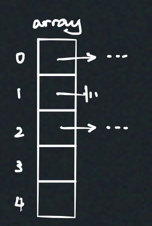
Lec 7 Hash Tables

Considerations

- Data structure
  - └ disk, memory
- Concurrency

Hash Table (mem)

Want $O(n)$ space, $O(1)$ access (average case) ($O(n)$ worse case)



▸ Hash Function — efficiency, pseudorandom, general

↙ cryptographic hash would be too slow

↑ works with many data types

Many funcs
- CRC-64
- Murmur Hash
- Google CityHash
- Facebook XX Hash ← good one

▸ Hash Scheme — organisation, retrieval alg, collision handling, etc.

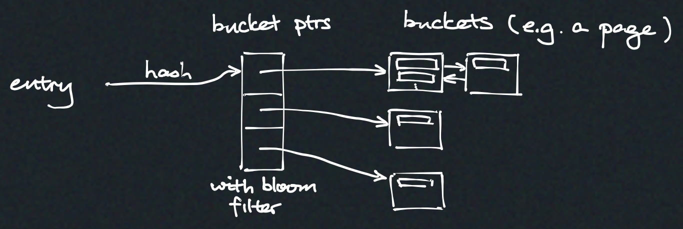
Static Schemes (when not gradually growing table size)
but doubling size of table allowed, just rarely

→ Open Addressing
- Linear Probing
  - └ when collision, find next free slot
  - └ when search, scan to find key starting from hashed index or hitting empty
  - └ when delete, either mark tombstone or shift subsequent entries
    - ↑ can just use bit map, only reorg table when too many tombstone
  - └ impled in Google absl
  - Good locality for cache!
  - → Generalisations: map to linked list, external var len values, etc.
    - or vector.. or linked blocks...
- Cuckoo Hashing
  - └ Use two indep hash funcs h_1, h_2, if one collide, use the other one
  - └ if both hashed to occupied position... try see if occupant can use another spot and kick it out
    - └ if this fails (circular kick out dependency), bump table size
      - aka cuckoo graph
  - └ theory: diminishing return beyond 2 hash funcs
    - "power of 2"
  - └ when search, hash both ways & look at each
  - $O(1)$ lookup, but not cache friendly

Load factor = num entries / max slots
└ in practice, keep in 80-90%

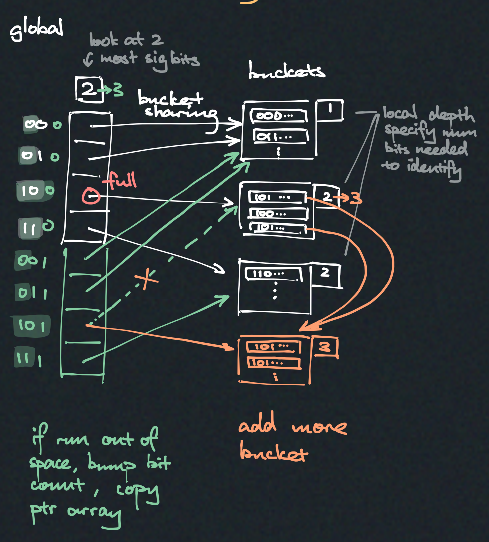
Optimisations
- Multi-table & use diff tables by key len
- Version numbering

Chained Hashing (mem & disk)



→ Bloom filter
└ compact, fast, no-false negative membership check
two hash funcs, set bit flag for each func
membership check by checking both bits from hashes are 1

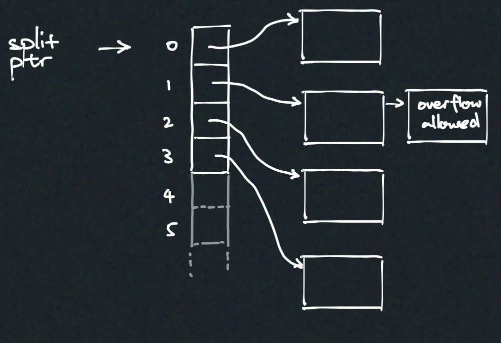
Extendible Hashing (disk)



Linear Hashing

Keep split ptr, allow overflow bucket
when overflow split at the split ptr,
grow table size by one, and add an
additional hash fn for new bucket
move split ptr down

Linear Hashing revisit



$h_0(k) = k \% n$

$h_1(k) = k \% (2n)$

for things before split ptr, use next hash func