Big Data Platforms

Yogesh Simmhan

simmhan@iisc.ac.in

Slide Credits:

- https://stanford.edu/~rezab/sparkclass/slides/itas_workshop.pdf
- https://www.slideshare.net/deanchen11/scala-bay-spark-talk
- https://databricks-training.s3.amazonaws.com/slides/advanced-spark-training.pdf
- Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing, M. Zaharia, et al., NSDI 2012
- http://spark.apache.org/docs/latest/programming-guide.html

2020/01/23

What is Big Data?

Image credits: http://www.seekbig.in/1128-tnpsc-economics-questions/

The term ***is*** fuzzy … *Handle with care!*

Wordle of "Thought Leaders'" definition of Big Data, © Jennifer Dutcher, 2014
https://datascience.berkeley.edu/what-is-big-data/

So…What is Big Data?

Data whose characteristics exceeds the capabilities of conventional *algorithms*, ***systems*** *and techniques* to derive useful value.

https://www.oreilly.com/ideas/what-is-big-data

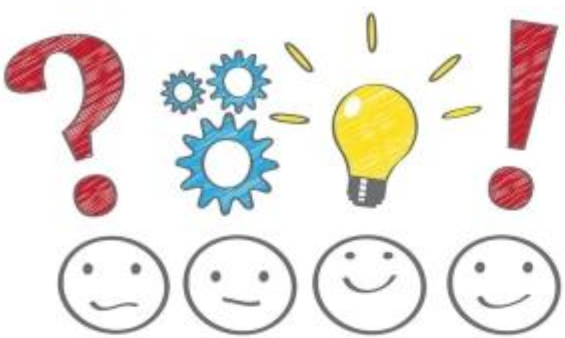
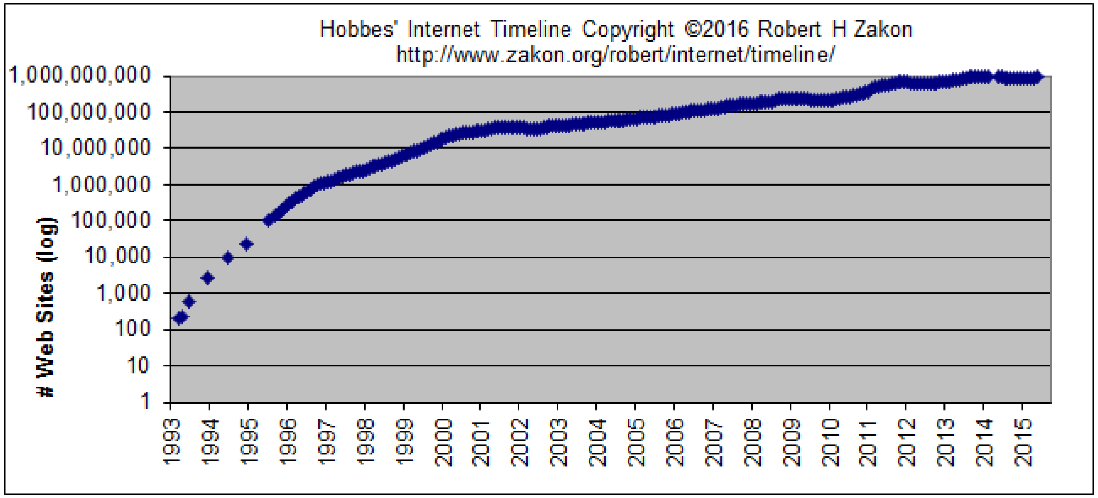


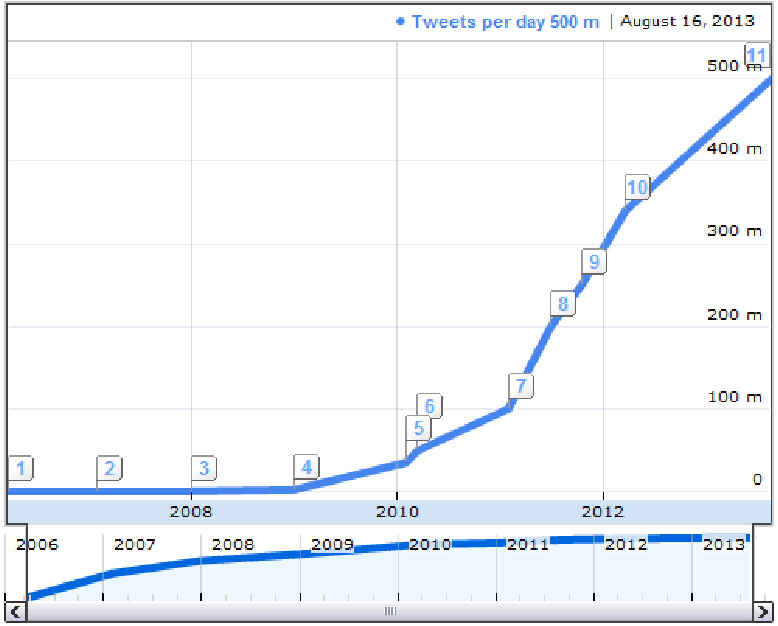
Image Credits: https://community.uservoice.com/wp-content/uploads/benefits-of-effective-questions-800x448-300x168.jpg

And, where does Big Data come from?

Web & Social Media

- Web search, Social Networks & Micro-blogs

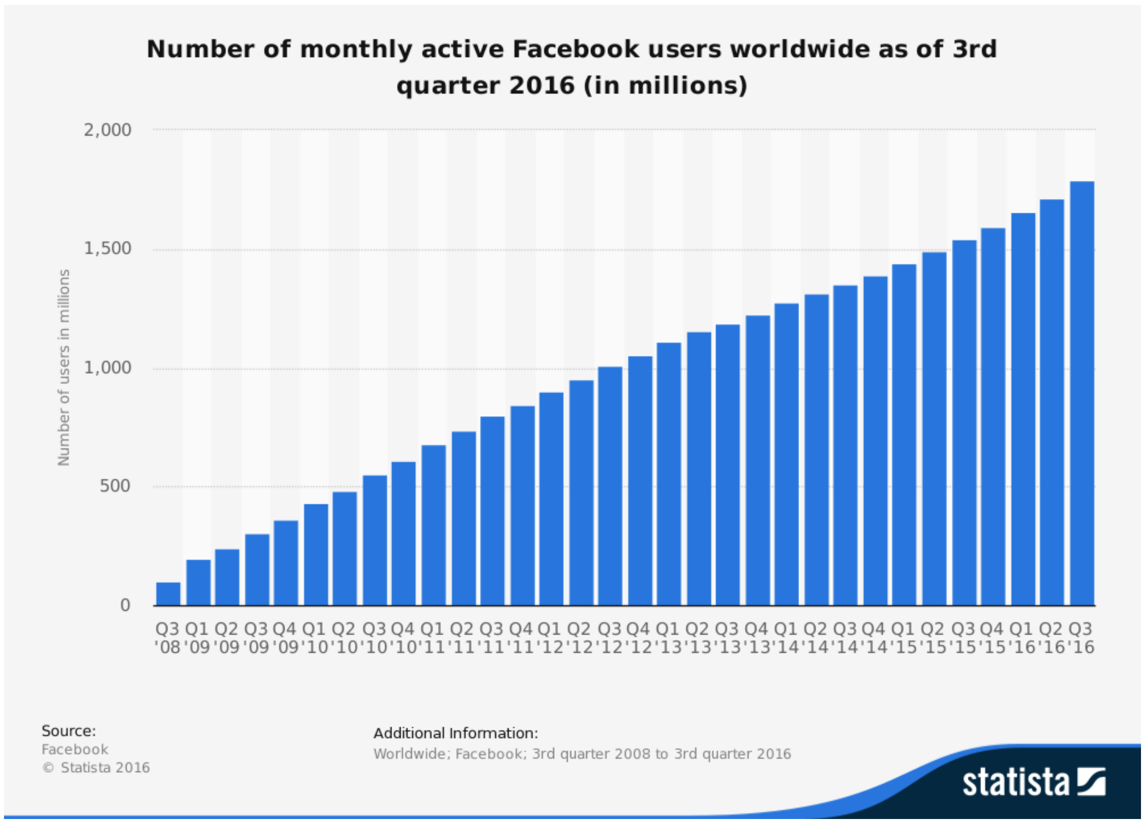


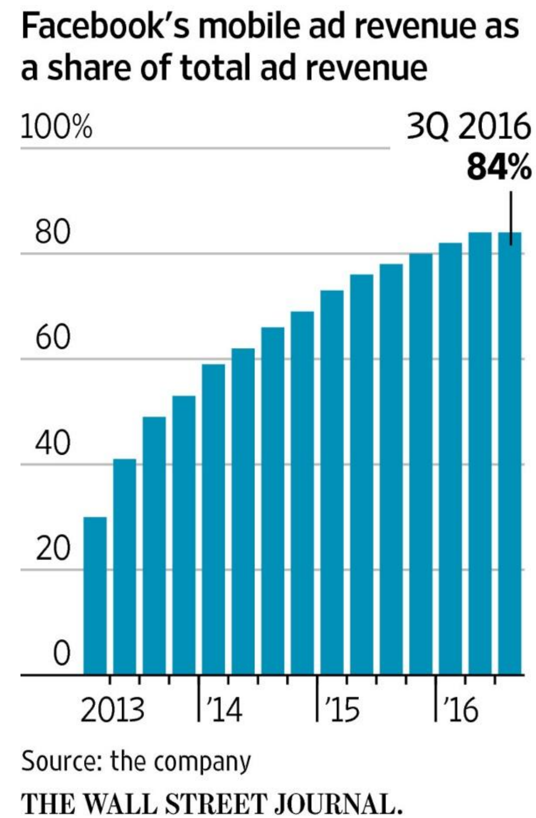


http://static4.businessinsider.com/image/56b089cedd0895437c8b45ef-2390-1265/untitled.png
http://www.internetlivestats.com/twitter-statistics/

Web & Social Media

- Social Networks & Micro-blogs



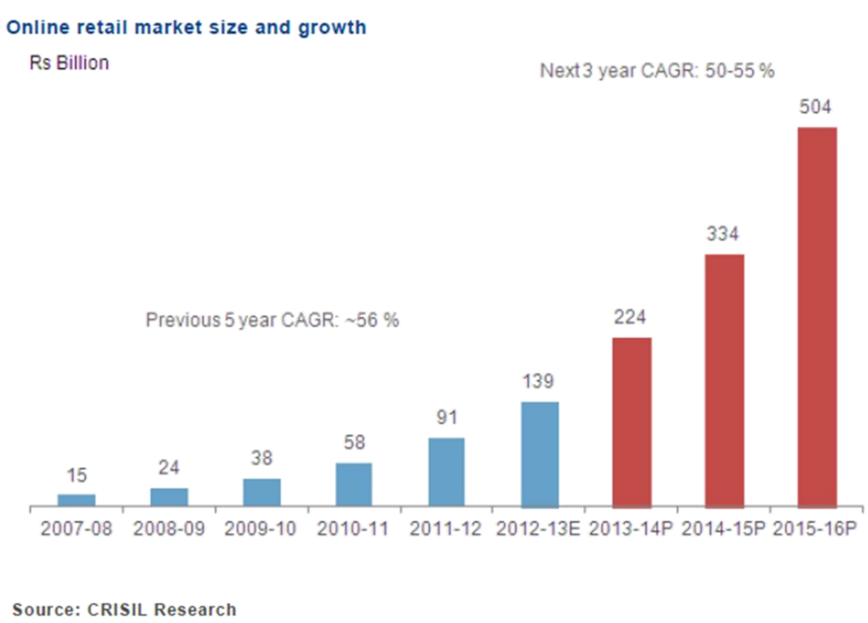


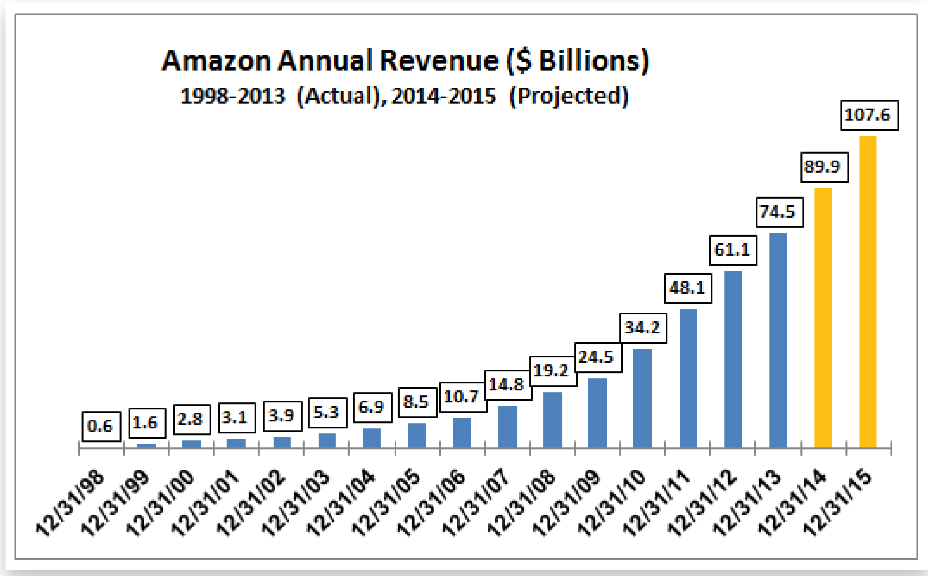
1.79 billion monthly active users as of September 30, 2016

https://www.statista.com/statistics/264810/number-of-monthly-active-facebook-users-worldwide/
http://www.wsj.com/articles/facebook-profit-jumps-sharply-1478117646
http://newsroom.fb.com/company-info/

Enterprises & Government

- Online retail & eCommerce



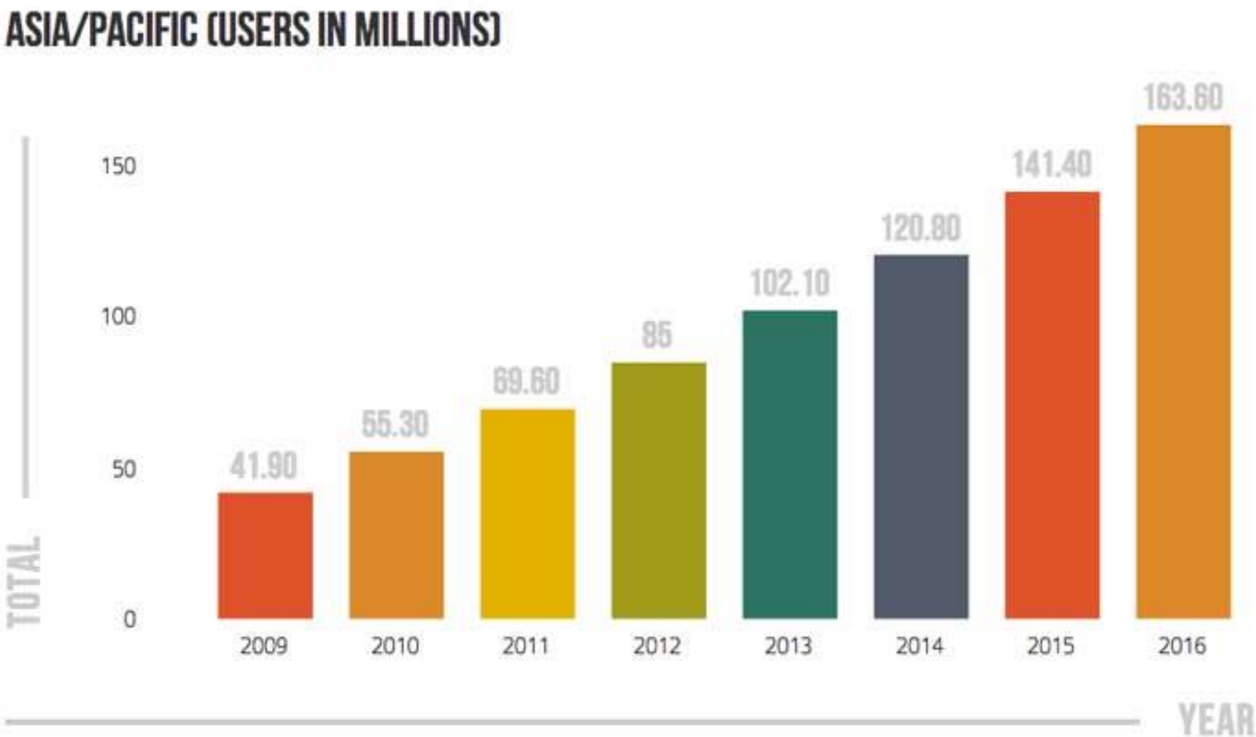


http://blogs.ft.com/beyond-brics/2014/02/28/online-retail-in-india-learning-to-evolve/

http://www.peridotcapital.com/2014/04/amazon-sales-growth-projections-for-next-two-years-appear-overly-optimistic.html

Enterprises & Government: Finance

- Mobile Transactions & FinTech

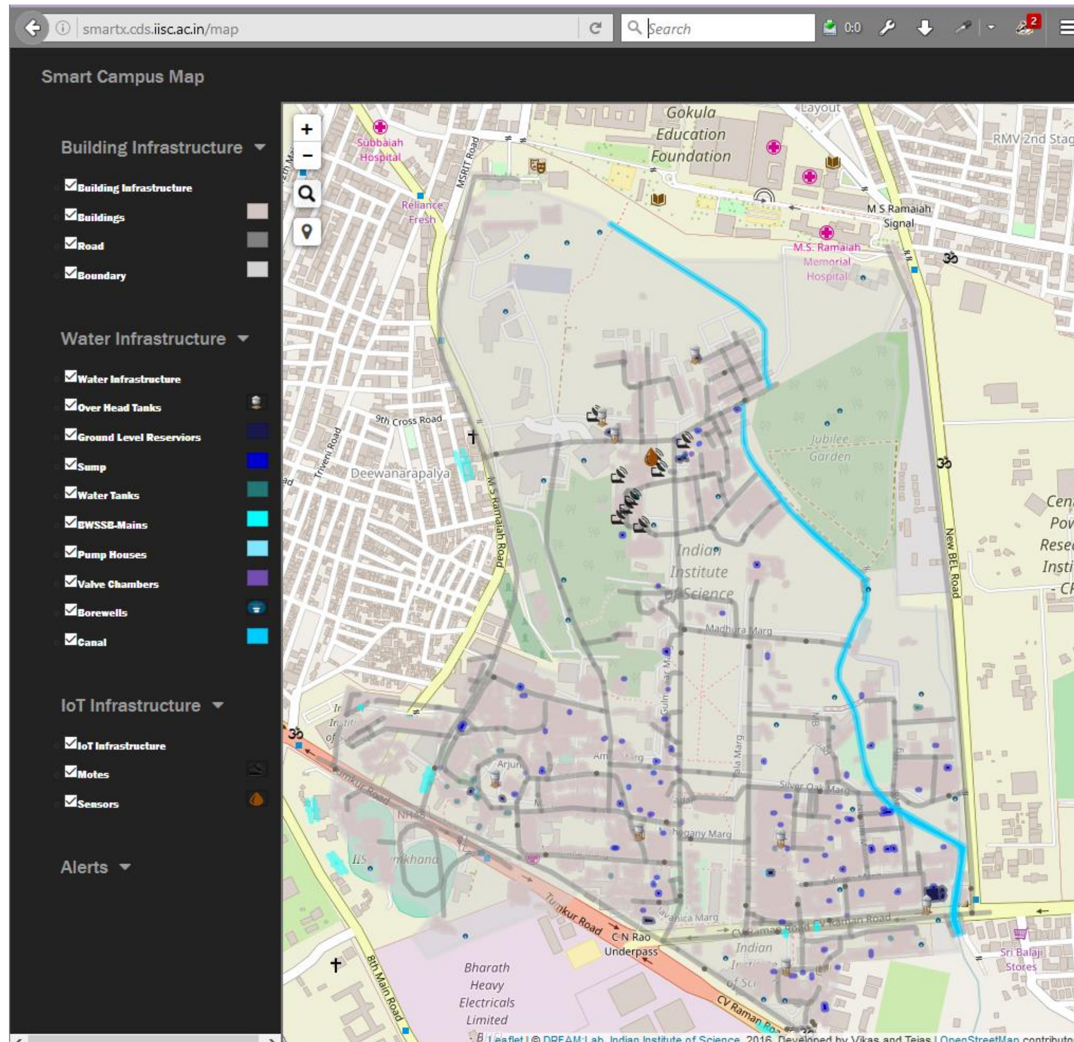


Since November 8, 2016, *Paytm* has surpassed its metrics -tripling *transactions per day* to 7.5 million

http://www.pymnts.com/in-depth/2015/mobile-transactions/
Is Paytm the Xerox of mobile payments?, ETtech.com-03-Jan-2017

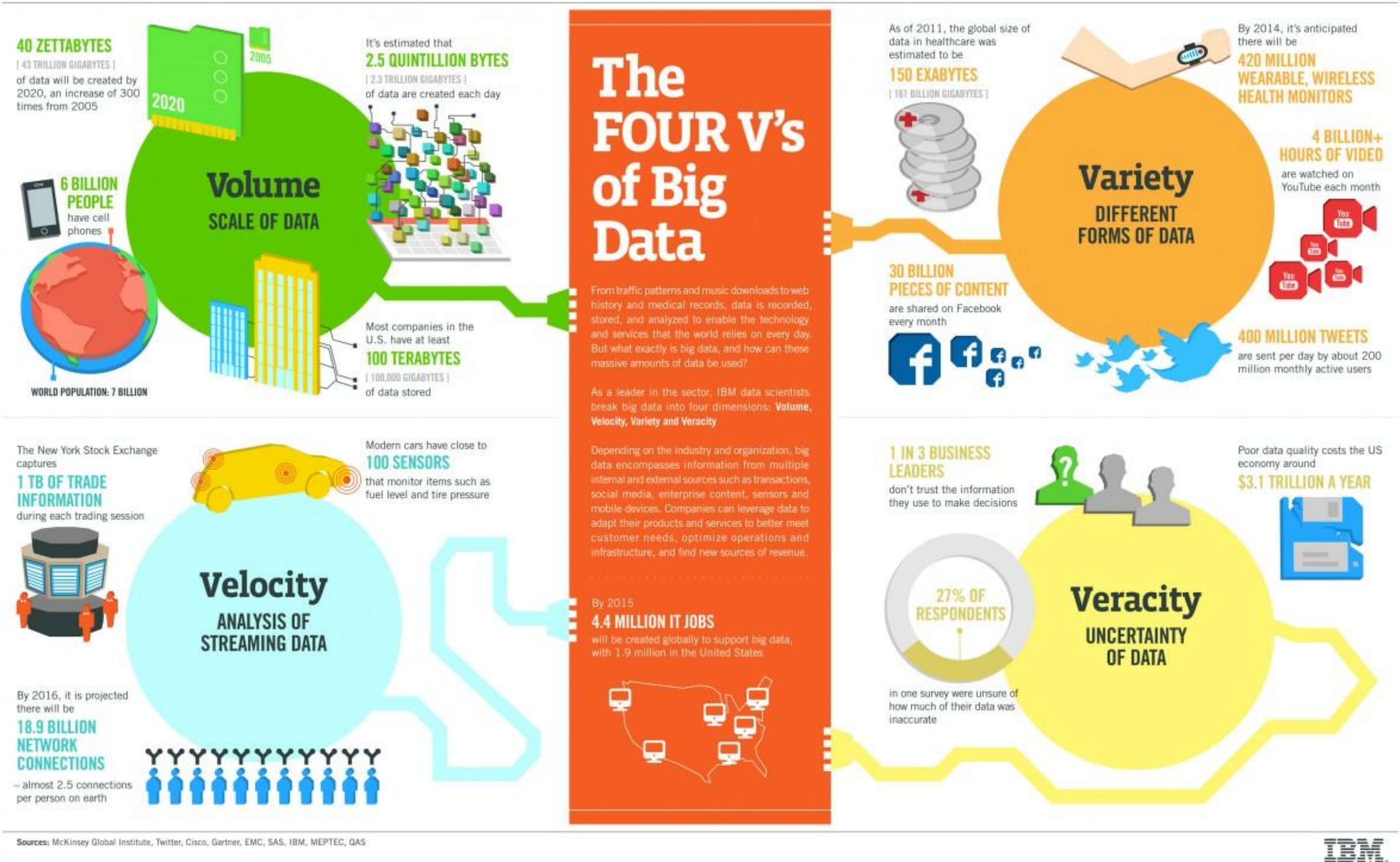
Internet of Everything

- Personal Devices
  - Smart Phones, Fitbit
- Smart Appliances
- Smart Cities
  - Power, Water, Transportation, Environment
- Smart Retail
- Millions of sensor data streams

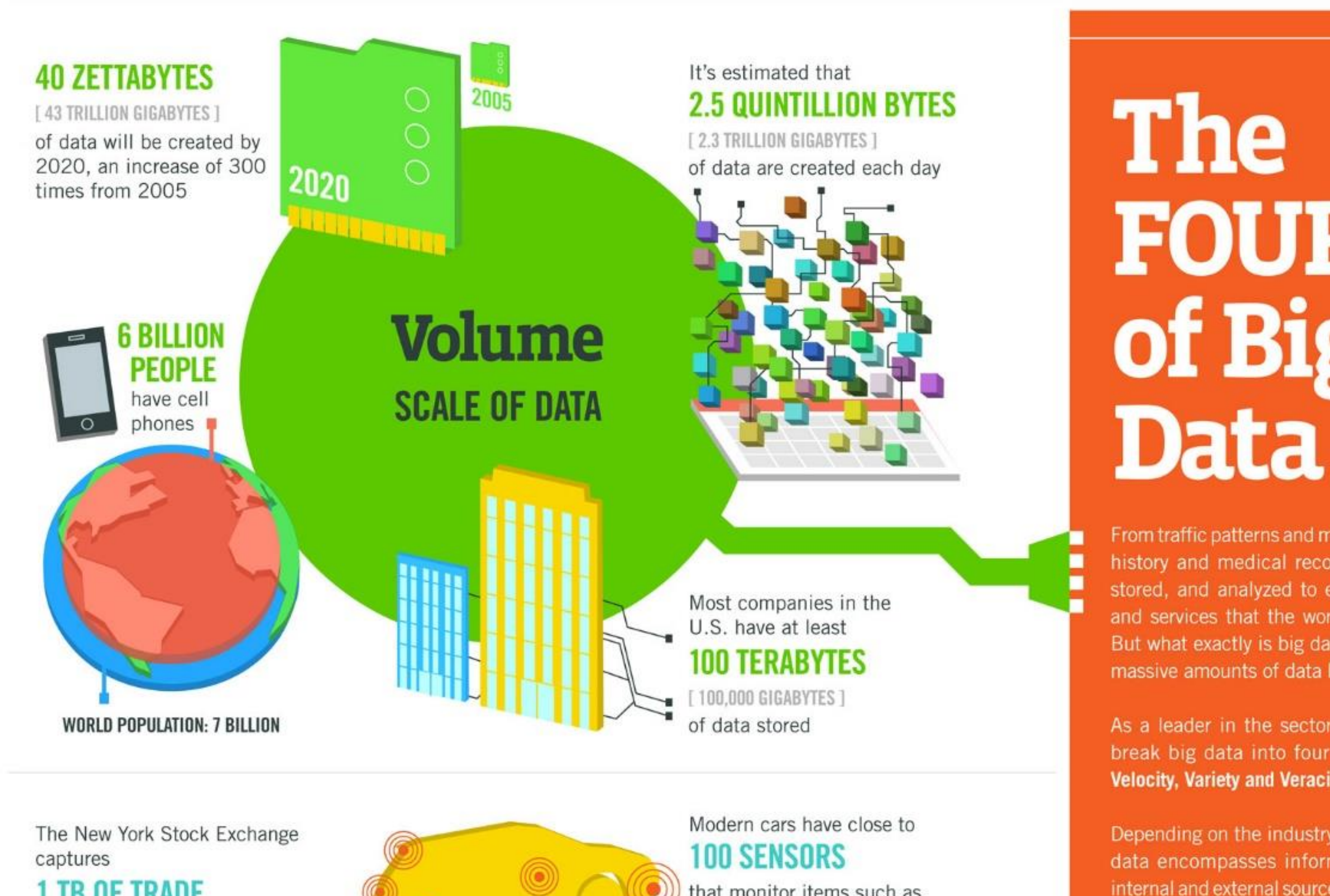


smartx.cds.iisc.ac.in

Why is Big Data Difficult?



http://www.ibmbigdatahub.com/infographic/four-vs-big-data


40 ZETTABYTES
[43 TRILLION GIGABYTES]
of data will be created by 2020, an increase of 300 times from 2005
2020
2005
It's estimated that
2.5 QUINTILLION BYTES
[2.3 TRILLION GIGABYTES]
of data are created each day
Volume
SCALE OF DATA
6 BILLION PEOPLE
have cell phones
WORLD POPULATION: 7 BILLION
Most companies in the U.S. have at least
100 TERABYTES
[100,000 GIGABYTES]
of data stored
The New York Stock Exchange captures
Modern cars have close to
100 SENSORS
The

WORLD POPULATION: 7 BILLION

U.S. have at least

100 TERABYTES

[100,000 GIGABYTES]

of data stored

and services that the
But what exactly is big
massive amounts of dat

As a leader in the sec
break big data into fo
Velocity, Variety and Vera

Depending on the indus
data encompasses inf
internal and external sou
social media, enterpris
mobile devices. Compa
adapt their products an
customer needs, opt
infrastructure, and find

The New York Stock Exchange captures

1 TB OF TRADE INFORMATION

during each trading session

Modern cars have close to

100 SENSORS

that monitor items such as fuel level and tire pressure

Velocity

ANALYSIS OF STREAMING DATA

By 2015

4.4 MILLION IT JO

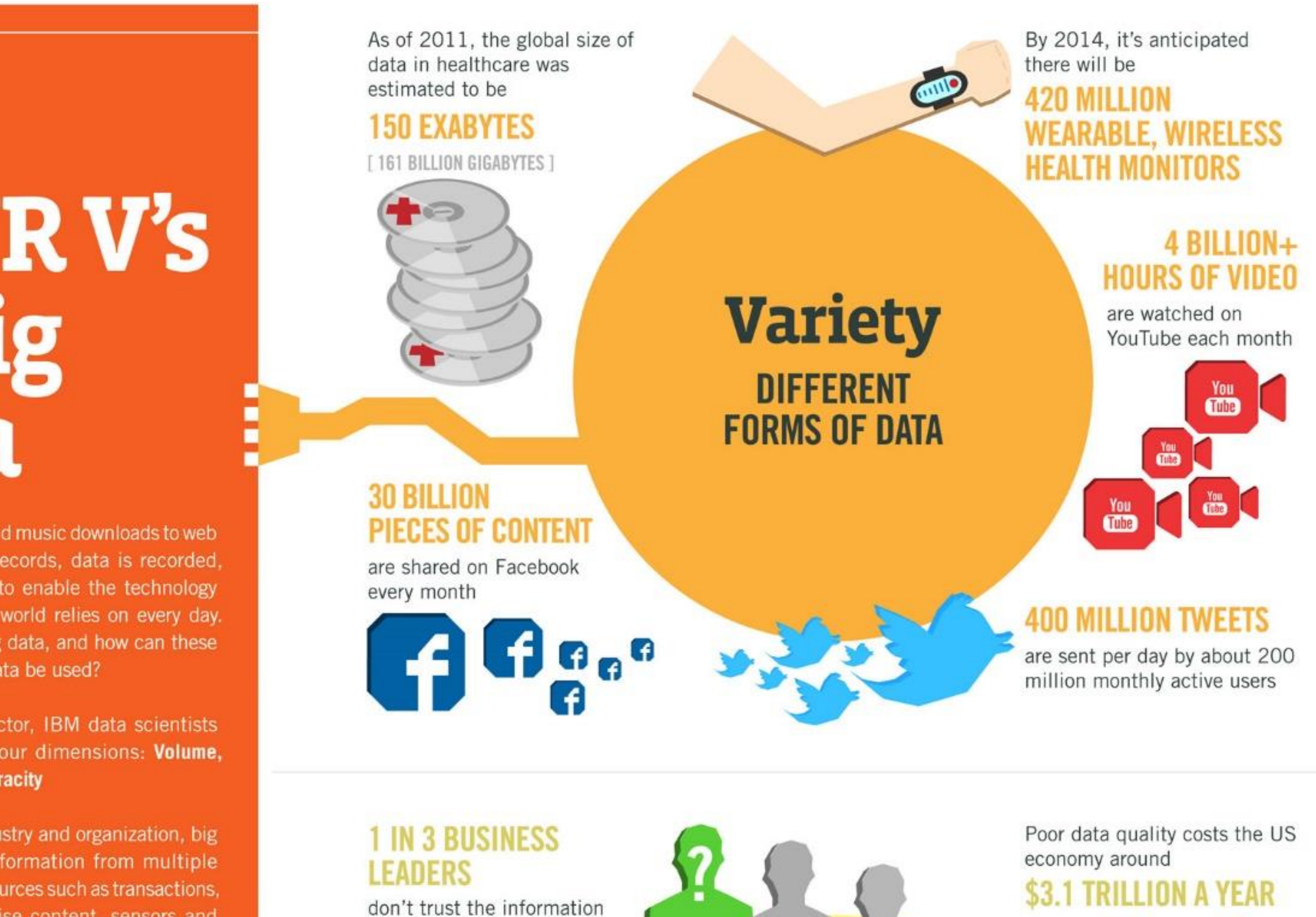
will be created globally
with 1.9 million in the

By 2016, it is projected there will be

18.9 BILLION NETWORK CONNECTIONS

– almost 2.5 connections per person on earth

Sources: McKinsey Global Institute, Twitter, Cisco, Gartner, EMC, SAS, IBM, MEPTEC, QAS


R V's
ig
d music downloads to web
ecords, data is recorded,
to enable the technology
world relies on every day.
data, and how can these
ta be used?
ctor, IBM data scientists
our dimensions: Volume,
racity
stry and organization, big
formation from multiple
urces such as transactions,
As of 2011, the global size of data in healthcare was estimated to be
150 EXABYTES
[161 BILLION GIGABYTES]
By 2014, it's anticipated there will be
420 MILLION WEARABLE, WIRELESS HEALTH MONITORS
Variety
DIFFERENT FORMS OF DATA
4 BILLION+ HOURS OF VIDEO
are watched on YouTube each month
30 BILLION PIECES OF CONTENT
are shared on Facebook every month
400 MILLION TWEETS
are sent per day by about 200 million monthly active users
1 IN 3 BUSINESS LEADERS
don't trust the information
Poor data quality costs the US economy around
$3.1 TRILLION A YEAR

rld relies on every day.
ata, and how can these
be used?

r, IBM data scientists
r dimensions: **Volume,**
ity

y and organization, big
rmation from multiple
ces such as transactions,
content, sensors and
es can leverage data to
services to better meet
mize operations and
ew sources of revenue.

S

o support big data,
nited States

every month

400 MILLION TWEETS

are sent per day by about 200 million monthly active users

1 IN 3 BUSINESS LEADERS

don't trust the information they use to make decisions

Poor data quality costs the US economy around

$3.1 TRILLION A YEAR

27% OF RESPONDENTS

in one survey were unsure of how much of their data was inaccurate

Veracity

UNCERTAINTY OF DATA

Data Analysis Lifecycle

Acquire

- Acquire Data
  - Sensors, Web logs & crawls, Transactions

Goal

- Define Analytics
  - Trends, Clusters, Outliers, Classification

Process

- Translate to Scalable Applications
  - Develop algorithms, Map to abstractions, Implement on Platforms

Data Platforms

- Acquire, manage, process Big Data
- At large scales
- To meet application needs

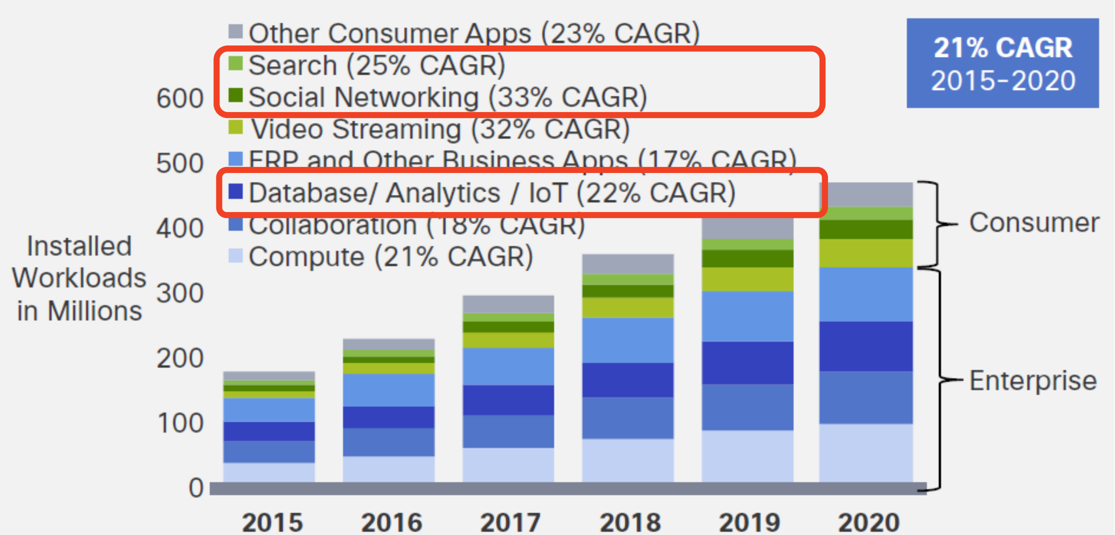
Distributed Systems

- Distributed Computing
  - Clusters of machines
  - Connected over network
- Distributed Storage
  - Disks attached to clusters of machines
  - Network Attached Storage
- *How can we make effective use of multiple machines?*

- **Commodity** clusters vs. **HPC** clusters
  - Commodity: Available off the shelf at large volumes
  - Lower Cost of Acquisition
  - Cost vs. Performance
    - Low disk bandwidth, and high network latency
    - CPU typically comparable (Xeon vs. i3/5/7)
    - Virtualization overhead on Cloud
- *How can we use many machines of modest capability?*

Growth of Cloud Data Centers

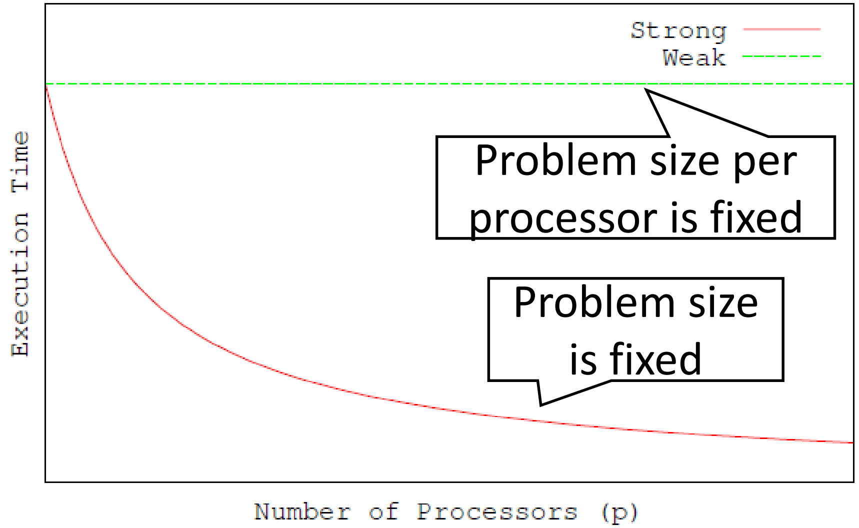
Figure 17. Global Data Center Workloads by Applications

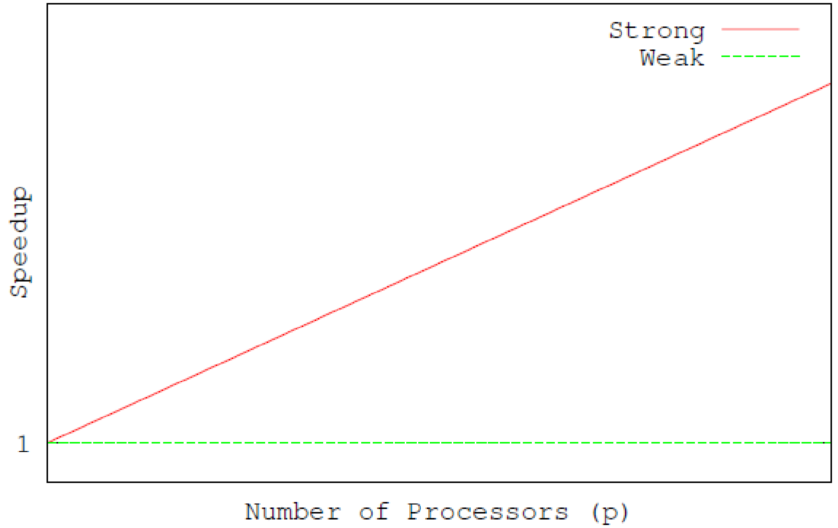


Source: Cisco Global Cloud Index, 2015-2020.

Cisco Global Cloud Index: Forecast and Methodology, 2015–2020, White Paper © 2016, Cisco

Ideal Strong/Weak Scaling





Scaling Theory and Machine Abstractions, Martha A. Kim, October 10, 2012

Scalability

- Strong vs. Weak Scaling
- **Strong Scaling**: How the performance varies with the # of processors for a *fixed total problem size*
- **Weak Scaling**: How the performance varies with the # of processors for a *fixed problem size per processor*
  - Big Data platforms are intended for “Weak Scaling”

Ease of Programming

- Programming distributed systems is difficult
  - Divide a job into multiple tasks
  - Understand dependencies between tasks: Control, Data
  - Coordinate and synchronize execution of tasks
  - Pass information between tasks
  - Avoid race conditions, deadlocks
- Parallel and distributed programming models/languages/abstractions/platforms try to make these easy
  - E.g. Assembly programming vs. C++ programming
  - E.g. C++ programming vs. Matlab programming

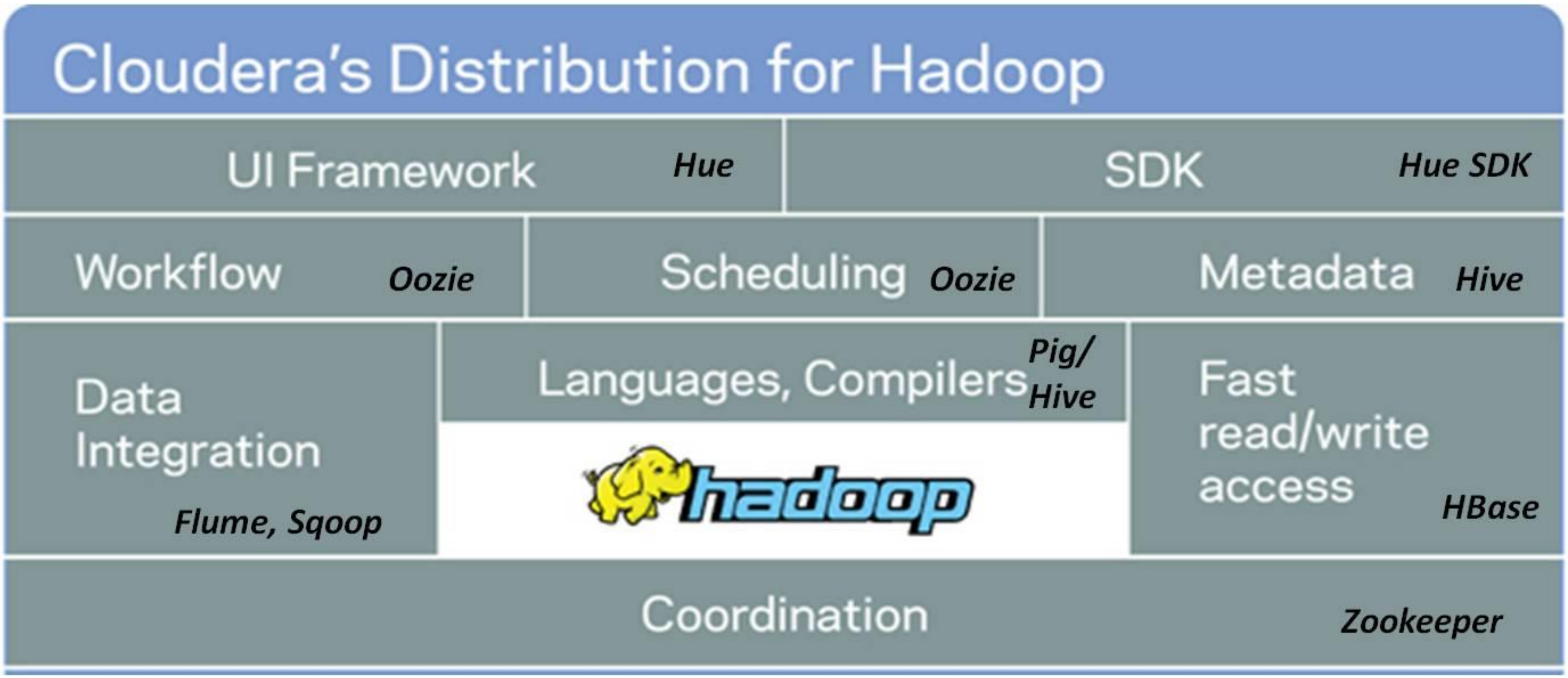
Availability, Failure

- Commodity clusters have lower reliability
  - Mass-produced
  - Cheaper materials
  - Smaller lifetime (~3 years)
- *How can applications easily deal with failures?*
- *How can we ensure availability in the presence of faults?*

Early Technologies

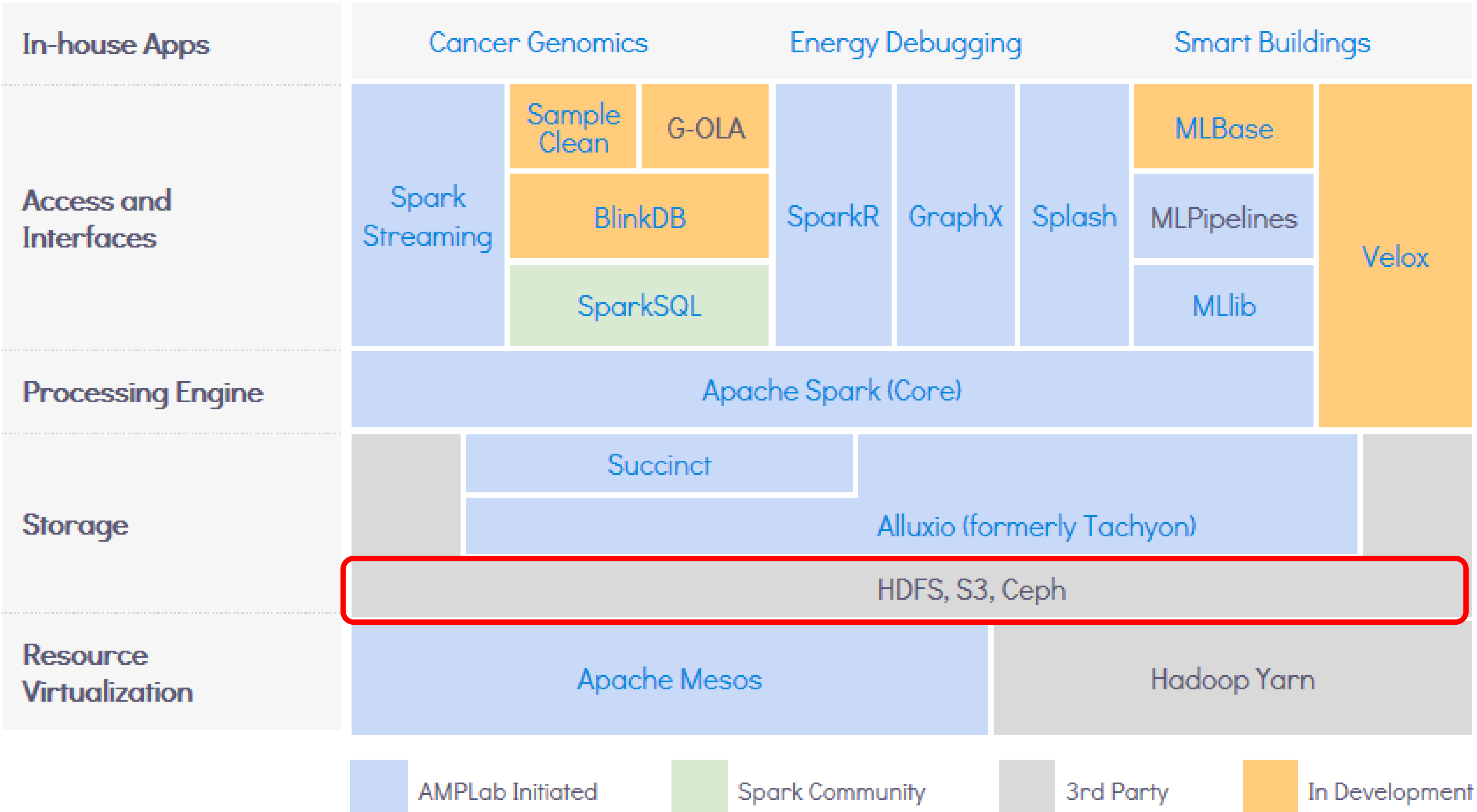
- **MapReduce** is a distributed data-parallel programming model from Google
- MapReduce works best with a distributed file system, called **Google File System (GFS)**
- **Hadoop** is the open source framework implementation from Apache that can execute the MapReduce programming model
- **Hadoop Distributed File System (HDFS)** is the open source implementation of the GFS design
- **Elastic MapReduce (EMR)** is Amazon's PaaS

Platforms…Think in terms of Stacks
Cloudera



Platforms…Think in terms of Stacks

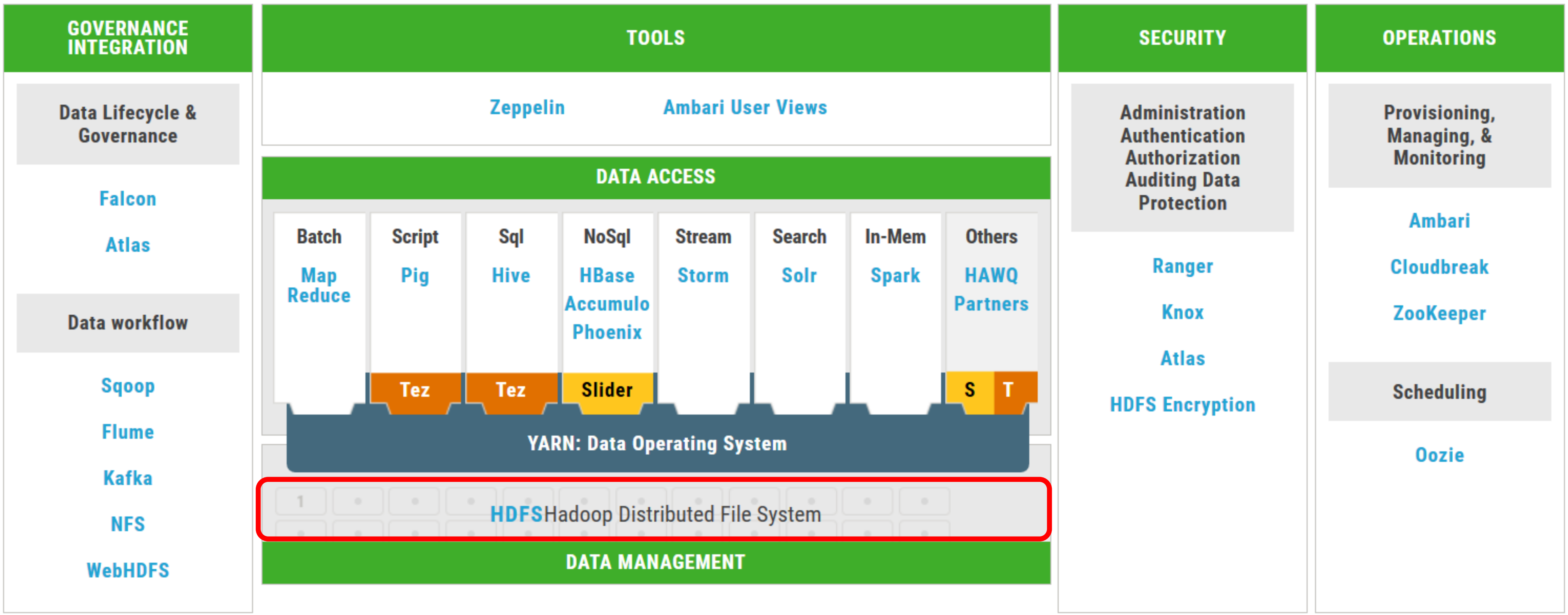
BDAS



https://amplab.cs.berkeley.edu/software/

Platforms…Think in terms of Stacks

HortonWorks



http://hortonworks.com/products/data-center/hdp/

Apache Spark

Slides & Additional Reading Courtesy

https://stanford.edu/~rezab/sparkclass/slides/itas_workshop.pdf

Resilient Distributed Datasets, Matei Zaharia

http://spark.apache.org/docs/2.1.1/programming-guide.html

http://spark.apache.org/docs/latest/api/java/index.html
https://www.gitbook.com/book/jaceklaskowski/mastering-apache-spark/details

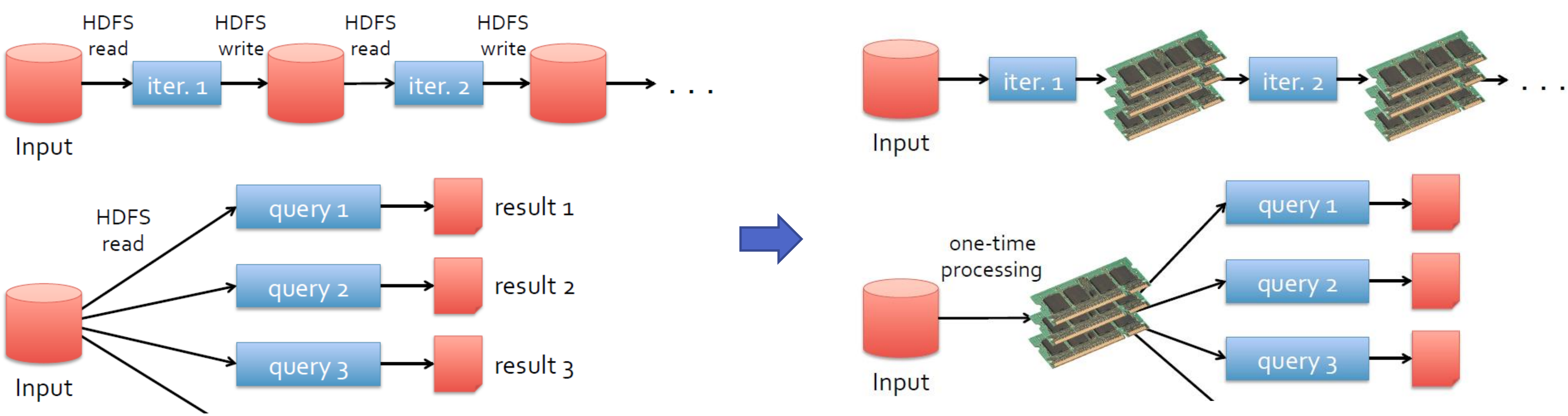
Apache Spark Internals, Pietro Michiardi, Eurecom

Why Spark?

- Ease of language definition
  - Typing, dataflows,
  - But Pig, Hive, HBase, etc. give you that
- Better performance using "In memory" compute
  - Multiple stages part of same job
  - Lazy evaluation, caching/persistence

In-memory computation

- Operate on data in (distributed) memory
  - Allows many operations to be performed locally
  - Write to disk only when data sharing required across workers
- This is unlike others like Hadoop Map/Reduce

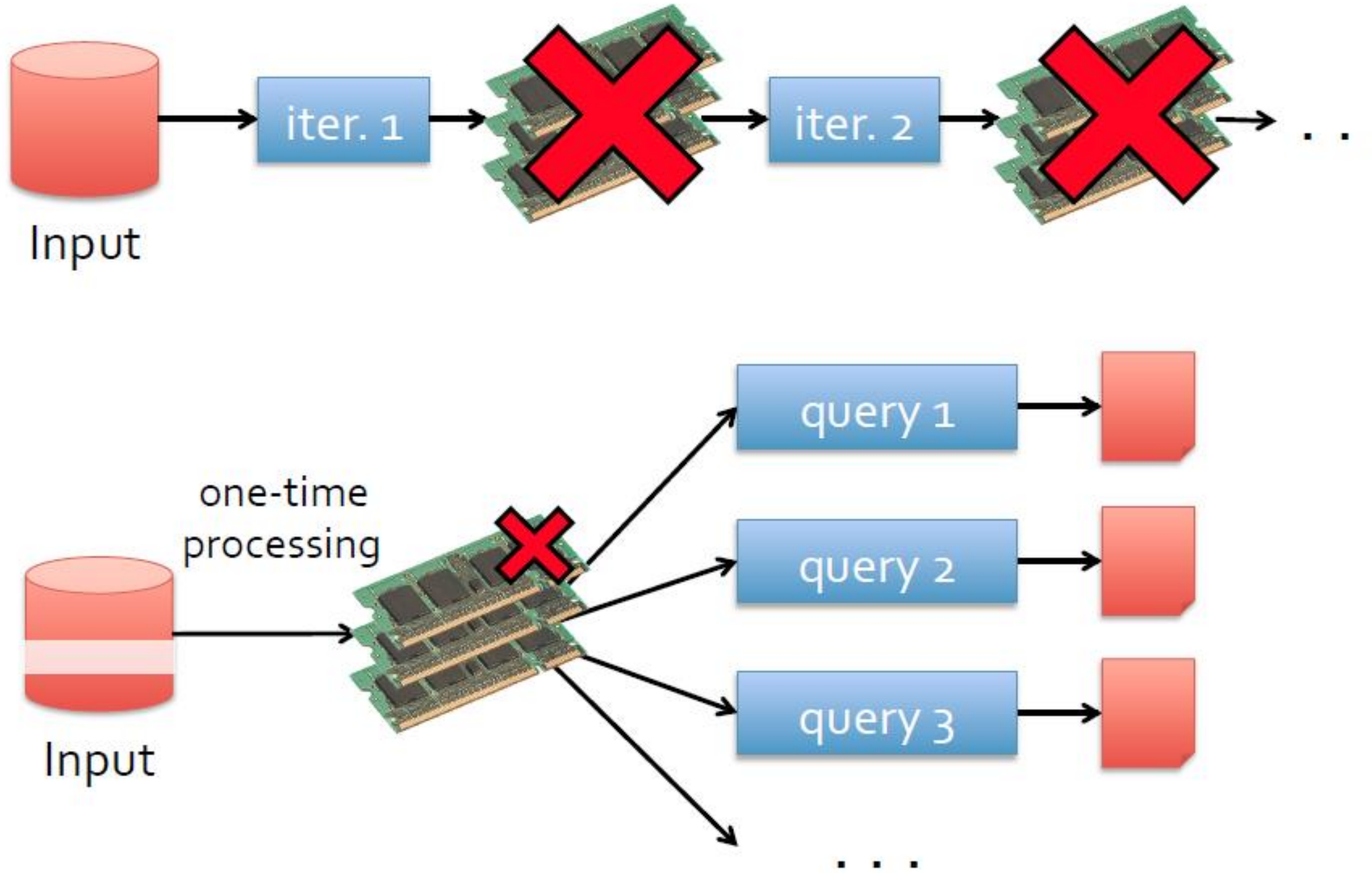


Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing, M. Zaharia, et al., *NSDI* 2012

RDD: The Secret Sauce

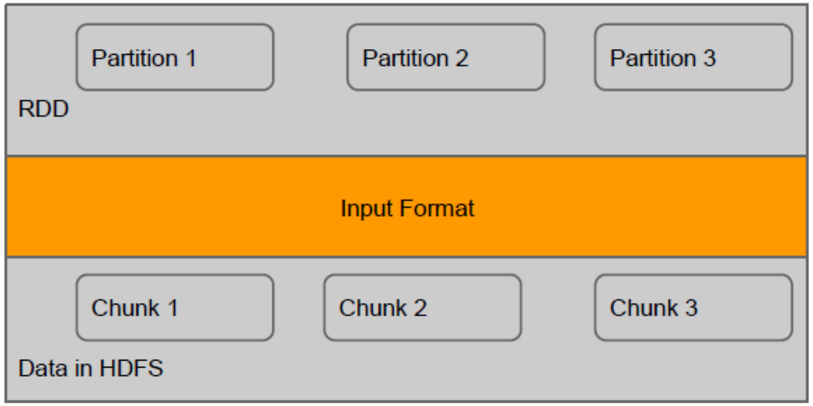
- RDD: Resilient Distributed Dataset
  - Immutable, partitioned collection of tuples
  - Operated on by *deterministic* transformations
    - Object-oriented flavor
    - RDD.operation() → RDD
- Recovery by re-computation
  - Maintains lineage of transformations
  - Recompute missing partitions *if failure happens*
  - Not possible/not automatic in Pig
- Allows caching & persistence for reuse

Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing, M. Zaharia, et al., *NSDI* 2012


Input
iter. 1
iter. 2
. . .
one-time
processing
Input
query 1
query 2
query 3
. . .

RDD Partitions

- RDD is internally a collection of partitions
  - Each partition holds a list of items
- Partitions may be present on a different machine
  - Partition is the *unit of execution*
  - Partition is the *unit of parallelism*
- They are immutable
  - Each transformation on an RDD generates a new RDD with different partitions
  - *Allows recovery of individual partitions*



RDD Operations

Allows composability into Dataflows

Transformations (define a new RDD)	map filter sample groupByKey reduceByKey sortByKey	flatMap union join cogroup cross mapValues
Actions (return a result to driver program)	collect reduce count save lookupKey	

https://grouplens.org/datasets/movielens/

A Sample Spark Program

- Movielens dataset, **movies.csv**
  - movieId,title,genres

```
m = sc.textFile("hdfs:///ml/movies.csv").cache()
[‘movieId,title,genres’]...
mcols = m.map(lambda l: l.split(",")).
mg = mcols.filter(lambda l: l[2] != 'genres’)
[‘92363’,‘Toy Story’,‘cartoon|action|children’]...
mgc = mg.map(lambda l: (len(l[2].split("|")), l))
[3,[‘92363’,‘Toy Story’,‘cartoon|action|children’]]...
maxgc = mgc.max()[0]
3
maxgcm = mgc.lookup(maxgc)
[3,[‘92363’,‘Toy Story’,‘cartoon|action|children’]]...
```

What is the average number of ratings given by users? What is the average value of the ratings given by users?

```
m = sc.textFile("hdfs:///user/ml/movies.csv").cache()
r = sc.textFile("hdfs:///user/ml/ratings.csv").cache()

rv = r.map(lambda l : l.split(",")[2]).filter(lambda l : l != 'rating')
rvs = rv.reduce(lambda a, b: float(a) + float(b))   # sum of ratings
rvc = rv.count()   # ratings count
print 'Avg rating value is', rvs/rvc

rc = r.count() - 1  # number of ratings
rud = r.map(lambda l : l.split(",")[0]).distinct()
ruc = (rud.count()-1)     # number of distinct users
print 'Avg ratings per user is', rc/ruc
```

For movies with more than 1 genre, what are the most and least likely pair of genres to occur together?

```
me = m.map(lambda l : l if l.find("\"") == -1 else l.partition("\"")[0] +
l[l.find("\"")+1:l.rfind("\"")-1].replace(",", ";") +
l.rpartition("\"")[2])

mg = me.map(lambda l:l.split(",")).filter(lambda l : l[2] != 'genres')
mgf = mg.flatMap(lambda l : zip([l[0]]*len(l[2].split("|")),
l[2].split("|")))

mgj = mgf.join(mgf).filter(lambda (m,g) : g[0] != g[1])
mgpc = mgj.map(lambda (m,g) : ('+'.join(sorted(g)),1))
msgp = mgpc.reduceByKey(lambda a, b: a + b).map(lambda (gp,s) : (s,gp))
gpmax = msgp.max()
gpmin = msgp.min()

print 'Genres pairs most likely to occur are',gpmax[1],'with a
freq',gpmax[0]
print 'Genres pairs least likely to occur are',gpmin[1],'with a
freq',gpmin[0]
```

Creating RDD

- Load external data from distributed storage
- Create logical RDD on which you can operate
- Support for different input formats
  - HDFS files, Cassandra, Java serialized, directory, gzipped
- Can control the number of partitions in loaded RDD
  - Default depends on external DFS, e.g. 128MB on HDFS

```
m = sc.textFile("hdfs:///ml/movies.csv").cache()
```

RDD Operations

- **Transformations**
  - From one RDD to one or more RDDs
  - Lazy evaluation upon "action"…*use with care*
  - Executed in a distributed manner

- **Actions**
  - Perform aggregations on RDD items
  - Return single (or distributed) results to "driver" code
  - RDD.**collect**() brings RDD partitions to single driver machine

RDD and PairRDD

- RDD is logically a collection of items with a generic type
- PairRDD is a 2-tuple, like a "Map", where each item in the collection is a <key,value> pair
  - But can have *duplicate keys*
- Transformation functions use RDD or PairRDD as input/output

Transformations

Transformation	Meaning
map(*func*)	Return a new distributed dataset formed by passing each element of the source through a function *func*.
filter(*func*)	Return a new dataset formed by selecting those elements of the source on which *func* returns true.
flatMap(*func*) Implicit in PySpark	Similar to map, but each input item can be mapped to 0 or more output items (so *func* should return a Seq rather than a single item).
sample(*withReplacement*, *fraction*, *seed*)	Sample a fraction *fraction* of the data, with or without replacement, using a given random number generator seed.
union(*otherDataset*)	Return a new dataset that contains the union of the elements in the source dataset and the argument.
intersection(*otherDataset*) Also removes duplicates	Return a new RDD that contains the intersection of elements in the source dataset and the argument.
distinct([*numTasks*]))	Return a new dataset that contains the distinct elements of the source dataset.

https://spark.apache.org/docs/latest/api/python/pyspark.html#pyspark.RDD

Transformations on PairRDD

aggregateByKey(*zeroValue*)(*seqOp*, *combOp*, [*numTasks*])	When called on a dataset of (K, V) pairs, returns a dataset of (K, U) pairs where the values for each key are aggregated using the given combine functions and a neutral "zero" value. Allows an aggregated value type that is different than the input value type, while avoiding unnecessary allocations. Like in `groupByKey`, the number of reduce tasks is configurable through an optional second argument.
join(*otherDataset*, [*numTasks*])	When called on datasets of type (K, V) and (K, W), returns a dataset of (K, (V, W)) pairs with all pairs of elements for each key. Outer joins are supported through `leftOuterJoin`, `rightOuterJoin`, and `fullOuterJoin`.

Aggregation: Average number of ratings given by users

```
[userId,movieId,rating,timestamp]
rv = r.map(lambda l: l.split(",")[2])
rfv = rv.filter(lambda l:
          l != 'rating’)
[rating]...
rvs = rfv.reduce(lambda a, b:
          float(a) + float(b))
rvc = rfv.count()
print rvs/rvc
```

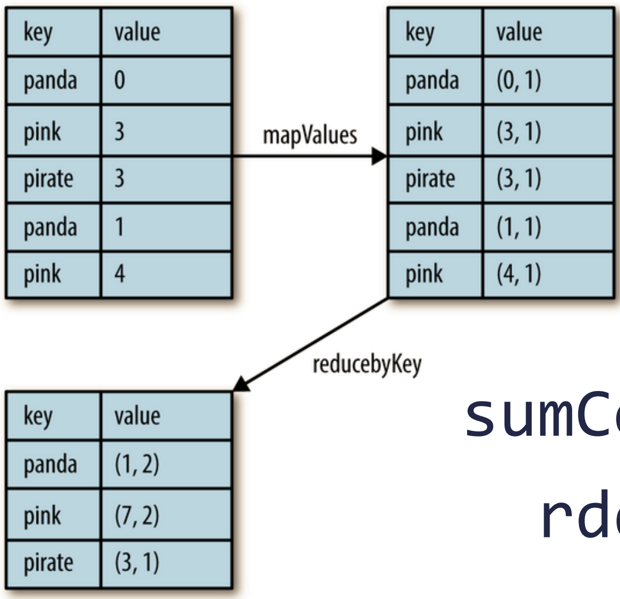
Action

Action

Actions

reduce(*func*)	Aggregate the elements of the dataset using a function *func* (which takes two arguments and returns one). The function should be commutative and associative so that it can be computed correctly in parallel.
collect()	Return all the elements of the dataset as an array at the driver program. This is usually useful after a filter or other operation that returns a sufficiently small subset of the data.
count()	Return the number of elements in the dataset.
countByKey()	Only available on RDDs of type (K, V). Returns a hashmap of (K, Int) pairs with the count of each key.
first()	Return the first element of the dataset (similar to take(1)).
take(*n*)	Return an array with the first *n* elements of the dataset.
takeSample(*withReplacement*, *num*, [*seed*])	Return an array with a random sample of *num* elements of the dataset, with or without replacement, optionally pre-specifying a random number generator seed.

Samples: Per-key average




```
sumCount =
   rdd.mapValues(x -> (x,1)).
     reduceByKey((x, y) ->
       (x[0]+y[0], x[1]+y[1]))
```

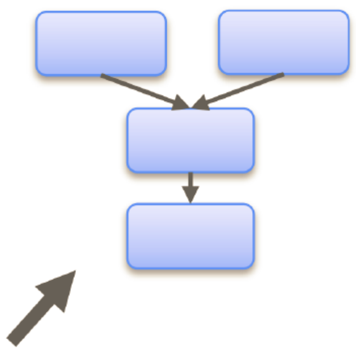

https://www.safaribooksonline.com/library/view/learning-spark/9781449359034/ch04.html

RDD Persistence & Caching

- RDDs can be reused in a dataflow
  - Branch, iteration
- But it will be re-evaluated each time it is reused!
- *Explicitly persist* RDD to reuse output of a dataflow path multiple times
- Multiple storage levels for persistence
  - Disk or memory
  - Serialized or object form in memory
  - Partial spill-to-disk possible
  - *Cache* indicates "persist" to memory

Distributed Execution

RDD Objects

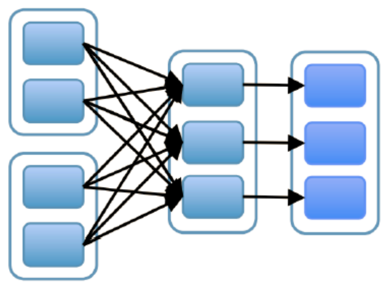


```
rdd1.join(rdd2)
    .groupBy(…)
    .filter(…)
    .count()
```

build operator DAG

DAG

Scheduler (DAGScheduler)



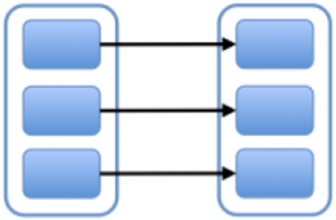
split graph into *stages* of tasks

submit each stage as ready

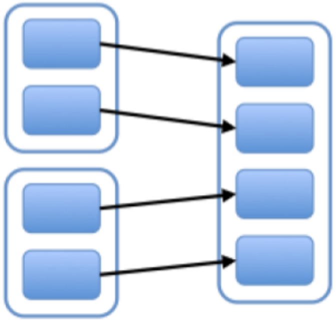
Execution Dependency

NARROW DEPENDENCY: Each partition of the parent RDD is used by at most one partition of the child RDD. Task can be executed locally and we don't have to shuffle.

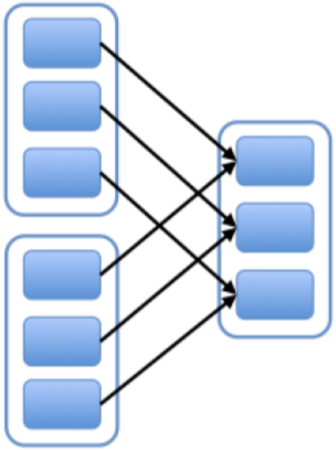
WIDE DEPENDENCY: Multiple child partitions may depend on one partition of the parent RDD. We have to shuffle data *unless the parents are hash-partitioned*



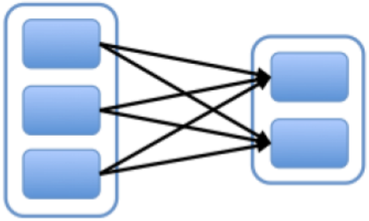
map, filter



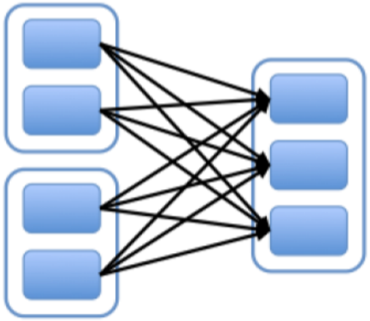
union



join with co-partitioned inputs

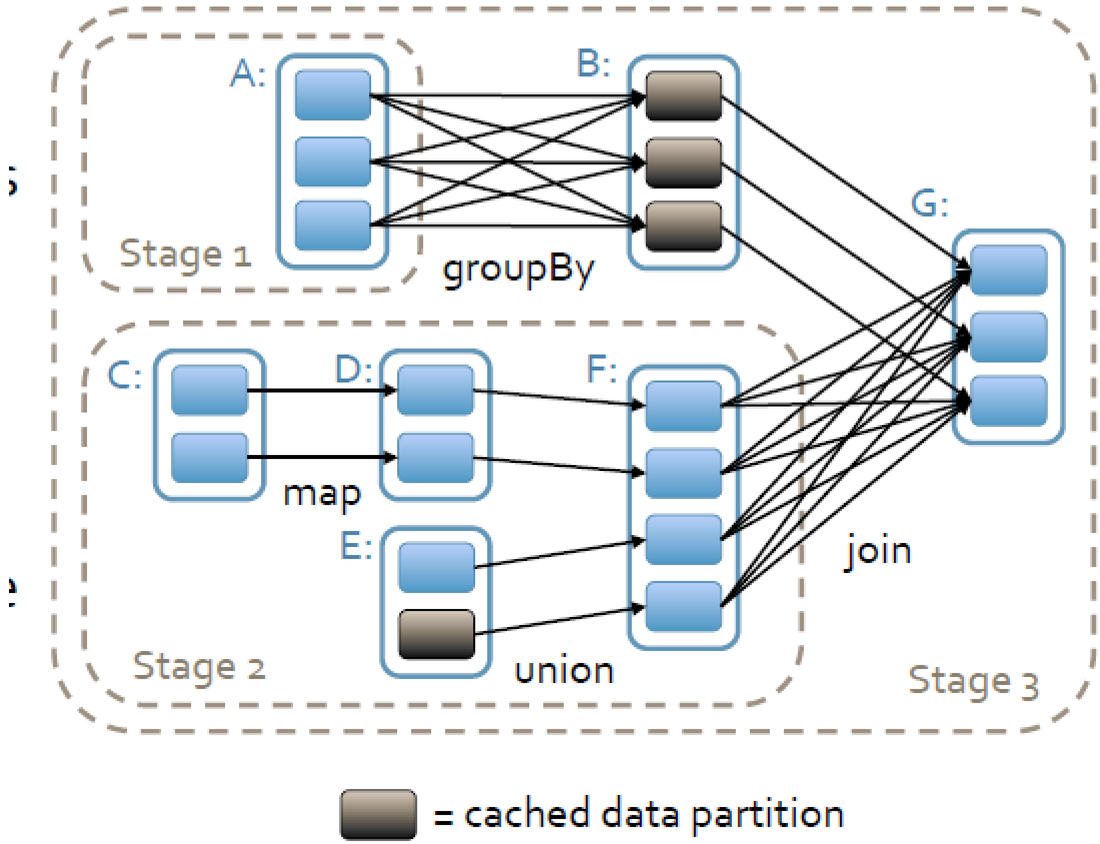


groupByKey

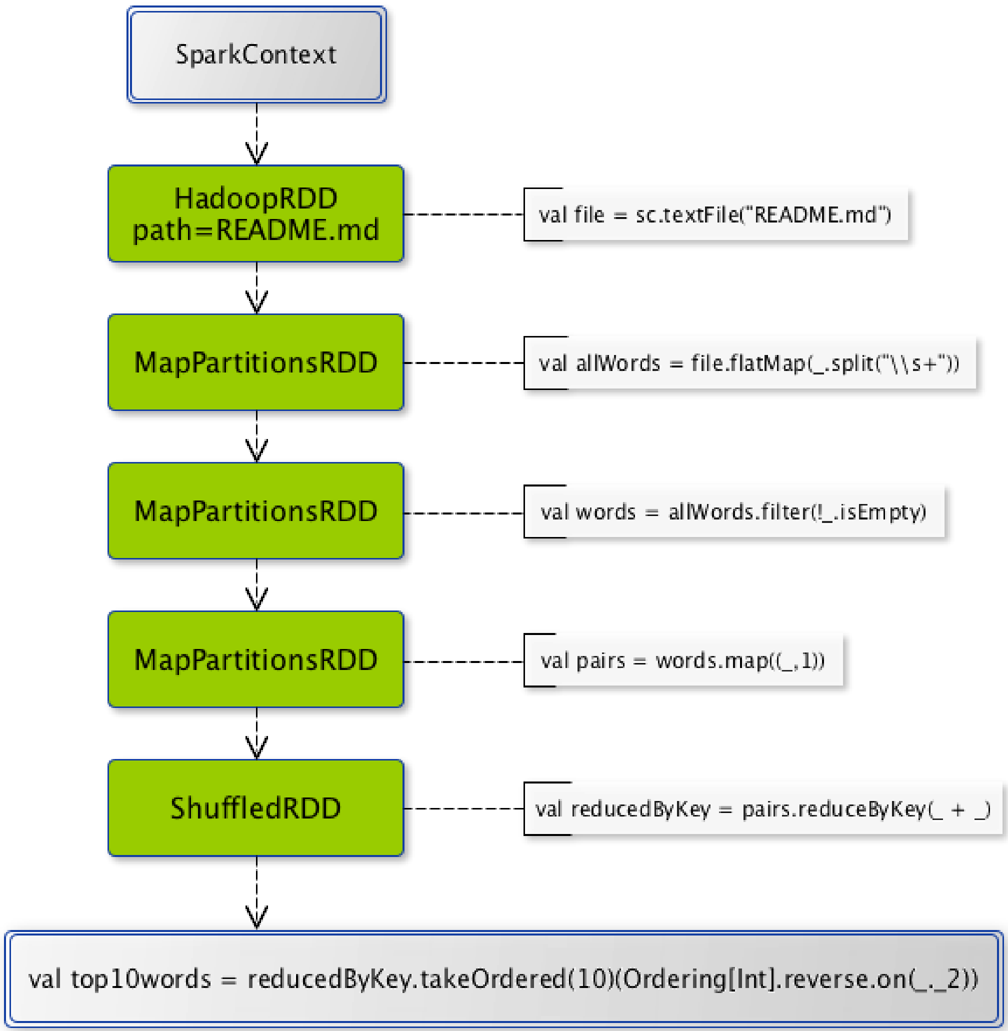


join with inputs not co-partitioned

Lazy Execution



From DAG to RDD lineage



https://jaceklaskowski.gitbooks.io/mastering-apache-spark/spark-rdd-transformations.html