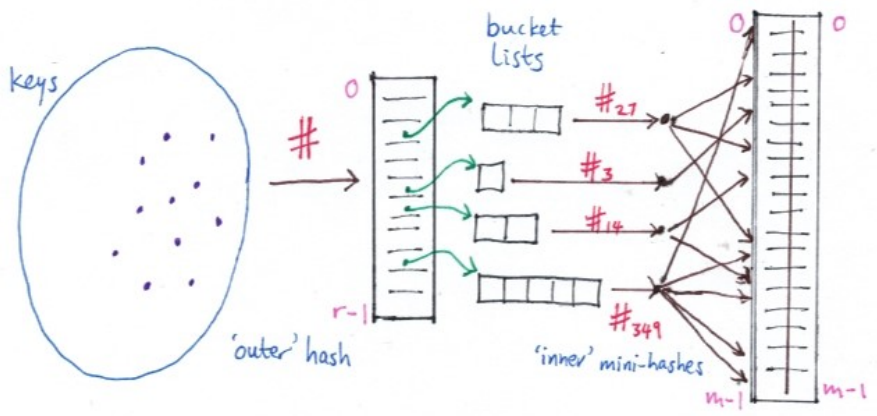
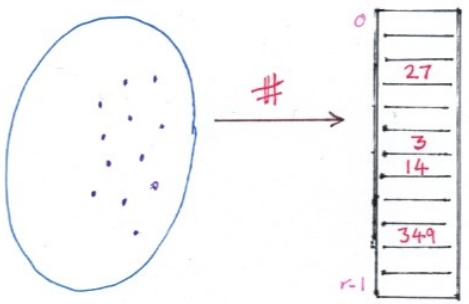
Perfect hashing: a state-of-the-art method

(Belazzougui, Botelho and Dietzfelbinger 2006).



Idea: Do traditional hash #, then choose separate mini-hash function for each bucket (from supply of suitable hash functions $\#_j$) so as to avoid clashes. To store the hash function itself, just need the table of j-values:



E.g. if $n/r = 5$ and each j-value fits in 9 bits, hash function can be stored in $\simeq$ 1.8 bits per key: tiny cost and independent of key length!

How do we choose the mini-hash functions $\#_j$?

Idea is blindingly simple, but it works:

- Go through the buckets largest-first, assigning a mini-hash $\#_j$ to each one and filling out main table accordingly.
- For each bucket, we choose $\#_j$ simply by trying out $\#_0, \#_1, \#_2, \ldots$ in turn, until we find one that gives no clashes with existing table entries (or internally within the bucket).

In early stages, buckets may be large, but table mostly empty: shouldn't take too long to find a $\#_j$ that works for that bucket.

Later on, table may be nearly full. But hopefully we're down to small buckets by then (1 or 2 keys), so still shouldn't take too long to find a suitable j (if load α not too close to 1).

Even if we sometimes have to search up to $j = 1000$, the number of bits occupied by j will stay small.

B, B, D report an information efficiency of 1.4 bits per key on large datasets (currently best known).