CC5212-1
Procesamiento Masivo de Datos
Otoño 2016

Lecture 7: Information Retrieval I

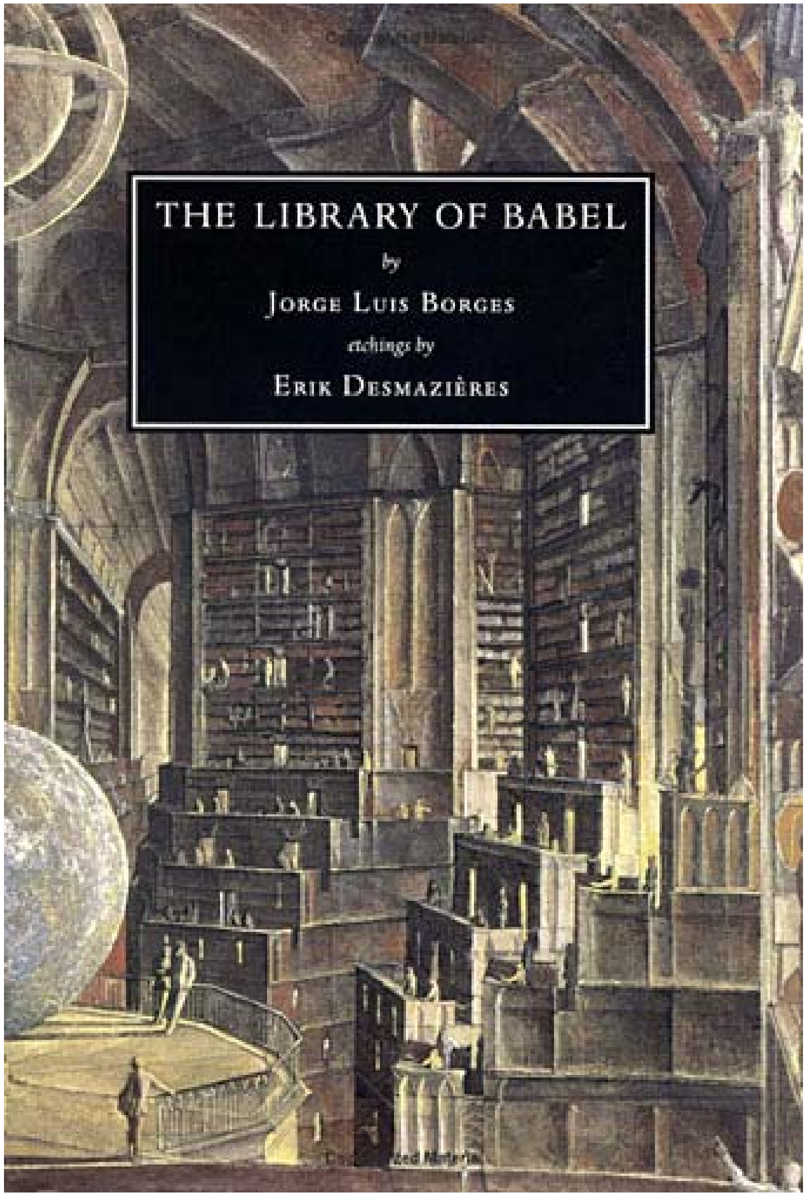
Aidan Hogan
aidhog@gmail.com


MapReduceBase
HDFS
grunt
Pig
replication
Sort
replicas
Hive
Rack-awareness
MapReduce
Partitioner
JobNode
JobTracker
ChunkServer
GFS
chunks
Hadoop
Reporter
Mapper
Writable
Shuffle
NameNode
Pipelined-reads
Reducer
Combiner
WritableComparable
DataNode
SecondaryNameNode

MANAGING TEXT DATA

Information Overload

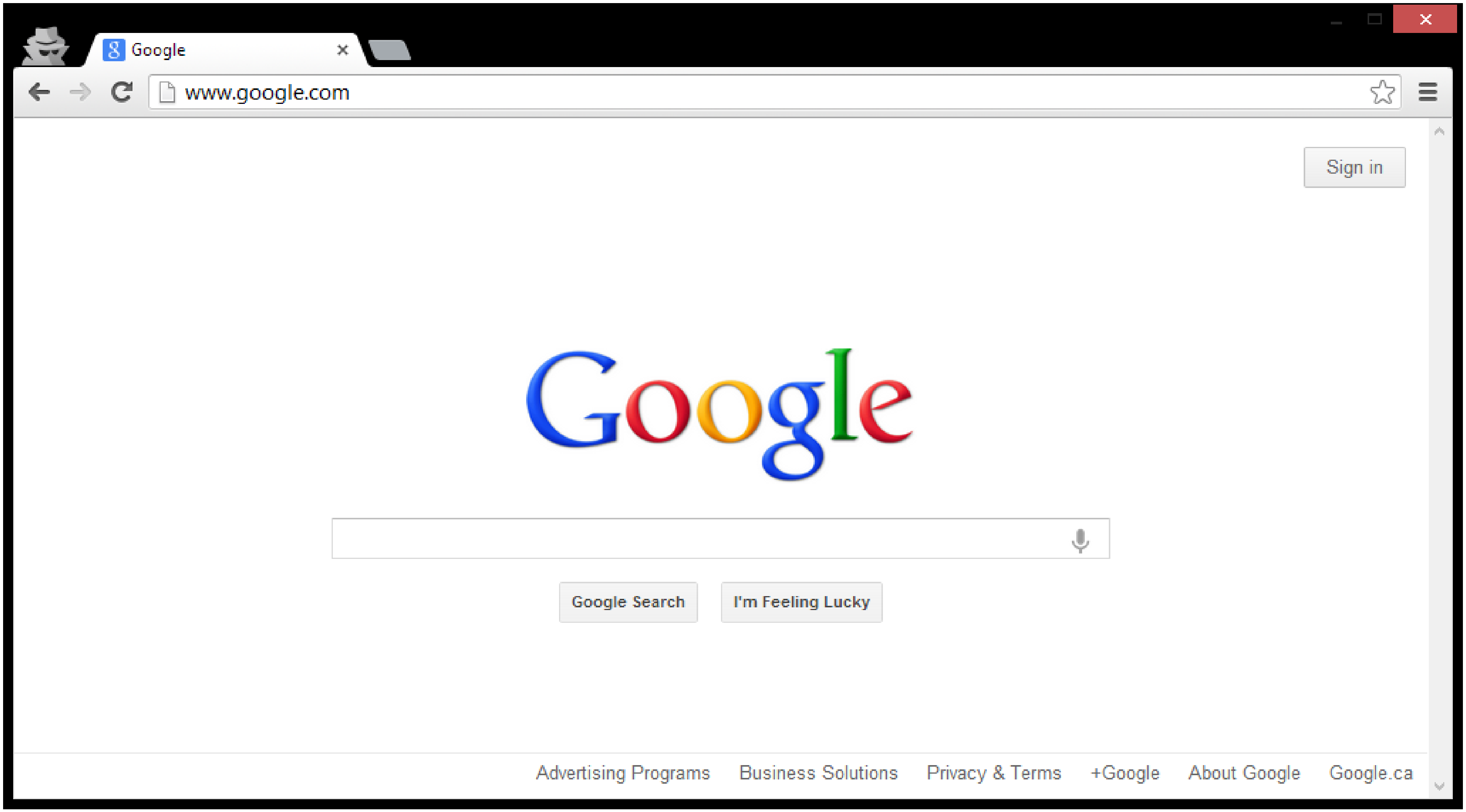
If we didn't have search …



- Contains all books with
  - 25 unique characters
  - 80 characters per line
  - 40 lines per page
  - 410 pages
  - 410 x 40 x 80 = 1,312,000 chars
  - $25^{1,312,000}$ books
- Would contain any book imaginable
  - Including a book with the location of useful books ;)

All information = Zero information

The book that indexes the library



SEARCH, QUERY, RETRIEVAL

Search, Query & Retrieval

- Search: the goal/aim of the user
- Query: the expression of a search
- Retrieval: the machine method to “solve” a query

… roughly speaking

Retrieval

1. Machine has a bunch of information resources of some sort (let's call it a set I)
   – e.g., documents, movie pages, actor descriptions
2. A user search wants to find some subset of I
   – e.g., Irish actors, documents about Hadoop
3. User expresses search criteria as a query Q
   – e.g., "irish actors", "hadoop", "SELECT ?movie …"
4. Retrieval engine returns results: R is a minimal subset of I relevant to Q
5. Results R may be ordered by a ranking
   – e.g., by most famous Irish actors

Data Retrieval

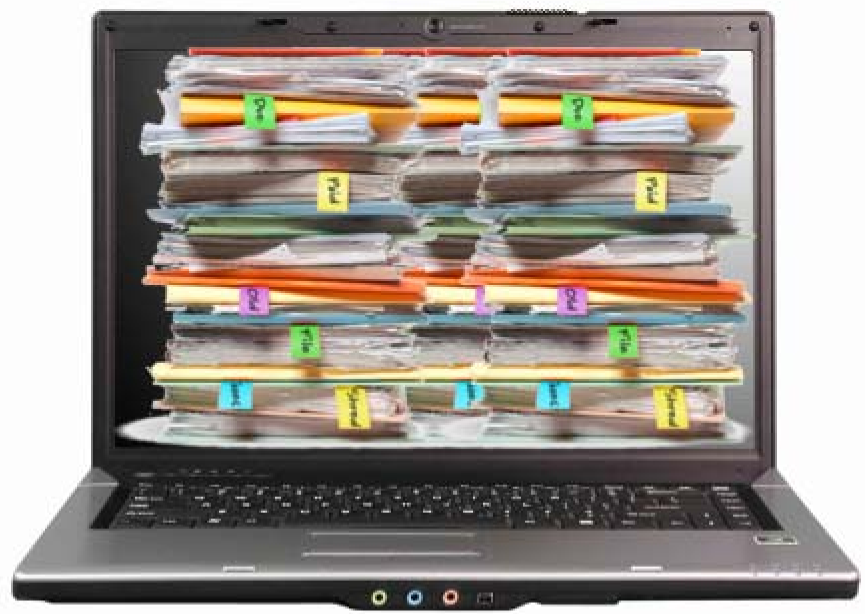
- Retrieval over "structured data"

- Typical of databases
  - I is a dataset, e.g., a set of relations
  - Q is a structured query, e.g., SQL
  - R is a list of tuples, possibly ordered

SELECT * FROM actor WHERE country = "Ireland" ORDER BY earnings;

Information Retrieval

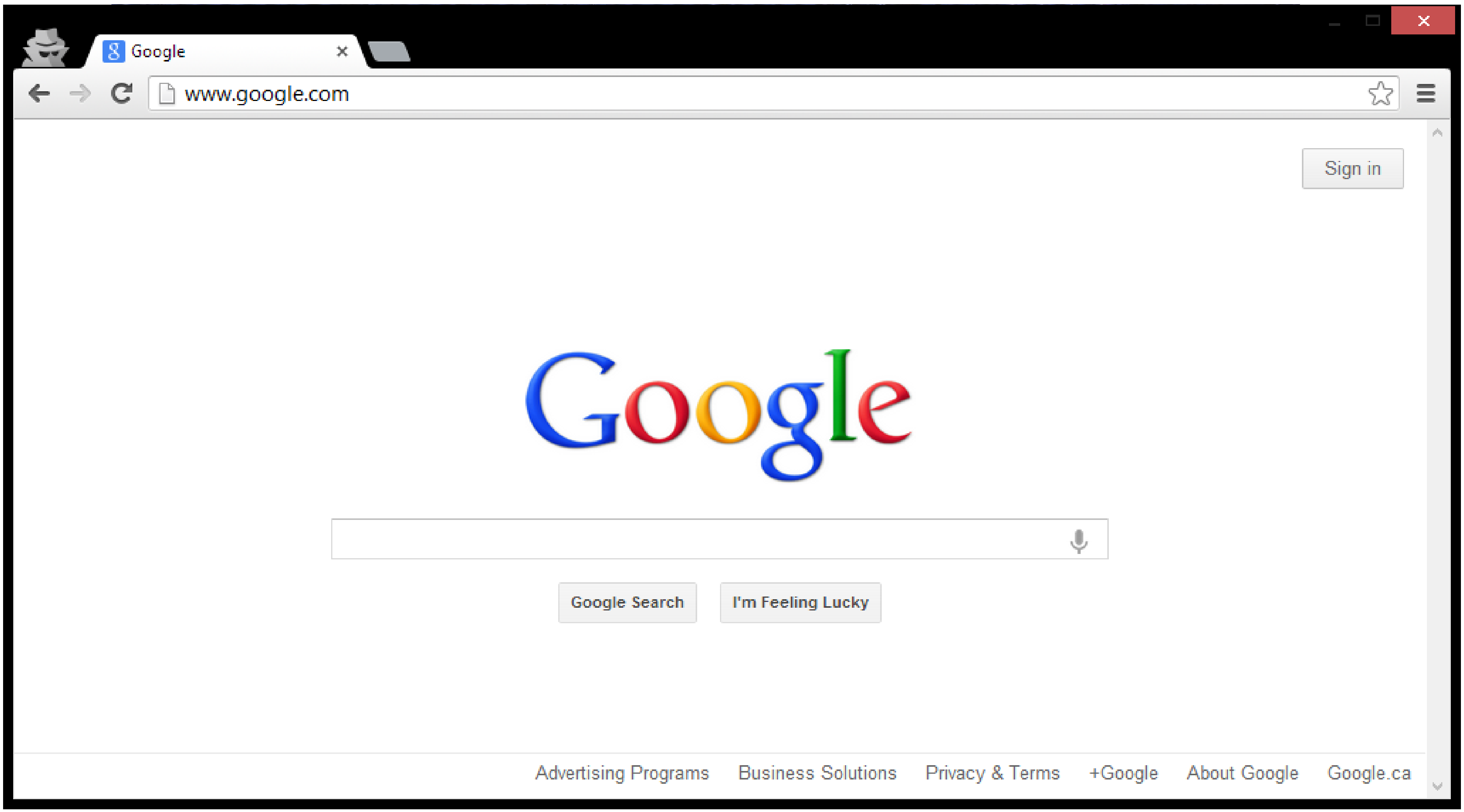
- Retrieval over "unstructured data" or textual data
- Typical of web search
  - I is a set of text documents, e.g., web pages
  - Q is a keyword query
  - R is a list of documents, e.g., relevant pages



"most famous Irish actors"

WEB SEARCH/RETRIEVAL

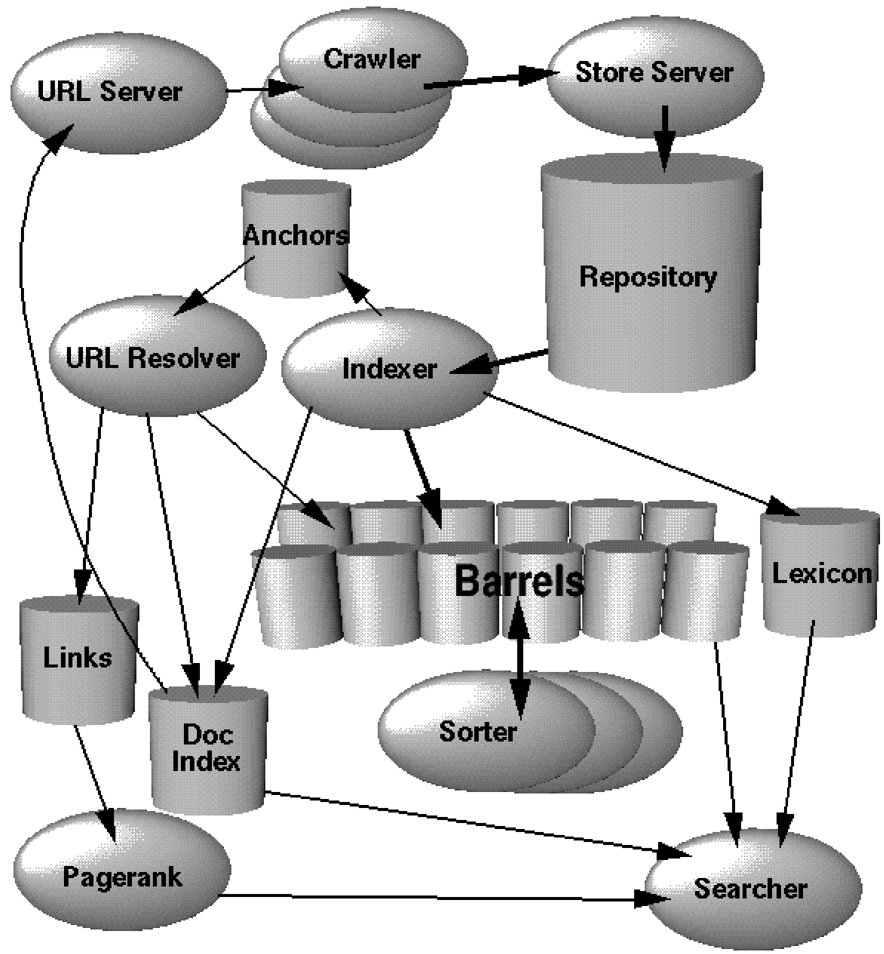
Inside Google



Google Architecture (ca. 1998)

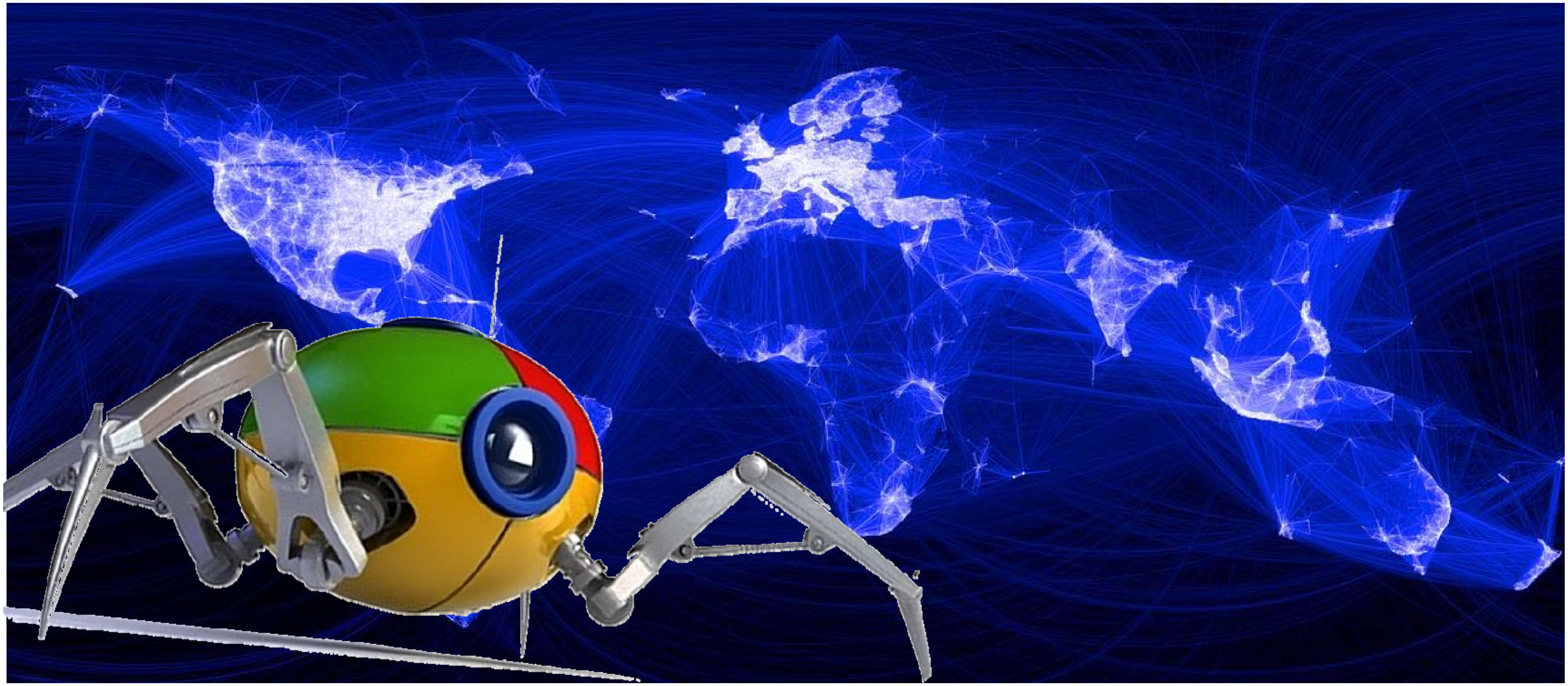
Information Retrieval

- Crawling
- Inverted indexing
- PageRank



INFORMATION RETRIEVAL: CRAWLING

How does Google know about the Web?



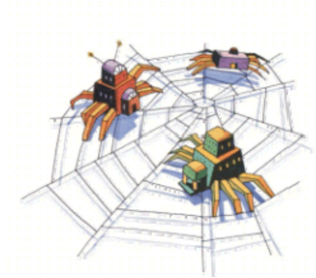
Crawling

- Download the Web. ☺

```
crawl(list seedUrls)
    frontier_i = seedUrls
    while(!frontier_i .isEmpty())
        new list frontier_i+1
        for url : frontier_i
                page = downloadPage(url)
                frontier_i+1.addAll(extractUrls(page))
                store(page)
        i++
```

What's missing from this code?

Crawling: Avoid Cycles



- Download the Web. ☺

```
crawl(list seedUrls)
    frontier_i = seedUrls
    new set urlsSeen
    while(!frontier_i .isEmpty())
        new list frontier_i+1
        for url : frontier_i
                page = downloadPage(url)
                urlsSeen.add(url)
                frontier_i+1.addAll(extractUrls(page).removeAll(urlsSeen))
                store(page)
        i++
```

What about the performance of this code?

Crawling: Catering for Slow Network

```
page = downloadPage(url)
```

```
C:\Users\Aidan>ping twitter.com

Pinging twitter.com [199.16.156.198] with 32 bytes of data:
Reply from 199.16.156.198: bytes=32 time=118ms TTL=50
Reply from 199.16.156.198: bytes=32 time=120ms TTL=50
Reply from 199.16.156.198: bytes=32 time=120ms TTL=50
Reply from 199.16.156.198: bytes=32 time=125ms TTL=50

Ping statistics for 199.16.156.198:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 118ms, Maximum = 125ms, Average = 120ms

C:\Users\Aidan>
```

- **Majority of the time spent will be spent waiting for connection**
- **Disk and CPU of crawling machine barely occupied**
- **Bandwidth will not be maximised (stop / start)**

Crawling: Multi-threading Important

```
crawl(list seedUrls)
    frontier_i = seedUrls
    new set urlsSeen
    while(!frontier_i .isEmpty())
        new list frontier_i+1
        new list threads
        for url : frontier_i
                thread = new DownloadPageThread.run(url,urlsSeen,fronter_i+1)
                threads.add(thread)
        threads.poll()
        i++
DownloadPageThread: run(url,urlsSeen,frontier_i+1)
    page = downloadPage(url)
    synchronised: urlsSeen.add(url)
    synchronised: frontier_i+1.addAll(extractUrls(page).removeAll(urlsSeen))
    synchronised: store(page)
```

Crawling: Multi-threading Important

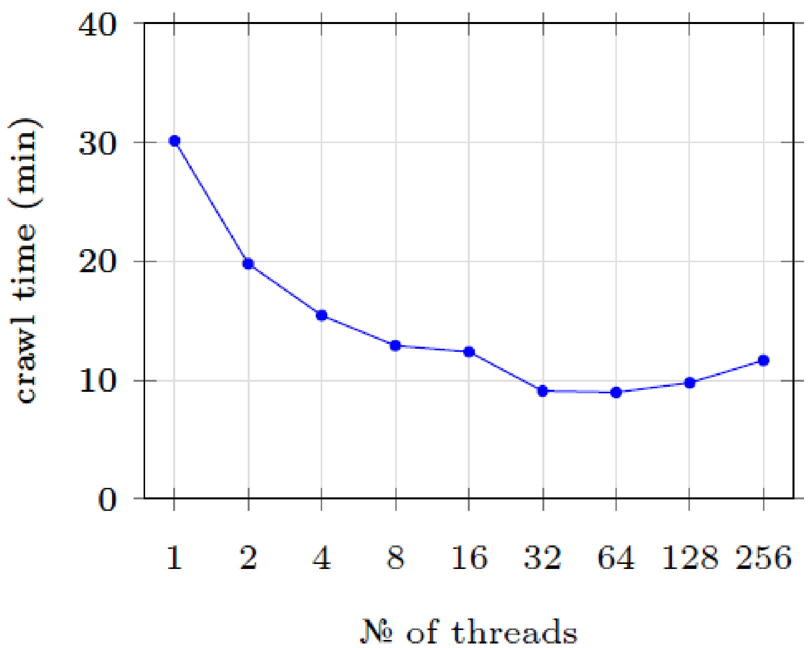


Figure 4.1: Total time taken to crawl 1,000 URIs with a varying number of threads

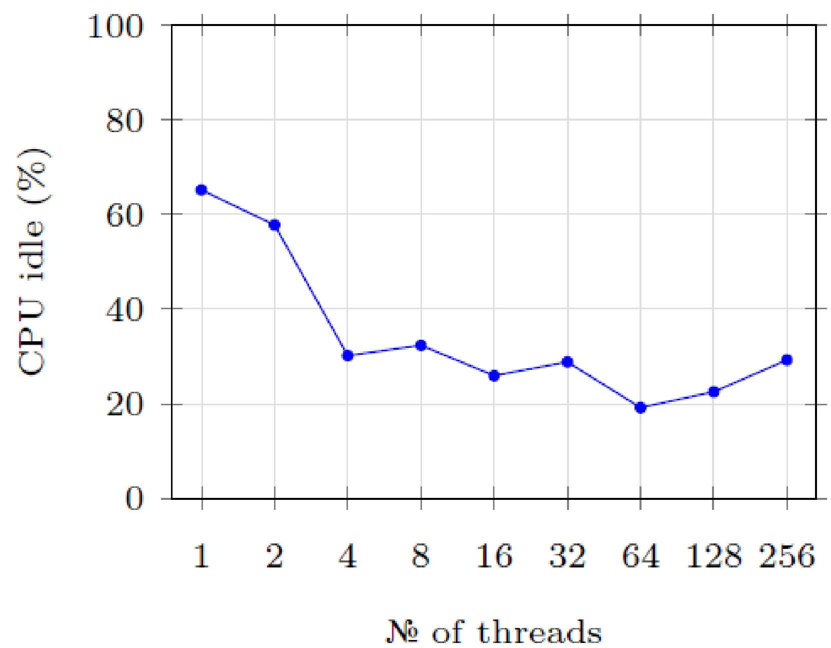
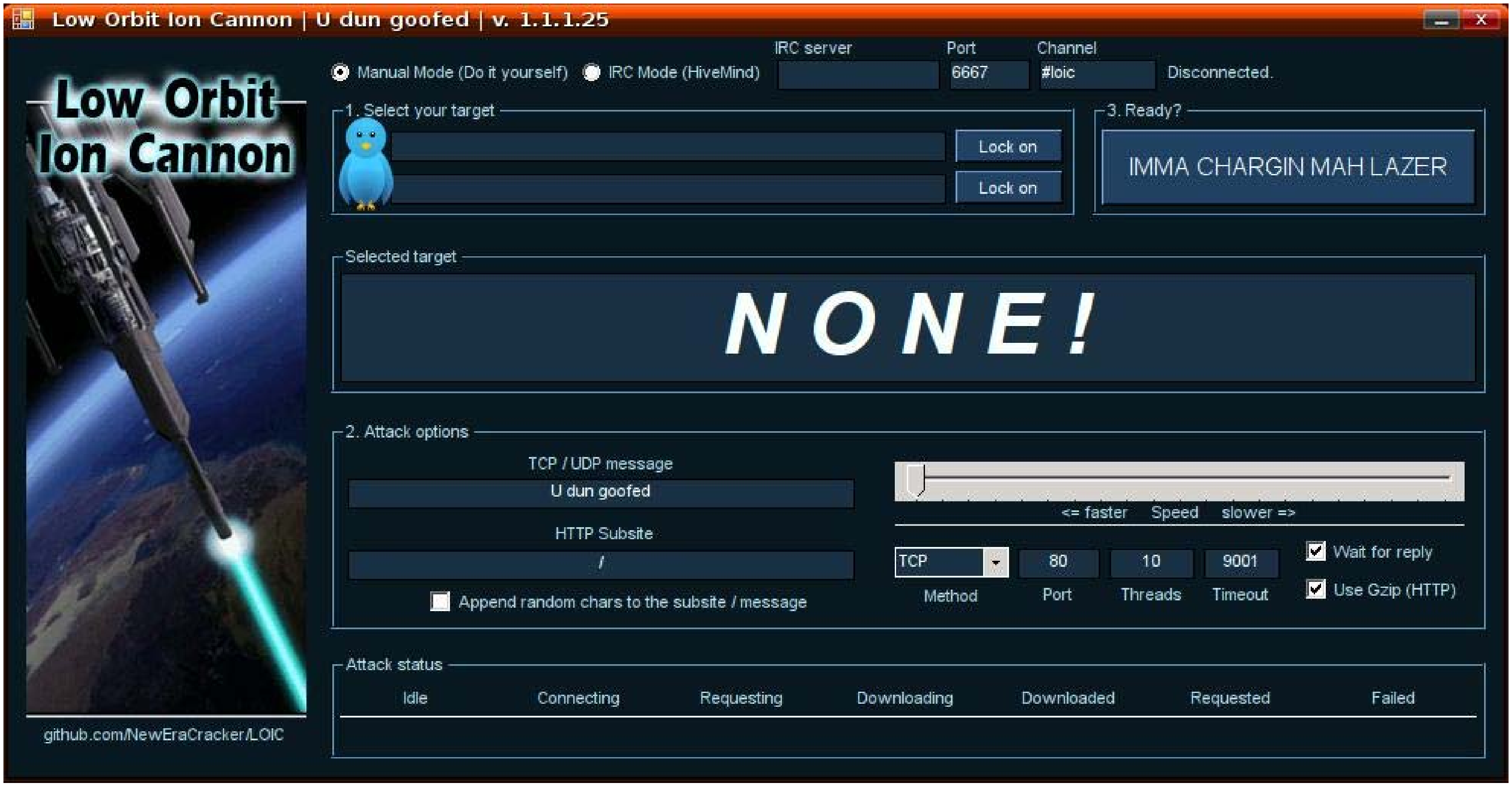


Figure 4.2: Average idle CPU % while crawling 1,000 URIs with a varying number of threads

Crawling: Important to be Polite!

- (Distributed) Denial of Server Attack: (D)DoS



Crawling: Avoid (D)DoSing

- But more likely your IP range will be banned by the web-site (DoS attack)

Crawling: Web-site Scheduler

```
crawl(list seedUrls)
    frontier_i = seedUrls
    new set urlsSeen
    while(!frontier_i .isEmpty())
        new list frontier_i+1
        new list threads
        for url : schedule(frontier_i) #maximise time between two pages on one site
                thread = new DownloadPageThread.run(url,urlsSeen,fronter_i+1)
                threads.add(thread)
        threads.poll()
        i++
DownloadPageThread: run(url,urlsSeen,frontier_i+1)
    page = downloadPage(url)
    synchronised: urlsSeen.add(url)
    synchronised: frontier_i+1.addAll(extractUrls(page) .removeAll(urlsSeen))
    synchronised: store(page)
```

Robots Exclusion Protocol

http://website.com/robots.txt

```
User-agent: *
Disallow: /
```

No bots allowed on the website.

```
User-agent: *
Disallow: /user/
Disallow: /main/login.html
```

No bots allowed in /user/ sub-folder or login page.

```
User-agent: googlebot
Disallow: /
```

Ban only the bot with “user-agent” googlebot.

Robots Exclusion Protocol (non-standard)

```
User-agent: googlebot
Crawl-delay: 10
```

Tell the googlebot to only crawl a page from this host no more than once every 10 seconds.

```
User-agent: *
Disallow: /
Allow: /public/
```

Ban everything but the /public/ folder for all agents

```
User-agent: *
Sitemap: http://example.com/main/sitemap.xml
```

Tell user-agents about your *site-map*

Site-Map

```xml
<?xml version="1.0" encoding="UTF-8"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
  <url>
    <loc>http://www.example.com/?id=who</loc>
    <lastmod>2009-09-22</lastmod>
    <changefreq>monthly</changefreq>
    <priority>0.8</priority>
  </url>
  <url>
    <loc>http://www.example.com/?id=what</loc>
    <lastmod>2009-09-22</lastmod>
    <changefreq>monthly</changefreq>
    <priority>0.5</priority>
  </url>
  <url>
    <loc>http://www.example.com/?id=how</loc>
    <lastmod>2009-09-22</lastmod>
    <changefreq>monthly</changefreq>
    <priority>0.5</priority>
  </url>
</urlset>
```

Crawling: Important Points

- Seed-list: Entry point for crawling
- Frontier: Extract links from current pages for next round
- Threading: Keep machines busy; mitigate waits for connection
- Seen-list: Avoid cycles
- Politeness: Don’t annoy web-sites
  - Set a *politeness delay* between crawling pages on the same web-site
  - Stick to what’s stated in the robots.txt file
  - Check for a site-map

Crawling: Distribution

- Similar benefits to multi-threading

How might we implement a distributed crawler?

```
for url : frontier_i-1
        map(url,count)
```

- Local frontier and seen-URL list!

What will be the bottleneck as machines increase?

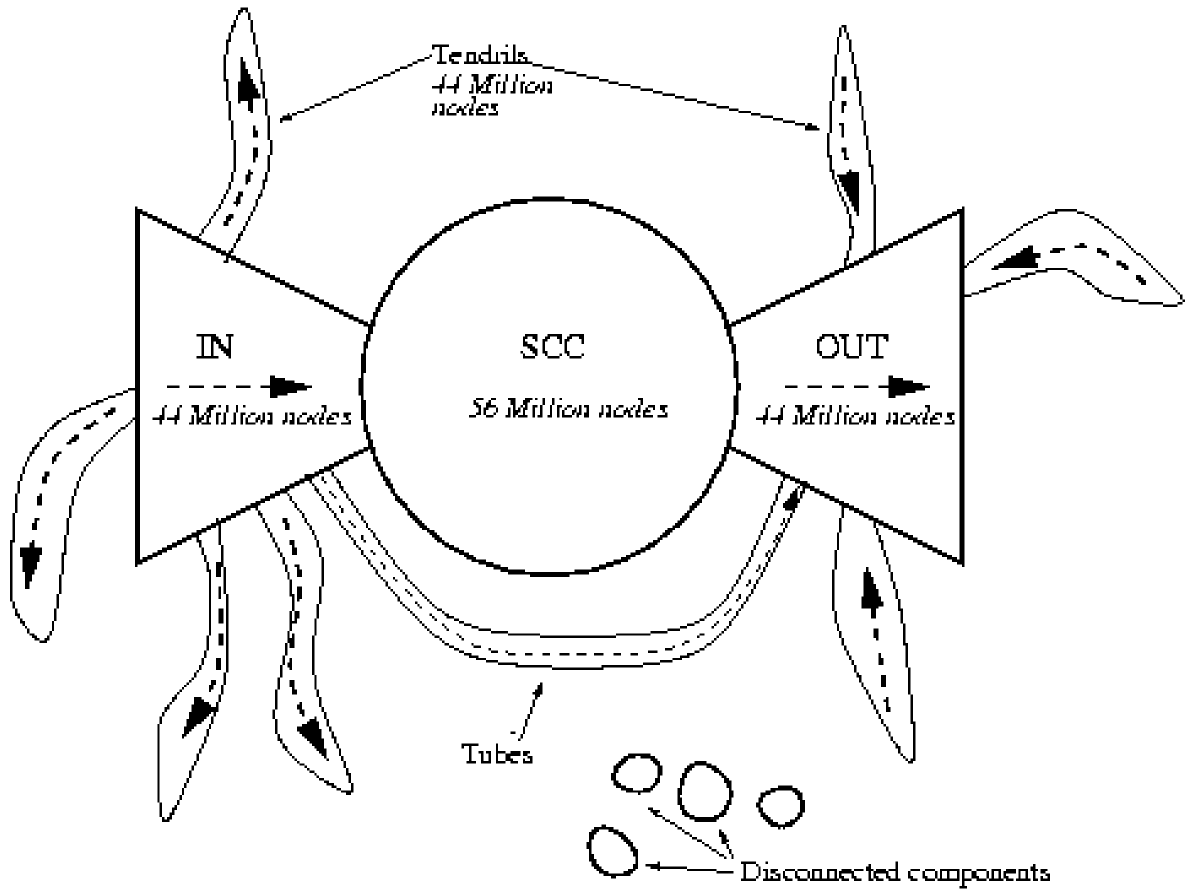
Crawling: Other Options

- Breadth-first: **As per the pseudo-code, crawl in rounds**
  - **Extract one-hop from seed URLs …**
  - **Extract n-hop from seed URLs**
- Depth-first: Follow first link in first page, first link in second page, etc.

Possible advantages of breadth vs. depth first?

- Best/topic-first: Rank the URLs according to topic, number of in-links, etc.
- Hybrid: A mix of strategies

Crawling: Inaccessible (Bow-Tie)



A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener, “Graph structure in the web,” Comput. Networks, vol. 33, no. 1-6, pp. 309–320, 2000

Crawling: Inaccessible (Deep-Web)

- Deep-web:
  - Dynamically-generated content
  - Password protected / firewalled
  - Dark Web

Remember: 46% of statistics are made up on the spot. ☺

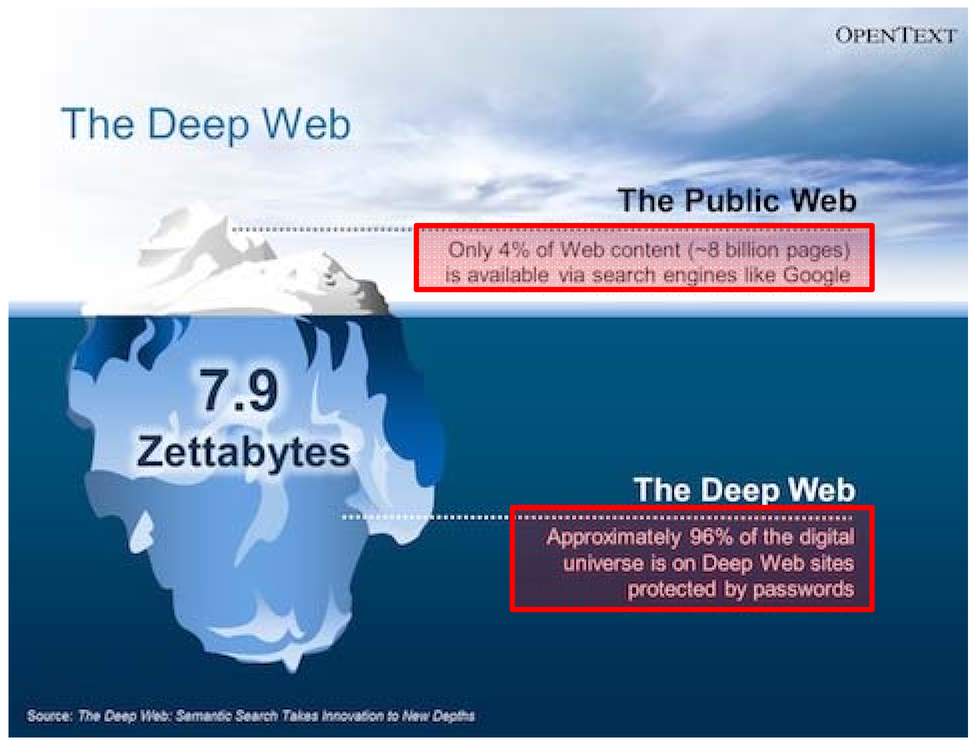


Image from http://www.legaltechnology.com/wp-content/uploads/2013/07/OpenText-EIM-Summary.pdf

Apache Nuche

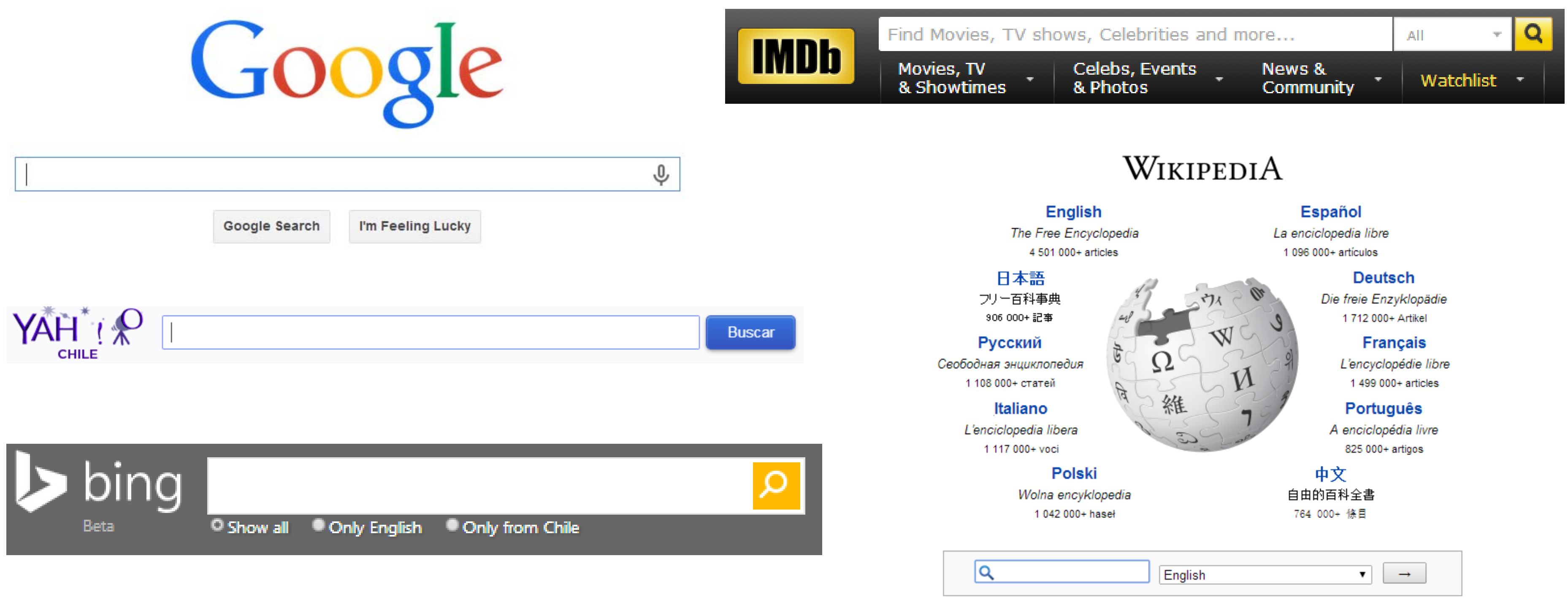
- Open-source crawling framework!
- Compatible with Hadoop!

https://nutch.apache.org/

INFORMATION RETRIEVAL: INVERTED-INDEXING

Inverted Index

- Inverted Index: A map from words to documents
  - “Inverted” because usually documents map to words
  - At the core of all keyword search applications



Inverted Index: Example

1

en.wikipedia.org/wiki/Fruitvale_Station

Fruitvale Station

From Wikipedia, the free encyclopedia

1 10 18 21 23 28 37 43 47 55 59 68 71 76

Fruitvale Station is a 2013 American drama film written and directed by Ryan Coogler.

Inverted index:

Term List	Posting Lists
a	(1,[21,96,103,…]), (2,[…]), …
american	(1,[28,123]), (5,[…]), …
and	(1,[57,139,…]), (2,[…]), …
by	(1,[70,157,…]), (2,[…]), …
directed	(1,[61,212,…]), (4,[…]), …
drama	(1,[38,87,…]), (16,[…]), …
…	…

Inverted Index: Example Search

american drama

- AND: Posting lists intersected (optimised!)
- OR: Posting lists unioned
- PHRASE: AND + check locations

Inverted index:

Word	Posting Lists
a	(1,[21,96,103,…]), (2,[…]), …
american	(1,[28,123]), (5,[…]), …
and	(1,[57,139,…]), (2,[…]), …
by	(1,[70,157,…]), (2,[…]), …
directed	(1,[61,212,…]), (4,[…]), …
drama	(1,[38,87,…]), (16,[…]), …
…	…

Inverted Index Flavours

- Record-level inverted index: Maps words to documents without positional information

- **Word-level inverted index: Additionally maps words with positional information**

Inverted Index: Word Normalisation

drama america

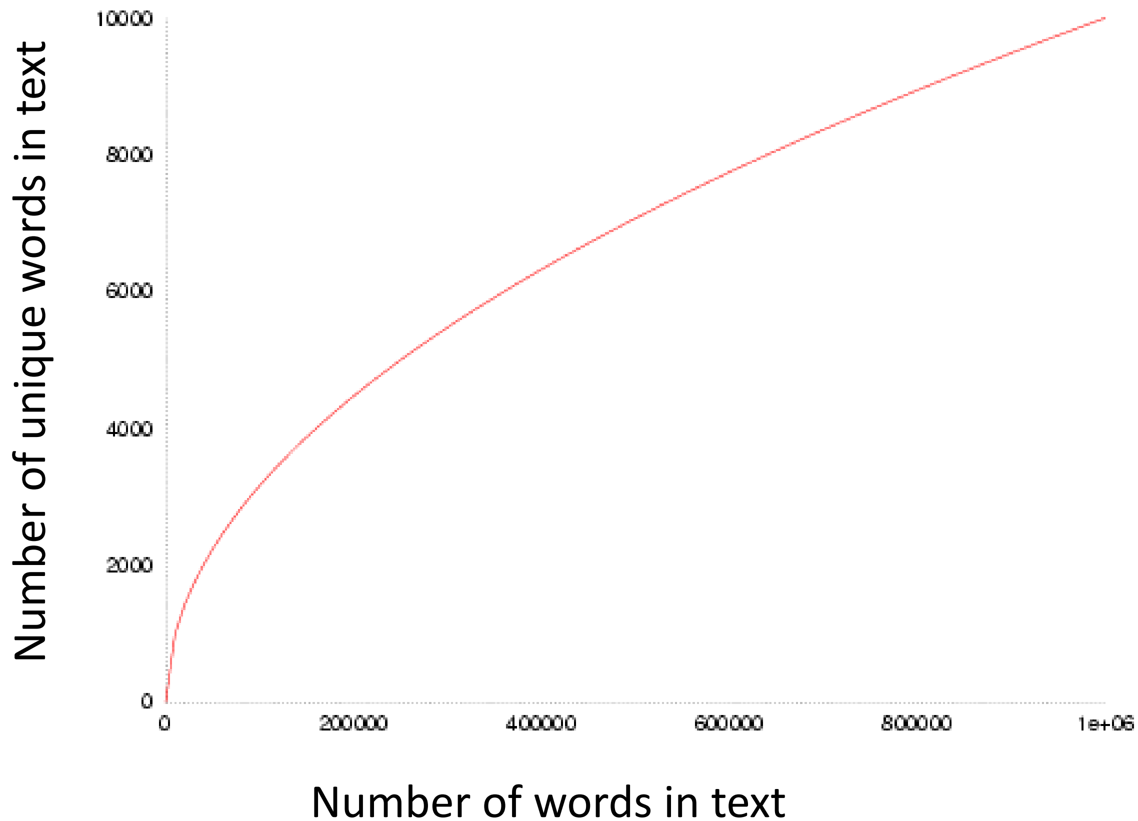
- Word normalisation: grammar removal, case, lemmatisation, accents, etc.
- Query side and/or index side

<u>Inverted index:</u>

Term List	Posting Lists
a	(1,[21,96,103,…]), (2,[…]), …
american	(1,[28,123]), (5,[…]), …
and	(1,[57,139,…]), (2,[…]), …
by	(1,[70,157,…]), (2,[…]), …
directed	(1,[61,212,…]), (4,[…]), …
drama	(1,[38,87,…]), (16,[…]), …
…	…

Inverted Index: Space

- Not so many unique words …
  - but lots of new proper nouns
  - Heap's law:
  - UW(n) ≈ Kn^{β}
  - English text
    - K ≈ 10
    - β ≈ 0.6

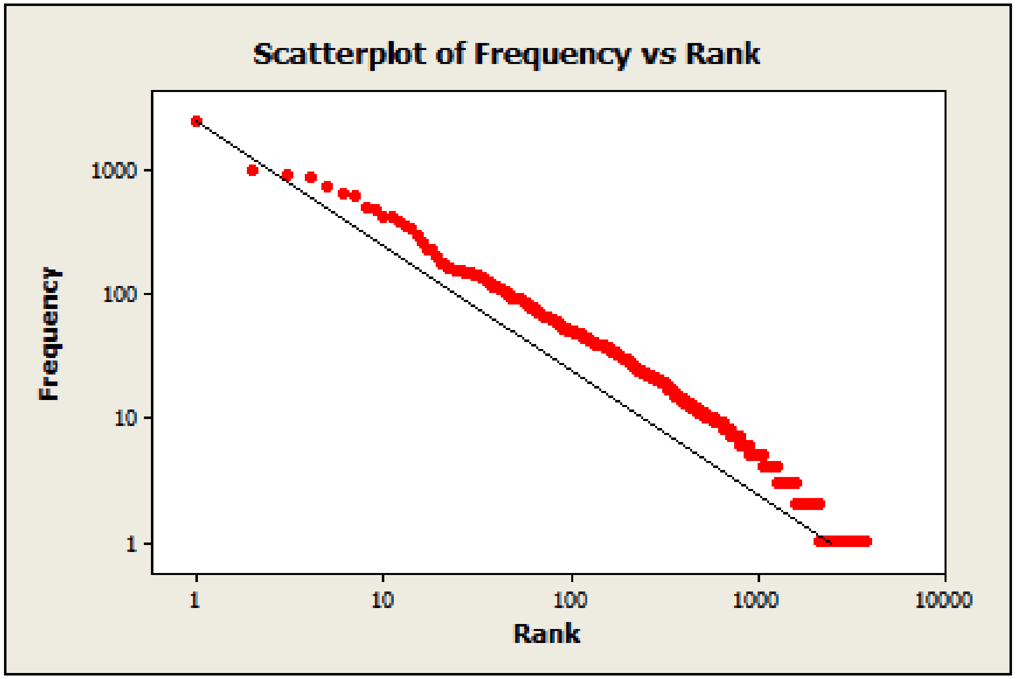


Inverted Index: Space

- As text size grows in a given document
  - (n words) …
  - Unique words (i.e., inverted index keys) grow sub-linearly: $O(n^{\beta})$ for $\beta \approx 0.6$
  - Positions of occurrence grow linearly $O(n)$

Inverted Index: Common Words

- Many occurrences of few words
  - Few occurrences of many words
  - Zipf's law
  - In English text:
    - "the" 7%
    - "of" 3.5%
    - "and" 2.7%
    - 135 words cover half of all occurrences



Zipf's law: ***the most popular word will occur twice as often as the second most popular word, thrice as often as the third most popular word, n times as often as the n-most popular word.***

Inverted Index: Common Words

- Expect long posting lists for common words
- Also expect more queries for common words

Inverted Index: Common Words

- Perhaps implement stop-words?
  - Most common words contain least information

the drama in america

- Perhaps implement block-addressing?

Fruitvale Station is a 2013 American drama film written and directed by Ryan Coogler.

Block 1 **Block 2**

Term List	Posting Lists
a	(1,[1,…]), (2,[…]), …
american	(1,[1,…]), (5,[…]), …
and	(1,[2, …]), (2,[…]), …
by	(1,[2, …]), (2,[…]), …
…	…

What is the effect on phrase search?

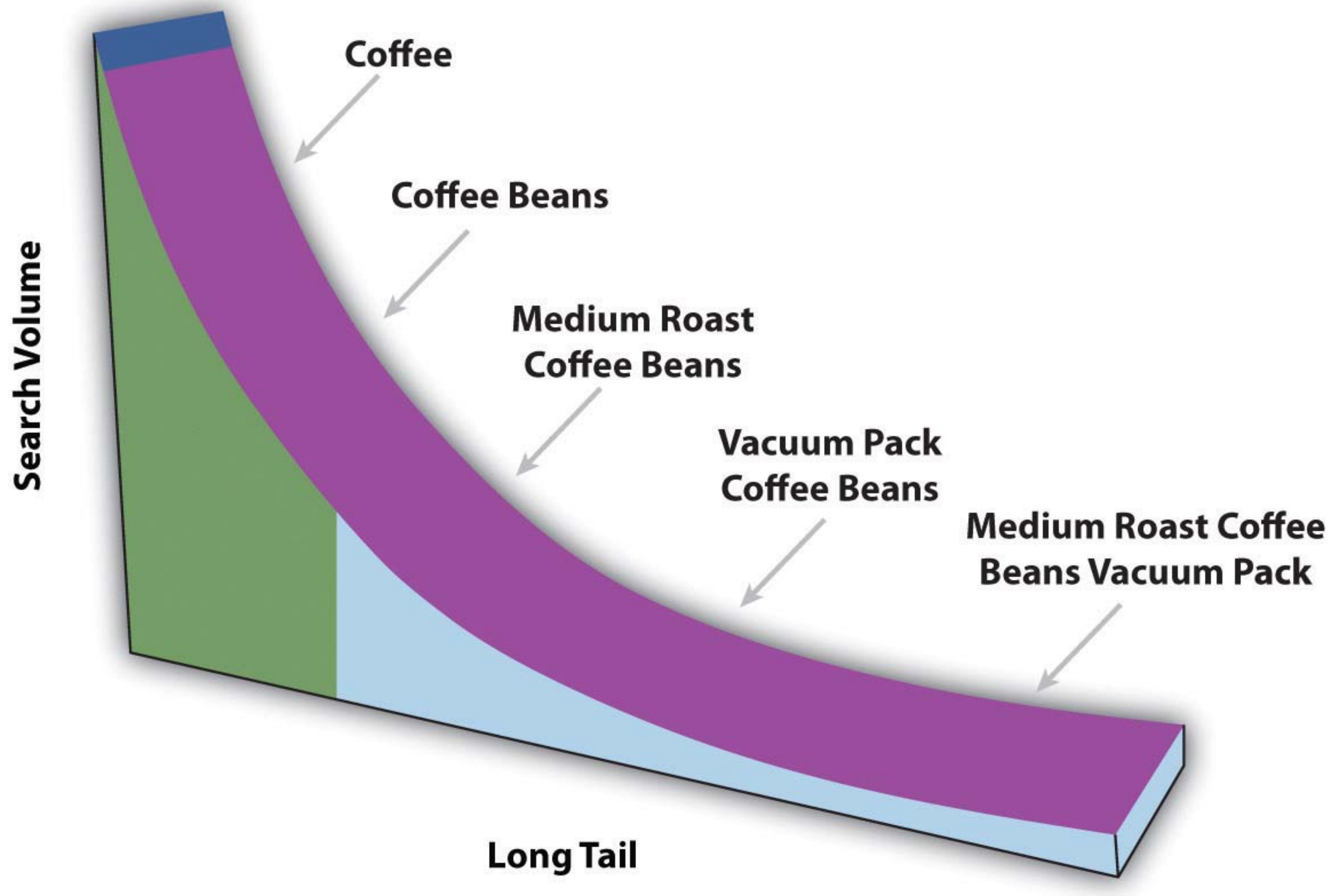
Search Implementation

- Vocabulary keys:
  - Hashing: O(1) lookups (assuming good hashing)
    - no range queries
    - relatively easy to update (though rehashing expensive!)
  - Sorting/B-Tree: O(log(u)) lookups, u unique words
    - range queries
    - tricky to update (standard methods for B-trees)
  - Tries: O(l) lookups, l length of the word
    - range queries, compressed, auto-completion!
    - referencing becomes tricky (esp. on disk)

Memory Sizes

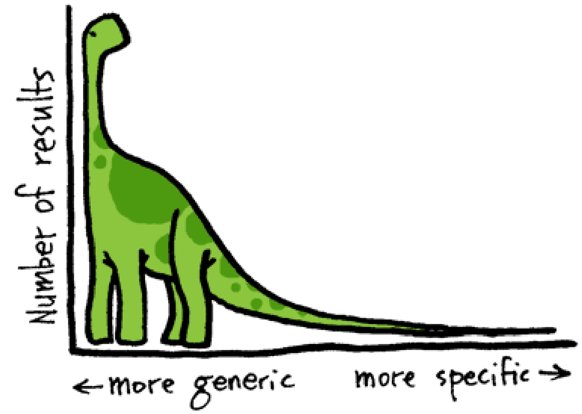
- Vocabulary keys:
  – Often will fit in memory!
  – Posting lists may be kept on disk
    - (hot regions cached)

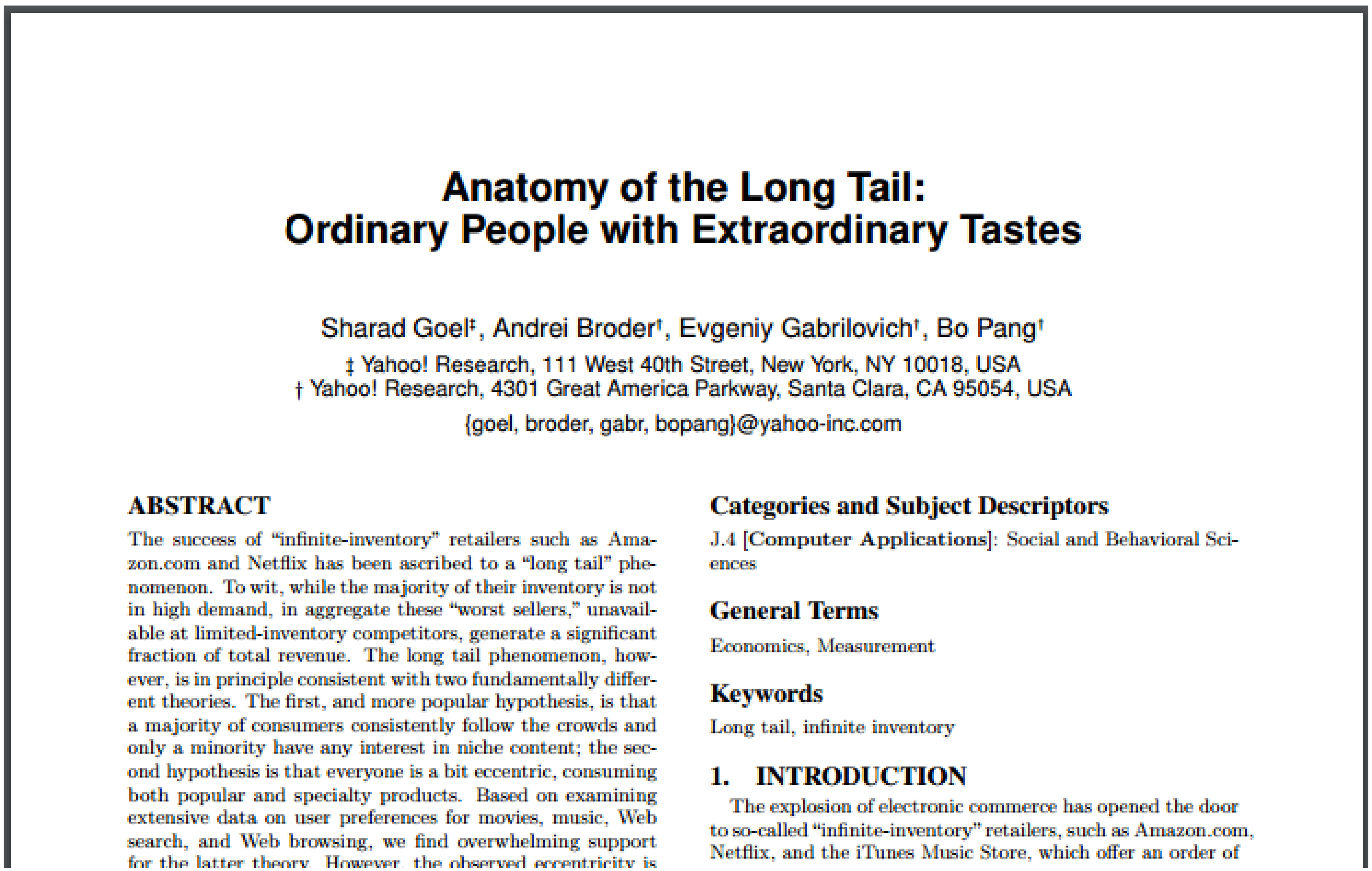
The Long Tail



What would this mean for a cache?

If interested in long tails …





Anatomy of the Long Tail: Ordinary People with Extraordinary Tastes

Sharad Goel[‡], Andrei Broder[†], Evgeniy Gabrilovich[†], Bo Pang[†]
‡ Yahoo! Research, 111 West 40th Street, New York, NY 10018, USA
† Yahoo! Research, 4301 Great America Parkway, Santa Clara, CA 95054, USA
{goel, broder, gabr, bopang}@yahoo-inc.com

ABSTRACT

The success of "infinite-inventory" retailers such as Amazon.com and Netflix has been ascribed to a "long tail" phenomenon. To wit, while the majority of their inventory is not in high demand, in aggregate these "worst sellers," unavailable at limited-inventory competitors, generate a significant fraction of total revenue. The long tail phenomenon, however, is in principle consistent with two fundamentally different theories. The first, and more popular hypothesis, is that a majority of consumers consistently follow the crowds and only a minority have any interest in niche content; the second hypothesis is that everyone is a bit eccentric, consuming both popular and specialty products. Based on examining extensive data on user preferences for movies, music, Web search, and Web browsing, we find overwhelming support for the latter theory. However, the observed eccentricity is

Categories and Subject Descriptors

J.4 [**Computer Applications**]: Social and Behavioral Sciences

General Terms

Economics, Measurement

Keywords

Long tail, infinite inventory

1. INTRODUCTION

The explosion of electronic commerce has opened the door to so-called "infinite-inventory" retailers, such as Amazon.com, Netflix, and the iTunes Music Store, which offer an order of

Compression techniques

- Numeric compression important

Term List	Posting Lists
country	(1), (2), (3), (4), (6), (7), …
…	…

Compression techniques: High Level

- Interval indexing
  - Example for record-level indexing
    - Could also be applied for block-level indexing, etc.

Term List	Posting Lists
country	(1), (2), (3), (4), (6), (7), …
…	…

Term List	Posting Lists
country	(1–4), (6–7),
…	…

Compression techniques: High Level

- Gap indexing
  - Example for record-level indexing
    - Could also be applied for block-level indexing, etc.

Term List	Posting Lists
country	(1), (3), (4), (8), (9), …
…	…

Term List	Posting Lists
country	(1), 2, 1, 4, 1
…	…

What's the benefit?	Repeated small numbers easier to compress

Compression techniques: Bit Level

- Variable length coding: bit-level techniques
- For example, Elias γ (gamma) encoding
  - Assumes many small numbers

z: integer to encode	n = $\lfloor \log_2(z) \rfloor$ coded in unary	a zero marker	next n binary numbers	final Elias γ code
1	0			0
2	1	0	0	100
3	1	0	1	101
4	11	0	00	11000
5	11	0	01	11001
6	11	0	10	11010
7	11	0	11	11011
8	111	0	000	1110000
…	…	…	…	…

Can you decode "0100001100011100001100 1"? <1,2,1,1,4,8,5>

Compression techniques: Bit Level

- Variable length coding: bit-level techniques
- For example, Elias δ (delta) encoding
  - Better for some distributions

z: integer to encode	$n = \lfloor \log_2(z) \rfloor$ coded in **Elias γ**	next *n* binary numbers	final Elias δ code
1	*0*		0
2	100	0	1000
3	100	1	1001
4	101	00	10100
5	101	01	10101
6	101	10	10110
7	101	11	10111
8	11000	000	11000000
…	…	…	…

Can you decode "0110000011001011001001"?	<1,9,3,1,17>

Compression techniques: Byte Level

- Use variable length byte codes
- Use last bit of byte to indicate if the number ends

- For example:

00100100	10100010	00000101	00100100

- 0010010 = 18, 1010001= 81, 100010010= 274

Parametric compression

- Previous methods "non-parametric"
  - Don't take an input value
- Other compression techniques parametric:
  - for example, Golomb-*3* code:

z: integer to encode	$n = \lfloor(z\text{-}1)/3\rfloor$ coded in unary	binary remainder	final Golomb-3 code
1	*0*	0	00
2	0	10	010
3	0	11	011
4	1	0	100
5	1	10	1010
6	1	11	1011
7	11	00	1100
8	11	010	11010

Other Optimisations

- Top-Doc: Order posting lists to give likely "top documents" first: good for top-*k* results

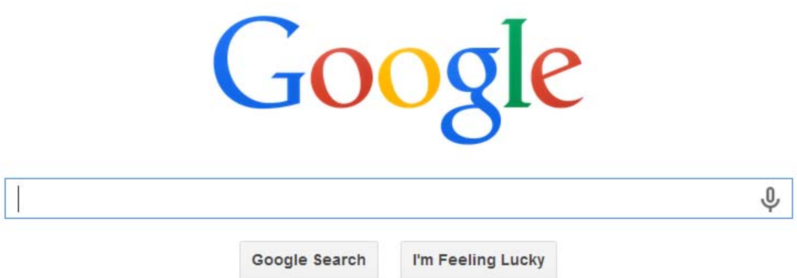
- Selectivity: Load the posting lists for the most rare keywords first; apply thresholds

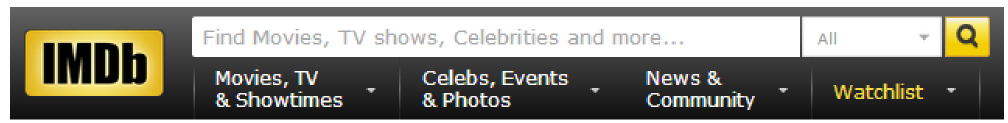
- Sharding: Distribute an index over multiple machines

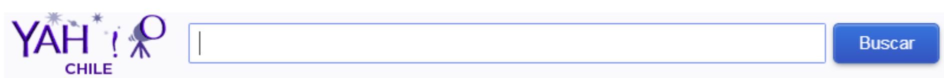
How might an inverted index be split over multiple machines?

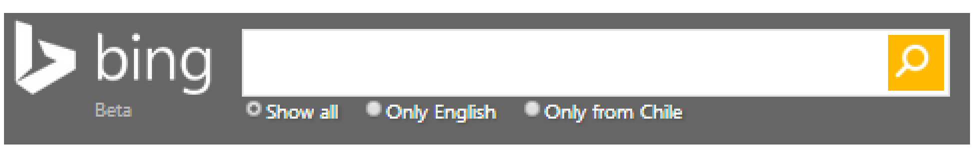
Extremely Scalable/Efficient

- When engineered correctly ☺





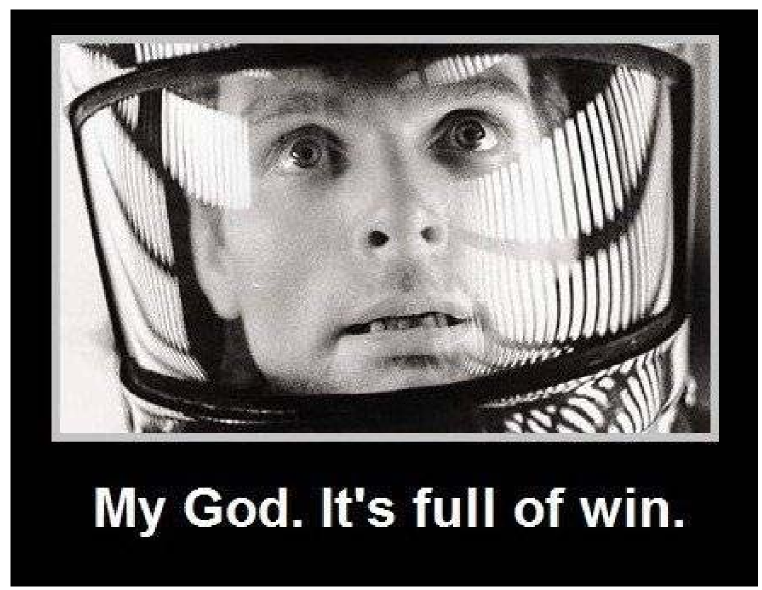




AN INVERTED INDEX SOLUTION

Apache Lucene

- Inverted Index
  - They built one so you don't have to!
- Open Source in Java



(Apache Solr)

- Built on top of Apache Lucene
- Lucene is the inverted index
- Solr is a distributed search platform, with distribution, fault tolerance, etc.
- (*We will work with Lucene*)

Apache Lucene: Indexing Documents

```
/**
 *
 * @param webData Tuples representing Web documents
 *                    with <url, title, text>
 * @param indexDir Directory on disk
 * @throws IOException
 */
public static void indexWeb(Iterator<String[]> webData, File indexDir) throws IOException{
    // open a directory on-disk for the inverted index
    Directory dir = FSDirectory.open(indexDir);
    // an analyser extracts terms (individual words)
    // from text ... analysers exist for different languages
    Analyzer analyzer = new StandardAnalyzer(Version.LUCENE_48);

    // this configures how the index will be written
    IndexWriterConfig iwc = new IndexWriterConfig(Version.LUCENE_48, analyzer);
    // we want to create an index so we pass CREATE
    iwc.setOpenMode(OpenMode.CREATE);

    // open a new index writer with given config and dir
    IndexWriter writer = new IndexWriter(dir, iwc);

    while(webData.hasNext()){
        String[] urlTitleText = webData.next();

        // a document represents the thing indexed
        // or a "result"
        Document d = new Document();
```

… continued …

Apache Lucene: Indexing Documents

… continued …

```
            // a document represents the thing indexed
            // or a "result"
            Document d = new Document();

            // StringField: stored as a normal String that's not tokenized
            // Field.Store.YES means it can be retrieved later
            Field url = new StringField("url", urlTitleText[0], Field.Store.YES);
            d.add(url);

            // TextField: will be tokenized and indexed by analyser
            Field title = new TextField("title", urlTitleText[1], Field.Store.YES);
            d.add(title);

            // same as above but this time the entire text cannot
            // be retrieved from the result
            Field text = new TextField("text", urlTitleText[2], Field.Store.NO);
            d.add(text);

            // can search by the time it was indexed but cannot retreive
            // time from the result
            Field modified = new LongField("modified", System.currentTimeMillis(), Field.Store.NO);
            d.add(modified);

            // write the document to the index
            writer.addDocument(d);
        }

        // close the writer
        writer.close();
    }
}
```

Apache Lucene: Searching Documents

```java
/**
 *
 * @param indexDir : the location of the index directory
 * @param keywordQuery : the keyword query to run
 * @throws IOException
 * @throws org.apache.lucene.queryparser.classic.ParseException
 */
public static ArrayList<String[]> runSearch(File indexDir, String keywordQuery) throws IOException,
                                                    org.apache.lucene.queryparser.classic.ParseException {
    // open a reader for the directory
    IndexReader reader = DirectoryReader.open(FSDirectory.open(indexDir));
    // open a searcher over the reader
    IndexSearcher searcher = new IndexSearcher(reader);
    // use the same analyser as the build
    Analyzer analyzer = new StandardAnalyzer(Version.LUCENE_48);

    // these boosts decide the relative importance of the
    // fields for the search ranking
    HashMap<String,Float> boosts = new HashMap<String,Float>();
    boosts.put("text", 1f); //<- default
    boosts.put("title", 5f);  //<- 5 times more important than text

    // this accepts queries/searches and parses them into
    // searches over the index
    MultiFieldQueryParser queryParser = new MultiFieldQueryParser(
            Version.LUCENE_48,
            new String[] {"title", "text"},
            analyzer, boosts);

    // parse the keyword query string into a query object
    Query query = queryParser.parse(keywordQuery);
```

Apache Lucene: Searching Documents

```
    // 10 is the top-k being looked for
    TopDocs results = searcher.search(query, 10);
    // get the documents (results) and their scores, they will be ordered by score
    ScoreDoc[] hits = results.scoreDocs;

    // total number of matching results
    System.out.println("Matching documents: "+results.totalHits);

    // to store results
    ArrayList<String[]> urlTitle = new ArrayList<String[]>();
    for(int i=0; i<hits.length; i++) {
        // get hit number i
        Document doc = searcher.doc(hits[i].doc);
        String title = doc.get("title");
        String url = doc.get("url");
        urlTitle.add(new String[]{title,url});
    }

    return urlTitle;
}
```

RECAP

Recap

- Information overload in Big Data

- Search: user intent
- Query: user expression of search
- Retrieval: machine methods to execute search

Recap

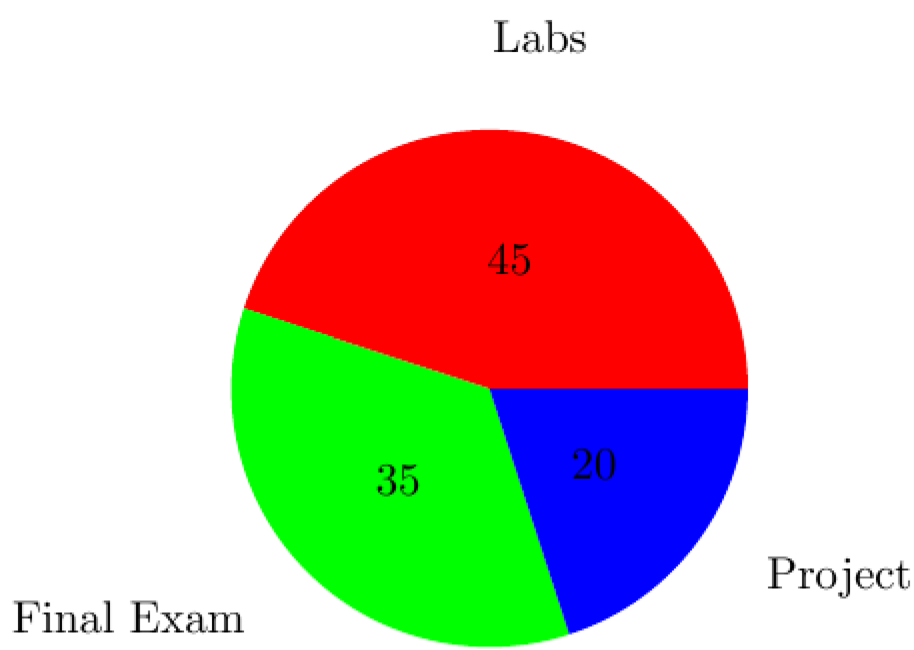
- Crawling:
  - Avoid cycles, multi-threading, politeness, ~~DDoS~~, robots exclusion, sitemaps, distribution, breadth-first, topical crawlers, deep web

- Inverted Indexing:
  - boolean queries, record-level vs. word-level vs. block-level, word normalisation, lemmatisation, space, Heap’s law, Zipf’s law, stop-words, tries, hashing, long tail, compression, interval coding, variable length encoding, Elias encoding, top doc, sharding, selectivity

CLASS PROJECTS

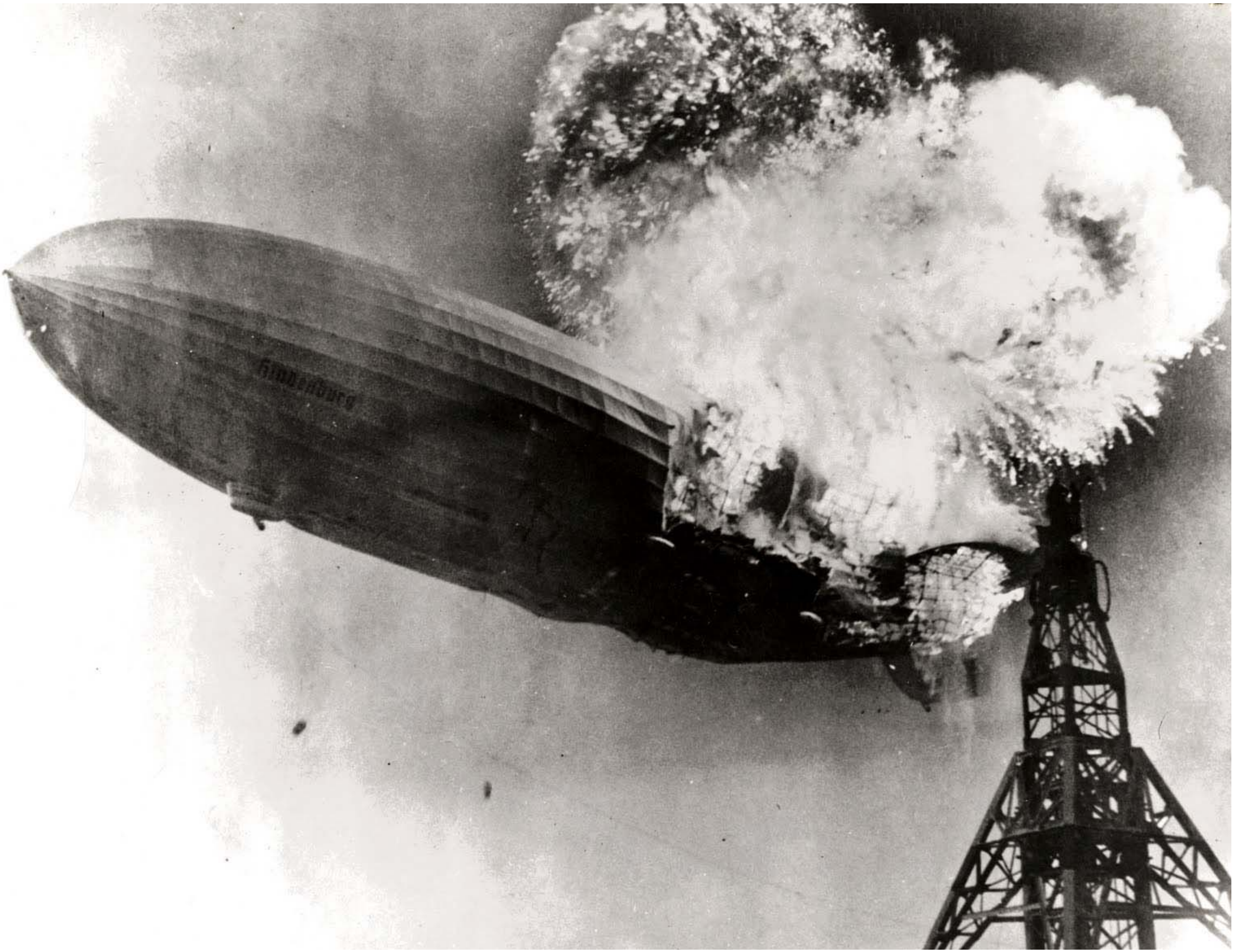
Course Marking

- 45% for Weekly Labs (~3% a lab!)
- 35% for Final Exam
- 20% for Small Class Project



Class Project

- Done in pairs (typically …)
- Goal: Use what you've learned to do something cool/fun (hopefully)
- Expected difficulty: A bit more than a lab's worth
  - But without guidance (can extend lab code)
- Marked on: Difficulty, appropriateness, scale, good use of techniques, presentation, coolness, creativity, value
  - Ambition is appreciated, even if you don't succeed: **feel free to bite off more than you can chew! I will take this into account.**
- Process:
  - Pair up (default random) by next Wednesday's lab (May 4^{th})
  - Start thinking up topics
  - If you need data or get stuck, I will (try to) help out
- Deliverables: 5/7 minute presentation & short report



Questions

?