MAKE
SCHOOL

HASH TABLES

The Ultimate Data Structure

MAKE SCHOOL

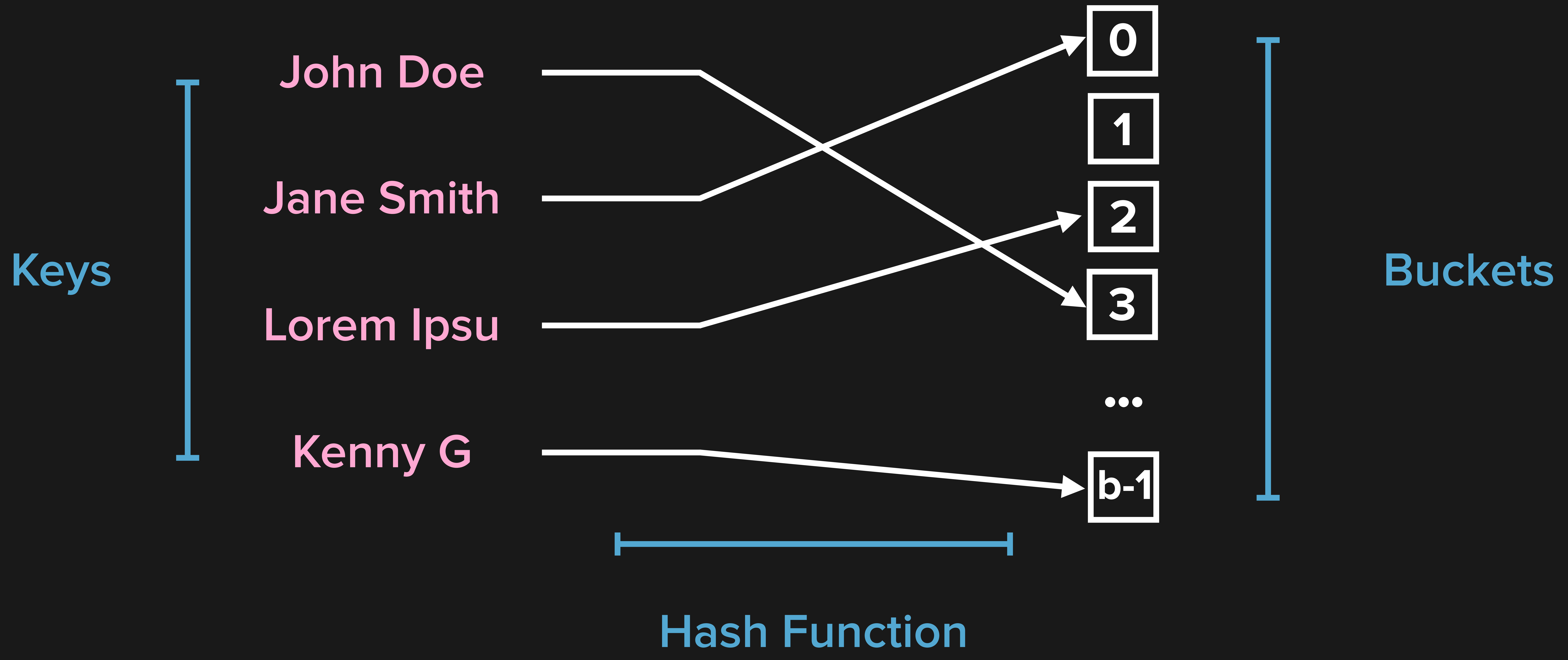
HASH TABLES

Maps keys ➔ values (any objects)

Python’s `dict()` / `{}` type is a hash table

Used because of strong average case performance (time complexity)

HASH TABLES


Keys
John Doe
Jane Smith
Lorem Ipsu
Kenny G
0
1
2
3
...
b-1
Buckets
Hash Function

HASH FUNCTIONS

Converts a variable-size input (key) to a fixed-size integer output (hash code)

Same input → same output

Input can be many types: number (int or float), string, or immutable collection

John Doe → 512340

Jane Smith → 408749

Lorem Ipsu → 943275

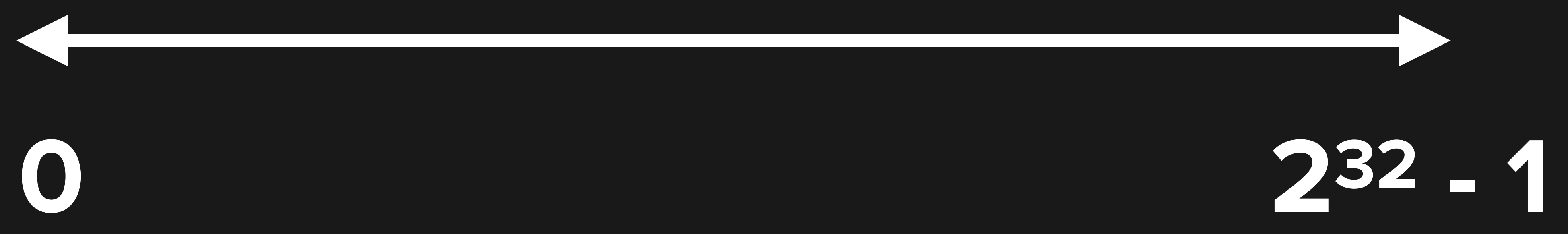
John Doe → 512340

IDEAL HASH FUNCTION*

Repeatable

Fast

Output is unsigned integer



Arbitrarily distributes keys among output space

Small differences in input result in large differences in output

*Different for cryptographic hash functions

STRING HASHING

Strings are sequences of characters

Characters have numerical values (ASCII codes)

Calculate a string’s hash code by adding up all characters’ ASCII codes (“Lose Lose” algorithm)

Note that `hash("dog") == hash("god")`

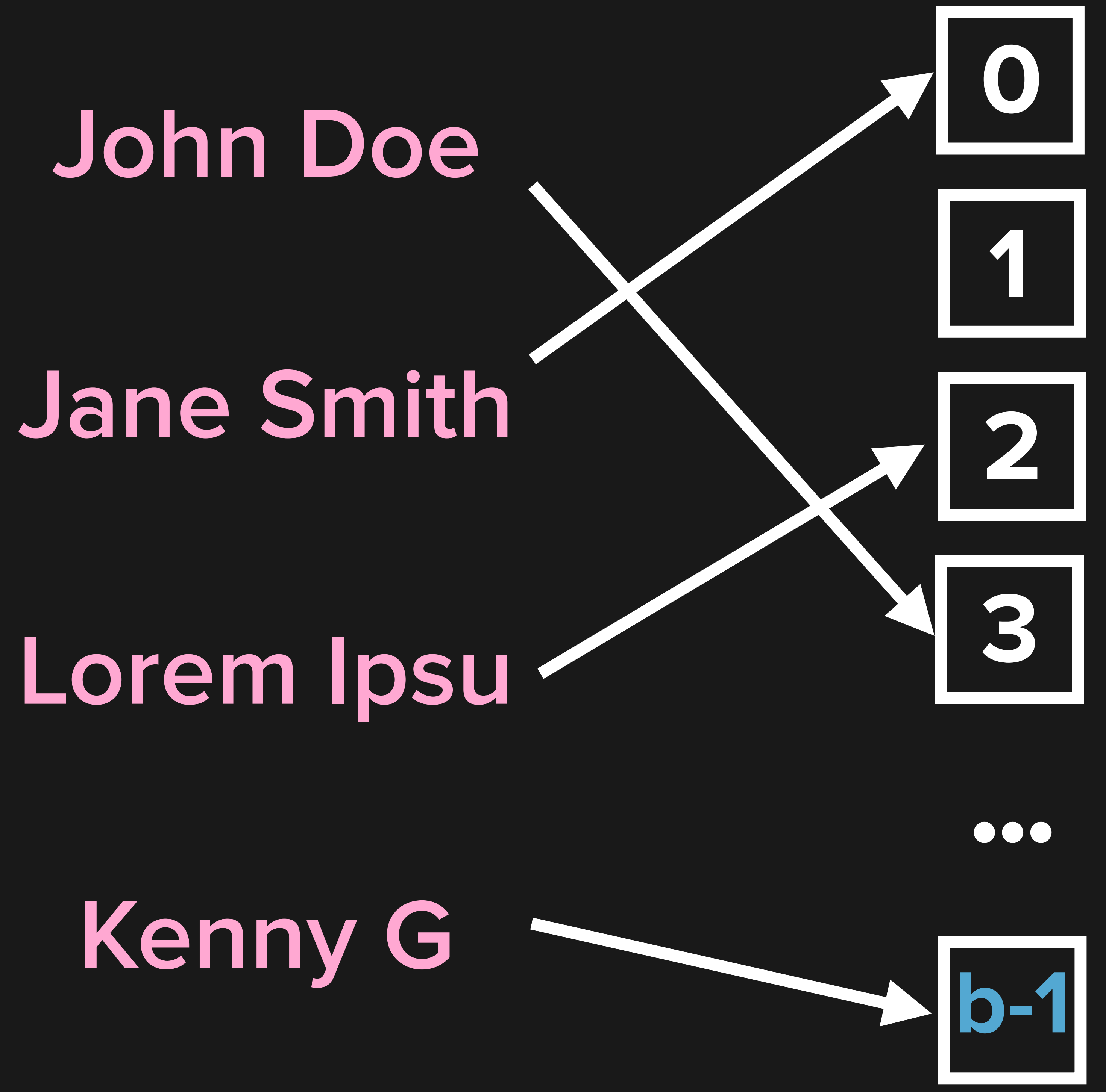
WHICH BUCKET?

Hash codes are very large integers, but we want the index of a bucket

We can use the modulus operator `%`

```
index = hash(key) % buckets
```

`index` ranges from `0` to `buckets-1`



HASH COLLISIONS

It is impossible to map all possible inputs to a fixed output space without some inputs generating the same output (hash code)

Different inputs (keys) generating the same output (hash code) is called a *hash collision*

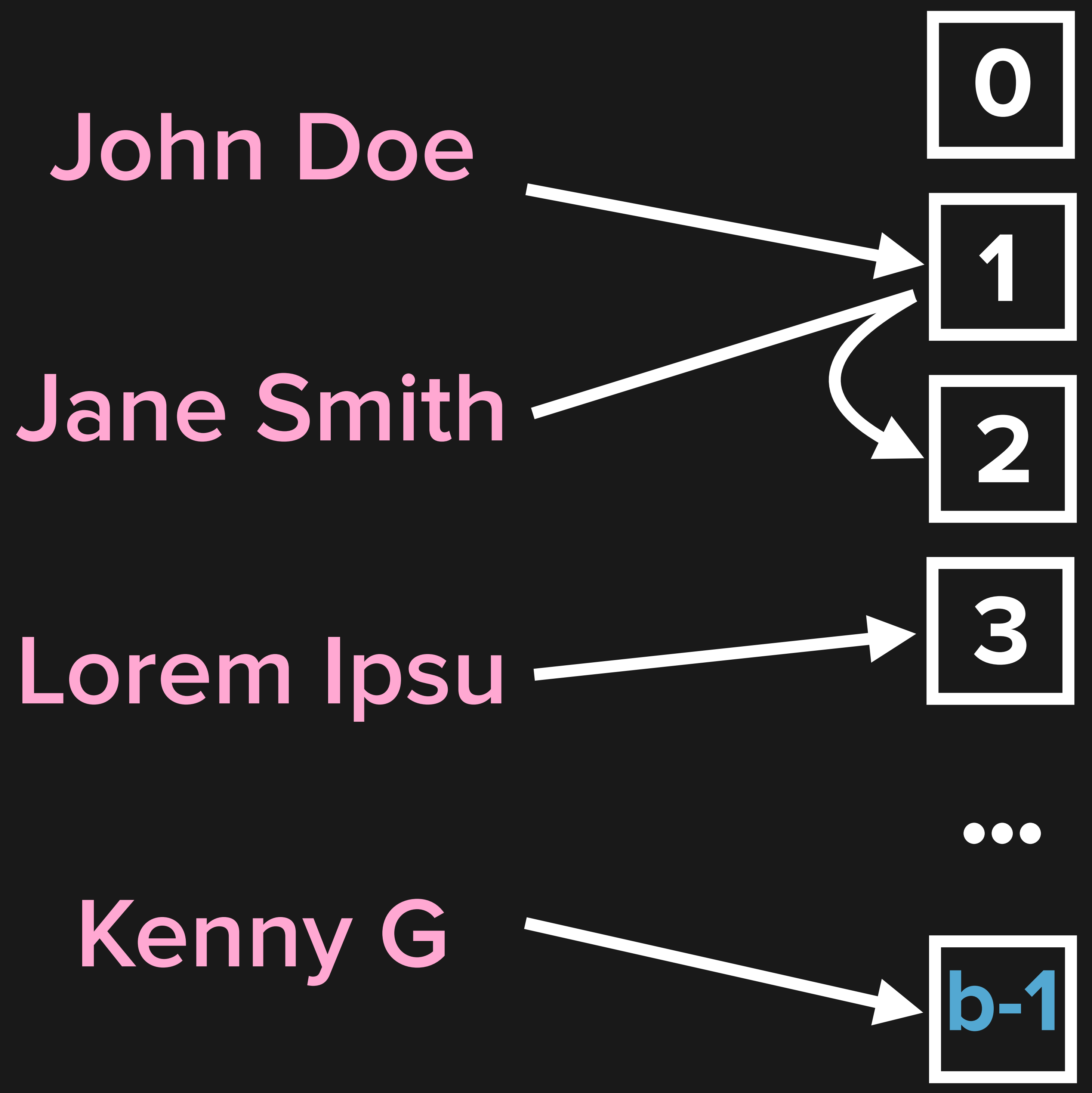
LINEAR PROBING

Each bucket contains at most one entry

On collision - find next open bucket, add entry there

To retrieve - find bucket, if that’s not entry, try next bucket until you find entry or empty bucket

Python’s `dict` uses probing



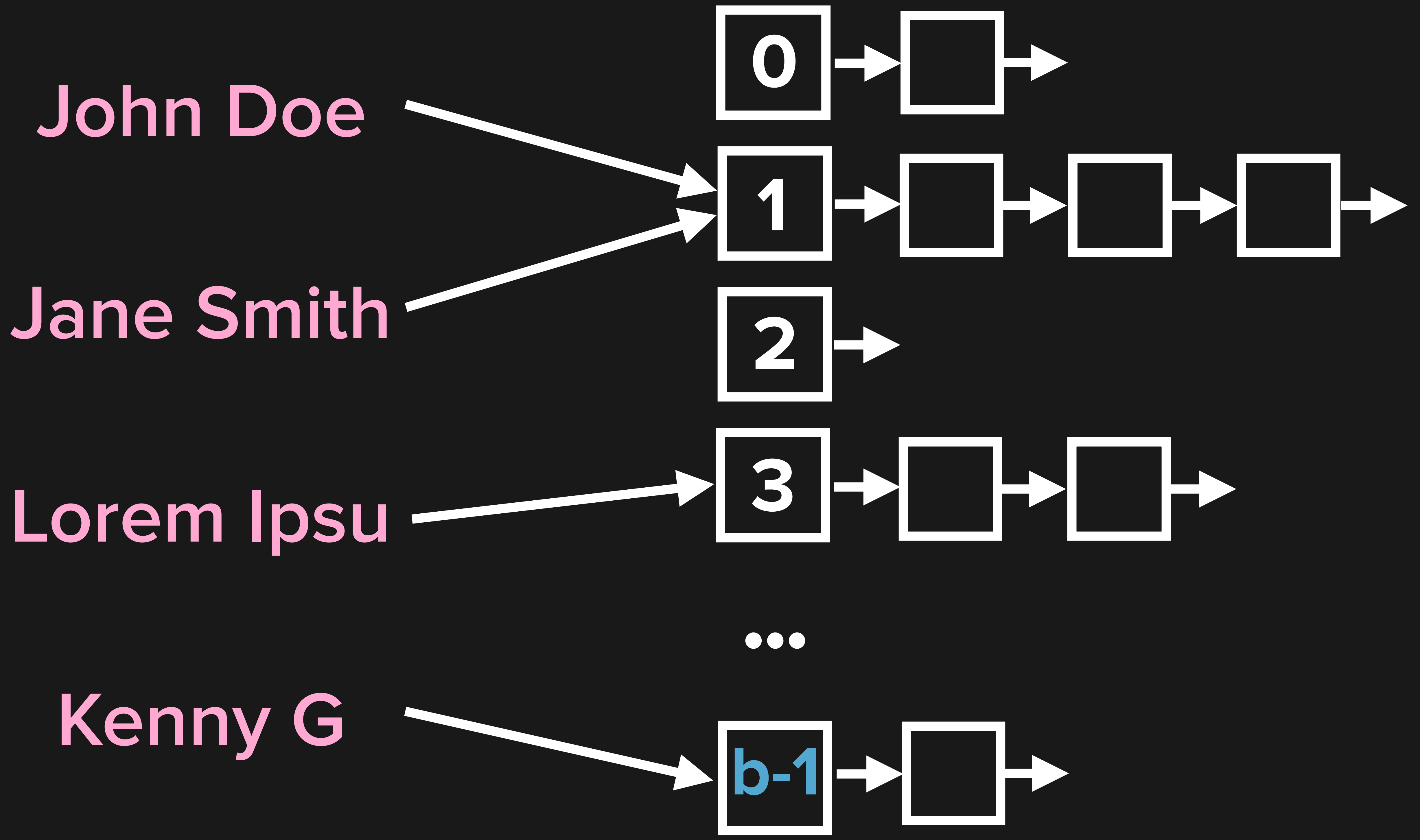
CHAINING

Each bucket contains a linked list of entries

On collision - add to the bucket's linked list

To retrieve - find bucket, find entry in linked list

We will use chaining to implement our hash table



LOAD FACTOR

```
Load Factor = entries / buckets
```

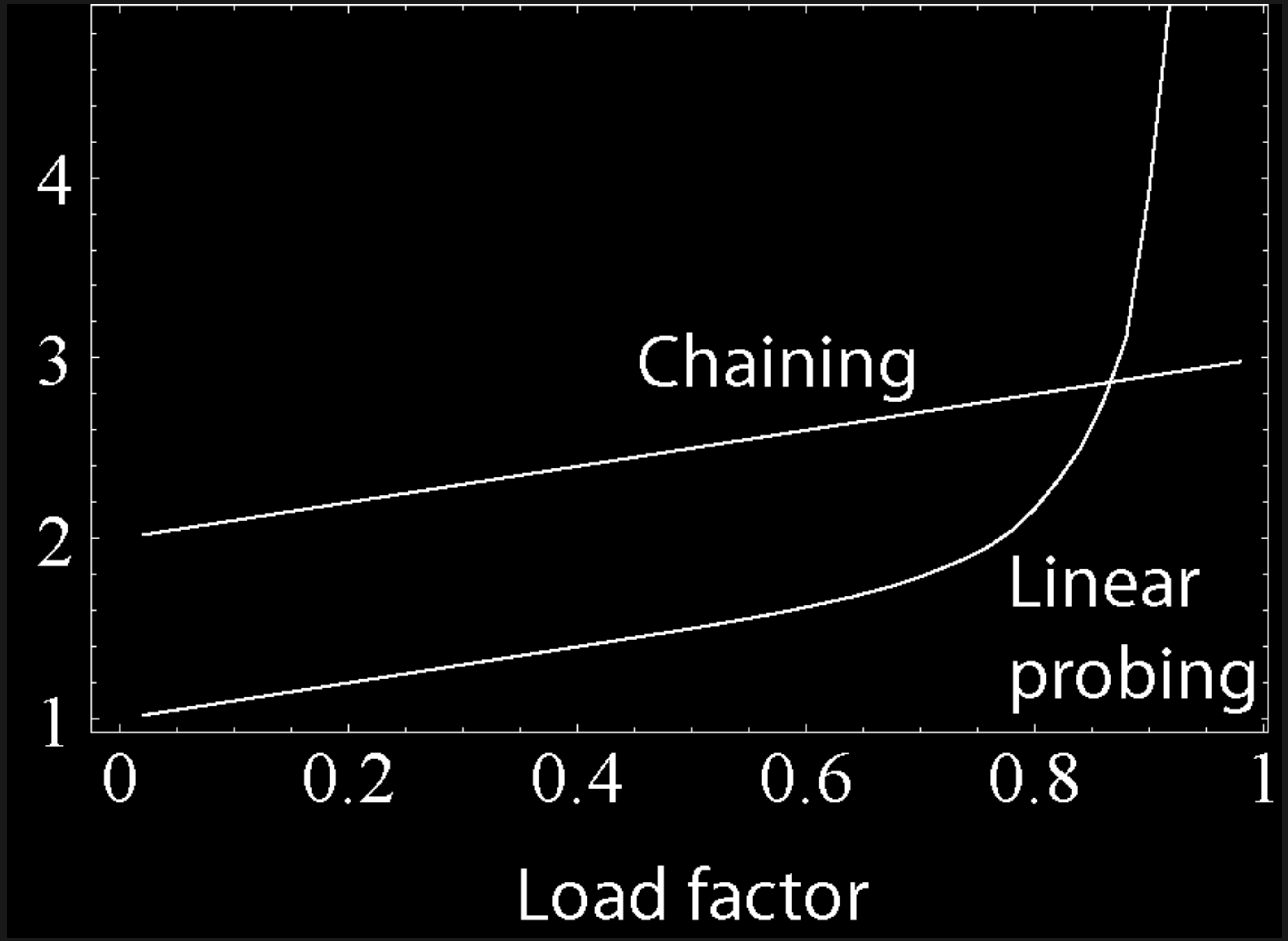
A hash table with 3 key-value entries in 8 buckets has a load factor of 3 / 8 = 0.375

A hash table with 76 key-value entries in 128 buckets has a load factor of 76 / 128 = 0.59375

LOAD FACTOR

Load factor affects performance

Collision resolution affects performance



COMPLEXITY ANALYSIS

	Average Case	Worst Case
Space	O(n)	O(n)
Search	O(1)	O(n)
Insert	O(1)	O(n)
Delete	O(1)	O(n)