CC5212-1

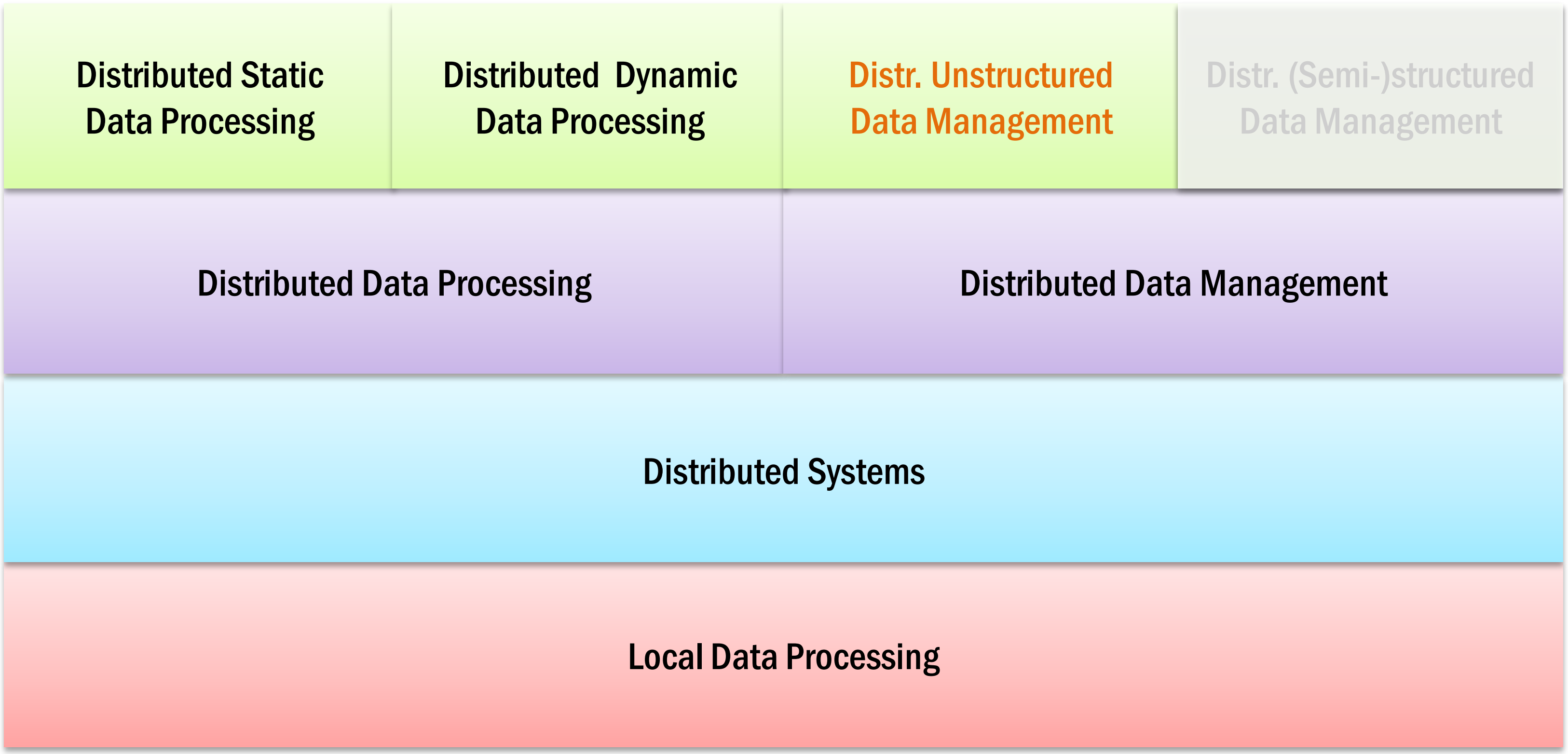
Procesamiento Masivo de Datos

Otoño 2023

Lecture 7

Information Retrieval: Crawling & Indexing

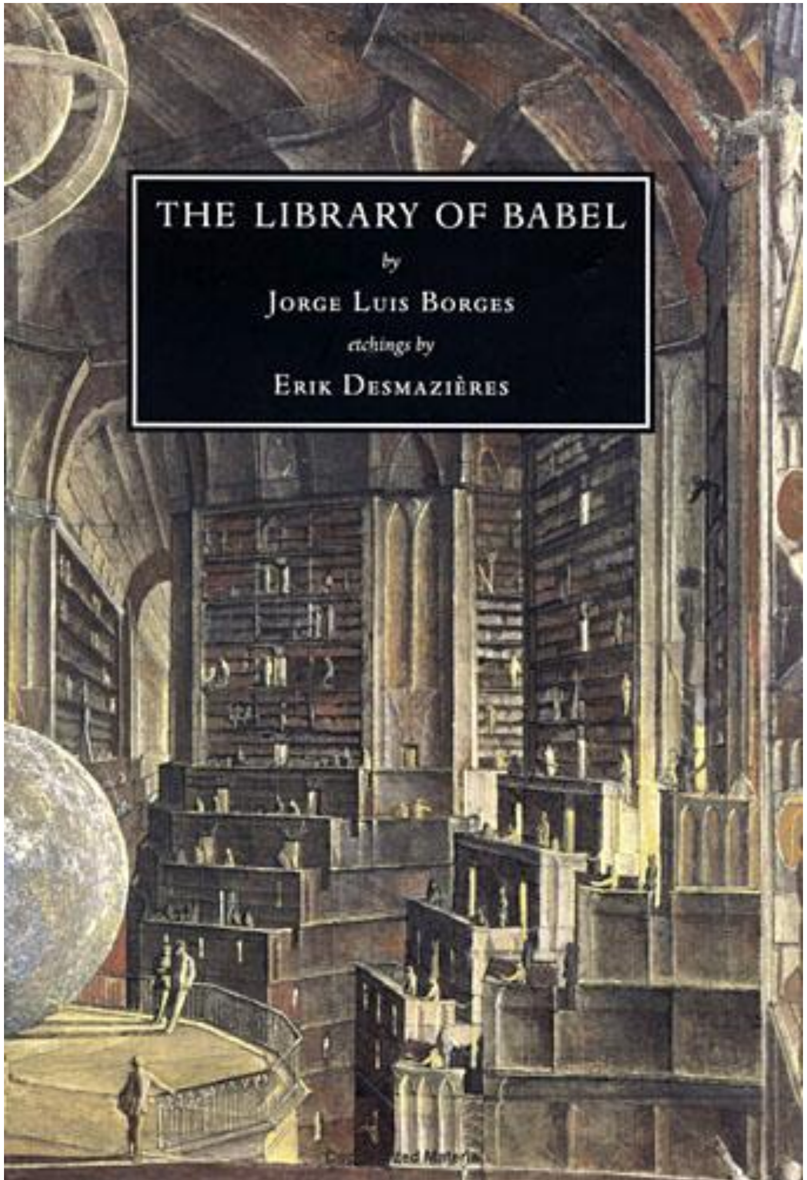
Aidan Hogan
aidhog@gmail.com


Distributed Static Data Processing
Distributed Dynamic Data Processing
Distr. Unstructured Data Management
Distr. (Semi-)structured Data Management
Distributed Data Processing
Distributed Data Management
Distributed Systems
Local Data Processing

MANAGING TEXT DATA

Information Overload

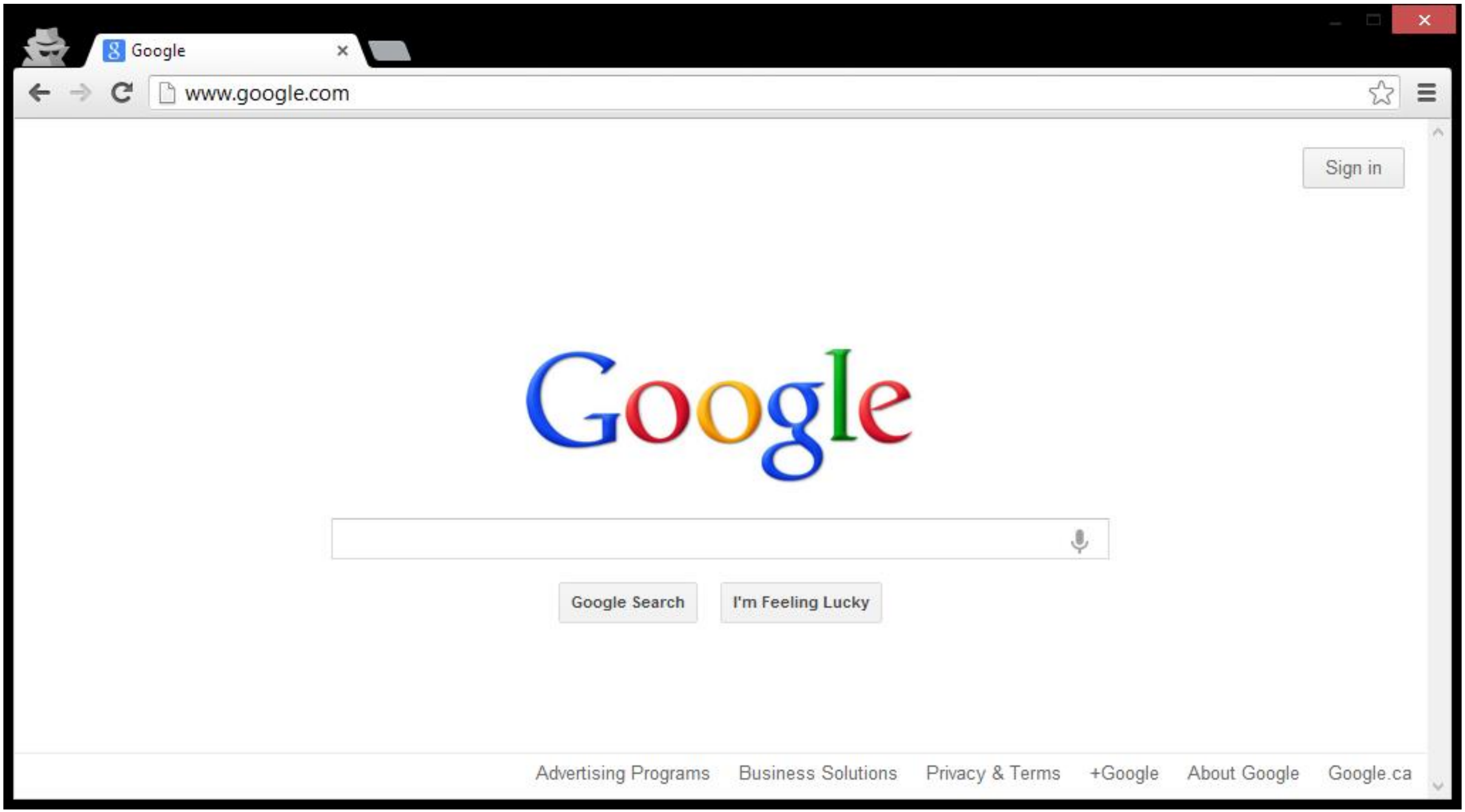
If we didn't have search …



- 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

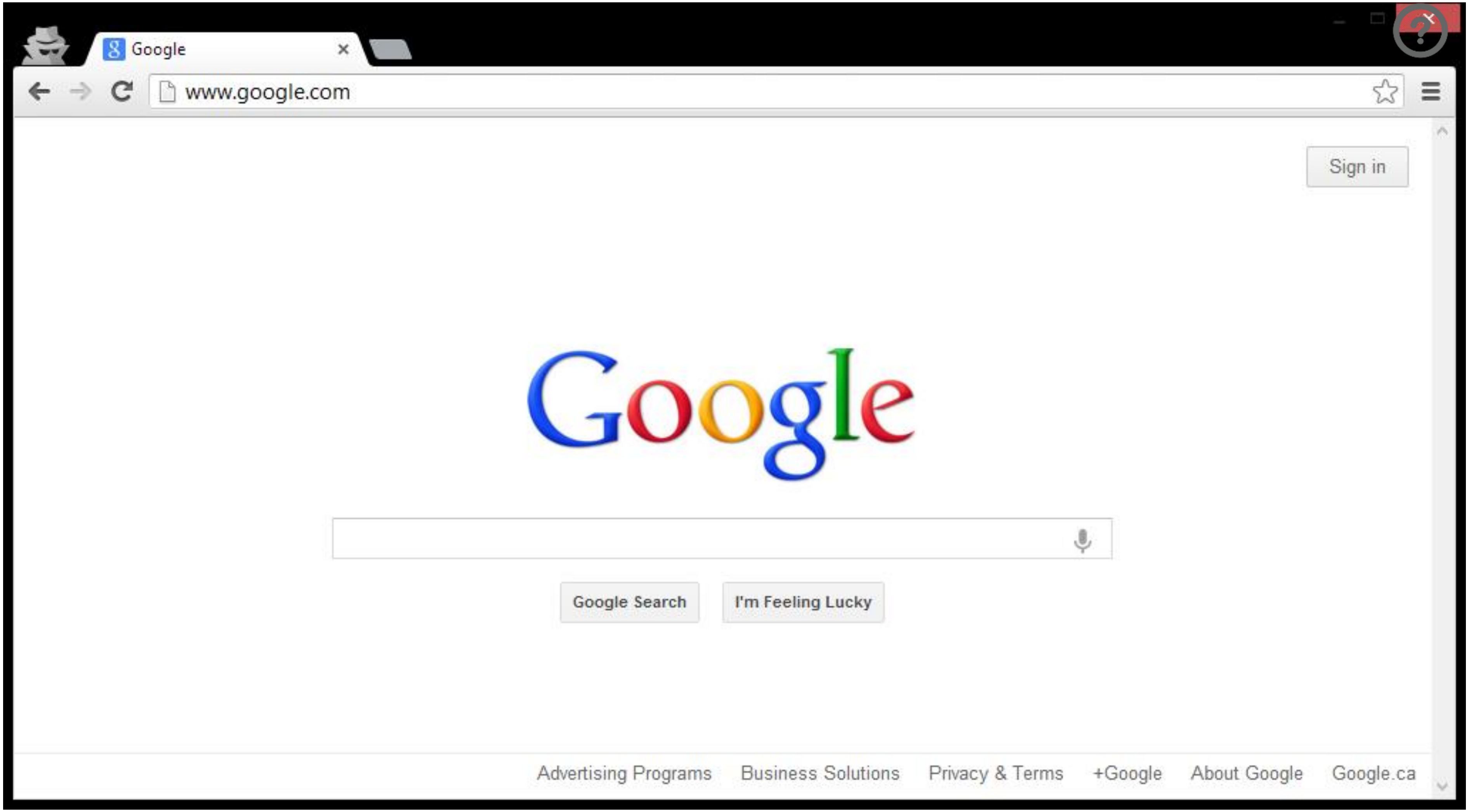
All information = Zero information

The book that indexes the library

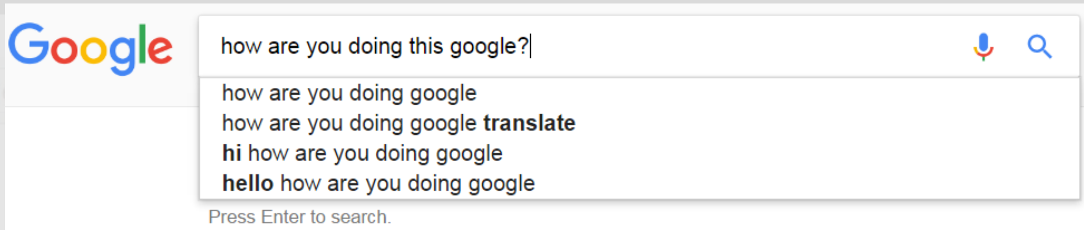


Web Search/Retrieval

Building Google Web-search



Building Google Web-search



What processes/algorithms does Google need to implement Web search?

Crawling

1. Parse links from webpages
2. Schedule links for crawling
3. Download pages, GOTO 1

Indexing

1. Parse keywords from webpages
2. Index keywords to webpages
3. Manage updates

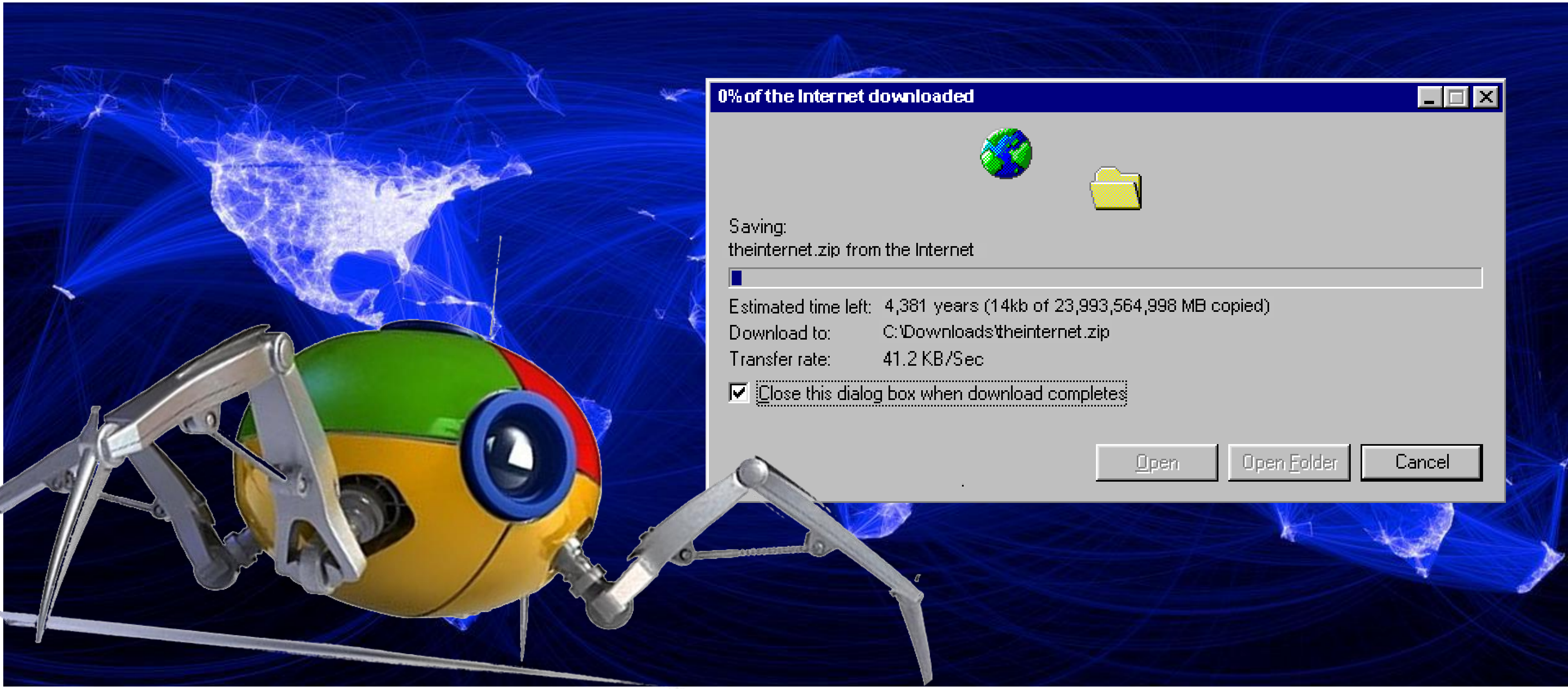
Ranking

1. How relevant is a page? (TF-IDF)
2. How important is it? (PageRank)
3. How many users clicked it?

...

Information Retrieval: Crawling

How does Google know about the Web?



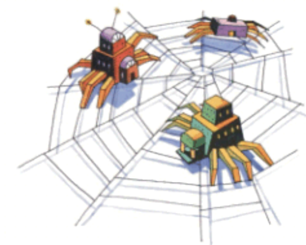
Crawling

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? ?

Crawling: Avoid Cycles

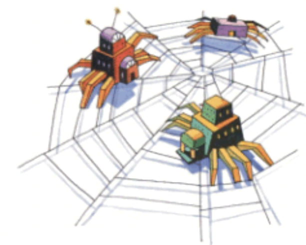


Download the Web. ☺

```
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++
```

Performance? ?

Crawling: Performance



Download the Web. ☺

```
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>
```

page = downloadPage(url)

- Majority of time spent waiting for connection
- Disk/CPU usage will be near 0
- Bandwidth will not be maximised

Performance?

Crawling: Performance

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++
```

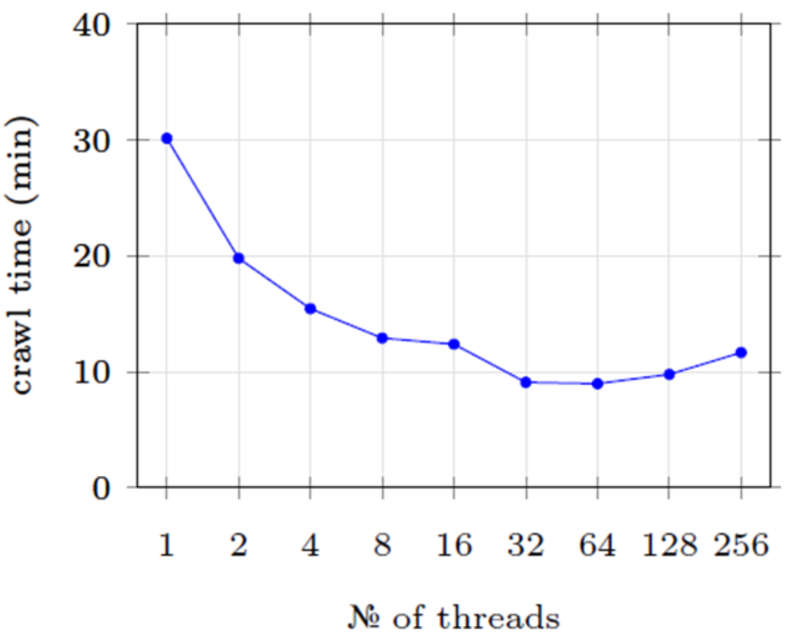
Solution?

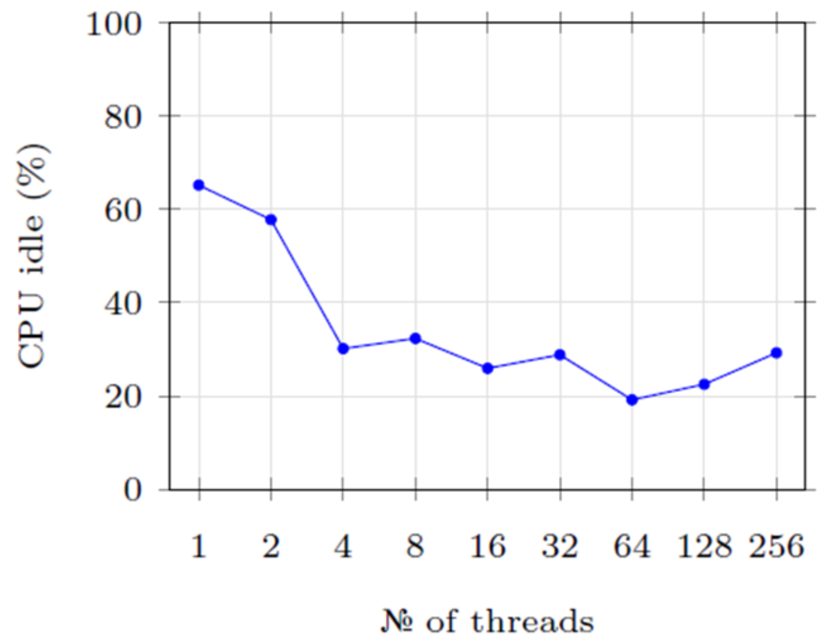
Crawling: Multi-threading Important

```
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 DownloadThread.run(url,urlsSeen,frontier_i+1)
                threads.add(thread)
        threads.poll()
        i++
  DownloadThread: 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

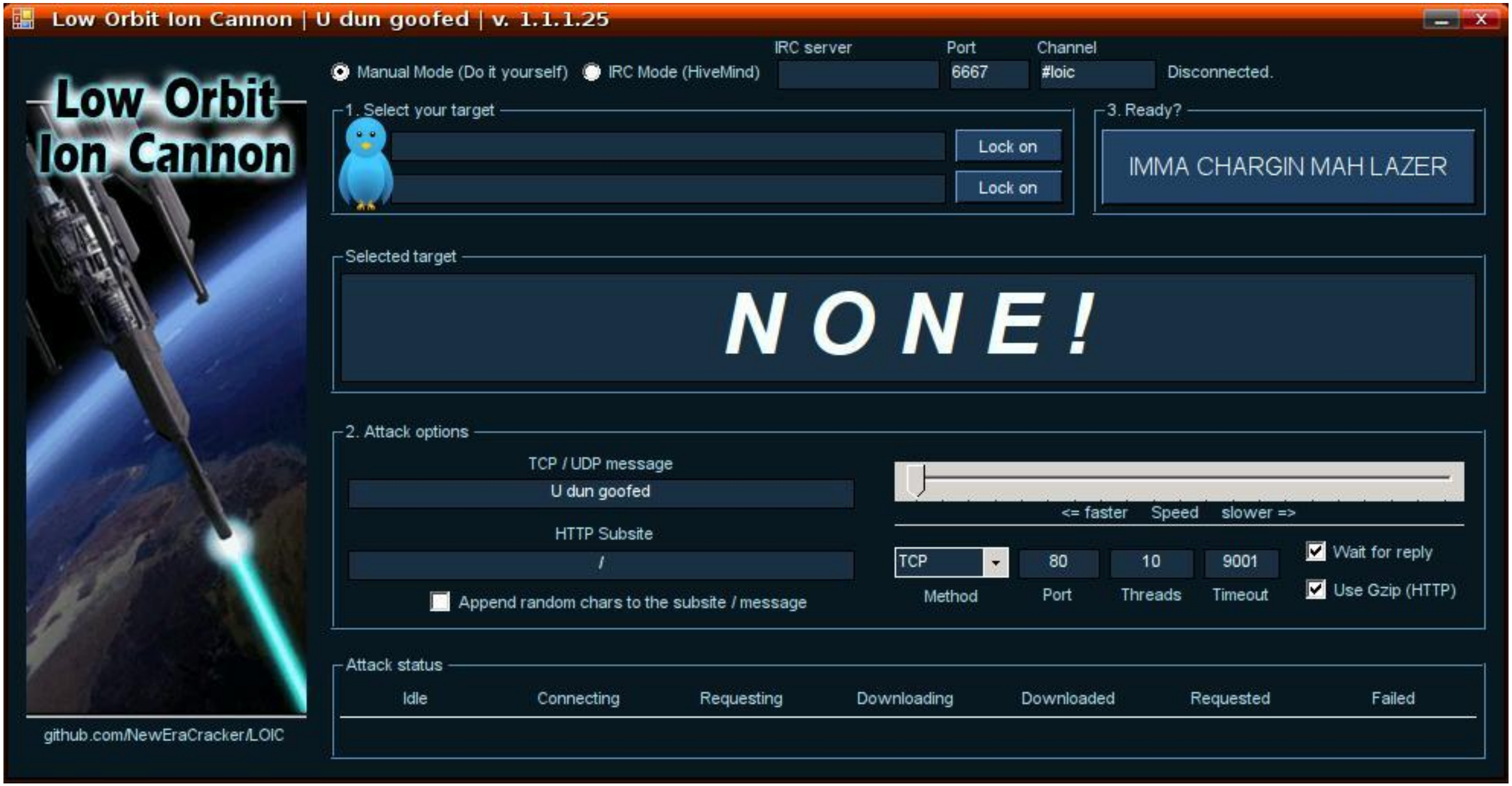
Crawl 1,000 URLs ...





Crawling: Important to be Polite!

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



Crawling: Avoid (D)DoSing

- Christopher Weatherhead
- 18 months prison

... more likely your IP range will be banned

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 gap between requests to one site
                thread = new DownloadThread.run(url,urlsSeen,frontier_i+1)
                threads.add(thread)
        threads.poll()
        i++
  DownloadThread: 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

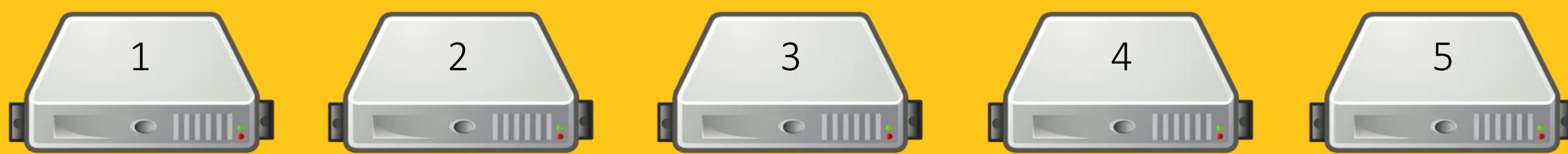
```
User-agent: *
Sitemap: http://example.com/main/sitemap.xml
```

Tell user-agents about your *site-map*

Crawling: Distribution

How might we implement a distributed crawler?

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



Similar benefits to multi-threading

What will be the bottleneck as machines increase?

Bandwidth or politeness delays

Apache Nutch

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

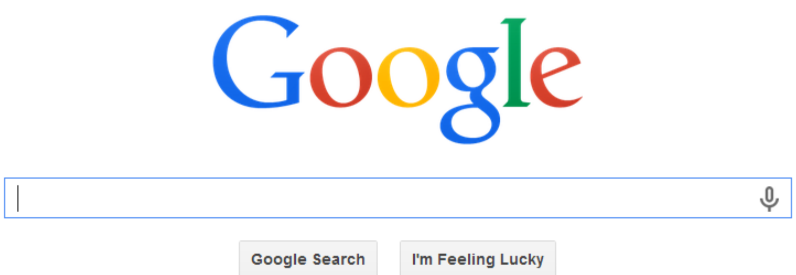
https://nutch.apache.org/

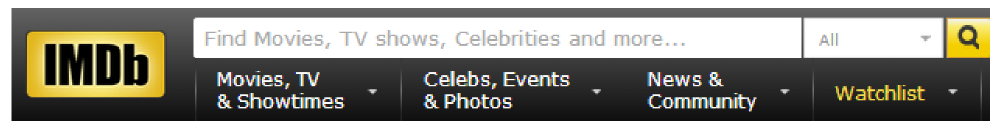
Information Retrieval: Inverted Indexing

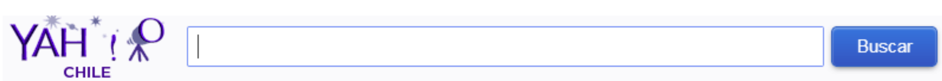
Inverted Index

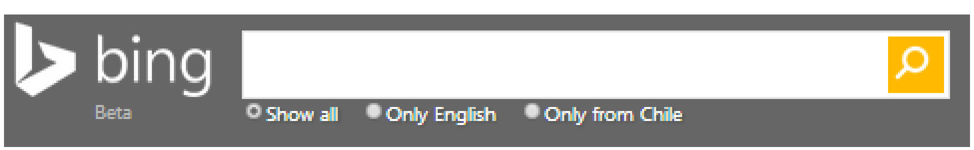
- Inverted Index: A map from words to documents
  - “Inverted” because usually documents map to words

Examples of applications?









Inverted Index: Example

1

en.wikipedia.org/wiki/Fruitvale_Station

Fruitvale Station

From Wikipedia, the free encyclopedia

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

<u>Inverted index:</u>

Term List	Posting List
a	(1,2,…)
american	(1,5,…)
and	(1,2,…)
by	(1,2,…)
directed	(1,2,…)
drama	(1,16,…)
…	…

Inverted Index: Example Search

american drama

- AND: Intersect posting lists
- OR: Union posting lists
- PHRASE: ???

How should we implement PHRASE?

Inverted index:

Term List	Posting List
a	(1,2,…)
american	(1,5,…)
and	(1,2,…)
by	(1,2,…)
directed	(1,2,…)
drama	(1,16,…)
…	…

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 48 56 60 69 72 77

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

Inverted index:

Term List	Posting List
a	(1,[21,96,103,…]), (2,[…]), …
american	(1,[28,123]), (5,[…]), …
and	(1,[56,139,…]), (2,[…]), …
by	(1,[69,157,…]), (2,[…]), …
directed	(1,[60,212,…]), (4,[…]), …
drama	(1,[37,87,…]), (16,[…]), …
…	…

Inverted Index: Flavours

Record-level inverted index:

Maps words to documents without positional information

Term List	Posting List
a	(1,2,…)
american	(1,5,…)
and	(1,2,…)
by	(1,2,…)
directed	(1,2,…)
drama	(1,16,…)
…	…

Word-level inverted index:

Additionally maps words with positional information

Term List	Posting List
a	(1,[21,96,103,…]), (2,[…]), …
american	(1,[28,123]), (5,[…]), …
and	(1,[56,139,…]), (2,[…]), …
by	(1,[69,157,…]), (2,[…]), …
directed	(1,[60,212,…]), (4,[…]), …
drama	(1,[37,87,…]), (16,[…]), …
…	…

Inverted Index: Word Normalisation

drama america

How can we solve this problem?

Inverted index:

Term List	Posting List
a	(1,[21,96,103,…]), (2,[…]), …
american	(1,[28,123]), (5,[…]), …
and	(1,[56,139,…]), (2,[…]), …
by	(1,[69,157,…]), (2,[…]), …
directed	(1,[60,212,…]), (4,[…]), …
drama	(1,[37,87,…]), (16,[…]), …
…	…

Inverted Index: Word Normalisation

drama america

How can we solve this problem?

Normalise words:

Stemming cuts the ends off of words using generic rules:

{ America , American , americas , americanise } → { america }

Inverted index:

Term List	Posting List
a	(1,[21,96,103,…]), (2,[…]), …
american	(1,[28,123]), (5,[…]), …
and	(1,[56,139,…]), (2,[…]), …
by	(1,[69,157,…]), (2,[…]), …
directed	(1,[60,212,…]), (4,[…]), …
drama	(1,[37,87,…]), (16,[…]), …
…	…

Inverted Index: Word Normalisation

drama america

How can we solve this problem?

Normalise words:

Stemming cuts the ends off of words using generic rules:

{ America , American , americas , americanise } → { america }

Lemmatisation uses knowledge of the word to normalise:

{ better , goodly , best } → { good }

Inverted index:

Term List	Posting List
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: Word Normalisation

drama america

How can we solve this problem?

Normalise words:

Stemming cuts the ends off of words using generic rules:

{ America , American , americas , americanise } → { america }

Lemmatisation uses knowledge of the word to normalise:

{ better , goodly , best } → { good }

Synonym expansion

{ film , movie } → { movie }

<u>Inverted index:</u>

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

Inverted Index: Word Normalisation

`drama america`

How can we solve this problem?

Normalise words:

Stemming cuts the ends off of words using generic rules:

{ America , American , americas , americanise } → { america }

Lemmatisation uses knowledge of the word to normalise:

{ better , goodly , best } → { good }

Synonym expansion

{ film , movie } → { movie }

- ➢ Language specific!
- ➢ Use same normalisation on query and document!

Inverted index:

Term List	Posting List
a	(1,[21,96,103,…]), (2,[…]), …
[illegible]	[illegible]), (5,[…]), …
and	(1,[57,139,…]), (2,[…]), …
by	(1,[79,157,…]), (2,[…]), …
directed	(1,[61,212,…]), (4,[…]), …
drama	(1,[38,87,…]), (16,[…]), …
…	…

Inverted Index: Space

Record-level inverted index:

Maps words to documents without positional information

Term List	Posting List
a	(1,2,…)
american	(1,5,…)
and	(1,2,…)
by	(1,2,…)
directed	(1,2,…)
drama	(1,16,…)
…	…

Space? $\sum_{d \in D} \mathrm{U}(d)$ (sum of unique words in all docs)

Word-level inverted index:

Additionally maps words with positional information

Term List	Posting List
a	(1,[21,96,103,…]), (2,[…]), …
american	(1,[28,123]), (5,[…]), …
and	(1,[56,139,…]), (2,[…]), …
by	(1,[69,157,…]), (2,[…]), …
directed	(1,[60,212,…]), (4,[…]), …
drama	(1,[37,87,…]), (16,[…]), …
…	…

Space? $\sum_{d \in D} \mathrm{W}(d)$ (sum of all word occurrences in all docs)

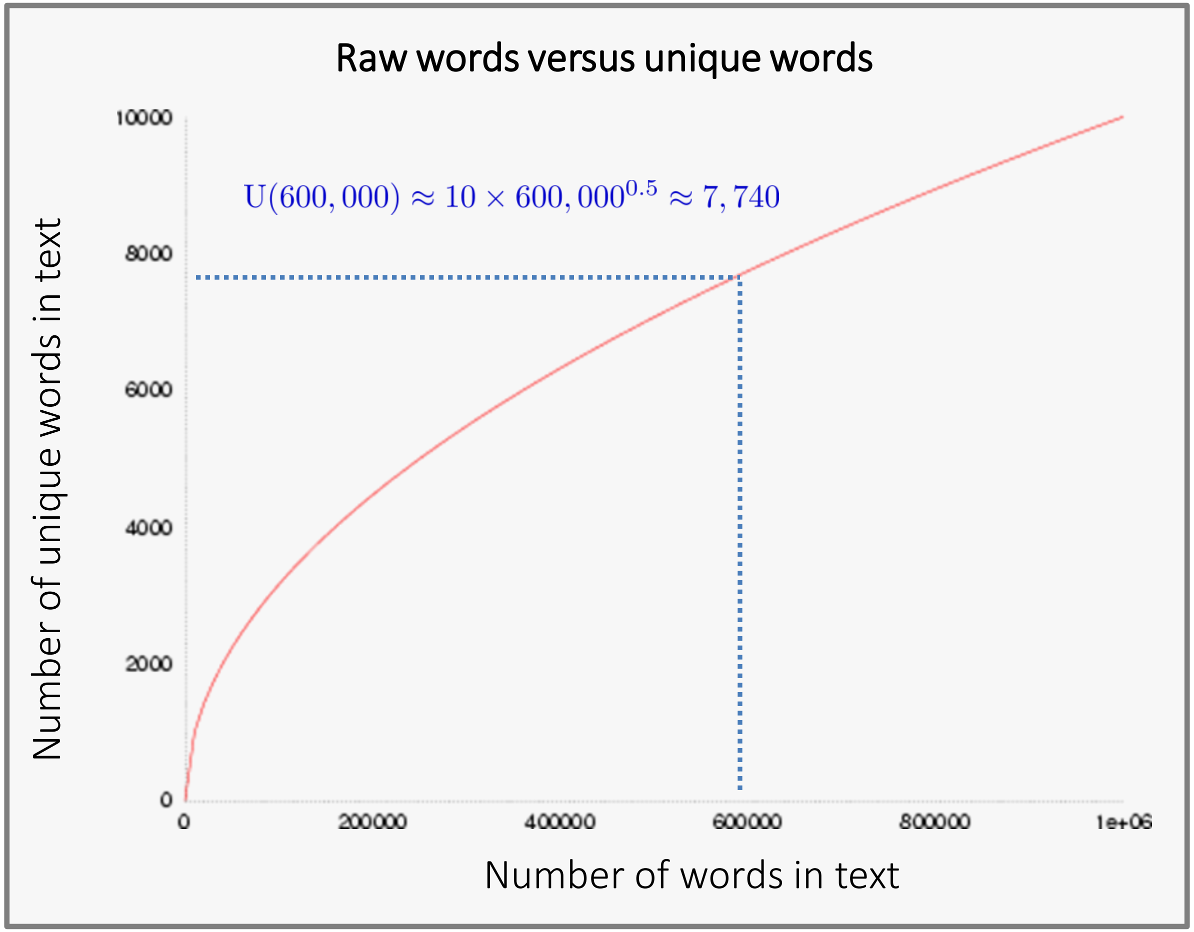
Inverted Index: Unique Words

Not so many unique words …

– Heap's law: $\mathrm{U}(n) \approx Kn^{\beta}$

– English text

- K ∈ [10,100]
- β ∈ [0.4,0.6]



Inverted Index: Space

$\mathrm{U}(d) \approx K \times \mathrm{W}(d)^{\beta}$ ⚠

Record-level inverted index:

Maps words to documents without positional information

Term List	Posting List
a	(1,2,…)
american	(1,5,…)
and	(1,2,…)
by	(1,2,…)
directed	(1,2,…)
drama	(1,16,…)
…	…

Space? $\sum_{d \in D} \mathrm{U}(d)$ (sum of unique words in all docs)

Word-level inverted index:

Additionally maps words with positional information

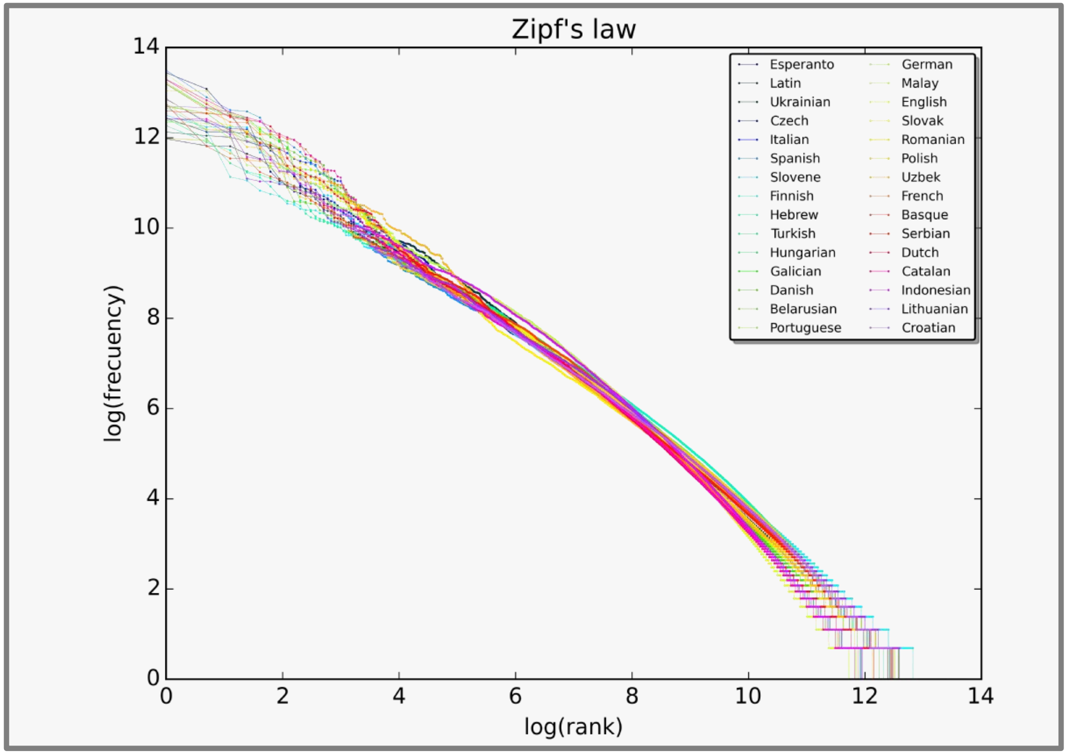
Term List	Posting List
a	(1,[21,96,103,…]), (2,[…]), …
american	(1,[28,123]), (5,[…]), …
and	(1,[56,139,…]), (2,[…]), …
by	(1,[69,157,…]), (2,[…]), …
directed	(1,[60,212,…]), (4,[…]), …
drama	(1,[37,87,…]), (16,[…]), …
…	…

Space? $\sum_{d \in D} \mathrm{W}(d)$ (sum of all word occurrences in all docs)

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.*

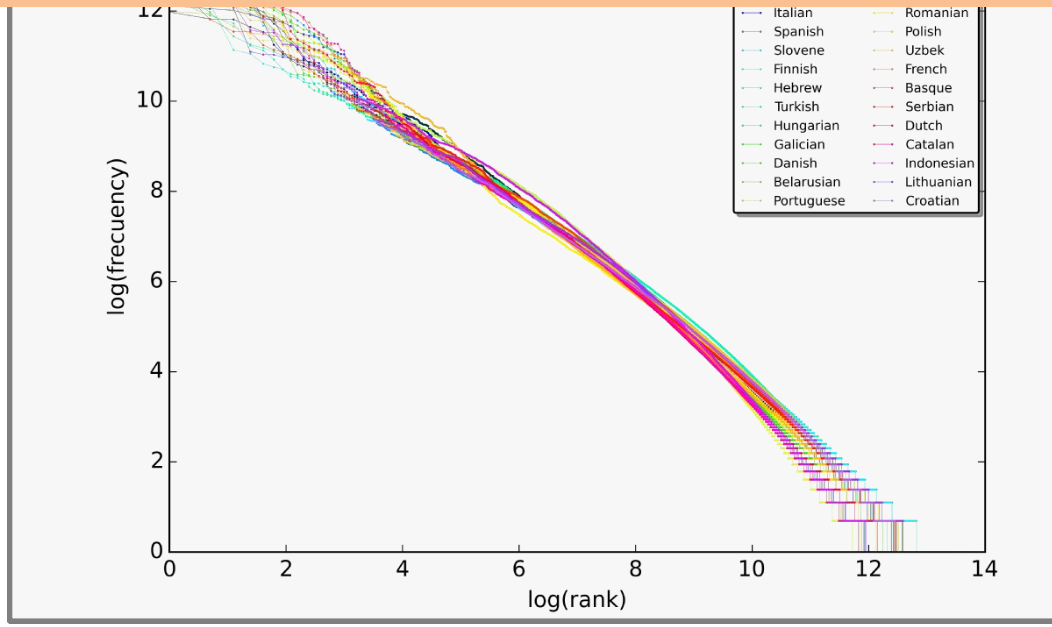
Inverted Index: Common Words

Many occurrences of few words

/ Few occurrences of many words

Expect long posting lists for common words ⚠

- 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

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

```
the drama in america
```

Inverted Index: Common Words

- Perhaps implement stop-words?
- Perhaps implement block-addressing?

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

Block 1 Block 2

What is the effect on phrase search?

Small blocks ~ okay
Big blocks ~ not okay

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

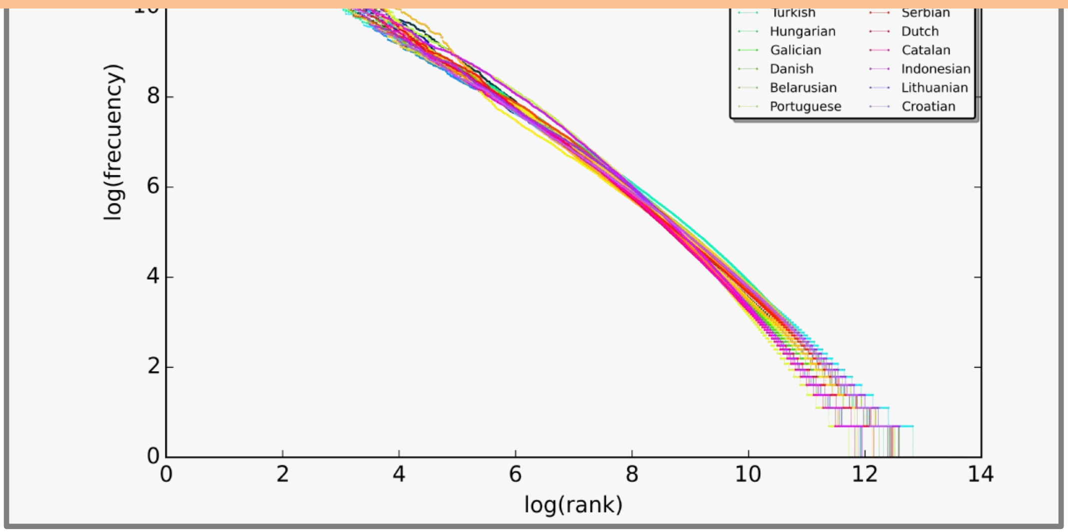
Inverted Index: Common Words

Many occurrences of few words / Few occurrences of many words

Expect long posting lists for common words

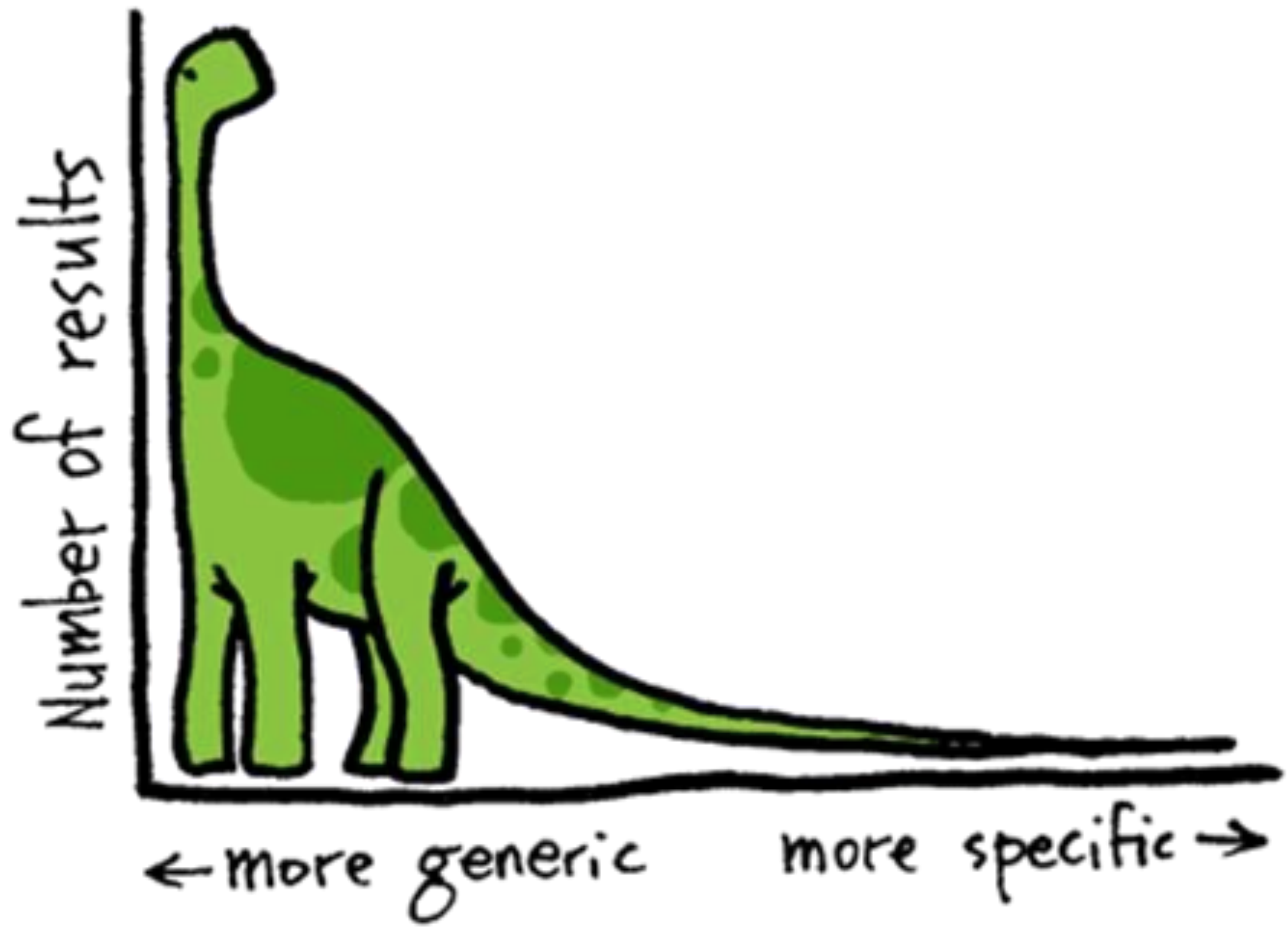
Expect more queries with common words

- “the” 7%
- “of” 3.5%
- “and” 2.7%
- 135 words cover half of all occurrences

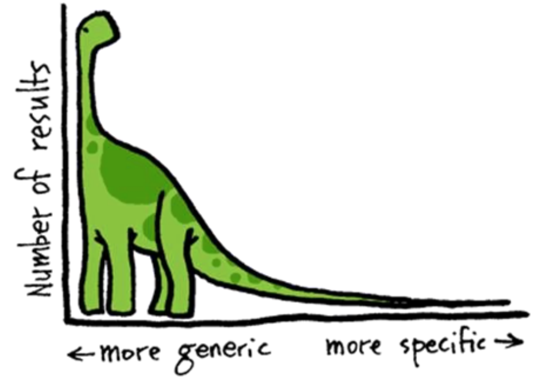


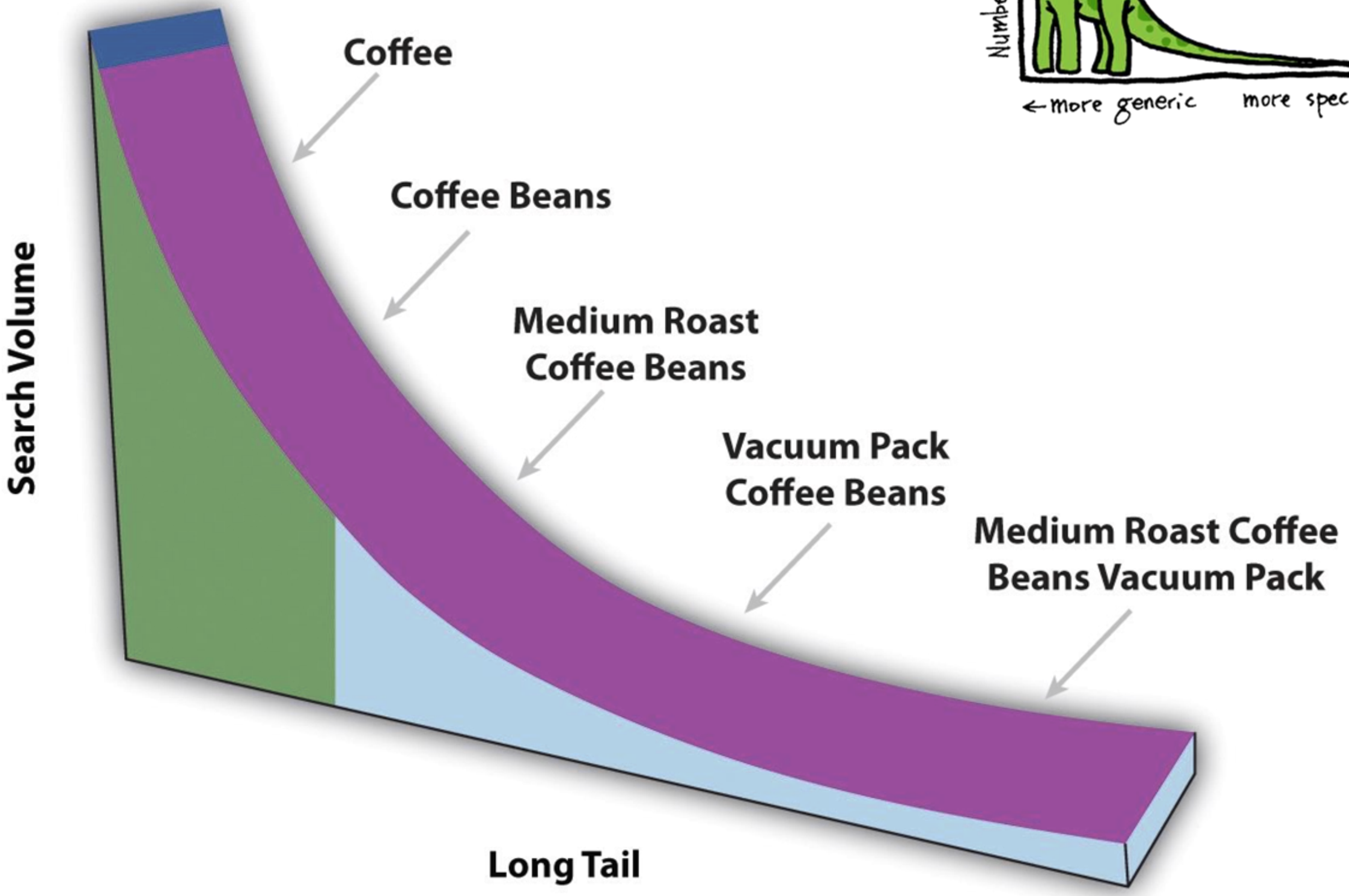
Zipf’s law: *the most popular word will occur twice as often as the second most popular word, thrice as often as the third most popular word, n times as often as the n-most popular word.*

The Long Tail of Search



The Long Tail of Search





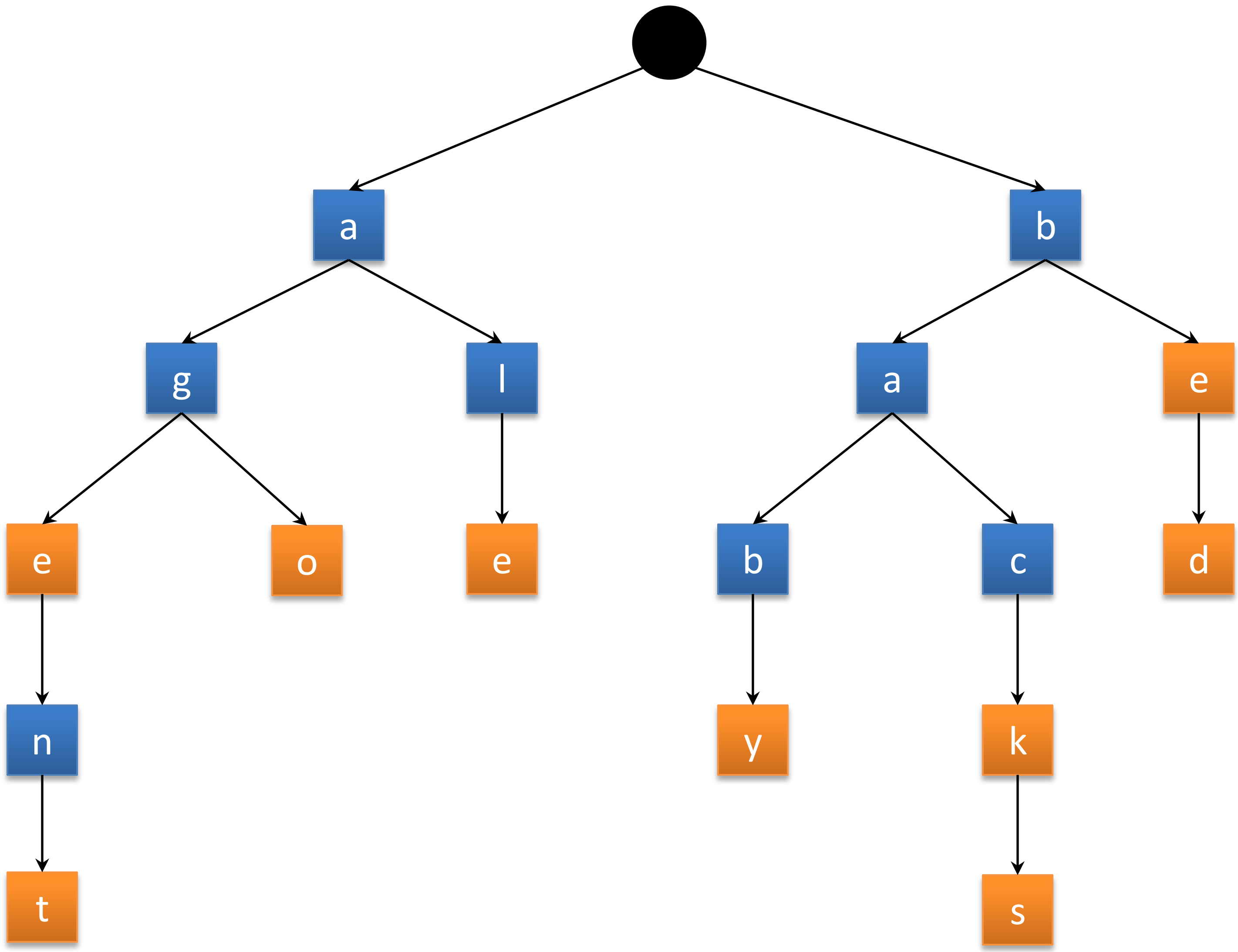
How to optimise for this? Caching for common queries like "coffee"

Search Implementation

- Vocabulary keys:
  - Hashing: O(1) lookups (assuming ideal hashing)
    - no range queries
  - Sorting/B-Tree: O(log(u)) lookups, u unique words
    - range queries
  - Tries: O(l) lookups, l length of the word
    - range queries, compression

Tries?

Trie



Memory Sizes

- Term list (vocabulary keys) small:
  - Often will fit in memory (with compression …)!
- Posting lists larger:
  - On disk / Hot regions <u>cached</u>

Term List	Posting List
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,[…]), …
…	…

Compression techniques

- Numeric compression important

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

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

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

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

$2\lfloor \log_2(z) \rfloor + 1$ bits

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 "0100001100011100001 1001"? ? <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

$\lfloor \log_2(z) \rfloor + 2\lfloor \log_2(\lfloor \log_2(z) \rfloor + 1) \rfloor + 1$ bits

z: integer to encode	$\lfloor \log_2(z) \rfloor + 1$ coded in Elias γ	next $\lfloor \log_2(z) \rfloor$ 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: Bit Level

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

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

Comparison

- Small values

z: input integer	Elias γ code	Elias δ code	Golomb-3 code
1	0	0	00
2	100	1000	010
3	101	1001	011
4	11000	10100	100
5	11001	10101	1010
6	11010	10110	1011
7	11011	10111	1100
8	1110000	11000000	11010

- Larger values

z: input integer	Elias γ code	Elias δ code	Golomb-3 code
100	1111110100100	10110100100	1111111...101
…			…

Compression techniques: Byte Level

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

- For example:

<table>
<tr><td>00100100</td><td>10100010</td><td>00000101</td><td>00100100</td></tr>
<tr><td>18</td><td>81</td><td colspan="2">274</td></tr>
</table>

Other Optimisations

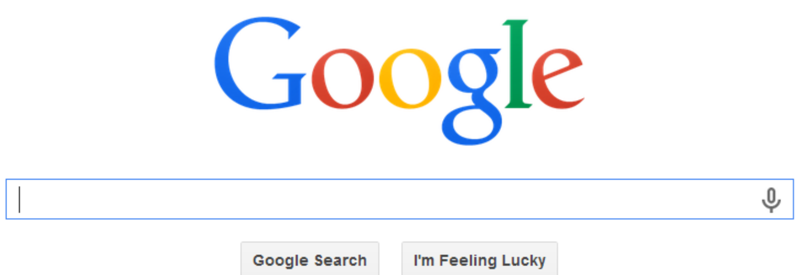
- Top-Doc: Order posting lists to give likely "top documents" first: good for top-*k* results

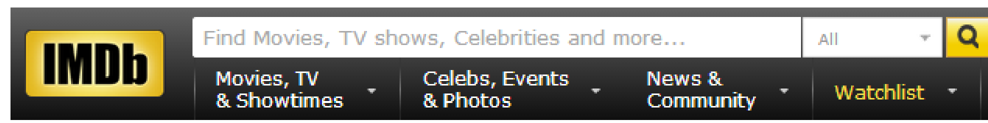
- Selectivity: Load the posting lists for the most rare keywords first; apply thresholds

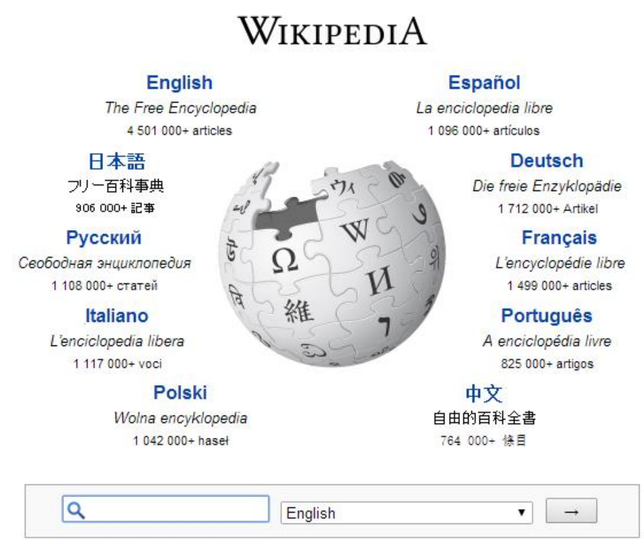
- Sharding: Distribute over multiple machines [...]

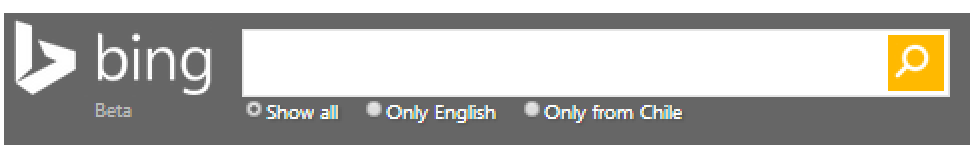
Extremely Scalable/Efficient

When engineered correctly ☺









DISTRIBUTING AN INVERTED INDEX

Inverted Index: Distribution

Term	Posting
and	1,3,4,5,6
ate	1,2,3
cat	3,4,6
dog	3,5,6,7
the	1,2,3,4,5,6,7
vet	4

How might we distribute an inverted index?

Split by word

Term	Posting
dog	3,5,6,7

Term	Posting
and	1,3,4,5,6
vet	4

Term	Posting
ate	1,2,3
the	1,2,3,4,5,6,7

Term	Posting
cat	3,4,6

Possible disadvantages?

- Complications for load balancing given common words
- AND or PHRASE search within each document involves multiple machines
- Difficult to store statistics, etc., for a document (not usually a big issue)

Inverted Index: Distribution

Term	Posting
and	1,3,4,5,6
ate	1,2,3
cat	3,4,6
dog	3,5,6,7
the	1,2,3,4,5,6,7
vet	4

How might we distribute an inverted index?

Split by document

Term	Posting
ate	2
cat	6
dog	6
the	2,6

Term	Posting
and	3,5
ate	3
cat	3
dog	5
the	3,5

Term	Posting
and	1
ate	1
the	1

Term	Posting
and	4
cat	4
dog	7
the	7
vet	4

Possible disadvantages?

- All searches require unions over multiple machines
- Might be load balancing issues for large documents (not usually a big issue)
- Longer term lists per machine (not usually a big issue)

LUCENE: TEXT INDEXING

Apache Lucene

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

Doug Cutting (above) & Mike Cafarella (below)

Lucene™

nutch

hadoop

Apache Solr

- Inverted Index
  - Based on Apache Lucene
  - Higher-level interfaces (and richer features)
  - Distributed by document  … 

Elasticsearch

- Inverted Index
  - Based on Apache Lucene
  - Higher-level interfaces (and richer features)
  - Distributed by document  … 

Elasticsearch: a word of warning

Reported Elasticsearch data breaches [edit]

- 2018-11-15 AWS Elasticsearch database belonging to VoxOx exposed tens of millions of text messages, including password reset links, two-factor codes, shipping notifications and more.[36]
- 2018-11-27 Elasticsearch database belonging to Urban Massage exposed more than 309,000 user records, including names, email addresses and phone numbers.[37]
- 2019-01-12 Elasticsearch server belonging to do-it-yourself chain, B&Q exposed personal details of individuals caught or suspected of stealing goods from stores.[38][39]
- 2019-01-21 Elasticsearch database belonging to Youth-run agency AIESEC exposed over 4 million intern applications including the applicant's name, gender, date of birth, and the reasons why the person was applying for the internship.[40]
- 2019-01-23 Elasticsearch database belonging to Ascension Data and Analytics exposed 24 million financial and banking documents, representing tens of thousands of loans and mortgages from some of the biggest banks in the U.S.[41]
- 2019-09-13 Elasticsearch database belonging to Dealer Leads exposed 198 million car buying records which contained the personal information of customers.[42]
- 2019-10-26 Elasticsearch database belonging to Adobe exposed 7.5 million customer records which contained email addresses, Adobe member IDs (usernames), country of origin, and what Adobe products they were using.[43]
- 2019-11-19 Elasticsearch database belonging to Conrad Electronic exposed 14 million customer records which contained postal addresses, in parts fax- and telephone numbers as well as IBANs on a fifth of the exposed data-records.[44]

https://en.wikipedia.org/wiki/Elasticsearch

Questions?