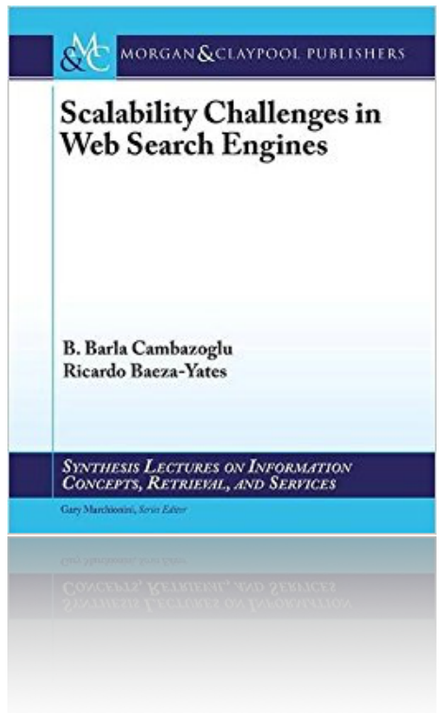
Disclaimer

- These slides have been prepared by B. Barla Cambazoglu, Director of Applied Science, NTENT Inc.

- Used in several IR tutorials and schools by Ricardo Baeza-Yates and B. Barla Cambazoglu.

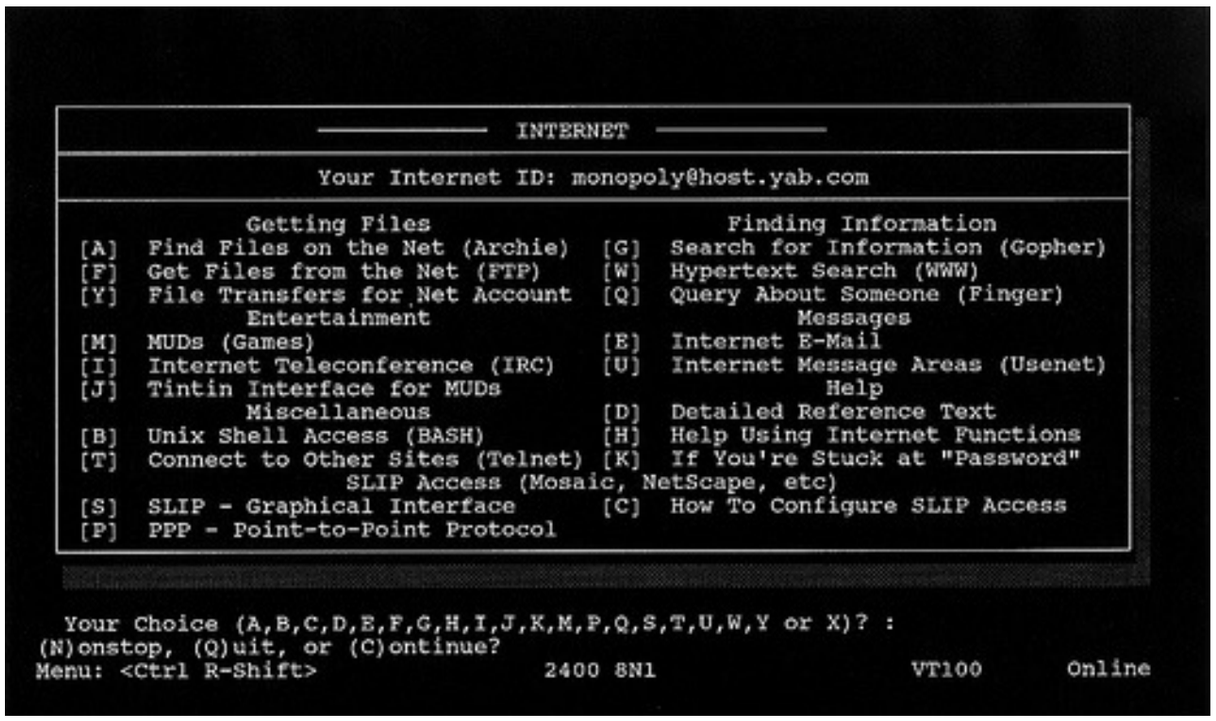
- Used with permission, please do not redistribute in any form.

- Please refer to:



Brief History of Web Search Engines

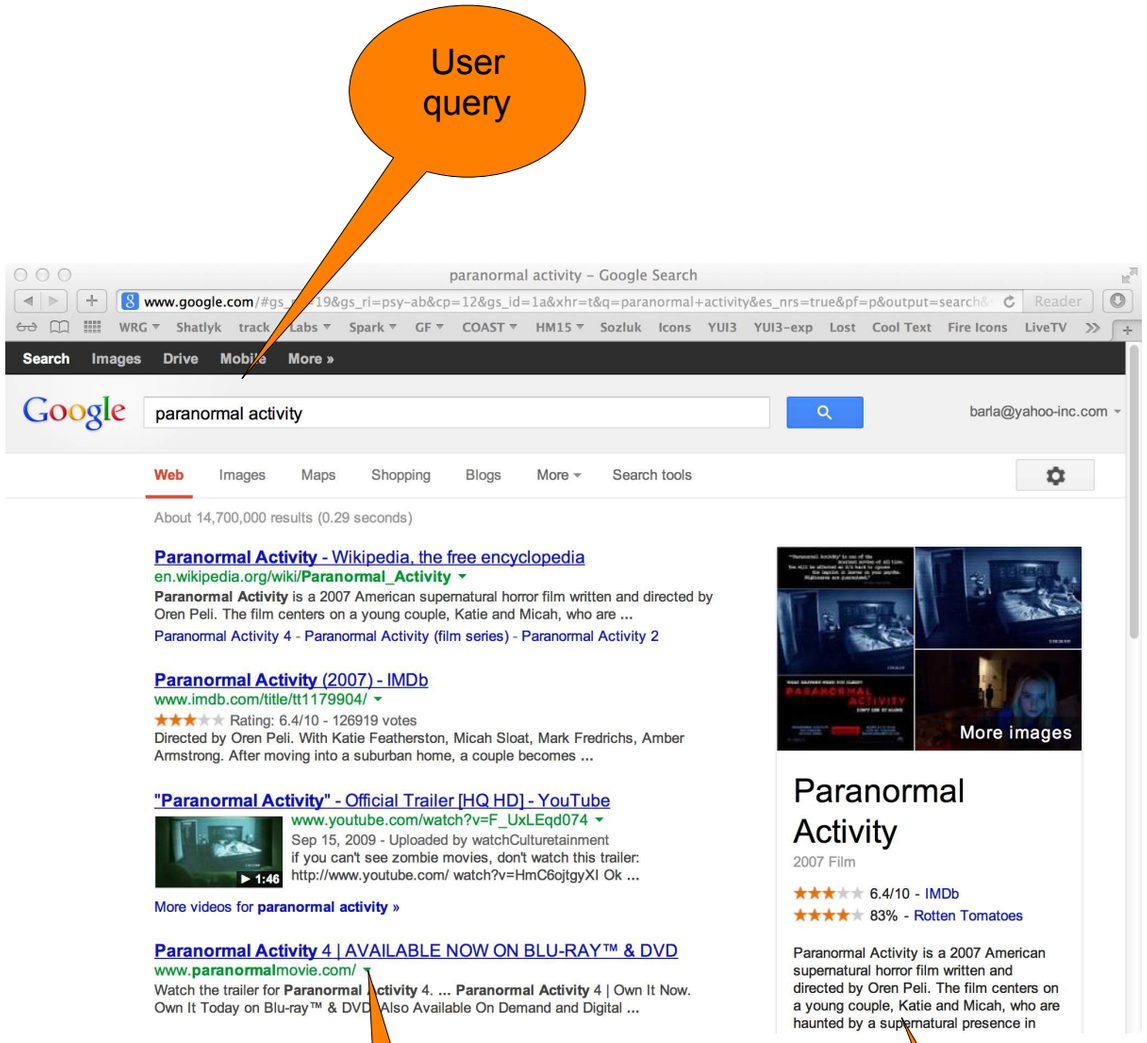
- Past
  - Before browsers
    - Gopher
  - During the bubble
    - Altavista
    - Lycos
    - Infoseek
    - Excite
    - HotBot
  - After the bubble
    - Google
    - Yahoo
    - Microsoft



- Now
  - Global
    - Google, Bing
  - Regional
    - Yandex, Baidu, Naver

- Future
  - Apple?
  - Facebook ?
  - …

Anatomy of a Search Engine Result Page



Actors in Web Search

- User's perspective: accessing information
  - high-quality search results
  - fast response to queries

- Search engine's perspective: monetization
  - attract more users
  - increase the ad revenue
  - reduce the operational costs

- Advertiser's perspective: publicity
  - attract more users to their site
  - pay little

What Makes Web Search Difficult?

- The Web
  - around 1.25 billion unique host names
  - estimated to contain hundreds of billion of pages
  - constantly changing
  - diverse in terms of content and data formats

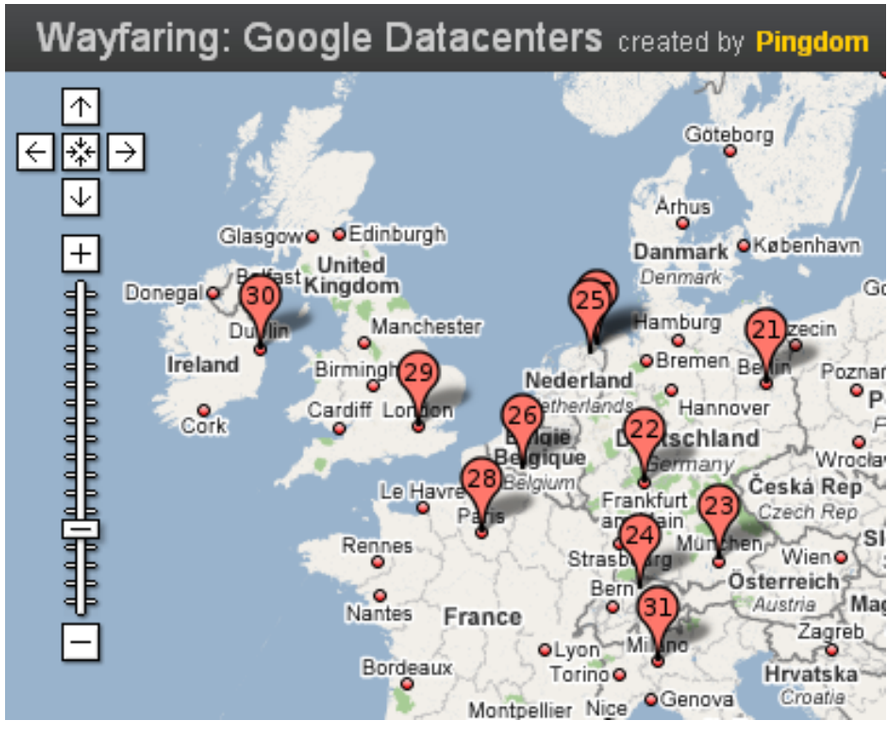
- Users
  - too many
  - diverse in terms of their culture, education, and demographics
  - very short queries (hard to understand the intent)
  - changing information needs
  - little patience (few queries posed & few answers seen)

Expectations from a Search Engine

- Crawl and index a large fraction of the Web.
- Maintain most recent copies of the content in the Web.
- Serve high-quality results for user queries.
- Scale to serve hundreds of millions of queries every day.
- Evaluate a typical query under several hundred milliseconds.

Search Infrastructure

- Quality and performance requirements imply large amounts of compute resources, i.e., very large data centers.
- High variation in data center sizes
  - hundreds of thousands of computers
  - a few computers

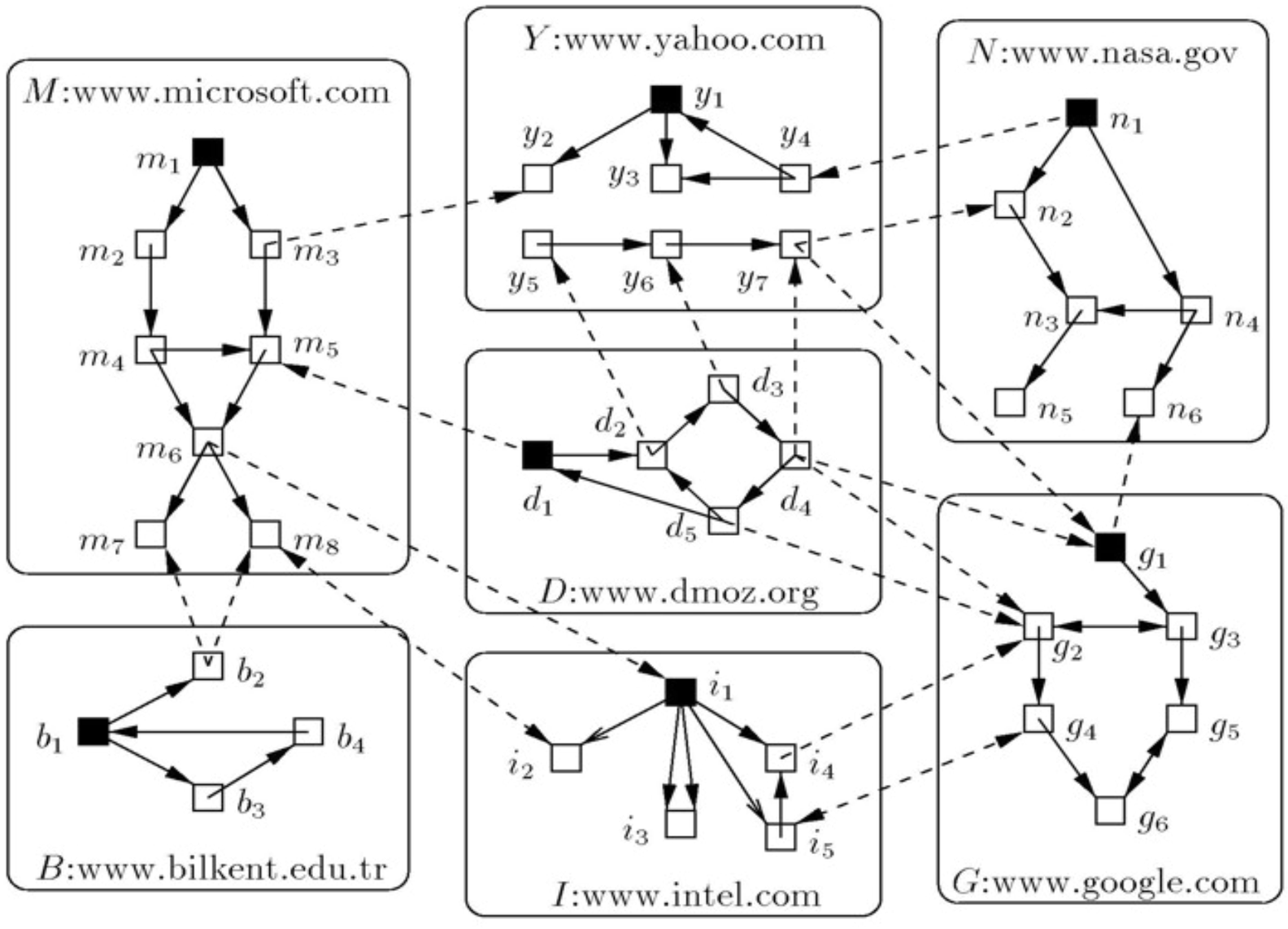


Web Crawling

- Web crawling is the process of locating, fetching, storing, and maintaining the pages available in the Web.
- Computer programs that perform this task are referred to as
  - crawlers
  - spiders
  - harvesters
- Web crawler repositories
  - cache the online content in the Web
  - provide quick access to the physical copies of pages in the Web
  - help to speed up the indexing process

Web Graph

- Web crawlers exploit the hyperlink structure of the Web to discover new web pages.

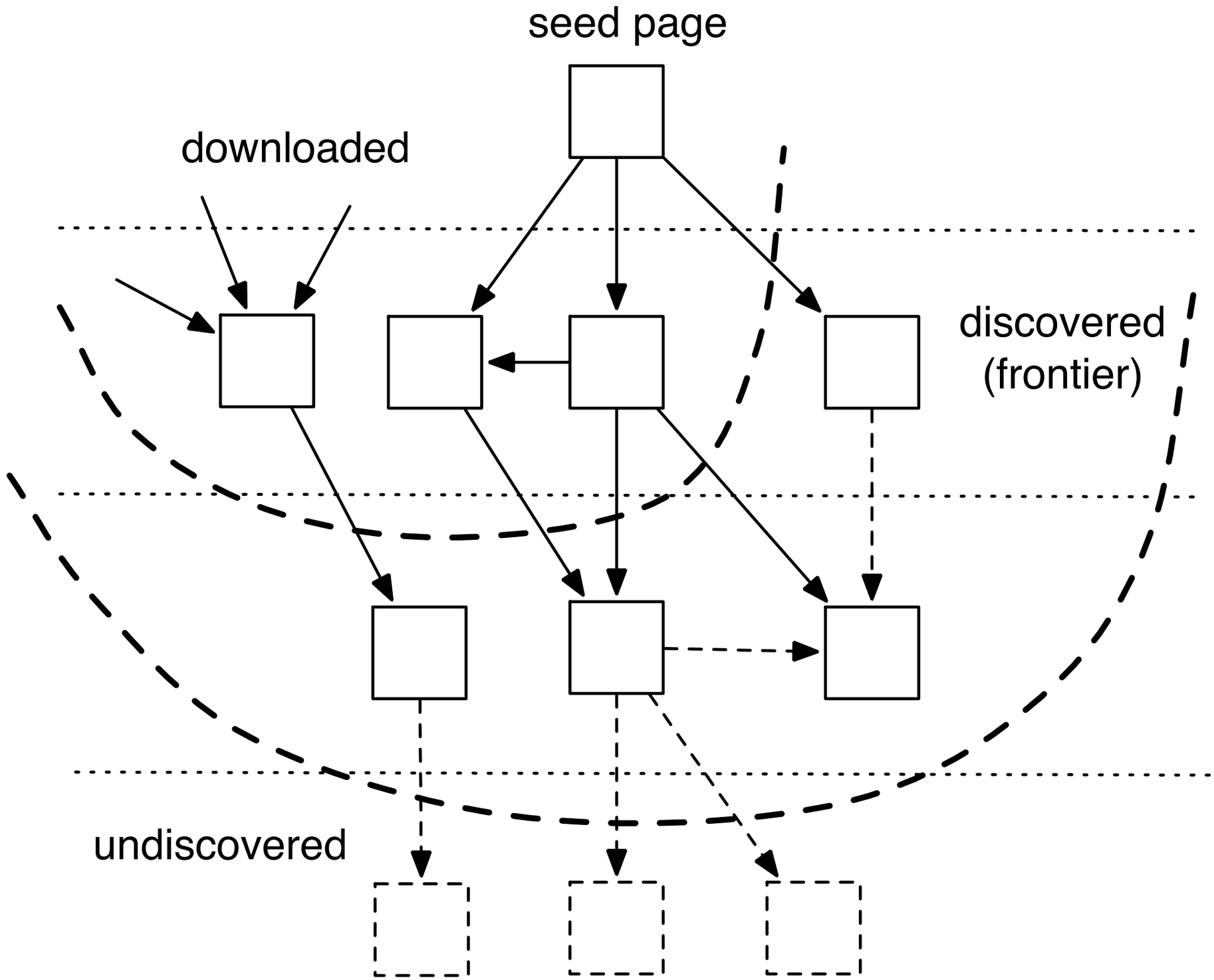


Batch-Mode Web Crawling

In each session, a typical batch-mode web crawler

1. inserts a set of seed pages into a fetch queue,
2. downloads a page from the fetch queue,
3. stores the downloaded page in a repository,
4. parses the page and extracts its outbound links,
5. adds the unseen links to the fetch queue,
6. repeats steps 2–6 until the fetch queue gets empty or a satisfactory number of pages are downloaded.

Incremental Web Crawling

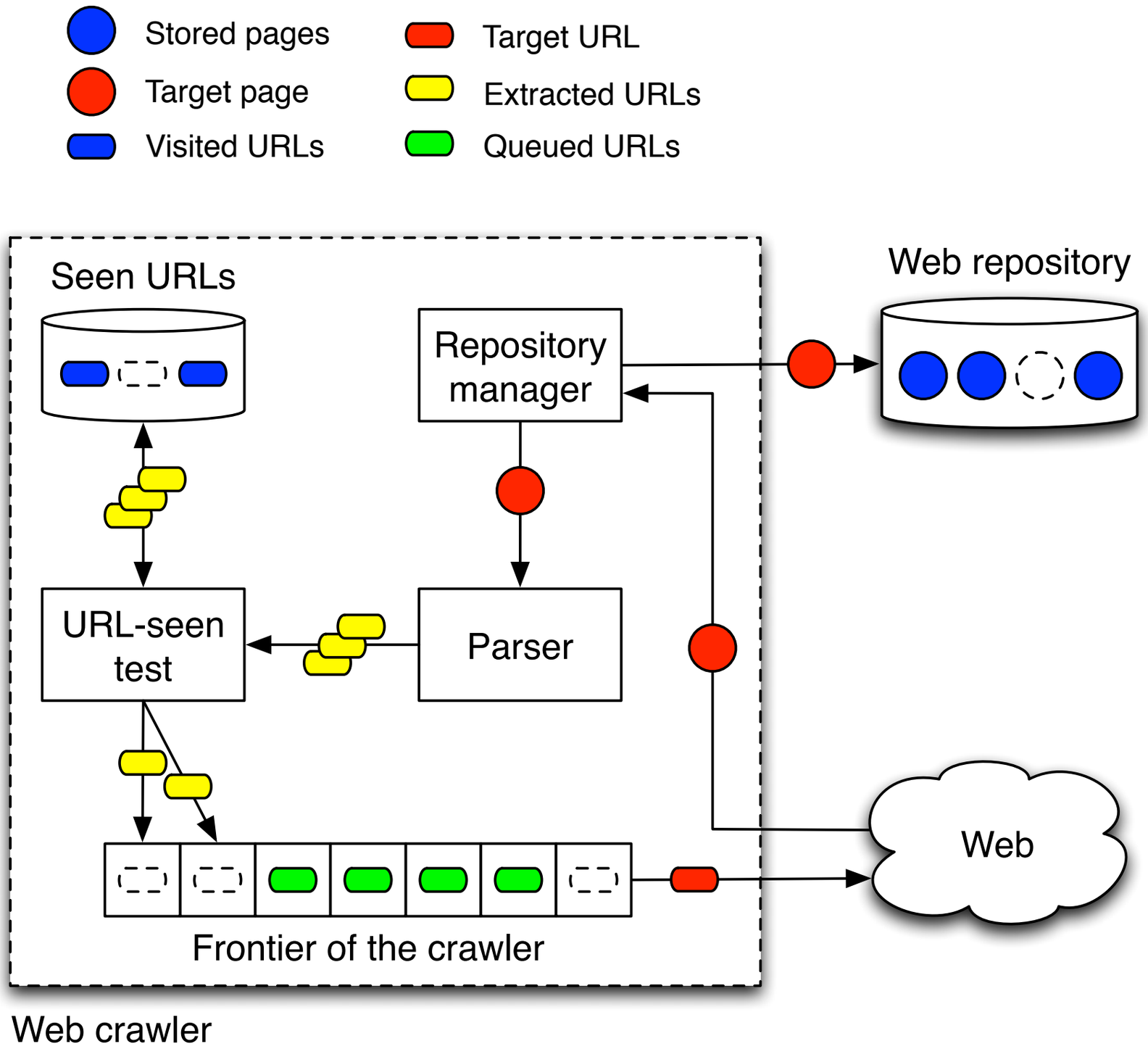


Queues

Commercial web crawlers maintain two download queues

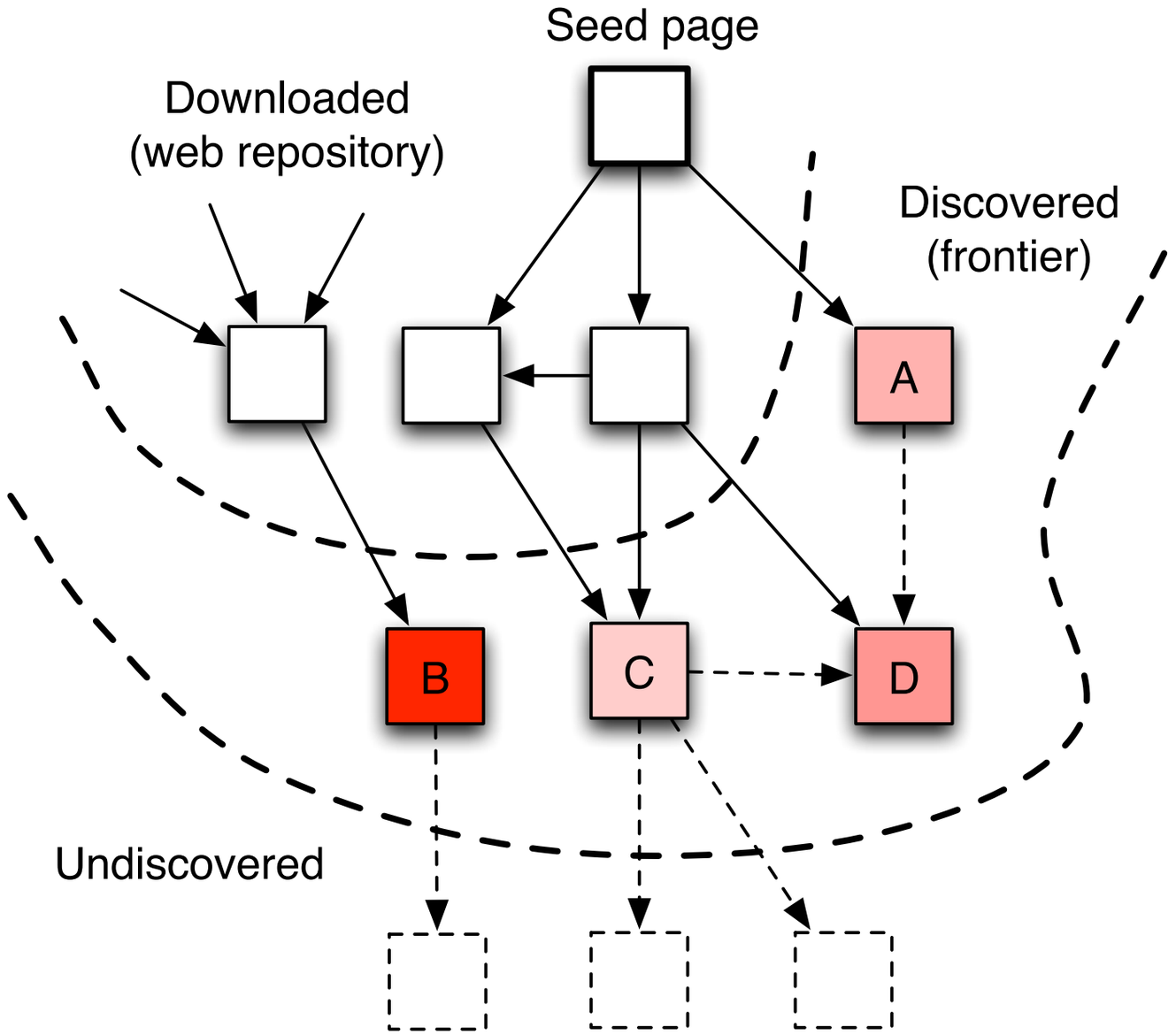
- discovery queue
  - downloads pages pointed by already discovered links
  - tries to increase the coverage of the crawler
- refreshing queue
  - redownloads already downloaded pages
  - tries to increase the freshness of the repository

Discovery


Stored pages
Target page
Visited URLs
Target URL
Extracted URLs
Queued URLs
Seen URLs
Repository manager
Web repository
URL-seen test
Parser
Web
Frontier of the crawler
Web crawler

URL Prioritization

- Random (A, B, C, D)
- Breadth-first (A)
- In-degree (C)
- PageRank (B)



Refreshing

	A	B	C	D
PageRank	0.0003	0.0007	0.0002	0.0001
Average daily click count	47	332	2	1974
Last download time	2 hours ago	1 day ago	8 days ago	6 hours ago
Estimated update frequency	daily	never	minutely	yearly

- Random (A, B, C, D)
- Age (C)
- Link quality (B)
- Search impact (D)
- Longevity (A)

Success Measures

- Quality measures
  - *web coverage*: percentage of the Web discovered or downloaded by the crawler
  - *repository quality*: percentage of useful pages in the repository
  - *repository freshness*: outdatedness of the local copies of pages relative to the pages' original copies on the Web

- Performance measures
  - *download rate*: number of bytes downloaded per unit of time

Challenges

- Dynamic nature of the Web
  - web growth
  - content change

- Malicious intent
  - hostile sites (e.g., spider traps, delay attacks)
  - spam sites (e.g., link farms, cloaking)

- Web site properties
  - unstable sites (e.g., variable host performance, unreliable networks)
  - sites with restricted content (e.g., robot exclusion),
  - soft 404 pages

Implementation Issues

- URL normalization
- DNS caching
- Multi-threading
- Politeness
- Robot exclusion protocol
- Sitemap protocol
- Mirror sites
- Data structures

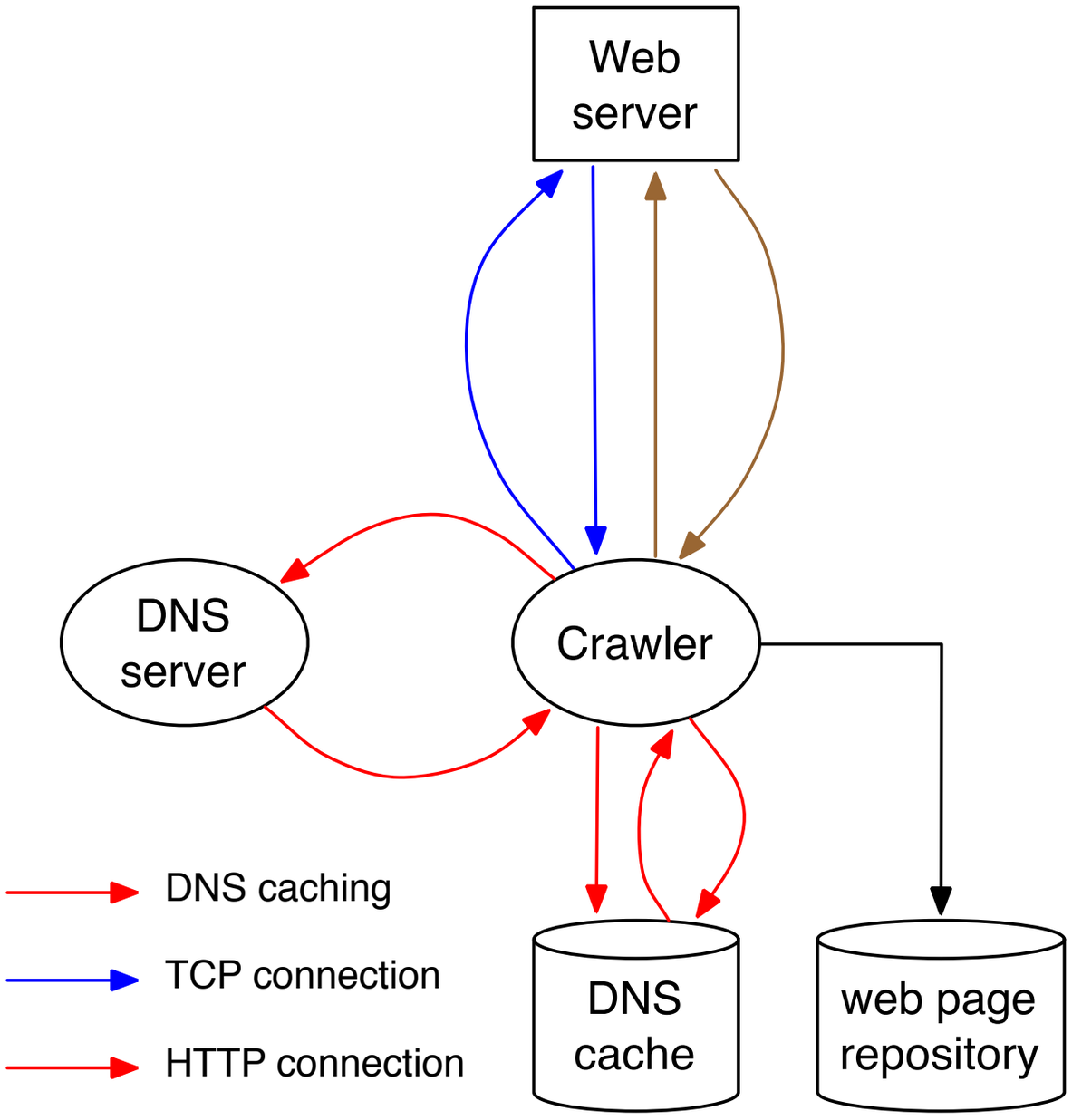
URL Normalization

- String processing operations applied to a URL in order to bring it to a standard form.
  - “http://www.CNN.com:80/x/../index.html” —> “cnn.com”
- Certain operations preserve the semantics of the URL
  - converting the protocol or host strings to lower case
  - removing/adding the default port (80)
- But, some may change it
  - converting the path string to lower case
  - removing/adding the leading “www” strings to the host string
  - removing/adding trailing slashes to the path string
  - converting relative paths to absolute paths

DNS Caching

- Before a web page is crawled, the host name needs to be resolved to an IP address.

- Since the same host name appears many times, DNS entries are locally cached by the crawler.



Multi-threading

Multi-threaded crawling

- crawling is a network-bound task
- crawlers employ multiple threads to crawl different web pages simultaneously, increasing their download rate significantly
- in practice, a single node can run around up to a hundred crawling threads
- multi-threading becomes infeasible when the number of threads is very large due to the overhead of context switching

Politeness

- Multi-threading leads to politeness issues.
- If not well-coordinated, the crawler may issue too many download requests at the same time, overloading
  - a web server
  - an entire sub-network
- A polite crawler
  - keeps at most one TCP-IP connection open for each web server
  - puts a delay between two consecutive downloads from the same server (the delay is usually in the 20–60 seconds range)

Robot Exclusion Protocol

- A standard from the early days of the Web.
- A text file (called robots.txt) provided by a web site to guide the web crawlers about which parts of the site should not be crawled.
- Crawlers often cache robots.txt files for efficiency purposes.

```
# robots.txt file

User-agent: googlebot          # all services
Disallow: /private/            # disallow this directory

User-agent: googlebot-news     # only the news service
Disallow: /                    # on everything

User-agent: *                  # all robots
Disallow: /something/          # on this directory

User-agent: *                  # all robots
Crawl-delay: 10                # wait at least 10 seconds

Disallow: /dir1/               # disallow this directory
Allow: /dir1/myfile.html       # allow a subdirectory

Host: www.example.com          # use this mirror
```

Sitemap Protocol

- An XML file (called sitemap.xml) provided by a web site to guide the web crawlers about which parts of the site should be crawled.

- A standard that complements the robot exclusion protocol.

```
<!-- sitemap.xml file -->

<?xml version="1.0" encoding="UTF-8"?>
<urlset xmlns= "http://www.sitemaps.org/schemas/
    sitemap/0.9">

<url>
<loc>http://www.test.com/</loc>
<lastmod>2015-04-03</lastmod>
<changefreq>weekly</changefreq>
<priority>0.65</priority>
</url>

<url>
...
<url>

</urlset>
```

Mirror Sites

- A mirror site is a replica of an existing site, used to reduce the network traffic or improve the availability of the original site.

- Mirror sites lead to redundant crawling and, in turn, reduced discovery rate and coverage for the crawler.

- Mirror sites can be detected by analyzing
  - URL similarity
  - link structure
  - content similarity

Data Structures

- Good implementation of data structures is crucial for the efficiency of a web crawler.
- The most critical data structure is the “seen URL” table
  - stores all URLs discovered so far and continuously grows as new URLs are discovered
  - consulted before each URL is added to the discovery queue
  - has high space requirements (mostly stored on the disk)
    - URLs are stored as MD5 hashes
    - frequent/recent URLs are cached in memory

Crawling Architectures

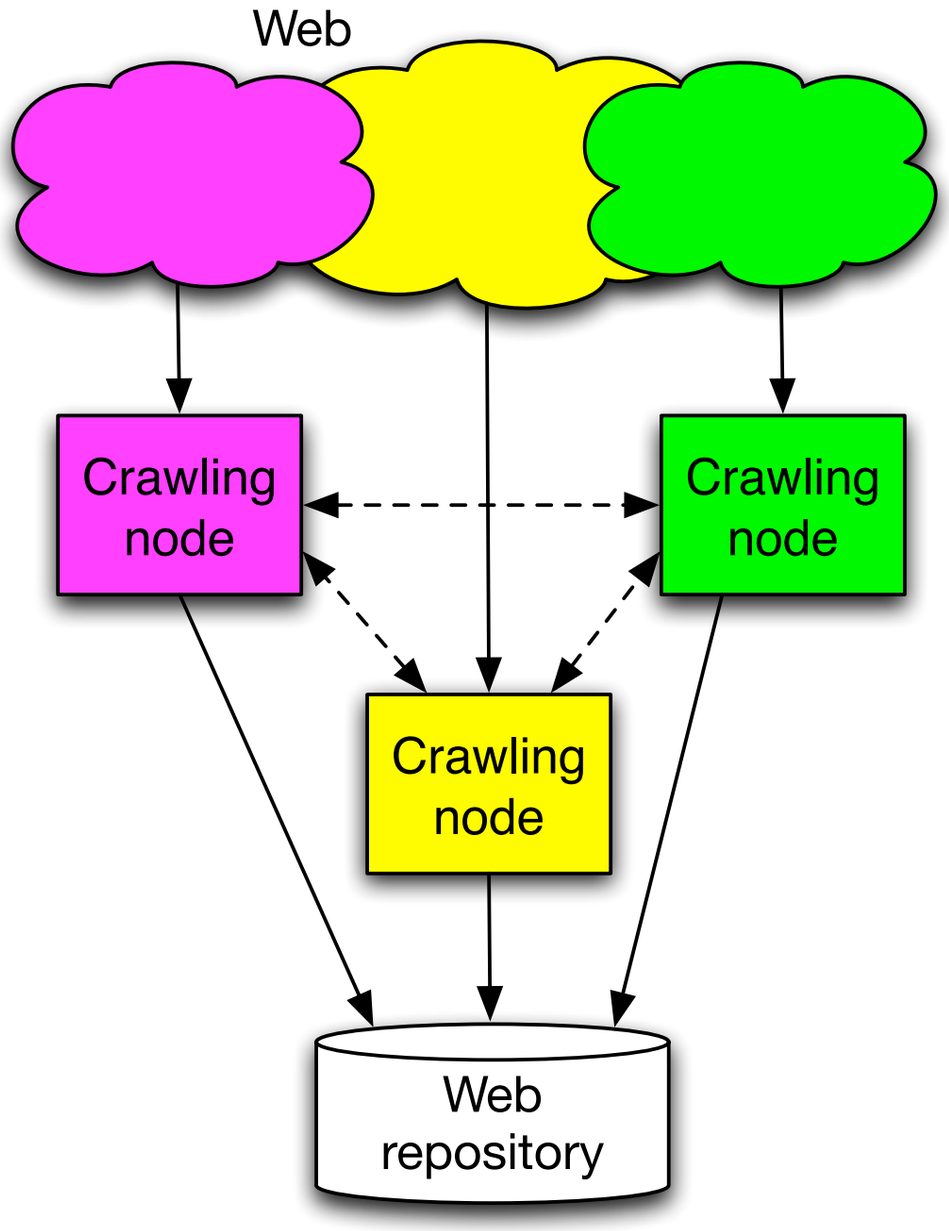
- Single node
  - CPU, RAM, and disk becomes a bottleneck
  - not scalable

- Multiple nodes
  - parallel crawler in a single data center
  - scalable

- Geographically distributed
  - parallel crawlers in multiple data centers
  - scalable
  - reduces the network latency

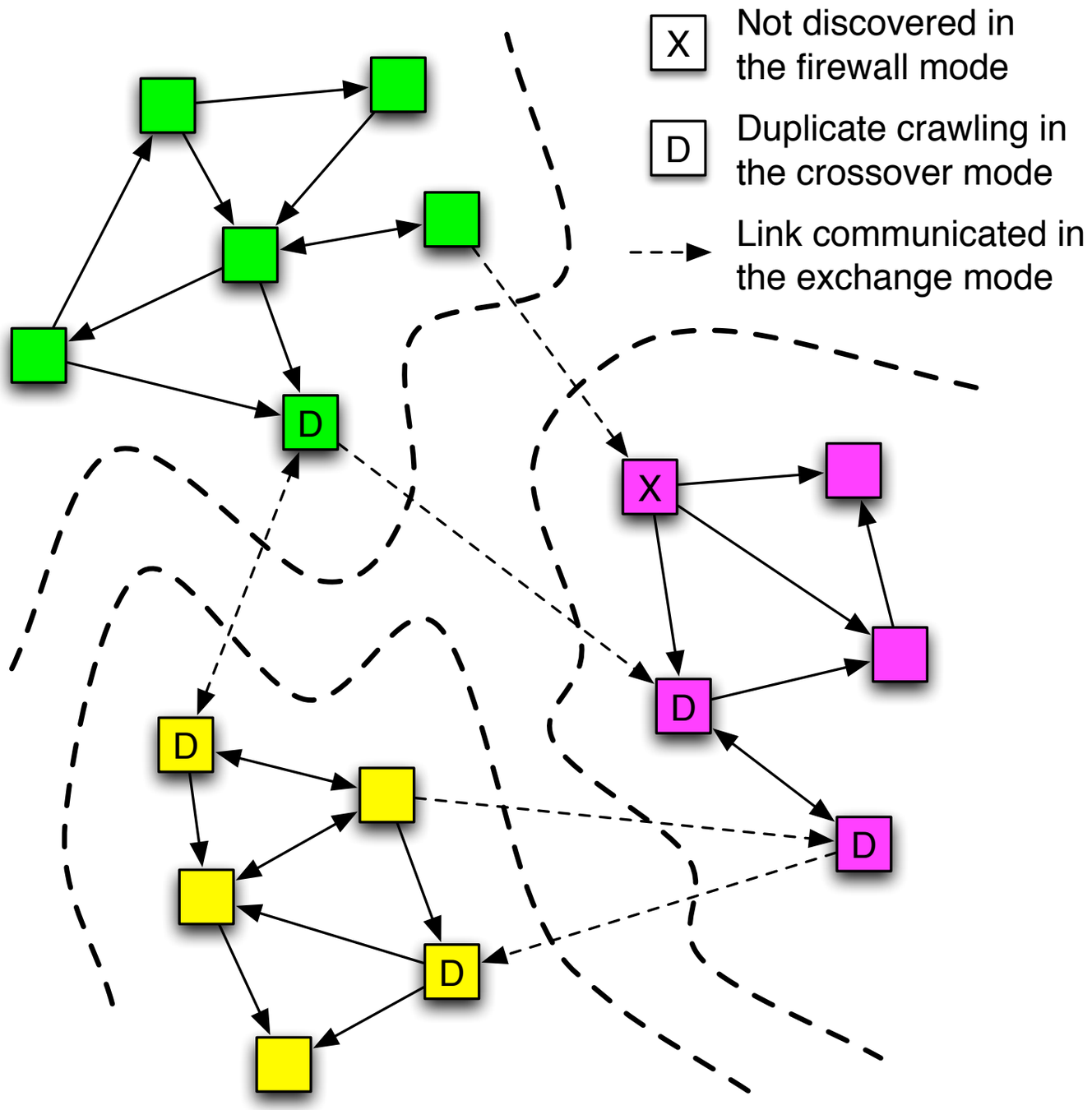
Parallel Web Crawling

- Uncoordinated crawling leads to duplicate pages in the repository.
- Web partitioning
  - based on the MD5 hashes of
    - URLs
    - host names
  - host-based partitioning is preferred since URL-based partitioning leads to politeness issues if the crawling decisions are not coordinated

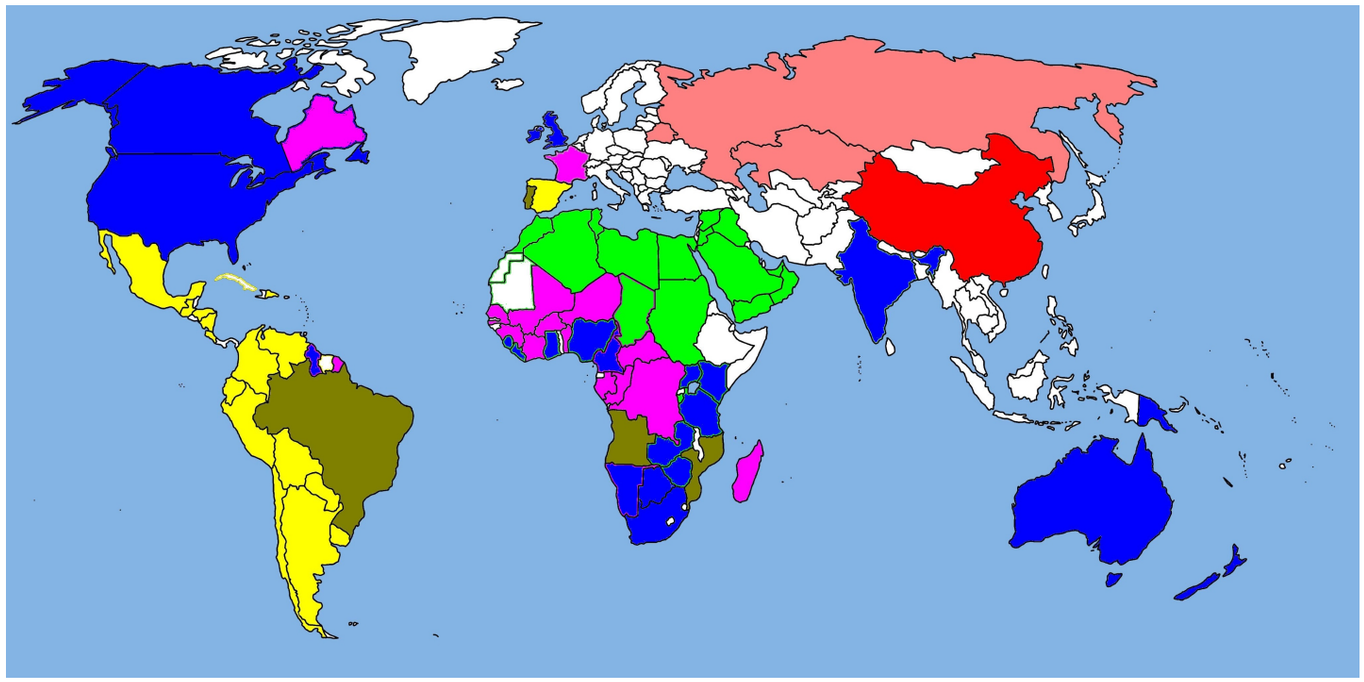


Coordination Between Nodes

- Firewall mode
  - lower coverage
- Crossover mode
  - duplicate pages
- Exchange mode
  - communication overhead

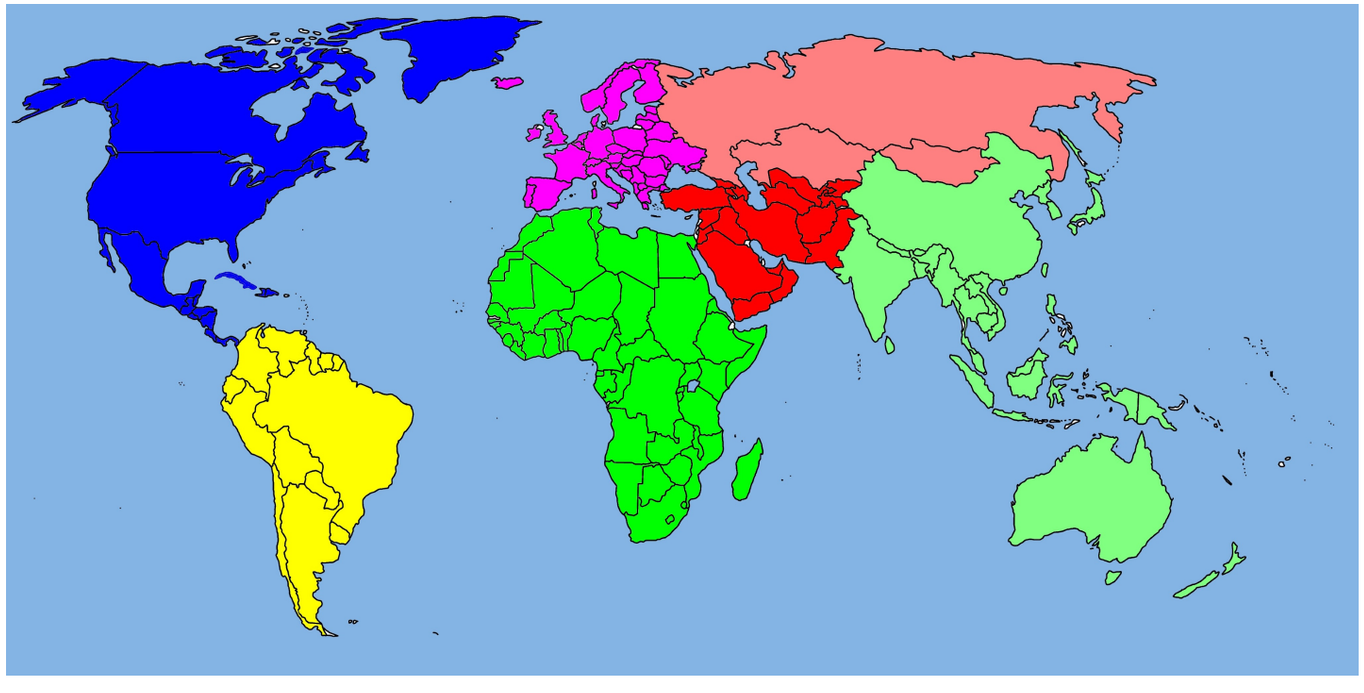


Geographically Distributed Web Crawling



- Language-based partitioning

- Region-based partitioning

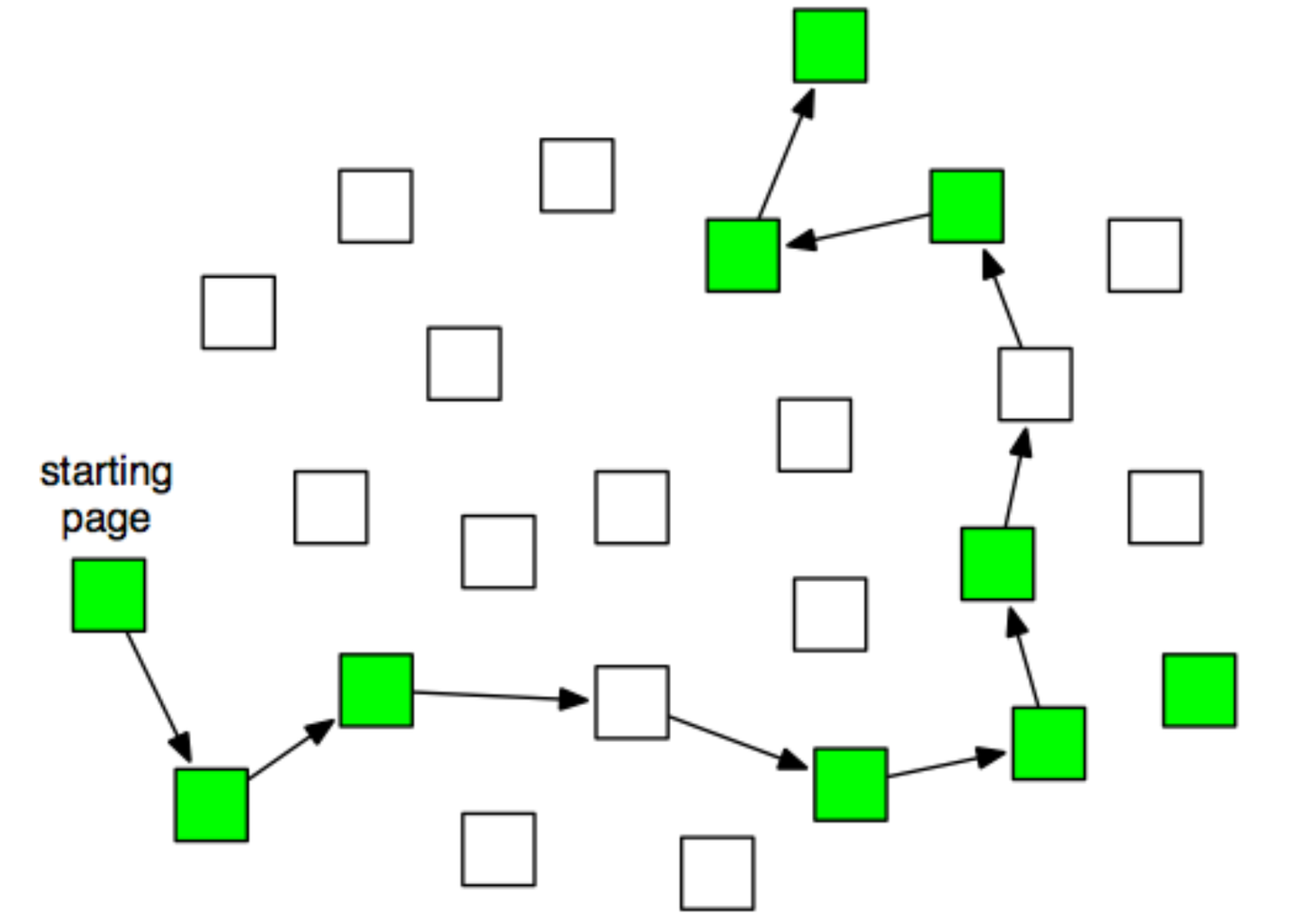


Focused Web Crawling

- The goal is to locate and download a large portion of web pages that match a given target theme as early as possible.

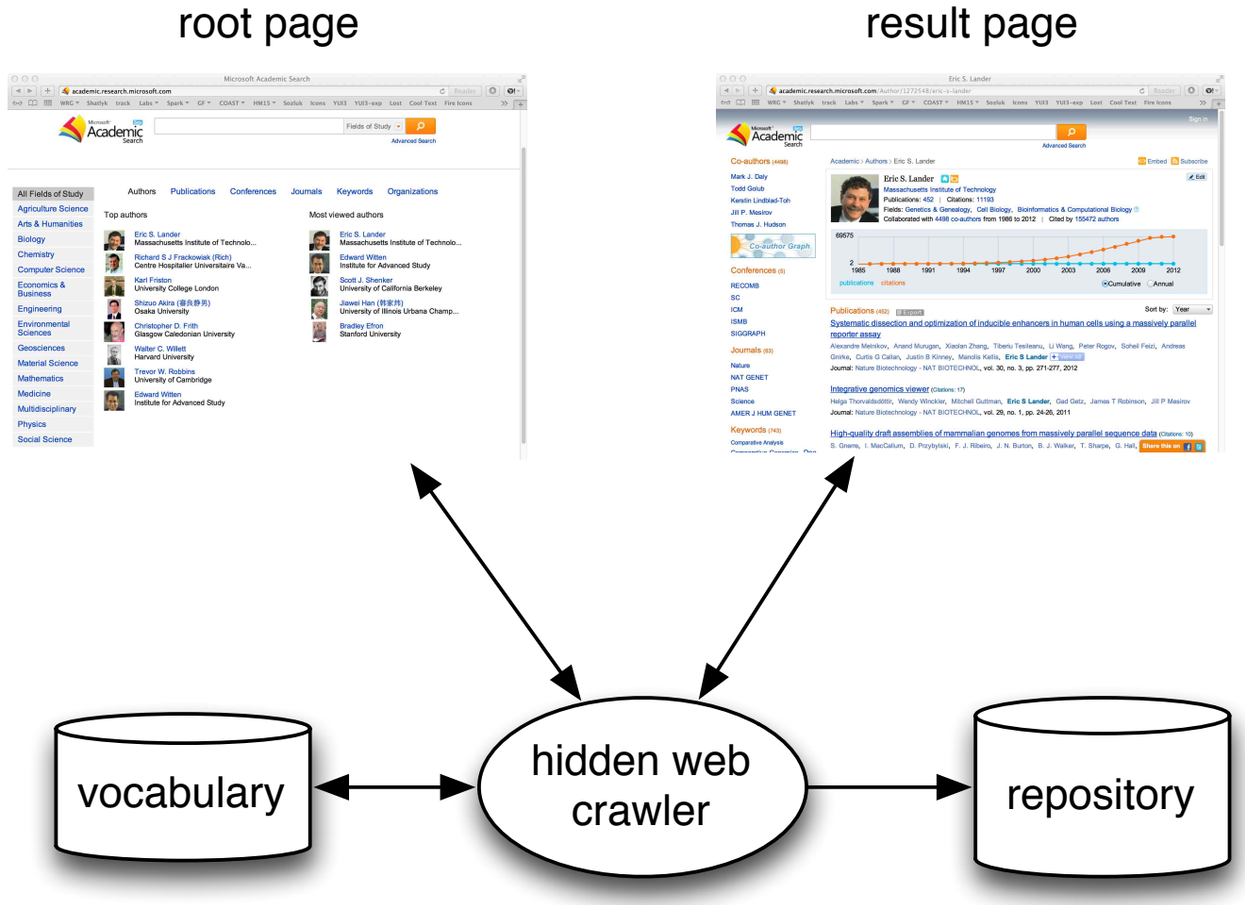
- Example themes
  - topic (nuclear energy)
  - genre (music)
  - type (forums)
  - demographics (kids)

- Features
  - URL patterns
  - referring page content
  - local graph structure

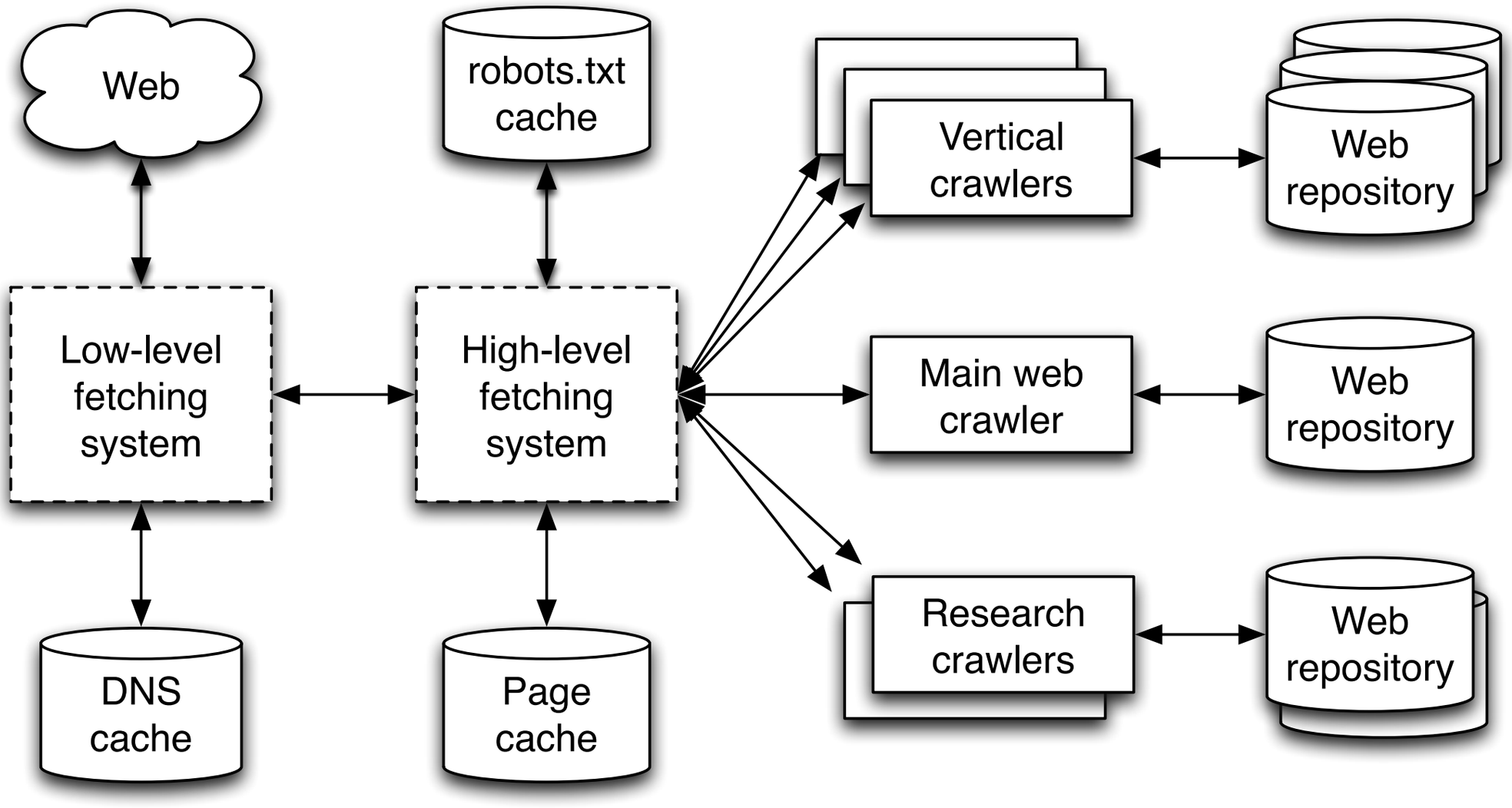


Hidden Web Crawling

- Hidden Web: web pages that a crawler cannot access by simply following the link structure.
- Examples
  - unlinked pages
  - private sites
  - scripted content
  - dynamic content



Common Fetching Infrastructure

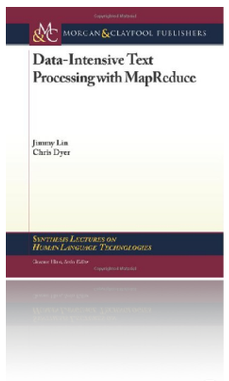


Open Source Web Crawlers

- BUbiNG: Distributed crawler (GNU GPLv3+)
- GRUB: Distributed crawler (GNU GPLv2)
- Heritrix: Internet Archive's crawler (Apache license)
- Norconex HTTP Collector: Multi-threaded crawler (Apache license)
- **Nutch**: Distributed crawler with Hadoop support (Apache License 2.0)
- PHP-Crawler: Script-based crawler (BSD license)
- Scrapy: Crawling framework (BSD license)
- Wget: Computer program to retrieve pages (GNU GPLv3+)

Pagerank

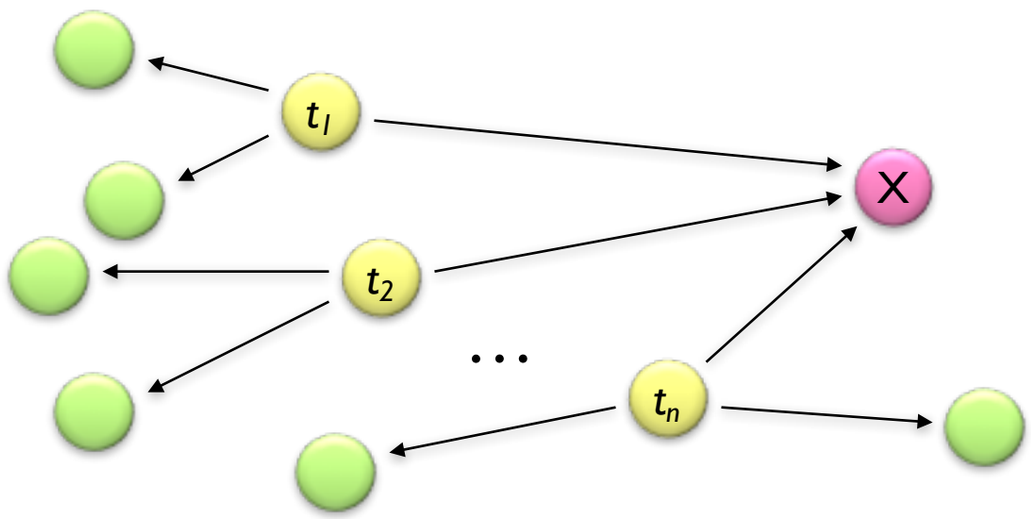
- **Random Surfers**
  - User starts at a random Web page
  - User randomly clicks on links, surfing from page to page
- **Pagerank**
  - Characterizes the amount of time spent on any given page
  - Mathematically, a probability distribution over pages
- **Web Ranking**
  - One of thousands of features used in web search

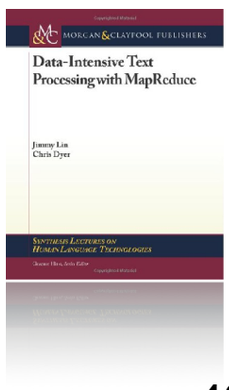


Definition

- Given page *x* with inlinks t_1, ..., t_n, where
  - *C(t)* is the out-degree of link *t*
  - *α* is probability of random jump
  - *N* is the total number of nodes in the graph

$$PR(x) = \alpha \left(\frac{1}{N}\right) + (1-\alpha) \sum_{i=1}^{n} \frac{PR(t_i)}{C(t_i)}$$

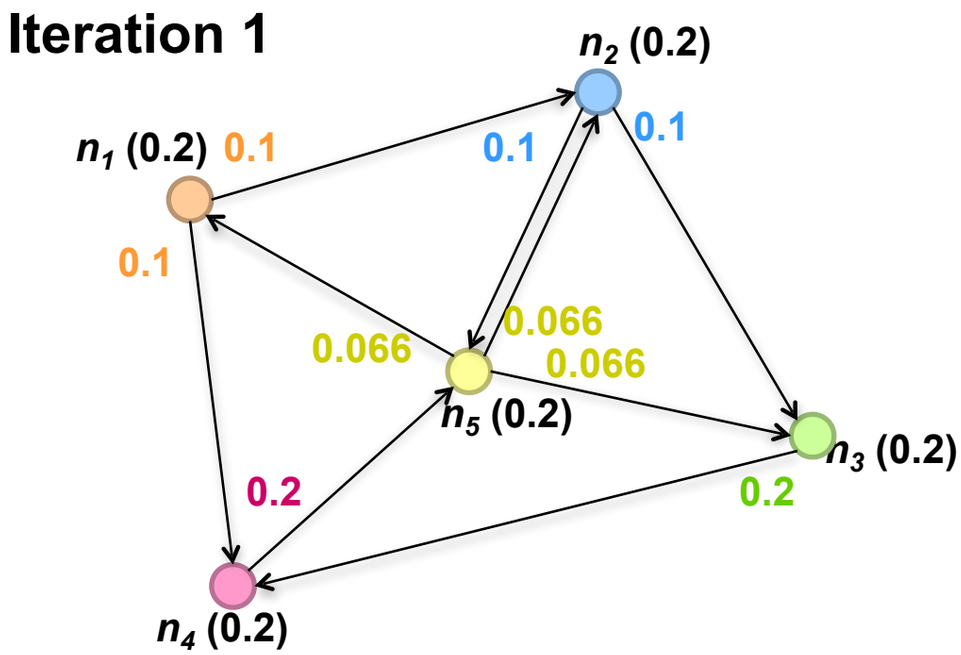


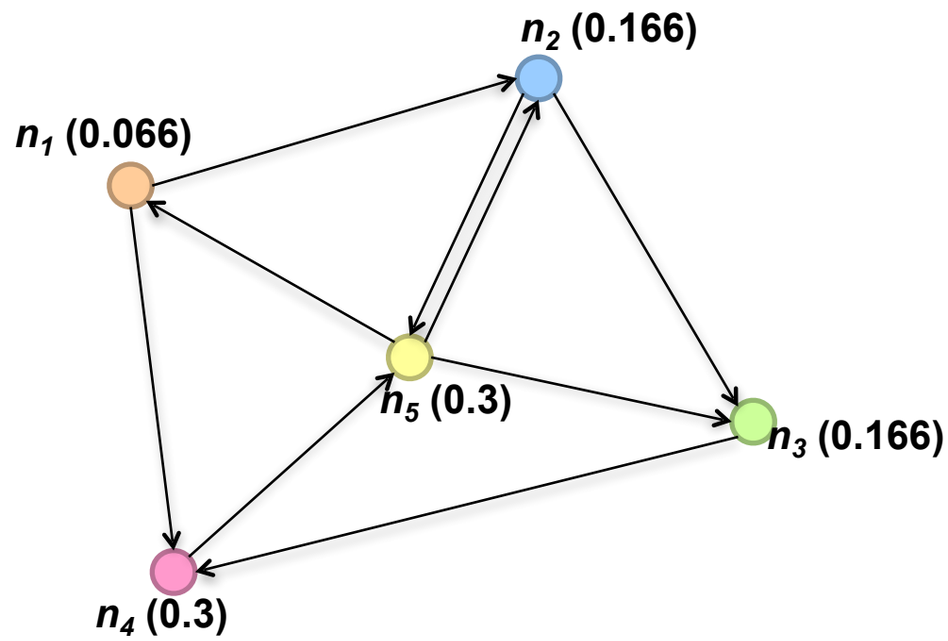


Algorithm Sketch

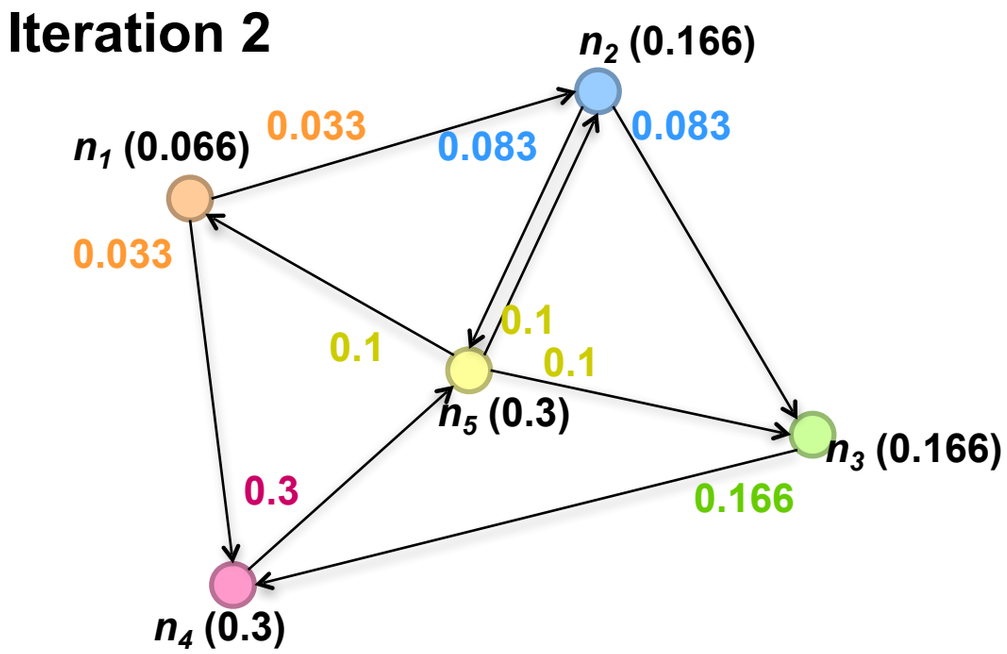
- Start with seed PR_i values
- Each page distributes PR_i mass to all pages it links to
- Each target page adds up mass from in-bound links to compute $PR_i + 1$
- Iterate until values converge

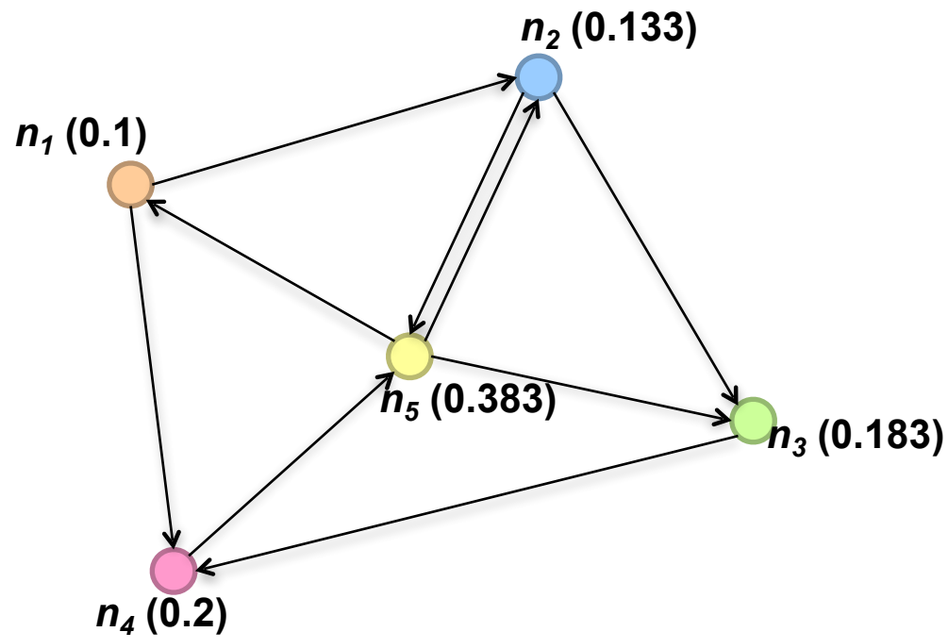
Simplified Algorithm Example (I)



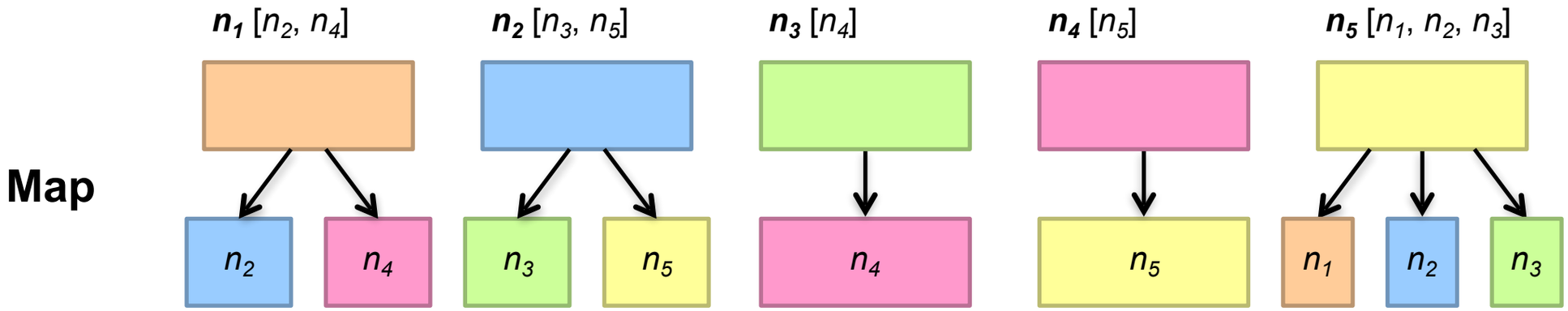


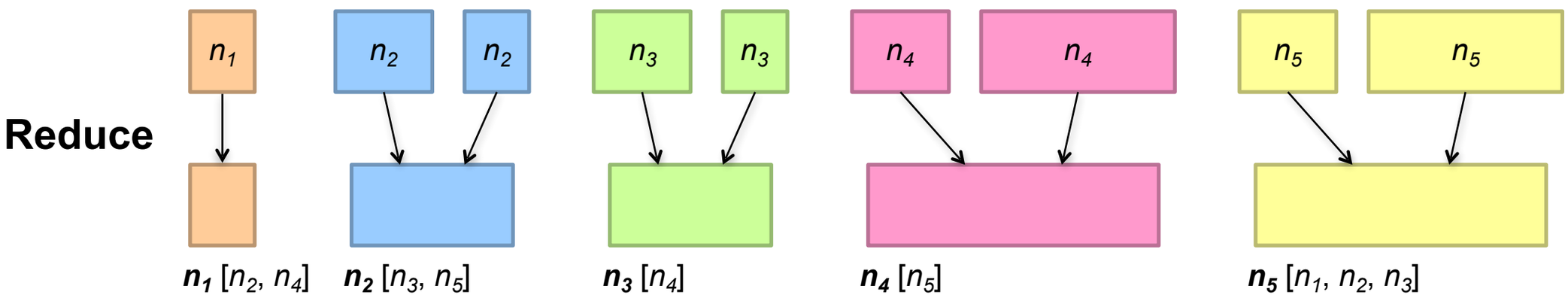
Simplified Algorithm Example (II)





Pagerank in MapReduce (I)





Pagerank in MapReduce (II)

```
1: class Mapper
2:     method Map(nid n, node N)
3:         p ← N.PageRank/|N.AdjacencyList|
4:         Emit(nid n, N)                          ▷ Pass along graph structure
5:         for all nodeid m ∈ N.AdjacencyList do
6:             Emit(nid m, p)                      ▷ Pass PageRank mass to neighbors

1: class Reducer
2:     method Reduce(nid m, [p1, p2, ...])
3:         M ← ∅
4:         for all p ∈ counts [p1, p2, ...] do
5:             if IsNode(p) then
6:                 M ← p                           ▷ Recover graph structure
7:             else
8:                 s ← s + p                       ▷ Sums incoming PageRank contributions
9:         M.PageRank ← s
10:        Emit(nid m, node M)
```