MAP REDUCE

Word Count Example

- We have a large file of words, one word to a line
- Count the number of times each distinct word appears in the file
- Sample application: analyze web server logs to find popular URLs

Word Count (2)

- Case 1: Entire file fits in memory

Word Count (2)

- Case 1: Entire file fits in memory
- Case 2: File too large for mem, but all <word, count> pairs fit in mem

Word Count (2)

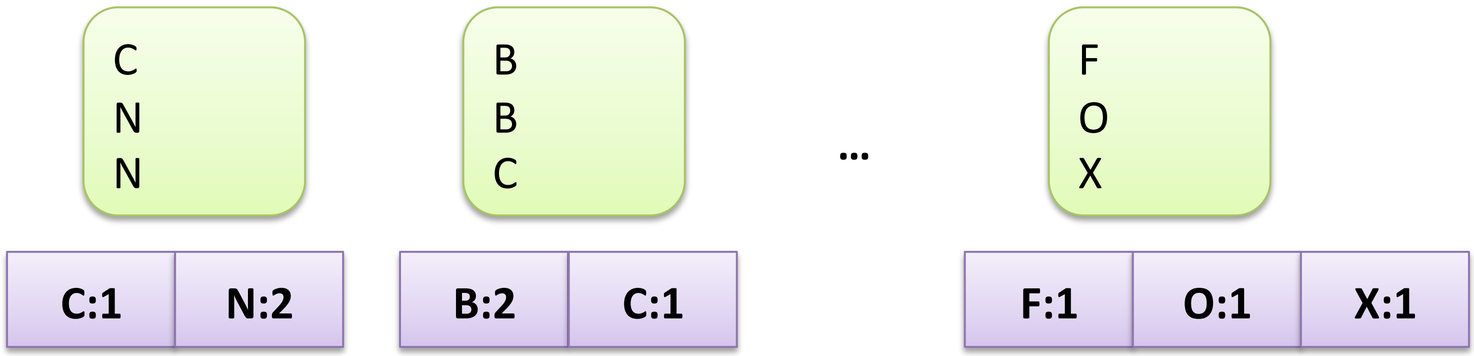
- Case 1: Entire file fits in memory
- Case 2: File too large for mem, but all <word, count> pairs fit in mem
- Case 3: File on disk, too many distinct words to fit in memory
  - `sort datafile | uniq –c`

Word Count (3)

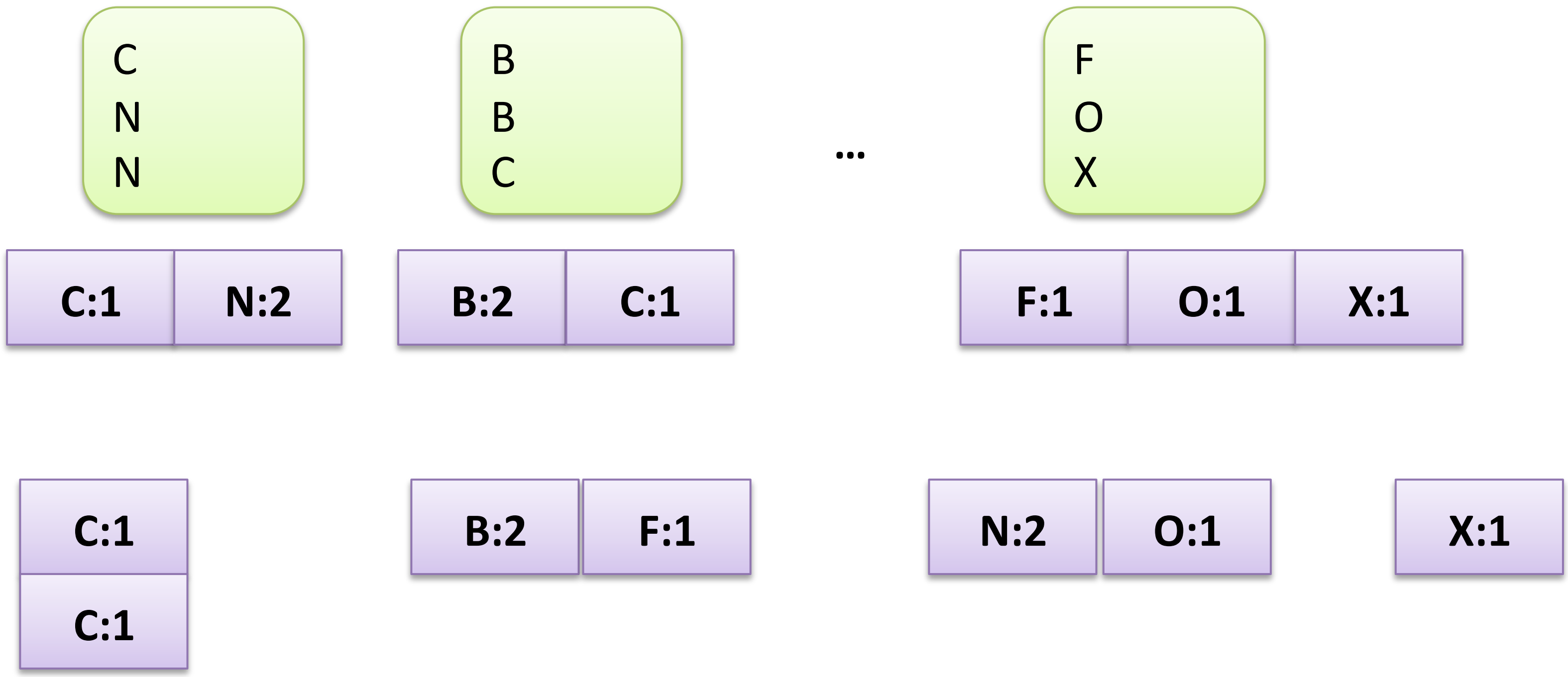
- A large corpus of documents, sharded across many disks in many machines
- Machines, disks, networks can fail
- Motivation for Google's MapReduce

Input: A set of files

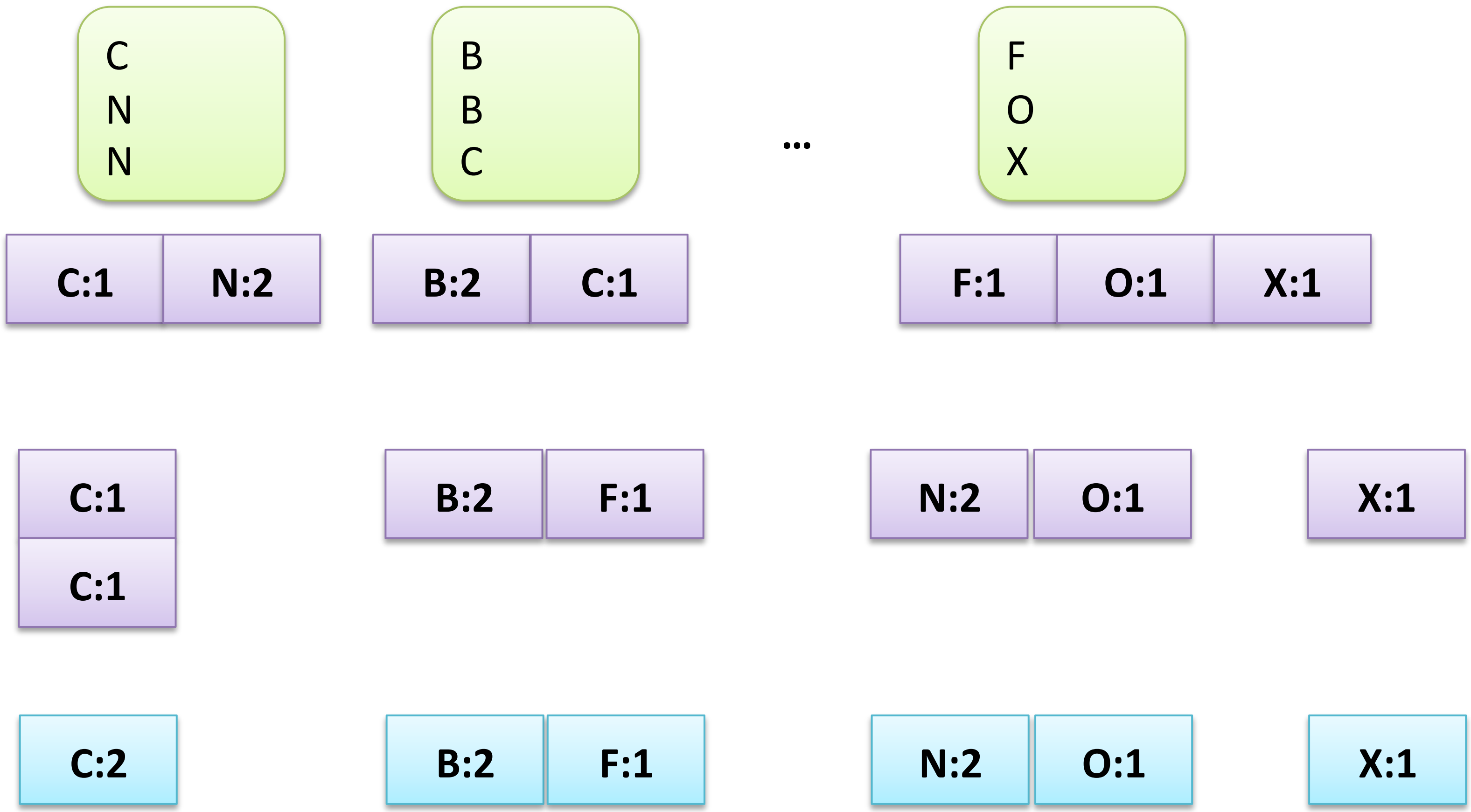
Map: Generate Word Count Per File



Partition (Optional)



Reduce



MapReduce

- Input: a set of key/value pairs
- User supplies two functions:
  - map(k,v) → list(k1,v1)
  - reduce(k1, list(v1)) → v2
- (k1,v1) is an intermediate key/value pair
- Output is the set of (k1,v2) pairs

Word Count using MapReduce

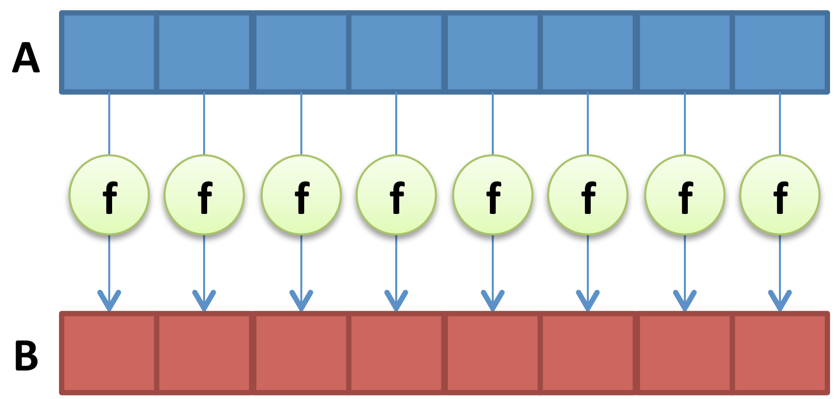
```
map(key, value):
// key: document name; value: text of document
  for each word w in value:
          emit(w, 1)


reduce(key, values):
// key: a word; value: an iterator over counts
          result = 0
          for each count v in values:
                    result += v
          emit(result)
```

Map and Reduce vs MapReduce

- The map and reduce operations in MapReduce are inspired by similar operations in functional programming

Map

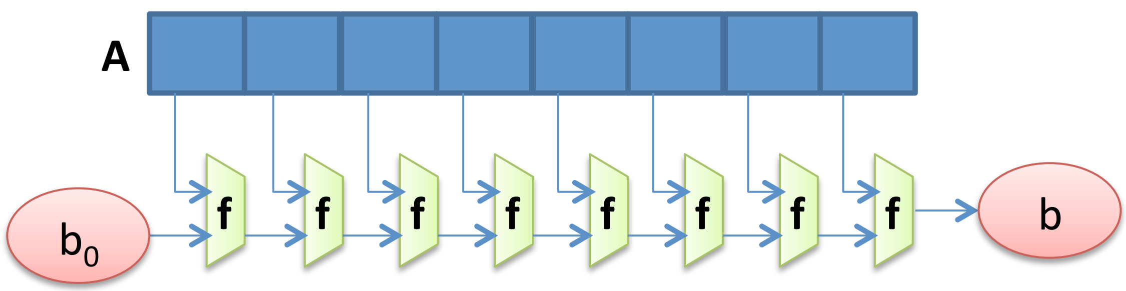
- Given a function f : (A) => B
- A collection a: A[]
- Generates a collection b: B[], where B[i] = f(A[i])



- Parallel.For, Paralle.ForEach
  - Where each loop iteration is independent

Reduce

- Given a function f: (A, B) => B
- A collection a: A[]
- An initial value b_0: B
- Generate a final value b: B
  - Where b = f(A[n-1], … f(A[1], f(A[0], b_0)))



Relationship to SQL

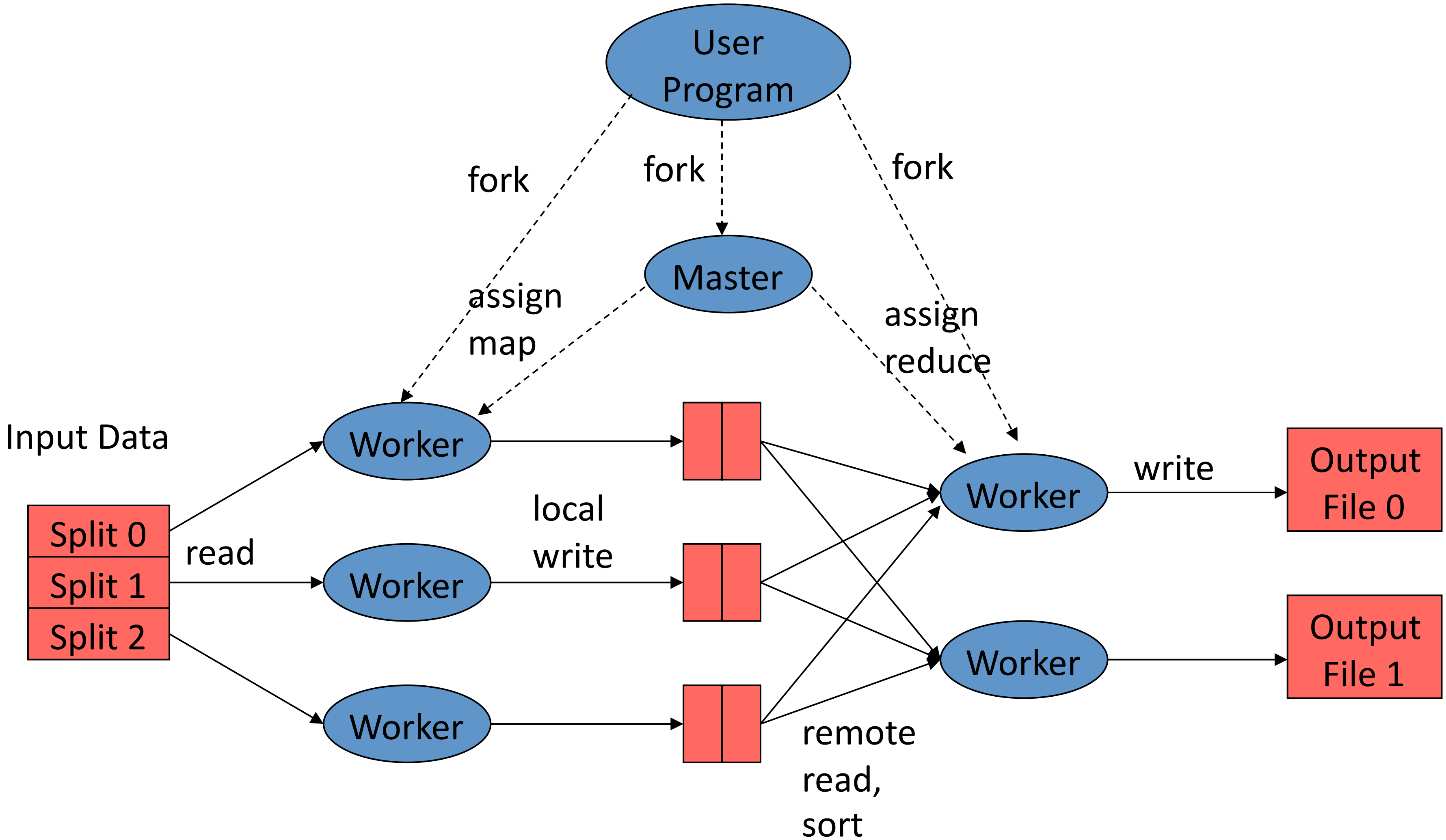
- Implementing word count in SQL

```
SELECT word Count(*) as wordCount
FROM files
GROUP BY word;

// where files is a (distributed)
// relation <name, posn, word>
```

Signs of a Good Abstraction

- Hides important details
  - But not too much
- Simple for lay programmers to use
- Not necessarily general
  - But not very restricted
  - Can be application/domain specific
- Allows efficient implementations
  - Automatic optimizations
  - Manual optimizations (by experts)

Distributed Execution Overview



Data flow

- Input, final output are stored on a distributed file system
  - Scheduler tries to schedule map tasks “close” to physical storage location of input data
- Intermediate results are stored on local FS of map and reduce workers
- Output is often input to another map reduce task

Coordination

- Master data structures
  - Task status: (idle, in-progress, completed)
  - Idle tasks get scheduled as workers become available
  - When a map task completes, it sends 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

Failures

- Map worker failure
  - Map tasks completed or in-progress at worker are reset to idle
  - Reduce workers are notified when task is rescheduled on another worker
- Reduce worker failure
  - Only in-progress tasks are reset to idle
- Master failure
  - MapReduce task is aborted and client is notified

How many Map and Reduce jobs?

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

Combiners

- Often a map task will produce many pairs of the form (k,v1), (k,v2), … for the same key k
  - E.g., popular words in Word Count
- Can save network time by pre-aggregating at mapper
  - combine(k1, list(v1)) → v2
  - Usually same as reduce function
- Works only if reduce function is commutative and associative

Partition Function

- Inputs to map tasks are created by contiguous splits of input file
- For reduce, we need to ensure that records with the same intermediate key end up at the same worker
- System uses a default partition function e.g., hash (key) mod R
- Sometimes useful to override
  - E.g., hash(hostname(URL)) mod R ensures URLs from a host end up in the same output file

Avoiding Stragglers

- A slow running task (straggler) can prolong overall execution
  - Overloaded machines
  - Slow disk
- Kill stragglers
- Fork redundant tasks and take the first

Example: Sorting

Example: Database Join

Can Mappers Push instead of Reducers Pulling Data ?