Data-Intensive Information Processing Applications — Session #4

Text Retrieval Algorithms

Jordan Boyd-Graber
University of Maryland

Thursday, February 24, 2010

Source: Wikipedia (Japanese rock garden)

Old Business

- When do VMs get initialized?
- HW1 Done!
- HW2 on the Horizon ...
- Projects
  - 2-3 People
  - Fairly open scope
  - Next few lectures should give you ideas

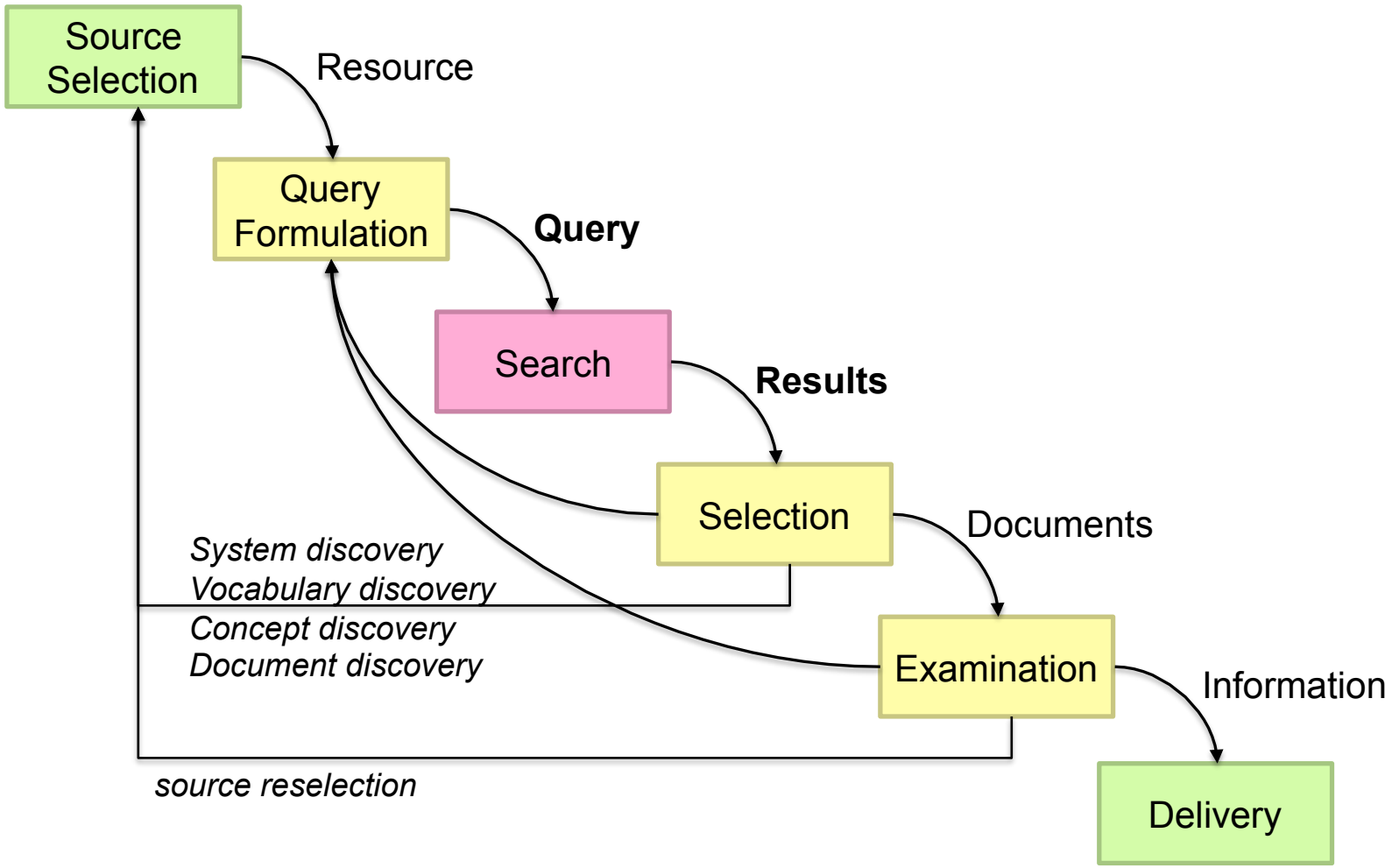
Today's Agenda

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

First, nomenclature…

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

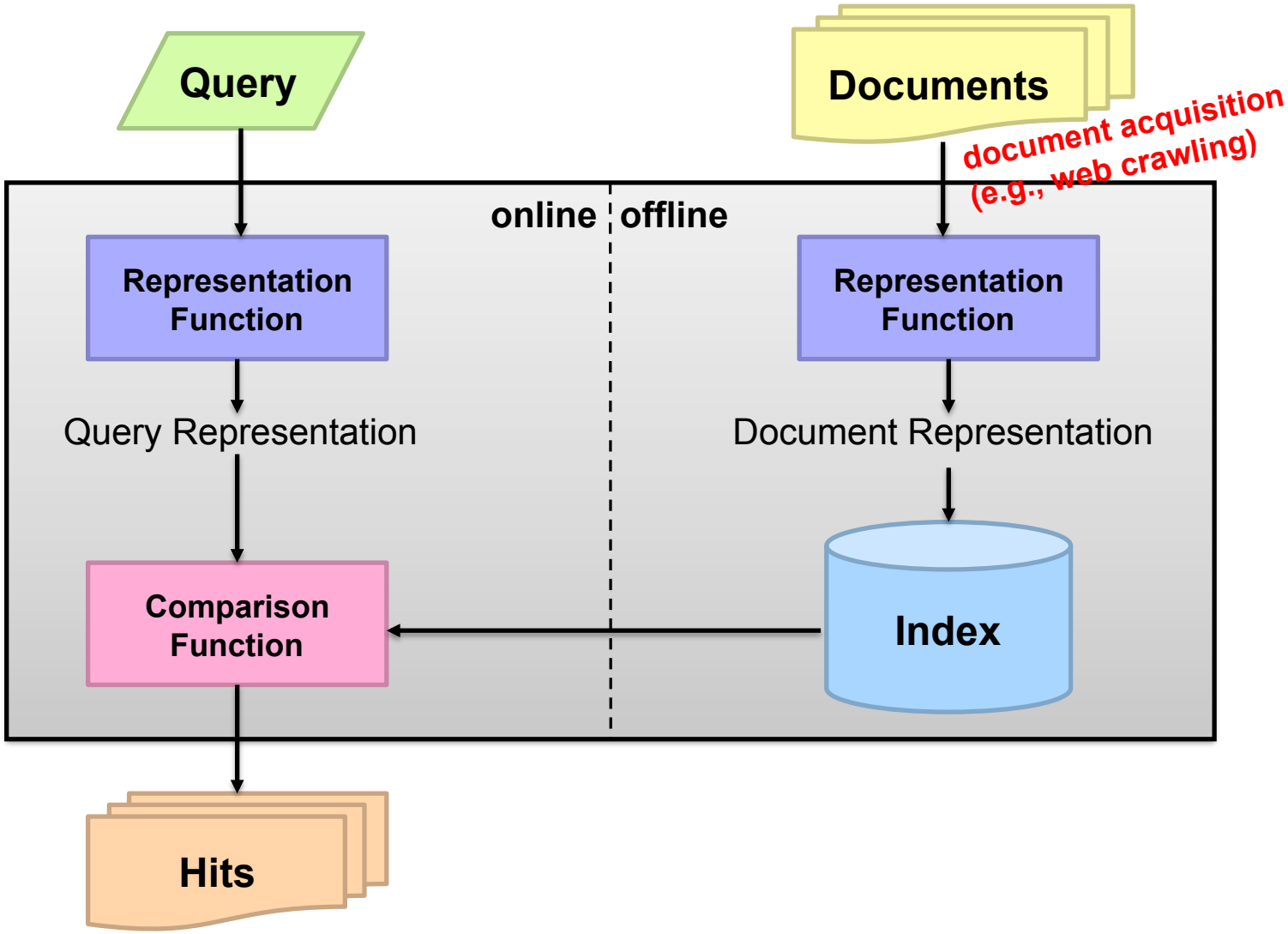
Concepts

Document Terms

"fateful star-crossed romance"

Do these represent the same concepts?

Abstract IR Architecture



How do we represent text?

- Remember: computers don't "understand" anything!
- "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!
- 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 2 .

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.

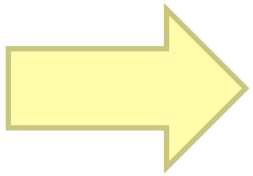
NEW YORK (CNN/Money) - McDonald's Corp. is cutting the amount of "bad" fat in its french fries nearly in half, the fast-food chain said Tuesday as it moves to make all its fried menu items healthier.

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

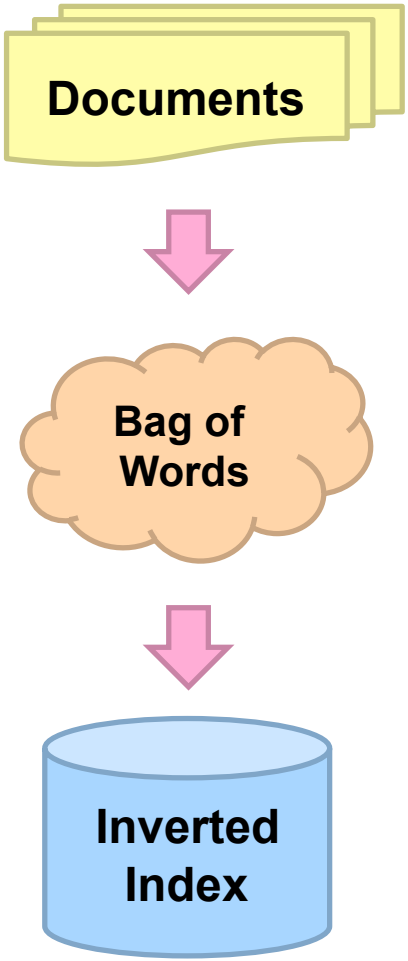
7 × french

6 × company, said, nutrition

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

…

Counting Words…



case folding, tokenization, stopword removal, stemming

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

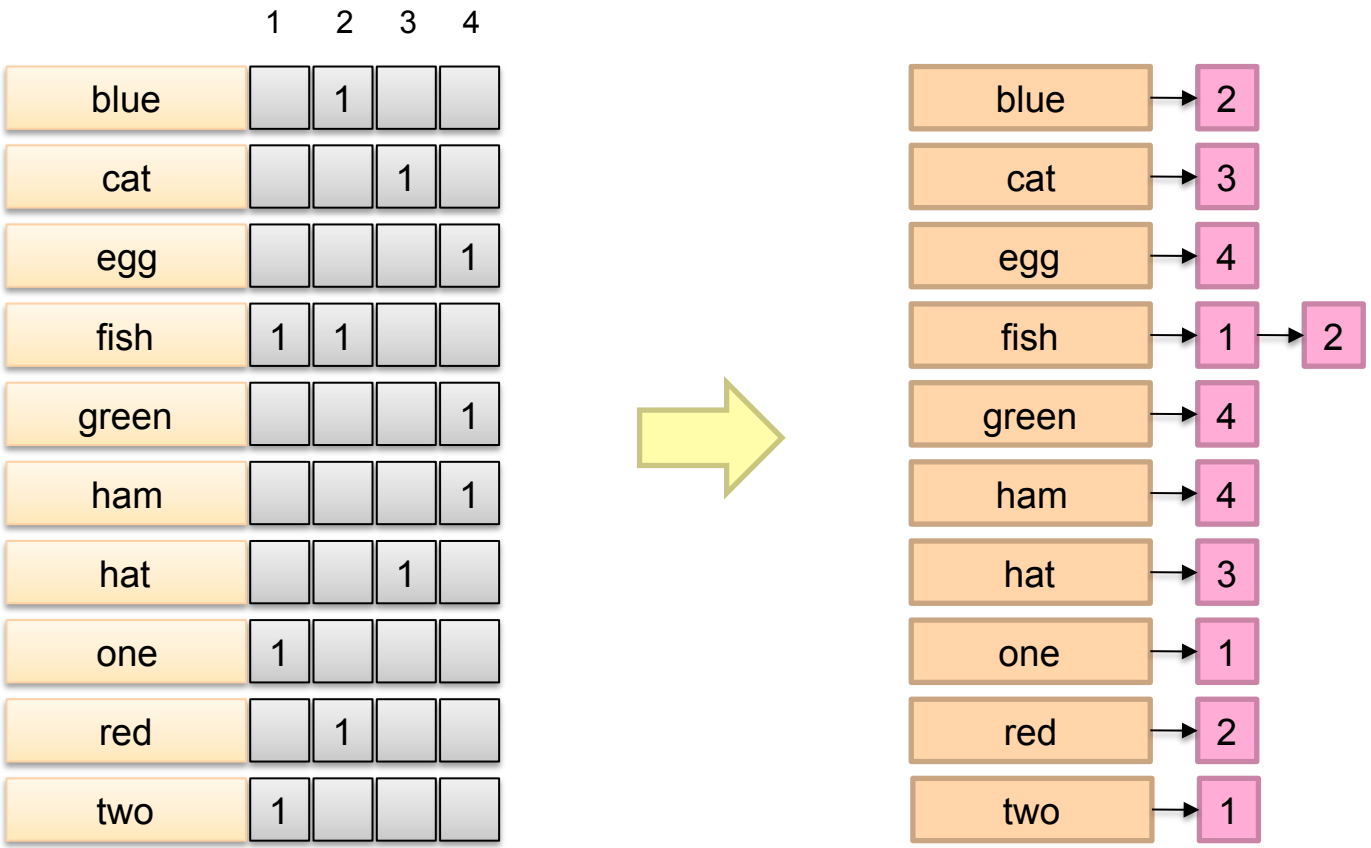
Inverted Index: Boolean Retrieval

Doc 1
one fish, two fish

Doc 2
red fish, blue fish

Doc 3
cat in the hat

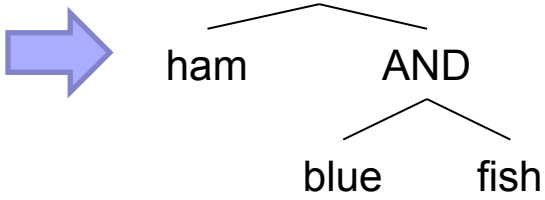
Doc 4
green eggs and ham



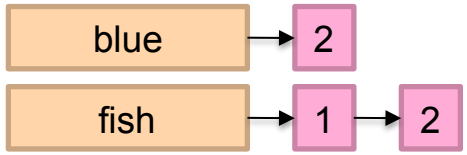
Boolean Retrieval

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

(blue AND fish) OR ham



  - For each clause, look up postings



  - Traverse postings and apply Boolean operator
- Efficiency analysis
  - Postings traversal is linear (assuming sorted postings)
  - Start with shortest posting first

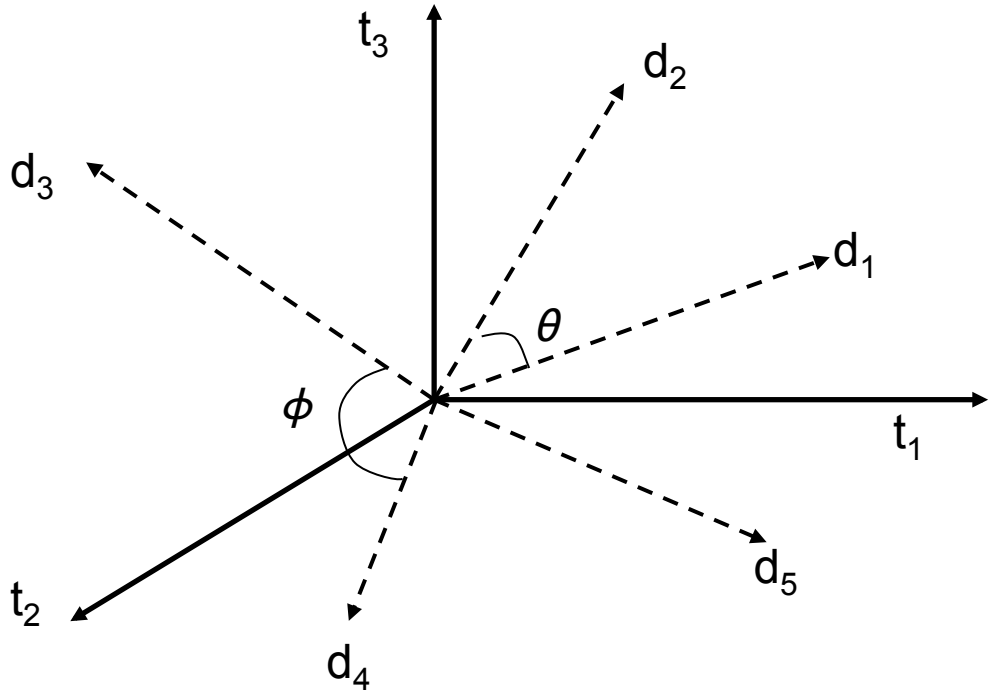
Strengths and Weaknesses

- Strengths
  - Precise, if you know the right strategies
  - Precise, if you have an idea of what you're looking for
  - Implementations are fast and efficient
- 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
- 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?
- Here's the intuition:
  - Terms that appear often in a document should get high weights
  - Terms that appear in many documents should get low weights
- How do we capture this mathematically?
  - Term frequency (local)
  - Inverse document frequency (global)

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***

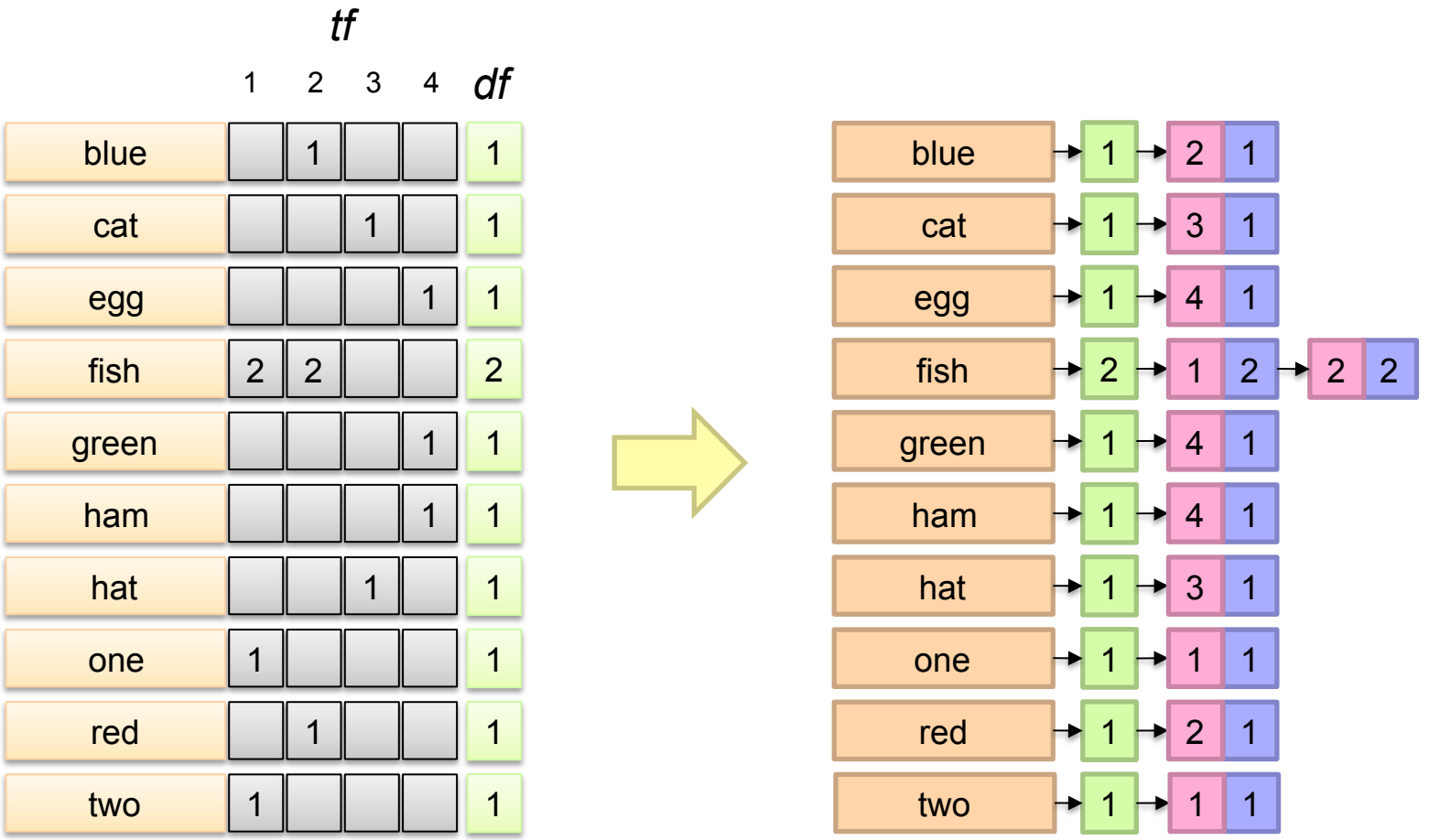
Inverted Index: TF.IDF

Doc 1 one fish, two fish

Doc 2 red fish, blue fish

Doc 3 cat in the hat

Doc 4 green eggs and ham



Positional Indexes

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

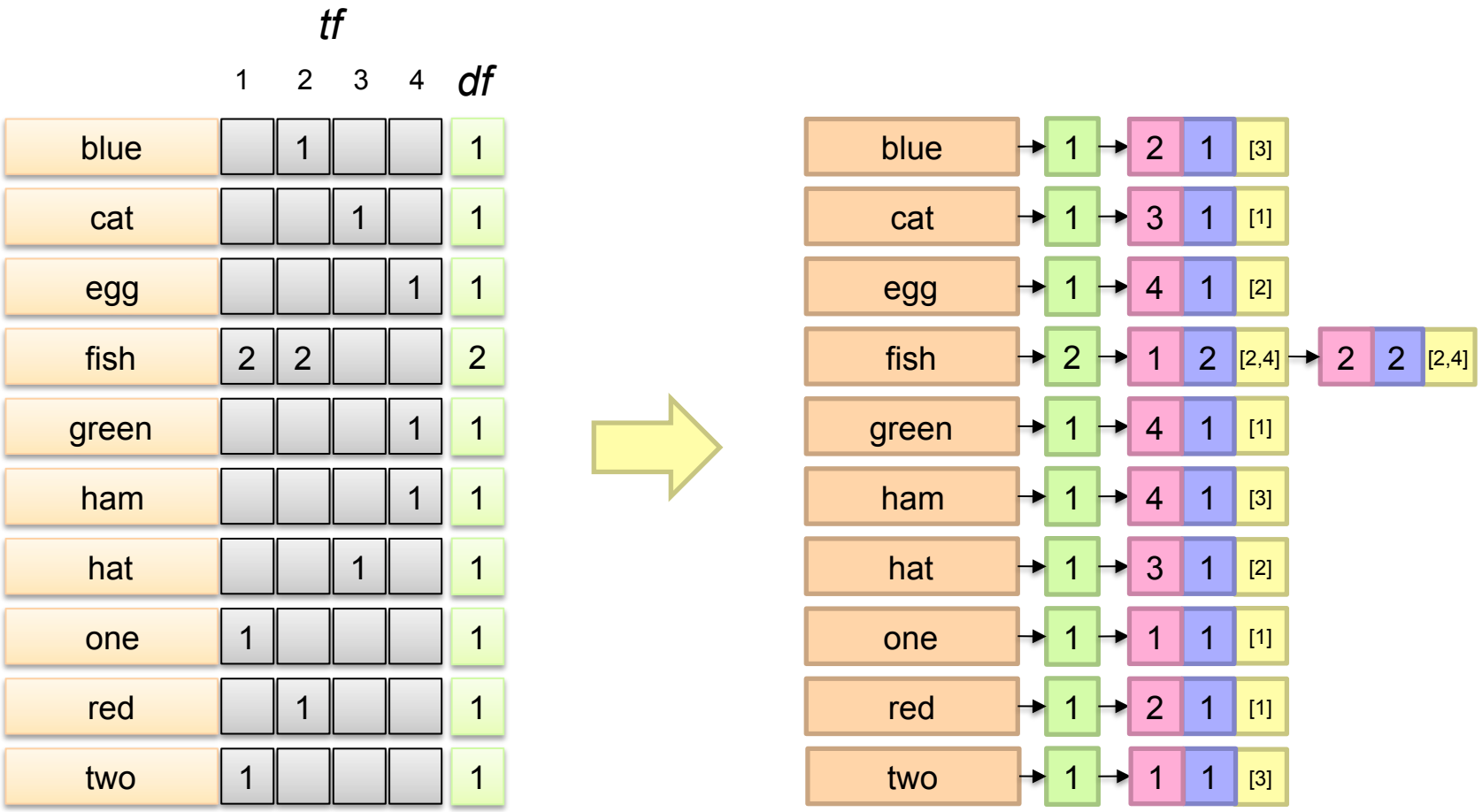
Inverted Index: Positional Information

Doc 1 one fish, two fish

Doc 2 red fish, blue fish

Doc 3 cat in the hat

Doc 4 green eggs and ham

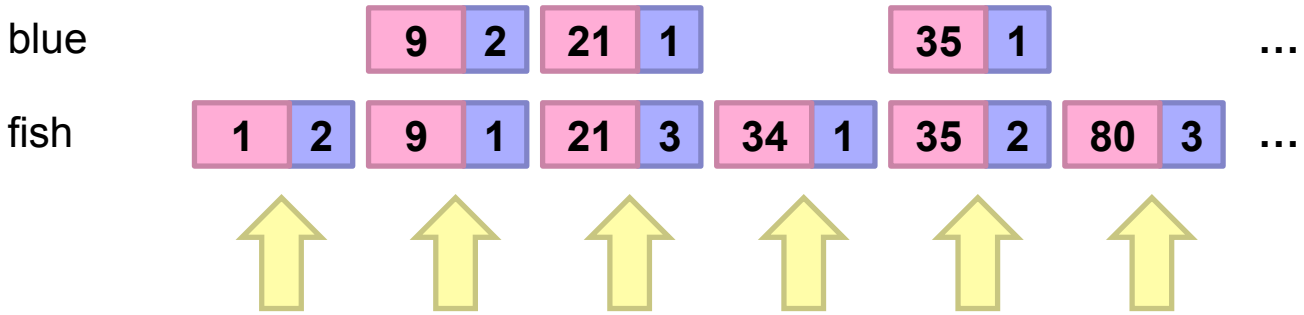


Retrieval in a Nutshell

- 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)

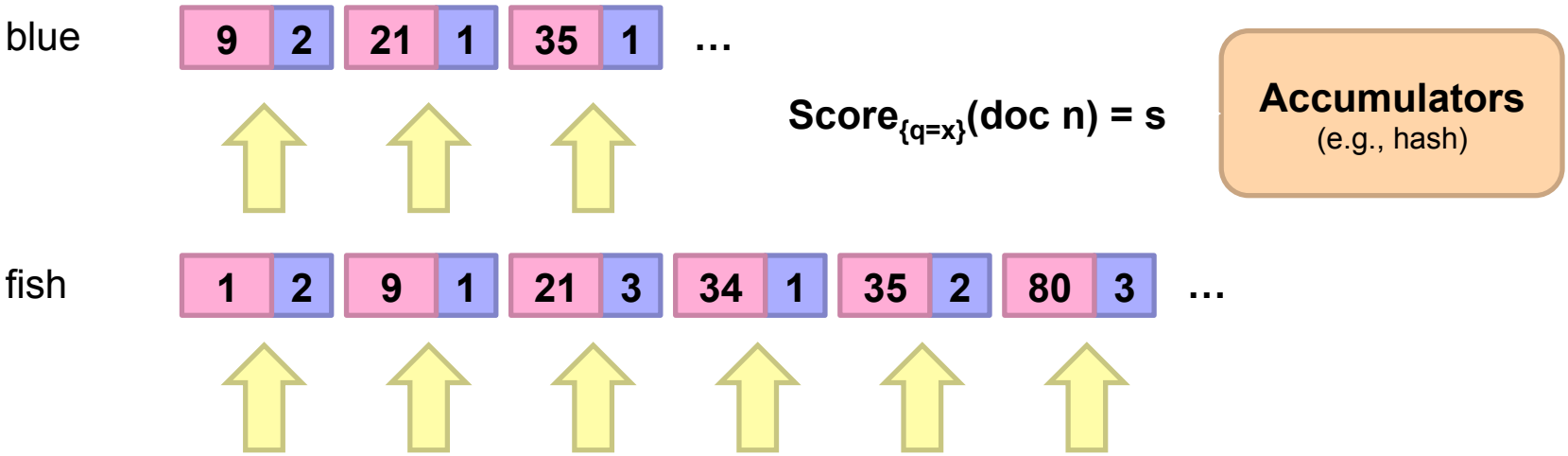
Document score in top k?

Yes: Insert document score, extract-min if queue too large
No: Do nothing

- 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)



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

Perfect for MapReduce!

Uh… not so good…

Indexing: Performance Analysis

- Fundamentally, a large sorting problem
  - Terms usually fit in memory
  - Postings usually don't
- How is it done on a single machine?
- How can it be done with MapReduce?
- First, let's characterize the problem size:
  - Size of vocabulary
  - Size of postings

Vocabulary Size: Heaps' Law

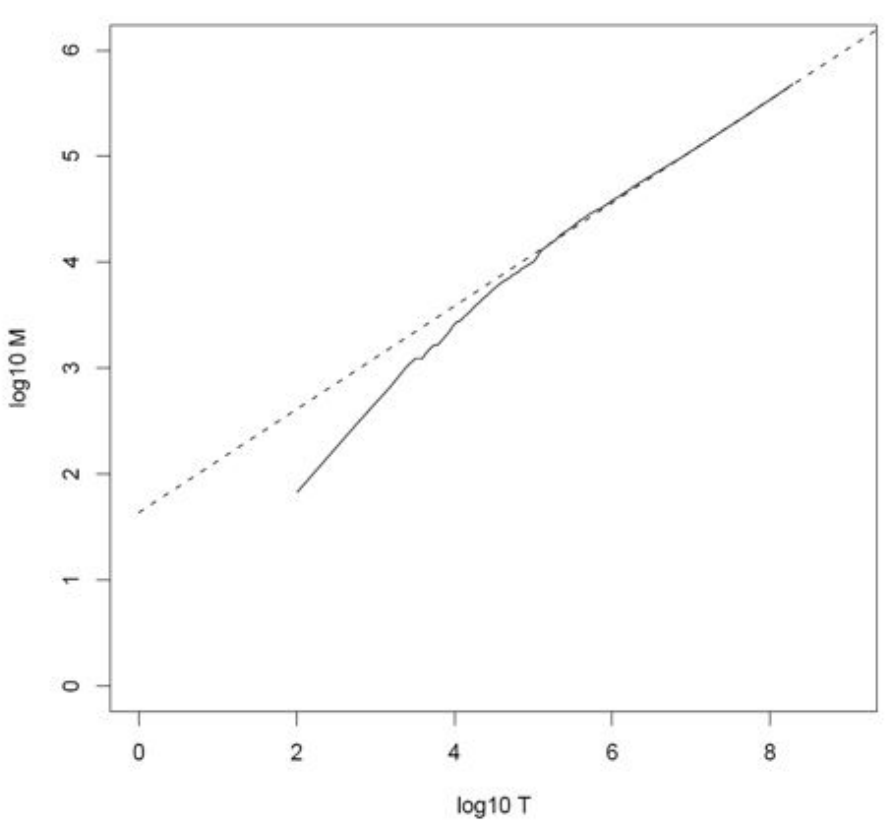
$$M = kT^b$$

M is vocabulary size
T is collection size (number of documents)
k and *b* are constants

Typically, *k* is between 30 and 100, *b* is between 0.4 and 0.6

- Heaps' Law: linear in log-log space
- Vocabulary size grows unbounded!

Heaps' Law for RCV1



k = 44
b = 0.49

First 1,000,020 terms:
Predicted = 38,323
Actual = 38,365

Reuters-RCV1 collection: 806,791 newswire documents (Aug 20, 1996-August 19, 1997)

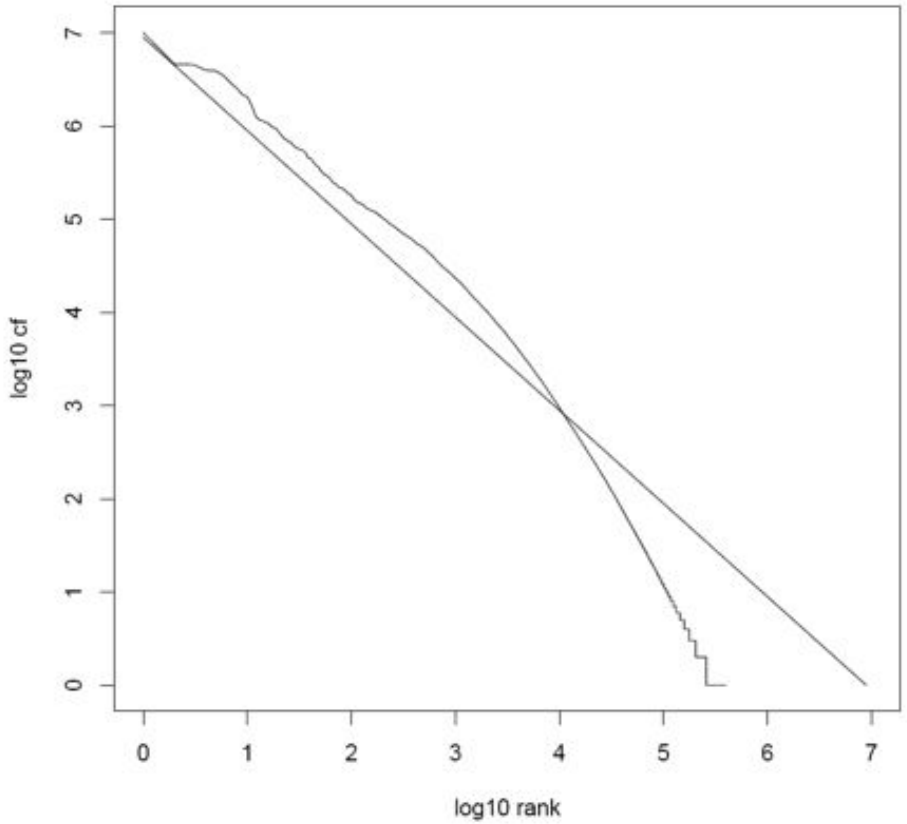
Postings Size: Zipf's Law

$$\mathrm{cf}_i = \frac{c}{i}$$

cf is the collection frequency of *i*-th common term
c is a constant

- Zipf's Law: (also) linear in log-log space
  - Specific case of Power Law distributions
- In other words:
  - A few elements occur very frequently
  - Many elements occur very infrequently

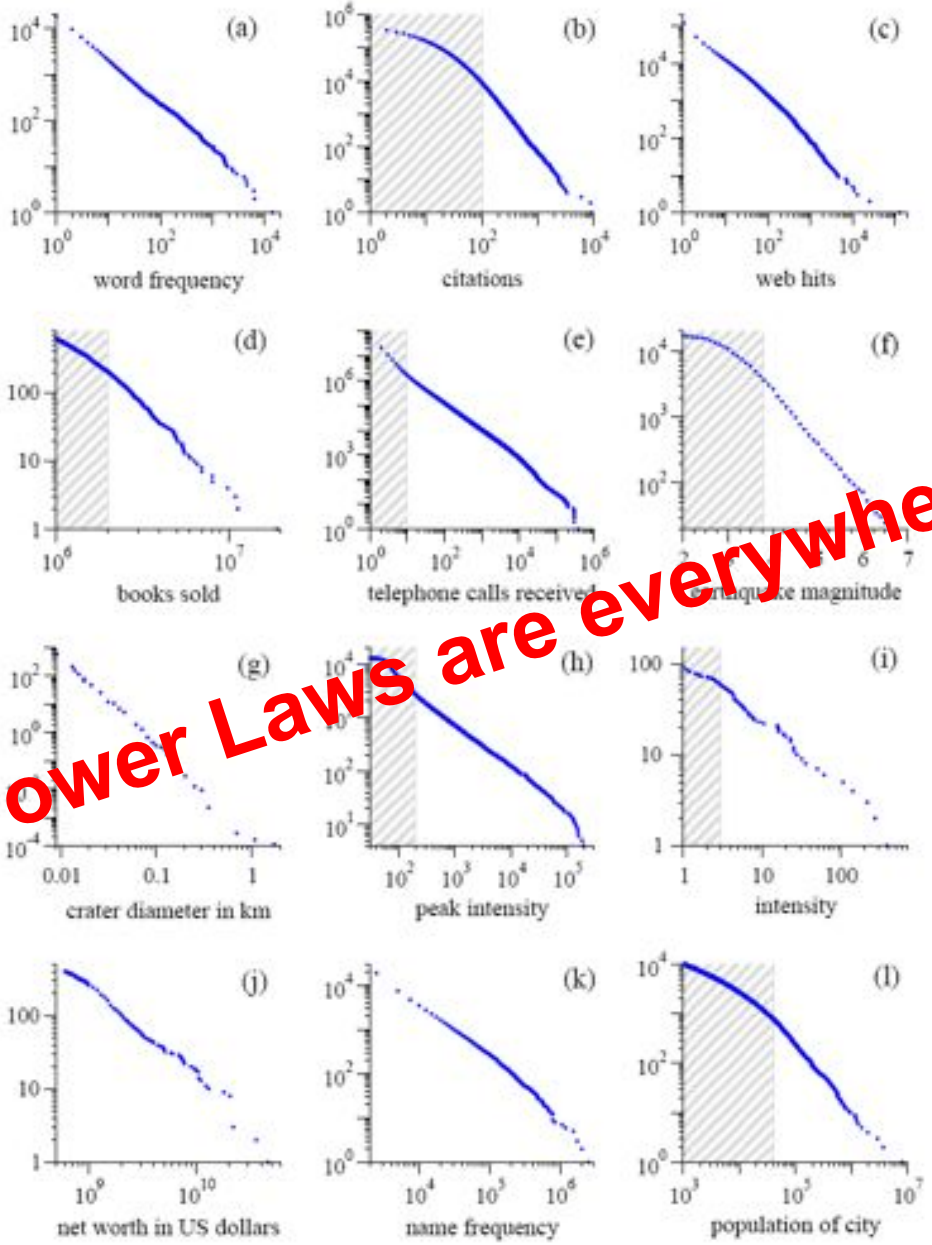
Zipf's Law for RCV1



Fit isn't that good… but good enough!

Reuters-RCV1 collection: 806,791 newswire documents (Aug 20, 1996-August 19, 1997)

Manning, Raghavan, Schütze, Introduction to Information Retrieval (2008)



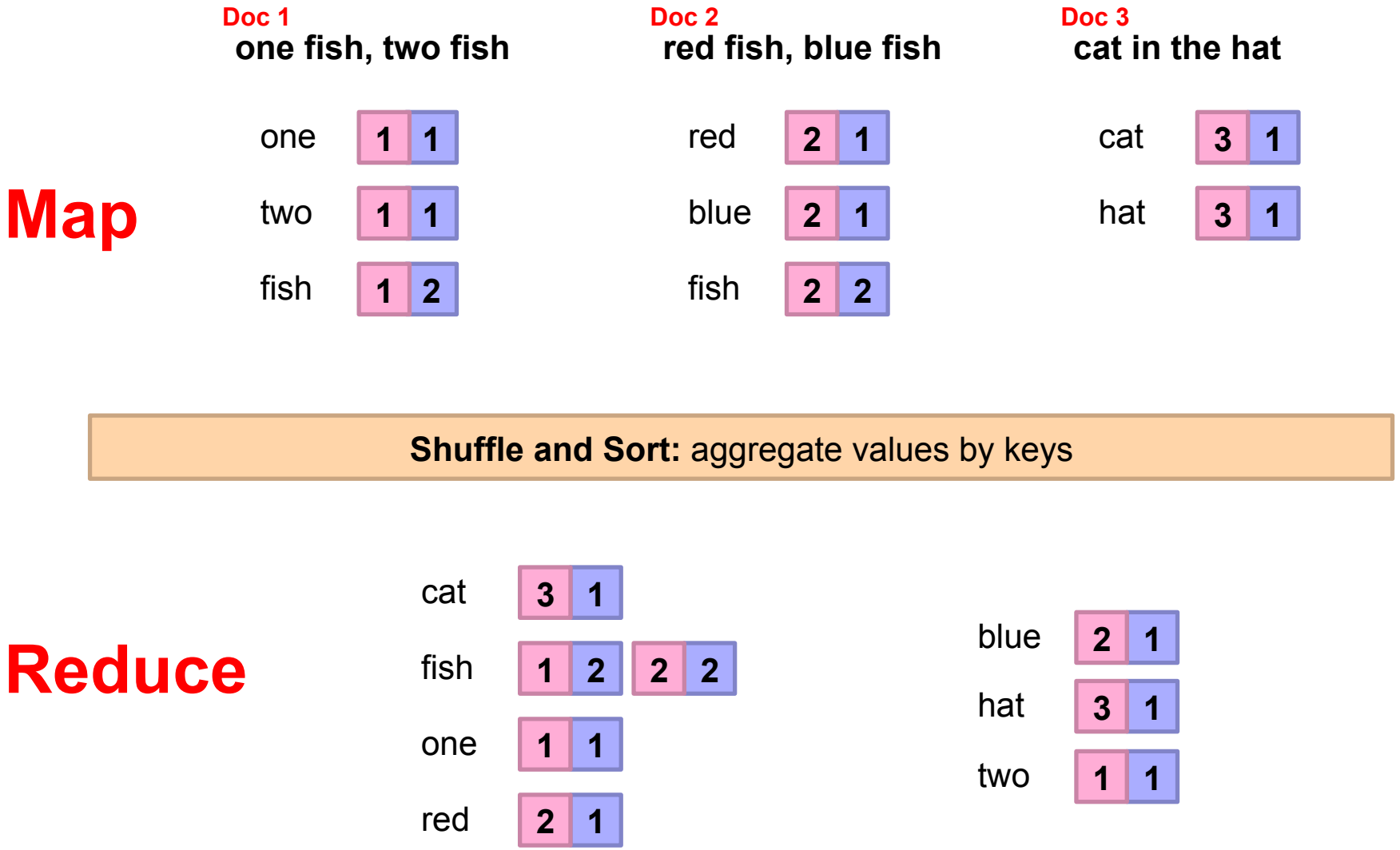
Power Laws are everywhere!

Figure from: Newman, M. E. J. (2005) “Power laws, Pareto distributions and Zipf's law.” Contemporary Physics 46:323–351.

MapReduce: Index Construction

- Map over all documents
  - Emit *term* as key, (*docno*, *tf)* as value
  - Emit other information 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!

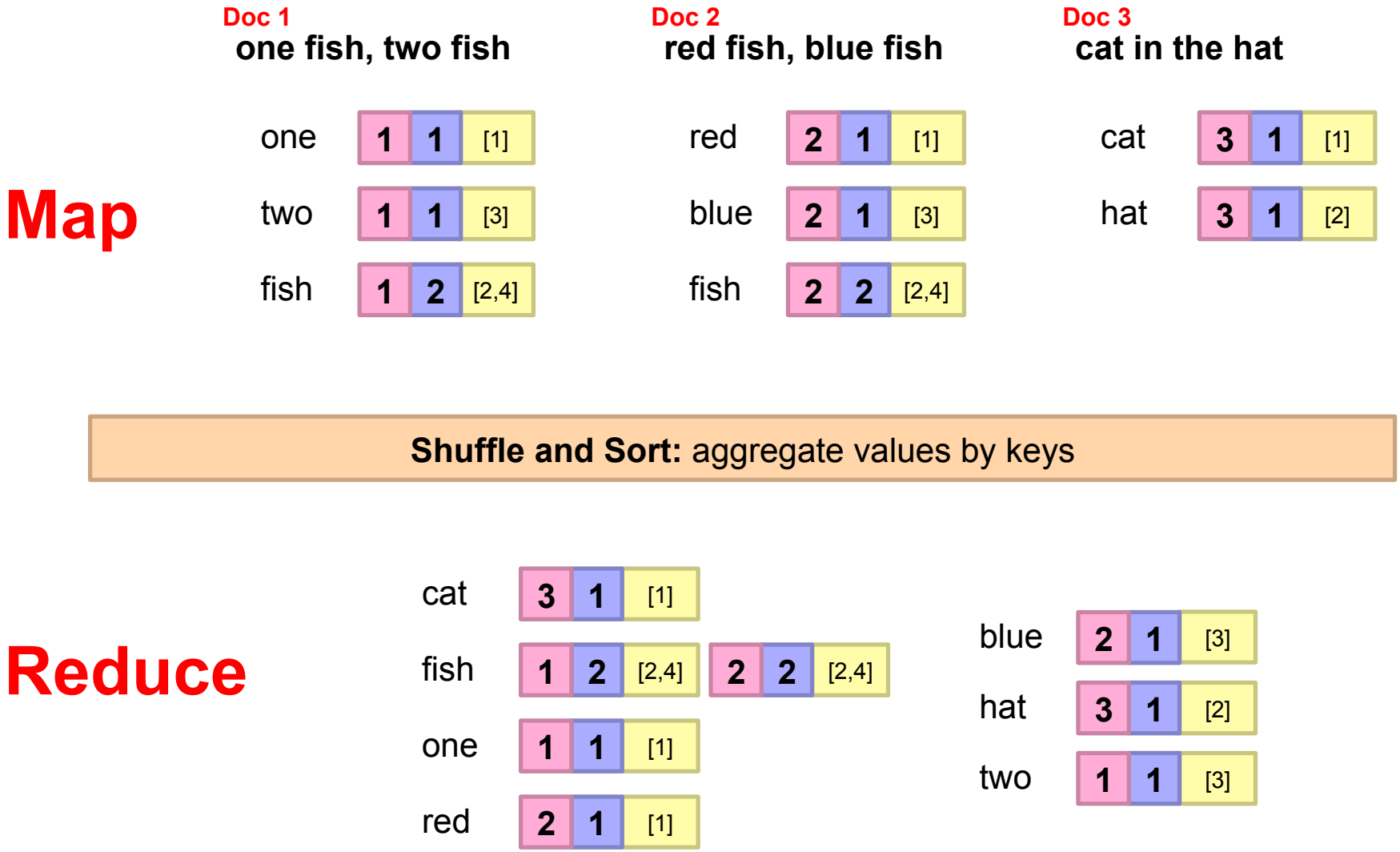
Inverted Indexing with MapReduce



Inverted Indexing: Pseudo-Code

```
1: class Mapper
2:     procedure Map(docid n, doc d)
3:         H ← new AssociativeArray
4:         for all term t ∈ doc d do
5:             H{t} ← H{t} + 1
6:         for all term t ∈ H do
7:             Emit(term t, posting ⟨n, H{t}⟩)

1: class Reducer
2:     procedure Reduce(term t, postings [⟨a1, f1⟩, ⟨a2, f2⟩ . . .])
3:         P ← new List
4:         for all posting ⟨a, f⟩ ∈ postings [⟨a1, f1⟩, ⟨a2, f2⟩ . . .] do
5:             Append(P, ⟨a, f⟩)
6:         Sort(P)
7:         Emit(term t, postings P)
```

Positional Indexes



Inverted Indexing: Pseudo-Code

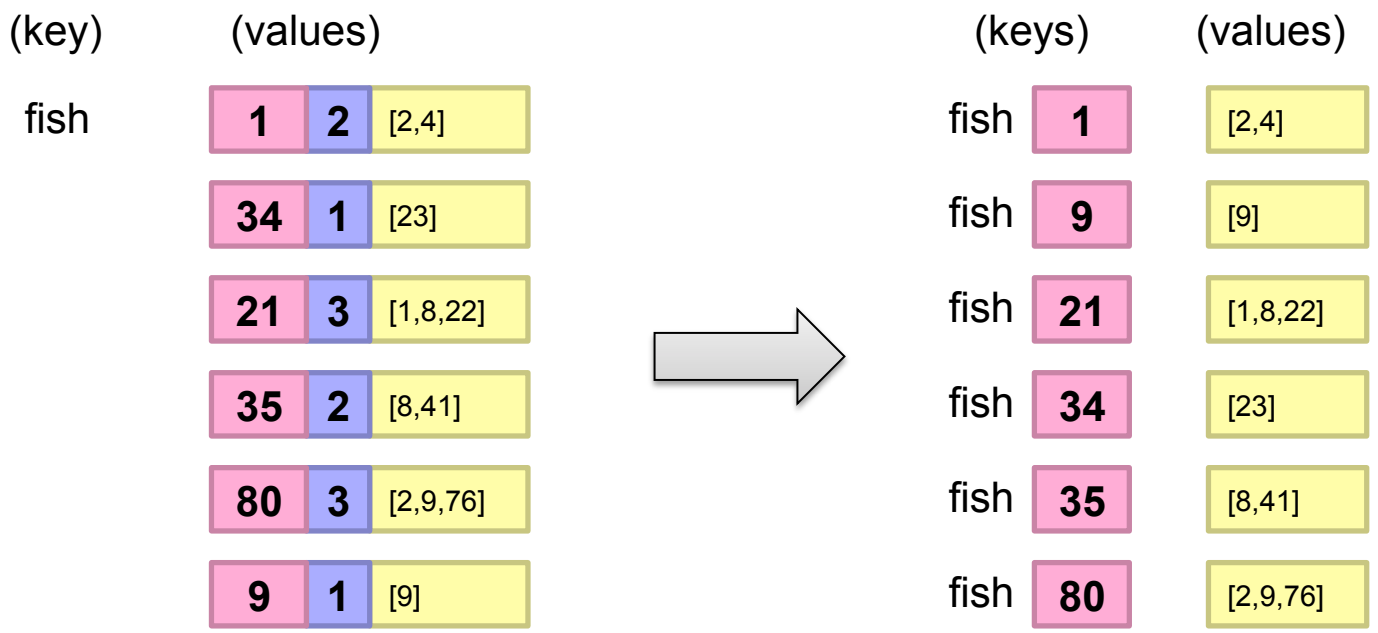
```
1: class Mapper
2:     procedure Map(docid n, doc d)
3:         H ← new AssociativeArray
4:         for all term t ∈ doc d do
5:             H{t} ← H{t} + 1
6:         for all term t ∈ H do
7:             Emit(term t, posting ⟨n, H{t}⟩)

1: class Reducer
2:     procedure Reduce(term t, postings [⟨a1, f1⟩, ⟨a2, f2⟩ . . .])
3:         P ← new List
4:         for all posting ⟨a, f⟩ ∈ postings [⟨a1, f1⟩, ⟨a2, f2⟩ . . .] do
5:             Append(P, ⟨a, f⟩)
6:         Sort(P)
7:         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?

Postings Encoding

Conceptually:

In Practice:

- Don't encode docnos, encode gaps (or *d*-gaps)
- But it's not obvious that this saves space…

Overview of Index Compression

- Byte-aligned vs. bit-aligned
- Non-parameterized bit-aligned
  - Unary codes
  - Truncated Binary
  - γ codes
  - δ codes
- Parameterized bit-aligned
  - Golomb codes (local Bernoulli model)

Want more detail? Read *Managing Gigabytes* by Witten, Moffat, and Bell!

Unary Codes

- $x \geq 1$ is coded as $x-1$ one bits, followed by 1 zero bit
  - 3 = 110
  - 4 = 1110
- Can't encode 0
- Great for small numbers… horrible for large numbers
  - Overly-biased for very small gaps

Watch out! Slightly different definitions in different textbooks

Truncated Binary

- You have pre-specified range N
- Insight: If N is not a power of 2, then you have unused codes
- If you expect smaller values to be more common, use fewer bits
- N=3 (1 unused): 0 = 0 (save a bit), 1=10 (shift by 1), 2=11 (add 1)
- N=5 (3 unused): 0 = 00 (save a bit), 1=01(save a bit), 3=110 (add 3), 4=111 (add 3)

γ codes

- $x \geq 1$ is coded in two parts: length and offset
  - Start with binary encoded, remove highest-order bit = offset
  - Length is number of binary digits, encoded in unary code
  - Concatenate length + offset codes
- Example: 9 in binary is 1001
  - Offset = 001
  - Length = 4, in unary code = 1110
  - γ code = 1110:001
- Analysis
  - Offset = $\lfloor \log x \rfloor$
  - Length = $\lfloor \log x \rfloor + 1$
  - Total = $2 \lfloor \log x \rfloor + 1$

δ codes

- Similar to γ codes, except that length is encoded in γ code
- Example: 9 in binary is 1001
  - Offset = 001
  - Length = 4, in γ code = 11000
  - δ code = 11000:001
- γ codes = more compact for smaller numbers
  δ codes = more compact for larger numbers

Golomb Codes

- $x \geq 1$, parameter b:
  - $q + 1$ in unary, where $q = \lfloor x / b \rfloor$
  - r in truncated binary, where $r = x - qb$, in $\lfloor \log b \rfloor$ or $\lceil \log b \rceil$ bits
- Example:
  - $b = 3$, $r = 0, 1, 2$ (0, 10, 11)
  - $b = 6$, $r = 0, 1, 2, 3, 4, 5$ (00, 01, 100, 101, 110, 111)
  - $x = 9$, $b = 3$: $q = 3$, $r = 0$, code = 110:0
  - $x = 9$, $b = 6$: $q = 1$, $r = 3$, code = 10:101
- Optimal $b \approx \ln(2)$ (N/df) ≈ 0.69 (N/df)
  - Different b for every term!

Comparison of Coding Schemes

	Unary	γ	δ	Golomb	
				b=3	b=6
1	0	0	0	0:0	0:00
2	10	10:0	100:0	0:10	0:01
3	110	10:1	100:1	10:0	0:100
4	1110	110:00	101:00	10:10	0:101
5	11110	110:01	101:01	10:11	0:110
6	111110	110:10	101:10	110:0	10:00
7	1111110	110:11	101:11	110:10	10:01
8	11111110	1110:000	11000:000	110:11	10:100
9	111111110	1110:001	11000:001	1110:0	10:101
10	1111111110	1110:010	11000:010	1110:10	10:110

Witten, Moffat, Bell, Managing Gigabytes (1999)

Index Compression: Performance

Comparison of Index Size (bits per pointer)

	Bible	TREC
Unary	262	1918
Binary	15	20
γ	6.51	6.63
δ	6.23	6.38
Golomb	6.09	5.84

←Recommend best practice

Bible: King James version of the Bible; 31,101 verses (4.3 MB)
TREC: TREC disks 1+2; 741,856 docs (2070 MB)

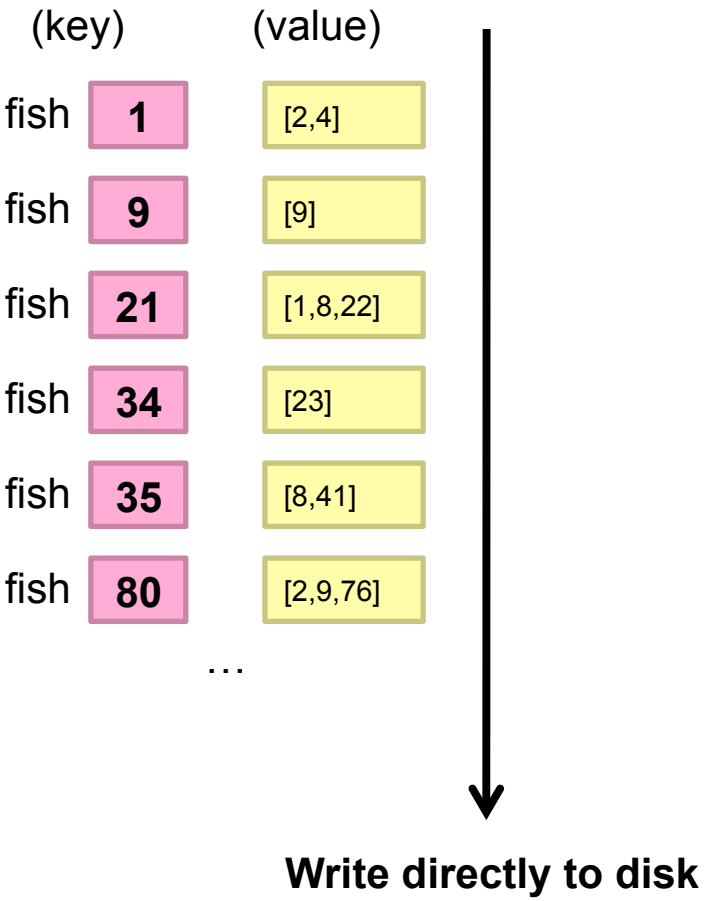
Witten, Moffat, Bell, Managing Gigabytes (1999)

Wait a minute ...

- I thought disk space was cheap
- Yes, but network bandwidth and caches are not
- More efficient representation means better throughput, more can fit in memory, less thrashing

- Still too much of a hassle? Protocol buffers do variable length encoding when serializing (but not as well)

Chicken and Egg?



But wait! How do we set the Golomb parameter *b*?

Recall: optimal $b \approx 0.69$ (N/df)

We need the *df* to set *b*…

But we don't know the *df* until we've seen all postings!

Sound familiar?

Getting the df

- In the mapper:
  - Emit “special” key-value pairs to keep track of *df*
- In the reducer:
  - Make sure “special” key-value pairs come first: process them to determine *df*
- Remember: proper partitioning!

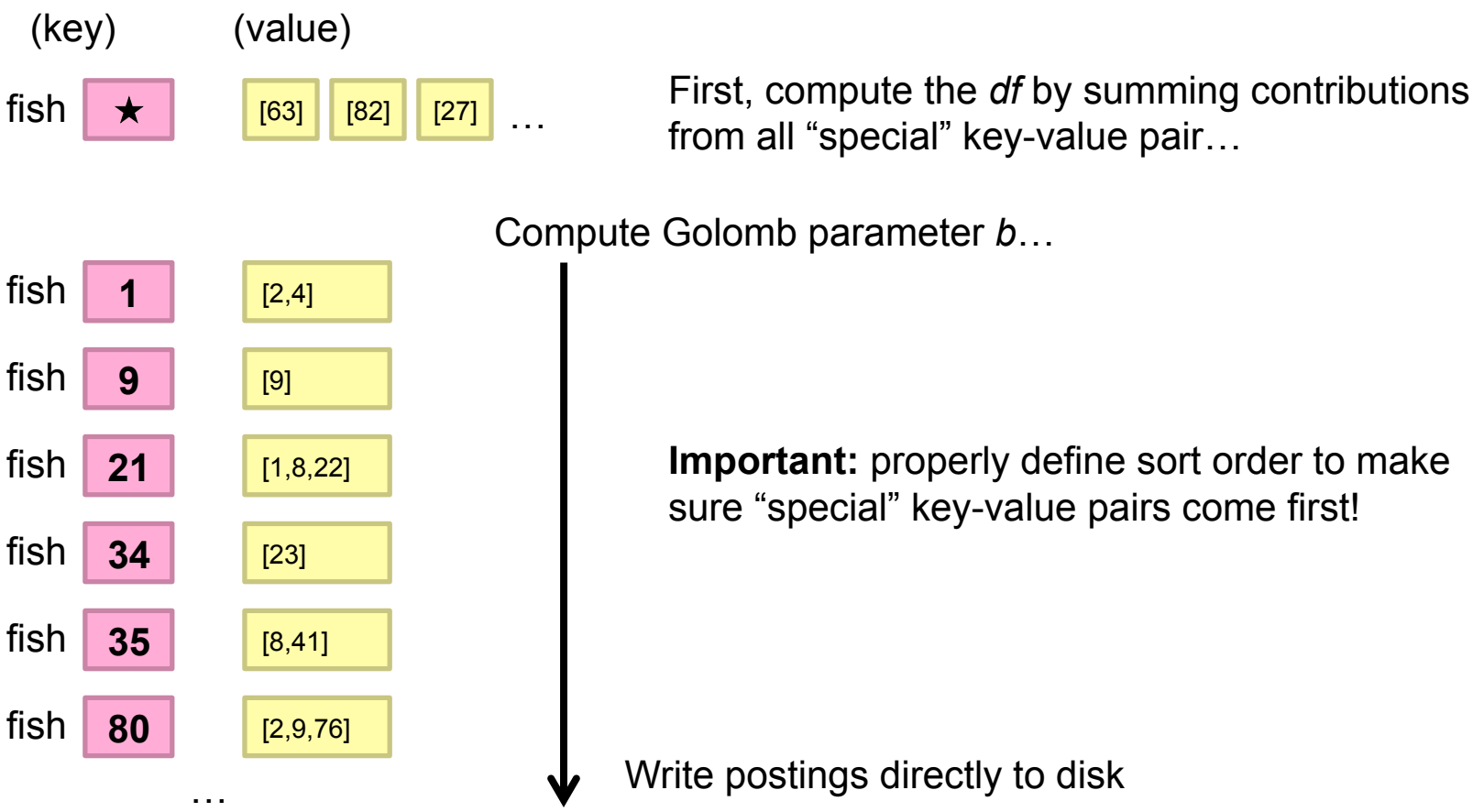
Getting the df: Modified Mapper

Doc 1
one fish, two fish

Input document…

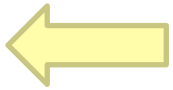
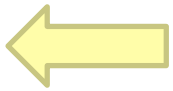
(key)		(value)	
fish	1	[2,4]	Emit normal key-value pairs…
one	1	[1]	
two	1	[3]	
fish	★	[1]	Emit “special” key-value pairs to keep track of *df*…
one	★	[1]	
two	★	[1]	

Getting the df: Modified Reducer



Where have we seen this before?

MapReduce it?

- The indexing problem  **Just covered**
  - Scalability is paramount
  - 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
- The retrieval problem  **Now**
  - Must have sub-second response time
  - For the web, only need relatively few results

Retrieval with MapReduce?

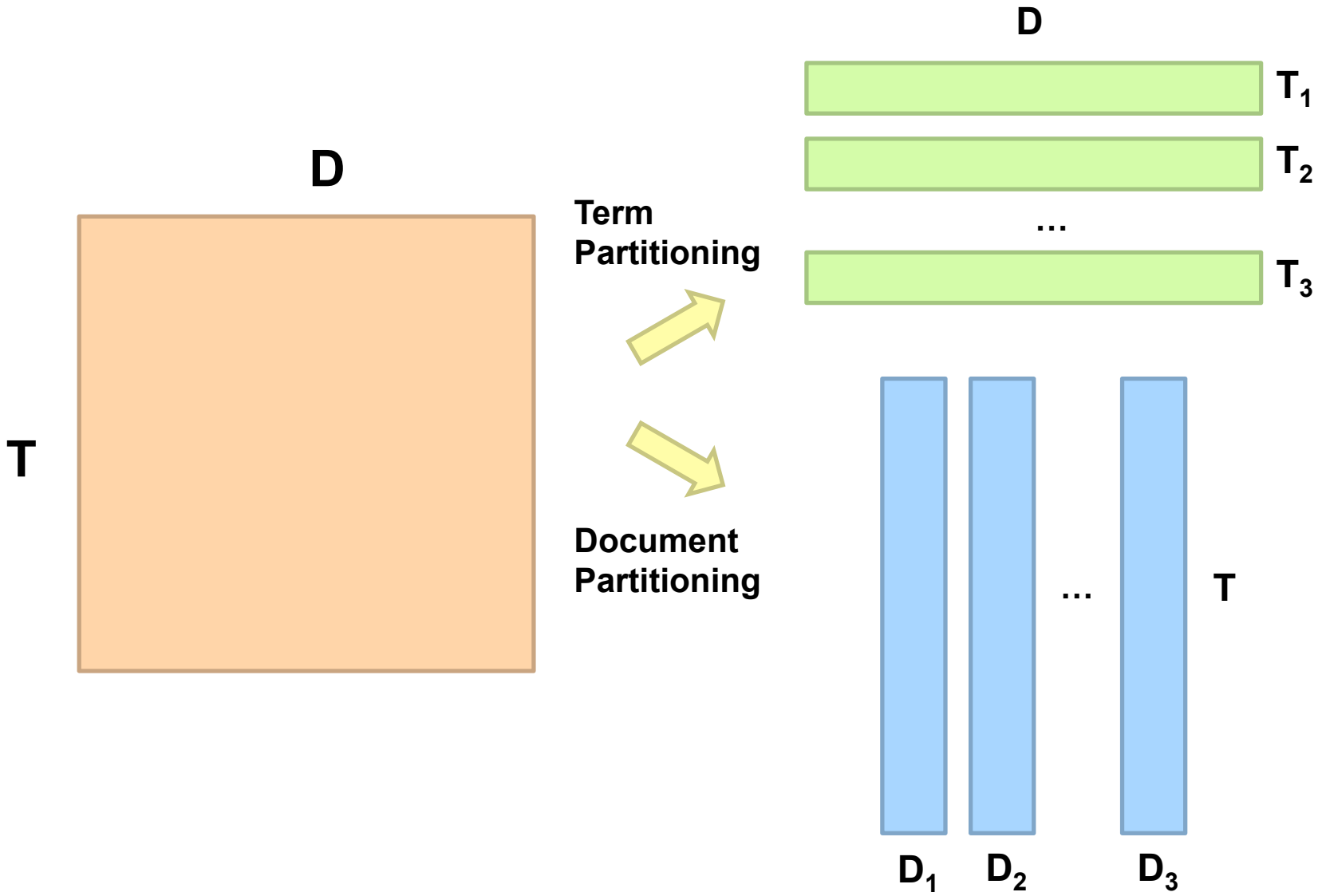
- MapReduce is fundamentally batch-oriented
  - Optimized for throughput, not latency
  - Startup of mappers and reducers is expensive
- MapReduce is not suitable for real-time queries!
  - Use separate infrastructure for retrieval…

Important Ideas

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

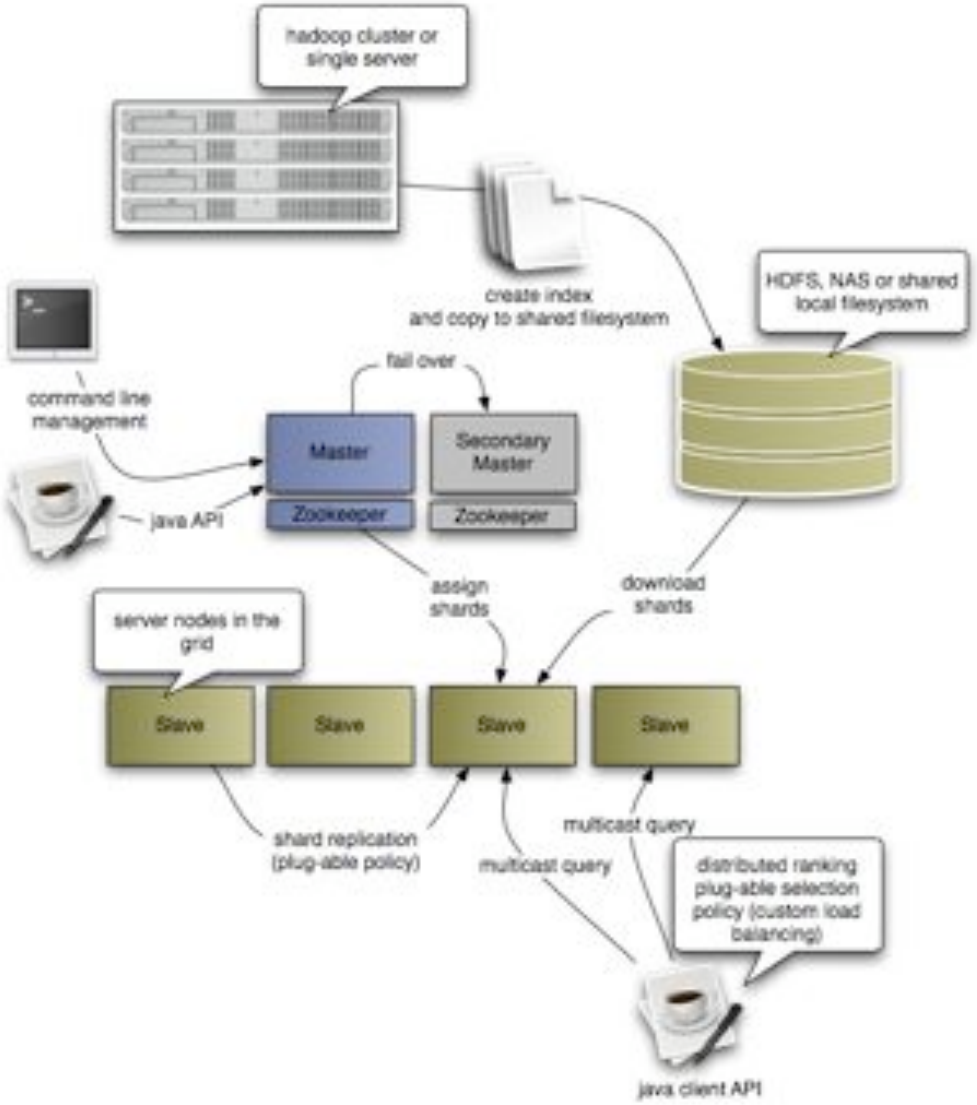
The rest is just details!

Term vs. Document Partitioning



Katta Architecture

(Distributed Lucene)



http://katta.sourceforge.net/

Streaming Dumbo

Streaming

- Lightweight way of using Hadoop
- Uses Unix pipes to communicate between any program that uses stdin / stdout
- Slower than native Java, but good for one-offs

Wordcount in Python (Streaming)

- Mapper -> Reducer

```
#!/usr/bin/env python
import sys

#--- get all lines from stdin ---
for line in sys.stdin:
    #--- remove leading and trailing whitespace---
    line = line.strip()

    #--- split the line into words ---
    words = line.split()

    #--- output tuples [word, 1] in tab-delimited format---
    for word in words:
        print '%s\t%s' % (word, "1")
```

```
#!/usr/bin/env python
import sys

last_key = None

# input comes from STDIN
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()

    # parse the input we got from mapper.py
    word, count = line.split('\t', 1)

    if word != last_key:
        if last_key:
          print '%s\t%s' % (last_key, sum)
        sum = 0

    sum += int(count)
    last_key = word
```

Streaming

- Everything is text
- Cumbersome to work with serialization
- Harder to use command line arguments
- Difficult to control sort (important!)
- Enter Dumbo ...

Dumbo

- Developed by Last.fm
- Python API
- Supports seamless serialization (duck typing)
- Supports primary and secondary keys for sorting / partitioning
- Still slower than native Java

Dumbo Word Count

- Code

```
from dumbo import *

def word_mapper(key, value):
    for word in value.split():
        yield word, 1

def runner(job):
    job.additer(word_mapper, sumreducer, combiner=sumreducer)

if __name__ == "__main__":
    main(runner)
```

- Command:
  - dumbo start wordcount.py -input combined.reviews -output tfidf –hadoop /usr/lib/hadoop

Dumbo tf-idf

- Three stages:
  - Turn documents into a bag of words keyed by document and word
  - Compute tf
  - Compute df and combine with tf to compute tf-idf
- Shows capabilities of Dumbo
  - Multiple iterations of MapReduce (see in Java next week)
  - Primary and secondary keys
  - Gracefully handling of types

Stage 1: Bag of Words

```
from dumbo import *
from math import log

def word_mapper(key, value):
    for word in value.split():
        yield (key, word), 1

def runner(job):
    job.additer(word_mapper, sumreducer, combiner=sumreducer)

if __name__ == "__main__":
    main(runner)
```

Compute tf (driver)

```
def runner(job):
    job.additer(word_mapper, sumreducer, combiner=sumreducer)
    multimapper = MultiMapper()
    multimapper.add("", doc_total_mapper)
    multimapper.add("", doc_term_mapper)
    job.additer(multimapper, TermFrequencyReducer, Combiner)

if __name__ == "__main__":
    main(runner)
```

Stage 2: Compute tf (map)

```
@primary
def doc_total_mapp      value):
    doc = key[0]
    yield doc, value

@secondary
def doc_term_ma      key, value):
    doc, word = key
    yield doc, (word, value)
```

```
class Reducer(JoinReducer):
    def __init__(self):
        self.sum = 0
    def primary(self, key, values):
        self.sum = sum(values)

class Combiner(JoinCombiner):
    def primary(self, key, values):
        yield key, sum(values)

class TermFrequencyReducer(Reducer):
    def secondary(self, key, values):
        for (doc, n) in values:
            yield key, (doc, float(n) / self.sum)
```

Why no secondary combiner?

Complete driver

```
def runner(job):
    job.additer(word_mapper, sumreducer, combiner=sumreducer)
    multimapper = MultiMapper()
    multimapper.add("", doc_total_mapper)
    multimapper.add("", doc_term_mapper)
    job.additer(multimapper, TermFrequencyReducer, Combiner)
    multimapper = MultiMapper()
    multimapper.add("", doc_freq_mapper)
    multimapper.add("", term_freq_mapper)
    job.additer(multimapper, TfIdfReducer, Combiner)
if __name__ == "__main__":
    main(runner)
```

Stage 3: Compute df (map) and tf-idf (reducer)

```
@primary
def doc_freq_[illegible]er(key, value):
    word = value[0]
    yield word, 1

@secondary
def term_freq_[illegible]pper(key, value):
    word = value[0]
    tf = value[1]
    doc = key
    yield word, (doc, tf)
```

```
class Reducer(JoinReducer):
    def __init__(self):
        self.sum = 0
    def primary(self, key, values):
        self.sum = sum(values)

class TfIdfReducer(Reducer):
    def __init__(self):
        Reducer.__init__(self)
        D = self.params["doccount"]
        self.doccount = float(D)

    def secondary(self, key, values):
        idf = log(self.doccount / self.sum)
        for (doc, tf) in values:
            yield (key, doc), tf * idf
```

What is the primary key doing?
Where does D come from?

Invocation

- dumbo start tfidf.py -input combined.reviews -output tfidf -param doccount=10 –hadoop /usr/lib/hadoop

Recap

- Information Retrieval
- Document Representation
- Representing Integers
- Dumbo, Python, and other ways of using Hadoop
- First glimpse of more complicated workflows

Source: Wikipedia (Japanese rock garden)