Machine Learning at Scale

James G. Shanahan [2]

Assistants: Liang Dai[1, 3]

[1]NativeX, [2]iSchool UC Berkeley, CA, [3]UC Santa Cruz

EMAIL: James_DOT_Shanahan_AT_gmail_DOT_com

Live Session #5

Feb 9, 2016

Live Session Outline

- **Housekeeping**
  - Please mute you microphones
  - Start RECORDING (bonus points for reminding me!)
- **Week 5**
  - Homework HW3, HW4, HW5
  - AWS (hope to have a access for week 6)
  - Async lecture recap plus Q&A [Jimi]
  - *Vector Space Model*
  - *Inverted index*
  - *Similarity*
  - Synonym detection in Map-Reduce
  - GMM
- **Wrapup**
  - Finish RECORDING (bonus points for reminding me!)
  - Click End Meeting

Important Links for Week 5

- **Group 3 Live session #5 recording**
  - https://learn.datascience.berkeley.edu/local/adobecp/launch.php?cpurl=p3da331975i&recording=y&livesessionid=11042.
  - Please watch the last hour for an explanation synonym detection and some hints!
- **Instructions for Peer grading of homework HW**
  - https://www.dropbox.com/s/97m31frthj4ac28/HOMEWORK%20GRADING%20INSTRUCTIONS%20for%20MIDS%20MLS?dl=0
- **Homework HW4 Folder Questions + Data**
  - HW4 is a group oriented homework
  - https://www.dropbox.com/sh/0cv65h44zylqwe3/AABL7xJysibPb3gSNvlJwAM8a?dl=0
- **Team assignments**
  - https://docs.google.com/spreadsheets/d/1ncFQl5Tovn-16slD8mYjP_nzMTPSfiGeLLzW8v_sMjg/edit?usp=sharing
- **Please submit your homeworks (one per team) going forward via this form (and not thru the ISVC):**
  - https://docs.google.com/forms/d/1ZOr9RnIe_A06AcZDB6K1mJN4vrLeSmS2PD6Xm3eOiis/viewform?usp=send_form

Cloud Computing Access: IBM's SoftLayer

- **Unfortunately, we will not have access to Amazon's cloud this week.**
- **Karthik, Prabhakar will help getting you set up on IBM's SoftLayer**
- **I will copy the data from Amazon's S3 to DropBox**

Unit 5 | Big Data Pipelines

Hide Contents

- 5.1 Weekly Introduction (3 mins)
- 5.2 Assigned Readings
- 5.3 Mobile Advertising Example (19 mins)
- 5.4 Data Warehousing OLTP OLAP (11 mins)
- 5.5 Database Relational Joins (12 mins)
- 5.6 Databases Operations in MapReduce (10 mins)
- 5.7 Relational Joins (11 mins)
- 5.8 Reduce Side Join (8 mins)
- 5.9 Reduce Side Join Example (4 mins)
  - 5.9.1 Quiz: Reducer-Side Join
- 5.10 Memory Backed Map Side Join (2 mins)
- 5.11 Map Side Join Merge Join (4 mins)

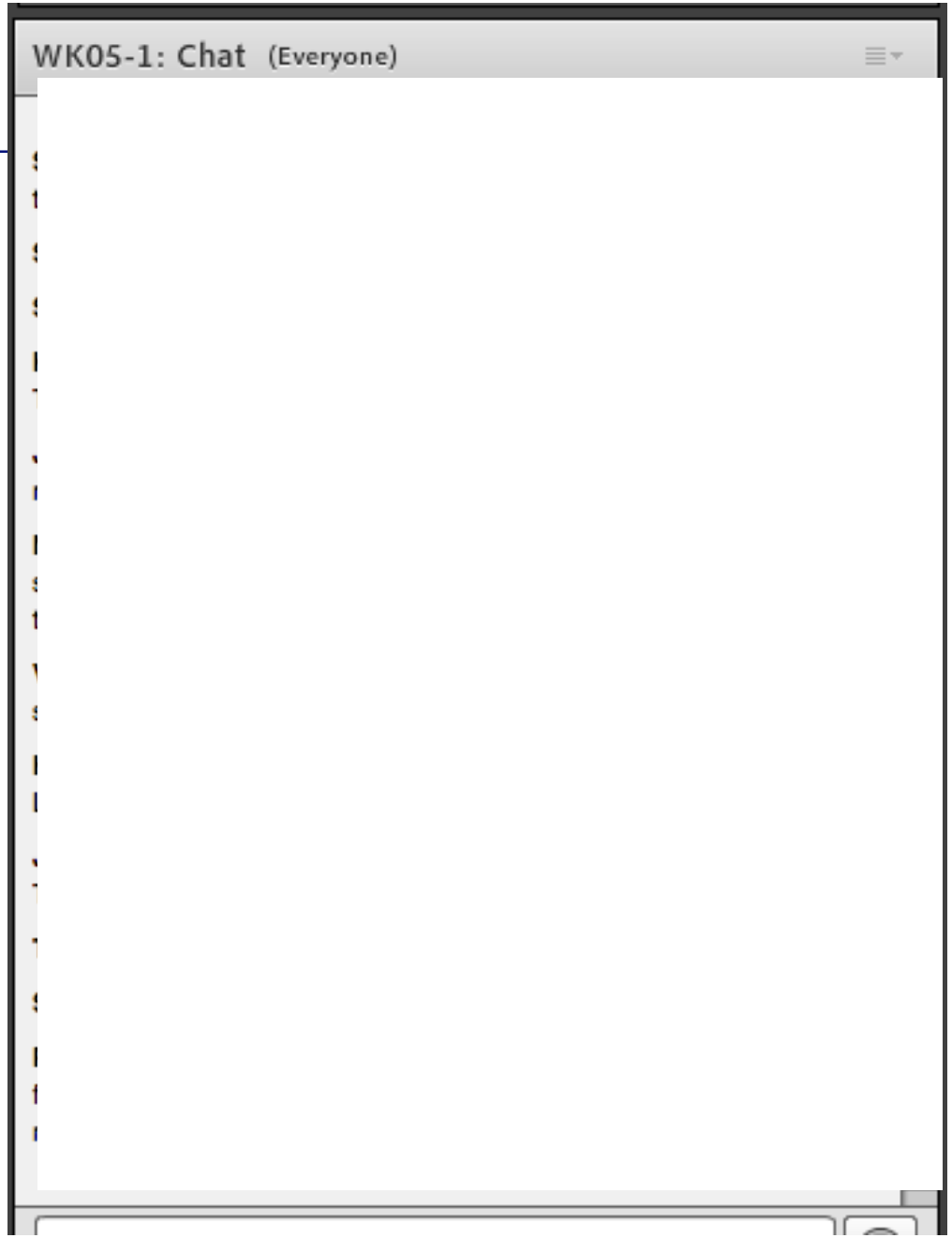
Join Question

- **Transaction Logfile/DB/CSV file (1 Billion transactions)**
  - User ID, Date, Time, Referring URL, item purchased, price, etc..
- **User Information/Location file/DB (1Million records)**
  - User ID, HomeCountry, HomeState, HomeZipCode, etc..
  - 5 numbers X 2 bytes X * 10^6 = 10^7 (Around 10 MEG)
- **Join Transaction DB with Location DB using the USER_ID (e.g., Phone number)**

TASK

- **Using Hadoop, what type of join would you recommend? Complete this job within one hour ever hour!**

- ..


WK05-1: Chat (Everyone)

- ..

Join part 2

- **Left table (Customer information table)**
- **Right table (Transaction table)**

- **Left/Right/Inner/Outer Join**

- **Right join:**
  - some customers may not exist

- **HashJoin? Reduce side Join?**

Advertising ~2% of US GDP; $140B WW

"*Half the money I spend on advertising is wasted; the trouble is, I don't know which half.*" - John Wanamaker, father of modern advertising.

- Less than 1% of all impressions lead to measureable ROI

$400 Million on Super Bowl Advertising TV/Online

Despite its problems (Attribution, etc.)

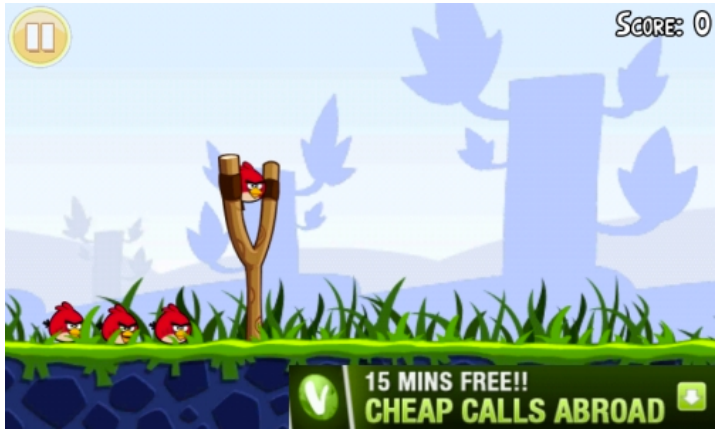
- **US GDP = $14.1 Trillion (Global $56 Trillion, $56x10^{12}$)**
- **US Advertising Spend**
  - ~$275 Billion across all media
    - (2% of GDP since the early 1900s)
- **In 2015, Worldwide online advertising was $150Billion**
  - I.e., about 20% of all ad spending across all media
  - $42 billion global mobile-advertising market in 2014
  - *$100 billion global mobile-advertising market in 2016*

Mobile Publisher: How do I make money?

Consumer:
App user

- **Paid app download**
- **In app purchases**
- **In app Advertising**

Publisher:
App Developer

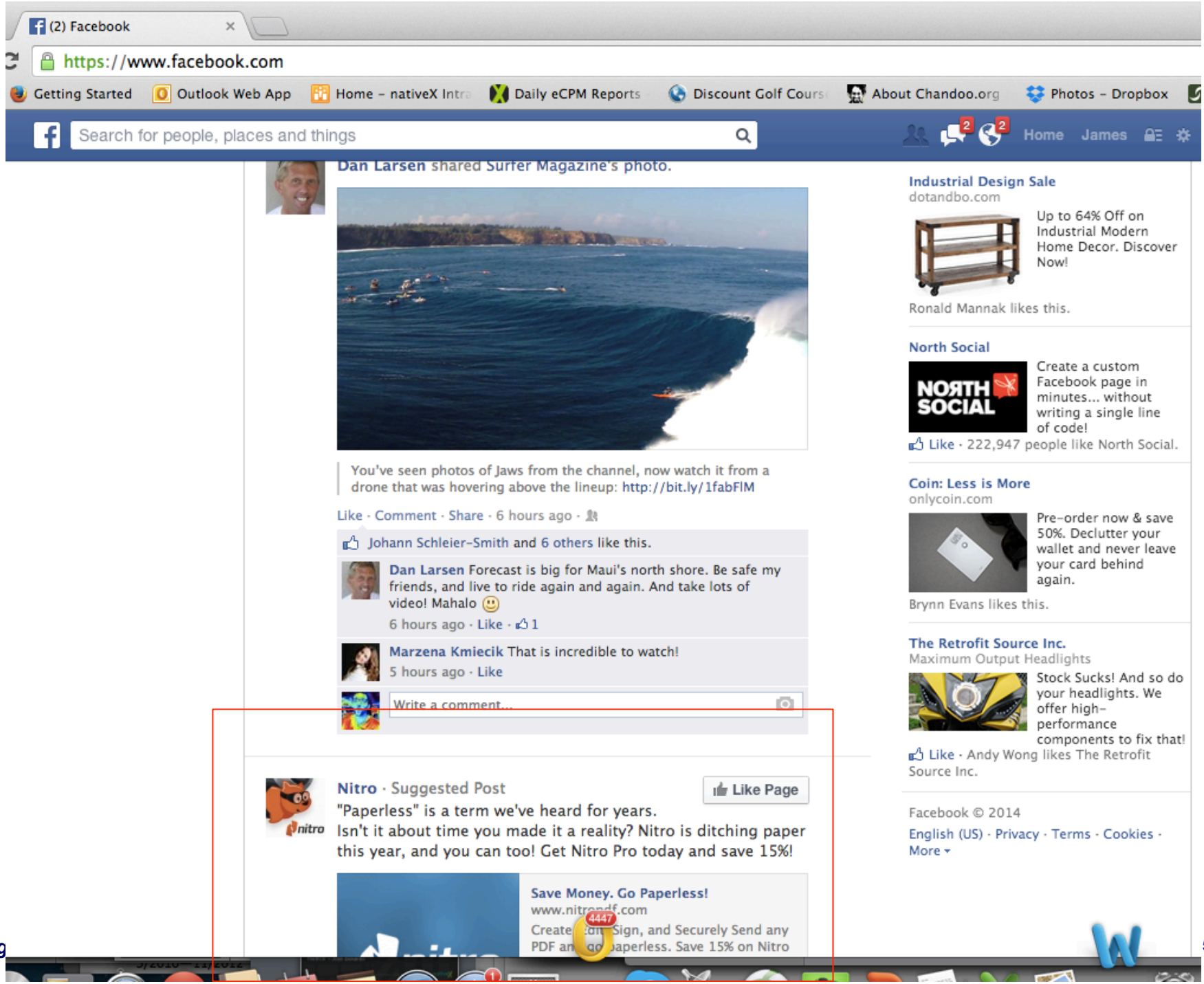


Making Money from Apps

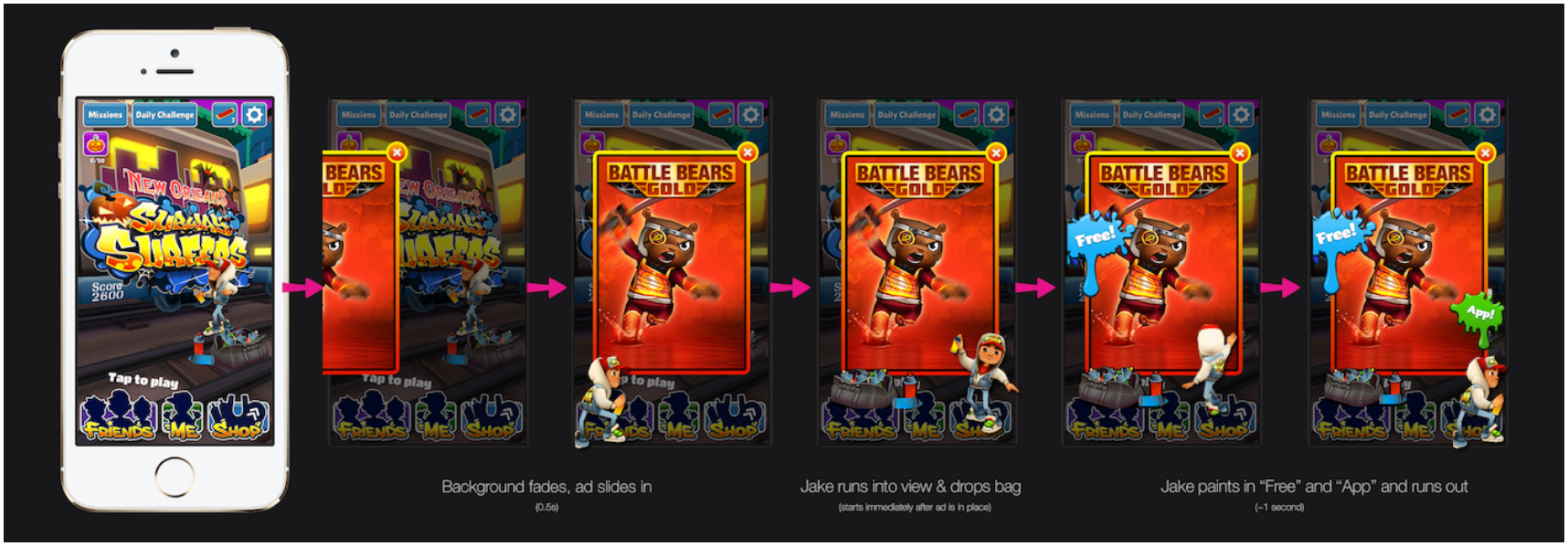
- **93% of downloaded apps in 2013 (globally) are free apps !**

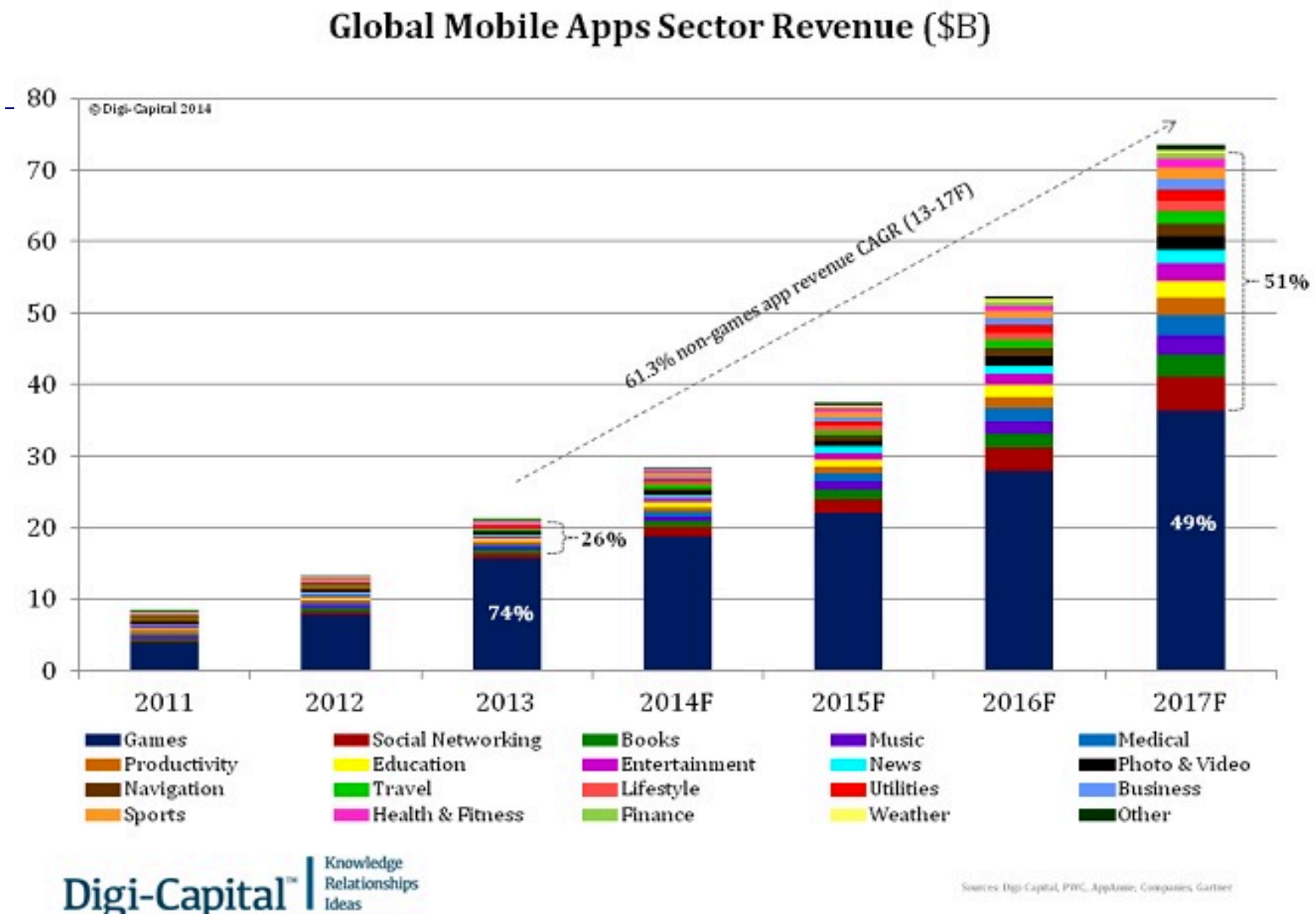
- **76% of revenue generated from apps (globally) in 2013 is from in-app purchases**
  - [http://www.forbes.com/sites/chuckjones/2013/03/31/apps-with-in-app-purchase-generate-the-highest-revenue/]
- **In the Freeium economy**
  - To make money from apps, publishers must maintain customer satisfaction through superior app performance and design,
  - then monetize though advertising and in-app purchases

[http://venturebeat.com/2014/03/27/mobile-app-monetization-freemium-is-king-but-in-app-ads-


(2) Facebook
https://www.facebook.com
Getting Started
Outlook Web App
Home – nativeX Intra
Daily eCPM Reports
Discount Golf Cours
About Chandoo.org
Photos – Dropbox
Search for people, places and things
Home
James
Dan Larsen shared Surfer Magazine's photo.
You've seen photos of Jaws from the channel, now watch it from a drone that was hovering above the lineup: http://bit.ly/1fabFlM
Like · Comment · Share · 6 hours ago ·
Johann Schleier-Smith and 6 others like this.
Dan Larsen Forecast is big for Maui's north shore. Be safe my friends, and live to ride again and again. And take lots of video! Mahalo
6 hours ago · Like · 1
Marzena Kmiecik That is incredible to watch!
5 hours ago · Like
Write a comment...
Nitro · Suggested Post
Like Page
"Paperless" is a term we've heard for years.
Isn't it about time you made it a reality? Nitro is ditching paper this year, and you can too! Get Nitro Pro today and save 15%!
Save Money. Go Paperless!
www.nitropdf.com
Industrial Design Sale
dotandbo.com
Up to 64% Off on Industrial Modern Home Decor. Discover Now!
Ronald Mannak likes this.
North Social
NORTH SOCIAL
Create a custom Facebook page in minutes... without writing a single line of code!
Like · 222,947 people like North Social.
Coin: Less is More
onlycoin.com
Pre-order now & save 50%. Declutter your wallet and never leave your card behind again.
Brynn Evans likes this.
The Retrofit Source Inc.
Maximum Output Headlights
Stock Sucks! And so do your headlights. We offer high-performance components to fix that!
Like · Andy Wong likes The Retrofit Source Inc.
Facebook © 2014
English (US) · Privacy · Terms · Cookies · More

Native Advertising





http://venturebeat.com/2014/04/29/mobile-apps-could-hit-70b-in-revenues-by-2017-as-non-game-categories-take-off/

Mobile Ad Spend to Top $100 Billion Worldwide in 2016, 51% of Digital Market

- **US and China will account for nearly 62% of global mobile ad spending next year**

Mobile Internet Ad Spending Worldwide, 2013-2019

	2013	2014	2015	2016	2017	2018	2019
Mobile internet ad spending (billions)	**$19.20**	**$42.63**	**$68.69**	**$101.37**	**$133.74**	**$166.63**	**$195.55**
—% change	117.9%	122.1%	61.1%	47.6%	31.9%	24.6%	17.4%
—% of digital ad spending	16.0%	29.4%	40.2%	51.1%	59.4%	65.9%	70.1%
—% of total media ad spending	3.7%	7.8%	11.9%	16.5%	20.5%	24.1%	26.8%

Note: includes display (banners, video and rich media) and search; excludes SMS, MMS and P2P messaging-based advertising; ad spending on tablets is included
Source: eMarketer, March 2015

186887 www.eMarketer.com

- http://www.emarketer.com/Article/Mobile-Ad-Spend-Top-100-Billion-Worldwide-2016-51-of-Digital-Market/1012299#sthash.FBfZAlaC.dpuf

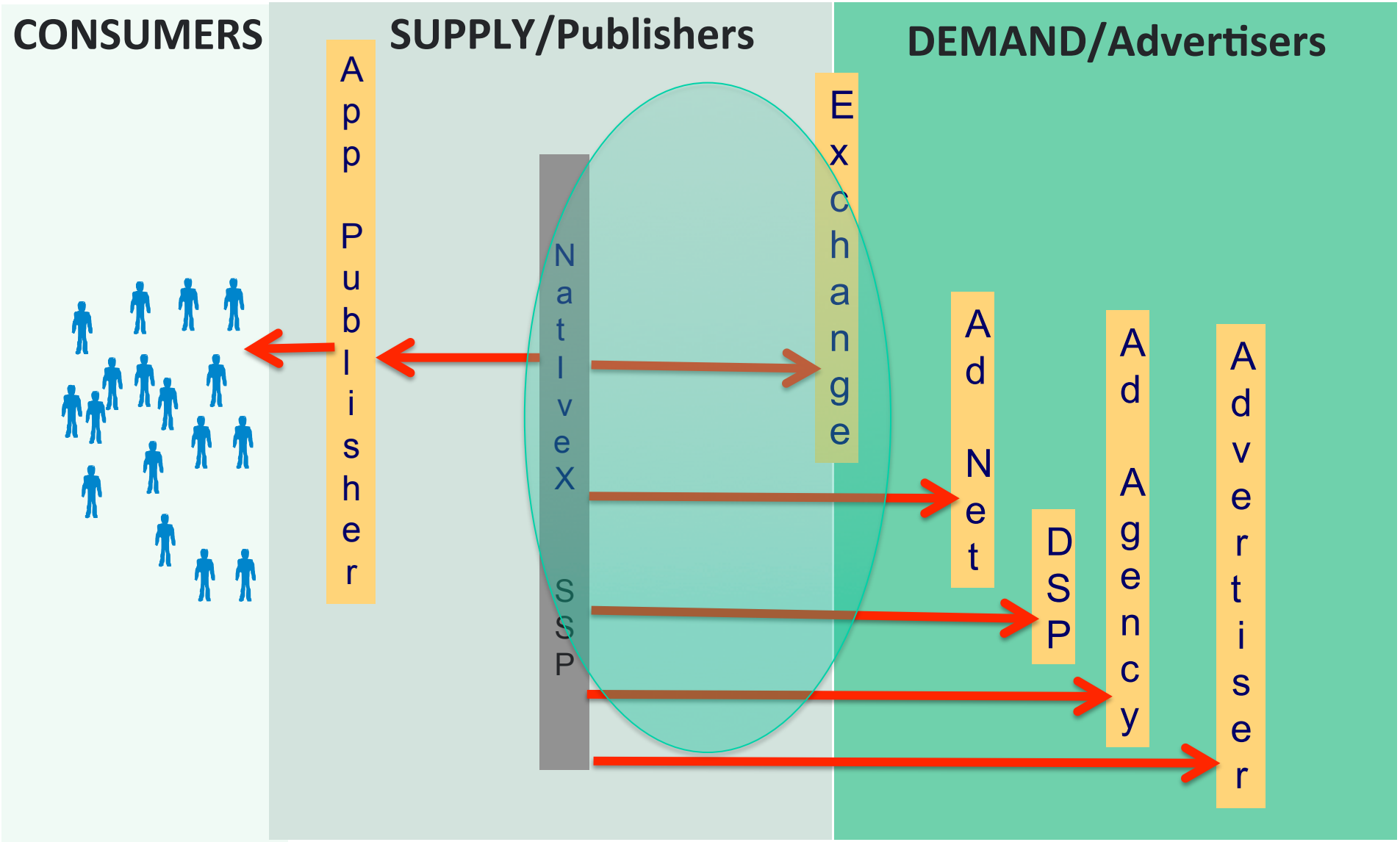
CPMs on Mobile are catching up

- **Unit of counting is 1,000 displays/impressions/views**
- **CPM: cost of displaying the ad 1,000 times**
- **Mobile Advertising: What is the average CPM on mobile?**
  - The effective cost per thousand impressions (CPM) for desktop web ads is about $3.50, while the CPM for mobile ads is just $0.75.
  - Video-based CPMs typically > $15-$20

http://www.quora.com/Mobile-Advertising/What-is-the-average-CPM-on-mobile

http://mashable.com/2012/10/23/mobile-ad-prices/

NativeX: Art and Science of Native Mobile Advertising



Ad serving data pipelines

Bid X $CTR_{AD,\ Context}$ X 1000 =$eCPM_{Ad}$

Realtime Batch

ETL
(Extract, Transform, and Load)

"OLTP" Data Pipeline

- Online Data
- Used in Real Time
- Used for Offer Serving

Data Science Predictive Analytics Pipeline

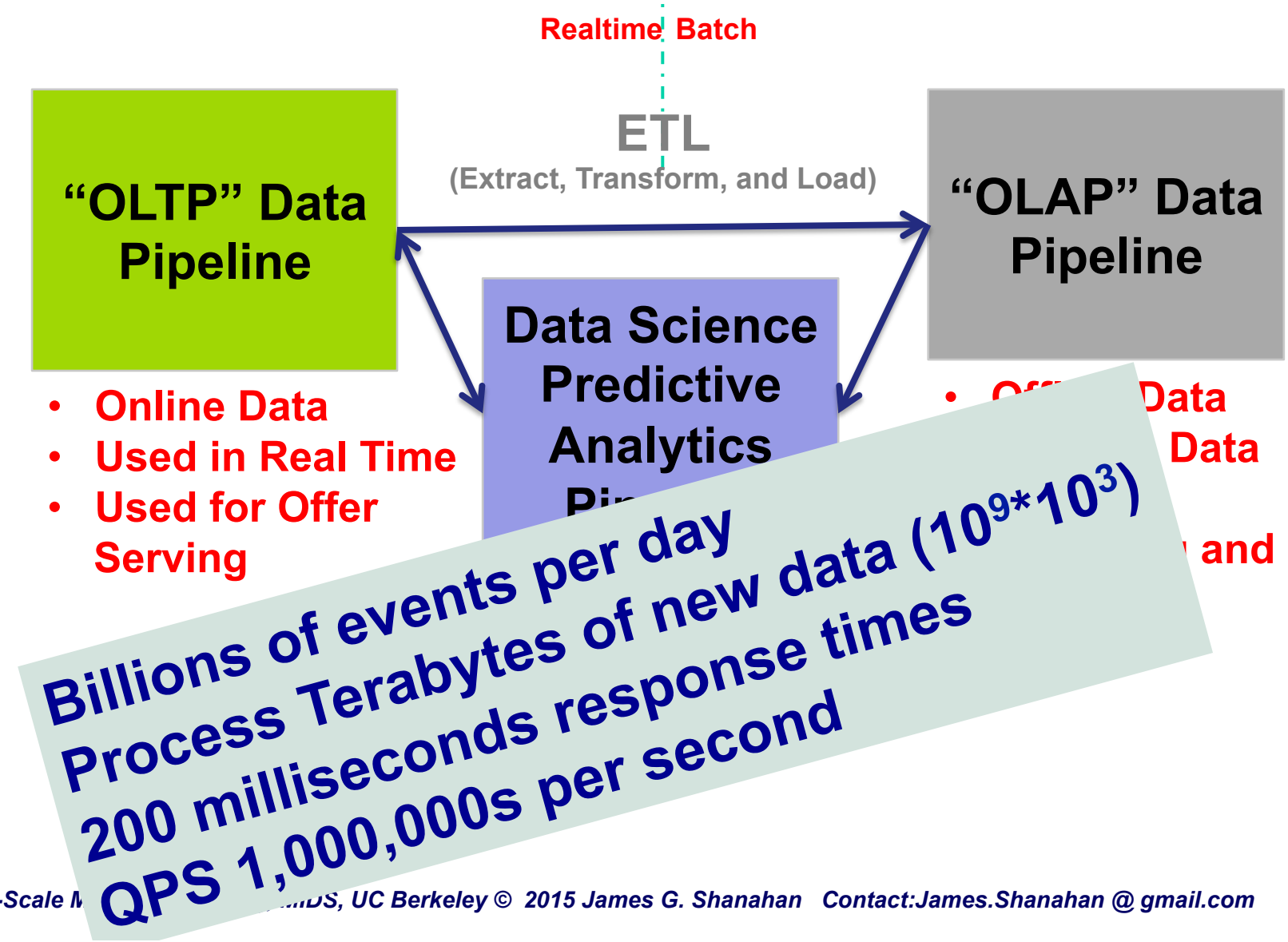
- Offline Batch Modeling
- Real-time Ad Serving

"OLAP" Data Pipeline

- Offline Data
- Logging Data
- Used for Reporting and Modeling

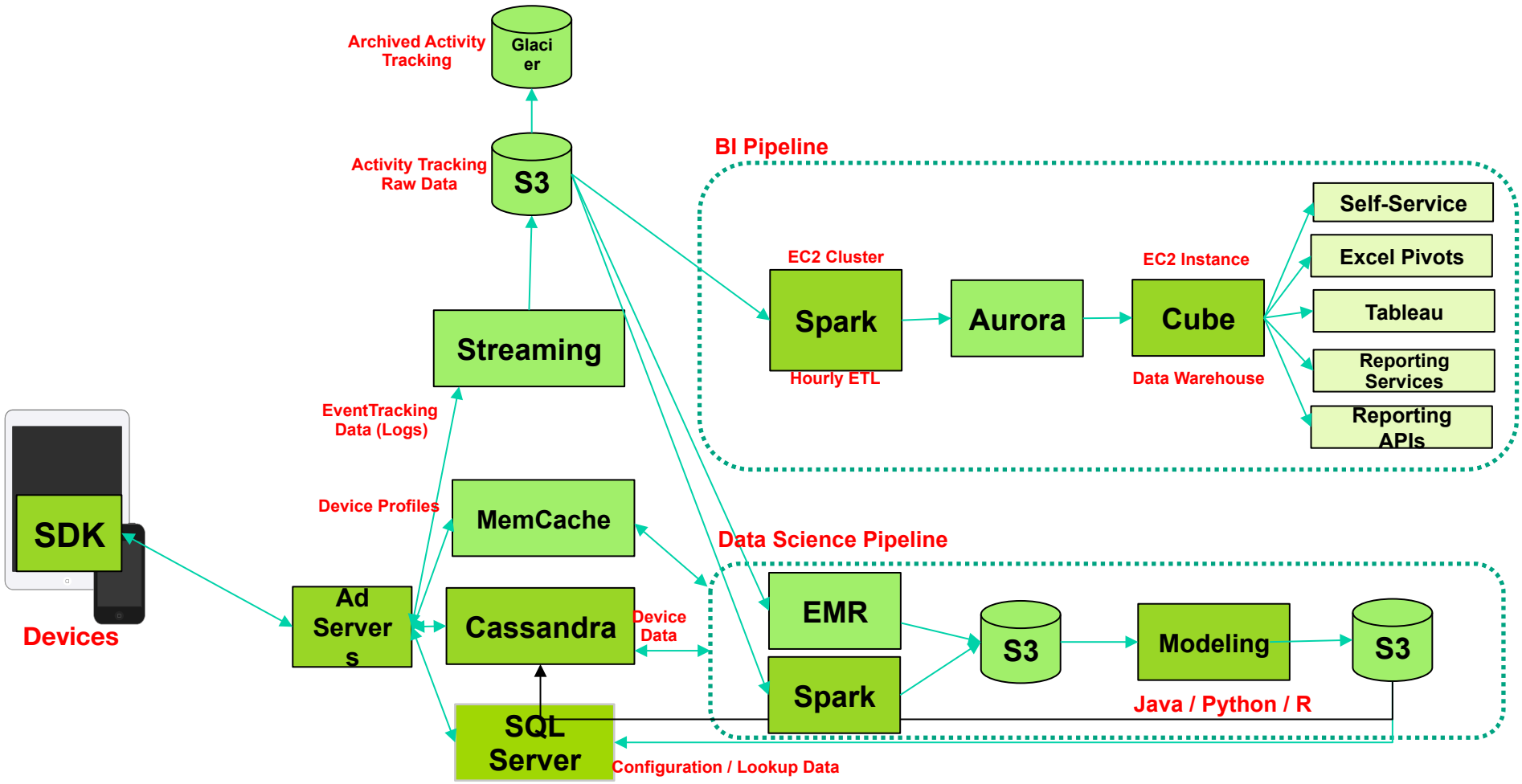
100 Milliseconds

Ad serving data pipelines

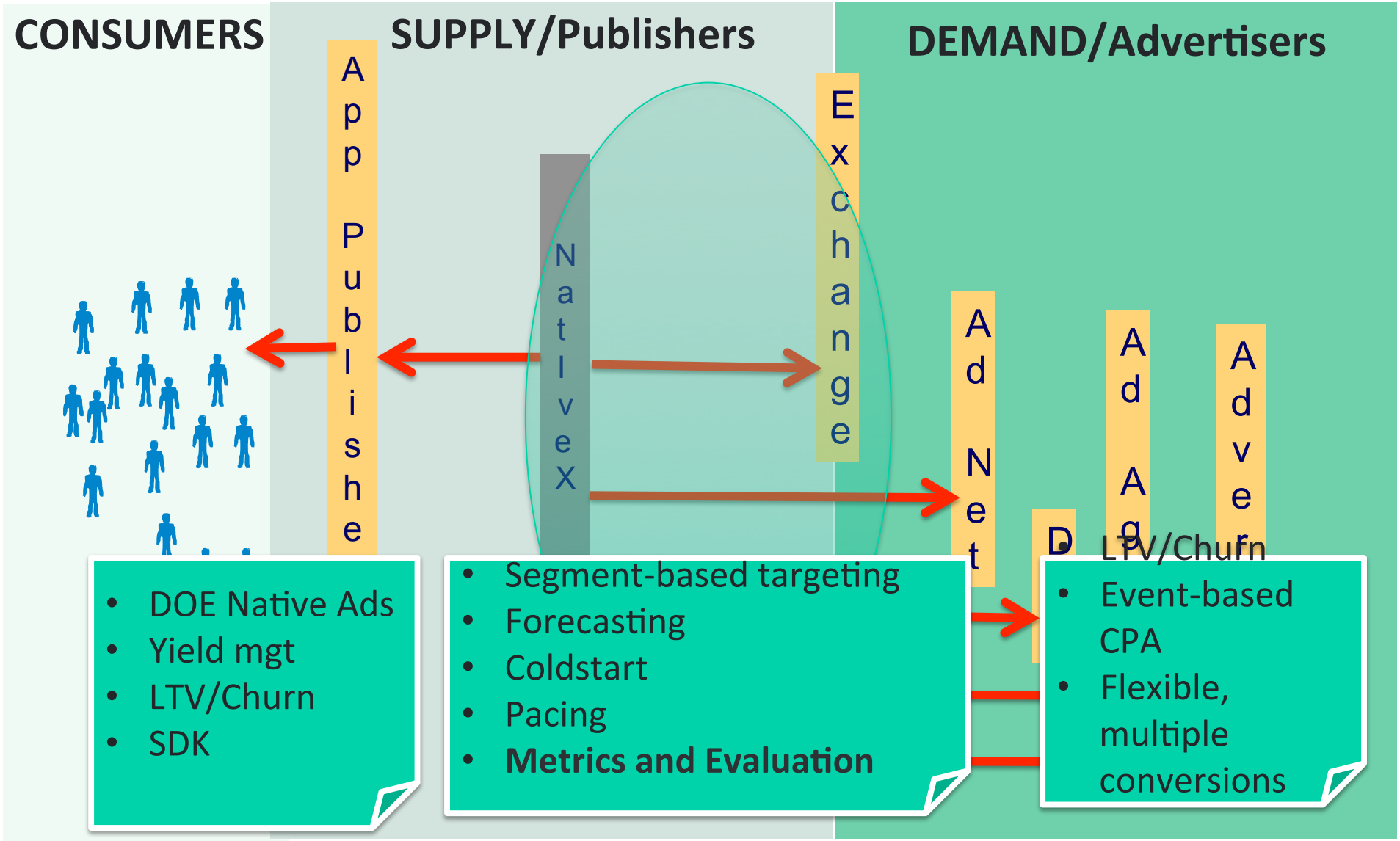


Ranking Ads (more) at Turn Inc.

Potential Ad serving architecture



NativeX: Art and Science of Native Mobile Advertising



DB concepts in practice

- 3rd NF
- Normal forms
- Denormalized data
- HIVE, SQL, Lateral joins, Exploding tables (Denormalized data)

Live Session Outline

- **Housekeeping**
  - Please mute you microphones
  - Start RECORDING (bonus points for reminding me!)
- **Week 5**
  - Homework HW3, HW4, HW5
  - AWS (hope to have a access for week 6)
  - Async lecture recap plus Q&A [Jimi]
  - *Vector Space Model*
  - *Inverted index*
  - *Similarity*
  - Synonym detection in Map-Reduce
  - GMM
- **Wrapup**
  - Finish RECORDING (bonus points for reminding me!)
  - Click End Meeting

- **VSM**

Vector space model (VSM)

- Vector space model or term vector model is an algebraic model for representing text documents (and any objects, in general) as vectors of identifiers, such as, for example, index terms.
- It is used in information filtering, information retrieval, indexing and relevancy rankings.
- Its first use was in the SMART Information Retrieval System.
- Embedding/representing our documents in VS.

VSM at work: doc, query

TF
Relative frequency
TF X IDF

Documents and queries are represented as vectors.

$$d_j = (w_{1,j}, w_{2,j}, \ldots, w_{t,j})$$
$$q = (w_{1,q}, w_{2,q}, \ldots, w_{n,q})$$

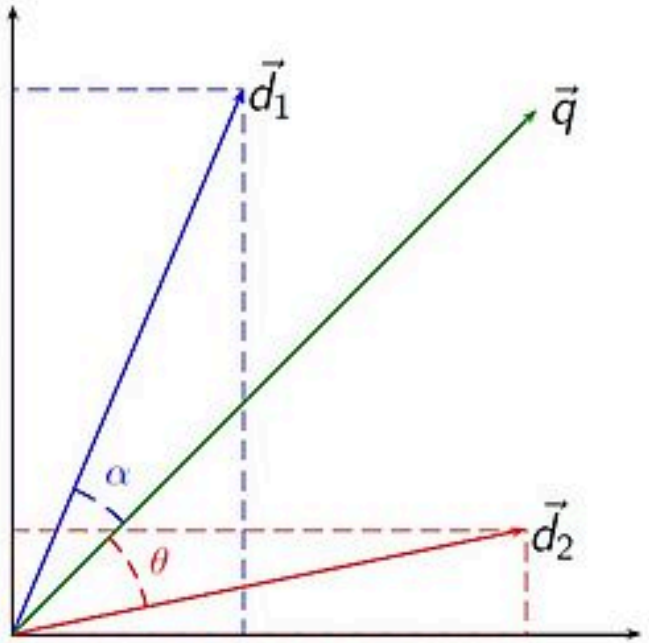
Each dimension corresponds to a separate term. If a term occurs in the document, its value in the vector is non-zero. Several different ways of computing these values, also known as (term) weights, have been developed. One of the best known schemes is tf-idf weighting (see the example below).

The definition of *term* depends on the application. Typically terms are single words, keywords, or longer phrases. If words are chosen to be the terms, the dimensionality of the vector is the number of words in the vocabulary (the number of distinct words occurring in the corpus).

Vector operations can be used to compare documents with queries.

Similarity: Cosine; Euclidean distance

Relevance rankings of documents in a keyword search can be calculated, using the assumptions of document similarities theory, by comparing the deviation of angles between each document vector and the original query vector where the query is represented as the same kind of vector as the documents.



In practice, it is easier to calculate the cosine of the angle between the vectors, instead of the angle itself:

$$\cos\theta = \frac{\mathbf{d_2} \cdot \mathbf{q}}{\|\mathbf{d_2}\| \, \|\mathbf{q}\|}$$

In the interval [0, 1]

Where $\mathbf{d_2} \cdot \mathbf{q}$ is the intersection (i.e. the dot product) of the document (d_2 in the figure to the right) and the query (q in the figure) vectors, $\|\mathbf{d_2}\|$ is the norm of vector d_2, and $\|\mathbf{q}\|$ is the norm of vector q. The norm of a vector is calculated as such:

$$\|\mathbf{q}\| = \sqrt{\sum_{i=1}^{n} q_i^2}$$

As all vectors under consideration by this model are elementwise nonnegative, a cosine value of zero means that the query and document vector are orthogonal and have no match (i.e. the query term does not exist in the document being considered). See cosine similarity for further information.

NLP treat words as discrete atomic symbols

- **Natural language processing systems traditionally treat words as discrete atomic symbols, and therefore 'cat' may be represented as Id537 and 'dog' as Id143.**
  - These encodings are arbitrary, and provide no useful information to the system regarding the relationships that may exist between the individual symbols.
  - This means that the model can leverage very little of what it has learned about 'cats' when it is processing data about 'dogs' (such that they are both animals, four-legged, pets, etc.).
- **Representing words as unique, discrete ids furthermore leads to data sparsity, and usually means that we may need more data in order to successfully train statistical models.**
- **Using vector representations can overcome some of these obstacles.**

Vector space models (VSMs): synonym

- Embedding documents in a continuous vector space
- Vector space models (VSMs) represent (embed) words in a continuous vector space where semantically similar words are mapped to nearby points ('are embedded nearby each other').
- VSMs have a long, rich history in NLP, but all methods depend in some way or another on the Distributional Hypothesis, which states that words that appear in the same contexts share semantic meaning.
- The different approaches that leverage this principle can be divided into two categories: count-based methods (e.g. Latent Semantic Analysis), and predictive methods (e.g. neural probabilistic language models).

VSM: synonym detection

- **Distributional Hypothesis:**
  - states that words that appear in the same contexts share semantic meaning.
- **The different approaches that leverage this principle can be divided into two categories:**
  - count-based methods (e.g. Latent Semantic Analysis), and
  - predictive methods (e.g. neural probabilistic language models).

Count versus Predictive methods

- **This distinction is elaborated in much more detail by Baroni et al., but in a nutshell:**
- **Count-based methods compute the statistics of how often some word co-occurs with its neighbor words in a large text corpus, and then map these count-statistics down to a small, dense vector for each word.**
  - Clustering, SVD
- **Predictive models directly try to predict a word from its neighbors in terms of learned small, dense embedding vectors (considered parameters of the model).**
  - Word2Vec and its variations

Predictive method: Word2Vec

- **Word2vec is a particularly computationally-efficient predictive model for learning word embeddings from raw text.**
  - It comes in two flavors, the Continuous Bag-of-Words model (CBOW) and the Skip-Gram model. Algorithmically, these models are similar, except that CBOW predicts target words (e.g. 'mat') from source context words ('the cat sits on the'), while the skip-gram does the inverse and predicts source context-words from the target words.
  - This inversion might seem like an arbitrary choice, but statistically it has the effect that CBOW smoothes over a lot of the distributional information (by treating an entire context as one observation).
  - For the most part, this turns out to be a useful thing for smaller datasets. However, skip-gram treats each context-target pair as a new observation, and this tends to do better when we have larger datasets.

VSM

- **Embeddings/representations**
- **Clustering**
- **SVD**
- **Word2Vec (deep learning)**
  - Gradient descent

Live Session Outline

- **Housekeeping**
  - Please mute you microphones
  - Start RECORDING (bonus points for reminding me!)
- **Week 5**
  - Homework HW3, HW4, HW5
  - AWS (hope to have a access for week 6)
  - Async lecture recap plus Q&A [Jimi]
  - *Vector Space Model*
  - *Inverted index*
  - *Similarity*
  - Synonym detection in Map-Reduce
  - GMM
- **Wrapup**
  - Finish RECORDING (bonus points for reminding me!)
  - Click End Meeting

- **Inverted index**

Book index

Index

Index terms → Postings

Dial type [4, 14]

Inverted Index

- In computer science, an inverted index (also referred to as postings file or inverted file) is an index data structure storing a mapping from content, such as words or numbers, to its locations in a database file, or in a document or a set of documents (named in contrast to a Forward Index, which maps from documents to content).
- The purpose of an inverted index is to allow fast full text searches, at a cost of increased processing when a document is added to the database. The inverted file may be the database file itself, rather than its index. It is the most popular data structure used in document retrieval systems… [wikiPedia]

Using MapReduce to Construct Indexes

Index terms → Postings

- **Records: key value pairs**
  - DocID/URL, Content of document
- **Inverted Index**
  - Index terms → Postings
  - Hadoop → [Hadoop.org; apached.hadoop.org/manual;…..]
  - Mapper(Key, value)
    - Yield ()
  - Reducer()
    - Yield

Write MR Job

Using MapReduce to Construct Indexes

Index terms → Postings

- **Records: key value pairs**
  - DocID, Content of document
- **Inverted Index**
  - Index terms → Postings

ALWAYS!

MAPPER: (DocID, Content)
tokenize content
Yield (<term>,[<docid, TF>])

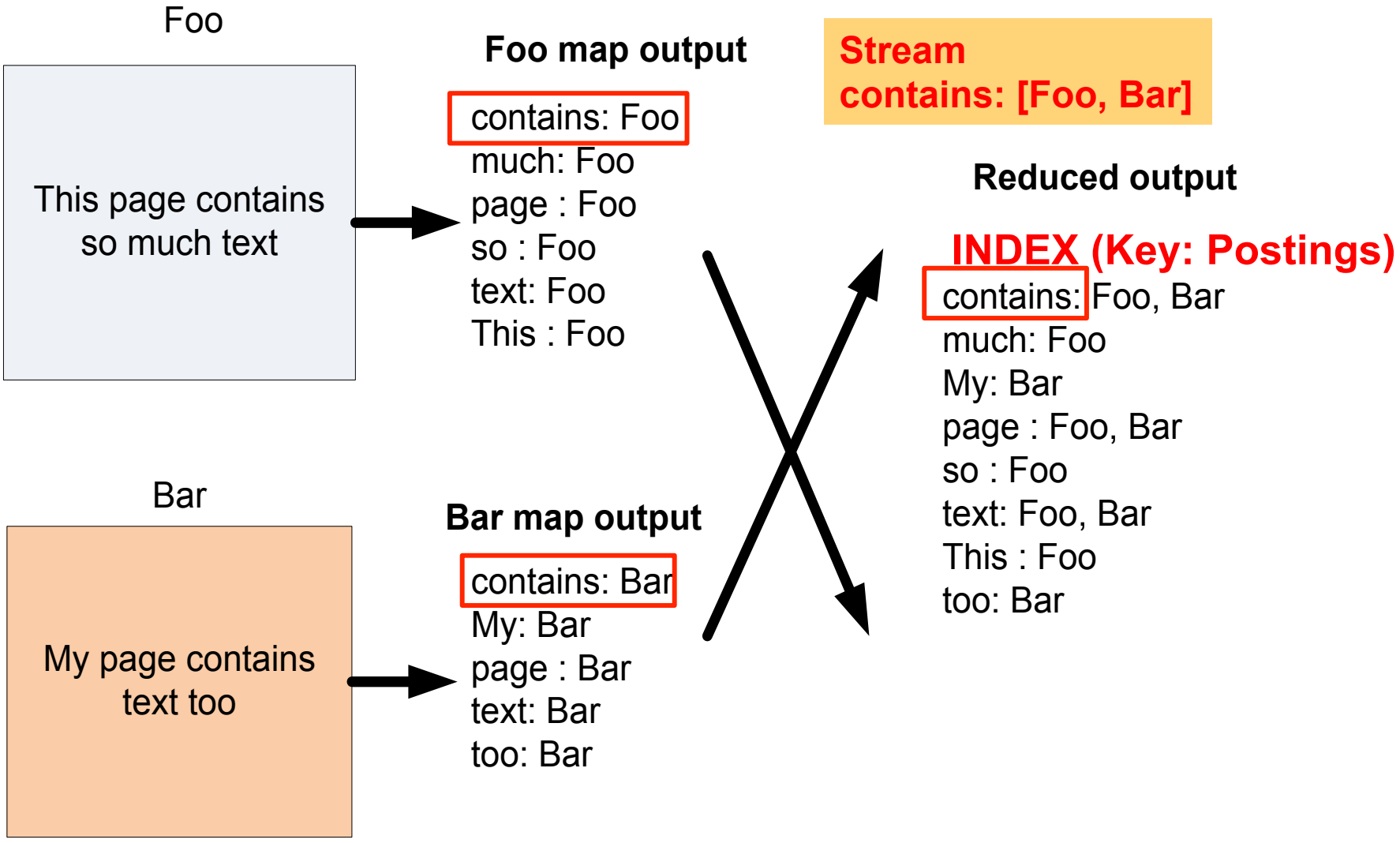
Combiner is the same as the reducer?

REDUCER: (<term>,[<docid, TF>….])
Yield (<term>,[<docid, TF>….])

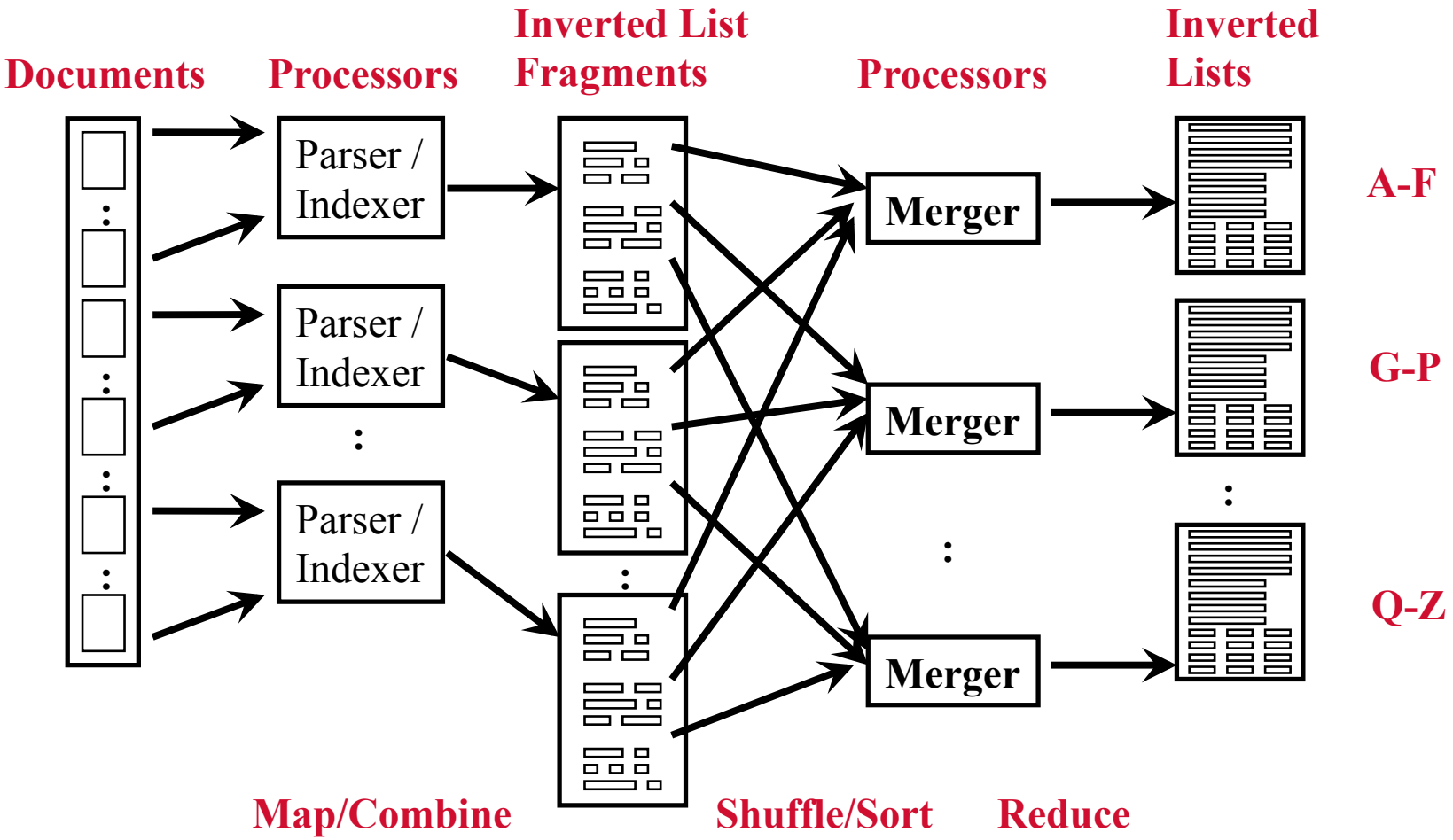
Will this work?

- **Mapper(docID, content of document)**
  - Yield <keyword, docID>, 1
- **Reducer(<keyword, docID>, count)**
  - yield <keyword, docID>, sum

Inverted Index: Data flow



Using MapReduce to Construct Indexes



- Solution to Inverted Index Code
- The following source code implements a solution to the inverted indexer problem posed at the checkpoint.
- The source code is structurally very similar to the source for Word Count; only a few lines really need to be modified.
- http://mapreduce-tutorial.blogspot.com/2011/04/solution-to-inverted-index-code.html

- JGS

Example TF IDF

In the classic vector space model proposed by Salton, Wong and Yang [1] the term-specific weights in the document vectors are products of local and global parameters. The model is known as term frequency-inverse document frequency model. The weight vector for document *d* is $\mathbf{v}_d = \left[w_{1,d}, w_{2,d}, \ldots, w_{N,d}\right]^T$, where

$$w_{t,d} = \mathrm{tf}_{t,d} \cdot \log \frac{|D|}{|\{d' \in D \,|\, t \in d'\}|}$$

and

- $\mathrm{tf}_{t,d}$ is term frequency of term *t* in document *d* (a local parameter)
- $\log \frac{|D|}{|\{d' \in D \,|\, t \in d'\}|}$ is inverse document frequency (a global parameter). $|D|$ is the total number of documents in the document set; $|\{d' \in D \,|\, t \in d'\}|$ is the number of documents containing the term *t*.

Using the cosine the similarity between document d_j and query *q* can be calculated as:

$$\mathrm{sim}(d_j, q) = \frac{\mathbf{d_j} \cdot \mathbf{q}}{\|\mathbf{d_j}\| \, \|\mathbf{q}\|} = \frac{\sum_{i=1}^{N} w_{i,j} w_{i,q}}{\sqrt{\sum_{i=1}^{N} w_{i,j}^2} \sqrt{\sum_{i=1}^{N} w_{i,q}^2}}$$

- ..

Advantages [edit]

The vector space model has the following advantages over the Standard Boolean model:

1. Simple model based on linear algebra
2. Term weights not binary
3. Allows computing a continuous degree of similarity between queries and documents
4. Allows ranking documents according to their possible relevance
5. Allows partial matching

Limitations [edit]

The vector space model has the following limitations:

1. Long documents are poorly represented because they have poor similarity values (a small scalar product and a large dimensionality)
2. Search keywords must precisely match document terms; word substrings might result in a "false positive match"
3. Semantic sensitivity; documents with similar context but different term vocabulary won't be associated, resulting in a "false negative match".
4. The order in which the terms appear in the document is lost in the vector space representation.
5. Theoretically assumes terms are statistically independent.
6. Weighting is intuitive but not very formal.

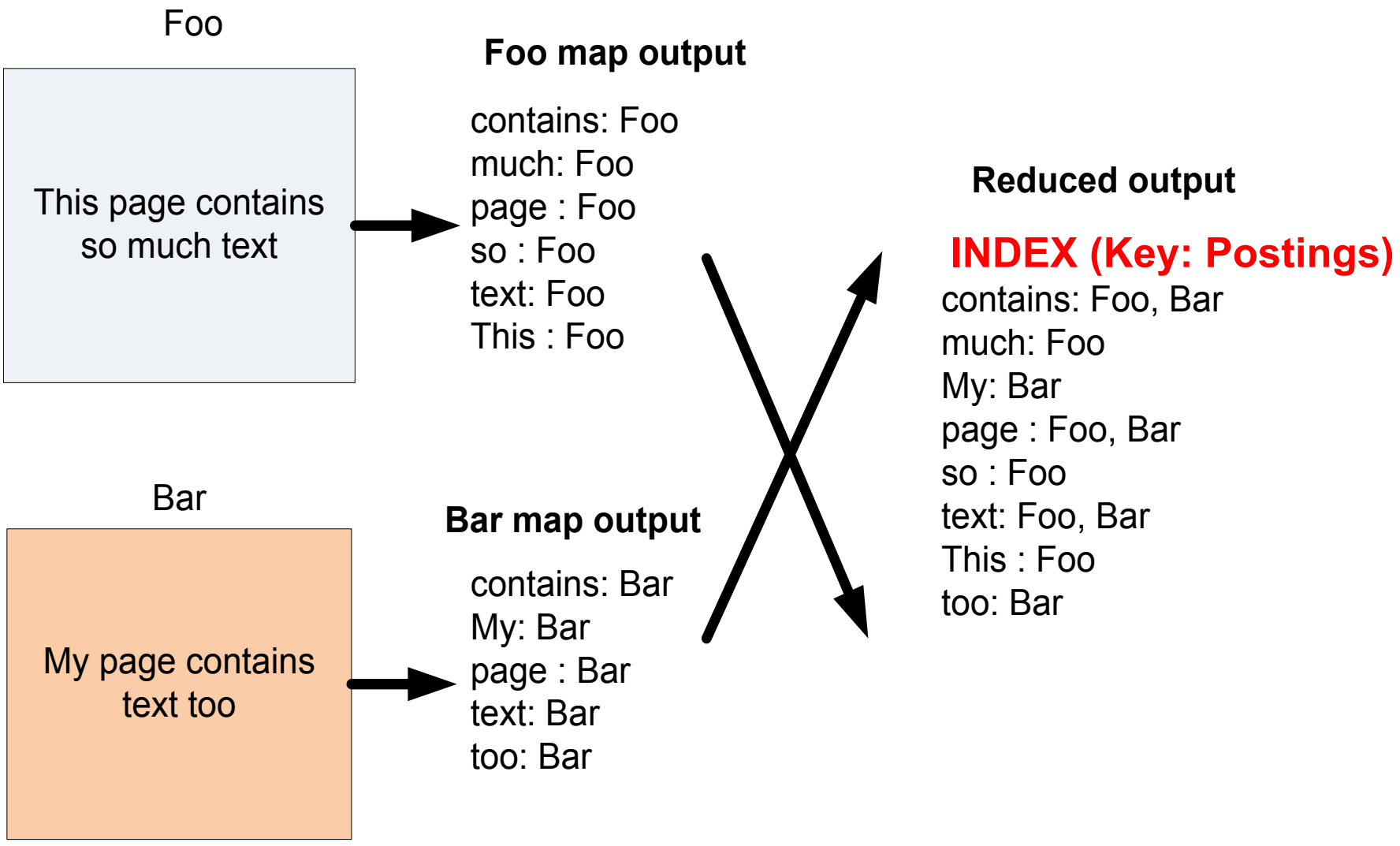
Many of these difficulties can, however, be overcome by the integration of various tools, including mathematical techniques such as singular value decomposition and lexical databases such as WordNet.

Information Retrieval and Map-Reduce Implementations

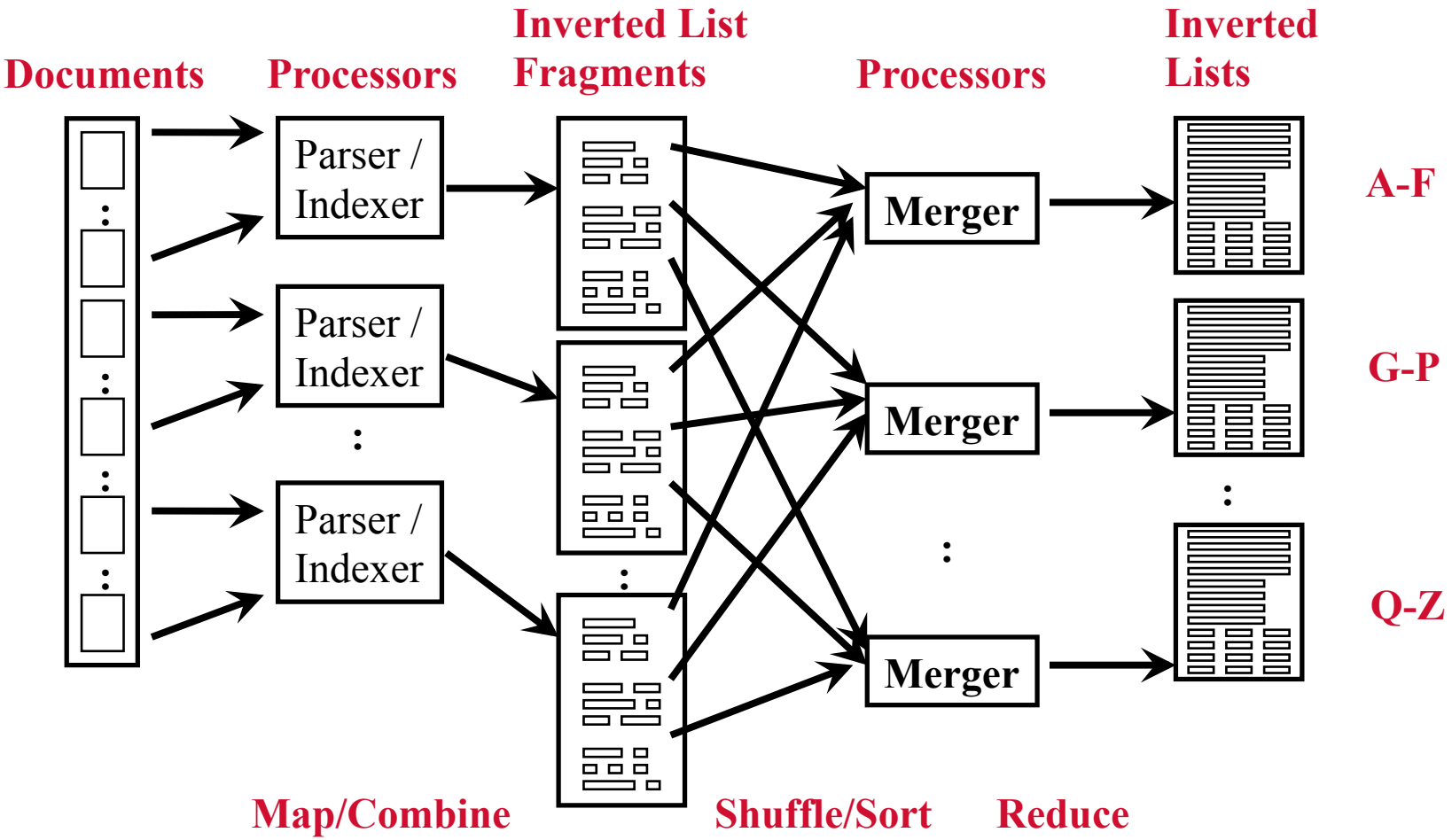
Adopted from Jimmy Lin's slides, which is licensed under a Creative Commons Attribution-Noncommercial-Share Alike 3.0 United States

- **Inverted index**

Inverted Index: Data flow



Using MapReduce to Construct Indexes



- Solution to Inverted Index Code
- The following source code implements a solution to the inverted indexer problem posed at the checkpoint.
- The source code is structurally very similar to the source for Word Count; only a few lines really need to be modified.
- http://mapreduce-tutorial.blogspot.com/2011/04/solution-to-inverted-index-code.html

- JGS

- **TF and IDF**

Information Retrieval and Map-Reduce Implementations

Adopted from Jimmy Lin's slides, which is licensed under a Creative Commons Attribution-Noncommercial-Share Alike 3.0 United States

Roadmap

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

Information Retrieval

- Examples (Average income in a city, Indexing, PageRank)
  - /Users/jshanahan/Publications/Lectures/TIM-251-2014-Spring/Lecture05-Graph-Algos-At-Scale/ESSIR_MapReduce_for_Indexing.pdf
  -

- Jimmy Lin's Slides
  - /Users/jshanahan/Publications/Lectures/TIM-251-2014-Spring/Lecture05-Graph-Algos-At-Scale/InformationRetrieval-via-MapReduce.pptx

Detailed Example of Index Construction

- http://www.google.com/url?sa=t&source=web&ct=html&cd=1&ved=undefined&url=http%3A%2F%2F74.125.155.132%2Fsearch%3Fq%3Dcache%3AEQjZmpSWs6oJ%3Acode.google.com%2Fedu%2Fsubmissions%2Fmapreduce%2Fhadoopcodelab.doc%2Bshuffle%2Bhadoop%26cd%3D1%26hl%3Den%26ct%3Dclnk%26gl%3Dus&ei=YzpBS9i7EiekswPw6vjWAw&usg=AFQjCNEddn_6wzTU3z-7KzIHc_PppZaikg&sig2=0MZJHAmOrPASIoG36Kxaow
- Construct IR system and run queries

Inverted index

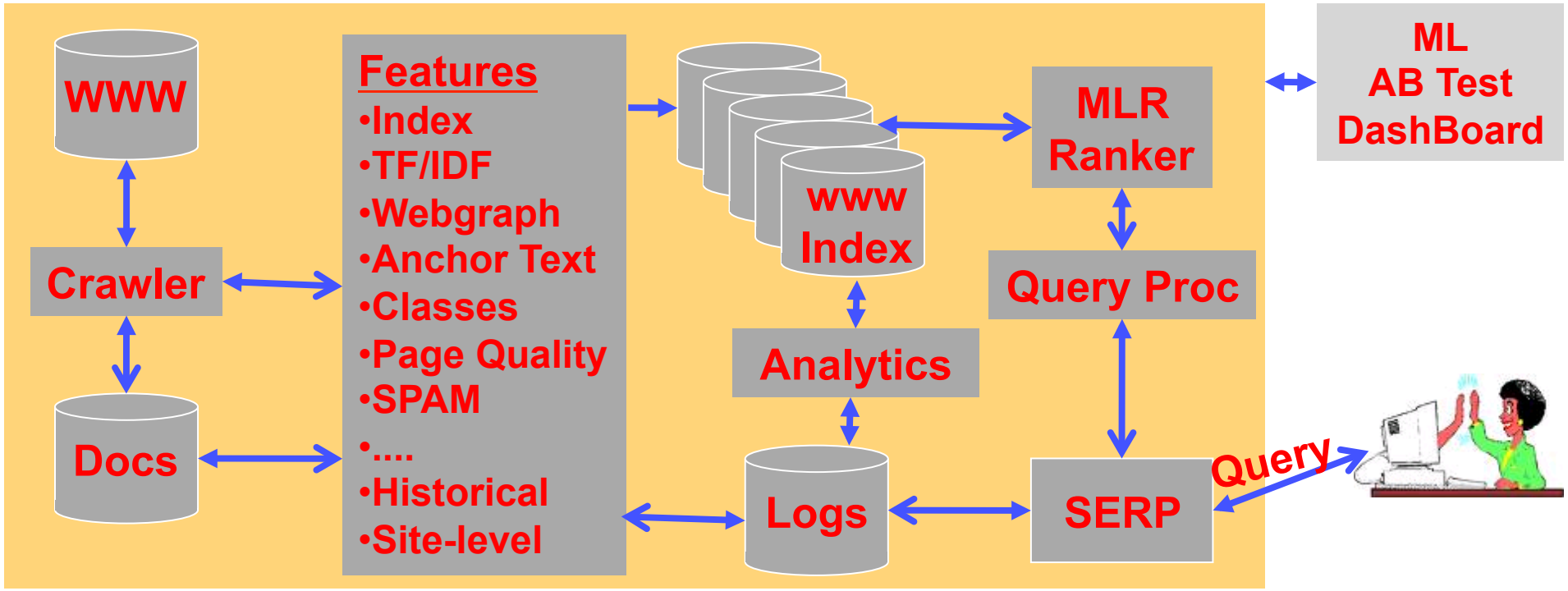
- …

term	offset	Documents
cold	2	1, 4
days	4	3, 6
hot	6	1, 4
like	8	4, 5
nine	10	3, 6
old	12	3, 6
pease	14	1, 2
porridge	16	1, 2
pot	18	2, 5
some	20	4, 5

dictionary

postings

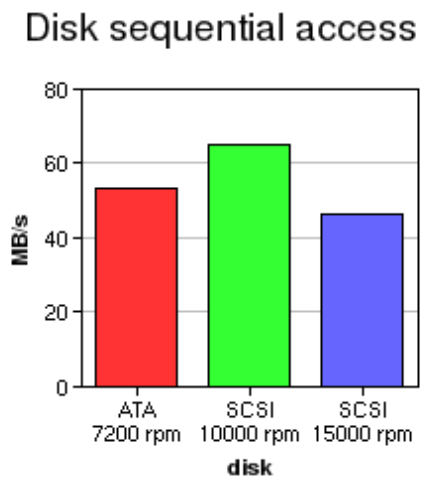
Search Engine Architecture



- Offline Processes
  - crawling, featurizing, Webgraph, Classification, updating ML models
- Online
  - query rewriting, ranking, reranking, merging, logging, analytics
- Realtime web indexing
- ML Framework

How to build Google in 30 mins?

- Look at index construction and retrieval (10Terabytes)
  - Retrieval is IO bound (limited)
  - Go from 28 hours to 2 milliseconds for a query round trip on
    - Store flat text files and access sequently; [28 hours to process a query]
    - Store an inverted index; [50 seconds]
    - Compress inverted index [7.5 (ignore disk latency, 7millisecs and CPU)]
    - Store an inverted index where posting entries are sorted by pagerank [2 millisecs]
- Adapted from Information Retrieval Modeling (IRM), Djoerd Hiemstra, University of Twente, RUSSIR 2009
  - http://romip.ru/russir2009/slides/irm/lecture6.pdf
  - lecture 6.



Inverted index

- …

term	offset	Documents
cold	2	1, 4
days	4	3, 6
hot	6	1, 4
like	8	4, 5
nine	10	3, 6
old	12	3, 6
pease	14	1, 2
porridge	16	1, 2
pot	18	2, 5
some	20	4, 5

dictionary

postings

- The inverted file entries are usually stored in order of increasing document number

  – [<*retrieval*; 7; [2, 23, 81, 98, 121, 126, 180]>

  (the term “retrieval” occurs in 7 documents with document identifiers 2, 23, 81, 98, etc.)

Query processing (3)

- Remember the Boolean model?
  - intersection, union and complement is done on posting lists
  - so, *information OR retrieval*

Information AND retrieval
23, 98

[<*retrieval*; 7; [2, 23, 81, 98, 121, 126, 139]>
[<*information*; 9; [1, 14, 23, 45, 46, 84, 98, 111, 120]>

  - union of posting lists gives:

[1, 2, 14, 23, 45, 46, 81, 84, 98, 111, 120, 121, 126, 139]

Inverted file compression (1)

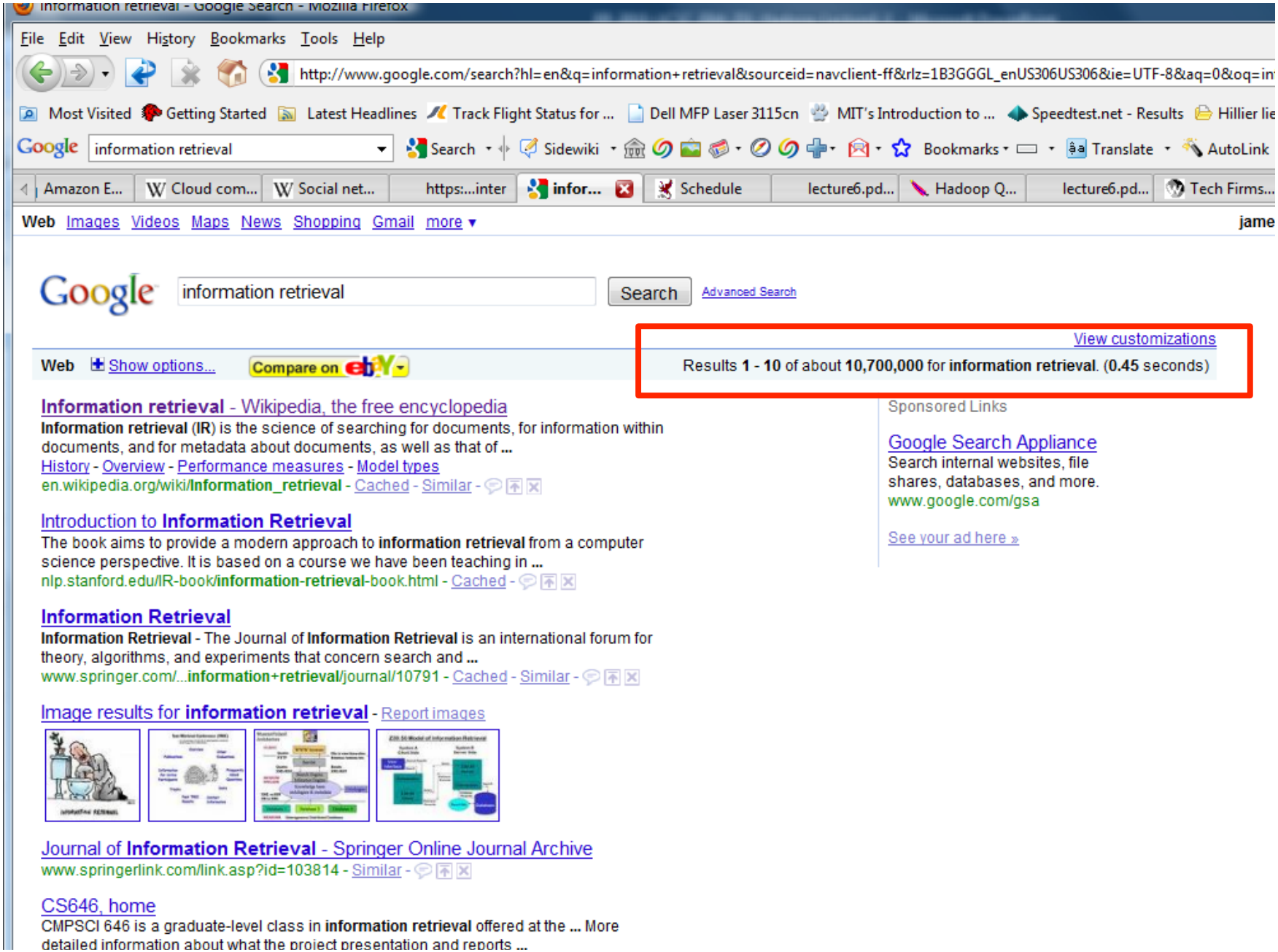
- Trick 1, store sequence of doc-ids:
  - [<*retrieval*; 7; [2, 23, 81, 98, 121, 126, 180]>

  as a sequence of gaps
  - [<*retrieval*; 7; [2, 21, 58, 17, 23, 5, 54]>

- No information is lost.
- Always process posting lists from the beginning so easily decoded into the original sequence

Query roundtrip from a day to 2 milliseconds

- Google's index size
  - 24 Million webpages in 1998 vs. 24 Billion in 2009 ($24 * 10^9$ webpages)
  - 180M queries per day or 2Million per second
  - Today it has 30 Trillion (10^{12}) unique webpages (230million sites) [WSJ, 1/16/2013]
- For simplicity lets assume 10^{10} web pages (10Billion pages) with about 200 terms on average where each term is 5 characters
  - 10^{10} x 200 x 5 ~= 10 TB to store raw text
  - 28 hours to process a query [load data to memory]
  - Store an inverted index; [50 seconds for query *information retrieval*]
  - Compress inverted index [7.5 seconds] using gaps and variable byte encoding
  - Store an inverted index where posting entries are sorted by pagerank [2 millisecs]
  - (ignore disk latency [7millisecs] and CPU time)


information retrieval - Google Search - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://www.google.com/search?hl=en&q=information+retrieval&sourceid=navclient-ff&rlz=1B3GGGL_enUS306US306&ie=UTF-8&aq=0&oq=in
Most Visited Getting Started Latest Headlines Track Flight Status for ... Dell MFP Laser 3115cn MIT's Introduction to ... Speedtest.net - Results Hillier lie
Google information retrieval Search Sidewiki Bookmarks Translate AutoLink
Amazon E... Cloud com... Social net... https:...inter infor... Schedule lecture6.pd... Hadoop Q... lecture6.pd... Tech Firms...
Web Images Videos Maps News Shopping Gmail more ▾ jame
Google information retrieval Search Advanced Search
View customizations
Web Show options... Compare on ebay
Results 1 - 10 of about 10,700,000 for information retrieval. (0.45 seconds)
Information retrieval - Wikipedia, the free encyclopedia
Information retrieval (IR) is the science of searching for documents, for information within documents, and for metadata about documents, as well as that of ...
History - Overview - Performance measures - Model types
en.wikipedia.org/wiki/Information_retrieval - Cached - Similar -
Introduction to Information Retrieval
The book aims to provide a modern approach to information retrieval from a computer science perspective. It is based on a course we have been teaching in ...
nlp.stanford.edu/IR-book/information-retrieval-book.html - Cached -
Information Retrieval
Information Retrieval - The Journal of Information Retrieval is an international forum for theory, algorithms, and experiments that concern search and ...
www.springer.com/...information+retrieval/journal/10791 - Cached - Similar -
Image results for information retrieval - Report images
Journal of Information Retrieval - Springer Online Journal Archive
www.springerlink.com/link.asp?id=103814 - Similar -
CS646, home
CMPSCI 646 is a graduate-level class in information retrieval offered at the ... More detailed information about what the project presentation and reports ...
Sponsored Links
Google Search Appliance
Search internal websites, file shares, databases, and more.
www.google.com/gsa
See your ad here »

Indexing Via MapReduce

- See following slides

- http://blog.cloudera.com/wp-content/uploads/2010/01/5-MapReduceAlgorithms.pdf

TF_IDF

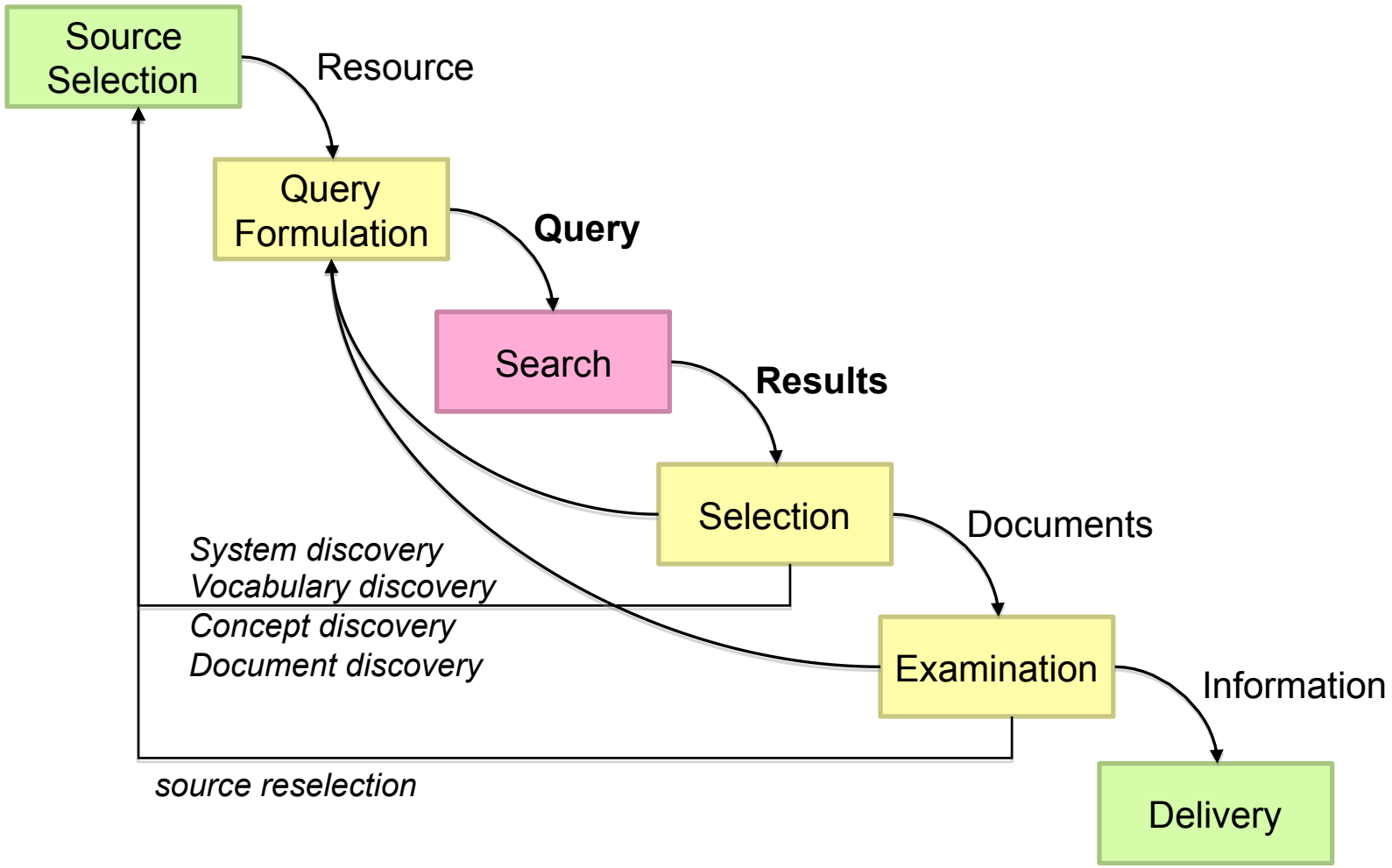
- See following slides

- http://blog.cloudera.com/wp-content/uploads/2010/01/5-MapReduceAlgorithms.pdf

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”

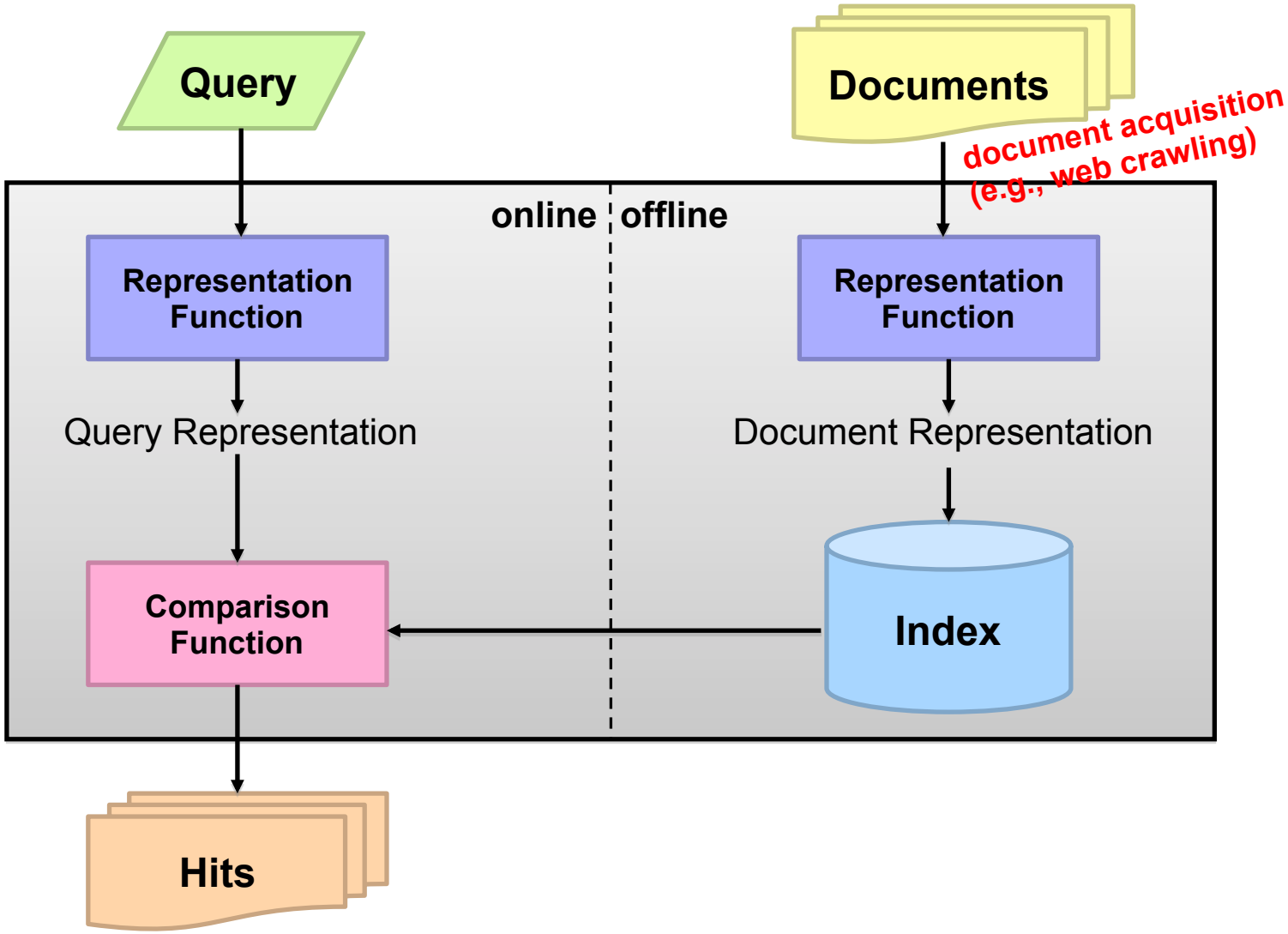
Concepts

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 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 आर्थकि सर्वेक्षण में वत्तिीय वर्ष 2005-06 में सात फ़ीसदी वकिास दर हासलि करने का आकलन कयिा है और कर सुधार पर ज़ोर दयिा 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.

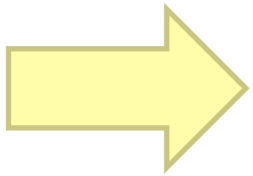
NEW YORK (CNN/Money) - McDonald's Corp. is cutting the amount of "bad" fat in its french fries nearly in half, the fast-food chain said Tuesday as it moves to make all its fried menu items healthier.

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

…

Information retrieval models

- An IR model governs how a document and a query are represented and how the relevance of a document to a user query is defined.
- Main models:
  - Boolean model
  - Vector space model
  - Statistical language model
  - etc

Boolean model

- Each document or query is treated as a **"bag" of words** or **terms**. Word sequence is not considered.
- Given a collection of documents D, let $V = \{t_1, t_2, ..., t_{|V|}\}$ be the set of distinctive words/terms in the collection. V is called the vocabulary.
- A weight $w_{ij} > 0$ is associated with each term t_i of a document $\mathbf{d}_j \in D$. For a term that does not appear in document $\mathbf{d}_j$, $w_{ij} = 0$.

$$\mathbf{d}_j = (w_{1j}, w_{2j}, ..., w_{|V|j}),$$

Boolean model (contd)

- Query terms are combined logically using the Boolean operators **AND**, **OR**, and **NOT.**
  - E.g., ((*data* AND *mining*) AND (NOT *text*))
- Retrieval
  - Given a Boolean query, the system retrieves every document that makes the query logically true.
  - Called **exact match**.
- The retrieval results are usually quite poor because term frequency is not considered.

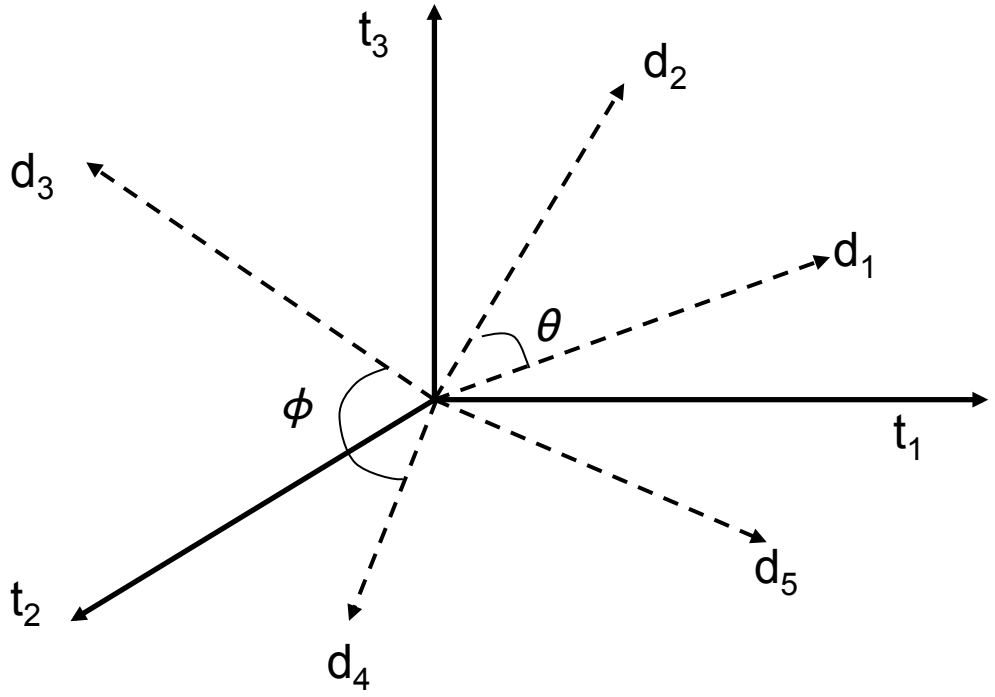
Boolean queries: Exact match

- The Boolean retrieval model is being able to ask a query that is a Boolean expression:
  - Boolean Queries are queries using *AND, OR* and *NOT* to join query terms
    - Views each document as a set of words
    - Is precise: document matches condition or not.
  - Perhaps the simplest model to build an IR system on
- Primary commercial retrieval tool for 3 decades.
- Many search systems you still use are Boolean:
  - Email, library catalog, Mac OS X Spotlight

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

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:

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

Vector space model

- Documents are also treated as a "bag" of words or terms.
- Each document is represented as a vector.
- However, the term weights are no longer 0 or 1. Each term weight is computed based on some variations of TF or TF-IDF scheme.

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

10^9 Documents in corpus
"information" occurs in 10^6
WHAT IS THE IDF USING LOG TO BASE 10?

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

$w_{i,j}$ **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***

$$tf_{ij} = \frac{f_{ij}}{\max\{f_{1j}, f_{2j}, \ldots, f_{|V|j}\}}$$

Retrieval in vector space model

- Query **q** is represented in the same way or slightly differently.
- Relevance of $\mathbf{d}_j$ to **q**: Compare the similarity of query **q** and document $\mathbf{d}_j$.
- Cosine similarity (the cosine of the angle between the two vectors)

$$cosine(\mathbf{d}_j, \mathbf{q}) = \frac{\langle \mathbf{d}_j \bullet \mathbf{q} \rangle}{\| \mathbf{d}_j \| \times \| \mathbf{q} \|} = \frac{\sum_{i=1}^{|V|} w_{ij} \times w_{iq}}{\sqrt{\sum_{i=1}^{|V|} w_{ij}^2} \times \sqrt{\sum_{i=1}^{|V|} w_{iq}^2}}$$

- Cosine is also commonly used in text clustering

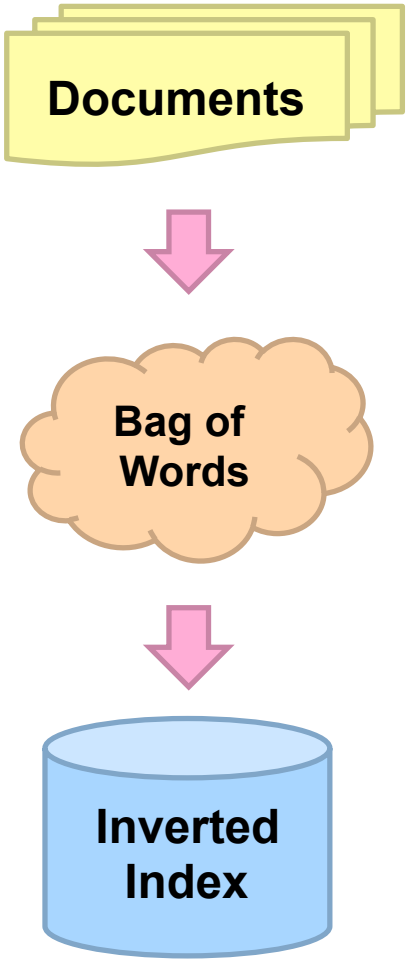
An Example

- A document space is defined by three terms:
  - hardware, software, users
  - the vocabulary
- A set of documents are defined as:
  - A1=(1, 0, 0), A2=(0, 1, 0), A3=(0, 0, 1)
  - A4=(1, 1, 0), A5=(1, 0, 1), A6=(0, 1, 1)
  - A7=(1, 1, 1) A8=(1, 0, 1). A9=(0, 1, 1)
- If the Query is "hardware and software"
- what documents should be retrieved?

An Example (cont.)

- In Boolean query matching:
  - document A4, A7 will be retrieved ("AND")
  - retrieved: A1, A2, A4, A5, A6, A7, A8, A9 ("OR")
- In similarity matching (cosine):
  - q=(1, 1, 0)
  - S(q, A1)=0.71, S(q, A2)=0.71, S(q, A3)=0
  - S(q, A4)=1, S(q, A5)=0.5, S(q, A6)=0.5
  - S(q, A7)=0.82, S(q, A8)=0.5, S(q, A9)=0.5
  - Document retrieved set (with ranking)=
    - {A4, A7, A1, A2, A5, A6, A8, A9}

Constructing Inverted Index (Word Counting)



case folding, tokenization, stopword removal, stemming

sy ax, sem tics, word k wledge, etc.

Stopwords removal

- Many of the most frequently used words in English are useless in IR and text mining – these words are called *stop words*.
  - the, of, and, to, ….
  - Typically about 400 to 500 such words
  - For an application, an additional domain specific stopwords list may be constructed
- Why do we need to remove stopwords?
  - Reduce indexing (or data) file size
    - stopwords accounts 20-30% of total word counts.
  - Improve efficiency and effectiveness
    - stopwords are not useful for searching or text mining
    - they may also confuse the retrieval system.

Stemming

- Techniques used to find out the root/stem of a word. E.g.,
  - user — engineering
  - users — engineered
  - used — engineer
  - using
- stem: use — engineer

Usefulness:

- improving effectiveness of IR and text mining
  - matching similar words
  - Mainly improve recall
- reducing indexing size
  - combing words with same roots may reduce indexing size as much as 40-50%.

Basic stemming methods

Using a set of rules. E.g.,

- remove ending
  - if a word ends with a consonant other than s,
    followed by an s, then delete s.
  - if a word ends in es, drop the s.
  - if a word ends in ing, delete the ing unless the remaining word consists only of one letter or of th.
  - If a word ends with ed, preceded by a consonant, delete the ed unless this leaves only a single letter.
  - ……
- transform words
  - if a word ends with “ies” but not “eies” or “aies” then “ies --> y.”

Inverted index

- The inverted index of a document collection is basically a data structure that
  - attaches each distinctive term with a list of all documents that contains the term.
- Thus, in retrieval, it takes constant time to
  - find the documents that contains a query term.
  - multiple query terms are also easy handle as we will see soon.

An example

Example 3: We have three documents of id_1, id_2, and id_3:

id_1: Web mining is useful.
1 2 3 4

id_2: Usage mining applications.
1 2 3

id_3: Web structure mining studies the Web hyperlink structure.
1 2 3 4 5 6 7 8

Applications:	id_2
Hyperlink:	id_3
Mining:	id_1, id_2, id_3
Structure:	id_3
Studies:	id_3
Usage:	id_2
Useful:	id_1
Web:	id_1, id_3

(A)

Applications:	<id_2, 1, [3]>
Hyperlink:	<id_3, 1, [7]>
Mining:	<id_1, 1, [2]>, <id_2, 1, [2]>, <id_3, 1, [3]>
Structure:	<id_3, 2, [2, 8]>
Studies:	<id_3, 1, [4]>
Usage:	<id_2, 1, [1]>
Useful:	<id_1, 1, [4]>
Web:	<id_1, 1, [1]>, <id_3, 2, [1, 6]>

(B)

<DocID, TF, position>

Fig. 6.7. Two inverted indices: a simple version and a more complex version

Search using inverted index

Given a query **q**, search has the following steps:

- Step 1 (**vocabulary search**): find each term/word in **q** in the inverted index.
- Step 2 (**results merging)**: Merge results to find documents that contain all or some of the words/terms in **q**.
- Step 3 (**Rank score computation**): To rank the resulting documents/pages, using,
  - content-based ranking
  - link-based ranking

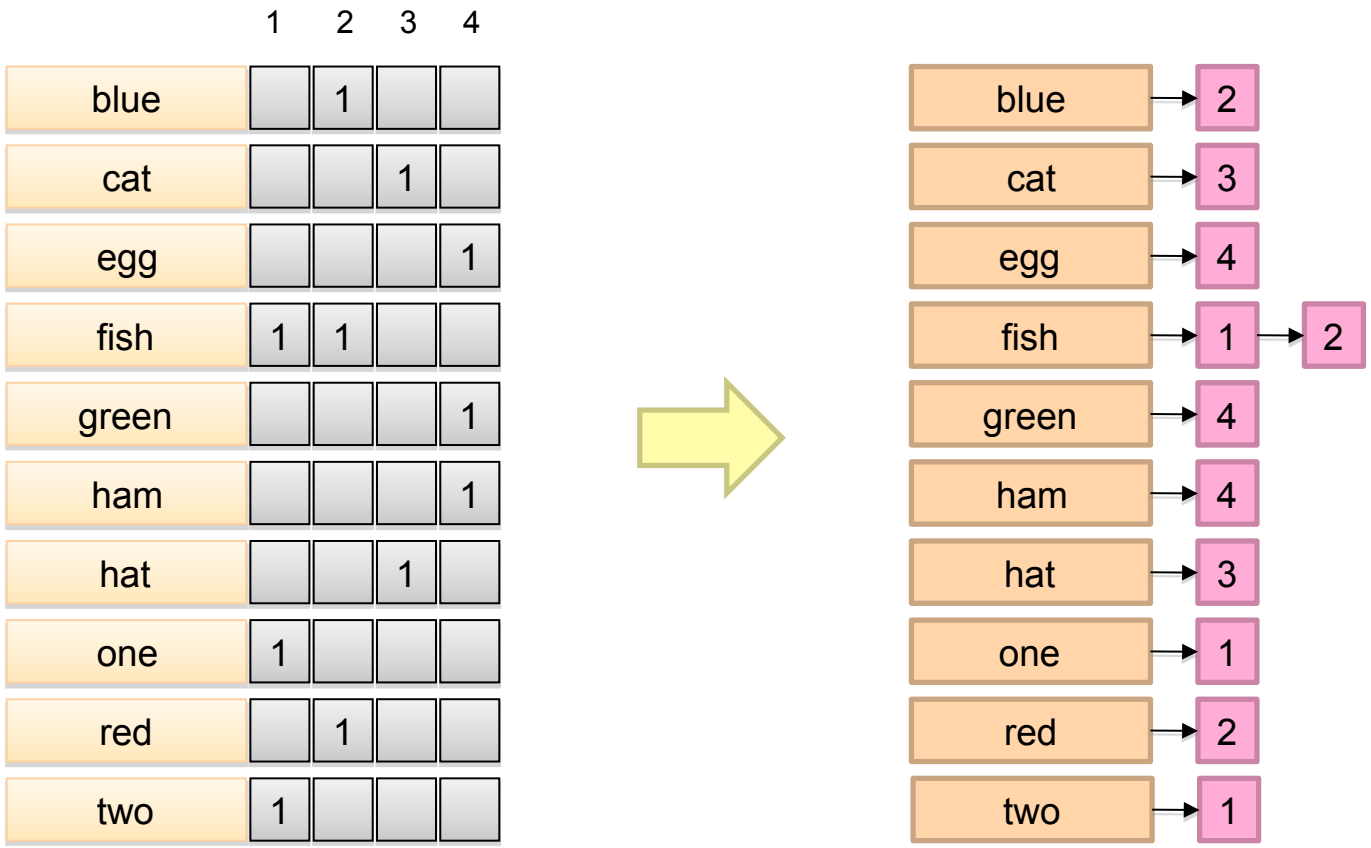
Inverted Index: Boolean Retrieval

Doc 1
one fish, two fish

Doc 2
red fish, blue fish

Doc 3
cat in the hat

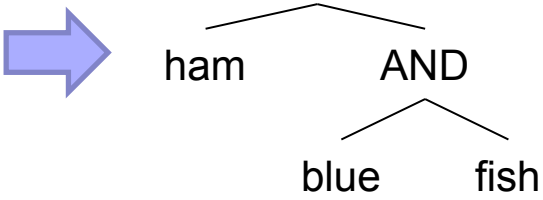
Doc 4
green eggs and ham



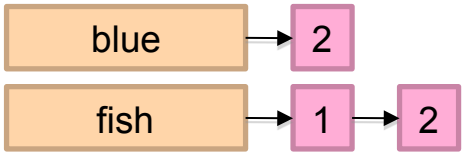
Boolean Retrieval

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

(blue AND fish) OR ham



  - For each clause, look up postings



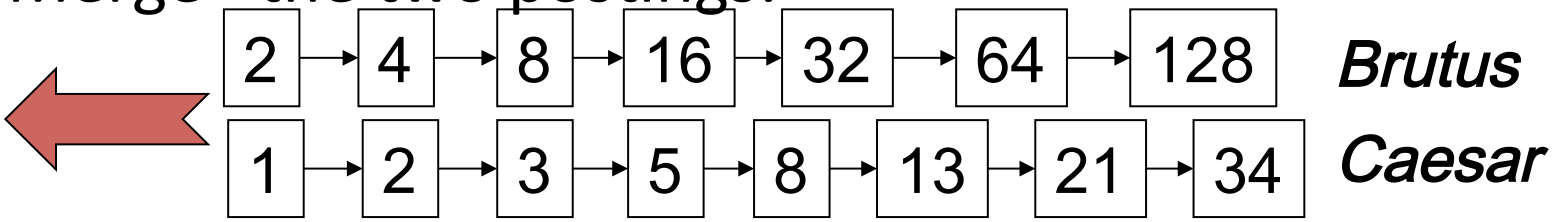
  - Traverse postings and apply Boolean operator
- Efficiency analysis
  - Postings traversal is linear (assuming sorted postings)
  - Start with shortest posting first

Query processing: AND

- Consider processing the query:

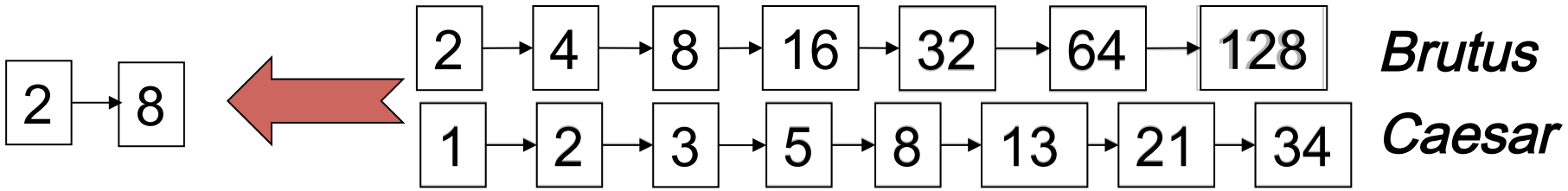
  Brutus *AND* ***Caesar***

  – Locate ***Brutus*** in the Dictionary;
    - Retrieve its postings.
  – Locate *Caesar* in the Dictionary;
    - Retrieve its postings.
  – "Merge" the two postings:



The merge

- Walk through the two postings simultaneously, in time linear in the total number of postings entries



If the list lengths are *x* and *y*, the merge takes O(*x*+*y*) operations.

Crucial: postings sorted by docID.

Intersecting two postings lists (a "merge" algorithm)

```
INTERSECT(p1, p2)
 1  answer ← ⟨ ⟩
 2  while p1 ≠ NIL and p2 ≠ NIL
 3  do if docID(p1) = docID(p2)
 4        then ADD(answer, docID(p1))
 5             p1 ← next(p1)
 6             p2 ← next(p2)
 7        else if docID(p1) < docID(p2)
 8                then p1 ← next(p1)
 9                else p2 ← next(p2)
10  return answer
```

Inverted Index: TF.IDF

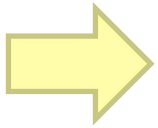
Doc 1 one fish, two fish

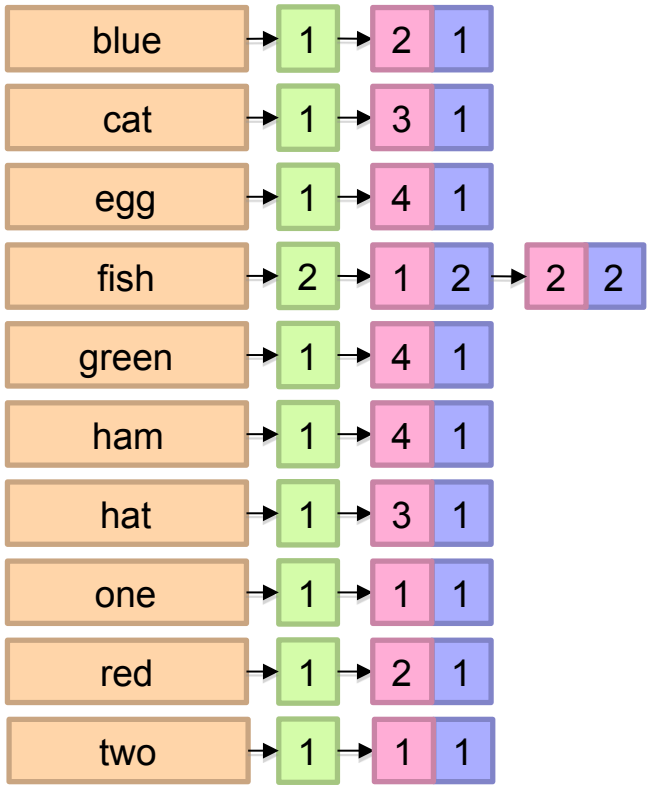
Doc 2 red fish, blue fish

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…

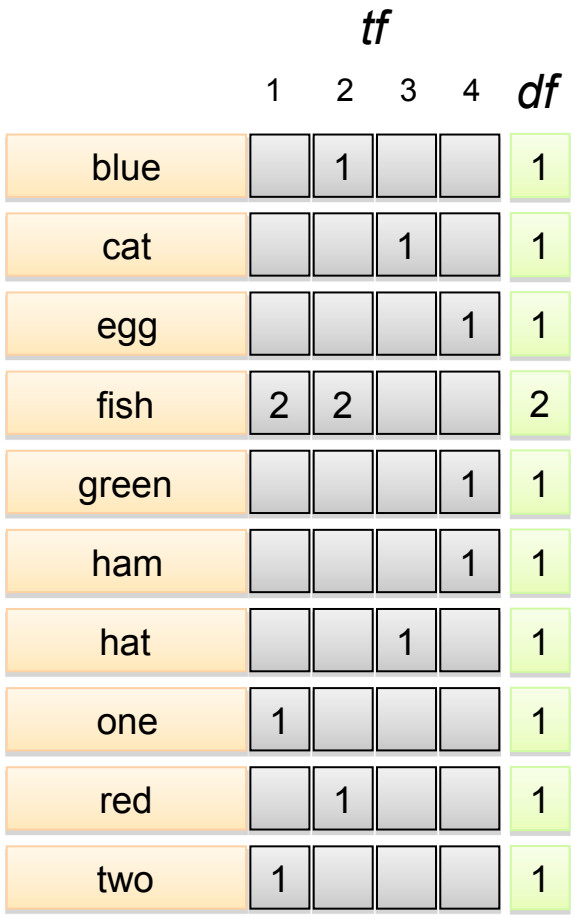
Inverted Index: Positional Information

Doc 1
one fish, two fish

Doc 2
red fish, blue fish

Doc 3
cat in the hat

Doc 4
green eggs and ham

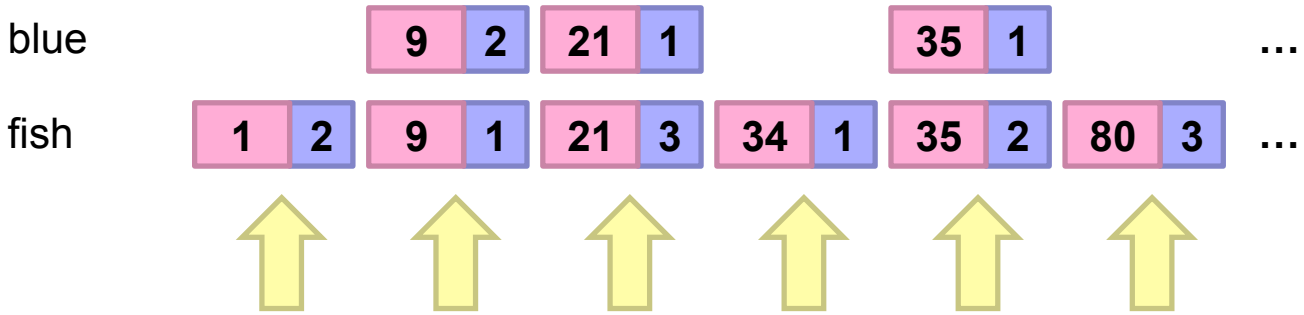


Retrieval in a Nutshell

- 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)

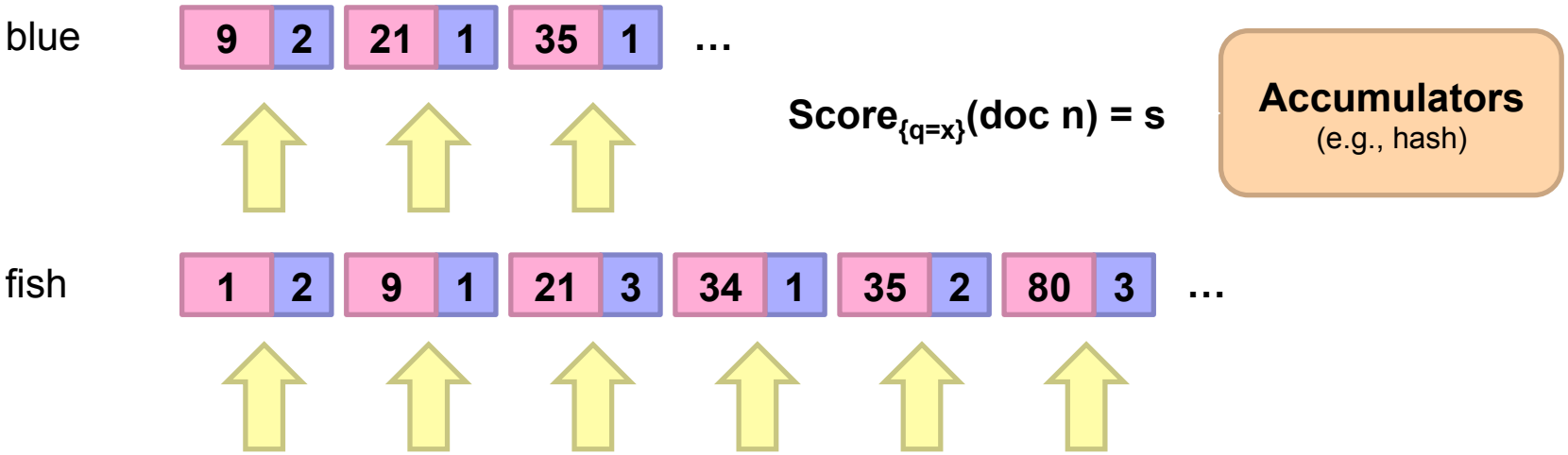
Document score in top k?

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

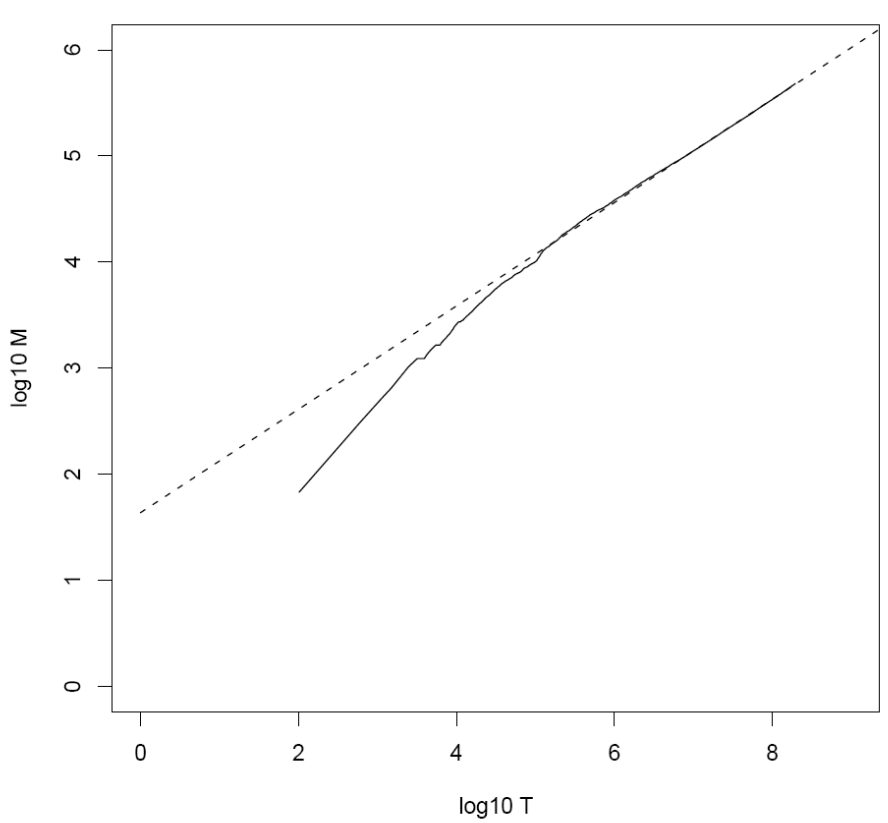
$$M = kT^b$$

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 states that given some corpus of natural language utterances, the frequency of any word is inversely proportional to rank in the frequency table. Thus the most frequent word will occur approximately twice as often as the second most frequent three times as often as the third most frequent word, etc.: the rank-frequency distribution is an inverse relation. For example, in Brown Corpus of American English text, the word "the" is the most frequently occurring word, and by itself accounts for nearly all word occurrences (69,971 out of slightly over 1 million). True to Zipf's Law, the second-place word "of" accounts for slightly 3.5% of words (36,411 occurrences), followed by "and" (28,852). Only 135 vocabulary items are needed to account for half the Brown Corpus.[4]

- 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

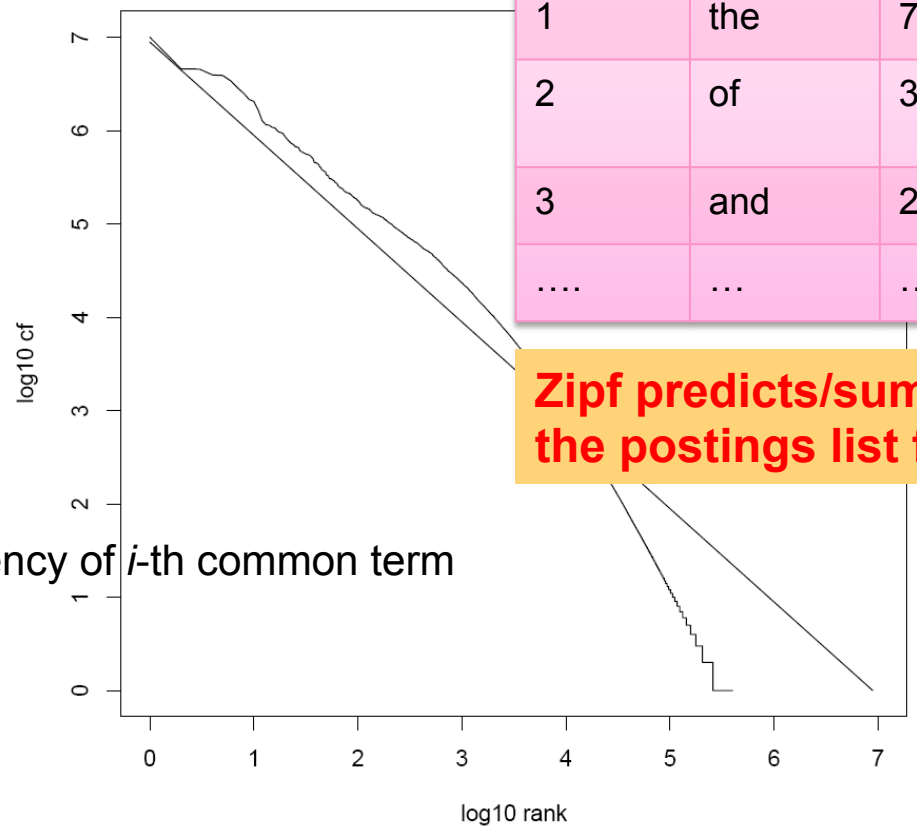
- The same relationship occurs in many other rankings unrelated to language, such as the population ranks of cities in various countries, corporation sizes, income rankings, ranks of number of people watching the same TV channel, and so on.
- The appearance of the distribution in rankings of cities by population was first noticed by Felix Auerbach in 1913. Empirically, a data set can be tested to see whether Zipf's law applies by checking the goodness of fit of an empirical distribution to the hypothesized power law distribution with a Kolmogorov-Smirnov test, and then comparing the (log) likelihood ratio of the power law distribution to alternative distributions like an exponential distribution or lognormal distribution.[6] When Zipf's law is checked for cities, a better fit has been found with b = 1.07; i.e. the n^{th}

Zipf's Law for RCV1

Word1, Number of Docs (sort in decr orde
The 10,000, 1
System, 5,000, 2
Hadoop, 2500, 3
…..

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

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



Brown Corpus of English
https://en.wikipedia.org/wiki/Zipf%27s_law

Rank	Word	Freq	Zipf's Prediction
1	the	70,000	70000
2	of	36,000	70,000/2c
3	and	28,000	70,000/3c
….	…	….	../4²

Zipf predicts/summarizes the size of the postings list for a term

Fit isn't that good… but good enough!

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

Zipf's Law can predict the frequency of terms
nth largest frequency is $1/n^{1.07}$ the size of the largest frequency

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

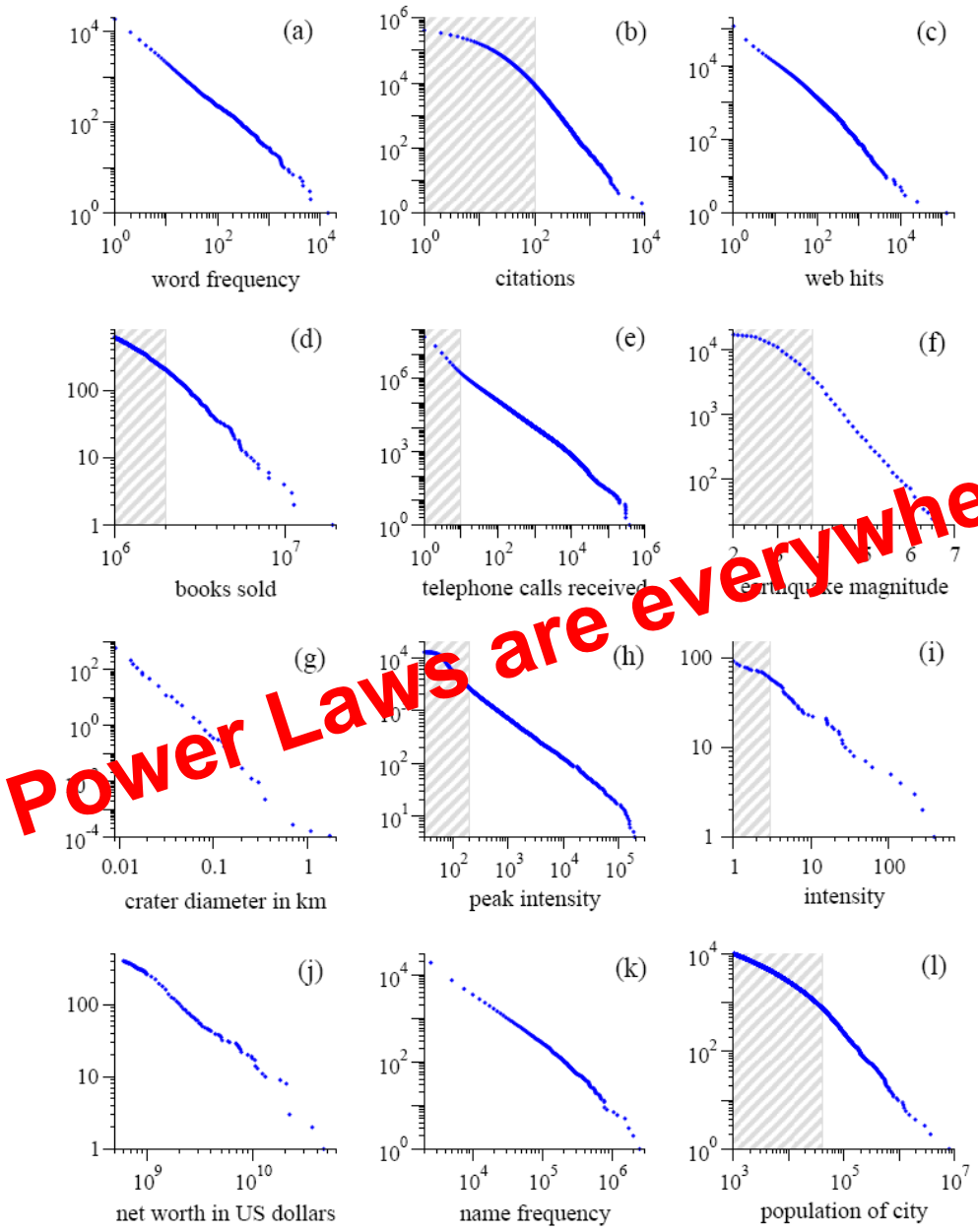
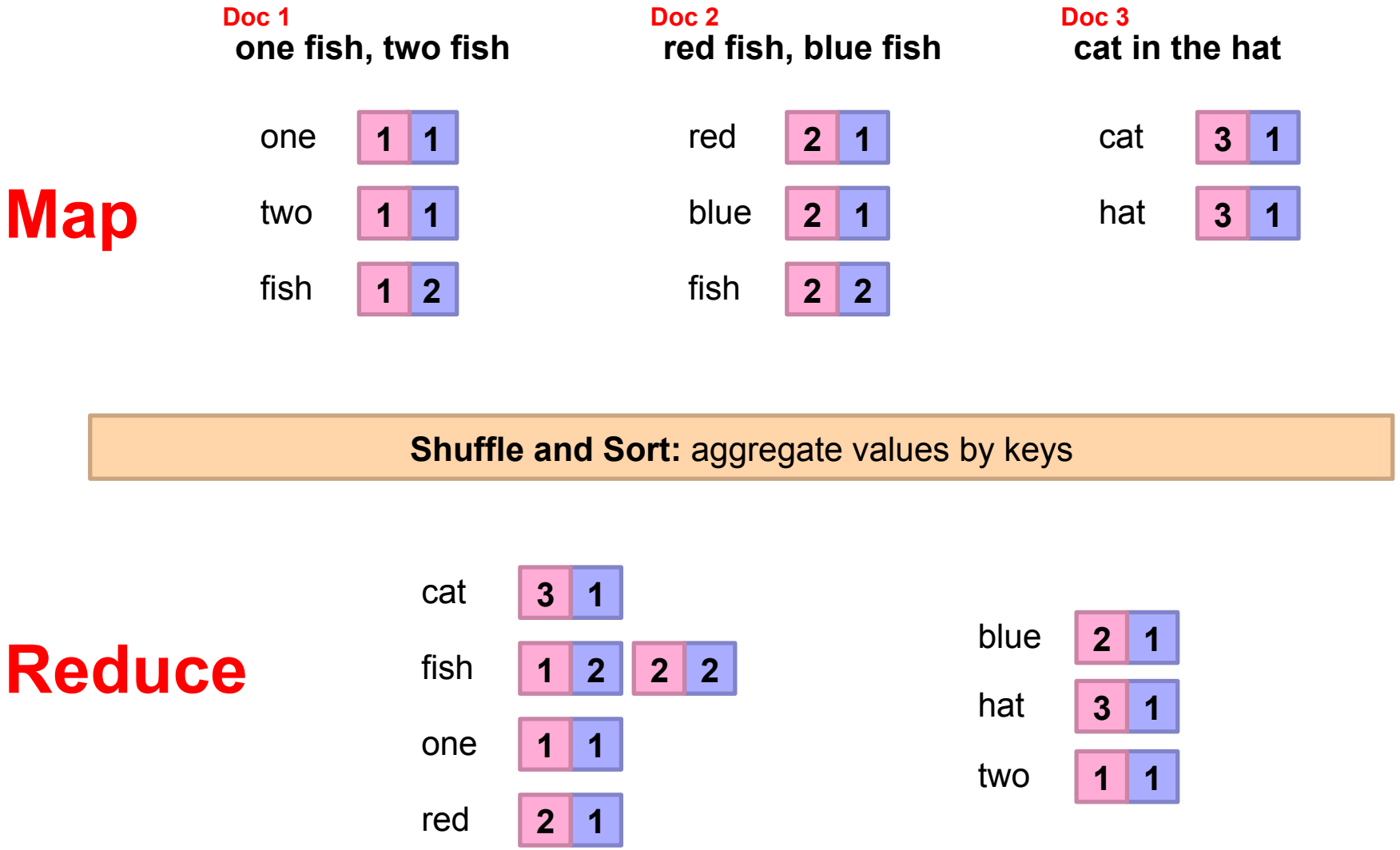


Figure from: Newman, M. E. J. (2005) “Power laws, Pareto distributions and Zipf's law.” Contemporary Physics 46:323–351.

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}⟩)
```

Term, posting<docId, TF>

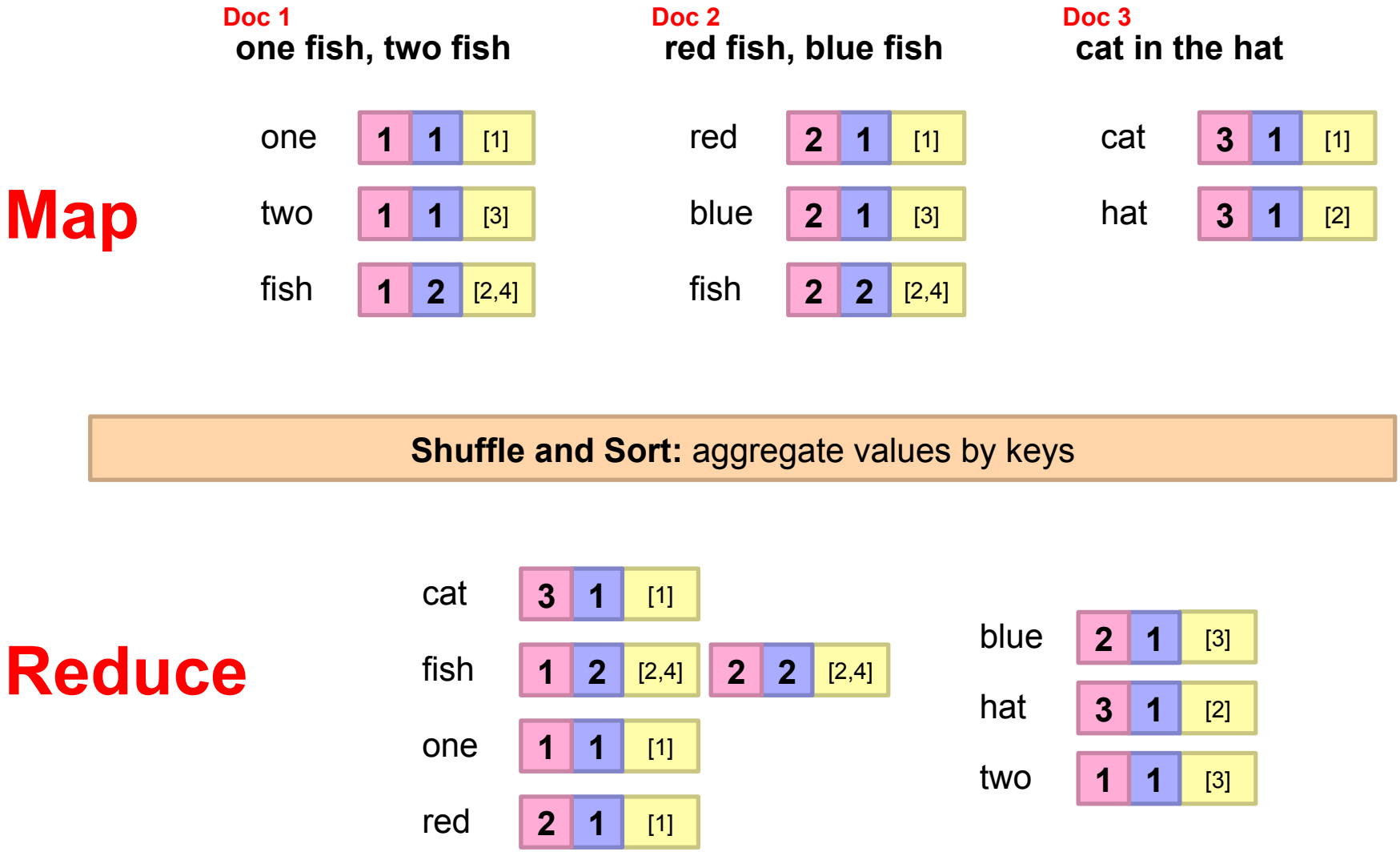
```
1: class Reducer
2:     procedure Reduce(term t, postings [⟨a1, f1⟩, ⟨a2, f2⟩ ...])
3:         P ← new List
4:         for all posting ⟨a, f⟩ ∈ postings [⟨a1, f1⟩, ⟨a2, f2⟩ ...] do
5:             Append(P, ⟨a, f⟩)
6:         Sort(P)
7:         Emit(term t, postings P)
```

Term,concatenate individual posting<docId, T

Sort by doc id

Emit term t and postings

Positional Indexes



Inverted Indexing: Pseudo-Code

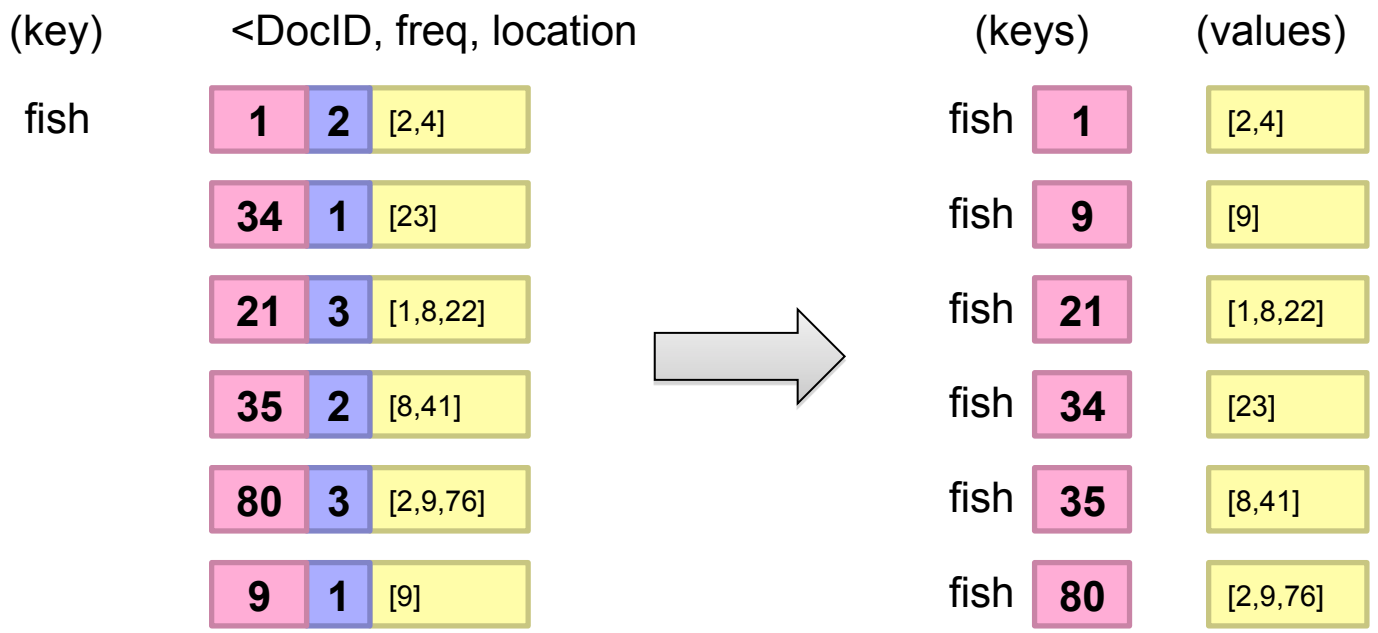
```
1: class Mapper
2:     procedure Map(docid n, doc d)
3:         H ← new AssociativeArray
4:         for all term t ∈ doc d do
5:             H{t} ← H{t} + 1
6:         for all term t ∈ H do
7:             Emit(term t, posting ⟨n, H{t}⟩)

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… secondary sorting



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
  - VarInt
  - Group VarInt
  - Simple-9
- 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!

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

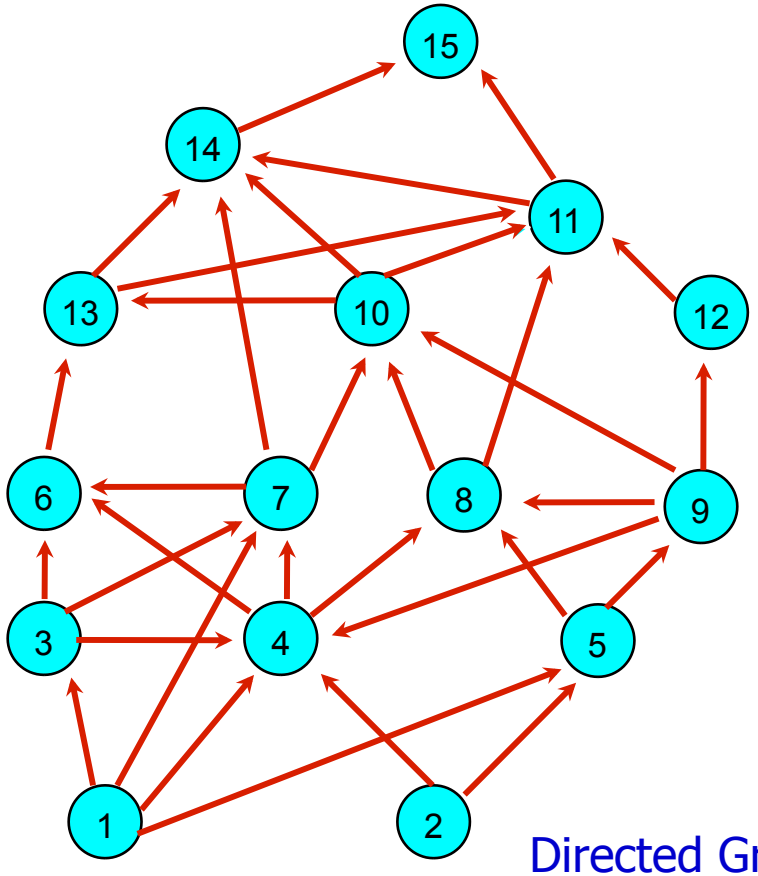
← Recommend best practice

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)

Reachability Query and Transitive Closure Representation

The problem: Given two vertices u and v in a directed graph G, is there a path from u to v ?



?Query(1,11)

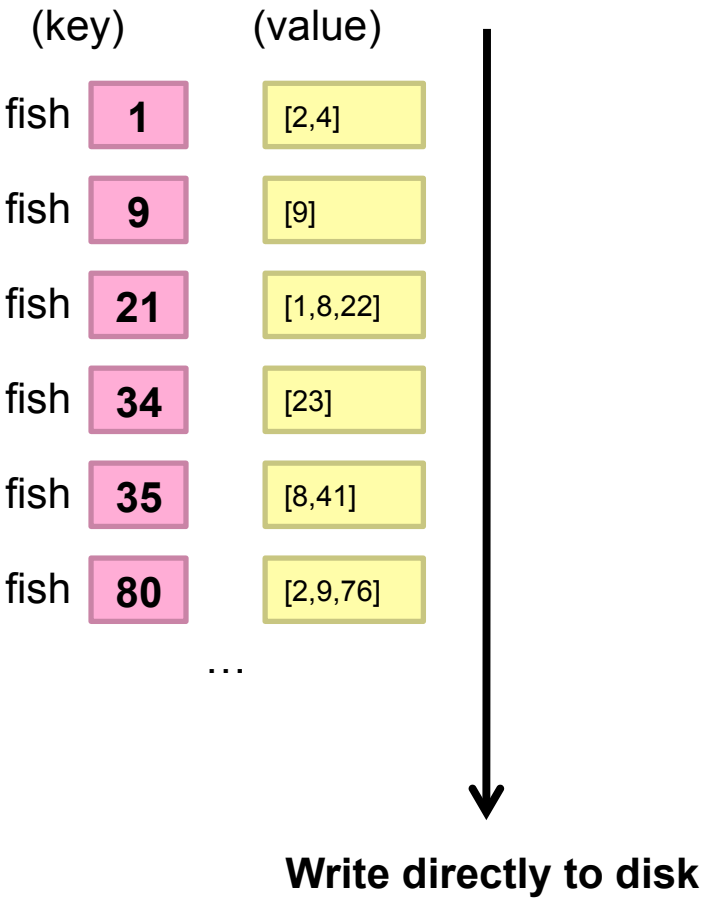
Yes

?Query(3,9)

No

Directed Graph → DAG (directed acyclic graph) by coalescing the strongly connected components

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!

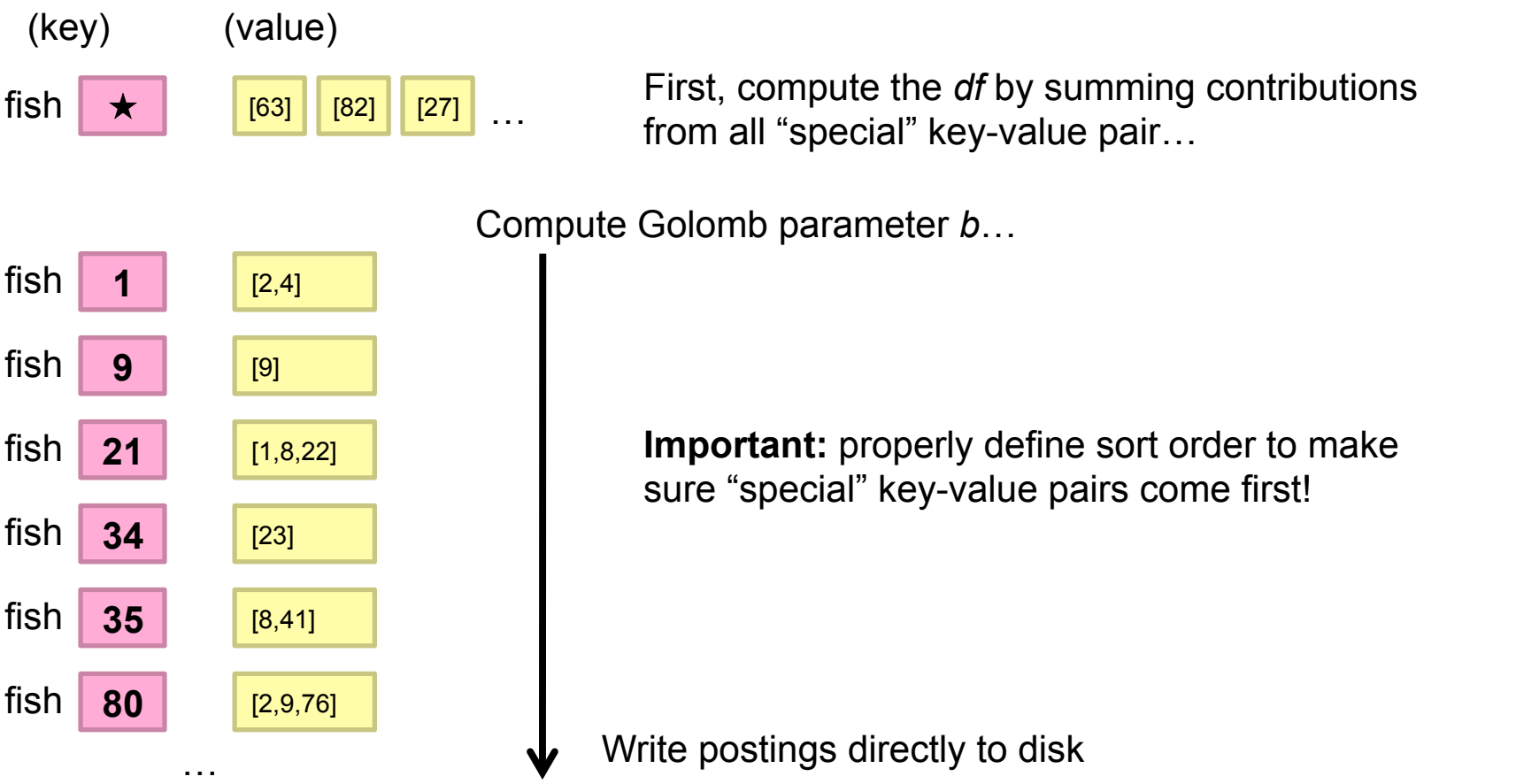
Getting the df: Modified Mapper

Doc 1
one fish, two fish

Input document…

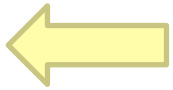
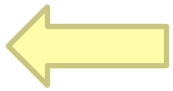
(key)		(value)	
fish	1	[2,4]	Emit normal key-value pairs…
one	1	[1]	
two	1	[3]	
fish	★	[1]	Emit “special” key-value pairs to keep track of *df*…
one	★	[1]	
two	★	[1]	

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?

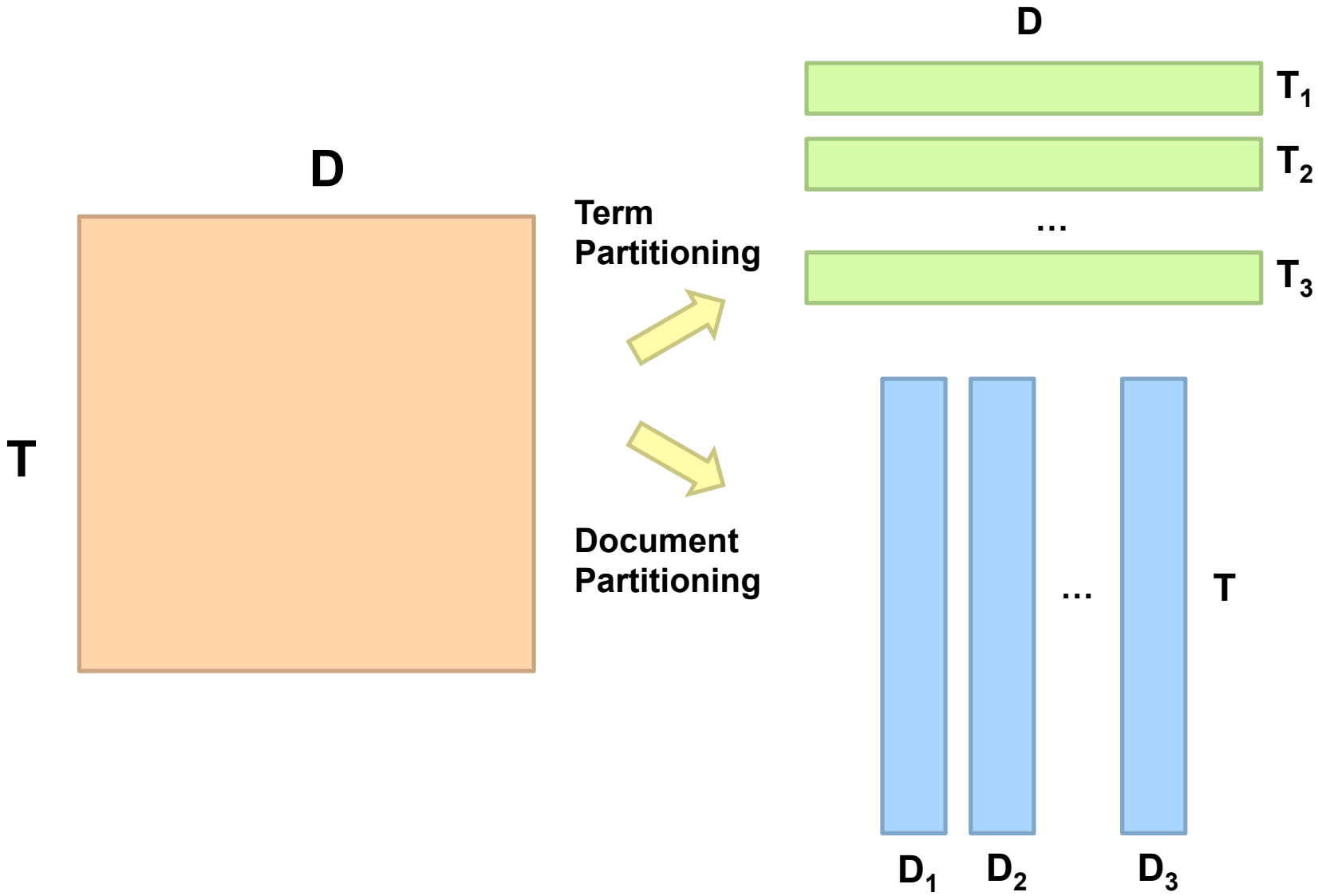
- MapReduce is fundamentally batch-oriented
  - Optimized for throughput, not latency
  - Startup of mappers and reducers is expensive
- MapReduce is not suitable for real-time queries!
  - Use separate infrastructure for retrieval…

Important Ideas

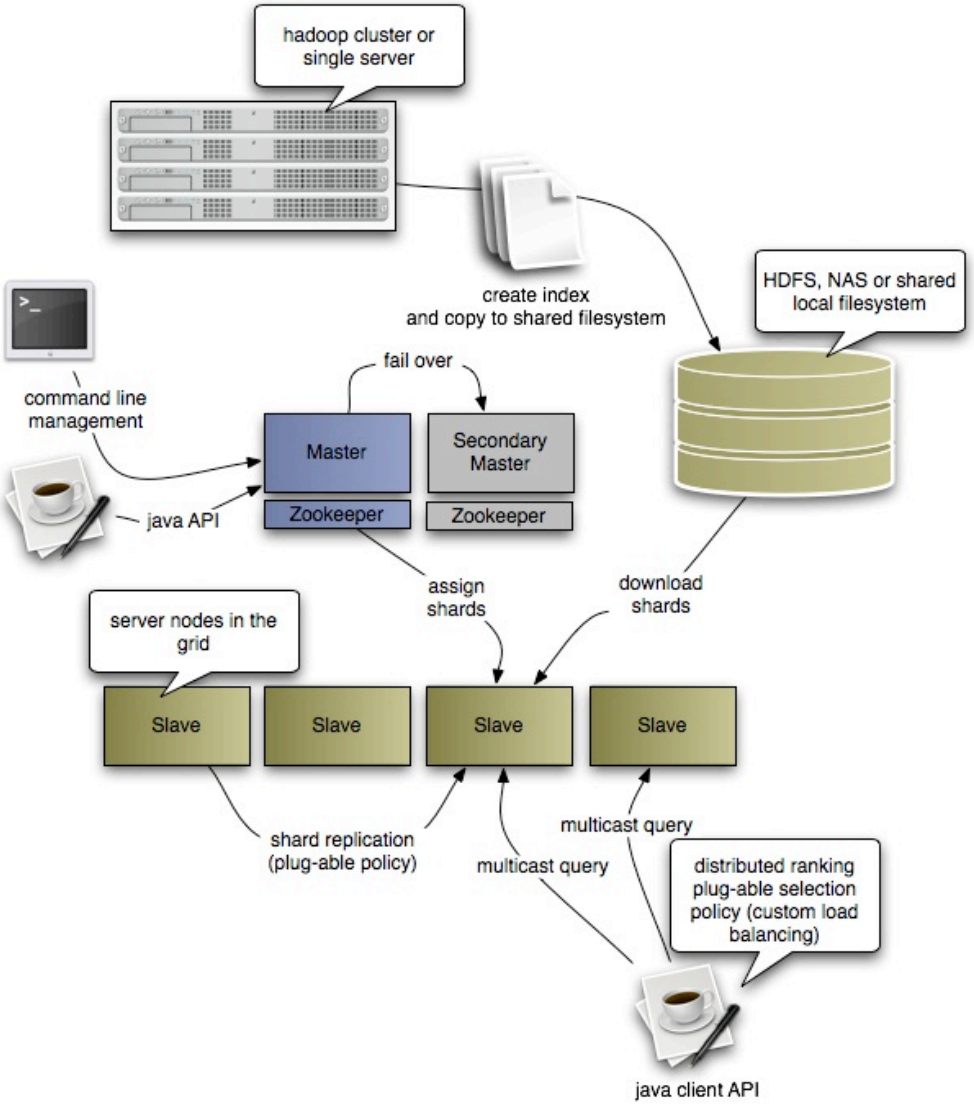
- Partitioning (for scalability)
- Replication (for redundancy)
- Caching (for speed)
- Routing (for load balancing)

The rest is just details!

Term vs. Document Partitioning



Katta Architecture
(Distributed Lucene)



http://katta.sourceforge.net/

Tweet Indexer

- **Access tweets on S3**
- **Phase 1: Build index and assign to clusters**
  - Retrieval using keyword
  - Keywords and cluster
- **Phase 2: after page rank lecture…do a text rank**
- **P**
- **P**

Enter a word and the index listings generated by the map reduce will be printed:

```
**********************
* Line Indexer Client *
**********************
Enter a word to query the index for.
 Or 'exit' to quit.
> ship
Index for: ship
119489 But we will ship him hence: and this vile deed
89045 Have lost my way for ever: I have a ship
89830 I will possess you of that ship and treasure.
34636 That their designment halts: a noble ship of Venice
34791 Third Gentleman    The ship is here put in,
37636 The riches of the ship is come on shore!
23092 Ere he take ship for France, and in Southampton.
74453 For there I'll ship them all for Ireland.
131177 Like to a ship that, having 'scaped a tempest,
108668 And ship from thence to Flanders.
132817 Whiles, in his moan, the ship splits on the rock,
61495 ANTIGONUS Thou art perfect then, our ship hath touch'd upon
65730 ship boring the moon with her main-mast, and anon
66517 Clown      I would you had been by the ship side, to have
114335 new ship to purge melancholy and air himself: for,
17361 Now am I like that proud insulting ship
```

Unigram Search Engine

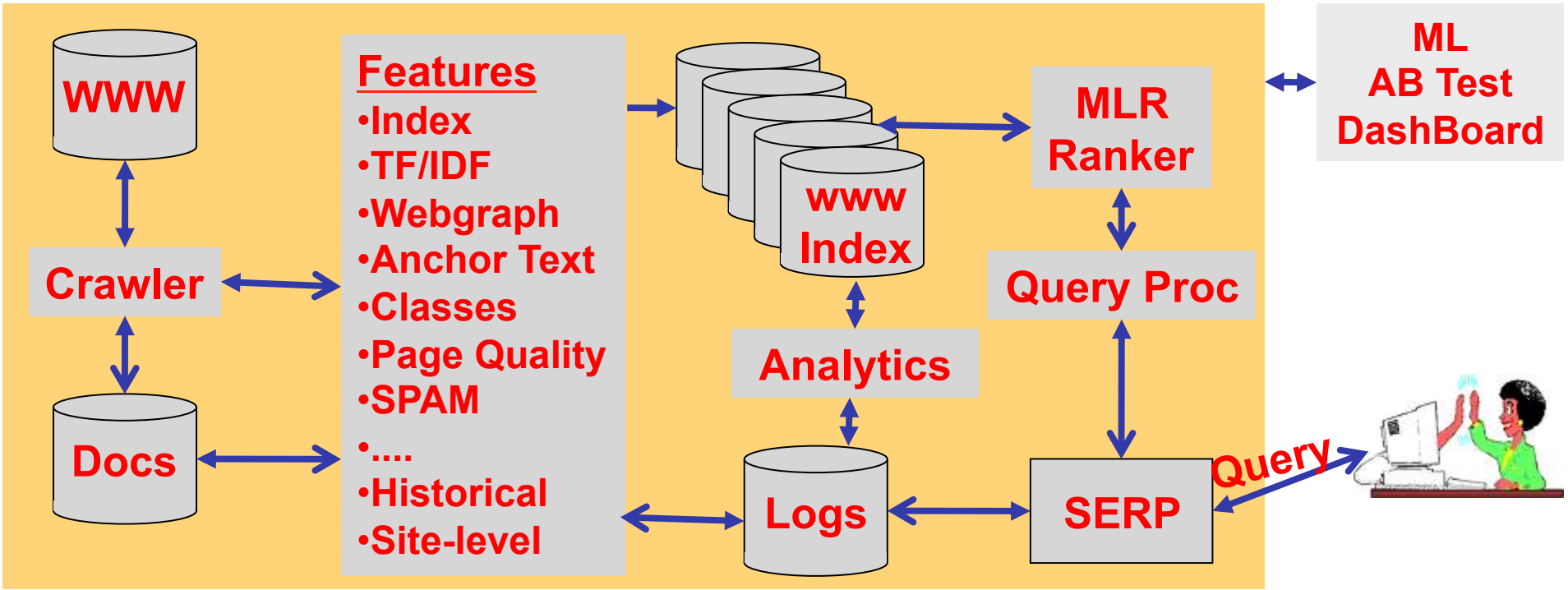
- https://courses.cs.washington.edu/courses/cse490h/07wi/hadoop-codelab.html

- **Unigram**
- **Extensions**
  - Bigram
  - Language models?

Detailed Example of Index Construction

- http://www.google.com/url?sa=t&source=web&ct=html&cd=1&ved=undefined&url=http%3A%2F%2F74.125.155.132%2Fsearch%3Fq%3Dcache%3AEQjZmpSWs6oJ%3Acode.google.com%2Fedu%2Fsubmissions%2Fmapreduce%2Fhadoopcodelab.doc%2Bshuffle%2Bhadoop%26cd%3D1%26hl%3Den%26ct%3Dclnk%26gl%3Dus&ei=YzpBS9i7ElekswPw6vjWAw&usg=AFQjCNEddn_6wzTU3z-7KzlHc_PppZaikg&sig2=0MZJHAmOrPASloG36Kxaow
- **Construct IR system and run queries**

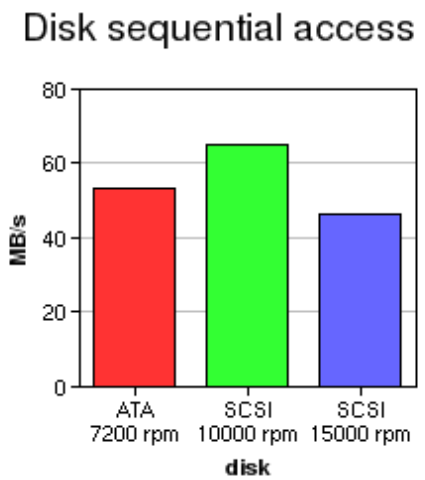
Search Engine Architecture



- Offline Processes
  - crawling, featurizing, Webgraph, Classification, updating ML models
- Online
  - query rewriting, ranking, reranking, merging, logging, analytics
- Realtime web indexing
- ML Framework

How to build Google in 30 mins?

- **Look at index construction and retrieval (10Terabytes)**
  - Retrieval is IO bound (limited)
  - Go from 28 hours to 2 milliseconds for a query round trip on
    - Store flat text files and access sequently; [28 hours to process a query]
    - Store an inverted index; [50 seconds]
    - Compress inverted index [7.5 (ignore disk latency, 7millisecs and CPU)]
    - Store an inverted index where posting entries are sorted by pagerank [2 millisecs]
- **Adapted from Information Retrieval Mode**
  Djoerd Hiemstra, University of Twente, RU
  - Available on ISM 251 website or
  - http://romip.ru/russir2009/slides/irm/lecture6.pdf
  - lecture 6.



Inverted index

- ...

term	offset	Documents
cold	2	1, 4
days	4	3, 6
hot	6	1, 4
like	8	4, 5
nine	10	3, 6
old	12	3, 6
pease	14	1, 2
porridge	16	1, 2
pot	18	2, 5
some	20	4, 5

dictionary

postings

- The inverted file entries are usually stored in order of increasing document number

  – [<*retrieval*; 7; [2, 23, 81, 98, 121, 126, 180]>

  (the term "retrieval" occurs in 7 documents with document identifiers 2, 23, 81, 98, etc.)

Query processing (3)

- Remember the Boolean model?
  - intersection, union and complement is done on posting lists
  - so, *information OR retrieval*

    [<*retrieval*; 7; [2, 23, 81, 98, 121, 126, 139]>

    [<*information*; 9; [1, 14, 23, 45, 46, 84, 98, 111, 120]>
  - union of posting lists gives:

    [1, 2, 14, 23, 45, 46, 81, 84, 98, 111, 120, 121, 126, 139]

Inverted file compression (1)

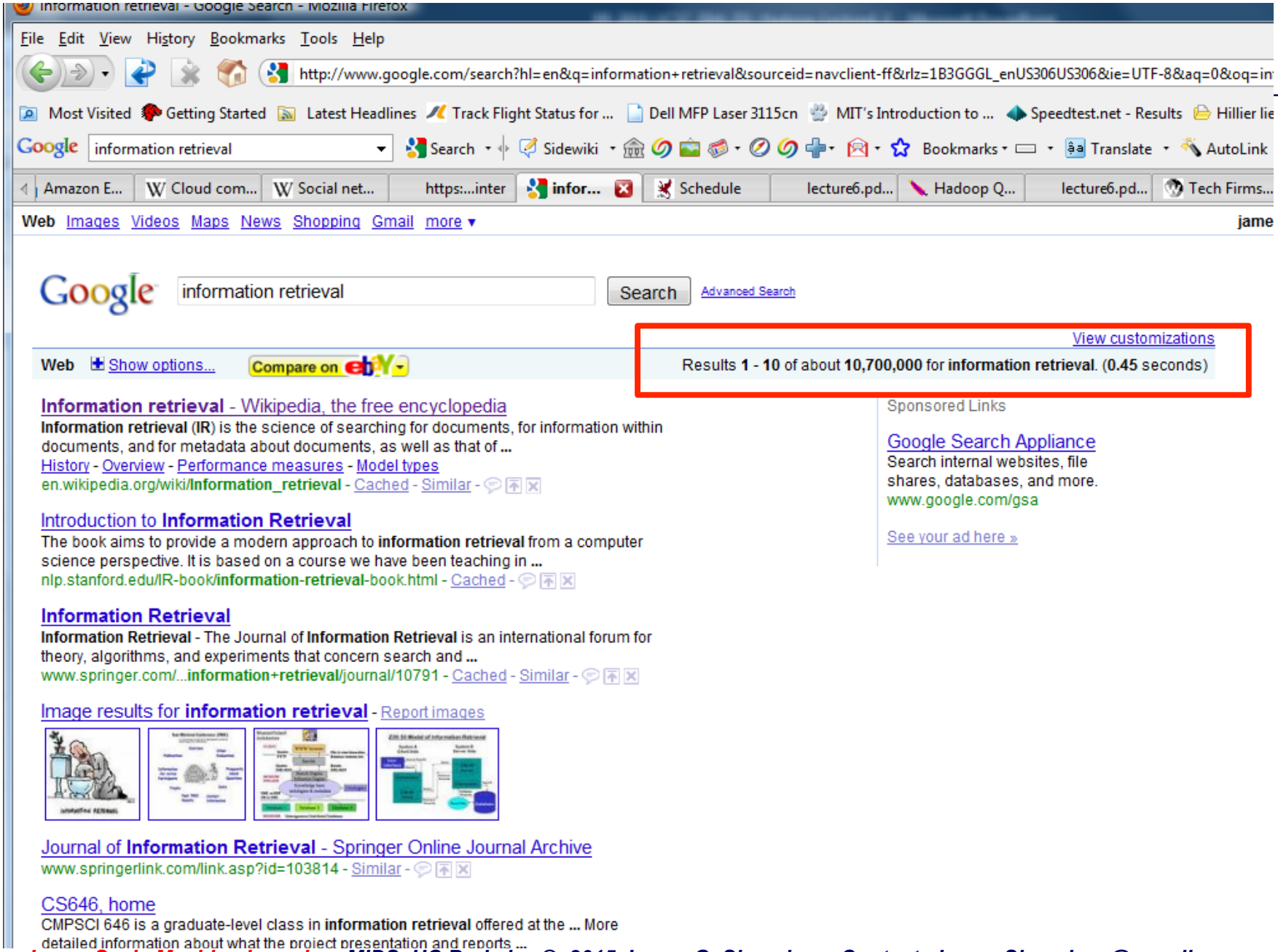
- Trick 1, store sequence of doc-ids:
  - [<*retrieval*; 7; [2, 23, 81, 98, 121, 126, 180]>

  as a sequence of gaps
  - [<*retrieval*; 7; [2, 21, 58, 17, 23, 5, 54]>

- No information is lost.
- Always process posting lists from the beginning so easily decoded into the original sequence

Query roundtrip from a day to 2 milliseconds

- **Google's index size**
  - 24 Million webpages in 1998 vs. 24 Billion in 2009 ($24 * 10^9$ webpages)
  - 180M queries per day or 2Million per second
  - Today it has 30 Trillion (10^{12}) unique webpages (230million sites) [WSJ, 1/16/2013]
- **For simplicity lets assume 10^{10} web pages (10Billion pages) with about 200 terms on average where each term is 5 characters**
  - 10^{10} x 200 x 5 ~= 10 TB to store raw text
  - 28 hours to process a query [load data to memory]
  - Store an inverted index; [50 seconds for query *information retrieval*]
  - Compress inverted index [7.5 seconds] using gaps and variable byte encoding
  - Store an inverted index where posting entries are sorted by pagerank [2 millisecs]
  - (ignore disk latency [7millisecs] and CPU time)


information retrieval - Google Search - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://www.google.com/search?hl=en&q=information+retrieval&sourceid=navclient-ff&rlz=1B3GGGL_enUS306US306&ie=UTF-8&aq=0&oq=in
Most Visited Getting Started Latest Headlines Track Flight Status for ... Dell MFP Laser 3115cn MIT's Introduction to ... Speedtest.net - Results Hillier lie
Google information retrieval Search Sidewiki Bookmarks Translate AutoLink
Amazon E... Cloud com... Social net... https:...inter infor... Schedule lecture6.pd... Hadoop Q... lecture6.pd... Tech Firms...
Web Images Videos Maps News Shopping Gmail more ▾ jame
Google information retrieval Search Advanced Search
View customizations
Web Show options... Compare on ebay
Results 1 - 10 of about 10,700,000 for information retrieval. (0.45 seconds)
Information retrieval - Wikipedia, the free encyclopedia
Information retrieval (IR) is the science of searching for documents, for information within documents, and for metadata about documents, as well as that of ...
History - Overview - Performance measures - Model types
en.wikipedia.org/wiki/Information_retrieval - Cached - Similar -
Introduction to Information Retrieval
The book aims to provide a modern approach to information retrieval from a computer science perspective. It is based on a course we have been teaching in ...
nlp.stanford.edu/IR-book/information-retrieval-book.html - Cached -
Information Retrieval
Information Retrieval - The Journal of Information Retrieval is an international forum for theory, algorithms, and experiments that concern search and ...
www.springer.com/...information+retrieval/journal/10791 - Cached - Similar -
Image results for information retrieval - Report images
Journal of Information Retrieval - Springer Online Journal Archive
www.springerlink.com/link.asp?id=103814 - Similar -
CS646, home
CMPSCI 646 is a graduate-level class in information retrieval offered at the ... More detailed information about what the project presentation and reports ...
Sponsored Links
Google Search Appliance
Search internal websites, file shares, databases, and more.
www.google.com/gsa
See your ad here »

MidTerm

- **Week 6**

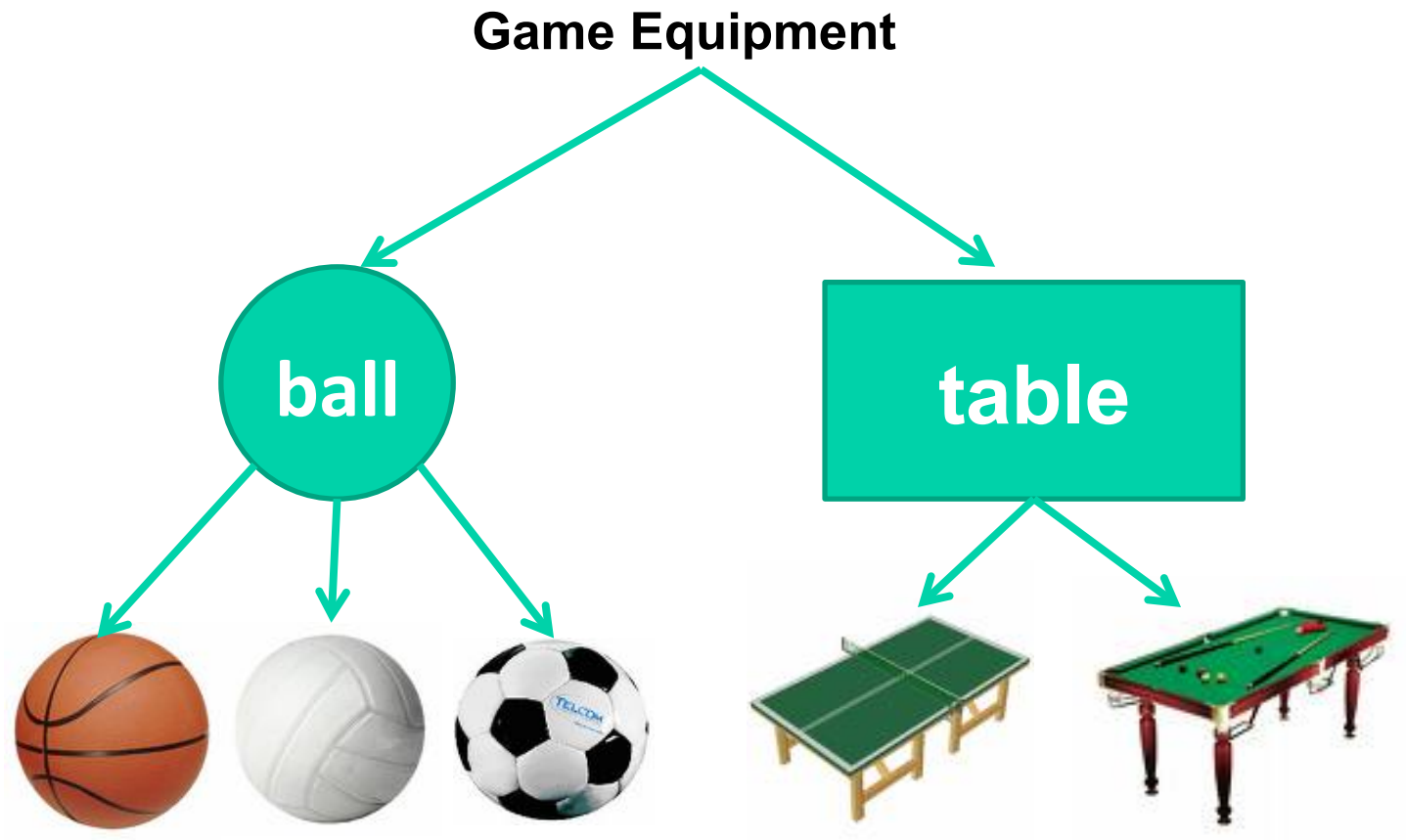
Live Session Outline

- **Housekeeping**
  - Please mute you microphones
  - Start RECORDING (bonus points for reminding me!)
- **Week 5**
  - Homework HW3, HW4, HW5
  - AWS (hope to have a access for week 6)
  - Async lecture recap plus Q&A [Jimi]
  - *Vector Space Model*
  - *Inverted index*
  - *Ontology; Similarity*
  - Synonym detection in Map-Reduce
  - GMM
- **Wrapup**
  - Finish RECORDING (bonus points for reminding me!)
  - Click End Meeting

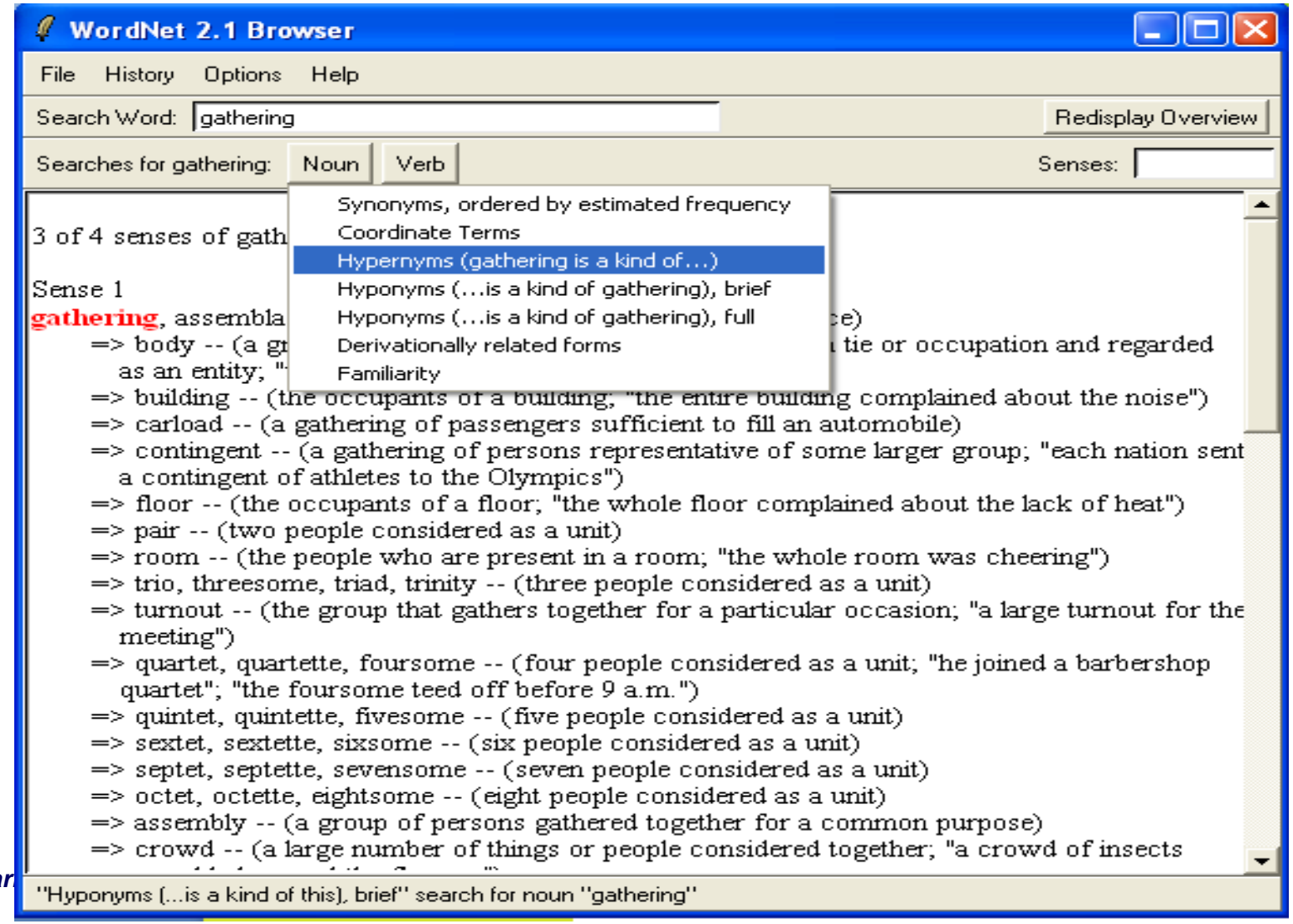
Introduction

- **Ontology is a data model that represents a set of concepts within a domain and the set of pair-wise relationships between those concepts.**

Example: A simple Ontology



Example: WordNet



Example: ODP


dmoz open directory project
Top: Arts: Movies: Characters (4)
Austin Powers Series@ (2)
Cartoons@ (4)
Harry Potter Series@ (8)
Lord of the Rings@ (77)
Star Trek@ (38)
Star Wars@ (166)
Thin Man Series@ (16)
X-Men Series@ (35)
Batman@ (84)
Boston Blackie@ (6)
Bulldog Drummond@ (4)
Charlie Chan@ (4)
Count of Monte Cristo, The@ (18)
Crime Doctor@ (2)
Dick Tracy@ (4)
Dirty Harry@ (18)
Dr. Dolittle@ (17)
Dr. Mabuse@ (28)
Dracula@ (36)
Ernest P. Worrell@ (10)
John Rambo@ (24)
King Kong@ (7)
Lassie@ (5)
Lew Harper@ (6)
Lone Ranger@ (3)
Lone Wolf, The@ (3)
Mad Max@ (27)
Mary Poppins@ (7)
Michael Myers@ (38)
Michael Shayne@ (5)
Mike Hammer@ (17)
Mr. Moto@ (4)

Introduction

- **Ontology Learning is the task to construct a well-defined ontology given**
  - a text corpus or
  - a set of concept terms

Introduction

- Ontology offers a nice way to summarize the important topics in a domain/collection
- Ontology facilitates knowledge sharing and reuse
- Ontology offers relational associations for reasoning and inference

NLP treat words as discrete atomic symbols

- **Natural language processing systems traditionally treat words as discrete atomic symbols, and therefore 'cat' may be represented as Id537 and 'dog' as Id143.**
  - These encodings are arbitrary, and provide no useful information to the system regarding the relationships that may exist between the individual symbols.
  - This means that the model can leverage very little of what it has learned about 'cats' when it is processing data about 'dogs' (such that they are both animals, four-legged, pets, etc.).
- **Representing words as unique, discrete ids furthermore leads to data sparsity, and usually means that we may need more data in order to successfully train statistical models.**
- **Using vector representations can overcome some of these obstacles.**

Live Session Outline

- **Housekeeping**
  - Please mute you microphones
  - Start RECORDING (bonus points for reminding me!)
- **Week 5**
  - Homework HW3, HW4, HW5
  - AWS (hope to have a access for week 6)
  - Async lecture recap plus Q&A [Jimi]
  - *Vector Space Model*
  - *Inverted index*
  - *Similarity*
  - Synonym detection in Map-Reduce
  - GMM
- **Wrapup**
  - Finish RECORDING (bonus points for reminding me!)
  - Click End Meeting

- **synonyms discovery algorithms**

Vector space models (VSMs)

- Vector space models (VSMs) represent (embed) words in a continuous vector space where semantically similar words are mapped to nearby points ('are embedded nearby each other').
- VSMs have a long, rich history in NLP, but all methods depend in some way or another on the Distributional Hypothesis, which states that words that appear in the same contexts share semantic meaning.
- The different approaches that leverage this principle can be divided into two categories: count-based methods (e.g. Latent Semantic Analysis), and predictive methods (e.g. neural probabilistic language models).

VSM

- **Distributional Hypothesis:**
  - states that words that appear in the same contexts share semantic meaning.
- **The different approaches that leverage this principle can be divided into two categories:**
  - count-based methods (e.g. Latent Semantic Analysis), and
  - predictive methods (e.g. neural probabilistic language models).

Count versus Predictive methods

- **This distinction is elaborated in much more detail by Baroni et al., but in a nutshell:**
- **Count-based methods compute the statistics of how often some word co-occurs with its neighbor words in a large text corpus, and then map these count-statistics down to a small, dense vector for each word.**
  - Clustering, SVD
- **Predictive models directly try to predict a word from its neighbors in terms of learned small, dense embedding vectors (considered parameters of the model).**
  - Word2Vec and its variations

Predictive method: Word2Vec

- **Word2vec is a particularly computationally-efficient predictive model for learning word embeddings from raw text.**
  - It comes in two flavors, the Continuous Bag-of-Words model (CBOW) and the Skip-Gram model. Algorithmically, these models are similar, except that CBOW predicts target words (e.g. 'mat') from source context words ('the cat sits on the'), while the skip-gram does the inverse and predicts source context-words from the target words.
  - This inversion might seem like an arbitrary choice, but statistically it has the effect that CBOW smoothes over a lot of the distributional information (by treating an entire context as one observation).
  - For the most part, this turns out to be a useful thing for smaller datasets. However, skip-gram treats each context-target pair as a new observation, and this tends to do better when we have larger datasets.

- **Embeddings/representations**
- **Clustering**
- **SVD**
- **Word2Vec (deep learning)**

Synonym detection Workflow

- **Problem understanding and EDA**
- **Stump out code**
- **Generate a unit/systems test**
  - For similarity function (e.g., Jaccard)
  - Unit tests for each mapper/reducer
- **Write code locally**
  - unit-test
    - each mapper independently locally
    - and each reducer independently locally
  - Systems test locally
- **Systems test in the cloud (with small dataset)**
- **Run system on entire dataset**
- **Do evaluation using wordnet**
- **Report findings**

Unit test:
Results computed by hand or
by another system

A: 1010011 AB= 0.6 0.4 (1-AB)
B: 1001011 BC=0.5 0.5
C: 1001000 AC=0.2 0.8

Bigram-based coocurrence: "Stripes" Pattern

- **Idea: group together pairs into an associative array**

(a, b) → 1
(a, c) → 2
(a, d) → 5
(a, e) → 3
(a, f) → 2

ABCDA → AB, AC, AD, AA, BA, BC, BD, BA, CA, CB, CD, CA, DA, DB, DB, DC, DA, AA, AB, A

a → {b: 1, c: 2, d: 5, e: 3, f: 2 }

b → {a: 1, c: 2, d: 5, e: 3, f: 2 }

2 possibilities
- square matrix
- Triangular

Extra credit

- **Each mapper takes a sentence:**
  - Generate all co-occurring term pairs
  - For each term, emit a → { b: $count_b$, c: $count_c$, d: $count_d$ … }
- **Reducers perform element-wise sum of associative arrays**

2+ mappers
1+ reducer

a → { b: 1, d: 5, e: 3 }
a → { b: 1, c: 2, d: 2, f: 2 }
a → { b: 2, c: 2, d: 7, e: 3, f: 2 }

Key: cleverly-constructed data structure brings together partial results

Pairs versus Stripes

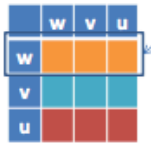
Pairs Design Pattern


```
map(docID a, doc d)
 for all term w in doc d do
  for all term u NEAR w do
   Emit(pair (w, u), count 1)

reduce(pair p, counts [c1, c2,…])
 sum = 0
 for all count c in counts do
  sum += c
 Emit(pair p, count sum)
```

- Can use combiner or in-mapper combining
- Good: easy to implement and understand
- Bad: huge intermediate-key space (shuffling/sorting cost!)
  - Quadratic in number of distinct terms

204

Stripes Design Pattern




```
map(docID a, doc d)
 for all term w in doc d do
  H = new hashMap
  for all term u NEAR w do  H{u} ++
  Emit(term w, stripe H)

reduce(term w, stripes [H1, H2,…])
 Hout = new hashMap
 for all stripe H in stripes do  Hout = ElementWiseSum(Hout, H)
 Emit(term w, stripe Hout)
```

- Can use combiner or in-mapper combining
- Good: much smaller intermediate-key space
  - Linear in number of distinct terms
- Bad: more difficult to implement, Map needs to hold entire stripe in memory

205

https://www.dropbox.com/s/p802029b3kj4oz1/PairsAnd-StripesAnd-Order-Inversion-Good%20Slides.pdf?dl=0

Build up stripes for each word: Term cooccurence matrix

Each word is associated with an object or data structure (could view as a document)

C12 denotes (w1 cooccurs with w2)

o1	**c11**	**c12**	**c13**	**c14**
o2	c21	c22	c23	c24
o3	c31	c32	c33	c34
o4	c41	c42	c43	c44

Assume a trigram world

W1 w4 **w2**

W1 w3 **w2**

Term co-occurrence matrix

Dense versus sparse representation

a → { b: 1, d: 5, e: 3 } SPARSE

a → {0,1,0,1,1,0,0,0,0….} DENSE not useful

Use Term coocurence matrix to calculate the term by term similarity matrix

Each word is associated with an object or data structure (could view as a document)

C12 denotes (w1 cooccurs with w2)

Assume a trigram world

W1 w4 **w2**

W1 w3 **w2**

o1	**c11**	**c12**	**c13**	**c14**
o2	c21	c22	c23	c24
o3	c31	c32	c33	c34
o4	c41	c42	c43	c44

Term by term similarity matrix

	o1	**o2**	**o3**	**o4**
o1	**Sim(w1, w1)**	**Sim(w1, w2)**	**Sim(w1, w3)**	**Sim(w1, w4)**
o2	s21	s22	s23	s24
o3	s31	s32	s33	s34
o4	s41	s42	s43	s44

Similarity(w11, w12)

Jaccard Similarity: set-based

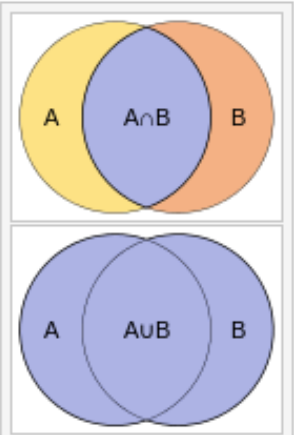
The **Jaccard index**, also known as the **Jaccard similarity coefficient** (originally coined *coefficient de communauté* by Paul Jaccard), is a statistic used for comparing the similarity and diversity of sample sets. The Jaccard coefficient measures similarity between finite sample sets, and is defined as the size of the intersection divided by the size of the union of the sample sets:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}.$$

(If A and B are both empty, we define $J(A,B) = 1$.)

$$0 \leq J(A,B) \leq 1.$$

The MinHash min-wise independent permutations locality sensitive hashing scheme may be used to efficiently compute an accurate estimate of the Jaccard similarity coefficient of pairs of sets, where each set is represented by a constant-sized signature derived from the minimum values of a hash function.



Intersection and union of two sets A and B

The **Jaccard distance**, which measures *dis*similarity between sample sets, is complementary to the Jaccard coefficient and is obtained by subtracting the Jaccard coefficient from 1, or, equivalently, by dividing the difference of the sizes of the union and the intersection of two sets by the size of the union:

$$d_J(A,B) = 1 - J(A,B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|}.$$

Jaccard Similarity

The **Jaccard index**, also known as the **Jaccard similarity coefficient** (originally c
by Paul Jaccard), is a statistic used for comparing the similarity and diversity of sar
measures similarity between finite sample sets, and is defined as the size of the int
union of the sample sets:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}.$$

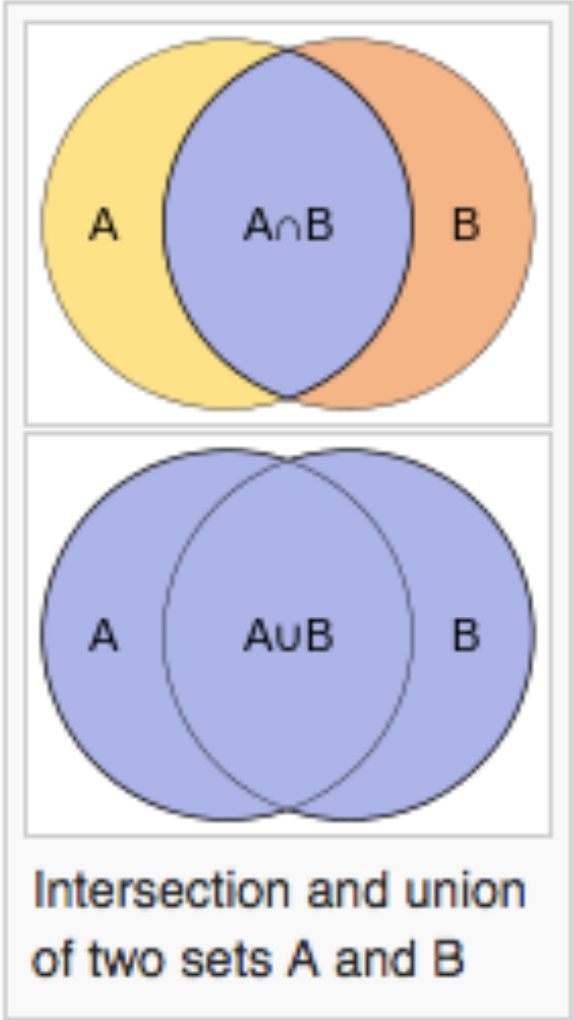
(If A and B are both empty, we define $J(A,B) = 1$.)

$$0 \leq J(A,B) \leq 1.$$

The MinHash min-wise independent permutations locality sensitive hashing schem
compute an accurate estimate of the Jaccard similarity coefficient of pairs of sets, v
constant-sized signature derived from the minimum values of a hash function.

The **Jaccard distance**, which measures *dis*similarity between sample sets, is com
subtracting the Jaccard coefficient from 1, or, equivalently, by dividing the differenc
the size of the union:

$$d_J(A,B) = 1 - J(A,B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|}.$$



Intersection and union of two sets A and B

d union
nd B

by
sets by

In Map-Reduce

|{a,c} AND {a, b}| = |{a}|

---------- --------

|{a,c} OR{a, b}| = |{a, b,c}|

$$J(A,B) = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}.$$

C1={a, c}

C2={a, b}

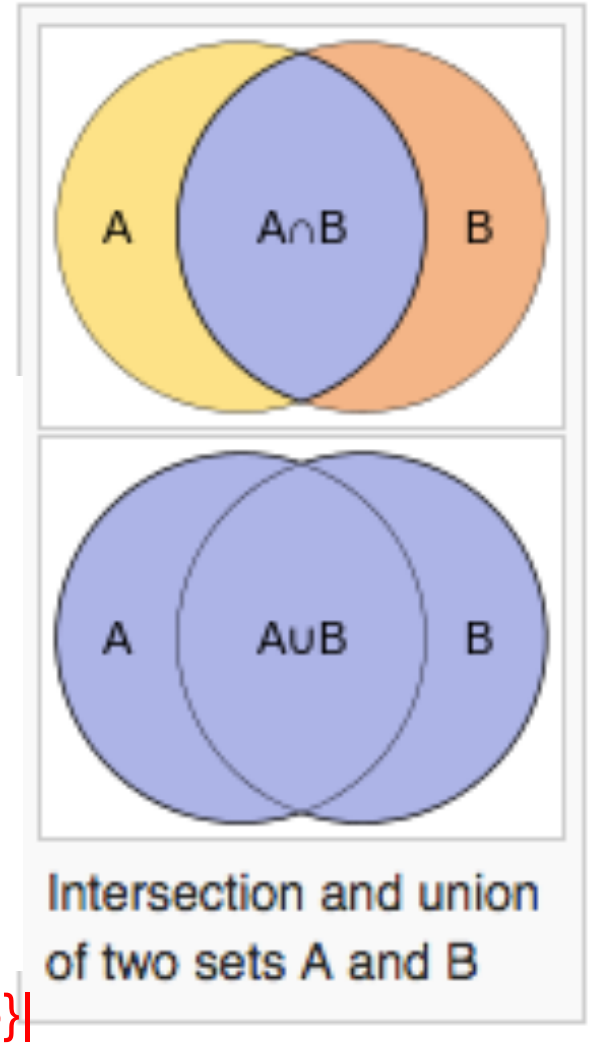
Type	C1	C2
a	1	1
b	0	1
c	1	0
d	0	0

Sim(C1, C2) =
= a/(a+b+c)

E.g.

	C1	C2
W1	0	1
W2	1	0
W3	1	1
W4	0	0
W5	1	1
w6	0	1

= Similarity 2/5 = 40%



Intersection and union of two sets A and B

Denominator

|{W2, W3 W5} union {w1, w3, w5, w6}|

|{1, 2,3,5, 6}| = 5

Example: Jaccard's

- **Only 1-1 and 1-0 count, 0-0 do not count**

A: 1010011 AB= 0.6 3/5 0.4 (1-AB)

B: 1001011 BC=0.5 0.5

C: 1001000 AC=0.2 0.8

$$\frac{|A \cap B|}{|A| + |B| - |A \cap B|}.$$

1010011
1001011

Jaccard(AB)= 0.6

A∧B =3
A∨B = 5

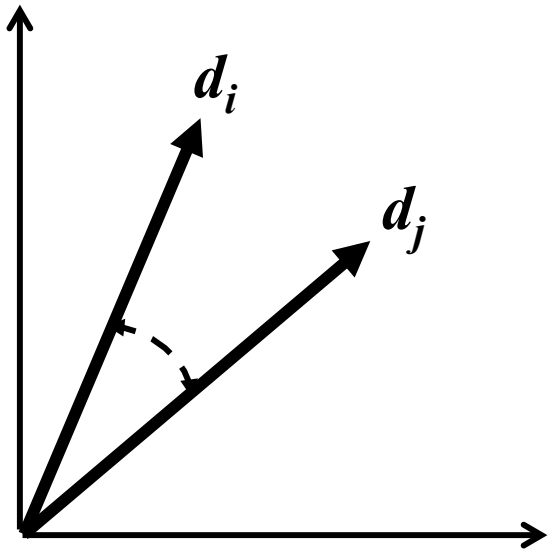
Jaccard: weaknesses

- Does not consider number of occurrences!
- No order information

Cosine: Similarity of Documents

$$sim(d_i, d_j) = \sum_{t \in V} w_{t,d_i} w_{t,d_j}$$

- **Simple inner product**
- **Cosine similarity**
- **Term weights**
  - Standard problem in IR
  - tf-idf, BM25, etc.



Cosine similarity

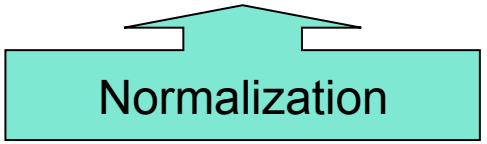
$$sim(d_j, d_k) = \frac{\vec{d}_j \cdot \vec{d}_k}{\left|\vec{d}_j\right|\left|\vec{d}_k\right|} = \frac{\sum_{i=1}^{n} w_{i,j} w_{i,k}}{\sqrt{\sum_{i=1}^{n} w_{i,j}^2}\sqrt{\sum_{i=1}^{n} w_{i,k}^2}}$$

Dot product

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

Dot product unit length documents/vectors

- **Cosine of angle between two vectors**
- **The denominator involves the lengths of the vectors.**



Normalized vectors

- **For normalized vectors, the cosine is simply the dot product:**

$$\cos(\vec{d}_j, \vec{d}_k) = \vec{d}_j \cdot \vec{d}_k$$

Example: Cosine

- Docs: Austen's *Sense and Sensibility*, *Pride and Prejudice*; Bronte's *Wuthering Heights*

	SaS	PaP	WH
affection	115	58	20
jealous	10	7	11
gossip	2	0	6

Raw Counts

	SaS	PaP	WH
affection	0.996	0.993	0.847
jealous	0.087	0.120	0.466
gossip	0.017	0.000	0.254

Normalized Counts

- cos(SAS, PAP) = .996 x .993 + .087 x .120 + .017 x 0.0 = 0.999
- cos(SAS, WH) = .996 x .847 + .087 x .466 + .017 x .254 = 0.929

Build up stripes for each word (psuedo document): AKA Term cooccurence matrix

Each word is associated with an object or data structure, a stripe of words (could view as a document)

C12 denotes (w1 cooccurs with w2)

Assume a trigram world

W1 w4 **w2**

W1 w3 **w2**

O1=hadoop

o2

o3

o4

c11	c12	c13	c14
c21	c22	c23	c24
c31	c32	c33	c34
c41	c42	c43	c44

The Xerox

Term co-occurrence matrix

	w1	w2	w3	w4
W1→o1	c11	c12	c13	c14

Stripe for w1 (can be viewed as pseudo-document)

	a	b	c	d
Doc1	1	0	1	0
Doc2	1	0	0	1

$$\frac{|A \cap B|}{|A| + |B| - |A \cap B|}.$$

Jaccard Similarity(Doc1, Doc2) = 1/3

Use Term co-ocurence matrix to calculate the term by term similarity matrix from word stripes (pseudo documents)

Each word is associated with an object or data structure (could view as a document)

C12 denotes (w1 cooccurs with w2)

Assume a trigram world

W1 w4 **w2**

W1 w3 **w2**

o1	**c11**	**c12**	**c13**	**c14**
o2	c21	c22	c23	c24
o3	c31	c32	c33	c34
o4	c41	c42	c43	c44

Pairwise: Term by term similarity matrix

	o1	**o2**	**o3**	**o4**
o1	**Sim(w1, w1)**	**Sim(w1, w2)**	**Sim(w1, w3)**	**Sim(w1, w4)**
o2	s21	s22	s23	s24
o3	s31	s32	s33	s34
o4	s41	s42	s43	s44

Similarity(w11, w12)

Use Term co-ocurence matrix to calculate the term by term similarity matrix from word stripes (pseudo documents)

Each word is associated with an object or data structure (could view as a document)

C12 denotes (w1 cooccurs with w2)

Assume a trigram world

o1	**c11**	**c12**	**c13**	**c14**
o2	c21	c22	c23	c24
o3	c31	c32	c33	
o4	c41			

Challenge
Pairwise Object Similarity in MapReduce

Similarity(w11, w12)

	o3	o4
	Sim(w11, w11)	**Sim(w11, w12)**	**Sim(w11, w13)**	**Sim(w11, w14)**
o2	s21	s22	s23	s24
o3	s31	s32	s33	s34
o4	s41	s42	s43	s44

Term by term similarity matrix

- **Memory requirements for similairty matrix is N^2**
- **Complexity of this similarity matrix calculation?**

- **Complexity of this similarity matrix calculation?**
- **$O(N^2)$**
- **Will not scale in Map**

Challenge
Pairwise Object Similarity in MapReduce

	o1	o2	o3	o4
o1	**Sim(w11, w11)**	**Sim(w11, w12)**	**Sim(w11, w13)**	**Sim(w11, w14)**
o2	s21	s22	s23	s24
o3	s31	s32	s33	s34
o4	s41	s42	s43	s44

Similarity(w11, w12)

Synonym detection Workflow

- **Problem understanding and EDA**
- **Stump out code**
- **Generate a unit/systems test**
  - For similarity function
  - Unit tests for each mapper/reducer
- **Write code locally**
  - unit-test
    - each mapper independently locally
    - and each reducer independently locally
  - Systems test locally
- **Systems test in the cloud (with small dataset)**
- **Run system on entire dataset**
- **Do evaluation using wordnet**
- **Report findings**

Results computed by hand or by another system

A: 1010011 AB= 0.6 0.4 (1-AB)
B: 1001011 BC=0.5 0.5
C: 1001000 AC=0.2 0.8

Pairwise object similarity matrix computation: How to do this?

Version 1: Crude approach

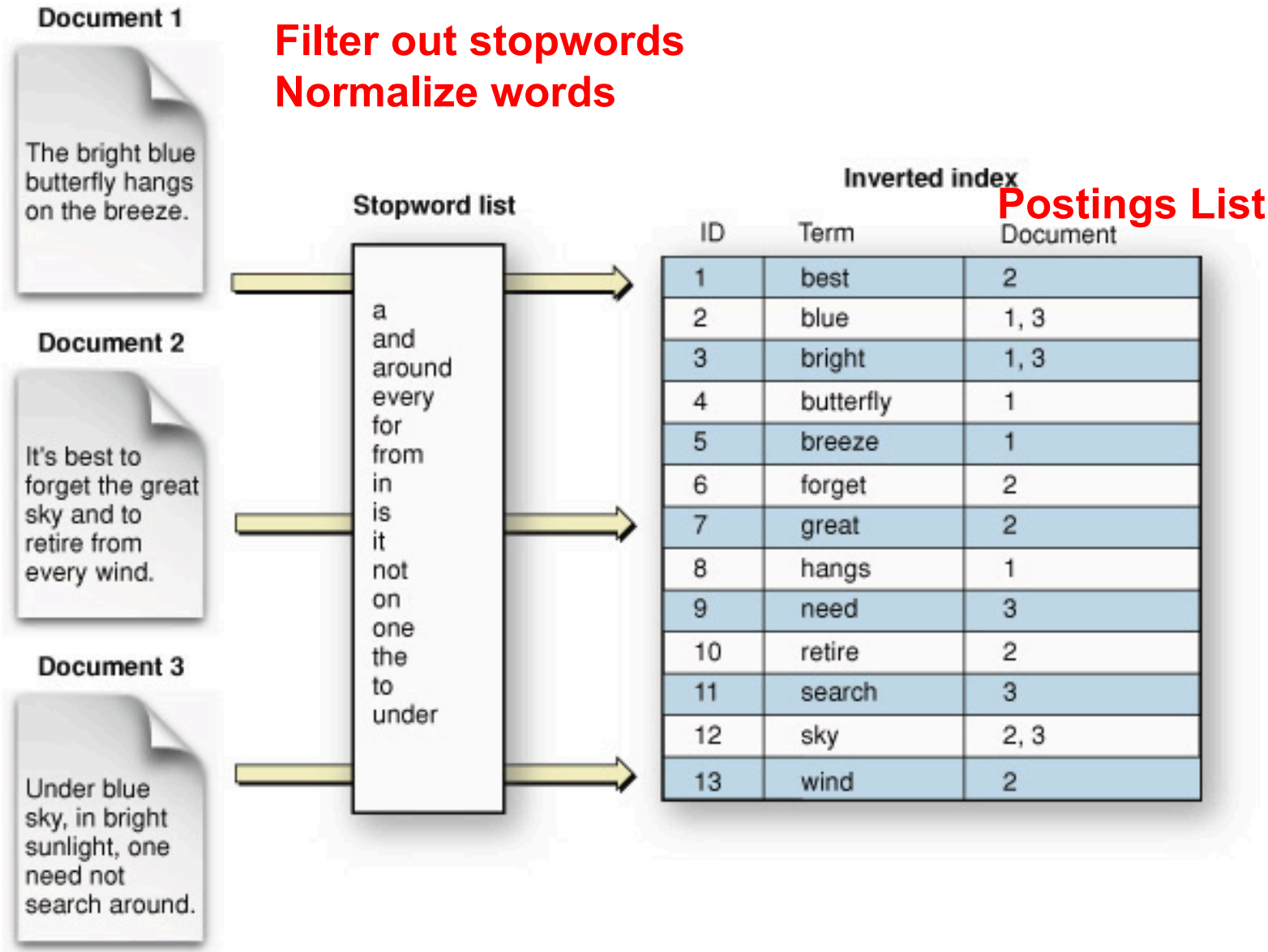
- **Complexity: ½ × N(N-1)**
- **Exploit assumptions and structure of problem**
  - 10,000 objects described using 10,000 words
  - Sparse or dense representation? (does not matter)
  - Provide the number of features (10,000)

- **Atomic unit of parallelism?**
- **Hierarchy**
  1. Each cell in the output similarity matrix can be calculated independent of each other.
  2. Then work back from there (each pairwise similarity cell) to intermediate results
  3. Then back to the raw input data

Inverted Index:

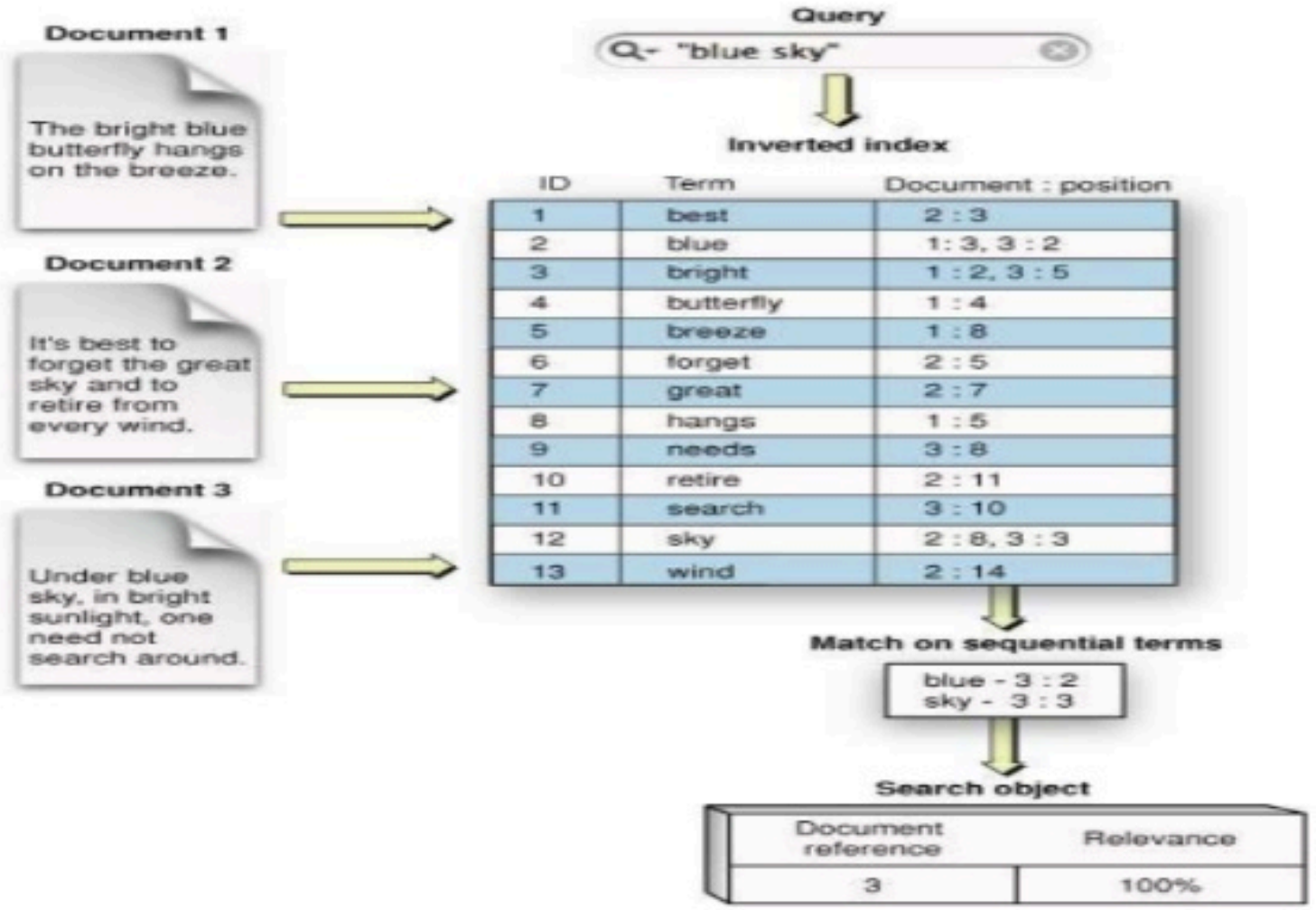
Each Posting consists of (DocID, PayLoad)

Filter out stopwords
Normalize words



ID	Term	Document
1	best	2
2	blue	1, 3
3	bright	1, 3
4	butterfly	1
5	breeze	1
6	forget	2
7	great	2
8	hangs	1
9	need	3
10	retire	2
11	search	3
12	sky	2, 3
13	wind	2

Lucene Internals - Inverted Index

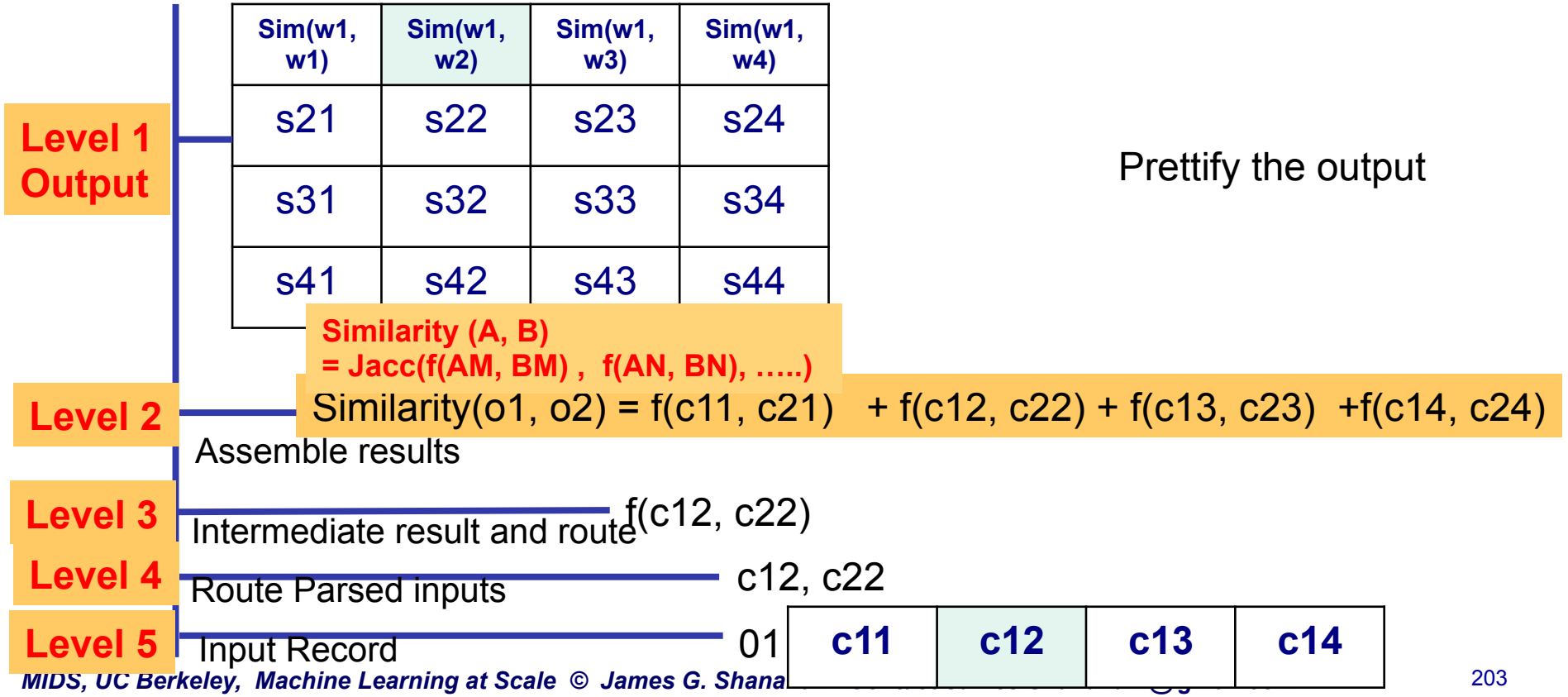


9

Credit: https://developer.apple.com/library/mac/documentation/userexperience/conceptual/SearchKitConcepts/searchKit_basics/searchKit_basics.html

Hierarchy of decomposition: Work back from output to inputs

- **Atomic unit of parallelism?**
- **Hierarchy (work back from output to input)**
  1. Each cell in the output similarity matrix is calculated independent of each other.
  2. Then work back from there (each pairwise similarity cell) to intermediate results
  3. Then back to the raw input data



Pairwise similarity Jaccard calculation

	w1	w2	w3	w4
o1	c11	c12	c13	c14
o2	c21	c22	c23	c24
o3	c31	c32	c33	c34
o4	c41	c42	c43	c44

$$\frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

Collect three quantities
- $|A \wedge B|$, $|A|$, $|B|$

Yields first row in item to item similarity matrix

	o1	o2	o3	o4
o1	Sim(w1, w1)	Sim(w1, w2)	Sim(w1, w3)	Sim(w1, w4)
o2	s21	s22	s23	s24
o3	s31	s32	s33	s34
o4	s41	s42	s43	s44

Each similarity calculation is decomposed in to a sum of $\mathbf{f(c_{ij}; c_{*j})}$; these components share a lot the same raw input data.

Similarity (A, B)
= Jacc(f(AM, BM) , f(AN, BN), …..)

Similarity(o1, o2) = f(c11, c21) + f(c12, c22) + f(c13, c23) +f(c14, c2

- **Output back to the raw input….decompose our output into intermediate values**

- **…**

Decompose the similarity into sum of intermediate quantities: f(…)s

Co-occurrence matrix

	w1	w2	w3	w4
o1	1	0	1	0
02	1	1	0	0

$$\frac{|A \cap B|}{|A| + |B| - |A \cap B|}.$$

Jaccard(o1, 02)= 1/3)

Collect three quantities
- $|A \wedge B|$, $|A|$, $|B|$
- These can

Similarity (A, B)
= Jacc(f(AM, BM) , f(AN, BN), …..)

Similarity(o1, o2) = f(c11, c21) + f(c12, c22) + f(c13, c23) +f(c14, c24)

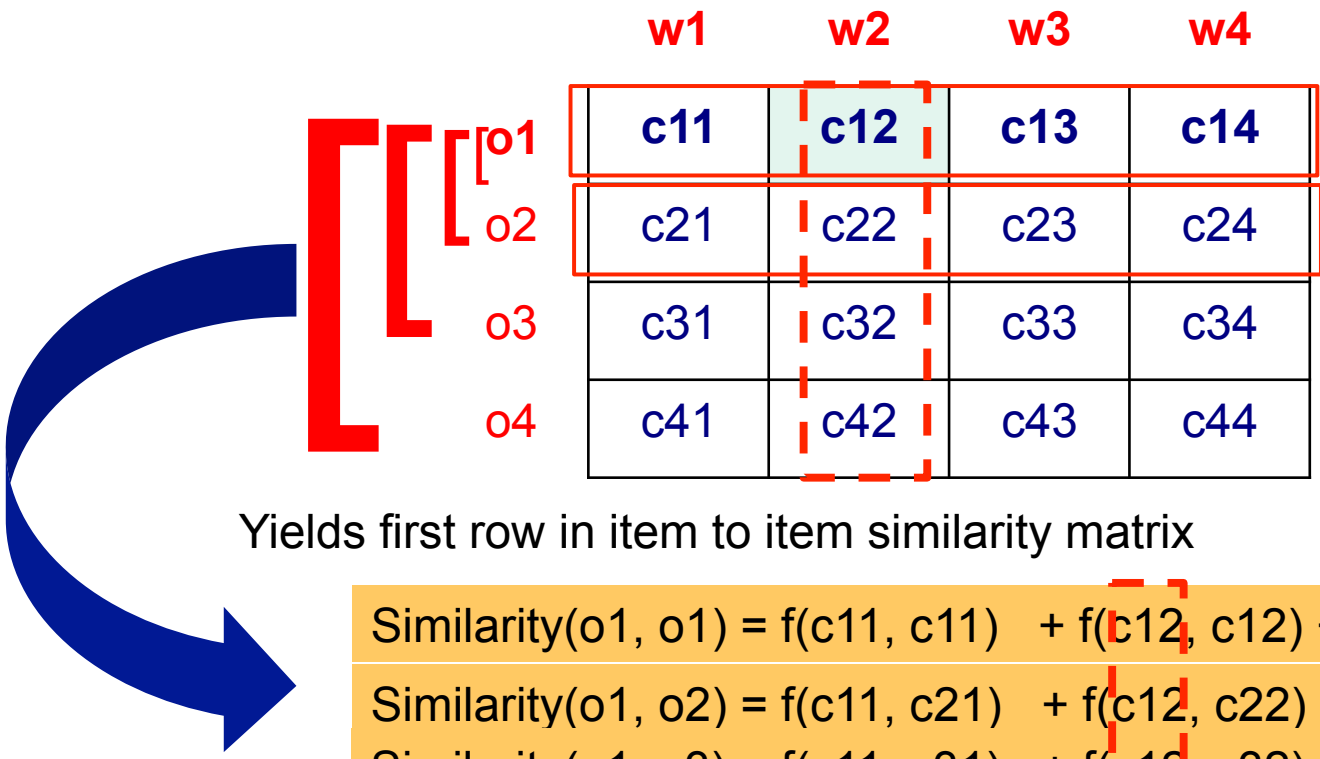
f(c11, c21) = Intersection: $1|A \wedge B|=1$; union: $|A|=1$, $|B|=1$
f(c12, c22) = Intersection: $1|A \wedge B|=0$; union: $|A|=0$, $|B|=1$

F(…) Needs c11, c21

f(c11, c21)
Intersection = 1 (or 0) if both (c11, c21) are 1
union 1 (or 0) if at least (c11, c21) has a 1

3-stage hierarchy from output to the input records
- **Similarity for a cell in the pairwise matrix**
- **Intermediate f()s**
- **Inputs for f()s**

Similarity calculations share a lot of the same data



$$\frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

Collect three quantities

- $|A \wedge B|$, $|A|$, $|B|$

Yields first row in item to item similarity matrix

Similarity(o1, o1) = f(c11, c11) + f(c12, c12) + f(c13, c13) +f(c14, c14)

Similarity(o1, o2) = f(c11, c21) + f(c12, c22) + f(c13, c23) +f(c14, c24)

Similarity(o1, o3) = f(c11, c31) + f(c12, c32) + f(c13, c33) +f(c14, c34)

Similarity(o1, o4) = f(c11, c41) + f(c12, c42) + f(c13, c43) +f(c14, c44)

Each similarity calculation is decomposed in to a sum of $\mathbf{f(c_{ij}; c_{*j})}$; these components share a lot of the same raw input data.

Decompose the similarity into sum of intermediate quantities: f(…)s

Co-occurrence matrix

o1	**1**	**0**	**1**	**0**
02	1	1	0	0

$$\frac{|A \cap B|}{|A| + |B| - |A \cap B|}.$$

Jaccard(o1, 02)= 1/3)

Collect three quantities
- |A∧B|, |A|, |B|
- These can

Similarity (A, B)
= Jacc(f(AM, BM) , f(AN, BN), …..)

Similarity(o1, o2) = f(c11, c21) + f(c12, c22) + f(c13, c23) +f(c14, c24)

f(c11, c21) = Intersection: 1|A∧B|=1; union: |A|=1, |B|=1
f(c12, c22) = Intersection: 1|A∧B|=0; union: |A|=0, |B|=1

F(…) Needs c11, c21

f(c11, c21)
Intersection = 1 (or 0) if both (c11, c21) are 1
union 1 (or 0) if at least (c11, c21) has a 1

3-stage hierarchy from output to the input records
- **Similarity for a cell in the pairwise matrix**
- **Intermediate f()s**
- **Inputs for f()s**

Each raw input value needs to be Dense emitted; so that it is made available for down stream intermediate calculations

Co-occurrence matrix

Divide and conquer the problem by columns but not by rows

	w1	w2	w3	w4
o1	c11	c12	c13	c14
o2	c21	c22	c23	c24
o3	c31	c32		
o4	c41	c42		

Dense emit each input value and route properly t the downstream fro the intermediae

Similarity(o1, o1) = f(c11, c11) + f(c12, c12) + f(c13, c13) +f(c14, c14
Similarity(o1, o2) = f(c11, c21) + f(c12, c22) + f(c13, c23) +f(c14, c24
Similarity(o1, o3) = f(c11, c31) + f(c12, c32) + f(c13, c33) +f(c14, c34
Similarity(o1, o4) = f(c11, c41) + f(c12, c42) + f(c13, c43) +f(c14, c44

Similarity(o1, o2) = f(c11, c21) + *f(c12, c22)* + f(c13, c23) +f(c14, c24)

Key (intermediate)	Final address part	Payload
1,2	1	C12=1 (TWICE)
2,2	***1***	***c12=1***
3,2	1	C12=1
4,3	1	C12=1
1,2	2	C22=1
2,2	***2***	***C22=1 (TWICE)***
3,2	2	C22=1
4,2	2	C22=1

C12 will be used in the calculation of f(c12, c*2)

Reducer:1,2 value f(c12, c

C22 will be used in the calculation of f(c22, c*2)

MapReduce Jobs for Pair Similarity

Similarity(01, 02) = f(c11, c21) + *f(c12, c22)* + f(c13, c23) +f(c14, c24)

- **Map-reduce Job 1: route data for each *f(c12, c22)***
  - Mapper route *cij* Route the basic components
  - Process components cij yield *f(c12, c22)*

$$\frac{|A \cap B|}{|A| + |B| - |A \cap B|}.$$

- **Map-reduce Job 2 (Similarity(01, 02))**
  - **Mapper does nothing; shuffle routes and sorts components**
  - **Reducer: aggregate sub components to :**
    - **Similarity(01, 02) = f(c11, c21) + *f(c12, c22)* + f(c13, c23) +f(c14, c24)**

- **Map-reduce Job 3 (cosmetics)**
  - Sort pairs by association:

Zebra, lion 0.99
Zebra, cheetah, 098

MapReduce Jobs for Pair Similarity

Version 1: Crude approach

) + f(c13, c23) +f(c14, c24)

- **Map-reduce Job 1: route data for each *f(c12, c22)***
  - Mapper route *cij* Route the basic compo nts
  - Process components cij yield *f(c12*

$$\frac{|A \cap B|}{|A| + |B| - |A \cap B|}.$$

Slow and clunky

- **Map-reduce J ilarity(01, 02))**
  - **Mapper ng; shuffle routes and sorts components**
  - **Reducer egate sub components to :**
    - **Similarity(01, 02) = f(c11, c21) + *f(c12, c22)* + f(c13, c23) +f(c14, c24)**

- **Map-reduce Job 3 (cosmetics)**
  - Sort pairs by association:

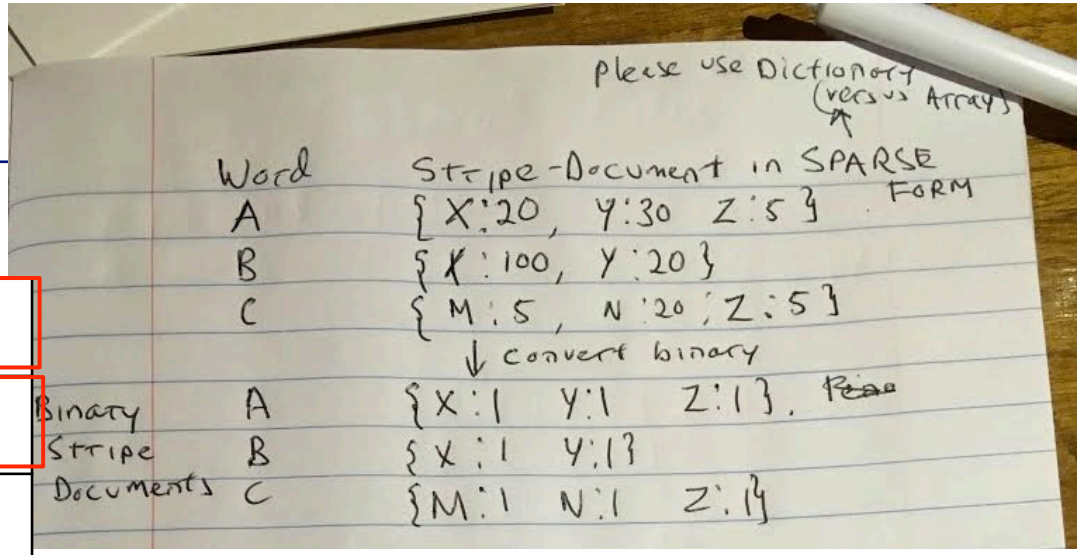
Zebra, lion 0.99
Zebra, cheetah, 098

Pairwise Similarity

- **Version 1: Clunky with dense emitting of raw input values**
  - Slow and clunky

- **Version 2: Elegant approach**
  - Level a cool data structure

	M	N	X	Y	Z
DocA	0	0	1	1	1
DocB	0	0	1	1	0
DocC	1	1	0	0	1



Inverted Index

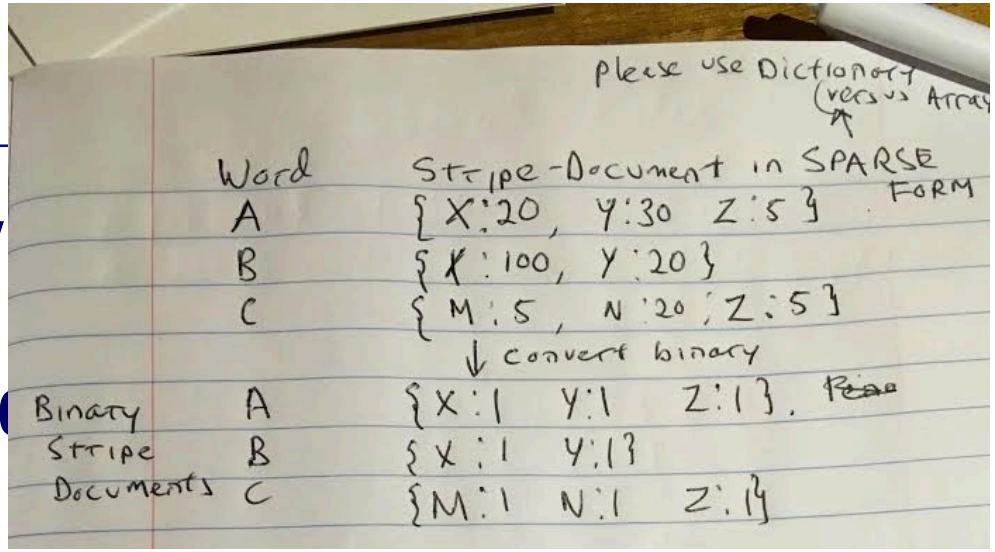
	A	B	C
M	0	0	1
N	0	0	1
X	1	1	0
Y	1	1	0
Z	1	0	1

Three terms, A,B,C and their corresponding strip-docs of co-occurring terms

DocA {X:20, Y:30, Z:5}
DocB {X:100, Y:20}
DocC {M:5, N:20, Z:5}

QUESTION:
A Inverse
B Randomize
C Transpose
D NoTA

- **Word, list of places w**
- **Zebra, d10, d100, d10,**



Co-occurrence Matrix

	w1	w2	w3	w4
w1=document1	1	0	1	0
W2=document2	1	1	0	0

Transpose the Co-occurrence Matrix

Inverted index

Inverted index
W1: {d1:1, d2:1, … d100, d10,000}
W2: {d2, d5, ……d999:1}

Mapper(Stripe)→ seq of term:doc:count

Reducer (term: {doc:count,…..} -→ postings list for ter

Decompose the similarity into sum of intermediate quantities: f(…)s

Co-occurrence matrix

o1	1	0	1	0
02	1	1	0	0

$$\frac{|A \cap B|}{|A| + |B| - |A \cap B|}.$$

Jaccard(o1, 02)= 1/3)

Similarity (A, B)
= Jacc(f(AM, BM) , f(AN, BN), …..)

Collect three quantities
- |A∧B|, |A|, |B|
- These can

Similarity(o1, o2) = f(c11, c21) + f(c12, c22) + f(c13, c23) +f(c14, c24)

f(c11, c21) = Intersection: 1|A∧B|=1; union: |A|=1, |B|=1
f(c12, c22) = Intersection: 1|A∧B|=0; union: |A|=0, |B|=1

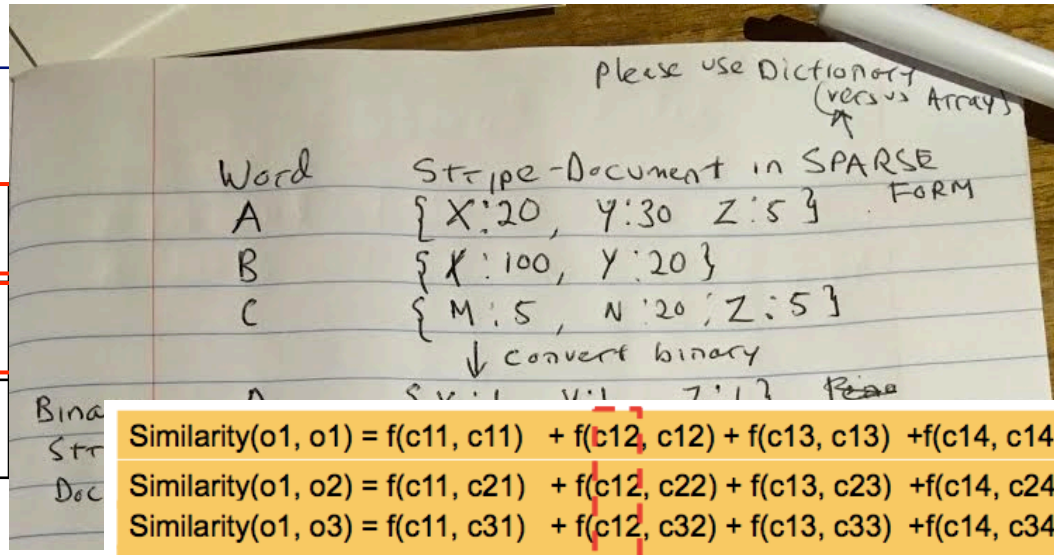
F(…) Needs c11, c21

f(c11, c21)
Intersection = 1 (or 0) if both (c11, c21) are 1
union 1 (or 0) if at least (c11, c21) has a 1

3-stage hierarchy from output to the input records
- **Similarity for a cell in the pairwise matrix**
- **Intermediate f()s**
- **Inputs for f()s**

$$\frac{|A \cap B|}{|A| + |B| - |A \cap B|}.$$

	M	N	X	Y	Z
A	0	0	1	1	1
B	0	0	1	1	0
C	1	1	0	0	1



Similarity(o1, o1) = f(c11, c11) + f(c12, c12) + f(c13, c13) +f(c14, c14
Similarity(o1, o2) = f(c11, c21) + f(c12, c22) + f(c13, c23) +f(c14, c24
Similarity(o1, o3) = f(c11, c31) + f(c12, c32) + f(c13, c33) +f(c14, c34
Similarity(o1, o4) = f(c11, c41) + f(c12, c42) + f(c13, c43) +f(c14, c44

Inverted Index

	A	B	C	D
M	0	0	1	
N	0	0	1	
X	1	1	0	1
Y	1	1	0	
Z	1	0	1	

Yield(C, 1)

Yield (AB, f(AB))
Yield(AD…)
Yield(BD)

$M_{choose\ 2}$

	A	B	C
A	Sim	Sim(AB)	Sim
B	Sim	Sim	Sim
C	Sim	Sim	Sim

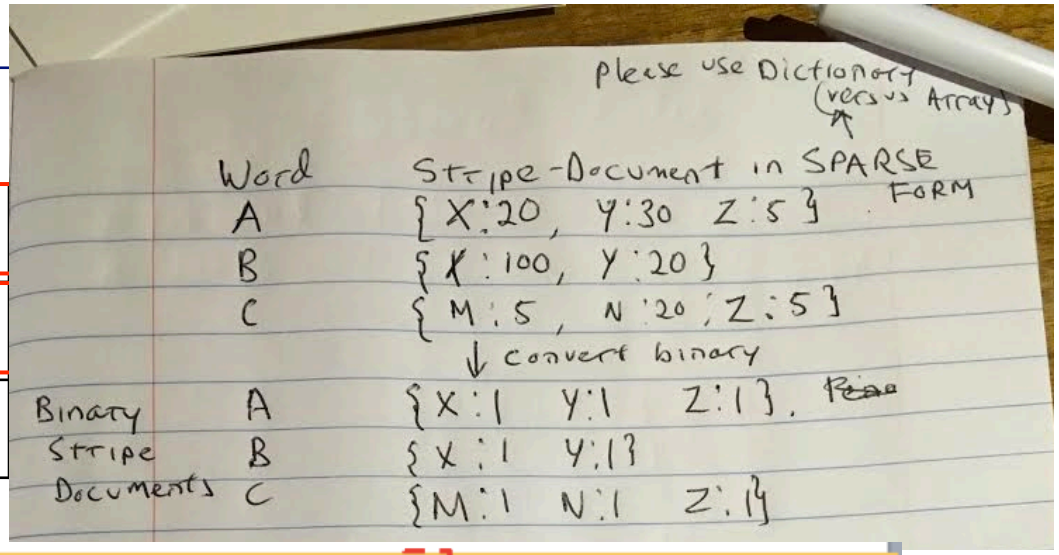
Similarity (A, B)
= Jacc(f(AM, BM) , f(AN, BN), …..)

F(AM, BM) Collect three quantities
- $|A \wedge B|$, |A|, |B|

Similarity (A, B)

$$\frac{|A \cap B|}{|A| + |B| - |A \cap B|}.$$

	M	N	X	Y	Z
A	0	0	1	1	1
B	0	0	1	1	0
C	1	1	0	0	1



Inverted Index

	A	B	C
M	0	0	1
N	0	0	1
X	1	1	0
Y	1	1	0
Z	1	0	1

NOTHING

NOTHING

Yield(AB, f(1,1)=[1,1,1,1])
AC
BC

Similarity(o1, o1) = f(c11, c11) + f(c12, c12) + f(c13, c13) +f(c14, c14)
Similarity(o1, o2) = f(c11, c21) + f(c12, c22) + f(c13, c23) +f(c14, c24)
Similarity(o1, o3) = f(c11, c31) + f(c12, c32) + f(c13, c33) +f(c14, c34)
Similarity(o1, o4) = f(c11, c41) + f(c12, c42) + f(c13, c43) +f(c14, c44)

A

B

C

Sim	Sim(AB)	Sim
Sim	Sim	Sim
Sim	Sim	Sim

Similarity (A, B)
= Jacc(f(AM, BM) , f(AN, BN), …..)

Similarity (A, B)

Aggregating by rows versus columns

Pairwise Document Similarity in MapReduce

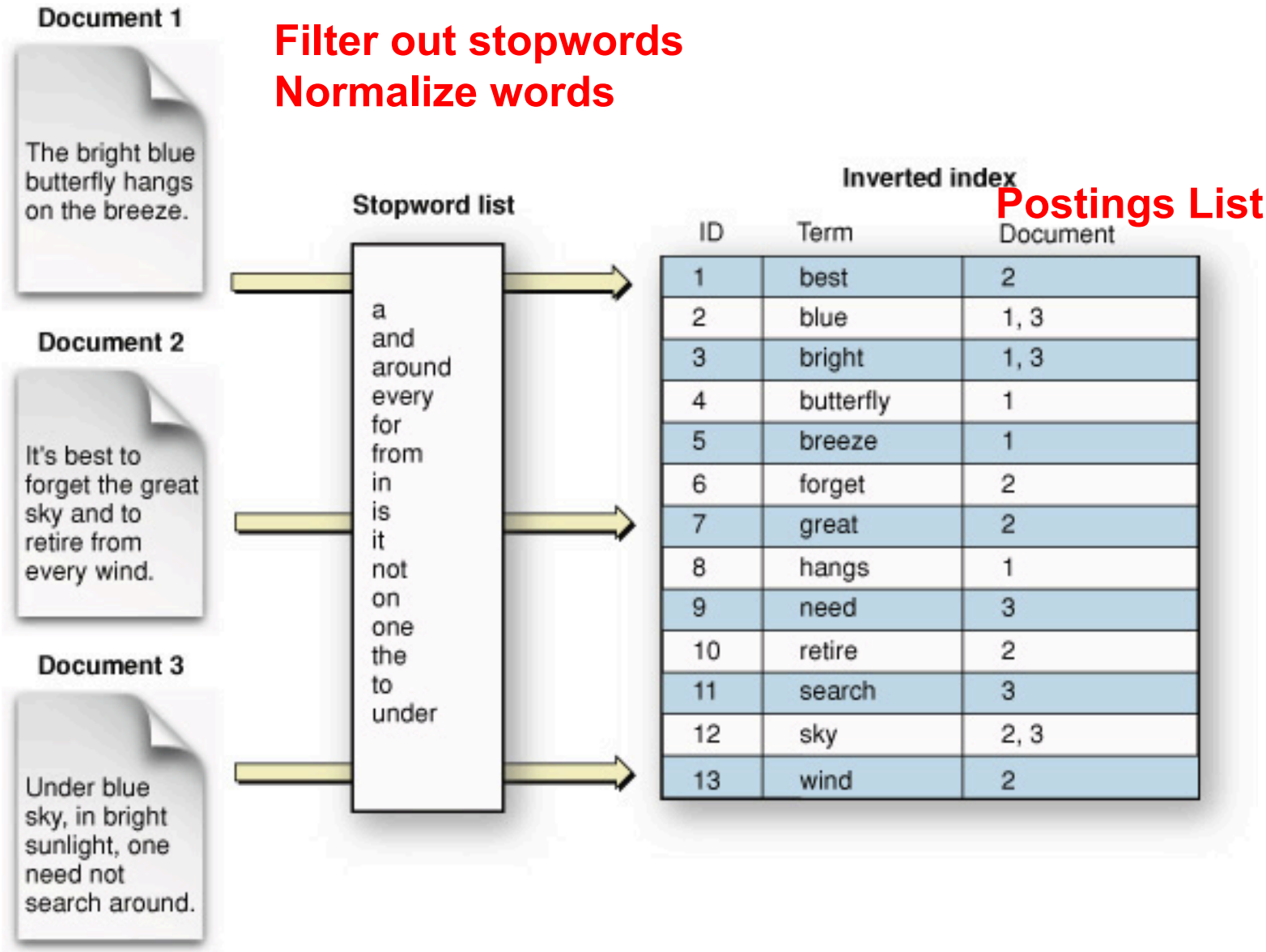
- Pairwise Document Similarity in Large Collections with MapReduce
- Tamer Elsayed,* Jimmy Lin,† and Douglas W. Oard†

- https://terpconnect.umd.edu/~oard/pdf/acl08elsayed2.pdf

Inverted Index:

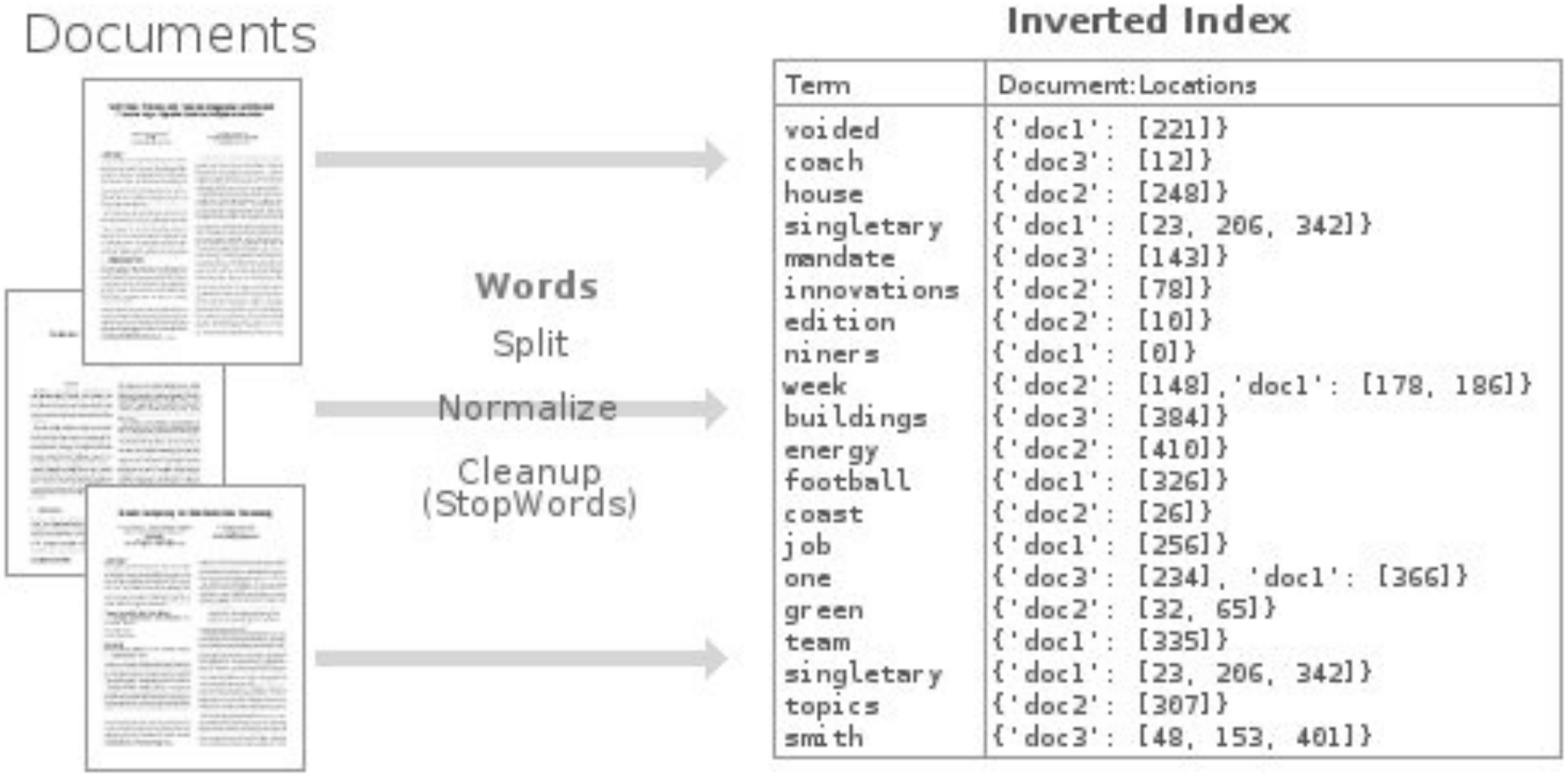
Each Posting consists of (DocID, PayLoad)

Filter out stopwords
Normalize words



ID	Term	Document
1	best	2
2	blue	1, 3
3	bright	1, 3
4	butterfly	1
5	breeze	1
6	forget	2
7	great	2
8	hangs	1
9	need	3
10	retire	2
11	search	3
12	sky	2, 3
13	wind	2

Inverted Index



Filter out stopwords and common words

- **Filter out documents with low IDF (high document**

TF-IDF

f_{ij} = frequency of term (feature) ***i*** in doc (item) ***j***

$$TF_{ij} = \frac{f_{ij}}{\max_k f_{kj}}$$

n_i = number of docs that mention term ***i***

N = total number of docs

$$IDF_i = \log \frac{N}{n_i}$$

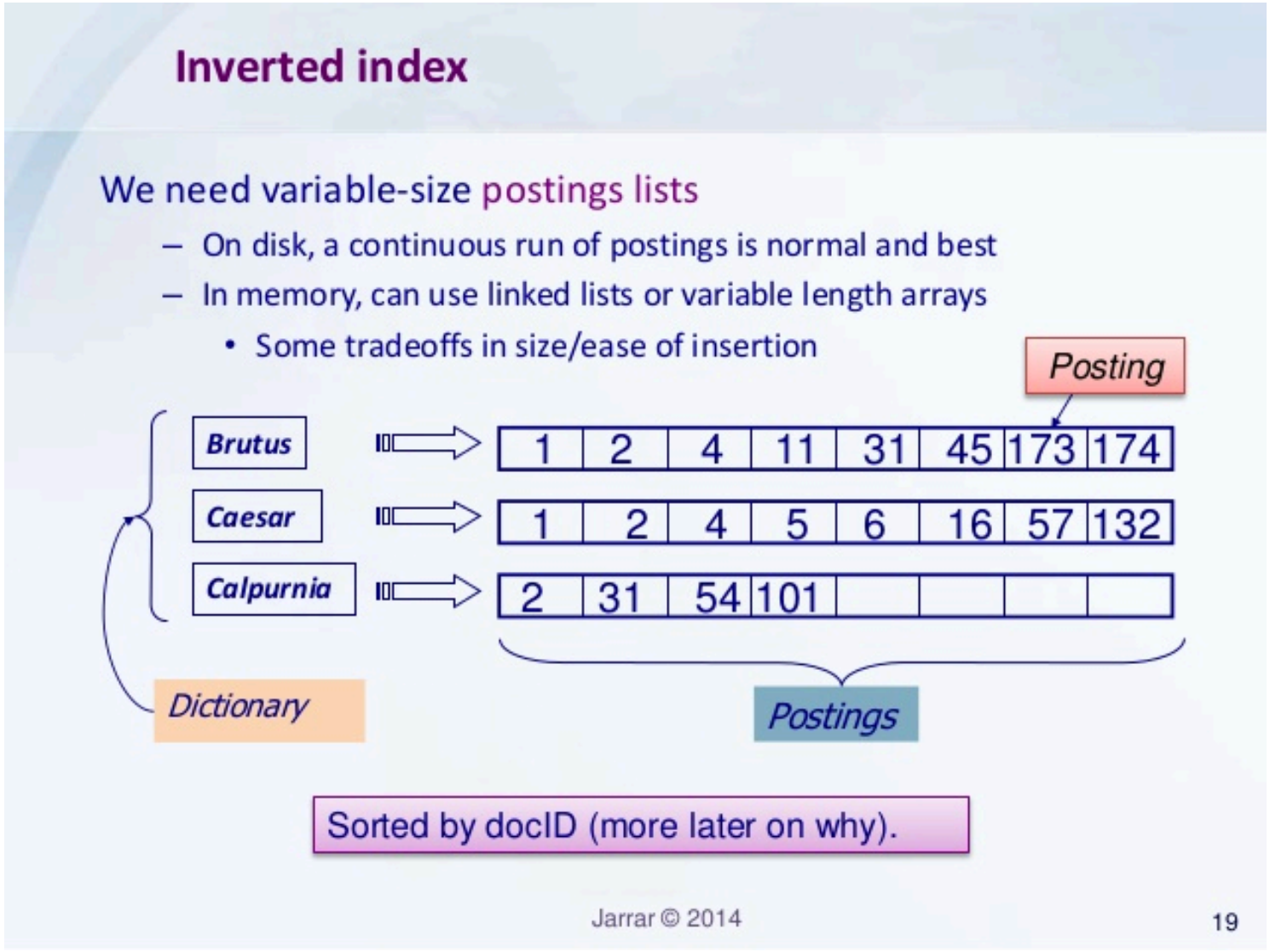
TF-IDF score: $w_{ij} = TF_{ij} \times IDF_i$

10M Docs
Term in 1M

Log(10M) – Log (1M)
7-6 = 1

10,000
7-4 = 3

Postings consist of list of postings/position information
Posting consists of (DocID, PayLoad)



Data Structure: inverted index

- In computer science, an inverted index (also referred to as postings file or inverted file) is an index data structure storing a mapping from content, such as words or numbers, to its locations in a database file, or in a document or a set of documents.

Build up stripes for each word. Each stripe can viewed as a pseudo document

Each word is associated with an object or data structure (could view as a document)

C12 denotes (w1 cooccurs with w2)

o1	**c11**	**c12**	**c13**	**c14**
o2	c21	c22	c23	c24
o3	c31	c32	c33	c34
o4	c41	c42	c43	c44

Assume a trigram world

W1 w4 **w2**

W1 w3 **w2**

Term co-occurrence matrix
Can be viewed as bunch of stripes. And then these stripes can be viewed as pseudo documents

Compute term by term similarity matrix

1. **Index these pseudo documents, i.e., build an inverted index from these pseudo documents (stripes)**
2. **Compute document-to-document (pairwise) similarity**

On single node

Create inverted Index from the Stripe Documents

Compute doc by doc similarity by summing up partial similarity scores

D1=Lion: Aardvark:10, Spark:2, zebra:6
D10=Cheetah: Spark:3, zebra:9

Stripe documents for terms Lion and Cheetah

Cosine Similarity of (D1, D10) = (10 x 0) + (2 x 3) + (6 x 9) Numerator part of Cosine

Now do the same calculation using an inverted index on single node

Index

Term, List of documents
(AKA Postings List)
$term_i$: {…($term_j$, count c_{IJ}),..}
Aardvark: {stripe1 or D1:10, D100:3, D10000:3}
…..
Spark: {D1:2, D10:3, D20:5, ……. D10,000:4}
…..
Zebra: {D1:6, D2:3, D4:4, D10:9}

BUT can NOT do it easily for with an inverted index

Give documents d1 and d2

Extract the set of words W that are in both documents
similarity =0
For each word w in the W do
  postings = Index[w]
  countd1 = countd2 =0
  for each (doc, count) in postings
    if (doc == d1) then countd1=count
    if (doc == d2) then countd2=count
  end for
  similarity = countd1 x countd2
End for

$$sim(d_i, d_j) = \sum_{t \in d_i \cap d_j} term\_contrib(t, d_i, d_j)$$

Sim(d1 and d2) *t in{"spark" and "zebra"}*

Compute doc by doc similarity by summing up partial similarity scores

D1=Lion: Aardvark, Spark, zebra, …, …
D10=Cheetah: Spark, zebra, …

Stripe documents for terms Lion and Cheetah

Index

Term, List of documents
(AKA Postings List)
$term_i$: {…($term_j$, count c_{IJ}),..}
Aardvark: {stripe1 or D1:1, D100:3, D10000
…..
Spark: {D1:2, D10:3, D20:5, ……. D10,000:
…..
Zebra: {D1:6, D2:3, D4:4, D10:9}

Similarity of stripe 1 (Lion) and 10 (cheetah)
Can be computed by visited the postings list for each shared term
Here we visit postings for words "Aardvark", "spark" and "zebra" (assume for now an oracle told us about these words of interest).
Then process each postings list and focus on the pair of documents of interest (D1 and D10 here).
Say for the Spark postings, using the cosine measure just take the product of 3 X 2 = 6
Repeat for Aardvark (0 partial similarity), and for Zebra (6X9 =54).
Then sum these partial similarities (6+0+54=60 for unnormalized cosine distance)

$$sim(d_i, d_j) = \sum_{t \in d_i \cap d_j} term\_contrib(t, d_i, d_j)$$

t in{"spark" and "zebra"}

How distribute the pairwise similarity calculation given the documents are embedded in an inverted index

- **How distribute the pairwise similarity calculation given the documents are embedded in an inverted index**

MapReduce Jobs: work back from output layer to the inputs

- **Problem: Pairwise similarity of (stripe) documents**
  - Unit of parallelism: pair similarity Sim(o1, o2)
- **Decompose the pair similarity measure into intermediate components:**

  Similarity(o1, o2) = f(c11, c21) + *f(c12, c22)* + f(c13, c23) +f(c14, c24)
- **For the intermediate components source the data that they use**
  - It turns out that Cij comes from the postings records that are input to the this item-to-item similarity job.
  - Record: $term_i$: {…($term_j$, count c_{IJ}),..}
  - Iterate over each element in the index and for each pair of elements in the postings list generate an intermediate component f(c, c)….

MapReduce Jobs: work back from output layer to the inputs

- **Problem: Pairwis**
  - Unit of parallelism:
- **Decompose the p**
  intermediate com

Similarity(o1, o2) **(c14, c24)**

- **For the intermedi** **that they use**
  - It turns out that Cij comes from the postings records that are input to the this item-to-item similarity job.
  - Record: $term_i$: {…($term_j$, count c_{IJ}),..}
  - Iterate over each element in the index and for each pair of elements in the postings list generate an intermediate component f(c, c)….

```
Mapper(key = word; value=postingsList)
For each pair of documents d1 and d2  in
the postings  do
    yield ( (d1,d2),  countd1 x
            countd2)
End for

Reducer ((d1, d2), partialSimilarities)
   yield ((d1, d2), sum partialSimilarities)
```

Normalized Vectors

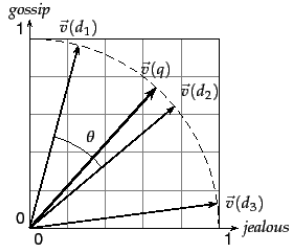
- **Assume stripe Document A = {X:1, Y:1, Z:1};**
- **The vector is length is**
  - SQRT{(1)^2, (1)^2, (1)^2} = SQRT(3)
- **So the normalized vector for Document**
  - A = {X:1/SQRT(3), Y:1/SQRT(3), Z:1/SQRT(3)}
- **with a vector length of for normalized Document A = 1 since:**

- **(1/SQRT(3))^2 + (1/SQRT(3))^2 + (1/SQRT(3))^3 = 1/3+1/3 + 1/3=1**

Dot Products – Cosine Similarity

We denote by $\vec{V}(d)$ the vector derived from document d, with one component in the vector for each dictionary term. Unless otherwise specified, the reader may assume that the components are computed using the tf-idf weighting scheme, although the particular weighting scheme is immaterial to the discussion that follows. The set of documents in a collection then may be viewed as a set of vectors in a vector space, in which there is one axis for each term. This representation loses the relative ordering of the terms in each document; recall our example from Section 6.2 (page), where we pointed out that the documents Mary is quicker than John and John is quicker than Mary are identical in such a *bag of words* representation.

How do we quantify the similarity between two documents in this vector space? A first attempt might consider the magnitude of the vector difference between two document vectors. This measure suffers from a drawback: two documents with very similar content can have a significant vector difference simply because one is much longer than the other. Thus the relative distributions of terms may be identical in the two documents, but the absolute term frequencies of one may be far larger.



► **Figure 6.4** Cosine similarity illustrated. $\text{sim}(d_1, d_2) = \cos\theta$.

To compensate for the effect of document length, the standard way of quantifying the similarity between two documents d_1 and d_2 is to compute the *cosine similarity* of their vector representations $\vec{V}(d_1)$ and $\vec{V}(d_2)$

$$\text{sim}(d_1, d_2) = \frac{\vec{V}(d_1) \cdot \vec{V}(d_2)}{|\vec{V}(d_1)||\vec{V}(d_2)|}, \tag{24}$$

where the numerator represents the *dot product* (also known as the *inner product*) of the vectors $\vec{V}(d_1)$ and $\vec{V}(d_2)$, while the denominator is the product of their *Euclidean lengths*. The dot product $\vec{x} \cdot \vec{y}$ of two vectors is defined as $\sum_{i=1}^{M} x_i y_i$. Let $\vec{V}(d)$ denote the document vector for d, with M components $\vec{V}_1(d) \ldots \vec{V}_M(d)$. The Euclidean length of d is defined to be $\sqrt{\sum_{i=1}^{M} \vec{V}_i^2(d)}$.

The effect of the denominator of Equation 24 is thus to *length-normalize* the vectors $\vec{V}(d_1)$ and $\vec{V}(d_2)$ to unit vectors $\vec{v}(d_1) = \vec{V}(d_1)/|\vec{V}(d_1)|$ and $\vec{v}(d_2) = \vec{V}(d_2)/|\vec{V}(d_2)|$. We can then rewrite (24) as

$$\text{sim}(d_1, d_2) = \vec{v}(d_1) \cdot \vec{v}(d_2). \tag{25}$$

Worked example. Consider the documents in Figure 6.9. We now apply Euclidean normalization to the tf values from the table, for each of the three documents in the table. The quantity $\sqrt{\sum_{i=1}^{M} \vec{V}_i^2(d)}$ has the values 30.56, 46.84 and 41.30 respectively for Doc1, Doc2 and Doc3. The resulting Euclidean normalized tf values for these documents are shown in Figure 6.11.

	Doc1	Doc2	Doc3
car	27	4	24
auto	3	33	0
insurance	0	33	29
best	14	0	17

Figure 6.9: Table of tf values for Exercise 6.2.2.

	Doc1	Doc2	Doc3
car	0.88	0.09	0.58
auto	0.10	0.71	0
insurance	0	0.71	0.70
best	0.46	0	0.41

Figure 6.11: Euclidean normalized tf values for documents in Figure 6.9.

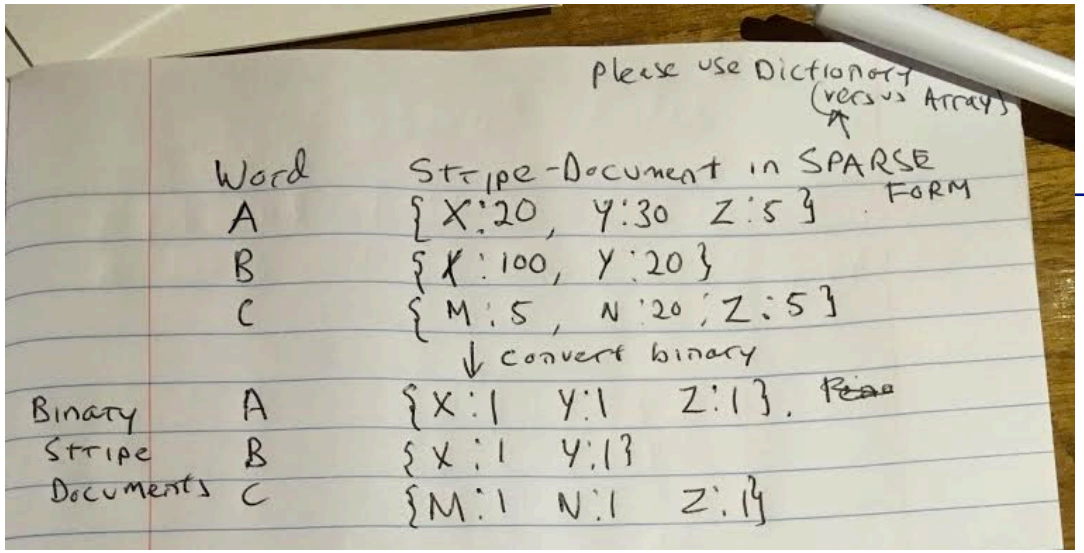
End worked example.

http://nlp.stanford.edu/IR-book/html/htmledition/tf-idf-weighting-1.html

http://nlp.stanford.edu/IR-book/html/htmledition/dot-products-1.html

Worked example in MapReduce for Cosine-based pairwise similarity matrix

- **See next slide**


Please use Dictionary (versus Array)
Word
Stripe-Document in SPARSE FORM
A
{X:20, Y:30 Z:5}
B
{X:100, Y:20}
C
{M:5, N:20, Z:5}
convert binary
Binary Stripe Documents
A
{X:1 Y:1 Z:1}.
B
{X:1 Y:1}
C
{M:1 N:1 Z:1}

- **Try to write the Map reduce functions yourself before going to the next slides**

Please use Dictionary (versus Arrays)

Word	Stripe-Document in SPARSE FORM
A	{X:20, Y:30 Z:5}
B	{X:100, Y:20}
C	{M:5, N:20, Z:5}

↓ convert binary

Binary Stripe Documents		
	A	{X:1 Y:1 Z:1}
	B	{X:1 Y:1}
	C	{M:1 N:1 Z:1}

length of stripe A $= \sqrt{(1)^2+(1)^2+(1)^2} = \sqrt{3}$

Normalized stripe documents		
	A	$\{X:\frac{1}{\sqrt{3}}, Y:\frac{1}{\sqrt{3}}, Z:\frac{1}{\sqrt{3}}\}$
	B	$\{X:\frac{1}{\sqrt{2}} \; Y:\frac{1}{\sqrt{2}}\}$
	C	$\{M:\frac{1}{\sqrt{3}} \; N:\frac{1}{\sqrt{3}} \; Z:\frac{1}{\sqrt{3}}\}$

Inverted Index	Index Term	Postings
	M	{StripeC: $\frac{1}{\sqrt{3}}$} ← record
	N	{StripeC: $\frac{1}{\sqrt{3}}$}
	X	{StripeA: $\frac{1}{\sqrt{3}}$, StripeB: $\frac{1}{\sqrt{2}}$}
	Y	{StripeA: $\frac{1}{\sqrt{3}}$ StripeB: $\frac{1}{\sqrt{2}}$}
	Z	{StripeA: $\frac{1}{\sqrt{3}}$ StripeC $\frac{1}{\sqrt{3}}$}

X∈A X∈B Y∈A Y∈B Z∈A Z∈B

Cosine(StripeA, StripeB) $= \frac{1}{\sqrt{3}} \cdot \frac{1}{\sqrt{2}} + \frac{1}{\sqrt{3}} \cdot \frac{1}{\sqrt{2}} + \frac{1}{\sqrt{3}} \cdot 0$

In terms of map-Reduce

Mapper processes the inverted index record by record emitting pairs of stripe documents and their associated partial/term-centric similarities.

postings for M & N yield nothing to the stream. WHY (Hint no pairs)

(StripeA, StripeB), $\frac{1}{\sqrt{6}}$

(StripeA, StripeB), $\frac{1}{\sqrt{6}}$

(StripeA, StripeC), $\frac{1}{\sqrt{9}}$

Reducer outputs

StripeA, StripeB $= \frac{2}{\sqrt{6}}$

StripeA, StripeC $= \frac{1}{\sqrt{9}}$

	StripeA	StripeB	StripeC
StripeA	1	$\frac{2}{\sqrt{6}}$	$\frac{1}{\sqrt{9}}$
StripeB	$\frac{2}{\sqrt{6}}$	1	0
StripeC	$\frac{1}{\sqrt{9}}$	0	1

Pairwise Similarity Matrix using Cosine Similar

Stripe Documents → Normalized Docs

Stripe for A {X:10/17, Y:4/17, Z:3/17}

Stripe for B {X:5/8, Y:3/8}

Stripe for C {M:3/7, N:1/7, Z:3/7}

Binarize the data (just to simplify)

Stripe for A {X:1, Y:1, Z:1}

Stripe for B {X:1, Y:1}

Stripe for C {M:1, N:1, Z:1}

Normalize the data to vectors of unit length for Cosine purposes

Stripe for A {X:1/√3, Y:1/√3, Z:1/√3}

Stripe for B {X:1/√2, Y:1/√2}

Stripe for C {M:1/√3, N:1/√3, Z:1/√3}

Cosine Inverted Index

Cosine (StripeA, StripeB) =(1/√3 X 1/√2 + 1/√3 X 1/√2) = 2/√6

Stripe for A {X:1/√3, Y:1/√3, Z:1/√3}

Stripe for B {X:1/√2, Y:1/√2, Z:0.0}

Stripe for C {M:1/√3, N:1/√3, Z:1/√3}

Build Cosine Inverted Index

Term

Posting list: postin with a docID and a payload of weight for the term in that doc

M: {StripeC:*1/√3*}

N: {StripeC:1/√3}

X: {StripeA:1/√3, StripeB:1/√2}

Y: {StripeA:1/√3, StripeB:1/√2}

Z: {StripeA:1/√3, StripeC:1/√3}

Mappers and Reducers for pairwise cosine Similarity

M: {StripeC:1/√3}
N: {StripeC:1/√3}
X: {StripeA:1/√3, StripeB:1/√2, StripeC:1/√19999}
Y: {StripeA:1/√3, StripeB:1/√2}
Z: {StripeA:1/√3, StripeC:1/√3}

Record: key, dictionary value

Bill O'Reilly
Doing it Live

Mapper(key = word; value=postingsList)
For each pair of documents (d1:payload1) and (d2:payload1) in the postings do
weight1= payload1; weight2= payload2;
yield ((d1,d2), weight1 x weight2)
End for

{StripeA, StripeB} 1/√6
{StripeA, StripeC} 1/√59997
{StripeB, StripeC} 1/√39998
{StripeA, StripeB} 1/√6
{StripeA, StripeC} 1/√9

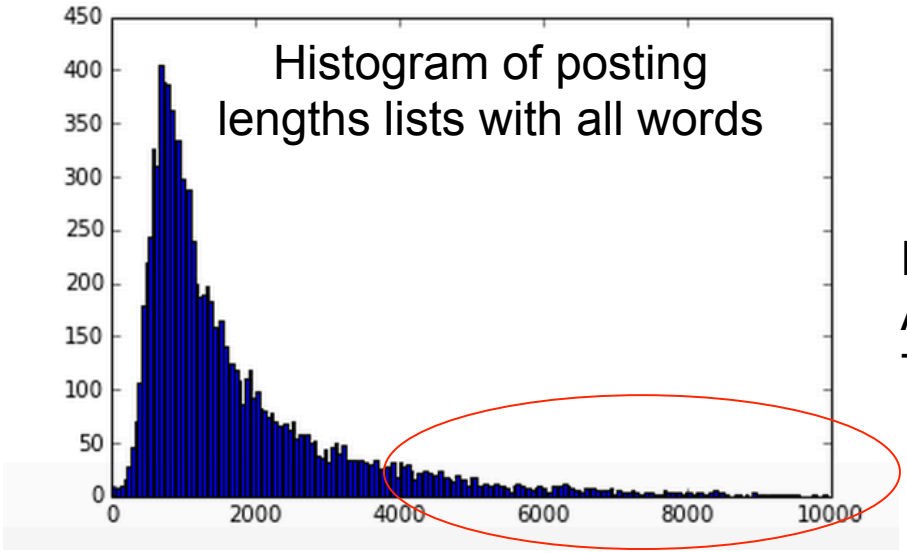
Reducer ((d1, d2), partialSimilarities)
yield ((d1, d2), sum partialSimilarities)

{StripeA, StripeB} 2/√6
{StripeA, StripeC} 1/√9 + *1/√59997*
{StripeB, StripeC} 1/√39998

Combiner?

Why is my job taking so long? Do some EDA and try to learn something from the data.

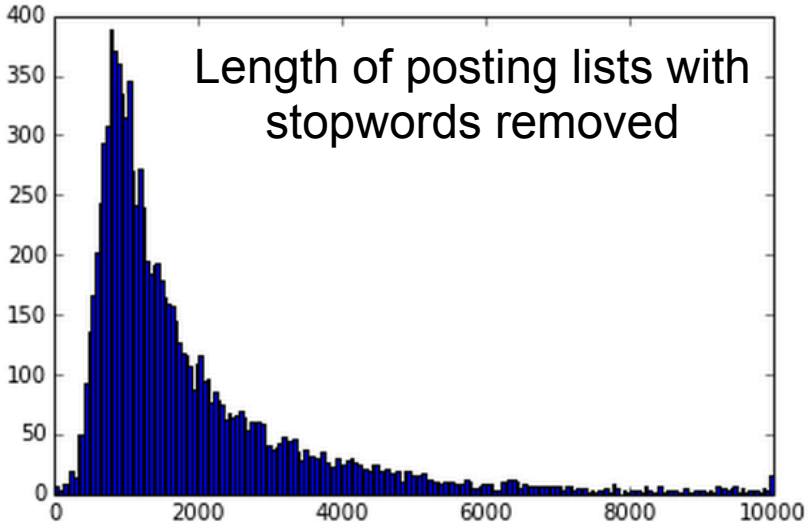
```
        9844.68,  9894.34,  9944.  ]),
<a list of 200 Patch objects>)
```



Examine the Length of posting lists with all words. Here we plot the histogram

10^8 = (10,000 X 9,999) pairs / 2

Posting length of >= 5,000?
Are a problem: provide no discrimination.
Think IDF type discrimination/separation.



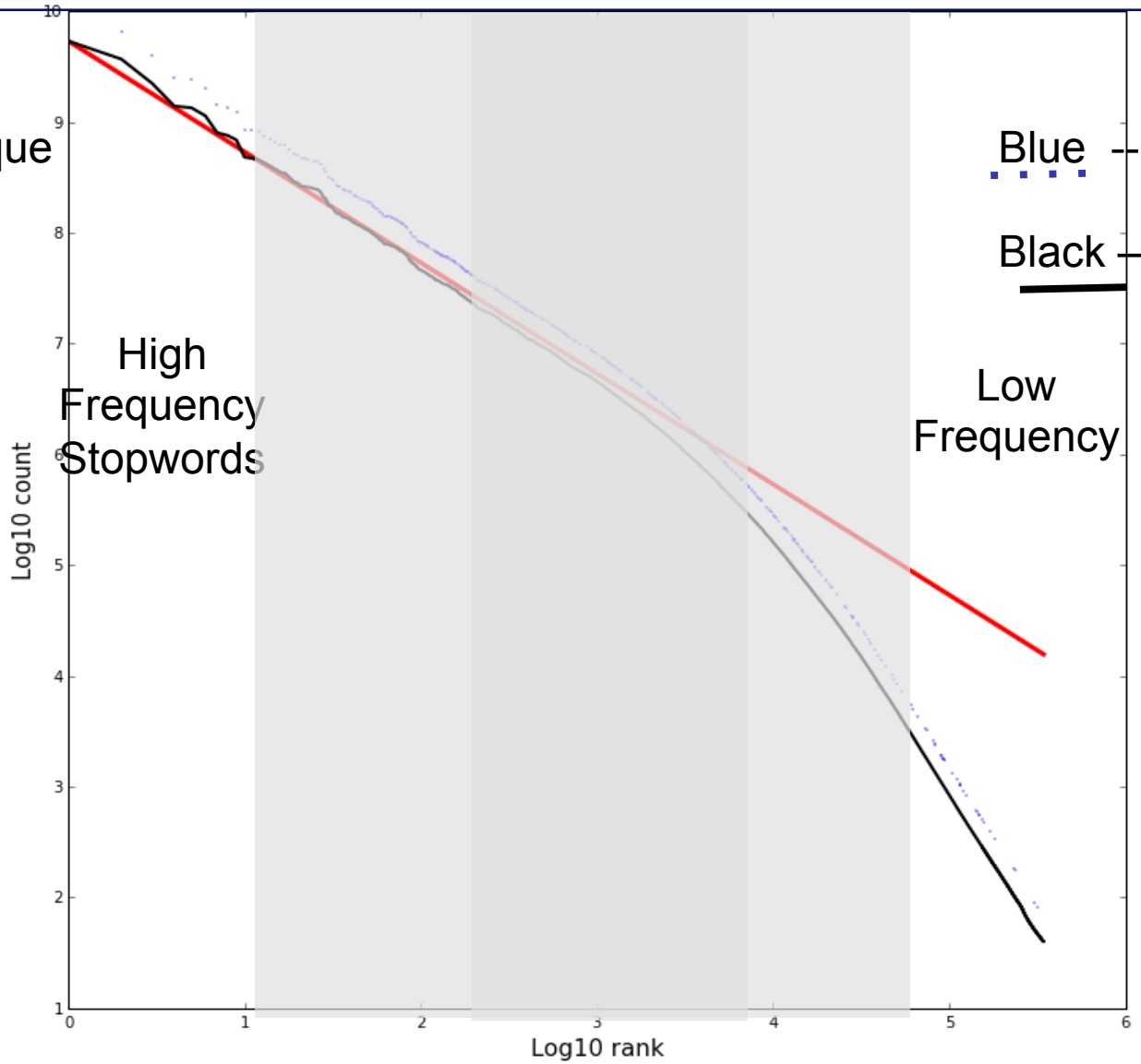
```
!head -25 sortedWordCounts.txt
```

```
the     5375699242
of      3691308874
to      2221164346
in      1387638591
a       1342195425
and     1135779433
that    798553959
is      756296656
be      688053106
as      481373389
was     469941121
for     454742998
it      426786974
not     398897768
with    372888370
on      351850709
by      344380381
have    316710855
he      288925226
which   281528146
his     263185718
at      260409177
had     256489364
I       255205575
are     247721045
```

```
from matplotlib import pyplot as plot
import re,math
%matplotlib inline

ranks = []
counts = []
stopWords = {}
stopRanks = []
stopCounts = []

f = open("stopWords.txt","r")
for word in f:
    word = word.strip()
    stopWords[word] = 1

rank = 1
f = open("sortedWordCounts.txt","r")
for line in f:
    line.strip()
    word,count = re.split("\t",line)
    count = int(count)
    if word in stopWords.keys():
        stopRanks.append(math.log(rank,10))
        stopCounts.append(math.log(count,10)+0.25)
    ranks.append(math.log(rank,10))
    counts.append(math.log(count,10))
    rank += 1

rnge = [min(ranks),max(ranks)]
b = max(counts)
m = -1
lin = [rnge[0]*m + b,rnge[1]*m + b]

plot.figure(figsize=(12,12))

plot.subplot(1,1,1)
plot.plot(rnge,lin,'red',lw=3)
plot.plot(ranks,counts,'black',lw=2)
plot.plot(stopRanks,stopCounts,'.',ms=1.25)
plot.xlabel('Log10 rank',fontsize=15)
plot.ylabel('Log10 count',fontsize=15)
```

Document Frequency: Log_{10}-Log_{10}

300,000 unique words



- **End of Lecture**