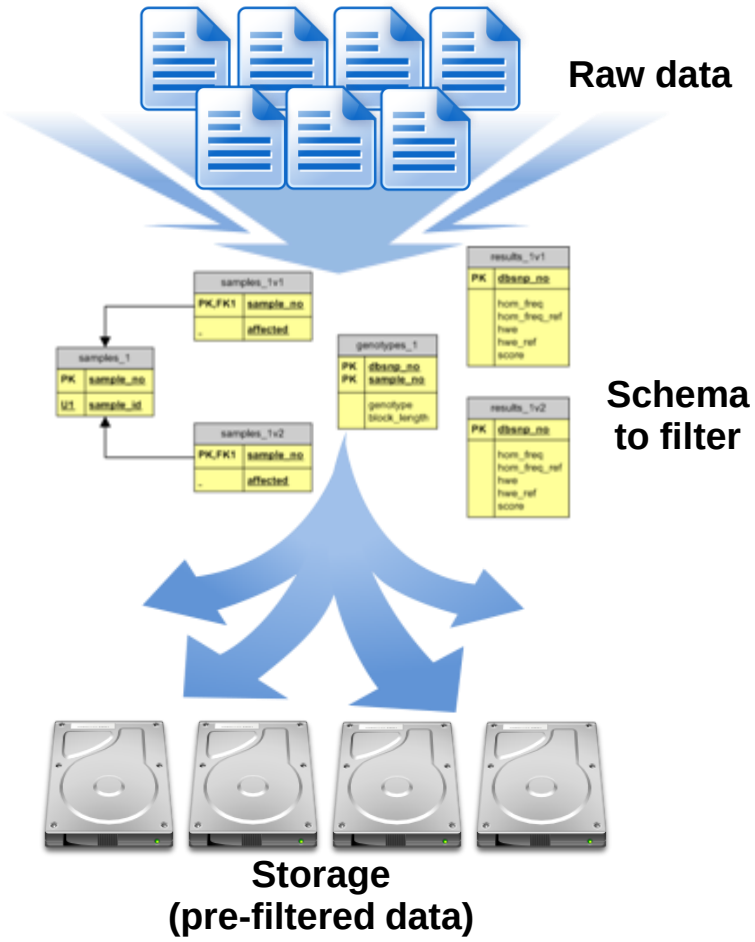
Hadoop Query Languages

Agenda

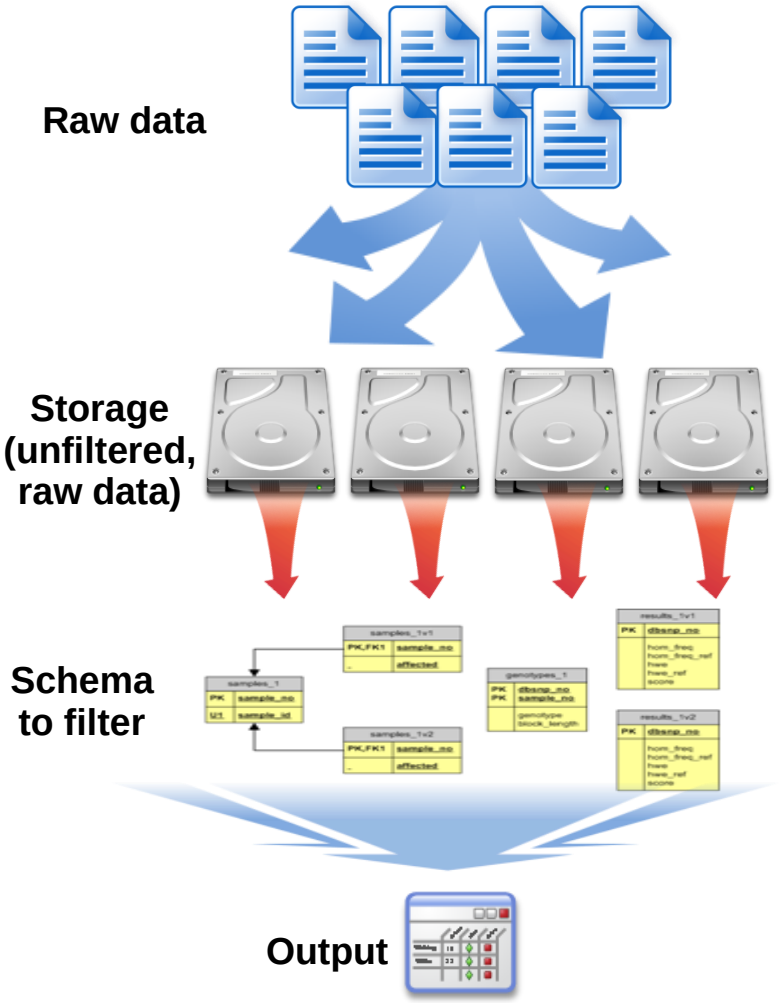
- **Databases vs Hadoop**
- **Hadoop Query Languages**
  - Pig
  - Hive
  - Jaql

Big Difference: Schema on Run

- **Regular database**
  - Schema on load



- **Big Data (Hadoop)**
  - Schema on run



RDBMS vs Hadoop

	RDBMS	Hadoop
Data sources	Structured data with known schemas	Unstructured and structured
Data type	Records, long fields, objects, XML	Files
Data Updates	Updates allowed	Only inserts and deletes
Language	SQL & XQuery	Pig (Pig Latin), Hive (HiveQL), Jaql
Processing type	Quick response, random access	Batch processing
Security	Security and auditing	Partial
Compress	Sophisticated data compression	Simple file compression
Hardware	Enterprise hardware	Commodity hardware
Data access	Random access (indexing)	Access files only (streaming)
History	~40 years of innovation	< 5 years old
Community	Widely used, abundant resources	Not widely adopted yet

How to Analyze Large Data Sets in Hadoop

- Although the Hadoop framework is implemented in Java, MapReduce applications do not need to be written in Java

- To abstract complexities of Hadoop programming model, a few application development languages have emerged that build on top of Hadoop:
  - Pig
  - Hive
  - Jaql

Jaql

Pig, Hive, Jaql – Similarities

- Reduced program size over Java
- Applications are translated to map and reduce jobs behind scenes
- Extension points for extending existing functionality
- Interoperability with other languages
- Not designed for random reads/writes or low-latency queries

Pig, Hive, Jaql – Differences

Characteristic	Pig	Jaql	Hive	BigSQL
Developed by	Yahoo!	IBM	Facebook	IBM
Language	Pig Latin	Jaql	HiveQL	Ansi-SQL
Type of language	Data flow	Data flow	SQL	SQL
Data structures supported	Complex	JSON, semi structured	Mostly structured	Mostly structured
Schema	Optional	Optional	Mandatory	Mandatory

Pig

- **The Pig platform is able to handle many kinds of data, hence the name**
- **Pig Latin is a data flow language**
- **Two components:**
  - Language Pig Latin
  - Runtime environment

- **Two execution modes:**
  - Local
    - Good for testing and prototyping

```
pig -x local
```

  - Distributed (MapReduce)
    - Need access to a Hadoop cluster and HDFS
    - Default mode

```
pig -x mapreduce
```

Pig

- **Three steps in a typical Pig program:**
  - LOAD
    - Load data from HDFS
  - TRANSFORM
    - Translated to a set of map and reduce tasks
    - Relational operators: FILTER, FOREACH, GROUP, UNION, etc.
  - DUMP or STORE
    - Display result on to the screen or store it in a file

- **Pig data types:**
  - Simple types:
    - int, long, float, double, chararray, bytearray, boolean
  - Complex types:
    - tuple: ordered set of fields `(John,18)`
    - bag: collection of tuples `{(John,18), (Mary, 29)}`
    - map: set of key/value pairs `[name#John, phone#1234567]`

Pig

- **Example: wordcount.pig**

```
input = LOAD './all_web_pages' AS (line:chararray);

-- Extract words from each line and put them into a pig bag
-- datatype, then flatten the bag to get one word on each row
words = FOREACH input GENERATE FLATTEN(TOKENIZE(line)) AS word;

-- create a group for each word
word_groups = GROUP words BY word;

-- count the entries in each group
word_count = FOREACH word_groups GENERATE COUNT(words) AS count, group;

-- order the records by count
ordered_word_count = ORDER word_count BY count DESC;
STORE ordered_word_count INTO './word_count_result';
```

- **How to run wordcount.pig?**
  - Local mode:

```
/bin/pig -x local wordcount.pig
```

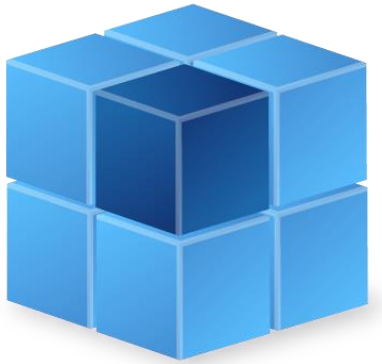
  - Distributed mode (MapReduce):

```
hadoop dfs -copyFromLocal all_web_pages input/all_web_pages
bin/pig -x mapreduce wordcount.pig
```

Hive

- **What is Hive?** 
  - Data warehouse infrastructure built on top of Hadoop
  - Provides an SQL-like language called HiveQL
  - Allows SQL developers and business analysts to leverage existing SQL skills
  - Offers built-in UDFs and indexing
- **What Hive is not?** 
  - Not designed for low-latency queries, unlike RDBMS such as DB2 and Netezza
  - Not schema on write
  - Not for OLTP
  - Not fully SQL compliant, only understand limited commands

Hive



- **Components**:
  - Shell
  - Driver
  - Compiler
  - Engine
  - Metastore
    - Holds table definition, physical layout
- **Data models**:
  - Tables
    - Analogous to tables in RDBMS, composed of columns
  - Partitions
    - For optimizing data access, e.g. range partition tables by date
  - Buckets
    - Data in each partition may in turn be divided into Buckets based on the hash of a column in the table

Hive

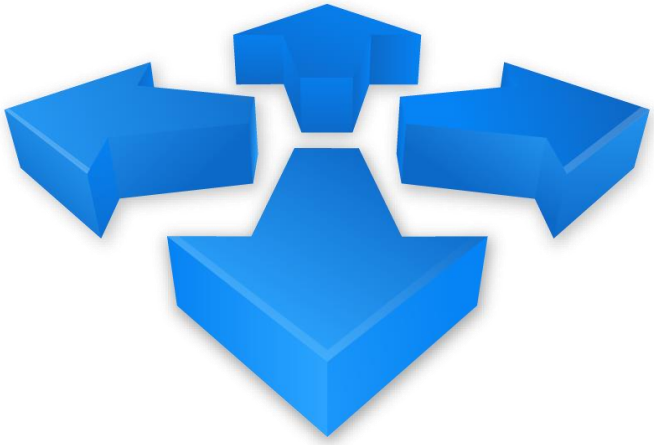
- **Example: movie ratings analysis**

```
-- create a table with tab-delimited text file format
hive> CREATE TABLE movie_ratings (
                userid INT,
                movieid INT,
                rating INT)
          ROW FORMAT DELIMITED
          FIELDS TERMINATED BY '\t'
          STORED AS TEXTFILE;

-- load data
hive> LOAD DATA INPATH 'hdfs://node/movie_data' OVERWRITE INTO
TABLE movie_ratings;

-- gather ratings per movie
hive> SELECT movieid, rating, COUNT(rating)
          FROM movie_ratings
          GROUP BY movieid, rating;
```

Jaql

Designed for easy manipulation and analytics of semi-structured data, support formats such as JSON, XML, CSV, flat files, etc.

Developed by IBM

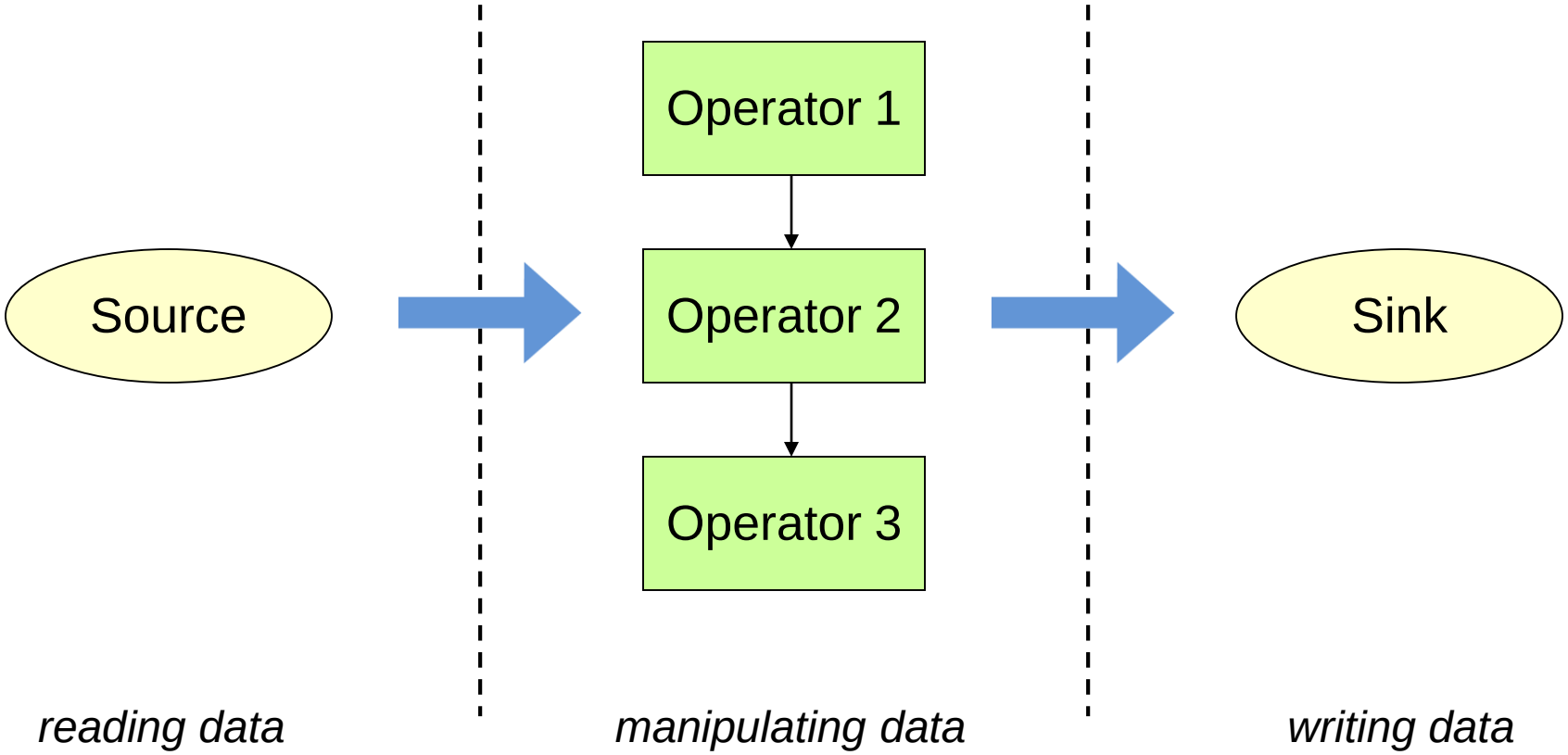


Flexibility with optional schema

Easy extensibility

Jaql – How does a Jaql query work?

- **A Jaql query can be thought of as a pipeline**
- **Data manipulation through operators**
  - FILTER, TRANSFORM, GROUP, JOIN, EXPAND, SORT, TOP

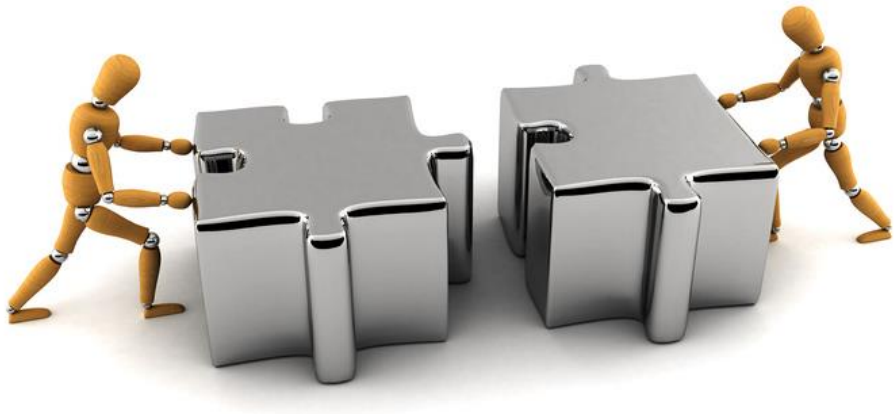


Jaql

- **In addition to core operators, Jaql also provides built-in functions**

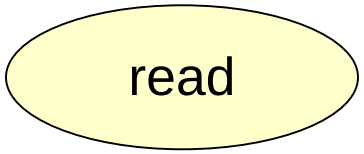
- **Data models:**
  - Atomic types: boolean, string, long, etc.
  - Complex types: array, record

- **Where to run Jaql queries?**
  - Shell: `jaqlshell`
    - Cluster mode
    - Local mode: for testing, prototyping purposes
  - Eclipse
  - Embedded in Java



Jaql

- **Example: find employees who are manager or have income > 50000**



Query

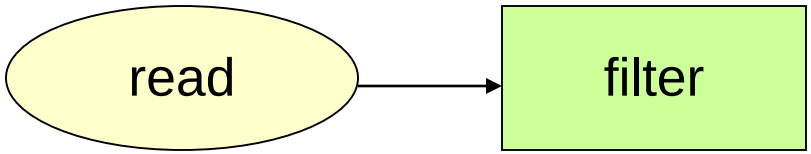
read(hdfs “employees”));

Data

```
[
        {name: "Jon Doe", income:
                 40000, mgr:
false},
        {name: "Vince Wayne",

        income: 52500, mgr:
                        false},
        {name: "Jane Dean",

        income: 72000, mgr:
                        true},
        {name: "Alex Smith",

        income: 35000, mgr:
                        false}
]
```

Jaql

- **Example: find employees who are manager or have income > 50000**



Query

read(hdfs("employees"))
-> **filter** $.mgr or $.income > 50000;

Data

```
[
        {name: "Vince Wayne",
        income: 52500, mgr:
                false},
        {name: "Jane Dean",
        income: 72000, mgr:
                true}
]
```

Jaql

- **Example: find employees who are manager or have income > 50000**

Query

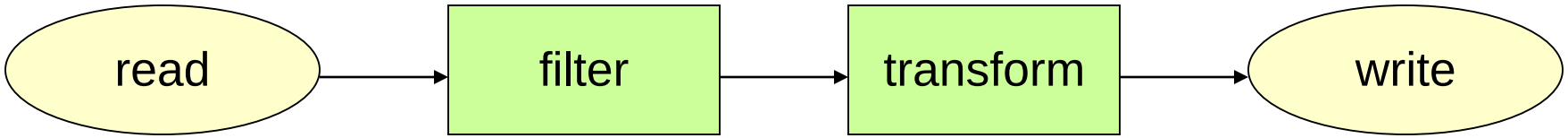
```
read(hdfs("employees"))
-> filter $.mgr or $.income > 50000
-> transform { $.name, $.income };
```

Data

```
[
        {name: "Vince Wayne",

        income: 52500},
        {name: "Jane Dean",

        income: 72000}
]
```

Jaql

- **Example: find employees who are manager or have income > 50000**



Query

```
read(hdfs("employees"))
-> filter $.mgr or $.income > 50000
-> transform { $.name, $.income }
-> write(hdfs("output"));
```

Data

```
[
        {name: "Vince Wayne",

        income: 52500},
        {name: "Jane Dean",

        income: 72000}
]
```

Questions?