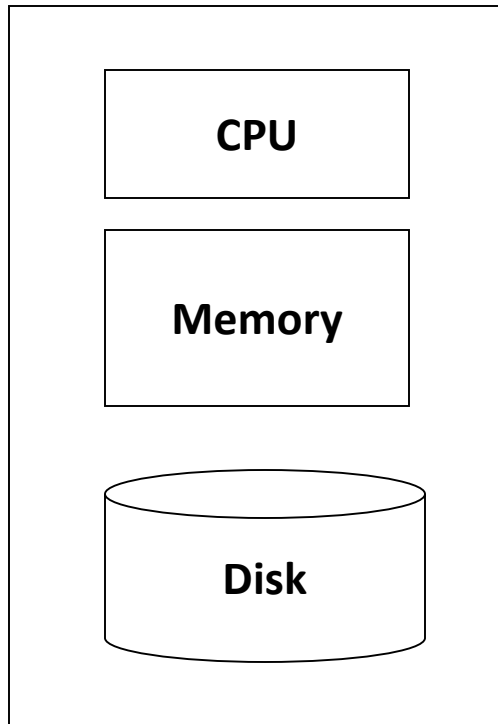


Map-Reduce

By Marina Barsky
Winter 2017, University of Toronto

Single Node Architecture



“Classical” Data Processing

We know how to do it efficiently

Block Nested Loop Join – single pass

$$B(R) + \frac{B(R)}{M-1} B(S)$$

if $\min(B(R), B(S)) \leq M-1$

Sort-Merge Join – 3 passes

$$3 * B(R) + 3 * B(S)$$

if $B(R) + B(S) < M^2$

Hash Join – 3 passes

$$3 * B(R) + 3 * B(S)$$

if $\min(B(R), B(S)) < M^2$

What if inputs are much-much larger?

What does *scalable* mean: operationally

In the past:

“Works even if data does not fit in main memory on a single machine”

- *Out-of-core* – large parts of inputs and outputs are on disk
- External-memory algorithms
 - Small memory footprint
 - Data is brought in chunks to main memory and the results are written to a local disk
- You have a guarantee that the algorithm will terminate

What does *scalable* mean: operationally

Now:

“Can make use of 1000s
cheap computers”

- Started from 2000s – no matter how big your server was, you were not able to bring data fast enough to memory from disk
- Use 1000s computers and apply them all to the same problem

Scale out (parallelize) vs. **scale up** (adding more memory)

What does *scalable* mean: algorithmically

In the past:

if you have N data items, you
perform no more than N^m
operations

- $O(N^m)$ - Polynomial-time algorithm \rightarrow tractable \rightarrow scalable
- $O(m^N)$ - Exponential \rightarrow not scalable \rightarrow not for big inputs, processing time increases too fast

What does *scalable* mean: algorithmically

Now:

if you have N data items, you perform no more than N^m/K operations for some large K

- Polynomial-time algorithms must be parallelizable

What does *scalable* mean: algorithmically

Future:

if you have N data items, you
perform no more than $N \log N$
operations

- Data is streaming (Large Synoptic Survey telescope – 30 TB/night)
- You have no more than one pass over the data (N) – make this pass count
- Insert data into some sort of compressed index ($\log N$)

You call an algorithm *scalable*

- In the past: polynomial-time algorithms
- • Now: parallel polynomial-time algorithms
- In the future: streaming algorithms

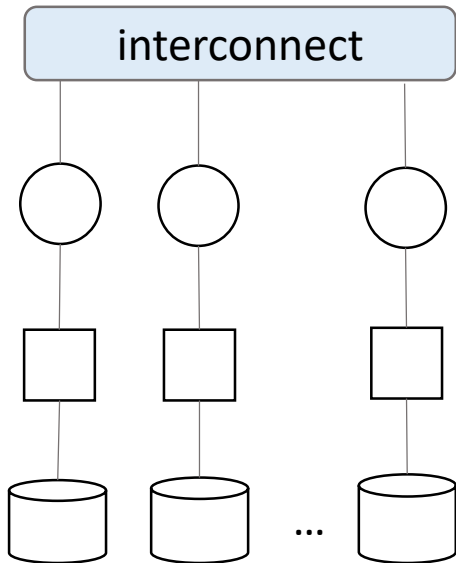
Motivation: Google Example

- 20+ billion web pages x 20KB = 400+ TB
- 1 computer reads 30-35 MB/sec from disk
 - **~4 months to just read the web!**
- ~1,000 hard drives to store the web
- Takes even more to **do** something useful with the data!
- **A standard architecture for such problems:**
 - Cluster of commodity Linux nodes
 - Commodity network (Ethernet) to connect them

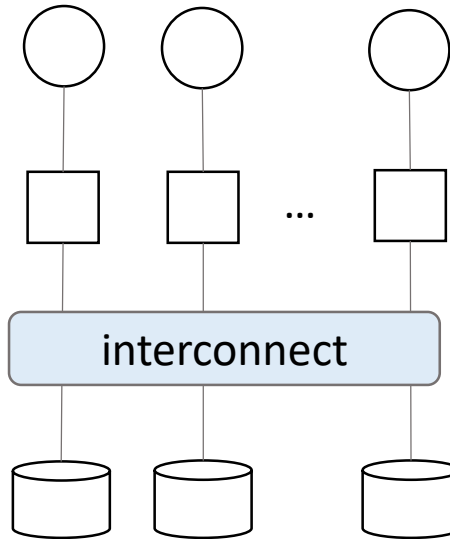
Scalability of parallel architectures

Logical multi-processor database designs

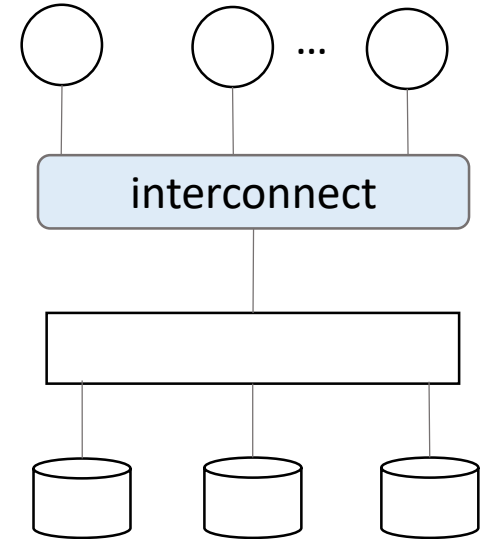
Shared nothing



Shared disk



Shared memory

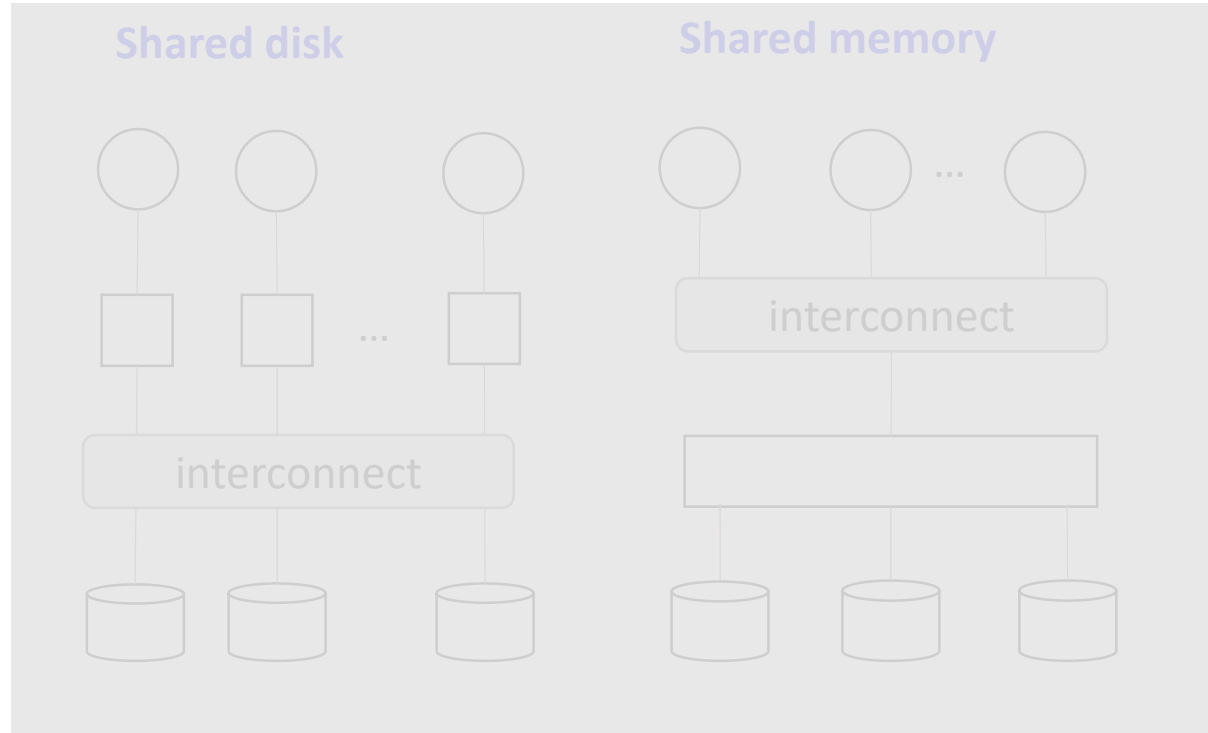
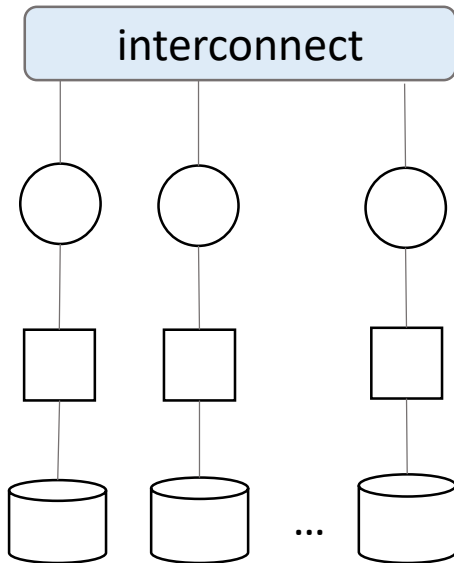



 =disk  =memory  =processor

Scalability of parallel architectures

Logical multi-processor database designs

Shared nothing



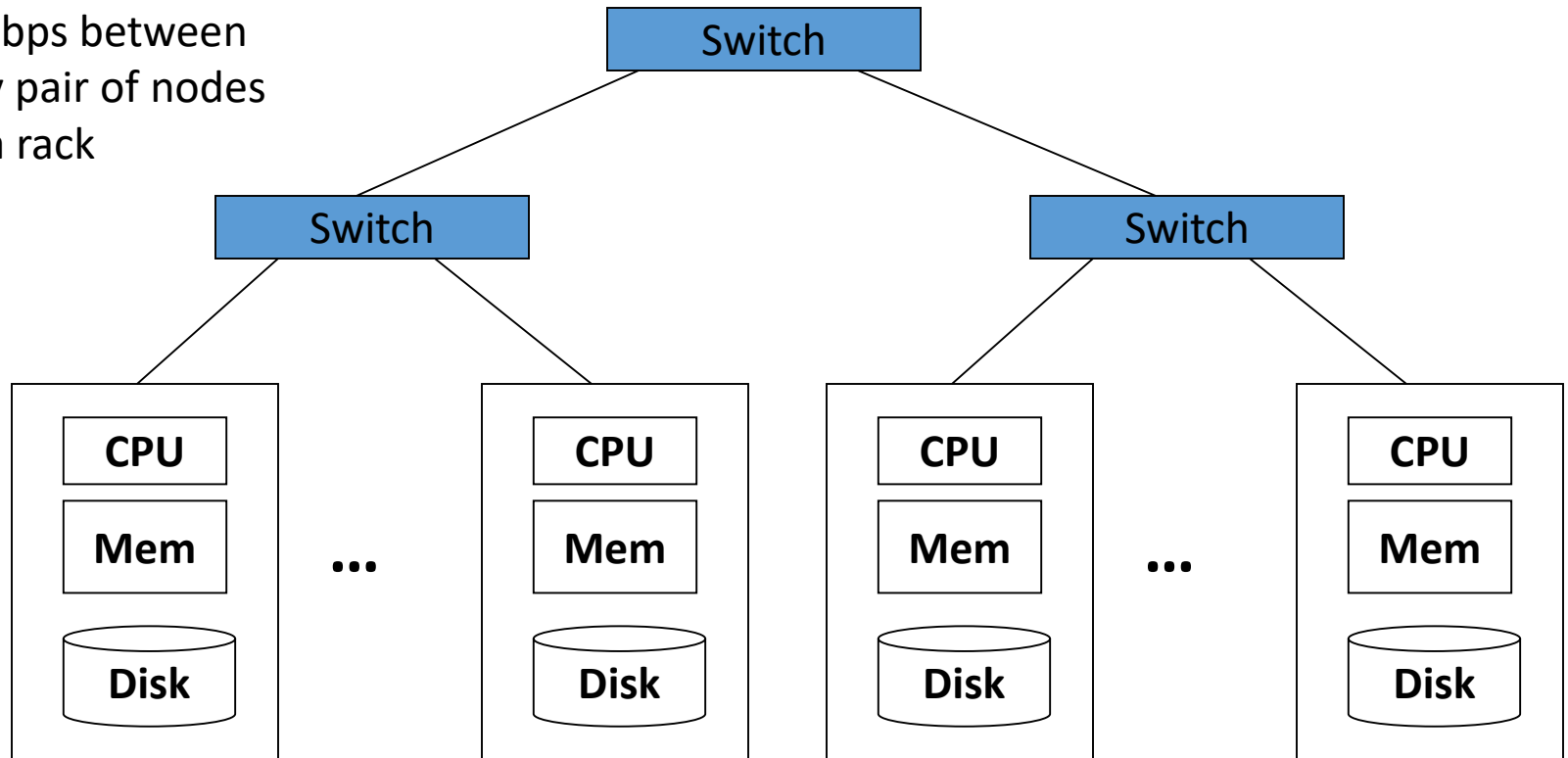
 =disk  =memory  =processor

Only **shared nothing** architecture truly scales, others reach the bottleneck of accessing the same data by multiple processors

Cluster Architecture

2-10 Gbps backbone between racks

1 Gbps between
any pair of nodes
in a rack



Each rack contains 16-64 nodes

In 2011 it was estimated that Google had 1M machines, <http://bit.ly/Shh0RO>



Processing (really) big inputs

- Scalability of algorithms
- Inherently parallelizable tasks
- Distributed file system
- Map-reduce computation
- Practice

Example 1: find matching DNA sequences

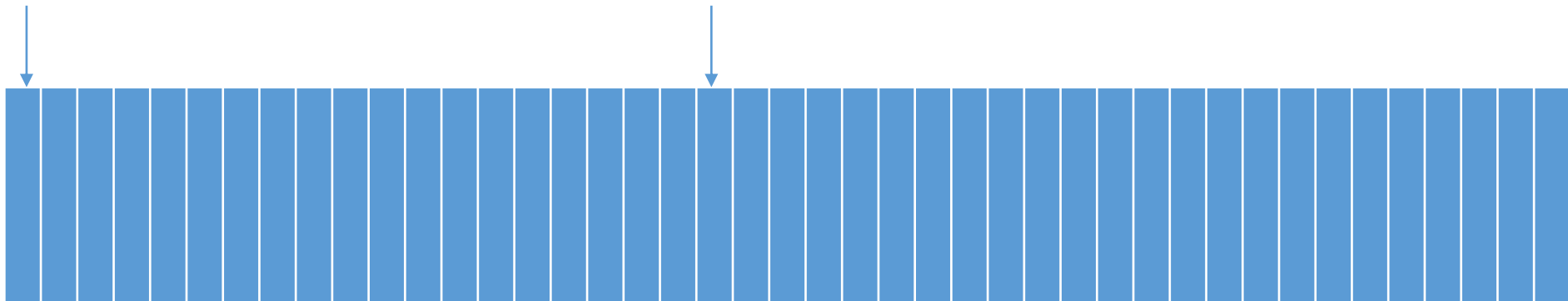
- Given a set of short sequences:
- Find all sequences equal to *GATTACGATATTA*

Search algorithm I

TAAAAAATATTA

GATTACGATTA

GATTACGATTA

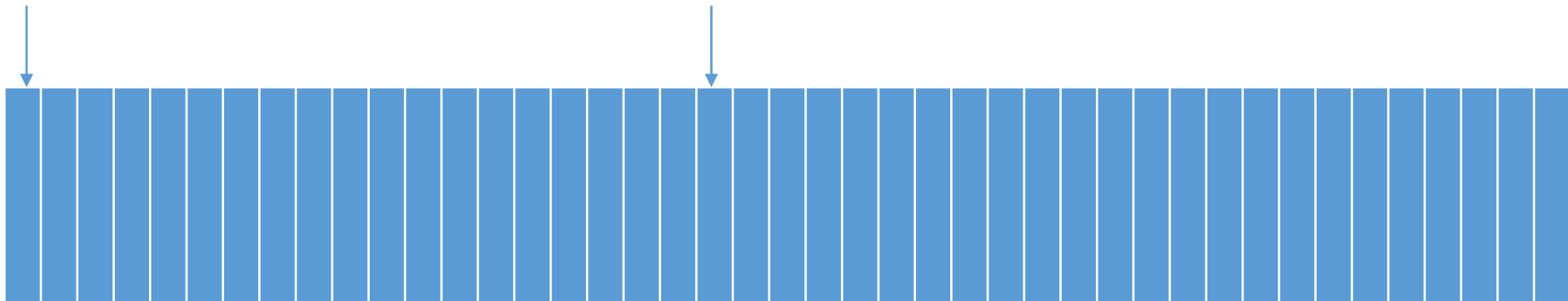


Search algorithm I

TAAAAAATATTA

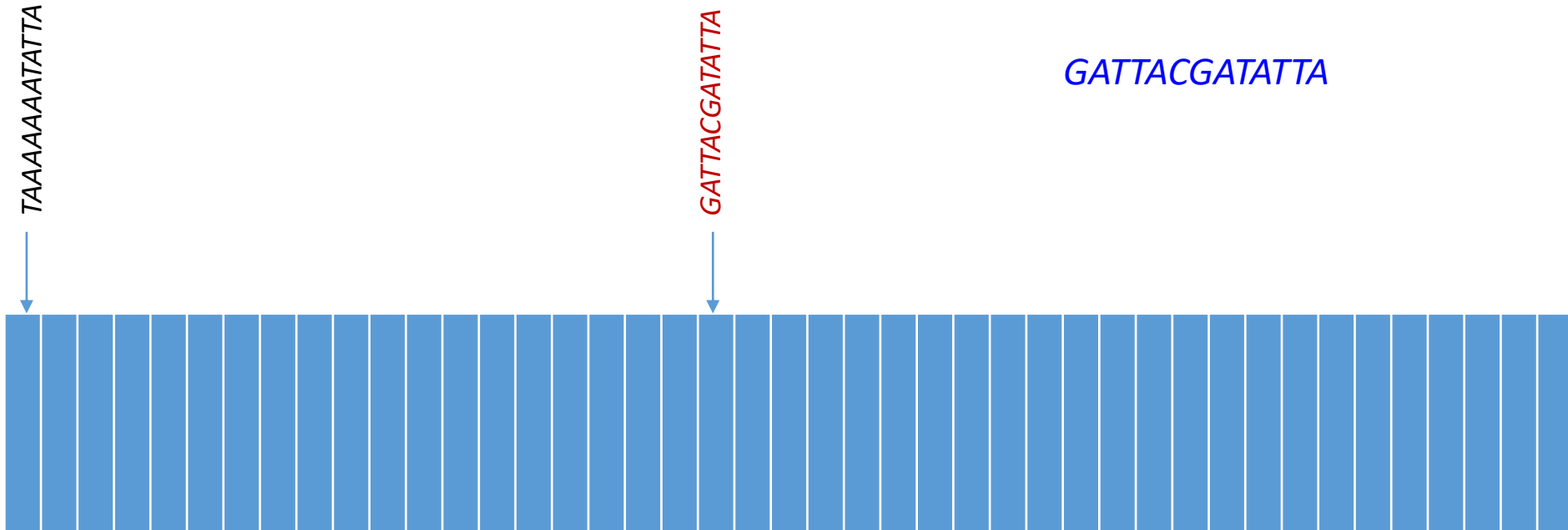
GATTACGATTA

GATTACGATTA



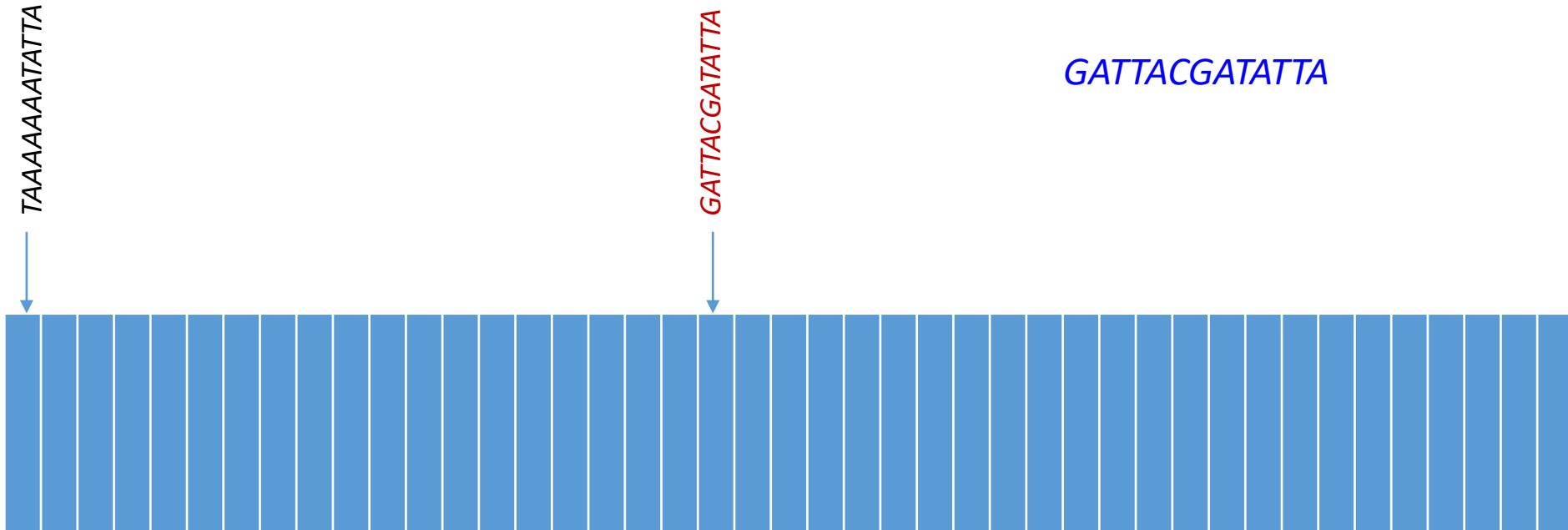
Step 20: found

Search algorithm I



$N = 40$ records \rightarrow 40 comparisons
 $O(N)$ algorithm

Search algorithm I

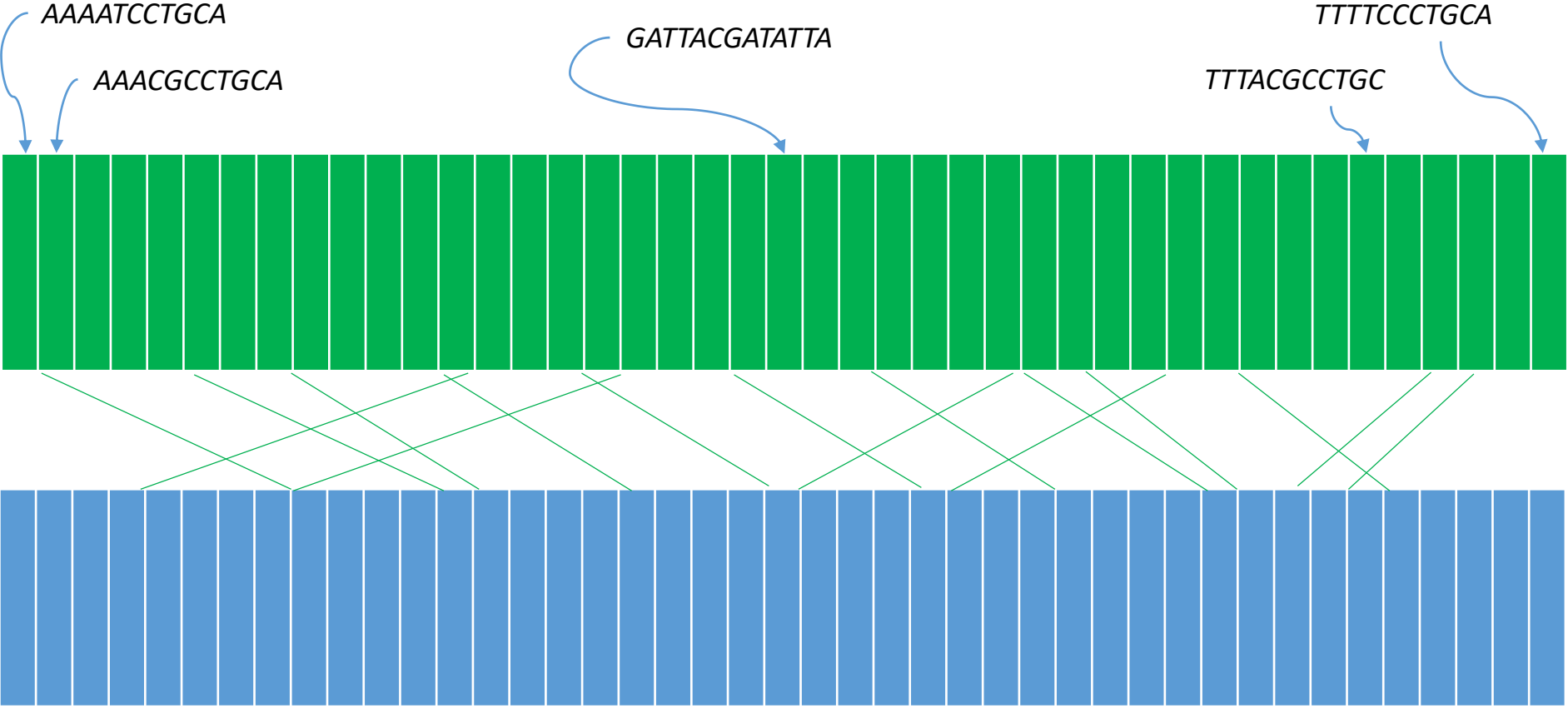


$N = 40$ records \rightarrow 40 comparisons
 $O(N)$ algorithm

Can we do any better?

Search algorithm II

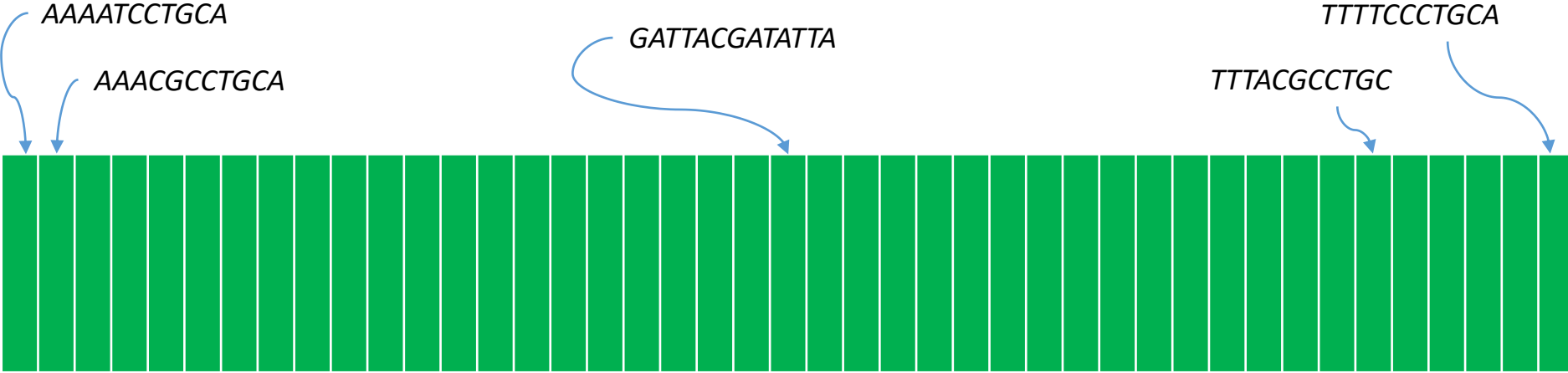
GATTACGATATTA



What if we **pre-sort** the sequences?

Search algorithm II

GATTACGATATTA



Binary search: log N time

Far better scalability !

Old-style scalability: implemented by DBMS

- Databases are proficient at “Needle in Haystack” problems – extracting small results from big datasets
- Guarantee that your query will always finish, regardless of dataset size
- Indexes are easily built and automatically used when appropriate
- You can take advantage of log N search without implementing index yourself. Most of the algorithmic work is done for you.

```
CREATE INDEX seq_idx ON sequences (seq)
```

```
SELECT seq FROM sequences  
WHERE seq = 'GATTACGATATTA'
```

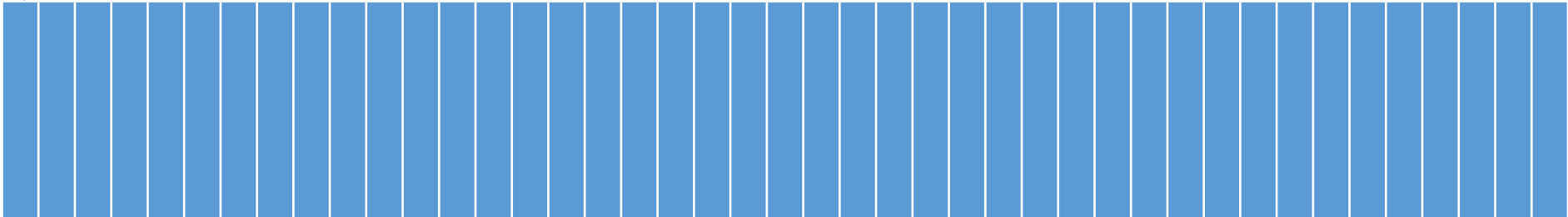
Example 2: read trimming

- Given a set of DNA reads – sequences of 100 characters long:
- Trim the final t (bp) characters of each sequence*
- Generate a new dataset of trimmed sequences

*The accuracy of sequencer drops abruptly after a certain length

Trim algorithm I

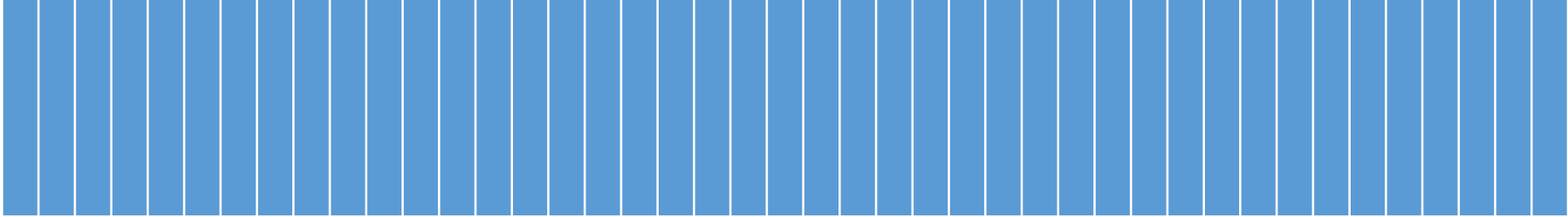
TAAAAAATATTA



- Time 0: *TAAAAAATATTA* → *TAAAAA*

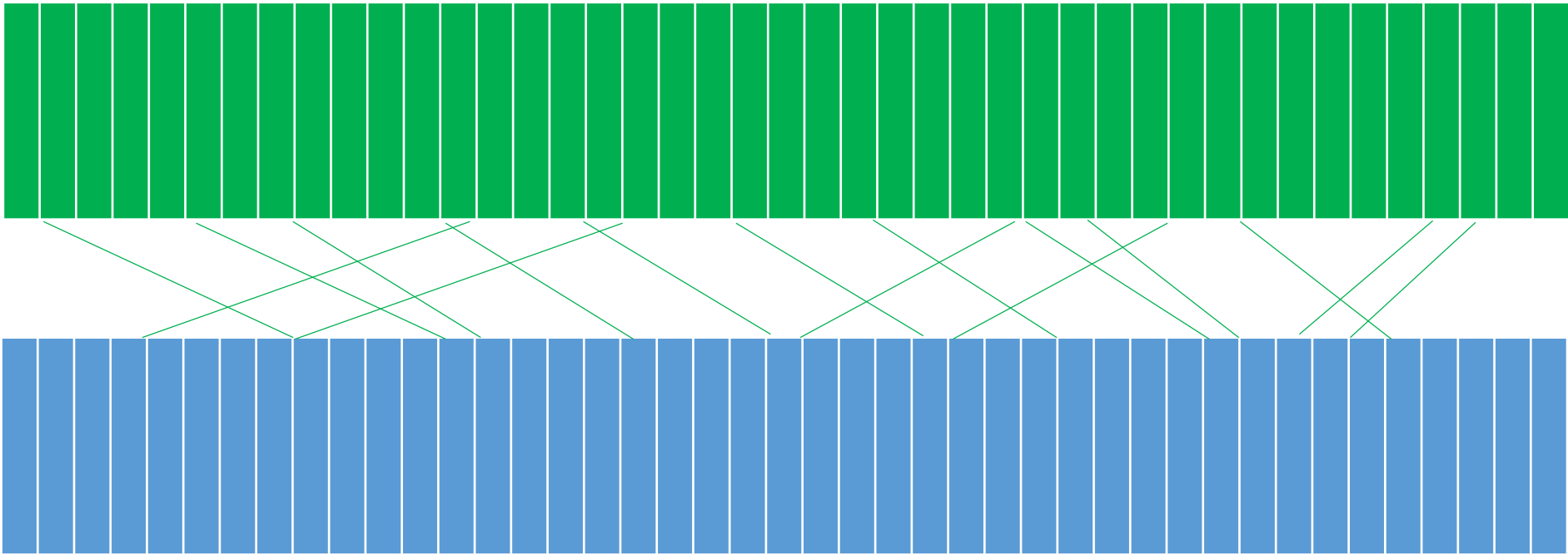
Trim algorithm I

CACCTAAATATTA



- Time 1: *CACCTAAATATTA* → *CACCTA*

Trim algorithm I

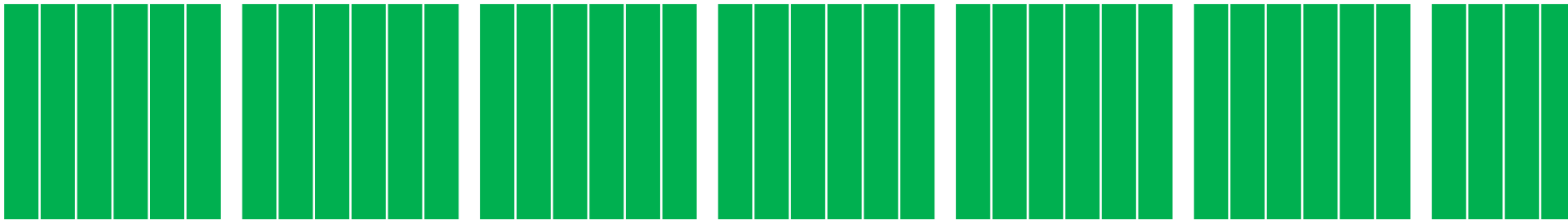
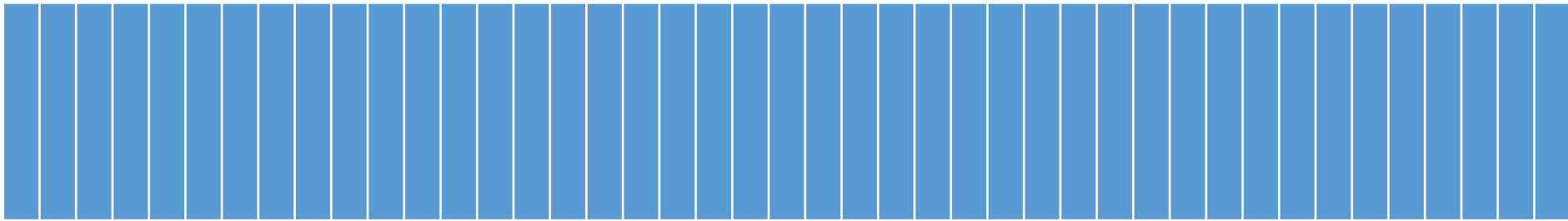


- The task is fundamentally linear in N : we have to touch every record no matter what

Can we do any better?

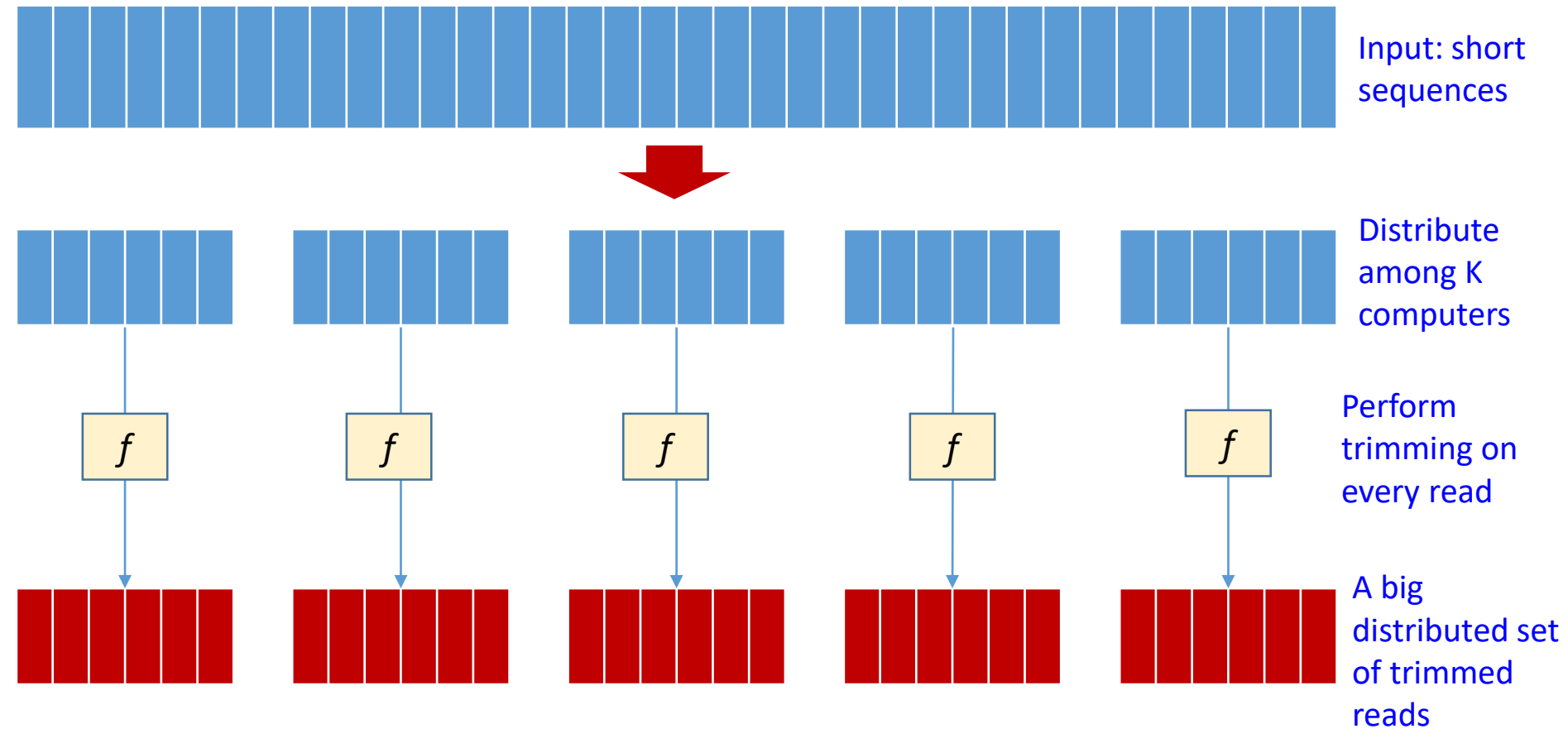
Will an index help?

Trim algorithm II

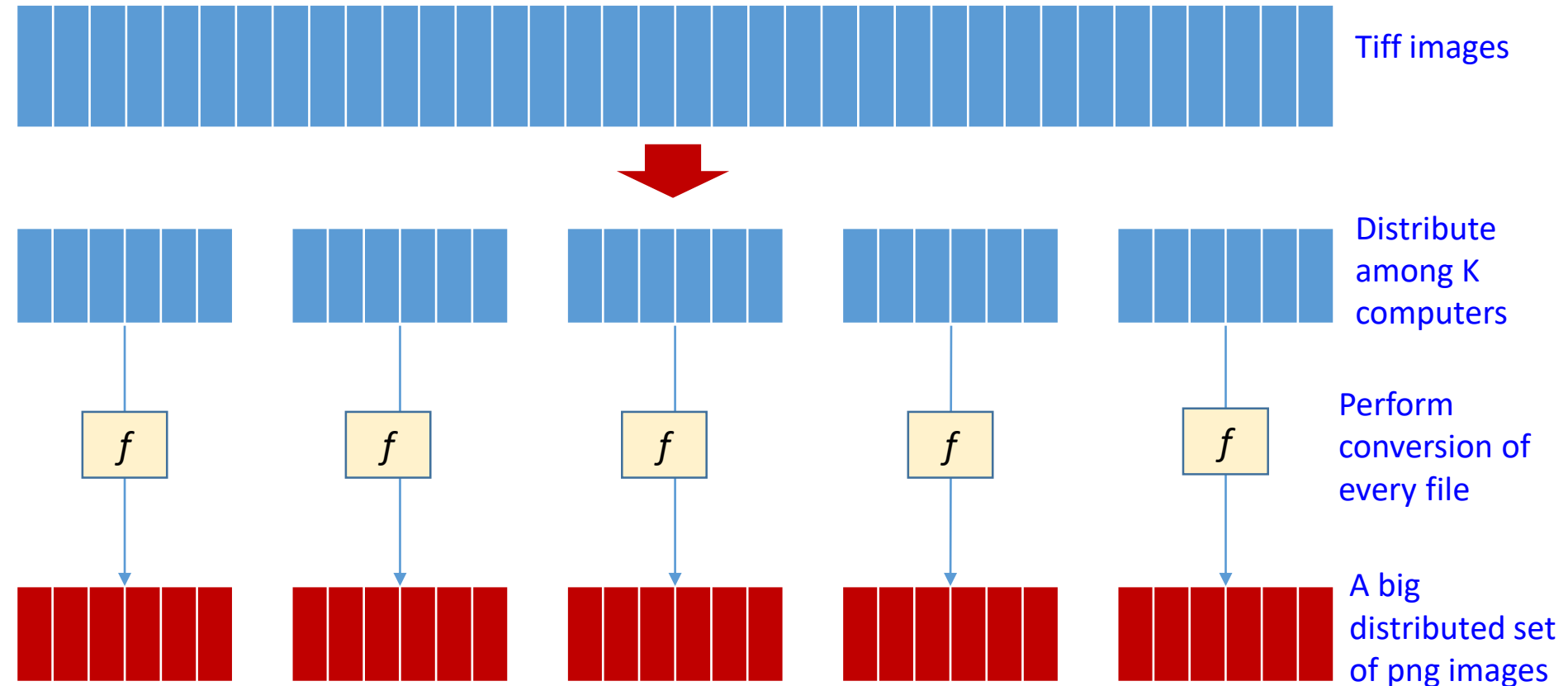


- We can break data into K pieces
- Assign each sub-task to a different machine
- Process each piece in parallel
- All work is finished in time N/K

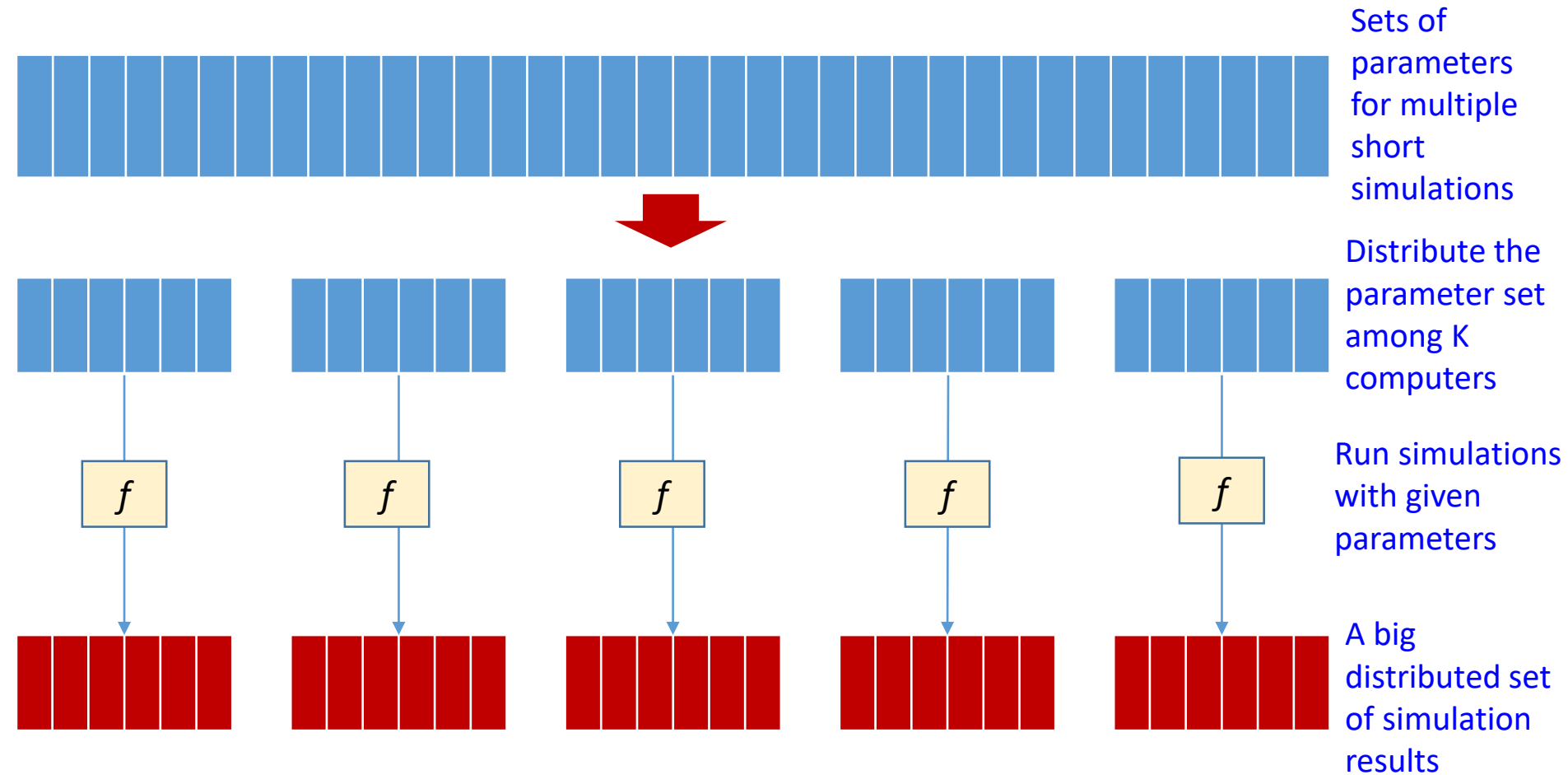
Schema of parallel “read trimming” task



Converting tiff images to png



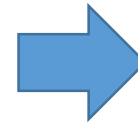
Simulations with multiple parameters



Compute word frequency of each word in a set of documents

The Declaration of Independence (abridged form)

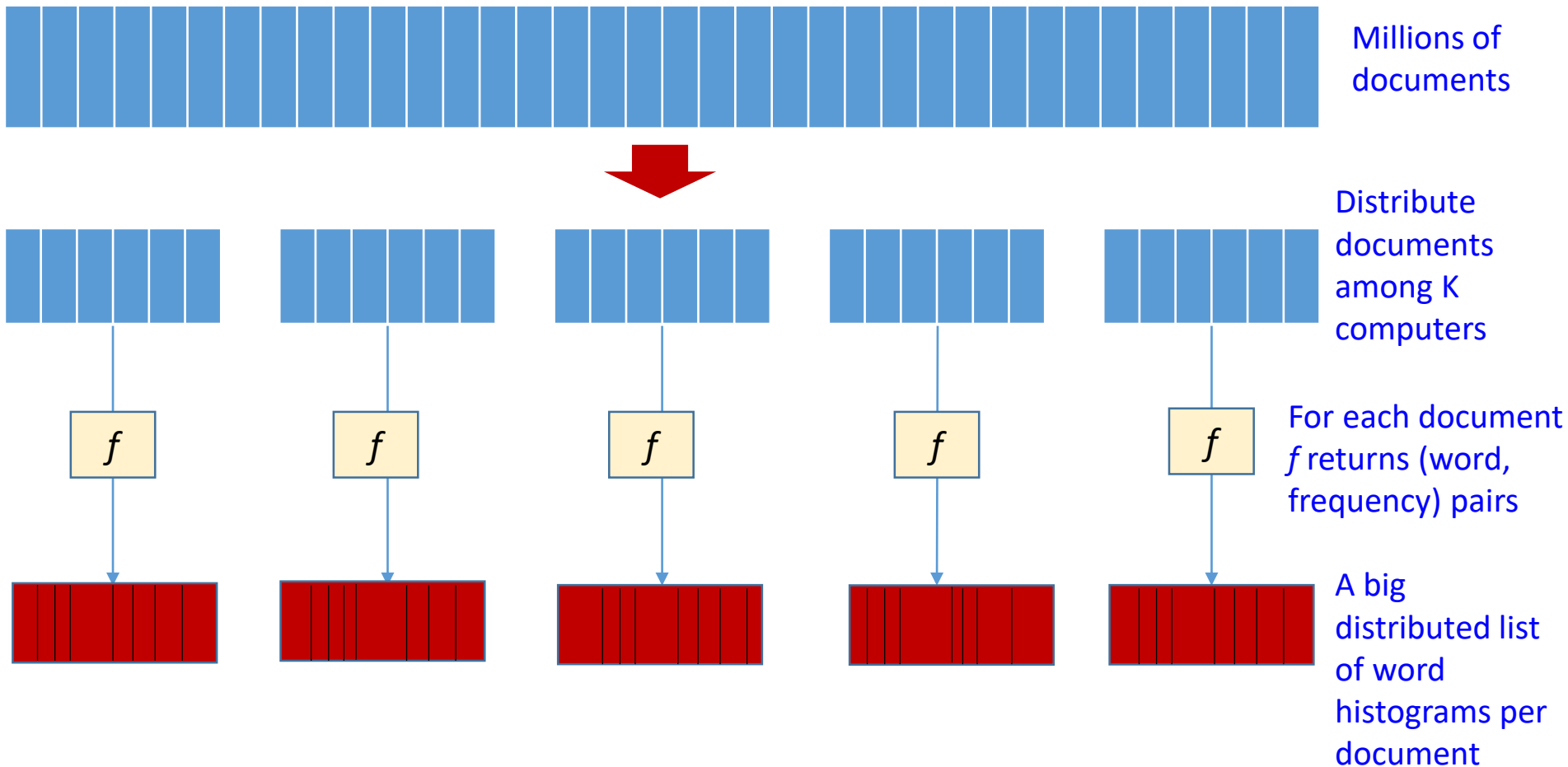
When, in the course of human events, it becomes necessary for one people to dissolve the political bonds which have connected them with another, and to assume among the powers of the earth, the separate and equal station to which the laws of nature and of nature's God entitle them, a decent respect to the opinions of mankind requires that they should declare the causes which impel them to the separation. We hold these truths to be self-evident, that all men are created equal, that they are endowed by their Creator with certain unalienable rights, that among these are life, liberty and the pursuit of happiness. That to secure these rights, governments are instituted among men, deriving their just powers from the consent of the governed. That whenever any form of government becomes destructive to these ends, it is the right of the people to alter or to abolish it, and to institute new government, laying its foundation on such principles and organizing its powers in such form, as to them shall seem most likely to effect their safety and happiness. Prudence, indeed, will dictate that governments long established should not be changed for light and transient causes; and accordingly all experience hath shown that mankind are more disposed to suffer, while evils are sufferable, than to right themselves by abolishing the forms to which they are accustomed. But when a long train of abuses and usurpations, pursuing invariably the same object evinces a design to reduce them under absolute despotism, it is their right, it is their duty, to throw off such government, and to provide new guards for their future security. — Such has been the patient sufferance of these colonies; and such is now the necessity which constrains them to alter their former systems of government. The history of the present King of Great Britain is a history of repeated injuries and usurpations, all having in direct object the establishment of an absolute tyranny over these states. To prove this, let facts be submitted to a candid world. He has refused his assent to laws, the most wholesome and necessary for the public good. He has forbidden his governors to pass laws of immediate and pressing importance, unless suspended in their operation till his assent should be obtained; and when so suspended, he has utterly neglected to attend to them. He has refused to pass other laws for the accommodation of large districts of people, unless those people would relinquish the right of representation in the legislature, a right inestimable to



(people, 2)
(government, 6)
(assume, 1)
(history, 2)
...

Single document processing example

Word frequencies

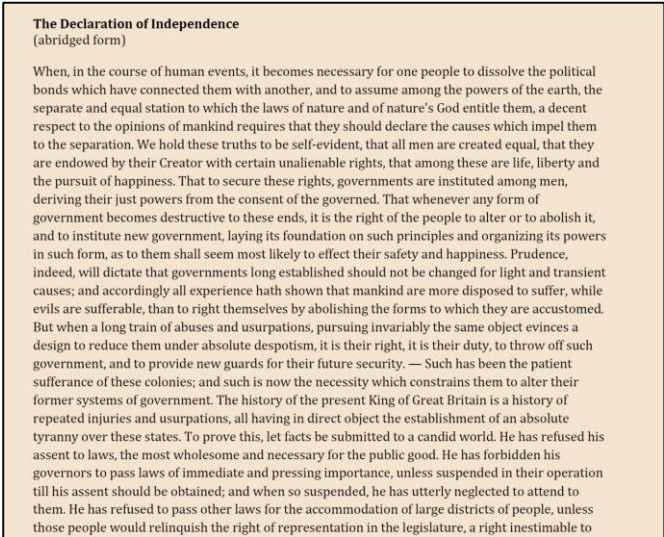
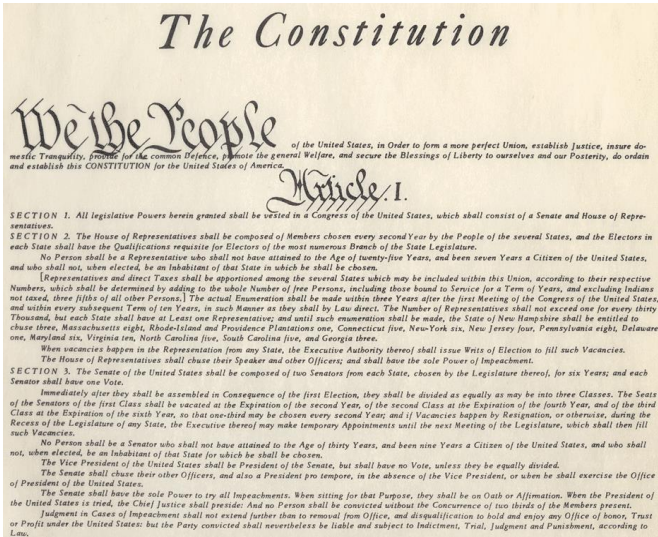


There is a pattern here ...

- A function that **maps** a read to a trimmed read
- A function that **maps** tiff image to png image
- A function that **maps** a set of parameters to a simulation results
- A function that **maps** a document to a histogram of word frequencies

The idea is to **abstract** the farming of parallel programs **into a general framework**, where the programmer only needs to provide the mapping function itself

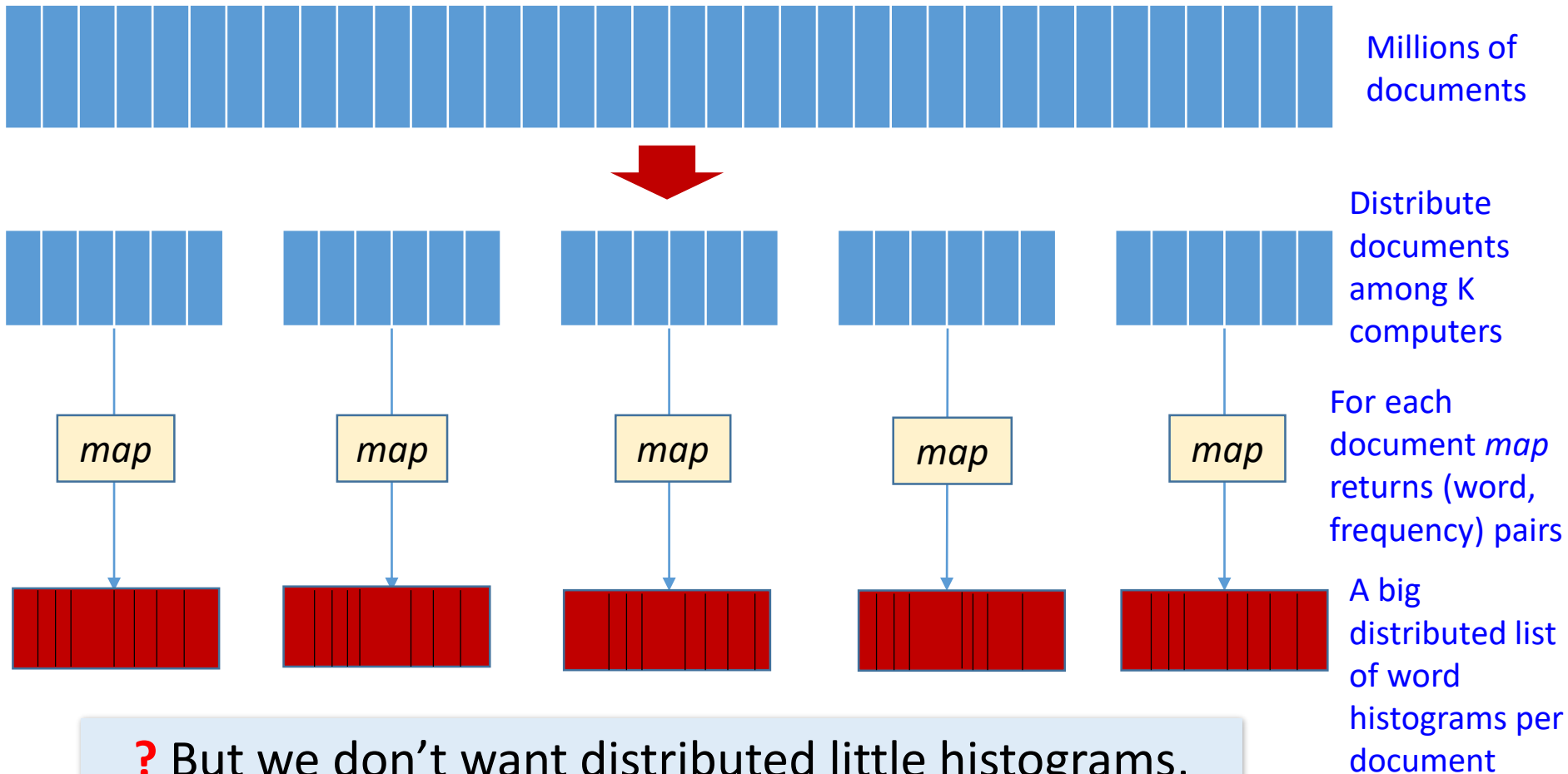
Different task: Compute word frequencies for all documents



- (people, 78)
- (government, 123)
- (assume, 23)
- (history, 38)

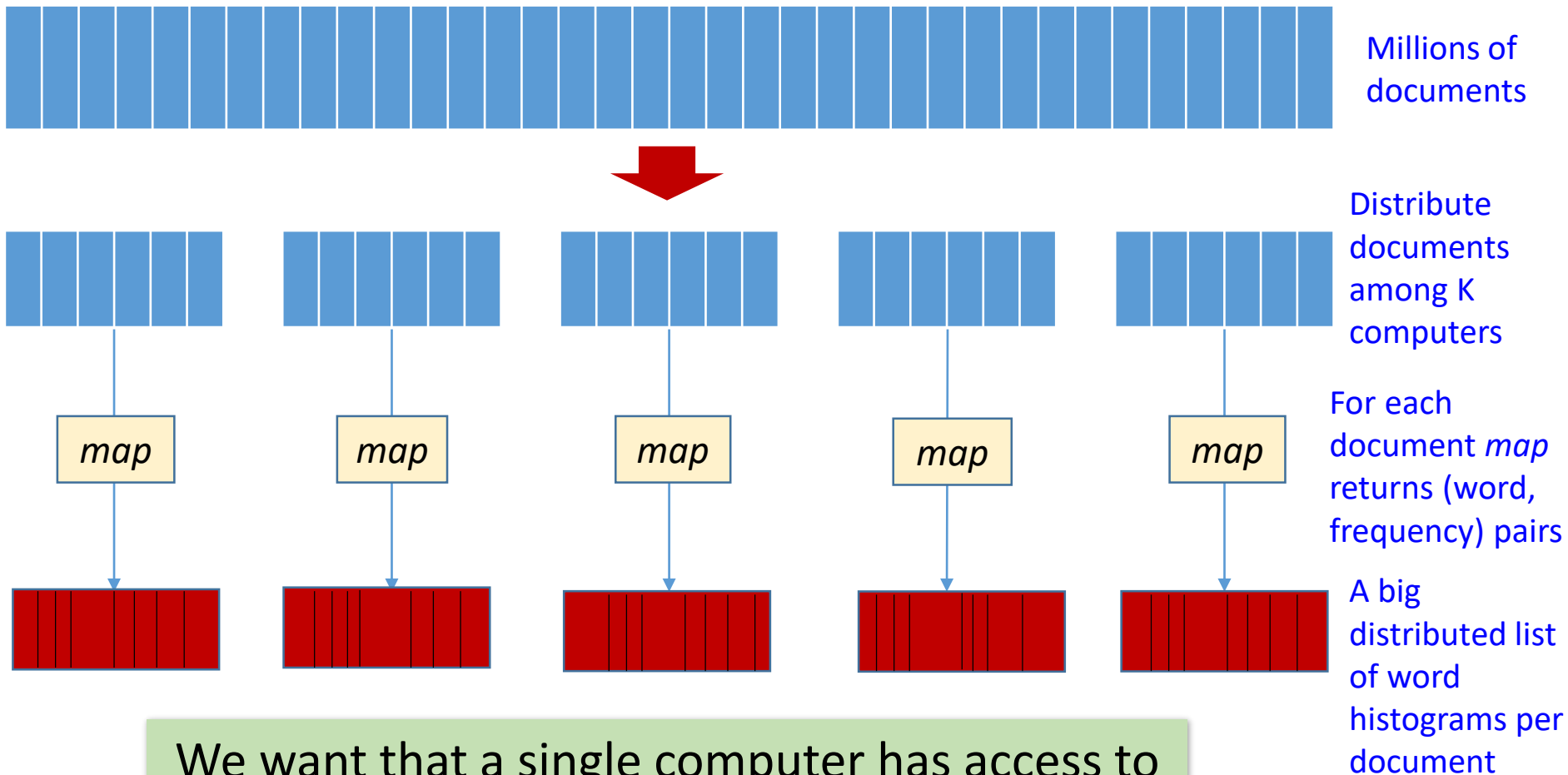
...

Word frequencies among all documents



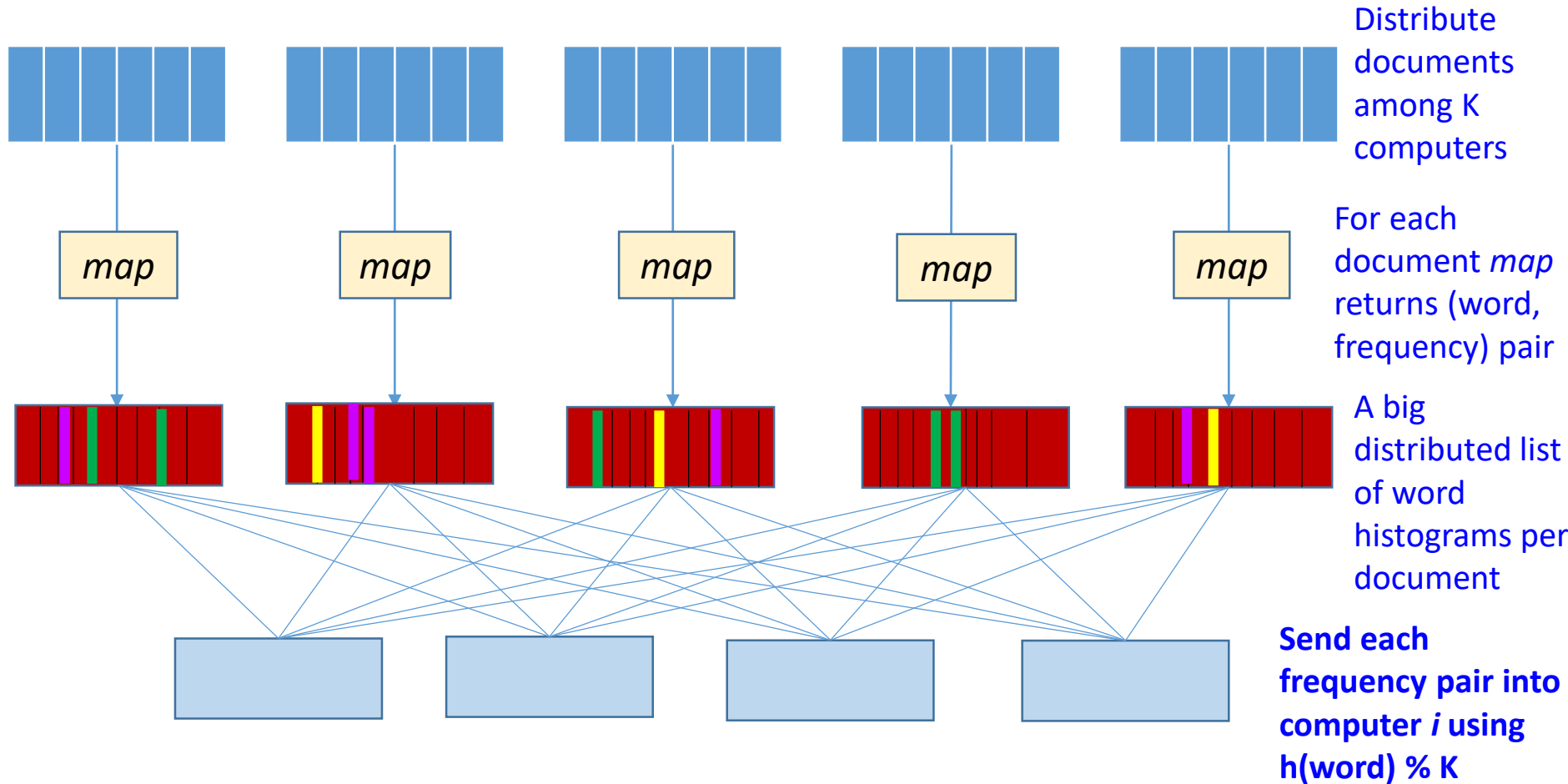
? But we don't want distributed little histograms, we want one big histogram

Word frequencies among all documents

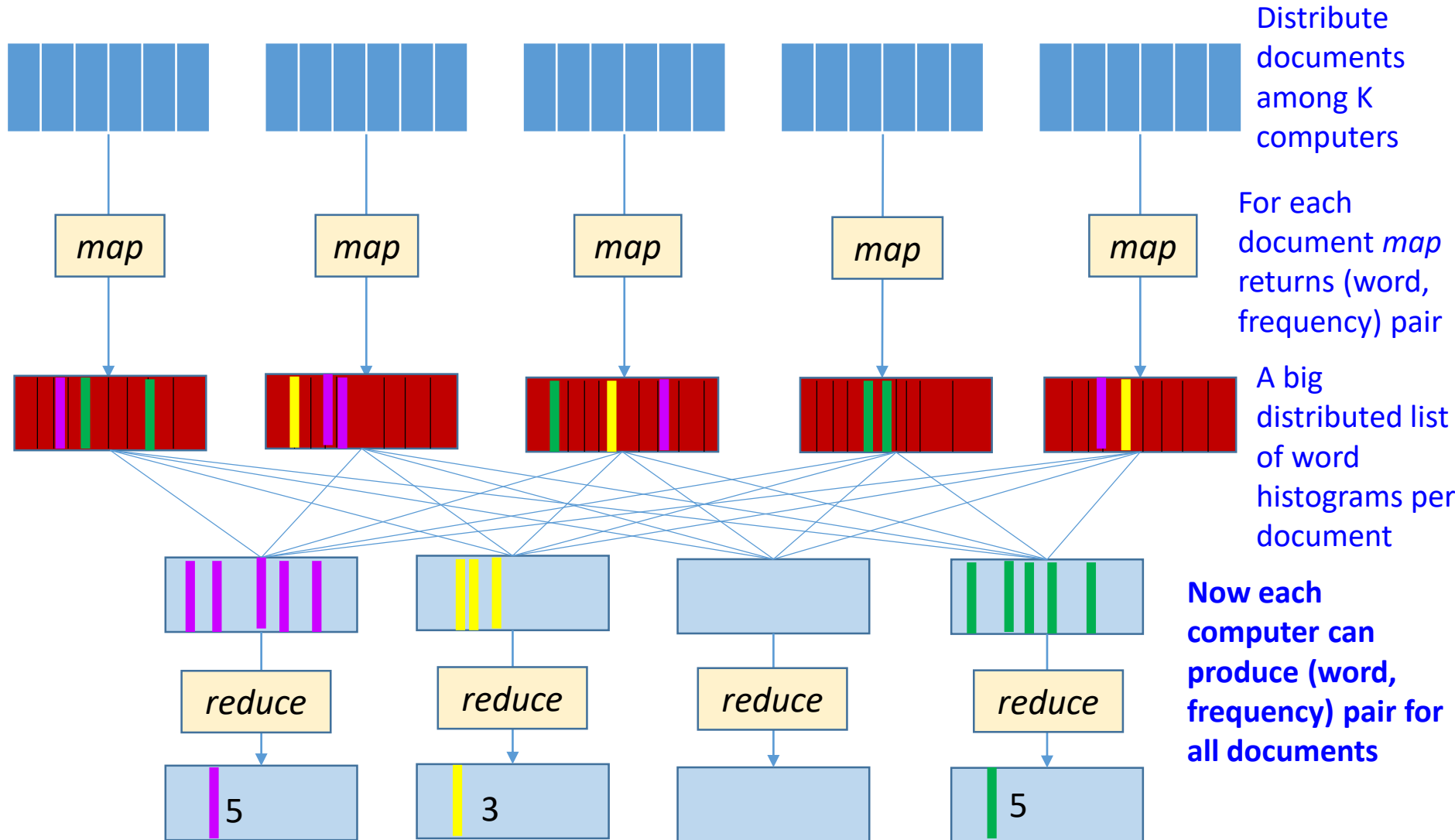


We want that a single computer has access to **all occurrences of a given word**

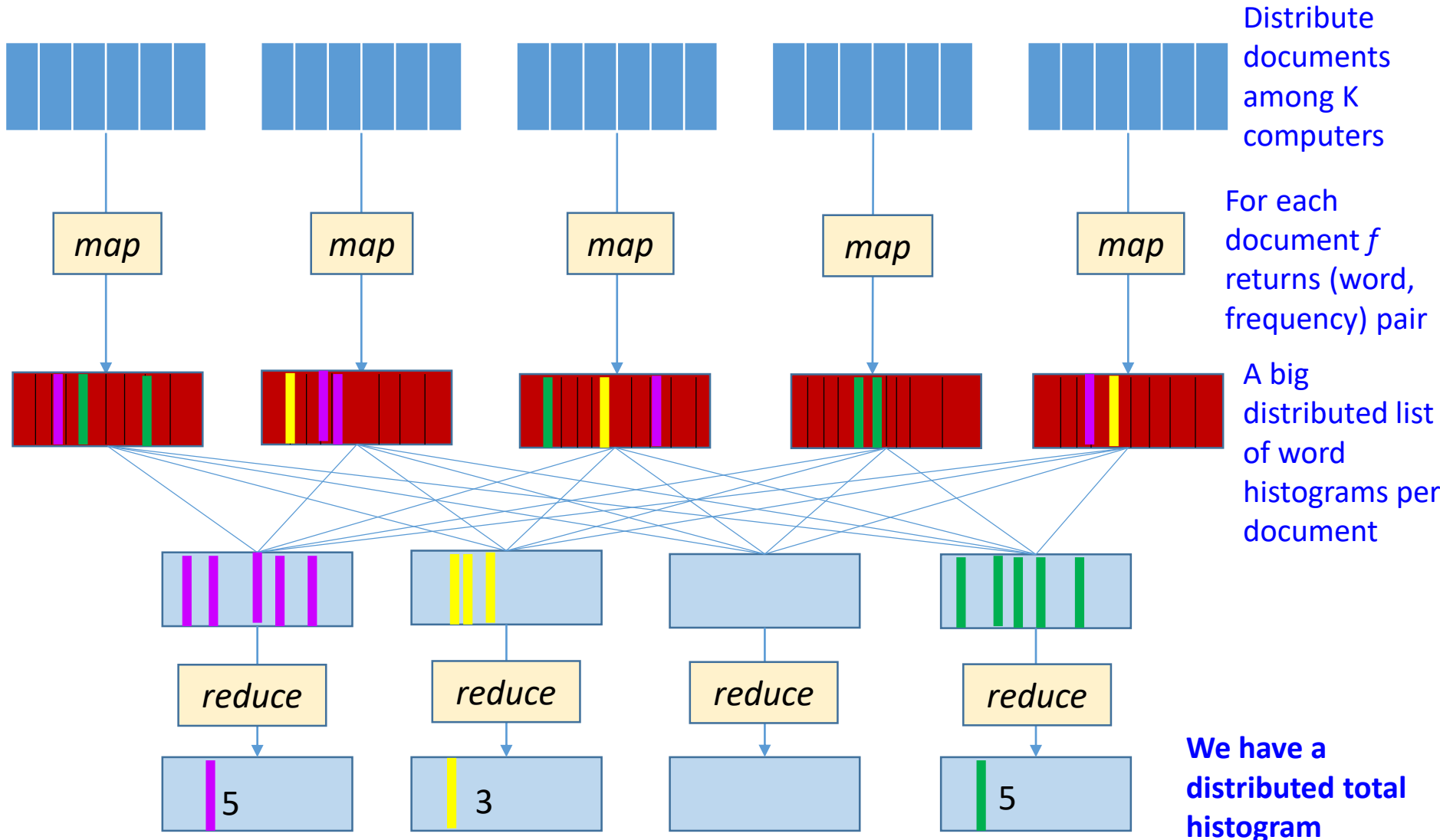
Word frequencies among all documents



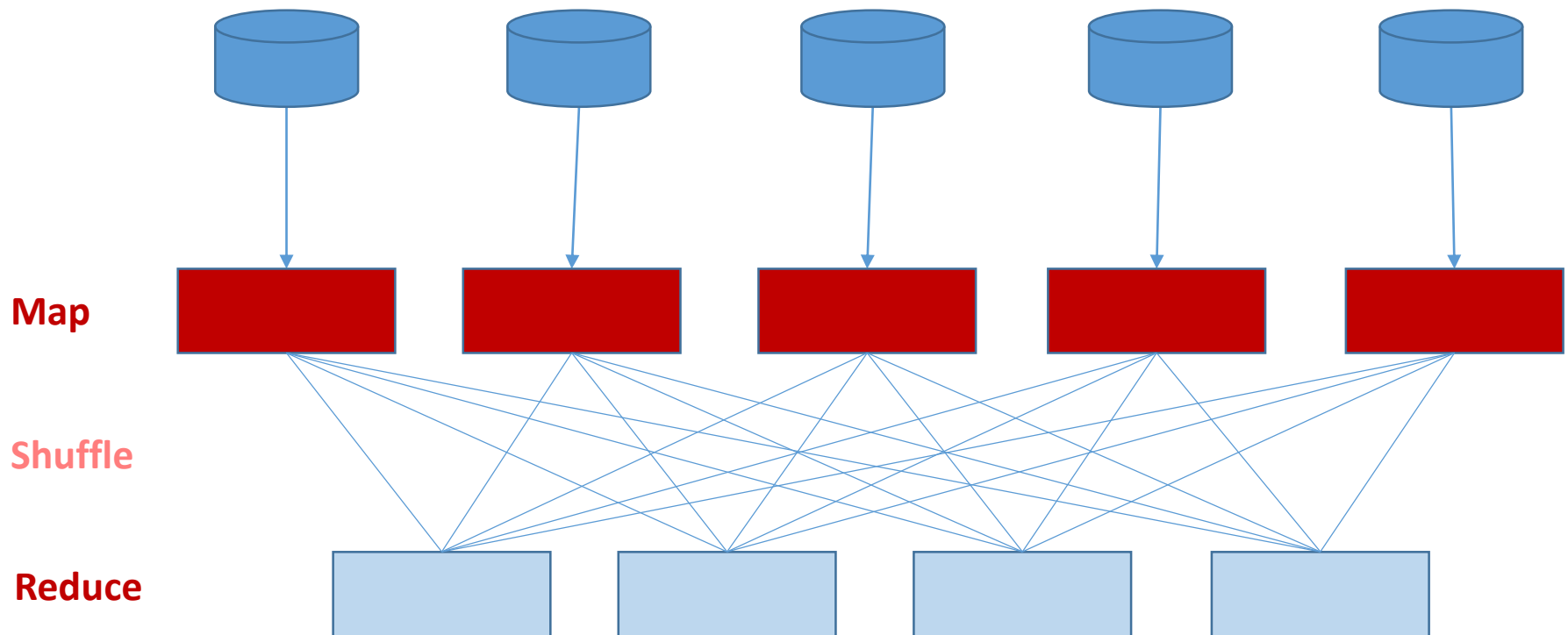
Word frequencies among all documents



Word frequencies among all documents



General idea: partitioning by hashing

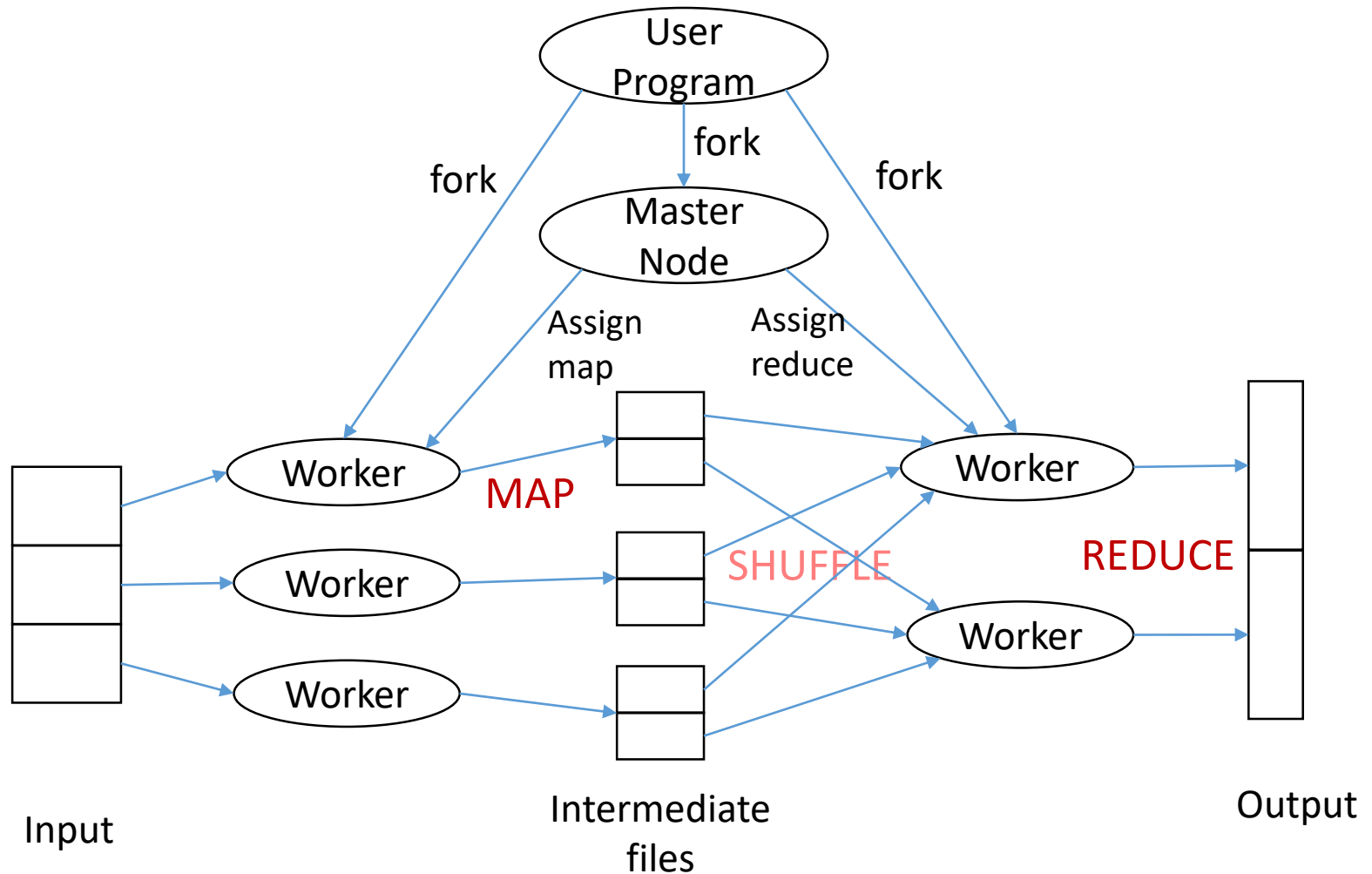


Only *map* and *reduce* differ from one application to another
Everything else is generic and is implemented in a map-
reduce framework

Map-reduce

- The user writes two functions: **map** and **reduce**
- A master controller divides the **input data into chunks**, and assigns different processors to execute the **map** function on each **chunk**
- Other processors, perhaps the same ones, are then assigned to perform the **reduce** function on **chunks of the output from the map** function

Map-reduce framework



Map-reduce principles

- **Storage Infrastructure – Distributed File System**

- Input is stored in chunks of ~64 MB on *compute nodes*, only *master node* knows where
- Google: GFS. Hadoop: HDFS

- **Programming model – Map-Reduce**

- *Sequentially* read a lot of data
- Extract something you care about
- *Group by key*: send to reducer

- **Data model**

- Input: a *bag of (input key, value)* pairs
- Output: a *bag of (output key, value)* pairs

Processing (really) big inputs

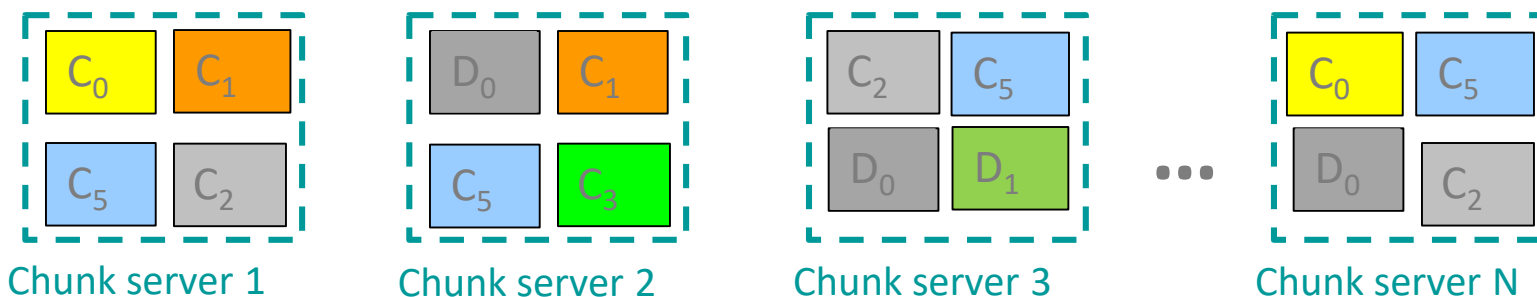
- Scalability of algorithms
- Inherently parallelizable tasks
- Distributed file system
- Map-reduce computation
- Practice

Distributed File System

- **Chunk servers**
 - 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
- **Master node** (*Name Node* in Hadoop's HDFS)
 - Stores metadata about where files are stored
 - Might be replicated
- **Client library for file access**
 - Talks to master to find chunk servers
 - Connects directly to chunk servers to access data
- **Works best for static files**
 - Files are rarely updated
 - Can only grow in size by appending new data to the end

Reliable distributed file system

- Data kept in “chunks” spread across machines
- Each chunk **replicated** on different machines
 - Seamless recovery from disk or machine failure



Bring computation directly to the data!

Chunk servers also serve as compute servers

Processing (really) big inputs

- Scalability of algorithms
- Inherently parallelizable tasks
- Distributed file system
- Map-reduce computation
- Practice

Map

- The input is in chunks on different nodes
- *Map* function is forked to the same chunk server where the data is
- The output of *map* function is partitioned by hashing the output key: $h(\text{key}) \% R$, where R is the number of reducers
- **The partitioned output** is written to **the same local disk** on a computing node where the input is

Shuffle

- The system then performs shuffling of the intermediate (key, value) pairs and **sends the data** to a corresponding **reduce node**, according to **hash(key)**. All data with the same key ends up on the same machine
- Creates Master file to store info about the locations of chunks for final output, which will also be distributed across chunk servers
- Already at the reducer: produces aggregated lists of values for each key

Reduce

- Each node to which a **reduce** tasks has been assigned takes one key at a time, and performs required operations on the corresponding list of values
- The final output is written to a local disk of a reducer, and the Master node is notified about where chunks of data reside
- **The output of a map-reduce program is a distributed file**

Example: what does it do?

map (input_key, input_value)

for each word *w* **in** input_value

emit_intermediate (w, 1)

reduce (*intermediate_key*, Iterator *intermediate_values*)

result: =0

for each *v* **in** *intermediate_values*

result += *v*

emit (*intermediate_key* , *result*)

Word count in Python

```
# To run:  
mr = MapReduce.MapReduce()  
  
inputdata = open(sys.argv[1])  
mr.execute(inputdata, mapper,  
           reducer)
```

```
def mapper (record):  
    # key: document identifier  
    # value: document contents  
    key = record[0]  
    value = record[1]  
    words = value.split()  
    for w in words:  
        mr.emit_intermediate(w, 1)  
  
def reducer (key, list_of_values):  
    # key: word  
    # value: list of occurrence counts  
    total = 0  
    for v in list_of_values:  
        total += v  
    mr.emit((key, total))
```

Example: word count

map (input_key, input_value)

for each word *w* **in** input_value

emit_intermediate (w, 1)

reduce (*intermediate_key*, Iterator *intermediate_values*)

result: =0

for each *v* **in** *intermediate_values*

result += *v*

emit (*intermediate_key* , *result*)

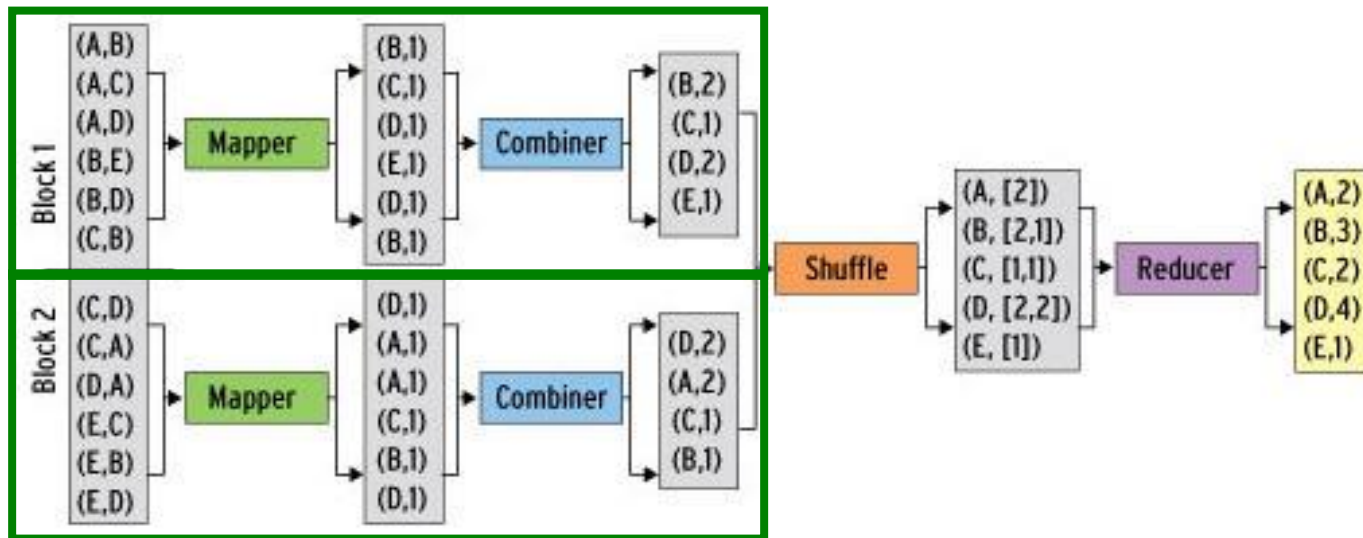
Without changing the **reduce** function,
improve performance of this algorithm

Refinement: Combiners

- Often a *map* task will produce many pairs of the form $(k, v_1), (k, v_2), \dots$ for the same key k
 - E.g., popular words in the word count example
- **Can save network time by pre-aggregating values in the mapper:**
 - Combine $(k, \text{list}(v_1)) \rightarrow (k, v_2)$
- Works only if *reduce* function is commutative and associative

Refinement: Combiners

- **Back to our word counting example:**
 - Combiner combines the values of all keys of a single mapper (single machine):



Much less data needs to be copied and shuffled!

Coordination: Master

- **Master node takes care of coordination:**
 - **Task status:** (idle, in-progress, completed)
 - **Idle tasks** get scheduled as workers become available
 - When a map task completes, it sends to the master the location and sizes of its R intermediate files, one for each reducer
 - Master pushes this info to reducers
- Master pings workers periodically to detect failures

How many Map and Reduce jobs?

- M map tasks, R reduce tasks
- **Rule of thumb:**
 - Make M much larger than the number of nodes in the cluster
 - One DFS chunk per map is common
 - Improves dynamic load balancing and speeds up recovery from worker failures
- **Usually R is smaller than M**
 - Because output is spread across R files

Map-reduce solves the following issues:

1: Copying data over a network takes time

- **Idea:**

- Bring computation close to the data. The file chunks are distributed across nodes and map programs are forked to the same machine – program comes to data

2: Machines fail

- One server may stay up to 3 years (1,000 days)
- If you have 1,000 servers, expect to loose 1/day
- Google had ~1M machines in 2011: 1,000 machines fail every day!

- **Idea:**

- Store files multiple times for reliability. Each file chunk is replicated in at least 3 nodes

3: Parallel programming is difficult

- Programmer only needs to provide *map* and *reduce* functions which fit the problem. Everything else – distribution, hashing, load balancing – is handled by the system