Step 1: Requirements

Functional requirements:
- Recursively fetches links from a set of starting pages
- Scalable such that it can crawl entire web
- Used to fetch hundreds of millions of web document
- Modular way with expectation that new functionality will be added e.g., new document types

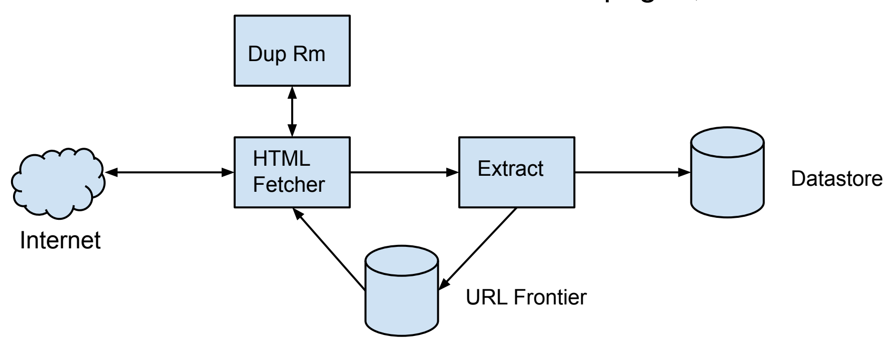
Design considerations:
- Assume only HTTP for now
- Expected number of pages to crawl is 15B

Step 2: Back of Envelope Calculations

Storage
- 15B/(4w*7d*86400s)=6200pages/s
- 15B*(100KB+500)=1.5PB
- 1.5PB/0.7=2.14PB (don't want to go above 70% full)

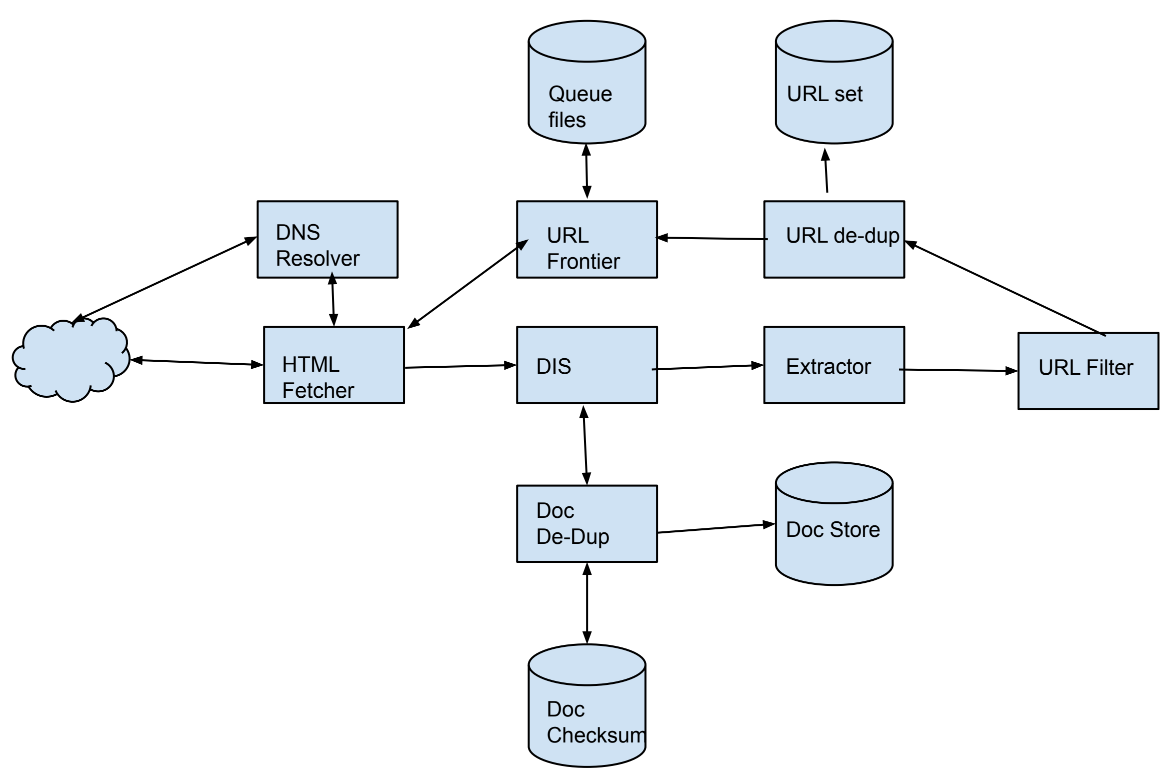
Step 3: High-Level Design

- Basic algorithm:
    - Pick URL from unvisited URL list
    - Determine IP address of host
    - Establish connection and download doc
    - Parse doc to look for new URLs
    - Add new URLs to list of unvisited URLs
    - Process downloaded doc e.g., store it, index it, etc
    - Repeat
- Usually use BFS
- Path-ascending crawling can find non-linked paths
- What makes this difficult problem:
    - Large volume of web pages
    - Rate of change on web pages
- High level design needs:
    - URL frontier to store list of URLs to download; prioritize which URLs tocrawl first
    - HTML fetcher to retrieve web page from server
    - Extractor to extract links from HTML documents
    - Duplicate eliminator to make sure same content is not extracted twice
    - Datastore to store retrieved paged, URLs and other metadata



Step 6: Detailed Design

- Assume crawler running on one server, and crawling done by multiple threads:
    - Remove absolute URL from shared URL frontier
    - For now, we just support HTTP, but should be designed modularly so future protocols can be supported
    - Thread downloads document and places it into a Document Input Stream (DIS)
    - This enables other modules to re-read the document multiple times
    - Worker invokes dedupe test to see if doc has been seen before
        - If so, doc is not processed and thread moves to next URL from URL frontier
- Next, crawler needs to process downloaded doc
    - Each doc can have different MIME types e.g., HTML, image, video
    - Implement MIME type modularly so we can move to other types later
    - HTML processing module extract all links from page
        - Each link converted to URL and tested against user-supplied URL filter to see if it should be downlooaded
        - If passes filter, worker sees if URL has already been seen
        - If not then add to URL frontier



- URL frontier
    - Distribute into multiple servers
    - Hash function maps each URL to a server that will be responsible for crawling it
    - Hash function to map each hostname to a thread number
    - Each thread has a FIFO queue. This prevents overloading target server by downloading too many pages at once.
- HTML fetcher
    - Download doc corresponding to a given URL
- Document input stream (DIS)
    - Cache document so it can be accessed by multiple servers
    - Cache small documents entirely in memory, larger docs written temporarily to disk
- Document dedupe test
    - Calculate a 64-bit checksum on every processed document and store in database
    - MD5 or SHA
- URL filters
    - Control set of URLs that are doownloaded
- Domain name resolution
    - Web crawler must use DNS to map webserver hostname into an IP address
- URL dedupe test
    - Dedupe links extracted from a URL document