Exploring Wikipedia with Spark

databrickstraining

Today's Objectives

- Learn just enough to build simple prototypes/POCs with Spark
- Fast paced, high level overview of all major Spark components
- Hands on with Spark's programming APIs *(DataFrame/SQL, RDD, Datasets)*
- Overview of Spark architecture: Core, Streaming, Standalone Mode, DAG
- Mix of beginner + advanced topics
- Not all slides/labs covered *(reference & homework material)*
- Lots of ideas, code & datasets to play around with after class

Data

Spark Analytics

Pageviews (March 2015) - 255 MB	→	Dataframes
Clickstream (Feb 2015) - 1.2 GB	→	Dataframes, SQL, GraphX
Pagecounts (last hour) - ~500 MB	→	RDD, Datasets, Dataframes
English Wikipedia (Mar 5, 2016) - 54 GB	→	Dataframes, SQL, ML
6 Lang Wikipedias (live streams)	→	Streaming, RDD, Dataframes, SQL

Large Scale Usage:

Largest cluster: 8000 nodes 

Largest single job: 1 petabyte 

Top streaming intake: 1 TB/hour 

2014 on-disk 100 TB sort record: 23 mins / 207 EC2 nodes 

Machine Learning

w/ Joseph Bradley @ Facebook

Machine Learning for Humans

- Make practical ML scalable and easy
- Inspired by scikit-learn

Algorithm coverage

Classification

- Logistic regression w/ elastic net
- Naive Bayes
- Streaming logistic regression
- Linear SVMs
- Decision trees
- Random forests
- Gradient-boosted trees
- Multilayer perceptron
- One-vs-rest

Regression

- Least squares w/ elastic net
- Isotonic regression
- Decision trees
- Random forests
- Gradient-boosted trees
- Streaming linear methods

Recommendation

- Alternating Least Squares

Frequent itemsets

- FP-growth
- Prefix span

Feature extraction & selection

- Binarizer
- Bucketizer
- Chi-Squared selection
- CountVectorizer
- Discrete cosine transform
- ElementwiseProduct
- Hashing term frequency
- Inverse document frequency
- MinMaxScaler
- Ngram
- Normalizer
- One-Hot Encoder
- PCA
- PolynomialExpansion
- RFormula
- SQLTransformer
- Standard scaler
- StopWordsRemover
- StringIndexer
- Tokenizer
- StringIndexer
- VectorAssembler
- VectorIndexer
- VectorSlicer
- Word2Vec

Clustering

- Gaussian mixture models
- K-Means
- Streaming K-Means
- Latent Dirichlet Allocation
- Power Iteration Clustering

Statistics

- Pearson correlation
- Spearman correlation
- Online summarization
- Chi-squared test
- Kernel density estimation

Linear algebra

- Local dense & sparse vectors & matrices
- Distributed matrices
  - Block-partitioned matrix
  - Row matrix
  - Indexed row matrix
  - Coordinate matrix
- Matrix decompositions

Model import/export

Pipelines

List based on Spark 1.5

High-level functionality

Learning tasks

- Classification
- Regression
- Recommendation
- Clustering
- Frequent itemsets

Data utilities

- Feature extraction & selection
- Statistics
- Linear algebra

Workflow utilities

- Model import/export
- Pipelines
- DataFrames
- Cross validation

Spark MLlib Components

spark.mllib
(package)

- Original “lower-level” API
- Built on top of RDDs
- Will go to maintenance mode starting Spark 2.0

vs.

spark.ml
(package)

- Newer “higher-level” API for constructing workflows
- Built on top of DataFrames

Term Frequency – Inverse Document Frequency

An information retrieval algorithm used to rank how important a word is to a collection of documents

---

TF: If a word appears frequently in a doc, it's important

IDF: But if a word appears in many docs (the, and, of), the word is not meaningful, so lower its score

TF-IDF

$$\text{TF}: \quad \frac{\text{\# of times word X appears in a document}}{\text{Total \# of words in the document}}$$

TF-IDF

the: 2
and: 1
politics: 1
government: 1
of: 1

of: 2
congress: 2
and: 1
the: 1
politics: 1

sports: 2
the: 1
and: 1
games: 1
olympics: 1

TF-IDF

the: 2 / 6 = 0.3
and: 1 / 6 = 0.16
politics: 1 / 6 = 0.16
government: 1 / 6 = 0.16
of: 1 / 6 = 0.16

of: 2 / 7 = 0.28
congress: 2 / 7 = 0.28
and: 1 / 7 = 0.14
the: 1 / 7 = 0.14
politics: 1 / 7 = 0.14

sports: 2 / 6 = 0.3
the: 1 / 6 = 0.16
and: 1 / 6 = 0.16
games: 1 / 6 = 0.16
olympics: 1 / 6 = 0.16

TF-IDF

the: 2 / 6 = 0.3
and: 1 / 6 = 0.16
politics: 1 / 6 = 0.16
government: 1 / 6 = 0.16
of: 1 / 6 = 0.16

of: 2 / 7 = 0.28
congress: 2 / 7 = 0.28
and: 1 / 7 = 0.14
the: 1 / 7 = 0.14
politics: 1 / 7 = 0.14

sports: 2 / 6 = 0.3
the: 1 / 6 = 0.16
and: 1 / 6 = 0.16
games: 1 / 6 = 0.16
olympics: 1 / 6 = 0.16

TF-IDF

IDF: $\log \left[\frac{\text{Total \# of documents}}{\text{\# of documents with word X in it}} \right]$

TF-IDF

$$\text{IDF}: \quad \log\left[\frac{3}{3}\right] = 0$$

TF-IDF

$$\text{IDF}: \quad \log\left[\frac{3}{2}\right] = 0.18$$

TF-IDF

$$\text{IDF}: \quad \log\left[\frac{3}{1}\right] = 0.48$$

TF-IDF

government: 0.16 * 0.48 = 0.08
politics: 0.16 * 0.18 = 0.03
of: 0.16 * 0.18 = 0.03
the: 0.3 * 0 = 0
and: 0.16 * 0 = 0

congress: 0.28 * 0.48 = 0.08
of: 0.28 * 0.18 = 0.05
politics: 0.14 * 0.18 = 0.03
the: 0.14 * 0 = 0
and: 0.14 * 0 = 0

sports: 0.3 * 0.48 = 0.144
olympics: 0.16 * 0.48 = 0.08
games: 0.16 * 0.48 = 0.08
the: 0.16 * 0 = 0
and: 0.16 * 0 = 0

TF-IDF

sports Search

government: 0.16 * 0.48 = 0.08
politics: 0.16 * 0.18 = 0.03
of: 0.16 * 0.18 = 0.03

congress: 0.28 * 0.48 = 0.08
of: 0.28 * 0.18 = 0.05
politics: 0.14 * 0.18 = 0.03

sports: 0.3 * 0.48 = 0.144
olympics: 0.16 * 0.48 = 0.08
games: 0.16 * 0.48 = 0.08

TF-IDF

politics **Search**

government: 0.16 * 0.48 = 0.08
politics: 0.16 * 0.18 = 0.03
of: 0.16 * 0.18 = 0.03

congress: 0.28 * 0.48 = 0.08
of: 0.28 * 0.18 = 0.05
politics: 0.14 * 0.18 = 0.03

sports: 0.3 * 0.48 = 0.144
olympics: 0.16 * 0.48 = 0.08
games: 0.16 * 0.48 = 0.08

K-means clustering

An unsupervised ML algorithm for classifying items into k groups.

---

k: The # of pre-chosen groups

K–means clustering







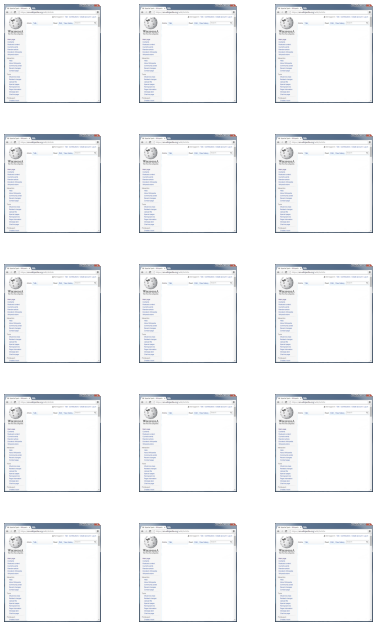


politics

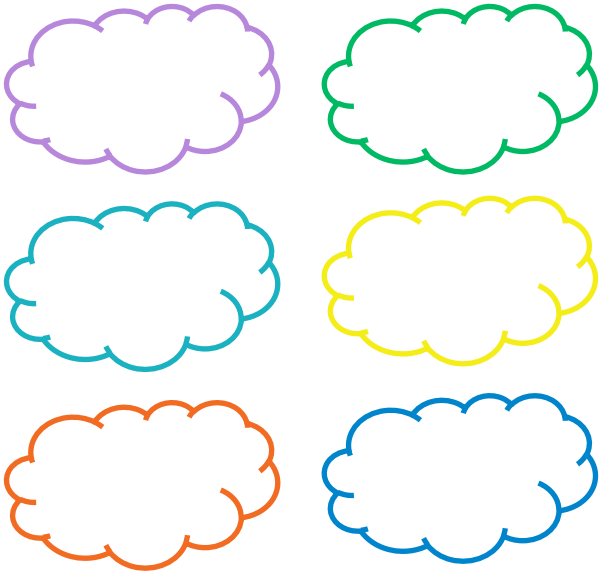

sports

K-means clustering

K-means clustering

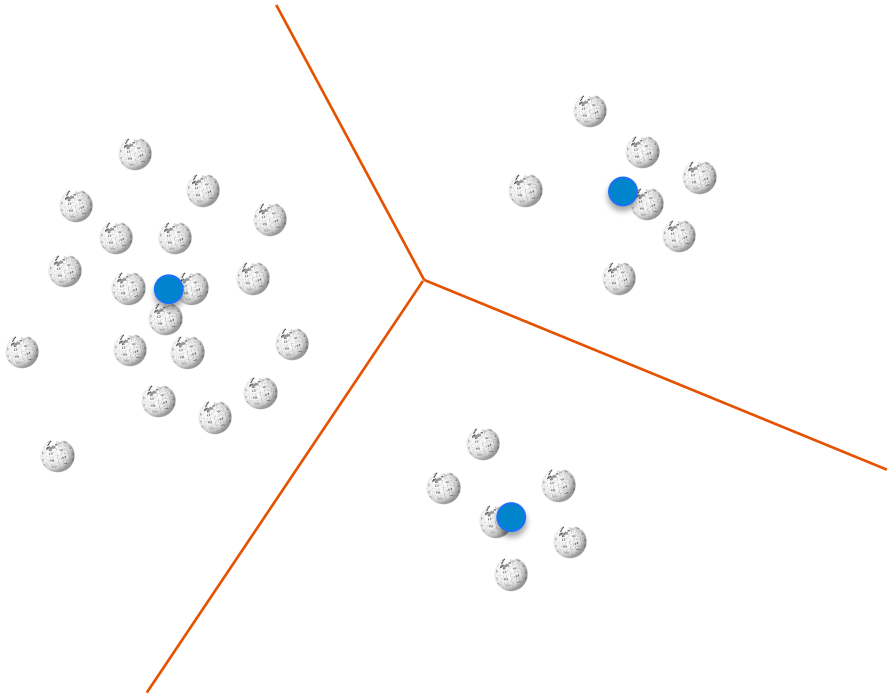


5 million articles



100 clusters

K-means clustering

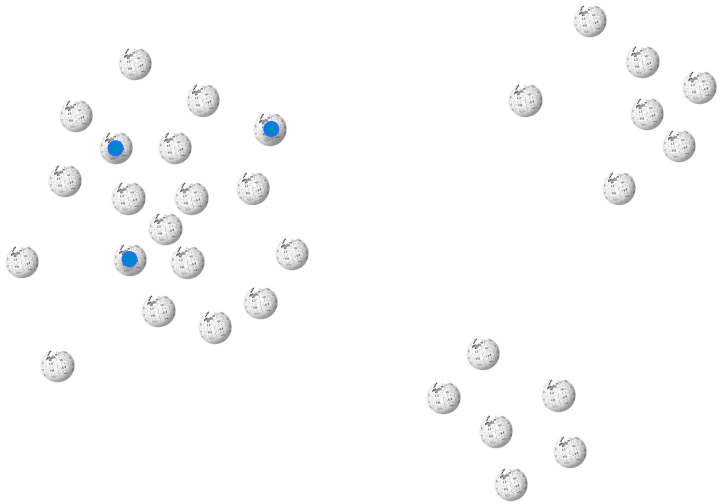


black dots: Wikipedia Articles (items)

red lines: Cluster Partitions

blue dots: Cluster Centroids

K-means clustering: Initialization



Approach #1:

Randomly select centroids to start

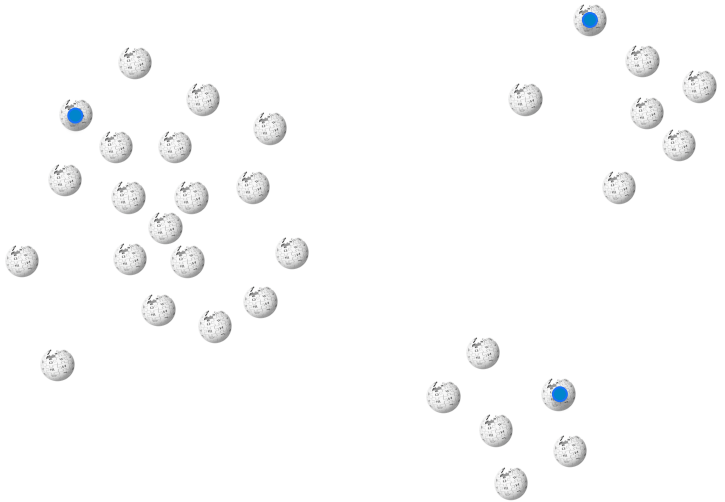
Pro:

Does not require full iteration

Con:

If a cluster is large compared to neighbors, initial centroids will likely end up in large cluster.

K-means clustering: Initialization



Approach #2:

k-means++

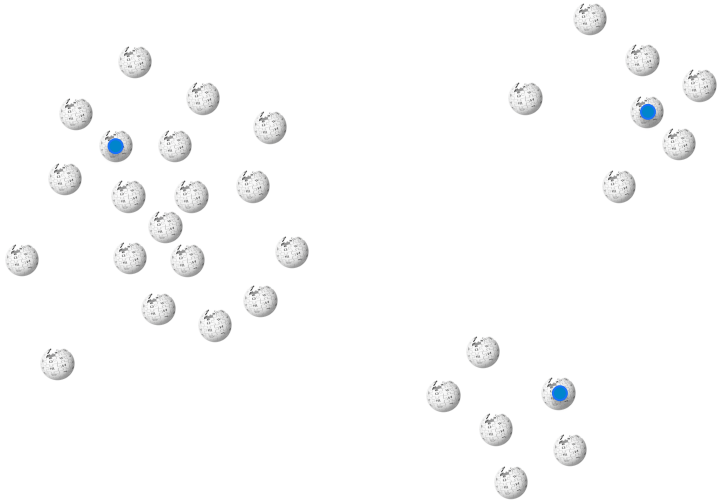
Pro:

Avoids the poor clusterings found by standard algorithm

Con:

Needs k passes over the data, so does not scale well to large data sets

K-means clustering: Initialization



Approach #3:

k-means||

Pro:

Also avoids the poor clusterings found by standard algorithm

Can select multiple candidate centroids in first iteration

K-means clustering: Initialization

(expert-only) Parameters

A list of advanced, expert-only (hyper-)parameter keys this algorithm can take. Users can set and get the parameter values through setters and getters, respectively.

final val **initMode**: Param[String]

Param for the initialization algorithm. This can be either "random" to choose random points as initial cluster centers, or "k-means||" to use a parallel variant of k-means++ (Bahmani et al., Scalable K-Means++, VLDB 2012). Default: k-means||.

Definition Classes	KMeansParams
Annotations	@Since("1.5.0")

final val **initSteps**: IntParam

Param for the number of steps for the k-means|| initialization mode.

Source: https://spark.apache.org/docs/1.6.0/api/scala/index.html#org.apache.spark.ml.clustering.KMeans

K-means clustering

k: 2

Article	Word X	Word Y
A	1.0	1.0
B	1.5	2.0
C	3.0	4.0
D	5.0	7.0
E	3.5	5.0
F	4.5	5.0
G	3.5	4.5

Source: http://mnemstudio.org

K-means clustering

k: 2

	Article	Mean Vector *(centroid)*
Group 1	A	(1.0, 1.0)
Group 2	D	(5.0, 7.0)

K-means clustering

k: 2

	Article	Mean Vector *(centroid)*
Cluster 1	A, B, C	(1.8, 2.3)
Cluster 2	D, E, F, G	(4.1, 5.4)

K-means clustering

k: 2

	Article	Distance to mean *(centroid)* of Cluster 1	Distance to mean *(centroid)* of Cluster 2
Cluster 1	A	1.5	5.4
	B	0.4	4.3
	C	2.1	1.8
Cluster 2	D	5.7	1.8
	E	3.2	0.7
	F	3.8	0.6
	G	2.8	1.1

K-means clustering

k: 2

	Article	Mean Vector *(centroid)*
Cluster 1	A, B	(1.3, 1.5)
Cluster 2	C, D, E, F, G	(3.9, 5.1)

K-means clustering

```scala
/**
 * :: Experimental ::
 * K-means clustering with support for k-means|| initialization proposed by Bahmani et al.
 *
 * @see [[http://dx.doi.org/10.14778/2180912.2180915 Bahmani et al., Scalable k-means++.]]
 */
@Since("1.5.0")
@Experimental
class KMeans @Since("1.5.0") (
    @Since("1.5.0") override val uid: String)
  extends Estimator[KMeansModel] with KMeansParams with DefaultParamsWritable {

  setDefault(
    k -> 2,
    maxIter -> 20,
    initMode -> MLlibKMeans.K_MEANS_PARALLEL,
    initSteps -> 5,
    tol -> 1e-4)

  @Since("1.5.0")
  override def copy(extra: ParamMap): KMeans = defaultCopy(extra)

  @Since("1.5.0")
  def this() = this(Identifiable.randomUID("kmeans"))

  /** @group setParam */
  @Since("1.5.0")
  def setFeaturesCol(value: String): this.type = set(featuresCol, value)
```

Source: https://github.com/apache/spark/blob/master/mllib/src/main/scala/org/apache/spark/ml/clustering/KMeans.scala#L216

Spark ML Pipelines

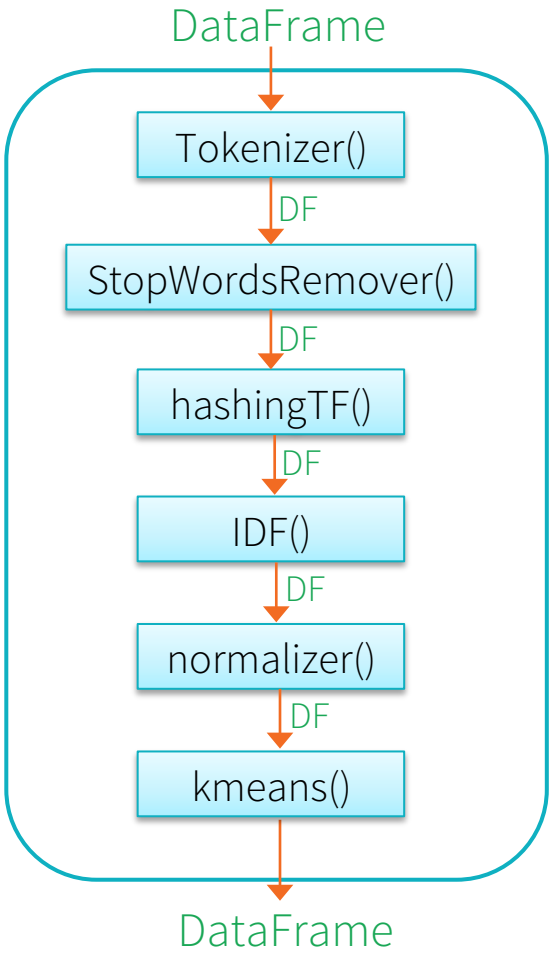
DataFrame: uses DF from Spark SQL as a ML dataset. Different columns can store text, feature vectors, true labels and predictions

Transformer: an algorithm which can transform one DataFrame into another DataFrame
(example: ML model is a transformer that transforms a DF with features into a DF with predictions)

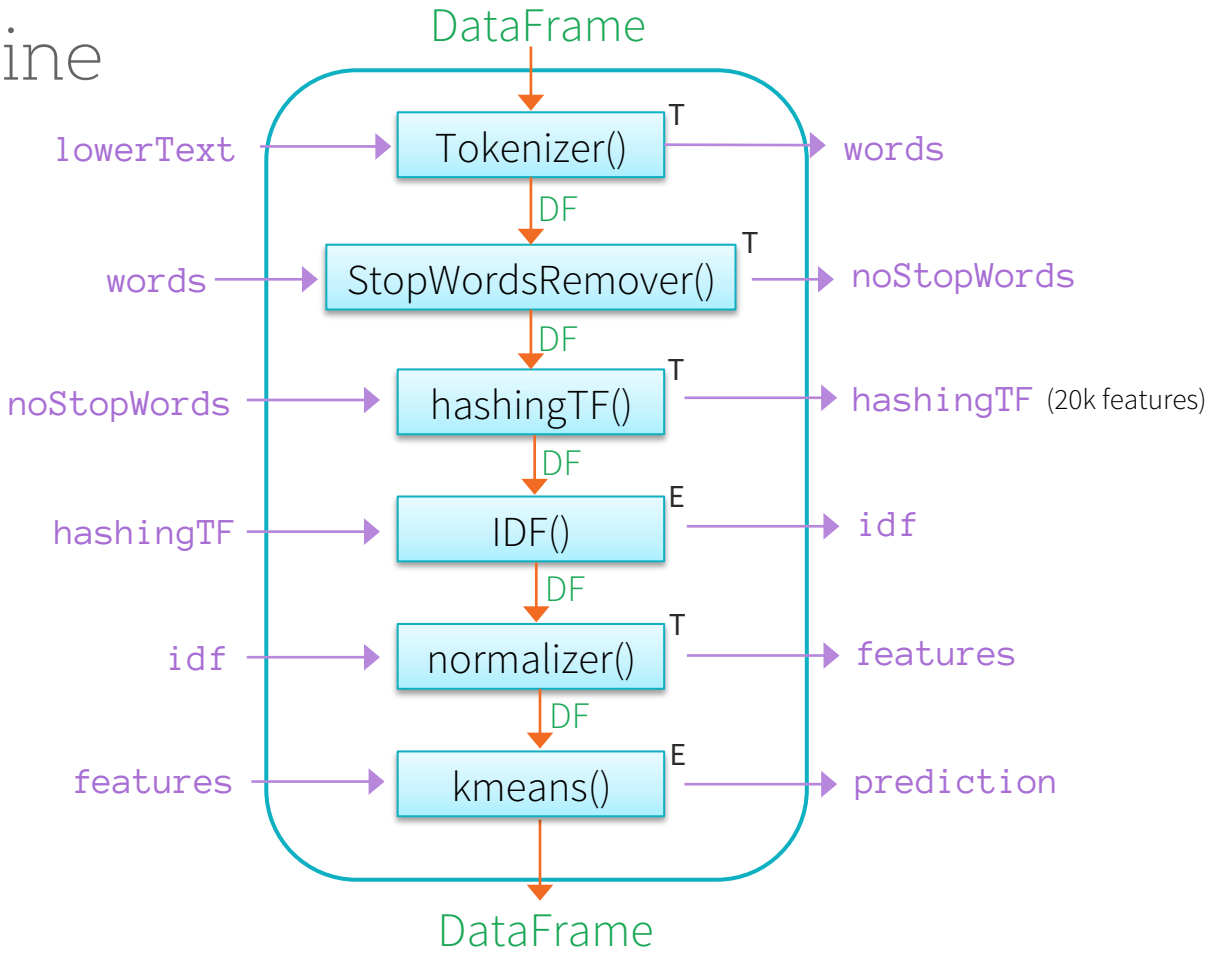
Estimator: an algorithm which can be fit on a DF to produce a Model
(example: a learning algorithm is an Estimator which trains on a DF and produces a model)

Pipeline: chains multiple Transformers and Estimators together to specify a ML workflow

Spark ML Pipeline



Spark ML Pipeline



Spark Streaming

Motivation

Stability condition for any streaming app

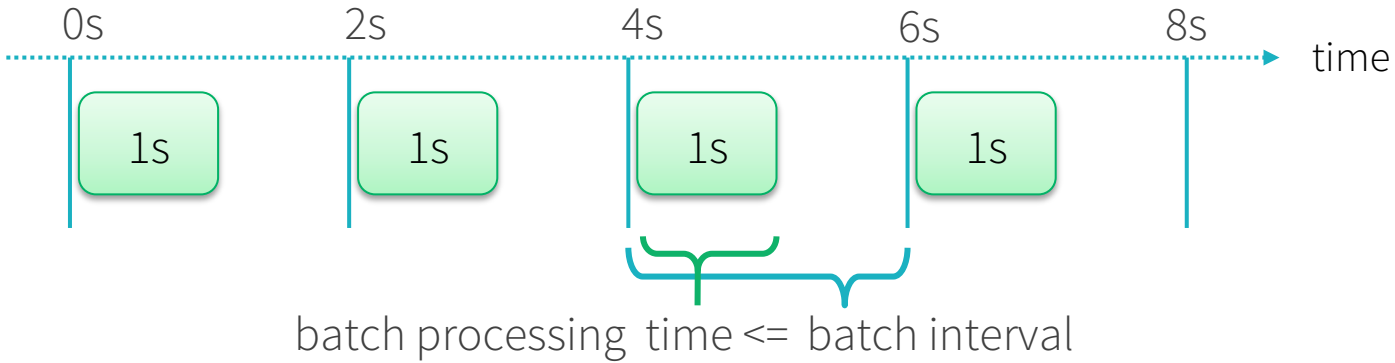
Receive data only as fast as the system can process it

Stability condition for Spark Streaming's "micro-batch" model

Finish processing previous batch before next one arrives

Stable micro-batch operation

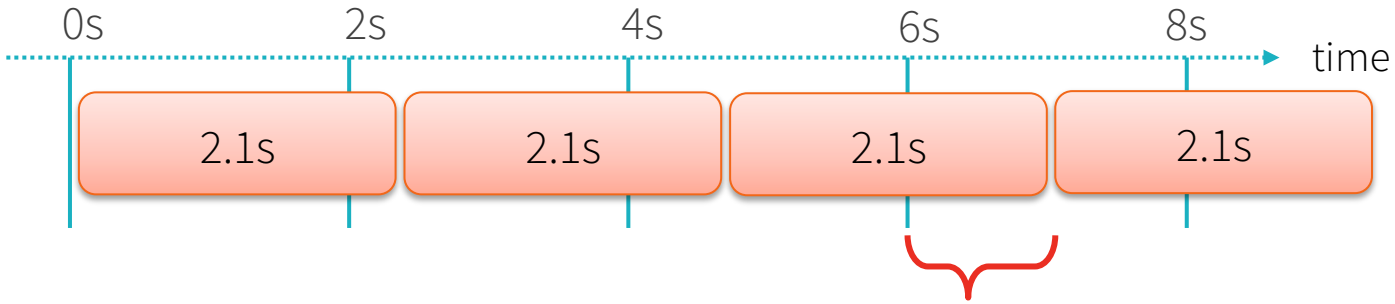
Spark Streaming runs micro-batches at fixed *batch intervals*



Previous batch is processed before next one arrives => stable

Unstable micro-batch operation

Spark Streaming runs micro-batches at fixed *batch intervals*



batch processing time > batch interval

scheduling delay builds up

Batches continuously gets delayed and backlogged => unstable

6 Edit Streams:

English : en  52.89.53.194:9002 ------->

German : de  54.68.10.240:9003 ------->

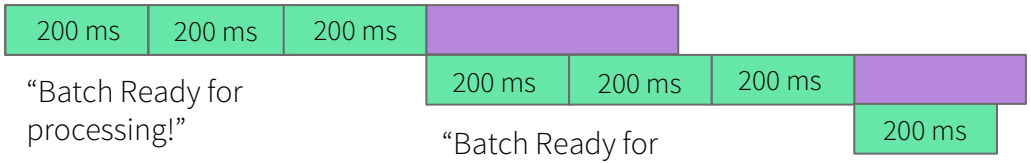
French : fr  54.68.10.240:9004 ------->

Russian : ru  54.68.10.240:9005 ------->

Italian : it  54.68.10.240:9006 ------->

Spanish : es  54.68.10.240:9007 ------->

Batch Interval: 600 ms

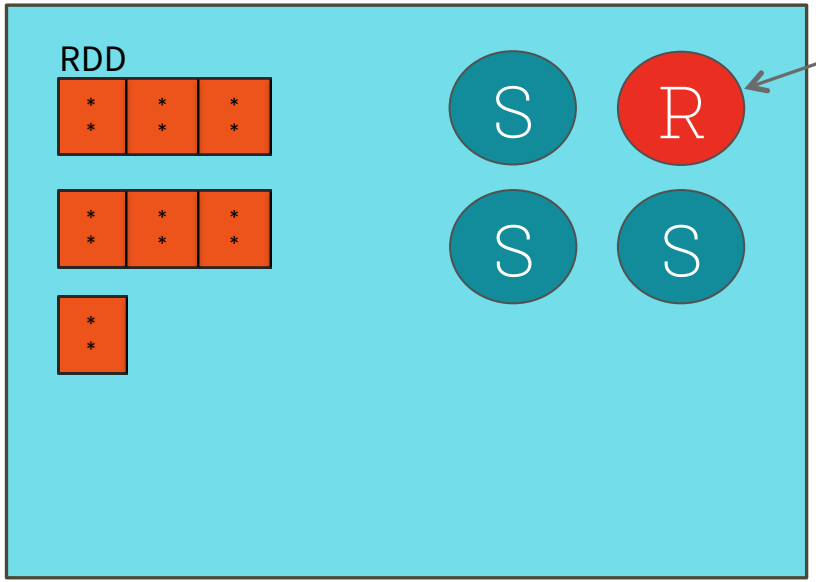


"Batch Ready for processing!"

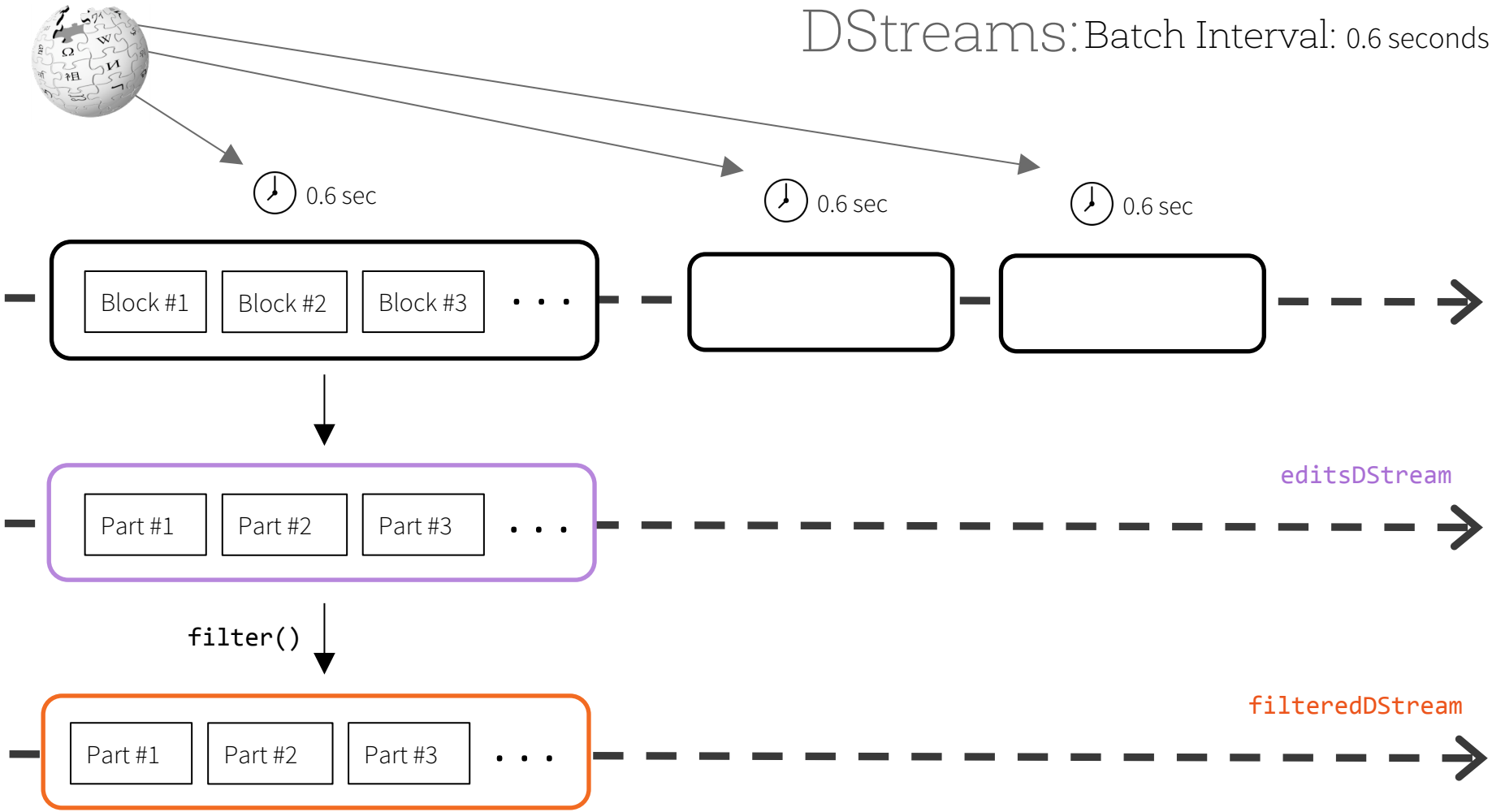
"Batch Ready for processing!"

:9002

English Edits

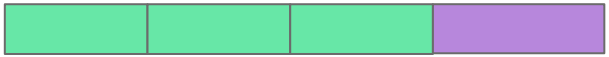


JVM

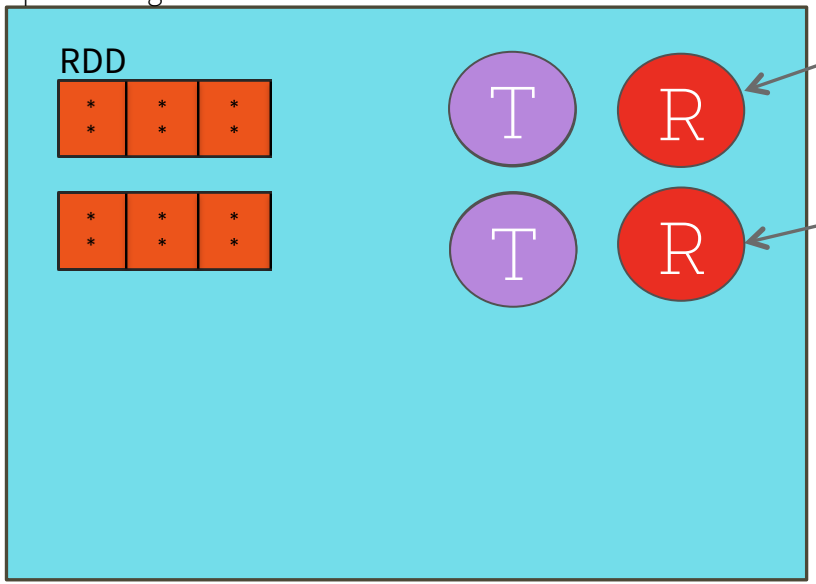

DStreams: Batch Interval: 0.6 seconds
0.6 sec
0.6 sec
0.6 sec
Block #1
Block #2
Block #3
editsDStream
Part #1
Part #2
Part #3
filter()
filteredDStream
Part #1
Part #2
Part #3

Batch Interval: 3 sec

"English Batch Ready for processing!"



"Spanish Batch Ready for processing!"



English Edits

Spanish Edits

Transformations on DStreams:

reduce()

filter()

map()

repartition()

flatMap()

updateStateByKey()

countByValue()

join()

union()

count()

transform()

reduceByKey()

cogroup()

Kafka:

Approach 1: Receiver-based Approach

This approach uses a Receiver to receive the data.

Approach 2: Direct Approach (No Receivers)

New receiver-less “direct” approach since Spark 1.3

Periodically queries Kafka for the latest offsets in each topic + partition and accordingly defines the offset ranges to process in each batch

```
val directKafkaStream = KafkaUtils.createDirectStream[
    [key class], [value class], [key decoder class], [value decoder class] ](
    streamingContext, [map of Kafka parameters], [set of topics to consume])
```

MLlib Operations on DStreams:

Streaming Linear Regression

Streaming Logistic Regression

Streaming KMeans

Online hypothesis testing (A/B testing)

Streaming DataFrames

Umbrella ticket to track what's needed to make streaming DataFrame a reality:

https://issues.apache.org/jira/browse/SPARK-8360

Streaming DataFrames

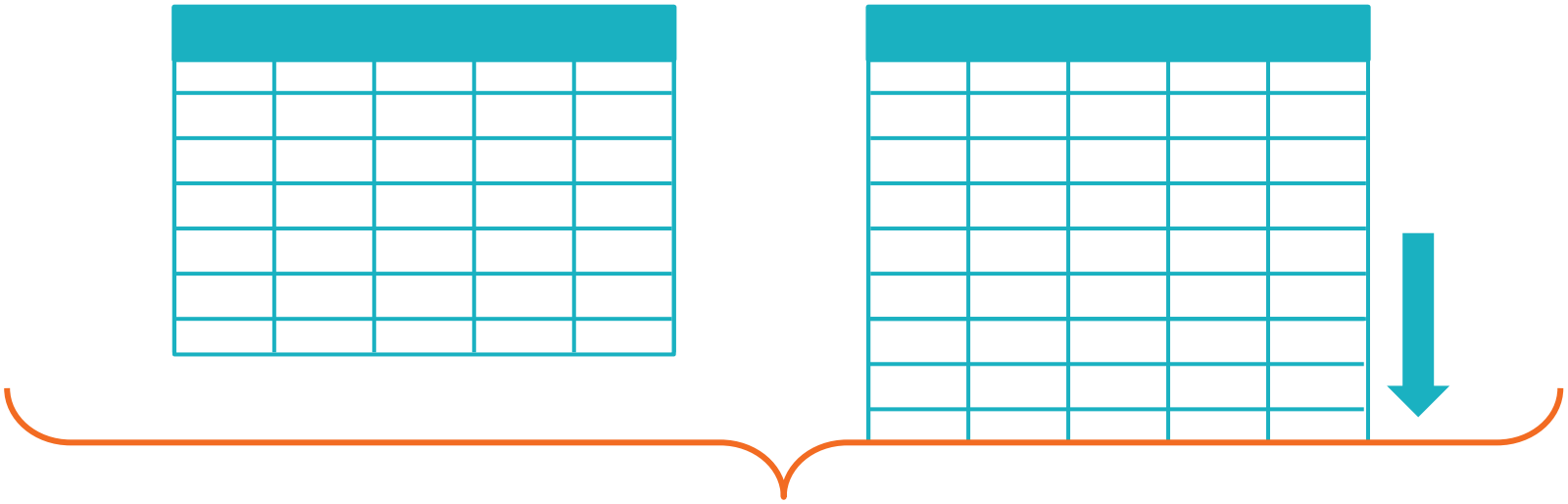
- Easier-to-use APIs (batch, streaming, and interactive)

```
val stream = read.kafka("...")
stream.window(5 mins, 10 secs)
  .agg(sum("sales"))
  .write.jdbc("mysql://...")
```

- And optimizations:
- - Tungsten backends
- - native support for out-of-order data
- - data sources and sinks

Spark 1.3
Static DataFrames

Spark 2.0
Continuous DataFrames

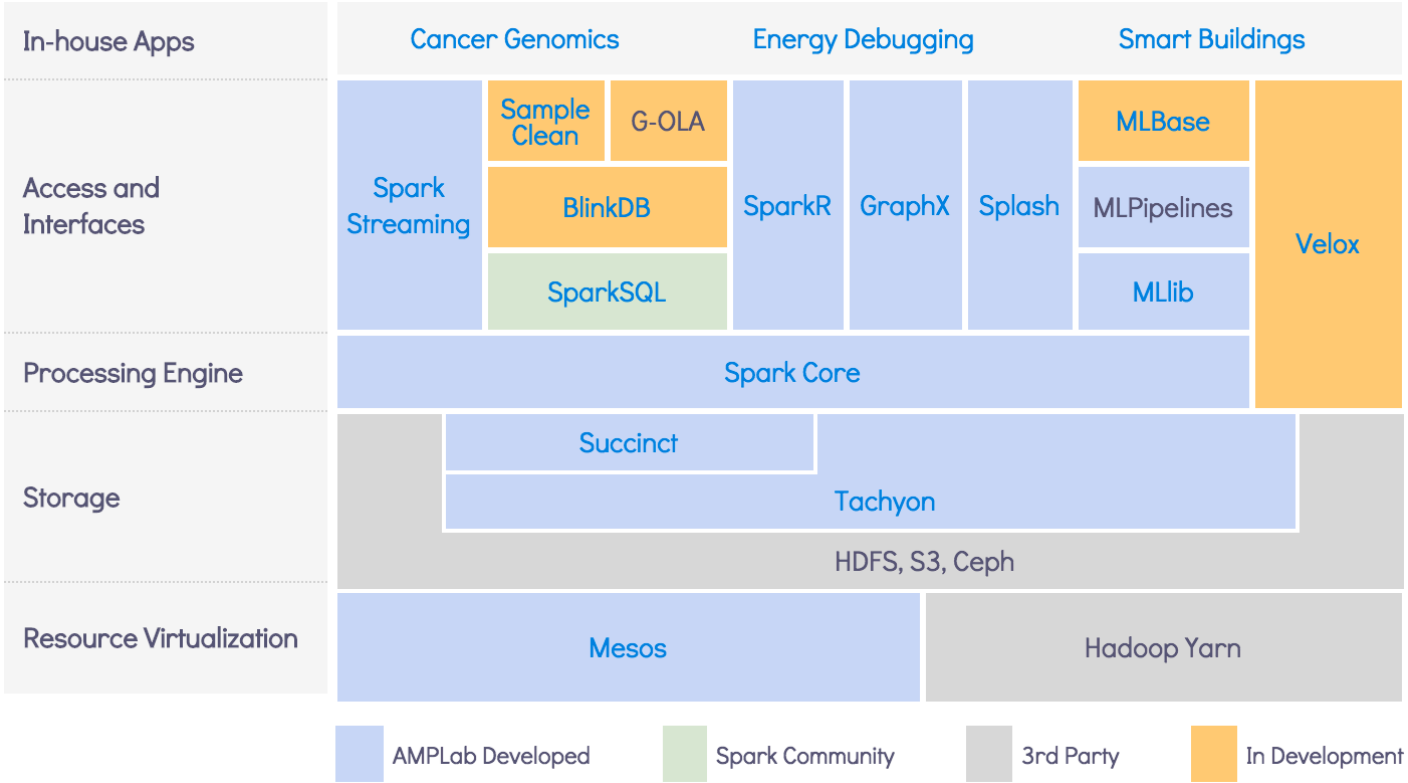


Single API !

Closing Remarks

BDAS: Berkeley Data Analytics Stack

amplab UC BERKELEY



Hardware Trends

	2010	2015	
Storage	50+MB/s (HDD)	500+MB/s (SSD)	10X
Network	1Gbps	10Gbps	10X
CPU	~3GHz	~3GHz	☹

Project Tungsten

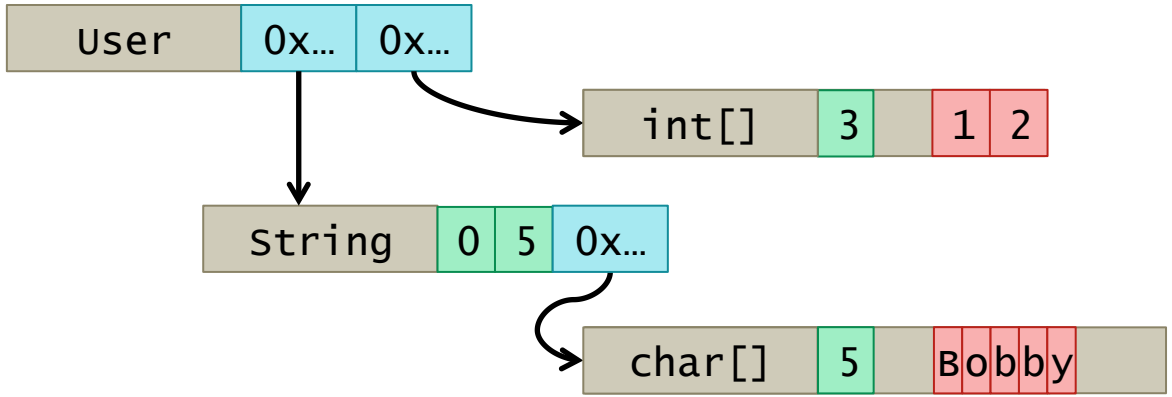
- Substantially speed up execution by optimizing CPU efficiency, via:

(1) Runtime code generation
(2) Exploiting cache locality
(3) Off-heap memory management

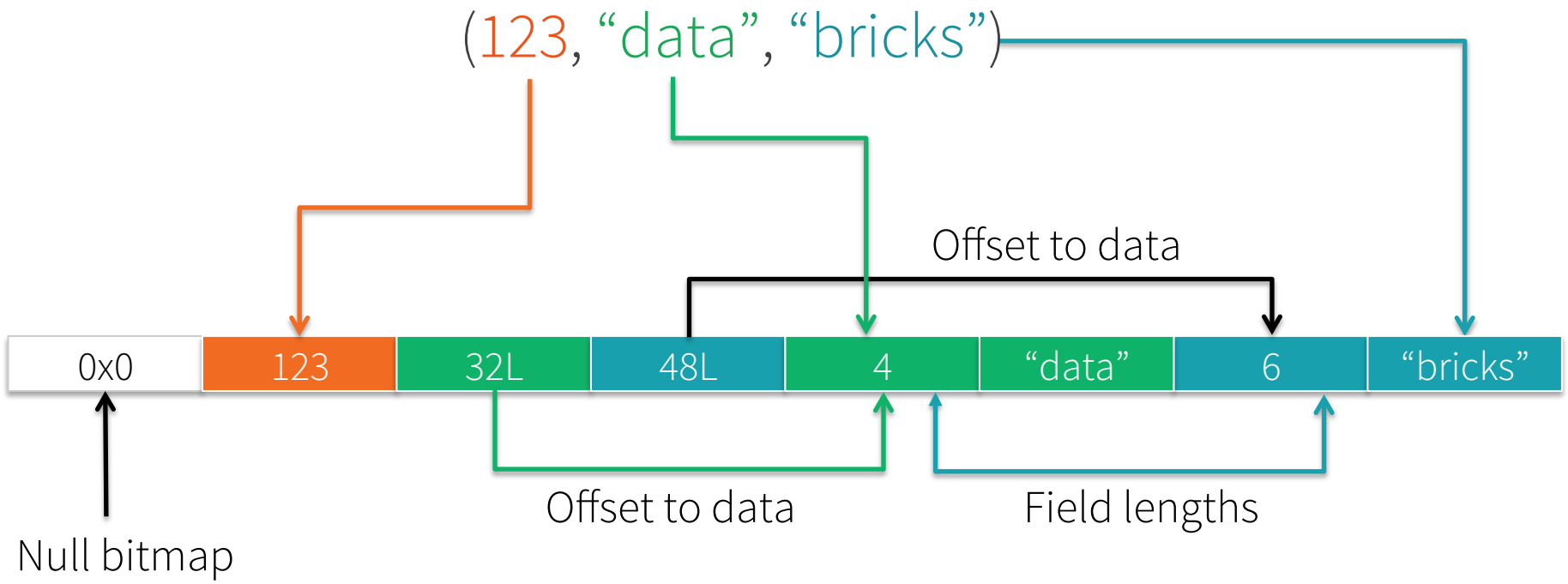
Challenge: Data Representation

- Java objects often many times larger than underlying fields

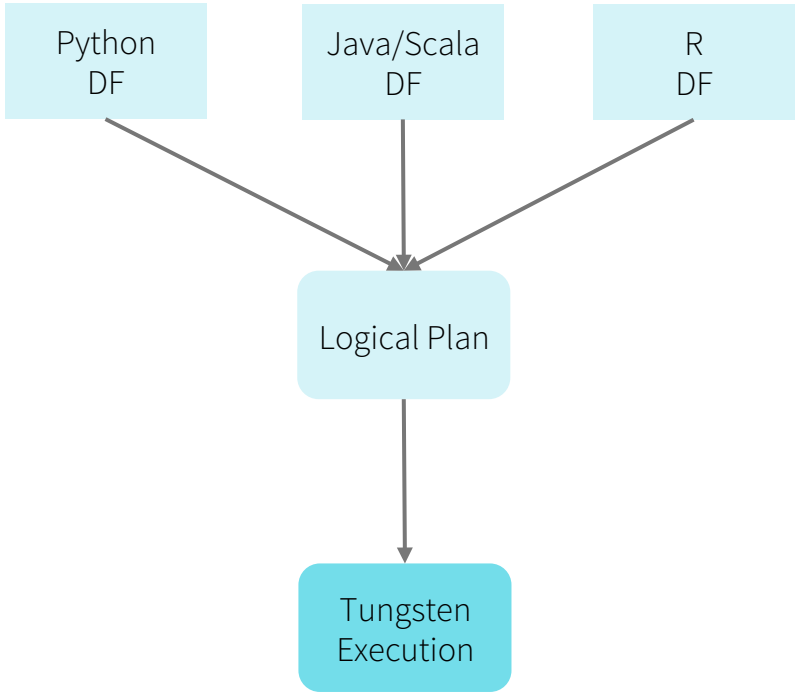
- `class User(name: String, friends: Array[Int])`
- `User("Bobby", Array(1, 2))`



Tungsten's Compact Encoding



From DataFrame to Tungsten



Initial phase in Spark 1.5

More work coming in 2016

BLOG > ANNOUNCEMENTS , VIRTUAL MACHINES

Largest VM in the Cloud

THURSDAY, JANUARY 8, 2015

DREW MCDANIEL
Principal Program Manager, Azure

G-Series Size Details

VM Size	Cores	RAM		
Standard_G1	2	2		
Standard_G2	4	5		
Standard_G3	8	11		
Standard_G4	16	22		
Standard_G5	32	448 GiB	6596 GB	64

AWS Announces X1 Instances For EC2 With 2TB Of Memory, Launching Next Year

Posted Oct 8, 2015 by Frederic Lardinois (@fredericl)

926 SHARES

Amazon today announced a massive new instance type for its AWS EC2 compute service. The

CrunchBase

Amazon

FOUNDED
1994

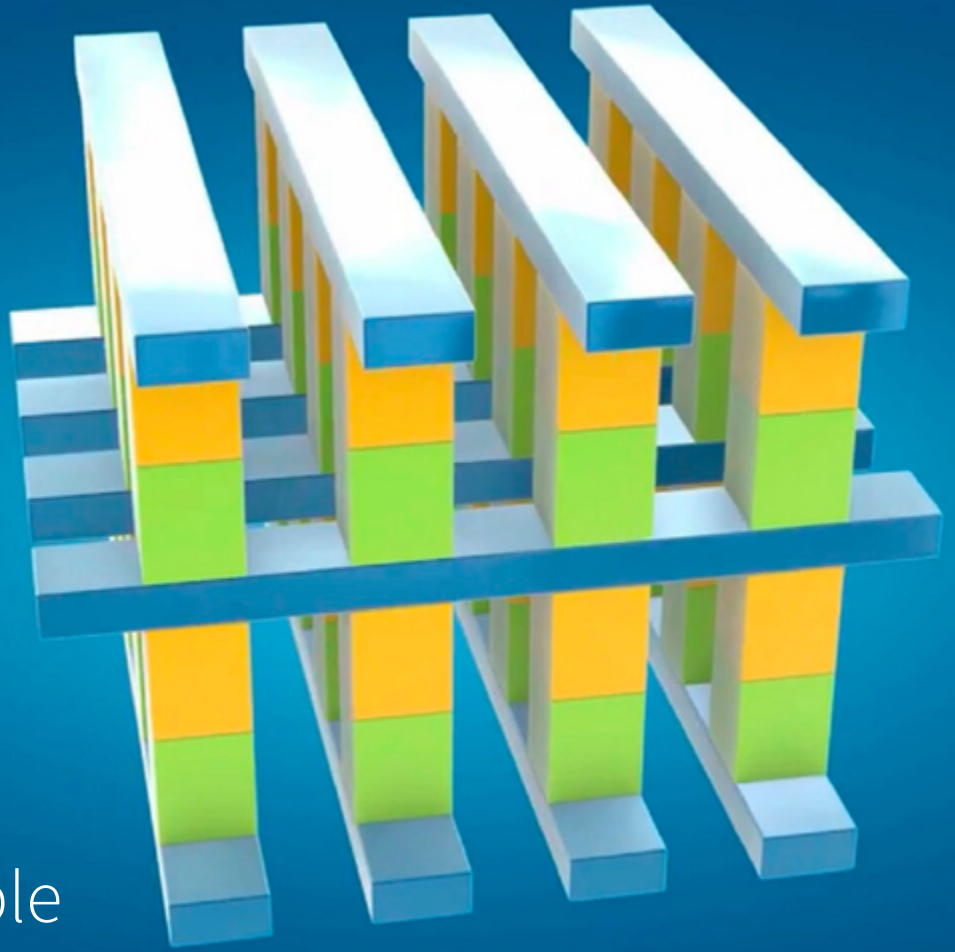
OVERVIEW
Amazon is an e-commerce retailer formed to provide consumers with products in tw It offers users with merchandise and cont purchased for resale from vendors and th by third-party sellers. Operating in North and International markets, Amazon provi services through websites such as amazo amazon.ca. It also enables authors, music filmmakers, ...

LOCATION
Seattle, WA

CATEGORIES
E-Commerce, Crowdsourcing, Groceries, Co Goods, Delivery, Software, Retail, Internet

FOUNDERS

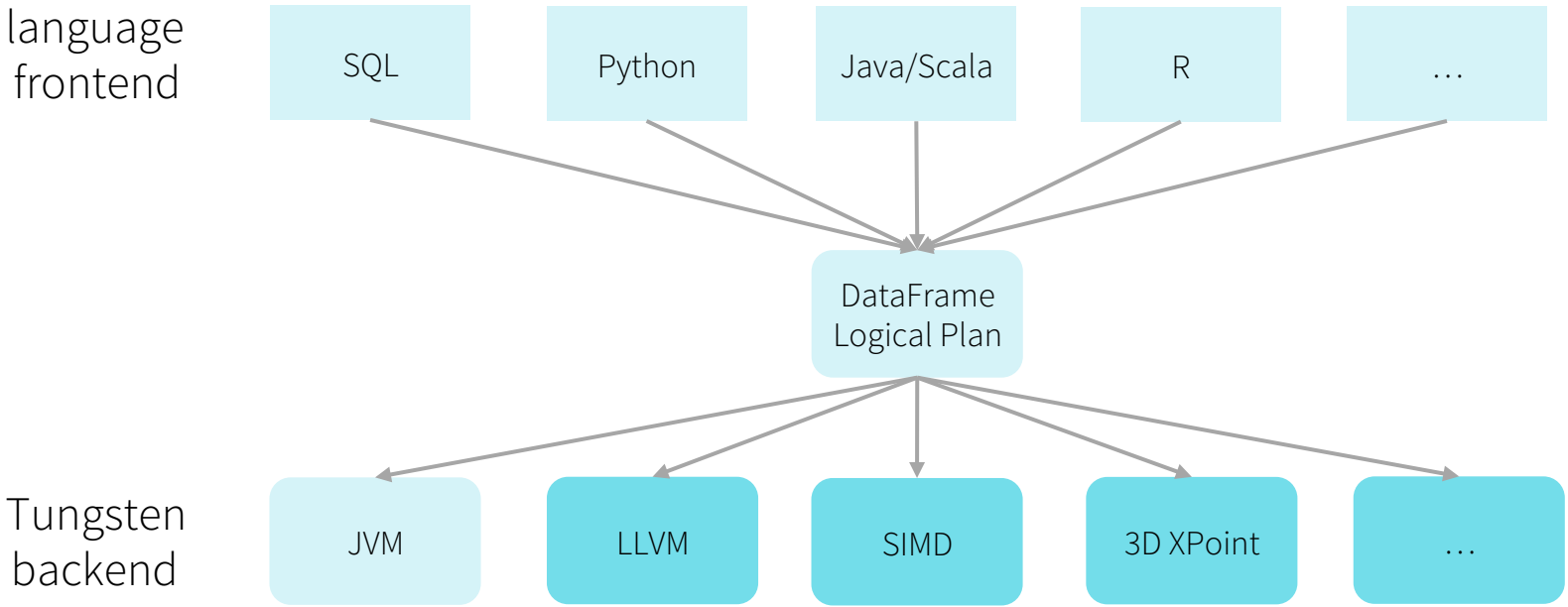
3D XPoint



- DRAM latency
- SSD capacity
- Byte addressible

Learn more: https://en.wikipedia.org/wiki/3D_XPoint

Unified API, One Engine, Automatically Optimized



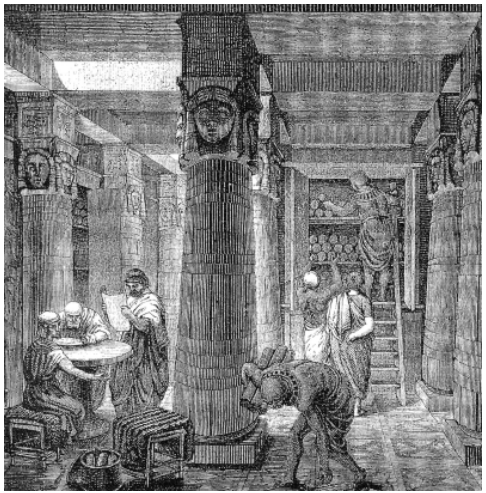
(1 Nov 2015)

Q) How many articles are needed to cover the sum of all human knowledge?

104 million!

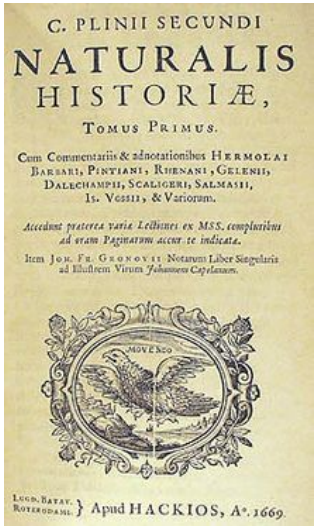
Source: https://en.wikipedia.org/wiki/User:Emijrp/All_human_knowledge

Library of Alexandria



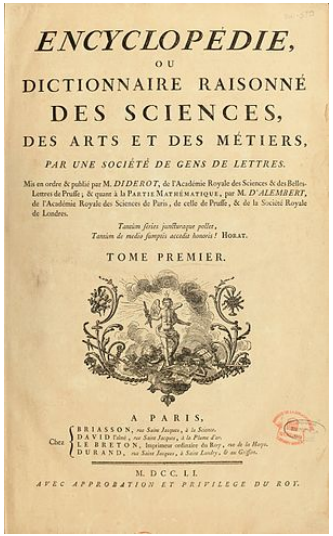
- From Egypt
- 3rd century B.C.
- 400,000 scrolls
- Burned down by Julius Caesar

Naturalis Historia



- From Roman Empire
- 77 A.D.
- 37 books

L ‘Encyclopédie

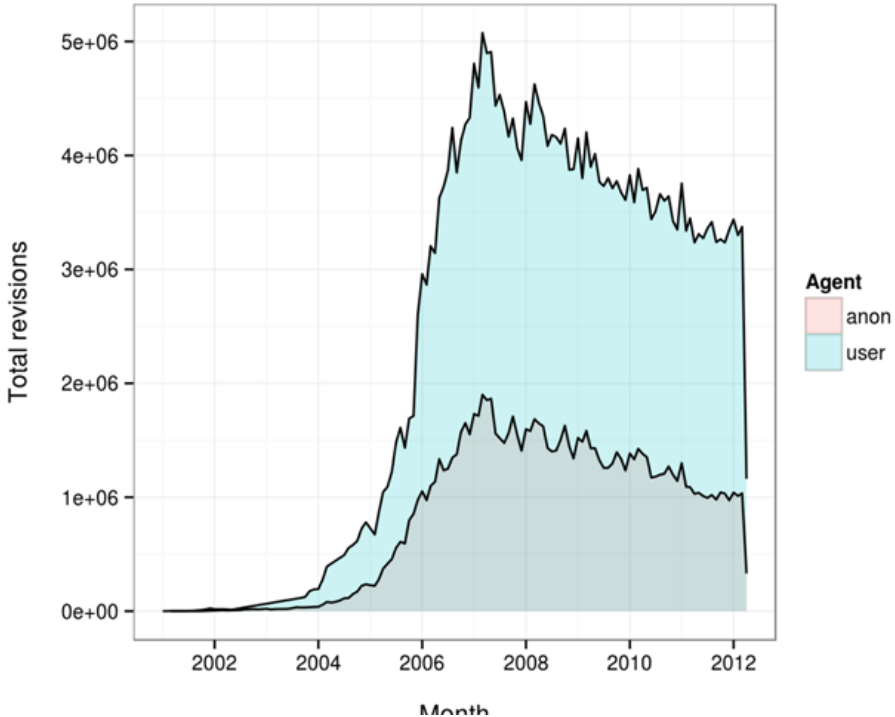


- French
- 1751
- 17 volumes
- 18,000 pages of text
- 75,000 entries

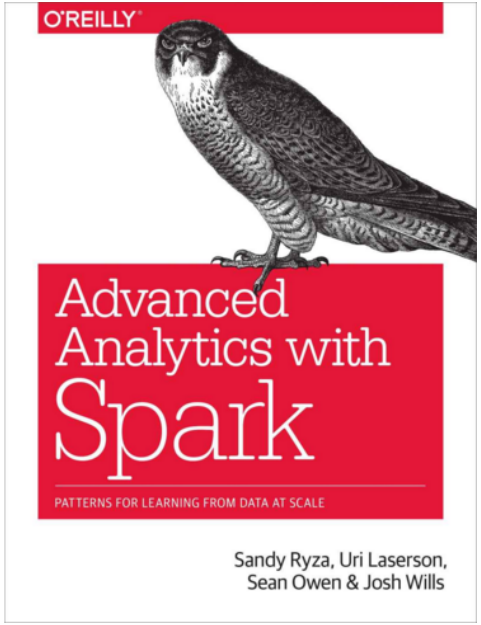
Upcoming Challenges

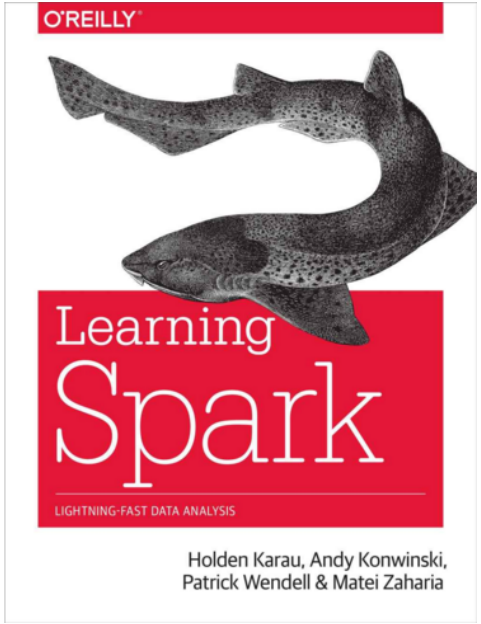
- Rise of smartphones
- Sock puppets, bias, paid editors…

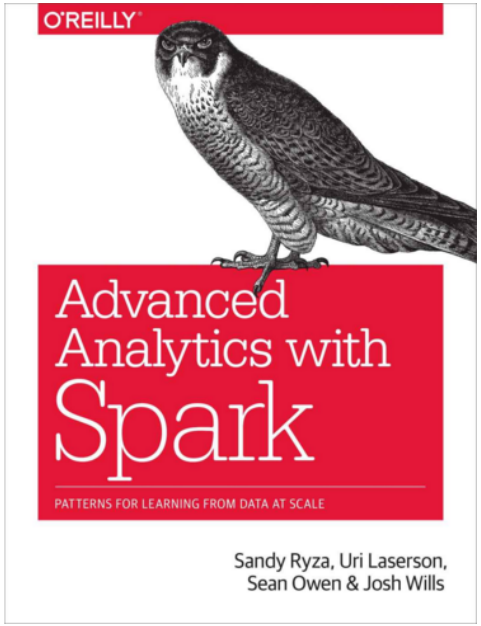
Drop in # of Edits:



Source:
https://meta.wikimedia.org/wiki/Research:Anonymous_editor_acquisition/Volume_and_impact


O'REILLY
Advanced Analytics with Spark
PATTERNS FOR LEARNING FROM DATA AT SCALE
Sandy Ryza, Uri Laserson,
Sean Owen & Josh Wills


O'REILLY
Learning Spark
LIGHTNING-FAST DATA ANALYSIS
Holden Karau, Andy Konwinski,
Patrick Wendell & Matei Zaharia



Chapter 6: Understanding Wikipedia with Latent Semantic Analysis

- The Term-Document Matrix
- Getting the Data
- Parsing and Preparing the Data
- Lemmatization
- Computing the TF-IDFs
- Singular Value Decomposition
- Finding Important Concepts
- Querying and Scoring with the Low-Dimensional Representation
- Term-Term Relevance
- Document-Document Relevance
- Term-Document Relevance
- Multiple-Term Queries

You will have access to Databricks and these datasets via Community Edition after class…

Good luck!