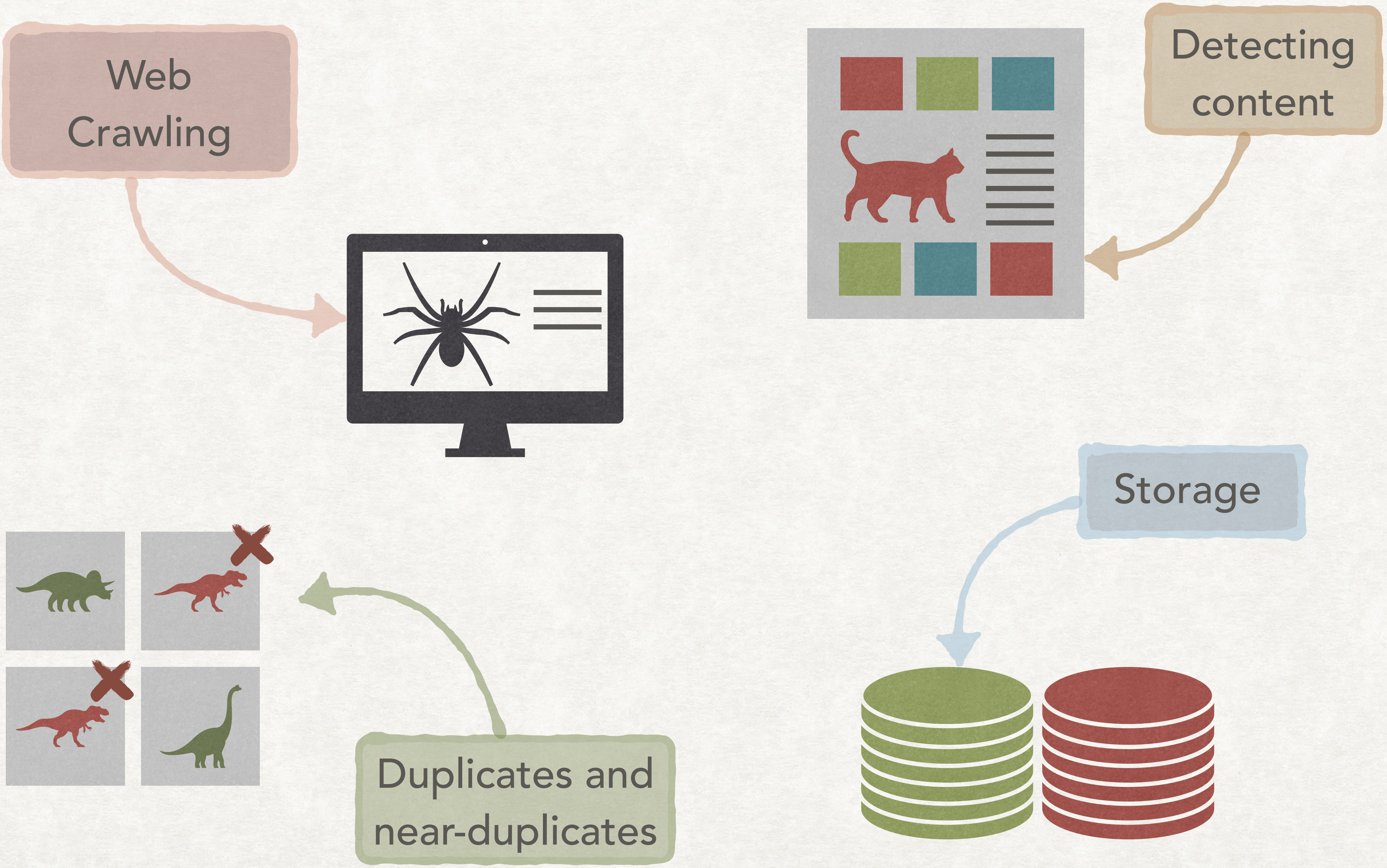
INFORMATION RETRIEVAL

Luca Manzoni
lmanzoni@units.it

Lecture 9

LECTURE OUTLINE

*NOW AVAILABLE VIA THE INFORMATION SUPERHIGHWAY



Web Crawling

Detecting content

Duplicates and near-duplicates

Storage

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)

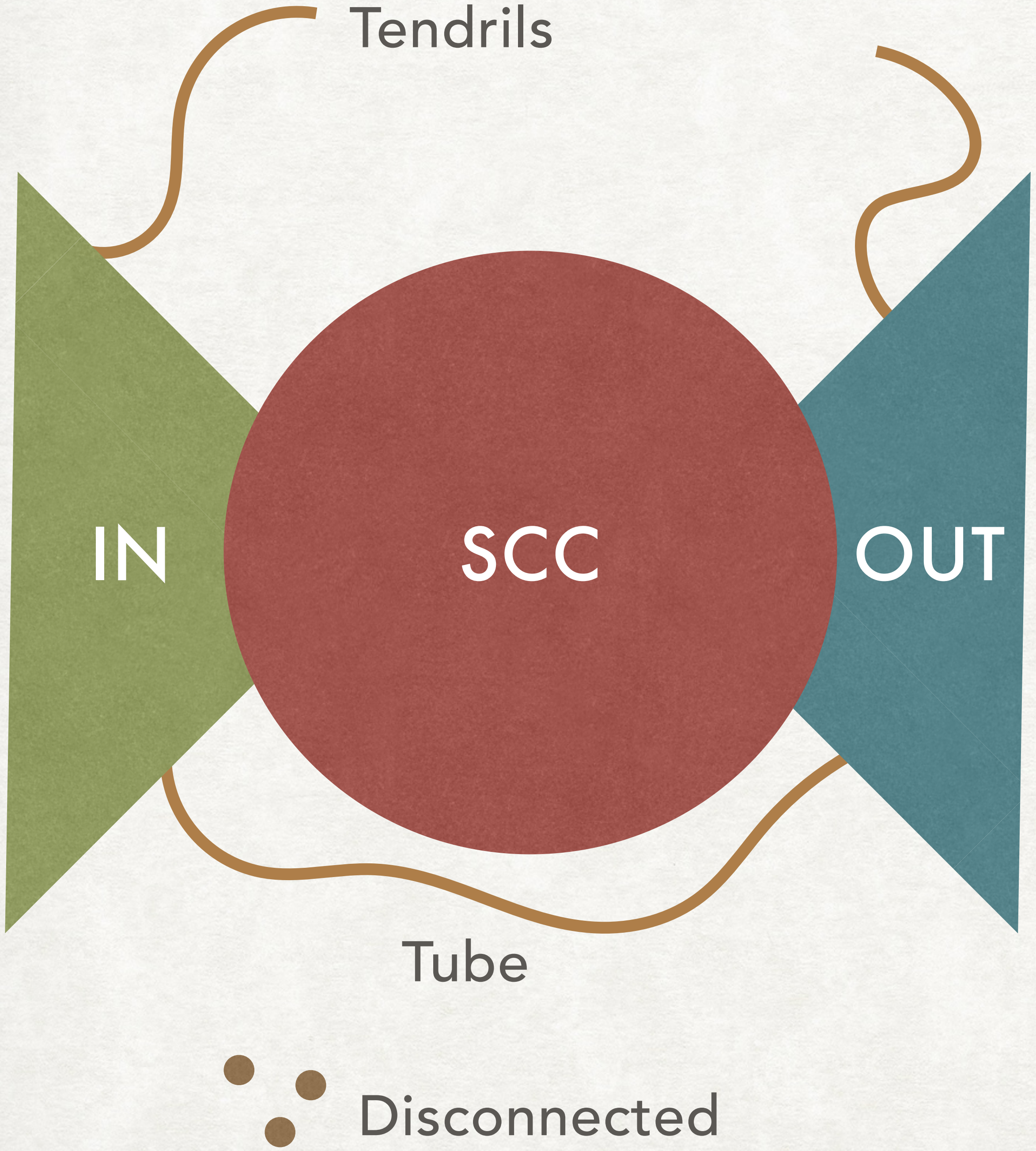
THE WEB AS A GRAPH

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.

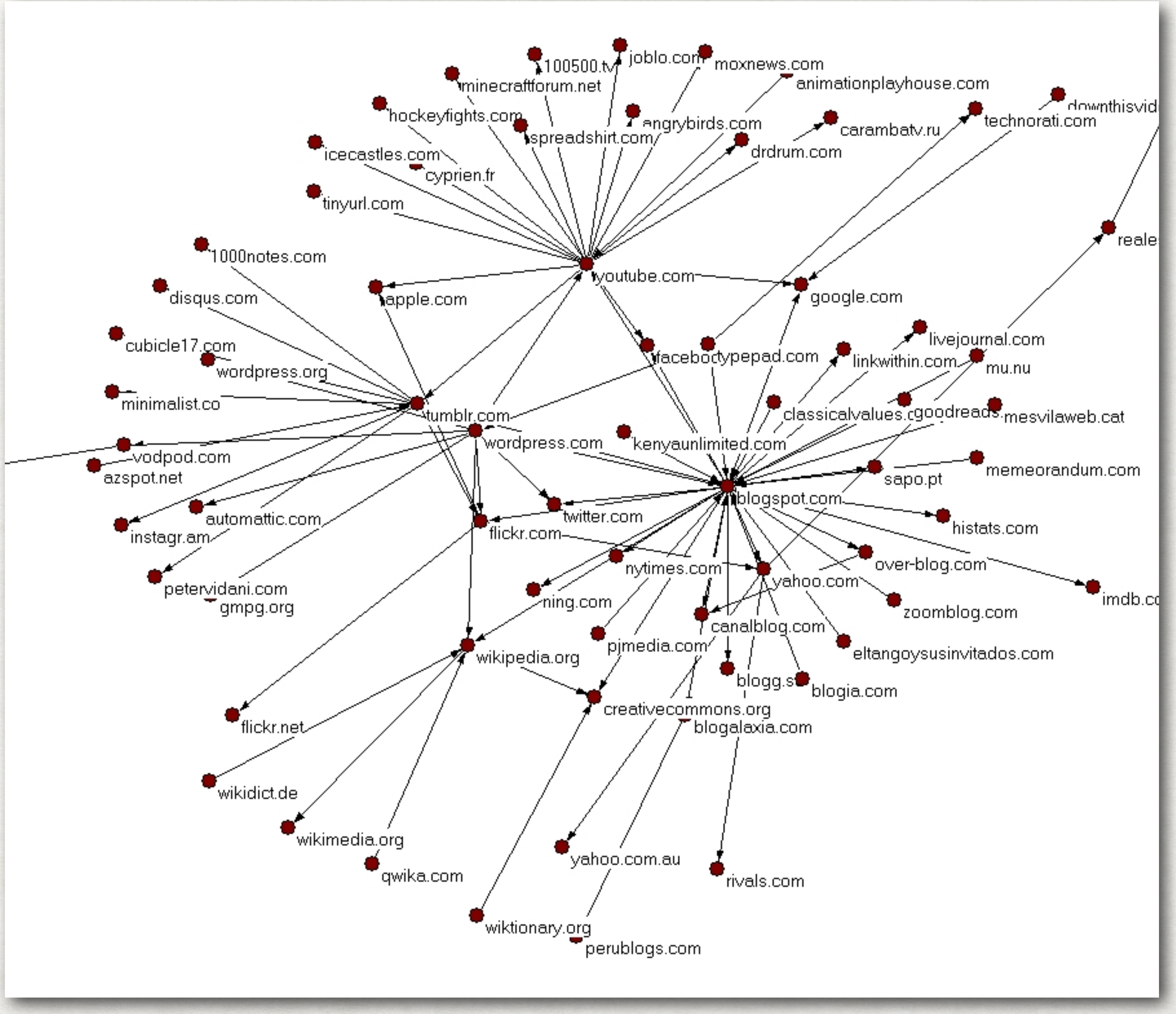
SOME REAL-WORLD DATA

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

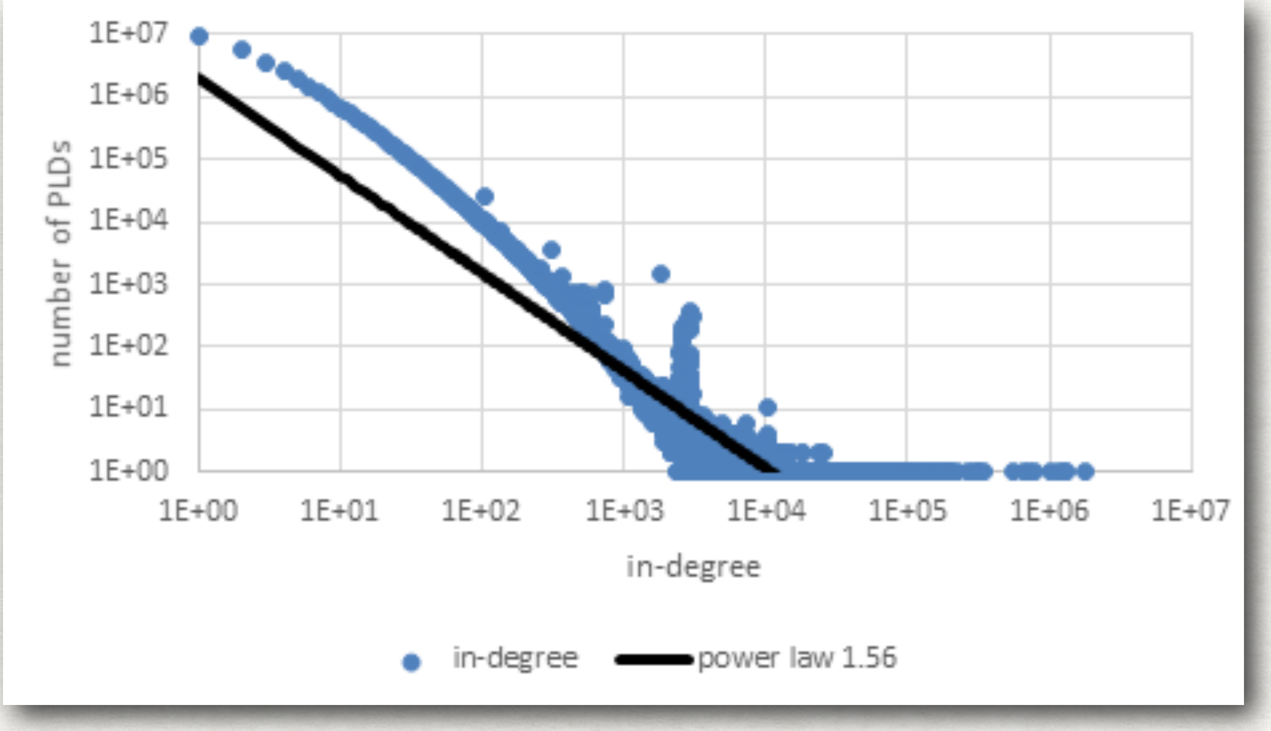
A GRAPH OF PLD



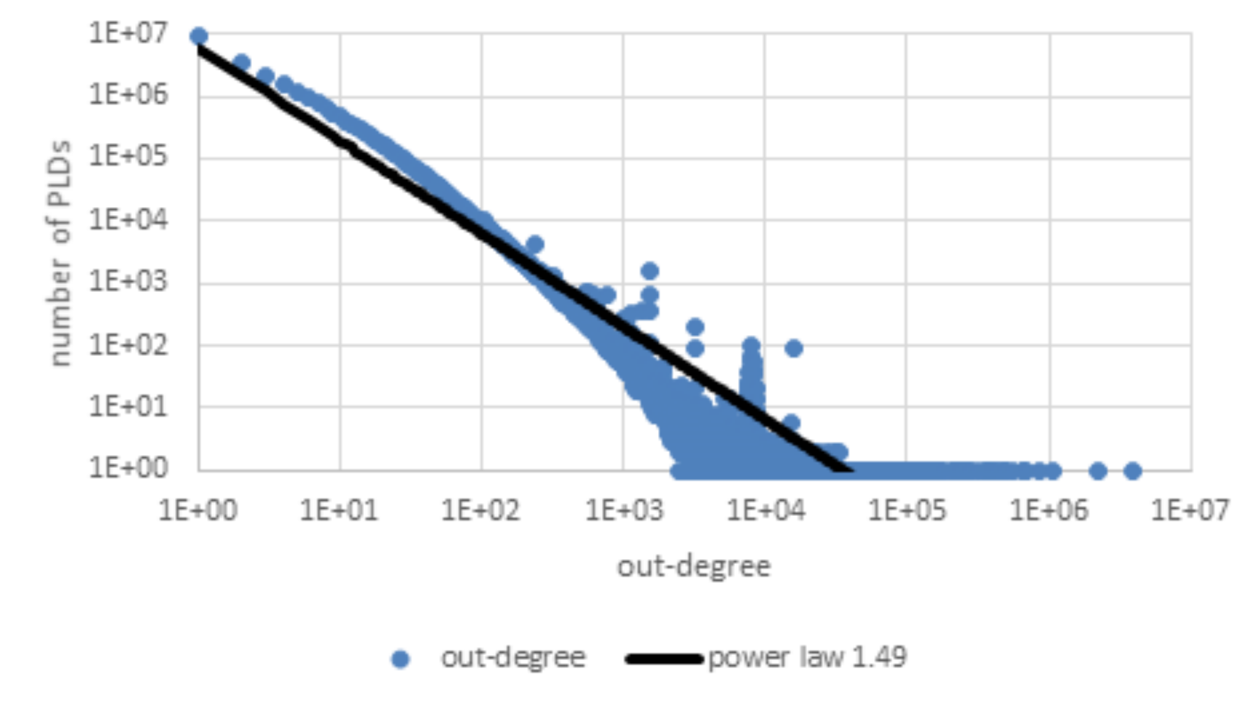
DISTRIBUTION OF DEGREES

AT THE PLD LEVEL

Both axes have a logarithmic scale



In-degree distribution
(PLD level)

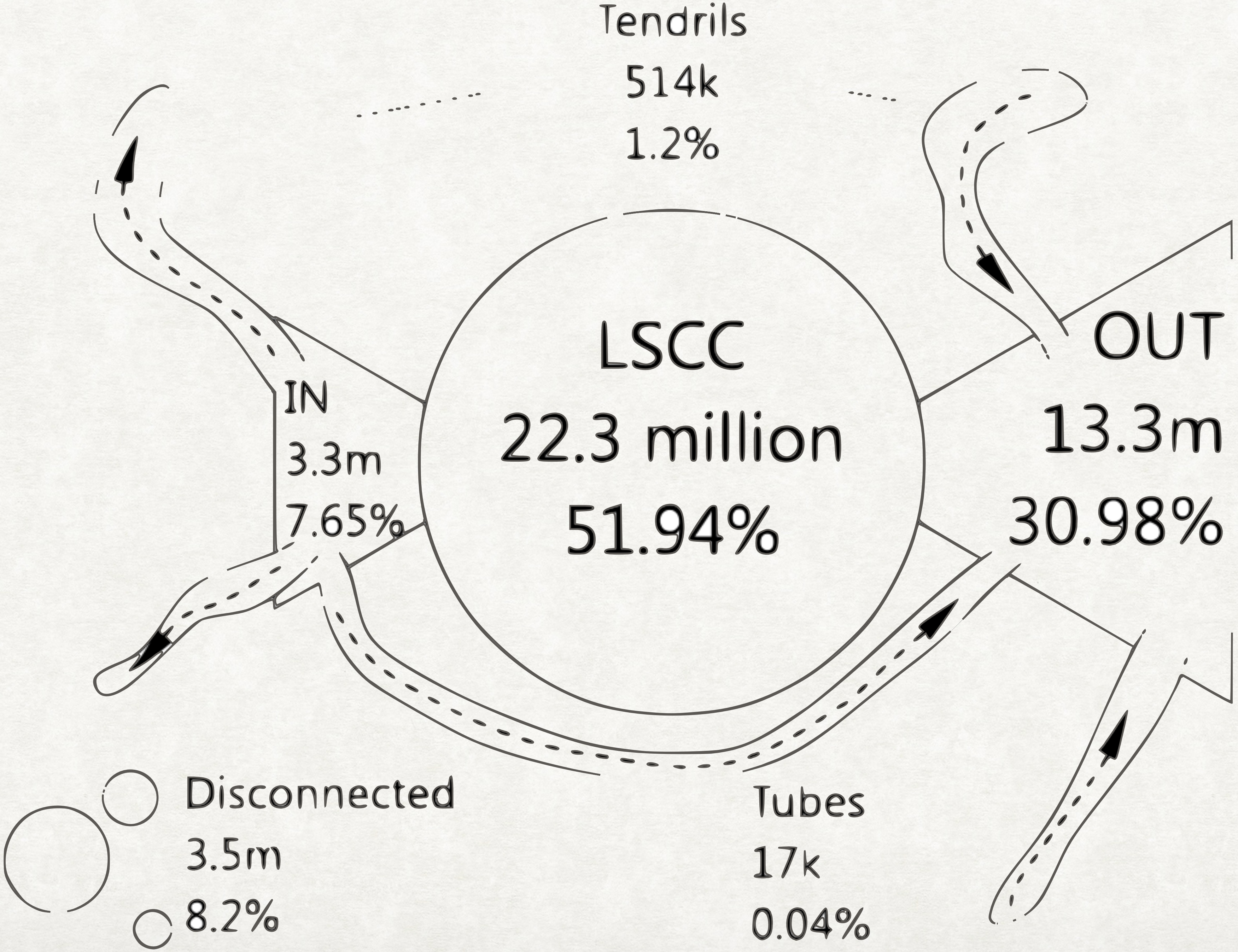


Out-degree distribution
(PLD level)

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



Tendrils
514k
1.2%

LSCC
22.3 million
51.94%

IN
3.3m
7.65%

OUT
13.3m
30.98%

Disconnected
3.5m
8.2%

Tubes
17k
0.04%

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

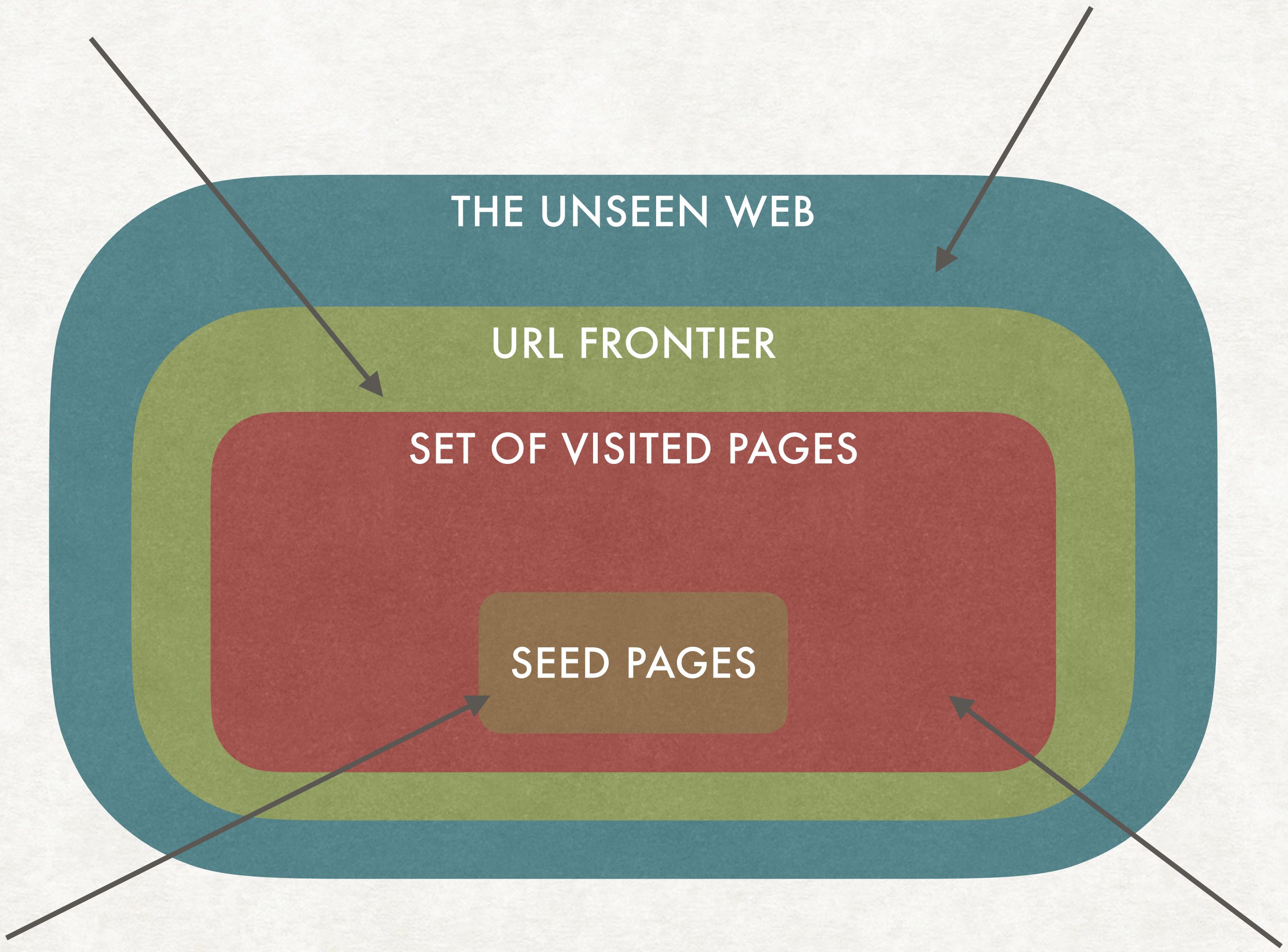
- Web crawling is the process of gathering pages from the Web to index them.
- The process is carried on by *web crawlers*, also called *spiders*.
- While retrieving a single web page is simple…
- …web crawling must take into account the scale of the web…
- …and the fact that the content to index is 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



THE UNSEEN WEB

URL FRONTIER

SET OF VISITED PAGES

SEED PAGES

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

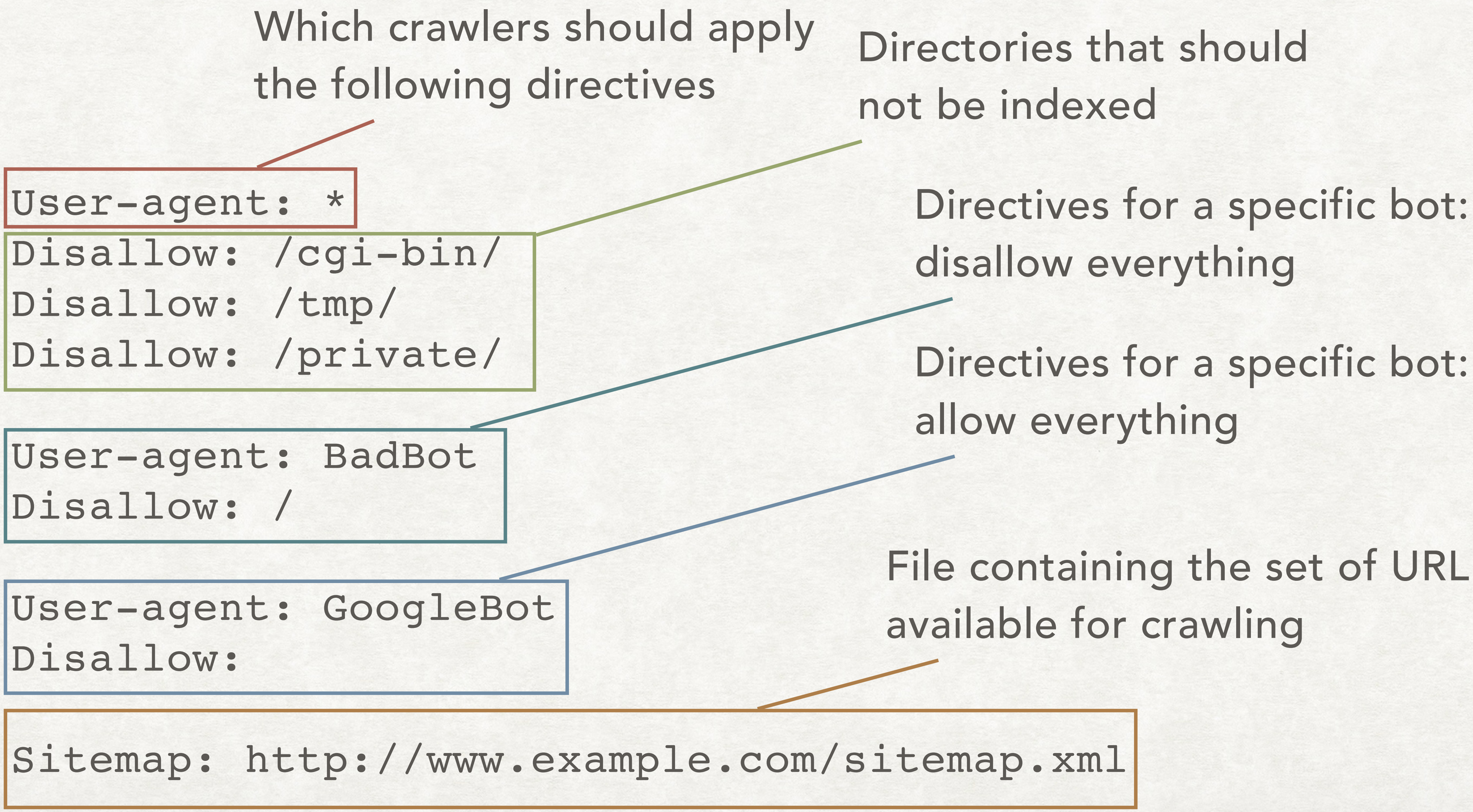
EVERY WEB CRAWLER MUST HAVE THEM

- **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



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

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

User-agent: BadBot
Disallow: /

User-agent: GoogleBot
Disallow:

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

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).

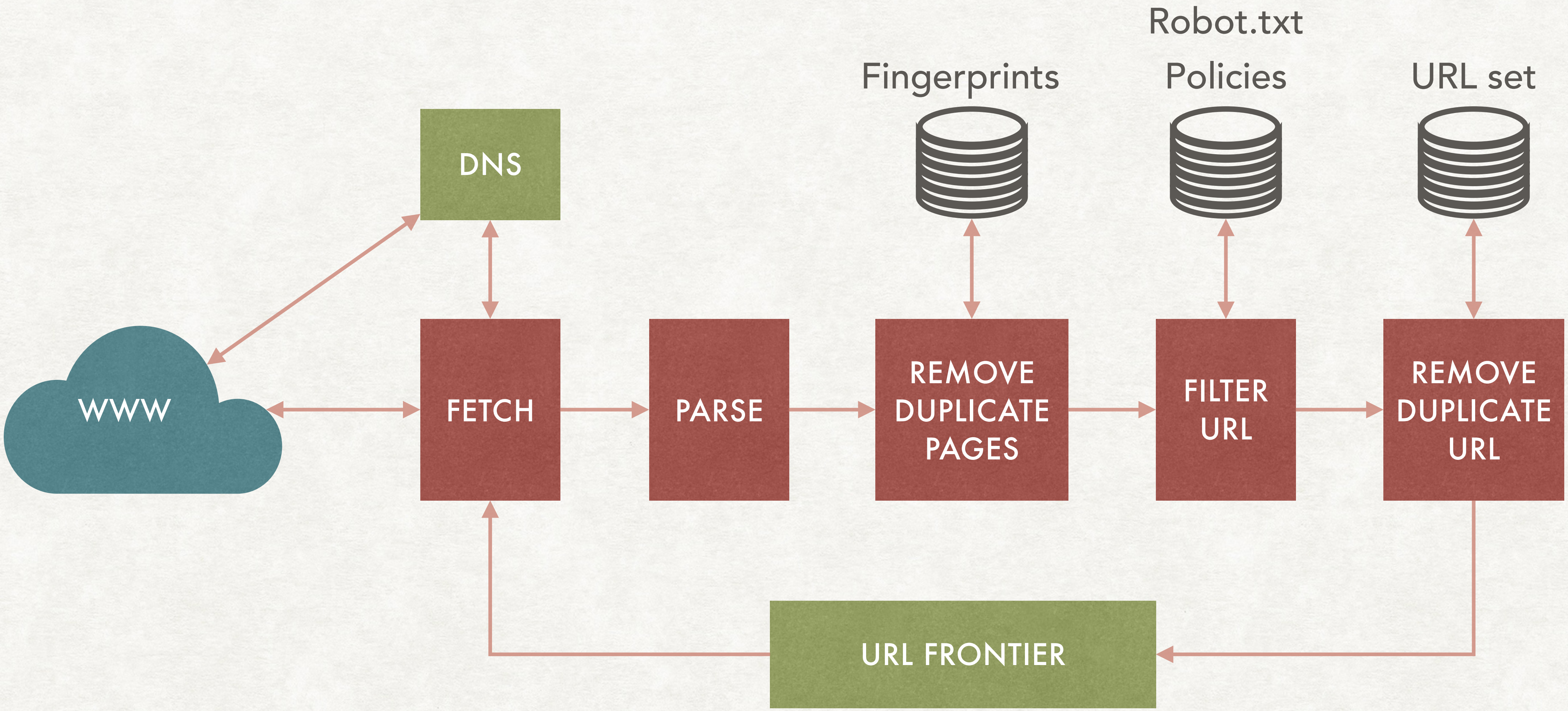
GOOD PROPERTIES OF A WEB CRAWLER

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.

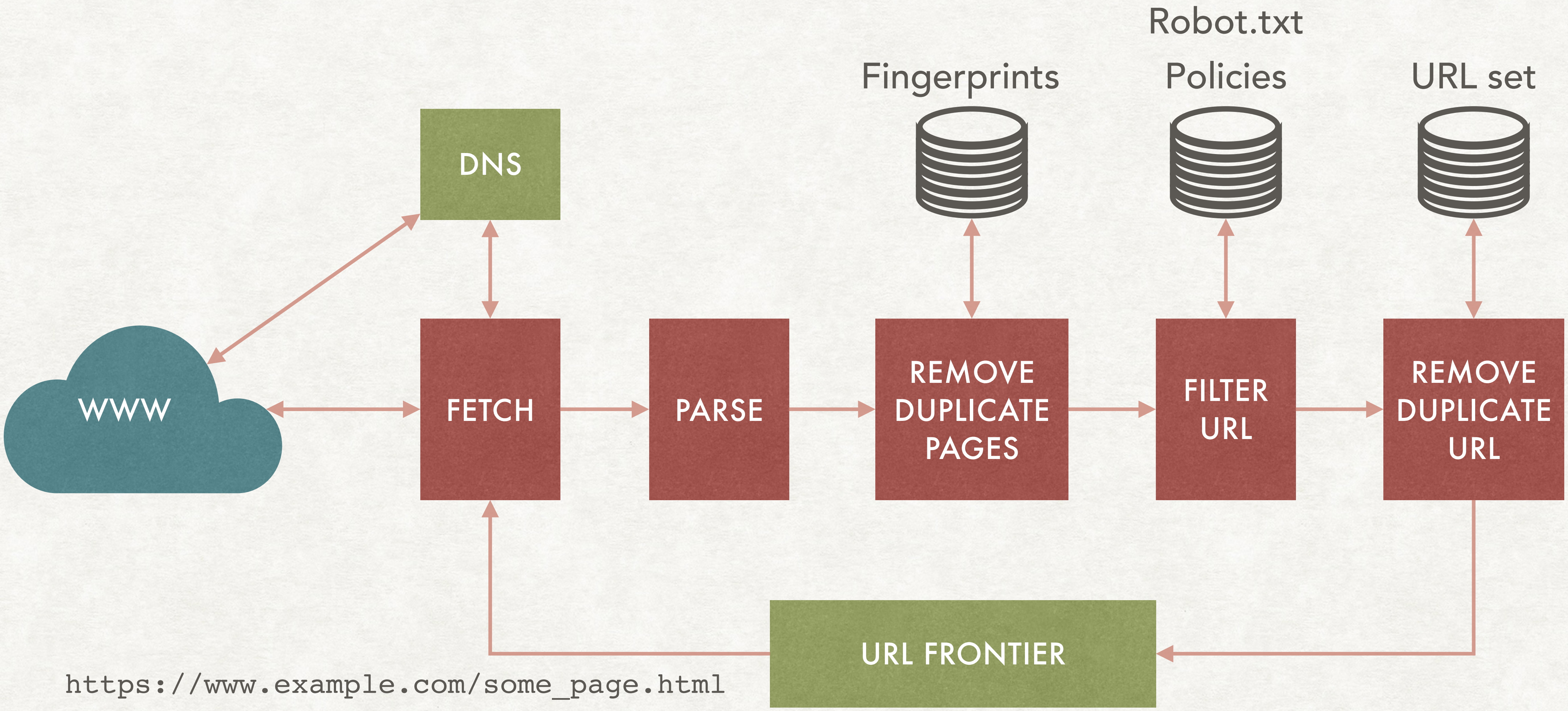
ARCHITECTURE OF A WEB CRAWLER

BASIC STRUCTURE



ARCHITECTURE OF A WEB CRAWLER

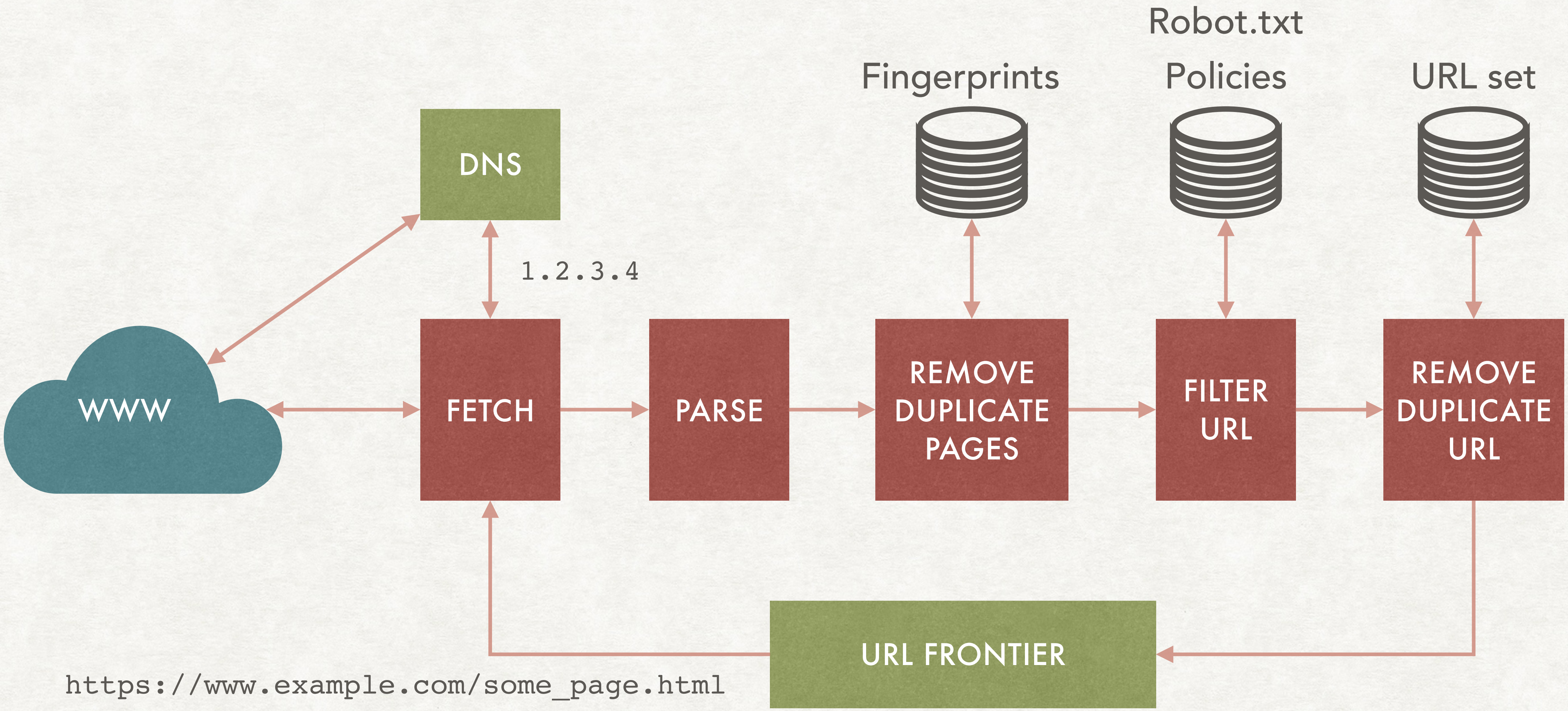
BASIC STRUCTURE



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

ARCHITECTURE OF A WEB CRAWLER

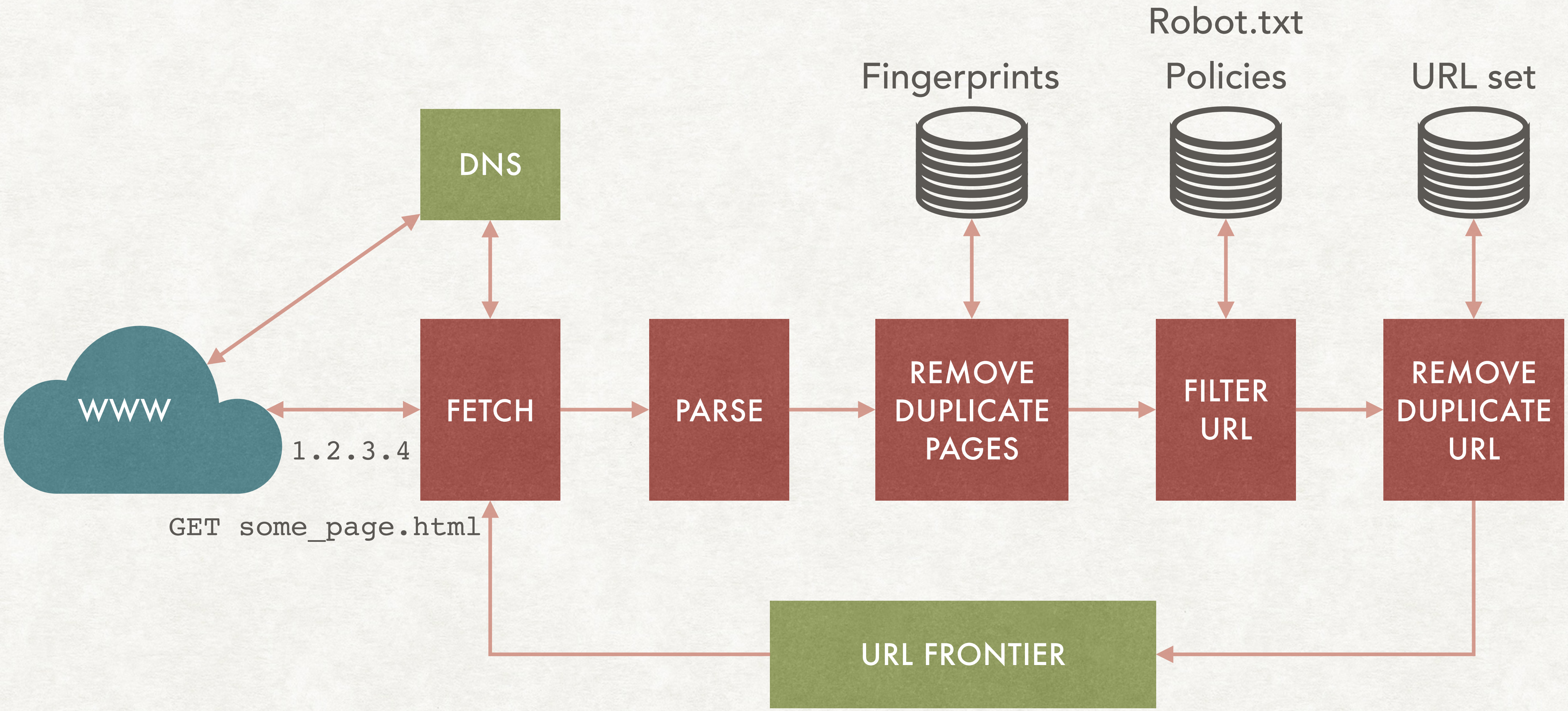
BASIC STRUCTURE



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

ARCHITECTURE OF A WEB CRAWLER

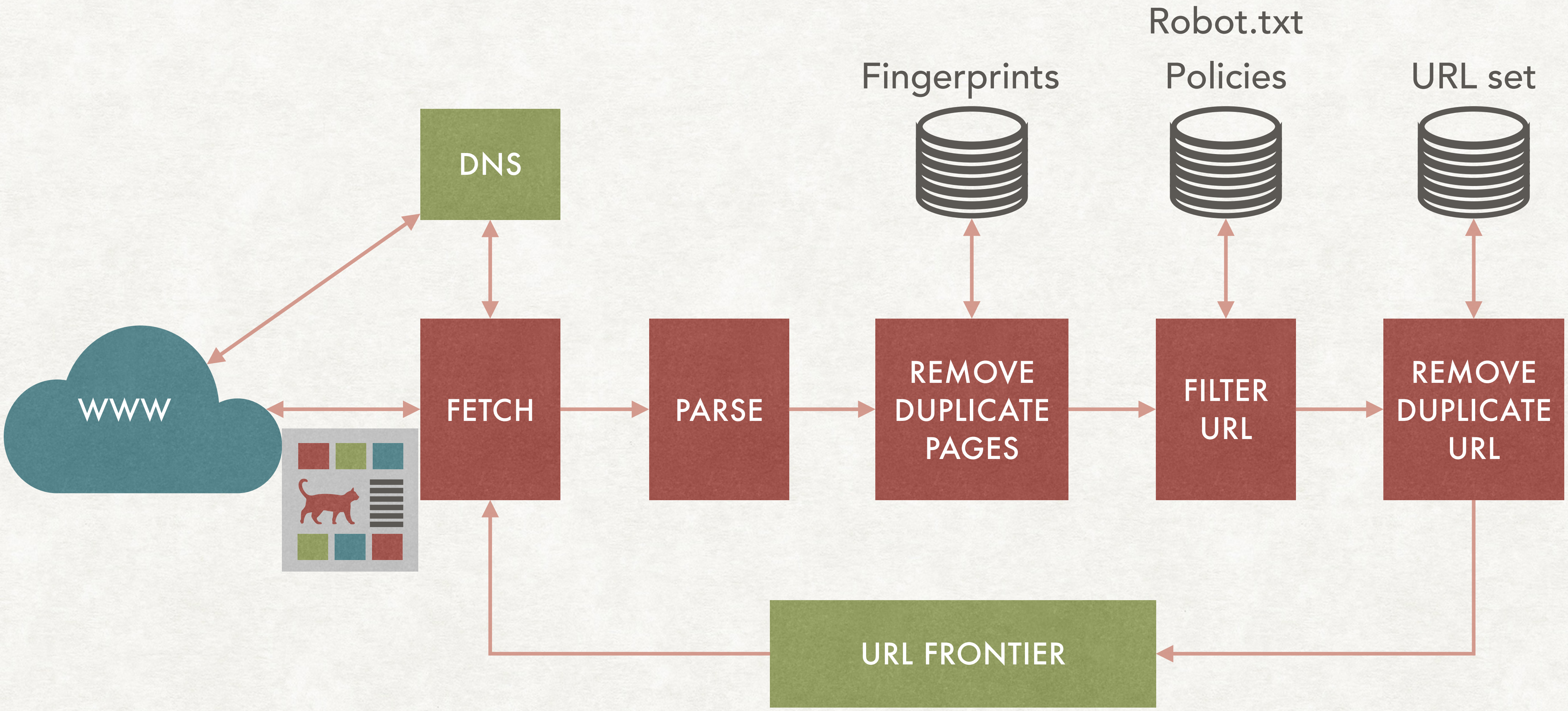
BASIC STRUCTURE



The fetch module asks for the web page to the server

ARCHITECTURE OF A WEB CRAWLER

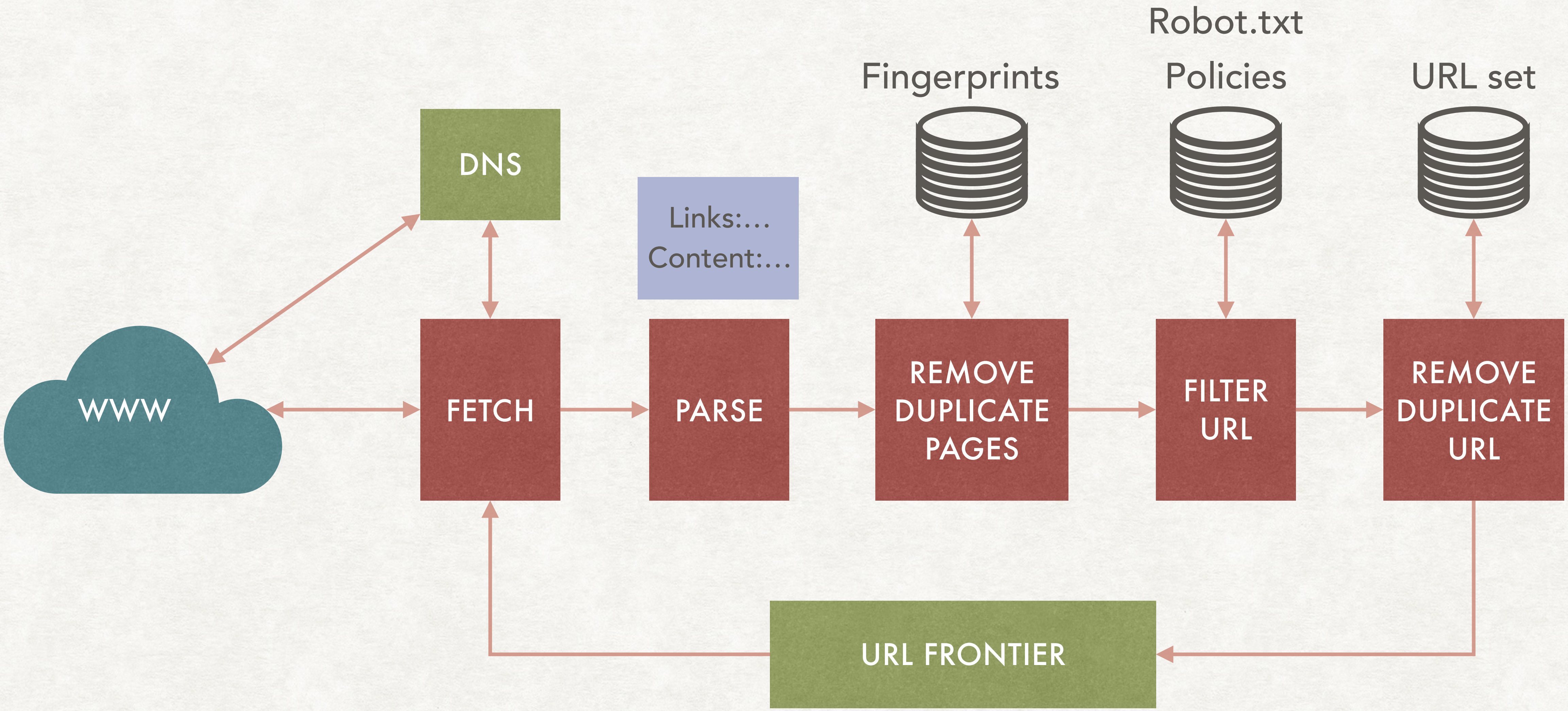
BASIC STRUCTURE



The fetch module receives the web page

ARCHITECTURE OF A WEB CRAWLER

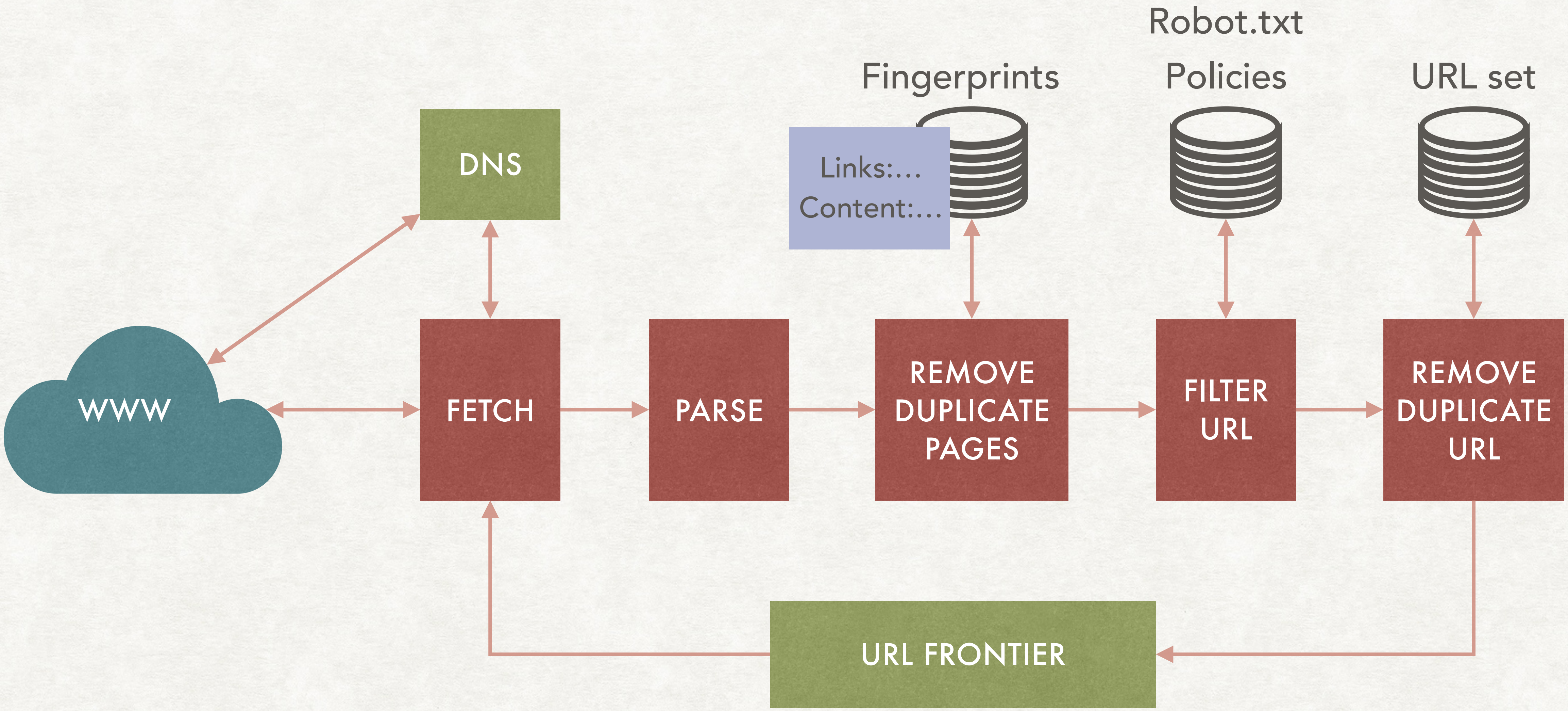
BASIC STRUCTURE



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

ARCHITECTURE OF A WEB CRAWLER

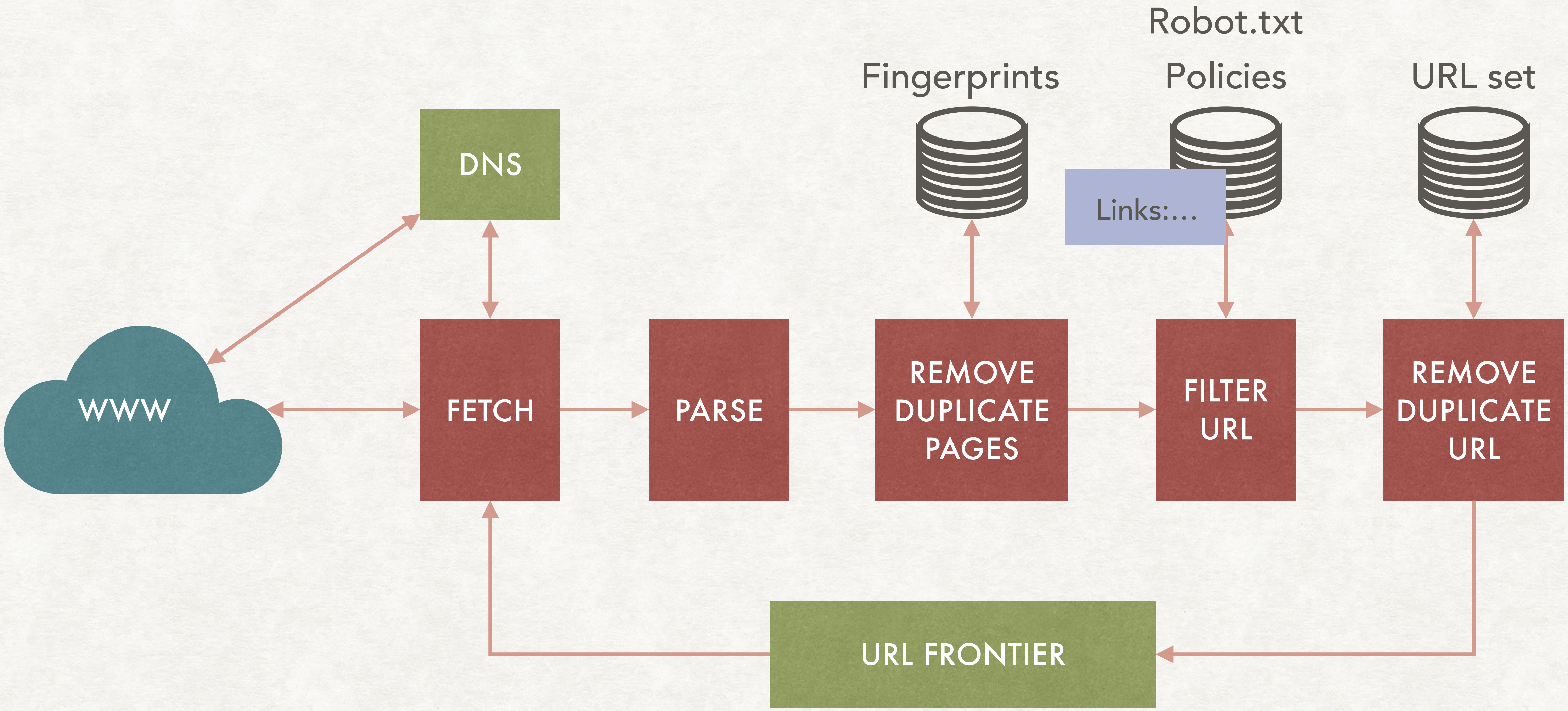
BASIC STRUCTURE



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

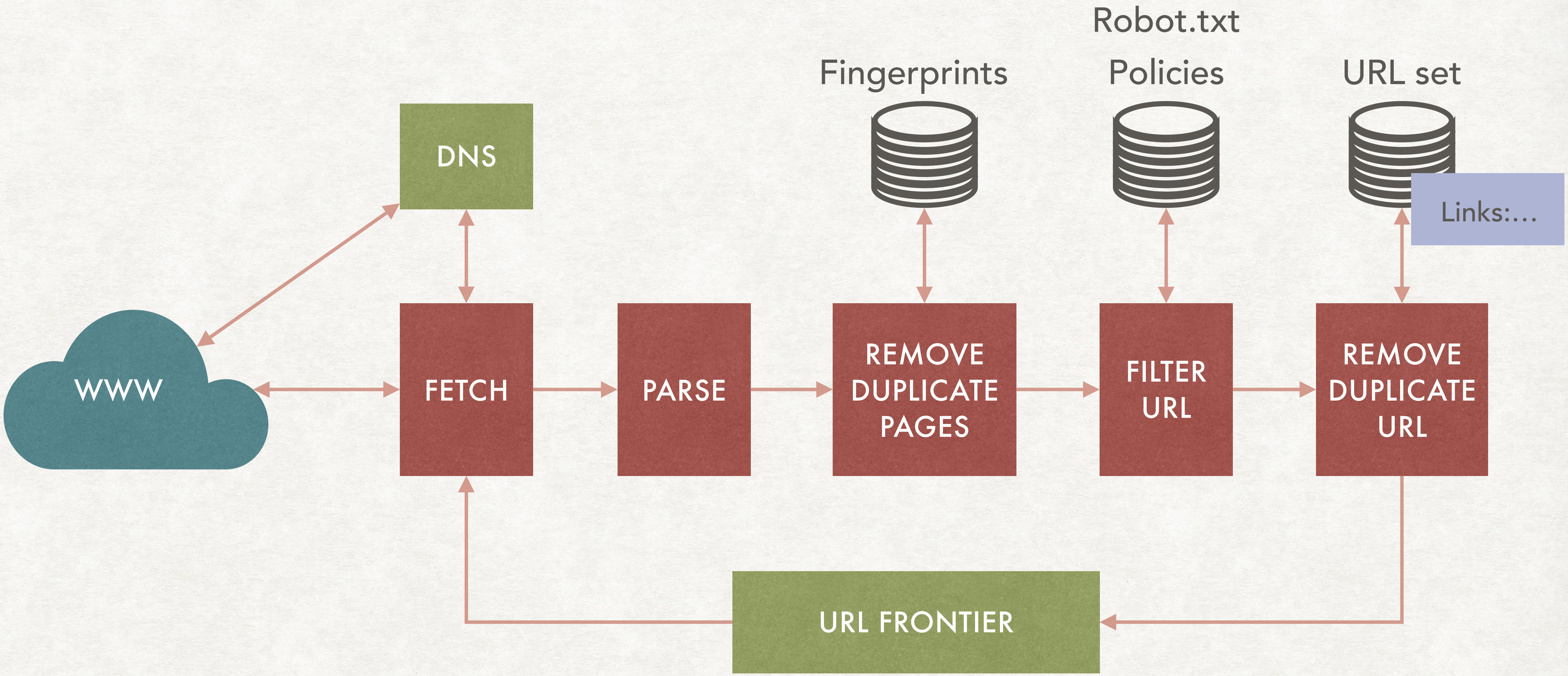
BASIC STRUCTURE



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

ARCHITECTURE OF A WEB CRAWLER

BASIC STRUCTURE



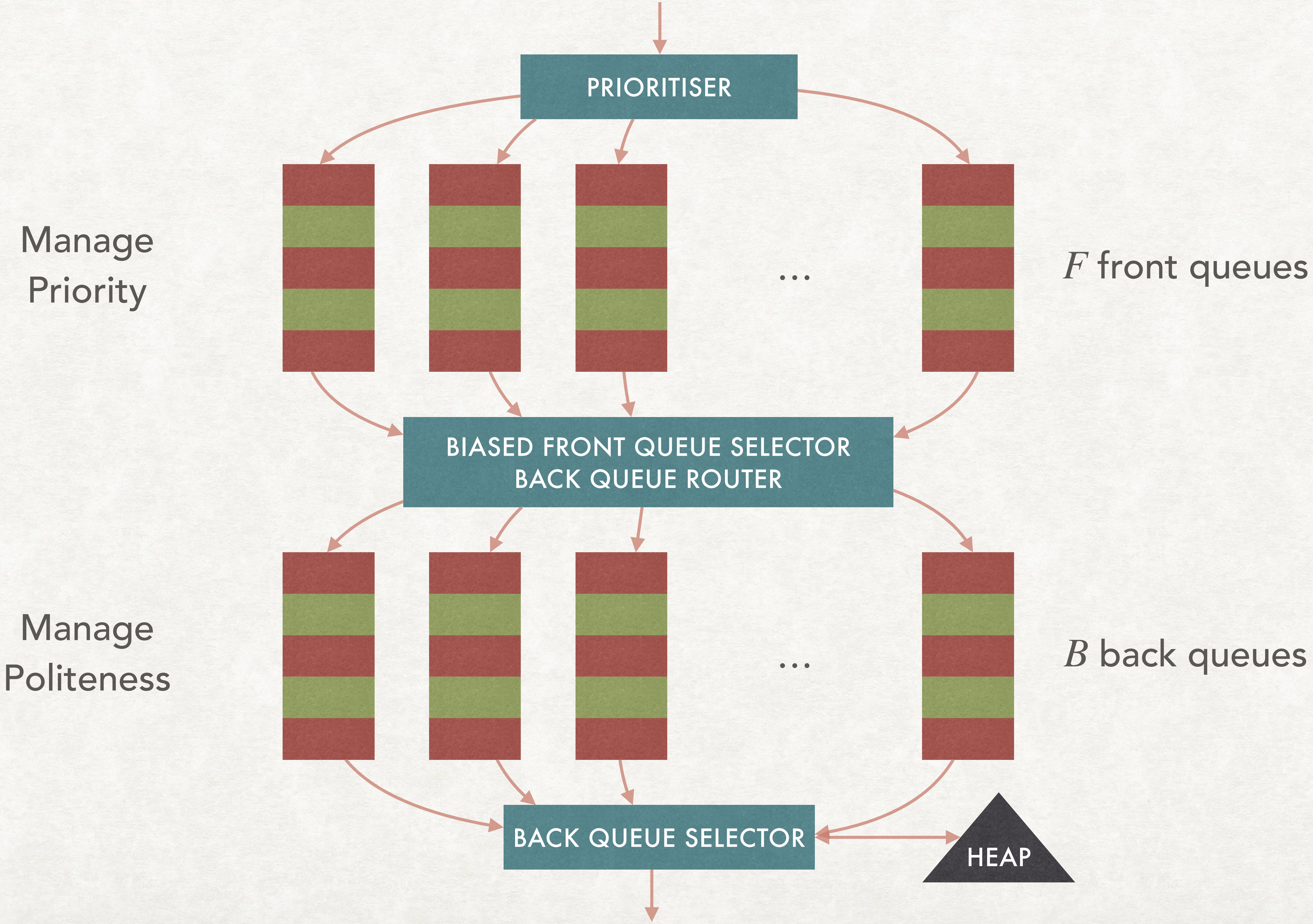
Finally, before being inserted into the URL frontier (the set of URL to visit), already visited URL are removed.

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



Manage Priority

Manage Politeness

PRIORITISER

F front queues

…

BIASED FRONT QUEUE SELECTOR
BACK QUEUE ROUTER

B back queues

…

BACK QUEUE SELECTOR

HEAP

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.

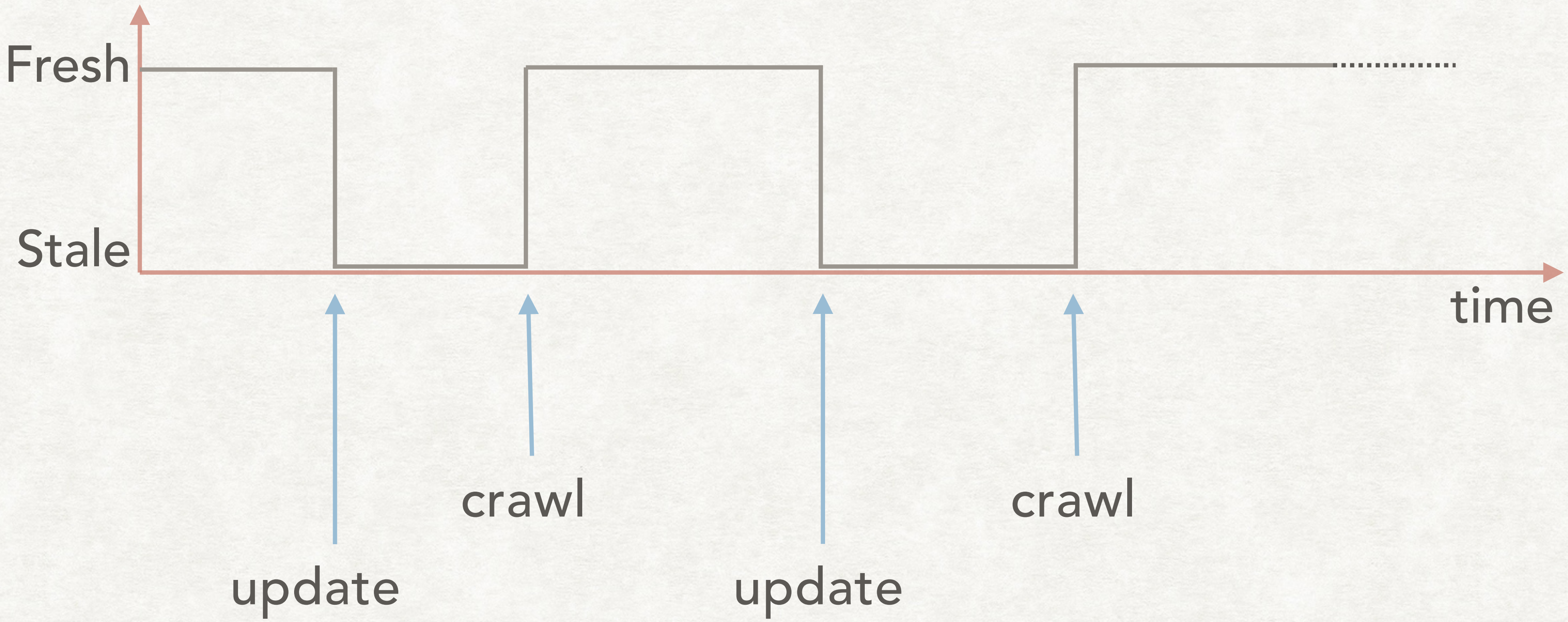
FRESHNESS

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.

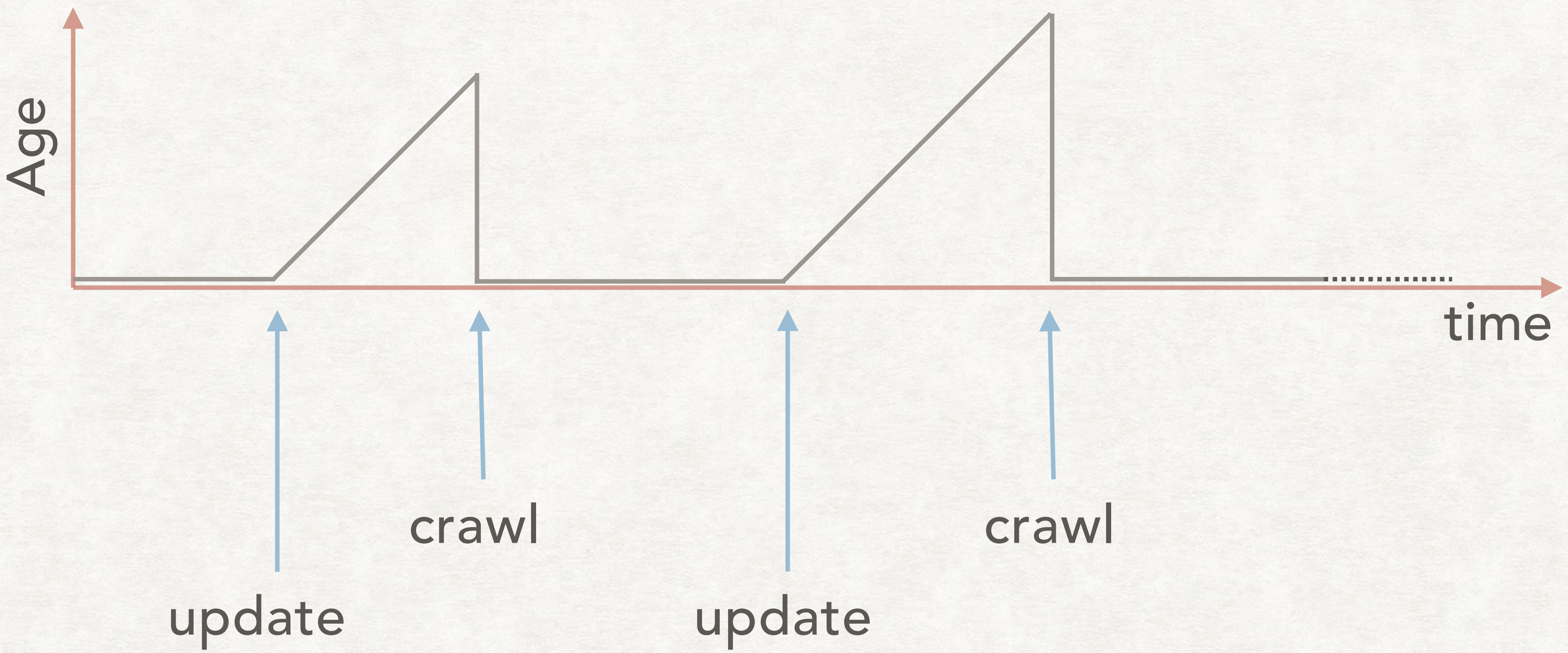
FRESHNESS

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.

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

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.

STORAGE
(AND GOOGLE BIGTABLE)

BIGTABLE

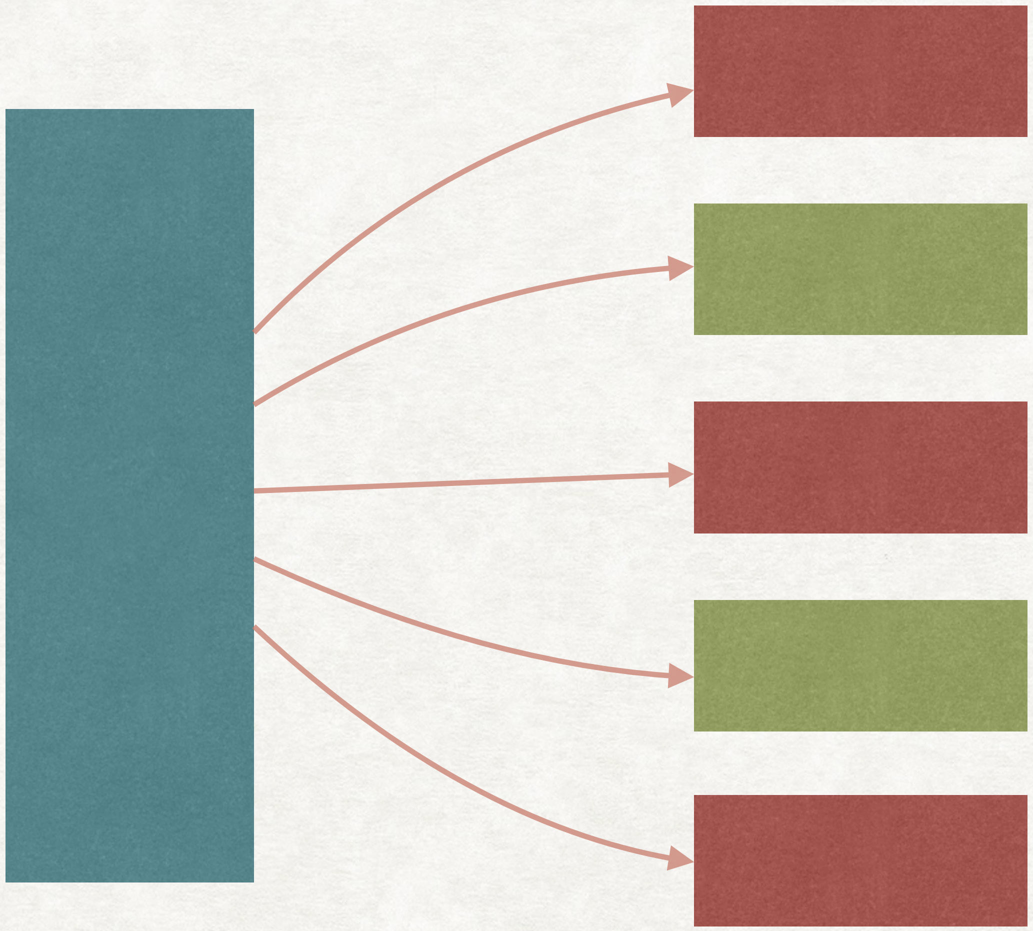
AND WIDE COLUMN STORES

- BigTable, by Google is a NoSQL database used internally at Google since 2005. From 2015 it is available as a service.
- Two similar open source projects are:
  - Apache HBase (https://hbase.apache.org)
  - Apache Accumulo (https://accumulo.apache.org)
- Maps three values into a byte array:
  a row and column keys as arbitrary strings plus a timestamp.
- Distributed database initially developed to store web pages.

Original BigTable paper: https://www.usenix.org/legacy/event/osdi06/tech/chang/chang.pdf

ONE TABLE

AND MANY TABLETS



One *logical* table…

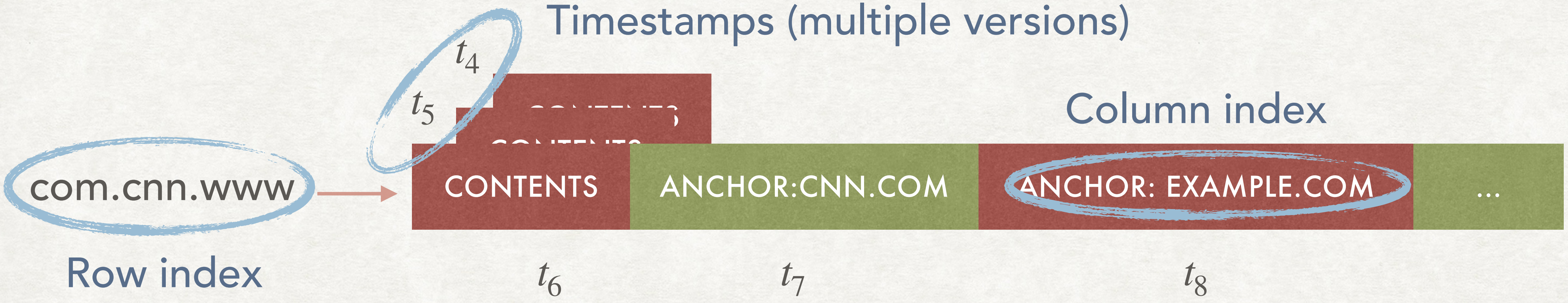
…is partitioned into multiple **tablets**…

…which are distributed across thousands of machines

- Each database contains exactly one (logical) table.
- The data are saved in an immutable way. This also helps in the case of failure recovery
- Any change to a tablet is recorded in a transaction log.
- Recent data are stored in RAM and periodically merged to reduce the total number of disk files.

ROWS IN BIGTABLE

- The data are organised by rows. Rows are partitioned into tablets based on their row key.
- Each row can have a different (and large) number of columns.
- Columns are grouped in column families. In fact, column keys have the syntax *family:quantifier*.
- The column family is the basic unit of access control and usually all data stored in the same family have the same type.



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.

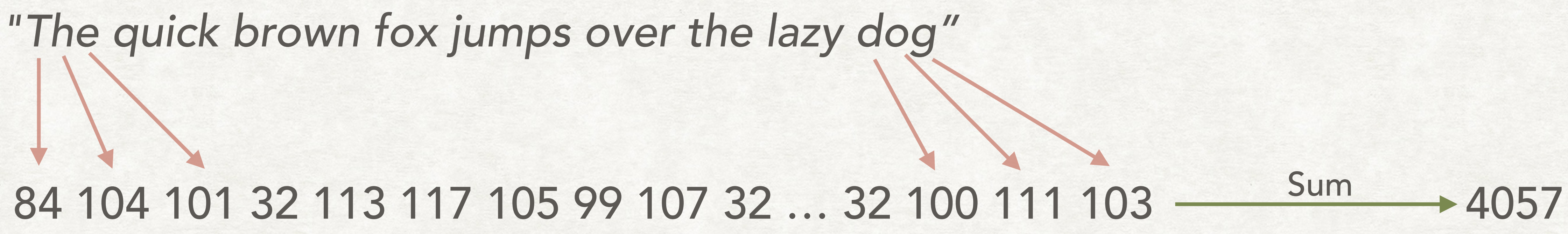
[1] Fetterly, Dennis, Mark Manasse, and Marc Najork. "On the evolution of clusters of near-duplicate web pages.”

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.

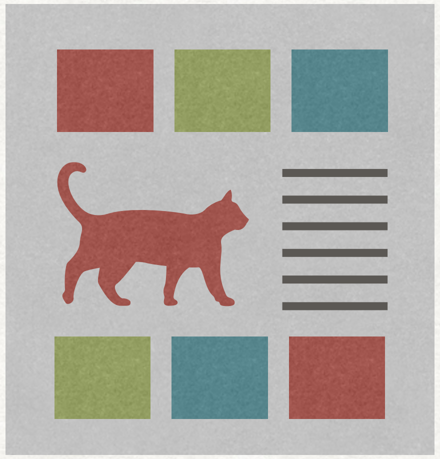
NEAR-DUPLICATES

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

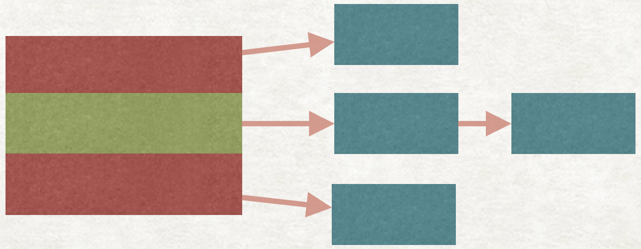
fox jumps over

The n-grams are hashed

1490

1400

The hashes are stored
in an inverted index



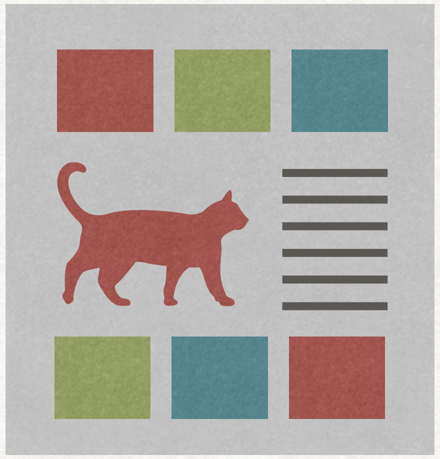
FINGERPRINTS

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

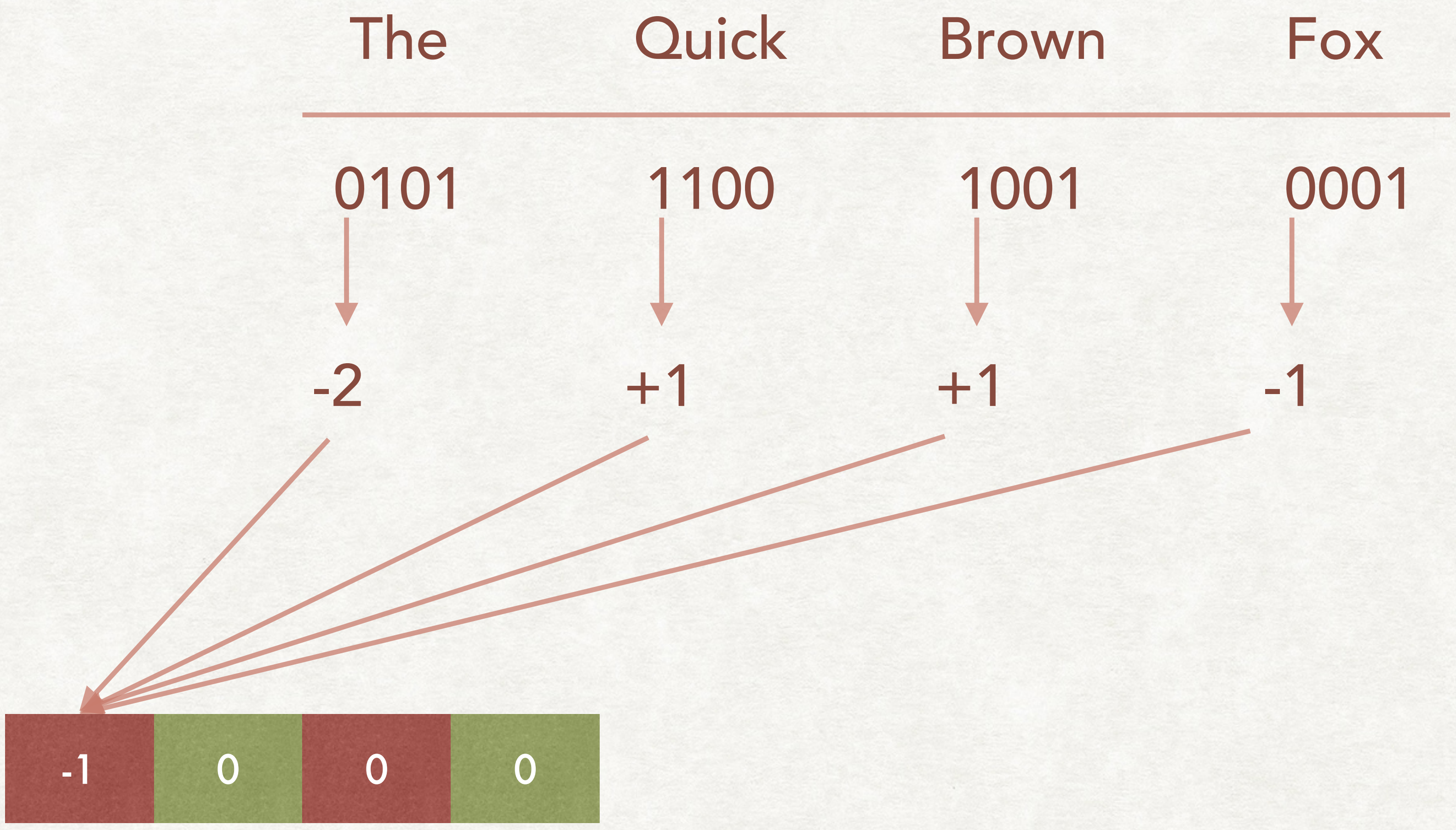
Start with a vector of size b with all positions initially set to 0

0	0	0	0

SIMHASH

A MORE RECENT FINGERPRINTING TECHNIQUE

The	2
Quick	1
Brown	1
Fox	1



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

-1	0	0	0

Look at the first bit of the hash of every word.

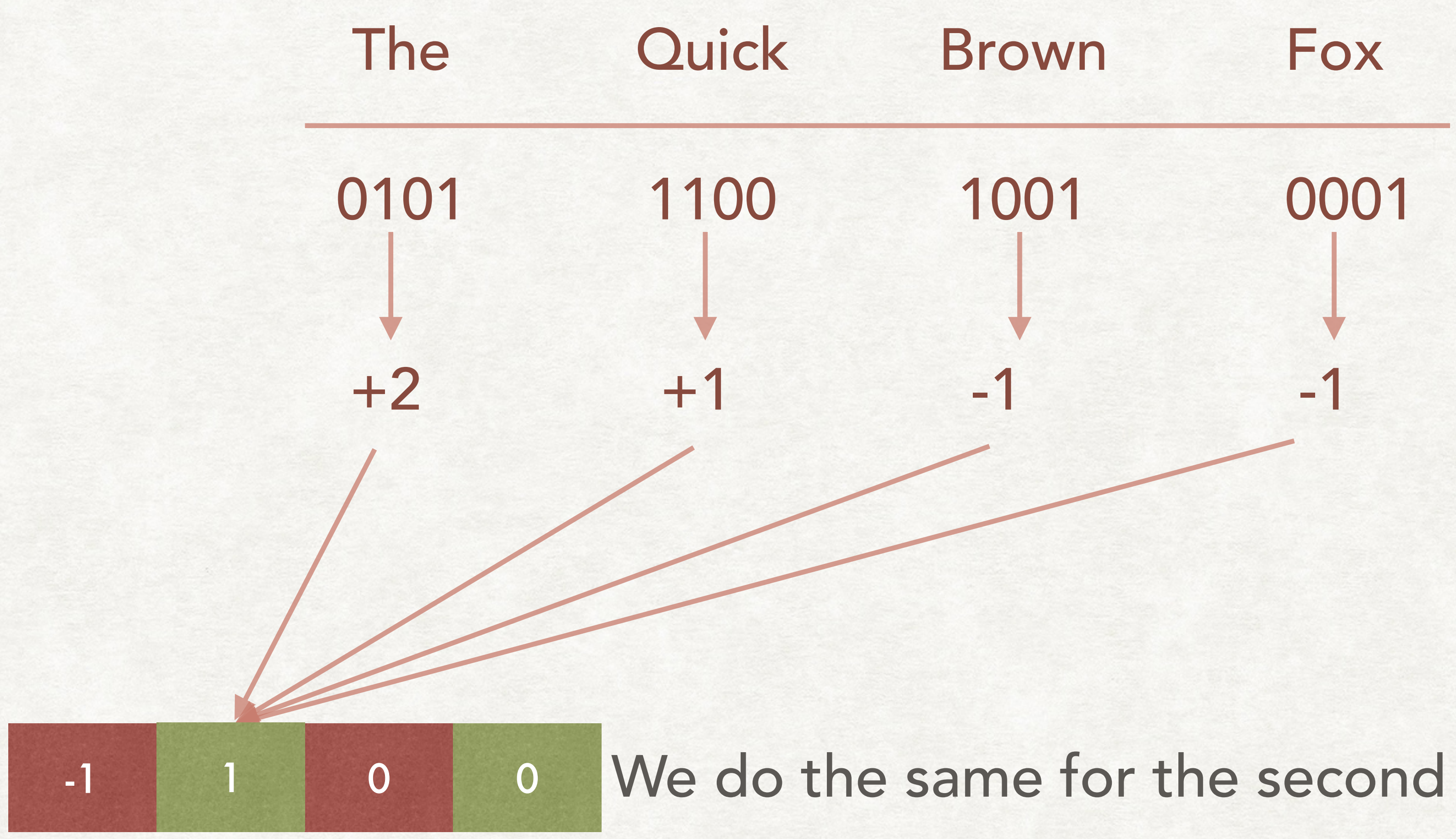
Add the weight to the word if the bit is 1.

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

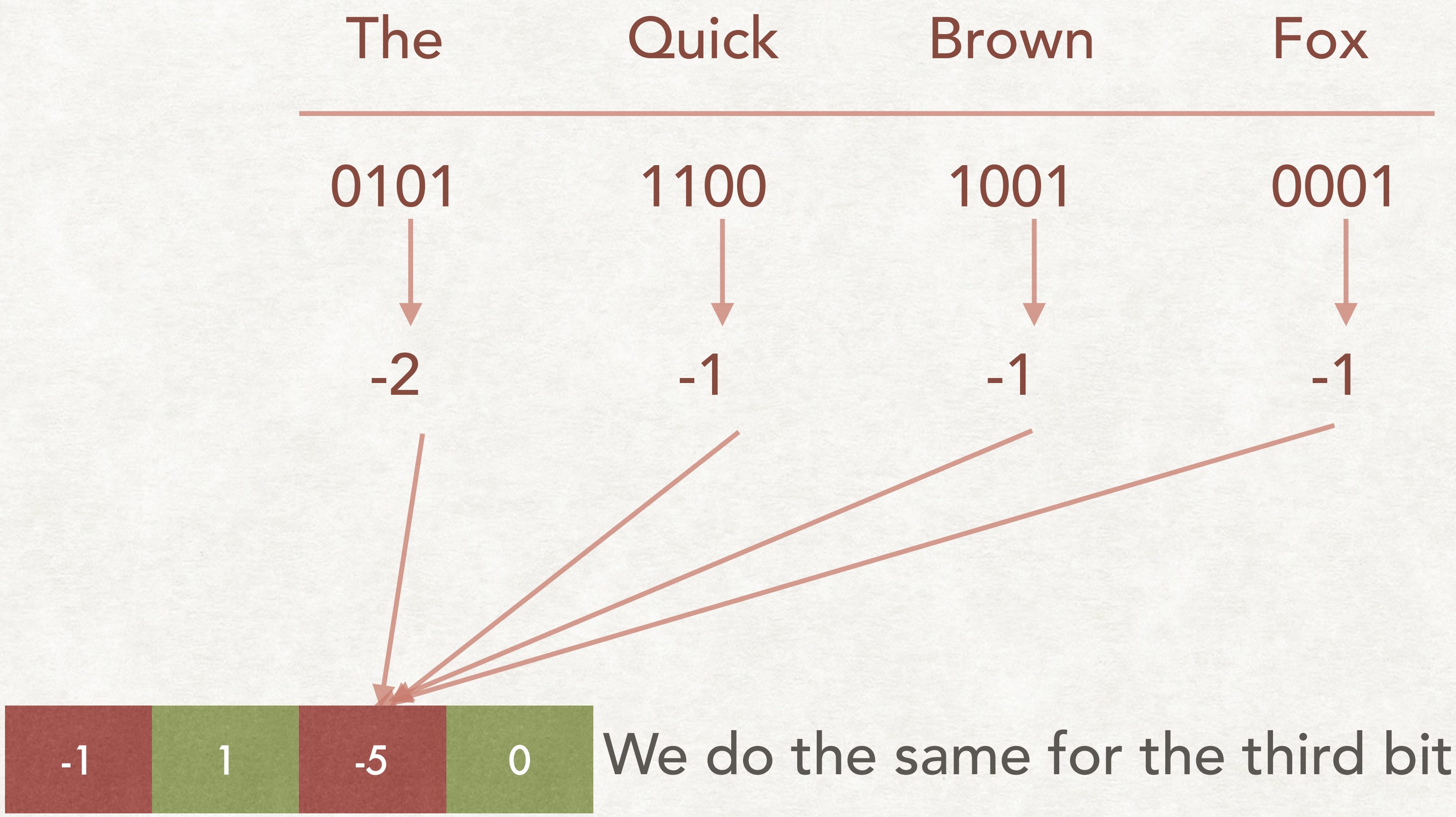
-1	1	0	0

We do the same for the second bit

SIMHASH

A MORE RECENT FINGERPRINTING TECHNIQUE

The	2
Quick	1
Brown	1
Fox	1



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

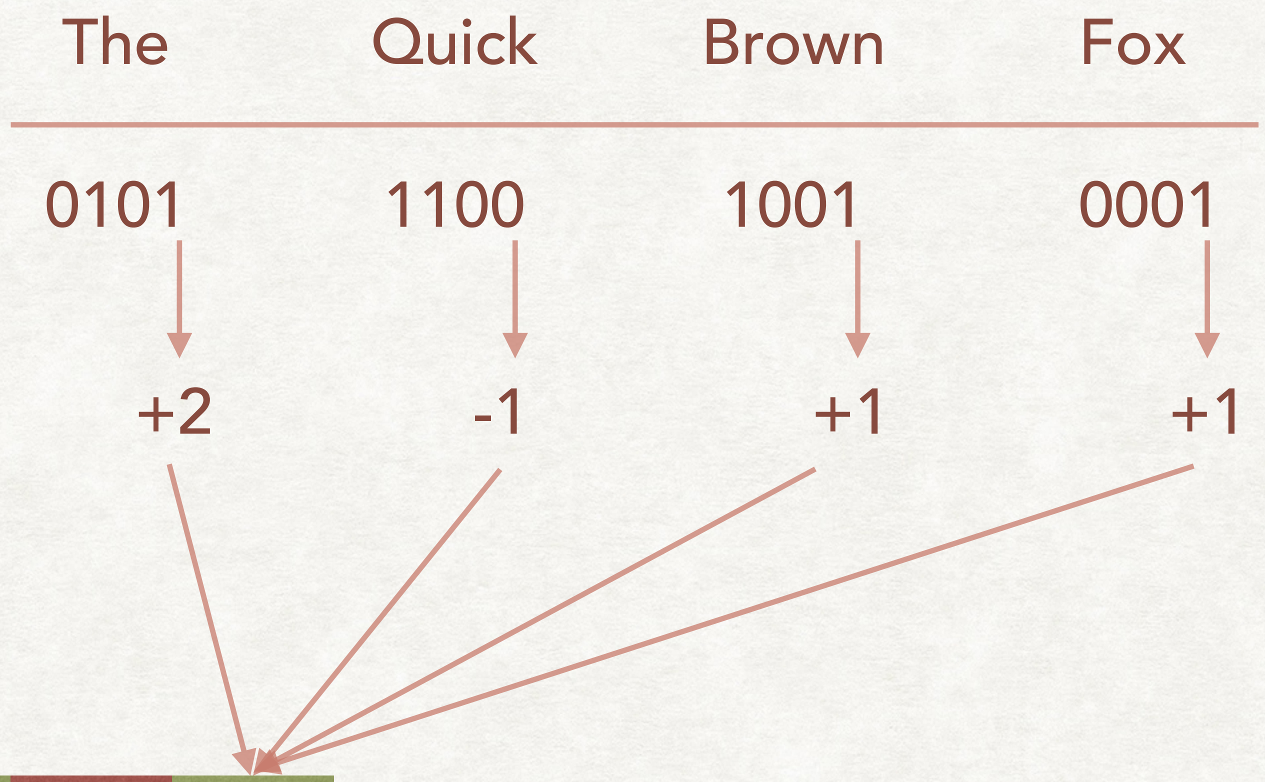
-1	1	-5	0

We do the same for the third bit

SIMHASH

A MORE RECENT FINGERPRINTING TECHNIQUE

The	2
Quick	1
Brown	1
Fox	1



-1	1	-5	3

We do the same for the fourth bit

The hash value of the document

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

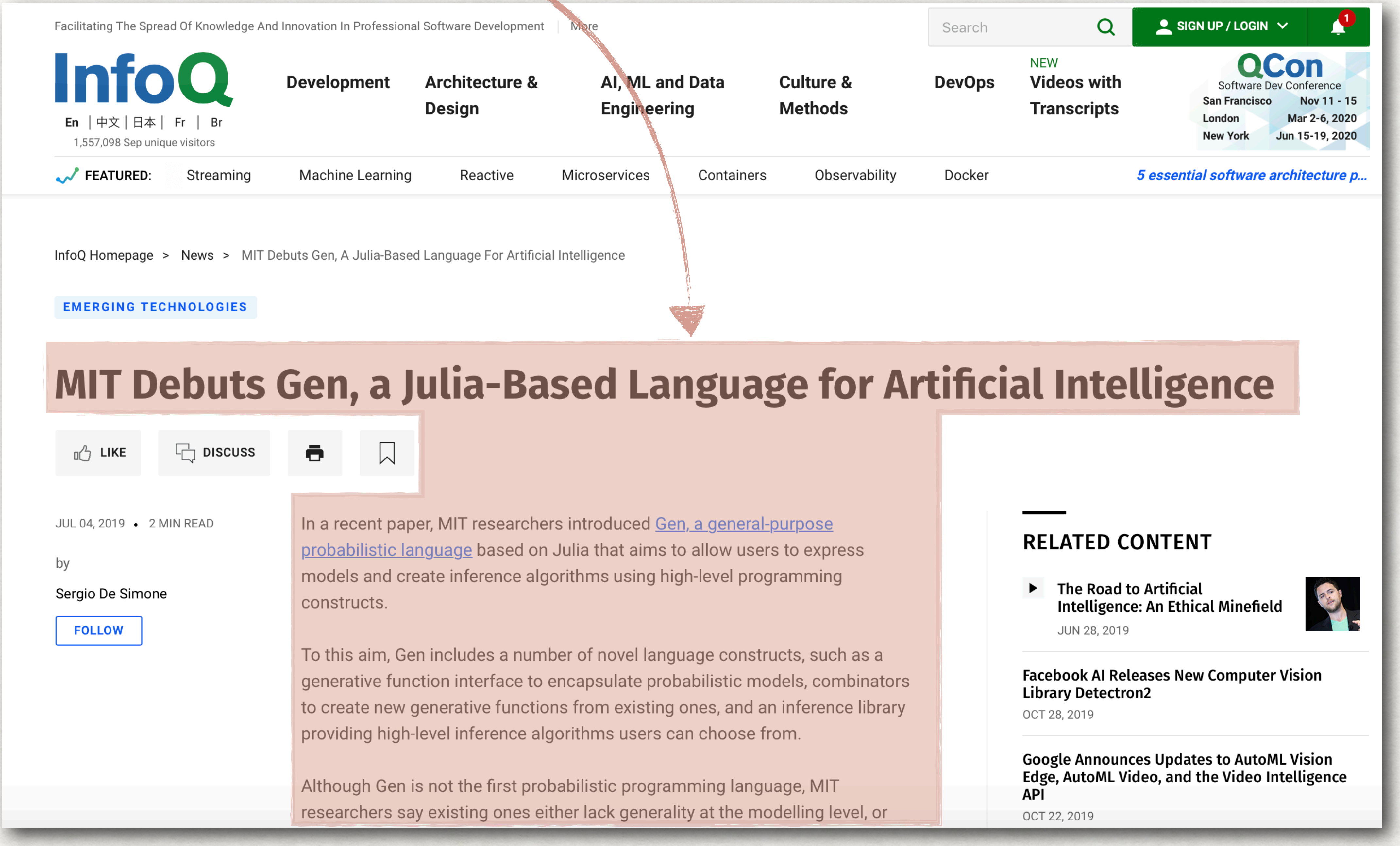
0	1	0	1

FINDING THE CONTENT

FINDING THE CONTENT

UNDERSTANDING THE PROBLEM

Main Content Block


Facilitating The Spread Of Knowledge And Innovation In Professional Software Development
More
Search
SIGN UP / LOGIN
1
InfoQ
En | 中文 | 日本 | Fr | Br
1,557,098 Sep unique visitors
Development
Architecture & Design
AI, ML and Data Engineering
Culture & Methods
DevOps
NEW
Videos with Transcripts
QCon
Software Dev Conference
San Francisco Nov 11 - 15
London Mar 2-6, 2020
New York Jun 15-19, 2020
FEATURED:
Streaming
Machine Learning
Reactive
Microservices
Containers
Observability
Docker
5 essential software architecture p...
InfoQ Homepage > News > MIT Debuts Gen, A Julia-Based Language For Artificial Intelligence
EMERGING TECHNOLOGIES
LIKE
DISCUSS
FOLLOW

MIT Debuts Gen, a Julia-Based Language for Artificial Intelligence

JUL 04, 2019 • 2 MIN READ

by

Sergio De Simone

In a recent paper, MIT researchers introduced Gen, a general-purpose probabilistic language based on Julia that aims to allow users to express models and create inference algorithms using high-level programming constructs.

To this aim, Gen includes a number of novel language constructs, such as a generative function interface to encapsulate probabilistic models, combinators to create new generative functions from existing ones, and an inference library providing high-level inference algorithms users can choose from.

Although Gen is not the first probabilistic programming language, MIT researchers say existing ones either lack generality at the modelling level, or

RELATED CONTENT

The Road to Artificial Intelligence: An Ethical Minefield
JUN 28, 2019

Facebook AI Releases New Computer Vision Library Detectron2
OCT 28, 2019

Google Announces Updates to AutoML Vision Edge, AutoML Video, and the Video Intelligence API
OCT 22, 2019

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.

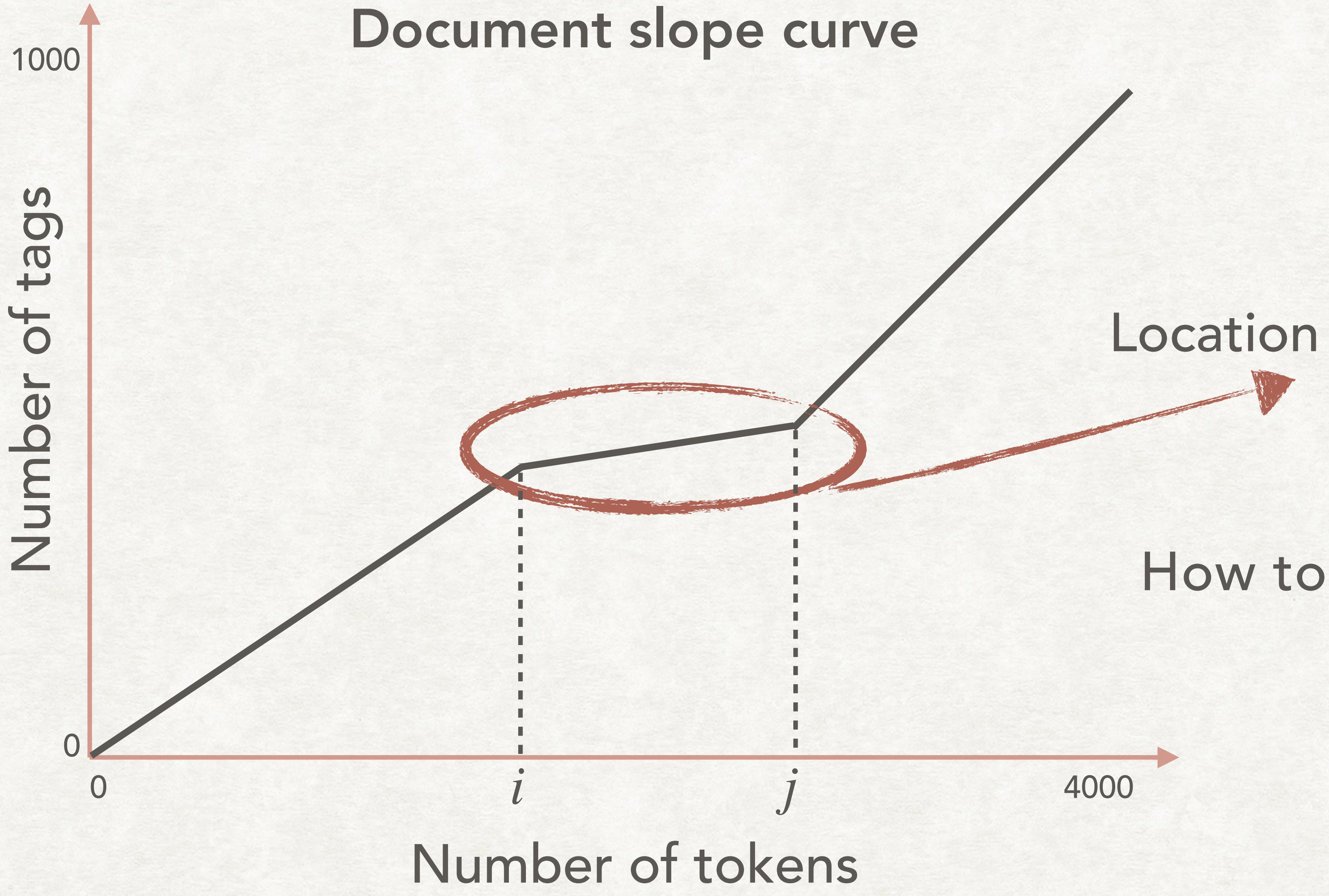
- From the point of view of the user the rest is *noise* that can have a negative effect on the ranking.

- 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



Location of the main content

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 $1 \le i < j \le N$ maximising:

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

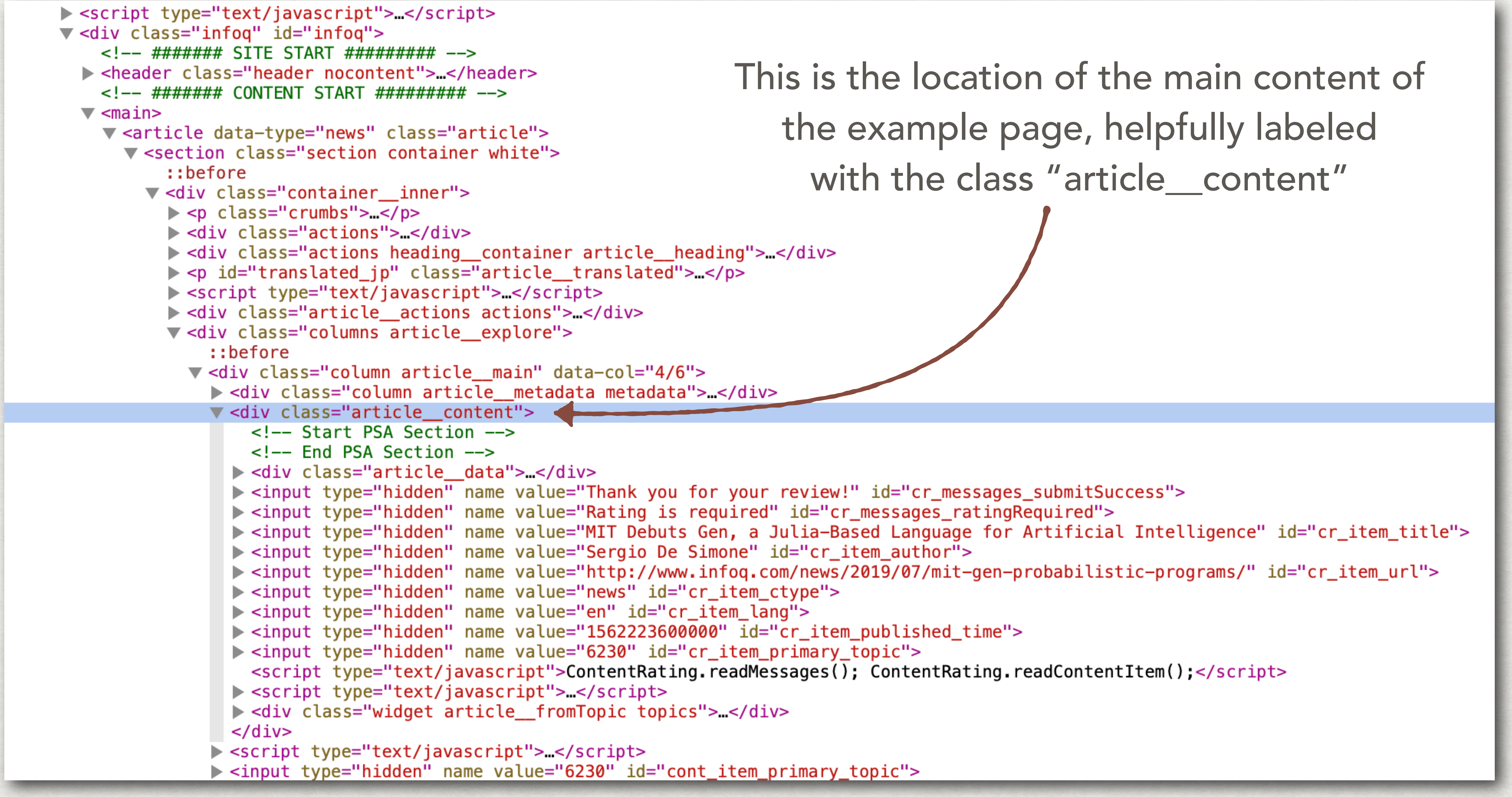
ANOTHER POSSIBILITY

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"