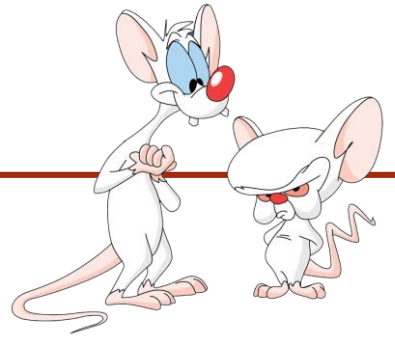
COMP 250
INTRODUCTION TO COMPUTER SCIENCE

33 - Hashing

Giulia Alberini, Fall 2022

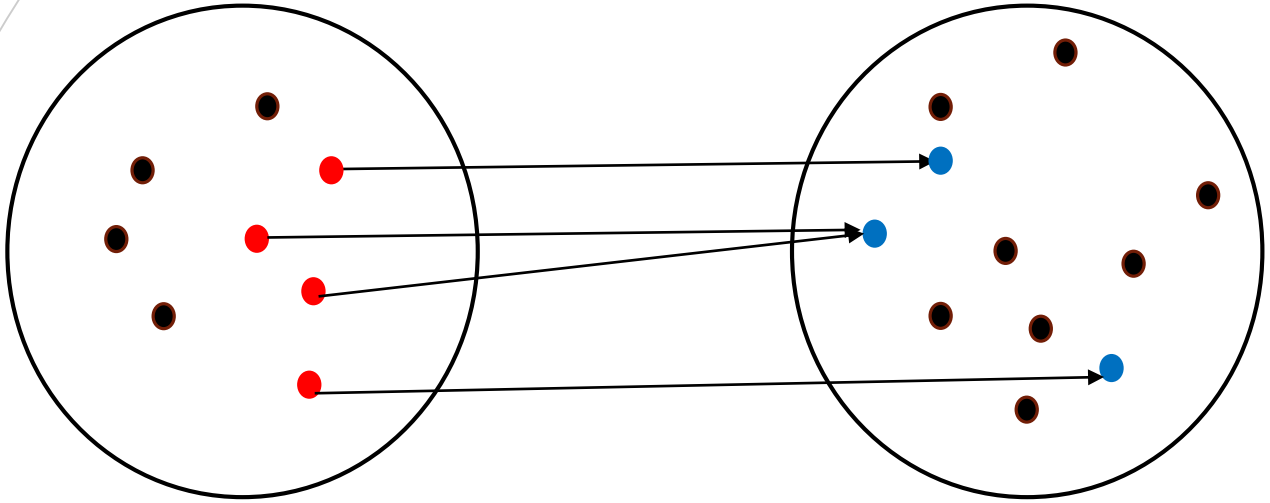
Slides adapted from Michael Langer's

WHAT ARE WE GOING TO DO TODAY?



- Hash Maps

RECALL: MAP



keys (type K)

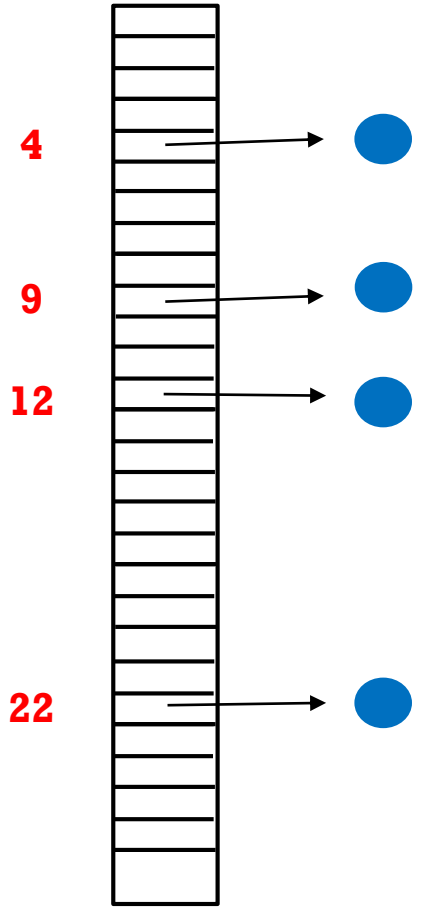
values (type V)

Each (key, value) pairs is an "entry".
For each key, there is at most one value.

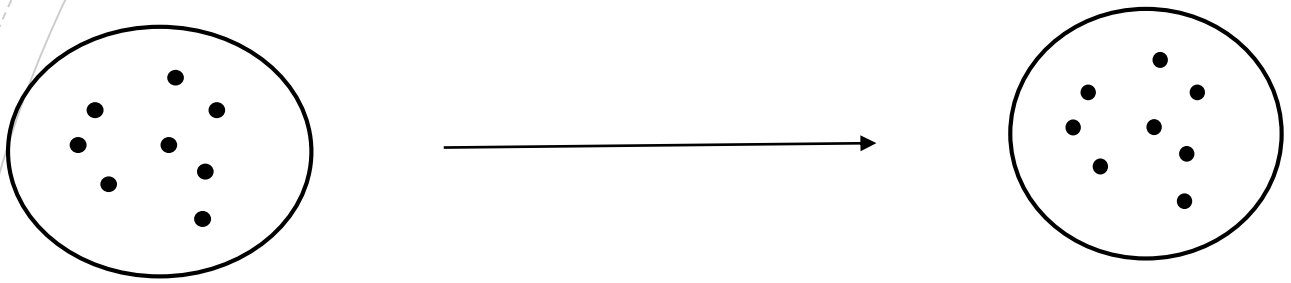
ARRAYS OF VALUES

Then, we could use an array of type V (value) and have O(1) access.

This would work well is keys are small integers.



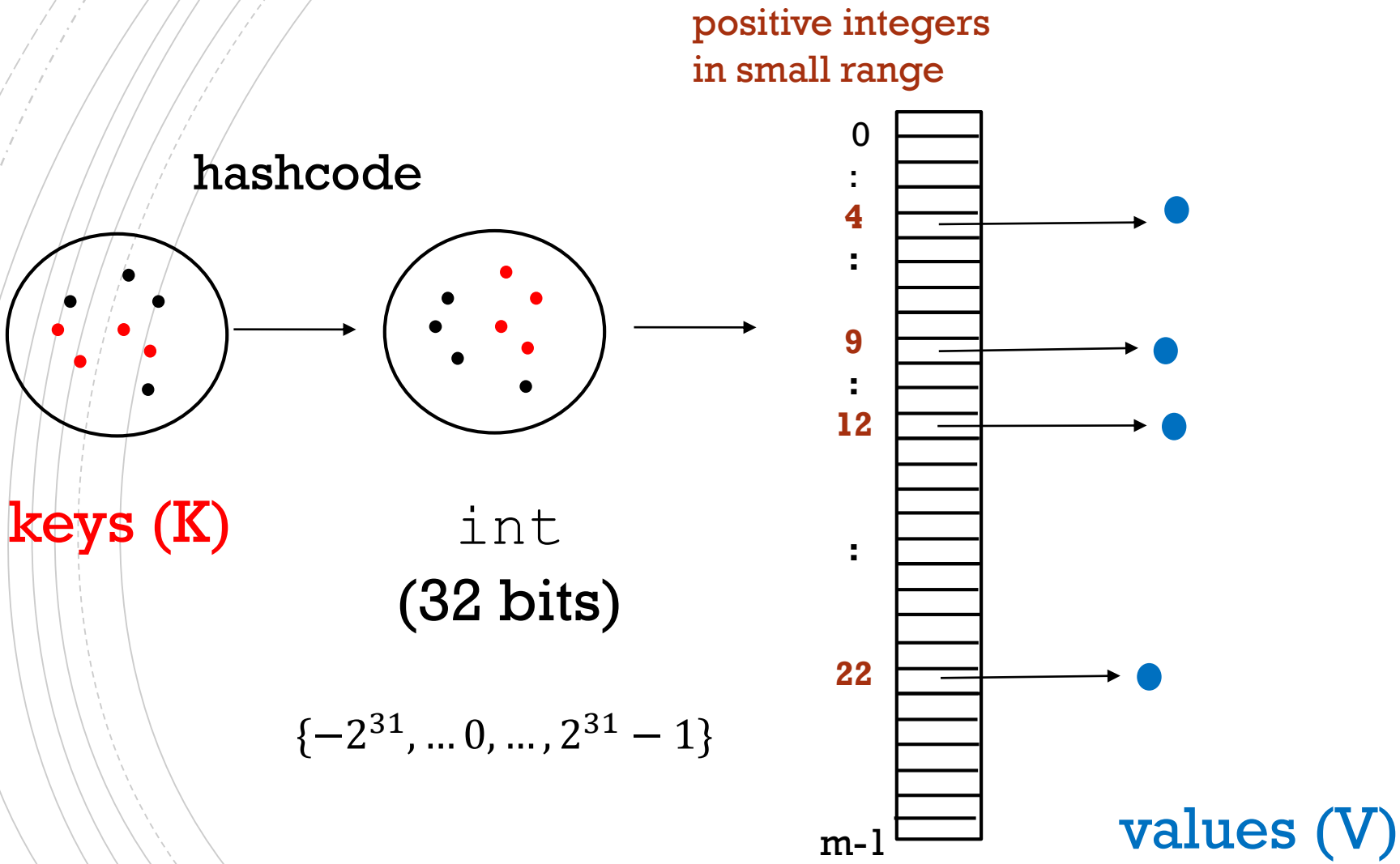
JAVA HASHCODE()



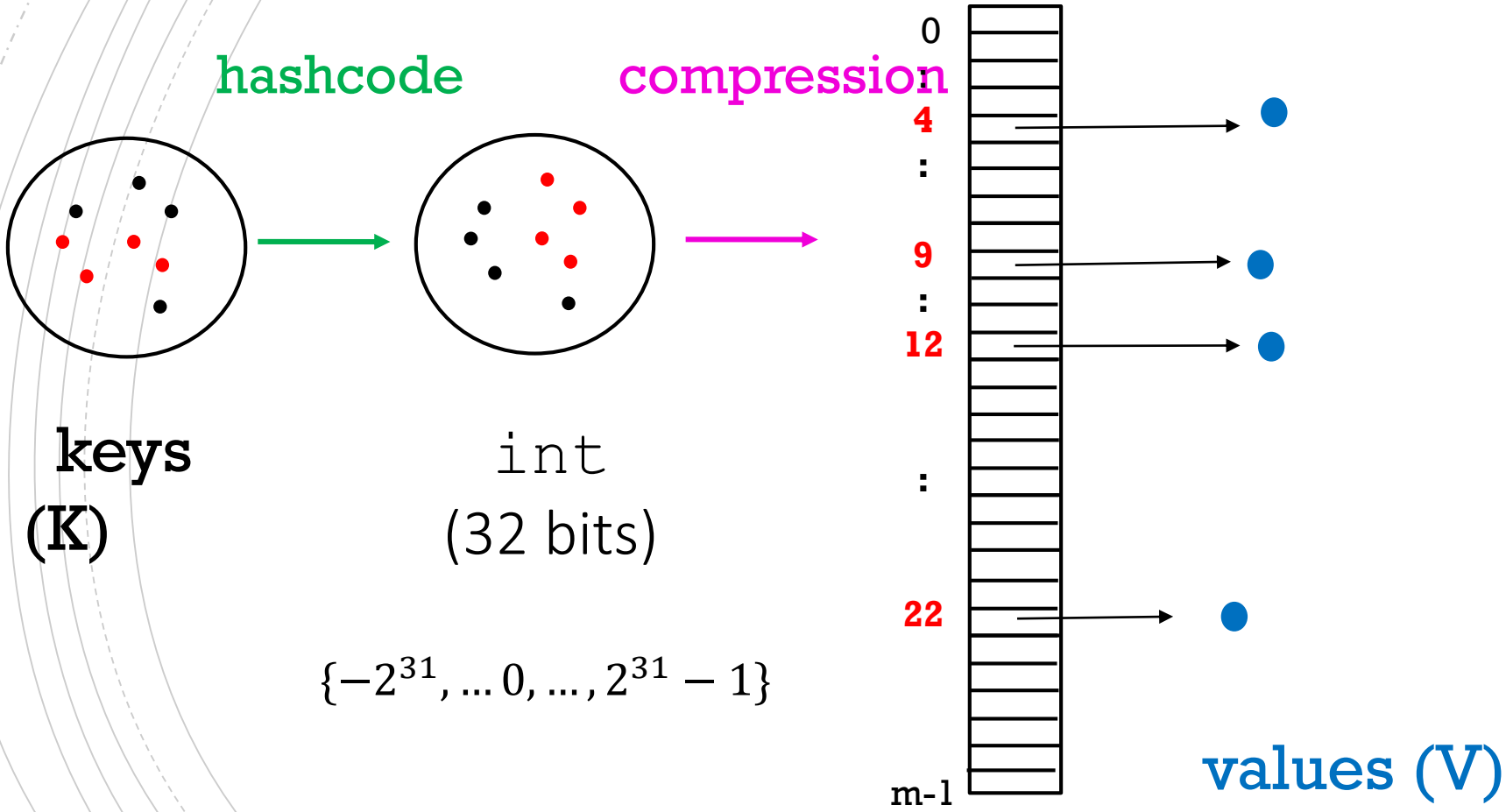
keys K

`int`
(32 bits)

TODAY: MAP COMPOSITION



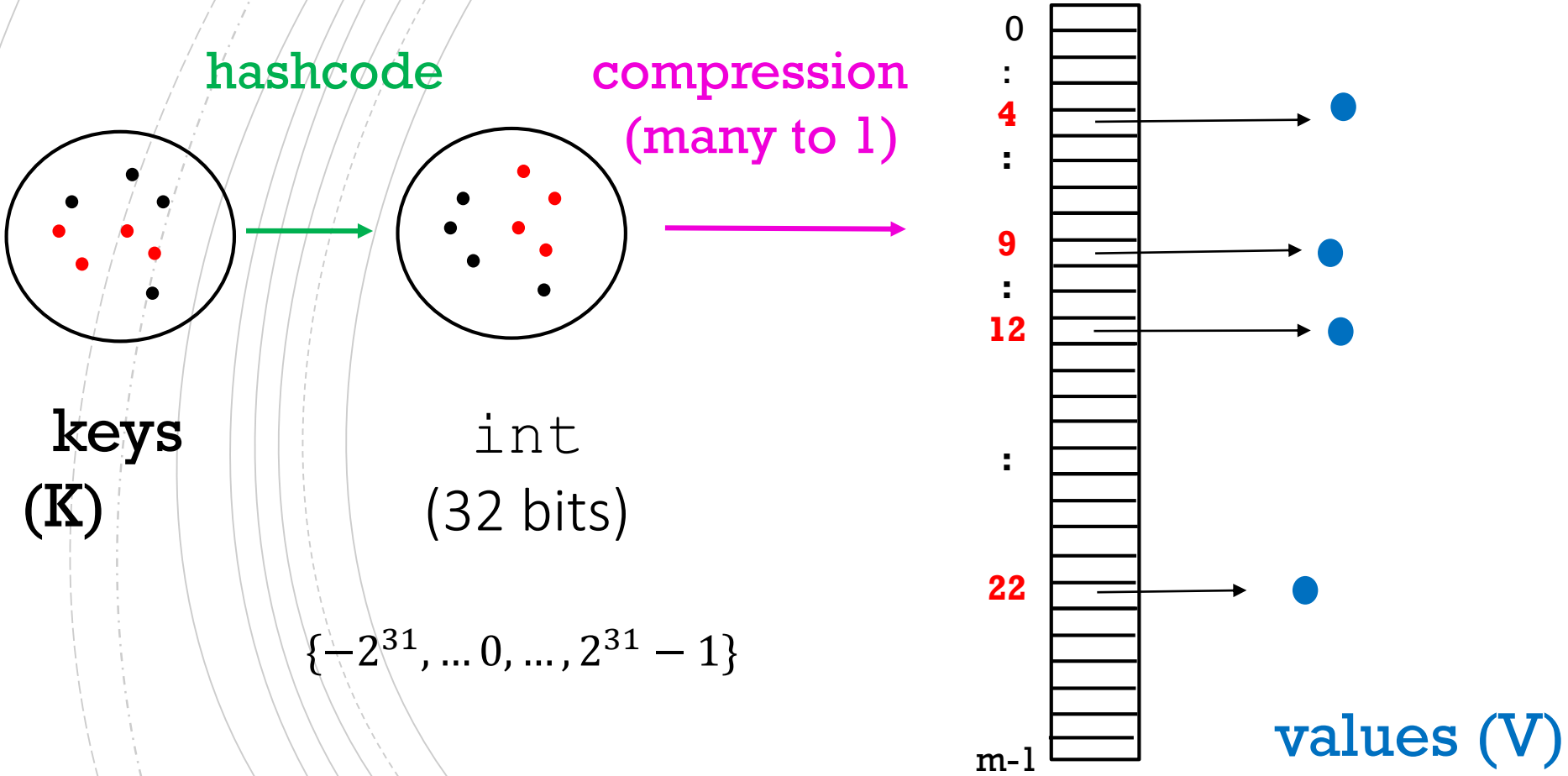
COMPRESSION MAP



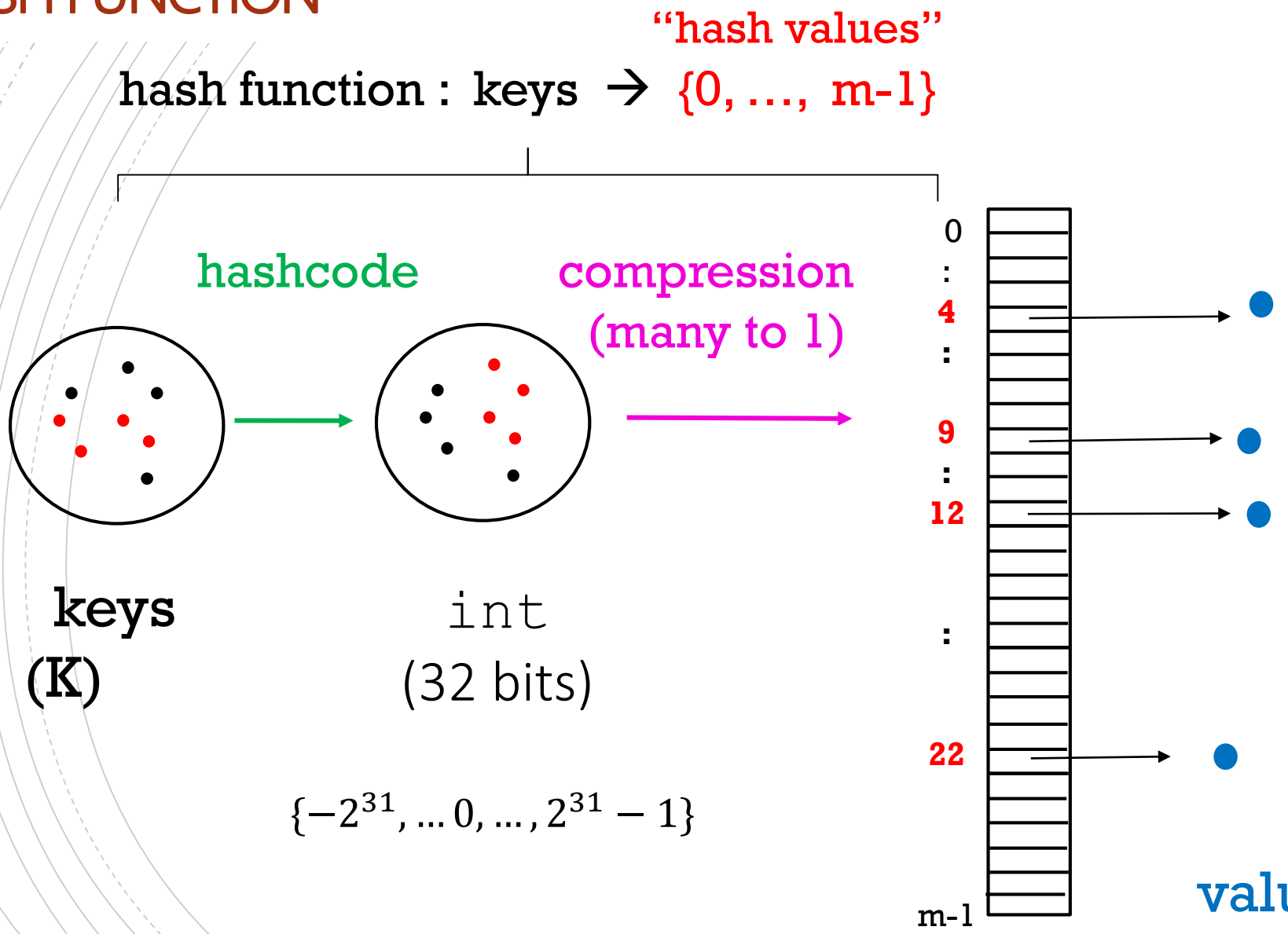
COMPRESSION MAP

compression: $i \rightarrow |i| \ mod \ m,$

where m is the length of the array.



HASH FUNCTION

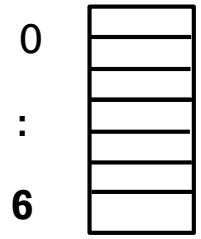


EXAMPLE

- Let m = 7

"hash function" ≡ compression ° hashCode

hash code	hash value (hash code % 7)
41	6
16	2
25	4
21	0
36	1
35	0
53	4

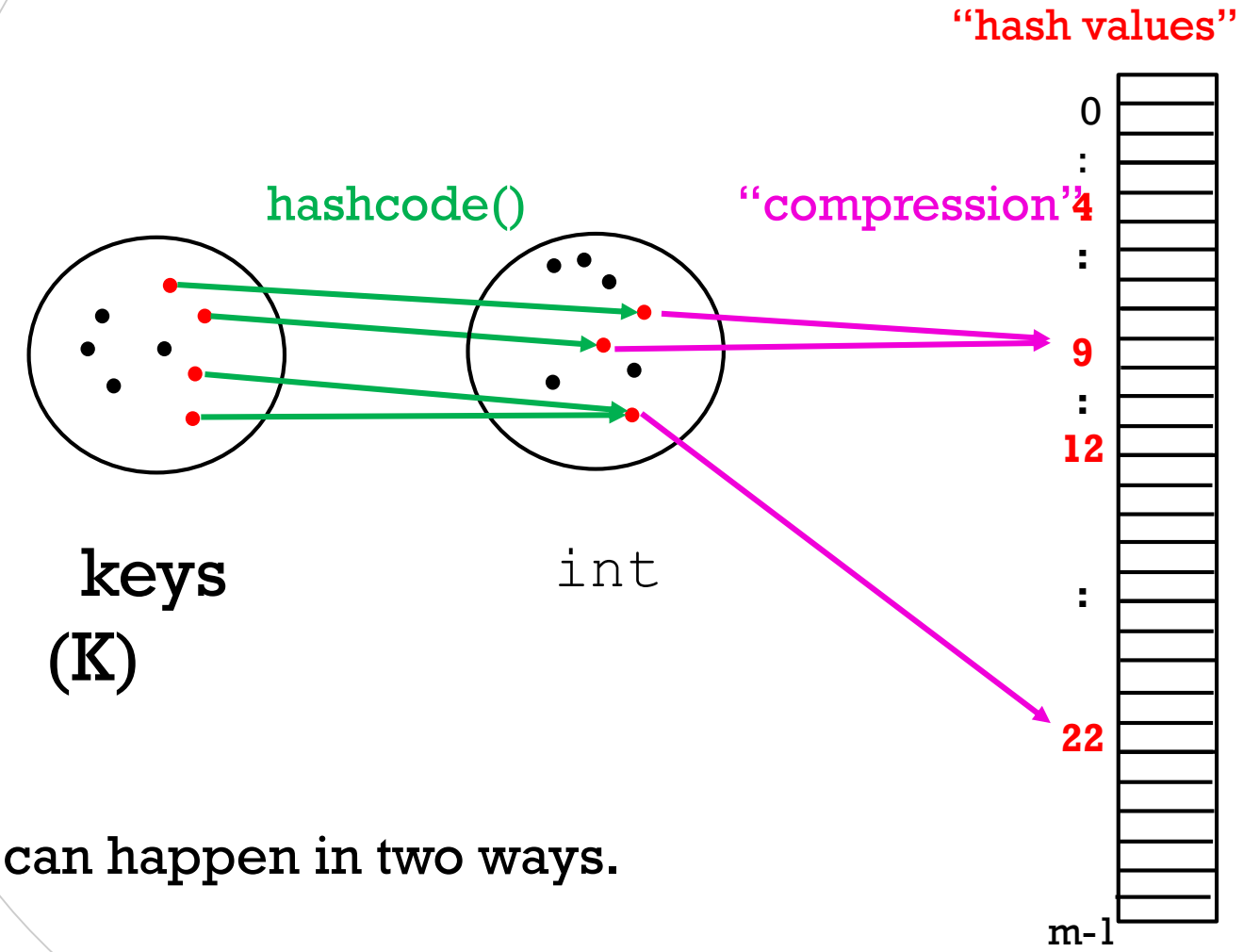


TERMINOLOGY

- A "hashCode" maps keys to `int`
- A "hash function" maps keys to "hash values"
- We use values both to refer to the values of the hash function as well as the values in the key-value pairs of the map we want to represent!

PROBLEM : COLLISIONS

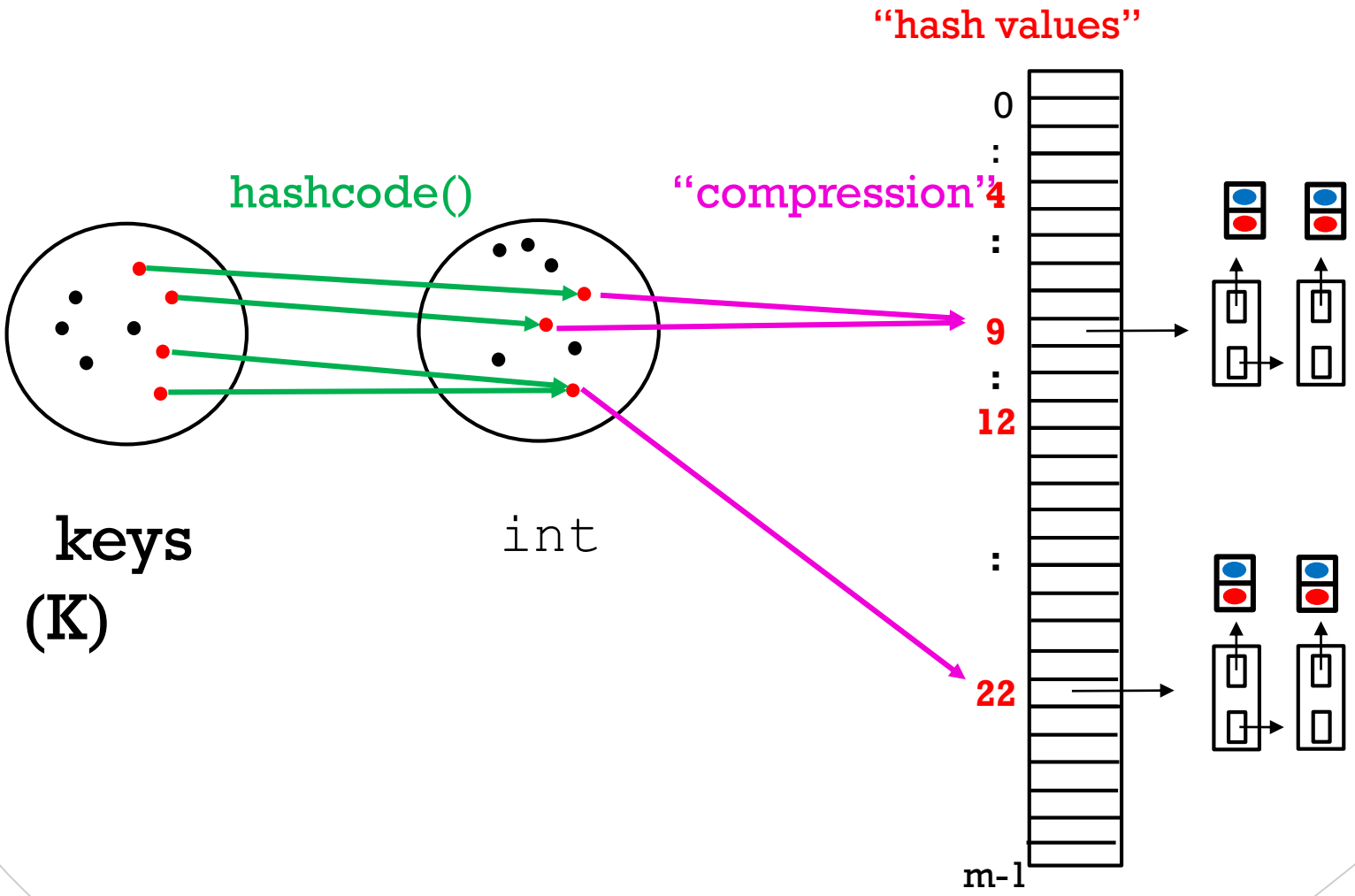
Two or more keys can map to the same hash value.



Collisions can happen in two ways.

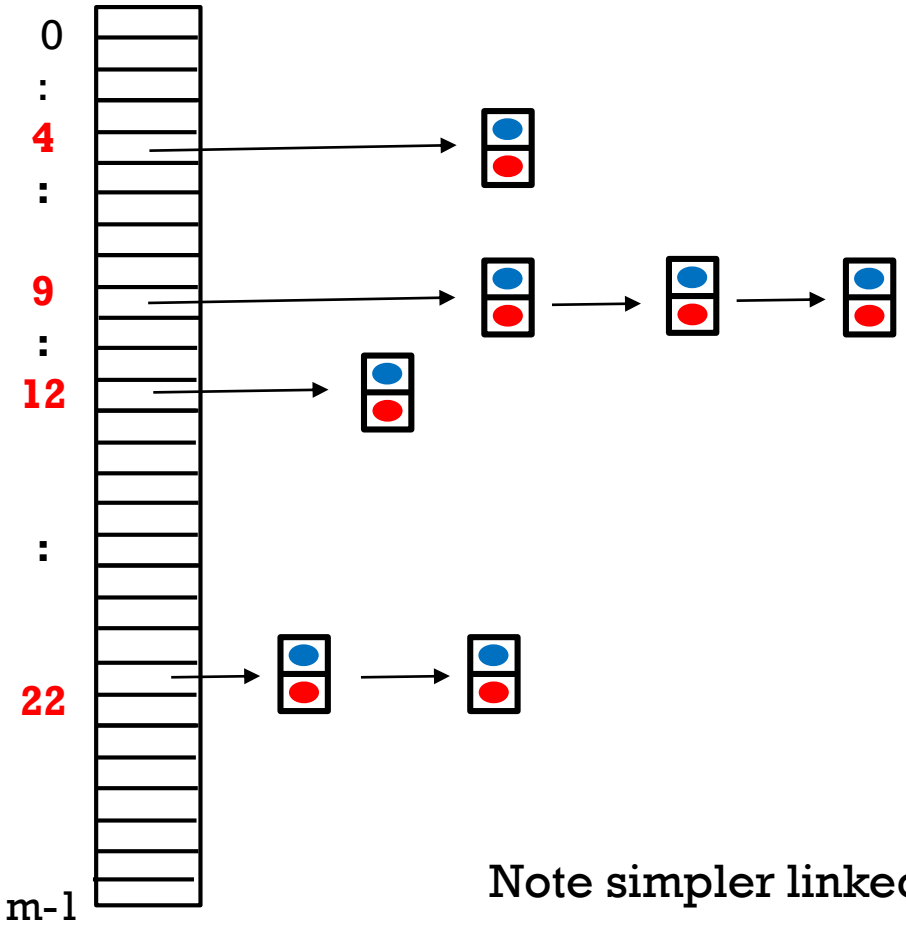
SOLUTION : HASH TABLE (OR HASH MAP)

Each array slot holds a singly linked list of entries



BUCKETS

Each array slot + linked list is called a bucket. This map has *m* buckets.



Note simpler linked list notation here.

OBSERVATIONS

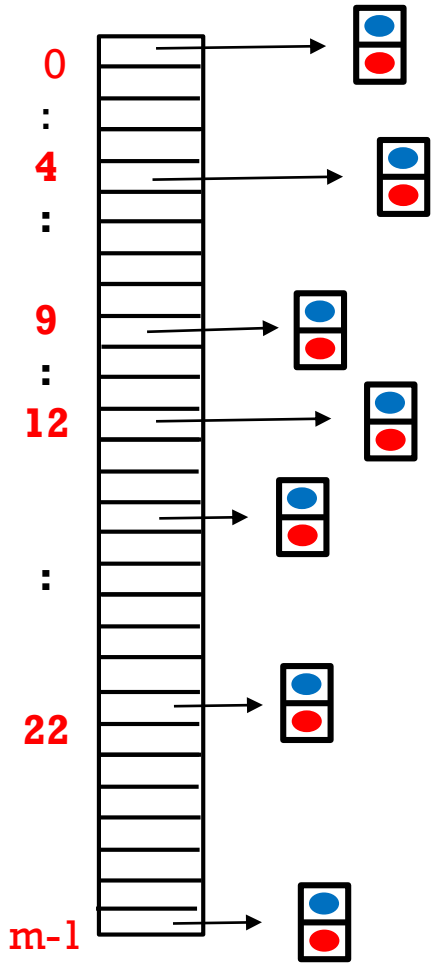
- Why is it necessary to store (key, value) pairs in the linked list? Why not just the values?

LOAD FACTOR

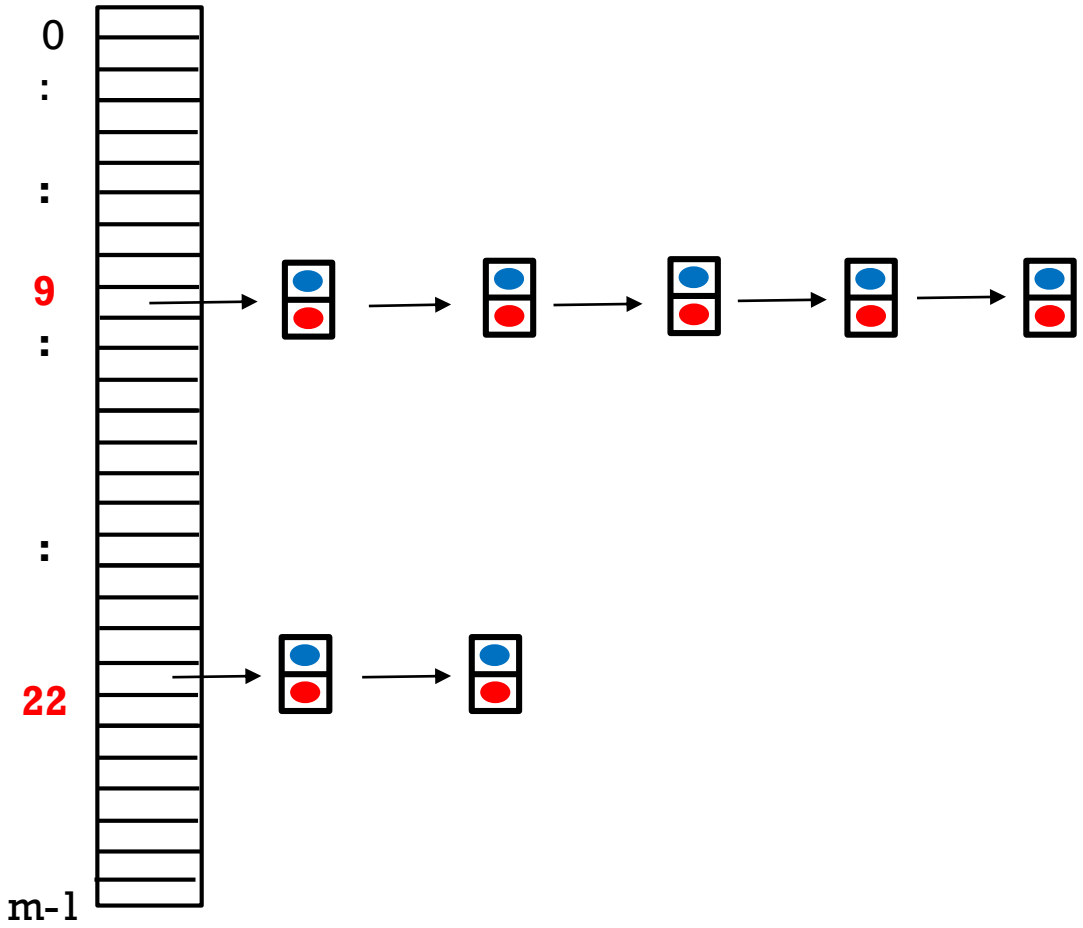
$$\equiv \frac{\text{number of (key, value) pairs in map}}{\text{number of buckets, } m}$$

One typically keeps the load factor below 1.
In the Java `HashMap` class, the default MAXIMUM load factor is 0.75

EXAMPLE OF A "GOOD HASH"



EXAMPLE OF A "BAD HASH"



EXAMPLE

$$h : K \rightarrow \{0, 1, \ldots, m-1\}$$

Example: Suppose keys are McGill Student IDs,

e.g. 260745918.

How many buckets to choose ?

Good hash function?

Bad hash function ?

EXAMPLE

$$h : K \rightarrow \{0, 1, \ldots, m-1\}$$

Example: Suppose keys are McGill Student IDs,

e.g. 260745918.

How many buckets to choose → number of entries

Good hash function?

Bad hash function ?

EXAMPLE

$$h : K \rightarrow \{0, 1, \ldots, m-1\}$$

Example: Suppose keys are McGill Student IDs,

e.g. 260745918.

How many buckets to choose ? → number of entries

Good hash function? → rightmost 5 digits

Bad hash function ?

EXAMPLE

$$h : K \rightarrow \{0, 1, \ldots, m-1\}$$

Example: Suppose keys are McGill Student IDs,

e.g. 260745918.

How many buckets to choose → number of entries

Good hash function? → rightmost 5 digits

Bad hash function ? → leftmost 5 digits

PERFORMANCE OF HASH MAPS

- put(key, value)
- get(key)
- remove(key)

If load factor is less than 1 and if hash function is good, then operations are O(1) "in practice". This beats all potential map data structures we discussed last video.

If we have a bad hash, we can choose a different hash function.

PERFORMANCE OF HASH MAPS

- put(key, value)
- get(key)
- remove(key)
- contains(value) ?

PERFORMANCE OF HASH MAPS

- put(key, value)
- get(key)
- remove(key)
- contains(value)

We will need to look through each of the *m* buckets (i.e. search each linked list for that value)

PERFORMANCE OF HASH MAPS

- put(key, value)
- get(key)
- remove(key)
- contains(value)
- getKeys()
- getValues()

These last three methods all require traversing the hash table which takes time $O(n + m)$ where n is the number of entries and m is the number of buckets.

JAVA `HashMap<K,V>` CLASS

- In constructor, you can specify initial number *m* of buckets, and maximum load factor
  (by default m = 16, and max load factor = 0.75)

- How is hash function specified ?

JAVA `HashMap<K,V>` CLASS

- In constructor, you can specify initial number m of buckets, and maximum load factor
  (by default m = 16, and max load factor = 0.75)

- How is hash function specified ?

  Use key's hashCode(), take absolute value, and compress it by taking mod of the number of buckets.

$$i \rightarrow |i| \; mod \; m$$

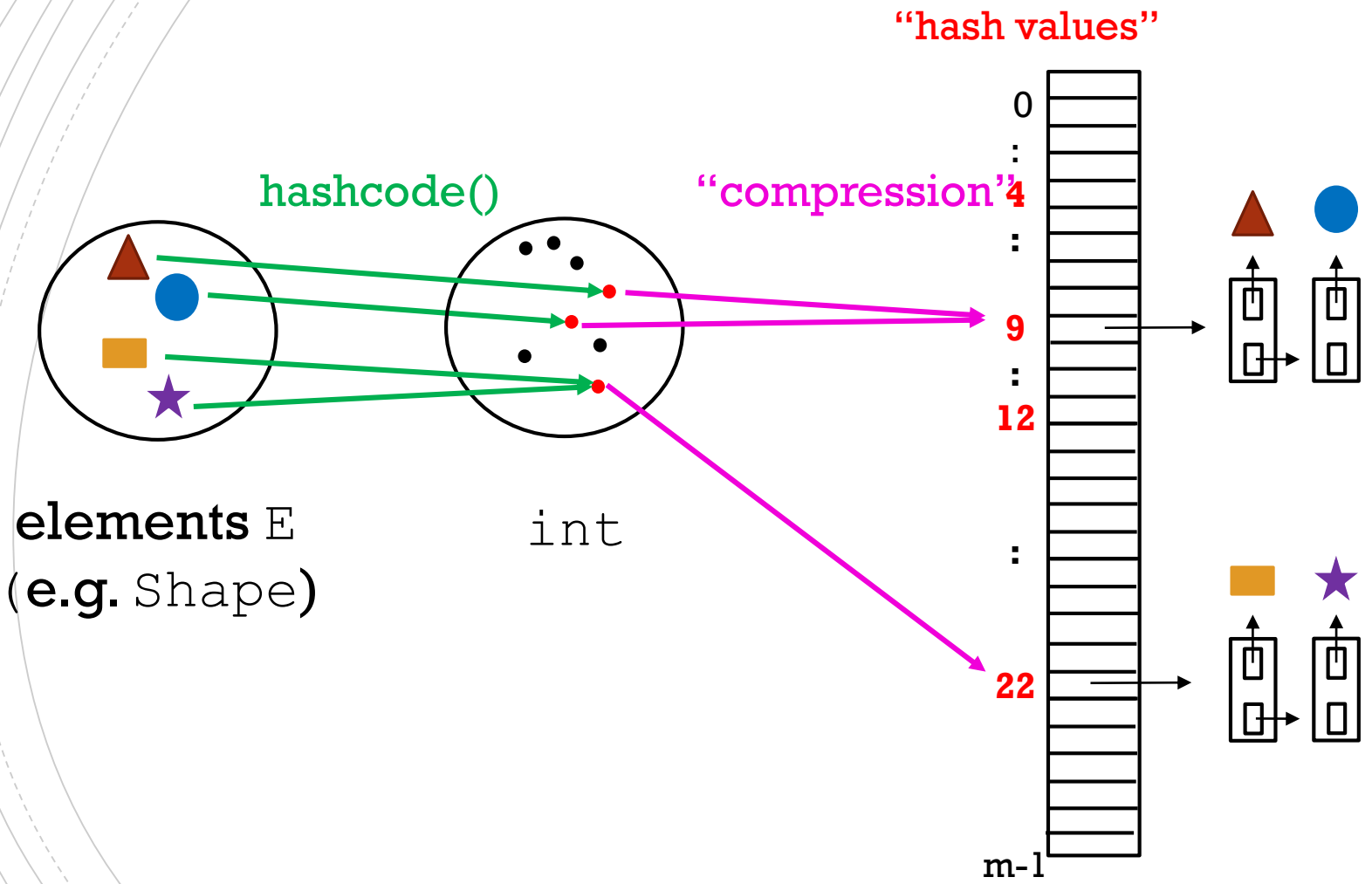
JAVA `HashSet<E>` CLASS

Similar to HashMap, but there are no values. Just use it to store a *set* of objects of some type. Operations:

- add(e)
- contains(e)
- remove(e)
- …

If hash function is good, then these operations are O(1). Note that this is not a list! There's no order in the elements and elements must be unique.

JAVA HashSet<E>



Coming next:

- Graphs