INFORMATION RETRIEVAL

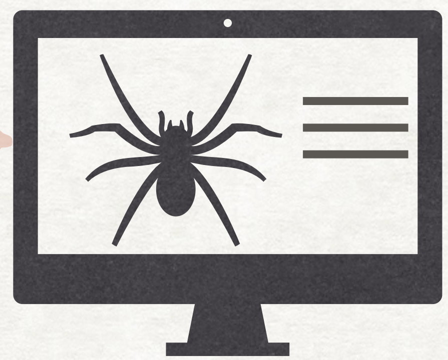
Luca Manzoni
lmanzoni@units.it

Lecture 6

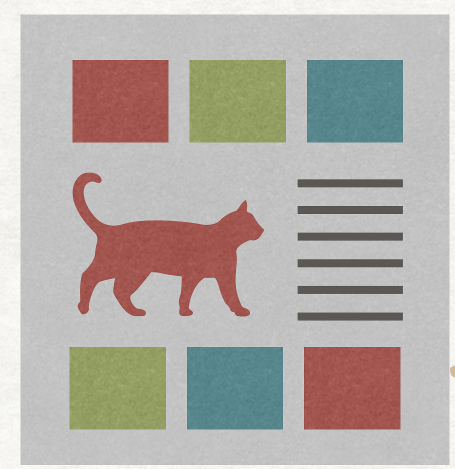
LECTURE OUTLINE

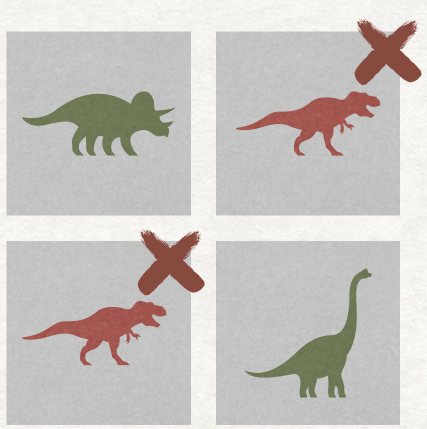
*NOW AVAILABLE VIA THE INFORMATION SUPERHIGHWAY

Web Crawling



Detecting content

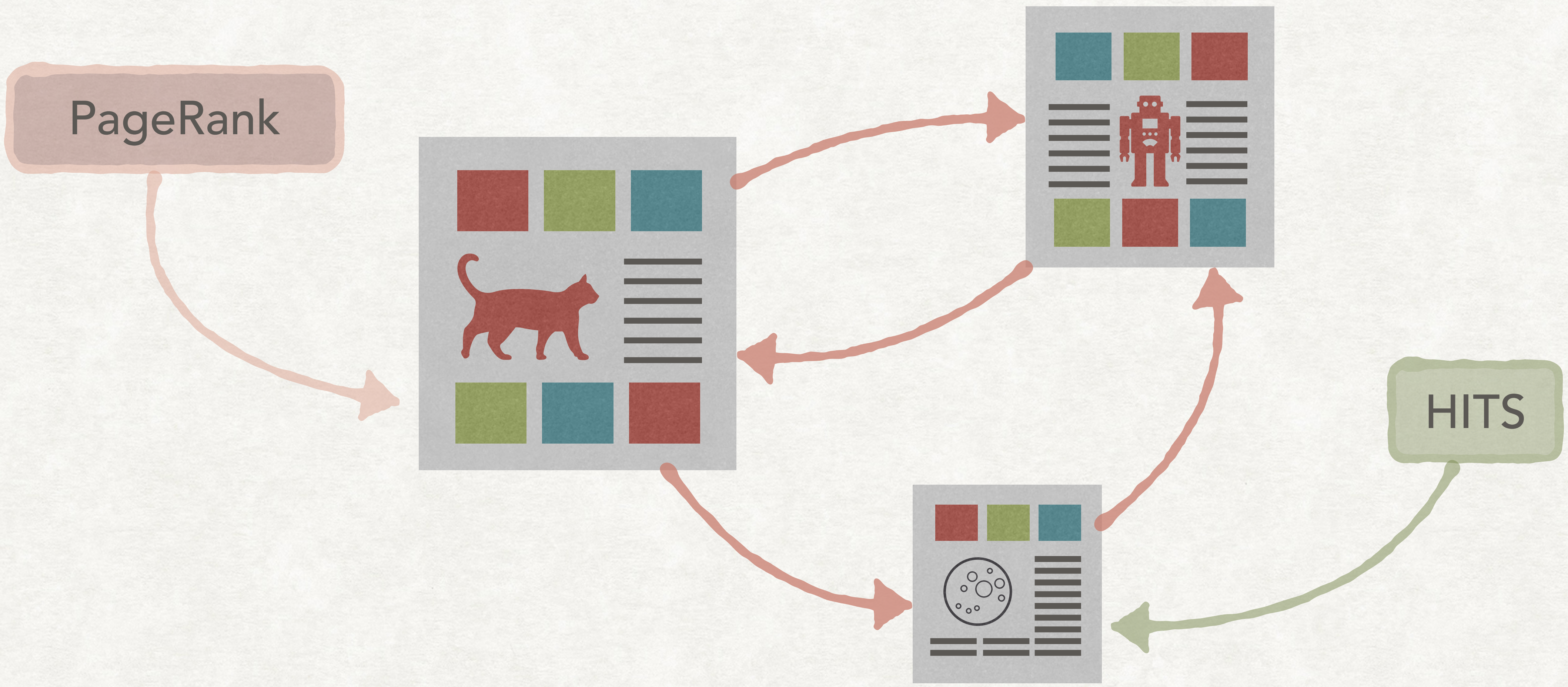




Duplicates and near-duplicates

LECTURE OUTLINE

*REQUIRES AT LEAST A 486



But first of all…

BASICS OF WEB SEARCH

TERMINOLOGY

BASICS

- **HTTP** and **HTTPS**. Protocols used to transmit web pages.
- **HTML**. The markup language used to encode web pages
- **URL**. Universal resource locator, (protocol + hostname + resource). E.g, *https://www.example.com/a/resource.html* has
  - *https*: protocol
  - *www.example.com*: hostname
  - */a/resource.html*: resource

TERMINOLOGY

LINKS

- **Static web pages**. The content does not change between multiple requests.
- **Dynamic web pages**. Automatically generated pages, e.g., in response to a query to a database.
- **Anchor text**. The text visualised for a link: `<a href=URL>Anchor text</a>`
- **In-links**: set of links that refer to a web page (notice that they are not contained in the web page).
- **Out-links**: set of links from a web page (this can be obtained by looking at the web page)

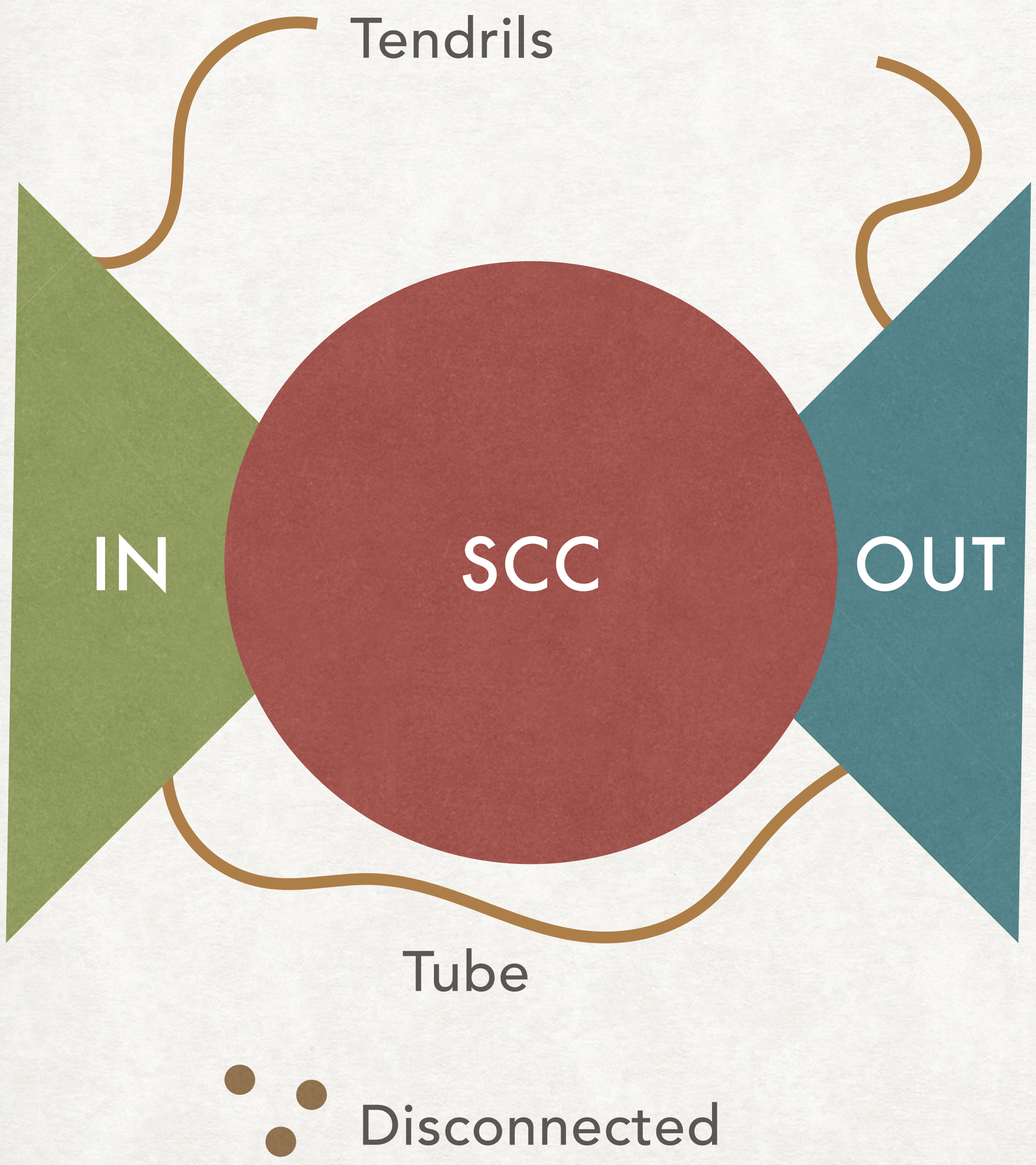
THE WEB AS A GRAPH

AT DIFFERENT LEVELS

- The web can be seen as a graph on different levels:
  - A page is a node of the graph, with outgoing edges given by the links that it contains.
  - A PLD (pay-level-domain, like example.com, amazon.com, etc.) is a node with outgoing edges given by all the links contained in the pages on the PLD.
- In both cases, the distribution of in-degrees and out-degrees of the nodes is far from the classical random graph model (the Erdős–Rényi model), it is more closely modelled by a power law distribution $f(x) = ax^{-k}$.

BOWTIE SHAPE

STRUCTURE OF WEB LINKS



- **SCC**. By following hyperlinks it is possible to reach each other page in SCC
- **IN**. Pages that can reach SCC, but cannot be reached by pages in SCC.
- **OUT**. Pages that can be reached from SCC, but cannot reach SCC.
- **Tubes**. Direct links from IN to OUT
- **Tendrils**. Pages reachable from IN that lead nowhere or that reach only pages in OUT.

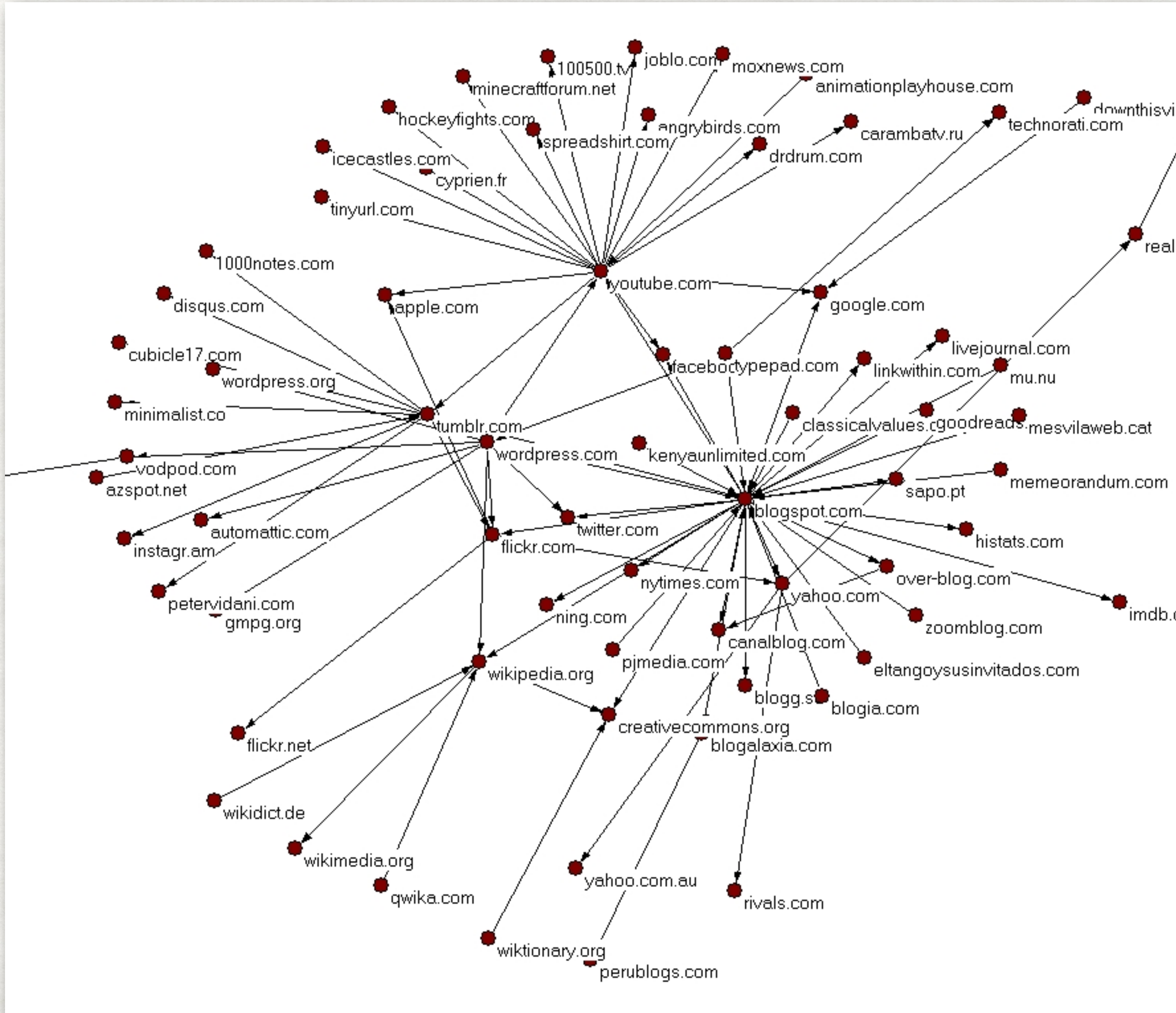
SOME REAL-WORLD DATA

2012 WDC HYPERLINK GRAPH

- The 2012 Web Data Commons (WDC) hyperlink graph includes about 3.5 billions of web pages and 128 billions of links.
- We will see the results on
  - The in- and out-degree distribution of the nodes.
  - The presence of a bow-tie shape.
  - All of this at the pay-level-domain (PLD) level.

Available at http://webdatacommons.org/hyperlinkgraph/2012-08/topology.html

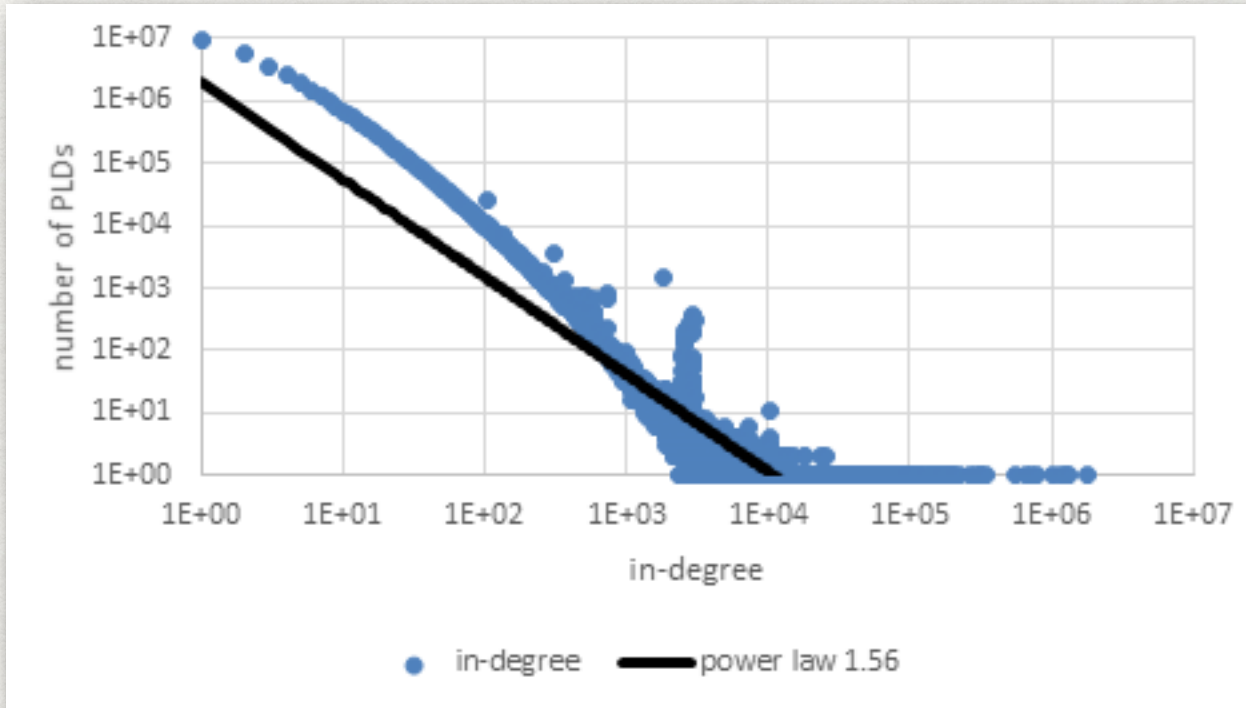
A GRAPH OF PLD

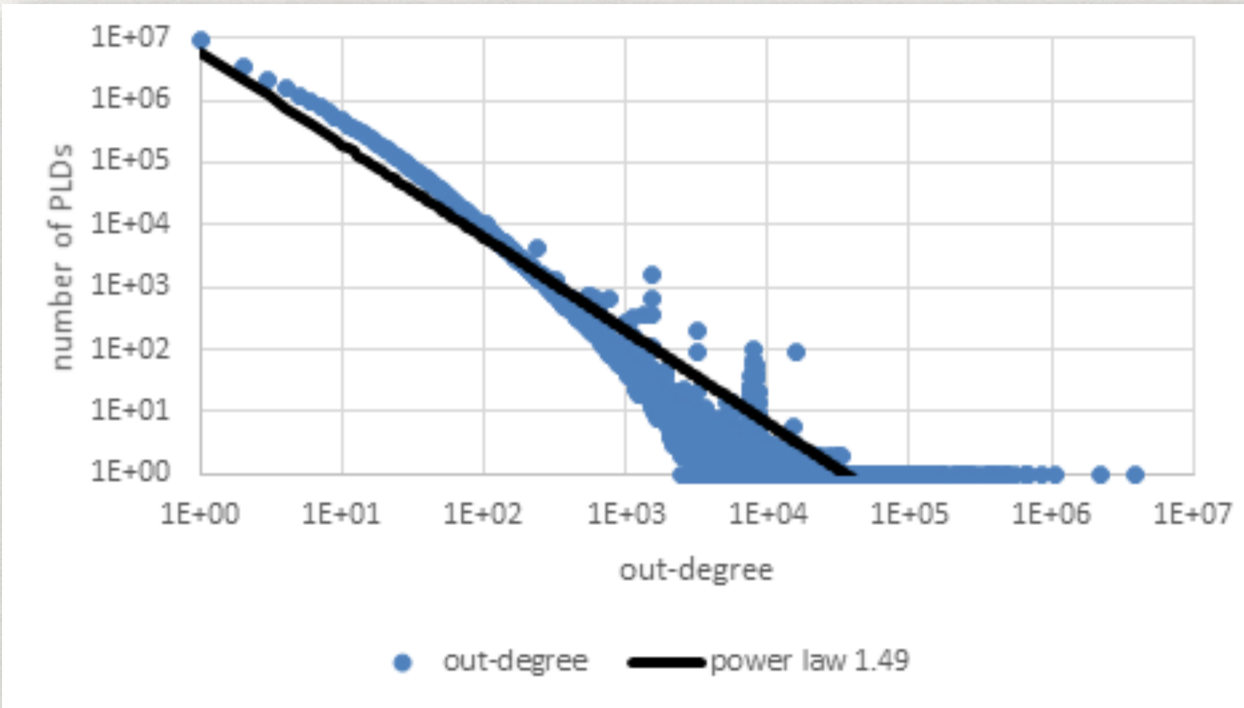


DISTRIBUTION OF DEGREES

AT THE PLD LEVEL

Both axes have a logarithmic scale





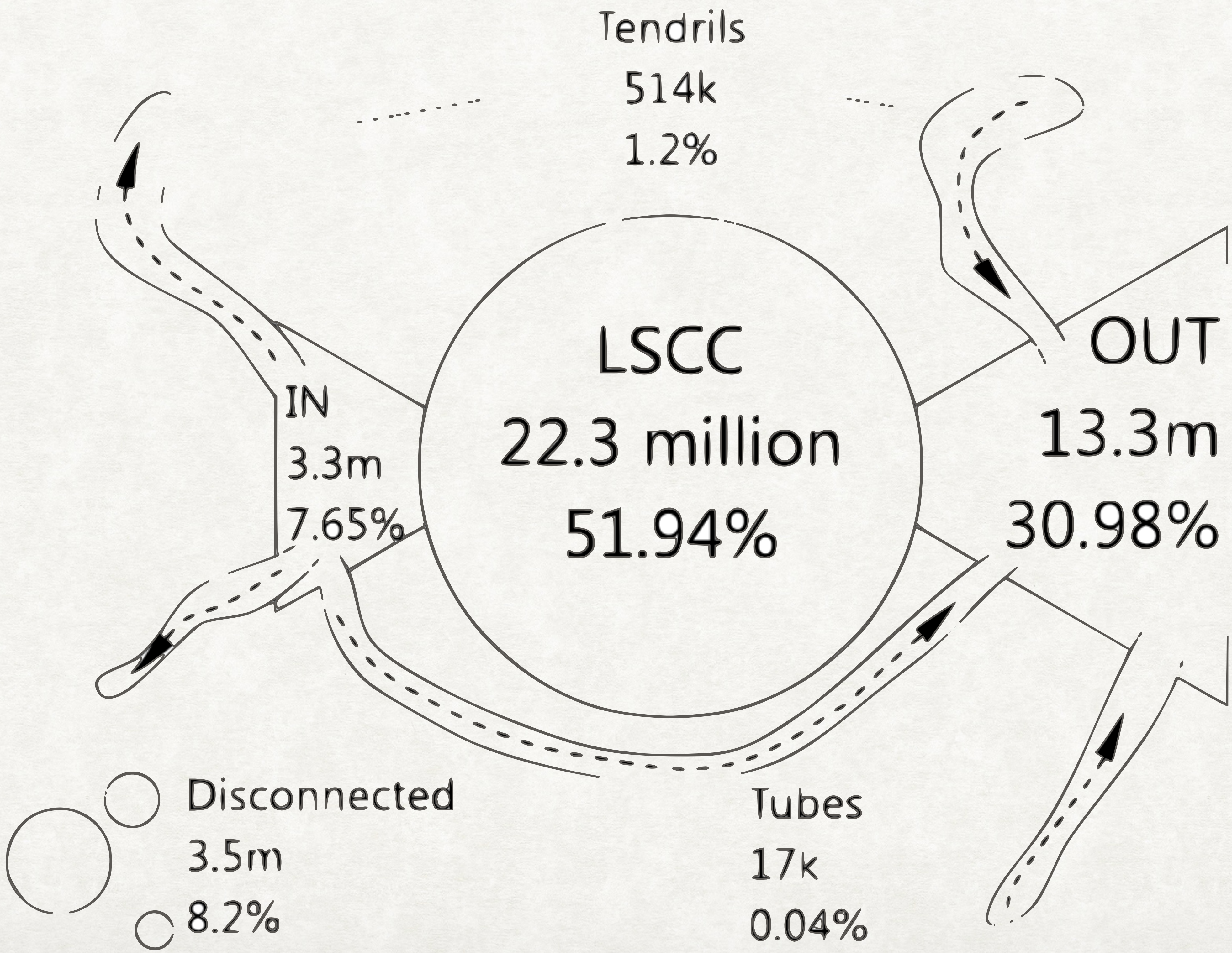
In-degree distribution (PLD level)

Out-degree distribution (PLD level)

While the exact distribution of incoming and outgoing links is not completely understood, a power law (i.e., $f(x) = ax^{-k}$) seems a good approximation.

THE BOWTIES STRUCTURE

AT THE PLD LEVEL



THE DEEP WEB

THE PART OF THE WEB THAT IS DIFFICULT TO INDEX

- Web pages that are difficult or impossible to index are part of the **deep** or **hidden** web.
- Not to be confused with the dark web/darknet, a small portion of the deep web that has been purposefully made inaccessible.
- It is estimated to be larger than the conventional web.
- Usually contains private sites (where login might be needed or there are no incoming links), form results, and scripted pages (e.g., were the links are generated by scripts).

WEB CRAWLERS

WEB CRAWLERS

AKA SPIDERS

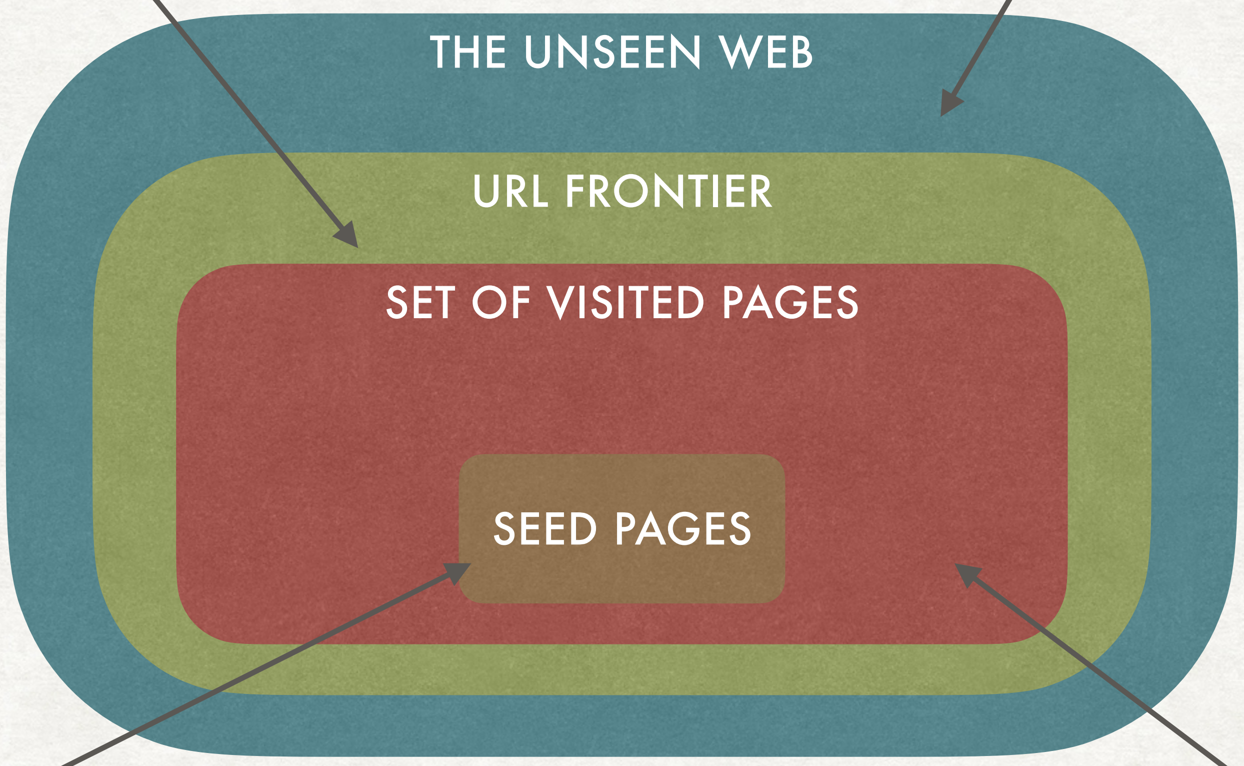
- Web crawling is the process of gathering pages from the Web to index them.
- The process is carried on by *web crawlers*, also called *spiders*.
- While retrieving a single web page is simple…
- …web crawling must take into account the scale of the web…
- …and the fact that the content to index is not under the control of the people building the index.

VISITING WEB PAGES

SEEN, UNSEEN, AND UNKNOWN PAGES

Set of pages with URL found by the crawler but not visited

Set of unvisited web pages for which the URL is not known



A set of known web pages from which the crawling starts

Set of pages that the crawler has visited

ESSENTIAL PROPERTIES OF A WEB CRAWLER

EVERY WEB CRAWLER MUST HAVE THEM

- **Robustness**. A web crawler must not be blocked by *spider traps*, web pages built to force a crawler to fetch an infinite amount of pages from a specific domain.
  - Sometimes spider traps are not malicious. Just imagine a "calendar" page that every time allows to go to the "next month" and generates the new pages dynamically.
- **Politeness**. A web crawler cannot overload a web server with requests. All requests to a domain must be adequately spaced in time and policies like the one in "robot.txt" must be adhered to.

ROBOT.TXT

WHAT WE CAN INDEX

A robot.txt file in a web server provides some information on what a crawler is allowed to index

```
User-agent: *
Disallow: /cgi-bin/
Disallow: /tmp/
Disallow: /private/

User-agent: BadBot
Disallow: /

User-agent: GoogleBot
Disallow:

Sitemap: http://www.example.com/sitemap.xml
```

Which crawlers should apply the following directives

Directories that should not be indexed

Directives for a specific bot: disallow everything

Directives for a specific bot: allow everything

File containing the set of URL available for crawling

GOOD PROPERTIES OF A WEB CRAWLER

A WEB CRAWLER SHOULD, IF POSSIBLE, HAVE THEM

- **Distributed**. Indexing the entire web from a single machine is infeasible, the web crawler should be able to execute from multiple machines
- **Scalable**. It should be possible to increase the crawl rate by simply adding more machine and bandwidth.
- **Performance and efficiency**. The crawler should try to make efficient use of system resources (e.g., by not blocking when waiting for the response from a server).

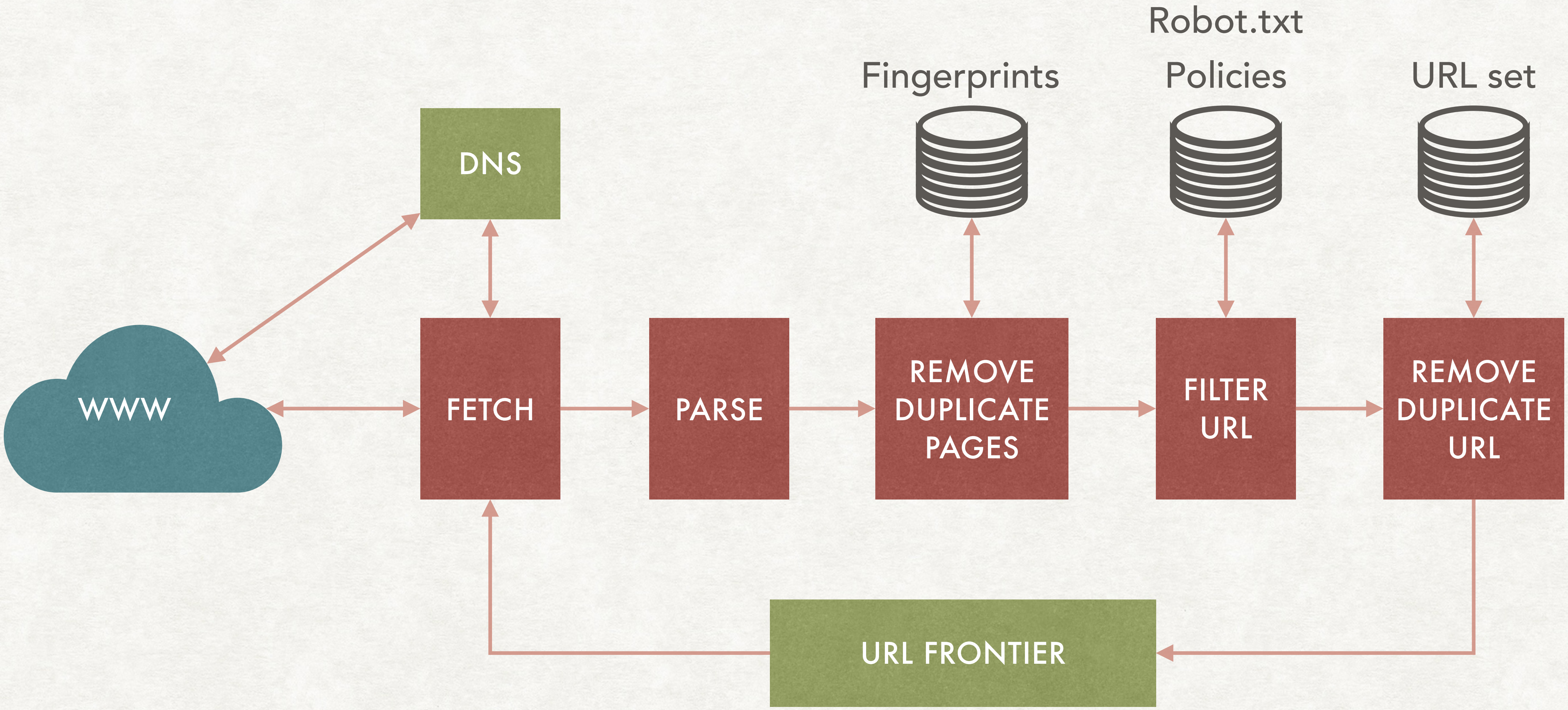
GOOD PROPERTIES OF A WEB CRAWLER

A WEB CRAWLER SHOULD, IF POSSIBLE, HAVE THEM

- **Quality**. The crawler should have a bias toward "useful" pages.
- **Freshness**. The content on the web is always changing, thus the crawler should revisit already visited pages to obtain a fresh copy.
    - A crawler should visit a page with a frequency that approximate the rate of change of the page.
- **Extensible**. There might be new data format, new protocols, etc. and the crawler should be able to be extended to handle them.

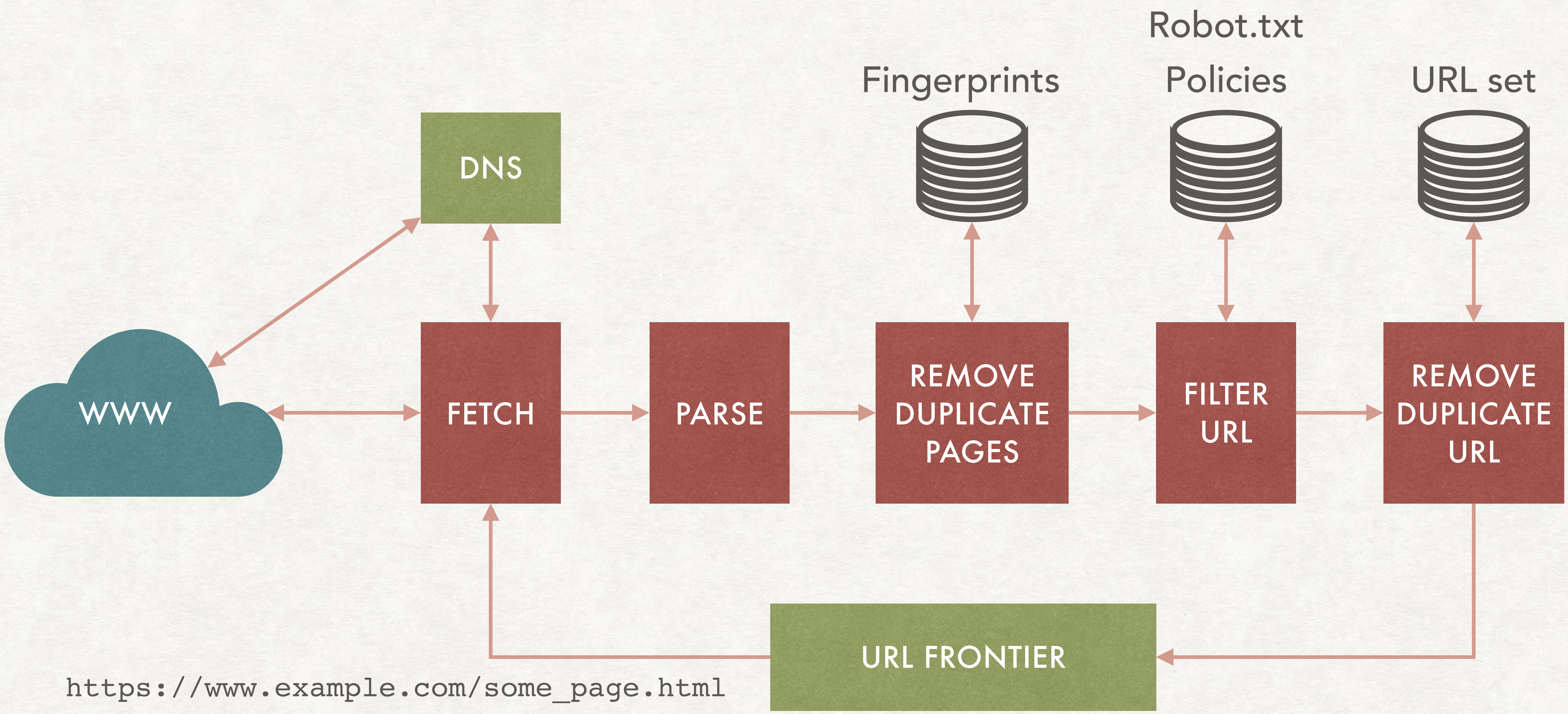
ARCHITECTURE OF A WEB CRAWLER

BASIC STRUCTURE



ARCHITECTURE OF A WEB CRAWLER

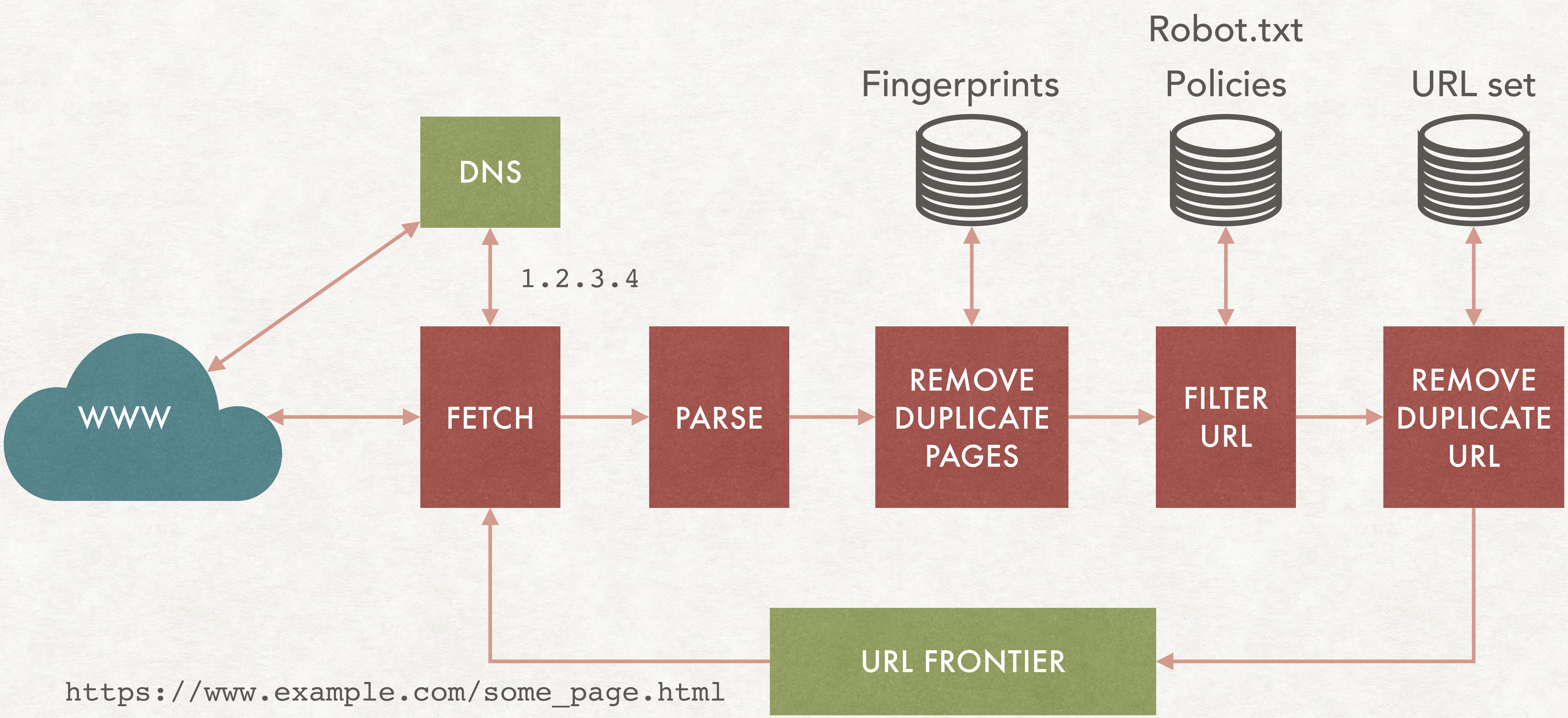
BASIC STRUCTURE



The fetch module retrive an URL to crawl from the URL frontier

ARCHITECTURE OF A WEB CRAWLER

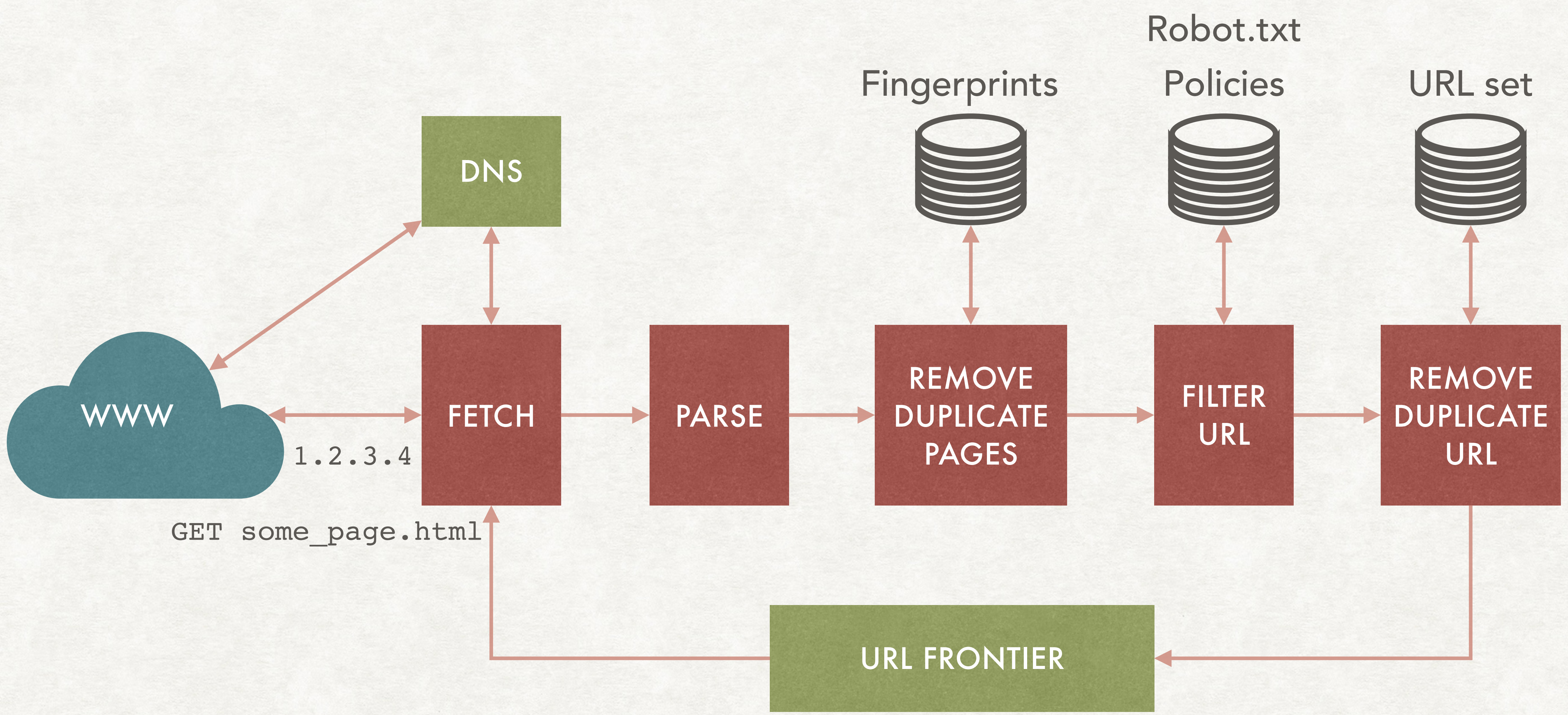
BASIC STRUCTURE



The DNS resolver find which IP address corresponds to www.example.com

ARCHITECTURE OF A WEB CRAWLER

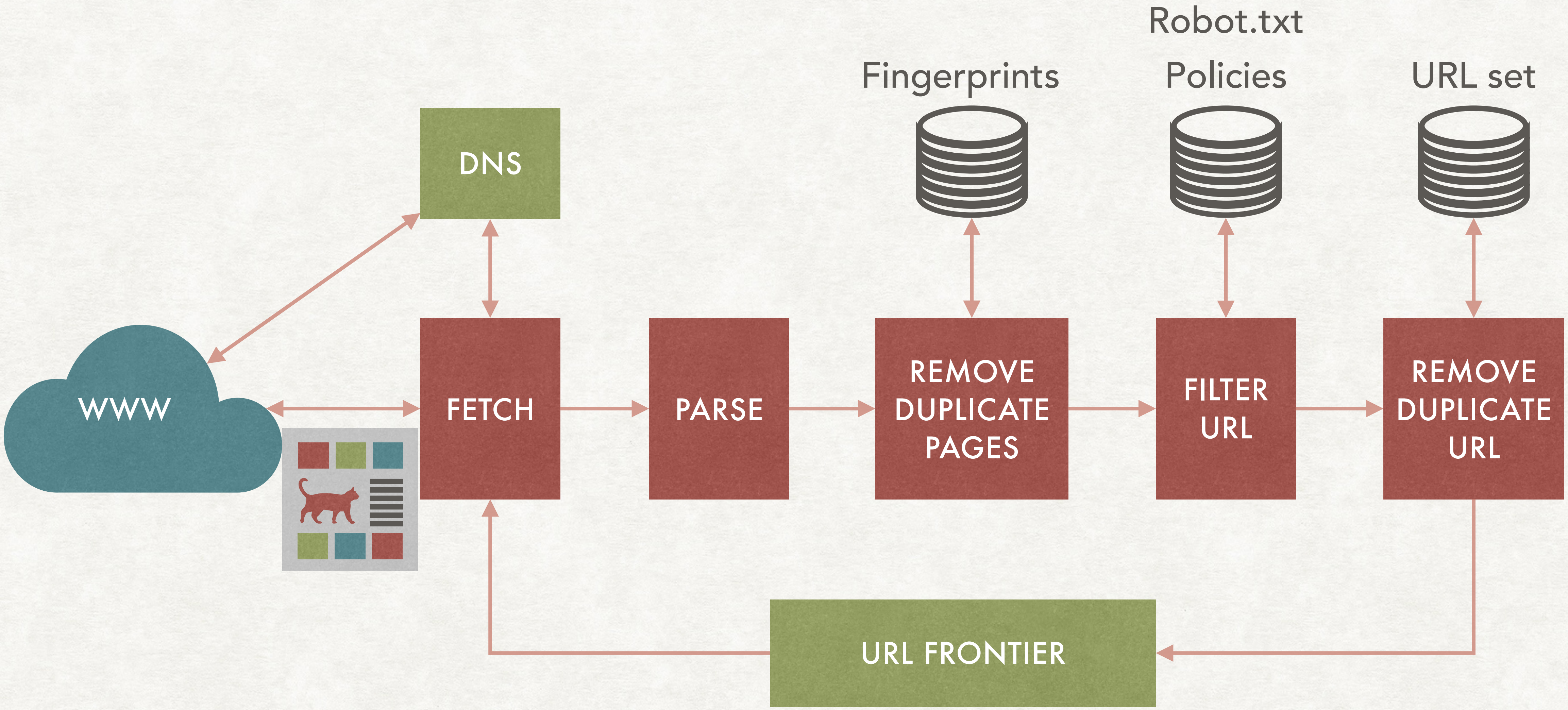
BASIC STRUCTURE



The fetch module asks for the web page to the server

ARCHITECTURE OF A WEB CRAWLER

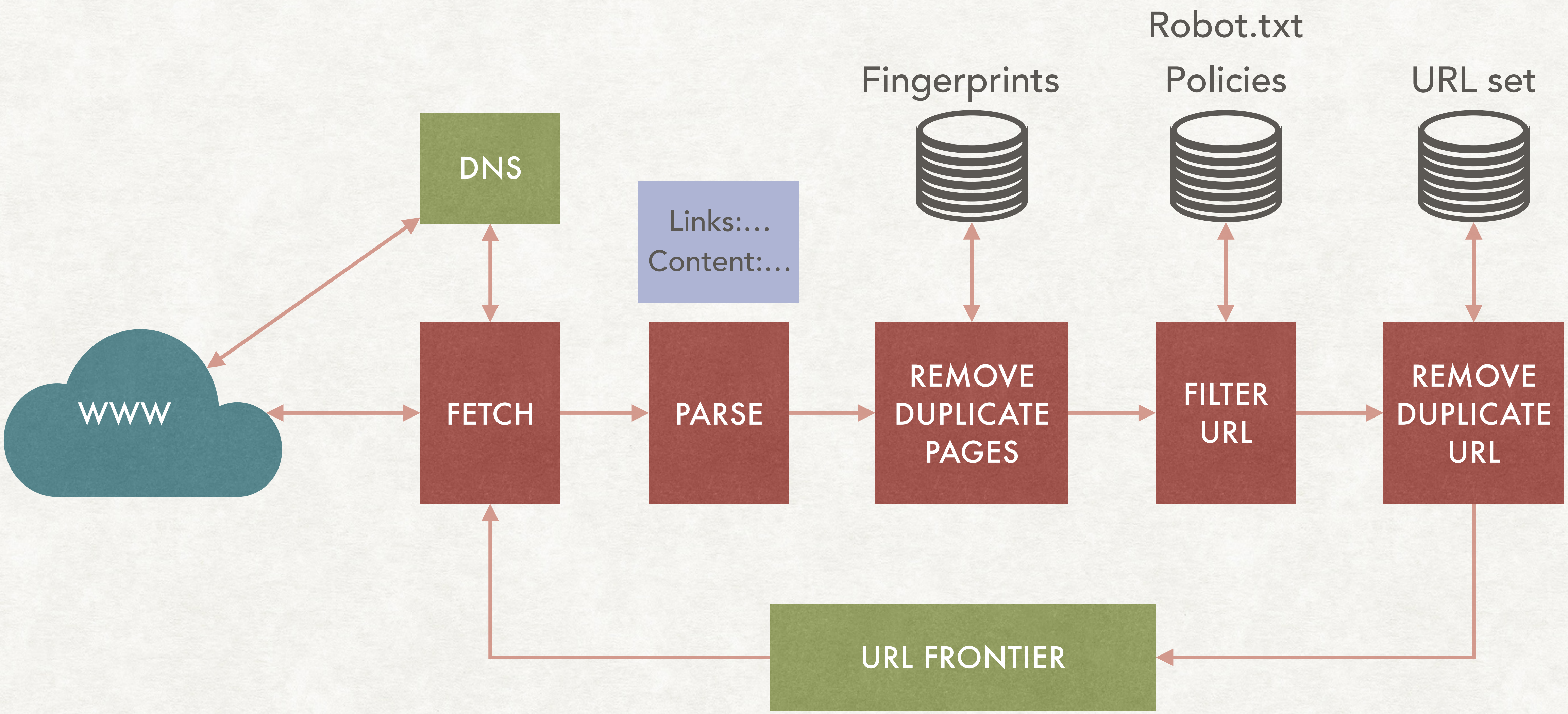
BASIC STRUCTURE



The fetch module receives the web page

ARCHITECTURE OF A WEB CRAWLER

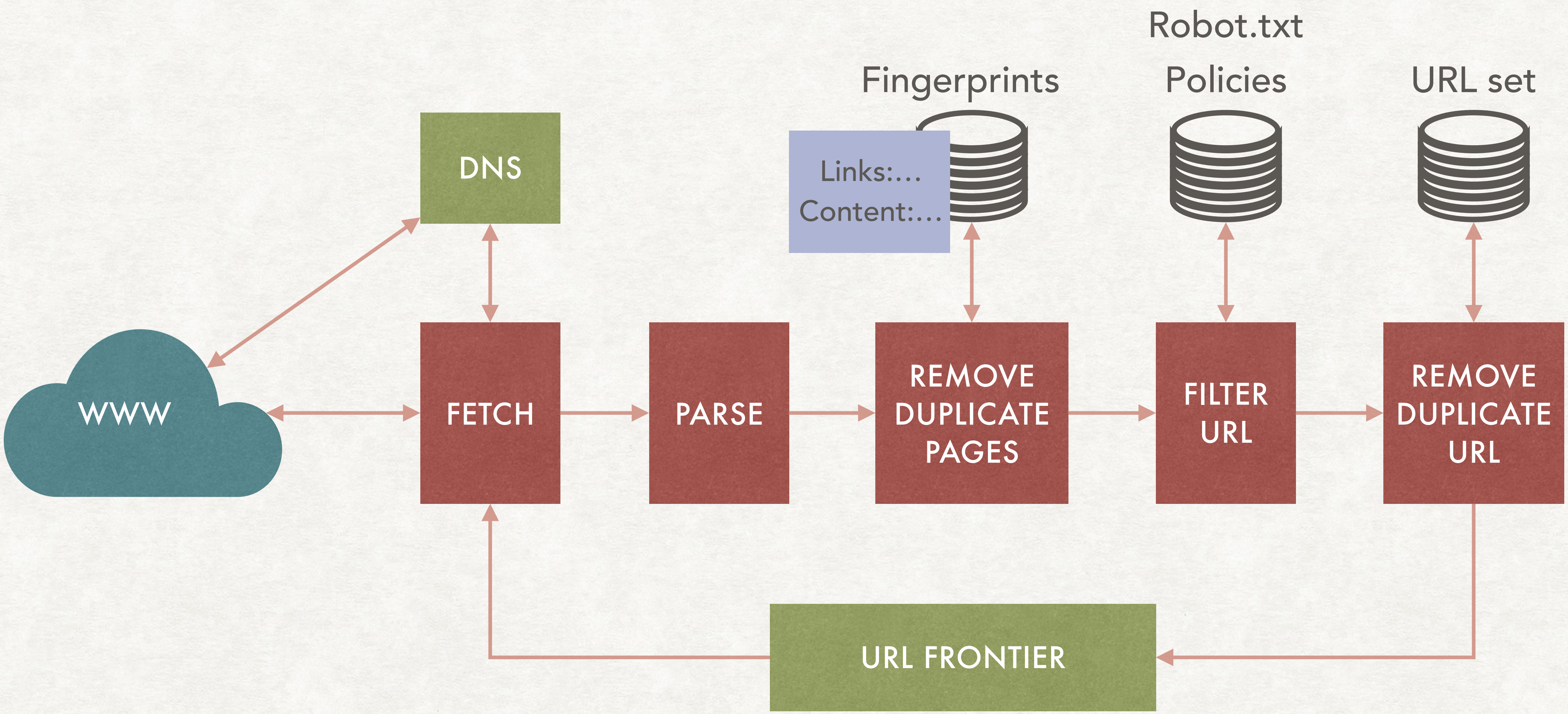
BASIC STRUCTURE



The page is parsed, the links and the main content extracted

ARCHITECTURE OF A WEB CRAWLER

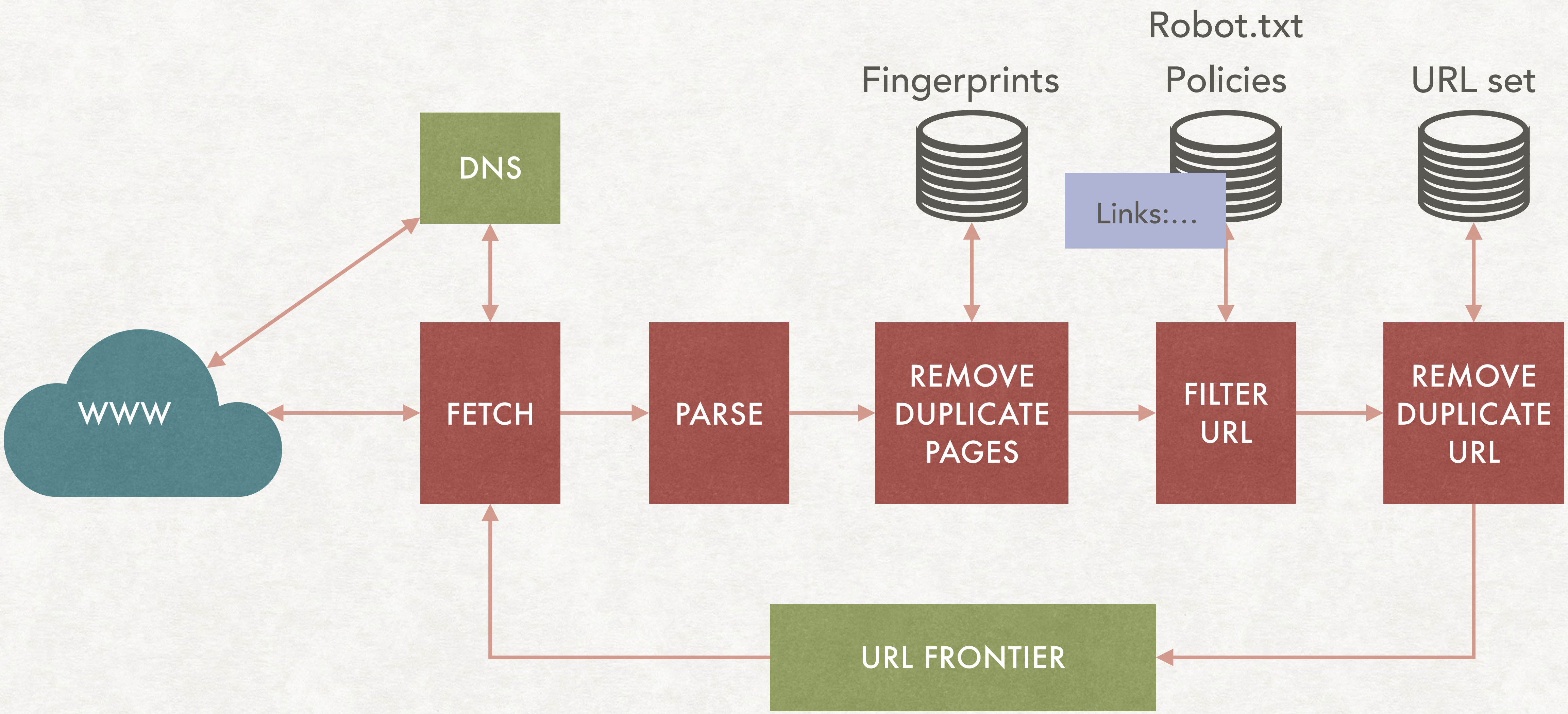
BASIC STRUCTURE



Before indexing the page is checked with a set of "fingerprints" of other pages to verify if it is a duplicate.

ARCHITECTURE OF A WEB CRAWLER

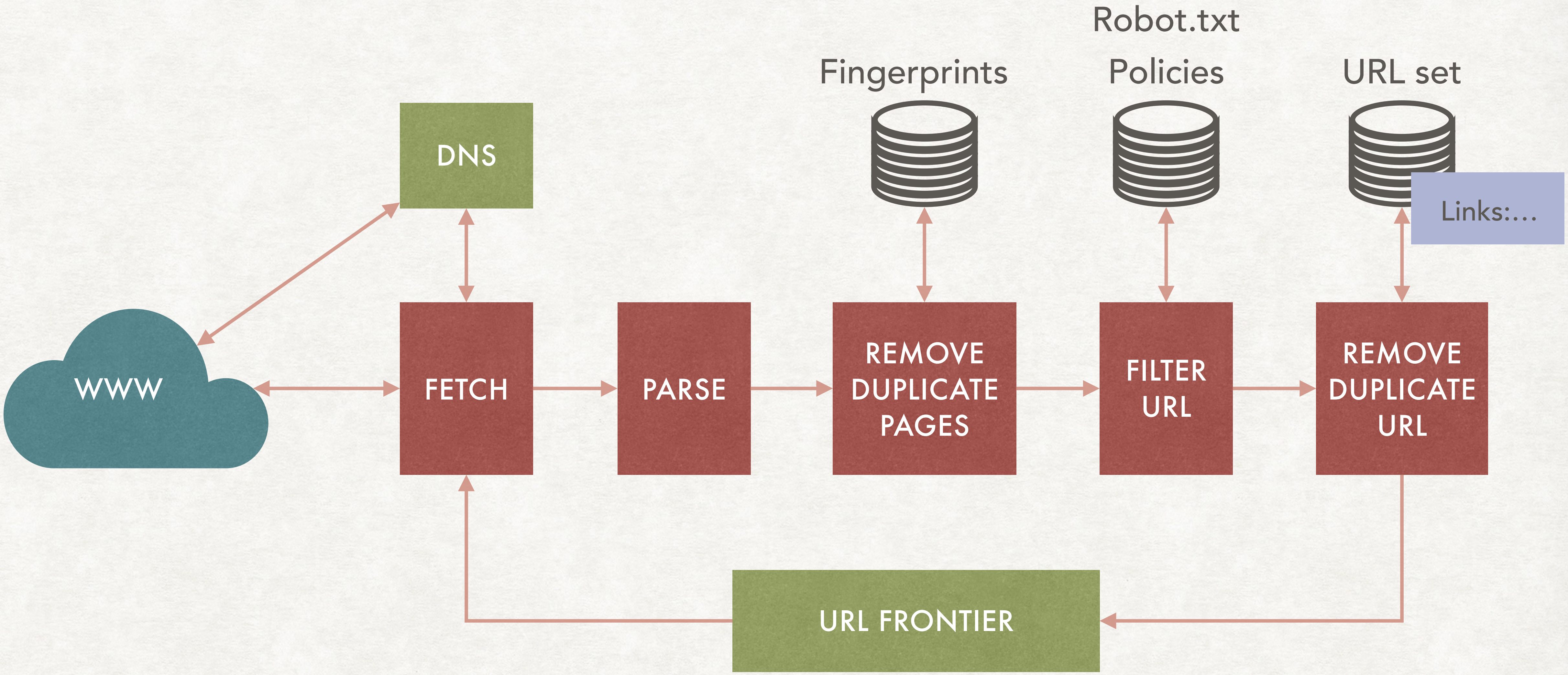
BASIC STRUCTURE



The newly extracted URL are normalised and filtered to eliminate the ones that should not be crawled.

ARCHITECTURE OF A WEB CRAWLER

BASIC STRUCTURE



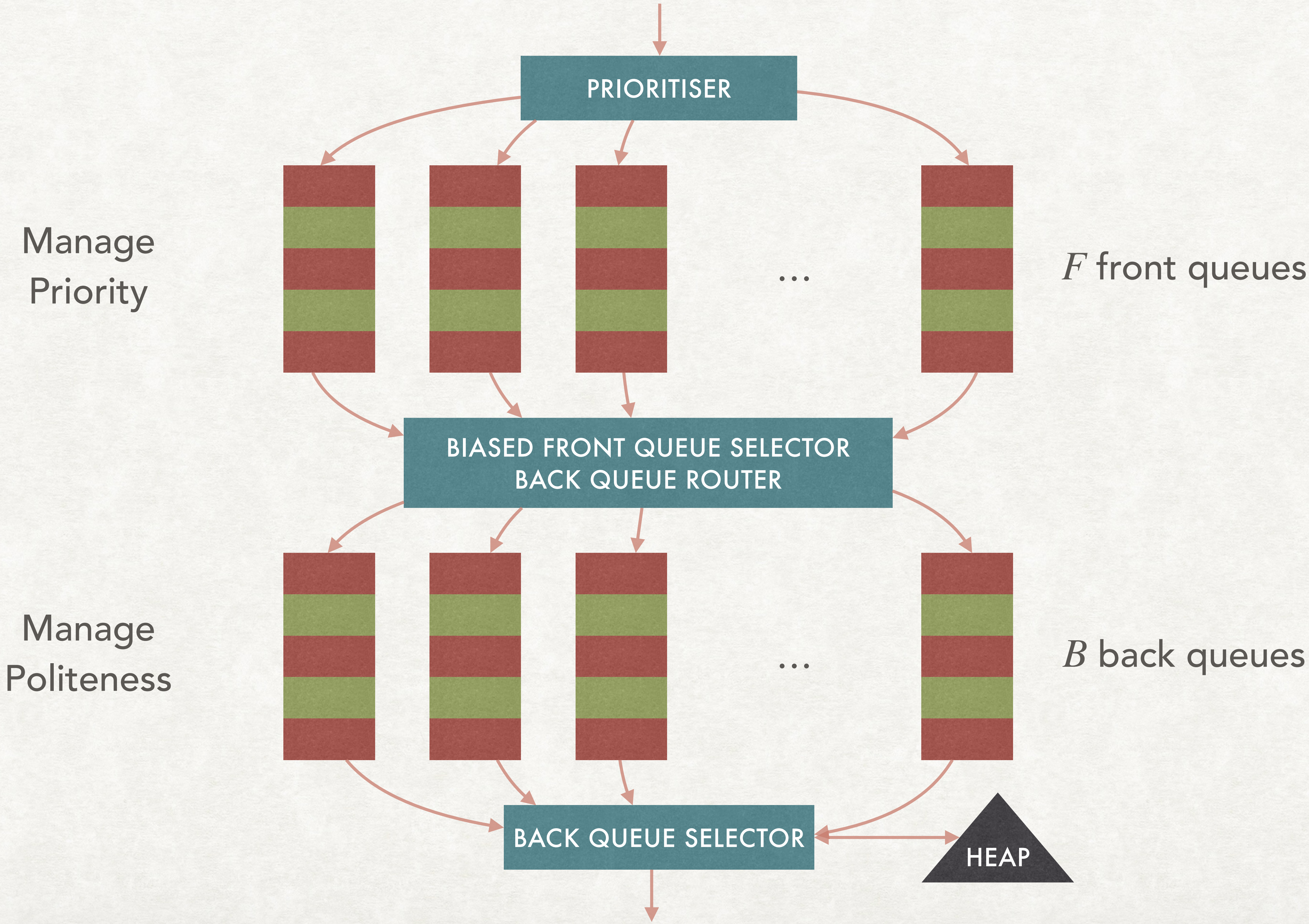
Finally, before being inserted into the URL frontier (the set of URL to visit), already visited URL are removed.

SELECTION OF THE NEXT URL

REQUIREMENTS

- We need an architecture that allows to:
  - Keep only one connection open to the host.
  - Ensure a waiting time of at least a few second between requests.
  - Have a bias for pages with higher priority.
- We present one possible architecture for achieving these goals.
- Multiple threads can extract URL from the URL frontier.

THE URL FRONTIER



FRONT AND BACK QUEUES

FOR PRIORITY AND POLITENESS

- The prioritiser assign an integer priority between 1 and F to each new URL
- There are F front FIFO queues (one for each priority).
- Each of the B back queues has the properties that:
  - The queue is non-empty while crawling is in progress.
  - Each queue contains URL from a single host.
  - To do so we need to keep a mapping from hosts to queues.

FRONT AND BACK QUEUES

FOR PRIORITY AND POLITENESS

- We keep an heap that returns the minimum time to wait to contact again an host.
- We extract the top of the heap, wait the required time, and extract a new URL from the corresponding queue.
- If the queue is now empty, then a new URL is taken from the front queues in a biased manner (i.e., higher probability of being selected to higher priority queues).
    - If the URL is from an host with an already assigned queue then it is inserted in that queue, and the extraction is repeated.

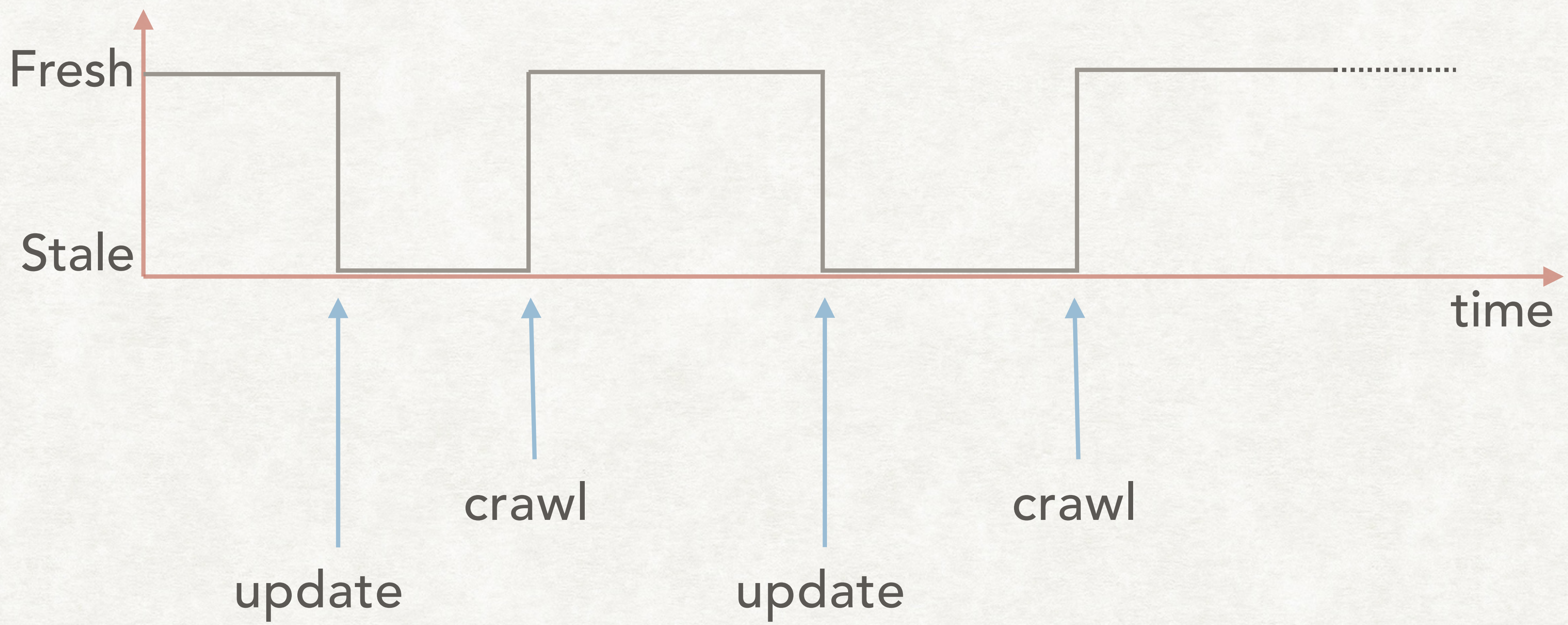
FRESHNESS

AND HOW TO SELECT WHAT TO RE-CRAWL

- A HEAD request is a kind of request where the server send some information about a page, but not the page itself. Among the information there is the "*Last-modified*" time.
- We can use HEAD requests to check pages for freshness.
- However, it is impossible to constantly check all pages.
- We must decide a policy on what pages to check.
- We have two metrics: **freshness** and **age**.

FRESHNESS

A BINARY WAY OF MEASURING "OLD" PAGES



A page is fresh if the crawler has the most recent copy of the page, otherwise the page is stale.

Freshness = fraction of web pages that are currently fresh.

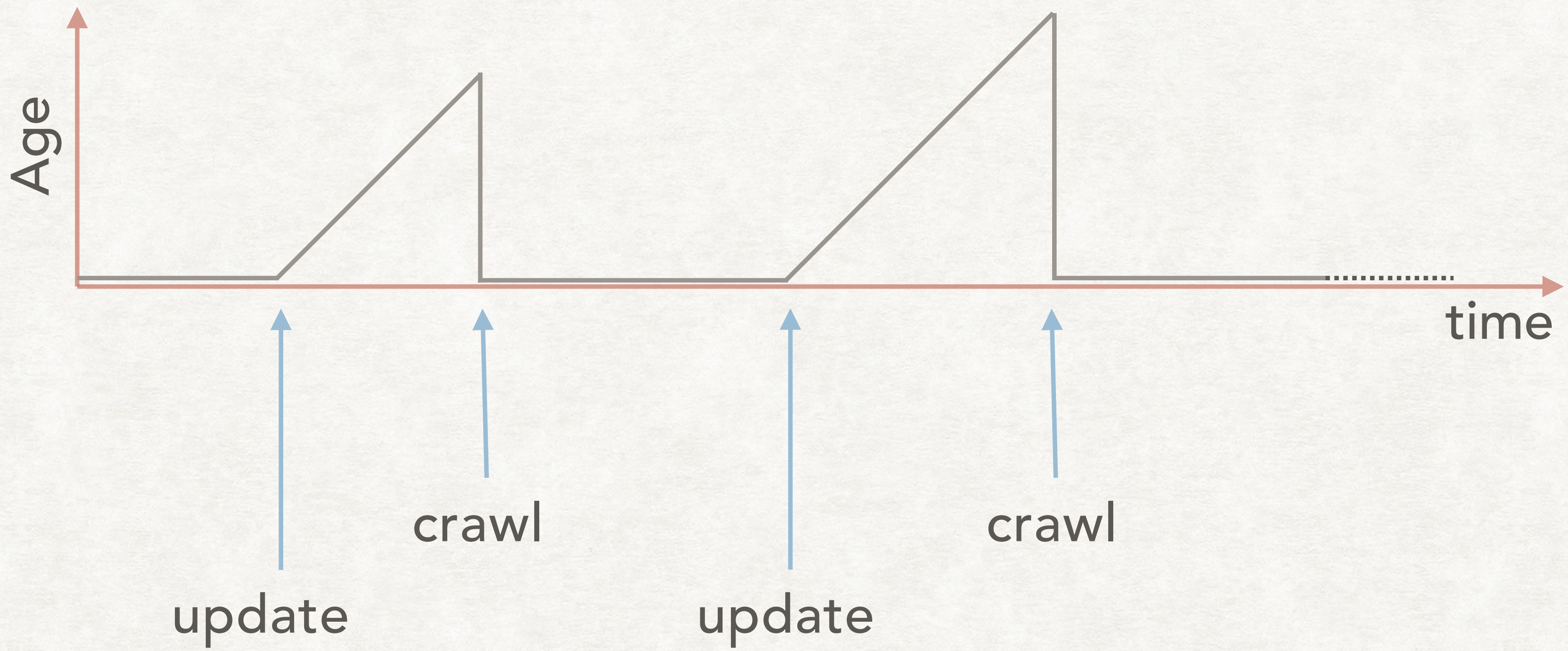
FRESHNESS

A BINARY WAY OF MEASURING "OLD" PAGES

- Should we optimise for freshness?
- Actually there can be unintended consequences.
- Suppose that a page updates very frequently (e.g., every minute).
- You will almost always have a stale copy of the page.
- If you have limited resources for crawling then a good strategy would be to never crawl that page again: it will always be stale after a very short time.
- Which is not what the user want. Hence we can optimise for **age**.

AGE

A MORE REFINED WAY OF FINDING OUTDATED PAGES



A page start ageing when it is modified. Its age returns to 0 when it is crawled again.

Age = time passed since the first update after a crawl event.

AGE

A MORE REFINED WAY OF FINDING OUTDATED PAGES

Suppose that a page is updated λ times a day.

Then its expected age at time t after it was visited last time is:

$$\text{Age}(\lambda, t) = \int_0^t P(\text{Change at time } x)(t - x)\, dx$$

The probability of a page changing at a certain time x can be estimated: according to studies, the updates to a web page follows a Poisson distribution, hence we obtain:

$$\text{Age}(\lambda, t) = \int_0^t \lambda e^{-\lambda x}(t - x)\, dx$$

AGE

A MORE REFINED WAY OF FINDING OUTDATED PAGES

By trying to minimise the expected age of a set of pages we will visit them all.

$$\mathrm{Age}(\lambda, t) = \frac{t + \lambda e^{-\lambda t} - 1}{\lambda} \qquad \frac{\partial^2 \mathrm{Age}(\lambda, t)}{\partial t^2} = \lambda e^{-\lambda t}$$

Notice that the rate of increase of the age function (its second derivative) is always positive for $\lambda > 0$ (which is always the case).

This means that *not* visiting a web page has an increasing cost the older the page gets. We will never conclude that we do not have to visit a web page.

DUPLICATES AND NEAR-DUPLICATES

THE PROBLEM

DUPLICATED WEB PAGES

- Studies show that about 30% of the crawled pages are duplicates or near-duplicates of the other 70%[1].
- Duplicates can be created by spam or plagiarism…
- …but also via mirror sites.
- Duplicates or near-duplicates provide very little information to the user while consuming resources for crawling and indexing.
- There exist algorithms to mitigate this problem, without comparing each document across all already-indexed documents.

[1] Fetterly, Dennis, Mark Manasse, and Marc Najork. "On the evolution of clusters of near-duplicate web pages.”

DETECTING EXACT DUPLICATES

CHECKSUMMING

The detection of exact duplicate is relatively easy;
it can be performed by comparing the *checksums* of the documents

One of the simplest kinds of checksums is
to simply sum all the bytes in the document

"The quick brown fox jumps over the lazy dog"

84 104 101 32 113 117 105 99 107 32 … 32 100 111 103 $\xrightarrow{\text{Sum}}$ 4057

There are more complex checksum algorithms where the position of the bytes
is considered (like CRC - cyclic redundancy check),

NEAR-DUPLICATES

WHAT THEY ARE AND HOW TO DETECT THEM

- Detecting near-duplicates is more complex…
- …but even *defining* them is more problematic:
- E.g., same text but different advertising/formatting
- Slight difference in text due to small edits
- In general a similarity measure is defined…
- …and two documents are considered near-duplicates above a certain threshold.

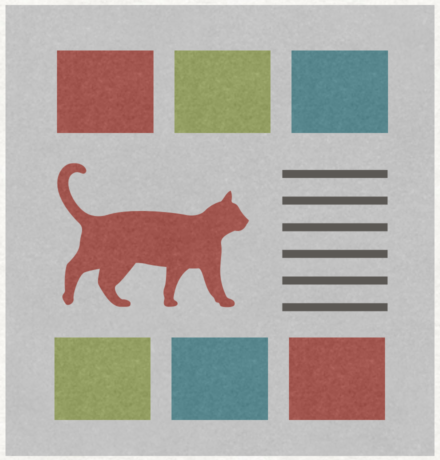
NEAR-DUPLICATES

TWO SCENARIOS

- Detecting near-duplicates can happen in two scenarios:
- **Search**. When the goal is to find the duplicates of a given document.
- **Discovery**. When, given a collection, the goal il to find all pairs of duplicates or near duplicates.
- Similarity-based IR techniques can be used in the *search* scenario.
- For the *discovery* scenario more efficient techniques are usually employed, e.g., **fingerprints**.

FINGERPRINTS

A POSSIBLE ALGORITHM



Web page

All non-word content is removed
The document is parsed into words

The quick brown fox jumps over the lazy dog

Words grouped in n-grams for some n

The quick brown
quick brown fox
brown fox jumps
fox jumps over
jumps over the
over the lazy
the lazy dog

Continues
in the next slide

FINGERPRINTS

A POSSIBLE ALGORITHM

Words grouped in n-grams for some n

The quick brown

quick brown fox

brown fox jumps

fox jumps over

jumps over the

over the lazy

the lazy dog

A subset of n-grams is selected

quick brown fox

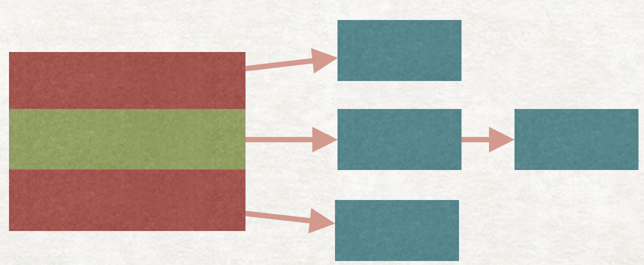
fox jumps over

The n-grams are hashed

1490

1400

The hashes are stored
in an inverted index



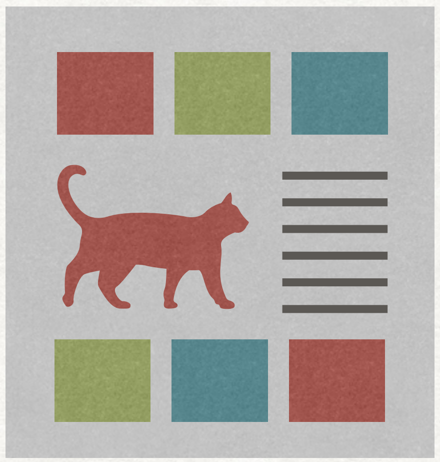
FINGERPRINTS

HOW TO SELECT A SUBSET

- Two documents are considered near-duplicates they share enough n-grams (by measuring, for example, the Jaccard coefficient).
- It is essential to have a “good” way of selecting which subset of n-grams to keep:
  - Random selection is a bad choice: the overlap between randomly selected n-grams of identical documents can be low!
  - A better choice is to select all n-grams starting with the same letter.
  - Another choice is to select all n-grams with hash value equal to $0 \bmod p$ for some choice of p.

SIMHASH

A MORE RECENT FINGERPRINTING TECHNIQUE



Web page

The	2
Quick	1
Brown	1
Fox	1

Extract a set of features (e.g., words) each with a weight (e.g., frequency)

For each word compute a unique hash of b bits (the desired size of the fingerprint)

The	Quick	Brown	Fox
0101	1100	1001	0001

Continues in the next slide

SIMHASH

A MORE RECENT FINGERPRINTING TECHNIQUE

The	2
Quick	1
Brown	1
Fox	1

The	Quick	Brown	Fox
0101	1100	1001	0001

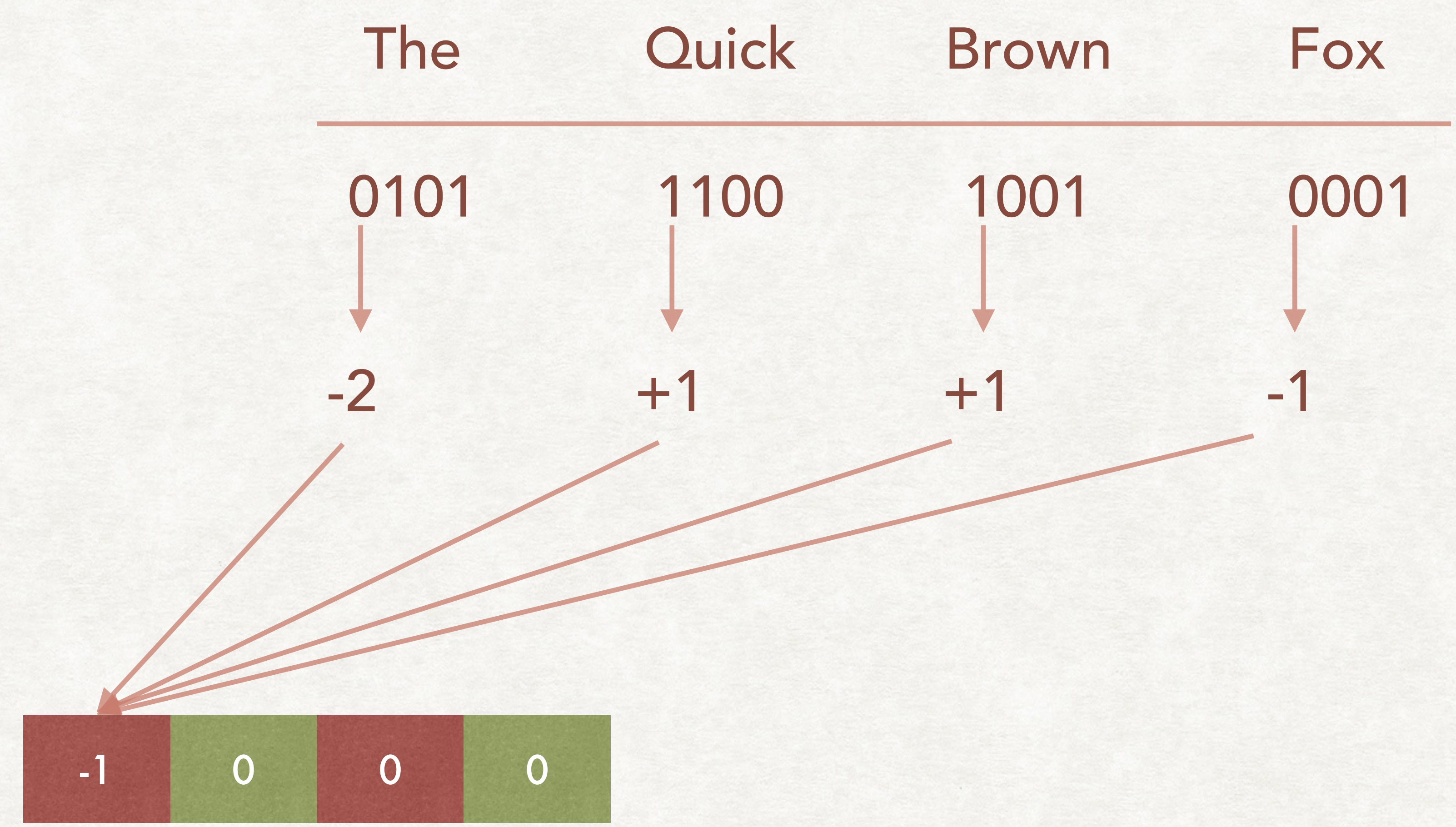
Start with a vector of size b with all positions initially set to 0

0	0	0	0

SIMHASH

A MORE RECENT FINGERPRINTING TECHNIQUE

The	2
Quick	1
Brown	1
Fox	1



Look at the first bit of the hash of every word.
Add the weight to the word if the bit is 1.
Subtract the weight of the word if the bit is 0

SIMHASH

A MORE RECENT FINGERPRINTING TECHNIQUE

The	2
Quick	1
Brown	1
Fox	1

The	Quick	Brown	Fox
0101	1100	1001	0001
+2	+1	-1	-1

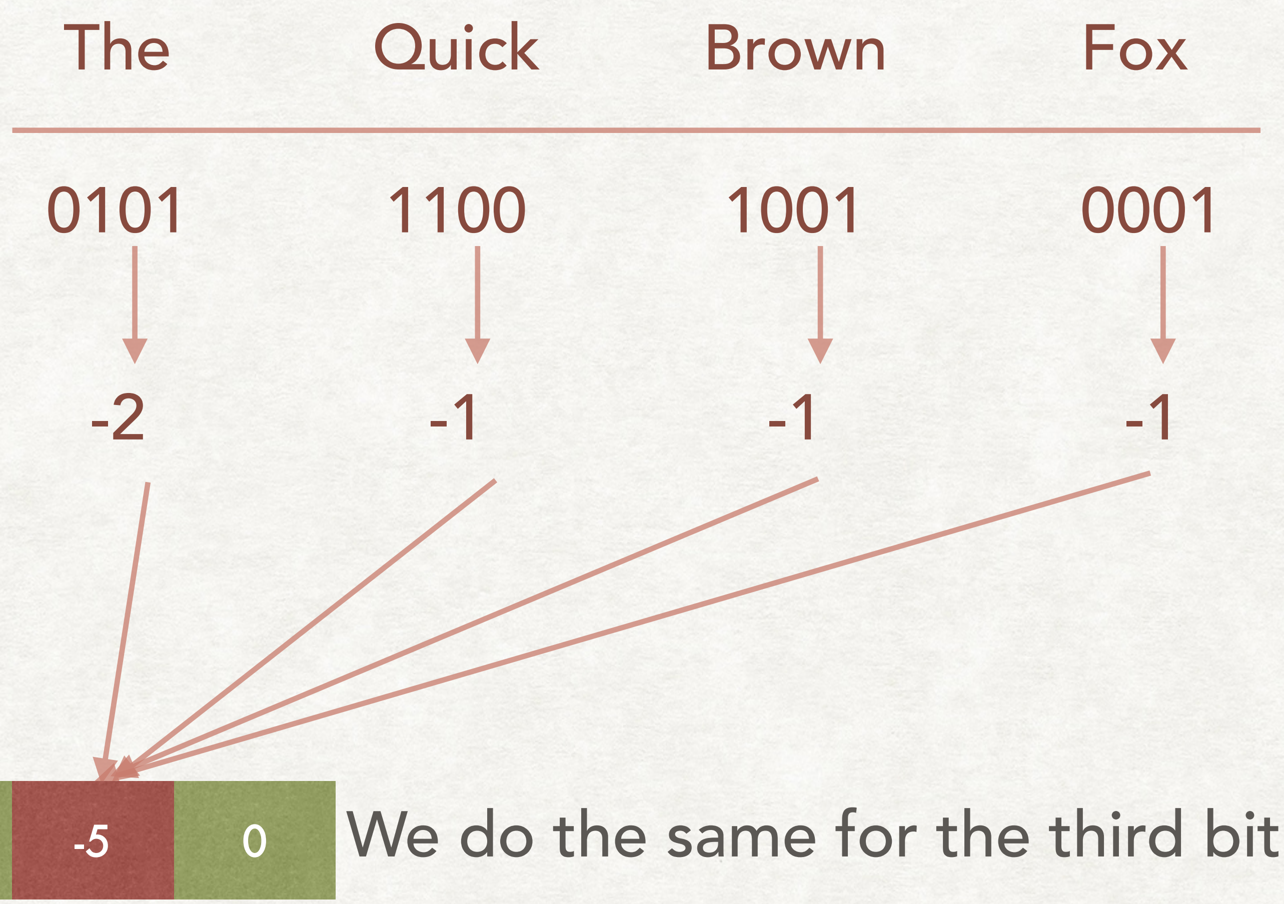
-1	1	0	0

We do the same for the second bit

SIMHASH

A MORE RECENT FINGERPRINTING TECHNIQUE

The	2
Quick	1
Brown	1
Fox	1



We do the same for the third bit

SIMHASH

A MORE RECENT FINGERPRINTING TECHNIQUE

The	2
Quick	1
Brown	1
Fox	1

The	Quick	Brown	Fox
0101	1100	1001	0001
+2	-1	+1	+1

The hash value of the document

-1	1	-5	3

We do the same for the fourth bit

We obtain a sequence of b bits by setting a bit to 1 for positive values and to 0 otherwise

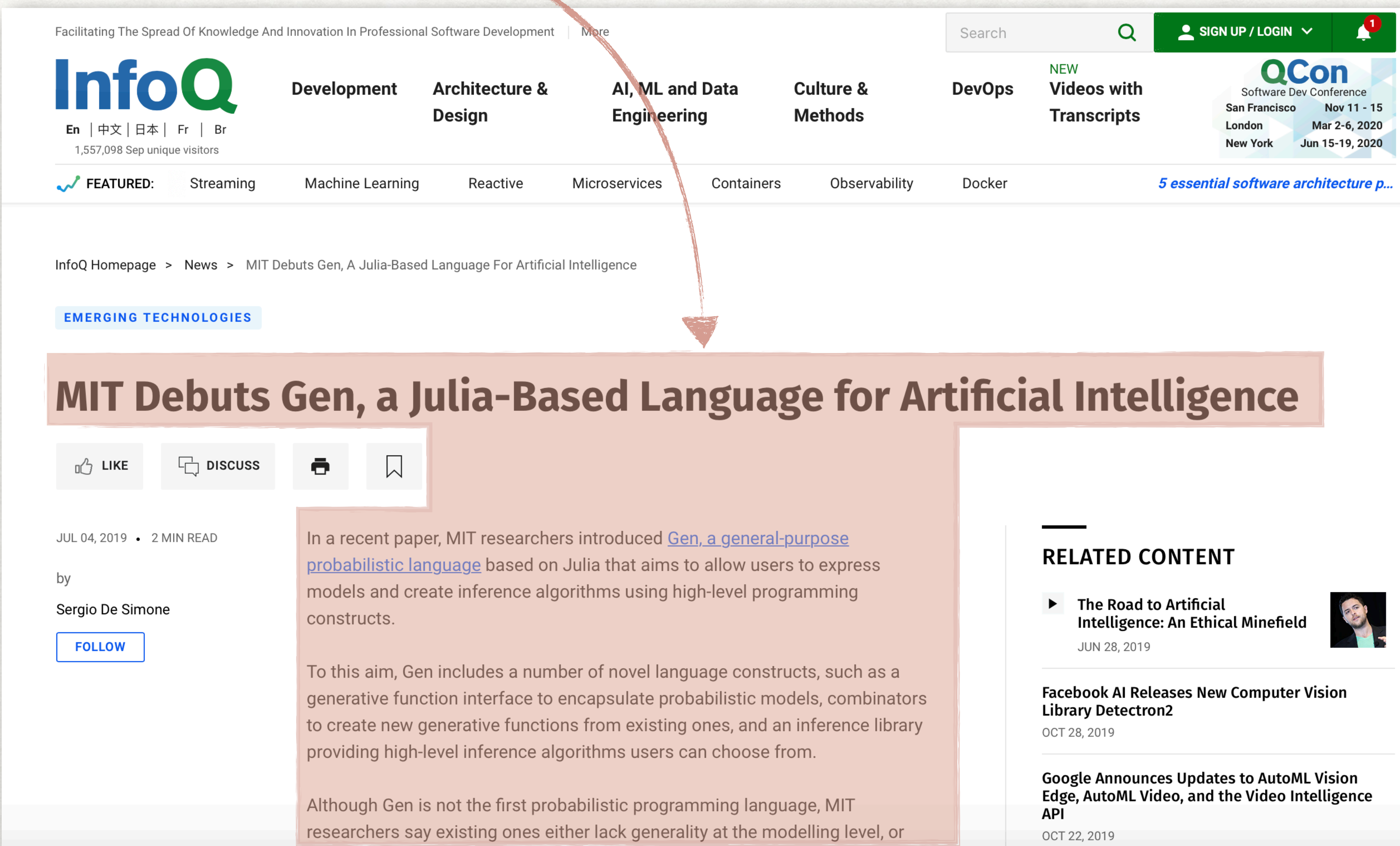
0	1	0	1

FINDING THE CONTENT

FINDING THE CONTENT

UNDERSTANDING THE PROBLEM

Main Content Block



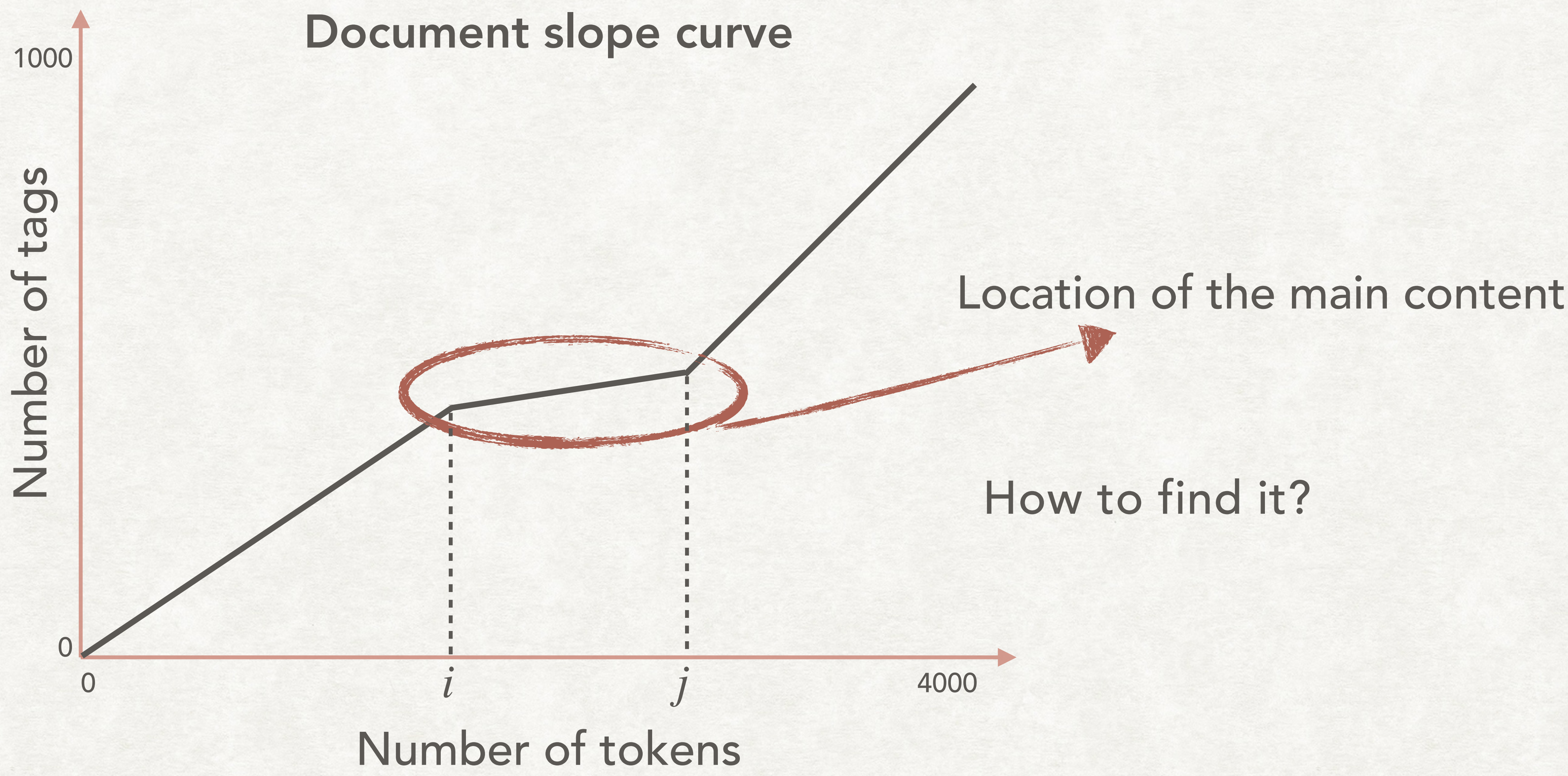
HOW TO FIND WHERE THE CONTENT IS

TAGS AND TOKENS

- The main content of the page might be only a fraction of the total area. The rest is advertisement, navigation links, etc.
- From the point of view of the user the rest is *noise* that can have a negative effect on the ranking.
- We need a way to identify the non-main content of the page and either ignore it or reduce its weight.
- An observation is that, usually, the main content of the page contains less tags than the rest of the page.

HOW TO FIND WHERE THE CONTENT IS

TAGS AND TOKENS



How to find it?

HOW TO FIND WHERE THE CONTENT IS

TAGS AND TOKENS

Document as a binary vector of length N (the number of tokens) with:

$$b_k = \begin{cases} 1 & \text{if the } k\text{-th term is a tag} \\ 0 & \text{otherwise} \end{cases}$$

Find two "cutting points" i and j with $1 \leq i < j \leq N$ maximising:

$$\underbrace{\sum_{k=1}^{i-1} b_k}_{\text{Tags before content}} + \underbrace{\sum_{k=i}^{j} (1 - b_k)}_{\text{non-tags in the content}} + \underbrace{\sum_{k=j+1}^{N} b_k}_{\text{Tags after content}}$$

ANOTHER POSSIBILITY

LOOKING AT THE DOM

- To parse a webpage a browser construct a representation using the HTML tags.
- This representation is the DOM (Document Object Model)
- It is a tree-like structure that can be navigated to find the major components of a web page.
- A set of heuristics and filtering techniques can be used to remove images, advertising, and leave only the content.
- It is also possible to analyse the visual feature of a page to identify the location of the main content.

ANOTHER POSSIBILITY

LOOKING AT THE DOM

```
▶ <script type="text/javascript">…</script>
▼ <div class="infoq" id="infoq">
    <!-- ####### SITE START ######### -->
  ▶ <header class="header nocontent">…</header>
    <!-- ####### CONTENT START ######### -->
  ▼ <main>
    ▼ <article data-type="news" class="article">
      ▼ <section class="section container white">
          ::before
        ▼ <div class="container__inner">
          ▶ <p class="crumbs">…</p>
          ▶ <div class="actions">…</div>
          ▶ <div class="actions heading__container article__heading">…</div>
          ▶ <p id="translated_jp" class="article__translated">…</p>
          ▶ <script type="text/javascript">…</script>
          ▶ <div class="article__actions actions">…</div>
          ▼ <div class="columns article__explore">
              ::before
            ▼ <div class="column article__main" data-col="4/6">
              ▶ <div class="column article__metadata metadata">…</div>
              ▼ <div class="article__content">
                  <!-- Start PSA Section -->
                  <!-- End PSA Section -->
                ▶ <div class="article__data">…</div>
                ▶ <input type="hidden" name value="Thank you for your review!" id="cr_messages_submitSuccess">
                ▶ <input type="hidden" name value="Rating is required" id="cr_messages_ratingRequired">
                ▶ <input type="hidden" name value="MIT Debuts Gen, a Julia-Based Language for Artificial Intelligence" id="cr_item_title">
                ▶ <input type="hidden" name value="Sergio De Simone" id="cr_item_author">
                ▶ <input type="hidden" name value="http://www.infoq.com/news/2019/07/mit-gen-probabilistic-programs/" id="cr_item_url">
                ▶ <input type="hidden" name value="news" id="cr_item_ctype">
                ▶ <input type="hidden" name value="en" id="cr_item_lang">
                ▶ <input type="hidden" name value="1562223600000" id="cr_item_published_time">
                ▶ <input type="hidden" name value="6230" id="cr_item_primary_topic">
                  <script type="text/javascript">ContentRating.readMessages(); ContentRating.readContentItem();</script>
                ▶ <script type="text/javascript">…</script>
                ▶ <div class="widget article__fromTopic topics">…</div>
                </div>
              ▶ <script type="text/javascript">…</script>
              ▶ <input type="hidden" name value="6230" id="cont_item_primary_topic">
```

This is the location of the main content of the example page, helpfully labeled with the class “article__content”

PAGERANK

HISTORY

BRIN, PAGE & GOOGLE

- PageRank is (part of) the algorithm used by Google in ranking the pages in its results.

- Developed in 1996 with the first paper on it published in 1998[2]: *"we present Google, a prototype of a large-scale search engine which makes heavy use of the structure present in hypertext "*

- Or, in other words, *"we take advantage of the link structure of the Web to produce a global "importance" ranking of every web page."*[1]

- The origin of PageRank can be traced back to methods in bibliometrics, sociometry, and possibly other fields.

[1]Page, L., Brin, S., Motwani, R. and Winograd, T., *The PageRank Citation Ranking: Bringing Order to the Web,* Stanford InfoLab, 1999

[2] Brin, S. and Page, L., *The anatomy of a large-scale hypertextual Web search engine,* Computer networks and ISDN systems, 30(1-7), pp.107-117, 1998

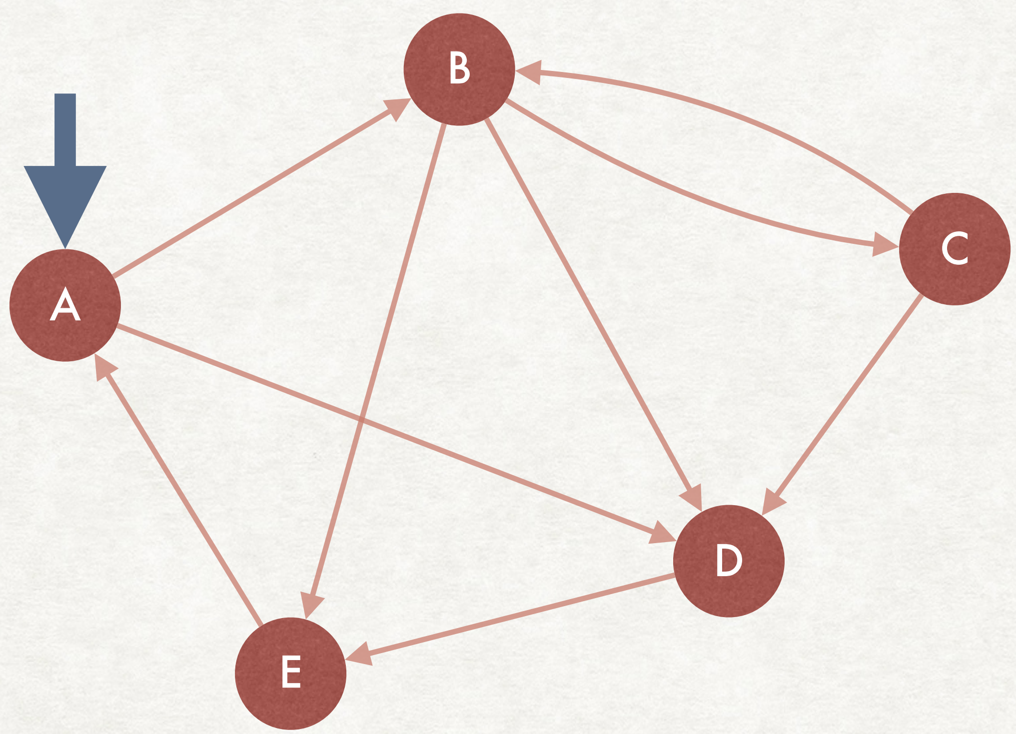
MAIN IDEA

USING LINKS TO GET SCORES

- We want to assign a value to each page that is independent from the query, i.e., a static score.
- We model a user randomly following links across web pages.
- What is the limit distribution of "where the user is" across all the pages?
- A user is without any memory of the page from where he/she came…
- …it seems like a case for using a Markov chain!

A SIMPLE EXAMPLE

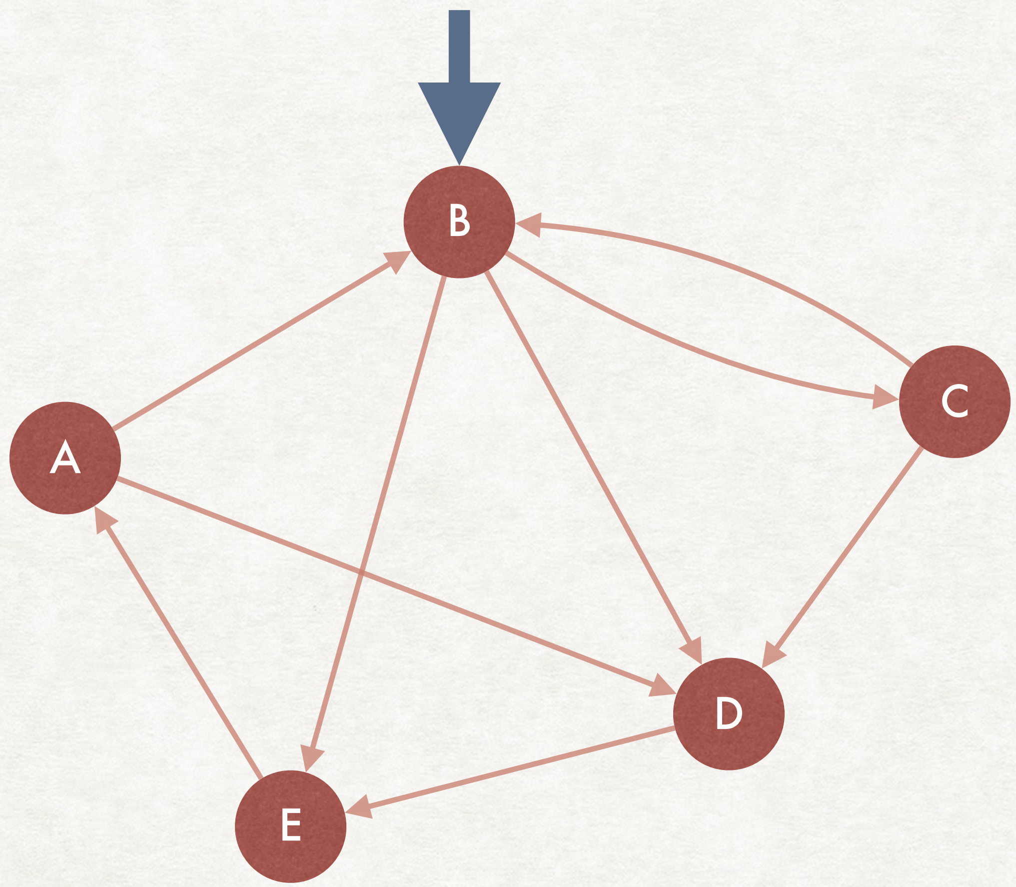
RANDOM WALK ON A GRAPH



We are visiting page A, where can move either to page B or D.
We select where to move uniformly at random.

A SIMPLE EXAMPLE

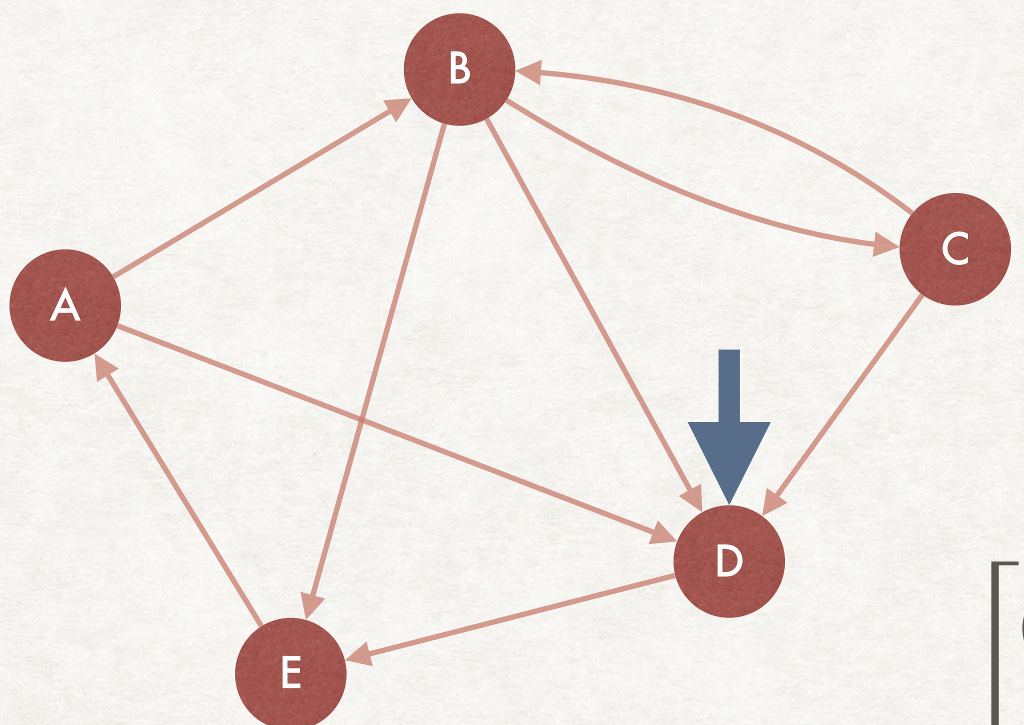
RANDOM WALK ON A GRAPH



We are visiting page B, where can move either to page C, D, or E.
We select where to move uniformly at random.

A SIMPLE EXAMPLE

RANDOM WALK ON A GRAPH



Probability of moving from node A to B

$$\begin{bmatrix} 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

We can formalise this random walk by defining a stochastic matrix

FORMALISATION AS A MARKOV CHAIN

AND THE STATIONARY DISTRIBUTION

Finding the probability distribution of the web page out idealised user is in then time tends to infinity

Is equivalent to

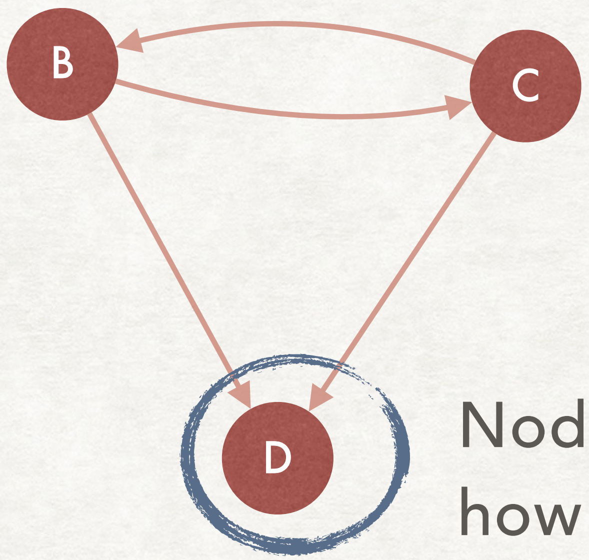
Finding the stationary distribution of the Markov chain with the following transition matrix:

$$R = \begin{bmatrix} 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

DANGLING NODES

NODES WITHOUT OUTGOING EDGES

A first problem that can appear in defining the stochastic matrix R is the presence of "dangling nodes"

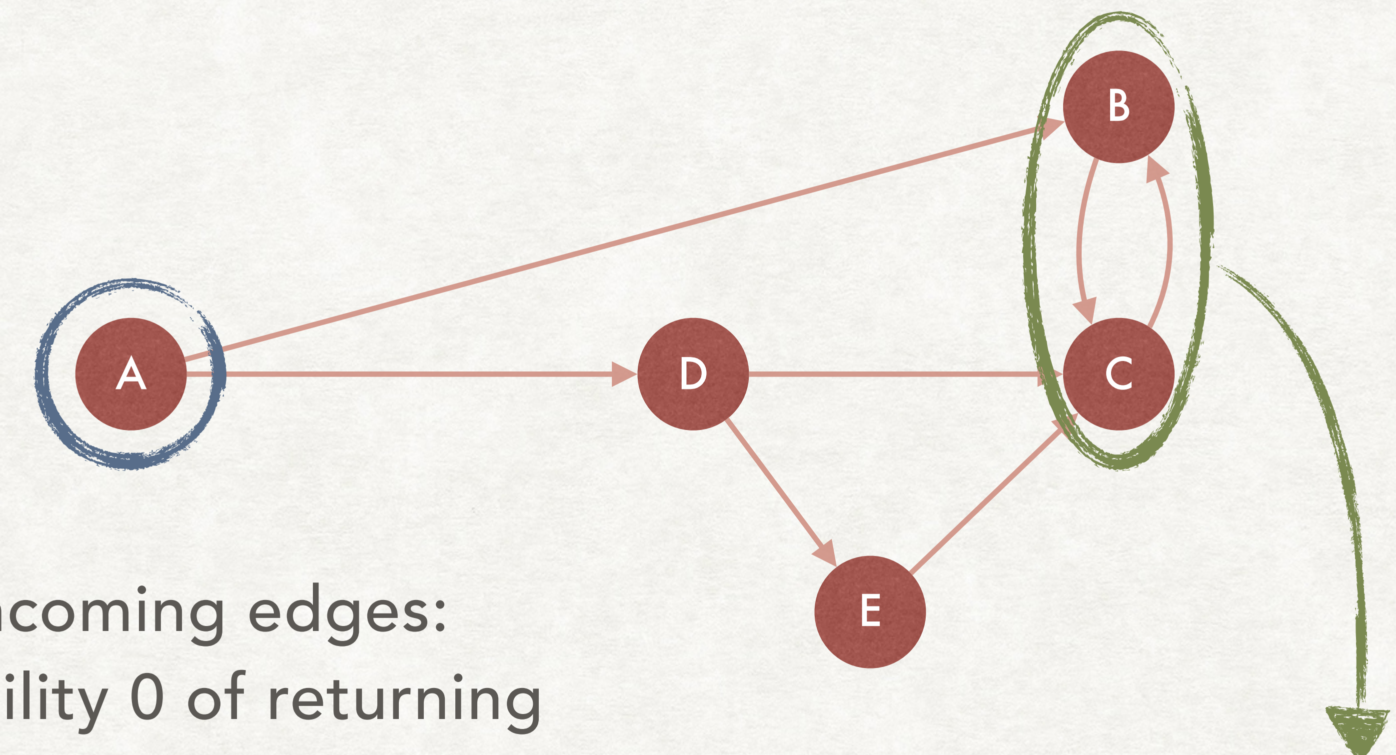


Node without outgoing edges: how to assign probabilities to go to another page?

A simple fix is to suppose that the user will go somewhere else uniformly at random: $\begin{bmatrix} \frac{1}{N} & \frac{1}{N} & \cdots & \frac{1}{N} \end{bmatrix}$

PROBLEMS

PAGES WITHOUT INCOMING OR OUTGOING LINKS



Node without incoming edges:
we have probability 0 of returning
to it once we leave it

The same problem is also present
for nodes D and E

Group of nodes without outgoing edges:
we can never leave them once entered

TELEPORTING

HOW TO MAKE THE USER SMARTER

- It is common to have “sinks” where it is impossible to exit by only following the links…
- …or pages that we cannot go back to.
- This produces an imbalance in our scores, that can potentially be exploited.
- In fact our idealised user can be a little bit smarter. At every page it can:
  - Move following one of the links in the page…
  - …or go to a random page

TELEPORTING

AND THE TRANSITION MATRIX

- Move to a linked page with probability $1 - \alpha$
- Move to random page with probability α
- $\alpha > 0$ can be considered a "damping factor" or "probability that our user decides to go to another website"

$$P = \alpha \begin{bmatrix} \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} \\ \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} \\ \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} \\ \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} \\ \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} \end{bmatrix} + (1 - \alpha) \begin{bmatrix} 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

TELEPORTING

AND THE TRANSITION MATRIX

- We assign a probability of $\frac{1}{N}$ of landing on any particular page.

- The previous matrix can also be written as:

$$P = \alpha \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} \end{bmatrix} + (1-\alpha) \begin{bmatrix} 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

"Jump vector"

TELEPORTING

AND THE TRANSITION MATRIX

$$P = \alpha \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} \end{bmatrix} + (1-\alpha) \begin{bmatrix} 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

We usually write it as $P = \alpha \vec{1}^T \vec{J} + (1-\alpha) R$

Can we find something about P that helps us in computing the PageRank of all pages (i.e., the stationary distribution)?
Can we have a solution that is independent from any initial guess that we might have to perform?

TELEPORTING

AND THE STATIONARY DISTRIBUTION

- With this “teleporting” trick we can now go to any other web page in one step.
- Which means that all entries of P are positive.
- Which means that we can apply the Perron-Frobenius theorem (actually one reformulation of it):

If P is a positive row (or column) stochastic matrix then:

1. The eigenvalue 1 is the largest eigenvalue and has multiplicity 1
2. There is a unique stochastic eigenvector for the eigenvalue 1

COMPUTING PAGERANK EXACTLY

USING THE PERRON-FROBENIUS THEOREM

The PageRank vector of the transition matrix P is the **unique** stochastic eigenvector corresponding to the eigenvalue 1

$$\vec{\pi}P = \lambda\vec{\pi}$$

In out case $\lambda = 1$, thus:

$$\vec{\pi}P = \vec{\pi}$$

Which is a linear system, we know how to solve it…

…except that P is a square matrix with a few billions of rows.

COMPUTING PAGERANK ITERATIVELY

A PRACTICAL APPROACH

- Usually we do not solve exactly the PageRank for a set of web pages.
- We use an iterative methods that, in fact, converges quite rapidly.
- The main idea is that, if we start from a stochastic vector $\vec{x}$, maybe giving equal probability to each page…
- …then $\vec{x}P^t$ for a large enough t would be a good approximation of the exact solution $\vec{\pi}$.

See also: Pavel Berkhin, A Survey on PageRank Computing, Internet Mathematics Vol. 2, No. 1: 73-120, 2005

COMPUTING PAGERANK ITERATIVELY

A PRACTICAL APPROACH

In pseudocode this could be expressed as:

Start with a random probability distribution

$\vec{x}_0 = \text{random}()$

`do`

$$\vec{x}_t = \vec{x}_{t-1}\left(\alpha\vec{1}^T\vec{J} + (1-\alpha)R\right)$$

`while` $|\vec{x}_t - \vec{x}_{t-1}|_1 \geq \varepsilon$

Update the vector by multiplying it by P

Until the difference between the vectors in two consecutive iterations is below $\varepsilon > 0$

TOPIC-SPECIFIC PAGERANK

USING PAGERANK FOR SPECIFIC TOPICS

- In addition to computing PageRank scores for all pages we can limit the computation to single topics.
- How?
- Simply change the probability distribution for the "teleportation", i.e., the "jump vector".
- Start with a (non-empty) set S of pages specific to a certain topic.
- Your jumps can only be inside S.

TOPIC-SPECIFIC PAGERANK

USING PAGERANK FOR SPECIFIC TOPICS

Given a set of pages S, we consider a topic-specific jump vector $\overrightarrow{J_S}$ in the equation:

$$P = \alpha \vec{1}^T \overrightarrow{J_S} + (1 - \alpha) R$$

With the elements of $\overrightarrow{J_S}$ now defined as:

$$\overrightarrow{J_{S_i}} = \begin{cases} \frac{1}{|S|} & \text{if } i \in S \\ 0 & \text{otherwise} \end{cases}$$

We will find a set $Y \supseteq S$ of pages with positive PageRank, thus obtaining the solution $\overrightarrow{\pi_S}$ of "topic specific PageRank for S"

PERSONALISED PAGERANK

FOR DIFFERENT USERS

- We might want to add a special PageRank score for every user, depending on the topics he/she is interested in.
- For example, based on a set of favorite web pages.
- However, performing the PageRank computation for every user is too expensive.
- We can use the *linearity of PageRank*.

PERSONALISED PAGERANK

AND LINEARITY OF PAGERANK

Let S_1 and S_2 be two disjoints of "topic specific" pages.

Suppose that the corresponding PageRank scores are $\overrightarrow{\pi_1}$ and $\overrightarrow{\pi_2}$.

For a user that is interested in the first topic with weight $w_1 \geq 0$ and in the second topic with weight $w_2 \geq 0$, with $w_1 + w_2 = 1$ we can compute the corresponding PageRank scores as

$$w_1\overrightarrow{\pi_1} + w_2\overrightarrow{\pi_2}$$

Hence we can compute personalised PageRank scores with a weighted sum of pre-computed scores.

MANIPULATION OF PAGERANK

AND REL=NOFOLLOW

- There is an implicating conflict between the indexing (especially the one performed by Google) and the people managing the websites.

- Google needs to keep the search results relevant to the user.

- Normal and spam websites wants to rank high in the search results.

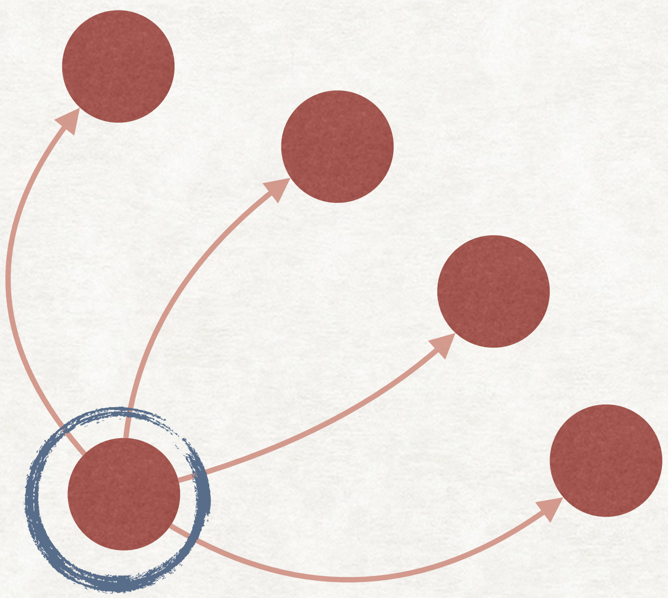
- To mitigate some of the problems, the “rel=nofollow” attribute was added to HTML.

- A link like:
  `<a href="http://www.example.com/" rel="nofollow">Link</a>`
  would not be considered for the purpose of computing the PageRank score.

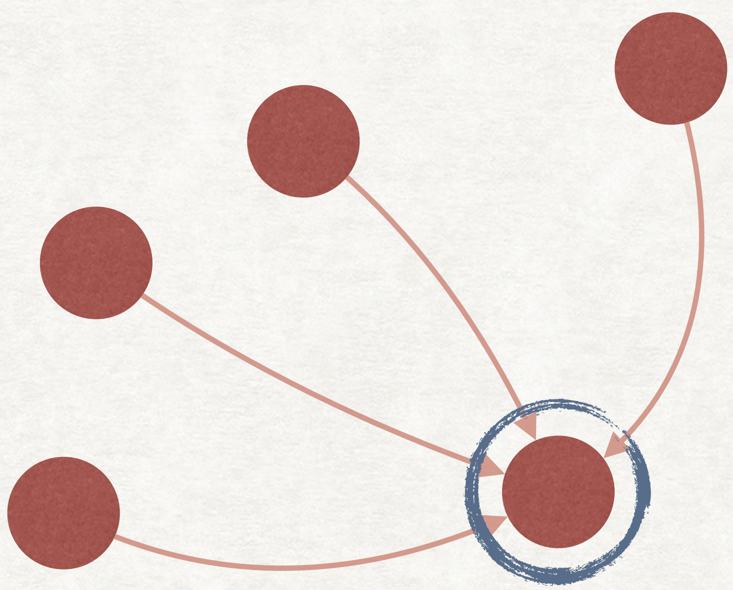
HITS

HUBS AND AUTHORITIES

TWO TYPES OF SCORES



This page links a lot of other pages. It can be considered an **hub**.



This page is linked by a lot of other pages. It can be considered an **authority**.

HUBS AND AUTHORITIES

TWO TYPES OF SCORES

- Hubs are pages that are important not for their content, but for the links that they provide toward pages with interesting content.
- Authorities are pages that are important for their content; therefore, they are linked by many pages.
- The **hyperlink-induced topic search** (HITS) algorithm assigns two different scores to each page, an authority and a hub score.
- The main idea behind the algorithm is:
  - A good hub points to pages with high authority score.
  - A good authority is pointed by pages with high hub scores.

HUBS AND AUTHORITIES

TWO TYPES OF SCORES

- Differently from PageRank, HITS is usually computed when the query is executed:
- A set of pages is obtained by some other methods (e.g., by looking at the text content of the page).
- We consider the subset of pages that we have retrieved (which will probably have very few links to each other) as a **root set**.
- We add to the root set all pages pointed and pointing to it.
- In this extended set we compute the two scores, that can now be used for ranking.

HOW TO COMPUTE SCORES

HUBS AND AUTHORITIES

The hub score $h(x)$ of a page x is defined as:

$$h(x) = \sum_{x \to y} a(y)$$

The sum of the authority scores for all pages *linked* by x.

The authority score $a(x)$ of a page x is defined as:

$$a(x) = \sum_{y \to x} h(y)$$

The sum of the hub scores for all pages *that links to* x.

HOW TO COMPUTE SCORES

HUBS AND AUTHORITIES

As with PageRank, we can compute the scores analytically.
But here we illustrate an iterative method

Start with all hub and authority scores set to 1.
At each time step $t > 0$ update them as:

$$\bar{h}_t(x) = \sum_{x \to y} a_{t-1}(y) \qquad \bar{a}_t(x) = \sum_{y \to x} h_{t-1}(y)$$

But if we only perform this update we might not converge!
We need to normalise the scores:

$$h_t(x) = \frac{\bar{h}_t(x)}{\sqrt{\sum_y (\bar{h}_{t-1}(y))^2}} \qquad a_t(x) = \frac{\bar{a}_t(x)}{\sqrt{\sum_y (\bar{a}_{t-1}(y))^2}}$$