Index Construction

Introduction to Information Retrieval

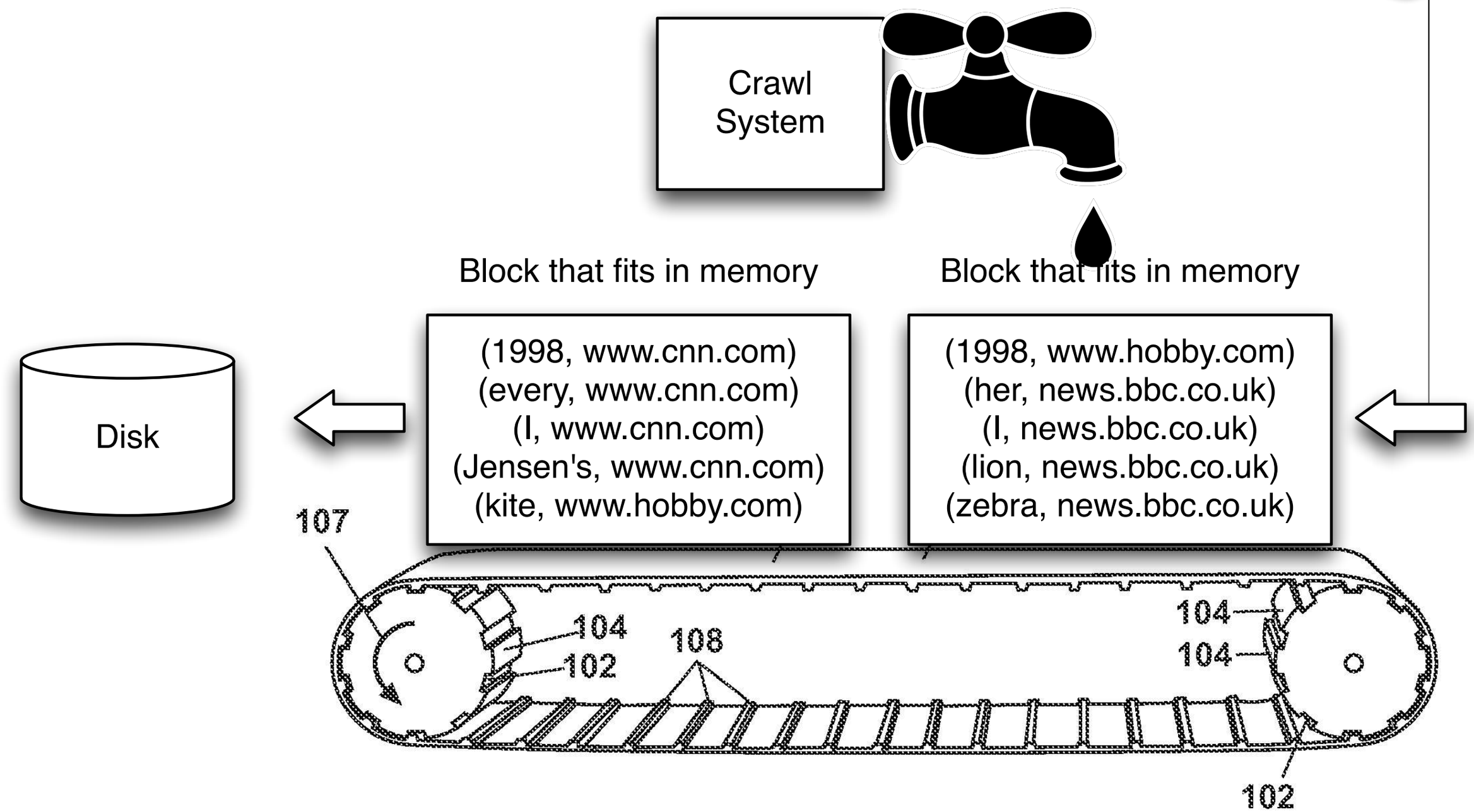
CS 150

Donald J. Patterson

Content adapted from Hinrich Schütze
http://www.informationretrieval.org

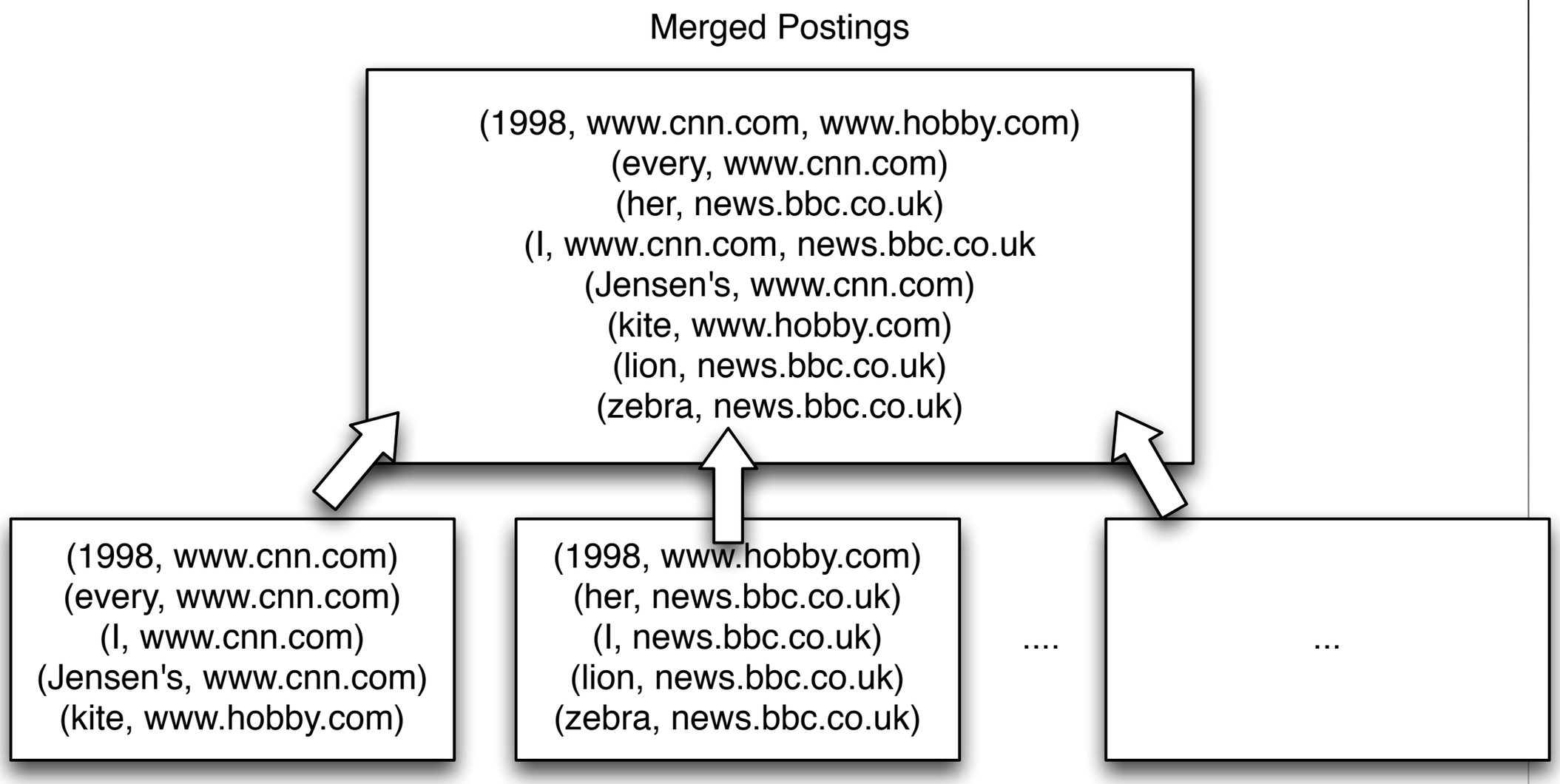
BSBI - Block sort-based indexing

Different way to sort index

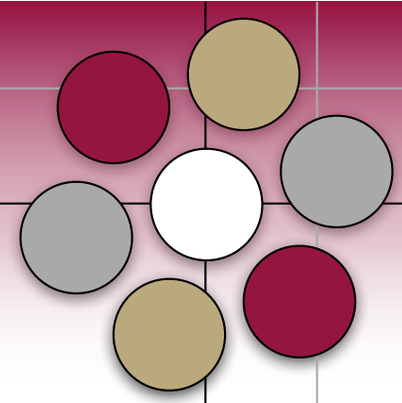


BSBI - Block sort-based indexing

Different way to sort index



BSBI - Block sort-based indexing



BLOCKSORTBASEDINDEXCONSTRUCTION()

```
1  n ← 0
2  while (all documents not processed)
3      do block ← PARSENEXTBLOCK()
4         BSBI-INVERT(block)
5         WRITEBLOCKTODISK(block, f_n)
6  MERGEBLOCKS(f_1, f_2..., f_n, f_merged)
```

BSBI - Block sort-based indexing

Block merge indexing

- Parse documents into (TermID, DocID) pairs until “block” is full
- Invert the block
  - Sort the (TermID,DocID) pairs
  - Compile into TermID posting lists
- Write the block to disk
- Then merge all blocks into one large postings file
  - Need 2 copies of the data on disk (input then output)

BSBI - Block sort-based indexing

Analysis of BSBI

- The dominant term is O(TlogT)
  - T is the number of (TermID,DocID) pairs
- But in practice ParseNextBlock takes the most time
- Then MergingBlocks
- Again, disk seeks times versus memory access times

BSBI - Block sort-based indexing

Analysis of BSBI

- 12-byte records (term, doc, meta-data)
- Need to sort T= 100,000,000 such 12-byte records by term
- Define a block to have 1,600,000 such records
  - can easily fit a couple blocks in memory
  - we will be working with 64 such blocks
- 64 blocks * 1,600,000 records * 12 bytes = 1,228,800,000 bytes
- Nlog2N comparisons is 5,584,577,250.93
- 2 touches per comparison at memory speeds (10e-6 sec) =
  - 55,845.77 seconds = 930.76 min = 15.5 hours

Index Construction

Overview

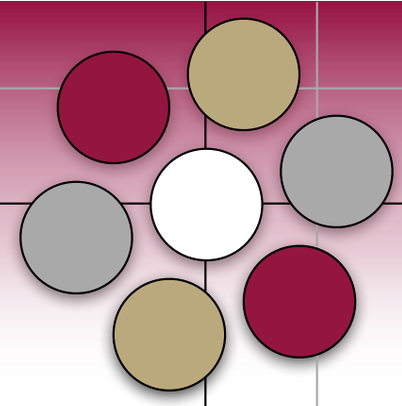
- Introduction
- Hardware
- BSBI - Block sort-based indexing
- SPIMI - Single Pass in-memory indexing
- Distributed indexing
- Dynamic indexing
- Miscellaneous topics

Single-Pass In-Memory Indexing

SPIMI

- BSBI is good but,
  - it needs a data structure for mapping terms to termIDs
  - this won't fit in memory for big corpora
- Straightforward solution
  - dynamically create dictionaries
  - store the dictionaries with the blocks
  - integrate sorting and merging

Single-Pass In-Memory Indexing



```
SPIMI-Invert(tokenStream)
 1  outputFile ← NewFile()
 2  dictionary ← NewHash()
 3  while (free memory available)
 4      do token ← next(tokenStream)
 5          if term(token) ∉ dictionary
 6              then postingsList ← AddToDictionary(dictionary, term(token))
 7              else  postingsList ← GetPostingsList(dictionary, term(token))
 8          if full(postingsList)
 9              then postingsList ← DoublePostingsList(dictionary, term(token))
10          AddToPostingsList(postingsList, docID(token))
11  sortedTerms ← SortTerms(dictionary)
12  WriteBlockToDisk(sortedTerms, dictionary, outputFile)
13  return outputFile
```

Single-Pass In-Memory Indexing

SPIMI

- So what is different here?
  - SPIMI adds postings directly to a posting list.
    - BSBI first collected (TermID,DocID pairs)
      - then sorted them
      - then aggregated the postings
- Each posting list is dynamic so there is no term sorting
- Saves memory because a term is only stored once
- Complexity is O(T)
- Compression enables bigger effective blocks

Single-Pass In-Memory Indexing

Large Scale Indexing

- Key decision in block merge indexing is block size
- In practice, spidering often interlaced with indexing
- Spidering bottlenecked by WAN speed and other factors

Why we use these algorithms

	Cloud	Disk Drive	SSD	RAM	CPU Cache
Storage Size	~ Infinite	~ PB	~ TB	~ GB	~ MB
Access Speed	~ 1s	.005 s	.00001 s	0.00000002 s	40 clock cycle

- Deal with storage size / speed tradeoff

Index Construction

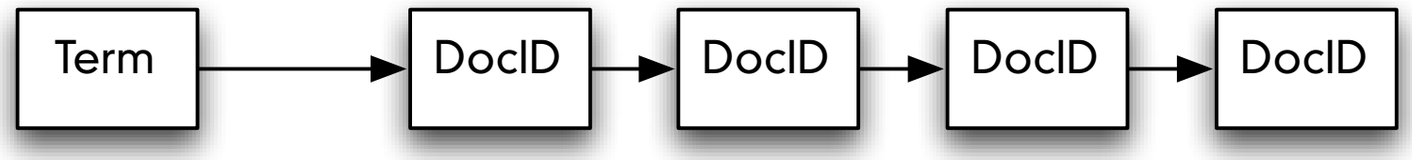
Overview

- Introduction
- Hardware
- BSBI - Block sort-based indexing
- SPIMI - Single Pass in-memory indexing
- Distributed indexing
- Dynamic indexing
- Miscellaneous topics

Index Construction

Review

- termID is an index given to a vocabulary word
  - e.g., “house” = 57820
- docID is an index given to a document
  - e.g., “news.bbc.co.uk” = 74291
- posting list is a data structure for the term-document matrix



- posting list is an inverted data structure

Index Construction

Review

- BSBI and SPIMI
  - are single pass indexing algorithms
  - leverage fast memory vs slow disk speeds
  - for data sets that won’t fit in entirely in memory
  - for data sets that will fit on a single disk

Index Construction

Review

- BSBI
  - builds (termID, docID) pairs until a block is filled
  - builds a posting list in the final merge
  - requires a vocabulary mapping word to termID
- SPMI
  - builds posting lists until a block is filled
  - combines posting lists in the final merge
  - uses terms directly (not termIDs)

Index Construction

- What if your documents don’t fit on a single disk?
  - Web-scale indexing
    - Use a distributed computing cluster
  - supported by “Cloud computing” companies

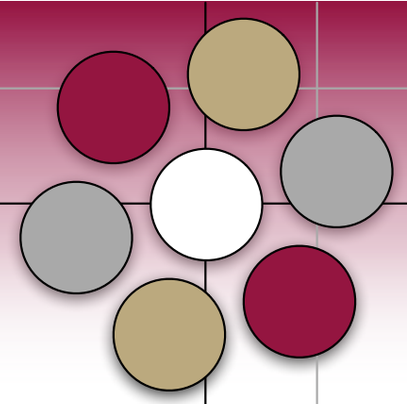
Distributed Indexing

- Other benefits of distributed processing
  - Individual machines are fault-prone
  - They slow down unpredictably or fail
    - Automatic maintenance
    - Software bugs
    - Transient network conditions
    - A truck crashing into the pole outside
    - Hardware fatigue and then failure

Distributed Indexing - Architecture

- The design of Google's indexing as of 2004
  - trajectory of big data since then

Distributed Indexing - Architecture

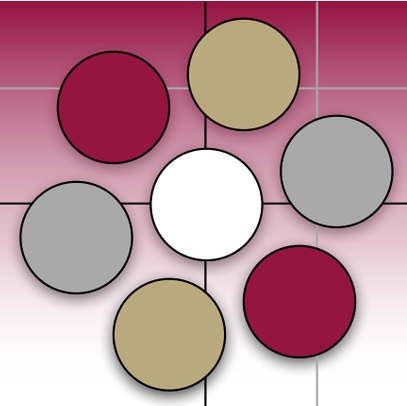


- Think of our task as two types of parallel tasks
  - Parsing
    - A Parser will read a document and output (t,d) pairs
  - Inverting
    - An Inverter will sort and write posting lists

Distributed Indexing - Architecture

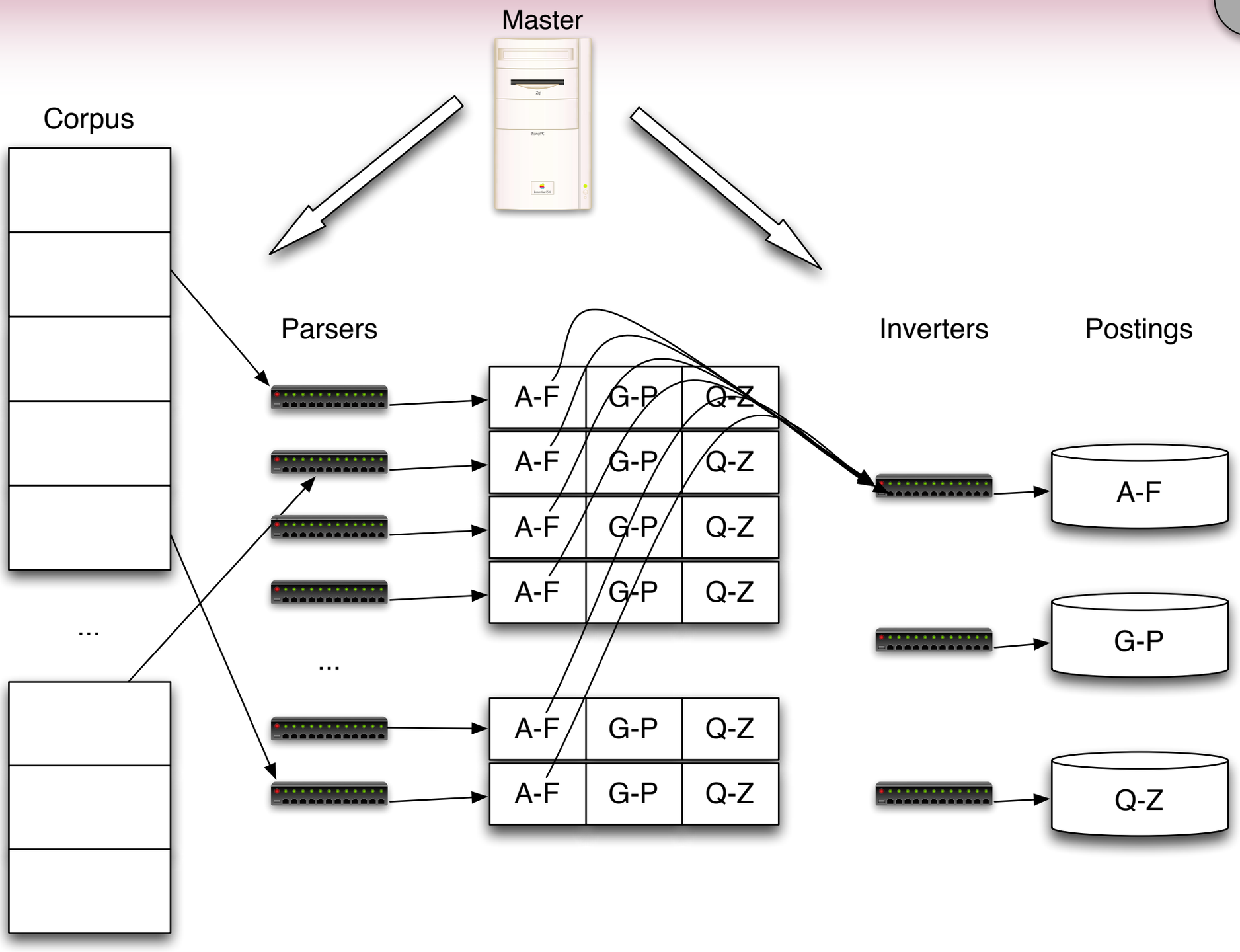
- Use an instance of MapReduce
  - A general architecture for distributed computing jobs
  - Manages interactions among clusters of
    - cheap commodity compute servers
    - aka nodes
- Uses Key-Value pairs as primary object of computation
- An open-source implementation is “Hadoop” by apache.org

Distributed Indexing - Architecture

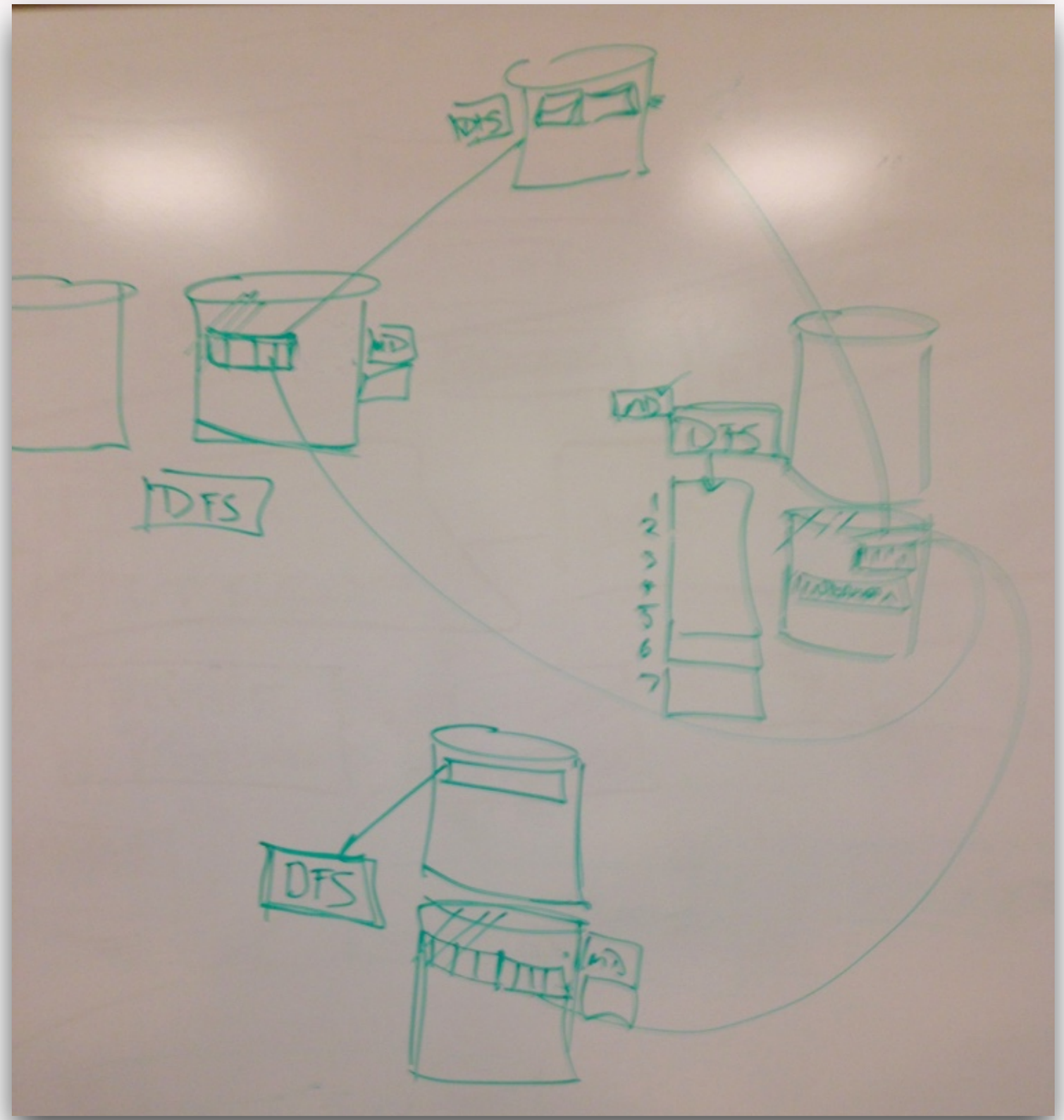


- Generally speaking in MapReduce
  - There is a map phase
    - This takes input and makes key-value pairs
    - this corresponds to the “parse” phase of BSBI and SPIMI
  - The map phase writes intermediate files
    - Results are bucketed into R buckets
  - There is a reduce phase
    - This is the “invert” phase of BSBI and SPIMI
    - There are R inverters

Distributed Indexing - Architecture



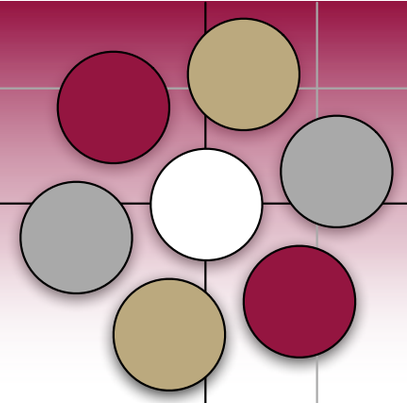
Distributed Indexing - Architecture



Distributed Indexing - Architecture

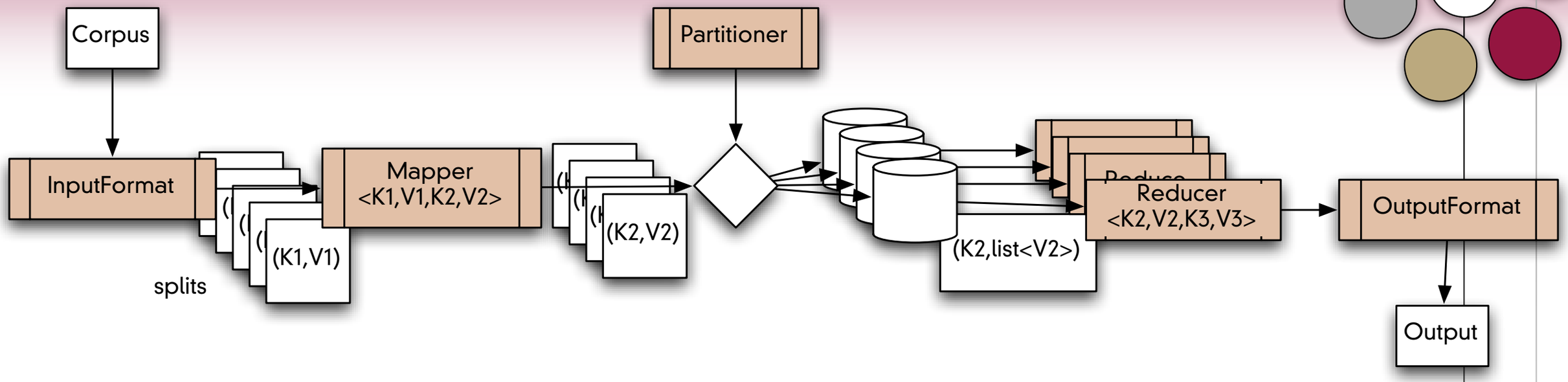
- Parsers and Inverters are not separate machines
  - They are both assigned from a pool
  - It is different code that gets executed
- Intermediate files are stored on a local disk
  - For efficiency
  - Part of the “invert” task is to talk to the parser machine and get the data.

Distributed Indexing - Hadoop



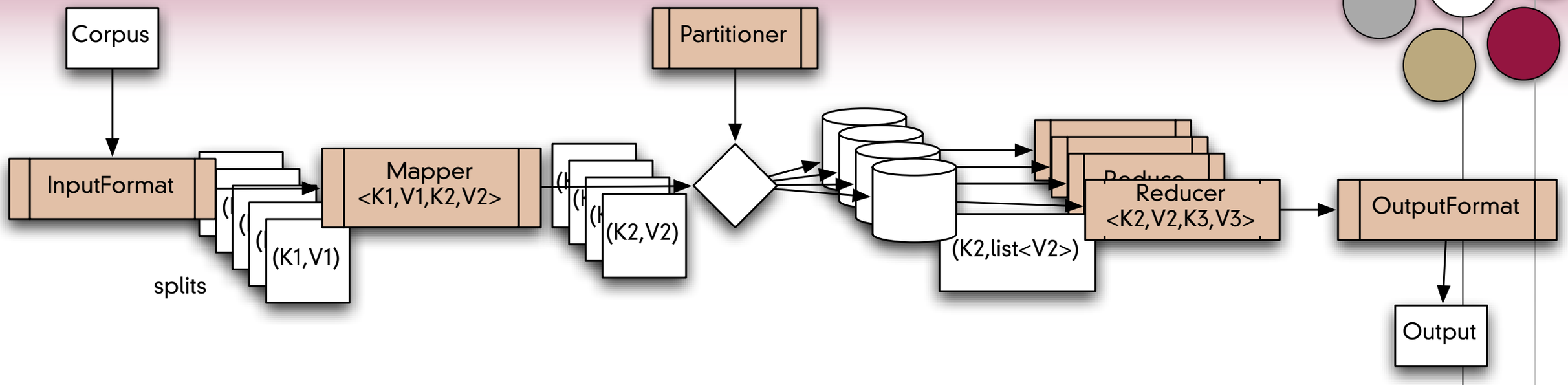
- Hadoop/MapReduce does
  - Hadoop manages fault tolerance
  - Hadoop manages job assignment
  - Hadoop manages a distributed file system
  - Hadoop provides a pipeline for data
- Hadoop/MapReduce does not
  - define data types
  - manipulate data

Distributed Indexing - Hadoop



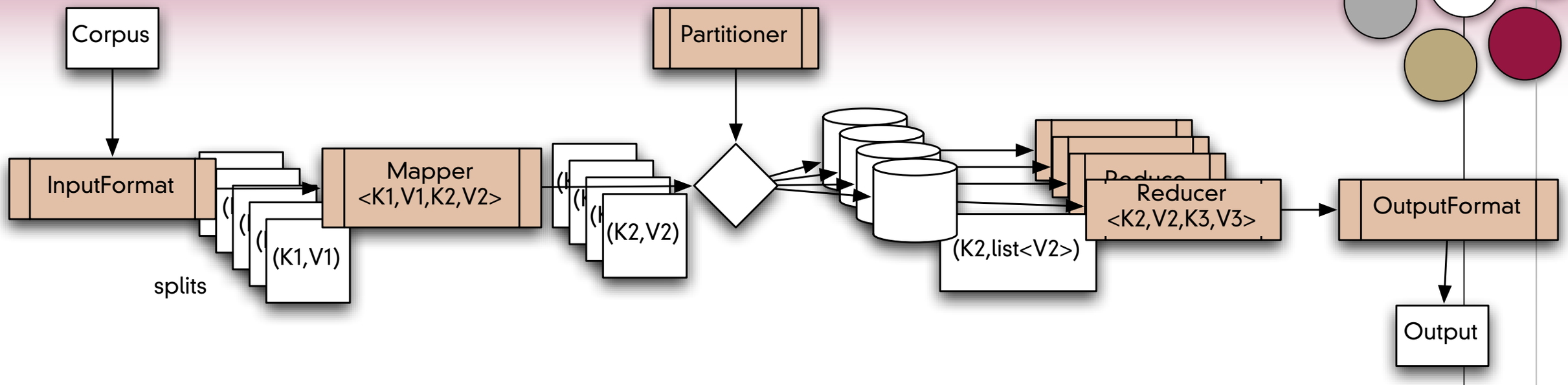
- InputFormat
- Creates splits
- One split is assigned to one mapper
- A split is a collection of <K1,V1> pairs

Distributed Indexing - Hadoop



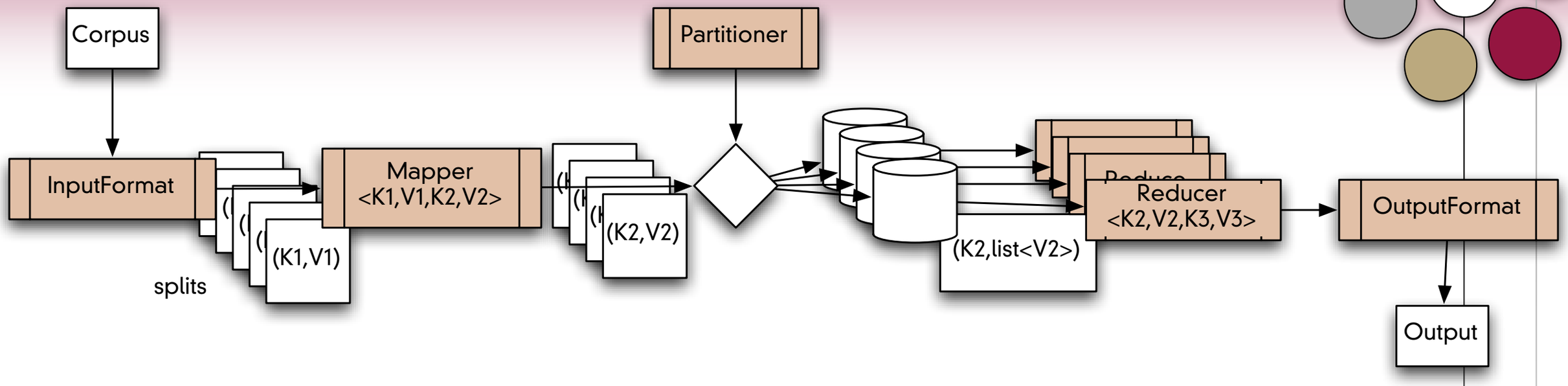
- InputFormat
  - Hadoop comes with NLineInputFormat which breaks text input into splits with N lines each
  - K1 = line number
  - V1 = text of line

Distributed Indexing - Hadoop



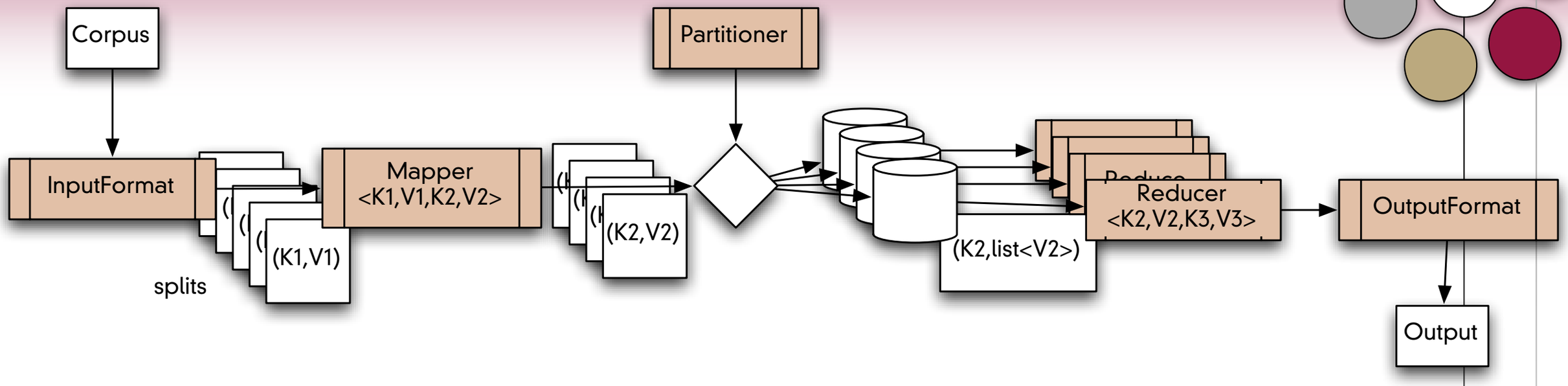
- Mapper<K1,V1,K2,V2>
- Takes a <K1,V1> pair as input
- Produces 0, 1 or more <K2,V2> pairs as output
- Optionally it can report progress with a Reporter

Distributed Indexing - Hadoop



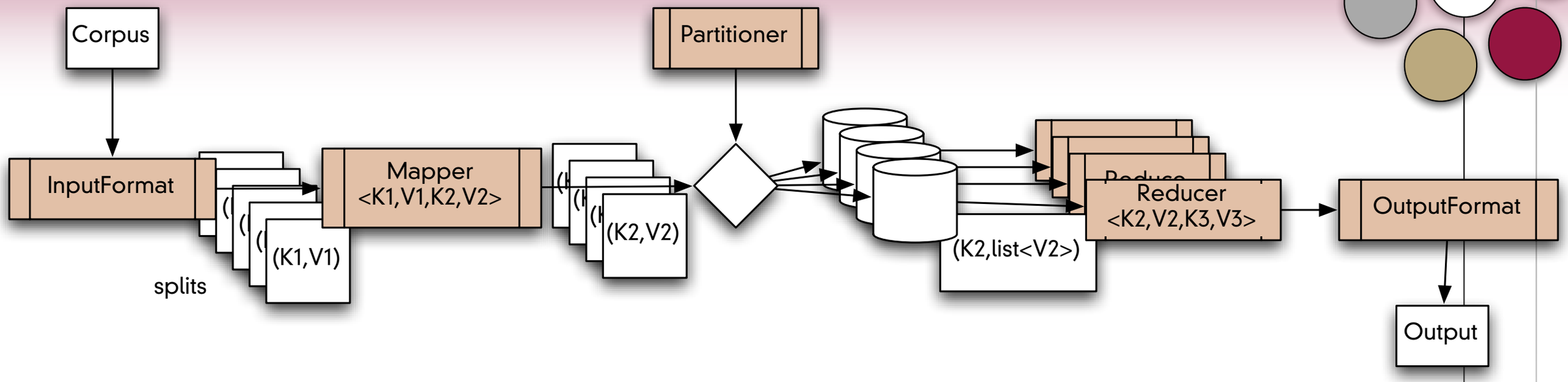
- Partitioner<K2,V2>
- Takes a <K2,V2> pair as input
- Produces a bucket number as output
- Default is HashPartitioner

Distributed Indexing - Hadoop



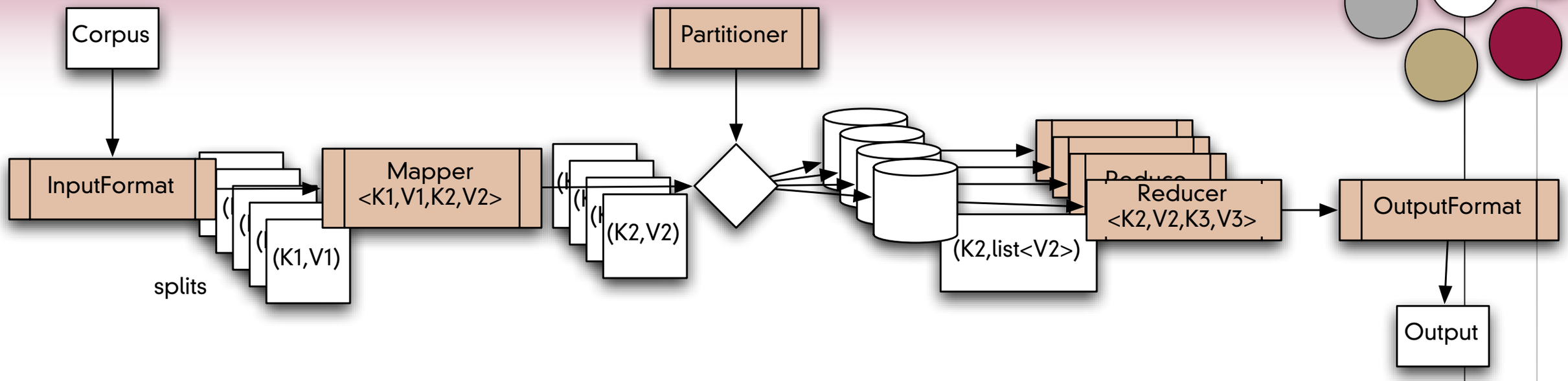
- Reducer<K2,V2,K3,V3>
- Takes a <K2,list<V2>> pair as input
- Produces <K3,V3> as output
- Output is not resorted

Distributed Indexing - Hadoop



- OutputFormat
- Does something with the output (like write it to disk)
- TextOutputFormat<K3,V3> comes with Hadoop

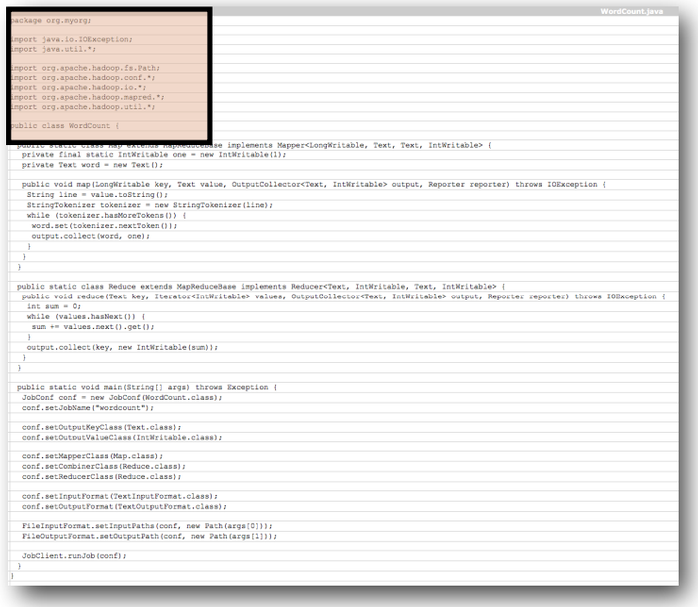
Hadoop example: WordCount



- Example: count the words in an input corpus

- InputFormat = TextInputFormat
- Mapper: separates words, outputs <Word, 1>
- Partitioner = HashPartitioner
- Reducer: counts the length of list<V2>, outputs <Word,count>
- OutputFormat = TextOutputFormat

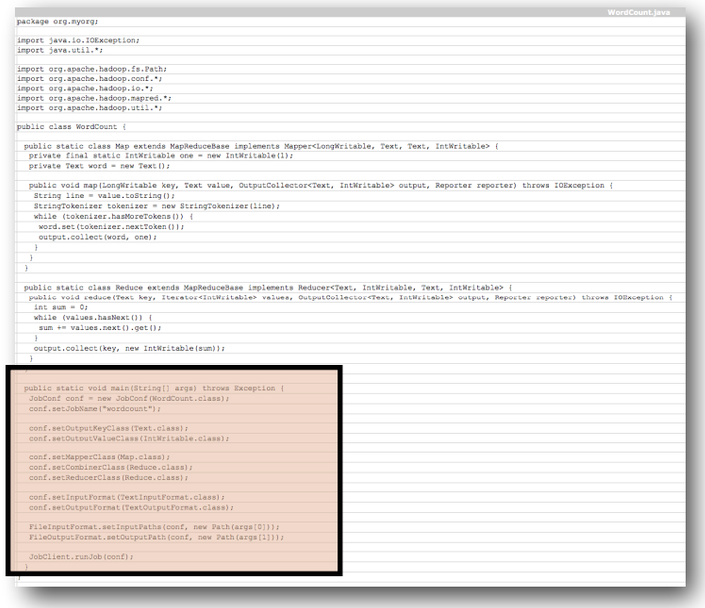
Hadoop example: WordCount


WordCount.java

```
package org.myorg;

import java.io.IOException;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.util.*;

public class WordCount {

 public static class Map extends MapReduceBase implements Mapper<LongWritable, Text, Text, IntWritable> {
  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();

  public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {
   String line = value.toString();
   StringTokenizer tokenizer = new StringTokenizer(line);
   while (tokenizer.hasMoreTokens()) {
    word.set(tokenizer.nextToken());
    output.collect(word, one);
   }
  }
 }

 public static class Reduce extends MapReduceBase implements Reducer<Text, IntWritable, Text, IntWritable> {
  public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {
   int sum = 0;
   while (values.hasNext()) {
    sum += values.next().get();
   }
   output.collect(key, new IntWritable(sum));
  }
 }

 public static void main(String[] args) throws Exception {
  JobConf conf = new JobConf(WordCount.class);
  conf.setJobName("wordcount");

  conf.setOutputKeyClass(Text.class);
  conf.setOutputValueClass(IntWritable.class);

  conf.setMapperClass(Map.class);
  conf.setCombinerClass(Reduce.class);
  conf.setReducerClass(Reduce.class);

  conf.setInputFormat(TextInputFormat.class);
  conf.setOutputFormat(TextOutputFormat.class);

  FileInputFormat.setInputPaths(conf, new Path(args[0]));
  FileOutputFormat.setOutputPath(conf, new Path(args[1]));

  JobClient.runJob(conf);
 }
}
```

```
package org.myorg;
import java.io.IOException;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.util.*;


public class WordCount {
```

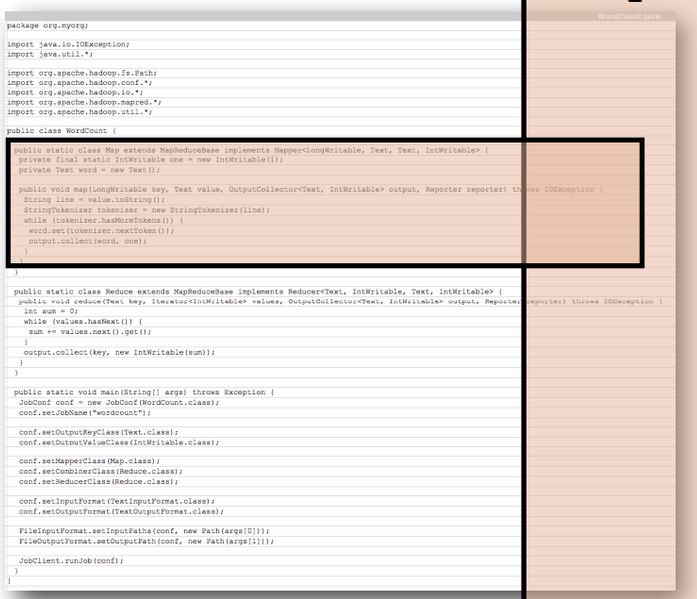
Hadoop example: WordCount


WordCount.java

```
package org.myorg;

import java.io.IOException;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.util.*;

public class WordCount {

 public static class Map extends MapReduceBase implements Mapper<LongWritable, Text, Text, IntWritable> {
   private final static IntWritable one = new IntWritable(1);
   private Text word = new Text();

   public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {
     String line = value.toString();
     StringTokenizer tokenizer = new StringTokenizer(line);
     while (tokenizer.hasMoreTokens()) {
       word.set(tokenizer.nextToken());
       output.collect(word, one);
     }
   }
 }

 public static class Reduce extends MapReduceBase implements Reducer<Text, IntWritable, Text, IntWritable> {
   public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {
     int sum = 0;
     while (values.hasNext()) {
       sum += values.next().get();
     }
     output.collect(key, new IntWritable(sum));
   }
 }

 public static void main(String[] args) throws Exception {
   JobConf conf = new JobConf(WordCount.class);
   conf.setJobName("wordcount");

   conf.setOutputKeyClass(Text.class);
   conf.setOutputValueClass(IntWritable.class);

   conf.setMapperClass(Map.class);
   conf.setCombinerClass(Reduce.class);
   conf.setReducerClass(Reduce.class);

   conf.setInputFormat(TextInputFormat.class);
   conf.setOutputFormat(TextOutputFormat.class);

   FileInputFormat.setInputPaths(conf, new Path(args[0]));
   FileOutputFormat.setOutputPath(conf, new Path(args[1]));

   JobClient.runJob(conf);
 }
}
```

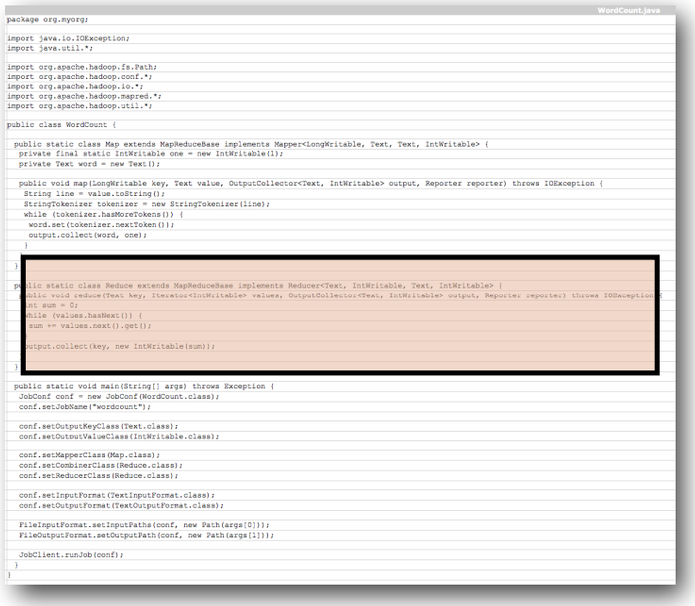
```
public static void main(String[] args) throws Exception {
    JobConf conf = new JobConf(WordCount.class);
    conf.setJobName("wordcount");

    conf.setInputFormat(TextInputFormat.class);
    FileInputFormat.setInputPaths(conf, new Path(args[0]));

    conf.setMapperClass(Map.class);
    conf.setReducerClass(Reduce.class);

    conf.setOutputKeyClass(Text.class);
    conf.setOutputValueClass(IntWritable.class);

    conf.setOutputFormat(TextOutputFormat.class);
    FileOutputFormat.setOutputPath(conf, new Path(args[1]));

    JobClient.runJob(conf);
}
```

Hadoop example: WordCount




```
public static class Map extends MapReduceBase implements
        Mapper<LongWritable, Text, Text, IntWritable> {

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();
    public void map(LongWritable key, Text value,
        OutputCollector<Text, IntWritable> output, Reporter reporter)
        throws IOException {

        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()) {
            word.set(tokenizer.nextToken());
            output.collect(word, one);
        }
    }
}
```

Hadoop example: WordCount

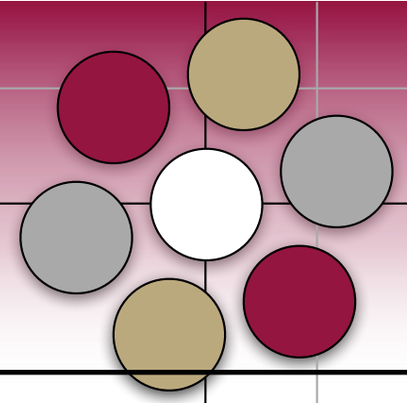



```
public static class Reduce extends MapReduceBase implements
Reducer<Text, IntWritable, Text, IntWritable> {

    public void reduce(Text key, Iterator<IntWritable> values,
    OutputCollector<Text, IntWritable> output, Reporter
    reporter) throws IOException {

        int sum = 0;
        while (values.hasNext()) {
            sum += values.next().get();
        }
        output.collect(key, new IntWritable(sum));
    }
}
```

Hadoop execution

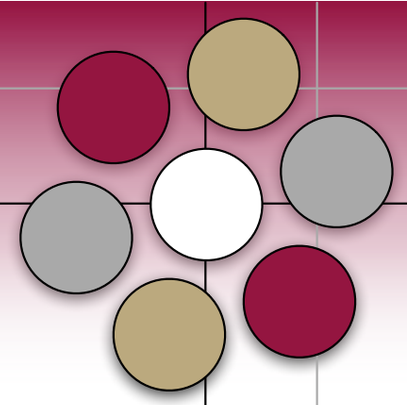
- The JobTracker
  - manages assignment of code to nodes
  - makes sure nodes are responding
  - makes sure data is flowing
  - the “master”
- The TaskTracker
  - runs on nodes
  - accepts jobs from the JobTracker
  - one of many “slaves”

Hadoop example: WordCount



```
> mkdir wordcount_classes
> javac -classpath ${HADOOP_HOME}/hadoop-${HADOOP_VERSION}-core.jar -d wordcount_classes
WordCount.java

>jar -cvf /usr/joe/wordcount.jar -C wordcount_classes/ .

> bin/hadoop dfs -ls /usr/joe/wordcount/input/
/usr/joe/wordcount/input/file01
/usr/joe/wordcount/input/file02

>bin/hadoop dfs -cat /usr/joe/wordcount/input/file01
Hello World Bye World

>bin/hadoop dfs -cat /usr/joe/wordcount/input/file02
Hello Hadoop Goodbye Hadoop
```

Hadoop example: WordCount



```
>bin/hadoop jar /usr/joe/wordcount.jar org.myorg.WordCount /usr/joe/wordcount/input /usr/
joe/wordcount/output

>bin/hadoop dfs -cat /usr/joe/wordcount/output/part-00000
Bye 1
Goodbye 1
Hadoop 2
Hello 2
World 2
```

Hadoop example: WordCount

Hadoop job_200709211549_0003 on localhost

User: hadoop
Job Name: streamjob34453.jar
Job File: /usr/local/hadoop-datastore/hadoop-hadoop/mapred/system/job_200709211549_0003/job.xml
Status: Succeeded
Started at : Fri Sep 21 16:07:10 CEST 2007
Finished at: Fri Sep 21 16:07:26 CEST 2007
Finished in: 16sec

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	3	0	0	3	0	0 / 0
reduce	100.00%	1	0	0	1	0	0 / 0

	Counter	Map	Reduce	Total
Job Counters	Launched map tasks	0	0	3
	Launched reduce tasks	0	0	1
	Data-local map tasks	0	0	3
Map-Reduce Framework	Map input records	77,637	0	77,637
	Map output records	103,909	0	103,909
	Map input bytes	3,659,910	0	3,659,910
	Map output bytes	1,083,767	0	1,083,767
	Reduce input groups	0	85,095	85,095
	Reduce input records	0	103,909	103,909
	Reduce output records	0	85,095	85,095

Change priority from NORMAL to: VERY_HIGH HIGH LOW VERY_LOW

Index Construction

Overview

- Introduction
- Hardware
- BSBI - Block sort-based indexing
- SPIMI - Single Pass in-memory indexing
- Distributed indexing
- Dynamic indexing
- Miscellaneous topics

Dynamic Indexing

- Documents come in over time
  - Postings need to be updated for terms already in dictionary
  - New terms need to get added to dictionary
- Documents go away
  - Get deleted, etc.

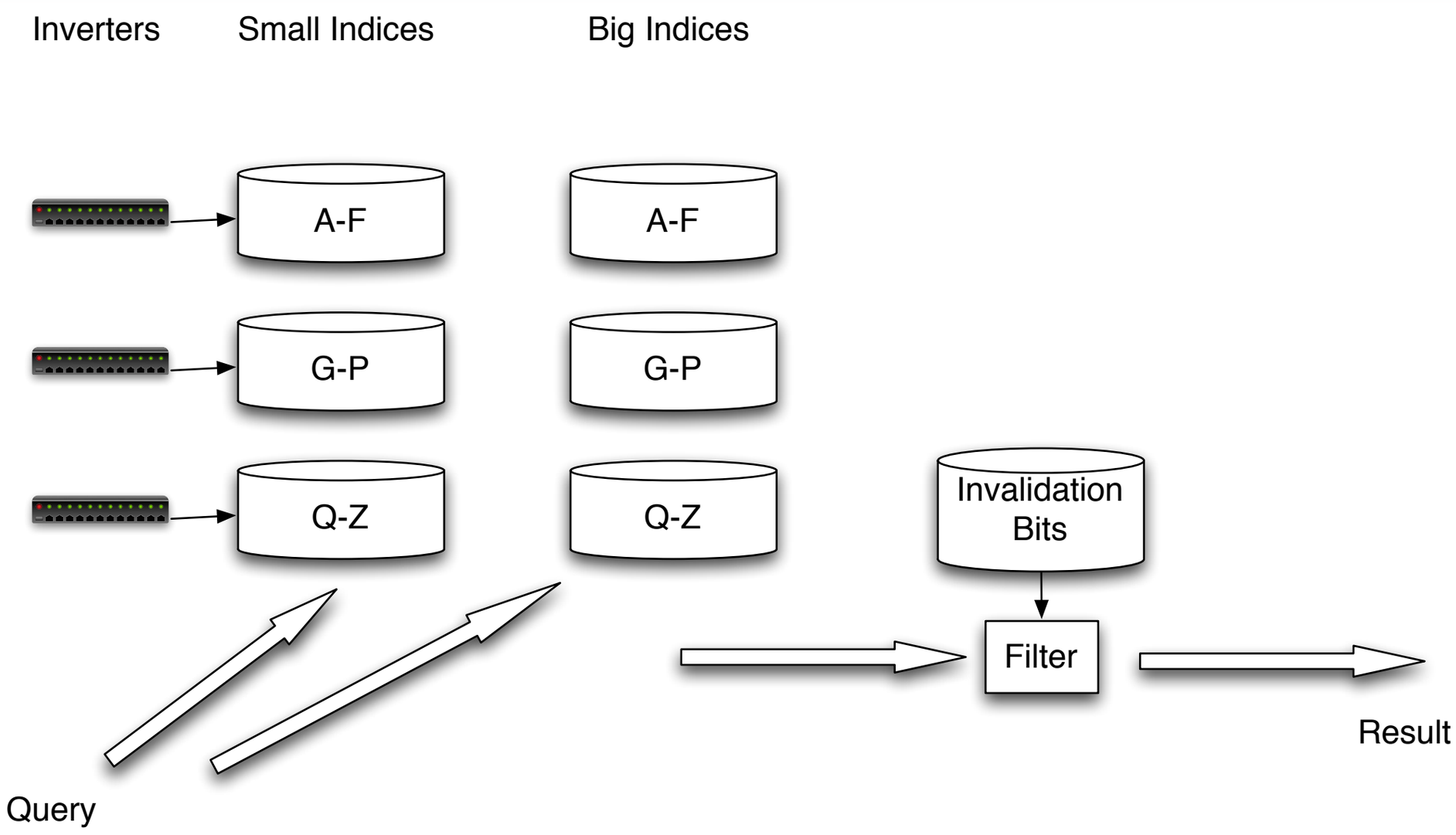
Dynamic Indexing

- Overview of solution
  - Maintain your “big” main index on disk
    - (or distributed disk)
  - Continuous crawling creates “small” indices in memory
  - Search queries are applied to both
    - Results merged

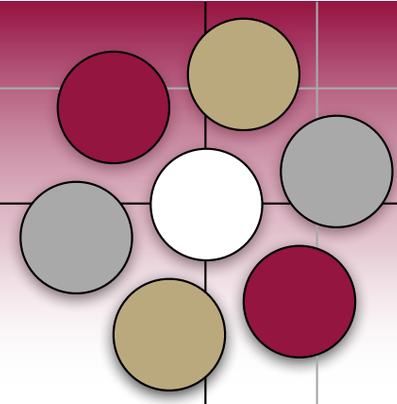
Dynamic Indexing

- Overview of solution
  - Document deletions
    - Invalidation bit for deleted documents
    - Just like contextual filtering,
      - results are filtered to remove invalidated docs
      - according to bit vector.
- Periodically merge “small” index into “big” index.

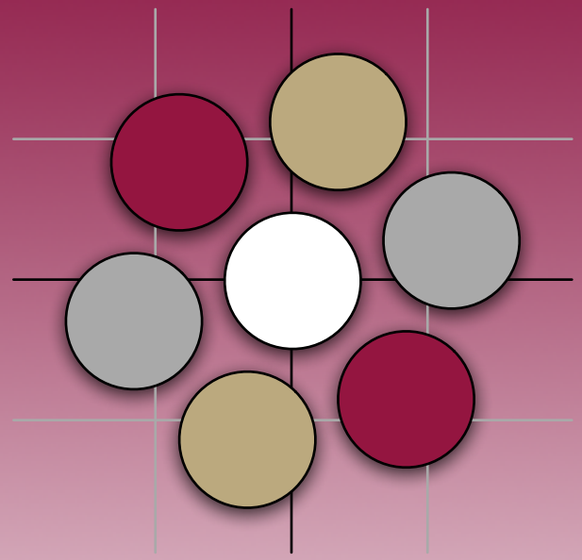
Dynamic Indexing



Dynamic Indexing



- Issues with big *and* small indexes
  - Corpus wide statistics are hard to maintain
    - Typical solution is to ignore small indices when computing stats
  - Frequent merges required
  - Poor performance during merge
    - unless well engineered
  - Logarithmic merging



WESTMONT COMPUTER SCIENCE