MAKE
SCHOOL

HASH TABLES

The Ultimate Data Structure

MAKE SCHOOL

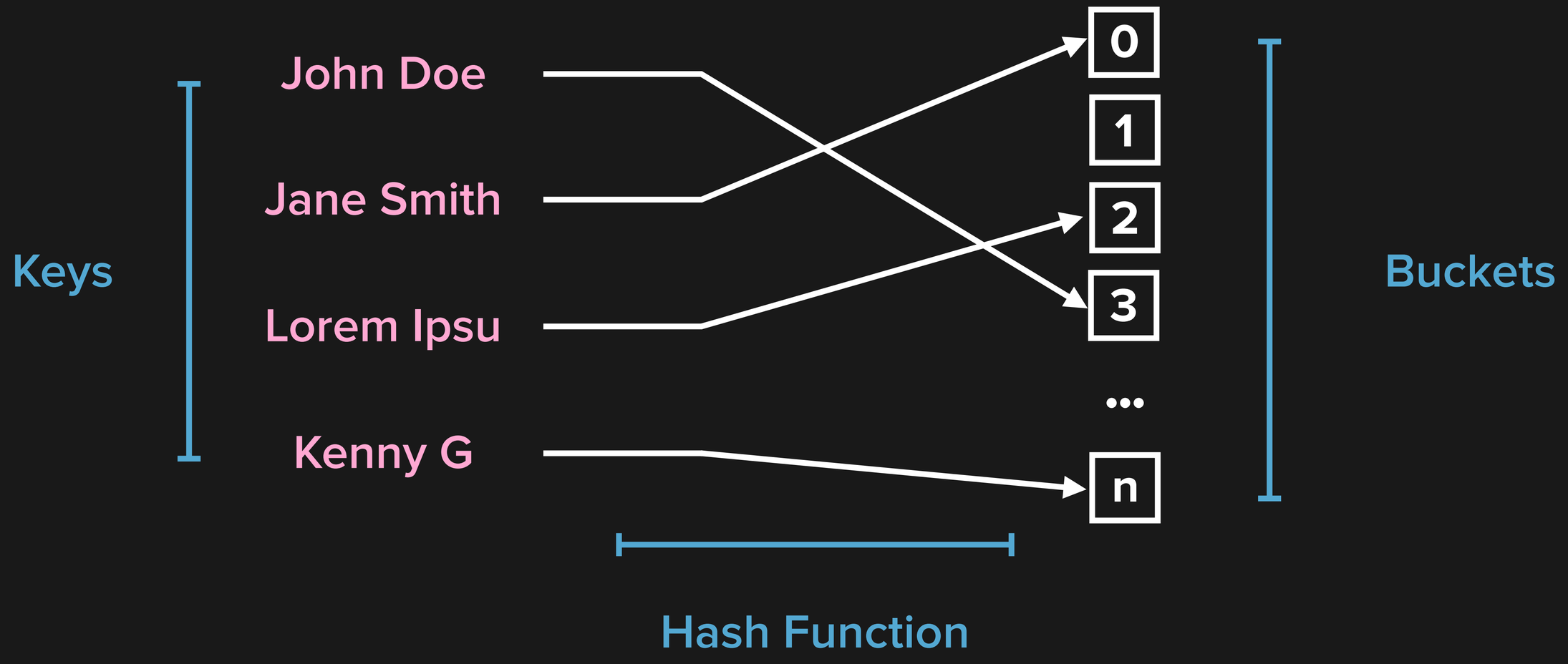
HASH TABLES

Maps keys ➜ objects

`dict()` creates a hash table

Used because of strong average case performance

HASH TABLES

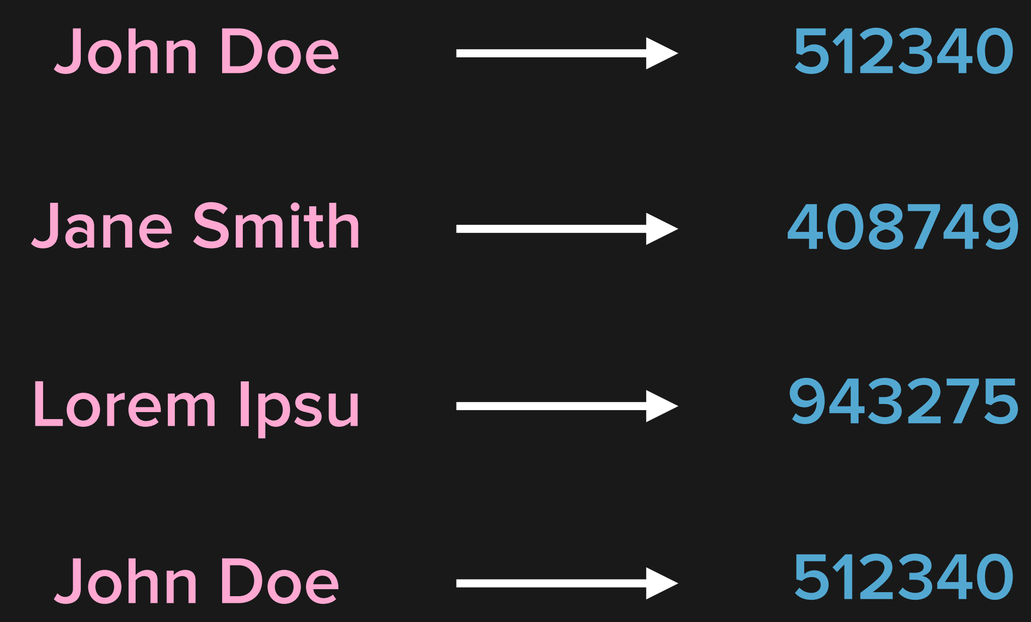


HASH FUNCTIONS

Converts a variable-size input to a fixed-size output

Same input → same output

Input can be anything - string, pointer, custom class



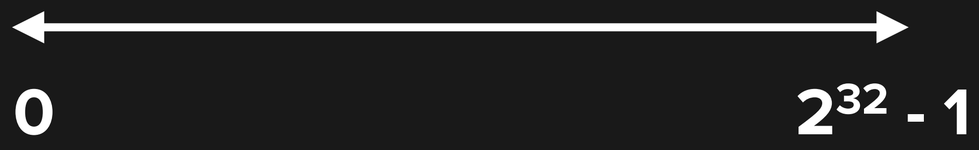
MAKE SCHOOL

IDEAL HASH*

Repeatable

Fast

Output is unsigned integer



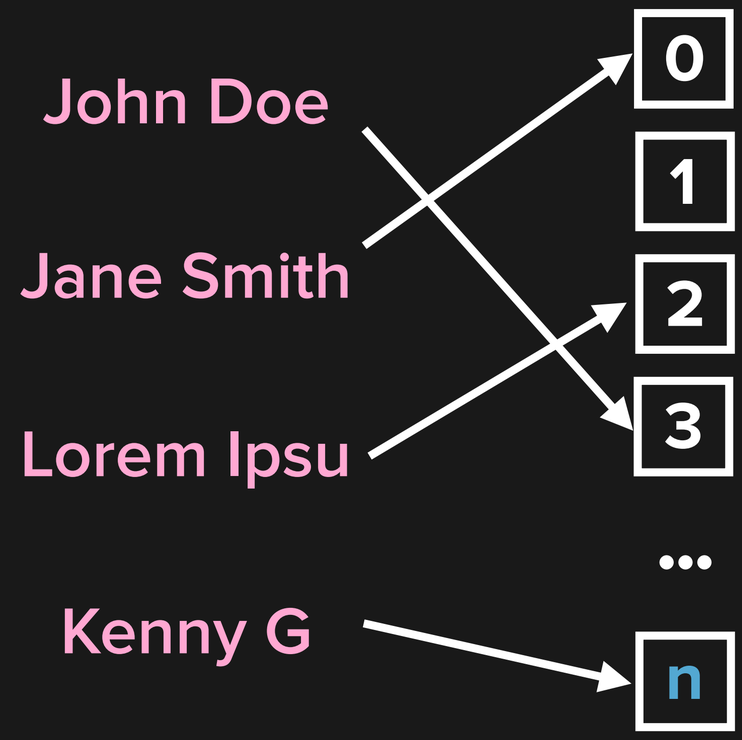
Randomly distributes keys
among output space

Small differences in input result
in large differences in output

**Different for cryptographic hash functions*

WHICH BUCKET?

```
bucket = hash(key) % n
```



COLLISIONS

It is impossible to map all possible input to a fixed output space without some inputs generating the same output

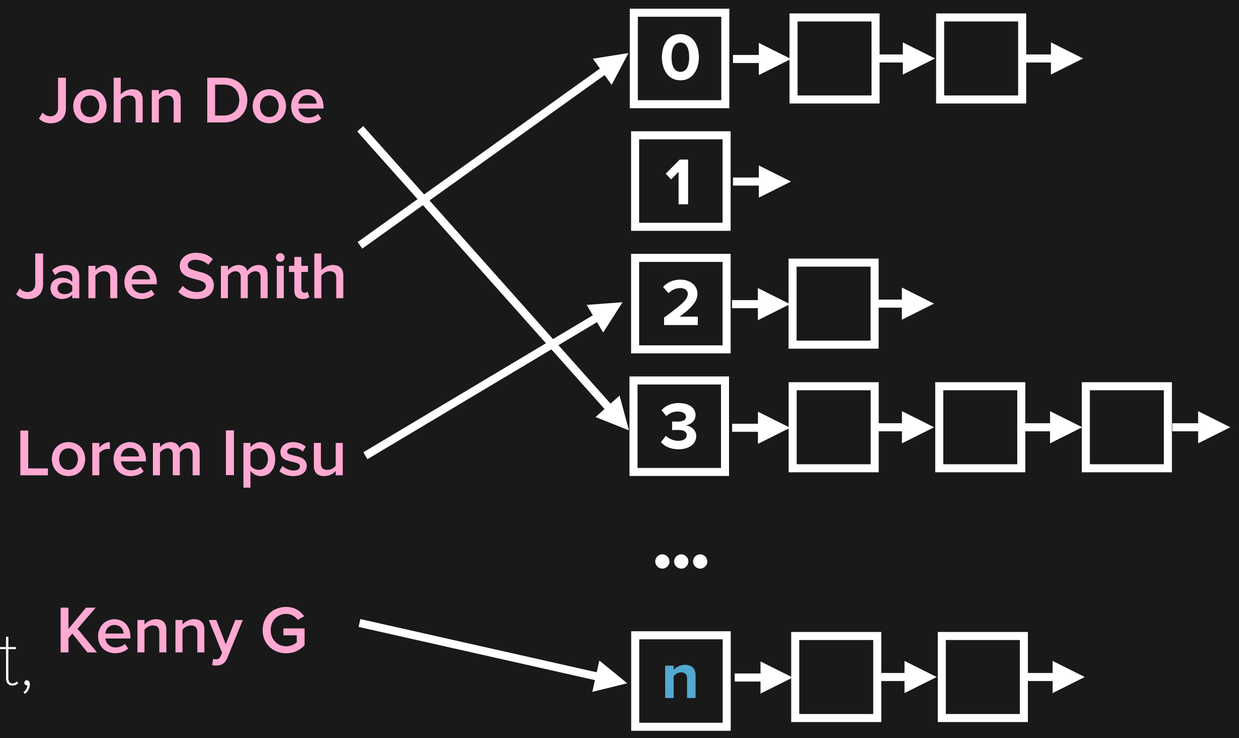
Differing input generating the same output is a collision

CHAINING

Each bucket contains a linked list

On collision - add to end of the linked list

To retrieve - find bucket, find in linked list



MAKE SCHOOL

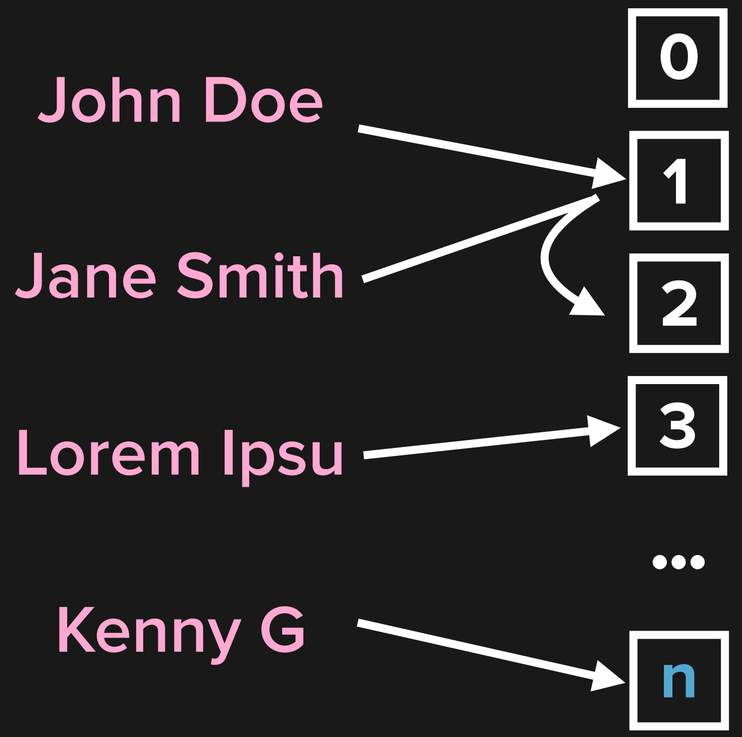
LINEAR PROBING

Each bucket contains one object

On collision - go to next open bucket, add object there

To retrieve - find bucket, if that's not object, iterate buckets until you find it

`dict` does it this way



MAKE SCHOOL

LOAD FACTOR

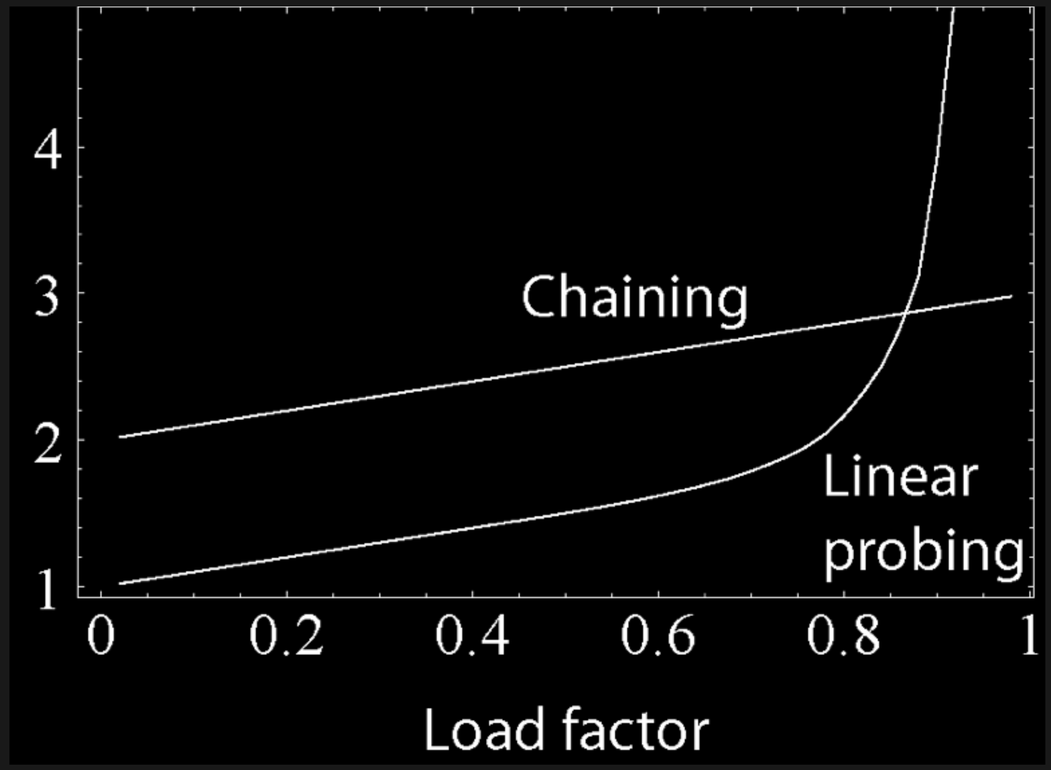
```
Load Factor = entries / buckets
```

For 76 entries in a 128 bucket hash table, that's 76 / 128 = 0.59375

LOAD FACTOR

Load factor affects performance

Collusion 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)

STRING HASHING

Strings are lists of chars

Chars have numerical values

Add up the chars - there's your hash! (Lose Lose algorithm)

But `hash("dog") == hash("god")`