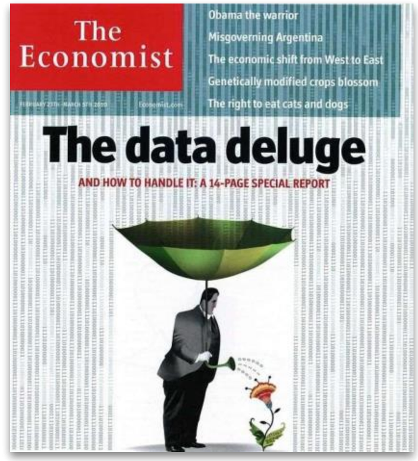
Big Data Analytics

What's the BIG deal?!

2008

2011

2010

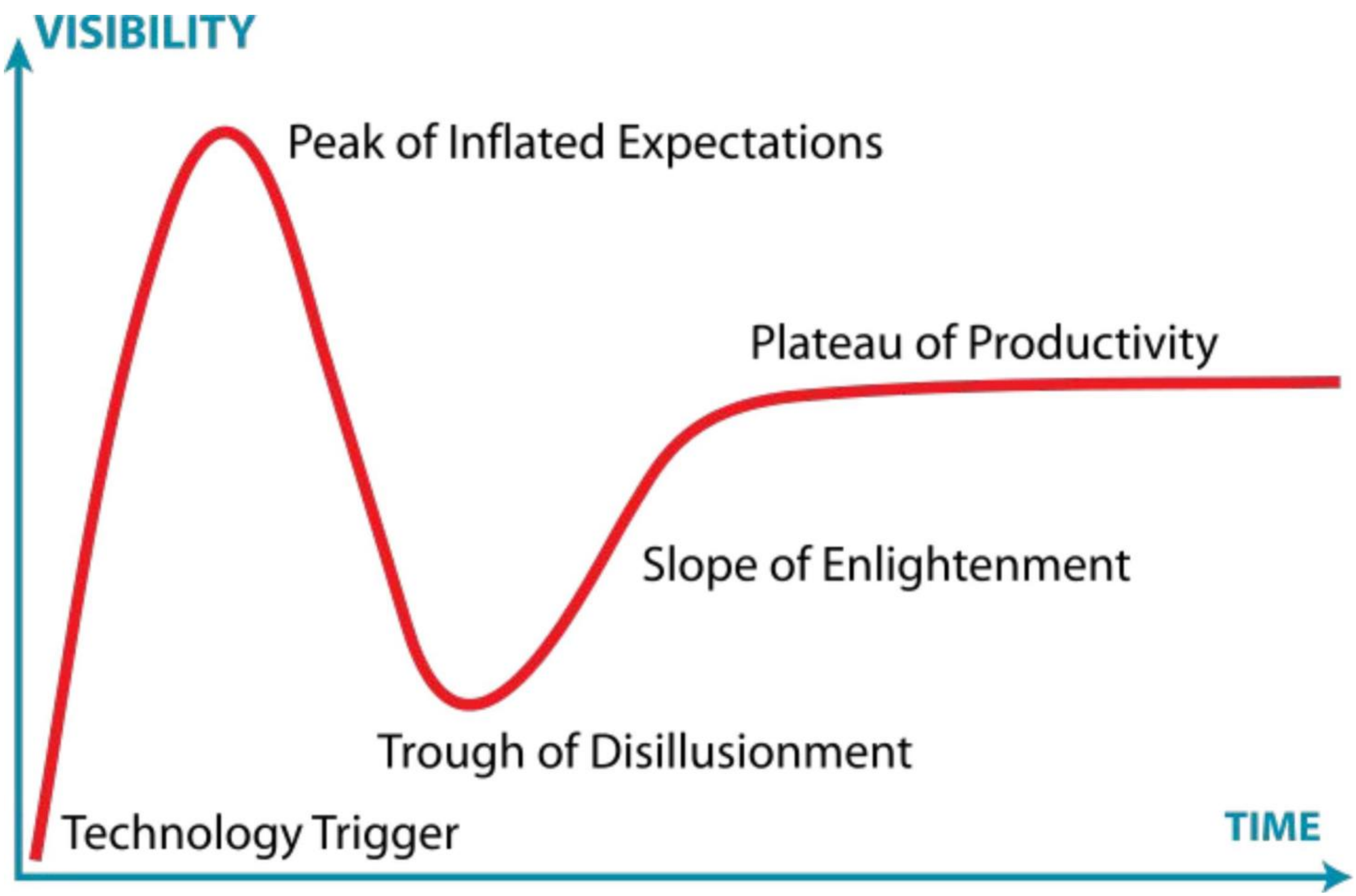


2011

2012

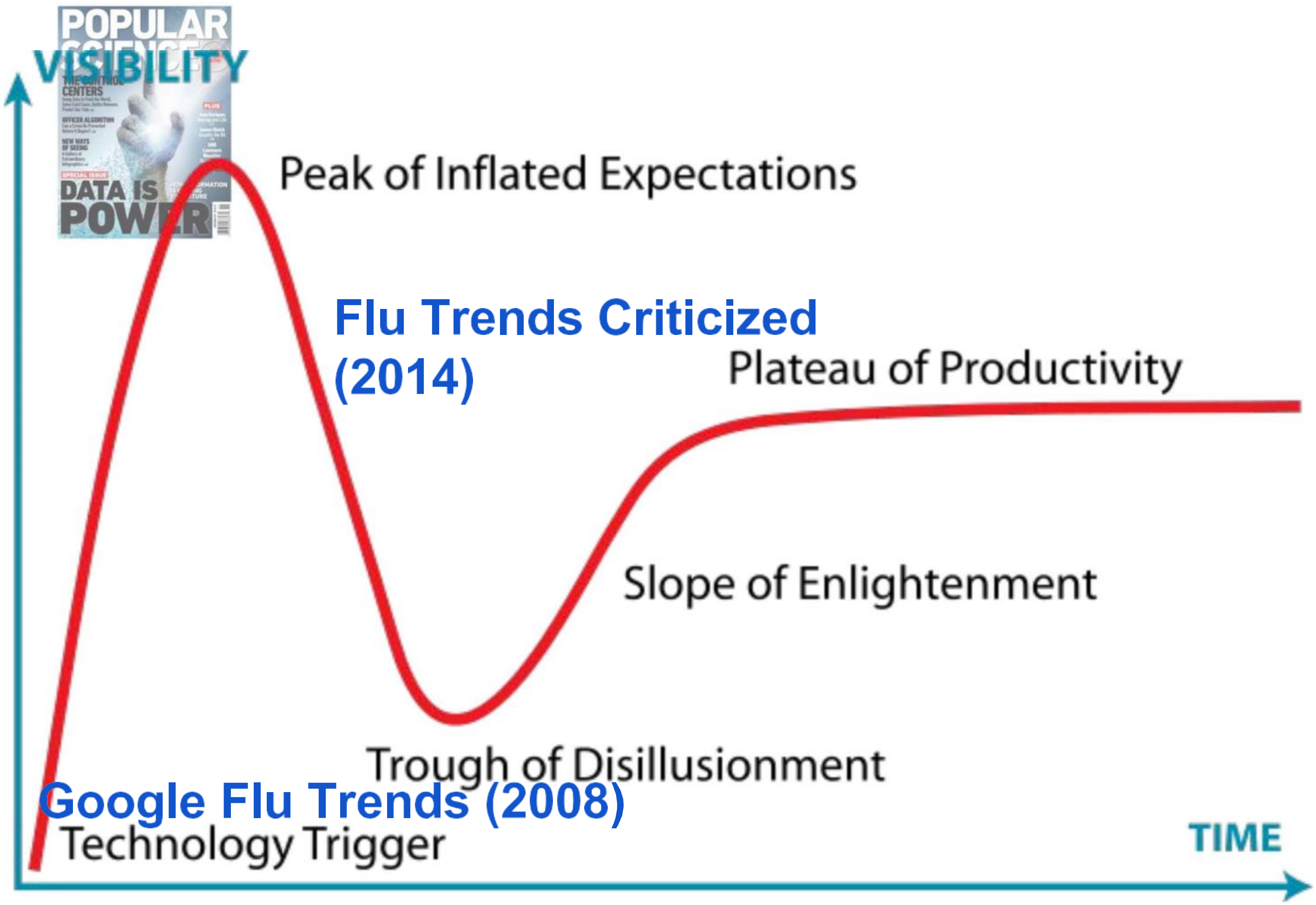
2018

What's the BIG deal?!



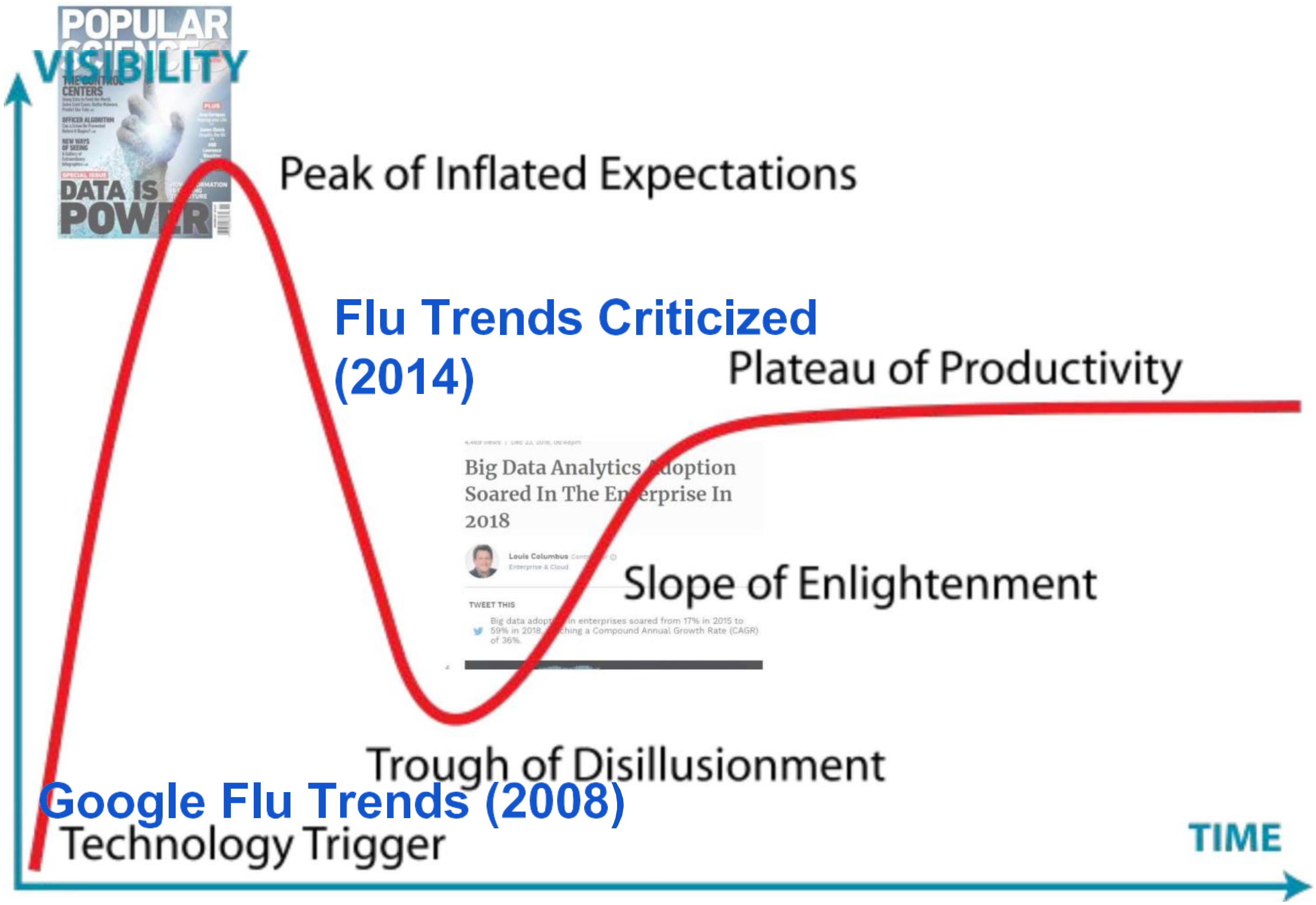
(Gartner Hype Cycle)

What’s the BIG deal?!



(Gartner Hype Cycle)

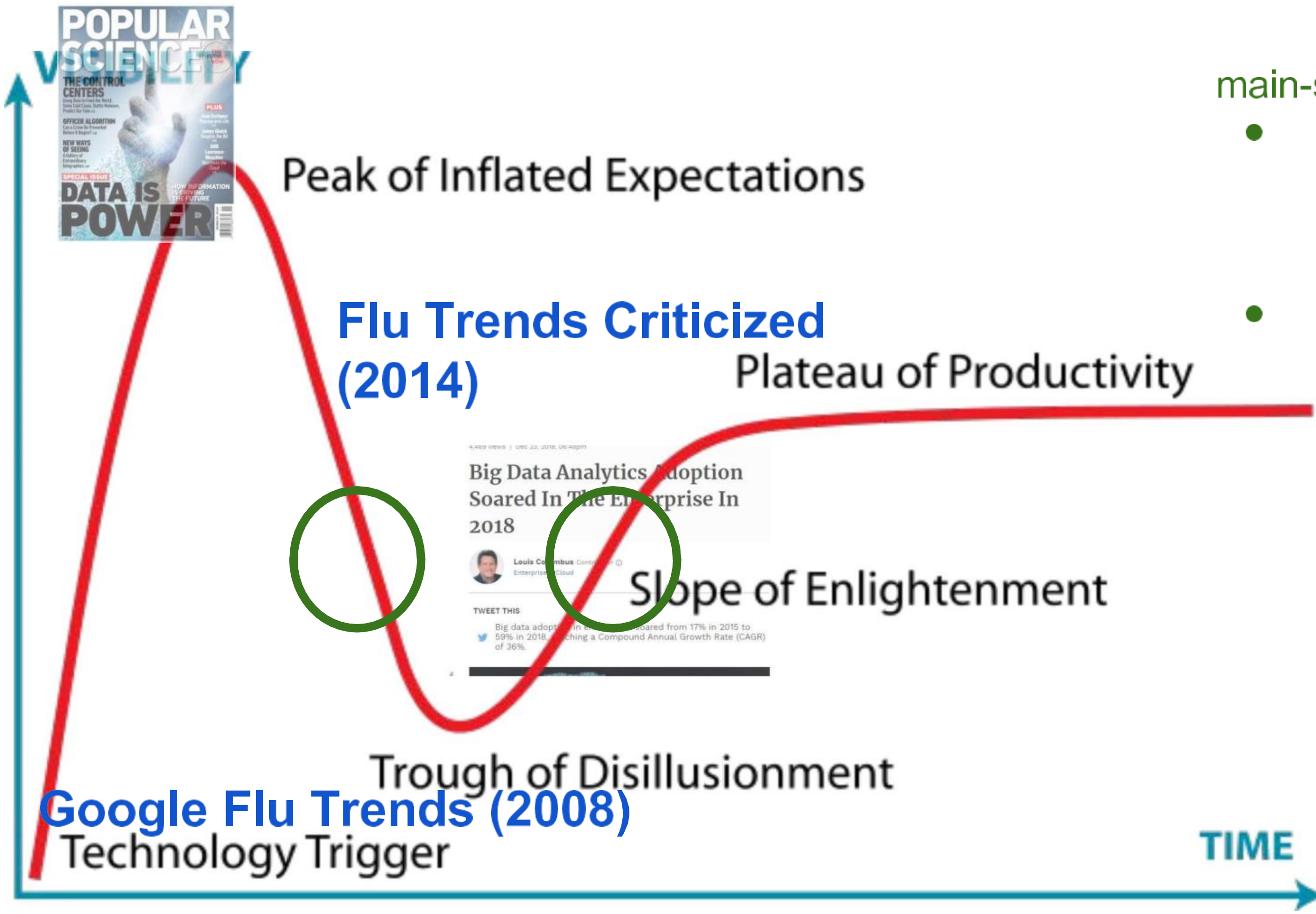
What's the BIG deal?!



(Gartner Hype Cycle)

Where are we today?

What’s the BIG deal?!

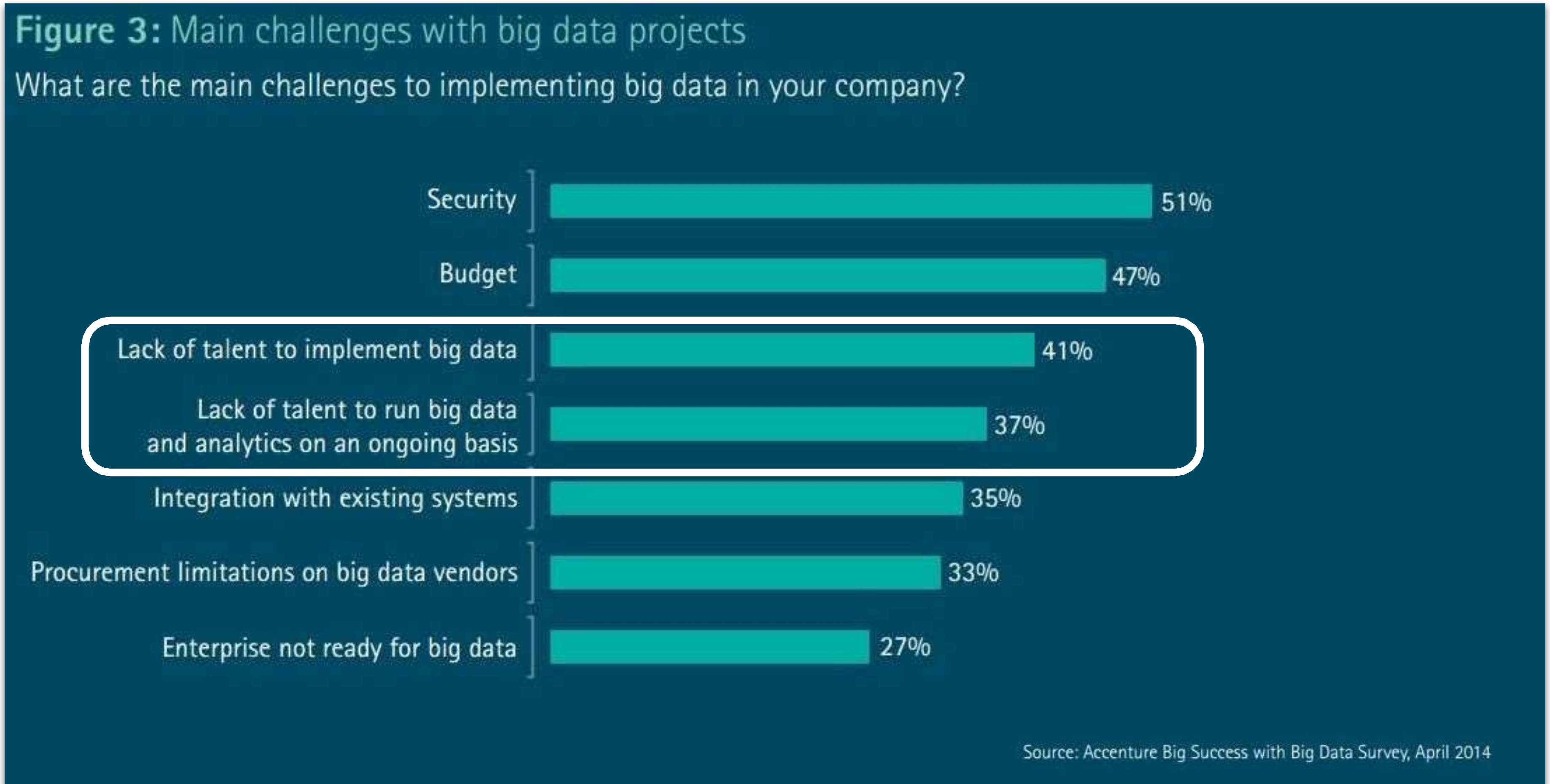


main-stream study being established

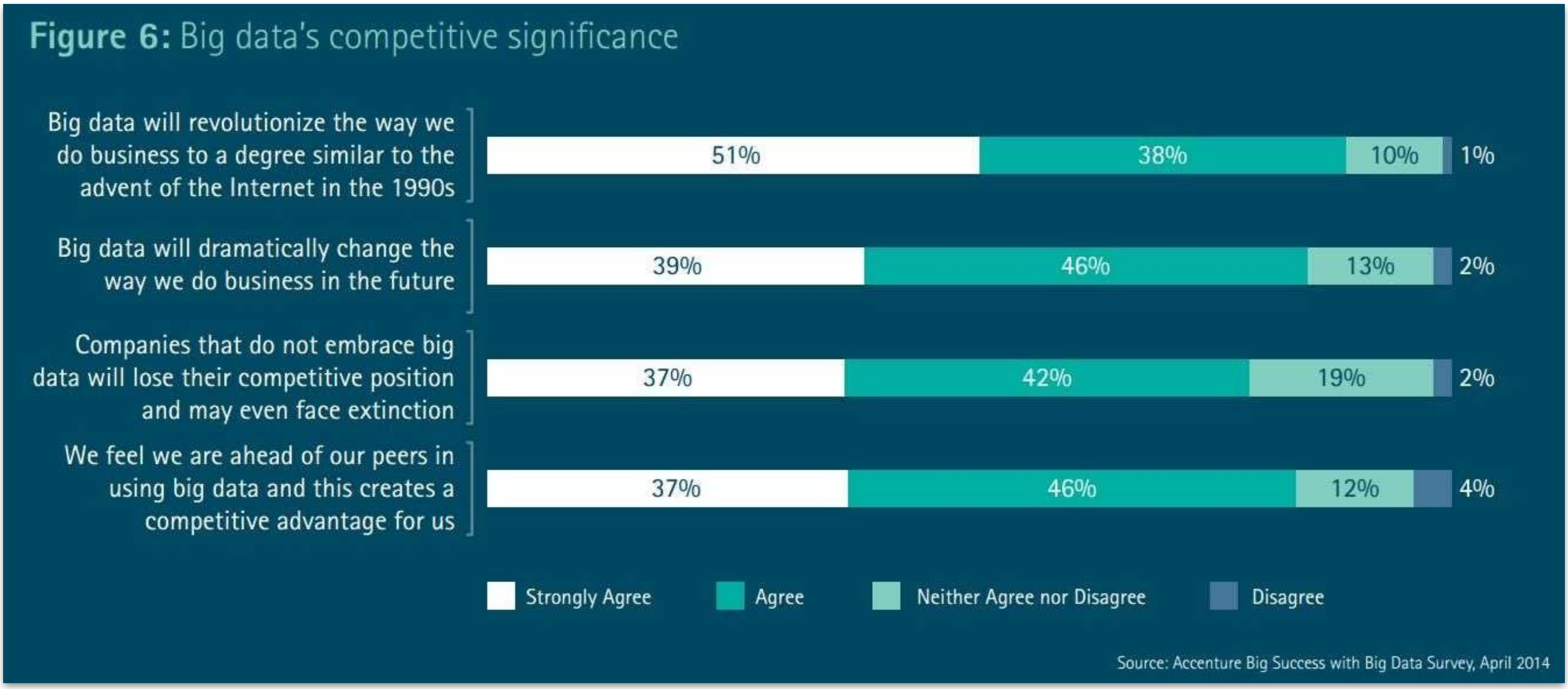
- Realization of what subfields are really doing “big data” (i.e. data mining, ML, Statistics, computational social sciences).
- Best practices being established.

(Gartner Hype Cycle)

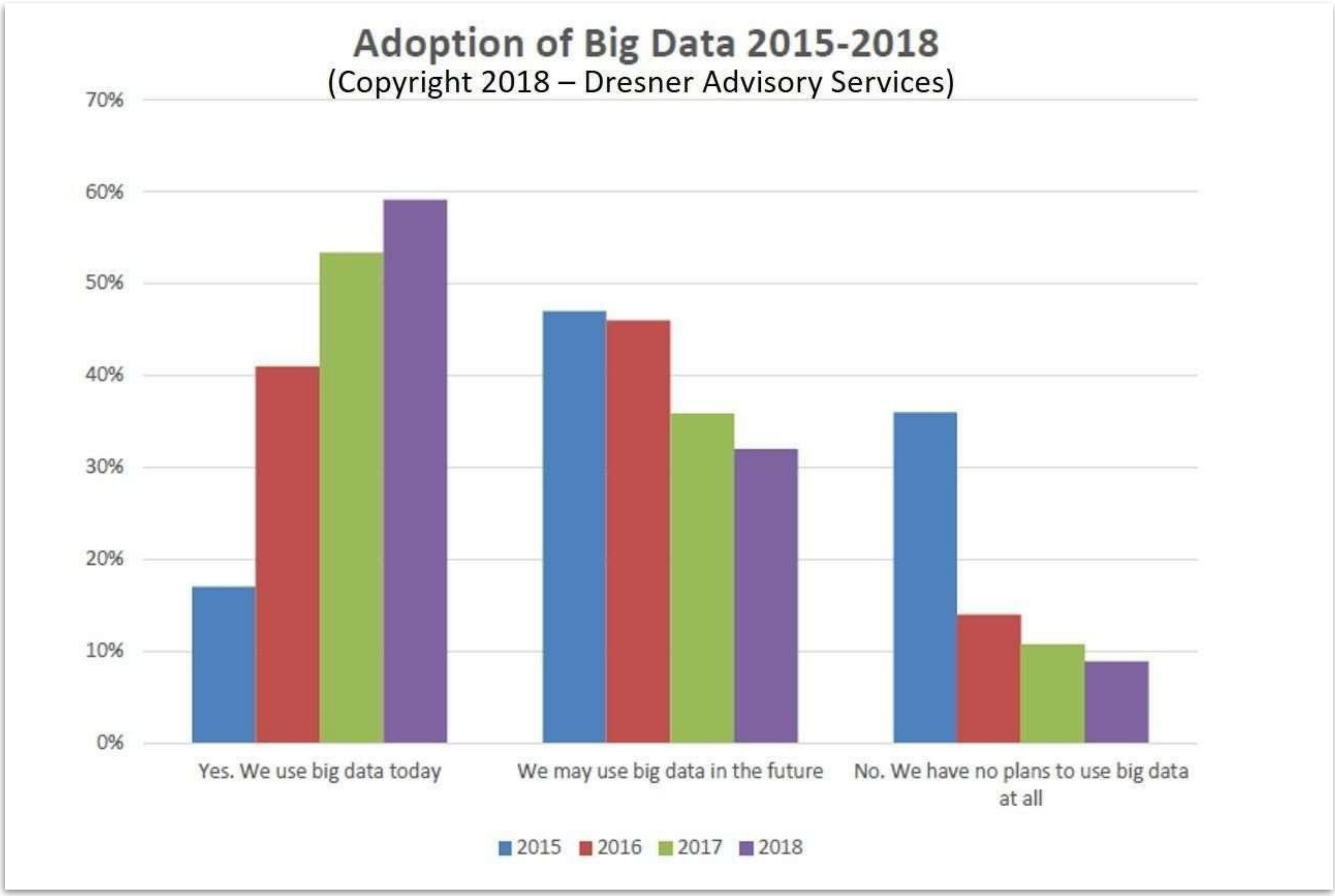
What’s the BIG deal?!



What's the BIG deal?!

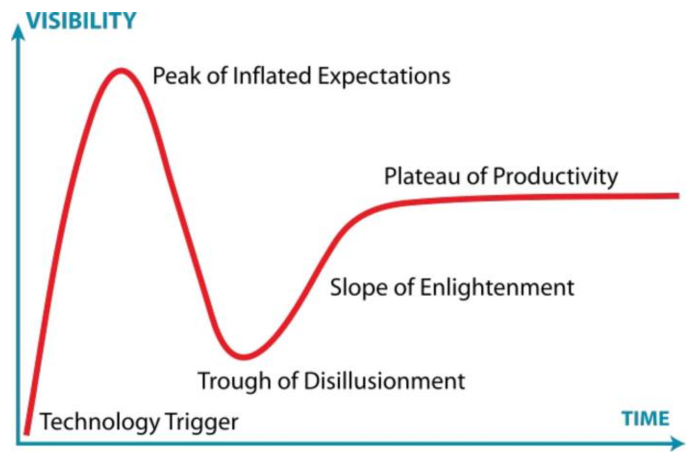


What's the BIG deal?!



https://www.forbes.com/sites/louiscolumbus/2018/12/23/big-data-analytics-adoption-soared-in-the-enterprise-in-2018/

What's the BIG deal?!

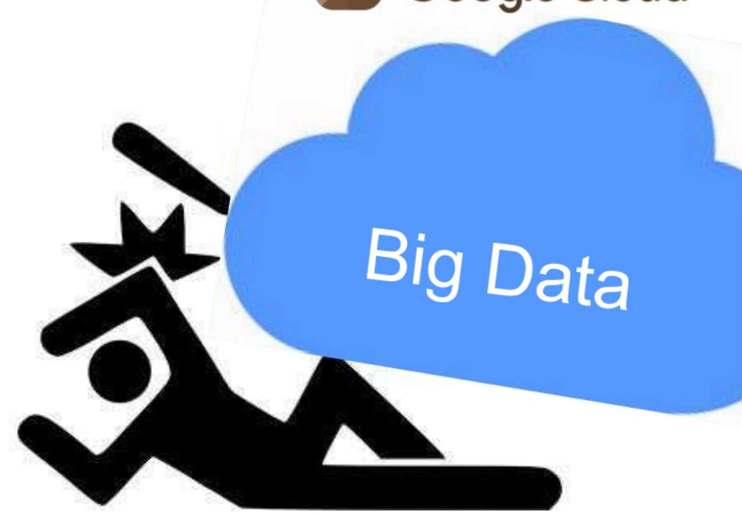


Reasons to be skeptical

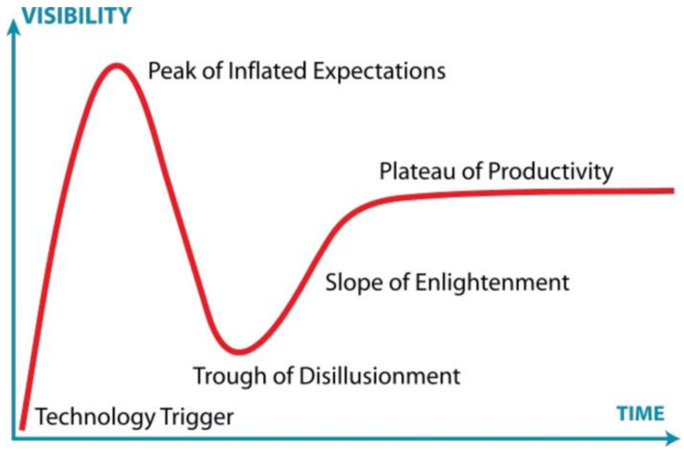
- Hype machine

- Downside of many tools:
  - Creates obfuscation: encourages seeing as magic black boxes
  - Less "standards": difficult to translate between, understand results

- Downside of large amounts of data:
  - Harder to "view"
  - Training takes longer
  - More prone to errors: rounding; heterogeneity



What’s the BIG deal?!



amazon web services™

Microsoft Azure

Google Cloud

Reasons to be skeptical

- Hype machine
- Downside of many tools:
  - Creates obfuscation: encourages seeing as magic black boxes
  - Less “standards”: difficult to translate between, understand results
- Downside of large amounts of data:
  - Harder to “view”
  - Training takes longer
  - More prone to errors: rounding; heterogeneity

Combat with:

- **Understanding *how* it works (theory)**
- ***When/where it works* (applied; experience)**

What is Big Data?

What is Big Data?

traditional
computer science

data that will not fit
in main memory.

What is Big Data?

traditional
computer science

data that will not fit
in main memory.

data with a *large*
number of observations
and/or features.

statistics

What is Big Data?

traditional
computer science

data that will not fit
in main memory.

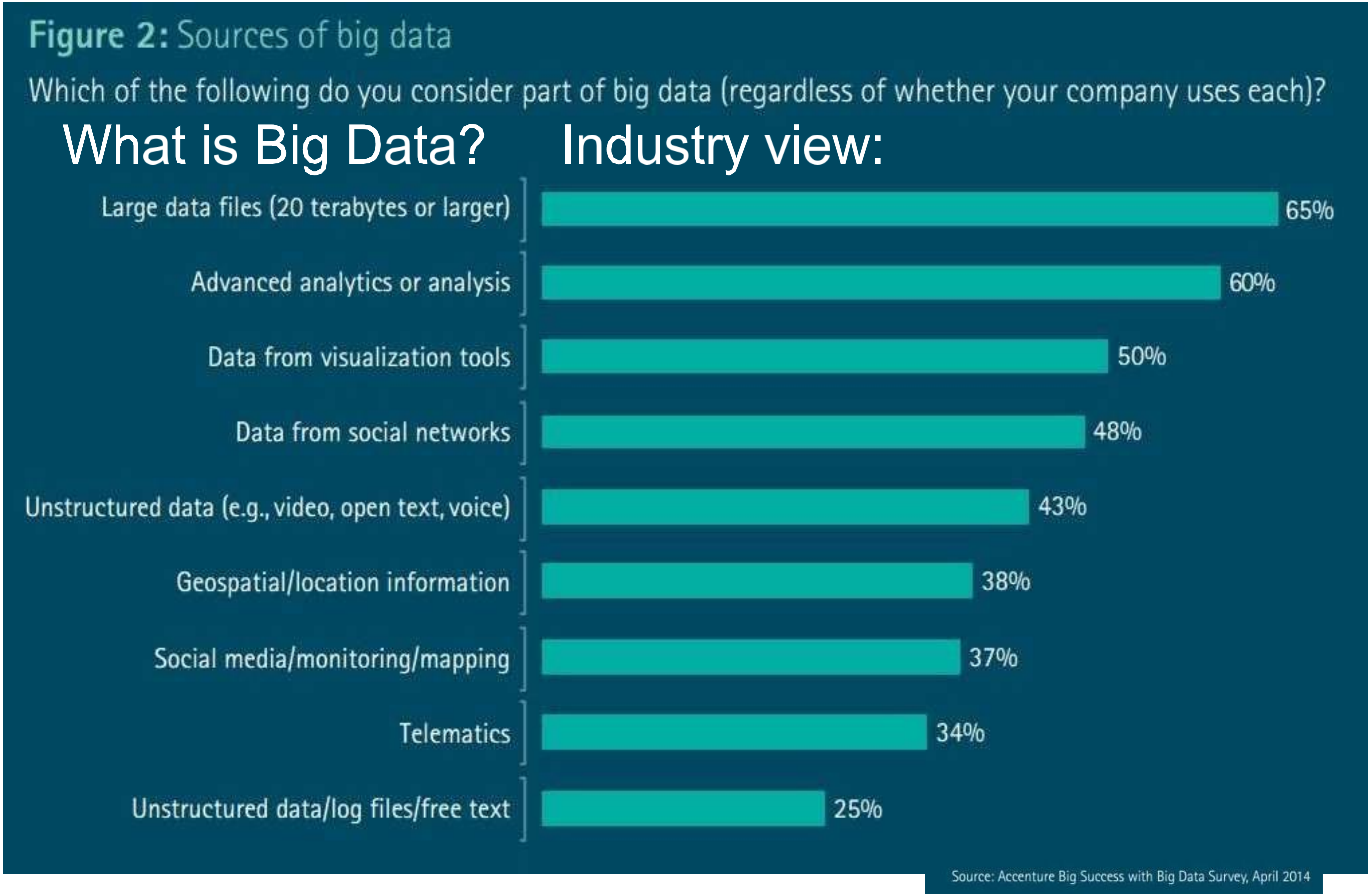
data with a *large*
number of observations
and/or features.

statistics

other fields

non-traditional sample size
(i.e. > 100 subjects); can't
analyze in stats tools (Excel).

What is Big Data? Industry view:



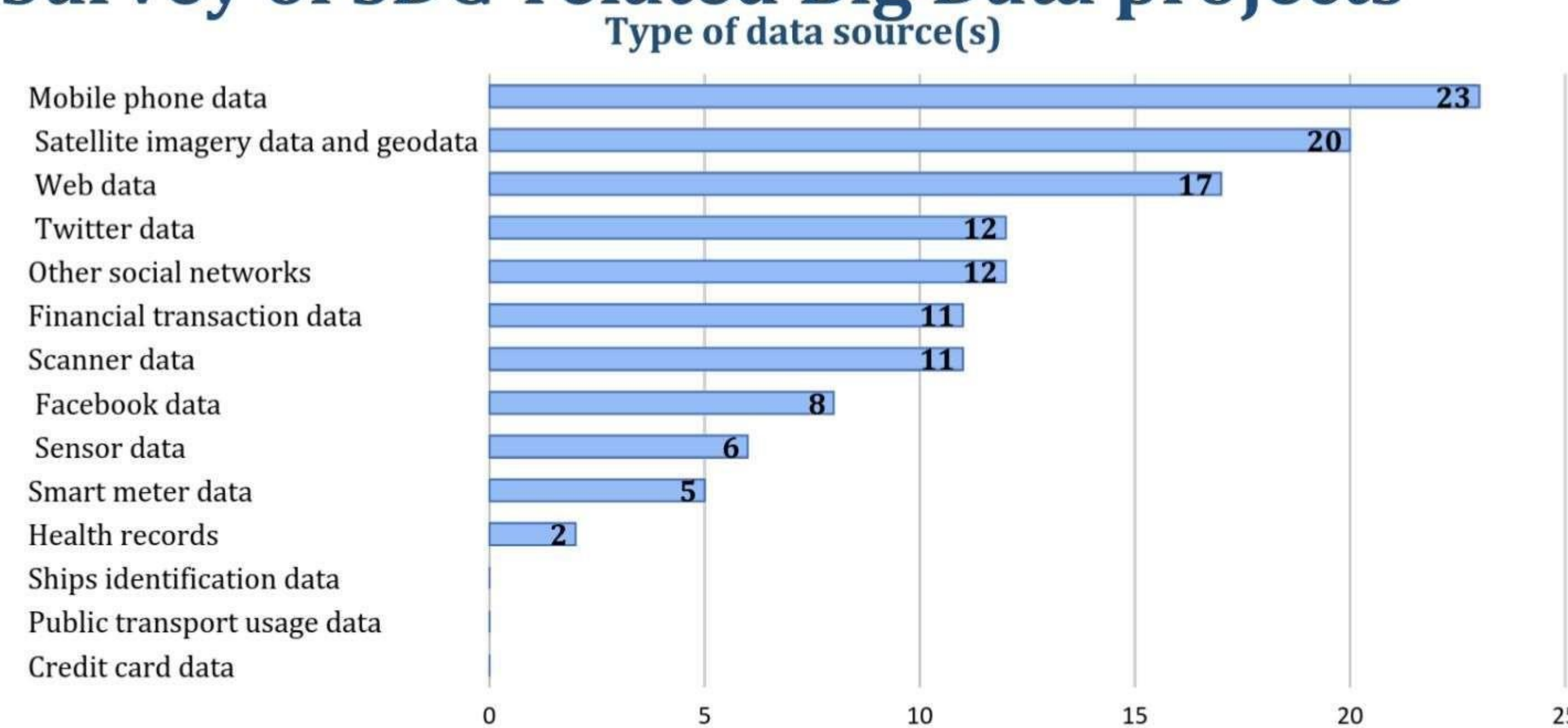
What is Big Data? Government view:

THE WORLD BANK
IBRD • IDA | WORLD BANK GROUP

(2016)

BigData
UN Global Working Group

1. Survey of SDG-related Big Data projects



- Mobile (23), Satellite imagery (20) and social media (12+12+8) are the most prominent sources

What is Big Data?

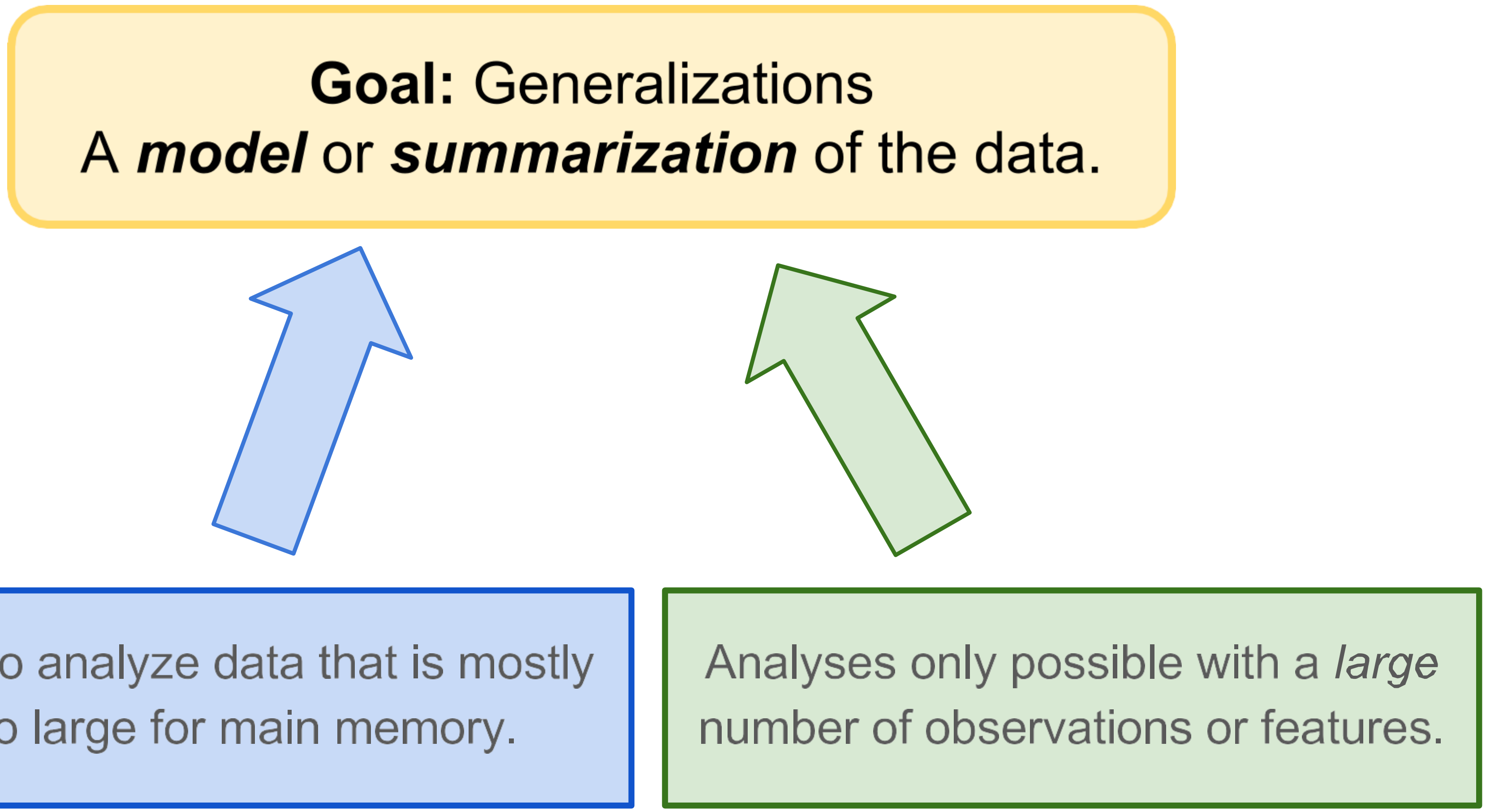
Short Answer:

Big Data ≈ Data Mining ≈ Predictive Analytics ≈ Data Science (Leskovec et al., 2014)

This Class:

How to analyze data that is mostly too large for main memory.	Analyses only possible with a *large* number of observations or features.

What is Big Data?



What is Big Data?

Goal: Generalizations
A *model* or *summarization* of the data.

E.g.

- Google's PageRank: *summarizes* web pages by a single number.
- Twitter financial market predictions: *Models* the stock market according to shifts in sentiment in Twitter.
- Distinguish tissue type in medical images: *Summarizes* millions of pixels into clusters.
- Mental health diagnosis in social media: *Models* presence of diagnosis as a distribution (a summary) of linguistic patterns.
- Frequent co-occurring purchases: *Summarize* billions of purchases as items that frequently are bought together.

What is Big Data?

Goal: Generalizations
A *model* or *summarization* of the data.

1. Descriptive analytics
   Describe (*generalizes*) the data itself

2. Predictive analytics
   Create something *generalizeable* to new data

Preliminaries

Ideas and methods that will repeatedly appear:

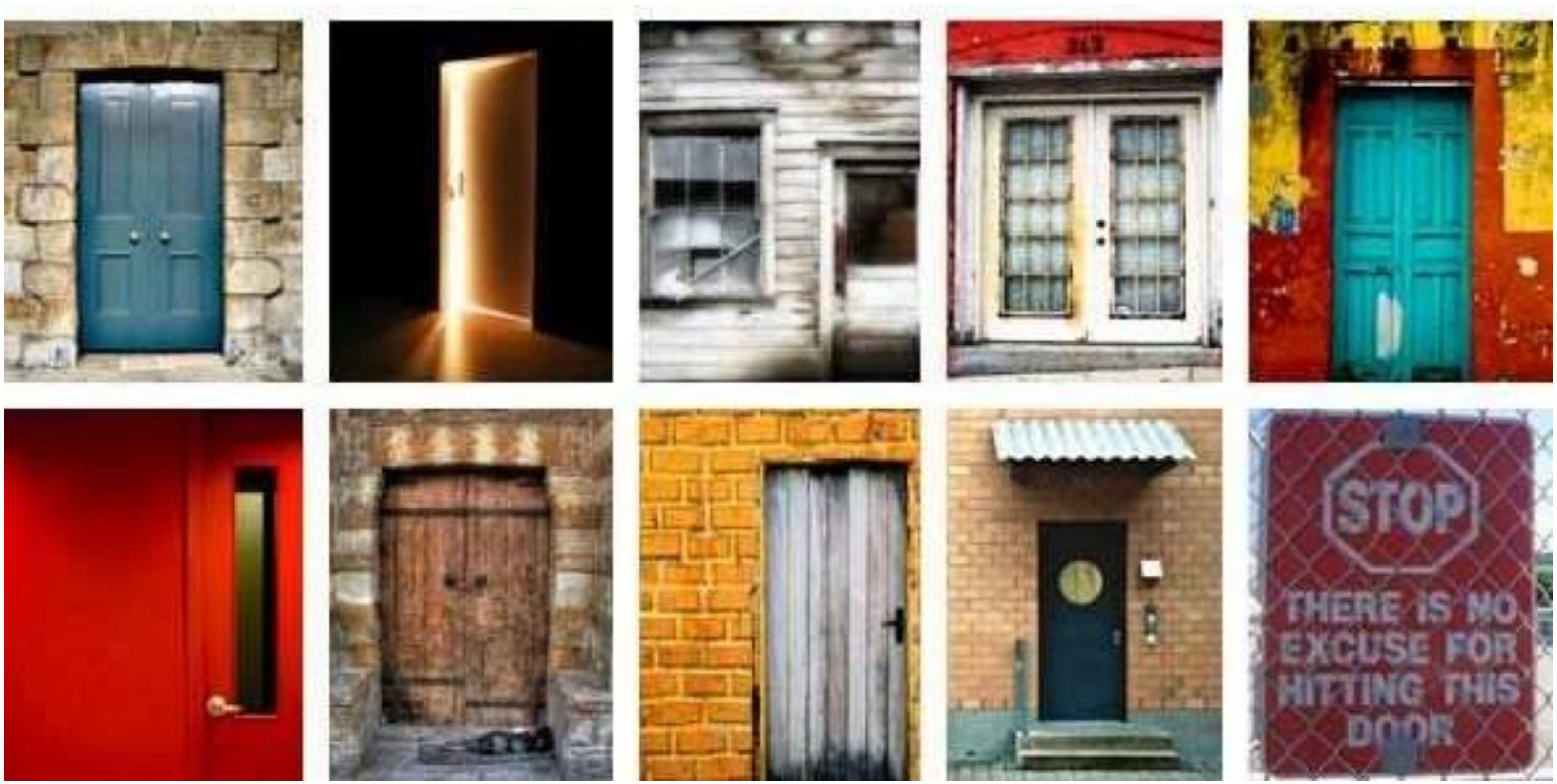
- Bonferroni's Principle
- Normalization (TF.IDF)

- Hash functions
- IO Bounded (Secondary Storage)
- Unstructured Data

- *Parallelism*
- *Functional Programming*

Statistical Limits. Goal: **Generalization**

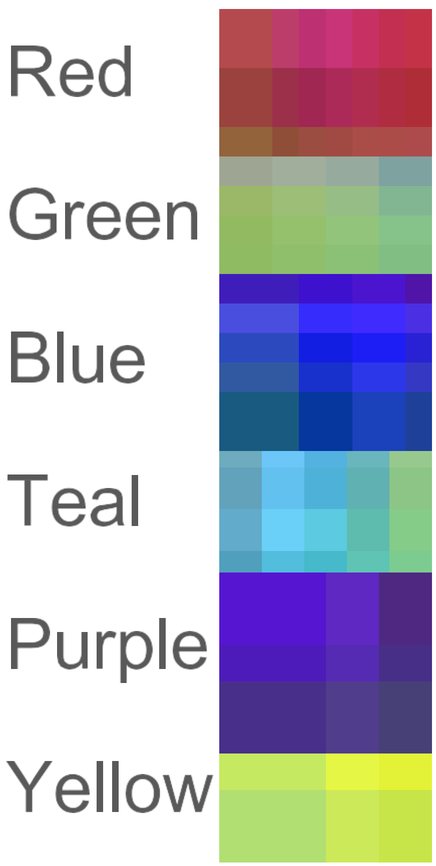
Bonferroni's Principle



Statistical Limits

Which iphone case will be least popular?

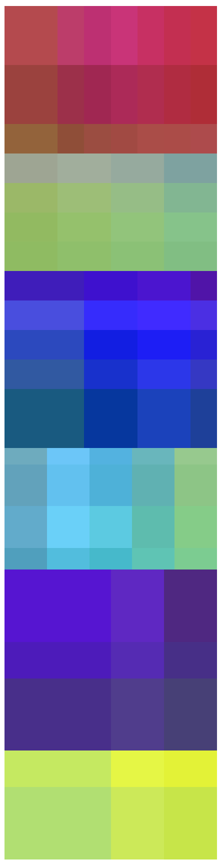
Bonferroni's Principle



Statistical Limits

Bonferroni's Principle

Red
Green
Blue
Teal
Purple
Yellow



Which iphone case will be least popular?

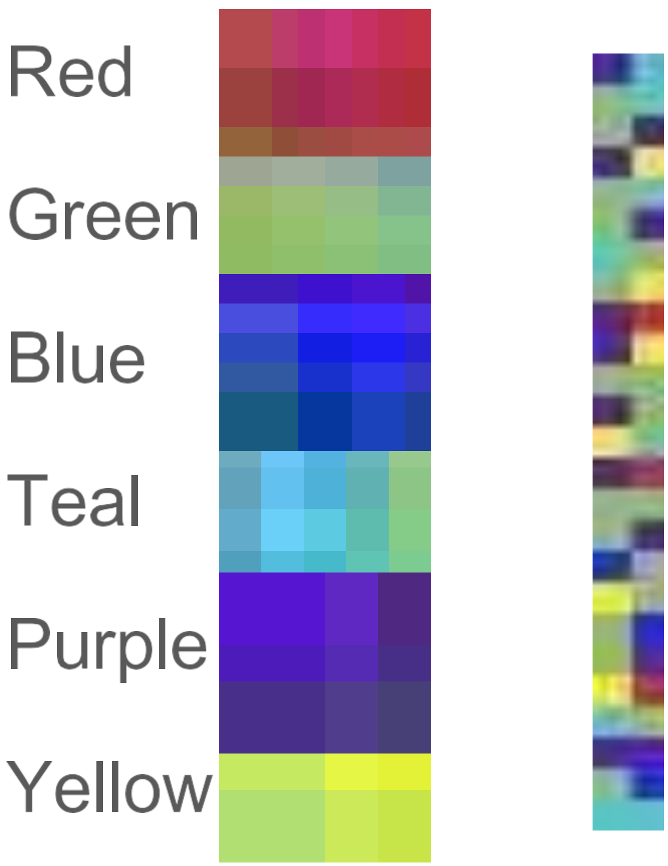
First 20 sales come in:

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20

Can you make any conclusions?

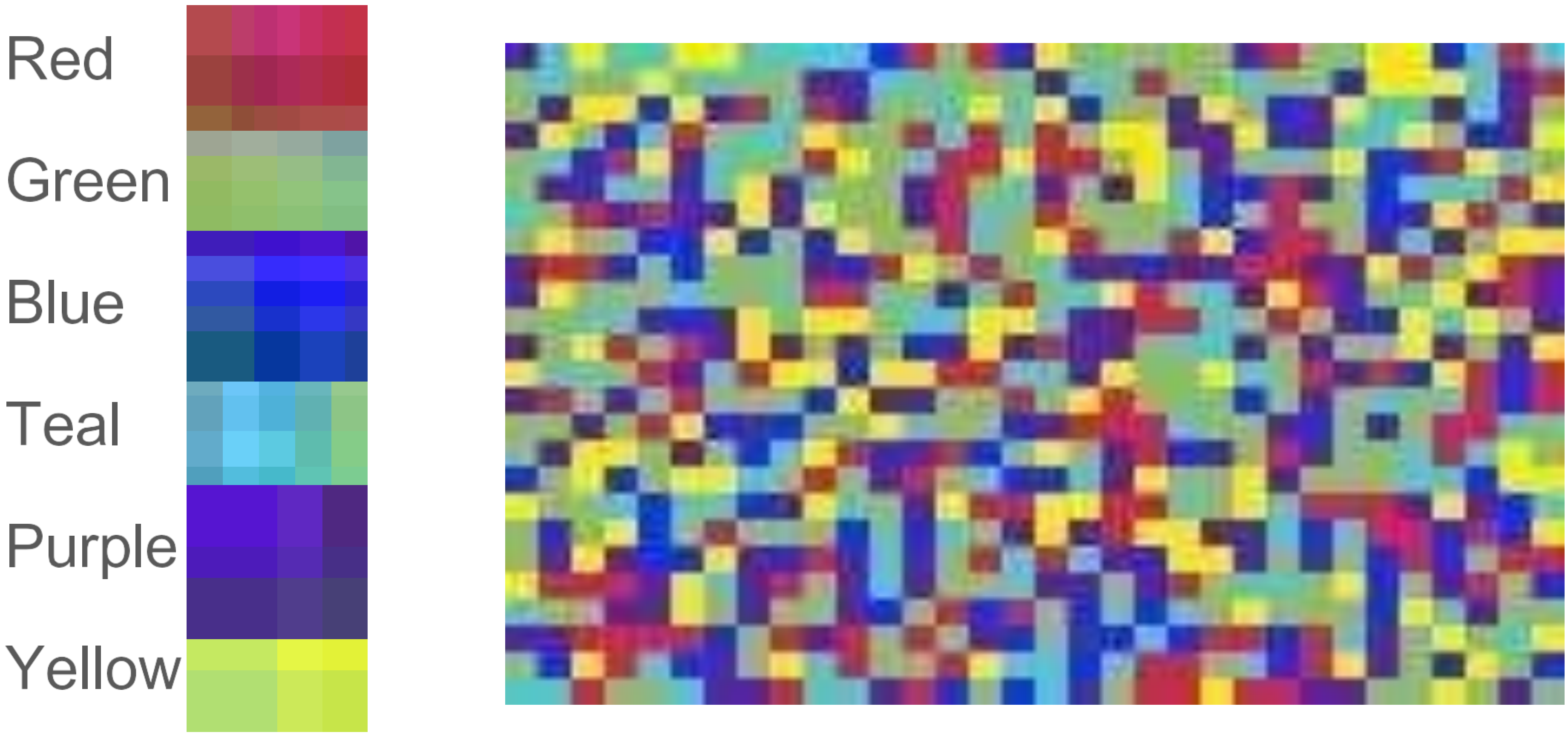
Statistical Limits

Bonferroni's Principle



Statistical Limits

Bonferroni's Principle



Statistical Limits. Goal: **Generalization**

Bonferroni's Principle

Roughly, calculating the probability of any of n *findings* being true requires n times the probability as testing for 1 finding.

https://xkcd.com/882/

In brief, one can only look for so many patterns (i.e. features) in the data before one finds something just by chance (i.e. finding something that does **not** generalize).

"Data mining" is a bad word in some communities!

Normalizing

Count data often need *normalizing* -- putting the numbers on the same "scale".

Prototypical example: TF.IDF of word *i* in document *j:*

Term Frequency:

$$tf_{ij} = \frac{count_{ij}}{\max_k count_{kj}}$$

$$tf.idf_{ij} = tf_{ij} \times idf_i$$

Inverse Document Frequency:

$$idf_i = log_2(\frac{docs_*}{docs_i}) \propto \frac{1}{\frac{docs_i}{docs_*}}$$

where docs is the number of documents containing word *i*.

Normalizing

Count data often need *normalizing* -- putting the numbers on the same "scale".

Prototypical example: TF.IDF of word *i* in document *j:*

Term Frequency:

$$tf_{ij} = \frac{count_{ij}}{\boxed{\max_k \, count_{kj}}}$$

$$tf.idf_{ij} = tf_{ij} \times idf_i$$

Inverse Document Frequency:

$$idf_i = \boxed{log_2(}\frac{docs_*}{docs_i}) \propto \frac{1}{\boxed{\frac{docs_i}{docs_*}}}$$

where docs is the number of documents containing word *i*.

Normalizing

Standardize: puts different sets of data (typically vectors or random variables) on the same scale with the same center.

- Subtract the mean (i.e. “mean center”)
- Divide by standard deviation

$$z_i = \frac{x_i - \bar{x}}{s_x}$$

Hash Functions and Indexes

Review:

h: *hash-key -> bucket-number*

Objective: uniformly distribute hash-keys across buckets.

Example: storing word counts.

Hash Functions and Indexes

Review:

h: *hash-key -> bucket-number*

Objective: uniformly distribute hash-keys across buckets.

Example: storing word counts.

$$h(word) = \left(\sum_{char \in word} ascii(char) \right) \% \, \#buckets$$

Hash Functions and Indexes

Review:

h: hash-key -> bucket-number

Objective: uniformly distribute hash-keys across buckets.

Example: storing word counts.

$$h(word) = \left(\sum_{char \in word} ascii(char) \right) \% \, \#buckets$$

Data structures utilizing hash-tables (i.e. O(1) lookup; dictionaries, sets in python) are a friend of big data algorithms! Review further if needed.

Hash Functions and Indexes

Review:

h: hash-key -> bucket-number

Objective: uniformly distribute hash-keys across buckets.

Example: storing word counts.

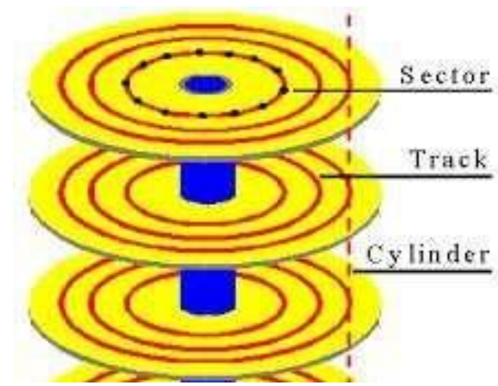
Database Indexes: Retrieve all records with a given *value.* (also review if unfamiliar / forgot)

Data structures utilizing hash-tables (i.e. O(1) lookup; dictionaries, sets in python) are a friend of big data algorithms! Review further if needed.

IO Bounded

Reading a word from disk versus main memory: 10^5 slower!

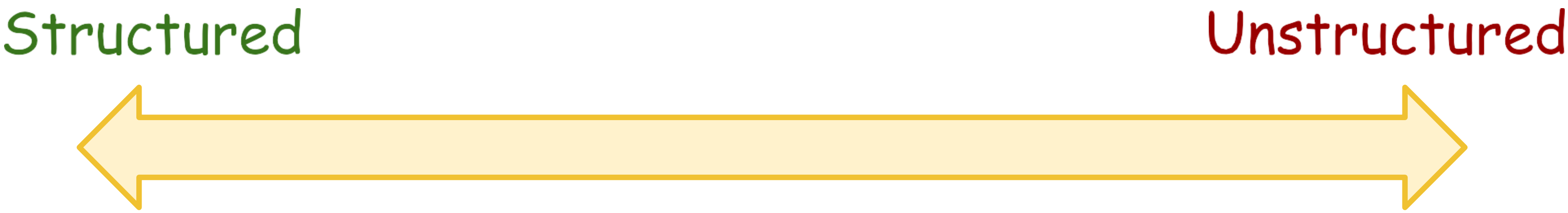
Reading many contiguously stored words is faster per word, but fast modern disks still only reach 150MB/s for sequential reads.



IO Bound: biggest performance bottleneck is reading / writing to disk.

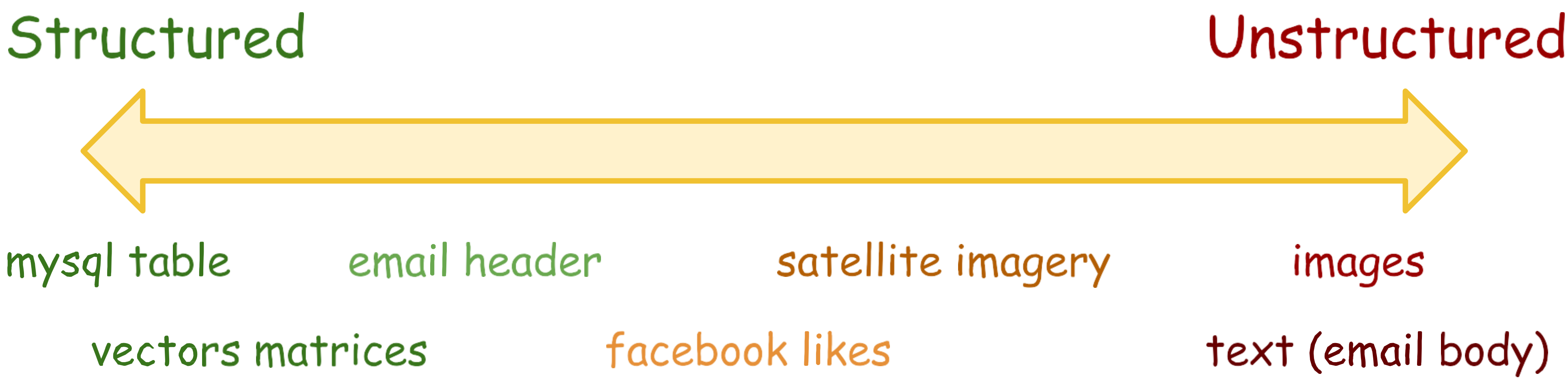
(starts around 100 GBs; ~10 minutes just to read).

Data



- Unstructured ≈ requires processing to get what is of interest
- Feature extraction used to turn unstructured into structured
- Near infinite amounts of potential features in unstructured data

Data



- Unstructured ≈ requires processing to get what is of interest
- Feature extraction used to turn unstructured into structured
- Near infinite amounts of potential features in unstructured data

Streaming Algorithms

Motivation

One often does not know when a set of data will end.

- Can not store
- Not practical to access repeatedly
- Rapidly arriving
- Does not make sense to ever “insert” into a database

Can not fit on disk but would like to generalize / summarize the data?

Motivation

One often does not know when a set of data will end.

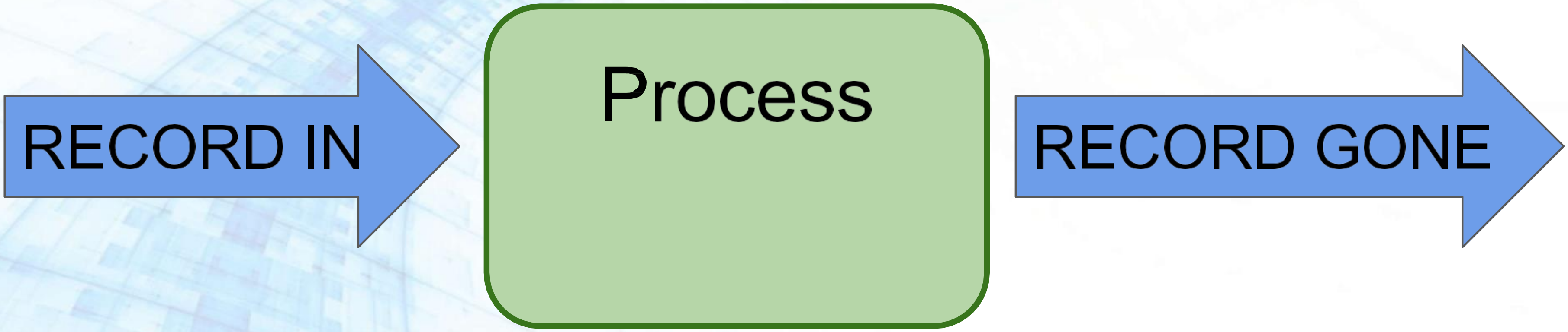
- Can not store
- Not practical to access repeatedly
- Rapidly arriving
- Does not make sense to ever “insert” into a database

Can not fit on disk but would like to generalize / summarize the data?

Examples: Google search queries
Satellite imagery data
Text Messages, Status updates
Click Streams

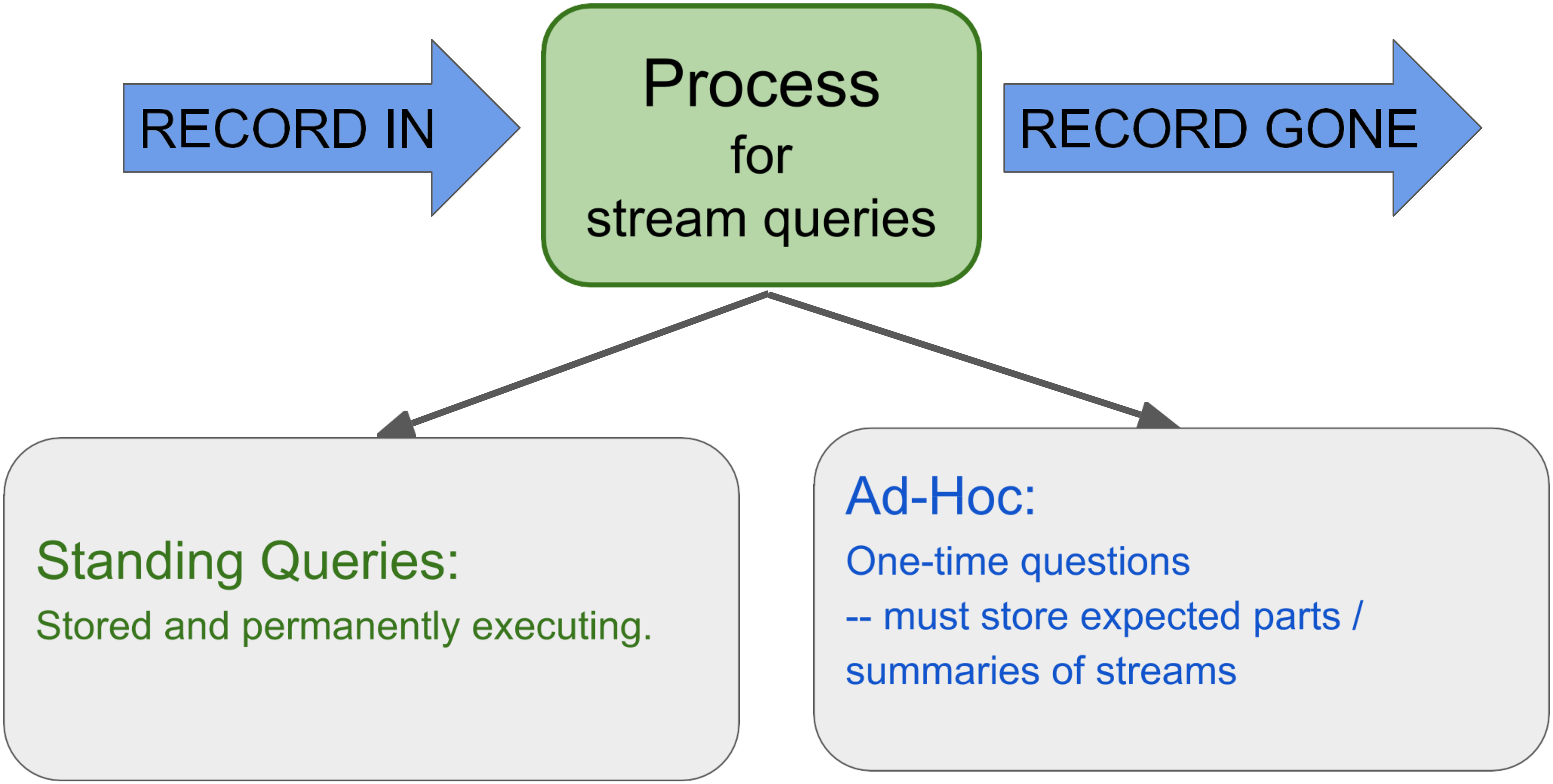
Motivation

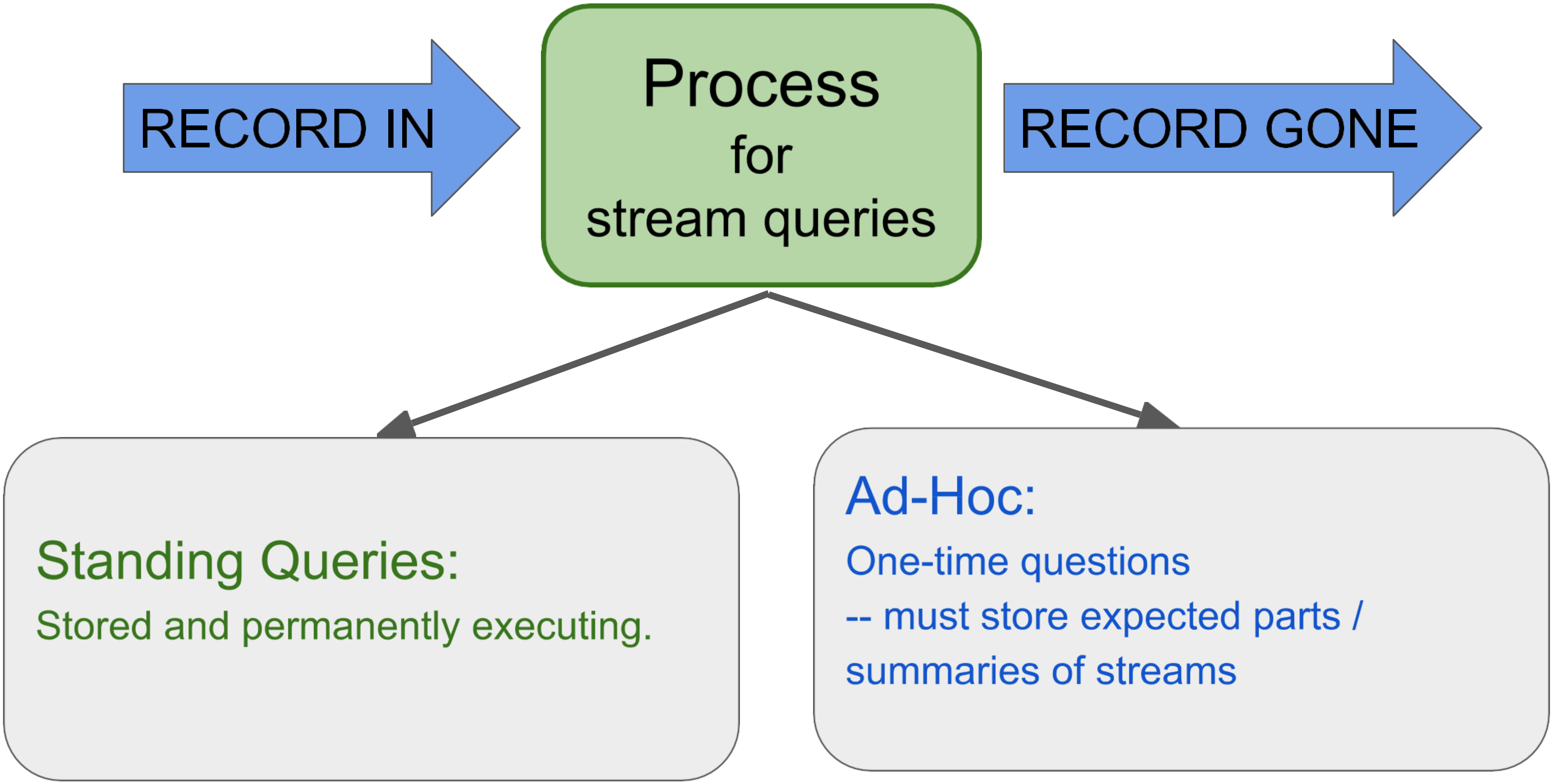
Often translate into O(N) algorithms.



We will cover the following algorithms:

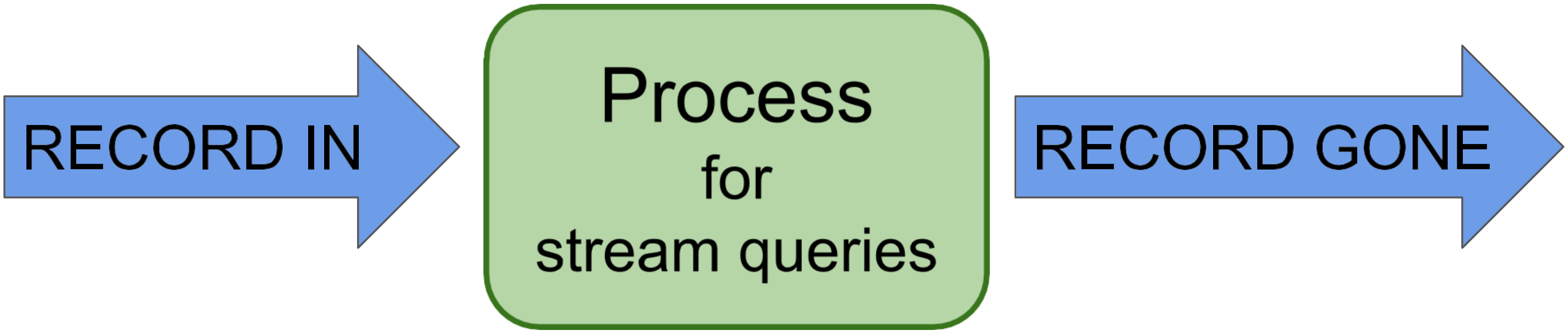
- General Stream Processing Model
- Sampling
- Filtering data according to a criteria
- Counting Distinct Elements


RECORD IN
Process
for
stream queries
RECORD GONE
Standing Queries:
Stored and permanently executing.
Ad-Hoc:
One-time questions
-- must store expected parts /
summaries of streams



E.g. How would you handle:

What is the mean of values seen so far?

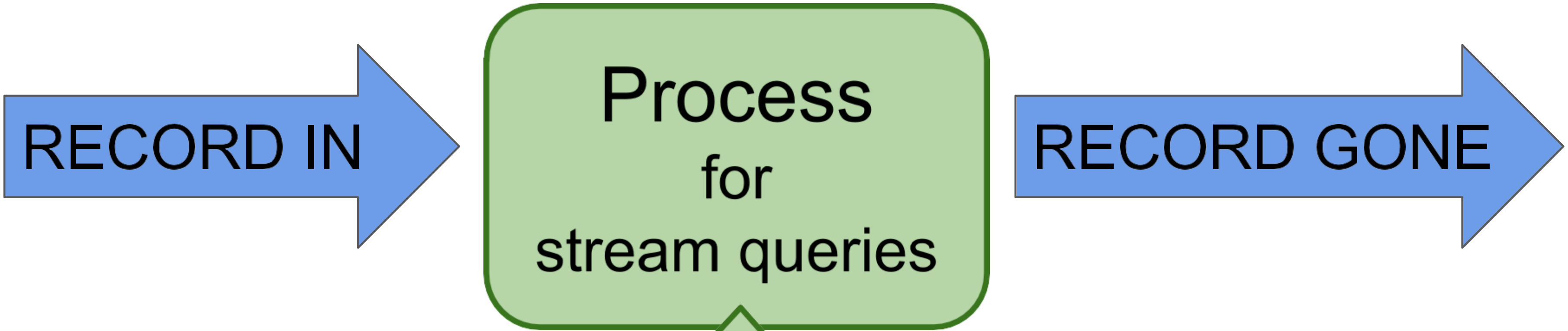


Important difference from typical database management:

- Input is not controlled by system staff.
- Input timing/rate is often unknown, controlled by users.

E.g. How would you handle:

What is the mean of values seen so far?



Might hold a sliding window of
records instead of single record.

.. , i, h, g, f, e, d, c, b, a

Important differen… …nagement:

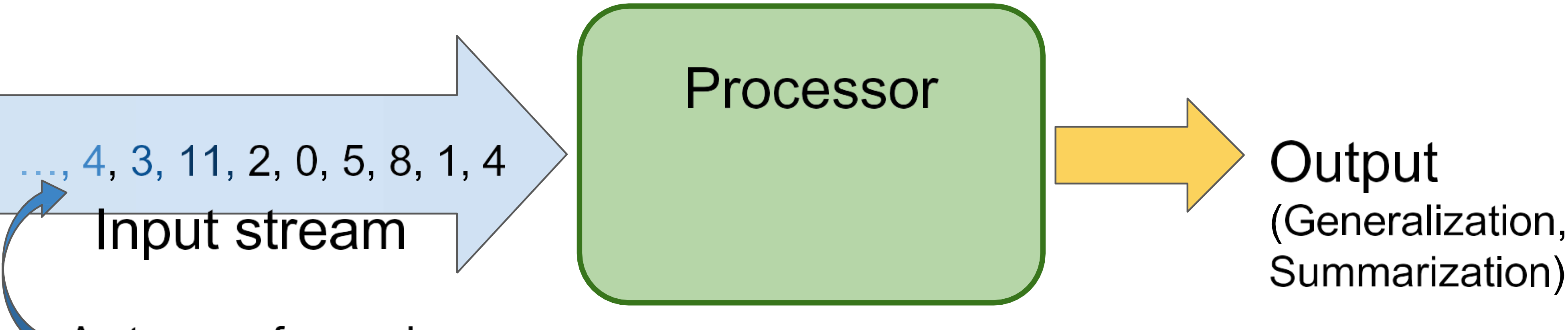
- Input is n…
- Input timing/rate is … …ntrolled by users.

E.g. How would you handle:

What is the mean of values seen so far?

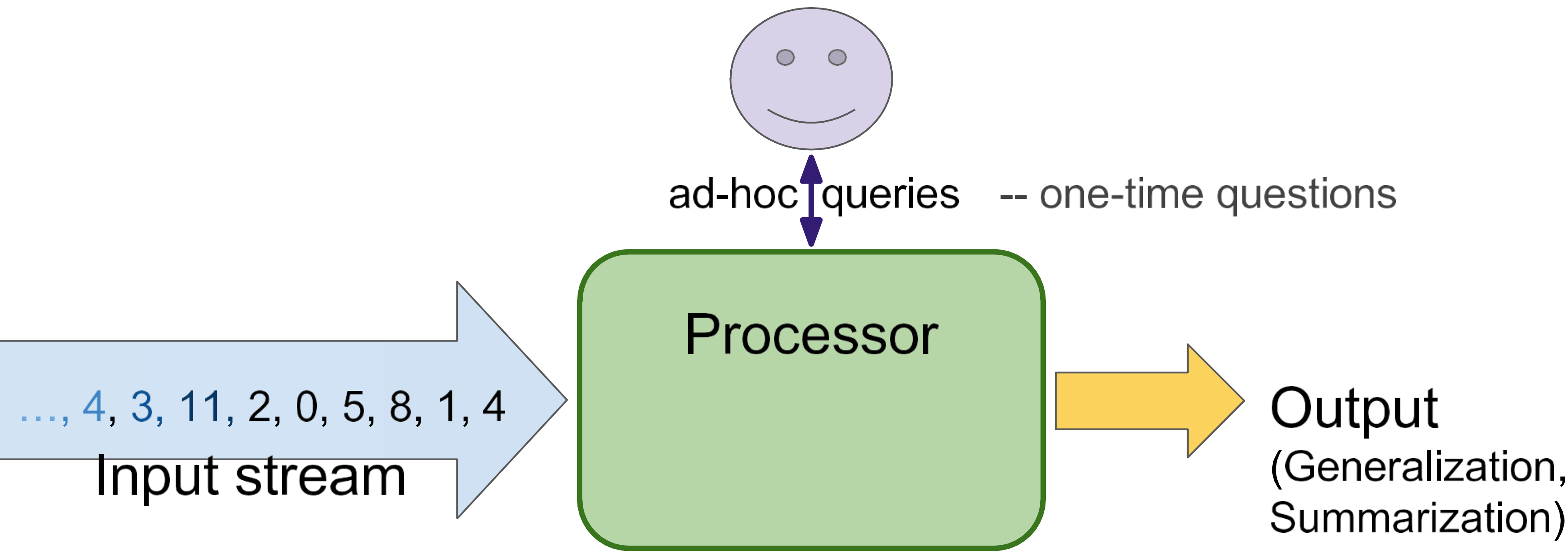
General Stream Processing Model

(Leskovec et al., 2014)

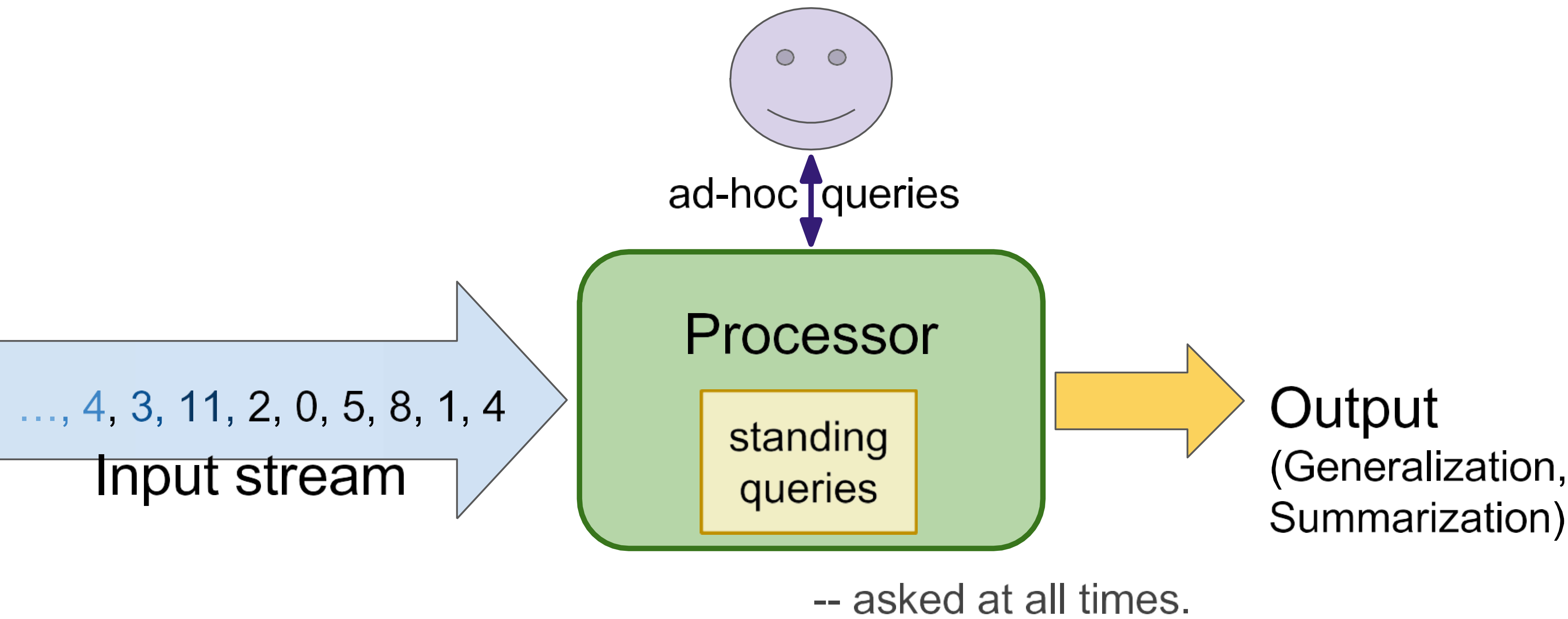


A stream of records
(also often referred to as “elements” or “tuples”)
Theoretically, could be anything! *search queries, numbers, bits, image files, ...*

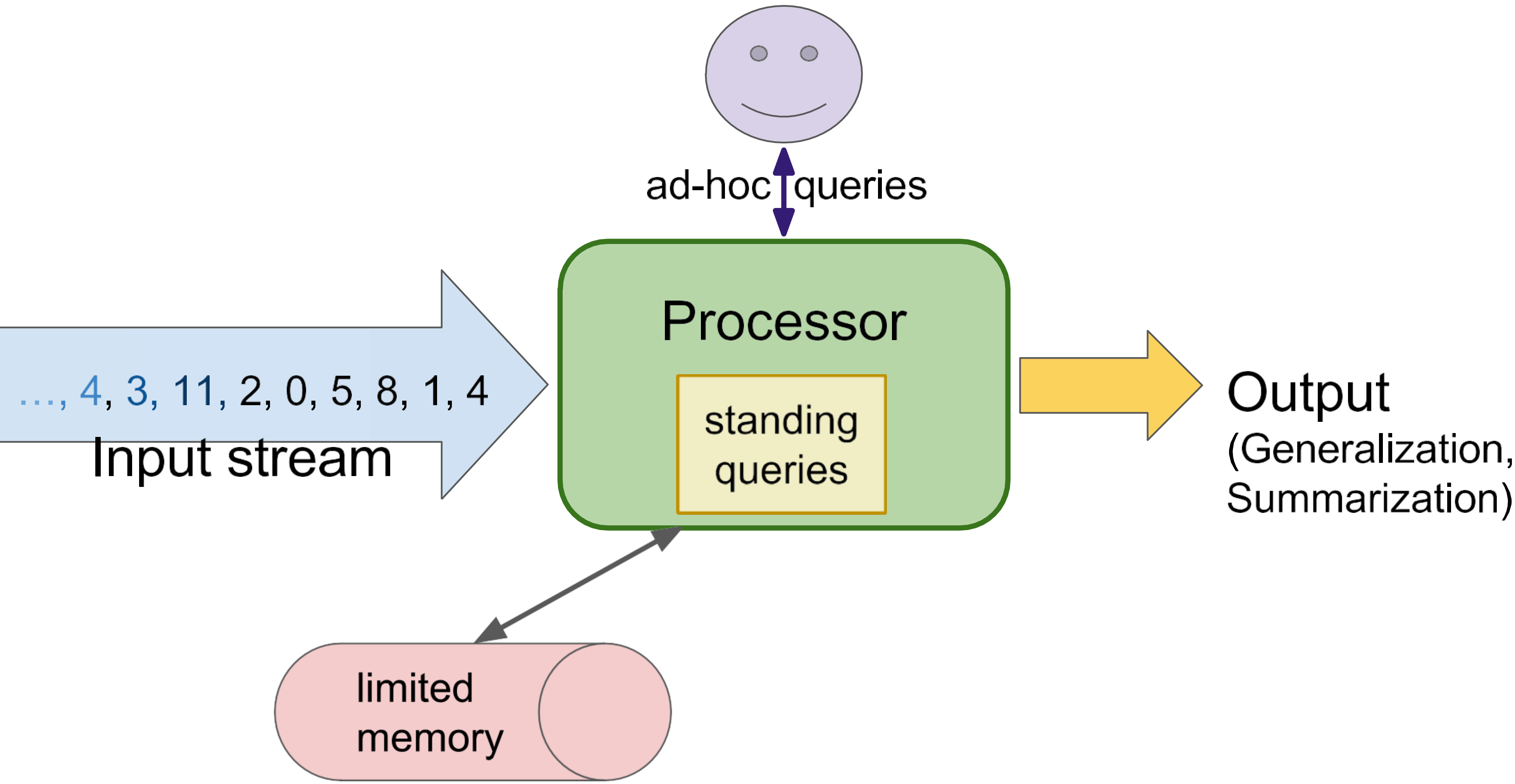
General Stream Processing Model



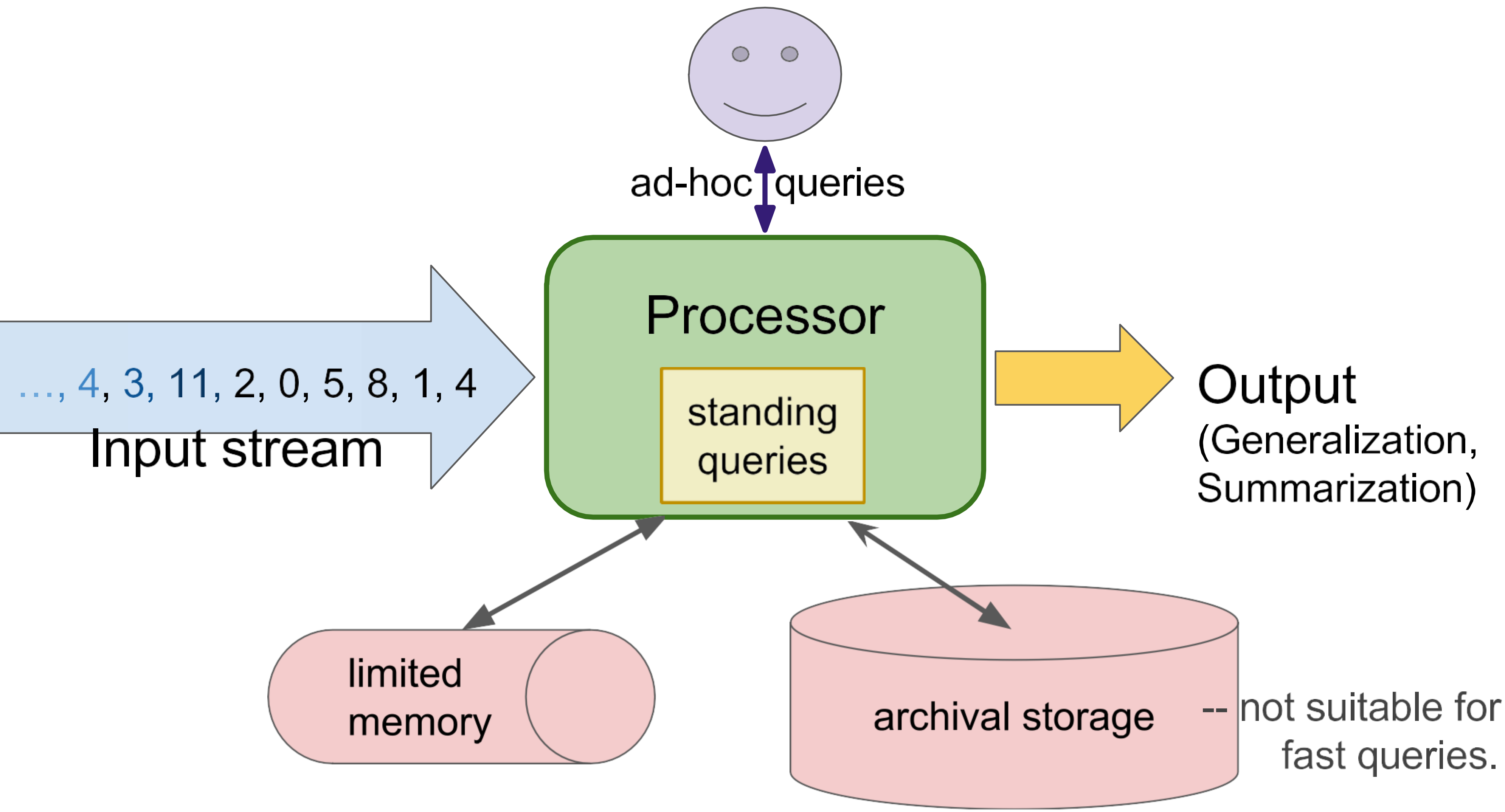
General Stream Processing Model



General Stream Processing Model



General Stream Processing Model

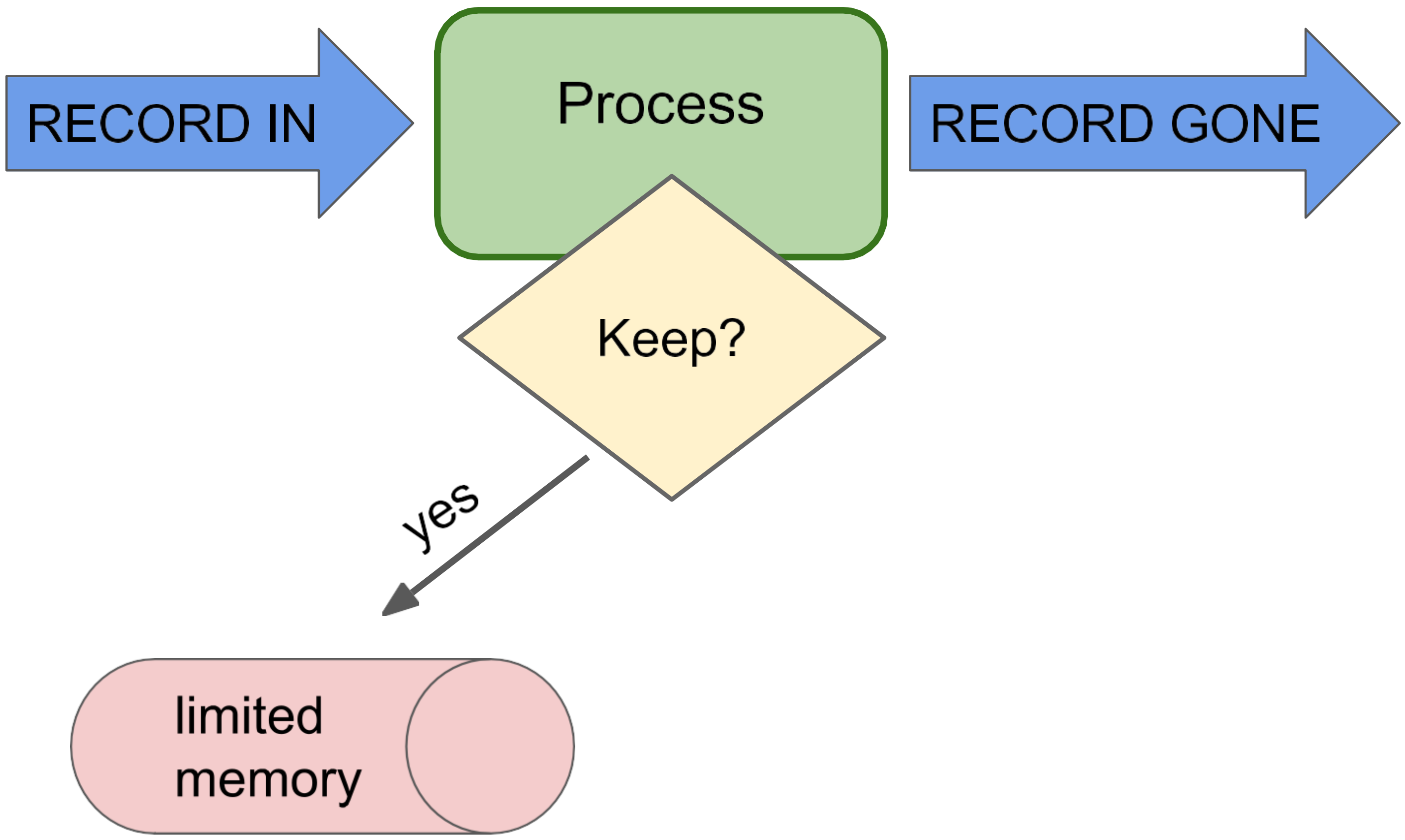


Sampling

Create a random sample for statistical analysis.

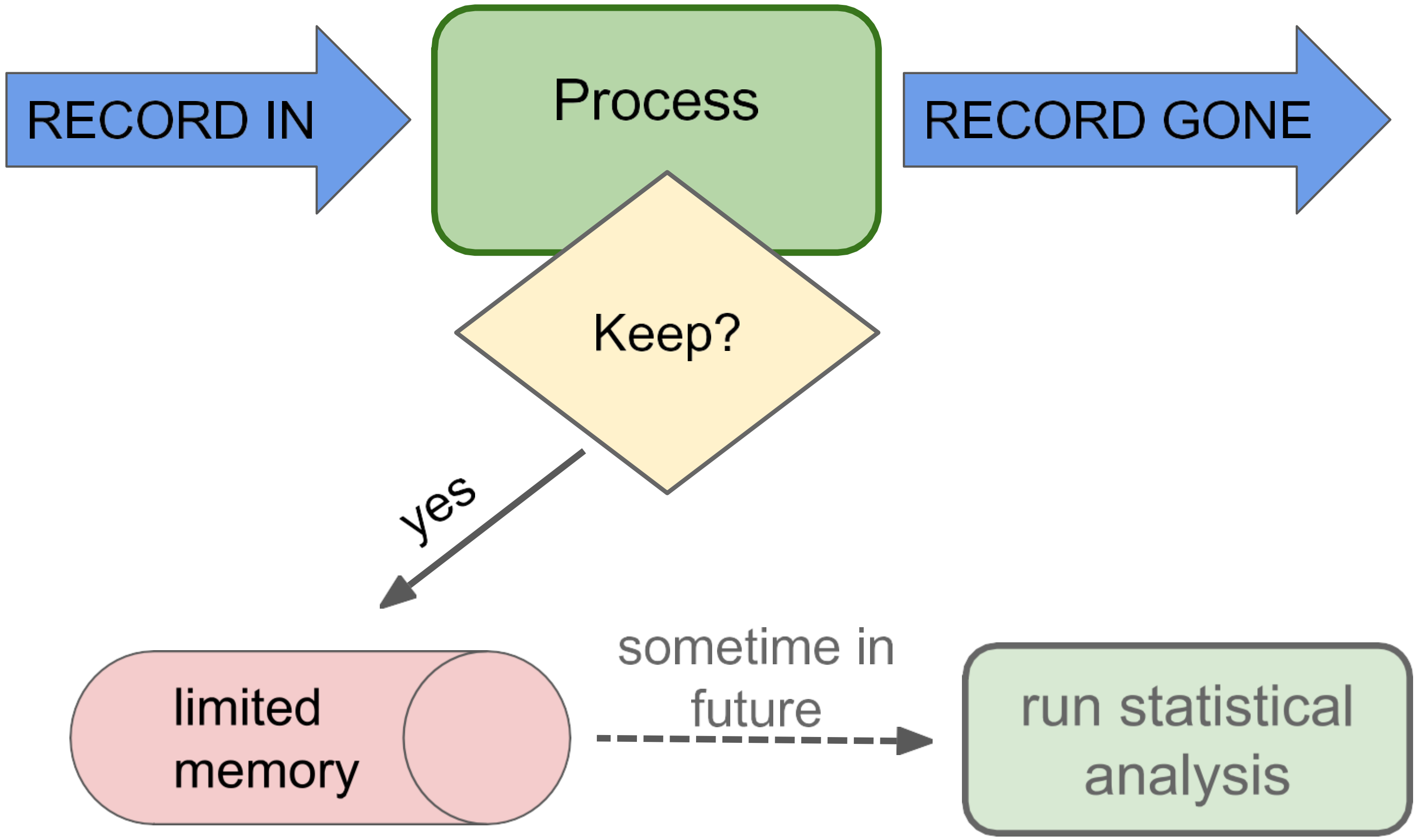
Sampling

Create a random sample for statistical analysis.



Sampling

Create a random sample for statistical analysis.



Sampling

Create a random sample for statistical analysis.

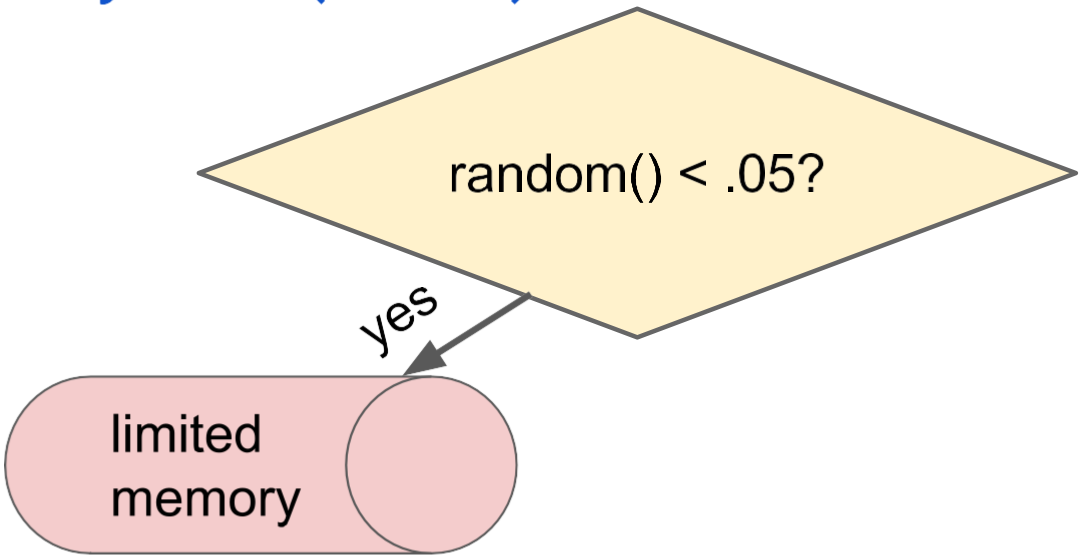
Simple Solution: generate a random number for each arriving record

Sampling

Create a random sample for statistical analysis.

Simple Solution: generate a random number for each arriving record

```
record = stream.next()
if random() <= .05: #keep: true 5% of the time
    memory.write(record)
```



Sampling

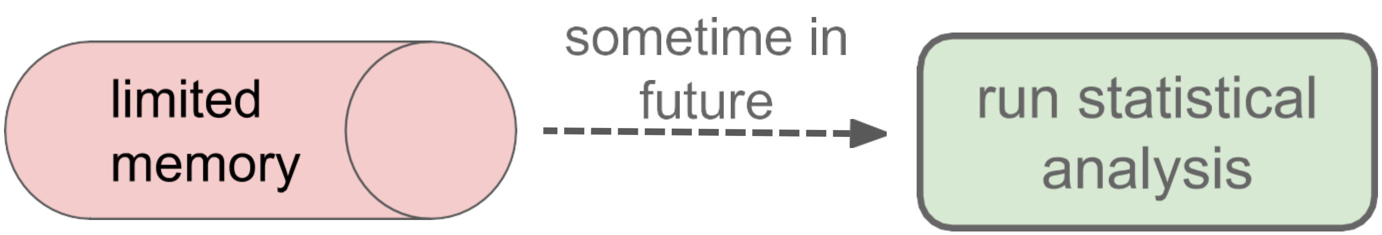
Create a random sample for statistical analysis.

Simple Solution: generate a random number for each arriving record

```
record = stream.next()
if random() <= .05: #keep: true 5% of the time
    memory.write(record)
```

Problem: records/rows often are not units-of-analysis for statistical analyses

E.g. user_ids for searches, tweets; location_ids for satellite images



Sampling

Create a random sample for statistical analysis.

Simple Solution: generate a random number for each arriving record

```
record = stream.next()
if random() <= perc: #keep: true perc% of the time
    memory.write(record)
```

Problem: records/rows often are not units-of-analysis for statistical analyses

E.g. user_ids for searches, tweets; location_ids for satellite images

Solution: hash into *N* = *1/perc* buckets; designate 1 bucket as “keep”.

```
if hash(record[‘user_id’]) == 1: #keep
```

Sampling

Create a random sample for statistical analysis.

Simple Solution: generate a random number for each arriving record

```
record = stream.next()
if random() <= perc: #keep: true perc% of the time
    memory.write(record)
```

Problem: records/rows often are not units-of-analysis for statistical analyses

E.g. user_ids for searches, tweets; location_ids for satellite images

Solution: hash into *N* = *1/perc* buckets; designate 1 bucket as “keep”.

```
if hash(record['user_id']) == 1: #keep
```

only need to store hash functions; may be part of standing query

Filtering Data

Filtering: Select elements with property x

Example: 40B safe email addresses for spam filter

Filtering Data

Filtering: Select elements with property x

Example: 40B safe email addresses for spam filter

The Bloom Filter (approximates; allows *false positives but not false negatives*)

Given:

|S| keys to filter; will be mapped to |B| bits

hashes = $h_1, h_2, \ldots, h_k$ independent hash functions

Filtering Data

Filtering: Select elements with property x

Example: 40B safe email addresses for spam filter

The Bloom Filter (approximates; allows *false positives but not false negatives*)

Given:

|S| keys to filter; will be mapped to |B| bits

hashes = $h_1, h_2, \ldots, h_k$ independent hash functions

Algorithm:

```
set all B to 0  #B is a bit vector
for each i in hashes, for each s in S:
  set B[hi(s)] = 1 #all bits resulting from
```

Filtering Data

Filtering: Select elements with property x

Example: 40B safe email addresses for spam filter

The Bloom Filter (approximates; allows *false positives but not false negatives*)

Given:

|S| keys to filter; will be mapped to |B| bits

hashes = $h_1, h_2, \ldots, h_k$ independent hash functions

Algorithm:

```
set all B to 0  #B is a bit vector
for each i in hashes, for each s in S:
  set B[h_i(s)] = 1 #all bits resulting from
  ... #usually embedded in other code
  while key x arrives next in stream #filter:
    if B[h_i(x)] == 1 for all i in hashes:
      do as if x is in S
    else: do as if x not in S
```

Filtering Data

Filtering: Select elements with property x

Example: 40B safe email addresses for spam filter

The Bloom Filter (approximates; allows *false positives but not false negatives*)

Given:

|S| keys to filter; will be mapped to |B| bits

hashes = $h_1, h_2, \ldots, h_k$ independent hash functions

Algorithm:

```
set all B to 0
for each i in hashes, for each s in S:
  set B[h_i(s)] = 1
  ... #usually embedded in other code
  while key x arrives next in stream
    if B[h_i(x)] == 1 for all i in hashes:
      do as if x is in S
    else: do as if x not in S
```

Filtering Data

Filtering: Select elements with property x

Example: 40B safe email addresses for spam filter

The Bloom Filter (approximates; allows *FPs*)

Given:

|S| keys to filter; will be mapped to |B| bits

hashes = $h_1, h_2, \ldots, h_k$ independent hash functions

Algorithm:

```
set all B to 0
for each i in hashes, for each s in S:
  set B[h_i(s)] = 1
  ... #usually embedded in other code
  while key x arrives next in stream
    if B[h_i(x)] == 1 for all i in hashes:
      do as if x is in S
    else: do as if x not in S
```

What is the probability of a *false positive*?

Q: What fraction of |B| are 1s?

(Leskovec et al., 2014)

Filtering Data

Filtering: Select elements with property x

Example: 40B safe email addresses for spam filter

The Bloom Filter (approximates; allows *FPs*)

Given:

|S| keys to filter; will be mapped to |B| bits

hashes = $h_1, h_2, \ldots, h_k$ independent hash functions

Algorithm:

```
set all B to 0
for each i in hashes, for each s in S:
  set B[h_i(s)] = 1
  ... #usually embedded in other code
  while key x arrives next in stream
    if B[h_i(x)] == 1 for all i in hashes:
      do as if x is in S
    else: do as if x not in S
```

What is the probability of a *false positive*?

Q: What fraction of |B| are 1s?

A: Analogy:

Throw |S| * *k* darts at *n* targets.

1 dart: $1/n$

d darts: $(1 - 1/n)^d$ = prob of 0

$= e^{-d/n}$ are **0s**

(Leskovec et al., 2014)

Filtering Data

Filtering: Select elements with property x

Example: 40B safe email addresses for spam filter

The Bloom Filter (approximates; allows *FPs*)

Given:

|S| keys to filter; will be mapped to |B| bits

hashes = $h_1, h_2, \ldots, h_k$ independent hash functions

Algorithm:

```
set all B to 0
for each i in hashes, for each s in S:
  set B[h_i(s)] = 1
  … #usually embedded in other code
  while key x arrives next in stream
    if B[h_i(x)] == 1 for all i in hashes:
      do as if x is in S
    else: do as if x not in S
```

What is the probability of a *false positive*?

Q: What fraction of |B| are 1s?

A: Analogy:

Throw |S| * k darts at n targets.

1 dart: $1/n$

d darts: $(1 - 1/n)^d$ = prob of 0

= $e^{-d/n}$ are **0s**

= e^{-1}
for large n

(Leskovec et al., 2014)

Filtering Data

Filtering: Select elements with property x

Example: 40B safe email addresses for spam filter

The Bloom Filter (approximates; allows *FPs*)

Given:

|S| keys to filter; will be mapped to |B| bits

hashes = $h_1, h_2, \ldots, h_k$ independent hash functions

Algorithm:

```
set all B to 0
for each i in hashes, for each s in S:
  set B[h_i(s)] = 1
  … #usually embedded in other code
  while key x arrives next in stream
    if B[h_i(x)] == 1 for all i in hashes:
      do as if x is in S
    else: do as if x not in S
```

What is the probability of a *false positive*?

Q: What fraction of |B| are 1s?

A: Analogy:

Throw |S| * k darts at n targets.

1 dart: $1/n$

d darts: $(1 - 1/n)^d$ = prob of 0

$= e^{-d/n}$ are **0s**

thus, $(1 - e^{-d/n})$ are **1s**

probability all k being 1?

(Leskovec et al., 2014)

Filtering Data

Filtering: Select elements with property x

Example: 40B safe email addresses for spam filter

The Bloom Filter (approximates; allows *FPs*)

Given:

|S| keys to filter; will be mapped to |B| bits

hashes = $h_1, h_2, \ldots, h_k$ independent hash functions

Algorithm:

```
set all B to 0
for each i in hashes, for each s in S:
  set B[h_i(s)] = 1
  … #usually embedded in other code
  while key x arrives next in stream
    if B[h_i(x)] == 1 for all i in hashes:
      do as if x is in S
    else: do as if x not in S
```

What is the probability of a *false positive*?

Q: What fraction of |B| are 1s?

A: Analogy:

Throw |S| * k darts at n targets.

1 dart: $1/n$

d darts: $(1 - 1/n)^d$ = prob of 0

= $e^{-d/n}$ are **0s**

thus, $(1 - e^{-d/n})$ are **1s**

probability all k being 1?

$\mathbf{(1 - e^{-(|S|*k)/n})^k}$

Note: Can expand S as stream continues as long as |B| has room (e.g. adding verified email addresses)

(Leskovec et al., 2014)

Counting Moments

Moments:

- Suppose m_i is the count of distinct element i in the data
- The kth moment of the stream is $\sum_{i \in \mathrm{Set}} m_i^k$

- 0th moment: count of distinct elements
- 1st moment: length of stream
- 2nd moment: sum of squares
  (measures *uneveness;* related to variance)

Counting Moments

Moments:

- Suppose m_i is the count of distinct element i in the data
- The kth m … m is $\sum_{i \in \mathrm{Set}} m_i^k$

Trivial: just increment a counter

- 0th momen … nct elements
- **1st moment: length of stream**
- 2nd moment: sum of squares
  (measures *uneveness;* related to variance)

Counting Moments

Applications
Counting…
distinct words in large document.
distinct websites (URLs).
users that visit a site.
unique queries to Alexa.

0th moment

- **0th moment: count of distinct elements**
- 1st moment: length of stream
- 2nd moment: sum of squares
(measures *uneveness;* related to variance)

Counting Moments

Applications
Counting…
distinct words in large document.
distinct websites (URLs).
users that visit a site.
unique queries to Alexa.

0th moment
One Solution: Just keep a set (hashmap, dictionary, heap)

Problem: Can't maintain that many in memory; disk storage is too slow

- **0th moment: count of distinct elements**
- 1st moment: length of stream
- 2nd moment: sum of squares
  (measures *uneveness;* related to variance)

Counting Moments

0th moment

Streaming Solution: Flajolet-Martin Algorithm

General idea:

- n -- suspected total number of elements observed
- pick a hash, *h,* to map each element to $\log_2 n$ bits (buckets)

- 2nd moment: sum of squares
  (measures *uneveness;* related to variance)

Counting Moments

0th moment

Streaming Solution: Flajolet-Martin Algorithm

General idea:

n -- suspected total number of elements observed

pick a hash, *h,* to map each element to $\log_2 n$ bits (buckets)

```
--------------------------------
R = 0 #potential max number of zeros at tail
for each stream element, e:
    r(e) = trailZeros(h(e) #num of trailing 0s from h(e)
    R = r(e) if r[e] > R

estimated_distinct_elements = 2^R
```

- 2nd moment: sum of squares
(measures *uneveness;* related to variance)

Counting Moments

0th moment

Streaming Solution: Flajolet-Martin Algorithm

General idea:

n -- suspected total number of elements observed

pick a hash, *h,* to map each element to $\log_2 n$ bits (buckets)

```
--------------------------------
R = 0 #potential max number of zeros at tail
for each stream element, e:
    r(e) = trailZeros(h(e) #num of trailing 0s from h(e)
    R = r(e) if r[e] > R

estimated_distinct_elements = 2^R # m
```

Mathematical Intuition

P(trailZeros(*h*(*e*)) >= *i*) = 2^{-i}

 # P(h(e) == __0) = .5; P(h(e) == __00) = .25; …

P(trailZeros(*h*(*e*)) < *i*) = $1 - 2^{-i}$

for m elements: $= (1 - 2^{-i})^m$

P(one *e* has tailZeros > i) = $1 - (1 - 2^{-i})^m$

$\approx 1 - e^{-m2^{\wedge}-i}$

If $2^R \gg m$, then $1 - (1 - 2^{-i})^m \approx 0$

If $2^R \ll m$, then $1 - (1 - 2^{-i})^m \approx 1$

- 2nd moment: sum of squares
  (measures *uneveness;* related to variance)

Counting Moments

0th moment

Streaming Solution: Flajolet-Martin Algorithm

General idea:

n -- suspected total number of elements observed

pick a hash, *h,* to map each element to $\log_2 n$ bits (buckets)

```
--------------------------------
R = 0 #potential max number of ze
for each stream element, e:
    r(e) = trailZeros(h(e) #num of
    R = r(e) if r[e] > R

estimated_distinct_elements = 2^R =
```

- 2nd moment: sum of squares (measures *uneveness;* related to variance)

Mathematical Intuition

P(`trailZeros(h(e))` >= *i*) = 2^{-i}

 # P(h(e) == __0) = .5; P(h(e) == __00) = .25; …

P(`trailZeros(h(e))` < *i*) = $1 - 2^{-i}$

for m elements: $= (1 - 2^{-i})^m$

P(one *e* has tailZeros > i) = $1 - (1 - 2^{-i})^m$

$\approx 1 - e^{-m2^{\wedge}-i}$

If 2^R >> m, then $1 - (1 - 2^{-i})^m \approx 0$

If 2^R << m, then $1 - (1 - 2^{-i})^m \approx 1$

Problem:

Unstable in practice.

Solution:

Multiple hash functions

but how to combine?

0th moment

Streaming Solution: Flajolet-Martin Algorithm

General idea:

n -- suspected total number of elements

pick a hash, *h*, to map each element to l

```
Rs = list()
for h in hashes:
    R = 0 #potential max number of zeros at tail
    for each stream element, e:
        r(e) = trailZeros(h(e) #num of trailing 0s from h(e)
        R = r(e) if r[e] > R
    Rs.append(2^R)

groupRs = Rs[i:i+log n] for i in range(0, len(Rs), log n)

estimated_distinct_elements = median(map(mean, groupRs))
```

Problem:

Unstable in practice.

Solution: Multiple hash functions

1. Partition into groups of size log n
2. Take mean in groups
3. Take median of group means

0th moment

Streaming Solution: Flajolet-Martin Algorithm

General idea:

n -- suspected total number of elements

pick a hash, *h*, to map each element to l

```
Rs = list()
for h in hashes:
    R = 0 #                          ros at tail
    fo
                                          ling 0s from h(e)

    Rs.appe

groupRs = Rs[i:i+log n] for i in range(0, len(Rs), log n)

estimated_distinct_elements = median(map(mean, groupRs))
```

Problem:

Unstable in practice.

Solution: Multiple hash functions

1. Partition into groups of size log n
2. Take mean in groups
3. Take median of group means

A good approach anytime one has many "low resolution" estimates of a true value.

Counting Moments

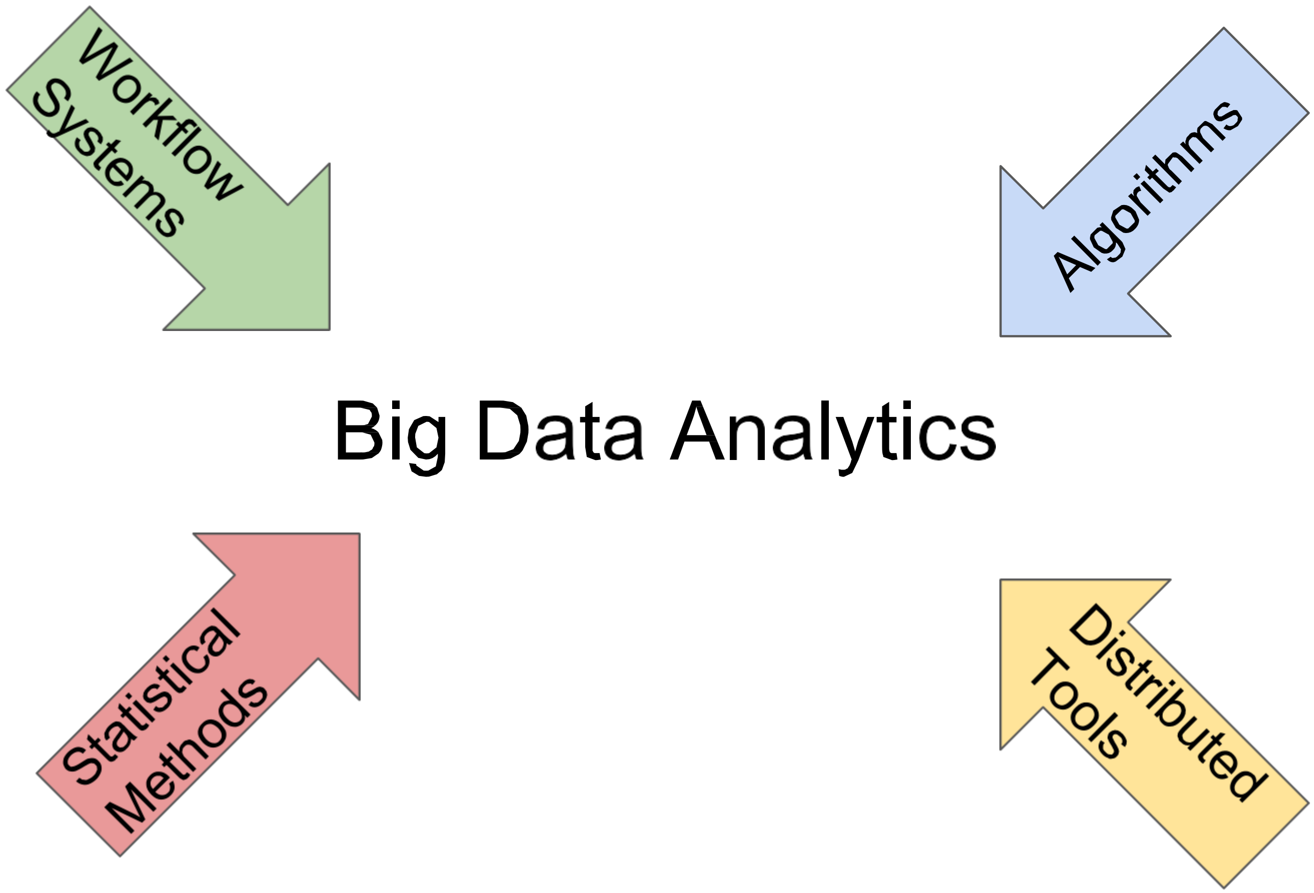
2nd moment

Streaming Solution: Alon-Matias-Szegedy Algorithm

(Exercise; Out of Scope; see in MMDS)

- 0th moment: count of distinct elements
- 1st moment: length of stream
- **2nd moment: sum of squares (measures *uneveness* related to variance)**

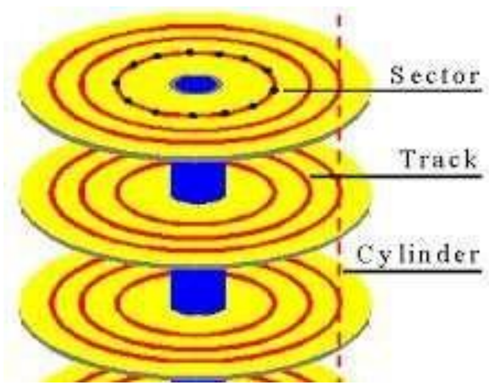
Hadoop
and
MapReduce


Workflow Systems
Algorithms
Big Data Analytics
Statistical Methods
Distributed Tools

IO Bounded

Reading a word from disk versus main memory: 10^5 slower!

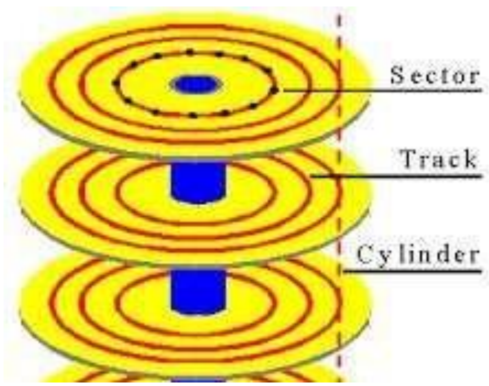
Reading many contiguously stored words is faster per word, but fast modern disks still only reach 150MB/s for sequential reads.



IO Bounded

Reading a word from disk versus main memory: 10^5 slower!

Reading many contiguously stored words is faster per word, but fast modern disks still only reach 150MB/s for sequential reads.

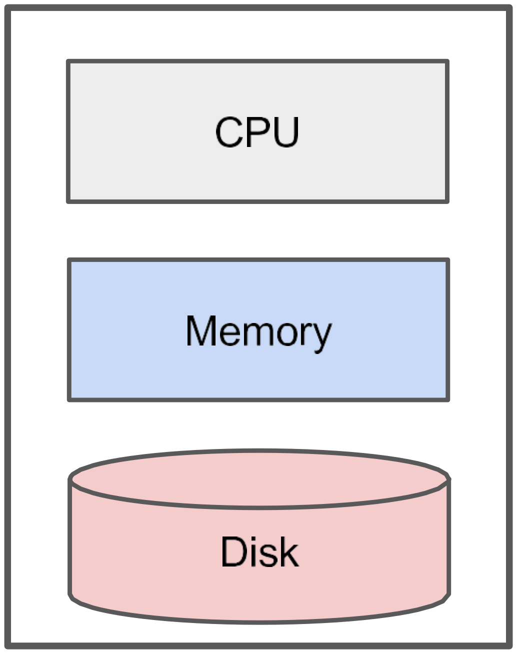


IO Bound: biggest performance bottleneck is reading / writing to disk.

starts around 100 GBs: ~10 minutes just to read

200 TBs: ~20,000 minutes = 13 days

Classical Big Data Analysis



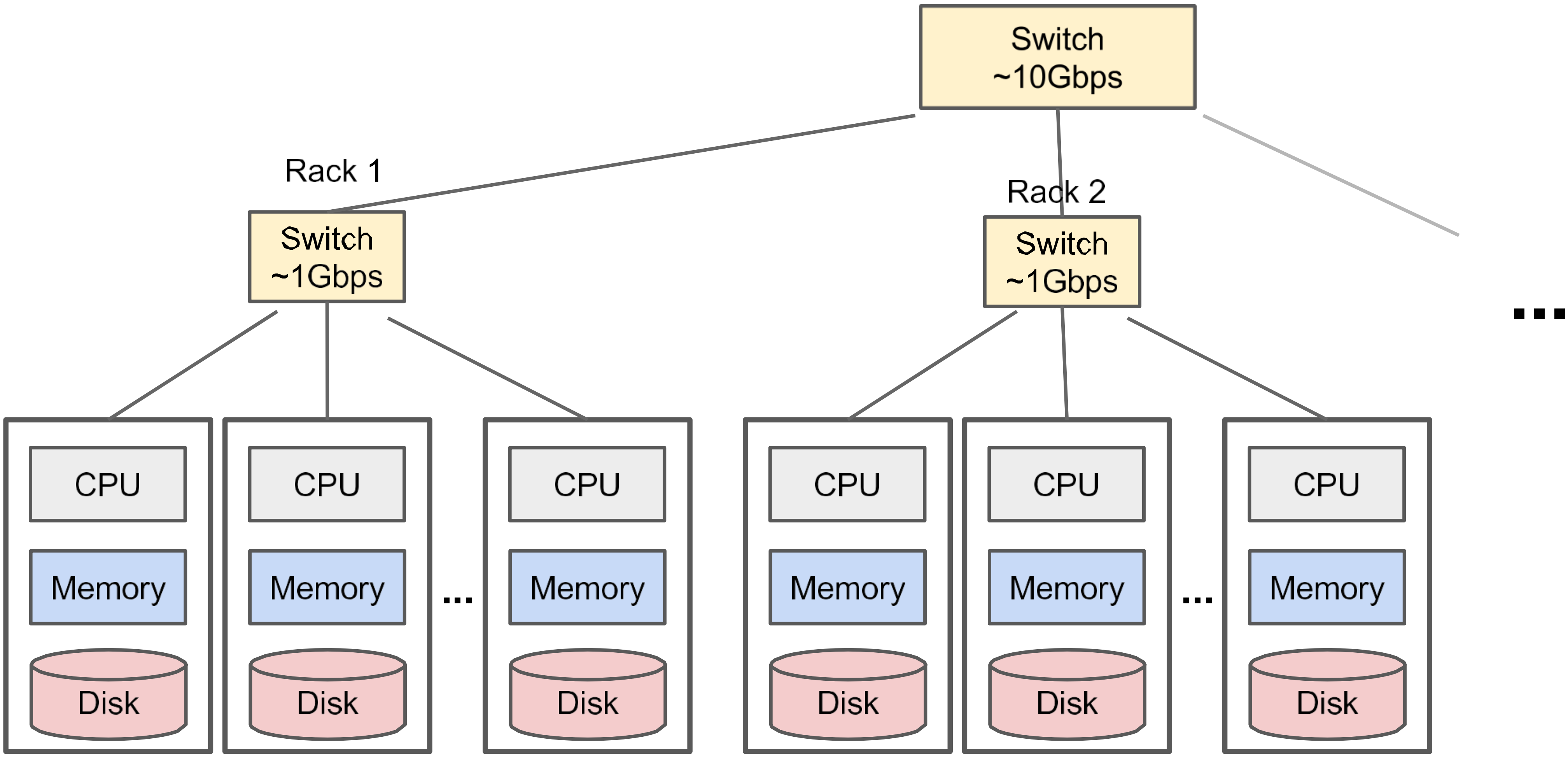
Classical focus: efficient use of disk.
e.g. Apache Lucene / Solr

Classical limitation: Still bounded when needing to process all of a large file.

IO Bound

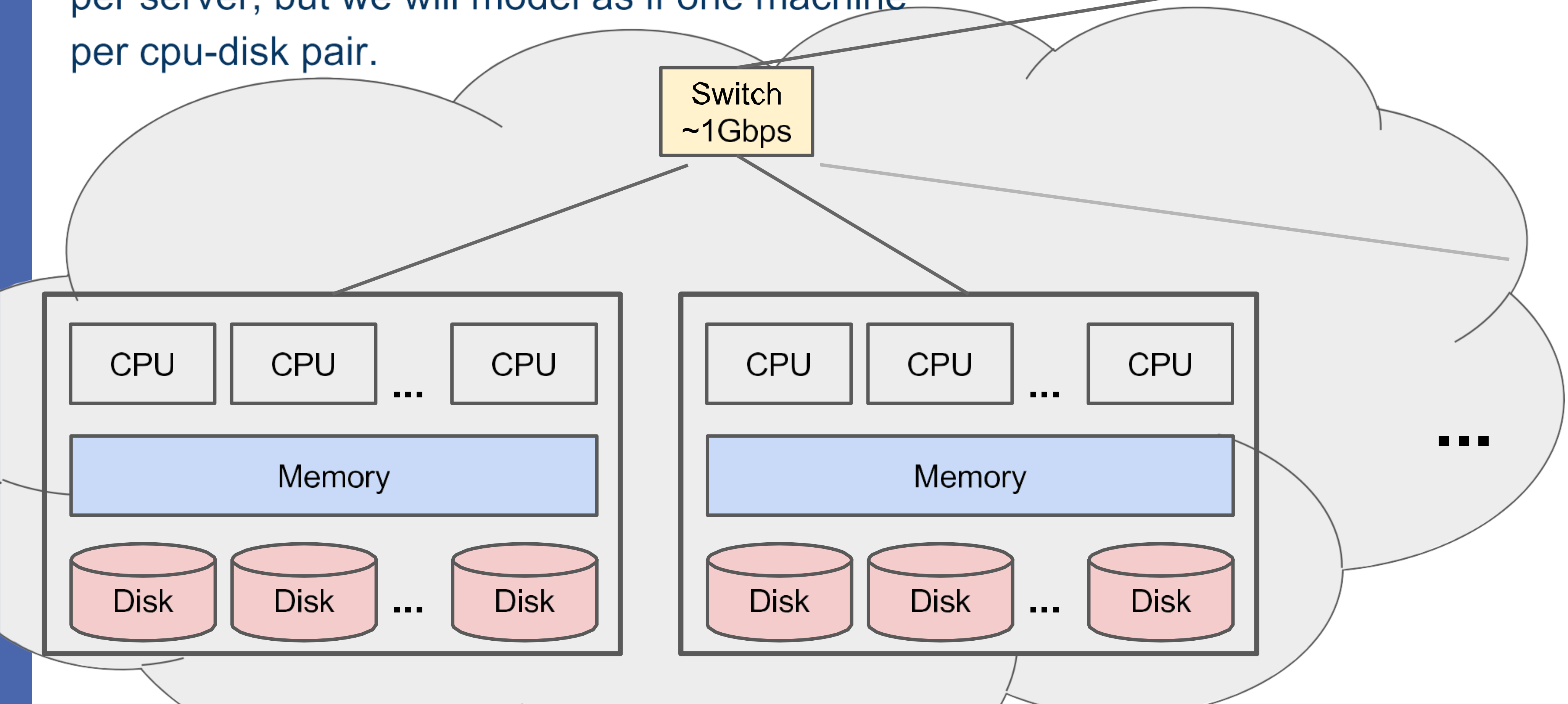
How to solve?

Distributed Architecture (Cluster)

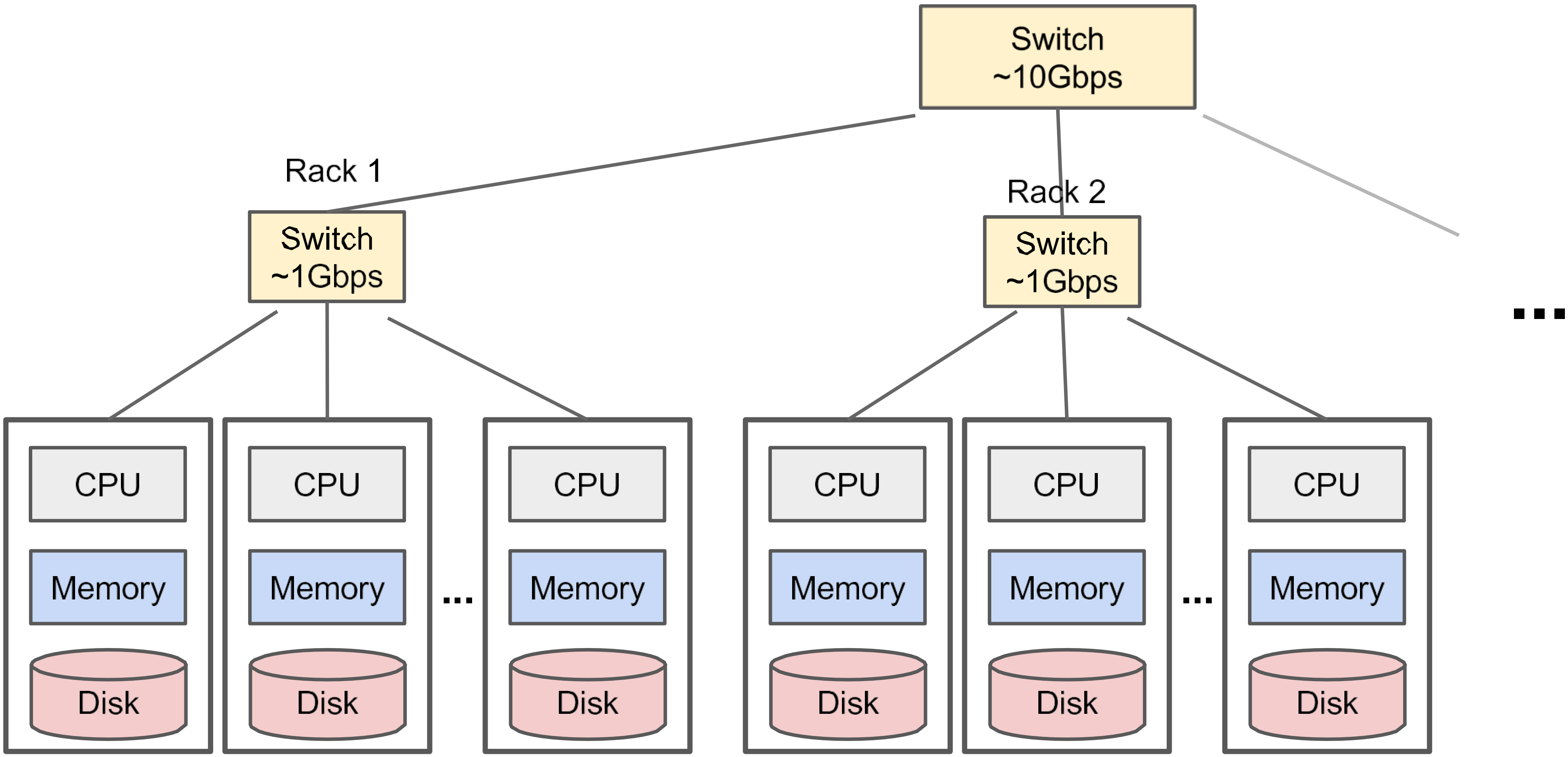


Distributed Architecture (Cluster)

In reality, modern setups often have multiple cpus and disks per server, but we will model as if one machine per cpu-disk pair.



Distributed Architecture (Cluster)



Challenges for IO Cluster Computing

1. Nodes fail
   1 in 1000 nodes fail a day

2. Network is a bottleneck
   Typically 1-10 Gb/s throughput

3. Traditional distributed programming is
   often ad-hoc and complicated

Challenges for IO Cluster Computing

1. Nodes fail
   1 in 1000 nodes fail a day
   Duplicate Data
2. Network is a bottleneck
   Typically 1-10 Gb/s throughput
   Bring computation to nodes, rather than data to nodes.
3. Traditional distributed programming is often ad-hoc and complicated
   Stipulate a programming system that can easily be distributed

Challenges for IO Cluster Computing

1. Nodes fail
   1 in 1000 nodes fail a day
   Duplicate Data
2. Network is a bottleneck
   Typically 1-10 Gb/s throughput
   Bring computation to nodes, rather than data to nodes.
3. Traditional distributed programming is often ad-hoc and complicated
   Stipulate a programming system that can easily be distributed

MapReduce Accomplishes

Distributed File System

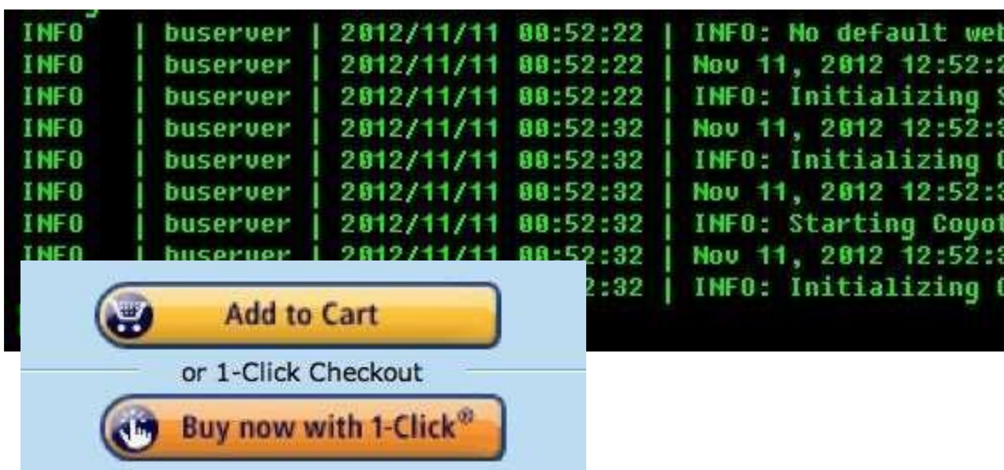
The effectiveness of MapReduce is in part simply due to use of a distributed filesystem!

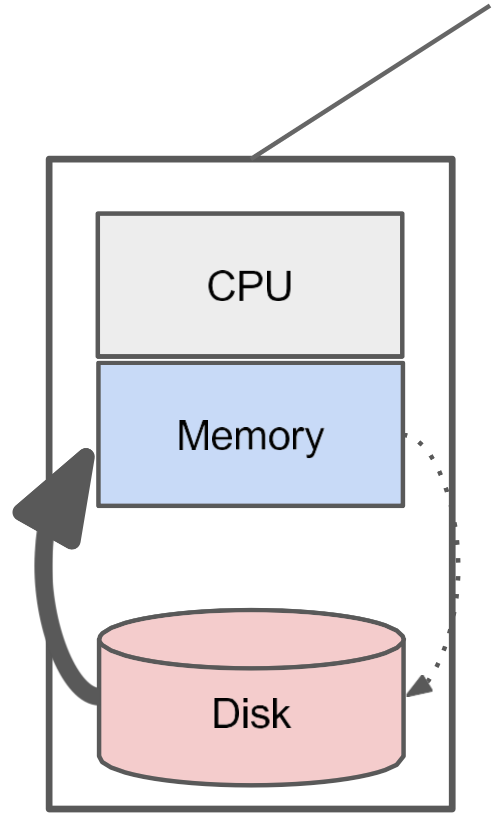
Characteristics for Big Data Tasks

Large files (i.e. >100 GB to TBs)

Reads are most common

No need to update in place
(append preferred)

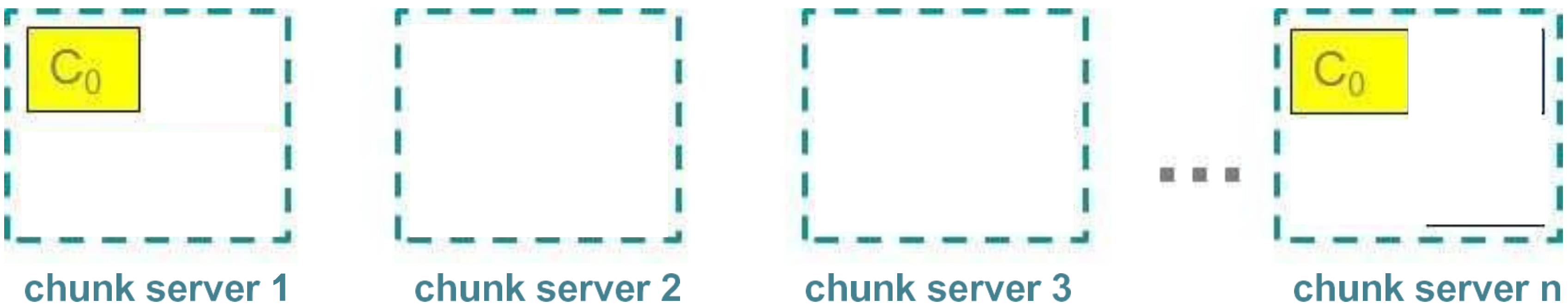




Distributed File System

(e.g. Apache HadoopDFS, GoogleFS, EMRFS)

C, D: Two different files

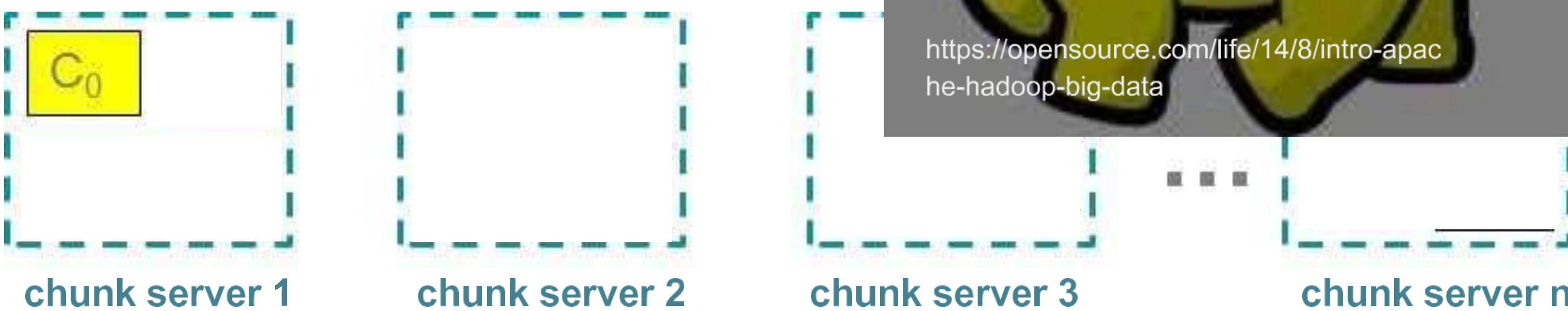


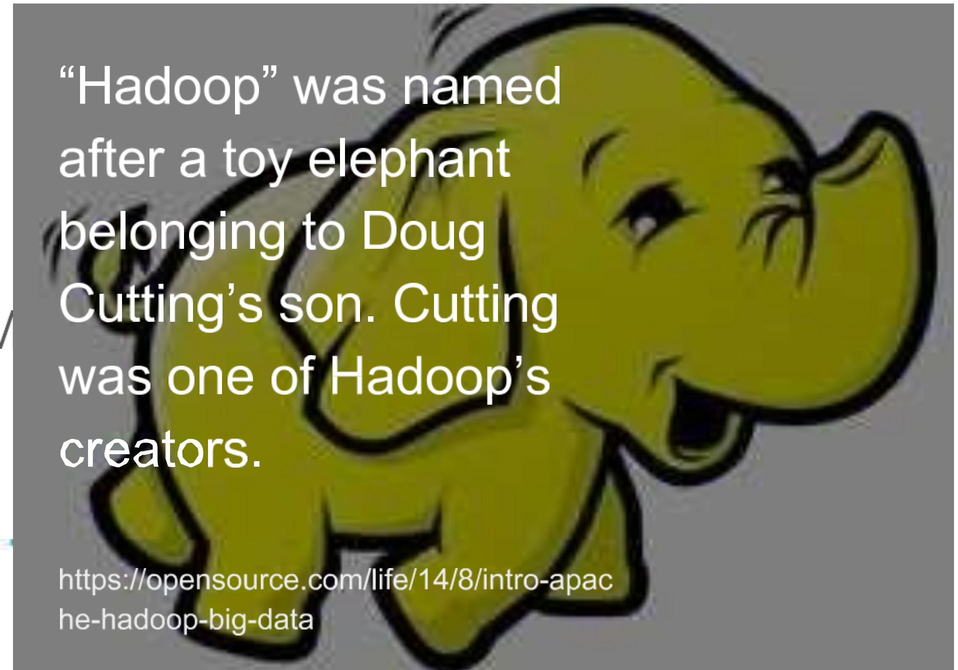
(Leskovec at al., 2014; http://www.mmds.org/)

Distributed File System

(e.g. Apache **Hadoop**DFS, GoogleFS, EM

C, D: Two different files



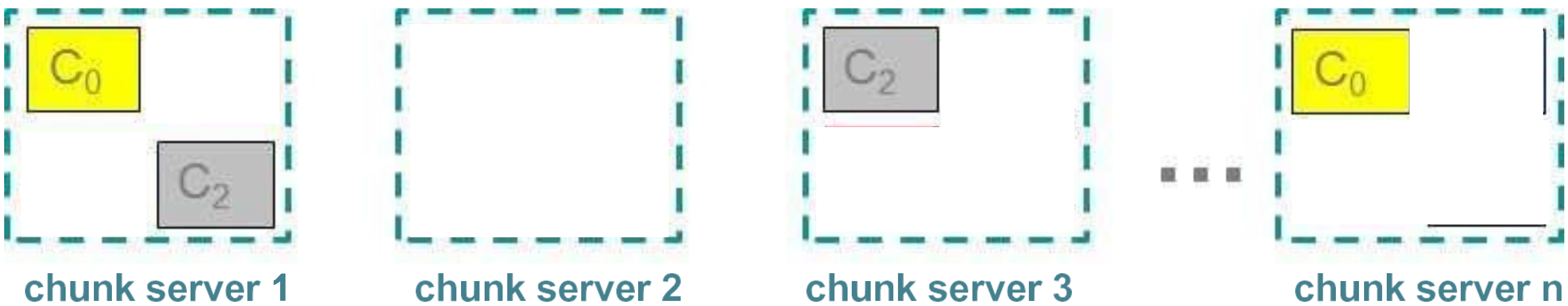


(Leskovec at al., 2014; http://www.mmds.org/)

Distributed File System

(e.g. Apache HadoopDFS, GoogleFS, EMRFS)

C, D: Two different files

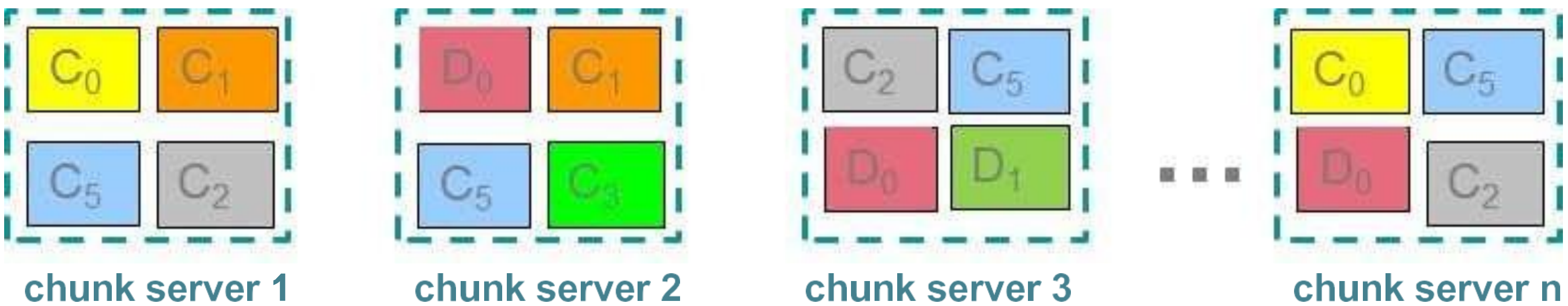


(Leskovec at al., 2014; http://www.mmds.org/)

Distributed File System

(e.g. Apache HadoopDFS, GoogleFS, EMRFS)

C, D: Two different files



(Leskovec at al., 2014; http://www.mmds.org/)

Components of a Distributed File System

Chunk servers (on Data Nodes)

File is split into contiguous chunks

Typically each chunk is 16-64MB

Each chunk replicated (usually 2x or 3x)

Try to keep replicas in different racks

(Leskovec at al., 2014; http://www.mmds.org/)

Components of a Distributed File System

Chunk servers (on Data Nodes)

File is split into contiguous chunks

Typically each chunk is 16-64MB

Each chunk replicated (usually 2x or 3x)

Try to keep replicas in different racks

Name node (aka master node)

Stores metadata about where files are stored

Might be replicated or distributed across data nodes.

Client library for file access

Talks to master to find chunk servers

Connects directly to chunk servers to access data

(Leskovec at al., 2014; http://www.mmds.org/)

Challenges for IO Cluster Computing

1. Nodes fail

   1 in 1000 nodes fail a day

   Duplicate Data (Distributed FS)
2. Network is a bottleneck

   Typically 1-10 Gb/s throughput

   Bring computation to nodes, rather than data to nodes.
3. Traditional distributed programming is often ad-hoc and complicated

   Stipulate a programming system that can easily be distributed

What is MapReduce?

noun.1* - A *style of programming

input chunks => map tasks | group_by keys | reduce tasks => output

"|" is the linux "pipe" symbol: passes stdout from first process to stdin of next.

E.g. counting words:

```
tokenize(document) | sort | uniq -c
```

What is MapReduce?

noun.1* - A *style of programming

input chunks => map tasks | group_by keys | reduce tasks => output

“|” is the linux “pipe” symbol: passes stdout from first process to stdin of next.

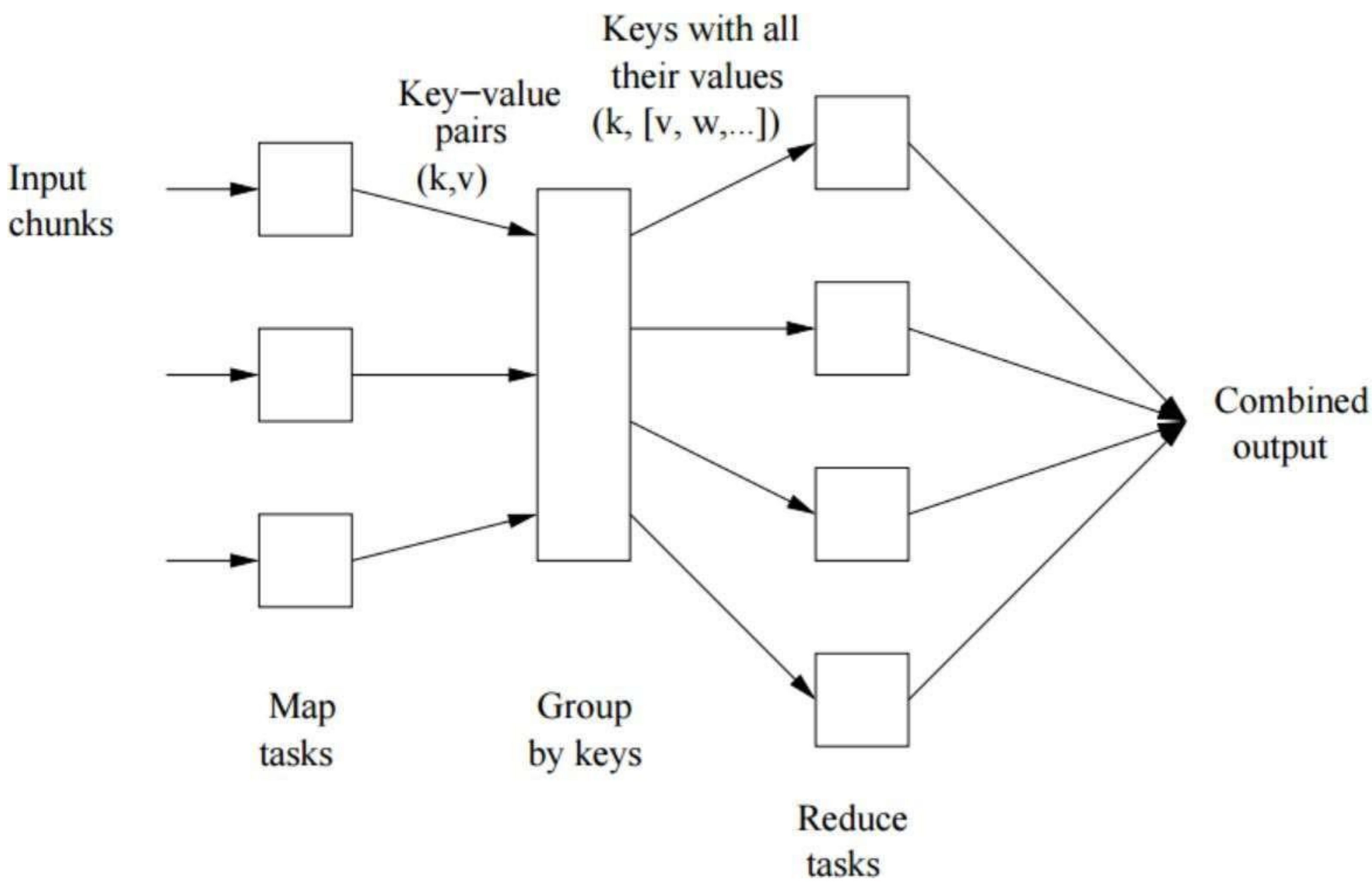
E.g. counting words:

```
tokenize(document) | sort | uniq -c
```

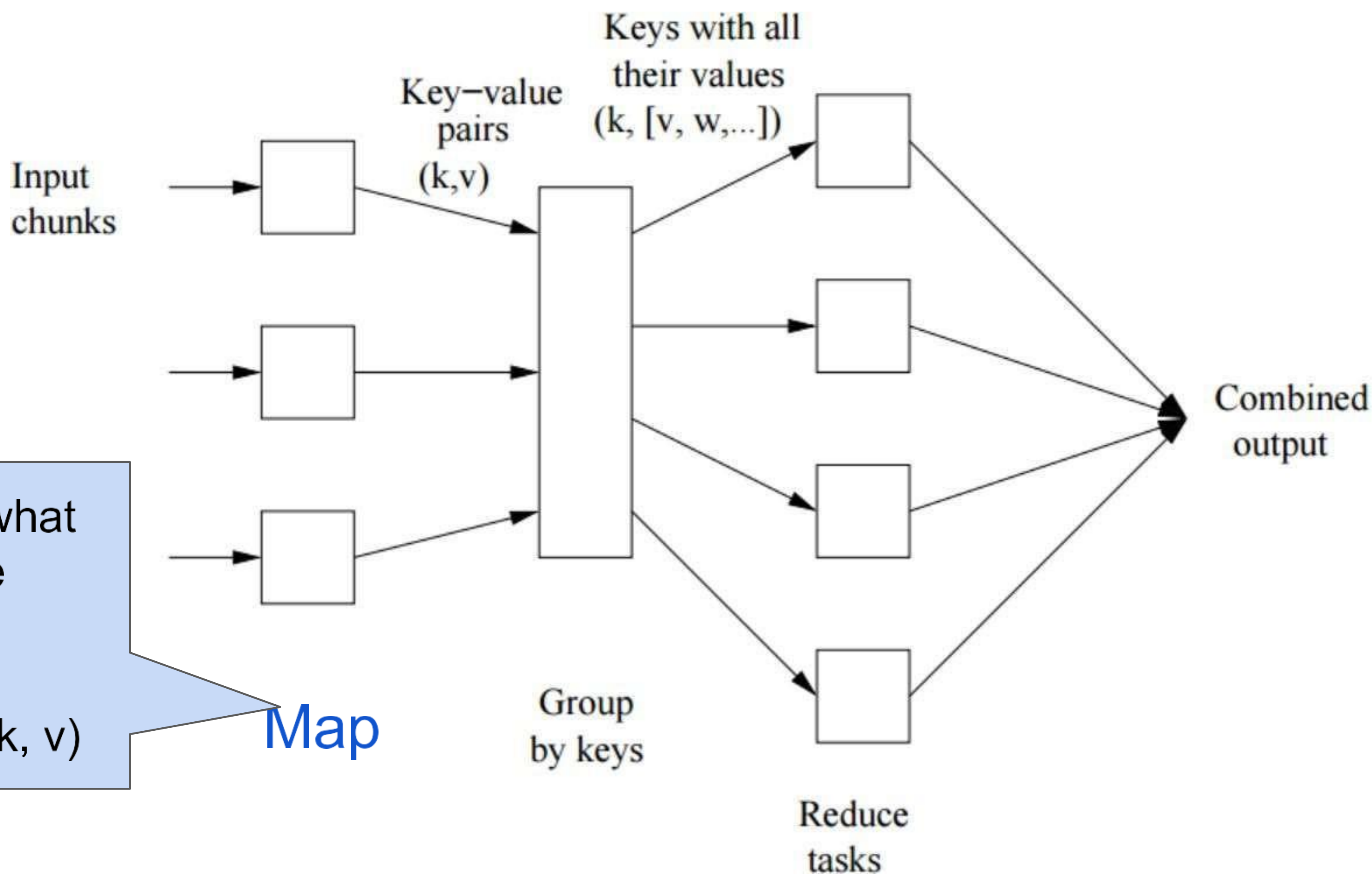
***noun.2* - A *system* that distributes MapReduce style programs across a distributed file-system.**

(e.g. Google’s internal “MapReduce” or apache.hadoop.mapreduce with hdfs)

What is MapReduce?



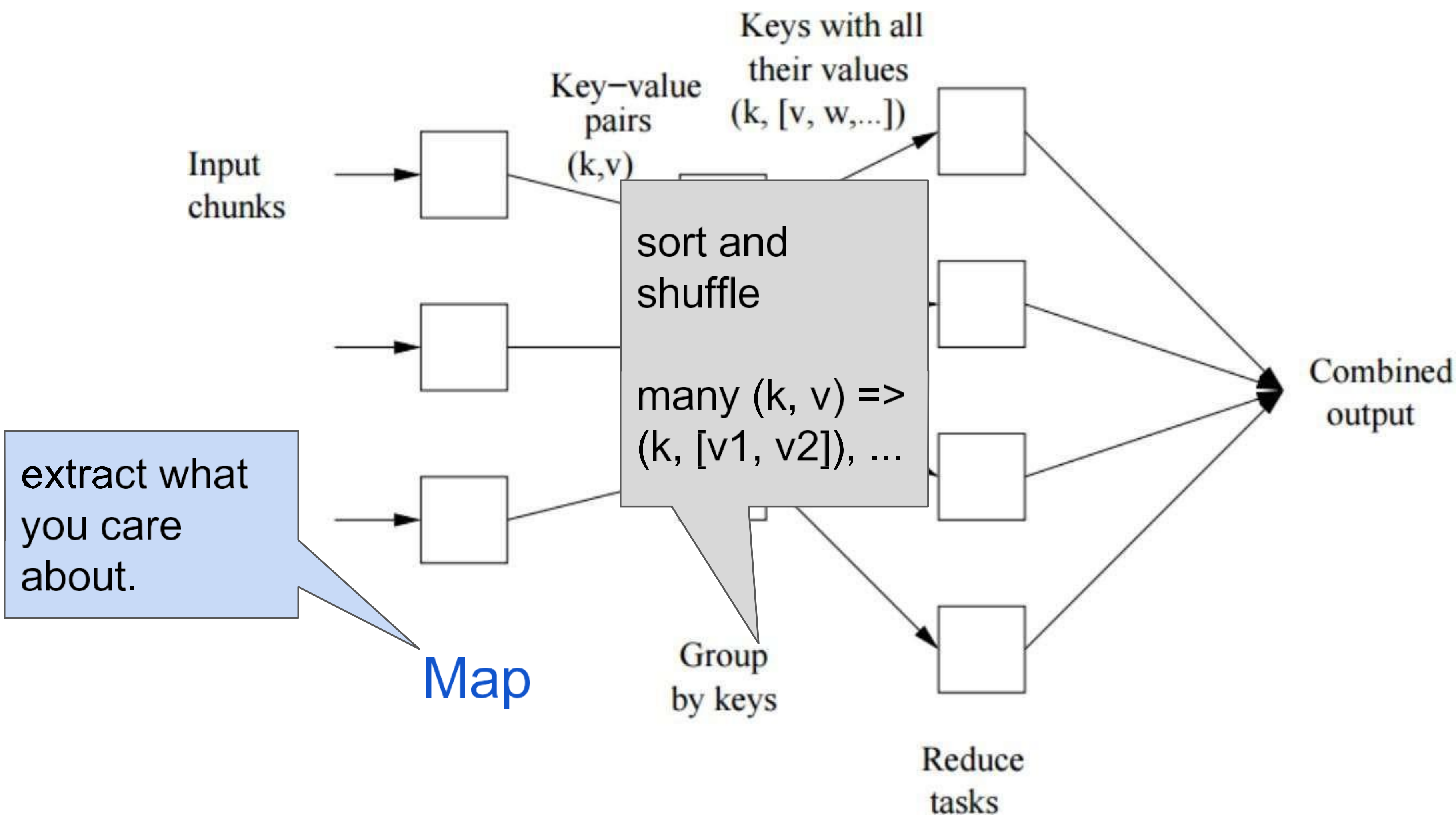
What is MapReduce?



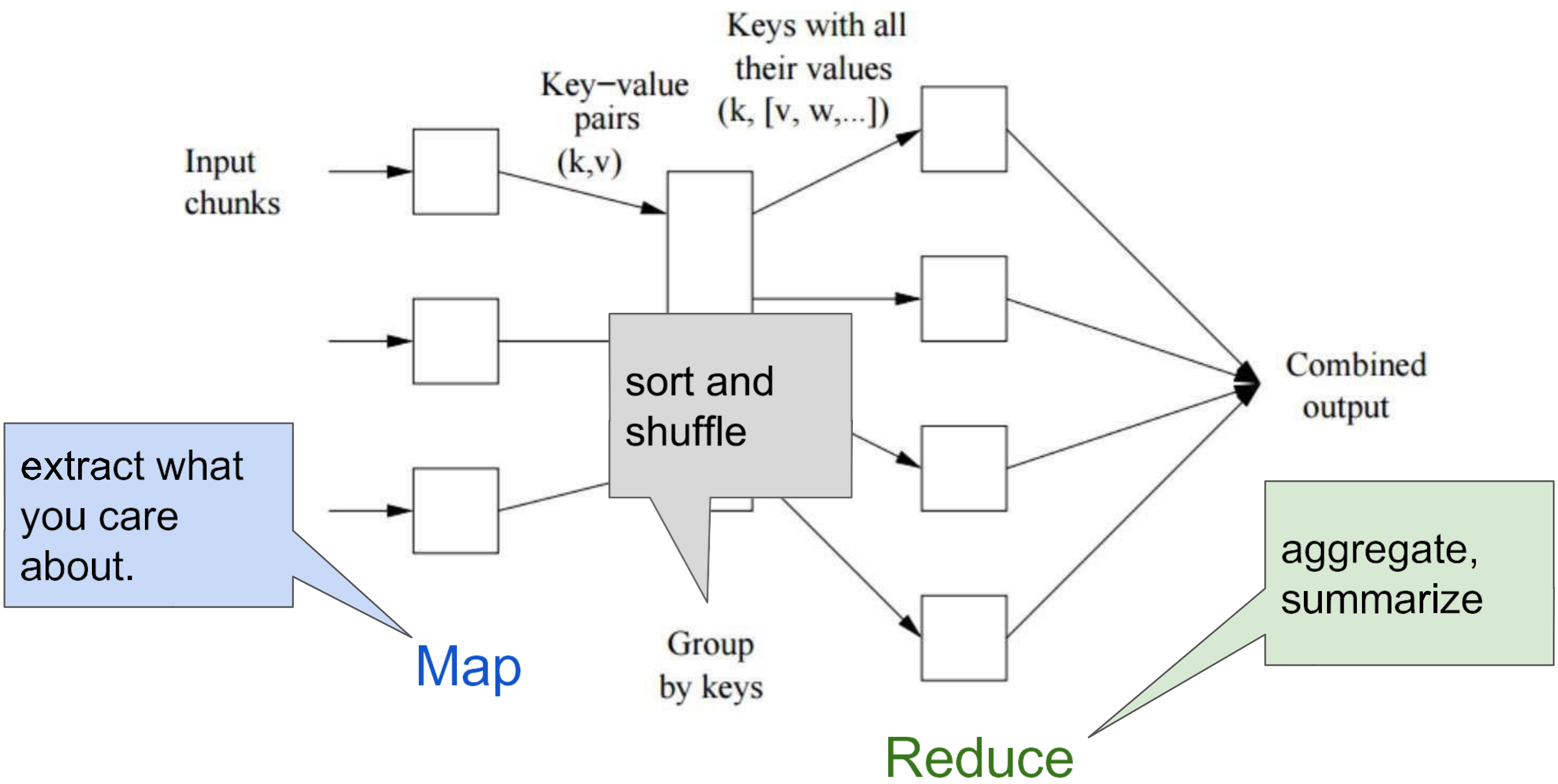
extract what you care about.

line => (k, v)

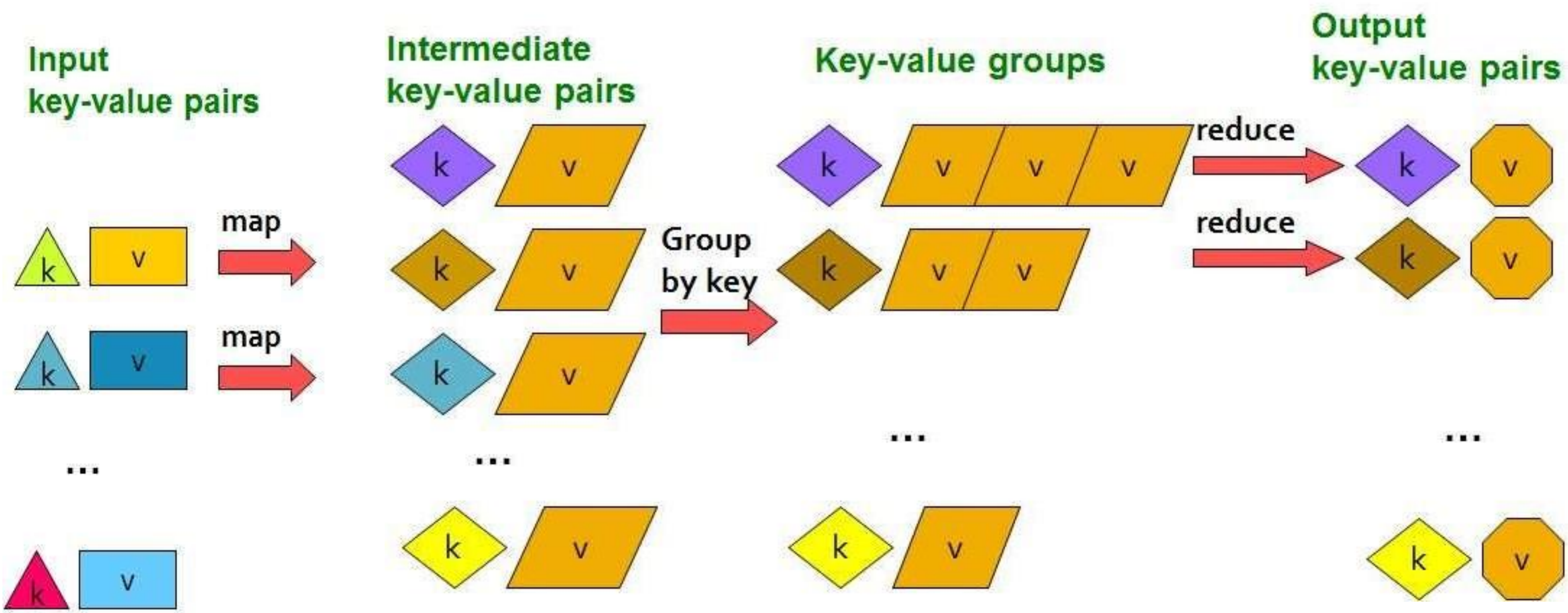
What is MapReduce?



What is MapReduce?

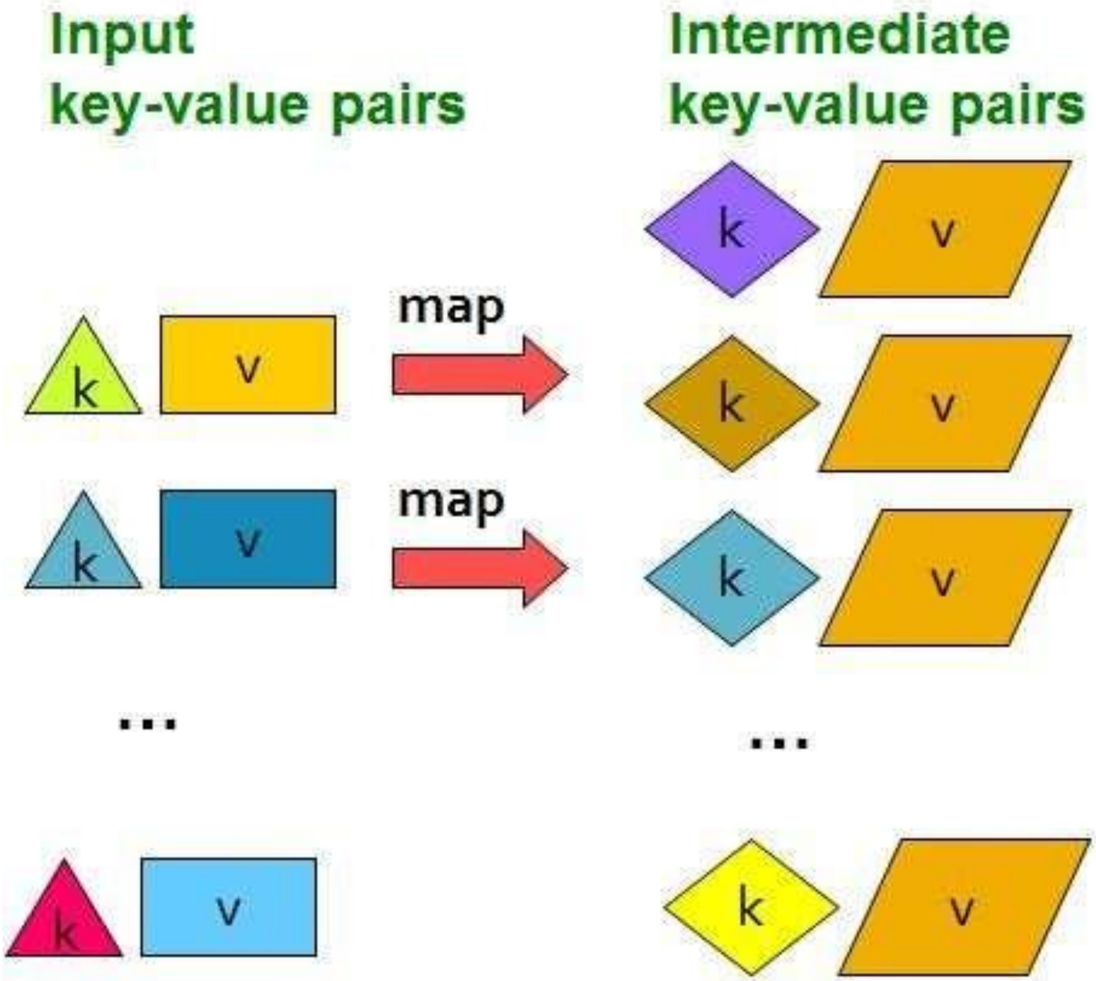


What is MapReduce?



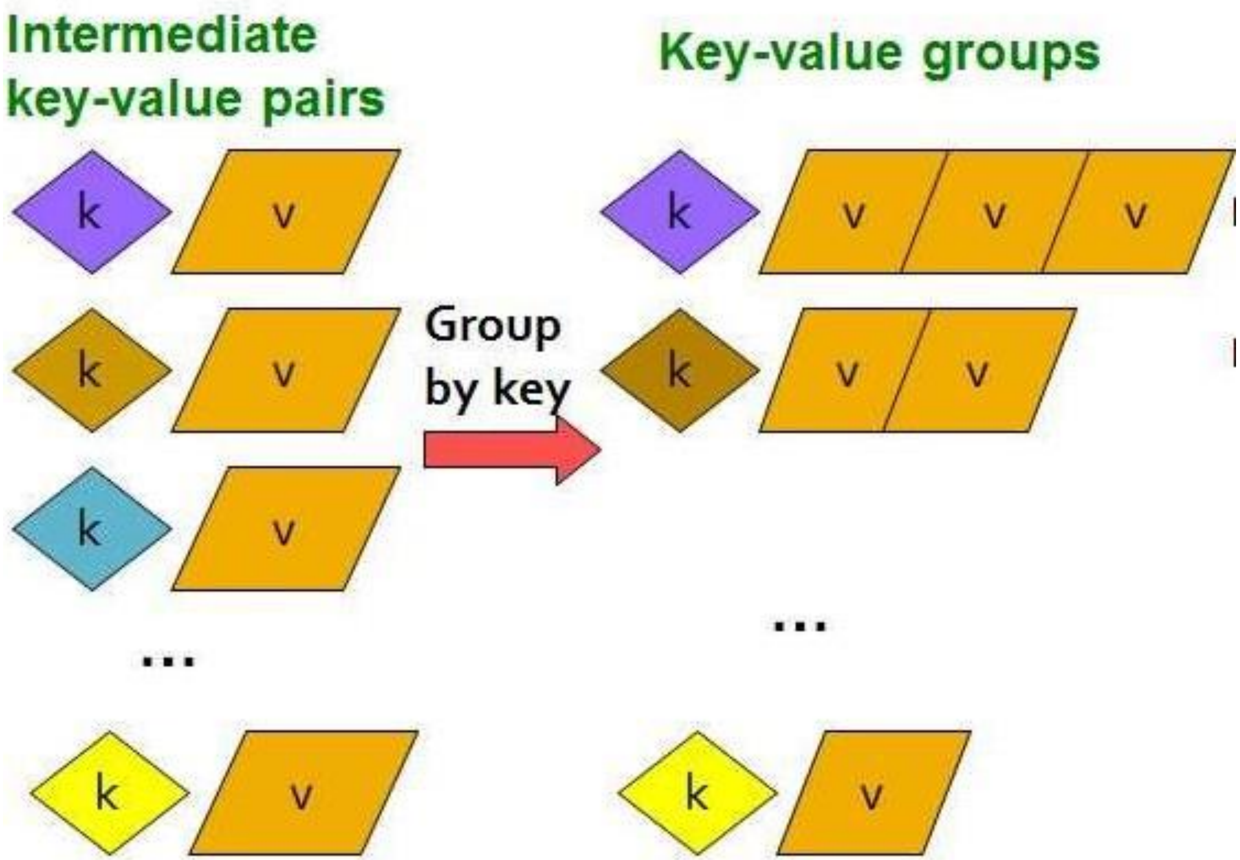
(Leskovec at al., 2014; http://www.mmds.org/)

The Map Step



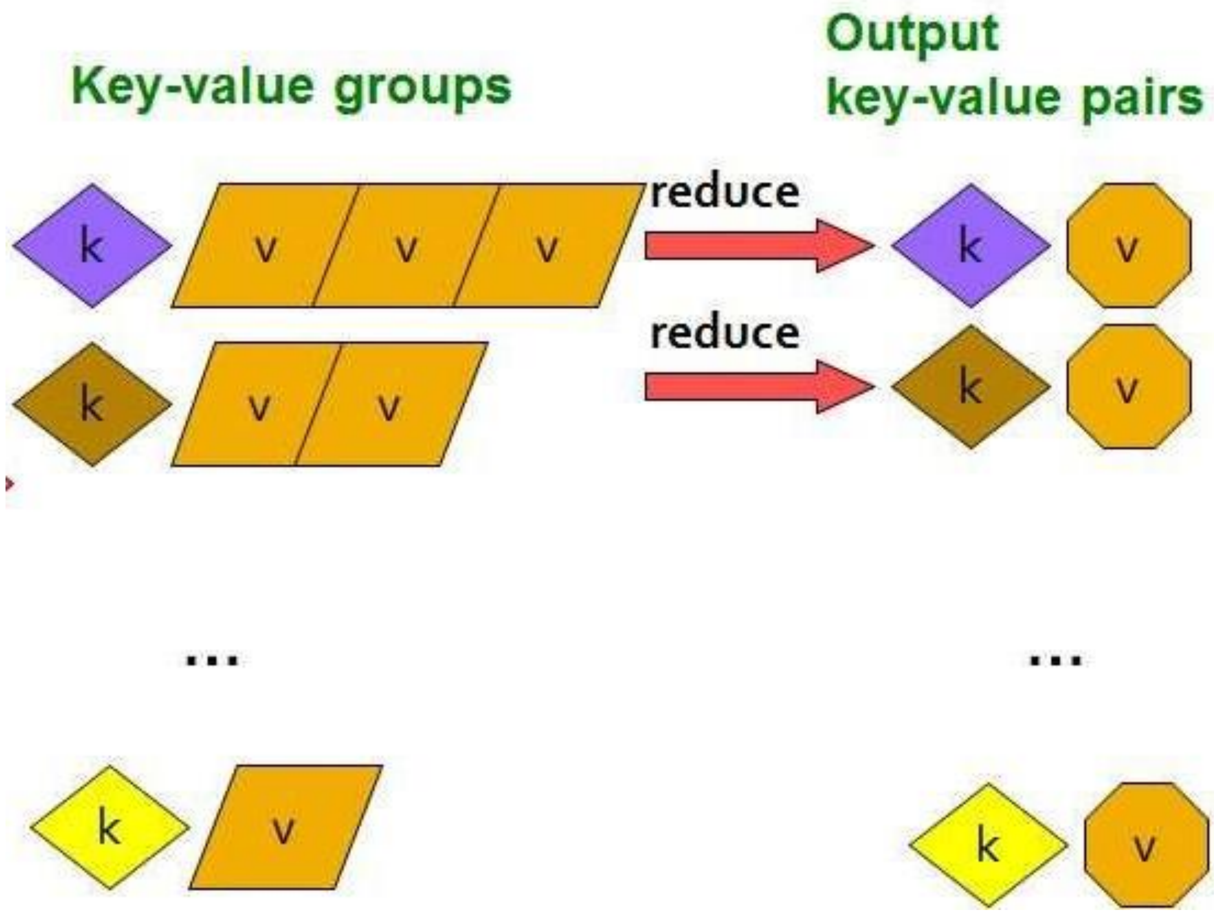
(Leskovec at al., 2014; http://www.mmds.org/)

The Sort / Group By Step



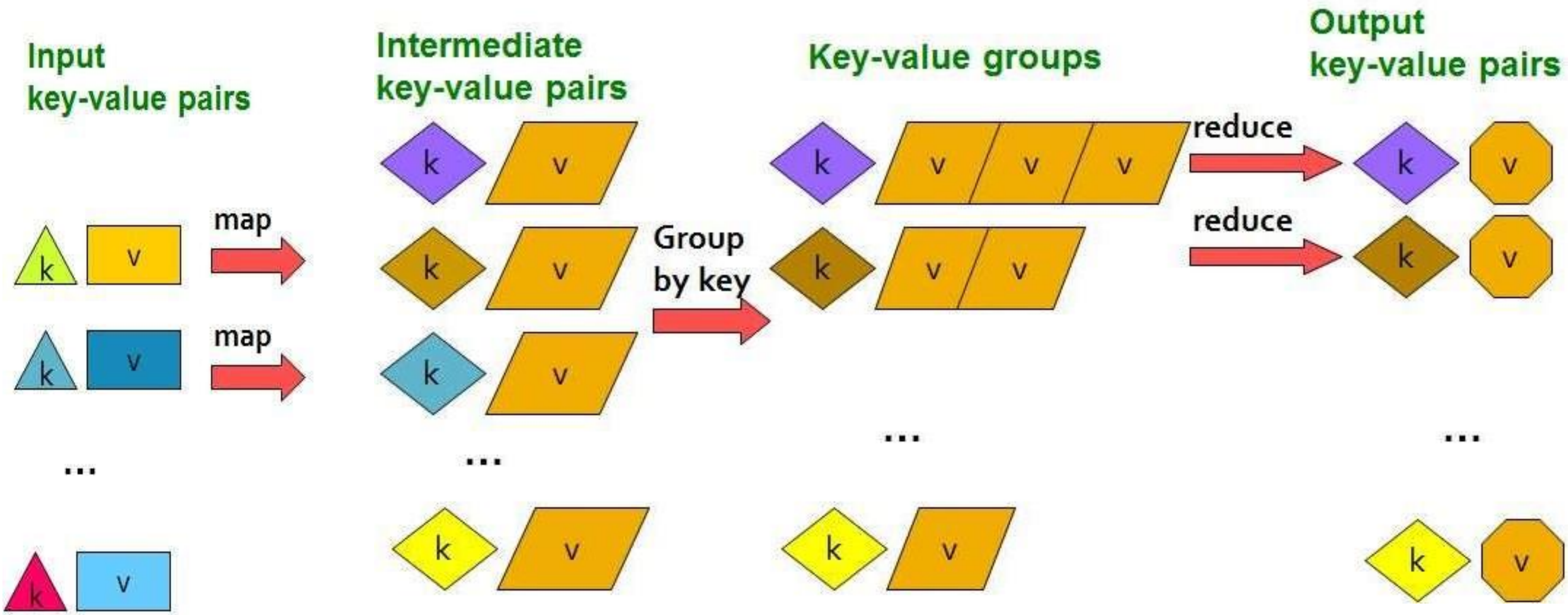
(Leskovec at al., 2014; http://www.mmds.org/)

The Reduce Step



(Leskovec at al., 2014; http://www.mmds.org/)

What is MapReduce?



(Leskovec at al., 2014; http://www.mmds.org/)

What is MapReduce?

Map: $(k,v) \rightarrow (k', v')^*$

(Written by programmer)

Group by key: $(k_1', v_1'), (k_2', v_2'), \ldots \rightarrow (k_1', (v_1', v', \ldots),$

(system handles) $(k_2', (v_1', v', \ldots), \ldots$

Reduce: $(k', (v_1', v', \ldots)) \rightarrow (k', v'')^*$

(Written by programmer)

Example: Word Count

```
tokenize(document) | sort | uniq -C
```

Example: Word Count

```
tokenize(document) | sort | uniq -C
```

Map: extract what you care about.

sort and shuffle

Reduce: aggregate, summarize

Example: Word Count

The crew of the space shuttle Endeavor recently returned to Earth as ambassadors, harbingers of a new era of space exploration. Scientists at NASA are saying that the recent assembly of the Dextre bot is the first step in a long-term space-based man/mache partnership. '"The work we're doing now -- the robotics we're doing -- is what we're going to need ....

Big document

(Leskovec at al., 2014; http://www.mmds.org/)

Provided by the programmer

MAP:
Read input and produces a set of key-value pairs

The crew of the space shuttle Endeavor recently returned to Earth as ambassadors, harbingers of a new era of space exploration. Scientists at NASA are saying that the recent assembly of the Dextre bot is the first step in a long-term space-based man/mache partnership. '"The work we're doing now -- the robotics we're doing -- is what we're going to need ……………………..

(The, 1)
(crew, 1)
(of, 1)
(the, 1)
(space, 1)
(shuttle, 1)
(Endeavor, 1)
(recently, 1)
….

Big document

(key, value)

Provided by the programmer

MAP:
Read input and produces a set of key-value pairs

Group by key:
Collect all pairs with same key

The crew of the space shuttle Endeavor recently returned to Earth as ambassadors, harbingers of a new era of space exploration. Scientists at NASA are saying that the recent assembly of the Dextre bot is the first step in a long-term space-based man/mache partnership. '"The work we're doing now -- the robotics we're doing -- is what we're going to need ……………………..

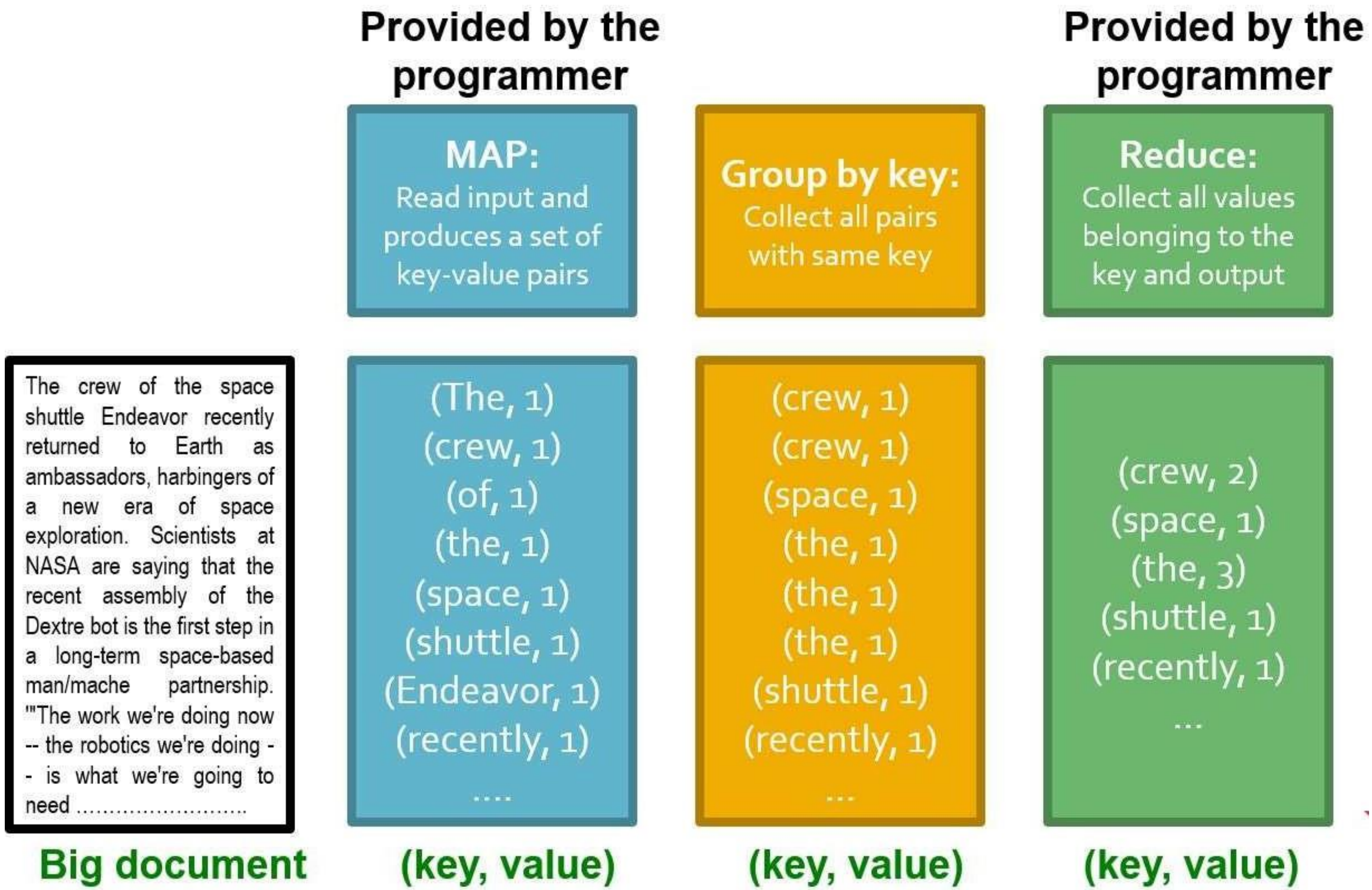
Big document

(The, 1)
(crew, 1)
(of, 1)
(the, 1)
(space, 1)
(shuttle, 1)
(Endeavor, 1)
(recently, 1)
….

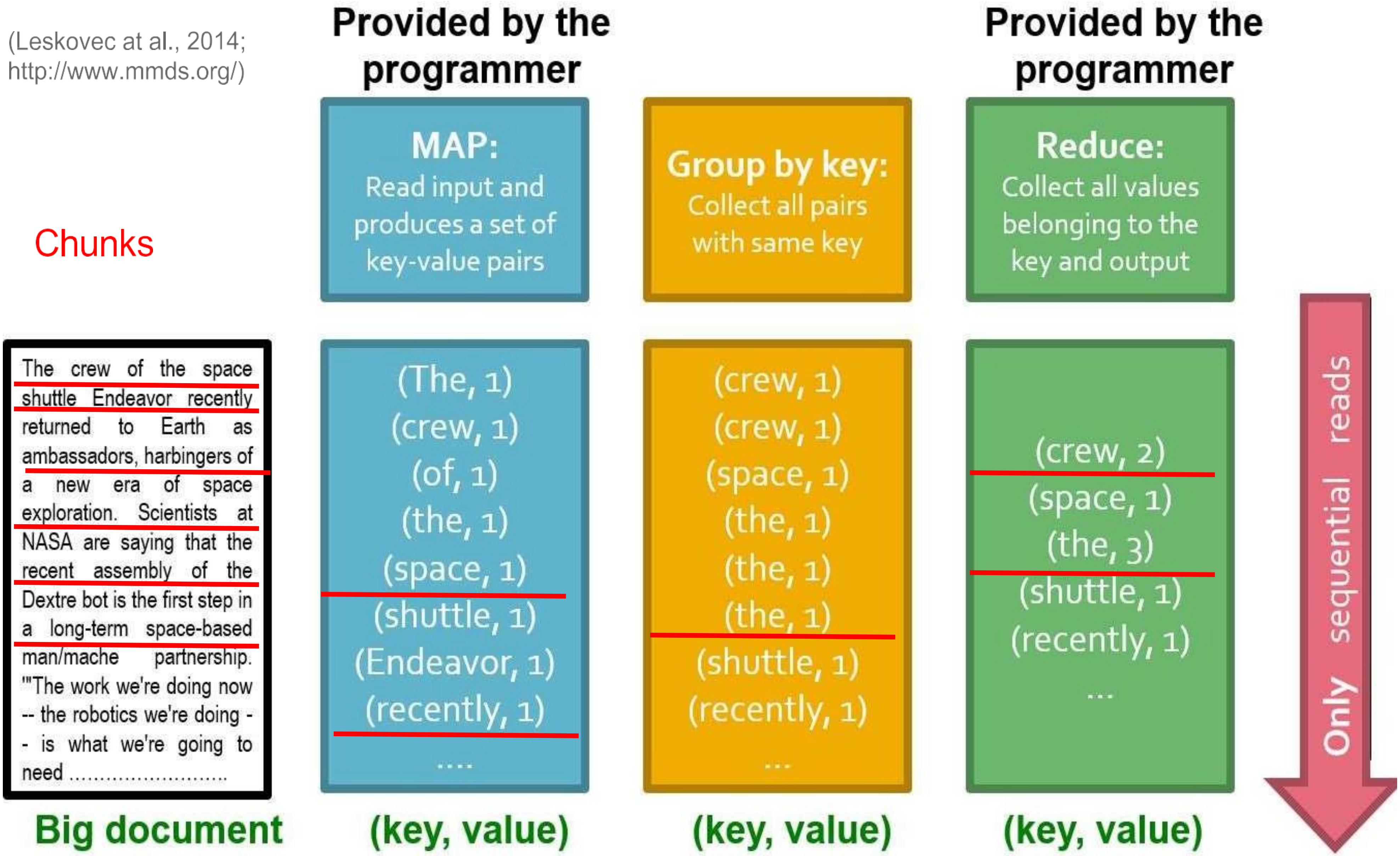
(key, value)

(crew, 1)
(crew, 1)
(space, 1)
(the, 1)
(the, 1)
(the, 1)
(shuttle, 1)
(recently, 1)
…

(key, value)


Provided by the programmer
Provided by the programmer
MAP:
Read input and produces a set of key-value pairs
Group by key:
Collect all pairs with same key
Reduce:
Collect all values belonging to the key and output
The crew of the space shuttle Endeavor recently returned to Earth as ambassadors, harbingers of a new era of space exploration. Scientists at NASA are saying that the recent assembly of the Dextre bot is the first step in a long-term space-based man/mache partnership. '"The work we're doing now -- the robotics we're doing -- is what we're going to need ....
(The, 1)
(crew, 1)
(of, 1)
(the, 1)
(space, 1)
(shuttle, 1)
(Endeavor, 1)
(recently, 1)
....
(crew, 1)
(crew, 1)
(space, 1)
(the, 1)
(the, 1)
(the, 1)
(shuttle, 1)
(recently, 1)
...
(crew, 2)
(space, 1)
(the, 3)
(shuttle, 1)
(recently, 1)
...
Big document
(key, value)
(key, value)
(key, value)

(Leskovec at al., 2014; http://www.mmds.org/)



Example: Word Count

```
@abstractmethod
def map(k, v):
    pass
```

```
@abstractmethod
def reduce(k, vs):
    pass
```

Example: Word Count (version 1)

```
def map(k, v):
    for w in tokenize(v):
        yield (w,1)
```

```
def reduce(k, vs):
    return len(vs)
```

Example: Word Count (version 1)

```
def map(k, v):
    for w in tokenize(v):
        yield (w,1)
```

```
def tokenize(s):
    #simple version
    return s.split(‘ ‘)
```

```
def reduce(k, vs):
    return len(vs)
```

Example: Word Count (version 2)

```
def map(k, v):
    counts = dict()
    for w in tokenize(v):
        try:
            counts[w] += 1
        except KeyError:
            counts[w] = 1
    for item in counts.iteritems():
        yield item
```

counts each word within the chunk
(try/except is faster than
“if w in counts”)

```
def reduce(k, vs):
    return sum(vs)
```

sum of counts from different chunks

Challenges for IO Cluster Computing

1. Nodes fail
   1 in 1000 nodes fail a day
   Duplicate Data (Distributed FS)
2. Network is a bottleneck
   Typically 1-10 Gb/s throughput (Sort & Shuffle)
   Bring computation to nodes, rather than data to nodes.
3. Traditional distributed programming is often ad-hoc and complicated
   Stipulate a programming system that can easily be distributed

Challenges for IO Cluster Computing

1. Nodes fail
   1 in 1000 nodes fail a day
   Duplicate Data (Distributed FS)
2. Network is a bottleneck
   Typically 1-10 Gb/s throughput (Sort & Shuffle)
   Bring computation to nodes, rather than data to nodes.
3. Traditional distributed programming is often ad-hoc and complicated (Simply requires Mapper and Reducer)
   Stipulate a programming system that can easily be distributed

Example: Relational Algebra

Select

Project

Union, Intersection, Difference

Natural Join

Grouping

Example: Relational Algebra

Select

Project

Union, Intersection, Difference

Natural Join

Grouping

Example: Relational Algebra

Select

$R(A_1,A_2,A_3,...)$, Relation *R*, Attributes A_*

return only those attribute tuples where condition *C* is true

Example: Relational Algebra

Select

R(A₁,A₂,A₃,...), Relation *R*, Attributes *A*₊

return only those attribute tuples where condition *C* is true

```
def map(k, v): #v is list of attribute tuples
    for t in v:
        if t satisfies C:
            yield (t, t)

def reduce(k, vs):
    For each v in vs:
        yield  (k, v)
```

Example: Relational Algebra

Natural Join

Given R_1 and R_2 return R_{join} -- union of all pairs of tuples that match given attributes.

Example: Relational Algebra

Natural Join

Given R_1 and R_2 return R_{join} -- union of all pairs of tuples that match given attributes.

```
def map(k, v): #k \in {R1, R2}, v is (R1=(A, B), R2=(B, C));B are matched attributes
    if k==”R1”:
        (a, b) = v
        yield (b,(R1,a))
    if k==”R2”:
        (b,c) = v
        yield (b,(R2,c))
```

Example: Relational Algebra

Natural Join

Given R_1 and R_2 return R_{join} -- union of all pairs of tuples that match given attributes.

```
def map(k, v): #k \in {R1, R2}, v is (R1=(A, B), R2=(B, C));B are matched attributes
    if k=="R1":
        (a, b) = v
        yield (b,(R1,a))
    if k=="R2":
        (b,c) = v
        yield (b,(R2,c))
```

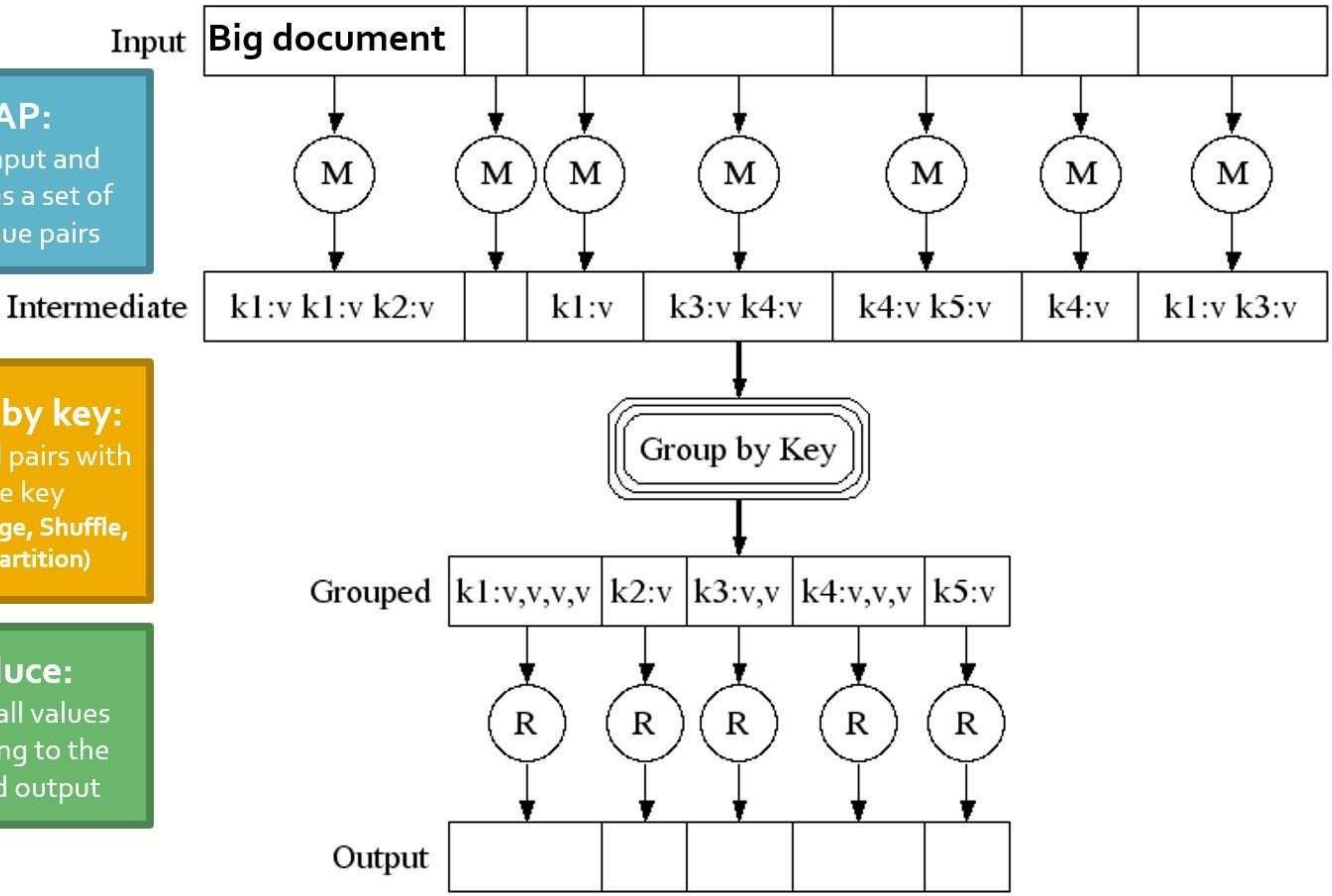
```
def reduce(k, vs):
    r1, r2 = [], []
    for (S, x) in vs: #separate rs
        if S == r1: r1.append(x)
        else: r2.append(x)
    for a in r1: #join as tuple
        for each c in r2:
            yield (Rjoin', (a, k, c)) #k is
```

Data Flow

MAP:
Read input and produces a set of key-value pairs

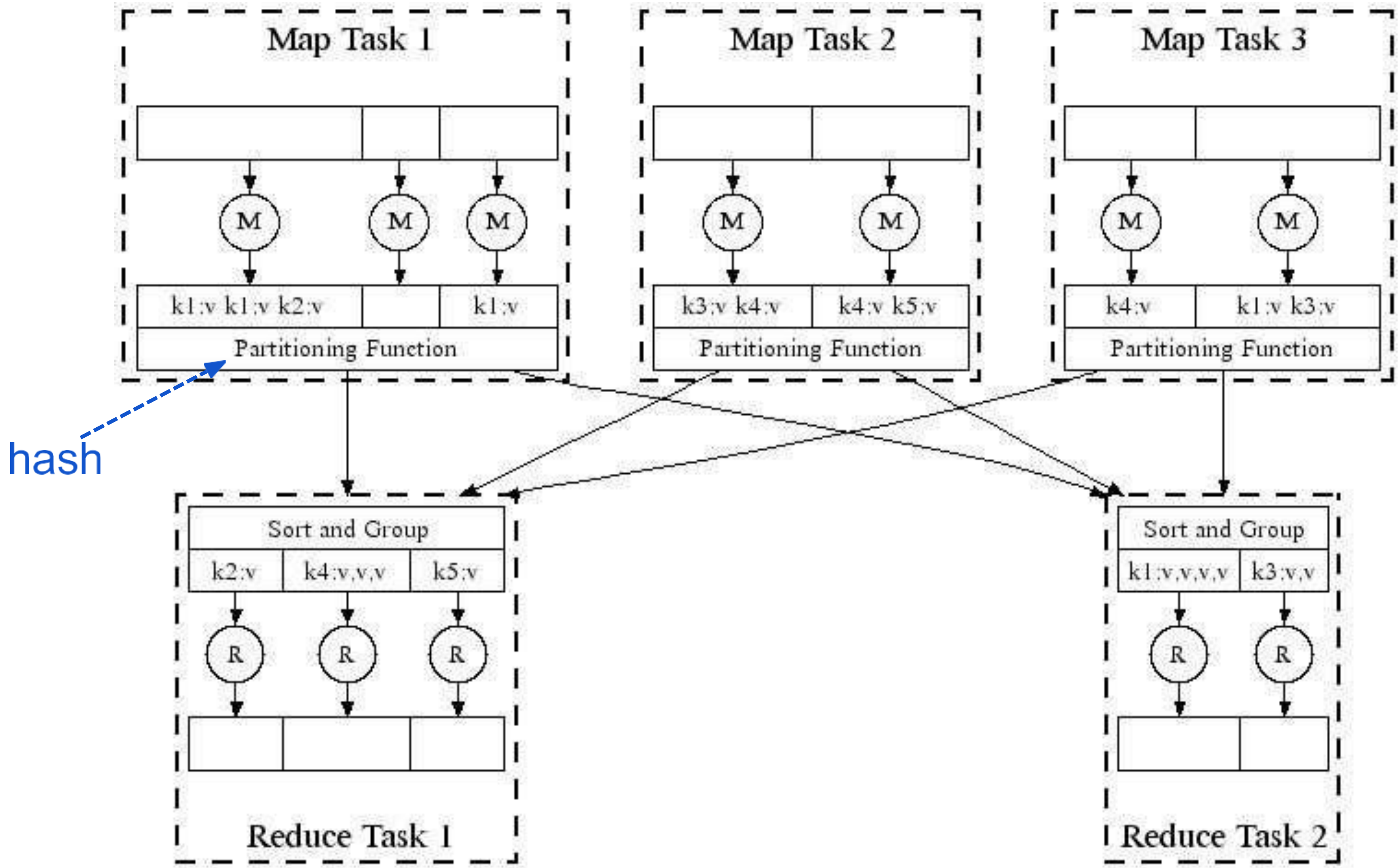
Group by key:
Collect all pairs with same key
(Hash merge, Shuffle, Sort, Partition)

Reduce:
Collect all values belonging to the key and output



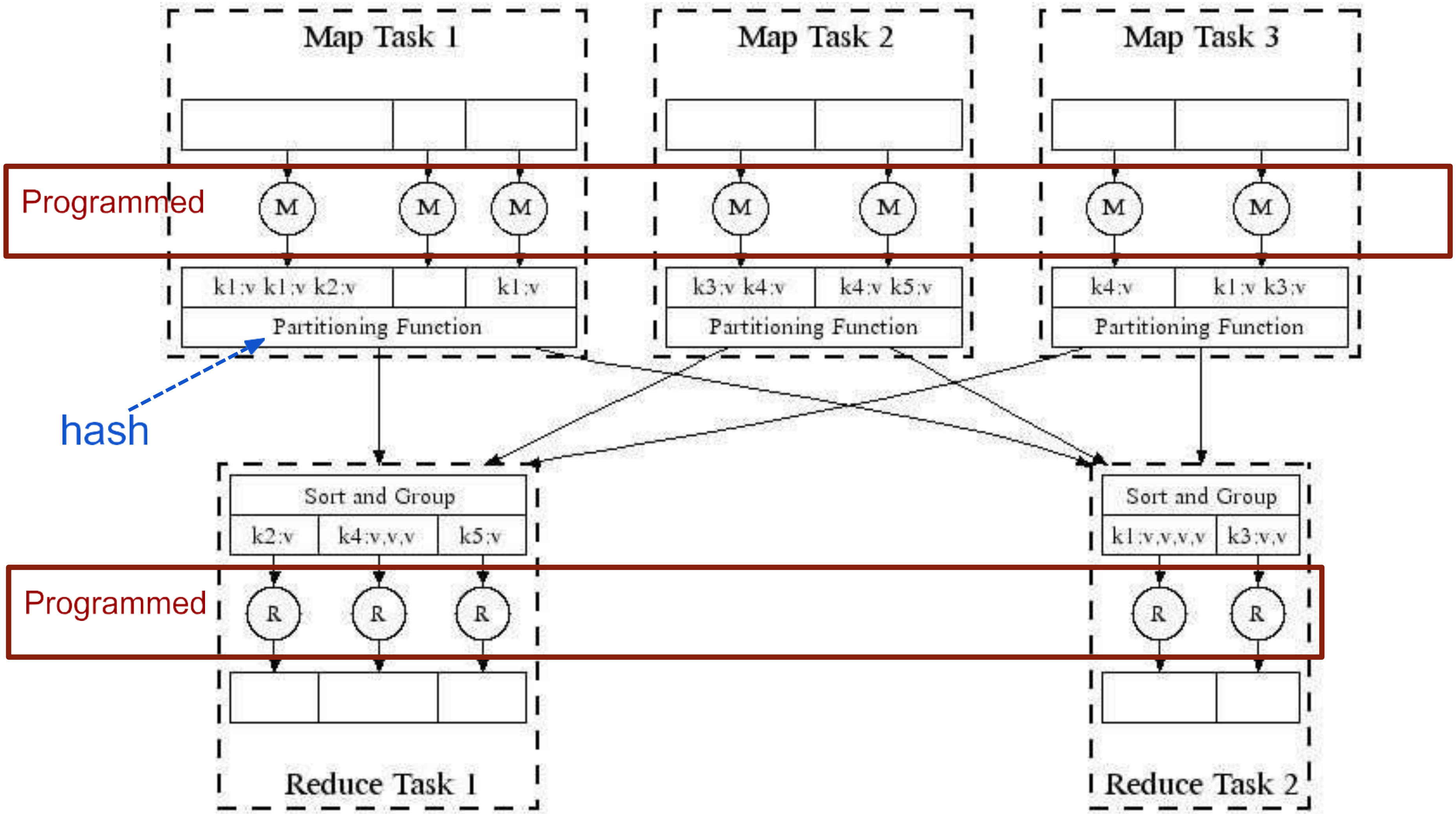
J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Data Flow: In Parallel



(Leskovec at al., 2014; http://www.mmds.org/)

Data Flow: In Parallel



(Leskovec at al., 2014; http://www.mmds.org/)

Data Flow

DFS ⇨ Map ⇨ Map’s Local FS ⇨ Reduce ⇨ DFS

Data Flow

MapReduce system handles:

- Partitioning
- Scheduling map / reducer execution
- Group by key

- Restarts from node failures
- Inter-machine communication

Data Flow

DFS ⇨ MapReduce ⇨ DFS

- Schedule map tasks near physical storage of chunk
- Intermediate results stored locally
- Master / Name Node coordinates

Data Flow

DFS → MapReduce → DFS

- Schedule map tasks near physical storage of chunk
- Intermediate results stored locally
- Master / Name Node coordinates
    - Task status: idle, in-progress, complete
    - Receives location of intermediate results and schedules with reducer
    - Checks nodes for failures and restarts when necessary
        - All map tasks on nodes must be completely restarted
        - Reduce tasks can pickup with reduce task failed

Data Flow

DFS → MapReduce → DFS

- Schedule map tasks near physical storage of chunk
- Intermediate results stored locally
- Master / Name Node coordinates
  - Task status: idle, in-progress, complete
  - Receives location of intermediate results and schedules with reducer
  - Checks nodes for failures and restarts when necessary
    - All map tasks on nodes must be completely restarted
    - Reduce tasks can pickup with reduce task failed

DFS → MapReduce → DFS → MapReduce → DFS

Data Flow

Skew: The degree to which certain tasks end up taking much longer than others.

Handled with:

- More reducers than reduce tasks
- More reduce tasks than nodes

Data Flow

Key Question: *How many Map and Reduce jobs?*

Data Flow

Key Question: *How many Map and Reduce jobs?*

M: map tasks, *R:* reducer tasks

A: If possible, one chunk per map task

and M >> |nodes| ≈≈ |cores|

(better handling of node failures, better load balancing)

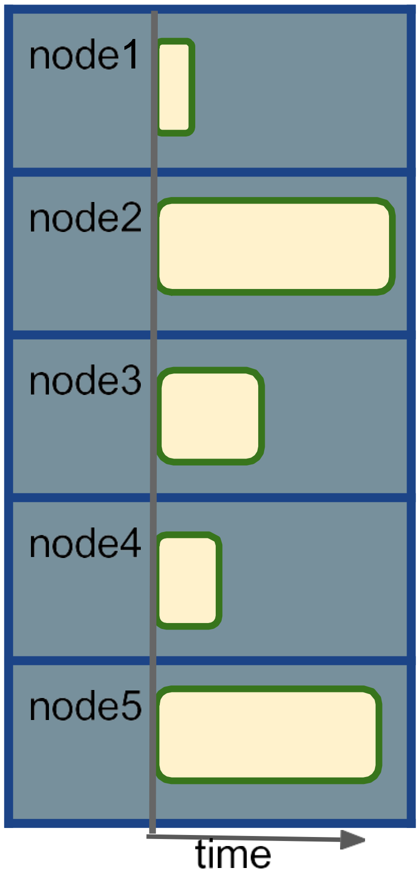
$R < M$

(reduces number of parts stored in DFS)

Data Flow

Reduce Task

version 1: few reduce tasks
(same number of reduce tasks as nodes)

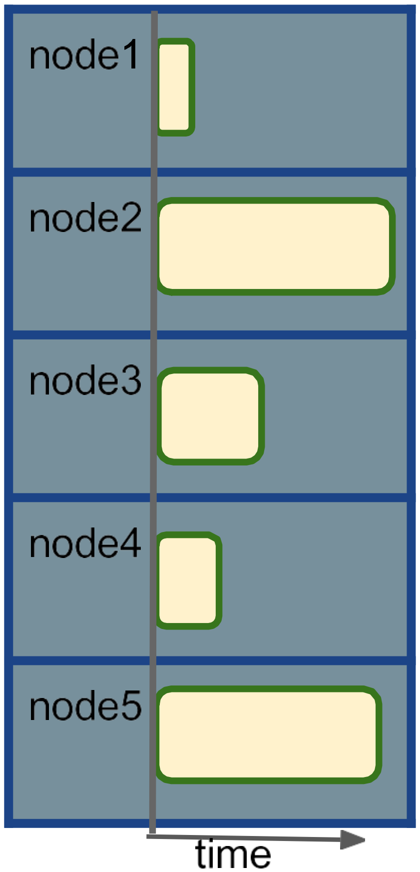


Reduce tasks represented by
time to complete task
(some tasks take much longer)

Data Flow

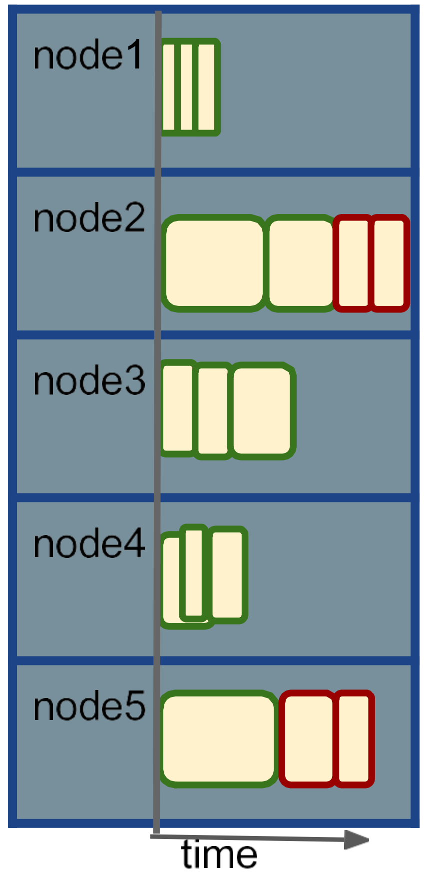
Reduce Task

version 1: few reduce tasks
(same number of reduce tasks as nodes)



Reduce tasks represented by
time to complete task
(some tasks take much longer)

version 2: more reduce tasks
(more reduce tasks than nodes)



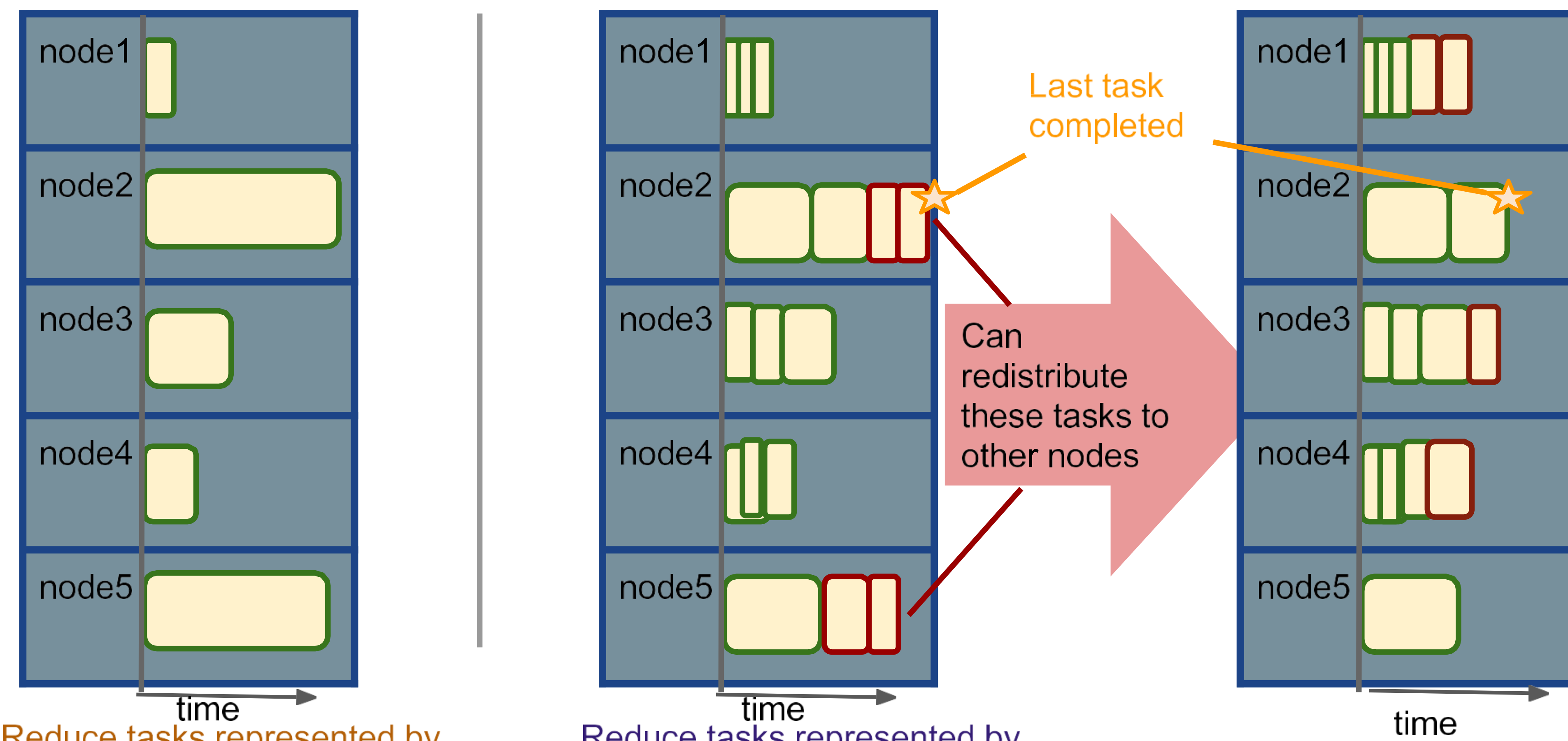
Reduce tasks represented by
time to complete task
(some tasks take much longer)

Data Flow

Reduce Task

version 1: few reduce tasks
(same number of reduce tasks as nodes)

version 2: more reduce tasks
(more reduce tasks than nodes)



Reduce tasks represented by
time to complete task
(some tasks take much longer)

Reduce tasks represented by
time to complete task
(some tasks take much longer)

(the last task now completes much earlier)

Communication Cost Model

How to assess performance?

(1) Computation: Map + Reduce + System Tasks

(2) Communication: Moving (key, value) pairs

Communication Cost Model

How to assess performance?

(1) Computation: Map + Reduce + System Tasks

(2) Communication: Moving (key, value) pairs

Ultimate Goal: wall-clock Time.

Communication Cost Model

How to assess performance?

(1) Computation: Map + Reduce + System Tasks

- Mappers and reducers often single pass O(n) within node
- System: sort the keys is usually most expensive
- Even if map executes on same node, disk read usually dominates
- In any case, can add more nodes

(2) Communication: Moving key, value pairs

Ultimate Goal: wall-clock Time.

Communication Cost Model

How to assess performance?

(1) Computation: Map + Reduce + System Tasks

(2) Communication: Moving key, value pairs

Often dominates computation.

- Connection speeds: 1-10 giga**bits** per sec;
- HD read: 50-150 giga**bytes** per sec
- Even reading from disk to memory typically takes longer than operating on the data.

Ultimate Goal: wall-clock Time

Communication Cost Model

How to assess performance?

Communication Cost = input size +
(sum of size of all map-to-reducer files)

(2) Communication: Moving key, value pairs

Often dominates computation.

- Connection speeds: 1-10 giga**bits** per sec;
  HD read: 50-150 giga**bytes** per sec
- Even reading from disk to memory typically takes longer than operating on the data.

Communication Cost Model

How to assess performance?

Communication Cost = input size + (sum of size of all map-to-reducer files)

(2) Communication: Moving key, value pairs

Often dominates computation.

- Connection speeds: 1-10 giga**bits** per sec;
  HD read: 50-150 giga**bytes** per sec
- Even reading from disk to memory typically takes longer than operating on the data.
- Output from reducer ignored because it's either small (finished summarizing data) or being passed to another mapreduce job.

Ultimate Goal: wall-clock Time

Example: Natural Join

R, S: Relations (Tables)   *R(A, B)* $\bowtie$ *S(B, C)*

Communication Cost = input size +
(sum of size of all map-to-reducer files)

DFS ⇨ Map ⇨ LocalFS ⇨ Network ⇨ Reduce ⇨ DFS ⇨ ?

Example: Natural Join

R, S: Relations (Tables)   *R(A, B)* ⋈ *S(B, C)*

Communication Cost = input size + (sum of size of all map-to-reducer files)

```
def map(k, v):
    if k=="R1":
        (a, b) = v
        yield (b,(R1,a))
    if k=="R2":
        (b,c) = v
        yield (b,(R2,c))
```

```
def reduce(k, vs):
    r1, r2 = [], []
    for (rel, x) in vs: #separate rs
        if rel == 'R': r1.append(x)
        else: r2.append(x)
    for a in r1: #join as tuple
        for each c in r2:
            yield (Rjoin', (a, k, c)) #k is
```

Example: Natural Join

R, S: Relations (Tables) *R(A, B)* ⋈ *S(B, C)*

Communication Cost = input size + (sum of size of all map-to-reducer files)

= |R1| + |R2| + (|R1| + |R2|)

= $O(|R1| + |R2|)$

```
def map(k, v):
    if k=="R1":
        (a, b) = v
        yield (b,(R1,a))
    if k=="R2":
        (b,c) = v
        yield (b,(R2,c))
```

```
def reduce(k, vs):
    r1, r2 = [], []
    for (rel, x) in vs: #separate rs
        if rel == 'R': r1.append(x)
        else: r2.append(x)
    for a in r1: #join as tuple
        for each c in r2:
            yield (Rjoin', (a, k, c)) #k is
```

Last Notes: Further Considerations for MapReduce

- Performance Refinements:
  - Backup tasks (aka speculative tasks)
    - Schedule multiple copies of tasks when close to the end to mitigate certain nodes running slow.
  - Combiners (like word count version 2 but done via reduce)
    - Run reduce right after map from same node before passing to reduce
    - Reduces communication cost
  - Override partition hash function

    E.g. instead of `hash(url)` use `hash(hostname(url))`

Spark

Situations where MapReduce is not efficient

DFS ⇨ Map ⇨ LocalFS ⇨ Network ⇨ Reduce ⇨ DFS ⇨ Map ⇨ ...

Situations where MapReduce is not efficient

- Long pipelines sharing data
- Interactive applications
- Streaming applications
- Iterative algorithms (optimization problems)

DFS ⇨ Map ⇨ LocalFS ⇨ Network ⇨ Reduce ⇨ DFS ⇨ Map ⇨ ...

(Anytime where MapReduce would need to write and read from disk a lot).

Situations where MapReduce is not efficient

- Long pipelines sharing data
- Interactive applications
- Streaming applications
- Iterative algorithms (optimization problems)

DFS ⇨ Map ⇨ LocalFS ⇨ Network ⇨ Reduce ⇨ DFS ⇨ Map ⇨ ...

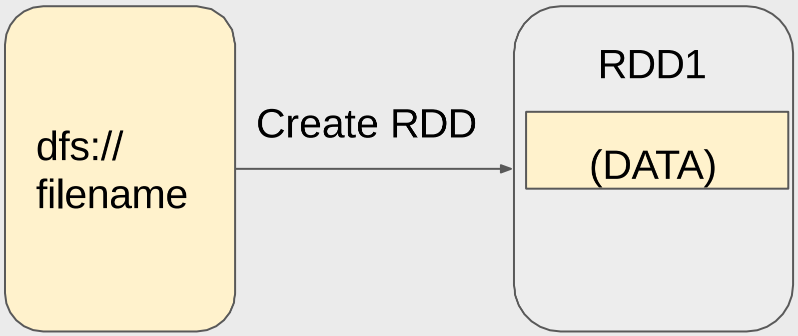
(Anytime where MapReduce would need to write and read from disk a lot).

Spark's Big Idea

- Resilient Distributed Datasets (RDDs) -- Read-only partitioned collection of records (like a DFS) but with a record of how the dataset was created as combination of *transformations* from other dataset(s).

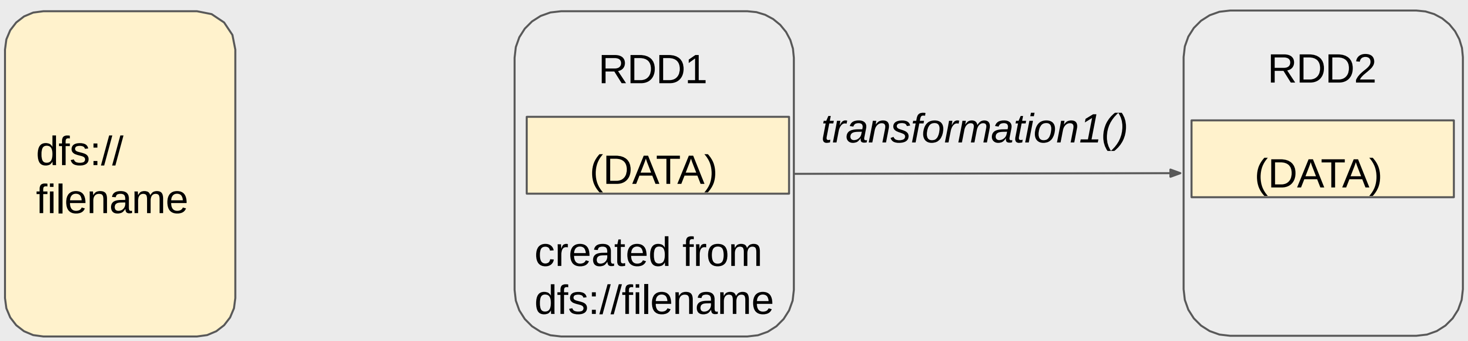
Spark’s Big Idea

- Resilient Distributed Datasets (RDDs) -- Read-only partitioned collection of records (like a DFS) but with a record of how the dataset was created as combination of *transformations* from other dataset(s).



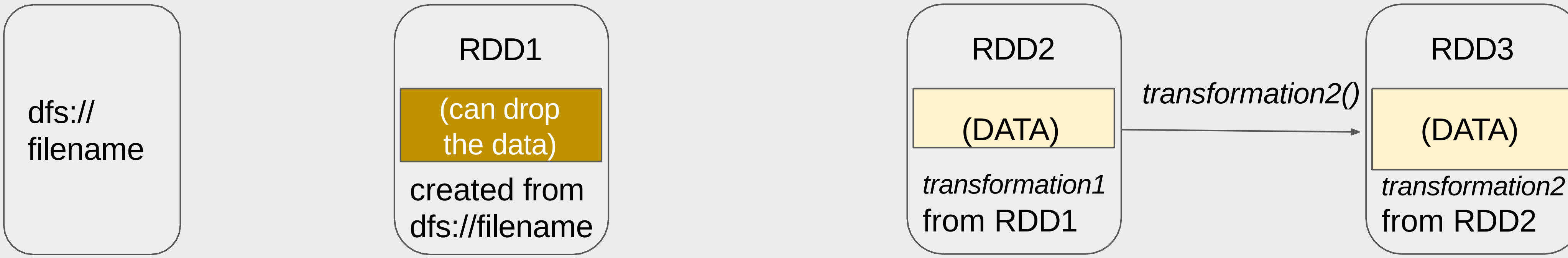
Spark's Big Idea

- Resilient Distributed Datasets (RDDs) -- Read-only partitioned collection of records (like a DFS) but with a record of how the dataset was created as combination of *transformations* from other dataset(s).



Spark's Big Idea

- Resilient Distributed Datasets (RDDs) -- Read-only partitioned collection of records (like a DFS) but with a record of how the dataset was created as combination of *transformations* from other dataset(s).



Spark's Big Idea

Resilient Distributed Datasets (RDDs) -- Read-only partitioned collection of records (like a DFS) but with a record of how the dataset was created as combination of *transformations* from other dataset(s).

- Enables rebuilding datasets on the fly.
- Intermediate datasets not stored on disk (and only in memory if needed and enough space)

⇨ Faster communication and I O

The Big Idea

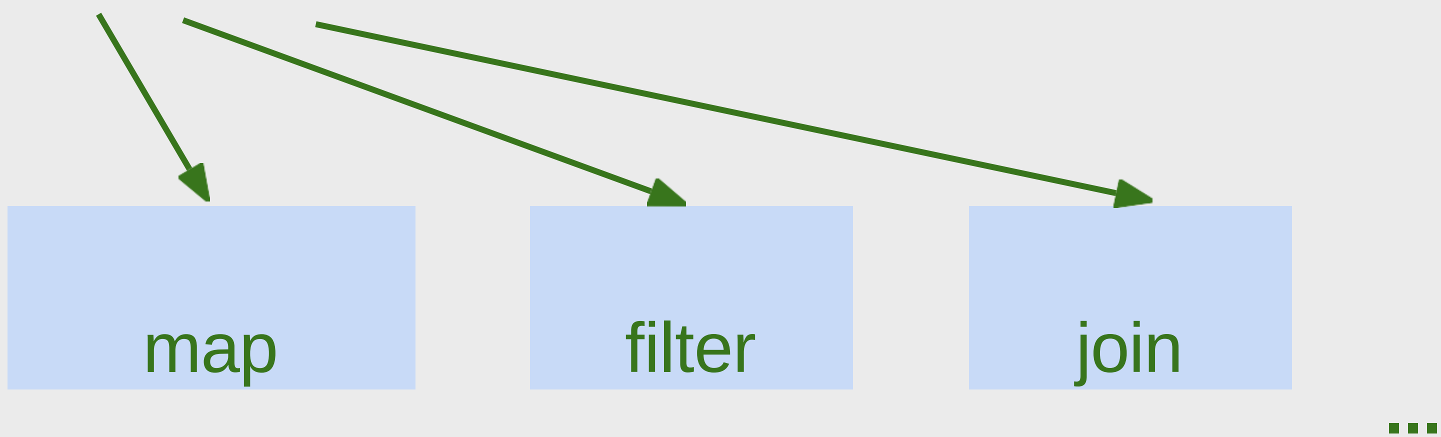
- Resilient Distributed Datasets (RDDs) -- Read-only partitioned collection of records (like a DFS) but with a record of how the dataset was created as combination of *transformations* from other dataset(s).

“Stable Storage”

Other RDDs

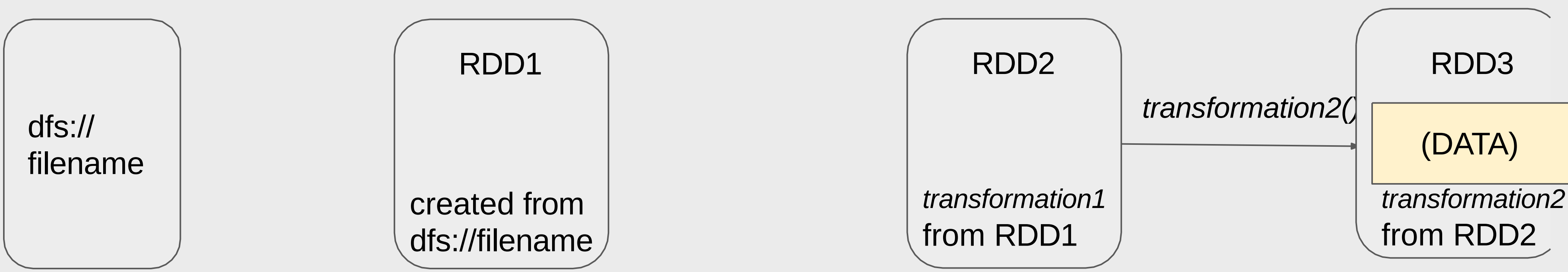
The Big Idea

- Resilient Distributed Datasets (RDDs) -- Read-only partitioned collection of records (like a DFS) but with a record of how the dataset was created as combination of *transformations* from other dataset(s).



Spark's Big Idea

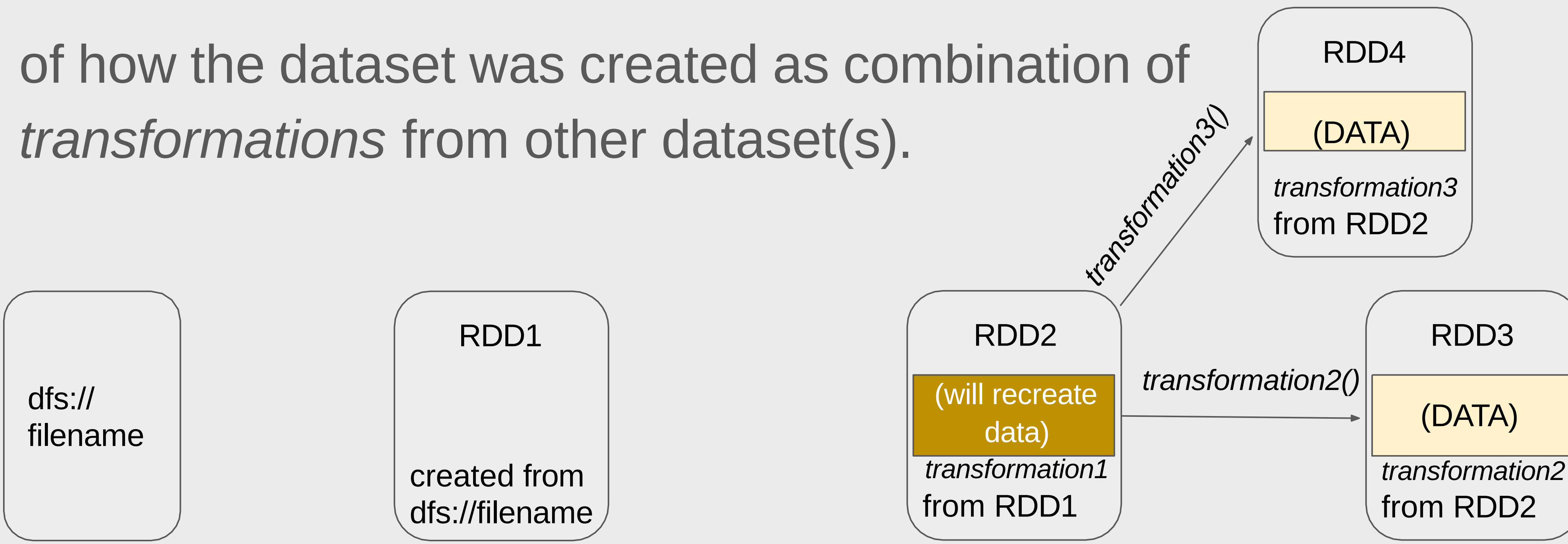
- **Resilient Distributed Datasets (RDDs)** -- Read-only partitioned collection of records (like a DFS) but with a record of how the dataset was created as combination of *transformations* from other dataset(s).



Spark's Big Idea

Resilient Distributed Datasets (RDDs) -- Read-only partitioned collection of records (like a DFS) but with a record

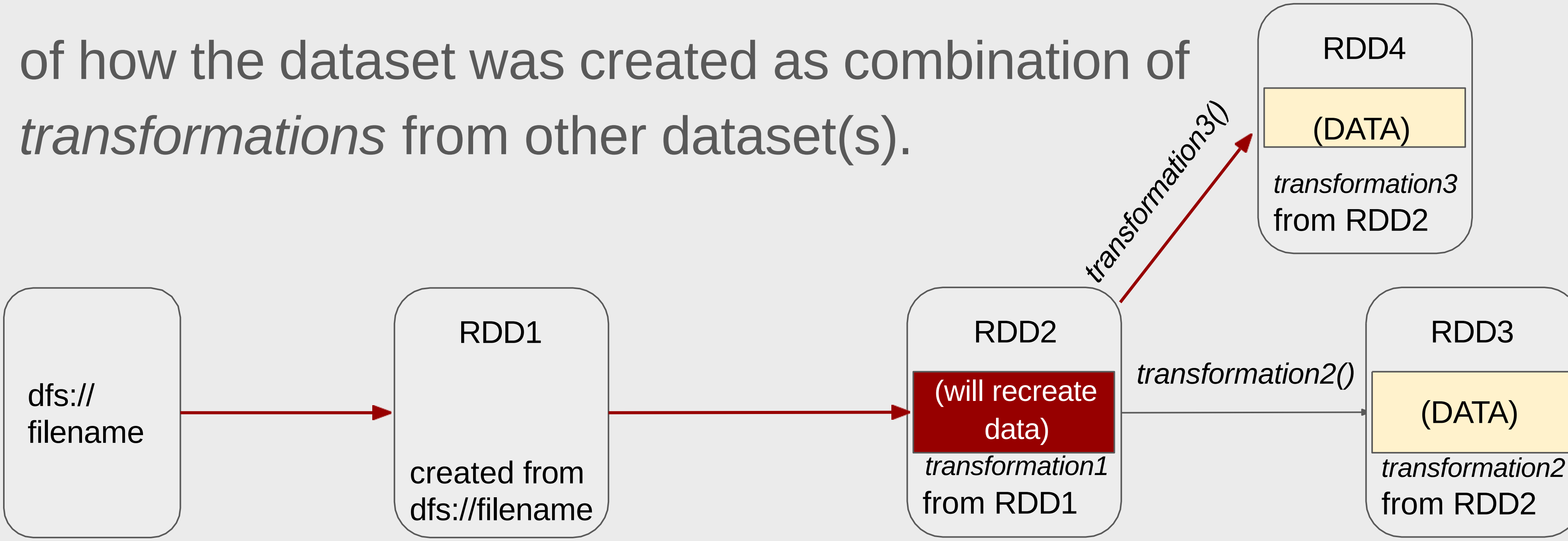
of how the dataset was created as combination of *transformations* from other dataset(s).



Spark's Big Idea

Resilient Distributed Datasets (RDDs) -- Read-only partitioned collection of records (like a DFS) but with a record

of how the dataset was created as combination of *transformations* from other dataset(s).



Original Transformations: RDD to RDD

Transformations	$map(f : \mathrm{T} \Rightarrow \mathrm{U})$	:	$\mathrm{RDD[T]} \Rightarrow \mathrm{RDD[U]}$
	$filter(f : \mathrm{T} \Rightarrow \mathrm{Bool})$	:	$\mathrm{RDD[T]} \Rightarrow \mathrm{RDD[T]}$
	$flatMap(f : \mathrm{T} \Rightarrow \mathrm{Seq[U]})$	:	$\mathrm{RDD[T]} \Rightarrow \mathrm{RDD[U]}$
	$sample(fraction : \mathrm{Float})$	:	$\mathrm{RDD[T]} \Rightarrow \mathrm{RDD[T]}$ (Deterministic sampling)
	$groupByKey()$	:	$\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, Seq[V])]}$
	$reduceByKey(f : (\mathrm{V}, \mathrm{V}) \Rightarrow \mathrm{V})$	:	$\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, V)]}$
	$union()$	:	$(\mathrm{RDD[T]}, \mathrm{RDD[T]}) \Rightarrow \mathrm{RDD[T]}$
	$join()$	:	$(\mathrm{RDD[(K, V)]}, \mathrm{RDD[(K, W)]}) \Rightarrow \mathrm{RDD[(K, (V, W))]}$
	$cogroup()$	:	$(\mathrm{RDD[(K, V)]}, \mathrm{RDD[(K, W)]}) \Rightarrow \mathrm{RDD[(K, (Seq[V], Seq[W]))]}$
	$crossProduct()$	:	$(\mathrm{RDD[T]}, \mathrm{RDD[U]}) \Rightarrow \mathrm{RDD[(T, U)]}$
	$mapValues(f : \mathrm{V} \Rightarrow \mathrm{W})$	:	$\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, W)]}$ (Preserves partitioning)
	$sort(c : \mathrm{Comparator[K]})$	:	$\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, V)]}$
	$partitionBy(p : \mathrm{Partitioner[K]})$	:	$\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, V)]}$

Table 2: Transformations and actions available on RDDs in Spark. Seq[T] denotes a sequence of elements of type T.

Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, Ion Stoica. [“Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing.”.] *NSDI 2012*. April 2012.

Original Transformations: RDD to RDD

Transformations	$map(f : \mathrm{T} \Rightarrow \mathrm{U})$	$\mathrm{RDD[T]} \Rightarrow \mathrm{RDD[U]}$
	$filter(f : \mathrm{T} \Rightarrow \mathrm{Bool})$	$\mathrm{RDD[T]} \Rightarrow \mathrm{RDD[T]}$
	$flatMap(f : \mathrm{T} \Rightarrow \mathrm{Seq[U]})$	$\mathrm{RDD[T]} \Rightarrow \mathrm{RDD[U]}$
	$sample(fraction : \mathrm{Float})$	$\mathrm{RDD[T]} \Rightarrow \mathrm{RDD[T]}$ (Deterministic sampling)
	$groupByKey()$	$\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, Seq[V])]}$
	$reduceByKey(f : (\mathrm{V}, \mathrm{V}) \Rightarrow \mathrm{V})$	$\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, V)]}$
	$union()$	$(\mathrm{RDD[T]}, \mathrm{RDD[T]}) \Rightarrow \mathrm{RDD[T]}$
	$join()$	$(\mathrm{RDD[(K, V)]}, \mathrm{RDD[(K, W)]}) \Rightarrow \mathrm{RDD[(K, (V, W))]}$
	$cogroup()$	$(\mathrm{RDD[(K, V)]}, \mathrm{RDD[(K, W)]}) \Rightarrow \mathrm{RDD[(K, (Seq[V], Seq[W]))]}$
	$crossProduct()$	$(\mathrm{RDD[T]}, \mathrm{RDD[U]}) \Rightarrow \mathrm{RDD[(T, U)]}$
	$mapValues(f : \mathrm{V} \Rightarrow \mathrm{W})$	$\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, W)]}$ (Preserves partitioning)
	$sort(c : \mathrm{Comparator[K]})$	$\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, V)]}$
	$partitionBy(p : \mathrm{Partitioner[K]})$	$\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, V)]}$

/ultiΣle 9esoqds

Table 2: Transformations and actions available on RDDs in Spark. Seq[T] denotes a sequence of elements of type T.

Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, Ion Stoica. "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing.". *NSDI 2012*. April 2012.

Original Transformations: RDD to RDD

Transformations	$map(f : \mathrm{T} \Rightarrow \mathrm{U})$	:	$\mathrm{RDD[T]} \Rightarrow \mathrm{RDD[U]}$
	$filter(f : \mathrm{T} \Rightarrow \mathrm{Bool})$	:	$\mathrm{RDD[T]} \Rightarrow \mathrm{RDD[T]}$
	$flatMap(f : \mathrm{T} \Rightarrow \mathrm{Seq[U]})$	:	$\mathrm{RDD[T]} \Rightarrow \mathrm{RDD[U]}$
	$sample(fraction : \mathrm{Float})$	:	$\mathrm{RDD[T]} \Rightarrow \mathrm{RDD[T]}$ (Deterministic sampling)
	$groupByKey()$	:	$\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, Seq[V])]}$
	$reduceByKey(f : (\mathrm{V}, \mathrm{V}) \Rightarrow \mathrm{V})$	:	$\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, V)]}$
	$union()$	:	$(\mathrm{RDD[T]}, \mathrm{RDD[T]}) \Rightarrow \mathrm{RDD[T]}$
	$join()$	:	$(\mathrm{RDD[(K, V)]}, \mathrm{RDD[(K, W)]}) \Rightarrow \mathrm{RDD[(K, (V, W))]}$
	$cogroup()$	:	$(\mathrm{RDD[(K, V)]}, \mathrm{RDD[(K, W)]}) \Rightarrow \mathrm{RDD[(K, (Seq[V], Seq[W]))]}$
	$crossProduct()$	:	$(\mathrm{RDD[T]}, \mathrm{RDD[U]}) \Rightarrow \mathrm{RDD[(T, U)]}$
	$mapValues(f : \mathrm{V} \Rightarrow \mathrm{W})$	:	$\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, W)]}$ (Preserves partitioning)
	$sort(c : \mathrm{Comparator[K]})$	:	$\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, V)]}$
	$partitionBy(p : \mathrm{Partitioner[K]})$	:	$\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, V)]}$

Table 2: Transformations and actions available on RDDs in Spark. Seq[T] denotes a sequence of elements of type T.

Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, Ion Stoica. "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing.". *NSDI 2012*. April 2012.

Original *Transformations*: RDD to RDD

Transformations	$map(f : \mathrm{T} \Rightarrow \mathrm{U})$	:	$\mathrm{RDD[T]} \Rightarrow \mathrm{RDD[U]}$
	$filter(f : \mathrm{T} \Rightarrow \mathrm{Bool})$	:	$\mathrm{RDD[T]} \Rightarrow \mathrm{RDD[T]}$
	$flatMap(f : \mathrm{T} \Rightarrow \mathrm{Seq[U]})$	:	$\mathrm{RDD[T]} \Rightarrow \mathrm{RDD[U]}$
	$sample(fraction : \mathrm{Float})$	:	$\mathrm{RDD[T]} \Rightarrow \mathrm{RDD[T]}$ (Deterministic sampling)
	$groupByKey()$	:	$\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, Seq[V])]}$
	$reduceByKey(f : (\mathrm{V}, \mathrm{V}) \Rightarrow \mathrm{V})$	:	$\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, V)]}$
	$union()$	:	$(\mathrm{RDD[T]}, \mathrm{RDD[T]}) \Rightarrow \mathrm{RDD[T]}$
	$join()$	:	$(\mathrm{RDD[(K, V)]}, \mathrm{RDD[(K, W)]}) \Rightarrow \mathrm{RDD[(K, (V, W))]}$
	$cogroup()$	:	$(\mathrm{RDD[(K, V)]}, \mathrm{RDD[(K, W)]}) \Rightarrow \mathrm{RDD[(K, (Seq[V], Seq[W]))]}$
	$crossProduct()$	:	$(\mathrm{RDD[T]}, \mathrm{RDD[U]}) \Rightarrow \mathrm{RDD[(T, U)]}$
	$mapValues(f : \mathrm{V} \Rightarrow \mathrm{W})$	:	$\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, W)]}$ (Preserves partitioning)
	$sort(c : \mathrm{Comparator[K]})$	:	$\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, V)]}$
	$partitionBy(p : \mathrm{Partitioner[K]})$	:	$\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, V)]}$

Original *Actions*: RDD to Value, Object, or Storage

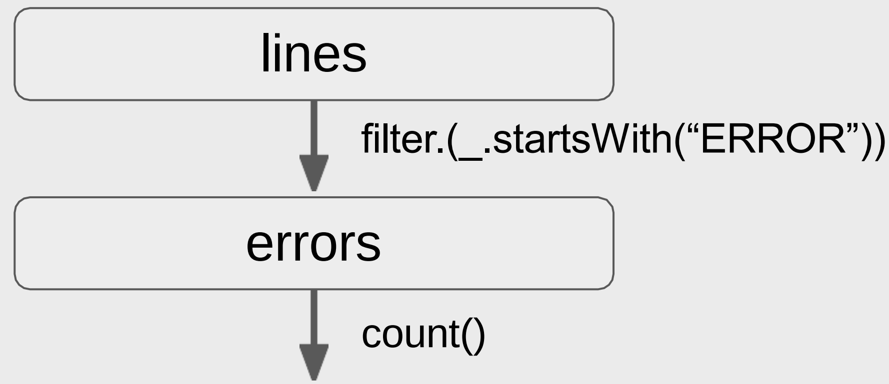
Actions	$count()$	:	$\mathrm{RDD[T]} \Rightarrow$ [redacted]
	$collect()$	:	$\mathrm{RDD[T]} \Rightarrow \mathrm{Seq[T]}$
	$reduce(f : (\mathrm{T}, \mathrm{T}) \Rightarrow \mathrm{T})$	:	$\mathrm{RDD[T]} \Rightarrow \mathrm{T}$
	$lookup(k : \mathrm{K})$	:	$\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{Seq[V]}$ (On hash/range partitioned RDDs)
	$save(path : \mathrm{String})$	:	Outputs RDD to a storage system, *e.g.*, HDFS

Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, Ion Stoica. "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing.". *NSDI 2012*. April 2012.

An Example

Count errors in a log file:

TYPE   MESSAGE   TIME

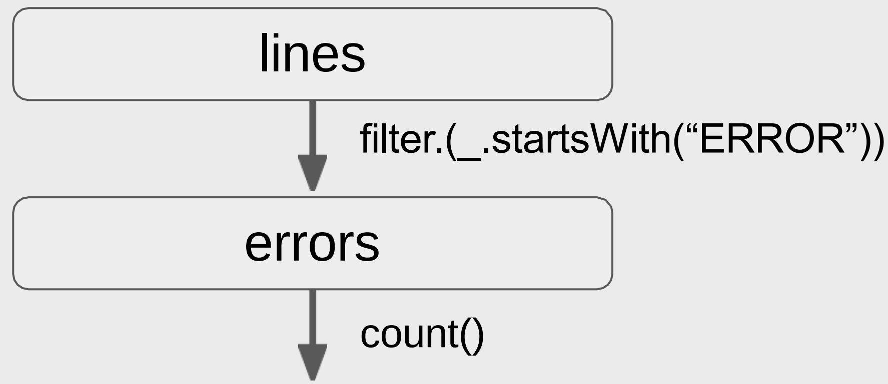


Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, Ion Stoica. [“Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing.”.] *NSDI 2012*. April 2012.

An Example

Count errors in a log file:

TYPE MESSAGE TIME



Pseudocode:

```
lines = sc.textFile(“dfs:...”) errors =
        lines.filter(_.startswith(“ERROR”)) errors.count
```

Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, Ion Stoica. “Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing.”. *NSDI 2012*. April 2012.

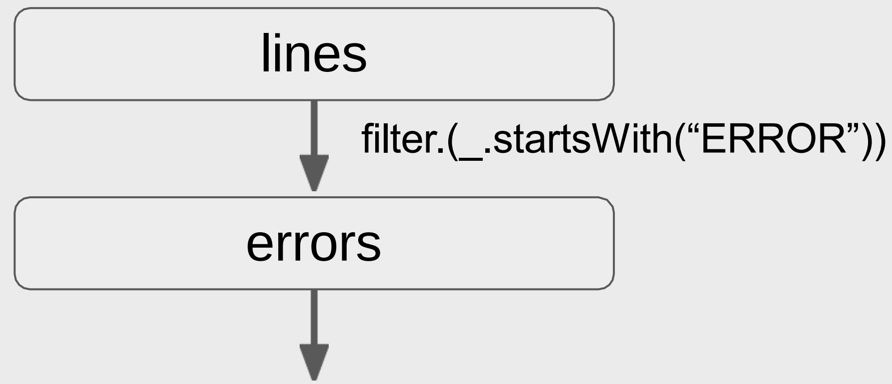
An Example

Collect times of hdfs-related errors

TYPE *MESSAGE* *TIME*

Pseudocode:

```
lines = sc.textFile("dfs:...") errors =
        lines.filter(_.startswith("ERROR")) errors.persist
errors.count
...
```



Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, Ion Stoica. "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing.". *NSDI 2012*. April 2012.

An Example

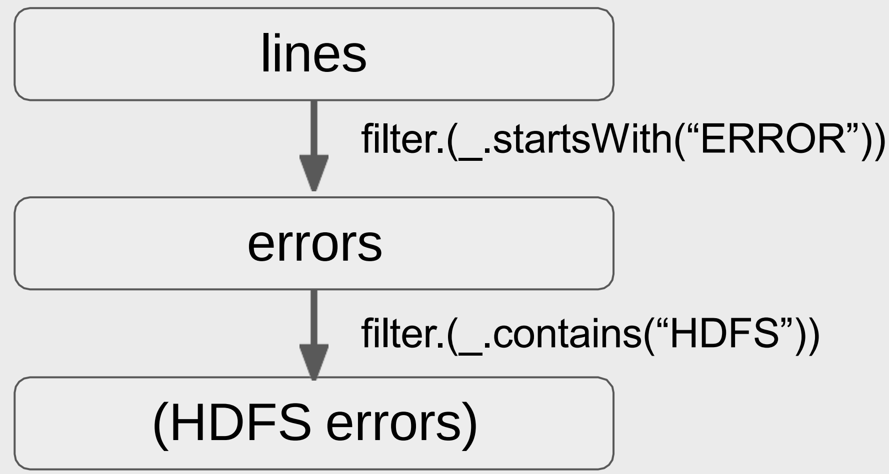
Collect times of hdfs-related errors

TYPE *MESSAGE* *TIME*

```
Pseudocode:

lines = sc.textFile(“dfs:...”) errors =
        lines.filter(_.startswith(“ERROR”)) errors.persist
```
~~errors.count~~ errors.filter(_.contains(“HDFS”))

...



Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, Ion Stoica. [“Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing.”.] *NSDI 2012*. April 2012.

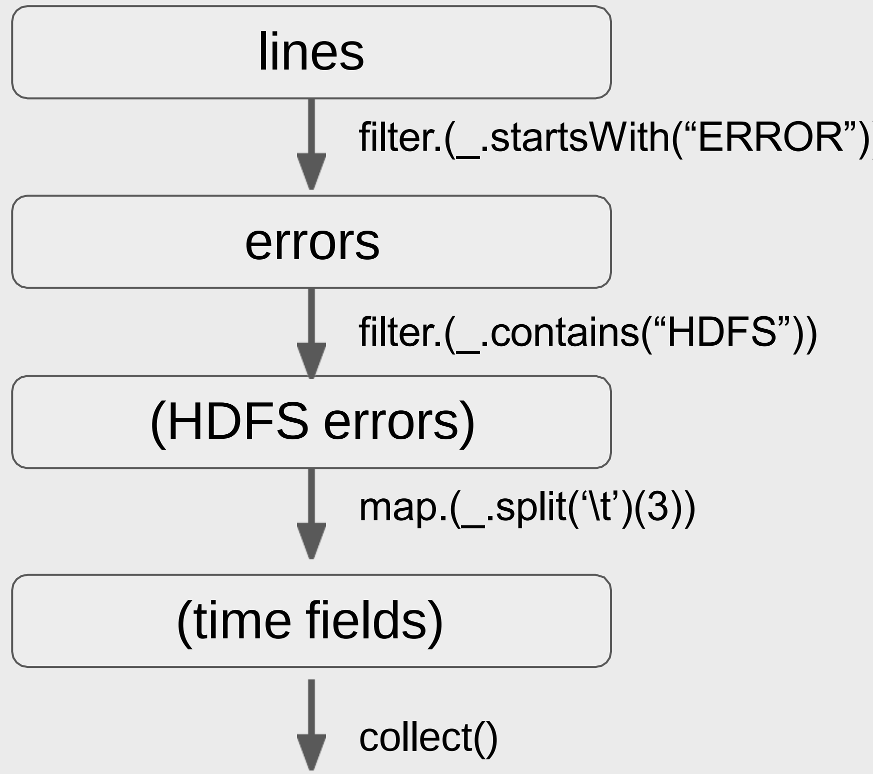
An Example

Collect times of hdfs-related errors

TYPE MESSAGE TIME

Pseudocode:

lines = sc.textFile("dfs:...") errors =
    lines.filter(_.startswith("ERROR")) errors.persist
~~errors.count~~ errors.filter(_.contains("HDFS"))
    .map(_split('\t')(3))
    .collect()



Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, Ion Stoica. "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing.". *NSDI 2012*. April 2012.

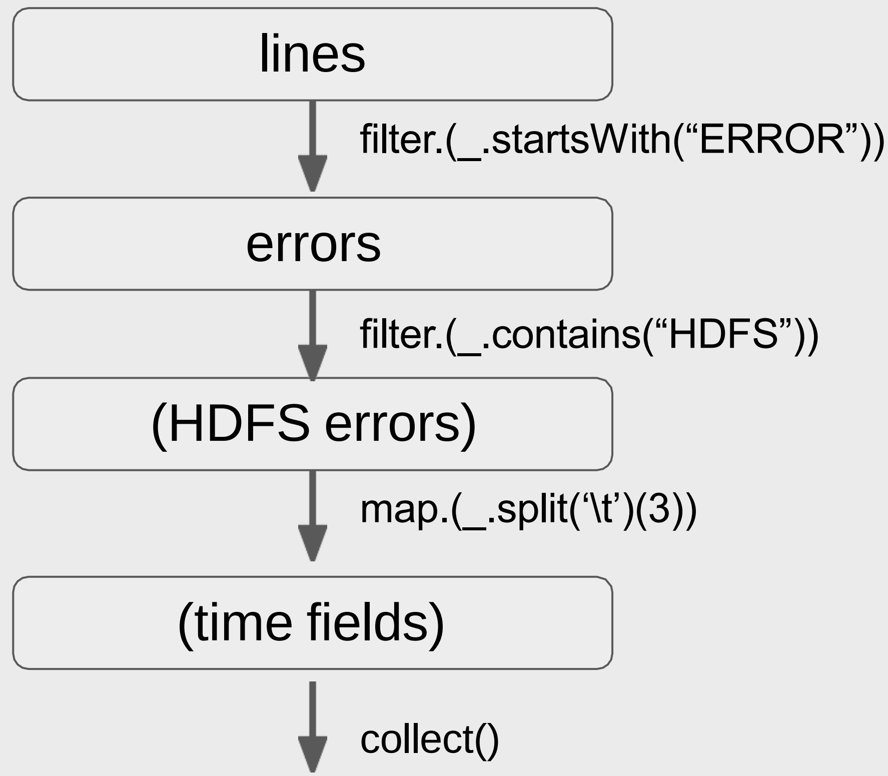
An Example

Collect times of hdfs-related errors

TYPE MESSAGE TIME

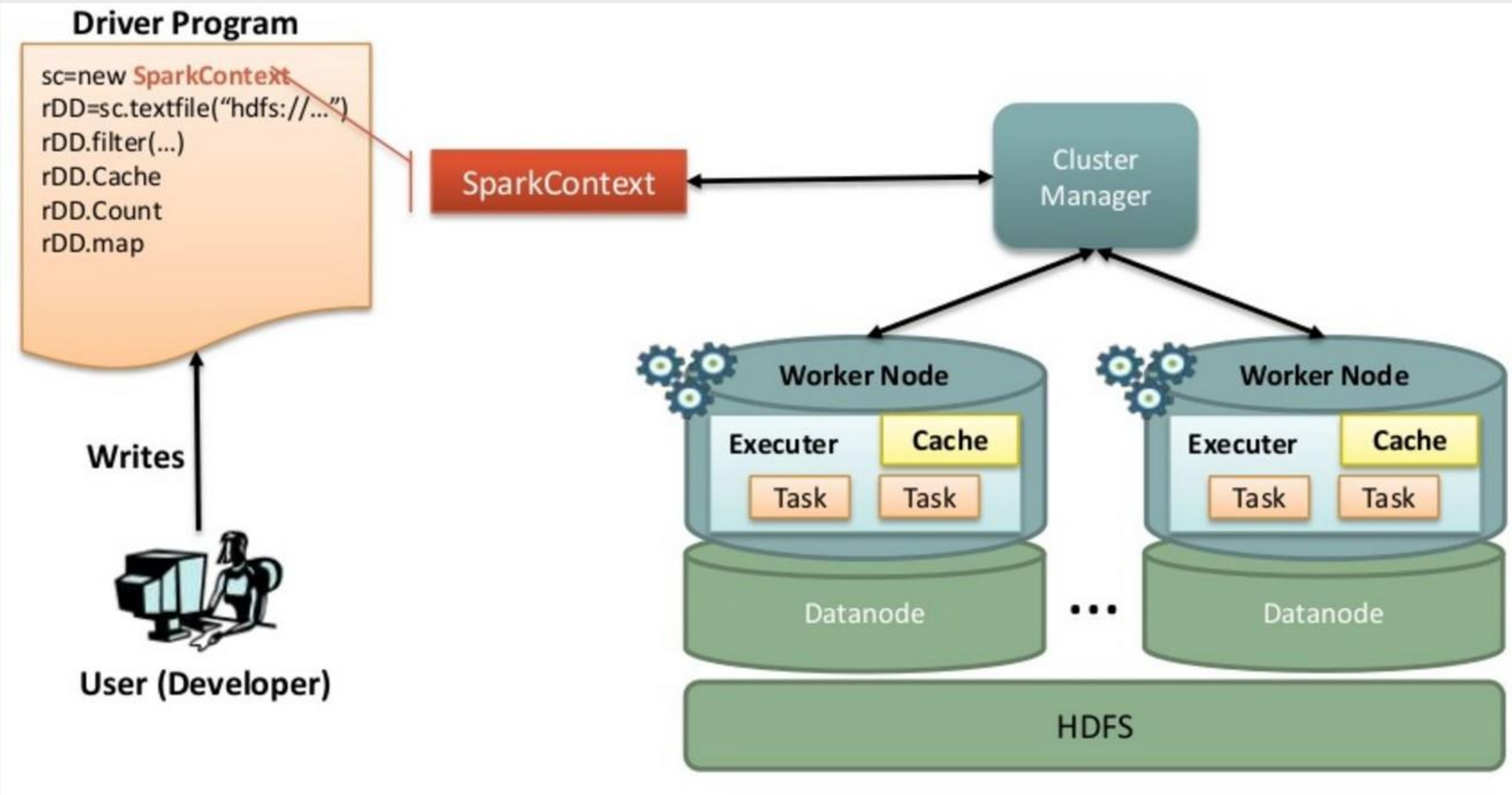
Pseudocode:

lines = sc.textFile(“dfs:...”) errors =
    lines.filter(_.startswith(“ERROR”)) errors.persist
~~errors.count~~ errors.filter(_.contains(“HDFS”))
      .map(_split(‘\t’)(3))
      .collect()



Functional Programming

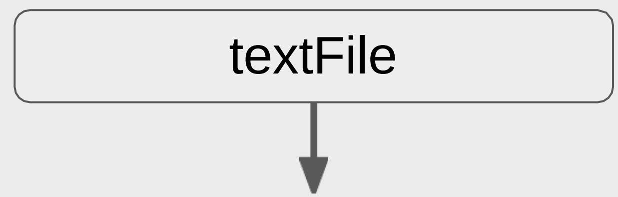
Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, Ion Stoica. “Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing.”. *NSDI 2012*. April 2012.



Gupta, Manish. Lightening Fast Big Data Analytics using Apache Spark. *UniCom 2014.*

An Example

Word Count

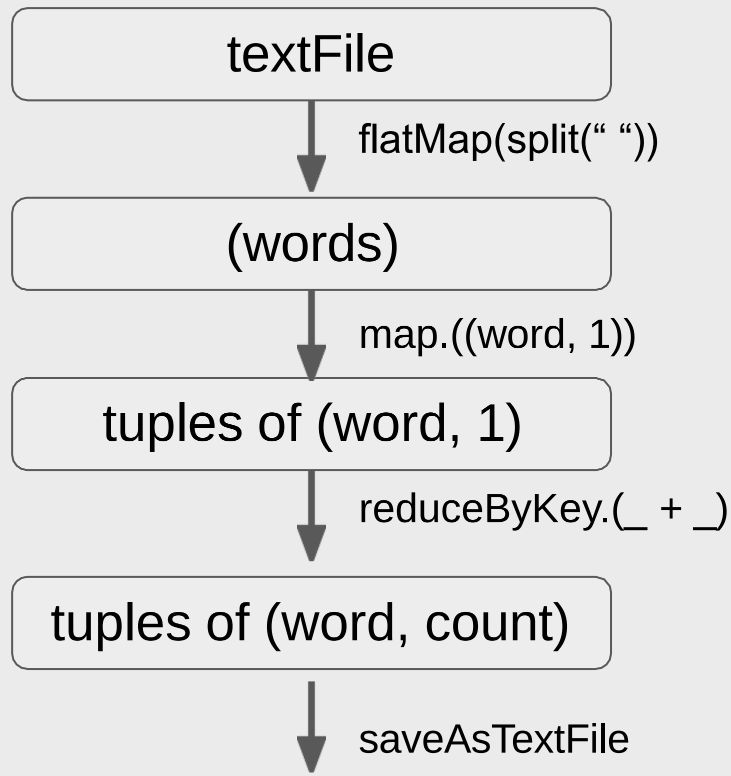


An Example

Word Count

Scala:

```scala
val textFile =
      sc.textFile("hdfs://...")
val counts = textFile
      .flatMap(line => line.split(" "))
      .map(word => (word, 1))
      .reduceByKey(_ + _)
counts.saveAsTextFile("hdfs://...")
```



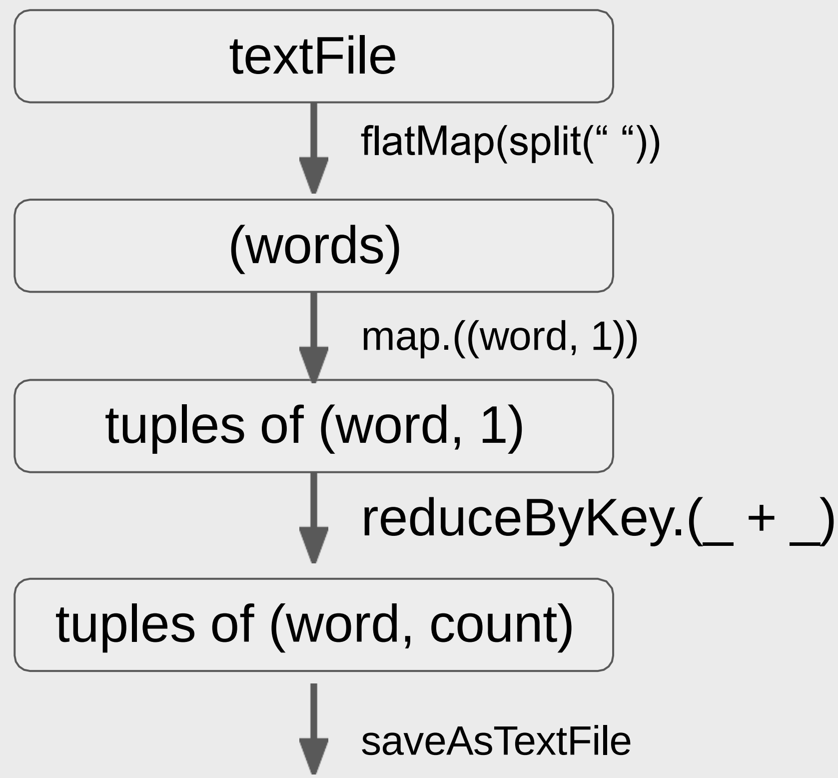
Apache Spark Examples
http://spark.apache.org/examples.html

An Example

Word Count

Python:

```
textFile =  sc.textFile("hdfs://...") counts =
textFile
        .flatMap(lambda line: line.split(" "))
        .map(lambda word: (word, 1))
        .reduceByKey(lambda a, b: a  +  b)
counts.saveAsTextFile("hdfs://...")
```



Apache Spark Examples
http://spark.apache.org/examples.html

Lazy Evaluation

Spark waits to **load data** and **execute transformations** until necessary -- ***lazy***
Spark tries to complete **actions** as immediately as possible -- ***eager***

Why?

- Only executes what is necessary to achieve action.

- Can optimize the complete *chain of operations* to reduce communication

Lazy Evaluation

Spark waits to *load data* and *execute transformations* until necessary -- ***lazy***
Spark tries to complete actions as quickly as possible -- ***eager***

Why?

- Only executes what is necessary to achieve action.
- Can optimize the complete *chain of operations* to reduce communication

e.g.

```
rdd.map(lambda r: r[1]*r[3]).take(5)          #only executes map for five records

rdd.filter(lambda r: "ERROR" in r[0]).map(lambda r: r[1]*r[3])
                                    #only passes through the data once
```

Broadcast Variables

Read-only objects can be shared across all nodes.

Broadcast variable is a wrapper: access object with .value

Python:

```
filterWords = [‘one’, ‘two’, ‘three’, ‘four’, …] fwBC =
sc.broadcast(set(filterWords))
```

Broadcast Variables

Read-only objects can be shared across all nodes.

Broadcast variable is a wrapper: access object with .value

Python:

```
filterWords = ['one', 'two', 'three', 'four', …] fwBC =
sc.broadcast(set(filterWords))

textFile =  sc.textFile("hdfs:...") counts =
textFile
        .map(lambda line: line.split(" "))
        .filter(lambda words: len(set(words) and word in fwBC.value) > 0)
        .flatMap(lambda word: (word, 1))
        .reduceByKey(lambda a, b: a  +  b)
counts.saveAsTextFile("hdfs:...")
```

Accumulators

Write-only objects that keep a running aggregation

Default Accumulator assumes sum function

```
initialValue = 0
sumAcc = sc.accumulator(initialValue)
rdd.foreach(lambda i: sumAcc.add(i))
print(sumAcc.value)
```

Accumulators

Write-only objects that keep a running aggregation

Default Accumulator assumes sum function

Custom Accumulator: Inherit (AccumulatorParam) as class and override methods

```
initialValue = 0
sumAcc = sc.accumulator(initialValue)
rdd.foreeach(lambda i: sumAcc.add(i))
print(minAcc.value)

class MinAccum(AccumulatorParam):
    def zero(self, zeroValue = np.inf):#overwrite this
        return zeroValue
    def addInPlace(self, v1, v2):#overwrite this
        return min(v1, v2)
minAcc = sc.accumulator(np.inf, minAccum())
rdd.foreeach(lambda i: minAcc.add(i))
print(minAcc.value)
```

Spark Overview

- RDD provides full recovery by backing up transformations from stable storage rather than backing up the data itself.
- RDDs, which are immutable, can be stored in memory and thus are often much faster.
- Functional programming is used to define transformation and actions on RDDs.

Spark Overview

- RDD provides full recovery by backing up transformations from stable storage rather than backing up the data itself.
- RDDs, which are immutable, can be stored in memory and thus are often much faster.
- Functional programming is used to define transformation and actions on RDDs.
- Still need Hadoop (or some DFS) to hold original or resulting data efficiently and reliably.
- Lazy evaluation enables optimizing chain of operations.
- Memory across Spark cluster should be large enough to hold entire dataset to fully leverage speed.
  - MapReduce may still be more cost-effective for very large data that does not fit in memory.

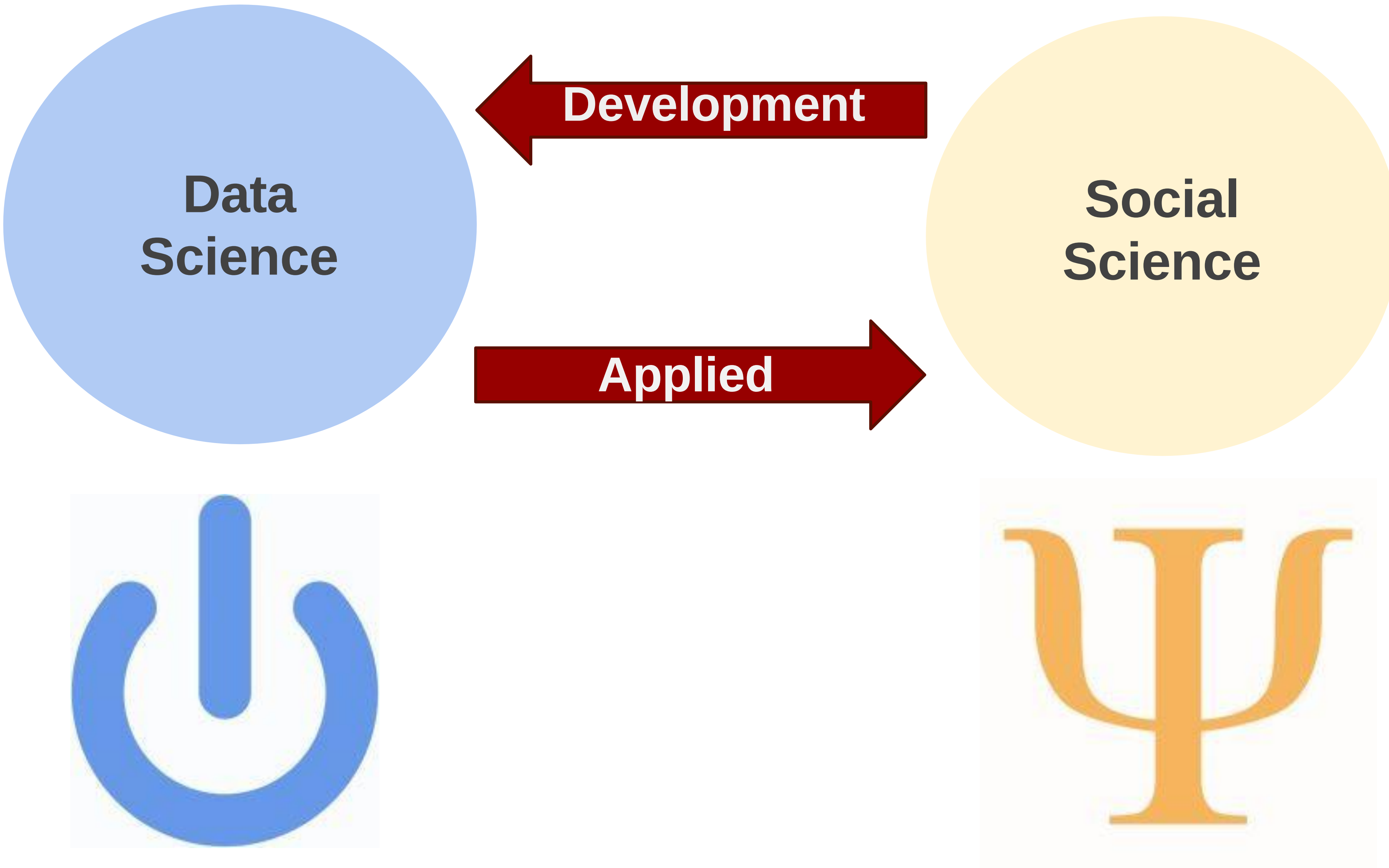
Big Data
and
Scientific Applications

Why Social Scientific Applications?

Applications that make a difference in the world.

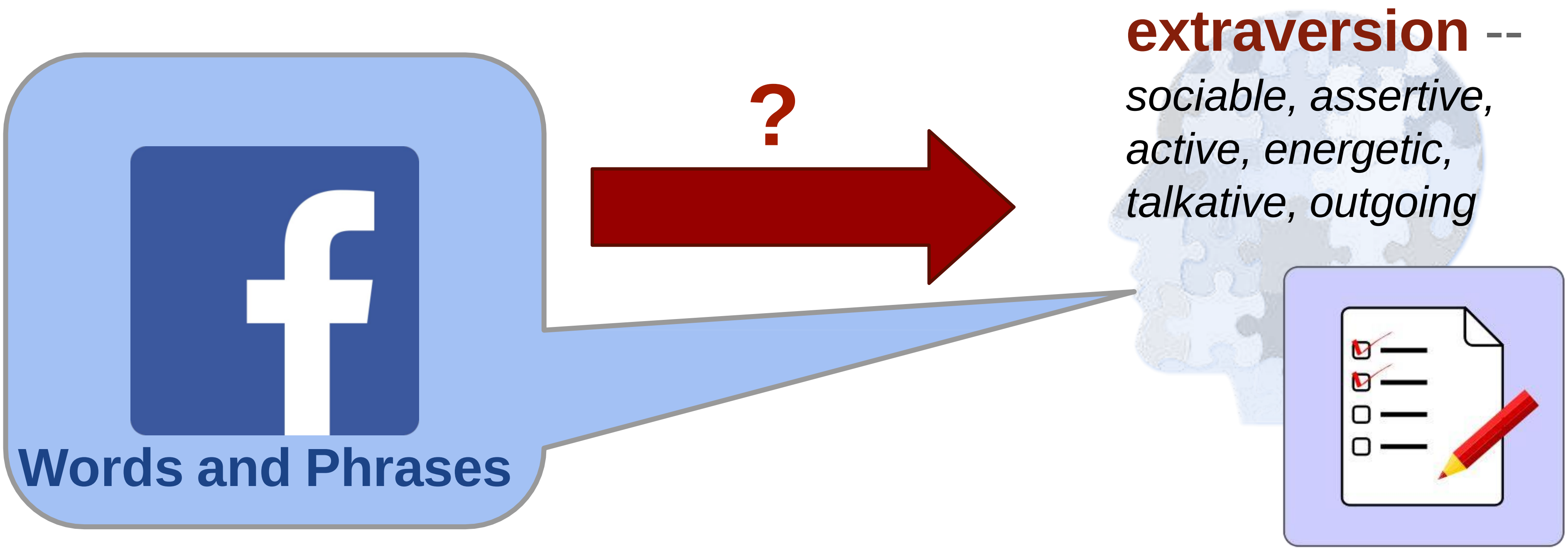
Often public data available.

Experience working toward an objective and/or using data to answer questions.


Data Science
Development
Applied
Social Science


SUSTAINABLE DEVELOPMENT GOALS
1 NO POVERTY
2 ZERO HUNGER
3 GOOD HEALTH AND WELL-BEING
4 QUALITY EDUCATION
5 GENDER EQUALITY
6 CLEAN WATER AND SANITATION
7 AFFORDABLE AND CLEAN ENERGY
8 DECENT WORK AND ECONOMIC GROWTH
9 INDUSTRY, INNOVATION AND INFRASTRUCTURE
10 REDUCED INEQUALITIES
11 SUSTAINABLE CITIES AND COMMUNITIES
12 RESPONSIBLE CONSUMPTION AND PRODUCTION
13 CLIMATE ACTION
14 LIFE BELOW WATER
15 LIFE ON LAND
16 PEACE, JUSTICE AND STRONG INSTITUTIONS
17 PARTNERSHIPS FOR THE GOALS
SUSTAINABLE DEVELOPMENT GOALS

Language Says A Lot About People



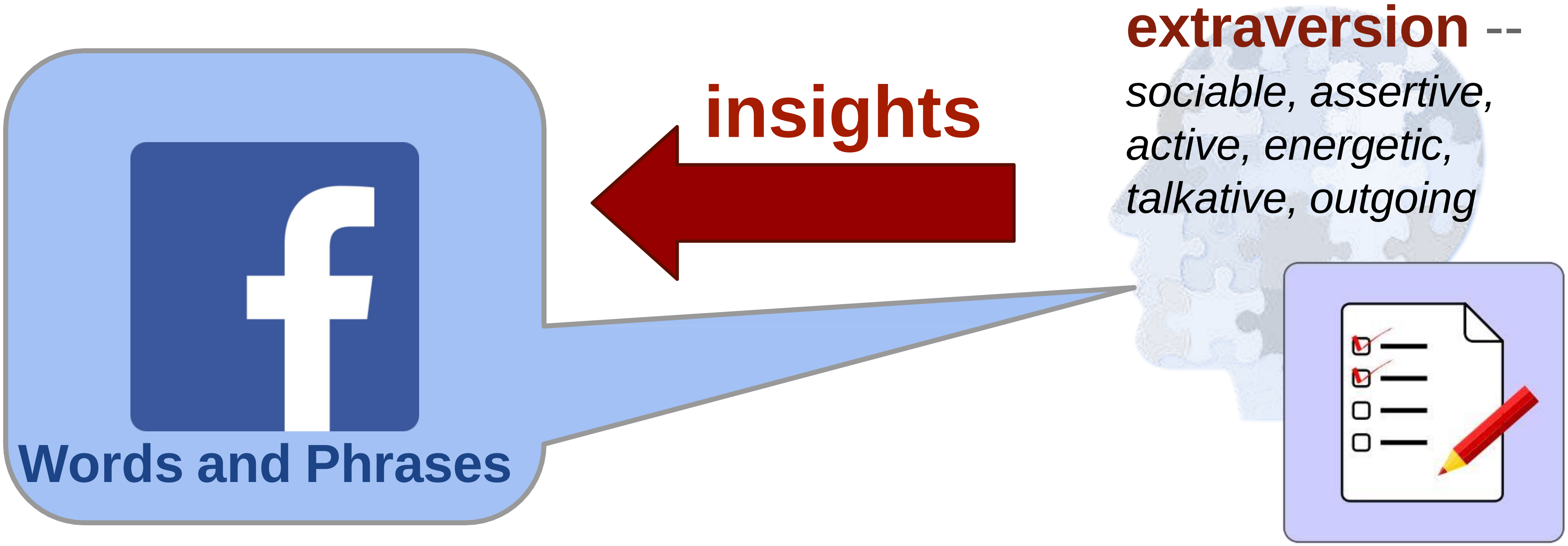
19M Facebook posts

75,000 personality surveys

Schwartz, H. A., Eichstaedt, J. C., Kern, M. L., Dziurzynski, L., Ramones, S. M., Agrawal, M., Shah, A., Kosinski, M., Stillwell, D., Seligman, M. E. P., & Ungar, L. H. (2013). **Personality, Gender, and Age in the Language of Social Media: The Open-Vocabulary Approach**. *In PLOS ONE 8(9)*.

Does language use reflect who we are?

Language Says A Lot About People



19M Facebook posts

75,000 personality surveys

Language Says A Lot About People

Words and Phrases

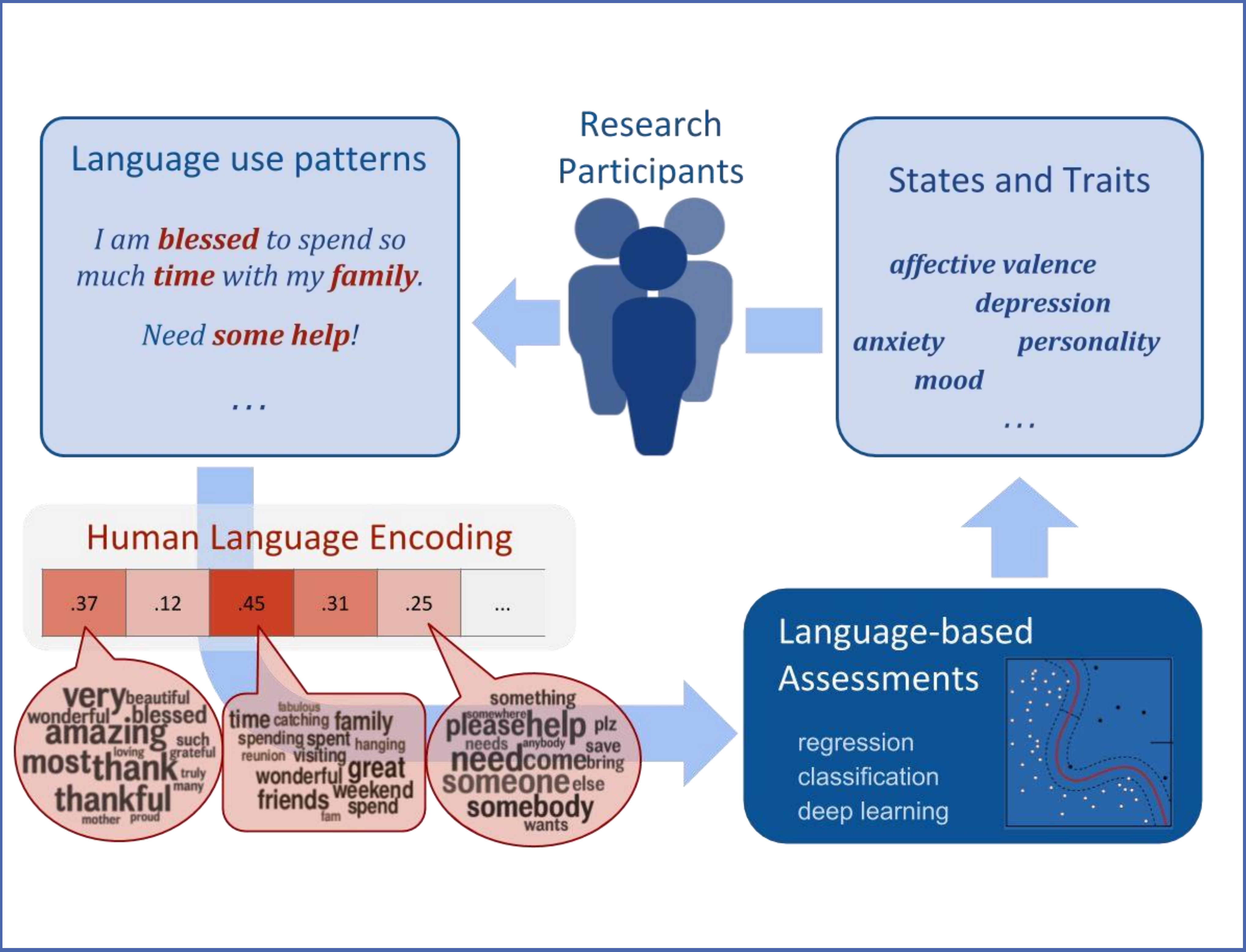
Predict?

extraversion --
sociable, assertive, active, energetic, talkative, outgoing

19M Facebook posts

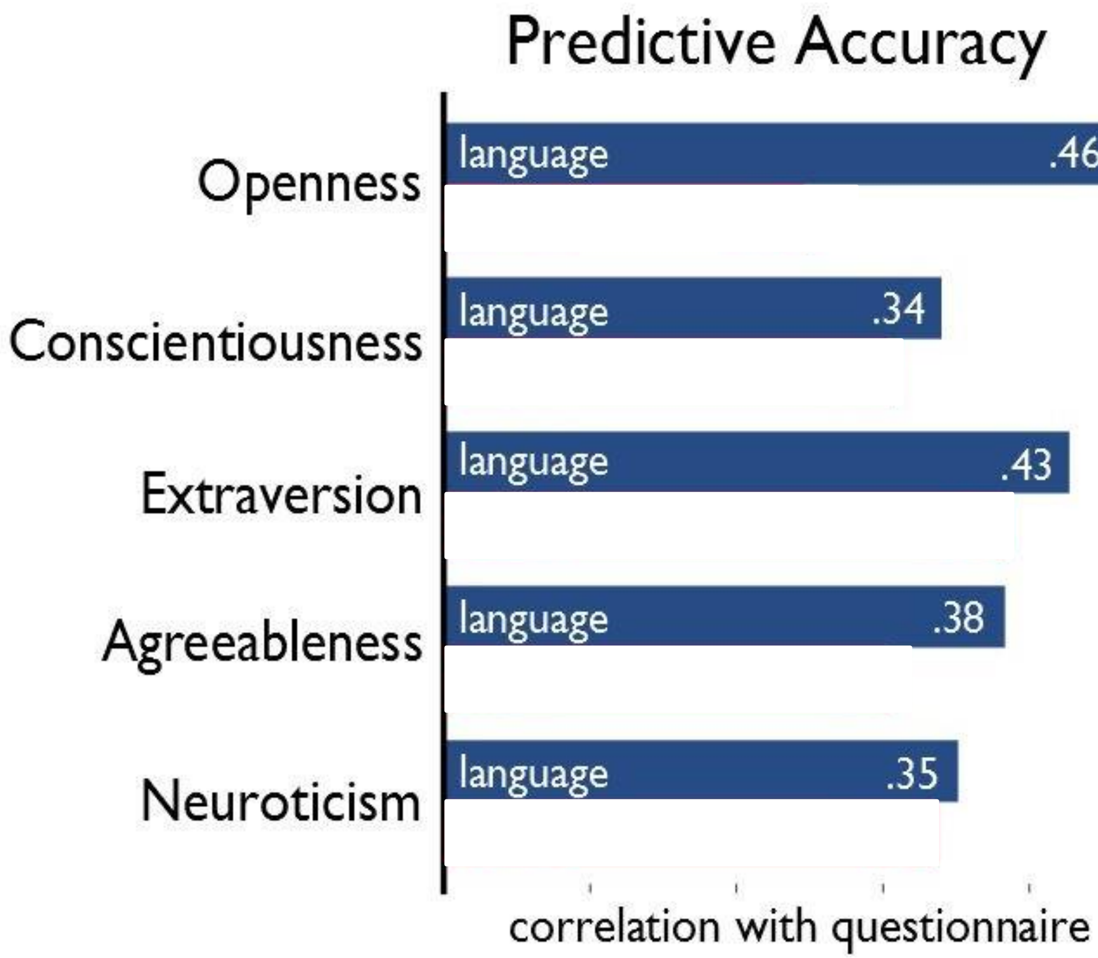
75,000 personality surveys

“Language-based Assessments”

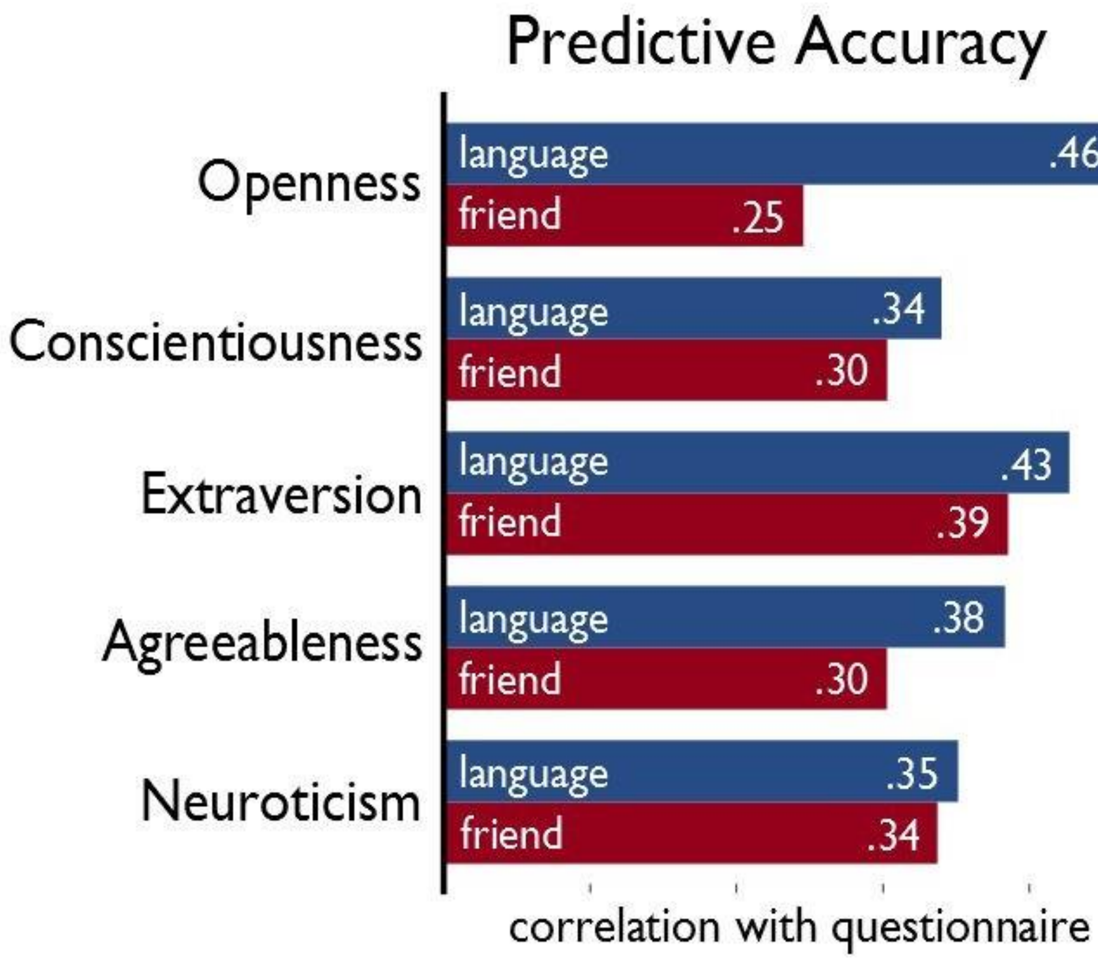


Research Participants
Language use patterns
I am blessed to spend so much time with my family.
Need some help!
...
States and Traits
affective valence
depression
anxiety
personality
mood
...
Human Language Encoding
.37
.12
.45
.31
.25
...
very beautiful wonderful blessed amazing such loving grateful most thank truly many thankful mother proud
fabulous time catching family spending spent hanging reunion visiting great wonderful weekend friends spend fam
something somewhere please help plz needs anybody save need come bring someone else somebody wants
Language-based Assessments
regression
classification
deep learning

Language-Based Assessment Evaluation



Language-Based Assessment Evaluation



Other Outcomes?

Characterizing Gratitude
(Carpenter et al., 2016)

Personality
(Schwartz et al., 2013; Park et al., 2015)

Life Satisfaction
(Schwartz et al., 2013; 2016)

Emotion / Affect
(Preotiuc-Pietro et al., 2016)

Mental Health
(Schwartz et al., 2013; Coppersmith et al., 2014; Eichstaedt et al., 2018)

Demographics
(Sap et al., 2014)

Dark Triad
(Preotiuc-Pietro et al., 2016)

Temporal Orientation
(Schwartz et al., 2015)

Spiritual/Religious Outcomes
(Yaden et al., 2016, 2017)

Meaning in Life
(Schwartz et al., 2016)

Causal Explanations
(Son et al., 2018)

Trustfulness
(Buffone et al., 2018)

Control
(Rouhizadeh et al., 2018)

Depth?

Life Satisfaction
(Schwartz et al., 2013; 2016)

Personality
(Schwartz et al., 2013; Park et al., 2015)

Characterizing Gratitude
(Carpenter et al., 2016)

Emotion / Affect
(Preotiuc-Pietro et al., 2016)

Mental Health
(Schwartz et al., 2013; Coppersmith et al., 2014; Eichstaedt et al., 2018)

Demographics
(Sap et al., 2014)

Dark Triad
(Preotiuc-Pietro et al., 2016)

Spiritual/Religious Outcomes
(Yaden et al., 2016, 2017)

Temporal Orientation
(Schwartz et al., 2015)

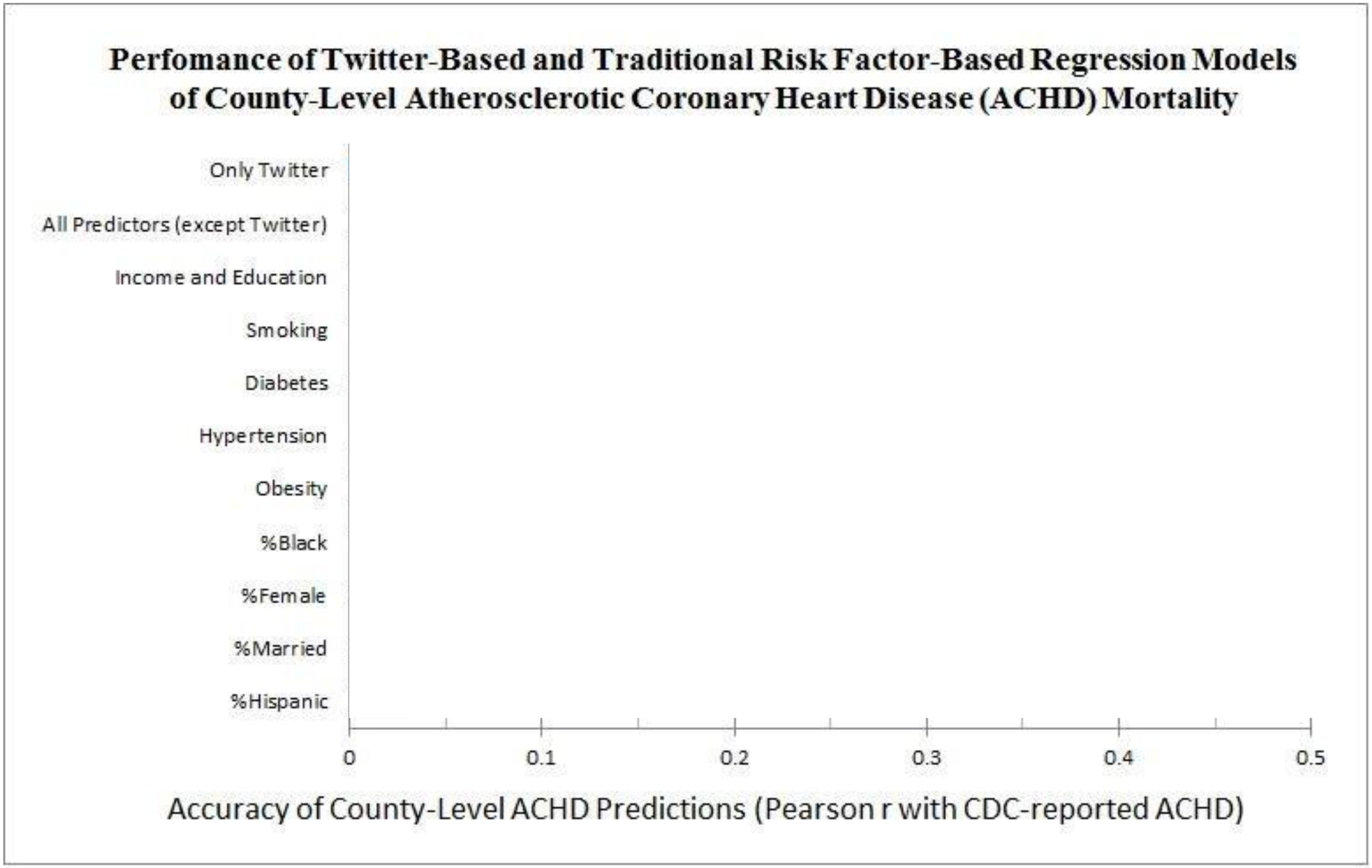
Meaning in Life
(Schwartz et al., 2016)

Causal Explanations
(Son et al., 2018)

Control
(Rouhizadeh et al., 2018)

Trustfulness
(Buffone et al., 2018)

Twitter Predicts Heart Disease



Eichstaedt, J. C., Schwartz, H. A., Kern, M. L., Park, G.,..., Ungar, L. H., & Seligman, M. E. (2015). Psychological Language on Twitter Predicts County-Level Heart Disease Mortality. *Psychological Science 26*(2), 159-169

Heart Disease Mortality Insight

Higher Status Occupations

$r = -.12$ to $r = -.13$

Positive Emotions, Engagement

$r = -.13$ to $r = -.14$

Anger, Hostility, Aggression

$r = .16$ to $r = .19$

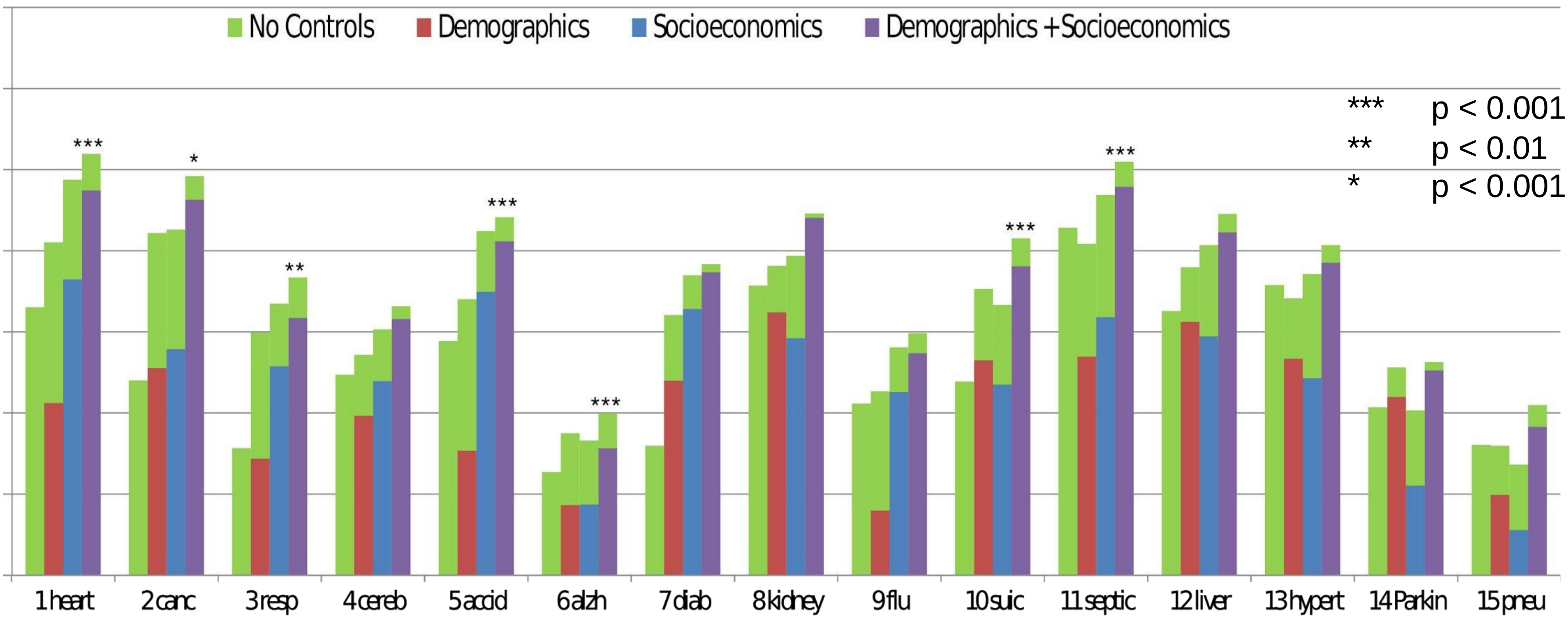
No Social Support

$r = .16$ to $r = .20$

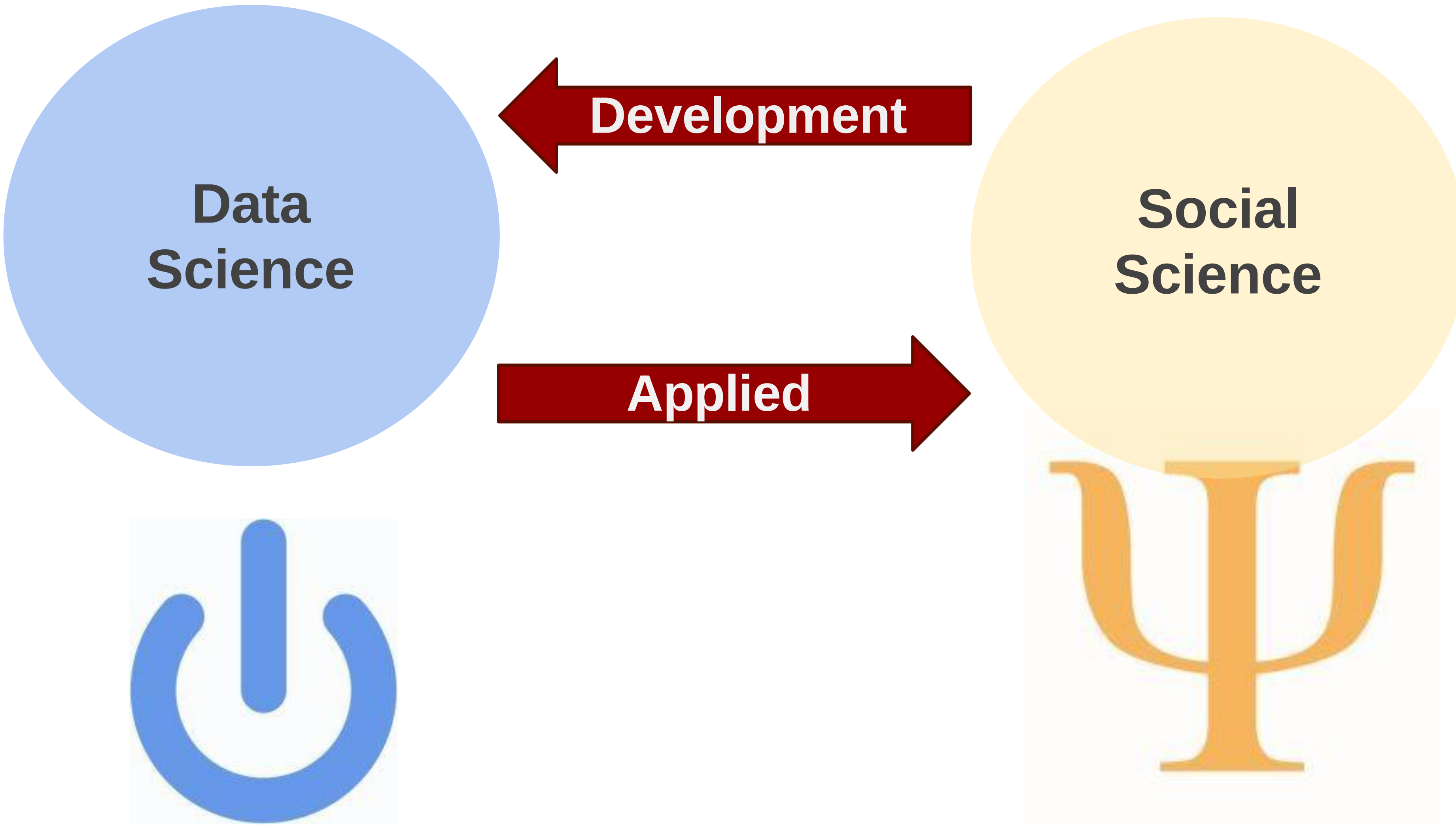
1. Diseases of heart
2. Malignant neoplasms (cancers)
3. Chronic lower respiratory
4. Cerebrovascular diseases (strokes)
5. Accidents, unintentional
6. Alzheimer’s disease
7. Diabetes melitus
8. Kidney Diseases
9. Influenza & Pneumonia
10. Intentional self-harm (suicide)
11. Septicemia
12. Liver Disease
13. Hypertension
14. Parkinson’s
15. Pneumonitus

TOP 15 Causes of Death, 2013

1. Diseases of heart
2. Malignant neoplasms (cancers)
3. Chronic lower respiratory
4. Cerebrovascular diseases (strokes)
5. Accidents, unintentional
6. Alzheimer's disease
7. Diabetes melitus
8. Kidney Diseases
9. Influenza & Pneumonia
10. Intentional self-harm (suicide)
11. Septicemia
12. Liver Disease
13. Hypertension
14. Parkinson's
15. Pneumonitus



How can your project make an impact?



The End