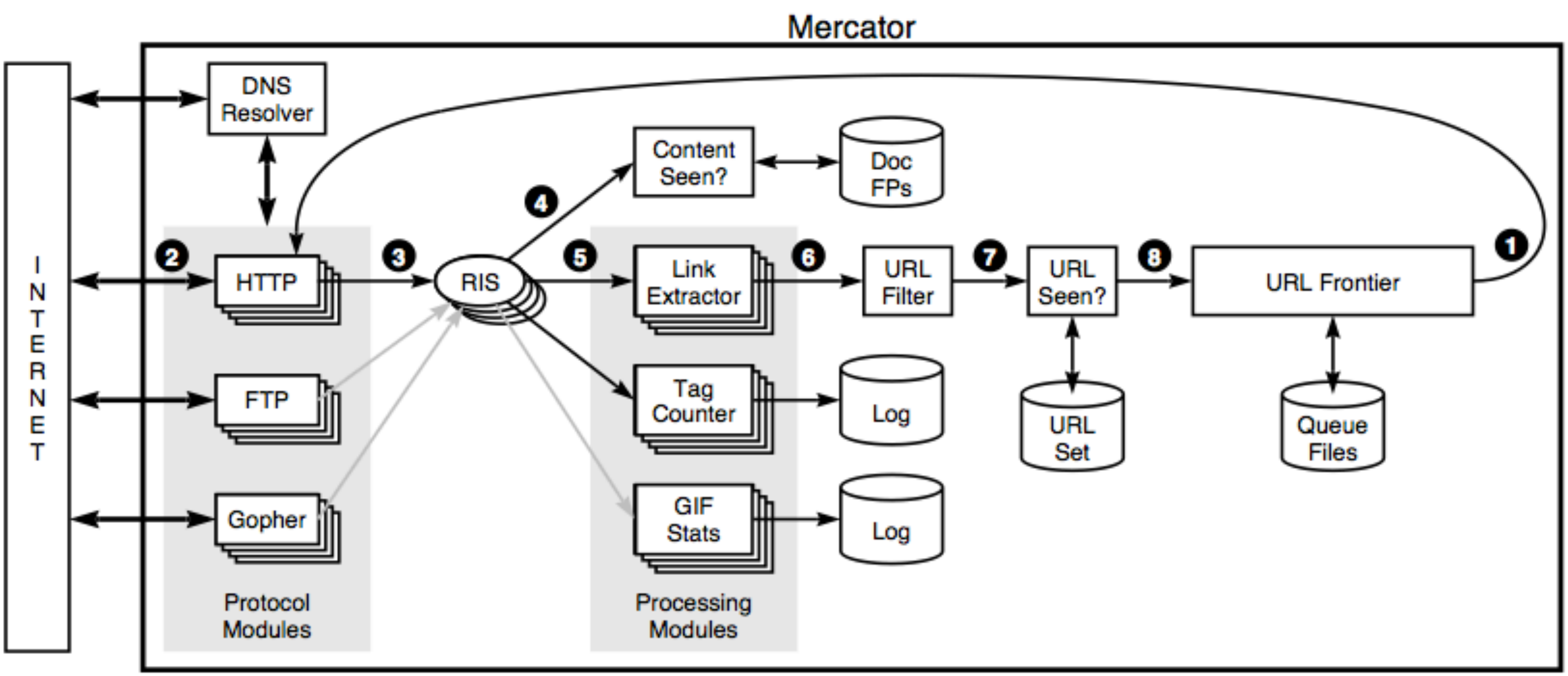
IR3: Web Search Implementation



Challenges

- Three challenges in web search:
  - Result relevance
  - Processing speed
  - Scaling to many documents
- So far we've discussed result relevance
- Today we'll cover speed and scaling

A Few Numbers

- 5 B - 100 B pages
  - Let's say 10 Billion for concreteness
- Assume 10KB per compressed page
  - 100 TB data to index
- 1 minute-1 month freshness
- 3-5B queries *per day* on Google alone

Outline

- Today we'll cover speed and scaling, including:
  - **Crawler design**
  - Inverted-index construction
  - Distributed search architecture
  - Deduplication

Crawlers

- To build a graph of the web, we need a program that visits every web page
- Web Robots AKA Web Wanderers, Crawlers, or Spiders
- Example: `googlebot`
  - http://www.robotstxt.org/db/googlebot.html

Crawler Design

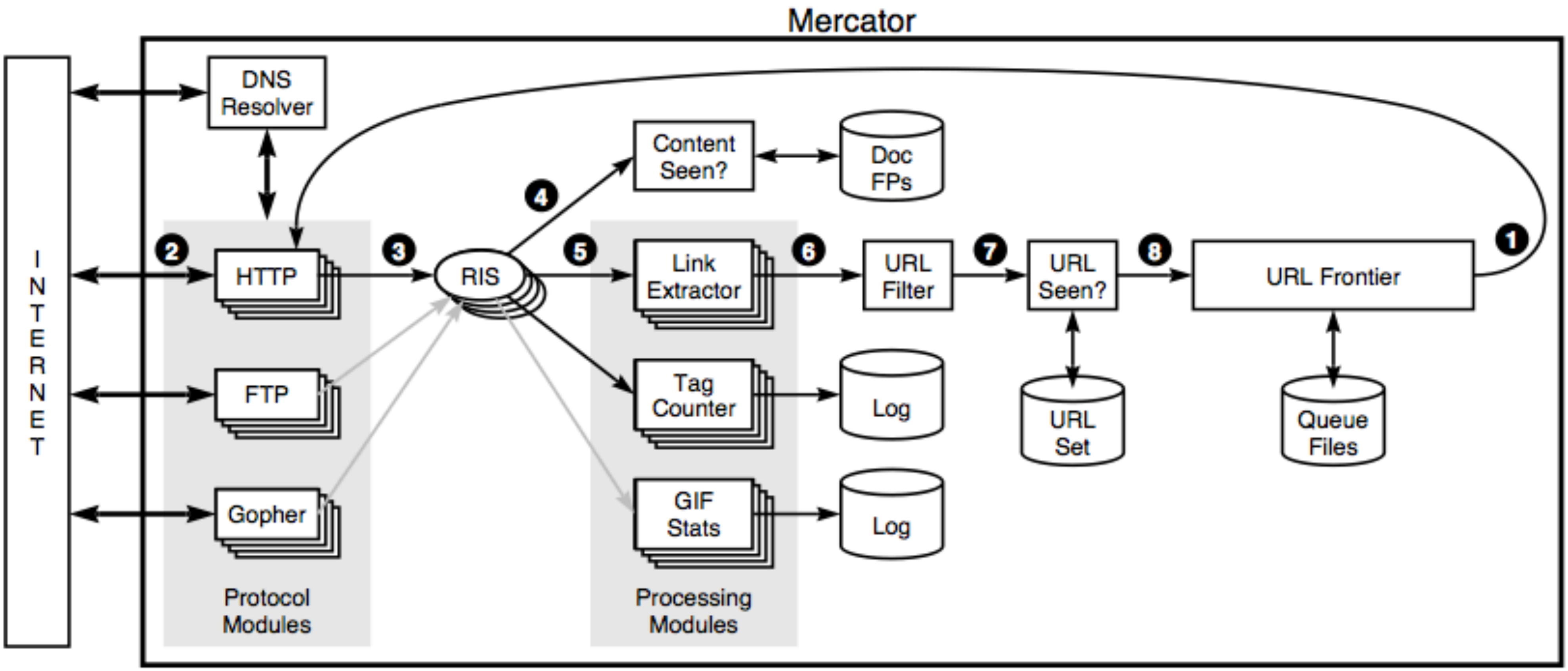
- To build a graph of the web, we need a program that visits every web page
- Web Robots AKA Web Wanderers, Crawlers, or Spiders

- Discussion: how would you do this?
- What data structures and algorithms would you use?

Mercator

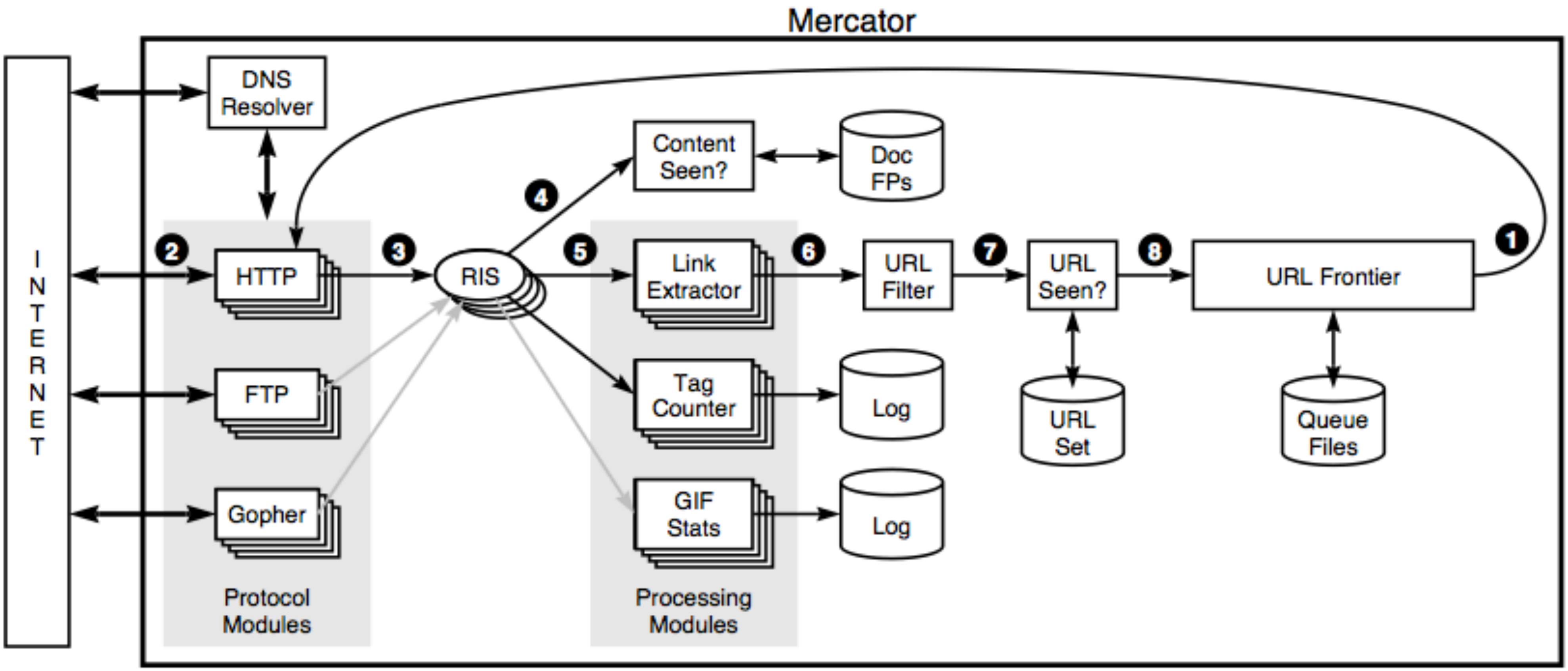
- Web crawler example: Mercator
- Mercator was the AltaVista crawler (1998)
- Exceptionally well-documented and useful, despite the many years
- Starts with seed URLs

http://webpages.uncc.edu/sakella/courses/cloud09/papers/Mercator.pdf



1. Remove URL from queue
2. Network protocols
3. Read w/ RewindInputStream (RIS)
4. Has document been seen before?

http://webpages.uncc.edu/sakella/courses/cloud09/papers/Mercator.pdf



5. Extract links
6. Download new URL?
7. Has URL been seen before?
8. Add URL to frontier

GOTO 1

http://webpages.uncc.edu/sakella/courses/cloud09/papers/Mercator.pdf

What is Crawled?

- All static web pages
  - Including selected non-HTML pages
  - Unless restricted by robots.txt
- What about dynamic web pages?
  - Crawler visits, but can index properly only if it can understand what the page is about

User agent

- When a browser or robot visits a page, it identifies itself with a `User-agent` string
  - For example, check yours out:
  - http://www.whatismyuseragent.net/
- Example from Google Chrome:
  - `Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/48.0.2564.103 Safari/537.36`
  - Previously used to indicate compatibility with the Mozilla rendering engine
  - During the "browser wars", some web sites would only send advanced features to some user agents

User agent

- You can spoof your user agent ☺
  - `curl -A "Mozilla/5.0" http://www.whatismyuseragent.net/`

- User agent switcher plug in for Chrome

User agent

- Similar to a browser, when robot visits a page, it identifies itself with a `User-agent`
  - Just like Firefox, Chrome, Safari, etc.
- You can request that robots not visit your site with `/robots.txt`

/robots.txt

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

- `User-agent: *`
- means this section applies to all robots.

- `Disallow: /`
- tells the robot that it should not visit any pages on the site.

/robots.txt

```
User-agent: Googlebot-Image
Disallow: /
```

- Tell Google Image search not to include images from your website

/robots.txt

```
User-agent:googlebot
Disallow:

User-agent: *
Disallow:/private/
Disallow:/~jag/pvt.html
```

- Default is to allow
- More specific Disallow applies

`/robots.txt`

- robots can ignore your `/robots.txt`
  - Malware robots that scan the web for security vulnerabilities
  - Email address harvesters used by spammers
- `/robots.txt` file is a publicly available file
  - Anyone can see what sections of your server you don't want robots to use
- `/robots.txt` directives can't prevent references to your URLs from other sites
  - A robot could navigate directly to a page from another website

Outline

- Today we'll cover speed and scaling, including:
  - Crawler design
  - **Inverted-index construction**
  - Distributed search architecture
  - Deduplication

Serving Results - speed

- After crawling is finished, we have a (big) database of documents
- To serve a search request, we need the terms in each doc

- You could just run `grep`
    - What is the complexity?

- How could we make this faster?

Serving Results - speed

- After crawling is finished, we have a (big) database of documents
- To server a search request, we need the terms in each doc

- You could just run `grep`
  - What is the complexity?
  - O(N), where N is the total size of all web docs!

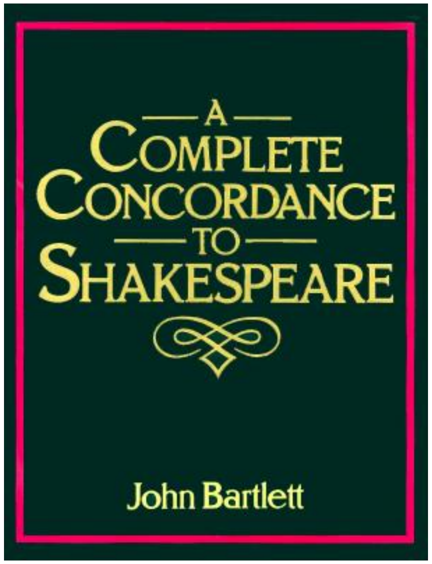
- How could we make this faster?

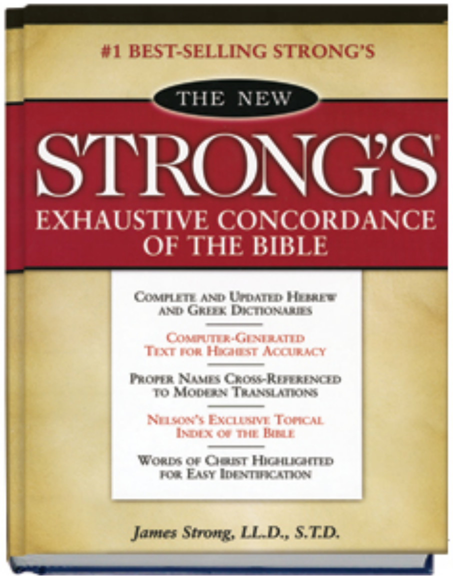
Inverted Index

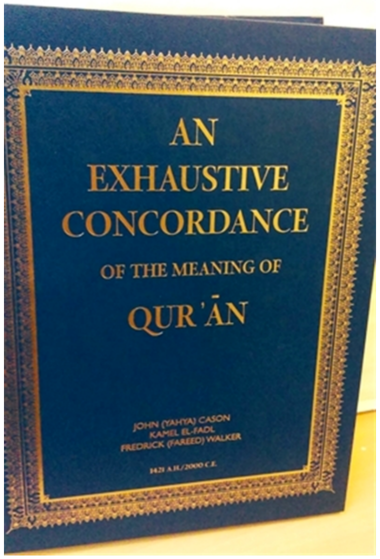
- A forward index is a list of words in each document
  - Doc -> words
- An *inverted index* maps words to docs that contain those words
  - Word -> docs
- Basic idea is: for each word, list all the documents where that word can be found
- Key to fast query processing

Inverted Index

- Inverted indexes were around before computers
- Example: concordance
- List of every word, in alphabetical order
- Constructed manually before computers!!!





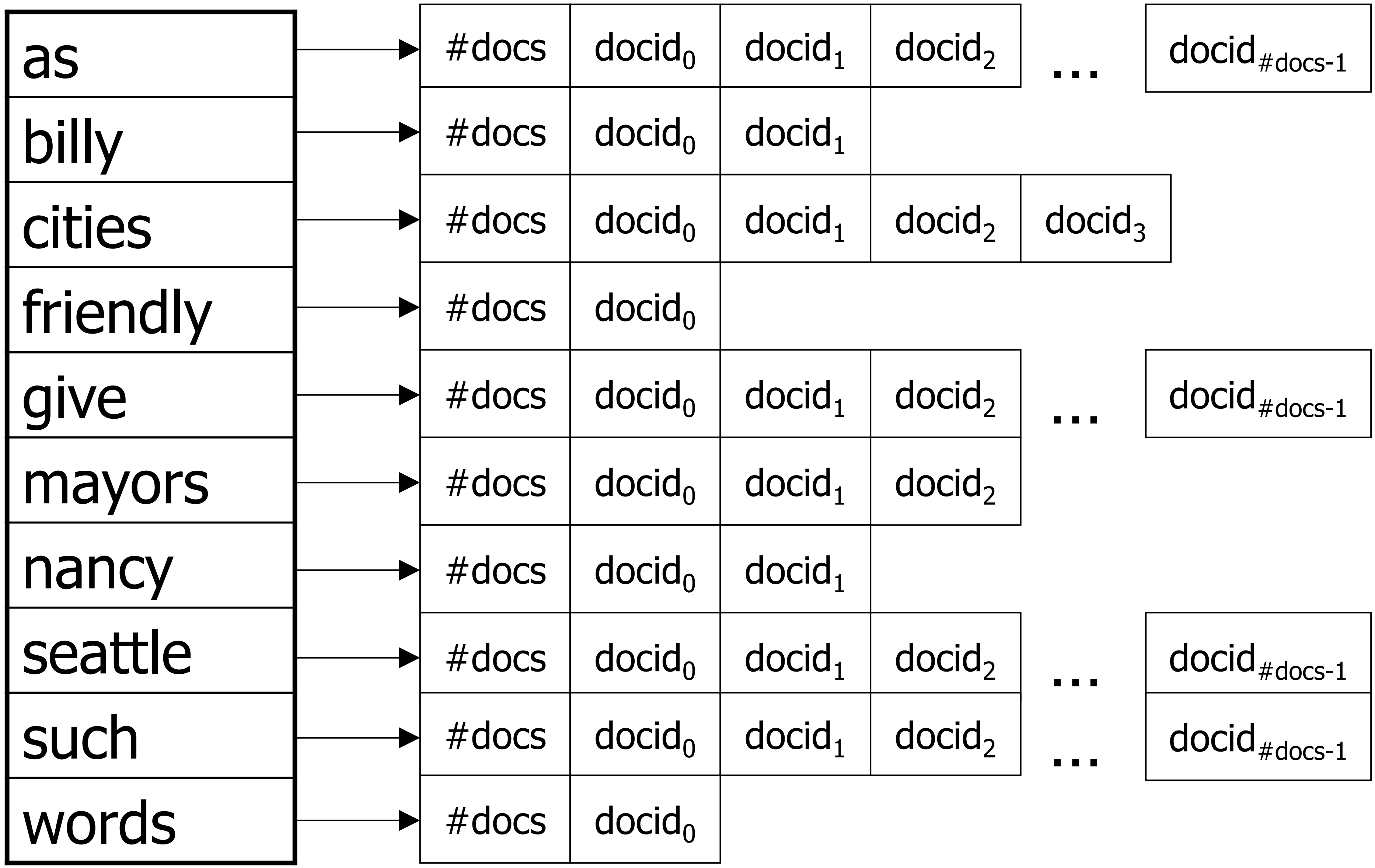


Inverted Index

- openshakespeare.org concordance: `horatio`

#	Work	Character	Line	Text
1	**Hamlet** [I, 1]	**Bernardo**	13	Well, good night. If you do meet **Horatio** and Marcellus, The rivals of my watch, bid them make haste.
2	**Hamlet** [I, 1]	**(stage directions)**	16	Enter **Horatio** and Marcellus.
3	**Hamlet** [I, 1]	**Bernardo**	26	Say- What, is **Horatio** there ?
4	**Hamlet** [I, 1]	**Bernardo**	29	Welcome, **Horatio**. Welcome, good Marcellus.
5	**Hamlet** [I, 1]	**Marcellus**	32	**Horatio** says 'tis but our fantasy, And will not let belief take hold of him Touching this dreaded sight, twice seen of us. Therefore I have entreated him along, With us to watch the minutes of this night, That, if again this apparition come, He may approve our eyes and speak to it.
6	**Hamlet** [I, 1]	**Marcellus**	54	Thou art a scholar; speak to it, **Horatio**.

http://www.opensourceshakespeare.org/search/search-results.php?link=con&searchtype=exact&hamlet&keyword1=horatio


as
billy
cities
friendly
give
mayors
nancy
seattle
such
words
#docs docid$_0$ docid$_1$ docid$_2$ … docid$_{\#docs-1}$
#docs docid$_0$ docid$_1$
#docs docid$_0$ docid$_1$ docid$_2$ docid$_3$
#docs docid$_0$
#docs docid$_0$ docid$_1$ docid$_2$ … docid$_{\#docs-1}$
#docs docid$_0$ docid$_1$ docid$_2$
#docs docid$_0$ docid$_1$
#docs docid$_0$ docid$_1$ docid$_2$ … docid$_{\#docs-1}$
#docs docid$_0$ docid$_1$ docid$_2$ … docid$_{\#docs-1}$
#docs docid$_0$

Exercise

- Suggest an algorithm for finding a list of docs for a search
- Example: "such as"
  - Looking for keywords, not phrase
- HINT: does it help if the docs are sorted by ID?

Exercise

- Suggest an algorithm for finding a list of docs for a search
- Example: "such as"
- One solution: merge intersection

Query: **such as**

as
billy
cities
friendly
give
mayors
nancy
seattle
such
words

#docs	$docid_0$	$docid_1$	$docid_2$	…	$docid_{\#docs-1}$
104	21	150	322		2501

1. Test for equality
2. Advance smaller pointer
3. Abort when a list is exhausted

#docs	$docid_0$	$docid_1$	$docid_2$	…	$docid_{\#docs-1}$
15	99	322	426		1309

Returned docs: 322

Inverted Index Construction

- Inverted index is very large.
  - Full text index usually larger than the text.
  - How can we build it efficiently?
- Remember:
  - Disk seeks are very expensive (5ms)
  - Continuous disk reads or writes are OK (50-120MB/sec)
  - Machines can have a lot of memory (often hundreds of GB), but disk is always much cheaper
  - Input is the tokenized document set

Basic Tasks

1. Compile term-termid, doc-docid maps
2. Assemble all termid-docid pairs
3. Sort pairs first by termid, then docid
4. Write out in inverted-index form

- **EASY!**
- Well, not if docs won’t fit into memory

Block sort-based indexing

- *External sort algorithms* work on sets larger than memory
- Block-Sort-Based Index Algorithm:
  n = 0
  While **docsRemain**
      n++
      block = ParseNextBlock()
      BSBI-Invert(**block**)
      WriteToDisk(block, f_n)
  MergeBlocks(f_1, …, f_n) => f_{merged}

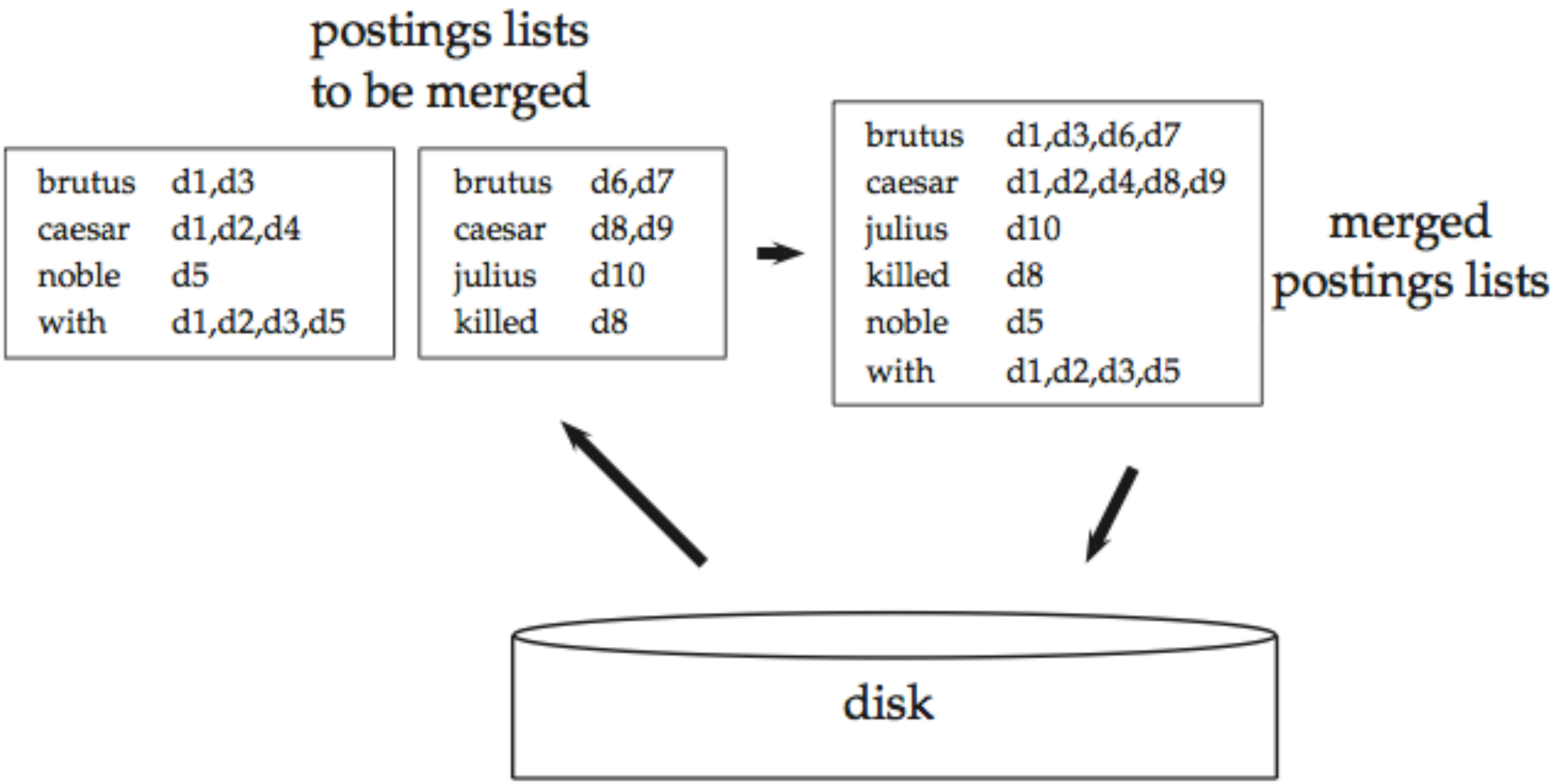
Block sort-based indexing

- ParseNextBlock accumulates termid-docid pairs in memory until block is full
- BSBI-Invert generates small in-memory inverted index

- So: we build a series of small in-memory inverted indexes, writing each one to disk
- Finally: we merge them

Block merging

Merging two lists:



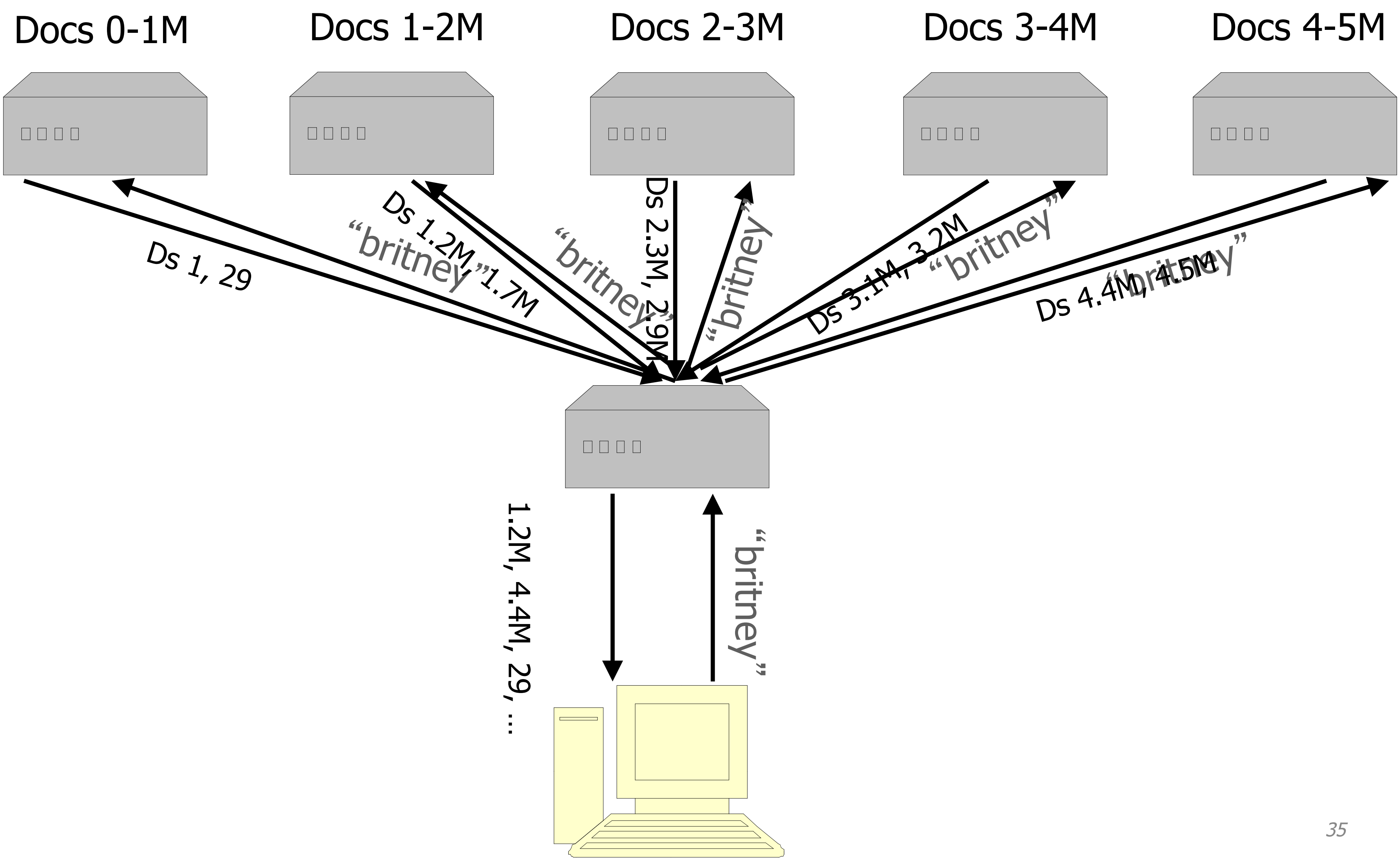
Outline

- Today we'll cover speed and scaling, including:
  - Crawler design
  - Inverted-index construction
  - **Distributed search architecture**
  - Deduplication

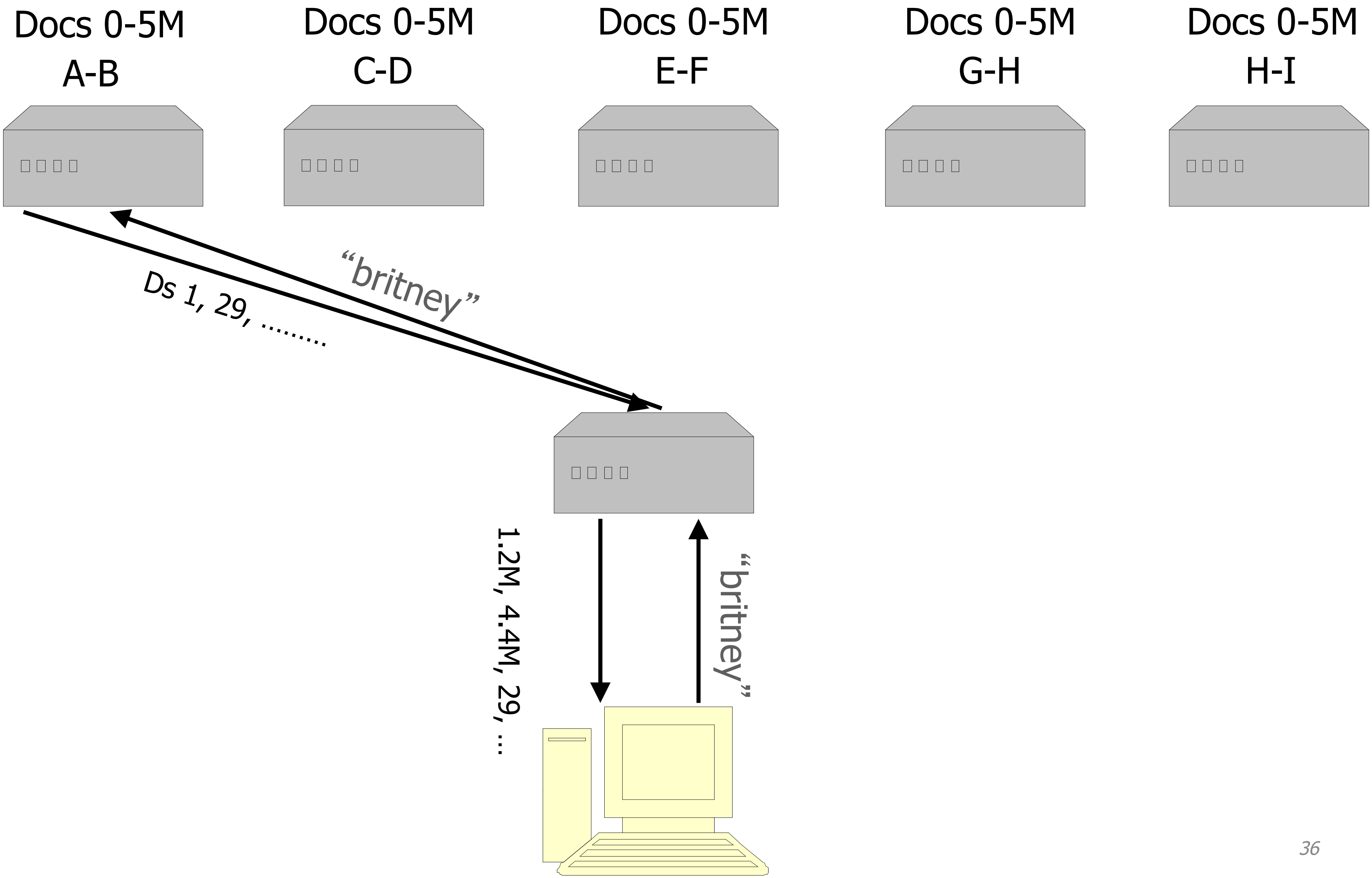
Distributed Searching

- Not even the inverted index is small enough for one machine to handle it
  - Billions of docs
  - Hundreds of millions of queries
- Also, what if the machine fails?
- Need to parallelize query processing
  - Segment by document
  - Segment by search term

Scaling - doc segmenting



Scaling - term segmenting



Segmentation

- What are the trade-offs of segment by documents vs. segment by term?
- What happens if a machine dies?

Segmentation

- Segment by document
  - Easy to partition (just MOD the docid)
  - Easy to add new documents
  - If machine fails, quality goes down but queries don't die
- Segment by term
  - Harder to partition (terms uneven)
  - Trickier to add a new document (need to touch many machines)
  - If machine fails, search term might disappear, but not critical pages (e.g., yahoo.com/index.html)

Outline

- Today we'll cover speed and scaling, including:
  - Crawler design
  - Inverted-index construction
  - Distributed search architecture
  - **Deduplication**

Deduplication

- How can you be sure a web page is worth indexing?
  - Has it changed meaningfully?
  - A clone of another site? (Weirdly common)

- Two problems to solve:

1. Are these two documents duplicates?
2. Find all duplicates

Are these two documents duplicates?

- Pages can have thousands of words. Comparing a pair of pages takes time.
  - How can you generate a page fingerprint?
  - What about a near-fingerprint?

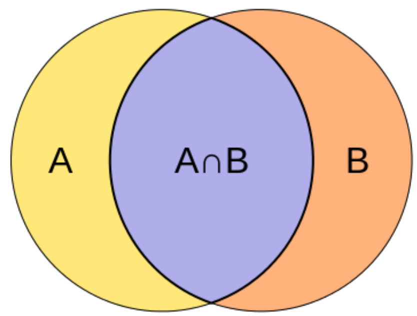
Shingling

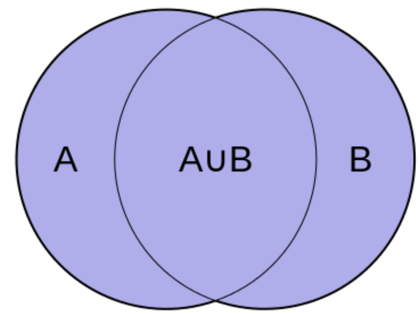
- Compute the k-shingles for a page
- "I think EECS 485 is a great class"
  - For k=3: "I think EECS", "think EECS 485", "EECS 485 is", etc
- If docs share lots shingles, they're dups
- Document now a *set* of shingles
- Convert a document comparison problem into a set comparison problem
  - How similar are two sets?

Jaccard similarity coefficient

- Jaccard similarity coefficient compares the similarity of the two sets of shingles (A and B)
- Size of the intersection / size of the union

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}.$$





- 0 for disjoint sets, 1 for equal sets
- What is the complexity?
- Assume A and B are size O(N)

Jaccard similarity coefficient

- What is the complexity?
- Assume A and B are size O(N)

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}.$$

- O(N) time with O(N) space using hash tables

Jaccard similarity coefficient

- Computing the Jaccard similarity coefficient is too slow for large documents
- Let's estimate it

- First, a question:
- Pick a random shingle from `A U B`
- What is the probability that this shingle is in the intersection?

Jaccard similarity coefficient

- Pick a random shingle from A U B
- What is the probability that this shingle is in the intersection?

$$\frac{|A \cap B|}{|A \cup B|}.$$

- That's the Jaccard similarity coefficent

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}.$$

Selecting shingles

- How can we efficiently select a random shingle that is present in at least one of A or B?
  - In other words, a random shingle from `A U B`

- A similar problem: select a random element from an array
  - Size is not known
  - Hint: let's say you have a perfect hash function

Min hash

- Hash each element and choose the element that corresponds to the min of the hashed values
  - The hash function maps inputs uniformly over the output
  - Selecting the *min(h(x))* is the same as selecting a random item *x*!
- This is called min hash

Min hash and duplicate detection

- Back to doc A and doc B
- Want to compare a random shingle that is present in at least one doc (A or B)
- Hash all the shingles
  - h(shingle) -> integer
- *min(h(shingle))* corresponds to a random shingle
- This shingle will be in A and B with probability

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}.$$

- Again, the Jaccard similarity coefficient

Min Hash Idea

- Now, pick the first *k* hashed values
- Alternatively, use *k* hash functions, and pick the min of each

- Finally, set union with k values instead of N values

Information Retrieval

- At the heart of web search
- Document-vector model, network model
- Metrics: precision, recall, Kendall's Tau
- Implementation:
  - Crawler
  - Inverted Index
  - Deduplication: minhash, Jacard Coefficient.

- Basis for project 5
- Semester-long course in EECS 486