Informatics 225
Computer Science 221

Information Retrieval

Lecture 18

These course materials borrow, with permission, from those of Prof. Cristina Videira Lopes, Addison Wesley 2008, Chris Manning, Pandu Nayak, Hinrich Schütze, Heike Adel, Sascha Rothe, Jerome H. Friedman, Robert Tibshirani, and Trevor Hastie. Powerpoint theme by Prof. André van der Hoek.

ON ASSIGNMENT 3

Project 3: Indexer

- Milestone #1: Implement a simple Indexer
  - **Start with a small set of files**. Short development cycle!
    - Traversing folders and reading JSON
    - Opening and reading one file at a time
    - Parsing (dealing with broken HTML!)
    - Tokenization & stemming
    - Simple in-memory inverted index
    - Simple index serialization to disk
  - Expand gradually to as much of the dataset as possible, until you hit memory limits
  - No need to scale up yet...

Project 3: Boolean search

- Milestone #2: Implement simple Boolean retrieval – AND only
  - E.g.
    - cristina lopes means AND
    - eppstein Wikipedia means AND
    - master of software engineering means AND
  - Required: text interface. startMyEngine
    - **Bonus** points if you implement a web GUI by MS3
  - No speed restrictions yet…

Project 3: Ranked search

- Milestone #3: Implement ranked retrieval
  - Scale up
  - Completely different approach, will be covered soon
  - May benefit from positional information; come back to these lectures
  - **Must perform under 300ms,** ideally ~100ms but no penalty for anything < 300ms. So come back to these lectures…
    - **Make sure to add a timer in your code.** You need to measure how long it takes between the time it receives the user query and the time it returns the result to the user *(no need to take into account the display time).*

INDEX ENGINEERING ISSUES

Small index

- Load it entirely into memory as a hash table / dictionary
- Find the postings by key on the query terms

Small index

- Load it entirely into memory as a hash table / dictionary
- Find the postings by key on the query terms

What if the index is larger than available memory?

Large index

- Does not fit in memory

Large index

- Does not fit in memory
- Must search for postings **in the file itself**
  - How?

Large index

- Does not fit in memory
- Must search for postings **in the file itself**
  - How?
  - 1) Scan the file looking for the query terms

Large index

- Does not fit in memory
- Must search for postings **in the file itself**
  - How?
  - 1) Scan the file looking for the query terms

What happens if terms are not sorted?

Large index

- Does not fit in memory
- Must search for postings **in the file itself**
  - How?
  - 1) Scan the file looking for the query terms: from O(n) to O(logN)
    - Remember: sorting the query terms helps!

Large index

- Does not fit in memory
- Must search for postings **in the file itself**
  - How?
  - 1) Scan the file looking for the query terms: from O(n) to O(logN)
    - Remember: sorting the query terms helps!
  - 2) Jump to right positions in the file: O(1)

Large index

- Does not fit in memory
- Must search for postings **in the file itself**
  - How?
  - 1) Scan the file looking for the query terms: from O(n) to O(logN)
    - Remember: sorting the query terms helps!
  - 2) Jump to right positions in the file: O(1)
    - **INDEX THE INDEX!**

Large index: index the index

- As you construct the inverted index, create another bookkeeping file with offsets into the inverted index file
  - term “ape” is at position 1311, “apple” is at position 1345, etc. OR
  - words starting with ‘b’ start at position 10035, words starting with ‘c’ start at position 20457, etc.

Large index: index the index

- As you construct the inverted index, create another bookkeeping file with offsets into the inverted index file
  - term “ape” is at position 1311, “apple” is at position 1345, etc. OR
  - words starting with ‘b’ start at position 10035, words starting with ‘c’ start at position 20457, etc.

- This secondary index will be much smaler than the inverted index, so it will (usually) fit in memory

Large index: index the index

- As you construct the inverted index, create another bookkeeping file with offsets into the inverted index file
  - term “ape” is at position 1311, “apple” is at position 1345, etc. OR
  - words starting with ‘b’ start at position 10035, words starting with ‘c’ start at position 20457, etc.

- This secondary index will be much smaler than the inverted index, so it will (usually) fit in memory

- Then: seek(position)

Large index: jump to words

- How?
  - **Seek** operations

Test.txt:

```
Hello world!
This is a test file.
For explaining how seek works.
```

Large index: jump to words

- How?
  - **Seek** operations

```
>>> f = open(”Test.txt")
>>> f.seek(4)
4
```

Test.txt:

```
Hello world!
This is a test file.
For explaining how seek works.
```

Large index: jump to words

- How?
  - **Seek** operations

Test.txt:

```
Hello world!
This is a test file.
For explaining how seek works.
```

```
>>> f = open("test.txt")
>>> f.seek(4)
4
>>> f.readline()
'o world!\n'
```

Large index: jump to words

- How?
  - **Seek** operations

Test.txt:

```
Hello world!
This is a test file.
For explaining how seek works.
```

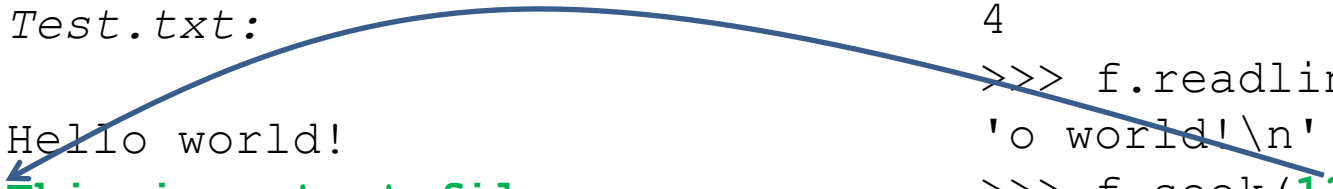
```
>>> f = open("test.txt")
>>> f.seek(4)
4
>>> f.readline()
'o world!\n'
>>> f.seek(13)
13
>>> f.readline()
'This is a test file.\n'
>>>
```

Large index: jump to words

- How?
  - **Seek** operations

Test.txt:

```
Hello world!
This is a test file.
For explaining how seek works.
```



```
>>> f = open("test.txt")
>>> f.seek(4)
4
>>> f.readline()
'o world!\n'
>>> f.seek(13)
13
>>> f.readline()
'This is a test file.\n'
>>>
```

Do you really need this in project 3?

- Maybe! (mostly yes… ☺)
- Remember the maximum budget of 300ms for query processing in MS3

Do you really need this in project 3?

- Maybe!
- Remember the maximum budget of 300ms for query processing in MS3
- And remember that you cannot load your entire index object into memory…

Other Auxiliary Structures

- *Document IDs*
  - Contains lookup table from docid to URL
  - Either hash table in memory or B-tree for larger collections
- *Vocabulary* or *lexicon*
  - Contains a lookup table from index terms to the byte offset of the inverted list in the inverted file
  - Either hash table in memory or B-tree for larger vocabularies
- Term statistics stored at start of inverted lists
- Collection statistics stored in separate file

Alternatives to single inverted index

- Split the inverted index file into alphabet ranges
- Create file system path trees
  index/a/
  index/b/
  index/c/
  etc.
- The search component needs to keep files open all the time
  - **Note: the files are open, but not in memory!**

Multiple files in multithreading: aggregate!

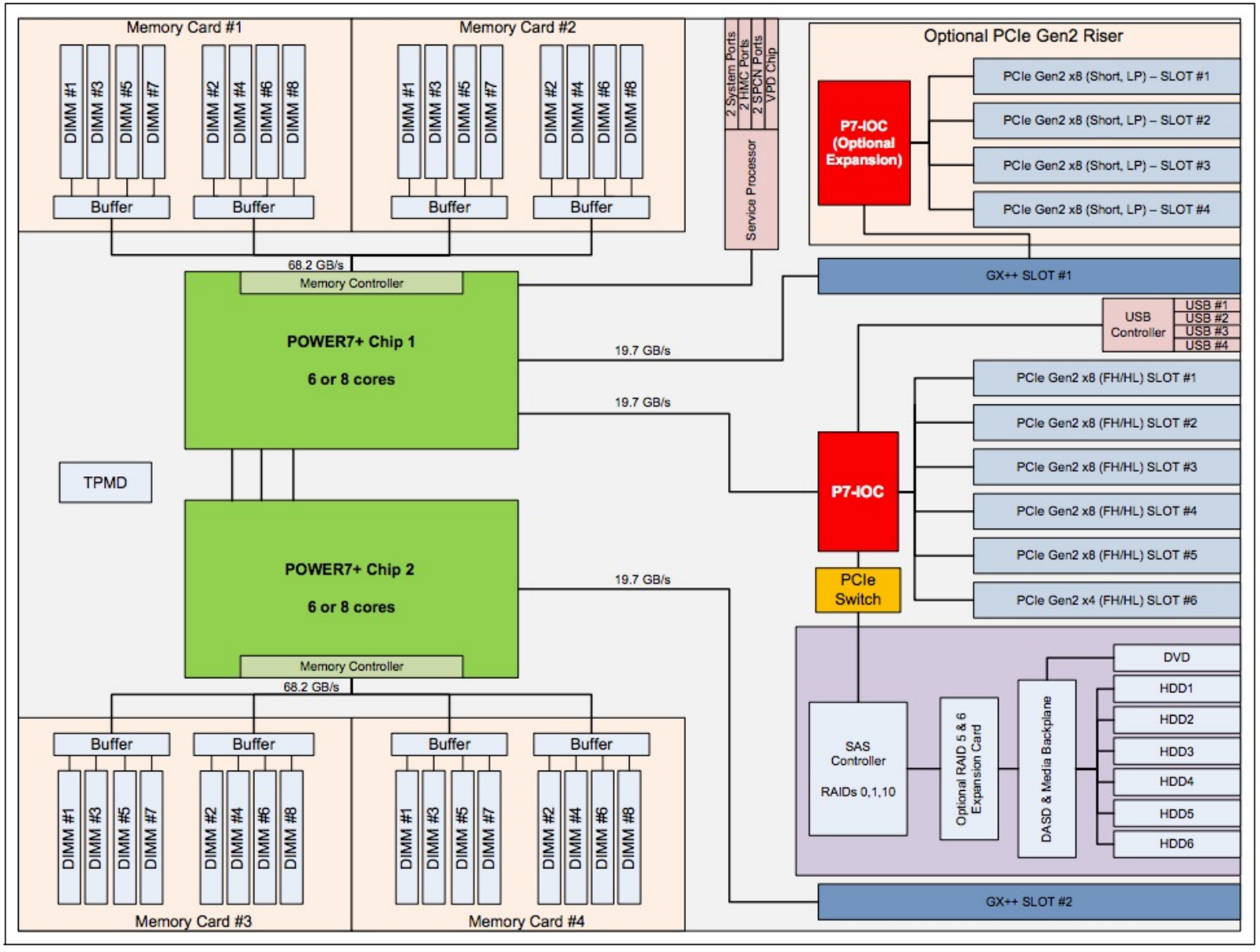


Figure 2-2 IBM Power 740 logical system diagram

SCALING IT UP

Distributed Indexing

- Distributed processing driven by need to index and analyze huge amounts of data (i.e., the Web)
- Large numbers of inexpensive servers used rather than larger, more expensive machines

Distributed Indexing

- Distributed processing driven by need to index and analyze huge amounts of data (i.e., the Web)
- Large numbers of inexpensive servers used rather than larger, more expensive machines
- *MapReduce* is a distributed programming tool designed for indexing and analysis tasks

Example

- Given a large text file that contains data about credit card transactions
  - Each line of the file contains a credit card number and an amount of money
  - Determine the number of unique credit card numbers

Example

- Given a large text file that contains data about credit card transactions
  - Each line of the file contains a credit card number and an amount of money
  - Determine the number of unique credit card numbers
- Could use hash table – memory problems
  - counting is simple with sorted file
- Similar with distributed approach
  - sorting and placement are crucial

MapReduce

- Distributed programming framework that focuses on data placement and distribution

MapReduce

- Distributed programming framework that focuses on data placement and distribution

- *Mapper*
  - Generally, transforms a list of items into another list of items of the same length

- *Reducer*
  - Transforms a list of items into a single item
  - But definitions are not so strict in terms of number of outputs

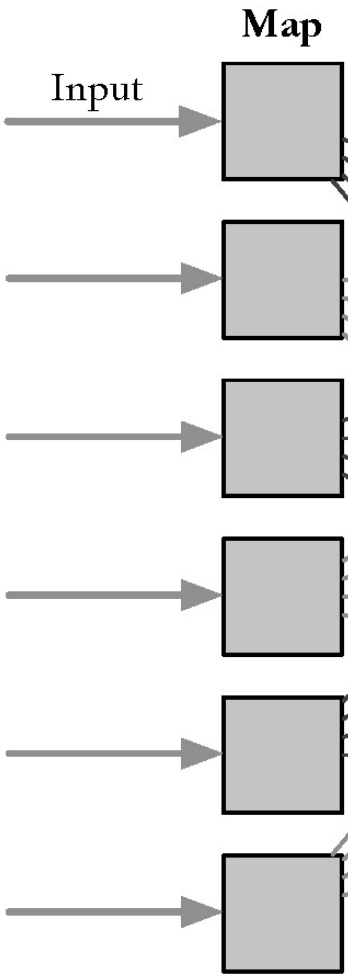
MapReduce

- Distributed programming framework that focuses on data placement and distribution
- *Mapper*
  - Generally, transforms a list of items into another list of items of the same length
- *Reducer*
  - Transforms a list of items into a single item
  - But definitions are not so strict in terms of number of outputs
- Many mapper and reducer tasks on a cluster of machines

MapReduce

- Basic process
  - *Map* stage which transforms data records into pairs, each with a key and a value
    - E.g. <key="term", value="docId">

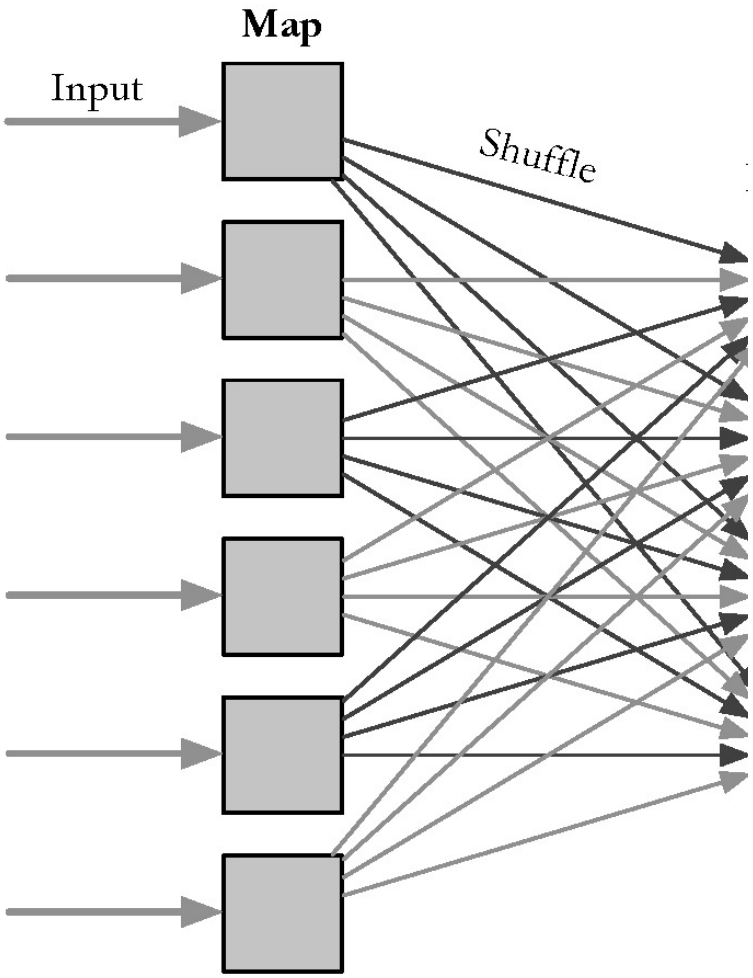
MapReduce



MapReduce

- Basic process
  - *Map* stage which transforms data records into pairs, each with a key and a value
    - E.g. <key="term", value="docId">
  - *Shuffle* uses a hash function so that all pairs with the same key end up next to each other and on the same machine
    - E.g. hash(term) mod (Numb. Red. Machines)

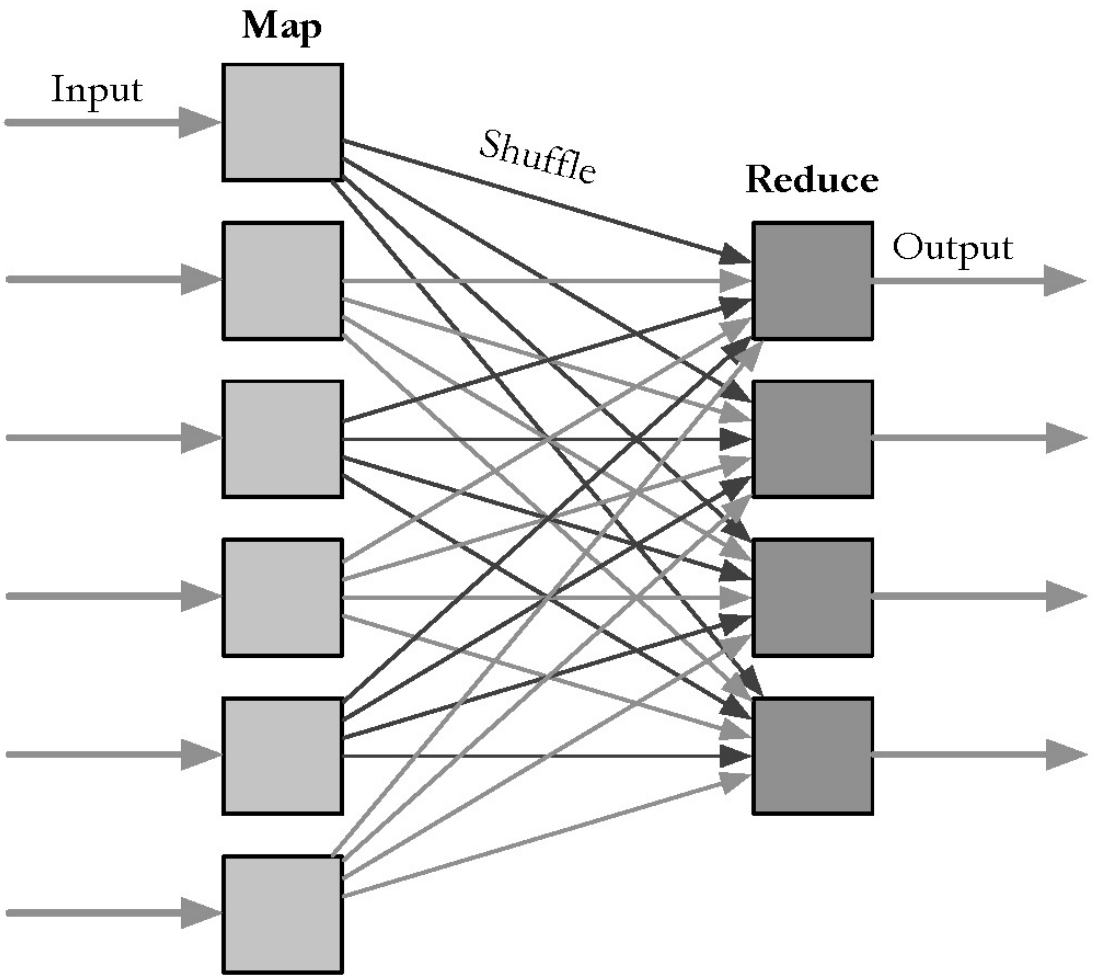
MapReduce



MapReduce

- Basic process
  - *Map* stage which transforms data records into pairs, each with a key and a value
    - E.g. <key="term", value="docId">
  - *Shuffle* uses a hash function so that all pairs with the same key end up next to each other and on the same machine
    - E.g. hash(term) mod (Numb. Red. Machines)
  - *Reduce* stage processes records in batches, where all pairs with the same key are processed at the same time
    - E.g. all <key="term", value="docId"> for a same key are inside the same machine, so you can "reduce" creating the posting list…

MapReduce



MapReduce

- Basic process
  - *Map* stage which transforms data records into pairs, each with a key and a value
  - *Shuffle* uses a hash function so that all pairs with the same key end up next to each other and on the same machine
  - *Reduce* stage processes records in batches, where all pairs with the same key are processed at the same time

- *Idempotence* of Mapper and Reducer provides fault tolerance
  - Multiple operations on same input gives same output
  - Library takes care of fault tolerance: if a machine fails, or if it is slow, computations are redundant, avoiding bottlenecks

Example

```
procedure MapCreditCards(input)
    while not input.done() do
        record ← input.next()
        card ← record.card
        amount ← record.amount
        Emit(card, amount)
    end while
end procedure

procedure ReduceCreditCards(key, values)
    total ← 0
    card ← key
    while not values.done() do
        amount ← values.next()
        total ← total + amount
    end while
    Emit(card, total)
end procedure
```

Indexing Example

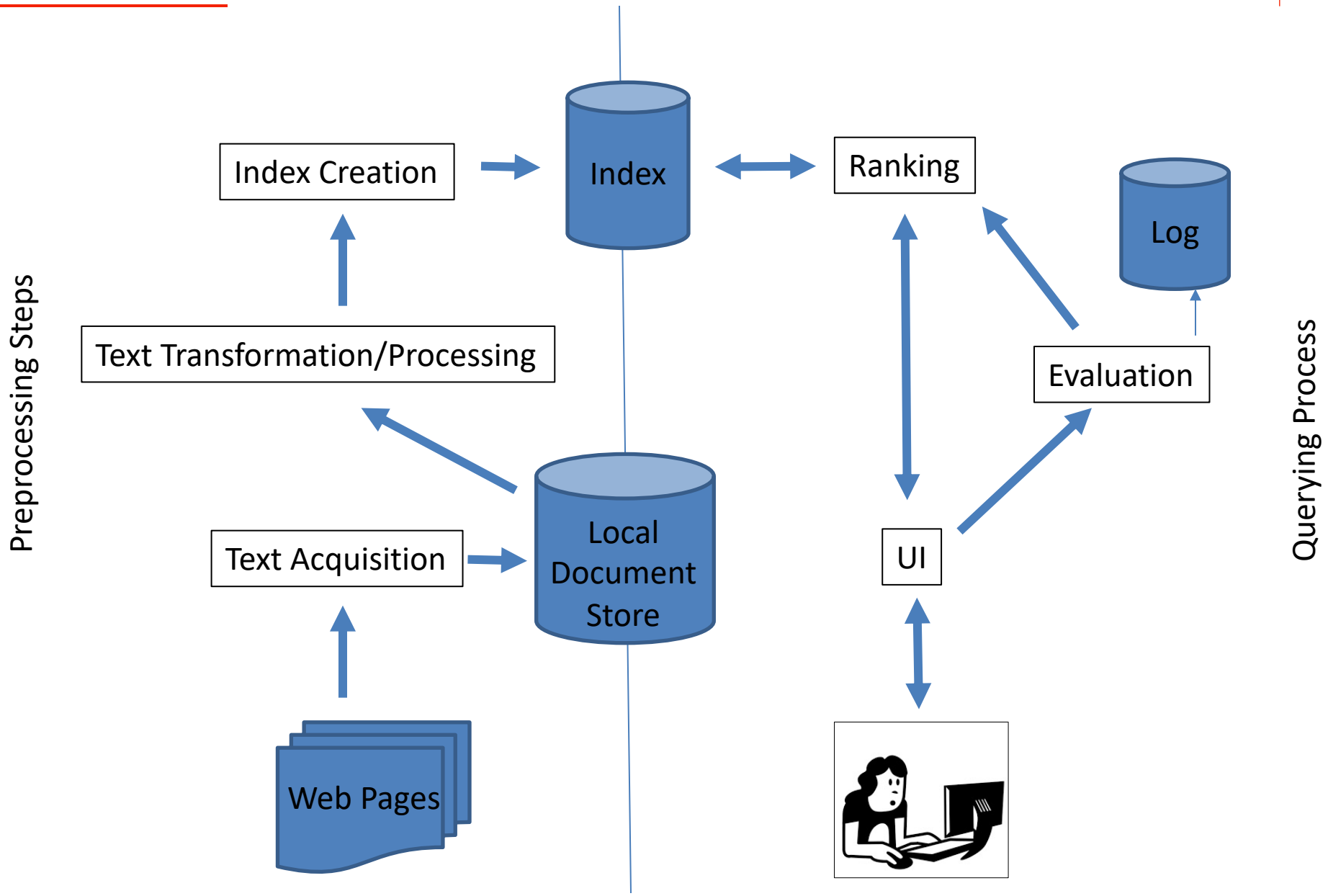
```
procedure MapDocumentsToPostings(input)
    while not input.done() do
        document ← input.next()
        number ← document.number
        position ← 0
        tokens ← Parse(document)
        for each word w in tokens do
            Emit(w, number:position)
            position = position + 1
        end for
    end while
end procedure

procedure ReducePostingsToLists(key, values)
    word ← key
    WriteWord(word)
    while not input.done() do
        EncodePosting(values.next())
    end while
end procedure
```

Map Reduce

- Typically used for data *streams*
  - No end
  - No “final index”
  - Only a temporarily valid index

Architecture



Updating: Merging Index or Result?

- Dynamic collections! The web changes continuously…

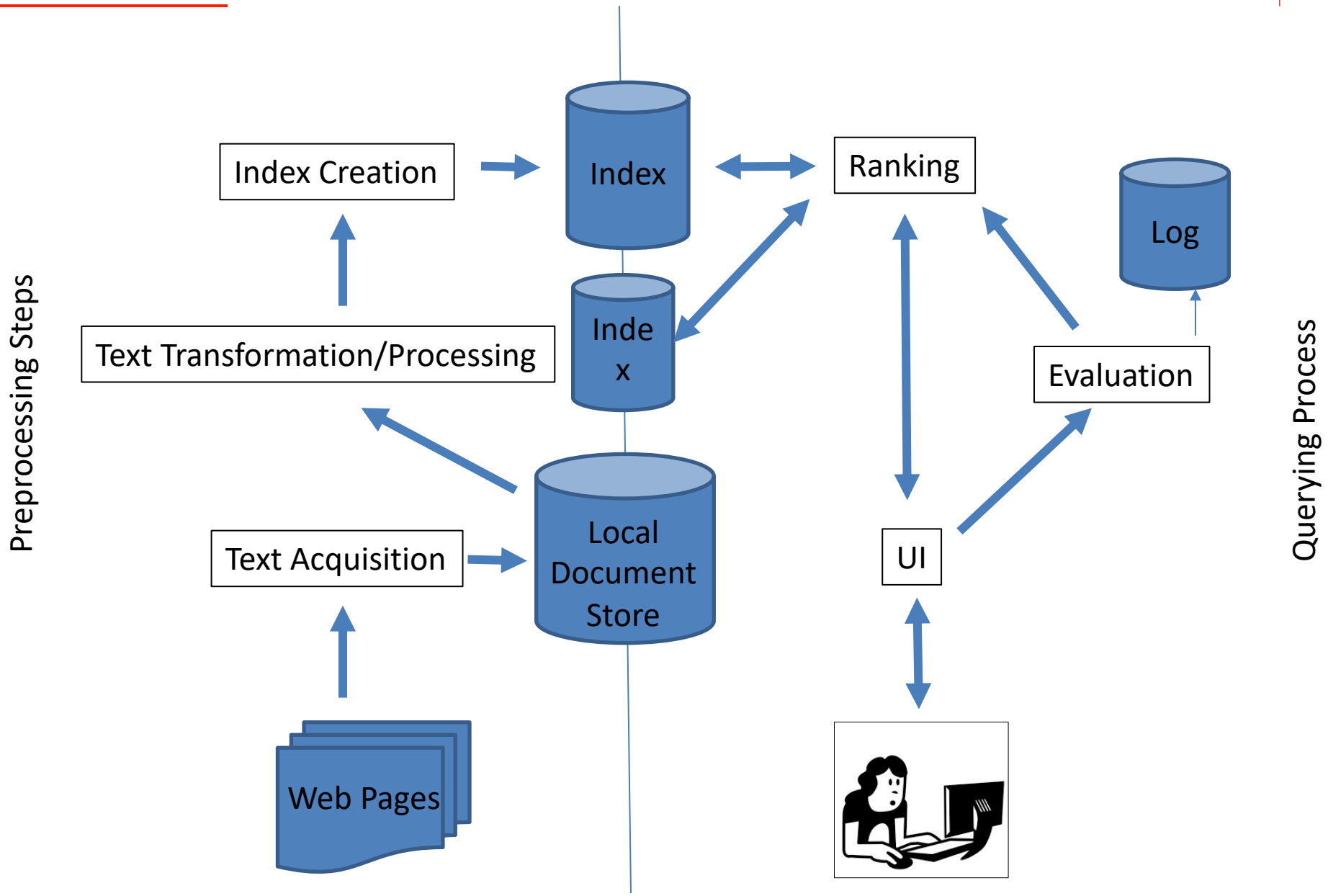
Updating: Merging Index or Result?

- Dynamic collections! The web changes continuously…
- **Index merging** (as we saw previously) is a good strategy for handling updates when they come in large batches

Updating: Merging Index or Result?

- Dynamic collections! The web changes continuously…
- **Index merging** (as we saw previously) is a good strategy for handling updates when they come in large batches
- For small updates this is very inefficient
  - **instead, create separate index for new documents, merge *results a posteriori* from the searchs in both indexes**
  - The small list could perhaps be in-memory: fast to update and search

Architecture



Updating: Merging Index or Result?

- Dynamic collections! The web changes continuously…
- **Index merging** (as we saw previously) is a good strategy for handling updates when they come in large batches
- For small updates this is very inefficient
  - **instead, create separate index for new documents, merge results a posteriori from the searchs in both indexes**
  - The small list could perhaps be in-memory: fast to update and search
- Deletions handled using *delete list*
  - Modifications done by putting old version on delete list, adding new version to new documents index

Architecture