Information Retrieval & Social Web

CS 525/DS 595

Worcester Polytechnic Institute

Department of Computer Science

Instructor: Prof. Kyumin Lee

Previous Class…

Cosine Similarity

Cosine: Example

term frequencies (counts)

How similar are these novels?
SaS: Sense and Sensibility
PaP: Pride and Prejudice
WH: Wuthering Heights

term	SaS	PaP	WH
AFFECTION	115	58	20
JEALOUS	10	7	11
GOSSIP	2	0	6
WUTHERING	0	0	38

Cosine: Example

term frequencies (counts)

term	SaS	PaP	WH
AFFECTION	115	58	20
JEALOUS	10	7	11
GOSSIP	2	0	6
WUTHERING	0	0	38

log frequency weighting

term	SaS	PaP	WH
AFFECTION	3.06	2.76	2.30
JEALOUS	2.0	1.85	2.04
GOSSIP	1.30	0	1.78
WUTHERING	0	0	2.58

(To simplify this example, we don't do idf weighting.)

Cosine: Example

log frequency weighting

term	SaS	PaP	WH
AFFECTION	3.06	2.76	2.30
JEALOUS	2.0	1.85	2.04
GOSSIP	1.30	0	1.78
WUTHERING	0	0	2.58

log frequency weighting & cosine normalization

term	SaS	PaP	WH
AFFECTION	0.789	0.832	0.524
JEALOUS	0.515	0.555	0.465
GOSSIP	0.335	0.0	0.405
WUTHERING	0.0	0.0	0.588

- cos(SaS,PaP) ≈

Cosine: Example

log frequency weighting

term	SaS	PaP	WH
AFFECTION	3.06	2.76	2.30
JEALOUS	2.0	1.85	2.04
GOSSIP	1.30	0	1.78
WUTHERING	0	0	2.58

log frequency weighting & cosine normalization

term	SaS	PaP	WH
AFFECTION	0.789	0.832	0.524
JEALOUS	0.515	0.555	0.465
GOSSIP	0.335	0.0	0.405
WUTHERING	0.0	0.0	0.588

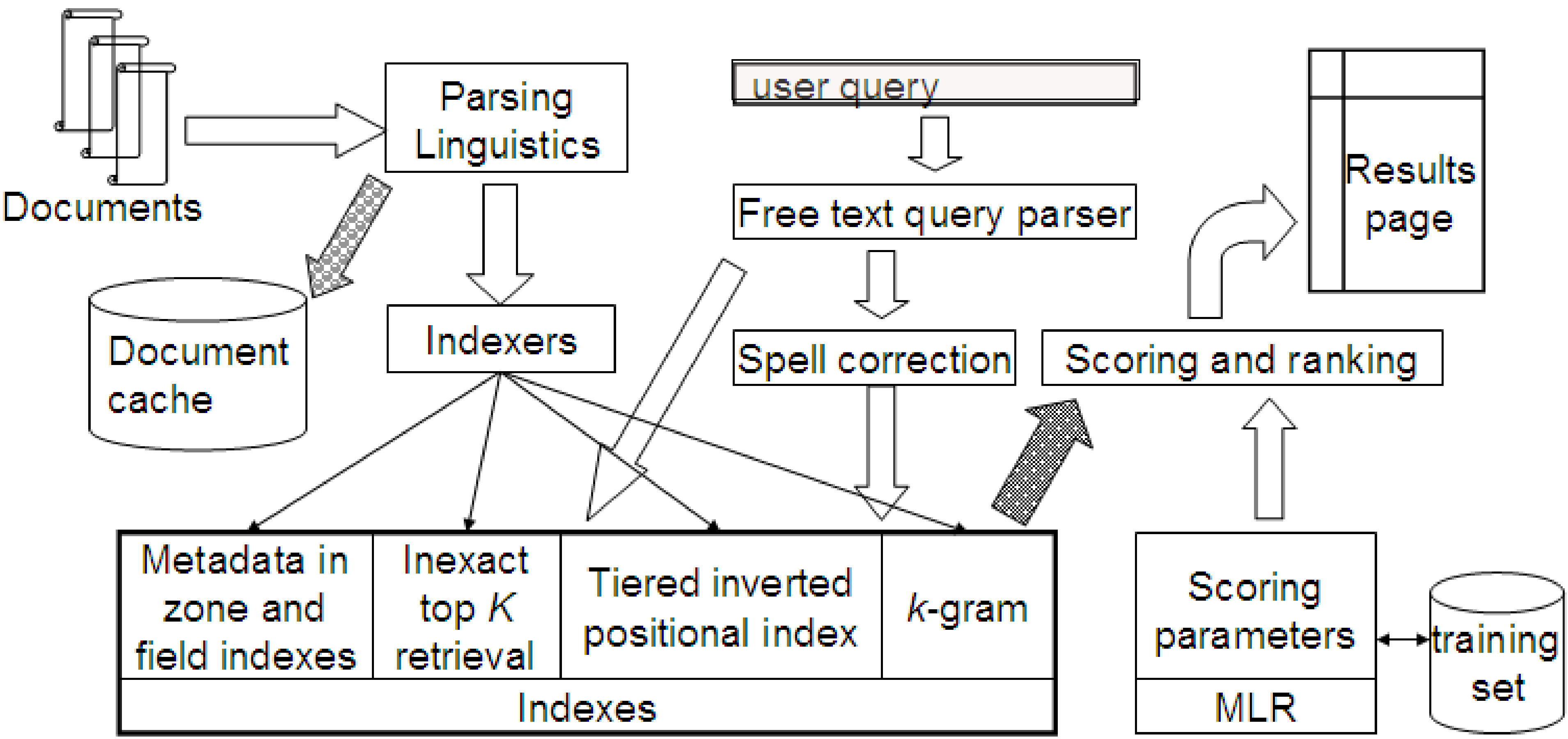
- cos(SaS,PaP) ≈ $0.789 * 0.832 + 0.515 * 0.555 + 0.335 * 0.0 + 0.0 * 0.0 \approx 0.94$.
- cos(SaS,WH) ≈ 0.79
- cos(PaP,WH) ≈ 0.69
- Why do we have cos(SaS,PaP) > cos(SAS,WH)?

Previous Class…

Cosine Similarity

Computing scores in a complete search system

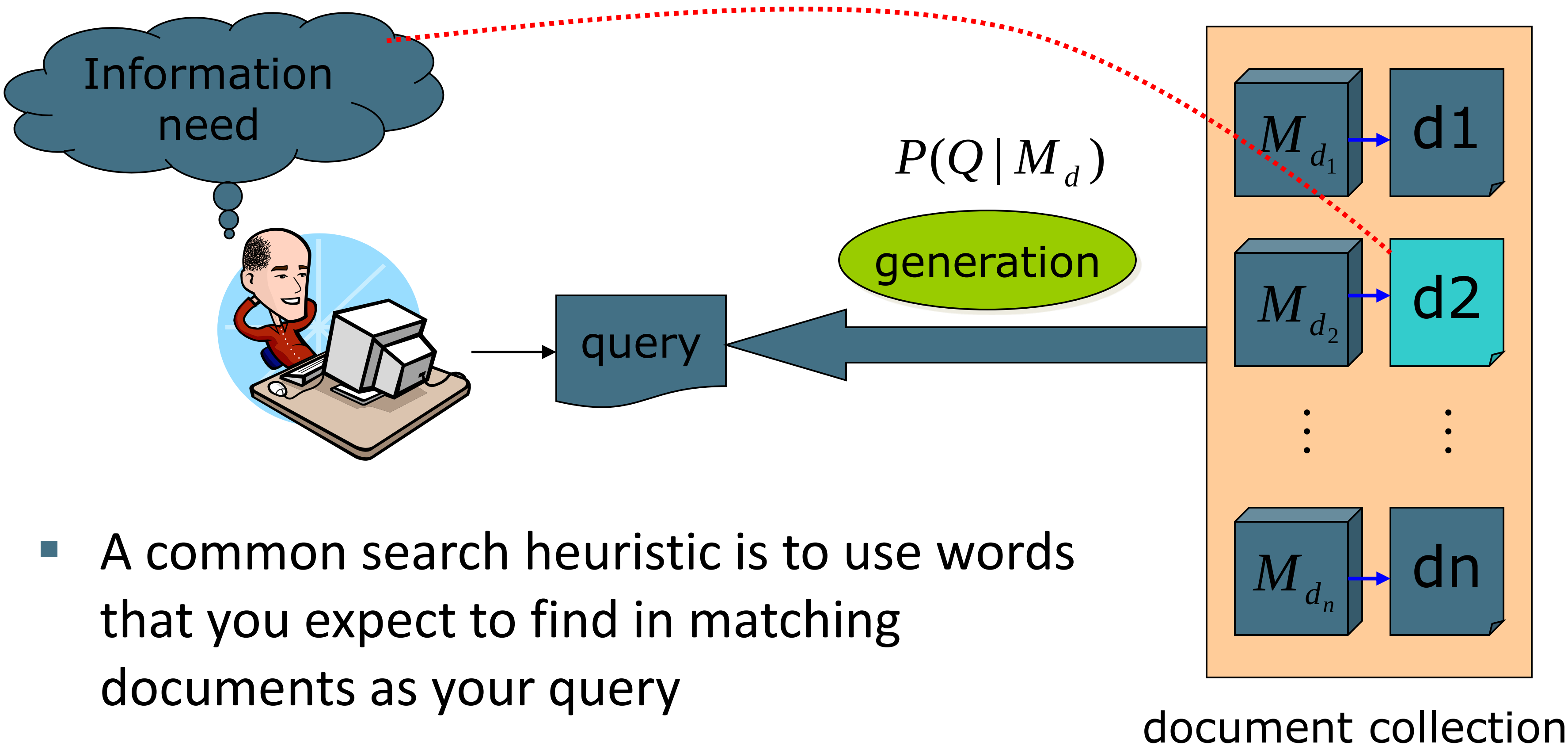
Putting it all together



Probabilistic IR

- Chapter 12
  - Statistical Language Models

IR based on Language Model (LM)



- A common search heuristic is to use words that you expect to find in matching documents as your query
- The LM approach directly exploits that idea!

Stochastic Language Models

- Models *probability* of generating strings in the language

Model M

0.2	the
0.1	a
0.01	man
0.01	woman
0.03	said
0.02	likes
…	

the	man	likes	the	woman
0.2	0.01	0.02	0.2	0.01

multiply

P(s | M) = 0.00000008

Stochastic Language Models

- Model *probability* of generating any string

Model M1	
0.2	the
0.01	class
0.0001	sayst
0.0001	pleaseth
0.0001	yon
0.0005	maiden
0.01	woman

Model M2	
0.2	the
0.0001	class
0.03	sayst
0.02	pleaseth
0.1	yon
0.01	maiden
0.0001	woman

the	class	pleaseth	yon	maiden
0.2	0.01	0.0001	0.0001	0.0005
0.2	0.0001	0.02	0.1	0.01

$$P(s|M2) > P(s|M1)$$

Using language models in IR

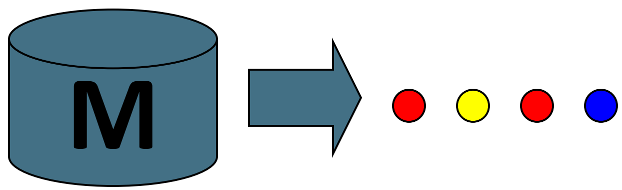
- Each document is treated as (the basis for) a language model.
- Given a query q
- Rank documents based on $P(d|q)$
- Bayes' rule

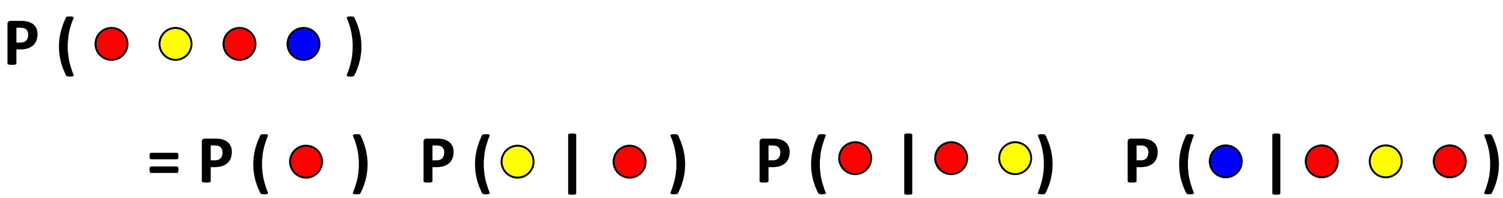
$$P(d|q) = \frac{P(q|d)P(d)}{P(q)}$$

- $P(q)$ is the same for all documents, so ignore
- $P(d)$ is the prior – often treated as the same for all d
  - But we can give a prior to "high-quality" documents, e.g., those with high PageRank.
- $P(q|d)$ is the probability of q given d.
- So to rank documents according to relevance to q, ranking according to $P(q|d)$ and $P(d|q)$ is equivalent.

Stochastic Language Models

- A statistical model for generating text
  - Probability distribution over a string/query in a given language





Unigram and higher-order models

P (🔴 🟡 🔴 🔵)

= P (🔴) P (🟡 | 🔴) P (🔴 | 🔴 🟡) P (🔵 | 🔴 🟡 🔴)

- Unigram Language Models

  P (🔴) P (🟡) P (🔴) P (🔵)

- Bigram (generally, *n*-gram) Language Models

  P (🔴) P (🟡 | 🔴) P (🔴 | 🟡) P (🔵 | 🔴)

- We use the unigram Language Models

Where we are

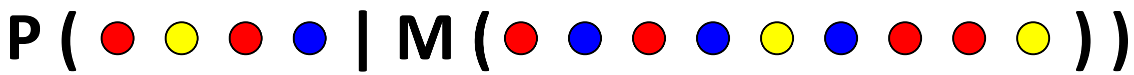
- In the LM approach to IR, we attempt to model the query generation process.
- Then we rank documents by the probability that a query would be observed as a random sample from the respective document model.
- That is, we rank according to $P(q|d)$.
- Next: how do we compute $P(q|d)$?

Retrieval based on probabilistic LM

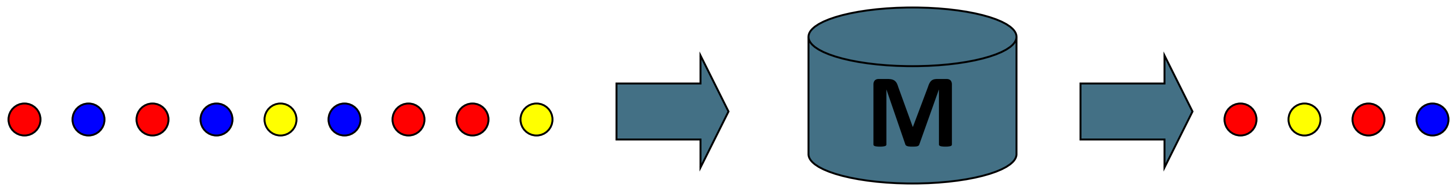
- Intuition
  - Users …
    - Have a reasonable idea of terms that are likely to occur in documents of interest.
    - They will choose query terms that distinguish these documents from others in the collection.
- Collection statistics …
  - Are integral parts of the language model.

The fundamental problem of LMs

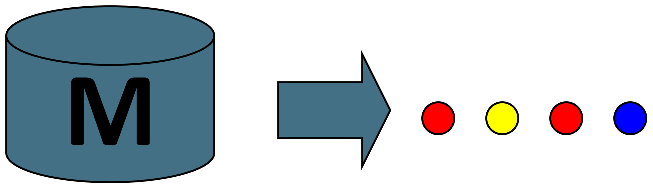
- Usually we don't know the model **M**
  - But have a sample of text representative of that model



- Estimate a language model from a sample
- Then compute the observation probability

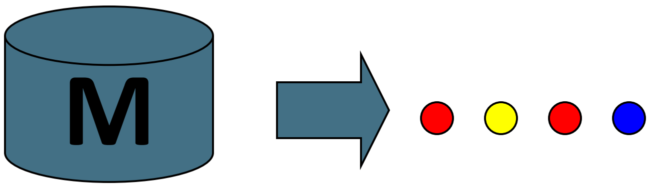


Example



- Doc 1 = “Today is a beautiful day.”
- p(today | M1) =
- p(is | M1) =
- p(a | M1) =
- p(beautiful | M1) =

Example



- Doc 2 = “Beautiful beautiful beautiful day!”
- p(today | M2) =
- p(is | M2) =
- p(a | M2) =
- p(beautiful | M2) =

Query generation probability

- Ranking formula

$$\hat{P}(q|M_d)$$

- The probability of producing the query given the language model of document *d* using *Maximum Likelihood Estimation* (MLE) is:
  - MLE means estimating a probability as the relative frequency. So this value makes the observed data maximally likely

$$\hat{P}(q|M_d) = \prod_{t \in q} \hat{P}_{\text{mle}}(t|M_d) = \prod_{t \in q} \frac{\text{tf}_{t,d}}{L_d}$$

Unigram assumption:
Given a particular language model, the query terms occur independently

M_d : language model of document d

$tf_{t,d}$: raw tf of term t in document d

L_d : total number of tokens in document d

Language Models (LMs)

- Unigram LM:
  - Bag-of-words model.
  - Multinomial distributions over words.

$$P(d) = \boxed{\frac{L_d!}{\mathrm{tf}_{t_1,d}!\mathrm{tf}_{t_2,d}!\cdots\mathrm{tf}_{t_M,d}!}} P(t_1)^{\mathrm{tf}_{t_1,d}} P(t_2)^{\mathrm{tf}_{t_2,d}} \cdots P(t_M)^{\mathrm{tf}_{t_M,d}}$$

multinomial coefficient, can leave out in practical calculations.

$$L_d = \sum_{1 \le i \le M} tf_{t_i,d}$$

The length of document d. M is the size of the vocabulary.

Query Likelihood Model

- Multinomial + Unigram:

$$P(q|M_d) = K_q \prod_{t \in V} P(t|M_d)^{\mathrm{tf}_{t,d}}$$

$$K_q = L_d! / (\mathrm{tf}_{t_1,d}! \mathrm{tf}_{t_2,d}! \cdots \mathrm{tf}_{t_M,d}!)$$

Multinomial coefficient for the query q.
Can be ignored.

- Retrieve based on a language model:
  - Infer a LM for each document.
  - Estimate $P(q|M_{di})$.
  - Rank the documents according to these probabilities.

Example

$$\hat{P}(q|M_d) = \prod_{t \in q} \hat{P}_{\text{mle}}(t|M_d) = \prod_{t \in q} \frac{\text{tf}_{t,d}}{L_d}$$

- Doc 1 = “Today is a beautiful day. “
- Doc 2 = “Beautiful beautiful beautiful day!“
- Query = “today beautiful”

Insufficient data

- Zero probability $\hat{P}(t|M_d) = 0$
  - May not wish to assign a probability of zero to a document that is missing one or more of the query terms

- General approach
  - A non-occurring term is possible, but no more likely than would be expected by chance in the collection.
  - If $tf_{(t,d)} = 0$, $\hat{P}(t|M_d) \le \mathrm{cf}_t / T$

cf_t : raw count of term t in the collection

T : raw collection size (total number of tokens in the collection)

Insufficient data

- We will use $\hat{P}(t|M_c)$ to "smooth" $P(t|d)$ away from zero.

- A simple idea that works well in practice is to use a mixture between the document multinomial and the collection multinomial distribution

Mixture model

$$\hat{P}(t|d) = \lambda \hat{P}_{\text{mle}}(t|M_d) + (1 - \lambda)\hat{P}_{\text{mle}}(t|M_c)$$

- Mixes the probability from the document with the general collection frequency of the word.

- High value of λ: “conjunctive-like” search – tends to retrieve documents containing all query words (suitable for short queries)
- Low value of λ: more disjunctive, suitable for long queries

- Correctly setting λ is very important for good performance.

Basic mixture model summary

$$P(q|d) \propto \prod_{1 \leq k \leq |q|} (\lambda P(t_k|M_d) + (1-\lambda)P(t_k|M_c))$$

individual-document model

general language model

General formulation of the LM for IR

Example

- Document collection (2 documents)
  - d_1: Xerox reports a profit but revenue is down
  - d_2: Lucent narrows quarter loss but revenue decreases further
- Model: MLE unigram from documents; $\lambda = \frac{1}{2}$
- Query: *revenue down*
  - $P(Q|d_1) =$
  - $P(Q|d_2) =$
-

Example

- Document collection (2 documents)
  - d_1: Xerox reports a profit but revenue is down
  - d_2: Lucent narrows quarter loss but revenue decreases further
- Model: MLE unigram from documents; $\lambda = ½$
- Query: *revenue down*
  - $P(Q|d_1) = [(1/8 + 2/16)/2] \times [(1/8 + 1/16)/2]$
    $= 1/8 \times 3/32 = 3/256$
  - $P(Q|d_2) = [(1/8 + 2/16)/2] \times [(0 + 1/16)/2]$
    $= 1/8 \times 1/32 = 1/256$
- Ranking: $d_1 > d_2$

Exercise

- Suppose, we've got 4 documents

DocID	Document text
1	click go the shears boys click click click
2	click click
3	metal here
4	metal shears click here

- Using the mixture model with lambda = 0.5, work out the per-doc probabilities for the query "click"

- collection model for “click” is
- collection model for “shears” is
- click in doc1:
- doc2:
- doc3:
- doc4:

- collection model for “click” is 7/16
- collection model for “shears” is 2/16
- click in doc1: 0.5 * 1/2 + 0.5 * 7/16 = 0.4688
- doc2: 0.7188
- doc3: 0.2188
- doc4: 0.3438

Exercise

- Suppose, we've got 4 documents

DocID	Document text
1	click go the shears boys click click click
2	click click
3	metal here
4	metal shears click here

- For the query "click shears", what's the ranking of the four documents?

click shears

- **Doc 4: 0.0645**
- Doc 1: 0.0586
- Doc 2: 0.0449
- Doc 3: 0.0137

Summary: LM

- LM approach assumes that documents and expressions of information problems are of the same type
- Computationally tractable, intuitively appealing

LMs vs. vector space model (1)

- LMs have some things in common with vector space models.
  - Term frequency is directed in the model.
    - But it is not scaled in LMs.
  - Probabilities are inherently “length-normalized”.
    - Cosine normalization does something similar for vector space.
  - Mixing document and collection frequencies has an effect similar to idf.
    - Terms rare in the general collection, but common in some documents will have a greater influence on the ranking.

LMs vs. vector space model (2)

- LMs vs. vector space model: differences
  - LMs: based on probability theory
  - Vector space: based on similarity, a geometric/ linear algebra notion
  - Collection frequency vs. document frequency
  - Details of term frequency, length normalization etc.

Language models for IR: Assumptions

- Simplifying assumption: Queries and documents are objects of same type. Not true!
  - There are other LMs for IR that do not make this assumption.
  - The vector space model makes the same assumption.
- Simplifying assumption: Terms are conditionally independent.
  - Again, vector space model makes the same assumption.
- Cleaner statement of assumptions than vector space
- Thus, better theoretical foundation than vector space
  - … but “pure” LMs perform much worse than “tuned” LMs.

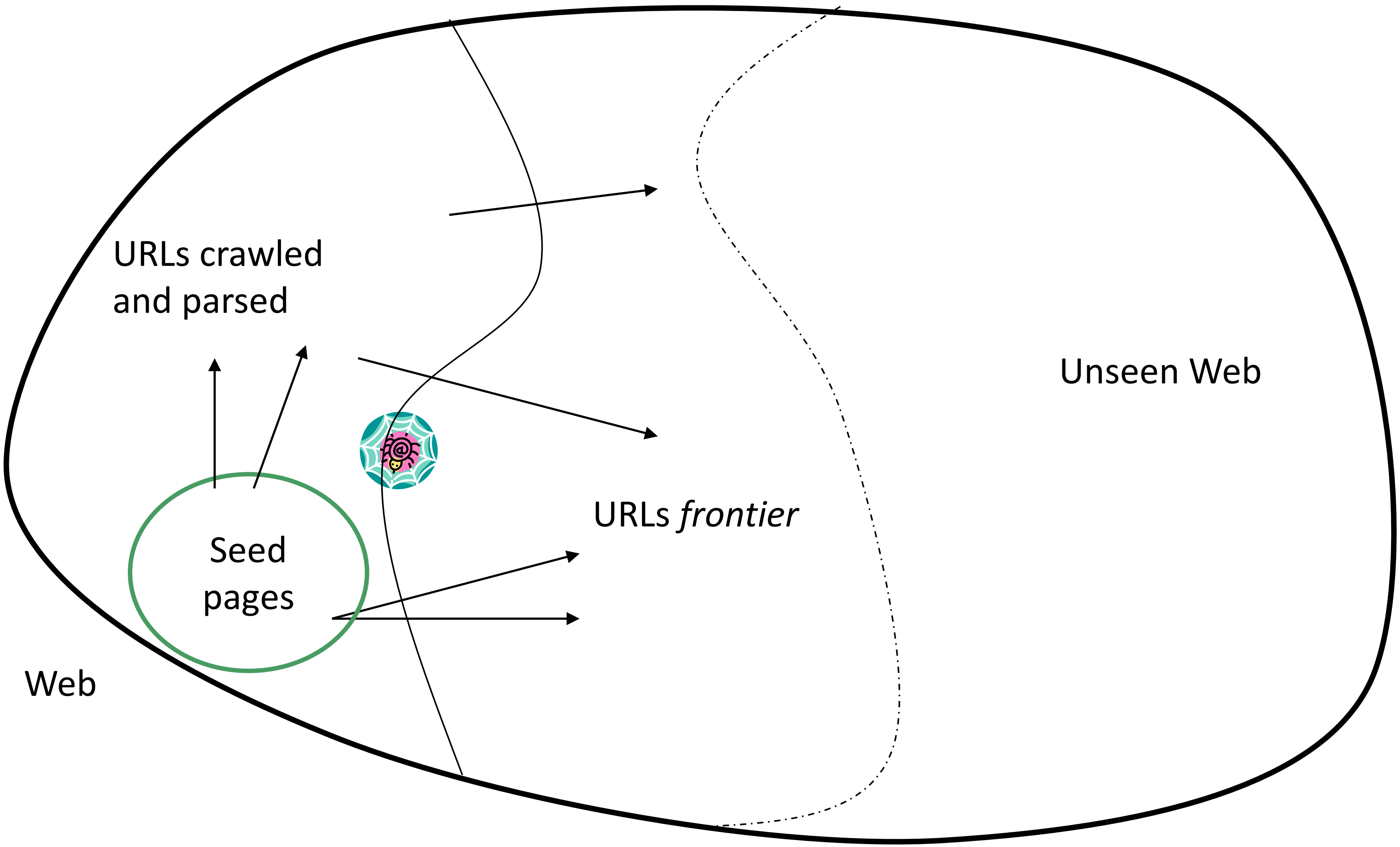
Next…

- Crawling
- Web APIs

Basic crawler operation

- Begin with known “seed” URLs
- Fetch and parse them
  - Extract URLs they point to
  - Place the extracted URLs on a queue
- Fetch each URL on the queue and repeat

Crawling picture



Simple picture – complications

- Web crawling isn't feasible with one machine
  - All of the above steps distributed
- Malicious pages
  - Spam pages
  - Spider traps – incl dynamically generated
- Even non-malicious pages pose challenges
  - Latency/bandwidth to remote servers vary
  - Webmasters' stipulations
    - How "deep" should you crawl a site's URL hierarchy?
  - Site mirrors and duplicate pages
- Politeness – don't hit a server too often

What any crawler *must* do

- Be Polite: Respect implicit and explicit politeness considerations
  - Only crawl allowed pages
  - Respect *robots.txt* (more on this shortly)

- Be Robust: Be immune to spider traps and other malicious behavior from web servers

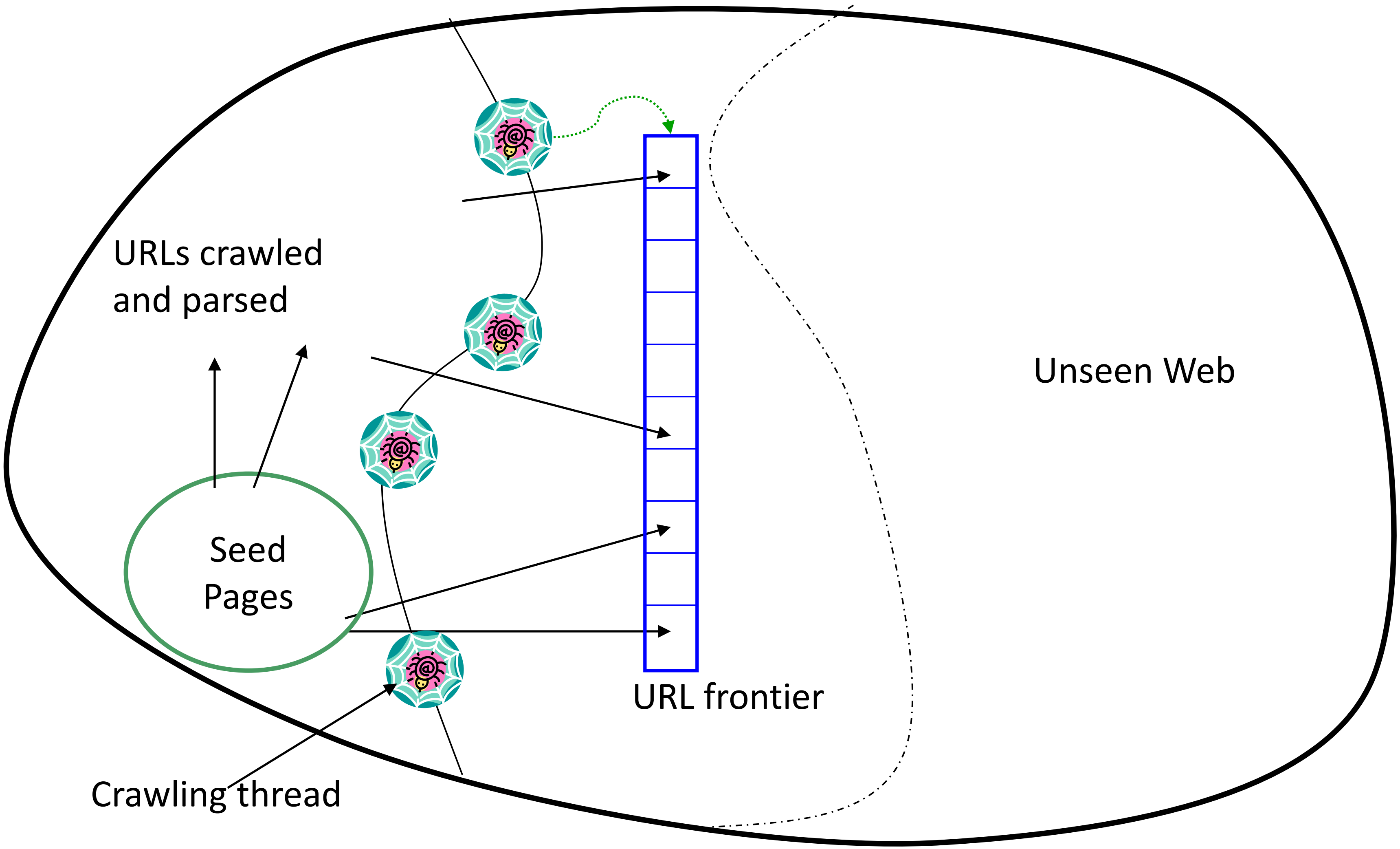
What any crawler *should* do

- Be capable of distributed operation: designed to run on multiple distributed machines
- Be scalable: designed to increase the crawl rate by adding more machines
- Performance/efficiency: permit full use of available processing and network resources

What any crawler *should* do

- Fetch pages of "higher quality" first
- Continuous operation: Continue fetching fresh copies of a previously fetched page
- Extensible: Adapt to new data formats, protocols

Updated crawling picture



URL frontier

- Can include multiple pages from the same host
- Must avoid trying to fetch them all at the same time
- Must try to keep all crawling threads busy

Explicit and implicit politeness

- 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
  - www.robotstxt.org/wc/norobots.html
- Website announces its request on what can(not) be crawled
  - For a server, create a file `/robots.txt`
  - This file specifies access restrictions

Robots.txt example

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

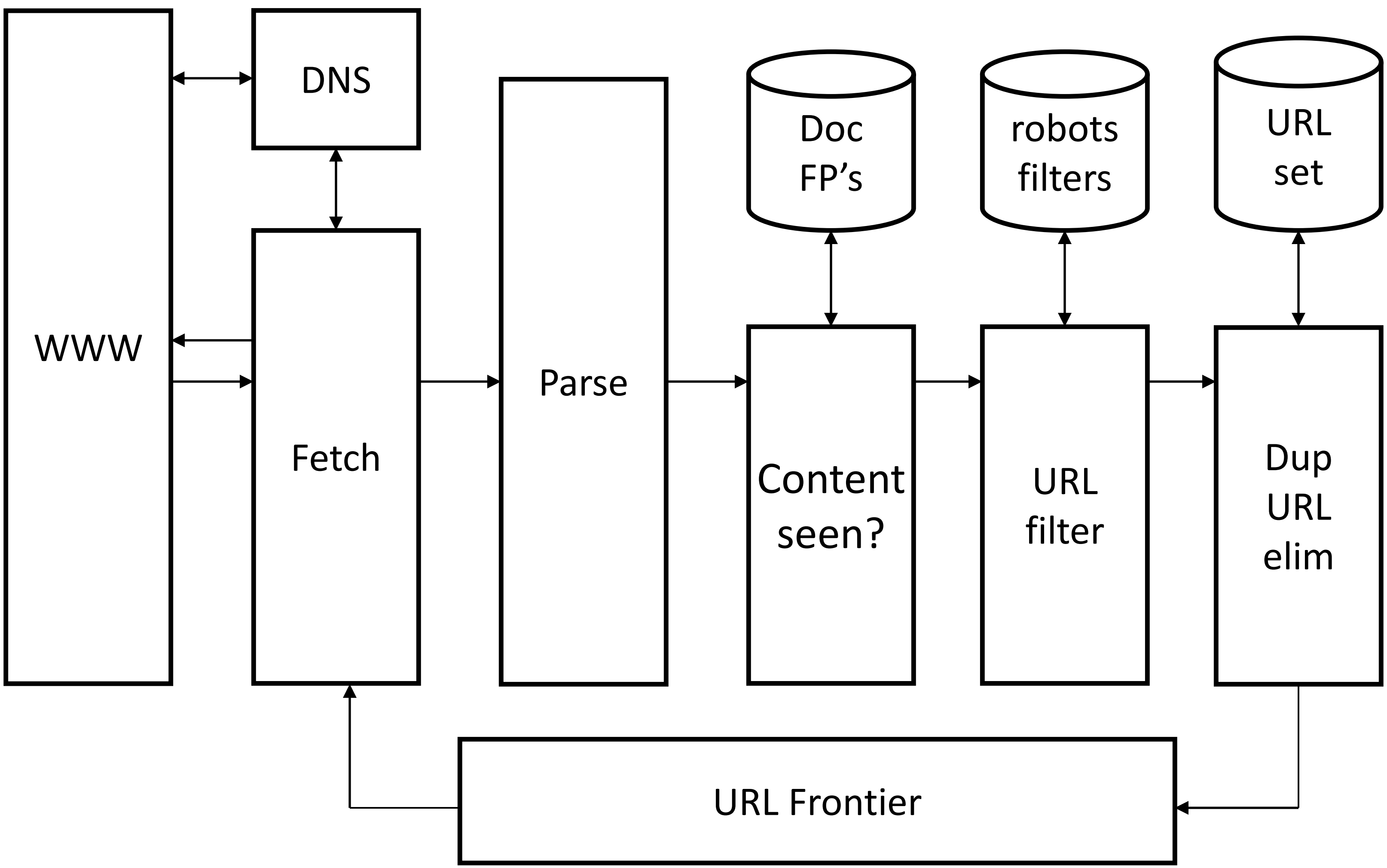
```
User-agent: *
Disallow: /yoursite/temp/

User-agent: searchengine
Disallow:
```

Processing steps in crawling

- Pick a URL from the frontier
- Fetch the document at the URL
- Parse the URL
  - Extract links from it to other docs (URLs)
- Check if URL has content already seen
  - If not, add to indexes
- For each extracted URL
  - Ensure it passes certain URL filter tests
  - Check if it is already in the frontier (duplicate URL elimination)

E.g., only crawl .edu, obey robots.txt, etc.

Basic crawl architecture



DNS (Domain Name Server)

- A lookup service on the internet
  - Given a URL, retrieve its IP address
  - Service provided by a distributed set of servers – thus, lookup latencies can be high (even seconds)
- Common OS implementations of DNS lookup are *blocking*: only one outstanding request at a time
- Solutions
  - DNS caching
  - Batch DNS resolver – collects requests and sends them out together

Parsing: URL normalization

- When a fetched document is parsed, some of the extracted links are *relative* URLs
- E.g., http://en.wikipedia.org/wiki/Main_Page has a relative link to /wiki/Wikipedia:General_disclaimer which is the same as the absolute URL http://en.wikipedia.org/wiki/Wikipedia:General_disclaimer
- During parsing, must normalize (expand) such relative URLs

Content seen?

- Duplication is widespread on the web
- If the page just fetched is already in the index, do not further process it
- This is verified using document fingerprints or shingles

Filters and robots.txt

- Filters – regular expressions for URLs to be crawled/not
- Once a robots.txt file is fetched from a site, need not fetch it repeatedly
  - Doing so burns bandwidth, hits web server
- Cache robots.txt files

Duplicate URL elimination

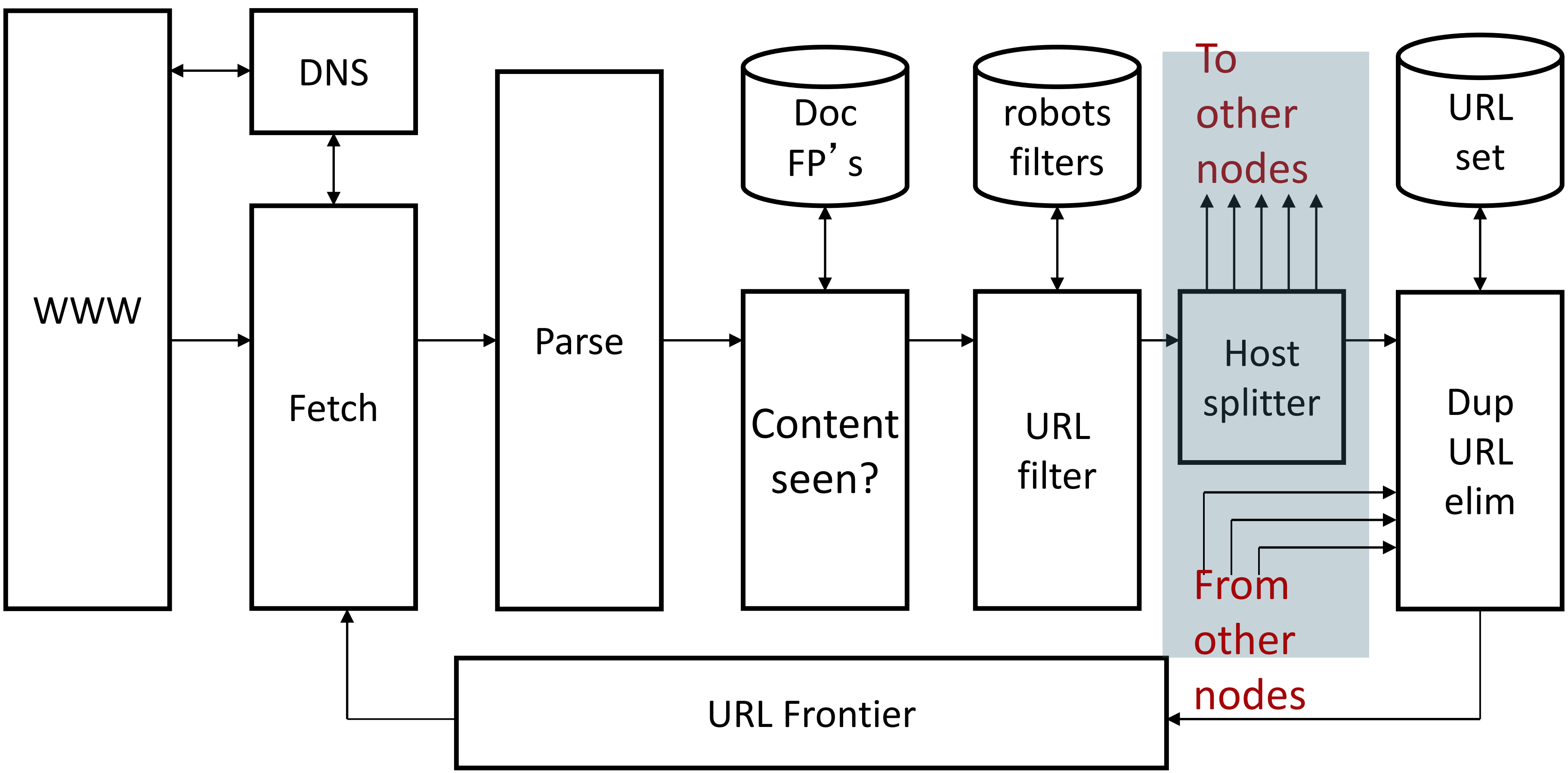
- For a non-continuous (one-shot) crawl, test to see if an extracted+filtered URL has already been passed to the frontier
- For a continuous crawl – see details of frontier implementation

Distributing the crawler

- Run multiple crawl threads, under different processes – potentially at different nodes
  - Geographically distributed nodes
- Partition hosts being crawled into nodes
  - Hash used for partition
- How do these nodes communicate and share URLs?

Communication between nodes

- Output of the URL filter at each node is sent to the Dup URL Eliminator of the appropriate node



URL frontier: two main considerations

- Politeness: do not hit a web server too frequently
- Freshness: crawl some pages more often than others
  - E.g., pages (such as News sites) whose content changes often

These goals may conflict with each other.

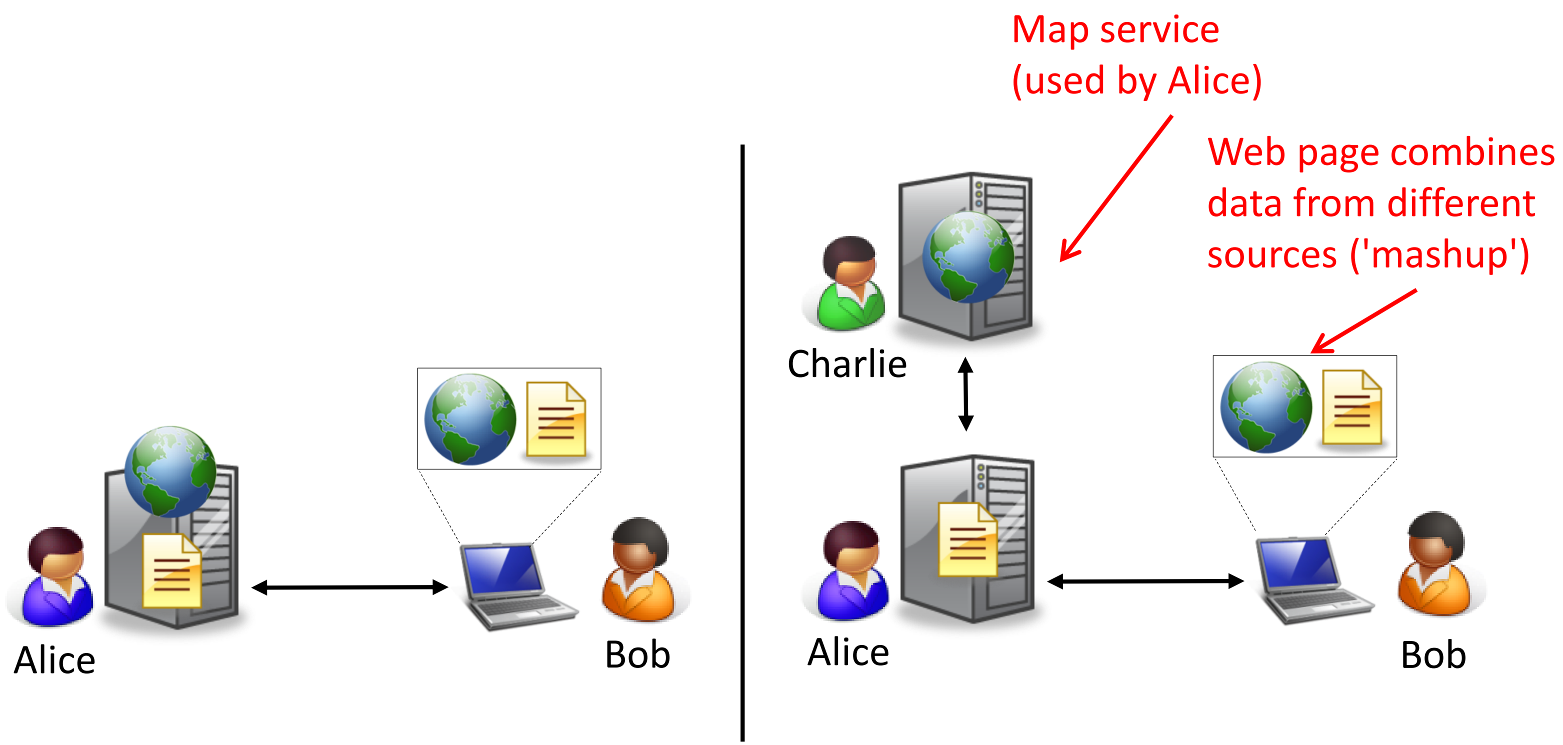
(E.g., simple priority queue fails – many links out of a page go to its own site, creating a burst of accesses to that site.)

Politeness – challenges

- Even if we restrict only one thread to fetch from a host, can hit it repeatedly
- Common heuristic: insert time gap between successive requests to a host that is >> time for most recent fetch from that host

Web APIs

What is a web service?

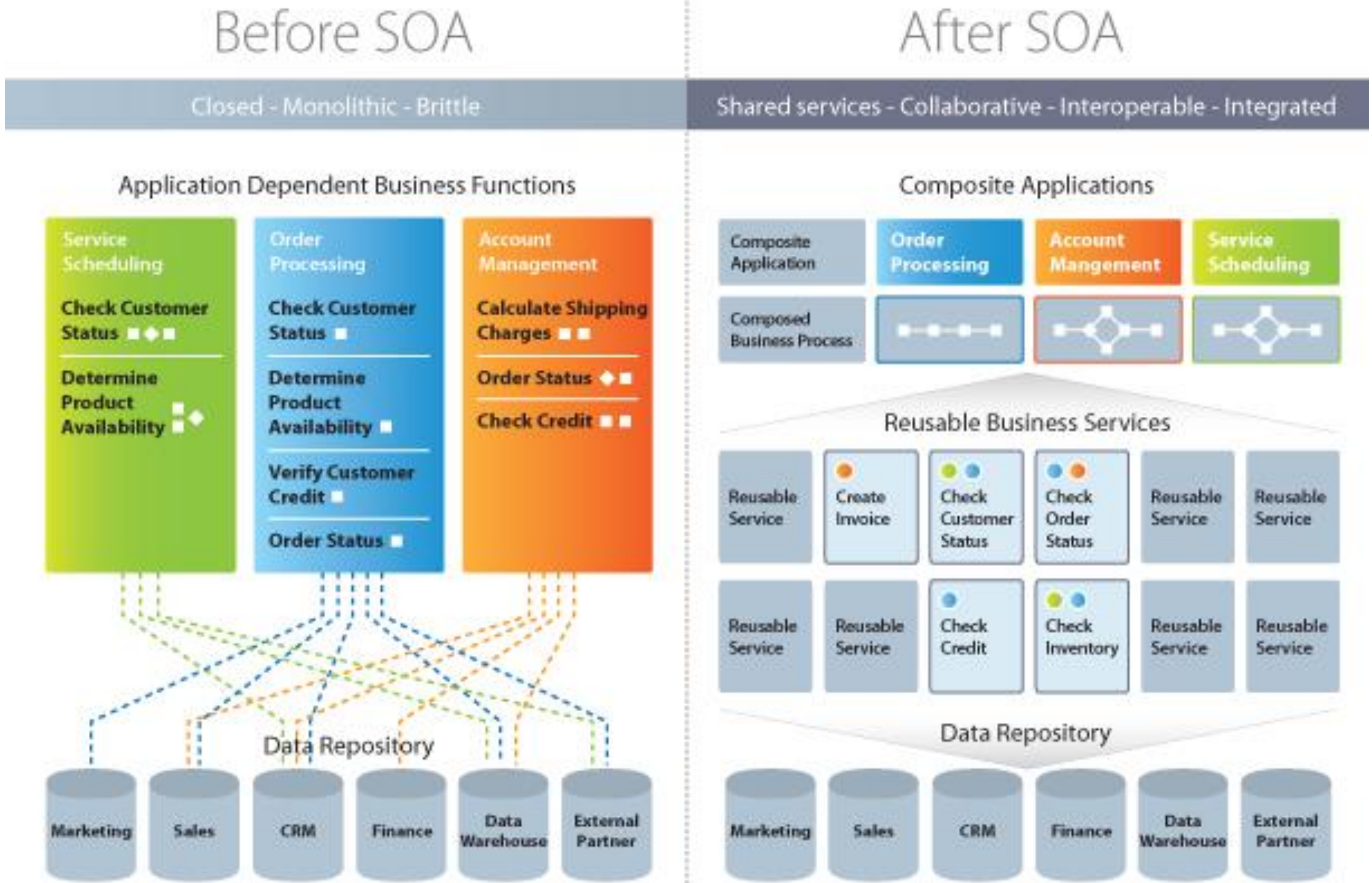


- Intuition: An application that is accessible to other applications over the web
  - Examples: Google Maps API, Facebook Graph API, eBay APIs, Amazon Web Services APIs, ...

http://www.programmableweb.com/apis/directory

Service-Oriented Architecture (SOA)

Service-oriented architecture (SOA) is a software design and software **architecture** design pattern based on distinct pieces of software providing application functionality as services to other applications. This is known as **service-orientation.** It is independent of any vendor, product or technology.


Before SOA
After SOA
Closed - Monolithic - Brittle
Shared services - Collaborative - Interoperable - Integrated
Application Dependent Business Functions
Service Scheduling
Check Customer Status
Determine Product Availability
Order Processing
Check Customer Status
Determine Product Availability
Verify Customer Credit
Order Status
Account Management
Calculate Shipping Charges
Order Status
Check Credit
Data Repository
Marketing
Sales
CRM
Finance
Data Warehouse
External Partner
Composite Applications
Composite Application
Order Processing
Account Mangement
Service Scheduling
Composed Business Process
Reusable Business Services
Reusable Service
Create Invoice
Check Customer Status
Check Order Status
Reusable Service
Reusable Service
Reusable Service
Reusable Service
Check Credit
Check Inventory
Reusable Service
Reusable Service
Data Repository
Marketing
Sales
CRM
Finance
Data Warehouse
External Partner

Over time, the level of abstraction at which functionality is specified, published and or consumed has gradually become higher and higher. We have progressed from modules, to objects, to components, and now to services.

http://msdn.microsoft.com/en-us/library/aa480021.aspx https://plus.google.com/+RipRowan/posts/eVeouesvaVX

- So one day Jeff Bezos issued a mandate. He's doing that all the time, of course, and people scramble like ants being pounded with a rubber mallet whenever it happens. But on one occasion -- back around 2002 I think, plus or minus a year -- he issued a mandate that was so out there, so huge and eyebulgingly ponderous, that it made all of his other mandates look like unsolicited peer bonuses.

- His Big Mandate went something along these lines:

- 1) All teams will henceforth expose their data and functionality through service interfaces.
- 2) Teams must communicate with each other through these interfaces.
- 3) There will be no other form of interprocess communication allowed: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network.
- 4) It doesn't matter what technology they use HTTP, Corba, Pubsub, custom protocols -- doesn't matter. Bezos doesn't care.

- 5) All service interfaces, without exception, must be designed from the ground up to be externalizable. That is to say, the team must plan and design to be able to expose the interface to developers in the outside world. No exceptions.
- 6) Anyone who doesn't do this will be fired.
- 7) Thank you; have a nice day!

- Ha, ha! You 150-odd ex-Amazon folks here will of course realize immediately that #7 was a little joke I threw in, because Bezos most definitely does not give a s*** about your day.

- #6, however, was quite real, so people went to work.

Available Web APIs

- Twitter: https://dev.twitter.com/
- Flickr: http://www.flickr.com/services/api/
- Google Maps: https://developers.google.com/maps/
- Facebook: http://developers.facebook.com/
- Foursquare: https://developer.foursquare.com/
- Yahoo Boss API: http://developer.yahoo.com/search/boss/
- Wikipedia API: http://www.mediawiki.org/wiki/API:Main_page
- Youtube API: http://code.google.com/apis/youtube/overview.html
- Openstreetmap API: http://wiki.openstreetmap.org/wiki/API
- Halo API: https://developer.haloapi.com/

- List of APIs: https://www.reddit.com/r/webdev/comments/3wrswc/what_are_some_fun_apis_to_play_with/