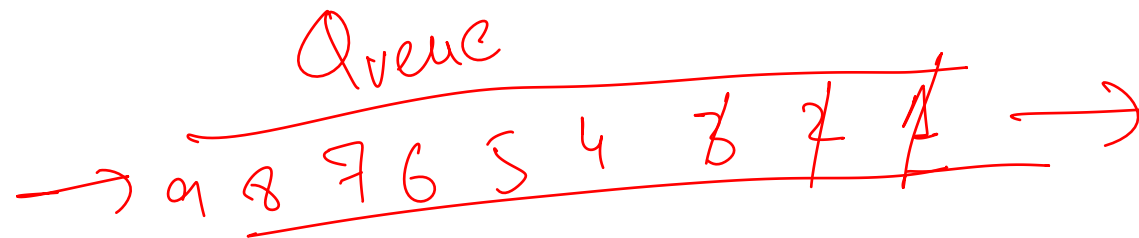


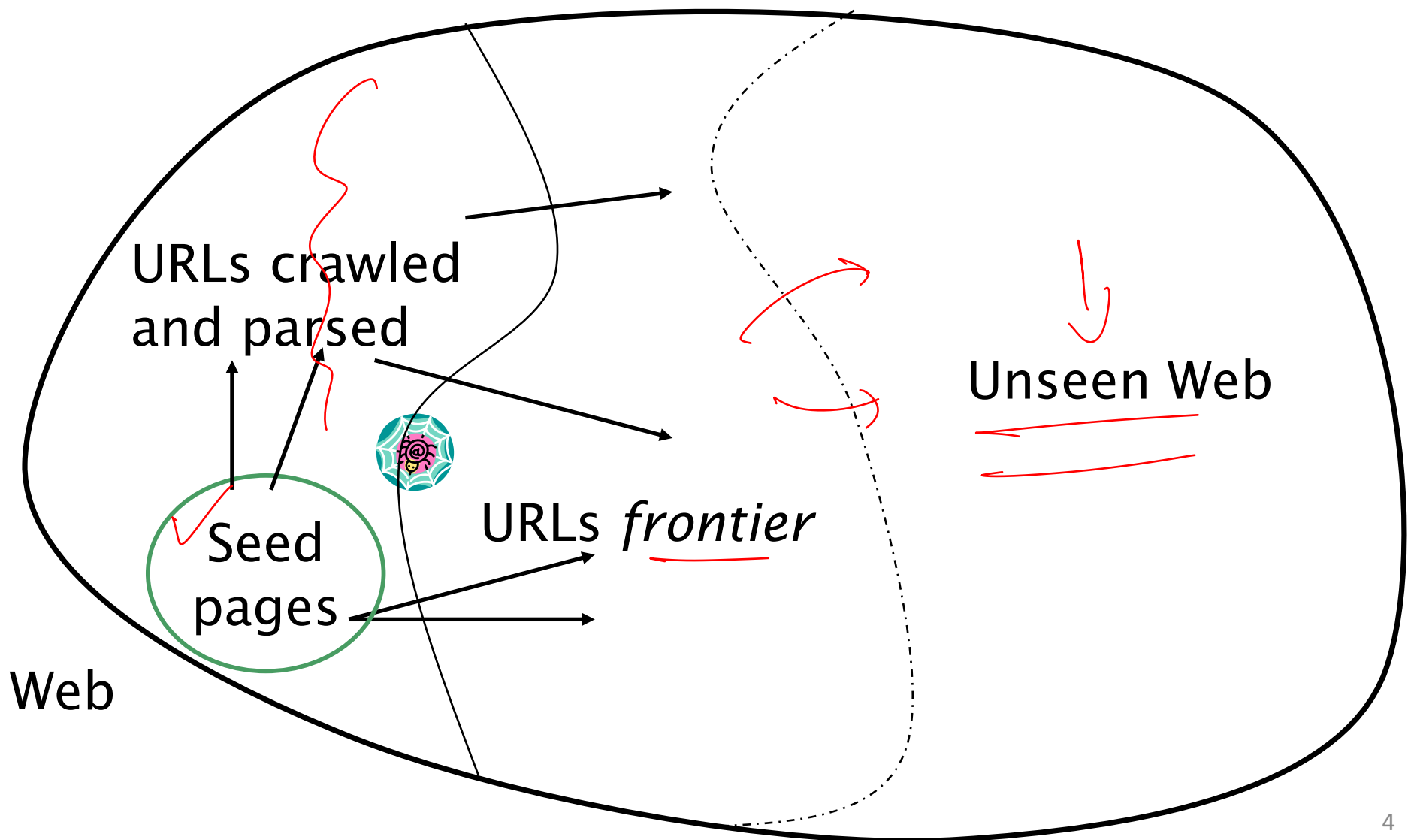
# Information Retrieval

# Basic crawler operation

- Begin with known “seed” URLs
- Fetch and parse them
  - Extract URLs they point to
  - Place the extracted URLs on a queue
- Fetch each URL on the queue and repeat

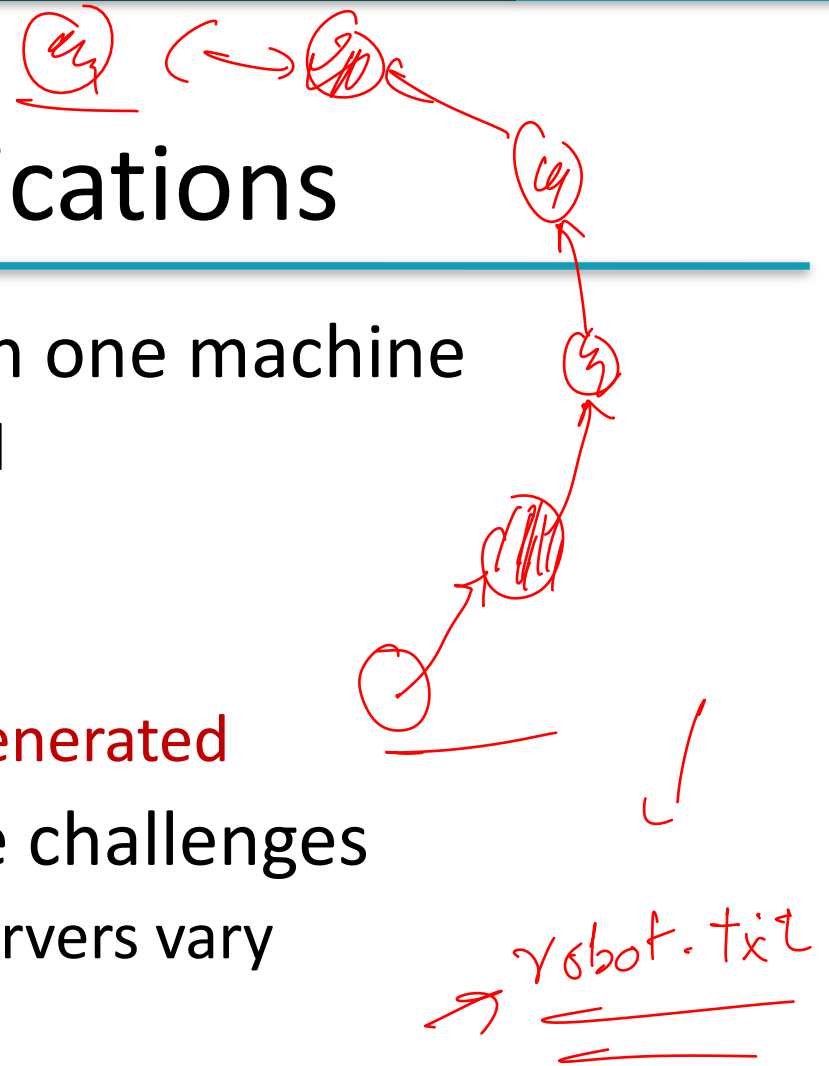


# Crawling picture




# Simple picture – complications

- Web crawling isn't feasible with one machine
  - All of the above steps distributed
- Malicious pages
  - Spam pages
  - Spider traps – incl dynamically generated
- Even non-malicious pages pose challenges
  - Latency/bandwidth to remote servers vary
  - Webmasters' stipulations
    - How “deep” should you crawl a site's URL hierarchy?
  - Site mirrors and duplicate pages
- Politeness – don't hit a server too often



# What any crawler *must* do

---

- Be Robust: Be immune to spider traps and other malicious behavior from web servers
- 
- Be Polite: Respect implicit and explicit politeness considerations

# Explicit and implicit politeness

---

- Explicit politeness: specifications from webmasters on what portions of site can be crawled
  - robots.txt
- Implicit politeness: even with no specification, avoid hitting any site too often

# Robots.txt

---

- Protocol for giving spiders (“robots”) limited access to a website, originally from 1994
  - [www.robotstxt.org/robotstxt.html](http://www.robotstxt.org/robotstxt.html)
- Website announces its request on what can(not) be crawled
  - For a server, create a file `/robots.txt`
  - This file specifies access restrictions

# Robots.txt example

---

- No robot should visit any URL starting with `"/yoursite/temp/"`, except the robot called `"searchengine"`:

```
User-agent: *
```

```
Disallow: /yoursite/temp/
```

```
User-agent: searchengine
```

```
Disallow:
```



# What any crawler *should* do

---

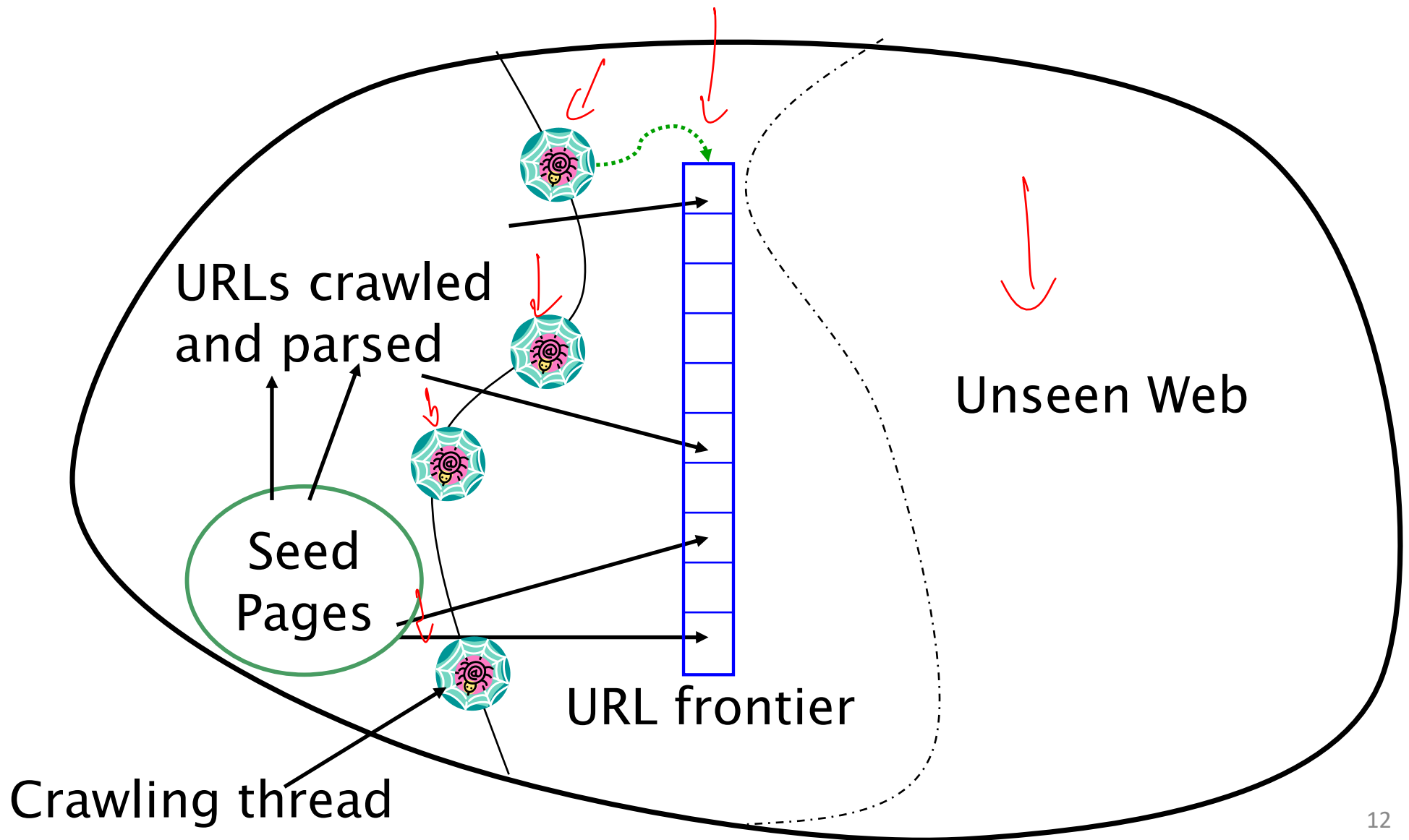
- Be capable of distributed operation: designed to run on multiple distributed machines
- Be scalable: designed to increase the crawl rate by adding more machines
- Performance/efficiency: permit full use of available processing and network resources

# What any crawler *should* do

---

- Fetch pages of “higher quality” first
- Continuous operation: Continue fetching fresh copies of a previously fetched page
- Extensible: Adapt to new data formats, protocols

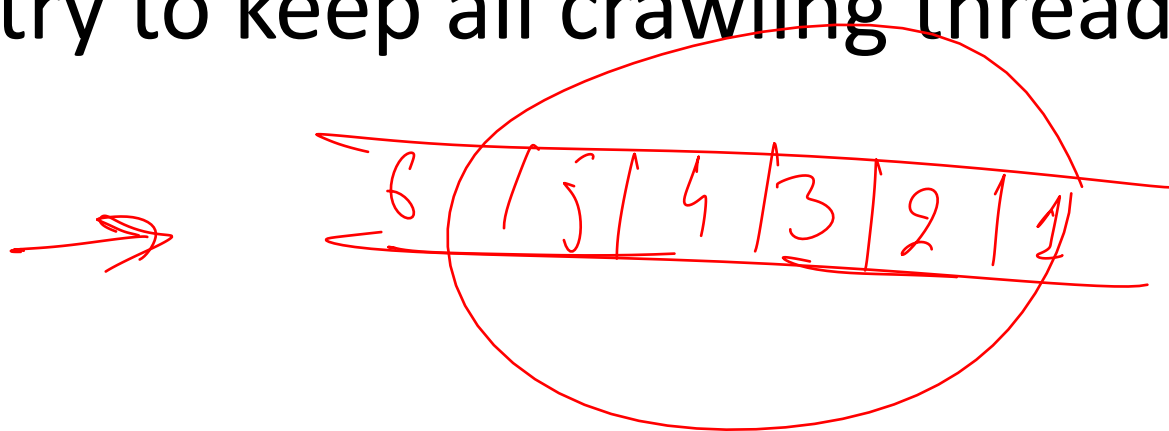
# Updated crawling picture



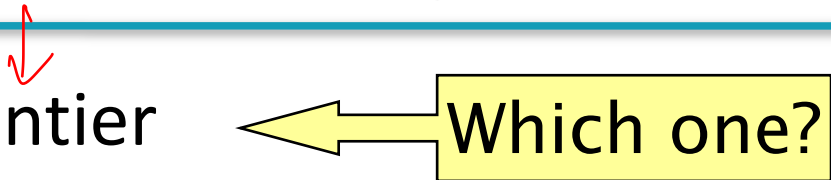




Politeness

## → URL frontier

- Can include multiple pages from the same host
- Must avoid trying to fetch them all at the same time
- Must try to keep all crawling threads busy



# Processing steps in crawling

- Pick a URL from the frontier 
- Fetch the document at the URL 
- Parse the URL 
  - Extract links from it to other docs (URLs) 
- Check if URL has content already seen 
  - If not, add to indexes
- For each extracted URL
  - Ensure it passes certain URL filter tests
  - Check if it is already in the frontier (duplicate URL elimination)

E.g., only crawl .edu,  
obey robots.txt, etc.

