Hash Tables

* Hash Function:

```
search key \rightarrow \text{Hash function} \rightarrow \text{bucket index}
```

Where:

- \( K = \text{search key} \)
- \( h(K) = \text{bucket index} \)

* Bucket Array:

1. away at block pointers in memory

```
0
1
2
```

2. Special (reserved) tracks/cylinder on disk

(\text{each bucket index = 1 block pointer location})
Hash Tables on Disks:

Logically:

overflow
pointer

bucket
index

0

d

1
e
2
c
q
f

hash function

\[
\begin{align*}
  h(d) &= \emptyset \\
  h(e) &= 1 \\
  h(c) &= 1 \\
  h(a) &= 2 \\
  h(f) &= 2
\end{align*}
\]
Physically:

\[ h(e) = 1 \]

(can be):

1. away wit 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 \[ \rightarrow \text{insert record} \]
  4. If hash bucket is full \[ \rightarrow \text{allocate overflow block} \]
Delete from Hash Table.

- Delete \( K \)

1. Compute \( k = h(K) \).

2. Find hash bucket:

   \[
   k \rightarrow \text{[bucket]} \rightarrow \text{[chains]} \rightarrow \text{[deleted record]}
   \]

   Delete record if found.

3. Optionally:

   Consolidate blocks at a bucket into few blocks.

**Note:** Consolidation can lead to oscillation! 😱

**Policy:** (throw out)

  (must \( \rightarrow \) delete \( \rightarrow \) rearrange \( \rightarrow \) split / sum / split?)
Efficiency of Hash Table Indexes.

- **Ideal**: no overflow blocks.

  \[ \rightarrow 1 \text{ block access to find record} \]

  (assuming that the block ptr table:

  \[ \begin{array}{c}
  0 \rightarrow 1 \\
  1 \rightarrow 1 \\
  2 \\
  \end{array} \]

  can fit in memory.

- **With many overflow blocks**

  \[ \rightarrow \text{slower} \]

  Solution: **Dynamic size hash table**.

  \[ \begin{array}{c}
  \text{size 1} \\
  \text{size 2} \\
  \end{array} \]
Trivial Dynamic Sized Hash Table

- Re-hashing.

Original hash table

\[ \begin{array}{c}
N \\
1 \\
\end{array} \]

when occupancy > \( \frac{1}{2} \rightarrow \) rehash

\[ \begin{array}{c}
N' \\
1 \\
\end{array} \]

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

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

Problem: Very time consuming!