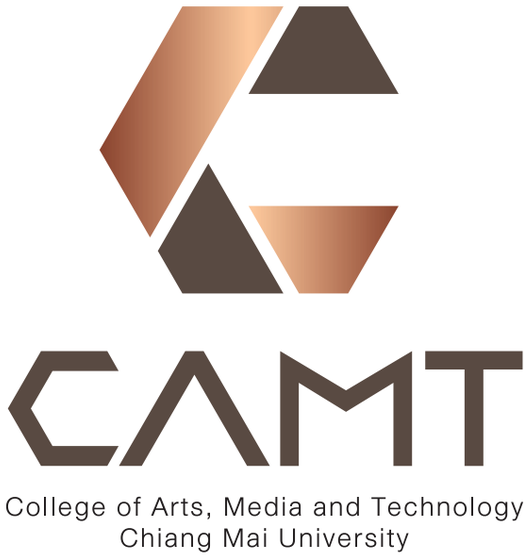
SE 481 Introduction to Information Retrieval

Module #6 — Web Search



Passakorn Phannachitta, D.Eng.

passakorn.p@cmu.ac.th

College of Arts, Media and Technology

Chiang Mai University, Chiangmai, Thailand

Agenda

- WWW
- Web search engine
- Link analysis

WWW

- Developed by Tim Berners-Lee in 1990 at CERN to organize research documents available on the Internet.
- Combined idea of documents available by FTP with the idea of hypertext to link documents.
- Developed initial HTTP network protocol, URLs, HTML, and first web server.

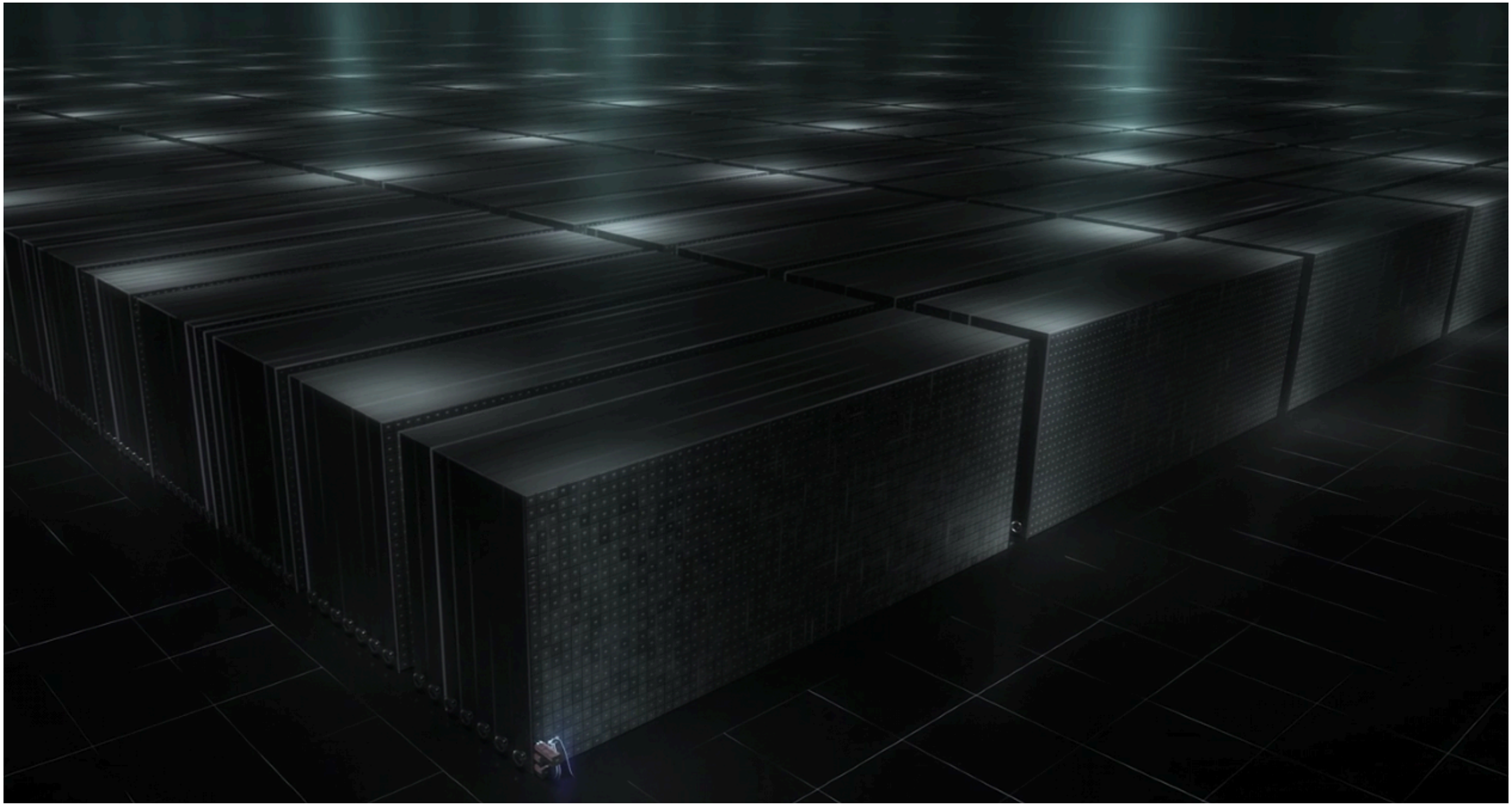
WWW — Pre history

- Ted Nelson developed idea of hypertext in 1965.
- Doug Engelbart invented the mouse and built the first implementation of hypertext in the late 1960's at SRI.
- ARPANET was developed in the early 1970's.
- The basic technology was in place in the 1970's; but it took the PC revolution and widespread networking to inspire the web and make it practical.

WWW — History

- Early browsers were developed in 1992.
- In 1993, Marc Andreessen and Eric Bina at UIUC NCSA developed the Mosaic browser and distributed it widely.
- Andreessen joined with James Clark (Stanford Prof. and Silicon Graphics founder) to form Mosaic Communications Inc. in 1994 (which became Netscape to avoid conflict with UIUC).
- Microsoft licensed the original Mosaic from UIUC and used it to build Internet Explorer in 1995.

Without search engines, the web wouldn’t work



Without search engines, the web wouldn’t work

- There is no incentive to create contents
- E.g., Pinterest

https://pepper.agency/blog/advertising-on-pinterest/

Search engine — Early history

- By late 1980's many files were available by anonymous FTP.
- In 1990, Alan Emtage of McGill Univ. developed Archie (short for "archives")
  - Assembled lists of files available on many FTP servers.
  - Allowed regex search of these file names.
- In 1993, Veronica and Jughead were developed to search names of text files available through Gopher servers.

Search engine — History

- In 1993, early web robots (spiders) were built to collect URL's:
  - Wanderer
  - ALIWEB (Archie-Like Index of the WEB)
  - WWW Worm (indexed URL's and titles for regex search)
- In 1994, Stanford grad students David Filo and Jerry Yang started manually collecting popular web sites into a topical hierarchy called Yahoo.

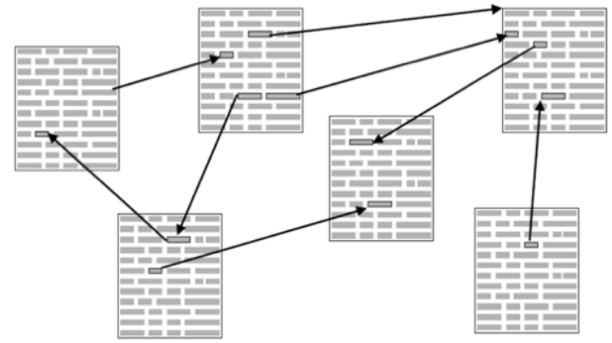
Search engine — History

- In early 1994, Brian Pinkerton developed WebCrawler as a class project at U Wash. (eventually became part of Excite and AOL).
- A few months later, Fuzzy Maudlin, a grad student at CMU developed Lycos. First to use a standard IR system as developed for the DARPA Tipster project.
  - First to index a large set of pages.
- In late 1995, DEC developed Altavista. Used a large farm of Alpha machines to quickly process large numbers of queries. Supported boolean operators, phrases, and “reverse pointer” queries.

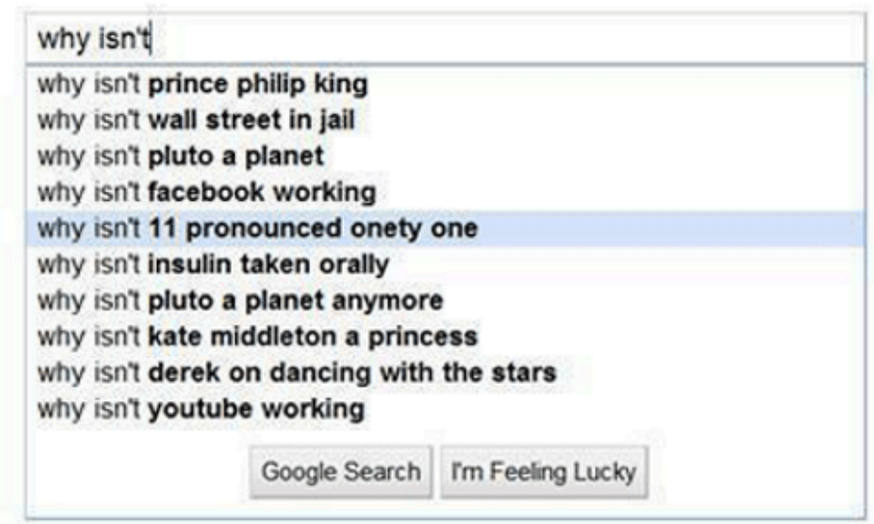
Search engine — History

- In 1998, Larry Page and Sergey Brin, Ph.D. students at Stanford, started Google.
  - Main advance is use of link analysis to rank results partially based on authority.
- Microsoft launched MSN Search in 1998 based on Inktomi (started from UC Berkeley in 1996), changed to Live Search in 2007, and Bing in 2009

The differences from traditional IR



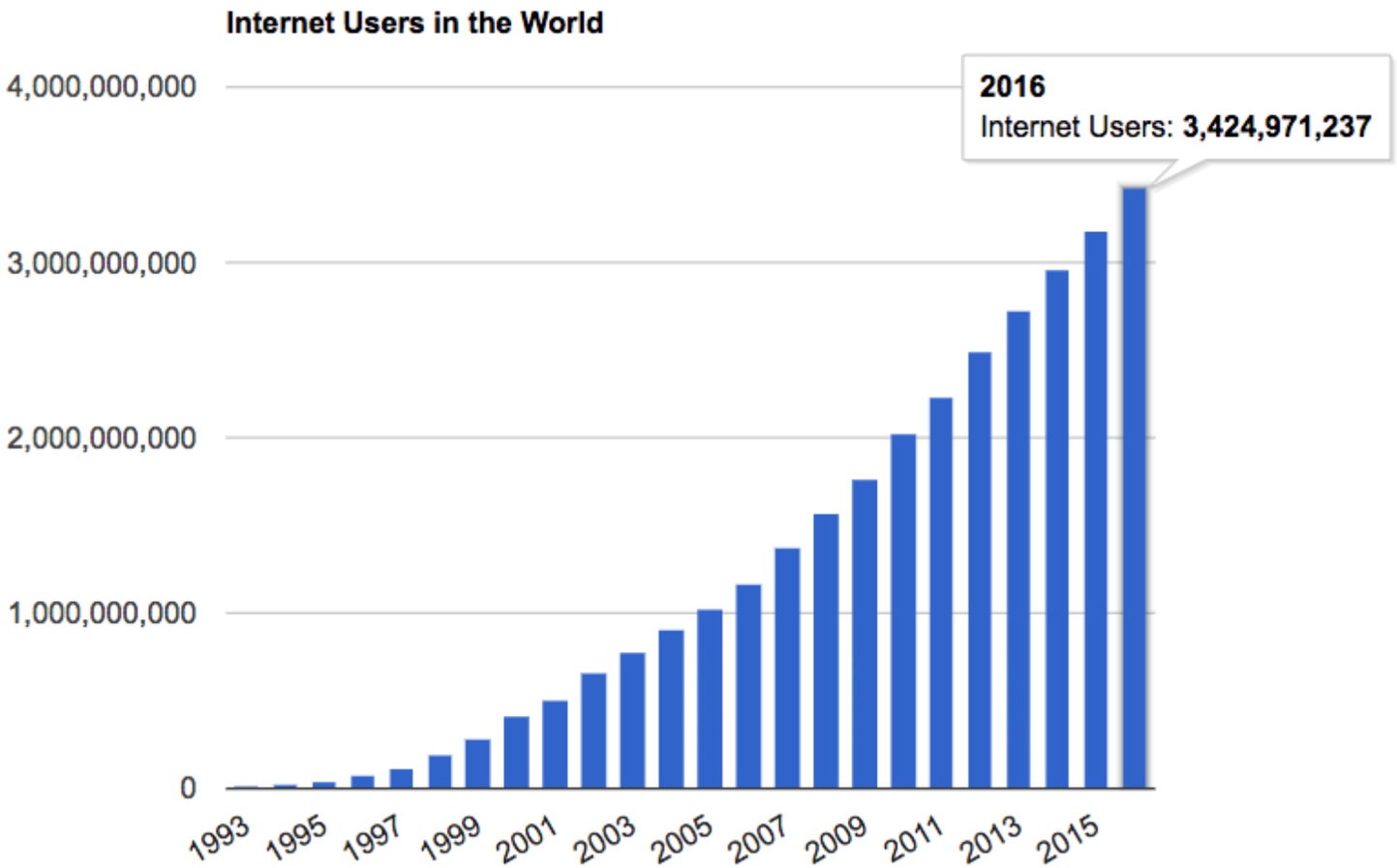
- It has links.
- Queries are much more varied.
- Users are much more varied and there are a lot of them.
- Documents are also much more varied
- Context is more important on the web than in many other IR applications.
- It contains a lot of Ads and Spam



Challenges

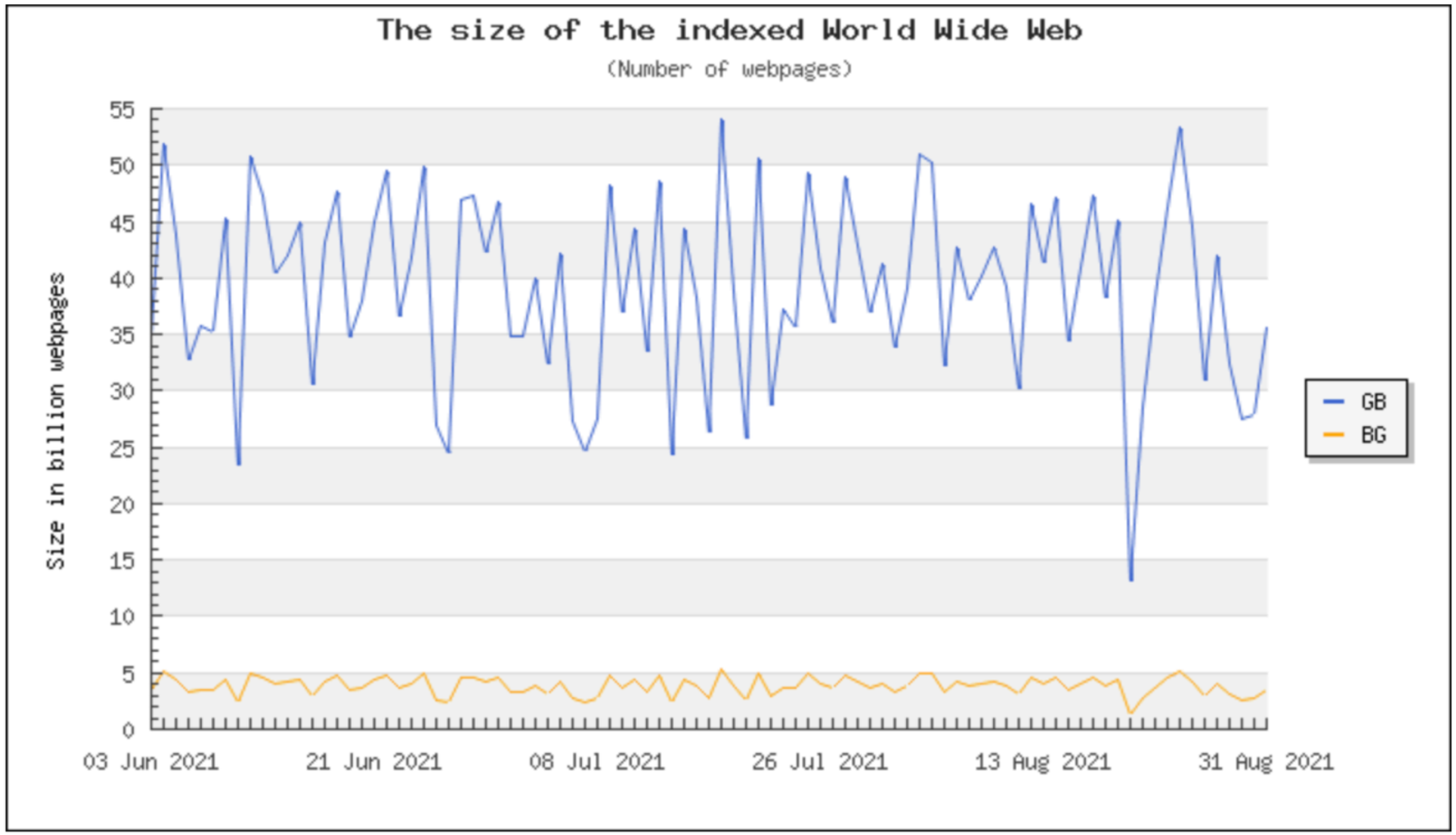
- **Distributed data** — Documents spread over millions of different web servers.
- **Volatile data** — Many documents change or disappear rapidly (e.g. dead links).
- **Large volume** — Billions of separate documents.
- **Unstructured and Redundant Data** — No uniform structure, HTML errors, up to 30% (near) duplicate documents.
- **Quality of Data** — No editorial control, false information, poor quality writing, typos, etc.
- **Heterogeneous Data** — Multiple media types (images, video), languages, character sets, etc.

#Internet users



http://www.internetlivestats.com/internet-users/#trend

Current size of the web



GB = Sorted on Google and Bing

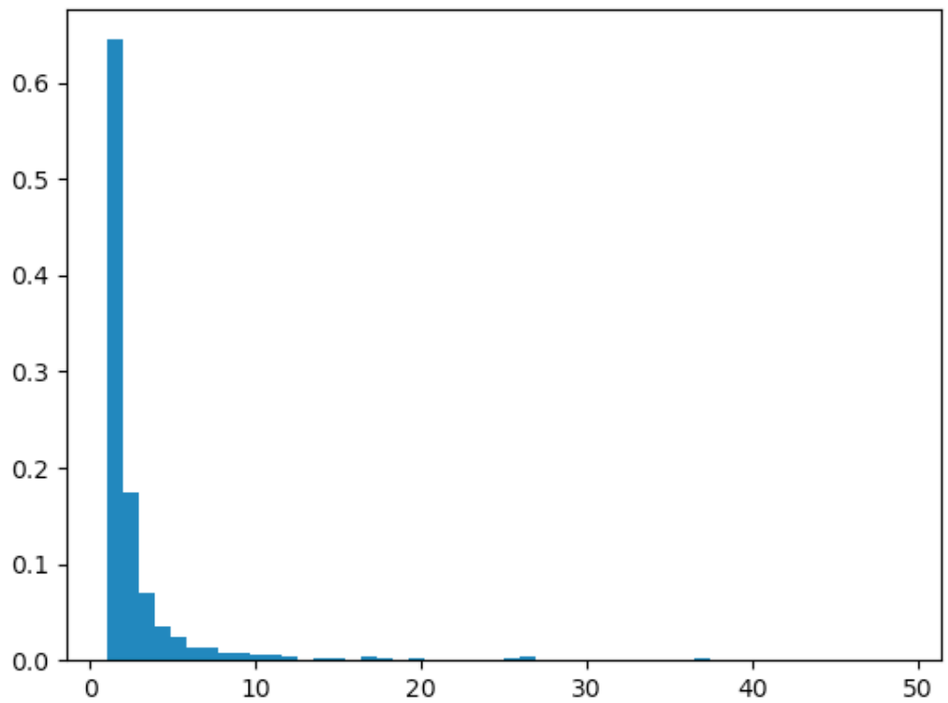
BG = Sorted on Bing and Google

https://www.worldwidewebsize.com/

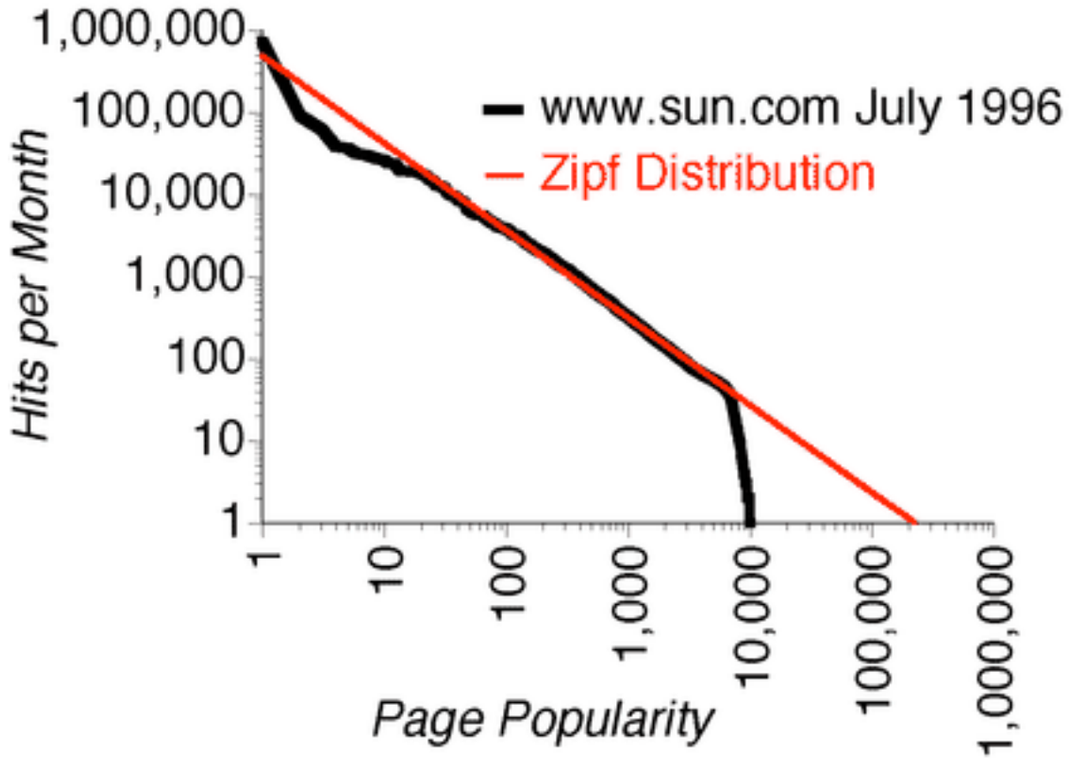
Zipf's law distribution

- Number of in-links/out-links to/from a page has a Zipf distribution.
- Length of web pages also has a Zipf distribution.
- Number of hits to a web page also has a Zipf distribution.

$$p(x) = \frac{x^{-}1}{\zeta(a)}$$



Zipf's law and page popularity



https://www.nngroup.com/articles/zipf-curves-and-website-popularity/

Business models for web search

- Advertisers pay for banner ads on the site that do not depend on a user's query.
  - **CPM**: Cost Per Mille (thousand impressions). Pay for each ad display.
  - **CPC**: Cost Per Click. Pay only when user clicks on ad.
    - **CTR**: Click Through Rate. Fraction of ad impressions that result in clicks throughs. CPC = CPM / (CTR * 1000)
    - **CPA**: Cost Per Action (Acquisition). Pay only when user actually makes a purchase on target site.
- Advertisers bid for "keywords". Ads for highest bidders displayed when user query contains a purchased keyword.
  - **PPC**: Pay Per Click. CPC for bid word ads (e.g. Google AdWords).

Business models for web search — History

- Initially, banner ads paid thru **CPM** were the norm.
- GoTo Inc. formed in 1997 and originates and patents bidding and **PPC** business model.
- Google introduces AdWords in fall 2000.
- GoTo renamed as Overture in Oct. 2001.
- Overture sues Google for use of PPC in Apr. 2002.
- Overture acquired by Yahoo in Oct. 2003.
- Google settles with Overture/Yahoo for 2.7 million shares of Class A common stock in Aug. 2004

Where's Google™ making its money?

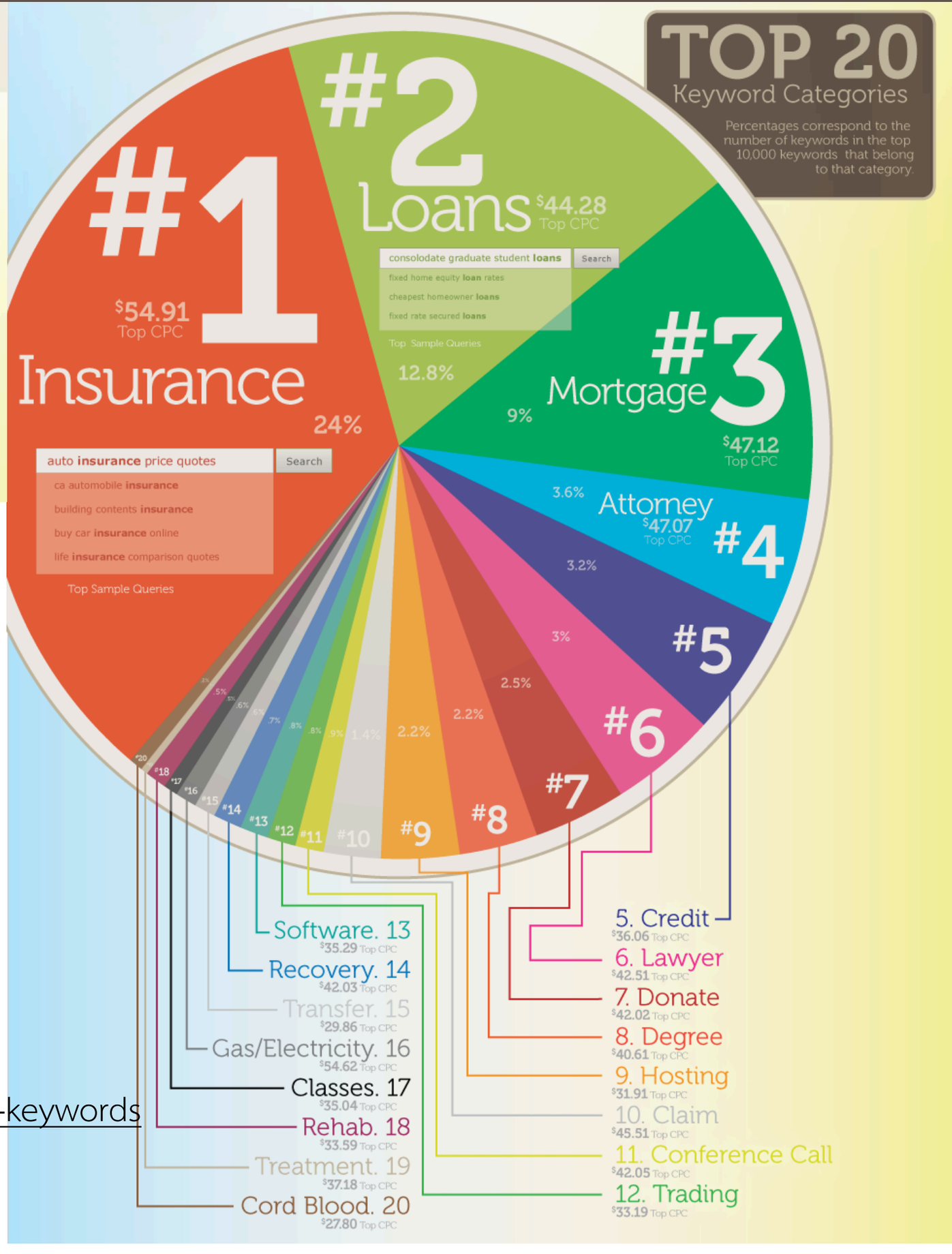
$32.2 billion
in total advertising revenue

SPOILER ALERT: ADVERTISING!

$33.3 billion
in total revenue *in Q3 2010 - Q2 2011*

of Google's Revenue
97%
is from advertising

Top 20
Most Expensive Keywords
in Google AdWords Advertising



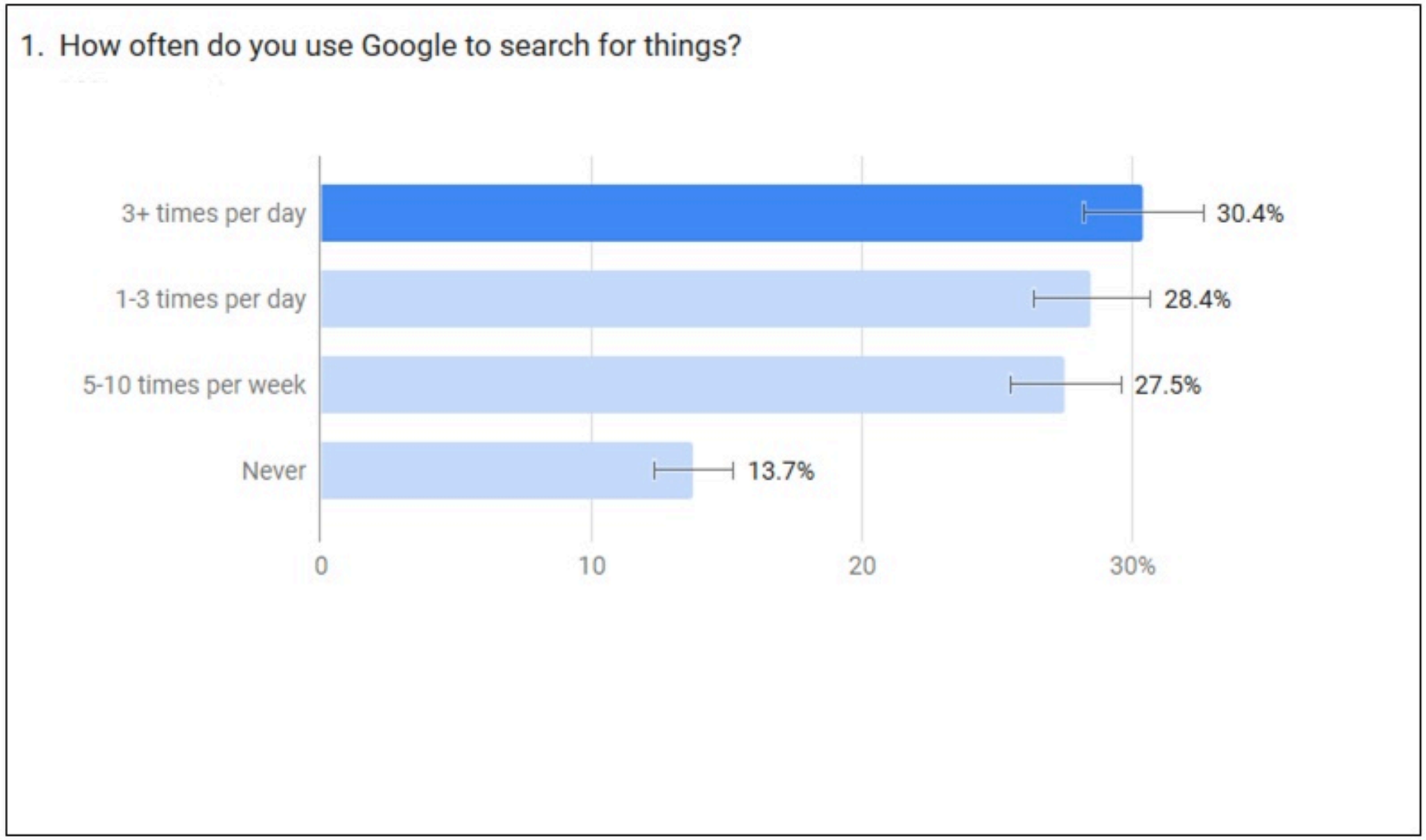
https://www.wordstream.com/articles/most-expensive-keywords

CAMT 2021
College of Arts, Media and Technology
Chiang Mai University

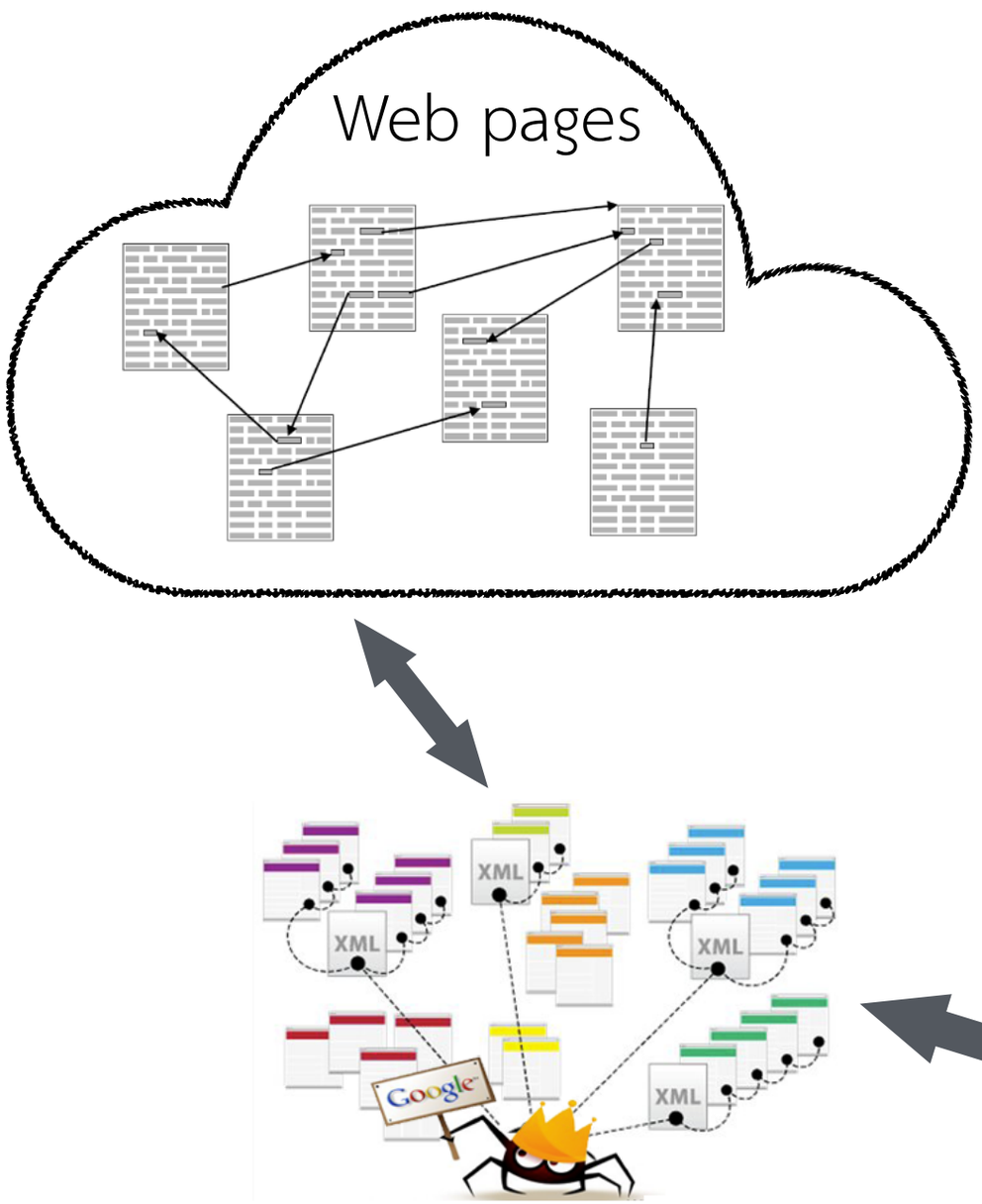
presented by WordStream

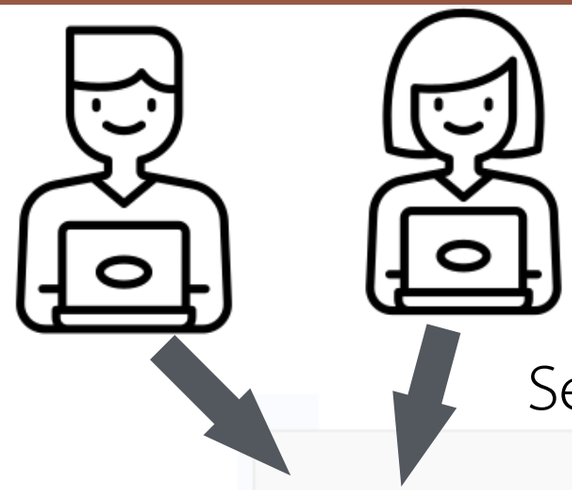
developed by: nowsourcing.com

According to Teknicks

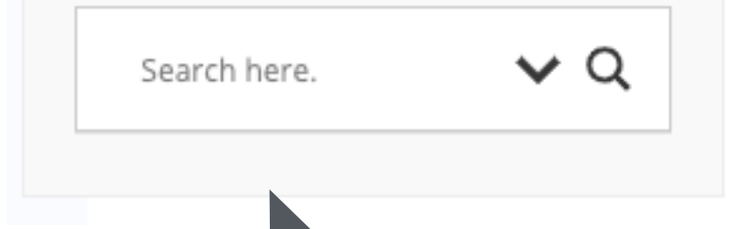


Web search using IR





Search interface



Web crawler

Indexer

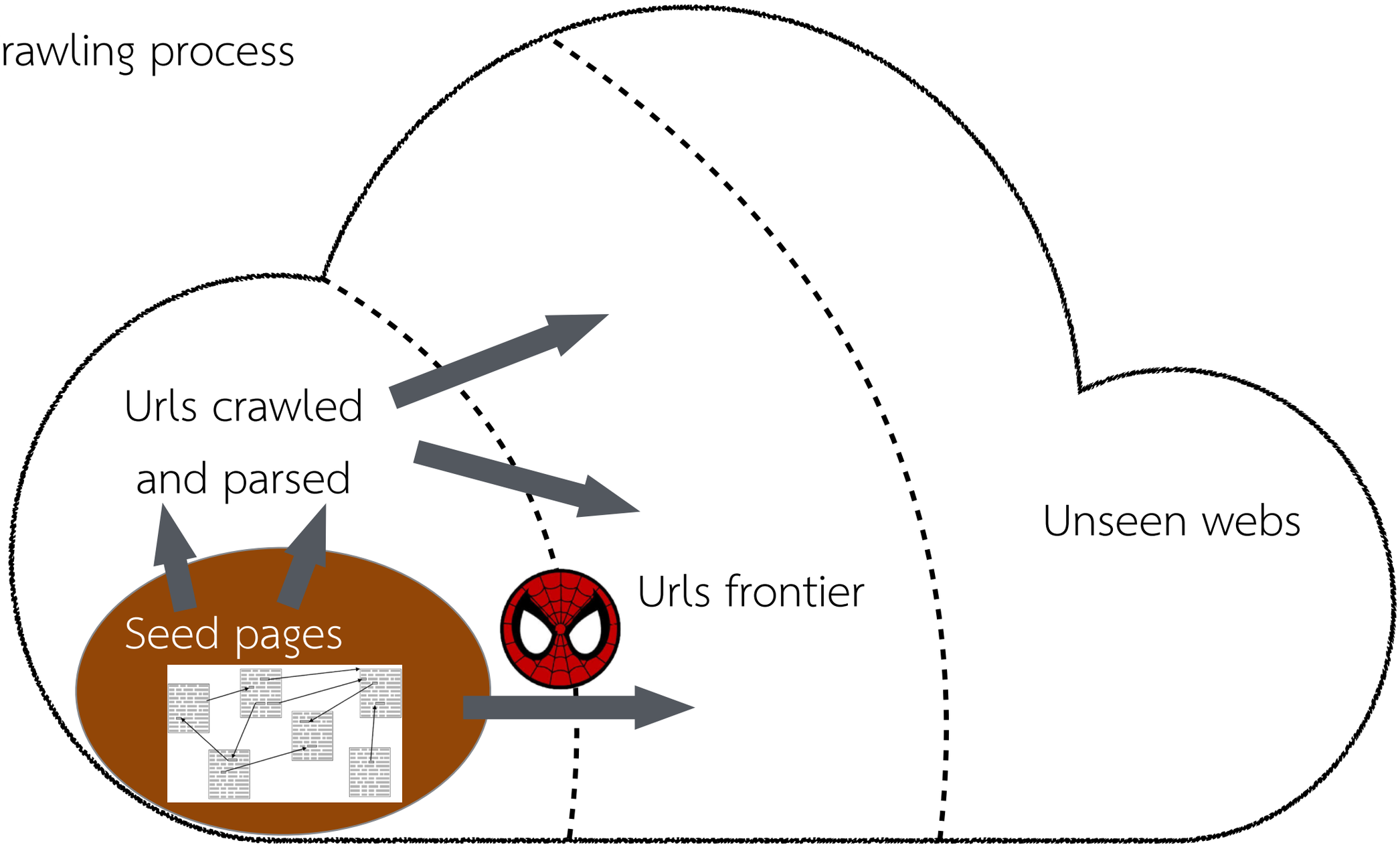
Indexes

Web crawler

- A.k.a., Web spiders (robots / bots / crawlers)
- Procedures —
  - Begin with a comprehensive set of seed url's from which to start the search by putting them all in the queue.
  - For each url, fetch, parse and extract all the urls in the page recursively to find additional pages.
  - Index all the urls to the newly found pages and place them in the queue
  - Repeat

Graphical explanation

Crawling process



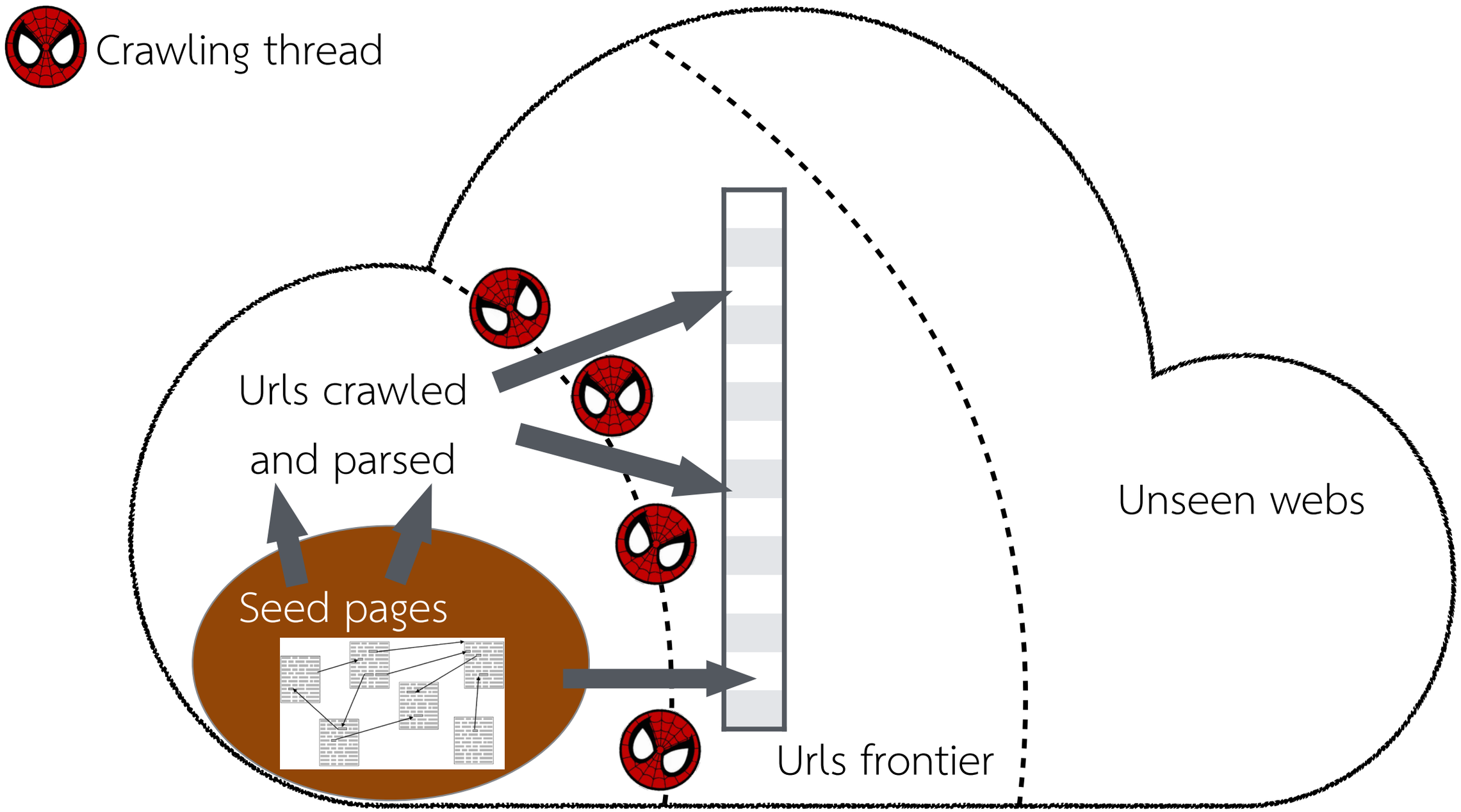
Some difficulties

- Not an app fit for one single computer — distributed computing required
- Full of malicious pages out there
- Mirrors and duplicate pages
- Resource availability — latency | bandwidth
- Etiquette

What to concern when building a crawler

- Capability — can be computed in parallel, can scale large, and have high efficiency
- Robustness — immune to numerous malicious behavior from web servers
- Politeness — respect **implicit** and **explicit** politeness considerations

Scalability



Duplication avoidance ~ efficiency

- URL normalization
  - E.g., https://www.jetbrains.com/pycharm/#chooseYourEdition

https://www.jetbrains.com/pycharm

- Then store the normalized URL in a big hash table, e.g., memCached or Redis
  - More sophisticated algorithms are also used, e.g., using computer networking algorithms such as slow start in TCP, or statistical algorithms such as Markov chain, or machine learning algorithms such as neural networks.

Politeness

- Politeness — respect **implicit** and **explicit** politeness considerations

  - **Explicit politeness:** specifications from webmasters on what portions of site can be crawled
    - robots.txt
  - **Implicit politeness:** even with no specification, avoid hitting any site too often

Robots.txt

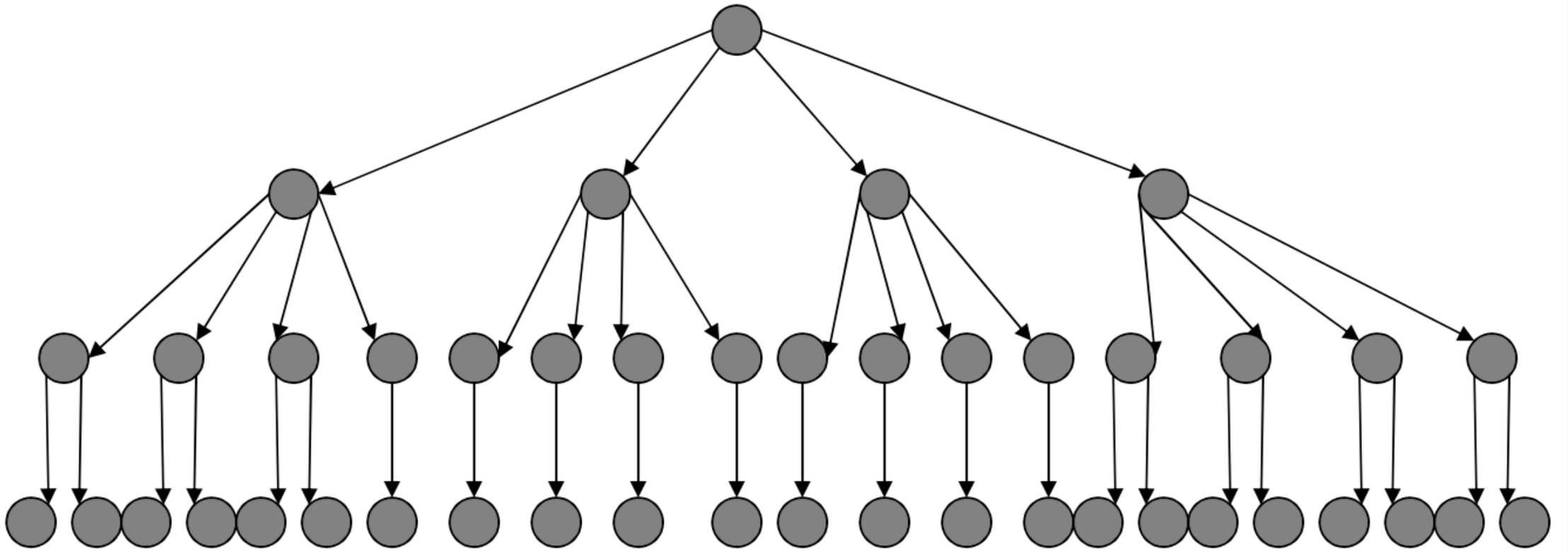
- Protocol for giving spiders (robots) limited access to a website
- Originally from 1994
- Details elaborated in http://www.robotstxt.org/robotstxt.html
- Website announces its request on what can(not) be crawled
- For a server, create a file /robots.txt to specify access restrictions
- E.g.,

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

User-agent: searchengine
Disallow:
```

No robot should visit any URL starting with "/yoursite/temp/", except the robot called searchengine":

Graph-search algorithm

- Should be Breadth-first search (BFS) or Depth-first search (DFS)?



Link extraction

- E.g., extract all links from https://cmu.ac.th/en/faculty/aboutus
  - 1 levels
  - 2 levels

- Note — Python has Scrappy but it does not fully support multithreading, so let's try building one from the scratch

- The code is modified from the original @ https://edmundmartin.com/multi-threaded-crawler-in-python/

A simple web crawler

```
01  class MultiThreadCrawler:
02      def __init__(self, base_url, depth):
03          self.base_url = base_url
04          extracted_url = urlparse(base_url)
05          parent = extracted_url.path[:extracted_url.path.rfind("/") + 1]
06          self.root_url = '{}://{}{}'.format(extracted_url.scheme, extracted_url.netloc, parent)
07          self.pool = ThreadPoolExecutor(max_workers=multiprocessing.cpu_count() - 1)
08          self.to_crawl = Queue()
09          self.to_crawl.put({self.base_url: depth})
10          self.stored_folder = Path(__file__).parent / '../crawled/'
11
12          if not Path(self.stored_folder).exists():
13              Path.mkdir(self.stored_folder)
14
15          if Path(self.stored_folder / 'url_list.pickle').exists():
16              with open(self.stored_folder / 'url_list.pickle', 'rb') as f:
17                  self.crawled_pages = pickle.load(f)
18              print(self.crawled_pages)
19          else:
20              self.crawled_pages = set([])
```

Entry point

Current directory

Create a queue

Add the entry url to the queue

Continue from the saved work or just start a new one

A simple web crawler

```
01      def run_scraper(self):
02          while True:
03              try:
04                  target = self.to_crawl.get(timeout=10)
05                  url, depth = [(k, target[k]) for k in target][0]
06                  if url not in self.crawled_pages:
07                      self.crawled_pages.add(url)
08                      job = self.pool.submit(self.get_page, url, depth - 1)
09                      job.add_done_callback(self.extract_page)
10              except Empty:
11                  with open(self.stored_folder / 'url_list.pickle', 'wb') as f:
12                      pickle.dump(self.crawled_pages, f, pickle.HIGHEST_PROTOCOL)
13                  with open(self.stored_folder / 'url_list.pickle', 'rb') as f:
14                      print(pickle.load(f))
15                  break
16              except Exception as e:
17                  print(e)
18                  continue
19
20
21  if __name__ == '__main__':
22      s = MultiThreadCrawler("https://camt.cmu.ac.th/index.php/en/", 1)
23      s.run_scraper()
```

Dequeue the first entry

If it has never been processed

If we cannot dequeue,
then it means our task is done

An entry url example

A simple web crawler

```
1   def extract_page(self, obj):
2       if obj.result():
3           result, url, depth = obj.result()
4           if result and result.status_code == 200:
5               url_lists = self.parse_links(result.text, depth)
6               self.parse_contents(url, result.text, url_lists)
```

```
1   def get_page(self, url, depth):
2       try:
3           res = requests.get(url, timeout=(3, 30)) ◀.............. Download the webpage
4           return res, url, depth
5       except requests.RequestException:
6           return
```

A simple web crawler

```
01  def parse_links(self, html, depth):
02      soup = BeautifulSoup(html, 'html.parser')
03      links = soup.find_all('a', href=True)
04      url_lists = []
05      for link in links:
06          url = link['href']
07          url = urljoin(self.root_url, url)
08          if depth >= 0 and '..' not in url and url not in self.crawled_pages:
09              print("Adding {}".format(url))
10              self.to_crawl.put({url: depth})
11              url_lists.append(url)
12      return url_lists
```

Find all links by extracting all the <a href>

Make full path from relative path if not already a full path

Add to the queue if not crawled depth has not reached the threshold.

A simple web crawler

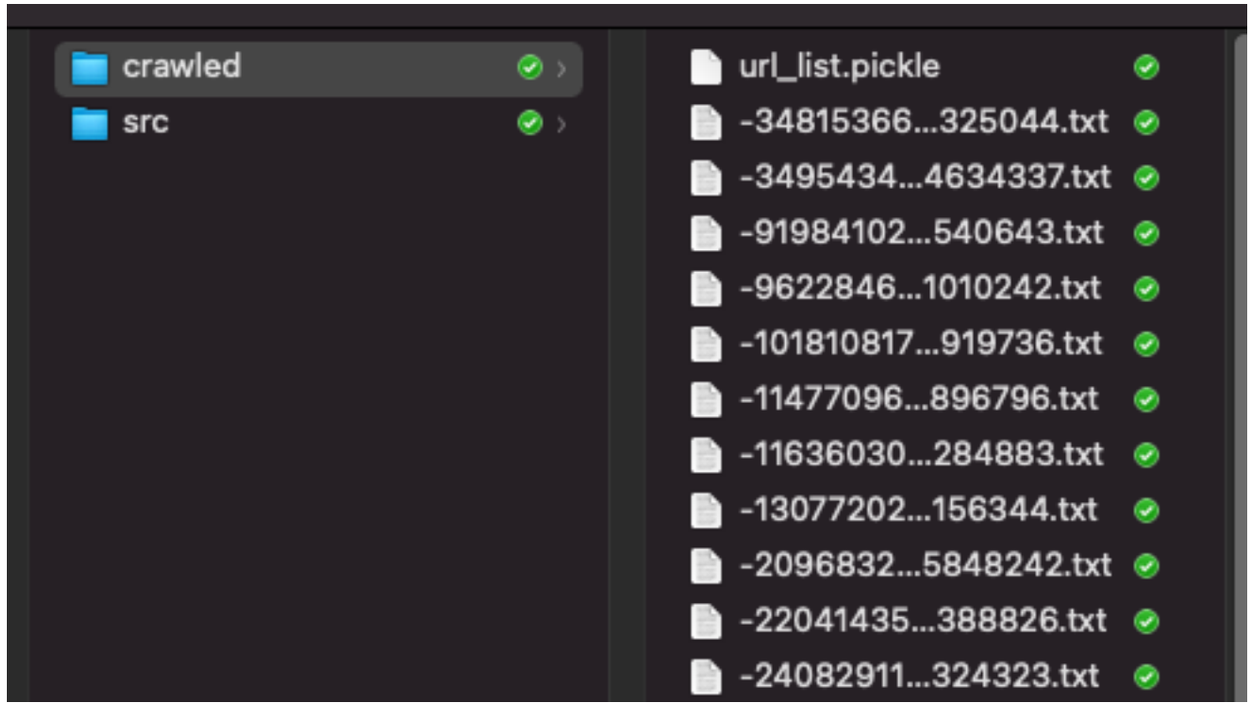
```
01  def parse_contents(self, url, html, url_lists):
02      def tag_visible(element):
03          if element.parent.name in ['style', 'script', 'head', 'title', 'meta', '[document]']:
04              return False
05          if isinstance(element, Comment):
06              return False
07          return True
08
09      try:
10          soup = BeautifulSoup(html, 'html.parser')
11          texts = soup.findAll(text=True)
12          visible_texts = filter(tag_visible, texts)
13
14          title = soup.find('title').string.strip()
15          text = u" ".join(t.strip() for t in visible_texts).strip()
16
17          with open(self.stored_folder / (str(hash(url)) + '.txt'), 'w', encoding='utf-8') as f:
18              json.dump({'url': url, 'title': title, 'text': text, 'url_lists': url_lists}, f,
ensure_ascii=False)
19      except:
20          pass
```

Get all the texts from visible tags

Parse into json format and write to a textual file

A simple web crawler

- After executing the main we will get
  - A pickle (binary file) storing a mapping between the webpage url and the hashed values of the url which we use them as file names.
  - File storing json {title: , text:, } one file per one webpage.

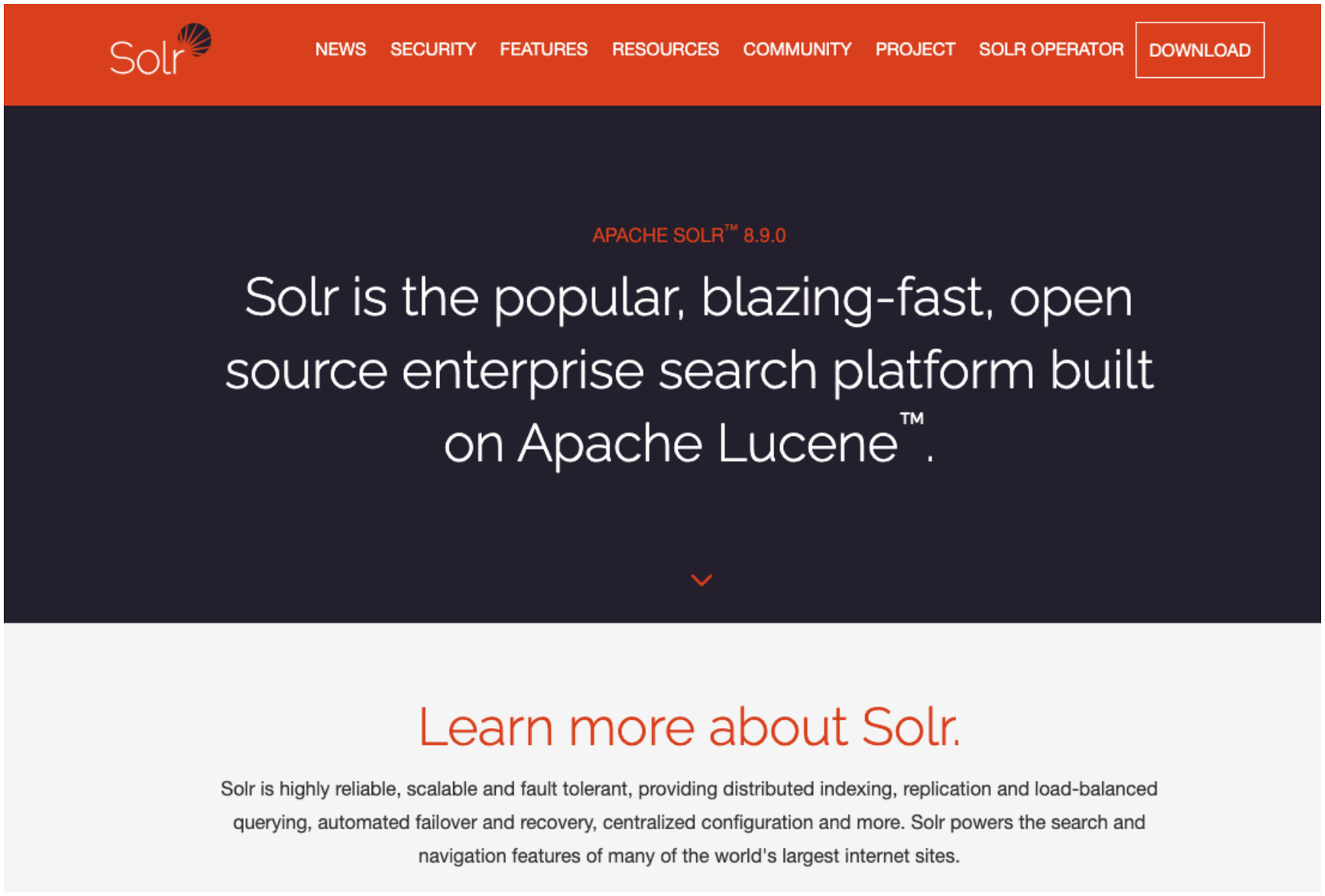


A simple web crawler

- Ideas for some improvement
  - Storing all in a database
  - Checking the content, e.g., language before storing them.

A simple web indexer

- Apache Solr



Ref: https://solr.apache.org/

Apache Solr

- Operations
  - **Indexing**: first of all, it converts the documents into a machine-readable format which is called Indexing.
  - **Querying**: understanding the terms of a query asked by the user. These terms can be images or keywords, for example.
  - **Mapping**: Solr maps the user query to the documents stored in the database to find the appropriate result.
  - **Ranking** the outcome: as soon as the engine searches the indexed documents, it ranks the outputs as per their relevance.

Apache Solr

- Installation
  - https://solr.apache.org/guide/8_9/installing-solr.html
  - For MacOS, simply `brew install solr`.

- Run solr
  - `solr start`

- Create a simple solr core
  - `solr create -c simple`

A simple indexer

```
1   def __init__(self):
2       self.crawled_folder = Path(__file__).parent / '../crawled/'
3       with open(self.crawled_folder / 'url_list.pickle', 'rb') as f:
4           self.file_mapper = pickle.load(f)
5       self.solr = pysolr.Solr('http://localhost:8983/solr/simple', always_commit=True, timeout = 10)
```

The folder storing the crawled data

Connecting to the Solr client

```
1   if __name__ == '__main__':
2       s = Indexer()
3       s.run_indexer()
4       results = s.solr.search('text:camt')
5       for result in results:
6           print("The title is '{0} ({1})'.".format(result['title'], result['url']))
7
```

A query example searching the text field

A simple indexer

```
1   def run_indexer(self):
2       self.solr.delete(q='*:*')
3       for file in os.listdir(self.crawled_folder):
4           if file.endswith(".txt"):
5               j = json.load(open(os.path.join(self.crawled_folder, file)))
6               j['id'] = j['url']
7               print(j)
8               self.solr.add(j)
```

For each file, passing it to solr

Search result example

```
The title is '['หน้าหลัก'] (['https://camt.cmu.ac.th/'])'.
The title is '['หน้าหลัก'] (['https://camt.cmu.ac.th/index.php/th/'
The title is '['Announcement'] (['https://camt.cmu.ac.th/index.ph
The title is '['Home'] (['https://camt.cmu.ac.th/index.php/en/'])
The title is '['ระบบจองห้อง/รถตู้'] (['https://meetingroom.camt.cmu.a
The title is '['ข่าวทั่วไป'] (['https://camt.cmu.ac.th/index.php/en/
The title is '['ปฐมนิเทศโครงการ Gifted School & Pre College CAMT 20
The title is '['CAMT::College of Arts, Media and Technology'] (['
The title is '['CAMT Orientation "ปฐมนิเทศนักศึกษาใหม่" ประจำปีการศึกษา 2
The title is '['CAMT ผนึกกำลัง Mass com  ร่วมกันสร้างหลักสูตรภาพยนตร์ดิจิทัล เ
```

Configuring Solr

org.apache.solr.search.similarities

Class ClassicSimilarityFactory

java.lang.Object
    org.apache.solr.schema.SimilarityFactory
        org.apache.solr.search.similarities.ClassicSimilarityFactory

Direct Known Subclasses:

`SweetSpotSimilarityFactory`

```
public class ClassicSimilarityFactory
extends SimilarityFactory
```

Factory for `ClassicSimilarity`

ClassicSimilarity is Lucene's original scoring implementation, based upon the Vector Space Model.

Optional settings:

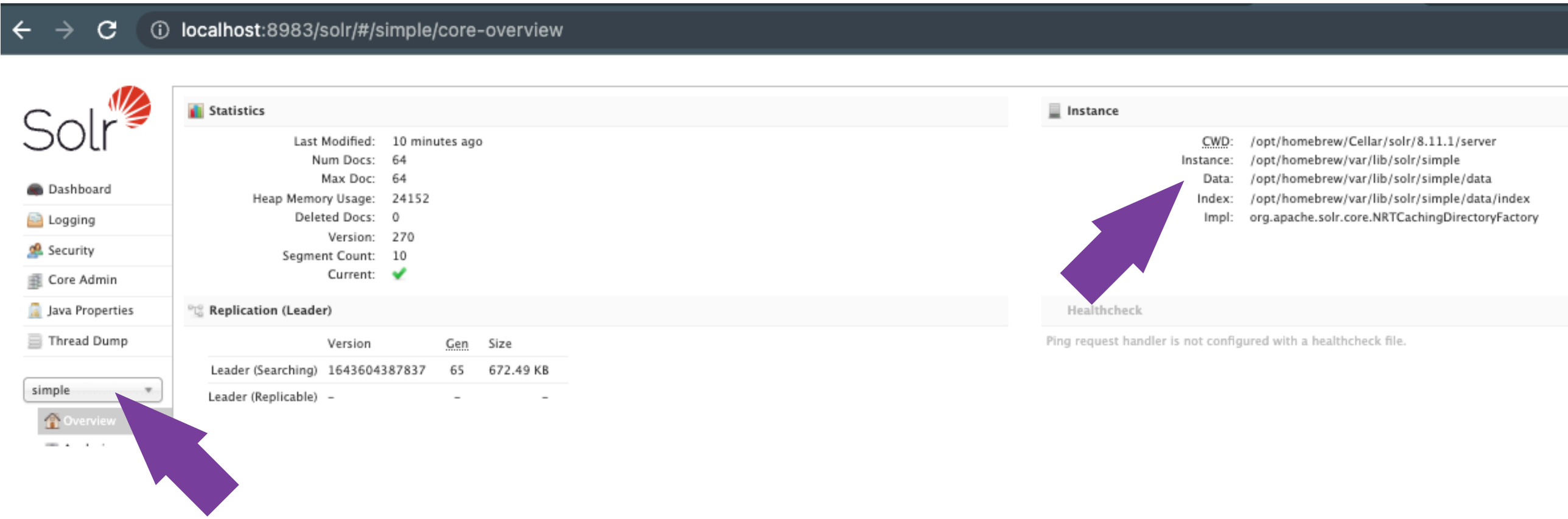
- discountOverlaps (bool): Sets `TFIDFSimilarity.setDiscountOverlaps(boolean)`

See Also:

`TFIDFSimilarity`

- Then, adding <similarity class="solr.ClassicSimilarityFactory" /> to the file named **managed-schema.xml** located under your simple solr core folder.

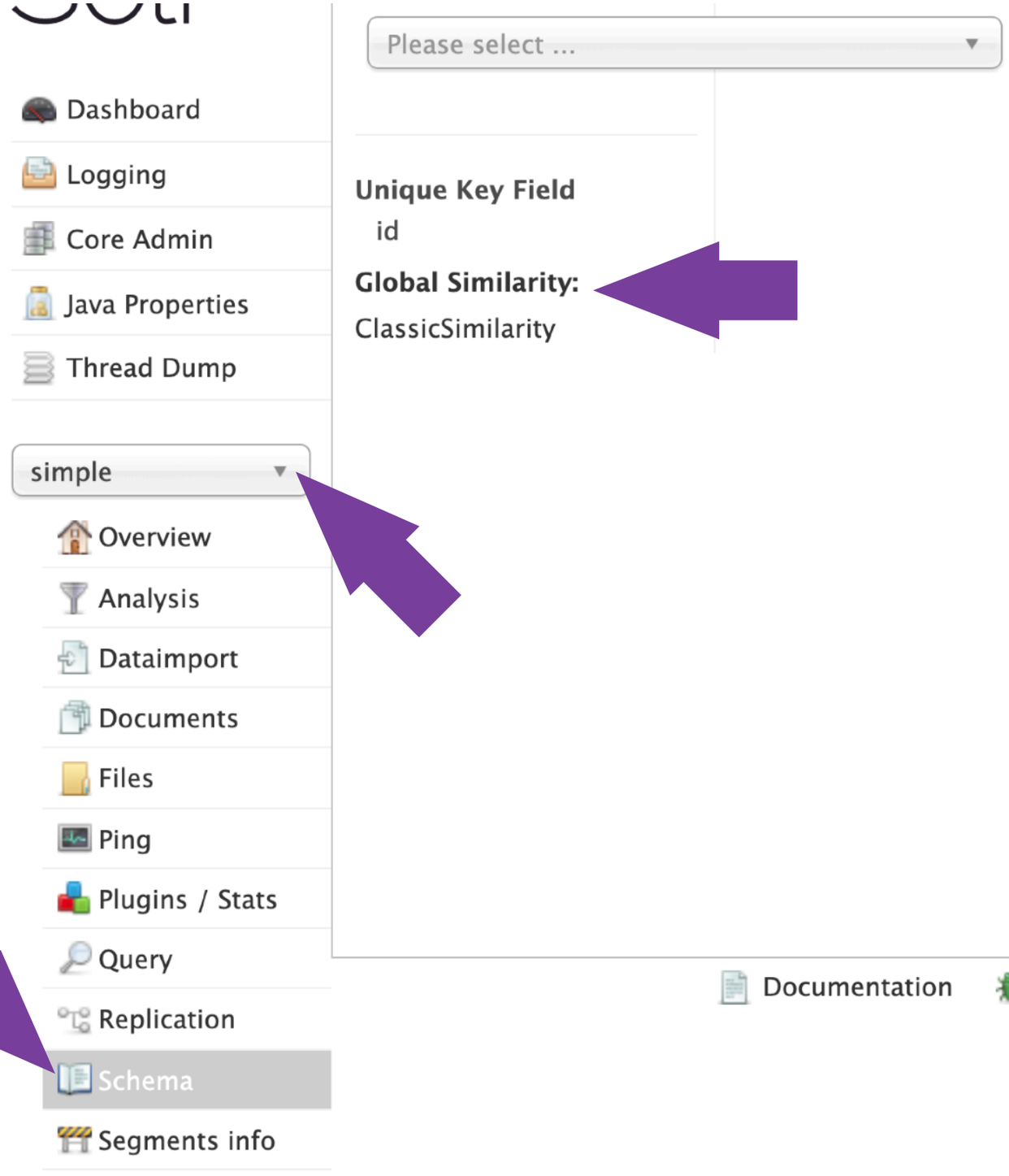
Solr core folder



- Then, adding <similarity class="solr.ClassicSimilarityFactory" /> to the file named **managed-schema.xml** located under your simple solr core folder.

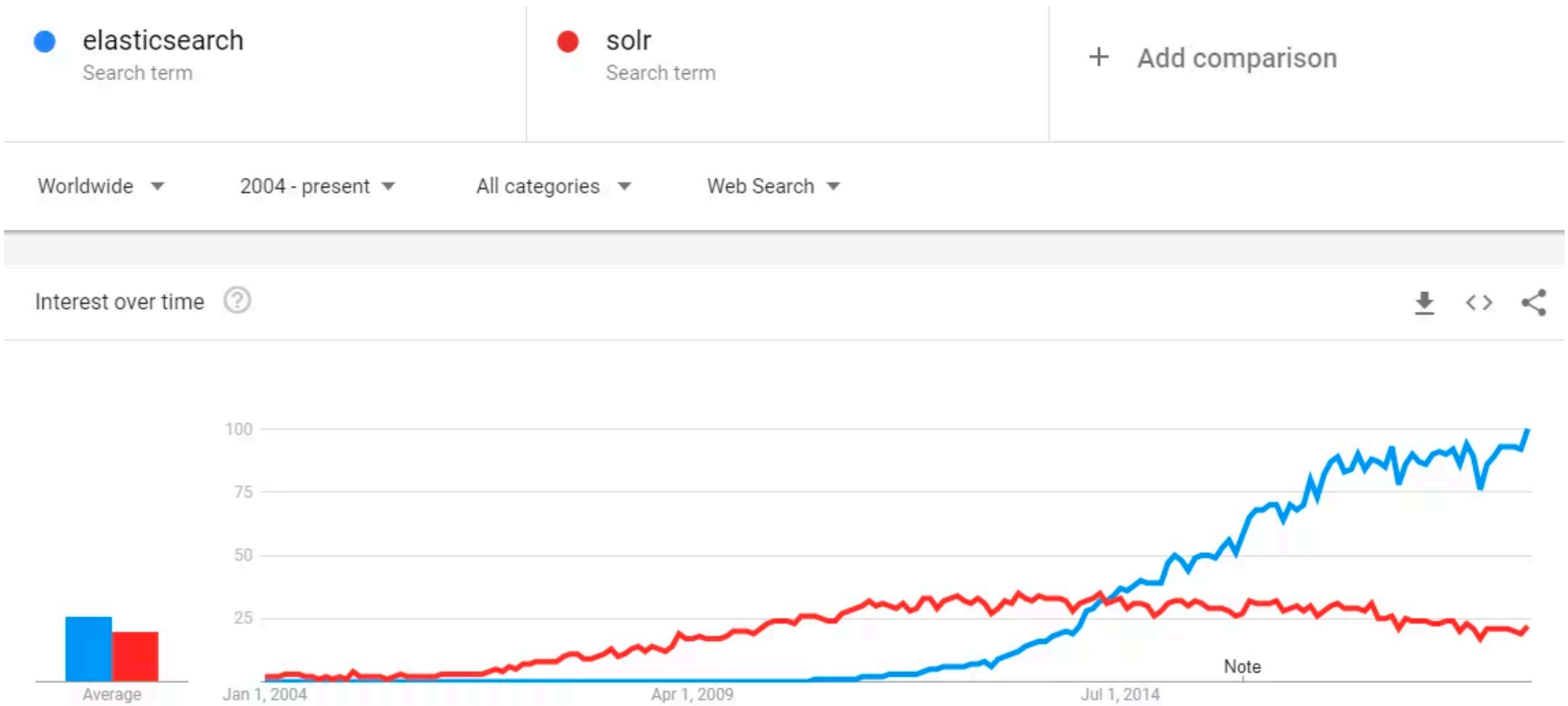
Configuring Solr

- Executing `solr restart` in the shell and check if the update is complete.



A simple web indexer

- Elastic search



Ref: https://sematext.com/blog/solr-vs-elasticsearch-differences/

Elasticsearch

QUERY & ANALYZE

Ask your data questions of all kinds

Search your way

Elasticsearch lets you perform and combine many types of searches — structured, unstructured, geo, metric — any way you want. Start simple with one question and see where it takes you.

Analyze at scale

It's one thing to find the 10 best documents to match your query. But how do you make sense of, say, a billion log lines? Elasticsearch aggregations let you zoom out to explore trends and patterns in your data.

SPEED

Elasticsearch is fast. Really, really fast.

Rapid results

When you get answers instantly, your relationship with your data changes. You can afford to iterate and cover more ground.

Powerful design

Being this fast isn't easy. We've implemented inverted indices with finite state transducers for full-text querying, BKD trees for storing numeric and geo data, and a column store for analytics.

All-inclusive

And since everything is indexed, you're never left with index envy. You can leverage and access all of your data at ludicrously awesome speeds.

Ref: https://www.elastic.co/elasticsearch/

Elasticsearch

- Installation
  - https://www.elastic.co/guide/en/elasticsearch/reference/current/install-elasticsearch.html
  - For MacOS, simply `brew tap elastic/tap` then `brew install elastic/tap/elasticsearch-full`.
- Run elasticsearch
  - `elasticsearch`

A simple indexer (using elasticsearch)

The folder storing the crawled data

```
1   def __init__(self):
2       self.crawled_folder = Path(__file__).parent / '../crawled/'
3       with open(self.crawled_folder / 'url_list.pickle', 'rb') as f:
4           self.file_mapper = pickle.load(f)
5       self.es_client = Elasticsearch("localhost:9200",http_auth=["elastic", "changeme"],)
```

Connecting to the elasticsearch client

A query example searching the text field

```
1   if __name__ == '__main__':
2       s = Indexer()
3       s.run_indexer()
4       query = {'query': {'bool': {'must': [{'match': {'text': 'camt'}}]}}}
5       results = s.es_client.search(index='simple', body=query)
6       print("Got %d Hits:" % results['hits']['total']['value'])
7       for hit in results['hits']['hits']:
8           print("The title is '{0} ({1})'.".format(hit["_source"]['title'], hit["_source"]['url']))
9
```

A simple indexer (using elasticsearch)

```
01  def run_indexer(self):
02      self.es_client.indices.create(index='simple', ignore=400)
03      self.es_client.indices.delete(index='simple',  ignore=[400, 404])
04
05      for file in os.listdir(self.crawled_folder):
06          if file.endswith(".txt"):
07              j = json.load(open(os.path.join(self.crawled_folder, file)))
08              j['id'] = j['url']
09              print(j)
10              self.es_client.index(index='simple', body=j)
```

For each file, passing it to elasticsearch

Search result example

```
The title is 'หน้าหลัก (https://camt.cmu.ac.th/)'.
The title is 'หน้าหลัก (https://camt.cmu.ac.th/index.php/th/)'.
The title is 'Announcement (https://camt.cmu.ac.th/index.php/en/2-uncategori
The title is 'Home (https://camt.cmu.ac.th/index.php/en/)'.
The title is 'ระบบจองห้อง/รถตู้ (https://meetingroom.camt.cmu.ac.th/)'.
The title is 'ข่าวทั่วไป (https://camt.cmu.ac.th/index.php/en/all-news-groups/2
The title is 'ปฐมนิเทศโครงการ Gifted School & Pre College CAMT 2021 (https://c
The title is 'CAMT::College of Arts, Media and Technology (https://service.c
The title is 'CAMT Orientation "ปฐมนิเทศนักศึกษาใหม่" ประจำปีการศึกษา 2564 (https://
The title is 'CAMT ผนึกกำลัง Mass com  ร่วมกันสร้างหลักสูตรภาพยนตร์ดิจิทัล เข้าสู่ตลาดหนังระดับ
```

Configuring elasticsearch

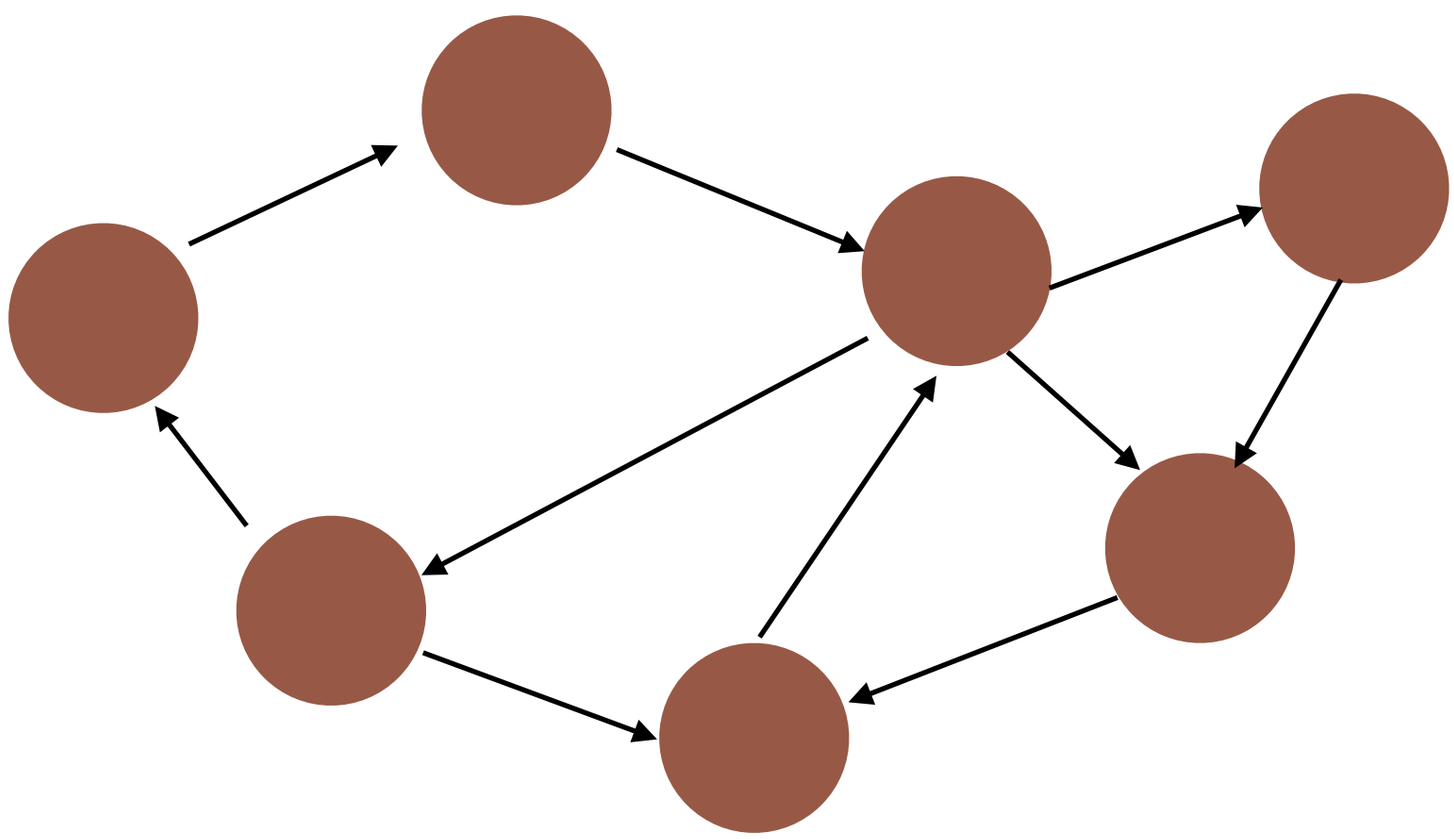
- Executing the three following commands in sequence in the shell

```
curl -X POST "localhost:9200/simple/_close"

curl -X PUT "localhost:9200/simple/_settings" -H 'Content-Type: application/json' -d'
{
  "index": {
    "similarity": {
      "default": {
        "type": "boolean"
      }
    }
  }
}'


curl -X POST "localhost:9200/simple/_open"
```

Link analysis

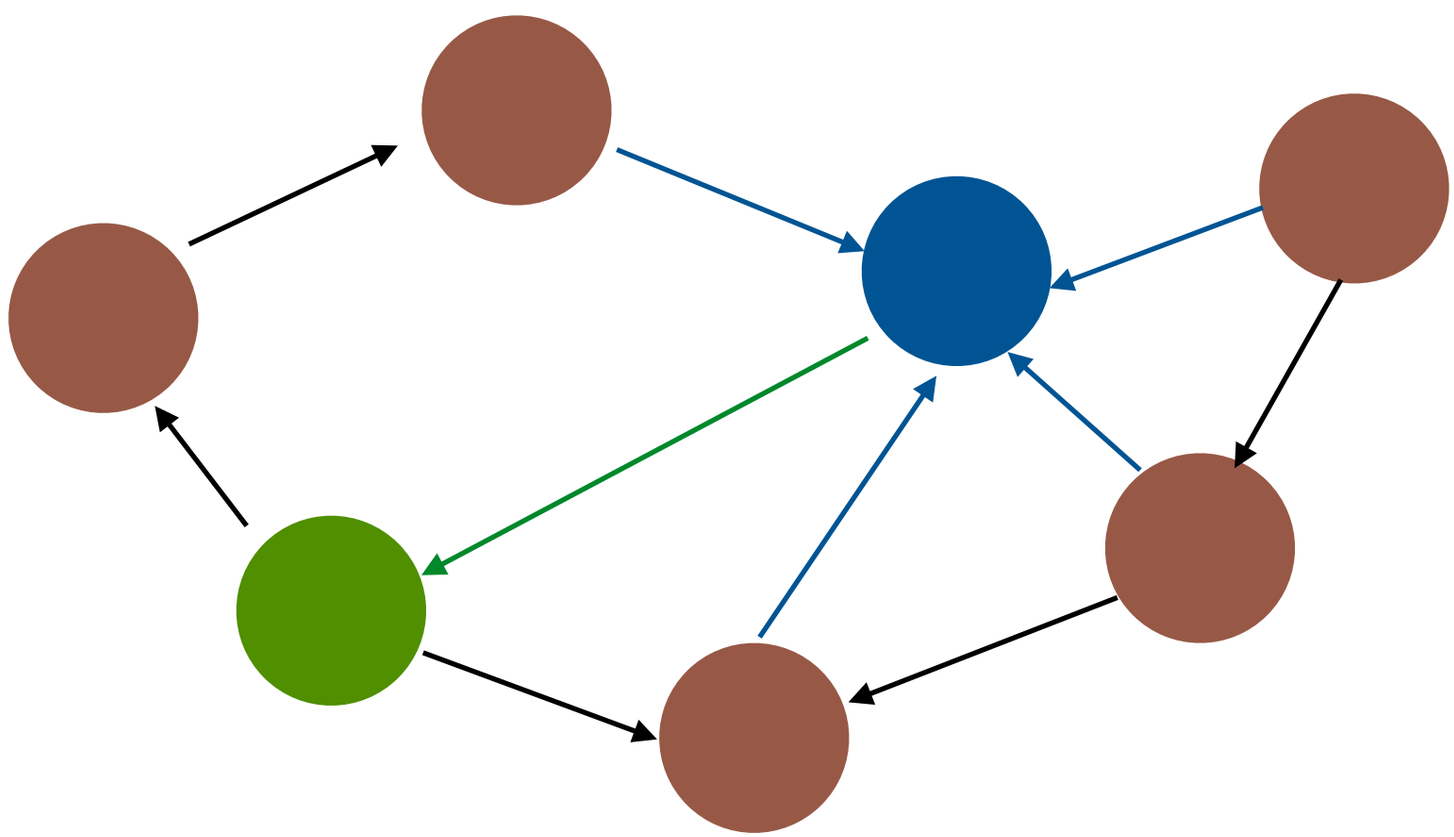
Link analysis

- Aside the textual properties, authority also appears to be an important information source for ranking.
  - Who is saying about whom



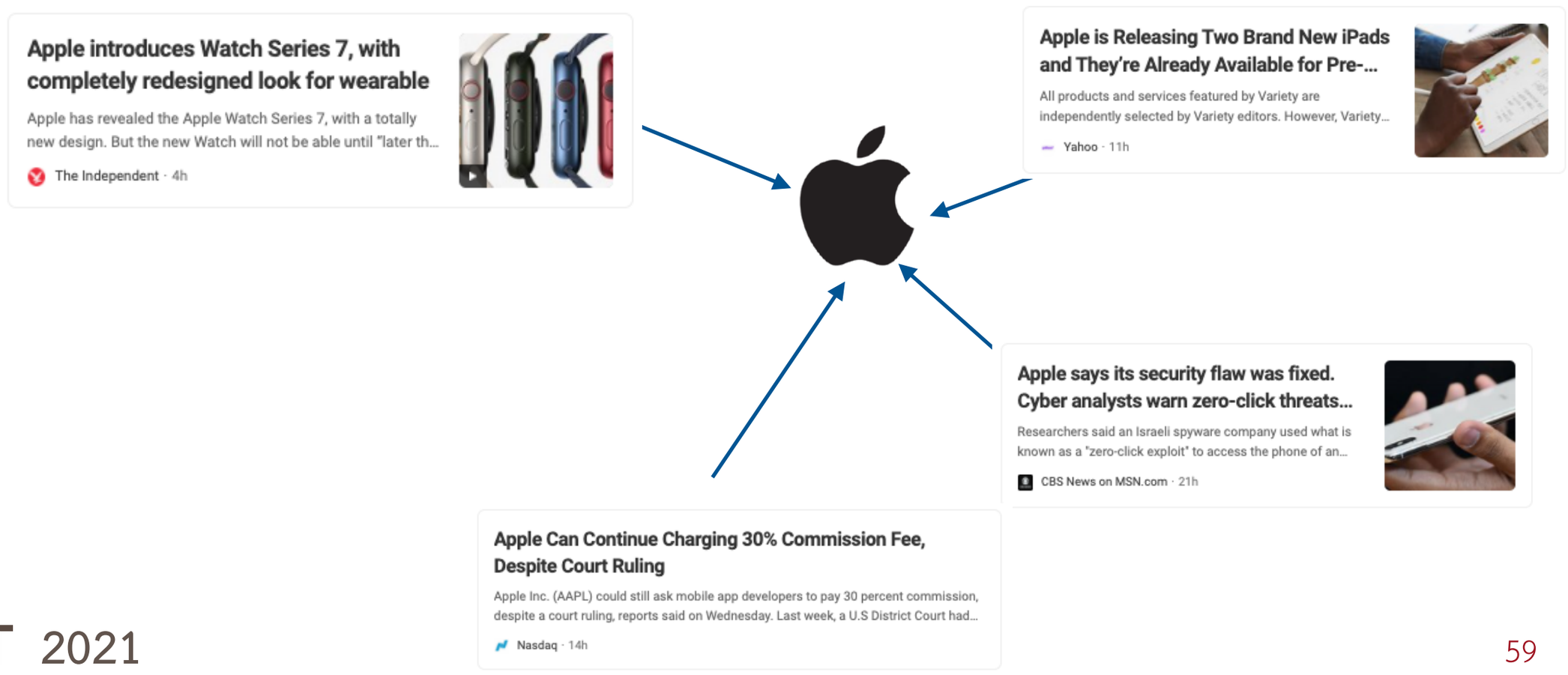
Link

- Powerful sources of authenticity and authority
  - If a big name refers to you, you should deserve an attention.
  - If a big name is whom so many people referring to.
  - If a bad guy refers to you, you should also deserve an attention.

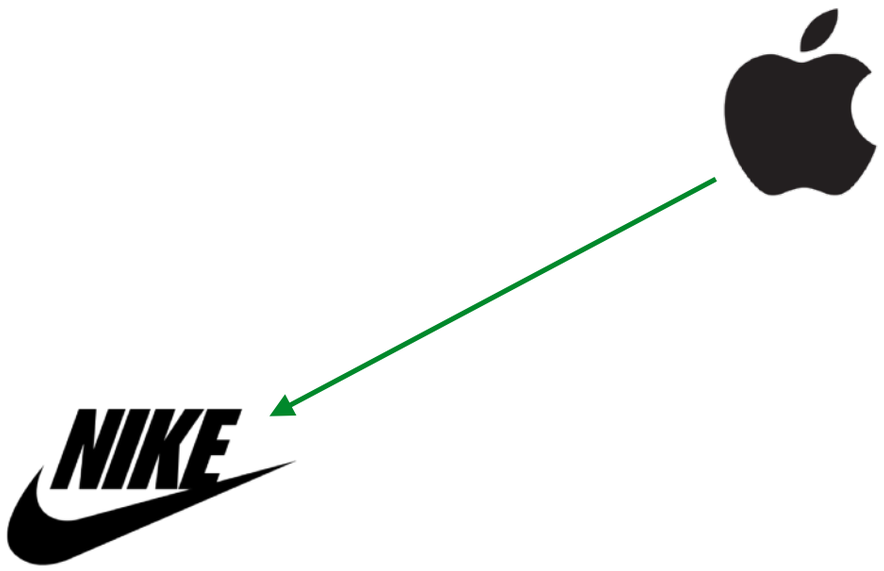


Link analysis

- We extract anchor text
  - If a big name refers to you, you should deserve an attention.
  - If a big name is whom so many people referring to.
  - If a bad guy refers to you, you should also deserve an attention.



Typically, big names point to big names, e.g.,

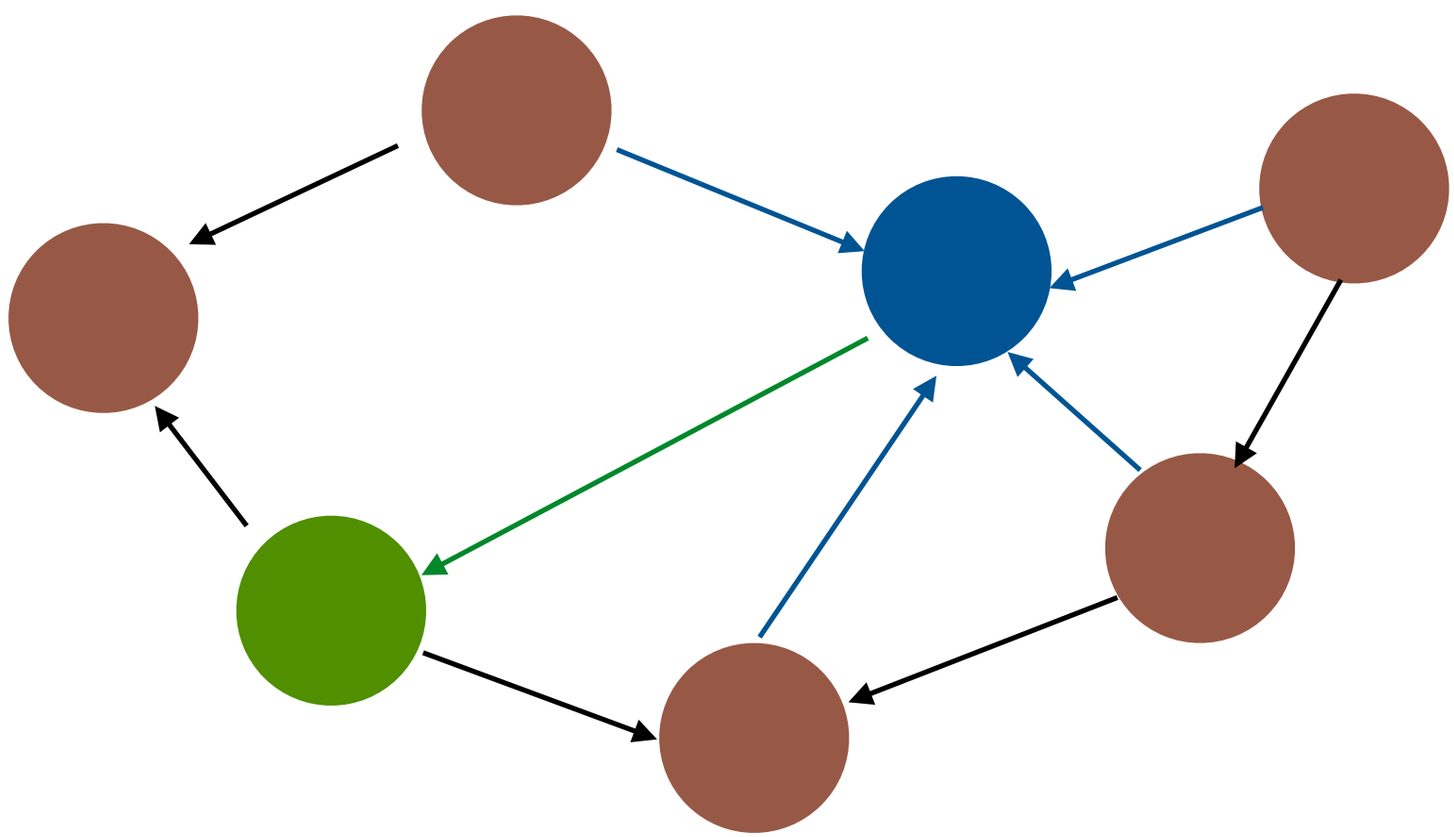


What if?


CAMT
College of Arts, Media and Technology
Chiang Mai University

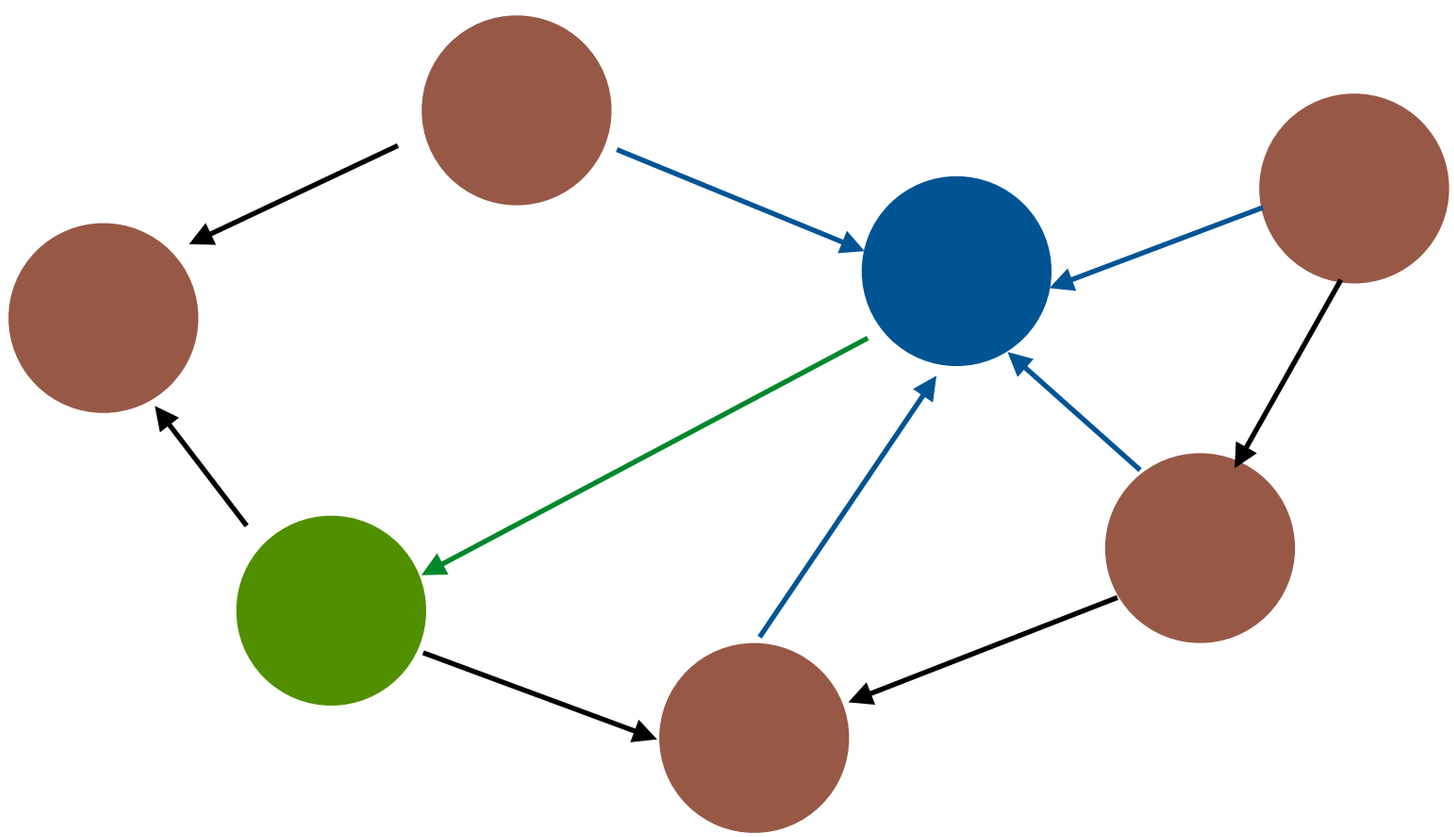
Challenge

- We don’t know how much score to give to a page until we fully analyze all the links.
  - E.g., We will know how much blue should give to green after we calculate the score for all the page pointing to blue.



Challenge

- We don’t know how much score to give to a page until we fully analyze all the links.
  - Unless we have a sensible approach, this will introduce a deadlock.

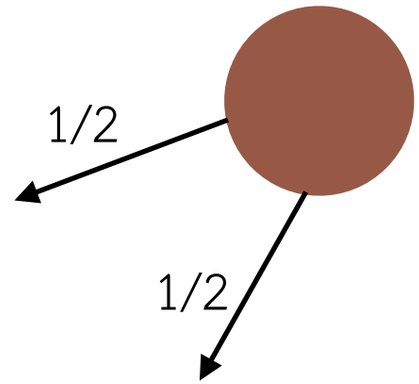


Pagerank scoring

- Pagerank attempts to solve this problem in an intelligent way.
  - Represent the entire system as probabilistic model.
  - Use iterative algorithm — iteratively propagate the scores of each webpage to the others, until the scoring of the entire web graph is converged

Pagerank scoring

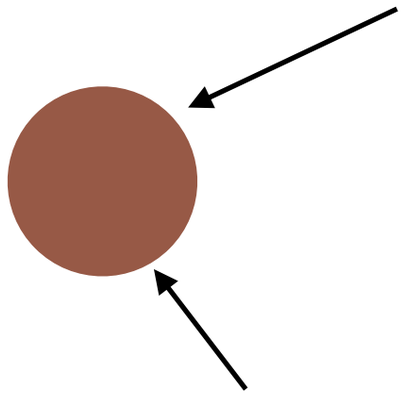
- The original Pagerank assumes that it is an equal probability a user will click a link over all the other links in a page, e.g.,



- Propagation of these probability will eventually converged.

Pagerank scoring

- Web pages are full of dead-ends, e.g.,



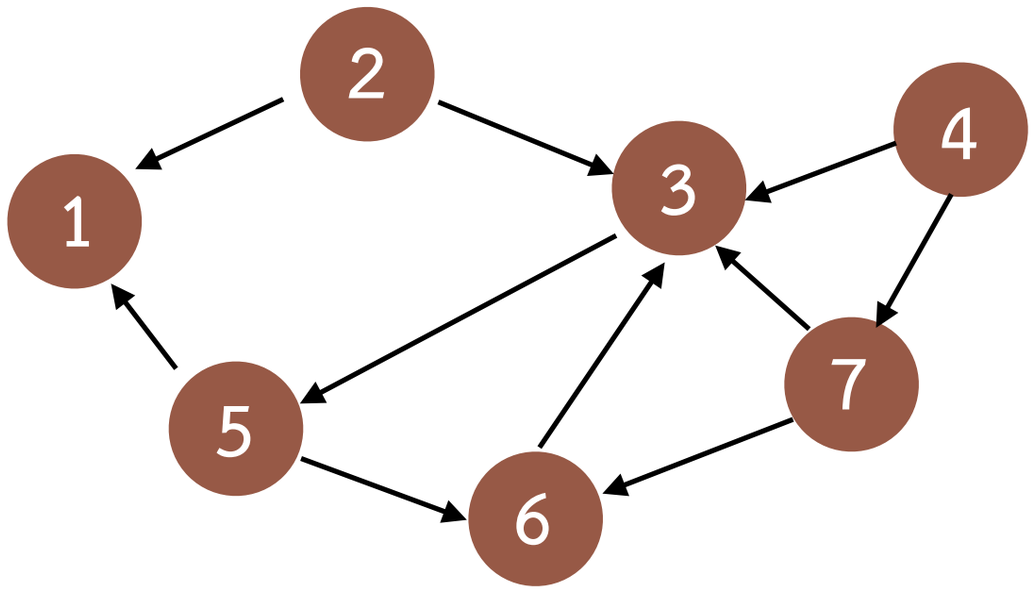
- These webs will not further propagate their scores

Teleporting

- Attempting to solve the dead-ends, by mimicking that user can simply put a url to any web and instantly leave a page

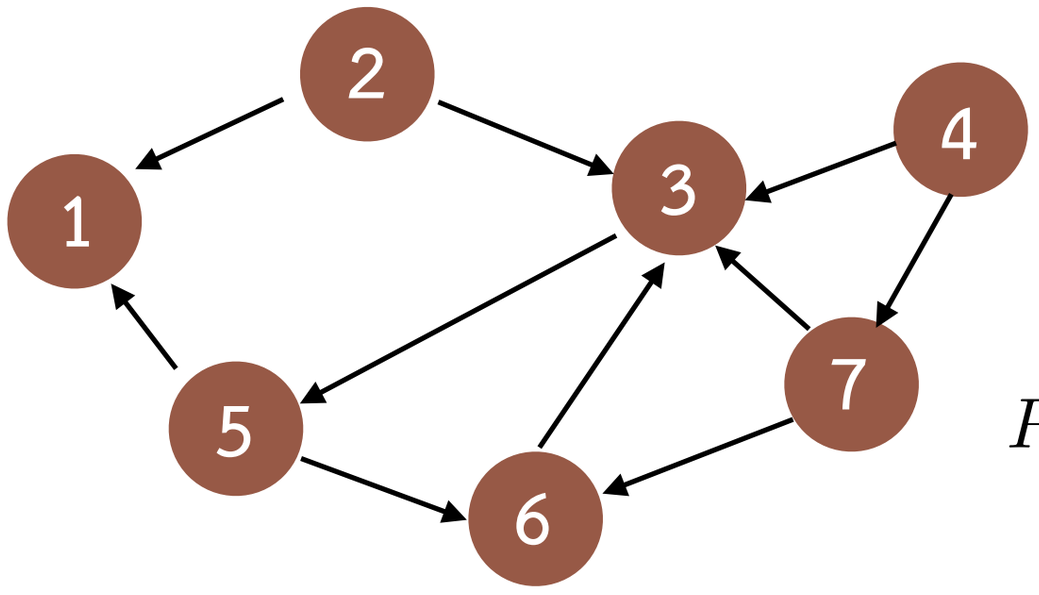
- In the original Pagerank paper, the link analysis score takes 85%, then the other 15% propagates to all the pages in the system.

Example



$$P = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{array}{c} \begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1/2 & 0 & 0 & 0 & 1/2 \\ 1/2 & 0 & 0 & 0 & 0 & 1/2 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/2 & 0 & 0 & 1/2 & 0 \end{pmatrix} \end{array}$$

Fixing teleporting



$$P' = \alpha P + (1-\alpha)ee^T/n, \text{where } \alpha = 0.85$$

$$P' = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{array}{c} \begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\ \begin{pmatrix} 1/7 & 1/7 & 1/7 & 1/7 & 1/7 & 1/7 & 1/7 \\ 25/56 & 3/140 & 25/56 & 3/140 & 3/140 & 3/140 & 3/140 \\ 3/140 & 3/140 & 3/140 & 3/140 & 61/70 & 3/140 & 3/140 \\ 3/140 & 3/140 & 25/56 & 3/140 & 3/140 & 3/140 & 25/56 \\ 25/56 & 3/140 & 3/140 & 3/140 & 3/140 & 25/56 & 3/140 \\ 3/140 & 3/140 & 61/70 & 3/140 & 3/140 & 3/140 & 3/140 \\ 3/140 & 3/140 & 25/56 & 3/140 & 3/140 & 25/56 & 3/140 \end{pmatrix} \end{array}$$

Power iteration

```
01  x0 = np.matrix([1/7] * 7)
02  P = np.matrix([
03                  [1/7, 1/7, 1/7, 1/7, 1/7, 1/7, 1/7],
04                  [25/56, 3/140, 25/56, 3/140, 3/140, 3/140, 3/140],
05                  [3/140, 3/140, 3/140, 3/140, 61/70, 3/140, 3/140],
06                  [3/140, 3/140, 25/56, 3/140, 3/140, 3/140, 25/56],
07                  [25/56, 3/140, 3/140, 3/140, 3/140, 25/56, 3/140],
08                  [3/140, 3/140, 61/70, 3/140, 3/140, 3/140, 3/140],
09                  [3/140, 3/140, 25/56, 3/140, 3/140, 25/56, 3/140],
10                ])
11  print(x0*P)
12  print(x0 * P * P)
13  print(x0 * P * P * P)
```

Power iteration

```
01  x0 = np.matrix([1/7] * 7)
02  P = np.matrix([
03                  [1/7, 1/7, 1/7, 1/7, 1/7, 1/7, 1/7],
04                  [25/56, 3/140, 25/56, 3/140, 3/140, 3/140, 3/140],
05                  [3/140, 3/140, 3/140, 3/140, 61/70, 3/140, 3/140],
06                  [3/140, 3/140, 25/56, 3/140, 3/140, 3/140, 25/56],
07                  [25/56, 3/140, 3/140, 3/140, 3/140, 25/56, 3/140],
08                  [3/140, 3/140, 61/70, 3/140, 3/140, 3/140, 3/140],
09                  [3/140, 3/140, 25/56, 3/140, 3/140, 25/56, 3/140],
10              ])
11
12  prev_Px = x0
13  Px = x0*P
14  i=0
15  while(any(abs(np.asarray(prev_Px).flatten()-np.asarray(Px).flatten()) > 1e-8)):
16      i+=1
17      prev_Px = Px
18      Px = Px * P
19
20  print('Converged in {0} iterations: {1}'.format(i, np.asarray(Px).flatten()))
```

PageRank score for the crawled webpage

```
class Pr:

    def __init__(self, alpha):
        self.crawled_folder = Path(__file__).parent / '../crawled/'
        self.alpha = alpha

    def url_extractor(self):
        url_maps = {}
        all_urls = set([])

        for file in os.listdir(self.crawled_folder):
            if file.endswith(".txt"):
                j = json.load(open(os.path.join(self.crawled_folder, file)))
                all_urls.add(j['url'])
                for s in j['url_lists']:
                    all_urls.add(s)
                url_maps[j['url']] = list(set(j['url_lists']))
        return url_maps, all_urls
```

```
if __name__ == '__main__':
    s = Pr(alpha=0.85)
    s.pr_calc()
```

PageRank score for the crawled webpage

```
01   def pr_calc(self):
02       url_maps, all_urls = self.url_extractor()
03       url_matrix = pd.DataFrame(columns=all_urls,index=all_urls)
04
05       for url in url_maps:
06           if len(url_maps[url]) > 0 and len(all_urls) > 0:
07               url_matrix.loc[url] = (1 - self.alpha) * (1 / len(all_urls))
08               url_matrix.loc[url, url_maps[url]] = url_matrix.loc[url, url_maps[url]] + (self.alpha * (1/
len(url_maps[url])))
09
10       url_matrix.loc[url_matrix.isnull().all(axis=1), :] = (1 / len(all_urls))
11       # print(url_matrix.sum(1).values)
12
13       x0 = np.matrix([1 / len(all_urls)] * len(all_urls))
14       P = np.asmatrix(url_matrix.values)
15
16       prev_Px = x0
17       Px = x0 * P
18       i = 0
19       while (any(abs(np.asarray(prev_Px).flatten() - np.asarray(Px).flatten()) > 1e-8)):
20           i += 1
21           prev_Px = Px
22           Px = Px * P
23
24       print('Converged in {0} iterations: {1}'.format(i, np.around(np.asarray(Px).flatten().astype(float), 5)))
25
26       self.pr_result = pd.DataFrame(Px, columns=url_matrix.index, index=['score']).T.loc[list(url_maps.keys())]
```

PageRank score for the crawled webpage

```
print(s.pr_result.o_markdown())
```

Adding the Pagerank score to Solr

```
01  class Indexer:
02
03      def __init__(self):
04          self.crawled_folder = Path(__file__).parent / '../crawled/'
05          with open(self.crawled_folder / 'url_list.pickle', 'rb') as f:
06              self.file_mapper = pickle.load(f)
07          self.solr = pysolr.Solr('http://localhost:8983/solr/simple', always_commit=True, timeout = 10)
08          self.pr = Pr(alpha=0.85)
09          self.pr.pr_calc()
10
11      def run_indexer(self):
12          self.solr.delete(q='*:*')
13          for file in os.listdir(self.crawled_folder):
14              if file.endswith(".txt"):
15                  j = json.load(open(os.path.join(self.crawled_folder, file)))
16                  j['id'] = j['url']
17                  j['pagerank'] = self.pr.pr_result.loc[j['id']].score
18                  print(j)
19                  self.solr.add(j)
20
21  if __name__ == '__main__':
22      s = Indexer()
23      s.run_indexer()
24      results = s.solr.search('text:camt',**{'defType':'edismax',
'boost':'mul(query($q),field(pagerank,min))'})
25      for result in results:
26          print("The title is '{0} ({1})'.".format(result['title'], result['url']))
```

Calculate the score and the results are stored at self.pr.pr_result

Boosting the score for a certain document

Let the query aggregate the pagerank score and the query score to produce the final relevance score.

Adding the Pagerank score to Elasticsearch

```
01  class Indexer:
02
03      def __init__(self):
04          self.crawled_folder = Path(__file__).parent / '../crawled/'
05          with open(self.crawled_folder / 'url_list.pickle', 'rb') as f:
06              self.file_mapper = pickle.load(f)
07          self.es_client = Elasticsearch("localhost:9200",http_auth=["elastic", "changeme"],)
08          self.pr = Pr(alpha=0.85)
09          self.pr.pr_calc()
10
11      def run_indexer(self):
12          self.es_client.indices.create(index='simple', ignore=400)
13          self.es_client.indices.delete(index='simple',  ignore=[400, 404])
14
15          for file in os.listdir(self.crawled_folder):
16              if file.endswith(".txt"):
17                  j = json.load(open(os.path.join(self.crawled_folder, file)))
18                  j['id'] = j['url']
19                  j['pagerank'] = self.pr.pr_result.loc[j['id']].score
20                  print(j)
21                  self.es_client.index(index='simple', body=j)
```

Calculate the score and the results are stored at self.pr.pr_result

Adding a filed named ‘pagerank’ to the document

Adding the Pagerank score to Elasticsearch

```
01  if __name__ == '__main__':
02      s = Indexer()
03      s.run_indexer()
04      query = {"query": {
05              "function_score": {
06                  "query": {
07                      "match": {
08                          "text": "camt"
09                      }
10                  },
11                  "functions": [{
12                      "field_value_factor": {
13                          "field": "pagerank"
14                      }
15                  }],
16                  "score_mode": "multiply"
17              }
18          }
19      }
20      results = s.es_client.search(index='simple', body=query)
21      print("Got %d Hits:" % results['hits']['total']['value'])
22      for hit in results['hits']['hits']:
23          print("The title is '{0} ({1})'.".format(hit["_source"]['title'], hit["_source"]['url']))
24
```

Let the query aggregate the pagerank score and the query score to produce the final relevance score.

Time for questions