Hash Tables

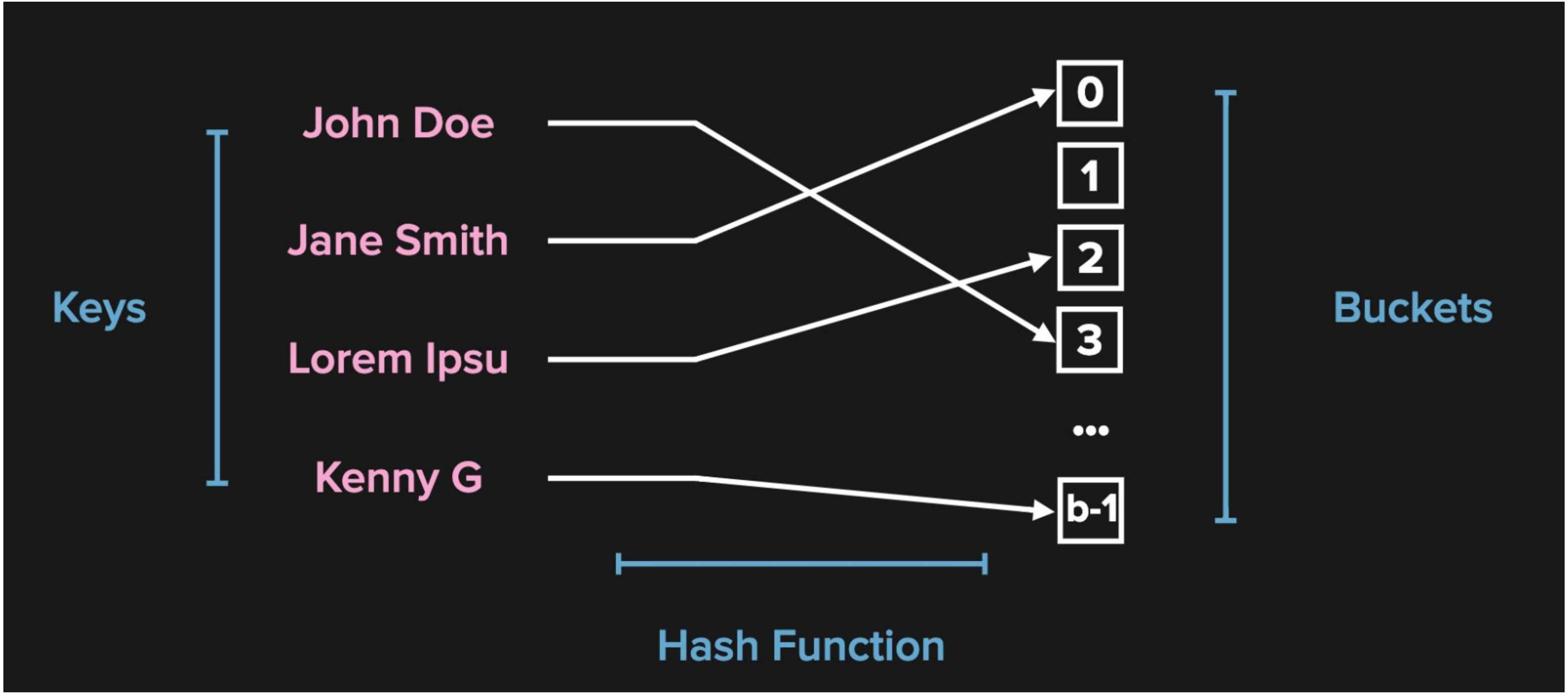
What we’re going to learn

- Understand what a hash table is and applications
- Understand hash table methods
- Understand how we can use the data structures we learned previously to build more complex data structures

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 Table

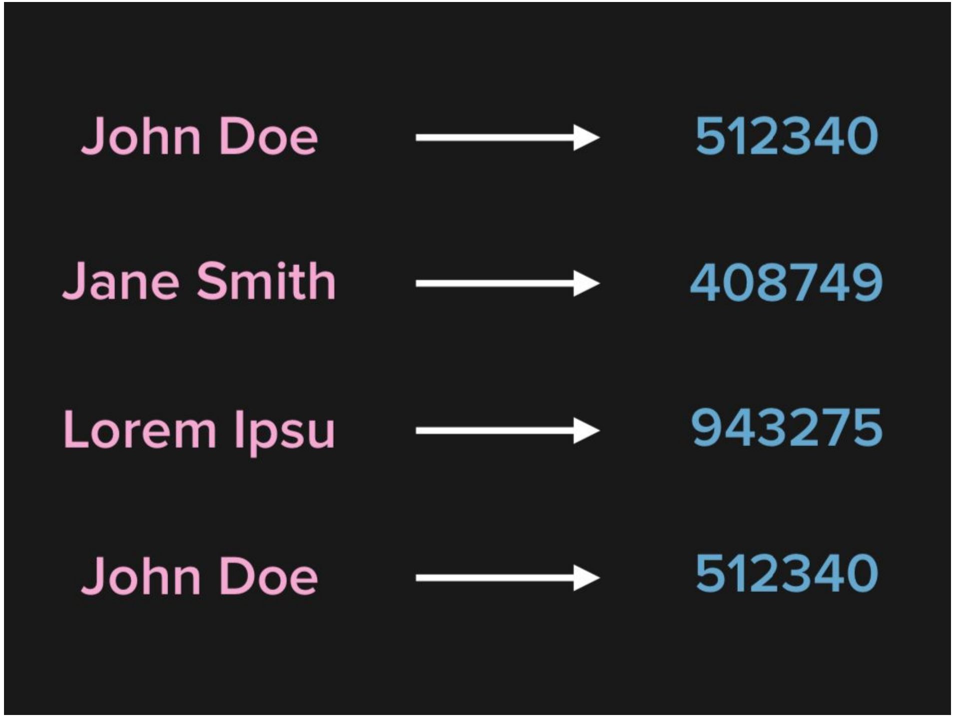

Keys
John Doe
Jane Smith
Lorem Ipsu
Kenny G
0
1
2
3
...
b-1
Buckets
Hash Function

Hash Function

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

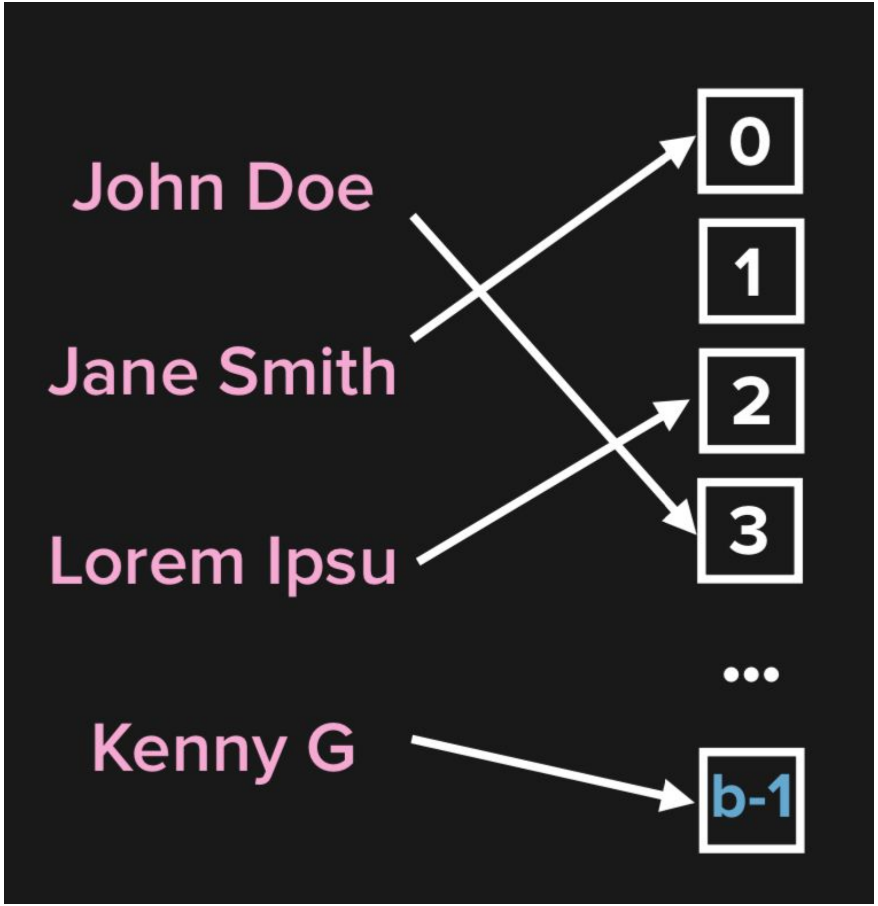
Which Bucket?

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

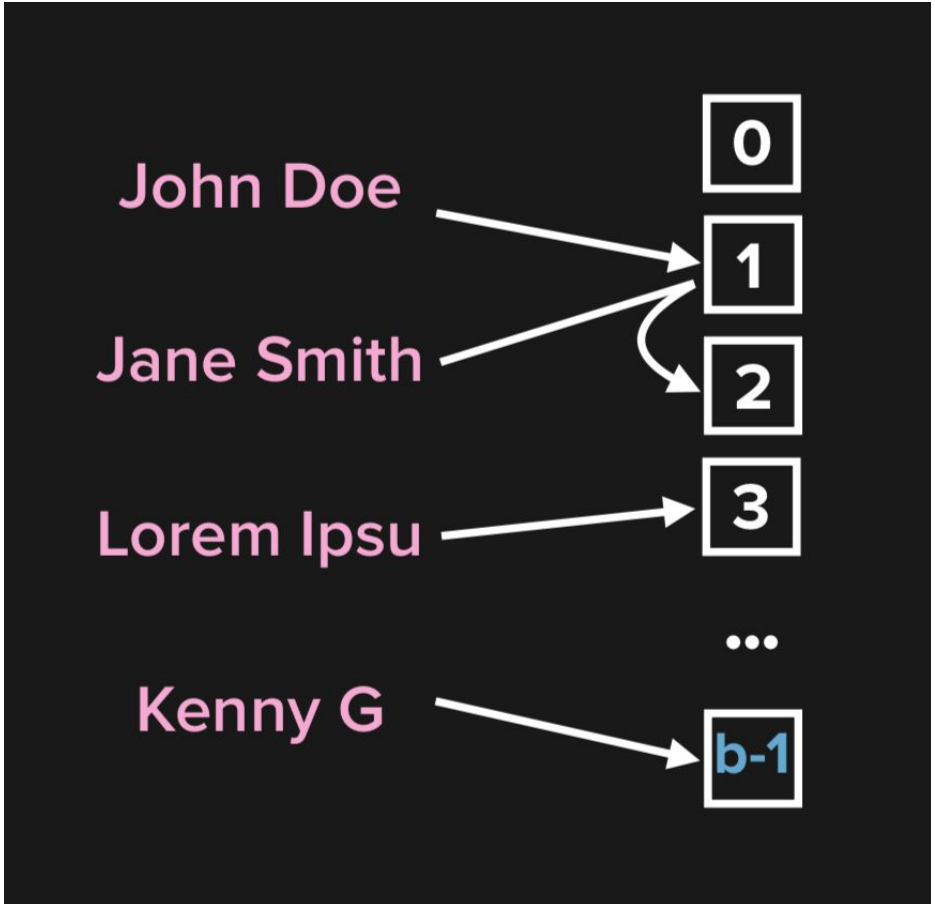
Linear Probing

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



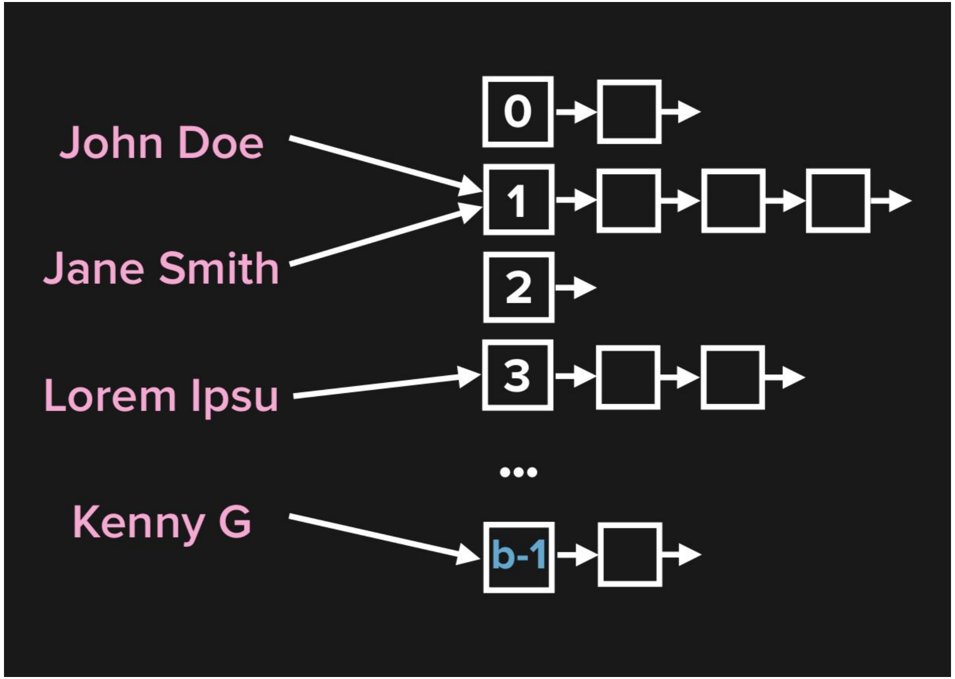
Chaining

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



Let’s Draw a Hash Table

Shout Outs