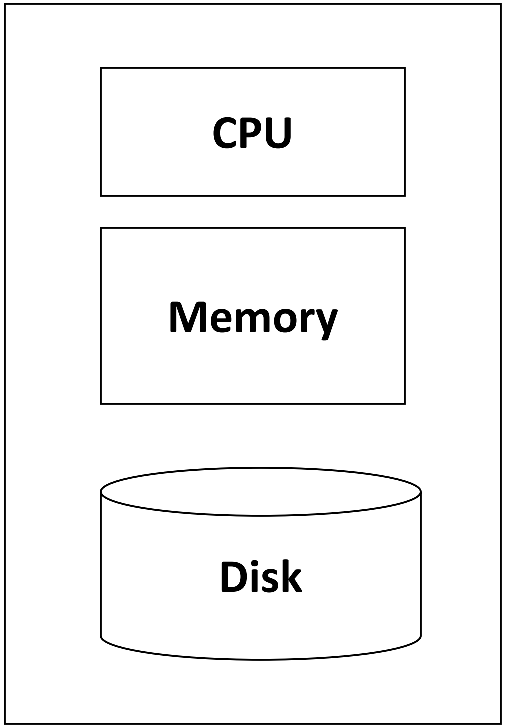
CMPT 321
FALL 2017

Map-Reduce

Lecture 08.02

By Marina Barsky

Single Node Architecture



“Classical” Data Processing

RDBMSs are very efficient

All the disk-based algorithms and data structures have great performance given the table is < certain size (typically 100GB)

What if the 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 → tractable → scalable
- $O(m^N)$ - Exponential → not scalable →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

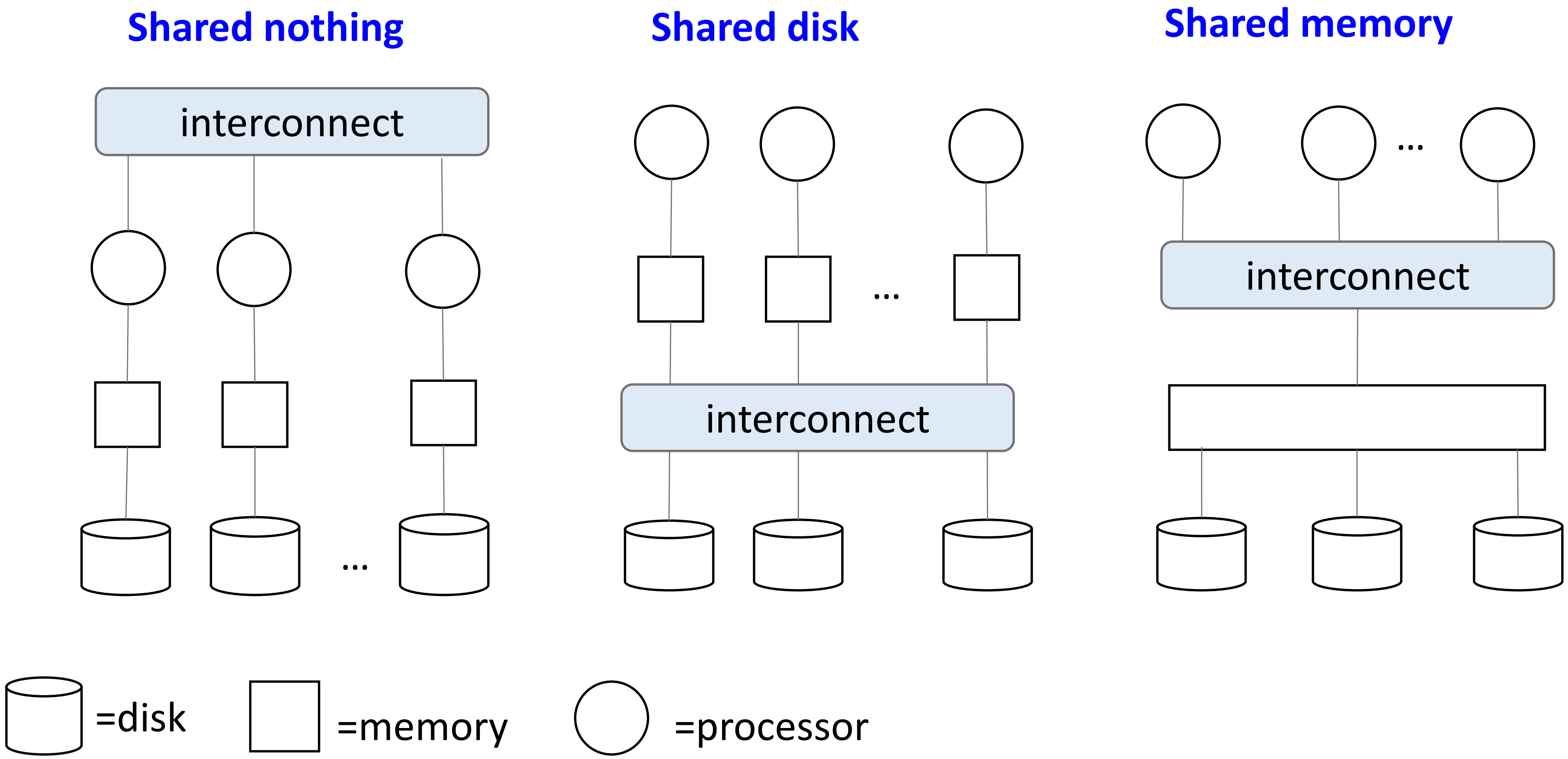
Motivation: Google Example

- 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

interconnect

Shared disk

interconnect

Shared memory

interconnect

=disk

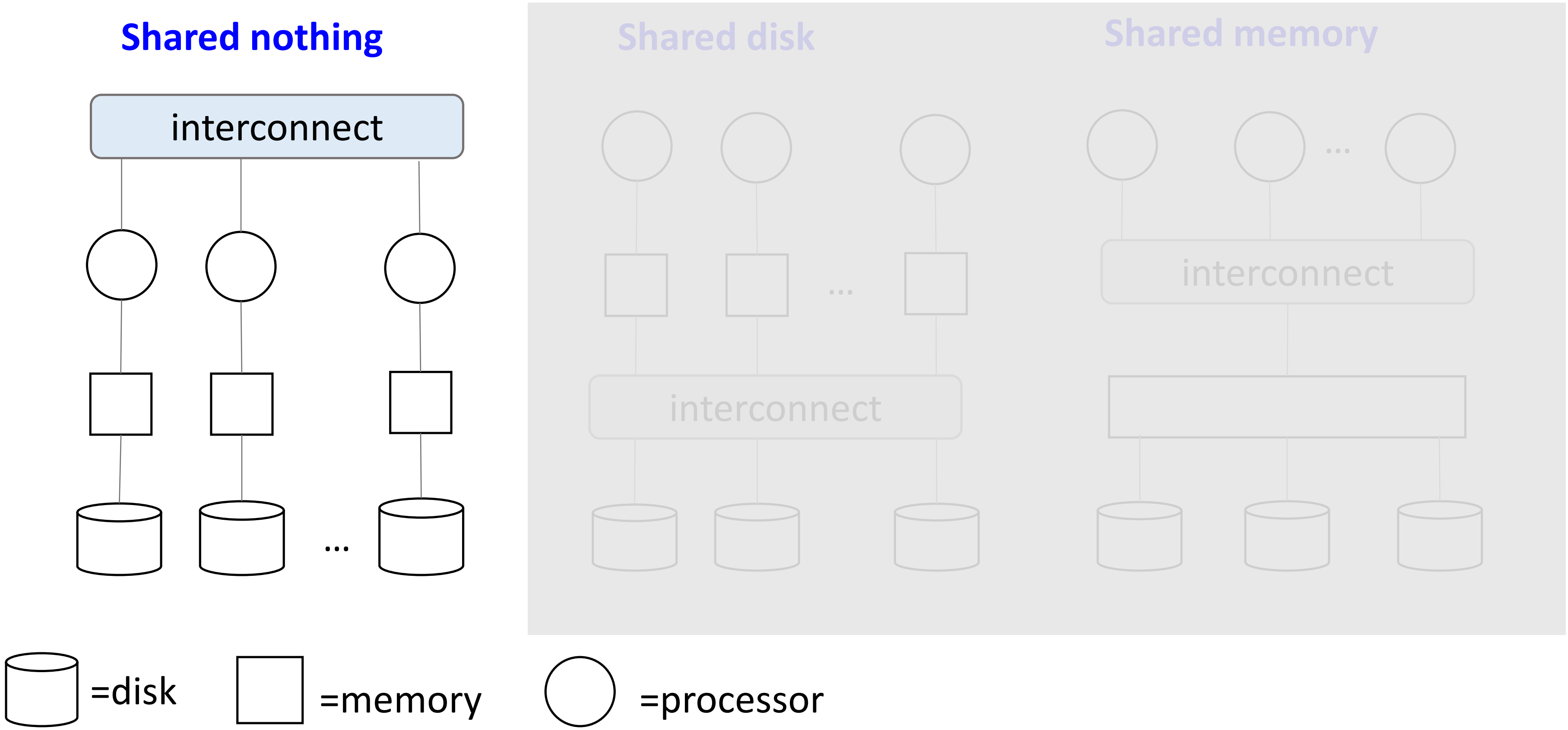
=memory

=processor

D. J. DeWitt, J. Gray, "Parallel Database Systems: the Future of High Performance Database Systems", ACM Communications, vol. 35(6), 85-98, June 1992.

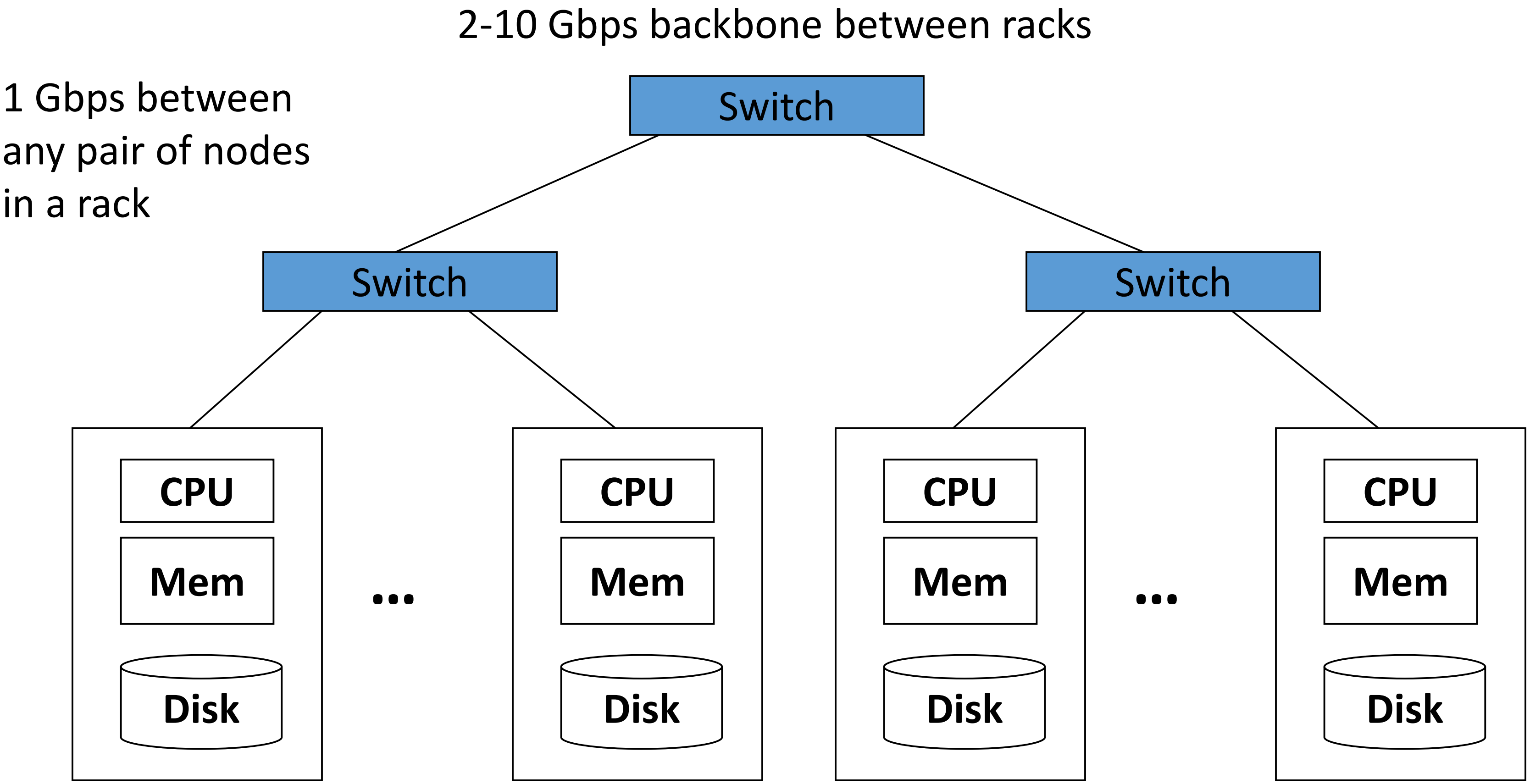
Scalability of parallel architectures

Logical multi-processor database designs



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

Switch

Switch

Switch

CPU

Mem

Disk

…

CPU

Mem

Disk

CPU

Mem

Disk

…

CPU

Mem

Disk

Each rack contains 16-64 nodes

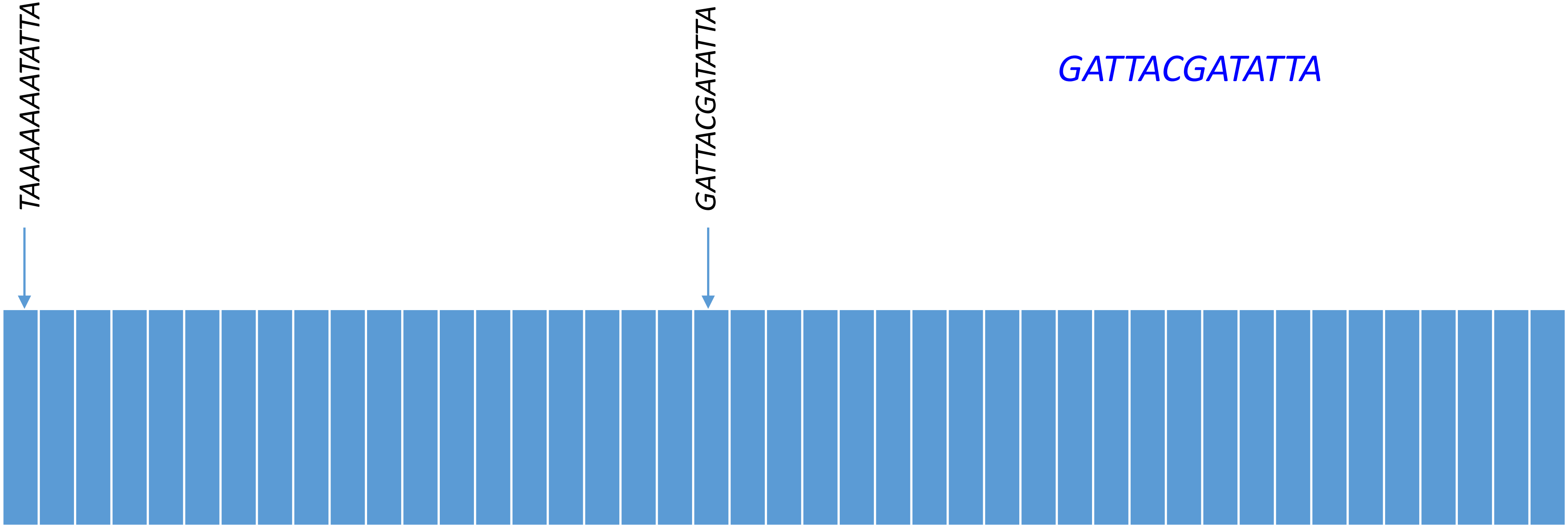
In 2011 it was estimated that Google had 1M machines, http://bit.ly/Shh0RO

Example 1: find matching DNA sequences

- Given a set of short sequences:

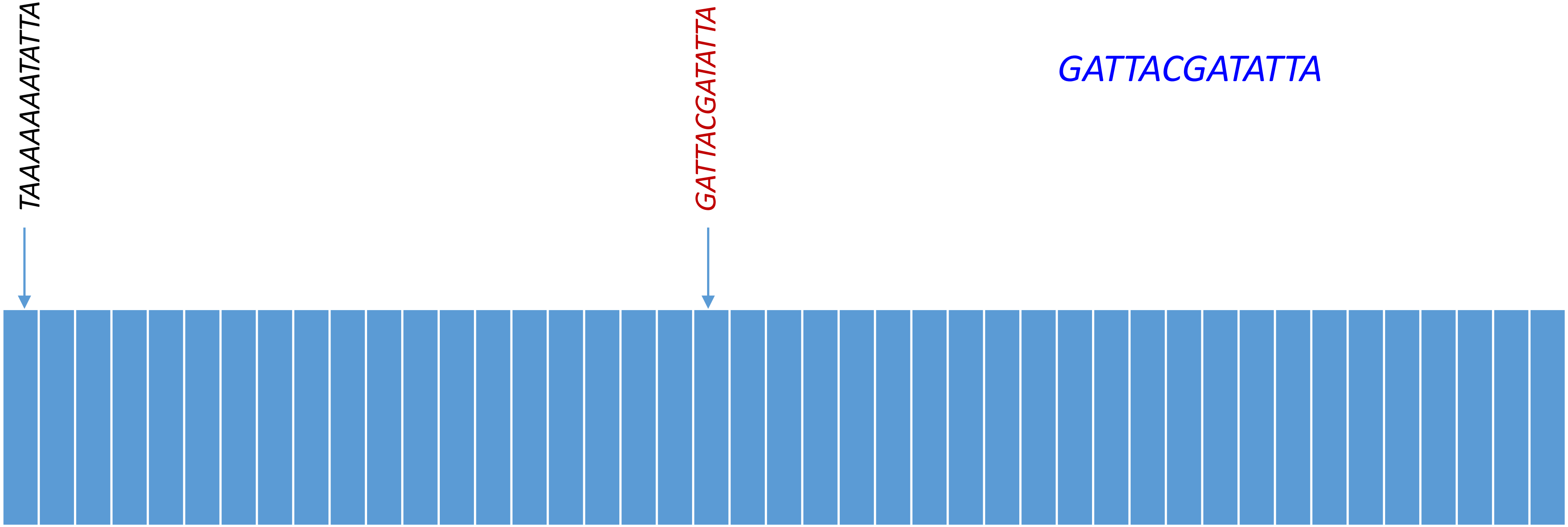
- Find all sequences equal to *GATTACGATATTA*

Search algorithm I



GATTACGATATTA

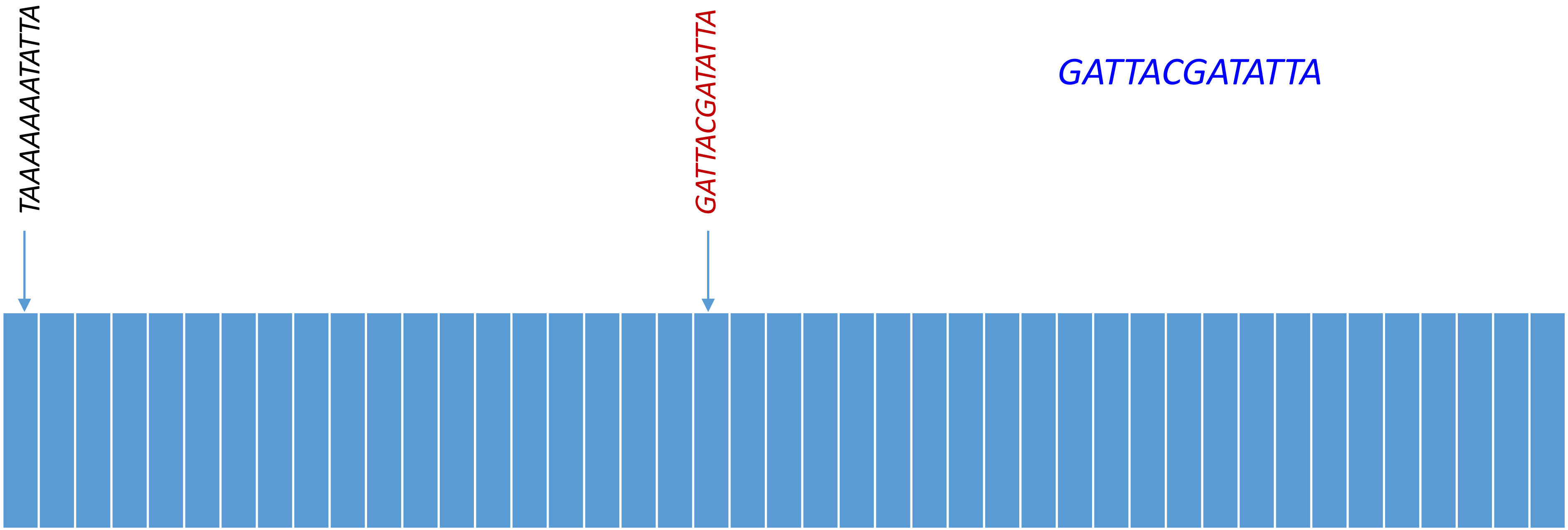
Search algorithm I



GATTACGATATTA

Step 20: found

Search algorithm I



TAAAAAAATATTA

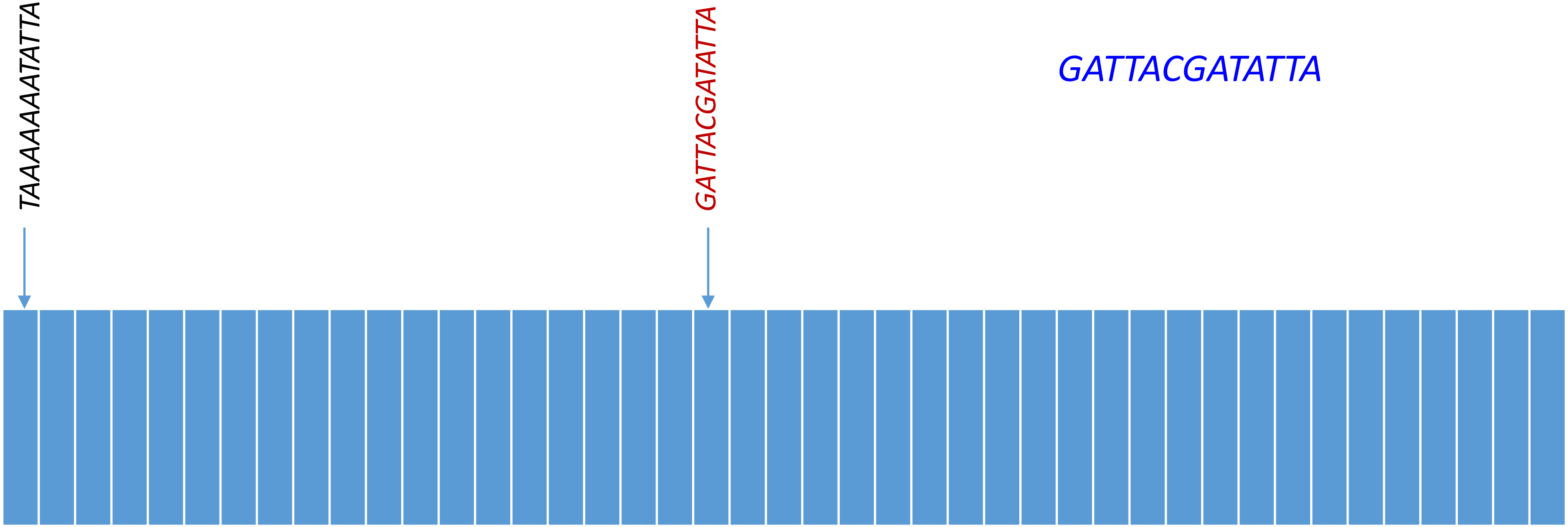
GATTACGATATTA

GATTACGATATTA

N = 40 records → 40 comparisons

O(N) algorithm

Search algorithm I



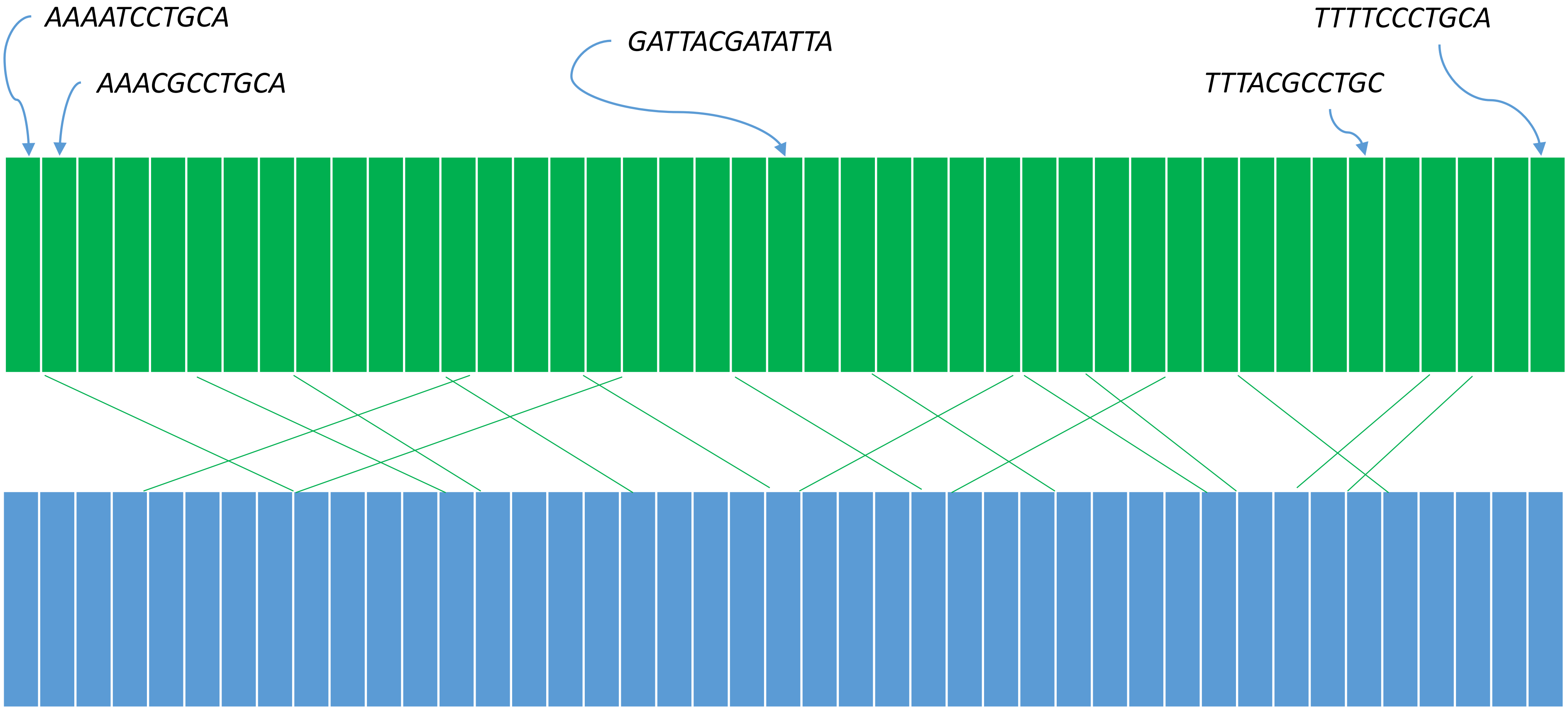
N = 40 records → 40 comparisons

O(N) algorithm

Can we do any better?

Search algorithm II

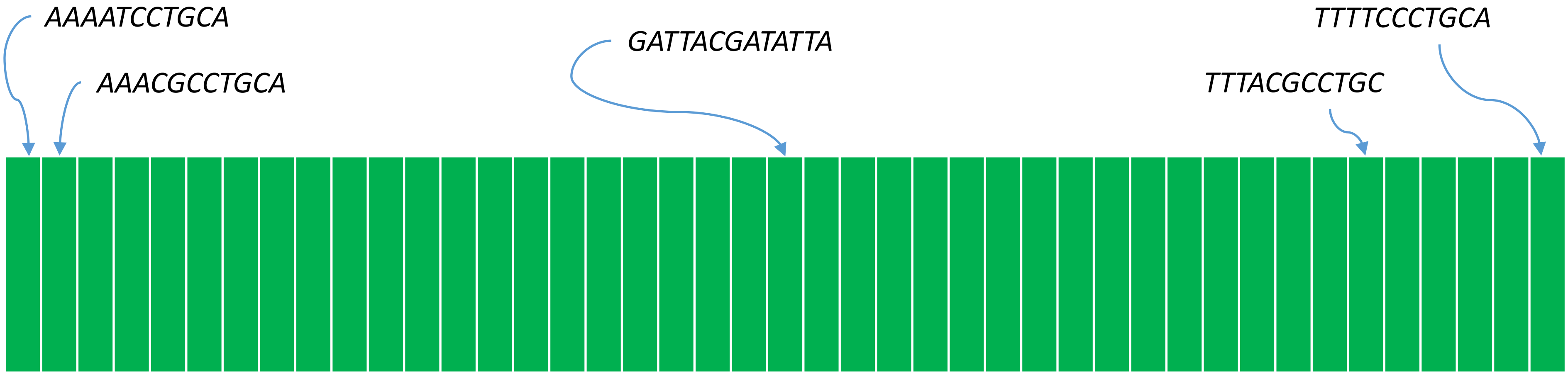
GATTACGATATTA



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

Search algorithm II

GATTACGATATTA



AAAATCCTGCA

AAACGCCTGCA

GATTACGATATTA

TTTTCCCTGCA

TTTACGCCTGC

Binary search: log N time

```
CREATE INDEX seq_idx ON sequences (seq)

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

Far better scalability !

Example 2: read trimming

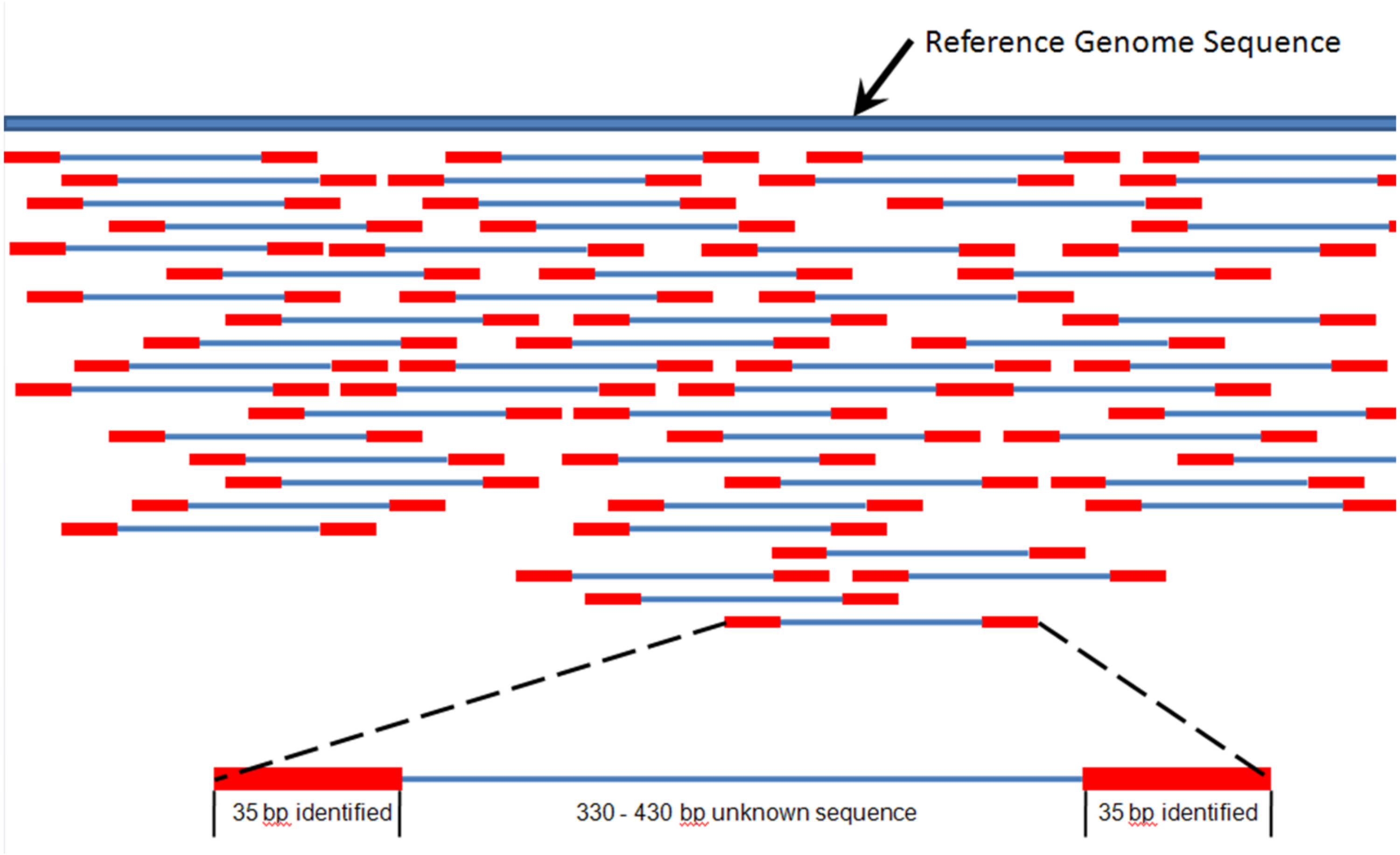
- Given a set of DNA reads – sequences of 100 characters long:

- Trim the final *t* (bp) characters of each sequence*
- Generate a new dataset of trimmed sequences

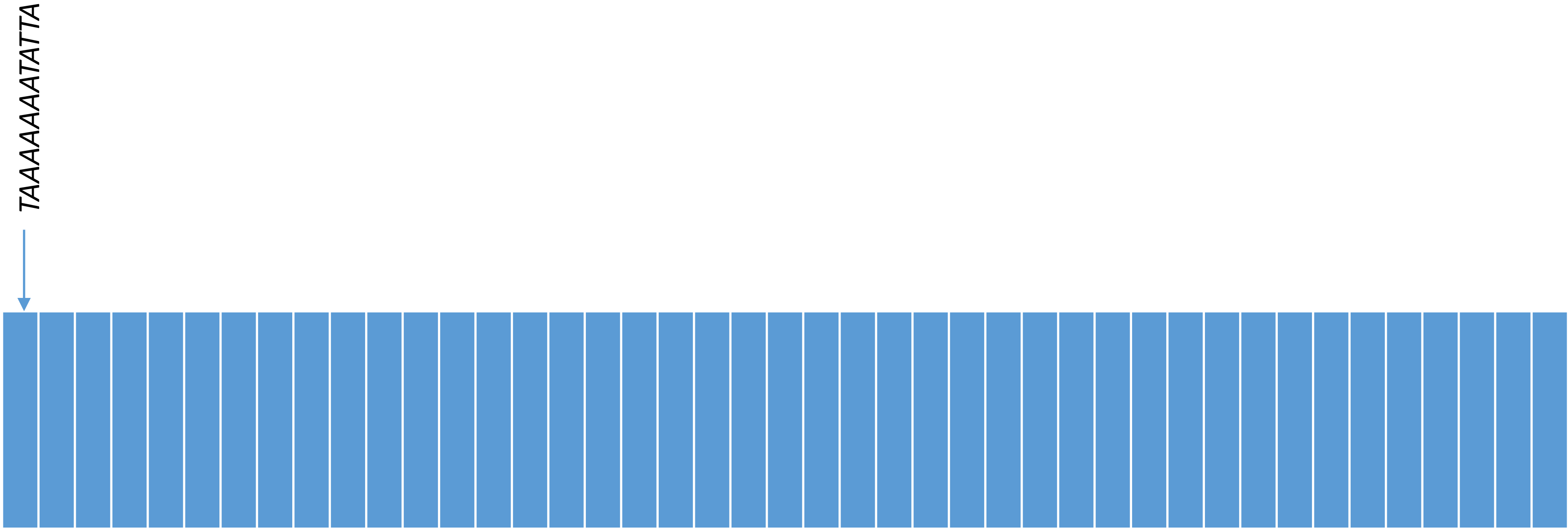
---

*The accuracy of sequencer drops abruptly after a certain length

Short raw DNA reads

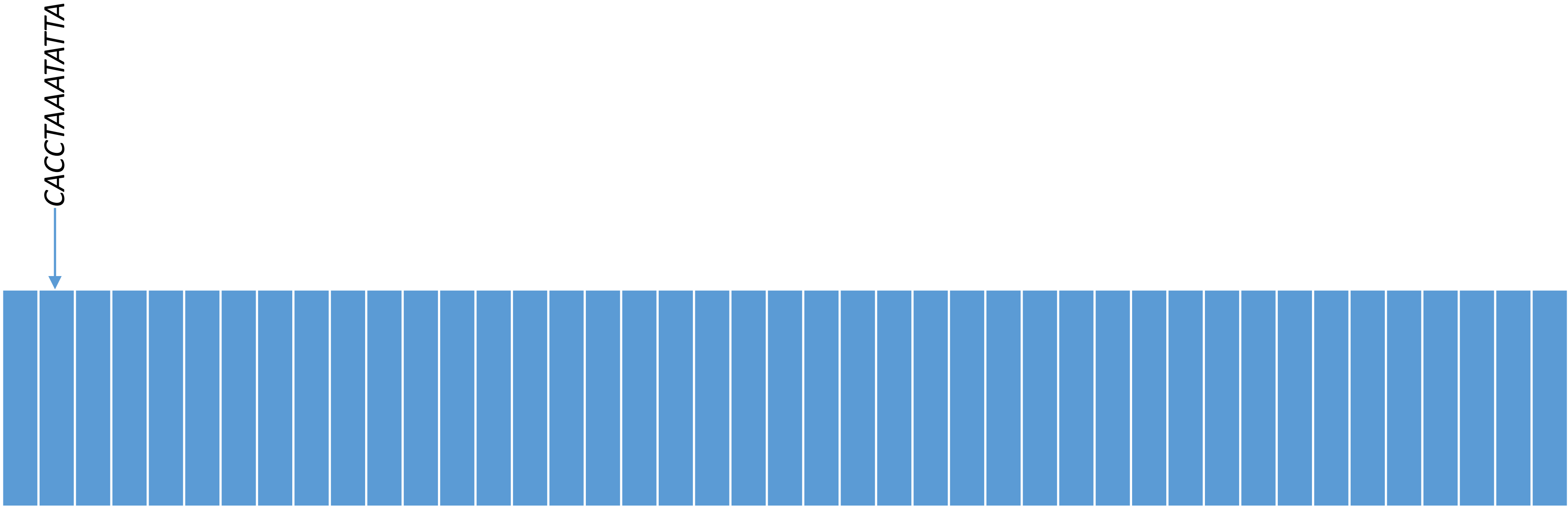


Trim algorithm I



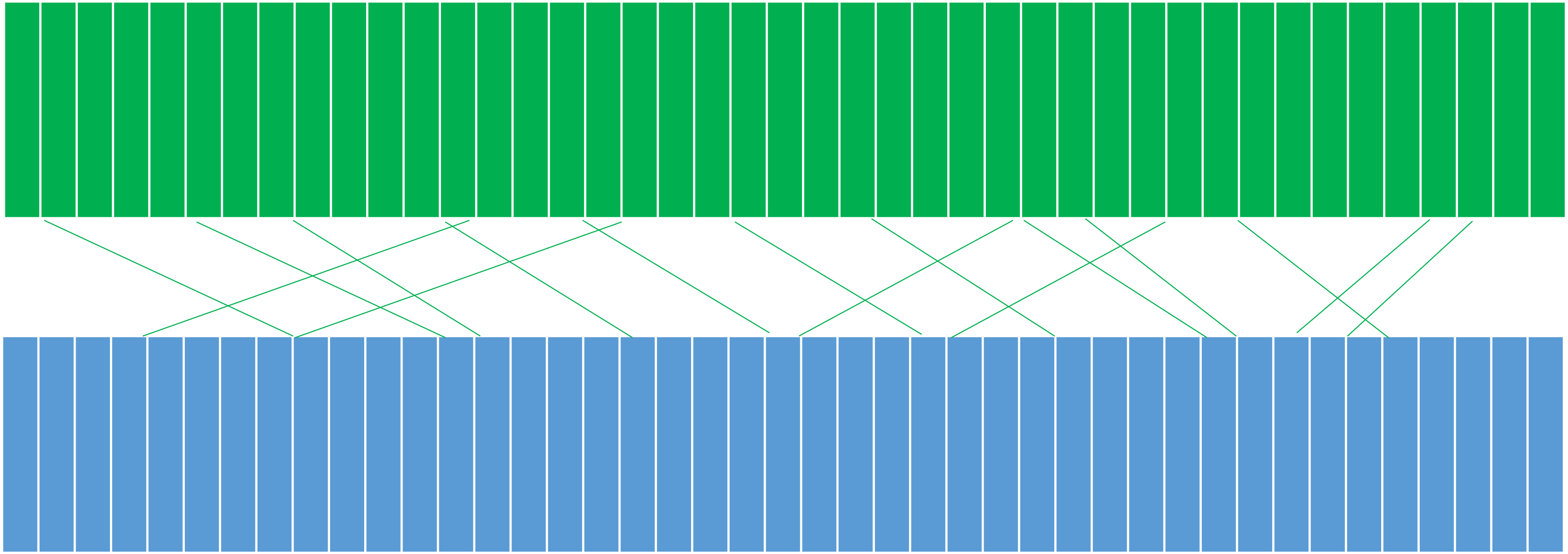
- Time 0: *TAAAAAAATATTA* → *TAAAAA*

Trim algorithm I



- Time 1: *CACCTAAATATTA* → *CACCTA*

Trim algorithm I

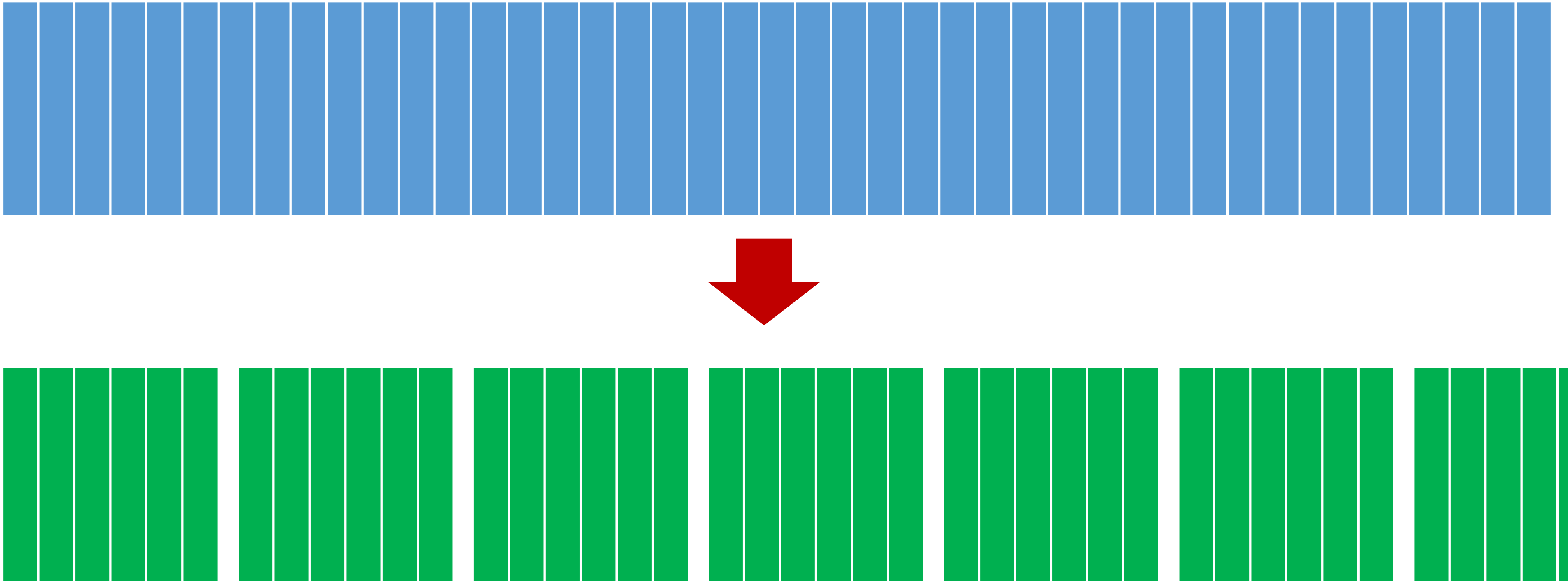


- The task is fundamentally linear in N: we have to touch every record no matter what

Can we do any better?

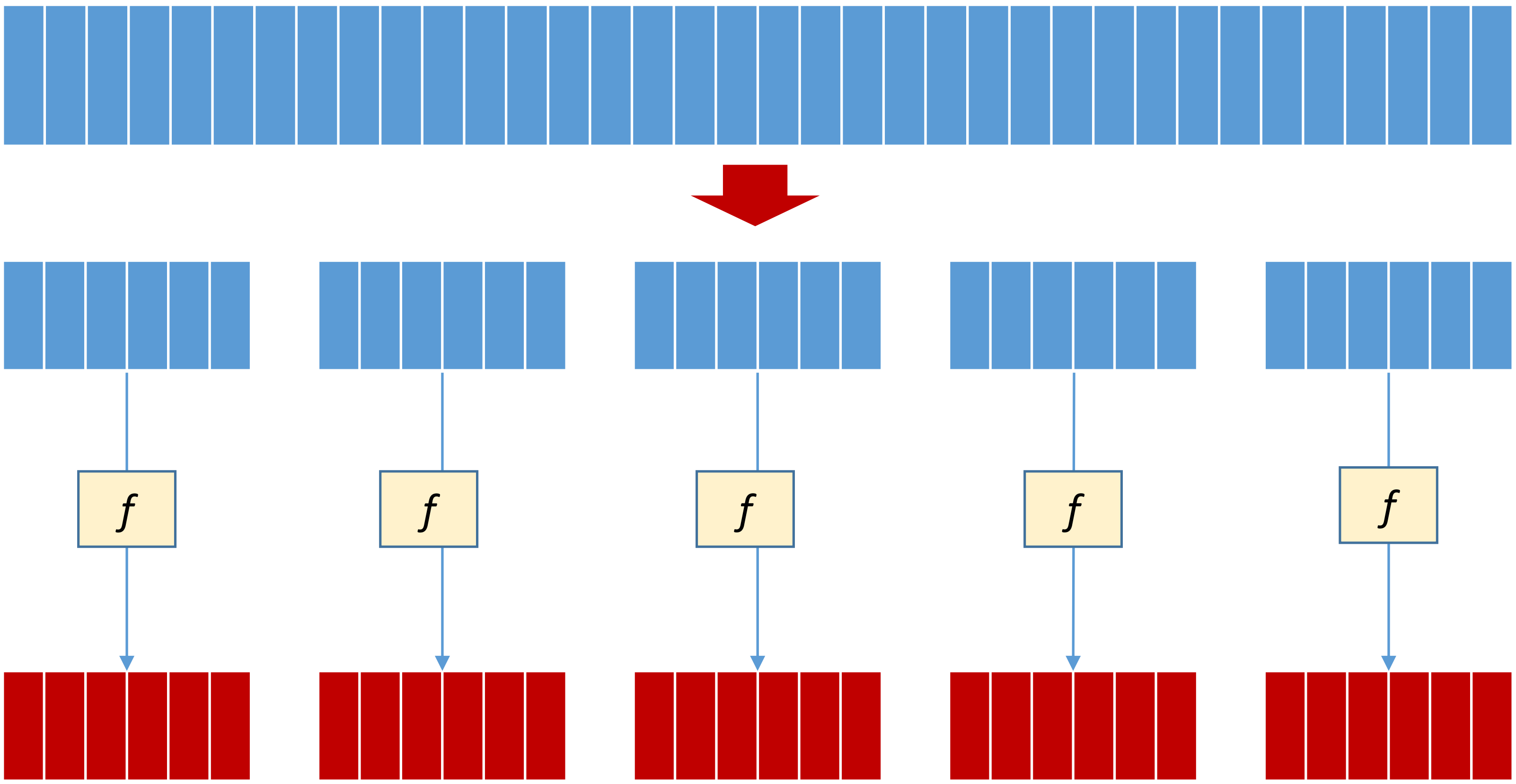
Will an index help?

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



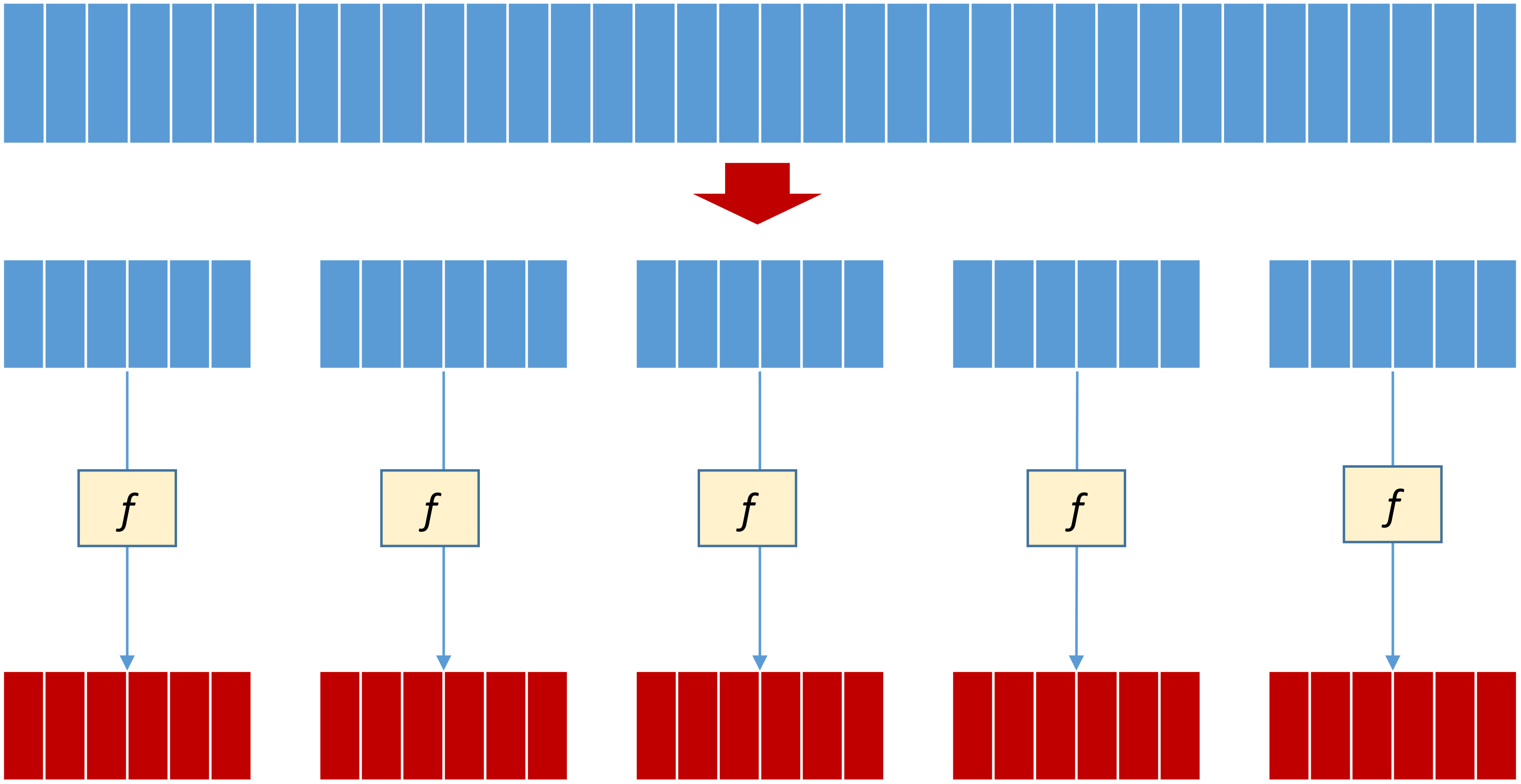
Input: short sequences

Distribute among K computers

Perform trimming on every read

A big distributed set of trimmed reads

Converting tiff images to png



Tiff images

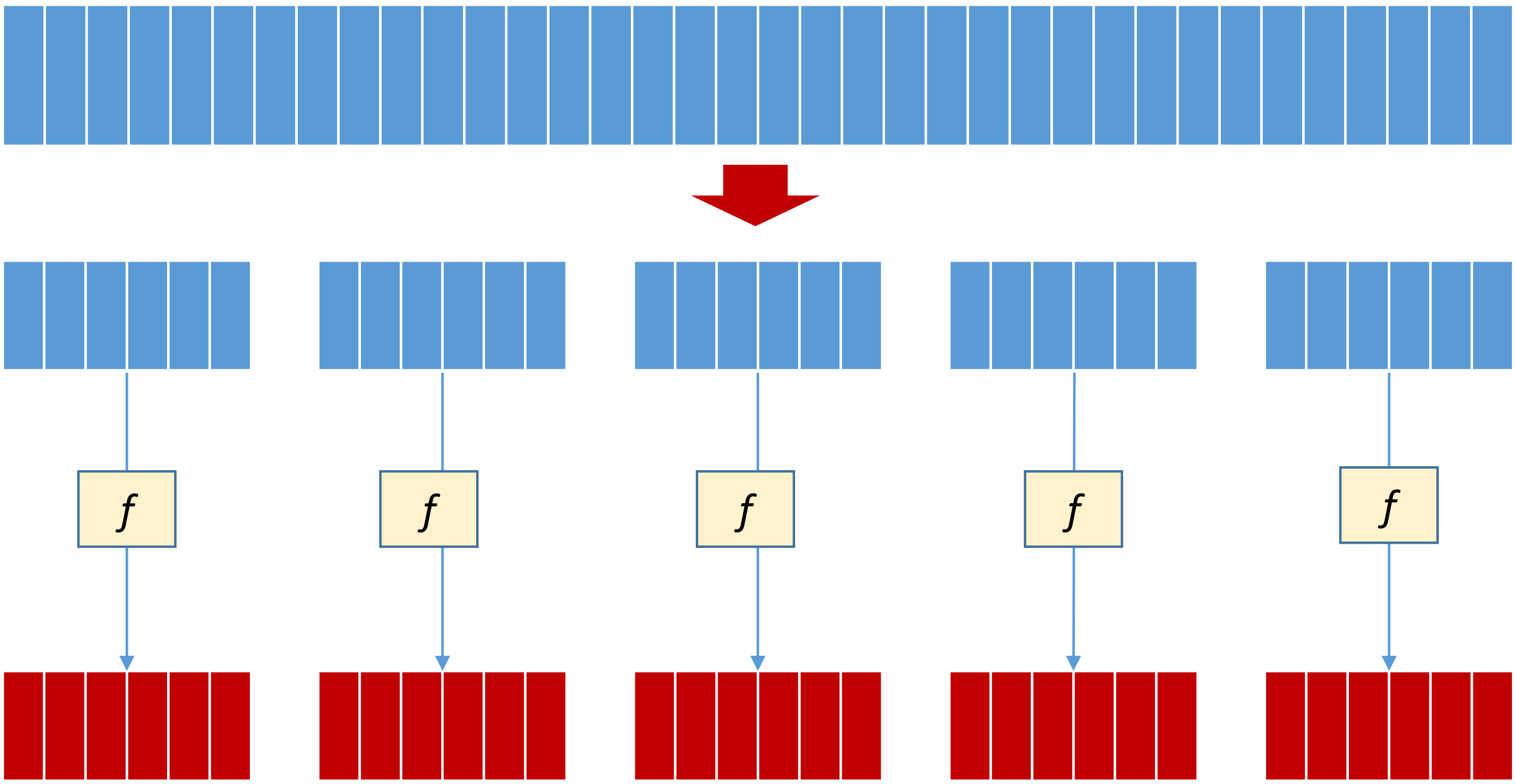
Distribute among K computers

Perform conversion of every file

A big distributed set of png images

https://aws.amazon.com/blogs/aws/new-york-times/

Simulations with multiple parameters



Sets of parameters for multiple short simulations

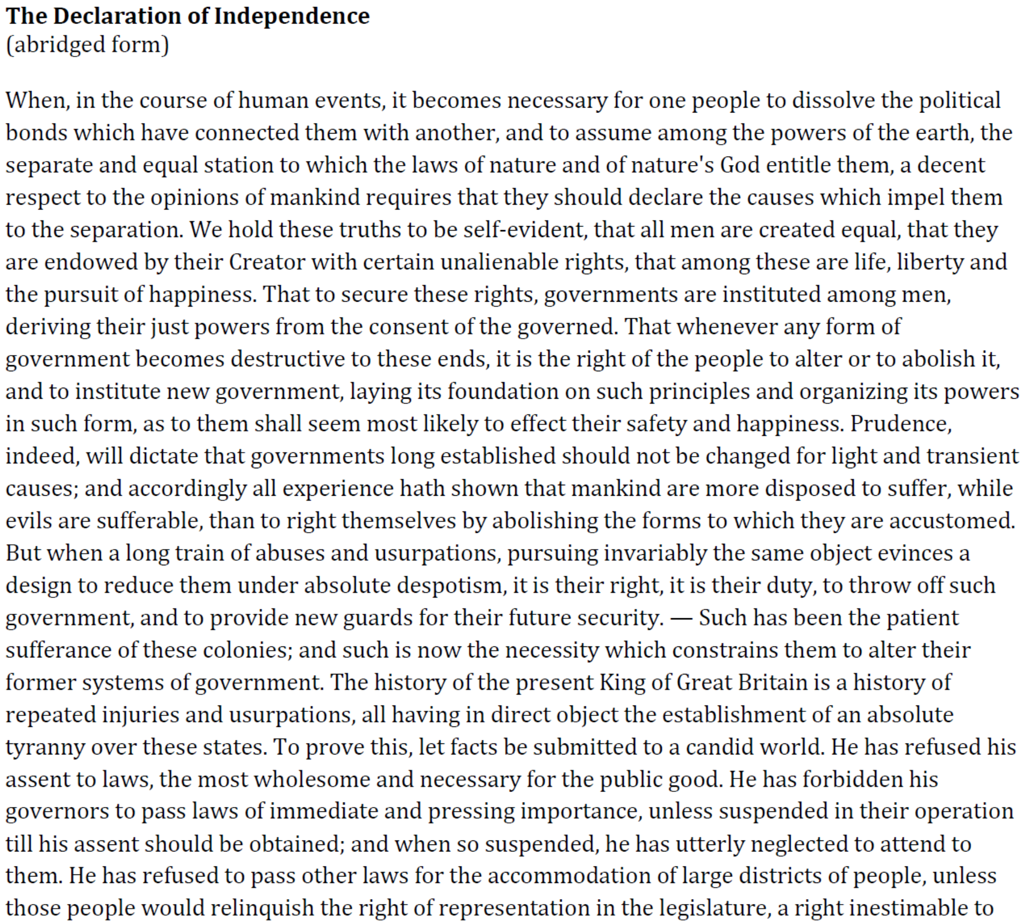
Distribute the parameter set among K computers

Run simulations with given parameters

A big distributed set of simulation results

https://www.sciencedaily.com/releases/2013/07/130712102844.htm

Compute word frequency of each word in a set of documents



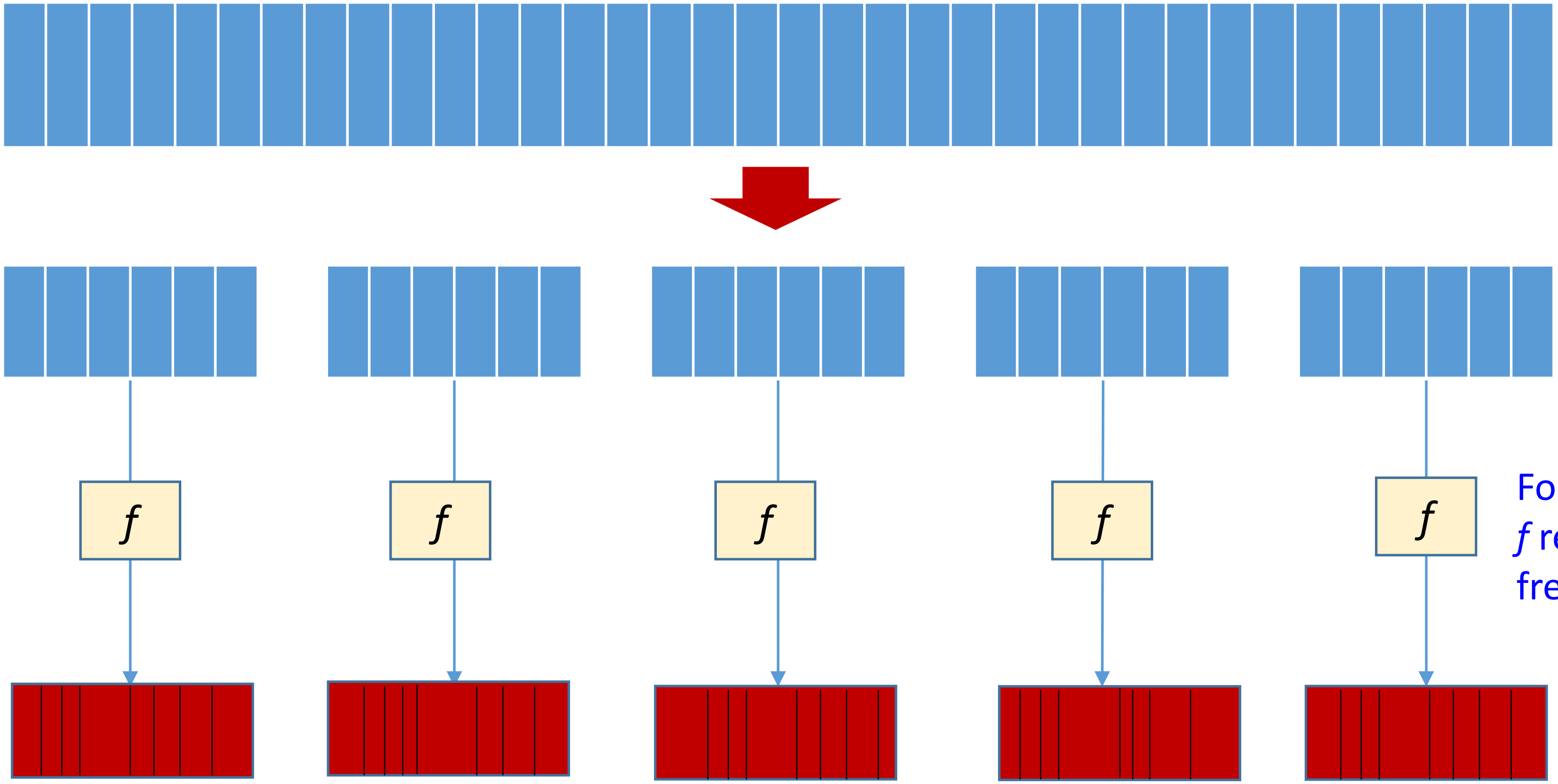
The Declaration of Independence
(abridged form)

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



Millions of documents

Distribute documents among K computers

For each document *f* returns (word, frequency) pairs

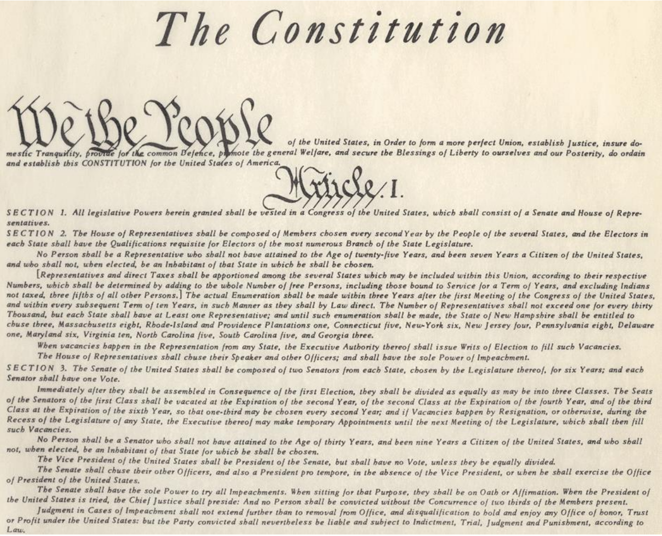
A big distributed list of word histograms per document

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



The Constitution

We the People of the United States, in Order to form a more perfect Union, establish Justice, insure domestic Tranquility, provide for the common Defence, promote the general Welfare, and secure the Blessings of Liberty to ourselves and our Posterity, do ordain and establish this CONSTITUTION for the United States of America.

Article. I.

SECTION 1. All legislative Powers herein granted shall be vested in a Congress of the United States, which shall consist of a Senate and House of Representatives.

SECTION 2. The House of Representatives shall be composed of Members chosen every second Year by the People of the several States, and the Electors in each State shall have the Qualifications requisite for Electors of the most numerous Branch of the State Legislature.

No Person shall be a Representative who shall not have attained to the Age of twenty-five Years, and been seven Years a Citizen of the United States, and who shall not, when elected, be an Inhabitant of that State in which he shall be chosen.

[Representatives and direct Taxes shall be apportioned among the several States which may be included within this Union, according to their respective Numbers, which shall be determined by adding to the whole Number of free Persons, including those bound to Service for a Term of Years, and excluding Indians not taxed, three fifths of all other Persons.] The actual Enumeration shall be made within three Years after the first Meeting of the Congress of the United States, and within every subsequent Term of ten Years, in such Manner as they shall by Law direct. The Number of Representatives shall not exceed one for every thirty Thousand, but each State shall have at Least one Representative; and until such enumeration shall be made, the State of New Hampshire shall be entitled to chuse three, Massachusetts eight, Rhode-Island and Providence Plantations one, Connecticut five, New-York six, New Jersey four, Pennsylvania eight, Delaware one, Maryland six, Virginia ten, North Carolina five, South Carolina five, and Georgia three.

When vacancies happen in the Representation from any State, the Executive Authority thereof shall issue Writs of Election to fill such Vacancies.

The House of Representatives shall chuse their Speaker and other Officers; and shall have the sole Power of Impeachment.

SECTION 3. The Senate of the United States shall be composed of two Senators from each State, chosen by the Legislature thereof, for six Years; and each Senator shall have one Vote.

Immediately after they shall be assembled in Consequence of the first Election, they shall be divided as equally as may be into three Classes. The Seats of the Senators of the first Class shall be vacated at the Expiration of the second Year, of the second Class at the Expiration of the fourth Year, and of the third Class at the Expiration of the sixth Year, so that one-third may be chosen every second Year; and if Vacancies happen by Resignation, or otherwise, during the Recess of the Legislature of any State, the Executive thereof may make temporary Appointments until the next Meeting of the Legislature, which shall then fill such Vacancies.

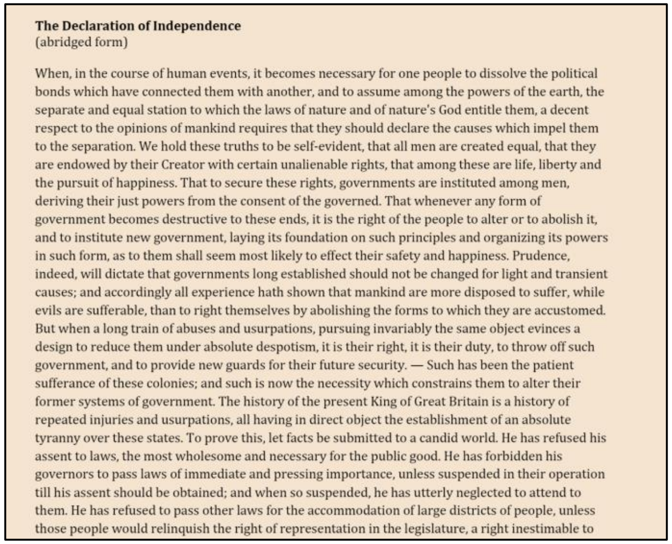
No Person shall be a Senator who shall not have attained to the Age of thirty Years, and been nine Years a Citizen of the United States, and who shall not, when elected, be an Inhabitant of that State for which he shall be chosen.

The Vice President of the United States shall be President of the Senate, but shall have no Vote, unless they be equally divided.

The Senate shall chuse their other Officers, and also a President pro tempore, in the absence of the Vice President, or when he shall exercise the Office of President of the United States.

The Senate shall have the sole Power to try all Impeachments. When sitting for that Purpose, they shall be on Oath or Affirmation. When the President of the United States is tried, the Chief Justice shall preside: And no Person shall be convicted without the Concurrence of two thirds of the Members present.

Judgment in Cases of Impeachment shall not extend further than to removal from Office, and disqualification to hold and enjoy any Office of honor, Trust or Profit under the United States: but the Party convicted shall nevertheless be liable and subject to Indictment, Trial, Judgment and Punishment, according to Law.



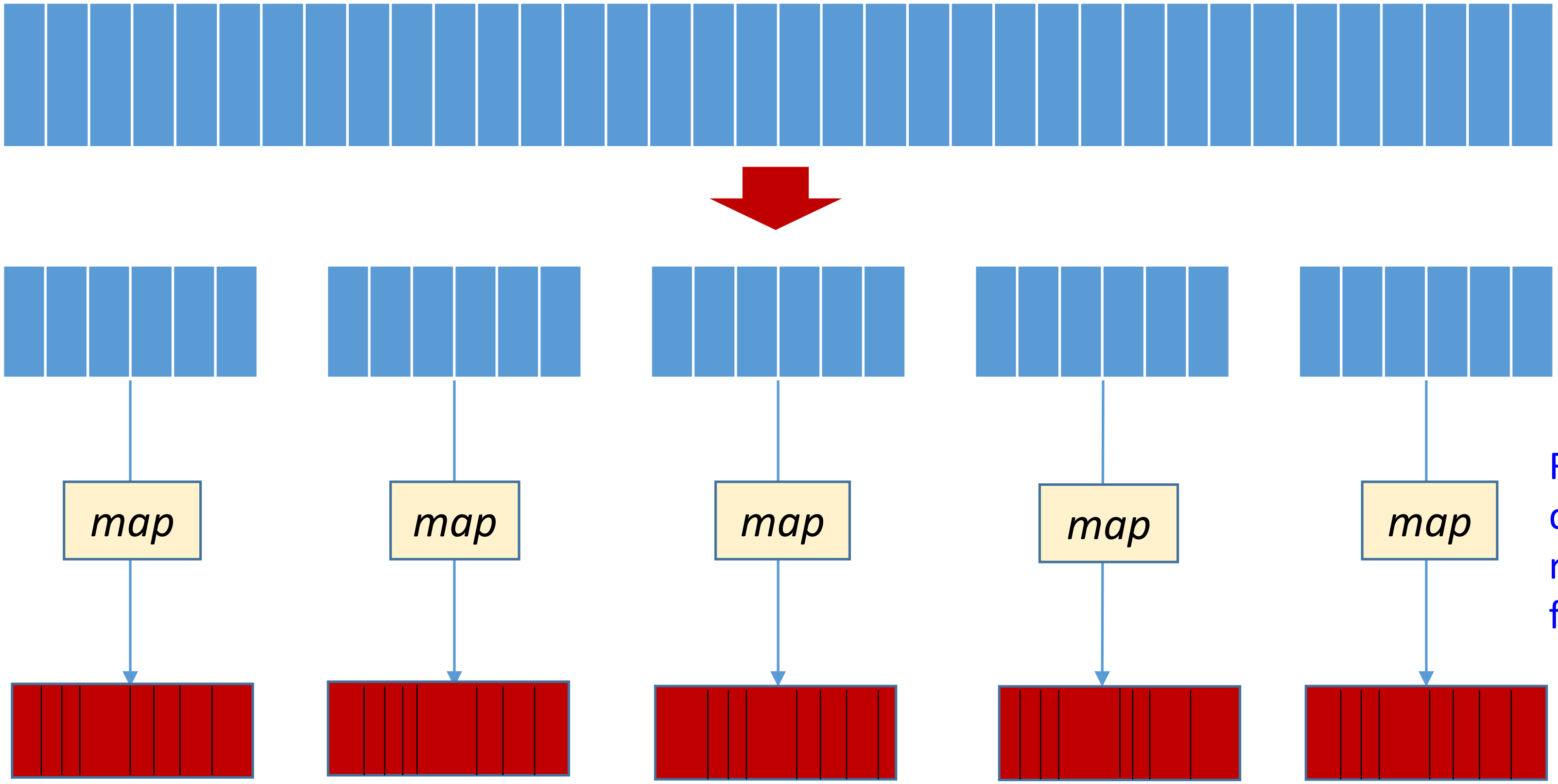
The Declaration of Independence
(abridged form)

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

To all to whom

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

Word frequencies among all documents



Millions of documents

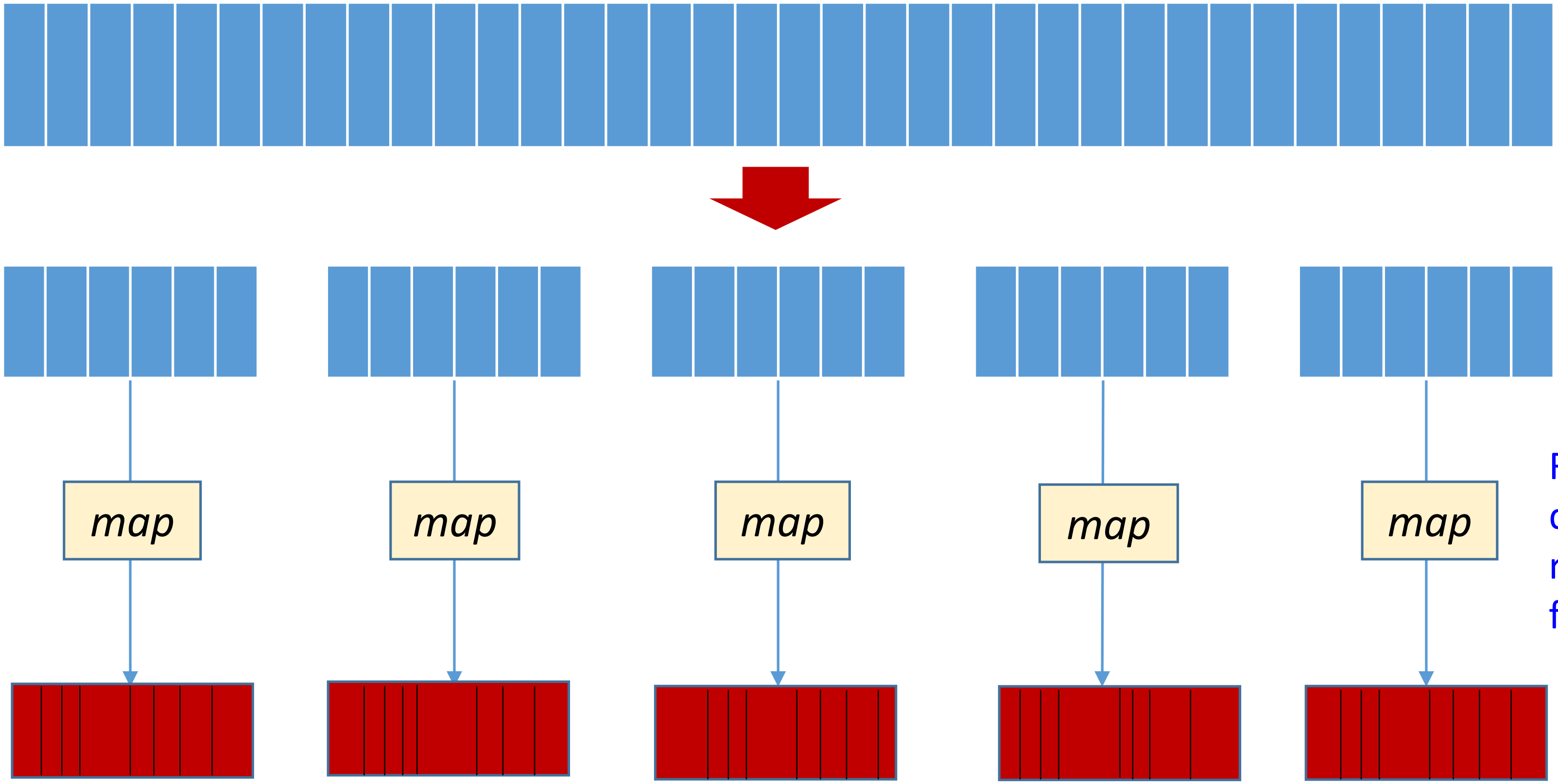
Distribute documents among K computers

For each document *map* returns (word, frequency) pairs

A big distributed list of word histograms per document

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

Word frequencies among all documents



map

map

map

map

map

Millions of documents

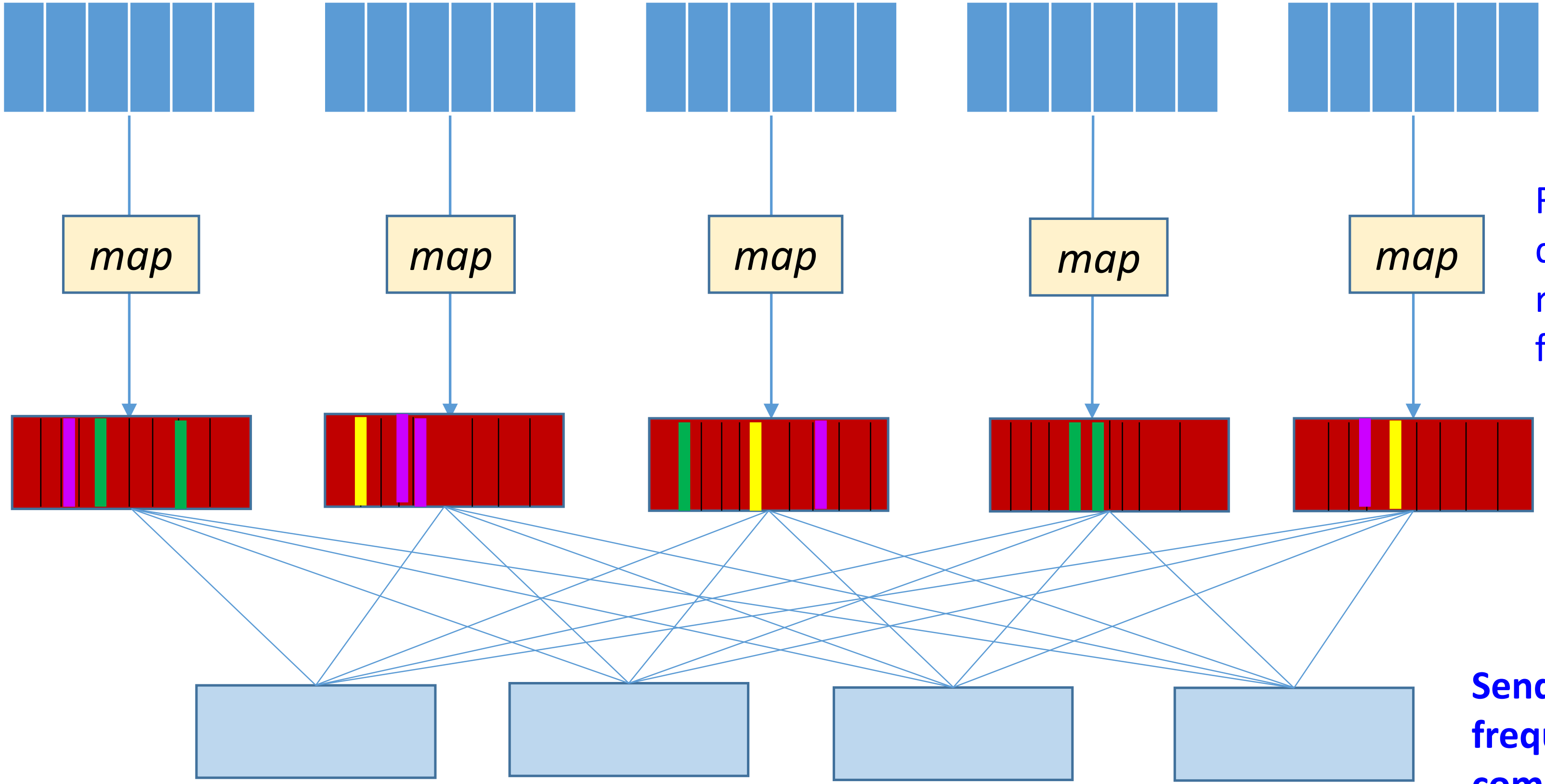
Distribute documents among K computers

For each document *map* returns (word, frequency) pairs

A big distributed list of word histograms per document

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

Word frequencies among all documents



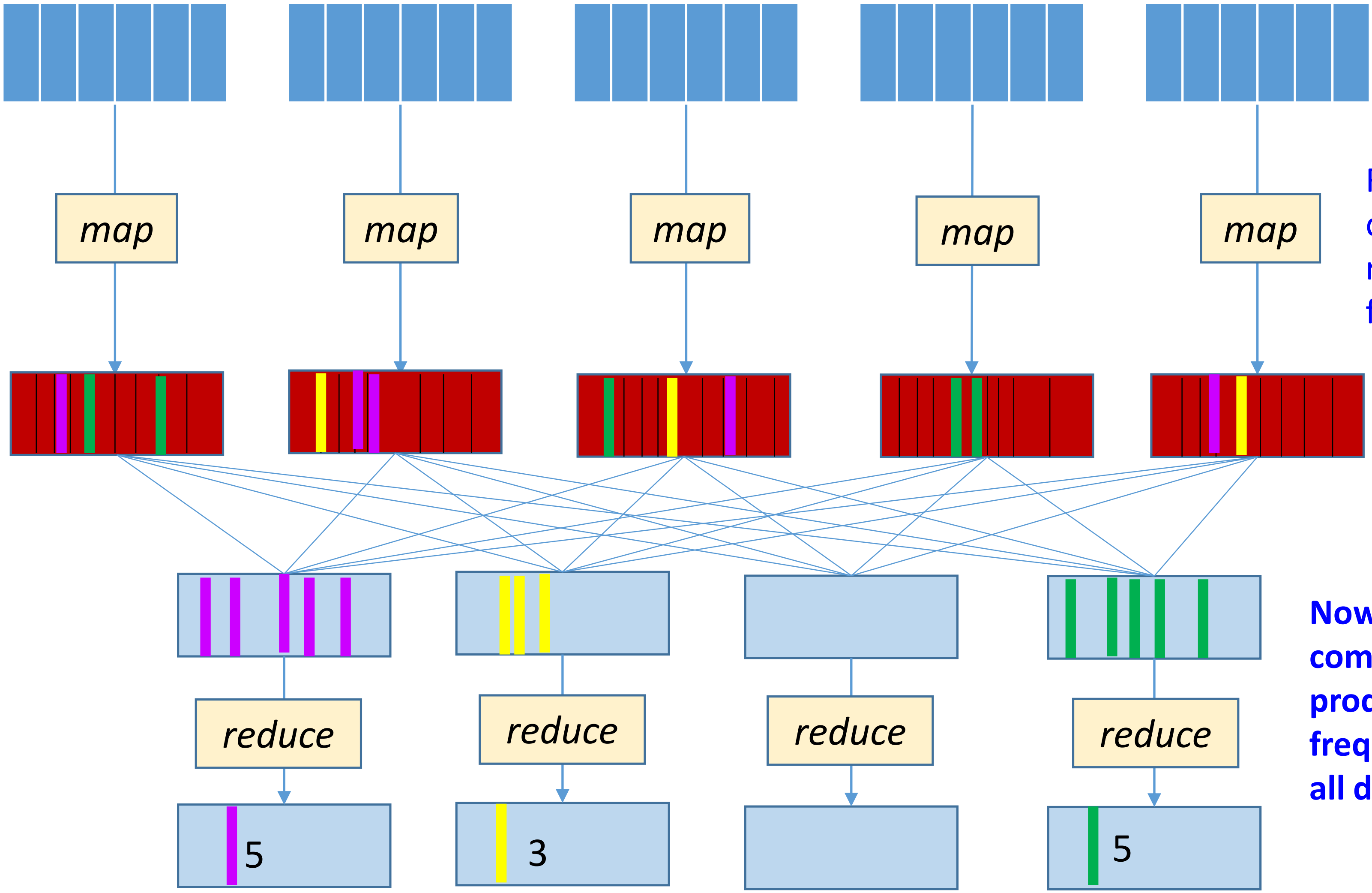
Distribute documents among K computers

For each document *map* returns (word, frequency) pair

A big distributed list of word histograms per document

Send each frequency pair into computer *i* using h(word) % K

Word frequencies among all documents



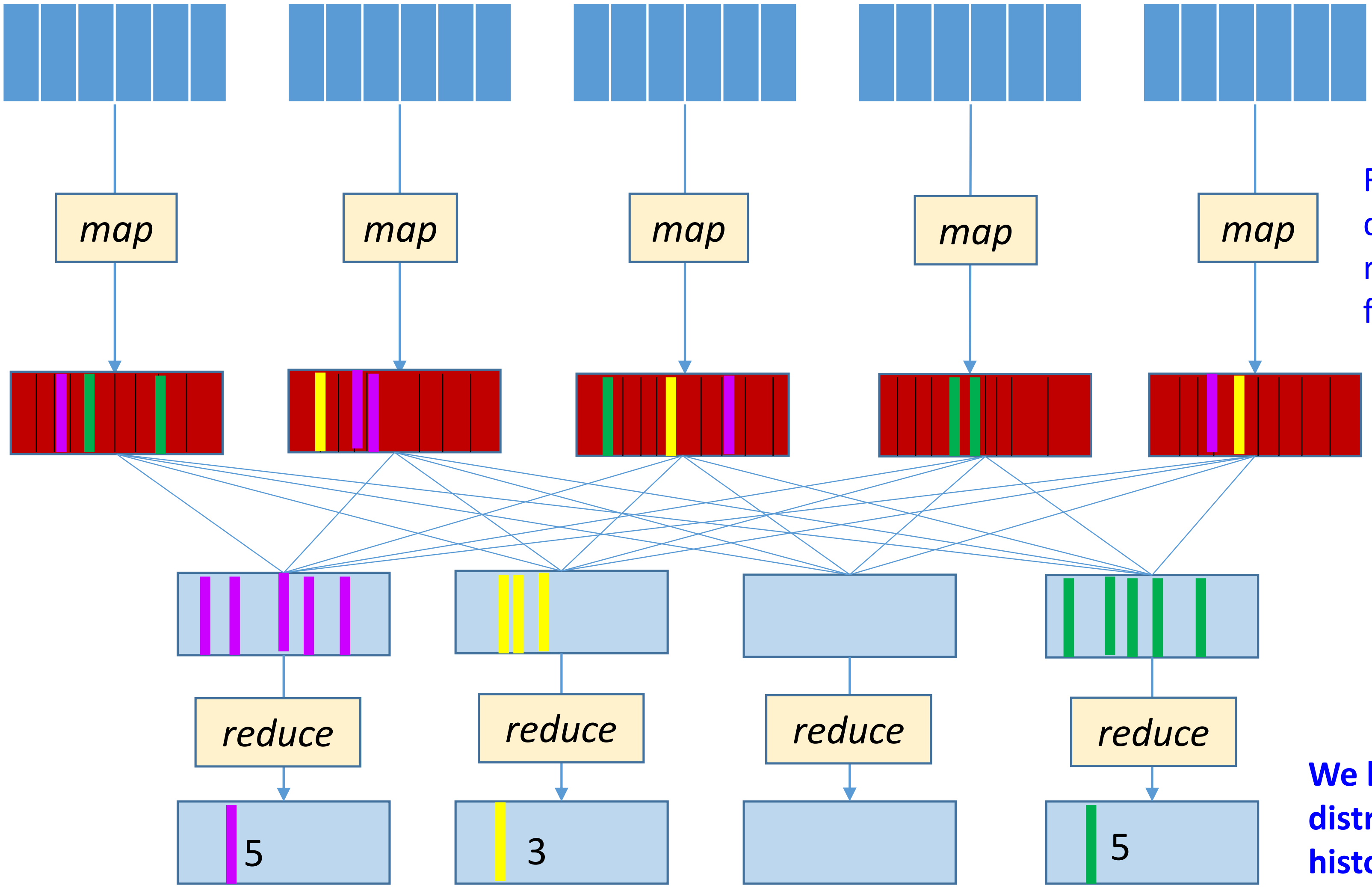
Distribute documents among K computers

For each document *map* returns (word, frequency) pair

A big distributed list of word histograms per document

Now each computer can produce (word, frequency) pair for all documents

Word frequencies among all documents



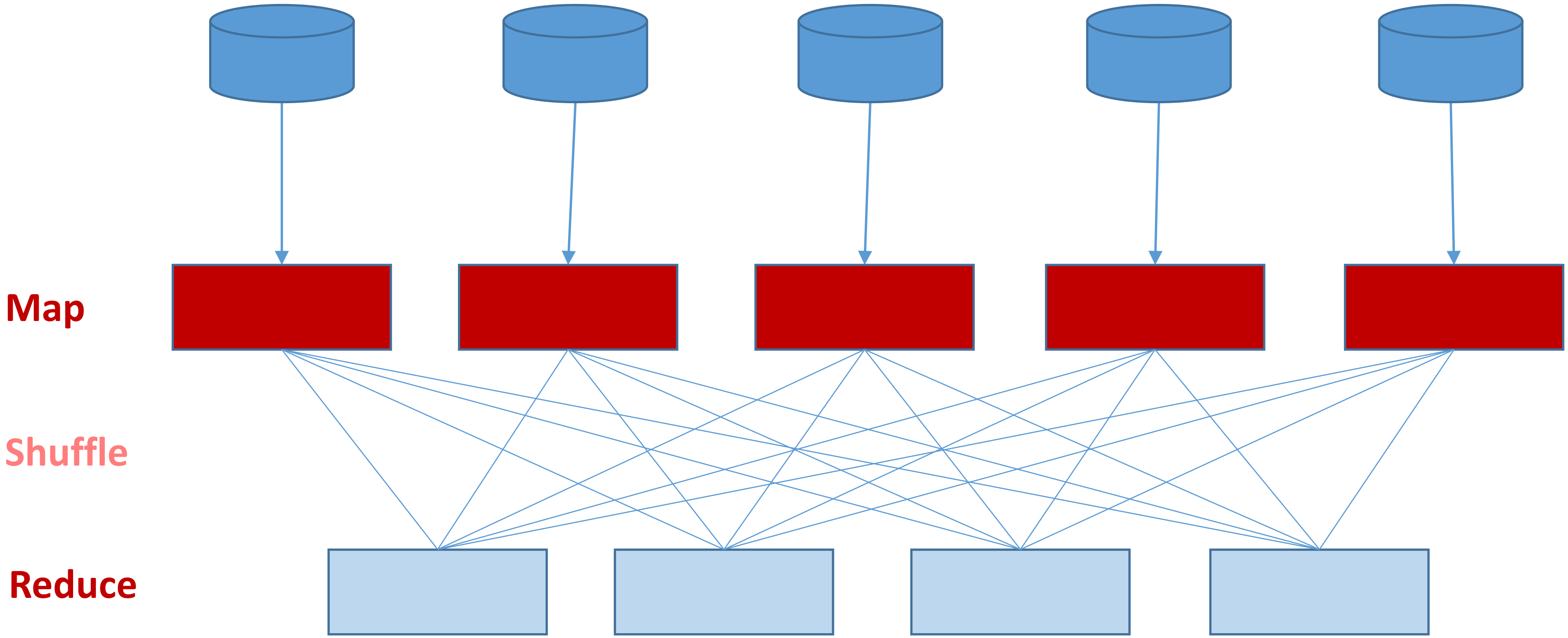
Distribute documents among K computers

For each document *f* returns (word, frequency) pair

A big distributed list of word histograms per document

We have a distributed general histogram

General idea: partitioning by hashing

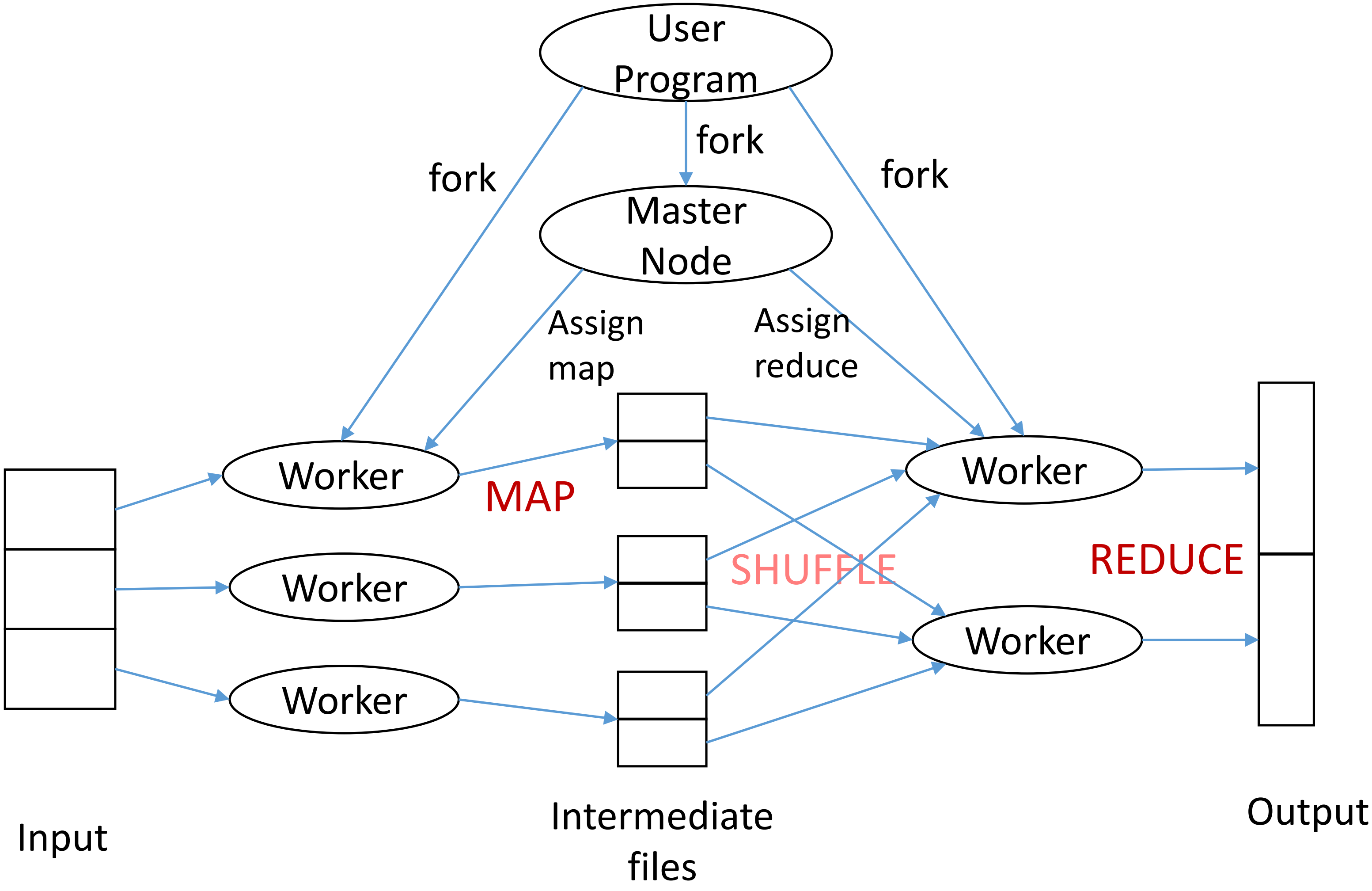


Only ***map*** and ***reduce*** differ from one application to another
Everything else is generic and is implemented in a map-reduce framework

Map-reduce

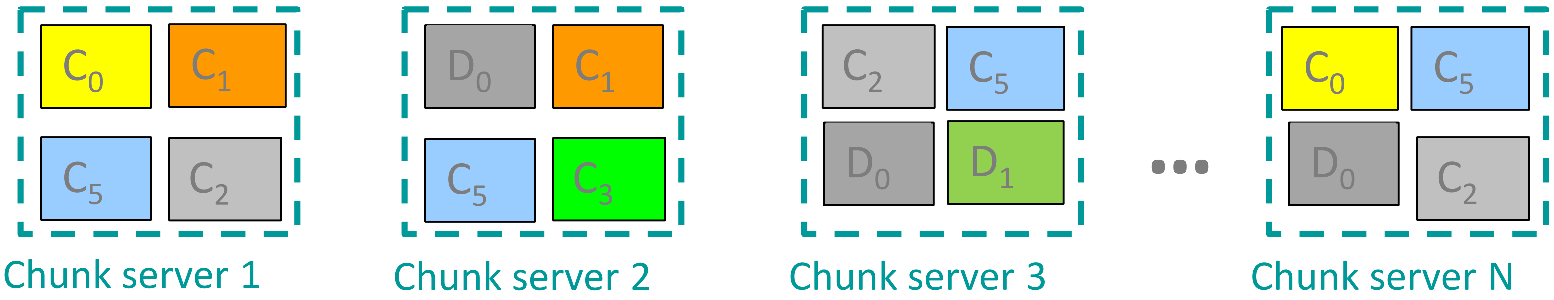
- The user writes two functions: map and reduce
- A master controller divides the **input data into chunks**, and assigns different processors to execute the map function on each chunk
- Other processors, perhaps the same ones, are then assigned to perform the reduce function on **chunks of the output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

Map-reduce framework



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

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(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 task 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)
```

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 $(k,v_1), (k,v_2), \ldots$ 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

Word count in Python

```python
# To run:
mr = MapReduce.MapReduce()


inputdata = open(sys.argv[1])
mr.execute(inputdata, mapper,
                      reducer)
```

```python
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))
```

In Java: https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html#Source+Code

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

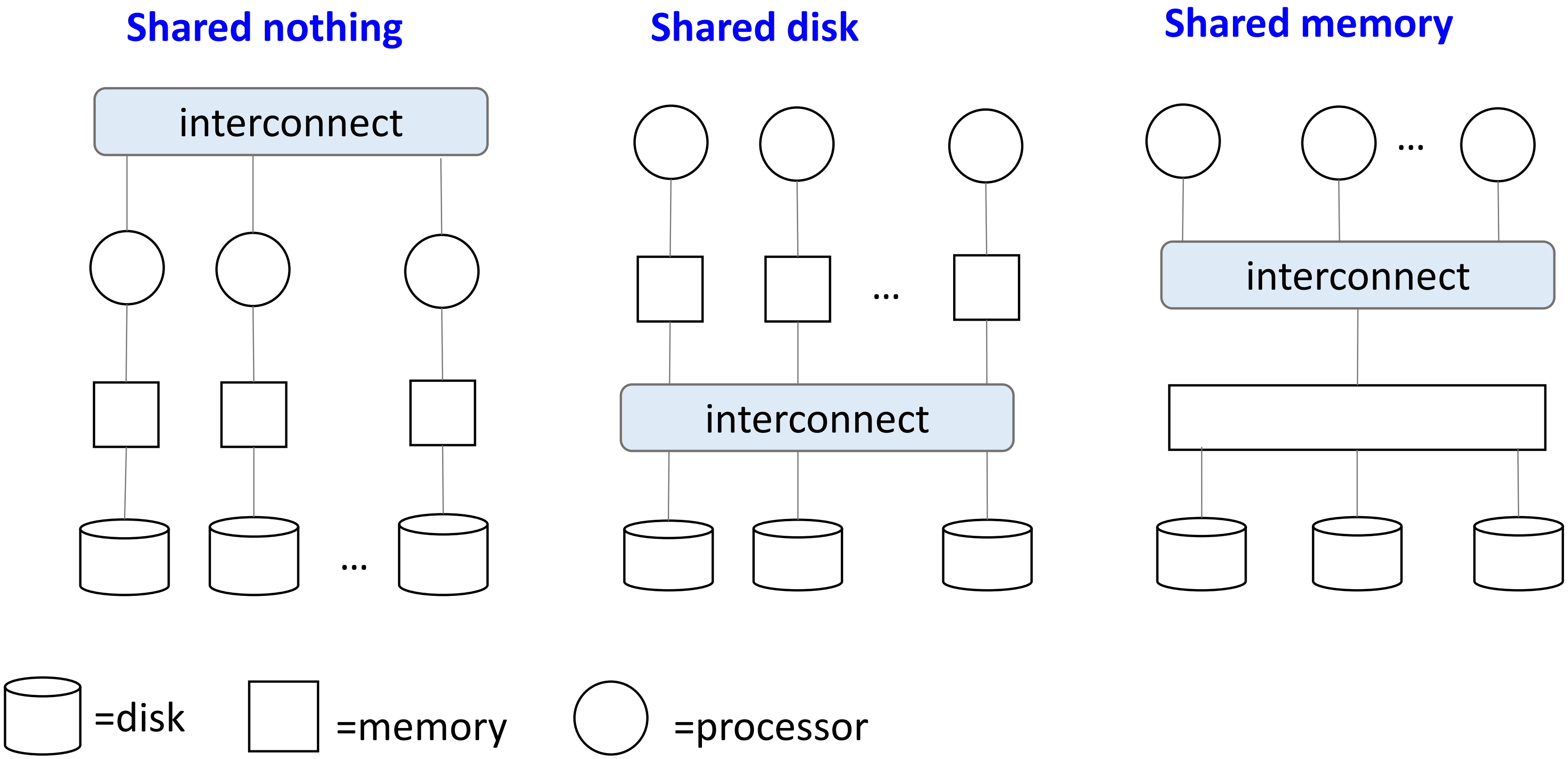
  - One server may stay up to 3 years (1,000 days)
  - If you have 1,000 servers, expect to loose 1/day
  - Google had ~1M machines in 2011: 1,000 machines fail every day!

- **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

What architecture is used for map-reduce?



Map-Reduce

Examples

Example 1: Language Model

- **Statistical machine translation:**
  - Need to count number of times every 5-word sequence occurs in a large corpus of documents

- **Very easy with MapReduce:**
  - **Map:**
    - Extract (5-word sequence, count) from document
  - **Reduce:**
    - Combine the counts

Example 2. Integers

- Design MapReduce algorithms to take a very large file of integers and produce as output:

(a) The largest integer.

(c) The same set of integers, but with each integer appearing only once.

(d) The count of the number of distinct integers in the input.

Max integer

```
map (file_id, Iterator numbers)
    max_local: = MIN_INTEGER
    for each number n in numbers
        if (n > max_local)
            max_local: = n
    emit_intermediate (“max”, max_local)

reduce (single_key, Iterator all_maxes)
    max_total: = MIN_INTEGER
    for each number n in all_maxes
        if (n > max_total)
            max_total : = n
    emit (“max_total” , max_total)
```

Example 3: Inverted index

- Each document has a unique document ID
- Forward index:
  - Given doc ID – retrieve document content
- Inverted index:
  - From document content to document ID
  - Similar (to secondary indexes) idea from information - retrieval community, but:
    - Record → document.
    - Search key → presence of a word in a document.

Inverted index for tweeter

- Input:
  - (tweet1, “I love pancakes for breakfast”)
  - (tweet2, “I dislike pancakes”)
  - (tweet3, “What should I eat for breakfast?”)
  - (tweet4, “I love to eat”)

- Output:
  - (“pancakes”, [twet1, tweet2])
  - (“breakfast”, [tweet1, tweet3])
  - (“eat”, [tweet3, tweet4])
  - (“love”, [tweet1, tweet4])

Inverted index

Input: distributed file with lines
(tweet_id, tweet_body)

```
map (input_key, input_value)
    for each line in input_value
        tokens: = split (line)
        tweet_id: = tokens[0]
        tweet_body: = tokens[1]
        for each word in tweet_body
            emit_intermediate (word, tweet_id)

reduce (word, Iterator tweet_ids)
```

Reduce is empty

Example 4: social network analysis

- Input:

→

Jim, Sue

Jim, Linn

Linn, Joe

Joe, Linn

Kai, Jim

Jim, Kai

- Output 1 Following (count):
- Jim, 3
- Sue, 0
- Linn, 1
- Joe, 1
- Kai, 1

- Output 2 Followers (count):
- Jim, 1
- Sue, 1
- Linn, 2
- Joe, 1
- Kai, 1

- Output 3 Friends (count):
- Jim, 1
- Sue, 0
- Linn, 1
- Joe, 1
- Kai, 1

Followers: list of followers for each user

```
map (file_name, edges)
    for each edge in edges
        emit_intermediate (edge[1], edge[0])

reduce (user_id, Iterator followers)
```

Example 5. Duplicate elimination

```
map (file_id, Iterator numbers)
    for each number n in numbers
        emit_intermediate (n, 1)

reduce (unique_number, Iterator all_occurrences)
        emit (unique_number , unique_number)
```

Example 6: PageRank and matrix-vector multiplication

- Originally, map-reduce was designed for fast computation of web page ranks using *PageRank* algorithm

How to rank web pages

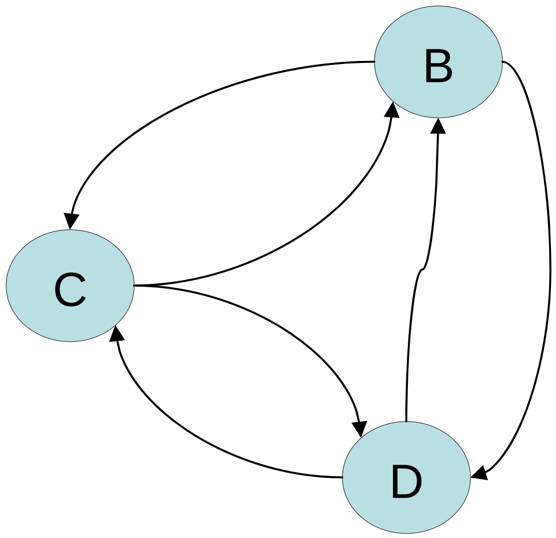
Definition: A webpage is important if many important pages link to it.

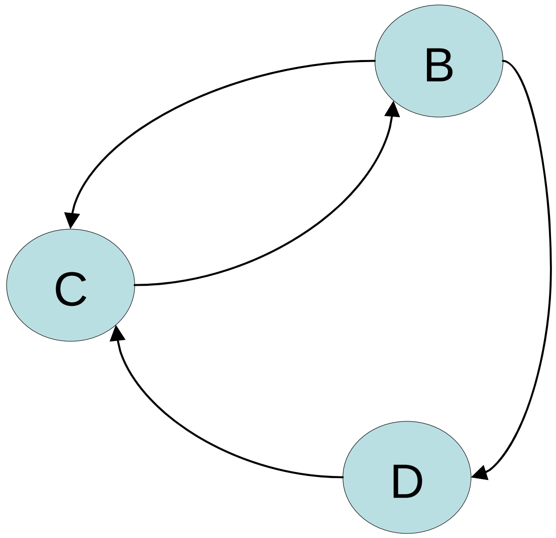
It seems that:

- a problem is the self-referential nature of this definition
- if we follow this line of reasoning, we might find that the importance of a web page depends on itself!

Modeling the web

What can we speculate about the relative importance of pages in each of these graphs, solely from the structure of the links (which is anyways the only information at hand)?



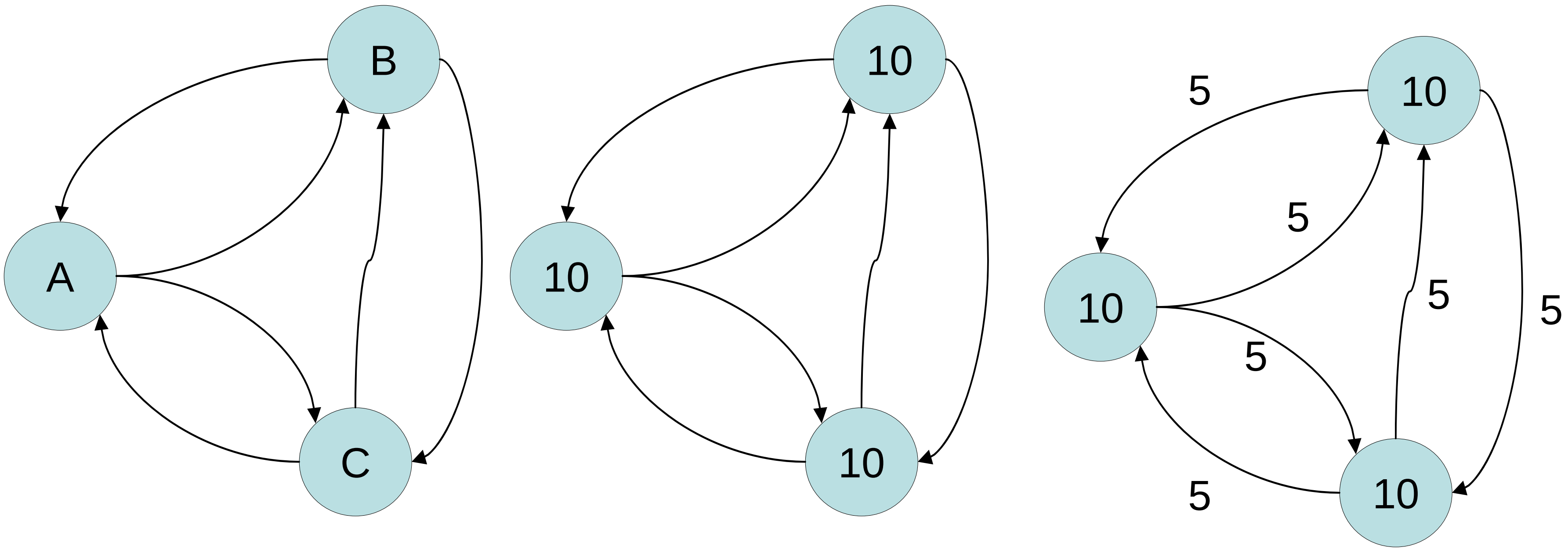


Model: traffic and mindless surfing

- Assumptions:
  - The WEB site is important if it *gets a lot of traffic*.
  - Let assume that everyone is surfing spending a second on each page and then randomly following one of the available links to a new page.
  - In this scheme it is convenient to make sure a surfer cannot get stuck, so we make the following

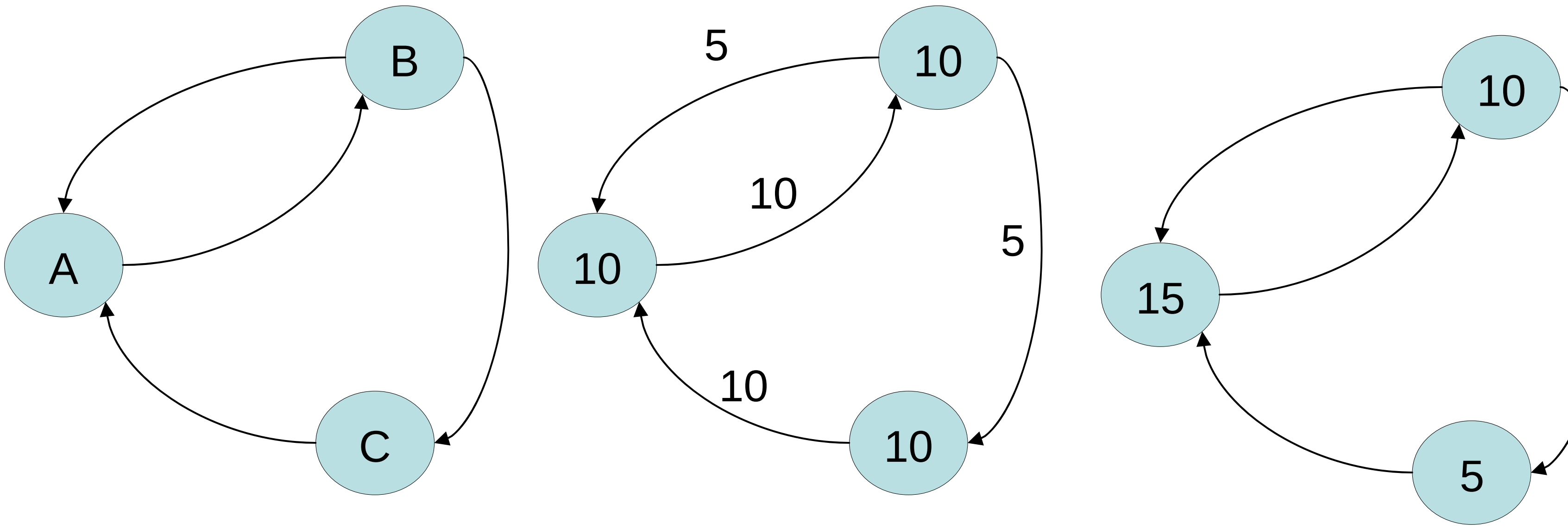
  STANDING ASSUMPTION: Each page has at least one outgoing link.

Stable traffic example



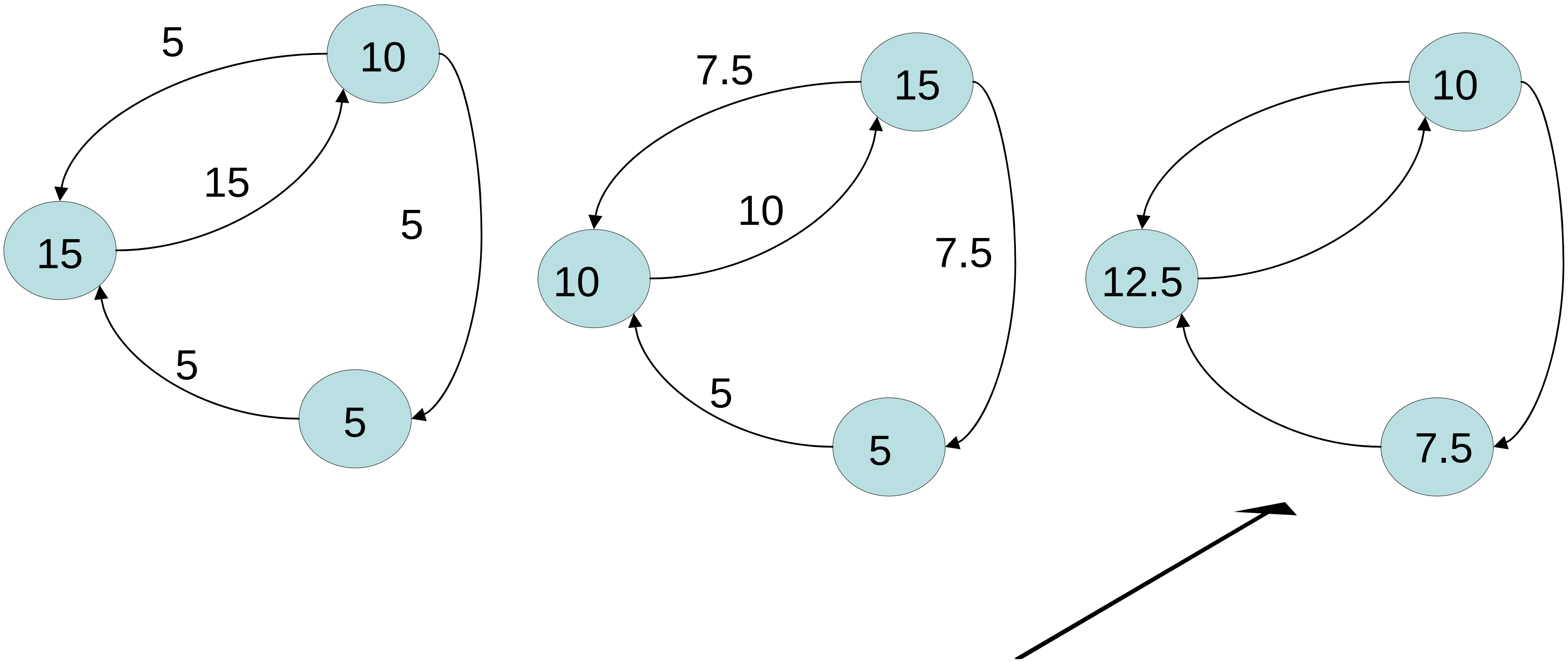
- We start with 10 surfers at each page
- At the first random click, 5 of the surfers at page A, say, go to page B, and the other 5 go to page C. So while each site sees all 10 of its visitors leave, it gets 5 + 5 incoming visitors to replace them:
- **So the amount of traffic at each page remains constant at 10.**

Unstable traffic example



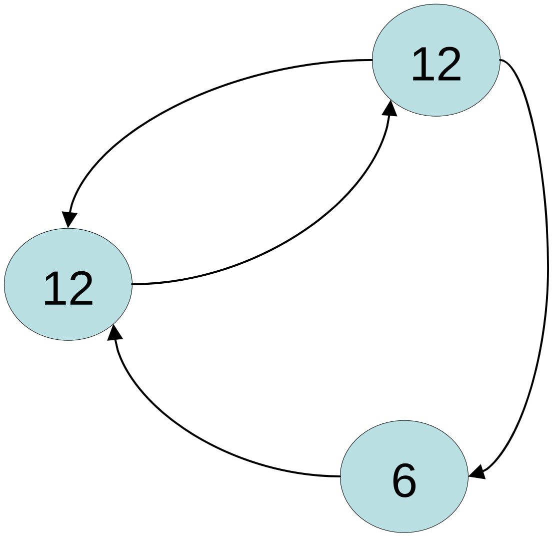
- We start with 10 surfers in each page
- After the first random click, 10 of the surfers at page A go to page B, since there is only 1 outgoing link from A etc…

Unstable traffic example contd.



- After the two next iterations it becomes
- Where is this leading? Do we ever reach a stable configuration, as in the first graph?

Traffic converges



- While the answer is no, it turns out that the process **converges** to the following distribution, which you can check oscillates around these values going forward in time
- This stable distribution is what the PageRank algorithm (in its most basic form) uses to assign a rank to each page: The two pages with 12 visitors are equally important, and each more important than the remaining page having 6 visitors.

Question

- How do we qualitatively explain why two of the pages in this model should be ranked equally, even though one has more incoming links than the other?

How to compute the stable distribution?

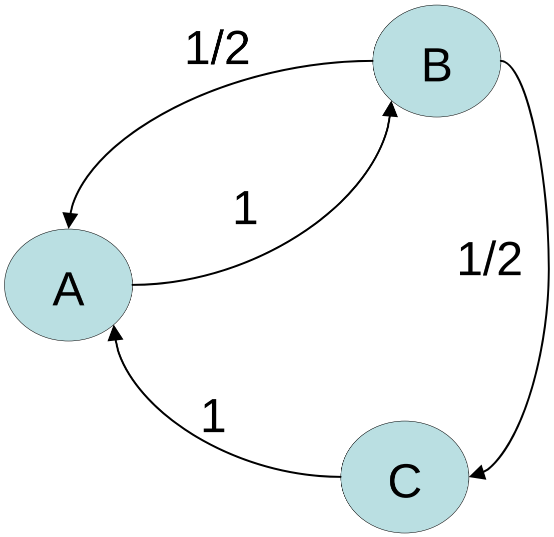
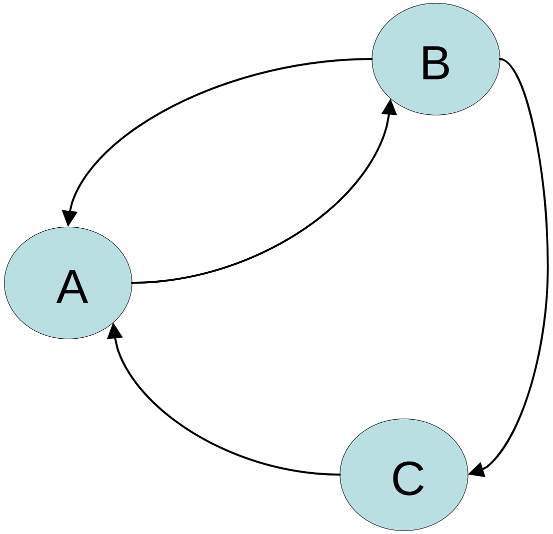


Table of transitions:

transition matrix based on outgoing links

Links from: ➡	A	B	C
A	0	1/2	1
B	1	0	0
C	0	1/2	0

Set initial importance for all pages to 1



Vector of importance

A	1
B	1
C	1

Transition matrix

	A	B	C
A	0	1/2	1
B	1	0	0
C	0	1/2	0

Iteration 1

Current Vector of importance

A	1
B	1
C	1

Transition matrix

	A	B	C
A	0	1/2	1
B	1	0	0
C	0	1/2	0

New Vector of importance

From B

From C

A	1*1/2 + 1*1 =1.5
B	1*1 = 1
C	1*1/2 = 1/2

Find new importance based on number of incoming visitors and their rank

Iteration 2

Current Vector of importance

A	1.5
B	1
C	0.5

Transition matrix

	A	B	C
A	0	1/2	1
B	1	0	0
C	0	1/2	0

New Vector of importance

From B

From C

A	1*1/2 + 0.5*1 =1
B	1.5*1 = 1.5
C	1*1/2 = 1/2

Find new importance based on number of incoming visitors and their rank

Iteration 3

Current Vector of importance

A	1
B	1.5
C	0.5

Transition matrix

	A	B	C
A	0	1/2	1
B	1	0	0
C	0	1/2	0

New Vector of importance

From B

From C

A	1.5*1/2 + 0.5*1 =1.25
B	1*1 = 1
C	1.5*1/2 = 0.75

Each entry of the vector is updated based on updated entries for other pages – they get updated together

Computing matrix-vector multiplication

$$A\mathbf{x} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \\ x_n \end{bmatrix} = \begin{bmatrix} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \end{bmatrix}$$

- Each entry of a new vector **y** is

$$y_i = \sum_{j=1\ldots n} a_{ij} * x_j$$

- In other words, it is a dot product of vector **x** with the corresponding row of matrix *A*

Note that the matrix is **very sparse**: each page has a limited number of outgoing and incoming links compared to the total number of web pages. So we are up to **compute several rounds of multiplication of a very sparse matrix by a very large vector**

Matrix-vector multiplication

V_{k-1} - Current Vector of importance

A	1
B	1.5
C	0.5

A - Transition matrix

	A	B	C
A	0	1/2	1
B	1	0	0
C	0	1/2	0

V_k - New Vector of importance

From B

From C

A	1.5*1/2 + 0.5*1 =1.25
B	1*1 = 1
C	1.5*1/2 = 0.75

The new vector at each iteration is the result of matrix-vector multiplication:

$V_k = A * V_{k-1}$

Basic matrix-vector multiplication in map-reduce: **input**

- Transition matrix (sparse), stored as tuples of type:

(i, j, A_{ij})

- Current vector of page importance, stored as tuples of type

(i, v_i)

Basic matrix-vector multiplication in map-reduce: **map**

Input: two types of tuples
(i, j ,A_{ij})
(i, v_i)

```
map
    for each tuple of type A emit_intermediate (i, (i,j,$A_{ij}$))
    for each tuple of type v
        for j from 1 to n
            emit_intermediate (j, (i, vi))
```

Step-by-step example: **reduce**

- At each reducer:

Multiply non-zero entries of row 1 of A by values of v, sum them up and emit result (1, ½+1)

- (1, [(1,2,**1/2**), (1,3,**1**), (1,1), (2,**1**), (3,**1**)])
- (2, [(2,1,1), (1,1), (2,1), (3,1)])
- (3, [(3,2,1/2), (1,1), (2,1), (3,1)])

Basic matrix-vector multiplication in map-reduce: **reduce**

- The Reduce function simply sums all the values associated with a given row ***i***. The result will be a pair **(i, new v_i)**.

We have a distributed file of new entries of v: finished one iteration of PageRank algorithm

Partitioned Matrix-Vector multiplication: main idea

$$A\mathbf{x} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \\ x_n \end{bmatrix} = \begin{bmatrix} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \end{bmatrix}$$

$$y_1 = \sum_{j=1\ldots k} part\ k$$

$$part\ 1 = \sum_{j=1\ldots2} a_{ij} * v_j$$

$$A\mathbf{x} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \\ x_n \end{bmatrix} = \begin{bmatrix} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \end{bmatrix}$$

- Partition matrix into strips, partition vector into chunks
- Entries i…j of vector v are multiplied only by columns i…j of matrix A
- We can perform these partial multiplications as an additional intermediate step of map-reduce, and sum the results in the final step
- The flexibility of map-reduce is that at each step both input and output are a set of key-value pairs

Implementations

- Google
  - Not available outside Google
- **Hadoop**
  - An open-source implementation in Java
  - Uses HDFS for stable storage
  - Download: http://lucene.apache.org/hadoop/
- Aster Data
  - Cluster-optimized SQL Database that also implements MapReduce

Summary

- Learned how to ***scale out*** processing of large inputs
- **Map-reduce** framework allows to implement only 2 functions and the system takes care of distributing computations across multiple machines
- Memory footprint is small. Need to care about the size of intermediate outputs – sending them across network may dominate the cost
- We can perform relational operations in map reduce, if the relations are too big to be processed on a single machine

Map-reduce vs. RDBMS

- RDBMS
  - Declarative query languages
  - Schemas
  - Logical data independence
  - Indexing
  - Algebraic optimization
  - ACID/Transactions

- Map-reduce
  - High scalability
  - Fault-tolerance
  - “One-person deployment”

Spark, Pig, Hive, DryadLINQ – try to incorporate this into map-reduce