105 |

Note: no referential integrity constraints have been used here.

# Basic SQL Query

```
SELECT    [DISTINCT] target-list
FROM      relation-list
WHERE     qualification
```

- *relation-list* : A list of relation names
  - possibly with a *range-variable* after each name
- *target-list* : A list of attributes of tables in *relation-list*
- *qualification* : Comparisons combined using AND, OR and NOT.
  - Comparisons are Attr *op* const or Attr1 *op* Attr2, where *op* is one of =≠<>≤≥
- *DISTINCT*: optional keyword indicating that the answer should not contain duplicates.
  - In SQL SELECT, the default is that duplicates are *not* eliminated! (Result is called a “multiset”)

# SQL Inner Joins

```
SELECT S.name, E.classid
 FROM Students S (INNER) JOIN Enrolled E
 ON S.sid=E.sid
```

**S**

| S.name | S.sid |
|---|---|
| Jones | 11111 |
| Smith | 22222 |
| Brown | 33333 |

**E**

| E.sid | E.classid |
|---|---|
| 11111 | History105 |
| 11111 | DataScience194 |
| 22222 | French150 |
| 44444 | English10 |

| S.name | E.classid |
|---|---|
| Jones | History105 |
| Jones | DataScience194 |
| Smith | French150 |

Note the previous version of this query (with no join keyword) is an “Implicit join”

# SQL Inner Joins

```
SELECT S.name, E.classid
 FROM Students S (INNER) JOIN Enrolled E
 ON S.sid=E.sid
```

**S**

| S.name | S.sid |
|---|---|
| Jones | 11111 |
| Smith | 22222 |
| Brown | 33333 |

**E**

| E.sid | E.classid |
|---|---|
| 11111 | History105 |
| 11111 | DataScience194 |
| 22222 | French150 |
| 44444 | English10 |

| S.name | E.classid |
|---|---|
| Jones | History105 |
| Jones | DataScience194 |
| Smith | French150 |

Unmatched keys

# What kind of Join is this?

```
SELECT S.name, E.classid
 FROM Students S ?? Enrolled E
 ON S.sid=E.sid
```

**S**

| S.name | S.sid |
|---|---|
| Jones | 11111 |
| Smith | 22222 |
| Brown | 33333 |

**E**

| E.sid | E.classid |
|---|---|
| 11111 | History105 |
| 11111 | DataScience194 |
| 22222 | French150 |
| 44444 | English10 |

| S.name | E.classid |
|---|---|
| Jones | History105 |
| Jones | DataScience194 |
| Smith | French150 |
| Brown | NULL |

# SQL Joins

```
SELECT S.name, E.classid
 FROM Students S LEFT OUTER JOIN Enrolled E
 ON S.sid=E.sid
```

**S**

| S.name | S.sid |
|---|---|
| Jones | 11111 |
| Smith | 22222 |
| Brown | 33333 |

**E**

| E.sid | E.classid |
|---|---|
| 11111 | History105 |
| 11111 | DataScience194 |
| 22222 | French150 |
| 44444 | English10 |

| S.name | E.classid |
|---|---|
| Jones | History105 |
| Jones | DataScience194 |
| Smith | French150 |
| Brown | NULL |

# What kind of Join is this?

```
SELECT S.name, E.classid
 FROM Students S ?? Enrolled E
 ON S.sid=E.sid
```

**S**

| S.name | S.sid |
|---|---|
| Jones | 11111 |
| Smith | 22222 |
| Brown | 33333 |

**E**

| E.sid | E.classid |
|---|---|
| 11111 | History105 |
| 11111 | DataScience194 |
| 22222 | French150 |
| 44444 | English10 |

| S.name | E.classid |
|---|---|
| Jones | History105 |
| Jones | DataScience194 |
| Smith | French150 |
| NULL | English10 |

# SQL Joins

```
SELECT S.name, E.classid
 FROM Students S RIGHT OUTER JOIN Enrolled E
 ON S.sid=E.sid
```

**S**

| S.name | S.sid |
|---|---|
| Jones | 11111 |
| Smith | 22222 |
| Brown | 33333 |

**E**

| E.sid | E.classid |
|---|---|
| 11111 | History105 |
| 11111 | DataScience194 |
| 22222 | French150 |
| 44444 | English10 |

| S.name | E.classid |
|---|---|
| Jones | History105 |
| Jones | DataScience194 |
| Smith | French150 |
| NULL | English10 |

# What kind of Join is this?

```
SELECT S.name, E.classid
  FROM Students S ?? Enrolled E
  ON S.sid=E.sid
```

**S**

| S.name | S.sid |
|---|---|
| Jones | 11111 |
| Smith | 22222 |
| Brown | 33333 |

**E**

| E.sid | E.classid |
|---|---|
| 11111 | History105 |
| 11111 | DataScience194 |
| 22222 | French150 |
| 44444 | English10 |

| S.name | E.classid |
|---|---|
| Jones | History105 |
| Smith | French150 |

# SQL Joins

```
SELECT S.name, E.classid
  FROM Students S LEFT SEMI JOIN Enrolled E
  ON S.sid=E.sid
```

**S**

| S.name | S.sid |
|---|---|
| Jones | 11111 |
| Smith | 22222 |
| Brown | 33333 |

**E**

| E.sid | E.classid |
|---|---|
| 11111 | History105 |
| 11111 | DataScience194 |
| 22222 | French150 |
| 44444 | English10 |

| S.name | E.classid |
|---|---|
| Jones | History105 |
| Smith | French150 |

# What kind of Join is this?

```
SELECT *
  FROM Students S ?? Enrolled E
```

**S**

| S.name | S.sid |
|---|---|
| Jones | 11111 |
| Smith | 22222 |

**E**

| E.sid | E.classid |
|---|---|
| 11111 | History105 |
| 11111 | DataScience194 |
| 22222 | French150 |

| S.name | S.sid | E.sid | E.classid |
|---|---|---|---|
| Jones | 11111 | 11111 | History105 |
| Jones | 11111 | 11111 | DataScience194 |
| Jones | 11111 | 22222 | French150 |
| Smith | 22222 | 11111 | History105 |
| Smith | 22222 | 11111 | DataScience194 |
| Smith | 22222 | 22222 | French150 |

# SQL Joins

```
SELECT *
  FROM Students S CROSS JOIN Enrolled E
```

**S**

| S.name | S.sid |
|---|---|
| Jones | 11111 |
| Smith | 22222 |

**E**

| E.sid | E.classid |
|---|---|
| 11111 | History105 |
| 11111 | DataScience194 |
| 22222 | French150 |

| S.name | S.sid | E.sid | E.classid |
|---|---|---|---|
| Jones | 11111 | 11111 | History105 |
| Jones | 11111 | 11111 | DataScience194 |
| Jones | 11111 | 22222 | French150 |
| Smith | 22222 | 11111 | History105 |
| Smith | 22222 | 11111 | DataScience194 |
| Smith | 22222 | 22222 | French150 |

# What kind of Join is this?

```
SELECT *
  FROM Students S, Enrolled E
  WHERE S.sid <= E.sid
```

**S**

| S.name | S.sid |
|---|---|
| Jones | 11111 |
| Smith | 22222 |

**E**

| E.sid | E.classid |
|---|---|
| 11111 | History105 |
| 11111 | DataScience194 |
| 22222 | French150 |

| S.name | S.sid | E.sid | E.classid |
|---|---|---|---|
| Jones | 11111 | 11111 | History105 |
| Jones | 11111 | 11111 | DataScience194 |
| Jones | 11111 | 22222 | French150 |
| Smith | 22222 | 22222 | French150 |

# Theta Joins

```
SELECT *
  FROM Students S, Enrolled E
  WHERE S.sid <= E.sid
```

**S**

| S.name | S.sid |
|---|---|
| Jones | 11111 |
| Smith | 22222 |

**E**

| E.sid | E.classid |
|---|---|
| 11111 | History105 |
| 11111 | DataScience194 |
| 22222 | French150 |

| S.name | S.sid | E.sid | E.classid |
|---|---|---|---|
| Jones | 11111 | 11111 | History105 |
| Jones | 11111 | 11111 | DataScience194 |
| Jones | 11111 | 22222 | French150 |
| Smith | 22222 | 22222 | French150 |

# OTHER “TABLE-LIKE” DATA MODELS

# Pandas/Python

- **Series**: a named, ordered dictionary
  - The keys of the dictionary are the **indexes**
  - Built on NumPy's **ndarray**
  - Values can be any Numpy data type object

- **DataFrame**: a table with named columns
  - Represented as a Dict (col_name -> series)
  - Each Series object represents a column

# Operations

- map() functions
- filter (apply predicate to rows)
- sort/group by
- aggregate: sum, count, average, max, min
- Pivot or reshape
- Relational:
  - union, intersection, difference, cartesian product (CROSS JOIN)
  - select/filter, project
  - join: natural join (INNER JOIN), theta join, semi-join, etc.
  - rename

# Matrices vs Databases

- Tools like Pandas give up some of the important safety features of RDBMS (e.g. ACID), but can be much faster.

Matrix multiply in SQL:

**A**

| row | col | value |
|---|---|---|
| 1 | 1 | 5.7 |
| 3 | 1 | 3.2 |
| 2 | 2 | -5 |

**B**

| row | col | value |
|---|---|---|
| 2 | 1 | 12.0 |
| 3 | 3 | 5.1 |

```
SELECT  A.row, B.col, SUM(A.value * B.value)
    FROM A JOIN B
    ON A.col = B.row
    GROUP BY A.row, B.col
```

You probably never want to do this, but the ***opposite*** direction (relational aggregate query → matrix mult.) can be very useful.

# What's Wrong with Tables?

- Too limited in structure?
- Too rigid?
- Too old fashioned?

# What's Wrong with (RDBMS) Tables?

- **Indices:** Typical RDBMS table storage is mostly indices
  - Can't afford this overhead for large datastores
- **Transactions:**
  - Safe state changes require journals etc., and are slow
- **Relations:**
  - Checking relations adds further overhead to updates
- **Sparse Data Support:**
  - RDBMS Tables are very wasteful when data is very sparse
  - Very sparse data is common in modern data stores
  - RDBMS tables might have dozens of columns, modern data stores might have many thousands.

# RDBMS tables – row based

Table:

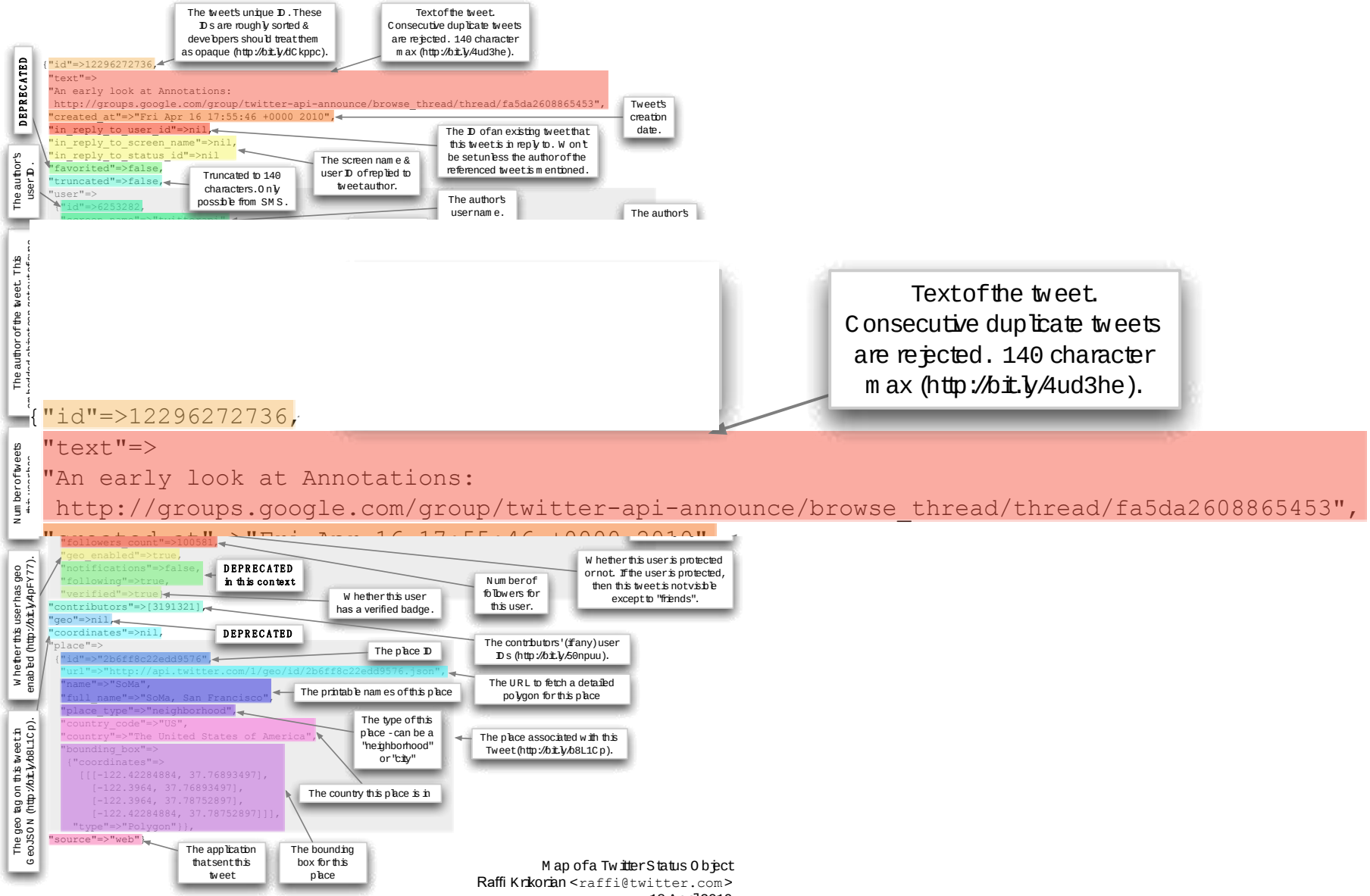
| sid | name | login | age | gpa |
|---|---|---|---|---|
| 53831 | Jones | jones@cs | 18 | 3.4 |
| 53831 | Smith | smith@ee | 18 | 3.2 |

Represented as:

| | | | | |
|---|---|---|---|---|
| 53831 | Jones | jones@cs | 18 | 3.4 |

| | | | | |
|---|---|---|---|---|
| 53831 | Smith | smith@ee | 18 | 3.2 |

# Tweet JSON Format




Map of a Twitter Status Object

Raffi Krikorian <raffi@twitter.com>

18 April 2010

# RDBMS tables – row based

Table:

| ID | name | login | loc | locid | LAT | LONG | ALT | State |
|---|---|---|---|---|---|---|---|---|
| 52841 | Jones | jones@cs | NULL | NULL | NULL | NULL | NULL | NULL |
| 53831 | Smith | smith@ee | NULL | NULL | NULL | NULL | NULL | NULL |
| 55541 | Brown | brown@ee | NULL | NULL | NULL | NULL | NULL | NULL |

Represented as:

| 52841 | Jones | jones@cs | NULL | NULL | NULL | NULL | NULL | NULL |
|---|---|---|---|---|---|---|---|---|

| 53831 | Smith | smith@ee | NULL | NULL | NULL | NULL | NULL | NULL |
|---|---|---|---|---|---|---|---|---|

| 55541 | Brown | brown@ee | NULL | NULL | NULL | NULL | NULL | NULL |
|---|---|---|---|---|---|---|---|---|

# Column-based store

Table:

| ID | name | login | loc | locid | LAT | LONG | ALT | State |
|---|---|---|---|---|---|---|---|---|
| 52841 | Jones | jones@cs | Albany | 2341 | 38.4 | 122.7 | 100 | CA |
| 53831 | Smith | smith@ee | NULL | NULL | NULL | NULL | NULL | NULL |
| 55541 | Brown | brown@ee | NULL | NULL | NULL | NULL | NULL | NULL |

Represented as column (key-value) stores:

| ID | name |
|---|---|
| 52841 | Jones |
| 53831 | Smith |
| 55541 | Brown |

| ID | login |
|---|---|
| 52841 | jones@cs |
| 53831 | smith@ee |
| 55541 | brown@ee |

| ID | loc |
|---|---|
| 52841 | Albany |

| ID | locid |
|---|---|
| 52841 | 2341 |

| ID | LAT |
|---|---|
| 52841 | 38.4 |

| ID | LONG |
|---|---|
| 52841 | 122.7 |

...


# BEYOND TABLES

# NoSQL Storage Systems

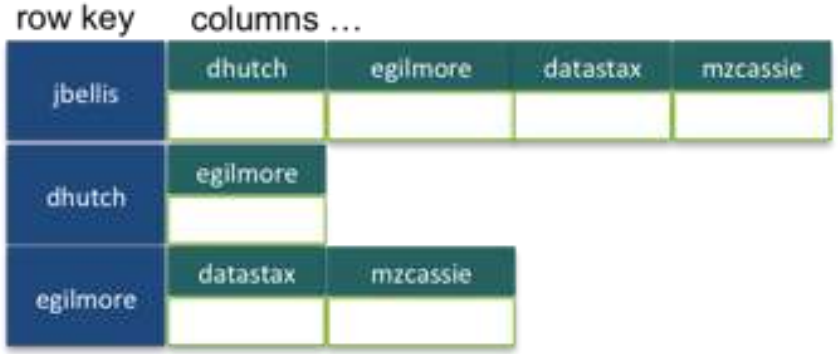

| | Data Model |
|---|---|
| Cassandra | Columnfamily |
| CouchDB | Document |
| HBase | Columnfamily |
| MongoDB | Document |
| Neo4J | Graph |
| Redis | Collection |
| Riak | Document |
| Scalaris | Key/value |
| Tokyo Cabinet | Key/value |
| Voldemort | Key/value |

# Column-Family Stores (Cassandra)

A column-family groups data columns together, and is analogous to a table.

Static column family from Apache Cassandra:


Dynamic Column family (Cassandra):



# CouchDB Data Model (JSON)

- "With CouchDB, no schema is enforced, so new document types with new meaning can be safely added alongside the old."
- A CouchDB document is an object that consists of named fields. Field values may be:
  - strings, numbers, dates,
  - ordered lists, associative maps

```
"Subject": "I like Plankton"
"Author": "Rusty"
"PostedDate": "5/23/2006"
"Tags": ["plankton", "baseball", "decisions"]
"Body": "I decided today that I don't like baseball. I like plankton."
```

# Prerequisites for “Schemaless” DBs

- Need **external** and **internal** representations for all data types that will be used.
- **Internal:** a dynamically-typed, object-oriented language (like Java)
- **External:** an extensible data description language: JSON or XML

- **For Performance:** Fast SerDe (Serialization and DeSerialization) so internal data structures can be efficiently pushed or extracted from disk or network.

# JSON format


```
{ "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
   "age": 25,
   "height_cm": 167.6,
   "address": {
       "streetAddress": "21 2nd Street",
       "city": "New York",
       "state": "NY",
       "postalCode": "10021-3100"
    }
}
```

# Prerequisites for “Schemaless” DBs

- JSON includes named fields in a tree structure. Primitive types (e.g. string, number, boolean,…) are implicit.

- We can read JSON data (or XML) and automatically create internal representations for complex data.

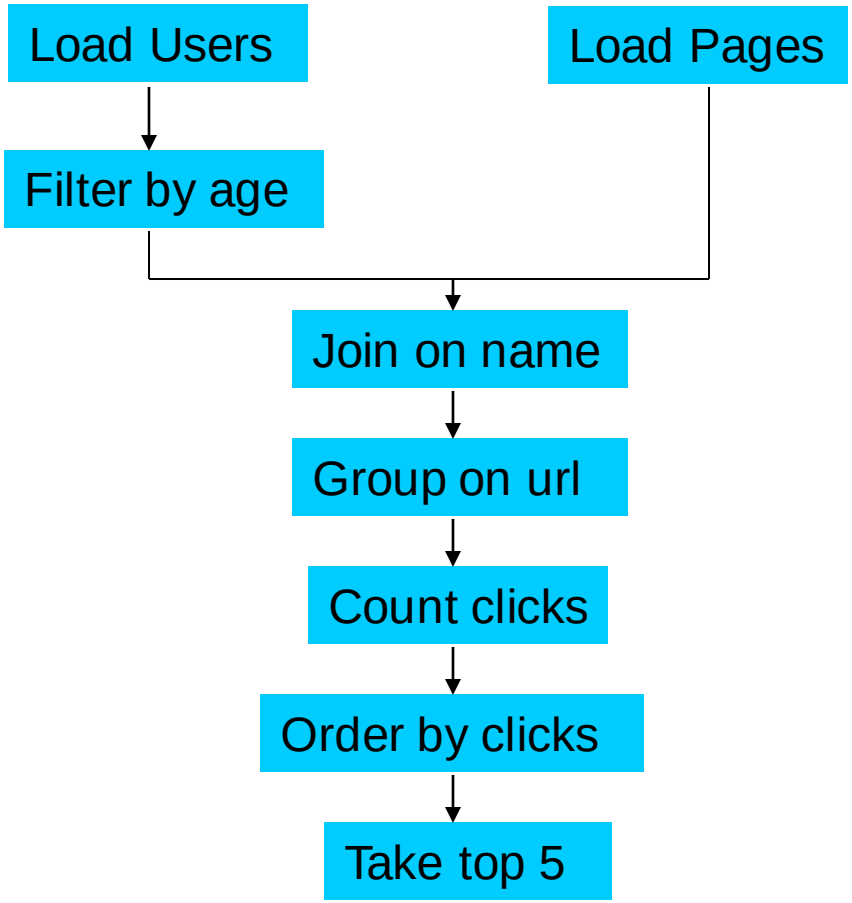
- Using the field names and object structure, we can query these objects once loaded.

# Pig

- Started at Yahoo! Research
- Runs about 50% of Yahoo!'s jobs
- Features:
  - Expresses sequences of MapReduce jobs
  - Data model: nested "bags" of items
    - Schema is optional
  - Provides relational (SQL) operators (JOIN, GROUP BY, etc)
  - Easy to plug in Java functions

# An Example Problem

Suppose you have user data in one file, website data in another, and you need to find the top 5 most visited pages by users aged 18-25.




Example from http://wiki.apache.org/pig-data/attachments/PigTalksPapers/attachments/ApacheConEurope09.ppt

# In MapReduce

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.Writable;
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.KeyValueTextInputFormat;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.RecordReader;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.mapred.SequenceFileInputFormat;
import org.apache.hadoop.mapred.SequenceFileOutputFormat;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.jobcontrol.Job;
import org.apache.hadoop.mapred.jobcontrol.JobControl;
import org.apache.hadoop.mapred.lib.IdentityMapper;

public class MRExample {
    public static class LoadPages extends MapReduceBase
        implements Mapper<LongWritable, Text, Text, Text> {

        public void map(LongWritable k, Text val,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // Pull the key out
            String line = val.toString();
            int firstComma = line.indexOf(',');
            String key = line.substring(0, firstComma);
            String value = line.substring(firstComma + 1);
            Text outKey = new Text(key);
            // Prepend an index to the value so we know which file
            // it came from.
            Text outVal = new Text("1" + value);
            oc.collect(outKey, outVal);
        }
    }
    public static class LoadAndFilterUsers extends MapReduceBase
        implements Mapper<LongWritable, Text, Text, Text> {

        public void map(LongWritable k, Text val,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // Pull the key out
            String line = val.toString();
            int firstComma = line.indexOf(',');
            String value = line.substring(firstComma + 1);
            int age = Integer.parseInt(value);
            if (age < 18 || age > 25) return;
            String key = line.substring(0, firstComma);
            Text outKey = new Text(key);
            // Prepend an index to the value so we know which file
            // it came from.
            Text outVal = new Text("2" + value);
            oc.collect(outKey, outVal);
        }
    }
    public static class Join extends MapReduceBase
        implements Reducer<Text, Text, Text, Text> {

        public void reduce(Text key,
                Iterator<Text> iter,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // For each value, figure out which file it's from and
store it
            // accordingly.
            List<String> first = new ArrayList<String>();
            List<String> second = new ArrayList<String>();

            while (iter.hasNext()) {
                Text t = iter.next();
                String value = t.toString();
                if (value.charAt(0) == '1')
first.add(value.substring(1));
                else second.add(value.substring(1));
                reporter.setStatus("OK");
            }

            // Do the cross product and collect the values
            for (String s1 : first) {
                for (String s2 : second) {
                    String outval = key + "," + s1 + "," + s2;
                    oc.collect(null, new Text(outval));
                    reporter.setStatus("OK");
                }
            }
        }
    }
    public static class LoadJoined extends MapReduceBase
        implements Mapper<Text, Text, Text, LongWritable> {

        public void map(
                Text k,
                Text val,
                OutputCollector<Text, LongWritable> oc,
                Reporter reporter) throws IOException {
            // Find the url
            String line = val.toString();
            int firstComma = line.indexOf(',');
            int secondComma = line.indexOf(',', firstComma);
            String key = line.substring(firstComma, secondComma);
            // drop the rest of the record, I don't need it anymore,
            // just pass a 1 for the combiner/reducer to sum instead.
            Text outKey = new Text(key);
            oc.collect(outKey, new LongWritable(1L));
        }
    }
    public static class ReduceUrls extends MapReduceBase
        implements Reducer<Text, LongWritable, WritableComparable,
Writable> {

        public void reduce(
                Text key,
                Iterator<LongWritable> iter,
                OutputCollector<WritableComparable, Writable> oc,
                Reporter reporter) throws IOException {
            // Add up all the values we see

            long sum = 0;
            while (iter.hasNext()) {
                sum += iter.next().get();
                reporter.setStatus("OK");
            }

            oc.collect(key, new LongWritable(sum));
        }
    }
    public static class LoadClicks extends MapReduceBase
        implements Mapper<WritableComparable, Writable, LongWritable,
Text> {

        public void map(
                WritableComparable key,
                Writable val,
                OutputCollector<LongWritable, Text> oc,
                Reporter reporter) throws IOException {
            oc.collect((LongWritable)val, (Text)key);
        }
    }
    public static class LimitClicks extends MapReduceBase
        implements Reducer<LongWritable, Text, LongWritable, Text> {

        int count = 0;
        public void reduce(
            LongWritable key,
            Iterator<Text> iter,
            OutputCollector<LongWritable, Text> oc,
            Reporter reporter) throws IOException {

            // Only output the first 100 records
            while (count < 100 && iter.hasNext()) {
                oc.collect(key, iter.next());
                count++;
            }
        }
    }
    public static void main(String[] args) throws IOException {
        JobConf lp = new JobConf(MRExample.class);
        lp.setJobName("Load Pages");
        lp.setInputFormat(TextInputFormat.class);
        lp.setOutputKeyClass(Text.class);
        lp.setOutputValueClass(Text.class);
        lp.setMapperClass(LoadPages.class);
        FileInputFormat.addInputPath(lp, new
Path("/user/gates/pages"));
        FileOutputFormat.setOutputPath(lp,
            new Path("/user/gates/tmp/indexed_pages"));
        lp.setNumReduceTasks(0);
        Job loadPages = new Job(lp);

        JobConf lfu = new JobConf(MRExample.class);
        lfu.setJobName("Load and Filter Users");
        lfu.setInputFormat(TextInputFormat.class);
        lfu.setOutputKeyClass(Text.class);
        lfu.setOutputValueClass(Text.class);
        lfu.setMapperClass(LoadAndFilterUsers.class);
        FileInputFormat.addInputPath(lfu, new
Path("/user/gates/users"));
        FileOutputFormat.setOutputPath(lfu,
            new Path("/user/gates/tmp/filtered_users"));
        lfu.setNumReduceTasks(0);
        Job loadUsers = new Job(lfu);

        JobConf join = new JobConf(MRExample.class);
        join.setJobName("Join Users and Pages");
        join.setInputFormat(KeyValueTextInputFormat.class);
        join.setOutputKeyClass(Text.class);
        join.setOutputValueClass(Text.class);
        join.setMapperClass(IdentityMapper.class);
        join.setReducerClass(Join.class);
        FileInputFormat.addInputPath(join, new
Path("/user/gates/tmp/indexed_pages"));
        FileInputFormat.addInputPath(join, new
Path("/user/gates/tmp/filtered_users"));
        FileOutputFormat.setOutputPath(join, new
Path("/user/gates/tmp/joined"));
        join.setNumReduceTasks(50);
        Job joinJob = new Job(join);
        joinJob.addDependingJob(loadPages);
        joinJob.addDependingJob(loadUsers);

        JobConf group = new JobConf(MRExample.class);
        group.setJobName("Group URLs");
        group.setInputFormat(KeyValueTextInputFormat.class);
        group.setOutputKeyClass(Text.class);
        group.setOutputValueClass(LongWritable.class);
        group.setOutputFormat(SequenceFileOutputFormat.class);
        group.setMapperClass(LoadJoined.class);
        group.setCombinerClass(ReduceUrls.class);
        group.setReducerClass(ReduceUrls.class);
        FileInputFormat.addInputPath(group, new
Path("/user/gates/tmp/joined"));
        FileOutputFormat.setOutputPath(group, new
Path("/user/gates/tmp/grouped"));
        group.setNumReduceTasks(50);
        Job groupJob = new Job(group);
        groupJob.addDependingJob(joinJob);

        JobConf top100 = new JobConf(MRExample.class);
        top100.setJobName("Top 100 sites");
        top100.setInputFormat(SequenceFileInputFormat.class);
        top100.setOutputKeyClass(LongWritable.class);
        top100.setOutputValueClass(Text.class);
        top100.setOutputFormat(SequenceFileOutputFormat.class);
        top100.setMapperClass(LoadClicks.class);
        top100.setCombinerClass(LimitClicks.class);
        top100.setReducerClass(LimitClicks.class);
        FileInputFormat.addInputPath(top100, new
Path("/user/gates/tmp/grouped"));
        FileOutputFormat.setOutputPath(top100, new
Path("/user/gates/top100sitesforusers18to25"));
        top100.setNumReduceTasks(1);
        Job limit = new Job(top100);
        limit.addDependingJob(groupJob);

        JobControl jc = new JobControl("Find top 100 sites for users
18 to 25");
        jc.addJob(loadPages);
        jc.addJob(loadUsers);
        jc.addJob(joinJob);
        jc.addJob(groupJob);
        jc.addJob(limit);
        jc.run();
    }
}
```

Example from http://wiki.apache.org/pig-data/attachments/PigTalksPapers/attachments/ApacheConEurope09.ppt

# In Pig Latin

```
Users    = load ‘users’ as (name, age);
Filtered = filter Users by
                  age >= 18 and age <= 25;
Pages    = load ‘pages’ as (user, url);
Joined   = join Filtered by name, Pages by user;
Grouped  = group Joined by url;
Summed   = foreach Grouped generate group,
                   count(Joined) as clicks;
Sorted   = order Summed by clicks desc;
Top5     = limit Sorted 5;

store Top5 into ‘top5sites’;
```

Example from http://wiki.apache.org/pig-data/attachments/PigTalksPapers/attachments/ApacheConEurope09.ppt

# Hive

- Developed at Facebook
- Relational database built on Hadoop
  - Maintains table schemas
  - SQL-like query language (which can also call Hadoop Streaming scripts)
  - Supports table partitioning, complex data types, sampling, some query optimization
- Used for most Facebook jobs
  - Less than 1% of daily jobs at Facebook use MapReduce directly!!! (SQL – or PIG – wins!)
  - Note: Google also has several SQL-like systems in use.

# DATASPACES – WHAT ARE THEY?

# Dataspaces

Inclusive

Deal with all the data of interest – in whatever form

Co-existence not Integration

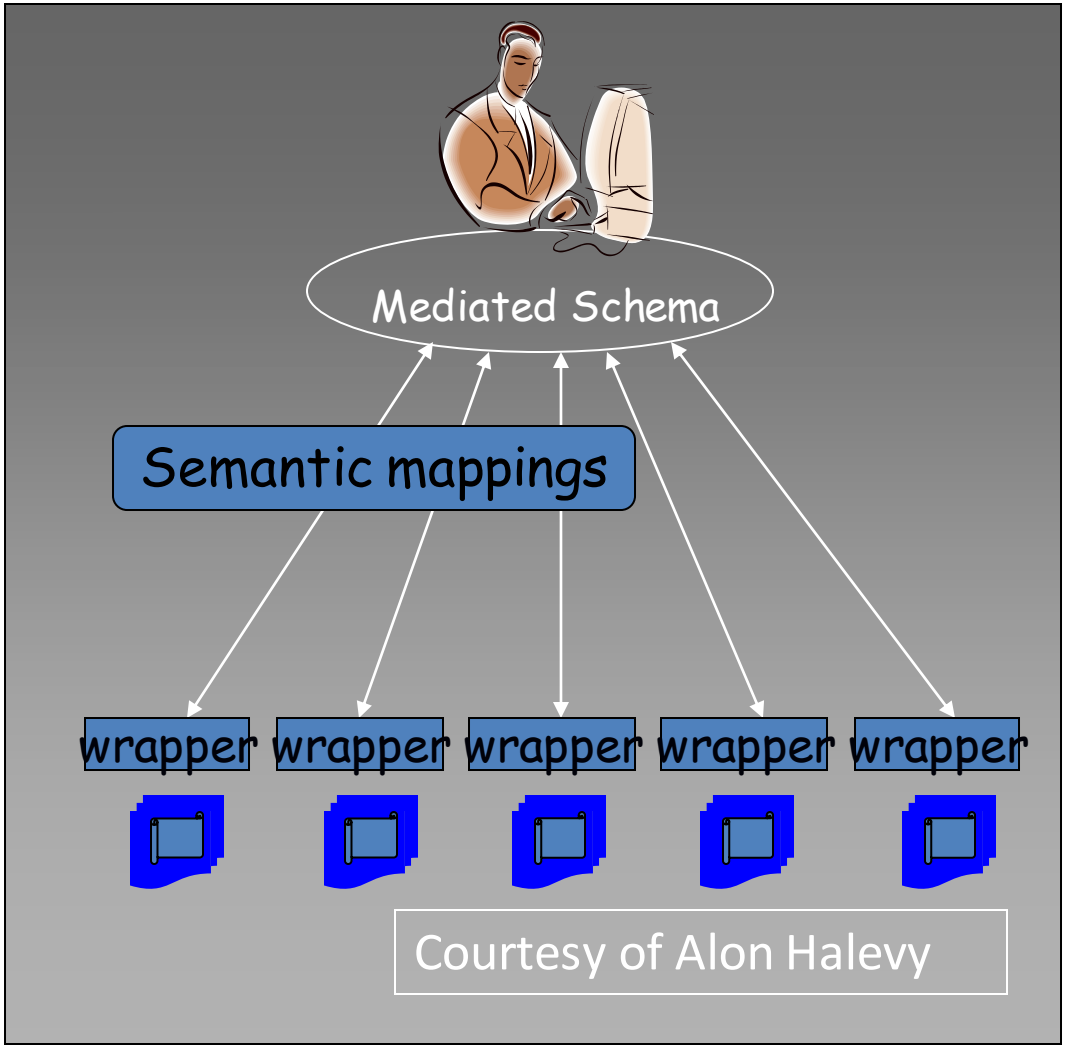
No integrated schema, no single warehouse, no ownership required

Pay-as-you-go

- Keyword search is bare minimum.
- More function and increased consistency as you add work.

# Compare to Data Integration

A quintessential schema-first approach.

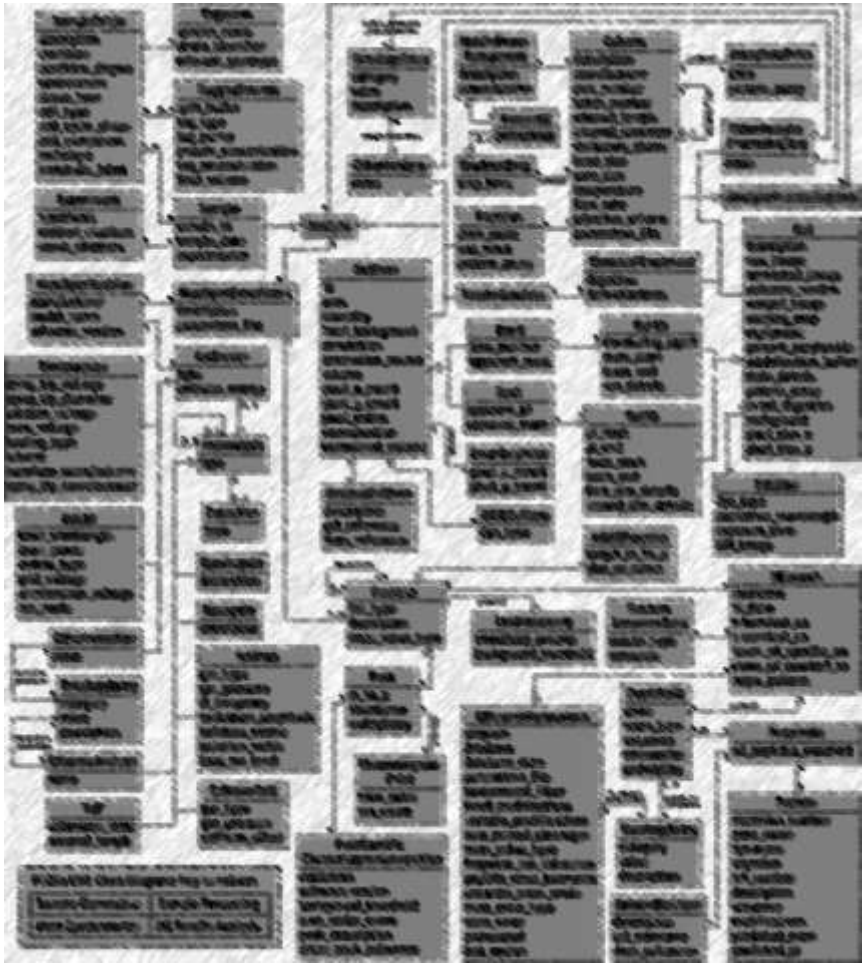


# Whither Structured Data?

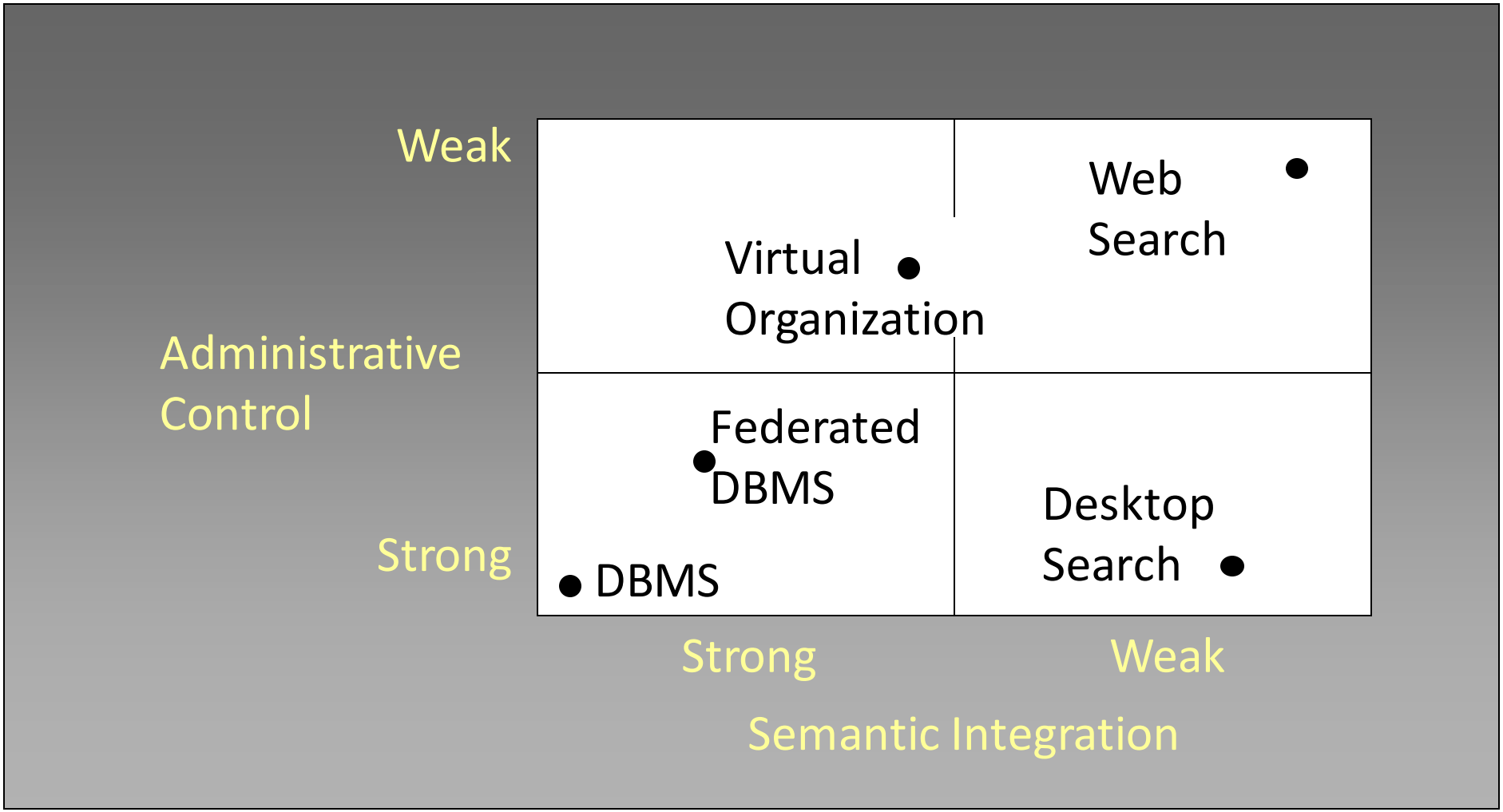
- Conventional Wisdom:

  only 20% of data is structured.

- Decreasing due to:
  - Consumer applications
  - Enterprise search
  - Media applications

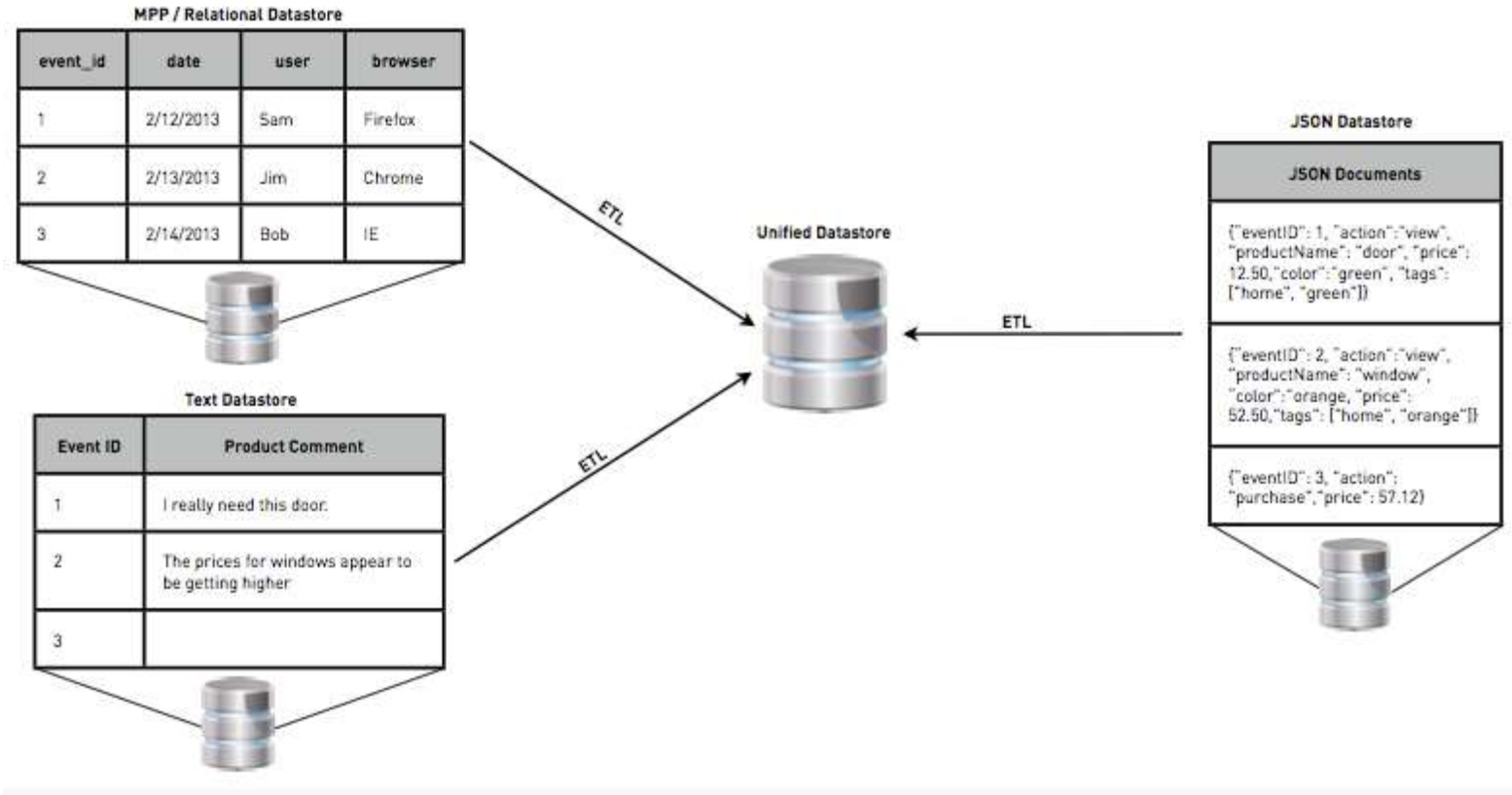


# An Alternative View



# A Recent Example

- Hadapt's "Schemaless SQL"



# A Recent Example

- Hadapt's "Schemaless SQL"

**actions**

| date | user | browser | json | product_comment |
|---|---|---|---|---|
| 2/12/2013 | Sam | Firefox | {"eventID": 1, "action":"view", "productName": "door", "price": 12.50,"color":"green", "tags": ["home", "green"]} | I really need this door. |
| 2/13/2013 | Jim | Chrome | {"eventID": 2, "action":"view", "productName": "window", "color":"orange, "price": 52.50,"tags": ["home", "orange"]} | The prices for windows appear to be getting higher. |
| 2/14/2013 | Bob | IE | {"eventID": 3, "action": "purchase","price": 57.12} | |

**actions**

| date | user | browser | product name | price | tags | color | action | product comment |
|---|---|---|---|---|---|---|---|---|
| 2/12/2013 | Sam | Firefox | door | 12.50 | home green | green | view | I really need this door. |
| 2/13/2013 | Jim | Chrome | window | 52.50 | home orange | orange | view | The prices for windows appear to be going higher. |
| 2/14/2013 | Bob | IE | | 57.12 | | | purchase | |

# “Schemaless SQL”

- Schema Evolution – adding a column

| date | user | browser | json | product_comment |
|---|---|---|---|---|
| 2/15/2013 | Jim | Firefox | {"eventID" 4,"action":"view","productName":"door","price": 12.50,"color":"green","tags":["home","green"],"size":"standard"} | I need a standard sized door. |

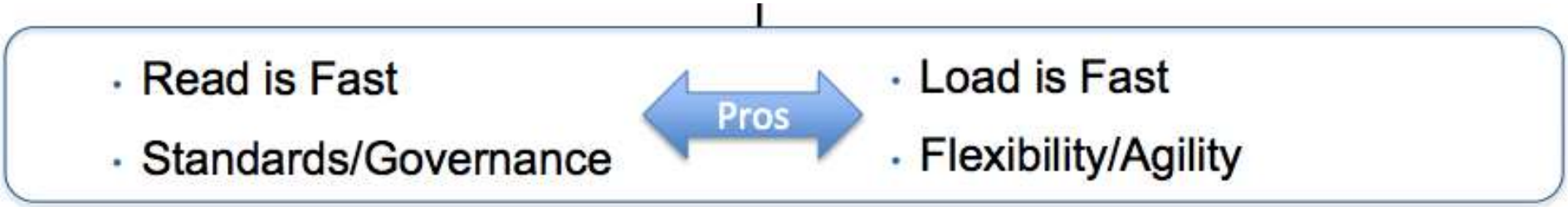
```
SELECT user, product_name , size
FROM actions
WHERE action='view'
AND date > 2/1/2013
AND matches('product_comment:higher')
```

actions

| date | user | browser | product name | price | tags | color | action | product comment | size |
|---|---|---|---|---|---|---|---|---|---|
| 2/12/2013 | Sam | Firefox | door | 12.50 | home green | green | view | I really need this door. | |
| 2/13/2013 | Jim | Chrome | window | 52.50 | home orange | orange | view | The prices for windows appear to be going higher. | |
| 2/14/2013 | Bob | IE | | 57.12 | | | purchase | | |
| 2/15/2013 | Jim | Firefox | door | 12.50 | home green | green | view | I need a standard sized door. | standard |

# Not “IF” But “WHEN”?

- “Schema on Write”
  - Traditional Approach
- “Schema on Read”
  - Data is simply copied to the file store, no transformation is needed.
  - A SerDe (Serializer/Deserlizer) is applied during read time to extract the required columns (late binding)
  - New data can start flowing anytime and will appear retroactively once the SerDe is updated to parse it.



# Summary

- Data Models, Tables, Structure, etc.
  - SQL
  - NoSQL
  - Schema on Read vs. Schema on Write