Index Construction

Introduction to Information Retrieval
INF 141/ CS 121
Donald J. Patterson

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

Overview

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

Why we use these algorithms

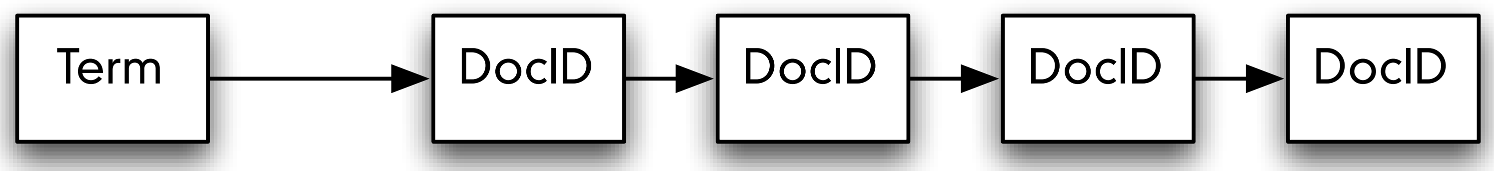
	Cloud	Disk Drive	SSD	RAM	CPU Cache
Storage Size	~ Infinite	~ PB	~ TB	~ GB	~ MB
Access Speed	~ 1s	.005 s	.00001 s	0.00000002 s	40 clock cycle

- Deal with storage size / speed tradeoff

Index Construction

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/index.html” = 74291
- posting list is a data structure for the term-document matrix



- posting list is an inverted data structure

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

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

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

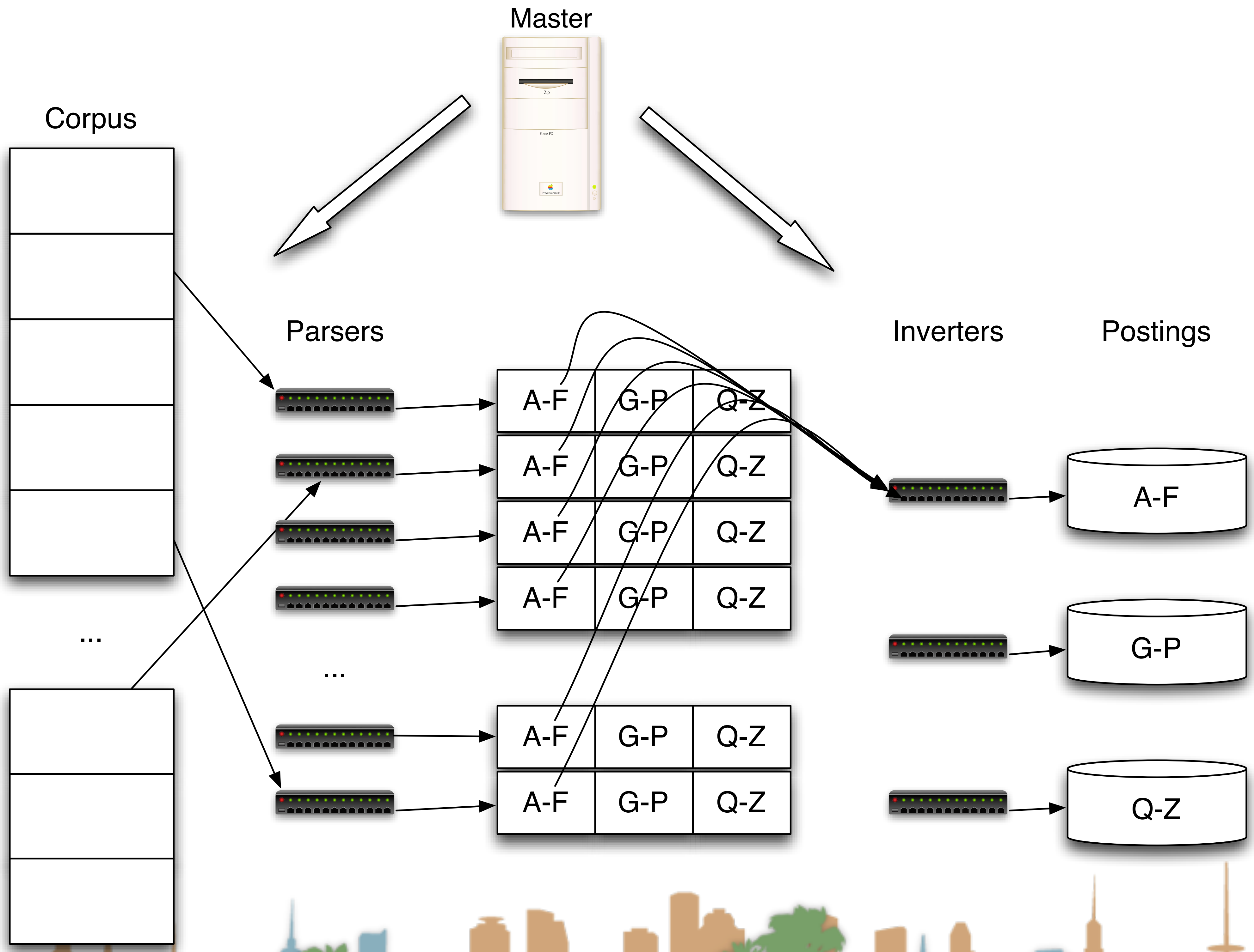
Distributed Indexing - Architecture

- 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



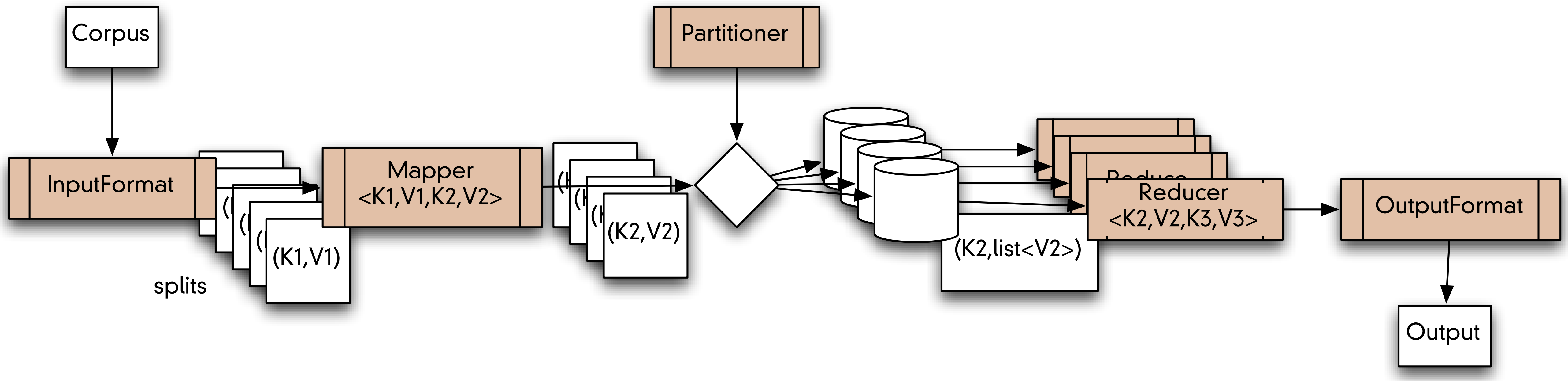
Distributed Indexing - Architecture

- 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

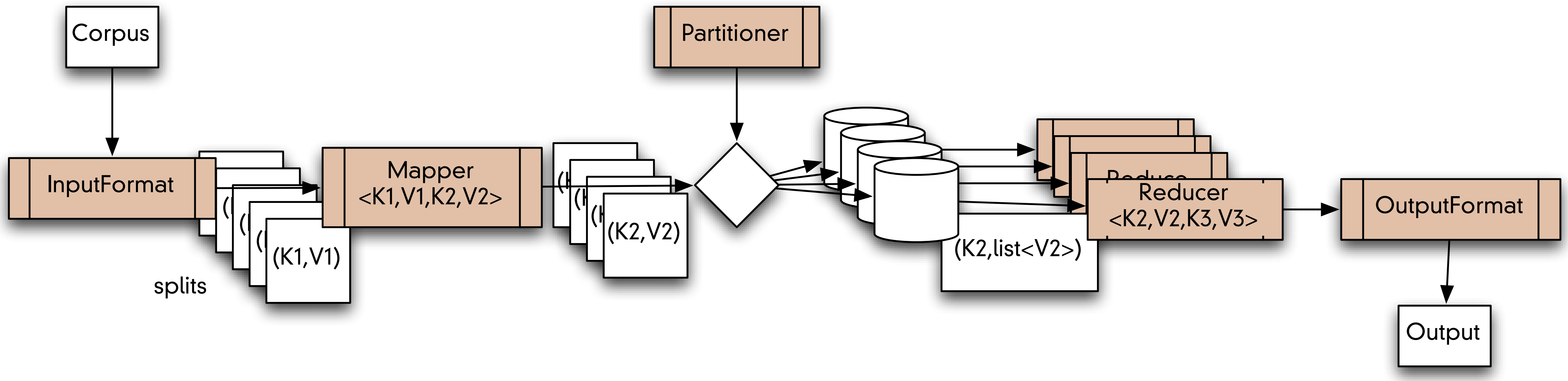
- Hadoop/MapReduce does
  - Hadoop manages fault tolerance
  - Hadoop manages job assignment
  - Hadoop manages a distributed file system
  - Hadoop provides a pipeline for data
- Hadoop/MapReduce does not
  - define data types
  - manipulate data

Distributed Indexing - Hadoop



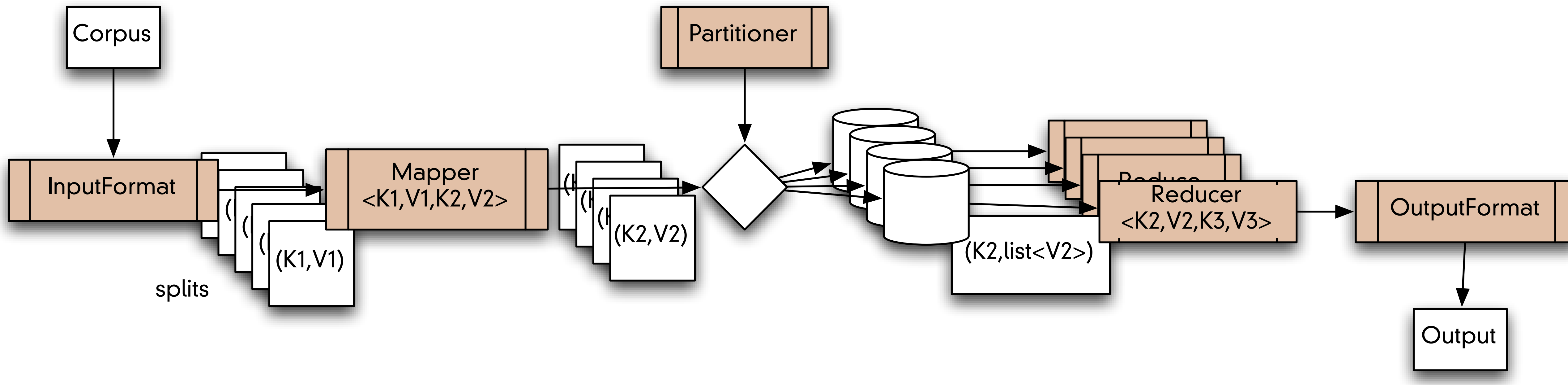
- InputFormat
  - Creates splits
  - One split is assigned to one mapper
  - A split is a collection of <K1,V1> pairs

Distributed Indexing - Hadoop



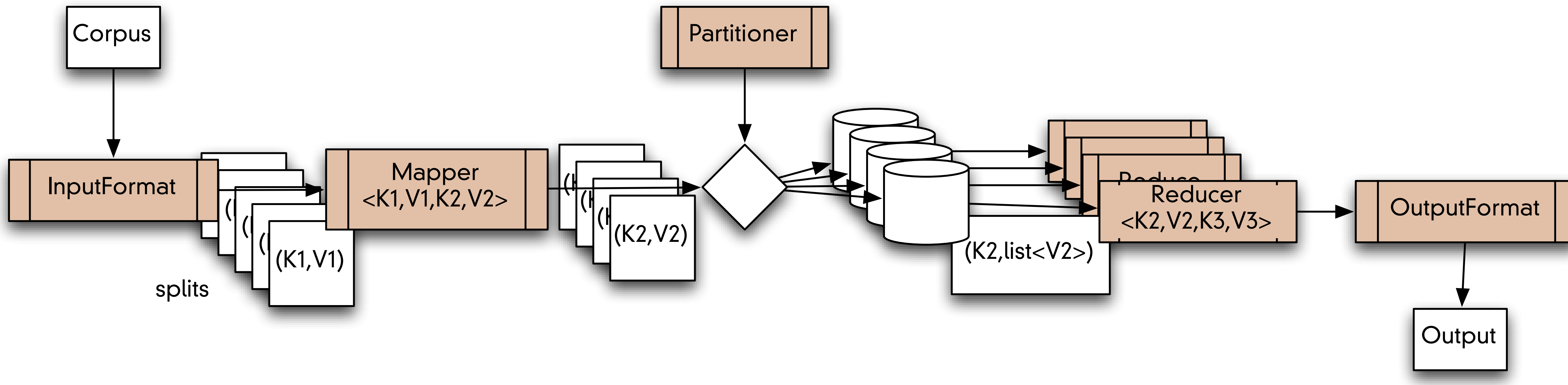
- InputFormat
  - Hadoop comes with NLineInputFormat which breaks text input into splits with N lines each
  - K1 = line number
  - V1 = text of line

Distributed Indexing - Hadoop



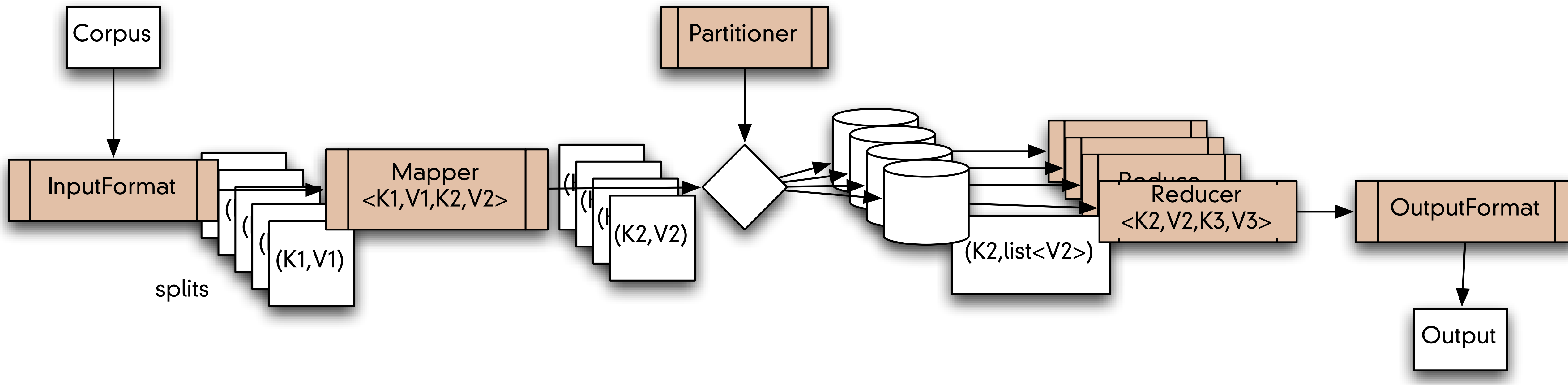
- Mapper<K1,V1,K2,V2>
  - Takes a <K1,V1> pair as input
  - Produces 0, 1 or more <K2,V2> pairs as output
  - Optionally it can report progress with a Reporter

Distributed Indexing - Hadoop



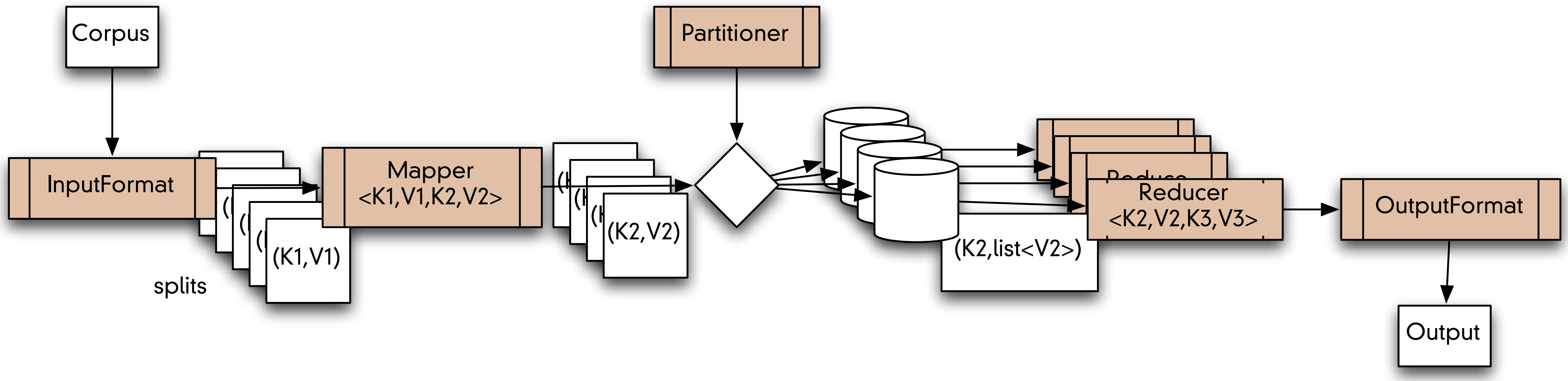
- Partitioner<K2,V2>
  - Takes a <K2,V2> pair as input
  - Produces a bucket number as output
  - Default is HashPartitioner

Distributed Indexing - Hadoop



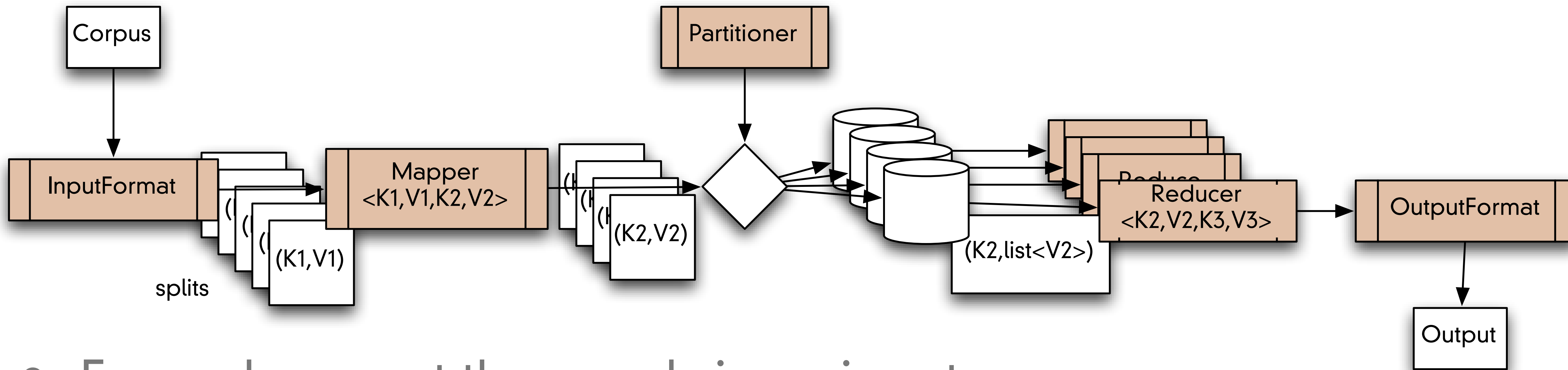
- Reducer<K2,V2,K3,V3>
  - Takes a <K2,list<V2>> pair as input
  - Produces <K3,V3> as output
  - Output is not resorted

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

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.

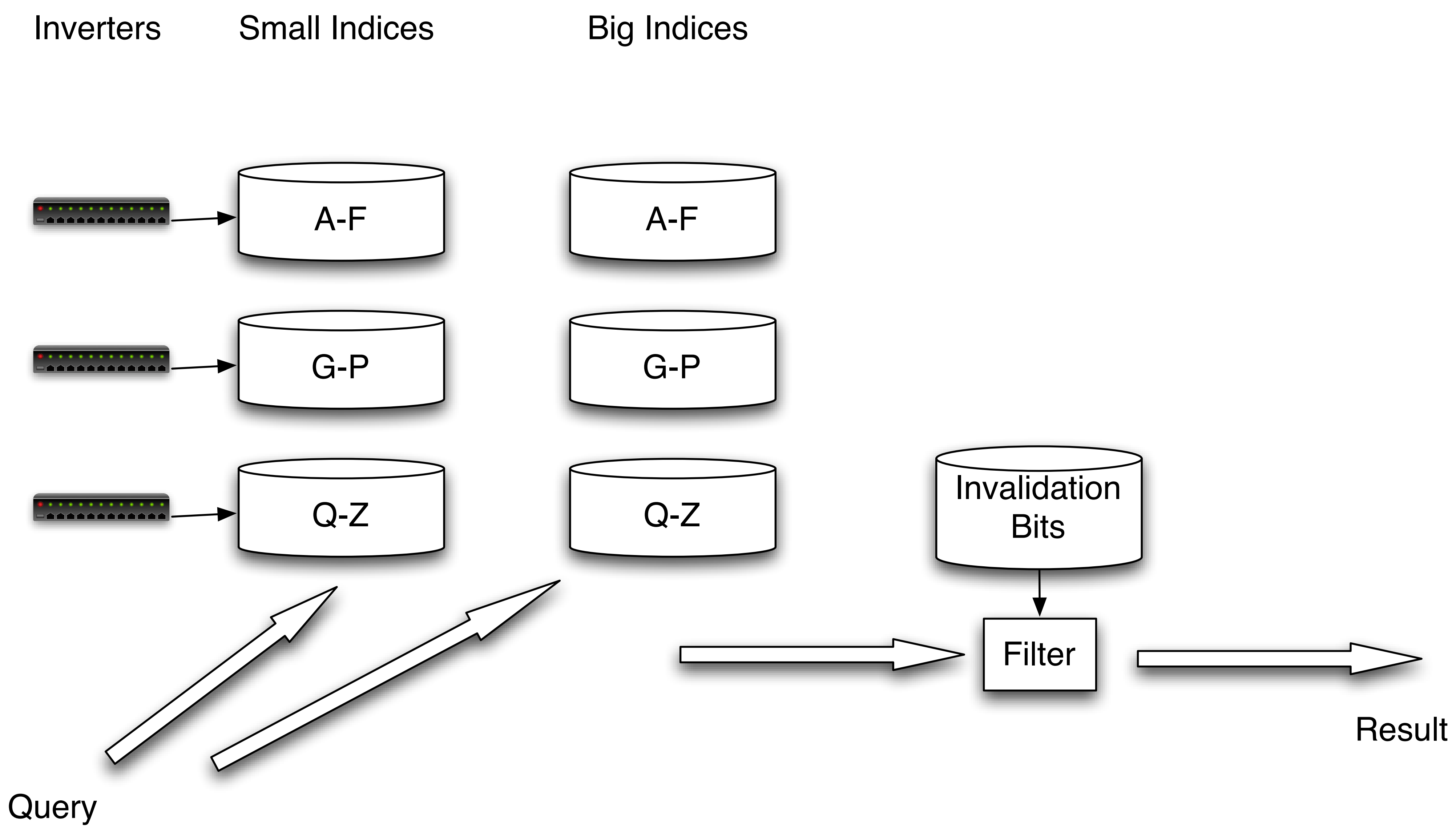
Dynamic Indexing

- 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

End of Chapter 4

LUCI