GENERAL ASSEMBLY

INTRO TO DATA SCIENCE

TOPICS IN DATA ENGINEERING

BIG DATA

BIG DATA

What does ***big data*** actually refer to?

BIG DATA: FUN FACTS

The Large Hadron Collider (LHC) produces roughly 30 petabytes per year

The *LHC Data Centre*:

- has 11,000 servers with 100,000 processor cores
- processes about one petabyte of data every day - the equivalent of around 210,000 DVDs

source: CERN Computing

BIG DATA

What does ***big data*** actually refer to?

A common approach is to talk about big data in terms of the 3 Vs:

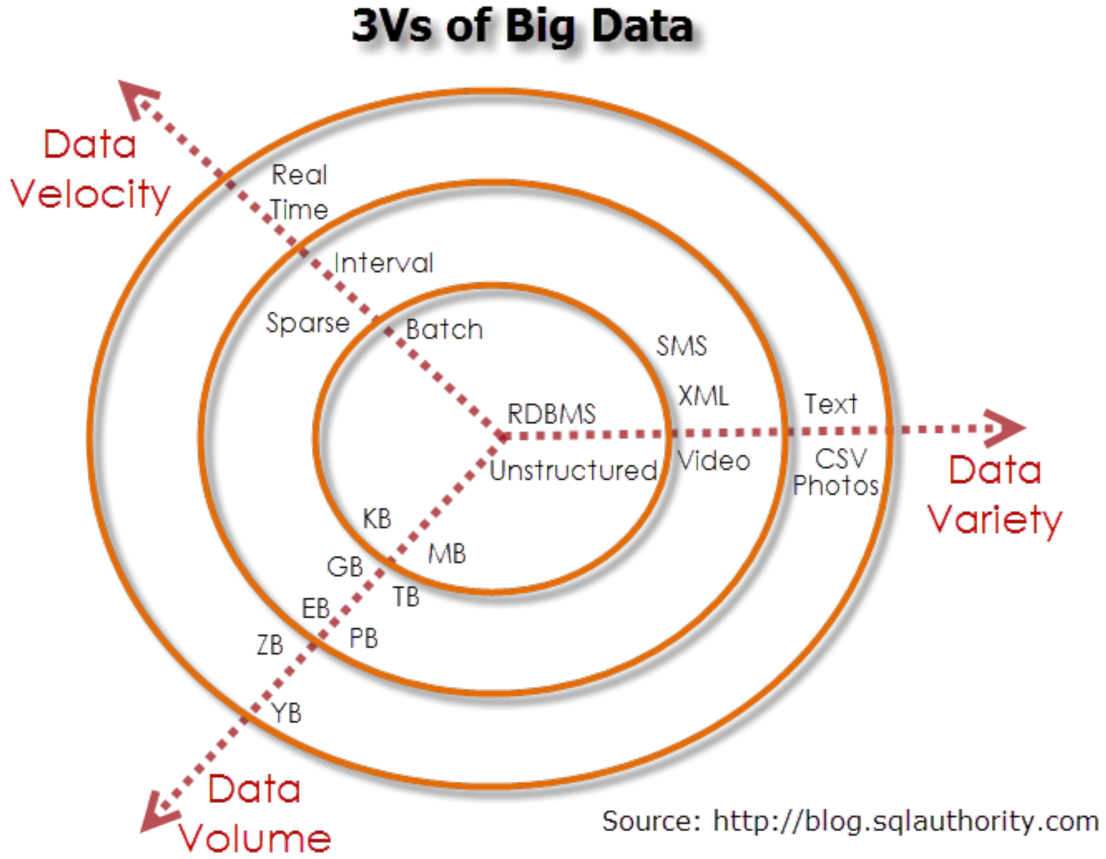
- Volume
- Velocity
- Variety

source: 3-D Data Management: Controlling Data Volume, Velocity and Variety (Feb 6, 2001)

BIG DATA

- ***Volume***
  - more than can fit in memory on your laptop, e.g. Amazon user behavior data
- ***Velocity***
  - faster than standard machines can process, e.g. Twitter Firehose
- ***Variety***
  - does not conform to a single structure, e.g. Google's cache of web pages

BIG DATA



3Vs of Big Data
Data Velocity
Real Time
Interval
Sparse
Batch
SMS
XML
Text
RDBMS
Unstructured
Video
CSV
Photos
Data Variety
KB
MB
GB
TB
EB
PB
ZB
YB
Data Volume
Source: http://blog.sqlauthority.com

Scaling Big Data Processing

How would you approach dealing with this kind of data?

One approach would be to get a huge supercomputer, but this has some obvious drawbacks:

- expensive
- difficult to maintain
- scalability is bounded

Scaling Big Data Processing

How would you approach dealing with this kind of data?

One approach would be to get a huge supercomputer, but this has some obvious drawbacks:

- expensive
- difficult to maintain
- ***scalability is bounded***

SCALING BIG DATA PROCESSING

Instead of one huge machine, what if we got a bunch of regular (commodity) machines?

- cheaper
- easier to maintain
- scalability is unbounded
  - “just add more nodes to the cluster”

Scaling Big Data Processing

Guiding Principles:

- Scale horizontally, not vertically
- Assume hardware failure is common
- Move code to the data
- Hide system level-details from developers (and data scientists)
- Seamless scalability

MapReduce

Scaling Big Data Processing

One day when I was having lunch with Richard Feynman, I mentioned to him that I was planning to start a company to build a parallel computer with a million processors. His reaction was unequivocal,

"That is positively the dopiest idea I ever heard."

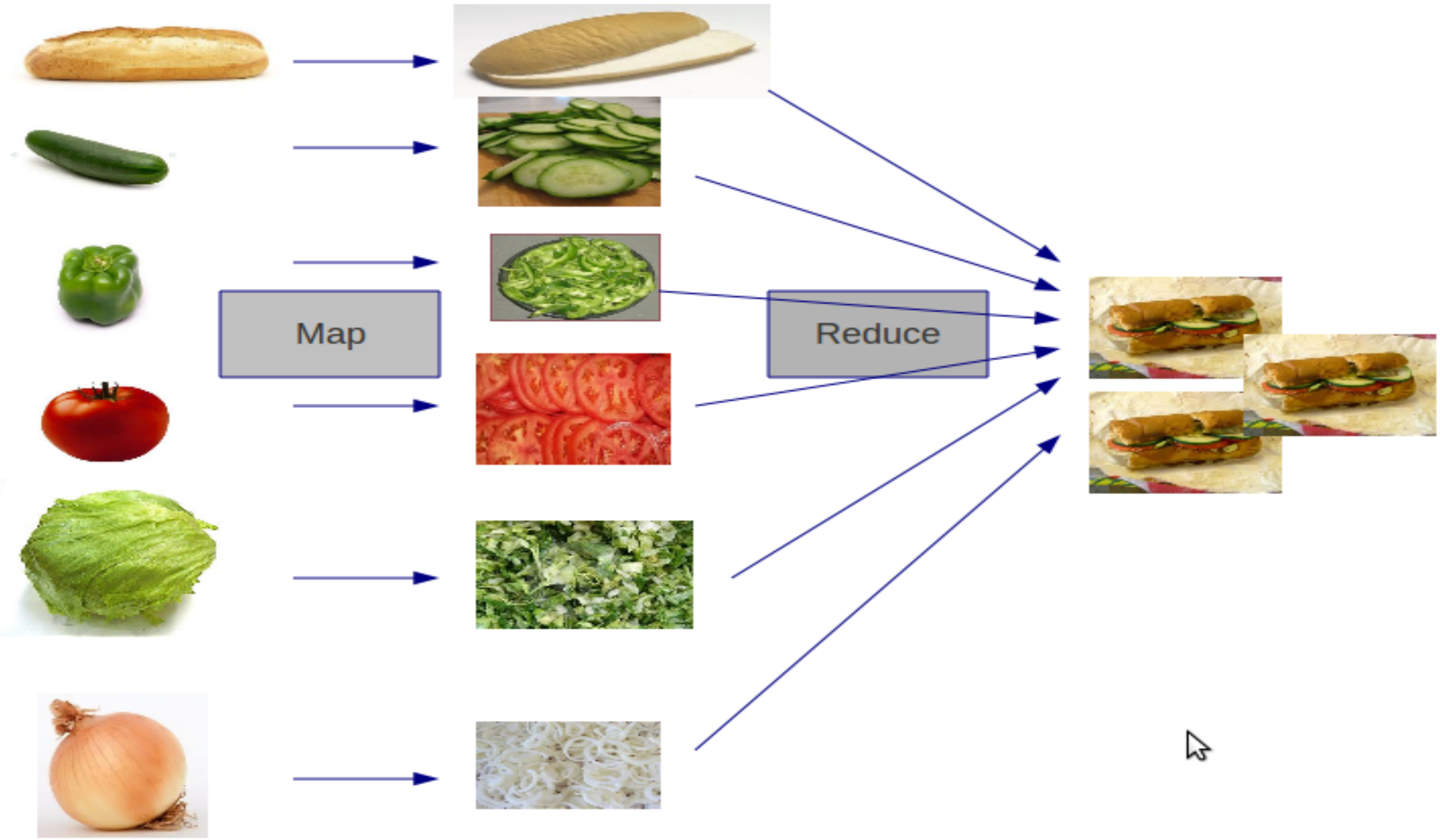
- Danny Hillis in *Physics Today* on the Connection Machine, reposted: Richard Feynman and The Connection Machine

MapReduce

From the famous Google Research Paper:

MapReduce: Simplified Data Processing on Large Clusters

- A programming model and an associated implementation for processing and generating large data sets
- MapReduce programs are automatically parallelized and executed on a large cluster of commodity machines


Map
Reduce

MapReduce

MapReduce involves splitting a problem into subtasks and processing these subtasks in parallel.

This takes place in two phases:

1. the mapper phase
2. the reducer phase

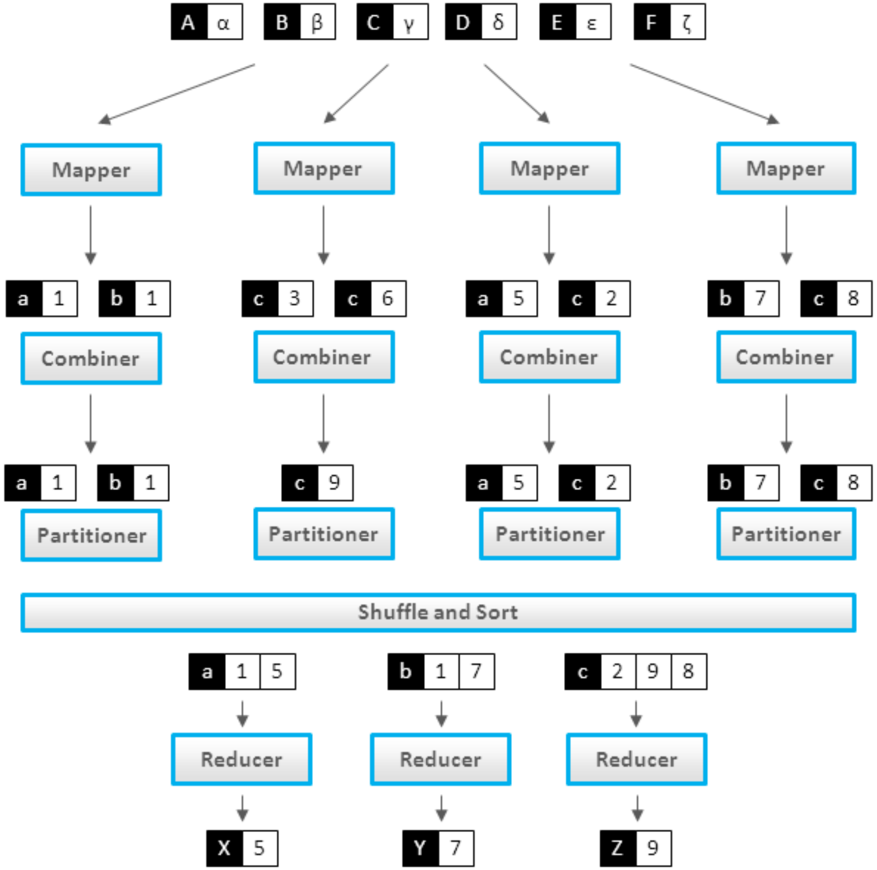
MapReduce

MapReduce involves splitting a problem into subtasks and processing these subtasks in parallel.

This takes place in (approximately) two phases:

1. the mapper phase
- shuffle/sort
2. the reducer phase

MAP-REDUCE



A α
B β
C γ
D δ
E ε
F ζ
Mapper
Mapper
Mapper
Mapper
a 1
b 1
c 3
c 6
a 5
c 2
b 7
c 8
Combiner
Combiner
Combiner
Combiner
a 1
b 1
c 9
a 5
c 2
b 7
c 8
Partitioner
Partitioner
Partitioner
Partitioner
Shuffle and Sort
a 1 5
b 1 7
c 2 9 8
Reducer
Reducer
Reducer
X 5
Y 7
Z 9

MapReduce

Additional intermediate steps in a MapReduce workflow:

- mappers – filter & transform data
- combiners – perform reducer operations on the mapper node (optional step, to reduce network traffic and disk I/O).
- partitioners – shuffle/sort/redirect mapper output
- reducers – aggregate results

MapReduce

The MapReduce framework handles a lot of messy details for you:

- parallelization & distribution (eg, input splitting)
- partitioning (shuffle/sort/redirect)
- fault-tolerance (fact: tasks/nodes will fail!)
- I/O scheduling
- status and monitoring

This (along with the functional semantics) allows you to focus on solving the problem instead of accounting & housekeeping details.

MapReduce: Hadoop

Hadoop is a popular open-source Java-based implementation of the map-reduce framework (including file storage for input/output).

You can download Hadoop and configure a set of machines to operate as a MapReduce cluster, or you can run it as a service via Amazon's Elastic MapReduce.

Hadoop is written in Java, but the Hadoop Streaming utility allows client code to be supplied as executables (eg, written in any language)

MapReduce: Hadoop

Hadoop Distributed File System (HDFS):

- Data is replicated in the (distributed) file system across several nodes
- This permits locality optimization (and fault tolerance) by allowing the mapper tasks to run on the same nodes where the data resides
- So we move code to data (instead of data to code), thus avoiding a lot of network traffic and disk I/O

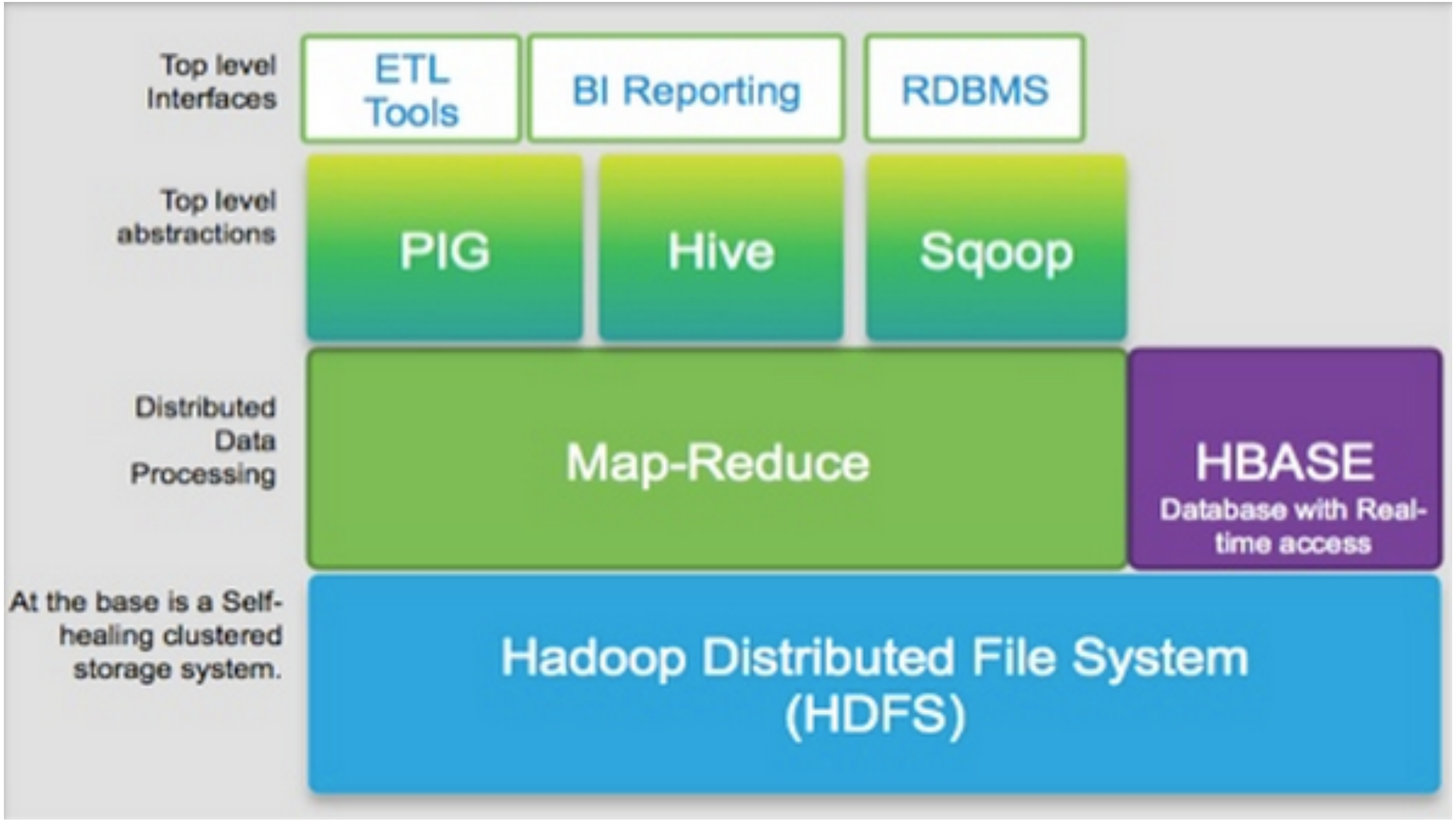
MapReduce

It’s possible to overlay the map-reduce framework with an additional declarative syntax.

This makes operations like `SELECT` & `JOIN` easier to implement and less error prone.

Popular examples include ***Pig*** and ***Hive***.

MAPREDUCE: HADOOP ECOSYSTEM



MAPREDUCE

source: http://www.slideshare.net/kevinweil/hadoop-pig-and-twitter-nosql-east-2009

MapReduce

A Real Pig Script

top_5.pig

```
users = load 'users.csv' as (username: chararray, age: int);
users_1825 = filter users by age >= 18 and age <= 25;

pages = load 'pages.csv' as (username: chararray, url: chararray);

joined = join users_1825 by username, pages by username;
grouped = group joined by url;
summed = foreach grouped generate group as url, COUNT(joined) AS views;
sorted = order summed by views desc;
top_5 = limit sorted 5;

store top_5 into 'top_5_sites.csv';
```

- Now, just for fun... the same calculation in vanilla Hadoop MapReduce.

MapReduce

MapReduce: Word Count

MapReduce: Word Count

MapReduce processes data in terms of key-value pairs:

```
input           <k1, v1>

mapper             <k1, v1> → <k2, v2>

(partitioner)   <k2, v2> → <k2, [all k2 values]>

reducer         <k2, [all k2 values]> → <k3, v3>
```

MapReduce: Word Count

Using the following input, we can implement the “Hello World” of MapReduce: a word count

```
where
where in
where in the
where in the world
where in the world is
where in the world is carmen
where in the world is carmen sandiego
```

MapReduce: Word Count

```
def mapper(doc_id, doc):
    terms = tokenize(doc)
    for term in terms:
        count = 1
        yield term, count
```

The mapper emits key-value pairs for each word encountered in the input data:

```
where  1
where  1
in     1
where  1
in     1
the    1
…
<term> 1
```

MapReduce: Word Count

The ***partitioner***

- internal to the MapReduce framework, so we don’t have to write this ourselves
- shuffles & sorts the mapper output, and redirects all intermediate results for a given key (e.g. term) to a single reducer

```
where     [1, 1, 1, 1, 1, 1, 1]
in        [1, 1, 1, 1, 1, 1]
the          [1, 1, 1, 1, 1]
world     [1, 1, 1, 1]
is        [1, 1 ,1]
carmen       [1, 1]
sandiego [1]
```

MAPREDUCE: WORD COUNT

```
def reducer(term, counts):
   total = 0
   for count in counts:
      total += 1
   yield term, total
```

The reducer receives all values for a given key and aggregates (in this case, sums) the results:

```
carmen     2
is         3
in         6
the        5
sandiego   1
where      7
world      4
```

FROM SQL TO HIVE QL

SQL

SQL is a query language designed to load, retrieve, and update data from relational databases

What are some of the most commonly known relational databases?

-

SQL

SQL is a query language to load, retrieve, and update data in relational databases

What are some of the most commonly known relational databases?

- MySQL
- PostgreSQL
- MS-SQL
- Oracle
- SQLite

SQL

SELECT: Allows you to retrieve information from a table

Syntax:

```
SELECT col1, col2
FROM table WHERE [some condition]
```

Example:

```
SELECT poll_title, poll_date
FROM polls
WHERE romney_pct > obama_pct
```

SQL

GROUP BY: Allows you to aggregate information.

Syntax:

```
SELECT col1, AVG(col2)
FROM table GROUP BY col1
```

Example:

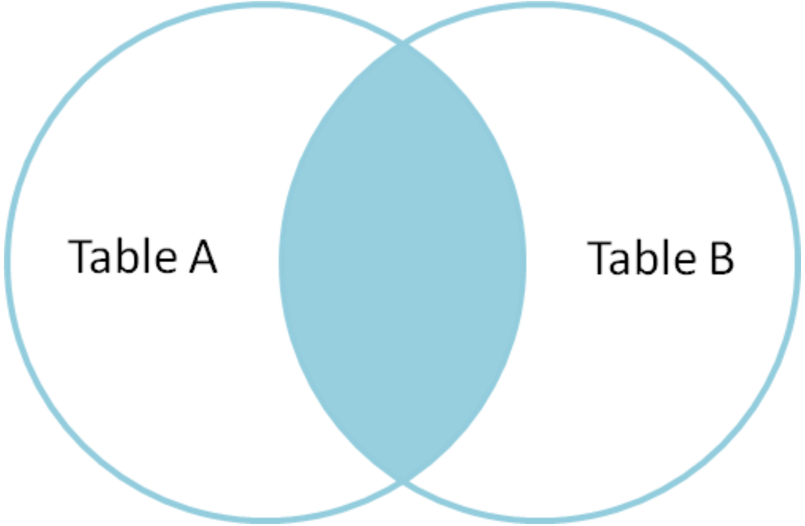
```
SELECT poll_date, AVG(obama_pct)
FROM polls GROUP BY poll_date
```

SQL

`JOIN`: Couple different kinds in Hive:

`JOIN:`

produces only the records that match both tables

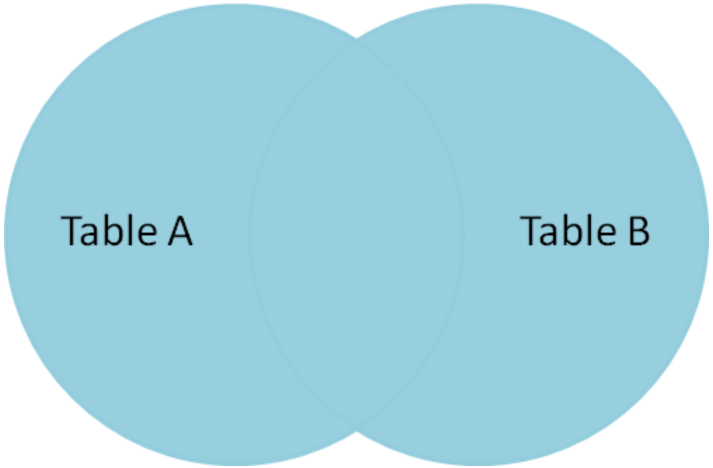


SQL

`JOIN`: Couple different kinds in Hive:

`FULL OUTER JOIN:`

produces all records in both tables, with matching records from both sides where available. If no match, the missing side will contain null

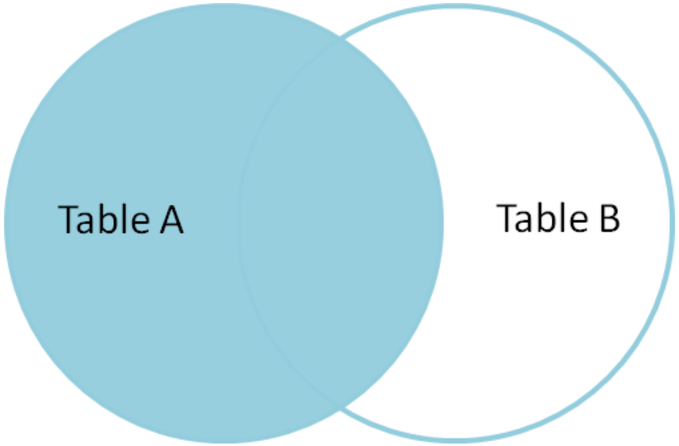


SQL

`JOIN`: Couple different kinds in Hive:

`LEFT OUTER JOIN (RIGHT OUTER JOIN):`

produces records from Table A, with matching records (where available) in Table B. If there is no match, the right side will contain null.



SQL

JOIN: Couple different kinds in Hive:

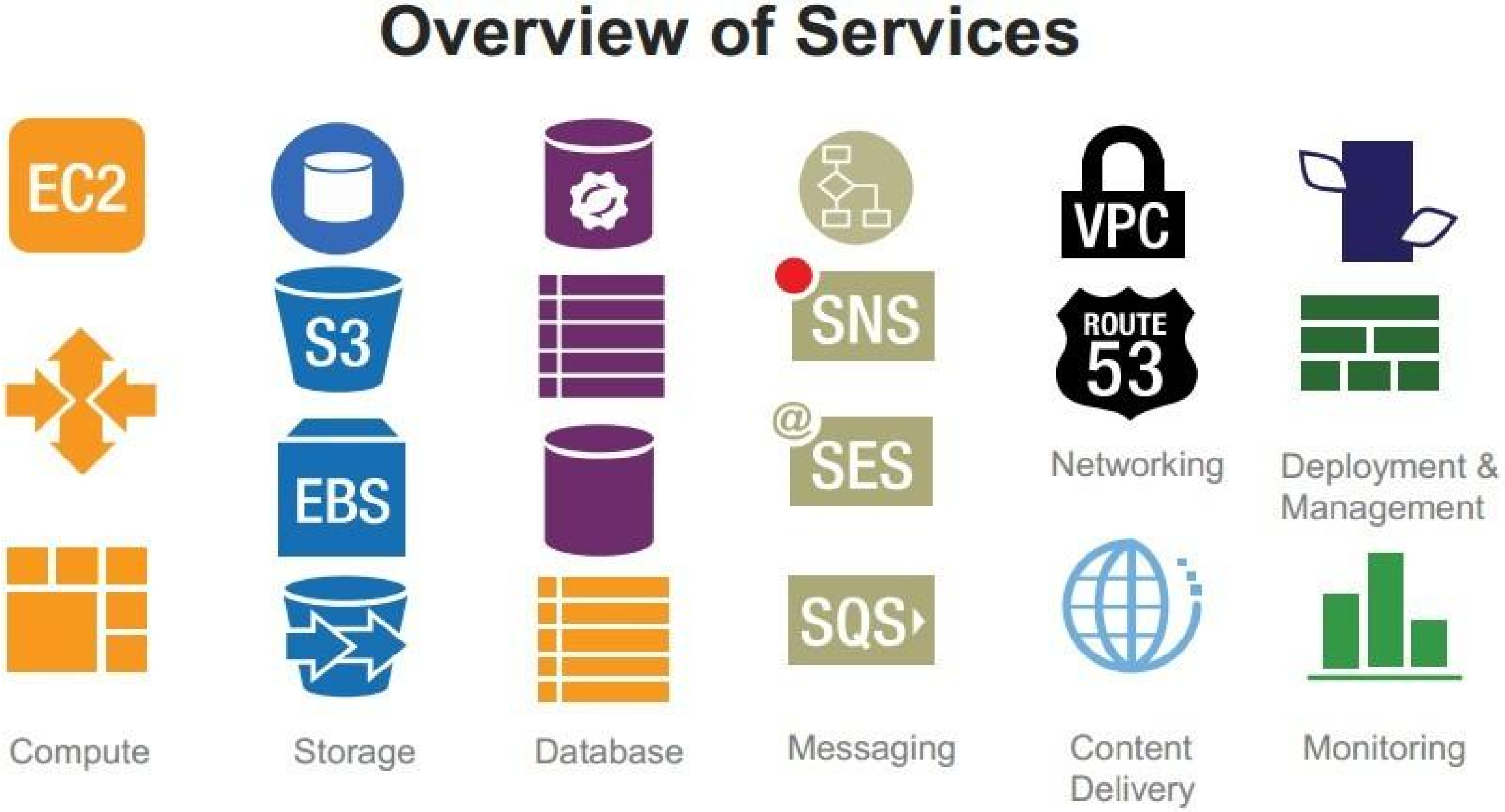
LEFT SEMI JOIN:
    produces only records from the left table

equivalent in vanilla SQL:

```
SELECT name FROM table_1 a WHERE EXISTS( SELECT *
FROM table_2 b WHERE (a.name=b.name))
```

AWS

AWS


Overview of Services
EC2
S3
EBS
SNS
@
SES
SQS
VPC
ROUTE
53
Networking
Deployment &
Management
Compute
Storage
Database
Messaging
Content
Delivery
Monitoring

AWS

Regions

- us-east-1 (North Virginia)
- us-west-1 (Northern California)
- ap-southeast-1 (Singapore)
- eu-west-1 (Ireland)
- also other regions in Asia, Europe, North and South America

Availability Zones (AZ)

- Each region has several availability zones

AWS

Common Services (for data science):

- S3: Simple Storage Service
- EC2: Elastic Cloud Compute
- EMR: Elastic MapReduce

AWS: S3 (SIMPLE STORAGE SERVICE)

- Sort of a file system
- Buckets ~ Folders, Keys ~ Files
  - Subdirectories aren't really folders, but rather just prefixes to the key name
- Region Specific -- but it usually doesn't matter (except for latency)
- Relatively Cheap
- Complex Permission Structure (about as complex as a real file system)

AWS: EC2 (Elastic Cloud Compute)

Instance Types:

- General Purpose
- Compute Optimized
- Memory Optimized
- Storage Optimized
- GPU Instances

AWS: EC2 (Elastic Cloud Compute)

Example Pricing for On-Demand Linux instances in US East

t2.micro • 1 CPU • 1 GB RAM • $0.013 per Hour	i2.8xlarge • 32 CPUs • 244 GB RAM • $6.82 per Hour
m3.xlarge • 4 CPUs • 15 GB RAM • $0.28 per Hour	r3.large • 2 CPUs • 15 GB RAM • $0.175 per Hour

Not all instance types available in all regions (e.g. High Storage Instances not available in us-west-1)

AWS: AMI (Amazon Machine Images)

- An ***AMI*** is a copyable snapshot of an instance's contents and configuration
- If you've ever run a virtual machine (e.g. VirtualBox, VMware, Parallels), you may be familiar
- EC2 instances start as AMIs that are then instantiated
- StarCluster uses AMIs extensively to preconfigure EC2 instances to create a cluster
- A well-configured ***AMI*** can save a lot of time
- ***AMI***s are region specific. Copies to other regions receive a new ID

AWS: EMR (Elastic MapReduce)

"Hadoop in the Cloud"

- Uses EC2 instances
- Web interface for provisioning cluster
- Installs Hadoop, etc. automatically
- Installs Hive, Pig, HBase automatically if desired
- Automatically configures Hadoop to be able to read your S3 buckets as if they were part of HDFS
- Charges a small premium on top of EC2 prices