Aurelius Corporate Solutions

Course- Big Data Hadoop (Basic)

Aurelius Corporate Solutions
A-125, Sector 63, Noida, Uttar Pradesh – 201307, India

1000+ Unique Technologies Projects Delivered | 500+ Corporate Customers Worldwide | 50000+ Professionals Trained on 40+ Domains in Over 30 Countries

Course Topics

- ✓ Welcome to Big Data World
  - ✓ Understanding Big Data
  - ✓ Hadoop Architecture
- ✓ HDFS
  - ✓ Deep dive in HDFS Architecture
  - ✓ HDFS APIs
  - ✓ Introduction to HDP Sandbox
  - ✓ HDFS Hands – 1 Hour
- ✓ Introduction to YARN & MR
  - ✓ Hadoop MapReduce framework
  - ✓ Programming in Map Reduce
- ✓ Advance Map Reduce
  - ✓ Understanding Counters
  - ✓ Differences between MR1 & MR2
  - ✓ Introduction to MR API
  - ✓ Overview of Map Side Join
  - ✓ Overview of Reduce Side Join
  - ✓ Map Reduce Hands On – 1 hour
- ✓ Hive
  - ✓ Analytics using Hive
  - ✓ Understanding HIVE QL
- ✓ Advanced Hive
  - ✓ Advance Hive
  - ✓ Hive Hands On – 1 Hour
- ✓ NoSQL & HBase
  - ✓ CAP Theorem
  - ✓ NoSQL Databases and HBASE
  - ✓ HBase Architecture
  - ✓ HBase Schema Design
  - ✓ Difference between Hive & Hbase
  - ✓ Hbase Hands On – 1 Hour
- ✓ Apache Spark
  - ✓ Introduction to Spark
  - ✓ Why Spark?
  - ✓ Spark Stack Overview
  - ✓ Overview of RDD, Data Frame & Data Set
  - ✓ Spark Actions & Transformation Overview

Topics for Day 1: Covered

- ✓ Team Introduction
- ✓ Introduction to Big Data – Why and What?
- ✓ Characteristics of Big Data (4Vs)
- ✓ Overview of Big Data Ecosystem
- ✓ What is Hadoop?
- ✓ History of Hadoop

Tea Break

- ✓ Components of Hadoop
- ✓ Introduction to HDFS
- ✓ HDFS Architecture – Name Node / Data Node, Concept of Blocks
- ✓ File Formats in Hadoop
- ✓ HDFS API walk through
- ✓ Anatomy of a File Write and Read

Lunch Break

- ✓ Overview of Lab environment – HDP sandbox etc.
- ✓ HDFS Hands on – Getting Familiar with HDFS most commonly used commands
- ✓ Introduction to Map Reduce
- ✓ Map Reduce Phases – Map, Shuffle-Sort and Reduce
- ✓ Map Reduce Job Submission Flow

Topics for Day 2: Covered

- ✓ Any question from Day 1
- ✓ Understanding Counters
- ✓ Difference Between MR1 & MR2

Tea Break

- ✓ Job Class, GenericOptionsParser, Mapper & Reducer
- ✓ Distributed Cache
- ✓ Custom Input Format
- ✓ Overview of Map Side Join & Reduce Side Join

Lunch Break

- ✓ Map Reduce Hands On
- ✓ Data Integration Choices – Sqoop, Flume
- ✓ Introduction to Hive
  - ✓ Hive Architecture
  - ✓ Working with Schema
  - ✓ Introduction to Hive QL
  - ✓ Partitioning & Bucketing

Topics for Today (Day 3)

- ✓ Any question from Day 2
- ✓ NoSQL & HBase
  - ✓ CAP Theorem
  - ✓ NoSQL Databases and HBASE

Tea Break

- ✓ NoSQL & HBase
  - ✓ HBase Architecture
  - ✓ HBase Schema Design
  - ✓ Difference between Hive & Hbase
  - ✓ Hbase Hands On – 1 Hour

Lunch Break

- ✓ Apache Spark
  - ✓ Introduction to Spark
  - ✓ Why Spark?
  - ✓ Spark Stack Overview
  - ✓ Overview of RDD, Data Fram
  - ✓ Spark Actions & Transformation Overview

Recap

- Create a partitioned table using PARTITIONED BY

```
CREATE EXTERNAL TABLE accounts_by_state(
    cust_id INT,
    fname STRING,
    lname STRING,
    address STRING,
    city STRING,
    state STRING,
    zipcode STRING)
  PARTITIONED BY (state STRING)
  ROW FORMAT DELIMITED
  FIELDS TERMINATED BY ','
  LOCATION '/loudacre/accounts_by_state';
```

Partition Columns

- **The partition column is displayed if you `DESCRIBE` the table**

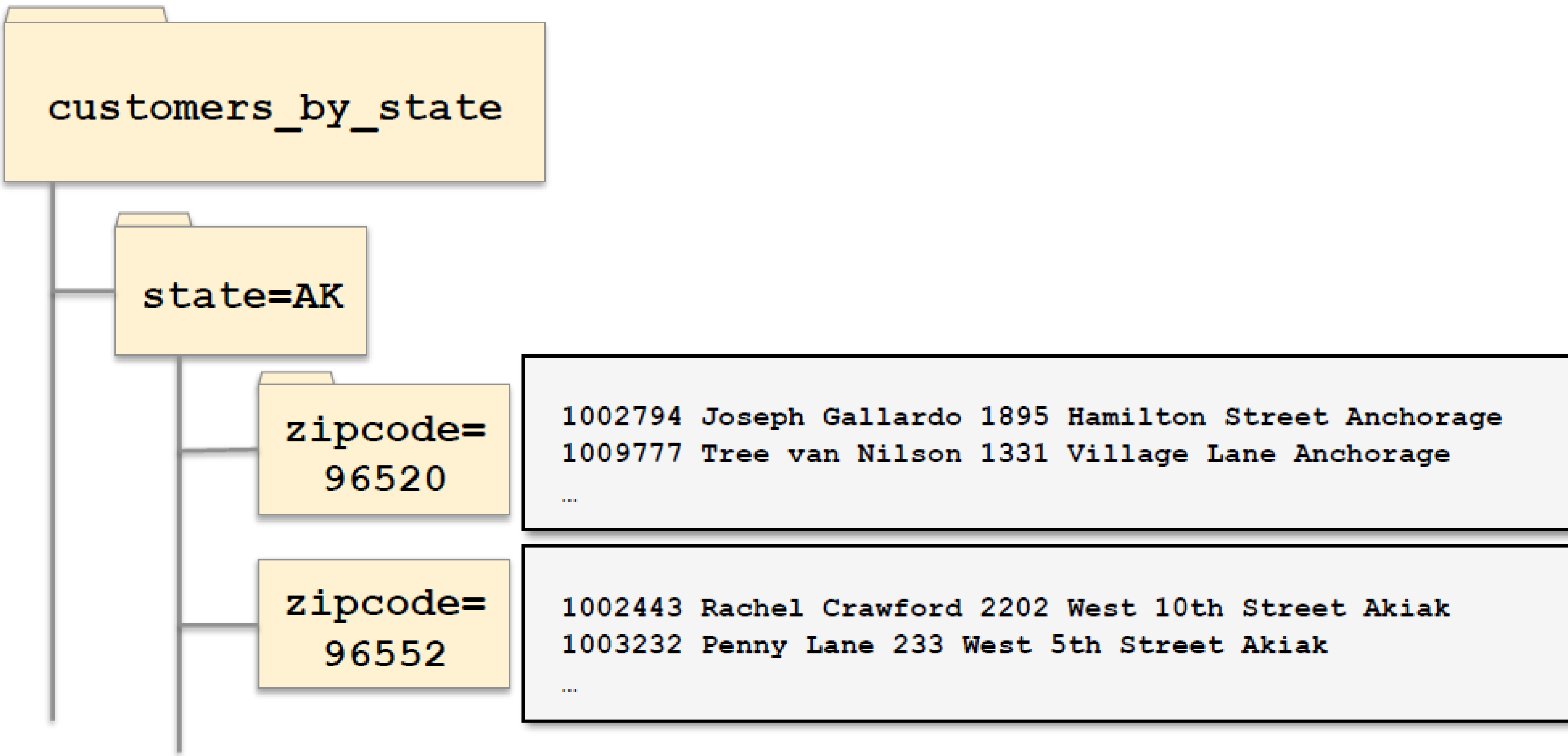
```
DESCRIBE accounts_by_state;
+----------+---------+----------+
| name     | type    | comment  |
+----------+---------+----------+
| cust_id  | int     |          |
| fname    | string  |          |
| lname    | string  |          |
| address  | string  |          |
| city     | string  |          |
| zipcode  | string  |          |
| state    | string  |          |
+----------+---------+----------+
```

A partition column is a “virtual column”; data is not stored in the file

Nested Partitions

- You can also created nested partitions

```
… PARTITIONED BY (state STRING, zipcode STRING)
```



Loading Data Into a Partitioned Table

- **Dynamic partitioning**
  - Impala/Hive add new partitions automatically as needed at load time
  - Data is stored into the correct partition (subdirectory) based on column value
- **Static partitioning**
  - You define new partitions using **`ADD PARTITION`**
  - When loading data, you specify which partition to store data in

Dynamic Partitioning

- **We can create new partitions dynamically from existing data**

```
INSERT OVERWRITE TABLE accounts_by_state
  PARTITION(state)
  SELECT cust_id, fname, lname, address,
    city, zipcode, state FROM accounts;
```

- **Partitions are automatically created based on the value of the *last* column**
  - If the partition does not already exist, it will be created
  - If the partition does exist, it will be overwritten

- **In older versions of Hive, dynamic partitioning is not enabled by default**
  - Enable it by setting these two properties

```
SET hive.exec.dynamic.partition=true;
SET hive.exec.dynamic.partition.mode=nonstrict;
```

- **Note: Hive variables set in Beeline are for the current session only**
  - Your system administrator can configure settings permanently

Creating Partitions from Existing Partition Directories in HDFS

- **Partition directories in HDFS can be created and populated outside Hive or Impala**
  - For example, by a Spark or MapReduce application
- **In Hive, use the `MSCK REPAIR TABLE` command to create (or recreate) partitions for an existing table**

```
MSCK REPAIR TABLE call_logs;
```

NoSQL


1980
1990
2000
2010
Rise of relational

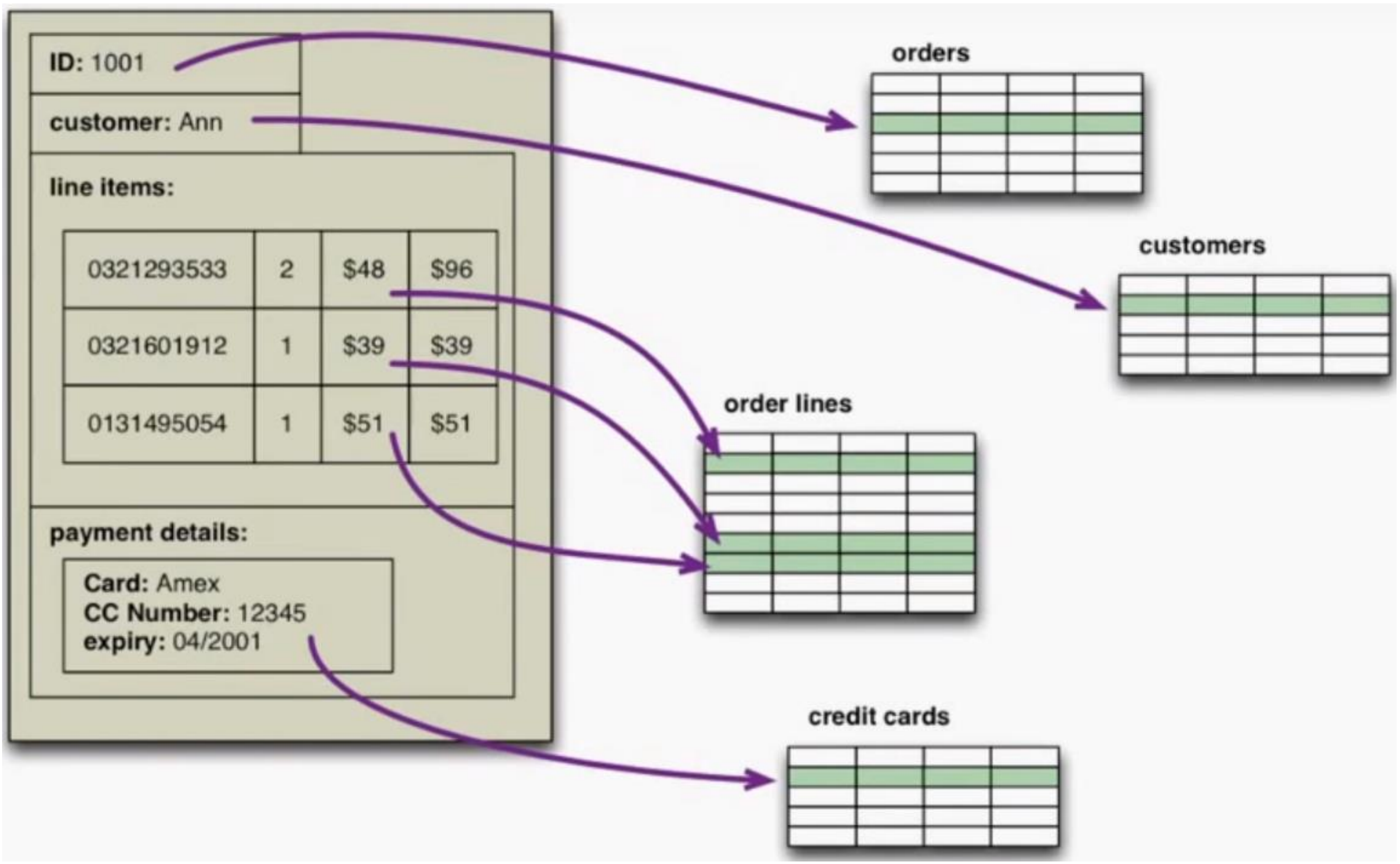
Persistence

Integration

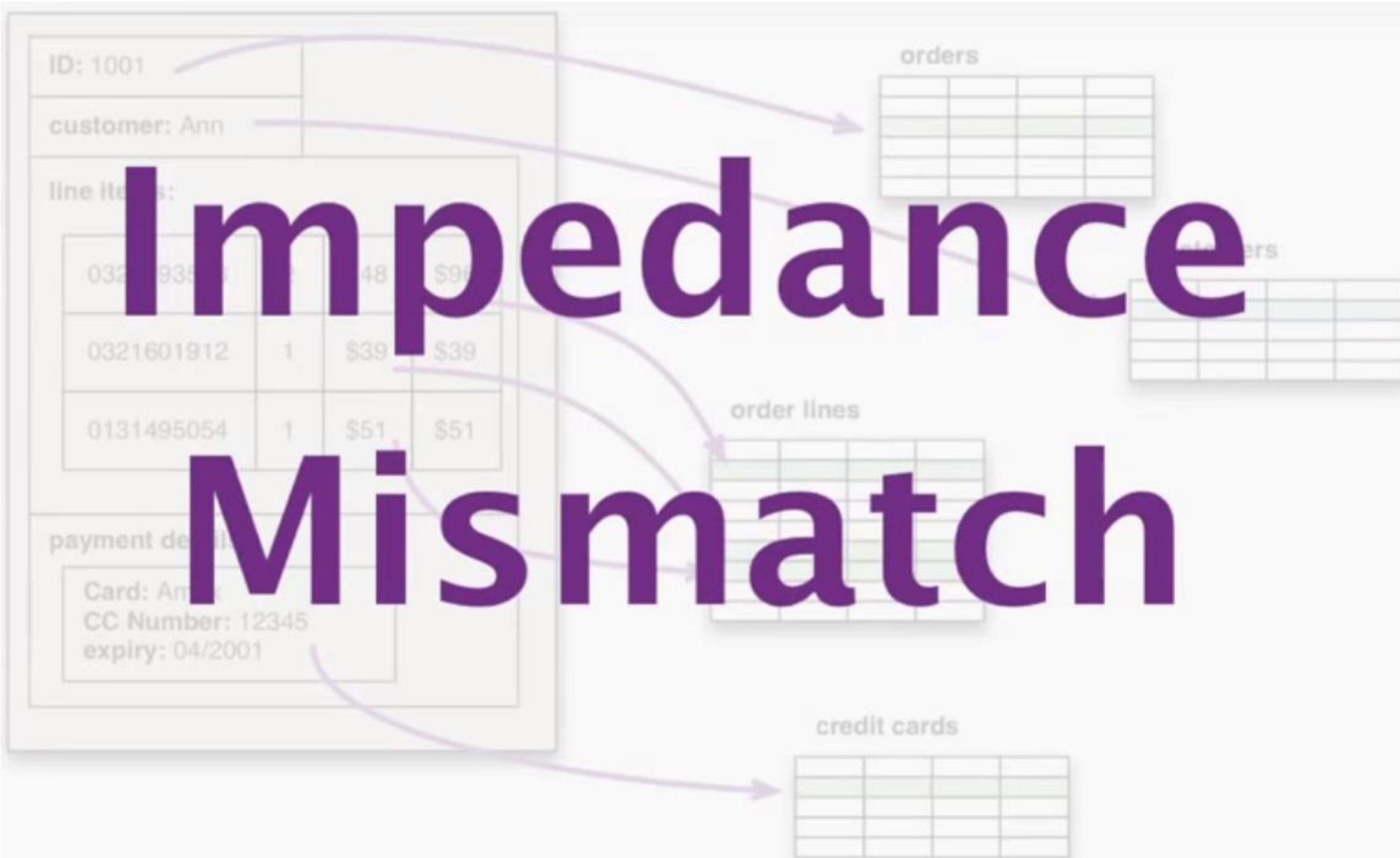
SQL

Transactions

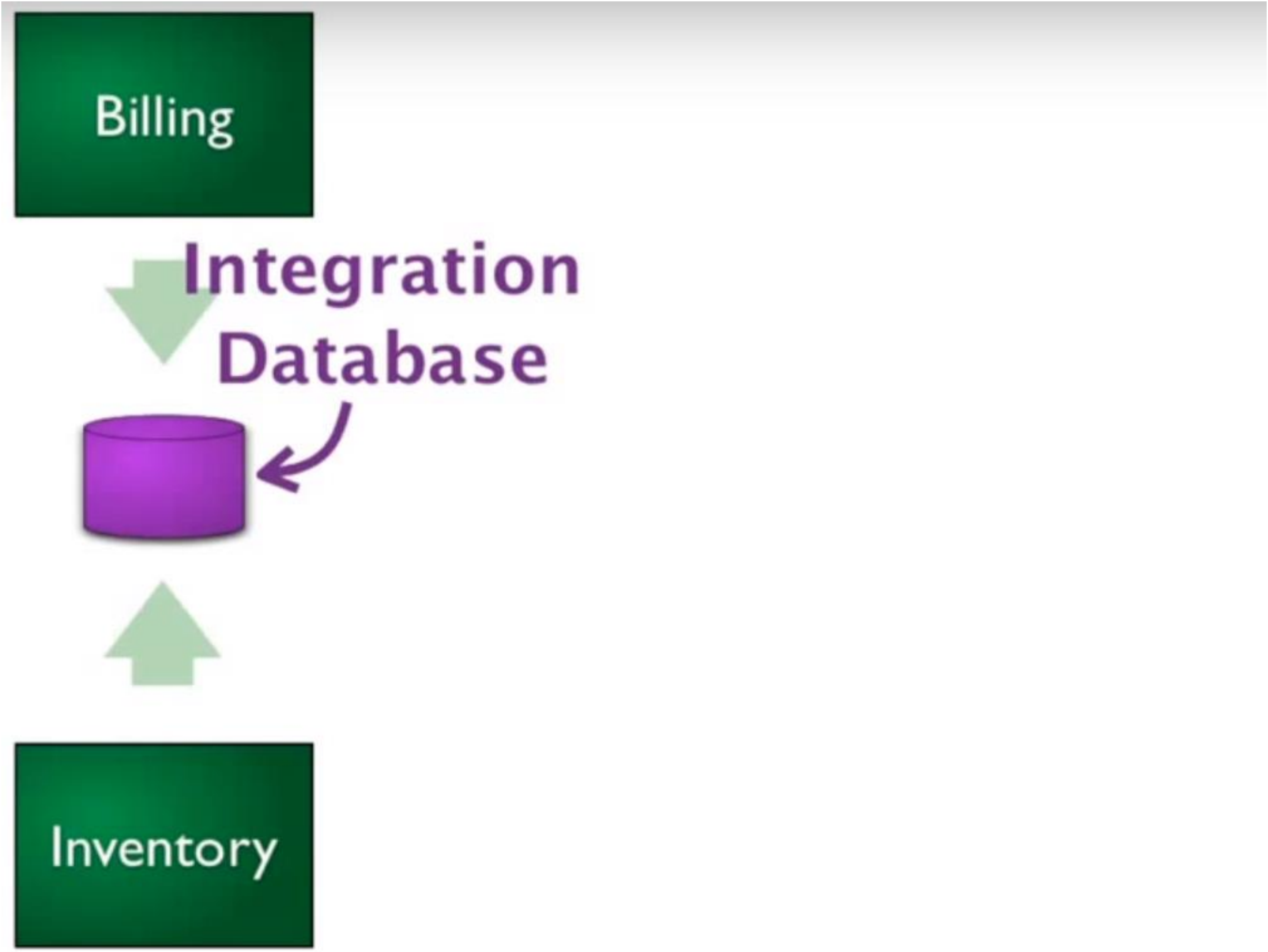
Reporting

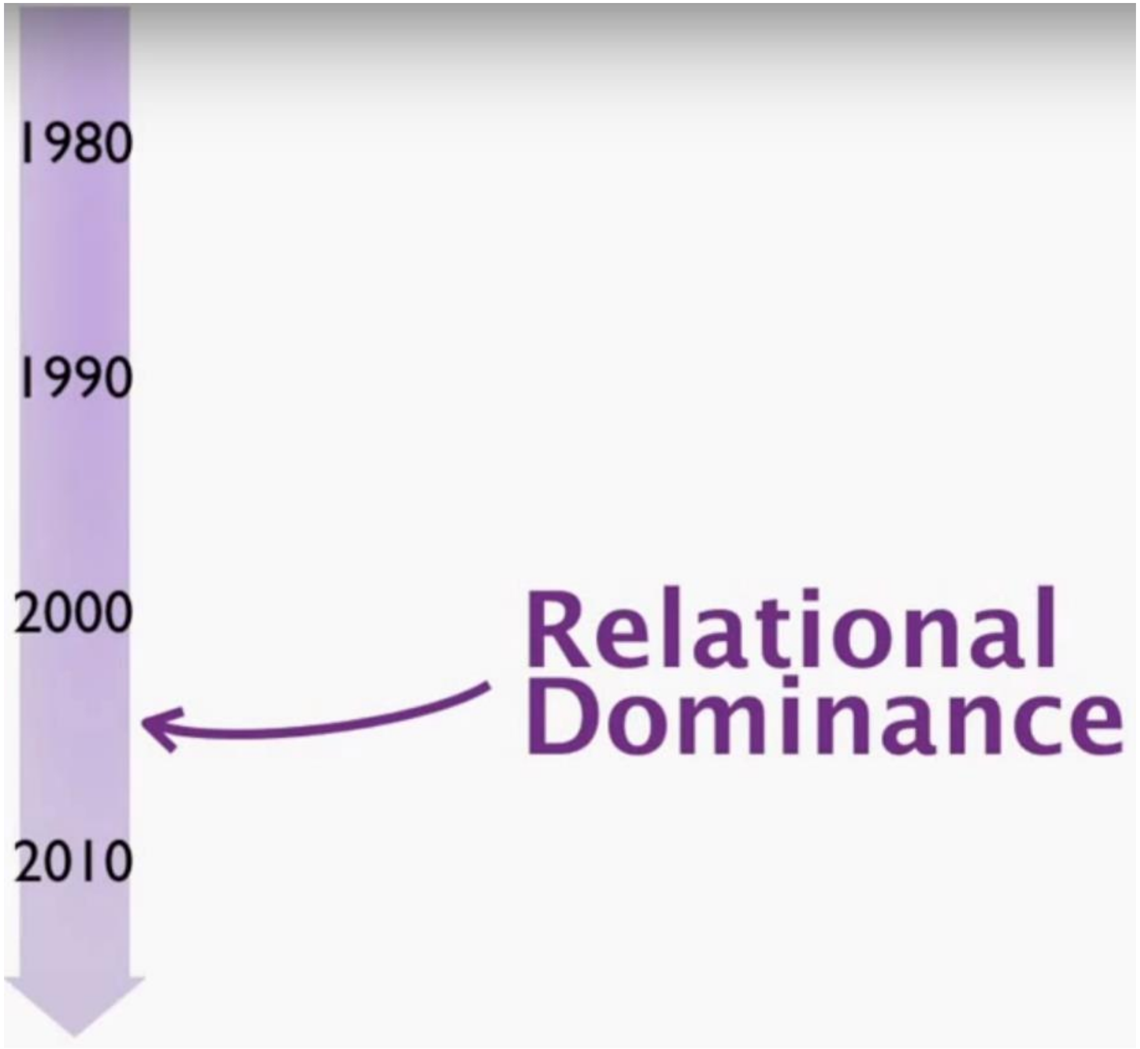

ID: 1001
customer: Ann
line items:
0321293533 2 $48 $96
0321601912 1 $39 $39
0131495054 1 $51 $51
payment details:
Card: Amex
CC Number: 12345
expiry: 04/2001
orders
customers
order lines
credit cards

Impedance Mismatch

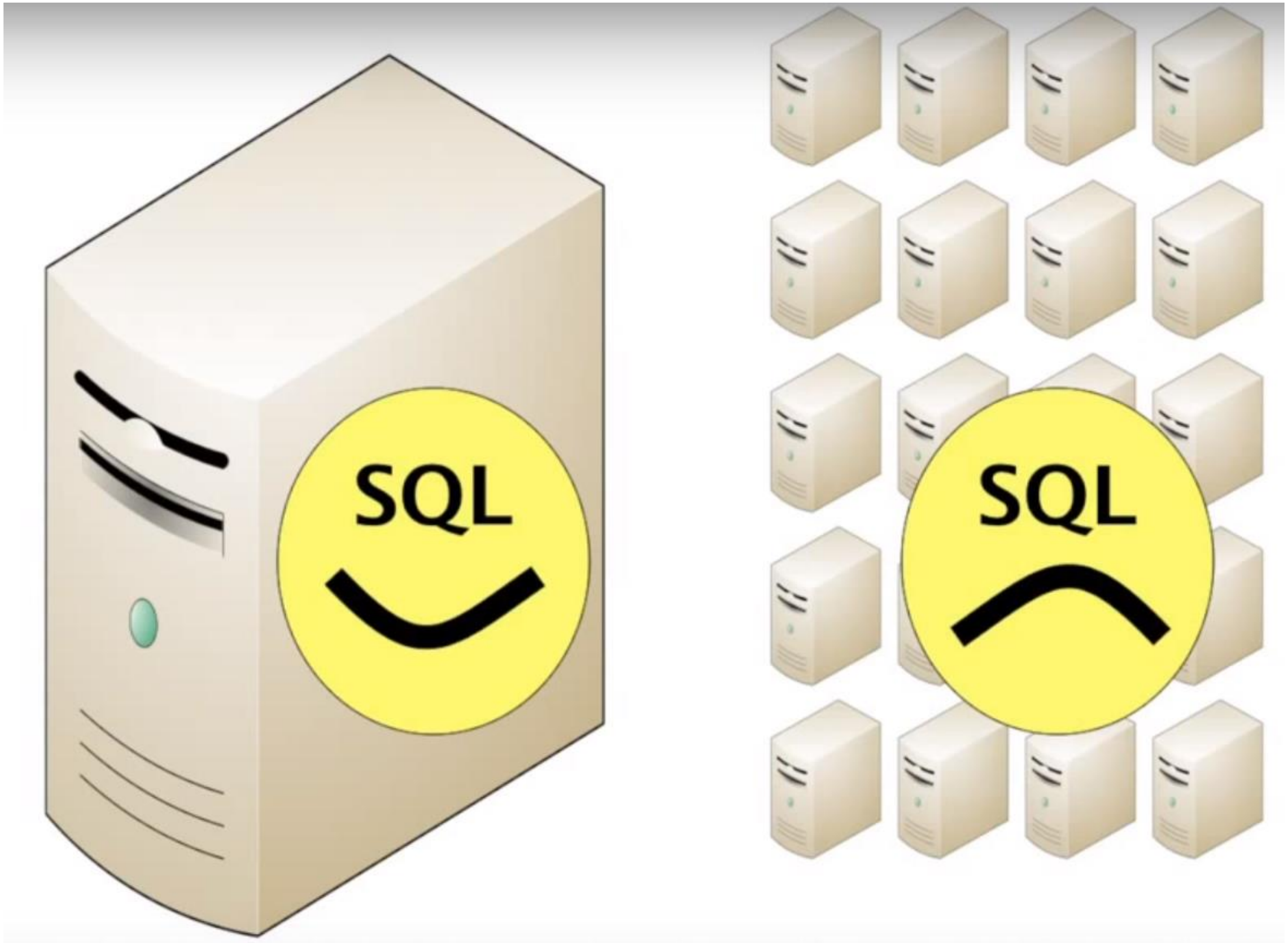



1980
1990
2000
2010
Rise of object databases

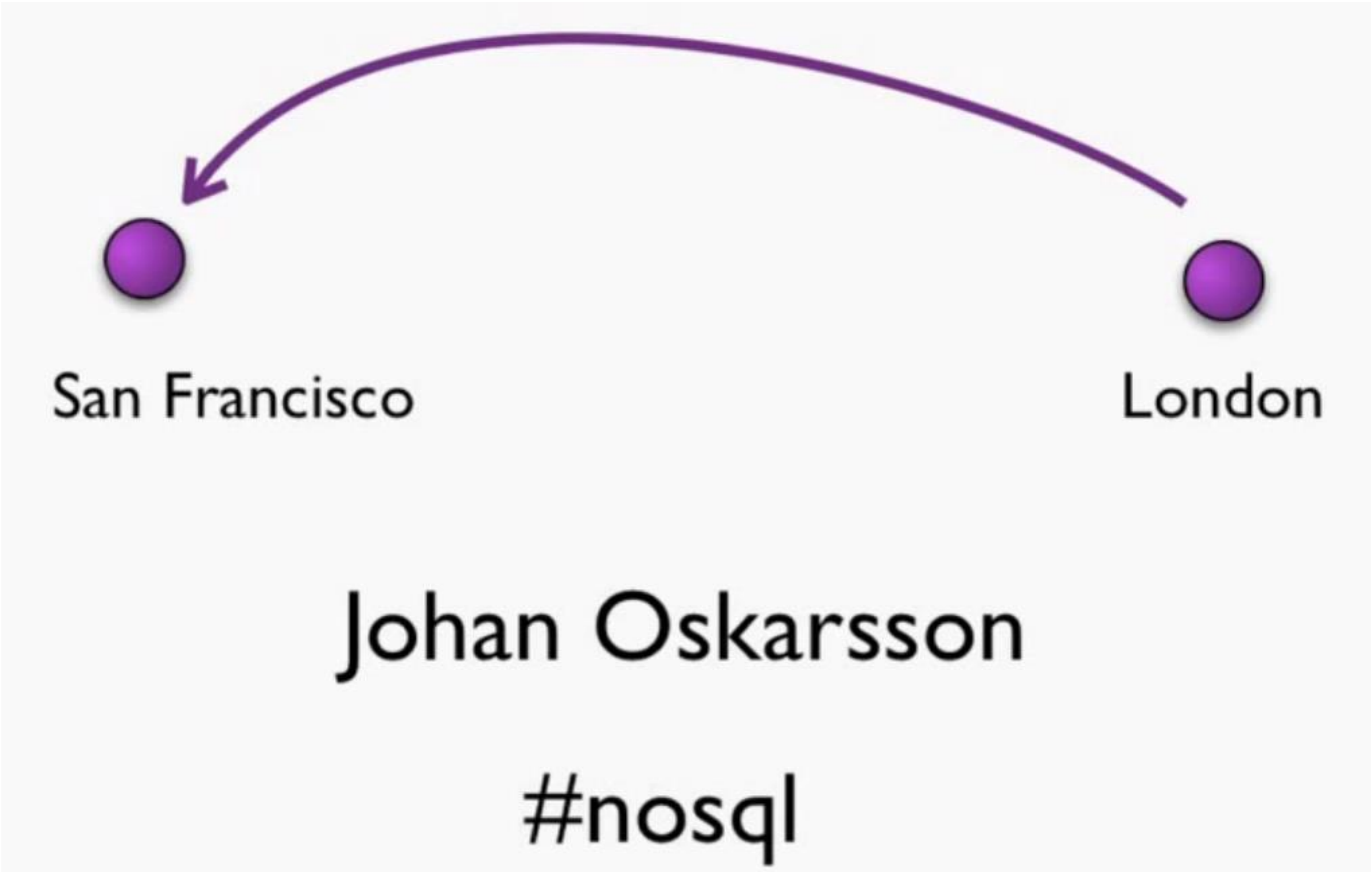

Billing
Integration Database
Inventory

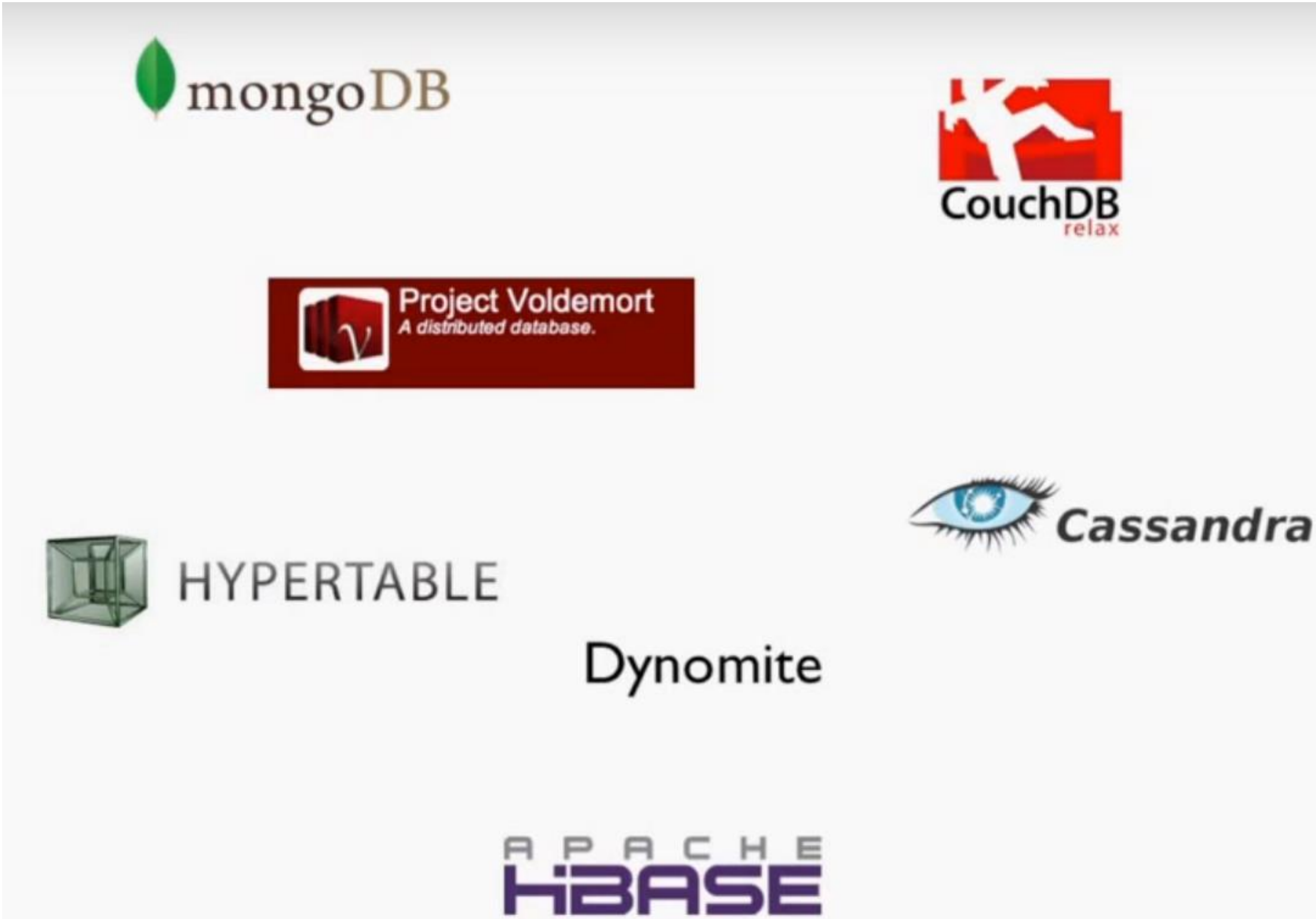

1980
1990
2000
2010
Relational
Dominance


Lots of
Traffic


SQL
SQL

Aurelius™ CORPORATE SOLUTIONS


San Francisco
London
Johan Oskarsson
#nosql



Aurelius™
CORPORATE SOLUTIONS

Characteristics of NoSQL

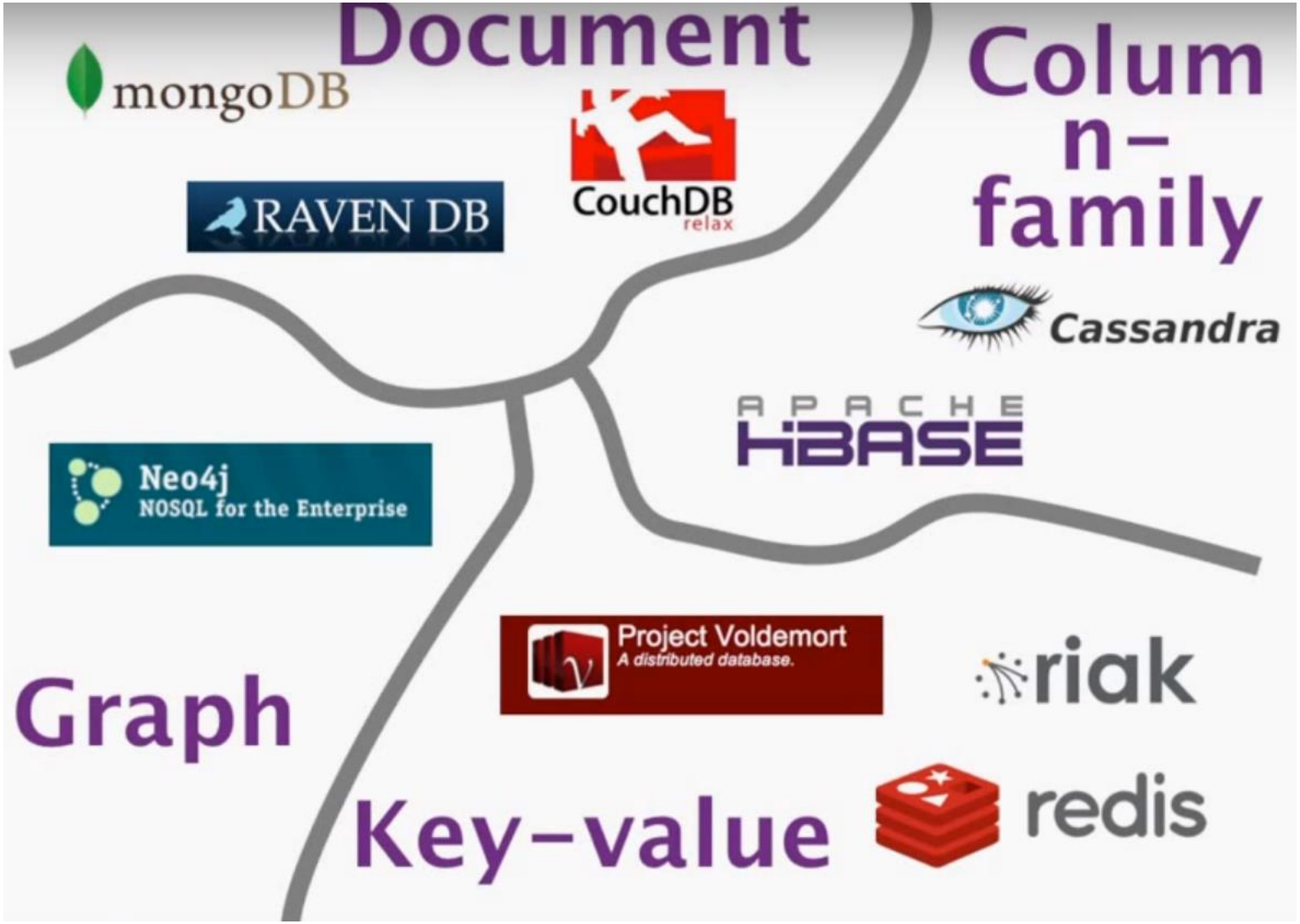
non-relational

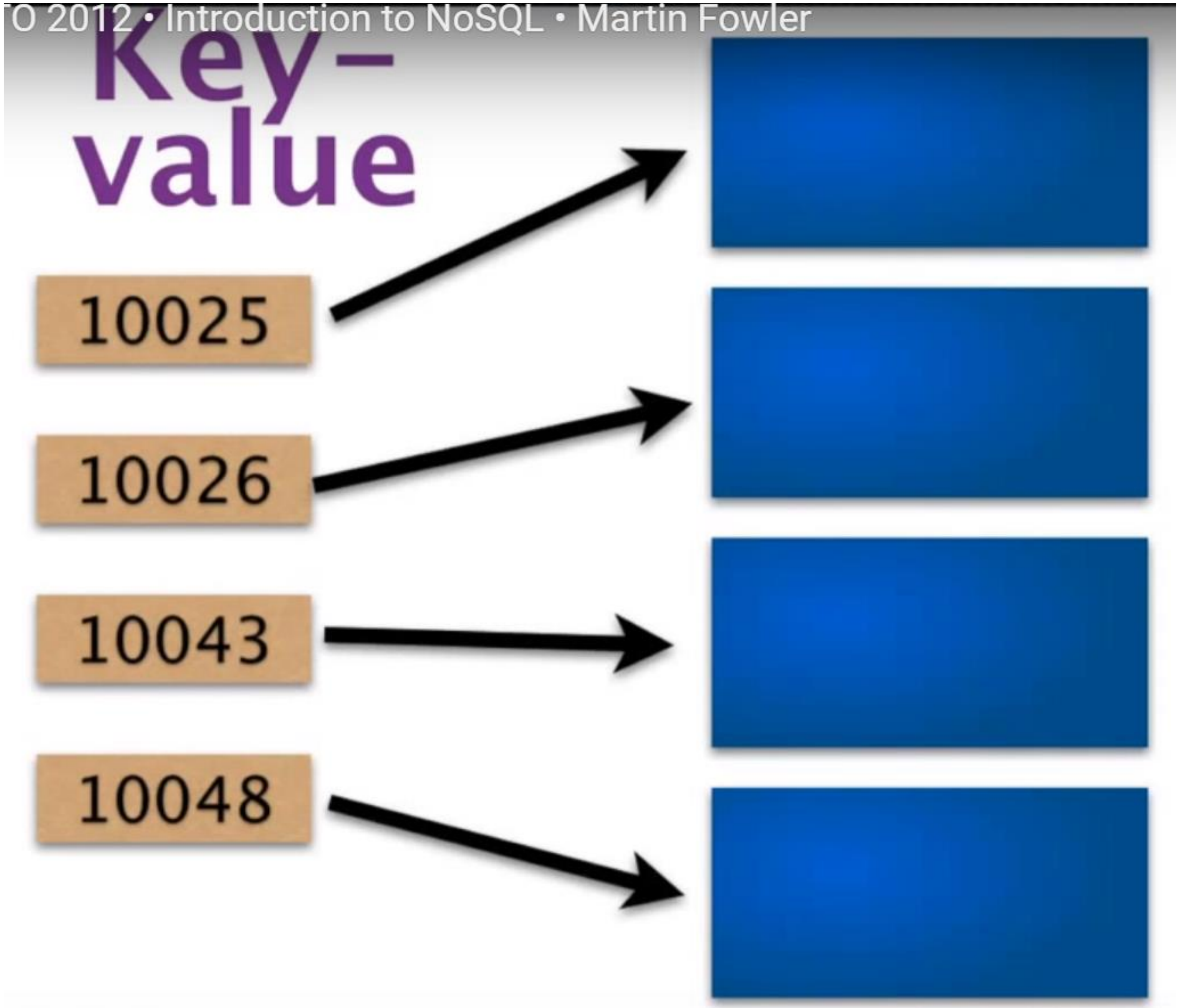
open-source

cluster-friendly

21st Century Web

schema-less


Document
mongoDB
RAVEN DB
CouchDB
relax
Column-family
Cassandra
APACHE
HBASE
Neo4j
NOSQL for the Enterprise
Project Voldemort
A distributed database.
riak
Graph
Key-value
redis

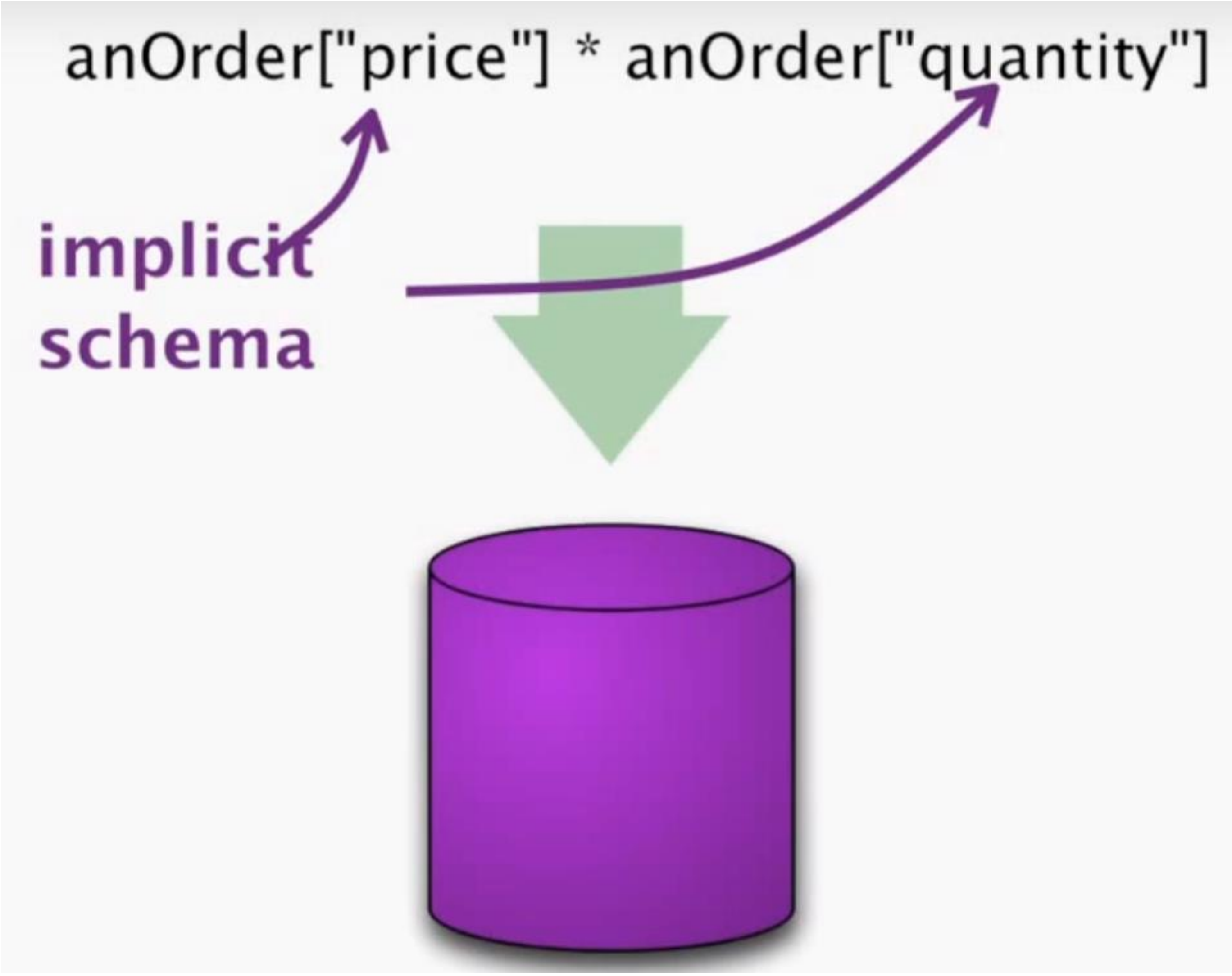

O 2012 • Introduction to NoSQL • Martin Fowler
Key-value
10025
10026
10043
10048

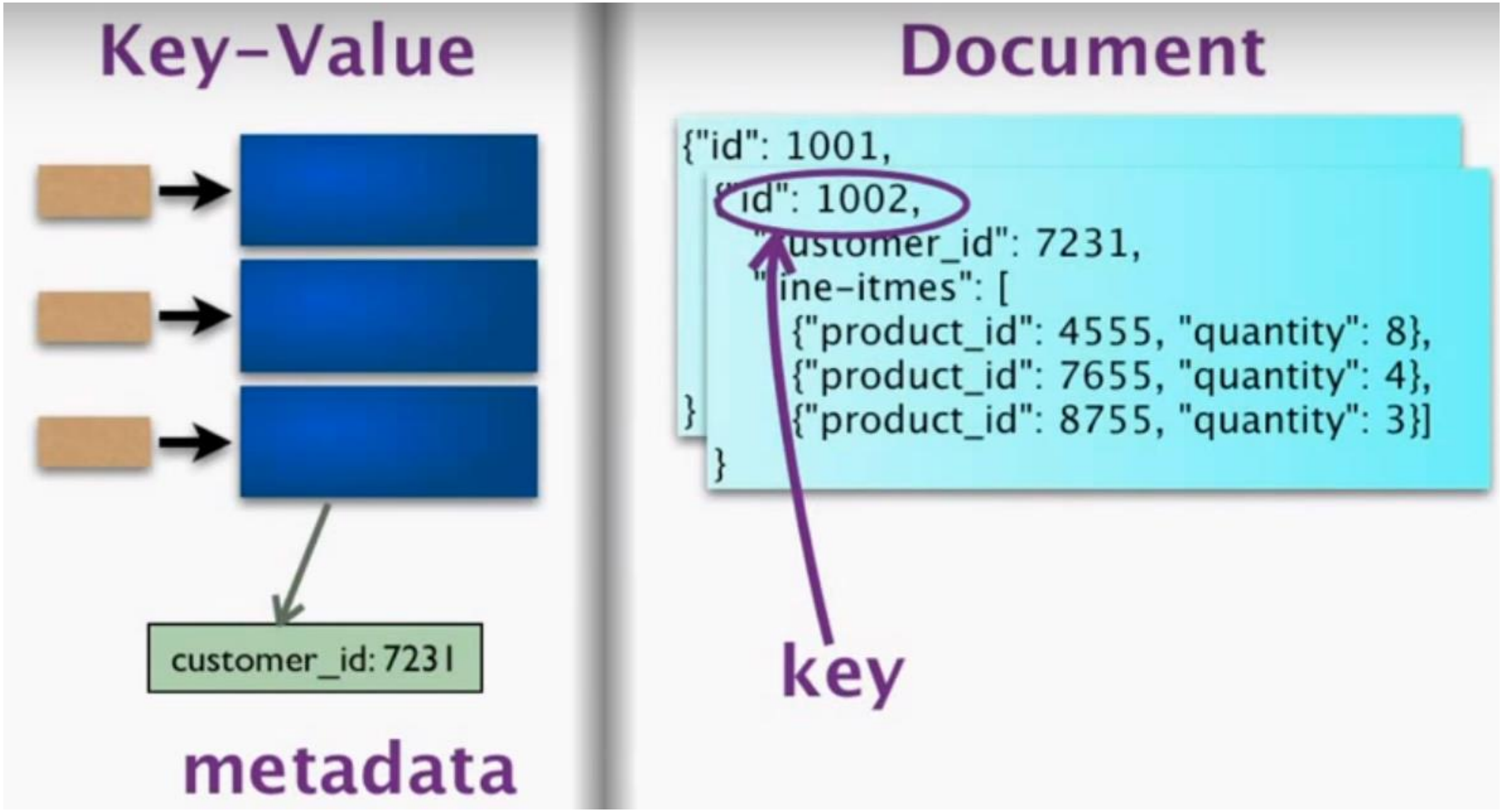
Document

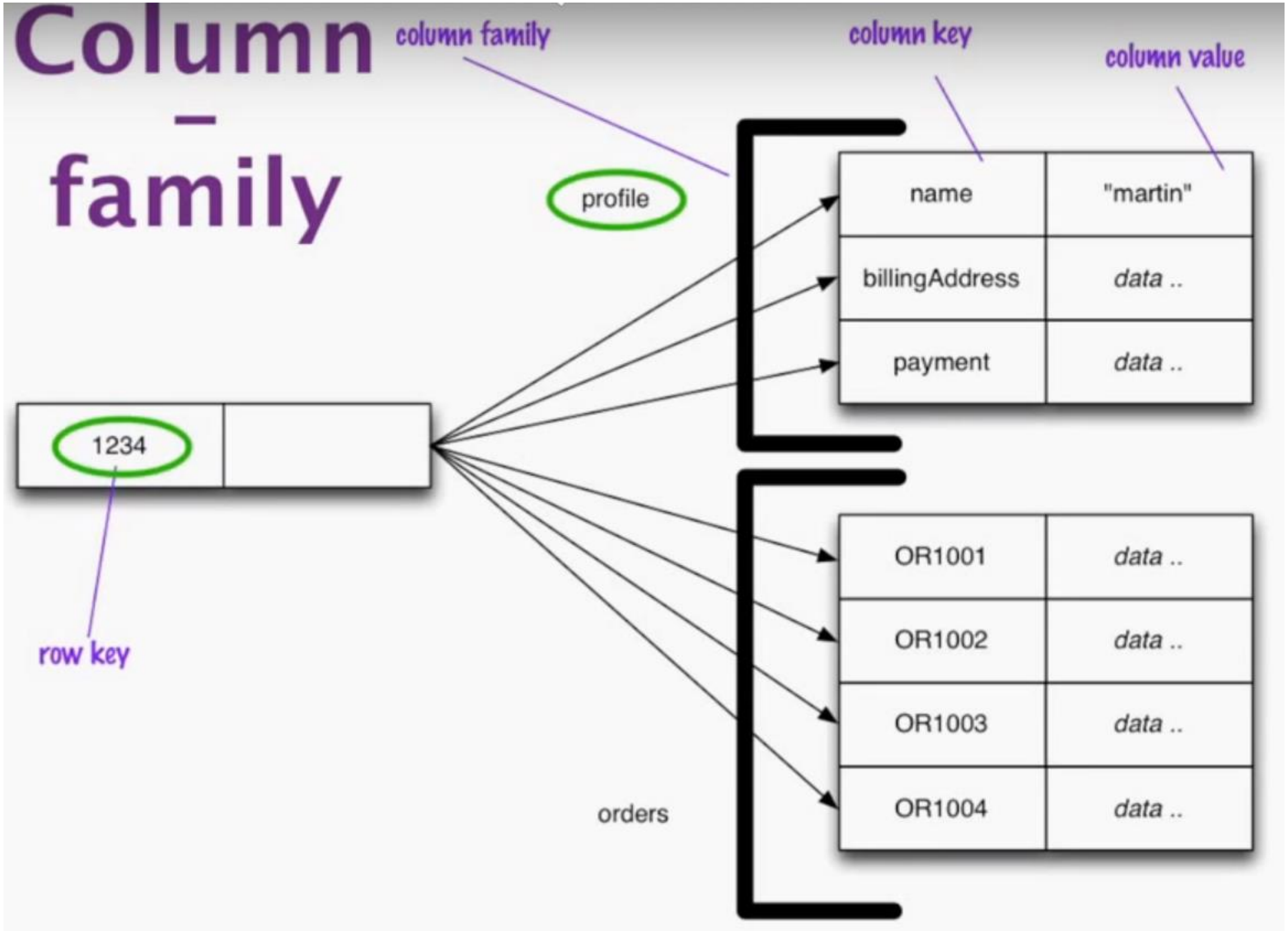
```
{"id": 1001,
"customer_id": 7231,
"line-itmes": [
{"product_id": 4555, "quantity": 8},
{"product_id": 7655, "quantity": 4}, {"product_id": 8755,
```

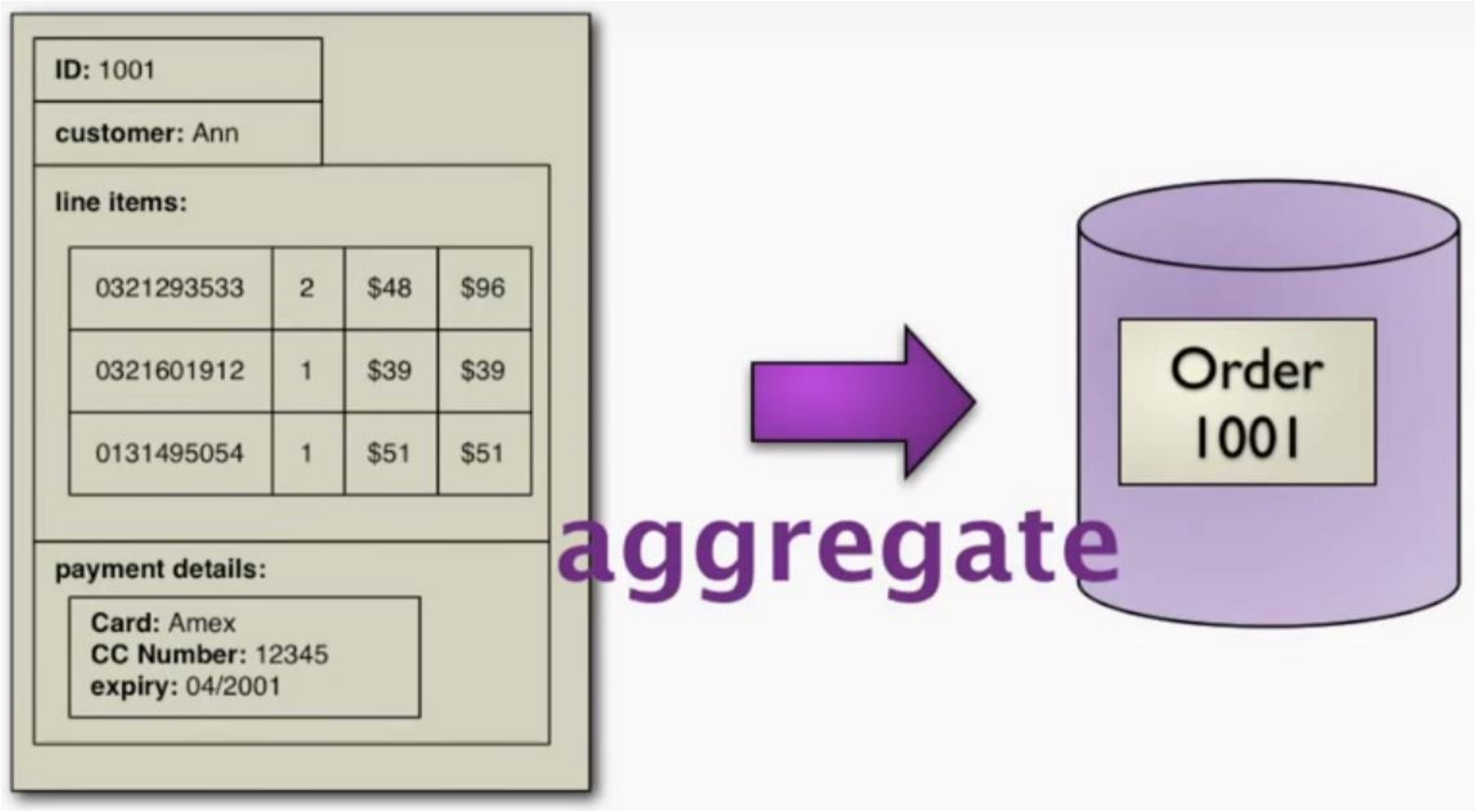
```
{"id": 1002,
"customer_id": 9831,
"line-itmes": [
{"product_id": 4555, "quantity": 3},
{"product_id": 2155, "quantity": 4}],
"discount-code": "Y"}
```

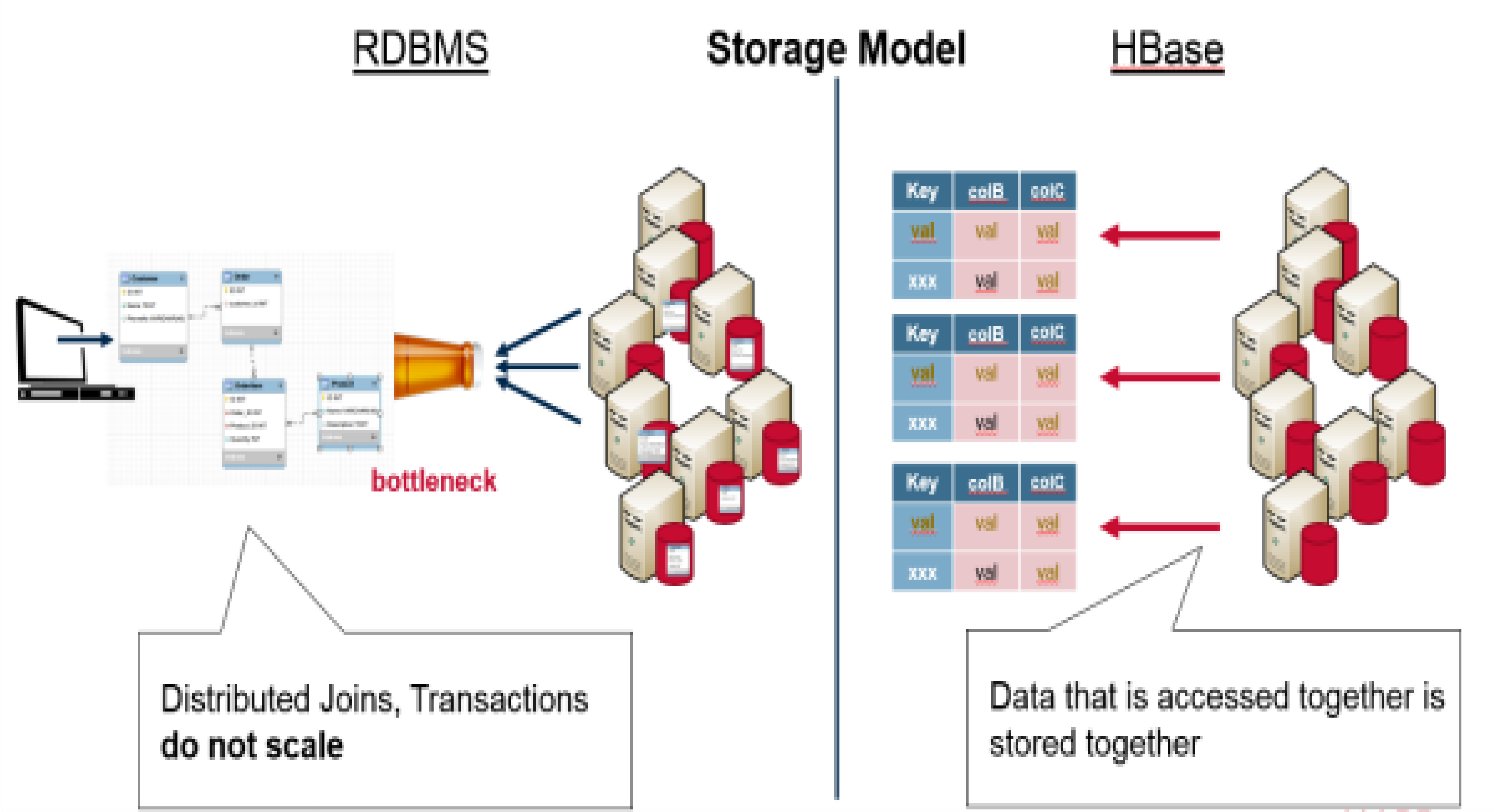
no schema


anOrder["price"] * anOrder["quantity"]
implicit
schema

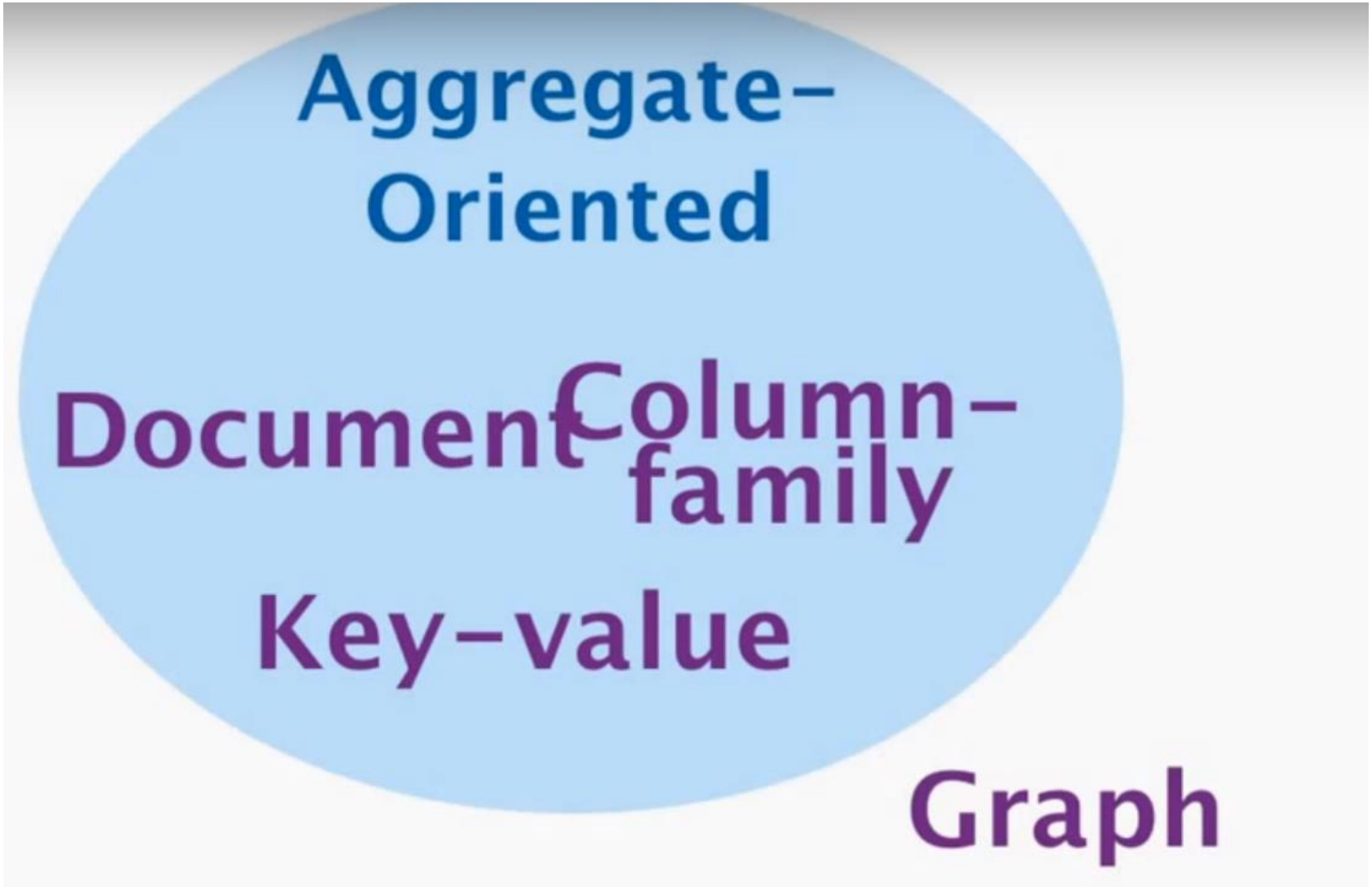

Key-Value
customer_id: 7231
metadata
Document
{"id": 1001,
}
{"id": 1002,
"customer_id": 7231,
"line-itmes": [
{"product_id": 4555, "quantity": 8},
{"product_id": 7655, "quantity": 4},
{"product_id": 8755, "quantity": 3}]
}
key

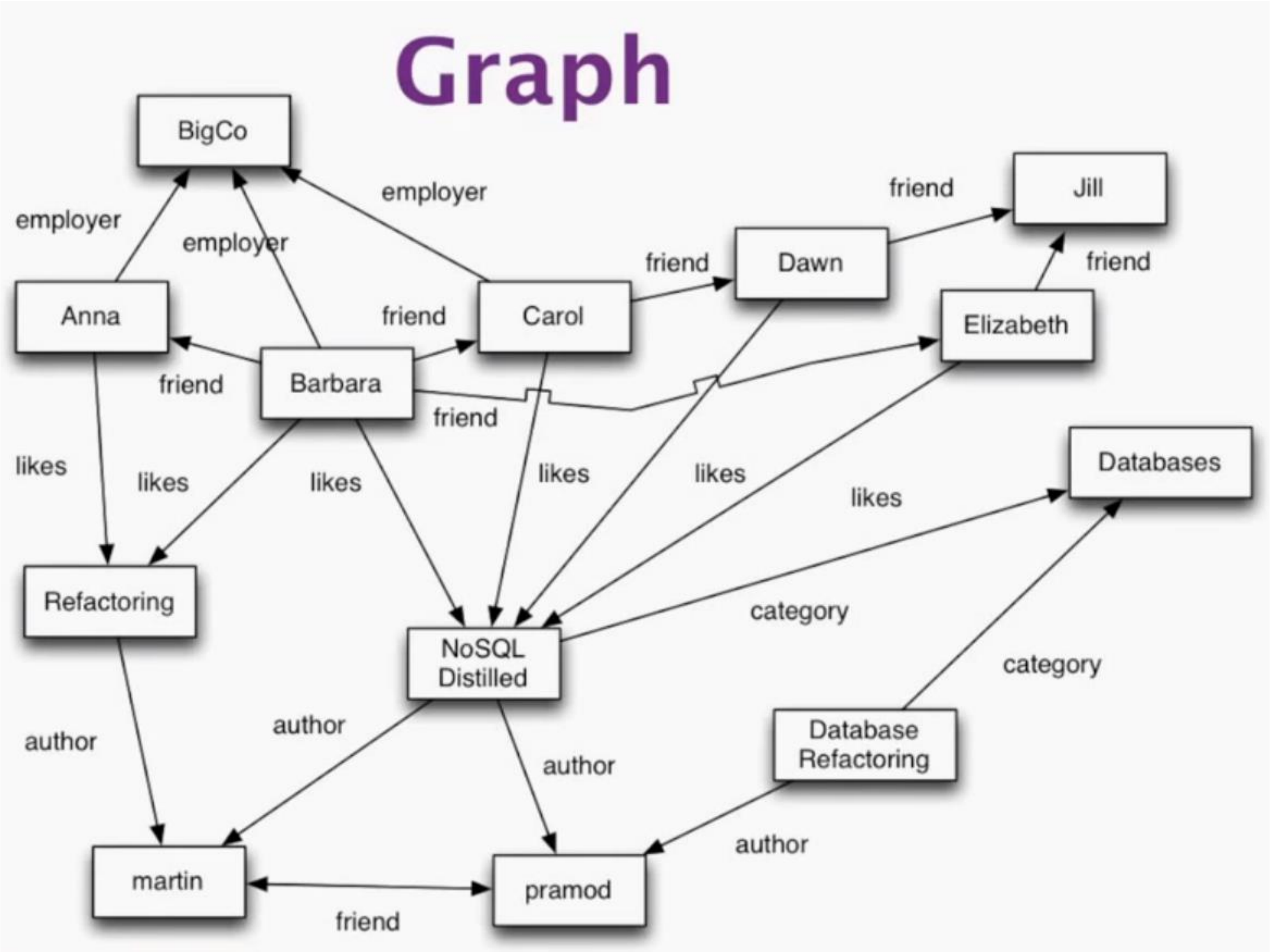

Column – family
column family
column key
column value
profile
name
"martin"
billingAddress
data ..
payment
data ..
1234
row key
orders
OR1001
data ..
OR1002
data ..
OR1003
data ..
OR1004
data ..


ID: 1001
customer: Ann
line items:
0321293533 2 $48 $96
0321601912 1 $39 $39
0131495054 1 $51 $51
payment details:
Card: Amex
CC Number: 12345
expiry: 04/2001
aggregate
Order
1001

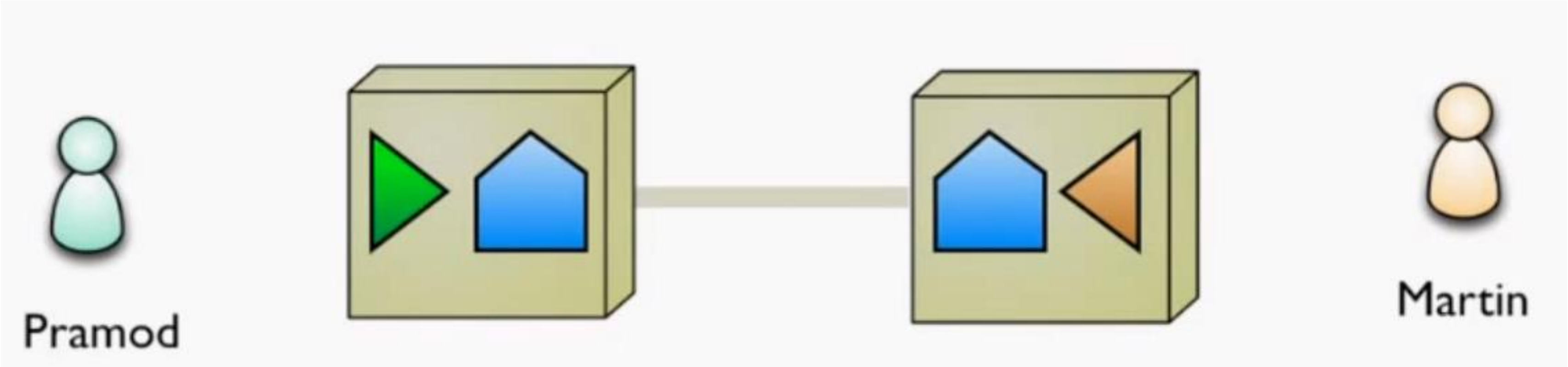


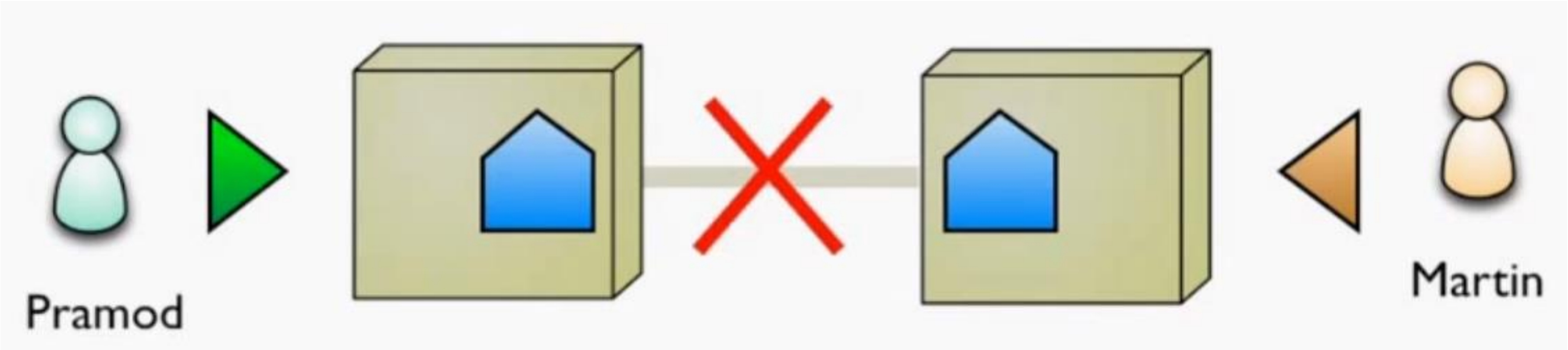
RDBMS
Storage Model
HBase
bottleneck
Key
colB
colC
val
xxx
Distributed Joins, Transactions
do not scale
Data that is accessed together is
stored together

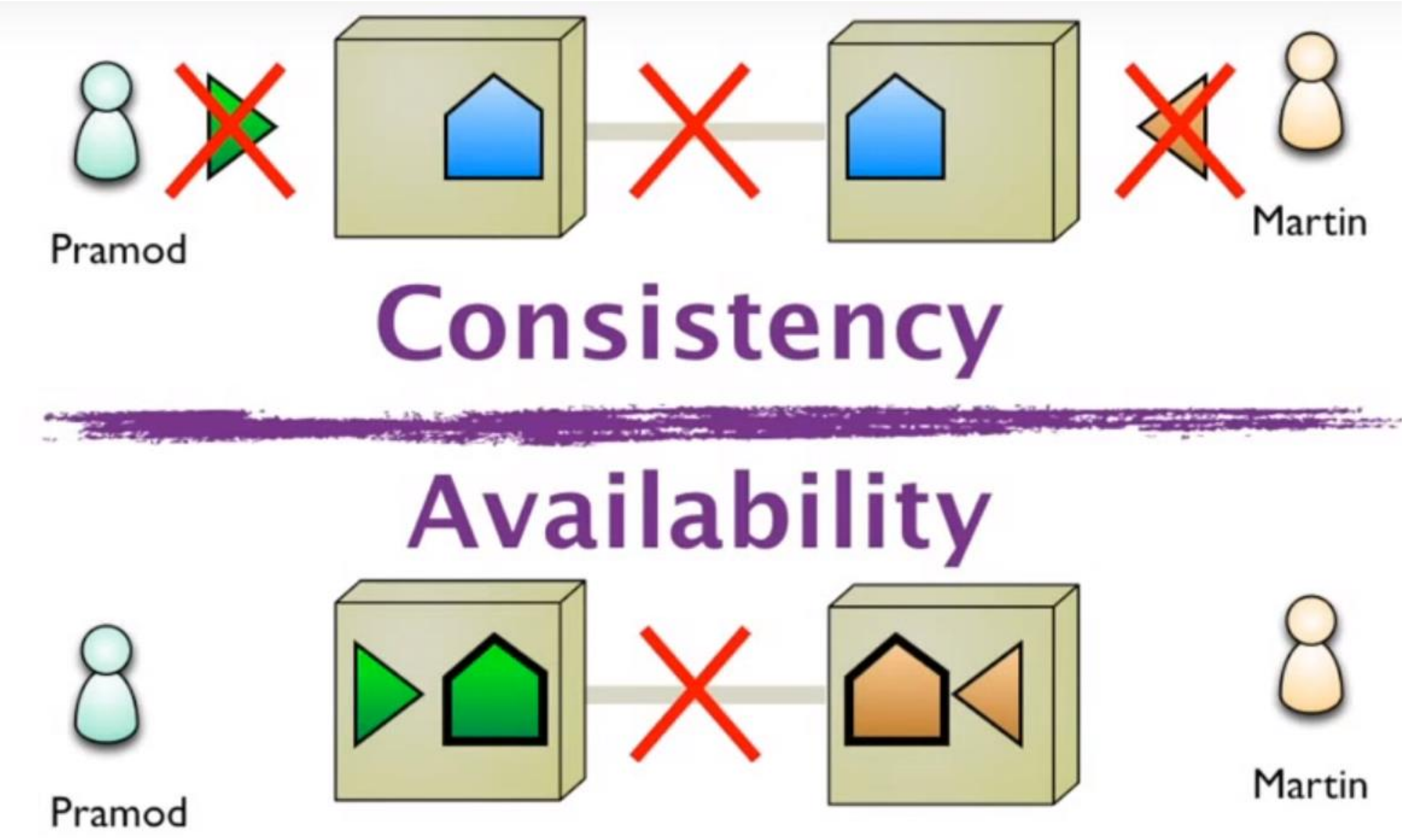

Aggregate-
Oriented
Document
Column-
family
Key-value
Graph

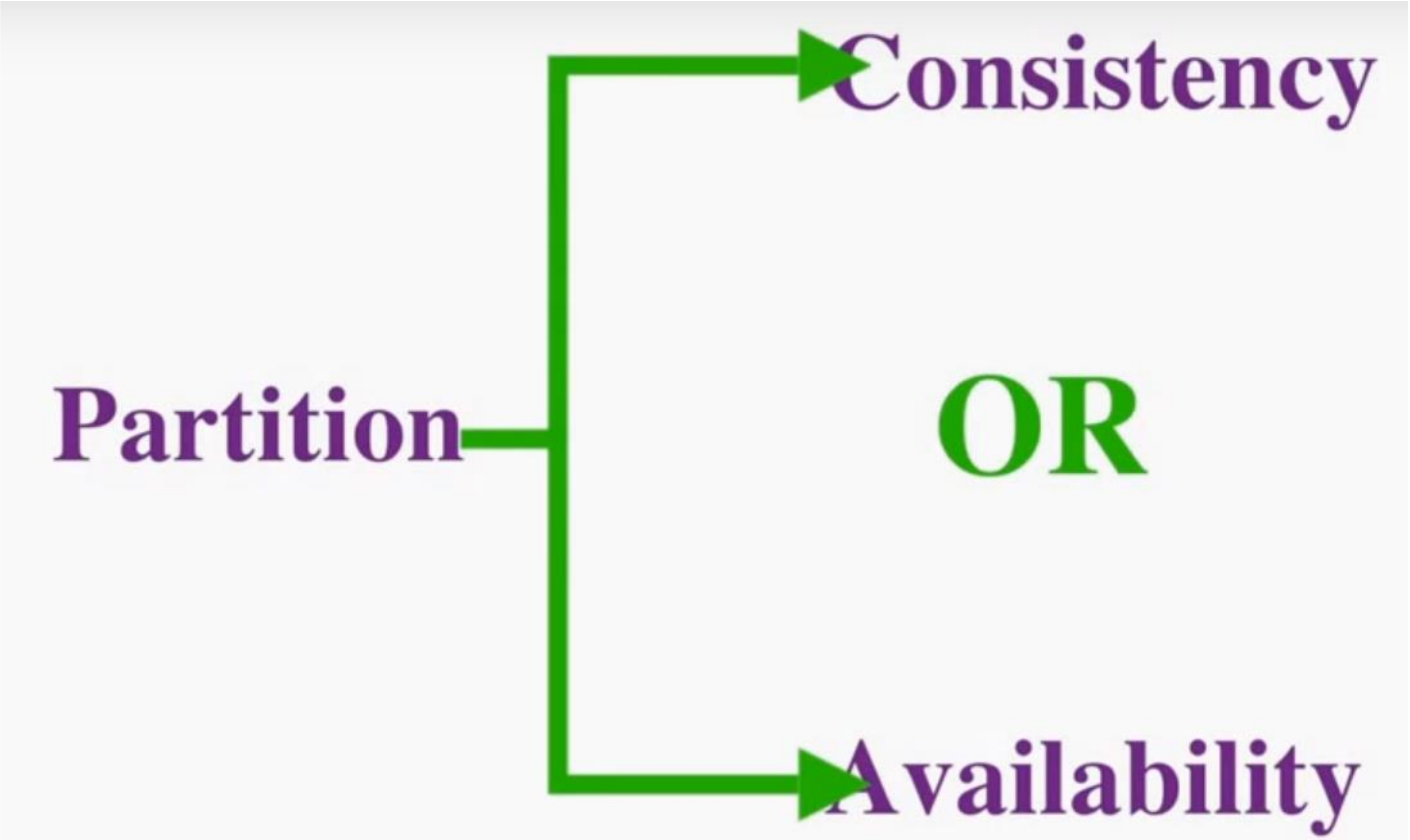


Graph
BigCo
employer
employer
employer
friend
Jill
friend
Dawn
friend
Anna
friend
Carol
Elizabeth
friend
Barbara
friend
Databases
likes
likes
likes
likes
likes
likes
Refactoring
category
NoSQL Distilled
category
author
author
author
Database Refactoring
author
martin
pramod
friend


Pramod
Martin


Pramod
Martin


Pramod
Martin
Consistency
Availability
Pramod
Martin


Consistency
Partition
OR
Availability

CAP Theorem

Consistency

Availability

PartitionTolerance

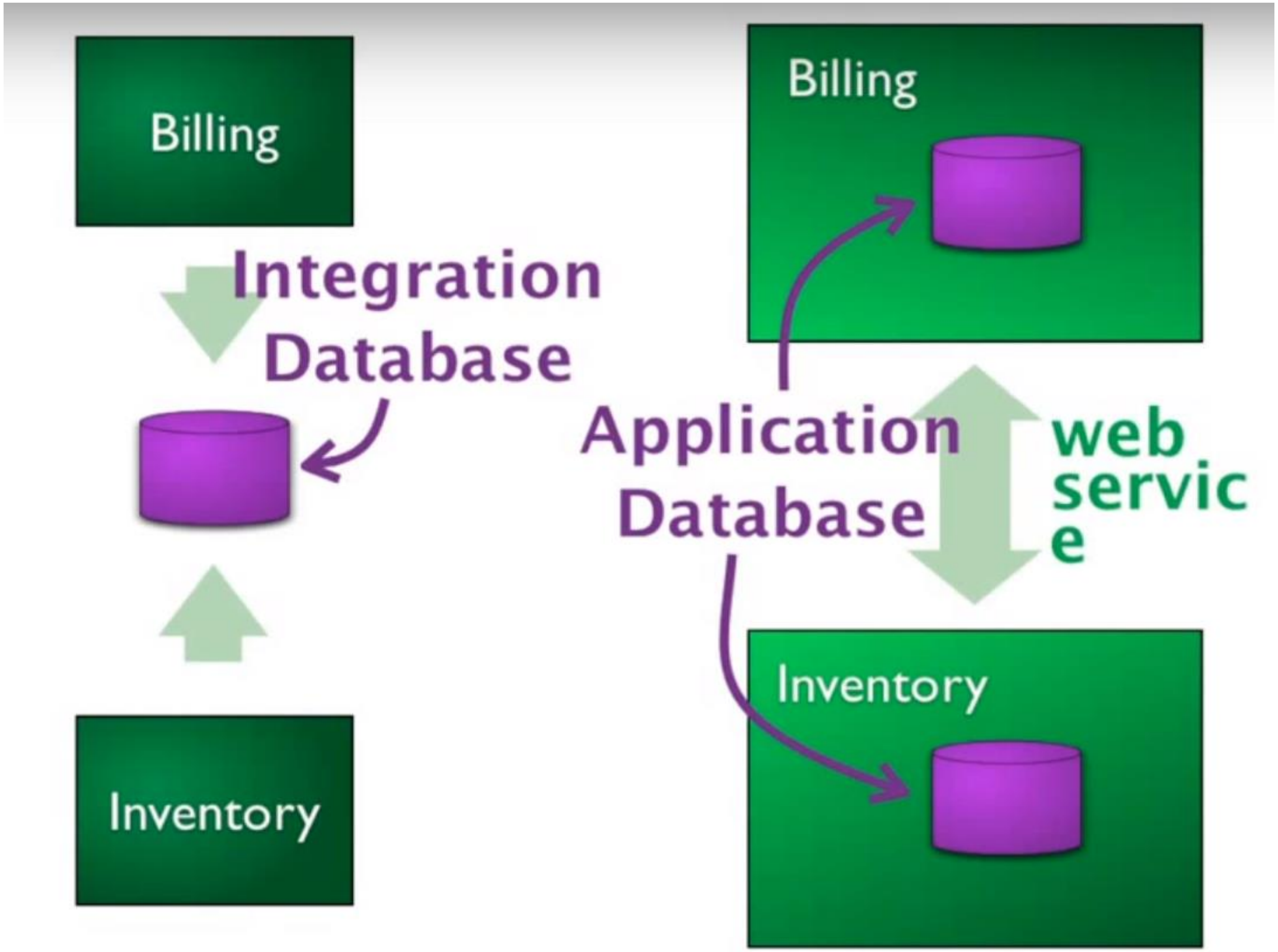
Pick any 2

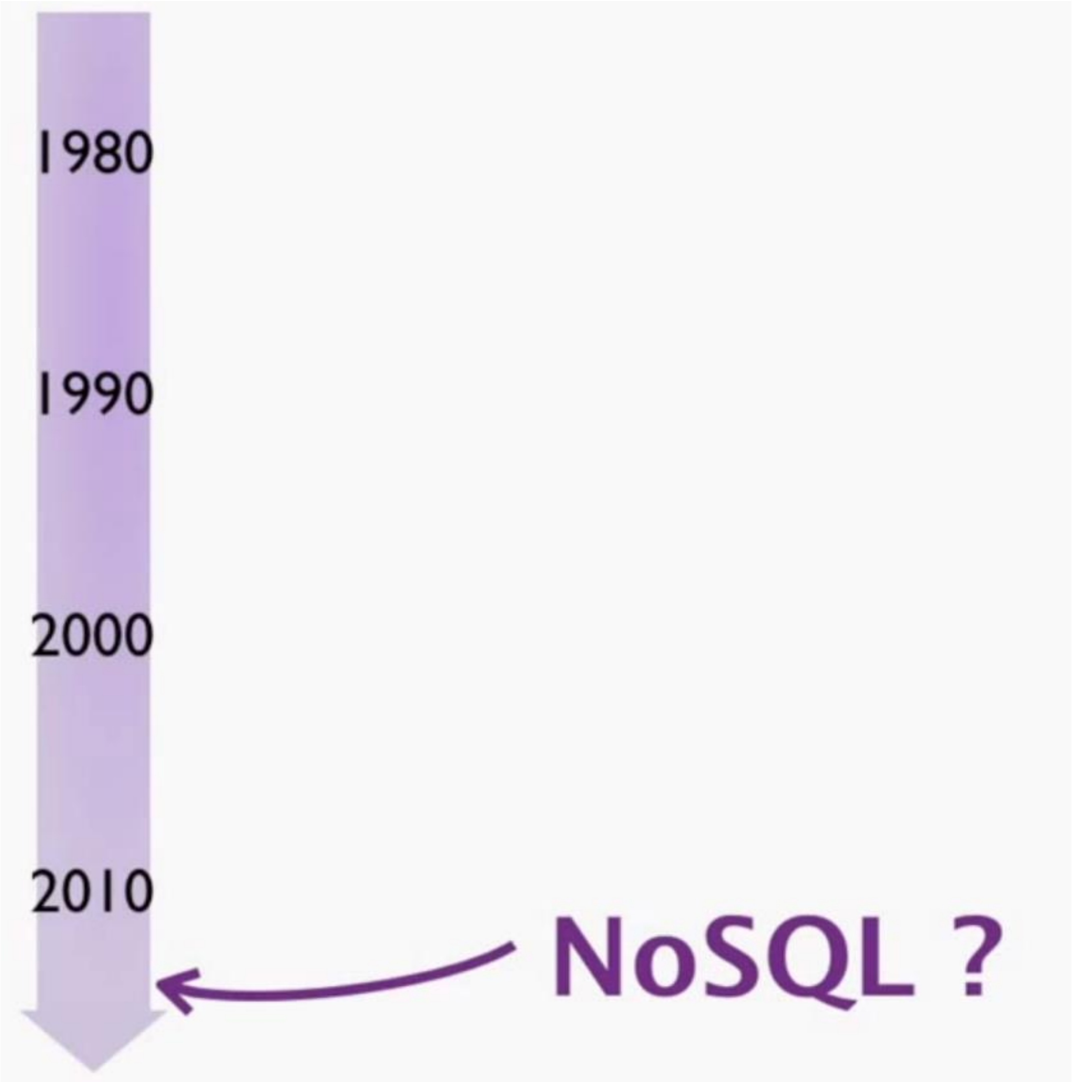
Relaxing Durability

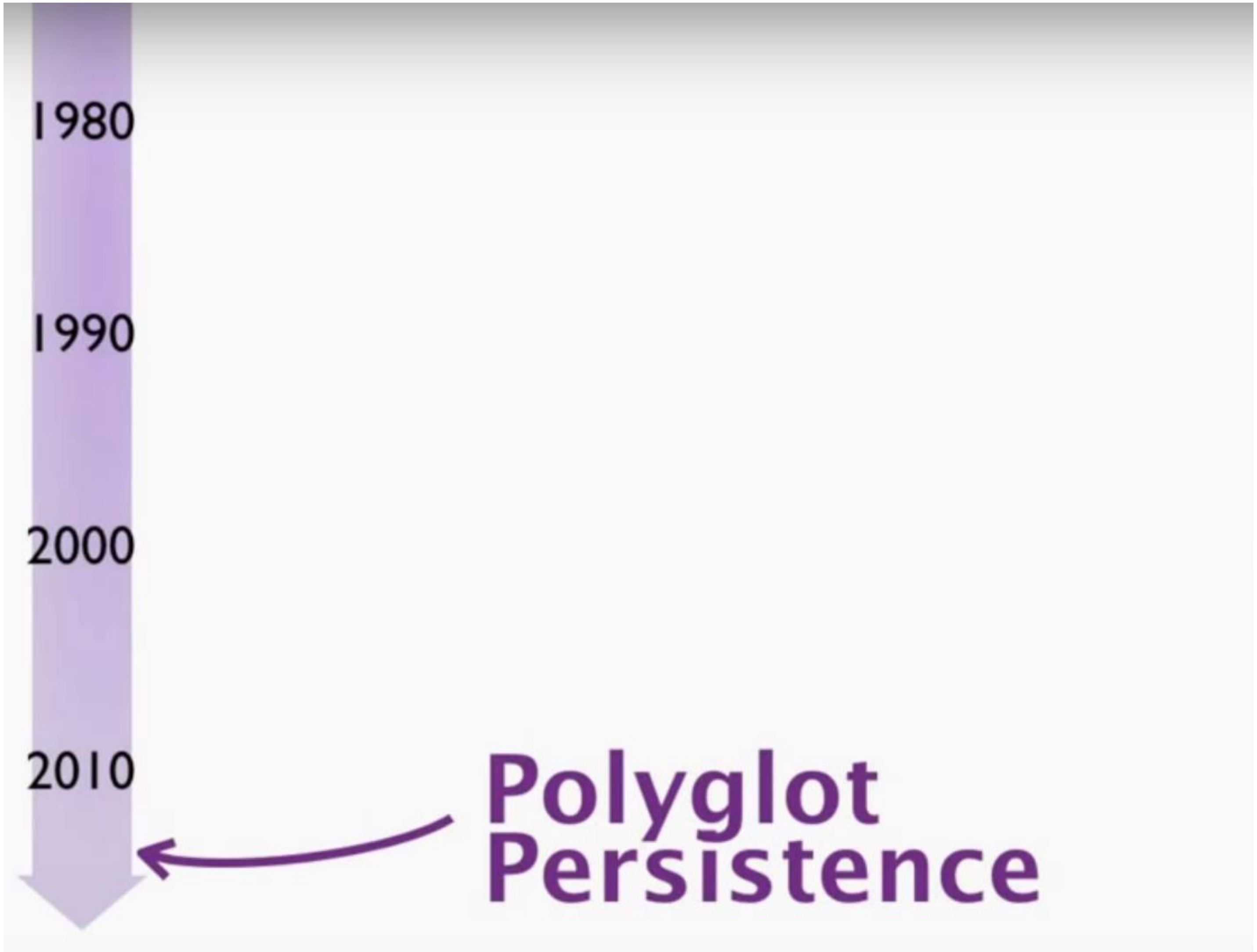
Eventual Consistency

Quorums

Read-Your-Writes Consistency


Billing
Integration
Database
Inventory
Billing
Application
Database
web
servic
e
Inventory


1980
1990
2000
2010
NoSQL ?


1980
1990
2000
2010
Polyglot
Persistence


Speculative Retailers Web Application
User sessions
Redis
Financial Data
RDBMS
Shopping Cart
Riak
Recomendations
Neo4J
Product Catalog
MongoDB
Reporting
RDBMS
Analytics
Cassandra
User activity logs
Cassandra

HBase

Apache Spark

Topics

- **What is Apache Spark?**
- **Using the Spark Shell**
- **RDDs (Resilient Distributed Datasets)**

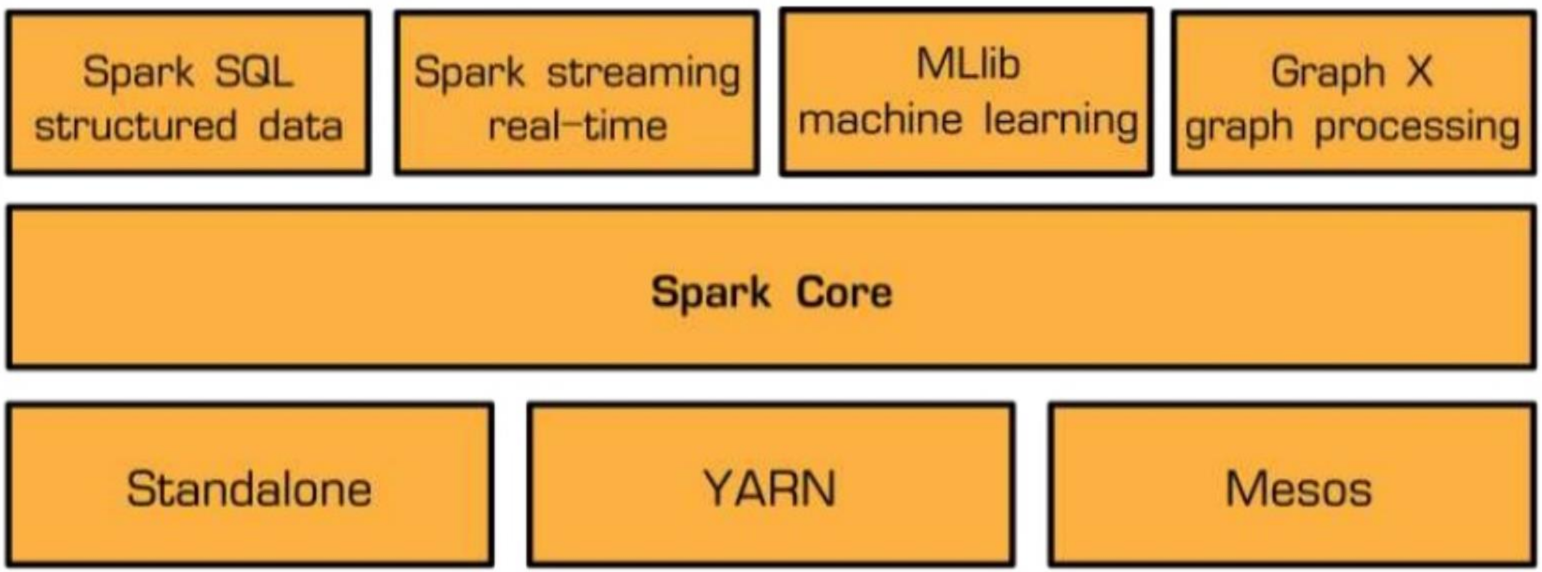
What is Apache Spark?

- **Apache Spark is a fast and general engine for large-scale data processing**
- **Written in Scala**
  - Functional programming language that runs in a JVM
- **Spark Shell**
  - Interactive – for learning or data exploration
  - Python or Scala
- **Spark Applications**
  - For large scale data processing
  - Python, Scala, or Java

Press Esc to exit full screen

Clip slide

Component Stack



Aurelius™ CORPORATE SOLUTIONS

Spark Shell

root@sandbox-hdp:~

```
[root@sandbox-hdp ~]# spark-shell
SPARK_MAJOR_VERSION is set to 2, using Spark2
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
18/03/17 17:44:08 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 404
Spark context Web UI available at http://172.17.0.2:4041
Spark context available as 'sc' (master = local[*], app id = local-1521308648434).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.2.0.2.6.3.0-235
      /_/

Using Scala version 2.11.8 (OpenJDK 64-Bit Server VM, Java 1.8.0_151)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

RDD (Resilient Distributed Dataset)

- **RDD (Resilient Distributed Dataset)**
  - Resilient – if data in memory is lost, it can be recreated
  - Distributed – processed across the cluster
  - Dataset – initial data can come from a file or be created programmatically
- **RDDs are the fundamental unit of data in Spark**
- **Most Spark programming consists of performing operations on RDDs**

Creating an RDD

- **Three ways to create an RDD**
  - From a file or set of files
  - From data in memory
  - From another RDD

Example: A File-Based RDD

Language: Scala

```
> val mydata = sc.textFile("purplecow.txt")
…
15/01/29 06:20:37 INFO storage.MemoryStore:
   Block broadcast_0 stored as values to
   memory (estimated size 151.4 KB, free 296.8
   MB)

> mydata.count()

…
15/01/29 06:27:37 INFO spark.SparkContext: Job
   finished: take at <stdin>:1, took
   0.160482078 s
4
```

File: purplecow.txt

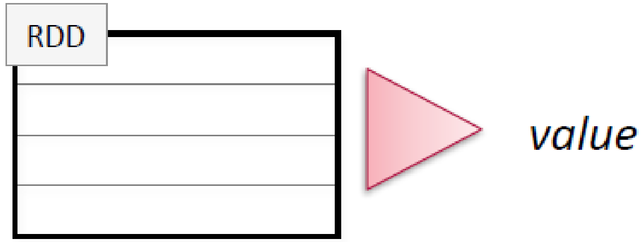
```
I've never seen a purple cow.
I never hope to see one;
But I can tell you, anyhow,
I'd rather see than be one.
```

RDD: mydata

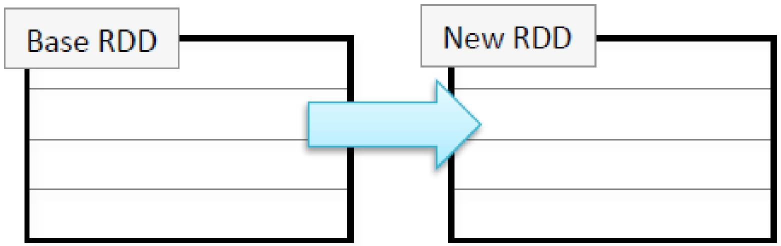
I've never seen a purple cow.
I never hope to see one;
But I can tell you, anyhow,
I'd rather see than be one.

RDD Operations

- **Two types of RDD operations**
  - Actions – return values



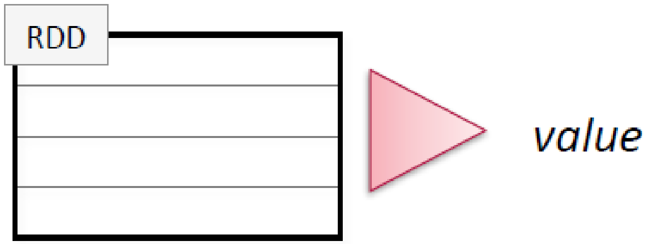
  - Transformations – define a new RDD based on the current one(s)



- **Pop quiz:**
  - Which type of operation is `count()`?

RDD Operations: Actions

- **Some common actions**
  - `count()` – return the number of elements
  - `take(n)` – return an array of the first *n* elements
  - `collect()` – return an array of all elements
  - `saveAsTextFile(file)` – save to text file(s)



Language: Python

```
> mydata =
  sc.textFile("purplecow.txt")

> mydata.count()
4

> for line in mydata.take(2):
    print line
I've never seen a purple cow.
I never hope to see one;
```

Language: Scala

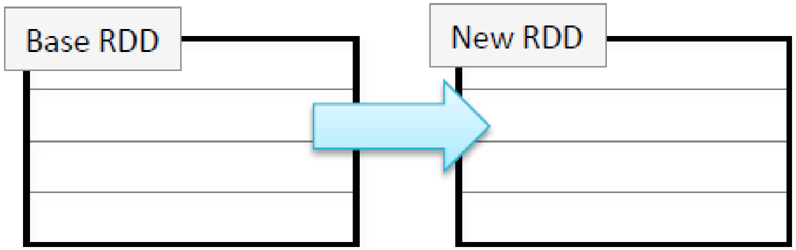
```
> val mydata =
  sc.textFile("purplecow.txt")

> mydata.count()
4

> for (line <- mydata.take(2))
   println(line)
I've never seen a purple cow.
I never hope to see one;
```

RDD Operations: Transformations

- **Transformations create a new RDD from an existing one**



- **RDDs are immutable**
  - Data in an RDD is never changed
  - Transform in sequence to modify the data as needed
- **Some common transformations**
  - `map(`*`function`*`)` – creates a new RDD by performing a function on each record in the base RDD
  - `filter(`*`function`*`)` – creates a new RDD by including or excluding each record in the base RDD according to a boolean function

Example: `map` and `filter` Transformations

Language: Python

Language: Scala

```
I've never seen a purple cow.
I never hope to see one;
But I can tell you, anyhow,
I'd rather see than be one.
```

```
map(lambda line: line.upper())
```

```
map(line => line.toUpperCase)
```

```
I'VE NEVER SEEN A PURPLE COW.
I NEVER HOPE TO SEE ONE;
BUT I CAN TELL YOU, ANYHOW,
I'D RATHER SEE THAN BE ONE.
```

```
filter(lambda line: line.startswith('I'))
```

```
filter(line => line.startsWith('I'))
```

```
I'VE NEVER SEEN A PURPLE COW.
I NEVER HOPE TO SEE ONE;
I'D RATHER SEE THAN BE ONE.
```

Lazy Execution

- **Data in RDDs is not processed until an *action* is performed**

Language: Scala

```
> val mydata = sc.textFile("purplecow.txt")
> val mydata_uc = mydata.map(line =>
  line.toUpperCase())
> val mydata_filt = mydata_uc.filter(line
  => line.startsWith("I"))
> mydata_filt.count()
3
```

File: purplecow.txt

```
I've never seen a purple cow.
I never hope to see one;
But I can tell you, anyhow,
I'd rather see than be one.
```

RDD: mydata

```
I've never seen a purple cow.
I never hope to see one;
But I can tell you, anyhow,
I'd rather see than be one.
```

RDD: mydata_uc

```
I'VE NEVER SEEN A PURPLE COW.
I NEVER HOPE TO SEE ONE;
BUT I CAN TELL YOU, ANYHOW,
I'D RATHER SEE THAN BE ONE.
```

RDD: mydata_filt

```
I'VE NEVER SEEN A PURPLE COW.
I NEVER HOPE TO SEE ONE;
I'D RATHER SEE THAN BE ONE.
```

Chaining Transformations (Python)

- **Same example in Python**

```
> mydata = sc.textFile("purplecow.txt")
> mydata_uc = mydata.map(lambda s: s.upper())
> mydata_filt = mydata_uc.filter(lambda s: s.startswith('I'))
> mydata_filt.count()
3
```

is exactly equivalent to

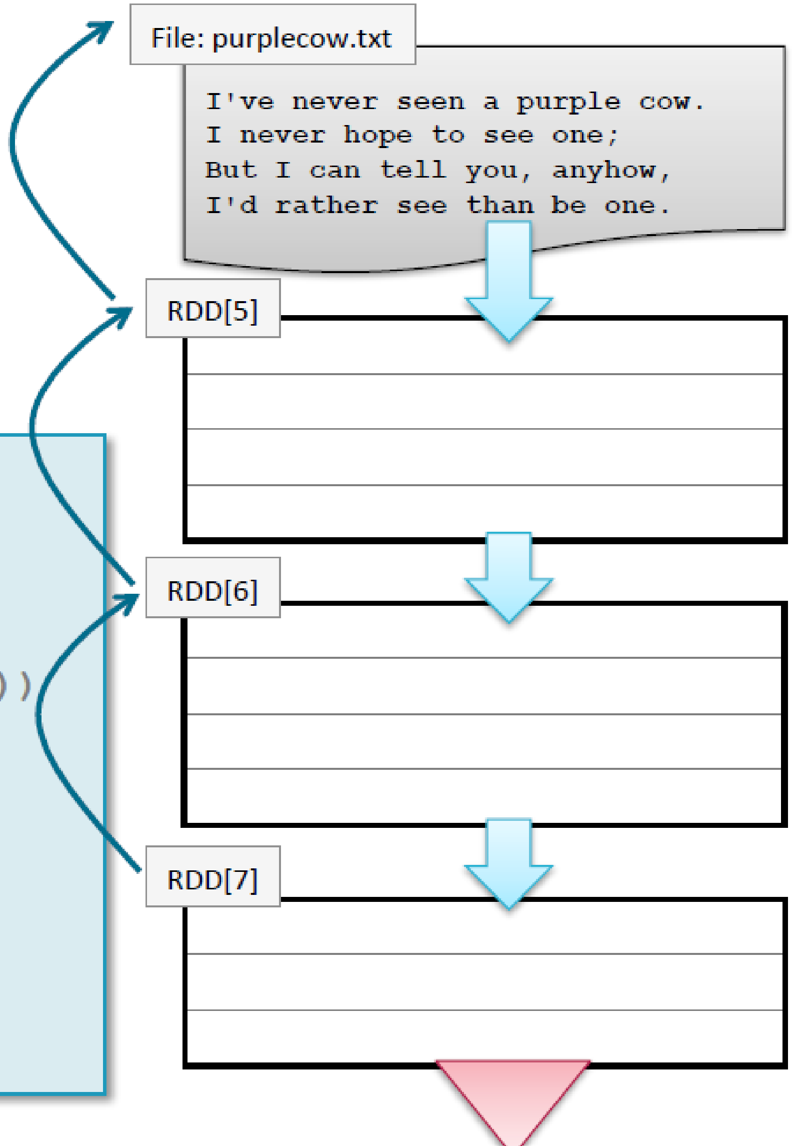
```
> sc.textFile("purplecow.txt").map(lambda line: line.upper()) \
   .filter(lambda line: line.startswith('I')).count()
3
```

RDD Lineage and `toDebugString` (Scala)

- **Spark maintains each RDD's *lineage* – the previous RDDs on which it depends**
- **Use `toDebugString` to view the lineage of an RDD**

```
> val mydata_filt =
    sc.textFile("purplecow.txt").
    map(line => line.toUpperCase()).
    filter(line => line.startsWith("I"))
> mydata_filt.toDebugString

(2) FilteredRDD[7] at filter …
 |  MappedRDD[6] at map …
 |  purplecow.txt MappedRDD[5] …
 |  purplecow.txt HadoopRDD[4] …
```



Example: **`flatMap`** and **`distinct`**

```
> sc.textFile(file) \
  .flatMap(lambda line: line.split()) \
  .distinct()
```

Language: Python

```
> sc.textFile(file).
   flatMap(line => line.split(' ')).
   distinct()
```

Language: Scala

I've never seen a purple cow.
I never hope to see one;
But I can tell you, anyhow,
I'd rather see than be one.

→

I've
never
seen
a
purple
cow
I
never
hope
to

→

I've
never
seen
a
purple
cow
I
hope
to
…

```
[root@sandbox-hdp ~]# date
Sat Mar 17 18:06:14 UTC 2018
[root@sandbox-hdp ~]# pyspark
SPARK_MAJOR_VERSION is set to 2, using Spark2
Python 2.6.6 (r266:84292, Aug 18 2016, 15:13:37)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-17)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
18/03/17 18:06:22 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.
/usr/hdp/current/spark2-client/python/pyspark/context.py:205: UserWarning: Support for Python 2.6 is deprecated as of Spark 2.0.0
  warnings.warn("Support for Python 2.6 is deprecated as of Spark 2.0.0")
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.2.0.2.6.3.0-235
      /_/

Using Python version 2.6.6 (r266:84292, Aug 18 2016 15:13:37)
SparkSession available as 'spark'.
>>>
>>> counts = sc.textFile("/user/root/input.txt").flatMap(lambda line: line.split()).map(lambda word: (word,1)).reduceByKey(lambda v1,v2
: v1+v2)
>>>
>>> counts.take(5)
[(u'world', 1), (u'hello', 2), (u'again', 1)]
>>>
>>> quit()
[root@sandbox-hdp ~]# hdfs dfs -cat /user/root/input.txt
hello world
hello again

[root@sandbox-hdp ~]#
```

DataFrames and SparkSQL

In this chapter you will learn

- **What Spark SQL is**
- **What features the DataFrame API provides**
- **How to create a SQLContext**
- **How to load existing data into a DataFrame**
- **How to query data in a DataFrame**

What is Spark SQL?

- **What is Spark SQL?**
  - Spark module for structured data processing
  - Replaces Shark (a prior Spark module, now deprecated)
  - Built on top of core Spark
- **What does Spark SQL provide?**
  - The DataFrame API – a library for working with data as tables
    - Defines DataFrames containing Rows and Columns
    - DataFrames are the focus of this chapter!
  - Catalyst Optimizer – an extensible optimization framework
  - A SQL Engine and command line interface

SQL Context

- **The main Spark SQL entry point is a SQL Context object**
  - Requires a SparkContext
  - The SQL Context in Spark SQL is similar to Spark Context in core Spark
- **There are two implementations**
  - **`SQLContext`**
    - basic implementation
  - **`HiveContext`**
    - Reads and writes Hive/HCatalog tables directly
    - Supports full HiveQL language
    - Requires the Spark application be linked with Hive libraries
    - Recommended starting with Spark 1.5

Creating a SQL Context

- `SQLContext` is created based on the `SparkContext`

Language: Python

```
from pyspark.sql import SQLContext
sqlCtx = SQLContext(sc)
```

Language: Scala

```
import org.apache.spark.sql.SQLContext
val sqlCtx = new SQLContext(sc)
import sqlCtx._
```

DataFrames

- **DataFrames are the main abstraction in Spark SQL**
  - Analogous to RDDs in core Spark
  - A distributed collection of data organized into named columns
  - Built on a base RDD containing **Row** objects

- **DataFrames can be created**
  - From an existing structured data source (Parquet file, JSON file, etc.)
  - From an existing RDD
  - By performing an operation or query on another DataFrame
  - By programmatically defining a schema

Example: Creating a DataFrame from a JSON File

Language: Python

```python
from pyspark.sql import SQLContext
sqlCtx = SQLContext(sc)
peopleDF = sqlCtx.jsonFile("people.json")
```

Language: Scala

```scala
val sqlCtx = new SQLContext(sc)
import sqlCtx._
val peopleDF = sqlCtx.jsonFile("people.json")
```

File: people.json

```
{"name":"Alice", "pcode":"94304"}
{"name":"Brayden", "age":30, "pcode":"94304"}
{"name":"Carla", "age":19, "pcode":"10036"}
{"name":"Diana", "age":46}
{"name":"Étienne", "pcode":"94104"}
```

age	name	pcode
null	Alice	94304
30	Brayden	94304
19	Carla	10036
46	Diana	null
null	Étienne	94104

Creating a DataFrame from a Data Source

- **Methods on the SQLContext object**
- **Convenience functions**
  - `jsonFile(filename)`
  - `parquetFile(filename)`
- **Generic base function: `load`**
  - `load(filename,source)` – load **`filename`** of type **`source`** (default Parquet)
  - `load(source,options...)` – load from a source of type **`source`** using options
  - Convenience functions are implemented by calling **`load`**
    - `jsonFile("people.json") = load("people.json", "json")`

Generic Load Function Example: JDBC

- **Example: Loading from a MySQL database**

```
val accountsDF = sqlCtx.load("jdbc",
   Map("url"-> "jdbc:mysql://dbhost/dbname?user=…&password=…",
       "dbtable" -> "accounts"))
```

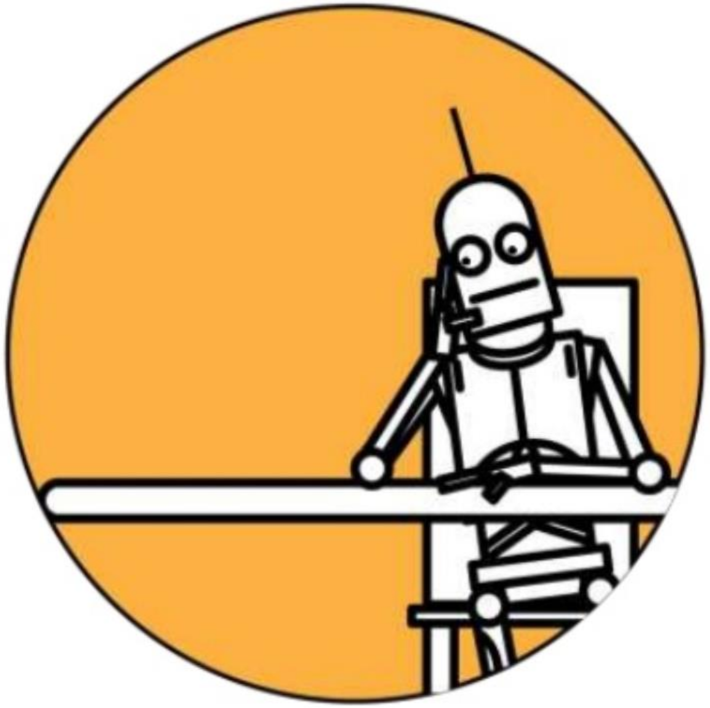
```
accountsDF = sqlCtx.load(source="jdbc", \
  url="jdbc:mysql://dbhost/dbname?user=…&password=…", \
  dbtable="accounts”)
```

Warning: Avoid direct access to databases in production environments, which may overload the DB or be interpreted as service attacks

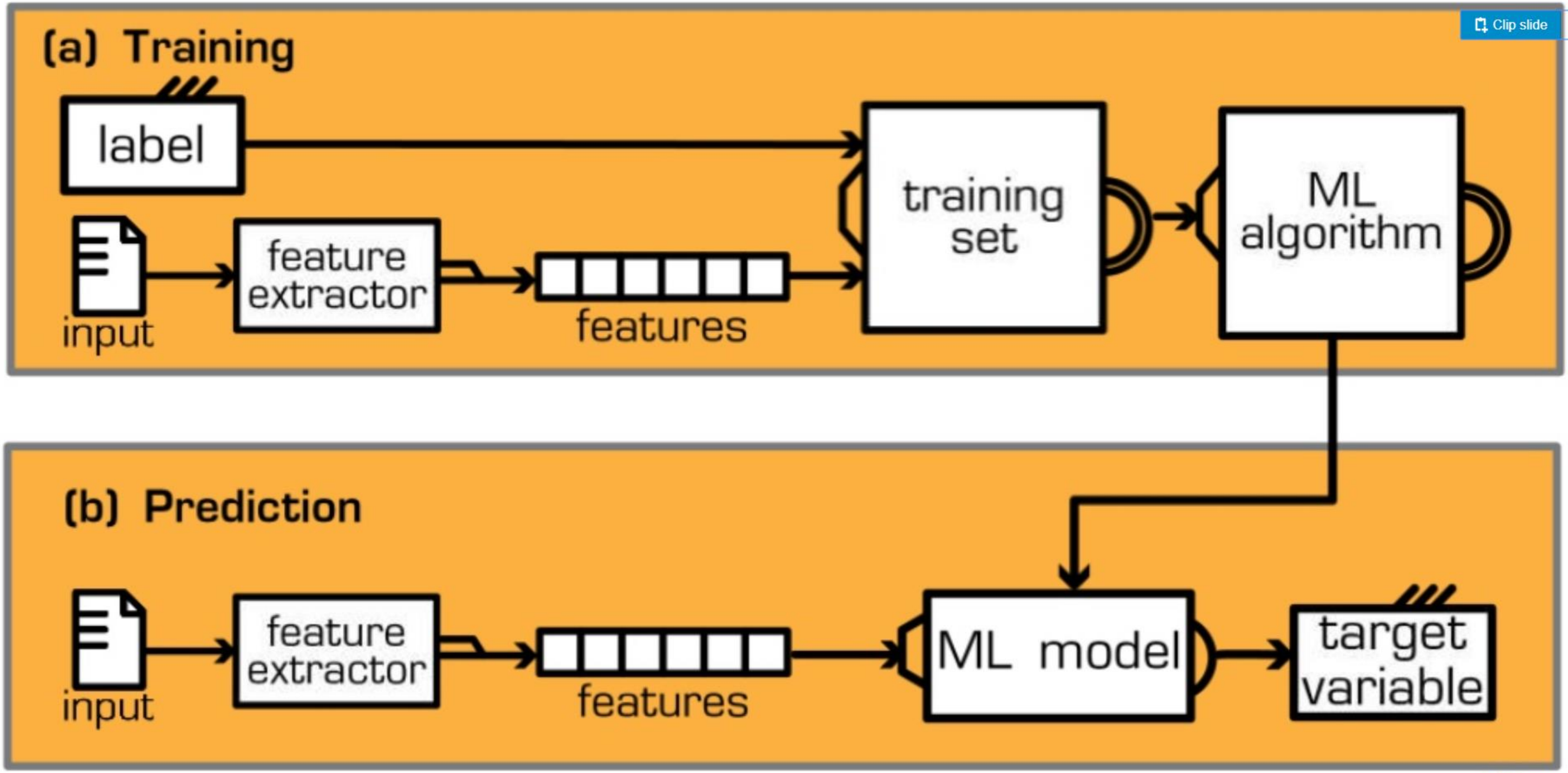
- Use Sqoop to import instead

Machine Learning

is the study of computer algorithms that improve automatically through experience



Clip slide



http://www.slideshare.net/liweiyang5/spark-mllib-training-material

javascript:void(0)

Clip slide

Spark MLlib

Is a library of ML algorithms and utilities designed to run in parallel on Spark cluster

Aurelius™ CORPORATE SOLUTIONS

Clip slide

spark.mllib Features

- Utilities: linear algebra, statistics, etc.
- Features extraction, features transforming, etc.
- Regression
- Classification
- Clustering
- Collaborative filtering, e.g. alternating least squares
- Dimensionality reduction
- And many more

http://spark.apache.org/docs/latest/mllib-guide.html


tusind tak
ngiyabonga
dziękuję
谢谢
dakujem vám
ありがとう
merci
suksema
danke
thank
baie dankie
धन्यवाद
molte grazie
gracias
takk
you
dank u
teşekkür edirem
mahalo
obrigada
obrigado
gràcies
teşekkür ederim
شكرا
tänan
tack så mycket