COSI 129a

Introduction to Big Data Analysis

Fall 2016

Map Reduce

What is MapReduce?

It is a **programming framework** introduced by Google in 2004 to support **scalable** parallel and fault-tolerant computations over large data sets on clusters of computers.

Motivation

- Originally:
  - Web data analysis by Google
- Many others followed later:
  - Scientific data analysis
  - Business data analysis
  - …

Google: The Data Challenge

- Jeffrey Dean, Google Fellow, PACT'06 keynote speech:
  - 20+ billion web pages x 20KB = 400 TB
  - One computer can read 30-35 MB/sec from disk
    - ~ 4 months to read the web
  - ~ 1,000 hard drives just to store the web
  - Even more to "do" something with the data

- MapReduce CACM'08 article:
  - 100,000 MR jobs executed in Google every day
  - Total data processed > **20 PB of data per day**

What is MapReduce?

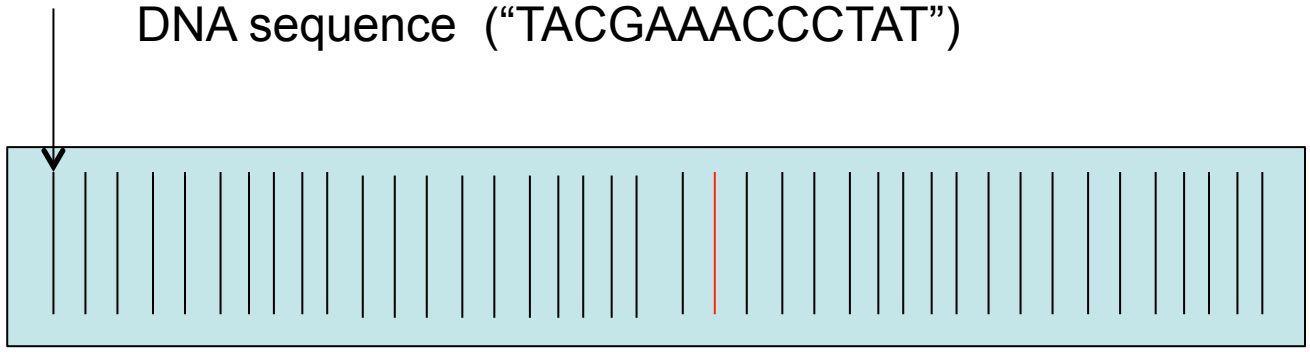
It is a **programming framework** introduced by Google in 2004 to support **scalable** parallel and fault-tolerant computations over large data sets on clusters of computers.

What Does Scalable Mean?

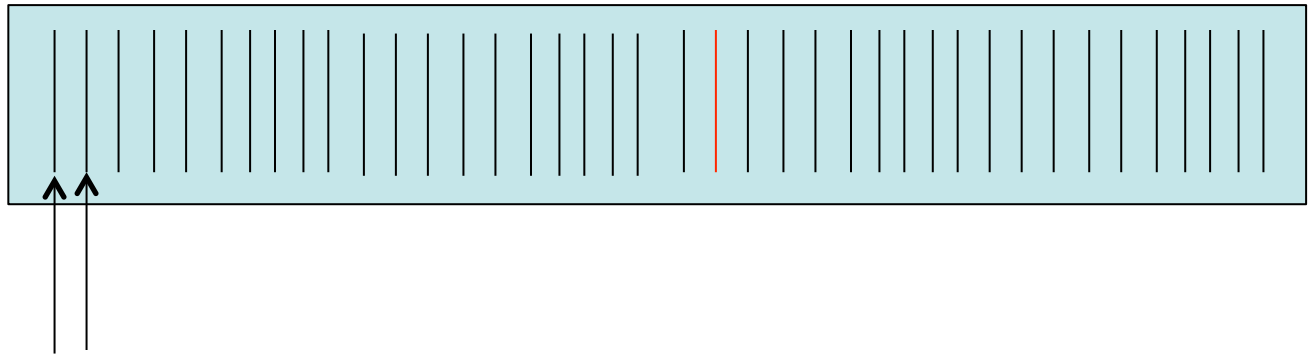
- Operationally
  - In the past: "Works even if data does not fit in main memory"
    Now: "Can we make use of 1000s of cheap computers"
- Algorithmically
  - In the past: "If you have N data items, you must do no more than N^m operations" : "polynomial time algorithms"
  - Now: "If you have N data items, you must do no more than N^m/k operations, for some large k (k: # machines)
    - Polynomial time algorithms must be parallelized
  - Soon: If you have N data items, you should not do more than N* log(N) operations
    - As data size goes up, you may only get to one pass of the data

Example: Find matching DNA sequences

- Given a set of sequences
  - Find all sequences equal to
    - "GATTACGAATTTA"



Slides from "Introduction to Data Science", Coursera, Bill Howe

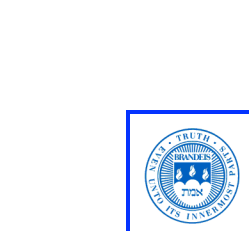


Solution 1: Linear Search

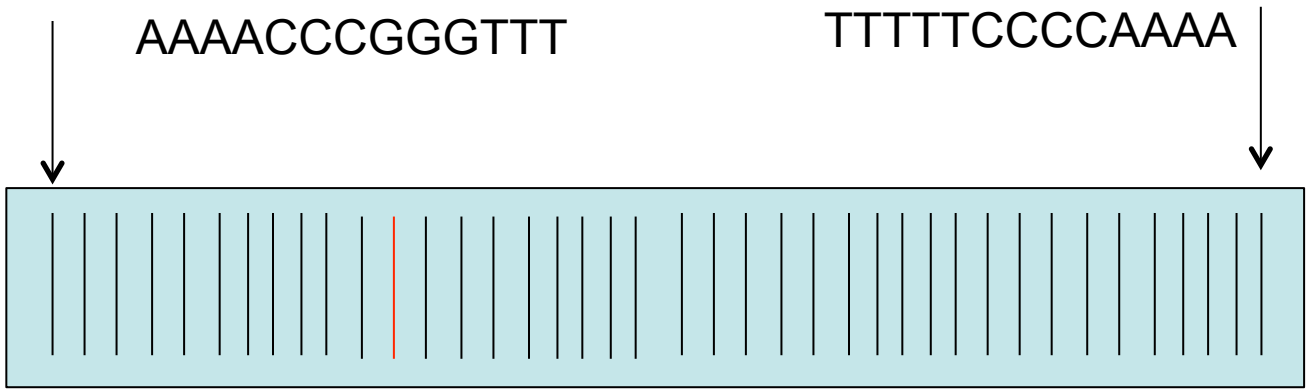
"CGATCCGTAAATT" == GATTACGAATTTA NO

- Check one sequence at a time with our search target
- Time: N sequences means N computations
- Algorithmic complexity is order N : O(N)

Slides from "Introduction to Data Science", Coursera, Bill Howe



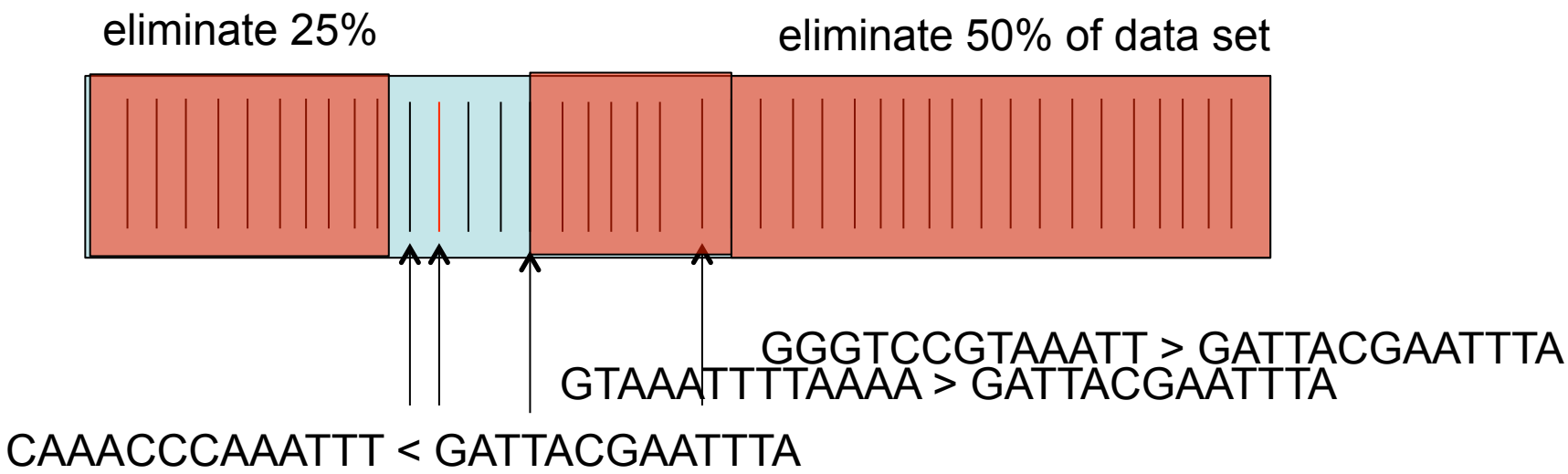
Solution 2: Binary Search



- Sort the sequences and start at the middle

Slides from "Introduction to Data Science", Coursera, Bill Howe

Solution 2: Binary Search

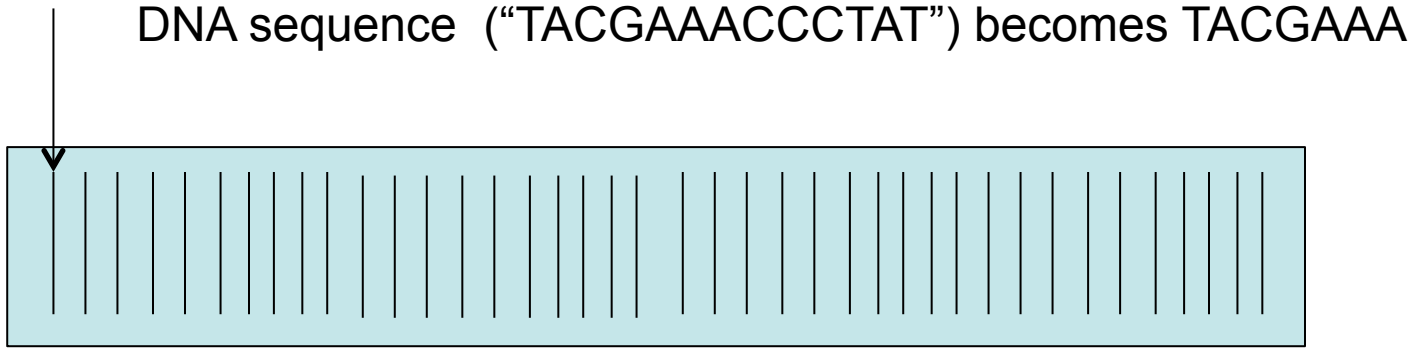


- Only 4 comparisons
- N records, log(N) comparisons
- Algorithmic complexity O(log(N)): more scalable

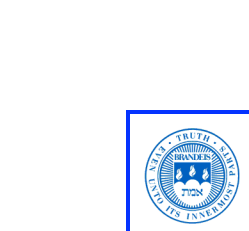
Slides from "Introduction to Data Science", Coursera, Bill Howe

Example: Read Trimming

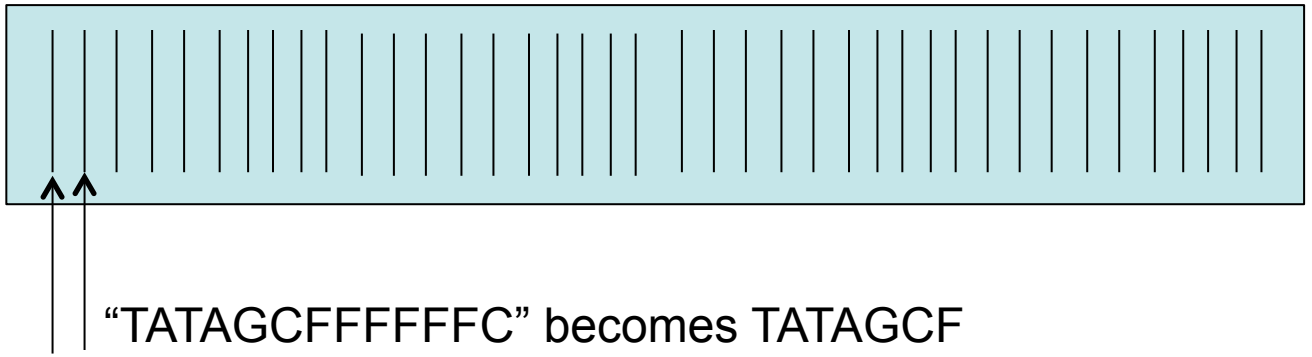
- Given a set of DNA sequences
- Trim the final n "characters" of each sequence
- Generate a new dataset



Slides from "Introduction to Data Science", Coursera, Bill Howe

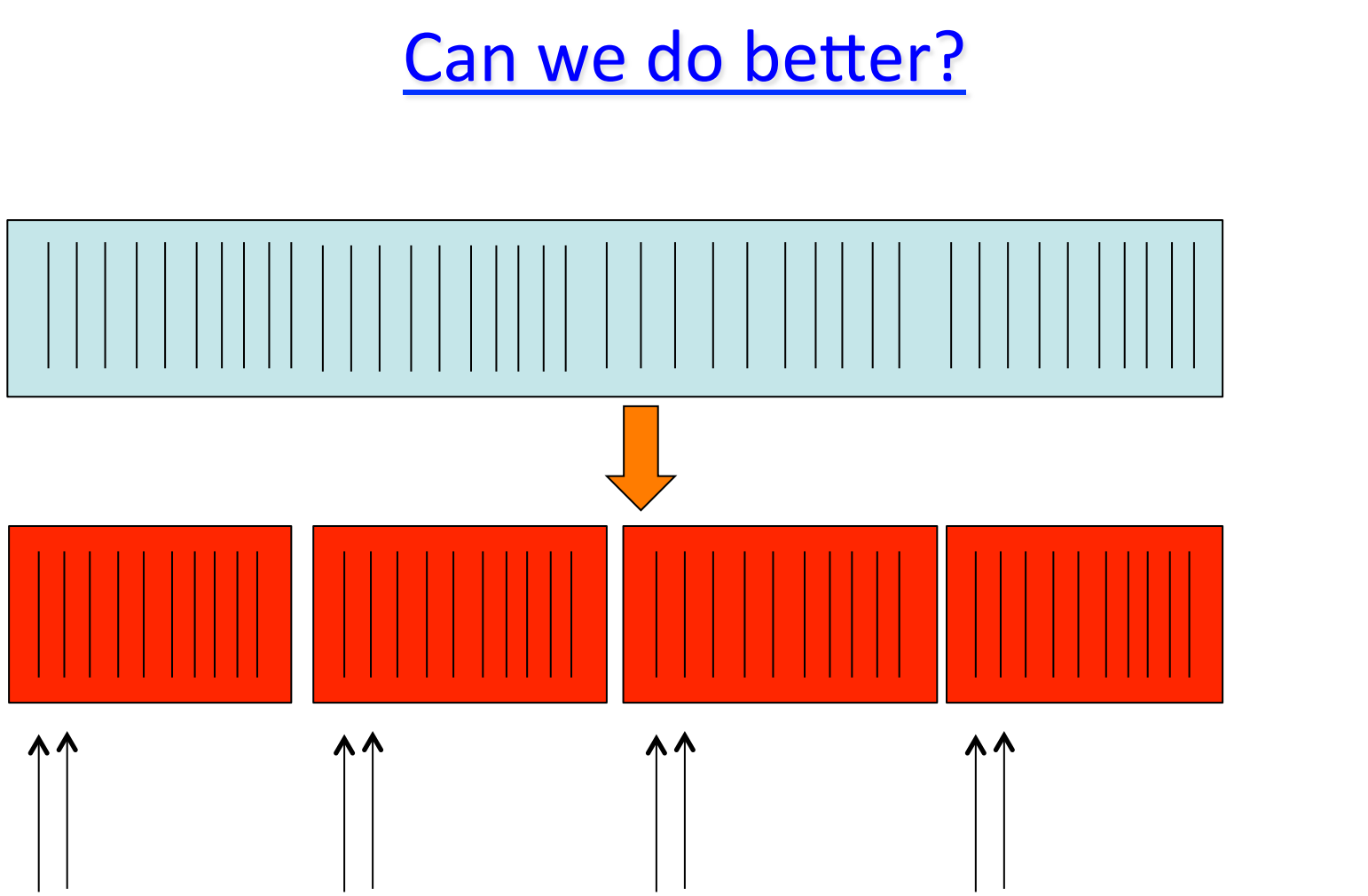


Solution 1: Linear Search Again



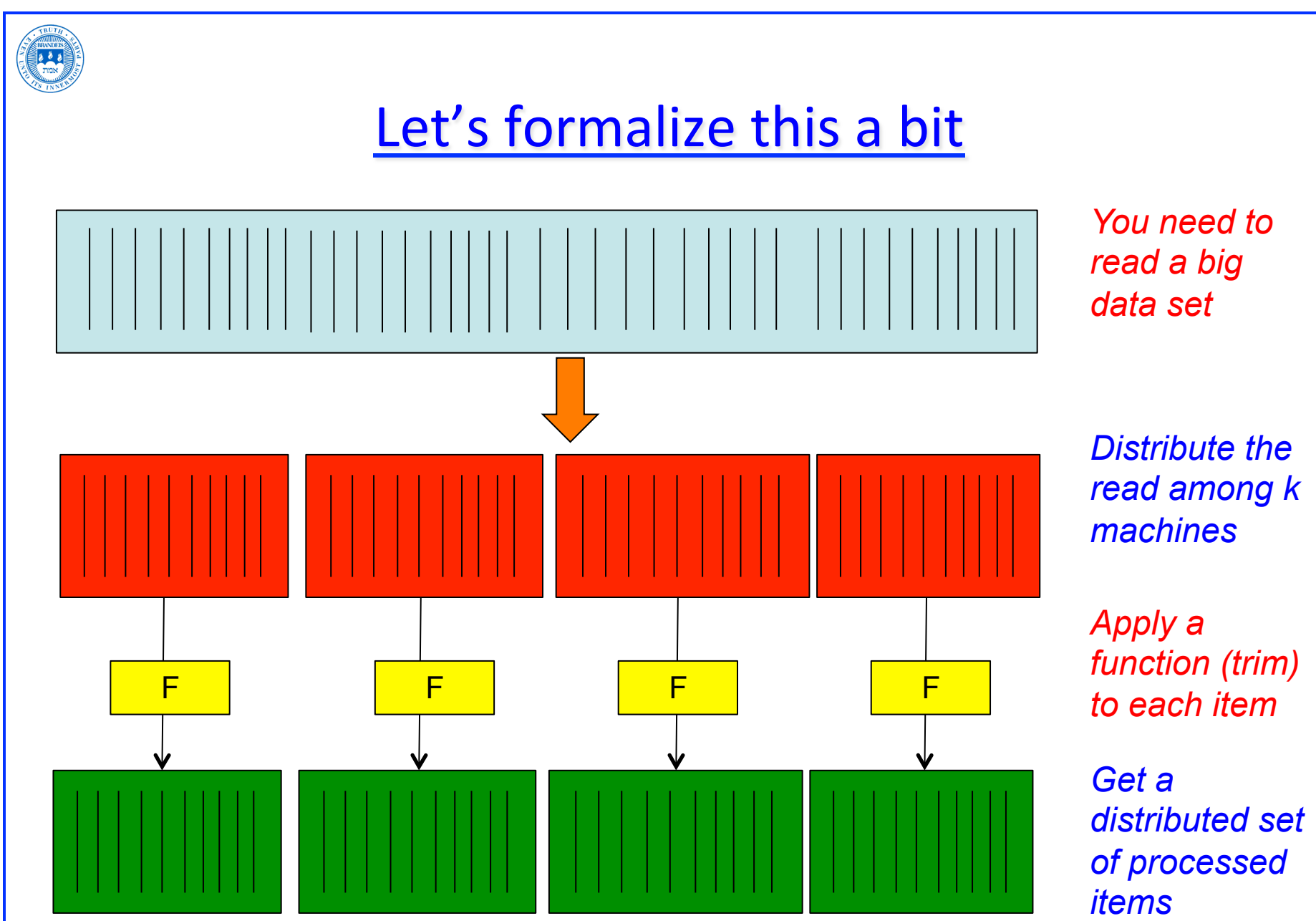
- Check one sequence at a time
- Algorithmic complexity is fundamentally O(N)
  - We have to read every data item no matter what

Slides from "Introduction to Data Science", Coursera, Bill Howe

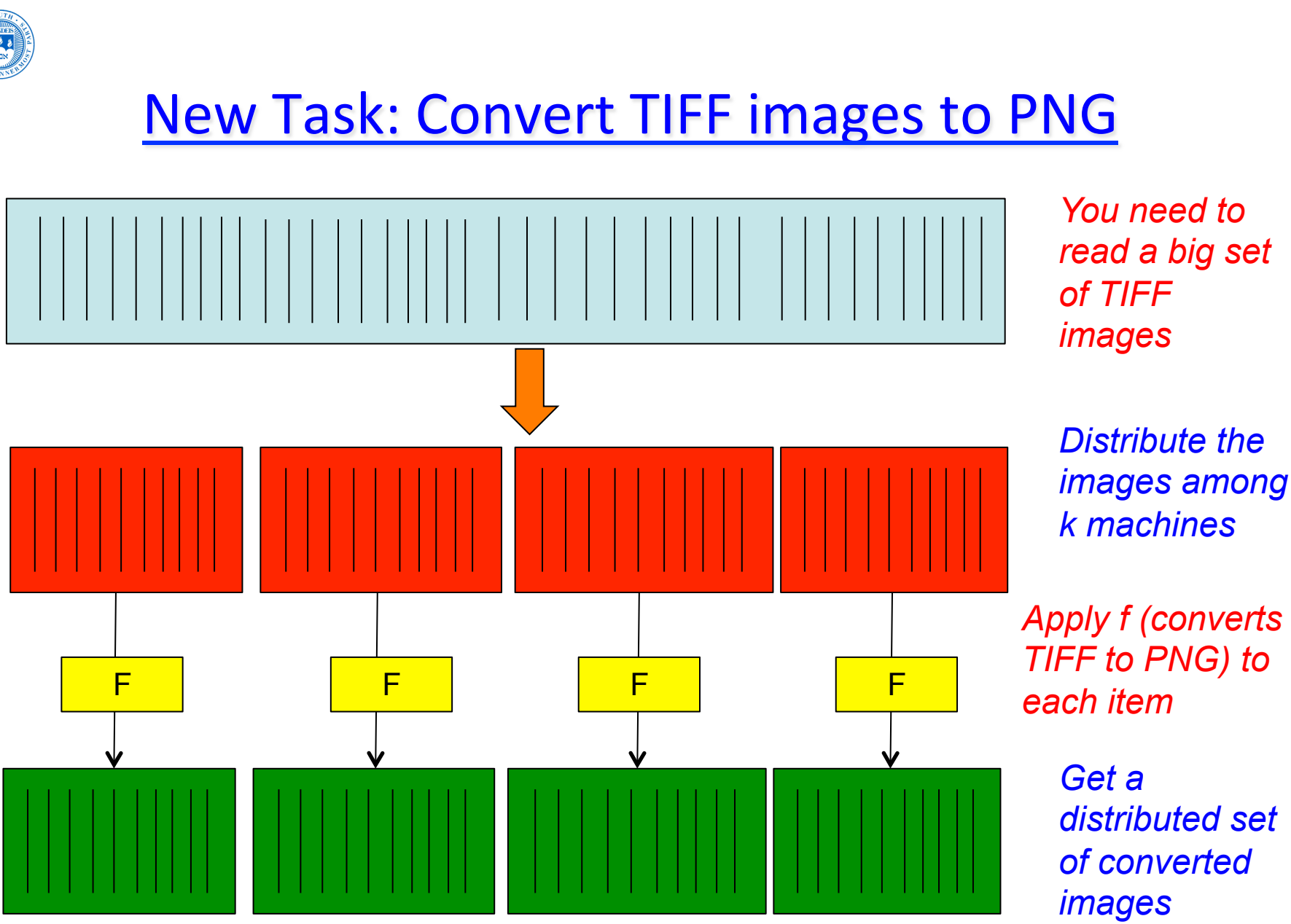


Can we do better?

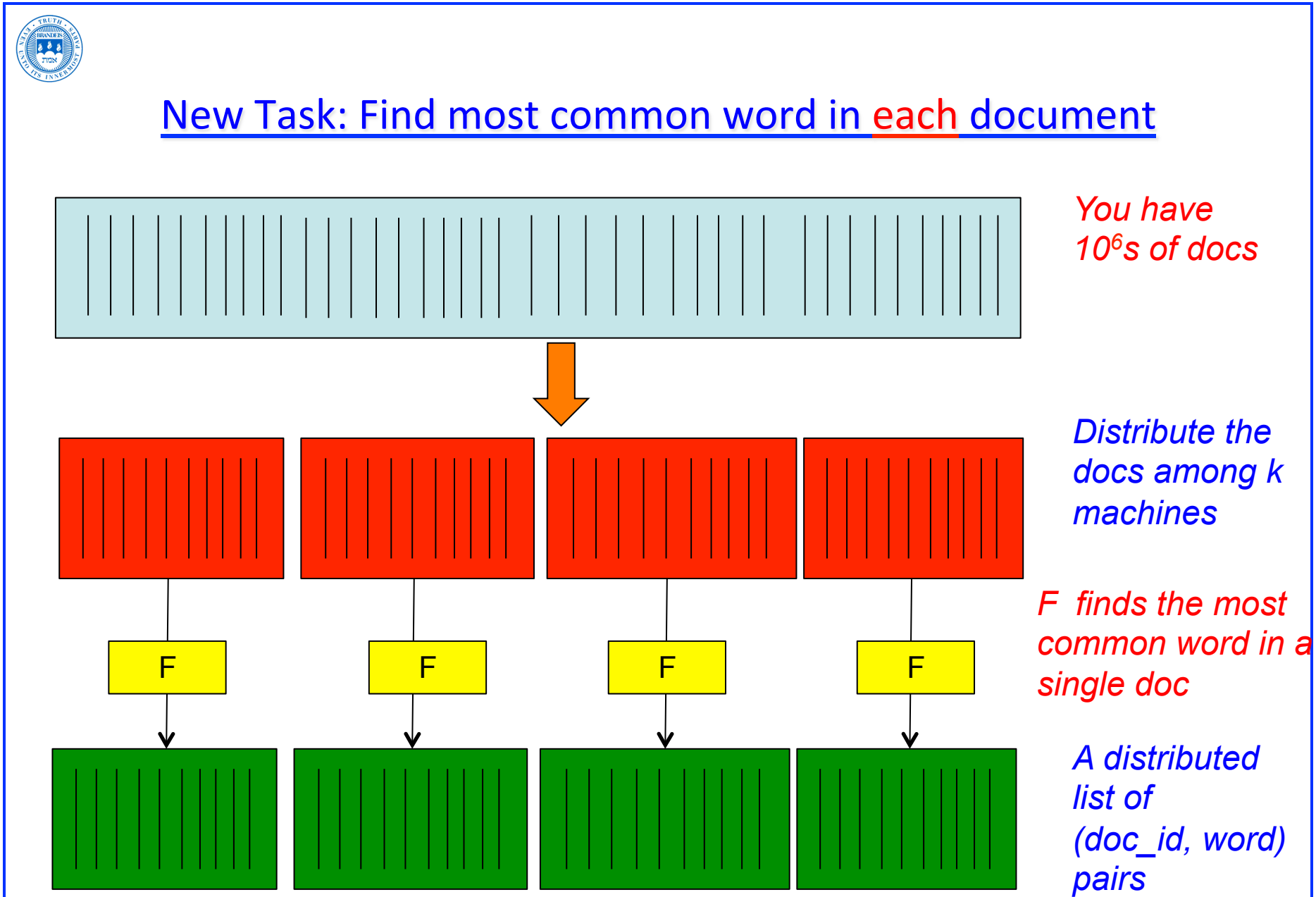
Takes O(N/k) , N: # of data items, k : # of machines



Slides from "Introduction to Data Science", Coursera, Bill Howe



Slides from "Introduction to Data Science", Coursera, Bill Howe



New Task: Find most common word in each document

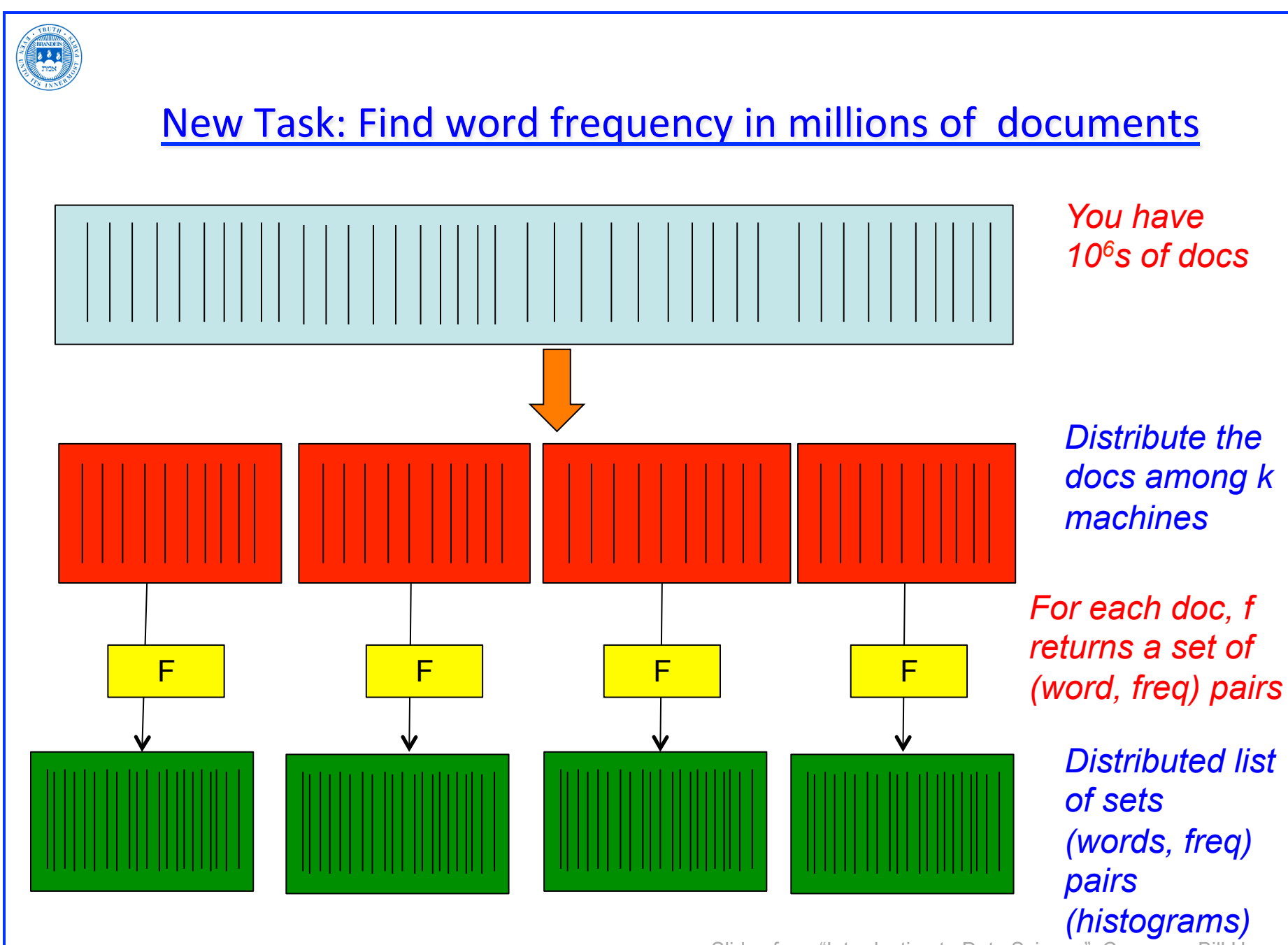
You have 10^6s of docs

Distribute the docs among k machines

F finds the most common word in a single doc

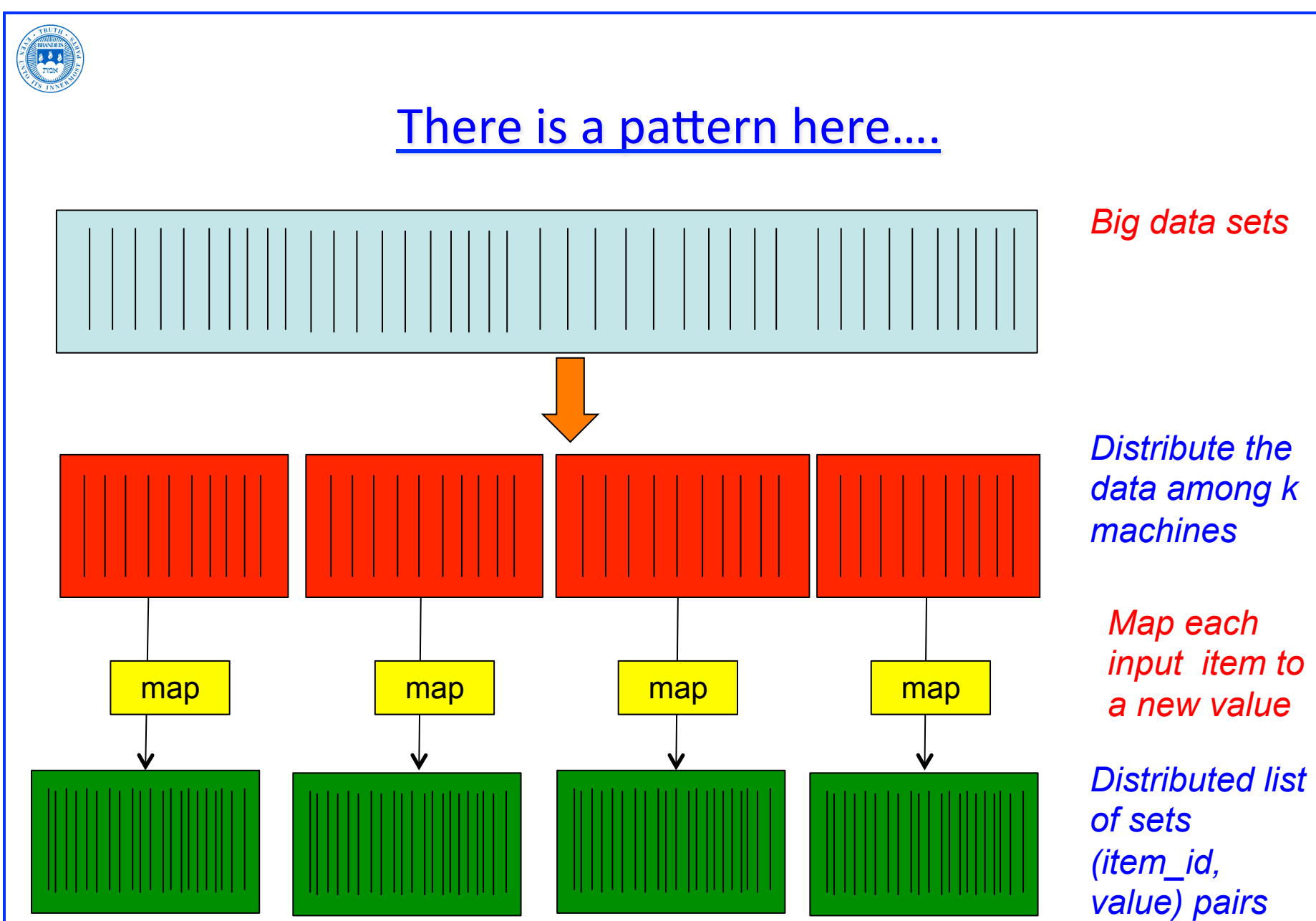
A distributed list of (doc_id, word) pairs

Slides from "Introduction to Data Science", Coursera, Bill Howe



New Task: Find word frequency in millions of documents

Slides from "Introduction to Data Science", Coursera, Bill Howe



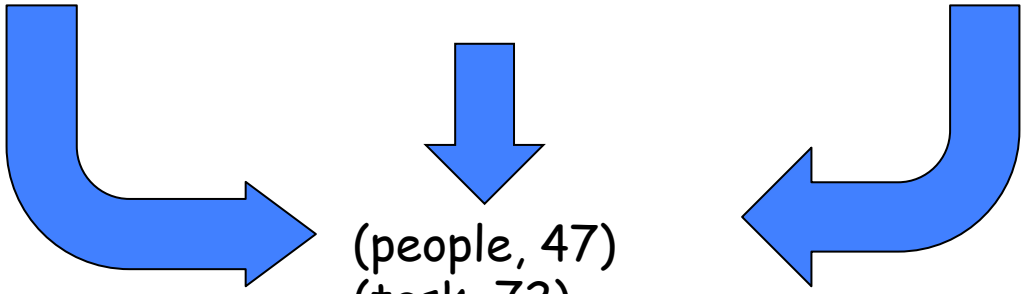
Slides from "Introduction to Data Science", Coursera, Bill Howe

New Task: Find word frequency across all documents

(people, 10)
(task, 34)
(receipt, 23)

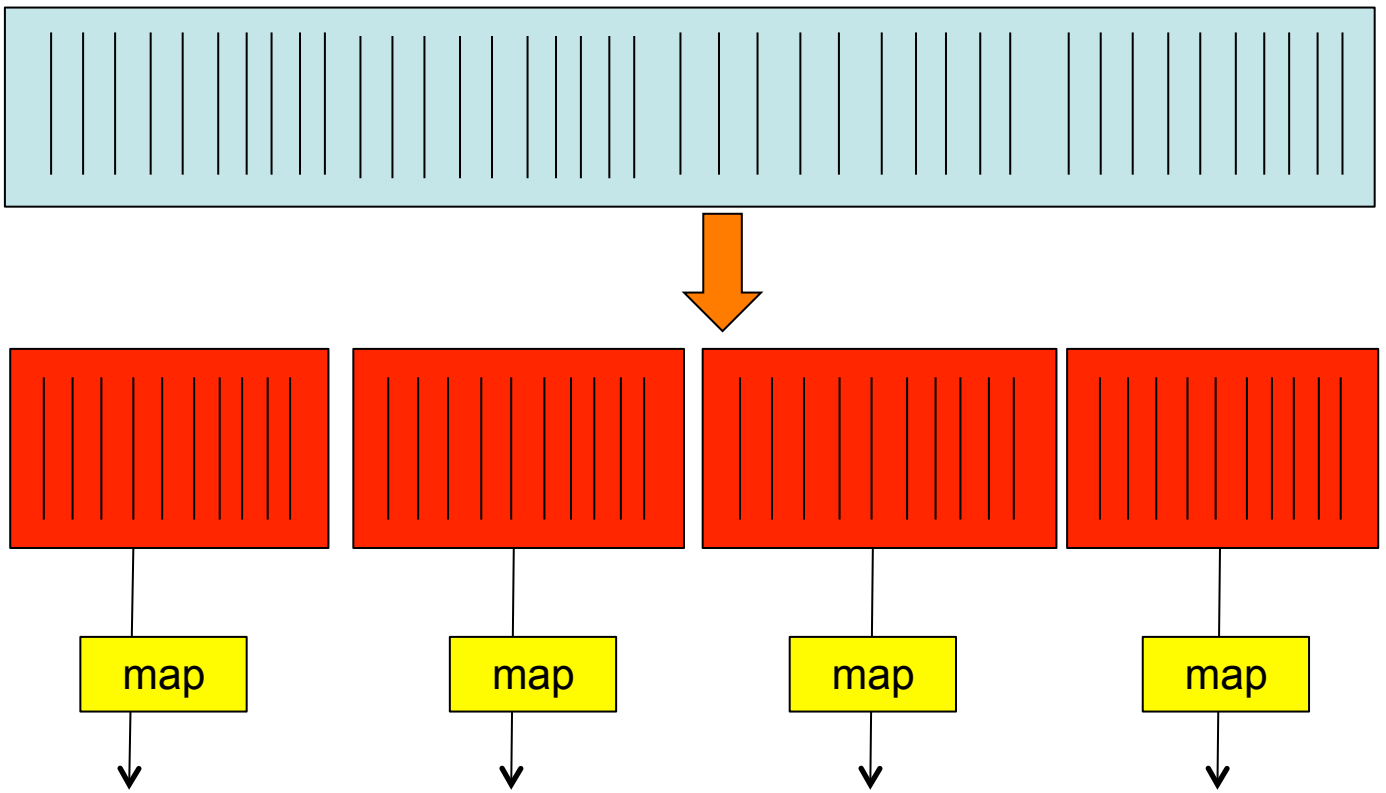
(people, 24)
(task, 14)
(receipt, 45)

(people, 13)
(task, 24)
(receipt, 76)



(people, 47)
(task, 72)
(receipt, 144)

Task: Word frequency across all documents



You have 10^6s of docs

Distribute the docs among k machines

For each doc, f returns a set of (word, freq) pairs

- *We do not want a bunch of histograms –we want one big histogram*
- *How can a single computer have access to every occurrence of a given word regardless of the document it appears in?*

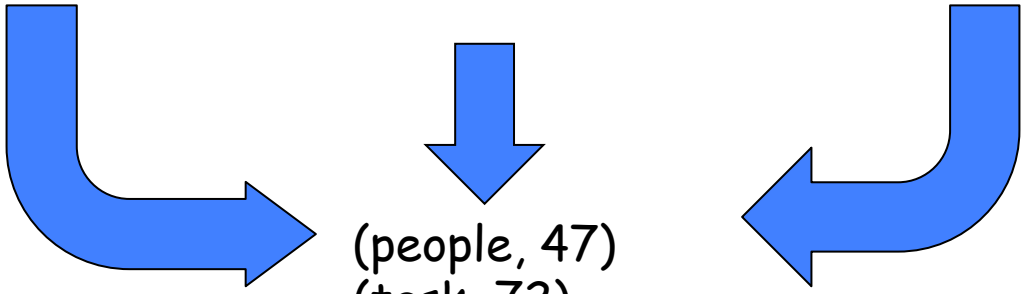
Slides from "Introduction to Data Science", Coursera, Bill Howe

New Task: Find word frequency across all documents

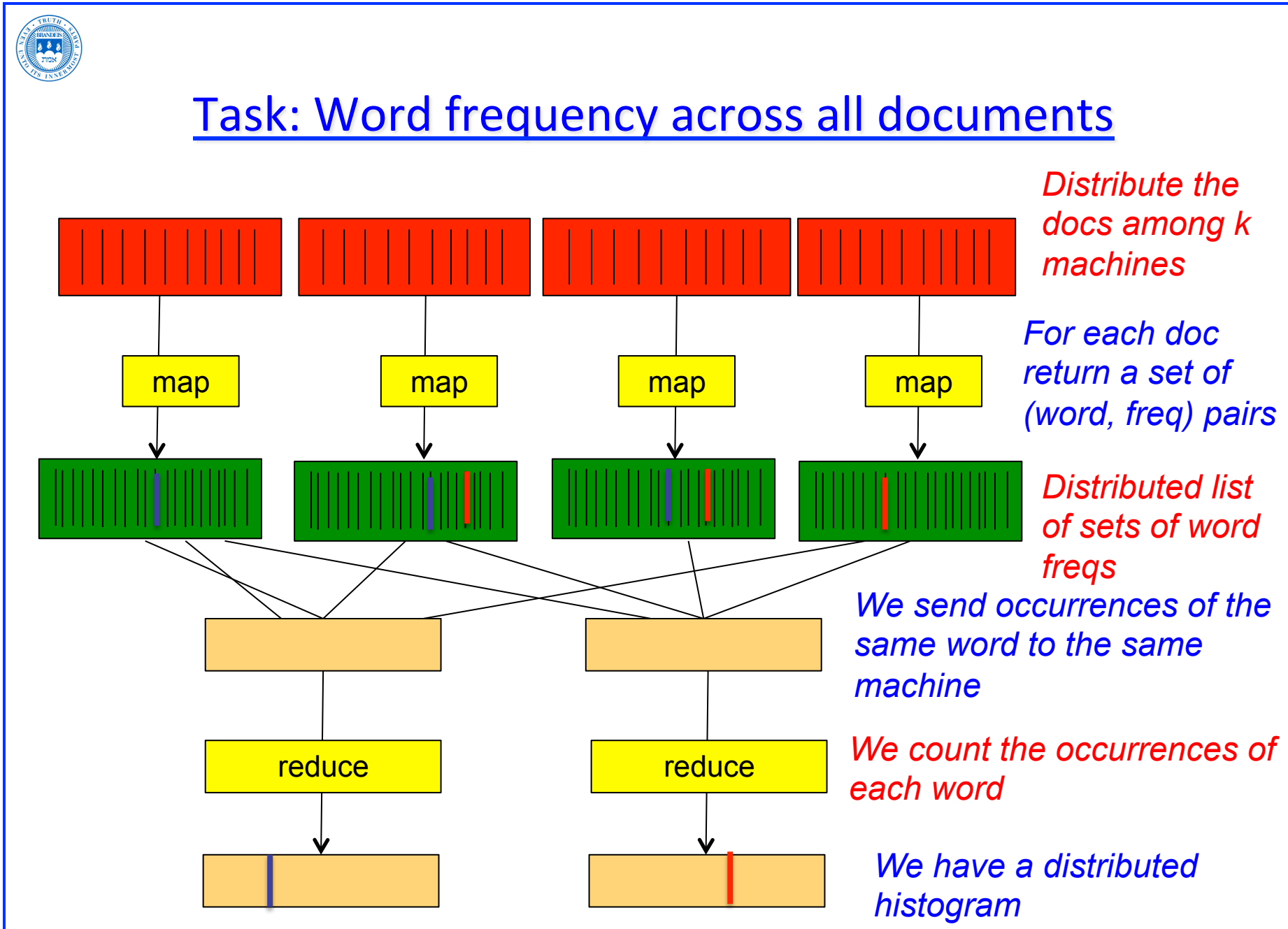
(people, 10)
(task, 34)
(receipt, 23)

(people, 24)
(task, 14)
(receipt, 45)

(people, 13)
(task, 24)
(receipt, 76)



(people, 47)
(task, 72)
(receipt, 144)



Task: Word frequency across all documents

Distribute the docs among k machines

For each doc return a set of (word, freq) pairs

Distributed list of sets of word freqs

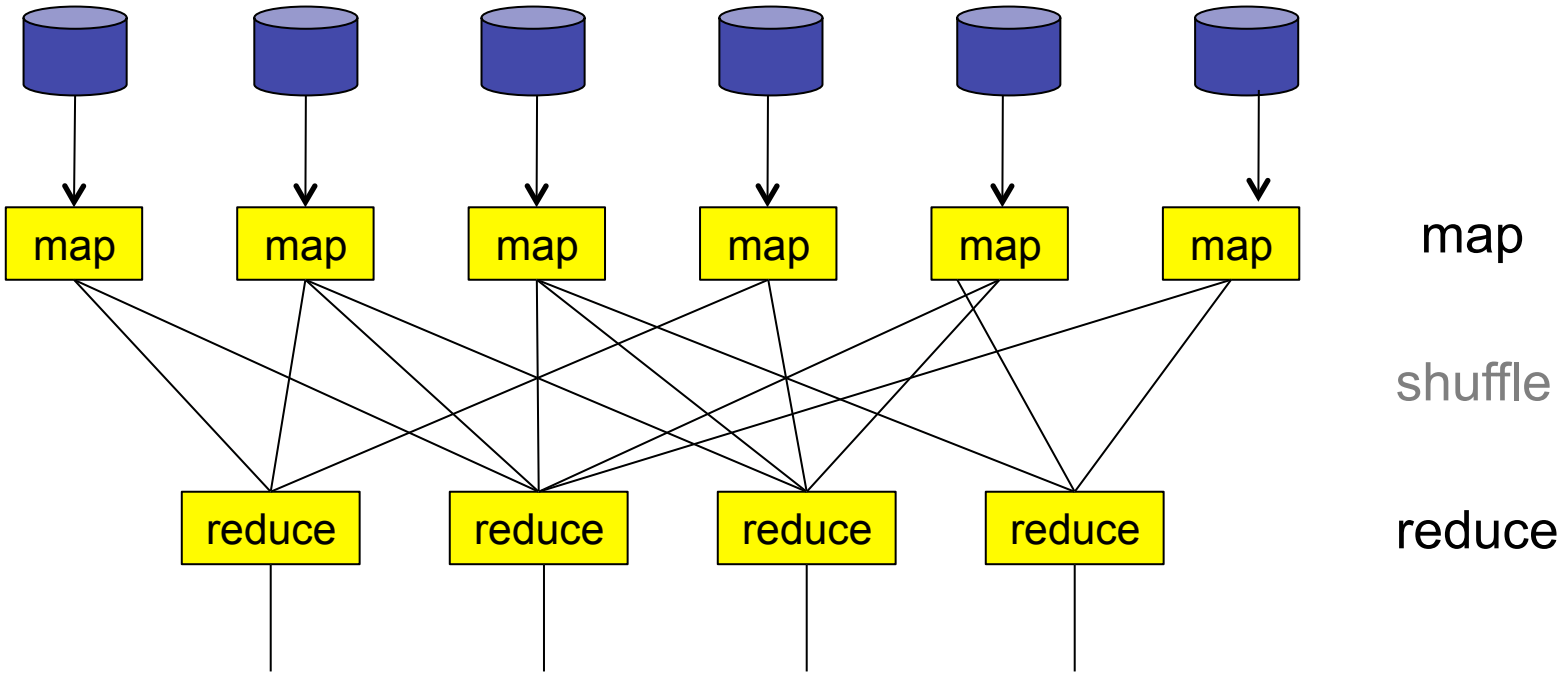
We send occurrences of the same word to the same machine

We count the occurrences of each word

We have a distributed histogram

Slides from "Introduction to Data Science", Coursera, Bill Howe

Map Reduce Framework



Slides from “Introduction to Data Science”, Coursera, Bill Howe

MapReduce in a Nutshell

- Here is the framework in simple terms:
  - Read lots of data.
  - **Map**: extract something that you care about from each record.
  - Shuffle and sort.
  - **Reduce**: aggregate, summarize, filter, or transform.
  - Write the results.

Map Reduce Programming Model

- Input/Output: each a set of key/value pairs
- Programmer specifies two functions:

map(in_key, in_value)->list (out_key, intemediate_value)

- Processes input key/value pair
- Produces set of intermediate pairs

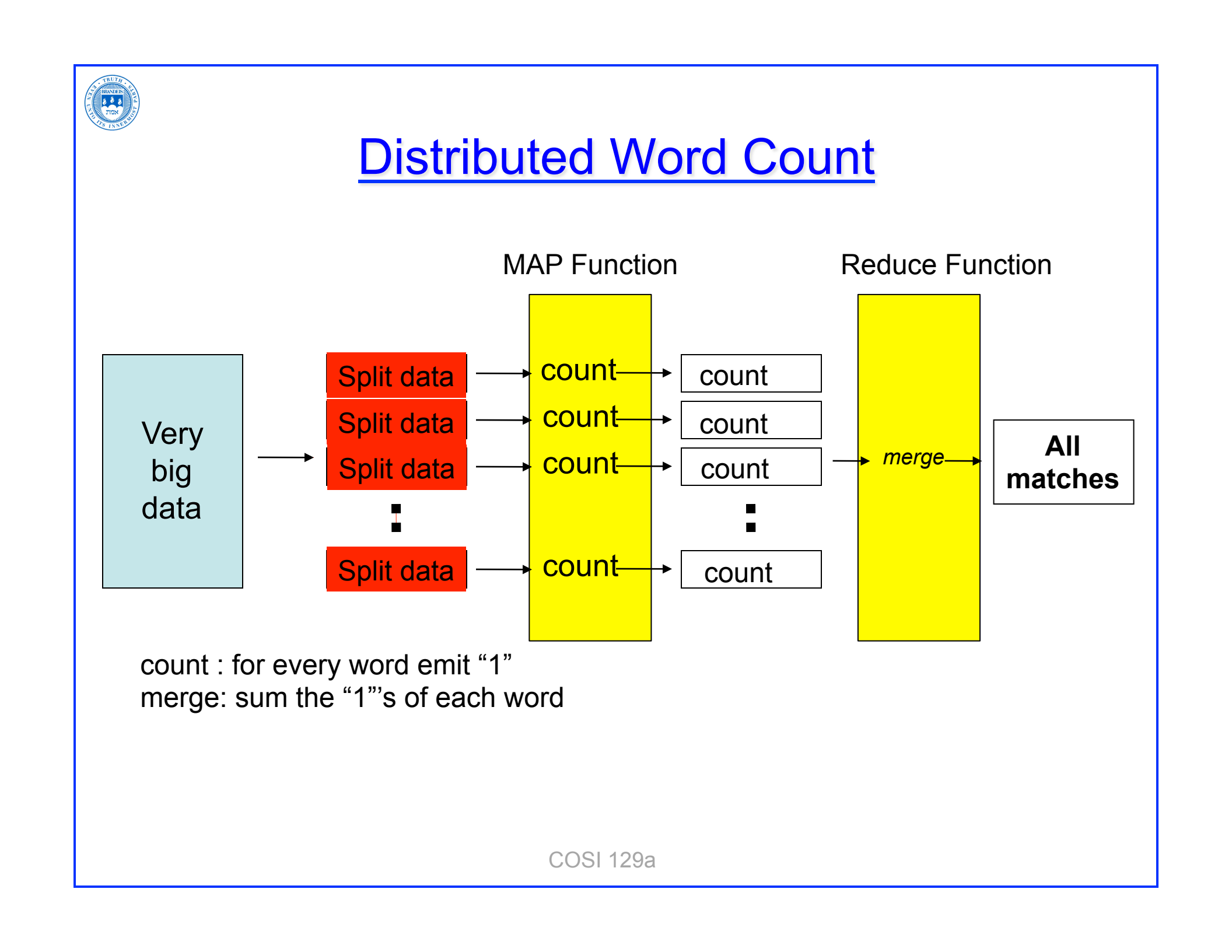
reduce(out_key, list (intemediate_pairs))->list (out_value)

- Combines all intermediate values for a particular key
- Produces a set of merged output values (usually just one)

MapReduce in a Nutshell

- Given:
  - a very large dataset
  - a well-defined computation to be performed on elements of this dataset (preferably, in a parallel fashion on a large cluster).

- MapReduce programming model:
  - Just express what you want to compute (map() & reduce()).
  - Don't worry about parallelization, fault tolerance, data distribution, load balancing (MapReduce takes care of these).
  - What changes from one application to another is the actual computation; the programming structure stays similar.



Distributed Word Count

count : for every word emit “1”
merge: sum the “1”’s of each word

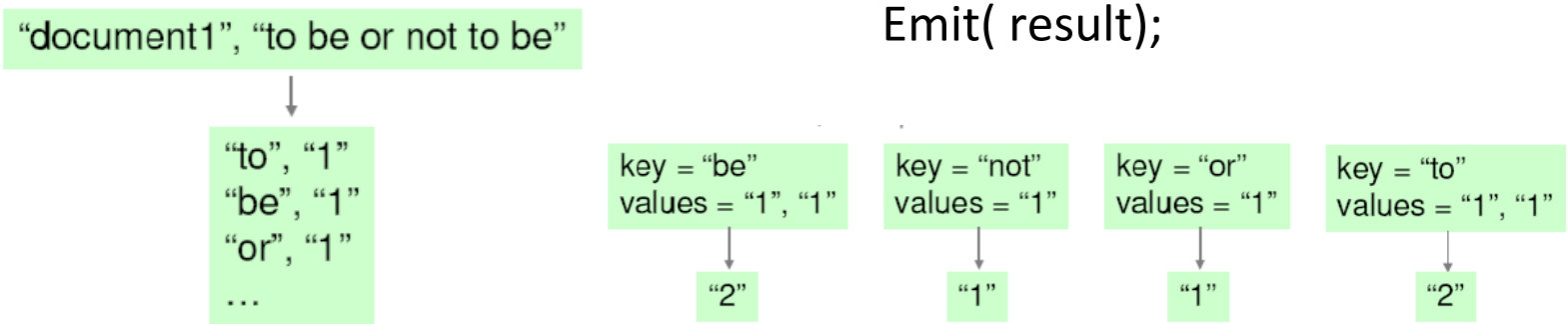
Distributed Word Count – MR Program

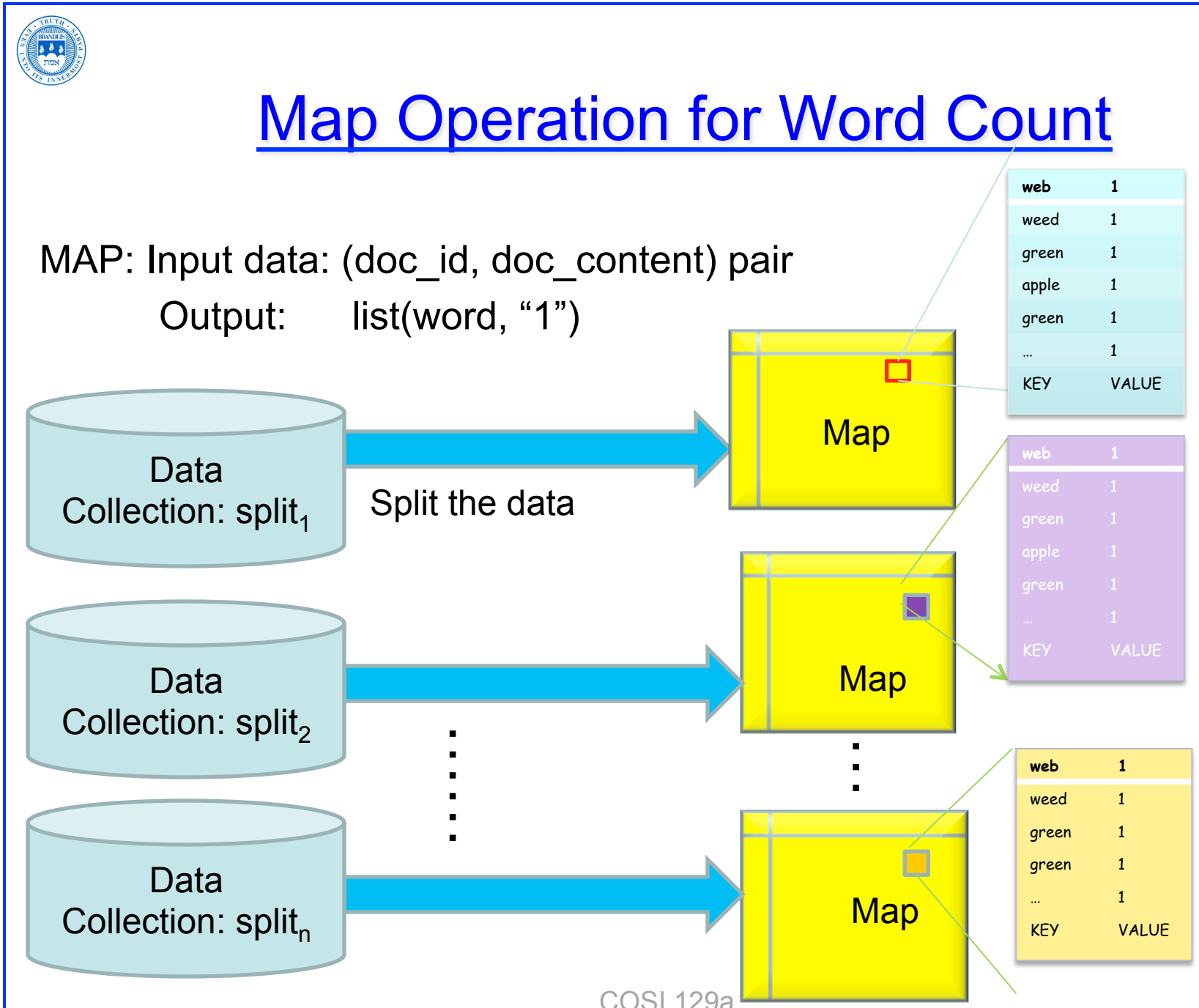
map$(k_1, v_1) \rightarrow$ **list**(k_2, v_2)

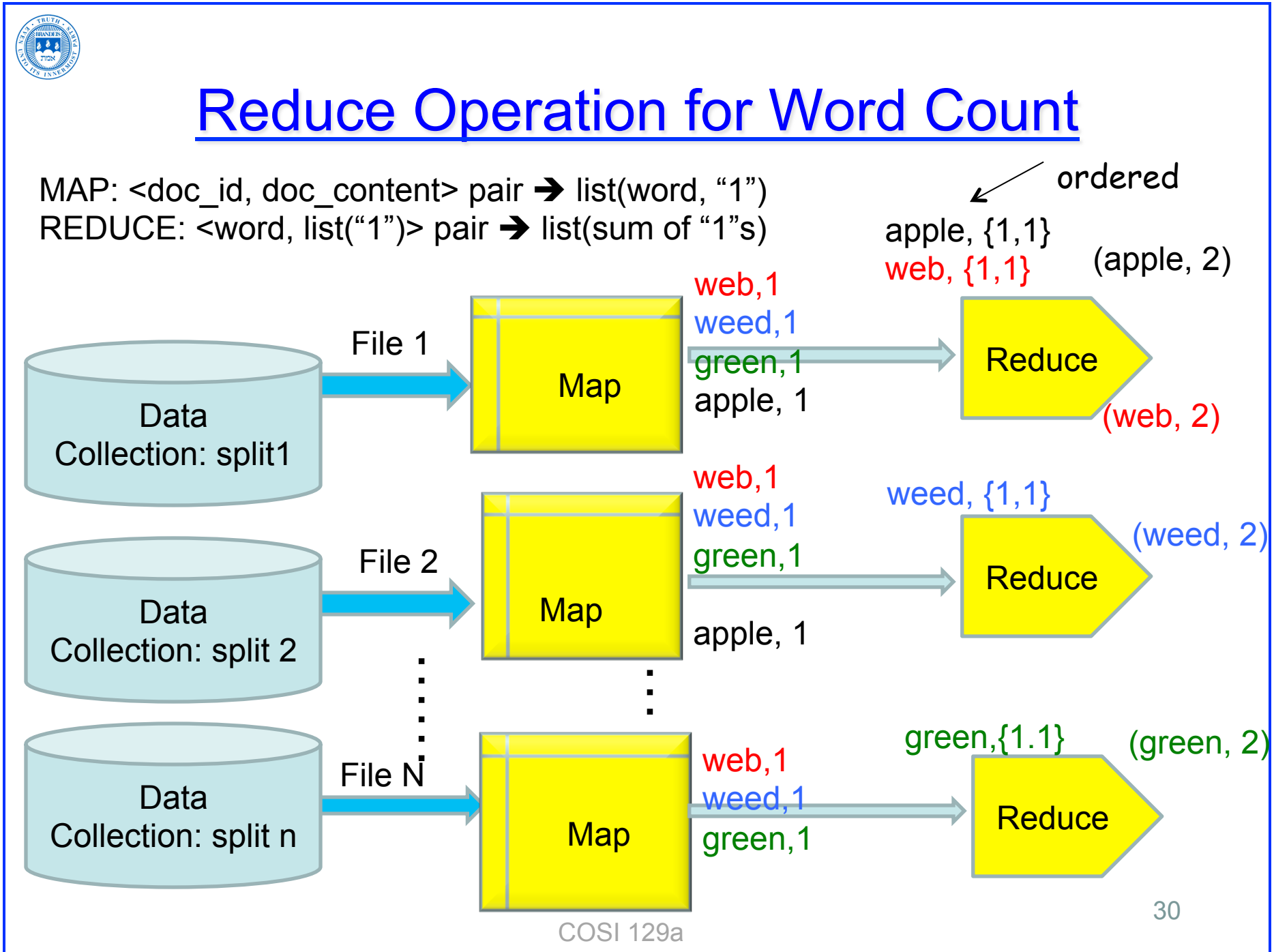
reduce$(k_2,$ **list**$(v_2)) \rightarrow$ **list**(v_2)

```
map (String key, String value):
    // key: document name
    // value: document contents
    for each word w in value:
        EmitIntermediate(w, 1);
```

```
reduce (String key, Iterator values):
    // key: a word
    // values: a list of counts
    int result = 0;
    for each v in values:
        result += v;
    Emit( result);
```

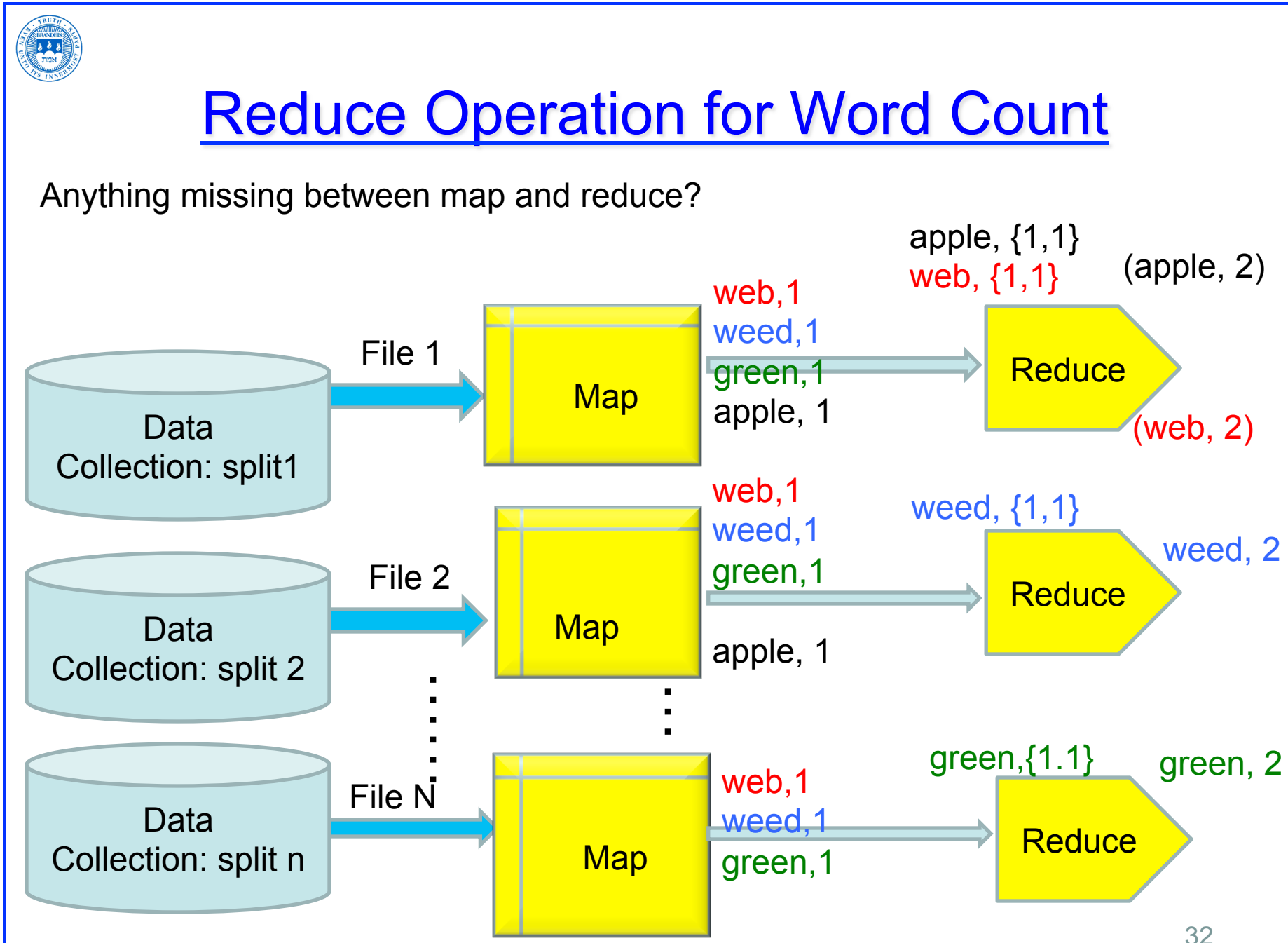



Map Operation for Word Count
MAP: Input data: (doc_id, doc_content) pair
Output: list(word, "1")
Data Collection: split1
Data Collection: split2
Data Collection: splitn
Split the data
Map
Map
Map
web 1
weed 1
green 1
apple 1
green 1
... 1
KEY VALUE
web 1
weed 1
green 1
apple 1
green 1
... 1
KEY VALUE
web 1
weed 1
green 1
green 1
... 1
KEY VALUE


Reduce Operation for Word Count
MAP: <doc_id, doc_content> pair ➔ list(word, "1")
REDUCE: <word, list("1")> pair ➔ list(sum of "1"s)
ordered
apple, {1,1}
web, {1,1}
(apple, 2)
Data Collection: split1
File 1
Map
web,1
weed,1
green,1
apple, 1
Reduce
(web, 2)
Data Collection: split 2
File 2
Map
web,1
weed,1
green,1
apple, 1
weed, {1,1}
Reduce
(weed, 2)
Data Collection: split n
File N
Map
web,1
weed,1
green,1
green,{1.1}
Reduce
(green, 2)

Reduce Abstraction

- Input: intermediate (key_2, (list($value_2$))) pairs
- Output: final value(s) for each intermediate key
  - Output: list($value_2$)
- Starting pairs are sorted by key
  - So all values with the same key will come one after the other
- Iterator supplies the values for a given key to the Reduce function.



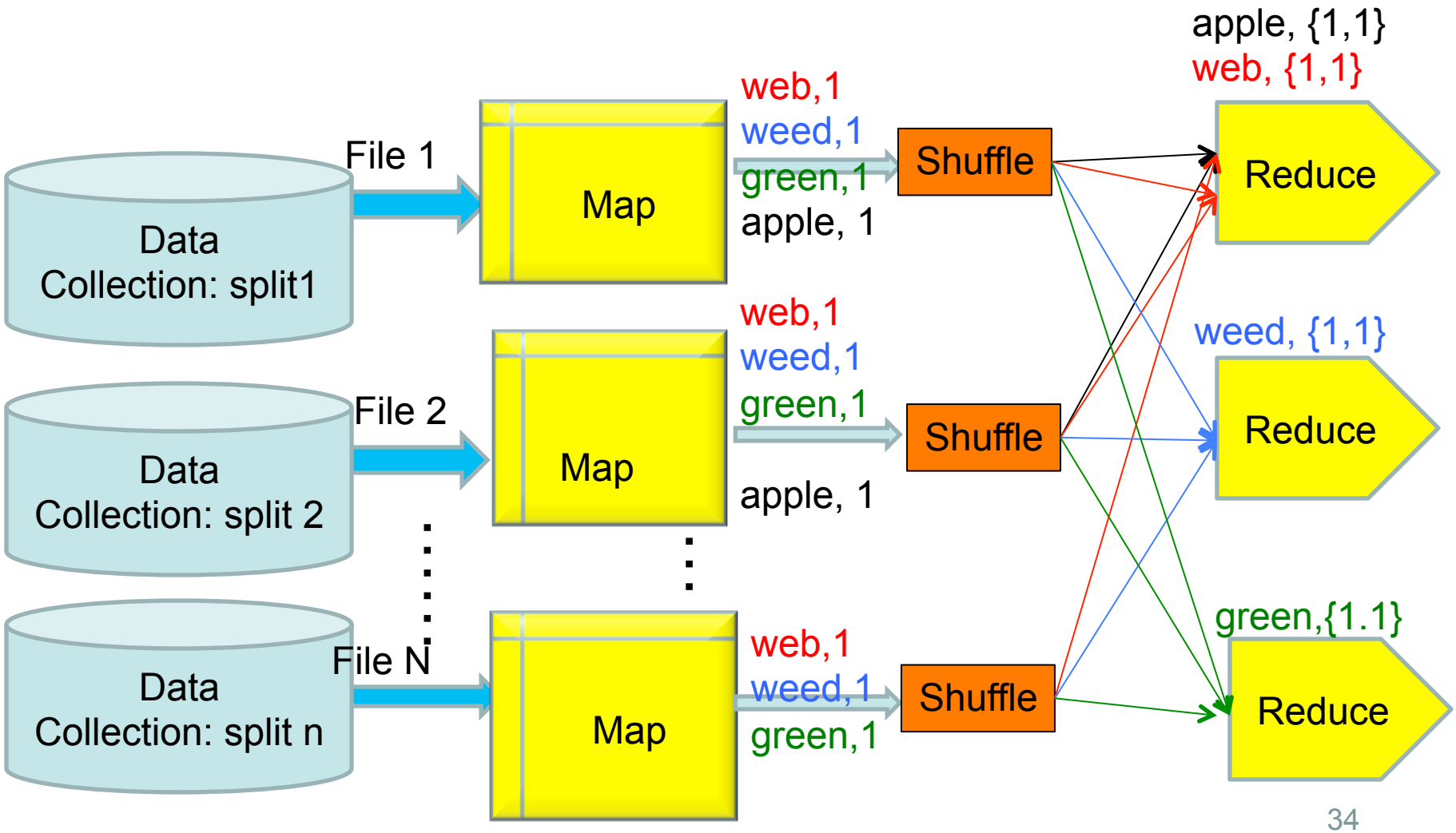
Reduce Operation for Word Count

Anything missing between map and reduce?

Shuffle Phase

- Shuffle phase
  - Groups together all intermediate values associated with the same intermediate key, key_2, and passes them to the Reduce function
  - If we have R reducers, then we partition the intermediate key space into R pieces using a partitioning function
    - E.g., hash(key_2) mod R
- Shuffle function is provided by the MapReduce library

Putting all together….



MapReduce Parallelization

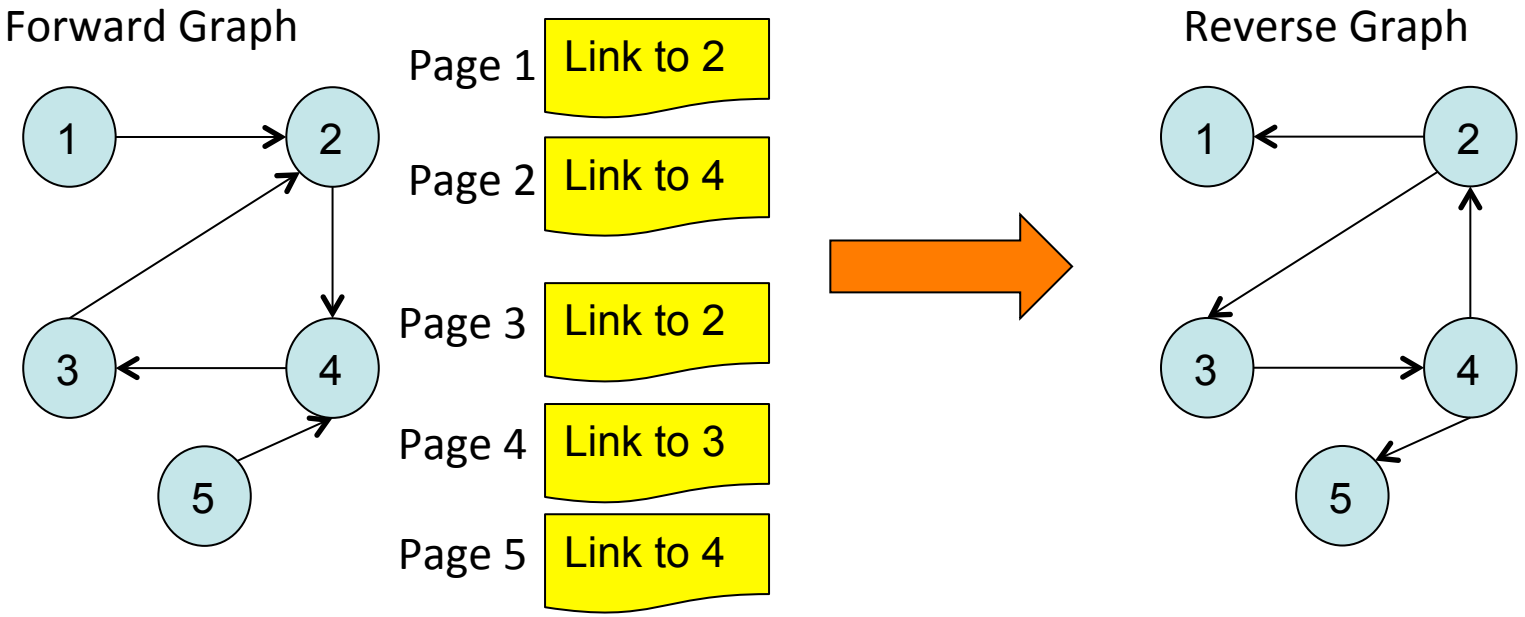
- map() functions run in parallel, creating different intermediate values from different input data sets.
- reduce() functions also run in parallel, each working on a different output key.
- All values are processed independently.
- **Bottleneck: The reduce phase can't start until the map phase is completely finished.**

New Task: Count URL Access Frequency

- Input file with web page requests
  - Log file format: (request_id, webpage_URL)
  - Want to have (URL, count), where count is how many times a URL has been requested in these log files
- How would you do it?
  - Map:
    - Input (file name, document contents)
    - Output: <webpage_URL, “1”>
  - Reduce:
    - Input <URL, list (“1”)>
    - Output count of “1” per each URL

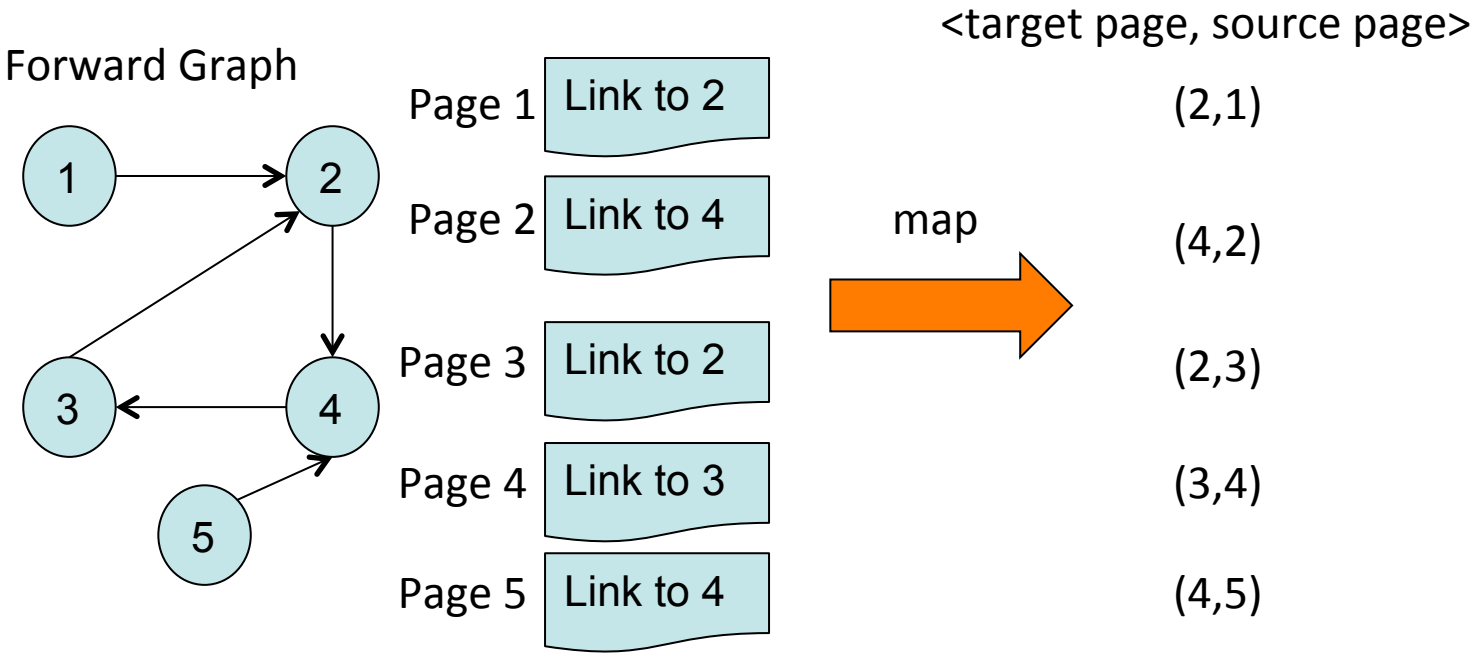
Reverse Web-link Graph

- Forward Web-Link Graph
- Nodes represent web pages
- Edges represent links from one page to the other

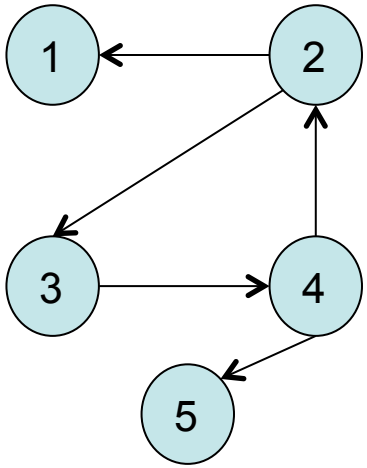


Reverse Web-link Graph

- Map
  - Input Key: document name, Input Value: document content
  - Output Key: target URL, Output Value: source URL



Reverse Web-link Graph



- Reduce
  - In: Key= target URL , Value: list of source URL
  - Out: <target, list(sourceURL)>

<target page, source page>

Page 1 Link to 2 → (2,1)

Page 2 Link to 4 → (4,2)

Page 3 Link to 2 → (2,3)

Page 4 Link to 3 → (3,4)

Page 5 Link to 4 → (4,5)

map

reduce

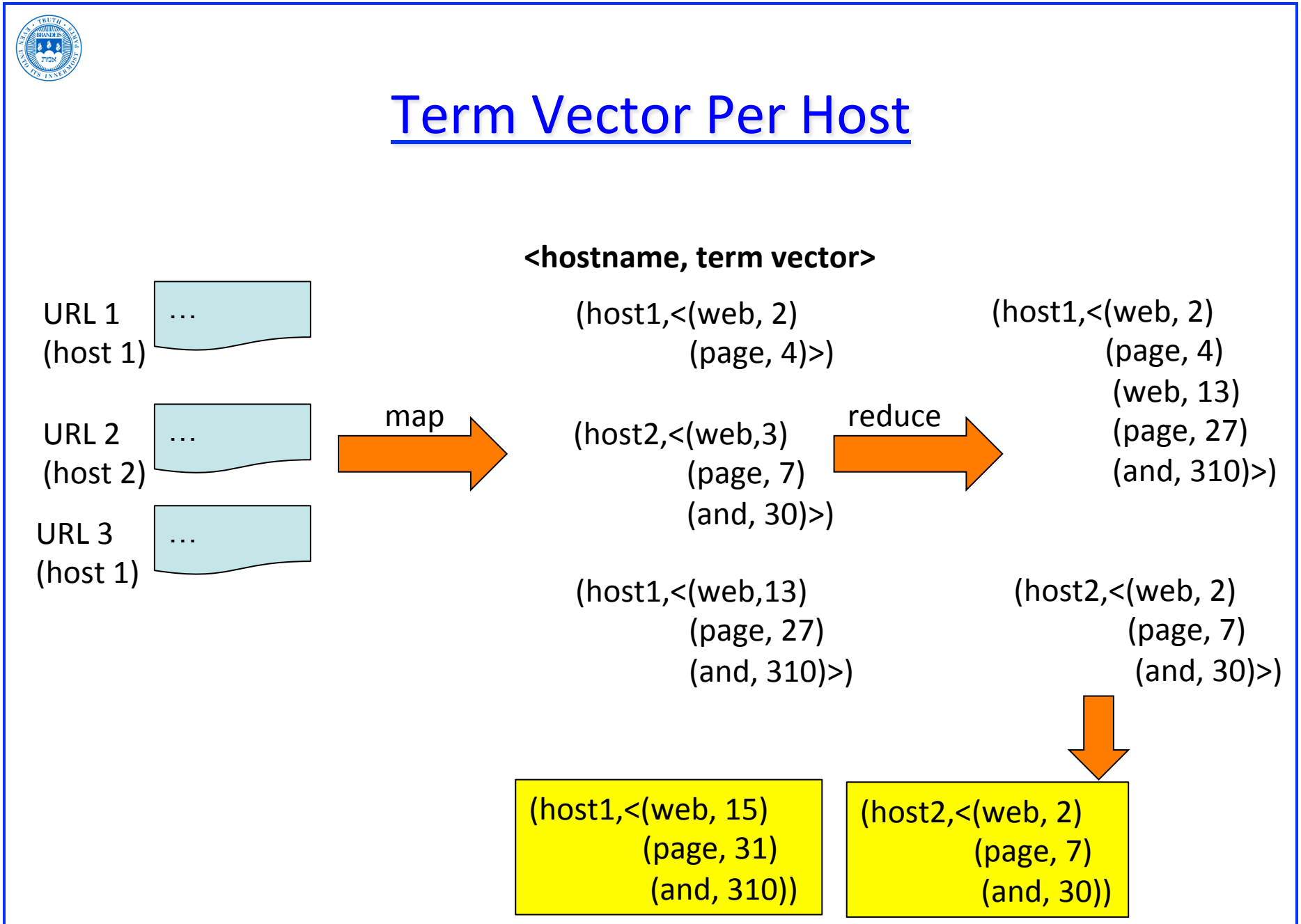
(2,<1,3>)

(3,<4>)

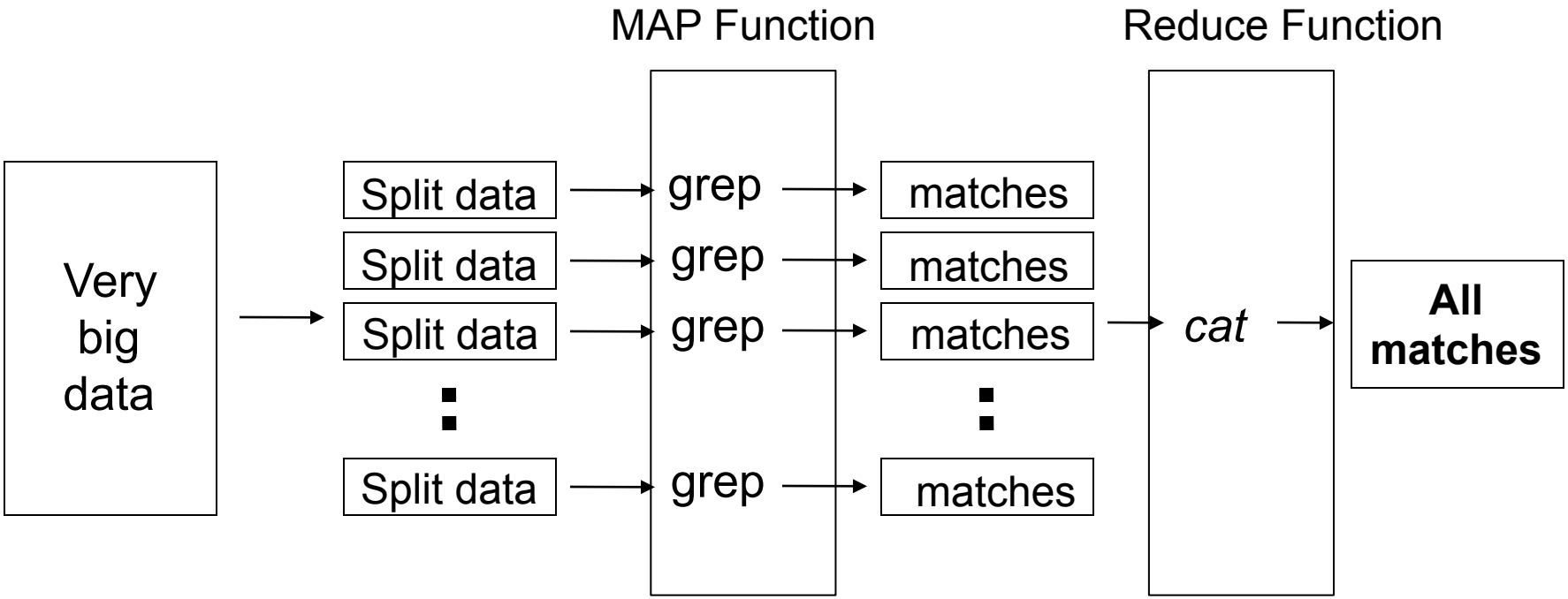
(4,<2,5>)

New Task: Term Vector Per Host

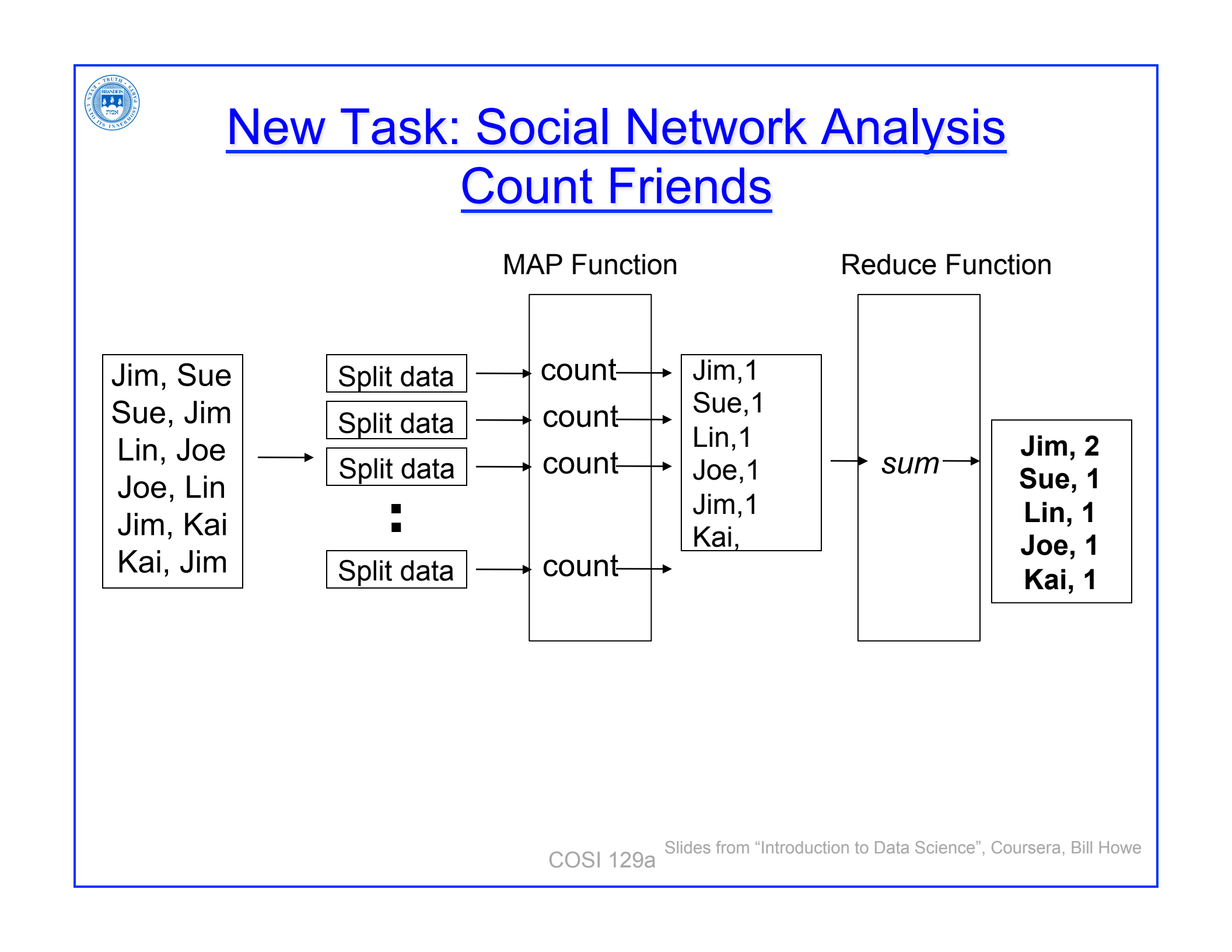
- Term Vector: <word, frequency> pairs
  - Only most important words (with frequency > x times)
- Input: web pages (URLs) to read from
  - Host: hostname is included in the web page URL
- Output: List of <hostname, term vector>
- Map:
  - In: Key: web page name, Value: web page content
  - Out: Key: Hostname and <term vector>
- Reduce
  - In: Key= hostname, Value: list<term vector>
  - Out: throws out infrequent words and emits <hostname, term vector>


Term Vector Per Host
<hostname, term vector>
URL 1
(host 1)
URL 2
(host 2)
URL 3
(host 1)
map
(host1,<(web, 2)
(page, 4)>)
(host2,<(web,3)
(page, 7)
(and, 30)>)
(host1,<(web,13)
(page, 27)
(and, 310)>)
reduce
(host1,<(web, 2)
(page, 4)
(web, 13)
(page, 27)
(and, 310)>)
(host2,<(web, 2)
(page, 7)
(and, 30)>)
(host1,<(web, 15)
(page, 31)
(and, 310))
(host2,<(web, 2)
(page, 7)
(and, 30))

New Task: Distributed Grep

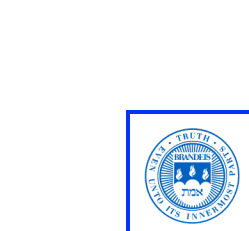


grep: for every line that includes the pattern return that line (match)
Cat : puts all the matches in the output file



New Task: Social Network Analysis
Count Friends

Slides from "Introduction to Data Science", Coursera, Bill Howe



New Task: Matrix Multiplication

1	3	4	-2
6	2	-3	1

x

1	-2
4	3
-3	-2
0	4

=

1	-9
23	4

C= A x B
A has dimensions L, M
B has dimensions, M, N

1x 1 + 3*4 + 4*(-3)+ (-2)*0 = 1

For each (i,k) in output table, sum (A[i,j) * B[j,k]) for all j in 1..M

Slides from "Introduction to Data Science", Coursera, Bill Howe

Matrix Multiplication in Map Reduce

C= A x B

A has dimensions L, M (2x4)

B has dimensions, M, N (4x2)

1	3	4	-2
6	2	-3	1

1	-2
4	3
-3	-2
0	4

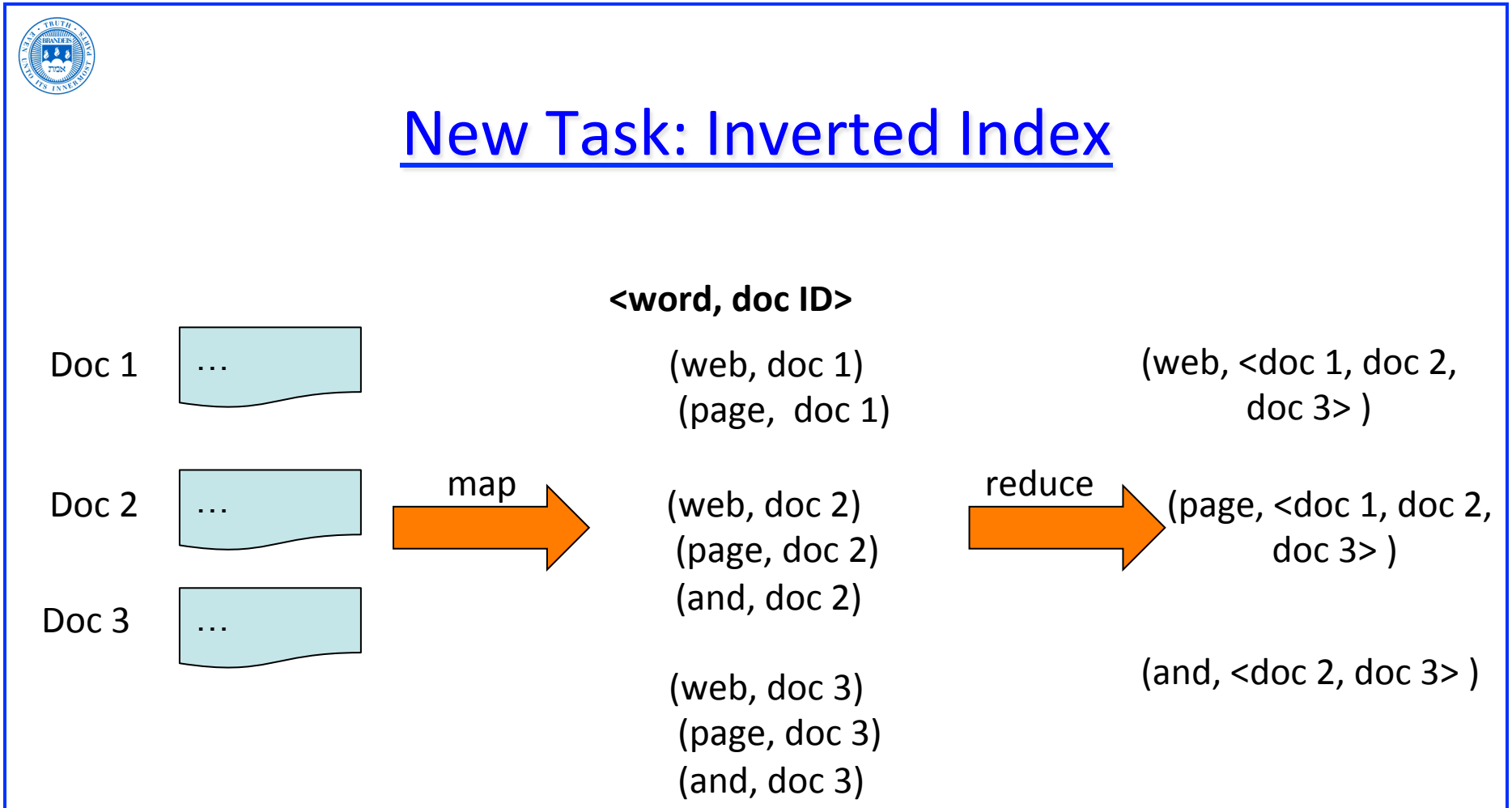
A(1,1) is used to get C (1,1) & C(1,2)
A(2,1) is used to get C(2,1) & C(2,2)
B(1,1) is used to get C (1,1) & C(2,1)
B(1,2) is used to get C(1,2) & C(2,1)

- Map:
  - for each element (i,j) in A emit ((i,k), A[i,j]) for k in 1..N
  - for each element (j,k) in B emit ((i,k), B[j,k]) for i in 1..L
- Reduce:
  - Key = (i,k)
  - Value = sum_j(A[i,j] * B[j,k])

Slides from "Introduction to Data Science", Coursera, Bill Howe

New Task: Inverted Index

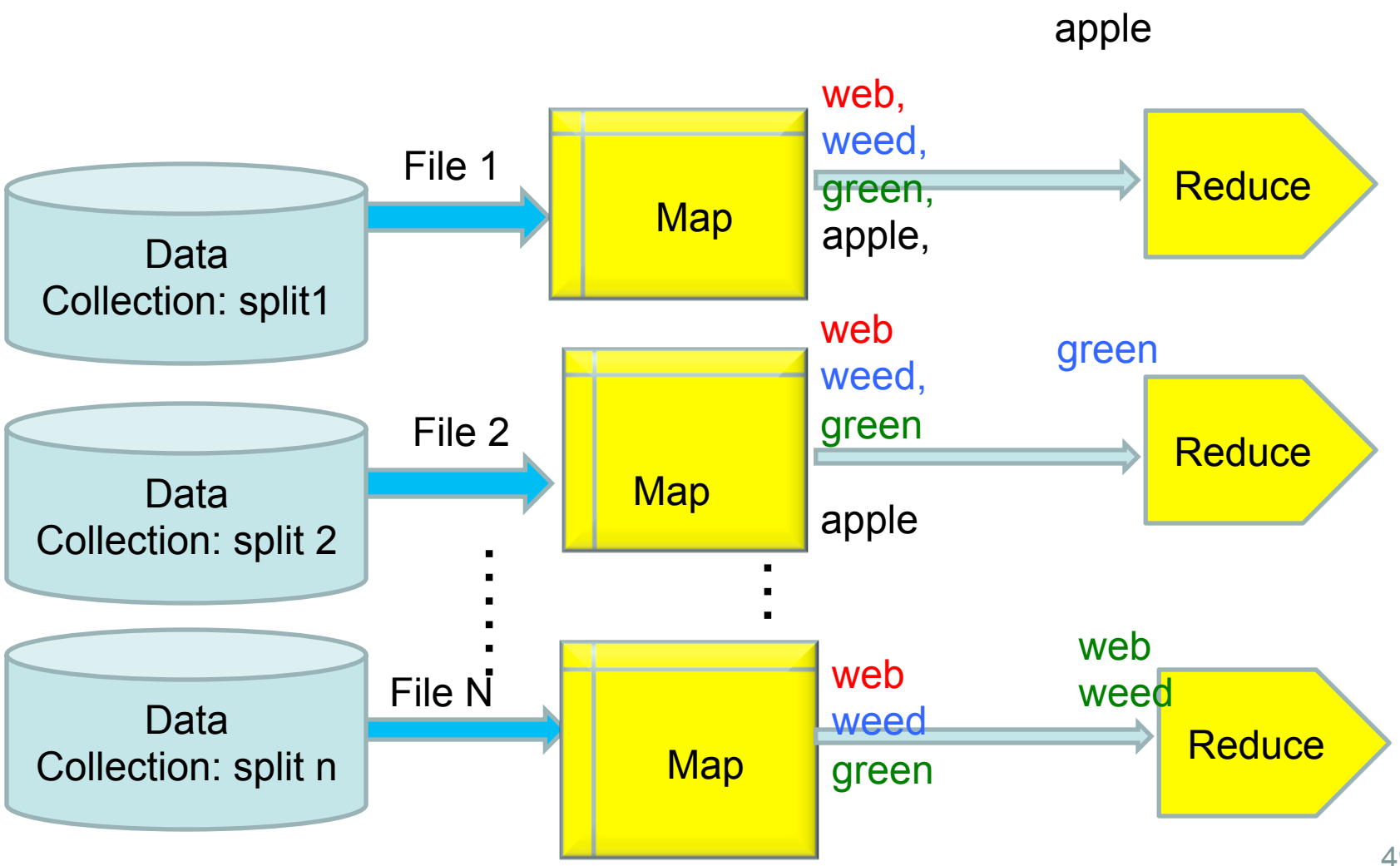
- For each word w, we store all the documents that contain w.
  - Identify each documents by its doc id
  - E.g., “people” -> {doc 1, doc 3, …}
    “task” -> {doc 4, doc 1, …}


New Task: Inverted Index
<word, doc ID>
Doc 1
Doc 2
Doc 3
…
…
…
map
(web, doc 1)
(page, doc 1)
(web, doc 2)
(page, doc 2)
(and, doc 2)
(web, doc 3)
(page, doc 3)
(and, doc 3)
reduce
(web, <doc 1, doc 2, doc 3>)
(page, <doc 1, doc 2, doc 3>)
(and, <doc 2, doc 3>)

New Task: Distributed Sort

- Input (key, record) pairs: sort by key
- Only need a map function:
  - For each record, emit <key, record> pair
- Reduce function emits all pairs unchanged
- Partition function: sorts the keys across the reducers
  - [a-c] to partition 1, [d-f] to partition 2, etc
- Pre-processing step: Collect sample of keys to collect info about the distribution of keys
  - Use them to compute split points for the final sorting pass

MR for Sort


apple
web,
weed,
green,
apple,
File 1
Data
Collection: split1
Map
Reduce
web
weed,
green
green
apple
File 2
Data
Collection: split 2
Map
Reduce
web
weed
File N
Data
Collection: split n
Map
web
weed
green
Reduce