Cloud Computing

Information Retrieval in the Cloud

Dell Zhang
Birkbeck, University of London
2018/19

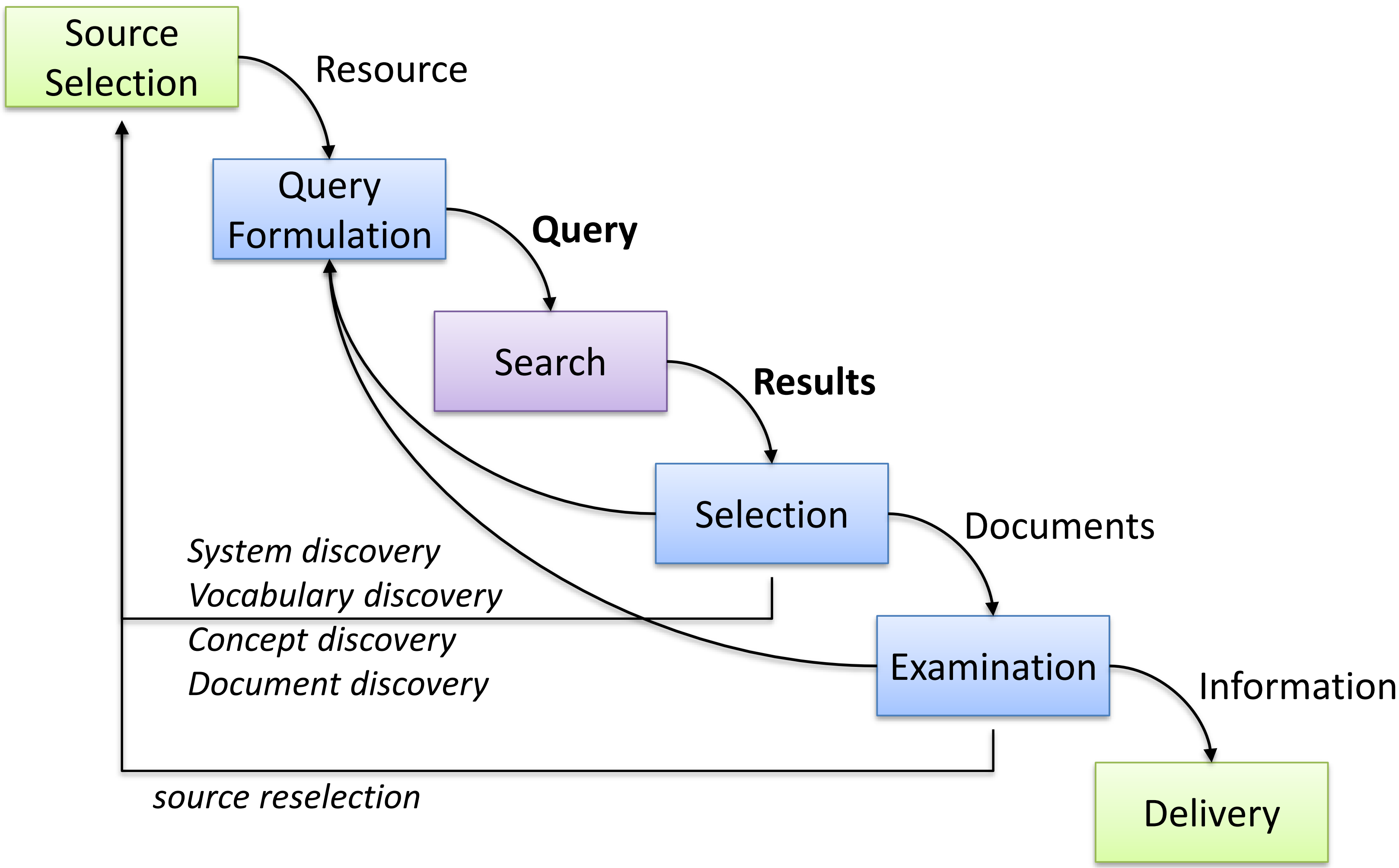
First, nomenclature…

- Information Retrieval (IR)
  - Focus on textual information (= text/document retrieval)
  - Other possibilities include image, video, music, …

First, nomenclature…

- What do we search?
  - Generically, “collections”
  - Less-frequently used, “corpora”
- What do we find?
  - Generically, “documents”
  - Even though we may be referring to web pages, PDFs, PowerPoint slides, paragraphs, etc.

Information Retrieval Cycle



The Central Problem in Search

Searcher

Concepts

Query Terms
"tragic love story"

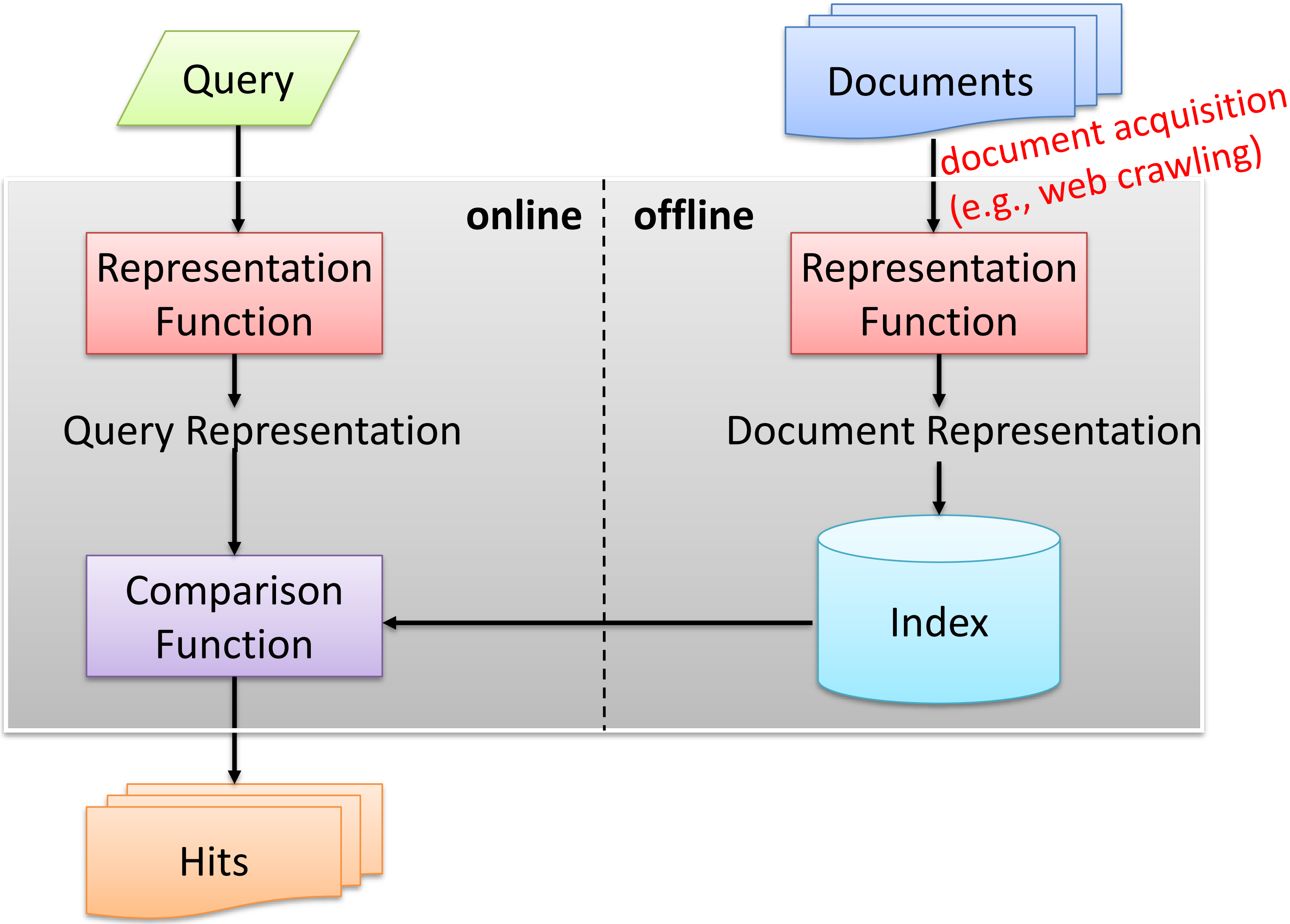
Author

Concepts

Document Terms
"fateful star-crossed romance"

Do these represent the same concepts?

Abstract IR Architecture



How do we represent text?

- "Bag of words"
  - Treat all the words in a document as index terms
  - Assign a "weight" to each term based on "importance"
    (or, in simplest case, presence/absence of word)
  - Disregard order, structure, meaning, etc. of the words
  - Simple, yet effective!

How do we represent text?

- Assumptions
  - Term occurrence is independent
  - Document relevance is independent
  - “Words” are well-defined

What’s a word?

天主教教宗若望保祿二世因感冒再度住進醫院。這是他今年第二度因同樣的病因住院。

وقال مارك ريجيف - الناطق باسم الخارجية الإسرائيلية - إن شارون قبل الدعوة وسيقوم للمرة الأولى بزيارة تونس، التي كانت لفترة طويلة المقر الرسمي لمنظمة التحرير الفلسطينية بعد خروجها من لبنان عام 1982.

Выступая в Мещанском суде Москвы экс-глава ЮКОСа заявил не совершал ничего противозаконного, в чем обвиняет его генпрокуратура России.

भारत सरकार ने आर्थिक सर्वेक्षण में वित्तीय वर्ष 2005-06 में सात फ़ीसदी विकास दर हासिल करने का आकलन किया है और कर सुधार पर ज़ोर दिया है

日米連合で台頭中国に対処…アーミテージ前副長官提言

조재영 기자= 서울시는 25일 이명박 시장이 `행정중심복합도시'' 건설안에 대해 `군대라도 동원해 막고싶은 심정''이라고 말했다는 일부 언론의 보도를 부인했다.

Sample Document

McDonald's slims down spuds

Fast-food chain to reduce certain types of fat in its french fries with new cooking oil.

NEW YORK (CNN/Money) - McDonald's Corp. is cutting the amount of "bad" fat in its french fries nearly in half, the fast-food chain said Tuesday as it moves to make all its fried menu items healthier.

But does that mean the popular shoestring fries won't taste the same? The company says no. "It's a win-win for our customers because they are getting the same great french-fry taste along with an even healthier nutrition profile," said Mike Roberts, president of McDonald's USA.

But others are not so sure. McDonald's will not specifically discuss the kind of oil it plans to use, but at least one nutrition expert says playing with the formula could mean a different taste.

Shares of Oak Brook, Ill.-based McDonald's (MCD: down $0.54 to $23.22, Research, Estimates) were lower Tuesday afternoon. It was unclear Tuesday whether competitors Burger King and Wendy's International (WEN: down $0.80 to $34.91, Research, Estimates) would follow suit. Neither company could immediately be reached for comment.

…

"Bag of Words"

14 × McDonalds

12 × fat

11 × fries

8 × new

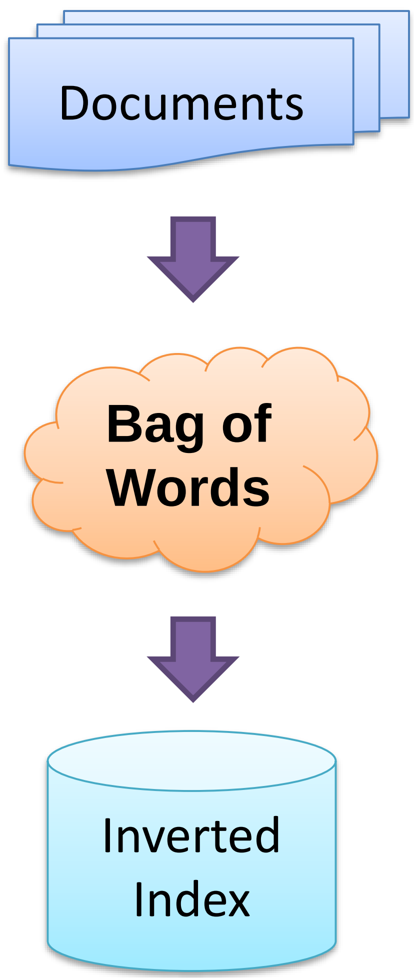
7 × french

6 × company, said, nutrition

5 × food, oil, percent, reduce, taste, Tuesday

…

Counting Words...



case folding, tokenization, stopword removal, stemming

~~syntax~~, ~~semantics~~, ~~world knowledge~~, etc.

Boolean Retrieval

- Users express queries as a Boolean expression
  - AND, OR, NOT
  - Can be arbitrarily nested
- Retrieval is based on the notion of sets
  - Any given query divides the collection into two sets: retrieved, not-retrieved
  - Pure Boolean systems do not define an ordering of the results

Inverted Index: Boolean Retrieval

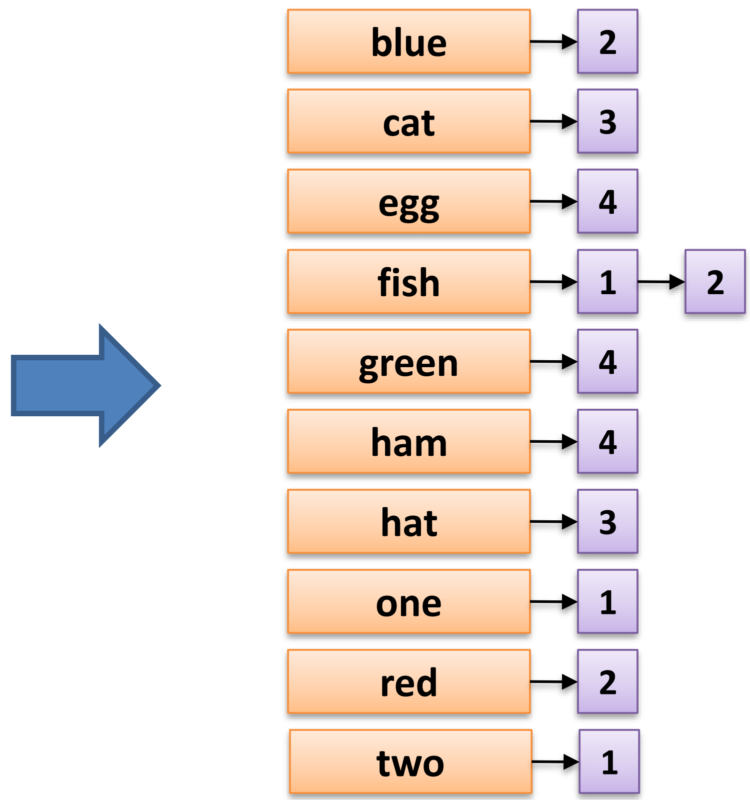
Doc 1
one fish, two fish

Doc 2
red fish, blue fish

Doc 3
cat in the hat

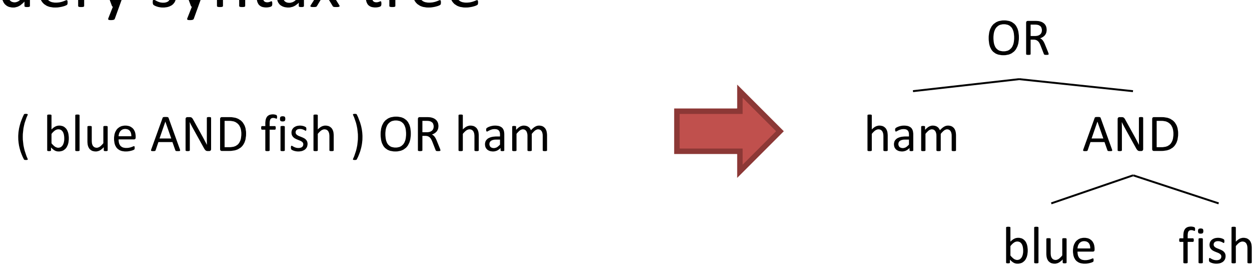
Doc 4
green eggs and ham

	1	2	3	4
blue		1		
cat			1	
egg				1
fish	1	1		
green				1
ham				1
hat			1	
one	1			
red		1		
two	1			

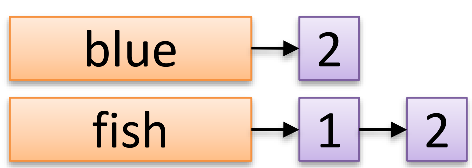


Boolean Retrieval

- To execute a Boolean query:
  - Build query syntax tree



  - For each clause, look up postings



  - Traverse postings and apply Boolean operator

Boolean Retrieval

- Efficiency analysis
  - Postings traversal is linear (assuming sorted postings)
  - Start with shortest posting first

Strengths and Weaknesses

- Strengths
  - Precise
    - If you know the right strategies
    - If you have an idea of what you're looking for
  - Implementations are fast and efficient

Strengths and Weaknesses

- Weaknesses
  - Users must learn Boolean logic
  - Boolean logic insufficient to capture the richness of language
  - No control over size of result set: either too many hits or none
  - When do you stop reading? All documents in the result set are considered “equally good”
  - What about partial matches? Documents that “don’t quite match” the query may be useful also

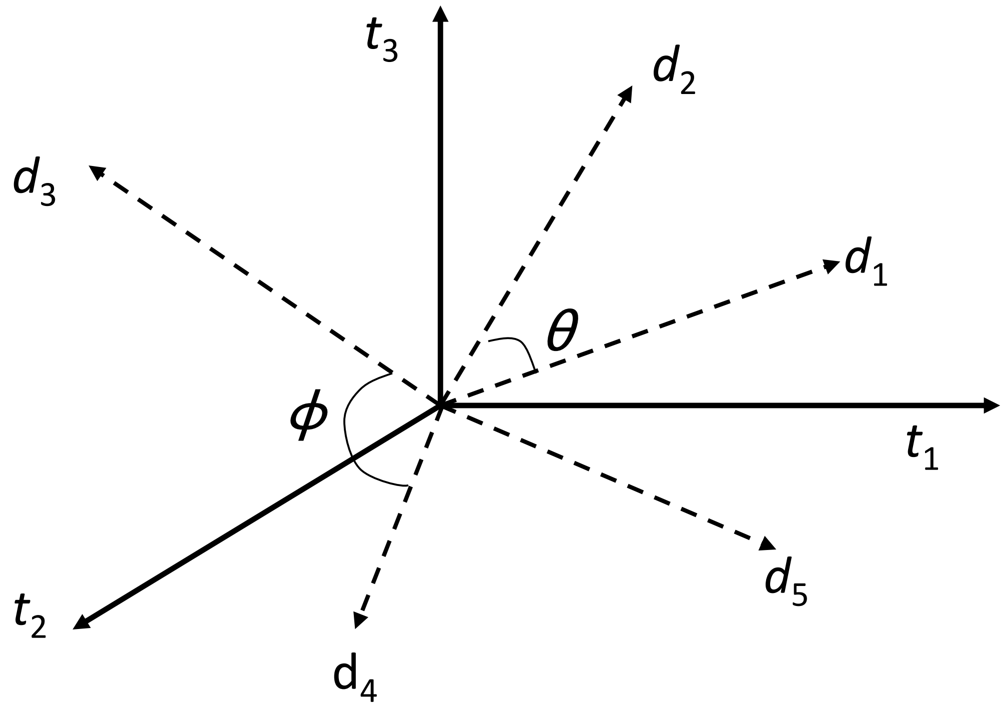
Ranked Retrieval

- Order documents by how likely they are to be relevant to the information need
  - Estimate relevance(q, d_i)
  - Sort documents by relevance
  - Display sorted results
- User model
  - Present hits one screen at a time, best results first
  - At any point, users can decide to stop looking

Ranked Retrieval

- How do we estimate relevance?
  - Assume document is relevant if it has a lot of query terms
  - Replace relevance(q, d_i) with sim(q, d_i)
  - Compute similarity of vector representations

Vector Space Model



Assumption:

Documents that are "close together" in vector space "talk about" the same things

Therefore, retrieve documents based on how close the document is to the query (i.e., similarity ~ "closeness")

Similarity Metric

- Use "angle" between the vectors:

$$\cos(\theta) = \frac{\vec{d}_j \cdot \vec{d}_k}{\left|\vec{d}_j\right|\left|\vec{d}_k\right|}$$

$$sim(d_j, d_k) = \frac{\vec{d}_j \cdot \vec{d}_k}{\left|\vec{d}_j\right|\left|\vec{d}_k\right|} = \frac{\sum_{i=1}^{n} w_{i,j} w_{i,k}}{\sqrt{\sum_{i=1}^{n} w_{i,j}^2}\sqrt{\sum_{i=1}^{n} w_{i,k}^2}}$$

- Or, more generally, inner products:

$$sim(d_j, d_k) = \vec{d}_j \cdot \vec{d}_k = \sum_{i=1}^{n} w_{i,j} w_{i,k}$$

Term Weighting

- Term weights consist of two components
  - Local: how important is the term in this document?
  - Global: how important is the term in the collection?

Term Weighting

- Here's the intuition:
  - Local: Terms that appear often in a document should get high weights
  - Global: Terms that appear in many documents should get low weights
- How do we capture this mathematically?
  - Local: Term Frequency (TF)
  - Global: Inverse Document Frequency (IDF)

TF.IDF Term Weighting

$$w_{i,j} = \mathrm{tf}_{i,j} \cdot \log \frac{N}{n_i}$$

$w_{i,j}$	weight assigned to term *i* in document *j*
$\mathrm{tf}_{i,j}$	number of occurrence of term *i* in document *j*
N	number of documents in entire collection
n_i	number of documents with term *i*

Inverted Index: TF.IDF

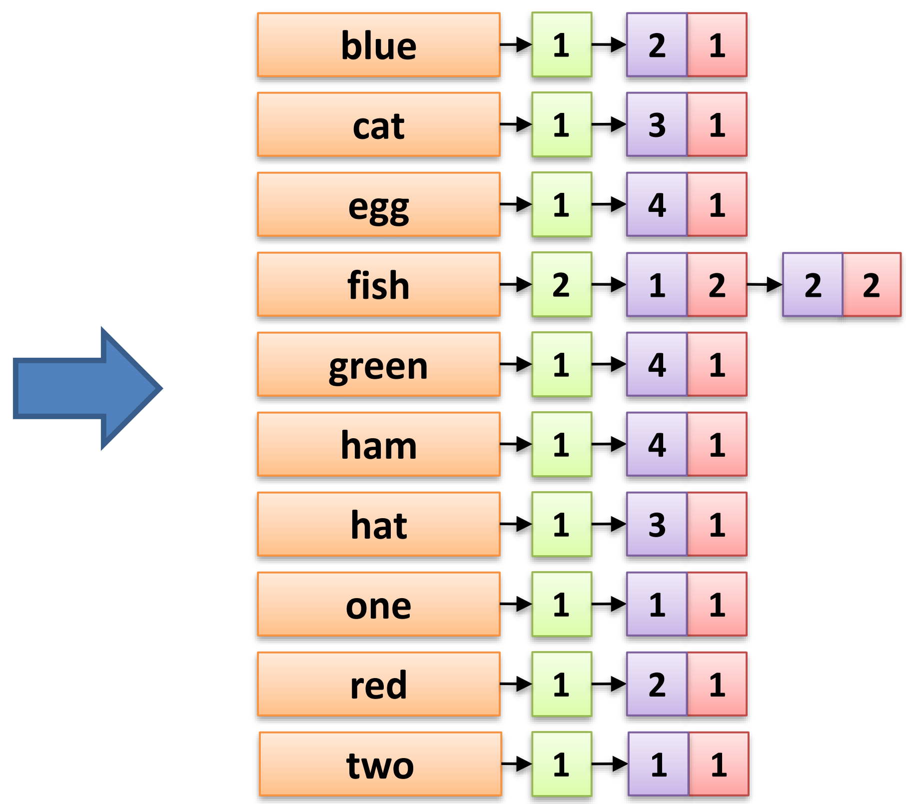
Doc 1
one fish, two fish

Doc 2
red fish, blue fish

Doc 3
cat in the hat

Doc 4
green eggs and ham

	tf 1	2	3	4	*df*
blue		1			1
cat			1		1
egg				1	1
fish	2	2			2
green				1	1
ham				1	1
hat			1		1
one	1				1
red		1			1
two	1				1



Positional Indexes

- Store term position in postings
- Supports richer queries (e.g., proximity)
- Naturally, leads to larger indexes…

Inverted Index: Positional Information

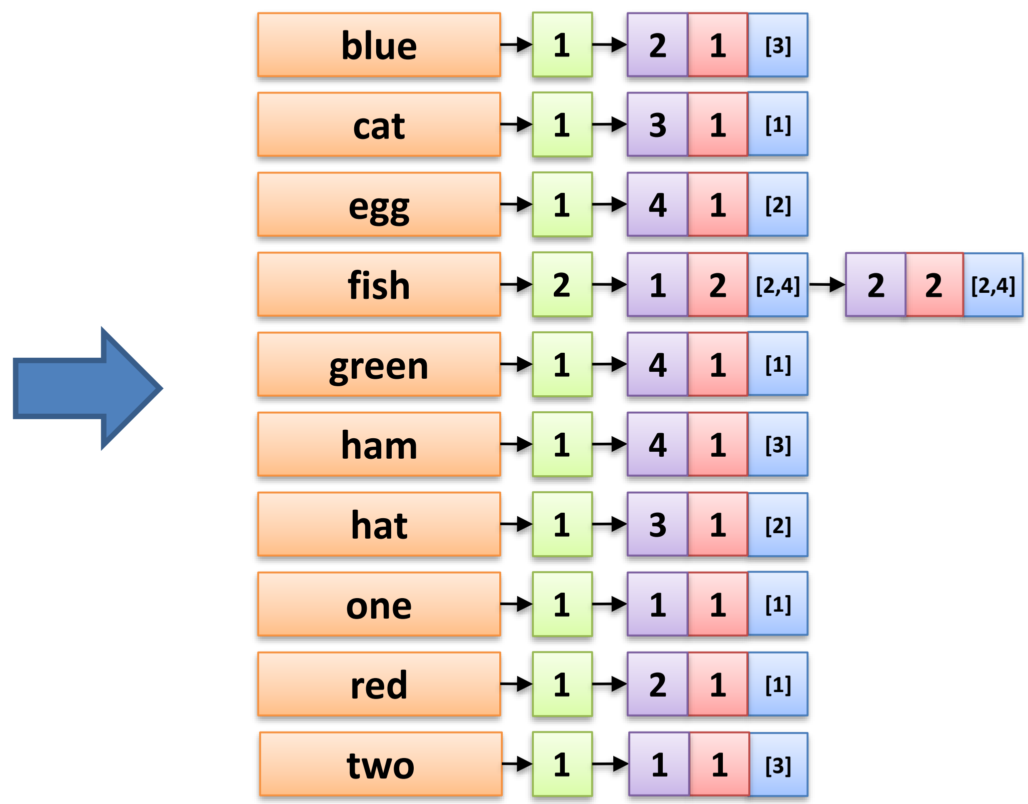
Doc 1
one fish, two fish

Doc 2
red fish, blue fish

Doc 3
cat in the hat

Doc 4
green eggs and ham

	tf 1	2	3	4	*df*
blue		1			1
cat			1		1
egg				1	1
fish	2	2			2
green				1	1
ham				1	1
hat			1		1
one	1				1
red		1			1
two	1				1

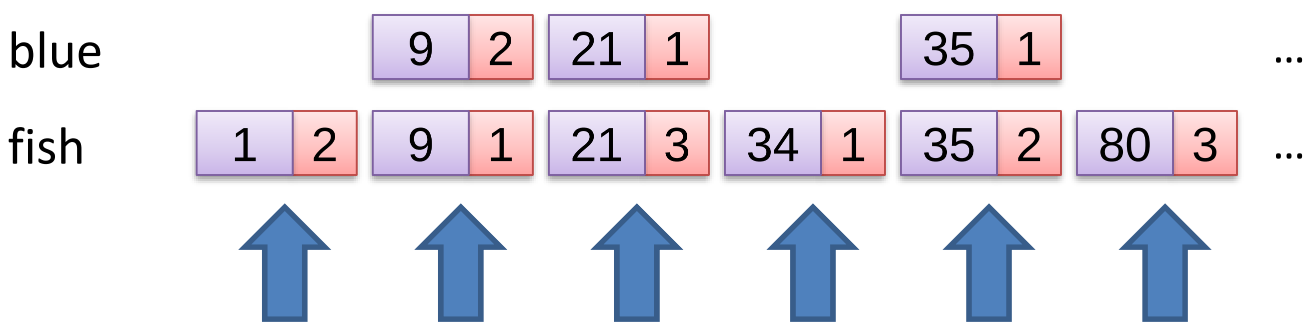


Retrieval in a Nutshell

- Look up postings lists corresponding to query terms
- Traverse postings for each query term
- Store partial query-document scores in accumulators
- Select top *k* results to return

Retrieval: Document-at-a-Time

- Evaluate documents one at a time (score all query terms)



Accumulators
(e.g. priority queue)

Document score in top *k*?

Yes: Insert document score, extract-min if queue too large

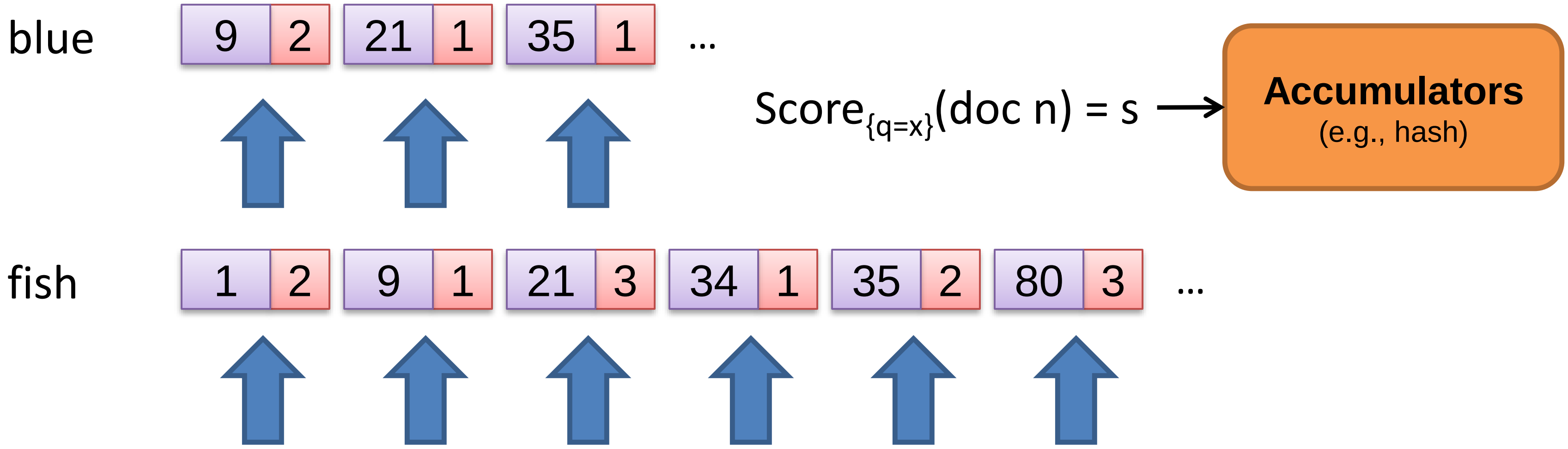
No: Do nothing

Retrieval: Document-at-a-Time

- Tradeoffs
  - Small memory footprint (good)
  - Must read through all postings (bad), but skipping possible
  - More disk seeks (bad), but blocking possible

Retrieval: Query-At-A-Time

- Evaluate documents one query term at a time
  - Usually, starting from most rare term (often with tf-sorted postings)



Retrieval: Query-At-A-Time

- Tradeoffs
  - Early termination heuristics (good)
  - Large memory footprint (bad), but filtering heuristics possible

MapReduce it?

- The indexing problem
  - Scalability is critical
  - Must be relatively fast, but need not be real time
  - Fundamentally a batch operation
  - Incremental updates may or may not be important
  - For the web, crawling is a challenge in itself

Perfect for MapReduce!

MapReduce it?

- The retrieval problem
  - Must have sub-second response time
  - For the web, only need relatively few results

Uh… not so good…

MapReduce: Index Construction

- Map over all documents
  - Emit *term* as key, (*docno*, *tf)* as value
  - Emit other info as necessary (e.g., term position)
- Sort/shuffle: group postings by term
- Reduce
  - Gather and sort the postings (e.g., by *docno* or *tf*)
  - Write postings to disk
- MapReduce does all the heavy lifting!

Inverted Indexing with MapReduce

Doc 1
one fish, two fish

Doc 2
red fish, blue fish

Doc 3
cat in the hat

Map

one 1 1
two 1 1
fish 1 2

red 2 1
blue 2 1
fish 2 2

cat 3 1
hat 3 1

Shuffle and Sort: aggregate values by keys

Reduce

cat 3 1
fish 1 2 2 2
one 1 1
red 2 1

blue 2 1
hat 3 1
two 1 1

Inverted Indexing: Pseudo-Code

class Mapper
    procedure Map(docid n, doc d)
        $H \leftarrow$ new AssociativeArray
        for all term $t \in$ doc d **do**
            $H\{t\} \leftarrow H\{t\} + 1$
        for all term $t \in H$ **do**
            Emit(term t, posting $\langle n, H\{t\}\rangle$)

class Reducer
    procedure Reduce(term t, postings $[\langle n_1, f_1\rangle, \langle n_2, f_2\rangle \ldots]$)
        $P \leftarrow$ new List
        for all posting $\langle a, f\rangle \in$ postings $[\langle n_1, f_1\rangle, \langle n_2, f_2\rangle \ldots]$ **do**
            Append($P, \langle a, f\rangle$)
        Sort(P)
        Emit(term t, postings P)

Positional Indexes

Doc 1
one fish, two fish

Doc 2
red fish, blue fish

Doc 3
cat in the hat

Map

one	1	1	[1]	red	2	1	[1]	cat	3	1	[1]
two	1	1	[3]	blue	2	1	[3]	hat	3	1	[2]
fish	1	2	[2,4]	fish	2	2	[2,4]				

Shuffle and Sort: aggregate values by keys

Reduce

cat	3	1	[1]			
fish	1	2	[2,4]	2	2	[2,4]
one	1	1	[1]			
red	2	1	[1]			

blue	2	1	[3]
hat	3	1	[2]
two	1	1	[3]

Inverted Indexing: Pseudo-Code

class MAPPER
  procedure MAP(docid n, doc d)
    $H \leftarrow$ new ASSOCIATIVEARRAY
    for all term $t \in$ doc d **do**
      $H\{t\} \leftarrow H\{t\} + 1$
    for all term $t \in H$ **do**
      EMIT(term t, posting $\langle n, H\{t\}\rangle$)

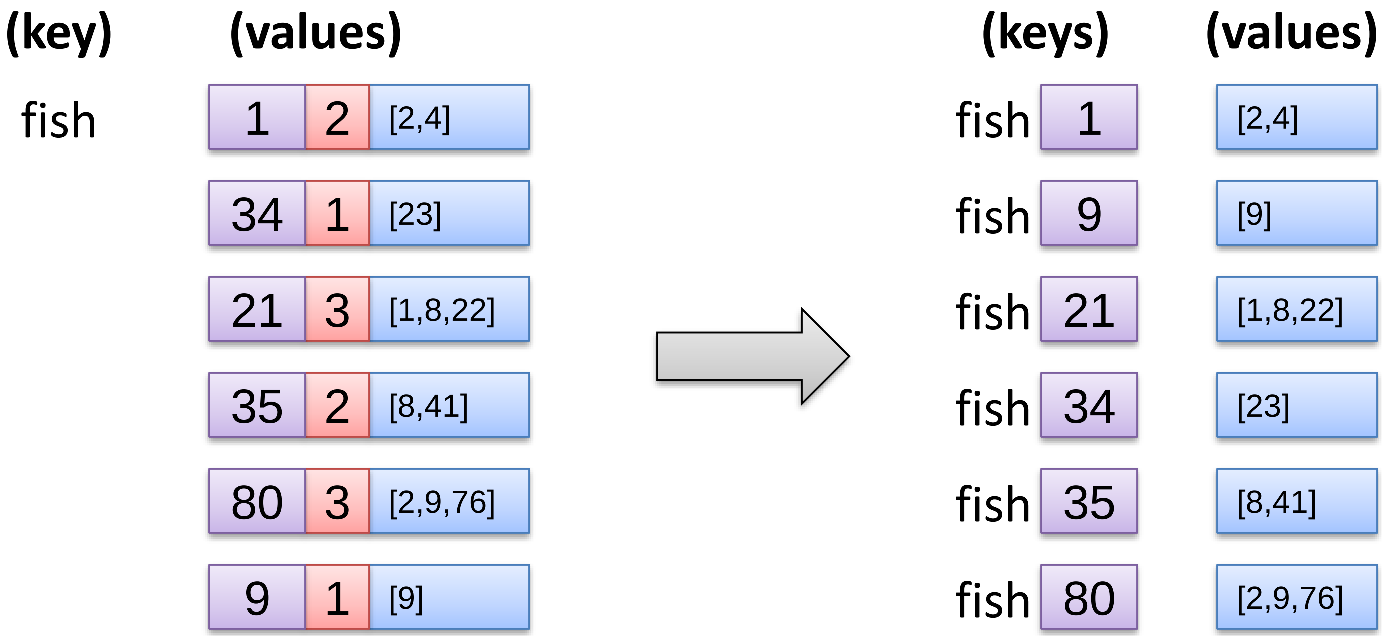
class REDUCER
  procedure REDUCE(term t, postings $[\langle n_1, f_1\rangle, \langle n_2, f_2\rangle \ldots]$)
    $P \leftarrow$ new LIST
    for all posting $\langle a, f\rangle \in$ postings $[\langle n_1, f_1\rangle, \langle n_2, f_2\rangle \ldots]$ **do**
      APPEND($P, \langle a, f\rangle$)
    SORT(P)
    EMIT(term t, postings P)

What's the problem?

Scalability Bottleneck

- Initial implementation:
  terms as keys, postings as values
  - Reducers must buffer all postings associated with key (to sort)
  - What if we run out of memory to buffer postings?
- Uh oh!

Another Try…



How is this different?

- Let the framework do the sorting
- Term frequency implicitly stored
- Directly write postings to disk!

Where have we seen this before?

Inverted Indexing: Pseudo-Code

```
class Mapper
    method Map(docid n, doc d)
        H ← new AssociativeArray
        for all term t ∈ doc d do
            H{t} ← H{t} + 1
        for all term t ∈ H do
            Emit(tuple ⟨t, n⟩, tf H{t})
```

```
class Reducer
    method Initialize
        t_prev ← ∅
        P ← new PostingsList
    method Reduce(tuple ⟨t, n⟩, tf [f])
        if t ≠ t_prev ∧ t_prev ≠ ∅ then
            Emit(term t_prev, postings P)
            P.Reset()
        P.Add(⟨n, f⟩)
        t_prev ← t
    method Close
        Emit(term t, postings P)
```

Retrieval with MapReduce?

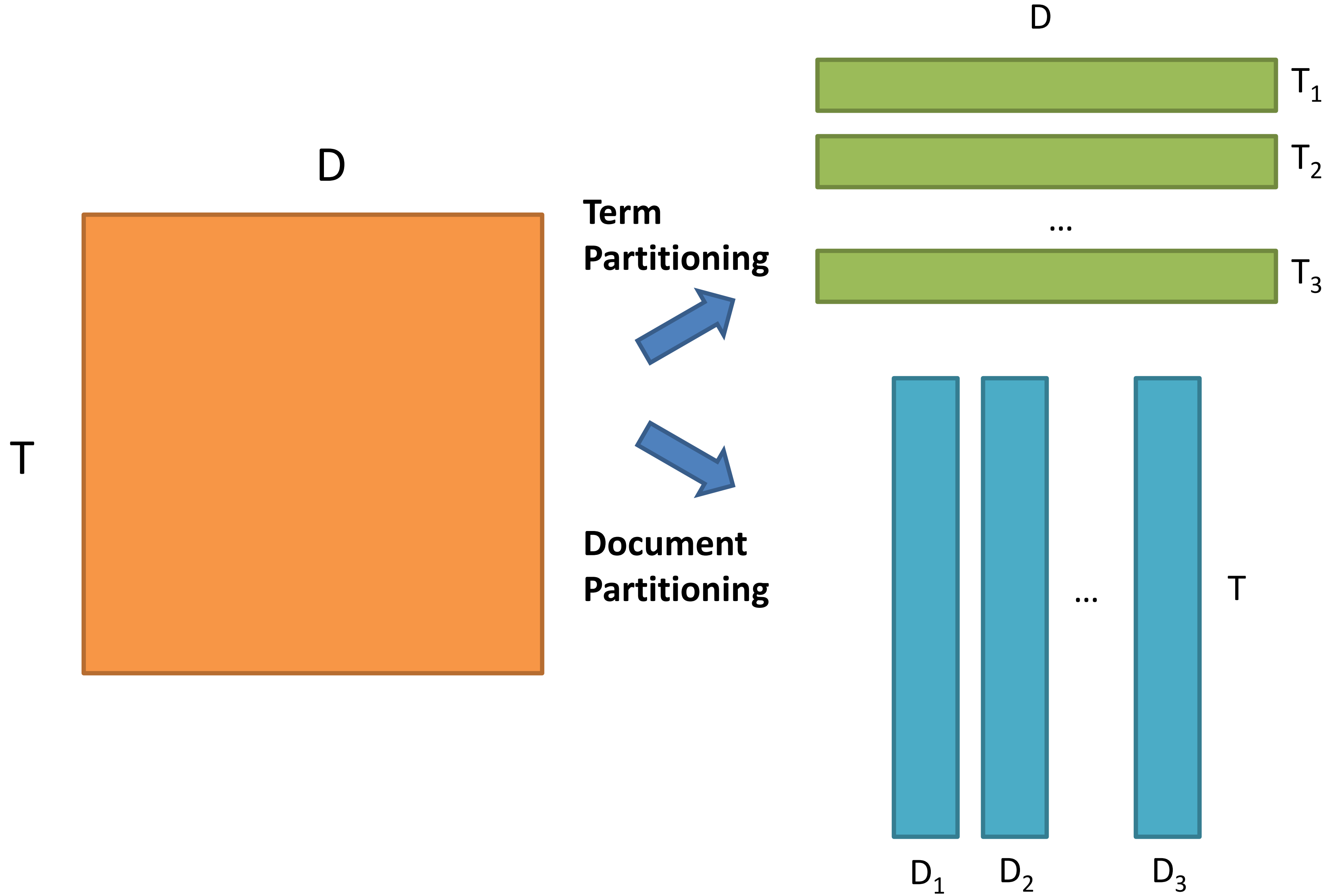
- MapReduce is fundamentally batch-oriented
  - Optimized for throughput, not latency
  - Startup of mappers and reducers is expensive
- MapReduce is not suitable for *real-time* queries!
  - Use separate infrastructure for retrieval…

Important Ideas

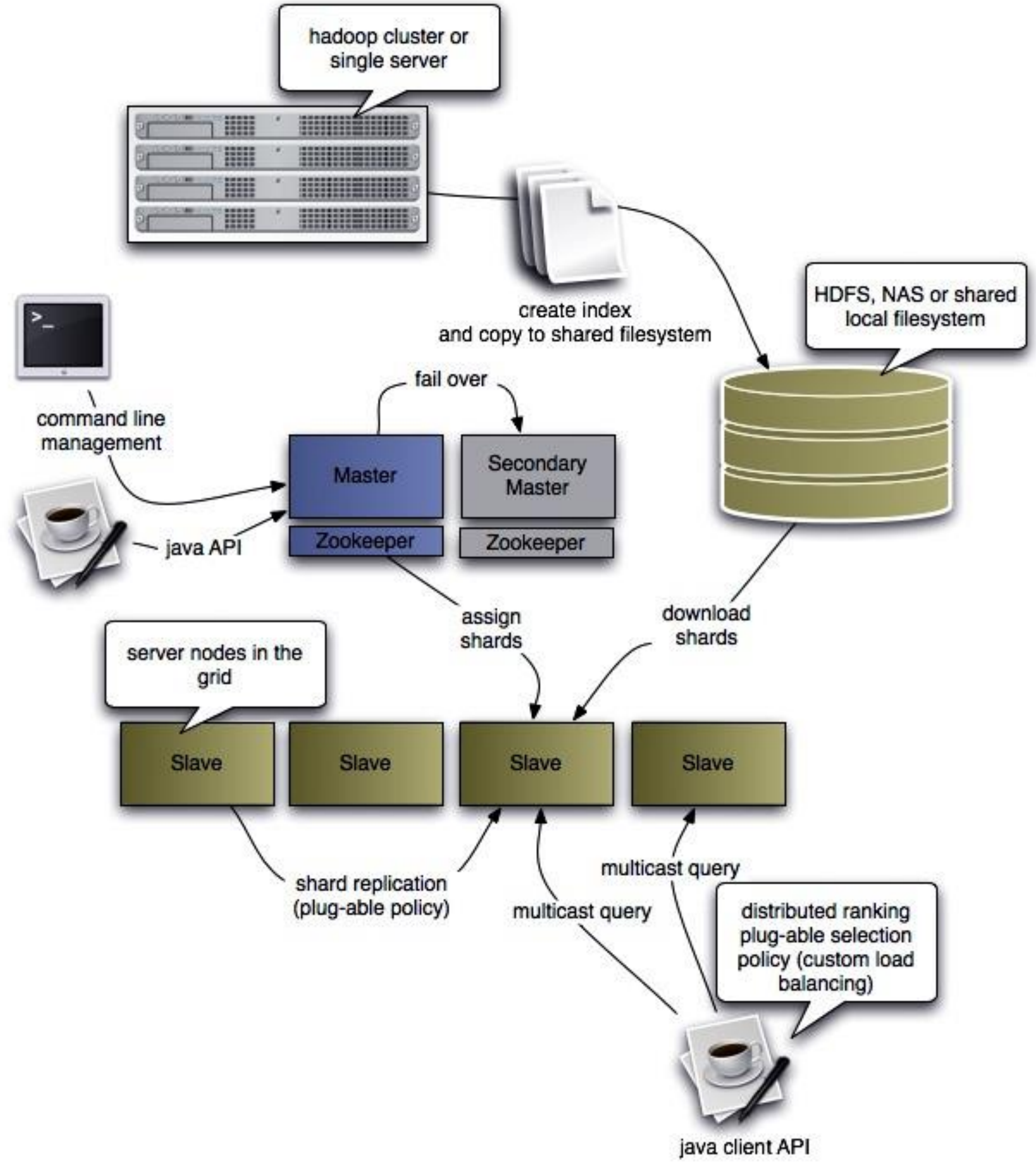
- Partitioning (for scalability)
- Replication (for redundancy)
- Caching (for speed)
- Routing (for load balancing)

The rest is just details!

Term vs. Document Partitioning



(Distributed Lucene)



http://katta.sourceforge.net/

Take Home Messages

- Introduction to Information Retrieval
- Basics of indexing and retrieval
- Inverted indexing in MapReduce
- Retrieval at scale