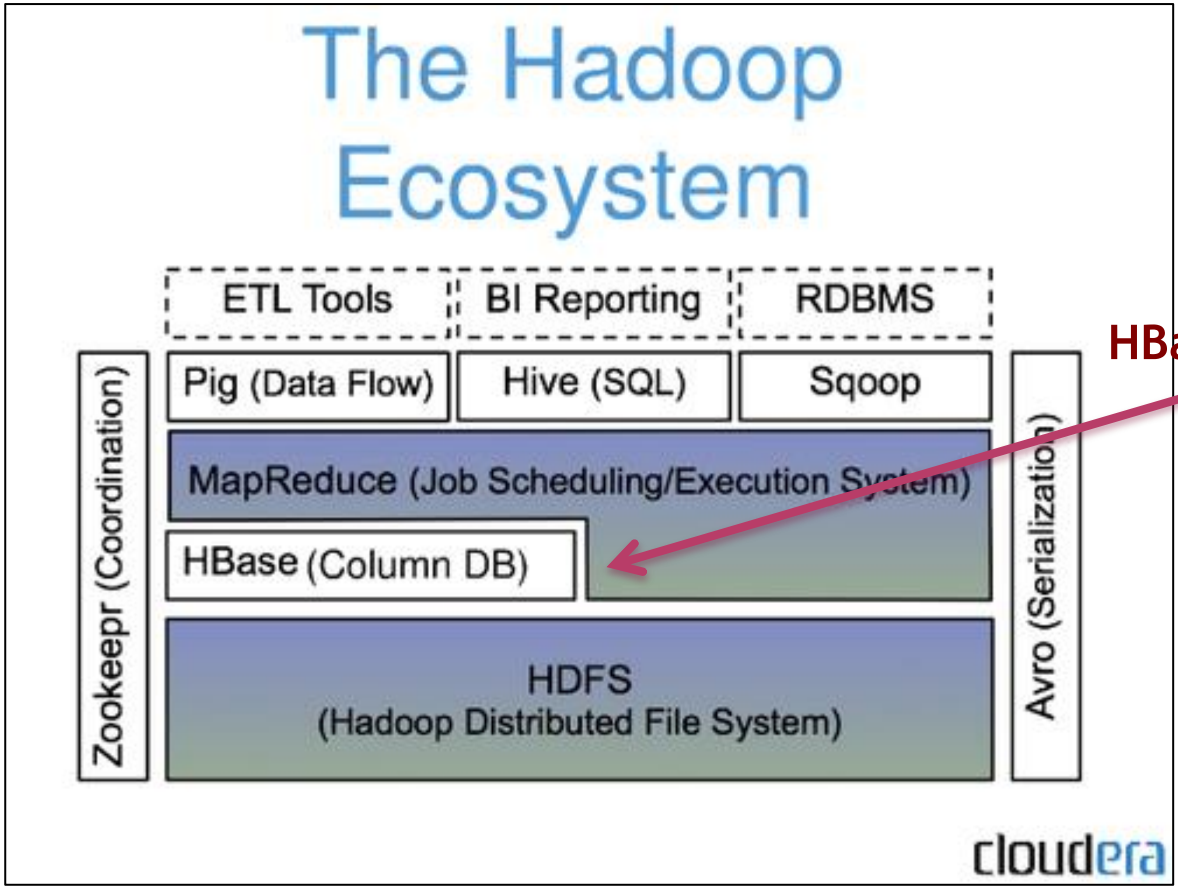
CASE STUDY: HADOOP

OUTLINE

- Hadoop - Basics
- HDFS
  - Goals
  - Architecture
  - Other functions
- MapReduce
  - Basics
  - Word Count Example
  - Handy tools
  - Finding shortest path example
- Related Apache sub-projects (Pig, HBase,Hive)

HBASE: PART OF HADOOP'S ECOSYSTEM



HBase is built on top of HDFS

HBase files are internally stored in HDFS

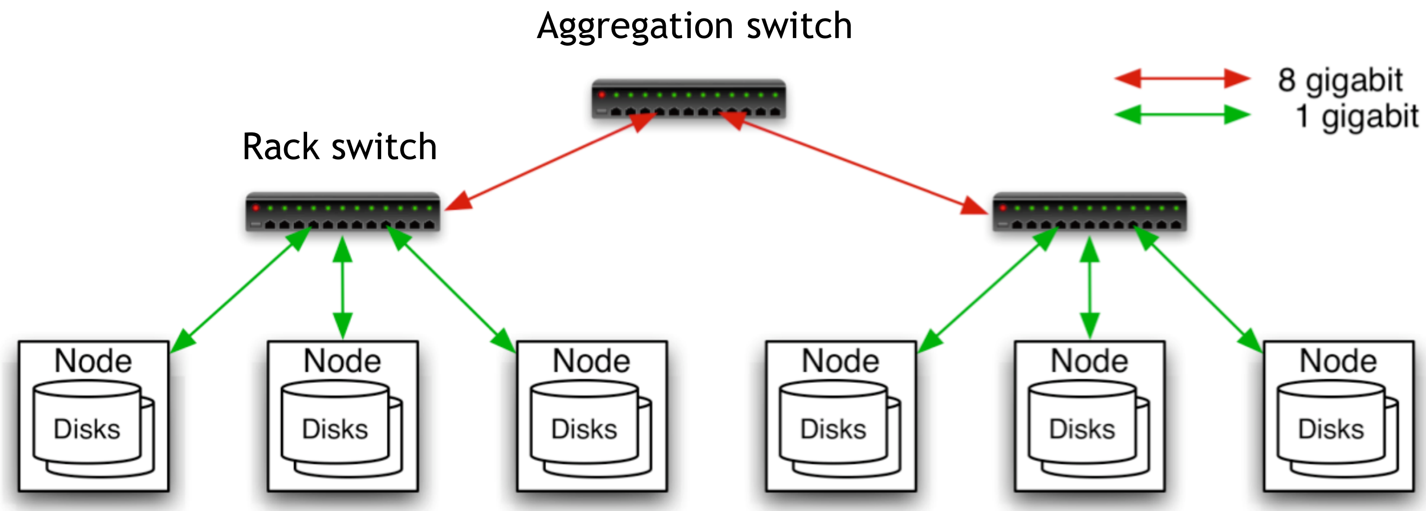
HADOOP - WHY ?

- Need to process huge datasets on large clusters of computers
- Very expensive to build reliability into each application
- Nodes fail every day
  - Failure is expected, rather than exceptional
  - The number of nodes in a cluster is not constant
- Need a common infrastructure
  - Efficient, reliable, easy to use
  - Open Source, Apache Licence

WHO USES HADOOP?

- Amazon/A9
- Facebook
- Google
- New York Times
- Veoh
- Yahoo!
- …. many more

COMMODITY HARDWARE



- Typically in 2 level architecture
  - Nodes are commodity PCs
  - 30-40 nodes/rack
  - Uplink from rack is 3-4 gigabit
  - Rack-internal is 1 gigabit

HADOOP DISTRIBUTED FILE SYSTEM (HDFS)

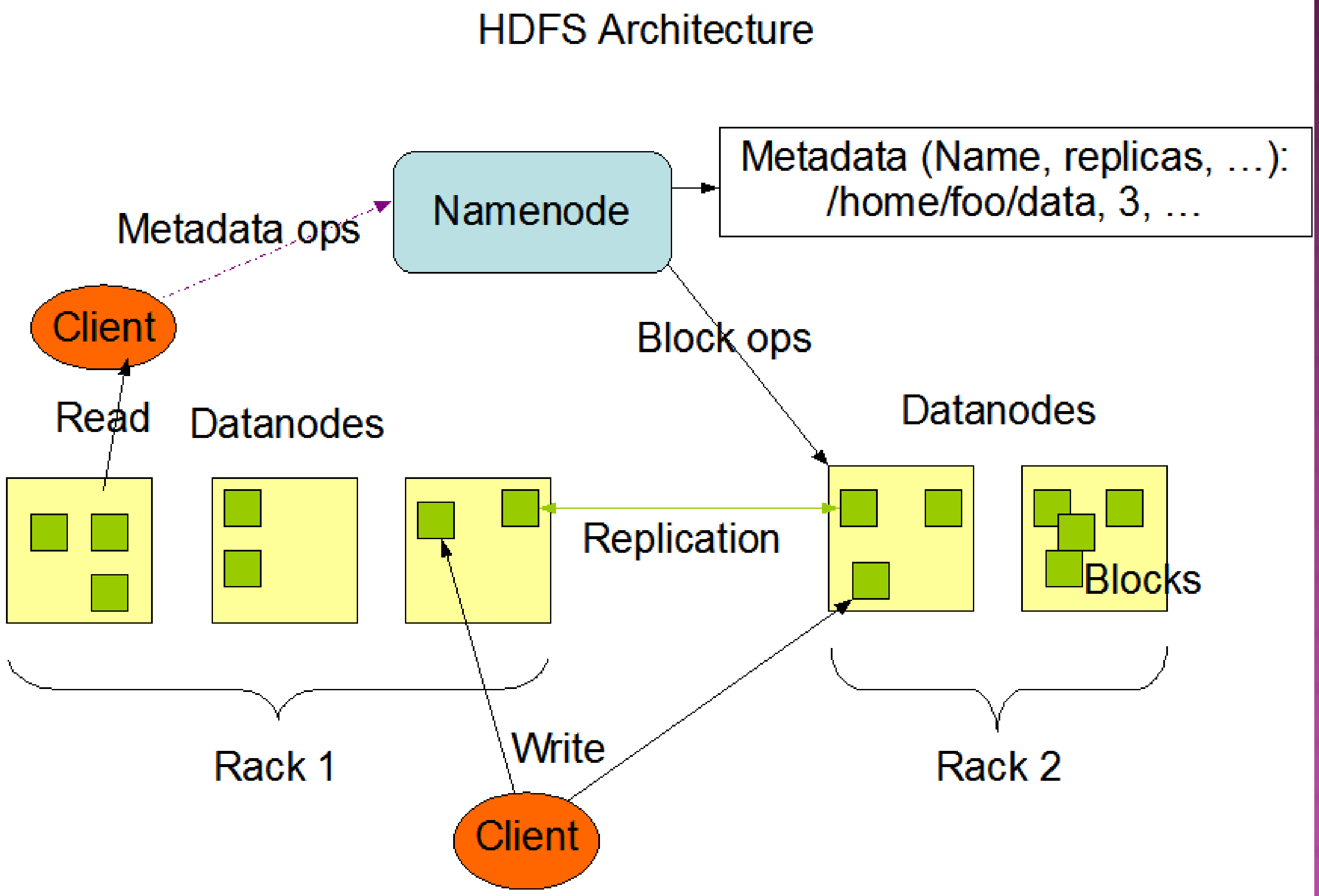
GOALS OF HDFS

- Very Large Distributed File System
  - 10K nodes, 100 million files, 10PB
- Assumes Commodity Hardware
  - Files are replicated to handle hardware failure
  - Detect failures and recover from them
- Optimized for Batch Processing
  - Data locations exposed so that computations can move to where data resides
  - Provides very high aggregate bandwidth

Distributed File System

- Single Namespace for entire cluster
- Data Coherency
  - Write-once-read-many access model
  - Client can only append to existing files
- Files are broken up into blocks
  - Typically 64MB block size
  - Each block replicated on multiple DataNodes
- Intelligent Client
  - Client can find location of blocks
  - Client accesses data directly from DataNode

HDFS ARCHITECTURE



FUNCTIONS OF A NAMENODE

- Manages File System Namespace
  - Maps a file name to a set of blocks
  - Maps a block to the DataNodes where it resides
- Cluster Configuration Management
- Replication Engine for Blocks

NAMENODE METADATA

- Metadata in Memory
  - The entire metadata is in main memory
  - No demand paging of metadata
- Types of metadata
  - List of files
  - List of Blocks for each file
  - List of DataNodes for each block
  - File attributes, e.g. creation time, replication factor
- A Transaction Log
  - Records file creations, file deletions etc

DATANODE

- A Block Server
  - Stores data in the local file system (e.g. ext3)
  - Stores metadata of a block (e.g. CRC)
  - Serves data and metadata to Clients
- Block Report
  - Periodically sends a report of all existing blocks to the NameNode
- Facilitates Pipelining of Data
  - Forwards data to other specified DataNodes

BLOCK PLACEMENT

- Current Strategy
  - One replica on local node
  - Second replica on a remote rack
  - Third replica on same remote rack
  - Additional replicas are randomly placed
- Clients read from nearest replicas
- Would like to make this policy pluggable

HEARTBEATS

- DataNodes send hearbeat to the NameNode
  - Once every 3 seconds
- NameNode uses heartbeats to detect DataNode failure

Replication Engine

- NameNode detects DataNode failures
  - Chooses new DataNodes for new replicas
  - Balances disk usage
  - Balances communication traffic to DataNodes

DATA CORRECTNESS

- Use Checksums to validate data
  - Use CRC32
- File Creation
  - Client computes checksum per 512 bytes
  - DataNode stores the checksum
- File access
  - Client retrieves the data and checksum from DataNode
  - If Validation fails, Client tries other replicas

NAMENODE FAILURE

- A single point of failure
- Transaction Log stored in multiple directories
  - A directory on the local file system
  - A directory on a remote file system (NFS/CIFS)
- Need to develop a real HA solution

Secondary Namenode

- Copies FsImage and Transaction Log from Namenode to a temporary directory
- Merges FSImage and Transaction Log into a new FSImage in temporary directory
- Uploads new FSImage to the NameNode
  - Transaction Log on NameNode is purged

USER INTERFACE

- Commads for HDFS User:
  - hadoop dfs -mkdir /foodir
  - hadoop dfs -cat /foodir/myfile.txt
  - hadoop dfs -rm /foodir/myfile.txt
- Commands for HDFS Administrator
  - hadoop dfsadmin -report
  - hadoop dfsadmin -decommision datanodename
- Web Interface
  - http://host:port/dfshealth.jsp

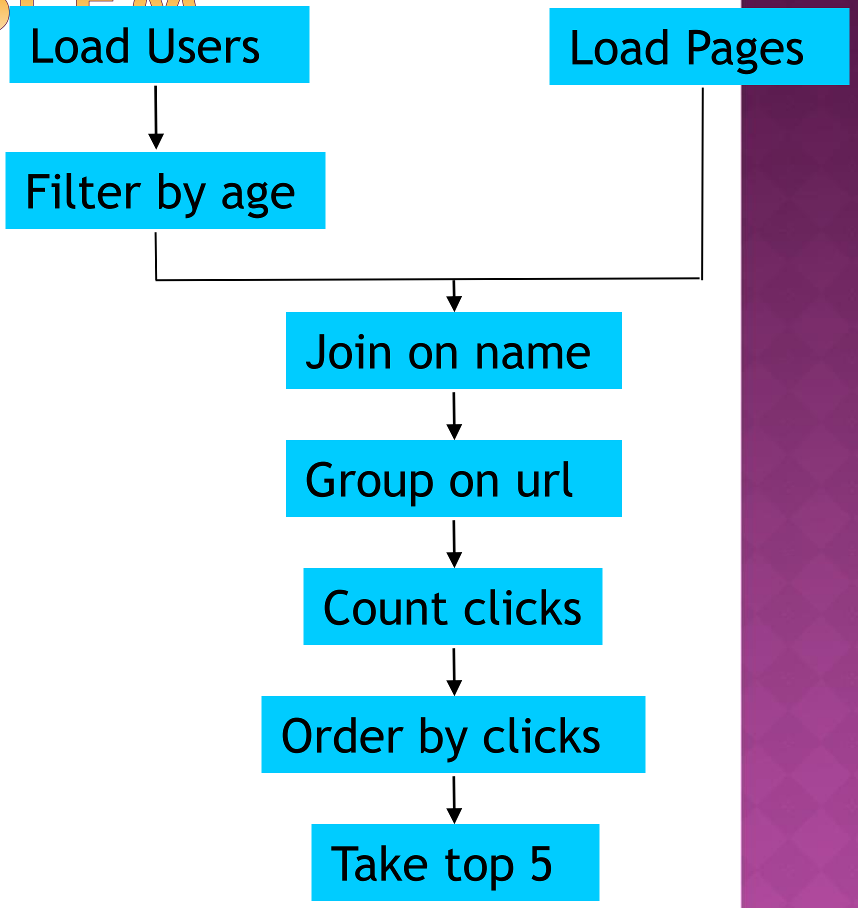
PIG

PIG

- Started at Yahoo! Research
- Now runs about 30% of Yahoo!'s jobs
- Features
  - Expresses sequences of MapReduce jobs
  - Data model: nested "bags" of items
  - Provides relational (SQL) operators (JOIN, GROUP BY, etc.)
  - Easy to plug in Java functions

AN EXAMPLE PROBLEM

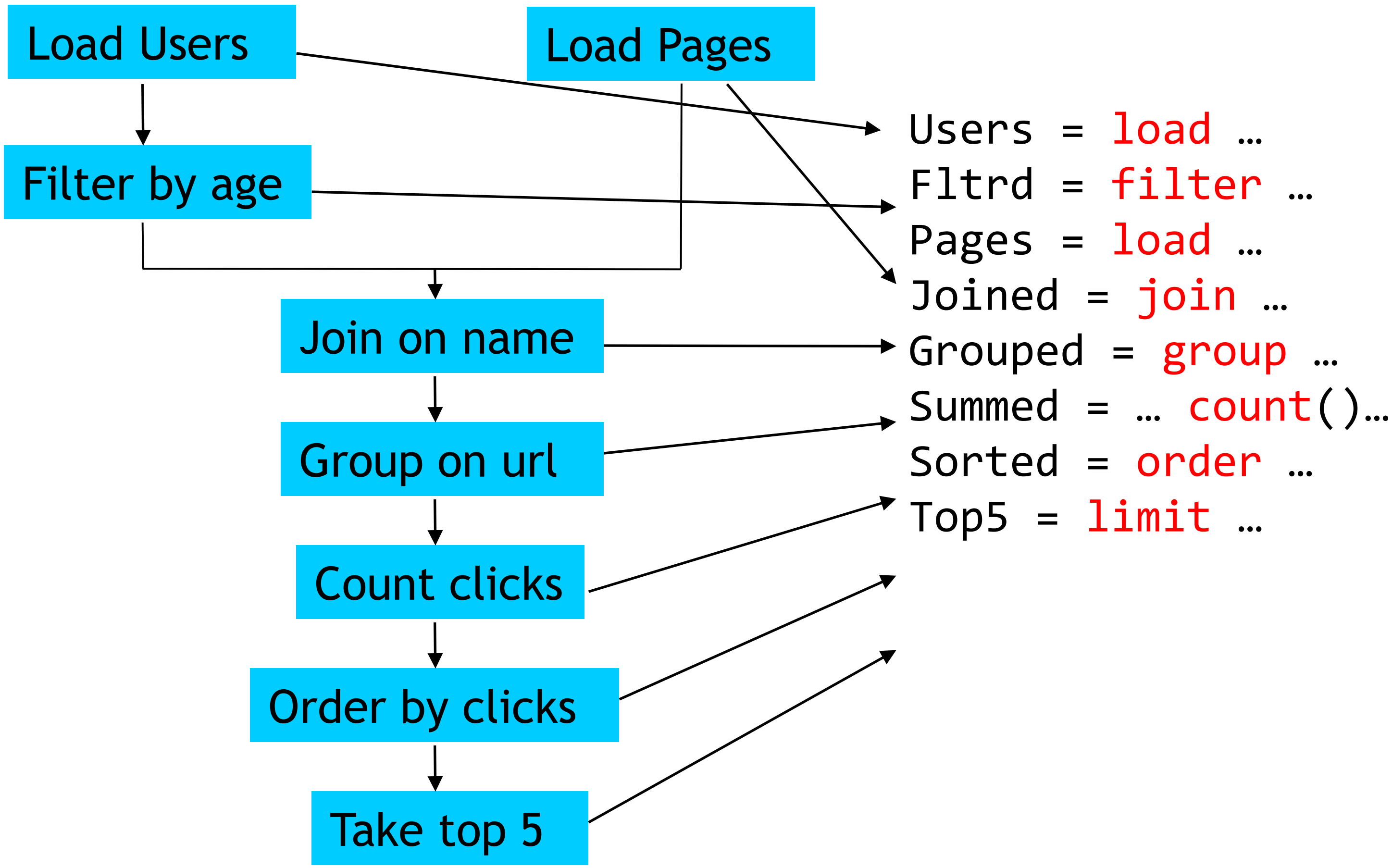
- Suppose you have user data in a file, website data in another, and you need to find the top 5 most visited pages by users aged 18-25



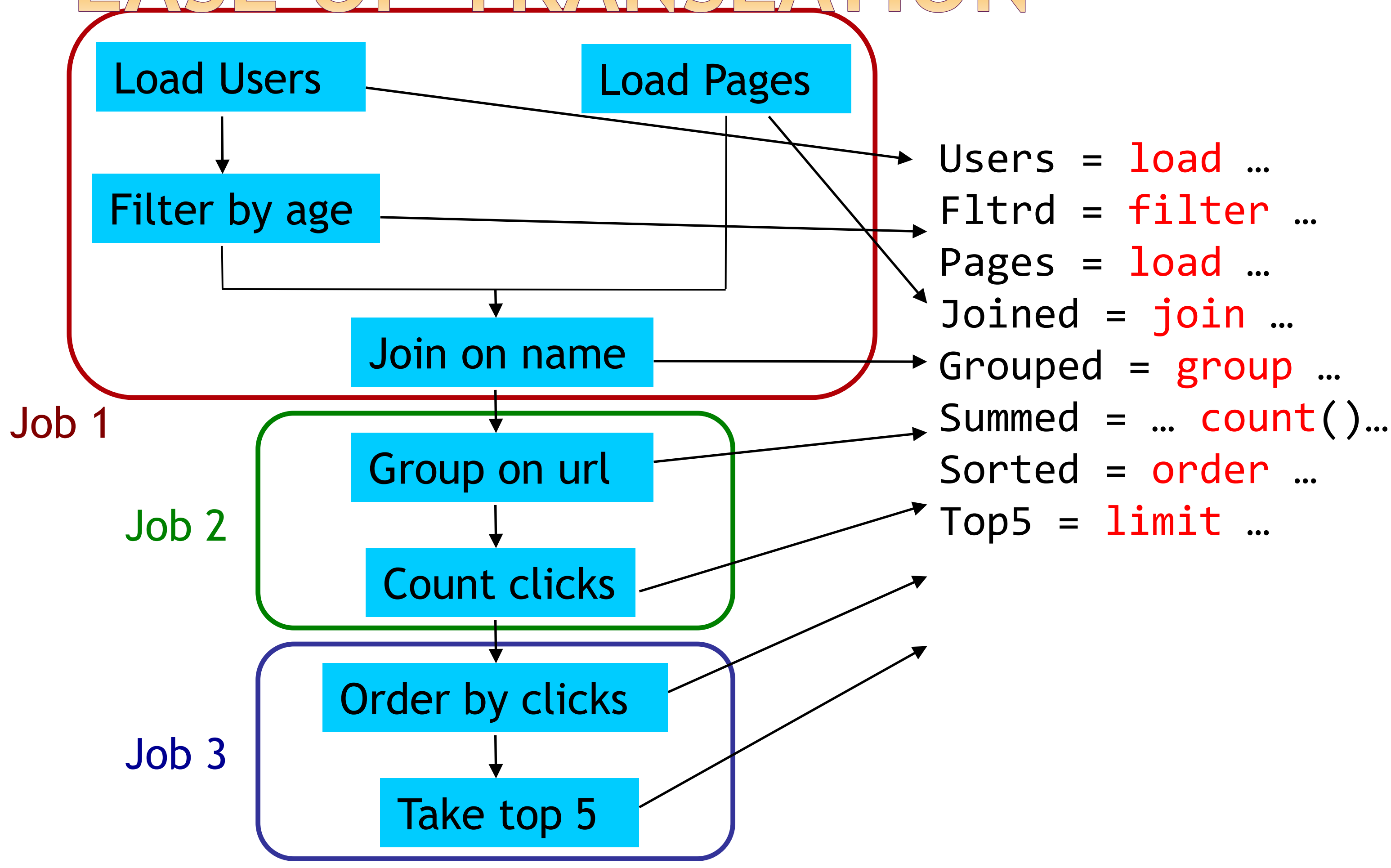
IN PIG LATIN

```
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’;
```

EASE OF TRANSLATION



EASE OF TRANSLATION



HBASE

HBASE - WHAT?

- Modeled on Google's Bigtable
- Row/column store
- Billions of rows/millions on columns
- Column-oriented - nulls are free
- Untyped - stores byte[]

HBASE - DATA MODEL

Row	Timestamp	Column family: animal:		Column family repairs:
		animal:type	animal:size	repairs:cost
enclosure1	t2	zebra		1000 EUR
	t1	lion	big	
enclosure2	…	…	…	…

HBASE - DATA STORAGE

Column family animal:

(enclosure1, t2, animal:type)	zebra
(enclosure1, t1, animal:size)	big
(enclosure1, t1, animal:type)	lion

Column family repairs:

(enclosure1, t1, repairs:cost)	1000 EUR

HBASE - CODE

```
HTable table = …
Text row = new Text("enclosure1");
Text col1 = new Text("animal:type");
Text col2 = new Text("animal:size");
BatchUpdate update = new BatchUpdate(row);
update.put(col1, "lion".getBytes("UTF-8"));
update.put(col2, "big".getBytes("UTF-8));
table.commit(update);

update = new BatchUpdate(row);
update.put(col1, "zebra".getBytes("UTF-8"));
table.commit(update);
```

HBASE - QUERYING

- Retrieve a cell

  Cell = table.getRow("enclosure1").getColumn("animal:type").getValue();

- Retrieve a row

  RowResult = table.getRow("enclosure1");

- Scan through a range of rows

  Scanner s = table.getScanner(new String[] { "animal:type" });

HIVE

HIVE

- Developed at Facebook
- Used for majority of Facebook jobs
- "Relational database" built on Hadoop
  - Maintains list of table schemas
  - SQL-like query language (HiveQL)
  - Can call Hadoop Streaming scripts from HiveQL
  - Supports table partitioning, clustering, complex data types, some optimizations

CREATING A HIVE TABLE

```
CREATE TABLE page_views(viewTime INT, userid BIGINT,
                   page_url STRING, referrer_url STRING,
                   ip STRING COMMENT 'User IP address')
COMMENT 'This is the page view table'
PARTITIONED BY(dt STRING, country STRING)
STORED AS SEQUENCEFILE;
```

- Partitioning breaks table into separate files for each (dt, country) pair

  Ex: /hive/page_view/dt=2008-06-08,country=USA

  /hive/page_view/dt=2008-06-08,country=CA

A SIMPLE QUERY

- Find all page views coming from xyz.com on March 31st:

```
SELECT page_views.*
FROM page_views
WHERE page_views.date >= '2008-03-01'
AND page_views.date <= '2008-03-31'
AND page_views.referrer_url like '%xyz.com';
```

- Hive only reads partition `2008-03-01,*` instead of scanning entire table

AGGREGATION AND JOINS

- Count users who visited each page by gender:

```
SELECT pv.page_url, u.gender, COUNT(DISTINCT u.id)
FROM page_views pv JOIN user u ON (pv.userid = u.id)
GROUP BY pv.page_url, u.gender
WHERE pv.date = '2008-03-03';
```

- Sample output:

page_url	gender	count(userid)
home.php	MALE	12,141,412
home.php	FEMALE	15,431,579
photo.php	MALE	23,941,451
photo.php	FEMALE	21,231,314

USING A HADOOP STREAMING MAPPER SCRIPT

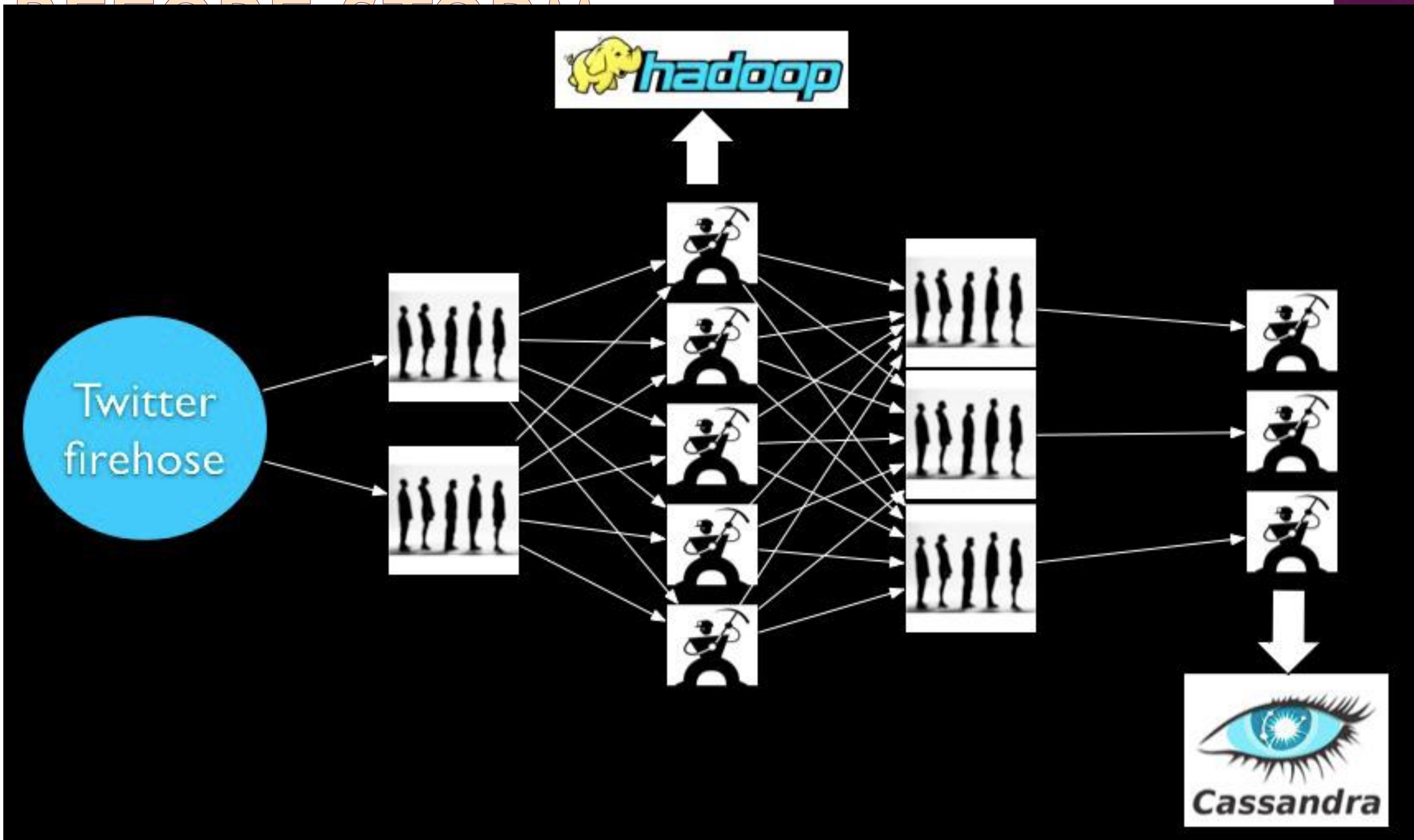
```
SELECT TRANSFORM(page_views.userid,
                 page_views.date)
USING 'map_script.py'
AS dt, uid CLUSTER BY dt
FROM page_views;
```

STORM

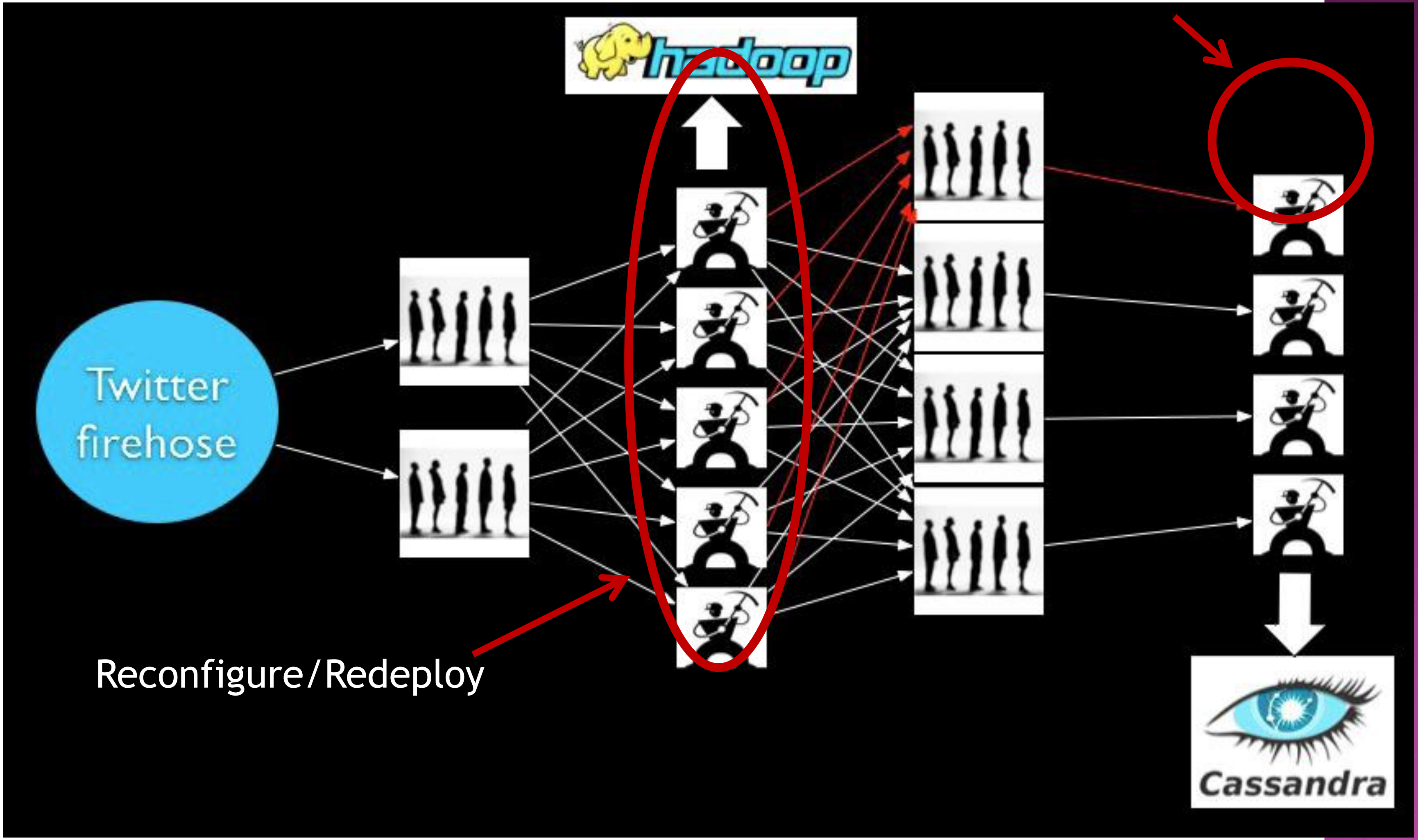
STORM

- Developed by BackType which was acquired by Twitter
- Lots of tools for data (i.e. batch) processing
  - Hadoop, Pig, HBase, Hive, …
- None of them are realtime systems which is becoming a real requirement for businesses
- Storm provides realtime computation
  - Scalable
  - Guarantees no data loss
  - Extremely robust and fault-tolerant
  - Programming language agnostic

BEFORE STORM



BEFORE STORM – ADDING A WORKER



PROBLEMS

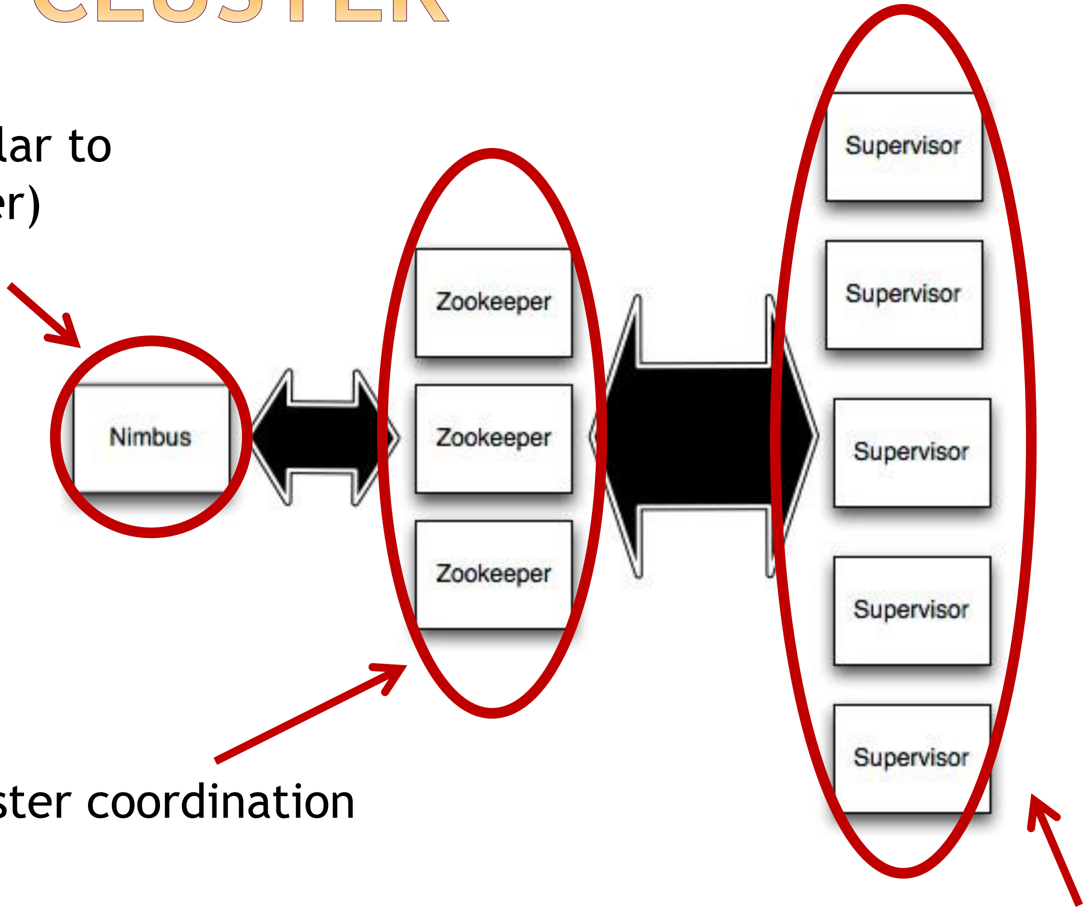
- Scaling is painful
- Poor fault-tolerance
- Coding is tedious

WHAT WE WANT

- Guaranteed data processing
- Horizontal scalability
- Fault-tolerance
- No intermediate message brokers!
- Higher level abstraction than message passing
- “Just works” !!

STORM CLUSTER

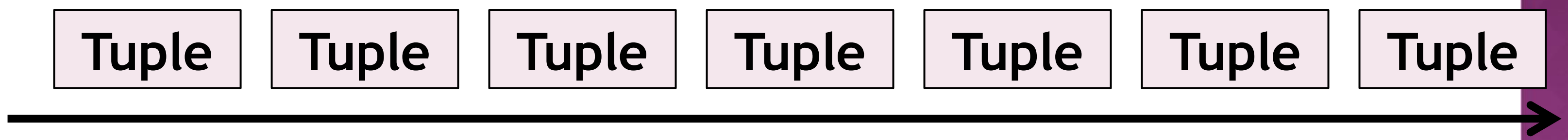
Master node (similar to Hadoop JobTracker)



Used for cluster coordination

Run worker processes

STREAMS



Unbounded sequence of tuples