

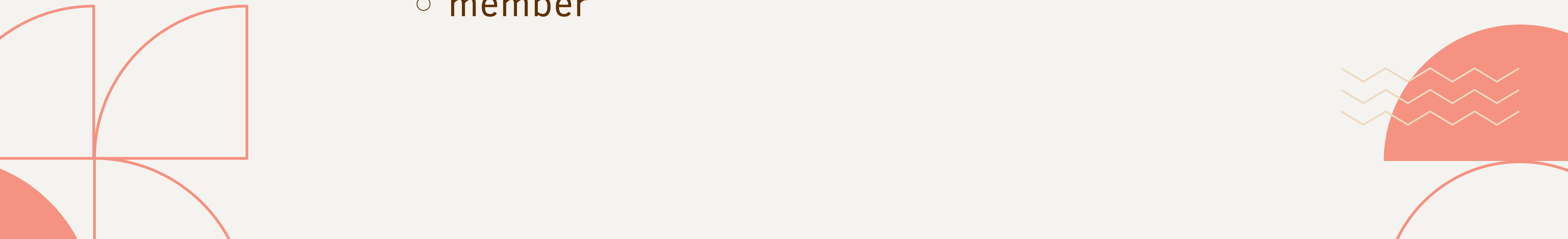

DICTIONARY

By Janie Lane Sabado





DICTIONARY

- fastest data structure
 - a set with operations such as:
 - insert
 - delete
 - member
 - Implementations:
 - Linked List
 - Array
 - Cursor-based
 - Hashing
- 
- 

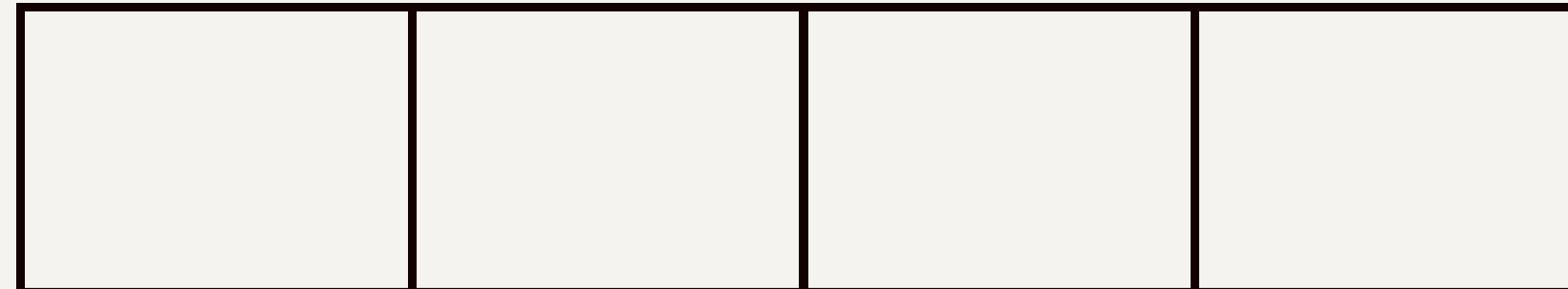
PURPOSE?

```
colors{ "red" , "green" , "blue" }
```

color[0]

color["red"]

color



0

1

2

3

color



red

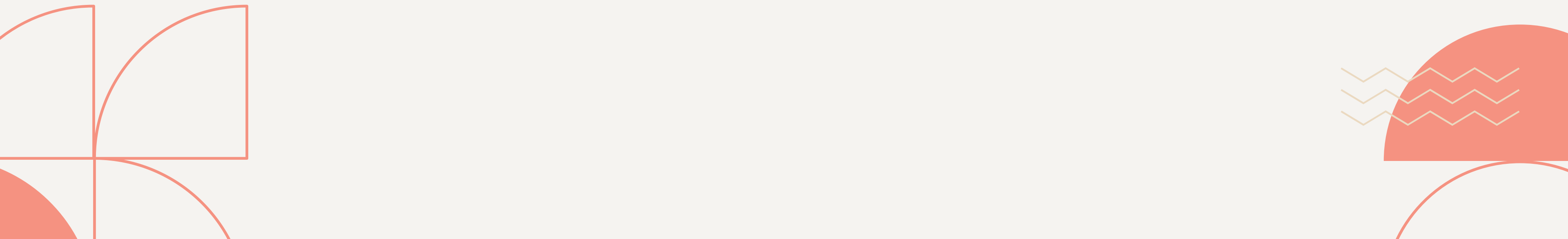
yellow

blue

green





PROBLEM

1. C is a structural language, not an object-oriented one
 2. Index access is only integer values in C
- 



SOL'N: HASHING

- a function that generates a unique key by determining the:
 - Location of the element
 - Starting point in searching for the location of the element
 - generates the index
 - does not know if the index being returned has value
- 
- 

HASH()

color

0	1	2	3

hash (green)

		green	
0	1	2	3

hash (red)

	red	green	
0	1	2	3

HASH() CONTEXT

the context of the hash() is dependent on
the coding context



EXAMPLE. Hash() will return the digit of an
integer in its ones place

```
int Hash(int num) {  
    return num%10;  
}
```





HASH() CONTEXT

Try this out.

- 1.Hash returns a digit in its hundredths place
 - 2.Hash accepts a last name and then it returns 0 if last name starts with A, 1 if B, ... , 25 if Z.
 - 3.Hash accepts an RGB value and returns a digit constrained in a 64 palette size.
- 
- 



TWO TYPES

1. OPEN HASHING (External Hashing)
 - stores the set in potentially unlimited space
 - *Dynamic Array, Linked List*
 2. CLOSED HASHING (Internal Hashing)
 - stores the set in a fixed space
 - *Static Array, Cursor-Based*
- 
- 

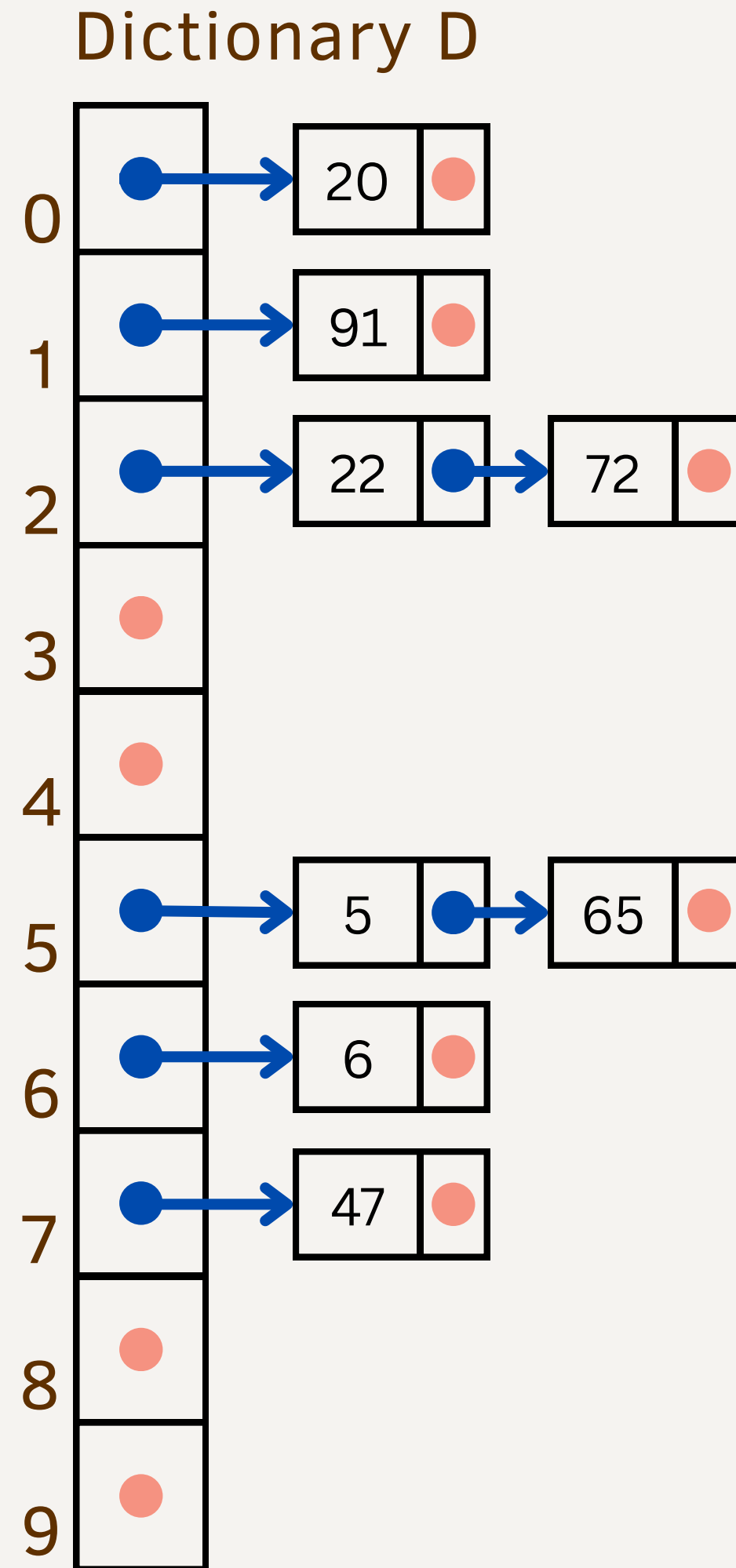


OPEN HASHING

OPEN HASHING

SET A =

{20, 47, 22, 5, 65, 72, 6, 91}



hash() -
generates a value
through % of 10

chaining

- solution to collisions
- uses linked list to chain values to its key
- Array of linked list

A decorative graphic consisting of three horizontal white lines at the top and a 4x4 grid of small white dots at the bottom, all set against a solid black background.





CLOSED HASHING

CLOSED HASHING

SET A =

{20, 47, 22, 5, 65, 72, 6, 91}

Dictionary D

0	20
1	91
2	22
3	
4	
5	5
6	6
7	47
8	
9	

hash() -
generates a value
through % of 10

Synonyms

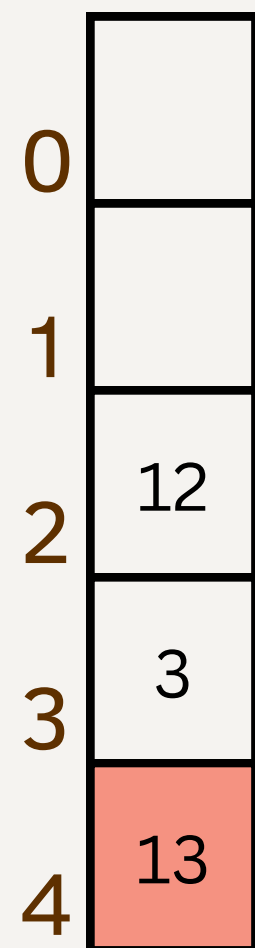
- 2 or more elements
with the same hash
value

Solution?

- Linear Hashing
- Progressive Overflow

LINEAR HASHING

- The element is placed in the next available position if collision occurs
- using concept of circular arrays



```
#define MAX 5;
```

```
hash(x) == (x+1) % MAX;
```

```
hash(3) == 3
```

```
hash(12) == 2
```

```
hash(13) == 3
```

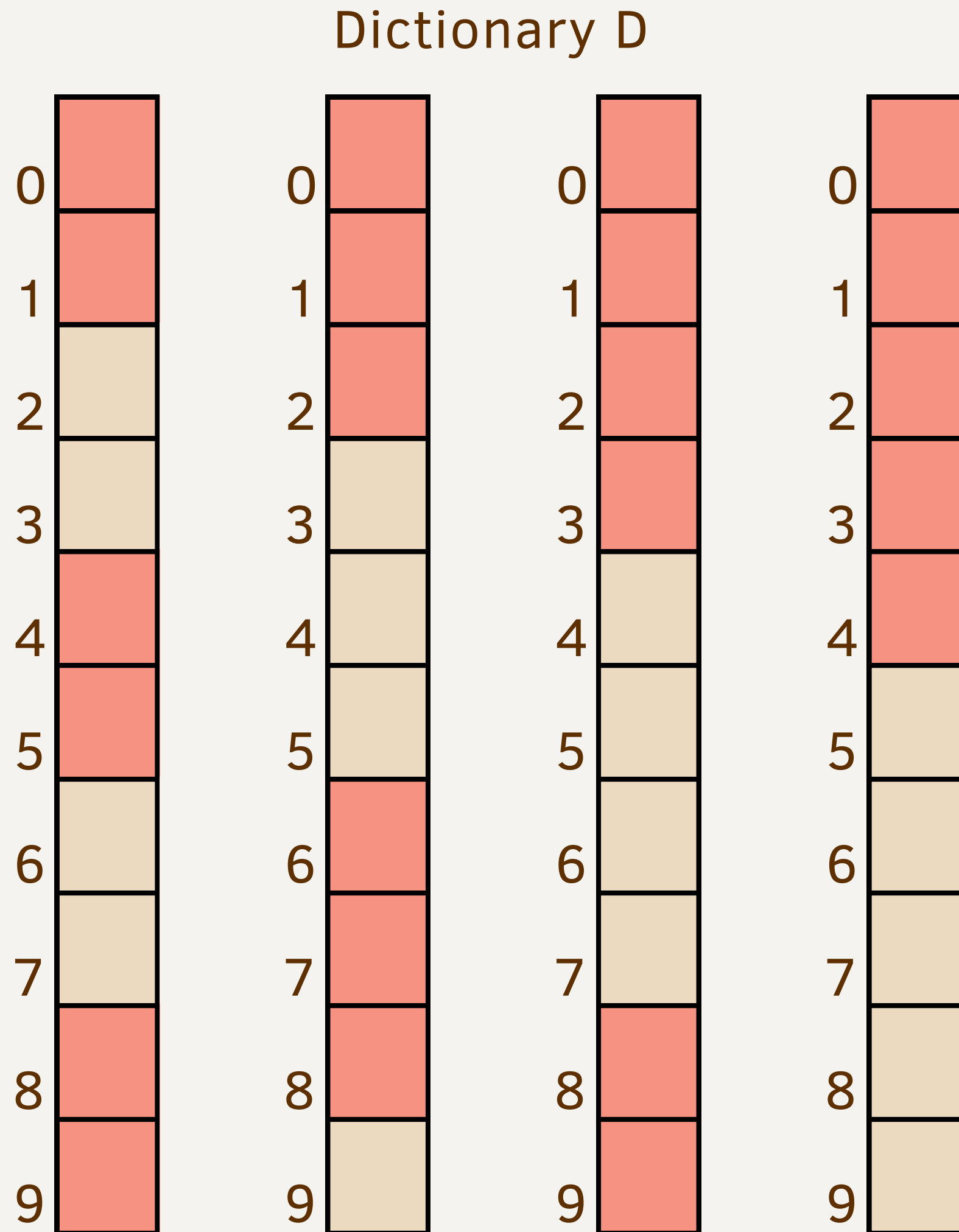
collision

causes **Displacement**

BUCKET

using a range of indexes to
limit space use

hash() -
adjust for the
range of the
bucket



PROGRESSIVE OVERFLOW

- dividing the space used into half, one used as the primary storage and the other as secondary storage

Dictionary D

elem next		elem next		Avail
0	Empty	-1	5	
1	Empty	-1	6	7
2	Empty	-1	7	8
3	Empty	-1	8	9
4	Empty	-1	9	-1
primary		secondary		

```
#define MAX 10
#define Empty " "
#define Delete -1
typedef struct{
    int elem;
    int next;
}node;
typedef struct{
    node table[MAX];
    int Avail;
}Dictionary
```

PROGRESSIVE OVERFLOW

SET A =

{20, 43, 22, 5, 65,
72, 6, 28, 91}

```
#define MAX 5;
```

```
hash(x) == x%MAX;
```

Dictionary D

	elem	next		elem	next	
0	Empty	-1	5		6	Avail <div>5</div>
1	Empty	-1	6		7	
2	Empty	-1	7		8	
3	Empty	-1	8		9	
4	Empty	-1	9		-1	
primary			secondary			

PROGRESSIVE OVERFLOW

SET A =

{20, 43, 22, 5, 65,
72, 6, 28, 91}

```
#define MAX 5;
```

```
hash(x) == x%MAX;
```

Dictionary D

primary		secondary		Avail
elem	next	elem	next	
0	20	5	6	<div>-1</div>
1	6	65	-1	
2	22	72	-1	
3	43	28	-1	
4	Empty	91	-1	

PROGRESSIVE OVERFLOW

Dictionary D

	elem	next		elem	next
0	20	5	5	5	6
1	6	9	6	65	-1
2	22	7	7	72	-1
3	43	8	8	28	-1
4	Empty	-1	9	91	-1
primary			secondary		

Avail

-1

Dictionary D

	elem		elem
0	20	5	5
1	6	6	65
2	22	7	72
3	43	8	28
4	Empty	9	91
primary		secondary	

Last

9

PERFECT HASH?







PERFECT HASH

- when a hash function returns a unique value
- not possible to get a perfect hash

PACKING DENSITY

- ratio of num of elements to be stored to number of available space
 - Perfect Ratio = 70:30 (only for closed hashing)
- 
- 



**THANK
YOU**