Web Search: Agenda

- Quick web overview
- How is the web different?
- Types of web information needs
- Crawling
- Link analysis

The WWW was invented in 1989, at CERN.

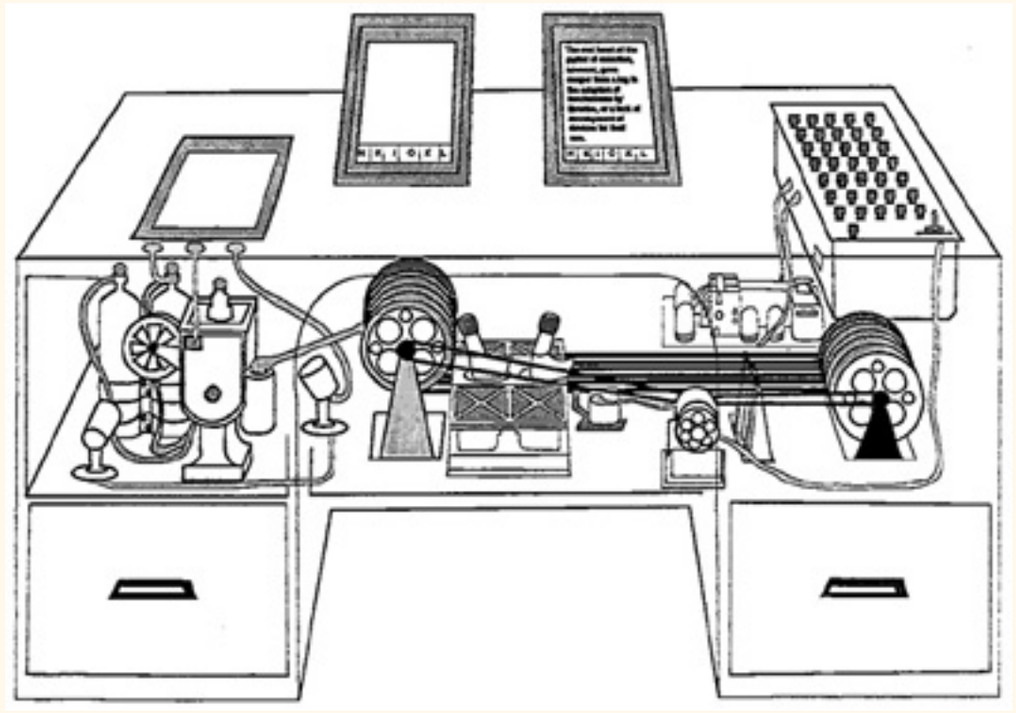
Sir Tim Berners-Lee
1955 –

Keys to the WWW: HTTP, HTML, and hypertext.

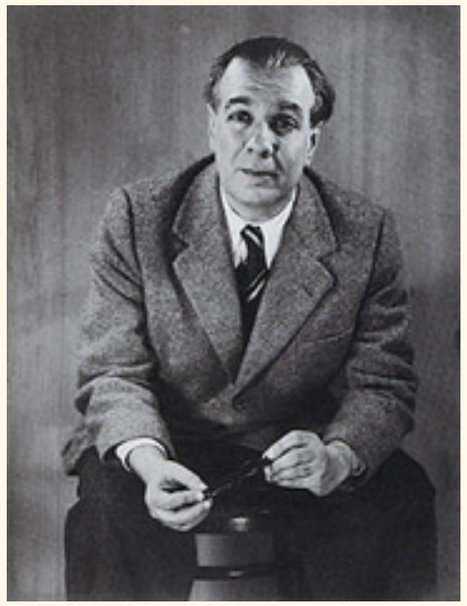
"I just had to take the hypertext idea and connect it to the Transmission Control Protocol and domain name system ideas and—ta-da!—the World Wide Web ... **Creating the web was really an act of desperation**, because the situation without it was very difficult when I was working at CERN later. Most of the technology involved in the web, like the hypertext, like the Internet, multifont text objects, had all been designed already. I just had to put them together. It was a step of generalising, going to a higher level of abstraction, thinking about all the documentation systems out there as being possibly part of a larger imaginary documentation system."

https://commons.wikimedia.org/wiki/File:NeXTcube_first_webserver.JPG
https://en.wikipedia.org/wiki/Tim_Berners-Lee

Depending on who you ask, the concept of hypertext was invented by either:



Vannevar Bush
1890 – 1974



Jorge Luis Borges
1899 – 1986

Regardless, it was not until the 1960s that actual hypertext systems were built...

Ted Nelson
1937 –

Douglas Engelbart
1925 – 2013

... and not until the 1970s that any were really usable.

What distinguished the web from previous information systems?

1. Simplicity
2. Decentralization
3. Hypertext

In other words: anybody could easily add any content and cross-link it any-which-way.

This led to incredibly rapid and uncontrolled growth...

... which in turn posed major challenges for search:

	Standard	Web
Index size	(relatively) small	petabytes+
Updates & Additions	rare	constant
Language	monolingual	aggressively multilingual
Content	consistent (news articles, etc.)	diverse formats
Authors	curated	uncontrolled
Behavior	cooperative	adversarial
Quality	homogeneous	heterogeneous, subjective

Web information needs fall into three broad categories:

Informational:

Seeking information on a broad topic;
answer usually not found on any single page
"Leukemia", "art galleries in Portland"

Navigational:

Seeking a specific website from a particular entity
"alaska airlines"

Transactional:

A prelude to the user performing a transaction (buying something, making a reservation, etc.)
"alaska airlines reservation desk"

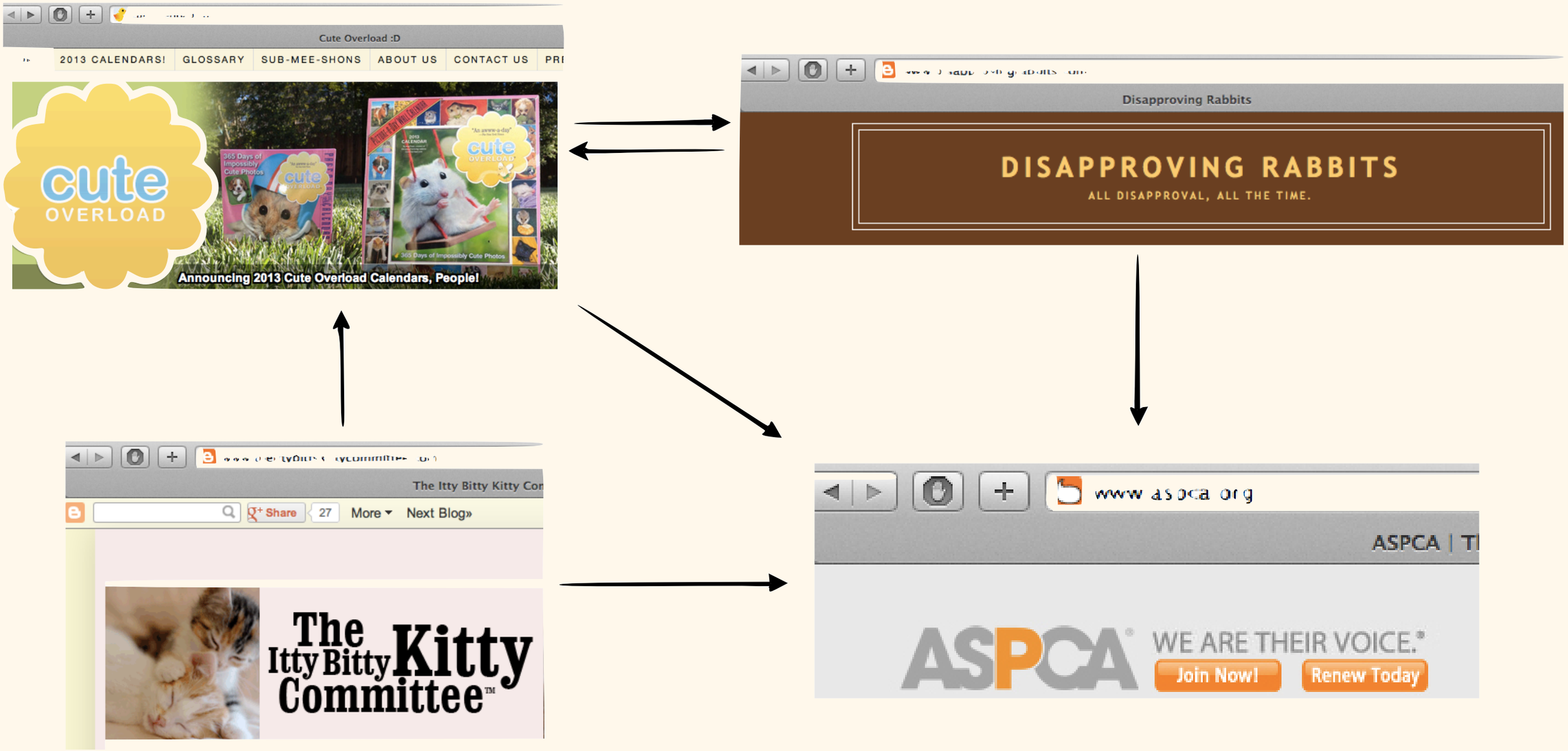
Search engines try to infer the category of need from a query (and user behavior: prior queries, clicks, etc.)...

... this is a huge area of research!

Early web search engnes took one of two basic approaches:

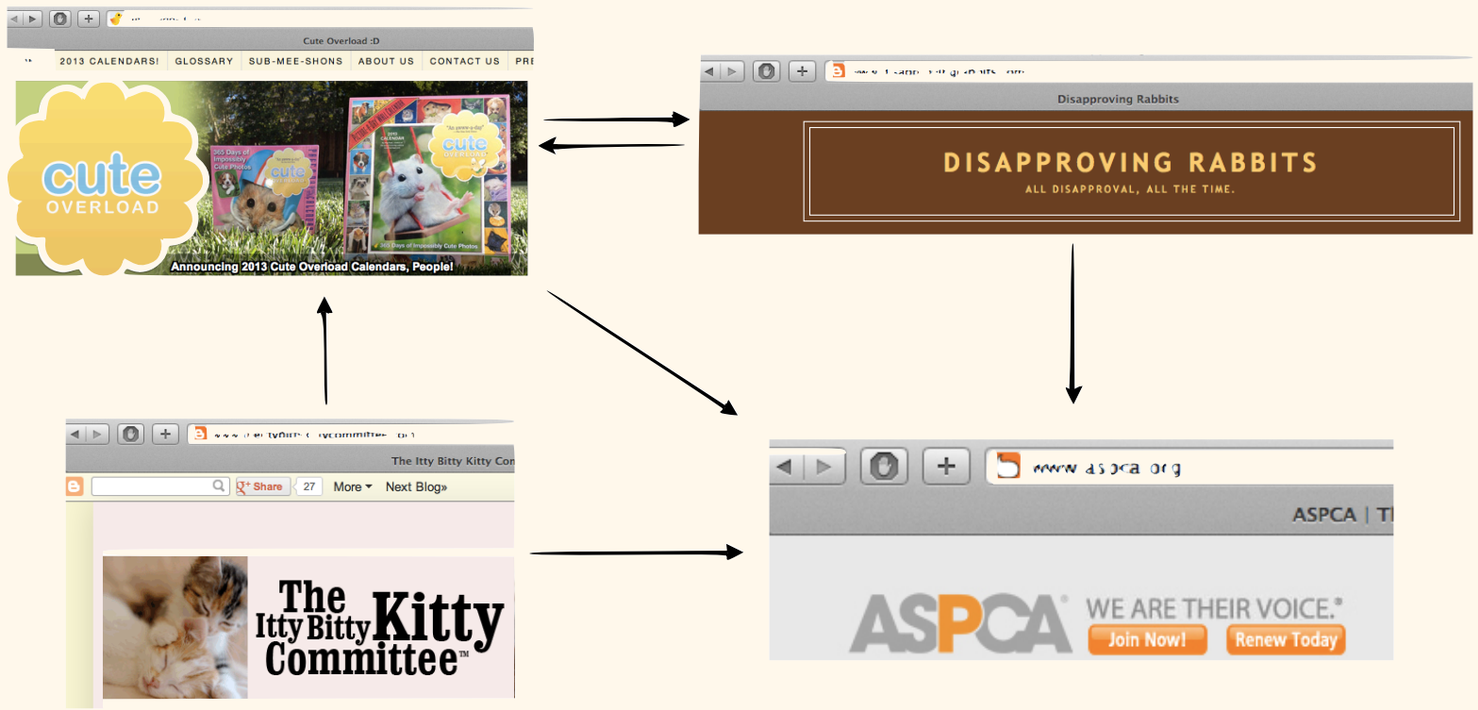
Both were based primarily on the *content* of the indexed web pages.

The web can be thought of as a directed graph...



This is a very useful formulation!

The web can be thought of as a directed graph...



A page's *in-degree*: # incoming links; *out-degree*: # outgoing links

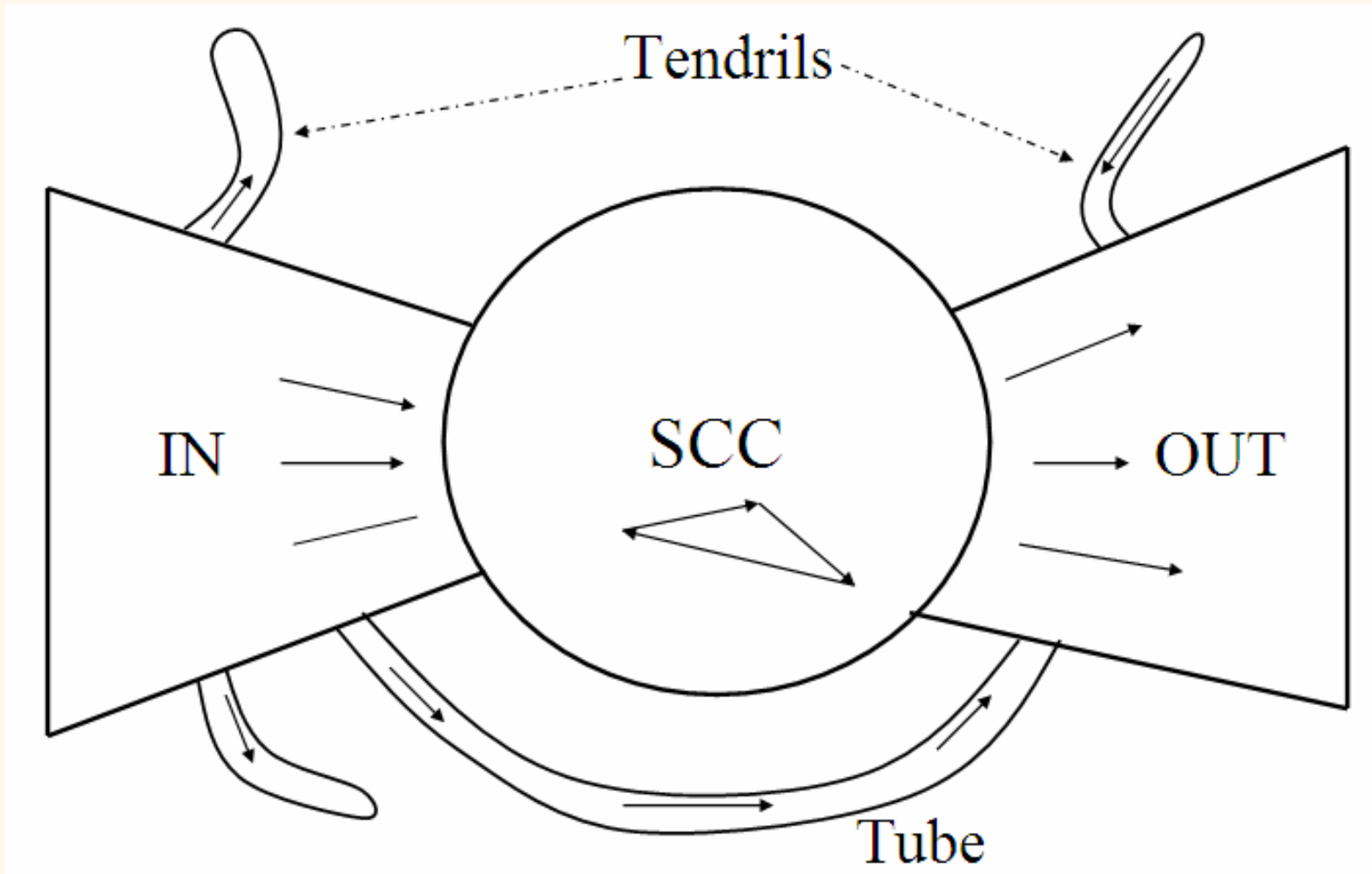
The web graph is *not* strongly connected;

Links are *non-random*;

The link distribution is often said to follow a power-law distribution:

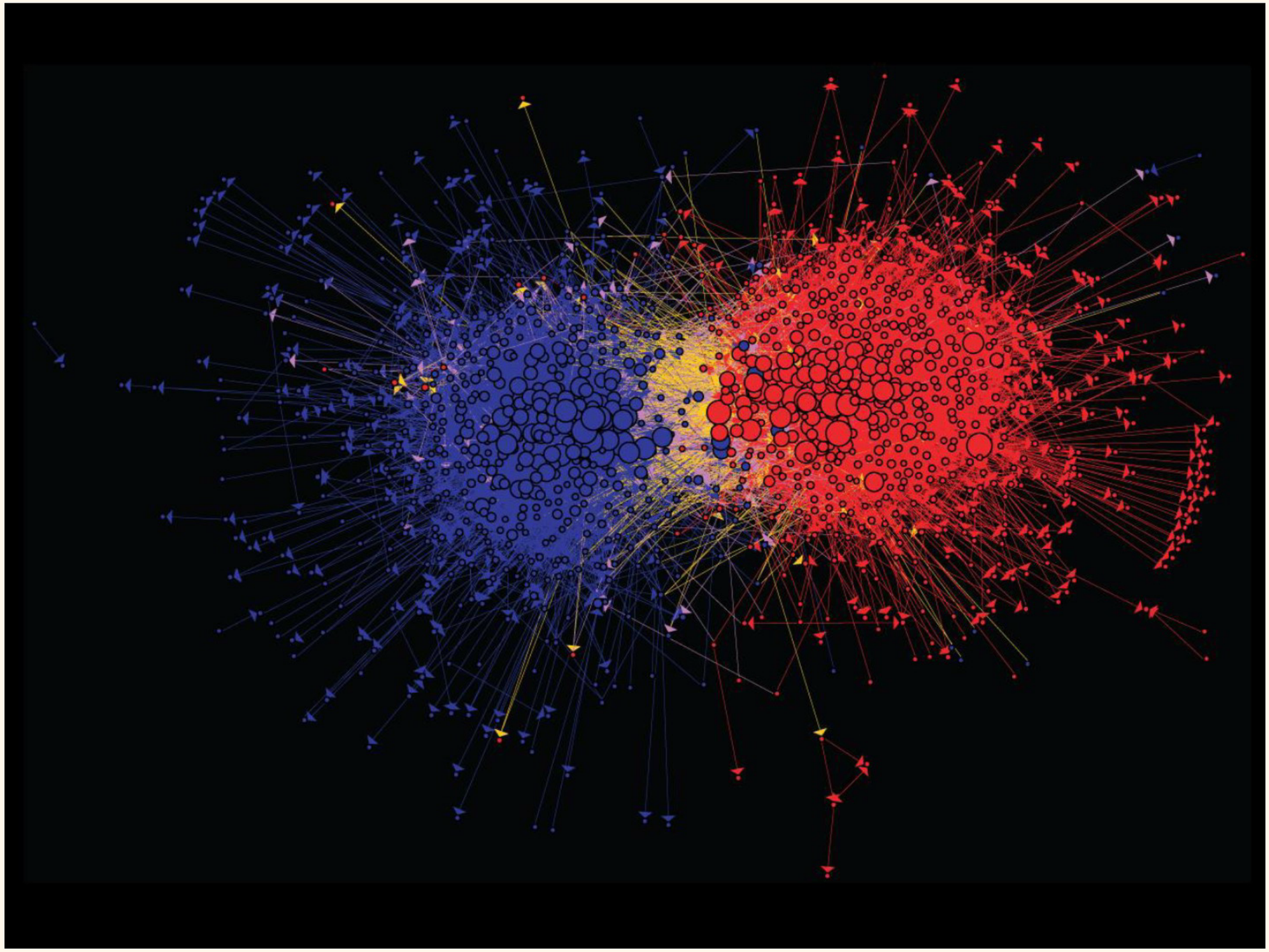
num. pages with in-degree of i is proportional to $1/i^{\alpha}$

The web can be thought of as a directed graph...



► **Figure 19.4** The bowtie structure of the Web. Here we show one tube and three tendrils.

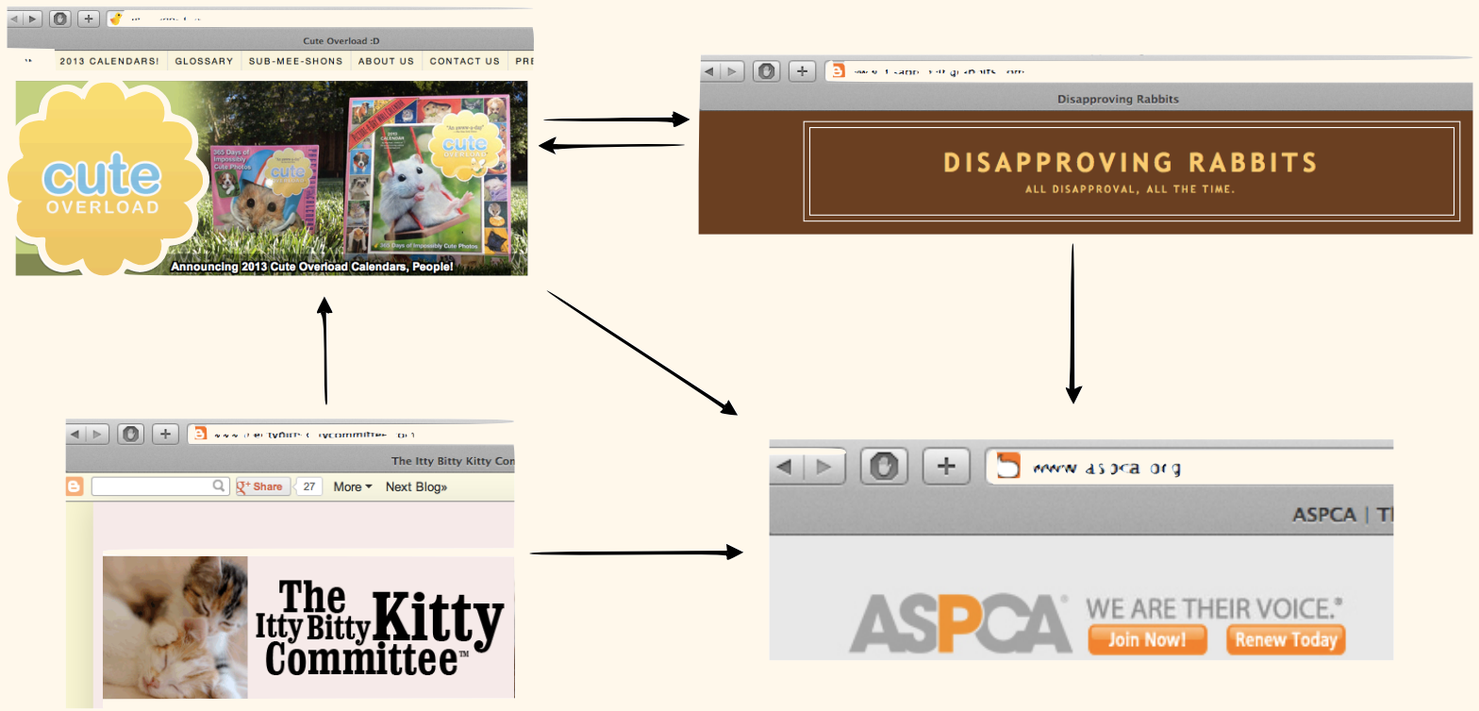
From Manning et al., Ch. 19



Connections between political blogs

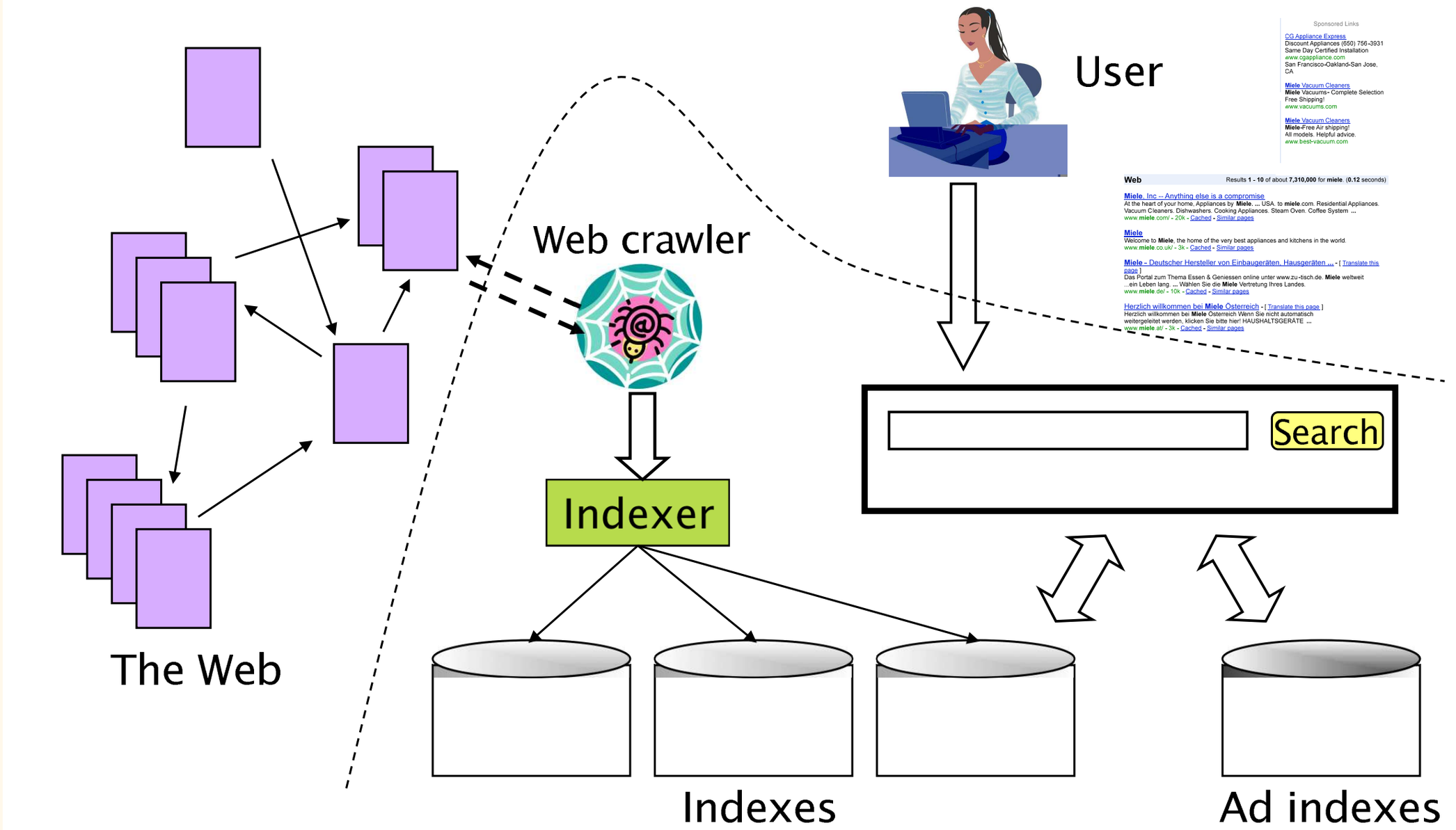
Polarization of the network [Adamic-Glance, 2005]

The web can be thought of as a directed graph...



We can use link data in two main ways:

1. Inferring *authority* or *trustworthiness* of a page;
2. Identifying new pages to be added to our index.



► **Figure 19.7** The various components of a web search engine.

From Manning et al., Ch. 19

"Crawling" is a basic and important piece of any search engine:

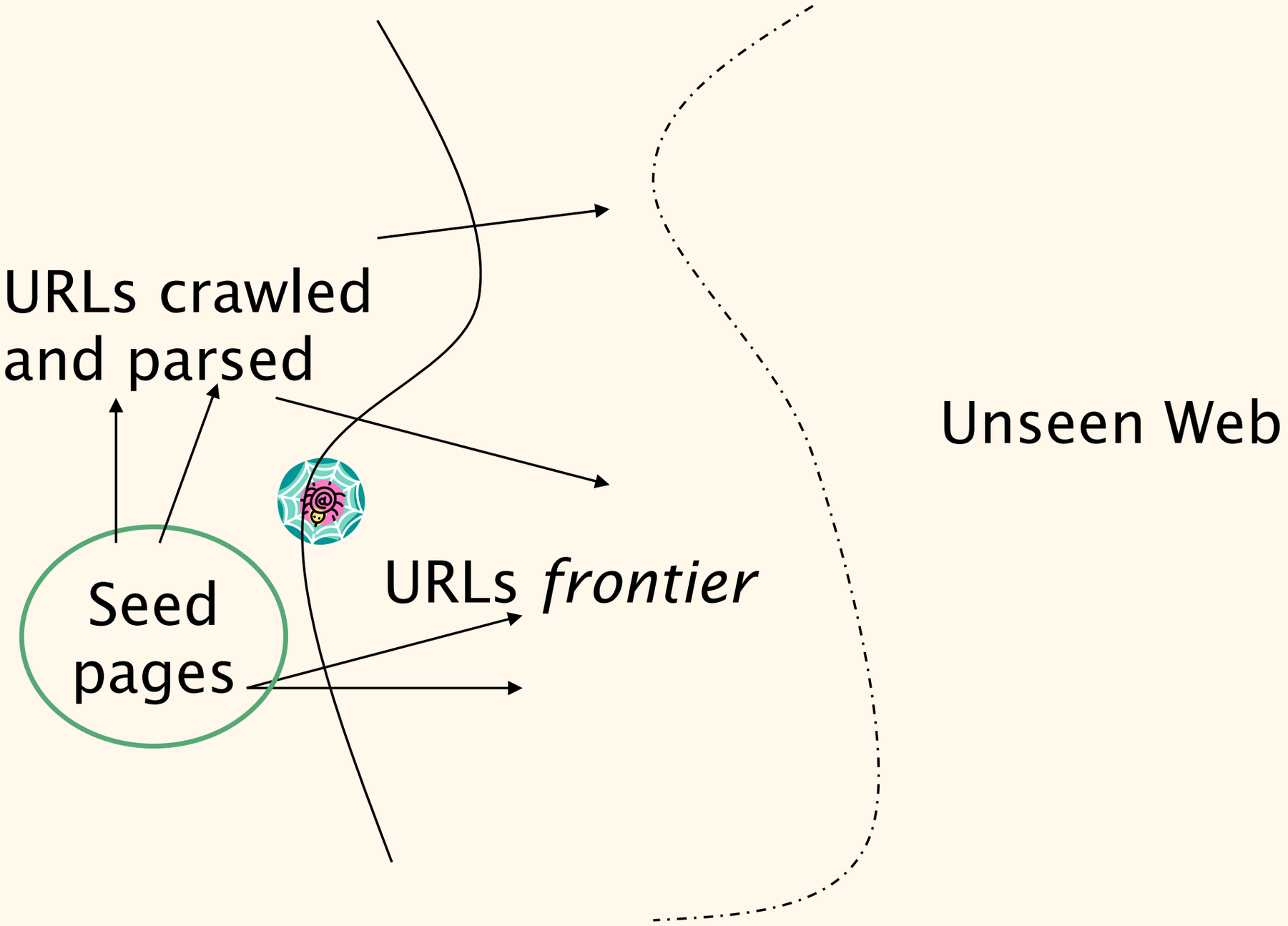
Starting with a set of "seed" URLs...

Fetch & parse, extract URLs...

Add to queue...

Fetch & parse each URL on queue, etc.

"Crawling" is a basic and important piece of any search engine:



From Manning et al., Ch. 19

This may sound easy, but it actually is not:

Crawling requires parallelism, with all the joy that brings...

Malicious pages seek to confound crawlers:

Spam, Spider traps (both static & dynamic)

Latency & Bandwidth vary widely across sites

Sites have different crawling policies (depth, frequency, etc.)

Duplicate pages, site mirrors, “dynamic” UIs, etc.

Politeness

There are two primary things that crawlers *must* do:

Be Robust:

Be immune to spider traps; gracefully handle bandwidth, timeout, and web server issues

Be Polite:

Respect implicit and explicit conventions

There are two primary things that crawlers *must* do:

Be Robust:

Be immune to spider traps; gracefully handle bandwidth, timeout, and web server issues

Be Polite:

Respect implicit and explicit conventions

Implicit politeness: avoid hitting a site too often, etc.

Explicit politeness: honor specifications from webmasters about what portions of the site may be crawled (`robots.txt`), etc.

Robots.txt:

A way to “control” what parts of your site get crawled (and by whom).

Dating from 1994, the protocol has remained (relatively) static.

Well-behaved crawlers look for a text file named “robots.txt” at the root of your website, and follow its directives:

```
User-Agent: *
Allow: /about_us
Disallow: /docs/private_files/
```

Robots.txt:

Extensions: directives for crawl frequency & rate, etc.

Sitemap, Host (to specify canonical mirrors, etc.)

Caution: a robots.txt file is *not* a security mechanism!

```
User-agent: *
Disallow: /_archive/
Disallow: /_resources/
Disallow: /academic/som/
Disallow: /bigbrain_courses_staging/
...
Disallow: /itgdba/
```

Quite the opposite, in fact...

Robots.txt:

> “During the reconnaissance stage of a web application testing, the tester (or attacker) usually uses a list of known subdirectories to brute force the server and find hidden resources.”
>
> http://thiébaud.fr/robots.txt.html

```
User-agent: *
Disallow: /admin/
Disallow: /stats/
Disallow: /internaljobs/
Disallow: /internaljobsbyorganization/
Disallow: /internaljobsearch/
```

http://www.behindthefirewalls.com/2013/07/using-robotstxt-to-locate-your-targets.html

There are two primary things that crawlers *must* do:

Be Robust:

Be immune to spider traps; gracefully handle bandwidth, timeout, and web server issues

Be Polite:

Respect implicit and explicit conventions

Implicit politeness: avoid hitting a site too often, etc.

Explicit politeness: honor specifications from webmasters about what portions of the site may be crawled (`robots.txt`), etc.

Beyond those requirements, there are several *"shoulds"*:

Be capable of distributed operation;

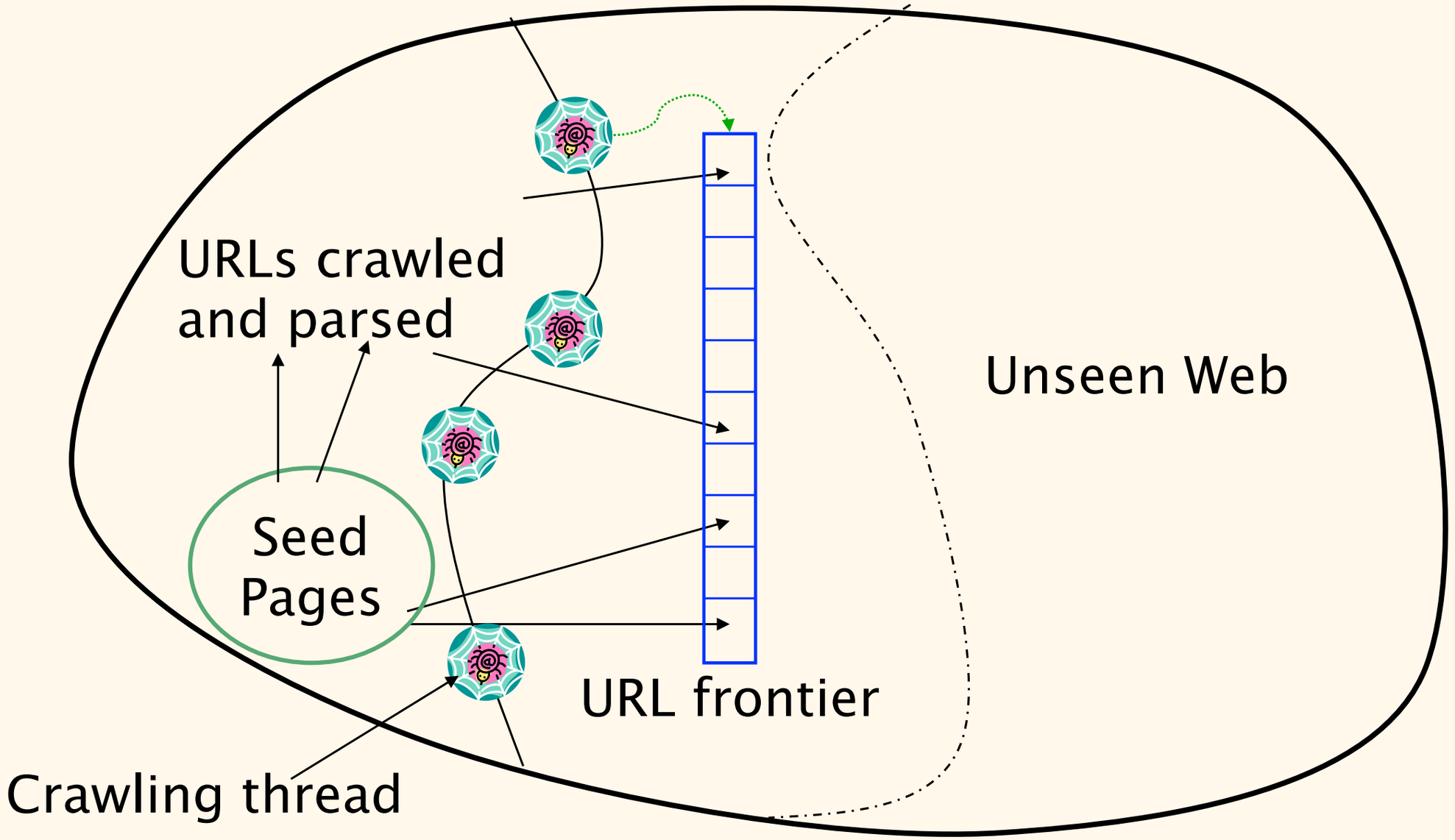
Be scalable (increase crawl rate by adding machines)

Be as efficient as possible (both in terms of CPU and network)

Be clever: fetch "higher quality" pages first, etc.

Operate continuously

Be easily extensible to support new protocols, document types, etc.

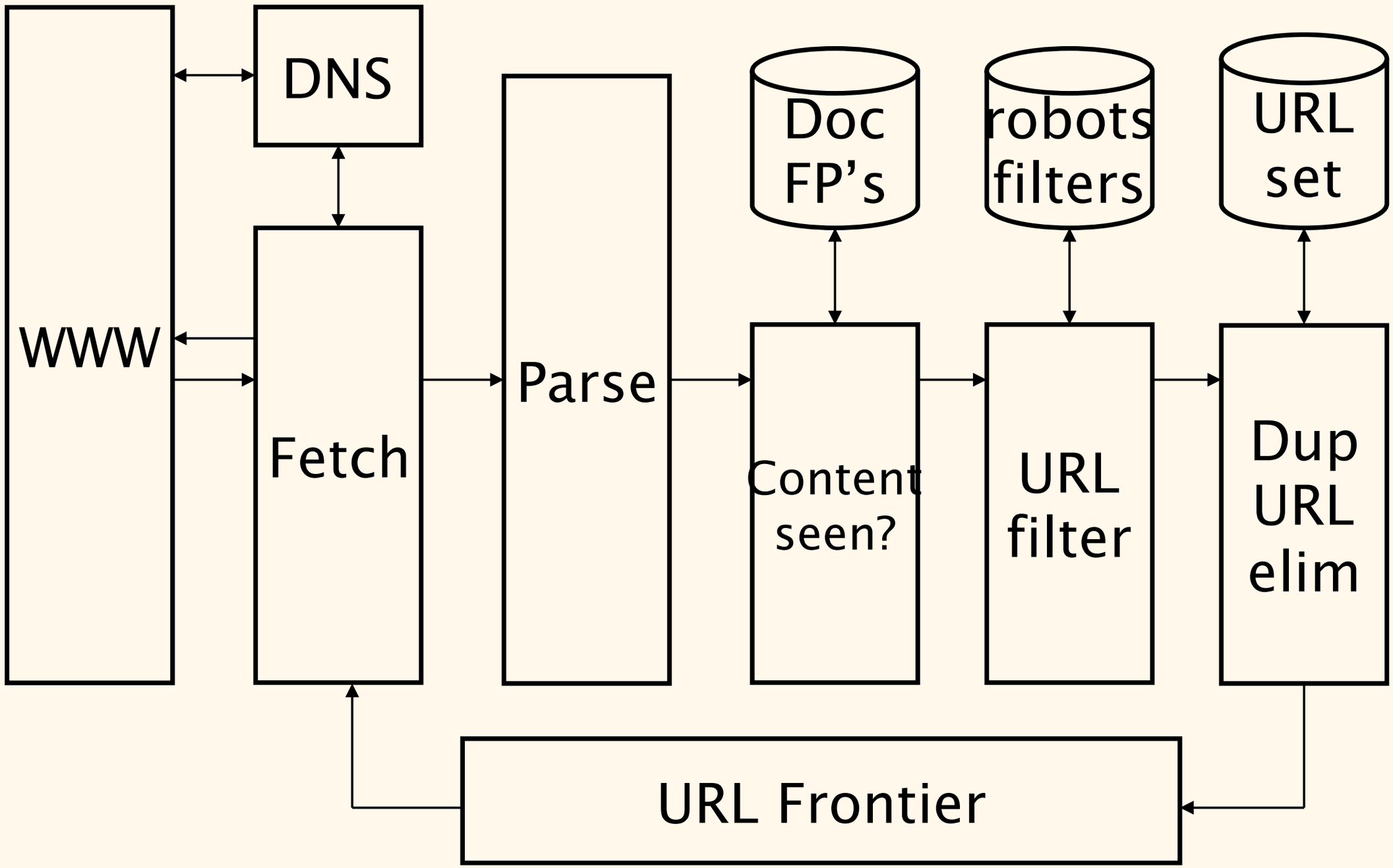


From Manning et al., Ch. 19

Which one?

- 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)

robots.txt filters, etc.


DNS
WWW
Fetch
Parse
Doc FP's
Content seen?
robots filters
URL filter
URL set
Dup URL elim
URL Frontier

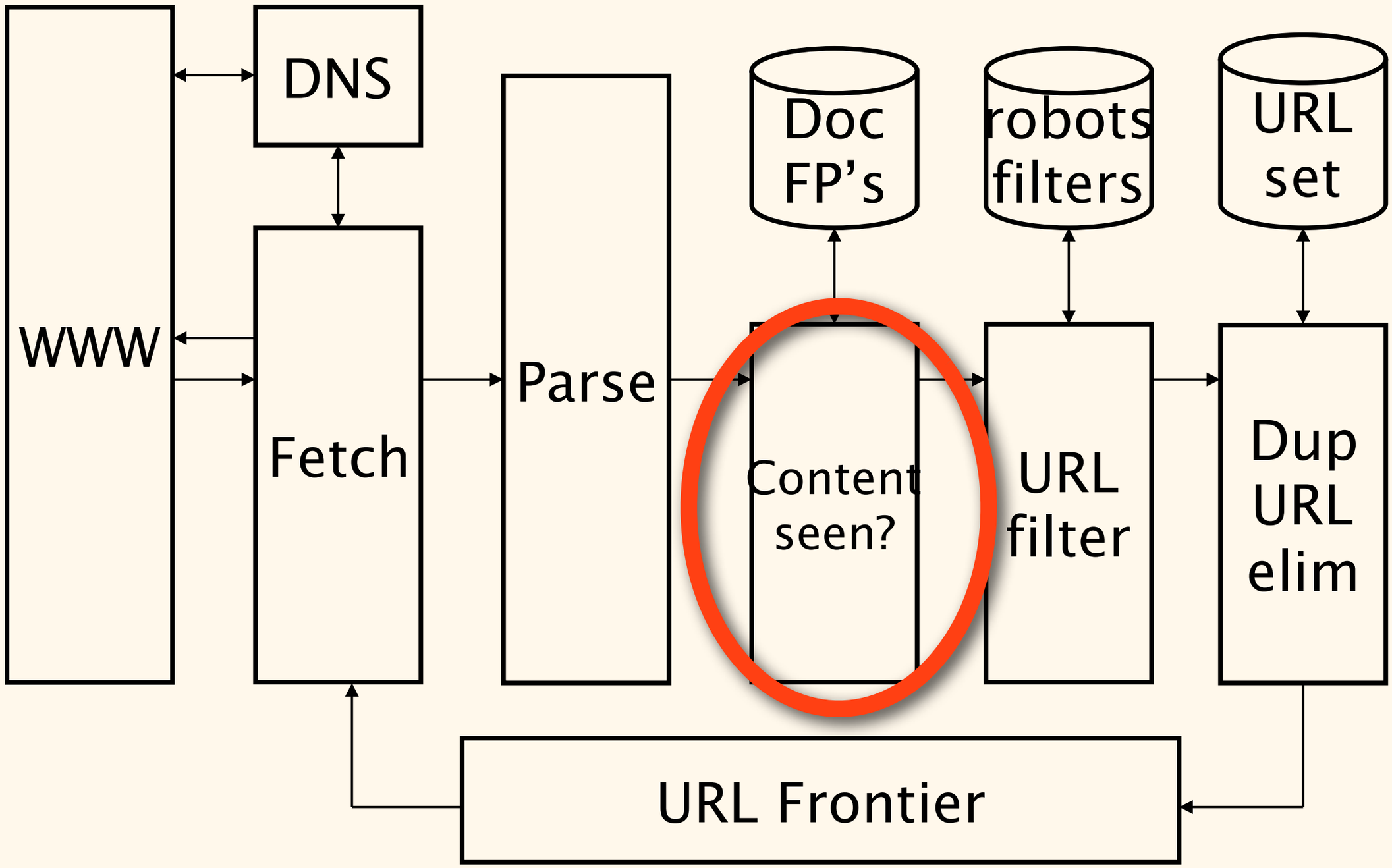
URL frontier: two main considerations

- 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.)

There are many solutions: see the book!


WWW
DNS
Fetch
Parse
Doc FP's
Content seen?
robots filters
URL filter
URL set
Dup URL elim
URL Frontier

Duplicate documents

- The web is full of duplicated content
- Strict duplicate detection = exact match
  - Not as common
- But many, many cases of near duplicates
  - E.g., Last modified date the only difference between two copies of a page

From Manning et al., Ch. 19

Duplicate/Near-Duplicate Detection

- *Duplication*: Exact match can be detected with fingerprints
- *Near-Duplication*: Approximate match
  - Overview
    - Compute syntactic similarity with an edit-distance measure
    - Use similarity threshold to detect near-duplicates
      - E.g., Similarity > 80% => Documents are “near duplicates”
      - Not transitive though sometimes used transitively

From Manning et al., Ch. 19

Computing Similarity

- Features:
  - Segments of a document (natural or artificial breakpoints)
  - Shingles (Word N-Grams)
  - ***a rose is a rose is a rose*** → 4-grams are

    a_rose_is_a

    rose_is_a_rose

    is_a_rose_is

    a_rose_is_a
- Similarity Measure between two docs (= sets of shingles)
  - Jaccard cooefficient: (Size_of_Intersection / Size_of_Union)

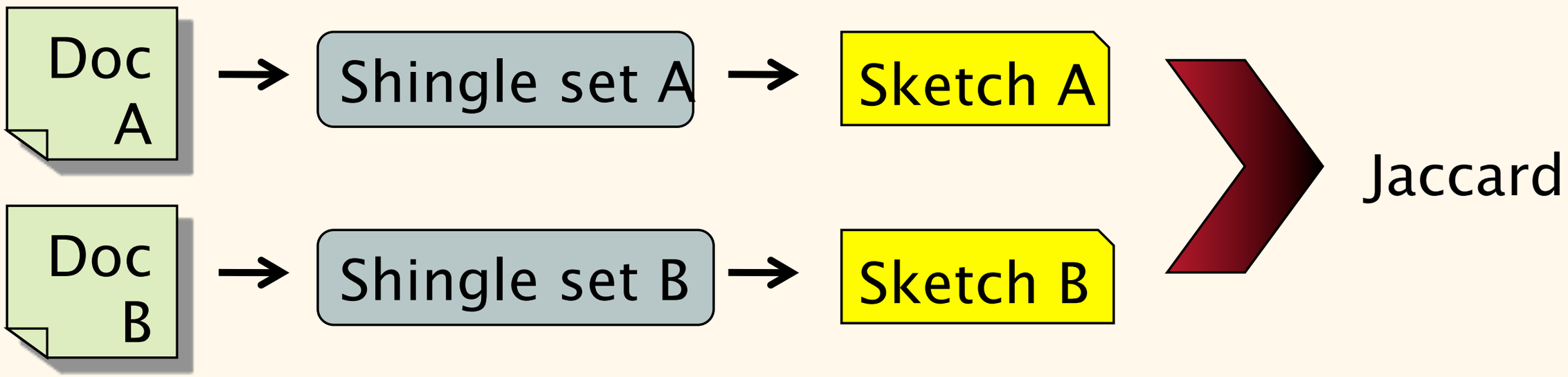
From Manning et al., Ch. 19

Shingles + Set Intersection

- Computing exact set intersection of shingles between all pairs of documents is expensive

- Approximate using a cleverly chosen subset of shingles from each (a *sketch*)

- Estimate (size_of_intersection / size_of_union) based on a short sketch



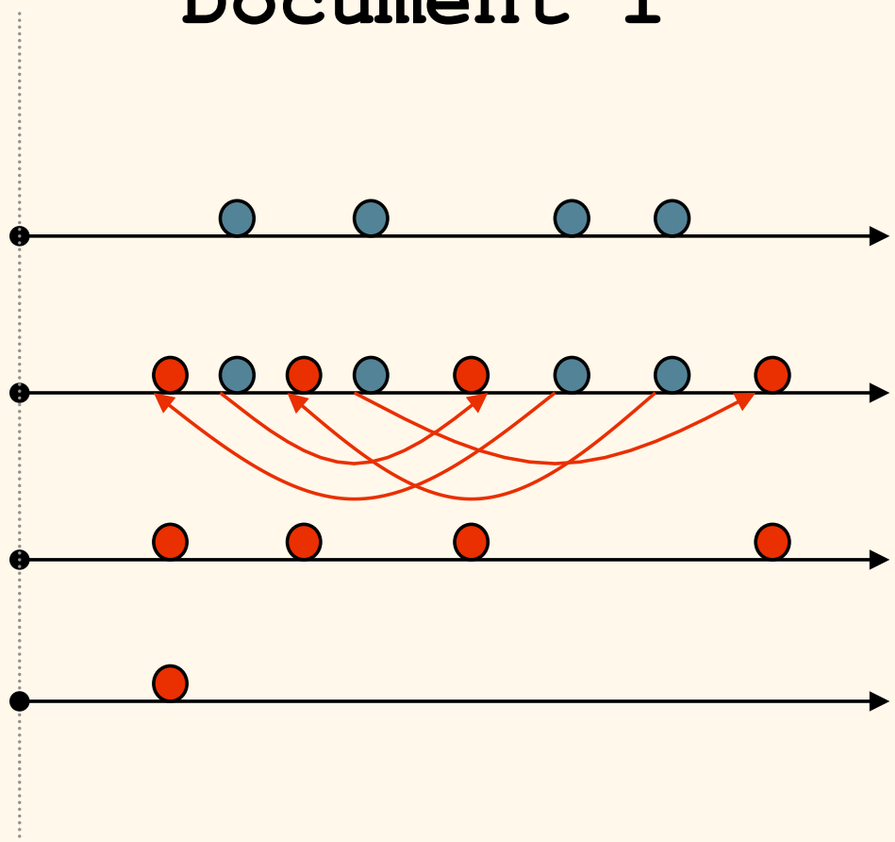
From Manning et al., Ch. 19

Sketch of a document

- Create a "sketch vector" (of size ~200) for each document
  - Documents that share ≥ *t* (say 80%) corresponding vector elements are deemed near duplicates
    - For doc *D*, sketch$_D$[*i*] is as follows:
      - Let f map all shingles in the universe to 1..2^m (e.g., f = fingerprinting)
      - Let π_i be a *random permutation* on 1..2^m
      - Pick MIN {π_i(f(s))} over all shingles *s* in *D*

From Manning et al., Ch. 19

Computing Sketch[i] for Doc1



Document 1

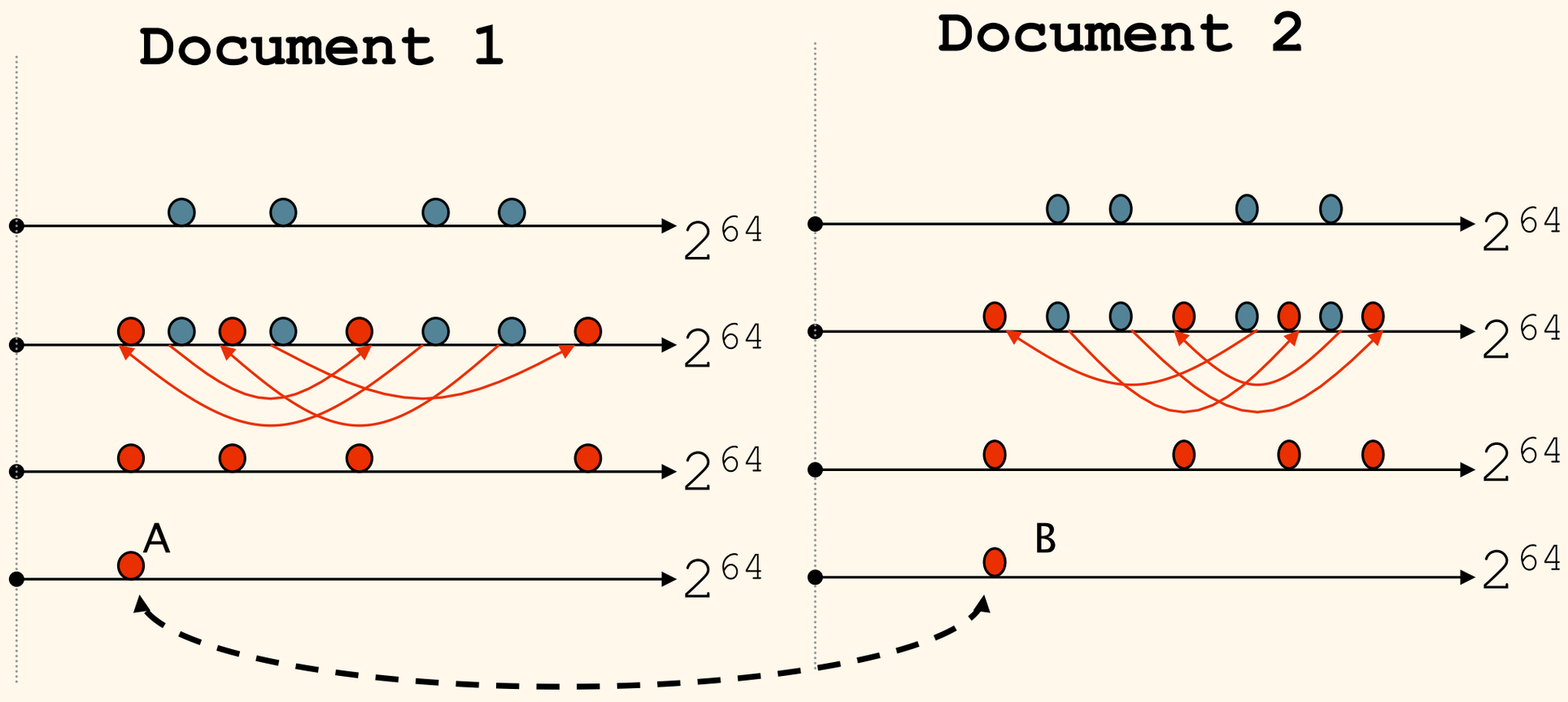
Start with 64-bit *f*(shingles)

Permute on the number line with π_i

Pick the min value

From Manning et al., Ch. 19

Test if Doc1.Sketch[i] = Doc2.Sketch[i]



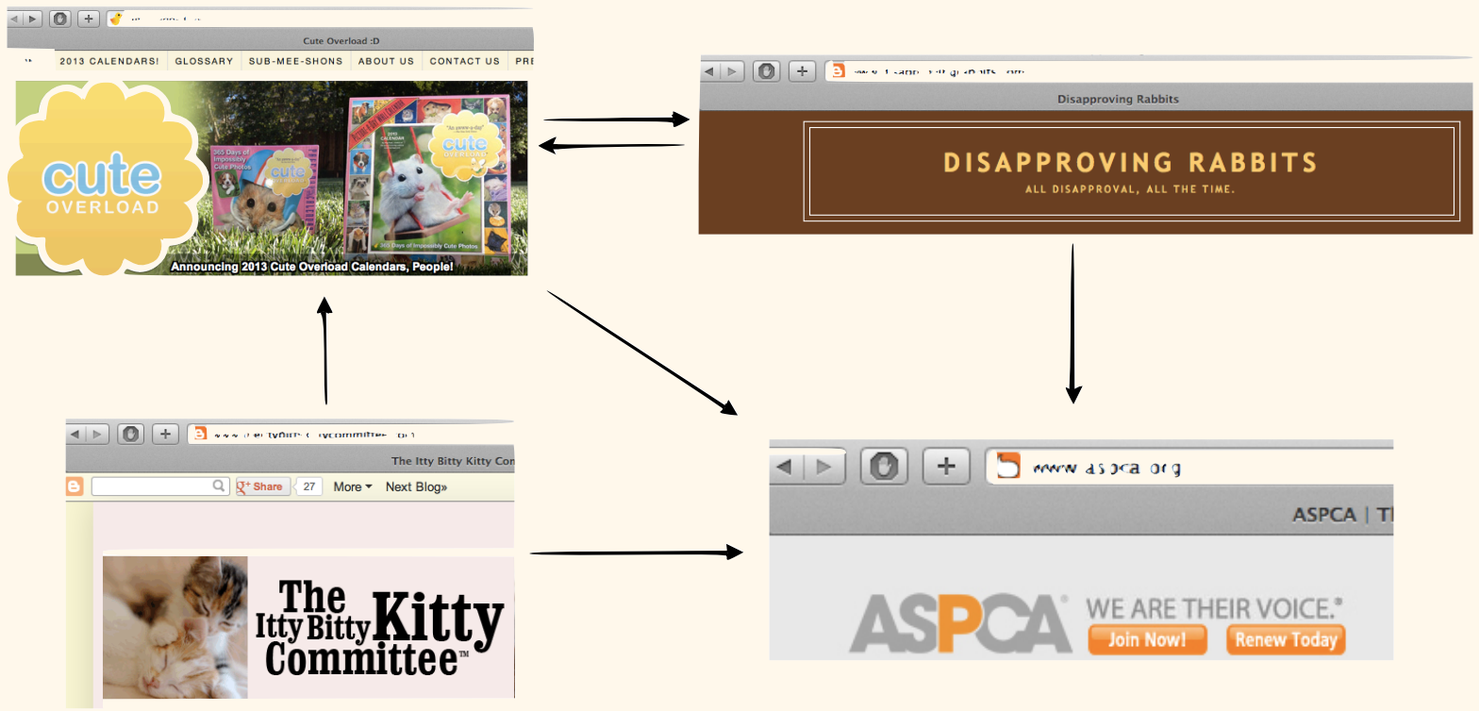
Test for 200 random permutations: $\pi_1, \pi_2, \ldots \pi_{200}$

From Manning et al., Ch. 19

- Shingling is a *randomized algorithm*
  - Our analysis did not presume any probability model on the inputs
  - It will give us the right (wrong) answer with some probability on *any input*
- We've described how to detect near duplication in a pair of documents
- In "real life" we'll have to concurrently look at many pairs
  - See text book for details

From Manning et al., Ch. 19

The web can be thought of as a directed graph...



We can use link data in two main ways:

1. Inferring *authority* or *trustworthiness* of a page;
2. Identifying new pages to be added to our index.

Link analysis piece...

Challenges to all of the above: spam

Since good search results mean more traffic, more customers, etc...

... people are strongly motivated to try and “game” search engine’s indexing processes.

> *“As the popularity of the Web has increased, the efforts to exploit the Web for commercial, social, or political advantage have grown, making it harder for search engines to discriminate between truthful signals of content quality and deceptive attempts to game search engines’ rankings.”*

Castillo C, Davison BD. Adversarial Web Search. Foundations and Trends® in Information Retrieval. 2010;4(5):377–486.

Early attempts at search engine spamming involved “keyword stuffing”:

Including (many!) additional instances of key words or phrases on the page...

... but doing it in a way that was invisible to human users (hiding in a metadata field, causing them to blend in with the background, etc.).

Modern spammers have to be more clever:

Cloaking:

Returning one page for humans, and another for index crawlers

Doorway pages:

Show a page that looks legit to a web crawler, but have all the outgoing links point to commercial pages

Scraping:

Steal high-quality content from another page, and then link to commercial pages (sending “quality points” from your scraped page to the crappy spam pages)

Link buying/exchange:

What it sounds like

Comment spam:

Putting your spam in higher-ranked websites’ comment sections

Foundations and Trends® in
Information Retrieval
Vol. 4, No. 5 (2010) 377–486

DOI: 10.1561/1500000021

now
the essence of knowledge

Adversarial Web Search

By Carlos Castillo and Brian D. Davison

Contents

Next up: search UI/UX.