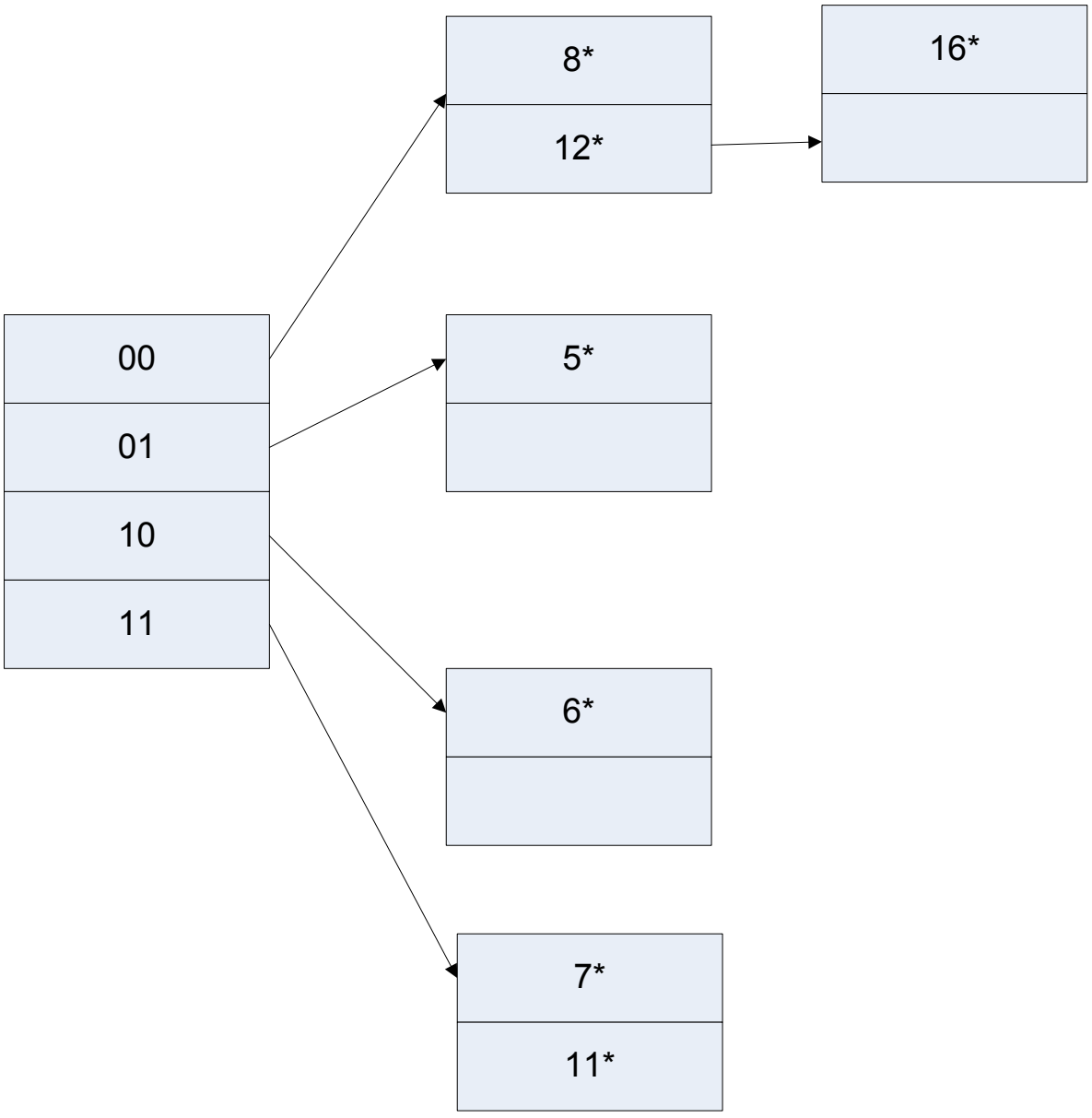
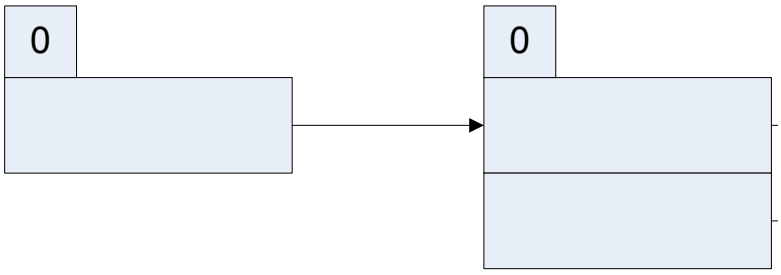
Static hashing

Static hashing has a fixed hash table size pointing to buckets. The data entries are put into the buckets, if the buckets fill an overflow bucket is created. Easy to implement, but can result in long overflow chains.

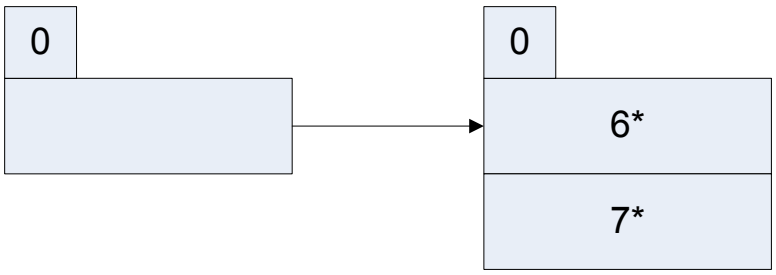


Extensible hashing

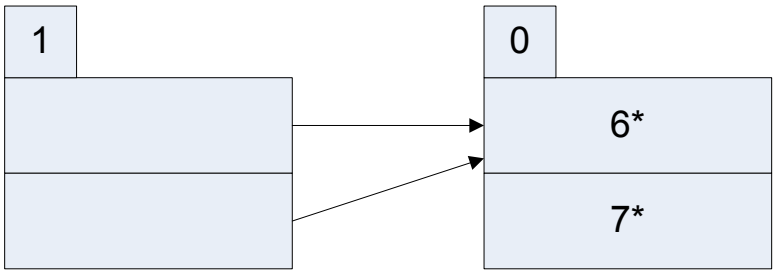
So our hash algorithm (h(x) = x mod 4) returns one of 4 values, 0, 1, 2, or 3. The following represent adding 6, 7, 8, 10, & 12 into a bucket size of 2. When we start out, we have the bucket address hash prefix set to 0, which says we are considering the first 0 bits of the hash value of each number.



Since the bucket address table has 2**0 entries (1), it is pointing to a bucket with two entries. This allows 6 & 7 to be entered without issue.

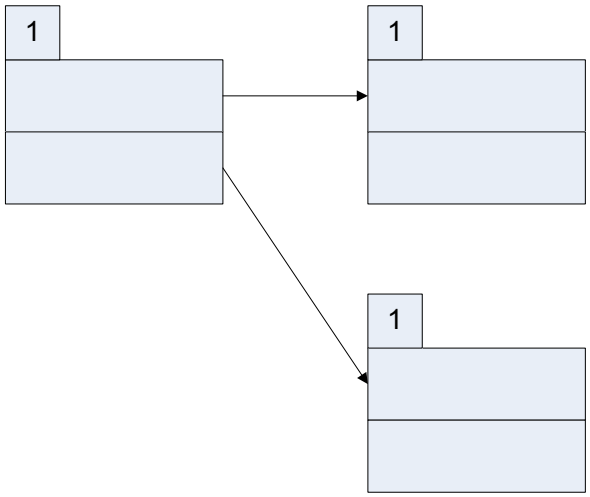


When we go to add 8, we hit the condition that the bucket is full. Since these are not duplicate hash values, we need to increase the hash suffix by 1, giving us 2**1 values in the bucket address table (2 entries). You can see this in Step 1.



Step 1.

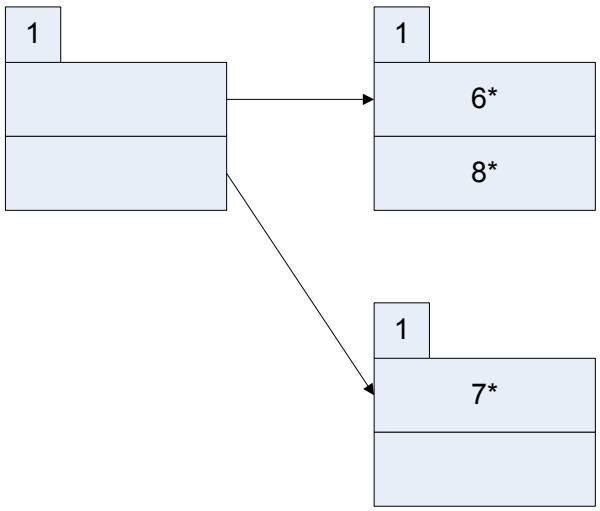
We then add another bucket and have the second entry in the bucket address table point to it (which is step 2 in the diag). We then rehash the values in the bucket that was full to begin with (6 & 7) using the new hash suffix (1) and place them in the bucket.



Step 2

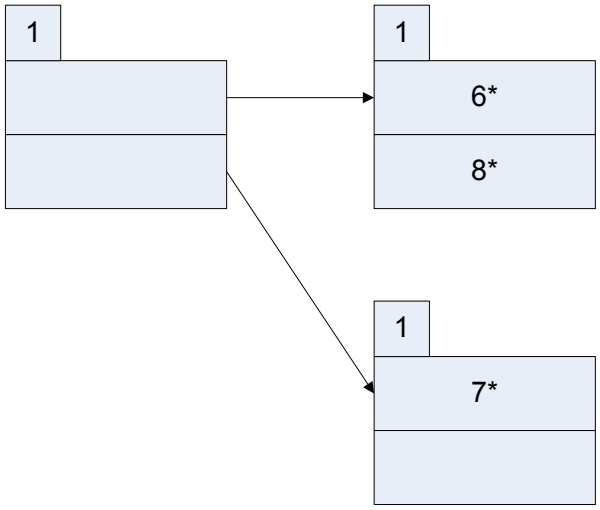
6 rehashes to 1 0, so using 1 bit of suffix, it goes in bucket 0
7 rehashes to 1 1, so using 1 bit of suffix it goes in bucket 1

Now we can place 8.
8 hashes to 0 0, so using 1 bit of suffix it goes to bucket 0.

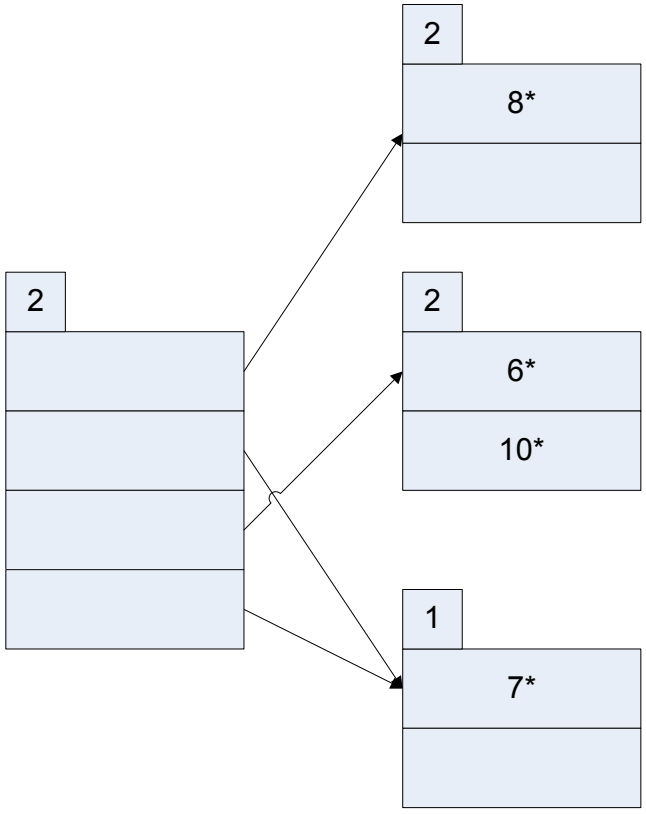


Step 3.

Ok, now we want to add index 10 to the table. We are starting with:



Because the hash values of the indexes are different, we know have to grow the table again. Now we have buckets for 00 (containing 8), 10 (containing 6 and 10), and 1 (containing 7). Since there have only been 1 index with a suffix of 1, we still only have 1 bucket and are only using 1 bit for it.



Note, when we expanded the hash table, we put both pointers in the hash suffix that didn't need to grow pointing to the same bucket. If that bucket were to fill, we would know by the difference in the level points (2 in the global struct, 1 in the local) and know we could split it and reset the pointers in the global struct.

Finally, we go to add 14. The bucket for 14 is full of indexes of the same hash value – so at this point we know we need to build an overflow bucket.