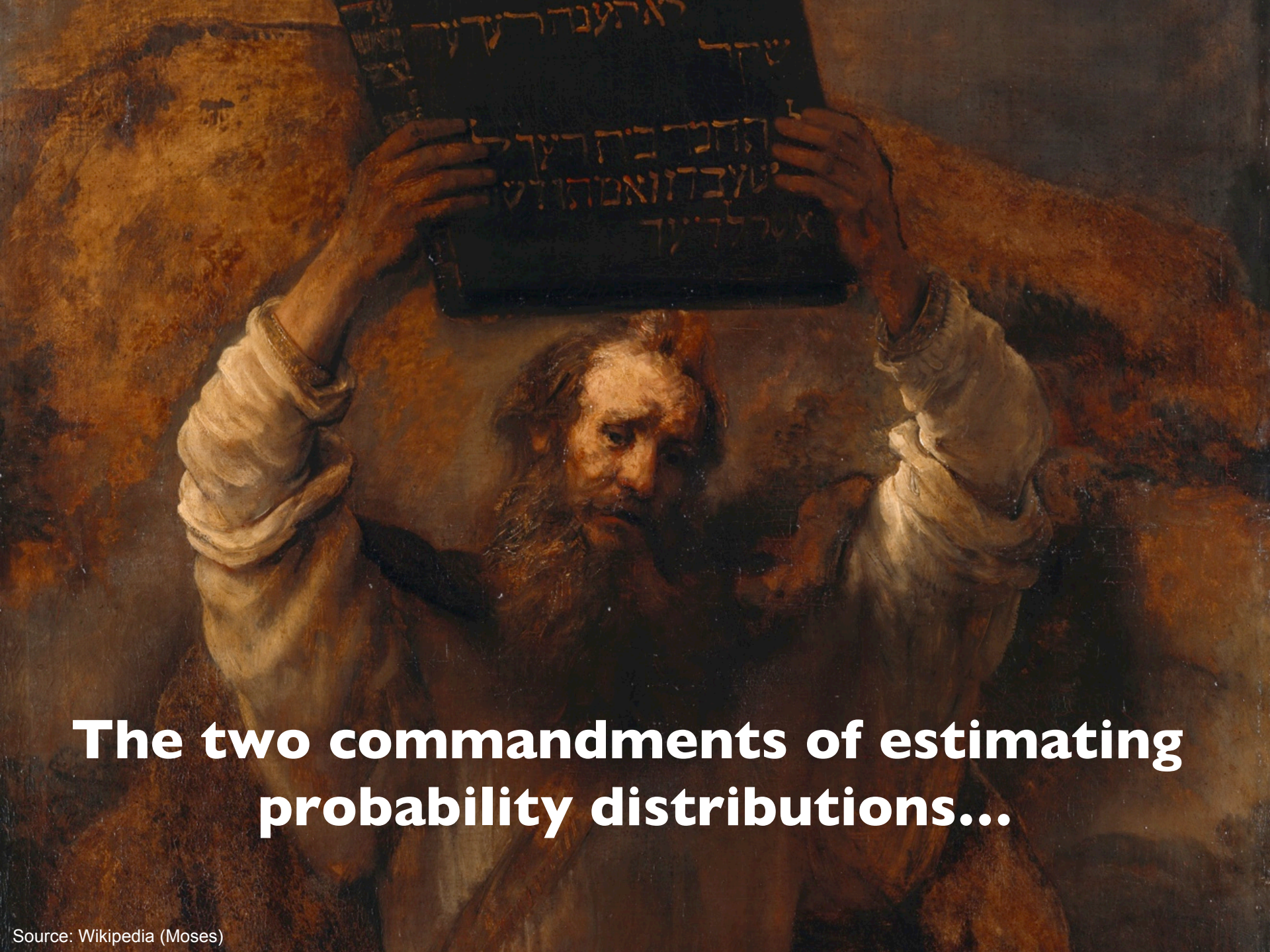
Data-Intensive Computing with MapReduce

Session 4: Language Models and Inverted Indexing

Jimmy Lin
University of Maryland
Thursday, February 14, 2013

Today's Agenda

- Real-world word counting: language models
  - How to break all the rules and get away with it
- Crash course on information retrieval
  - Basics of indexing and retrieval
  - Inverted indexing in MapReduce



The two commandments of estimating probability distributions…

Source: Wikipedia (Moses)

Probabilities must sum up to one

Source: http://www.flickr.com/photos/37680518@N03/7746322384/

Source: http://www.flickr.com/photos/brettmorrison/3732910565/

Source: Warner Bros. Pictures

Source: Google

Source: http://www.flickr.com/photos/guvnah/7861418602/

Count.

```
1: class Mapper
2:     method Map(docid a, doc d)
3:         for all term t ∈ doc d do
4:             Emit(term t, count 1)
```

```
1: class Reducer
2:     method Reduce(term t, counts [c1, c2, ...])
3:         sum ← 0
4:         for all count c ∈ counts [c1, c2, ...] do
5:             sum ← sum + c
6:         Emit(term t, count s)
```

What's the non-toy application of word count?

Language Models

$$P(w_1, w_2, \ldots, w_T)$$

$$= P(w_1)P(w_2|w_1)P(w_3|w_1, w_2) \ldots P(w_T|w_1, \ldots, w_{T-1})$$

[chain rule]

Is this tractable?

Approximating Probabilities

Basic idea: limit history to fixed number of words *N* (Markov Assumption)

$$P(w_k|w_1,\ldots,w_{k-1}) \approx P(w_k|w_{k-N+1},\ldots,w_{k-1})$$

N=1: Unigram Language Model

$$P(w_k|w_1,\ldots,w_{k-1}) \approx P(w_k)$$

$$\Rightarrow P(w_1,w_2,\ldots,w_T) \approx P(w_1)P(w_2)\ldots P(w_T)$$

Approximating Probabilities

Basic idea: limit history to fixed number of words *N* (Markov Assumption)

$$P(w_k|w_1,\ldots,w_{k-1}) \approx P(w_k|w_{k-N+1},\ldots,w_{k-1})$$

N=2: Bigram Language Model

$$P(w_k|w_1,\ldots,w_{k-1}) \approx P(w_k|w_{k-1})$$

$$\Rightarrow P(w_1,w_2,\ldots,w_T) \approx P(w_1|<\mathrm{S}>)P(w_2|w_1)\ldots P(w_T|w_{T-1})$$

Approximating Probabilities

Basic idea: limit history to fixed number of words *N* (Markov Assumption)

$$P(w_k|w_1,\ldots,w_{k-1}) \approx P(w_k|w_{k-N+1},\ldots,w_{k-1})$$

N=3: Trigram Language Model

$$P(w_k|w_1,\ldots,w_{k-1}) \approx P(w_k|w_{k-2},w_{k-1})$$

$$\Rightarrow P(w_1,w_2,\ldots,w_T) \approx P(w_1|<\mathrm{S}><\mathrm{S}>)\ldots P(w_T|w_{T-2}w_{T-1})$$

Building *N*-Gram Language Models

- Compute maximum likelihood estimates (MLE) for individual *n*-gram probabilities
  - Unigram: $P(w_i) = \frac{C(w_i)}{N}$
  - Bigram: $P(w_i, w_j) = \frac{C(w_i, w_j)}{N}$

    $$P(w_j|w_i) = \frac{P(w_i, w_j)}{P(w_i)} = \frac{C(w_i, w_j)}{\sum_w C(w_i, w)} = \frac{C(w_i, w_j)}{C(w_i)}$$

  - Generalizes to higher-order *n*-grams
- We already know how to do this in MapReduce!

Thou shalt smooth!

- Zeros are bad for any statistical estimator
  - Need better estimators because MLEs give us a lot of zeros
  - A distribution without zeros is "smoother"
- The Robin Hood Philosophy: Take from the rich (seen *n*-grams) and give to the poor (unseen *n*-grams)
  - And thus also called discounting
  - Make sure you still have a valid probability distribution!
- Lots of techniques:
  - Laplace, Good-Turing, Katz backoff, Jelinek-Mercer
  - Kneser-Ney represents best practice

Stupid Backoff

- Let's break all the rules:

$$S(w_i|w_{i-k+1}^{i-1}) = \begin{cases} \frac{f(w_{i-k+1}^{i})}{f(w_{i-k+1}^{i-1})} & \text{if } f(w_{i-k+1}^{i}) > 0 \\ \alpha S(w_i|w_{i-k+2}^{i-1}) & \text{otherwise} \end{cases}$$

$$S(w_i) = \frac{f(w_i)}{N}$$

- But throw *lots* of data at the problem!

Source: Brants et al. (EMNLP 2007)

Stupid Backoff Implementation

- Same basic idea as "pairs" approach discussed previously
- A few optimizations:
  - Convert words to integers, ordered by frequency (take advantage of VByte compression)
  - Replicate unigram counts to all shards

Stupid Backoff Implementation

- Straightforward approach: count each order separately

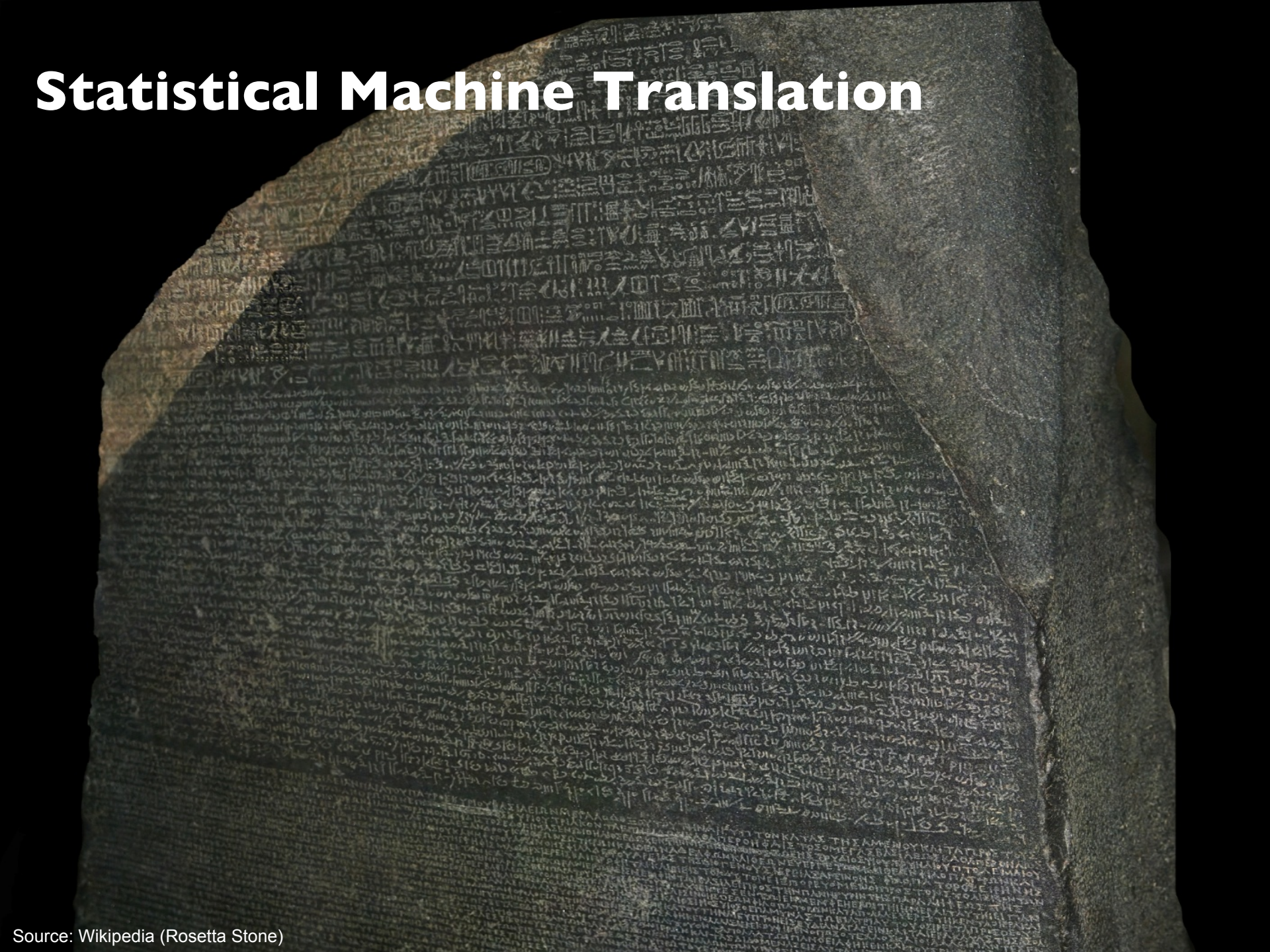
A B ← remember this value
A B C
A B D
A B E
…

- More clever approach: count *all* orders together

A B ← remember this value
A B C ← remember this value
A B C P
A B C Q
A B D ← remember this value
A B D X
A B D Y
…

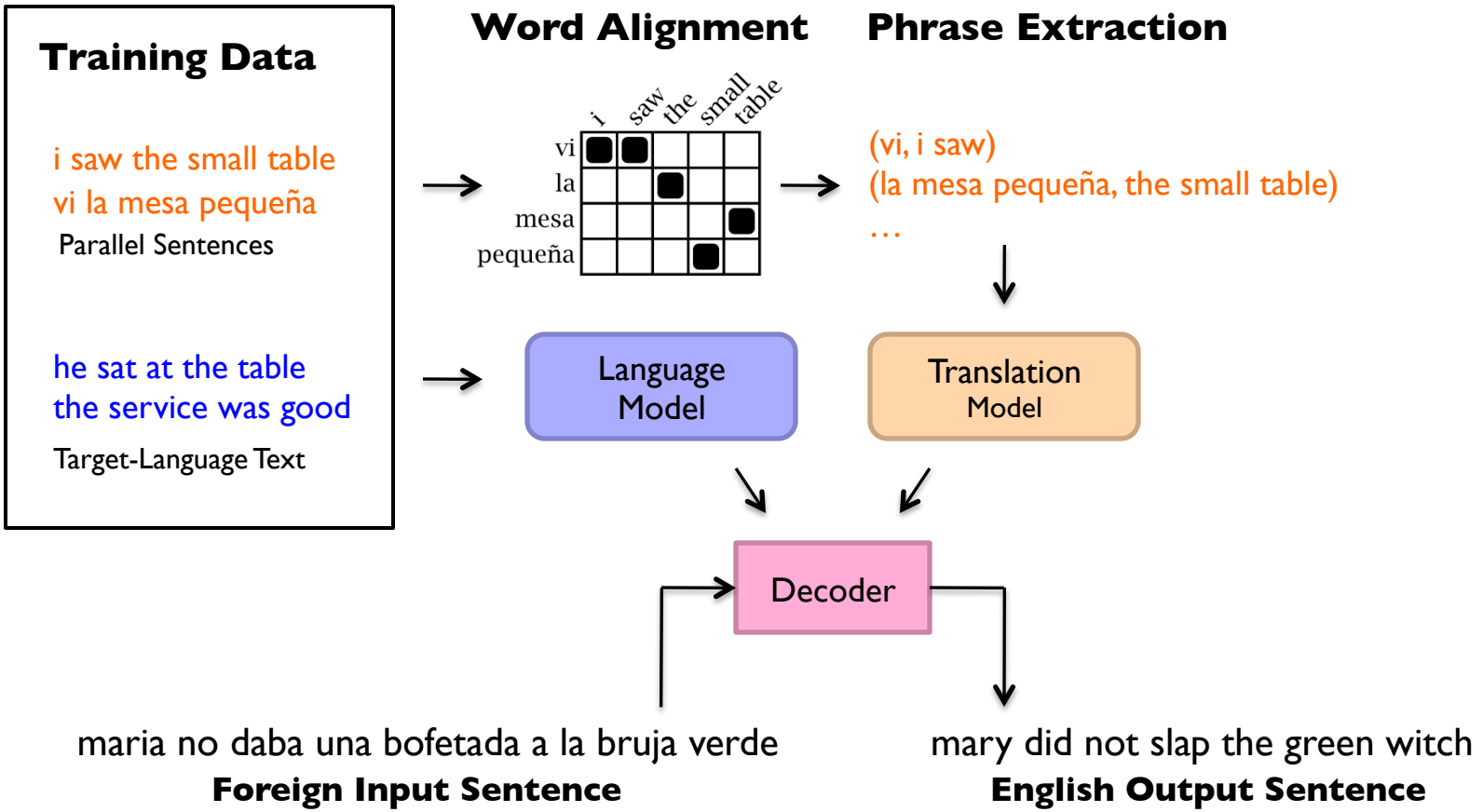

State of the art smoothing (less data)
vs. Count and normalize (more data)
EVERLAST
Source: Wikipedia (Boxing)

Statistical Machine Translation



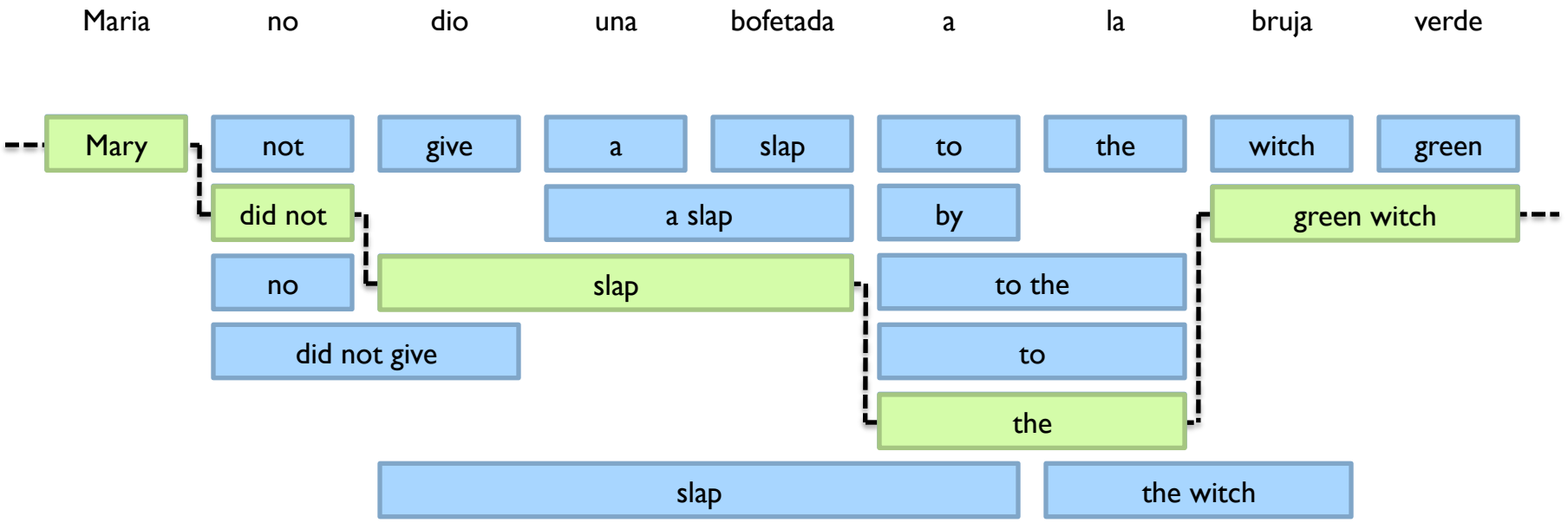
Source: Wikipedia (Rosetta Stone)

Statistical Machine Translation



$$\hat{e}_1^I = \arg\max_{e_1^I} \left[P(e_1^I \mid f_1^J) \right] = \arg\max_{e_1^I} \left[P(e_1^I) P(f_1^J \mid e_1^I) \right]$$

Translation as a Tiling Problem



$$\hat{e}_1^I = \arg\max_{e_1^I} \left[P(e_1^I \mid f_1^J) \right] = \arg\max_{e_1^I} \left[P(e_1^I) P(f_1^J \mid e_1^I) \right]$$

$$P(e|f) = \frac{P(e) \cdot P(f|e)}{P(f)}$$

$$\hat{e} = \arg\max_{e} P(e)P(f|e)$$

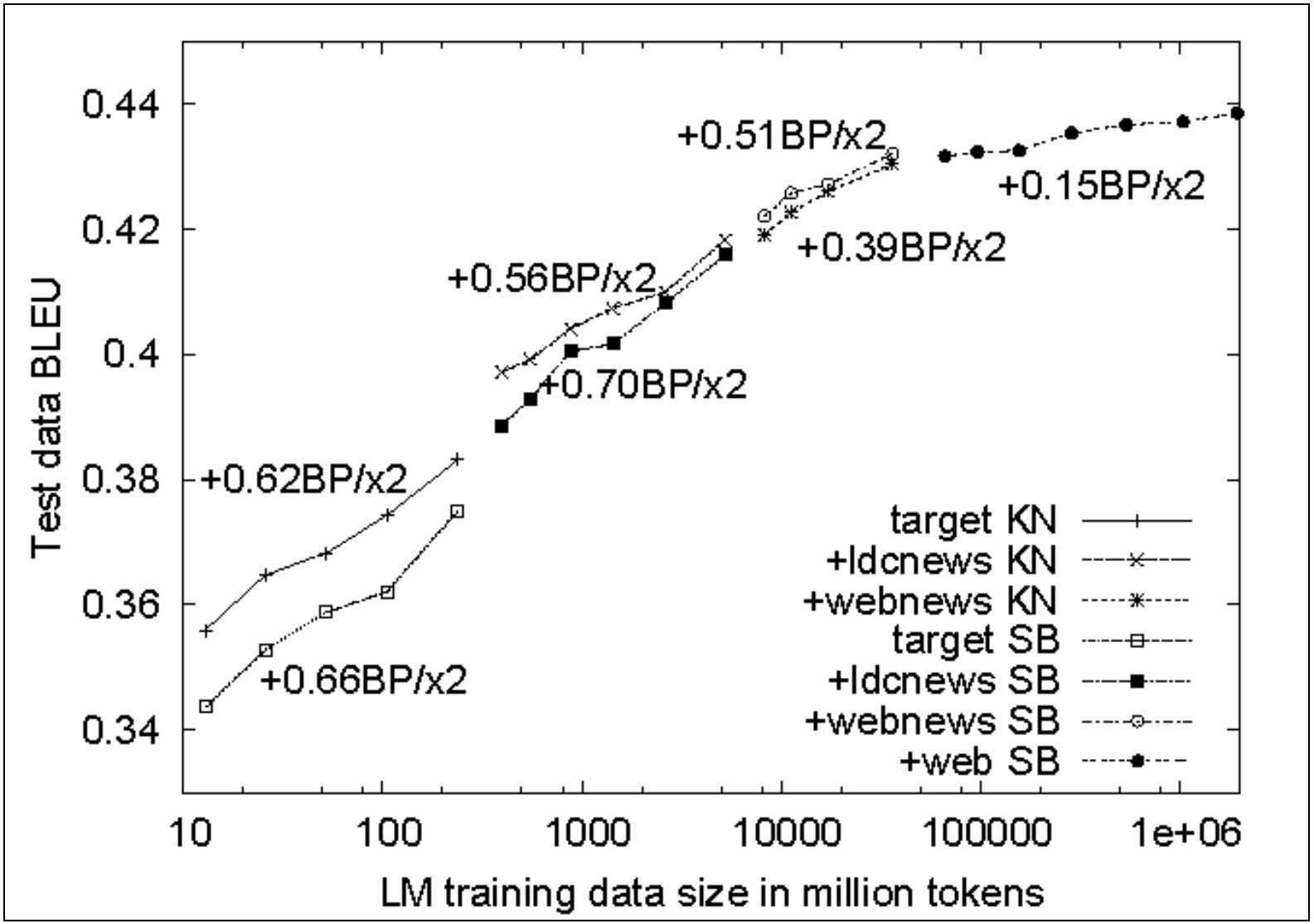
Source: http://www.flickr.com/photos/johnmueller/3814846567/in/pool-56226199@N00/

Results: Running Time

	target	*webnews*	*web*
# tokens	237M	31G	1.8T
vocab size	200k	5M	16M
# n-grams	257M	21G	300G
LM size (SB)	2G	89G	1.8T
time (SB)	20 min	8 hours	1 day
time (KN)	2.5 hours	2 days	–
# machines	100	400	1500

Source: Brants et al. (EMNLP 2007)

Results: Translation Quality



Source: Brants et al. (EMNLP 2007)

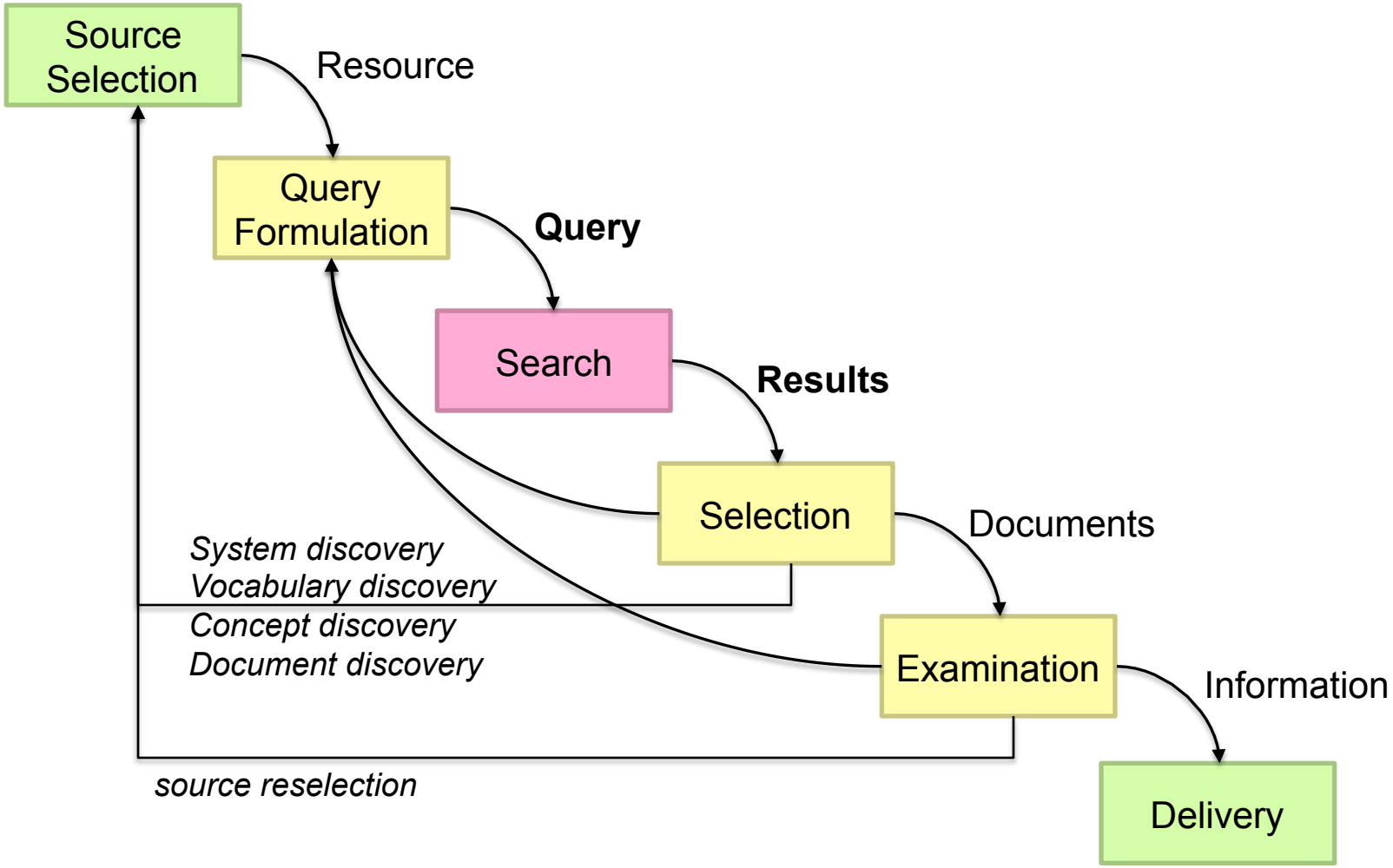
Today's Agenda

- Real-world word counting: language models
  - How to break all the rules and get away with it
- Crash course on information retrieval
  - Basics of indexing and retrieval
  - Inverted indexing in MapReduce

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

Author

Concepts

↓

Query Terms

"tragic love story"

Concepts

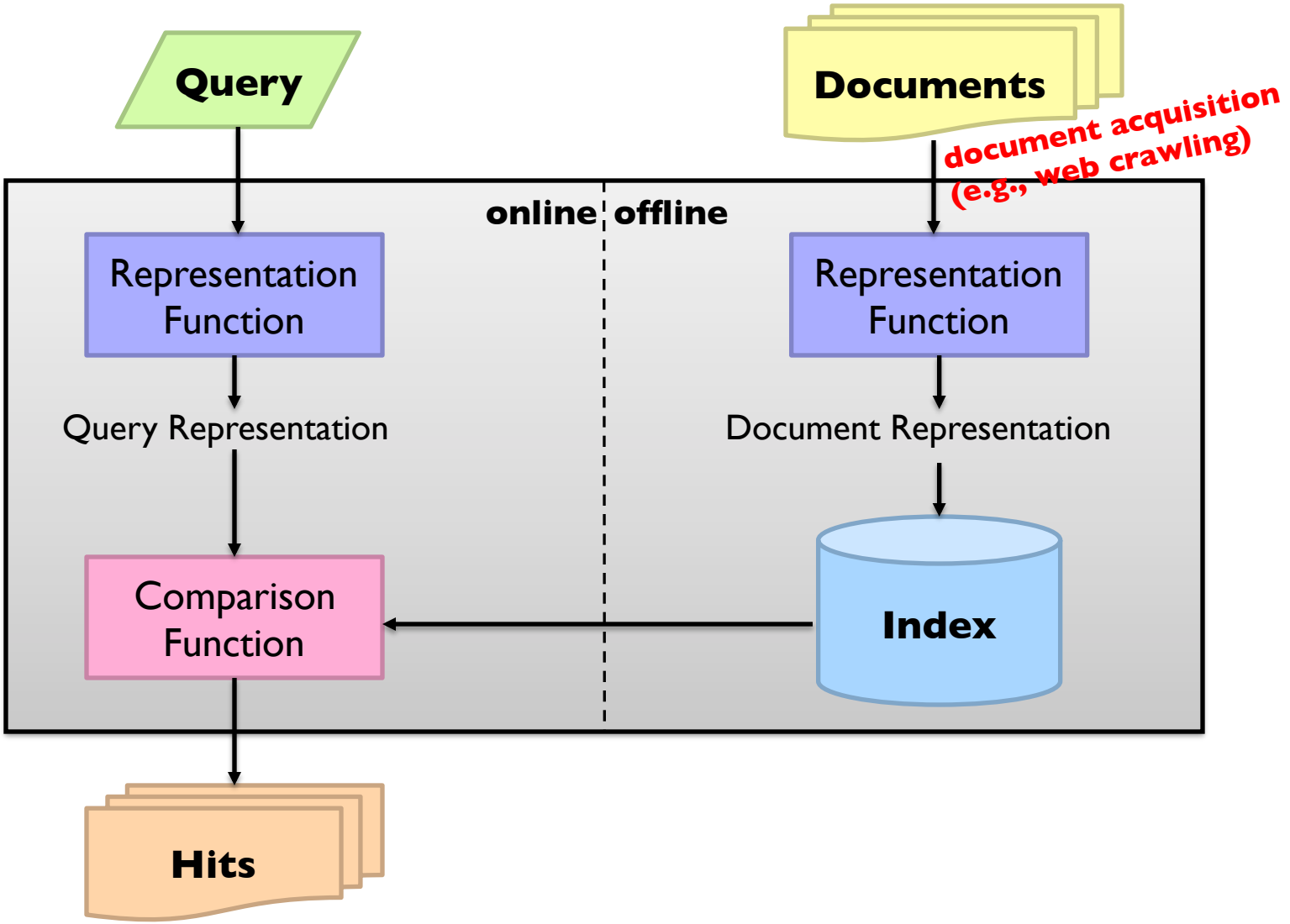
↓

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。

وق ال مارك ريجيري - في الناطق باسم
الخارجية الإسرائيلية - إن شارون قبل
الدعوة وسيقوم للمرة الأولى بزيارة
تونس، التي كانت لفترة طويلة مقر
الرئيس الفلسطيني الراحل ياسر عرفات
منذ خروجه 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.

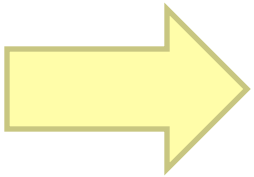
NEW YORK (CNN/Money) - McDonald's Corp. is cutting the amount of "bad" fat in its french fries nearly in half, the fast-food chain said Tuesday as it moves to make all its fried menu items healthier.

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

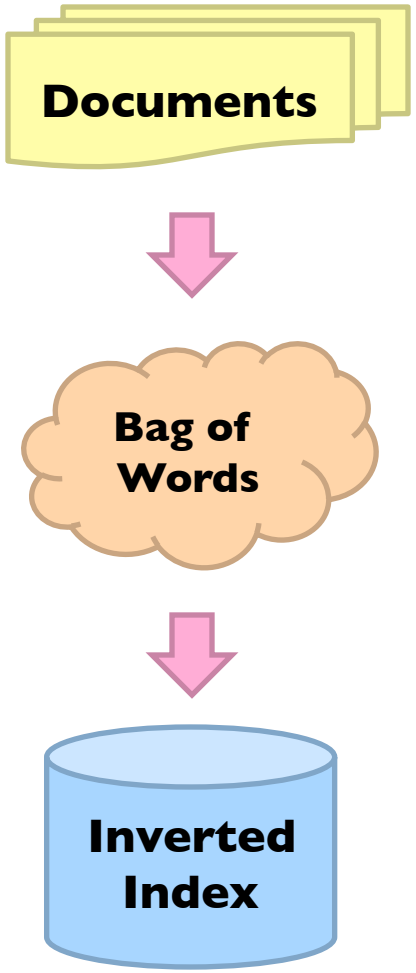
7 × french

6 × company, said, nutrition

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

…

Counting Words…



case folding, tokenization, stopword removal, stemming

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

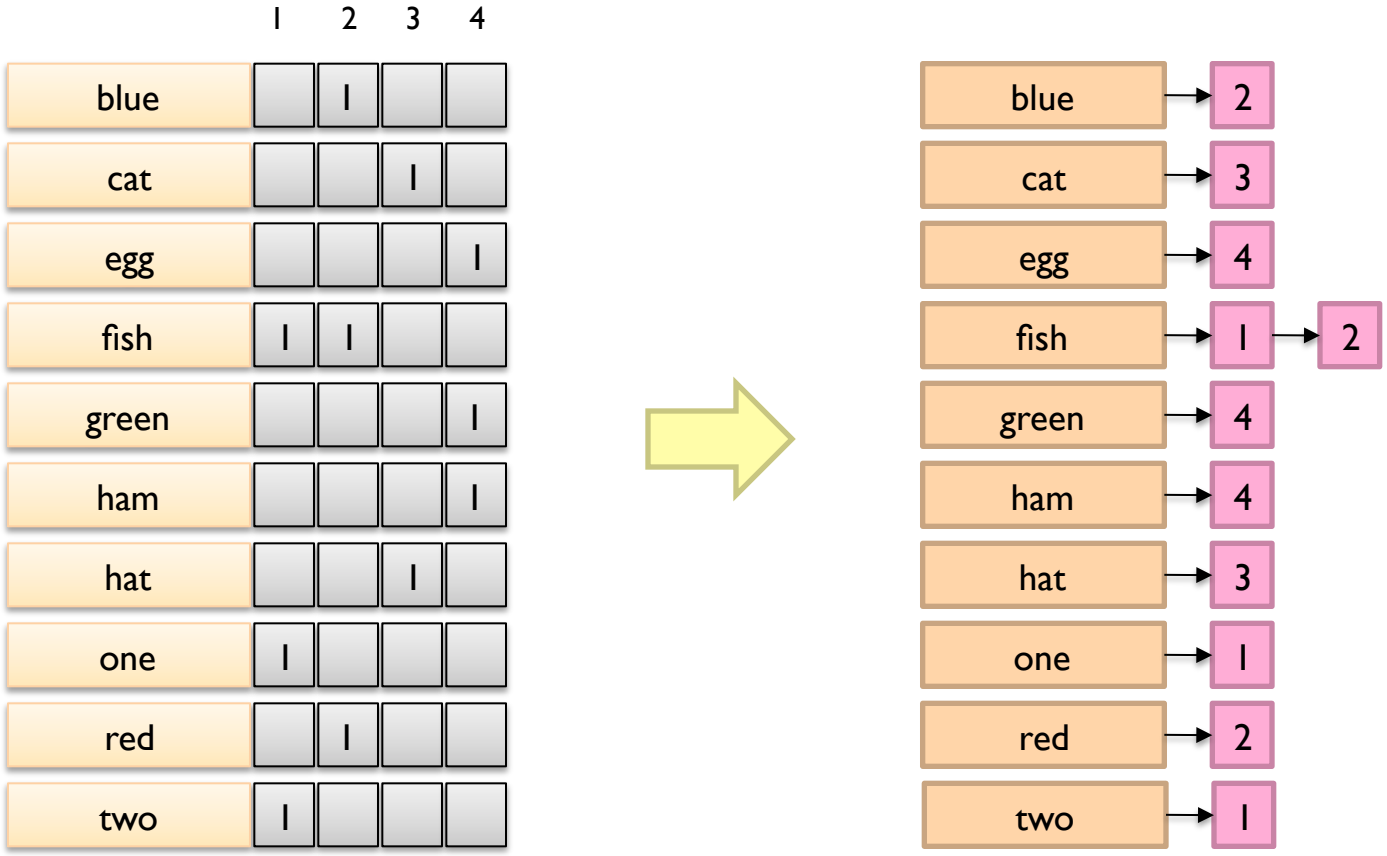
Inverted Index: Boolean Retrieval

Doc 1
one fish, two fish

Doc 2
red fish, blue fish

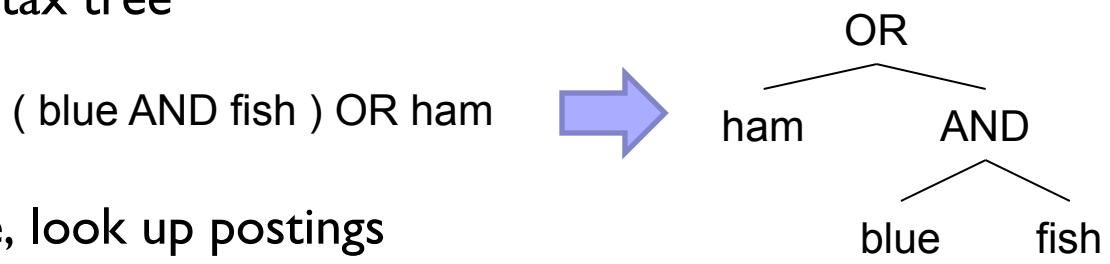
Doc 3
cat in the hat

Doc 4
green eggs and ham

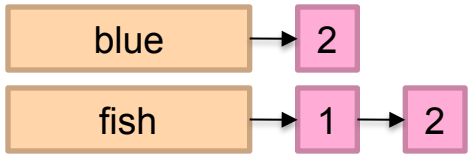


Boolean Retrieval

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



  - 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

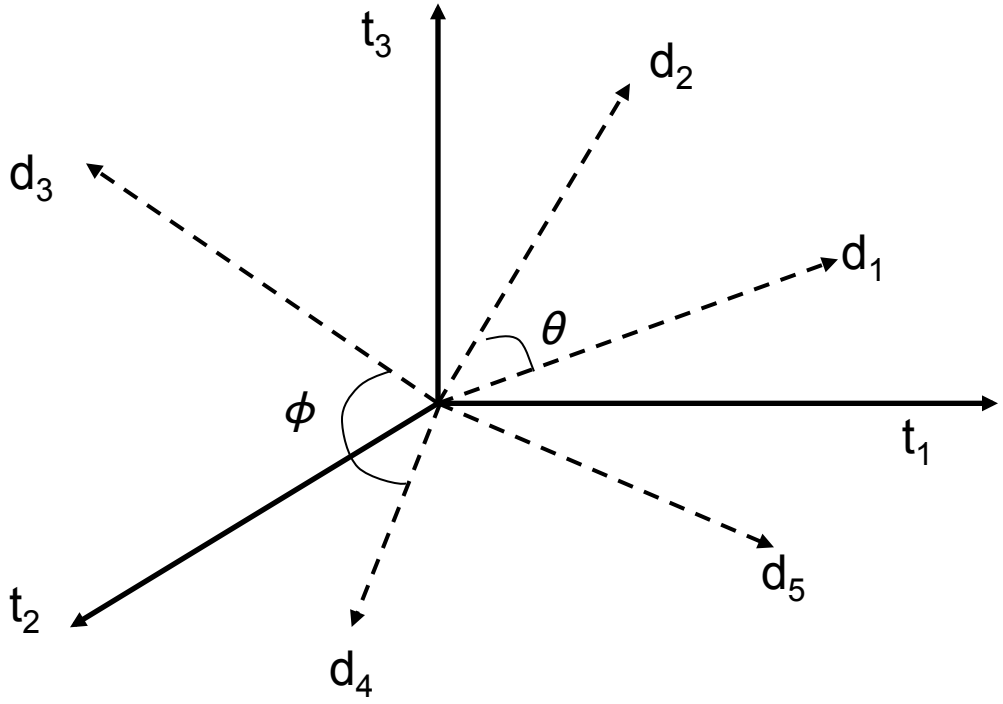
Strengths and Weaknesses

- 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
  - 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:

$$d_j = [w_{j,1}, w_{j,2}, w_{j,3}, \ldots w_{j,n}]$$
$$d_k = [w_{k,1}, w_{k,2}, w_{k,3}, \ldots w_{k,n}]$$

$$\cos\theta = \frac{d_j \cdot d_k}{|d_j||d_k|}$$

$$\text{sim}(d_j, d_k) = \frac{d_j \cdot d_k}{|d_j||d_k|} = \frac{\sum_{i=0}^{n} w_{j,i} w_{k,i}}{\sqrt{\sum_{i=0}^{n} w_{j,i}^2} \sqrt{\sum_{i=0}^{n} w_{k,i}^2}}$$

- Or, more generally, inner products:

$$\text{sim}(d_j, d_k) = d_j \cdot d_k = \sum_{i=0}^{n} w_{j,i} w_{k,i}$$

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*

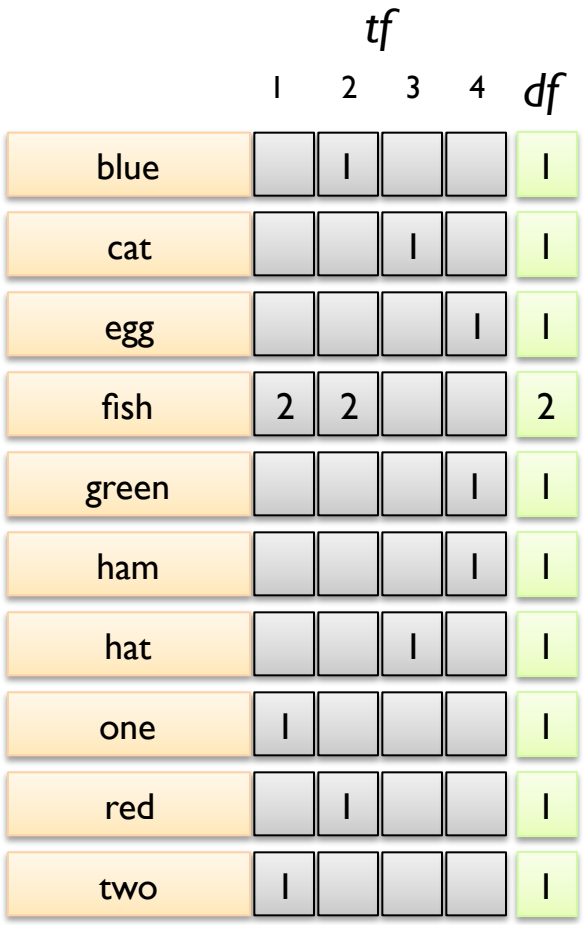
Inverted Index: TF.IDF

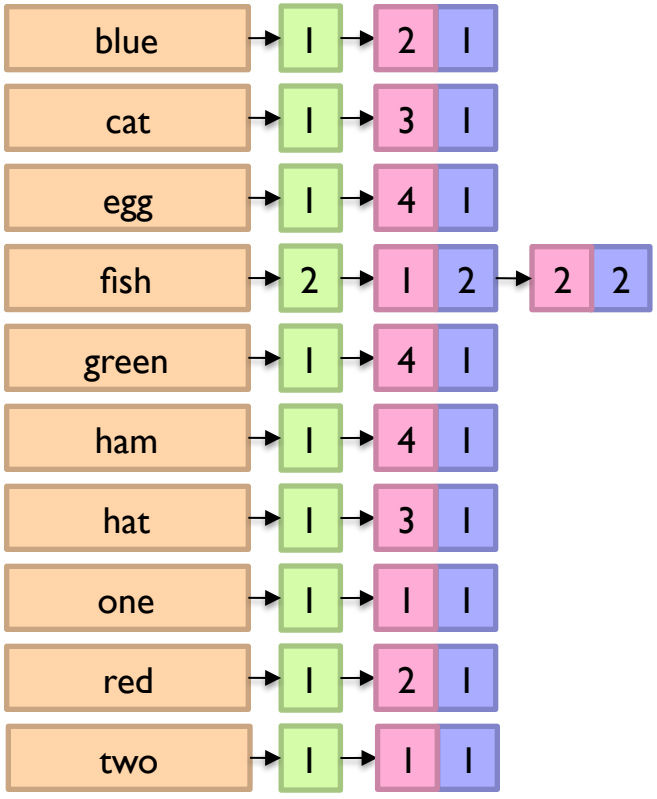
Doc 1
one fish, two fish

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…

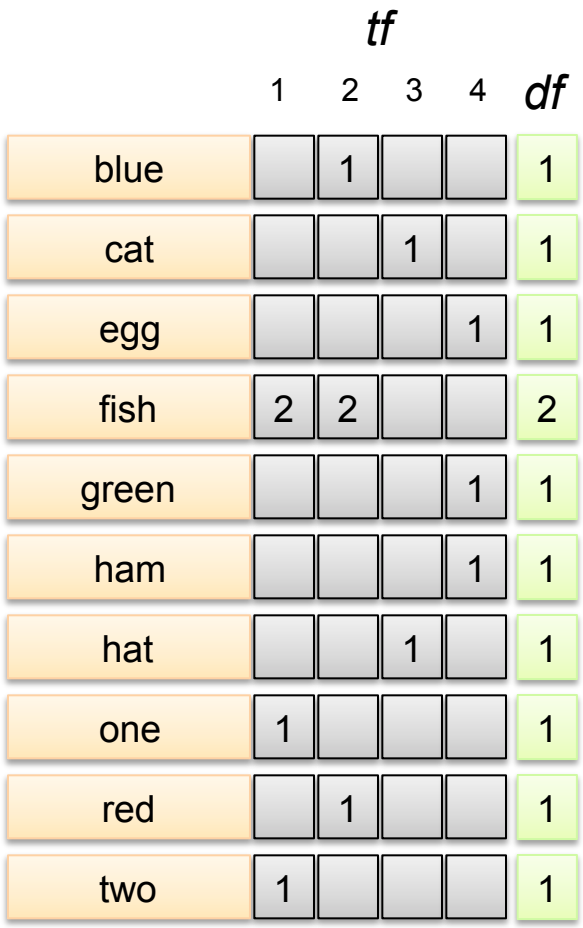
Inverted Index: Positional Information

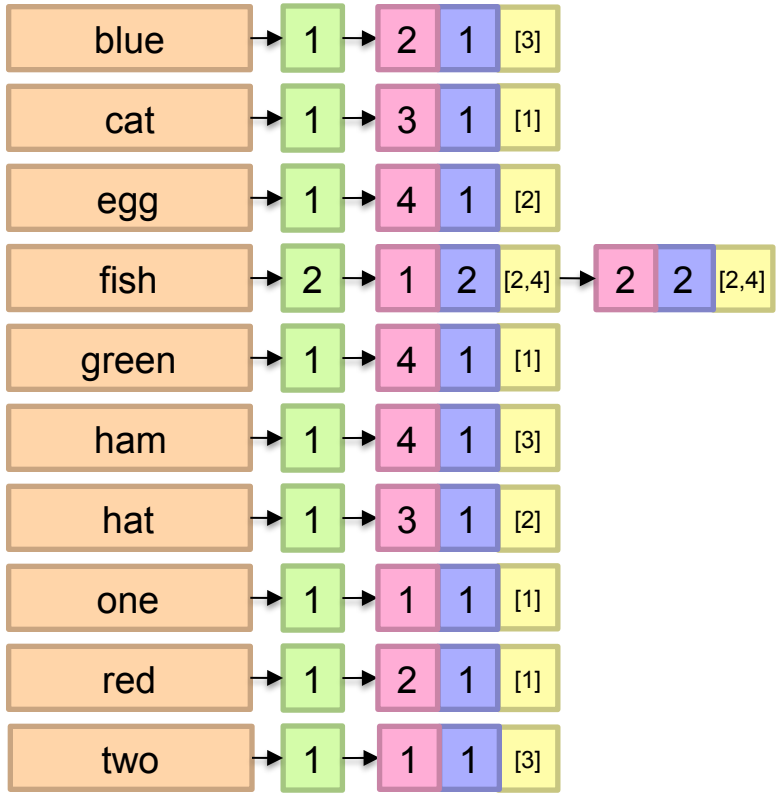
Doc 1
one fish, two fish

Doc 2
red fish, blue fish

Doc 3
cat in the hat

Doc 4
green eggs and ham



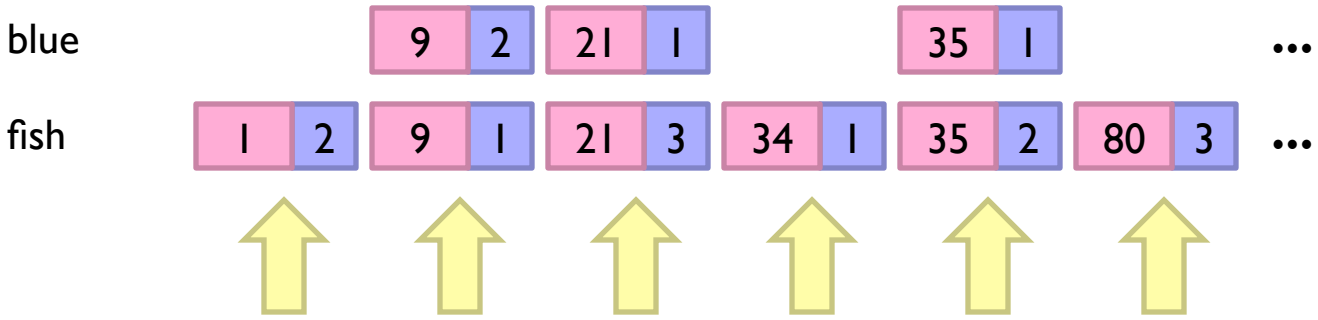


Retrieval in a Nutshell

- 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)

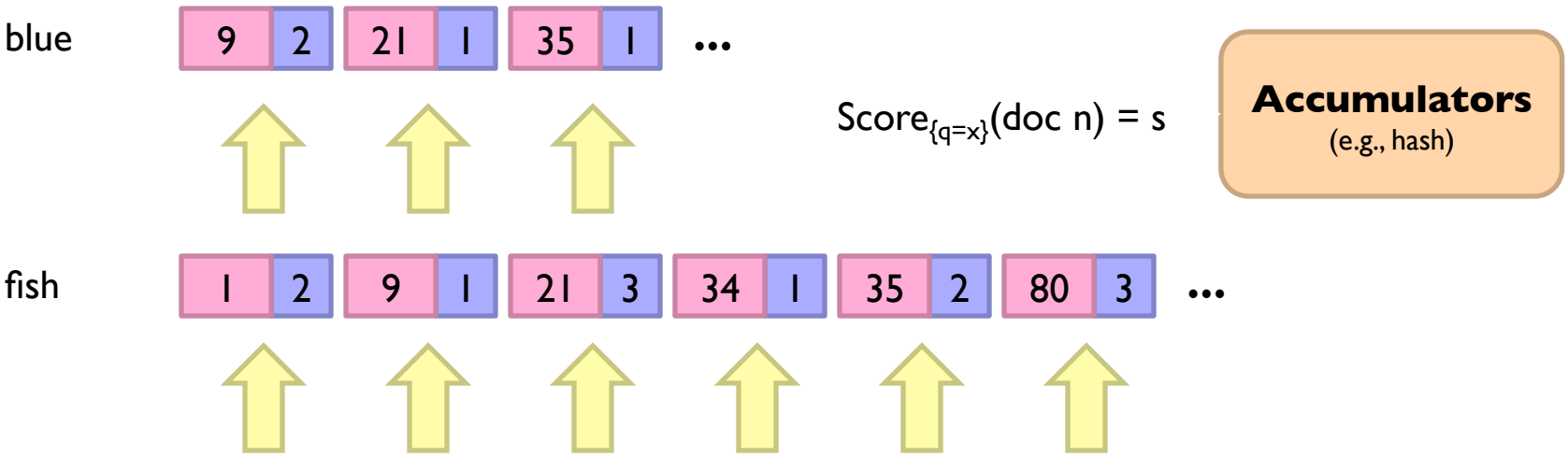
Document score in top k?

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

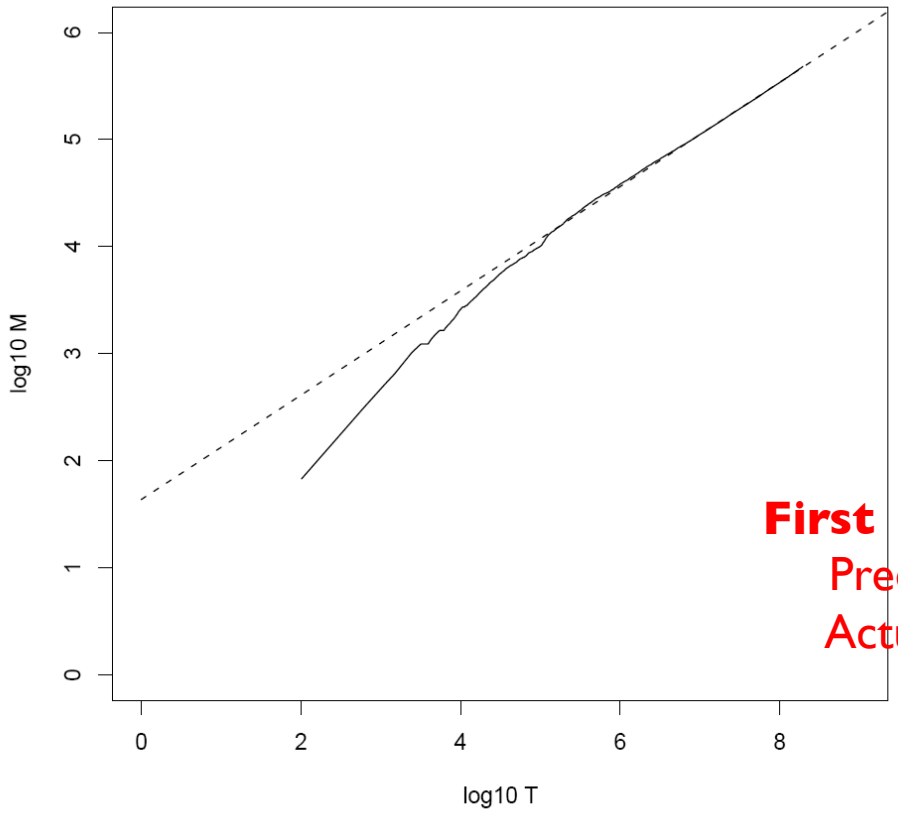
$$M = kT^b$$

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)

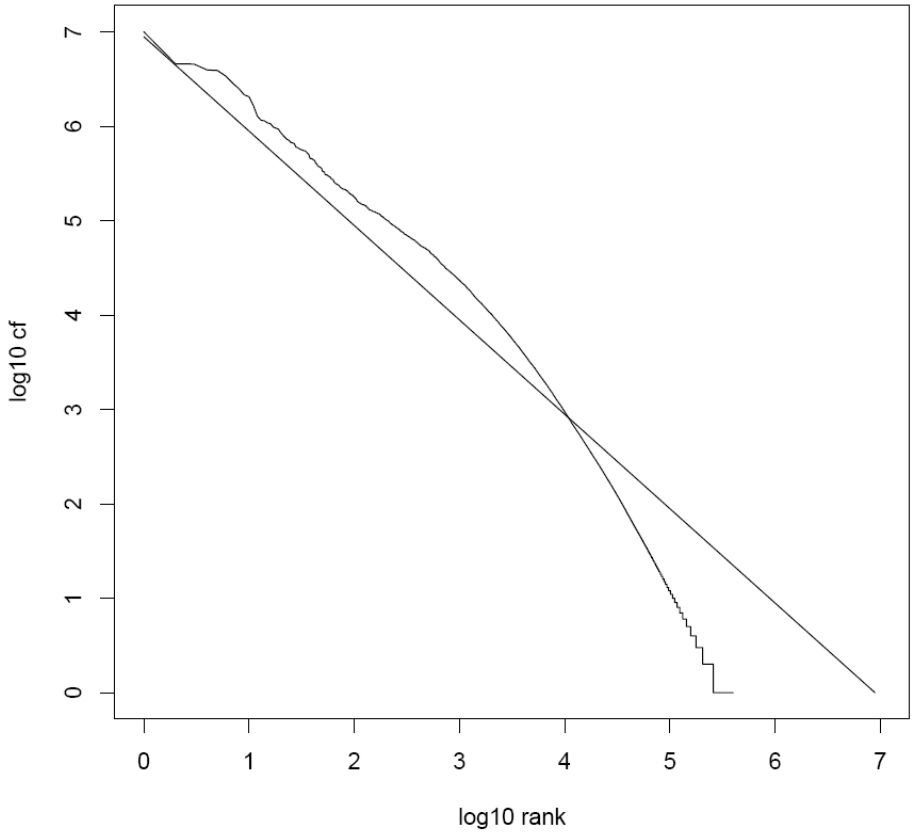
Postings Size: Zipf's Law

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

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)

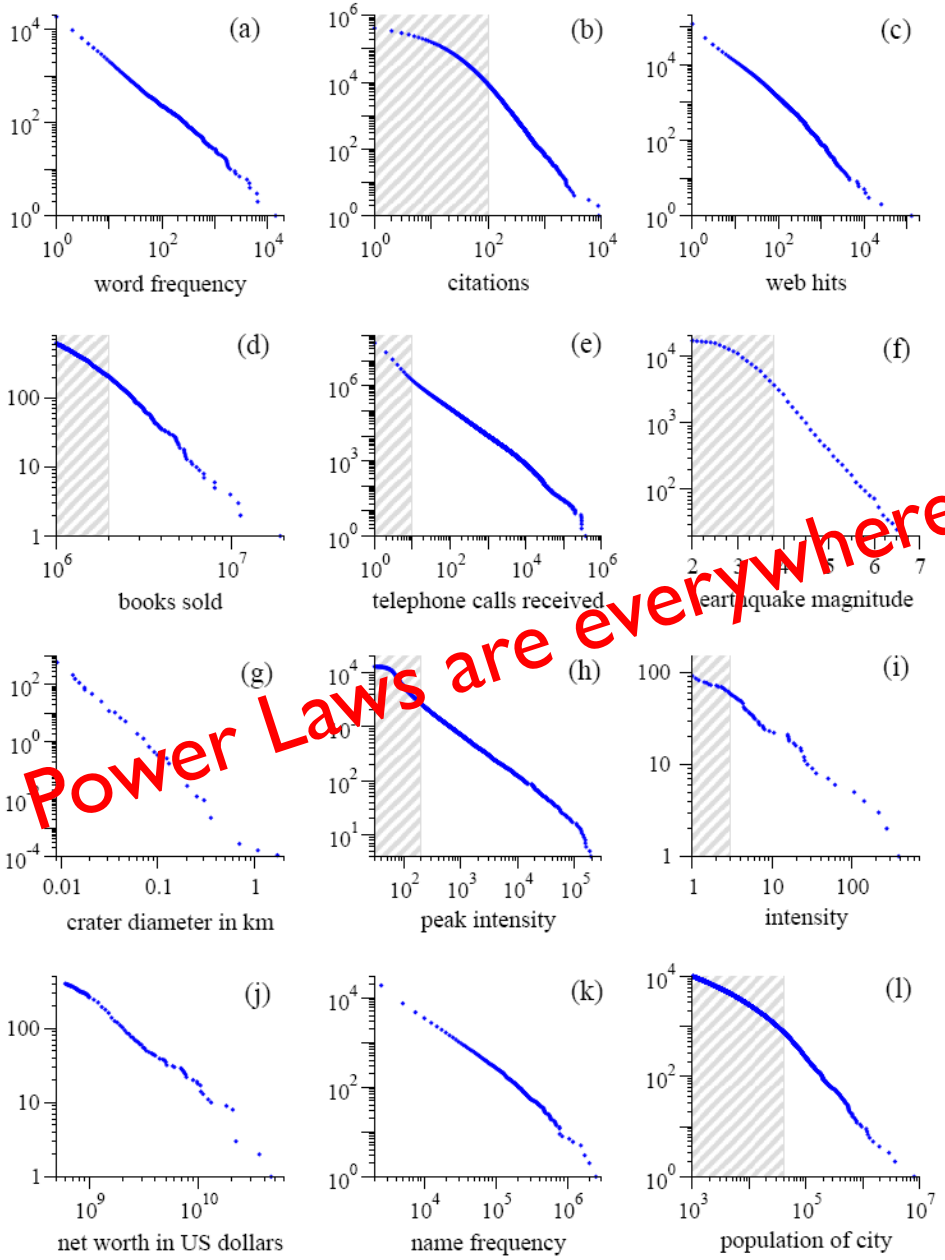
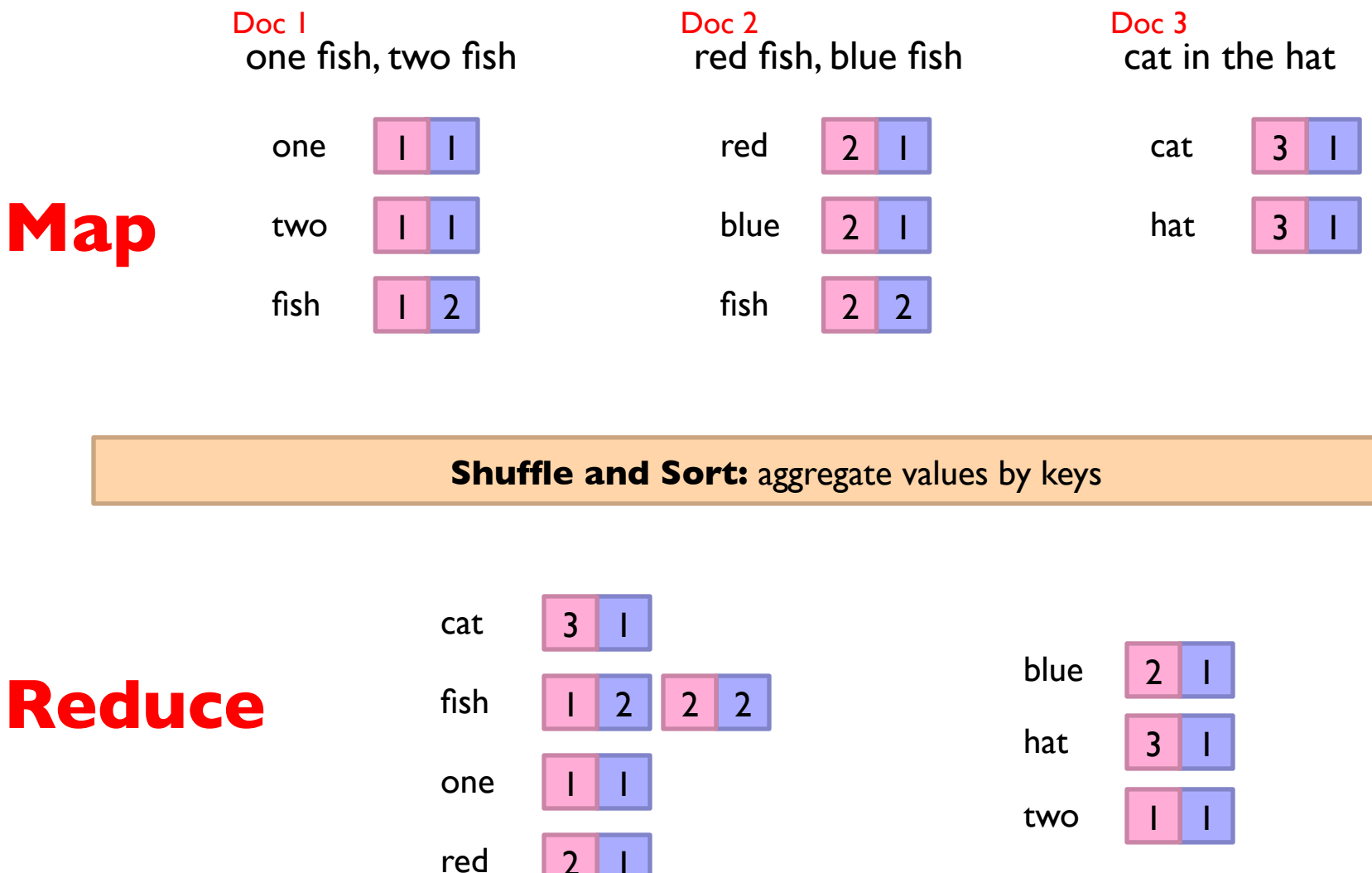


Figure from: Newman, M. E. J. (2005) “Power laws, Pareto distributions and Zipf's law.” Contemporary Physics 46:323–351.

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!

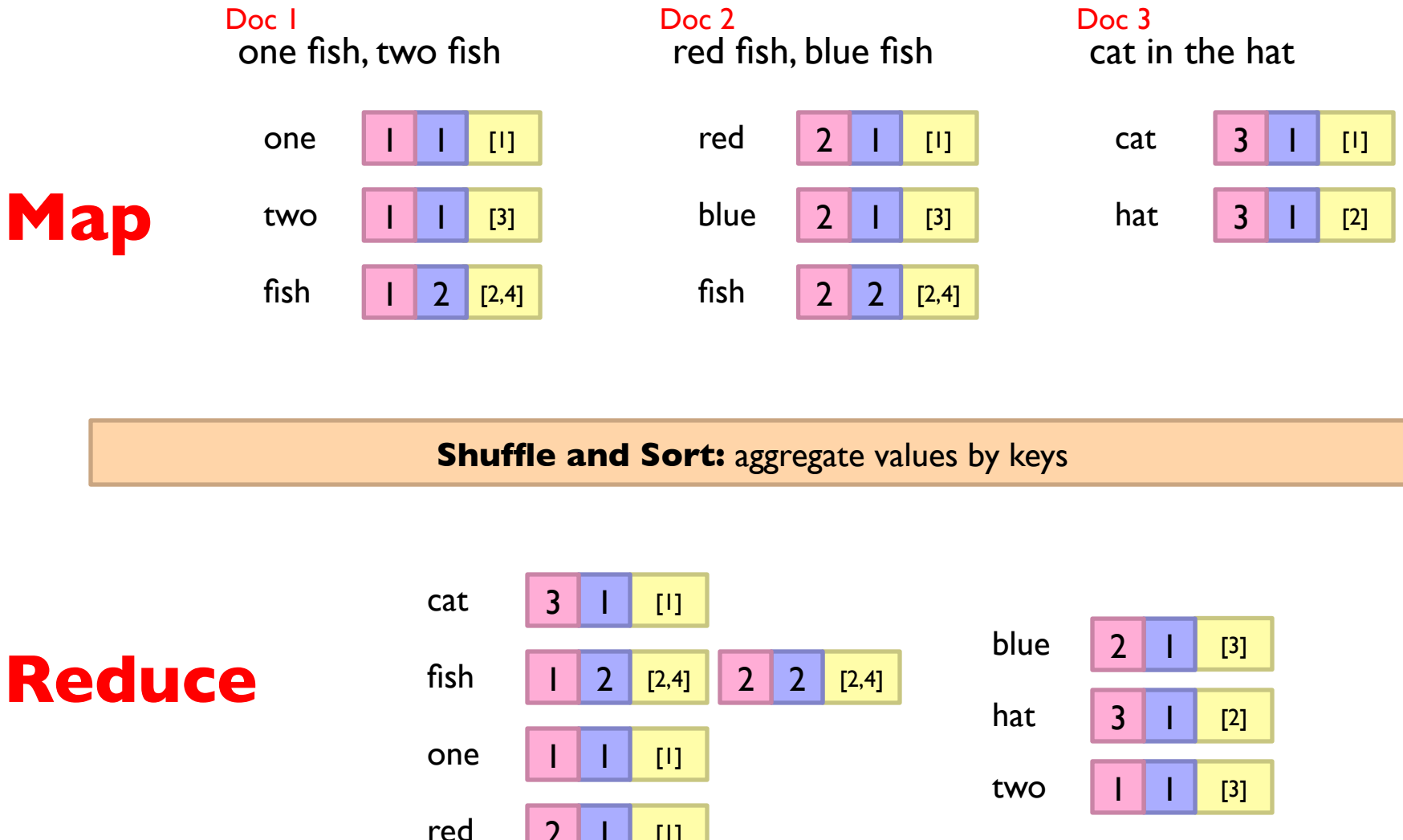
Inverted Indexing with MapReduce



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

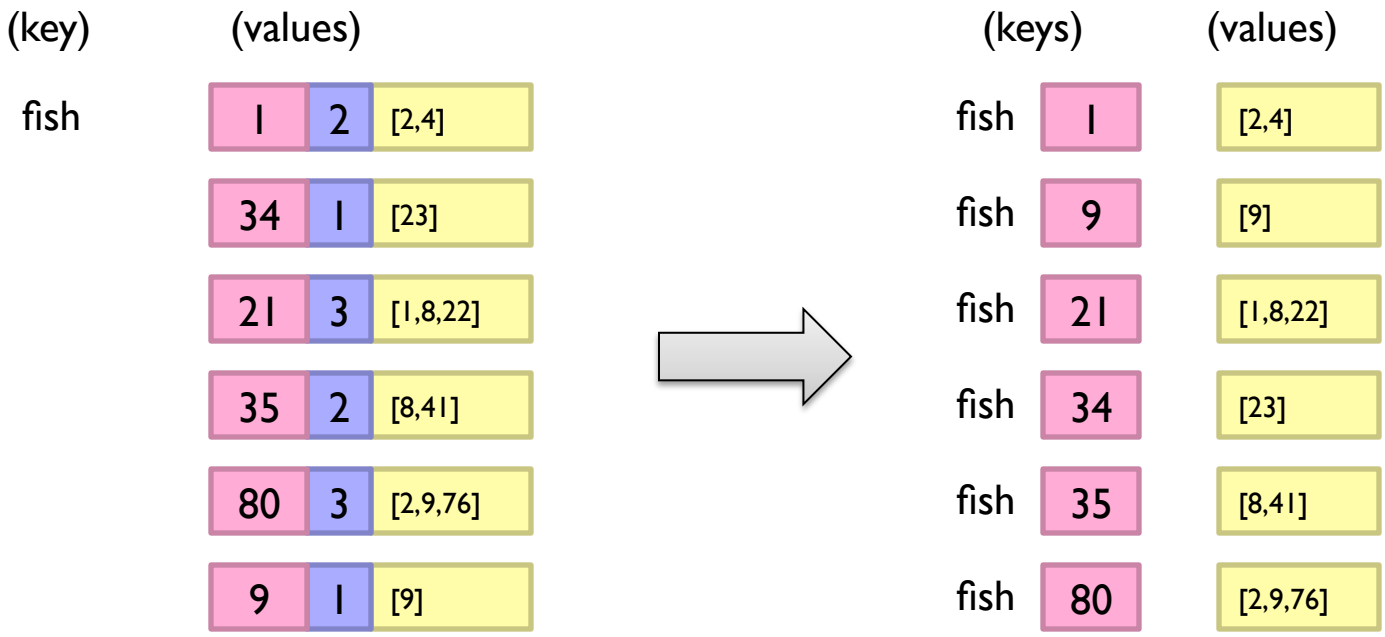
```
1: class Mapper
2:     procedure Map(docid n, doc d)
3:         H ← new AssociativeArray
4:         for all term t ∈ doc d do
5:             H{t} ← H{t} + 1
6:         for all term t ∈ H do
7:             Emit(term t, posting ⟨n, H{t}⟩)

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 save space…

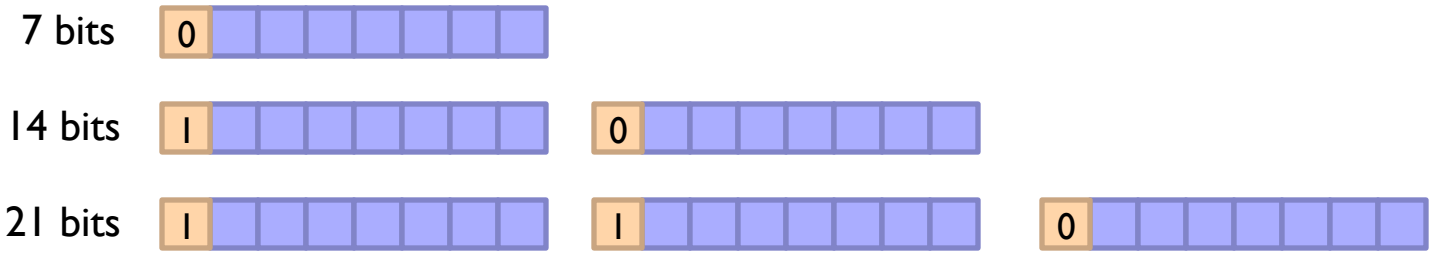
Overview of Index Compression

- Byte-aligned vs. bit-aligned
- Byte-aligned technique
  - VByte
  - Simple9 and variants
  - PForDelta
- Non-parameterized bit-aligned
  - Unary codes
  - γ codes
  - δ codes
- Parameterized bit-aligned
  - Golomb codes (local Bernoulli model)

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

VByte

- Simple idea: use only as many bytes as needed
  - Need to reserve one bit per byte as the "continuation bit"
  - Use remaining bits for encoding value



- Works okay, easy to implement…

Unary Codes

- $x \geq 1$ is coded as x-1 one bits, followed by 1 zero bit
  - 3 = 110
  - 4 = 1110
- Great for small numbers… horrible for large numbers
  - Overly-biased for very small gaps

Watch out! Slightly different definitions in different textbooks

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 $q = \lfloor (x - 1) / b \rfloor$
  - r in binary, where $r = x - qb - 1$, 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)
  - $x = 9$, $b = 3$: $q = 2$, $r = 2$, code = 110:11
  - $x = 9$, $b = 6$: $q = 1$, $r = 2$, code = 10:100
- Optimal $b \approx 0.69$ (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	0:11	0:100
4	1110	110:00	101:00	10:0	0:101
5	11110	110:01	101:01	10:10	0:110
6	111110	110:10	101:10	10:11	0:111
7	1111110	110:11	101:11	110:0	10:00
8	11111110	1110:000	11000:000	110:10	10:01
9	111111110	1110:001	11000:001	110:11	10:100
10	1111111110	1110:010	11000:010	1110:0	10:101

Witten, Moffat, Bell, Managing Gigabytes (1999)

Index Compression: Performance

Comparison of Index Size (bits per pointer)

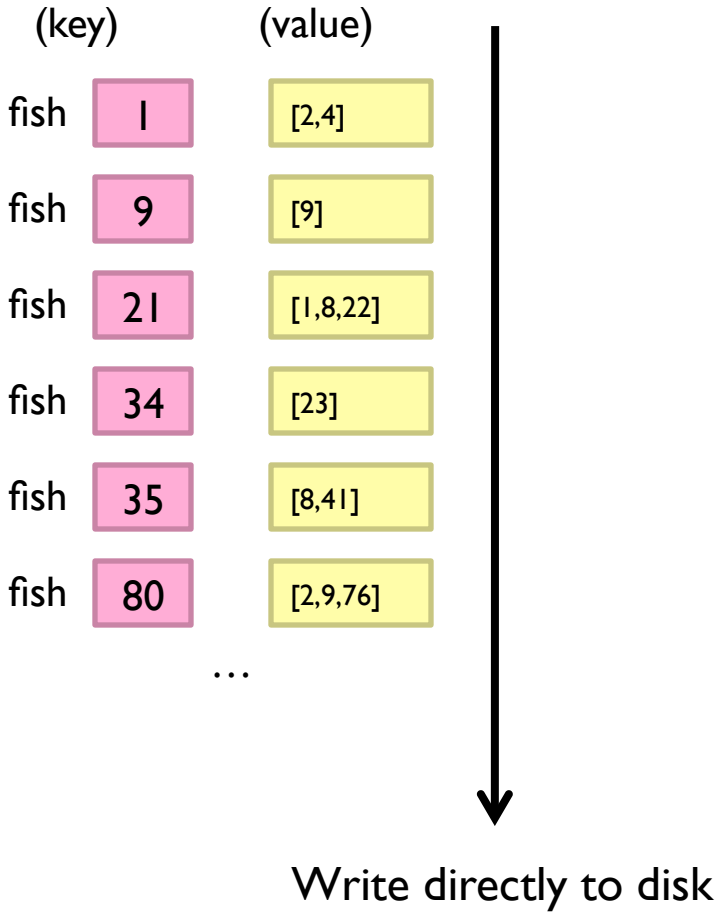
	Bible	TREC
Unary	262	1918
Binary	15	20
γ	6.51	6.63
δ	6.23	6.38
Golomb	6.09	5.84

← One of the best techniques

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)

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

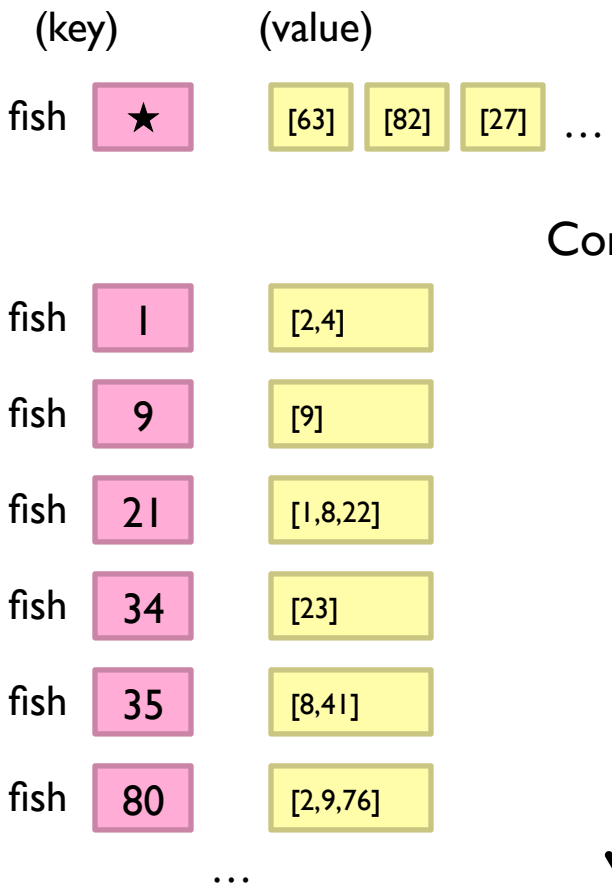
Doc 1

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



First, compute the *df* by summing contributions from all "special" key-value pair…

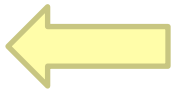
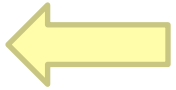
Compute Golomb parameter *b*…

Important: properly define sort order to make sure "special" key-value pairs come first!

Write postings directly to disk

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?

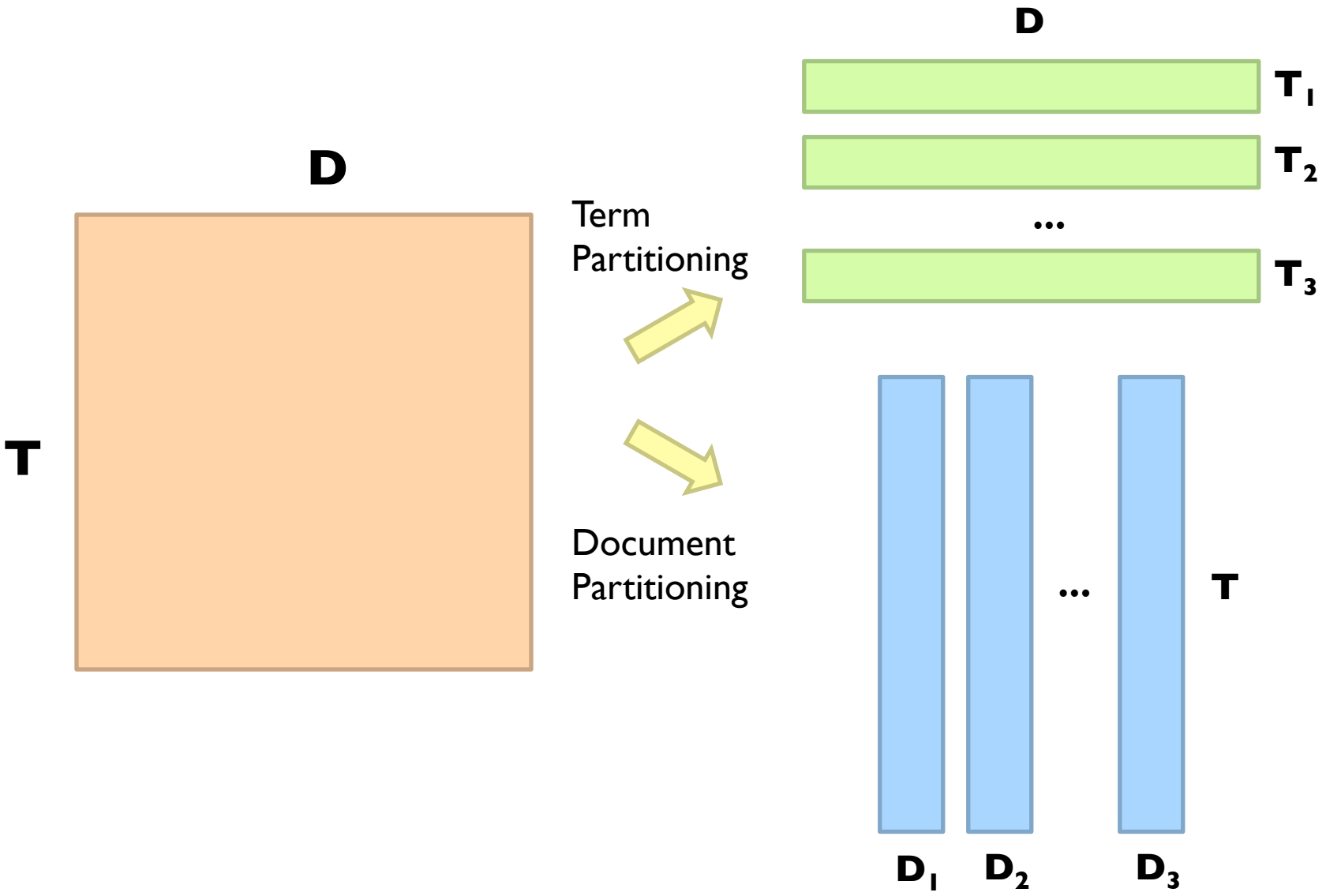
- MapReduce is fundamentally batch-oriented
  - Optimized for throughput, not latency
  - Startup of mappers and reducers is expensive
- MapReduce is not suitable for real-time queries!
  - Use separate infrastructure for retrieval…

Important Ideas

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

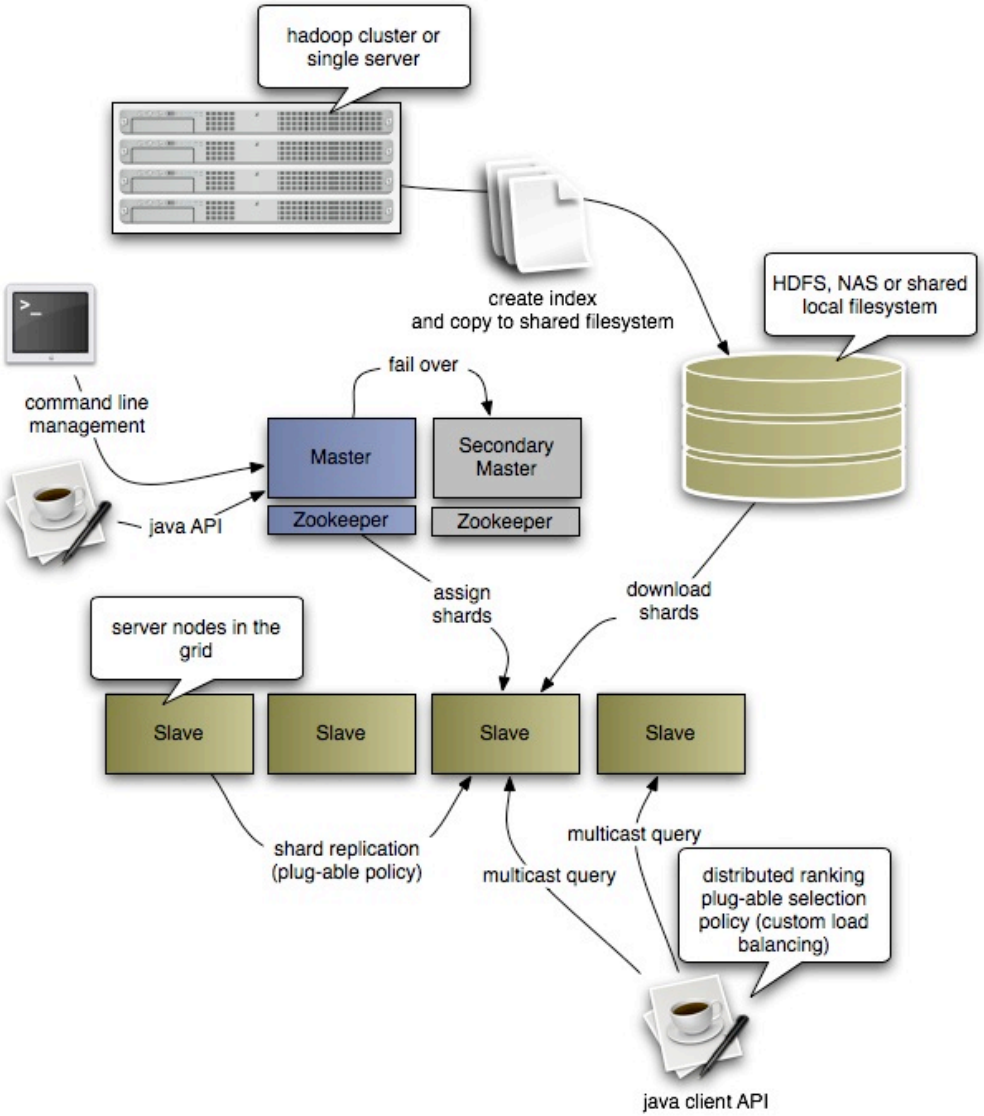
The rest is just details!

Term vs. Document Partitioning



Katta Architecture

(Distributed Lucene)



http://katta.sourceforge.net/

Source: Wikipedia (Japanese rock garden)