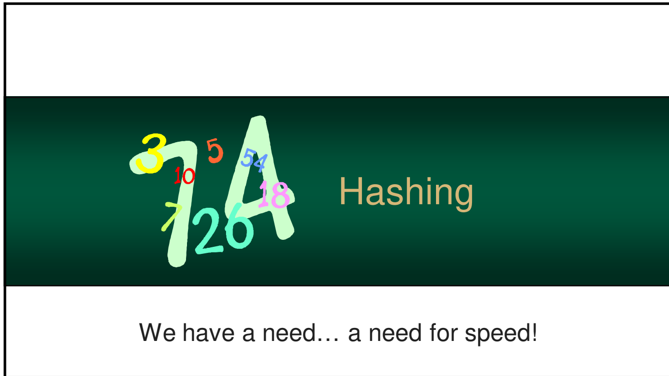
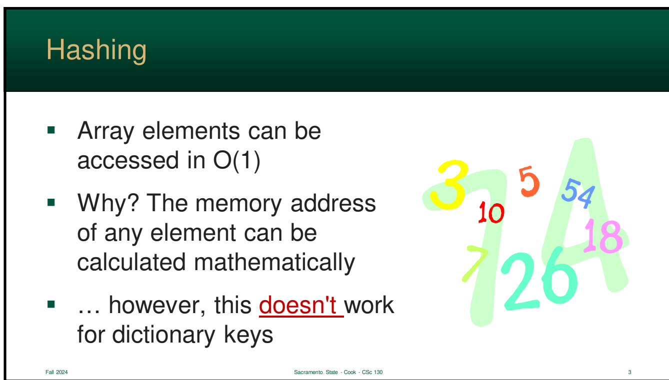
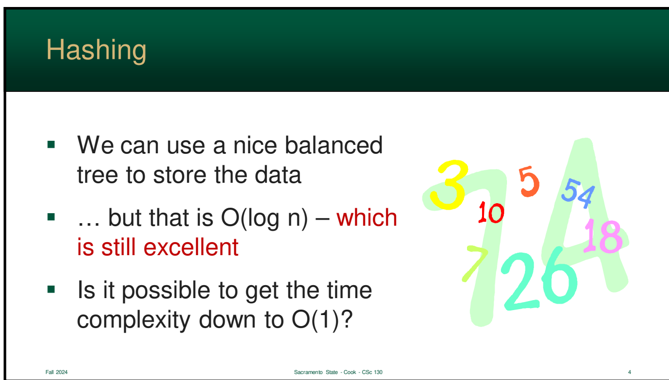
1



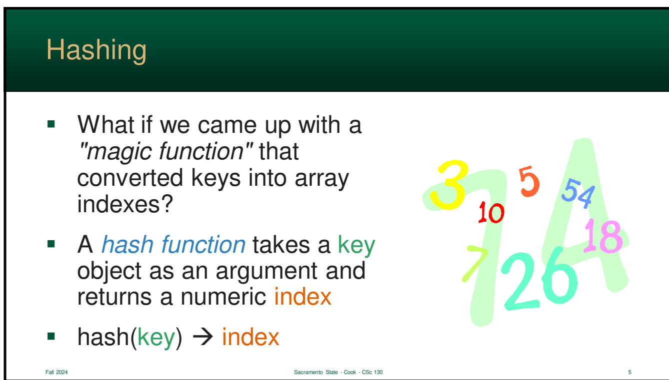
2



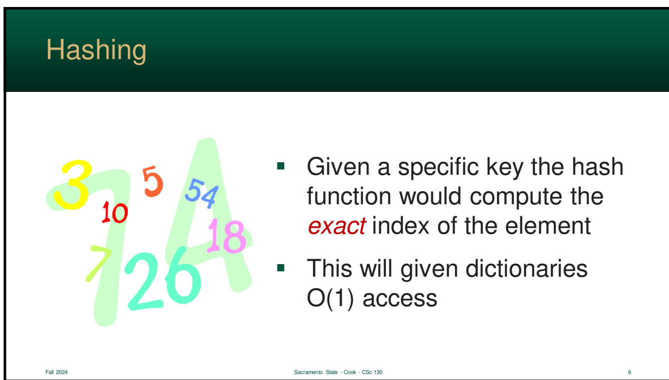
3



4



5



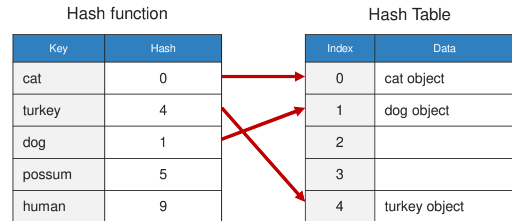
6

Hash Mathematics

- A hash function maps keys into indexes in a *hash table*
- For element e with key k and h is hash function…
  - e is stored in position $h(k)$ in the hash table *(an array)*
  - to find/store for e, compute $h(k)$ to locate position

7

Using a Hash



Hash function

Key	Hash
cat	0
turkey	4
dog	1
possum	5
human	9

Hash Table

Index	Data
0	cat object
1	dog object
2	
3	
4	turkey object

8

Example Hash Function

- `hash("Rick") = 5`
- `hash("Morty") = 1`
- `hash("Jerry") = 0`
- `hash("Beth") = 4`

Array	
0	Jerry
1	Morty
2	
3	
4	Beth
5	Rick

9

But, there are Problems

- Simple hash functions
  - work for implementing dictionaries
  - but most apps have key ranges that are too large for 1-1 mapping between hashes and keys
- Example:
  - key range from 0 to 65,535
  - collection will have no more than 100 items at any given time
  - impractical to use a hash table with 65,536 slots!

10

Finding the Hash Function

- There is **no** magic function
  - **only** in rare cases, with a limited key range, a perfect function can exist
  - however, for real World cases, there is no function possible
- So, we can take a different approach
  - don't use the hash value as a finishing point
  - use it as a location to start looking

11

Collisions

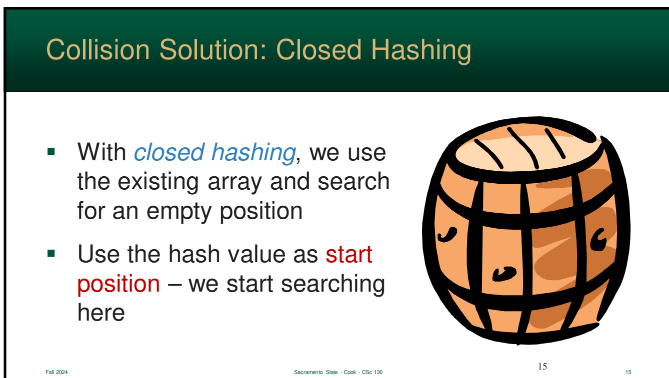
- When two keys hash to the same array location, this is called a *collision*
- What do we do?
  - normally collisions are "first come, first serve"
  - the first key that hashes to the location gets it
  - so, we need to decide what do with the second item that hash to the same location
  - there are two solutions…

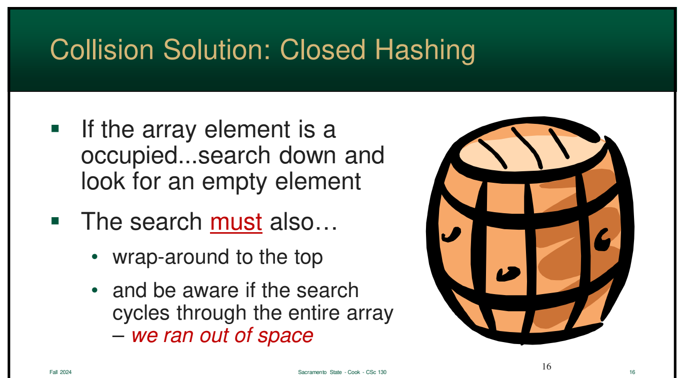
12

Example Hash Function

- `hash("Ash") = 5`
- `hash("Jesse") = 1`
- `hash("Brock") = 0`
- `hash("Misty") = 4`
- `hash("James") = 1`

Array	
0	Brock
1	COLLISION
2	
3	
4	Misty
5	Ash





Closed Hash Example

- Let assume we are adding "Pacman" which has a hash value of **0**
- This collides with "Dig Dug"
- We search down for the next empty element

0	Dig Dug	✖
1	Q-Bert	
2		
3		
4	Fix-It Felix, Jr	
5	Frogger	

Closed Hash Example

- Now, at index **1**, we collide with Q-Bert
- We search down for the next empty element

0	Dig Dug	✖
1	Q-Bert	✖
2		
3		
4	Fix-It Felix, Jr	
5	Frogger	

Closed Hash Example

- Finally, we find an empty array element
- Pacman is stored here
- Note, this is **2** positions off from the original hash

0	Dig Dug	✖
1	Q-Bert	✖
2	**Pacman**	
3		
4	Fix-It Felix, Jr	
5	Frogger	

19

Closed Hashing Clustering

- One problem with the closed hashing is the tendency to form *clusters*
- A cluster is a group of continuous used array elements – with no open slots
- What happens?
  - the bigger a cluster gets, the more likely it is that new keys will hash into it *(and collide)*
  - it then grows larger and larger
  - the hash will eventually degrade to **O(n)**

20

Efficiency of Closed Hashing

- Hash tables are surprisingly efficient
- Although collisions cause searching, tables, items can be found near O(1)
- Even if the table is nearly full (leading to long searches), efficiency is still quite high

21

Closed Hashing Pitfalls

- Closed hashing is not the best solution
- It requires a static array
  - the array cannot be increased at runtime (or the hash fails)
  - hence, the array is finite
- Clustering causes O(n) degradation

22

Closed Hashing Pitfalls

- You cannot delete items
  - it creates empty slots in clusters!
  - this can prevent an item, *added in a cluster*, from being found below the gap
  - there are work-rounds, but it gets **convoluted**

23

Closed Delete Problem

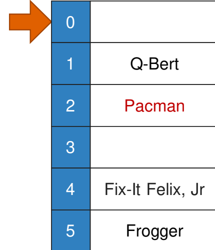
- "Dig Dug" and "Pacman" hash to **0**
- When "Pacman" was added it had to be stored at **2**
- This is a cluster

0	Dig Dug
1	Q-Bert
2	Pacman
3	
4	Fix-It Felix, Jr
5	Frogger

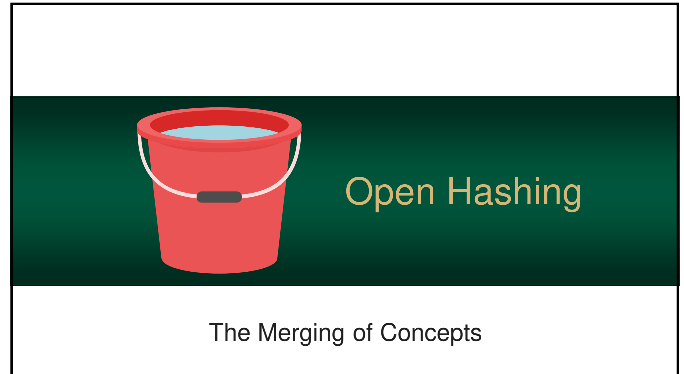
24

Closed Delete Problem

- If "Dig Dug" is deleted, the array element is empty
- If there is a search for "Pacman", it will hash to **0** and it won't be found



0	
1	Q-Bert
2	Pacman
3	
4	Fix-It Felix, Jr
5	Frogger



The Merging of Concepts

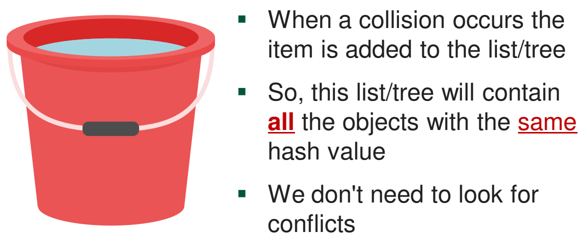
Open Hashing

- With *open hashing*, we don't store individual objects in each array element
- Instead, each array element is a linked list or a tree (preferably balanced)

Open Hashing

- So, our hash table is an array of either linked lists or trees
- This approach is also known as *bucket hashing* since each list/tree acts a "hash bucket"

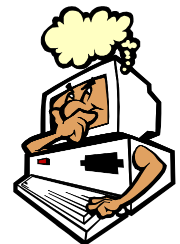
Collision Solution: Open Hashing



- When a collision occurs the item is added to the list/tree
- So, this list/tree will contain **all** the objects with the same hash value
- We don't need to look for conflicts

When an Object is Looked Up…

- Compute the hash value
- And then search the targeted list/tree
- For example, for a balanced tree, searching would be O(1) at best and O(log n) at worst



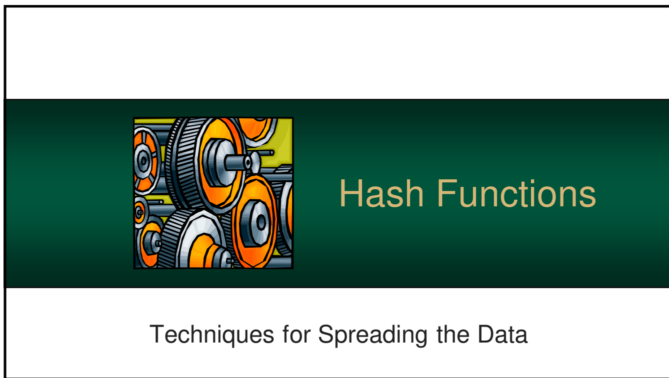
Open Hash Example

- Adding "Voltron" with a hash of 0
- This collides with "Alpha 5"
- Open hashing just adds the item to the linked list

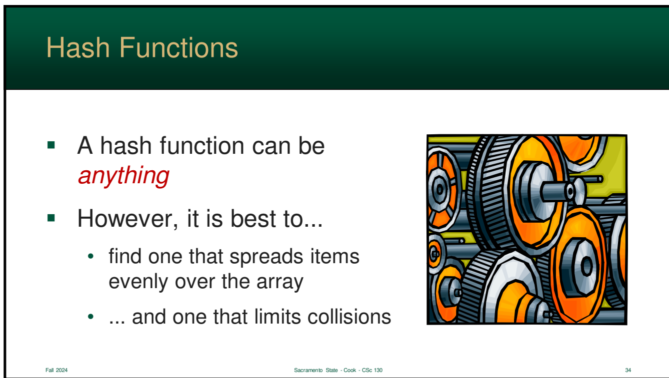
Index	Value	Linked list
0	Alpha 5	→ Voltron
1	HAL	
2		
3		
4	Bender	
5	Twiki	

31

32



33



34

Random Hashing

- Most hashing algorithms use a pseudo-random number generator
- This essentially scatters the items “randomly” throughout the hash table
- ... but there is no real "random" numbers in computers – only chaotic series

35

Popular Algorithm: Modulus

- Uses the formula: $h(k) = k \bmod N$
  - k is a raw key value produced by some internal function
  - we don't care "how" this was produced
  - N is the size of the array
- Selecting N
  - table size N is usually a prime number
  - it prevents patterns – which can cause collisions

36

Popular Algorithm: MAD

- Based on multiply, add, and divide (MAD)
- Uses the formula: $h(k) = (a * k + b) \bmod N$
  - a and b are both constants
  - eliminates patterns provided $a \bmod N \neq 0$
  - this is the same formula used to create (pseudo) random number generators

37

Let's Hash The Students!

Closed hashing

38

Let's Hash The Students!

- Let's add your names to a closed hash table
- First, let's figure out *how* we want to hash your names
- Then, I'll ask for 5 volunteers

39

Section 1: Name Digits mod 6

	Name	Hash
1	Jason	5
2	Alex	4
3	Andrew	0
4	Alexander	3
5	Fred	4

Hash Table	
0	Andrew
1	Fred
2	
3	Alexander
4	Alex
5	Jason

40

Section 1: Birthday Day of the Month mod 6

	Name	Hash
1	Alex	2
2	Jason	31
3	Andrew	21
4	Ritchie	17
5	Bob	13

Hash Table	
0	
1	Jason
2	Alex
3	Andrew
4	Bob
5	Ritchie

41

Section 2: Name Digits mod 6

	Name	Hash
1	Zyale	5
2	Ryan	4
3	Iftekhar	2
4	Sydney	0
5	Margarette	4

Hash Table	
0	Sydney
1	Margarette
2	Iftekhar
3	
4	Ryan
5	Zyale

42

Section 2: Birthday Day of the Month mod 6

	Name	Hash
1	Ryan	3
2	Zyale	1
3	Eric	1
4	Rachel	1
5	Pavan	3

Hash Table	
0	
1	Zyale
2	Eric
3	Ryan
4	Rachel
5	Pavan

43