# Design TinyUrl

## Brute Force

- Hash the url
- Pros
    - Simple
    - No need to save into a DB

- Cons
    - Collisions

## Better Solution

- Save the URLs in a DB
- Hash the URL with its id in the DB

# How Google Search works?

## Links

- https://www.google.com/search/howsearchworks/crawling-indexing/
- https://softwareengineering.stackexchange.com/questions/38324/how-would-you-implement-google-search

## Crawling Infrastructure

- Webcrawler job is to get list of webpages and dump them into a set
- Most important part is to not get stuck in an infinite loop
- **Distrubuted Web Crawling** - Web crawlers will each run on different machines with different domains
- Web Crawlers will save data into a store which will then be used to create an index
- Also have some sort of logic to prevent infinite loops

## Indexing Infrascructure

- Indexer will run as a MapReduce Job

- Map will create an inverted index on each individual machine
- Reduce will combine all inverted indices into a single one

### Serving Results

- Cache the results if you can, but for something really big, this may not be feasible, since 10% of queries have not been seen before
- Use a NoSQL database for quick reads, ie MongDB

# How Facebook Newsfeed Works

### Links

- https://code.facebook.com/posts/1737605303120405/dragon-a-distributed-graph-query-engine/
- https://neo4j.com/news/how-facebook-matured-its-data-structure-and-stepped-into-the-graph-world/

### Ranking Algorithm

- Eventual Consistency
- Should be highly available
- AP System
- TAO distributed data store

# How Dropbox Works

- Namespace
- Need something that is Consistent and fault tolerant.

# How Twitter Works

### Links

- https://blog.twitter.com/engineering/en_us/a/2014/manhattan-our-real-time-multi-tenant-distributed-database-for-twitter-scale.html

### Design Goals

- Reliability - Want it to work in any conditions

- Availability - Prefer availability over consistency
- Low latency

### Twitter

## How to find K most frequent words in a huge file

- Typical MapReduce
- Use a Trie to store each word
- Count up all the words on each Machine
- In the reduce step you get the highest ones