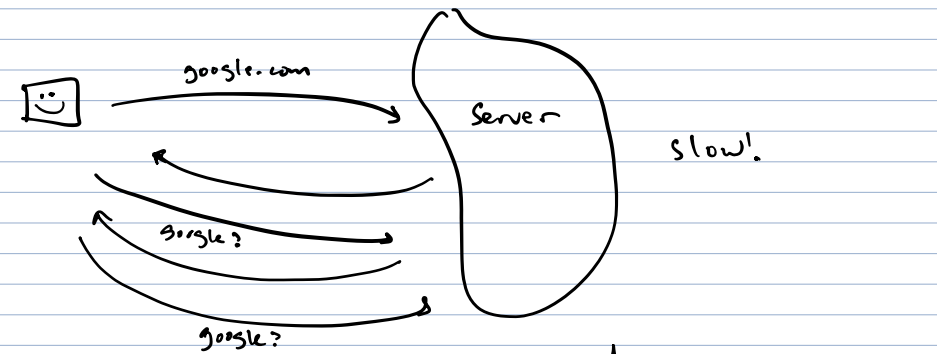
Lecture 1: Modern Hashing.

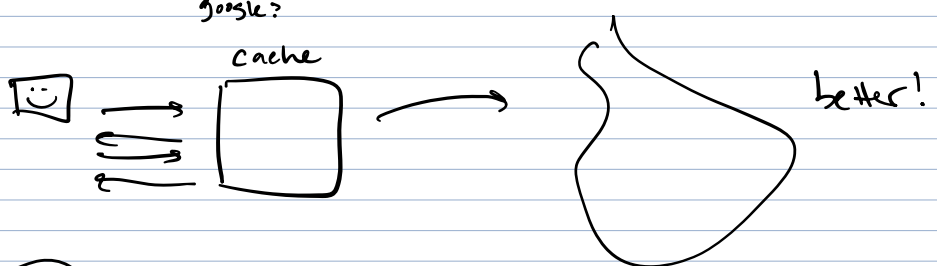
Syllabus recap:

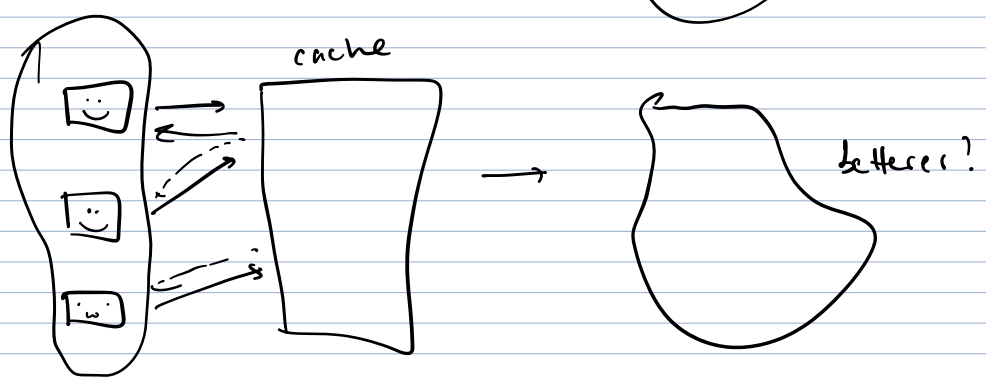
- Preregs: CSE 311/312
  (can msg me about add codes but they're pretty strict).
- One topic (related) per week.
- One HW per topic
  - 100% of grade!:
  - collaboration policy
  - no late days but lowest dropped
  - GPT allowed but maybe not so useful.
  - Submit written on Gradescope, code on canvas.
- OH / TAs

Consistent Hashing (Karger, Lehman, Leighton, Levine, Lewin, Panigrahy).

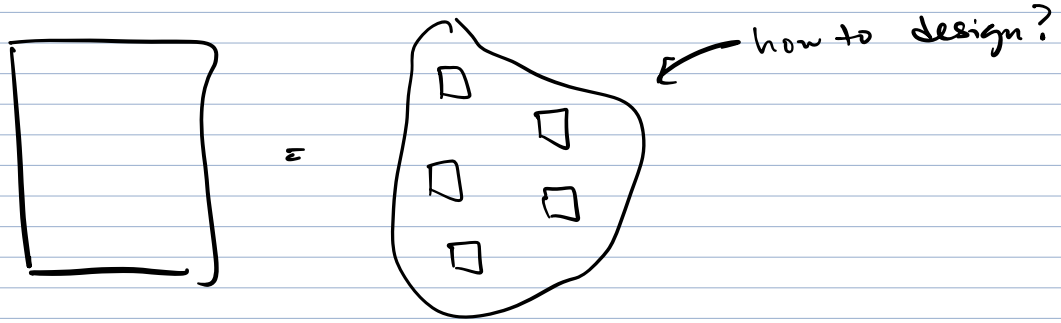
Q: How to design cache for internet?







Problem: This cache usually will not fit on 1 machine. Must distribute across m machines



Idea 1: Brute force search. → too slow!

Idea 2: Hashing

Recall: $h: \Sigma^* \to [m]$ (e.g. $h(x) = SHA(x) \bmod m$).

$x_1, \dots, x_n$ a set of URLs.

assign x_i to $h(x_i)$.

If h is "pseudorandom" then # i assigned to any bucket $\approx \frac{n}{m}$.

To lookup x in cache: check machine $h(x)$.

done...?

Problem: what if a machine fails / is added?

$SHA(x) \bmod 100 \neq SHA(x) \bmod 99 \neq SHA(x) \bmod 101$ (usually).

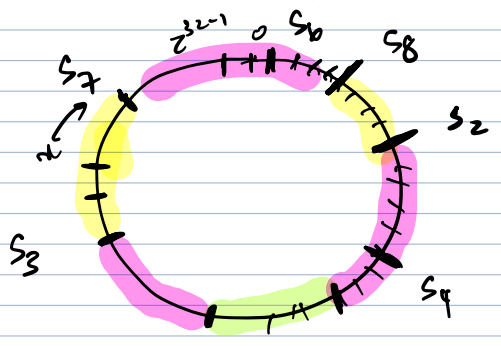
So we'd need to redistribute (almost) all the keys.
(all but a $\sim \frac{1}{m}$ fraction) BAD!

Consistent Hashing: Take $h: \Sigma^* \to Z^{32}-1$ (some large number),

and treat this as indexing a position on a circle.

hash each server $s_1, \dots, s_m$

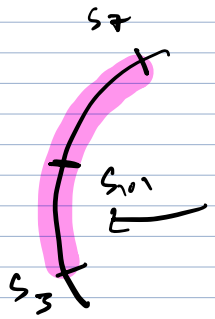
For data x, store it on first server clockwise of $h(x)$



s_1 s_5

If h is "uniform-like", then # of x in [redacted] $\approx \frac{n}{m}$.

If server is added, what needs to be re-ordered?



$\sim \frac{1}{m}$ fraction of points!

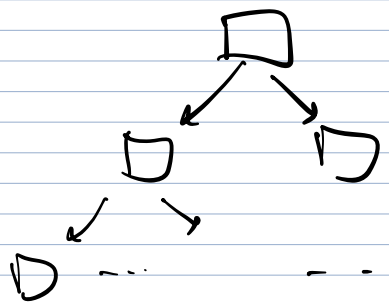
If server is removed, similar.

How to perform lookup?

given x with hash $h(x)$, find s with smallest $h(s)$ satisfying $h(s) \geq h(x)$ (ignoring the wraparound).

Naively: need to compute m hash functions (slow).

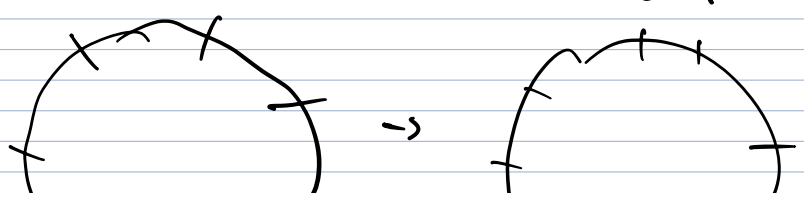
Ben: use a balanced search tree (BST).



$\log(m)$

(see also: rendezvous hashing).

Problem: variance. In expectation, each server will see $\sim \frac{1}{m}$ traffic. But this is not with high probability

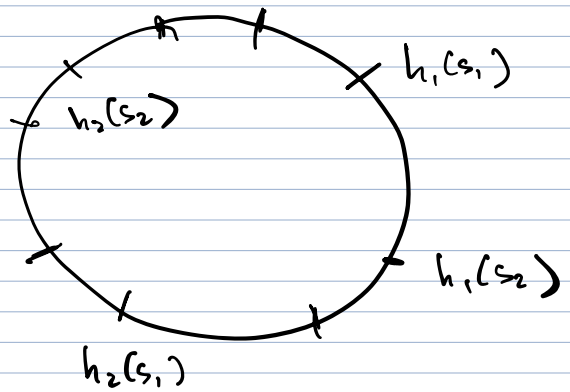


Solution: Variance reduction via virtual copies.

Idea: use k hash functions

$$h_1, \ldots, h_k : \Sigma^* \to \mathbb{Z}^{32} - 1.$$

Associate each server with k intervals, given by the positions of these k hash functions.



The lengths of the intervals "average out". Can show that if $k = \log_2(m)$, then all servers have load $O(\frac{n}{m})$ whp.

Also useful for heterogeneous servers: If one one server is bigger, add more virtual copies!