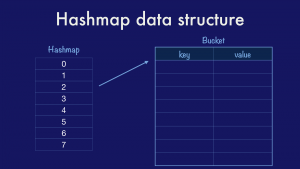
**The hashmap data structure**

The second part of a hashmap is the way data is stored.  


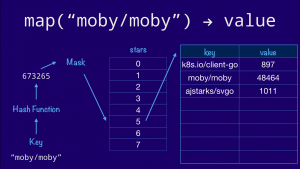
The classical hashmap is an array of *buckets* each of which contains a pointer to an array of key/value entries. In this case our hashmap has eight buckets (as this is the value that the Go implementation uses) and each bucket can hold up to eight entries each (again drawn from the Go implementation). Using powers of two allows the use of cheap bit masks and shifts rather than expensive division.

As entries are added to a map, assuming a good hash function distribution, then the buckets will fill at roughly the same rate. Once the number of entries across each bucket passes some percentage of their total size, known as the *load factor,* then the map will grow by doubling the number of buckets and redistributing the entries across them.

With this data structure in mind, if we had a map of project names to GitHub stars, how would we go about inserting a value into the map?



We start with the key, feed it through our hash function, then mask off the bottom few bits to get the correct offset into our bucket array. This is the bucket that will hold all the entries whose hash ends in three (011 in binary). Finally we walk down the list of entries in the bucket until we find a free slot and we insert our key and value there. If the key was already present, we’d just overwrite the value.



Now, lets use the same diagram to look up a value in our map. The process is similar. We hash the key as before, then masking off the lower 3 bits, as our bucket array contains 8 entries, to navigate to the fifth bucket (101 in binary). If our hash function is correct then the string "moby/moby" will always hash to the same value, so we know that the key will not be in any other bucket. Now it’s a case of a linear search through the bucket comparing the key provided with the one stored in the entry.

**Four properties of a hash map**

That was a very high level explanation of the classical hashmap. We’ve seen there are four properties you need to implement a hashmap;

* 1. You need a hash function for the key.
  2. You need an equality function to compare keys.
  3. You need to know the size of the key and,
  4. You need to know the size of the value because these affect the size of the bucket structure, which the compiler needs to know, as you walk or insert into that structure, how far to advance in memory.