Hash Tables

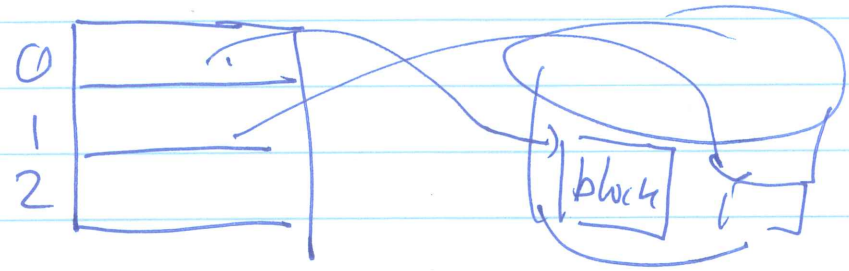
- **Hash Function:**

search key → ("hash key") → [Hash function] → bucket index [0 .. B-1]

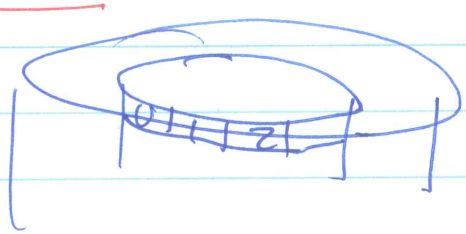
K = search key
h(K) = bucket index

- **Bucket Array:**

(1) array of block pointers in memory



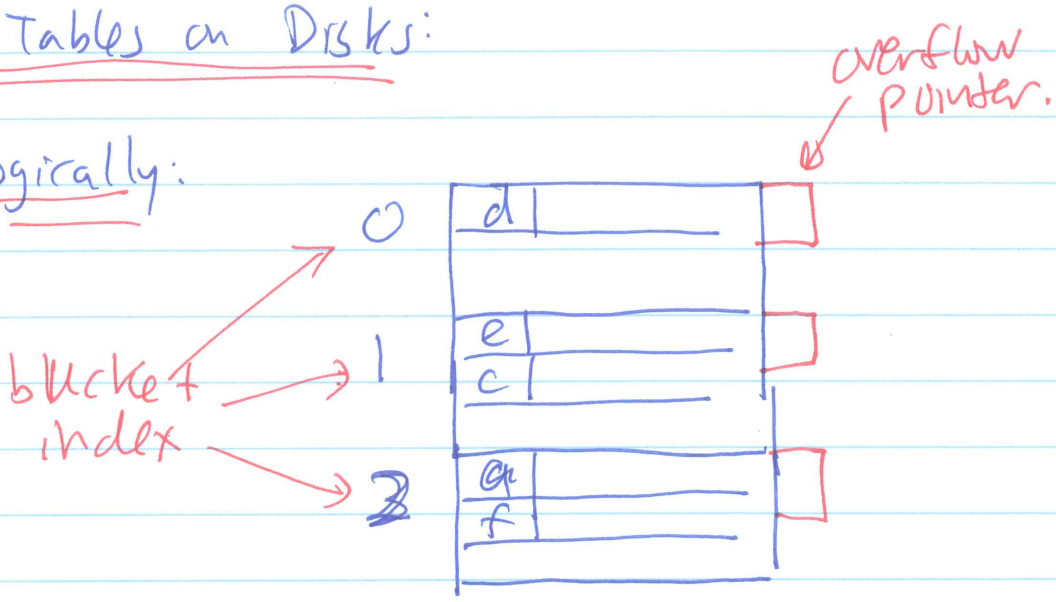
(2) Special (reserved) tracks/cylinders on disk

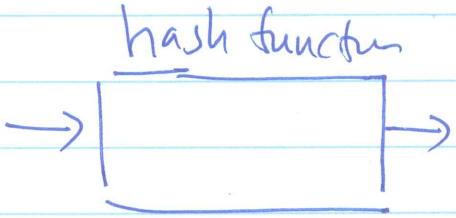


(each bucket index = 1 block pointer location)

Hash Tables on Disks:

Logically:



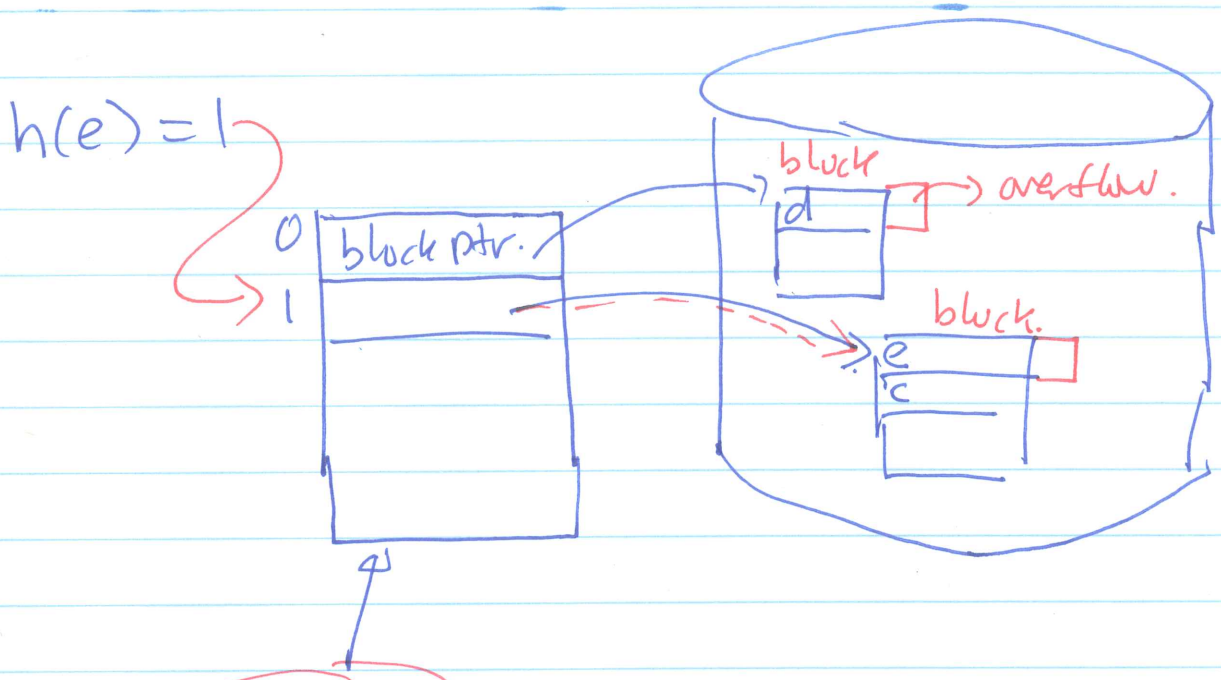


$h(d) = 0$ $\quad$ $h(a) = 2$

$h(e) = 1$ $\quad$ $h(f) = 2$

$h(c) = 1$

Physically:



can be:

(1) array of block pointers in memory

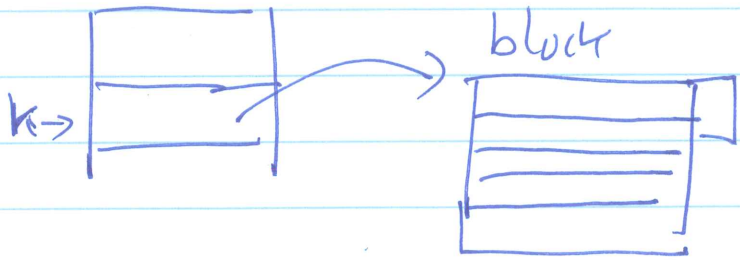
(2) special (reserved) cylinder on disk.

Traditional Hash Index

- Insert(K):

(1) compute $k = h(K)$

(2) Find hash bucket:

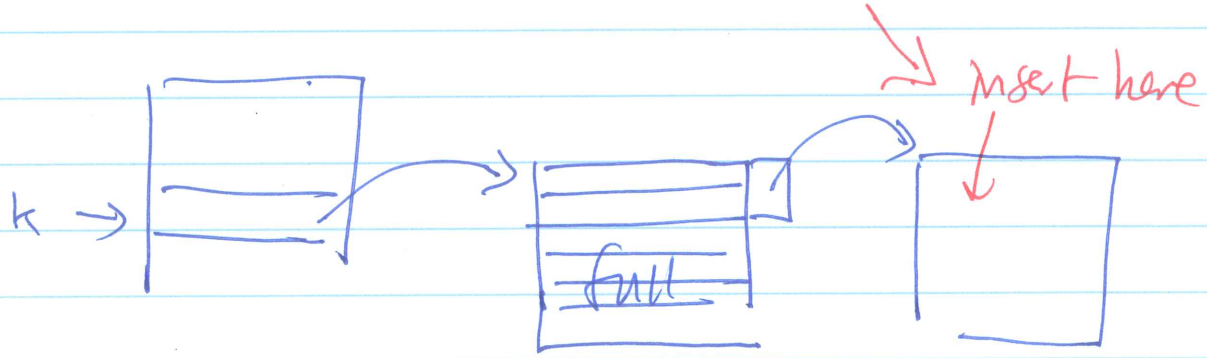


(3) If hash bucket has space

→ insert record.

(4) If hash bucket is full

→ allocate overflow block.

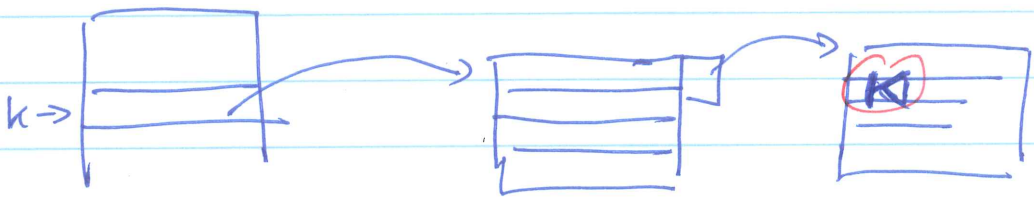


Delete from Hash Table.

- Delete (K)

(1) compute $k = h(K)$.

(2) Find hash bucket:



Delete record if found.

(3) Optionally:

Consolidate blocks of a bucket into few blocks.

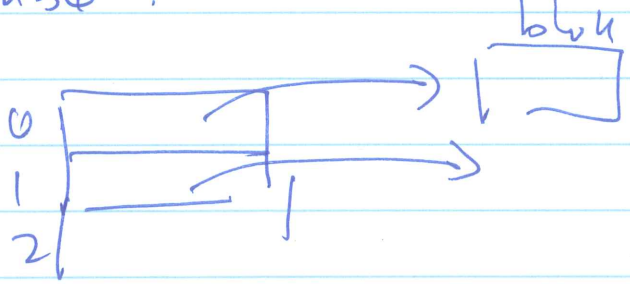
Note: consolidation can lead to oscillation!!!

policy (thresholds) → (insert → delete → insert → split / join / split ...)

Efficiency of Hash Table indexes.

- ideal: no overflow blocks.

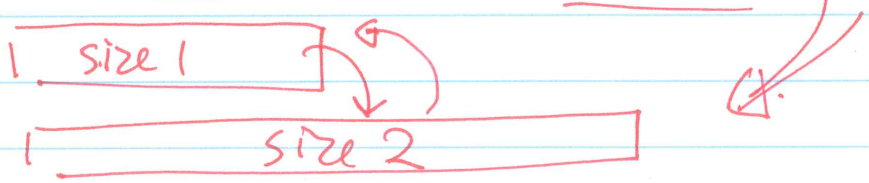
  ⇒ 1 block access to find record.
  (assuming that the block ptr table:



  can fit in memory.

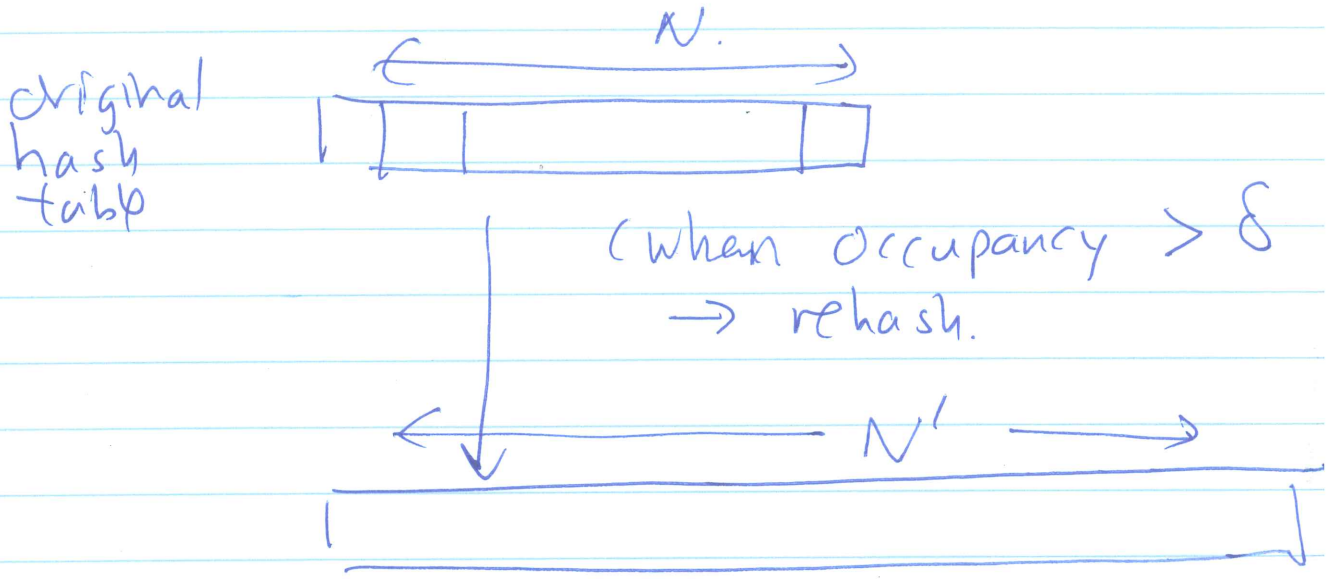
- With many overflow blocks
  → slower.

Solution ⇒ Dynamic size hash table.



Trivial Dynamic Sized Hash Table

- Re-hashing.



(1) Go through every record (key) in original hash table.

(2) Compute new hash index (location) and store in new hash table.

Problem: Very time consuming!